

このページは機械翻訳したものです。

MySQL 8.0 リファレンスマニュアル

MySQL NDB Cluster 8.0 を含む

概要

これは MySQL リファレンスマニュアルです。MySQL 8.0 から 8.0.25、および NDB のバージョン 8.0 から 8.0.25-ndb-8.0.25 に基づく NDB Cluster リリースについてそれぞれ説明します。まだリリースされていない MySQL バージョンの機能のドキュメントが含まれている場合があります。リリースされたバージョンの詳細は、「[MySQL 8.0 リリースノート](#)」を参照してください。

MySQL 8.0 の機能. このマニュアルでは、MySQL 8.0 のエディションによっては含まれていない機能について説明します。このような機能は、ご自身にライセンス付与されている MySQL 8.0 のエディションに含まれていない場合があります。MySQL 8.0 の使用しているエディションに含まれる機能に関する質問がある場合は、MySQL 8.0 ライセンス契約をご覧くださいか、Oracle セールス担当者に連絡してください。

各リリースの変更内容の詳細は、[MySQL 8.0 リリースノート](#)を参照してください。

ライセンス情報を含む法的情報については、[序文と法的通知](#)を参照してください。

MySQL の使用方法のヘルプは、「[MySQL フォーラム](#)」を参照してください。この「[MySQL フォーラム](#)」では、他の MySQL ユーザーとの問題について説明できます。

ドキュメント生成日: 2022-06-14 (revision: 111)

目次

序文と法的通知	xxv
1 一般情報	1
1.1 このマニュアルについて	2
1.2 MySQL データベース管理システムの概要	4
1.2.1 MySQL とは	4
1.2.2 MySQL の主な機能	5
1.2.3 MySQL の歴史	8
1.3 MySQL 8.0 の新機能	8
1.4 MySQL 8.0 で追加、非推奨または削除されたサーバーおよびステータスの変数とオプション	44
1.5 MySQL の情報源	61
1.6 質問またはバグをレポートする方法	62
1.7 MySQL の標準への準拠	66
1.7.1 標準 SQL に対する MySQL 拡張機能	67
1.7.2 MySQL と標準 SQL との違い	69
1.7.3 MySQL における制約の処理	71
1.8 クレジット	73
1.8.1 MySQL への貢献者	73
1.8.2 ドキュメント作成者および翻訳者	77
1.8.3 MySQL をサポートするパッケージ	78
1.8.4 MySQL の作成に使用されたツール	79
1.8.5 MySQL のサポーター	79
2 MySQL のインストールとアップグレード	81
2.1 一般的なインストールガイド	83
2.1.1 サポートされるプラットフォーム	83
2.1.2 インストールする MySQL のバージョンと配布の選択	84
2.1.3 MySQL の取得方法	85
2.1.4 MD5 チェックサムまたは GnuPG を用いたパッケージの完全性の確認	85
2.1.5 インストールのレイアウト	97
2.1.6 コンパイラ固有のビルドの特徴	97
2.2 一般的なバイナリを使用した MySQL の Unix/Linux へのインストール	98
2.3 Microsoft Windows に MySQL をインストールする	100
2.3.1 Microsoft Windows 上での MySQL のインストールレイアウト	103
2.3.2 インストールパッケージの選択	103
2.3.3 MySQL Installer for Windows	105
2.3.4 <code>noinstall</code> ZIP アーカイブを使用した Microsoft Windows への MySQL のインストール	127
2.3.5 Microsoft Windows MySQL Server インストールのトラブルシューティング	134
2.3.6 Windows でのインストール後の手順	136
2.3.7 Windows プラットフォームの制限事項	137
2.4 macOS への MySQL のインストール	139
2.4.1 macOS への MySQL のインストールに関する一般的なノート	139
2.4.2 ネイティブパッケージを使用した macOS への MySQL のインストール	140
2.4.3 MySQL 起動デーモンのインストールおよび使用	145
2.4.4 MySQL Preference Pane のインストールと使用	148
2.5 Linux に MySQL をインストールする	152
2.5.1 MySQL Yum リポジトリを使用して MySQL を Linux にインストールする	153
2.5.2 MySQL APT リポジトリを使用して MySQL を Linux にインストールする	157
2.5.3 MySQL SLES リポジトリを使用して MySQL を Linux にインストールする	157
2.5.4 Oracle の RPM パッケージを使用した Linux への MySQL のインストール	157
2.5.5 オラクルからの Debian パッケージを使用して MySQL を Linux にインストールする	161
2.5.6 Docker での Linux への MySQL のデプロイ	163
2.5.7 ネイティブソフトウェアリポジトリから MySQL を Linux にインストールする	173
2.5.8 Juju を使用した Linux への MySQL のインストール	175
2.5.9 systemd を使用した MySQL Server の管理	175
2.6 Unbreakable Linux Network (ULN) を使用した MySQL のインストール	180
2.7 Solaris への MySQL のインストール	180
2.7.1 Solaris PKG を使用して Solaris に MySQL をインストールする	181
2.8 FreeBSD に MySQL をインストールする	182

2.9 ソースから MySQL をインストールする	183
2.9.1 ソースのインストール方法	183
2.9.2 ソースインストールの前提条件	184
2.9.3 ソースインストールの MySQL のレイアウト	185
2.9.4 標準ソース配布を使用して MySQL をインストールする	185
2.9.5 開発ソースツリーを使用して MySQL をインストールする	189
2.9.6 SSL ライブラリサポートの構成	191
2.9.7 MySQL ソース構成オプション	191
2.9.8 MySQL のコンパイルに関する問題	218
2.9.9 MySQL の構成とサードパーティーツール	219
2.9.10 MySQL Doxygen ドキュメントコンテンツの生成	220
2.10 インストール後のセットアップとテスト	220
2.10.1 データディレクトリの初期化	221
2.10.2 サーバーの起動	226
2.10.3 サーバーのテスト	228
2.10.4 初期 MySQL アカウントの保護	230
2.10.5 MySQL を自動的に起動および停止する	232
2.11 MySQL のアップグレード	233
2.11.1 始める前に	233
2.11.2 アップグレードパス	234
2.11.3 MySQL のアップグレードプロセスの内容	234
2.11.4 MySQL 8.0 での変更	238
2.11.5 アップグレード用のインストールの準備	248
2.11.6 Unix/Linux での MySQL バイナリまたはパッケージベースのインストールのアップグレード	251
2.11.7 MySQL Yum リポジトリを使用する MySQL のアップグレード	255
2.11.8 MySQL APT リポジトリを使用する MySQL のアップグレード	257
2.11.9 MySQL SLES リポジトリを含む MySQL のアップグレード	257
2.11.10 Windows 上の MySQL をアップグレードする	257
2.11.11 MySQL の Docker インストールのアップグレード	259
2.11.12 アップグレードのトラブルシューティング	259
2.11.13 テーブルまたはインデックスの再作成または修復	259
2.11.14 MySQL データベースのほかのマシンへのコピー	261
2.12 MySQL のダウングレード	262
2.13 Perl のインストールに関する注釈	262
2.13.1 Unix に Perl をインストールする	262
2.13.2 Windows に ActiveState Perl をインストールする	263
2.13.3 Perl DBI/DBD インタフェース使用の問題	263
3 チュートリアル	265
3.1 サーバーへの接続とサーバーからの切断	265
3.2 クエリーの入力	266
3.3 データベースの作成と使用	269
3.3.1 データベースの作成と選択	270
3.3.2 テーブルの作成	270
3.3.3 テーブルへのデータのロード	272
3.3.4 テーブルからの情報の取り出し	273
3.4 データベースとテーブルに関する情報の取得	285
3.5 バッチモードでの MySQL の使用	286
3.6 一般的なクエリーの例	287
3.6.1 カラムの最大値	287
3.6.2 特定のカラムの最大値が格納されている行	288
3.6.3 グループごとのカラムの最大値	288
3.6.4 特定のカラムのグループごとの最大値が格納されている行	288
3.6.5 ユーザー定義の変数の使用	289
3.6.6 外部キーの使用	290
3.6.7 2 つのキーを使用した検索	291
3.6.8 日ごとの訪問数の計算	291
3.6.9 AUTO_INCREMENT の使用	292
3.7 Apache での MySQL の使用	294
4 MySQL プログラム	295
4.1 MySQL プログラムの概要	296

4.2 MySQL プログラムの使用	299
4.2.1 MySQL プログラムの起動	299
4.2.2 プログラムオプションの指定	300
4.2.3 サーバーに接続するためのコマンドオプション	312
4.2.4 コマンドオプションを使用した MySQL Server への接続	319
4.2.5 URI 類似文字列またはキーと値のペアを使用したサーバーへの接続	322
4.2.6 DNS SRV レコードを使用したサーバーへの接続	327
4.2.7 接続トランスポートプロトコル	328
4.2.8 接続圧縮制御	330
4.2.9 環境変数の設定	333
4.3 サーバーおよびサーバーの起動プログラム	334
4.3.1 mysqld — MySQL サーバー	334
4.3.2 mysqld_safe — MySQL サーバー起動スクリプト	334
4.3.3 mysql.server — MySQL サーバー起動スクリプト	340
4.3.4 mysqld_multi — 複数の MySQL サーバーの管理	342
4.4 インストール関連プログラム	346
4.4.1 comp_err — MySQL エラーメッセージファイルのコンパイル	346
4.4.2 mysql_secure_installation — MySQL インストールのセキュリティー改善	347
4.4.3 mysql_ssl_rsa_setup — SSL/RSA ファイルの作成	350
4.4.4 mysql_tzinfo_to_sql — タイムゾーンテーブルのロード	353
4.4.5 mysql_upgrade — MySQL テーブルのチェックとアップグレード	353
4.5 クライアントプログラム	362
4.5.1 mysql — MySQL コマンドラインクライアント	362
4.5.2 mysqladmin — A MySQL Server 管理プログラム	393
4.5.3 mysqlcheck — テーブル保守プログラム	404
4.5.4 mysqldump — データベースバックアッププログラム	414
4.5.5 mysqlimport — データインポートプログラム	439
4.5.6 mysqlpump — データベースバックアッププログラム	447
4.5.7 mysqlshow — データベース、テーブル、およびカラム情報の表示	465
4.5.8 mysqlslap — ロードエミュレーションクライアント	472
4.6 管理およびユーティリティプログラム	483
4.6.1 ibd2sdi — InnoDB テーブルスペース SDI 抽出ユーティリティ	483
4.6.2 innochecksum — オフライン InnoDB ファイルチェックサムユーティリティ	486
4.6.3 myisam_ftdump — 全文インデックス情報の表示	491
4.6.4 myisamchk — MyISAM テーブルメンテナンスユーティリティ	492
4.6.5 myisamlog — MyISAM ログファイルの内容の表示	508
4.6.6 myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成	509
4.6.7 mysql_config_editor — MySQL 構成ユーティリティ	514
4.6.8 mysqlbinlog — バイナリログファイルを処理するためのユーティリティ	520
4.6.9 mysqldumpslow — スロークエリーログファイルの要約	544
4.7 プログラム開発ユーティリティ	546
4.7.1 mysql_config — クライアントのコンパイル用オプションの表示	546
4.7.2 my_print_defaults — オプションファイルからのオプションの表示	547
4.8 その他のプログラム	548
4.8.1 lz4_decompress — mysqlpump LZ4-Compressed 出力の解凍	548
4.8.2 perror — MySQL エラーメッセージ情報の表示	549
4.8.3 zlib_decompress — mysqlpump ZLIB 圧縮出力の解凍	549
4.9 環境変数	550
4.10 MySQL での Unix シグナル処理	552
5 MySQL サーバーの管理	555
5.1 MySQL Server	556
5.1.1 サーバーの構成	556
5.1.2 サーバー構成のデフォルト値	557
5.1.3 サーバー構成の検証	558
5.1.4 サーバーオプション、システム変数およびステータス変数リファレンス	559
5.1.5 サーバースystem変数リファレンス	604
5.1.6 サーバースtatus変数リファレンス	628
5.1.7 サーバークマンドオプション	644
5.1.8 サーバースystem変数	669
5.1.9 システム変数の使用	806

5.1.10	サーバーステータス変数	834
5.1.11	サーバー SQL モード	854
5.1.12	接続管理	864
5.1.13	IPv6 サポート	871
5.1.14	ネットワーク名前空間のサポート	875
5.1.15	MySQL Server でのタイムゾーンのサポート	880
5.1.16	リソースグループ	885
5.1.17	サーバー側ヘルプのサポート	889
5.1.18	クライアントセッション状態の変更のサーバートラッキング	890
5.1.19	サーバーの停止プロセス	892
5.2	MySQL データディレクトリ	894
5.3	mysql システムスキーマ	895
5.4	MySQL Server ログ	900
5.4.1	一般クエリーログおよびスロークエリーログの出力先の選択	900
5.4.2	エラーログ	903
5.4.3	一般クエリーログ	921
5.4.4	バイナリログ	922
5.4.5	スロークエリーログ	937
5.4.6	サーバーログの保守	940
5.5	MySQL のコンポーネント	942
5.5.1	コンポーネントのインストールおよびアンインストール	942
5.5.2	コンポーネント情報の取得	943
5.5.3	エラーログコンポーネント	943
5.5.4	クエリー属性コンポーネント	945
5.6	MySQL Server プラグイン	945
5.6.1	プラグインのインストールおよびアンインストール	946
5.6.2	サーバープラグイン情報の取得	950
5.6.3	MySQL Enterprise Thread Pool	951
5.6.4	リライタクエリーリライトプラグイン	958
5.6.5	ddl_rewriter プラグイン	966
5.6.6	バージョントークン	968
5.6.7	クローンプラグイン	978
5.6.8	MySQL プラグインサービス	1000
5.7	MySQL Server のユーザー定義関数	1007
5.7.1	ユーザー定義関数のインストールおよびアンインストール	1008
5.7.2	ユーザー定義関数情報の取得	1009
5.8	1 つのマシン上での複数の MySQL インスタンスの実行	1009
5.8.1	複数のデータディレクトリのセットアップ	1011
5.8.2	Windows 上での複数の MySQL インスタンスの実行	1012
5.8.3	Unix 上での複数の MySQL インスタンスの実行	1014
5.8.4	複数サーバー環境でのクライアントプログラムの使用	1015
5.9	MySQL のデバッグ	1016
5.9.1	MySQL サーバーのデバッグ	1016
5.9.2	MySQL クライアントのデバッグ	1022
5.9.3	LOCK_ORDER ツール	1022
5.9.4	DEBUG パッケージ	1027
6	セキュリティ	1031
6.1	一般的なセキュリティの問題	1032
6.1.1	セキュリティガイドライン	1032
6.1.2	パスワードをセキュアな状態にする	1034
6.1.3	攻撃者に対する MySQL のセキュアな状態の維持	1036
6.1.4	セキュリティ関連の mysqld オプションおよび変数	1038
6.1.5	MySQL を通常ユーザーとして実行する方法	1038
6.1.6	LOAD DATA LOCAL のセキュリティ上の考慮事項	1039
6.1.7	クライアントプログラミングのセキュリティガイドライン	1042
6.2	アクセス制御とアカウント管理	1043
6.2.1	アカウントのユーザー名とパスワード	1044
6.2.2	MySQL で提供される権限	1045
6.2.3	付与テーブル	1062
6.2.4	アカウント名の指定	1070

6.2.5	ロール名の指定	1072
6.2.6	アクセス制御、ステージ 1: 接続の検証	1073
6.2.7	アクセス制御、ステージ 2: リクエストの確認	1076
6.2.8	アカウントの追加、権限の割当ておよびアカウントの削除	1077
6.2.9	予約済アカウント	1081
6.2.10	ロールの使用	1081
6.2.11	アカウントカテゴリ	1087
6.2.12	部分取消しを使用した権限の制限	1091
6.2.13	権限変更が有効化される時期	1096
6.2.14	アカウントパスワードの割り当て	1097
6.2.15	パスワード管理	1098
6.2.16	期限切れパスワードのサーバー処理	1108
6.2.17	プラグブル認証	1110
6.2.18	プロキシユーザー	1115
6.2.19	アカウントロック	1122
6.2.20	アカウントリソース制限の設定	1122
6.2.21	MySQL への接続の問題のトラブルシューティング	1124
6.2.22	SQL ベースのアカウントアクティビティ監査	1128
6.3	暗号化された接続の使用	1129
6.3.1	暗号化接続を使用するための MySQL の構成	1130
6.3.2	暗号化された接続 TLS プロトコルおよび暗号	1136
6.3.3	SSL および RSA 証明書とキーの作成	1142
6.3.4	SSH を使用した Windows から MySQL へのリモート接続	1150
6.4	セキュリティコンポーネントおよびプラグイン	1151
6.4.1	認証プラグイン	1151
6.4.2	Connection-Control プラグイン	1215
6.4.3	パスワード検証コンポーネント	1221
6.4.4	MySQL キーリング	1231
6.4.5	MySQL Enterprise Audit	1279
6.4.6	監査メッセージコンポーネント	1346
6.4.7	MySQL Enterprise Firewall	1348
6.5	MySQL Enterprise Data Masking and De-Identification	1371
6.5.1	MySQL Enterprise Data Masking and De-Identification 要素	1373
6.5.2	MySQL Enterprise Data Masking and De-Identification のインストールまたはアンインストール	1373
6.5.3	MySQL Enterprise Data Masking and De-Identification の使用	1374
6.5.4	MySQL Enterprise Data Masking and De-Identification ユーザー定義関数リファレンス	1379
6.5.5	MySQL Enterprise Data Masking and De-Identification ユーザー定義関数の説明	1380
6.6	MySQL Enterprise Encryption	1388
6.6.1	MySQL Enterprise Encryption のインストール	1389
6.6.2	MySQL Enterprise Encryption の使用方法と例	1389
6.6.3	MySQL Enterprise Encryption ユーザー定義関数リファレンス	1391
6.6.4	MySQL Enterprise Encryption ユーザー定義関数の説明	1392
6.7	SELinux	1395
6.7.1	SELinux が有効かどうかの確認	1396
6.7.2	SELinux モードの変更	1396
6.7.3	MySQL Server SELinux ポリシー	1397
6.7.4	SELinux ファイルコンテキスト	1397
6.7.5	SELinux TCP ポートコンテキスト	1398
6.7.6	SELinux のトラブルシューティング	1400
6.8	FIPS のサポート	1401
7	バックアップとリカバリ	1405
7.1	バックアップとリカバリの種類	1406
7.2	データベースバックアップ方法	1409
7.3	バックアップおよびリカバリ戦略の例	1410
7.3.1	バックアップポリシーの確立	1411
7.3.2	リカバリへのバックアップの使用	1413
7.3.3	バックアップ戦略サマリー	1413
7.4	バックアップへの mysqldump の使用	1414
7.4.1	mysqldump による SQL フォーマットでのデータのダンプ	1414
7.4.2	SQL フォーマットバックアップのリロード	1415

7.4.3	mysqldump による区切りテキストフォーマットでのデータのダンプ	1416
7.4.4	区切りテキストフォーマットバックアップのリロード	1417
7.4.5	mysqldump のヒント	1417
7.5	Point-in-Time (増分) リカバリ	1419
7.5.1	バイナリログを使用したポイントインタイムリカバリ	1419
7.5.2	イベントの位置を使用したポイントインタイムリカバリ	1421
7.6	MyISAM テーブルの保守とクラッシュリカバリ	1422
7.6.1	クラッシュリカバリへの myisamchk の使用	1423
7.6.2	MyISAM テーブルのエラーのチェック方法	1424
7.6.3	MyISAM テーブルの修復方法	1424
7.6.4	MyISAM テーブルの最適化	1426
7.6.5	MyISAM テーブル保守スケジュールのセットアップ	1427
8	最適化	1429
8.1	最適化の概要	1430
8.2	SQL ステートメントの最適化	1432
8.2.1	SELECT ステートメントの最適化	1432
8.2.2	サブクエリー、導出テーブル、ビュー参照および共通テーブル式の最適化	1479
8.2.3	INFORMATION_SCHEMA クエリーの最適化	1492
8.2.4	パフォーマンススキーマクエリーの最適化	1494
8.2.5	データ変更ステートメントの最適化	1496
8.2.6	データベース権限の最適化	1497
8.2.7	その他の最適化のヒント	1497
8.3	最適化とインデックス	1498
8.3.1	MySQL のインデックスの使用の仕組み	1498
8.3.2	主キーの最適化	1499
8.3.3	SPATIAL インデックス最適化	1499
8.3.4	外部キーの最適化	1500
8.3.5	カラムインデックス	1500
8.3.6	マルチカラムインデックス	1501
8.3.7	インデックスの使用の確認	1503
8.3.8	InnoDB および MyISAM インデックス統計コレクション	1503
8.3.9	B ツリーインデックスとハッシュインデックスの比較	1504
8.3.10	インデックス拡張の使用	1506
8.3.11	生成されたカラムインデックスのオプティマイザによる使用	1508
8.3.12	不可視のインデックス	1509
8.3.13	降順インデックス	1511
8.3.14	TIMESTAMP カラムからのインデックス付きリックアップ	1512
8.4	データベース構造の最適化	1514
8.4.1	データサイズの最適化	1514
8.4.2	MySQL データ型の最適化	1515
8.4.3	多数のテーブルの最適化	1516
8.4.4	MySQL での内部一時テーブルの使用	1518
8.4.5	データベースおよびテーブルの数に対する制限	1521
8.4.6	テーブルサイズの制限	1521
8.4.7	テーブルカラム数と行サイズの制限	1523
8.5	InnoDB テーブルの最適化	1525
8.5.1	InnoDB テーブルのストレージレイアウトの最適化	1525
8.5.2	InnoDB トランザクション管理の最適化	1526
8.5.3	InnoDB の読み取り専用トランザクションの最適化	1527
8.5.4	InnoDB redo ログの最適化	1527
8.5.5	InnoDB テーブルの一括データロード	1528
8.5.6	InnoDB クエリーの最適化	1530
8.5.7	InnoDB DDL 操作の最適化	1530
8.5.8	InnoDB ディスク I/O の最適化	1530
8.5.9	InnoDB 構成変数の最適化	1533
8.5.10	多くのテーブルのあるシステムに対する InnoDB の最適化	1535
8.6	MyISAM テーブルの最適化	1535
8.6.1	MyISAM クエリーの最適化	1535
8.6.2	MyISAM テーブルの一括データロード	1536
8.6.3	REPAIR TABLE ステートメントの最適化	1537

8.7 MEMORY テーブルの最適化	1538
8.8 クエリー実行プランの理解	1539
8.8.1 EXPLAIN によるクエリーの最適化	1539
8.8.2 EXPLAIN 出力フォーマット	1540
8.8.3 拡張 EXPLAIN 出力形式	1552
8.8.4 名前付き接続の実行計画情報の取得	1554
8.8.5 クエリーパフォーマンスの推定	1555
8.9 クエリーオプティマイザの制御	1555
8.9.1 クエリー計画評価の制御	1555
8.9.2 切り替え可能な最適化	1556
8.9.3 オプティマイザヒント	1565
8.9.4 インデックスヒント	1579
8.9.5 オプティマイザコストモデル	1581
8.9.6 オプティマイザ統計	1584
8.10 バッファリングとキャッシュ	1587
8.10.1 InnoDB バッファプールの最適化	1587
8.10.2 MyISAM キーキャッシュ	1587
8.10.3 プリベアドステートメントおよびストアプログラムのキャッシュ	1592
8.11 ロック操作の最適化	1593
8.11.1 内部ロック方法	1593
8.11.2 テーブルロックの問題	1595
8.11.3 同時挿入	1597
8.11.4 メタデータのロック	1597
8.11.5 外部ロック	1600
8.12 MySQL サーバーの最適化	1601
8.12.1 ディスク I/O の最適化	1601
8.12.2 シンボリックリンクの使用	1603
8.12.3 メモリーの使用の最適化	1605
8.13 パフォーマンスの測定 (ベンチマーク)	1611
8.13.1 式と関数の速度の測定	1611
8.13.2 独自のベンチマークの使用	1611
8.13.3 performance_schema によるパフォーマンスの測定	1612
8.14 サーバースレッド (プロセス) 情報の確認	1612
8.14.1 プロセスリストへのアクセス	1612
8.14.2 スレッドのコマンド値	1614
8.14.3 一般的なスレッドの状態	1616
8.14.4 レプリケーションソーススレッドの状態	1622
8.14.5 レプリケーション I/O スレッドの状態	1622
8.14.6 レプリケーション SQL スレッドの状態	1623
8.14.7 レプリケーション接続スレッドの状態	1624
8.14.8 NDB Cluster スレッドの状態	1624
8.14.9 イベントスケジューラスレッドの状態	1625
9 言語構造	1627
9.1 リテラル値	1627
9.1.1 文字列リテラル	1627
9.1.2 数値リテラル	1630
9.1.3 日付リテラルと時間リテラル	1630
9.1.4 16 進数リテラル	1632
9.1.5 ビット値リテラル	1634
9.1.6 boolean リテラル	1635
9.1.7 NULL 値	1635
9.2 スキーマオブジェクト名	1635
9.2.1 識別子の長さ制限	1637
9.2.2 識別子の修飾子	1638
9.2.3 識別子の小文字と大文字の区別	1639
9.2.4 識別子とファイル名のマッピング	1641
9.2.5 関数名の構文解析と解決	1642
9.3 キーワードと予約語	1645
9.4 ユーザー定義変数	1673
9.5 式	1676

9.6 クエリー属性	1680
9.7 コメント	1683
10 文字セット、照合順序、Unicode	1685
10.1 一般の文字セットおよび照合順序	1686
10.2 MySQL での文字セットと照合順序	1687
10.2.1 文字セットレパートリー	1689
10.2.2 メタデータ用の UTF-8	1691
10.3 文字セットと照合順序の指定	1692
10.3.1 照合の命名規則	1692
10.3.2 サーバー文字セットおよび照合順序	1693
10.3.3 データベース文字セットおよび照合順序	1693
10.3.4 テーブル文字セットおよび照合順序	1695
10.3.5 カラム文字セットおよび照合順序	1695
10.3.6 文字列リテラルの文字セットおよび照合順序	1696
10.3.7 各国語文字セット	1698
10.3.8 文字セットイントロデューサ	1698
10.3.9 文字セットと照合順序の割り当ての例	1700
10.3.10 ほかの DBMS との互換性	1701
10.4 接続文字セットおよび照合順序	1701
10.5 アプリケーションの文字セットおよび照合順序の構成	1707
10.6 エラーメッセージ文字セット	1708
10.7 カラム文字セットの変換	1709
10.8 照合順序の問題	1710
10.8.1 SQL ステートメントでの COLLATE の使用	1710
10.8.2 COLLATE 句の優先順位	1711
10.8.3 文字セットと照合順序の互換性	1711
10.8.4 式での照合の強制性	1711
10.8.5 バイナリ照合順序と_bin 照合順序	1712
10.8.6 照合順序の効果の例	1715
10.8.7 INFORMATION_SCHEMA 検索での照合の使用	1716
10.9 Unicode のサポート	1718
10.9.1 utf8mb4 文字セット (4 バイトの UTF-8 Unicode エンコーディング)	1719
10.9.2 utf8mb3 文字セット (3 バイトの UTF-8 Unicode エンコーディング)	1720
10.9.3 utf8 文字セット (utf8mb3 のエイリアス)	1720
10.9.4 ucs2 文字セット (UCS-2 Unicode エンコーディング)	1721
10.9.5 utf16 文字セット (UTF-16 Unicode エンコーディング)	1721
10.9.6 utf16le 文字セット (UTF-16LE Unicode エンコーディング)	1721
10.9.7 utf32 文字セット (UTF-32 Unicode エンコーディング)	1721
10.9.8 3 バイト Unicode 文字セットと 4 バイト Unicode 文字セット間の変換	1722
10.10 サポートされる文字セットおよび照合順序	1724
10.10.1 Unicode 文字セット	1725
10.10.2 西ヨーロッパの文字セット	1731
10.10.3 中央ヨーロッパの文字セット	1733
10.10.4 南ヨーロッパおよび中東の文字セット	1733
10.10.5 バルト語の文字セット	1734
10.10.6 キリル文字の文字セット	1734
10.10.7 アジアの文字セット	1735
10.10.8 バイナリ文字セット	1739
10.11 文字セットの制約	1740
10.12 エラーメッセージ言語の設定	1740
10.13 文字セットの追加	1741
10.13.1 文字定義配列	1742
10.13.2 複雑な文字セットの文字列照合のサポート	1743
10.13.3 複雑な文字セットのマルチバイト文字のサポート	1743
10.14 文字セットへの照合順序の追加	1744
10.14.1 照合順序の実装タイプ	1745
10.14.2 照合順序 ID の選択	1747
10.14.3 8 ビットの文字セットへの単純な照合順序の追加	1748
10.14.4 Unicode 文字セットへの UCA 照合順序の追加	1749
10.15 文字セットの構成	1755

10.16 MySQL Server のロケールサポート	1756
11 データ型	1761
11.1 数値データ型	1762
11.1.1 数値データ型の構文	1762
11.1.2 整数型 (真数値) - INTEGER、INT、SMALLINT、TINYINT、MEDIUMINT、BIGINT	1765
11.1.3 固定小数点型 (真数値) - DECIMAL、NUMERIC	1766
11.1.4 浮動小数点型 (概数値) - FLOAT、DOUBLE	1766
11.1.5 ビット値型 - BIT	1767
11.1.6 数値型の属性	1767
11.1.7 範囲外およびオーバーフローの処理	1768
11.2 日時データ型	1769
11.2.1 日時データ型の構文	1770
11.2.2 DATE、DATETIME、および TIMESTAMP 型	1772
11.2.3 TIME 型	1775
11.2.4 YEAR 型	1775
11.2.5 TIMESTAMP および DATETIME の自動初期化および更新機能	1776
11.2.6 時間値での小数秒	1779
11.2.7 日付と時間型間での変換	1780
11.2.8 日付の 2 桁の年	1780
11.3 文字列データ型	1781
11.3.1 文字列データ型の構文	1781
11.3.2 CHAR および VARCHAR 型	1784
11.3.3 BINARY および VARBINARY 型	1786
11.3.4 BLOB 型と TEXT 型	1787
11.3.5 ENUM 型	1788
11.3.6 SET 型	1791
11.4 空間データ型	1793
11.4.1 空間データ型	1794
11.4.2 OpenGIS ジオメトリモデル	1795
11.4.3 サポートされる空間データ形式	1800
11.4.4 ジオメトリの整形形式と妥当性	1803
11.4.5 空間参照システムのサポート	1804
11.4.6 空間カラムの作成	1805
11.4.7 空間カラムへのデータ移入	1805
11.4.8 空間データのフェッチ	1806
11.4.9 空間分析の最適化	1806
11.4.10 空間インデックスの作成	1807
11.4.11 空間インデックスの使用	1808
11.5 JSON データ型	1809
11.6 データ型デフォルト値	1824
11.7 データ型のストレージ要件	1826
11.8 カラムに適した型の選択	1830
11.9 その他のデータベースエンジンのデータ型の使用	1830
12 関数と演算子	1833
12.1 SQL 関数および演算子リファレンス	1835
12.2 ユーザー定義関数参照	1852
12.3 式評価での型変換	1856
12.4 演算子	1858
12.4.1 演算子の優先順位	1859
12.4.2 比較関数と演算子	1860
12.4.3 論理演算子	1866
12.4.4 割り当て演算子	1868
12.5 フロー制御関数	1869
12.6 数値関数と演算子	1872
12.6.1 算術演算子	1873
12.6.2 数学関数	1874
12.7 日付および時間関数	1883
12.8 文字列関数および演算子	1901
12.8.1 文字列比較関数および演算子	1914
12.8.2 正規表現	1917

12.8.3 関数結果の文字セットと照合順序	1926
12.9 MySQL で使用されるカレンダー	1927
12.10 全文検索関数	1927
12.10.1 自然言語全文検索	1929
12.10.2 ブール全文検索	1932
12.10.3 クエリー拡張を使用した全文検索	1937
12.10.4 全文ストップワード	1937
12.10.5 全文制限	1941
12.10.6 MySQL の全文検索の微調整	1942
12.10.7 全文インデックス付けのためのユーザー定義照合の追加	1945
12.10.8 ngram 全文パーサー	1946
12.10.9 MeCab フルテキストパーサープラグイン	1948
12.11 キャスト関数と演算子	1952
12.12 XML 関数	1965
12.13 ビット関数と演算子	1975
12.14 暗号化関数と圧縮関数	1985
12.15 ロック関数	1990
12.16 情報関数	1993
12.17 空間分析関数	2003
12.17.1 空間関数のリファレンス	2003
12.17.2 空間関数による引数処理	2007
12.17.3 WKT 値からジオメトリ値を作成する関数	2007
12.17.4 WKB 値からジオメトリ値を作成する関数	2009
12.17.5 ジオメトリ値を作成する MySQL 固有の関数	2011
12.17.6 ジオメトリ形式変換関数	2012
12.17.7 ジオメトリプロパティ関数	2013
12.17.8 空間演算子関数	2024
12.17.9 ジオメトリオブジェクト間の空間関係をテストする関数	2030
12.17.10 空間 Geohash 関数	2036
12.17.11 空間 GeoJSON 関数	2038
12.17.12 空間集計関数	2040
12.17.13 空間の便利な関数	2042
12.18 JSON 関数	2045
12.18.1 JSON 関数リファレンス	2046
12.18.2 JSON 値を作成する関数	2048
12.18.3 JSON 値を検索する関数	2049
12.18.4 JSON 値を変更する関数	2063
12.18.5 JSON 値属性を返す関数	2071
12.18.6 JSON テーブル関数	2074
12.18.7 JSON スキーマ検証関数	2078
12.18.8 JSON ユーティリティ関数	2083
12.19 グローバルトランザクション識別子 (GTID) で使用される機能	2088
12.20 集計関数	2090
12.20.1 集計関数の説明	2090
12.20.2 GROUP BY 修飾子	2098
12.20.3 MySQL での GROUP BY の処理	2104
12.20.4 機能依存性の検出	2107
12.21 ウィンドウ関数	2110
12.21.1 Window 関数の説明	2110
12.21.2 Window 関数の概念と構文	2116
12.21.3 ウィンドウ機能フレーム仕様	2120
12.21.4 名前付きウィンドウ	2123
12.21.5 ウィンドウ機能の制限事項	2124
12.22 パフォーマンススキーマ関数	2124
12.23 内部関数	2127
12.24 その他の関数	2128
12.25 高精度計算	2142
12.25.1 数値の型	2142
12.25.2 DECIMAL データ型の特性	2142
12.25.3 式の処理	2143

12.25.4	丸め動作	2145
12.25.5	高精度計算の例	2146
13	SQL ステートメント	2149
13.1	データ定義ステートメント	2150
13.1.1	アトミックデータ定義ステートメントのサポート	2150
13.1.2	ALTER DATABASE ステートメント	2156
13.1.3	ALTER EVENT ステートメント	2160
13.1.4	ALTER FUNCTION ステートメント	2162
13.1.5	ALTER INSTANCE ステートメント	2162
13.1.6	ALTER LOGFILE GROUP ステートメント	2163
13.1.7	ALTER PROCEDURE ステートメント	2165
13.1.8	ALTER SERVER ステートメント	2165
13.1.9	ALTER TABLE ステートメント	2165
13.1.10	ALTER TABLESPACE ステートメント	2187
13.1.11	ALTER VIEW ステートメント	2188
13.1.12	CREATE DATABASE ステートメント	2189
13.1.13	CREATE EVENT ステートメント	2189
13.1.14	CREATE FUNCTION ステートメント	2194
13.1.15	CREATE INDEX ステートメント	2194
13.1.16	CREATE LOGFILE GROUP ステートメント	2207
13.1.17	CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント	2209
13.1.18	CREATE SERVER ステートメント	2213
13.1.19	CREATE SPATIAL REFERENCE SYSTEM ステートメント	2214
13.1.20	CREATE TABLE ステートメント	2218
13.1.21	CREATE TABLESPACE ステートメント	2268
13.1.22	CREATE TRIGGER ステートメント	2275
13.1.23	CREATE VIEW ステートメント	2277
13.1.24	DROP DATABASE ステートメント	2281
13.1.25	DROP EVENT ステートメント	2282
13.1.26	DROP FUNCTION ステートメント	2282
13.1.27	DROP INDEX ステートメント	2282
13.1.28	DROP LOGFILE GROUP ステートメント	2282
13.1.29	DROP PROCEDURE および DROP FUNCTION ステートメント	2283
13.1.30	DROP SERVER ステートメント	2283
13.1.31	DROP SPATIAL REFERENCE SYSTEM ステートメント	2283
13.1.32	DROP TABLE ステートメント	2284
13.1.33	DROP TABLESPACE ステートメント	2285
13.1.34	DROP TRIGGER ステートメント	2286
13.1.35	DROP VIEW ステートメント	2286
13.1.36	RENAME TABLE ステートメント	2287
13.1.37	TRUNCATE TABLE ステートメント	2288
13.2	データ操作ステートメント	2290
13.2.1	CALL ステートメント	2290
13.2.2	DELETE ステートメント	2291
13.2.3	DO ステートメント	2295
13.2.4	HANDLER ステートメント	2295
13.2.5	IMPORT TABLE ステートメント	2297
13.2.6	INSERT ステートメント	2299
13.2.7	LOAD DATA ステートメント	2307
13.2.8	LOAD XML ステートメント	2316
13.2.9	REPLACE ステートメント	2323
13.2.10	SELECT ステートメント	2325
13.2.11	サブクエリー	2344
13.2.12	TABLE ステートメント	2358
13.2.13	UPDATE ステートメント	2360
13.2.14	VALUES ステートメント	2364
13.2.15	WITH (共通テーブル式)	2366
13.3	トランザクションステートメントおよびロックステートメント	2376
13.3.1	START TRANSACTION、COMMIT および ROLLBACK ステートメント	2376
13.3.2	ロールバックできないステートメント	2379

13.3.3	暗黙的なコミットを発生させるステートメント	2379
13.3.4	SAVEPOINT、ROLLBACK TO SAVEPOINT および RELEASE SAVEPOINT ステートメント	2380
13.3.5	LOCK INSTANCE FOR BACKUP および UNLOCK INSTANCE ステートメント	2381
13.3.6	LOCK TABLES および UNLOCK TABLES ステートメント	2382
13.3.7	SET TRANSACTION ステートメント	2387
13.3.8	XA トランザクション	2390
13.4	レプリケーションステートメント	2395
13.4.1	ソースサーバーを制御する SQL ステートメント	2395
13.4.2	レプリケーションサーバーを制御するための SQL ステートメント	2398
13.4.3	グループレプリケーションを制御するための SQL ステートメント	2433
13.5	プリペアドステートメント	2437
13.5.1	PREPARE ステートメント	2441
13.5.2	EXECUTE ステートメント	2443
13.5.3	DEALLOCATE PREPARE ステートメント	2443
13.6	複合ステートメントの構文	2443
13.6.1	BEGIN ... END 複合ステートメント	2443
13.6.2	ステートメントラベル	2444
13.6.3	DECLARE ステートメント	2444
13.6.4	ストアドプログラム内の変数	2445
13.6.5	フロー制御ステートメント	2446
13.6.6	カーソル	2450
13.6.7	条件の処理	2452
13.6.8	条件処理の制約	2476
13.7	データベース管理ステートメント	2477
13.7.1	アカウント管理ステートメント	2477
13.7.2	リソースグループ管理ステートメント	2520
13.7.3	テーブル保守ステートメント	2523
13.7.4	コンポーネント、プラグインおよびユーザー定義関数のステートメント	2536
13.7.5	CLONE ステートメント	2540
13.7.6	SET ステートメント	2541
13.7.7	SHOW ステートメント	2547
13.7.8	その他の管理ステートメント	2598
13.8	ユーティリティステートメント	2610
13.8.1	DESCRIBE ステートメント	2610
13.8.2	EXPLAIN ステートメント	2611
13.8.3	HELP ステートメント	2614
13.8.4	USE ステートメント	2615
14	MySQL データディクショナリ	2617
14.1	データディクショナリスキーマ	2617
14.2	ファイルベースのメタデータ記憶域の削除	2618
14.3	ディクショナリデータのトランザクション記憶域	2619
14.4	ディクショナリオブジェクトキャッシュ	2619
14.5	INFORMATION_SCHEMA とデータディクショナリの統合	2620
14.6	シリアライズディクショナリ情報 (SDI)	2622
14.7	データディクショナリの使用方法の違い	2622
14.8	データディクショナリの制限事項	2624
15	InnoDB ストレージエンジン	2625
15.1	InnoDB 入門	2626
15.1.1	InnoDB テーブルを使用する利点	2628
15.1.2	InnoDB テーブルのベストプラクティス	2629
15.1.3	InnoDB がデフォルトのストレージエンジンであるかどうかの確認	2629
15.1.4	InnoDB を使用したテストおよびベンチマーク	2629
15.2	InnoDB および ACID モデル	2630
15.3	InnoDB マルチバージョン	2631
15.4	InnoDB のアーキテクチャ	2632
15.5	InnoDB インメモリー構造	2633
15.5.1	バッファプール	2633
15.5.2	変更バッファ	2638
15.5.3	適応型ハッシュインデックス	2641
15.5.4	ログバッファ	2642

15.6 InnoDB オンディスク構造	2642
15.6.1 テーブル	2642
15.6.2 インデックス	2665
15.6.3 テーブルスペース	2672
15.6.4 二重書き込みバッファ	2693
15.6.5 redo ログ	2695
15.6.6 undo ログ	2699
15.7 InnoDB のロックおよびトランザクションモデル	2700
15.7.1 InnoDB ロック	2701
15.7.2 InnoDB トランザクションモデル	2705
15.7.3 InnoDB のさまざまな SQL ステートメントで設定されたロック	2713
15.7.4 ファントム行	2716
15.7.5 InnoDB のデッドロック	2717
15.7.6 トランザクションスケジューリング	2720
15.8 InnoDB の構成	2721
15.8.1 InnoDB の起動構成	2721
15.8.2 読み取り専用操作の InnoDB の構成	2727
15.8.3 InnoDB バッファプールの構成	2728
15.8.4 InnoDB のスレッド並列性の構成	2741
15.8.5 InnoDB バックグラウンド I/O スレッドの数の構成	2742
15.8.6 Linux での非同期 I/O の使用	2742
15.8.7 InnoDB I/O Capacity の構成	2743
15.8.8 スピンロックのポーリングの構成	2744
15.8.9 ページ構成	2745
15.8.10 InnoDB のオプティマイザ統計の構成	2747
15.8.11 インデックスページのマージしきい値の構成	2756
15.8.12 専用 MySQL Server の自動構成の有効化	2759
15.9 InnoDB のテーブルおよびページの圧縮	2761
15.9.1 InnoDB テーブルの圧縮	2761
15.9.2 InnoDB ページ圧縮	2774
15.10 InnoDB の行フォーマット	2777
15.11 InnoDB のディスク I/O とファイル領域管理	2783
15.11.1 InnoDB ディスク I/O	2783
15.11.2 ファイル領域管理	2784
15.11.3 InnoDB チェックポイント	2785
15.11.4 テーブルのデフラグ	2785
15.11.5 TRUNCATE TABLE によるディスク領域の再利用	2786
15.12 InnoDB とオンライン DDL	2786
15.12.1 オンライン DDL 操作	2787
15.12.2 オンライン DDL のパフォーマンスと同時実行性	2799
15.12.3 オンライン DDL 領域の要件	2802
15.12.4 オンライン DDL を使用した DDL ステートメントの簡略化	2803
15.12.5 オンライン DDL 失敗条件	2803
15.12.6 オンライン DDL の制限事項	2804
15.13 InnoDB 保存データ暗号化	2805
15.14 InnoDB の起動オプションおよびシステム変数	2813
15.15 InnoDB INFORMATION_SCHEMA テーブル	2894
15.15.1 圧縮に関する InnoDB INFORMATION_SCHEMA テーブル	2895
15.15.2 InnoDB INFORMATION_SCHEMA トランザクションおよびロック情報	2896
15.15.3 InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル	2903
15.15.4 InnoDB INFORMATION_SCHEMA FULLTEXT インデックステーブル	2908
15.15.5 InnoDB INFORMATION_SCHEMA バッファプールテーブル	2911
15.15.6 InnoDB INFORMATION_SCHEMA メトリックテーブル	2915
15.15.7 InnoDB INFORMATION_SCHEMA 一時テーブル情報テーブル	2922
15.15.8 INFORMATION_SCHEMA.FILES からの InnoDB テーブルスペースメタデータの取得	2923
15.16 InnoDB の MySQL パフォーマンススキーマとの統合	2924
15.16.1 パフォーマンススキーマを使用した InnoDB テーブルの ALTER TABLE の進行状況のモニタリング	2926
15.16.2 パフォーマンススキーマを使用した InnoDB Mutex 待機のモニタリング	2928
15.17 InnoDB モニター	2931

15.17.1 InnoDB モニターのタイプ	2931
15.17.2 InnoDB モニターの有効化	2932
15.17.3 InnoDB 標準モニターおよびロックモニターの出力	2933
15.18 InnoDB のバックアップとリカバリ	2937
15.18.1 InnoDB バックアップ	2938
15.18.2 InnoDB のリカバリ	2938
15.19 InnoDB と MySQL レプリケーション	2941
15.20 InnoDB memcached プラグイン	2942
15.20.1 InnoDB memcached プラグインの利点	2943
15.20.2 InnoDB memcached のアーキテクチャー	2944
15.20.3 InnoDB memcached プラグインの設定	2947
15.20.4 InnoDB memcached の複数の get および Range クエリーのサポート	2952
15.20.5 InnoDB memcached プラグインのセキュリティーに関する考慮事項	2954
15.20.6 InnoDB memcached プラグイン用のアプリケーションの記述	2956
15.20.7 InnoDB memcached プラグインとレプリケーション	2967
15.20.8 InnoDB memcached プラグインの内部	2970
15.20.9 InnoDB memcached プラグインのトラブルシューティング	2974
15.21 InnoDB のトラブルシューティング	2976
15.21.1 InnoDB の I/O に関する問題のトラブルシューティング	2977
15.21.2 InnoDB のリカバリの強制的な実行	2977
15.21.3 InnoDB データディクショナリの操作のトラブルシューティング	2979
15.21.4 InnoDB のエラー処理	2980
15.22 InnoDB の制限	2980
15.23 InnoDB の制限および制限事項	2981
16 代替ストレージエンジン	2983
16.1 ストレージエンジンの設定	2986
16.2 MyISAM ストレージエンジン	2987
16.2.1 MyISAM 起動オプション	2989
16.2.2 キーに必要な容量	2991
16.2.3 MyISAM テーブルのストレージフォーマット	2991
16.2.4 MyISAM テーブルの問題点	2994
16.3 MEMORY ストレージエンジン	2995
16.4 CSV ストレージエンジン	2999
16.4.1 CSV テーブルの修復と確認	3000
16.4.2 CSV の制限	3000
16.5 ARCHIVE ストレージエンジン	3001
16.6 BLACKHOLE ストレージエンジン	3002
16.7 MERGE ストレージエンジン	3004
16.7.1 MERGE テーブルの長所と短所	3007
16.7.2 MERGE テーブルの問題点	3008
16.8 FEDERATED ストレージエンジン	3009
16.8.1 FEDERATED ストレージエンジンの概要	3009
16.8.2 FEDERATED テーブルの作成方法	3010
16.8.3 FEDERATED ストレージエンジンの注記とヒント	3013
16.8.4 FEDERATED ストレージエンジンのリソース	3014
16.9 EXAMPLE ストレージエンジン	3014
16.10 ほかのストレージエンジン	3015
16.11 MySQL ストレージエンジンアーキテクチャーの概要	3015
16.11.1 プラガブルストレージエンジンのアーキテクチャー	3016
16.11.2 共通データベースサーバーレイヤー	3016
17 レプリケーション	3019
17.1 レプリケーションの構成	3020
17.1.1 バイナリログファイルの位置ベースのレプリケーション構成の概要	3021
17.1.2 バイナリログファイルの位置ベースのレプリケーションの設定	3021
17.1.3 グローバルトランザクション識別子を使用したレプリケーション	3032
17.1.4 オンラインサーバーでの GTID モードの変更	3053
17.1.5 MySQL マルチソースレプリケーション	3059
17.1.6 レプリケーションおよびバイナリロギングのオプションと変数	3065
17.1.7 一般的なレプリケーション管理タスク	3144
17.2 レプリケーションの実装	3150

17.2.1	レプリケーション形式	3150
17.2.2	レプリケーションチャンネル	3157
17.2.3	レプリケーションスレッド	3161
17.2.4	リレーログおよびレプリケーションメタデータリポジトリ	3163
17.2.5	サーバーがレプリケーションフィルタリングルールをどのように評価するか	3169
17.3	レプリケーションのセキュリティ	3176
17.3.1	暗号化接続を使用するためのレプリケーションの設定	3177
17.3.2	バイナリログファイルとリレーログファイルの暗号化	3179
17.3.3	レプリケーション権限チェック	3183
17.4	レプリケーションソリューション	3189
17.4.1	バックアップ用にレプリケーションを使用する	3189
17.4.2	レプリカの予期しない停止の処理	3193
17.4.3	行ベースのレプリケーションの監視	3195
17.4.4	異なるソースおよびレプリカのストレージエンジンでのレプリケーションの使用	3195
17.4.5	スケールアウトのためにレプリケーションを使用する	3196
17.4.6	異なるレプリカへの異なるデータベースのレプリケート	3198
17.4.7	レプリケーションパフォーマンスを改善する	3199
17.4.8	フェイルオーバー中のソースの切替え	3200
17.4.9	非同期接続フェイルオーバーによるソースの切替え	3202
17.4.10	準同期レプリケーション	3204
17.4.11	遅延レプリケーション	3209
17.5	レプリケーションの注釈とヒント	3212
17.5.1	レプリケーションの機能と問題	3212
17.5.2	MySQL バージョン間のレプリケーション互換性	3236
17.5.3	レプリケーションセットアップをアップグレードする	3237
17.5.4	レプリケーションのトラブルシューティング	3238
17.5.5	レプリケーションバグまたは問題を報告する方法	3239
18	グループレプリケーション	3241
18.1	グループレプリケーションのバックグラウンド	3242
18.1.1	レプリケーションテクノロジー	3243
18.1.2	グループレプリケーションのユースケース	3245
18.1.3	マルチプライマリモードとシングルプライマリモード	3246
18.1.4	グループレプリケーションサービス	3250
18.1.5	グループレプリケーションプラグインのアーキテクチャ	3252
18.2	はじめに	3254
18.2.1	単一プライマリモードでのグループレプリケーションのデプロイ	3254
18.2.2	グループレプリケーションのローカルでのデプロイ	3265
18.3	グループレプリケーションの監視	3266
18.3.1	グループレプリケーションサーバーの状態	3267
18.3.2	replication_group_members テーブル	3268
18.3.3	replication_group_member_stats テーブル	3268
18.4	グループレプリケーション操作	3269
18.4.1	オンライングループの構成	3269
18.4.2	トランザクション一貫性保証	3273
18.4.3	分散リカバリ	3278
18.4.4	ネットワークパーティション化	3292
18.4.5	IPv6 および IPv6 と IPv4 の混合グループのサポート	3297
18.4.6	グループレプリケーションでの MySQL Enterprise Backup の使用	3299
18.5	グループレプリケーションセキュリティ	3304
18.5.1	グループレプリケーション IP アドレスの権限	3304
18.5.2	Secure Socket Layer (SSL) を使用したグループ通信接続の保護	3306
18.5.3	分散リカバリ接続の保護	3308
18.6	グループレプリケーションのパフォーマンス	3311
18.6.1	グループ通信スレッドの微調整	3311
18.6.2	フロー制御	3312
18.6.3	メッセージ圧縮	3313
18.6.4	メッセージの断片化	3314
18.6.5	XCom キャッシュ管理	3315
18.6.6	障害検出およびネットワークパーティション化へのレスポンス	3316
18.7	グループレプリケーションのアップグレード	3321

18.7.1	グループ内の異なるメンバーバージョンの組合せ	3322
18.7.2	グループレプリケーションのオフラインアップグレード	3324
18.7.3	グループレプリケーションのオンラインアップグレード	3324
18.8	グループレプリケーションシステム変数	3328
18.9	要件と制限事項	3361
18.9.1	グループレプリケーションの要件	3361
18.9.2	グループレプリケーションの制限事項	3363
18.10	よくある質問	3365
19	MySQL Shell	3371
20	ドキュメントストアとしての MySQL の使用	3373
20.1	MySQL ドキュメントストアへのインタフェース	3374
20.2	ドキュメントストアの概念	3374
20.3	JavaScript クイックスタートガイド: ドキュメントストア用の MySQL Shell	3375
20.3.1	MySQL Shell	3376
20.3.2	world_x データベースのダウンロードおよびインポート	3377
20.3.3	ドキュメントとコレクション	3378
20.3.4	リレーショナルテーブル	3388
20.3.5	テーブル内のドキュメント	3393
20.4	Python クイックスタートガイド: ドキュメントストア用の MySQL Shell	3394
20.4.1	MySQL Shell	3395
20.4.2	world_x データベースのダウンロードおよびインポート	3396
20.4.3	ドキュメントとコレクション	3397
20.4.4	リレーショナルテーブル	3407
20.4.5	テーブル内のドキュメント	3412
20.5	X プラグイン	3413
20.5.1	X プラグイン のインストールの確認	3413
20.5.2	X プラグイン の無効化	3414
20.5.3	X プラグイン での暗号化接続の使用	3414
20.5.4	Caching SHA-2 認証プラグインでの X プラグイン の使用	3415
20.5.5	X プラグイン での接続圧縮	3415
20.5.6	X プラグイン のオプションおよび変数	3418
20.5.7	X プラグイン の監視	3437
21	InnoDB クラス	3439
22	InnoDB ReplicaSet	3443
23	MySQL NDB Cluster 8.0	3445
23.1	NDB Cluster の概要	3448
23.1.1	NDB Cluster のコア概念	3450
23.1.2	NDB Cluster ノード、ノードグループ、フラグメントレプリカ、およびパーティション	3452
23.1.3	NDB Cluster のハードウェア、ソフトウェア、およびネットワーク要件	3455
23.1.4	NDB Cluster の新機能	3456
23.1.5	NDB 8.0 で追加、非推奨または削除されたオプション、変数、およびパラメータ	3474
23.1.6	MySQL Server NDB Cluster と比較した InnoDB の使用	3479
23.1.7	NDB Cluster の既知の制限事項	3481
23.2	NDB Cluster のインストール	3492
23.2.1	Linux での NDB Cluster のインストール	3494
23.2.2	Windows への NDB Cluster のインストール	3502
23.2.3	NDB Cluster の初期構成	3509
23.2.4	NDB Cluster の初期起動	3511
23.2.5	テーブルとデータを含む NDB Cluster の例	3512
23.2.6	NDB Cluster の安全なシャットダウンと再起動	3515
23.2.7	NDB Cluster のアップグレードおよびダウングレード	3516
23.2.8	NDB Cluster Auto-Installer (サポートされなくなりました)	3518
23.3	NDB Cluster の構成	3540
23.3.1	NDB Cluster のクイックテスト設定	3541
23.3.2	NDB Cluster 構成パラメータ、オプション、および変数の概要	3543
23.3.3	NDB Cluster 構成ファイル	3560
23.3.4	NDB Cluster での高速インターコネクトの使用	3724
23.4	NDB Cluster プログラム	3724
23.4.1	ndbd — NDB Cluster データノードデーモン	3724
23.4.2	ndbinfo_select_all — ndbinfo テーブルからの選択	3731

23.4.3	ndbmtd	— NDB Cluster データノードデーモン (マルチスレッド)	3733
23.4.4	ndb_mgmd	— NDB Cluster 管理サーバーデーモン	3734
23.4.5	ndb_mgm	— NDB Cluster 管理クライアント	3741
23.4.6	ndb_blob_tool	— NDB Cluster テーブルの BLOB および TEXT カラムのチェックおよび修復	3743
23.4.7	ndb_config	— NDB Cluster 構成情報の抽出	3745
23.4.8	ndb_delete_all	— NDB テーブルからのすべての行の削除	3753
23.4.9	ndb_desc	— NDB テーブルの表示	3754
23.4.10	ndb_drop_index	— NDB テーブルからのインデックスの削除	3760
23.4.11	ndb_drop_table	— NDB テーブルの削除	3761
23.4.12	ndb_error_reporter	— NDB エラーレポートユーティリティ	3761
23.4.13	ndb_import	— NDB への CSV データのインポート	3763
23.4.14	ndb_index_stat	— NDB インデックス統計ユーティリティ	3774
23.4.15	ndb_move_data	— NDB データコピーユーティリティ	3779
23.4.16	ndb_perror	— NDB エラーメッセージ情報の取得	3782
23.4.17	ndb_print_backup_file	— NDB バックアップファイルの内容の出力	3783
23.4.18	ndb_print_file	— NDB デスクデータファイル内容の出力	3784
23.4.19	ndb_print_frag_file	— NDB フラグメントリストファイルの内容の出力	3784
23.4.20	ndb_print_schema_file	— NDB スキーマファイル内容の出力	3785
23.4.21	ndb_print_sys_file	— NDB システムファイル内容の出力	3785
23.4.22	ndb_redo_log_reader	— クラスタ redo ログの内容の確認および印刷	3786
23.4.23	ndb_restore	— NDB Cluster バックアップの復元	3788
23.4.24	ndb_select_all	— NDB テーブルの行の出力	3814
23.4.25	ndb_select_count	— NDB テーブルの行数の出力	3817
23.4.26	ndb_setup.py	— NDB Cluster 用ブラウザベースの Auto-Installer の起動 (非推奨)	3818
23.4.27	ndb_show_tables	— NDB テーブルのリストの表示	3821
23.4.28	ndb_size.pl	— NDBCLUSTER サイズ要件エスチメータ	3822
23.4.29	ndb_top	— NDB スレッドの CPU 使用率情報の表示	3825
23.4.30	ndb_waiter	— NDB Cluster が特定のステータスに達するまで待機	3830
23.4.31	ndbxfrm	— NDB Cluster によって作成されたファイルの圧縮、圧縮解除、暗号化、および復号化	3832
23.4.32	NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション		3834
23.5	NDB Cluster の管理		3838
23.5.1	NDB Cluster 管理クライアントのコマンド		3839
23.5.2	NDB Cluster ログメッセージ		3843
23.5.3	NDB Cluster で生成されるイベントレポート		3858
23.5.4	NDB Cluster 起動フェーズのサマリー		3869
23.5.5	NDB Cluster のローリング再起動の実行		3871
23.5.6	NDB Cluster のシングルユーザーモード		3873
23.5.7	NDB Cluster データノードのオンラインでの追加		3874
23.5.8	NDB Cluster のオンラインバックアップ		3884
23.5.9	NDB Cluster での MySQL Server の使用		3889
23.5.10	NDB Cluster デスクデータテーブル		3890
23.5.11	NDB Cluster での ALTER TABLE を使用したオンライン操作		3896
23.5.12	NDB_STORED_USER での分散 MySQL 権限		3899
23.5.13	NDB API 統計のカウントと変数		3900
23.5.14	ndbinfo: NDB Cluster 情報データベース		3910
23.5.15	NDB Cluster の INFORMATION_SCHEMA テーブル		3976
23.5.16	クイックリファレンス: NDB Cluster SQL ステートメント		3976
23.5.17	NDB Cluster のセキュリティの問題		3982
23.6	NDB Cluster レプリケーション		3989
23.6.1	NDB Cluster レプリケーション: 省略形と記号		3991
23.6.2	NDB Cluster レプリケーションの一般的な要件		3991
23.6.3	NDB Cluster レプリケーションの既知の問題		3992
23.6.4	NDB Cluster レプリケーションスキーマおよびテーブル		3999
23.6.5	NDB Cluster のレプリケーションの準備		4001
23.6.6	NDB Cluster レプリケーションの開始 (シングルレプリケーションチャンネル)		4003
23.6.7	NDB Cluster レプリケーションでの 2 つのレプリケーションチャンネルの使用		4005
23.6.8	NDB Cluster レプリケーションによるフェイルオーバーの実装		4005
23.6.9	NDB Cluster レプリケーションによる NDB Cluster バックアップ		4007
23.6.10	NDB Cluster レプリケーション: 双方向および循環レプリケーション		4013

23.6.11 NDB Cluster レプリケーションの競合解決	4016
23.7 NDB Cluster リリースノート	4028
24 パーティション化	4029
24.1 MySQL のパーティショニングの概要	4030
24.2 パーティショニングタイプ	4032
24.2.1 RANGE パーティショニング	4034
24.2.2 LIST パーティショニング	4038
24.2.3 COLUMNS パーティショニング	4040
24.2.4 HASH パーティショニング	4047
24.2.5 KEY パーティショニング	4049
24.2.6 サブパーティショニング	4051
24.2.7 MySQL パーティショニングによる NULL の扱い	4053
24.3 パーティション管理	4057
24.3.1 RANGE および LIST パーティションの管理	4057
24.3.2 HASH および KEY パーティションの管理	4063
24.3.3 パーティションとサブパーティションをテーブルと交換する	4064
24.3.4 パーティションの保守	4071
24.3.5 パーティションに関する情報を取得する	4072
24.4 パーティションプルーニング	4074
24.5 パーティション選択	4077
24.6 パーティショニングの制約と制限	4082
24.6.1 パーティショニングキー、主キー、および一意キー	4088
24.6.2 ストレージエンジンに関連するパーティショニング制限	4091
24.6.3 関数に関連するパーティショニング制限	4092
25 ストアドオブジェクト	4095
25.1 ストアドプログラムの定義	4096
25.2 ストアドルーチンの使用	4097
25.2.1 ストアドルーチンの構文	4097
25.2.2 ストアドルーチンと MySQL 権限	4098
25.2.3 ストアドルーチンのメタデータ	4099
25.2.4 ストアドプロシージャ、関数、トリガー、および LAST_INSERT_ID()	4099
25.3 トリガーの使用	4099
25.3.1 トリガーの構文と例	4100
25.3.2 トリガーのメタデータ	4103
25.4 イベントスケジューラの使用	4104
25.4.1 イベントスケジューラの概要	4104
25.4.2 イベントスケジューラの構成	4105
25.4.3 イベント構文	4107
25.4.4 イベントメタデータ	4107
25.4.5 イベントスケジューラのスレータス	4108
25.4.6 イベントスケジューラと MySQL 権限	4108
25.5 ビューの使用	4111
25.5.1 ビューの構文	4111
25.5.2 ビュー処理アルゴリズム	4111
25.5.3 更新可能および挿入可能なビュー	4112
25.5.4 WITH CHECK OPTION 句の表示	4115
25.5.5 ビューのメタデータ	4116
25.6 ストアドオブジェクトのアクセス制御	4116
25.7 ストアドプログラムバイナリロギング	4119
25.8 ストアドプログラムの制約	4125
25.9 ビューの制約	4128
26 INFORMATION_SCHEMA テーブル	4131
26.1 はじめに	4132
26.2 INFORMATION_SCHEMA ADMINISTRABLE_ROLE_AUTHORIZATIONS テーブル	4135
26.3 INFORMATION_SCHEMA APPLICABLE_ROLES テーブル	4136
26.4 INFORMATION_SCHEMA CHARACTER_SETS テーブル	4136
26.5 INFORMATION_SCHEMA CHECK_CONSTRAINTS テーブル	4137
26.6 INFORMATION_SCHEMA COLLATIONS テーブル	4137
26.7 INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY テーブル	4138
26.8 INFORMATION_SCHEMA COLUMNS テーブル	4138

26.9 INFORMATION_SCHEMA COLUMNS_EXTENSIONS テーブル	4141
26.10 INFORMATION_SCHEMA COLUMN_PRIVILEGES テーブル	4141
26.11 INFORMATION_SCHEMA COLUMN_STATISTICS テーブル	4142
26.12 INFORMATION_SCHEMA ENABLED_ROLES テーブル	4143
26.13 INFORMATION_SCHEMA ENGINES テーブル	4143
26.14 INFORMATION_SCHEMA EVENTS テーブル	4144
26.15 INFORMATION_SCHEMA FILES テーブル	4147
26.16 INFORMATION_SCHEMA KEY_COLUMN_USAGE テーブル	4155
26.17 INFORMATION_SCHEMA ndb_transid_mysql_connection_map テーブル	4156
26.18 INFORMATION_SCHEMA KEYWORDS テーブル	4157
26.19 INFORMATION_SCHEMA OPTIMIZER_TRACE テーブル	4157
26.20 INFORMATION_SCHEMA PARAMETERS テーブル	4158
26.21 INFORMATION_SCHEMA PARTITIONS テーブル	4159
26.22 INFORMATION_SCHEMA PLUGINS テーブル	4162
26.23 INFORMATION_SCHEMA PROCESSLIST テーブル	4163
26.24 INFORMATION_SCHEMA PROFILING テーブル	4165
26.25 INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS テーブル	4166
26.26 INFORMATION_SCHEMA RESOURCE_GROUPS テーブル	4167
26.27 INFORMATION_SCHEMA ROLE_COLUMN_GRANTS テーブル	4167
26.28 INFORMATION_SCHEMA ROLE_ROUTINE_GRANTS テーブル	4168
26.29 INFORMATION_SCHEMA ROLE_TABLE_GRANTS テーブル	4169
26.30 INFORMATION_SCHEMA ROUTINES テーブル	4169
26.31 INFORMATION_SCHEMA SCHEMATA テーブル	4172
26.32 INFORMATION_SCHEMA SCHEMATA_EXTENSIONS テーブル	4173
26.33 INFORMATION_SCHEMA SCHEMA_PRIVILEGES テーブル	4173
26.34 INFORMATION_SCHEMA STATISTICS テーブル	4174
26.35 INFORMATION_SCHEMA ST_GEOMETRY_COLUMNS テーブル	4176
26.36 INFORMATION_SCHEMA ST_SPATIAL_REFERENCE_SYSTEMS テーブル	4177
26.37 INFORMATION_SCHEMA ST_UNITS_OF_MEASURE テーブル	4178
26.38 INFORMATION_SCHEMA TABLES テーブル	4179
26.39 INFORMATION_SCHEMA TABLES_EXTENSIONS テーブル	4182
26.40 INFORMATION_SCHEMA TABLESPACES テーブル	4183
26.41 INFORMATION_SCHEMA TABLESPACES_EXTENSIONS テーブル	4183
26.42 INFORMATION_SCHEMA TABLE_CONSTRAINTS テーブル	4183
26.43 INFORMATION_SCHEMA TABLE_CONSTRAINTS_EXTENSIONS テーブル	4184
26.44 INFORMATION_SCHEMA TABLE_PRIVILEGES テーブル	4184
26.45 INFORMATION_SCHEMA TRIGGERS テーブル	4185
26.46 INFORMATION_SCHEMA USER_ATTRIBUTES テーブル	4187
26.47 INFORMATION_SCHEMA USER_PRIVILEGES テーブル	4188
26.48 INFORMATION_SCHEMA VIEWS テーブル	4189
26.49 INFORMATION_SCHEMA VIEW_ROUTINE_USAGE テーブル	4190
26.50 INFORMATION_SCHEMA VIEW_TABLE_USAGE テーブル	4191
26.51 INFORMATION_SCHEMA InnoDB テーブル	4191
26.51.1 INFORMATION_SCHEMA INNODB_BUFFER_PAGE テーブル	4191
26.51.2 INFORMATION_SCHEMA INNODB_BUFFER_PAGE_LRU テーブル	4195
26.51.3 INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS テーブル	4198
26.51.4 INFORMATION_SCHEMA INNODB_CACHED_INDEXES テーブル	4201
26.51.5 INFORMATION_SCHEMA INNODB_CMP および INNODB_CMP_RESET テーブル	4201
26.51.6 INFORMATION_SCHEMA INNODB_CMPMEM および INNODB_CMPMEM_RESET テーブル	4203
26.51.7 INFORMATION_SCHEMA INNODB_CMP_PER_INDEX および INNODB_CMP_PER_INDEX_RESET テーブル	4204
26.51.8 INFORMATION_SCHEMA INNODB_COLUMNS テーブル	4205
26.51.9 INFORMATION_SCHEMA INNODB_DATAFILES テーブル	4207
26.51.10 INFORMATION_SCHEMA INNODB_FIELDS テーブル	4207
26.51.11 INFORMATION_SCHEMA INNODB_FOREIGN テーブル	4208
26.51.12 INFORMATION_SCHEMA INNODB_FOREIGN_COLS テーブル	4209
26.51.13 INFORMATION_SCHEMA INNODB_FT_BEING_DELETED テーブル	4209
26.51.14 INFORMATION_SCHEMA INNODB_FT_CONFIG テーブル	4210
26.51.15 INFORMATION_SCHEMA INNODB_FT_DEFAULT_STOPWORD テーブル	4211

26.51.16	INFORMATION_SCHEMA INNODB_FT_DELETED テーブル	4212
26.51.17	INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE テーブル	4213
26.51.18	INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE テーブル	4214
26.51.19	INFORMATION_SCHEMA INNODB_INDEXES テーブル	4215
26.51.20	INFORMATION_SCHEMA INNODB_LOCKS テーブル	4217
26.51.21	INFORMATION_SCHEMA INNODB_LOCK_WAITS テーブル	4218
26.51.22	INFORMATION_SCHEMA INNODB_METRICS テーブル	4218
26.51.23	INFORMATION_SCHEMA INNODB_SESSION_TEMP_TABLESPACES テーブル	4220
26.51.24	INFORMATION_SCHEMA INNODB_TABLES テーブル	4221
26.51.25	INFORMATION_SCHEMA INNODB_TABLESPACES テーブル	4222
26.51.26	INFORMATION_SCHEMA INNODB_TABLESPACES_BRIEF テーブル	4224
26.51.27	INFORMATION_SCHEMA INNODB_TABLESTATS ビュー	4225
26.51.28	INFORMATION_SCHEMA INNODB_TEMP_TABLE_INFO テーブル	4226
26.51.29	INFORMATION_SCHEMA INNODB_TRX テーブル	4227
26.51.30	INFORMATION_SCHEMA INNODB_VIRTUAL テーブル	4230
26.52	INFORMATION_SCHEMA スレッドプールテーブル	4231
26.52.1	INFORMATION_SCHEMA TP_THREAD_GROUP_STATE テーブル	4231
26.52.2	INFORMATION_SCHEMA TP_THREAD_GROUP_STATS テーブル	4232
26.52.3	INFORMATION_SCHEMA TP_THREAD_STATE テーブル	4232
26.53	INFORMATION_SCHEMA の接続制御テーブル	4233
26.53.1	INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS テーブル	4233
26.54	INFORMATION_SCHEMA MySQL Enterprise Firewall テーブル	4233
26.54.1	INFORMATION_SCHEMA MYSQL_FIREWALL_USERS テーブル	4233
26.54.2	INFORMATION_SCHEMA MYSQL_FIREWALL_WHITELIST テーブル	4234
26.55	SHOW ステートメントの拡張	4234
27	MySQL パフォーマンススキーマ	4237
27.1	パフォーマンススキーマクイックスタート	4239
27.2	パフォーマンススキーマビルド構成	4244
27.3	パフォーマンススキーマ起動構成	4245
27.4	パフォーマンススキーマ実行時構成	4247
27.4.1	パフォーマンススキーマイベントタイミング	4247
27.4.2	パフォーマンススキーマイベントフィルタリング	4250
27.4.3	イベントの事前フィルタリング	4251
27.4.4	インストゥルメントによる事前フィルタリング	4252
27.4.5	オブジェクトによる事前フィルタリング	4253
27.4.6	スレッドによる事前フィルタリング	4255
27.4.7	コンシューマによる事前フィルタリング	4257
27.4.8	コンシューマ構成の例	4259
27.4.9	フィルタリング操作のインストゥルメントまたはコンシューマの指定	4264
27.4.10	インストゥルメントされているものの特長	4264
27.5	パフォーマンススキーマクエリ	4265
27.6	パフォーマンススキーマインストゥルメント命名規則	4265
27.7	パフォーマンススキーマステータスマニタリング	4269
27.8	パフォーマンススキーマの原子的および分子的イベント	4272
27.9	現在および過去のイベントのパフォーマンススキーマテーブル	4272
27.10	パフォーマンススキーマのステートメントダイジェストとサンプリング	4273
27.11	パフォーマンススキーマの一般的なテーブル特性	4277
27.12	パフォーマンススキーマテーブルの説明	4278
27.12.1	パフォーマンススキーマテーブルインデックス	4279
27.12.2	パフォーマンススキーマセットアップテーブル	4282
27.12.3	パフォーマンススキーマインスタンステーブル	4289
27.12.4	パフォーマンススキーマ待機イベントテーブル	4294
27.12.5	パフォーマンススキーマステージイベントテーブル	4299
27.12.6	パフォーマンススキーマステートメントイベントテーブル	4305
27.12.7	パフォーマンススキーマのトランザクションテーブル	4315
27.12.8	パフォーマンススキーマ接続テーブル	4322
27.12.9	パフォーマンススキーマ接続属性テーブル	4325
27.12.10	パフォーマンススキーマのユーザー定義変数テーブル	4329
27.12.11	パフォーマンススキーマレプリケーションテーブル	4329

27.12.12 パフォーマンススキーマ NDB Cluster テーブル	4348
27.12.13 パフォーマンススキーマロックテーブル	4350
27.12.14 パフォーマンススキーマシステム変数テーブル	4358
27.12.15 パフォーマンススキーマのステータス変数のテーブル	4363
27.12.16 パフォーマンススキーマスレッドプールテーブル	4364
27.12.17 パフォーマンススキーマクローンテーブル	4369
27.12.18 パフォーマンススキーマサマリーテーブル	4371
27.12.19 パフォーマンススキーマのその他のテーブル	4397
27.13 パフォーマンススキーマオプションおよび変数リファレンス	4415
27.14 パフォーマンススキーマコマンドオプション	4418
27.15 パフォーマンススキーマシステム変数	4419
27.16 パフォーマンススキーマステータス変数	4435
27.17 パフォーマンススキーマのメモリー割り当てモデル	4438
27.18 パフォーマンススキーマとプラグイン	4439
27.19 問題を診断するためのパフォーマンススキーマの使用	4439
27.19.1 パフォーマンススキーマを使用したクエリープロファイリング	4440
27.19.2 親イベント情報の取得	4442
27.20 パフォーマンススキーマの制約	4443
28 MySQL sys スキーマ	4445
28.1 sys スキーマを使用するための前提条件	4445
28.2 sys スキーマの使用	4446
28.3 sys スキーマ進捗レポート	4447
28.4 sys スキーマオブジェクトリファレンス	4447
28.4.1 sys スキーマオブジェクトインデックス	4447
28.4.2 sys スキーマのテーブルおよびトリガー	4451
28.4.3 sys スキーマビュー	4453
28.4.4 sys スキーマスタアドプロシージャ	4491
28.4.5 sys スキーマスタアドファンクション	4509
29 Connector および API	4521
29.1 MySQL Connector/C++	4524
29.2 MySQL Connector/J	4524
29.3 MySQL Connector/NET	4524
29.4 MySQL Connector/ODBC	4524
29.5 MySQL Connector/Python	4525
29.6 MySQL Connector/Node.js	4525
29.7 MySQL C API	4525
29.8 MySQL PHP API	4525
29.9 MySQL Perl API	4525
29.10 MySQL Python API	4526
29.11 MySQL Ruby API	4526
29.11.1 MySQL/Ruby API	4526
29.11.2 Ruby/MySQL API	4526
29.12 MySQL Tcl API	4526
29.13 MySQL Eiffel ラッパー	4526
30 MySQL Enterprise Edition	4529
30.1 MySQL Enterprise Monitor の概要	4529
30.2 MySQL Enterprise Backup の概要	4530
30.3 MySQL Enterprise Security の概要	4531
30.4 MySQL Enterprise Encryption の概要	4531
30.5 MySQL Enterprise Audit の概要	4531
30.6 MySQL Enterprise Firewall の概要	4531
30.7 MySQL Enterprise Thread Pool の概要	4532
30.8 MySQL Enterprise Data Masking and De-Identification の概要	4532
31 MySQL Workbench	4533
32 OCI マーケットプレイス上の MySQL	4535
32.1 Oracle Cloud Infrastructure に MySQL EE をデプロイするための前提条件	4535
32.2 Oracle Cloud Infrastructure での MySQL EE のデプロイ	4535
32.3 ネットワークアクセスの構成	4537
32.4 Connecting	4537
32.5 Maintenance	4538

A MySQL 8.0 のよくある質問	4539
A.1 MySQL 8.0 FAQ: 全般	4539
A.2 MySQL 8.0 FAQ: ストレージエンジン	4541
A.3 MySQL 8.0 FAQ: サーバー SQL モード	4541
A.4 MySQL 8.0 FAQ: ストアドプロシージャおよびストアドファンクション	4542
A.5 MySQL 8.0 FAQ: トリガー	4546
A.6 MySQL 8.0 FAQ: ビュー	4548
A.7 MySQL 8.0 FAQ: INFORMATION_SCHEMA	4549
A.8 MySQL 8.0 FAQ: 移行	4549
A.9 MySQL 8.0 FAQ: セキュリティー	4550
A.10 MySQL 8.0 FAQ: NDB Cluster	4551
A.11 MySQL 8.0 FAQ: MySQL の中国語、日本語、および韓国語の文字セット	4563
A.12 MySQL 8.0 FAQ: コネクタおよび API	4573
A.13 MySQL 8.0 FAQ : C API、libmysql	4573
A.14 MySQL 8.0 FAQ: レプリケーション	4574
A.15 MySQL 8.0 FAQ: MySQL Enterprise Thread Pool	4577
A.16 MySQL 8.0 FAQ : InnoDB 変更バッファ	4578
A.17 MySQL 8.0 FAQ : InnoDB 保存データ暗号化	4580
A.18 MySQL 8.0 FAQ : 仮想化のサポート	4582
B エラーメッセージと一般的な問題	4583
B.1 エラーメッセージのソースと要素	4583
B.2 エラー情報インタフェース	4585
B.3 問題および一般的なエラー	4587
B.3.1 問題の原因を判別する方法	4587
B.3.2 MySQL プログラム使用時の一般的なエラー	4588
B.3.3 管理関連の問題	4598
B.3.4 クエリー関連の問題	4605
B.3.5 オプティマイザ関連の問題	4611
B.3.6 テーブル定義関連の問題	4612
B.3.7 MySQL の既知の問題	4613
C インデックス	4617
MySQL 用語集	5335

序文と法的通知

これは、MySQL Database System バージョン 8.0 (リリース 8.0.29 まで) のリファレンスマニュアルです。MySQL 8.0 のマイナーバージョン間での相違点は、リリース番号 (8.0.x) に関連した現在のテキストに記されます。ライセンス情報については、[法的通知](#)を参照してください。

MySQL 8.0 と以前のバージョンとの間に機能などの多くの相違点があるため、このマニュアルは MySQL ソフトウェアの古いバージョンで用することは想定されていません。MySQL ソフトウェアの以前のリリースを使用している場合は、該当するマニュアルを参照してください。たとえば、[MySQL 5.7 リファレンスマニュアル](#)には、5.7 シリーズの MySQL ソフトウェアリリースが取り上げられています。

information-MySQL 8.0 のライセンス。この製品には、ライセンスのもとで使用されるサードパーティ製ソフトウェアが含まれる場合があります。コマーシャルリリースの MySQL 8.0 を使用している場合、「[MySQL 8.0 コマーシャルリリース ライセンス情報ユーザーマニュアル](#)」でライセンス情報 (このコマーシャルリリースに含まれる可能性があるサードパーティソフトウェアに関連するライセンス情報など) を参照してください。MySQL 8.0 のコミュニティリリースを使用している場合、「[MySQL 8.0 コミュニティリリース ライセンス情報ユーザーマニュアル](#)」でライセンス情報 (この Community リリースに含まれる可能性のあるサードパーティソフトウェアに関連するライセンス情報など) を確認してください。

information-MySQL NDB Cluster 8.0 のライセンス。コマーシャルリリースの MySQL NDB Cluster 8.0 を使用している場合、「[MySQL NDB Cluster 8.0 コマーシャルリリース ライセンス情報ユーザーマニュアル](#)」でライセンス情報 (このコマーシャルリリースに含まれる可能性があるサードパーティソフトウェアに関連するライセンス情報など) を参照してください。MySQL NDB Cluster 8.0 のコミュニティリリースを使用している場合は、この Community リリースに含まれている可能性のあるサードパーティ製ソフトウェアに関連するライセンス情報など、ライセンス情報について「[MySQL NDB Cluster 8.0 コミュニティリリース ライセンス情報ユーザーマニュアル](#)」を参照してください。

法律上の注意点

Copyright © 1997, 2021, Oracle and/or its affiliates.

このソフトウェアおよび関連ドキュメントの使用と開示は、ライセンス契約の制約条件に従うものとし、知的財産に関する法律により保護されています。ライセンス契約で明示的に許諾されている場合もしくは法律によって認められている場合を除き、形式、手段に関係なく、いかなる部分も使用、複製、複製、翻訳、放送、修正、ライセンス供与、送信、配布、発表、実行、公開または表示することはできません。このソフトウェアのリバース・エンジニアリング、逆アセンブル、逆コンパイルは互換性のために法律によって規定されている場合を除き、禁止されています。

ここに記載された情報は予告なしに変更される場合があります。また、誤りが無いことの保証はいたしかねます。誤りを見つけた場合は、オラクルまでご連絡ください。

このソフトウェアまたは関連ドキュメントが米国政府機関もしくは米国政府機関に代わってこのソフトウェアまたは関連ドキュメントをライセンスされた者に提供される場合は、次の通知が適用されます:

U.S. GOVERNMENT END USERS: Oracle プログラム (オペレーティングシステム、統合ソフトウェア、提供されたハードウェアに埋め込まれたプログラム、インストールまたはアクティブ化されたプログラム、およびそのようなプログラムの変更を含む) および Oracle コンピュータのドキュメント、あるいは米国政府機関のエンドユーザーが配信またはアクセスするその他の Oracle データは、該当する連邦政府調達規約および政府機関固有の補足規約に従う商用コンピュータソフトウェアまたは商用コンピュータソフトウェアのドキュメントです。そのため、使用、複製、複製、リリース、表示、開示、変更、デリバティブ作業の準備、i) Oracle プログラム (オペレーティングシステム、統合ソフトウェア、提供されたハードウェアに埋め込まれたプログラム、インストールまたはアクティブ化されたプログラム、およびそのようなプログラムの変更を含む)、ii) Oracle コンピュータのドキュメントや iii) その他の Oracle データは、該当する契約に含まれているライセンスに指定されている権利および制限の対象となります。Oracle クラウドサービスの米国政府の使用を管理する用語は、このようなサービスに適用可能な契約によって定義されます。その他の権限は米国政府には付与されません。

このソフトウェアまたはハードウェアは、さまざまな情報管理アプリケーションで一般的に使用するために開発されています。個人傷害のリスクを引き起こす可能性のあるアプリケーションなど、本質的に危険なアプリケーションでの使用を目的とした開発や使用は行われません。このソフトウェアまたはハードウェアを危険が伴うアプリケーションで使用する場合は、安全に使用するために、適切なフェイルセーフ、バックアップ、冗長性などの対策を講じる必要があります。Oracle Corporation およびその関連企業は、このソフトウェアまたはハードウェアを危険なアプリケーションで使用することによって生じる損害に対する責任を免責します。

Oracle および Java は、Oracle およびその関連企業の登録商標です。その他の社名、商品名等は各社の商標または登録商標である場合があります。

Intel および Intel Inside は、Intel Corporation の商標または登録商標です。すべての SPARC の商標はライセンスのもとで使用し、SPARC International, Inc. の商標または登録商標です。AMD、Epyc および AMD ロゴは、Advanced Micro Devices の商標または登録商標です。UNIX は、The Open Group の登録商標です。

このソフトウェアまたはハードウェアとドキュメントは、サードパーティのコンテンツ、製品およびサービスへのアクセス、あるいはそれらに関する情報を提供する場合があります。Oracle Corporation およびその関連企業は、ユーザーと Oracle の間で適切な契約に特に記載されていないかぎり、サードパーティのコンテンツ、製品およびサービスに関するあらゆる種類の保証を責任を負い、明示的に免責するものではありません。Oracle Corporation およびその関連企業は、サードパーティコンテンツ、製品またはサービスへのアクセスまたは使用によって発生した損失、コストまたは損害について責任を負いません。ただし、ユーザーと Oracle 間の適用可能な契約で規定されている場合を除きます。

このドキュメントは、GPL ライセンスに基づき配布されるものではありません。このドキュメントの使用は、次の条項に従います。

個人的な使用にかぎり、このドキュメントの出力コピーを作成できます。実際のコンテンツをいっさい変更したり編集したりしないかぎり、ほかの形式に変換することは許可されています。このドキュメントはいかなる形態またはいかなるメディアであっても公開または配布されないものとします。ただし、ドキュメントが同じメディアでソフトウェアと一緒に配布されるという条件で、Oracle がこのドキュメントを配布する方法と類似した方法で（すなわち、このソフトウェアを掲載する Web サイトから電子的にダウンロードする場合）、または CD-ROM や類似のメディアでドキュメントを配布する場合を除きます。出力コピーを配布するなど、ほかの形式で使用したり、別の出版物でこのドキュメントの全部または一部を使用したりする場合は、権限を有する Oracle の代表者から事前の書面による同意を得る必要があります。Oracle およびその関連会社は、上記で特に認められている場合を除き、このドキュメントに対するあらゆる権利を保有します。

ドキュメントのアクセシビリティについて

アクセシビリティに対する Oracle のコミットメントの詳細は、<https://www.oracle.com/corporate/accessibility/> の Oracle Accessibility Program の web サイトを参照してください。

Oracle Support へのアクセス

サポートを購入した Oracle のお客様は、My Oracle Support を介して電子サポートにアクセスできます。詳細は、<https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab> を参照してください。

第 1 章 一般情報

目次

1.1 このマニュアルについて	2
1.2 MySQL データベース管理システムの概要	4
1.2.1 MySQL とは	4
1.2.2 MySQL の主な機能	5
1.2.3 MySQL の歴史	8
1.3 MySQL 8.0 の新機能	8
1.4 MySQL 8.0 で追加、非推奨または削除されたサーバーおよびステータスの変数とオプション	44
1.5 MySQL の情報源	61
1.6 質問またはバグをレポートする方法	62
1.7 MySQL の標準への準拠	66
1.7.1 標準 SQL に対する MySQL 拡張機能	67
1.7.2 MySQL と標準 SQL との違い	69
1.7.3 MySQL における制約の処理	71
1.8 クレジット	73
1.8.1 MySQL への貢献者	73
1.8.2 ドキュメント作成者および翻訳者	77
1.8.3 MySQL をサポートするパッケージ	78
1.8.4 MySQL の作成に使用されたツール	79
1.8.5 MySQL のサポータ	79

MySQLソフトウェアは、マルチスレッドおよびマルチユーザー対応の非常に高速で堅固な SQL (構造化クエリー言語) データベースサーバーを提供します。MySQL Server は、ミッションクリティカルで負荷が高い運用システムにも、大規模に配備されるソフトウェアへの組み込みにも対応するように設計されています。Oracle および Java はオラクルおよびその関連会社の登録商標です。MySQL は Oracle Corporation およびその関連企業の商標であり、Oracle による書面による明示的な認可なしに使用できません。その他の社名、商品名等は各社の商標または登録商標である場合があります。

MySQL ソフトウェアはデュアルライセンス製品です。ユーザーは、GNU General Public License (<http://www.fsf.org/licenses/>) の条件で、MySQL ソフトウェアをオープンソース製品として使用するか、Oracle からの標準的な商用ライセンスを購入するかを選択できます。ライセンスポリシーの詳細は、<http://www.mysql.com/company/legal/licensing/>を参照してください。

このマニュアルで特に興味深いセクションは次のとおりです。

- MySQL Database Server の機能の説明については、[セクション1.2.2 「MySQL の主な機能」](#)を参照してください。
- MySQL の新しい機能の概要については、[セクション1.3 「MySQL 8.0 の新機能」](#)を参照してください。各バージョンの変更内容の詳細は、[リリースノート](#)を参照してください。
- インストールの手順については、[第2章 「MySQL のインストールとアップグレード」](#)を参照してください。MySQL のアップグレードの詳細は、[セクション2.11 「MySQL のアップグレード」](#)を参照してください。
- MySQL Database Server のチュートリアルについては、[第3章 「チュートリアル」](#)を参照してください。
- MySQL Server の構成と管理の詳細は、[第5章 「MySQL サーバーの管理」](#)を参照してください。
- MySQL でのセキュリティーの詳細は、[第6章 「セキュリティー」](#)を参照してください。
- レプリケーションサーバーの設定の詳細は、[第17章 「レプリケーション」](#)を参照してください。
- 高度な機能および管理ツールを備えた商用 MySQL リリースである MySQL Enterprise の詳細は、[第30章 「MySQL Enterprise Edition」](#)を参照してください。
- MySQL Database Server とその機能についてよくある質問に対する答えは、[付録A 「MySQL 8.0 のよくある質問」](#)を参照してください。
- 新しい機能とバグ修正の履歴については、[リリースノート](#)を参照してください。

重要

問題またはバグをレポートするには、[セクション1.6「質問またはバグをレポートする方法」](#)で説明する手順を使用してください。MySQL Server でセキュリティバグが見つかった場合は、[<secalert_us@oracle.com>](mailto:secalert_us@oracle.com) に電子メールメッセージを送信して、すぐにお知らせください。例外: サポートのお客様は、セキュリティのバグを含むすべての問題を Oracle Support までレポートしてください。

1.1 このマニュアルについて

これは、MySQL Database System バージョン 8.0 (リリース 8.0.29 まで) のリファレンスマニュアルです。MySQL 8.0 のマイナーバージョン間での相違点は、リリース番号 (8.0.x) に関連した現在のテキストに記されます。ライセンス情報については、[法的通知](#)を参照してください。

MySQL 8.0 と以前のバージョンとの間に機能などの多くの相違点があるため、このマニュアルは MySQL ソフトウェアの古いバージョンで用いることは想定されていません。MySQL ソフトウェアの以前のリリースを使用している場合は、該当するマニュアルを参照してください。たとえば、[MySQL 5.7 リファレンスマニュアル](#)には、5.7 シリーズの MySQL ソフトウェアリリースが取り上げられています。

これはリファレンスマニュアルであるため、SQL やリレーショナルデータベースの概念に関する一般的な説明は記載していません。また、使用しているオペレーティングシステムやコマンド行インタプリタの使用法も記載していません。

MySQL データベースソフトウェアは継続して開発が行われているため、リファレンスマニュアルも頻繁に更新されます。マニュアルの最新版は、オンラインで <https://dev.mysql.com/doc/> の検索可能なフォームから入手できます。ダウンロード可能な HTML バージョンや PDF バージョンなど、他の形式も使用できます。

MySQL 自体のソースコードには、Doxygen を使用して記述された内部ドキュメントが含まれています。生成された Doxygen コンテンツは、<https://dev.mysql.com/doc/index-other.html> で使用できます。[セクション2.9.10「MySQL Doxygen ドキュメントコンテンツの生成」](#)の手順を使用して、このコンテンツを MySQL ソース配布からローカルに生成することもできます。

MySQL の使用に関する質問がある場合は、[MySQL Community Slack](#) に参加するか、フォーラムで質問してください。[MySQL フォーラムにおける MySQL コミュニティサポート](#)を参照してください。マニュアル自体の追加または修正に関する提案がある場合は、<http://www.mysql.com/company/contact/> に送信してください。

表記規則および構文規則

このマニュアルは、次の表記規則に従って記載されています。

- このスタイルのテキストは、SQL ステートメント、データベース、テーブル、カラム名、プログラムリスト、ソースコード、環境変数に使用されます。例: 「付与テーブルをリロードするには、`FLUSH PRIVILEGES` ステートメントを使用します。」
- このスタイルのテキストは、例の中で入力する文字の例を示します。
- このスタイルのテキストは、`mysql` (MySQL コマンド行のクライアントプログラム) および `mysqld` (MySQL Server 実行可能ファイル) である実行可能プログラムおよびスクリプトの名前を示します。
- このスタイルのテキストは、独自に選択する値に置き換える変数入力に使用されます。
- このスタイルのテキストは、強調に使用されます。
- このスタイルのテキストは、表の見出しや特に重要なことを示すときに使用されます。
- このスタイルのテキストは、プログラムの実行方法に影響を与えるか、またはプログラムが特定の方法で機能するために必要な情報を提供するプログラムオプションを示すために使用されます。例: 「`--host` オプション (短縮形 `-h`) は、`mysql` クライアントプログラムに対して接続すべき MySQL Server のホスト名または IP アドレスを指示します」。
- ファイル名とディレクトリ名は次のように表示されます。「グローバル `my.cnf` ファイルは、`/etc` ディレクトリにあります。」

- 文字シーケンスは次のように表示されます。「ワイルドカードを指定するには、『%』文字を使用します。」

特定のプログラム内から実行されるコマンドが示される場合、そのコマンドの前に記されるプロンプトは、どのコマンドが使用されるかを示しています。たとえば、`shell>` はログインシェルから実行するコマンドを示し、`root-shell>` は同様ですが `root` として実行する必要がある、`mysql>` は `mysql` クライアントプログラムから実行するステートメントを示します。

```
shell> type a shell command here
root-shell> type a shell command as root here
mysql> type a mysql statement here
```

領域によっては、コマンドを 2 つの異なる環境で実行すべきであることを示すために、異なるシステムが区別される場合があります。たとえば、レプリケーションの操作中に、コマンドの先頭に `source` および `replica` を付けることができます:

```
source> type a mysql command on the replication source here
replica> type a mysql command on the replica here
```

「shell」は、コマンドインタプリタです。UNIX の場合、これは通常、`sh`、`cs`、`bash` などのプログラムです。Windows の場合、同等のプログラムは `command.com` または `cmd.exe` で、通常コンソールウィンドウで実行します。

例に示されるコマンドまたはステートメントを入力する場合、その例に示されるプロンプトを入力する必要はありません。

データベース名、テーブル名、およびカラム名は多くの場合、ステートメントに代入する必要があります。このような代入が必要であることを示す場合、このマニュアルでは `db_name`、`tbl_name`、および `col_name` を使用します。たとえば、次のようなステートメントがあるとします。

```
mysql> SELECT col_name FROM db_name.tbl_name;
```

これは、同様のステートメントを入力する場合、データベース名、テーブル名、およびカラム名を自分で指定することを意味します。たとえば、次のようになります。

```
mysql> SELECT author_name FROM biblio_db.author_list;
```

SQL キーワードでは大文字と小文字は区別されず、大文字と小文字は区別されません。このマニュアルでは大文字を使用します。

構文の説明では、角かっこ (「`[`」 および 「`]`」) はオプションの語または句を示します。たとえば、次のステートメントでは、`IF EXISTS` はオプションです。

```
DROP TABLE [IF EXISTS] tbl_name
```

構文要素が複数の選択肢で構成される場合、選択肢は縦棒 (「`|`」) で区切られます。一連の選択肢から 1 つのメンバーを選択できる場合、選択肢は角かっこ (「`[`」 および 「`]`」) 内にリストされます。

```
TRIM([BOTH | LEADING | TRAILING] [remstr] FROM] str)
```

一連の選択肢から 1 つのメンバーを選択しなければならない場合、選択肢は中かっこ (「`{`」 および 「`}`」) 内にリストされます。

```
{DESCRIBE | DESC} tbl_name [col_name | wild]
```

省略記号 (...) は、ステートメントの一部が省略されていることを示し、通常、複雑な構文を簡単に記しています。たとえば、`SELECT ... INTO OUTFILE` は、ステートメントのほかの部分に `INTO OUTFILE` 句が続く `SELECT` ステートメントを省略したものです。

省略記号は、ステートメントで前にある構文要素を繰り返すことができる場合にも使用されます。次の例では、`reset_option` 値を、最初のもの以外はそれぞれカンマを前に付けて、複数回繰り返すことができます。

```
RESET reset_option [,reset_option] ...
```

シェル変数を設定するコマンドは、Bourne シェル構文を使用して示されます。たとえば、`CC` 環境変数を設定し、`configure` コマンドを実行するシーケンスは、Bourne シェル構文では次のように示されます。

```
shell> CC=gcc ./configure
```

`cs` または `tcsh` を使用する場合、多少異なるコマンドを入力する必要があります。

```
shell> setenv CC gcc
shell> ./configure
```

マニュアル執筆

リファレンスマニュアルのソースファイルは、DocBook XML 形式で記述されています。HTML バージョンとその他の形式は、主に DocBook XSL スタイルシートを使用して自動的に作成されます。DocBook の詳細は、<http://docbook.org/>を参照してください。

このマニュアルは当初、David Axmark と Michael 「Monty」 Widenius によって執筆されました。引き続き、Chris Cole、Paul DuBois、Margaret Fisher、Edward Gilmore、Stefan Hinz、David Moss、Philip Olson、Daniel Price、Daniel So および Jon Stephens で構成される MySQL ドキュメントチームによってメンテナンスされています。

1.2 MySQL データベース管理システムの概要

1.2.1 MySQL とは

MySQL は、もっとも普及しているオープンソース SQL データベース管理システムで、オラクル社により開発、流通、およびサポートが行われています。

MySQL web サイト (<http://www.mysql.com/>) には、MySQL ソフトウェアに関する最新情報が記載されています。

- MySQL はデータベース管理システムです。

データベースとは、構造化されたデータの集合です。簡単なショッピングリストから絵画のギャラリー、または社内ネットワークの膨大な量の情報など、さまざまなものがあります。コンピュータデータベースに格納されているデータに対して追加、アクセス、および処理などを行うには、MySQL Server のようなデータベース管理システムが必要となります。コンピュータは大量のデータの処理に非常に優れているので、データベース管理システムは、スタンドアロンのユーティリティーまたはほかのアプリケーションの一部として、データ処理の中心的な役割を果たします。

- MySQL データベースはリレーショナルデータベースです。

リレーショナルデータベースは、すべてのデータを 1 つの大きな保管場所に置くのではなく、独立したテーブルに保存します。データベースの構造は、速度を最適化した物理ファイルにわかれています。データベース、テーブル、ビュー、行、およびカラムなどのオブジェクトをもつ論理モデルは、柔軟なプログラミング環境を提供します。さまざまなデータフィールドの間の関係を統治する、1 対 1、1 対多、一意、必須またはオプションなどのルールと、さまざまなテーブル間の「ポインタ」を設定します。適切に設計されたデータベースを使用すれば、アプリケーションにはデータの矛盾、重複、孤立、期限切れ、または欠損が決して現れないように、データベースはこれらのルールを実施します。

「MySQL」の、SQL の部分は「Structured Query Language」（構造化クエリー言語）を表しています。SQL は、データベースにアクセスするために使用されるもっとも一般的な標準化言語です。プログラミング環境によって、SQL を直接入力（たとえばレポートの作成）、別の言語で作成されたコードに SQL ステートメントを埋め込み、または SQL 構文を隠す言語固有の API を使用する場合があります。

SQL は ANSI/ISO SQL 標準で定義されます。SQL 標準は 1986 年以降開発が繰り返され、複数のバージョンがあります。本マニュアルでは、「SQL-92」は 1992 年にリリースされた標準に、「SQL:1999」は 1999 年にリリースされた標準に、そして「SQL:2003」は標準の現バージョンに対応しています。「SQL 標準」という用語は、任意の時点における現行バージョンの SQL 標準を表す場合に使用します。

- MySQL ソフトウェアはオープンソースです。

オープンソースとは、そのソフトウェアをだれでも使用および修正できることを意味します。MySQL ソフトウェアは、だれもが無料でインターネットからダウンロードし、使用することができます。必要に応じて、ソースコードを調べ、ニーズに合わせて変更することができます。MySQL ソフトウェアは GPL (GNU General Public License)、<http://www.fsf.org/licenses/>、を使用して、さまざまな状況で実行するものとしなないものを定義します。GPL では不都合な場合や商用アプリケーションに MySQL コードを組み込む必要がある場合は、商用ライセンス版を購入することができます。詳細は、『MySQL ライセンス概要』(<http://www.mysql.com/company/legal/licensing/>)を参照してください。

- MySQL Database Server は非常に高速で信頼性が高く、スケーラブルで使いやすいです。

それを求めているのであれば、ぜひお試しください。MySQL Server は、デスクトップまたはラップトップ上で、ほかのアプリケーションや Web サーバーなどと併用して、ほとんどまたはまったく処理を要することなく快適に実行できます。マシン全体を MySQL 専用にする場合、使用可能なすべてのメモリー、CPU パワー、および I/O 能力を利用するように設定を調整できます。MySQL は、ネットワークで接続されたマシンのクラスターにスケールアップすることもできます。

MySQL Server は当初、既存のソリューションよりもはるかに速く大規模なデータベースを処理することを目的として開発され、多くを要求される運用環境で数年間にわたって使用されています。引き続き開発が行われていますが、MySQL Server は現在、便利で多彩な関数を備えています。その接続性、速度、安全性によって、MySQL Server はインターネット上のデータベースへのアクセスに非常に適しています。

- MySQL Server は、クライアント/サーバー組み込みシステムで機能します。

MySQL Database Software は、様々なバックエンド、いくつかの異なるクライアントプログラムとライブラリ、管理ツールおよび幅広いアプリケーションプログラミングインタフェース (API) をサポートするマルチスレッド SQL サーバーで構成されるクライアント/サーバーシステムです。

また、MySQL Server は、アプリケーションにリンクできる埋込みマルチスレッドライブラリとしても提供されており、スタンドアロン製品をより小さく、迅速かつ容易に管理できます。

- 大量の MySQL ソフトウェアが提供されており、使用可能です。

MySQL Server には、ユーザーと緊密に協力して開発された、実際的な一連の機能があります。必要なアプリケーションや言語ですでに MySQL Database Server がサポートされていると思われます。

「MySQL」の正式な読み方は、「マイエスキューエル」(「マイシークエル」ではありません) ですが、「マイシークエル」やその他のローカライズされた方法で発音してもかまいません。

1.2.2 MySQL の主な機能

このセクションでは MySQL Database Software の重要な特徴の一部を説明します。ほとんどの場合、ロードマップはすべてのバージョンの MySQL に適用されます。シリーズごとに新しく導入される MySQL の機能については、対応するマニュアルの「新機能」セクションを参照してください。

- MySQL 8.0: [What Is New in MySQL 8.0](#)
- MySQL 5.7: [What Is New in MySQL 5.7](#)
- MySQL 5.6: [What Is New in MySQL 5.6](#)

内部および移植性

- C および C++ で記述されています。
- さまざまなコンパイラでテストされています。
- さまざまなプラットフォームで動作します。 <https://www.mysql.com/support/supportedplatforms/database.html> を参照してください。
- 移植性のために、CMake を使用して構成します。
- Purify (商用メモリーリーク検出システム) と GPL ツールの Valgrind (<http://developer.kde.org/~sewardj/>) でテストされています。
- 独立モジュールを備えた多層サーバー設計を使用しています。
- カーネルスレッドを使用してマルチスレッド化されるように設計されており、複数の CPU(使用可能な場合) を簡単に使用できます。
- トランザクションストレージエンジンと非トランザクションストレージエンジンを備えています。
- インデックス圧縮を備えた非常に高速な B-tree ディスクテーブル(MyISAM) を使用しています。

- 別のストレージエンジンの追加が比較的容易になるよう設計されています。これは、社内データベースへの SQL インタフェースを追加する場合に便利です。
- スレッドベースの非常に高速なメモリー割り当てシステムを使用しています。
- 最適化されたネストループ結合を使用して非常に高速な結合を実行します。
- インメモリーハッシュテーブルを実装し、一時テーブルとして使用します。
- 高度に最適化されたクラスライブラリを使用して SQL 関数が実装されるため、最大限の速度が確保されます。通常は、クエリーの初期化後にメモリー割り当てが行われることはありません。
- クライアント/サーバーネットワーク環境で使用するために、サーバーを独立したプログラムとして提供しています。単独のアプリケーションに組み込み (リンク) できるライブラリとしても提供されています。このようなアプリケーションは単一で、あるいはネットワーク環境の整っていない場所でも使用することができます。

データ型

- 多数のデータタイプ: 1、2、3、4、および 8 バイト長の符号付き/符号なし整数、[FLOAT](#)、[DOUBLE](#)、[CHAR](#)、[VARCHAR](#)、[BINARY](#)、[VARBINARY](#)、[TEXT](#)、[BLOB](#)、[DATE](#)、[TIME](#)、[DATETIME](#)、[TIMESTAMP](#)、[YEAR](#)、[SET](#)、[ENUM](#)、および [OpenGIS 空間型](#)。第 11 章「[データ型](#)」を参照してください。
- 固定長および可変長の文字列型。

ステートメントと関数

- クエリーの [SELECT](#) 句および [WHERE](#) 句での演算子と関数の完全なサポート。例:

```
mysql> SELECT CONCAT(first_name, ' ', last_name)
-> FROM citizen
-> WHERE income/dependents > 10000 AND age > 30;
```

- SQL の [GROUP BY](#) 句および [ORDER BY](#) 句の完全なサポート。グループ関数 ([COUNT\(\)](#)、[AVG\(\)](#)、[STD\(\)](#)、[SUM\(\)](#)、[MAX\(\)](#)、[MIN\(\)](#)、および [GROUP_CONCAT\(\)](#)) のサポート。
- 標準の SQL 構文および ODBC 構文での [LEFT OUTER JOIN](#) および [RIGHT OUTER JOIN](#) のサポート。
- 標準 SQL で必要な、テーブルおよびカラムにおけるエイリアスのサポート。
- 変更された (影響を受けた) 行の数を返す [DELETE](#)、[INSERT](#)、[REPLACE](#)、および [UPDATE](#) のサポート。サーバーに接続する際にフラグを設定することで、代わりに一致したレコードの数を返すことも可能です。
- データベース、ストレージエンジン、テーブル、およびインデックスに関する情報を取得する、MySQL 固有の [SHOW](#) ステートメントのサポート。標準 SQL に従って実装された [INFORMATION_SCHEMA](#) データベースのサポート。
- オプティマイザによるクエリーの解決方法を表示する [EXPLAIN](#) ステートメント。
- 関数名の、テーブル名やカラム名との独立性。たとえば、[ABS](#) は有効なカラム名です。唯一の制限事項は、関数呼び出しで、関数名とその後に続く「(」との間にスペースを使用できないことです。[セクション 9.3「キーワードと予約語」](#)を参照してください。
- 同じステートメント内で、さまざまなデータベースのテーブルを参照することができます。

セキュリティ

- 非常に柔軟でセキュアな権限およびパスワードシステム。ホストベースの検証が可能です。
- サーバーに接続する際にすべてのパスワードトラフィックが暗号化されるので、パスワードは安全です。

拡張性と制限

- 大規模なデータベースのサポート。当社は、MySQL Server を使用して 50,000,000 レコードが格納されたデータベースを処理しています。また、MySQL Server を使用して 200,000 テーブル、約 5,000,000,000 行を処理しているユーザーもいます。

- テーブルあたり最大 64 個のインデックスをサポートします。各インデックスは、1 から 16 個のカラムまたはカラムの一部で構成されます。InnoDB テーブルの最大インデックス幅は 767 バイトまたは 3072 バイトです。セクション15.22「InnoDB の制限」を参照してください。MyISAM テーブルの最大インデックス幅は 1000 バイトです。セクション16.2「MyISAM ストレージエンジン」を参照してください。インデックスでは、CHAR、VARCHAR、BLOB、あるいは TEXT 型のカラムのプリフィクスを使用することができます。

接続性

- クライアントは複数のプロトコルを使用して MySQL Server に接続できます。
 - クライアントは、あらゆるプラットフォームで TCP/IP ソケットを使用して接続することができます。
 - Windows システムでは、named_pipe システム変数を有効にしてサーバーを起動すると、クライアントは名前付きパイプを使用して接続できます。Windows サーバーでは、shared_memory システム変数を有効にして起動した場合、共有メモリ接続もサポートされます。クライアントは --protocol=memory オプションを使用して共有メモリで接続できます。
 - Unix システムでは、クライアントは Unix ドメインソケットファイルを使用して接続することができます。
- MySQL クライアントプログラムはさまざまな言語で記述できます。C 言語で記述されたクライアントライブラリは C、C++、あるいは C バインディングを提供する任意の言語で記述されたクライアントでも使用可能です。
- C、C++、Eiffel、Java、Perl、PHP、Python、Ruby、および Tcl 用の API が提供されており、MySQL クライアントを多くの言語で記述できます。第29章「Connector および API」を参照してください。
- Connector/ODBC (MyODBC) インタフェースによって、ODBC (Open DataBase Connectivity) 接続を使用するクライアントプログラムに MySQL サポートが提供されます。たとえば、MS Access を使用して MySQL Server に接続することができます。クライアントは、Windows と Unix のどちらで実行されていてもかまいません。Connector/ODBC ソースが使用可能です。ほかの多くの機能と同様に、ODBC 2.5 のすべての機能がサポートされます。「MySQL Connector/ODBC Developer Guide」を参照してください。
- Connector/J インタフェースは JDBC 接続を使用する Java クライアントプログラムの MySQL サポートを提供しています。クライアントは、Windows と Unix のどちらで実行されていてもかまいません。Connector/J ソースが使用可能です。MySQL Connector/J 5.1 Developer Guide を参照してください。
- MySQL Connector/NET を使用すると、開発者は MySQL との安全で高パフォーマンスなデータ接続を必要とする .NET アプリケーションを簡単に作成できます。必要な ADO.NET インタフェースを実装し、ADO.NET 対応のツールに統合します。開発者は、選択した .NET 言語を使用してアプリケーションを構築できます。MySQL Connector/NET は、100% Pure C#で記述された完全管理 ADO.NET ドライバです。「MySQL Connector/NET Developer Guide」を参照してください。

ローカライズ

- サーバーは、クライアントに多数の言語でエラーメッセージを送信することができます。セクション10.12「エラーメッセージ言語の設定」を参照してください。
- latin1 (cp1252)、german、big5、ujis、複数の Unicode 文字セットなど、複数の異なる文字セットの完全なサポート。たとえば、スカンジナビア語の文字「å」、「ä」、および「ö」をテーブル名やカラム名で使用できます。
- すべてのデータが、選択した文字セットで保存されます。
- ソートおよび比較は、デフォルトの文字セットおよび照合順序に従って行われます。これは、MySQL サーバーの起動時に変更できます(セクション10.3.2「サーバー文字セットおよび照合順序」を参照)。非常に高度なソートの例については、チェコ語のソートコードを参照してください。MySQL Server ではさまざまな文字セットがサポートされており、コンパイル時および実行時に指定することができます。
- サーバーのタイムゾーンは動的に変更でき、個々のクライアントは独自のタイムゾーンを指定できます。セクション5.1.15「MySQL Server でのタイムゾーンのサポート」を参照してください。

クライアントとツール

- MySQL には複数のクライアントとユーティリティプログラムが含まれます。これには、mysqldump および mysqladmin といったコマンド行プログラム、そして MySQL Workbench などのグラフィックプログラムも含まれます。

- MySQL Server には、テーブルのチェック、最適化、および修復を行う SQL ステートメントのサポートが組み込まれています。これらのステートメントは、`mysqlcheck` クライアントを介してコマンド行から使用可能です。また、MySQL には、`MyISAM` テーブルでこれらの操作を実行するための `myisamchk` という非常に高速なコマンド行ユーティリティが組み込まれています。第4章「MySQL プログラム」を参照してください。
- MySQL プログラムを `--help` または `-?` オプションを指定して呼び出すと、オンラインヘルプを参照できます。

1.2.3 MySQL の歴史

当初は、高速で低レベルな独自の (ISAM) ルーチンを使用してテーブルに接続するために `mSQL` データベースシステムを使用するつもりでした。しかし、いくつかのテストを行なった結果、`mSQL` はそれほど高速でも柔軟でもないため、ニーズに合わないという結論に至りました。これが要因となって、私たちのデータベースへの新しい SQL インタフェースを開発することになりました。ただし、`mSQL` とほとんど同じ API インタフェースを使用することにしました。この API は、`mSQL` で使用するために記述されたサードパーティーのコードを MySQL で使用するために簡単に移植できるように設計されています。

MySQL は、共同創設者 Monty Widenius の娘の My にちなんで名づけられました。

MySQL のドルフィン (当社のロゴ) の名前は「Sakila」です。「ドルフィンのネーミング」コンテストで、ユーザーによって提案された膨大な数の名前の中から選ばれました。受賞者の名前は、アフリカのエスワティニ (旧称スワジランド) のオープンソースソフトウェア開発者 Ambrose Twebaze によって提出されました。アンブローズによると、Sakila というフェミニン名は、エスワティニ語のローカル言語である SiSwati のルーツを持っています。また、Sakila は、Ambrose の出生国であるウガンダに近い、タンザニアのアルーシャにある町の名前でもあります。

1.3 MySQL 8.0 の新機能

このセクションでは、MySQL 8.0 で追加された機能、非推奨になった機能、および削除された機能について要約しています。コンパニオンセクションには、MySQL 8.0 で追加、非推奨または削除された MySQL サーバーのオプションおよび変数がリストされます。セクション1.4「MySQL 8.0 で追加、非推奨または削除されたサーバーおよびステータスの変数とオプション」を参照してください。

- [MySQL 8.0 で追加された機能](#)
- [MySQL 8.0 で非推奨となった機能](#)
- [MySQL 8.0 で削除された機能](#)

MySQL 8.0 で追加された機能

次の機能が、MySQL 8.0 に追加されました。

- **データディクショナリ。** MySQL には、データベースオブジェクトに関する情報を格納するトランザクションデータディクショナリが組み込まれています。以前の MySQL リリースでは、ディクショナリデータはメタデータファイルおよび非トランザクションテーブルに格納されていました。詳細は、第14章「MySQL データディクショナリ」を参照してください。
- **アトミックデータ定義ステートメント (アトミック DDL)。** アトミック DDL ステートメントは、DDL 操作に関連付けられたデータディクショナリ更新、ストレージエンジン操作およびバイナリログ書き込みを組み合わせ、単一のアトミックトランザクションにします。詳細は、セクション13.1.1「アトミックデータ定義ステートメントのサポート」を参照してください。
- **アップグレード手順。** 以前は、新しいバージョンの MySQL をインストールした後、次の起動時に MySQL サーバーは自動的にデータディクショナリテーブルをアップグレードしていましたが、その後、DBA が手動で `mysql_upgrade` を起動して、`mysql` スキーマ内のシステムテーブルおよび `sys` スキーマやユーザースキーマなどの他のスキーマ内のオブジェクトをアップグレードする必要がありました。

MySQL 8.0.16 では、サーバーは以前 `mysql_upgrade` によって処理されていたタスクを実行します。新しい MySQL バージョンのインストール後、サーバーは次の起動時に必要なすべてのアップグレードタスクを自動的に実行するようになり、DBA が `mysql_upgrade` を呼び出す必要はなくなりました。また、サーバーはヘルプテーブルの内容を更新します (`mysql_upgrade` では更新されませんでした)。新しい `--upgrade` サーバーオプションは、サーバーがデータディクショナリおよびサーバーの自動アップグレード操作を実行する方法を制御します。詳細は、セクション2.11.3「MySQL のアップグレードプロセスの内容」を参照してください。

- セキュリティとアカウント管理。セキュリティを向上させ、アカウント管理における DBA の柔軟性を高めるために、次の拡張機能が追加されました:
 - `mysql` システムデータベース内の付与テーブルが `InnoDB` (トランザクション) テーブルになりました。以前は、これらは `MyISAM` (非トランザクション) テーブルでした。付与テーブルのストレージエンジン変更により、アカウント管理ステートメントの動作が変わります。以前は、複数のユーザーを指定したアカウント管理ステートメント (`CREATE USER` や `DROP USER` など) は、一部のユーザーに対して成功し他のユーザーに対して失敗することがありました。現在は、各ステートメントはトランザクションに対応し、指定されたすべてのユーザーに対して成功するか、エラーが発生した場合はロールバックされて何の影響も与えなくなりました。ステートメントが成功した場合はバイナリログに書き込まれますが、失敗した場合は書き込まれず、ロールバックが発生して変更は行われません。詳細は、[セクション13.1.1「アトミックデータ定義ステートメントのサポート」](#)を参照してください。
 - 新しい `caching_sha2_password` 認証プラグインが使用可能です。 `sha256_password` プラグインと同様に、 `caching_sha2_password` は SHA-256 パスワードハッシングを実装しますが、接続時の待機時間の問題に対処するためにキャッシュを使用します。ほかにも多くのトランスポートプロトコルをサポートしており、RSA キーペアベースのパスワード交換機能のために OpenSSL とのリンクを必要としません。 [セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#)を参照してください。

`caching_sha2_password` および `sha256_password` 認証プラグインは、 `mysql_native_password` プラグインよりもセキュアなパスワード暗号化を提供し、 `caching_sha2_password` は `sha256_password` よりも優れたパフォーマンスを提供します。 `caching_sha2_password` のこれらの優れたセキュリティおよびパフォーマンス特性のため、現在は優先認証プラグインであり、 `mysql_native_password` ではなくデフォルトの認証プラグインでもあります。サーバー操作のためのデフォルトのプラグインのこの変更による影響、およびサーバーとクライアントおよびコネクタとの互換性については、 [優先認証プラグインとしての `caching_sha2_password`](#) を参照してください。
 - MySQL では、権限の名前付きコレクションであるロールがサポートされるようになりました。ロールは作成および削除できます。ロールには、付与された権限およびロールから取り消された権限を付与できます。ロールは、ユーザーアカウントに対して付与または取り消すことができます。アカウントに適用可能なアクティブなロールは、アカウントに付与されているロールの中から選択でき、そのアカウントのセッション中に変更できます。詳細は、 [セクション6.2.10「ロールの使用」](#)を参照してください。
 - MySQL には、ユーザーアカウントカテゴリーの概念が組み込まれ、システムユーザーと通常のユーザーは `SYSTEM_USER` 権限を持っているかどうかによって区別されるようになりました。 [セクション6.2.11「アカウントカテゴリー」](#)を参照してください。
 - 以前は、特定のスキーマを除き、グローバルに適用される権限を付与できませんでした。これは、 `partial_revokes` システム変数が有効な場合に可能になりました。 [セクション6.2.12「部分取消しを使用した権限の制限」](#)を参照してください。
 - `GRANT` ステートメントには、ステートメントの実行に使用する権限コンテキストに関する追加情報を指定する `AS user [WITH ROLE]` 句があります。この構文は SQL レベルで表示できますが、主な目的は、部分的な取消しによって課される権限付与者権限制限のすべてのノード間で均一なレプリケーションを有効にすることです。これにより、これらの制限がバイナリログに表示されます。 [セクション13.7.1.6「GRANT ステートメント」](#)を参照してください。
 - MySQL では、パスワード履歴に関する情報が保持されるようになり、以前のパスワードの再利用に関する制限が有効になりました。DBA は、一定の変更回数または一定期間にわたって、以前使ったパスワードの再指定ができ

ないように制限することができます。アカウント単位およびグローバルにパスワード再利用ポリシーを設定することができます。

アカウントのパスワードを変更する場合、現在のパスワードの入力を要求できるようになりました。これにより、DBA は、現在のパスワードを知らないユーザーがパスワードを変更するのを阻止できます。パスワード検証ポリシーは、グローバルにもアカウントごとにも設定できます。

アカウントはデュアルパスワードを持つことができるようになりました。これにより、停止時間なしで複雑な複数サーバーシステムで段階的なパスワード変更をシームレスに実行できます。

MySQL では、管理者がユーザーアカウントを構成できるようになり、パスワードの誤りによる連続したログイン失敗が多すぎると一時アカウントがロックされるようになりました。必要な失敗数とロック時間は、アカウントごとに構成できます。

これらの新機能により、DBA はパスワード管理をより完全に制御できます。詳細は、[セクション6.2.15「パスワード管理」](#)を参照してください。

- OpenSSL を使用してコンパイルされた場合、MySQL で FIPS モードがサポートされるようになり、OpenSSL ライブラリおよび FIPS オブジェクトモジュールを実行時に使用できるようになりました。FIPS モードでは、許容される暗号化アルゴリズムの制限や長いキー長の要件などの暗号化操作に条件が適用されます。[セクション6.8「FIPS のサポート」](#)を参照してください。
- サーバーが新しい接続に使用する TLS コンテキストは、実行時に再構成可能になりました。この機能は、SSL 証明書が期限切れになるまで実行されている MySQL サーバーを再起動しないようにする場合などに役立ちます。[サーバー側のランタイム構成および暗号化された接続の監視](#)を参照してください。
- OpenSSL 1.1.1 では、暗号化された接続用に TLS v1.3 プロトコルがサポートされ、MySQL 8.0.16 以上では、サーバーとクライアントの両方が OpenSSL 1.1.1 以上を使用してコンパイルされている場合に TLS v1.3 もサポートされます。[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#)を参照してください。
- MySQL は、名前付きパイプ上のクライアントに付与されるアクセス制御を、Windows での正常な通信に必要な最小値に設定するようになりました。新しい MySQL クライアントソフトウェアは、追加構成なしで名前付きパイプ接続を開くことができます。古いクライアントソフトウェアをすぐにアップグレードできない場合は、新しい `named_pipe_full_access_group` システム変数を使用して、名前付きパイプ接続を開くために必要な権限を Windows グループに付与できます。フルアクセスグループのメンバーシップは制限され、一時的である必要があります。
- リソース管理。 MySQL では、リソースグループの作成および管理がサポートされるようになり、グループで使用可能なリソースに従ってスレッドが実行されるように、サーバー内で実行されているスレッドを特定のグループに割り当てることができます。グループ属性を使用すると、そのリソースを制御して、グループ内のスレッドによるリソース消費を有効にしたり制限したりすることができます。DBA は、様々なワークロードに応じてこれらの属性を変更できます。>現在、CPU 時間は管理可能なリソースであり、CPU コア、ハイパースレッド、ハードウェアスレッドなどを含む用語として「仮想 CPU」の概念で表現されています。サーバーは起動時に使用可能な仮想 CPU の数を決定し、適切な権限を持つデータベース管理者はこれらの CPU をリソースグループに関連付け、スレッドをグループに割り当てることができます。詳細は、[セクション5.1.16「リソースグループ」](#)を参照してください。
- テーブルの暗号化管理。 暗号化のデフォルトを定義して強制することで、テーブルの暗号化をグローバルに管理できるようになりました。`default_table_encryption` 変数は、新しく作成されたスキーマおよび一般テーブルスペースの暗号化デフォルトを定義します。スキーマの暗号化のデフォルトは、スキーマの作成時に `DEFAULT ENCRYPTION` 句を使用して定義することもできます。デフォルトでは、テーブルは作成されたスキーマまたは一般テーブルスペースの暗号化を継承します。暗号化のデフォルトは、`table_encryption_privilege_check` 変数を有効にすることで適用されます。権限チェックは、`default_table_encryption` 設定とは異なる暗号化設定を使用してスキーマまたは一般テーブルスペースを作成または変更する場合、またはデフォルトのスキーマ暗号化とは異なる暗号化設定を使用してテーブルを作成または変更する場合に発生します。`TABLE_ENCRYPTION_ADMIN` 権限では、`table_encryption_privilege_check` が有効な場合にデフォルトの暗号化設定をオーバーライドできます。詳細は、[スキーマおよび一般テーブルスペースの暗号化デフォルトの定義](#)を参照してください。

- InnoDB の拡張機能。 次の InnoDB の拡張機能が追加されました。
- 現在の最大自動インクリメントカウンタ値は、値が変更されるたびに redo ログに書き込まれ、チェックポイントごとにエンジン固有のシステムテーブルに保存されます。これらの変更により、現在の最大自動インクリメントカウンタ値がサーバーの再起動後も保持されます。さらに、次のようになります:
- サーバーを再起動しても、`AUTO_INCREMENT = N` テーブルオプションの効果は取り消されなくなりました。自動インクリメントカウンタを特定の値に初期化した場合、または自動インクリメントカウンタ値を大きな値に変更した場合、新しい値はサーバーの再起動後も保持されます。
- `ROLLBACK` 操作の直後にサーバーを再起動しても、ロールバックされたトランザクションに割り当てられた自動増分値は再利用されなくなりました。
- `AUTO_INCREMENT` カラムの値を現在の最大自動増分値より大きい値 (`UPDATE` 操作など) に変更すると、新しい値が永続化され、後続の `INSERT` 操作では新しい大きい値から始まる自動増分値が割り当てられます。

詳細は、[セクション15.6.1.6「InnoDB での AUTO_INCREMENT 処理」](#) および [InnoDB AUTO_INCREMENT カウンタの初期化](#) を参照してください。

- インデックスツリーの破損が発生すると、InnoDB は破損フラグを redo ログに書き込み、破損フラグを安全にクラッシュさせます。また、InnoDB は、インメモリー破損フラグデータを各チェックポイントのエンジン専用システムテーブルに書き込みます。リカバリ中、InnoDB は、インメモリーテーブルおよびインデックスオブジェクトを破損としてマークする前に、両方の場所から破損フラグを読み取り、結果をマージします。
- InnoDB memcached プラグインは、複数の `get` 操作 (単一の memcached クエリーで複数のキーと値のペアをフェッチ) および範囲クエリーをサポートしています。 [セクション15.20.4「InnoDB memcached の複数の get および Range クエリーのサポート」](#) を参照してください。
- デッドロック検出を無効にするには、新しい動的変数 `innodb_deadlock_detect` を使用できます。同時実行性の高いシステムでは、多数のスレッドが同じロックを待機している場合、デッドロック検出によって速度が低下する可能性があります。デッドロック検出を無効にし、デッドロック発生時のトランザクションロールバックの `innodb_lock_wait_timeout` 設定に依存する方が効率的な場合があります。
- 新しい `INFORMATION_SCHEMA.INNODB_CACHED_INDEXES` テーブルには、インデックスごとに InnoDB バッファプールにキャッシュされたインデックスページの数レポートされます。
- InnoDB 一時テーブルが共有一時テーブルスペースである `ibtmp1` に作成されます。
- InnoDB [tablespace encryption feature](#) では、redo ログおよび undo ログデータの暗号化がサポートされています。 [redo ログの暗号化](#) および [undo ログの暗号化](#) を参照してください。
- InnoDB は、`SELECT ... FOR SHARE` および `SELECT ... FOR UPDATE` のロック読取りステートメントで `NOWAIT` および `SKIP LOCKED` オプションをサポートしています。 `NOWAIT` では、リクエストされた行が別のトランザクションによってロックされている場合、ステートメントはただちに戻されます。 `SKIP LOCKED` では、ロックされた行が結果セットから削除されます。 [NOWAIT および SKIP LOCKED による読取り同時実行性のロック](#) を参照してください。

`SELECT ... FOR SHARE` は `SELECT ... LOCK IN SHARE MODE` に置き換わりませんが、`LOCK IN SHARE MODE` は下位互換性のために引き続き使用できます。ステートメントは同等です。ただし、`FOR UPDATE` および `FOR SHARE` は `NOWAIT`、`SKIP LOCKED` および `OF tbl_name` オプションをサポートしています。 [セクション13.2.10「SELECT ステートメント」](#) を参照してください。

`OF tbl_name` では、名前付きテーブルにロッククエリーが適用されます。

- `ADD PARTITION`、`DROP PARTITION`、`COALESCE PARTITION`、`REORGANIZE PARTITION` および `REBUILD PARTITION ALTER TABLE` オプションは、ネイティブパーティション化インプレース API でサポートされており、`ALGORITHM={COPY|INPLACE}` および `LOCK` 句とともに使用できます。

`ALGORITHM=INPLACE` を使用した `DROP PARTITION` は、パーティションに格納されているデータを削除し、パーティションを削除します。ただし、`ALGORITHM=COPY` または `old_alter_table=ON` を使用した `DROP PARTITION` では、パーティションテーブルが再構築され、削除されたパーティションから互換性のある

PARTITION ... VALUES 定義を持つ別のパーティションへのデータの移動が試行されます。別のパーティションに移動できないデータは削除されます。

- **InnoDB** ストレージエンジンは、独自のストレージエンジン固有のデータディクショナリではなく、MySQL データディクショナリを使用するようになりました。データディクショナリの詳細は、[第14章「MySQL データディクショナリ」](#)を参照してください。
- **mysql** システムテーブルおよびデータディクショナリテーブルは、MySQL データディレクトリの **mysql.ibd** という名前の単一の **InnoDB** テーブルスペースファイルに作成されるようになりました。以前は、これらのテーブルは **mysql** データベースディレクトリ内の個々の **InnoDB** テーブルスペースファイルに作成されていました。
- MySQL 8.0 では、次の **undo** テーブルスペースの変更が導入されています:
 - デフォルトでは、**undo** ログは、MySQL インスタンスの初期化時に作成される 2 つの **undo** テーブルスペースに存在するようになりました。**undo** ログはシステムテーブルスペースに作成されなくなりました。
 - MySQL 8.0.14 では、**CREATE UNDO TABLESPACE** 構文を使用して、実行時に選択した場所に追加の **undo** テーブルスペースを作成できます。

```
CREATE UNDO TABLESPACE tablespace_name ADD DATAFILE 'file_name.ibu';
```

CREATE UNDO TABLESPACE 構文を使用して作成された **undo** テーブルスペースは、**DROP UNDO TABLESPACE** 構文を使用して実行時に削除できます。

```
DROP UNDO TABLESPACE tablespace_name;
```

ALTER UNDO TABLESPACE 構文を使用すると、**undo** テーブルスペースをアクティブまたは非アクティブとしてマークできます。

```
ALTER UNDO TABLESPACE tablespace_name SET {ACTIVE|INACTIVE};
```

テーブルスペースの状態を示す **STATE** カラムが **INFORMATION_SCHEMA.INNODB_TABLESPACES** テーブルに追加されました。**undo** テーブルスペースは、削除する前に **empty** 状態である必要があります。

- **innodb_undo_log_truncate** 変数はデフォルトで有効になっています。
- **innodb_rollback_segments** 変数は、**undo** テーブルスペースごとのロールバックセグメントの数を定義します。以前は、**innodb_rollback_segments** は MySQL インスタンスのロールバックセグメントの合計数を指定していました。この変更により、同時トランザクションに使用可能なロールバックセグメントの数が増加します。ロールバックセグメントを増やすと、同時トランザクションが **undo** ログに個別のロールバックセグメントを使用する可能性が高くなり、リソースの競合が少なくなります。
- バッファープールの事前フラッシュおよびフラッシュの動作に影響を与える変数のデフォルト値が変更されました:
 - **innodb_max_dirty_pages_pct_lwm** のデフォルト値は 10 になりました。前のデフォルト値の 0 は、バッファープールの事前フラッシュを無効にします。値 10 を指定すると、バッファープール内のダーティページの割合が 10% を超える場合に事前フラッシュが有効になります。事前フラッシュを有効にすると、パフォーマンスの一貫性が向上します。
 - **innodb_max_dirty_pages_pct** のデフォルト値が 75 から 90 に増加しました。**InnoDB** は、ダーティページの割合がこの値を超えないように、バッファープールからデータをフラッシュしようとしています。デフォルト値を大きくすると、バッファープール内のダーティページの割合が高くなります。
- デフォルトの **innodb_autoinc_lock_mode** 設定は 2 (インターリーブ) になりました。インターリーブロックモードでは、複数行の挿入を並列に実行できるため、同時実行性とスケーラビリティが向上します。新しい **innodb_autoinc_lock_mode** のデフォルト設定は、MySQL 5.7 のデフォルトのレプリケーションタイプがステートメントベースレプリケーションから行ベースレプリケーションに変更されたことを反映しています。ステートメントベースのレプリケーションでは、SQL ステートメントの実行順序に合わせて自動増分値が正しい順序で割り当てられるように、連続した自動増分ロックモード (前のデフォルト) が必要ですが、行ベースのレプリ

ケーションでは SQL ステートメントの実行順序に影響しません。詳細は、[InnoDB AUTO_INCREMENT のロックモード](#)を参照してください。

ステートメントベースレプリケーションを使用するシステムでは、新しい `innodb_autoinc_lock_mode` のデフォルト設定によって、シーケンシャルな自動インクリメント値に依存するアプリケーションが壊れる可能性があります。以前のデフォルトに戻すには、`innodb_autoinc_lock_mode` を 1 に設定します。

- 一般的なテーブルスペースの名前変更は、`ALTER TABLESPACE ... RENAME TO` 構文でサポートされています。
- デフォルトで無効になっている新しい `innodb_dedicated_server` 変数を使用すると、サーバーで検出されたメモリ量に応じて `InnoDB` で次のオプションを自動的に構成できます:

- `innodb_buffer_pool_size`
- `innodb_log_file_size`
- `innodb_flush_method`

このオプションは、専用サーバーで実行する MySQL サーバーインスタンスを対象としています。詳細は、[セクション 15.8.12 「専用 MySQL Server の自動構成の有効化」](#)を参照してください。

- 新しい `INFORMATION_SCHEMA.INNODB_TABLESPACES_BRIEF` ビューでは、`InnoDB` テーブルスペースの領域、名前、パス、フラグおよび領域タイプのデータが提供されます。
- MySQL にバンドルされている「[zlib ライブラリ](#)」バージョンは、バージョン 1.2.3 からバージョン 1.2.11 に引き上げられました。MySQL は `zlib` ライブラリを使用して圧縮を実装します。

`InnoDB` 圧縮テーブルを使用する場合、関連するアップグレードの影響については、[セクション 2.11.4 「MySQL 8.0 での変更」](#)を参照してください。

- シリアライズされたディクショナリ情報 (SDI) は、グローバル一時テーブルスペースおよび `undo` テーブルスペースファイルを除くすべての `InnoDB` テーブルスペースファイルに存在します。SDI は、テーブルおよびテーブルスペースオブジェクトのシリアライズされたメタデータです。SDI データが存在すると、メタデータの冗長性が提供されます。たとえば、データディクショナリが使用できなくなった場合、ディクショナリオブジェクトメタ

データをテーブルスペースファイルから抽出できます。SDI 抽出は、`ibd2sdi` ツールを使用して実行されます。SDI データは `JSON` 形式で格納されます。

SDI データをテーブルスペースファイルに含めると、テーブルスペースのファイルサイズが増加します。SDI レコードには、デフォルトで 16KB の単一のインデックスページが必要です。ただし、SDI データは格納時に圧縮され、ストレージフットプリントが削減されます。

- `InnoDB` ストレージエンジンはアトミック DDL をサポートするようになり、操作中にサーバーが停止した場合でも、DDL 操作が完全にコミットまたはロールバックされるようになりました。詳細は、[セクション 13.1.1 「アトミックデータ定義ステートメントのサポート」](#) を参照してください。
 - `innodb_directories` オプションを使用すると、サーバーがオフラインのときに、テーブルスペースファイルを新しい場所に移動またはリストアできます。詳細は、[セクション 15.6.3.6 「サーバーがオフラインのときのテーブルスペースファイルの移動」](#) を参照してください。
 - 次の redo ロギング最適化が実装されました:
 - ユーザースレッドは、書きみを同期せずにログバッファに同時に書き込むことができるようになりました。
 - ユーザースレッドは、ダーティページを緩やかな順序でフラッシュリストに追加できるようになりました。
 - 専用ログスレッドは、システムバッファへのログバッファの書き込み、ディスクへのシステムバッファのフラッシュ、書き込みおよびフラッシュされた redo についてのユーザースレッドへの通知、緩和されたフラッシュリストの順序に必要なラグの維持およびチェックポイントの書き込みを担当するようになりました。
 - フラッシュされた redo を待機しているユーザースレッドによるスピン遅延の使用を構成するためのシステム変数が追加されました:
 - `innodb_log_wait_for_flush_spin_hwm`: フラッシュされた redo の待機中にユーザースレッドがスピンしなくなる最大平均ログフラッシュ時間を定義します。
 - `innodb_log_spin_cpu_abs_lwm`: フラッシュされた redo の待機中にユーザースレッドがスピンしなくなる CPU 使用率の最小量を定義します。
 - `innodb_log_spin_cpu_pct_hwm`: フラッシュされた redo の待機中にユーザースレッドがスピンしなくなる CPU 使用率の最大量を定義します。
 - `innodb_log_buffer_size` 変数が動的になり、サーバーの実行中にログバッファのサイズ変更が可能になりました。
- 詳細は、[セクション 8.5.4 「InnoDB redo ロギングの最適化」](#) を参照してください。
- MySQL 8.0.12 では、ラージオブジェクト (LOB) データに対する小さい更新で undo ロギングがサポートされているため、サイズが 100 バイト以下の LOB 更新のパフォーマンスが向上します。以前は、LOB の更新は少なくと

も 1 つの LOB ページのサイズでしたが、数バイトしか変更できない更新には最適ではありませんでした。この拡張機能は、LOB データの部分更新のために MySQL 8.0.4 で追加されたサポートに基づいています。

- MySQL 8.0.12 では、[ALGORITHM=INSTANT](#) は次の [ALTER TABLE](#) 操作でサポートされています:
 - カラムの追加。この機能は、「[「インスタント ADD COLUMN」](#)」とも呼ばれます。制限が適用されます。[セクション15.12.1「オンライン DDL 操作」](#)を参照してください。
 - 仮想カラムの追加または削除。
 - カラムのデフォルト値の追加または削除。
 - [ENUM](#) または [SET](#) カラムの定義の変更。
 - インデックスタイプの変更。
 - テーブルの名前の変更。

[ALGORITHM=INSTANT](#) をサポートする操作では、データディクショナリのメタデータのみが変更されます。テーブルに対するメタデータロックは行われず、テーブルデータは影響を受けず、操作は即時に行われます。明示的に指定しない場合、[ALGORITHM=INSTANT](#) はそれをサポートする操作によってデフォルトで使用されます。[ALGORITHM=INSTANT](#) が指定されているがサポートされていない場合、操作はエラーですぐに失敗します。

[ALGORITHM=INSTANT](#) をサポートする操作の詳細は、[セクション15.12.1「オンライン DDL 操作」](#)を参照してください。

- MySQL 8.0.13 の時点で、[TempTable](#) ストレージエンジンはバイナリラージオブジェクト (BLOB) 型のカラムの格納をサポートしています。この拡張により、BLOB データを含む一時テーブルを使用するクエリーのパフォーマンスが向上します。以前は、BLOB データを含む一時テーブルは、[internal_tmp_disk_storage_engine](#) によって定義されたディスク上のストレージエンジンに格納されていました。詳細は、[セクション8.4.4「MySQL での内部一時テーブルの使用」](#)を参照してください。
- MySQL 8.0.13 では、[InnoDB](#) の保存データ暗号化機能は一般的なテーブルスペースをサポートしています。以前は、file-per-table テーブルスペースのみを暗号化できました。一般テーブルスペースの暗号化をサポートするために、[CREATE TABLESPACE](#) および [ALTER TABLESPACE](#) 構文が拡張され、[ENCRYPTION](#) 句が追加されました。

[INFORMATION_SCHEMA.INNODB_TABLESPACES](#) テーブルに、テーブルスペースが暗号化されているかどうかを示す [ENCRYPTION](#) カラムが含まれるようになりました。

一般的なテーブルスペース暗号化操作の監視を許可するために、[stage/innodb/alter tablespace \(encryption\)](#) パフォーマンススキーマステージインストゥルメントが追加されました。

- [innodb_buffer_pool_in_core_file](#) 変数を無効にすると、[InnoDB](#) バッファプールページが除外され、コアファイルのサイズが小さくなります。この変数を使用するには、[core_file](#) 変数を有効にし、オペレーティングシステムで [madvise\(\)](#) に対する [MADV_DONTDUMP](#) の POSIX 以外の拡張機能をサポートする必要があります。これは Linux 3.4 以降でサポートされています。詳細は、[セクション15.8.3.7「コアファイルからのバッファプールページの除外」](#)を参照してください。
- MySQL 8.0.13 では、オプティマイザによって作成されたユーザー作成の一時テーブルおよび内部一時テーブルは、一時テーブルスペースのプールからセッションに割り当てられたセッション一時テーブルスペースに格納されます。セッションが切断されると、その一時テーブルスペースは切り捨てられ、プールに解放されます。以前

のリリースでは、一時テーブルはグローバル一時テーブルスペース (`ibtmp1`) に作成されており、一時テーブルの削除後にディスク領域がオペレーティングシステムに戻されませんでした。

`innodb_temp_tablespaces_dir` 変数は、セッション一時テーブルスペースが作成される場所を定義します。デフォルトの場所は、データディレクトリ内の `#innodb_temp` ディレクトリです。

`INNODB_SESSION_TEMP_TABLESPACES` テーブルは、セッション一時テーブルスペースに関するメタデータを提供します。

グローバル一時テーブルスペース (`ibtmp1`) には、ユーザーが作成した一時テーブルに対する変更のロールバックセグメントが格納されるようになりました。

- MySQL 8.0.14 では、`InnoDB` はクラスタ化されたパラレルインデックス読取りをサポートしているため、`CHECK TABLE` のパフォーマンスを向上させることができます。この機能は、セカンダリインデックススキャンには適用されません。パラレルクラスタインデックス読取りを実行するには、`innodb_parallel_read_threads` セッション変数を 1 より大きい値に設定する必要があります。デフォルト値は 4 です。パラレルクラスタインデックス読取りの実行に使用されるスレッドの実際数は、`innodb_parallel_read_threads` 設定またはスキャンするインデックスサブツリーの数 (いずれか小さい方) によって決まります。
- 8.0.14 では、`innodb_dedicated_server` 変数が有効な場合、ログファイルのサイズと数は、自動的に構成されたバッファプールサイズに従って構成されます。以前は、サーバーで検出されたメモリー量に従ってログファイルサイズが構成されており、ログファイルの数は自動的に構成されませんでした。
- 8.0.14 では、`CREATE TABLESPACE` ステートメントの `ADD DATAFILE` 句はオプションで、`FILE` 権限のないユーザーがテーブルスペースを作成できます。`ADD DATAFILE` 句を指定せずに `CREATE TABLESPACE` ステートメントを実行すると、一意のファイル名でテーブルスペースデータファイルが暗黙的に作成されます。
- デフォルトでは、TempTable ストレージエンジンが占有しているメモリー量が `temptable_max_ram` 変数で定義されているメモリー制限を超えると、TempTable ストレージエンジンはメモリーマップされた一時ファイルのディスクからの割り当てを開始します。MySQL 8.0.16 では、この動作は `temptable_use_mmap` 変数によって制御されます。`temptable_use_mmap` を無効にすると、TempTable ストレージエンジンは、オーバーフローメカニズムとしてメモリーマップされたファイルの代わりに `InnoDB` ディスク上の内部一時テーブルを使用します。詳細は、[内部一時テーブルストレージエンジン](#) を参照してください。
- MySQL 8.0.16 では、`InnoDB` の保存データ暗号化機能は `mysql` システムテーブルスペースの暗号化をサポートしています。`mysql` システムテーブルスペースには、`mysql` システムデータベースおよび MySQL データディクショナリテーブルが含まれます。詳細は、[セクション 15.13 「InnoDB 保存データ暗号化」](#) を参照してください。
- MySQL 8.0.16 で導入された `innodb_spin_wait_pause_multiplier` 変数を使用すると、スレッドが `mutex` または `rw-lock` の取得を待機したときに発生するスピンロックポーリング遅延の期間をより詳細に制御できます。遅延は、プロセッサアーキテクチャーごとに `PAUSE` 命令の期間の違いを考慮して、より細かくチューニングできます。詳細は、[セクション 15.8.8 「スピンロックのポーリングの構成」](#) を参照してください。
- 大規模なデータセットの `InnoDB` のパラレル読取りスレッドパフォーマンスは、読取りスレッドの使用率の向上、パラレルスキャン中に発生するプリフェッチアクティビティのための読取りスレッド I/O の削減、およびパーティションのパラレルスキャンのサポートにより、MySQL 8.0.17 で向上しました。

パラレル読取りスレッド機能は、`innodb_parallel_read_threads` 変数によって制御されます。最大設定は 256 になりました。これは、すべてのクライアント接続のスレッドの合計数です。スレッド制限に達すると、接続は単一スレッドの使用にフォールバックします。
- MySQL 8.0.18 で導入された `innodb_idle_flush_pct` 変数を使用すると、アイドル期間中のページフラッシュに制限を設定できるため、ソリッドステートストレージデバイスの存続期間を延長できます。[アイドル期間中のバッファフラッシュの制限](#) を参照してください。
- ヒストグラム統計を生成するための `InnoDB` データの効率的なサンプリングは、MySQL 8.0.19 の時点でサポートされています。[ヒストグラム統計分析](#) を参照してください。
- MySQL 8.0.20 の時点では、二重書込みバッファ記憶域は二重書込みファイルにあります。以前のリリースでは、記憶域はシステムテーブルスペースに存在していました。システムテーブルスペースから記憶域を移動する

と、書込み待機時間が短縮され、スループットが向上し、二重書込みバッファページの配置に関して柔軟性が提供されます。拡張二重書込みバッファ構成には、次のシステム変数が導入されました:

- [innodb_doublewrite_dir](#)

二重書込みバッファファイルディレクトリを定義します。

- [innodb_doublewrite_files](#)

二重書込みファイルの数を定義します。

- [innodb_doublewrite_pages](#)

バッチ書込みのスレッド当たりの二重書込みページの最大数を定義します。

- [innodb_doublewrite_batch_size](#)

バッチで書き込む二重書込みページの数を定義します。

詳細は、[セクション15.6.4「二重書き込みバッファ」](#)を参照してください。

- ロックを待機しているトランザクションに優先順位を付ける競合対応トランザクションスケジューリング (CATS) アルゴリズムは、MySQL 8.0.20 で改善されました。トランザクションスケジューリングの重み計算が完全に個別のスレッドで実行されるようになり、計算のパフォーマンスと精度が向上しました。

トランザクションのスケジューリングにも使用されていた先入れ先出し (FIFO) アルゴリズムが削除されました。FIFO アルゴリズムは CATS アルゴリズムの改善によって不要になりました。FIFO アルゴリズムによって以前に実行されたトランザクションスケジューリングが CATS アルゴリズムによって実行されるようになりました。

[INFORMATION_SCHEMA.INNODB_TRX](#) テーブルに [TRX_SCHEDULE_WEIGHT](#) カラムが追加され、CATS アルゴリズムによって割り当てられたトランザクションスケジューリングの重みの問い合わせが可能になりました。

コードレベルのトランザクションスケジューリングイベントを監視するために、次の [INNODB_METRICS](#) カウンタが追加されました:

- [lock_rec_release_attempts](#)

レコードロックの解放の試行回数。

- [lock_rec_grant_attempts](#)

レコードロックの付与を試行する回数。

- [lock_schedule_refreshes](#)

トランザクションスケジューリングの重みを更新するために待機グラフが分析された回数。

詳細は、[セクション15.7.6「トランザクションスケジューリング」](#)を参照してください。

- MySQL 8.0.21 の時点では、テーブルおよび行リソースのロックキューへのアクセスを必要とする操作の同時実行性を向上させるために、ロックシステム mutex ([lock_sys->mutex](#)) がシャードラッチに置き換えられ、ロックキューがテーブルおよびページキューシャードのロックにグループ化され、各シャードが専用 mutex で保護されました。以前は、シングルロックシステム mutex はすべてのロックキューを保護していました。これは、同時実行性の高いシステムでの競合のポイントでした。新しいシャード実装では、ロックキューへのより詳細なアクセスが許可されます。

ロックシステム mutex ([lock_sys->mutex](#)) は、次のシャードラッチに置き換えられました:

- 64 個の読取り/書込みロックオブジェクト ([rw_lock_t](#)) で構成されるグローバルラッチ ([lock_sys->latches.global_latch](#))。個々のロックキューにアクセスするには、共有グローバルラッチとロックキューシャードのラッチが必要です。すべてのロックキューへのアクセスを必要とする操作では、排他的なグローバルラッチが使用され、すべてのテーブルおよびページロックキューのシャードがラッチされます。

- 512 の mutex の配列として実装されるテーブルシャードラッチ (`lock_sys->latches.table_shards.mutexes`)。各 mutex は 512 のテーブルロックキューシャードのいずれかが専用です。
- 512 個の mutex の配列として実装されるページシャードラッチ (`lock_sys->latches.page_shards.mutexes`)。各 mutex は 512 個のページロックキューシャードのいずれかが専用です。

単一ロックシステム相互排他ロックをモニタリングするためのパフォーマンススキーマ `wait/synch/mutex/innodb/lock_mutex` インストゥルメントは、新しいグローバルシャード、テーブルシャード、およびページシャードラッチをモニタリングするためのインストゥルメントに置き換えられました:

- `wait/synch/sxlock/innodb/lock_sys_global_rw_lock`
 - `wait/synch/mutex/innodb/lock_sys_table_mutex`
 - `wait/synch/mutex/innodb/lock_sys_page_mutex`
- MySQL 8.0.21 では、`DATA DIRECTORY` 句を使用してデータディレクトリの外部で作成されたテーブルおよびテーブルパーティションのデータファイルは、InnoDB で認識されているディレクトリに制限されます。この変更により、データベース管理者はテーブルスペースデータファイルが作成される場所を制御でき、リカバリ中にデータファイルが検出されるようになります。

一般テーブルスペースおよびファイルごとのテーブルスペースのデータファイル (`.ibd` ファイル) は、InnoDB で直接認識されないかぎり、undo テーブルスペースディレクトリ (`innodb_undo_directory`) に作成できなくなりました。

既知のディレクトリは、`datadir`、`innodb_data_home_dir` および `innodb_directories` 変数で定義されているディレクトリです。

file-per-table テーブルスペースに存在する InnoDB テーブルを切り捨てると、既存のテーブルスペースが削除され、新しいテーブルスペースが作成されます。MySQL 8.0.21 では、InnoDB はデフォルトの場所に新しいテーブルスペースを作成し、現在のテーブルスペースディレクトリが不明な場合はエラーログに警告を書き込みます。 `TRUNCATE TABLE` で現在の場所にテーブルスペースを作成するには、`TRUNCATE TABLE` を実行する前に `innodb_directories` 設定にディレクトリを追加します。

- MySQL 8.0.21 では、`ALTER INSTANCE {ENABLE|DISABLE} INNODB REDO_LOG` 構文を使用して redo ロギングを有効化および無効化できます。この機能は、新しい MySQL インスタンスにデータをロードするためのものです。redo ロギングを無効にすると、redo ログの書き込みが回避され、データのロードが高速化されます。

新しい `INNODB_REDO_LOG_ENABLE` 権限では、redo ロギングの有効化および無効化が許可されます。

新しい `Innodb_redo_log_enabled` ステータス変数を使用すると、redo ロギングステータスを監視できます。

[redo ロギングの無効化](#)を参照してください。

- 起動時に、InnoDB は、テーブルスペースファイルが別の場所に移動された場合に備えて、データディクショナリに格納されているテーブルスペースファイルパスに対して既知のテーブルスペースファイルのパスを検証します。MySQL 8.0.21 で導入された新しい `innodb_validate_tablespace_paths` 変数を使用すると、テーブルスペースパス検証を無効にできます。この機能は、テーブルスペースファイルを移動しない環境を対象としています。テーブルスペースパス検証を無効にすると、多数のテーブルスペースファイルがあるシステムでの起動時間が短縮されます。

詳細は、[セクション15.6.3.7「テーブルスペースパス検証の無効化」](#)を参照してください。

- MySQL 8.0.21 の時点では、アトミック DDL をサポートするストレージエンジンでは、行ベースレプリケーションが使用されているときに、`CREATE TABLE ... SELECT` ステートメントがバイナリログに 1 つのトランザクションとして記録されます。以前は、2 つのトランザクションとしてログに記録されていました。1 つはテーブルの作成用、もう 1 つはデータの挿入用です。この変更により、`CREATE TABLE ... SELECT` ステートメントは行ベースレプリケーションに対して安全になり、GTID ベースレプリケーションでの使用が許可されるようになりました。詳細は、[セクション13.1.1「アトミックデータ定義ステートメントのサポート」](#)を参照してください。
- ビジー状態のシステムで undo テーブルスペースを切り捨てると、バッファプールから古い undo テーブルスペースページを削除し、新しい undo テーブルスペースの初期ページをディスクにフラッシュするフラッシュ操作が

関連付けられているため、パフォーマンスに影響する可能性があります。この問題に対処するために、MySQL 8.0.21 でフラッシュ操作が削除されました。

古い undo テーブルスペースページは、最近最も使用されなくなるが、次の完全チェックポイントで削除されると、パッシブに解放されます。新しい undo テーブルスペースの初期ページは、切捨て操作中にディスクにフラッシュされるのではなく redo ログに記録されるようになりました。これにより、undo テーブルスペースの切捨て操作の永続性も向上します。

undo テーブルスペースの切捨て操作の数が多すぎることが原因で発生する潜在的な問題を回避するために、チェックポイント間の同じ undo テーブルスペースに対する切捨て操作は 64 に制限されるようになりました。この制限を超えると、undo テーブルスペースは非アクティブにできますが、次のチェックポイントまで切り捨てられません。

廃止された undo 切捨てフラッシュ操作に関連付けられた `INNODB_METRICS` カウンタが削除されました。削除されたカウンタには次が含まれます: `undo_truncate_sweep_count`, `undo_truncate_sweep_usec`, `undo_truncate_flush_count` および `undo_truncate_flush_usec`。

[セクション 15.6.3.4 「undo テーブルスペース」](#) を参照してください。

- MySQL 8.0.22 の時点では、新しい `innodb_extend_and_initialize` 変数を使用して、InnoDB が Linux 上の file-per-table および一般的なテーブルスペースに領域を割り当てる方法を構成できます。デフォルトでは、操作にテーブルスペースの追加領域が必要な場合、InnoDB はそのテーブルスペースにページを割り当て、それらのページに NULL を物理的に書き込みます。この動作は、新しいページが頻繁に割り当てられる場合のパフォーマンスに影響します。Linux システムで `innodb_extend_and_initialize` を無効にして、新しく割り当てられたテーブルスペースへの NULL の物理的な書き込みを回避できます。 `innodb_extend_and_initialize` が無効になっている場合、領域は `posix_fallocate()` コールを使用して割り当てられます。このコールは、物理的に NULL を書き込まずに領域を予約します。

`posix_fallocate()` 操作はアトミックではないため、テーブルスペースファイルへの領域の割当てとファイルメタデータの更新の間に障害が発生する可能性があります。このような障害が発生すると、新しく割り当てられたページは初期化されていない状態のままになり、InnoDB がこれらのページにアクセスしようとしたときに障害が発生する可能性があります。このシナリオを回避するために、InnoDB は新しいテーブルスペースページを割り当てる前に redo ログレコードを書き込みます。ページ割当て操作が中断されると、リカバリ中に redo ログレコードから操作がリプレイされます。

- MySQL 8.0.23 では、InnoDB は暗号化されたテーブルスペースに属する二重書き込みファイルページの暗号化をサポートしています。ページは、関連付けられたテーブルスペースの暗号化キーを使用して暗号化されます。詳細は、[セクション 15.13 「InnoDB 保存データ暗号化」](#) を参照してください。
- MySQL 8.0.23 で導入された `temptable_max_mmap` 変数は、TempTable ストレージエンジンが内部一時テーブルデータのディスクへの格納を開始する前に、メモリーマップ (MMAP) ファイルから割り当てることができるメモリーの最大量を定義します。0 に設定すると、MMAP ファイルからの割り当てが無効になります。詳細は、[セクション 8.4.4 「MySQL での内部一時テーブルの使用」](#) を参照してください。
- MySQL 8.0.23 で導入された `AUTOEXTEND_SIZE` オプションは、テーブルスペースが一杯になったときに InnoDB がテーブルスペースのサイズを拡張する量を定義し、テーブルスペースのサイズを大きく拡張できるようにします。 `AUTOEXTEND_SIZE` オプションは、`CREATE TABLE`, `ALTER TABLE`, `CREATE TABLESPACE` および `ALTER TABLESPACE` ステートメントでサポートされています。詳細は、[セクション 15.6.3.9 「テーブルスペースの AUTOEXTEND_SIZE 構成」](#) を参照してください。

`AUTOEXTEND_SIZE` サイズカラムが `INFORMATION_SCHEMA.INNODB_TABLESPACES` テーブルに追加されました。

- 文字セットのサポート。 デフォルトの文字セットが `latin1` から `utf8mb4` に変更されました。 `utf8mb4` 文字セットには、MySQL の Unicode で使用可能な最初の日本語固有の照合である `utf8mb4_ja_0900_as_cs` を含む、いくつかの新しい照合順序があります。詳細は、[セクション 10.10.1 「Unicode 文字セット」](#) を参照してください。

- JSON の拡張機能。 MySQL JSON 機能が次のように拡張または追加されました:

- `JSON_EXTRACT()` の結果で `JSON_UNQUOTE()` をコールするのと同等の `->>` (インラインパス) 演算子が追加されました。

これは、MySQL 5.7 で導入されたカラムパス演算子 `->` の改良です。 `col->>"$.path"` は `JSON_UNQUOTE(col->"$.path")` と同等です。 インラインパス演算子は、`SELECT` カラムリスト、`WHERE` 句と `HAVING` 句、`ORDER BY` 句と `GROUP BY` 句など、`JSON_UNQUOTE(JSON_EXTRACT())` を使用できる任意の場所で使用できます。 詳細は、演算子の説明および [JSON パス構文](#) を参照してください。

- 2 つの JSON 集計関数 `JSON_ARRAYAGG()` および `JSON_OBJECTAGG()` が追加されました。 `JSON_ARRAYAGG()` は、引数としてカラムまたは式を取り、結果を単一の JSON 配列カラムとして集計します。 式は任意の MySQL データ型に評価できます。これは JSON 値である必要はありません。 `JSON_OBJECTAGG()` では、キーおよび値として解釈される 2 つのカラムまたは式が使用され、結果は単一の JSON オブジェクトとして返されます。 詳細および例については、[セクション12.20「集計関数」](#) を参照してください。

- 既存の JSON 値を読みやすい形式で出力する JSON ユーティリティ関数 `JSON_PRETTY()` が追加されました。 各 JSON オブジェクトメンバーまたは配列値は別々の行に出力され、子オブジェクトまたは配列はその親に対して 2 文字のスペースでインデントされます。

この関数は、JSON 値として解析できる文字列でも機能します。

詳細および例については、[セクション12.18.8「JSON ユーティリティ関数」](#) を参照してください。

- `ORDER BY` を使用してクエリーで JSON 値をソートする場合、各値は固定サイズの 1K の一部ではなく、ソートキーの変長部分で表されるようになりました。 多くの場合、これにより過剰な使用量が削減される可能性があります。 たとえば、スカラー `INT` または `BIGINT` 値には、実際には非常に少ないバイト数が必要であるため、この領域の残り (最大 90% 以上) はパディングによって占有されています。 この変更には、パフォーマンスに関して次の利点があります:

- ソートバッファ領域がより効率的に使用されるようになったため、ファイルソートは固定長のソートキーを使用する場合と同じくらい早くディスクにフラッシュする必要はありません。 つまり、メモリー内でソートできるデータが増え、不要なディスクアクセスが回避されます。

- 短いキーは長いキーよりも迅速に比較できるため、パフォーマンスが著しく向上します。 これは、メモリー内で完全に実行されるソートと、ディスクへの書き込みおよびディスクからの読み取りが必要なソートに当てはまりません。

- JSON カラム値の部分的なインプレース更新のサポートが MySQL 8.0.2 に追加されました。 これは、JSON カラムの更新時に以前に行ったように、既存の JSON 値を完全に削除してその場所に新しい JSON 値を書き込むよりも効率的です。 この最適化を適用するには、`JSON_SET()`、`JSON_REPLACE()` または `JSON_REMOVE()` を使用して更新を適用する必要があります。 更新中の JSON ドキュメントに新しい要素を追加することはできません。 ドキュメント内の値は、更新前よりも多くの領域を取ることはできません。 要件の詳細は、[JSON 値の部分更新](#) を参照してください。

JSON ドキュメントの部分更新はバイナリログに書き込むことができ、完全な JSON ドキュメントをロギングするよりも少ない領域を占有します。 ステートメントベースのレプリケーションが使用されている場合、部分更新は常にそのように記録されます。 これを行ベースのレプリケーションで使用するには、まず `binlog_row_value_options=PARTIAL_JSON` を設定する必要があります。 詳細は、この変数の説明を参照してください。

- JSON ユーティリティ関数 `JSON_STORAGE_SIZE()` および `JSON_STORAGE_FREE()` が追加されました。 `JSON_STORAGE_SIZE()` は、部分更新の前に JSON ドキュメントのバイナリ表現に使用される記憶領域をバイト単位で返します (前の項目を参照)。 `JSON_STORAGE_FREE()` では、`JSON_SET()` または `JSON_REPLACE()` を使用して部分的に更新された後、JSON 型のテーブルのカラムに残っている領域の量が表示されます。 新しい値のバイナリ表現が前の値より小さい場合、これはゼロより大きくなります。

これらの各関数は、JSON ドキュメントの有効な文字列表現も受け入れます。 このような値の場合、`JSON_STORAGE_SIZE()` は JSON ドキュメントへの変換後にバイナリ表現で使用される領域を返します。 JSON ドキュメントの文字列表現を含む変数の場合、`JSON_STORAGE_FREE()` はゼロを返します。 いずれの

関数も、その (null 以外の) 引数を有効な JSON ドキュメントとして解析できない場合はエラーを生成し、引数が NULL の場合は NULL を生成します。

詳細および例については、[セクション12.18.8「JSON ユーティリティ関数」](#)を参照してください。

[JSON_STORAGE_SIZE\(\)](#) および [JSON_STORAGE_FREE\(\)](#) は、MySQL 8.0.2 に実装されました。

- MySQL 8.0.2 では、XPath 式で `[$[1 to 5]]`などの範囲をサポートするようになりました。また、このバージョンでの `last` キーワードと相対アドレス指定のサポートが追加され、`[$[last]]`は常に配列の最後 (最も高い番号) の要素を選択し、`[$[last-1]]`は最後の 1 つ前の要素を選択するようになりました。`last` およびそれを使用する式は、範囲定義に含めることもできます。たとえば、`[$[last-2 to last-1]]`は、配列の最後を除いた手前 2 つの要素を返します。追加情報および例については、[JSON 値の検索および変更](#)を参照してください。
- [RFC 7396](#) に準拠するための JSON マージ関数が追加されました。[JSON_MERGE_PATCH\(\)](#) は、2 つの JSON オブジェクトで使用される場合、次のセットの結合をメンバーとして持つ単一の JSON オブジェクトにマージします:
 - 2 番目のオブジェクトに同じキーを持つメンバがない最初のオブジェクトの各メンバ。
 - 最初のオブジェクトに同じキーを持つメンバーが存在せず、その値が JSON `null` リテラルではない、2 番目のオブジェクトの各メンバー。
 - 両方のオブジェクトに存在し、2 番目のオブジェクトの値が JSON `null` リテラルではないキーを持つ各メンバー。

この作業の一環として、[JSON_MERGE\(\)](#) 関数の名前は [JSON_MERGE_PRESERVE\(\)](#) に変更されました。[JSON_MERGE\(\)](#) は、MySQL 8.0 で [JSON_MERGE_PRESERVE\(\)](#) のエイリアスとして引き続き認識されますが、非推奨になり、MySQL の将来のバージョンで削除される予定です。

詳細および例については、[セクション12.18.4「JSON 値を変更する関数」](#)を参照してください。

- [RFC 7159](#) およびほとんどの JavaScript パーサーと同様の、重複キーの「最後の重複キー優先」正規化を実装しました。この動作の例を次に示します。ここでは、キー `x` を持つ右端のメンバーのみが保持されます:

```
mysql> SELECT JSON_OBJECT('x', '32', 'y', '[true, false]',
>      'x', '"abc"', 'x', '100') AS Result;
+-----+
| Result |
+-----+
| {"x": "100", "y": "[true, false]} |
+-----+
1 row in set (0.00 sec)
```

次の例に示すように、MySQL JSON カラムに挿入された値もこの方法で正規化されます:

```
mysql> CREATE TABLE t1 (c1 JSON);

mysql> INSERT INTO t1 VALUES ('{"x": 17, "x": "red", "x": [3, 5, 7]}');

mysql> SELECT c1 FROM t1;
+-----+
| c1 |
+-----+
| {"x": [3, 5, 7]} |
+-----+
```

これは、このような場合に「最初の重複キー優先」アルゴリズムが使用されていた以前のバージョンの MySQL とは互換性のない変更です。

詳細および例については、[JSON 値の正規化、マージおよび自動ラップ](#)を参照してください。

- MySQL 8.0.4 に `JSON_TABLE()` 関数が追加されました。この関数は、JSON データを受け入れ、指定されたカラムを持つリレーショナルテーブルとして戻します。

この関数の構文は、`JSON_TABLE(expr, path COLUMNS column_list) [AS] alias` です。ここで、`expr` は JSON データを返す式、`path` はソースに適用される JSON パス、`column_list` はカラム定義のリストです。次に例を示します:

```
mysql> SELECT *
-> FROM
-> JSON_TABLE(
-> [{"a":3,"b":"0"}, {"a":"3","b":"1"}, {"a":2,"b":1}, {"a":0}, {"b":[1,2]}],
-> "$[*]" COLUMNS(
-> rowid FOR ORDINALITY,
->
-> xa INT EXISTS PATH "$.a",
-> xb INT EXISTS PATH "$.b",
->
-> sa VARCHAR(100) PATH "$.a",
-> sb VARCHAR(100) PATH "$.b",
->
-> ja JSON PATH "$.a",
-> jb JSON PATH "$.b"
-> )
-> ) AS jt1;
```

rowid	xa	xb	sa	sb	ja	jb
1	1	1	3	0	3	"0"
2	1	1	3	1	"3"	"1"
3	1	1	2	1	2	1
4	1	0	0	NULL	0	NULL
5	0	1	NULL	NULL	NULL	[1, 2]

JSON ソース式には、JSON リテラル、テーブルのカラム、`JSON_EXTRACT(t1, data, '$.post.comments')` などの JSON を返す関数コールなど、有効な JSON ドキュメントを生成する任意の式を指定できます。詳細は、[セクション12.18.6「JSON テーブル関数」](#)を参照してください。

- データ型のサポート。MySQL では、データ型指定でのデフォルト値としての式の使用がサポートされるようになりました。これには、以前はデフォルト値を割り当てることができなかった `BLOB`、`TEXT`、`GEOMETRY` および `JSON` データ型のデフォルト値としての式の使用が含まれます。詳細は、[セクション11.6「データ型デフォルト値」](#)を参照してください。
- オプティマイザ。次のオプティマイザ拡張機能が追加されました:
 - MySQL で不可視インデックスがサポートされるようになりました。不可視のインデックスはオプティマイザではまったく使用されませんが、それ以外の場合は正常にメンテナンスされます。インデックスはデフォルトで可視化されます。不可視のインデックスを使用すると、インデックスが必要になった場合に元に戻す必要がある破壊的な変更を行わずに、クエリーのパフォーマンスに対するインデックスの削除の影響をテストできます。[セクション8.3.12「不可視のインデックス」](#)を参照してください。
 - MySQL で降順インデックスがサポートされるようになりました: インデックス定義内の `DESC` は無視されなくなりましたが、キー値が降順で格納されます。以前は、インデックスを逆の順序でスキャンできましたが、パフォーマンスが低下していました。降順インデックスは順にスキャンできるため、より効率的です。降順インデックスを使用すると、オプティマイザが複数カラムインデックスを使用できるようになります (最も効率的なスキャン順序で一部のカラムに昇順が混在し、他のカラムに降順が混在している場合)。[セクション8.3.13「降順インデックス」](#)を参照してください。
 - MySQL では、カラム値ではなく式の値をインデックス付けする関数インデックスキー部分の作成がサポートされるようになりました。関数キーパーツを使用すると、`JSON` 値など、インデックス化できない値のインデックス化が可能です。詳細は、[セクション13.1.15「CREATE INDEX ステートメント」](#)を参照してください。

- MySQL 8.0.14 以降では、定数リテラル式から発生する自明の `WHERE` 条件は、後で最適化するのではなく、準備中に削除されます。このプロセスの前半で条件を削除すると、次のような簡易条件を持つ外部結合を含むクエリーの結合を簡略化できます:

```
SELECT * FROM t1 LEFT JOIN t2 ON condition_1 WHERE condition_2 OR 0 = 1
```

オプティマイザは、準備中に `0 = 1` が常に `false` であることを確認し、`OR 0 = 1` を冗長にして削除し、次の状態のままにします:

```
SELECT * FROM t1 LEFT JOIN t2 ON condition_1 where condition_2
```

これで、オプティマイザは、次のようにクエリーを内部結合としてリライトできます:

```
SELECT * FROM t1 LEFT JOIN t2 WHERE condition_1 AND condition_2
```

詳細は、[セクション8.2.1.9「外部結合の最適化」](#)を参照してください。

- MySQL 8.0.16 以降では、MySQL は最適化時に定数折りたたみを使用して、実行時に行ごとに行うのではなく、カラムと定数値 (定数がカラムの範囲外またはカラムのタイプに対する範囲境界) の比較を処理できます。たとえば、`TINYINT UNSIGNED` カラム `c` を含むテーブル `t` の場合、オプティマイザは `WHERE c < 256` などの条件を `WHERE 1` にリライト (および条件を完全に最適化) したり、`WHERE c >= 255` を `WHERE c = 255` にリライトできます。

詳しくは[セクション8.2.1.14「定数 - フォールディングの最適化」](#),をご覧ください。

- MySQL 8.0.16 以降、`IN` サブクエリーで使用される準結合最適化を `EXISTS` サブクエリーにも適用できるようになりました。また、オプティマイザでは、サブクエリーにアタッチされた `WHERE` 条件のわずかな相関等価述語が解除され、`IN` サブクエリーの式と同様に処理できるようになりました。これは、`EXISTS` サブクエリーと `IN` サブクエリーの両方に適用されます。

詳細は、[セクション8.2.2.1「準結合変換による IN および EXISTS サブクエリー述語の最適化」](#)を参照してください。

- MySQL 8.0.17 では、サーバーはコンテキスト化フェーズ中に不完全な SQL 述語 (つまり、`value` がカラム名または定数式で、比較演算子が使用されていない `WHERE value` 形式の述語) を `WHERE value <> 0` として内部的にリライトするため、クエリーリゾルバ、クエリーオプティマイザおよびクエリーエグゼキュータは完全な述語のみ処理すればよくなりました。

この変更の表示可能な効果の 1 つは、ブール値の場合、`EXPLAIN` 出力に `1` および `0` ではなく `true` および `false` が表示されるようになったことです。

この変更によるもう 1 つの効果は、SQL ブールコンテキストで JSON 値を評価すると、JSON 整数 0 に対して暗黙の比較が行われることです。次のように作成および移入されたテーブルについて考えてみます:

```
mysql> CREATE TABLE test (id INT, col JSON);
```

```
mysql> INSERT INTO test VALUES (1, '{"val":true}'), (2, '{"val":false}');
```

以前は、`IS TRUE` を使用した次のクエリーに示すように、サーバーは、抽出された `true` または `false` の値を SQL ブールコンテキストで比較するときに SQL ブールに変換しようとしていました:

```
mysql> SELECT id, col, col->"$.val" FROM test WHERE col->"$.val" IS TRUE;
+----+-----+-----+
| id | col          | col->"$.val" |
+----+-----+-----+
| 1  | {"val": true} | true         |
+----+-----+-----+
```

MySQL 8.0.17 以降では、抽出された値を JSON 整数 0 と暗黙の照合により、異なる結果をもたらします:

```
mysql> SELECT id, col, col->"$.val" FROM test WHERE col->"$.val" IS TRUE;
+----+-----+-----+
| id | col          | col->"$.val" |
+----+-----+-----+
| 1  | {"val": true} | true         |
+----+-----+-----+
```

```
| 2 | {"val": false} | false |
+-----+-----+-----+
```

MySQL 8.0.21 以降では、次に示すように、抽出された値に対して `JSON_VALUE()` を使用して、テストを実行する前に型変換を実行できます:

```
mysql> SELECT id, col, col->"$.val" FROM test
-> WHERE JSON_VALUE(col, "$.val" RETURNING UNSIGNED) IS TRUE;
+-----+-----+-----+
| id | col      | col->"$.val" |
+-----+-----+-----+
| 1 | {"val": true} | true      |
+-----+-----+-----+
```

また、MySQL 8.0.21 以降では、この方法で SQL ブールコンテキストの抽出された値を比較する際に、サーバーによって警告「Evaluating a JSON value in SQL boolean context does an implicit comparison against JSON integer 0; if this is not what you want, consider converting JSON to an SQL numeric type with `JSON_VALUE RETURNING` when comparing extracted values in an SQL boolean context in this manner. (SQL ブールコンテキストで JSON 値を評価すると、JSON の整数 0 との暗黙の比較が行われます。想定と違う場合は、`JSON_VALUE RETURNING` を使用して JSON を SQL 数値型に変換することを検討してください)」が提供されます。

- MySQL 8.0.17 以降では、`NOT IN (subquery)` または `NOT EXISTS (subquery)` を含む `WHERE` 条件は内部的にアンチ結合に変換されます。(アンチ結合は、結合先のテーブルにある、結合条件に一致する行以外のすべての行を返します。)サブクエリーのテーブルが最優先で処理されるようになったことで、遅いサブクエリーが削除されて結果的に問い合わせの実行が速くなります。

これは、外部結合用の既存の `IS NULL (Not exists)` 最適化に似ており、それを再利用したものです。[EXPLAIN 追加情報](#) を参照してください。

- MySQL 8.0.21 以降、多くの場合、単一テーブルの `UPDATE` ステートメントまたは `DELETE` ステートメントで準結合変換またはサブクエリーの実体化を使用できるようになりました。これは、次に示す形式のステートメントに適用されます:
 - `UPDATE t1 SET t1.a=value WHERE t1.a IN (SELECT t2.a FROM t2)`
 - `DELETE FROM t1 WHERE t1.a IN (SELECT t2.a FROM t2)`

これは、次の条件を満たす単一テーブルの `UPDATE` または `DELETE` に対して実行できます:

- `UPDATE` ステートメントまたは `DELETE` ステートメントでは、`[NOT] IN` 述語または `[NOT] EXISTS` 述語を持つサブクエリーが使用されます。
- ステートメントには `ORDER BY` 句がなく、`LIMIT` 句もありません。

(`UPDATE` および `DELETE` の複数テーブルバージョンでは、`ORDER BY` または `LIMIT` はサポートされていません。)

- ターゲットテーブルでは、読取り/書込み削除はサポートされていません (`NDB` テーブルにのみ関連します)。
- 準結合またはサブクエリーの実体化は、サブクエリーに含まれるヒントおよび `optimizer_switch` の値に基づいて実行できます。

準結合最適化が適格な単一テーブル `DELETE` または `UPDATE` に使用されている場合、これはオプティマイザトレースに表示されます: 複数テーブルのステートメントの場合はトレースに `join_optimization` オブジェクトがありますが、単一テーブルのステートメントの場合はありません。変換は、`EXPLAIN FORMAT=TREE` または `EXPLAIN ANALYZE` の出力にも表示されます。単一テーブルのステートメントは `<not executable by iterator executor>` を示し、複数テーブルのステートメントは完全な計画をレポートします。

MySQL 8.0.21 以降では、`REPEATABLE READ` より弱いトランザクション分離レベルのために、`InnoDB` テーブルを使用する複数テーブルの `UPDATE` ステートメントで半一貫性読取りがサポートされます。

- ハッシュ結合パフォーマンスの向上。MySQL 8.0.23 はハッシュ結合に使用されるハッシュテーブルを再実装するため、ハッシュ結合のパフォーマンスがいくつか向上します。この作業には問題 (Bug #31516149, Bug

#99933) の修正が含まれており、結合バッファ (`join_buffer_size`) に割り当てられたメモリーの約 2/3 のみをハッシュ結合で実際に使用できます。

通常、新しいハッシュテーブルは古いハッシュテーブルより高速で、位置合せ、キー/値、および同等のキーが多数あるシナリオで使用されるメモリーが少なくなります。また、ハッシュテーブルのサイズが増えると、サーバーは古いメモリーを解放できるようになりました。

- 共通テーブル式. MySQL では、非再帰テーブルと再帰テーブルの両方の共通テーブル式がサポートされるようになりました。共通テーブル式を使用すると、名前付き一時結果セットを使用できます。この結果セットは、`SELECT` ステートメントおよびその他の特定のステートメントの前に `WITH` 句を記述する形で実装されます。詳細は、[セクション 13.2.15 「WITH \(共通テーブル式\)」](#) を参照してください。

MySQL 8.0.19 では、再帰的共通テーブル式 (CTE) の再帰的 `SELECT` 部分で `LIMIT` 句がサポートされています。`LIMIT` と `OFFSET` もサポートされています。詳しくは [再帰的な共通テーブル式](#) をご覧ください。
- ウィンドウ関数. MySQL では、クエリーの各行について、その行に関連する行を使用して計算を実行するウィンドウ関数がサポートされるようになりました。これには、`RANK()`、`LAG()`、`NTILE()` などの関数が含まれます。また、複数の既存の集計関数をウィンドウ関数 (`SUM()` や `AVG()` など) として使用できるようになりました。詳細は、[セクション 12.21 「ウィンドウ関数」](#) を参照してください。
- ラテラル導出テーブル. 導出テーブルの前に `LATERAL` キーワードを付けて、同じ `FROM` 句内の前述のテーブルのカラムを参照 (依存) できるように指定できるようになりました。ラテラル導出テーブルを使用すると、非ラテラル導出テーブルでは実行できない特定の SQL 操作や、より効率的な回避策が必要になる可能性があります。[セクション 13.2.11.9 「ラテラル導出テーブル」](#) を参照してください。
- 単一テーブル `DELETE` ステートメントのエイリアス. MySQL 8.0.16 以降では、単一テーブルの `DELETE` ステートメントでテーブルのエイリアスの使用がサポートされます。
- 正規表現のサポート. 以前は、MySQL は Henry Spencer 正規表現ライブラリを使用して正規表現演算子 (`REGEXP`、`RLIKE`) をサポートしていました。完全な Unicode サポートを提供し、マルチバイトセーフな International Components for Unicode (ICU) を使用して、正規表現のサポートが再実装されました。`REGEXP_LIKE()` 関数は、`REGEXP` 演算子および `RLIKE` 演算子 (現在はその関数のシノニム) の方法で正規表現の一致を実行します。また、`REGEXP_INSTR()`、`REGEXP_REPLACE()` および `REGEXP_SUBSTR()` の各関数を使用して、一致する位置を検索し、部分文字列の置換および抽出をそれぞれ実行できます。`regexp_stack_limit` および `regexp_time_limit` システム変数は、照合エンジンによるリソース消費を制御します。詳細については、[セクション 12.8.2 「正規表現」](#) を参照してください。正規表現を使用するアプリケーションが実装の変更の影響を受ける方法の詳細は、[正規表現の互換性に関する考慮事項](#) を参照してください。
- 内部一時テーブル. `TempTable` ストレージエンジンは、インメモリー内部一時テーブルのデフォルトエンジンとして `MEMORY` ストレージエンジンを置き換えます。`TempTable` ストレージエンジンは、`VARCHAR` および `VARBINARY` カラムの効率的なストレージを提供します。`internal_tmp_mem_storage_engine` セッション変数は、インメモリー内部一時テーブルのストレージエンジンを定義します。許可される値は、`TempTable` (デフォルト) および `MEMORY` です。`temptable_max_ram` 変数は、データがディスクに格納される前に `TempTable` ストレージエンジンが使用できるメモリーの最大量を定義します。
- ロギング. エラーロギングは、MySQL コンポーネントアーキテクチャを使用するようにリライトされました。従来のエラーロギングは組込みコンポーネントを使用して実装され、システムログを使用したロギングはロード可能コンポーネントとして実装されます。また、ロード可能な JSON ログライターも使用できます。有効にするログコンポーネントを制御するには、`log_error_services` システム変数を使用します。詳細は、[セクション 5.4.2 「エラーログ」](#) を参照してください。
- バックアップロック. 新しいタイプのバックアップロックでは、オンラインバックアップ中に DML が許可されますが、一貫性のないスナップショットになる可能性がある操作は防止されます。新しいバックアップロックは、`LOCK INSTANCE FOR BACKUP` および `UNLOCK INSTANCE` 構文でサポートされています。これらのステートメントを使用するには、`BACKUP_ADMIN` 権限が必要です。
- レプリケーション. MySQL レプリケーションが次のように拡張されました:
 - MySQL レプリケーションでは、コンパクトなバイナリ形式を使用した JSON ドキュメントへの部分的な更新のバイナリロギングをサポートし、完全な JSON ドキュメントを記録するよりもログの領域を節約できるようになりました。このようなコンパクトロギングは、ステートメントベースのロギングが使用されているときに自動的に実行され、新しい `binlog_row_value_options` システム変数を `PARTIAL_JSON` に設定することで有効にできます。詳細は、[JSON 値の部分更新](#) および `binlog_row_value_options` の説明を参照してください。

- **接続管理.** MySQL Server では、管理接続専用に TCP/IP ポートを構成できるようになりました。これは、通常接続用のネットワークインターフェースで許可されていた追加の管理用接続 (`max_connections` の接続がすでに確立されている場合でも 1 つだけ追加で管理接続が可能) の代替となります。 [セクション5.1.12.1「接続インターフェース」](#) を参照してください。

MySQL では、サーバーへの接続を介して送信されるバイト数を最小限に抑えるために、圧縮の使用をより詳細に制御できるようになりました。以前は、特定の接続が `zlib` 圧縮アルゴリズムを圧縮解除または使用していました。 `zstd` アルゴリズムを使用して、 `zstd` 接続の圧縮レベルを選択することもできます。許可される圧縮アルゴリズムは、サーバー側でも接続元 (クライアントプログラムおよび、ソース/レプリカレプリケーションやグループレプリケーションに参加しているサーバー) の側でも設定することができます。詳細は、 [セクション4.2.8「接続圧縮制御」](#) を参照してください。

- **構成.** MySQL 全体で許可されているホスト名の最大長は、以前の 60 文字の制限から 255 文字まで ASCII 文字になりました。これは、データディクショナリ内のホスト名関連のカラム、 `mysql` システムスキーマ、パフォーマンススキーマ、 `INFORMATION_SCHEMA` および `sys` スキーマ、 `MASTER_HOST` 値 (`CHANGE MASTER TO` 属性用)、 `Host` カラム (`SHOW PROCESSLIST` ステートメントの出力内)、アカウント名内のホスト名 (`account-management` ステートメントおよび `DEFINER` 属性内で使用など)、およびホスト名関連のシステム変数に適用されます。

注意事項:

- 許可されるホスト名の長さを増やすと、ホスト名カラムにインデックスがあるテーブルに影響する可能性があります。たとえば、ホスト名をインデックス付ける `mysql` システムスキーマのテーブルには、長いインデックス値を格納するために `DYNAMIC` の明示的な `ROW_FORMAT` 属性が含まれるようになりました。
- ファイル名を値として持つ構成設定の中には、サーバーのホスト名に基づいて構築されるものがあります。許容される値は、255 文字のホスト名を含むような長いファイル名を許可していないケースのように、基礎となるオペレーティングシステムによって制約されます。これは、 `general_log_file`、 `log_error`、 `pid_file`、 `relay_log`、 `slow_query_log_file` システム変数および対応するオプションに影響します。ホスト名に基づく値が OS に対して長すぎる場合は、明示的に短い値を指定する必要があります。
- サーバーで 255 文字のホスト名がサポートされるようになりましたが、 `--ssl-mode=VERIFY_IDENTITY` オプションを使用して確立されるサーバーへの接続は、OpenSSL でサポートされるホスト名の最大長によって制約されます。ホスト名の一致は、SSL 証明書の 2 つのフィールドに関連し、最大長は次のとおりです: 共通名: 最大長 64; サブジェクト代替名: RFC#1034 に従った最大長。
- **プラグイン.** 以前は、MySQL プラグインは C または C++ で記述できました。プラグインで使用される MySQL ヘッダーファイルに C++ コードが含まれるようになりました。つまり、プラグインは C ではなく C++ で記述する必要があります。
- **C API.** MySQL C API では、MySQL サーバーとのノンブロック通信用の非同期関数がサポートされるようになりました。各関数は、既存の同期関数に対応する非同期関数です。同期関数は、サーバー接続からの読取りまたはサーバー接続への書き込みを待機する必要がある場合にブロックします。非同期関数を使用すると、アプリケーションはサーバー接続での作業を続行する準備ができているかどうかを確認できます。そうでない場合、アプリケーションが他の作業を行った後で、再確認することができます。 [C API Asynchronous Interface](#) を参照してください。
- **キャストの追加のターゲットタイプ.** `CAST()` および `CONVERT()` の関数は、 `DOUBLE`、 `FLOAT` および `REAL` 型への変換をサポートするようになりました。MySQL 8.0.17 で追加されました。 [セクション12.11「キャスト関数と演算子」](#) を参照してください。
- **JSON スキーマ検証.** MySQL 8.0.17 には、JSON ドキュメントを再度 JSON スキーマで検証するための `JSON_SCHEMA_VALID()` および `JSON_SCHEMA_VALIDATION_REPORT()` の 2 つの関数が追加されています。 `JSON_SCHEMA_VALID()` は、ドキュメントがスキーマに対して検証される場合は `TRUE` (1) を返し、検証されない場合は `FALSE` (0) を返します。 `JSON_SCHEMA_VALIDATION_REPORT()` は、検証の結果に関する詳細情報を含む JSON ドキュメントを返します。次のステートメントは、これらの両方の関数に適用されます:
 - スキーマは、JSON スキーマ仕様のドラフト 4 に準拠する必要があります。
 - `required` 属性がサポートされています。
 - 外部リソースおよび `$ref` キーワードはサポートされていません。
 - 正規表現パターンがサポートされています。無効なパターンは暗黙的に無視されます。

詳細および例については、[セクション12.18.7「JSON スキーマ検証関数」](#)を参照してください。

- 複数值インデックス。MySQL 8.0.17 以降、InnoDB では複数值インデックスの作成がサポートされています。これは、値の配列を格納し、単一のデータレコードに対して複数のインデックスレコードを持つことができる JSON カラムに定義されたセカンダリインデックスです。このようなインデックスでは、`CAST(data->$$.zipcode' AS UNSIGNED ARRAY)` などのキー部分定義が使用されます。複数值インデックスは、`EXPLAIN` の出力で表示できるように、適切なクエリのために MySQL オプティマイザによって自動的に使用されます。

この作業の一環として、MySQL では、JSON ドキュメントを操作するための新しい関数 `JSON_OVERLAPS()` および新しい `MEMBER OF()` 演算子が追加され、さらに次のリストで説明するように、`CAST()` 関数が新しい `ARRAY` キーワードで拡張されます:

- `JSON_OVERLAPS()` では、2 つの JSON 文書が比較されます。共通のキーと値のペアまたは配列要素が含まれている場合、この関数は TRUE (1) を返し、それ以外の場合は FALSE (0) を返します。両方の値がスカラーの場合、関数は等価性の単純なテストを実行します。一方の引数が JSON 配列で、もう一方がスカラーの場合、スカラーは配列要素として扱われます。したがって、`JSON_OVERLAPS()` は `JSON_CONTAINS()` の補完として機能します。
- `MEMBER OF()` は、最初オペランド (スカラーまたは JSON ドキュメント) が 2 番目のオペランドとして渡された JSON 配列のメンバーであるかどうかをテストし、メンバーである場合は TRUE (1)、そうでない場合は FALSE (0) を返します。オペランドの型変換は実行されません。
- `CAST(expression AS type ARRAY)` では、`json_path` の JSON ドキュメントにある JSON 配列を SQL 配列にキャストすることで、関数インデックスを作成できます。型指定子は、`CAST()` ですすでにサポートされているものに限られます。ただし `BINARY` (サポートされていません) を除きます。`CAST()` (および `ARRAY` キーワード) のこの使用法は、InnoDB でのみサポートされ、複数值インデックスの作成にのみ使用できます。

例を含む複数值インデックスの詳細は、[複数值インデックス](#)を参照してください。[セクション12.18.3「JSON 値を検索する関数」](#)では、`JSON_OVERLAPS()` および `MEMBER OF()` に関する情報を使用例とともに提供します。

- `time_zone` を使用したオプティマイザヒント。MySQL 8.0.17 では、`SET_VAR` によるオプティマイザヒントに `time_zone` セッション変数を使用可能です。
- redo ログのアーカイブ。MySQL 8.0.17 では、InnoDB は redo ログのアーカイブをサポートしています。redo ログレコードをコピーするバックアップユーティリティは、バックアップ操作の進行中に redo ログの生成に対応できない場合があり、その結果、それらのレコードが上書きされるために redo ログレコードが失われます。redo ログアーカイブ機能は、redo ログレコードをアーカイブファイルに順次書き込むことで、この問題に対処します。バックアップユーティリティでは、必要に応じてアーカイブファイルから redo ログレコードをコピーできるため、データが失われる可能性を回避できます。詳細は、[redo ログのアーカイブ](#)を参照してください。
- クローンプラグイン。MySQL 8.0.17 の時点で、MySQL は、InnoDB データをローカルまたはリモートの MySQL サーバーインスタンスからクローニングできるクローンプラグインを提供します。ローカルクローニング操作では、MySQL インスタンスが実行されているのと同じサーバーまたはノードにクローンデータが格納されます。リモートクローニング操作では、クローニング操作が開始されたドナー MySQL サーバーインスタンスから受信者サーバーまたはノードに、ネットワーク経由でクローンデータが転送されます。

クローンプラグインはレプリケーションをサポートします。クローニング操作では、クローニングデータに加えて、ドナーからレプリケーション座標が抽出および転送され、受信者に適用されるため、グループレプリケーションメンバーおよびレプリカのプロビジョニングにクローンプラグインを使用できます。プロビジョニングにクローンプラグインを使用すると、多数のトランザクションをレプリケートするよりもはるかに高速かつ効率的になります。グループレプリケーションメンバーは、シードメンバーからグループデータを取得する最も効率的な方法をメンバーが自動的に選択できるように、代替のリカバリ方法としてクローンプラグインを使用するように構成することもできます。

詳細は、[セクション5.6.7「クローンプラグイン」](#) および [セクション18.4.3.2「分散リカバリのためのクローニング」](#)を参照してください。

- ハッシュ結合の最適化。MySQL 8.0.18 以降、結合内のテーブルの各ペアに少なくとも 1 つの等価結合条件が含まれ、結合条件にはインデックスが適用されない場合は常にハッシュ結合が使用されます。ハッシュ結合はインデックスを必要としませんが、単一テーブルの述語にのみ適用されるインデックスで使用できます。ほとんどの場合、ハッシュ結合はブロックネストループアルゴリズムより効率的です。ここに示すような結合は、次の方法で最適化できます:

```
SELECT *
FROM t1
JOIN t2
  ON t1.c1=t2.c1;

SELECT *
FROM t1
JOIN t2
  ON (t1.c1 = t2.c1 AND t1.c2 < t2.c2)
JOIN t3
  ON (t2.c1 = t3.c1)
```

ハッシュ結合は、デカルト積 (結合条件が指定されていない場合) にも使用できます。

[EXPLAIN FORMAT=TREE](#) または [EXPLAIN ANALYZE](#) を使用して、ハッシュ結合の最適化が特定のクエリーに使用されているかどうかを確認できます。(MySQL 8.0.20 以降では、[FORMAT=TREE](#) を省略して [EXPLAIN](#) を使用することもできます。)

ハッシュ結合で使用可能なメモリー量は、[join_buffer_size](#) の値によって制限されます。この量を超えるメモリーを必要とするハッシュ結合がディスク上で実行されます。ディスク上のハッシュ結合で使用できるディスクファイルの数は、[open_files_limit](#) によって制限されます。

MySQL 8.0.19 の時点では、MySQL 8.0.18 で導入された [hash_join](#) オプティマイザスイッチはサポートされなくなりました ([hash_join=on](#) は [optimizer_switch](#) の値の一部として表示されますが、設定しても効果はなくなります)。[HASH_JOIN](#) および [NO_HASH_JOIN](#) オプティマイザヒントもサポートされなくなりました。スイッチとヒントの両方が非推奨になりました。将来の MySQL リリースで削除される予定です。MySQL 8.0.18 以降では、[NO_BNL](#) オプティマイザスイッチを使用してハッシュ結合を無効にできます。

MySQL 8.0.20 以降では、ブロックのネストドロープは MySQL サーバーで使用されなくなり、クエリーに等価結合条件が含まれていない場合でも、ブロックのネストドロープが以前に使用された場合は常にハッシュ結合が使用されます。これは、内部非等価結合、準結合、アンチ結合、左外部結合および右外部結合に適用されます。[optimizer_switch](#) システム変数、[BNL](#) および [NO_BNL](#) オプティマイザヒントに対する [block_nested_loop](#) フラグは引き続きサポートされますが、以降はハッシュ結合の使用のみが制御されます。また、内部結合と外部結合 (準結合とアンチ結合を含む) の両方でバッチキーアクセス (BKA) を使用できるようになりました。BKA では、結合バッファメモリーが増分的に割り当てられるため、個々のクエリーで実際には解決に必要な大量のリソースを使用する必要はありません。内部結合の BKA は、MySQL 8.0.18 以降でのみサポートされています。

また、MySQL 8.0.20 は、以前のバージョンの MySQL で使用されていたエグゼキュータをイテレータエグゼキュータに置き換えます。この作業には、準結合として最適化されていない [IN](#) クエリーに対する [WHERE value IN \(SELECT column FROM table WHERE ...\)](#) 形式のクエリーを制御する古いインデックスサブクエリーエンジンの置換、および以前は古いエグゼキュータに依存していた同じ形式で実体化されたクエリーが含まれます。

詳細および例については、[セクション8.2.1.4「ハッシュ結合の最適化」](#)を参照してください。[Batched Key Access 結合](#)も参照してください。

- EXPLAIN ANALYZE ステートメント。新しい形式の [EXPLAIN](#) ステートメント [EXPLAIN ANALYZE](#) が MySQL 8.0.18 に実装され、クエリーの処理に使用されるイテレータごとに [SELECT](#) ステートメントの実行に関する拡張情報が [TREE](#) 形式で提供され、見積りコストをクエリーの実際のコストと比較できるようになりました。この情報には、起動コスト、合計コスト、このイテレータによって返された行数および実行されたループ数が含まれます。

MySQL 8.0.21 以降では、このステートメントは [FORMAT=TREE](#) 指定子もサポートしています。サポートされている形式は [TREE](#) のみです。

詳しくは[EXPLAIN ANALYZE による情報の取得](#),をご覧ください。

- クエリーキャスト注入。8.0.18 以降のバージョンでは、MySQL は、引数のデータ型と予想されるデータ型が一致しない式および条件内のクエリーアイテムツリーにキャスト操作を注入します。これはクエリー結果や実行速度には影響しませんが、以前のリリースの MySQL との下位互換性を維持しながら、クエリーを SQL 標準に準拠したものと同じように実行します。

このような暗黙的なキャストは、時間型 ([DATE](#), [DATETIME](#), [TIMESTAMP](#), [TIME](#)) と数値型 ([SMALLINT](#), [TINYINT](#), [MEDIUMINT](#), [INT/INTEGER](#), [BIGINT](#); [DECIMAL](#) / [NUMERIC](#); [FLOAT](#), [DOUBLE](#), [REAL](#); [BIT](#)) の間で実行されます。いずれかの標準数値比較演算子 ([=](#), [>=](#), [>](#), [<](#), [<=](#), [<>](#) / [!=](#), または [<=>](#)) を使用して実行されるものです。この

場合、まだ **DOUBLE** ではない値はキャストされます。キャストインジェクションは、**DATE** または **TIME** の値と **DATETIME** の値の比較に対しても実行されるようになりました。引数は **DATETIME** として必要に応じてキャストされます。

MySQL 8.0.21 以降、このようなキャストは、文字列型を他の型と比較するときにも実行されます。キャストされる文字列型には、**CHAR**、**VARCHAR**、**BINARY**、**VARBINARY**、**BLOB**、**TEXT**、**ENUM** および **SET** があります。文字列型の値を数値型または **YEAR** と比較する場合、文字列のキャストは **DOUBLE** になります。他の引数の型が **FLOAT**、**DOUBLE** または **REAL** でない場合は、**DOUBLE** にもキャストされます。文字列型を **DATETIME** または **TIMESTAMP** 値と比較する場合、文字列は **DATETIME** にキャストされ、文字列型を **DATE** と比較する場合、文字列は **DATE** にキャストされます。

次に示すように、**EXPLAIN ANALYZE**、**EXPLAIN FORMAT=JSON** または **EXPLAIN FORMAT=TREE** の出力を表示することで、特定のクエリーにキャストが注入されるタイミングを確認できます：

```
mysql> CREATE TABLE d (dt DATETIME, d DATE, t TIME);
Query OK, 0 rows affected (0.62 sec)

mysql> CREATE TABLE n (i INT, d DECIMAL, f FLOAT, dc DECIMAL);
Query OK, 0 rows affected (0.51 sec)

mysql> CREATE TABLE s (c CHAR(25), vc VARCHAR(25),
->  bn BINARY(50), vb VARBINARY(50), b BLOB, t TEXT,
->  e ENUM('a', 'b', 'c'), se SET('x', 'y', 'z'));
Query OK, 0 rows affected (0.50 sec)

mysql> EXPLAIN FORMAT=TREE SELECT * from d JOIN n ON d.dt = n.i;G
***** 1. row *****
EXPLAIN: -> Inner hash join (cast(d.dt as double) = cast(n.i as double))
(cost=0.70 rows=1)
-> Table scan on n (cost=0.35 rows=1)
-> Hash
-> Table scan on d (cost=0.35 rows=1)

mysql> EXPLAIN FORMAT=TREE SELECT * from s JOIN d ON d.dt = s.c;G
***** 1. row *****
EXPLAIN: -> Inner hash join (d.dt = cast(s.c as datetime(6))) (cost=0.72 rows=1)
-> Table scan on d (cost=0.37 rows=1)
-> Hash
-> Table scan on s (cost=0.35 rows=1)

1 row in set (0.01 sec)

mysql> EXPLAIN FORMAT=TREE SELECT * from n JOIN s ON n.d = s.c;G
***** 1. row *****
EXPLAIN: -> Inner hash join (cast(n.d as double) = cast(s.c as double)) (cost=0.70 rows=1)
-> Table scan on s (cost=0.35 rows=1)
-> Hash
-> Table scan on n (cost=0.35 rows=1)

1 row in set (0.00 sec)
```

このようなキャストは、**EXPLAIN [FORMAT=TRADITIONAL]** を実行することでも確認できます。この場合、**EXPLAIN** ステートメントの実行後に **SHOW WARNINGS** を発行する必要があります。

- **TIMESTAMP** および **DATETIME** のタイムゾーンサポート。 MySQL 8.0.19 の時点では、サーバーは日時 (**TIMESTAMP** および **DATETIME**) 値が挿入されたタイムゾーンオフセットを受け入れます。このオフセットは、**time_zone** システム変数の設定時に採用されたフォーマットと同じフォーマットを使用しますが、オフセットの時間部分が 10 未満で、'-00:00' が許可されていない場合に先行ゼロが必要になる点が異なります。タイムゾーンオフセットを含む日時リテラルの例には、'2019-12-11 10:40:30-05:00'、'2003-04-14 03:30:00+10:00' および '2020-01-01 15:35:45+05:30' があります。

日時値を選択する場合、タイムゾーンオフセットは表示されません。

タイムゾーンオフセットを組み込む日時リテラルは、プリペアドコンパイルされたステートメントのパラメータ値として使用できます。

この作業の一環として、**time_zone** システム変数の設定に使用される値も、-14:00 から +14:00 までの範囲に制限されるようになりました。(MySQL タイムゾーンテーブルがロードされている場合は、**time_zone** に、'EST'、'Posix/

`Australia/Brisbane`、および `Europe/Stockholm` といった名前の値を割り当てることができます。タイムゾーンテーブルへの移入を参照してください。

詳細および例は、[セクション 5.1.15 「MySQL Server でのタイムゾーンのサポート」](#) および [セクション 11.2.2 「DATE、DATETIME、および TIMESTAMP 型」](#) を参照してください。

- JSON スキーマの CHECK 制約の失敗に関する正確な情報。 `JSON_SCHEMA_VALID()` を使用して CHECK 制約を指定する場合、MySQL 8.0.19 以降では、このような制約の失敗の理由に関する正確な情報が提供されます。

例および詳細は、`JSON_SCHEMA_VALID()` および CHECK 制約を参照してください。 [セクション 13.1.20.6 「CHECK 制約」](#) も参照してください。

- ON DUPLICATE KEY UPDATE を使用した行およびカラムのエイリアス。 MySQL 8.0.19 以降では、エイリアスを使用して、挿入する行およびそのカラム (オプション) を参照できます。カラム `a` および `b` を含むテーブル `t` で、次の INSERT ステートメントを考えてみます:

```
INSERT INTO t SET a=9,b=5
ON DUPLICATE KEY UPDATE a=VALUES(a)+VALUES(b);
```

新しい行にエイリアス `new` を使用し、場合によっては、この行のカラムにエイリアス `m` および `n` を使用すると、INSERT ステートメントを様々な方法でリライトできます。次に例をいくつか示します:

```
INSERT INTO t SET a=9,b=5 AS new
ON DUPLICATE KEY UPDATE a=new.a+new.b;

INSERT INTO t VALUES(9,5) AS new
ON DUPLICATE KEY UPDATE a=new.a+new.b;

INSERT INTO t SET a=9,b=5 AS new(m,n)
ON DUPLICATE KEY UPDATE a=m+n;

INSERT INTO t VALUES(9,5) AS new(m,n)
ON DUPLICATE KEY UPDATE a=m+n;
```

詳細および例については、[セクション 13.2.6.2 「INSERT ... ON DUPLICATE KEY UPDATE ステートメント」](#) を参照してください。

- SQL 標準の明示的なテーブル句およびテーブル値コンストラクタ。 SQL 標準に従って、テーブル値コンストラクタおよび明示的なテーブル句が追加されました。これらは、それぞれ TABLE ステートメントおよび VALUES ステートメントとして MySQL 8.0.19 に実装されます。

TABLE ステートメントの形式は `TABLE table_name` で、`SELECT * FROM table_name` と同等です。ORDER BY 句および LIMIT 句 (後者はオプションで OFFSET で併用可) はサポートされますが、個々のテーブルのカラムを選択することはできません。TABLE は、同等の SELECT ステートメントを使用できる場所であればどこでも使用できます。これには、JOIN、UNION、INSERT ... SELECT、REPLACE、CREATE TABLE ... SELECT ステートメントおよびサブクエリーが含まれます。例:

- `TABLE t1 UNION TABLE t2` は `SELECT * FROM t1 UNION SELECT * FROM t2` と同等です
- `CREATE TABLE t2 TABLE t1` は `CREATE TABLE t2 SELECT * FROM t1` と同等です
- `SELECT a FROM t1 WHERE b > ANY (TABLE t2)` は、`SELECT a FROM t1 WHERE b > ANY (SELECT * FROM t2)` と同等です。

VALUES は、INSERT、REPLACE または SELECT ステートメントにテーブルの値を指定するために使用でき、VALUES キーワードの後にカンマで区切られた一連の行コンストラクタ (ROW()) が続きます。たとえば、SQL 標準に準拠した `INSERT INTO t1 VALUES ROW(1,2,3), ROW(4,5,6), ROW(7,8,9)` というステートメントは、MySQL 固有の `INSERT INTO t1 VALUES (1,2,3), (4,5,6), (7,8,9)` と同等です。テーブルと同じように、VALUES テーブルの値コンストラクタから選択することもできます (これには、テーブルのエイリアスを指定

しなければならないことに注意)。これは他の `SELECT` と同じように使えます。JOIN、UNION およびサブクエリーを含みます。

`TABLE` および `VALUES` の詳細とその使用例は、このドキュメントの次のセクションを参照してください:

- [セクション13.2.12「TABLE ステートメント」](#)
- [セクション13.2.14「VALUES ステートメント」](#)
- [セクション13.1.20.4「CREATE TABLE ... SELECT ステートメント」](#)
- [セクション13.2.6.1「INSERT ... SELECT ステートメント」](#)
- [セクション13.2.10.2「JOIN 句」](#)
- [セクション13.2.11「サブクエリー」](#)
- [セクション13.2.10.3「UNION 句」](#)
- `FORCE INDEX`、`IGNORE INDEX` のオプティマイザヒント。MySQL 8.0 では、[セクション8.9.4「インデックスヒント」](#) で説明されている従来のインデックスヒントと同様に機能するインデックスレベルのオプティマイザヒントが導入されています。新しいヒントとそれに相当する `FORCE INDEX` または `IGNORE INDEX` のヒントを次に示します:
 - `GROUP_INDEX: FORCE INDEX FOR GROUP BY` と同等
`NO_GROUP_INDEX: IGNORE INDEX FOR GROUP BY` と同等
 - `JOIN_INDEX: FORCE INDEX FOR JOIN` と同等
`NO_JOIN_INDEX: IGNORE INDEX FOR JOIN` と同等
 - `ORDER_INDEX: FORCE INDEX FOR ORDER BY` と同等
`NO_ORDER_INDEX: IGNORE INDEX FOR ORDER BY` と同等
 - `INDEX: GROUP_INDEX` に `JOIN_INDEX` と `ORDER_INDEX` を加えたものと同じです。修飾子のない `FORCE INDEX` と同等です
`NO_INDEX: NO_GROUP_INDEX` に `NO_JOIN_INDEX` と `NO_ORDER_INDEX` を加えたものと同じです。修飾子のない `IGNORE INDEX` と同等です

たとえば、次の 2 つのクエリーは同等です。

```
SELECT a FROM t1 FORCE INDEX (i_a) FOR JOIN WHERE a=1 AND b=2;
SELECT /*+ JOIN_INDEX(t1 i_a) */ a FROM t1 WHERE a=1 AND b=2;
```

前述のオプティマイザヒントは、既存のインデックスレベルのオプティマイザヒントと同じ構文および使用方法の基本ルールに従います。

これらのオプティマイザヒントは、将来の MySQL リリースで非推奨になり、その後 MySQL から削除する予定の `FORCE INDEX` および `IGNORE INDEX` を置き換えることを目的としています。 `USE INDEX` に同等の単一のものには実装されません。かわりに、`NO_INDEX`、`NO_JOIN_INDEX`、`NO_GROUP_INDEX` または `NO_ORDER_INDEX` のいずれかを使用して同じ効果を得ることができます。

詳細および使用例は、[インデックスレベルのオプティマイザヒント](#) を参照してください。

- `JSON_VALUE()` 関数。MySQL 8.0.21 には、`JSON` カラムのインデックス付けを簡略化するための新しい関数 `JSON_VALUE()` が実装されています。最も基本的な形式では、引数として `JSON` ドキュメントおよびそのドキュメント内の単一の値を指す `JSON` パスを取り、オプションで `RETURNING` キーワードを使用して戻り型を指定できます。 `JSON_VALUE(json_doc, path RETURNING type)` は次と同等です:

```
CAST(
  JSON_UNQUOTE(JSON_EXTRACT(json_doc, path))
```



```
AS type
);
```

`JSON_TABLE()` で採用されている場合と同様に、`ON EMPTY`、`ON ERROR`、またはその両方の句を指定することもできます。

`JSON_VALUE()` を使用して、次のように `JSON` カラムの式にインデックスを作成できます:

```
CREATE TABLE t1(
  j JSON,
  INDEX i1 ((JSON_VALUE(j, '$.id' RETURNING UNSIGNED)))
);
INSERT INTO t1 VALUES ROW(('{ "id": "123", "name": "shoes", "price": "49.95" }');
```

次に示すように、この式を使用するクエリーではインデックスを使用できます:

```
SELECT name, price FROM t1
WHERE JSON_VALUE(j, '$.id' RETURNING UNSIGNED) = 123;
```

多くの場合、これは、`JSON` カラムから生成カラムを作成し、生成カラムにインデックスを作成するよりも簡単です。

詳細および例は、`JSON_VALUE()` の説明を参照してください。

- **ユーザーコメントとユーザー属性** MySQL 8.0.21 では、ユーザーアカウントの作成または更新時にユーザーコメントおよびユーザー属性を設定する機能が導入されています。ユーザーコメントは、`CREATE USER` ステートメントまたは `ALTER USER` ステートメントで使用される `COMMENT` 句に引数として渡される任意のテキストで構成されます。ユーザー属性は、これらのステートメントのいずれかで使用される `ATTRIBUTE` 句に引数として渡される `JSON` オブジェクトの形式のデータで構成されます。属性には、`JSON` オブジェクト表記法の有効なキーと値のペアを含めることができます。単一の `CREATE USER` ステートメントまたは `ALTER USER` ステートメントで使用できるのは、`COMMENT` または `ATTRIBUTE` のいずれかのみです。

ユーザーコメントとユーザー属性は `JSON` オブジェクトとして内部的に格納され、コメントテキストはキーとして `comment` を持つ要素の値として格納されます。この情報は、`INFORMATION_SCHEMA.USER_ATTRIBUTES` テーブルの `ATTRIBUTE` カラムから取得できます。`JSON` 形式であるため、MySQL `JSON` 関数および演算子を使用して内容を解析できます ([セクション12.18「JSON 関数」](#)を参照)。ユーザー属性に対する後続の変更は、`JSON_MERGE_PATCH()` 関数を使用する場合と同様に現在の値とマージされます。

例:

```
mysql> CREATE USER 'mary'@'localhost' COMMENT 'This is Mary Smith's account';
Query OK, 0 rows affected (0.33 sec)

mysql> ALTER USER 'mary'@'localhost'
-> ATTRIBUTE '{"fname":"Mary", "lname":"Smith"}';
Query OK, 0 rows affected (0.14 sec)

mysql> ALTER USER 'mary'@'localhost'
-> ATTRIBUTE '{"email":"mary.smith@example.com"}';
Query OK, 0 rows affected (0.12 sec)

mysql> SELECT
-> USER,
-> HOST,
-> ATTRIBUTE->"$.fname" AS 'First Name',
-> ATTRIBUTE->"$.lname" AS 'Last Name',
-> ATTRIBUTE->"$.email" AS 'Email',
-> ATTRIBUTE->"$.comment" AS 'Comment'
-> FROM INFORMATION_SCHEMA.USER_ATTRIBUTES
-> WHERE USER='mary' AND HOST='localhost'G
***** 1. row *****
USER: mary
HOST: localhost
First Name: Mary
Last Name: Smith
Email: mary.smith@example.com
Comment: This is Mary Smith's account
```


1 row in set (0.00 sec)

詳細および例は、[セクション13.7.1.3「CREATE USER ステートメント」](#)、[セクション13.7.1.1「ALTER USER ステートメント」](#) および [セクション26.46「INFORMATION_SCHEMA.USER_ATTRIBUTES テーブル」](#) を参照してください。

- 新しい optimizer_switch フラグ。 MySQL 8.0.21 では、次のリストに示すように、optimizer_switch システム変数に 2 つの新しいフラグが追加されます:

- prefer_ordering_index フラグ

デフォルトでは、MySQL は LIMIT 句を持つ ORDER BY または GROUP BY クエリーに対して順序付けされたインデックスを使用しようとします。オプティマイザは、この結果、実行速度が速くなると判断します。このようなクエリーに対して異なる最適化を選択する方が実際にパフォーマンスが向上する場合がありますため、prefer_ordering_index フラグを off に設定することで、この最適化を無効にできるようになりました。

このフラグのデフォルト値は on です。

- subquery_to_derived フラグ

このフラグが on に設定されている場合、オプティマイザは適格なスカラーサブクエリーを導出テーブルの結合に変換します。たとえば、クエリー `SELECT * FROM t1 WHERE t1.a > (SELECT COUNT(a) FROM t2)` は `SELECT t1.a FROM t1 JOIN (SELECT COUNT(t2.a) AS c FROM t2) AS d WHERE t1.a > d.c` としてリライトされます。

この最適化は、SELECT, WHERE, JOIN 句または HAVING 句の一部であるサブクエリーに適用できます。1 つ以上の集計関数は含まれますが、GROUP BY 句は含まれません。相関関係はなく、非決定的関数は使用されません。

最適化は、IN, NOT IN, EXISTS または NOT EXISTS の引数であり、GROUP BY を含まないテーブルサブクエリーにも適用できます。たとえば、クエリー `SELECT * FROM t1 WHERE t1.b < 0 OR t1.a IN (SELECT t2.a + 1 FROM t2)` は `SELECT a, b FROM t1 LEFT JOIN (SELECT DISTINCT 1 AS e1, t2.a AS e2 FROM t2) d ON t1.a + 1 = d.e2 WHERE t1.b < 0 OR d.e1 IS NOT NULL` としてリライトされます。

この最適化は、ほとんどの場合に顕著なパフォーマンスの向上をもたらさないため、通常は無効になっています。フラグはデフォルトで off に設定されます。

詳細は、[セクション8.9.2「切り替え可能な最適化」](#)を参照してください。 [セクション8.2.1.19「LIMIT クエリーの最適化」](#)、[セクション8.2.2.1「準結合変換による IN および EXISTS サブクエリー述語の最適化」](#) および [セクション8.2.2.4「マージまたは実体化を使用した導出テーブル、ビュー参照および共通テーブル式の最適化」](#) も参照してください。

- XML の拡張機能。 MySQL 8.0.21 では、LOAD XML ステートメントで XML の CDATA セクションのインポートがサポートされるようになりました。
- YEAR 型へのキャストがサポートされるようになりました。 MySQL 8.0.22 以降、サーバーは YEAR へのキャストを許可します。CAST() 関数と CONVERT() 関数の両方で、単一桁、2 桁および 4 桁の YEAR 値がサポートされています。1 桁および 2 桁の値の場合、使用できる範囲は 0-99 です。4 桁の値は 1901-2155 の範囲内である必要があります。YEAR は、JSON_VALUE() 関数の戻り型としても使用できます。この関数は 4 桁の年のみをサポートします。

文字列、日時および浮動小数点値はすべて YEAR にキャストできます。GEOMETRY 値の YEAR へのキャストはサポートされていません。

変換ルールを含む詳細は、CONVERT() 関数の説明を参照してください。

- TIMESTAMP 値の UTC としての取得。 MySQL 8.0.22 以降では、CAST(value AT TIME_ZONE specifier AS DATETIME) を使用した、取得時のシステムタイムゾーンから UTC DATETIME への TIMESTAMP カラム値の変換がサポートされています。ここで、指定子は [INTERVAL] '+00:00' または 'UTC' のいずれかです。キャストによって

返される `DATETIME` 値の精度は、必要に応じて小数点以下 6 桁まで指定できます。 `ARRAY` キーワードは、この構成ではサポートされていません。

タイムゾーンオフセットを使用してテーブルに挿入された `TIMESTAMP` 値もサポートされます。 `AT TIME ZONE` の使用は、`CONVERT()` またはその他の MySQL 関数や構造体ではサポートされていません。

詳細および例は、`CAST()` 関数の説明を参照してください。

- ダンプファイル出力の同期。 MySQL 8.0.22 以降では、`SELECT INTO DUMPFILE` および `SELECT INTO OUTFILE` ステートメントによるファイルへの書き込み時に定期的な同期がサポートされます。これを有効にするには、`select_into_disk_sync` システム変数を `ON` に設定します。書き込みバッファのサイズは、`select_into_buffer_size` に設定された値によって決まります。デフォルトは 131072 (2^{17}) バイトです。

また、ディスクへの同期後のオプションの遅延は、`select_into_disk_sync_delay` を使用して設定できます。デフォルトは遅延なし (0 ミリ秒) です。

詳細は、この項目で前述した変数の説明を参照してください。

- ステートメントの準備の単一化。 MySQL 8.0.22 の時点では、プリペアドステートメントは、実行されるたびに一度ではなく、最初に一度だけ準備されます。これは、`PREPARE` の実行時に行われます。これは、ストアードプロシージャ内のすべてのステートメントにも当てはまります。このステートメントは、ストアードプロシージャが最初に実行されたときに一度準備されます。

この変更の結果、プリペアドステートメントで使用される動的パラメータが解決される方法も、次に示す方法で変更されます:

- プリペアドステートメントのパラメータには、ステートメントの準備時にデータ型が割り当てられます。ステートメントが再準備されないかぎり、ステートメントの後続の実行ごとに型が保持されます。次を参照してください。

準備されたステートメント内の特定のパラメータまたはユーザー変数に別のデータ型を使用して最初の実行後にステートメントを実行すると、ステートメントが再準備される可能性があります。このため、準備されたステートメントを再実行するときは、指定されたパラメータに同じデータ型を使用することをお勧めします。

- ウィンドウ関数を使用する次の構造体は、SQL 標準に合わせるために受け入れられなくなりました:

- `NTILE(NULL)`
- `NTH_VALUE(expr, NULL)`
- `LEAD(expr, nn)` および `LAG(expr, nn)` (`nn` は負数)

これにより、SQL 標準への準拠が容易になります。詳細は、個々の関数の説明を参照してください。

- プリペアドステートメント内で参照されるユーザー変数のデータ型は、ステートメントの準備時に決定されるようになりました。この型は、ステートメントの後続の実行ごとに保持されます。
- ストアドプロシージャ内で実行されるステートメントによって参照されるユーザー変数のデータ型は、ステートメントの初回実行時に決定されるようになりました。この型は、格納されているストアードプロシージャの後続の起動でも保持されます。
- `SELECT expr1, expr2, ... FROM table ORDER BY ?` 形式のプリペアドステートメントを実行するときに、パラメータに整数値 `N` を渡すと、選択リストの `Nth` 式による結果の順序付けが行われなくなります。 `ORDER BY constant` で期待される通りには、結果が順序付けされなくなります。

プリペアドステートメントとして、またはストアードプロシージャ内で使用されるステートメントとして最初に一度だけ準備することにより、ステートメントのパフォーマンスが向上します。これは、繰返し準備の追加コストが削減されるためです。これにより、MySQL での多数の問題の原因となった準備構造の複数のロールバックを回避することもできます。

詳細は、[セクション 13.5.1「PREPARE ステートメント」](#)を参照してください。

- LEFT JOIN 処理としての RIGHT JOIN. MySQL 8.0.22 点では、サーバーは [RIGHT JOIN](#) のすべてのインスタンスを [LEFT JOIN](#) として内部的に処理することで、パース時に完全な変換が行われないいくつかの特別なケースを排除しました。
- 導出条件プッシュダウン最適化. MySQL 8.0.22 (以降) は、実体化導出テーブルを持つクエリーの導出条件プッシュダウンを実装します。 [SELECT * FROM \(SELECT i, j FROM t1\) AS dt WHERE i > constant](#) などのクエリーでは、多くの場合、外部 [WHERE](#) 条件を導出テーブルにプッシュできるようになりました (この場合、 [SELECT * FROM \(SELECT i, j FROM t1 WHERE i > constant\) AS dt](#) になります)。

以前は、導出テーブルが実体化されてマージされていない場合、MySQL はテーブル全体を実体化し、[WHERE](#) 条件で行を修飾していました。導出条件プッシュダウン最適化を使用して [WHERE](#) 条件をサブクエリーに移動すると、多くの場合、処理が必要な行数が減り、クエリーの実行に必要な時間が短縮されます。

導出テーブルで集計関数またはウィンドウ関数が使用されていない場合は、外部 [WHERE](#) 条件を実体化導出テーブルに直接プッシュダウンできます。導出テーブルに [GROUP BY](#) があり、ウィンドウ関数を使用しない場合、外部 [WHERE](#) 条件を [HAVING](#) 条件として導出テーブルにプッシュダウンできます。導出テーブルがウィンドウ関数を使用し、外部 [WHERE](#) がウィンドウ関数の [PARTITION](#) 句で使用されるカラムを参照している場合は、[WHERE](#) 条件をプッシュダウンすることもできます。

導出条件プッシュダウンは、[optimizer_switch](#) システム変数 [derived_condition_pushdown](#) フラグで示されるように、デフォルトで有効になっています。MySQL 8.0.22 で追加されたフラグは、デフォルトで [on](#) に設定されています。特定のクエリーの最適化を無効にするには、[NO_DERIVED_CONDITION_PUSHDOWN](#) オプティマイザヒント (こちら MySQL 8.0.22 で追加) を使用できます。 [derived_condition_pushdown](#) が [off](#) に設定されているために最適化が無効になっている場合は、[DERIVED_CONDITION_PUSHDOWN](#) を使用して特定のクエリーに対して最適化を有効にできます。

導出条件プッシュダウン最適化は、[UNION](#) 句または [LIMIT](#) 句を含む導出テーブルには使用できません。また、サブクエリーを使用する条件自体はプッシュダウンできず、外部結合の内部テーブルでもある導出テーブルに [WHERE](#) 条件をプッシュダウンすることもできません。追加情報および例については、[セクション8.2.2.5「導出条件プッシュダウン最適化」](#) を参照してください。

- MySQL 付与テーブルでの非ロック読み取り. MySQL 8.0.22 では、MySQL 付与テーブルに対する同時 DML 操作および DDL 操作を許可するために、MySQL 付与テーブルに対して以前に行ロックを取得した読取り操作は、非ロック読取りとして実行されます。

MySQL 付与テーブルで非ロック読み取りとして実行されるようになった操作には、次のものがあります：

- 任意のトランザクション分離レベルを使用して、結合リストおよびサブクエリー ([SELECT ... FOR SHARE](#) ステートメントを含む) を介して付与テーブルからデータを読み取る [SELECT](#) ステートメントおよびその他の読取り専用ステートメント。
- 任意のトランザクション分離レベルを使用して (結合リストまたはサブクエリーを介して) 付与テーブルからデータを読み取り、変更を伴わない DML 操作。

追加情報については [テーブル同時実行性の付与](#) を参照してください。

MySQL 8.0 で非推奨となった機能

次の機能は MySQL 8.0 では非推奨であり、将来のシリーズで削除される可能性があります。ほかのオプションがあれば、それを利用するためにアプリケーションを更新する必要があります。

上位の MySQL シリーズで削除された MySQL 8.0 で非推奨になった機能を使用するアプリケーションの場合、MySQL 8.0 ソースから上位シリーズのレプリカにレプリケートするとステートメントが失敗するか、ソースとレプリカに異なる影響を与える可能性があります。このような問題を回避するには、8.0 で非推奨になった機能を使用するアプリケーションを改訂して回避し、可能な場合は代替機能を使用する必要があります。

- [utf8mb3](#) 文字セットは非推奨です。かわりに [utf8mb4](#) を使用してください。
- [caching_sha2_password](#) は MySQL 8.0 のデフォルトの認証プラグインであり、[sha256_password](#) 認証プラグインの機能のスーパーセットを提供するため、[sha256_password](#) は非推奨になりました。将来のバージョンの MySQL で削除される予定です。 [sha256_password](#) を使用して認証する MySQL アカントは、かわりに [caching_sha2_password](#) を使用するように移行する必要があります。

- `validate_password` プラグインは、コンポーネントインフラストラクチャを使う形式で再実装されました。プラグイン形式の `validate_password` は引き続き使用可能ですが、非推奨になりました。将来のバージョンの MySQL で削除される予定です。プラグインを使用する MySQL インストールでは、かわりにコンポーネントの使用に移行する必要があります。 [セクション6.4.3.3「パスワード検証コンポーネントへの移行」](#) を参照してください。
- `ALTER TABLESPACE` および `DROP TABLESPACE` ステートメントの `ENGINE` 句は非推奨になりました。
- `PAD_CHAR_TO_FULL_LENGTH` SQL モードは非推奨です。
- `AUTO_INCREMENT` のサポートは、`FLOAT` 型および `DOUBLE` 型のカラム (およびシノニム) では非推奨です。このようなカラムから `AUTO_INCREMENT` 属性を削除するか、整数型に変換することを検討してください。
- `UNSIGNED` 属性は、`FLOAT`、`DOUBLE` および `DECIMAL` (およびシノニム) 型のカラムでは非推奨です。このようなカラムには、かわりに単純な `CHECK` 制約の使用を検討してください。
- `FLOAT` および `DOUBLE` (およびシノニム) 型のカラムの桁数を指定するための `FLOAT(M,D)` および `DOUBLE(M,D)` 構文は、非標準 MySQL 拡張機能です。この構文は非推奨です。
- `ZEROFILL` 属性は、整数データ型の表示幅属性と同様に、数値データ型では非推奨です。これらの属性の効果を生成する別の方法の使用を検討してください。たとえば、アプリケーションでは、`LPAD()` 関数を使用して、必要な幅まで数値をゼロ埋めたり、書式設定された数値を `CHAR` カラムに格納したりできます。
- 文字列データ型の場合、`BINARY` 属性は、カラム文字セット (またはカラム文字セットが指定されていない場合はテーブルのデフォルト文字セット) のバイナリ (`_bin`) 照合順序を指定するための短縮形である非標準の MySQL 拡張機能です。MySQL 8.0 では、`utf8mb4` 文字セットに複数の `_bin` 照合順序があるため、この `BINARY` の非標準の使用はあいまいです。このため、`BINARY` 属性は非推奨になりました。将来のバージョンの MySQL ではサポートされなくなる予定です。かわりに、明示的な `_bin` 照合を使用するようにアプリケーションを調整する必要があります。

`BINARY` を使用してデータ型または文字セットを指定する方法は変わりません。

- 標準の SQL `AND`、`OR` および `NOT` 演算子のシノニムである非標準の C 形式の `&&`、`||` および `!` 演算子は、それぞれ非推奨になりました。非標準演算子を使用するアプリケーションは、標準演算子を使用するように調整する必要があります。

注記

`PIPES_AS_CONCAT` SQL モードが有効になっていないがぎり、`||` の使用は非推奨です。その場合、`||` は SQL 標準の文字列連結演算子を示します。

- `JSON_MERGE()` 関数は非推奨です。かわりに `JSON_MERGE_PRESERVE()` を使用してください。
- `SQL_CALC_FOUND_ROWS` クエリー修飾子および付随する `FOUND_ROWS()` 関数は非推奨です。代替戦略の詳細は、`FOUND_ROWS()` の説明を参照してください。
- `CREATE TEMPORARY TABLE` での `TABLESPACE = innodb_file_per_table` 句および `TABLESPACE = innodb_temporary` 句のサポートは、MySQL 8.0.13 で非推奨になりました。
- `SELECT` ステートメントの場合、`FROM` の後に `INTO` 句を使用することはできませんが、`SELECT` の最後には使用できません (MySQL 8.0.20 の時点では非推奨です)。ステートメントの最後に `INTO` を配置することをお勧めしません。

`UNION` ステートメントの場合、`INTO` を含む次の 2 つのバリエーションは、MySQL 8.0.20 で非推奨になりました:

- クエリー式の後続のクエリーブロックでは、`FROM` の前に `INTO` を使用します。
- クエリー式のカッコで囲まれた後続ブロックでは、`FROM` に対する相対位置に関係なく、`INTO` を使用します。

[セクション13.2.10.1「SELECT ... INTO ステートメント」](#) および [セクション13.2.10.3「UNION 句」](#) を参照してください。

- `FLUSH HOSTS` は、MySQL 8.0.23 の時点では非推奨です。代わりに、パフォーマンススキーマ `host_cache` テーブルを切り捨てます:

```
TRUNCATE TABLE performance_schema.host_cache;
```


TRUNCATE TABLE 操作には、テーブルに対する DROP 権限が必要です。

- `mysql` システムスキーマ内のシステムテーブルおよび他のスキーマ内のオブジェクトをアップグレードする機能が MySQL サーバーに移動されたため、`mysql_upgrade` クライアントは非推奨になりました。 [セクション 2.11.3 「MySQL のアップグレードプロセスの内容」](#) を参照してください。
- `--no-dd-upgrade` サーバーオプションは非推奨です。 データディクショナリおよびサーバーのアップグレード動作をより細かく制御できる `--upgrade` オプションに置き換えられています。
- データディレクトリが作成され、MySQL のバージョン番号の格納に使用される `mysql_upgrade_info` ファイルは非推奨になりました。 将来のバージョンの MySQL で削除される予定です。
- `relay_log_info_file` システム変数および `--master-info-file` オプションは非推奨になりました。 以前は、これらは `relay_log_info_repository=FILE` および `master_info_repository=FILE` の設定時にリレーログ情報ログおよびソース情報ログの名前を指定するために使用されていましたが、これらの設定は非推奨になりました。 リレーログ情報ログおよびソース情報ログのファイルの使用は、MySQL 8.0 のデフォルトであるクラッシュセーフレプリカテーブルに置き換えられました。
- `max_length_for_sort_data` システム変数は、オプティマイザの変更によって廃止され、効果がないため、非推奨になりました。
- サーバーへの接続を圧縮するためのこれらのレガシーパラメータは非推奨です: `--compress` クライアントのコマンド行オプション、`mysql_options()` C API 関数の `MYSQL_OPT_COMPRESS` オプション、`slave_compressed_protocol` システム変数。 かわりに使用するパラメータの詳細は、 [セクション 4.2.8 「接続圧縮制御」](#) を参照してください。
- `MYSQL_PWD` 環境変数を使用した MySQL パスワードの指定は非推奨です。
- `VALUES()` を使用した `INSERT ... ON DUPLICATE KEY UPDATE` の新しい行値へのアクセスは、MySQL 8.0.20 では非推奨です。 かわりに、新しい行とカラムにエイリアスを使用します。
- `JSON_TABLE()` の呼び出し時に `ON EMPTY` の前に `ON ERROR` を指定することは SQL 標準に反しているため、この構文は MySQL で非推奨になりました。 MySQL 8.0.20 以降では、試行するたびにサーバーによって警告が出力されます。 単一の `JSON_TABLE()` 起動でこれらの句の両方を指定する場合は、`ON EMPTY` が最初に使用されていることを確認してください。
- インデックス接頭辞を持つカラムは、テーブルパーティション化キーの一部としてサポートされるいません。 以前は、パーティションテーブルの作成、変更またはアップグレード時に許可されていましたが、テーブルパーティション化関数によって除外され、これがサーバーによって警告を発することはありませんでした。 以前許可されていた動作は非推奨になり、将来のバージョンの MySQL で削除される (パーティション化キーでこのようなカラムを使用すると、`CREATE TABLE` または `ALTER TABLE` ステートメントが拒否される) 可能性があります。

MySQL 8.0.21 では、インデックス接頭辞を使用するカラムがパーティション化キーの一部として指定されるたびに、そのようなカラムごとに警告が生成されます。 提案されたパーティション化キーのすべてのカラムにインデックス接頭辞があるために `CREATE TABLE` または `ALTER TABLE` ステートメントが拒否されると、結果のエラーに拒否の正確な理由が示されるようになりました。 どちらの場合も、これには、空の `PARTITION BY KEY()` 句を使用して、パーティション化関数で使われるカラムがテーブルの主キーのカラムとして暗黙的に定義されるケースが含まれます。

詳細および例については、 [カラムインデックス接頭辞はキーパーティション化ではサポートされていません](#) を参照してください。
- InnoDB memcached プラグインは、MySQL 8.0.22 の時点では非推奨です。 将来のバージョンの MySQL ではサポートされなくなる予定です。

MySQL 8.0 で削除された機能

次の項目は廃止され、MySQL 8.0 で削除されました。 ほかのオプションがあれば、それを利用するためにアプリケーションを更新する必要があります。

MySQL 8.0 で削除された機能を使用する MySQL 5.7 アプリケーションの場合、MySQL 5.7 ソースから MySQL 8.0 レプリカにレプリケートするとステートメントが失敗するか、ソースとレプリカに異なる影響を与える可能性があります。

す。このような問題を回避するには、MySQL 8.0 で削除された機能を使用するアプリケーションを改訂して回避し、可能な場合は代替機能を使用する必要があります。

- `innodb_locks_unsafe_for_binlog` システム変数が削除されました。 `READ COMMITTED` 分離レベルは、同様の機能を提供します。
- MySQL 8.0.0 で導入された `information_schema_stats` 変数は、MySQL 8.0.3 で `information_schema_stats_expiry` によって削除および置換されました。

`information_schema_stats_expiry` は、キャッシュされた `INFORMATION_SCHEMA` テーブル統計の有効期限設定を定義します。詳細は、[セクション8.2.3「INFORMATION_SCHEMA クエリーの最適化」](#)を参照してください。

- 廃止された `InnoDB` システムテーブルに関連するコードは、MySQL 8.0.3 で削除されました。 `InnoDB` システムテーブルに基づく `INFORMATION_SCHEMA` ビューは、データディクショナリテーブルの内部システムビューに置き換えられました。影響を受ける `InnoDB INFORMATION_SCHEMA` ビューの名前が変更されました:

表 1.1 名前が変更された `InnoDB` 情報スキーマビュー

旧名称	新規名
<code>INNODB_SYS_COLUMNS</code>	<code>INNODB_COLUMNS</code>
<code>INNODB_SYS_DATAFILES</code>	<code>INNODB_DATAFILES</code>
<code>INNODB_SYS_FIELDS</code>	<code>INNODB_FIELDS</code>
<code>INNODB_SYS_FOREIGN</code>	<code>INNODB_FOREIGN</code>
<code>INNODB_SYS_FOREIGN_COLS</code>	<code>INNODB_FOREIGN_COLS</code>
<code>INNODB_SYS_INDEXES</code>	<code>INNODB_INDEXES</code>
<code>INNODB_SYS_TABLES</code>	<code>INNODB_TABLES</code>
<code>INNODB_SYS_TABLESPACES</code>	<code>INNODB_TABLESPACES</code>
<code>INNODB_SYS_TABLESTATS</code>	<code>INNODB_TABLESTATS</code>
<code>INNODB_SYS_VIRTUAL</code>	<code>INNODB_VIRTUAL</code>

MySQL 8.0.3 以上にアップグレードした後、以前の `InnoDB INFORMATION_SCHEMA` ビュー名を参照するスクリプトを更新します。

- アカウント管理に関連する次の機能が削除されました:
 - `GRANT` を使用したユーザーの作成。かわりに、`CREATE USER` を使用してください。この変更により、`NO_AUTO_CREATE_USER` SQL モードが `GRANT` ステートメントで不要になったため、これも削除されました。オプションファイルの `sql_mode` オプションにこの値の記述があることによって `mysqld` の起動ができない場合は、サーバーログにエラーが書き込まれるようになります。
 - `GRANT` を使用した権限割当て以外のアカウントプロパティの変更。これには、認証、SSL およびリソース制限のプロパティが含まれます。かわりに、`CREATE USER` を使用してアカウント作成時にこのようなプロパティを設定するか、後で `ALTER USER` を使用して変更します。
 - `CREATE USER` および `GRANT` の `IDENTIFIED BY PASSWORD 'auth_string'` 構文。かわりに、`IDENTIFIED WITH auth_plugin AS 'auth_string' for CREATE USER and ALTER USER` を使用します。ここで、`'auth_string'` 値は指定されたプラグインと互換性のある形式です。

また、`IDENTIFIED BY PASSWORD` 構文が削除されたため、`log_builtin_as_identified_by_password` システム変数は不要になり、削除されました。

- `PASSWORD()` 関数。なお、`PASSWORD()` の削除とは、`SET PASSWORD ... = PASSWORD('auth_string')` 構文が使用できなくなったことを意味します。
- `old_passwords` システム変数。

- クエリーキャッシュが削除されました。削除には次の項目が含まれます:
 - `FLUSH QUERY CACHE` および `RESET QUERY CACHE` ステートメント。
 - これらのシステム変数: `query_cache_limit`, `query_cache_min_res_unit`, `query_cache_size`, `query_cache_type`, `query_cache_wlock_invalidate`。
 - これらのステータス変数: `Qcache_free_blocks`, `Qcache_free_memory`, `Qcache_hits`, `Qcache_inserts`, `Qcache_lowmem_prunes`, `Qcache_not_cached`, `Qcache_queries_in_cache`, `Qcache_total_blocks`。
 - これらのスレッド状態: `checking privileges on cached query`, `checking query cache for query`, `invalidating query cache entries`, `sending cached result to client`, `storing result in query cache`, `Waiting for query cache lock`。
 - `SQL_CACHE SELECT` 修飾子。

これらの非推奨のクエリーキャッシュアイテムは非推奨のままですが、効果はありません。将来の MySQL リリースで削除される予定です:

- `SQL_NO_CACHE SELECT` 修飾子。
- `ndb_cache_check_time` システム変数。

`have_query_cache` システム変数は非推奨のままであり、常に `NO` の値を持ちます。将来の MySQL リリースで削除される予定です。

- データディクショナリはデータベースオブジェクトに関する情報を提供するため、サーバーはデータディレクトリ内のディレクトリ名をチェックしてデータベースを検索しなくなります。その結果、`--ignore-db-dir` オプションと `ignore_db_dirs` システム変数は無意味となり、削除されました。
- メタデータログとも呼ばれる DDL ログが削除されました。MySQL 8.0.3 以降、この機能はデータディクショナリの `innodb_ddl_log` テーブルによって処理されます。[DDL ログの表示](#)を参照してください。
- `tx_isolation` および `tx_read_only` システム変数が削除されました。かわりに `transaction_isolation` および `transaction_read_only` を使用してください。
- `.frm` ファイルが廃止されたため、`sync_frm` システム変数は削除されました。
- `secure_auth` システム変数および `--secure-auth` クライアントオプションが削除されました。`mysql_options()` C API 関数の `MYSQL_SECURE_AUTH` オプションが削除されました。
- `multi_range_count` システム変数が削除されます。
- `log_warnings` システム変数および `--log-warnings` サーバーオプションが削除されました。かわりに `log_error_verbosity` システム変数を使用してください。
- `sql_log_bin` システム変数のグローバルスコープが削除されました。`sql_log_bin` にはセッションスコープのみとなり、`@@GLOBAL.sql_log_bin` へのアクセスに依存するアプリケーションは修正する必要があります。
- `metadata_locks_cache_size` および `metadata_locks_hash_instances` システム変数が削除されます。
- 未使用の `date_format`, `datetime_format`, `time_format` および `max_tmp_tables` システム変数が削除されます。
- これらの非推奨の互換性 SQL モードは削除されました: `DB2`, `MAXDB`, `MSSQL`, `MYSQL323`, `MYSQL40`, `ORACLE`, `POSTGRESQL`, `NO_FIELD_OPTIONS`, `NO_KEY_OPTIONS`, `NO_TABLE_OPTIONS`。これらを `sql_mode` システム変数に割り当てたり、`mysqldump --compatible` オプションの許可された値として使用したりすることはできなくなりました。

`MAXDB` の削除とは、`CREATE TABLE` または `ALTER TABLE` の `TIMESTAMP` データ型が `TIMESTAMP` として扱われ、`DATETIME` として扱われなくなることを意味します。

- `GROUP BY` 句の非推奨の `ASC` 修飾子または `DESC` 修飾子は削除されます。以前に `GROUP BY` ソートに依存していたクエリーでは、以前の MySQL バージョンとは異なる結果が生成される場合があります。特定のソート順序を生成するには、`ORDER BY` 句を指定します。
- `EXPLAIN` ステートメントの `EXTENDED` および `PARTITIONS` キーワードは削除されました。これらのキーワードは、その効果が常に有効になっているため不要です。

- 次の暗号化関連項目が削除されます:

- `ENCODE()` および `DECODE()` の機能。
- `ENCRYPT()` 関数。
- `DES_ENCRYPT()` 関数、`DES_DECRYPT()` 関数、`--des-key-file` オプション、`have_crypt` システム変数、`FLUSH` ステートメントの `DES_KEY_FILE` オプション、および `HAVE_CRYPT CMake` オプション。

削除された暗号化機能のかわり: `ENCRYPT()` の場合は、一方向ハッシュのかわりに `SHA2()` を使用することを検討してください。その他の場合は、かわりに `AES_ENCRYPT()` および `AES_DECRYPT()` の使用を検討してください。

- MySQL 5.7 では、複数の名前で使用可能ないくつかの空間関数は、空間関数ネームスペースの一貫性を高めるために非推奨になりました。各空間関数名が `ST_`(正確な操作を実行する場合) または `MBR`(最小境界矩形に基づいて操作を実行する場合) で始まるという目標です。MySQL 8.0 では、非推奨となった関数は削除され、対応する `ST_` および `MBR` 関数のみが残されます:

- これらの関数は、`MBR` 名を優先して削除されます: `Contains()`, `Disjoint()`, `Equals()`, `Intersects()`, `Overlaps()`, `Within()`。

- これらの関数は、`ST_` 名を優先して削除されます: `Area()`, `AsBinary()`, `AsText()`, `AsWKB()`, `AsWKT()`, `Buffer()`, `Centroid()`, `ConvexHull()`, `Crosses()`, `Dimension()`, `Distance()`, `EndPoint()`, `Envelope()`, `ExteriorRing()`, `GeomCollFromText()`, `GeomCollFromWKB()`, `GeomFromText()`, `GeomFromWKB()`, `GeometryCollectionFromText()`, `GeometryCollectionFromWKB()`, `GeometryFromText()`, `GeometryFromWKB()`, `GeometryN()`, `GeometryType()`, `InteriorRingN()`, `IsClosed()`, `IsEmpty()`, `IsSimple()`, `LineFromText()`, `LineFromWKB()`, `LineStringFromText()`, `LineStringFromWKB()`, `MLineFromText()`, `MLineFromWKB()`, `MPointFromText()`, `MPointFromWKB()`, `MPolyFromText()`, `MPolyFromWKB()`, `MultiLineStringFromText()`, `MultiLineStringFromWKB()`, `MultiPointFromText()`, `MultiPointFromWKB()`, `MultiPolygonFromText()`, `MultiPolygonFromWKB()`, `NumGeometries()`, `NumInteriorRings()`, `NumPoints()`, `PointFromText()`, `PointFromWKB()`, `PointN()`, `PolyFromText()`, `PolyFromWKB()`, `PolygonFromText()`, `PolygonFromWKB()`, `SRID()`, `StartPoint()`, `Touches()`, `X()`, `Y()`。

- `GLength()` は、`ST_Length()` を優先して削除されます。

- セクション12.17.4「WKB 値からジオメトリ値を作成する関数」で説明されている関数は、WKB 文字列またはジオメトリ引数を以前に受け入れていました。ジオメトリ引数は許可されなくなり、エラーが発生します。ジオメトリ引数を使用しないでクエリーを移行するためのガイドラインについては、そのセクションを参照してください。
- パーサーは、SQL ステートメントで `\N` を `NULL` のシノニムとして処理しなくなりました。かわりに `NULL` を使用してください。

この変更は、`NULL` が `\N` によって引き続き表される `LOAD DATA` または `SELECT ... INTO OUTFILE` で実行されるテキストファイルのインポートまたはエクスポート操作には影響しません。セクション13.2.7「LOAD DATA ステートメント」を参照してください。

- `PROCEDURE ANALYSE()` 構文が削除されました。

- クライアント側の `--ssl` および `--ssl-verify-server-cert` オプションが削除されました。 `--ssl=1` または `--enable-ssl` のかわりに `--ssl-mode=REQUIRED` を使用します。 `--ssl=0`、`--skip-ssl` または `--disable-ssl` のかわりに `--ssl-mode=DISABLED` を使用します。 `--ssl-verify-server-cert` オプションのかわりに `--ssl-mode=VERIFY_IDENTITY` を使用します。(サーバー側の `--ssl` オプションは変更されません。)

C API の場合、`mysql_options()` の `MYSQL_OPT_SSL_ENFORCE` および `MYSQL_OPT_SSL_VERIFY_SERVER_CERT` オプションはクライアント側の `--ssl` および `--ssl-verify-server-cert` オプションに対応し、削除されます。かわりに、`SSL_MODE_REQUIRED` または `SSL_MODE_VERIFY_IDENTITY` のオプション値を指定して `MYSQL_OPT_SSL_MODE` を使用します。

- `--temp-pool` サーバーオプションが削除されました。

- `ignore_builtin_innodb` システム変数が削除されます。

- サーバーは、特殊文字を含む pre-MySQL 5.1 データベース名を、`#mysql50#`接頭辞が追加された 5.1 形式に変換しなくなりました。これらの変換は実行されなくなったため、`mysqlcheck` の `--fix-db-names` および `--fix-table-`

`names` オプション、`ALTER DATABASE` ステートメントの `UPGRADE DATA DIRECTORY NAME` 句および `Com_alter_db_upgrade` ステータス変数は削除されます。

アップグレードは、あるメジャーバージョンから別のメジャーバージョン (5.0 から 5.1、5.1 から 5.5 など) へのアップグレードでのみサポートされるため、古い 5.0 データベース名を現在のバージョンの MySQL に変換する必要はほとんどありません。回避策として、より新しいリリースにアップグレードする前に、MySQL 5.0 インストールを MySQL 5.1 にアップグレードします。

- `mysql_install_db` プログラムが MySQL ディストリビューションから削除されました。データディレクトリの初期化は、かわりに `--initialize` または `--initialize-insecure` オプションを指定して `mysqld` を起動することで実行する必要があります。さらに、`mysql_install_db` で使用された `mysqld` の `--bootstrap` オプションが削除され、`mysql_install_db` のインストール場所を制御する `INSTALL_SCRIPTDIR CMake` オプションが削除されました。
- 汎用パーティション化ハンドラが MySQL サーバーから削除されました。特定のテーブルのパーティション化をサポートするには、テーブルに使用されるストレージエンジンが独自の (「native」) パーティショニングハンドラを提供する必要があります。`--partition` および `--skip-partition` オプションは MySQL Server から削除され、パーティション関連のエントリは `SHOW PLUGINS` の出力または `INFORMATION_SCHEMA.PLUGINS` テーブルに表示されなくなります。

現在、2 つの MySQL ストレージエンジンがネイティブのパーティショニングサポートを提供しています: `InnoDB` および `NDB`。これらのうち、MySQL 8.0 でサポートされているのは `InnoDB` のみです。他のストレージエンジンを使用して MySQL 8.0 でパーティションテーブルを作成しようとすると、失敗します。

アップグレードの影響。 `InnoDB` (`MyISAM` など) 以外のストレージエンジンを使用した MySQL 5.7 以前から MySQL 8.0 へのパーティションテーブルの直接アップグレードはサポートされていません。このようなテーブルを処理するには、次の 2 つのオプションがあります:

- `ALTER TABLE ... REMOVE PARTITIONING` を使用してテーブルのパーティション化を削除します。
- `ALTER TABLE ... ENGINE=INNODB` を使用して、テーブルに使用されるストレージエンジンを `InnoDB` に変更します。

サーバーを MySQL 8.0 にアップグレードする前に、パーティション化された `InnoDB` 以外のテーブルごとに、前述の 2 つ以上の操作を実行する必要があります。そうしないと、アップグレード後にそのようなテーブルを使用できません。

パーティション化がサポートされていない記憶域エンジンを使用してパーティション化されたテーブルを作成するテーブル作成ステートメントがエラー (`ER_CHECK_NOT_IMPLEMENTED`) で失敗するようになりました。そのため、MySQL 8.0 にインポートする古いバージョンの MySQL から出力されたダンプファイル (`mysqldump` によって出力されたものなど) に記載された、パーティション化されたテーブルを作成するステートメントが、ネイティブパーティショニングハンドラを持たない `MyISAM` などの記憶域エンジンを指定していないことを確認する必要があります。これを行うには、次のいずれかを実行します:

- `InnoDB` 以外の `STORAGE ENGINE` オプションの値を使用する `CREATE TABLE` ステートメントからパーティション化への参照を削除します。
- ストレージエンジンを `InnoDB` として指定するか、デフォルトで `InnoDB` をテーブルストレージエンジンとして使用できるようにします。

詳細は、[セクション 24.6.2 「ストレージエンジンに関連するパーティショニング制限」](#) を参照してください。

- システム変数およびステータス変数の情報は、`INFORMATION_SCHEMA` で管理されなくなりました。これらのテーブルは削除されます: `GLOBAL_VARIABLES`, `SESSION_VARIABLES`, `GLOBAL_STATUS`, `SESSION_STATUS`。かわりに、対応する「パフォーマンススキーマ」テーブルを使用してください。[セクション 27.12.14 「パフォーマンススキーマシステム変数テーブル」](#) および [セクション 27.12.15 「パフォーマンススキーマのステータス変数のテーブル」](#) を参照してください。さらに、`show_compatibility_56` システム変数が削除されました。これは、`INFORMATION_SCHEMA` テーブルのシステム変数およびステータス変数の情報が「パフォーマンススキーマ」テーブルに移動され、不要になった移行期間に使用されました。これらのステータス変数は削除されます: `Slave_heartbeat_period`, `Slave_last_heartbeat`, `Slave_received_heartbeats`, `Slave_retried_transactions`, `Slave_running`。提供される情報は、「パフォーマンススキーマ」のテーブルにあります。[Migrating to Performance Schema System and Status Variable Tables](#) を参照してください。

- パフォーマンススキーマ `setup_timers` テーブルは、`performance_timers` テーブルの `TICK` 行と同様に削除されました。
- `libmysqld` 埋込みサーバーライブラリが次とともに削除されます:
 - `mysql_options()` `MYSQL_OPT_GUESS_CONNECTION`, `MYSQL_OPT_USE_EMBEDDED_CONNECTION`, `MYSQL_OPT_USE_REMOTE_CONNECTION` および `MYSQL_SET_CLIENT_IP` のオプション
 - `mysql_config --libmysqld-libs`, `--embedded-libs` および `--embedded` のオプション
 - CMake `WITH_EMBEDDED_SERVER`, `WITH_EMBEDDED_SHARED_LIBRARY` および `INSTALL_SECURE_FILE_PRIV_EMBEDDED` のオプション
 - (ドキュメント化されていない)`mysql --server-arg` オプション
 - `mysqltest --embedded-server`, `--server-arg` および `--server-file` のオプション
 - `mysqltest_embedded` および `mysql_client_test_embedded` のテストプログラム
- `mysql_plugin` ユーティリティが削除されました。代替方法には、`--plugin-load` または `--plugin-load-add` オプションを使用したサーバー起動時、または `INSTALL PLUGIN` ステートメントを使用した実行時のプラグインのロードが含まれます。
- `resolveip` ユーティリティが削除されます。かわりに、`nslookup`、`host` または `dig` を使用できます。
- `resolve_stack_dump` ユーティリティが削除されます。正式な MySQL ビルドからのスタックトレースは常にシンボル化されるため、`resolve_stack_dump` を使用する必要はありません。
- 次のサーバーエラーコードは使用されておらず、削除されています。これらのエラーのいずれかをテストするアプリケーションを更新する必要があります。

```

ER_BINLOG_READ_EVENT_CHECKSUM_FAILURE
ER_BINLOG_ROW_RBR_TO_SBR
ER_BINLOG_ROW_WRONG_TABLE_DEF
ER_CANT_ACTIVATE_LOG
ER_CANT_CHANGE_GTID_NEXT_IN_TRANSACTION
ER_CANT_CREATE_FEDERATED_TABLE
ER_CANT_CREATE_SROUTINE
ER_CANT_DELETE_FILE
ER_CANT_GET_WD
ER_CANT_SET_GTID_PURGED_WHEN_GTID_MODE_IS_OFF
ER_CANT_SET_WD
ER_CANT_WRITE_LOCK_LOG_TABLE
ER_CREATE_DB_WITH_READ_LOCK
ER_CYCLIC_REFERENCE
ER_DB_DROP_DELETE
ER_DELAYED_NOT_SUPPORTED
ER_DIFF_GROUPS_PROC
ER_DISK_FULL
ER_DROP_DB_WITH_READ_LOCK
ER_DROP_USER
ER_DUMP_NOT_IMPLEMENTED
ER_ERROR_DURING_CHECKPOINT
ER_ERROR_ON_CLOSE
ER_EVENTS_DB_ERROR
ER_EVENT_CANNOT_DELETE
ER_EVENT_CANT_ALTER
ER_EVENT_COMPILE_ERROR
ER_EVENT_DATA_TOO_LONG
ER_EVENT_DROP_FAILED
ER_EVENT_MODIFY_QUEUE_ERROR
ER_EVENT_NEITHER_M_EXPR_NOR_M_AT
ER_EVENT_OPEN_TABLE_FAILED
ER_EVENT_STORE_FAILED
ER_EXEC_STMT_WITH_OPEN_CURSOR
ER_FAILED_ROUTINE_BREAK_BINLOG
ER_FLUSH_MASTER_BINLOG_CLOSED
ER_FORM_NOT_FOUND
ER_FOUND_GTID_EVENT_WHEN_GTID_MODE_IS_OFF__UNUSED
ER_FRM_UNKNOWN_TYPE

```

```
ER_GOT_SIGNAL
ER_GRANT_PLUGIN_USER_EXISTS
ER_GTID_MODE_REQUIRES_BINLOG
ER_GTID_NEXT_IS_NOT_IN_GTID_NEXT_LIST
ER_HASHCHK
ER_INDEX_REBUILD
ER_INNODB_NO_FT_USES_PARSER
ER_LIST_OF_FIELDS_ONLY_IN_HASH_ERROR
ER_LOAD_DATA_INVALID_COLUMN_UNUSED
ER_LOGGING_PROHIBIT_CHANGING_OF
ER_MALFORMED_DEFINER
ER_MASTER_KEY_ROTATION_ERROR_BY_SE
ER_NDB_CANT_SWITCH_BINLOG_FORMAT
ER_NEVER_USED
ER_NISAMCHK
ER_NO_CONST_EXPR_IN_RANGE_OR_LIST_ERROR
ER_NO_FILE_MAPPING
ER_NO_GROUP_FOR_PROC
ER_NO_RAID_COMPILED
ER_NO_SUCH_KEY_VALUE
ER_NO_SUCH_PARTITION_UNUSED
ER_OBSOLETE_CANNOT_LOAD_FROM_TABLE
ER_OBSOLETE_COL_COUNT_DOESNT_MATCH_CORRUPTED
ER_ORDER_WITH_PROC
ER_PARTITION_SUBPARTITION_ERROR
ER_PARTITION_SUBPART_MIX_ERROR
ER_PART_STATE_ERROR
ER_PASSWD_LENGTH
ER_QUERY_ON_MASTER
ER_RBR_NOT_AVAILABLE
ER_SKIPPING_LOGGED_TRANSACTION
ER_SLAVE_CHANNEL_DELETE
ER_SLAVE_MULTIPLE_CHANNELS_HOST_PORT
ER_SLAVE_MUST_STOP
ER_SLAVE_WAS_NOT_RUNNING
ER_SLAVE_WAS_RUNNING
ER_SP_GOTO_IN_HNDLR
ER_SP_PROC_TABLE_CORRUPT
ER_SQL_MODE_NO_EFFECT
ER_SR_INVALID_CREATION_CTX
ER_TABLE_NEEDS_UPG_PART
ER_TOO_MUCH_AUTO_TIMESTAMP_COLS
ER_UNEXPECTED_EOF
ER_UNION_TABLES_IN_DIFFERENT_DIR
ER_UNSUPPORTED_BY_REPLICATION_THREAD
ER_UNUSED1
ER_UNUSED2
ER_UNUSED3
ER_UNUSED4
ER_UNUSED5
ER_UNUSED6
ER_VIEW_SELECT_DERIVED_UNUSED
ER_WRONG_MAGIC
ER_WSAS_FAILED
```

- 非推奨の [INFORMATION_SCHEMA.INNODB_LOCKS](#) および [INNODB_LOCK_WAITS](#) テーブルは削除されます。代わりに、パフォーマンススキーマ [data_locks](#) および [data_lock_waits](#) テーブルを使用してください。

注記

MySQL 5.7 では、[INNODB_LOCKS](#) テーブルの [LOCK_TABLE](#) カラムと [sys](#) スキーマ [innodb_lock_waits](#) および [x\\$innodb_lock_waits](#) ビューの [locked_table](#) カラムには、スキーマ/テーブル名の値が結合されています。MySQL 8.0 では、[data_locks](#) テーブルおよび [sys](#) スキーマビューに個別のスキーマ名およびテーブル名のカラムが含まれます。[セクション28.4.3.9「innodb_lock_waits および x\\$innodb_lock_waits のビュー」](#)を参照してください。

- InnoDB では、圧縮一時テーブルはサポートされなくなりました。[innodb_strict_mode](#) が有効な場合 (デフォルト)、[ROW_FORMAT=COMPRESSED](#) または [KEY_BLOCK_SIZE](#) が指定されていると、[CREATE TEMPORARY TABLE](#) はエラーを返します。[innodb_strict_mode](#) が無効な場合は、警告が発行され、圧縮されていない行形式を使用して一時テーブルが作成されます。

- InnoDB では、MySQL データディレクトリ外にテーブルスペースデータファイルを作成する際に、`.isl` ファイル (InnoDB シンボリックリンクファイル) は作成されなくなりました。 `innodb_directories` オプションでは、データディレクトリ外で作成されたテーブルスペースファイルの検索がサポートされるようになりました。

この変更により、`.isl` ファイルを手動で変更してサーバーがオフラインのときにリモートテーブルスペースを移動することはサポートされなくなりました。リモートテーブルスペースファイルの移動は、`innodb_directories` オプションでサポートされるようになりました。 [セクション15.6.3.6「サーバーがオフラインのときのテーブルスペースファイルの移動」](#) を参照してください。

- 次の InnoDB ファイル形式変数が削除されました:

- `innodb_file_format`
- `innodb_file_format_check`
- `innodb_file_format_max`
- `innodb_large_prefix`

MySQL 5.1 で以前のバージョンの InnoDB と互換性のあるテーブルを作成するには、ファイル形式変数が必要でした。MySQL 5.1 が製品ライフサイクルの最後に到達したため、これらのオプションは不要になりました。

`FILE_FORMAT` カラムが `INNODB_TABLES` テーブルおよび `INNODB_TABLESPACES` 「情報スキーマ」テーブルから削除されました。

- XA トランザクションでの双方向コミットのサポートを可能にする `innodb_support_xa` システム変数が削除されました。XA トランザクションでの双方向コミットの InnoDB サポートは、常に有効です。
- DTrace のサポートが削除されました。
- `JSON_APPEND()` 関数が削除されました。かわりに `JSON_ARRAY_APPEND()` を使用してください。
- 共有 InnoDB テーブルスペースへのテーブルパーティションの配置のサポートは、MySQL 8.0.13 で削除されました。共有テーブルスペースには、InnoDB システムテーブルスペースおよび一般テーブルスペースが含まれます。共有テーブルスペースのパーティションの識別および file-per-table テーブルスペースへの移動の詳細は、[セクション2.11.5「アップグレード用のインストールの準備」](#) を参照してください。
- `SET` 以外のステートメントでのユーザー変数の設定のサポートは、MySQL 8.0.13 で非推奨になりました。この機能は、MySQL 9.0 で削除されることがあります。
- `--ndb_perror` オプションが削除されました。かわりに `ndb_perror` ユーティリティを使用してください。
- `innodb_undo_logs` 変数が削除されました。 `innodb_rollback_segments` 変数は同じ機能を実行するため、かわりに使用する必要があります。
- `Innodb_available_undo_logs` ステータス変数が削除されました。テーブルスペースごとに使用可能なロールバックセグメントの数は、`SHOW VARIABLES LIKE 'innodb_rollback_segments';` を使用して取得できます
- MySQL 8.0.14 では、以前に非推奨となった `innodb_undo_tablespaces` 変数は構成できなくなりました。詳細は、[セクション15.6.3.4「undo テーブルスペース」](#) を参照してください。
- `ALTER TABLE ... UPGRADE PARTITIONING` ステートメントのサポートは削除されました。
- MySQL 8.0.16 では、`internal_tmp_disk_storage_engine` システム変数のサポートが削除されました。ディスク上の内部一時テーブルでは、常に InnoDB ストレージエンジンが使用されるようになりました。詳細は、[オンディスク内部一時テーブルのストレージエンジン](#) を参照してください。
- `DISABLE_SHARED_CMAKE` オプションは未使用であり、削除されました。

1.4 MySQL 8.0 で追加、非推奨または削除されたサーバーおよびステータスの変数とオプション

- [導入されたオプションおよび変数: MySQL 8.0](#)

- [非推奨となったオプションおよび変数: MySQL 8.0](#)
- [削除されたオプションおよび変数: MySQL 8.0](#)

このセクションでは、MySQL 8.0 で初めて追加された、非推奨になった、または削除されたサーバー変数、ステータス変数およびオプションを示します。

導入されたオプションおよび変数: MySQL 8.0

次のシステム変数、ステータス変数およびサーバーオプションが追加されました: MySQL 8.0.

- [Acl_cache_items_count](#): キャッシュされた権限オブジェクトの数. 追加されたバージョン: MySQL 8.0.0.
- [Audit_log_current_size](#): 監査ログファイルの現在のサイズ. 追加されたバージョン: MySQL 8.0.11.
- [Audit_log_event_max_drop_size](#): 削除された最大の監査イベントのサイズ. 追加されたバージョン: MySQL 8.0.11.
- [Audit_log_events](#): 処理された監査イベントの数. 追加されたバージョン: MySQL 8.0.11.
- [Audit_log_events_filtered](#): フィルタされた監査イベントの数. 追加されたバージョン: MySQL 8.0.11.
- [Audit_log_events_lost](#): 削除された監査イベントの数. 追加されたバージョン: MySQL 8.0.11.
- [Audit_log_events_written](#): 書き込まれた監査イベントの数. 追加されたバージョン: MySQL 8.0.11.
- [Audit_log_total_size](#): 書き込まれた監査イベントの合計サイズ. 追加されたバージョン: MySQL 8.0.11.
- [Audit_log_write_waits](#): 書き込みが遅延された監査イベントの数. 追加されたバージョン: MySQL 8.0.11.
- [Authentication_ldap_sasl_supported_methods](#): SASL LDAP 認証でサポートされる認証方式. 追加されたバージョン: MySQL 8.0.21.
- [Caching_sha2_password_rsa_public_key](#): caching_sha2_password 認証プラグイン RSA 公開キー値. 追加されたバージョン: MySQL 8.0.4.
- [Com_alter_resource_group](#): ALTER RESOURCE GROUP ステートメントの数. 追加されたバージョン: MySQL 8.0.3.
- [Com_alter_user_default_role](#): ALTER USER ... DEFAULT ROLE ステートメントの数. 追加されたバージョン: MySQL 8.0.0.
- [Com_clone](#): CLONE ステートメントの数. 追加されたバージョン: MySQL 8.0.2.
- [Com_create_resource_group](#): CREATE RESOURCE GROUP ステートメントの数. 追加されたバージョン: MySQL 8.0.3.
- [Com_create_role](#): CREATE ROLE ステートメントの数. 追加されたバージョン: MySQL 8.0.0.
- [Com_drop_resource_group](#): DROP RESOURCE GROUP ステートメントの数. 追加されたバージョン: MySQL 8.0.3.
- [Com_drop_role](#): DROP ROLE ステートメントの数. 追加されたバージョン: MySQL 8.0.0.
- [Com_grant_roles](#): GRANT ROLE ステートメントの数. 追加されたバージョン: MySQL 8.0.0.
- [Com_install_component](#): INSTALL COMPONENT ステートメントの数. 追加されたバージョン: MySQL 8.0.0.
- [Com_replica_start](#): START REPLICA および START SLAVE ステートメントの数. 追加されたバージョン: MySQL 8.0.22.
- [Com_replica_stop](#): STOP REPLICA および STOP SLAVE ステートメントの数. 追加されたバージョン: MySQL 8.0.22.
- [Com_restart](#): RESTART ステートメントの数. 追加されたバージョン: MySQL 8.0.4.
- [Com_revoke_roles](#): REVOKE ROLES ステートメントの数. 追加されたバージョン: MySQL 8.0.0.
- [Com_set_resource_group](#): SET RESOURCE GROUP ステートメントの数. 追加されたバージョン: MySQL 8.0.3.

- [Com_set_role](#): SET ROLE ステートメントの数. 追加されたバージョン: MySQL 8.0.0.
- [Com_show_replica_status](#): SHOW REPLICA STATUS および SHOW SLAVE STATUS ステートメントの数. 追加されたバージョン: MySQL 8.0.22.
- [Com_show_replicas](#): SHOW REPLICAS および SHOW SLAVE HOSTS ステートメントの数. 追加されたバージョン: MySQL 8.0.22.
- [Com_uninstall_component](#): UINSTALL COMPONENT ステートメントの数. 追加されたバージョン: MySQL 8.0.0.
- [Compression_algorithm](#): 現在の接続の圧縮アルゴリズム. 追加されたバージョン: MySQL 8.0.18.
- [Compression_level](#): 現在の接続の圧縮レベル. 追加されたバージョン: MySQL 8.0.18.
- [Connection_control_delay_generated](#): サーバーが接続リクエストを遅延した回数. 追加されたバージョン: MySQL 8.0.1.
- [Current_tls_ca](#): ssl_ca システム変数の現在の値. 追加されたバージョン: MySQL 8.0.16.
- [Current_tls_capath](#): ssl_capath システム変数の現在の値. 追加されたバージョン: MySQL 8.0.16.
- [Current_tls_cert](#): ssl_cert システム変数の現在の値. 追加されたバージョン: MySQL 8.0.16.
- [Current_tls_cipher](#): ssl_cipher システム変数の現在の値. 追加されたバージョン: MySQL 8.0.16.
- [Current_tls_ciphersuites](#): ssl_ciphersuites システム変数の現在の値. 追加されたバージョン: MySQL 8.0.16.
- [Current_tls_crl](#): ssl_crl システム変数の現在の値. 追加されたバージョン: MySQL 8.0.16.
- [Current_tls_crlpath](#): ssl_crlpath システム変数の現在の値. 追加されたバージョン: MySQL 8.0.16.
- [Current_tls_key](#): ssl_key システム変数の現在の値. 追加されたバージョン: MySQL 8.0.16.
- [Current_tls_version](#): ssl_version システム変数の現在の値. 追加されたバージョン: MySQL 8.0.16.
- [Error_log_buffered_bytes](#): error_log テーブルで使用されるバイト数. 追加されたバージョン: MySQL 8.0.22.
- [Error_log_buffered_events](#): error_log テーブルのイベント数. 追加されたバージョン: MySQL 8.0.22.
- [Error_log_expired_events](#): error_log テーブルから破棄されたイベントの数. 追加されたバージョン: MySQL 8.0.22.
- [Error_log_latest_write](#): error_log テーブルへの最終書き込み時間. 追加されたバージョン: MySQL 8.0.22.
- [Firewall_access_denied](#): MySQL Enterprise Firewall によって拒否されたステートメントの数. 追加されたバージョン: MySQL 8.0.11.
- [Firewall_access_granted](#): MySQL Enterprise Firewall で受け入れられたステートメントの数. 追加されたバージョン: MySQL 8.0.11.
- [Firewall_cached_entries](#): MySQL Enterprise Firewall によって記録されたステートメントの数. 追加されたバージョン: MySQL 8.0.11.
- [InnoDB_redo_log_enabled](#): InnoDB redo ログステータス. 追加されたバージョン: MySQL 8.0.21.
- [InnoDB_system_rows_deleted](#): システムスキーマテーブルから削除された行数. 追加されたバージョン: MySQL 8.0.19.
- [InnoDB_system_rows_inserted](#): システムスキーマテーブルに挿入された行数. 追加されたバージョン: MySQL 8.0.19.
- [InnoDB_system_rows_read](#): システムスキーマテーブルから読み取られた行数. 追加されたバージョン: MySQL 8.0.19.
- [InnoDB_undo_tablespaces_active](#): アクティブな undo テーブルスペースの数. 追加されたバージョン: MySQL 8.0.14.
- [InnoDB_undo_tablespaces_explicit](#): ユーザー作成 undo テーブルスペースの数. 追加されたバージョン: MySQL 8.0.14.

- [Innodb_undo_tablespaces_implicit](#): InnoDB によって作成された undo テーブルスペースの数. 追加されたバージョン: MySQL 8.0.14.
- [Innodb_undo_tablespaces_total](#): undo テーブルスペースの合計数. 追加されたバージョン: MySQL 8.0.14.
- [Mysqlx_bytes_received_compressed_payload](#): 圧縮メッセージペイロードとして受信したバイト数. 圧縮解除前に測定されます. 追加されたバージョン: MySQL 8.0.19.
- [Mysqlx_bytes_received_uncompressed_frame](#): 圧縮メッセージペイロードとして受信したバイト数 (解凍後に測定). 追加されたバージョン: MySQL 8.0.19.
- [Mysqlx_bytes_sent_compressed_payload](#): 圧縮後に測定された、圧縮メッセージペイロードとして送信されたバイト数. 追加されたバージョン: MySQL 8.0.19.
- [Mysqlx_bytes_sent_uncompressed_frame](#): 圧縮されたメッセージペイロードとして送信されたバイト数 (圧縮前に測定). 追加されたバージョン: MySQL 8.0.19.
- [Mysqlx_compression_algorithm](#): このセッションの X Protocol 接続に使用されている圧縮アルゴリズム. 追加されたバージョン: MySQL 8.0.20.
- [Mysqlx_compression_level](#): このセッションの X Protocol 接続に使用されている圧縮レベル. 追加されたバージョン: MySQL 8.0.20.
- [Secondary_engine_execution_count](#): セカンダリエンジンにオフロードされたクエリーの数. . 追加されたバージョン: MySQL 8.0.13.
- [activate_all_roles_on_login](#): 接続時にすべてのユーザーロールをアクティブ化するかどうか. 追加されたバージョン: MySQL 8.0.2.
- [admin-ssl](#): 接続暗号化の有効化. 追加されたバージョン: MySQL 8.0.21.
- [admin_address](#): 管理インタフェースでの接続用にバインドする IP アドレス. 追加されたバージョン: MySQL 8.0.14.
- [admin_port](#): 管理インタフェースでの接続に使用する TCP/IP 番号. 追加されたバージョン: MySQL 8.0.14.
- [admin_ssl_ca](#): 信頼できる SSL 認証局のリストを含むファイル. 追加されたバージョン: MySQL 8.0.21.
- [admin_ssl_capath](#): 信頼できる SSL 認証局の証明書ファイルを含むディレクトリ. 追加されたバージョン: MySQL 8.0.21.
- [admin_ssl_cert](#): X.509 証明書を含むファイル. 追加されたバージョン: MySQL 8.0.21.
- [admin_ssl_cipher](#): 接続の暗号化に許可される暗号. 追加されたバージョン: MySQL 8.0.21.
- [admin_ssl_crl](#): 証明書失効リストを含むファイル. 追加されたバージョン: MySQL 8.0.21.
- [admin_ssl_crlpath](#): 証明書失効リストファイルを含むディレクトリ. 追加されたバージョン: MySQL 8.0.21.
- [admin_ssl_key](#): X.509 キーを含むファイル. 追加されたバージョン: MySQL 8.0.21.
- [admin_tls_ciphersuites](#): 暗号化された接続に許可される TLSv1.3 暗号スイート. 追加されたバージョン: MySQL 8.0.21.
- [admin_tls_version](#): 暗号化された接続に許可される TLS プロトコル. 追加されたバージョン: MySQL 8.0.21.
- [audit-log](#): 監査ログプラグインをアクティブにするかどうか. 追加されたバージョン: MySQL 8.0.11.
- [audit_log_buffer_size](#): 監査ログバッファのサイズ. 追加されたバージョン: MySQL 8.0.11.
- [audit_log_compression](#): 監査ログファイルの圧縮方法. 追加されたバージョン: MySQL 8.0.11.
- [audit_log_connection_policy](#): 接続関係のイベントの監査ロギングポリシー. 追加されたバージョン: MySQL 8.0.11.
- [audit_log_current_session](#): 現在のセッションを監査するかどうか. 追加されたバージョン: MySQL 8.0.11.
- [audit_log_encryption](#): 監査ログファイルの暗号化方法. 追加されたバージョン: MySQL 8.0.11.
- [audit_log_exclude_accounts](#): 監査しないアカウント. 追加されたバージョン: MySQL 8.0.11.

- [audit_log_file](#): 監査ログファイルの名前. 追加されたバージョン: MySQL 8.0.11.
- [audit_log_filter_id](#): 現在の監査ログフィルタの ID. 追加されたバージョン: MySQL 8.0.11.
- [audit_log_flush](#): 監査ログファイルを閉じて再度開きます. 追加されたバージョン: MySQL 8.0.11.
- [audit_log_format](#): 監査ログファイル形式. 追加されたバージョン: MySQL 8.0.11.
- [audit_log_include_accounts](#): 監査するアカウント. 追加されたバージョン: MySQL 8.0.11.
- [audit_log_password_history_keep_days](#): アーカイブ監査ログの暗号化パスワードを保持する日数. 追加されたバージョン: MySQL 8.0.17.
- [audit_log_policy](#): 監査ロギングポリシー. 追加されたバージョン: MySQL 8.0.11.
- [audit_log_prune_seconds](#): 監査ログファイルがブルーニングの対象になるまでの秒数. 追加されたバージョン: MySQL 8.0.24.
- [audit_log_read_buffer_size](#): 監査ログファイル読取りバッファサイズ. 追加されたバージョン: MySQL 8.0.11.
- [audit_log_rotate_on_size](#): このサイズの監査ログファイルを閉じて再度開きます. 追加されたバージョン: MySQL 8.0.11.
- [audit_log_statement_policy](#): ステートメント関係のイベントの監査ロギングポリシー. 追加されたバージョン: MySQL 8.0.11.
- [audit_log_strategy](#): 監査ロギング戦略. 追加されたバージョン: MySQL 8.0.11.
- [authentication_ldap_sasl_auth_method_name](#): 認証方式名. 追加されたバージョン: MySQL 8.0.11.
- [authentication_ldap_sasl_bind_base_dn](#): LDAP サーバーベース識別名. 追加されたバージョン: MySQL 8.0.11.
- [authentication_ldap_sasl_bind_root_dn](#): LDAP サーバルート識別名. 追加されたバージョン: MySQL 8.0.11.
- [authentication_ldap_sasl_bind_root_pwd](#): LDAP サーバーのルートバインドパスワード. 追加されたバージョン: MySQL 8.0.11.
- [authentication_ldap_sasl_ca_path](#): LDAP サーバー認証局ファイル名. 追加されたバージョン: MySQL 8.0.11.
- [authentication_ldap_sasl_group_search_attr](#): LDAP サーバークラス検索属性. 追加されたバージョン: MySQL 8.0.11.
- [authentication_ldap_sasl_group_search_filter](#): LDAP カスタムグループ検索フィルタ. 追加されたバージョン: MySQL 8.0.11.
- [authentication_ldap_sasl_init_pool_size](#): LDAP サーバーの初期接続プールサイズ. 追加されたバージョン: MySQL 8.0.11.
- [authentication_ldap_sasl_log_status](#): LDAP サーバーのログレベル. 追加されたバージョン: MySQL 8.0.11.
- [authentication_ldap_sasl_max_pool_size](#): LDAP サーバーの最大接続プールサイズ. 追加されたバージョン: MySQL 8.0.11.
- [authentication_ldap_sasl_referral](#): LDAP 検索リフェラルを有効にするかどうか. 追加されたバージョン: MySQL 8.0.20.
- [authentication_ldap_sasl_server_host](#): LDAP サーバーのホスト名または IP アドレス. 追加されたバージョン: MySQL 8.0.11.
- [authentication_ldap_sasl_server_port](#): LDAP サーバーのポート番号. 追加されたバージョン: MySQL 8.0.11.
- [authentication_ldap_sasl_tls](#): LDAP サーバーへの暗号化接続を使用するかどうか. 追加されたバージョン: MySQL 8.0.11.
- [authentication_ldap_sasl_user_search_attr](#): LDAP サーバーユーザー検索属性. 追加されたバージョン: MySQL 8.0.11.

- [authentication_ldap_simple_auth_method_name](#): 認証方式名. 追加されたバージョン: MySQL 8.0.11.
- [authentication_ldap_simple_bind_base_dn](#): LDAP サーバーベース識別名. 追加されたバージョン: MySQL 8.0.11.
- [authentication_ldap_simple_bind_root_dn](#): LDAP サーバールート識別名. 追加されたバージョン: MySQL 8.0.11.
- [authentication_ldap_simple_bind_root_pwd](#): LDAP サーバーのルートバインドパスワード. 追加されたバージョン: MySQL 8.0.11.
- [authentication_ldap_simple_ca_path](#): LDAP サーバー認証局ファイル名. 追加されたバージョン: MySQL 8.0.11.
- [authentication_ldap_simple_group_search_attr](#): LDAP サーバークラウド検索属性. 追加されたバージョン: MySQL 8.0.11.
- [authentication_ldap_simple_group_search_filter](#): LDAP カスタムグループ検索フィルタ. 追加されたバージョン: MySQL 8.0.11.
- [authentication_ldap_simple_init_pool_size](#): LDAP サーバーの初期接続プールサイズ. 追加されたバージョン: MySQL 8.0.11.
- [authentication_ldap_simple_log_status](#): LDAP サーバーのログレベル. 追加されたバージョン: MySQL 8.0.11.
- [authentication_ldap_simple_max_pool_size](#): LDAP サーバーの最大接続プールサイズ. 追加されたバージョン: MySQL 8.0.11.
- [authentication_ldap_simple_referral](#): LDAP 検索リフェラルを有効にするかどうか. 追加されたバージョン: MySQL 8.0.20.
- [authentication_ldap_simple_server_host](#): LDAP サーバーのホスト名または IP アドレス. 追加されたバージョン: MySQL 8.0.11.
- [authentication_ldap_simple_server_port](#): LDAP サーバーのポート番号. 追加されたバージョン: MySQL 8.0.11.
- [authentication_ldap_simple_tls](#): LDAP サーバーへの暗号化接続を使用するかどうか. 追加されたバージョン: MySQL 8.0.11.
- [authentication_ldap_simple_user_search_attr](#): LDAP サーバークラウド検索属性. 追加されたバージョン: MySQL 8.0.11.
- [authentication_windows_log_level](#): Windows 認証プラグインのロギングレベル. 追加されたバージョン: MySQL 8.0.11.
- [authentication_windows_use_principal_name](#): Windows 認証プラグインの主体名を使用するかどうか. 追加されたバージョン: MySQL 8.0.11.
- [binlog_encryption](#): このサーバー上のバイナリログファイルおよびリレーログファイルの暗号化を有効にします. 追加されたバージョン: MySQL 8.0.14.
- [binlog_expire_logs_seconds](#): この秒数後にバイナリログをパージ. 追加されたバージョン: MySQL 8.0.1.
- [binlog_rotate_encryption_master_key_at_startup](#): サーバー起動時のバイナリログマスターキーのローテーション. 追加されたバージョン: MySQL 8.0.14.
- [binlog_row_metadata](#): 行ベースのロギングを使用している場合に、テーブル関連のすべてのメタデータをバイナリログに記録するか、最小限のメタデータのみを記録するか. 追加されたバージョン: MySQL 8.0.1.
- [binlog_row_value_options](#): 行ベースレプリケーションの部分 JSON 更新のバイナリロギングを有効にします. 追加されたバージョン: MySQL 8.0.3.
- [binlog_transaction_compression](#): バイナリログファイル内のトランザクションペイロードの圧縮を有効にします. 追加されたバージョン: MySQL 8.0.20.
- [binlog_transaction_compression_level_zstd](#): バイナリログファイル内のトランザクションペイロードの圧縮レベル. 追加されたバージョン: MySQL 8.0.20.
- [binlog_transaction_dependency_history_size](#): 一部の行を最後に更新したトランザクションを参照するために保持される行ハッシュの数. 追加されたバージョン: MySQL 8.0.1.

- [binlog_transaction_dependency_tracking](#): レプリカマルチスレッドアプリケーションでパラレルに実行できるトランザクションを評価する依存性情報 (コミットタイムスタンプまたはトランザクション書き込みセット) のソース. 追加されたバージョン: MySQL 8.0.1.
- [caching_sha2_password_auto_generate_rsa_keys](#): RSA キーペアファイルを自動生成するかどうか. 追加されたバージョン: MySQL 8.0.4.
- [caching_sha2_password_digest_rounds](#): `caching_sha2_password` 認証プラグインのハッシュラウンドの数. 追加されたバージョン: MySQL 8.0.24.
- [caching_sha2_password_private_key_path](#): SHA2 認証プラグインの秘密キーのパス名. 追加されたバージョン: MySQL 8.0.3.
- [caching_sha2_password_public_key_path](#): SHA2 認証プラグインの公開キーパス名. 追加されたバージョン: MySQL 8.0.3.
- [clone_autotune_concurrency](#): リモートクローニング操作のためのスレッドの動的生成を有効にします. 追加されたバージョン: MySQL 8.0.17.
- [clone_buffer_size](#): ドナー MySQL サーバーインスタンスの中間バッファのサイズを定義. 追加されたバージョン: MySQL 8.0.17.
- [clone_ddl_timeout](#): クローニング操作がバックアップロックを待機する秒数. 追加されたバージョン: MySQL 8.0.17.
- [clone_enable_compression](#): クローニング中のネットワークレイヤーでのデータ圧縮を有効にします. 追加されたバージョン: MySQL 8.0.17.
- [clone_max_concurrency](#): クローニング操作の実行に使用される同時スレッドの最大数. 追加されたバージョン: MySQL 8.0.17.
- [clone_max_data_bandwidth](#): リモートクローニング操作のための MiB での最大データ転送速度/秒. 追加されたバージョン: MySQL 8.0.17.
- [clone_max_network_bandwidth](#): リモートクローニング操作のための MiB の最大ネットワーク転送速度/秒. 追加されたバージョン: MySQL 8.0.17.
- [clone_ssl_ca](#): 認証局 (CA) ファイルへのパスを指定. 追加されたバージョン: MySQL 8.0.14.
- [clone_ssl_cert](#): 公開キー証明書ファイルへのパスを指定. 追加されたバージョン: MySQL 8.0.14.
- [clone_ssl_key](#): 秘密キーファイルへのパスを指定. 追加されたバージョン: MySQL 8.0.14.
- [clone_valid_donor_list](#): リモートクローニング操作のドナーホストアドレスを定義. 追加されたバージョン: MySQL 8.0.17.
- [connection_control_failed_connections_threshold](#): 遅延が発生する前の連続した接続試行の失敗. 追加されたバージョン: MySQL 8.0.1.
- [connection_control_max_connection_delay](#): 失敗した接続試行に対するサーバーレスポンスの最大遅延 (ミリ秒). 追加されたバージョン: MySQL 8.0.1.
- [connection_control_min_connection_delay](#): 失敗した接続試行に対するサーバーレスポンスの最小遅延 (ミリ秒). 追加されたバージョン: MySQL 8.0.1.
- [create_admin_listener_thread](#): 管理インタフェース上の接続に専用リスニングスレッドを使用するかどうか. 追加されたバージョン: MySQL 8.0.14.
- [cte_max_recursion_depth](#): 共通テーブル式の最大再帰深度. 追加されたバージョン: MySQL 8.0.3.
- [ddl-rewriter](#): `ddl_rewriter` プラグインをアクティブ化するかどうか. 追加されたバージョン: MySQL 8.0.16.
- [default_collation_for_utf8mb4](#): `utf8mb4` 文字セットのデフォルトの照合順序. 追加されたバージョン: MySQL 8.0.11.
- [default_table_encryption](#): デフォルトのスキーマおよびテーブルスペースの暗号化設定. 追加されたバージョン: MySQL 8.0.16.
- [dagnet.Status](#): `dagnet.log_error_filter_rules` への最新の割当ての結果. 追加されたバージョン: MySQL 8.0.12.

- [dragnet.log_error_filter_rules](#): エラーロギングのフィルタールール. 追加されたバージョン: MySQL 8.0.4.
- [early-plugin-load](#): 必須の組み込みプラグインをロードする前、およびストレージエンジンの初期化前にロードするプラグインを指定. 追加されたバージョン: MySQL 8.0.0.
- [generated_random_password_length](#): 生成されるパスワードの最大長. 追加されたバージョン: MySQL 8.0.18.
- [group_replication_advertise_recovery_endpoints](#): 分散リカバリ用に提供される接続. 追加されたバージョン: MySQL 8.0.21.
- [group_replication_autorejoin_tries](#): メンバーがグループに自動的に再参加しようとした回数. 追加されたバージョン: MySQL 8.0.16.
- [group_replication_clone_threshold](#): ドナーと受信者の間のトランザクション番号のギャップ。このギャップを超えると、リモートクローニング操作が状態転送に使用されます. 追加されたバージョン: MySQL 8.0.17.
- [group_replication_communication_debug_options](#): グループレプリケーションコンポーネントのデバッグメッセージのレベル. 追加されたバージョン: MySQL 8.0.3.
- [group_replication_communication_max_message_size](#): グループレプリケーション通信の最大メッセージサイズ。大きいメッセージは断片化されます. 追加されたバージョン: MySQL 8.0.16.
- [group_replication_consistency](#): どのグループが提供するかを保証するトランザクション一貫性のタイプ. 追加されたバージョン: MySQL 8.0.14.
- [group_replication_exit_state_action](#): グループから離れる場合のインスタンスの動作. 追加されたバージョン: MySQL 8.0.12.
- [group_replication_flow_control_hold_percent](#): 未使用のままにするグループ割当て制限の割合. 追加されたバージョン: MySQL 8.0.2.
- [group_replication_flow_control_max_commit_quota](#): グループの最大フロー制御割当て制限. 追加されたバージョン: MySQL 8.0.2.
- [group_replication_flow_control_member_quota_percent](#): メンバーが目標の計算時に使用可能であるとみなす目標の割合. 追加されたバージョン: MySQL 8.0.2.
- [group_replication_flow_control_min_quota](#): メンバーごとに割り当てることができる最低フロー制御目標. 追加されたバージョン: MySQL 8.0.2.
- [group_replication_flow_control_min_recovery_quota](#): 別のグループメンバーがリカバリしているため、メンバーごとに割り当てることができる最小割当て容量. 追加されたバージョン: MySQL 8.0.2.
- [group_replication_flow_control_period](#): フロー制御の反復間で待機する秒数を定義. 追加されたバージョン: MySQL 8.0.2.
- [group_replication_flow_control_release_percent](#): フロー制御でライターメンバーを制限する必要がなくなった場合のグループ割当て制限の解放方法. 追加されたバージョン: MySQL 8.0.2.
- [group_replication_ip_allowlist](#): グループへの接続が許可されているホストのリスト (MySQL 8.0.22 以降). 追加されたバージョン: MySQL 8.0.22.
- [group_replication_member_expel_timeout](#): グループメンバーの障害が疑われ、グループから削除されてグループメンバーシップが再構成されるまでの時間. 追加されたバージョン: MySQL 8.0.13.
- [group_replication_member_weight](#): このメンバーがプライマリとして選択される可能性. 追加されたバージョン: MySQL 8.0.2.
- [group_replication_message_cache_size](#): グループ通信エンジンメッセージキャッシュの最大メモリー (XCom). 追加されたバージョン: MySQL 8.0.16.
- [group_replication_recovery_compression_algorithm](#): 送信リカバリ接続に許可される圧縮アルゴリズム. 追加されたバージョン: MySQL 8.0.18.
- [group_replication_recovery_get_public_key](#): ドナーからの公開キーのフェッチに関するプリファレンスを受け入れるかどうか. 追加されたバージョン: MySQL 8.0.4.

- [group_replication_recovery_public_key_path](#): 公開キー情報を受け入れるには、追加されたバージョン: MySQL 8.0.4.
- [group_replication_recovery_tls_ciphersuites](#): クライアントとしてのこのインスタンスとの接続暗号化に TLSv1.3 が使用されている場合に許可される暗号スイート (メンバーに参加). 追加されたバージョン: MySQL 8.0.19.
- [group_replication_recovery_tls_version](#): クライアントとしての接続暗号化に許可された TLS プロトコル (参加メンバー). 追加されたバージョン: MySQL 8.0.19.
- [group_replication_recovery_zstd_compression_level](#): zstd 圧縮を使用するリカバリ接続の圧縮レベル. 追加されたバージョン: MySQL 8.0.18.
- [group_replication_tls_source](#): グループレプリケーションの TLS マテリアルのソース. 追加されたバージョン: MySQL 8.0.21.
- [group_replication_unreachable_majority_timeout](#): 少数民族がグループから離れるまで待機する時間. 追加されたバージョン: MySQL 8.0.2.
- [histogram_generation_max_mem_size](#): ヒストグラム統計を作成するための最大メモリー. 追加されたバージョン: MySQL 8.0.2.
- [immediate_server_version](#): 即時レプリケーションソースであるサーバーの MySQL Server リリース番号. 追加されたバージョン: MySQL 8.0.14.
- [information_schema_stats_expiry](#): キャッシュされたテーブル統計の有効期限設定. 追加されたバージョン: MySQL 8.0.3.
- [innodb_buffer_pool_debug](#): バッファプールのサイズが 1GB 未満の場合に、複数のバッファプールインスタンスを許可. 追加されたバージョン: MySQL 8.0.0.
- [innodb_buffer_pool_in_core_file](#): コアファイルへのバッファプールページの書き込みを制御. 追加されたバージョン: MySQL 8.0.14.
- [innodb_checkpoint_disabled](#): 故意のサーバイグジットが常にリカバリを開始するようにチェックポイントを無効にします. 追加されたバージョン: MySQL 8.0.2.
- [innodb_ddl_log_crash_reset_debug](#): DDL ログクラッシュインサーションカウンタをリセットするデバッグオプション. 追加されたバージョン: MySQL 8.0.3.
- [innodb_deadlock_detect](#): デッドロック検出を有効または無効にします. 追加されたバージョン: MySQL 8.0.0.
- [innodb_dedicated_server](#): バッファプールサイズ、ログファイルサイズ、およびフラッシュ方法の自動構成を有効にします. 追加されたバージョン: MySQL 8.0.3.
- [innodb_directories](#): 起動時にテーブルスペースデータファイルをスキャンするディレクトリを定義. 追加されたバージョン: MySQL 8.0.4.
- [innodb_doublewrite_batch_size](#): バッチごとに書き込む二重書き込みページの数. 追加されたバージョン: MySQL 8.0.20.
- [innodb_doublewrite_dir](#): 二重書き込みバッファファイルディレクトリ. 追加されたバージョン: MySQL 8.0.20.
- [innodb_doublewrite_files](#): 二重書き込みファイルの数. 追加されたバージョン: MySQL 8.0.20.
- [innodb_doublewrite_pages](#): スレッド当たりの二重書き込みページ数. 追加されたバージョン: MySQL 8.0.20.
- [innodb_extend_and_initialize](#): Linux での新しいテーブルスペースページの割当て方法を制御. 追加されたバージョン: MySQL 8.0.22.
- [innodb_fsync_threshold](#): 新規ファイルの作成時に InnoDB が fsync をコールする頻度を制御. 追加されたバージョン: MySQL 8.0.13.
- [innodb_idle_flush_pct](#): InnoDB がアイドル状態の場合の I/O 操作の制限. 追加されたバージョン: MySQL 8.0.18.
- [innodb_log_checkpoint_fuzzy_now](#): InnoDB にファジーチェックポイントの書き込みを強制するデバッグオプション. 追加されたバージョン: MySQL 8.0.13.

- [innodb_log_spin_cpu_abs_lwm](#): フラッシュされた redo の待機中にユーザースレッドがスピンしなくなる最小 CPU 使用量. 追加されたバージョン: MySQL 8.0.11.
- [innodb_log_spin_cpu_pct_hwm](#): フラッシュされた redo の待機中にユーザースレッドがスピンしなくなる CPU 使用率の最大量. 追加されたバージョン: MySQL 8.0.11.
- [innodb_log_wait_for_flush_spin_hwm](#): フラッシュされた redo の待機中にユーザースレッドがスピンしなくなる最大平均ログフラッシュ時間. 追加されたバージョン: MySQL 8.0.11.
- [innodb_log_writer_threads](#): redo ログの書き込みおよびフラッシュ専用のログライタースレッドを有効にします. 追加されたバージョン: MySQL 8.0.22.
- [innodb_parallel_read_threads](#): パラレルインデックス読取りのスレッド数. 追加されたバージョン: MySQL 8.0.14.
- [innodb_print_ddl_logs](#): DDL ログをエラーログに出力するかどうか. 追加されたバージョン: MySQL 8.0.3.
- [innodb_redo_log_archive_dirs](#): ラベル付き redo ログアーカイブディレクトリ. 追加されたバージョン: MySQL 8.0.17.
- [innodb_redo_log_encrypt](#): 暗号化されたテーブルスペースの redo ログデータの暗号化を制御. 追加されたバージョン: MySQL 8.0.1.
- [innodb_scan_directories](#): InnoDB リカバリ中にテーブルスペースファイルをスキャンするディレクトリを定義. 追加されたバージョン: MySQL 8.0.2.
- [innodb_spin_wait_pause_multiplier](#): スピン待機ループで PAUSE 命令の数を決定するために使用される乗数値. 追加されたバージョン: MySQL 8.0.16.
- [innodb_stats_include_delete_marked](#): 永続的な InnoDB 統計の計算時に削除マーク付きレコードを含める. 追加されたバージョン: MySQL 8.0.1.
- [innodb_temp_tablespace_dir](#): セッション一時テーブルスペースのパス. 追加されたバージョン: MySQL 8.0.13.
- [innodb_tmpdir](#): オンライン ALTER TABLE 操作中に作成された一時テーブルファイルのディレクトリの場所. 追加されたバージョン: MySQL 8.0.0.
- [innodb_undo_log_encrypt](#): 暗号化テーブルスペースの undo ログデータの暗号化を制御. 追加されたバージョン: MySQL 8.0.1.
- [innodb_validate_tablespace_paths](#): 起動時のテーブルスペースパス検証を有効にします. 追加されたバージョン: MySQL 8.0.21.
- [internal_tmp_mem_storage_engine](#): 内部インメモリー一時テーブルに使用するストレージエンジン. 追加されたバージョン: MySQL 8.0.2.
- [keyring-migration-destination](#): 鍵移行先キーリングプラグイン. 追加されたバージョン: MySQL 8.0.4.
- [keyring-migration-host](#): キー移行のために実行中のサーバーに接続するためのホスト名. 追加されたバージョン: MySQL 8.0.4.
- [keyring-migration-password](#): キー移行のために実行中のサーバーに接続するためのパスワード. 追加されたバージョン: MySQL 8.0.4.
- [keyring-migration-port](#): キー移行のために実行中のサーバーに接続するための TCP/IP ポート番号. 追加されたバージョン: MySQL 8.0.4.
- [keyring-migration-socket](#): キー移行のために実行中のサーバーに接続するための Unix ソケットファイルまたは Windows 名前付きパイプ. 追加されたバージョン: MySQL 8.0.4.
- [keyring-migration-source](#): キー移行ソースキーリングプラグイン. 追加されたバージョン: MySQL 8.0.4.
- [keyring-migration-user](#): キー移行のために実行中のサーバーに接続するためのユーザー名. 追加されたバージョン: MySQL 8.0.4.
- [keyring_aws_cmek_id](#): AWS キーリングプラグインの顧客マスターキー ID 値. 追加されたバージョン: MySQL 8.0.11.
- [keyring_aws_conf_file](#): AWS キーリングプラグイン構成ファイルの場所. 追加されたバージョン: MySQL 8.0.11.

- [keyring_aws_data_file](#): AWS キーリングプラグインの記憶域ファイルの場所. 追加されたバージョン: MySQL 8.0.11.
- [keyring_aws_region](#): AWS キーリングプラグインリージョン. 追加されたバージョン: MySQL 8.0.11.
- [keyring_encrypted_file_data](#): [keyring_encrypted_file](#) プラグインデータファイル. 追加されたバージョン: MySQL 8.0.11.
- [keyring_encrypted_file_password](#): [keyring_encrypted_file](#) プラグインパスワード. 追加されたバージョン: MySQL 8.0.11.
- [keyring_hashicorp_auth_path](#): HashiCorp Vault AppRole 認証パス. 追加されたバージョン: MySQL 8.0.18.
- [keyring_hashicorp_ca_path](#): [keyring_hashicorp](#) CA ファイルへのパス. 追加されたバージョン: MySQL 8.0.18.
- [keyring_hashicorp_caching](#): [keyring_hashicorp](#) キャッシュを有効にするかどうか. 追加されたバージョン: MySQL 8.0.18.
- [keyring_hashicorp_commit_auth_path](#): 使用中の [keyring_hashicorp_auth_path](#) 値. 追加されたバージョン: MySQL 8.0.18.
- [keyring_hashicorp_commit_ca_path](#): 使用中の [keyring_hashicorp_ca_path](#) 値. 追加されたバージョン: MySQL 8.0.18.
- [keyring_hashicorp_commit_caching](#): 使用中の [keyring_hashicorp_caching](#) 値. 追加されたバージョン: MySQL 8.0.18.
- [keyring_hashicorp_commit_role_id](#): 使用中の [keyring_hashicorp_role_id](#) 値. 追加されたバージョン: MySQL 8.0.18.
- [keyring_hashicorp_commit_server_url](#): 使用中の [keyring_hashicorp_server_url](#) 値. 追加されたバージョン: MySQL 8.0.18.
- [keyring_hashicorp_commit_store_path](#): 使用中の [keyring_hashicorp_store_path](#) 値. 追加されたバージョン: MySQL 8.0.18.
- [keyring_hashicorp_role_id](#): HashiCorp Vault AppRole 認証ロール ID. 追加されたバージョン: MySQL 8.0.18.
- [keyring_hashicorp_secret_id](#): HashiCorp Vault AppRole 認証シークレット ID. 追加されたバージョン: MySQL 8.0.18.
- [keyring_hashicorp_server_url](#): HashiCorp Vault サーバー URL. 追加されたバージョン: MySQL 8.0.18.
- [keyring_hashicorp_store_path](#): HashiCorp Vault ストアパス. 追加されたバージョン: MySQL 8.0.18.
- [keyring_oci_ca_certificate](#): ピア認証用の CA 証明書ファイル. 追加されたバージョン: MySQL 8.0.22.
- [keyring_oci_compartment](#): OCI コンパートメント OCID. 追加されたバージョン: MySQL 8.0.22.
- [keyring_oci_encryption_endpoint](#): OCI 暗号化サーバーエンドポイント. 追加されたバージョン: MySQL 8.0.22.
- [keyring_oci_key_file](#): OCI RSA 秘密キーファイル. 追加されたバージョン: MySQL 8.0.22.
- [keyring_oci_key_fingerprint](#): OCI RSA 秘密キーファイルのフィンガープリント. 追加されたバージョン: MySQL 8.0.22.
- [keyring_oci_management_endpoint](#): OCI 管理サーバーエンドポイント. 追加されたバージョン: MySQL 8.0.22.
- [keyring_oci_master_key](#): OCI マスターキー OCID. 追加されたバージョン: MySQL 8.0.22.
- [keyring_oci_secrets_endpoint](#): OCI シークレットサーバーエンドポイント. 追加されたバージョン: MySQL 8.0.22.
- [keyring_oci_tenancy](#): OCI テナンス OCID. 追加されたバージョン: MySQL 8.0.22.
- [keyring_oci_user](#): OCI ユーザー OCID. 追加されたバージョン: MySQL 8.0.22.
- [keyring_oci_vaults_endpoint](#): OCI ボールトサーバーエンドポイント. 追加されたバージョン: MySQL 8.0.22.
- [keyring_oci_virtual_vault](#): OCI ボールト OCID. 追加されたバージョン: MySQL 8.0.22.
- [keyring_okv_conf_dir](#): Oracle Key Vault キーリングプラグイン構成ディレクトリ. 追加されたバージョン: MySQL 8.0.11.

- [keyring_operations](#): キーリング操作が有効かどうか. 追加されたバージョン: MySQL 8.0.4.
- [lock_order](#): 実行時に LOCK_ORDER ツールを有効にするかどうか. 追加されたバージョン: MySQL 8.0.17.
- [lock_order_debug_loop](#): LOCK_ORDER ツールでループとしてフラグ付けされた依存性が検出されたときにデバッグアサートを実行するかどうか. 追加されたバージョン: MySQL 8.0.17.
- [lock_order_debug_missing_arc](#): LOCK_ORDER ツールで宣言されていない依存性が検出されたときにデバッグアサートを実行するかどうか. 追加されたバージョン: MySQL 8.0.17.
- [lock_order_debug_missing_key](#): パフォーマンススキーマで適切に計測されていないオブジェクトが LOCK_ORDER ツールで検出されたときにデバッグ表明を行うかどうか. 追加されたバージョン: MySQL 8.0.17.
- [lock_order_debug_missing_unlock](#): まだ保持されている間に破棄されたロックが LOCK_ORDER ツールで検出されたときにデバッグアサートを実行するかどうか. 追加されたバージョン: MySQL 8.0.17.
- [lock_order_dependencies](#): lock_order_dependencies.txt ファイルへのパス. 追加されたバージョン: MySQL 8.0.17.
- [lock_order_extra_dependencies](#): 2 番目の依存ファイルへのパス. 追加されたバージョン: MySQL 8.0.17.
- [lock_order_output_directory](#): LOCK_ORDER ツールがログを書き込むディレクトリ. 追加されたバージョン: MySQL 8.0.17.
- [lock_order_print_txt](#): ロック順グラフ分析を実行し、テキストレポートを印刷するかどうか. 追加されたバージョン: MySQL 8.0.17.
- [lock_order_trace_loop](#): LOCK_ORDER ツールでループとしてフラグ付けされた依存性が検出された場合にログファイルのトレースを出力するかどうか. 追加されたバージョン: MySQL 8.0.17.
- [lock_order_trace_missing_arc](#): LOCK_ORDER ツールで宣言されていない依存性が検出された場合にログファイルのトレースを出力するかどうか. 追加されたバージョン: MySQL 8.0.17.
- [lock_order_trace_missing_key](#): LOCK_ORDER ツールがパフォーマンススキーマで正しく計測されていないオブジェクトを検出したときにログファイルのトレースを出力するかどうか. 追加されたバージョン: MySQL 8.0.17.
- [lock_order_trace_missing_unlock](#): まだ保持されている間に破棄されたロックが LOCK_ORDER ツールで検出された場合にログファイルのトレースを出力するかどうか. 追加されたバージョン: MySQL 8.0.17.
- [log_error_filter_rules](#): エラーロギングのフィルタールール. 追加されたバージョン: MySQL 8.0.2.
- [log_error_services](#): エラーロギングに使用するコンポーネント. 追加されたバージョン: MySQL 8.0.2.
- [log_error_suppression_list](#): 抑制する警告/情報エラーログメッセージ. 追加されたバージョン: MySQL 8.0.13.
- [log_slow_extra](#): スロークエリログファイルに追加情報を書き込むかどうか. 追加されたバージョン: MySQL 8.0.14.
- [mandatory_roles](#): すべてのユーザーに自動的に付与されるロール. 追加されたバージョン: MySQL 8.0.2.
- [mysql_firewall_mode](#): MySQL Enterprise Firewall が動作しているかどうか. 追加されたバージョン: MySQL 8.0.11.
- [mysql_firewall_trace](#): ファイアウォールトレースを有効にするかどうか. 追加されたバージョン: MySQL 8.0.11.
- [mysqlx](#): X Plugin を初期化するかどうか. 追加されたバージョン: MySQL 8.0.11.
- [mysqlx_compression_algorithms](#): X Protocol 接続で許可される圧縮アルゴリズム. 追加されたバージョン: MySQL 8.0.19.
- [mysqlx_deflate_default_compression_level](#): X Protocol 接続での Deflate アルゴリズムのデフォルトの圧縮レベル. 追加されたバージョン: MySQL 8.0.20.
- [mysqlx_deflate_max_client_compression_level](#): X Protocol 接続で Deflate アルゴリズムに許可される最大圧縮レベル. 追加されたバージョン: MySQL 8.0.20.
- [mysqlx_interactive_timeout](#): 対話型クライアントがタイムアウトするまで待機する秒数. 追加されたバージョン: MySQL 8.0.4.

- [mysqlx_lz4_default_compression_level](#): X Protocol 接続での LZ4 アルゴリズムのデフォルトの圧縮レベル. 追加されたバージョン: MySQL 8.0.20.
- [mysqlx_lz4_max_client_compression_level](#): X Protocol 接続での LZ4 アルゴリズムの最大許容圧縮レベル. 追加されたバージョン: MySQL 8.0.20.
- [mysqlx_read_timeout](#): 読取り操作のブロックが完了するまで待機する秒数. 追加されたバージョン: MySQL 8.0.4.
- [mysqlx_wait_timeout](#): 接続からのアクティビティを待機する秒数. 追加されたバージョン: MySQL 8.0.4.
- [mysqlx_write_timeout](#): 書き込み操作のブロックが完了するまで待機する秒数. 追加されたバージョン: MySQL 8.0.4.
- [mysqlx_zstd_default_compression_level](#): X Protocol 接続での zstd アルゴリズムのデフォルトの圧縮レベル. 追加されたバージョン: MySQL 8.0.20.
- [mysqlx_zstd_max_client_compression_level](#): X Protocol 接続で zstd アルゴリズムに許可される最大圧縮レベル. 追加されたバージョン: MySQL 8.0.20.
- [named_pipe_full_access_group](#): 名前付きパイプへのフルアクセス権を付与された Windows グループの名前. 追加されたバージョン: MySQL 8.0.14.
- [no-dd-upgrade](#): 起動時のデータディクショナリテーブルの自動アップグレードの防止. 追加されたバージョン: MySQL 8.0.4.
- [no-monitor](#): RESTART に必要なモニタープロセスをフォークしない. 追加されたバージョン: MySQL 8.0.12.
- [original_commit_timestamp](#): トランザクションが元のソースでコミットされた時刻. 追加されたバージョン: MySQL 8.0.1.
- [original_server_version](#): トランザクションが最初にコミットされたサーバーの MySQL Server リリース番号. 追加されたバージョン: MySQL 8.0.14.
- [partial_revokes](#): 部分失効が有効かどうか. 追加されたバージョン: MySQL 8.0.16.
- [password_history](#): パスワードの再利用前に必要なパスワード変更の数. 追加されたバージョン: MySQL 8.0.3.
- [password_require_current](#): パスワード変更に現在のパスワード検証が必要かどうか. 追加されたバージョン: MySQL 8.0.13.
- [password_reuse_interval](#): パスワードを再利用するまでに必要な経過日数. 追加されたバージョン: MySQL 8.0.3.
- [performance_schema_max_digest_sample_age](#): クエリーの再サンプリング期間 (秒). 追加されたバージョン: MySQL 8.0.3.
- [performance_schema_show_processlist](#): SHOW PROCESSLIST 実装の選択. 追加されたバージョン: MySQL 8.0.22.
- [persist_only_admin_x509_subject](#): 永続的に制限されたシステム変数の永続化を可能にする SSL 証明書 X.509 サブジェクト. 追加されたバージョン: MySQL 8.0.14.
- [persisted_globals_load](#): 永続構成設定をロードするかどうか. 追加されたバージョン: MySQL 8.0.0.
- [print_identified_with_as_hex](#): SHOW CREATE USER では、16 進数で印刷不可能な文字を含むハッシュ値を出力. 追加されたバージョン: MySQL 8.0.17.
- [protocol_compression_algorithms](#): 着信接続に許可される圧縮アルゴリズム. 追加されたバージョン: MySQL 8.0.18.
- [regex_stack_limit](#): 正規表現一致スタックサイズ制限. 追加されたバージョン: MySQL 8.0.4.
- [regex_time_limit](#): 正規表現一致タイムアウト. 追加されたバージョン: MySQL 8.0.4.
- [replication_optimize_for_static_plugin_config](#): 準同期レプリケーションの共有ロック. 追加されたバージョン: MySQL 8.0.23.
- [replication_sender_observe_commit_only](#): 準同期レプリケーションのための制限付きコールバック. 追加されたバージョン: MySQL 8.0.23.
- [require_row_format](#): 内部サーバーで使用. 追加されたバージョン: MySQL 8.0.19.

- [resultset_metadata](#): サーバーが結果セットメタデータを返すかどうか. 追加されたバージョン: MySQL 8.0.3.
- [rpl_read_size](#): バイナリログファイルおよびリレーログファイルから読み取られるデータの最小量をバイト単位で設定. 追加されたバージョン: MySQL 8.0.11.
- [secondary_engine_cost_threshold](#): セカンダリエンジンへのクエリーオフロードの最適化コストしきい値. 追加されたバージョン: MySQL 8.0.16.
- [select_into_buffer_size](#): OUTFILE または DUMPFILE エクスポートファイルに使用されるバッファのサイズ. `read_buffer_size` をオーバーライド. 追加されたバージョン: MySQL 8.0.22.
- [select_into_disk_sync](#): OUTFILE または DUMPFILE エクスポートファイルのバッファをフラッシュした後、データをストレージデバイスと同期します。OFF にすると、同期が無効になり、デフォルト値になります。追加されたバージョン: MySQL 8.0.22.
- [select_into_disk_sync_delay](#): `select_into_sync_disk = ON` の場合、OUTFILE または DUMPFILE エクスポートファイルバッファの各同期後の遅延をミリ秒単位で設定します。それ以外の場合は無効です。追加されたバージョン: MySQL 8.0.22.
- [show_create_table_skip_secondary_engine](#): SHOW CREATE TABLE 出力から SECONDARY ENGINE 句を除外するかどうか. 追加されたバージョン: MySQL 8.0.18.
- [show_create_table_verbosity](#): SHOW CREATE TABLE にデフォルト値がある場合でも ROW_FORMAT を表示するかどうか. 追加されたバージョン: MySQL 8.0.11.
- [sql_require_primary_key](#): テーブルに主キーが必要かどうか. 追加されたバージョン: MySQL 8.0.13.
- [ssl_fips_mode](#): サーバー側で FIPS モードを有効にするかどうか. 追加されたバージョン: MySQL 8.0.11.
- [syseventlog.facility](#): syslog メッセージの機能. 追加されたバージョン: MySQL 8.0.13.
- [syseventlog.include_pid](#): サーバー PID を syslog メッセージに含めるかどうか. 追加されたバージョン: MySQL 8.0.13.
- [syseventlog.tag](#): syslog メッセージ内のサーバー識別子のタグ. 追加されたバージョン: MySQL 8.0.13.
- [table_encryption_privilege_check](#): TABLE_ENCRYPTION_ADMIN 権限チェックを有効にします。追加されたバージョン: MySQL 8.0.16.
- [temptable_max_mmap](#): TempTable ストレージエンジンがメモリーマップされた一時ファイルから割り当てることができるメモリーの最大量. 追加されたバージョン: MySQL 8.0.23.
- [temptable_max_ram](#): データがディスクに格納される前に TempTable ストレージエンジンが占有できるメモリーの最大量を定義. 追加されたバージョン: MySQL 8.0.2.
- [temptable_use_mmap](#): `temptable_max_ram` しきい値に達したときに、TempTable ストレージエンジンがメモリーマップされたファイルを割り当てることができるかどうかを定義. 追加されたバージョン: MySQL 8.0.16.
- [thread_pool_algorithm](#): スレッドプールアルゴリズム. 追加されたバージョン: MySQL 8.0.11.
- [thread_pool_high_priority_connection](#): 現在のセッションの優先度が高いかどうか. 追加されたバージョン: MySQL 8.0.11.
- [thread_pool_max_active_query_threads](#): グループ当たりのアクティブなクエリースレッドの最大許容数. 追加されたバージョン: MySQL 8.0.19.
- [thread_pool_max_unused_threads](#): 未使用スレッドの最大許容数. 追加されたバージョン: MySQL 8.0.11.
- [thread_pool_prio_kickup_timer](#): ステートメントが優先度の高い実行に移動されるまでの期間. 追加されたバージョン: MySQL 8.0.11.
- [thread_pool_size](#): スレッドプール内のスレッドグループの数. 追加されたバージョン: MySQL 8.0.11.
- [thread_pool_stall_limit](#): ステートメントがストールとして定義されるまでの期間. 追加されたバージョン: MySQL 8.0.11.
- [tls_ciphersuites](#): 暗号化された接続に許可される TLSv1.3 暗号スイート. 追加されたバージョン: MySQL 8.0.16.

- `upgrade`: 起動時の自動アップグレードの制御. 追加されたバージョン: MySQL 8.0.16.
- `use_secondary_engine`: セカンダリエンジンを使用してクエリーを実行するかどうか. 追加されたバージョン: MySQL 8.0.13.
- `validate-config`: サーバー構成の検証. 追加されたバージョン: MySQL 8.0.16.
- `validate_password.check_user_name`: パスワードをユーザー名と照合するかどうか. 追加されたバージョン: MySQL 8.0.4.
- `validate_password.dictionary_file`: `validate_password` 辞書ファイル. 追加されたバージョン: MySQL 8.0.4.
- `validate_password.dictionary_file_last_parsed`: デイクシオナリファイルの最終解析日. 追加されたバージョン: MySQL 8.0.4.
- `validate_password.dictionary_file_words_count`: 辞書ファイル内の単語数. 追加されたバージョン: MySQL 8.0.4.
- `validate_password.length`: `validate_password` 必要なパスワード長. 追加されたバージョン: MySQL 8.0.4.
- `validate_password.mixed_case_count`: `validate_password` 必要な大文字/小文字の数. 追加されたバージョン: MySQL 8.0.4.
- `validate_password.number_count`: `validate_password` 必要な数字の数. 追加されたバージョン: MySQL 8.0.4.
- `validate_password.policy`: `validate_password` パスワードポリシー. 追加されたバージョン: MySQL 8.0.4.
- `validate_password.special_char_count`: `validate_password` 必要な特殊文字の数. 追加されたバージョン: MySQL 8.0.4.
- `version_compile_zlib`: コンパイル済み `zlib` ライブラリのバージョン. 追加されたバージョン: MySQL 8.0.11.
- `windowing_use_high_precision`: ウィンドウ関数を高精度に計算するかどうか. 追加されたバージョン: MySQL 8.0.2.

非推奨となったオプションおよび変数: MySQL 8.0

次のシステム変数、ステータス変数およびオプションが非推奨になりました: MySQL 8.0.

- `Compression`: クライアント接続がクライアント/サーバープロトコルで圧縮を使用するかどうか. 非推奨になったバージョン: MySQL 8.0.18.
- `expire_logs_days`: この日数後にバイナリログをパージ. 非推奨になったバージョン: MySQL 8.0.3.
- `group_replication_ip_whitelist`: グループへの接続が許可されているホストのリスト. 非推奨になったバージョン: MySQL 8.0.22.
- `innodb_undo_tablespaces`: ロールバックセグメントが分割されるテーブルスペースファイルの数. 非推奨になったバージョン: MySQL 8.0.4.
- `log_bin_use_v1_row_events`: サーバーがバージョン 1 バイナリログ行イベントを使用しているかどうか. 非推奨になったバージョン: MySQL 8.0.18.
- `log_syslog`: `syslog` にエラーメッセージを書き込むかどうか. 非推奨になったバージョン: MySQL 8.0.2.
- `master-info-file`: ソースを記憶し、I/O レプリケーションスレッドがソースバイナリログ内にあるファイルの場所と名前. 非推奨になったバージョン: MySQL 8.0.18.
- `master_info_repository`: ソースバイナリログ内のソース情報およびレプリケーション I/O スレッドの場所を含む接続メタデータリポジトリをファイルまたはテーブルに書き込むかどうか. 非推奨になったバージョン: MySQL 8.0.23.
- `max_length_for_sort_data`: ソートされたレコード内のバイト数. 非推奨になったバージョン: MySQL 8.0.20.
- `no-dd-upgrade`: 起動時のデータデイクシオナリテーブルの自動アップグレードの防止. 非推奨になったバージョン: MySQL 8.0.16.
- `relay_log_info_file`: レプリカがリレーログに関する情報を記録するアプライアンスのメタデータリポジトリのファイル名. 非推奨になったバージョン: MySQL 8.0.18.

- [relay_log_info_repository](#): リレーログ内のレプリケーション SQL スレッドの場所をファイルまたはテーブルに書き込むかどうか。非推奨になったバージョン: MySQL 8.0.23.
- [slave_compressed_protocol](#): ソース/レプリカプロトコルの圧縮の使用。非推奨になったバージョン: MySQL 8.0.18.
- [slave_rows_search_algorithms](#): レプリカのバッチ更新に使用される検索アルゴリズムを決定します。このリストの任意の 2 または 3: INDEX_SEARCH, TABLE_SCAN, HASH_SCAN。非推奨になったバージョン: MySQL 8.0.18.
- [symbolic-links](#): MyISAM テーブルのシンボリックリンクの許可。非推奨になったバージョン: MySQL 8.0.2.

削除されたオプションおよび変数: MySQL 8.0

次のシステム変数、ステータス変数およびオプションが削除されました: MySQL 8.0.

- [Com_alter_db_upgrade](#): ALTER DATABASE ... UPGRADE DATA DIRECTORY NAME ステートメントの数。削除されたバージョン: MySQL 8.0.0.
- [InnoDB_available_undo_logs](#): InnoDB ロールバックセグメントの合計数。アクティブなロールバックセグメントの数を表示する `innodb_rollback_segments` とは異なります。削除されたバージョン: MySQL 8.0.2.
- [Qcache_free_blocks](#): クエリーキャッシュ内の空きメモリーブロック数。削除されたバージョン: MySQL 8.0.3.
- [Qcache_free_memory](#): クエリーキャッシュの空きメモリー量。削除されたバージョン: MySQL 8.0.3.
- [Qcache_hits](#): クエリーキャッシュヒットの数。削除されたバージョン: MySQL 8.0.3.
- [Qcache_inserts](#): クエリーキャッシュ挿入の数。削除されたバージョン: MySQL 8.0.3.
- [Qcache_lowmem_prunes](#): キャッシュ内の空きメモリー不足のためにクエリーキャッシュから削除されたクエリーの数。削除されたバージョン: MySQL 8.0.3.
- [Qcache_not_cached](#): キャッシュされていないクエリーの数 (`query_cache_type` 設定のためキャッシュ不可またはキャッシュされていません)。削除されたバージョン: MySQL 8.0.3.
- [Qcache_queries_in_cache](#): クエリーキャッシュに登録されたクエリーの数。削除されたバージョン: MySQL 8.0.3.
- [Qcache_total_blocks](#): クエリーキャッシュ内のブロックの合計数。削除されたバージョン: MySQL 8.0.3.
- [Slave_heartbeat_period](#): レプリカレプリケーションハートビート間隔 (秒)。削除されたバージョン: MySQL 8.0.1.
- [Slave_last_heartbeat](#): 最新のハートビート信号が受信された時期を `TIMESTAMP` 形式で示します。削除されたバージョン: MySQL 8.0.1.
- [Slave_received_heartbeats](#): 前回のリセット以降にレプリカが受信したハートビートの数。削除されたバージョン: MySQL 8.0.1.
- [Slave_retried_transactions](#): 起動後にレプリケーション SQL スレッドがトランザクションを再試行した合計回数。削除されたバージョン: MySQL 8.0.1.
- [Slave_running](#): このサーバーのレプリカとしての状態 (レプリケーション I/O スレッドステータス)。削除されたバージョン: MySQL 8.0.1.
- [bootstrap](#): MySQL インストールスクリプトが使用。削除されたバージョン: MySQL 8.0.0.
- [date_format](#): DATE 書式 (未使用)。削除されたバージョン: MySQL 8.0.3.
- [datetime_format](#): DATETIME/TIMESTAMP 形式 (未使用)。削除されたバージョン: MySQL 8.0.3.
- [des-key-file](#): 指定されたファイルから、`des_encrypt()` および `des_encrypt` の鍵をロード。削除されたバージョン: MySQL 8.0.3.
- [group_replication_allow_local_disjoint_gtids_join](#): グループに存在しないトランザクションがある場合でも、現在のサーバーがグループに参加できるようにします。削除されたバージョン: MySQL 8.0.4.
- [have_crypt](#): `crypt()` システムコールの可用性。削除されたバージョン: MySQL 8.0.3.
- [ignore-db-dir](#): ディレクトリを非データベースディレクトリとして処理。削除されたバージョン: MySQL 8.0.0.

- [ignore_builtin_innodb](#): 組み込み InnoDB を無視。削除されたバージョン: MySQL 8.0.3.
- [ignore_db_dirs](#): 非データベースディレクトリとして処理されるディレクトリ。削除されたバージョン: MySQL 8.0.0.
- [innodb_checksums](#): InnoDB チェックサムの検証を有効化。削除されたバージョン: MySQL 8.0.0.
- [innodb_disable_resize_buffer_pool_debug](#): InnoDB バッファプールのサイズ変更を無効にします。削除されたバージョン: MySQL 8.0.0.
- [innodb_file_format](#): 新しい InnoDB テーブルの形式。削除されたバージョン: MySQL 8.0.0.
- [innodb_file_format_check](#): InnoDB がファイル形式の互換性の確認を行うかどうか。削除されたバージョン: MySQL 8.0.0.
- [innodb_file_format_max](#): 共有テーブルスペースのファイル形式タグ。削除されたバージョン: MySQL 8.0.0.
- [innodb_large_prefix](#): カラムプリフィクスインデックスの長いキーを有効化。削除されたバージョン: MySQL 8.0.0.
- [innodb_locks_unsafe_for_binlog](#): InnoDB がネクストキーロックを使用しないことを強制。代わりに行レベルロックのみを使用。削除されたバージョン: MySQL 8.0.0.
- [innodb_scan_directories](#): InnoDB リカバリ中にテーブルスペースファイルをスキャンするディレクトリを定義。削除されたバージョン: MySQL 8.0.4.
- [innodb_stats_sample_pages](#): インデックス分布統計のサンプルを収集するインデックスページ数。削除されたバージョン: MySQL 8.0.0.
- [innodb_support_xa](#): XA 2 フェーズコミットの InnoDB サポートの有効化。削除されたバージョン: MySQL 8.0.0.
- [innodb_undo_logs](#): InnoDB で使用される undo ログ (ロールバックセグメント) の数 (`innodb_rollback_segments` のエイリアス)。削除されたバージョン: MySQL 8.0.2.
- [internal_tmp_disk_storage_engine](#): 内部一時テーブル用のストレージエンジン。削除されたバージョン: MySQL 8.0.16.
- [log-warnings](#): クリティカルでない警告をログファイルに書き込みます。削除されたバージョン: MySQL 8.0.3.
- [log_builtin_as_identified_by_password](#): CREATE/ALTER USER、GRANT を下位互換性のある方法で記録するかどうか。削除されたバージョン: MySQL 8.0.11.
- [log_error_filter_rules](#): エラーロギングのフィルタールール。削除されたバージョン: MySQL 8.0.4.
- [log_syslog](#): syslog にエラーメッセージを書き込むかどうか。削除されたバージョン: MySQL 8.0.13.
- [log_syslog_facility](#): syslog メッセージの機能。削除されたバージョン: MySQL 8.0.13.
- [log_syslog_include_pid](#): サーバー PID を syslog メッセージに含めるかどうか。削除されたバージョン: MySQL 8.0.13.
- [log_syslog_tag](#): syslog メッセージ内のサーバー識別子のタグ。削除されたバージョン: MySQL 8.0.13.
- [max_tmp_tables](#): 未使用。削除されたバージョン: MySQL 8.0.3.
- [metadata_locks_cache_size](#): メタデータロックキャッシュのサイズ。削除されたバージョン: MySQL 8.0.13.
- [metadata_locks_hash_instances](#): メタデータロックハッシュ数。削除されたバージョン: MySQL 8.0.13.
- [multi_range_count](#): 範囲選択中に一度にテーブルハンドラに送信する範囲の最大数。削除されたバージョン: MySQL 8.0.3.
- [old_passwords](#): PASSWORD() のパスワードハッシュ方式を選択。削除されたバージョン: MySQL 8.0.11.
- [partition](#): パーティション化サポートを有効化 (または無効化)。削除されたバージョン: MySQL 8.0.0.
- [query_cache_limit](#): これより大きい結果をキャッシュしない。削除されたバージョン: MySQL 8.0.3.
- [query_cache_min_res_unit](#): 結果の領域が割り当てられる単位の最小サイズ (すべての結果データの書き込み後に最後の単位が切り捨てられます)。削除されたバージョン: MySQL 8.0.3.

- [query_cache_size](#): 古いクエリーの結果を格納するために割り当てられたメモリー。削除されたバージョン: MySQL 8.0.3.
- [query_cache_type](#): クエリーキャッシュタイプ。削除されたバージョン: MySQL 8.0.3.
- [query_cache_wlock_invalidate](#): 書き込みの LOCK 時にクエリーキャッシュ内のクエリーを無効化。削除されたバージョン: MySQL 8.0.3.
- [secure_auth](#): 古い (4.1 より前の) パスワードを持つアカウントの認証を許可しない。削除されたバージョン: MySQL 8.0.3.
- [show_compatibility_56](#): SHOW STATUS/VARIABLES の互換性。削除されたバージョン: MySQL 8.0.1.
- [skip-partition](#): ユーザー定義のパーティション化を有効にしない。削除されたバージョン: MySQL 8.0.0.
- [sync_frm](#): .frm を作成時にディスクに同期します。デフォルトで有効。削除されたバージョン: MySQL 8.0.0.
- [temp-pool](#): このオプションを使用すると、作成されるほとんどの一時ファイルで、新しいファイルごとに一意の名前ではなく、小さな名前のセットが使用されます。削除されたバージョン: MySQL 8.0.1.
- [time_format](#): TIME 形式 (未使用)。削除されたバージョン: MySQL 8.0.3.
- [tx_isolation](#): デフォルトのトランザクション分離レベル。削除されたバージョン: MySQL 8.0.3.
- [tx_read_only](#): デフォルトのトランザクションアクセスモード。削除されたバージョン: MySQL 8.0.3.

1.5 MySQL の情報源

このセクションでは、MySQL web サイト、メーリングリスト、ユーザーフォーラム、インターネットリレーチャットなど、役立つ追加情報のソースを示します。

- [MySQL Web サイト](#)
- [MySQL フォーラムにおける MySQL コミュニティーサポート](#)
- [MySQL Enterprise](#)

MySQL Web サイト

MySQL ドキュメントの主要な web サイトは、<https://dev.mysql.com/doc/> です。オンラインおよびダウンロード可能なドキュメント形式は、MySQL リファレンスマニュアル、MySQL コネクタなどで使用できます。

MySQL 開発者は、「[MySQL Server ブログ](#)」としての新機能および今後の機能に関する情報を提供します。

MySQL フォーラムにおける MySQL コミュニティーサポート

<http://forums.mysql.com> のフォーラムは重要なコミュニティリソースです。多くのフォーラムに参加可能です。次のカテゴリがあります。

- Migration (移行)
- MySQL Usage (MySQL の使い方)
- MySQL Connector
- Programming Languages (プログラム言語)
- Tools (ツール)
- 3rd-Party Applications (サードパーティーによるアプリケーション)
- Storage Engines (ストレージエンジン)
- MySQL Technology (MySQL テクノロジー)

- SQL Standards (SQL の標準化)
- Business (ビジネス)

MySQL Enterprise

Oracle は、MySQL Enterprise のフォーラムで技術サポートを提供しています。ビジネスに不可欠な本番アプリケーションに MySQL DBMS を使用している組織の場合は、MySQL Enterprise は次を含む商用サブスクリプション製品です。

- MySQL Enterprise Server
- MySQL Enterprise Monitor
- Monthly Rapid Update および Quarterly Service Pack
- MySQL Knowledge Base
- 24 時間 365 日の技術およびコンサルタントサポート

MySQL Enterprise は複数の層で利用可能で、ニーズにもっとも適したサービスのレベルを柔軟に選択できます。詳細については、[MySQL Enterprise](#) を参照してください。

1.6 質問またはバグをレポートする方法

問題についてバグレポートを投稿する前に、それが実際にバグであることと、バグがまだレポートされていないことを確認してください。

- まず、<https://dev.mysql.com/doc/> で MySQL オンラインマニュアルを検索してください。マニュアルは、常に最新の状態にしておくために、新しく見つかった問題に対する解決策によって頻繁に更新されています。また、問題に関する解決方法が新しいバージョンにすでに組み込まれている可能性が高いため、マニュアルに付随するリリースノートは特に有用です。リリースノートはマニュアル用の場所から入手可能です。
- SQL ステートメントに対するパースエラーが生じた場合は、構文を念入りにチェックしてください。そこで問題が見当たらなければ、現在使用している MySQL Server のバージョンが、使用されている構文をサポートしていない可能性が高くなります。最新のバージョンを使用しており、マニュアルが使用された構文をカバーしていない場合、MySQL Server はそのステートメントをサポートしません。

マニュアルがその構文をカバーしているが、MySQL Server が旧バージョンの場合、MySQL の変更履歴を調べ、いつその構文が実装されたのかを確認してください。この場合、新しいバージョンの MySQL Server へアップグレードするという選択肢もあります。

- 一般的な問題の解決法については次を参照してください。[セクション B.3 「問題および一般的なエラー」](#)。
- <http://bugs.mysql.com/> でバグデータベースを検索して、バグがすでにレポートおよび解決されているかどうかを確認します。
- <http://www.mysql.com/search/> を使用して、MySQL web サイトにあるすべての Web ページ (マニュアルを含む) を検索することもできます。

マニュアル、バグデータベース、およびメーリングリストのアーカイブで回答を見つけることができなかった場合、お近くの MySQL の専門家にお問い合わせください。それでも質問に対する回答を見つけることができなかった場合は、次のガイドラインに従ってバグをレポートしてください。

通常バグをレポートする場合は、<http://bugs.mysql.com/> にアクセスしてください。これはバグデータベースのアドレスです。このバグデータベースは一般に公開されているので、だれでも参照および検索することができます。システムにログインすると、新しいレポートを入力できます。

<http://bugs.mysql.com/> のバグデータベースに投稿され、所定のリリースで修正されたバグは、リリースノートに記載されます。

MySQL Server でセキュリティバグが見つかった場合は、[<secalert_us@oracle.com>](mailto:secalert_us@oracle.com) に電子メールメッセージを送信して、すぐにお知らせください。例外: サポートのお客様は、セキュリティのバグを含むすべての問題を <http://support.oracle.com/> の Oracle Support までレポートしてください。

他のユーザーとの問題について話し合うには、[MySQL Community Slack](#) を使用できます。

よいバグレポートを書くのは時間がかかるものですが、最初に正しく行うことで報告者と弊社の双方の時間を節約できます。そのバグの完全なテストケースを含むよいバグレポートを提供していただければ、次のリリースではその問題を修正できる可能性が非常に高くなります。このセクションでは、あまり役に立たないことをして時間を無駄にすることがないように、レポートの正しい書き方を紹介します。このセクションを注意深く読み、ここに記載されているすべての情報がレポートに含まれているか、確認してください。

できれば、MySQL Server の最新の製品版または開発版を使用して問題をテストしてから投稿してください。テストケースに対して `mysql test < script_file` を使用するか、バグレポートに含まれているシェルまたは Perl のスクリプトを実行するだけで、だれでもバグを再現できるはずですが、弊社で再現が可能なバグであれば、次の MySQL リリースで修正される可能性が高くなります。

問題に関する適切な説明がバグレポートに記載されていると、もっとも効果的です。そのため、問題につながったすべての操作の適切な例を挙げ、問題自体を詳細に記述してください。もっとも効果的なレポートは、バグや問題を再現する方法を示す詳細な例が記載されたものです。[セクション5.9「MySQL のデバッグ」](#)を参照してください。

情報が多すぎるレポートに対応することはできますが、少なすぎるレポートに対応することはできません。多くの場合、問題の原因がわかっているか、細部を重要でないと考え、事実を省略してしまいます。記載するかどうかわからないときは、記載することをお勧めします。最初のレポートに十分な情報を記載していなかったために、再度質問して回答を待たなければならなくなるよりも、レポートに数行を追加する方が、はるかに時間が節約される上に、煩わしくありません。

バグレポートでもっともよくある誤りは、(a) 使用している MySQL ディストリビューションのバージョン番号を記載していない、(b) MySQL Server がインストールされているプラットフォームの説明 (プラットフォームの種類およびバージョン番号を含む) が十分でないというものです。これは非常に重要な情報なので、ほとんどの場合、この情報が記載されていないバグレポートは役に立ちません。「なぜうまく動作しないのか?」という質問が頻繁に寄せられます。その場合、要求した機能がその MySQL バージョンに実装されていなかったり、レポートに記載されているバグが新しい MySQL バージョンですでに修正されていたりすることがあります。エラーがプラットフォーム依存である場合もよくあります。そのような場合、オペレーティングシステムやプラットフォームのバージョン番号を知らずに問題を修正することはほとんど不可能です。

また、MySQL をソースからコンパイルした場合は、コンパイラが問題に関連している場合は、コンパイラに関する情報も記載してください。ユーザーがコンパイラのバグを見つけて、それが MySQL 関連の問題であると考えることがよくあります。ほとんどのコンパイラは常に開発中なので、バージョンごとに改良されています。問題がコンパイラに関連するものであるかどうかを判断するには、使用しているコンパイラを知る必要があります。コンパイルに関するすべての問題はバグとみなし、適宜レポートしてください。

プログラムでエラーメッセージが生成された場合、そのメッセージをレポートに記載することが非常に重要です。アーカイブから情報を検索しようとする場合、レポートされたエラーメッセージがプログラムで生成されたものと正確に一致している方が効果的です。(大文字小文字の違いにも注意してください。) エラーメッセージ全体をレポートにコピー&ペーストしてください。記憶に頼ってエラーメッセージを思い出そうとしないでください。

Connector/ODBC (MyODBC) に関する問題がある場合、トレースファイルを生成し、レポートとともに送信してください。[How to Report Connector/ODBC Problems or Bugs](#) を参照してください。

レポートに、`mysql` コマンド行ツールを使用して実行したテストケースからの長いクエリー出力が含まれる場合、`--vertical` オプション (または `\G` ステートメントターミネータ) を使用すると、読みやすくなります。このセクションで後述される `EXPLAIN SELECT` の例では、`\G` の使用方法を示します。

レポートには次の情報を記載してください。

- 使用している MySQL ディストリビューションのバージョン番号 (MySQL 5.7.10 など)。実行しているバージョンを確認するには、`mysqladmin version` を実行します。`mysqladmin` プログラムは、MySQL インストールディレクトリの下に `bin` ディレクトリにあります。
- 問題が発生したマシンの製造元とモデル。
- オペレーティングシステムの名前とバージョン。Windows を使用している場合、名前とバージョン番号を取得するには、通常「マイコンピュータ」アイコンをダブルクリックし、「ヘルプ/バージョン情報」メニューをプルダウンします。ほとんどの Unix 系のオペレーティングシステムでは、この情報を取得するには、コマンド `uname -a` を実行します。

- メモリー (実メモリーと仮想メモリー) の量が関連することもあります。不確かな場合は、これらの値を記載します。
- MySQL インストールの `docs/INFO_BIN` ファイルの内容。このファイルには、MySQL の構成およびコンパイル方法に関する情報が含まれています。
- MySQL ソフトウェアのソースディストリビューションを使用している場合、使用したコンパイラの名前とバージョン番号を記載します。バイナリディストリビューションを使用している場合は、ディストリビューション名を記載します。
- コンパイル時に問題が発生した場合、正確なエラーメッセージ、およびエラーが発生したファイル内の問題のあるコード付近の数行のコンテキストを記載します。
- `mysqld` が停止した場合は、`mysqld` が予期せず終了する原因となったステートメントも報告する必要があります。通常、この情報を取得するには、クエリーロギングを有効にして `mysqld` を実行し、`mysqld` の終了後にログを参照します。セクション5.9「MySQL のデバッグ」を参照してください。
- データベーステーブルが問題に関連している場合、`SHOW CREATE TABLE db_name.tbl_name` ステートメントからの出力をバグレポートに記載します。これは、データベース内のテーブルの定義を取得する非常に簡単な方法です。この情報は、発生した状況と同じ状況を再現する際に役立ちます。
- 問題発生時の SQL モードも有効な情報になる場合があるので、`sql_mode` システム変数の値をレポートしてください。ストアドプロシージャ、ストアドファンクション、およびトリガーオブジェクトでは、関連する `sql_mode` の値は、そのオブジェクトが作成された際に有効だったものです。ストアドプロシージャまたはストアドファンクションでは、`SHOW CREATE PROCEDURE` または `SHOW CREATE FUNCTION` ステートメントは関連する SQL モードを示しています。また、`INFORMATION_SCHEMA` で情報を問い合わせできます。

```
SELECT ROUTINE_SCHEMA, ROUTINE_NAME, SQL_MODE
FROM INFORMATION_SCHEMA.ROUTINES;
```

トリガーに対しては次のステートメントが利用できます。

```
SELECT EVENT_OBJECT_SCHEMA, EVENT_OBJECT_TABLE, TRIGGER_NAME, SQL_MODE
FROM INFORMATION_SCHEMA.TRIGGERS;
```

- 性能に関連するバグや `SELECT` ステートメントに関する問題については、`EXPLAIN SELECT ...` の出力、および少なくとも `SELECT` ステートメントが生成する行の数も含めてください。関連する各テーブルについて、`SHOW CREATE TABLE tbl_name` からの出力も含めてください。状況について情報を提供できればできるほど、有効な手助けが可能となります。

下記は非常によいバグレポートの例です。 `mysql` コマンド行ツールを使ってステートメントが実行されています。読みにくい長い出力行に対して、`\G` ステートメントターミネータが使用されていることに注意してください。

```
mysql> SHOW VARIABLES;
mysql> SHOW COLUMNS FROM ...G
<output from SHOW COLUMNS>
mysql> EXPLAIN SELECT ...G
<output from EXPLAIN>
mysql> FLUSH STATUS;
mysql> SELECT ...;
<A short version of the output from SELECT,
including the time taken to run the query>
mysql> SHOW STATUS;
<output from SHOW STATUS>
```

- `mysqld` の実行中にバグまたは問題が発生した場合、その異常を再現する入力スクリプトを提供します。このスクリプトには、必要なソースファイルを含める必要があります。スクリプトによって再現される状況が実際に発生した状況に近いほど、効果的です。再現可能なテストケースを作成できる場合は、バグレポートに添付してアップロードしてください。

スクリプトを提供できない場合は、少なくとも `mysqladmin variables extended-status processlist` からの出力をレポートに記載して、システムの動作に関する情報を提供してください。

- 数行ではテストケースを再現できない場合や、テストテーブルが大きすぎてバグレポートに含めることができない場合 (10 行を超える場合)、`mysqldump` を使用してテーブルをダンプし、問題を説明する `README` ファイルを作成してください。 `tar` と `gzip` または `zip` を使用してファイルの圧縮アーカイブを作成します。 <http://bugs.mysql.com/>

のバグデータベースのバグレポートを開始してから、バグレポートの「ファイル」タブをクリックして、アーカイブをバグデータベースにアップロードする指示に従ってください。

- MySQL Server でステートメントから正しくない結果が生成されたと思う場合、結果だけでなく、正しいと考える結果と、その考えの根拠を示す説明も記載します。
- 問題のサンプルを提供する際には、テーブル名や変数名などは、新しい名前を使用するよりも実際の状況で存在するものを使用するのが適切です。問題が、テーブルや変数の名前に関連している可能性があります。このようなケースはほとんどないと思われませんが、後悔するよりも安全策をとるべきです。結局、実際の状況に基づいたサンプルを提供する方がユーザーにとって簡単であると同時に、当社にとっても効率的です。バグレポート内に、ほかのユーザーに公開したくないデータがある場合は、前述のように「ファイル」タブを使用してアップロードできます。データが実際に最高機密で、当社にも公開したくない場合は、ほかの名前を使用したサンプルを提供することもできますが、これは最後の選択肢です。
- 可能な場合、関連するプログラムのすべてのオプションを記載します。たとえば、`mysqld` サーバーを開始する際に使用したオプションや MySQL クライアントプログラムの実行に使用したオプションを記載します。`mysqld` や `mysql` のようなプログラムや、`configure` スクリプトのオプションは多くの場合、問題解決の鍵となり、非常に重要です。これらを記載することは、決して無駄ではありません。問題が、Perl や PHP などの言語で記述されたプログラムに関係する場合は、言語プロセッサのバージョン番号だけでなく、プログラムが使用するモジュールのバージョンも記載します。たとえば、`DBI` モジュールや `DBD::mysql` モジュールを使用する Perl スクリプトがある場合、Perl、`DBI`、および `DBD::mysql` のバージョン番号を記載します。
- 質問が権限システムに関連する場合は、`mysqladmin reload` の出力と、接続しようとしたときに表示されるすべてのエラーメッセージを含めてください。権限をテストする場合は、`mysqladmin reload version` を実行し、問題のあるプログラムに接続してみてください。
- バグのパッチがある場合、それを含めます。ただし、当社はパッチだけを必要としているわけではありません。パッチによって修正されるバグを示すテストケースなどの必要な情報が記載されていない場合、当社はそのパッチを使用できません。当社がパッチに関連する問題を見つける場合もあれば、パッチをまったく理解できない場合もあります。そのような場合は、パッチを使用できません。

パッチの目的を正確に確認することができない場合、当社はそのパッチを使用しません。この場合、テストケースが役立つこととなります。発生する可能性があるすべての状況がパッチによって処理されることを示してください。パッチが機能しないボーダーラインケースが見つかった場合 (それがまれなケースでも)、そのパッチは役に立たない可能性があります。

- 何がバグであるか、そのバグがなぜ発生するのか、そのバグが何に関連するのかについての推測は、間違っていることが多いです。MySQL チームでさえも、最初にデバッグを使用しなければ、そのようなことを推測してバグの実際の原因を特定することはできません。
- ユーザー自身で問題を解決しようとしたことがほかのユーザーにわかるように、リファレンスマニュアルやメールアーカイブを確認したことをバグレポートに記載します。
- データが壊れていると思われる場合や、特定のテーブルにアクセスした際にエラーが発生した場合は、まず `CHECK TABLE` でテーブルをチェックしてください。そのステートメントでエラーがレポートされた場合、
 - `InnoDB` クラッシュリカバリメカニズムは、サーバーが強制終了したあとに再起動した場合にクリーンアップを処理します。そのため、通常の操作ではテーブルを「修復」する必要はありません。`InnoDB` テーブルでエラーが生じる場合は、サーバーを再起動して問題が継続するかどうか、またはエラーがメモリー内のキャッシュデータのみに影響するのかを確認します。ディスク上のデータが壊れている場合、影響を受けたテーブルをダンプできるように、`innodb_force_recovery` オプションを有効にして再起動してみてください。
 - トランザクショナルでないテーブルについては、`REPAIR TABLE` または `myisamchk` を使用して修復を試みてください。第5章「MySQL サーバーの管理」を参照してください。

Windows を実行している場合は、`lower_case_table_names` の値を `SHOW VARIABLES LIKE 'lower_case_table_names'` ステートメントを使用して確認します。この変数は、データベースおよびテーブル名の大文字/小文字をサーバーがどのように扱うかに対して影響を与えます。ある値に対しての効果は [セクション 9.2.3「識別子の小文字と大文字の区別」](#) で説明されているとおりです。

- テーブルが頻繁に壊れる場合、その問題が発生する状況と理由を特定する必要があります。この場合、MySQL データディレクトリのエラーログに、発生した問題に関する情報が格納されていることがあります。(そのファイルは、名前に `.err` のサフィクスが付いたファイルです。) [セクション 5.4.2「エラーログ」](#) を参照してください。こ

のファイル内の関連する情報をバグレポートに記載します。通常、更新中に何も強制終了しなかった場合、`mysqld` はテーブルを破損させないでください。`mysqld` が停止した原因が特定されれば、当社はその問題の解決策をはるかに簡単に提供できます。[セクションB.3.1「問題の原因を判別する方法」](#)を参照してください。

- 可能な場合、最新バージョンの MySQL Server をダウンロードしてインストールし、これによって問題が解決されるかどうかを確認します。MySQL ソフトウェアのすべてのバージョンが詳細にテストされているため、問題なく動作するはずですが、すべてにおいて最大限の下位互換性が確保されているため、MySQL のバージョンを問題なく切り替えることができると当社は確信しています。[セクション2.1.2「インストールする MySQL のバージョンと配布の選択」](#)を参照してください。

1.7 MySQL の標準への準拠

このセクションでは、MySQL と ANSI/ISO SQL 標準との関連について説明します。MySQL Server には SQL 標準に対する多数の拡張機能があり、ここでは、それらの拡張機能と使用方法について説明します。また、MySQL Server に不足している機能と、この不足分に対処する方法に関する情報も示します。

SQL 標準は 1986 年以降開発が繰り返され、複数のバージョンがあります。このマニュアルでは、「SQL-92」は 1992 年にリリースされた標準を指します。「SQL:1999」、「SQL:2003」、「SQL:2008」および「SQL:2011」は、対応する年にリリースされた標準のバージョンを参照します。最新バージョンは最新バージョンです。「SQL 標準」や「標準 SQL」という語句は、常に SQL 標準の現バージョンを意味します。

製品に関する当社の主な目標の 1 つは、速度や信頼性を犠牲にすることなく、SQL 標準への準拠に向けた取り組みを続けることです。大部分のユーザーにとって MySQL Server が大幅に使いやすくなるのであれば、当社は SQL に対する拡張機能や SQL 以外の機能のサポートを積極的に追加します。[HANDLER](#) インタフェースがこの方針の一例です。[セクション13.2.4「HANDLER ステートメント」](#)を参照してください。

24 時間 365 日のミッションクリティカルな使用にも、負荷のかかる Web またはロギングの使用にも対応できるように、トランザクションデータベースと非トランザクションデータベースのサポートを継続しています。

MySQL Server は当初、小規模なコンピュータシステムで中規模のデータベース (1,000 万から 1 億行、または 1 テーブルあたり約 100M バイト) を操作できるように設計されました。現在、MySQL Server はテラバイトサイズのデータベースを処理しています。

MySQL レプリケーション機能は重要な機能を提供しますが、リアルタイムサポートは対象としていません。

MySQL は ODBC レベル 0 から 3.51 をサポートします。

MySQL は、[NDBCLUSTER](#) ストレージエンジンを使用した高可用性データベースクラスタリングをサポートします。[第23章「MySQL NDB Cluster 8.0」](#)を参照してください。

ほとんどの W3C XPath 標準をサポートする XML 機能を実装しています。[セクション12.12「XML 関数」](#)を参照してください。

MySQL では、RFC 7159 で定義され、ECMAScript 標準 (ECMA-262) に基づくネイティブ JSON データ型がサポートされています。[セクション11.5「JSON データ型」](#)を参照してください。MySQL では、SQL:2016 標準の公開前のドラフトで指定された SQL/JSON 関数のサブセットも実装されます。詳細は、[セクション12.18「JSON 関数」](#)を参照してください。

SQL モードの選択

MySQL Server は異なる SQL モードで動作でき、`sql_mode` システム変数の値に応じて異なるクライアントにこれらの異なるモードを適用できます。DBA はサイトサーバーの動作要件に一致するグローバル SQL モードを設定でき、各アプリケーションはアプリケーションのセッション SQL モードをアプリケーション独自の要件に設定できます。

モードは MySQL がサポートする SQL 構文と、MySQL が実行するデータ検証に影響します。これにより、MySQL をさまざまな環境で使用したり、MySQL をほかのデータベースサーバーと一緒に使用したりすることが、さらに容易になります。

SQL モードの設定の詳細は、[セクション5.1.11「サーバー SQL モード」](#)を参照してください。

ANSI モードでの MySQL の実行

ANSI モードで MySQL Server を実行するには、`--ansi` オプションを付けて `mysqld` を起動します。ANSI モードでのサーバーの実行は、次のオプションを指定してサーバーを起動する場合と同じです。

```
--transaction-isolation=SERIALIZABLE --sql-mode=ANSI
```

実行時に同じ効果を得るには、次の 2 つのステートメントを実行します。

```
SET GLOBAL TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
SET GLOBAL sql_mode = 'ANSI';
```

次のように、`sql_mode` システム変数値を 'ANSI' に設定すると、ANSI モードに関連するすべての SQL モードオプションが有効になります。

```
mysql> SET GLOBAL sql_mode='ANSI';  
mysql> SELECT @@GLOBAL.sql_mode;  
-> 'REAL_AS_FLOAT,PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,ANSI'
```

`--ansi` を使用して ANSI モードでサーバーを実行した場合は、`--ansi` オプションがトランザクション分離レベルも設定するため、SQL モードを 'ANSI' に設定した場合とは少し異なります。

[セクション5.1.7「サーバーコマンドオプション」](#) を参照してください。

1.7.1 標準 SQL に対する MySQL 拡張機能

MySQL Server では、他の SQL DBMS で検出されない可能性のある拡張機能がサポートされています。これらを使用する場合は、コードが他の SQL サーバーに移植できない可能性があることを警告してください。場合によっては、MySQL 拡張機能を含むコードを記述しても、次の形式のコメントを使用することで移植することができます。

```
/*! MySQL-specific code */
```

この場合、MySQL Server は他の SQL ステートメントと同様にコメント内のコードを解析して実行しますが、他の SQL サーバーは拡張を無視する必要があります。たとえば、MySQL Server は次のステートメントの `STRAIGHT_JOIN` キーワードを認識しますが、他のサーバーは認識できません:

```
SELECT /*! STRAIGHT_JOIN */ col1 FROM table1,table2 WHERE ...
```

!文字のあとにバージョン番号を追加すると、コメント内の構文は、MySQL のバージョンが指定されたバージョン番号以上の場合にだけ実行されます。次のコメントの `KEY_BLOCK_SIZE` 句は、MySQL 5.1.10 以上のサーバーによってのみ実行されます:

```
CREATE TABLE t1(a INT, KEY (a)) /*!50110 KEY_BLOCK_SIZE=1024 */;
```

次の説明は、MySQL 拡張機能をカテゴリでまとめた一覧です。

- ディスク上のデータの構成

MySQL Server は、各データベースを MySQL データディレクトリ下のディレクトリにマップし、データベース内のテーブルをデータベースディレクトリ内のファイル名にマップします。したがって、大/小文字が区別されるファイル名 (ほとんどの Unix システムなど) を持つオペレーティングシステム上の MySQL Server では、データベース名とテーブル名は大/小文字が区別されます。 [セクション9.2.3「識別子の大きい文字と小さい文字の区別」](#) を参照してください。

- 一般言語構文

- デフォルトでは、文字列は"および'で囲むことができます。 `ANSI_QUOTES` SQL モードが有効な場合、文字列は'でのみ囲むことができ、サーバーは"で囲まれた文字列を識別子として解釈します。
- \は文字列内のエスケープ文字です。
- SQL ステートメントでは、`db_name.tbl_name` 構文を使用して、さまざまなデータベースのテーブルにアクセスできます。同様の機能を備えた SQL サーバーもありますが、これは `User space` と呼ばれます。MySQL Server は、`CREATE TABLE ralph.my_table ... IN my_tablespace` など、ステートメントで使用されるようなテーブルスペースをサポートしません。

- SQL ステートメント構文
 - [ANALYZE TABLE](#)、[CHECK TABLE](#)、[OPTIMIZE TABLE](#)、および [REPAIR TABLE](#) ステートメント。
 - [CREATE DATABASE](#)、[DROP DATABASE](#)、および [ALTER DATABASE](#) ステートメント。 [セクション 13.1.12 「CREATE DATABASE ステートメント」](#)、[セクション 13.1.24 「DROP DATABASE ステートメント」](#)、および [セクション 13.1.2 「ALTER DATABASE ステートメント」](#) を参照してください。
 - [DO](#) ステートメント。
 - クエリー最適マイザによるテーブルの処理方法に関する説明を取得する [EXPLAIN SELECT](#)。
 - [FLUSH](#) および [RESET](#) ステートメント。
 - [SET](#) ステートメント。 [セクション 13.7.6.1 「変数代入の SET 構文」](#) を参照してください。
 - [SHOW](#) ステートメント。 [セクション 13.7.7 「SHOW ステートメント」](#) を参照してください。 MySQL 固有の [SHOW](#) ステートメントの多くによって生成される情報は、[SELECT](#) を使用して [INFORMATION_SCHEMA](#) をクエリーすることによって、より標準的な方法で取得できます。 [第 26 章 「INFORMATION_SCHEMA テーブル」](#) を参照してください。
 - [LOAD DATA](#) の使用。多くの場合、この構文は Oracle [LOAD DATA](#) と互換性があります。 [セクション 13.2.7 「LOAD DATA ステートメント」](#) を参照してください。
 - [RENAME TABLE](#) の使用。 [セクション 13.1.36 「RENAME TABLE ステートメント」](#) を参照してください。
 - [DELETE](#) + [INSERT](#) の代替としての [REPLACE](#) の使用。 [セクション 13.2.9 「REPLACE ステートメント」](#) を参照してください。
 - [ALTER TABLE](#) ステートメントにおける [CHANGE col_name](#)、[DROP col_name](#)、あるいは [DROP INDEX](#)、[IGNORE](#) または [RENAME](#) の使用。 [ALTER TABLE](#) ステートメントにおける、複数の [ADD](#)、[ALTER](#)、[DROP](#)、または [CHANGE](#) 句の使用。 [セクション 13.1.9 「ALTER TABLE ステートメント」](#) を参照してください。
 - インデックス名の使用、カラムのプリフィクス上のインデックス、および [CREATE TABLE](#) ステートメントでの [INDEX](#) または [KEY](#) の使用。 [セクション 13.1.20 「CREATE TABLE ステートメント」](#) を参照してください。
 - [CREATE TABLE](#) を用いた [TEMPORARY](#) または [IF NOT EXISTS](#) の使用。
 - [DROP TABLE](#) および [DROP DATABASE](#) を用いた [IF EXISTS](#) の使用。
 - 1 つの [DROP TABLE](#) ステートメントで複数のテーブルを破棄できる機能。
 - [UPDATE](#) および [DELETE](#) ステートメントの [ORDER BY](#) および [LIMIT](#) 句。
 - [INSERT INTO tbl_name SET col_name = ...](#) 構文。
 - [INSERT](#) および [REPLACE](#) ステートメントの [DELAYED](#) 句。
 - [INSERT](#)、[REPLACE](#)、[DELETE](#)、および [UPDATE](#) ステートメントの [LOW_PRIORITY](#) 句。
 - [SELECT](#) ステートメントにおける [INTO OUTFILE](#) または [INTO DUMPFILE](#) の使用。 [セクション 13.2.10 「SELECT ステートメント」](#) を参照してください。
 - [SELECT](#) ステートメントにおける [STRAIGHT_JOIN](#) や [SQL_SMALL_RESULT](#) などのオプション。
 - [GROUP BY](#) 句で、選択したすべてのカラムの名前を列挙する必要はありません。これにより、ごく一部ではありますが、きわめて一般的なクエリーのパフォーマンスが向上します。 [セクション 12.20 「集計関数」](#) を参照してください。
 - [ORDER BY](#) を用いるだけでなく [GROUP BY](#) を用いても、[ASC](#) および [DESC](#) を指定できます。
 - [:=](#) 割り当て演算子で、ステートメント内の変数を設定する機能。 [セクション 9.4 「ユーザー定義変数」](#) を参照してください。

- データ型
 - [MEDIUMINT](#)、[SET](#)、および [ENUM](#) データ型と、さまざまな [BLOB](#) および [TEXT](#) データ型。
 - [AUTO_INCREMENT](#)、[BINARY](#)、[NULL](#)、[UNSIGNED](#)、および [ZEROFILL](#) データ型の属性。
- 関数と演算子
 - ほかの SQL 環境を使用していたユーザーにわかりやすいように、MySQL Server では多数の関数のエイリアスがサポートされています。たとえば、すべての文字列関数で、標準の SQL 構文と ODBC 構文の両方がサポートされています。
 - MySQL Server は、`||` および `&&` 演算子が、C プログラミング言語の場合と同様に論理 OR および AND を意味することを理解しています。MySQL Server では、`||` と `OR` はシノニムであり、`&&` と `AND` も同様です。この優れた構文のために、MySQL Server では、文字列の連結に標準 SQL の `||` 演算子を使用することができません。その代わりに、[CONCAT\(\)](#) を使用します。[CONCAT\(\)](#) は任意の数の引数を取るため、`||` 演算子の使用を MySQL Server に変換することは簡単です。
 - `value_list` に複数の要素がある場合の、[COUNT\(DISTINCT value_list\)](#) の使用。
 - デフォルトでは、文字列比較では大/小文字は区別されず、ソート順序は現在の文字セット (デフォルトでは [utf8mb4](#)) の照合順序によって決定されます。かわりに、大/小文字を区別する比較を実行するには、[BINARY](#) 属性を使用してカラムを宣言するか、[BINARY](#) キャストを使用する必要があります。これにより、字句の順序付けではなく、基礎となる文字コード値を使用して比較が実行されます。
 - `%` 演算子は [MOD\(\)](#) のシノニムです。つまり、`N % M` は [MOD\(N,M\)](#) と同等です。`%` は、C プログラムと、PostgreSQL との互換性のためにサポートされています。
 - [SELECT](#) ステートメントの出力カラムリスト ([FROM](#) の左側) の式で、`=`、`<>`、`<=`、`<`、`>=`、`>`、`<<`、`>>`、`<=>`、[AND](#)、[OR](#)、または [LIKE](#) 演算子を使用できます。例:

```
mysql> SELECT col1=1 AND col2=2 FROM my_table;
```
 - [LAST_INSERT_ID\(\)](#) 関数は、最新の [AUTO_INCREMENT](#) 値を返します。[セクション12.16「情報関数」](#)を参照してください。
 - [LIKE](#) は、数値に対して使用できます。
 - [REGEXP](#) および [NOT REGEXP](#) 拡張正規表現演算子。
 - 1 つまたは複数の引数を使用する [CONCAT\(\)](#) または [CHAR\(\)](#)。(MySQL Server では、これらの関数は引数をいくつでも使用することができます。)
 - [BIT_COUNT\(\)](#)、[CASE](#)、[ELT\(\)](#)、[FROM_DAYS\(\)](#)、[FORMAT\(\)](#)、[IF\(\)](#)、[MD5\(\)](#)、[PERIOD_ADD\(\)](#)、[PERIOD_DIFF\(\)](#)、[TO_DAYS\(\)](#) および [WEEKDAY\(\)](#) の機能。
 - 部分文字列を削除する [TRIM\(\)](#) の使用。標準 SQL では、1 つの文字しか削除できません。
 - [GROUP BY](#) 関数 [STD\(\)](#)、[BIT_OR\(\)](#)、[BIT_AND\(\)](#)、[BIT_XOR\(\)](#)、および [GROUP_CONCAT\(\)](#)。[セクション12.20「集計関数」](#)を参照してください。

1.7.2 MySQL と標準 SQL との違い

MySQL Server を ANSI SQL 標準および ODBC SQL 標準に準拠したものにするのを目標としていますが、次のように、MySQL Server が異なる動作を示すことがあります。

- MySQL と標準 SQL の権限システムには、いくつかの違いがあります。たとえば MySQL では、テーブルを削除しても、テーブルの権限が自動的に取り消されません。テーブルの権限を取り消すには、明示的に [REVOKE](#) ステートメントを発行する必要があります。詳細は、[セクション13.7.1.8「REVOKE ステートメント」](#)を参照してください。
- [CAST\(\)](#) 関数は、[REAL](#) または [BIGINT](#) に対するキャストをサポートしていません。[セクション12.11「キャスト関数と演算子」](#)を参照してください。

1.7.2.1 SELECT INTO TABLE の違い

MySQL Server では、Sybase SQL 拡張機能 `SELECT ... INTO TABLE` はまだサポートされていません。MySQL Server では代わりに、基本的に同じ処理を行う `INSERT INTO ... SELECT` 標準 SQL 構文がサポートされています。セクション13.2.6.1「`INSERT ... SELECT` ステートメント」を参照してください。例:

```
INSERT INTO tbl_temp2 (fld_id)
  SELECT tbl_temp1.fld_order_id
  FROM tbl_temp1 WHERE tbl_temp1.fld_order_id > 100;
```

あるいは、`SELECT ... INTO OUTFILE` または `CREATE TABLE ... SELECT` を使用することもできます。

ユーザー定義変数で `SELECT ... INTO` を使用できます。カーソルとローカル変数を用いてストアルーチン内でも同じ構文を使用できます。セクション13.2.10.1「`SELECT ... INTO` ステートメント」を参照してください。

1.7.2.2 UPDATE の違い

式で更新されるテーブルのカラムにアクセスする場合、`UPDATE` はそのカラムの現在の値を使用します。次のステートメントの2番目の割り当ては、`col2` を元の `col1` 値ではなく、現在の (更新された) `col1` 値に設定します。この結果、`col1` と `col2` の値が同じになります。この動作は標準 SQL とは異なります。

```
UPDATE t1 SET col1 = col1 + 1, col2 = col1;
```

1.7.2.3 FOREIGN KEY 制約の違い

外部キー制約の MySQL 実装は、次の点で SQL 標準と異なります:

- 親テーブルに同じ参照キー値を持つ複数の行がある場合、`InnoDB` は、同じキー値を持つ他の親行が存在しないかのように外部キーチェックを実行します。たとえば、`RESTRICT` 型制約を定義し、複数の親行を持つ子行がある場合、`InnoDB` では親行の削除は許可されません。
- `ON UPDATE CASCADE` または `ON UPDATE SET NULL` が同じカスケード中に以前に更新していた同じテーブルを更新するように再帰する場合、`RESTRICT` のように機能します。つまり、自己参照型 `ON UPDATE CASCADE` または `ON UPDATE SET NULL` 操作は使用できません。この目的は、カスケード更新で発生する無限ループを回避することです。反対に、自己参照型 `ON DELETE SET NULL` は、自己参照型 `ON DELETE CASCADE` と同様に動作できます。カスケード操作は、15 レベルよりも深くネストされる可能性がありません。
- 多くの行を挿入、削除、または更新する SQL ステートメントでは、外部キー制約 (一意の制約など) が行ごとにチェックされます。外部キーチェックを行なっているとき、`InnoDB` は、調べる必要がある子レコードまたは親レコード上に共有行レベルロックを設定します。MySQL は外部キー制約の確認を即座に行います。その確認がトランザクションコミットまで遅延されることはありません。SQL 標準によると、デフォルトの動作は遅延チェックにするべきです。つまり、SQL ステートメント全体が処理されたあとにはじめて、制約がチェックされます。これは、外部キーを使用して自己参照する行を削除できないことを意味します。
- `InnoDB` を含むストレージエンジンは、参照整合性制約定義で使用される `MATCH` 句を認識または適用しません。明示的な `MATCH` 句を使用しても効果はなく、`ON DELETE` 句および `ON UPDATE` 句は無視されます。`MATCH` の指定は避けてください。

SQL 標準の `MATCH` 句は、参照テーブルの主キーと比較する際にコンポジット (複数カラム) 外部キーの `NULL` 値がどのように処理されるかを制御します。MySQL は、基本的に `MATCH SIMPLE` で定義されたセマンティクスを実装します。これにより、外部キーをすべてまたは一部 `NULL` にすることができます。その場合、このような外部キーを含む (子テーブル) 行は、参照される (親) テーブルのどの行とも一致しなくても挿入できます。(トリガーを使用して他のセマンティクスを実装できます。)

- MySQL では、パフォーマンス上の理由から、参照カラムをインデックス付けする必要があります。ただし、MySQL では、参照されるカラムが `UNIQUE` であるか、`NOT NULL` として宣言されている必要はありません。

`UNIQUE` でないキーを参照する `FOREIGN KEY` 制約は、標準 SQL ではなく `InnoDB` の拡張機能です。一方、`NDB` ストレージエンジンでは、外部キーとして参照される任意のカラムに明示的な一意キー (または主キー) が必要です。

一意でないキーまたは `NULL` 値を含むキーへの外部キー参照の処理は、`UPDATE` や `DELETE CASCADE` などの操作に対して適切に定義されていません。`UNIQUE` (`PRIMARY` を含む) および `NOT NULL` キーのみを参照する外部キーを使用することをお勧めします。

- MySQL は、参照がカラム指定の一部として定義されている「インライン REFERENCES 仕様」(SQL 標準で定義されているもの)を解析しますが、無視します。MySQL は、個別の FOREIGN KEY 指定の一部として指定されている場合のみ REFERENCES 句を受け入れます。外部キーをサポートしていない(MyISAM などの)ストレージエンジンの場合、MySQL Server は、外部キーの指定を解析して無視します。

外部キー制約の詳細は、[セクション13.1.20.5「FOREIGN KEY の制約」](#)を参照してください。

1.7.2.4 コメントの先頭としての「--」

標準 SQL では、コメントに C 構文 `/* this is a comment */` が使用され、MySQL Server でも同様にこの構文がサポートされています。[セクション9.7「コメント」](#)で説明するように、MySQL は、MySQL 固有の SQL をコメントに埋め込めるようにする、この構文の拡張機能もサポートします。

標準 SQL では、「--」をコメント開始連続文字として使用します。MySQL Server では、# をコメント開始文字として使用します。MySQL Server は、-- コメントスタイルのバリエーションもサポートしています。つまり、-- コメント開始連続文字には空白(または改行などの制御文字)を続ける必要があります。この空白は、`payment` の値を自動的に挿入する次のような構造構文を使用した自動生成の SQL クエリーでの問題を防止するために必要です。

```
UPDATE account SET credit=credit-payment
```

`payment` の値が負数 (-1 など) の場合、どうなるかを考えてみてください。

```
UPDATE account SET credit=credit--1
```

`credit--1` は SQL では有効な式ですが、-- はコメントの先頭として解釈され、式の一部は破棄されます。結果として、意図したものとはまったく異なる意味を持つステートメントとなってしまいます。

```
UPDATE account SET credit=credit
```

このステートメントでは、値が変更されることは一切ありません。これは、コメントを -- で開始できるようにすると、深刻な結果を招く可能性があること示します。

実装を使用するには、MySQL Server で開始コメントシーケンスとして認識されるように、-- の後にスペースが必要です。したがって、`credit--1` は安全に使用できます。

別の安全な機能は、`mysql` コマンド行クライアントで -- で始まる行を無視するというものです。

1.7.3 MySQL における制約の処理

MySQL では、ロールバックを許可するトランザクションテーブルと、許可しない非トランザクションテーブルの両方を操作できます。このため、MySQL とほかの DBMS とでは制約の処理が多少異なります。エラーが起きたときに変更をロールバックできない非トランザクションテーブルに、多数の行を挿入または更新した場合を扱う必要があります。

基本的な考え方は、MySQL Server が、実行するステートメントを構文解析している間に検出できるすべてのものに対してエラーを生成しようとし、ステートメントを実行する間に発生するエラーからリカバリしようとするというものです。ほとんどの場合でこれを行います。すべての場合にはまだです。

エラーが発生したときに MySQL で可能な選択肢は、途中でステートメントを中止するか、問題からリカバリするためにできるかぎりのことを行なって処理を続行するかです。デフォルトでは、サーバーは後者の方法に従います。これは、たとえば、サーバーは無効な値をもっとも近い有効な値に強制的に修正するということです。

不良データ値の処理や、エラー時にステートメント実行を継続するか中止するかをより適切に制御できるようにする複数の SQL モードオプションを利用できます。これらのオプションを使用して、不適切な入力を拒絶するほかの DBMS と同じように、より従来に近い方法で機能するように MySQL Server を構成できます。サーバー起動時に、SQL モードをグローバルに設定して、すべてのクライアントに影響を与えることができます。実行時に個々のクライアントで SQL モードを設定できるため、各クライアントは要件にもっとも適した動作を選択できます。[セクション5.1.11「サーバー SQL モード」](#)を参照してください。

次のセクションでは、さまざまな種類の制約を MySQL Server で処理する方法について説明します。

1.7.3.1 PRIMARY KEY および UNIQUE インデックス制約

通常、主キー制約、一意キー制約または外部キー制約に違反するデータ変更ステートメント (`INSERT` や `UPDATE` など) でエラーが発生します。InnoDB などのトランザクションストレージエンジンを使用している場合、MySQL ではステートメントが自動的にロールバックされます。非トランザクションストレージエンジンを使用している場合、MySQL は、エラーが発生した行でステートメントの処理を停止し、残りの行を未処理のままにしておきます。

MySQL では、`INSERT` や `UPDATE` などに対して `IGNORE` キーワードをサポートします。これを使用する場合は、MySQL は、プライマリキーまたは一意キーの違反を無視し、次の行を処理し続けます。使用しているステートメントに関するセクション (セクション13.2.6 「`INSERT` ステートメント」やセクション13.2.13 「`UPDATE` ステートメント」など) を参照してください。

`mysql_info()` C API 関数で、実際に挿入または更新される行数についての情報を取得できます。 `SHOW WARNINGS` ステートメントを使用することもできます。 `mysql_info()` およびセクション13.7.7.42 「`SHOW WARNINGS` ステートメント」を参照してください。

InnoDB テーブルおよび NDB テーブルでは、外部キーがサポートされます。セクション1.7.3.2 「`FOREIGN KEY` の制約」を参照してください。

1.7.3.2 FOREIGN KEY の制約

外部キーを使用すると、複数のテーブルにわたる関連データをクロス参照することができ、外部キー制約は、この分散したデータの整合性の維持に役立ちます。

MySQL は、`CREATE TABLE` および `ALTER TABLE` ステートメントにおける `ON UPDATE` および `ON DELETE` 外部キー参照をサポートします。使用可能な参照アクションは、`RESTRICT`、`CASCADE`、`SET NULL` および `NO ACTION` (デフォルト) です。

`SET DEFAULT` は MySQL Server でもサポートされていますが、現在、InnoDB では無効として拒否されています。MySQL は遅延した制約のチェックをサポートしないので、`NO ACTION` は `RESTRICT` として扱われます。外部キーについて MySQL によってサポートされる正確な構文については、セクション13.1.20.5 「`FOREIGN KEY` の制約」を参照してください。

`MATCH FULL`、`MATCH PARTIAL`、および `MATCH SIMPLE` は許可されますが、同じステートメントで使用されるすべて `ON DELETE` 句と `ON UPDATE` 句を MySQL Server に無視させるため、これらを使用しないでください。MySQL では `MATCH` オプションはほかの効果がなく、事実上常に `MATCH SIMPLE` セマンティクスが強制されます。

MySQL では、外部キーカラムにインデックスを付ける必要があります。外部キー制約はあるが所定のカラムのインデックスがないテーブルを作成する場合、インデックスが作成されます。

`INFORMATION_SCHEMA.KEY_COLUMN_USAGE` テーブルから、外部キーに関する情報を取得できます。このテーブルに対するクエリーの例を次に示します。

```
mysql> SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME, CONSTRAINT_NAME
> FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE
> WHERE REFERENCED_TABLE_SCHEMA IS NOT NULL;
+-----+-----+-----+-----+
| TABLE_SCHEMA | TABLE_NAME | COLUMN_NAME | CONSTRAINT_NAME |
+-----+-----+-----+-----+
| fk1          | myuser      | myuser_id   | f                |
| fk1          | product_order | customer_id | f2               |
| fk1          | product_order | product_id  | f1               |
+-----+-----+-----+-----+
3 rows in set (0.01 sec)
```

InnoDB テーブルの外部キーに関する情報は、`INFORMATION_SCHEMA` データベースの `INNODB_FOREIGN` テーブルおよび `INNODB_FOREIGN_COLS` テーブルにもあります。

InnoDB テーブルおよび NDB テーブルでは、外部キーがサポートされます。

1.7.3.3 無効なデータに対する制約の施行

デフォルトでは、MySQL 8.0 は無効または不適切なデータ値を拒否し、発生したステートメントを中断します。サーバーが厳密な SQL モード (セクション5.1.11 「サーバー SQL モード」を参照) を無効にすることで、データ入力のために有効な値に強制的に強制するなど、無効な値を忘れやすいようにこの動作を変更できますが、これはお薦めしません。

旧バージョンの MySQL では、デフォルトで忘れてた動作が採用されていました。この動作の詳細は、[Constraints on Invalid Data](#) を参照してください。

1.7.3.4 ENUM および SET の制約

ENUM および SET カラムによって、特定の値セットのみを含むカラムを効率的に定義することができます。 [セクション11.3.5「ENUM 型」](#) および [セクション11.3.6「SET 型」](#) を参照してください。

厳密モードが無効になっていないかぎり (推奨されませんが、[セクション5.1.11「サーバー SQL モード」](#) を参照)、ENUM または SET カラムの定義は、カラムに入力された値に対する制約として機能します。これらの条件を満たさない値には、エラーが発生します。

- ENUM 値は、カラム定義に一覧表示されている値のいずれかが、それと内部数値的に同等のものである必要があります。その値はエラー値 (つまり、0 または空の文字列) にはなりません。ENUM('a','b','c') として定義されたカラムでは、"、'd'、'ax' などの値は無効であり拒否されます。
- SET 値は空の文字列にするか、カンマで区切られたカラム定義に一覧表示されている値だけから構成された値にする必要があります。SET('a','b','c') として定義されたカラムの場合、'd' や 'a,b,c,d' などの値は無効であり拒否されず。

無効な値に対するエラーは、INSERT IGNORE または UPDATE IGNORE を使用している場合、厳密モードで抑制できます。この場合、エラーではなく警告が発せられます。ENUM に対しては、その値はエラーメンバー (0) として挿入されます。SET に対しては、無効な部分文字列が削除されるという点を除いて、その値は与えられたとおりに挿入されます。たとえば、'a,x,b,y' は 'a,b' の値となります。

1.8 クレジット

次のセクションでは、MySQL を今日の形にするのを支援した開発者、貢献者、および支援者をリストします。

1.8.1 MySQL への貢献者

MySQL Server と MySQL マニュアルの全著作権はオラクル社およびその関連会社が保有しますが、MySQL ディストリビューションに何らかの形で貢献していただいた方々に感謝申し上げます。貢献者は、次のとおりです。順不同です。

- Gianmassimo Vigazzola <qwerg@mbox.vol.it> または <qwerg@tin.it>

Win32/NT への最初の移植。

- Per Eric Olsson

動的レコード形式に対する建設的な批評と実際のテスト。

- Irena Pancirov <irena@mail.yacc.it>

Borland コンパイラを使用した Win32 の移植。mysqlshutdown.exe および mysqlwatch.exe。

- David J. Hughes

シェアウェア SQL データベースの作成に尽力。MySQL AB の前身である TcX で、mSQL を始めましたが、それでは目的を達成できないことがわかったため、アプリケーションビルダー Unireg に対する SQL インタフェースの作成に転換しました。mysqladmin および mysql クライアントは、mSQL の対応するプログラムに大きな影響を受けています。MySQL 構文が mSQL のスーパーセットになるよう非常に努力しました。無料の mSQL プログラムを MySQL API に移植しやすくするため、API のアイデアの多くは mSQL に基づいたものです。MySQL ソフトウェアには、mSQL のコードは含まれません。ディストリビューションの 2 つのファイル (client/insert_test.c および client/select_test.c) は、mSQL ディストリビューションの対応する (著作権のない) ファイルに基づいていますが、例として、コードを mSQL から MySQL Server に変換するために必要な変更を示すために変更されています。(mSQL の著作権は David J. Hughes にあります。)

- Patrick Lynch

<http://www.mysql.com/> の取得を支援します。

- Fred Lindberg
MySQL メーリングリストを処理する `qmail` を設定し、MySQL メーリングリストの管理に関して多大な支援をいただいています。
- Igor Romanenko <igor@frog.kiev.ua>
`mysqldump` (以前の `msqldump` ですが、Monty により移植および拡張されています)。
- Yuri Dario
MySQL OS/2 の移植の継続と拡張。
- Tim Bunce
`mysqlhotcopy` の作成者。
- Zarko Mocnik <zarko.mocnik@dem.si>
スロベニア語のソート。
- "TAMITO" <tommy@valley.ne.jp>
`_MB` 文字セットのマクロと、`ujis` および `sjis` 文字セット。
- Joshua Chamas <joshua@chamas.com>
同時挿入の基礎、拡張日付構文、NT でのデバッグ、および MySQL メーリングリストでの回答。
- Yves Carlier <Yves.Carlier@rug.ac.be>
ユーザーのアクセス権を表示するプログラム `mysqlaccess`。
- Rhys Jones <rhys@wales.com> (および GWE Technologies Limited)
初期 JDBC ドライバの 1 つ。
- Dr Xiaokun Kelvin ZHU <X.Zhu@brad.ac.uk>
初期 JDBC ドライバの 1 つと、ほかの MySQL 関連 Java ツールのさらなる開発。
- James Cooper <pixel@organic.com>
自分自身のサイトで検索可能なメーリングリストアーカイブのセットアップ。
- Rick Mehalick <Rick_Mehalick@i-o.com>
MySQL Server のグラフィカルな X クライアントである `xmysql`。
- Doug Sisk <sisk@wix.com>
Red Hat Linux 対応 MySQL の RPM パッケージの提供。
- Diemand Alexander V. <axeld@vial.ethz.ch>
Red Hat Linux-Alpha 対応 MySQL の RPM パッケージの提供。
- Antoni Pamies Olive <toni@readysoft.es>
Intel および SPARC 対応の多くの MySQL クライアントの RPM バージョンの提供。
- Jay Bloodworth <jay@pathways.sde.state.sc.us>
MySQL バージョン 3.21 対応 RPM バージョンの提供。
- David Sacerdote <davids@secnet.com>

DNS ホスト名のセキュアなチェックの方針。

- Wei-Jou Chen <jou@nematic.iew.nctu.edu.tw>

中国語 (BIG5) キャラクタのサポート。

- Wei He <hewei@mail.ied.ac.cn>

中国語 (GBK) 文字セット対応の多くの機能。

- Jan Pazdziora <adelton@fi.muni.cz>

チェコ語のソート順序。

- Zeev Suraski <bourbon@netvision.net.il>

[FROM_UNIXTIME\(\)](#) の時間書式、[ENCRYPT\(\)](#) 関数、および [bison](#) のアドバイザー。メーリングリストのアクティブなメンバー。

- Luuk de Boer <luuk@wxs.nl>

ベンチマークスイートを [DBI/DBD](#) に移植 (および拡張)。 [crash-me](#) とベンチマークの実行の大きな助けとなりました。一部の新規日付関数。 [mysql_setpermission](#) スクリプト。

- Alexis Mikhailov <root@medinf.chuvashia.su>

ユーザー定義関数 (UDF)、[CREATE FUNCTION](#) および [DROP FUNCTION](#)。

- Andreas F. Bobak <bobak@relog.ch>

ユーザー定義関数の [AGGREGATE](#) 拡張。

- Ross Wakelin <R.Wakelin@march.co.uk>

MySQL-Win32 用 InstallShield の設定の支援。

- Jethro Wright III <jetman@li.net>

[libmysql.dll](#) ライブラリ。

- James Pereria <jpereira@iafrica.com>

MySQL Server を管理する Win32 GUI ツール Mysqlmanager。

- Curt Sampson <cjs@portal.ca>

MIT-pthreads の NetBSD/Alpha および NetBSD 1.3/i386 への移植。

- Martin Ramsch <m.ramsch@computer.org>

MySQL チュートリアル内のサンプル。

- Steve Harvey

[mysqlaccess](#) の安全性の改善。

- Persistent Systems Private Limited の Konark IA-64 Centre

MySQL Server の Win64 への移植の支援。

- Albert Chin-A-Young.

Tru64 用アップデートの構成、大きなファイルのサポート、優れた TCP ラッパーのサポート。

- John Birrell

OS/2 用 `pthread_mutex()` のエミュレーション。

- Benjamin Pflugmann

`INSERTS` を処理する拡張 `MERGE` テーブル。MySQL メーリングリストのアクティブなメンバー。

- Jocelyn Fournier

(特に MySQL 4.1 サブクエリーコードの) 無数のバグを発見しレポートしました。

- Marc Liyanage

OS X パッケージの保守および OS X パッケージの作成方法に関する非常に有益なフィードバックを提供。

- Robert Rutherford

QNX 移植に関する非常に有益な情報とフィードバックを提供。

- NDB Cluster の以前の開発者

夏期の学生、修士論文生、従業員など、多くの方がさまざまな方法で参加しました。合計で 100 名を超え、ここで全員の名前を挙げることはできません。注目に値するのは、1999 年までコードベースの約 1/3 に貢献してきた Ataulah Dabaghi です。また、NDB Cluster のアーキテクチャーの基礎にブロック、信号、およびクラッシュ追跡機能を提供した AXE システムの開発者にも深く感謝いたします。1992 年から現在まで、開発に予算を割り当てるのに十分なアイデアであることを確信した方々も称賛に値します。

- Google 社

MySQL ディストリビューション、Mark Callaghan の SMP Performance パッチその他のパッチへの貢献で、Google 社に感謝いたします。

その他の貢献者、バグ発見者、およびテスター: James H. Thompson、Maurizio Menghini、Wojciech Tryc、Luca Berra、Zarko Mocnik、Wim Bonis、Elmar Haneke、<jehamby@lightside>、<psmith@BayNetworks.com>、<duane@connect.com.au>、Ted Deppner <ted@psyber.com>、Mike Simons、Jaakko Hyvatti。

メーリングリストのメンバーから提供された多くのバグレポートおよびパッチ。

MySQL メーリングリストの質問に回答いただいた方々に感謝いたします。

- Daniel Koch <dkoch@amcity.com>

lrix セットアップ。

- Luuk de Boer <luuk@wxs.nl>

ベンチマークの質問。

- Tim Sailer <tps@users.buoy.com>

`DBD::mysql` の質問。

- Boyd Lynn Gerber <gerberb@zenez.com>

SCO 関連の質問。

- Richard Mehalick <RM186061@shellus.com>

`xmysql` 関連の質問と基本的なインストールの質問。

- Zeev Suraski <bourbon@netvision.net.il>

Apache モジュール構成の質問 (ログおよび権限)、PHP 関連の質問、SQL 構文関係の質問およびその他の一般的な質問。

- Francesc Guasch <frankie@citel.upc.es>
一般的な質問。
- Jonathan J Smith <jsmith@wtp.net>
Linux の OS 固有事項、SQL 構文、および何らかの作業を要するその他の事項に関する質問。
- David Sklar <sklar@student.net>
PHP および Perl からの MySQL の使用。
- Alistair MacDonald <A.MacDonald@uel.ac.uk>
柔軟性があり、Linux およびおそらく HP-UX も操作できます。
- John Lyon <jllyon@imag.net>
.rpm ファイルまたはソースからのコンパイルのいずれかによる、MySQL の Linux システムへのインストールに関する質問。
- Lorvid Ltd. <lorvid@WOLFENET.com>
簡単な請求/ライセンス/サポート/著作権の問題。
- Patrick Sherrill <patrick@coconet.com>
ODBC および VisualC++ インタフェースに関する質問。
- Randy Harmon <rjharmon@uptimecomputers.com>
DBD、Linux、一部の SQL 構文に関する質問。

1.8.2 ドキュメント作成者および翻訳者

MySQL のドキュメント作成、およびドキュメントまたはエラーメッセージ翻訳にご協力いただいた方々を次に記載します。

- Kim Aldale
Monty と David の初期の英語ドキュメントのリライト支援。
- Michael J. Miller Jr. <mke@terrapin.turbolift.com>
初回の MySQL マニュアルの作成。FAQ (ずいぶん前に MySQL マニュアルになりました) の多くのスペリングまたは言語の修正。
- Yan Cailin
2000 年初めに、Big5 および HK コード化バージョンが準拠している簡体字中国語に MySQL リファレンスマニュアルをはじめて翻訳。
- Jay Flaherty <fty@mediapulse.com>
マニュアル内の Perl DBI/DBD セクションの大部分に携わりました。
- Paul Southworth <pauls@etext.org>, Ray Loyzaga <yar@cs.su.oz.au>
リファレンスマニュアルの校正。
- Therrien Gilbert <gilbert@ican.net>, Jean-Marc Pouyot <jmp@scalaire.fr>
フランス語のエラーメッセージ。
- Petr Snajdr, <snajdr@pvt.net>

チェコ語のエラーメッセージ。

- Jaroslaw Lewandowski <jtotel@itnet.com.pl>

ポーランド語のエラーメッセージ。

- Miguel Angel Fernandez Roiz

スペイン語のエラーメッセージ。

- Roy-Magne Mo <rmo@www.hivolda.no>

ノルウェー語のエラーメッセージと MySQL 3.21.xx のテスト。

- Timur I. Bakeyev <root@timur.tatarstan.ru>

ロシア語のエラーメッセージ。

- <brenno@dewinter.com> & Filippo Grassilli <phil@hyppo.com>

イタリア語のエラーメッセージ。

- Dirk Munzinger <dirk@trinity.saar.de>

ドイツ語のエラーメッセージ。

- Billik Stefan <billik@sun.uniag.sk>

スロバキア語のエラーメッセージ。

- Stefan Saroiu <tzoompy@cs.washington.edu>

ルーマニア語のエラーメッセージ。

- Peter Feher

ハンガリー語のエラーメッセージ。

- Roberto M. Serqueira

ポルトガル語のエラーメッセージ。

- Carsten H. Pedersen

デンマーク語のエラーメッセージ。

- Arjen Lentz

オランダ語のエラーメッセージの以前の部分翻訳の完成 (一貫性とスペリングについてもチェック)。

1.8.3 MySQL をサポートするパッケージ

MySQL で多くのユーザーが使用するもっとも重要な API/パッケージ/アプリケーションの作成者/保守者のリストを次に記載します。

すべてのパッケージを記載すると保守が非常に困難になるため、すべてを記載することはできません。ほかのパッケージについては、<http://solutions.mysql.com/software/> のソフトウェアポータルを参照してください。

- Tim Bunce、Alligator Descartes

[DBD](#) (Perl) インタフェース。

- Andreas Koenig <a.koenig@mind.de>

MySQL Server 用 Perl インタフェース。

- Jochen Wiedmann <wiedmann@neckar-alb.de>
Perl `DBD::mysql` モジュールの保守。
- Eugene Chan <eugene@acenet.com.sg>
MySQL Server 用 PHP の移植。
- Georg Richter
MySQL 4.1 のテストとバグ検出。MySQL 4.1 で使用するための新規 PHP 5.0 `mysqli` 拡張 (API)。
- Giovanni Maruzzelli <maruzz@matrice.it>
iODBC (Unix ODBC) の移植用。
- Xavier Leroy <Xavier.Leroy@inria.fr>
LinuxThreads (Linux 上で MySQL Server が使用) の作成者。

1.8.4 MySQL の作成に使用されたツール

MySQL を作成するために使用したツールの一覧を次に示します。ここで、これらのツールを作成した方々に感謝申し上げます。これらのツールがなければ、MySQL は今日の形になりませんでした。

- フリーソフトウェア財団 (Free Software Foundation)
優れたコンパイラ (`gcc`)、優れたデバッガ (`gdb`)、および `libc` ライブラリを提供 (このライブラリの `strto.c` を使用して、Linux で動作するいくつかのコードを作成しました)。
- フリーソフトウェア財団と XEmacs 開発チーム
非常に優れたエディタ/環境を提供。
- Julian Seward
MySQL のバグ検出に役立った非常に優れたメモリーチェッカーツール `valgrind` の作成者。これがなければバグ検出は困難だったでしょう。
- Dorothea Lütkehaus および Andreas Zeller
`gdb` に対する優れたグラフィカルフロントエンドである `DDD` (データ表示デバッガ) を提供。

1.8.5 MySQL のサポーター

MySQL Server と MySQL マニュアル の全著作権はオラクル社およびその関連会社が保有しますが、新機能の開発に対する融資、MySQL Server の開発用のハードウェアの提供など、MySQL Server の開発を助成していただいた次の会社に感謝申し上げます。

- VA Linux / Andover.net
レプリケーションに対する資金提供。
- NuSphere
MySQL マニュアルの編集。
- Stork Design studio
1998-2000 の間に使用されている MySQL web サイト。
- Intel
Windows および Linux プラットフォーム上での開発に貢献。

- Compaq
Linux/Alpha 上での開発に貢献。
- SWSOft
埋め込み `mysqld` バージョンでの開発。
- FutureQuest
`--skip-show-database` オプション。

第 2 章 MySQL のインストールとアップグレード

目次

2.1 一般的なインストールガイド	83
2.1.1 サポートされるプラットフォーム	83
2.1.2 インストールする MySQL のバージョンと配布の選択	84
2.1.3 MySQL の取得方法	85
2.1.4 MD5 チェックサムまたは GnuPG を用いたパッケージの完全性の確認	85
2.1.5 インストールのレイアウト	97
2.1.6 コンパイラ固有のビルドの特徴	97
2.2 一般的なバイナリを使用した MySQL の Unix/Linux へのインストール	98
2.3 Microsoft Windows に MySQL をインストールする	100
2.3.1 Microsoft Windows 上での MySQL のインストールレイアウト	103
2.3.2 インストールパッケージの選択	103
2.3.3 MySQL Installer for Windows	105
2.3.4 noinstall ZIP アーカイブを使用した Microsoft Windows への MySQL のインストール	127
2.3.5 Microsoft Windows MySQL Server インストールのトラブルシューティング	134
2.3.6 Windows でのインストール後の手順	136
2.3.7 Windows プラットフォームの制限事項	137
2.4 macOS への MySQL のインストール	139
2.4.1 macOS への MySQL のインストールに関する一般的なノート	139
2.4.2 ネイティブパッケージを使用した macOS への MySQL のインストール	140
2.4.3 MySQL 起動デーモンのインストールおよび使用	145
2.4.4 MySQL Preference Pane のインストールと使用	148
2.5 Linux に MySQL をインストールする	152
2.5.1 MySQL Yum リポジトリを使用して MySQL を Linux にインストールする	153
2.5.2 MySQL APT リポジトリを使用して MySQL を Linux にインストールする	157
2.5.3 MySQL SLES リポジトリを使用して MySQL を Linux にインストールする	157
2.5.4 Oracle の RPM パッケージを使用した Linux への MySQL のインストール	157
2.5.5 オラクルからの Debian パッケージを使用して MySQL を Linux にインストールする	161
2.5.6 Docker での Linux への MySQL のデプロイ	163
2.5.7 ネイティブソフトウェアリポジトリから MySQL を Linux にインストールする	173
2.5.8 Juju を使用した Linux への MySQL のインストール	175
2.5.9 systemd を使用した MySQL Server の管理	175
2.6 Unbreakable Linux Network (ULN) を使用した MySQL のインストール	180
2.7 Solaris への MySQL のインストール	180
2.7.1 Solaris PKG を使用して Solaris に MySQL をインストールする	181
2.8 FreeBSD に MySQL をインストールする	182
2.9 ソースから MySQL をインストールする	183
2.9.1 ソースのインストール方法	183
2.9.2 ソースインストールの前提条件	184
2.9.3 ソースインストールの MySQL のレイアウト	185
2.9.4 標準ソース配布を使用して MySQL をインストールする	185
2.9.5 開発ソースツリーを使用して MySQL をインストールする	189
2.9.6 SSL ライブラリサポートの構成	191
2.9.7 MySQL ソース構成オプション	191
2.9.8 MySQL のコンパイルに関する問題	218
2.9.9 MySQL の構成とサードパーティーツール	219
2.9.10 MySQL Doxygen ドキュメントコンテンツの生成	220
2.10 インストール後のセットアップとテスト	220
2.10.1 データディレクトリの初期化	221
2.10.2 サーバーの起動	226
2.10.3 サーバーのテスト	228
2.10.4 初期 MySQL アカountの保護	230
2.10.5 MySQL を自動的に起動および停止する	232
2.11 MySQL のアップグレード	233

2.11.1 始める前に	233
2.11.2 アップグレードパス	234
2.11.3 MySQL のアップグレードプロセスの内容	234
2.11.4 MySQL 8.0 での変更	238
2.11.5 アップグレード用のインストールの準備	248
2.11.6 Unix/Linux での MySQL バイナリまたはパッケージベースのインストールのアップグレード	251
2.11.7 MySQL Yum リポジトリを使用する MySQL のアップグレード	255
2.11.8 MySQL APT リポジトリを使用する MySQL のアップグレード	257
2.11.9 MySQL SLES リポジトリを含む MySQL のアップグレード	257
2.11.10 Windows 上の MySQL をアップグレードする	257
2.11.11 MySQL の Docker インストールのアップグレード	259
2.11.12 アップグレードのトラブルシューティング	259
2.11.13 テーブルまたはインデックスの再作成または修復	259
2.11.14 MySQL データベースのほかのマシンへのコピー	261
2.12 MySQL のダウングレード	262
2.13 Perl のインストールに関する注釈	262
2.13.1 Unix に Perl をインストールする	262
2.13.2 Windows に ActiveState Perl をインストールする	263
2.13.3 Perl DBI/DBD インタフェース使用の問題	263

この章では MySQL の取得とインストールの方法を説明します。最初に手順の概要を、後半のセクションで詳細を説明します。MySQL をはじめてインストールするのではなく、現行のバージョンの MySQL を新しいバージョンにアップグレードする場合は、[セクション2.11「MySQL のアップグレード」](#)を参照してください。アップグレードの手順およびアップグレード前に考慮すべき問題点に関する情報を記載しています。

別のデータベースシステムから MySQL に移行する場合は、[セクションA.8「MySQL 8.0 FAQ: 移行」](#)を参照してください。この [セクションA.8「MySQL 8.0 FAQ: 移行」](#)には、移行の問題に関する一般的な質問への回答が含まれています。

MySQL のインストールは、一般に次の手順に従います。

1. 使用しているプラットフォームで MySQL が動作し、サポートされているかどうかを判断します。

すべてのプラットフォームが MySQL の実行に等しく適しているわけではなく、MySQL が実行されることがわかっているすべてのプラットフォームが Oracle Corporation によって公式にサポートされているわけではないことに注意してください。公式にサポートされているプラットフォームの詳細は、MySQL web サイトの <https://www.mysql.com/support/supportedplatforms/database.html> を参照してください。

2. インストールする配布を選択します。

MySQL には使用可能なバージョンがいくつかあり、ほとんどが複数の配布形式で使用可能です。バイナリ (コンパイル済み) プログラムあるいはソースコードを含むパッケージ済み配布を選択できます。不確かな場合にはバイナリ配布をご使用ください。Oracle では、最近の開発を表示して新しいコードをテストするユーザーのために、MySQL ソースコードへのアクセスも提供されます。どのバージョンのどの配布を選択したらよいかは [セクション2.1.2「インストールする MySQL のバージョンと配布の選択」](#)を参照してください。

3. インストールする配布をダウンロードします。

その手順は、[セクション2.1.3「MySQL の取得方法」](#)を参照してください。配布の完全性を確認するには、[セクション2.1.4「MD5 チェックサムまたは GnuPG を用いたパッケージの完全性の確認」](#)の指示に従ってください。

4. 配布をインストールします。

MySQL をバイナリの配布からインストールするには、[セクション2.2「一般的なバイナリを使用した MySQL の Unix/Linux へのインストール」](#)を参照してください。

MySQL をソースの配布からインストールする、あるいは現在の開発ソースツリーからインストールする場合は、[セクション2.9「ソースから MySQL をインストールする」](#)を参照してください。

5. インストール後のセットアップが必要な場合は実行します。

MySQL のインストール後に、MySQL サーバーが正しく動作していることを確認するには、[セクション2.10「インストール後のセットアップとテスト」](#)を参照してください。[セクション2.10.4「初期 MySQL アカウントの保護」](#)の情報も参照してください。このセクションでは、最初の MySQL `root` ユーザーアカウントを割り当てるまでパスワードを持たないを保護する方法について説明します。このセクションはバイナリ配布およびソース配布の MySQL のインストールの両方に適用されます。

- MySQL のベンチマークスクリプトを実行する場合、Perl の MySQL サポートが必要になります。[セクション 2.13「Perl のインストールに関する注釈」](#)を参照してください。

さまざまなプラットフォームおよび環境での MySQL のインストール手順は、プラットフォーム別に提供されます。

- Unix、Linux、FreeBSD

一般的なバイナリ (たとえば、`.tar.gz` パッケージ) を使用して MySQL をほとんどの Linux および Unix プラットフォームにインストールする手順は、[セクション2.2「一般的なバイナリを使用した MySQL の Unix/Linux へのインストール」](#)を参照してください。

MySQL をすべてソースコード配布またはソースコードリポジトリからビルドする場合の詳細は、[セクション 2.9「ソースから MySQL をインストールする」](#)を参照してください。

インストール、構成、およびソースからのビルドに関するプラットフォーム固有のヘルプは、対応するプラットフォームのセクションを参照してください。

- Linux (配布固有の方法を含む) は、[セクション2.5「Linux に MySQL をインストールする」](#)を参照してください。
- IBM AIX は、[セクション2.7「Solaris への MySQL のインストール」](#)を参照してください。
- FreeBSD は、[セクション2.8「FreeBSD に MySQL をインストールする」](#)を参照してください。

- Microsoft Windows

MySQL Installer または圧縮バイナリを使用して Microsoft Windows に MySQL をインストールする手順は、[セクション2.3「Microsoft Windows に MySQL をインストールする」](#)を参照してください。

Microsoft Visual Studio を使用して MySQL をソースコードからビルドする方法の詳細は、[セクション2.9「ソースから MySQL をインストールする」](#)を参照してください。

- macOS

バイナリパッケージ形式とネイティブ PKG 形式の両方の使用を含む、macOS へのインストールについては、[セクション2.4「macOS への MySQL のインストール」](#)を参照してください。

macOS 起動デーモンを使用して MySQL を自動的に起動および停止する方法の詳細は、[セクション2.4.3「MySQL 起動デーモンのインストールおよび使用」](#)を参照してください。

MySQL Preference Pane については、[セクション2.4.4「MySQL Preference Pane のインストールと使用」](#)を参照してください。

2.1 一般的なインストールガイド

次の数セクションには、配布の選択、ダウンロード、および確認に必要な情報が含まれています。この章の次のセクション以降で、選択した配布のインストールの方法を説明します。バイナリ配布については、[セクション2.2「一般的なバイナリを使用した MySQL の Unix/Linux へのインストール」](#)の指示、または使用しているプラットフォームに対応するセクションがあればそちらを参照してください。MySQL をソースからビルドするには、[セクション2.9「ソースから MySQL をインストールする」](#)を参照してください。

2.1.1 サポートされるプラットフォーム

MySQL プラットフォームのサポートは時間の経過とともに進化します。最新の更新については、<https://www.mysql.com/support/supportedplatforms/database.html>を参照してください。

2.1.2 インストールする MySQL のバージョンと配布の選択

MySQL のインストールを準備するときに、使用するバージョンと配布形式 (バイナリまたはソース) を決定します。

まず、開発リリースと General Availability (GA) リリースのどちらをインストールするかを決定します。開発リリースには最新の機能がありますが、本番での使用は推奨されません。GA リリースは、本番リリースまたは安定リリースとも呼ばれ、本番で使用するためのものです。最新の GA リリースを使用することをお勧めします。

MySQL 8.0 のネーミングスキームでは、3 つの数字とオプションの接尾辞 (mysql-8.0.1-dmr など) で構成されるリリース名を使用します。リリース名の番号は次のように解釈されます。

- 最初の番号 (8) はメジャーバージョン番号です。
- 2 番目の番号 (0) はマイナーバージョン番号です。メジャー番号とマイナー番号と一緒にリリースシリーズ番号を構成します。シリーズ番号は、安定した機能セットを説明します。
- 3 番目の番号 (1) はリリースシリーズ内でのバージョン番号です。これは、新しいバグ修正リリースごとに増分されます。ほとんどの場合、シリーズ内の最新バージョンが最適な選択肢です。

リリース名には、リリースの安定性レベルを示す接尾辞を含めることもできます。シリーズ内のリリースは、サフィクスが順に進展していくことで、安定性レベルの改善を示します。サフィクスには次のようなものがあります。

- dmr は、開発マイルストーンリリース (DMR) を示します。MySQL 開発では、マイルストーンモデルを使用します。マイルストーンモデルでは、各マイルストーンに完全にテストされた機能の小さなサブセットが導入されます。あるマイルストーンから次のマイルストーンに至るまで、これらの早期リリースを試行するコミュニティメンバーから提供されたフィードバックに基づいて、機能インタフェースが変更されたり、機能が削除されたりする場合があります。マイルストーンリリース内の機能は、試作レベルの品質であるとみなされます。
- rc は、リリース候補 (RC) を示します。リリース候補は安定しているとみなされ、すべての MySQL 内部テストに合格しています。RC リリースでは新機能がまだ導入されている可能性があります。フォーカスはバグの修正に移り、シリーズの前半で導入された機能を安定させます。
- 接尾辞がない場合は、General Availability (GA) または Production リリースを示します。GA リリースは安定しており、以前のリリース段階を正常に通過しており、信頼性が高く、深刻なバグがないと考えられ、本番システムでの使用に適しています。

シリーズ内での開発は DMR リリースから始まり、その後 RC リリースが続き、最後に GA ステータスリリースに到達します。

インストールする MySQL のバージョンを選択した後、オペレーティングシステムにインストールする配布形式を決定します。ほとんどのユースケースでは、バイナリ配布を選択するのが適切です。バイナリ配布は、Linux の RPM パッケージや macOS の DMG パッケージなど、多くのプラットフォームでネイティブ形式で使用できます。配布は、Zip アーカイブや圧縮 tar ファイルなど、より一般的な形式でも利用可能です。Windows では、[MySQL Installer](#) を使用してバイナリ配布をインストールできます。

状況によっては、ソース配布から MySQL をインストールすることをお勧めします:

- MySQL を明示的な場所にインストールしたい場合。標準のバイナリの配布はインストールの場所にかかわらずすぐに動作しますが、MySQL コンポーネントを希望する場所に配置することで柔軟性をさらに向上させる必要に駆られる場合があります。
- 標準バイナリ配布に含まれていない可能性のある機能を使用して `mysqld` を構成する場合。機能の可用性を確保するために使用される最も一般的な追加オプションのリストを次に示します:
 - TCP ラッパーサポートのための `-DWITH_LIBWRAP=1`。
 - 圧縮に依存する機能のための `-DWITH_ZLIB={system|bundled}`
 - デバッグサポートのための `-DWITH_DEBUG=1`

詳細は [セクション 2.9.7 「MySQL ソース構成オプション」](#) を参照してください。

- `mysqld` を、標準のバイナリの配布に含まれる一部の機能を使用しないように構成する場合。

- あなたは、MySQL を構成する C および C++ コードを読み取ったり変更しようと考えています。この目的のために、ソース配布を取得します。
- ソースの配布には、バイナリの配布より多くのテストおよび例が含まれています。

2.1.3 MySQL の取得方法

MySQL の最新バージョンおよびダウンロードの説明書については、ダウンロードページ <https://dev.mysql.com/downloads/> を参照してください。

パッケージ管理システムとして Yum を使用する RPM ベースの Linux プラットフォームでは、[MySQL Yum Repository](#) を使用して MySQL をインストールできます。詳細は、[セクション2.5.1 「MySQL Yum リポジトリを使用して MySQL を Linux にインストールする」](#) を参照してください。

Debian ベースの Linux プラットフォームの場合、MySQL は「[MySQL APT リポジトリ](#)」を使用してインストールできます。詳細は、[セクション2.5.2 「MySQL APT リポジトリを使用して MySQL を Linux にインストールする」](#) を参照してください。

SUSE Linux Enterprise Server (SLES) プラットフォームでは、[MySQL SLES Repository](#) を使用して MySQL をインストールできます。詳細は、[セクション2.5.3 「MySQL SLES リポジトリを使用して MySQL を Linux にインストールする」](#) を参照してください。

最新の開発ソースの取得については、[セクション2.9.5 「開発ソースツリーを使用して MySQL をインストールする」](#) を参照してください。

2.1.4 MD5 チェックサムまたは GnuPG を用いたパッケージの完全性の確認

ニーズに合った MySQL パッケージをダウンロードした後、インストールする前に、パッケージがそのまま、改ざんされていないことを確認します。完全性の確認には 3 つの方法があります。

- MD5 チェックサム
- [GnuPG](#) (GNU Privacy Guard) を使用した暗号署名
- RPM パッケージの場合、内蔵 RPM 完全性確認メカニズム

次のセクションではこれらの方法の使用方法について説明します。

MD5 チェックサムまたは GPG 署名が一致しない場合には、別のミラーサイトから対応するパッケージを再度ダウンロードします。

2.1.4.1 MD5 チェックサムの確認

MySQL パッケージをダウンロードしたら、MD5 チェックサムが MySQL のダウンロードページのチェックサムと一致することを確認してください。各パッケージには固有のチェックサムがあり、ダウンロードしたパッケージに対して確認できます。各 MySQL 製品のダウンロードページに正しい MD5 チェックサムがリストされます。ダウンロードするファイル (製品) の MD5 チェックサムと比較する必要があります。

各オペレーティングシステムおよびセットアップで、MD5 チェックサムをチェックするための独自のツールが提供されています。通常、このコマンドは `md5sum` という名前か、または `md5` という名前で、一部のオペレーティングシステムではまったく提供されていません。Linux の場合は GNU テキストユーティリティパッケージの一部で、さまざまなプラットフォームに利用できます。ソースコードを <http://www.gnu.org/software/textutils/> からダウンロードすることもできます。OpenSSL がインストールされている場合は、代わりにコマンド `openssl md5 package_name` を使用できます。`md5` コマンド行ユーティリティの Windows 実装は、<http://www.fourmilab.ch/md5/> から入手できます。`winMd5Sum` はグラフィカルな MD5 チェックツールで、<http://www.nullriver.com/index/products/winmd5sum> から入手できます。Microsoft Windows の例では、`md5.exe` という名前を想定しています。

Linux および Microsoft Windows の例

```
shell> md5sum mysql-standard-8.0.29-linux-i686.tar.gz
```

```
aaab65abbec64d5e907dcd41b8699945 mysql-standard-8.0.29-linux-i686.tar.gz
```

```
shell> md5.exe mysql-installer-community-8.0.29.msi
aaab65abbec64d5e907dcd41b8699945 mysql-installer-community-8.0.29.msi
```

結果のチェックサム (16 進数の文字列) が、ダウンロードページの対応するパッケージの真下のチェックサムと一致していることを確認してください。

注記

アーカイブファイルの内部に含まれるファイルではなく、アーカイブファイル (たとえば .zip、.tar.gz、または .msi ファイル) のチェックサムを必ず確認してください。つまり、内容を抽出する前のファイルを確認します。

2.1.4.2 GnuPG を使用した署名確認

パッケージの完全性および信頼性を確認する別の方法は、暗号署名を使用するものです。これは、[MD5 チェックサム](#)を使用するより信頼性が高いですが、手間がかかります。

MySQL のダウンロード可能なパッケージは、[GnuPG](#) (GNU Privacy Guard) を使用して署名されます。[GnuPG](#) は、Phil Zimmerman 氏の周知の Pretty Good Privacy (PGP) のオープンソース版です。ほとんどの Linux の配布にはデフォルトで [GnuPG](#) がインストールされています。それ以外の場合は、[GnuPG](#) の詳細およびそれを取得してインストールする方法について、<http://www.gnupg.org/> を参照してください。

特定のパッケージの署名を確認するには、まず公開 GPG ビルド鍵を取得する必要があります。これは <http://pgp.mit.edu/> からダウンロードできます。取得する鍵は mysql-build@oss.oracle.com という名前です。または、次のテキストから直接キーをコピーして貼り付けることもできます:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1

mQGibD4+owwRBAC14GifUfCyEDSlePvEW3SAFUdJBtoQHH/nJKZyQT7h9bPIUWC3
RODjQRyCITRrdwyrKUGku2FmeVGwn2u2WmDMNABLnpPrWPKbDck96+OmSLN9brZ
fw2vOUgCmYv2hW0hyDHuvYIQA/BThQoADgj8AW6/0Lo7V1W9/8VuHP0gQwCgvzV3
BqOxRznNCRcRxAuAuVztHRcEAJooQK1+iSiunZMYD1WufeXfshc57S/+yeJkegNW
hxr9pRwVArNYJdDRT+rf2RUe3vpquKNQU/hnEIUHRJQqYH08gTxvxXNQC7fJYL
K2HtkrPbP72vvsEKMYhhr0eKCbtlGfls9krjJ6sBgACyP/Vb7hiPwxh6rDZ7ITNe
kYpXBACmWpP8NJTkamEnPCia2Z0OHODANwpUkP4317jsDmgtoBZX9qnrAXw+uNDI
QJEXM6FSbi0LLtZciNIYsafwAPEOMDKpMqAK6IyisNtPvaLd8IH0bPAnWqcyefep
rv0sxxqUEMcM3o7wmgfN83POkDasDbs3pjjwPhxvhz6//62zQJ7Q2TXITUUwqUmVs
ZWFzZSBFbmdpbmVlcmluZyA8bXlzcWwtYnVpbGRAb3NzLm9yYWNsZS5jb20+igwE
ExECACwCGyMCHgECF4ACGQEGCwkIBwMChUKCQgCawUWAgMBAAUCXEBY+wUJl87e
5AAKCRcMcY07UHLh9RZPAJ9uvm0zlfzCN+DHxHVaoFLFjdVYTCfborsC9tmEZYa
whhogeBkZkorbylaQTEQIAKQIblwYLCQgHAWlEFQIIAwQWAgMBAh4BAheAAhkB
BQJtAdRmBQkaZsVLaAoEJlxjTtQcuH1X4MAoKNLWAbCBUj96637kv6Xa/fJuX5m
AJwPtmgDfJue2iuhXdtRfEPT19SB6ohmBBMRagAmAhsjBgsJCAcDAGQVAggDBBYC
AwECHgECF4AFk53PioFCRP7AhUACgkQjHGN01By4fUmzACEJdfqgc9gWtUhgmcM
AOmG4RjwuxcAoKfM+U8yMOGELi+TRif7MtKEms6piGkEEcACACgYMGcwkIBwMC
BBUCCAMEFgIDAQIeAQIXgAIZAUQUZSR0gUJFTchqgAKCRcMcY07UHLh9YtAAJ9X
rA/ymImozPZn+A9ls8/uwMctSQCfaQMnq1dNkhH2kyByc3Rx9/W2xfqJARwEEAE
AAFYAIAS6+UACgkQ8a1C+GoXHivrWwfi/dtLk/x+NC2VMDlg+vOeM0qgG1lhXZfi
NsEisvGaz4m8fSFRGe+1bvvfDoKRxiGXU48RusjixzvBb6KTMuY6.JpOVfz9Dj3
H9spYriHa+i6rYsXZlpOhLiMnTy7NH2OvYCyNzSS/cilUACIFh/2NH8zNT5CNF
1uPNRs7HsHzzz7pOITjtTWiF4cq/lj6Z6CNrmdj+SiMvjYN9u6sdEKGtoNtpycgD
5HGKR+17Nd/7v56yhaUe4FpuvsNXig86K9tl6MUFs8CUyy7Hj3kVBZOUWVBM053k
nGdALSyqQR50DA3jMGKVI4ZnHje2RvWRmFT5YWoRTMxUSQPMLpBNikBHAQQAQIA
BgUCU1B+vQAKCRAohbcD0zcc8dWwCACWXXWDXicAWRUw+j3ph8dr9u3Siltjn3wB
c7clpckWPUlVtZ7lGgzlVB0s8hH4xgkSA+zLzL6u56mpUzskF17f113Ac9GgPm4
0M5vmmR9hwiD1HdZtGfbD+wklqgitNLoRcGdRf/+U7x09GhSS7Bf339suniX6sM
gXSC4L32D3zDjF5icGdb0kj+3lCRmp853dGyA3ff9yUiBkxkNawpi7Vz3D2ddU
pOF3BP+8NKpG4P2+srKgfBd4HidclSQc3rY4vaTkEkLkG0nNA6U4rYgOa7wiT
SsxFlntMMzaRg53QtK0+YkH0KuZR3GY8B7pi+tlgyCyVR7mlFo7rQEcbBABCAG
BQJWgVd0A0AJEEZu4b/gk4UKk9MH/Rnt7EccPjSJC5CrB2AU5LY2Dsr+PePI2ubP
WsEdG82qSjjGpbhIH8Lsg/PzQoGHIFWmmZWJktrT+dcgLbs3b2VwCNACwE8jOHd
UKQhEowgomNvHiBHKHjP4/IF68KOPiO/2mxyYkmpM7BWf3k857DJ5cT3iJLoN7
zF40qls/p09ePvnmStpplbtUn7XPO+1/Ee8VHzimABom52PkQluxNiVuzLVn3bS
Wqrd5eucqLk6yzjPXd2XhDHWC9Twp168GePru6EzQtusi0m6S/sHgEXqh/IxrfZV
JlljF75JvosZq5zeulr0i6kOij+Y1p6MFfihITZ1gTmk+CLvK2JASIEAECAAwF
Ak53QS4FAwASDQAACgkQlxC4m8pXrXwJ8Qf/be/UO9mqfoc2sMyhwMpn4/fdBWwF
```

LkA12FXQDOQMwH9HsmEjnfUgYKXschZRi+DuHXe1P7I8G2aQLubhBsQf9ejKvRF
TzuWMQkdq+6Koulxv6ofkCcv3d1xtO2W7nb5yxcPVBPrRfGFGebJvZa58DymCNg
yGtAU6AOz4veavNmI2+GIDQsY66+YDvZ+CxwzdYu+HDV9HmrJfc6deM0mnBn7SR
jqzxJPgoTQhihTav6q/R5/2p5NvQ/H84OgS6GjosfGc2duUDzCP/kheMRKfzuyKC
OHQPtUulj8++gfbHtEU7IDUX1So3c9n0PdpBeVclsDbpRnCNxQWU4mBot4kBlgQQ
AQIADAUCToi2GQUADABJ1AAAKCRCXELibyleftLZAB/9oRqx+NC98UQD/wlxCRytz
vi/MuPnbgQUPLHEap10tvEi33S/H/xDR/tcGoFY4cjAvo5skZXeWq93Av7PACUb
zkg0X0eSr2oL6wy66xfov72AwSuX+iUK68qtKaLqRLitM02y8aNRV/ggKvt7UMVg
mOvs5yLaYobvYGaFC2CIfkNot2MIVnQZCmnYBCwOktPGkExiu2yZMifcYgXqcpH
KVF659KeF2cM2d4xYm8HJqkSGGW306LFVSVeRwG+wbtgLPd5bM/T2b3fJ35ra
CSMLZearRTq8aygPI+XM7MM2eR946aw6jmOsgNBERbvldQj6LudAZj+8imcXV2K
iQEIbBABAgAMBQJQOmdnRBQMAEnUAAAOJEJcQuJvKV618AvllAIEF1ZJ+Ry7W0dKF
50eQ/ynaYUigzN92fW/9zB8yuQlNgkFJGidYmBci1tr1siziVJFusR3ZongAPGK
/SUTA9Y6KwLhmc7c5UnEHklq/NfdMZZWV5lykXlctqW0sbb+z1ecEd4G8u9j5ll
MO1B36rQayYAPoeXLX8dY4VyFLVGAQ00rWQBZYFzrpw16ATWbWGJP332NSfCk4zZq
6kXEW07q0st3YBgAAGdNQyEeZCa4d4pBR5X6189Kjg6GDnlcaiOF6HO6PLr9fRIL
r5ObCgU+G9gEHfiVwDEV9E+7/Bq2pYZ9whhkBqWQzdpXTNTM24uaEhE01EPO5zeC
O214q6mJASIEEAECAAwFAk6rpgEFAwASdQAACgkQlxC4m8pXrXzAhwf/f9O99z16
3Y5FZVlxexyqXQ/Mct9uKHuXEVnRFYbA49dQLD4S73N+zN7gn9fJEqcBo4w8qVUV
94U/ta/VbLkdtNREypIPM4XY8YE5Wfd9bfg3q1PbEiVjk995sBF+2+To99YYKst
gXPqjH0jUEyDmexOj+hsP8Rc63kvkx36V/Ba4ONRYFefGAhKDMigL2YAhc1UkG
tkGTuLmLCGwlv6iVDZ3R.Jf5375VfnaHv7eXfwQxCwE+BxG3CURjrfjaxMTmMP
yAG2rhDp5oTUEvqDYnbo5UxYOmrsjvF4FzXwqerEIXJUkUzSh0pp7RxHB/1CxD
s7D1F1hlgFuNikBlgQQAQIADAUCTrzZHAUDABJ1AAAKCRCXELibyleftMUjP/4s
07dREULIBnA1D6qr3fHsQJNZqbAuyDlvgGGLWzoyEDs+1JMFFlaa+EeLlo1386GU
2DammDC23p3B79uQhJeD2Z1TcVg4cA64Sff/CHca5coeRSrdAiudzU/cglGtXIP
/OaFamXgdMxAhloLFBShPCZkyb00phVa8+xelVDrk1HByZsNIXy/SSK8U26S2PVZ
2o14fWvKbJ1Aga8nBlgQQAQIADAUCWY/D8P2mi3RABiuZgfzkmKL5idH/wSkfnFKdTgJzssdCc
1jZEGVks5rFYCwOrJARHeP/tnsb/UxKBEsNtO7e3N2e/rLVnEyKvIO066hz7xZK/V
NBSp3k3q4XPK41IHyz2iQEIbBABAgAMBQJQOzqO8BQMAEnUAAAOJEJcQuJvKV618
2twH/0lZjXLN45nvlFfEJ75a+i9ZSLiQR8sHL4GpEScFKI0a0IT4IVAIY2RKg+
MAs2eHmOUfkuGws5jluRZ9RqKrc61sY0XQV9/7znY9Db16ghX04JjknOKs/fPi87
rvKkB/QxJWS8qbb/erRmW+cPNjbRXTFPS5JlWFWHA16ieFEpvdAgKv6nfvJvTq1r
jPDcnIA9CJN2SmUFx9Qx3SRc6ITbam1hfFnY6sCh6AUhXLI2f1mq1xH9PqEy42Um
68prRqTyJ7lox1g/UDDKeeUcAg7T1viZ7uXpS3Wq4zZo4yOpaJfLDR3pl5g2Zk
SNGTMO6aySE4OABt8i1Pc1Pm6AmJASIEEAECAAwFAk7yPFYFAwASdQAACgkQlxC4
m8pXrXzXiAf9FrXe0lgcPM+tYOWMLhv5gXJi2VUBaLxpyRmXk/JcmxInKq1GCd3y
D4/FLHnu3ZcCz/ukIPAbZXWIO06ewq0LWsRtklmJjWiedH+hGyaTv95VklorRIBd
8nBaJ6M98rjMBHTFwVwvJQFVf4FLRjQZqHlvjCkq2Dd9BWJpGXvr/gpKkmMJYNK
/ftZRCChb35N119WRpOhj9u808OPcqKvZBcPwFgV5cEBzmAC94J7JcD8+S8lk8
iUJMQGGL3QcmZOBovzh86h7KTSEBHLXl832z89H1hLeuLbnXoGLV3zeUFSxkv
1h35LhZLqIMDQRXLuUzXGhMBpLhPyGWRJ4kBlgQQAQIADAUCWYJfWUDABJ1AAAK
CRCXELibyleftABvB/9Cy69cjOqLgywITs3Cpg//40jmdhSAVxiiJivP6J5bubFH
DJJVTx541Dv54h4TG2BQuueQ4q1VcPsgw+rHcdhPymZGRz1rxddQQGh1Dv0Bod2c
3PJVSYPsRrSzcJkjhOtVrBdjK4mkZb5aFtza+Tor9kxjz4FcXVd4KAS+hHQHYHc
Ar8t2eOLzq8EFTULeGiSoNn+PVzvdzfndphK+8F2jfQ2UKuc01O7k0Yn9xZVx0
OG6fE1gStzLV7C5amWLRd8+Xh+MN0G8MgNglpBoExsEMMIPBYSUHa6lxdpMNMuib
rlyVncE9X8QOJhmt8K0sNn/EdbuldJNGYbDLt7O4iQEIbBABAgAMBQJPFdTCBQMA
EnUAAAOJEJcQuJvKV6184owH+wZulpezXnSxigeH1sig72QEXMrNd5DVHHCJdig3
bo+K5YmmN710/m5z+63XKUEWpd6/knajObgckThzWftNeK1SSFQGPmoYZP9EZnSU
7L+d/SupExbj842G5LYagrCyMGtlxRywWEmb172TKS/JOK0JiOdvYy+PHrZSu0D
TVQ7cJh1BmPsbz7zzxjmc15+7B7K7RHZHq45nDLolabwDacj7BXvBK0Ajz4QyJ
GQUJXC7q+88l+ptPvOXIE5nl/NbiCJOMI6d/bWN1KwYrC80fZuFaznfQcPyUaDw
yRaun+K3kEjj2wXecq+yMmLUEp01TKsUeOL50HD6hHH07W+JASIEEAECAAwFAk85
bQsFAwASdQAACgkQlxC4m8pXrXwKPQgAlkbUsTr7nkq+haOk0jKpaHWEbRMEGMrB
l3F7E+RDO6V/8y4Jtn04EYDc8GgZMBah+mOgelNq3y8jRMYV5jvtZxv2MWYFUcjM
kVBKeqhi/pGEjmuDmdt3DIPv3Z+fMTMRmAoc1981iY/go8PVPg/+nrR6cFK2xxnO
R8TAcikJBFESfkkORg1tDzjYv1B5ZIEkpplep15ahJBBQ7cpYhTdY6Yk0SzoJ8w
EdffLsaNxrRuWLRhWzZU7p9bFzfb/7OHc21dJnB7wkv5VvtgE+jiQw9tOKaf5hc
SgRYuF6heu+B25gc5Uu88lo409mZ7oxQ6hDCn7JHvzh0rhmSN+Kid4kBlgQQAQIA
DAUCT0qQRUDABJ1AAAKCRCXELibyleftC9UB/4o2ggJYM0CLxEP0GU8UKOh3+/
zm1DN7Qe4kY2iCif1plKHQaTgt5FfgRCFaiXcVv7WzGz/FnmxonR1leLi+kfRlwy
PPnoI/AWPCy/NO4C15KjnsSmsdDUpObwZ4KYsdilZR7ViJu2swdAlgnXBUwriRJR
7CK4TAKrTeonRgVsrVx8vt//8/cYj73CLq8oY/KK0iHiQrSwo44uyhdiFIAssyX
n6/2E+wOzgvPexNSNNROHQ8pbjq+NTY6GwKlGsaej3UTRwQ7psvKXz8y7xdzmOAr
/khGvxB5gjkx02pimjeia8v66aH6rbnoJMAovNUS4EHdHnul4rovC8Kf9iQEI
BBABAgAMBQJPVdsBQMAEnUAAAOJEJcQuJvKV618vEIALFXPBZcAO1SnQarBLzy
YMVZzumPvSXKnUHAO+6kApXPJ+qFRdUaSNshZxVKY9Zryblu4ol/fLUTt0CiiSD
lxD6L4GXEem4VYYC4I4P03bVsJnGITLFWQGhm27EmjVoTiD8Ch7kPq2EXr3dMRgzj
pdz+6aHGSUFodLTPXufDvW83bEWGaRVuTJKw+wLrcuRqQ+ucWJgJGwcE4zeHjZad
Jx1XUm1X+Bb173uiCussyjhhQVVNU7QEdrijuscaZ/H38wjUwNbyXDPB4I8quC1
knQ0wSHr7gKpM+E9nhiS14poRqU18u78/sJ2MUPXnQA6533C238/LP8JgqB+BiQ
BTSJASIEEAECAAwFAk9ng3cFAwASdQAACgkQlxC4m8pXrXxQRAf/UZlkkpFJj1om
9hIRz7gS+17YvTakSzpo+Tbc3C7aqKjpir6TIMK9cb9HGTHo2Xp1N3FtQL72NvO
6CcJpBURbvSyb4i0hrm/YcbUC4Y3eajWhkRS3iVGNFbc/rHthViz0r6Y5lhXX16

aVkdV5CIFWaf3BiUK0FnHrZiy4FPacUXCwEjv3uf8MpxV5oEmo8V5h1h4TL3obyUz
qrlmFrEMYE/12IkE8iR5KWCaF8eFyl56HL3PPI90JMQBXzhwsFoWCPuwjfm5w6sW
Ll/ZynwxlJ9CRz9c2vK6aJ8DRu3OfBKN1iiEcNEynksDnNXEm5Xkz3p5pYdq
e9BLzUQCdYkBlgQQAQIADAUCT3inRgUDABJ1AAAKCRXCXELibyletfGMKCADJ97qk
geBntQ+ZtIKSFyXznAugYQmbzJld8U6eGSQnQkM40Vd62UJZLdA8MjJWKS8y4A4L2
0cl14zs5tK6G9Q72BxQOW5xkxLLASw1/8WYeYebw7ZA+sPG/q9v3klkr3sv64mMa
enZtsxykexRGyCumxLjzlAcl1drWJGUYE2Kl6uzQS7jb+3PNBLoQvz6nb3YRZ+CG
Ly9D41SIK+fpnV8r4iqhu7r4LmAQ7Q1DF9aoGaYvn2+xlGyWHxJAUet4xkMNOLp6
k9RF1nbNe4l/sqeCB25CZHCTEvHdjSGTD2yJR5jfoWkwO9w8DZG1Q9WrWqki4hSB
l0cmcvO34pC1SJyziQEIBBAbAgAMBQJPinQFBQMAEnUAAAOJEJcQuJvK618CFEI
AJp5BbcV7+JBMRSvkoUcAWDoJSP2ug9zGw5FB8J90PDefKWCKs5Tjayf2TvM5ntq
5DE9SGaXblolwa74FozlglhMZ4AtY9Br+oyPJ5S844wpAmWmf6cNnEPFAHQkQ+b
dJYpRVNd9lzagJP261P3S+S9T2UeHVdOJBgWlq9Mbs4InZzWsnZfQ4Lsz0aPqe48
tkU8hw+nflby994qlwNolk/u+l/JbNz5zDY91oscXTRl2jV1qBgKYwwCXxyB39
fyVpRl+7QnqbTWcCICVFL+uuYpPOHjdoKNqhzEguAUQQLOB9msPTXfa2hG+32ZYg
5pzl5V7GCHq0K06u5Cj3TGJASIEEAECaAwFAk+cQEFAwASdQAACgkQlxC4m8pX
rXzi7AgAx8wJzNdD7UlgdKmrAK/YqH7arSssb33Xf45sVHDpUVA454DXeBrZpi+
zEuo03o5BhAuf38cwfkbV6jN1mC2N0FZfpy4v7RHXKLYr7tr6r+DRn1L1giX5ybx
CgY0fLAXkwsCwUKGKABWxkz9b/beEXaO2rMt+7DBUdpAOP5FNRQ8WLRWBcMGQiaT
S4YcNDaiNkrSP8CMLQP+04hQjahxwCgBnksylciqz3Y5/MreybNnTOrdjVdsF0Oe
t0uL0iWUzV1FfaGldb/oBQLg+e1B74p5+q3aF8YI97qAZpPa1qiQzWIDX8LX9QX
EFyZ3mvqzGrxkFocXleNPgWT8fRuokBlgQQAQIADAUCT64N/QUdABJ1AAAKCRXCX
ELibyletfDOGCACKfQjQISrWUEUrYYZpoBP7DE+YdIIgumt5l6vBrmxt/5OEhr
+dWwuoiyC5tm9CvJbuZup8anWffzTTJmPRPsmE4z7Ek+3CNMVM2wlynsL0t1pRFK
4/5RNjRLbwl6EtoCQfLcZJ//SB56sK4DoFKH28Ok4cplESPnoMqA3QafdSEA/FL
qvZV/iPgtZ7vjqkMgRxAIU4M4fvKe3iXkAEXGxtmgdXHVfOkmHrxJ2DTSvM7/19z
jGJeu2MhIKHyqEmCk6hLjxyCE5pAH59KlBAQOP1bS28xiRskBApm2wN+LOZWzC62
HhEReQ50inCGuuubk0PqUQnyYc+UFXrFpcliQEIBBAbAgAMBQJpV9lVBQMAEnUA
AAoJEJcQuJvK618AZgH/iRFFCi4jqvoqji1fi7yNPZVOMMO2H13Ks+AfcjRtHuV
aa30u50ND7TH+XQe6yerTapLh3aAm/sNP99aTxluwRSlYKeOds93+XVSGRqPBgbF
/vxv0ykok3p6L9Dxw5cL8JrBhMzOJrEkIBfkwN8tWlcXPRFQvcdByv3M3DZTU
qY+UHnOxHvSzsl+LJ0S9Xcd9C5bvYfabmYJvG5eRS3pj1L/y3a6yw6hvY+JtnQAk
t05TdeHMLgQH/z8v9wxDzmE0un8LyoC2Jx5TpkQsJSejwK6b3coxVBIngu6+C
qDAimObZLw6H9xYIK0Fos7j5bQZEWUO7OLBgjcMOqJASIEEAECaAwFAk/Rpc8F
AwASdQAACgkQlxC4m8pXrXw49Qf/TdNbn2htQ+cRWarszOx8BLEiW/x6PvYUQpZ
nV/0qvhKzJlUJm9hQpCA0AsOjhtCN6Cy8KXbk/TvPm9D/Nk6HWwD1PomzrJVfK2
ywGFluTR+HlUKSp7mzm5ym0ws5cPq731m31RUQU8ndJLr9YOf5VL8NqmcOAU
4E8d68BbmVCC5MMr0901FKwKznShfpy7VYN25/BASj8dhmynBYEqRqToOJB6Cnd
JhdTlbfR4SirqAYZg3XeqGhByytEHE1x7FMWWFyhdNtsnAVYBbWqAzBs8lF9Jd
Mhaf0VQU/4z10gVrRtXLR/ixrCi+P4cMfOQkqd6pwqWkaXt6okBlgQQAQIADAUC
T+NxiAUDABJ1AAAKCRXCXELibyletfBFBAC6+0TUDJcNaqOxOG1KVY6KYg9NCL8
pwNK+RKNK/N1V+WGJQH7qDMwRoOn3yogrHax4xleOWiilrvHK006drS1DjsymhR
Sm2Xbe/8pYmEbuJ9vHh3b/FTChmSAO7dDjSKdWD3dvaY8lSsuDDqPdTX8FzOfrXC
M22C/YpG7oUG2A5svE1b+yismP4KmnVNWAEpEuPZcnEMPFgop3haH9gX2+mj/btDB
Yr6p9kAgly17nigtNTNjtI0dMLu43alzedCYHqOINHiB049jkJs54fMGBJf9qPtc
m0k44xyKd1/JXWMDnUmtwKsChAXJS3Y0ciMglx6tqYUtnDrP4l6q1rfriQEIBBAb
AgAMBQJp9T1VBQMAEnUAAAOJEJcQuJvK618J9wIAI1lId9SMbEHF6PKXRe154IE
pap5imMU/IGTj+9ZcXmif8o2PoMMmb3/E1k+EZUaeSBoOmjS8C2gwd5XFWRrWAD
RIKpG5XsL4h5wmN2fj1ororrJXvqH427PLRQK9yzdwG4+9HTBOxjoS8qZT9pky
AJZzAydAMqyseRHgNo0vMwlgR4jo+GcFGQHRf3laUjvVfUPOmlj7afopFdlZml
GaSF0TXBzqcZ1chFv/eTBclulKRvlaDee5FgV7+nLH2nKOARCLVv/+8uDi2zbr83
lp5x2tD3XuUZ0ZwXdoAQWcrLdmGb4lkxbGxvCtsaJHaLXWQ2m760RjUcWVMEBKJ
ASIEEAECaAwFAIAGyWfAwASdQAACgkQlxC4m8pXrXwyVAgAvuvElyuGkniWOLv
uHEusUv/+2GCBg6qV+IEpVtbTCCgijYR5GasSp1gpZ5r4BocOlbGdjJGHTpyK8
xD1i+6qZwUyHNRg2POXUVzceNl2hhouwPLOifcmTwAKU76TEV3L5STvil3hWgUR2
yEUZ3Ut0IGVV6uPER9jpR3qd6O3PouFkwf+NaGTye4jioLAy3aYwtZCUXzvYmNLP
90K4y+5yauZteLmNeq26miKC/NQu4snNFCIPbGRjHD1ex9KDiAMttOgN4WEq7srT
rYgtT531WY4deHpnGoPIHPuAfC0H+S6YWuMbgfcb6dV+Rrd8lj6zM3B/PcjmsYUf
OPdPtIkBlgQQAQIADAUCBgtfQUdABJ1AAAKCRXCXELibyletfAm3CACQlw21Lfeq
d8RmlITsfnFG/sfM3MwZcVfEAtsY3fTK9NiyU0B3yX0PU3ei37qEW+50BzqiStf
5VhNvLfbZR+yPou7o2MAP31mq3Uc6grpTV64BRikCmRWg40WmJNl1hv7AN/0atgj
ATYQXgnEw7mfF0XZTMTD6cmrz/A9nTPVgZDzxpOMgCCC1ZK4Vp9qKdCYUaHpX
3sqnDf+gpVIHkTCMgWLYQOeX5NI+fgnq6JppaQ3ySZRUdr+uFU0s0uvDRvl/cn+ur
ri92wdDnczjFumKvz/clJAg5TG2Jv1Jx3wecALsVqQ3g7f7v1OMaghl5FEbQND
29L9cZe/ZmkriQEIBBIBCgAMBQJVoNxyBYMHhh+AAAOJEEoz7NUmyPxLD1EH/2eh
7+4+8A1IPLy2L9xcNt2bilfFP2pEjG6ulBoMKpHvuTCgtX6ZPdHpM7uUOje/F1
CCNOIPB533U1NI0WIKndwNUJjughToRM+caMUdYyc4kQm29Se6hMPDfyswXE5Bwe
PmoOm4xWPVOH/cvN04zyluxdlQZnQF/nJg6PMsz4w5z+K6NGGm24NEPcc72iv+6R
Uc/ry/7v5cV4uH05+104mmNV5yLecQF13cHy2JlNgIHXPslxTZbeJX7qxxE7TQh
5nviSPgdk89oB5jFSx4g1efXiwTLIP7lbDlxHduomyQuH9yqmPZMbkJt9uZDc8Zz
MYSdDwlcZjFle5bGKfjJAhwEEAECAAYFAISanFIACgkQzdzHm25lclLdvg//oAEP
qdN5VTkWEoDFjDS4i6t8+0KzdDWDacVfWkJ8RAo1M2SkidXnlvzysZd2VHp5Pq7
i4LYCZo5IDkertQ6LwaQxc4X6myKY4LTA652ObFqsSfgh9kVw+aJBBAYeahPQ8CDD
+Y123+MY5wTsj4qt7KffNzy78vLbYnVnVRQ3/CboVix0SRzgo130i7n3B0lihwXy
5goy9ikjzZevejMEffjeRCgory9j5RvHH9PF3JvUtHCS4f+kkLmbQJ1XqNDVD

hlFzjz8oUzzl8YXy3im5MY7Zuq4P4wWil7rkIFMjTYSpz/evxkVlkR74qOngT2pY
VHLyJkqwh56i0aXcjMZiuu2cymUt2LB9IsaMyWBNJjXr2doRGMafjuR5ZaitmML
y2Wix9mVWk7tkwllxmT/IW6Np0qMhDZcWYqPRpf7+MqY3YMK4552b8aDMjXrnO
Owl.sz+UI4bZa1r9dguLWlt2C2b5C1RQ9AsQBPwg7h5P+HhRuFAuDKK+vgV8FRuzR
JeKkFqwB4y0Nv7BzKbFKmP+V+/krRv+/Dyz9Bz/jyAQgw02u1tPupH9BGHlRyLuN
yCJFTSNj7G+OLU0/4XNph5OOC7sy+AMZcsl/gst/TXCizRcCuApNTPDaenACpbv
g8OolzmNWhh4LXbAUHCKmY/hEw9PvTZA1xKHgyJAhwEEGCAAYFAIJYsKQACgkQ
oirk60MpxUV2XQ/!b2/uvThkkbeOegusDC4AZfnL/V3mgk4iYy4AC9hum0R0rNI
XDR51P1TEw9mC1btHj+7m7lq1a5ke5wIC7ENZiir0yPqeWgL5+LC98dzL85hqA
wloGeOfMhrlaVbAZEj4yQTAJDA35vZHVsqmp87i0m+fxZ04OBLXZBw86EoAAZ7Q
EoH4qfCt9k1T363tNnIm3mEvkQ5WjE1R9uchJa1g7hdINQVkjFmPzrJK9f14z5
6Dto89Po4Sge48jDH0pias4HATYHsxW819nz5jZzGcxLnFRRR5iITVZi9qzsHP7N
bUh3qxuWCHS9xziXpOcSZY848xXw63Y5jDJfzupzu/KHj6CzXYJUEeqp9MluoGb
/BCCPEzd20ovyxfuM/BRcc6DvE6sTDF/UES21ROqfuwtJ6qJYWX+IBlgyCjv4o
RdbzxUleePuzqCzmrwIXtoOKW0Rlj4SCeF9yCwUMBTGW5/nCLmN4dww1KW2RP2Eg
4ERbuUy7QnwrP5UCI+0ISZJyYUISf98fmPidQsetUK9Cj+Q5jP2GXwELXWnIK6h
K/6jXp+ESEXSDqIE53vAfe7LwfhIP/D5M71D2h62sdIOm3m7xMOnM5tKIBiV+
4jJSMriCT62zo710+6ilGqmUUyEiI6Ppvo8yuanXkYRCFJpSSP7VP0bBqIzGQT
EQIAJgUCtnc9dglblwUJEPzpwYLCQgHAWlEFQIIAWQWAgMBAh4BAheAAoAJELxx
jTtQcuH1U4AolKjhd70899d+7JfQ3LD7zeeyl0AJ9Z+YyE1HZSznYi73brScil
blV6sbQ7TtXITUUwvgUfJ2FznZSBzaWduaW5nlGtleSAod3d3Lm15c3FsLmNvbSkg
PGJ1aWxkQG15c3FsLmNvbT6lBwQwEQIALwUCtnc9rSgdIGJ1aWxkQG15c3FsLmNv
bSB3aWxkIHN0b3Agd29ya2luZyZb29uAAoJELxxjTtQcuH1tT0An3EMrSjEkUv2
9OX05JkLiVfQr0DPAJAwKtL1ycnLPv15pGmVszav8JyWn3IhBBMRAGAdBQJHrJS0
BQkNMfioBqsHCgMEAxUDAgMwAgECF4AAEgkQjHGNO1By4fUHZUdQRwABAa6SAJ9/
PgZQSPNeQ6LvvZCALeBJOBt7QCffgs+vWP18JutdZc7XiawgAN9vmmiTAQTEQIA
DAUCPj6j0QWDCWYUawAKCRBJUOEqsNKR8iThAJ9ZsR4o37dNgyI77nEqP6RAIJqa
YgCeNTPTEVY+VXHR/jfyfo0bVurRxT2ITAQTEQIADAUCPKCAwWDCWliiQACRC2
9c1NxrokP5aRAKCIaegaMyiPKenmmm8xeTJSR+fkQCgrv0TqHyvCRINmi6LPucx
GKwfy7KIRgQQEQIABgUCP6zjrwAKCRcvxSNleINOD/aWAKDBUieGwwAFNhn2n8gGJ
Sw8IAulSgCDHMZLAS26NDP8T2iejsfUOR5sNriIrgQQEQIABgUCP7RDdwAKCRF
lq+rMhNOZsbDAJ0WoPV+TWILtZG3wYqg5LuHM03faQCeKuVvCmdPtro06xDzeeTX
VrZ14+GIRgQQEQIABgUCQ1uz6gAKCRCL2C5vMLLXh90AJ0QsqhdAqTak3SBnO2w
zuSOWiDIUwCdFExsdDtXf1cL3Q4ilo+OTdrTW2CIRgQTEQIABgUCRPEzJgAKCRD2
ScT0YJNTDAPxAKCJtQT9LCHFyfwKNGGBgKja0zi9wCcCG3MvnbZDUUqDVeubUZ
61Sont+ITAQEQIADAUCQYHLAQWDBiLZiwAKCRAYWdAfZ3uh7EKNAJwPywk0Nz+Z
Lybw4YNQ7H1UxZycaQCePvH4Y4P5CHGjeYj9Sx2gQCE2SNx+ITAQEQIADAUCQYHL
NAWDBiLZWAACKRCRBwfr4hO2kijAJ0VU1VQHzF7yYVeg+bh31nng900kwCeJl8D
9mx8neg4wspqvgXRA8+t2saITAQEQIADAUCQYHLyGwDBiLZKgAKCRBrcOzZxcP0
cwmqAJsFjOvY9c5eA/zyMrOZ1uPB6pd4QCdGyzgbYb/eaPu6FMvV9PvleNZRel
TAQEQIADAUCQdCTJAWDBdQraAAKCRB9JcoKwSmnmwJVAKCG9a+Q+qjCzDzDtZKx
5NzDW1+W+QcEL68seX8OoiXLQuRlflmPmRv2m9+ITAQEQIADAUCQitbugWDBXl
0gAKCRDM6G5JFue5q/MTAKCTMvICQtLKizD0sYdwVLHXJrUvgCffmdeS6aDpwln
U0/yvYjg1xiYuiqITAQEQIADAUCQCPzOgWDB3pLUgAKCRA8oR80IPr4YSZcAJwP
4DncDk4YzvDvnRbXW6SriJn1yQCdEy+d0CqfhdM7HGUs+PZQ9mJKBKqITAQSEQIA
DAUCQD36gWDB2ap0gAKCRDy11xj45xlnLLfAKC0NzCVqrbTDRw25cUss14RwRoJ
PACeLpEc3zSahJUB0NNGTNIpwiTczCITAQSEQIADAUCQQA4KhAWDBpaaCAAKCRA5
yiv0PWqKX/zdAJ4hNn3AjitcAymLrLhIZQvib551mwCgw6FEhGLjZ+as0W681luc
wZ6PzW+ITAQSEQIADAUCQoCINAWDBSP/WAAKCRaedcCfFI0OmKJwPUDhS1eTz
gnXcdKgf353LbjvXgCeLCWyyj/2d0gIk6SqaPI2UcWrrqilTAQTEQIADAUCPK1N
hAWDCVdXCAAKCRAtu3a/rdTJMwUMAKKVPk1Up/kyPrIsVKU/Nv3bOTZACfW5za
HX38jDCuxsjlr/084n4kw/ulTAQTEQIADAUCQdeAdgWDBc0kFgAKCRBm79vIzYL9
Pj+8AJ9d7rGJlchZTCSYVnaStv6jP+AEACeNH5yiltqieRBCcLcaccqYK81oml
TAQTEQIADAUCQhIBDgWDBYwjfgAKCRB2wQMcojFuoaDuAJ9CLYdysef7IsW42UfW
hl6HjxkzSgCfePXS4hEmmGicdrPiJQ/W21aB0GIZQQTEQIAHQULBwoDBAMVawID
FglBAheABQJLc/KBQkQ8/OnABIHZUdQRwABAQKQjHGNO1By4fWw2wCeJilgEarL
8eEYfDdYTRdqE45HkoAnjFSZY8Zg/iXeErHI0r04BRukNVgiHsEMBEcADsFAkJ3
NfU0HQBP3BzL4iulHNob3VsZCBoYXZIIJZw4gbG9jYwWhlEknBSAqc28qIHNO
dXBpZC4uLgAKCRA5yiv0PWqKX+9HAJ0WjTx/rqgouK4QCROV/2IOU+jMQQCfYSC8
JgslleN8aiyuStTyrk0VWClijwQwEQIATwUCRW8Av0gdAFNob3VsZCBoYXZIIJGJ
ZW4gYSBsb2NhbCBzaWduYXR1cmUsIG9yIHVnbVW0aGluZyAtIFdURiB3YXMgSSB0
aGlu42luZz8ACgkQcor9D1qil+g+wCfcFWoo5qUl4XTE9K8tH3Q+XGWeYYAnji
KxjtOXc0ls+BlqXbfZ9uqBsiQliBBABAgAMBQJBGcuFbYMGltkHAAoJEKj75s5m
oURoqC8QAIISudocbJRhTAROOPOmsRreyp46Jdp3l1oFDGcPfkZSBWWh8L+cJh
dyclwWSeZ1D2h9S5t4EnoE0khsS6wBpuAuih5s//coRqllLKEdhTmNqulCH5m
imCzc5zXWZDw0hPL2InGsZMuh2QCwAkB4RTBM+r18cUXMLV4YHKyjIvADhsIPP/
MKUj6rJNsUDmDq1GiJdOjySjtCFjYADIQYSD7zcd1vpqQLThnZBESvEoCqumEFOP
xemNU6xABOCL+pUpB40PE6Un6Krr5h6yZxYZ/N5vzt0Y3B5UUMkgYDSpbulNvaU
TFiOxEU3gJvXc1+h0BsxM7FwBZnuMA8LEA+UdQb76YcyuFBcRohmcEUtitudLu84
E2BZ2NSBdymRQKSinhRxsEWIH6Txm1gtJLynYsvPi4B4JxKbb+awnFPusL8W+gFz
jbygeKdyqYgKj3M79h3geaY7Q75Kx1UogioKcb15Vzg47OQCWeeERneRjeAdx
EQiwGA/ARHvOP/1l0LQA7j2P1xTtrBqqC2ufDB+v+jhXaCXstKSW11Tbv/b0d6
454UaOUV7RisN39PE2zFvJvY7bwfiwUJvMl4rWJAEOLIDtRt2h8JahDObm
3CWkpadjw57S5v1c/mw+xV9yTgVx5YUfC/788L1HNKXfeVDq8zbAiQliBBMBAgAM
BQJcnwocBYMFBZpwAAoJENjCCglajFFIPIT4P/25zvP8ixqV85igs3rRqMBtBsj+

5EoEW6DjnlGhoi26y1nasC2frVasWG7i4JlM0U3WfLZERGDJR/nqIOCEqsP5gS3
43N7r4UpDkBsYh0WxH/ZtST5lIFK3zd7XgtxvqKL98/OSgijH2W2Sj9DgpjtO+T
iegg7igtJzw7Vax9z/LQH2xhRQKZR9yernwMSYaj72i9SyWbK3k0+e95fGnlR5pF
zIGq320rYHgD7v9yoQ2t1klsAxK6e3b7Z+RiJG6cAU8o8F0kGxjWzF4v8D1op7S+
loRdB0Bap01koOKLyt3+g4/33/2UxsW50BtfqcyNJVu4bZns1YsAgQDOOanBhg8
lp5XPIDxH6J/3997n5JNj/nk5ojfd8nYfe/5TjflWNjput6tZ7frEki1wl6pTNbv
V9C1eLUJMSXfdZyHtUXmiP9DKNpsucCUEBKWRKLqnsHLkLYydsleUJ8+ciKc+EWh
FxEY+MI72cXAaz5BuW9L8KHnzZZfez/ZJabiARQpFfjOwAnmhZj9r++TEKRLrEr96
taUI9/8nVPv6LbnBpcM38T6dJ639YvuH3ilAqmPPw50YvgllEe4BUyD5r52Seq
8XQowouG0uBX4vs7zgWfUyA/s9ebfGalw+uJd/56Xl9l6q5CghqB/yt1EceFEnF
CAjQc2SeRo6qzx22IEEYEEBECAAYFAkSAbycACgkQCQyYeUxD5vWdCAcFQsVk/XGI
ITFyFVQ3IR/3Wt7zqBMAoNhsolcX8VUfs2BzxPvVGS3y+5Q9IEEYEEBECAAYFAkUw
ntcACgkQOI4l6LNBlykyFgCbCw5glii0RTDJsDniuJDcu/NPqEAniSq9iTaLjgF
HzbaizU8arsVCB5IEEYEEBECAAYFAkWho2sACgkQu9u2hBuwKr6bjwCfa7ZK6O+X
mT08Ssysg4DEoZnK4L9UAoLWgHuYg35wbZYx+ZUTh98diGU/miFOEEExECAB0FAj4+
owwFCQlImAYAFcWcKAwQDFQMCAxYCAQIXgAAKCRCMcY07UHLh9XGOAJ4pVME15/DG
rUDohtGv2z8a7yv4AgCeKlp0jWUWE525QocBwms7ezxd6slyXQQTEQIAHQUR6yU
zWUJDTBYqAULBwoDBAMVAwIDFgIBaheAAoJElxjtTqCuH1dCoAoLC6RtsD9K3N
7NOxcp3PYOzH2oqzAKCFHn0jSxk7E8by3sh+Ay8yVv0BYhdBBMRAGAdBQsHCgME
AxUDAgMWAqECF4AFakequSEFCQ0ufRUACgkQjHGNO1By4fUdtwCfRNcueXikBMy7
tE2BbfwEyTLBTFAAniQGBkmcARV57nqauGhe1ED/vdgiFOEEExECAB0FCwckAwQD
FQMCAXYCAQIXgAUCS3AuZQUJEPpyWQAKCRCMcY07UHLh9aA+AKCHDKOBKBrG8tO
g9Blub3LfnMvHQCeIOot1hHHUlsTIXAUrD8+ubleZaJARwEEgECAYFAkVcigMA
CgkQ3PTTrHsNvDi8eQgf/dS0R9Klozz8ik79w00N0sdojY0Na0NTfmTcHg30XJo
G62cXYgc3+Tjnd+pYhi5gyBixF/L8k/kPVPzX9W0YfwChZDsftw0iDvmGxOswiN
jzSo0lhWq86/nEL30KH9AHC1XFNRw8WZYq9Z1qUXHHJ2rDARaedvpKH0jzRY0N
dx6R2zNyHDx2mIfCQ9wDchWeUJdAv0uHrQ0HV9+qx7IW/Q3L/V5AuU0tiowyABl
PPYrB6x9vt2Eg9X8BGC8szfHozVEwkzV96iyHbpddeAOAeA40VPW1MMFCmlHxi2s9/N
JrSrTPVfQOH5f99h7Hbpq/ETw0loX1FKo7vndMnHZNfEnl+PDXLcdMYQgijYzhT
xER4vY0UKU8ekSshUy4zOX7XSJxwqPUvps8qs/Tvojlf+vDjvgFYHvkgvS+shp8
Oh/exg9vKETBlgU87Jsgsn/SN2LrR/Jhl0aLd0G0iQ+u/HmVvYdQUMFaCzWk/BKN
XPzmGZEUJZ3RfNbYa19Mo7hcE3js76nh5YmXfVxbTggVu4kdFkiQEiBBABAgAMBQJ
M06IBQMAEnUAAAOJcJcQuJvKV618F4gH/innejlHffGMk8jYix4ZZT7pW6Apyol+
N9ly85H4L+8rVQrtCTHyq0VkcN3wPSwtfZszUF/0qP6P8sLJNJ1BtrHxLORYjJpm
gveeyHPzA2oJl6imqWUTiW822fjY/azwhvZFzxmVbFJ+r5N/Z57+la4t9LTSqTN
HzMUyAXKDaAqzZeK7P0E6XUaaeygbjWjBLQ1O0ezozAy+Kk/gXApMDCGfUHSFeZ
mgtfcbXLM2XFQpMUooETD2R8MUsd+xnQsff/k6pQOLxi+jUESwSr/qmVlk6gZ4D
pemBjuhcxYlxJYUaXZmn5s+off4GFxRqXoY7I9Z+tcM9AX37Im6S+JASIEEAEC
AAwFAkpEcgofAwASdQAACgkQlxC4m8pXrXz2mgf/RQkpmMM+5r8znx2TpRAGHi5w
ktvdFxlVpaOBWE28NDwTrpcoMqo9kzAiuEQjVNihbP21wR3kvnQ84rTAH0mLC2l
uyybggqpwzOUH+Wi0o+vk8ZA0A0dStWRN8uqneCsd1XnqDe1rvqC4/9yY223tLmA
kPvz54ka2vX9GdJ3kxMWewhrVQSLCktQpygU0dujGTDqJtnk0WcBhVf9T87lv3W2
eGdPielzHU58trXezmGfj21d56G5ZFK8co7RrTt4qdznt80gh1BTGmhLzjMPLTe
dcMusm3D1QB9lTogcG94ghSf9tEKmmRJ6OnnWM5K9KcL63E5oj2/IY9H54wSYk
lgQQAQIADAUCSIY+RwUDABJ1AAAKCRCXELibyletfoOQB/0dyJBiBjgf+8d3yNID
pDKtLhZyw8crjPBVdOgX12xalUYBTGcQITRVHSGgzffD5BQXeuUuWhpl4QB0u21c
EPPwSMiWiXbtwF5q6RVf3PZGJ9fmFuTkPRO7SruZeVDo9WP8HjbQtOLukYf566e
grzAYR9p74UgWfptDmrqrRTobiuvsFBxosbeRCvEQCRn0n+p5D9hCVB88tUPHnO
WA4mlduAFZDxQWTApKQ92frHiBqy+M1JFezz2OM3fYN+Dqo/Cb7ZwOAA2dbwS7o
y4sXEHbWJonjsgpQwFYB23tsFuuM4uZwVEbjg+bveglDsStbDlfgArXSL0+ak
IFcHiQEiBBABAgAMBQJKAaQEBQMAEnUAAAOJcJcQuJvKV618rH0H/iCciD4U6YZN
JBj0GN7/Xt851t9FWocmcaC+qtuXnkFhplXkxZVOCU4VBMs4GBBoqflvagBTyFV4
Di+W8Uxr+1ju3l/HvofXwdwNkGG6zNBhWSjdwGpGwPvh5ryV1OfLX/mgQgdDmx
vqz5+kFDUJ4m7uLaeuU2j1T0IR4zU0yAsbt7J3hwfJqCXHOc9bm5nvJwMrSm+sdC
TP5HjUlWHR9mTe8xuZvj6sO/w0P4AqIMxjC9W7pT9q0ofG2KSTwt7Fwb05sbG4U
QYOJe4+Soh3+KjAa1c0cvmIh4cKX9qfCWwhhdeNfh1A9VTHhnl5zTv/UjvnQtjhl
H/Fq1eBSKcSASIEEAECAAwFAkp5LgoFAwASdQAACgkQlxC4m8pXrXwY6wgAg3f8
76L3qDZTYIFAWs3pXB18GsUr1DEkTIEDZMKDM3wPmhaWBR1hMA3y6p3aaCUyJJ
BeneXzgyU9uqCxCpC78d5qc3xs/Jd/SswzNYuvuzLYow5wN5L31SLmQTQ8KqE0uo
RynBmtDCQ4M2UKifSnn+0+3mPh85LVAS481GNpL+VvFYtKesWNU40+98Yg6L9NG
WwR1fsQbcokZoa44Jz7Y7f81ObC4r/X1DgPj2+d4AU/plzDcdribNOypr+7340e
cnaGO4Lsgd19b1CvqgJlgrQuu3kRvd+Ero2RYpDv6GVK8Ea0Lto4+b/Ae8cLXAh
QnaWQCEWmw+AU4Jbz4kBlgQQAQIADAUCSo5fvQUADABJ1AAAKCRCXELibyletfoA08
B/9w8yJdc8k+k07U30wR/RUg3Yb2lBDygmY091mVsyB0RGixBDXEPoxBqGKAXiV1
QSMAXM2VKRsuKahY2HFkPbyhZtjbdTa7Pr/bSnPvRhAh9GNWvRg2Kp3qXDdVj9x
ywEghKvxcEIVXtNRpbqRoKmhZlExvUQck5DM1VwFReeYloxxg4035WADHVMdngQ
S2t8P2WaU/p8EzhFGg6X8KtOID68zGboaJe0hj2VdC+Jc+KdjRfE3fW5toid/o
DKJaW6lB3WkXb0g6D/2hrEJbX3headChHKSB8eQdOR9bcJdHhU8csd501qmrhC
ctmvIpeWQZdlQdk6sABPWeeCiQEiBBABAgAMBQJKoBjHQBMAEnUAAAOJcJcQuJvK
V618Ml8H/1D88/g/p9fSvor4Wu5WlImbg8zEAik3BlXOruEFWda6nART6M9E7e+P1
++UHZsWys6l9R0pWxRLG1Yy9jLec2Y3nUtb20m65p+IVeKR2a9PHW35WZDV90YP
GZabKkO1clLeWLvgp9LRjZ+AErg+IjHqsULXro1dwewLTB/gg9l2vgNv6dKxyKak

nM/GrqZLATAq2KoaE/u/6lzfRFZlZnLtzH8X7+nS+V8v9liY4ntrpkrbvFk30U6
WJp79oBIWwnW/84RbxutRoEwSar/TLwVRkcZyRXeJTApbnLGNQ/IDO1o1d7+Vbjd
q/Sg/cKHf7NthCwkQNsCnHL0f51gZCJASIEEAECaAwFAkqoEAAFawASdQAACgkK
lxC4m8pXrXwE/AfiXD4R/A5R6lr/nCvKwCTKJmalajssuAclEa2pMnFZY0/8rzLO
+Gp8p0qFH9C4LfwA0NvR5q6X/swuROf4zxjSvNcdlQVafJZDDEgJ5GXzsPplrv
SAI9jS3LL7fSWDZgKuUe0a4qx7A0NgyGMUYGhP+QIRFa8vWEBI9fAnd/0mMqAeBV
qQyOH0X1fW1Ca2Jn4NKfuMy9GEvRddVlb1LvoNvtXPNzeeKMyN9Jdx1MFwssy
COBP2DayJKTmjvqPEc/YOjOowoN5sJ/jn4mVSTvvtOoLiReSs6GSCAJMVXN7eYS
/Oyq6lu1JdCJvmb8N2WixAZtAvGf8OA7CWXKvYkBlgQQAQIADAUCSrnHiQUADABJ1
AAAKCRCXELibyletFPChB/9uECTi1dZeNuFsd0/RuGyRUvIrrhJE6WccOrL09par
rPbewbKBmjSzB0MygJGvcC06mPNUquJ7WpxKsFmfg4vJBPIADFKtRUY9BLzjC
eotWchPHFBVW9ftPbaQvISuU7d89NLjDDM5xrh80puDIApoxQLDolrh3T1kpZx56
jSWv0geIFUMbXAzmqkJSyL4Xdh1aqzgUbREd7Xf2ICzuh0sV6V7c/AwWtjWEGEsA
HZaiQDyWzWbC18GwRMLIAzGwB/AScFDQRCKZJdJL+Ql8Yt6z+ZMvR8g7CIU5PKY
dhiif2UVTQwLAoW7INRCQAQcGjK3IMlz7SO/yk4HmVUIQEiBBABAgAMBQJK3gjG
BQMAEnUAAAOJEJcQuJvKV618jkEH+wb0Zv9z7xQgpLMowVufBQVU8/z7P5ASumyB
PUO3+0JVxSHBhICKKQ7n11m1fhuGt2fCxXhSU6LzXj36rsKRY0531GZ9QhvqFUtQH
3Xb21QLIJC4UKJG2JSSCdcuA/x98bwp2v7O03rn7ndCS16CwXnRV3geQoNipRKMS
DajKpPzV1RiZm8pMKqEb8WSw352xWoOcxuffjlsOEwvJ85SEGAZ9tmllkZoc7Ai
QONDvii9b8AYhQ60RIQC0HP2ASSmK0V92VeFPxHmAygdDQgZNVtbVxgnnt7oTNEu
VRXNY+z4OFBArp7R+cTsvijDRZY4kML1n22hUybwoxUEvjZV2+JASIEEAECaAwF
AkrvOIQFAwASdQAACgkQlxC4m8pXrXrPAgArXiNgZirNuBhNcXlkzkCHLx5wnV
e4SmTpbWzTwWw7+qk7d4I9hIwtdlmsORINzo7f4ShSUzJX2GciNaXhaHRo7+y5O
Zbu82jQB09aQkKQ/nibKkYCuqxqUrobTE+m+DuYz3JUQZm2PsPcHLS8mX9cxvrJUncP
nXEV0DRaq71SGWDprtqvBbp6i38aY3slhYgz8wM5m1szKDijywmBYcfEhldtoztz
hm7wZshzRWQX1+Rf/plsnk+OzBla34crSemTnacv/B7278z2XAYziPNFuqz0xu+
iltOmYmayfNWAmumuW9NcuwWmlth6Mc2HLrpo0ZBheJ6iuDMPsHnwqdB/4kBlgQQ
AQIADAUCSwd2GUDABJ1AAAKCRCXELibyletFP6tB/4m1w0BtlkJgtS6E+B/ns14
z4A4PGors+n+MYm05qzvi+EnDF/sytCmVcKeimrtvDcofDKAFFvJcYXfnJdGwM
Pu0SJMRL5KKCirkAwZmU/saxOgoB5QLNw+DHPteJ3w9GmWIGxlqG1r15WC5dudzBC
y3FsnjYUg3jaLHOO9yXb5h0kUTORfUKdvrA1gxF2KoaTzWqGoaPPnHoqb88rjt
zk817gDqoXnz8wLxa0ZYvTC/McxdtWtrwXlft+krmmQ18ilZENE2hvVLNJVuluU
oiWLeHA8INCQ4W4WTdLc1mCnCjGTMX/MN41uLH0C9Ka4R6wEaqj4IPdk1B/1TV+Q
iQEiBBABAgAMBQJLEYGrBQMAEnUAAAOJEJcQuJvKV618naiH/2t9aH5mBTKBN6fU
qhrf79vlsjt/QNS5qisBISZMX3/1/0Gu6WnXkPsfDCUJMWcJmCnV7KJ2UwxTHHG
VpAst9r2afUNxRyqZwzwytkuZok0XngAEDYDDBS3ssu2R4uWLCsC2ysXEqO/5
t5YrTWJZrfeiphTApY5hxrMujvqy3kEwKKbImz91cDeilS+YCBcaj5n/1dMYf7
8U8C6ieurxAg/L8h6x25VM4lX4MmG2T8QGtKkUXd+Fd/KYWmf0LE5LLPknf0Hhw
oVsIPXeinp4F5HK/5wzviv4YZpzuTqs9NlKcMsa4luuPOB0FDf0pn+OFQbEg9QwY
2gCozK+JASIEEAECaAwFAksjTdqFAwASdQAACgkQlxC4m8pXrXwlogf/XBGBXRVX
LMArN4Szc0JwT3/tUCriTkb3+vzKJRG90zFhYAccjn7w+7JkQicq6quQG1EH2X4
/SU6ps1DLqGHHiJW3ZhxQScLZmhAYsh2qG4GP/UVW3QJX7c61t+H30lvWg2cr
wqCxxFZAgkAAkr9xcHWFZJEQeXoob6cCZObaUnHSANdmC6s5IUxXYa2bml7Q3UB4
4KczDvAfbPZKJOW9k0qb3lc11zx+vGdyZFbm4R0+3LPp/vT0b3GiSbbF9IU1GOXh
VaphrgFFa76dmjfhCkPpIXAkK1VSIU/aPGAefduTFMdlSZpdMtJ5AULjGcszBDIR
pLIPxvqVaoZpgIkBlgQQAQIADAUCSycmkgUDABJ1AAAKCRCXELibyletFHINCAcP
1YespiHfQt2alcscE5zgfETEHHic8Ai6pNkU9HT4TeWcFHEDe5QqfYcpjLrQvBXS
k5vXeitbyRdv+e+j5Z+HyHjiG8nAQBL6qy9eHqQE4+d7gYs6DTk7sG9ZMYphREb
ltzD+F4hVQCqdLT8LNR0eVFN7ehqECScDaCG8/Qtyi+I0M902/Yn+mz0iOilUdWJ
9x6LPAIlnb1gsYDEyLjwGIZml0r5Kh9wYoV4vnNezFbx01uRiW0B7iaPjIEsbt
OOkp7wx2aX+DM3N9F3BtalY8XnzcnomNm83SNsgmgrZljpQlUnNqIHNM8DupQ+I
Wov5gtl6pTC7CgeVTvYRiQEiBBABAgAMBQJLOGXuBQMAEnUAAAOJEJcQuJvKV618
ll4IAKJ9mm4j0c8fe9+uDl8eCJRbzNbVXm8zWzpA8GUtQAakwxoK332QP1Wa1P
odni/e3EMhsSREOZJv79YqGxGRBTE9Kb/VjM34nas4XSnXKW28XWhKylw+XwQAI
nY2swFHh+83Htr/mwTdJfS2aEYI2zboBvd/JZCdhOGU2GH737S/3uEczokKfVQ/w
OTM8X1xWwlyWqz23k/DsGcuDs9IA2g7Mx7DSqBtVjaTkn9h0zAtzXLDkmp4SAUVj
cZ83WdpFre5WnizZjdXIBMM5OCexp5WpmzyHLTnaBFK4jEmnSk5C2Rnoyp8lvz6g
Ecq1tRbEXijRw++d2TFYJwLkTiJASIEEAECaAwFAktKMicFAwASdQAACgkQlxC4
m8pXrXxqHQGuYY5scKrh0m/GS9EYnyC9494IOi06iyU0CpE6oBC31M3hfX/Dbj
UbcS5szZNU+2CPY04ujQLZ7suN7+tJG6pZFfMevajT9+jsL+NPMF8RLdLOVYmbI
TmSQGNO+XGEYaKYH5oZleIW5AKCgi2ozkdFIBBLAx7Kqo/FyybhkURFEcvEyVmfg
3KLv7lIix/YLfoCMCJ/Lcm9/llSFB1n8Nvg66Xd533DKoHjueD3jyaNAVlo2mq/
slAv++kntvOib3GDK5pFwHZ78WwiCpsWZpE5gzAnzJ1Y0WEigRo0PVLu3clO0jLG
23d+H/CbfZ8rkajHjeCDQF7YVmP0t0nYpYkBlgQQAQIADAUCS1v+ZgUDABJ1AAAK
CRCXELibyletFNS/CACq2TtkB86mjQM+cJ74+dWBvJ2aFuURuzxm95i9Q/W/hU08
2iMbC3+0k2oD8CrTOe61P+3oRyLjv/UEDUNzLncNe2YsA9JeV+4hvpW5Vp3Om13
089fCKZUobqslXNkHwYU+zAaZJXEUgRmRz0HbQleAMOWF4oa226uo1e4ws1Jhc+
F3E/APcrYfBqBUdL05hapQLditYpsBjldiBGpjzidMLE2wX2W4ZpAdN0U6BlylqR
mTPjbSkvzS9kSWFmfhQgnBDKEYJpVZgE1sN52rYC1sDeGeiukXlzjVov9MMhYMWa
Zo3R5o3F2iIM/BK6FbC252lf/Mhu3ICuXujNBZNYiQEiBBABAgAMBQJLbSH4BQMA
EnUAAAOJEJcQuJvKV618kd0IAJLLwDh6gvgAIBfKlQJXqQxUdcSOOVMAWtlHGWOy
ozjgomZZBkRL8dtCDr9YBMcj5czcQ3qpmLJdppXhKB+kjV2iUXfDMsfXwJ4wLfs
8FNnXw8H5U01oBkGH/Ku6ngL9Vwt+MjYHtCkww9QueUKZnDudX9qlzLAI+mwSTu
A6+fy4VWl6g40AA0v3exaQM55YR/UhlKunpGG9o8Qkq77dMEbTmPomBoLbOMRB3Dd
MAvVU6G2l6Pcb7KobVCuObN6batXARV/G8sw+nfzJ16fir/KobZT2A6m+Jrjq4dl

```
F14jlLbz16O5JGUPArYn2G2ddBdSAy7dtFSVhWWiWC9n88q5Ag0EPj6jHRAIAO/h
iX8WzHWOMLJT54x/axeDdq1rBDf5cWmaCWHN2ujNNlpx5emoU9v7QStsNUCOGB
bXkeO4Ar7YG+jtSR33zqNh3y5kQ0YkY3dQ0wh6nsl+wh4XIY/3TUZVtmdJeUBRH
JlfVNFYad2hX1guFI37Ny1PoZAFsxO82g+XB/Se8r/+sbmVcONdcdeFKrE3FjLt
ljNQcx6I9Q2Oy8KDxG/zvUZG3+H5i3tdRMYGmuD6gEV0GXOHYUopzLeit1+Aa0
bCk36Mwbu+BeOw/CJW3+b0mB27hOaf9aCA855IP6fJFvtxcblq8nHlqhU3Dc9tec
sl9/S1xZ5S8ylG/xRsAAwUH/i8KqmvAhq0X7DgCcYputwh37cuZiHOa1Ep07JRm
BCDgkdQXkGrsj2Wzw7Aw/TGdWWkmm2pxb8BRui5cfcZFO7c6vryi6FpJuLucX975
+eVY50ndWkPXk1HF4i+HJwRqE2zliN/RHMs4LJcwXQvvd43EE3AO6eiVFbD+qA
AdxUFOeLbIKNBHPG7DPG9xL+Ni5rKE+TXShxsB7F0z7ZdJZOG0JODmox7IstQT
GoaU9u41oyZTliXPIFidJoiZCh7fdurP8pn3X+R5HUNXMr7M+ba8ISNxce/F3kmH
0L7rsKqdh9d/aVxhJINJ+inVDnrXWVoXu9GBjT8Nco1iU9SIVAQYEQIADAUCTnc9
7QUJE/sBuAASB2VHUEcAAQEJEIxxjTtQcuH1FJsAmwWK9vmwRj/y9gTnJ8PWf0BV
roUTAKCIYAhZuX2nUNwH4vIEJQHDqYa5yQ==
=ghXk
-----END PGP PUBLIC KEY BLOCK-----
```

ビルド鍵を個人の公開 GPG 鍵リングにインポートするには、`gpg --import` を使用します。たとえば、鍵を `mysql_pubkey.asc` ファイルに保存した場合、インポートコマンドは次のようになります。

```
shell> gpg --import mysql_pubkey.asc
gpg: key 5072E1F5: public key "MySQL Release Engineering
<mysql-build@oss.oracle.com>" imported
gpg: Total number processed: 1
gpg:      imported: 1
gpg: no ultimately trusted keys found
```

公開鍵 ID の `5072E1F5` を使用して、公開鍵サーバーから鍵をダウンロードすることもできます。

```
shell> gpg --recv-keys 5072E1F5
gpg: requesting key 5072E1F5 from hkp server keys.gnupg.net
gpg: key 5072E1F5: "MySQL Release Engineering <mysql-build@oss.oracle.com>"
1 new user ID
gpg: key 5072E1F5: "MySQL Release Engineering <mysql-build@oss.oracle.com>"
53 new signatures
gpg: no ultimately trusted keys found
gpg: Total number processed: 1
gpg:      new user IDs: 1
gpg:      new signatures: 53
```

鍵を RPM 構成にインポートして RPM インストールパッケージを検証する場合、鍵を直接インポートできるはずで
す。

```
shell> rpm --import mysql_pubkey.asc
```

問題が生じた場合、または RPM 固有の情報が必要な場合は、[セクション 2.1.4.4 「RPM を使用した署名確認」](#) を参照してください。

公開ビルド鍵をダウンロードしてインポートしたあと、所望の MySQL パッケージとそれに対応する署名をダウンロードします。これもダウンロードページで入手できます。次の表の例に示すように、署名ファイルの名前は配布ファイルと同じで、拡張子 `.asc` が付いています。

表 2.1 ソースファイルの MySQL パッケージと署名ファイル

ファイルタイプ	ファイル名
配布ファイル	<code>mysql-standard-8.0.29-linux-i686.tar.gz</code>
署名ファイル	<code>mysql-standard-8.0.29-linux-i686.tar.gz.asc</code>

両方のファイルが同じディレクトリにあることを確認してから、次のコマンドを実行して配布ファイルの署名を確認します。

```
shell> gpg --verify package_name.asc
```

ダウンロードしたパッケージが有効な場合は、次のような `Good signature` メッセージが表示されます:

```
shell> gpg --verify mysql-standard-8.0.29-linux-i686.tar.gz.asc
gpg: Signature made Tue 01 Feb 2011 02:38:30 AM CST using DSA key ID 5072E1F5
```



```
gpg: Good signature from "MySQL Release Engineering <mysql-build@oss.oracle.com>"
```

Good signature メッセージは、ファイルの署名が弊社のサイトに記載されている署名と比較して有効であることを示します。しかし、次のような警告が表示される場合もあります。

```
shell> gpg --verify mysql-standard-8.0.29-linux-i686.tar.gz.asc
gpg: Signature made Wed 23 Jan 2013 02:25:45 AM PST using DSA key ID 5072E1F5
gpg: checking the trustdb
gpg: no ultimately trusted keys found
gpg: Good signature from "MySQL Release Engineering <mysql-build@oss.oracle.com>"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:       There is no indication that the signature belongs to the owner.
Primary key fingerprint: A4A9 4068 76FC BD3C 4567 70C8 8C71 8D3B 5072 E1F5
```

セットアップおよび構成に依存するので、これは正常です。これらの警告の説明は次のとおりです。

- gpg: no ultimately trusted keys found: これは、特定の鍵がユーザーまたはユーザーの信頼する Web によって「最終的に信頼されて」いないことを意味します。これは、ファイルの署名を確認する目的に関しては問題ありません。
- WARNING: This key is not certified with a trusted signature! There is no indication that the signature belongs to the owner.: これは、ユーザーの所有している鍵が弊社の本当の公開鍵であることをどの程度信用しているかに言及しています。これは個人的な判断です。理想的なのは MySQL 開発者が鍵を直接手渡すことですが、一般的には鍵はダウンロードされます。ダウンロードは改ざんされていますか？おそらくそうではないでしょう。しかし、その判断はユーザー次第です。信頼網をセットアップするのが、信頼するための 1 つの方法です。

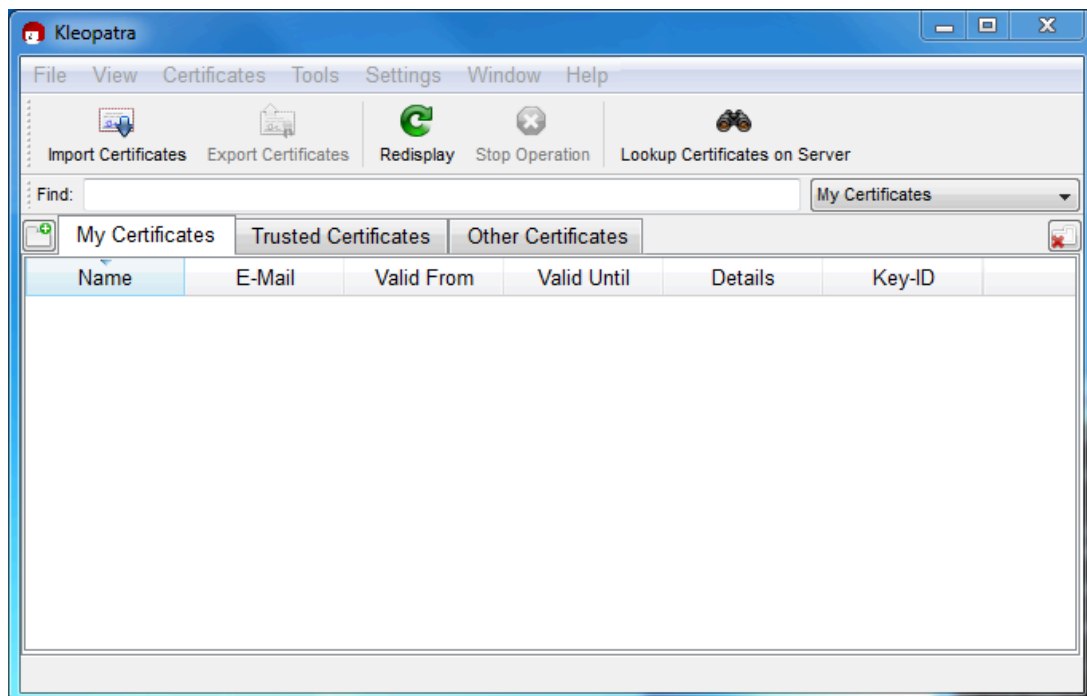
公開鍵を処理する方法の詳細は GPG のドキュメントを参照してください。

2.1.4.3 Gpg4win for Windows を使用した署名確認

セクション2.1.4.2「GnuPG を使用した署名確認」のセクションでは、GPG を使用して MySQL ダウンロードを確認する方法を説明しています。このガイドは Microsoft Windows にも適用されますが、もう 1 つの選択肢は [Gpg4win](#) などの GUI ツールを使用することです。ほかのツールを使用してもかまいませんが、ここでの例は Gpg4win に基づいており、バンドルの [Kleopatra](#) GUI を利用しています。

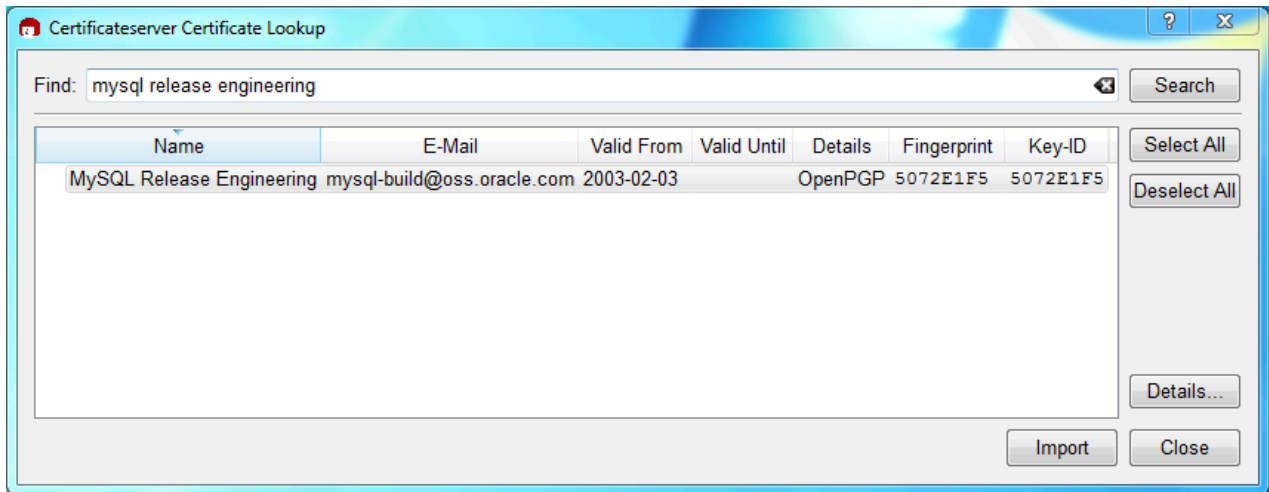
Gpg4win をダウンロードしてインストールしてから Kleopatra をロードします。次のようなダイアログが表示されるはずですが。

図 2.1 Kleopatra: 初期画面



次に、MySQL Release Engineering 証明書を追加します。そのためには、「File」、「Lookup Certificates on Server」をクリックします。検索ボックスに「Mysql Release Engineering」と入力して「Search」をクリックします。

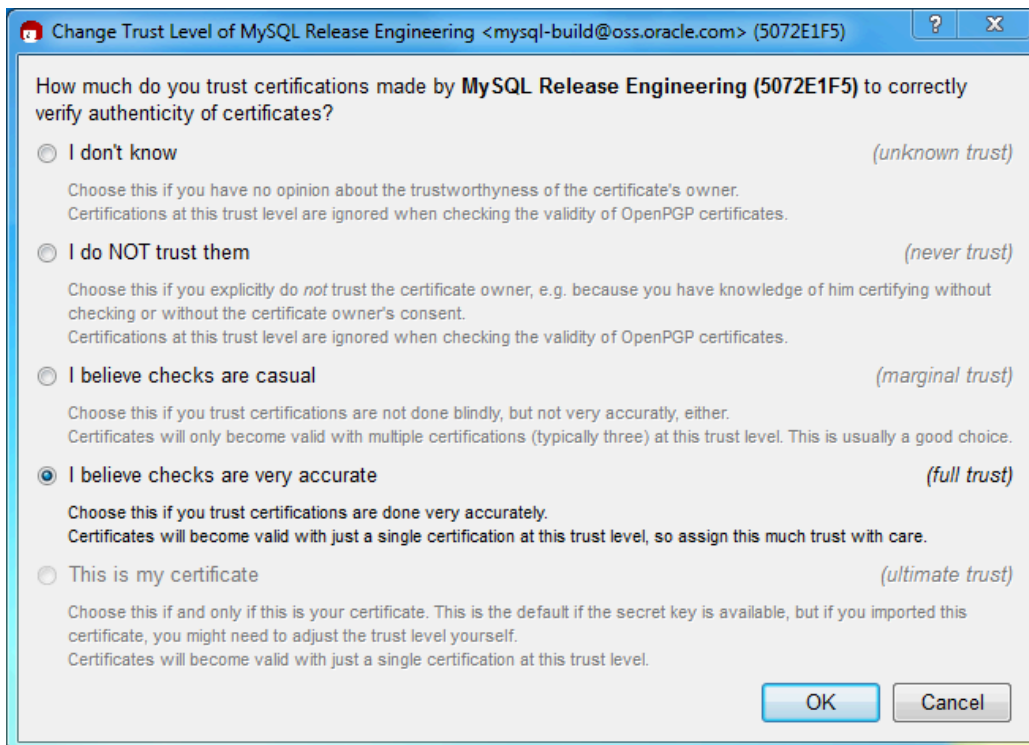
図 2.2 Kleopatra: サーバーウィザードでの証明書のルックアップ: 証明書の検索



「MySQL Release Engineering」証明書を選択します。Fingerprint および Key-ID は「5072E1F5」でなければなりません。そうでない場合は、「Details...」を選択して証明書が有効であることを確認します。次に、「Import」をクリックしてインポートします。インポートダイアログが表示されたら、Okay を選択すると、この証明書が「インポート済証明書」タブの下にリストされます。

次に、弊社の証明書の信頼レベルを構成します。弊社の証明書を選択して、メインメニューから「Certificates」、「Change Owner Trust...」を選択します。弊社の証明書に対して、「I believe checks are very accurate」を選択することを推奨します。そうしないと弊社の署名を確認することができません。「「チェックが非常に正確であると思う」」を選択して「完全信頼」を有効にし、「OK」を押します。

図 2.3 Kleopatra: MySQL リリースエンジニアリングの信頼レベルの変更



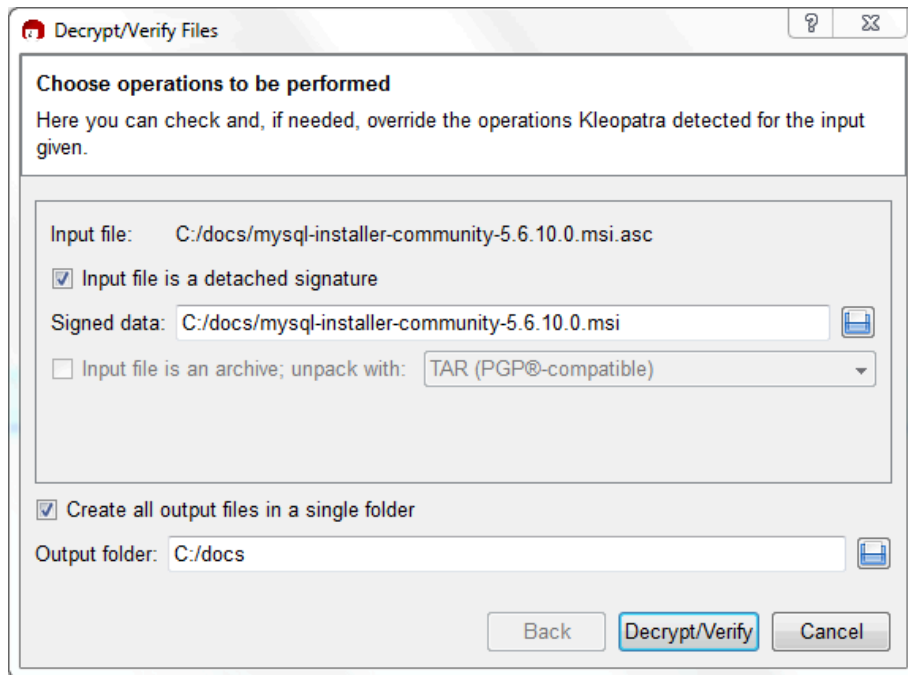
次に、ダウンロードした MySQL パッケージファイルを確認します。これには、パッケージファイルと署名の両方のファイルが必要です。次の表の例に示すように、署名ファイルの名前はパッケージファイルと同じで、拡張子 `.asc` が付いていなければなりません。署名は各 MySQL 製品のダウンロードページにリンクされています。この署名で `.asc` ファイルを作成しなければなりません。

表 2.2 MySQL Installer for Microsoft Windows の MySQL パッケージと署名ファイル

ファイルタイプ	ファイル名
配布ファイル	<code>mysql-installer-community-8.0.29.msi</code>
署名ファイル	<code>mysql-installer-community-8.0.29.msi.asc</code>

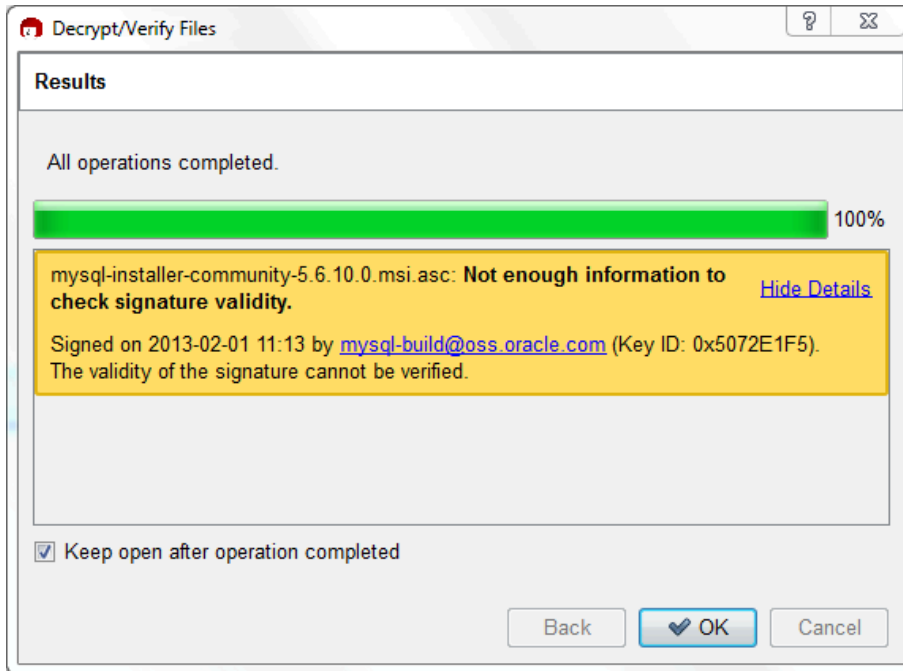
両方のファイルが同じディレクトリにあることを確認してから、次のコマンドを実行して配布ファイルの署名を確認します。署名 (`.asc`) ファイルを Kleopatra にドラッグ&ドロップするか、「File」、「Decrypt/Verify Files...」からダイアログをロードしてから `.msi` ファイルまたは `.asc` ファイルのいずれかを選択します。

図 2.4 Kleopatra: ファイルの復号化と検証ダイアログ



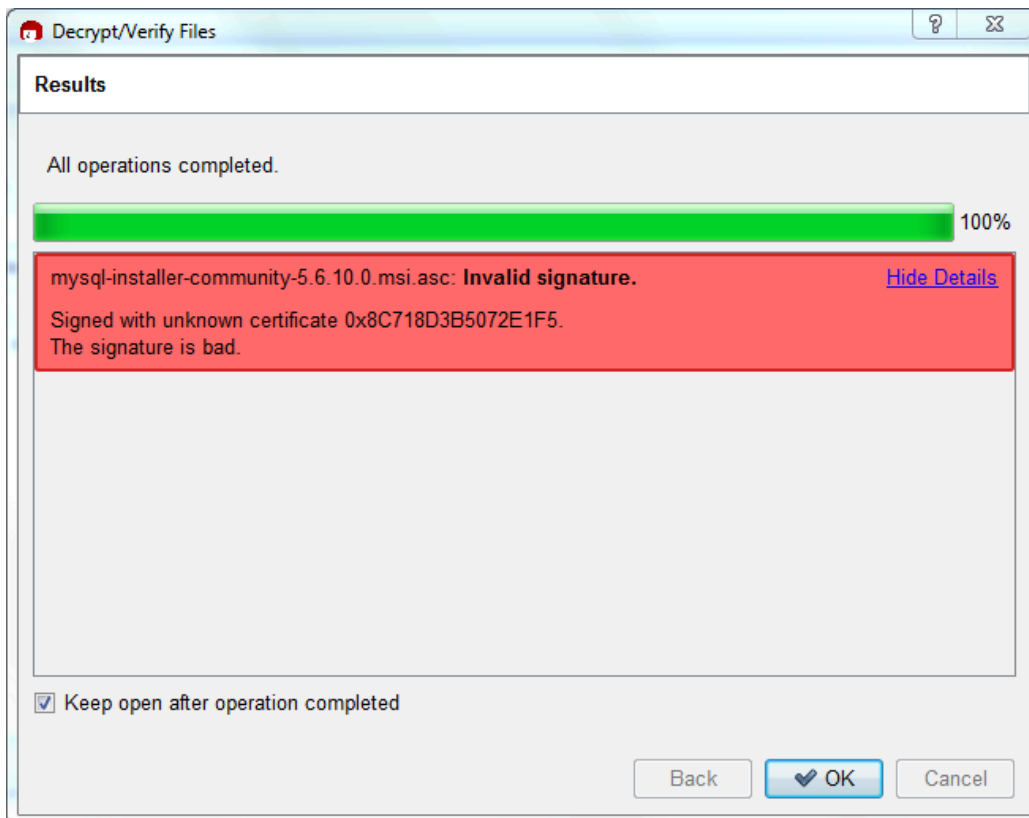
「Decrypt/Verify」をクリックしてファイルをチェックします。最も一般的な 2 つの結果は次の図のようになります。黄色の警告は問題のように見えますが、次のことはファイルチェックが成功したことを意味します。このインストーラを実行できます。

図 2.5 Kleopatra: 結果の復号化と検証ダイアログ: すべての操作が完了しました



赤い「署名が正しくありません」エラーが表示される場合は、ファイルが無効であることを意味します。このエラーが表示された場合は、MSI ファイルを実行しないでください。

図 2.6 Kleopatra: 結果の復号化と検証ダイアログ: Bad



セクション2.1.4.2「GnuPGを使用した署名確認」のセクションでは、緑色の Good signature 結果が表示されない理由について説明します。

2.1.4.4 RPM を使用した署名確認

RPM パッケージには、個別の署名はありません。RPM パッケージには内蔵の GPG 署名および MD5 チェックサムがあります。次のコマンドを実行してパッケージを確認できます。

```
shell> rpm --checksig package_name.rpm
```

例:

```
shell> rpm --checksig MySQL-server-8.0.29-0.linux_glibc2.5.i386.rpm
MySQL-server-8.0.29-0.linux_glibc2.5.i386.rpm: md5 gpg OK
```

注記

RPM 4.1 を使用していて、「(GPG) NOT OK (MISSING KEYS:GPG#5072e1f5)」と表示された場合、MySQL 公開ビルド鍵を自分の GPG 鍵リングにインポートした場合でも、この鍵をまず RPM 鍵リングにインポートする必要があります。RPM 4.1 は、お客様個人の GPG 鍵リング (あるいは GPG そのもの) を使用しなくなりました。RPM はシステム全体にわたるアプリケーションであり、ユーザーの GPG 公開鍵リングはユーザー固有のファイルであるため、RPM はむしろ独立した鍵リングを保持します。MySQL 公開鍵を個人の RPM 鍵リングにインポートするには、まず鍵を取得してから、`rpm --import` を使用して鍵をインポートします。例:

```
shell> gpg --export -a 5072e1f5 > 5072e1f5.asc
shell> rpm --import 5072e1f5.asc
```

あるいは、`rpm` は鍵を直接 URL からロードすることもサポートしていますので、このマニュアルページを使用できません。

```
shell> rpm --import https://dev.mysql.com/doc/refman/8.0/en/checking-gpg-signature.html
```

MySQL 公開鍵を取得する必要がある場合には、セクション2.1.4.2「GnuPGを使用した署名確認」を参照してください。

2.1.5 インストールのレイアウト

インストールのレイアウトは、インストールの種類 (ネイティブパッケージ、バイナリ tarball、ソース tarball など) によって異なります。そのため、異なるシステムを管理する場合や異なるインストールソースを使用する場合に混乱がちです。個々のレイアウトは、次に記載するように、対応するインストールの種類またはプラットフォームの章にあります。オラクル社以外のベンダーからのインストールのレイアウトは、これらのレイアウトと異なる場合がありますのでご注意ください。

- [セクション2.3.1「Microsoft Windows 上での MySQL のインストールレイアウト」](#)
- [セクション2.9.3「ソースインストールの MySQL のレイアウト」](#)
- [表2.3「一般的な Unix/Linux バイナリパッケージの MySQL インストールのレイアウト」](#)
- [表2.12「MySQL Developer Zone からの Linux RPM パッケージの MySQL インストールレイアウト」](#)
- [表2.7「macOS での MySQL のインストールレイアウト」](#)

2.1.6 コンパイラ固有のビルドの特徴

場合によっては、MySQL のビルドに使用するコンパイラによって、使用できる機能が影響を受けることがあります。このセクションのノートは、オラクル社が提供するバイナリ配布またはご自身でソースからコンパイルしたものに適用されます。

`icc` (Intel C++ コンパイラ) ビルド

icc でビルドされたサーバーは次の特徴があります。

- SSL のサポートは含まれません。

2.2 一般的なバイナリを使用した MySQL の Unix/Linux へのインストール

オラクル社は、MySQL の一連のバイナリ配布を提供しています。これらには、多数のプラットフォーム用の圧縮 tar ファイル (.tar.xz 拡張子を持つファイル) の形式の汎用バイナリ配布、および選択したプラットフォーム用のプラットフォーム固有のパッケージ形式のバイナリが含まれます。

このセクションでは、Unix/Linux プラットフォームでの圧縮 tar ファイルバイナリディストリビューションからの MySQL のインストールについて説明します。その他のプラットフォーム固有のバイナリパッケージ形式については、このマニュアルの他のプラットフォーム固有のセクションを参照してください。たとえば、Windows の配布は [セクション2.3「Microsoft Windows に MySQL をインストールする」](#) を参照してください。様々な配布形式で MySQL を取得する方法については、[セクション2.1.3「MySQL の取得方法」](#) を参照してください。

MySQL 圧縮 tar ファイルのバイナリ配布の名前の形式は `mysql-VERSION-OS.tar.xz` で、`VERSION` は数値 (8.0.29 など) で、`OS` は配布対象のオペレーティングシステムのタイプを示します (`pc-linux-i686` や `winx64` など)。

Linux 汎用バイナリ配布用の MySQL 圧縮 tar ファイルの「最小インストール」バージョンもあり、`mysql-VERSION-OS-GLIBCVER-ARCH-minimal.tar.xz` という形式の名前が付けられています。最小インストール配布では、デバッグバイナリが除外され、デバッグシンボルが取り除かれて、通常のバイナリ配布より大幅に小さくなります。最小限のインストール配布をインストールする場合は、次の手順のファイル名形式の違いに合わせて調整してください。

警告

- Yum や APT などのオペレーティングシステムのネイティブパッケージ管理システムを使用して以前に MySQL をインストールした場合、ネイティブバイナリを使用したインストールで問題が発生することがあります。以前の MySQL インストールが (パッケージ管理システムを使用して) 完全に削除されていること、および古いバージョンのデータファイルなどの追加ファイルも削除されていることを確認します。また、`/etc/my.cnf` や `/etc/mysql` ディレクトリなどの構成ファイルを確認して削除する必要があります。

サードパーティパッケージを公式の MySQL パッケージに置き換える方法の詳細は、関連する [「APT ガイド」](#) または [Yum guide](#) を参照してください。

- MySQL には、`libaio` ライブラリへの依存関係があります。このライブラリがローカルにインストールされていない場合、データディレクトリの初期化および後続のサーバー起動ステップは失敗します。必要に応じて、適切なパッケージマネージャーを使用してインストールします。たとえば、Yum ベースのシステムでは次のようにします。

```
shell> yum search libaio # search for info
shell> yum install libaio # install library
```

または、APT ベースのシステムの場合:

```
shell> apt-cache search libaio # search for info
shell> apt-get install libaio1 # install library
```

- Oracle Linux 8 / Red Hat 8 (EL8): これらのプラットフォームでは、デフォルトでファイル `lib64/libtinfo.so.5` はインストールされません。これは、パッケージ `mysql-VERSION-el7-x86_64.tar.gz` および `mysql-VERSION-linux-glibc2.12-x86_64.tar.xz` の MySQL クライアント `bin/mysql` に必要です。この問題を回避するには、`ncurses-compat-libs` パッケージをインストールします:

```
shell> yum install ncurses-compat-libs
```

圧縮された tar ファイルのバイナリ配布をインストールするには、選択したインストール場所 (通常は `/usr/local/mysql`) で解凍します。これにより、次の表に示すディレクトリが作成されます。

表 2.3 一般的な Unix/Linux バイナリパッケージの MySQL インストールのレイアウト

ディレクトリ	ディレクトリの内容
bin	mysqld サーバー、クライアントプログラムおよびユーティリティプログラム
docs	情報形式の MySQL マニュアル
man	Unix マニュアルページ
include	インクルード(ヘッダー) ファイル
lib	ライブラリ
share	データベースインストールのエラーメッセージ、ディクショナリおよび SQL
support-files	その他のサポートファイル

mysqld バイナリのデバッグバージョンは、[mysqld-debug](#) として利用可能です。ソース配布から独自のデバッグバージョンの MySQL をコンパイルするには、適切な構成オプションを使用してデバッグのサポートを有効にします。[セクション2.9「ソースから MySQL をインストールする」](#)を参照してください。

MySQL バイナリ配布をインストールして使用する場合、コマンドシーケンスは次のようになります:

```
shell> groupadd mysql
shell> useradd -r -g mysql -s /bin/false mysql
shell> cd /usr/local
shell> tar xvf /path/to/mysql-VERSION-OS.tar.xz
shell> ln -s full-path-to-mysql-VERSION-OS mysql
shell> cd mysql
shell> mkdir mysql-files
shell> chown mysql:mysql mysql-files
shell> chmod 750 mysql-files
shell> bin/mysqld --initialize --user=mysql
shell> bin/mysql_ssl_rsa_setup
shell> bin/mysqld_safe --user=mysql &
# Next command is optional
shell> cp support-files/mysql.server /etc/init.d/mysql.server
```

注記

この手順では、システムへの root (管理者) アクセスがあるものとします。または、`sudo` (Linux) または `pfexec` (Solaris) コマンドを使用して、各コマンドに接頭辞を付けることもできます。

`mysql-files` ディレクトリは、インポートおよびエクスポート操作を特定のディレクトリに制限する `secure_file_priv` システム変数の値として使用する便利な場所を提供します。[セクション5.1.8「サーバーシステム変数」](#)を参照してください。

バイナリ配布のインストールに関する前述の説明を、次に詳細に説明します。

mysql ユーザーおよびグループの作成

`mysqld` の実行に使用するユーザーおよびグループがシステムにまだない場合は、作成する必要がある場合があります。次のコマンドは、`mysql` グループおよび `mysql` ユーザーを作成します。ユーザーやグループを `mysql` のではなく別の名前に変更したい場合があります。その場合は、以降の説明では適切な名前に置き換えてください。`useradd` および `groupadd` の構文は、Unix/Linux のバージョンによって若干異なる場合や、`adduser` や `addgroup` などの名前が異なる場合があります。

```
shell> groupadd mysql
shell> useradd -r -g mysql -s /bin/false mysql
```

注記

ユーザーは所有権の目的でのみ必要であり、ログイン目的では必要ないため、`useradd` コマンドは `-r` および `-s /bin/false` オプションを使用して、サーバーホストへのログイン権限を持

たないユーザーを作成します。 `useradd` でサポートされていない場合は、これらのオプションを省略します。

配布の取得とアンパック

配布をアンパックするディレクトリを選択してそこに移動します。この例では、配布を `/usr/local` の下にアンパックします。したがって、この説明では `/usr/local` にファイルおよびディレクトリを作成する許可があるものとします。そのディレクトリが保護されている場合は、インストールを `root` として実行する必要があります。

```
shell> cd /usr/local
```

配布ファイルを [セクション2.1.3「MySQLの取得方法」](#) の説明に従って取得します。所定のリリースでは、すべてのプラットフォームのバイナリの配布は同じ MySQL のソース配布でビルドされています。

ディストリビューションを解凍すると、インストールディレクトリが作成されます。 `z` オプションがサポートされている場合、 `tar` は配布を解凍して解凍できます:

```
shell> tar xvf /path/to/mysql-VERSION-OS.tar.xz
```

`tar` コマンドが `mysql-VERSION-OS` という名前のディレクトリを作成します。

圧縮された `tar` ファイルのバイナリ配布から MySQL をインストールするには、配布を解凍する GNU `XZ Utils` と、それを解凍する適切な `tar` がシステムに必要です。

注記

MySQL Server 8.0.12 で圧縮アルゴリズムが Gzip から XZ に変更され、汎用バイナリファイル拡張子が `.tar.gz` から `.tar.xz` に変更されました。

GNU `tar` が機能することが知られています。一部のオペレーティングシステムで提供される標準の `tar` は、MySQL 配布内の長いファイル名をアンパックできません。GNU `tar` をダウンロードしてインストールするか、プリインストールバージョンの GNU `tar` が利用可能であればそれを使用します。通常、これは `gnutar`、`gtar`、または `tar` という名前です (`/usr/sfw/bin` または `/usr/local/bin` などの GNU または Free Software ディレクトリ内)。GNU `tar` は、<http://www.gnu.org/software/tar/> から入手可能です。

`tar` で `xz` 形式がサポートされていない場合は、`xz` コマンドを使用してディストリビューションを解凍し、`tar` を使用して解凍します。前述の `tar` コマンドを次の代替コマンドに置き換えて、配布を展開して抽出します。

```
shell> xz -dc /path/to/mysql-VERSION-OS.tar.xz | tar x
```

次に、`tar` によって作成されたインストールディレクトリへのシンボリックリンクを作成します:

```
shell> ln -s full-path-to-mysql-VERSION-OS mysql
```

`ln` コマンドは、インストールディレクトリへのシンボリックリンクを作成します。これにより、より簡単に `/usr/local/mysql` と呼ぶことができます。MySQL を使用しているときに常にクライアントプログラムのパス名を入力する必要がないようにするには、`/usr/local/mysql/bin` ディレクトリを `PATH` 変数に追加します:

```
shell> export PATH=$PATH:/usr/local/mysql/bin
```

インストール後のセットアップの実行

インストールプロセスの残りの部分では、配布の所有権とアクセス権限の設定、データディレクトリの初期化、MySQL サーバーの起動および構成ファイルの設定を行います。その手順は、[セクション2.10「インストール後のセットアップとテスト」](#)を参照してください。

2.3 Microsoft Windows に MySQL をインストールする

重要

MySQL 8.0 Server を Windows プラットフォームで実行するには、Microsoft Visual C++ 2015 再配布可能パッケージが必要です。サーバーを設置する前に、パッケージがシステムにインストールされていることを確認するようにしてください。このパッケージは、「[Microsoft ダウンロードセンター](#)」で入手できます。また、MySQL デバッグバイナリには Visual Studio 2015 をインストールする必要があります。

MySQL は、Microsoft Windows 64-bit オペレーティングシステムでのみ使用できます。サポートされる Windows プラットフォームの情報については、<https://www.mysql.com/support/supportedplatforms/database.html> を参照してください。

Microsoft Windows に MySQL をインストールするには、様々な方法があります。

MySQL Installer メソッド

最も簡単で推奨される方法は、MySQL Installer (Windows の場合) をダウンロードし、特定のバージョンの MySQL Server を次のようにインストールして構成することです：

1. MySQL Installer を <https://dev.mysql.com/downloads/installer/> からダウンロードして実行します。

注記

標準の MySQL Installer とは異なり、小さいバージョンの [web-community](#) には MySQL アプリケーションはバンドルされませんが、インストールするように選択した MySQL 製品のみがダウンロードされます。

2. MySQL 製品の初期インストールに使用する設定タイプを決定します。例：
 - 開発者デフォルト: 選択したバージョンの MySQL Server および MySQL 開発に関連するその他の MySQL ツール (MySQL Workbench など) を含む設定タイプを提供します。
 - サーバーのみ: 他の製品を含まない、選択したバージョンの MySQL Server の設定を提供します。
 - カスタム: MySQL Server およびその他の MySQL 製品の任意のバージョンを選択できます。
3. サーバーインスタンス (および製品) をインストールし、最初にサーバーインスタンスに対して次のいずれかのレベルの可用性を選択して、サーバー構成を開始します：
 - スタンドアロン MySQL Server / Classic MySQL レプリケーション (デフォルト)
高可用性なしで実行するようにサーバーインスタンスを構成します。
 - InnoDB クラスター
MySQL Group Replication に基づく次のような構成オプションを提供します：
 - ローカルホスト上のサンドボックス InnoDB クラスターで複数のサーバーインスタンスを構成します (テストのみ)。
 - 新しい InnoDB クラスターを作成し、シードインスタンスを構成するか、既存の InnoDB クラスターに新しいサーバーインスタンスを追加します。
4. 画面の指示に従って、構成プロセスを完了します。個々のステップの詳細は、[MySQL Installer での MySQL Server の構成](#) を参照してください。

MySQL がインストールされました。MySQL をサービスとして構成した場合、システムを再起動するたびに Windows によって MySQL サーバーが自動的に起動されます。また、このプロセスでは、MySQL Installer アプリケーションがローカルホストにインストールされます。このアプリケーションは、後で MySQL サーバーのアップグレードまたは再構成に使用できます。

注記

MySQL Workbench をシステムにインストールした場合は、それを使用して新しい MySQL サーバー接続を確認することを検討してください。デフォルトでは、プログラムは MySQL のインストール後に自動的に起動します。

インストールの追加情報

MySQL は、標準アプリケーションまたは Windows サービスとして実行できます。サービスを使用することで、標準 Windows サービス管理ツールを介してサーバーの動作をモニターおよび制御できます。詳細については、[セクション 2.3.4.8 「Windows のサービスとして MySQL を起動する」](#) を参照してください。

RESTART ステートメントに対応するために、MySQL サーバーはサービスまたはスタンドアロンとして実行されるとフォークし、モニタープロセスがサーバープロセスを監視できるようにします。この場合、2つの `mysqld` プロセスがあります。**RESTART** 機能が必要ない場合は、`--no-monitor` オプションを使用してサーバーを起動できます。[セクション13.7.8.8「RESTART ステートメント」](#)を参照してください。

一般的には、Windows に MySQL をインストールするには管理者権限のアカウントを使用する必要があります。そうでない場合、`PATH` 環境変数の編集あるいは **サービス管理マネージャー** にアクセスするなどの操作の場合に、問題が発生することがあります。インストール時に、管理者権限を持つユーザーを使用して MySQL を実行する必要はありません。

Windows プラットフォームで MySQL を使用する場合の制限のリストは、[セクション2.3.7「Windows プラットフォームの制限事項」](#)を参照してください。

MySQL Server パッケージのほかに、アプリケーションまたは開発環境で MySQL を使用するために、追加のコンポーネントが必要またはあると便利な場合があります。次のものが含まれますがこれに限りません。

- ODBC を使用して MySQL Server に接続するには、Connector/ODBC ドライバが必要です。インストールおよび構成の説明を含む詳細情報は、[MySQL Connector/ODBC Developer Guide](#) を参照してください。

注記

MySQL Installer によってコネクタ/ODBC がインストールおよび構成されます。

- .NET アプリケーションで MySQL サーバーを使用するには、Connector/.NET ドライバが必要です。インストールおよび構成の説明を含む詳細情報は、[MySQL Connector/.NET Developer Guide](#) を参照してください。

注記

MySQL Installer により、MySQL Connector/.NET がインストールおよび構成されます。

MySQL の Windows 版配布は、<https://dev.mysql.com/downloads/> からダウンロードできます。[セクション2.1.3「MySQL の取得方法」](#)を参照してください。

MySQL for Windows は、ここで詳しく説明するいくつかの配布形式で使用できます。一般的には、MySQL Installer を使用するとよいでしょう。これには、古い MSI よりも多くの機能と MySQL 製品が含まれており、圧縮ファイルよりも簡単に使用できるため、MySQL を起動して実行するための追加ツールは必要ありません。MySQL Installer は、MySQL Server および追加の MySQL 製品を自動的にインストールし、オプションファイルを作成してサーバーを起動し、デフォルトのユーザーアカウントを作成できるようにします。パッケージ選択の詳細は、[セクション2.3.2「インストールパッケージの選択」](#)を参照してください。

- MySQL Installer ディストリビューションには、MySQL Server および MySQL Workbench や MySQL for Visual Studio などの追加の MySQL 製品が含まれます。MySQL Installer を使用して、将来これらの製品をアップグレードすることもできます (<https://dev.mysql.com/doc/mysql-compat-matrix/en/> を参照)。

MySQL Installer を使用して MySQL をインストールする方法の説明は、[セクション2.3.3「MySQL Installer for Windows」](#)を参照してください。

- 標準バイナリ配布 (圧縮ファイルとしてパッケージ化) には、選択した場所に解凍する必要があるすべてのファイルが含まれます。このパッケージには、完全な Windows MSI Installer パッケージ内のすべてのファイルが含まれますが、インストールプログラムは含まれません。

圧縮ファイルを使用して MySQL をインストールする手順については、[セクション2.3.4「noinstall ZIP アーカイブを使用した Microsoft Windows への MySQL のインストール」](#)を参照してください。

- ソース配布形式には、Visual Studio コンパイラシステムを使用して実行可能ファイルをビルドするためのすべてのコードとサポートファイルが含まれます。

Windows 上で MySQL をソースからビルドする方法の詳細は、[セクション2.9「ソースから MySQL をインストールする」](#)を参照してください。

Windows での MySQL の考慮事項

- 大規模テーブルのサポート

4G バイト以上のテーブルが必要な場合、NTFS 以降のファイルシステムに MySQL をインストールします。テーブルの作成時には、必ず `MAX_ROWS` と `AVG_ROW_LENGTH` を使用します。 [セクション13.1.20「CREATE TABLE ステートメント」](#) を参照してください。

- MySQL およびウイルスチェックソフトウェア

MySQL のデータおよび一時テーブル含むディレクトリに、Norton/Symantec Anti-Virus などのウイルススキャンソフトウェアがあると、MySQL のパフォーマンス、およびウイルススキャンソフトウェアがファイルの内容をスパムを含むものと誤認するという問題が生じる可能性があります。これは、ウイルススキャンソフトウェアのフィンガープリント解析メカニズムと、MySQL がさまざまなファイルを高速で更新する方法が潜在的なセキュリティリスクと認識される可能性があるということが原因です。

MySQL Server のインストール後、MySQL のテーブルデータを格納するために使用するメインディレクトリ (`datadir`) 上でのウイルススキャンを無効にすることが推奨されます。通常、ウイルススキャンソフトウェアには特定のディレクトリを無視するシステムが組み込まれており、有効にできます。

また、MySQL はデフォルトで、標準の Windows 一時ディレクトリに一時ファイルを作成します。一時ファイルがスキャンされることも避けるために、MySQL の一時ファイル用に独立した一時ディレクトリを構成し、このディレクトリをウイルススキャンの除外リストに追加します。これを行うには、`tmpdir` パラメータの構成オプションを `my.ini` 構成ファイルに追加します。詳細については、 [セクション2.3.4.2「オプションファイルの作成」](#) を参照してください。

2.3.1 Microsoft Windows 上での MySQL のインストールレイアウト

Windows 上の MySQL 8.0 の場合、MySQL Installer で実行されるインストールのデフォルトのインストールディレクトリは `C:\Program Files\MySQL\MySQL Server 8.0` です。ZIP アーカイブ方法を使用して MySQL をインストールする場合は、`C:\mysql` にインストールすることをお勧めします。しかし、サブディレクトリのレイアウトは同じです。

すべてのファイルは、次の表に示される構造を使用してこの親ディレクトリ内に配置されています。

表 2.4 Microsoft Windows の MySQL のデフォルトインストールレイアウト

ディレクトリ	ディレクトリの内容	メモ
<code>bin</code>	<code>mysqld</code> サーバー、クライアントプログラムおよびユーティリティプログラム	
<code>%PROGRAMDATA%\MySQL\MySQL Server 8.0\</code>	ログファイル、データベース	Windows システム変数 <code>%PROGRAMDATA%</code> のデフォルトは <code>C:\ProgramData</code> です。
<code>docs</code>	リリースドキュメント	MySQL Installer では、 <code>Modify</code> 操作を使用してこのオプションフォルダを選択します。
<code>include</code>	インクルード (ヘッダー) ファイル	
<code>lib</code>	ライブラリ	
<code>share</code>	エラーメッセージ、文字セットファイル、サンプル構成ファイル、データベースインストールのための SQL を含む種々のサポートファイル	

2.3.2 インストールパッケージの選択

MySQL 8.0 には、Windows に MySQL をインストールする際に選択できるインストールパッケージ形式が複数あります。このセクションで説明するパッケージ形式は次のとおりです:

- MySQL Installer
- MySQL noinstall ZIP アーカイブ

- [MySQL Docker イメージ](#)

プログラムデータベース (PDB) ファイル (ファイル名拡張子 `pdb`) は、問題が発生した場合に MySQL インストールをデバッグするための情報を提供します。これらのファイルは、MySQL の ZIP アーカイブ配布に含まれています (MSI 配布には含まれていません)。

MySQL Installer

このパッケージは、[mysql-installer-community-8.0.29.0.msi](#) または [mysql-installer-commercial-8.0.29.0.msi](#) と同様のファイル名を持ち、MSI を使用して MySQL サーバーおよびその他の製品を自動的にインストールします。MySQL Installer は、自身およびインストールされている各製品に更新をダウンロードして適用します。また、インストールされた MySQL サーバー (サンドボックス InnoDB クラスタテスト設定を含む) および MySQL Router も構成します。MySQL Installer は、ほとんどのユーザーに推奨されます。

MySQL Installer では、次のような他の多くの MySQL 製品をインストールおよび管理 (追加、変更、アップグレードおよび削除) できます:

- アプリケーション - MySQL Workbench、MySQL for Visual Studio、MySQL Shell および MySQL Router (<https://dev.mysql.com/doc/mysql-compat-matrix/en/> を参照)
- コネクタ - MySQL Connector/C++, MySQL Connector/NET, Connector/ODBC, MySQL Connector/Python, MySQL Connector/J, MySQL Connector/Node.js
- ドキュメント - 『MySQL マニュアル』 (PDF 形式)、サンプルおよび例

MySQL Installer は、MySQL でサポートされているすべてのバージョンの Windows で動作します (<https://www.mysql.com/support/supportedplatforms/database.html> を参照)。

注記

MySQL Installer は Microsoft Windows のネイティブコンポーネントではなく .NET に依存しているため、サーバーコアバージョンの Windows Server などの最小限のインストールオプションでは機能しません。

MySQL Installer を使用して MySQL をインストールする手順は、[セクション2.3.3 「MySQL Installer for Windows」](#) を参照してください。

MySQL noinstall ZIP アーカイブ

これらのパッケージには、GUI を除き、完全な MySQL Server インストールパッケージにあるファイルが含まれています。この形式には自動インストーラは含まれていないため、手動でインストールして構成する必要があります。

`noinstall` ZIP アーカイブは、2 つの別々の圧縮ファイルに分割されます。メインパッケージの名前は `mysql-VERSION-winx64.zip` です。これには、システムで MySQL を使用するために必要なコンポーネントが含まれています。オプションの MySQL テストスイート、MySQL ベンチマークスイートおよびデバッグバイナリ/情報コンポーネント (PDB ファイルを含む) は、`mysql-VERSION-winx64-debug-test.zip` という名前の別の圧縮ファイルにあります。

`noinstall` ZIP アーカイブのインストールを選択した場合は、[セクション2.3.4 「noinstall ZIP アーカイブを使用した Microsoft Windows への MySQL のインストール」](#) を参照してください。

MySQL Docker イメージ

Windows プラットフォーム上の Oracle によって提供される MySQL Docker イメージの使用の詳細は、[セクション2.5.6.3 「Docker を使用した Windows およびその他の Linux 以外のプラットフォームへの MySQL のデプロイ」](#) を参照してください。

警告

Oracle によって提供される MySQL Docker イメージは、Linux プラットフォーム専用で構築されています。他のプラットフォームはサポートされておらず、Oracle から MySQL Docker イメージを実行しているユーザーは独自のリスクでこれを実行しています。

2.3.3 MySQL Installer for Windows

MySQL Installer は、Microsoft Windows で実行される MySQL 製品のインストールおよび構成の複雑さを軽減するために設計されたスタンドアロンアプリケーションです。次の MySQL 製品がサポートされています:

- MySQL Servers

MySQL Installer では、複数の個別の MySQL サーバーインスタンスを同じホストに同時にインストールおよび管理できます。たとえば、MySQL Installer では、MySQL 5.6、MySQL 5.7 および MySQL 8.0 の個別のインスタンスを同じホストにインストール、構成およびアップグレードできます。MySQL Installer では、メジャーバージョン番号とマイナーバージョン番号の間のサーバーアップグレードは許可されませんが、リリースシリーズ内でのアップグレードは許可されます (8.0.21 から 8.0.22 へのアップグレードなど)。

注記

MySQL Installer では、コミュニティとコマーシャルの両方のリリースの MySQL サーバーを同じホストにインストールすることはできません。同じホストに両方のリリースが必要な場合は、[ZIP archive](#) ディストリビューションを使用していずれかのリリースをインストールすることを検討してください。

- MySQL アプリケーション

MySQL Workbench、MySQL Shell、MySQL Router および MySQL for Visual Studio。

- MySQL Connector

MySQL Connector/NET、MySQL Connector/Python、MySQL Connector/ODBC、MySQL Connector/J および MySQL Connector/C++。MySQL Connector/Node.js をインストールするには、<https://dev.mysql.com/downloads/connector/nodejs/> を参照してください。

- ドキュメントとサンプル

PDF 形式の『MySQL リファレンスマニュアル』およびバージョン別の MySQL データベースサンプル。

インストール要件

MySQL Installer には、Microsoft .NET Framework 4.5.2 以上が必要です。このバージョンがホストコンピュータにインストールされていない場合は、「[Microsoft の web サイト](#)」にアクセスしてダウンロードできます。

完全バンドルに含まれていない最新の MySQL 製品のメタデータを含むマニフェストをダウンロードするには、インターネット接続が必要です。MySQL Installer では、アプリケーションを初めて起動した後、構成可能な間隔で定期的にマニフェストのダウンロードが試行されます ([MySQL Installer options](#) を参照)。または、[MySQL Installer dashboard](#) で Catalog をクリックして、更新されたマニフェストを手動で取得することもできます。

注記

初回または後続のマニフェストのダウンロードが失敗すると、エラーがログに記録され、セッション中に MySQL 製品へのアクセスが制限される場合があります。MySQL Installer は、初期マニフェスト構造が更新されるまで、起動のたびにマニフェストのダウンロードを試みます。製品の検索の詳細は、[インストールする製品の検索](#) を参照してください。

MySQL Installer コミュニティリリース

<https://dev.mysql.com/downloads/installer/> からソフトウェアをダウンロードして、Windows 用のすべての MySQL 製品の Community リリースをインストールします。次の MySQL Installer パッケージオプションのいずれかを選択します:

- Web : MySQL Installer および構成ファイルのみが含まれます。web パッケージオプションでは、インストール対象として選択した MySQL 製品のみがダウンロードされますが、ダウンロードごとにインターネット接続が必要です。このファイルのサイズは約 2 MB です。ファイル名の形式は `mysql-installer-community-web-VERSION.N.msi` で、`VERSION` は 8.0 などの MySQL サーバーのバージョン番号、`N` は 0 から始まるパッケージ番号です。

- 完全または現在のバンドル: Windows 用のすべての MySQL 製品 (MySQL サーバーを含む) をバンドルします。ファイルサイズは 300 MB を超え、名前の形式は `mysql-installer-community-VERSION.N.msi` で、`VERSION` は 8.0 などの MySQL Server のバージョン番号、`N` は 0 から始まるパッケージ番号です。

MySQL Installer 商用リリース

<https://edelivery.oracle.com/> からソフトウェアをダウンロードして、Windows 用の MySQL 製品の商用リリース (標準または Enterprise Edition) をインストールします。My Oracle Support (MOS) アカウントにログインしている場合、Commercial リリースには Community リリースで使用可能な現在および以前の GA バージョンがすべて含まれますが、開発マイルストーンバージョンは除外されます。ログインしていない場合は、すでにダウンロードしたバンドル製品のリストのみが表示されます。

Commercial リリースには、次の製品も含まれています:

- ワークベンチ SE/EE
- MySQL Enterprise Backup
- MySQL Enterprise Firewall

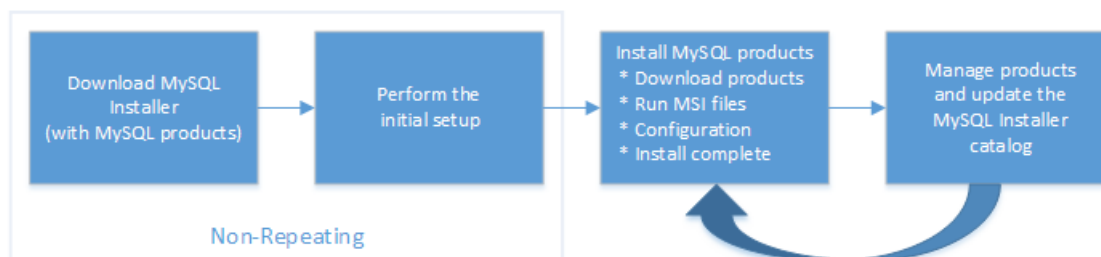
コマーシャルリリースは MOS アカウントと統合されます。ナレッジベースのコンテンツおよびパッチについては、[My Oracle Support](#) を参照してください。

2.3.3.1 MySQL Installer の初期設定

- [設定タイプの選択](#)
- [パスの競合](#)
- [要件のチェック](#)
- [MySQL Installer 構成ファイル](#)

MySQL Installer を初めてダウンロードする場合は、MySQL 製品の初期インストールを順を追って説明します。次の図に示すように、初期設定はプロセス全体のワンタイムアクティビティです。MySQL Installer は、初期設定中にホストにインストールされた既存の MySQL 製品を検出し、管理対象の製品のリストに追加します。

図 2.7 MySQL Installer プロセスの概要



MySQL Installer は、初期設定時に構成ファイル (後述) をホストのハードドライブに抽出します。MySQL Installer は 32-bit アプリケーションですが、32-bit バイナリと 64-bit バイナリの両方をインストールできます。

初期設定では、MySQL グループの下にスタートメニューへのリンクが追加されます。Start、「すべてのプログラム」、「MySQL」、「MySQL インストーラ」をクリックして MySQL Installer を開きます。

設定タイプの選択

初期設定時に、ホストにインストールする MySQL 製品を選択するよう求められます。別の方法として、設定要件に一致する事前定義済の設定タイプを使用することもできます。デフォルトでは、GA 製品と事前リリース製品の

両方が、「開発者デフォルト」、「クライアントのみ」および Full 設定タイプとともにダウンロードおよびインストールに含まれます。これらの設定タイプを使用する場合にのみ GA 製品を含めるように製品セットを制限するには、「GA 製品のみをインストール」オプションを選択します。

次の設定タイプのいずれかを選択すると、初期インストールのみが決定され、後で Windows 用の MySQL 製品をインストールまたは更新する機能は制限されません:

- 開発者デフォルト: MySQL を使用したアプリケーション開発を補完する次の製品をインストールします:
 - [MySQL Server](#) (MySQL Installer のダウンロード時に選択したバージョンをインストールします。)
 - [MySQL Shell](#)
 - [MySQL Router](#)
 - [MySQL Workbench](#)
 - [MySQL for Visual Studio](#)
 - 「[MySQL コネクタ](#)」 (for .NET / Python / ODBC / Java / C++)
 - [MySQL ドキュメント](#)
 - [MySQL のサンプルと例](#)
- Server only: MySQL Server のみをインストールします。この設定タイプでは、MySQL Installer のダウンロード時に選択した General Availability (GA) または開発リリースサーバーがインストールされます。デフォルトのインストールパスとデータベースが使用されます。
- クライアントのみ: 最新の MySQL アプリケーションおよび MySQL コネクタのみをインストールします。この設定タイプは [Developer Default](#) タイプと似ていますが、MySQL サーバーや、通常はサーバーにバンドルされている [mysql](#)、[mysqladmin](#) などのクライアントプログラムが含まれていない点が異なります。
- Full: 使用可能なすべての MySQL 製品をインストールします。
- カスタム: カスタム設定タイプを使用すると、[MySQL Installer catalog](#) から個々の MySQL 製品をフィルタして選択できます。

注記

MySQL Server バージョン 8.0.20(以前)、5.7 および 5.6 の場合、MySQL Installer の実行に使用するアカウントにはサーバーデータファイルをインストールするための十分な権限がない可能性があり、[ExecSecureObjects](#) MSI アクションを実行できないため、インストールが中断される可能性があります。続行するには、サーバーのインストールを再試行する前に、「サーバーデータファイル」機能の選択を解除します。詳細は、[Product Features To Install](#) を参照してください。

「サーバーデータファイル」チェックボックスは、MySQL Server 8.0.21 以上の機能ツリーから削除されました。

[Custom](#) 設定タイプを使用して、次をインストールします:

- 通常のダウンロード場所から入手できない製品または製品のバージョン。カタログには、リリース前(または開発)と GA の間の他のリリースを含む、すべての製品リリースが含まれます。
- 代替インストールパスまたはデータベース(あるいはその両方)を使用する MySQL サーバーのインスタンス。パスの調整方法については、[セクション2.3.3.2「MySQL Installer を使用した代替サーバーパスの設定」](#)を参照してください。
- 同じホスト上の複数の MySQL サーバーバージョン (5.6、5.7、8.0 など)。
- 事前決定済設定タイプとして提供されていない製品と機能の特定の組合せ。たとえば、Windows 用のすべてのクライアントアプリケーションをインストールするかわりに、MySQL Workbench などの単一の製品をインストールできます。

パスの競合

インストールする製品のデフォルトのインストールまたはデータフォルダ (MySQL サーバーで必要) がすでにホストに存在する場合、ウィザードには、各競合を識別するための「パスの競合」ステップが表示され、既存のフォルダ内のファイルが新しいインストールによって上書きされないようにするアクションを実行できます。このステップは、MySQL Installer が競合を検出した場合にのみ初期設定で表示されます。

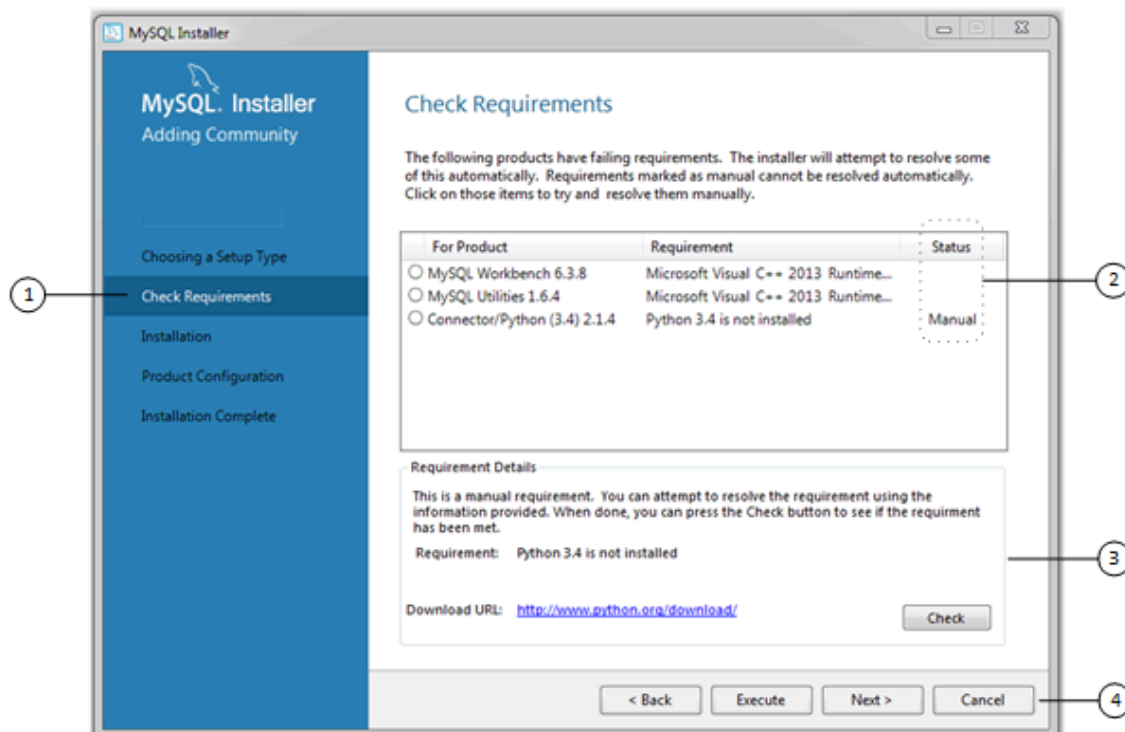
パスの競合を解決するには、次のいずれかを実行します:

- リストから製品を選択して競合オプションを表示します。競合しているパスを示す警告記号が表示されます。参照ボタンを使用して新しいパスを選択し、次へをクリックします。
- 必要に応じて、戻るをクリックして別の設定タイプまたは製品バージョンを選択します。Custom 設定タイプでは、個々の製品バージョンを選択できます。
- 次へをクリックして競合を無視し、既存のフォルダ内のファイルを上書きします。
- 既存の製品を削除します。取消をクリックして初期設定を停止し、MySQL Installer を閉じます。スタートメニューから MySQL Installer を再度開き、MySQL Installer dashboard からの削除操作を使用して、インストールされた製品をホストから削除します。

要件のチェック

MySQL Installer では、`package-rules.xml` ファイルのエントリを使用して、各製品の前提条件となるソフトウェアがホストにインストールされているかどうかを判断します。要件チェックが失敗すると、MySQL Installer には、ホストの更新に役立つ「要件のチェック」ステップが表示されます。要件は、インストール用に新しい製品 (またはバージョン) をダウンロードするたびに評価されます。次の図に、このステップの主な領域を示して説明します。

図 2.8 要件のチェック



要件チェックエレメントの説明

1. 初期設定の現在のステップを表示します。このリストのステップは、ホストにすでにインストールされている製品、前提条件となるソフトウェアの可用性、およびホストにインストールする製品によって若干変わる場合があります。

2. 保留中のすべてのインストール要件を製品別にリストし、ステータスを次のように示します:
 - Status カラムに空白がある場合、MySQL Installer は必要なソフトウェアのダウンロードおよびインストールを試行できます。
 - Status カラムの手動という語は、要件を手動で満たす必要があることを意味します。要件の詳細を表示するには、リストで各製品を選択します。
3. 各手動解決に役立つ要件を詳細に説明します。可能な場合は、ダウンロード URL が提供されます。必要なソフトウェアをダウンロードしてインストールした後、Check をクリックして、要件が満たされていることを確認します。
4. 続行するには、次の集合演算を提供します:
 - 戻る - 前のステップに戻ります。この処理を使用すると、別の設定タイプを選択できます。
 - Execute - MySQL Installer で、すべてのアイテムに必要なソフトウェアを手動ステータスなしでダウンロードしてインストールしようとしています。手動要件は自分で解決し、Check をクリックして確認します。
 - 次へ - 要件のチェックステップに失敗した製品を含めずに、要件を自動的に適用してインストールに進むリクエストを実行しないでください。
 - 取消 - MySQL 製品のインストールを停止します。MySQL Installer はすでにインストールされているため、スタートメニューから MySQL Installer を開き、ダッシュボードから追加をクリックすると、初期設定が再度開始されます。使用可能な管理操作の詳細は、[製品カタログ](#) を参照してください。

MySQL Installer 構成ファイル

すべての MySQL Installer ファイルは、[C:\Program Files \(x86\)](#) および [C:\ProgramData](#) フォルダ内にあります。次のテーブルでは、MySQL Installer をスタンドアロンアプリケーションとして定義するファイルおよびフォルダについて説明します。

注記

MySQL Installer を更新またはアンインストールしても、インストールされた MySQL 製品は変更も削除もされません。

表 2.5 MySQL Installer 構成ファイル

ファイルまたはフォルダ	説明	フォルダ階層
MySQL Installer for Windows	このフォルダには、同様の機能を持つコマンドラインプログラムである MySQL Installer および MySQLInstallerConsole.exe の実行に必要なすべてのファイルが含まれています。	C:\Program Files (x86)
Templates	Templates フォルダには、MySQL サーバーのバージョンごとに 1 つのファイルがあります。テンプレートファイルには、一部の値を動的に計算するためのキーと式が含まれています。	C:\ProgramData\MySQL\MySQL Installer for Windows\Manifest
package-rules.xml	このファイルには、インストールするすべての製品の前提条件が含まれています。	C:\ProgramData\MySQL\MySQL Installer for Windows\Manifest
products.xml	products ファイル (または製品カタログ) には、ダウンロード可能なすべての製品のリストが含まれます。	C:\ProgramData\MySQL\MySQL Installer for Windows\Manifest
Product Cache	Product Cache フォルダには、完全パッケージにバンドルされているか、	C:\ProgramData\MySQL\MySQL Installer for Windows

ファイルまたはフォルダ	説明	フォルダ階層
	後でダウンロードされたすべてのスタンドアロン .msi ファイルが含まれています。	

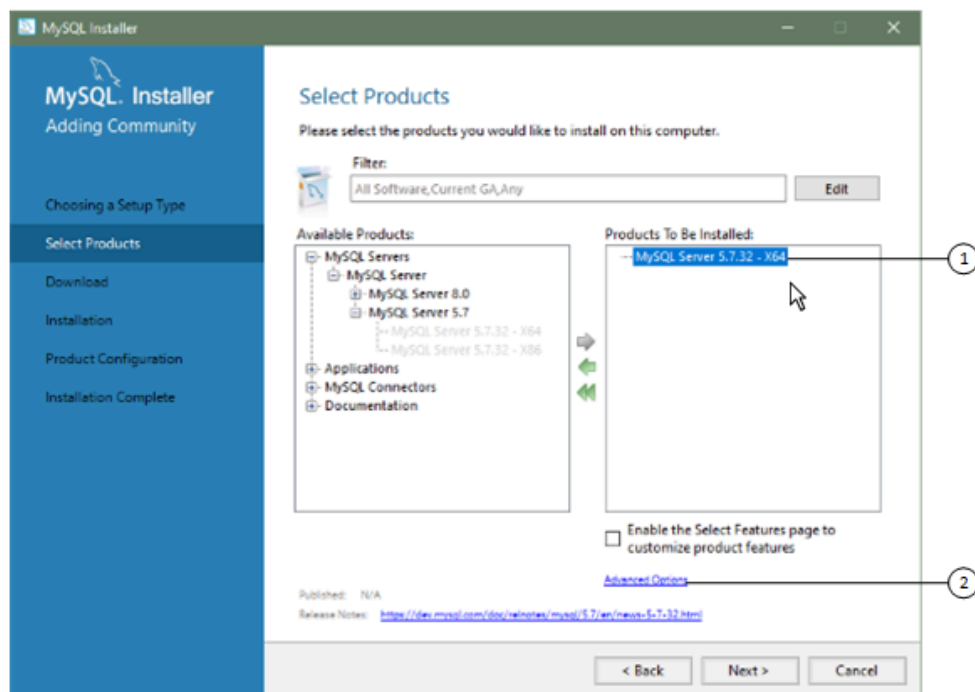
2.3.3.2 MySQL Installer を使用した代替サーバーパスの設定

MySQL サーバーのインストール時に、デフォルトのインストールパスまたはデータパス (あるいはその両方) を変更できます。サーバーをインストールした後は、サーバーインスタンスを削除して再インストールしないとパスを変更できません。

MySQL サーバーのパスを変更するには

- 変更する MySQL サーバーを特定し、「拡張オプション」リンクを有効にします。
 - 次のいずれかを実行して、「製品の選択」ページに移動します:
 - これが MySQL Installer の **initial setup** の場合は、**Custom** 設定タイプを選択して次へをクリックします。
 - MySQL Installer がすでにインストールされている場合は、スタートメニューから起動し、ダッシュボードから追加をクリックします。
 - 編集をクリックして、「使用可能な製品」に表示される製品リストにフィルタを適用します (**インストールする製品の検索** を参照)。
 - サーバーインスタンスを選択した状態で、矢印を使用して、選択したサーバーを「インストールする製品」リストに移動します。
 - サーバーをクリックして選択します。サーバーを選択すると、インストールする製品のリストの下にある「拡張オプション」リンクが有効になります (次の図を参照)。
- 「拡張オプション」をクリックして、代替パス名を入力できるダイアログボックスを開きます。パス名が検証されたら、次へをクリックして構成ステップを続行します。

図 2.9 MySQL Server パスの変更



2.3.3.3 MySQL Installer でのインストールワークフロー

MySQL Installer には、Windows 用の新しい MySQL 製品をインストールおよび構成するためのウィザードのようなツールが用意されています。一度だけ実行される初期設定とは異なり、MySQL Installer は新しい製品をダウンロードまたはインストールするたびにウィザードを起動します。初回インストールの場合、初期設定のステップはインストールのステップに直接進みます。製品の選択については、[インストールする製品の検索](#) を参照してください。

注記

MySQL Installer を実行しているユーザーには、生成されたすべてのファイル (`my.ini` など) に対する完全な権限が付与されます。これは、`SYSTEM` が所有する `%ProgramData%` の MySQL サーバーデータディレクトリなど、特定の製品のファイルおよびディレクトリには適用されません。

ホストにインストールおよび構成される製品は、様々なステップで入力が必要になる可能性のある一般的なパターンに従います。既存の MySQL サーバーのバージョン (またはアップグレード用に選択したバージョン) と互換性のない製品をインストールしようとする、不一致の可能性についてアラートが表示されます。

MySQL Installer には、異なるワークフローに適用される次の一連のアクションが用意されています:

- 製品の選択. 初期設定時に `Custom` 設定タイプを選択した場合、または [MySQL Installer dashboard](#) から「追加」をクリックした場合、MySQL Installer はサイドバーにこのアクションを含めます。このページから、フィルタを適用して使用可能な製品リストを変更し、矢印キーを使用してインストールする製品リストに移動する 1 つ以上の製品を選択できます。

このページのチェックボックスをオンにすると、機能の選択アクションがアクティブになり、製品のダウンロード後に製品機能をカスタマイズできます。

- ダウンロード. `web` 以外の完全な MySQL Installer パッケージをインストールした場合、すべての `.msi` ファイルは初期設定時に `Product Cache` フォルダにロードされ、再度ダウンロードされることはありません。それ以外の場合は、`Execute` をクリックしてダウンロードを開始します。各製品のステータスは、`Ready to Download`、`Downloading`、`Downloaded` の順に変わります。
- インストールする機能の選択 (デフォルトでは無効). MySQL Installer で製品 `.msi` ファイルをダウンロードした後、製品の選択処理中にオプションのチェックボックスを有効にした場合は、機能をカスタマイズできます。インストール後に製品機能をカスタマイズするには、[MySQL Installer dashboard](#) で `Modify` をクリックします。
- インストール. リスト内の各製品のステータスは、`Ready to Install`、`Installing` および最後に `Complete` に変更されます。プロセス中に、「詳細の表示」をクリックしてインストールアクションを表示します。

この時点でインストールを取り消すと、製品はインストールされますが、サーバー (インストールされている場合) はまだ構成されていません。サーバー構成を再起動するには、スタートメニューから MySQL Installer を開き、ダッシュボードの適切なサーバーの横にある「再構成」をクリックします。

- 製品構成. このステップは、MySQL Server、MySQL Router およびサンプルにのみ適用されます。リスト内の各セクション目のステータスは、`Ready to Configure` を示しています。次へをクリックして、リスト内のすべてのアイテムの構成ウィザードを起動します。このステップで表示される構成オプションは、インストールするように選択したデータベースまたはルーターのバージョンによって異なります。

`Execute` をクリックして構成オプションの適用を開始するか、戻る (繰返し) をクリックして各構成ページに戻ります。

- インストールが完了しました. このステップでは、構成を必要としない製品のインストールを完了します。ログをクリップボードにコピーし、MySQL Workbench や MySQL Shell などの特定のアプリケーションを起動できます。終了をクリックして、[MySQL Installer dashboard](#) を開きます。

MySQL Installer での MySQL Server の構成

MySQL Installer は、MySQL サーバーの初期構成を実行します。例:

- MySQL サーバーの構成に使用される構成ファイル (`my.ini`) が作成されます。このファイルに書き込まれる値は、インストールのプロセスで選択した内容に影響されます。一部の定義はホストに依存します。たとえば、ホストのコア数が 3 未満の場合、`query_cache` は有効になります。

注記

クエリーキャッシュは MySQL 5.7 で非推奨になり、MySQL 8.0 (以降) で削除されました。

- デフォルトでは、MySQL サーバーの Windows サービスが追加されます。
- MySQL サーバーのデフォルトのインストールおよびデータパスを提供します。デフォルトのパスを変更する方法については、[セクション 2.3.3.2 「MySQL Installer を使用した代替サーバーパスの設定」](#) を参照してください。
- オプションで、DB 管理者、DB デザイナー、バックアップ管理などの一般的なロールに基づいて構成可能な権限を持つ MySQL サーバーユーザーアカウントを作成できます。オプションで、[MysqSys](#) という名前の制限付き権限を持つ Windows ユーザーを作成し、このユーザーが MySQL Server を実行します。

MySQL Workbench でユーザーアカウントを追加して構成することもできます。

- 「拡張オプションの表示」をチェックすると、追加の「ロギングオプション」を設定できます。これには、エラーログ、一般ログ、スロークエリーログ (クエリーの実行に必要な秒数の構成を含む) およびバイナリログのカスタムファイルパスの定義が含まれます。

構成プロセス中に次へをクリックして次のステップに進むか、戻るをクリックして前のステップに戻ります。最後のステップで Execute をクリックして、サーバー構成を適用します。

次の各セクションでは、Windows 上の MySQL サーバーに適用されるサーバー構成オプションについて説明します。インストールしたサーバーのバージョンによって、構成できるステップとオプションが決まります。MySQL サーバーの構成には、一部またはすべてのステップが含まれる場合があります。

タイプとネットワーキング

- Server の構成タイプ

セットアップを記述する MySQL Server の構成タイプを選択します。この設定では、MySQL サーバーインスタンスに割り当てるシステムリソース (メモリー) の量を定義します。


- 開発: 他の多くのアプリケーションをホストするコンピュータ。通常、これは個人用ワークステーションです。この設定では、最小メモリー量を使用するように MySQL を構成します。
- サーバー: 他のいくつかのアプリケーション (web サーバーなど) は、このコンピュータで実行する必要があります。サーバー設定は、中量のメモリーを使用するように MySQL を構成します。
- 専用: MySQL サーバーの実行専用のコンピュータ。このサーバーでは他の主要なアプリケーションは実行されないため、この設定では、使用可能なメモリーの大部分を使用するように MySQL を構成します。

- 接続性

接続オプションは、MySQL への接続方法を制御します。次のようなオプションがあります。

- TCP/IP: このオプションはデフォルトで選択されています。TCP/IP ネットワーキングを無効にして、ローカルホスト接続のみを許可できます。TCP/IP 接続オプションを選択すると、次の項目を変更できます:

- クラシック MySQL プロトコル 接続用の Port。デフォルト値は **3306** です。
- MySQL 8.0 サーバーのみを構成する場合に表示される「X プロトコル ポート」。デフォルト値は **33060** です
- 「ネットワークアクセス用の Windows ファイアウォールポートを開く」 (TCP/IP 接続の場合はデフォルトで選択されています)。

ポート番号がすでに使用されている場合は、デフォルト値の横に情報アイコン () が表示され、新しいポート番号を指定するまで次へは無効になります。

- 名前付きパイプ: [named_pipe](#) システム変数の設定と同様に、パイプ名を有効にして定義します。デフォルト名は [MySQL](#) です。

- 共有メモリ: `shared_memory` システム変数の設定と同様に、メモリ名を有効にして定義します。デフォルト名は `MySQL` です。
- 高度な構成

「詳細およびロギングオプションの表示」をチェックして、後のステップでカスタムロギングおよび拡張オプションを設定します。ロギングオプションステップでは、エラーログ、一般ログ、スロークエリーログ (クエリーの実行に必要な秒数の構成を含む) およびバイナリログのカスタムファイルパスを定義できます。拡張オプションステップでは、レプリケーショントポロジでバイナリロギングが有効になっている場合に必要ないサーバー ID を設定できます。
- MySQL Enterprise Firewall (Enterprise Edition のみ)

デフォルトでは、「MySQL Enterprise Firewall の有効化」チェックボックスは選択解除されています。特定のタイプの攻撃に対する保護を提供するセキュリティリストを有効にするには、このオプションを選択します。追加のインストール後の構成が必要です ([セクション6.4.7 「MySQL Enterprise Firewall」](#) を参照)。

重要

MySQL 8.0.19 には、サーバーの構成ステップで MySQL Enterprise Firewall を選択した場合にサーバーの起動を妨げる問題があります。サーバーの起動操作が失敗した場合は、取消をクリックして構成プロセスを終了し、ダッシュボードに戻ります。サーバーをアンインストールする必要があります。

回避策は、MySQL Enterprise Firewall を選択せずに MySQL Installer を実行することです。(つまり、「MySQL Enterprise Firewall の有効化」チェックボックスを選択しないでください。)その後、手動インストールの手順を使用して MySQL Enterprise Firewall をインストールします ([セクション6.4.7.2 「MySQL Enterprise Firewall のインストールまたはアンインストール」](#) を参照)。

認証方法

「認証方法」のステップは、MySQL 8.0.4 以上のインストールまたはアップグレード中のみ表示されます。2つのサーバー側認証オプションの中から選択できます。次のステップで作成する MySQL ユーザーアカウントでは、このステップで選択した認証方式が使用されます。

`libmysqlclient` 8.0 を使用する MySQL 8.0 コネクタおよびコミュニティドライバは、`mysql_native_password` のデフォルト認証プラグインをサポートするようになりました。ただし、この新しい認証方式をサポートするようにクライアントおよびアプリケーションを更新できない場合は、レガシー認証に `mysql_native_password` を使用するように MySQL サーバーを構成できます。この変更の影響の詳細は、[優先認証プラグインとしての `caching_sha2_password`](#) を参照してください。

MySQL 8.0.4 以上をインストールまたはアップグレードする場合は、次のいずれかの認証方法を選択します:

- 認証に強力なパスワード暗号化を使用 (推奨)

MySQL 8.0 は、改善された強力な SHA256 ベースのパスワード方式に基づく新しい認証をサポートしています。すべての新しい MySQL サーバーのインストールでは、今後この方法を使用することをお勧めします。

重要

サーバー上の `caching_sha2_password` 認証プラグインには、新しいバージョンのコネクタおよびクライアントが必要です。これにより、新しい MySQL 8.0 デフォルト認証のサポートが追加されます。

- レガシー認証方式の使用 (MySQL 5.x の互換性を維持)


古い MySQL 5.x レガシー認証方式の使用は、次の場合にのみ検討する必要があります:

- アプリケーションは、MySQL 8.0 コネクタおよびドライバを使用するように更新できません。
- 既存のアプリケーションの再コンパイルは実行できません。

- 更新された言語固有のコネクタまたはドライバは、まだ使用できません。

アカウントとロール

- root アカウントのパスワード

root パスワードの割当てが必要で、他の MySQL Installer 操作を実行するときにパスワードの入力を求められます。パスワードの強度は、表示されたボックスでパスワードを繰り返すと評価されます。パスワード要件またはステータスに関する説明情報を表示するには、マウスポインタを情報アイコン () の上に移動します。

- MySQL ユーザーアカウント (オプション)

ユーザーの追加またはユーザーの編集をクリックして、事前定義済ロールを持つ MySQL ユーザーアカウントを作成または変更します。次に、必要なアカウント資格証明を入力します:

- ユーザー名: MySQL ユーザー名の長さは最大 32 文字です。
- ホスト: サーバーへのリモート接続が必要な場合は、`localhost`(ローカル接続のみ) または `<All Hosts (%)>` を選択します。
- ロール: `DB Admin` などの事前定義済の各ロールは、独自の権限セットで構成されます。たとえば、`DB Admin` ロールには、`DB Designer` ロールより多くの権限があります。「ロール」ドロップダウンリストには、各ロールの説明が表示されます。
- パスワード: パスワードの強度評価は、パスワードの入力時に実行されます。パスワードを確認する必要があります。MySQL では、空白または空のパスワードが許可されます (セキュアでないといみなされます)。

MySQL Installer 商用リリースのみ: 商用製品である MySQL Enterprise Edition for Windows は、Windows で外部認証を実行する認証方式もサポートしています。Windows オペレーティングシステムによって認証されたアカウントは、追加のパスワードを指定せずに MySQL サーバーにアクセスできます。

Windows 認証を使用する新しい MySQL アカウントを作成するには、ユーザー名を入力し、「ホスト」および「ロール」の値を選択します。Windows 認証をクリックして、`authentication_windows` プラグインを有効にします。Windows セキュリティトークン領域で、MySQL ユーザー名で認証できる各 Windows ユーザー (またはグループ) のトークンを入力します。MySQL アカウントには、ローカル Windows ユーザーとドメインに属する Windows ユーザーの両方のセキュリティトークンを含めることができます。複数のセキュリティトークンはセミコロン (;) で区切れ、ローカルおよびドメインアカウントには次の形式を使用します:

- ローカルアカウント

各ローカルユーザーまたはグループのセキュリティトークンとして、単純な Windows ユーザー名 (`finley;jeffrey;admin` など) を入力します。

- ドメインアカウント

Windows ドメインのユーザーおよびグループを入力するには、標準の Windows 構文 (`domain \ domainuser`) または MySQL 構文 (`domain \ \ domainuser`) を使用します。

ドメインアカウントの場合、MySQL Installer を実行しているアカウントに Active Directory をクエリーする権限がないと、ドメイン内の管理者の資格証明を使用する必要がある場合があります。この場合は、「Active Directory ユーザーの検証」を選択してドメイン管理者の資格証明をアクティブ化します。

Windows 認証では、トークンを追加または変更するたびに、すべてのセキュリティトークンをテストできます。セキュリティトークンのテストをクリックして、各トークンを検証 (または再検証) します。無効なトークンは、赤の X アイコンと赤のトークンテキストとともに説明的なエラーメッセージを生成します。すべてのトークンが有効 (X アイコンのない緑色のテキスト) として解決されたら、OK をクリックして変更を保存できます。

Windows サービス

Windows プラットフォームでは、MySQL サーバーはオペレーティングシステムによって管理される名前付きサービスとして実行でき、Windows の起動時に自動的に起動するように構成できます。または、手動構成が必要な実行可能プログラムとして実行するように MySQL サーバーを構成することもできます。

- 「MySQL サーバーを Windows サービスとして構成」(デフォルトで選択されます。)

デフォルトの構成オプションが選択されている場合は、次の項目も選択できます:

- システム起動時の MySQL Server の起動

選択すると(デフォルト)、サービス起動タイプは自動に設定されます。選択しない場合、起動タイプは手動に設定されます。

- Windows サービスの実行方法

「標準システムアカウント」が選択されている場合(デフォルト)、サービスはネットワークサービスとしてログオンします。

「カスタムユーザー」オプションには、Microsoft Windows にサービスとしてログオンする権限が必要です。このユーザーが必要な権限で構成されるまで、次へボタンは無効になります。

Windows でカスタムユーザーアカウントを設定するには、スタートメニューでローカルセキュリティポリシーを検索します。ローカルセキュリティポリシーウィンドウで、「ローカルポリシー」、「ユーザー権限の割当て」、「サービスとしてログオン」の順に選択してプロパティダイアログを開きます。ユーザーまたはグループの追加をクリックしてカスタムユーザーを追加し、各ダイアログで OK をクリックして変更を保存します。

- 「Windows サービス」オプションの選択を解除

ログインオプション

このステップは、「タイプとネットワーキング」のステップで「拡張構成の表示」チェックボックスが選択されている場合に使用できます。このステップを今すぐ有効にするには、戻るをクリックして「タイプとネットワーキング」ステップに戻り、チェックボックスを選択します。

拡張構成オプションは、次の MySQL ログファイルに関連しています:

- [Error Log](#)
- [General Log](#)
- [Slow Query Log](#)
- [Bin Log](#)

注記

バイナリログは、MySQL 5.7 以降ではデフォルトで有効になっています。

高度なオプション

このステップは、「タイプとネットワーキング」のステップで「拡張構成の表示」チェックボックスが選択されている場合に使用できます。このステップを今すぐ有効にするには、戻るをクリックして「タイプとネットワーキング」ステップに戻り、チェックボックスを選択します。

拡張構成オプションには次のものがあります:

- サーバー ID

レプリケーショントポロジで使用される一意の識別子を設定します。バイナリログが有効な場合は、サーバー ID を指定する必要があります。デフォルトの ID 値は、サーバーのバージョンによって異なります。詳細は、[server_id](#) システム変数の説明を参照してください。

- テーブル名の大/小文字

サーバーの初期および後続の構成時に、次のオプションを設定できます。MySQL 8.0 リリースシリーズの場合、これらのオプションはサーバーの初期構成にのみ適用されます。

- 小文字

[lower_case_table_names](#) オプション値を 1 (デフォルト) に設定します。この場合、テーブル名は小文字でディスクに格納され、比較では大/小文字は区別されません。

- 指定した大/小文字を保持

[lower_case_table_names](#) オプション値を 2 に設定します。テーブル名は指定したとおりに格納されますが、小文字で比較されます。

サーバー構成の適用

Execute をクリックすると、すべての構成設定が MySQL サーバーに適用されます。「構成ステップ」タブを使用して、各アクションの進行状況を追跡します。成功すると、各アクションのアイコンが白から緑(チェックマーク付き)に切り替わります。それ以外の場合、個々のアクションがタイムアウトすると、プロセスは停止し、エラーメッセージが表示されます。「Log」タブをクリックしてログを表示します。

インストールが正常に完了し、終了をクリックすると、MySQL Installer およびインストールされた MySQL 製品が Microsoft Windows のスタートメニューの [MySQL](#) グループの下に追加されます。MySQL Installer を開くと、インストールされている MySQL 製品がリストされ、他の MySQL Installer 操作が可能な [dashboard](#) がロードされます。

MySQL Installer での MySQL Router の構成

MySQL Installer は、Windows でビジネスクリティカルなアプリケーションを開発および管理するためのツールスイートをダウンロードしてインストールします。スイートは、アプリケーション、コネクタ、ドキュメントおよびサンプルで構成されます。

[initial setup](#) の実行中に、[Server only](#) 以外の事前に決定された設定タイプを選択して、最新の GA バージョンのツールをインストールします。[Custom](#) 設定タイプを使用して、個々のツールまたは特定のバージョンをインストールします。MySQL Installer がすでにホストにインストールされている場合は、「追加」操作を使用して、MySQL Installer ダッシュボードからツールを選択してインストールします。

MySQL Router 構成

MySQL Installer には、MySQL アプリケーションと InnoDB クラスタ の間のトラフィックを転送するために MySQL Router 8.0 のインストール済インスタンスをブートストラップできる構成ウィザードが用意されています。構成すると、MySQL Router はローカルの Windows サービスとして実行されます。

注記

初期インストール後、およびインストールされたルーターを明示的に再構成するときに、MySQL Router を構成するように求められます。対照的に、アップグレード操作では、アップグレードした製品を構成する必要はなく、構成を求めるプロンプトも表示されません。

MySQL Router を構成するには、次を実行します:

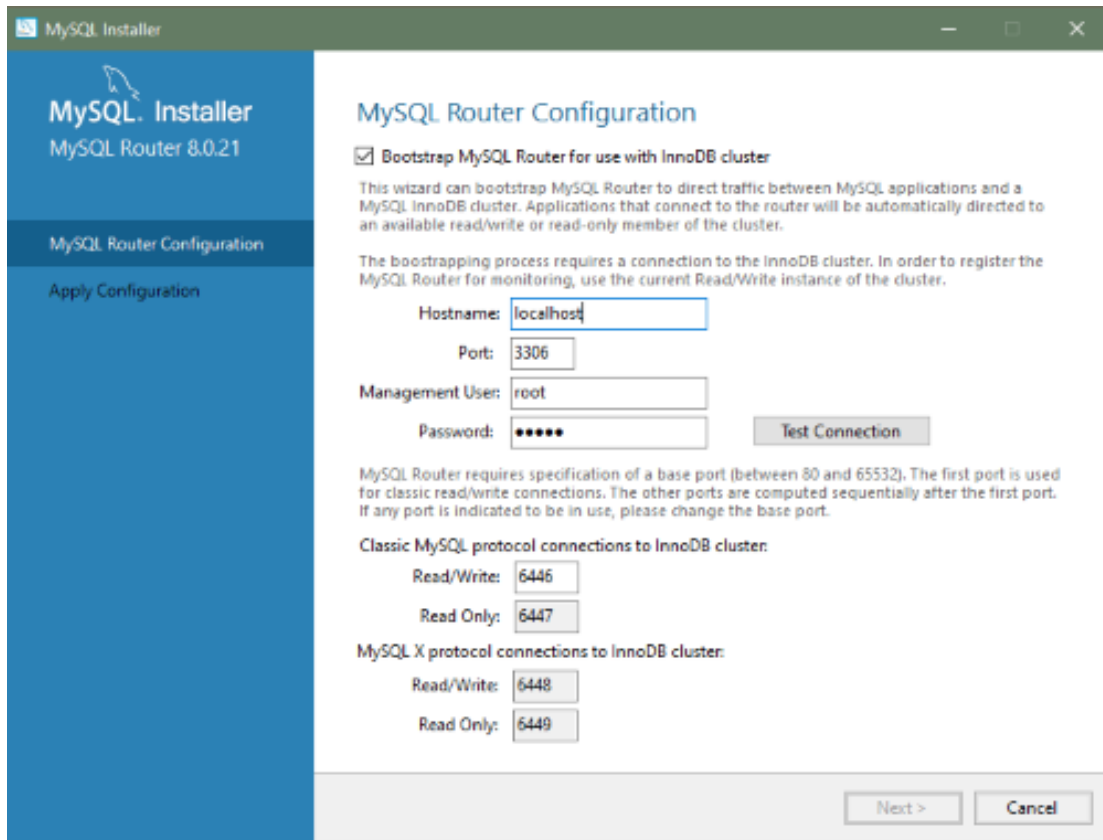
1. InnoDB クラスタ を設定します。
2. MySQL Installer を使用して、MySQL Router アプリケーションをダウンロードしてインストールします。インストールが終了すると、構成ウィザードによって情報の入力を求められます。「InnoDB クラスタ 用の MySQL Router の構成」チェックボックスを選択して構成を開始し、次の構成値を指定します:
 - ホスト名: InnoDB クラスタ (デフォルトでは [localhost](#)) のプライマリ (シード) サーバーのホスト名。
 - ポート: InnoDB クラスタ (デフォルトでは [3306](#)) のプライマリ (シード) サーバーのポート番号。
 - 管理ユーザー: root レベルの権限を持つ管理ユーザー。

- パスワード: 管理ユーザーのパスワード。
- InnoDB クラスタ へのクラシック MySQL プロトコル接続

Read/Write: 最初のベースポート番号を未使用の番号 (80 から 65532) に設定すると、ウィザードによって残りのポートが選択されます。

次の図は、最初のベースポート番号が 6446 に指定され、残りのポートが 6447、6448 および 6449 に設定された MySQL Router 構成ページの例を示しています。

図 2.10 MySQL Router 構成



3. 次に、Execute の順にクリックして、構成を適用します。終了をクリックして MySQL Installer を閉じるか、[MySQL Installer dashboard](#) に戻ります。

MySQL Router の構成後、root アカウントは、`root@%` (リモート) ではなく、`root@localhost` (ローカル) としてのみユーザーテーブルに存在します。ルーターとクライアントが配置されている場所に関係なく、両方がシードサーバーと同じホストに配置されている場合でも、ルーターを通過するすべての接続は、ローカルではなくリモートであるとサーバーによって表示されます。その結果、ローカルホストを使用してサーバーに接続した場合 (次の例を参照)、認証は行われません。

```
shell> \c root@localhost:6446
```

2.3.3.4 MySQL Installer 製品カタログおよびダッシュボード

このセクションでは、MySQL Installer 製品カタログ、ダッシュボード、および製品の選択とアップグレードに関連するその他のアクションについて説明します。

- [製品カタログ](#)
- [MySQL Installer ダッシュボード](#)

- [インストールする製品の検索](#)
- [MySQL Server のアップグレード](#)
- [MySQL Server の削除](#)
- [MySQL Installer のアップグレード](#)

製品カタログ

製品カタログには、「[MySQL のダウンロード](#)」からダウンロードできる Microsoft Windows 用のリリース済 MySQL 製品の完全なリストが格納されます。デフォルトでは、インターネット接続が存在する場合、MySQL Installer は起動時に 7 日ごとにカタログの更新を試みます。ダッシュボードからカタログを手動で更新することもできます (後述)。

最新のカタログは、次のアクションを実行します:

- 「製品の選択」ページの「使用可能な製品」ペインに移入します。このステップは、次を選択すると表示されます:
 - [initial setup](#) 中の [Custom](#) 設定タイプ。
 - ダッシュボードからの「追加」操作。
- ダッシュボードにリストされているインストール済製品に対して製品更新がいつ使用可能になるかを識別します。

カタログには、すべての開発リリース (プレリリース)、一般リリース (現在の GA) およびマイナーリリース (その他のリリース) が含まれます。カタログ内の製品は、ダウンロードする MySQL Installer のリリースによって多少異なります。

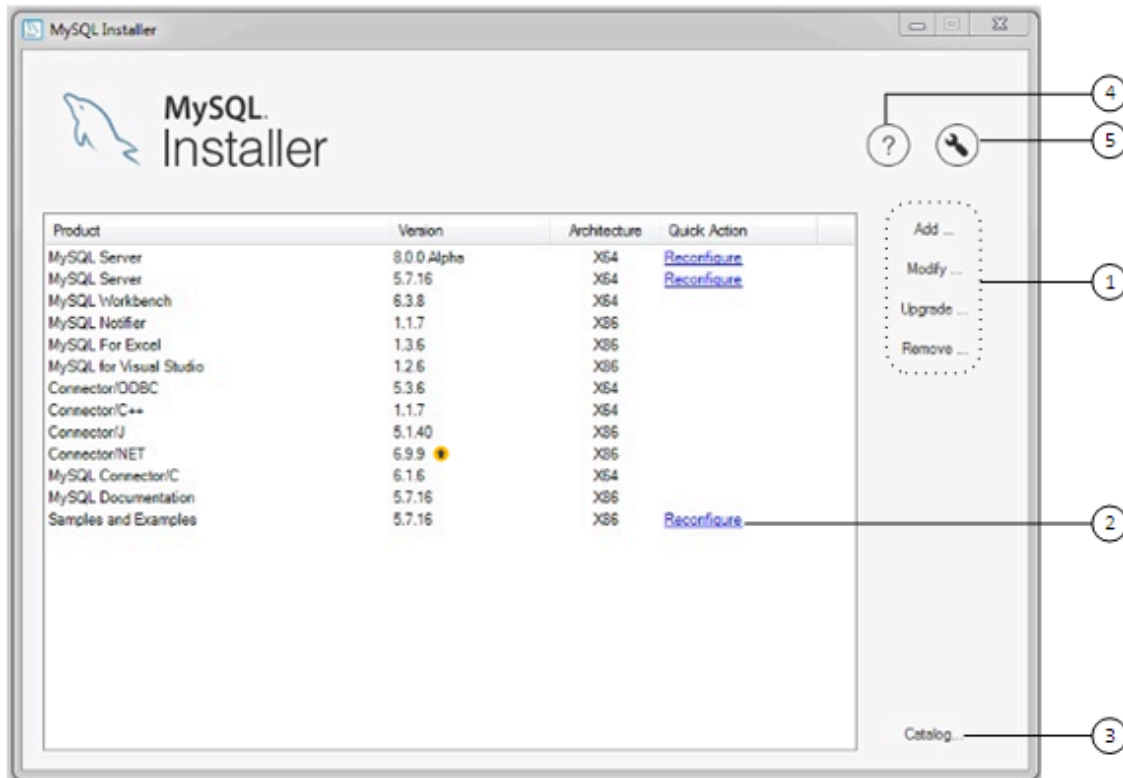
MySQL Installer ダッシュボード

MySQL Installer ダッシュボードは、[initial setup](#) の終了後に MySQL Installer を起動したときに表示されるデフォルトビューです。設定が終了する前に MySQL Installer を閉じた場合、MySQL Installer はダッシュボードを表示する前に初期設定を再開します。

注記

Oracle Lifetime Sustaining Support の対象となる製品 (インストールされている場合) がダッシュボードに表示される場合があります。MySQL for Excel や MySQL Notifier などのこれらの製品は、変更または削除のみが可能です。

図 2.11 MySQL Installer ダッシュボードの要素



MySQL Installer ダッシュボードの要素の説明

- MySQL Installer ダッシュボードの操作には、カタログにリストされているインストール済の製品または製品に適用される様々なアクションが用意されています。次の操作を開始するには、最初に操作リンクをクリックしてから、管理する製品を選択します:
 - 追加: この操作により、「製品の選択」ページが開きます。ここでフィルタを調整し、必要に応じてダウンロードする 1 つ以上の製品を選択して、インストールを開始できます。フィルタの使用に関するヒントは、[インストールする製品の検索](#) を参照してください。

矢印を使用して、各製品を「使用可能な製品」カラムから「インストールする製品」カラムに移動します。機能をカスタマイズできる「製品機能」ページを有効にするには、関連するチェックボックス (デフォルトでは無効) をクリックします。

注記

MySQL Server バージョン 8.0.20(以前)、5.7 および 5.6 の場合、MySQL Installer の実行に使用するアカウントにはサーバーデータファイルをインストールするための十分な権限がない可能性があり、[ExecSecureObjects](#) MSI アクションを実行できないため、インストールが中断される可能性があります。続行するには、サーバーのインストールを再試行する前に、「サーバーデータファイル」機能の選択を解除します。

「サーバーデータファイル」チェックボックスは、MySQL Server 8.0.21 以上の機能ツリーから削除されました。

- Modify: この操作を使用して、インストールされている製品に関連付けられている機能を追加または削除します。変更できる機能は、製品によって複雑さが異なります。「プログラムショートカット」チェックボックスが選択されている場合、製品は [MySQL](#) グループの下のスタートメニューに表示されます。
- アップグレード: この操作により、アップグレードする製品の選択ページがロードされ、すべてのアップグレード候補が移入されます。インストールされた製品には複数のアップグレードバージョンを含めることができ、

操作には現在の製品カタログが必要です。MySQL Installer は、選択したすべての製品を一度にアップグレードします。詳細の表示をクリックして、MySQL Installer によって実行されるアクションを表示します。

- 削除: この操作により、「製品の削除」ページが開き、ホストにインストールされている MySQL 製品が移入されます。削除 (アンインストール) する MySQL 製品を選択し、Execute をクリックして削除プロセスを開始します。操作中、インジケータには、実行されたステップの数がすべてのステップの割合として表示されます。

削除する製品を選択するには、次のいずれかを実行します:


- 1 つまたは複数の製品のチェックボックスをオンにします。
- すべての製品を選択するには、Product チェックボックスを選択します。

2. インストールされている各サーバーの横にあるクイックアクションカラムの「再構成」リンクでは、サーバーの現在の構成値がロードされ、すべての構成ステップが繰り返されて、オプションと値を変更できます。これらの項目を再構成するには、root 権限を持つ資格証明を指定する必要があります。Log タブをクリックして、MySQL Installer によって実行される各構成ステップの出力を表示します。

完了すると、MySQL Installer はサーバーを停止し、構成の変更を適用して、サーバーを再起動します。各構成オプションの詳細は、[MySQL Installer での MySQL Server の構成](#) を参照してください。特定の MySQL サーバーバージョンに関連付けられたインストール済 [Samples and Examples](#) を再構成して、新機能設定を適用することもできます (存在する場合)。


3. Catalog リンクを使用すると、MySQL 製品の最新のカタログを手動でダウンロードし、それらの製品変更を MySQL Installer と統合できます。catalog-download アクションは、ホストにすでにインストールされている製品のアップグレードを実行しません。かわりに、ダッシュボードに戻り、新しいバージョンのインストール済製品ごとにバージョンカラムに矢印アイコンを追加します。「アップグレード」操作を使用して、新しい製品バージョンをインストールします。


Catalog リンクを使用して、新しいカタログをダウンロードせずに各製品の現在の変更履歴を表示することもできます。変更履歴のみを表示するには、「現時点では更新しない」チェックボックスを選択します。

4. MySQL Installer バージョン情報アイコン () には、MySQL Installer の現在のバージョンおよび MySQL に関する一般情報が表示されます。バージョン番号は戻るボタンの上にあります。

ヒント


MySQL Installer の問題を報告する場合は、常にこのバージョン番号を含めてください。

MySQL 情報 () に加えて、サイドパネルから次のアイコンを選択することもできます:

- MySQL Installer のライセンスアイコン ()。

この製品には、ライセンスのもとで使用されるサードパーティ製ソフトウェアが含まれる場合があります。MySQL Installer のコマーシャルリリースを使用している場合は、このコマーシャルリリースに含まれている可能性があるサードパーティソフトウェアに関連するライセンス情報など、ライセンス情報のための MySQL Installer Commercial ライセンス情報ユーザーマニュアル がアイコンによって開きます。コミュニティリリースの MySQL Installer を使用している場合は、この Community リリースに含まれている可能性があるサードパーティソフトウェアに関連するライセンス情報を含む、ライセンス情報のための MySQL Installer Community ライセンス情報ユーザーマニュアル がアイコンによって開きます。

- 最新の MySQL 製品ドキュメント、ブログ、web セミナーなどへのリソースリンクアイコン ()。

5. MySQL Installer オプションアイコン () には、次のタブがあります:

- 製品カタログ: カタログの自動更新を管理します。デフォルトでは、MySQL Installer は起動時に 7 日ごとにカタログ更新をチェックします。新しい製品または製品バージョンが使用可能な場合、MySQL Installer はそれら

をカタログに追加し、ダッシュボードにリストされているインストール済製品のバージョン番号の横に矢印アイコン (↑) を挿入します。

製品カタログオプションを使用して、自動更新を有効または無効にし、自動カタログダウンロード間の日数をリセットします。起動時に、MySQL Installer は設定した日数を使用して、ダウンロードを試行するかどうかを決定します。MySQL Installer でカタログのダウンロード中にエラーが発生した場合、このアクションは次の起動時に繰り返されます。

- 接続設定: MySQL Installer によって実行されるいくつかの操作には、インターネットアクセスが必要です。このオプションを使用すると、デフォルト値を使用して接続を検証したり、別の URL (リストから選択した URL または手動で追加した URL) を使用できます。「手動」オプションを選択すると、新しい URL を追加でき、リスト内のすべての URL を移動または削除できます。「自動」オプションを選択すると、接続が確立されるまで、MySQL Installer はリスト内の各デフォルト URL に (順番に) 接続しようとします。接続できない場合は、エラーが発生します。

インストールする製品の検索

カタログ内の MySQL 製品はカテゴリ別にリストされます: MySQL Servers、アプリケーション、MySQL コネクタおよびドキュメント。デフォルトでは、最新の GA バージョンのみが「使用可能な製品」ペインに表示されます。製品のプレリリースまたは旧バージョンを探している場合は、デフォルトリストに表示されないことがあります。

注記

製品カタログを最新の状態に保ちます。MySQL Installer ダッシュボードで Catalog をクリックして、最新のマニフェストをダウンロードします。

デフォルトの製品リストを変更するには、ダッシュボードで「追加」をクリックして「製品の選択」ページを開き、編集をクリックしてダイアログボックスを開きます (次の図を参照)。設定を変更し、フィルタをクリックします。

図 2.12 使用可能な製品のフィルタ

The screenshot shows a dialog box for filtering products. It has the following elements:

- Text:** An empty text input field.
- Category:** A dropdown menu currently set to "All Software".
- Maturity:** A dropdown menu currently set to "Current GA".
- Already Downloaded:** An unchecked checkbox.
- Architecture:** Three radio buttons: "Any" (selected), "32-bit", and "64-bit".
- Filter:** A button at the bottom right.

次のフィールドのいずれかまたは複数のリセットして、使用可能な製品のリストを変更します:

- テキスト: テキストでフィルタします。
- カテゴリ: すべてのソフトウェア (デフォルト)、MySQL Servers、アプリケーション、MySQL コネクタまたはドキュメント (サンプルおよびドキュメント用)。
- 満期: 現在のバンドル (最初はフルパッケージでのみ表示されます)、プレリリース、現在の GA またはその他のリリース。警告が表示された場合は、MySQL Installer ダッシュボードで Catalog をクリックして、最新の製品マニフェストがあることを確認します。MySQL Installer がマニフェストをダウンロードできない場合、表示される製品の範囲は、バンドル製品、Product Cache フォルダにあるスタンドアロン製品 MSI、またはその両方に限定されません。

注記

リリース前成熟度フィルタを選択した場合、MySQL Installer の商用リリースには MySQL 製品は表示されません。開発中の製品は、Community リリースの MySQL Installer からのみ入手できます。

- すでにダウンロードされています (チェックボックスはデフォルトで選択解除されています)。ダウンロードした製品のみを表示および管理できます。
- アーキテクチャ: 任意 (デフォルト)、32 ビットまたは 64 ビット。

MySQL Server のアップグレード

重要なサーバーのアップグレード条件:

- MySQL Installer では、メジャーリリースバージョン間またはマイナーリリースバージョン間のサーバーアップグレードは許可されませんが、5.7.18 から 5.7.19 へのアップグレードなど、リリースシリーズ内でのアップグレードは許可されます。
- マイルストーンリリース間 (またはマイルストーンリリースから GA リリースへ) のアップグレードはサポートされていません。マイルストーンリリースでは大幅な開発変更が行われ、サーバーの起動時に互換性の問題や問題が発生する可能性があります。
- MySQL 8.0.16 サーバー以上へのアップグレードの場合、チェックボックスを使用すると、データディクショナリテーブルを通常どおりチェックおよび処理しながら、システムテーブルのアップグレードチェックおよびプロセスをスキップできます。MySQL Installer では、以前のサーバーアップグレードがスキップされたとき、またはサーバーがサンドボックス InnoDB クラスタとして構成されたときに、チェックボックスは表示されません。この動作は、MySQL Server がアップグレードを実行する方法の変更を表し ([セクション 2.11.3 「MySQL のアップグレードプロセスの内容」](#) を参照)、MySQL Installer が構成プロセスに適用する一連のステップを変更します。

「システムテーブルのアップグレードチェックおよびプロセスをスキップします。(非推奨)」を選択すると、MySQL Installer によって、アップグレードされたサーバーが `--upgrade=MINIMAL` サーバーオプションで起動され、データディクショナリのみがアップグレードされます。 `--upgrade=MINIMAL` オプションを指定せずにサーバーを停止してから再起動すると、サーバーは必要に応じてシステムテーブルを自動的にアップグレードします。

アップグレード構成 (システムテーブルをスキップ) が完了すると、Log タブおよびログファイルに次の情報が表示されます:

```
WARNING: The system tables upgrade was skipped after upgrading MySQL Server. The server will be started now with the --upgrade=MINIMAL option, but then each time the server is started it will attempt to upgrade the system tables, unless you modify the Windows service (command line) to add --upgrade=MINIMAL to bypass the upgrade.
```

```
FOR THE BEST RESULTS: Run mysqld.exe --upgrade=FORCE on the command line to upgrade the system tables manually.
```

新しいサーバーバージョンを選択するには:

1. 「アップグレード」をクリックします。「アップグレード可能な製品」ペインの製品名の横にあるチェックボックスにチェックマークが付いていることを確認します。現時点ではアップグレードしない製品の選択を解除します。

注記

同じリリースシリーズのサーバーマイルストーンリリースの場合、MySQL Installer はサーバーのアップグレードの選択を解除し、アップグレードがサポートされていないことを示す警告を表示し、続行のリスクを識別して、論理アップグレードを手動で実行するステップのサマリーを提供します。サーバーのアップグレードは、独自のリスクで再選択できます。マイルストーンリリースで論理アップグレードを実行する手順は、[論理アップグレード](#) を参照してください。

2. リスト内の製品をクリックして強調表示します。このアクションにより、選択した製品で使用可能な各バージョンの詳細が「アップグレード可能なバージョン」ペインに移入されます: バージョン番号、公開日、およびそのバージョンのリリースノートを開くための [Changes](#) リンク。

MySQL Server の削除

ローカル MySQL サーバーを削除するには:

1. ローカルデータディレクトリを削除するかどうかを決定します。データディレクトリを保持している場合、別のサーバーインストールでデータを再利用できます。このオプションはデフォルトで有効になっています (データディレクトリを削除します)。
2. Execute をクリックして、ローカルサーバーのアンインストールを開始します。削除対象として選択したすべての製品もこの時点でアンインストールされることに注意してください。
3. (オプション)Log タブをクリックして、MySQL Installer によって実行された現在のアクションを表示します。

MySQL Installer のアップグレード

MySQL Installer はコンピュータにインストールされたままであり、他のソフトウェアと同様に、以前のバージョンからアップグレードできます。場合によっては、互換性のために他の MySQL ソフトウェアで MySQL Installer のアップグレードが必要になることがあります。このセクションでは、現在のバージョンの MySQL Installer を識別する方法と、MySQL Installer を手動でアップグレードする方法について説明します。

インストールされている MySQL Installer のバージョンを検索するには:

1. 検索メニューから MySQL Installer を起動します。MySQL Installer ダッシュボードが開きます。
2. MySQL Installer バージョン情報アイコン (?) をクリックします。バージョン番号は戻るボタンの上にあります。

MySQL Installer のオンデマンドアップグレードを開始するには:

1. MySQL Installer がインストールされているコンピュータをインターネットに接続します。
2. 検索メニューから MySQL Installer を起動します。MySQL Installer ダッシュボードが開きます。
3. ダッシュボード下部の Catalog をクリックして、カタログの更新ウィンドウを開きます。
4. Execute をクリックしてプロセスを開始します。インストールされているバージョンの MySQL Installer をアップグレードできる場合は、アップグレードを開始するように求められます。
5. 次へをクリックしてカタログに対するすべての変更を確認し、終了をクリックしてダッシュボードに戻ります。
6. (新しい) インストール済バージョンの MySQL Installer を確認します (前の手順を参照)。

2.3.3.5 MySQLInstallerConsole リファレンス

[MySQLInstallerConsole.exe](#) には、MySQL Installer と同様のコマンドライン機能が用意されています。これは、MySQL Installer が最初に実行されたときにインストールされ、[MySQL Installer for Windows](#) ディレクトリ内で使用できます。デフォルトでは、`C:\Program Files (x86)\MySQL\MySQL Installer for Windows` にあり、コンソールは管理権限で実行する必要があります。

使用するには、Start、「アクセサリ」を選択してコマンドプロンプトを起動し、「コマンドプロンプト」を右クリックして [Run as administrator](#) を選択します。また、コマンドラインから、オプションで [MySQLInstallerConsole.exe](#) が配置されているディレクトリに変更します:

```
C:\> cd Program Files (x86)\MySQL\MySQL Installer for Windows
C:\Program Files (x86)\MySQL\MySQL Installer for Windows> MySQLInstallerConsole.exe help
===== Start Initialization =====
MySQL Installer is running in Community mode

Attempting to update manifest.
Initializing product requirements
Loading product catalog
Checking for product catalog snippets
Checking for product packages in the bundle
Categorizing product catalog
Finding all installed packages.
Your product catalog was last updated at 11/1/2016 4:10:38 PM
===== End Initialization =====

The following commands are available:

Configure - Configures one or more of your installed programs.
```


Help - Provides list of available commands.
 Install - Install and configure one or more available MySQL programs.
 List - Provides an interactive way to list all products available.
 Modify - Modifies the features of installed products.
 Remove - Removes one or more products from your system.
 Status - Shows the status of all installed products.
 Update - Update the current product catalog.
 Upgrade - Upgrades one or more of your installed programs.

MySQL 製品名

MySQLInstallerConsole コマンドの多くは、カタログ内の MySQL 製品を表すキーワードを受け入れます。次のテーブルに、コマンドで使用できる有効なキーワードの現在のセットを示します。

表 2.6 MySQLInstallerConsole 用の MySQL 製品キーワード

キーワード	MySQL 製品
server	MySQL Server
workbench	MySQL Workbench
shell	MySQL Shell
visual	MySQL for Visual Studio
router	MySQL Router
バックアップ	MySQL Enterprise Backup
net	MySQL Connector/NET
odbc	MySQL Connector/ODBC
c++	MySQL Connector/C++
python	MySQL Connector/Python
j	MySQL Connector/J
documentation	MySQL Server ドキュメント
samples	MySQL サンプル (sakila および world データベース)

MySQLInstallerConsole コマンドオプション

MySQLInstallerConsole.exe は、次のコマンドオプションをサポートしています:

注記

コロン文字 (:) を含む構成ブロック値は、引用符で囲む必要があります。たとえば、installDir="C:\MySQL\MySQL Server 8.0"です。

- `configure [product1]:[setting]=[value]; [product2]:[setting]=[value]; [...]`

システムに 1 つ以上の MySQL 製品を構成します。製品ごとに複数の setting=value ペアを構成できます。

次のようなスイッチがあります。

`-showsettings` `-showsettings` の後に製品名を渡して、選択した製品に使用可能なオプションを表示します。

`-silent` 確認プロンプトを無効にします。

```
C:\> MySQLInstallerConsole configure -showsettings server
C:\> MySQLInstallerConsole configure server:port=3307
```

- `help [command]`

使用例を含むヘルプメッセージを表示して終了します。コマンドに固有のヘルプを受信するには、そのコマンドを渡します。

```
C:\> MySQLInstallerConsole help
C:\> MySQLInstallerConsole help install
```

- `install [product]:[features]:[config block]:[config block]:[config block]; [...]`

システムに MySQL 製品をインストールします。プレリリース製品が使用可能な場合、`-type` スイッチの値が `Developer`、`Client` または `Full` のときに GA 製品とプレリリース製品の両方がインストールされます。これらの設定タイプを使用する場合にのみ、`-only_ga_products` スイッチを使用して製品セットを GA 製品に制限します。

次のようなスイッチと構文オプションがあります。

- `-only_ga_products` GA 製品のみを含むように製品セットを制限します。
- `-type=[SetupType]` 事前定義済のソフトウェアセットをインストールします。設定タイプは次のいずれかです:
 - `Developer`: 完全な開発環境をインストールします。
 - `Server`: 単一の MySQL Server をインストールします。
 - `Client`: クライアントプログラムとライブラリをインストールします。
 - `Full`: すべてをインストールします。
 - `Custom`: ユーザーが選択した製品をインストールします。これはデフォルトのオプションです。

注記

カスタム以外の設定タイプは、他の MySQL 製品がインストールされていない場合にのみ有効です。

- `-showsettings` `-showsettings` の後に製品名を渡して、選択した製品に使用可能なオプションを表示します。
- `-silent` 確認プロンプトを無効にします。
- `[product]` 各製品は、セミコロン区切りのバージョン修飾子を使用するかどうかにかかわらず、`product keyword` で指定できます。プロダクトキーワードだけを渡すと、プロダクトの最新バージョンが選択されます。そのバージョンの製品で複数のアーキテクチャーが使用可能な場合、このコマンドは対話型の確認のためにマニフェストリストの最初のアーキテクチャーを返します。または、`-silent` スイッチを使用して、`product` キーワードの後に正確なバージョンおよびアーキテクチャー (`x86` または `x64`) を渡すこともできます。
- `[features]` MySQL 製品に関連付けられたすべての機能は、デフォルトでインストールされます。機能ブロックは、機能のセミコロン区切りリストまたはすべての機能を選択するアスタリスク文字 (*) です。機能を削除するには、`modify` コマンドを使用します。
- `[config block]` 1 つ以上の構成ブロックを指定できます。各構成ブロックは、セミコロンで区切られたキーと値のペアのリストです。ブロックには、`config` タイプキーまたは `user` タイプキーを含めることができます。定義されていない場合は、`config` がデフォルトタイプです。

コロン文字 (:) を含む構成ブロック値は、引用符で囲む必要があります。たとえば、`installdir="C:\MySQL\MySQL Server 8.0"` です。製品ごとに定義できる構成タイプブロックは 1 つのみです。ユーザーブロックは、製品のインストール時に作成されるユーザーごとに定義する必要があります。

注記

`user` タイプキーは、製品の再構成時にはサポートされません。

```
C:\> MySQLInstallerConsole install server;5.6.25.*:port=3307;serverid=2:type=user;username=foo;password=bar;role=DBManager
C:\> MySQLInstallerConsole install server;5.6.25;x64 -silent
```

^ で区切られた追加の構成ブロックを適合させる例を示します:

```
C:\> MySQLInstallerConsole install server;5.6.25;x64.*:type=config;openfirewall=true; ^
      generallog=true;binlog=true;serverid=3306;enable_tcpip=true;port=3306;rootpasswd=pass; ^
```

```
installid="C:\MySQL\MySQL Server 5.6":type=user;datadir="C:\MySQL\data";username=foo;password=bar;role=DBManager
```

- リスト

利用可能なすべての MySQL 製品を検索できるインタラクティブなコンソールをリストします。
[MySQLInstallerConsole list](#) を実行してコンソールを起動し、検索する部分文字列を入力します。

```
C:\> MySQLInstallerConsole list
```

- `modify [product1:-removelist][+addlist] [product2:-removelist][+addlist] [...]`

以前にインストールされた MySQL 製品の機能を変更または表示します。製品の機能を表示するには、コマンドに `product` キーワードを追加します。次に例を示します:

```
C:\> MySQLInstallerConsole modify server
```

このコマンドの構文オプション:

`-silent` 確認プロンプトを無効にします。

```
C:\> MySQLInstallerConsole modify server:+documentation
```

```
C:\> MySQLInstallerConsole modify server:-debug
```

- `remove [product1] [product2] [...]`

システム上から 1 つまたは複数の製品を削除します。次のようなスイッチと構文オプションがあります。

* * を渡して、すべての MySQL 製品を削除します。

`-continue` エラーが発生した場合でも操作を続行します。

`-silent` 確認プロンプトを無効にします。

```
C:\> MySQLInstallerConsole remove *
```

```
C:\> MySQLInstallerConsole remove server
```

- `status`

システムにインストールされている MySQL 製品の簡単な概要を提供します。情報には、製品名とバージョン、アーキテクチャー、インストール日、およびインストールの場所が含まれます。

```
C:\> MySQLInstallerConsole status
```

- `update`

最新の MySQL 製品カタログをシステムにダウンロードします。成功すると、次回 [MySQLInstaller](#) または [MySQLInstallerConsole](#) が実行されたときにカタログが適用されます。

```
C:\> MySQLInstallerConsole update
```

注記

「Automatic Catalog Update」 GUI オプションはこのコマンドを Windows のタスクスケジューラから実行します。

- `upgrade [product1:version] [product2:version] [...]`

システム上の 1 つまたは複数の製品をアップグレードします。次のような構文オプションがあります。

* * を渡してすべての製品を最新バージョンにアップグレードするか、特定の製品を渡します。

! MySQL 製品を最新バージョンにアップグレードするには、バージョン番号として ! を渡します。

`-silent` 確認プロンプトを無効にします。

```
C:\> MySQLInstallerConsole upgrade *
```

```
C:\> MySQLInstallerConsole upgrade workbench:8.0.21
```

```
C:\> MySQLInstallerConsole upgrade workbench!  
C:\> MySQLInstallerConsole upgrade workbench:8.0.21 visual:1.2.9
```

2.3.4 noinstall ZIP アーカイブを使用した Microsoft Windows への MySQL のインストール

非インストールパッケージからインストールするユーザーは、このセクションの説明に従って手動で MySQL をインストールできます。ZIP アーカイブパッケージから MySQL をインストールするプロセスは次のとおりです:

1. メインアーカイブを目的のインストールディレクトリに抽出

オプション: MySQL ベンチマークおよびテストスイートを実行する予定の場合は、デバッグテストアーカイブも抽出

2. オプションファイルを作成する
3. MySQL のサーバータイプを選択する
4. MySQL の初期化
5. MySQL Server を起動する
6. デフォルトのユーザーアカウントをセキュアにする

この手順はこのあとのセクションで説明します。

2.3.4.1 インストールアーカイブを抽出する

MySQL を手動でインストールするには、次の手順に従います。

1. 以前のバージョンからアップグレードする場合にはアップグレードの手順を始める前に [セクション 2.11.10 「Windows 上の MySQL をアップグレードする」](#) を参照してください。
2. 管理者権限を持つユーザーとしてログインしていることを確認します。
3. インストールの場所を選択します。通常、MySQL Server は `C:\mysql` にインストールされます。MySQL を `C:\mysql` にインストールしない場合、起動時あるいはオプションファイルでインストールディレクトリへのパスを指定する必要があります。 [セクション 2.3.4.2 「オプションファイルの作成」](#) を参照してください。

注記

MySQL Installer が MySQL を `C:\Program Files\MySQL` にインストールします。

4. 優先ファイル圧縮ツールを使用して、選択したインストール場所にインストールアーカイブを抽出します。ツールによっては、選択したインストール場所のフォルダにアーカイブを抽出します。その場合、そのサブフォルダの内容を、選択したインストール場所に移動します。

2.3.4.2 オプションファイルの作成

サーバーの起動時にスタートアップオプションを指定する必要がある場合には、それらをコマンド行で指示するかオプションファイルに配置します。サーバーの起動時に常に使用するオプションは、オプションファイルを使用して MySQL の構成を指定すると非常に便利です。これは次の状況に特に当てはまります。

- インストールまたはデータディレクトリの場所が、デフォルトの場所 (`C:\Program Files\MySQL\MySQL Server 8.0` および `C:\Program Files\MySQL\MySQL Server 8.0\data`) とは異なり場合。
- メモリー、キャッシュ、InnoDB 構成情報などのサーバー設定を調整する必要がある場合。

MySQL Server が Windows 上で起動する際、Windows ディレクトリ `C:\`、および MySQL インストールディレクトリなど、いくつかの場所にあるオプションファイルを検索します (場所の完全なリストについては、[セクション 4.2.2.2 「オプションファイルの使用」](#) を参照してください)。通常、Windows ディレクトリの名前は、`C:\WINDOWS` のようになります。次のコマンドを使用して `WINDIR` 環境変数の値から正確な場所を割り出すことができます。

```
C:\> echo %WINDIR%
```

MySQL は、各場所の `my.ini` ファイル内のオプションを最初に検索し、次に `my.cnf` ファイルを調べます。ただし、混乱を避けるためにはファイルを 1 つだけ使用するのが最善です。お客様の PC が、`C:` がブートドライブではないブートローダーを使用している場合、`my.ini` ファイルしか使用できません。どちらのオプションファイルを使用するにしろ、プレーンテキストファイルでなければなりません。

注記

MySQL Installer を使用して MySQL Server をインストールすると、`my.ini` がデフォルトの場所に作成され、MySQL Installer を実行しているユーザーには、この新しい `my.ini` ファイルに対する完全な権限が付与されます。

つまり、必ず MySQL Server ユーザーが `my.ini` ファイルを読み取る権限があるようにしてください。

MySQL 配布に含まれている参考例のオプションファイルを使用することもできます。詳細は [セクション5.1.2「サーバー構成のデフォルト値」](#) を参照してください。

オプションファイルは、ノートパッドなどのテキストエディタで作成および変更ができます。たとえば、MySQL を `E:\mysql` にインストールし、データディレクトリが `E:\mydata\data` にある場合は、`[mysqld]` セクションを含むオプションファイルを作成して、`basedir` および `datadir` オプションの値を指定できます。

```
[mysqld]
# set basedir to your installation path
basedir=E:/mysql
# set datadir to the location of your data directory
datadir=E:/mydata/data
```

Microsoft Windows のパス名は、オプションファイル内でバックスラッシュではなく (フォワード) スラッシュを使用して指定されます。バックスラッシュを使用する場合は、2 つ使用します。

```
[mysqld]
# set basedir to your installation path
basedir=E:\mysql
# set datadir to the location of your data directory
datadir=E:\mydata\data
```

オプションファイル値でのバックスラッシュの使用に関するルールは [セクション4.2.2.2「オプションファイルの使用」](#) にあります。

ZIP アーカイブには、`data` ディレクトリは含まれません。データディレクトリを作成し、`mysql` システムデータベースにテーブルを移入して MySQL インストールを初期化するには、`--initialize` または `--initialize-insecure` のいずれかを使用して MySQL を初期化します。追加情報については [セクション2.10.1「データディレクトリの初期化」](#) を参照してください。

データディレクトリを異なる場所で使用したい場合、`data` ディレクトリの内容全体を新しい場所にコピーしてください。たとえば、代わりに `E:\mydata` をデータディレクトリとして使用する場合は、次の 2 つのを行う必要があります。

1. `data` ディレクトリおよびその内容すべてを、デフォルトの場所 (`C:\Program Files\MySQL\MySQL Server 8.0\data` など) から `E:\mydata` に移動します。
2. サーバーの起動時に常に新しいデータディレクトリの場所を指定するには `--datadir` オプションを使用します。

2.3.4.3 MySQL サーバータイプの選択

次の表は、Windows で使用できる MySQL 8.0 のサーバーを示しています。

バイナリ	説明
<code>mysqld</code>	名前付きパイプのサポートがある最適化バイナリ
<code>mysqld-debug</code>	<code>mysqld</code> と同様ですが、完全なデバッグと自動メモリー割り当ての確認を行なってコンパイルされています

以前のバイナリはすべて最新の Intel プロセッサに最適化されていますが、Intel i386 クラスあるいはそれ以上のプロセッサで動作するはずです。

配布内の各サーバーは、同じストレージエンジンセットをサポートします。SHOW ENGINES ステートメントは、指定されたサーバーがサポートするエンジンを表示します。

すべての Windows MySQL 8.0 Server は、データベースディレクトリのシンボリックリンクをサポートします。

MySQL はすべての Windows プラットフォームで TCP/IP をサポートしています。named_pipe システム変数を有効にしてサーバーを起動した場合、Windows 上の MySQL サーバーでは名前付きパイプもサポートされます。名前付きパイプの使用時に MySQL サーバーの停止で問題が発生したユーザーがいるため、この変数を明示的に有効にする必要があります。多くの Windows 構成において、名前付きパイプは TCP/IP より遅いため、デフォルトではプラットフォームにかかわらず TCP/IP が使用されます。

2.3.4.4 データディレクトリの初期化

noinstall パッケージを使用して MySQL をインストールした場合、データディレクトリは含まれません。データディレクトリを初期化するには、[セクション 2.10.1 「データディレクトリの初期化」](#) の手順を使用します。

2.3.4.5 サーバーをはじめて起動する

このセクションでは MySQL サーバーの起動に関する一般的な概要を説明します。次の数セクションでは、MySQL サーバーのコマンド行あるいは Windows のサービスとしての起動に関して、より具体的な情報を提供します。

ここでの情報は、主に noinstall バージョンを使用して MySQL をインストールした場合、または MySQL Installer ではなく MySQL を手動で構成およびテストする場合に適用されます。

これらのセクションの例では、MySQL がデフォルトの場所の C:\Program Files\MySQL\MySQL Server 8.0 にインストールされていることを前提とします。異なる場所に MySQL をインストールしている場合には例に示したパス名を調整してください。

クライアントには 2 つのオプションがあります。TCP/IP が使用でき、サーバーが名前付きパイプの接続をサポートしている場合には名前付きパイプも使用できます。

MySQL for Windows では、サーバーが shared_memory システム変数を有効にして起動されている場合、共有メモリー接続もサポートされます。クライアントは --protocol=MEMORY オプションを使用して共有メモリーで接続できます。

どのサーバーバイナリを実行するかについては、[セクション 2.3.4.3 「MySQL サーバータイプの選択」](#) を参照してください。

テストはコンソールウィンドウ (あるいは「DOS ウィンドウ」) のコマンドプロンプトで行います。これにより、ウィンドウにサーバーのステータスに関するメッセージが表示されますので、容易に確認できます。構成に何か問題があった場合には、これらのメッセージにより問題の特定と修正が容易になります。

注記

MySQL を起動する前に、データベースを初期化する必要があります。初期化プロセスの詳細は、[セクション 2.10.1 「データディレクトリの初期化」](#) を参照してください。

サーバーを起動するには、次のコマンドを入力します。

```
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqld" --console
```

InnoDB サポートを含むサーバーでは、次のようなメッセージがサーバーの起動時に表示されます (パス名とサイズは異なる場合があります)。

```
InnoDB: The first specified datafile c:\ibdata\ibdata1 did not exist:
InnoDB: a new database to be created!
InnoDB: Setting file c:\ibdata\ibdata1 size to 209715200
InnoDB: Database physically writes the file full: wait...
InnoDB: Log file c:\iblogs\ib_logfile0 did not exist: new to be created
```

```
InnoDB: Setting log file c:\blogs\ib_logfile0 size to 31457280
InnoDB: Log file c:\blogs\ib_logfile1 did not exist: new to be created
InnoDB: Setting log file c:\blogs\ib_logfile1 size to 31457280
InnoDB: Log file c:\blogs\ib_logfile2 did not exist: new to be created
InnoDB: Setting log file c:\blogs\ib_logfile2 size to 31457280
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
InnoDB: creating foreign key constraint system tables
InnoDB: foreign key constraint system tables created
011024 10:58:25 InnoDB: Started
```

サーバーが起動シーケンスを終了すると、次のようなメッセージが表示されます。このメッセージが表示されると、サーバーのクライアント接続の用意が整ったことを意味します。

```
mysqld: ready for connections
Version: '8.0.29' socket: '' port: 3306
```

サーバーは、診断を生成して出力をコンソールに書き込み続けます。クライアントプログラムを実行する新しいコンソールウィンドウを開くことができます。

`--console` オプションを省略すると、サーバーは診断の出力をデータディレクトリ (デフォルトでは `C:\Program Files\MySQL\MySQL Server 8.0\data`) のエラーログに出カします。エラーログは、`.err` 拡張子の付いたファイルで、`--log-error` オプションを使用して設定できます。

注記

MySQL 付与テーブルの初期 `root` アカウントにはパスワードがありません。サーバーの起動後、[セクション2.10.4「初期 MySQL アカウントの保護」](#) の手順を使用してパスワードを設定する必要があります。

2.3.4.6 Windows のコマンド行からの MySQL の起動

MySQL Server はコマンド行から手動で起動できます。これは Windows のどのバージョンでもできます。

コマンド行から `mysqld` サーバーを起動するには、コンソールウィンドウ (あるいは「DOS ウィンドウ」) を開け、次のコマンドを入力します。

```
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqld"
```

`mysqld` へのパスはお客様のシステムの MySQL のインストール場所によって異なる場合があります。

MySQL Server を停止するには次のコマンドを実行します。

```
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqladmin" -u root shutdown
```

注記

MySQL の `root` ユーザーアカウントにパスワードが設定されている場合は、`mysqladmin` を `-p` オプションを使って起動し、要求されたらパスワードを入力する必要があります。

このコマンドは MySQL 管理ユーティリティ `mysqladmin` を起動してサーバーに接続し、サーバーにシャットダウンを命令します。そのコマンドは MySQL `root` ユーザーとして接続します。これは MySQL の許可システムのデフォルトの管理アカウントです。

注記

MySQL 権限付与システムのユーザーは、Microsoft Windows のオペレーティングシステムユーザーから完全に独立しています。

`mysqld` が起動しない場合、エラーログを確認してサーバーがその問題の原因を示すメッセージを書き込んでいないか確認します。デフォルトでは、エラーログは `C:\Program Files\MySQL\MySQL Server 8.0\data` ディレクトリにあります。サフィクス `.err` の付いたファイルです。または、`--log-error` オプションで渡して指定できます。または、`--console` オプションを使用してサーバーを起動することもできます。この場合、問題の解決に役立つ有用な情報がサーバーの画面に表示されることがあります。

最後の選択肢は、`mysqld` を `--standalone` および `--debug` オプションで起動することです。この場合、`mysqld` がログファイル `C:\mysqld.trace` に書き出し、その中に `mysqld` が起動しない理由が含まれているはずですが。セクション 5.9.4 「[DEBUG パッケージ](#)」を参照してください。

`mysqld --verbose --help` を使用して、`mysqld` がサポートするすべてのオプションを表示します。

2.3.4.7 MySQL ツールの PATH をカスタマイズする

警告

システムの `PATH` を手動で編集するには最大限の注意が必要です。既存の `PATH` の値の一部でも間違っただけで削除したり変更したりすると、誤動作を引き起こしたりシステムが使用できなくなったりする場合があります。

MySQL プログラムの起動を簡単にするために、MySQL の `bin` ディレクトリのパス名を Windows システムの `PATH` 環境変数に追加できます。

- Windows デスクトップで、「マイコンピュータ」アイコンを右クリックして「プロパティ」を選択します。
- 次に、表示された「システムのプロパティ」メニューから「詳細設定」タブを選択し、「環境変数」ボタンをクリックします。
- 「システム環境変数」で、「Path」を選択し、次に「編集」ボタンをクリックします。「システム変数の編集」のダイアログが表示されます。
- 「変数値」と示されたスペースに表示されたテキストの最後にカーソルを持って行きます。（「End」キーを使用して、カーソルが確実にそのスペースのテキストのいちばん後ろにあるようにします）。次に、MySQL `bin` ディレクトリの完全なパス名を入力します（`C:\Program Files\MySQL\MySQL Server 8.0\bin` など）。

注記

このフィールドにあるほかの値とこのパスを区切るために、セミコロンが必要です。

開いているダイアログがすべて閉じるまで、「OK」をクリックしてこのダイアログとその他のダイアログを順番に閉じます。これで、開いた新しいコマンドシェルで新しい `PATH` 値を使用できるようになり、システム上の任意のディレクトリから DOS プロンプトに名前を入力することで、パスを指定せずに MySQL 実行可能プログラムを起動できるようになります。これにはサーバー、`mysql` クライアント、および `mysqladmin` と `mysqldump` などのすべての MySQL コマンド行ユーティリティーが含まれています。

同じマシンで複数の MySQL サーバーを動作させている場合には、MySQL `bin` ディレクトリを Windows `PATH` に追加しないでください。

2.3.4.8 Windows のサービスとして MySQL を起動する

Windows で MySQL を実行する場合、Windows の起動および停止時に MySQL が自動的に起動および停止するように、Windows サービスとしてインストールすることが推奨されています。MySQL Server をサービスとしてインストールすると、`NET` コマンドを使用してコマンド行から、あるいはグラフィカル `Services` ユーティリティーで管理することもできます。一般に、MySQL を Windows サービスとしてインストールするときは、管理者権限のあるアカウントを使ってログインするようにしてください。

`Services` ユーティリティー (Windows `Service Control Manager`) は、Windows コントロールパネルにあります。競合を避けるために、サーバーのインストールまたはコマンド行からの削除操作の際には、`Services` ユーティリティーを開くことを推奨します。

サービスのインストール

現在サーバーを実行中の場合は、MySQL を Windows のサービスとしてインストールする前に次のコマンドを使用して停止してください。

```
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqladmin"
-u root shutdown
```

注記

MySQL の `root` ユーザーアカウントにパスワードが設定されている場合は、`mysqladmin` を `-p` オプションを使って起動し、要求されたらパスワードを入力する必要があります。

このコマンドは MySQL 管理ユーティリティー `mysqladmin` を起動してサーバーに接続し、サーバーにシャットダウンを命令します。そのコマンドは MySQL `root` ユーザーとして接続します。これは MySQL の許可システムのデフォルトの管理アカウントです。

注記

MySQL 権限付与システムのユーザーは、Windows のオペレーティングシステムユーザーから完全に独立しています。

サーバーをこのコマンドを使用してサービスとしてインストールします。

```
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqld" --install
```

サービスのインストールコマンドはサーバーは起動しません。それについてはこのセクションで後述します。

MySQL プログラムの起動を簡単にするために、MySQL の `bin` ディレクトリのパス名を Windows システムの `PATH` 環境変数に追加できます。

- Windows デスクトップで、「マイコンピュータ」アイコンを右クリックして「プロパティ」を選択します。
- 次に、表示された「システムのプロパティ」メニューから「詳細設定」タブを選択し、「環境変数」ボタンをクリックします。
- 「システム環境変数」で、「Path」を選択し、次に「編集」ボタンをクリックします。「システム変数の編集」のダイアログが表示されます。
- 「変数値」と示されたスペースに表示されたテキストの最後にカーソルを持って行きます。（「End」キーを使用して、カーソルが確実にそのスペースのテキストのいちばん後ろにあるようにします）。次に、MySQL `bin` ディレクトリの完全なパス名を入力します（`C:\Program Files\MySQL\MySQL Server 8.0\bin` など）。このフィールドにあるほかの値とこのパスを区切るために、セミコロンが必要です。開いているダイアログがすべて閉じるまで、「OK」をクリックしてこのダイアログとその他のダイアログを順番に閉じます。この段階で、システムのどのディレクトリからでも、DOS プロンプトにパスを指定せずに MySQL の実行プログラム名を入力して、すべての MySQL 実行プログラムを呼び出すことができます。これにはサーバー、`mysql` クライアント、および `mysqladmin` と `mysqldump` などのすべての MySQL コマンド行ユーティリティーが含まれています。

同じマシンで複数の MySQL サーバーを動作させている場合には、MySQL `bin` ディレクトリを Windows `PATH` に追加しないでください。

警告

システムの `PATH` を手動で編集する際には最大限の注意が必要です。既存の `PATH` の値の一部でも間違っただけで削除したり変更したりすると、誤動作を引き起こしたりシステムが使用できなくなったりする場合があります。

サービスをインストールする際に、次の追加引数を使用できます。

- `--install` オプションの直後にサービス名を指定できます。デフォルトのサービス名は `MySQL` です。
- サービス名を指定すると、そのあとに 1 つのオプションを指定できます。規則では、`--defaults-file=file_name` でオプションファイル名を指定し、サーバーが起動時にそこからオプションを読み取ります。
`--defaults-file` 以外に 1 つのオプションを使用することは可能ですが、お勧めできません。`--defaults-file` を使用すると、指名したオプションファイルに複数のスタートアップオプションを指定できるので、より柔軟です。
- サービス名のあとに `--local-service` オプションも指定できます。これにより、システム権限の制限付き `LocalService` Windows アカウントを使用してサーバーを起動できます。`--defaults-file` および `--local-service` の両方がサービス名のあとに指定される場合、どのような順序でもかまいません。

MySQL Server を Windows のサービスとしてインストールした場合、サーバーが使用するサービス名およびオプションファイルは次のルールで決められます。

- service-installation コマンドで、`--install` オプションのあとにサービス名またはデフォルトのサービス名 (MySQL) が指定されていない場合、サーバーは MySQL のサービス名を使用し、標準オプションファイル内の `[mysqld]` グループからオプションを読み取ります。
- サービスインストールのコマンドで MySQL 以外のサービス名を `--install` オプションのあとに指定した場合、サーバーはそのサービス名を使用します。標準のオプションファイルの `[mysqld]` グループと、サービスと名前が同じグループからオプションを読み取ります。これにより、`[mysqld]` グループをすべての MySQL サービスで使用するべきオプション用に、サービス名の付いたオプショングループをそのサービス名でインストールされたサーバー用に使用できます。
- サービスインストールのコマンドでサービス名のあとに `--defaults-file` オプションを指定した場合、サーバーは前の項目で説明したのと同じ方法でオプションを読み取りますが、名前付きのファイルからのみオプションを読み取り、標準のオプションファイルは無視します。

さらに複雑な例として、次のコマンドがあります。

```
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqld"
--install MySQL --defaults-file=C:\my-opts.cnf
```

ここでは、`--install` オプションのあとにデフォルトのサービス名 (MySQL) が指定されています。`--defaults-file` オプションが指定されていないければ、このコマンドによりサーバーは標準のオプションファイルから `[mysqld]` グループを読み取ります。しかし、`--defaults-file` オプションが指定されているので、サーバーは `[mysqld]` オプショングループのオプションを、指名されたファイルのみから読み取ります。

注記

Windows では、サーバーが `--defaults-file` および `--install` オプションで起動される場合、`--install` が最初でなければなりません。それ以外の場合、`mysqld.exe` は MySQL サーバーを起動しようとします。

MySQL サービスを実行する前に、Windows の [Services](#) ユーティリティで Start パラメータとしてオプションを指定することもできます。

最後に、MySQL サービスを開始する前に、サービスを実行するオペレーティングシステムユーザーのユーザー変数 `%TEMP%` および `%TMP%` (設定されている場合は `%TMPDIR%`) が、ユーザーが書き込みアクセス権を持つフォルダを指していることを確認します。MySQL サービスを実行するためのデフォルトユーザーは `LocalSystem` で、`%TEMP%` および `%TMP%` のデフォルト値は `C:\Windows\Temp` で、`LocalSystem` ディレクトリにはデフォルトで書き込みアクセス権があります。ただし、そのデフォルト設定に変更があった場合 (たとえば、サービスを実行するユーザーや前述のユーザー変数への変更、または `--tmpdir` オプションを使用して一時ディレクトリを別の場所に配置した場合)、一時ディレクトリへの書き込みアクセス権が適切なユーザーに付与されていないため、MySQL サービスの実行に失敗する可能性があります。

サービスの開始

MySQL サーバーインスタンスがサービスとしてインストールされると、Windows は常にサービスを自動的に起動します。サービスは、[Services](#) ユーティリティからすぐに起動することも、`sc start mysql_service_name` または `NET START mysql_service_name` コマンドを使用して起動することもできます。`SC` および `NET` コマンドでは、大/小文字は区別されません。

サービスとして起動する場合、`mysqld` はコンソールウィンドウにアクセスできないので、メッセージは表示されません。`mysqld` が起動しない場合、サーバーがエラーログにその問題の原因を知らせるメッセージを書き込んでいないか確認します。エラーログは MySQL データディレクトリ (`C:\Program Files\MySQL\MySQL Server 8.0\data` など) にあります。`.err` のサフィクスが付いたファイルです。

MySQL Server をサービスとしてインストールした場合、サービスを実行しているときに Windows がシャットダウンすると Windows が自動的にサービスを停止します。サーバーは、[Services](#) ユーティリティ、`sc stop mysql_service_name` コマンド、`NET START mysql_service_name` コマンドまたは `mysqladmin shutdown` コマンドを使用して手動で停止することもできます。

サービスがブートプロセスで自動的に実行されないように、サーバーをマニュアルサービスとしてインストールすることもできます。これには `--install` オプションではなく `--install-manual` オプションを使用します。

```
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqld" --install-manual
```


サービスの削除

サービスとしてインストールされているサーバーを削除するには、まず `SC STOP mysqlid_service_name` または `NET STOP mysqlid_service_name` を実行して停止します (実行中の場合)。次に、`SC DELETE mysqlid_service_name` を使用して削除します:

```
C:\> SC DELETE mysql
```

または、`mysqlid --remove` オプションを使用してサービスを削除します。

```
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqlid" --remove
```

`mysqlid` をサービスとして実行していない場合は、コマンド行で起動できます。その手順は、[セクション 2.3.4.6 「Windows のコマンド行からの MySQL の起動」](#) を参照してください。

インストール中に問題が発生した場合は、[セクション 2.3.5 「Microsoft Windows MySQL Server インストールのトラブルシューティング」](#) を参照してください。

Windows サービスの停止または削除の詳細は、[セクション 5.8.2.2 「Windows サービスとして複数の MySQL インスタンスの起動」](#) を参照してください。

2.3.4.9 MySQL インストールのテスト

次のコマンドのいずれかを実行して、MySQL Server が動作しているかテストできます。

```
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqlshow"  
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqlshow" -u root mysql  
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqladmin" version status proc  
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysql" test
```

`mysqlid` のクライアントプログラムから TCP/IP 接続への反応が鈍い場合、おそらくの DNS に問題があります。この場合、`skip_name_resolve` システム変数を有効にして `mysqlid` を起動し、MySQL 付与テーブルの `Host` カラムで `localhost` および IP アドレスのみを使用します。(IP アドレスを指定するアカウントが存在することを確認してください。存在しない場合、接続できない可能性があります。)

`--pipe` あるいは `--protocol=PIPE` オプションを指定するか、あるいは `.` (ピリオド) をホスト名として指定すると、MySQL クライアントに TCP/IP ではなく名前付けパイプ接続を強制的に使用させることができます。デフォルトのパイプ名を使用しない場合には `--socket` オプションを使用してパイプ名を指定します。

`root` アカウントのパスワードを設定した場合、匿名アカウントを削除した場合、または新規ユーザーアカウントを作成した場合に MySQL Server に接続するには、前述のコマンドに適切な `-u` および `-p` オプションを使用しなければなりません。[セクション 4.2.4 「コマンドオプションを使用した MySQL Server への接続」](#) を参照してください。

`mysqlshow` に関する詳細は、[セクション 4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」](#) を参照してください。

2.3.5 Microsoft Windows MySQL Server インストールのトラブルシューティング

MySQL をインストールして最初に起動する際に、エラーが発生して MySQL サーバーが起動できない場合があります。このセクションは、エラーの診断と修正に役立ちます。

サーバーの問題のトラブルシューティングを行う際の、最初のリソースは [エラーログ](#) です。MySQL Server は、サーバーが起動しない原因となるエラーに関する情報を記録するのにエラーログを使用しています。エラーログは、`my.ini` ファイルで指定された [データディレクトリ](#) にあります。データディレクトリのデフォルトの場所は、Windows 7 および Windows Server 2008 では `C:\Program Files\MySQL\MySQL Server 8.0\data`、または `C:\ProgramData\MySQL` です。`C:\ProgramData` ディレクトリはデフォルトでは非表示です。ディレクトリと内容を表示するには、フォルダオプションを変更する必要があります。エラーログとその内容の理解については、[セクション 5.4.2 「エラーログ」](#) を参照してください。

起こりうるエラーについては、MySQL サービスの起動中に表示されるコンソールメッセージも調べてください。`mysqlid` をサービスとしてインストールした後、コマンドラインから `SC START mysqlid_service_name` または `NET START mysqlid_service_name` コマンドを使用して、MySQL サーバーのサービスとしての起動に関するエラーメッセージを表示します。[セクション 2.3.4.8 「Windows のサービスとして MySQL を起動する」](#) を参照してください。

MySQL をインストールしてサーバーを最初に起動する際によく生じる、その他の一般的なエラーメッセージを次の例に示します。

- MySQL Server が `mysql` 権限データベースまたはその他の重要なファイルを見つけられない場合、次のメッセージを表示します。

```
System error 1067 has occurred.  
Fatal error: Can't open and lock privilege tables:  
Table 'mysql.user' doesn't exist
```

これらのメッセージは、MySQL のベースディレクトリまたはデータディレクトリが、デフォルトの場所 (それぞれ `C:\Program Files\MySQL\MySQL Server 8.0` および `C:\Program Files\MySQL\MySQL Server 8.0\data`) と異なる場所にインストールされている場合によく生じます。

この状況は、MySQL をアップグレードして新しい場所にインストールしたが、構成ファイルが新しい場所を反映するように更新されていない場合に生じます。さらに、新旧の構成ファイルが対立している場合もあります。MySQL をアップグレードする際は必ず旧構成ファイルを削除するか名前を変更してください。

MySQL を `C:\Program Files\MySQL\MySQL Server 8.0` 以外のディレクトリにインストールした場合は、構成 (`my.ini`) ファイルを使用して MySQL Server がそのことを認識できるようにしてください。 `my.ini` ファイルを Windows ディレクトリ (通常 `C:\WINDOWS`) に置きます。 `WINDIR` 環境変数の値から正確な場所を割り出すためには、コマンドプロンプトから次のコマンドを発行します。

```
C:\> echo %WINDIR%
```

オプションファイルはノートパッドなどのテキストエディタで作成および変更できます。たとえば、MySQL が `E:\mysql` にインストールされていて、データディレクトリが `D:\MySQLdata` にある場合は、オプションファイルを作成して `[mysqld]` セクションを設定し、 `basedir` および `datadir` オプションの値を指定できます。

```
[mysqld]  
# set basedir to your installation path  
basedir=E:/mysql  
# set datadir to the location of your data directory  
datadir=D:/MySQLdata
```

Microsoft Windows のパス名は、オプションファイル内でバックスラッシュではなく (フォワード) スラッシュを使用して指定されます。バックスラッシュを使用する場合は、2 つ使用します。

```
[mysqld]  
# set basedir to your installation path  
basedir=C:\\Program Files\\MySQL\\MySQL Server 8.0  
# set datadir to the location of your data directory  
datadir=D:\\MySQLdata
```

オプションファイル値でのバックスラッシュの使用に関するルールは [セクション4.2.2.2「オプションファイルの使用」](#) にあります。

MySQL 構成ファイルの `datadir` の値を変更する場合は、MySQL Server を再起動する前に既存の MySQL データディレクトリの内容を移動する必要があります。

[セクション2.3.4.2「オプションファイルの作成」](#) を参照してください。

- まず既存の MySQL サービスを停止して削除してから MySQL Installer を使用して MySQL をインストールせずに、MySQL を再インストールまたはアップグレードすると、次のエラーが生じる場合があります。

```
Error: Cannot create Windows service for MySql. Error: 0
```

これは Configuration Wizard がサービスをインストールしようとしたときに既存のサービスが同じ名前が存在する場合に発生します。

この問題の解決方法の 1 つは、Configuration Wizard を使用する際に `mysql` 以外のサービス名を選択することです。これにより、新しいサービスが正しくインストールされ、古いサービスはそのままにできます。これは特に問題はありませんが、使用しない古いサービスは削除したほうがよいでしょう。

古い `mysql` サービスを完全に削除するには、管理者権限を持つユーザーとして、コマンド行で次のコマンドを実行します。

```
C:\> SC DELETE mysql
[SC] DeleteService SUCCESS
```

ご使用のバージョンの Windows で SC ユーティリティを使用できない場合は、<http://www.microsoft.com/windows2000/techinfo/reskit/tools/existing/delsrv-o.asp> から `delsrv` ユーティリティをダウンロードし、`delsrv mysql` 構文を使用します。

2.3.6 Windows でのインストール後の手順

このセクションで説明する次のようなほとんどのタスクを実行する GUI ツールが存在します:

- **MySQL Installer:** MySQL 製品のインストールとアップグレードに使用されます。
- **MySQL Workbench:** MySQL サーバーを管理し、SQL ステートメントを編集します。

必要に応じて、データディレクトリを初期化し、MySQL 付与テーブルを作成します。MySQL Installer によって実行される Windows インストール操作によって、データディレクトリが自動的に初期化されます。ZIP アーカイブパッケージからのインストールの場合は、[セクション2.10.1「データディレクトリの初期化」](#)の説明に従ってデータディレクトリを初期化します。

パスワードについては、MySQL Installer を使用して MySQL をインストールした場合、初期 `root` アカウントにすでにパスワードが割り当てられている可能性があります。([セクション2.3.3「MySQL Installer for Windows」](#) を参照してください。) そうでない場合には、[セクション2.10.4「初期 MySQL アカウントの保護」](#)にあるパスワード割り当て手順を使用します。

パスワードを割り当てる前に、いくつかのクライアントプログラムを実行して、サーバーに接続できること、およびサーバーが正しく動作していることを確認することをお勧めします。サーバーが実行されていることを確認します ([セクション2.3.4.5「サーバーをはじめて起動する」](#) を参照)。Windows の起動時に自動的に実行される MySQL サービスを設定することもできます ([セクション2.3.4.8「Windows のサービスとして MySQL を起動する」](#) を参照)。

これらの手順では、現在の場所が MySQL インストールディレクトリであり、ここで使用する MySQL プログラムを含む `bin` サブディレクトリがあることを前提としています。そうでない場合は、コマンドパス名を適宜調整します。

MySQL Installer ([セクション2.3.3「MySQL Installer for Windows」](#) を参照) を使用して MySQL をインストールした場合、デフォルトのインストールディレクトリは `C:\Program Files\MySQL\MySQL Server 8.0` です:

```
C:\> cd "C:\Program Files\MySQL\MySQL Server 8.0"
```

ZIP アーカイブからインストールするための一般的なインストール場所は、`C:\mysql` です:

```
C:\> cd C:\mysql
```

または、`bin` ディレクトリを `PATH` 環境変数設定に追加します。これにより、コマンドインタプリタは MySQL プログラムを適切に検索できるため、パス名ではなく名前のみを入力してプログラムを実行できます。 [セクション2.3.4.7「MySQL ツールの PATH をカスタマイズする」](#) を参照してください。

サーバーを実行した状態で、次のコマンドを発行して、サーバーから情報を取得できることを確認します。出力は次のようになります。

どのようなデータベースが存在するかを表示するには、`mysqlshow` を使用します。

```
C:\> bin\mysqlshow
+-----+
| Databases |
+-----+
| information_schema |
| mysql            |
| performance_schema |
| sys              |
+-----+
```

インストールされているデータベースのリストは異なる場合がありますが、常に `mysql` および `information_schema` 以上が含まれています。

前述のコマンド (および `mysql` などのその他の MySQL プログラムのコマンド) は、正しい MySQL アカウントが存在しないと機能しない場合があります。たとえば、プログラムがエラーで終了する場合やすべてのデータベースを表示

できない場合があります。MySQL Installer を使用して MySQL をインストールすると、指定したパスワードで `root` ユーザーが自動的に作成されます。この場合、`-u root` および `-p` オプションを使用してください。(初期 MySQL アカウントをすでに保護している場合は、これらのオプションを使用する必要があります。) `-p` では、クライアントプログラムによって `root` パスワードの入力が求められます。例:

```
C:\> bin\mysqlshow -u root -p
Enter password: (enter root password here)
+-----+
| Databases |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
```

データベース名を指定すると、`mysqlshow` はそのデータベース内のテーブルのリストを表示します。

```
C:\> bin\mysqlshow mysql
Database: mysql
+-----+
| Tables |
+-----+
| columns_priv |
| component |
| db |
| default_roles |
| engine_cost |
| func |
| general_log |
| global_grants |
| gtid_executed |
| help_category |
| help_keyword |
| help_relation |
| help_topic |
| innodb_index_stats |
| innodb_table_stats |
| ndb_binlog_index |
| password_history |
| plugin |
| procs_priv |
| proxies_priv |
| role_edges |
| server_cost |
| servers |
| slave_master_info |
| slave_relay_log_info |
| slave_worker_info |
| slow_log |
| tables_priv |
| time_zone |
| time_zone_leap_second |
| time_zone_name |
| time_zone_transition |
| time_zone_transition_type |
| user |
+-----+
```

`mysql` プログラムを使用して、`mysql` データベース内のテーブルから情報を選択します。

```
C:\> bin\mysql -e "SELECT User, Host, plugin FROM mysql.user" mysql
+-----+-----+-----+
| User | Host | plugin |
+-----+-----+-----+
| root | localhost | caching_sha2_password |
+-----+-----+-----+
```

`mysql` および `mysqlshow` の詳細は、[セクション4.5.1「mysql — MySQL コマンドラインクライアント」](#) および [セクション4.5.7「mysqlshow — データベース、テーブル、およびカラム情報の表示」](#) を参照してください。

2.3.7 Windows プラットフォームの制限事項

Windows プラットフォームでの MySQL の使用には、次の制限が適用されます:

- プロセスメモリー

Windows 32 ビットプラットフォームでは、デフォルトで、MySQL などの単一プロセス内で 2G バイトを超える RAM を使用できません。これは、Windows 32 ビットでの物理アドレスの制限が 4G バイトであり、Windows 内のデフォルト設定では、カーネル (2G バイト) とユーザー/アプリケーション (2G バイト) とに仮想アドレス空間を分割するためです。

Windows の一部のバージョンには、カーネルアプリケーションを減らすことによってより大きなアプリケーションに対応するブート時設定があります。または、2G バイト以上を使用するには、64 ビットバージョンの Windows を使用します。

- ファイルシステムエイリアス

MyISAM テーブルの使用時には、Windows でエイリアスを使用して、別のボリューム上のデータファイルにリンクしてからメインの MySQL **datadir** の場所に戻るようにはリンクできません。

この機能は、多くの場合、データおよびインデックスファイルを RAID またはその他の高速ソリューションに移動するために使用されます。

- ポート数の制限

Windows システムにはクライアント接続のポートがおよそ 4,000 あり、1 つのポート接続が閉じるとそのポートを再度利用できるようになるまで 2 から 4 分かかります。クライアントがサーバーとの接続と切断を高い頻度で繰り返す環境では、閉じたポートが再度利用できるようになる前に、利用できるポートがすべて使用されてしまうことがあります。このようになると、MySQL Server は動作中であっても反応していないように見えます。ポートは、マシンで実行されている他のアプリケーションでも使用できます。この場合、MySQL で使用可能なポートの数は少なくなります。

この問題の詳細は、<http://support.microsoft.com/default.aspx?scid=kb;en-us;196271> を参照してください。

- DATA DIRECTORY および INDEX DIRECTORY

セクション 15.6.1.2 「外部でのテーブルの作成」 で説明されているように、**CREATE TABLE** ステートメントの **DATA DIRECTORY** 句は、Windows for InnoDB テーブルでのみサポートされます。**MyISAM** およびその他のストレージエンジンの場合、**CREATE TABLE** 用の **DATA DIRECTORY** および **INDEX DIRECTORY** 句は、機能しない **realpath()** 呼び出しを持つ Windows およびその他のプラットフォームでは無視されます。

- DROP DATABASE

別のセッションで使用されているデータベースは削除できません。

- 大文字と小文字を区別しない名前

Windows ではファイル名の大文字と小文字は区別されないため、Windows では MySQL データベースとテーブル名の大文字と小文字も区別されません。唯一の制約は、特定のステートメント全体で大文字と小文字を変更せずに、データベース名とテーブル名を指定する必要があるということだけです。**セクション 9.2.3 「識別子の文字の区別」** を参照してください。

- ディレクトリ名とファイル名

Windows では、MySQL Server は現行の ANSI コードページと互換性のあるディレクトリ名とファイル名のみをサポートします。たとえば、次の日本語のディレクトリ名は西口ケール (コードページ 1252) では機能しません:

```
datadir="C:/私たちのプロジェクトのデータ"
```

LOAD DATA のデータファイルパス名など、SQL ステートメントで参照されるディレクトリ名およびファイル名にも同じ制限が適用されます。

- \ (パス名の区切り文字)

Windows でのパス名のコンポーネントは、\ 文字で区切られますが、これは MySQL のエスケープ文字でもあります。**LOAD DATA** または **SELECT ... INTO OUTFILE** を使用している場合は、/ 文字で Unix スタイルのファイル名を使用します:


```
mysql> LOAD DATA INFILE 'C:/tmp/skr.txt' INTO TABLE skr;  
mysql> SELECT * INTO OUTFILE 'C:/tmp/skr.txt' FROM skr;
```

または、\文字を2重にする必要があります。

```
mysql> LOAD DATA INFILE 'C:\\tmp\\skr.txt' INTO TABLE skr;  
mysql> SELECT * INTO OUTFILE 'C:\\tmp\\skr.txt' FROM skr;
```

- パイプに関する問題

パイプは Windows のコマンド行プロンプトからでは確実に機能しません。パイプに ^Z / CHAR(24) が含まれている場合、Windows はファイルの最後だと勘違いしてプログラムを中止します。

これは主に、次のようにバイナリログを適用するときに問題になります。

```
C:\> mysqlbinlog binary_log_file | mysql --user=root
```

ログを適用するときに問題が発生し、その原因が ^Z / CHAR(24) 文字によるものだと考えられる場合は、次の回避法を使用できます。

```
C:\> mysqlbinlog binary_log_file --result-file=/tmp/bin.sql  
C:\> mysql --user=root --execute "source /tmp/bin.sql"
```

後者のコマンドを使用して、バイナリデータを含む SQL ファイルを確実に読み取ることもできます。

2.4 macOS への MySQL のインストール

MySQL サーバーがサポートする macOS バージョンのリストは、<https://www.mysql.com/support/supportedplatforms/database.html> を参照してください。

MySQL for macOS は、様々な形式で使用できます:

- ネイティブパッケージインストーラ: ネイティブ macOS インストーラ (DMG) を使用して、MySQL のインストールを順を追って説明します。詳細は、[セクション2.4.2「ネイティブパッケージを使用した macOS への MySQL のインストール」](#)を参照してください。パッケージインストーラは、macOS で使用できます。インストールの実行に使用するユーザーは、管理者権限を持っていないければなりません。
- 圧縮 TAR アーカイブ。Unix `tar` および `gzip` コマンドを使用してパッケージ化されたファイルを使用します。この方法を使用するには、`Terminal` ウィンドウを開く必要があります。この方法を使用する場合、管理者権限は必要ありません。この方法を使用すると、どこでも MySQL サーバーをインストールできます。この方法の使用については、`tarball` の使用に関する一般的な説明 [セクション2.2「一般的なバイナリを使用した MySQL の Unix/Linux へのインストール」](#)を使用できます。

パッケージインストーラには、コアインストールに加えて、インストールの管理を簡略化するための [セクション2.4.3「MySQL 起動デーモンのインストールおよび使用」](#) および [セクション2.4.4「MySQL Preference Pane のインストールと使用」](#) も含まれています。

macOS での MySQL の使用の詳細は、[セクション2.4.1「macOS への MySQL のインストールに関する一般的なノート」](#)を参照してください。

2.4.1 macOS への MySQL のインストールに関する一般的なノート

次の問題と注記に留意してください。

- その他の MySQL インストール: インストール手順では、Homebrew などのパッケージマネージャーによる MySQL のインストールは認識されません。インストールおよびアップグレードプロセスは、弊社が提供する MySQL パッケージ用です。他のインストールが存在する場合は、ポートの競合を回避するために、このインストーラを実行する前にそれらを停止することを検討してください。

Homebrew: たとえば、Homebrew を使用して MySQL Server をデフォルトの場所にインストールした場合、MySQL インストーラは別の場所にインストールされ、Homebrew からバージョンはアップグレードされません。このシナリオでは、デフォルトで同じポートを使用しようとする複数の MySQL インストールで終了します。brew サービス `mysql` の停止を実行して Homebrew MySQL サービスを停止するなど、このインストーラを実行する前に他の MySQL Server インスタンスを停止します。

- Launchd: MySQL 構成オプションを変更する launchd デーモンがインストールされます。必要に応じて編集を検討してください。詳細は、次のドキュメントを参照してください。また、macOS 10.10 では、起動されたデーモンを優先して起動アイテムのサポートが削除されました。macOS 「システムプリファレンス」のオプションの MySQL 設定ペインでは、launchd デーモンが使用されます。
- ユーザー: MySQL ディレクトリおよびデータを所有する特定の `mysql` ユーザーを作成する必要がある (または必要な) 場合があります。これは、`ディレクトリユーティリティー` を通じて実行可能であり、`mysql` ユーザーがすでに存在している必要があります。シングルユーザーモードで使用するには、`_mysql` (アンダースコアのプリフィクスに注意してください) のエントリが `/etc/passwd` システムファイルにすでに存在している必要があります。
- Data: MySQL パッケージインストーラは MySQL の内容をバージョンおよびプラットフォーム固有のディレクトリにインストールするため、これを使用してバージョン間でデータベースをアップグレードおよび移行できます。`data` ディレクトリを古いバージョンから新しいバージョンにコピーするか、代替の `datadir` 値を指定してデータディレクトリの場所を設定する必要があります。デフォルトでは、MySQL ディレクトリは `/usr/local/` の下にインストールされます。
- エイリアス: シェルリソースファイルにエイリアスを追加すると、`mysql` や `mysqladmin` などの一般的に使用されるプログラムにコマンドラインから簡単にアクセスできるようになります。`bash` での構文は次のようになります。

```
alias mysql=/usr/local/mysql/bin/mysql
alias mysqladmin=/usr/local/mysql/bin/mysqladmin
```

`tcsh` では次を使用します。

```
alias mysql /usr/local/mysql/bin/mysql
alias mysqladmin /usr/local/mysql/bin/mysqladmin
```

`/usr/local/mysql/bin` to your `PATH` 環境変数を追加するとさらによいでしょう。これを行うには、シェルの適切な起動ファイルを変更します。詳細については、[セクション4.2.1「MySQLプログラムの起動」](#)を参照してください。

- 削除中: 前のインストールから MySQL データベースファイルをコピーし、新しいサーバーを正常に起動したら、古いインストールファイルを削除してディスク領域を節約することを検討する必要があります。さらに、`/Library/Receipts/mysql-VERSION.pkg` にある、古いバージョンのパッケージを入れたディレクトリも削除する必要があります。
- レガシー: OS X 10.7 より前は、MySQL サーバーは OS X サーバーにバンドルされていました。

2.4.2 ネイティブパッケージを使用した macOS への MySQL のインストール

パッケージは、ディスクイメージ (`.dmg`) ファイルにあり、最初に Finder でそのアイコンをダブルクリックしてマウントする必要があります。すると、イメージがマウントされ、そのコンテンツが表示されます。

注記

インストールを続行する前に、コマンドラインで MySQL Manager アプリケーション (macOS サーバーの場合)、プリファレンスペインまたは `mysqladmin shutdown` を使用して、実行中のすべての MySQL サーバーインスタンスを停止してください。

パッケージインストーラを使用して MySQL をインストールするには:

1. MySQL パッケージインストーラを含むディスクイメージ (`.dmg`) ファイル (コミュニティバージョンは「[ここ](#)」で使用可能) をダウンロードします。ファイルをダブルクリックしてディスクイメージをマウントし、その内容を確認します。

ディスクから MySQL インストーラパッケージをダブルクリックします。ダウンロードした MySQL のバージョンに従って名前が付けられます。たとえば、MySQL サーバー 8.0.29 の場合は、`mysql-8.0.29-macos-10.13-x86_64.pkg` という名前になります。

2. ウィザードの最初の導入画面では、インストールする MySQL サーバーのバージョンを参照します。続行をクリックしてインストールを開始します。

MySQL コミュニティエディションには、関連する GNU General Public License のコピーが表示されます。「Continue」、「Agree」の順にクリックして続行します。

3. 「インストールタイプ」ページで、Install をクリックしてすべてのデフォルトを使用してインストールウィザードを実行し、カスタマイズをクリックしてインストールするコンポーネント (MySQL サーバー、MySQL テスト、

プリファレンス(ペイン、起動されたサポート)を変更できます -- MySQL Test 以外はすべてデフォルトで有効になっています)。

注記

インストール場所の変更オプションは表示されますが、インストール場所は変更できません。

図 2.13 MySQL パッケージインストーラウィザード: インストールタイプ

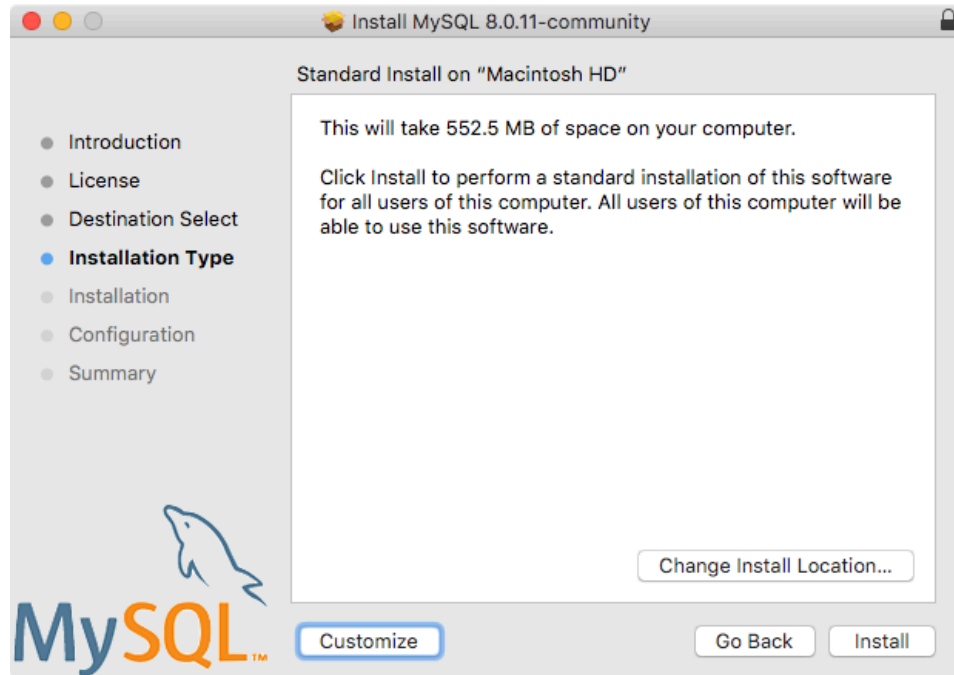
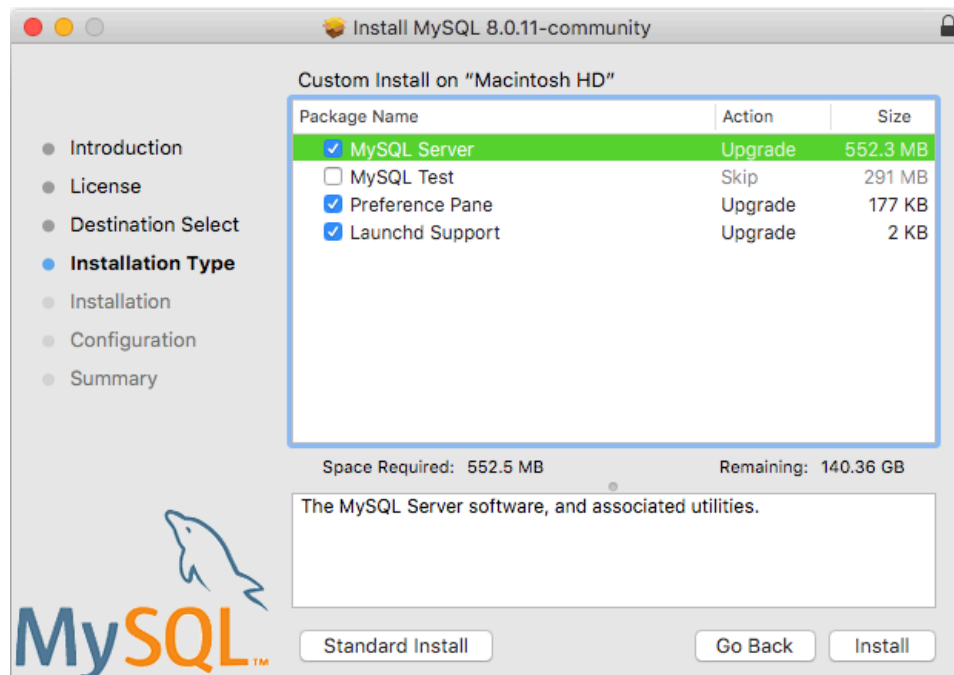


図 2.14 MySQL パッケージインストーラウィザード: カスタマイズ



4. Install をクリックして、MySQL Server をインストールします。現在の MySQL Server インストールをアップグレードする場合、インストールプロセスはここで終了します。それ以外の場合は、新しい MySQL Server インストールのウィザード追加構成ステップに従います。
5. 新しい MySQL Server が正常にインストールされたら、パスワードのデフォルトの暗号化タイプを選択して構成ステップを完了し、root パスワードを定義して、起動時に MySQL サーバーを有効(または無効)にします。
6. デフォルトの MySQL 8.0 パスワードメカニズムは `caching_sha2_password` (Strong) で、このステップで `mysql_native_password` (レガシー) に変更できます。

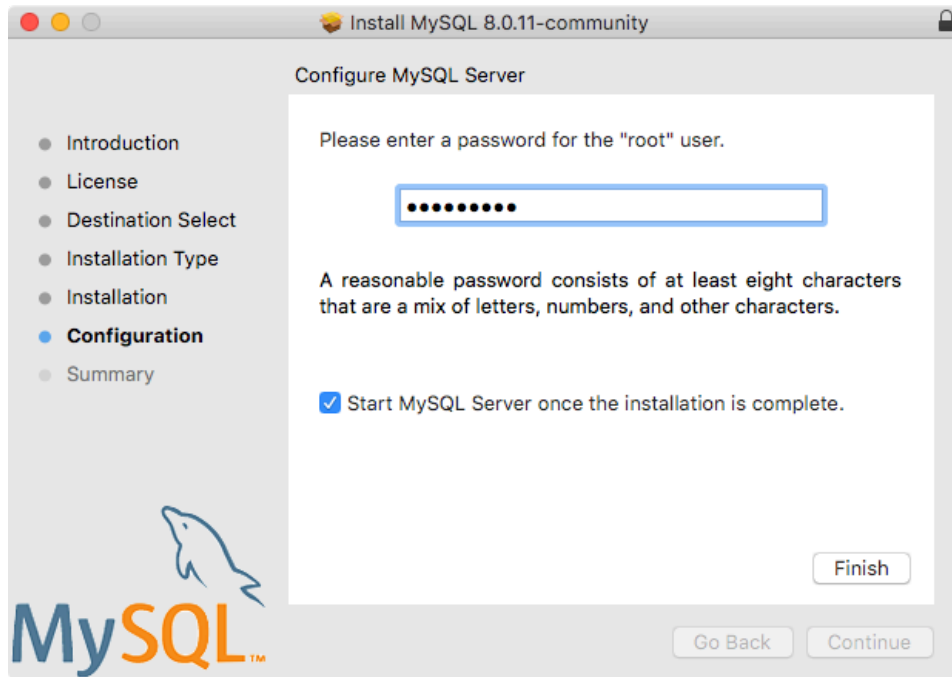
図 2.15 MySQL パッケージインストーラウィザード: パスワード暗号化タイプの選択



レガシーパスワードメカニズムを選択すると、生成された起動ファイルが変更され、`ProgramArguments` で `--default_authentication_plugin=mysql_native_password` が設定されます。強力なパスワード暗号化を選択しても、デフォルトの MySQL Server 値 (`caching_sha2_password`) が使用されるため、`--default_authentication_plugin` は設定されません。

7. root ユーザーのパスワードを定義し、構成ステップの完了後に MySQL Server を起動するかどうかも切り替えます。

図 2.16 MySQL パッケージインストーラウィザード: ルートパスワードの定義



8. 「まとめ」は最後のステップで、成功した完全な MySQL Server インストールを参照します。ウィザードでクローズを実行します。

図 2.17 MySQL パッケージインストーラウィザード: まとめ



MySQL サーバーがインストールされました。MySQL を起動しないことを選択した場合は、コマンドラインから `launchctl` を使用するが、MySQL のプリファレンスペインで「起動」をクリックして MySQL を起動します。詳細

については、[セクション2.4.3「MySQL 起動デーモンのインストールおよび使用」](#)、および[セクション2.4.4「MySQL Preference Pane のインストールと使用」](#)を参照してください。MySQL 設定ペインまたは `launchd` を使用して、ブート時に自動的に起動するように MySQL を構成します。

Package Installer を使用してインストールする場合、ファイルは `/usr/local` 内の、インストールするバージョンとプラットフォームに一致する名前のディレクトリにインストールされます。たとえば、インストーラファイル `mysql-8.0.29-macos10.15-x86_64.dmg` は、`/usr/local/mysql` へのシンボリックリンクを使用して MySQL を `/usr/local/mysql-8.0.29-macos10.15-x86_64/` にインストールします。次のテーブルに、この MySQL インストールディレクトリのレイアウトを示します。

注記

macOS のインストールプロセスでは、サンプルの `my.cnf` MySQL 構成ファイルは作成またはインストールされません。

表 2.7 macOS での MySQL のインストールレイアウト

ディレクトリ	ディレクトリの内容
<code>bin</code>	<code>mysqld</code> サーバー、クライアントプログラムおよびユーティリティプログラム
<code>data</code>	ログファイル、データベース (<code>/usr/local/mysql/data/</code> <code>mysqld.local.err</code> はデフォルトのエラーログ)
<code>docs</code>	リリースノートおよびビルド情報などのヘルパードキュメント
<code>include</code>	インクルード(ヘッダー) ファイル
<code>lib</code>	ライブラリ
<code>man</code>	Unix マニュアルページ
<code>mysql-test</code>	MySQL テストスイート ('MySQL Test'は、インストーラパッケージ (DMG) の使用時にインストールプロセス中にデフォルトで無効になっています)
<code>share</code>	エラーメッセージ、 <code>dictionary.txt</code> 、リライタ SQL などのその他のサポートファイル
<code>support-files</code>	<code>mysqld_multi.server</code> 、 <code>mysql.server</code> 、 <code>mysql-log-rotate</code> などのスクリプトをサポートします。
<code>/tmp/mysql.sock</code>	MySQL Unix ソケットの場所

2.4.3 MySQL 起動デーモンのインストールおよび使用

macOS では、起動デーモンを使用して、MySQL などのプロセスおよびアプリケーションを自動的に起動、停止および管理します。

デフォルトでは、macOS のインストールパッケージ (DMG) によって、次のような plist 定義を含む `Library/LaunchDaemons/com.oracle.oss.mysql.mysqld.plist` という名前の起動ファイルがインストールされます:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>Label</key>      <string>com.oracle.oss.mysql.mysqld</string>
  <key>ProcessType</key> <string>Interactive</string>
  <key>Disabled</key>   <false/>
  <key>RunAtLoad</key>  <true/>
  <key>KeepAlive</key>  <true/>
  <key>SessionCreate</key> <true/>
  <key>LaunchOnlyOnce</key> <false/>
  <key>UserName</key>   <string>_mysql</string>
  <key>GroupName</key>  <string>_mysql</string>
  <key>ExitTimeOut</key> <integer>600</integer>
  <key>Program</key>    <string>/usr/local/mysql/bin/mysqld</string>
  <key>ProgramArguments</key>
```

```
<array>
  <string>/usr/local/mysql/bin/mysqld</string>
  <string>--user=_mysql</string>
  <string>--basedir=/usr/local/mysql</string>
  <string>--datadir=/usr/local/mysql/data</string>
  <string>--plugin-dir=/usr/local/mysql/lib/plugin</string>
  <string>--log-error=/usr/local/mysql/data/mysqld.local.err</string>
  <string>--pid-file=/usr/local/mysql/data/mysqld.local.pid</string>
  <string>--keyring-file-data=/usr/local/mysql/keyring/keyring</string>
  <string>--early-plugin-load=keyring_file=keyring_file.so</string>
</array>
<key>WorkingDirectory</key> <string>/usr/local/mysql</string>
</dict>
</plist>
```

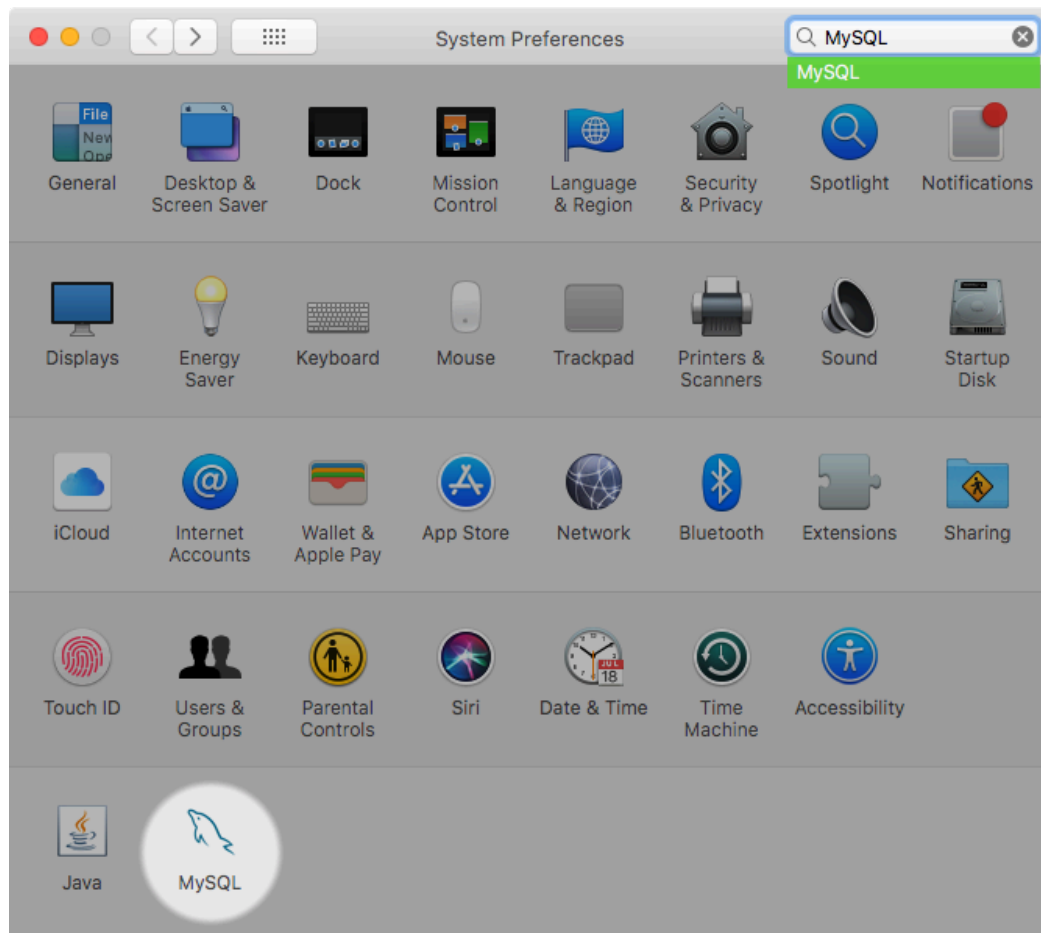
注記

plist DOCTYPE 宣言を追加すると lint チェックに合格しているにもかかわらず、起動された操作が失敗すると報告されるユーザーもいます。copy-n-paste エラーである可能性があります。前述のスニペットを含むファイルの md5 チェックサムは d925f05f6d1b6ee5ce5451b596d6baed です。

起動されたサービスを有効にするには、次のいずれかを実行します:

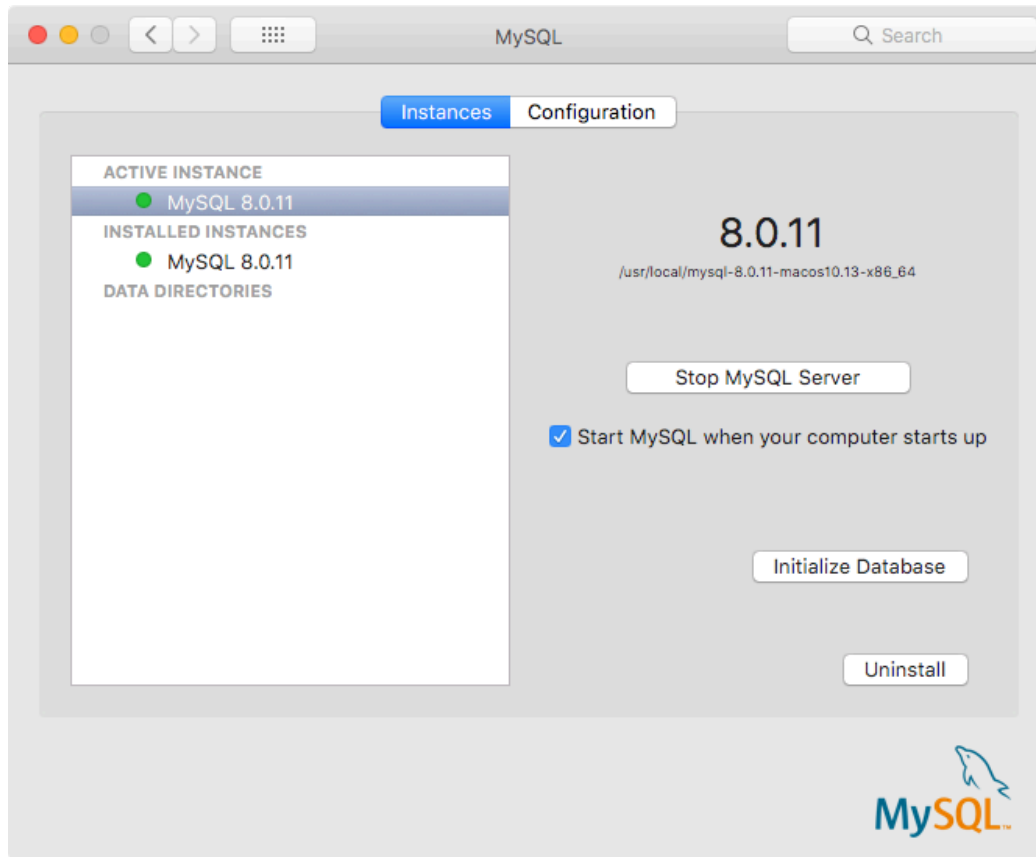
- macOS システムプリファレンスを開き、MySQL プリファレンスパネルを選択して、MySQL Server の起動を実行します。

図 2.18 MySQL Preference Pane: 場所



「インスタンス」ページには、MySQL を起動または停止するオプションが含まれており、データベースの初期化によって `data/` ディレクトリが再作成されます。アンインストールは、MySQL Server をアンインストールし、オプションで MySQL 設定パネルおよび起動された情報をアンインストールします。

図 2.19 MySQL プリファレンスペイン: インスタンス



- または、起動されたファイルを手動でロードします。

```
shell> cd /Library/LaunchDaemons
shell> sudo launchctl load -F com.oracle.oss.mysql.mysqlid.plist
```

- ブート時に自動的に起動するように MySQL を構成するには、次の手順を実行します:

```
shell> sudo launchctl load -w com.oracle.oss.mysql.mysqlid.plist
```

注記

MySQL サーバーをアップグレードする場合、起動されたインストールプロセスによって、MySQL サーバー 5.7.7 以下とともにインストールされた古い起動アイテムが削除されます。

アップグレードすると、`com.oracle.oss.mysql.mysqlid.plist` という名前の既存の起動ファイルも置換されます。

その他の起動関連情報:

- plist エントリはコマンドライン引数として渡されるため、`my.cnf` エントリをオーバーライドします。プログラムオプションを渡す方法の詳細は、[セクション4.2.2「プログラムオプションの指定」](#)を参照してください。

- ProgramArguments セクションでは、プログラムに渡されるコマンド行オプション (この場合は `mysqld` バイナリ) を定義します。
- デフォルトの plist 定義は、より洗練されたユースケースを念頭に置いて記述されます。より複雑な設定の場合は、一部の引数を削除し、かわりに `my.cnf` などの MySQL 構成ファイルに依存できます。
- plist ファイルを編集する場合は、MySQL の再インストールまたはアップグレード時にインストーラオプションの選択を解除します。そうしないと、編集した plist ファイルが上書きされ、すべての編集内容が失われます。

デフォルトの plist 定義では複数の ProgramArguments が定義されているため、これらの引数のほとんどを削除し、かわりに `my.cnf` MySQL 構成ファイルを使用してそれらを定義できます。例:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer/DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>Label</key>      <string>com.oracle.oss.mysql.mysqld</string>
  <key>ProcessType</key> <string>Interactive</string>
  <key>Disabled</key>   <false/>
  <key>RunAtLoad</key>  <true/>
  <key>KeepAlive</key>  <true/>
  <key>SessionCreate</key> <true/>
  <key>LaunchOnlyOnce</key> <false/>
  <key>UserName</key>   <string>_mysql</string>
  <key>GroupName</key> <string>_mysql</string>
  <key>ExitTimeOut</key> <integer>600</integer>
  <key>Program</key>   <string>/usr/local/mysql/bin/mysqld</string>
  <key>ProgramArguments</key>
  <array>
    <string>/usr/local/mysql/bin/mysqld</string>
    <string>--user=_mysql</string>
    <string>--basedir=/usr/local/mysql</string>
    <string>--datadir=/usr/local/mysql/data</string>
    <string>--plugin-dir=/usr/local/mysql/lib/plugin</string>
    <string>--log-error=/usr/local/mysql/data/mysqld.local.err</string>
    <string>--pid-file=/usr/local/mysql/data/mysqld.local.pid</string>
    <string>--keyring-file-data=/usr/local/mysql/keyring/keyring</string>
    <string>--early-plugin-load=keyring_file=keyring_file.so</string>
  </array>
  <key>WorkingDirectory</key> <string>/usr/local/mysql</string>
</dict>
</plist>
```

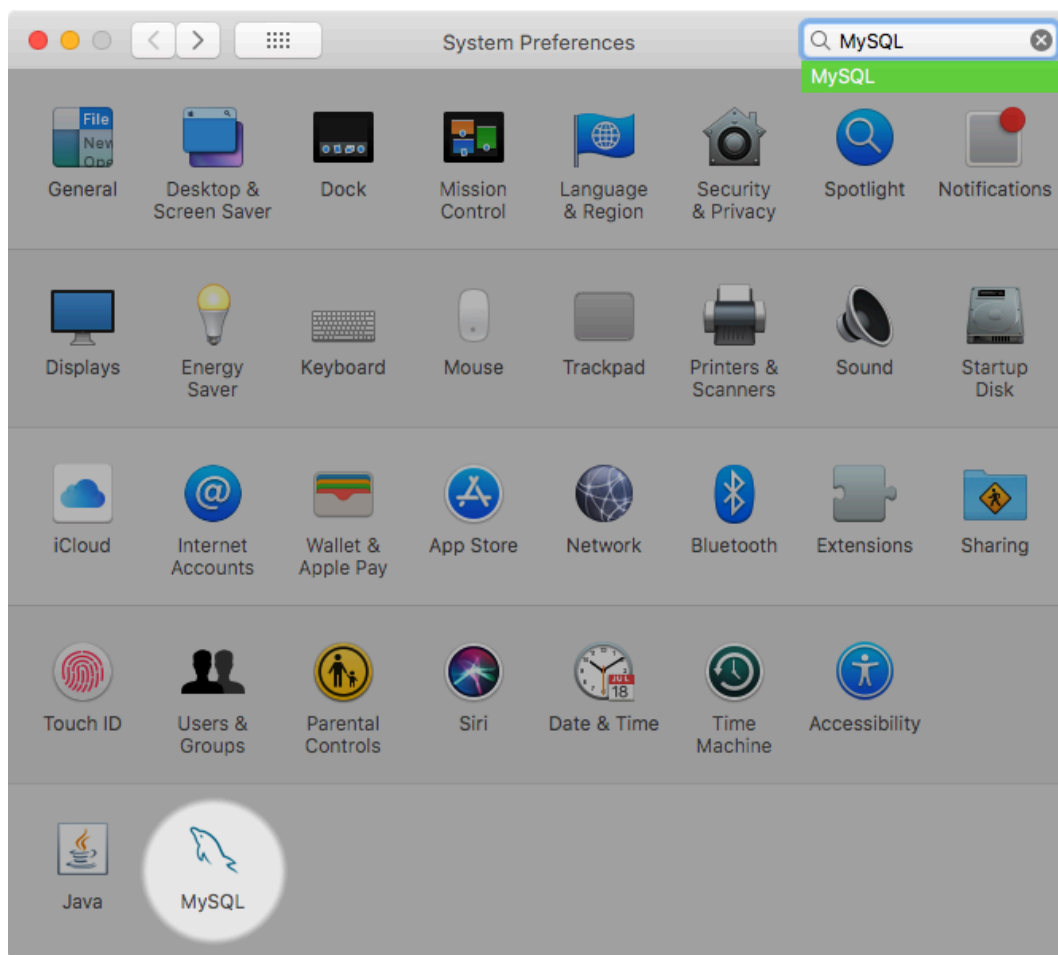
この場合、`my.cnf` で定義されている可能性があるデフォルトの plist ProgramArguments 定義から、`basedir`, `datadir`, `plugin_dir`, `log_error`, `pid_file`, `keyring_file_data`、および `--early-plugin-load` オプションが削除されました。

2.4.4 MySQL Preference Pane のインストールと使用

MySQL インストールパッケージには、MySQL インストールのブート中に自動起動を起動、停止、および制御できる MySQL 設定ペインが含まれています。

このプリファレンスペインはデフォルトでインストールされ、システムのシステムプリファレンスウィンドウの下にリストされます。

図 2.20 MySQL Preference Pane: 場所

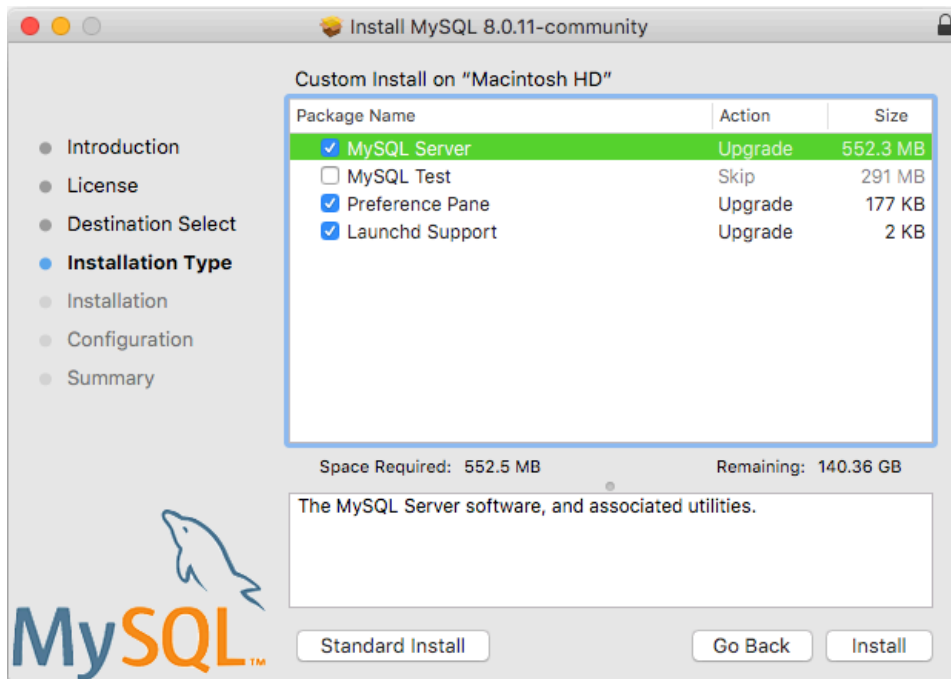


MySQL プリファレンスペインは、MySQL Server をインストールする DMG ファイルと同じ DMG ファイルとともにインストールされます。通常は MySQL Server とともにインストールされますが、単独でもインストールできます。

MySQL プリファレンスペインをインストールするには:

1. MySQL Server のインストールプロセスを、[セクション2.4.2「ネイティブパッケージを使用した macOS への MySQL のインストール」](#)のドキュメントの記述に従って実行します。
2. 「Installation Type」の手順で、「Customize」をクリックします。「プリファレンスペイン」オプションがここにリストされ、デフォルトで有効になっています。選択が解除されていないことを確認してください。MySQL Server などのその他のオプションは選択または選択解除できます。

図 2.21 MySQL パッケージインストーラウィザード: カスタマイズ



3. インストールプロセスを完了します。

注記

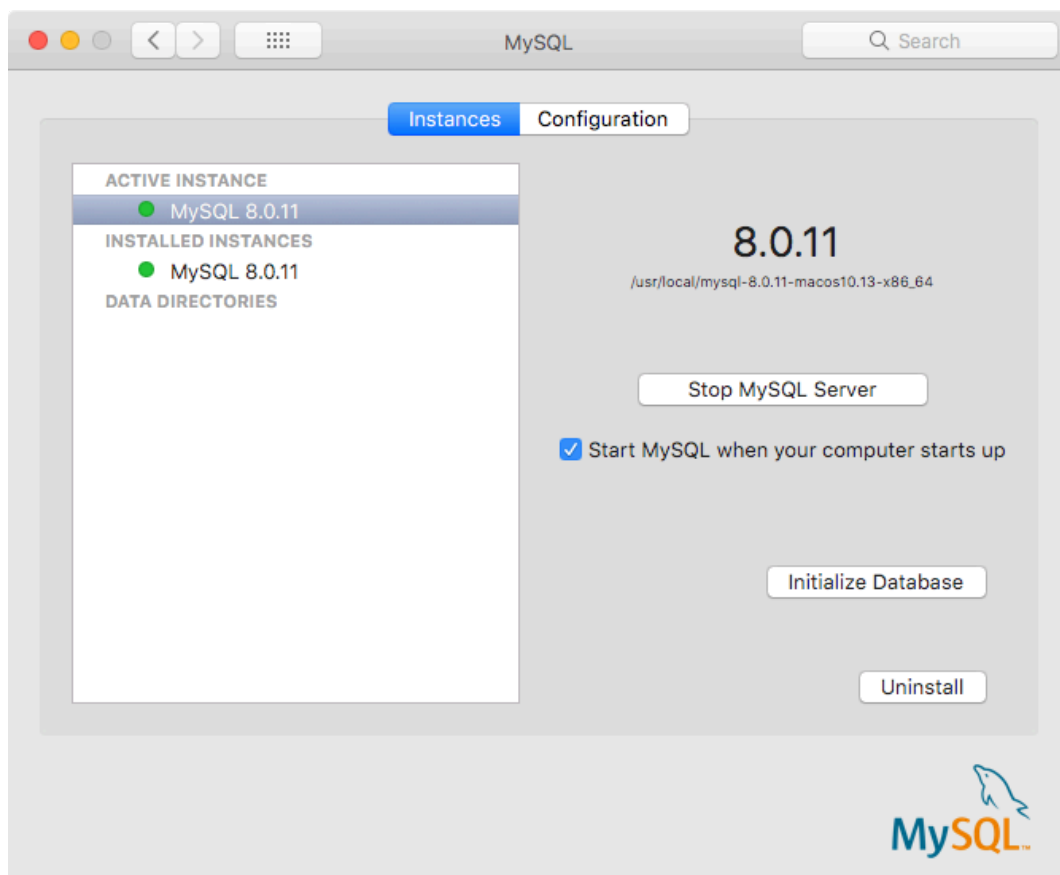
MySQL 設定ペインは、デフォルトの場所にインストールされている MySQL パッケージのインストールからインストールされた MySQL のインストールのみを開始および停止します。

MySQL プリファレンスペインがインストールされると、このプリファレンスペインを使用して MySQL サーバースタンスを制御できます。

「インスタンス」ページには、MySQL を起動および停止するオプションが含まれており、データベースの初期化によって `data/` ディレクトリが再作成されます。アンインストールにより、MySQL Server がアンインストールされ、オプションで問題および起動情報がアンインストールされます。

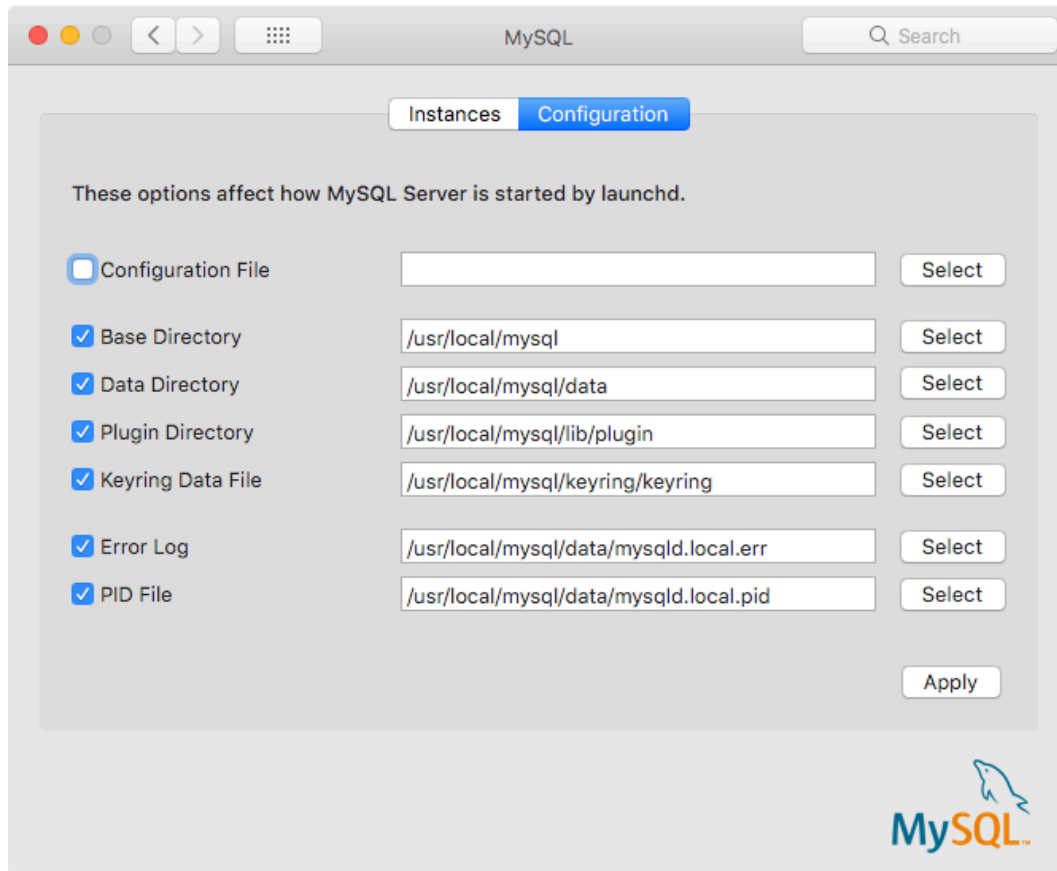
「インスタンス」ページには、MySQL を起動または停止するオプションが含まれており、データベースの初期化によって `data/` ディレクトリが再作成されます。アンインストールは、MySQL Server をアンインストールし、オプションで MySQL 設定パネルおよび起動された情報をアンインストールします。

図 2.22 MySQL プリファレンスペイン: インスタンス



「構成」ページには、MySQL 構成ファイルへのパスを含む MySQL Server オプションが表示されます。

図 2.23 MySQL プリファレンスペイン: 構成



MySQL Preference Pane は、MySQL Server の現在のステータスを表示します。サーバーが実行していない場合は (赤で) 「stopped」、サーバーがすでに起動している場合は (緑で) 「running」と表示します。Preference Pane は、MySQL Server が自動的に起動するように設定されているかどうかに関する現在の設定も表示します。

2.5 Linux に MySQL をインストールする

Linux は、MySQL をインストールするための何種類かのソリューションをサポートします。オラクルからの配布のいずれかを使用することを推奨します。いくつかのインストール方法が使用可能です。

表 2.8 Linux のインストール方法と情報

型	設定方法	追加情報
Apt	「MySQL Apt リポジトリ」の有効化	Documentation
Yum	「MySQL Yum リポジトリ」の有効化	Documentation
Zypper	「MySQL SLES リポジトリ」の有効化	Documentation
RPM	特定のパッケージの「ダウンロード」	Documentation
DEB	特定のパッケージの「ダウンロード」	Documentation
Generic	汎用パッケージの「ダウンロード」	Documentation
ソース	source からコンパイル	Documentation
Docker	Docker Hub for MySQL Community Edition の使用; My Oracle Support から MySQL Enterprise Edition の Docker イメージをダウンロード	Documentation

型	設定方法	追加情報
Oracle Unbreakable Linux Network	ULN チャネルの使用	Documentation

代わりにシステムのパッケージマネージャーを使用して、Linux 配布のネイティブソフトウェアリポジトリからのパッケージで、MySQL を自動的にダウンロードしてインストールできます。これらのネイティブパッケージは多くの場合、使用可能な最新のリリースから数バージョン遅れています。通常、開発マイルストーンリリース (DMR) はネイティブリポジトリで使用可能になっていないため、インストールできません。ネイティブパッケージインストーラの使用の詳細は、[セクション2.5.7「ネイティブソフトウェアリポジトリから MySQL を Linux にインストールする」](#)を参照してください。

注記

多くの Linux インストールでは、マシンの起動時に MySQL が自動的に起動するように設定する必要があります。多くのネイティブパッケージインストールではこの操作は自動的に行われますが、ソース、バイナリ、および RPM ソリューションでは個別の設定が必要になる場合があります。必要なスクリプト `mysql.server` は、MySQL インストールディレクトリの下に `support-files` ディレクトリ、または MySQL ソースツリーにあります。自動的に起動およびシャットダウンするには、それを `/etc/init.d/mysql` としてインストールします。[セクション4.3.3「mysql.server — MySQL サーバー起動スクリプト」](#)を参照してください。

2.5.1 MySQL Yum リポジトリを使用して MySQL を Linux にインストールする

Oracle Linux、Red Hat Enterprise Linux、CentOS および Fedora の「[MySQL Yum リポジトリ](#)」には、MySQL サーバー、クライアント、MySQL ワークベンチ、MySQL コーティリテイ、MySQL Router、MySQL Shell、Connector/ODBC、Connector/Python などをインストールするための RPM パッケージが用意されています (すべてのディストリビューションで使用できるわけではありません。[追加の MySQL 製品およびコンポーネントの Yum でのインストールの詳細を参照してください](#))。

開始する前に

MySQL は、広く使用されるオープンソースのソフトウェアとして、オリジナルの形式または再パッケージされた形式で、さまざまなダウンロードサイトやソフトウェアリポジトリなどのさまざまなソースから、多くのシステムに広くインストールされています。次の手順では、サードパーティ配布 RPM パッケージを使用しているシステムに MySQL がまだインストールされていないことを前提としています。インストールされていない場合は、[セクション2.11.7「MySQL Yum リポジトリを使用する MySQL のアップグレード」](#) または [Replacing a Third-Party Distribution of MySQL Using the MySQL Yum Repository](#) に記載されている手順に従ってください。

MySQL のインストールを最初から実行する手順

MySQL の最新の GA バージョンを MySQL Yum リポジトリでインストールするには、次の手順に従ってください。

MySQL Yum リポジトリの追加

まず、MySQL Yum リポジトリをシステムのリポジトリリストに追加します。この操作は一度だけ必要で、MySQL が提供する RPM をインストールすることで実行できます。次の手順に従います。

- MySQL Developer Zone の「[Download MySQL Yum Repository](#)」ページ (<https://dev.mysql.com/downloads/repo/yum/>) に移動します。
- 使用しているプラットフォーム用のリリースパッケージを選択してダウンロードします。
- ダウンロードしたリリースパッケージを次のコマンドでインストールし、`platform-and-version-specific-package-name` をダウンロードした RPM パッケージの名前に置き換えます:

```
shell> sudo yum install platform-and-version-specific-package-name.rpm
```

EL6 ベースのシステムでは、コマンドは次の形式です。

```
shell> sudo yum install mysql80-community-release-el6-{version-number}.noarch.rpm
```

EL7 ベースのシステムの場合:


```
shell> sudo yum install mysql80-community-release-el7-{version-number}.noarch.rpm
```

EL8-based システムの場合:

```
shell> sudo yum install mysql80-community-release-el8-{version-number}.noarch.rpm
```

Fedora 33 の場合:

```
shell> sudo dnf install mysql80-community-release-fc33-{version-number}.noarch.rpm
```

Fedora 32 の場合:

```
shell> sudo dnf install mysql80-community-release-fc32-{version-number}.noarch.rpm
```

インストールコマンドにより、MySQL Yum リポジトリがシステムのリポジトリリストに追加され、ソフトウェアパッケージの完全性をチェックするために GnuPG 鍵がダウンロードされます。GnuPG 鍵チェックの詳細は、[セクション2.1.4.2「GnuPG を使用した署名確認」](#)を参照してください。

次のコマンドを使用して、MySQL Yum リポジトリが正常に追加されたことを確認できます (dnf 対応システムの場合、コマンドの `yum` を `dnf` に置き換えます):

```
shell> yum repolist enabled | grep "mysql.*-community.*"
```

注記

MySQL Yum リポジトリがシステムで有効になると、`yum update` コマンド (dnf 対応システムの場合は `dnf upgrade`) によるシステム全体の更新によって、システム上の MySQL パッケージがアップグレードされ、MySQL Yum リポジトリ内でネイティブサードパーティパッケージの置換が検出された場合は置換されます。システムで発生する可能性のある影響については、[セクション2.11.7「MySQL Yum リポジトリを使用する MySQL のアップグレード」](#)を参照してください。[共有クライアントライブラリのアップグレード](#)を参照してください。

リリースシリーズの選択

MySQL Yum リポジトリを使用する場合、デフォルトで最新の GA シリーズ (現在の MySQL 8.0) がインストール用に選択されます。それでよい場合は、次の手順の[MySQL のインストール](#)に進んでください。

MySQL Yum リポジトリ内では、MySQL Community Server の異なるリリースシリーズが、異なるサブリポジトリにホストされています。最新の GA シリーズ (現在の MySQL 8.0) のサブリポジトリはデフォルトで有効になっており、他のすべてのシリーズ (MySQL 8.0 シリーズなど) のサブリポジトリはデフォルトで無効になっています。このコマンドを使用して、MySQL Yum リポジトリ内のすべてのサブリポジトリを表示し、どのサブリポジトリが有効化または無効化されているかを確認します (dnf 対応システムの場合は、コマンド内の `yum` を `dnf` に置き換えます):

```
shell> yum repolist all | grep mysql
```

最新の GA シリーズの最新のリリースをインストールする場合は、構成は不要です。最新の GA シリーズ以外の特定のシリーズの最新リリースをインストールするには、インストールコマンドを実行する前に、最新の GA シリーズのサブリポジトリを無効にし、その特定のシリーズのサブリポジトリを有効にします。プラットフォームで `yum-config-manager` がサポートされている場合は、次のコマンドを発行して、5.7 シリーズのサブリポジトリを無効にし、8.0 シリーズのサブリポジトリを有効にします:

```
shell> sudo yum-config-manager --disable mysql57-community  
shell> sudo yum-config-manager --enable mysql80-community
```

dnf 対応プラットフォームの場合:

```
shell> sudo dnf config-manager --disable mysql57-community
```

```
shell> sudo dnf config-manager --enable mysql80-community
```

`yum-config-manager` または `dnf config-manager` コマンドの他に、`/etc/yum.repos.d/mysql-community.repo` ファイルを手動で編集してリリースシリーズを選択することもできます。ファイル内の、リリースシリーズのサブリポジトリの典型的なエントリを次に示します。

```
[mysql57-community]
name=MySQL 5.7 Community Server
baseurl=http://repo.mysql.com/yum/mysql-5.7-community/el/6/$basearch/
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-mysql
```

構成したいサブリポジトリのエントリを探し、`enabled` オプションを編集します。`enabled=0` を指定してサブリポジトリを無効にするか、`enabled=1` を指定してサブリポジトリを有効にします。たとえば、MySQL 8.0 をインストールするには、前述の MySQL 5.7 のサブリポジトリエントリに `enabled=0` があり、8.0 シリーズのエントリに `enabled=1` があることを確認します:

```
# Enable to use MySQL 8.0
[mysql80-community]
name=MySQL 8.0 Community Server
baseurl=http://repo.mysql.com/yum/mysql-8.0-community/el/6/$basearch/
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-mysql
```

一度に有効にするサブリポジトリは、1つのリリースシリーズのものみにしてください。複数のリリースシリーズのサブリポジトリが有効になっている場合、Yum は最新のシリーズを使用します。

次のコマンドを実行し、その出力を確認して、正しいサブリポジトリが有効化および無効化されていることを確認します (dnf 対応システムの場合、コマンドの `yum` を `dnf` に置き換えます):

```
shell> yum repolist enabled | grep mysql
```

デフォルトの MySQL モジュールの無効化

(EL8 システムのみ) RHEL8 や Oracle Linux 8 などの EL8-based システムには、デフォルトで有効になっている MySQL モジュールが含まれています。このモジュールを無効にしないかぎり、MySQL リポジトリによって提供されるパッケージがマスクされます。含まれているモジュールを無効にして MySQL リポジトリパッケージを表示するには、次のコマンドを使用します (dnf 対応システムの場合は、コマンドの `yum` を `dnf` に置き換えます):

```
shell> sudo yum module disable mysql
```

MySQL のインストール

次のコマンドを使用して MySQL をインストールします (dnf 対応システムの場合は、コマンドの `yum` を `dnf` に置き換えます):

```
shell> sudo yum install mysql-community-server
```

これにより、MySQL Server (`mysql-community-server`)、クライアントのパッケージ (`mysql-community-client`)、クライアントとサーバー用の一般的なエラーメッセージと文字セット (`mysql-community-common`)、および共有クライアントライブラリ (`mysql-community-libs`) など、サーバーを実行するために必要なコンポーネントのパッケージがインストールされます。

MySQL Server の起動

次のコマンドで MySQL Server を起動します。

```
shell> systemctl start mysqld
```

次のコマンドで MySQL Server のステータスをチェックできます。

```
shell> systemctl status mysqld
```

オペレーティングシステムが `systemd` に対応している場合は、`stop`、`start`、`status` などの標準 `systemctl` (または引数を逆にした `service`) コマンドと `restart` を使用して MySQL サーバースerviceを管理する必要があります。 `mysqld` サービスはデフォルトで有効になっており、システムの再起動時に開始されます。 詳細は、[セクション2.5.9「systemdを使用したMySQL Serverの管理」](#)を参照してください。

サーバーの最初の起動時に、サーバーのデータディレクトリが空の場合、次のようになります:

- サーバーが初期化されます。
- SSL 証明書およびキーファイルがデータディレクトリに生成されます。
- `validate_password` がインストールされ、有効になっています。
- スーパーユーザーアカウント'`root`'@'`localhost`'が作成されます。スーパーユーザーのパスワードが設定され、エラーログファイルに格納されます。表示するには、次のコマンドを使用します:

```
shell> sudo grep 'temporary password' /var/log/mysqld.log
```

生成された一時パスワードを使用してログインし、スーパーユーザーアカウントのカスタムパスワードを設定することで、`root` パスワードをできるだけ早く変更します:

```
shell> mysql -uroot -p
```

```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'MyNewPass4!';
```

注記

`validate_password` はデフォルトでインストールされます。 `validate_password` で実装されているデフォルトのパスワードポリシーでは、パスワードに大文字、小文字、数字および特殊文字を 1 文字以上含める必要があり、パスワードの合計長は 8 文字以上である必要があります。

インストール後の手順については、[セクション2.10「インストール後のセットアップとテスト」](#)を参照してください。

注記

EL7 ベースのプラットフォームの互換性情報: プラットフォームのネイティブソフトウェアリポジトリからの次の RPM パッケージは、MySQL Server をインストールする MySQL Yum リポジトリからのパッケージとは互換性がありません。MySQL Yum リポジトリを使用して MySQL をインストールした後は、これらのパッケージをインストールできません (その逆も同様です)。

- `akonadi-mysql`

追加の MySQL 製品およびコンポーネントの Yum でのインストール

Yum を使用して、MySQL の個々のコンポーネントのインストールおよび管理を実行できます。これらのコンポーネントの一部は、MySQL Yum リポジトリのサブリポジトリにホストされています。たとえば、MySQL Connectors は MySQL Connectors Community サブリポジトリ、MySQL Workbench は MySQL Tools Community にあります。次のコマンドを使用して、プラットフォームで使用可能なすべての MySQL コンポーネントのパッケージを MySQL Yum リポジトリからリストできます (dnf 対応システムの場合は、コマンド内の `yum` を `dnf` に置き換えます):

```
shell> sudo yum --disablerepo=* --enablerepo='mysql*-community*' list available
```

任意のパッケージを次のコマンドでインストールし、`package-name` をパッケージの名前に置き換えます (dnf 対応システムの場合は、コマンド内の `yum` を `dnf` に置き換えます):

```
shell> sudo yum install package-name
```

たとえば、Fedora に MySQL Workbench をインストールするには、次のようにします:

```
shell> sudo dnf install mysql-workbench-community
```

共有クライアントライブラリをインストールするには (dnf 対応システムの場合は、コマンドの `yum` を `dnf` に置き換えます):

```
shell> sudo yum install mysql-community-libs
```

プラットフォーム固有のノート

ARM のサポート

ARM 64-bit (aarch64) は Oracle Linux 7 でサポートされており、Oracle Linux 7 ソフトウェアコレクションリポジトリ (ol7_software_collections) が必要です。たとえば、サーバーを設置するには、次のようにします:

```
shell> yum-config-manager --enable ol7_software_collections
shell> yum install mysql-community-server
```

注記

ARM 64-bit (aarch64) は、MySQL 8.0.12 の Oracle Linux 7 でサポートされています。

既知の制限事項

8.0.12 リリースでは、`yum install` ステップの実行後に `ln -s /opt/oracle/oracle-asmtoolset-1/root/usr/lib64 /usr/lib64/gcc7` を実行して、`libstdc++7` パスを調整する必要があります。

Yum での MySQL の更新

インストールのほかに、MySQL 製品およびコンポーネントの更新も、MySQL Yum リポジトリを使用して実行できます。詳細は、[セクション2.11.7「MySQL Yum リポジトリを使用する MySQL のアップグレード」](#)を参照してください。

2.5.2 MySQL APT リポジトリを使用して MySQL を Linux にインストールする

MySQL APT リポジトリには、現在の Debian および Ubuntu リリースで MySQL サーバー、クライアントおよびその他のコンポーネントをインストールおよび管理するための `deb` パッケージが用意されています。

MySQL APT リポジトリの使用に関する説明は、[A Quick Guide to Using the MySQL APT Repository](#)にあります。

2.5.3 MySQL SLES リポジトリを使用して MySQL を Linux にインストールする

MySQL SLES リポジトリは、MySQL Server、クライアント、その他のコンポーネントを SUSE Enterprise Linux Server にインストールして管理するために、RPM パッケージを提供します。

MySQL SLES リポジトリを使用する手順は、「[A MySQL SLES リポジトリの使用法のクイックガイド](#)」で入手できます。

2.5.4 Oracle の RPM パッケージを使用した Linux への MySQL のインストール

RPM ベースの Linux ディストリビューションに MySQL をインストールするには、Oracle が提供する RPM パッケージを使用することをお勧めします。Community Edition of MySQL の場合、これらを取得するには次の 2 つのソースがあります:

- MySQL ソフトウェアリポジトリから:
 - MySQL Yum リポジトリ (詳細は [セクション2.5.1「MySQL Yum リポジトリを使用して MySQL を Linux にインストールする」](#)を参照)。
 - MySQL SLES リポジトリ (詳細は [セクション2.5.3「MySQL SLES リポジトリを使用して MySQL を Linux にインストールする」](#)を参照)。
- 「MySQL 開発者ゾーン」の「[MySQL コミュニティサーバーのダウンロード](#)」ページから。

注記

MySQL の RPM 配布はほかのベンダーからも提供されています。これらは、機能、機能および表記規則 (通信設定を含む) において Oracle によって構築されたものとは異なる場合があります。このマニュアルのインストール手順は必ずしも適用されないことに注意してください。代わりに、ベンダーの説明書を参照してください。

MySQL RPM パッケージ

表 2.9 MySQL Community Edition の RPM パッケージ

パッケージ名	サマリー
<code>mysql-community-client</code>	MySQL クライアントアプリケーションおよびツール
<code>mysql-community-common</code>	サーバーおよびクライアントライブラリの共通ファイル
<code>mysql-community-devel</code>	MySQL データベースクライアントアプリケーションの開発ヘッダーファイルおよびライブラリ
<code>mysql-community-embedded-compat</code>	ライブラリのバージョン 18 を使用するアプリケーションの互換性を持つ埋込みライブラリとしての MySQL サーバー
<code>mysql-community-libs</code>	MySQL データベースクライアントアプリケーションの共有ライブラリ
<code>mysql-community-libs-compat</code>	以前の MySQL インストールの共有互換性ライブラリ
<code>mysql-community-server</code>	データベースサーバーおよび関連ツール
<code>mysql-community-server-debug</code>	サーバーおよびプラグインバイナリをデバッグ
<code>mysql-community-test</code>	MySQL サーバーのテストスイート
<code>mysql-community</code>	ソースコード RPM は、選択した OS に応じて <code>mysql-community-8.0.29-1.el7.src.rpm</code> のようになります
追加の *debuginfo* RPM	複数の <code>debuginfo</code> パッケージがあります: <code>mysql-community-client-debuginfo</code> , <code>mysql-community-libs-debuginfo</code> , <code>mysql-community-server-debug-debuginfo</code> , <code>mysql-community-server-debuginfo</code> , および <code>mysql-community-test-debuginfo</code> .

表 2.10 MySQL Enterprise Edition の RPM パッケージ

パッケージ名	サマリー
<code>mysql-commercial-backup</code>	MySQL Enterprise Backup (8.0.11 で追加)
<code>mysql-commercial-client</code>	MySQL クライアントアプリケーションおよびツール
<code>mysql-commercial-common</code>	サーバーおよびクライアントライブラリの共通ファイル
<code>mysql-commercial-devel</code>	MySQL データベースクライアントアプリケーションの開発ヘッダーファイルおよびライブラリ
<code>mysql-commercial-embedded-compat</code>	ライブラリのバージョン 18 を使用するアプリケーションの互換性を持つ埋込みライブラリとしての MySQL サーバー
<code>mysql-commercial-libs</code>	MySQL データベースクライアントアプリケーションの共有ライブラリ
<code>mysql-commercial-libs-compat</code>	以前の MySQL インストールの共有互換性ライブラリ。ライブラリのバージョンは、使用しているディストリビューションによってデフォルトでインストールされるライブラリのバージョンと一致
<code>mysql-commercial-server</code>	データベースサーバーおよび関連ツール
<code>mysql-commercial-test</code>	MySQL サーバーのテストスイート
追加の *debuginfo* RPM	複数の <code>debuginfo</code> パッケージがあります: <code>mysql-commercial-client-debuginfo</code> , <code>mysql-commercial-libs-debuginfo</code> , <code>mysql-commercial-server-debug-debuginfo</code> , <code>mysql-commercial-server-debuginfo</code> , and <code>mysql-commercial-test-debuginfo</code> .

RPM のフルネームの構文は次のとおりです:


```
packagename-version-distribution-arch.rpm
```

distribution および **arch** の値は、パッケージが構築された Linux ディストリビューションおよびプロセッサタイプを示します。配布識別子のリストは、次のテーブルを参照してください:

表 2.11 MySQL Linux RPM パッケージ配布識別子

配分値	使用目的
el {version} ({version} は el8 などの Enterprise Linux のメジャーバージョン)	EL6、EL7 および EL8-based プラットフォーム (たとえば、Oracle Linux、Red Hat Enterprise Linux および CentOS の対応するバージョン)
fc {version}。ここで、{version} はメジャー Fedora バージョン (fc33 など) です	Fedora 33, 34, および 35
sles12	SUSE Linux Enterprise Server 12

RPM パッケージ内のすべてのファイル (`mysql-community-server` など) を表示するには、次のコマンドを使用します:

```
shell> rpm -qpl mysql-community-server-version-distribution-arch.rpm
```

このセクションの残りの部分の説明は、MySQL リポジトリを介してではなく、Oracle から直接ダウンロードした RPM パッケージを使用するインストールプロセスにのみ適用されます。

一部のパッケージには依存関係が存在します。多くのパッケージをインストールする場合は、RPM バンドル `tar` ファイルをダウンロードして、前述のすべての RPM パッケージを個別にダウンロードする必要がないようにします。

ほとんどの場合、機能する標準 MySQL インストールを取得するには、`mysql-community-server`、`mysql-community-client`、`mysql-community-libs`、`mysql-community-common` および `mysql-community-libs-compat` パッケージをインストールする必要があります。このような標準の基本インストールを実行するには、これらのすべてのパッケージが含まれるフォルダに移動し (可能であれば、類似した名前の RPM パッケージはない)、次のコマンドを発行します:

```
shell> sudo yum install mysql-community-{server,client,common,libs}-*
```

`yum` を SLES 用の `zypper` および Fedora 用の `dnf` に置き換えます。

`yum` などの高レベルのパッケージ管理ツールを使用してパッケージをインストールすることをお勧めしますが、直接 `rpm` コマンドを優先するユーザーは `yum install` コマンドを `rpm -Uvh` コマンドに置き換えることができます。ただし、`rpm -Uvh` を使用すると、インストールプロセスが実行される可能性のある依存関係の問題のため、インストールプロセスが失敗しやすくなります。

クライアントプログラムのみをインストールするには、インストールするパッケージのリストで `mysql-community-server` をスキップし、次のコマンドを発行します:

```
shell> sudo yum install mysql-community-{client,common,libs}-*
```

`yum` を SLES 用の `zypper` および Fedora 用の `dnf` に置き換えます。

RPM パッケージを使用して MySQL を標準インストールすると、次のテーブルに示すように、ファイルおよびリソースがシステムディレクトリの下に作成されます。

表 2.12 MySQL Developer Zone からの Linux RPM パッケージの MySQL インストールレイアウト

ファイルまたはリソース	事業所
クライアントプログラムおよびスクリプト	<code>/usr/bin</code>
<code>mysqld</code> サーバー	<code>/usr/sbin</code>
構成ファイル	<code>/etc/my.cnf</code>
データディレクトリ	<code>/var/lib/mysql</code>
エラーログファイル	RHEL、Oracle Linux、CentOS または Fedora プラットフォームの場合: <code>/var/log/mysqld.log</code> SLES 用: <code>/var/log/mysql/mysqld.log</code>
<code>secure_file_priv</code> の値	<code>/var/lib/mysql-files</code>

ファイルまたはリソース	事業所
System V init スクリプト	RHEL、Oracle Linux、CentOS または Fedora プラットフォームの場合: /etc/init.d/mysqld SLES 用: /etc/init.d/mysqld
Systemd サービス	RHEL、Oracle Linux、CentOS または Fedora プラットフォームの場合: mysqld SLES 用: mysqld
PID ファイル	/var/run/mysql/mysqld.pid
Socket	/var/lib/mysql/mysql.sock
キーリングディレクトリ	/var/lib/mysql-keyring
Unix マニュアルページ	/usr/share/man
インクルード (ヘッダー) ファイル	/usr/include/mysql
ライブラリ	/usr/lib/mysql
その他のサポートファイル (エラーメッセージ、文字セットファイルなど)	/usr/share/mysql

インストールでは、[mysql](#) という名前のユーザーおよび [mysql](#) という名前のグループもシステムに作成されます。

注記

古いパッケージを使用して以前のバージョンの MySQL をインストールすると、[/usr/my.cnf](#) という名前の構成ファイルが作成される場合があります。ファイルの内容を調べ、目的の設定をファイル/[etc/my.cnf](#) ファイル内に移行してから、[/usr/my.cnf](#) を削除することを強くお勧めします。

MySQL は、インストールプロセスの最後に自動的に起動されません。Red Hat Enterprise Linux、Oracle Linux、CentOS および Fedora システムの場合は、次のコマンドを使用して MySQL を起動します:

```
shell> systemctl start mysqld
```

SLES システムの場合、コマンドは同じですが、サービス名は異なります:

```
shell> systemctl start mysql
```

オペレーティングシステムが `systemd` に対応している場合は、`stop`、`start`、`status` などの標準 `systemctl` (または引数を逆にした `service`) コマンドと `restart` を使用して MySQL サーバサービスを管理する必要があります。 `mysqld` サービスはデフォルトで有効になっており、システムの再起動時に開始されます。 `systemd` プラットフォームで動作が異なる場合があることに注意してください: たとえば、データディレクトリの場所を変更すると、問題が発生する可能性があります。詳細は、[セクション 2.5.9 「systemd を使用した MySQL Server の管理」](#) を参照してください。

RPM および DEB パッケージを使用したアップグレードインストール中に、アップグレードの発生時に MySQL サーバが実行されている場合は、MySQL サーバが停止し、アップグレードが発生して MySQL サーバが再起動されます。1 つの例外: アップグレード中にエディションも変更された場合 (コミュニティから商用、またはその逆)、MySQL サーバは再起動されません。

サーバの最初の起動時に、サーバのデータディレクトリが空の場合、次のようになります:

- サーバが初期化されます。
- SSL 証明書およびキーファイルがデータディレクトリに生成されます。
- `validate_password` がインストールされ、有効になっています。
- スーパーユーザーアカウント `'root'@'localhost'` が作成されます。スーパーユーザーのパスワードが設定され、エラーログファイルに格納されます。これを表示するには、RHEL、Oracle Linux、CentOS、および Fedora システムで次のコマンドを使用します:

```
shell> sudo grep 'temporary password' /var/log/mysqld.log
```

SLES システムでは、次のコマンドを使用します:

```
shell> sudo grep 'temporary password' /var/log/mysql/mysql.log
```

次のステップでは、生成された一時パスワードを使用してログインし、スーパーユーザーアカウントのカスタムパスワードを設定します:

```
shell> mysql -uroot -p
```

```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'MyNewPass4!';
```

注記

`validate_password` はデフォルトでインストールされます。 `validate_password` で実装されているデフォルトのパスワードポリシーでは、パスワードに大文字、小文字、数字および特殊文字を 1 文字以上含める必要があり、パスワードの合計長は 8 文字以上である必要があります。

インストール中に問題が発生した場合は、エラーログファイル `/var/log/mysql.log` にデバッグ情報が記録されている可能性があります。

一部の Linux ディストリビューションでは、`mysqld` で使用可能なファイルディスクリプタの数の制限を増やす必要がある場合があります。 [セクション B.3.2.16 「ファイルが見つからず同様のエラーが発生しました」](#) を参照してください

複数の MySQL バージョンからのクライアントライブラリのインストール。 以前のライブラリに対してリンクされた古いアプリケーションとの互換性を維持する場合などに、複数のクライアントライブラリバージョンをインストールできます。 古いクライアントライブラリをインストールするには、`rpm` で `--oldpackage` オプションを使用します。 たとえば、MySQL 8.0 の `libmysqlclient.21` を含む EL6 システムに `mysql-community-libs-5.5` をインストールするには、次のようなコマンドを使用します:

```
shell> rpm --oldpackage -ivh mysql-community-libs-5.5.50-2.el6.x86_64.rpm
```

デバッグパッケージ。 `debug package` でコンパイルされた MySQL Server の特別なバリエーションがサーバー RPM パッケージに含まれています。 デバッグおよびメモリ割当てチェックを実行し、サーバーの実行中にトレースファイルを生成します。 そのデバッグバージョンを使用するには、MySQL をサービスまたは `/usr/sbin/mysqld` で起動するかわりに、`/usr/sbin/mysqld-debug` で起動します。 使用できるデバッグオプションについては、[セクション 5.9.4 「DBUG パッケージ」](#) を参照してください。

注記

デバッグビルドのデフォルトプラグインディレクトリが、MySQL 8.0.4 で `/usr/lib64/mysql/plugin` から `/usr/lib64/mysql/plugin/debug` に変更されました。 以前は、デバッグビルドのために `plugin_dir` を `/usr/lib64/mysql/plugin/debug` に変更する必要がありました。

ソース SRPM から RPM を再構築中。 MySQL のソースコード SRPM パッケージをダウンロードできます。 これらをそのまま使用して、標準の `rpmbuild` ツールチェーンで MySQL RPM を再構築できます。

2.5.5 オラクルからの Debian パッケージを使用して MySQL を Linux にインストールする

オラクルは、MySQL を Debian や Debian のと類似の Linux システムにインストールするための Debian パッケージを提供します。 このパッケージは 2 種類のチャネルから入手可能です。

- 「[MySQL APT リポジトリ](#)」。 これは、MySQL 製品をインストールおよび更新するための簡単で便利な方法を提供するため、Debian に似たシステムに MySQL をインストールする場合に推奨される方法です。 詳細は、[セクション 2.5.2 「MySQL APT リポジトリを使用して MySQL を Linux にインストールする」](#) を参照してください。
- [MySQL Developer Zone のダウンロード領域](#)。 詳細は、[セクション 2.1.3 「MySQL の取得方法」](#) を参照してください。 そこで入手可能な Debian パッケージと、インストールの手順についての情報を次に示します。
- MySQL Developer Zone には、MySQL の様々なコンポーネントを現在の Debian および Ubuntu プラットフォームにインストールするための様々な Debian パッケージが用意されています。 推奨される方法は tarball バンド

ルを使用するものです。これには MySQL の基本的なセットアップに必要なパッケージが含まれます。tarball バンドルは、`mysql-server_MVER-DVER_CPU.deb-bundle.tar` という形式の名前を持ちます。MVER は MySQL のバージョン、DVER は Linux 配布のバージョンです。次の表に示すように、CPU 値は、パッケージのビルド対象になっているプロセッサのタイプまたはファミリを示します。

表 2.13 MySQL Debian および Ubuntu インストールパッケージの CPU 識別子

CPU 値	対象のプロセッサタイプまたはファミリ
i386	Pentium プロセッサ以上、32 ビット
amd64	64 ビットの x86 プロセッサ

- tarball のダウンロード後、次のコマンドでアンパックします。

```
shell> tar -xvf mysql-server_MVER-DVER_CPU.deb-bundle.tar
```

- システムに `libaio` ライブラリがまだ存在しない場合は、インストールが必要になることがあります:

```
shell> sudo apt-get install libaio1
```

- 次のコマンドを使用して、MySQL サーバーパッケージを事前構成します:

```
shell> sudo dpkg-preconfigure mysql-community-server_*.deb
```

MySQL インストールの root ユーザーのパスワードを入力するよう求められます。インストールに関するその他の質問が表示される場合もあります。

重要

設定した root パスワードを必ず記憶してください。後でパスワードを設定するユーザーは、ダイアログボックスの password フィールドを空白のままにして、単に OK を押すことができます。その場合、サーバーへのルートアクセスは、Unix ソケットファイルを使用した接続用に [MySQL Socket Peer-Credential Authentication Plugin](#) を使用して認証されます。root パスワードは、後で `mysql_secure_installation` を使用して設定できます。

- MySQL サーバーの基本インストールの場合は、データベース共通ファイルパッケージ、クライアントパッケージ、クライアントメタパッケージ、サーバーパッケージおよびサーバーメタパッケージを (この順序で) インストールします。これは、単一のコマンドを使用して実行できます:

```
shell> sudo dpkg -i mysql-{common,community-client,client,community-server,server}_*.deb
```

パッケージ名には、`server-core` および `client-core` を含むパッケージもあります。これらにはバイナリのみが含まれ、標準パッケージによって自動的にインストールされます。これらを単独でインストールしても、MySQL の設定は機能しません。

`dpkg` によって満たされていない依存関係が警告された場合は、`apt-get` を使用して修正できます:

```
sudo apt-get -f install
```

ファイルがシステムにインストールされる場所は次のとおりです。

- (`my.cnf` などの) すべての構成ファイルは `/etc/mysql` の下にあります
- すべてのバイナリ、ライブラリ、ヘッダーなどは、`/usr/bin` および `/usr/sbin` の下にあります
- データディレクトリは `/var/lib/mysql` の下にあります

注記

MySQL の Debian 配布はほかのベンダーからも提供されています。それらは、オラクルによってビルドされたものとは (通信セットアップなどの) 機能や規則が異なる場合があり、それらのインストールにはこのマニュアルの説明が必ずしも適用されないことに注意してください。代わりに、ベンダーの説明書を参照してください。

2.5.6 Docker での Linux への MySQL のデプロイ

Docker デプロイメントフレームワークは、MySQL Server の簡単なインストールおよび構成をサポートしています。このセクションでは、MySQL Server Docker イメージの使用方法について説明します。

MySQL Server Docker イメージを使用する前に、システムに Docker がインストールされている必要があります。手順については、「[Docker のインストール](#)」を参照してください。

重要

`sudo` で `docker` コマンドを実行するか、`docker` ユーザーグループを作成してから、`docker` コマンドを実行するユーザーを追加する必要があります。詳細は、「[ここ](#)」を参照してください。Docker コンテナは常に `root` 権限で実行されるため、「[Docker デーモン攻撃対象領域](#)」を理解し、関連するリスクを適切に軽減する必要があります。

2.5.6.1 Docker を使用した MySQL Server デプロイメントの基本ステップ

警告

MySQL チームによって管理される MySQL Docker イメージは、Linux プラットフォーム専用に構築されています。他のプラットフォームはサポートされておらず、これらの MySQL Docker イメージを使用するユーザーは独自のリスクでこれを実行しています。Linux 以外のオペレーティングシステムでこれらのコンテナを実行する際の既知の制限については、[the discussion here](#) を参照してください。

- [MySQL Server Docker イメージのダウンロード](#)
- [MySQL Server インスタンスの起動](#)
- [コンテナ内から MySQL Server への接続](#)
- [コンテナシェルアクセス](#)
- [MySQL コンテナの停止および削除](#)
- [MySQL Server コンテナのアップグレード](#)
- [Docker での MySQL Server のデプロイに関するその他のトピック](#)

MySQL Server Docker イメージのダウンロード

別のステップでサーバーイメージをダウンロードする必要は厳密にはありませんが、Docker コンテナを作成する前にこのステップを実行すると、ローカルイメージが最新の状態になります。MySQL Community Edition イメージをダウンロードするには、次のコマンドを実行します:

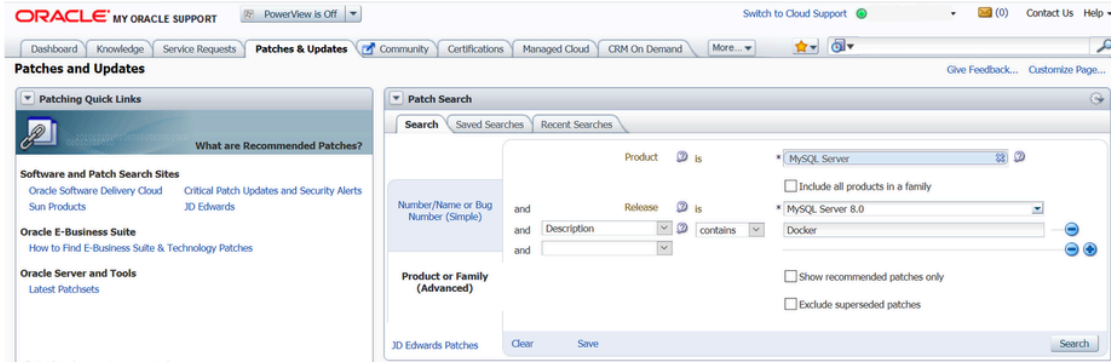
```
docker pull mysql/mysql-server:tag
```

`tag` は、プルするイメージバージョン (5.6, 5.7, 8.0 や `latest` など) のラベルです。:tag を省略すると、`latest` ラベルが使用され、MySQL Community Server の最新の GA バージョンのイメージがダウンロードされます。「[Docker Hub の mysql/mysql-server ページ](#)」で使用可能なバージョンについては、タグのリストを参照してください。

MySQL Enterprise Edition イメージをダウンロードするには、[My Oracle Support](#) の web サイトにアクセスし、Oracle アカウントにサインインして、ランディングページで次のステップを実行します:

- 「パッチと更新版」タブを選択します。
- 「パッチ検索」リージョンに移動し、「検索」タブで「製品またはファミリ (拡張)」サブタブに切り替えます。
- Product フィールドに「MySQL Server」と入力し、「リリース」フィールドに目的のバージョン番号を入力します。
- 追加フィルタのドロップダウンを使用して「説明」を選択 - 「次を含む」で、テキストフィールドに「Docker」と入力します。

次の図に、MySQL Server 8.0 用の MySQL Enterprise Edition イメージの検索設定を示します:



- 検索ボタンをクリックし、結果リストから目的のバージョンを選択してダウンロードボタンをクリックします。
- 表示される「ファイルのダウンロード」ダイアログボックスで、Docker イメージの .zip ファイルをクリックしてダウンロードします。

ダウンロードした .zip アーカイブを解凍して (mysql-enterprise-server-version.tar) 内の tarball を取得し、次のコマンドを実行してイメージをロードします:

```
docker load -i mysql-enterprise-server-version.tar
```

次のコマンドを使用して、ダウンロードした Docker イメージをリストできます:

```
shell> docker images
REPOSITORY          TAG          IMAGE ID        CREATED        SIZE
mysql/mysql-server  latest      3157d7f55f8d   4 weeks ago    241MB
```

MySQL Server インスタンスの起動

MySQL Server の新しい Docker コンテナを起動するには、次のコマンドを使用します:

```
docker run --name=container_name --restart on-failure -d image_name:tag
```

MySQL Server Docker イメージのダウンロード で説明されているように、イメージ名は `docker images` コマンドを使用して取得できます。

サーバーコンテナのカスタム名を指定するための `--name` オプションはオプションです。コンテナ名が指定されていない場合はランダムに生成されます。

`--restart` オプションは、コンテナの「再起動ポリシー」を構成するためのものです。クライアントセッション内でのサーバーの再起動のサポートを有効にするには、値 `on-failure` に設定する必要があります (たとえば、`RESTART` ステートメントがクライアントによって実行された場合や [configuration of an InnoDB cluster instance](#) 中に実行された場合)。再起動のサポートを有効にして、クライアントセッション内で再起動を発行すると、サーバーとコンテナが停止してから再起動します。サーバーの再起動のサポートは、MySQL 8.0.21 以降で使用できます。

たとえば、MySQL Community Server の新しい Docker コンテナを起動するには、次のコマンドを使用します:

```
docker run --name=mysql1 --restart on-failure -d mysql/mysql-server:8.0
```

My Oracle Support からダウンロードした Docker イメージを使用して MySQL Enterprise Server の新しい Docker コンテナを起動するには、次のコマンドを使用します:

```
docker run --name=mysql1 --restart on-failure -d mysql/enterprise-server:8.0
```

指定した名前およびタグの Docker イメージが以前の `docker pull` または `docker run` コマンドによってダウンロードされていない場合、イメージはダウンロードされます。コンテナの初期化が開始され、`docker ps` コマンドを実行すると、実行中のコンテナのリストにコンテナが表示されます。例:

```
shell> docker ps
CONTAINER ID  IMAGE          COMMAND                  CREATED        STATUS                    PORTS          NAMES
a24888f0d6f4  mysql/mysql-server  "/entrypoint.sh my..."  14 seconds ago  Up 13 seconds (health: starting)  3306/tcp, 33060/tcp  mysql1
```

コンテナの初期化には時間がかかる場合があります。サーバーを使用する準備が整うと、`docker ps` コマンドの出力のコンテナの `STATUS` が `(health: starting)` から `(healthy)` に変わります。

前述の `docker run` コマンドで `-d` オプションを使用すると、コンテナがバックグラウンドで実行されます。次のコマンドを使用して、コンテナからの出力を監視します:

```
docker logs mysql1
```

初期化が終了すると、コマンド出力には root ユーザー用に生成されたランダムなパスワードが含まれます。たとえば、次のコマンドを使用してパスワードを確認します:

```
shell> docker logs mysql1 2>&1 | grep GENERATED  
GENERATED ROOT PASSWORD: Axegh3kAJyDLaRuBemecis&EShOs
```

コンテナ内から MySQL Server への接続

サーバーの準備ができたら、起動した MySQL Server コンテナ内で `mysql` クライアントを実行し、MySQL Server に接続できます。次のように、`docker exec -it` コマンドを使用して、起動した Docker コンテナ内の `mysql` クライアントを起動します:

```
docker exec -it mysql1 mysql -uroot -p
```

要求されたら、生成された root パスワードを入力します (パスワードの検索方法については、前述の [MySQL Server インスタンスの起動](#) の最後のステップを参照してください)。`MYSQL_ONETIME_PASSWORD` オプションはデフォルトで `true` であるため、`mysql` クライアントをサーバーに接続したあと、次のステートメントを発行してサーバーの root パスワードをリセットする必要があります:

```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'password';
```

`password` を任意のパスワードに置き換えます。パスワードがリセットされると、サーバーは使用可能になります。

コンテナシェルアクセス

MySQL Server コンテナにシェルアクセスできるようにするには、`docker exec -it` コマンドを使用してコンテナ内で `bash` シェルを起動します:

```
shell> docker exec -it mysql1 bash  
bash-4.2#
```

その後、コンテナ内で Linux コマンドを実行できます。たとえば、コンテナ内のサーバーデータディレクトリのコンテンツを表示するには、次のコマンドを使用します:

```
bash-4.2# ls /var/lib/mysql  
auto.cnf  ca.pem  client-key.pem  ib_logfile0  ibdata1  mysql  mysql.sock.lock  private_key.pem  server-cert.pem  sys  
ca-key.pem  client-cert.pem  ib_buffer_pool  ib_logfile1  ibtmp1  mysql.sock  performance_schema  public_key.pem  server-key.pem
```

MySQL コンテナの停止および削除

作成した MySQL Server コンテナを停止するには、次のコマンドを使用します:

```
docker stop mysql1
```

`docker stop` は `SIGTERM` シグナルを `mysqld` プロセスに送信するため、サーバーは正常にシャットダウンされます。

また、コンテナ (MySQL Server コンテナの場合は `mysqld`) のメインプロセスが停止すると、Docker コンテナは自動的に停止します。

MySQL Server コンテナを再起動するには:

```
docker start mysql1
```

単一のコマンドを使用して MySQL Server コンテナを停止して再起動するには:

```
docker restart mysql1
```

MySQL コンテナを削除するには、まず停止してから、`docker rm` コマンドを使用します:

```
docker stop mysql1
```

```
docker rm mysql1
```

`Docker volume for the server data directory` を同時に削除する場合は、`docker rm` コマンドに `-v` オプションを追加します。

MySQL Server コンテナのアップグレード

重要

- MySQL へのアップグレードを実行する前に、[セクション2.11「MySQL のアップグレード」](#)の手順に注意してください。ここで説明されているその他の手順の中では、アップグレードの前にデータベースをバックアップすることが特に重要です。
- このセクションの手順では、サーバーのデータと構成がホストに保持されている必要があります。詳細は、[データおよび構成の変更の永続化](#)を参照してください。

MySQL 5.7 の Docker インストールを 8.0 にアップグレードするには、次のステップに従います:

- MySQL 5.7 サーバーを停止します (この例では、コンテナ名は `mysql57`):

```
docker stop mysql57
```

- MySQL 8.0 Server Docker イメージをダウンロードします。 [MySQL Server Docker イメージのダウンロード](#) の手順を参照してください。MySQL 8.0 には正しいタグを使用してください。
- (この例では [bind-mounting](#) によって) ホストに永続化されている古いサーバーデータおよび構成 (必要に応じて [セクション2.11「MySQL のアップグレード」](#) を参照) を使用して、新しい MySQL 8.0 Docker コンテナ (この例では `mysql80`) を起動します。MySQL Community Server の場合は、次のコマンドを実行します:

```
docker run --name=mysql80 \
--mount type=bind,src=/path-on-host-machine/my.cnf,dst=/etc/my.cnf \
--mount type=bind,src=/path-on-host-machine/datadir,dst=/var/lib/mysql \
-d mysql/mysql-server:8.0
```

必要に応じて、`mysql/mysql-server` を正しいリポジトリ名に調整し、次のように置き換えます [My Oracle Support](#) からダウンロードされた `mysql/enterprise-server` for MySQL Enterprise Edition イメージ。

- サーバーの起動が終了するまで待機します。サーバーのステータスは、`docker ps` コマンドを使用して確認できます (その方法は、[MySQL Server インスタンスの起動](#)を参照してください)。
- MySQL 8.0.15 以前の場合: MySQL 8.0 Server コンテナで `mysql_upgrade` ユーティリティを実行します (MySQL 8.0.16 以降では不要):

```
docker exec -it mysql80 mysql_upgrade -uroot -p
```

プロンプトが表示されたら、古い MySQL 5.7 Server の root パスワードを入力します。

- MySQL 8.0 Server コンテナを再起動して、アップグレードを終了します:

```
docker restart mysql80
```

Docker での MySQL Server のデプロイに関するその他のトピック

サーバー構成、データと構成の永続化、サーバーエラーログおよびコンテナ環境変数など、Docker を使用した MySQL Server のデプロイの詳細は、[セクション2.5.6.2「Docker での MySQL Server のデプロイに関するその他のトピック」](#)を参照してください。

2.5.6.2 Docker での MySQL Server のデプロイに関するその他のトピック

注記

次のほとんどのサンプルコマンドでは、(`docker pull` コマンドや `docker run` コマンドなどを使用して) 指定する必要がある場合、`mysql/mysql-server` が Docker イメージリポジトリとして使用されます。イメージが別のリポジトリからのものである場合は、次のように変更 [My Oracle Support](#) からダウンロードされた `mysql/enterprise-server` for MySQL Enterprise Edition イメージ。

- [Docker 用に最適化された MySQL インストール](#)
- [MySQL Server の構成](#)

- [データおよび構成の変更の永続化](#)
- [追加の初期化スクリプトの実行](#)
- [別の Docker コンテナ内のアプリケーションから MySQL への接続](#)
- [サーバーエラーログ](#)
- [Docker での MySQL Enterprise Backup の使用](#)
- [既知の問題](#)
- [Docker の環境変数](#)

Docker 用に最適化された MySQL インストール

MySQL 用の Docker イメージはコードサイズ用に最適化されています。つまり、Docker コンテナで MySQL インスタンスを実行する大部分のユーザーに関連することが予想される重要なコンポーネントのみが含まれています。MySQL Docker のインストールは、次の点で共通の non-Docker のインストールとは異なります:

- 含まれるバイナリは、次のものに制限されます:
 - `/usr/bin/my_print_defaults`
 - `/usr/bin/mysql`
 - `/usr/bin/mysql_config`
 - `/usr/bin/mysql_install_db`
 - `/usr/bin/mysql_tzinfo_to_sql`
 - `/usr/bin/mysql_upgrade`
 - `/usr/bin/mysqladmin`
 - `/usr/bin/mysqlcheck`
 - `/usr/bin/mysqldump`
 - `/usr/bin/mysqldump`
 - `/usr/bin/mysqlbackup` (MySQL Enterprise Edition 8.0 の場合のみ)
 - `/usr/sbin/mysqld`
- すべてのバイナリが削除され、デバッグ情報は含まれません。

MySQL Server の構成

MySQL Docker コンテナを起動するときに、`docker run` コマンドを使用して構成オプションをサーバーに渡すことができます。例:

```
docker run --name mysql1 -d mysql/mysql-server:tag --character-set-server=utf8mb4 --collation-server=utf8mb4_col
```

このコマンドは、`utf8mb4` をデフォルトの文字セットとして、`utf8mb4_col` をデータベースのデフォルトの照合順序として使用して、MySQL Server を起動します。

MySQL Server を構成する別の方法は、構成ファイルを準備し、コンテナ内のサーバー構成ファイルの場所にマウントすることです。詳細は、[データおよび構成の変更の永続化](#) を参照してください。

データおよび構成の変更の永続化

Docker コンテナは原則的に一時的なものであり、コンテナが削除または破損した場合、データまたは構成は失われることが予想されます (ディスクカッション「[ここ](#)」を参照)。ただし、「[Docker ポリウム](#)」には、Docker コンテ

ナ内で作成されたデータを永続化するメカニズムが用意されています。初期化時に、MySQL Server コンテナはサーバーデータディレクトリの Docker ボリュームを作成します。コンテナで `docker inspect` コマンドを実行するための JSON 出力には `Mount` キーがあり、その値によってデータディレクトリボリュームに関する情報が提供されます:

```
shell> docker inspect mysql1
...
"Mounts": [
  {
    "Type": "volume",
    "Name": "4f2d463cfc4bdd4bae9cb098c97d7da3337195ed2c6572bc0b89f7e845d27652",
    "Source": "/var/lib/docker/volumes/4f2d463cfc4bdd4bae9cb098c97d7da3337195ed2c6572bc0b89f7e845d27652/_data",
    "Destination": "/var/lib/mysql",
    "Driver": "local",
    "Mode": "",
    "RW": true,
    "Propagation": ""
  }
],
...
```

出力には、データがホストに永続化されるソースフォルダ `/var/lib/docker/volumes/4f2d463cfc4bdd4bae9cb098c97d7da3337195ed2c6572bc0b89f7e845d27652/_data` が、コンテナ内のサーバーデータディレクトリである `/var/lib/mysql` にマウントされていることが示されています。

データを保持する別の方法は、コンテナの作成時に `--mount` オプションを使用してホストディレクトリを `bind-mount` に保存することです。同じ方法を使用して、サーバーの構成を永続化できます。次のコマンドは、MySQL Server コンテナを作成し、データディレクトリとサーバー構成ファイルの両方をバインドマウントします:

```
docker run --name=mysql1 \
--mount type=bind,src=/path-on-host-machine/my.cnf,dst=/etc/my.cnf \
--mount type=bind,src=/path-on-host-machine/datadir,dst=/var/lib/mysql \
-d mysql/mysql-server:tag
```

このコマンドは、`path-on-host-machine/my.cnf` を `/etc/my.cnf` (コンテナ内のサーバー構成ファイル) にマウントし、`path-on-host-machine/datadir` を `/var/lib/mysql` (コンテナ内のデータディレクトリ) にマウントします。バインドマウントが機能するには、次の条件を満たす必要があります:

- 構成ファイル `path-on-host-machine/my.cnf` がすでに存在し、ユーザー `mysql` を使用してサーバーを起動するための仕様が含まれている必要があります:

```
[mysql]
user=mysql
```

ファイルに他のサーバー構成オプションを含めることもできます。

- データディレクトリ `path-on-host-machine/datadir` がすでに存在している必要があります。サーバーの初期化を実行するには、ディレクトリが空である必要があります。データが事前移入されたディレクトリをマウントしてサーバーを起動することもできますが、データを作成したサーバーと同じ構成で Docker コンテナを起動する必要があり、コンテナの起動時に必要なホストファイルまたはディレクトリがマウントされていることを確認する必要があります。

追加の初期化スクリプトの実行

作成直後にデータベースで実行する `.sh` または `.sql` スクリプトがある場合は、それらをホストディレクトリに配置してから、そのディレクトリをコンテナ内の `/docker-entrypoint-initdb.d/` にマウントできます。例:

```
docker run --name=mysql1 \
--mount type=bind,src=/path-on-host-machine/scripts/,dst=/docker-entrypoint-initdb.d/ \
-d mysql/mysql-server:tag
```

別の Docker コンテナ内のアプリケーションから MySQL への接続

Docker ネットワークを設定すると、別の Docker コンテナ内のクライアントアプリケーションがサーバーコンテナ内の MySQL Server にアクセスできるように、複数の Docker コンテナが相互に通信できるようになります。まず、Docker ネットワークを作成します:

```
docker network create my-custom-net
```


次に、サーバーおよびクライアントコンテナを作成して起動するときに、`--network` オプションを使用して、作成したネットワークに配置します。例:

```
docker run --name=mysql1 --network=my-custom-net -d mysql/mysql-server
```

```
docker run --name=myapp1 --network=my-custom-net -d myapp
```

その後、Docker は指定されたコンテナ名に対して DNS を自動的に設定するため、`myapp1` コンテナは `mysql1` ホスト名を使用して `mysql1` コンテナに接続できます (逆も同様です)。次の例では、`myapp1` コンテナ内から `mysql` クライアントを実行して、独自のコンテナ内のホスト `mysql1` に接続します:

```
docker exec -it myapp1 mysql --host=mysql1 --user=myuser --password
```

コンテナのその他のネットワーク技術については、Docker ドキュメントの「[Docker コンテナネットワークング](#)」のセクションを参照してください。

サーバーエラーログ

サーバーコンテナを使用して MySQL Server を初めて起動したときに、次のいずれかの条件に該当する場合、`server error log` は生成されません:

- ホストからのサーバー構成ファイルがマウントされていますが、ファイルにシステム変数 `log_error` が含まれていません (サーバー構成ファイルのバインドおよびマウントに関する [データおよび構成の変更の永続化](#) を参照)。
- ホストからのサーバー構成ファイルはマウントされていませんが、Docker 環境変数 `MYSQL_LOG_CONSOLE` は `true` (MySQL 8.0 サーバーコンテナの変数のデフォルト状態) です。その後、MySQL Server エラーログは `stderr` にリダイレクトされるため、エラーログは Docker コンテナログに記録され、`docker logs mysqlid-container` コマンドを使用して表示できます。

いずれかの条件に該当する場合に MySQL Server でエラーログを生成するには、[configure the server](#) の `--log-error` オプションを使用して、コンテナ内の特定の場所にエラーログを生成します。エラーログを永続化するには、[データおよび構成の変更の永続化](#) の説明に従って、コンテナ内のエラーログの場所にホストファイルをマウントします。ただし、コンテナ内の MySQL Server に、マウントされたホストファイルへの書き込みアクセス権があることを確認する必要があります。

Docker での MySQL Enterprise Backup の使用

[MySQL Enterprise Backup](#) は、[MySQL Enterprise Edition](#) で使用可能な MySQL Server の商用ライセンスバックアップユーティリティです。MySQL Enterprise Backup は MySQL Enterprise Edition の Docker インストールに含まれています。

次の例では、Docker コンテナで MySQL Server がすでに実行されていることを前提としています (Docker で MySQL Server インスタンスを起動する方法については、[セクション 2.5.6.1 「Docker を使用した MySQL Server デプロイメントの基本ステップ」](#) を参照してください)。MySQL Enterprise Backup で MySQL Server をバックアップするには、サーバーデータディレクトリにアクセスできる必要があります。これは、たとえば、サーバーの起動時に [binding a host directory on the data directory of the MySQL Server](#) によって実現できます:

```
docker run --name=mysqlserver \
--mount type=bind,src=/path-on-host-machine/datadir,dst=/var/lib/mysql \
-d mysql/enterprise-server:8.0
```

このコマンドを使用すると、MySQL Server は MySQL Enterprise Edition の Docker イメージで起動され、ホストディレクトリ `/path-on-host-machine/datadir` はサーバーコンテナ内のサーバーデータディレクトリ (`/var/lib/mysql`) にマウントされます。また、サーバーの起動後、MySQL Enterprise Backup がサーバーにアクセスするために必要な権限も設定されていることを前提としています (詳細は、[Grant MySQL Privileges to Backup Administrator](#) を参照)。次のステップを使用して、MySQL Server インスタンスをバックアップおよびリストアします。

Docker とともに MySQL Enterprise Backup を使用して、Docker コンテナで実行されている MySQL Server インスタンスをバックアップするには:

1. MySQL Server コンテナが実行されているのと同じホストで、MySQL Enterprise Edition のイメージを使用して別のコンテナを起動し、MySQL Enterprise Backup コマンド `backup-to-image` を使用してバックアップを実行します。最後のステップで作成したバインドマウントを使用して、サーバーデータディレクトリへのアクセスを提供

します。また、ホストディレクトリ (この例では `/path-on-host-machine/backups/`) をコンテナ内のバックアップの記憶域フォルダ (この例では `/data/backups`) にマウントして、作成中のバックアップを永続化します。次に、[My Oracle Support](#) からダウンロードした Docker イメージを使用して MySQL Enterprise Backup を起動する、このステップのサンプルコマンドを示します:

```
shell> docker run \
--mount type=bind,src=/path-on-host-machine/datadir,dst=/var/lib/mysql \
--mount type=bind,src=/path-on-host-machine/backups/,dst=/data/backups \
--rm mysql/enterprise-server:8.0 \
mysqlbackup -umysqlbackup -ppassword --backup-dir=/tmp/backup-tmp --with-timestamp \
--backup-image=/data/backups/db.mbi backup-to-image

[Entrypoint] MySQL Docker Image 8.0.11-1.1.5
MySQL Enterprise Backup version 8.0.11 Linux-4.1.12-61.1.16.el7uek.x86_64-x86_64 [2018-04-08 07:06:45]
Copyright (c) 2003, 2018, Oracle and/or its affiliates. All Rights Reserved.

180921 17:27:25 MAIN INFO: A thread created with Id '140594390935680'
180921 17:27:25 MAIN INFO: Starting with following command line ...
...

-----
Parameters Summary
-----
Start LSN      : 29615616
End LSN        : 29651854
-----

mysqlbackup completed OK!
```

`mysqlbackup` による出力の終わりをチェックして、バックアップが正常に完了していることを確認することが重要です。

2. バックアップジョブが終了するとコンテナは終了し、開始に使用した `--rm` オプションを使用すると、コンテナは終了後に削除されます。イメージバックアップが作成され、バックアップを格納する最後のステップでマウントされたホストディレクトリにあります:

```
shell> ls /tmp/backups
db.mbi
```

MySQL Enterprise Backup と Docker を使用して Docker コンテナ内の MySQL Server インスタンスをリストアするには:

1. MySQL Server コンテナを停止します。これにより、内部で実行されている MySQL Server も停止されます:

```
docker stop mysqlserver
```

2. ホストで、MySQL Server データディレクトリの bind マウント内のすべての内容を削除します:

```
rm -rf /path-on-host-machine/datadir/*
```

3. MySQL Enterprise Edition のイメージを使用してコンテナを起動し、MySQL Enterprise Backup コマンド `copy-back-and-apply-log` を使用してリストアを実行します。サーバーをバックアップしたときと同様に、バックアップのサーバーデータディレクトリとストレージフォルダをバインド/マウントします:

```
shell> docker run \
--mount type=bind,src=/path-on-host-machine/datadir,dst=/var/lib/mysql \
--mount type=bind,src=/path-on-host-machine/backups/,dst=/data/backups \
--rm mysql/enterprise-server:8.0 \
mysqlbackup --backup-dir=/tmp/backup-tmp --with-timestamp \
--datadir=/var/lib/mysql --backup-image=/data/backups/db.mbi copy-back-and-apply-log

[Entrypoint] MySQL Docker Image 8.0.11-1.1.5
MySQL Enterprise Backup version 8.0.11 Linux-4.1.12-61.1.16.el7uek.x86_64-x86_64 [2018-04-08 07:06:45]
Copyright (c) 2003, 2018, Oracle and/or its affiliates. All Rights Reserved.

180921 22:06:52 MAIN INFO: A thread created with Id '139768047519872'
180921 22:06:52 MAIN INFO: Starting with following command line ...
...
180921 22:06:52 PCR1 INFO: We were able to parse ibbackup_logfile up to
lsn 29680612.
180921 22:06:52 PCR1 INFO: Last MySQL binlog file position 0 155, file name binlog.000003
```

```
180921 22:06:52 PCR1 INFO: The first data file is '/var/lib/mysql/ibdata1'
and the new created log files are at '/var/lib/mysql/'
180921 22:06:52 MAIN INFO: No Keyring file to process.
180921 22:06:52 MAIN INFO: Apply-log operation completed successfully.
180921 22:06:52 MAIN INFO: Full Backup has been restored successfully.

mysqlbackup completed OK! with 3 warnings
```

バックアップジョブが終了するとコンテナは終了し、開始時に `--rm` オプションを使用すると、コンテナは終了後に削除されます。

4. サーバーコンテナを再起動します。これにより、リストアされたサーバーも再起動されます:

```
docker restart mysqlserver
```

または、リストアされたデータディレクトリで新しい MySQL Server を起動します:

```
docker run --name=mysqlserver2 \
--mount type=bind,src=/path-on-host-machine/datadir,dst=/var/lib/mysql \
-d mysql/enterprise-server:8.0
```

サーバーにログオンして、リストアされたデータでサーバーが実行されていることを確認します。

既知の問題

- サーバーシステム変数 `audit_log_file` を使用して監査ログファイル名を構成する場合、それとともに `loose option modifier` を使用すると、Docker はサーバーを起動できません。

Docker の環境変数

MySQL Server コンテナを作成する場合、`--env` オプション (`-e`) を使用して、次の環境変数のいずれかまたは複数指定することで、MySQL インスタンスを構成できます。

メモ

- サーバーの初期化が試行されないため、マウントするデータディレクトリが空でない場合、次の変数は何の効果もありません (詳細は [データおよび構成の変更の永続化](#) を参照)。フォルダ内の既存のコンテンツ (古いサーバー設定を含む) は、コンテナの起動時に変更されません。
- `MYSQL_RANDOM_ROOT_PASSWORD`、`MYSQL_ONETIME_PASSWORD`、`MYSQL_ALLOW_EMPTY_PASSWORD` などのブール変数は、長さがゼロ以外の文字列で設定することによって true になります。したがって、「0」、「false」または `no` などに設定しても、false にはなりません、実際には true になります。これは、MySQL Server コンテナの既知の問題です。
- `MYSQL_RANDOM_ROOT_PASSWORD`: この変数が true (`MYSQL_ROOT_PASSWORD` が設定されているか、`MYSQL_ALLOW_EMPTY_PASSWORD` が true に設定されていないかぎり、デフォルトの状態) の場合、Docker コンテナの起動時にサーバールートユーザーのランダムパスワードが生成されます。パスワードはコンテナの `stdout` に出力され、コンテナのログを参照して確認できます ([MySQL Server インスタンスの起動](#) を参照)。
- `MYSQL_ONETIME_PASSWORD`: 変数が true (`MYSQL_ROOT_PASSWORD` が設定されているか、`MYSQL_ALLOW_EMPTY_PASSWORD` が true に設定されていないかぎり、デフォルトの状態) の場合、root ユーザーのパスワードは期限切れとして設定され、MySQL を通常どおり使用する前に変更する必要があります。
- `MYSQL_DATABASE`: この変数を使用すると、イメージの起動時に作成するデータベースの名前を指定できます。`MYSQL_USER` および `MYSQL_PASSWORD` でユーザー名とパスワードが指定されている場合、ユーザーが作成され、このデータベース (`GRANT ALL` に対応) へのスーパーユーザーアクセス権が付与されます。指定されたデータベースは `CREATE DATABASE IF NOT EXIST` ステートメントによって作成されるため、データベースがすでに存在する場合、変数は無効です。
- `MYSQL_USER`、`MYSQL_PASSWORD`: これらの変数は、ユーザーを作成してそのユーザーパスワードを設定するために組み合わせて使用され、ユーザーには `MYSQL_DATABASE` 変数で指定されたデータベースに対するスーパーユーザー権限が付与されます。ユーザーを作成するには、`MYSQL_USER` と `MYSQL_PASSWORD` の両方が必要

です。2つの変数のいずれかが設定されていない場合、もう一方は無視されます。両方の変数が設定されているが、`MYSQL_DATABASE` が設定されていない場合、ユーザーは権限なしで作成されます。

注記

このメカニズムを使用してルートスーパーユーザーを作成する必要はありません。このスーパーユーザーは、`MYSQL_ALLOW_EMPTY_PASSWORD` が true でないかぎり、`MYSQL_ROOT_PASSWORD` および `MYSQL_RANDOM_ROOT_PASSWORD` の説明で説明されているメカニズムのいずれかによってデフォルトでパスワードが設定されて作成されます。

- `MYSQL_ROOT_HOST`: デフォルトでは、MySQL によって 'root'@'localhost' アカウントが作成されます。このアカウントは、[コンテナ内から MySQL Server への接続](#) で説明されているように、コンテナ内からのみ接続できます。他のホストからのルート接続を許可するには、この環境変数を設定します。たとえば、デフォルトの Docker ゲートウェイ IP である値 `172.17.0.1` では、コンテナを実行するホストマシンからの接続が許可されます。このオプションで使用できるエントリは 1 つのみですが、ワイルドカード (`MYSQL_ROOT_HOST=172.*.*` や `MYSQL_ROOT_HOST=%` など) を使用できます。
- `MYSQL_LOG_CONSOLE`: 変数が true (MySQL 8.0 サーバーコンテナのデフォルト状態) の場合、MySQL Server エラーログは `stderr` にリダイレクトされるため、エラーログは Docker コンテナログに記録され、`docker logs mysql-d-container` コマンドを使用して表示できます。

注記

ホストのサーバー構成ファイルがマウントされている場合、変数は無効です (構成ファイルのバインド時の [データおよび構成の変更の永続化](#) を参照)。

- `MYSQL_ROOT_PASSWORD`: この変数は、MySQL root アカウントに設定されるパスワードを指定します。

警告

コマンドラインでの MySQL root ユーザーパスワードの設定はセキュアではありません。パスワードを明示的に指定するかわりに、パスワードファイルのコンテナファイルパスを使用して変数を設定し、コンテナファイルパスにパスワードを含むホストからファイルをマウントすることもできます。パスワードファイルの場所がまだ公開されているため、これはそれでも安全ではありません。`MYSQL_RANDOM_ROOT_PASSWORD` と `MYSQL_ONETIME_PASSWORD` の両方のデフォルト設定を true にすることをお勧めします。

- `MYSQL_ALLOW_EMPTY_PASSWORD`。root ユーザーのパスワードを空白にしてコンテナを起動できるようにするには、true に設定します。

警告

この変数を true に設定すると、MySQL インスタンスを完全に保護せずに完全なスーパーユーザーアクセスを取得できるため、セキュアではありません。`MYSQL_RANDOM_ROOT_PASSWORD` と `MYSQL_ONETIME_PASSWORD` の両方のデフォルト設定を true にすることをお勧めします。

2.5.6.3 Docker を使用した Windows およびその他の Linux 以外のプラットフォームへの MySQL のデプロイ

警告

Oracle によって提供される MySQL Docker イメージは、Linux プラットフォーム専用構築されています。他のプラットフォームはサポートされておらず、Oracle から MySQL Docker イメージを実行しているユーザーは独自のリスクでこれを実行しています。このセクションでは、Linux 以外のプラットフォームでイメージを使用する場合の既知の問題について説明します。

Windows 上の Oracle から MySQL Server Docker イメージを使用する際の既知の問題は次のとおりです:

- コンテナの MySQL データディレクトリにバインドする場合 (詳細は [データおよび構成の変更の永続化](#) を参照)、`--socket` オプションを使用してサーバーソケットファイルの場所を MySQL データディレクトリ外の場所に設定する必要があります。そうしないと、サーバーの起動に失敗します。これは、Docker for Windows でファイルマウントを処理する方法では、ホストファイルをソケットファイルにバインドできないためです。

2.5.7 ネイティブソフトウェアリポジトリから MySQL を Linux にインストールする

多くの Linux 配布は、ネイティブソフトウェアリポジトリの中に 1 つのバージョンの MySQL Server、クライアントツール、および開発コンポーネントを含み、プラットフォームの標準パッケージ管理システムでインストールできます。このセクションでは、これらのパッケージ管理システムを使用して MySQL をインストールするための基本的な手順を説明します。

重要

ネイティブパッケージは多くの場合、使用可能な最新のリリースから数バージョン遅れています。通常、開発マイルストーンリリース (DMR) はネイティブリポジトリで使用可能になっていないため、インストールできません。次に進む前に、[セクション2.5「Linux に MySQL をインストールする」](#)に記述されているその他のインストールオプションを確認することが推奨されます。

配布固有の手順を次に示します。

- Red Hat Linux、Fedora、CentOS

注記

多くの Linux ディストリビューションでは、プラットフォーム固有のソフトウェアリポジトリではなく、MySQL Yum リポジトリを使用して MySQL をインストールできます。詳細は、[セクション2.5.1「MySQL Yum リポジトリを使用して MySQL を Linux にインストールする」](#)を参照してください。

Red Hat および同様の配布では、MySQL 配布はいくつかの個別のパッケージ (クライアントツール用の `mysql`、サーバーおよび関連ツール用の `mysql-server`、およびライブラリ用の `mysql-libs`) にわかれています。Perl や Python など、異なる言語および環境からの接続性を提供する場合は、ライブラリは必須です。

インストールするには、`yum` コマンドを使用してインストールするパッケージを指定します。例:

```
root-shell> yum install mysql mysql-server mysql-libs mysql-server
Loaded plugins: presto, refresh-packagekit
Setting up Install Process
Resolving Dependencies
--> Running transaction check
--> Package mysql.x86_64 0:5.1.48-2.fc13 set to be updated
--> Package mysql-libs.x86_64 0:5.1.48-2.fc13 set to be updated
--> Package mysql-server.x86_64 0:5.1.48-2.fc13 set to be updated
--> Processing Dependency: perl-DBD-MySQL for package: mysql-server-5.1.48-2.fc13.x86_64
--> Running transaction check
--> Package perl-DBD-MySQL.x86_64 0:4.017-1.fc13 set to be updated
--> Finished Dependency Resolution
```

Dependencies Resolved

```
=====
Package      Arch      Version      Repository    Size
=====
Installing:
mysql        x86_64    5.1.48-2.fc13  updates      889 k
mysql-libs   x86_64    5.1.48-2.fc13  updates      1.2 M
mysql-server x86_64    5.1.48-2.fc13  updates      8.1 M
Installing for dependencies:
perl-DBD-MySQL x86_64    4.017-1.fc13  updates      136 k
```

Transaction Summary

```
=====
Install    4 Package(s)
Upgrade   0 Package(s)
```



```
Total download size: 10 M
Installed size: 30 M
Is this ok [y/N]: y
Downloading Packages:
Setting up and reading Presto delta metadata
Processing delta metadata
Package(s) data still to download: 10 M
(1/4): mysql-5.1.48-2.fc13.x86_64.rpm | 889 kB 00:04
(2/4): mysql-libs-5.1.48-2.fc13.x86_64.rpm | 1.2 MB 00:06
(3/4): mysql-server-5.1.48-2.fc13.x86_64.rpm | 8.1 MB 00:40
(4/4): perl-DBD-MySQL-4.017-1.fc13.x86_64.rpm | 136 kB 00:00
-----
Total                201 kB/s | 10 MB 00:52
Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction
Installing  : mysql-libs-5.1.48-2.fc13.x86_64          1/4
Installing  : mysql-5.1.48-2.fc13.x86_64             2/4
Installing  : perl-DBD-MySQL-4.017-1.fc13.x86_64      3/4
Installing  : mysql-server-5.1.48-2.fc13.x86_64      4/4

Installed:
mysql.x86_64 0:5.1.48-2.fc13      mysql-libs.x86_64 0:5.1.48-2.fc13
mysql-server.x86_64 0:5.1.48-2.fc13

Dependency Installed:
perl-DBD-MySQL.x86_64 0:4.017-1.fc13

Complete!
```

MySQL および MySQL Server はすでにインストールされています。サンプル構成ファイルは `/etc/my.cnf` にインストールされます。MySQL サーバーを起動するには、`systemctl` を使用します:

```
shell> systemctl start mysqld
```

データベーステーブルがまだ存在しない場合は、自動的に作成されます。ただし、サーバーで `mysql_secure_installation` を実行して root のパスワードを設定してください。

- Debian、Ubuntu、Kubuntu

注記

サポートされている Debian および Ubuntu バージョンの場合、MySQL はプラットフォーム固有のソフトウェアリポジトリではなく「MySQL APT リポジトリ」を使用してインストールできます。詳細は、[セクション2.5.2「MySQL APT リポジトリを使用して MySQL を Linux にインストールする」](#)を参照してください。

Debian および関連する配布では、ソフトウェアリポジトリに MySQL のパッケージが `mysql-client` と `mysql-server` の 2 つあります。それぞれクライアントコンポーネントとサーバーコンポーネントです。希望するバージョンの MySQL を確実にインストールするため、`mysql-client-5.1` のように明示的にバージョンを指定するとよいでしょう。

依存関係を含めてダウンロードしてインストールするには、インストールするパッケージを指定して `apt-get` コマンドを使用します。

注記

入手可能な最新バージョンを確実にダウンロードするために、インストールの前に `apt-get` インデックスファイルを更新してください。

注記

`apt-get` コマンドは、一般的なツールおよびアプリケーション環境を提供するために、MySQL サーバーを含む多数のパッケージをインストールします。これは、メインの MySQL パッケージに加えて多数のパッケージをインストールするということを意味しません。

インストール中に初期データベースが作成され、MySQL ルートパスワード (および確認) の入力を求められます。構成ファイルが `/etc/mysql/my.cnf` に作成されます。init スクリプトが `/etc/init.d/mysql` に作成されます。

サーバーはすでに起動しているはずです。次を使用してサーバーを自動的に起動および停止できます。

```
root-shell> service mysql [start|stop]
```

サービスは 2、3 および 4 の実行レベルに自動的に追加され、停止スクリプトは単一、停止および再起動レベルで実行されます。

2.5.8 Juju を使用した Linux への MySQL のインストール

Juju デプロイメントフレームワークは、MySQL サーバーの簡単なインストールおよび構成をサポートします。その手順は、<https://jujucharms.com/mysql/> を参照してください。

2.5.9 systemd を使用した MySQL Server の管理

RPM または Debian パッケージを使用して MySQL を次の Linux プラットフォームにインストールする場合、サーバーの起動と停止は systemd によって管理されます:

- RPM パッケージプラットフォーム:
 - Enterprise Linux バリエーションのバージョン 7 以上
 - SUSE Linux Enterprise Server 12 以上
 - Fedora 29 以上
- Debian ファミリープラットフォーム:
 - Debian プラットフォーム
 - Ubuntu プラットフォーム

systemd を使用するプラットフォームに汎用バイナリディストリビューションから MySQL をインストールする場合は、「MySQL 8.0 セキュアデプロイメントガイド」のインストール後の設定に関するセクションに記載されている手順に従って、MySQL の systemd サポートを手動で構成できます。

systemd を使用するプラットフォーム上のソース配布から MySQL をインストールする場合は、`-DWITH_SYSTEMD=1` CMake オプションを使用して配布を構成することで、MySQL の systemd サポートを取得します。セクション 2.9.7 「MySQL ソース構成オプション」を参照してください。

次のセクションでは、これらのトピックについて説明します:

- [systemd の概要](#)
- [MySQL の systemd の構成](#)
- [systemd を使用した複数の MySQL インスタンスの構成](#)
- [mysqld_safe から systemd への移行](#)

注記

systemd support for MySQL がインストールされているプラットフォームでは、`mysqld_safe` や System V 初期化スクリプトなどのスクリプトは不要であり、インストールされません。たとえば、`mysqld_safe` はサーバーの再起動を処理できますが、systemd は同じ機能を提供し、アプリケーション固有のプログラムを使用するのではなく、他のサービスの管理と一貫した方法で処理します。

systemd には、systemd support for MySQL がインストールされているプラットフォームで複数の MySQL インスタンスを管理する機能があるため、`mysqld_multi` および `mysqld_multi.server` は不要であり、インストールされません。

systemd の概要

systemd は、MySQL サーバーの自動起動および停止を提供します。また、`systemctl` コマンドを使用した手動サーバー管理も有効になります。例:

```
shell> systemctl {start|stop|restart|status} mysqld
```

または、System V システムと互換性のある `service` コマンドを (引数を逆にして) 使用します:

```
shell> service mysqld {start|stop|restart|status}
```

注記

`systemctl` コマンド (および代替の `service` コマンド) では、MySQL サービス名が `mysqld` でない場合、適切な名前を使用します。たとえば、Debian ベースおよび SLES システムでは、`mysqld` ではなく `mysql` を使用します。

systemd のサポートには、次のファイルが含まれます:

- `mysqld.service` (RPM プラットフォーム)、`mysql.service` (Debian プラットフォーム): systemd サービスユニット構成ファイル (MySQL サービスの詳細を含む)。
- `mysqld@.service` (RPM プラットフォーム)、`mysql@.service` (Debian プラットフォーム): `mysqld.service` または `mysql.service` と似ていますが、複数の MySQL インスタンスの管理に使用されます。
- `mysqld.tmpfiles.d: tmpfiles` 機能をサポートするための情報を含むファイル。このファイルは、`mysql.conf` という名前でインストールされます。
- `mysqld_pre_systemd` (RPM プラットフォーム)、`mysql-system-start` (Debian プラットフォーム): ユニットファイルのサポートスクリプト。このスクリプトは、ログの場所がパターン (RPM プラットフォームの場合は `/var/log/mysql*.log`、Debian プラットフォームの場合は `/var/log/mysql/*.log`) と一致する場合にのみ、エラーログファイルの作成に役立ちます。それ以外の場合は、`mysqld` プロセスを実行しているユーザーのエラーログディレクトリが書き込み可能であるか、エラーログが存在し、書き込み可能である必要があります。

MySQL の systemd の構成

MySQL の systemd オプションを追加または変更するには、次の方法を使用できます:

- ローカライズされた systemd 構成ファイルを使用します。
- systemd を配置して、MySQL サーバードキュメントの環境変数を設定します。
- `MYSQLD_OPTS` systemd 変数を設定します。

ローカライズされた systemd 構成ファイルを使用するには、`/etc/systemd/system/mysqld.service.d` ディレクトリを作成します (存在しない場合)。そのディレクトリに、目的の設定をリストする `[Service]` セクションを含むファイルを作成します。例:

```
[Service]
LimitNOFILE=max_open_files
Nice=nice_level
LimitCore=core_file_limit
Environment="LD_PRELOAD=/path/to/malloc/library"
Environment="TZ=time_zone_setting"
```

ここでは、このファイルの名前として `override.conf` を使用します。systemd の新しいバージョンでは、次のコマンドがサポートされており、エディタが開き、ファイルを編集できます:

```
systemctl edit mysqld # RPM platforms
systemctl edit mysql # Debian platforms
```

`override.conf` を作成または変更するたびに、systemd 構成をリロードしてから、MySQL サービスを再起動するように systemd に指示します:

```
systemctl daemon-reload
systemctl restart mysqld # RPM platforms
```

```
systemctl restart mysql # Debian platforms
```

systemd では、MySQL オプションファイルの[mysqld]、[mysqld_safe]または[safe_mysqld]グループの設定ではなく、特定のパラメータに `override.conf` 構成方法を使用する必要があります:

- 一部のパラメータでは、systemd 自体がその値を知っている必要があります、MySQL オプションファイルを読み取って取得できないため、`override.conf` を使用する必要があります。
- それ以外の場合は、systemd を使用して `mysqld_safe` が認識するオプションのみを使用して設定可能な値を指定するパラメータを指定する必要があります。これは、対応する `mysqld` パラメータがないためです。

`mysqld_safe` ではなく systemd を使用方法の詳細は、[mysqld_safe から systemd への移行](#) を参照してください。

`override.conf` では次のパラメータを設定できます:

- MySQL サーバーで使用可能なファイル記述子の数を設定するには、`mysqld` の `open_files_limit` システム変数または `mysqld_safe` の `--open-files-limit` オプションではなく、`override.conf` で `LimitNOFILE` を使用します。
- コアファイルの最大サイズを設定するには、`mysqld_safe` の `--core-file-size` オプションではなく、`override.conf` で `LimitCore` を使用します。
- MySQL サーバーのスケジューリング優先度を設定するには、`mysqld_safe` の `--nice` オプションではなく、`override.conf` の `Nice` を使用します。

一部の MySQL パラメータは、環境変数を使用して構成されます:

- `LD_PRELOAD`: MySQL サーバーが特定のメモリー割り当てライブラリを使用する必要がある場合は、この変数を設定します。
- `NOTIFY_SOCKET`: この環境変数は、`mysqld` が起動完了およびサービスステータス変更の通知を systemd と通信するために使用するソケットを指定します。これは、`mysqld` サービスの起動時に systemd によって設定されます。`mysqld` サービスは、変数設定を読み取り、定義された場所に書き込みます。

MySQL 8.0 では、`mysqld` は `Type=notify` プロセス起動タイプを使用します。(`Type=forking` は MySQL 5.7 で使用されていました。) `Type=notify` では、systemd によってソケットファイルが自動的に構成され、パスが `NOTIFY_SOCKET` 環境変数にエクスポートされます。

- `TZ`: この変数を設定して、サーバーのデフォルトのタイムゾーンを指定します。

systemd によって管理される MySQL サーバードプロセスで使用する環境変数値を指定するには、複数の方法があります:

- `override.conf` ファイルで `Environment` 行を使用します。構文については、このファイルの使用方法を説明している前述の例を参照してください。
- `/etc/sysconfig/mysql` ファイルに値を指定します (ファイルが存在しない場合は作成します)。次の構文を使用して値を割り当てます:

```
LD_PRELOAD=/path/to/malloc/library
TZ=time_zone_setting
```

`/etc/sysconfig/mysql` を変更した後、サーバーを再起動して変更を有効にします:

```
systemctl restart mysqld # RPM platforms
systemctl restart mysql # Debian platforms
```

systemd 構成ファイルを直接変更せずに `mysqld` のオプションを指定するには、`MYSQLD_OPTS` systemd 変数を設定または設定解除します。例:

```
systemctl set-environment MYSQLD_OPTS="--general_log=1"
systemctl unset-environment MYSQLD_OPTS
```

`MYSQLD_OPTS` は、`/etc/sysconfig/mysql` ファイルでも設定できます。

systemd 環境を変更した後、サーバーを再起動して変更を有効にします:

```
systemctl restart mysqld # RPM platforms
systemctl restart mysql # Debian platforms
```

systemd を使用するプラットフォームでは、サーバーの起動時に空の場合、データディレクトリが初期化されます。これは、データディレクトリが一時的に消えたりリモートマウントである場合に問題になる可能性があります: マウントポイントは空のデータディレクトリのように見え、新しいデータディレクトリとして初期化されます。この自動初期化動作を抑制するには、`/etc/sysconfig/mysqld` ファイルで次の行を指定します (ファイルが存在しない場合は作成します):

```
NO_INIT=true
```

systemd を使用した複数の MySQL インスタンスの構成

このセクションでは、MySQL の複数のインスタンスに対して systemd を構成する方法について説明します。

注記

systemd には systemd サポートがインストールされているプラットフォームで複数の MySQL インスタンスを管理する機能があるため、`mysqld_multi` および `mysqld_multi.server` は不要であり、インストールされません。

複数インスタンス機能を使用するには、`my.cnf` オプションファイルを変更して、各インスタンスのキーオプションの構成を含めます。これらのファイルの場所は一般的です:

- `/etc/my.cnf` または `/etc/mysql/my.cnf` (RPM プラットフォーム)
- `/etc/mysql/mysql.conf.d/mysqld.cnf` (Debian プラットフォーム)

たとえば、`replica01` および `replica02` という名前の 2 つのインスタンスを管理するには、次のようなものをオプションファイルに追加します:

RPM プラットフォーム:

```
[mysqld@replica01]
datadir=/var/lib/mysql-replica01
socket=/var/lib/mysql-replica01/mysql.sock
port=3307
log-error=/var/log/mysqld-replica01.log

[mysqld@replica02]
datadir=/var/lib/mysql-replica02
socket=/var/lib/mysql-replica02/mysql.sock
port=3308
log-error=/var/log/mysqld-replica02.log
```

Debian プラットフォーム:

```
[mysqld@replica01]
datadir=/var/lib/mysql-replica01
socket=/var/lib/mysql-replica01/mysql.sock
port=3307
log-error=/var/log/mysql/replica01.log

[mysqld@replica02]
datadir=/var/lib/mysql-replica02
socket=/var/lib/mysql-replica02/mysql.sock
port=3308
log-error=/var/log/mysql/replica02.log
```

systemd でサポートされているデリミタは `@` のみであるため、ここに示されているレプリカ名はデリミタとして `@` を使用します。

その後、インスタンスは次のような通常の systemd コマンドによって管理されます:

```
systemctl start mysqld@replica01
systemctl start mysqld@replica02
```

ブート時にインスタンスを実行できるようにするには、次の手順を実行します:


```
systemctl enable mysqld@replica01
systemctl enable mysqld@replica02
```

ワイルドカードの使用もサポートされます。たとえば、次のコマンドはすべてのレプリカインスタンスのステータスを表示します:

```
systemctl status 'mysqld@replica*'
```

同じマシン上の複数の MySQL インスタンスを管理するために、systemd は自動的に異なる単位ファイルを使用します:

- `mysqld.service` (RPM プラットフォーム) ではなく `mysqld@.service`
- `mysql.service` (Debian プラットフォーム) ではなく `mysql@.service`

ユニットファイルでは、`%i` および `%i` は `@` マーカーの後に渡されたパラメータを参照し、特定のインスタンスの管理に使用されます。次のようなコマンドの場合:

```
systemctl start mysqld@replica01
```

systemd は、次のようなコマンドを使用してサーバーを起動します:

```
mysqld --defaults-group-suffix=@%i ...
```

その結果、`[server]`、`[mysqld]` および `[mysqld@replica01]` オプショングループが読み取られ、サービスのそのインスタンスに使用されます。

注記

Debian プラットフォームでは、AppArmor は、サーバーが `/var/lib/mysql-replica*` またはデフォルト以外の場所を読み書きできないようにします。これに対処するには、`/etc/apparmor.d/usr.sbin.mysqld` でプロファイルをカスタマイズまたは無効化する必要があります。

注記

Debian プラットフォームでは、MySQL のアンインストール用のパッケージ化スクリプトは、現在 `mysqld@` インスタンスを処理できません。パッケージを削除またはアップグレードする前に、追加のインスタンスを手動で停止する必要があります。

mysqld_safe から systemd への移行

systemd を使用して MySQL を管理するプラットフォームには `mysqld_safe` がインストールされていないため、そのプログラムに対して (`[mysqld_safe]` または `[safe_mysqld]` オプショングループなどで) 以前に指定したオプションは、別の方法で指定する必要があります:

- 一部の `mysqld_safe` オプションは `mysqld` でも認識され、`[mysqld_safe]` または `[safe_mysqld]` オプショングループから `[mysqld]` グループに移動できます。これには、`--pid-file`、`--open-files-limit` または `--nice` は含まれません。これらのオプションを指定するには、前述の `override.conf` systemd ファイルを使用します。

注記

systemd プラットフォームでは、`[mysqld_safe]` および `[safe_mysqld]` オプショングループの使用はサポートされておらず、予期しない動作を引き起こす可能性があります。

- 一部の `mysqld_safe` オプションには、代替の `mysqld` プロシージャがあります。たとえば、`syslog` ログインを有効にするための `mysqld_safe` オプションは `--syslog` で、これは非推奨です。エラーログ出力をシステムログに書き込むには、[セクション 5.4.2.8 「システムログへのエラーログイン」](#) の手順を使用します。
- `mysqld` で認識されない `mysqld_safe` オプションは、`override.conf` または環境変数で指定できます。たとえば、`mysqld_safe` では、サーバーが特定のメモリー割当てライブラリを使用する必要がある場合、これは `--malloc-lib` オプションを使用して指定します。systemd を使用してサーバーを管理するインストールの場合は、前述のように、かわりに `LD_PRELOAD` 環境変数を設定します。

2.6 Unbreakable Linux Network (ULN) を使用した MySQL のインストール

セクション2.5「Linux に MySQL をインストールする」で説明するように、Linux は、MySQL をインストールするための何種類かのソリューションをサポートします。このセクションで説明する方法の1つは、Oracle Unbreakable Linux Network (ULN) からのインストールです。Oracle Linux および ULN の詳細は、<http://linux.oracle.com/>にあります。

ULN を使用するには、ULN ログインを取得し、ULN でのインストールに使用するマシンを登録する必要があります。これは、[ULN FAQ](#) で詳細に説明しています。このページでは、パッケージをインストールおよび更新する方法についても説明します。

Community パッケージと Commercial パッケージの両方がサポートされています：

- コミュニティバージョンには、MySQL 8.0 など、MySQL Server のバージョンごとに1つのチャンネルがあります。これらは、すべての ULN ユーザーが使用できます。
- 商用バージョンでは、3つのチャンネルを利用できます：「MySQL 8.0 Commercial Server」、「MySQL 8.0 Connectors Commercial」および「MySQL 8.0 Tools Commercial」。

oracle linux.com で商用 MySQL ULN パッケージにアクセスするには、CSI に MySQL (Enterprise または Standard) の有効な商用ライセンスを提供する必要があります。この文書では、有効な購入は 60944、60945、64911 および 64912 です。適切な CSI により、ULN GUI インタフェースで商用 MySQL サブスクリプションチャンネルを使用できるようになります。

注記

Oracle Linux 8 は、MySQL 8.0.17 でサポートされています。

ULN を使用して MySQL をインストールしたあと、[このセクション](#)、特に[セクション2.5.4「Oracle の RPM パッケージを使用した Linux への MySQL のインストール」](#)に、サーバーの起動および停止に関する情報があります。

ULN を使用するようにパッケージソースを変更し、使用している MySQL のビルドを変更しない場合は、データをバックアップし、既存のバイナリを削除して ULN のものに置き換えます。ビルドの変更が必要な場合は、新しいバイナリの配置後にデータを再構築する必要がある場合に備えて、バックアップをダンプ ([mysqldump](#) または [mysqlpump](#)、あるいは [MySQL Shell backup utility](#) から) にすることをお勧めします。ULN へのこのシフトがバージョン境界を超えている場合は、先に進む前にこのセクションを参照してください：[セクション2.11「MySQL のアップグレード」](#)。

2.7 Solaris への MySQL のインストール

注記

MySQL 8.0 では、Solaris 11.4 以上がサポートされます

Solaris 上の MySQL は、様々な形式で使用できます。

- ネイティブの Solaris PKG 形式を使用するインストールの詳細は、[セクション2.7.1「Solaris PKG を使用して Solaris に MySQL をインストールする」](#)を参照してください。
- 標準の tar バイナリインストールを使用するには、[セクション2.2「一般的なバイナリを使用した MySQL の Unix/Linux へのインストール」](#)の注記を参照してください。インストール前後に必要な可能性のある、Solaris 固有の注記については、このセクションの最後の注記およびヒントをチェックしてください。

重要

インストールパッケージは Oracle Developer Studio 12.6 ランタイムライブラリに依存しており、MySQL インストールパッケージを実行する前にインストールする必要があります。Oracle Developer Studio [「ここ」](#) のダウンロードオプションを参照してください。インストールパッケージでは、完全な Oracle Developer Studio ではなくランタイムライブラリのみをインストールできます。[「Oracle Solaris 11 へのランタイムライブラリのみをインストール」](#)の手順を参照してください。

tarball または PKG 形式の Solaris 用のバイナリ MySQL 配布を取得するには、<https://dev.mysql.com/downloads/mysql/8.0.html> にアクセスしてください。

MySQL を Solaris にインストールして使用する上で知っておくべき追加の注記:

- MySQL を `mysql` ユーザーおよびグループで使用する場合は、`groupadd` コマンドおよび `useradd` コマンドを使用します。

```
groupadd mysql
useradd -g mysql -s /bin/false mysql
```

- Solaris `tar` は長いファイル名を処理できないため、バイナリ tarball 配布を使用して MySQL を Solaris にインストールする場合は、GNU `tar` (`gtar`) を使用して配布を解凍します。システムに GNU `tar` がない場合は、次のコマンドを使用してインストールします:

```
pkg install archiver/gnu-tar
```

- `forcedirectio` オプションを使用して、`InnoDB` ファイルを格納するすべてのファイルシステムをマウントする必要があります。(デフォルトではこのオプションなしでマウントされます。) そうしないと、このプラットフォームで `InnoDB` ストレージエンジンを使用するときにパフォーマンスが大幅に低下します。
- MySQL を自動的に起動する場合は、`support-files/mysql.server` を `/etc/init.d` にコピーして、それに `/etc/rc3.d/S99mysql.server` という名前のシンボリックリンクを作成します。
- あまりにも多くのプロセスが急激に `mysqld` に接続を試みた場合、MySQL ログに次のエラーが記録されます。

```
Error in accept: Protocol error
```

この問題の回避策としてサーバーを `--back_log=50` オプションで起動するとよいでしょう。

- Solaris でコアファイルの生成を構成するには、`coreadm` コマンドを使用します。`setuid()` アプリケーションでコアを生成するとセキュリティ上の問題があるため、デフォルトでは Solaris は `setuid()` プログラムではコアファイルをサポートしません。ただし、この動作は `coreadm` を使用して変更できます。現在のユーザーに対して `setuid()` コアファイルを有効にすると、それらはモード 600 を使用して生成され、スーパーユーザーによって所有されます。

2.7.1 Solaris PKG を使用して Solaris に MySQL をインストールする

バイナリ tarball 配布ではなく、ネイティブ「Solaris PKG」形式のバイナリパッケージを使用して、Solaris に MySQL をインストールできます。

重要

インストールパッケージは Oracle Developer Studio 12.6 ランタイムライブラリに依存しており、MySQL インストールパッケージを実行する前にインストールする必要があります。Oracle Developer Studio 「[ここ](#)」のダウンロードオプションを参照してください。インストールパッケージでは、完全な Oracle Developer Studio ではなくランタイムライブラリのみをインストールできます。「[Oracle Solaris 11 へのランタイムライブラリのみをインストール](#)」の手順を参照してください。

このパッケージを使用するには、対応する `mysql-VERSION-solaris11-PLATFORM.pkg.gz` ファイルをダウンロードして解凍します。例:

```
shell> gunzip mysql-8.0.29-solaris11-x86_64.pkg.gz
```

新しいパッケージをインストールするには、`pkgadd` を使用して画面の指示に従います。この操作を実行するにはルート権限が必要です。

```
shell> pkgadd -d mysql-8.0.29-solaris11-x86_64.pkg
```

```
The following packages are available:
 1 mysql   MySQL Community Server (GPL)
           (i86pc) 8.0.29
```

```
Select package(s) you wish to process (or 'all' to process
all packages). (default: all) [?,??,q]:
```

PKG インストーラは、必要なすべてのファイルおよびツールをインストールし、そのあとデータベースが存在しない場合はデータベースを初期化します。インストールを完了するには、インストールの最後の指示に示されるように、MySQL のルートパスワードを設定してください。あるいは、インストールに同梱されている [mysql_secure_installation](#) スクリプトを実行することもできます。

デフォルトでは、PKG パッケージは MySQL をルートパス `/opt/mysql` にインストールします。インストールのルートパスを変更できるのは `pkgadd` を使用している場合のみで、これを使用すると MySQL を異なる Solaris ゾーンにインストールできます。特定のディレクトリにインストールする必要がある場合は、バイナリの `tar` ファイルの配布を使用してください。

`pkg` インストーラは、適切な MySQL スタートアップスクリプトを `/etc/init.d/mysql` にコピーします。MySQL を自動的にスタートアップおよびシャットダウンできるようにするためには、このファイルと `init` スクリプトディレクトリとの間にリンクを作成してください。たとえば、MySQL の安全なスタートアップおよびシャットダウンを確実にするには、次のコマンドを使用して適切なリンクを追加します。

```
shell> ln /etc/init.d/mysql /etc/rc3.d/S91mysql
shell> ln /etc/init.d/mysql /etc/rc0.d/K02mysql
```

MySQL を削除する場合、インストールされるパッケージ名は `mysql` です。これを `pkgrm` コマンドと合わせて使用してインストールを削除できます。

Solaris パッケージファイル形式を使用している場合にアップグレードするには、更新パッケージをインストールする前に既存のインストールを削除する必要があります。パッケージを削除しても既存のデータベース情報は削除されません。サーバー、バイナリ、およびサポートファイルのみが削除されます。したがって、通常のアップグレード手順は次のようになります。

```
shell> mysqladmin shutdown
shell> pkgrm mysql
shell> pkgadd -d mysql-8.0.29-solaris11-x86_64.pkg
shell> mysqld_safe &
shell> mysql_upgrade # prior to MySQL 8.0.16 only
```

アップグレードを実行する前に、[セクション2.11「MySQL のアップグレード」](#)の注記を確認してください。

2.8 FreeBSD に MySQL をインストールする

このセクションでは、FreeBSD Unix のバリエーションへの MySQL のインストールに関する情報を提供します。

オラクルが提供するバイナリ配布を使用して、MySQL を FreeBSD にインストールできます。詳細については、[セクション2.2「一般的なバイナリを使用した MySQL の Unix/Linux へのインストール」](#)を参照してください。

MySQL のインストールでもっとも容易な(かつ推奨される)方法は `mysql-server` ポートおよび `mysql-client` ポートを使用することです。それらは <http://www.freebsd.org/> で入手できます。これらのポートを使用することで次のメリットがあります。

- 使用している FreeBSD バージョンで動作することが知られている、すべての最適化を行なった実働可能な MySQL。
- 自動構成およびビルド。
- `/usr/local/etc/rc.d` にインストールされたスタートアップスクリプト。
- どのファイルがインストールされているかを確認するための `pkg_info -L` の使用。
- 使用しているマシンで MySQL が不要でなくなった場合にそれを削除できる `pkg_delete` の使用。

MySQL ビルドプロセスが機能するには GNU make (`gmake`) が必要です。GNU make が利用できない場合、MySQL をコンパイルする前にインストールする必要があります。

注記

`ldd mysqlld` に従った前提条件ライブラリ:
`libthr`、`libcrypt`、`libkrb5`、`libm`、`librt`、`libexecinfo`、`libunwind`、および `libssl`。

ポートシステムを使用してインストールするには:

```
# cd /usr/ports/databases/mysql80-server
# make
...
# cd /usr/ports/databases/mysql80-client
# make
...
```

標準ポートインストールでは、サーバーは `/usr/local/libexec/mysql` に置かれ、MySQL Server のスタートアップスクリプトは `/usr/local/etc/rc.d/mysql-server` に置かれます。

BSD 実装に関する追加の注記:

- ポートシステムを使用して MySQL をインストールしたあとに削除するには:

```
# cd /usr/ports/databases/mysql80-server
# make deinstall
...
# cd /usr/ports/databases/mysql80-client
# make deinstall
...
```

- MySQL の現在の日付に問題がある場合には、`TZ` 変数の設定が役に立ちます。 [セクション4.9「環境変数」](#) を参照してください。

2.9 ソースから MySQL をインストールする

MySQL をソースコードからビルドすると、ビルドパラメータ、コンパイラ最適化、およびインストールの場所をカスタマイズできます。MySQL を実行できることがわかっているシステムのリストは、<https://www.mysql.com/support/supportedplatforms/database.html> を参照してください。

ソースからのインストールに進む前に、使用しているプラットフォーム用に事前コンパイルされたバイナリ配布を Oracle が作成しているかどうか、およびそれがニーズに適合しているかどうかをチェックしてください。弊社は、パフォーマンスを最適化するために、最善のオプションでバイナリをビルドすることを保証するべく多大な努力をしています。バイナリ配布のインストール手順は、[セクション2.2「一般的なバイナリを使用した MySQL の Unix/Linux へのインストール」](#) を参照してください。

ビルドオプションを使用して、プラットフォーム上でバイナリ配布を生成するために Oracle で使用するビルドオプションと同じか類似したビルドオプションを使用してソース配布から MySQL を構築する場合は、バイナリ配布を取得して解凍し、`docs/INFO_BIN` ファイルを参照します。このファイルには、MySQL 配布の構成およびコンパイル方法に関する情報が含まれています。

警告

非標準オプションを使用して MySQL を構築すると、機能、パフォーマンスまたはセキュリティが低下する可能性があります。

MySQL ソースコードには、Doxygen を使用して記述された内部ドキュメントが含まれます。生成された Doxygen コンテンツは、<https://dev.mysql.com/doc/index-other.html> で入手できます。 [セクション2.9.10「MySQL Doxygen ドキュメントコンテンツの生成」](#) の手順を使用して、このコンテンツを MySQL ソース配布からローカルに生成することもできます。

2.9.1 ソースのインストール方法

MySQL をソースからインストールするには、2つの方法があります。

- 標準の MySQL ソース配布を使用します。標準の配布を取得するには、[セクション2.1.3「MySQL の取得方法」](#) を参照してください。標準の配布からのビルドの詳細は、[セクション2.9.4「標準ソース配布を使用して MySQL をインストールする」](#) を参照してください。

標準の配布は、圧縮 tar ファイル、Zip アーカイブ、または RPM パッケージとして入手可能です。配布ファイルは、`mysql-VERSION.tar.gz`、`mysql-VERSION.zip`、または `mysql-VERSION.rpm` という形式の名前を持ちます。ここで、`VERSION` は `8.0.29` などの番号です。ソース配布のファイル名は汎用でプラットフォーム名を含まないのに対し、バイナリ配布名にはその配布が対象とするシステムのタイプを示すプラットフォーム名が含まれる (たとえば、`pc-linux-i686` または `winx64`) ことから、ソースファイルを事前コンパイル済みのバイナリ配布と区別できます。

- MySQL 開発ツリーを使用します。開発ツリーからのビルドの詳細は、[セクション2.9.5「開発ソースツリーを使用して MySQL をインストールする」](#)を参照してください。

2.9.2 ソースインストールの前提条件

MySQL をソースからインストールするためには、いくつかの開発ツールが必要です。これらのツールの一部は、標準ソース配布または開発ソースツリーのいずれを使用するかにかかわらず必要です。その他のツールは、どちらのインストール方法を使用するかによって必要かどうかが決まります。

ソースから MySQL をインストールするには、インストール方法に関係なく、次のシステム要件を満たす必要があります:

- **CMake**。これはすべてのプラットフォームでビルドフレームワークとして使用されます。CMake は <http://www.cmake.org> からダウンロードできます。
- 優良な **make** プログラム。一部のプラットフォームには独自の **make** 実装が付属していますが、GNU **make** 3.75 以上を使用することを強くお勧めします。これは、使用しているシステムですでに **gmake** として使用可能になっている場合があります。GNU **make** は、<http://www.gnu.org/software/make/> から入手可能です。
- MySQL 8.0 ソースコードでは、C++14 機能を使用できます。サポートされているすべてのプラットフォームで適切なレベルの C++14 サポートを有効にするには、次の最小コンパイラバージョンが適用されます:
 - GCC 5.3 (Linux)
 - Clang 4.0 (FreeBSD)
 - XCode 9 (macOS)
 - Developer Studio 12.6 (Solaris)
 - Visual Studio 2017 (Windows)
- MySQL C API をコンパイルするには、C++ または C99 コンパイラが必要です。
- SSL ライブラリは、暗号化された接続、乱数生成のエントロピ、およびその他の暗号化関連の操作をサポートするために必要です。デフォルトでは、ビルドはホストシステムにインストールされている OpenSSL ライブラリを使用します。ライブラリを明示的に指定するには、CMake の起動時に **WITH_SSL** オプションを使用します。追加情報については [セクション2.9.6「SSL ライブラリサポートの構成」](#)を参照してください。
- Boost C++ ライブラリは、MySQL をビルドするために必要です (ただし、使用するためには必要ありません)。MySQL コンパイルには、特定の Boost バージョンが必要です。通常、これは現在の Boost バージョンですが、特定の MySQL ソース配布に異なるバージョンが必要な場合、構成プロセスは停止し、必要な Boost バージョンを示すメッセージが表示されます。Boost とそのインストール手順については、「[公式サイト](#)」を参照してください。Boost のインストール後、CMake の起動時に **WITH_BOOST** オプションを定義して、Boost ファイルが配置されている場所をビルドシステムに通知します。例:

```
cmake . -DWITH_BOOST=/usr/local/boost_version_number
```

必要に応じて、インストールに合わせてパスを調整します。

- **ncurses** ライブラリ。
- 十分な空きメモリー。大きなソースファイルのコンパイル時に「内部コンパイラエラー」などの問題が発生した場合は、メモリーが少なすぎる可能性があります。仮想マシンでコンパイルする場合は、メモリー割当てを増やしてみてください。
- テストスクリプトを実行する場合は、Perl が必要です。ほとんどの Unix 類似システムには Perl が含まれます。Windows では、ActiveState Perl などのバージョンが使用できます。

MySQL を標準ソース配布からインストールするには、配布ファイルをアンパックするために次のツールのいずれかが必要です。

- **.tar.gz** で圧縮された **tar** ファイルの場合: 配布を圧縮解除するための GNU **gunzip**、およびそれをアンパックするための妥当な **tar**。使用している **tar** プログラムが **z** オプションをサポートする場合は、ファイルの展開とアンパックの両方を実行できます。

GNU tar が機能することが知られています。一部のオペレーティングシステムで提供される標準の tar は、MySQL 配布内の長いファイル名をアンパックできません。GNU tar をダウンロードしてインストールするか、プリインストールバージョンの GNU tar が利用可能であればそれを使用します。通常、これは gnutar、gtar、または tar という名前です (/usr/sfw/bin または /usr/local/bin などの GNU または Free Software ディレクトリ内)。GNU tar は、<http://www.gnu.org/software/tar/> から入手可能です。

- .zip Zip アーカイブの場合: WinZip または .zip ファイルを読み取ることができるその他のツール。
- .rpm RPM パッケージの場合: 配布のビルドに使用される rpmbuild プログラムでアンパックできます。

MySQL を開発ソースツリーからインストールするには、次の追加ツールが必要です。

- 開発ソースコードを取得するには、Git リビジョン管理システムが必要です。「[GitHub ヘルプ](#)」では、様々なプラットフォームで Git をダウンロードしてインストールする手順を示します。MySQL は、2014 年 9 月に正式に GitHub に参加しました。MySQL の GitHub への移動の詳細は、MySQL Release Engineering ブログ [MySQL on GitHub](#) のお知らせを参照してください。
- <http://www.gnu.org/software/bison/> から入手可能な bison 2.1 以上。(バージョン 1 はサポートされなくなりました。)可能であれば bison の最新バージョンを使用します。問題が生じた場合は、以前のバージョンに戻るのではなく、より新しいバージョンにアップグレードします。

bison は、<http://www.gnu.org/software/bison/> から入手可能です。bison for Windows は、<http://gnuwin32.sourceforge.net/packages/bison.htm> からダウンロードできます。「Complete package, excluding sources」のラベルのパッケージをダウンロードします。Windows では、bison のデフォルトの場所は C:\Program Files\GnuWin32 ディレクトリです。ディレクトリ名にスペースが含まれるため、一部のユーティリティーでは bison を検索できない場合があります。また、パスにスペースがある場合 Visual Studio がフリーズすることがあります。これらの問題は、空白を含まないディレクトリ (C:\GnuWin32 など) にインストールすることで解決できます。

- Solaris Express では、bison に加えて m4 をインストールする必要があります。m4 は <http://www.gnu.org/software/m4/> から入手できます。

注記

プログラムをインストールする必要がある場合は、PATH 環境変数を、プログラムが置かれるディレクトリを含むように変更します。[セクション4.2.9「環境変数の設定」](#)を参照してください。

問題が発生してバグをレポートする必要がある場合には、[セクション1.6「質問またはバグをレポートする方法」](#)の手順に従ってください。

2.9.3 ソースインストールの MySQL のレイアウト

デフォルトでは、MySQL をソースからコンパイルしたあとにインストールすると、インストール手順によりファイルが /usr/local/mysql にインストールされます。インストールディレクトリ下のコンポーネントの場所は、バイナリ配布の場合と同じです。[表2.3「一般的な Unix/Linux バイナリパッケージの MySQL インストールのレイアウト」](#)および[セクション2.3.1「Microsoft Windows 上での MySQL のインストールレイアウト」](#)を参照してください。デフォルトとは異なるインストールの場所を構成するには、[セクション2.9.7「MySQL ソース構成オプション」](#)で説明されるオプションを使用します。

2.9.4 標準ソース配布を使用して MySQL をインストールする

標準ソース配布を使用して MySQL をインストールするには:

1. システムが、[セクション2.9.2「ソースインストールの前提条件」](#)にリストされるツール要件を満たすことを確認します。
2. 配布ファイルを [セクション2.1.3「MySQL の取得方法」](#)の説明に従って取得します。
3. このセクションの説明に従って、配布の構成、ビルド、およびインストールを実行します。
4. [セクション2.10「インストール後のセットアップとテスト」](#)の説明に従って、インストール後の手順を実行します。

MySQL は、すべてのプラットフォームでビルドフレームワークとして CMake を使用します。ここに記載する説明で、動作するインストールを作成できるでしょう。CMake を使用して MySQL をビルドする方法の詳細は、CMake による MySQL Server のビルド方法を参照してください。

ソース RPM から開始する場合は、インストールするバイナリ RPM を、次のコマンドを使用して作成します。rpmbuild がいない場合は代わりに rpm を使用します。

```
shell> rpmbuild --rebuild --clean MySQL-VERSION.src.rpm
```

結果として 1 つまたは複数の RPM パッケージが生成されます。セクション 2.5.4 「Oracle の RPM パッケージを使用した Linux への MySQL のインストール」の指示に従ってインストールします。

圧縮 tar ファイルまたは Zip アーカイブソース配布からのインストールのシーケンスは、ソース配布はすべてのプラットフォームで使用されること、および配布を構成してコンパイルする手順が含まれる点を除き、一般的なバイナリ配布のプロセスと同様です (セクション 2.2 「一般的なバイナリを使用した MySQL の Unix/Linux へのインストール」を参照してください)。たとえば、Unix の圧縮 tar ファイルのソース配布では、基本的なインストールコマンドシーケンスは次のようになります。

```
# Preconfiguration setup
shell> groupadd mysql
shell> useradd -r -g mysql -s /bin/false mysql
# Beginning of source-build specific instructions
shell> tar zxvf mysql-VERSION.tar.gz
shell> cd mysql-VERSION
shell> mkdir bld
shell> cd bld
shell> cmake ..
shell> make
shell> make install
# End of source-build specific instructions
# Postinstallation setup
shell> cd /usr/local/mysql
shell> mkdir mysql-files
shell> chown mysql:mysql mysql-files
shell> chmod 750 mysql-files
shell> bin/mysqld --initialize --user=mysql
shell> bin/mysqld_ssl_rsa_setup
shell> bin/mysqld_safe --user=mysql &
# Next command is optional
shell> cp support-files/mysql.server /etc/init.d/mysql.server
```

このあと、ソースビルド固有の手順について、より詳細に説明します。

注記

ここに示した手順では、MySQL アカウントにパスワードは設定しません。その手順のあとは、セクション 2.10 「インストール後のセットアップとテスト」に進み、インストール後のセットアップとテストを実行します。

- 構成前のセットアップの実行
- 配布の取得とアンパック
- 配布の構成
- 配布のビルド
- 配付のインストール
- インストール後のセットアップの実行

構成前のセットアップの実行

Unix では、MySQL サーバーの実行および実行に使用する mysql ユーザーおよびグループを設定し、データベースディレクトリを所有します。詳細は、mysql ユーザーおよびグループの作成を参照してください。次に、指定された場合を除き、mysql ユーザーとして次の手順を実行します。

配布の取得とアンパック

配布をアンパックするディレクトリを選択してそこに移動します。

配布ファイルを [セクション2.1.3「MySQL の取得方法」](#) の説明に従って取得します。

配布を現在のディレクトリにアンパックします。

- 圧縮 `tar` ファイルをアンパックするには、`tar` が `z` オプションをサポートする場合、このコマンドで配布の圧縮解除とアンパックを実行できます。

```
shell> tar zxvf mysql-VERSION.tar.gz
```

`tar` が `z` オプションをサポートしない場合は、`gunzip` を使用して配布をパックし、`tar` を使用してアンパックします。

```
shell> gunzip < mysql-VERSION.tar.gz | tar xvf -
```

あるいは、`CMake` を使用して配布を圧縮解除してアンパックすることも可能です。

```
shell> cmake -E tar zxvf mysql-VERSION.tar.gz
```

- Zip アーカイブをアンパックするには、`WinZip` または `.zip` ファイルを読み取ることができるその他のツールを使用します。

配布ファイルをアンパックすると、`mysql-VERSION` という名前のディレクトリが作成されます。

配布の構成

アンパックした配布のトップレベルのディレクトリに場所を変更します。

```
shell> cd mysql-VERSION
```

ツリーをクリーンな状態に保つために、ソースツリーの外部にビルドします。最上位ソースディレクトリの名前が `mysql-src` で、現在の作業ディレクトリの下にある場合は、同じレベルの `bld` という名前のディレクトリにビルドできます。ディレクトリを作成してそこに移動します:

```
shell> mkdir bld
shell> cd bld
```

ビルドディレクトリを構成します。最低限の構成を行うコマンドには、構成のデフォルトをオーバーライドするオプションは含まれません。

```
shell> cmake ../mysql-src
```

ビルドディレクトリはソースツリーの外部にある必要はありません。たとえば、トップレベルのソースツリーの下の `bld` という名前のディレクトリにビルドできます。これを行うには、現在の作業ディレクトリとして `mysql-src` から開始し、ディレクトリ `bld` を作成してそこに移動します:

```
shell> mkdir bld
shell> cd bld
```

ビルドディレクトリを構成します。最低限の構成を行うコマンドには、構成のデフォルトをオーバーライドするオプションは含まれません。

```
shell> cmake ..
```

複数のソースツリーが同じレベルにある場合 (たとえば、複数のバージョンの MySQL をビルドする場合)、2 番目の方法が有利です。最初の方法では、すべてのビルドディレクトリを同じレベルに置くため、それぞれに一意の名前が必要です。2 番目の方法では、各ソースツリー内のビルドディレクトリに同じ名前を使用できます。次の手順では、この 2 番目の方法を想定しています。

Windows では、開発環境を指定します。たとえば、次のコマンドはそれぞれ 32 ビットまたは 64 ビットの MySQL ビルドを構成します。

```
shell> cmake .. -G "Visual Studio 12 2013"
shell> cmake .. -G "Visual Studio 12 2013 Win64"
```

macOS で Xcode IDE を使用するには:

```
shell> cmake .. -G Xcode
```

`cmake` を実行する場合、コマンド行にオプションを追加するとよいでしょう。次にいくつかの例を示します。

- `-DBUILD_CONFIG=mysql_release`: オラクルが公式な MySQL リリースのバイナリ配布を生成するために使用するのと同じビルドオプションでソースを構成します。
- `-DCMAKE_INSTALL_PREFIX=dir_name`: 特定の場所にインストールするように配布を構成します。
- `-DCPACK_MONOLITHIC_INSTALL=1`: `make package` が、複数のファイルではなく単独のインストールファイルを生成するようにします。
- `-DWITH_DEBUG=1`: 配付をデバッグサポート付きでビルドします。

オプションのより詳細なリストは、[セクション2.9.7「MySQL ソース構成オプション」](#)を参照してください。

構成オプションをリストするには、次のコマンドのいずれかを使用します。

```
shell> cmake .. -L # overview
shell> cmake .. -LH # overview with help text
shell> cmake .. -LAH # all params with help text
shell> ccmake .. # interactive display
```

`CMake` が失敗する場合は、異なるオプションで再実行して再構成する必要がある場合があります。再構成を行う場合は、次に注意してください。

- `CMake` を以前に実行したあとで実行すると、以前の起動時に収集した情報を使用する場合があります。この情報は `CMakeCache.txt` に格納されています。`CMake` が起動すると、情報がまだ正しいことを前提として、そのファイルが検索され、その内容 (存在する場合) が読み取られます。この仮定は再構成した場合には無効です。
- `CMake` を実行するたびに、`make` を再実行して再コンパイルする必要があります。しかし、前のビルドの古いオブジェクトファイルが異なる構成オプションでコンパイルされている場合、それらを最初に削除する場合があります。

古いオブジェクトファイルまたは構成情報が使用されないようにするには、`CMake` を再実行する前に、Unix でビルドディレクトリで次のコマンドを実行します:

```
shell> make clean
shell> rm CMakeCache.txt
```

あるいは、Windows の場合:

```
shell> devenv MySQL.sln /clean
shell> del CMakeCache.txt
```

[MySQL Community Slack](#) に問い合わせる前に、`CMakeFiles` ディレクトリ内のファイルで障害に関する有用な情報を確認します。バグレポートを提出する際は、[セクション1.6「質問またはバグをレポートする方法」](#)の説明に従ってください。

配布のビルド

Unix の場合:

```
shell> make
shell> make VERBOSE=1
```

2 番目のコマンドは、コンパイル済みの各ソースに対するコマンドを表示するため、`VERBOSE` をセットします。

GNU `make` を使用し、それが `gmake` としてインストールされているシステムでは、代わりに `gmake` を使用します。

Windows の場合:

```
shell> devenv MySQL.sln /build RelWithDebInfo
```

コンパイル段階に進んだが、配布がビルドされない場合は、[セクション2.9.8「MySQL のコンパイルに関する問題」](#)を参照してください。それでも問題が解決しない場合は、[セクション1.6「質問またはバグをレポートする方法」](#)を参照して、それをバグデータベースに入力してください。必要なツールの最新バージョンをインストール済みで、構成ファイルを処理しようとしてそれらのツールがクラッシュする場合は、それもレポートしてください。ただ

し、[コマンドが見つかりません](#)というエラーや、必要なツールに関して同様の問題がある場合は、レポートしないでください。代わりに、必要なツールがすべてインストール済みであり、シェルがそれらを検索できるように `PATH` 変数が正しく設定されていることを確認してください。

配付のインストール

Unix の場合:

```
shell> make install
```

これは、ファイルを構成されたインストールディレクトリ (デフォルトでは `/usr/local/mysql`) にインストールします。コマンドを `root` として実行する必要がある場合があります。

特定のディレクトリにインストールするには、コマンド行に `DESTDIR` パラメータを追加します。

```
shell> make install DESTDIR="/opt/mysql"
```

あるいは、任意の場所にインストールできるインストールパッケージファイルを生成します。

```
shell> make package
```

この操作は、一般的なバイナリ配布パッケージのようにインストールできる、1つまたは複数の `.tar.gz` ファイルを生成します。 [セクション2.2「一般的なバイナリを使用した MySQL の Unix/Linux へのインストール」](#) を参照してください。 `CMake` を `-DCPACK_MONOLITHIC_INSTALL=1` を使用して実行すると、その操作で単独のファイルが作成されます。 そうでない場合は複数のファイルが作成されます。

Windows では、データディレクトリを生成してから、`.zip` アーカイブインストールパッケージを作成します。

```
shell> devenv MySQL.sln /build RelWithDebInfo /project initial_database
shell> devenv MySQL.sln /build RelWithDebInfo /project package
```

結果の `.zip` アーカイブは任意の場所にインストールできます。 [セクション2.3.4「noinstall ZIP アーカイブを使用した Microsoft Windows への MySQL のインストール」](#) を参照してください。

インストール後のセットアップの実行

インストールプロセスの残りの部分は、構成ファイルのセットアップ、コアデータベースの作成、および MySQL Server の開始などです。 その手順は、 [セクション2.10「インストール後のセットアップとテスト」](#) を参照してください。

注記

MySQL の付与テーブルにリストされているアカウントには、最初はパスワードがありません。 サーバーの起動後に、 [セクション2.10「インストール後のセットアップとテスト」](#) の説明に従って、それらにパスワードを設定する必要があります。

2.9.5 開発ソースツリーを使用して MySQL をインストールする

このセクションでは、 [GitHub](#) でホストされている最新の開発ソースコードから MySQL をインストールする方法について説明します。 このリポジトリホスティングサービスから MySQL Server ソースコードを取得するには、ローカルの MySQL Git リポジトリを設定します。

[GitHub](#) では、MySQL Server およびその他の MySQL プロジェクトは [MySQL](#) ページにあります。 MySQL Server プロジェクトは、複数の MySQL シリーズのブランチを含む単一のリポジトリです。

MySQL は、2014 年 9 月に正式に [GitHub](#) に参加しました。 MySQL の [GitHub](#) への移動の詳細は、MySQL Release Engineering ブログ [MySQL on GitHub](#) のお知らせを参照してください。

- [開発ソースからのインストールの前提条件](#)
- [MySQL Git リポジトリのセットアップ](#)

開発ソースからのインストールの前提条件

開発ソースツリーから MySQL をインストールするには、システムが [セクション2.9.2「ソースインストールの前提条件」](#) にリストされているツール要件を満たしている必要があります。

MySQL Git リポジトリのセットアップ

マシンに MySQL Git リポジトリを設定するには:

1. MySQL Git リポジトリをマシンにクローニングします。次のコマンドは、MySQL Git リポジトリを `mysql-server` という名前のディレクトリにクローニングします。接続速度によっては、初期ダウンロードの完了に時間がかかる場合があります。

```
~$ git clone https://github.com/mysql/mysql-server.git
Cloning into 'mysql-server'...
remote: Counting objects: 1198513, done.
remote: Total 1198513 (delta 0), reused 0 (delta 0), pack-reused 1198513
Receiving objects: 100% (1198513/1198513), 1.01 GiB | 7.44 MiB/s, done.
Resolving deltas: 100% (993200/993200), done.
Checking connectivity... done.
Checking out files: 100% (25510/25510), done.
```

2. クローニング操作が完了すると、ローカルの MySQL Git リポジトリは次のようになります。

```
~$ cd mysql-server
~/mysql-server$ ls
client      extra       msys        storage
cmake       include     packaging   strings
CMakeLists.txt  INSTALL    plugin      support-files
components   libbinlogevents  README     testclients
config.h.cmake libbinlogstandalone  router     unittest
configure.cmake libmysql     run_doxygen.cmake  utilities
Docs         libservices  scripts      VERSION
Doxyfile-ignored LICENSE      share        vio
Doxyfile.in  man         sql          win
doxygen_resources  mysql-test  sql-common
```

3. `git branch -r` コマンドを使用して、MySQL リポジトリのリモート追跡ブランチを表示します。

```
~/mysql-server$ git branch -r
origin/5.5
origin/5.6
origin/5.7
origin/8.0
origin/HEAD -> origin/8.0
origin/cluster-7.2
origin/cluster-7.3
origin/cluster-7.4
origin/cluster-7.5
origin/cluster-7.6
```

4. ローカルリポジトリでチェックアウトされているブランチを表示するには、`git branch` コマンドを発行します。MySQL Git リポジトリをクローニングすると、最新の MySQL GA ブランチが自動的にチェックアウトされます。アスタリスクはアクティブなブランチを示します。

```
~/mysql-server$ git branch
* 8.0
```

5. 以前の MySQL ブランチをチェックアウトするには、ブランチ名を指定して `git checkout` コマンドを実行します。たとえば、MySQL 5.7 ブランチをチェックアウトするには:

```
~/mysql-server$ git checkout 5.7
Checking out files: 100% (9600/9600), done.
Branch 5.7 set up to track remote branch 5.7 from origin.
Switched to a new branch '5.7'
```

6. MySQL Git リポジトリを最初にセットアップしたあとの変更を取得するには、更新したいブランチに切り替えて `git pull` コマンドを発行します。

```
~/mysql-server$ git checkout 8.0
~/mysql-server$ git pull
```

コミット履歴を調べるには、`git log` オプションを使用します。

```
~/mysql-server$ git log
```

GitHub [MySQL](#) サイトでコミット履歴とソースコードを参照することもできます。

質問がある変更またはコードが表示された場合は、[MySQL Community Slack](#) に問い合わせてください。パッチの貢献については、[MySQL Server への貢献](#)を参照してください。

- MySQL Git リポジトリをクローニングしてビルドするブランチをチェックアウトしたら、MySQL Server をソースコードからビルドできます。説明は[セクション2.9.4「標準ソース配布を使用して MySQL をインストールする」](#)にあります。配布の取得およびアンパックに関する部分はスキップします。

本番環境マシン上の配布ソースツリーからのビルドをインストールする場合は注意が必要です。インストールコマンドが、ライブラリをインストールを上書きすることがあります。MySQL がインストール済みであり、それを上書きしたくない場合は、`CMAKE_INSTALL_PREFIX`、`MYSQL_TCP_PORT`、および `MYSQL_UNIX_ADDR` オプションの値を、本番環境サーバーで使用したものとは異なるものにして、`CMake` を実行します。複数のサーバーが相互に干渉することを防ぐ方法の詳細は、[セクション5.8「1 つのマシン上での複数の MySQL インスタンスの実行」](#)を参照してください。

新しいインストールを厳密に検証します。たとえば、新機能をクラッシュさせてみてください。 `make test` の実行から始めてみてください。 [The MySQL Test Suite](#)を参照してください。

2.9.6 SSL ライブラリサポートの構成

SSL ライブラリは、暗号化された接続、乱数生成のエントロピ、およびその他の暗号化関連の操作をサポートするために必要です。

ソース配布から MySQL をコンパイルする場合、`CMake` は、インストールされている OpenSSL ライブラリをデフォルトで使用するよう配布を構成します。

OpenSSL を使用してコンパイルするには、次の手順を使用します:

- OpenSSL 1.0.1 以上がシステムにインストールされていることを確認します。インストールされている OpenSSL のバージョンが 1.0.1 より低い場合、`CMake` は MySQL の構成時にエラーを生成します。OpenSSL を入手する必要がある場合は、<http://www.openssl.org> にアクセスしてください。
- `WITH_SSL` `CMake` オプションは、MySQL のコンパイルに使用する SSL ライブラリを決定します ([セクション 2.9.7「MySQL ソース構成オプション」](#)を参照)。デフォルトは、OpenSSL を使用する `-DWITH_SSL=system` です。これを明示的にするには、`CMake` コマンドラインでそのオプションを指定します。例:

```
cmake . -DWITH_SSL=system
```

このコマンドは、インストールされた OpenSSL ライブラリが使用されるように配布を構成します。または、OpenSSL インストールのパス名を明示的に指定するには、次の構文を使用します。これは、`CMake` が間違ったバージョンを選択しないようにするために、複数のバージョンの OpenSSL がインストールされている場合に役立ちます:

```
cmake . -DWITH_SSL=path_name
```

- 配布をコンパイルし、インストールします。

`mysqld` サーバーが暗号化された接続をサポートしているかどうかを確認するには、`have_ssl` システム変数の値を調べます:

```
mysql> SHOW VARIABLES LIKE 'have_ssl';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_ssl      | YES   |
+-----+-----+
```

値が `YES` の場合、サーバーは暗号化された接続をサポートします。値が `DISABLED` の場合、サーバーは暗号化された接続をサポートできますが、暗号化された接続を使用できるようにする適切な `--ssl-xxx` オプションで起動されませんでした。[セクション6.3.1「暗号化接続を使用するための MySQL の構成」](#)を参照してください。

2.9.7 MySQL ソース構成オプション

CMake プログラムを使用すると、MySQL ソース配布の構成方法を大幅に制御できます。通常これは、**CMake** コマンド行でオプションを使用して行います。**CMake** がサポートするオプションの詳細は、トップレベルのソースディレクトリで次のコマンドのいずれかを実行します。

```
cmake . -LH
ccmake .
```

特定の環境変数を使用して **CMake** に影響を与えることもできます。[セクション4.9「環境変数」](#)を参照してください。

boolean オプションでは、値を 1 または **ON** に指定してオプションを有効にするか、0 または **OFF** に指定して無効にします。

多くのオプションはコンパイル時のデフォルトを構成し、それらはサーバー起動時にオーバーライドできます。たとえば、デフォルトのインストールベースディレクトリ、TCP/IP ポート番号、および Unix ソケットファイルを構成する **CMAKE_INSTALL_PREFIX**、**MYSQL_TCP_PORT**、および **MYSQL_UNIX_ADDR** の各オプションは、サーバー起動時に **mysqld** の **--basedir**、**--port**、および **--socket** オプションで変更できます。該当する場合は、構成オプションの説明で対応する **mysqld** スタートアップオプションを示します。

次のセクションでは、**CMake** オプションについてより詳しく説明します。

- [CMake オプションリファレンス](#)
- [一般オプション](#)
- [インストールレイアウトオプション](#)
- [ストレージエンジンオプション](#)
- [機能オプション](#)
- [コンパイラフラグ](#)
- [NDB Cluster をコンパイルするための CMake オプション](#)

CMake オプションリファレンス

次の表は、使用可能な **CMake** オプションを示しています。**デフォルト**カラムで、**PREFIX** は **CMAKE_INSTALL_PREFIX** オプションの値を意味します。これは、インストールのベースディレクトリを指定します。この値は、一部のインストールサブディレクトリの親の場所として使用されます。

表 2.14 「MySQL ソース構成オプションリファレンス (CMake)」

形式	説明	デフォルト	導入	削除
ADD_GDB_INDEX	生成を有効にするかどうか。バイナリの gdb_index セクション		8.0.18	
BUILD_CONFIG	公式なリリースと同じビルドオプションを使用			
BUNDLE_RUNTIME_LIBS	ランタイムライブラリと Windows 用のサーバー MSI および Zip パッケージのバンドル	OFF		
CMAKE_BUILD_TYPE	生成するビルドのタイプ	RelWithDebInfo		
CMAKE_CXX_FLAGS	C++ コンパイラのフラグ			
CMAKE_C_FLAGS	C コンパイラのフラグ			
CMAKE_INSTALL_PREFIX	インストールの基本ディレクトリ	/usr/local/mysql		

形式	説明	デフォルト	導入	削除
COMPILATION_COMMENT	コンパイル環境に関するコメント			
COMPILATION_COMMENT	MySQLで使用するコンパイル環境に関するコメント		8.0.14	
COMPRESS_DEBUG	実行可能ファイルのデバッグセッションの圧縮	OFF	8.0.22	
CPACK_MONOLITHIC	パッケージビルドが単独のファイルを生成するかどうか	OFF		
DEFAULT_CHARSET	デフォルトのサーバー文字セット	utf8mb4		
DEFAULT_COLLATION	デフォルトのサーバー照合順序	utf8mb4_0900_ai_ci		
DISABLE_PSI_COND	Exclude Performance Schema 状態インストゥルメンテーション	OFF		
DISABLE_PSI_DATA_LOCK	パフォーマンススキーマデータロックインストゥルメンテーションを除外	OFF		
DISABLE_PSI_ERROR	パフォーマンススキーマサーバーエラーの計測を除外	OFF		
DISABLE_PSI_FILE	Exclude Performance Schema ファイルインストゥルメンテーション	OFF		
DISABLE_PSI_IDLE	Exclude Performance Schema アイドルインストゥルメンテーション	OFF		
DISABLE_PSI_MEMORY	Exclude Performance Schema メモリーインストゥルメンテーション	OFF		
DISABLE_PSI_METADATA	Exclude Performance Schema メタデータインストゥルメンテーション	OFF		
DISABLE_PSI_MUTEX	Exclude Performance Schema 相互排他ロックインストゥルメンテーション	OFF		
DISABLE_PSI_PS	パフォーマンススキーマのプリペアドステートメントの除外	OFF		
DISABLE_PSI_RWLOCK	Exclude Performance Schema rwlock イン	OFF		

形式	説明	デフォルト	導入	削除
	ストウルメンテーション			
DISABLE_PSI_SOCKETS	Exclude Performance Schema ソケットインストウルメンテーション	OFF		
DISABLE_PSI_SP	Exclude Performance Schema ストアドプログラムインストウルメンテーション	OFF		
DISABLE_PSI_STAGE	Exclude Performance Schema ステージングインストウルメンテーション	OFF		
DISABLE_PSI_STATEMENTS	Exclude Performance Schema ステートメントインストウルメンテーション	OFF		
DISABLE_PSI_STATEMENTS_DIGEST	Exclude Performance Schema statements_digest インストウルメンテーション	OFF		
DISABLE_PSI_TABLES	Exclude Performance Schema テーブルインストウルメンテーション	OFF		
DISABLE_PSI_THREADS	パフォーマンススキーマスレッドインストウルメンテーションを除外	OFF		
DISABLE_PSI_TRANSACTIONS	パフォーマンススキーマのトランザクションインストウルメンテーションの除外	OFF		
DISABLE_SHARED	共有ライブラリを構築せず、位置依存コードをコンパイル	OFF		8.0.18
DOWNLOAD_BOOST	Boost ライブラリをダウンロードするかどうか	OFF		
DOWNLOAD_BOOST_TIMEOUT	Boost ライブラリをダウンロードするためのタイムアウト (秒)	600		
ENABLED_LOCAL_INFILE	LOAD DATA の LOCAL を有効にするかどうか	OFF		
ENABLED_PROFILING	クエリープロファイリングコードを有効にするかどうか	ON		

形式	説明	デフォルト	導入	削除
ENABLE_DOWNLOADS	オプションファイルをダウンロードするかどうか	OFF		
ENABLE_EXPERIMENTAL_STORAGE_ENGINE	実験的な MySQL システム変数を有効にするかどうか	OFF		
ENABLE_GCOV	gcov サポートをインクルードするかどうか			
ENABLE_GPROF	gprof の有効化 (Linux ビルドのみに最適化)	OFF		
FORCE_INSOURCE_BUILD	インソース内ビルドを強制するかどうか	OFF	8.0.14	
FORCE_UNSUPPORTED_COMPILER	サポートされていないコンパイラを許可するかどうか	OFF		
FPROFILE_GENERATE	プロファイルガイド付き最適化データを生成するかどうか	OFF	8.0.19	
FPROFILE_USE	プロファイルガイド付き最適化データを使用するかどうか	OFF	8.0.19	
IGNORE_AIO_CHECK	- DBUILD_CONFIG=mysql_release とともに、libaio チェックを無視する	OFF		
INSTALL_BINDIR	ユーザー実行可能ファイルディレクトリ	PREFIX/bin		
INSTALL_DOCDIR	ドキュメントディレクトリ	PREFIX/docs		
INSTALL_DOCREADMEDIR	README ファイルディレクトリ	PREFIX		
INSTALL_INCLUDEDIR	Header ファイルディレクトリ	PREFIX/include		
INSTALL_INFODIR	Info ファイルディレクトリ	PREFIX/docs		
INSTALL_LAYOUT	事前定義インストールレイアウトの選択	STANDALONE		
INSTALL_LIBDIR	Library ファイルディレクトリ	PREFIX/lib		
INSTALL_MANDIR	Manual ページディレクトリ	PREFIX/man		
INSTALL_MYSQLKEYRINGDIR	Keyring file プラグインデータファイルのディレクトリ	プラットフォーム固有		
INSTALL_MYSQLSHAREDIR	共有データディレクトリ	PREFIX/share		
INSTALL_MYSQLTESTDIR	mysql-test ディレクトリ	PREFIX/mysql-test		

形式	説明	デフォルト	導入	削除
INSTALL_PKGCONFIG	mysqlclient.pc pkg-config ファイルのディレクトリ	INSTALL_LIBDIR/ pkgconfig		
INSTALL_PLUGIN_DIR	Plugin ディレクトリ	PREFIX/lib/plugin		
INSTALL_PRIV_LIBDIR	インストールプライベートライブラリディレクトリ		8.0.18	
INSTALL_SBINDIR	サーバー実行可能ファイルディレクトリ	PREFIX/bin		
INSTALL_SECURE_FILE	SecureFilePriv のデフォルト値	プラットフォーム固有		
INSTALL_SHAREDIR	aclocal/mysql.m4 インストールディレクトリ	PREFIX/share		
INSTALL_STATIC_LIB	静的ライブラリをインストールするかどうか	ON		
INSTALL_SUPPORTFILE	その他のサポートファイルディレクトリ	PREFIX/support-files		
LINK_RANDOMIZE	mysql バイナリ内のシンボルの順序をランダム化するかどうか	OFF		
LINK_RANDOMIZE_SEED	LINK_RANDOMIZE オプションのシード値	mysql		
MAX_INDEXES	テーブルあたりの最大インデックス数	64		
MUTEX_TYPE	InnoDB 相互排他ロックタイプ	event		
MYSQLX_TCP_PORT	X プラグインで使用される TCP/IP ポート番号	33060		
MYSQLX_UNIX_ADDR	X プラグインで使用される Unix ソケットファイル	/tmp/mysqlx.sock		
MYSQL_DATADIR	データディレクトリ			
MYSQL_MAINTAINER	MySQL 管理者固有の開発環境を有効にするかどうか	OFF		
MYSQL_PROJECT_NAME	Windows/macOS プロジェクト名	MySQL		
MYSQL_TCP_PORT	TCP/IP ポート番号	3306		
MYSQL_UNIX_ADDR	Unix ソケットファイル	/tmp/mysql.sock		
NDB_UTILS_LINK_DYNAMIC	NDB ツールを ndbclient に動的にリンクさせます		8.0.22-ndb-8.0.22	
ODBC_INCLUDES	ODBC インクルードディレクトリ			
ODBC_LIB_DIR	ODBC ライブラリディレクトリ			

形式	説明	デフォルト	導入	削除
OPTIMIZER_TRACE	オプティマイザのトレースをサポートするかどうか			
REPRODUCIBLE_BUILD	ビルドの場所と時間に関係なくビルド結果を作成するには、特に注意してください			
SYSCONFDIR	オプションファイルディレクトリ			
SYSTEMD_PID_DIR	systemd 下の PID ファイルのディレクトリ	/var/run/mysqld		
SYSTEMD_SERVICE_NAME	systemd の下の MySQL サービスの名前	mysqld		
TMPDIR	tmpdir のデフォルト値			
USE_LD_GOLD	GNU ゴールドリンカーを使用するかどうか	ON		
USE_LD_LLD	llvm lld リンカーを使用するかどうか	ON	8.0.16	
WIN_DEBUG_NO_INLINE	関数インライン化を無効にするかどうか	OFF		
WITHOUT_xxx_STORAGE_ENGINE	xxx をビルドから除外			
WITH_ANT	GCS Java ラッパーを構築するための Ant へのパス			
WITH_ASAN	AddressSanitizer 有効	OFF		
WITH_ASAN_SCOPE	AddressSanitizer - fsanitize-address-use-after-scope Clang フラグの有効化	OFF		
WITH_AUTHENTICATION_OPAE_PLUGIN	OPAe 認証プラグインを構築できない場合にエラーを報告するかどうか	OFF		
WITH_AUTHENTICATION_PAM_PLUGIN	PAM 認証プラグインのビルド	OFF		
WITH_AWS_SDK	Amazon Web Services ソフトウェア開発キットの場所			
WITH_BOOST	Boost ライブラリソースの場所			
WITH_CLIENT_PROTOCOL_CURL	クライアント側プロトコルトレースフレームワークのビルド	ON		
WITH_CURL	curl ライブラリの場所			

形式	説明	デフォルト	導入	削除
WITH_DEBUG	デバッグサポートをインクルードするか	OFF		
WITH_DEFAULT_COMPILER_OPTIONS	デフォルトのコンパイラオプションを使用するか	ON		
WITH_DEFAULT_FEATURES	デフォルトの機能セットを使用するか	ON		8.0.22
WITH_EDITLINE	どの libedit/editline ライブラリを使用するか	bundled		
WITH_GMOCK	googlemock 分布へのパス			
WITH_ICU	ICU サポートのタイプ	bundled		
WITH_INNOODB_EXTRA_DEBUG	InnoDB の追加のデバッグサポートをインクルードするか。	OFF		
WITH_INNOODB_MEMCACHED	Memcached 共有ライブラリを生成するかどうか。	OFF		
WITH_JEMALLOC	-jemalloc とリンクするか	OFF	8.0.16	
WITH_KEYRING_TEST	キーリングテストプログラムの作成	OFF		
WITH_LIBEVENT	どの libevent ライブラリを使用するか	bundled		
WITH_LIBWRAP	libwrap (TCP ラッパー) サポートをインクルードするか	OFF		
WITH_LOCK_ORDER	LOCK_ORDER ツールを有効にするかどうか	OFF	8.0.17	
WITH_LSAN	AddressSanitizer なしで LeakSanitizer を実行するか	OFF	8.0.16	
WITH_LTO	リンク時最適化の有効化	OFF	8.0.13	
WITH_LZ4	LZ4 ライブラリサポートのタイプ	bundled		
WITH_LZMA	LZMA ライブラリサポートのタイプ	bundled		8.0.16
WITH_MECAB	MeCab のコンパイル			
WITH_MSAN	MemorySanitizer の有効化	OFF		
WITH_MSVCRT_DEBUG	Visual Studio CRT メモリーリークのトレースを有効化	OFF		
WITH_MYSQLX	X プロトコルを無効にするかどうか	ON		

形式	説明	デフォルト	導入	削除
WITH_NUMA	NUMA メモリー割当てポリシーの設定			
WITH_PROTOBUF	使用するプロトコルバッファパッケージ	bundled		
WITH_RAPID	迅速な開発サイクルプラグインを構築するかどうか	ON		
WITH_RAPIDJSON	RapidJSON サポートのタイプ	bundled	8.0.13	
WITH_RE2	RE2 ライブラリサポートのタイプ	bundled		8.0.18
WITH_ROUTER	MySQL Router を構築するかどうか	ON	8.0.16	
WITH_SSL	SSL サポートのタイプ	system		
WITH_SYSTEMD	systemd サポートファイルのインストールを有効にします	OFF		
WITH_SYSTEMD_DEBUG	追加の systemd デバッグ情報の有効化	OFF	8.0.22	
WITH_SYSTEM_LIBS	明示的に設定されていないライブラリオプションのシステム値の設定	OFF		
WITH_TCMALLOC	-tcmalloc とリンクするかどうか	OFF	8.0.22	
WITH_TEST_TRACE_PLUGIN	テスト側プロトコルトレースプラグインのビルド	OFF		
WITH_TSAN	ThreadSanitizer の有効化	OFF		
WITH_UBSAN	未定義動作のサニタイザの有効化	OFF		
WITH_UNIT_TESTS	ユニットテストを使用した MySQL のコンパイル	ON		
WITH_UNIXODBC	unixODBC サポートの有効化	OFF		
WITH_VALGRIND	Valgrind ヘッダーファイルをコンパイルするかどうか	OFF		
WITH_ZLIB	zlib サポートのタイプ	bundled		
WITH_ZSTD	zstd サポートのタイプ	bundled	8.0.18	
WITH_XXX_STORAGE_ENGINE	XXX エンジン xxx をサーバーに静的にコンパイル			

一般オプション

- `-DBUILD_CONFIG=mysql_release`

このオプションは、オラクルが公式な MySQL リリースのバイナリ配布を生成するために使用するのと同じビルドオプションでソース配布を構成します。

- `-DBUNDLE_RUNTIME_LIBRARIES=bool`

ランタイムライブラリを Windows 用のサーバー MSI および Zip パッケージにバンドルするかどうか。

- `-DCMAKE_BUILD_TYPE=type`

生成するビルドのタイプ

- `RelWithDebInfo`: 最適化を有効にし、デバッグ情報を生成します。これはデフォルトの MySQL ビルドタイプです。
- `Release`: 最適化を有効にしますが、ビルドサイズを減らすためにデバッグ情報を省略します。このビルドタイプは、MySQL 8.0.13 で追加されました。
- `Debug`: 最適化を無効にし、デバッグ情報を生成します。このビルドタイプは、`WITH_DEBUG` オプションが有効な場合も使用されます。つまり、`-DWITH_DEBUG=1` は `-DCMAKE_BUILD_TYPE=Debug` と同じ効果を持ちます。
- `-DCPACK_MONOLITHIC_INSTALL=bool`

このオプションは、`make package` 操作が複数のインストールパッケージファイルを作成するか、単独のファイルを作成するかに影響します。無効の場合、この操作は複数のインストールパッケージファイルを作成します。これは完全な MySQL インストールのサブセットのみをインストールする場合に便利です。有効の場合、すべてをインストールするための単独のファイルを作成します。

- `-DFORCE_INSOURCE_BUILD=bool`

ソース内ビルドを強制するかどうかを定義します。アウトオブソースビルドでは同じソースからの複数のビルドが許可されるため、ビルドディレクトリを削除することでクリーンアップを迅速に実行できます。ソース内ビルドを強制するには、`-DFORCE_INSOURCE_BUILD=ON` を使用して `CMake` を起動します。

インストールレイアウトオプション

`CMAKE_INSTALL_PREFIX` オプションは、ベースインストールディレクトリを示します。コンポーネントの場所を示す、`INSTALL_xxx` という形式の名前を持つその他のオプションは、プリフィクスに相対的なものとして解釈され、その値は相対パス名です。それらの値はプリフィクスを含みません。

- `-DCMAKE_INSTALL_PREFIX=dir_name`

インストールのベースディレクトリ。

この値は、サーバー起動時に `--basedir` オプションで設定できます。

- `-DINSTALL_BINDIR=dir_name`

ユーザープログラムをインストールする場所。

- `-DINSTALL_DOCDIR=dir_name`

ドキュメントをインストールする場所。

- `-DINSTALL_DOCREADMEDIR=dir_name`

`README` ファイルをインストールする場所。

- `-DINSTALL_INCLUDEDIR=dir_name`

ヘッダーファイルをインストールする場所。

- `-DINSTALL_INFODIR=dir_name`

Info ファイルをインストールする場所。

- `-DINSTALL_LAYOUT=name`

事前定義インストールレイアウトを選択します。

- **STANDALONE**: `.tar.gz` および `.zip` パッケージで使用されるのと同じレイアウト。これはデフォルトです。
- **RPM**: RPM パッケージと同様のレイアウト。
- **SVR4**: Solaris パッケージレイアウト。
- **DEB**: DEB パッケージレイアウト (実験的)。

事前定義のレイアウトを選択できますが、ほかのオプションを指定することによって、個々のコンポーネントのインストール場所を変更できます。例:

```
cmake . -DINSTALL_LAYOUT=SVR4 -DMYSQL_DATADIR=/var/mysql/data
```

`INSTALL_LAYOUT` の値によって、`secure_file_priv`、`keyring_encrypted_file_data` および `keyring_file_data` システム変数のデフォルト値が決まります。 [セクション5.1.8「サーバースystem変数」](#) および [セクション6.4.4.13「キーリングシステム変数」](#) のこれらの変数の説明を参照してください。

- `-DINSTALL_LIBDIR=dir_name`

ライブラリファイルをインストールする場所。

- `-DINSTALL_MANDIR=dir_name`

マニュアルページをインストールする場所。

- `-DINSTALL_MYSQLKEYRINGDIR=dir_path`

`keyring_file` プラグインデータファイルの場所として使用するデフォルトのディレクトリ。デフォルト値はプラットフォーム固有で、`INSTALL_LAYOUT CMake` オプションの値によって異なります。 [セクション5.1.8「サーバースystem変数」](#) の `keyring_file_data` システム変数の説明を参照してください。

- `-DINSTALL_MYSQLSHAREDIR=dir_name`

共有データファイルをインストールする場所。

- `-DINSTALL_MYSQLTESTDIR=dir_name`

`mysql-test` ディレクトリをインストールする場所。このディレクトリのインストールを抑制するには、オプションを空の値 (`-DINSTALL_MYSQLTESTDIR=`) に明示的に設定します。

- `-DINSTALL_PKGCONFIGDIR=dir_name`

`pkg-config` で使用する `mysqlclient.pc` ファイルをインストールするディレクトリ。 `INSTALL_LIBDIR` が `/mysql` で終わっていないかぎり、デフォルト値は `INSTALL_LIBDIR/pkgconfig` です。この場合は、最初に削除されます。

- `-DINSTALL_PLUGINDIR=dir_name`

プラグインディレクトリの場所。

この値は、サーバー起動時に `--plugin_dir` オプションで設定できます。

- `-DINSTALL_PRIV_LIBDIR=dir_name`

動的ライブラリディレクトリの場所。

デフォルトの場所: RPM = `/usr/lib64/mysql/private/`、DEB = `/usr/lib/mysql/private/` および TAR = `lib/private/`。

このオプションは MySQL 8.0.18 で追加されました。

Protobuf 用: これはプライベートな場所であるため、ローダー (Linux 上の ld-linux.so など) はヘルプなしで `libprotobuf.so` ファイルを見つけることができない場合があります。ローダーをガイドするために、値 `$ORIGIN/..` `$INSTALL_PRIV_LIBDIR` を持つ `RPATH` が `mysqld` および `mysqlxtest` に追加されます。これはほとんどの場合に機能しますが、[Resource Group](#) 機能を使用する場合、`mysqld` は `setsuid` であり、ローダーは `$ORIGIN` を含む `RPATH` を無視します。これを解決するには、ターゲットの宛先が既知であるため、ディレクトリへの明示的なフルパスが `mysqld` の DEB および RPM バリエーションに設定されます。tarball インストールでは、`patchelf` などのツールを使用した `mysqld` のパッチ適用が必要です。

- `-DINSTALL_SBINDIR=dir_name`

`mysqld` サーバーをインストールする場所。

- `-DINSTALL_SECURE_FILE_PRIVDIR=dir_name`

`secure_file_priv` システム変数のデフォルト値。デフォルト値はプラットフォーム固有で、`INSTALL_LAYOUT` CMake オプションの値によって異なります。[セクション5.1.8「サーバーシステム変数」](#) の `secure_file_priv` システム変数の説明を参照してください。

- `-DINSTALL_SHAREDIR=dir_name`

`aclocal/mysqld.m4` をインストールする場所。

- `-DINSTALL_STATIC_LIBRARIES=bool`

静的ライブラリをインストールするかどうか。デフォルトは `ON` です。`OFF` に設定されている場合、これらのライブラリはインストールされません: `libmysqlclient.a`、`libmysqldservices.a`。

- `-DINSTALL_SUPPORTFILESDIR=dir_name`

追加のサポートファイルをインストールする場所。

- `-DLINK_RANDOMIZE=bool`

`mysqld` バイナリ内のシンボルの順序をランダム化するかどうか。デフォルトは `OFF` です。このオプションは、デバッグ目的でのみ有効にしてください。

- `-DLINK_RANDOMIZE_SEED=val`

`LINK_RANDOMIZE` オプションのシード値。値は文字列です。デフォルトは、任意の選択肢である `mysql` です。

- `-DMYSQL_DATADIR=dir_name`

MySQL データディレクトリの場所。

この値は、サーバー起動時に `--datadir` オプションで設定できます。

- `-DODBC_INCLUDES=dir_name`

ODBC インクルードディレクトリの場所。Connector/ODBC の構成中に使用されることがあります。

- `-DODBC_LIB_DIR=dir_name`

ODBC ライブラリディレクトリの場所。Connector/ODBC の構成中に使用されることがあります。

- `-DSYSCONFDIR=dir_name`

デフォルトの `my.cnf` オプションファイルディレクトリ。

この場所はサーバー起動時にはセットできませんが、`--defaults-file=file_name` オプションを使用して、指定されたオプションファイルでサーバーを起動できます。ここで、`file_name` はファイルへのフルパス名です。

- `-DSYSTEMD_PID_DIR=dir_name`

`systemd` によって MySQL が管理されている場合に PID ファイルを作成するディレクトリの名前。デフォルトは `/var/run/mysqld` です。これは、`INSTALL_LAYOUT` の値に従って暗黙的に変更される場合があります。

`WITH_SYSTEMD` が有効になっていないかぎり、このオプションは無視されます。

- `-DSYSTEMD_SERVICE_NAME=name`

MySQL が systemd によって管理されている場合に使用する MySQL サービスの名前。デフォルトは `mysqld` です。これは、`INSTALL_LAYOUT` の値に従って暗黙的に変更される場合があります。

`WITH_SYSTEMD` が有効になっていないかぎり、このオプションは無視されます。

- `-DTMPDIR=dir_name`

`tmpdir` システム変数に使用されるデフォルトの場所。指定しない場合は、値はデフォルトで `P_tmpdir` in `<stdio.h>` になります。

ストレージエンジンオプション

ストレージエンジンはプラグインとしてビルドされます。プラグインは、静的モジュール (サーバー内にコンパイル) または動的モジュール (使用する前に、`INSTALL_PLUGIN` ステートメントまたは `--plugin-load` オプションを使用してサーバーにインストールする必要のあるダイナミックライブラリとしてビルド) としてビルドできます。一部のプラグインは、静的または動的ビルドをサポートしない場合があります。

InnoDB, MyISAM, MERGE, MEMORY および CSV エンジンは必須 (常にサーバーにコンパイルされる) であり、明示的にインストールする必要はありません。

ストレージエンジンをサーバー内に静的にコンパイルするには、`-DWITH_engine_STORAGE_ENGINE=1` を使用します。使用可能な `engine` 値には、`ARCHIVE`, `BLACKHOLE`, `EXAMPLE`, `FEDERATED`、`NDB` または `NDBCLUSTER` (`NDB` サポート) があります。例:

```
-DWITH_ARCHIVE_STORAGE_ENGINE=1
-DWITH_BLACKHOLE_STORAGE_ENGINE=1
```

注記

パフォーマンススキーマサポートなしでコンパイルすることはできません。特定のタイプのインストールメンテーションなしでコンパイルする場合は、次の `CMake` オプションを使用して実行できます:

```
DISABLE_PSI_COND
DISABLE_PSI_DATA_LOCK
DISABLE_PSI_ERROR
DISABLE_PSI_FILE
DISABLE_PSI_IDLE
DISABLE_PSI_MEMORY
DISABLE_PSI_METADATA
DISABLE_PSI_MUTEX
DISABLE_PSI_PS
DISABLE_PSI_RWLOCK
DISABLE_PSI_SOCKET
DISABLE_PSI_SP
DISABLE_PSI_STAGE
DISABLE_PSI_STATEMENT
DISABLE_PSI_STATEMENT_DIGEST
DISABLE_PSI_TABLE
DISABLE_PSI_THREAD
DISABLE_PSI_TRANSACTION
```

たとえば、`mutex` インストールメンテーションなしでコンパイルするには、`-DDISABLE_PSI_MUTEX=1` オプションを使用して MySQL を構成します。

ストレージエンジンをビルドから除外するには、`-DWITH_engine_STORAGE_ENGINE=0` を使用します。例:

```
-DWITH_ARCHIVE_STORAGE_ENGINE=0
-DWITH_EXAMPLE_STORAGE_ENGINE=0
-DWITH_FEDERATED_STORAGE_ENGINE=0
```

`-DWITHOUT_engine_STORAGE_ENGINE=1` を使用して構築からストレージエンジンを除外することもできます (ただし、`-DWITH_engine_STORAGE_ENGINE=0` をお勧めします)。例:


```
-DWITHOUT_ARCHIVE_STORAGE_ENGINE=1  
-DWITHOUT_EXAMPLE_STORAGE_ENGINE=1  
-DWITHOUT_FEDERATED_STORAGE_ENGINE=1
```

あるストレージエンジンに対して `-DWITH_engine_STORAGE_ENGINE` も `-DWITHOUT_engine_STORAGE_ENGINE` も指定されていない場合、そのエンジンは共有モジュールとしてビルドされるか、あるいは共有モジュールとしてビルドできない場合は除外されます。

機能オプション

- `-DADD_GDB_INDEX=bool`

このオプションは、バイナリでの `.gdb_index` セクションの生成を有効にするかどうかを決定します。これにより、デバッガへのロードが高速になります。このオプションはデフォルトで無効になっています。 `lld` リンカーが使用され、 `lld` または GNU `gold` 以外のリンカーが使用されている場合は無効になります。

このオプションは MySQL 8.0.18 で追加されました。

- `-DCOMPILATION_COMMENT=string`

コンパイル環境に関する説明コメント。MySQL 8.0.14 では、 `mysqld` は `COMPILATION_COMMENT_SERVER` を使用します。他のプログラムは引き続き `COMPILATION_COMMENT` を使用します。

- `-DCOMPRESS_DEBUG_SECTIONS=bool`

バイナリ実行可能ファイルのデバッグセクションを圧縮するかどうか (Linux のみ)。実行可能デバッグセクションを圧縮すると、構築プロセス中に余分な CPU 時間がかかるため、領域を節約できます。

デフォルトは `OFF` です。このオプションが明示的に設定されていないが、 `COMPRESS_DEBUG_SECTIONS` 環境変数が設定されている場合、このオプションはその変数から値を取得します。

このオプションは MySQL 8.0.22 で追加されました。

- `-DCOMPILATION_COMMENT_SERVER=string`

`mysqld` で使用するコンパイル環境に関する説明コメント (`version_comment` システム変数の設定など)。このオプションは MySQL 8.0.14 で追加されました。8.0.14 より前では、サーバーは `COMPILATION_COMMENT` を使用します。

- `-DDEFAULT_CHARSET=charset_name`

サーバーの文字セット。デフォルトでは、MySQL は `utf8mb4` 文字セットを使用します。

`charset_name`

は、 `binary`、 `armscii8`、 `ascii`、 `big5`、 `cp1250`、 `cp1251`、 `cp1256`、 `cp1257`、 `cp850`、 `cp852`、 `cp866`、 `cp932`、 `dec8`、 `eucjms`、 `eu` のいずれかにできます。許可される文字セットは、 `cmake/character_sets.cmake` ファイルに `CHARSETS_AVAILABLE` の値としてリストされています。

この値は、サーバー起動時に `--character_set_server` オプションで設定できます。

- `-DDEFAULT_COLLATION=collation_name`

サーバーの照合順序。デフォルトでは、MySQL は `utf8mb4_0900_ai_ci` を使用します。各文字セットにどの照合順序を使用するかを決めるには `SHOW COLLATION` ステートメントを使用します。

この値は、サーバー起動時に `--collation_server` オプションで設定できます。

- `-DDISABLE_PSI_COND=bool`

パフォーマンススキーマ条件インストゥルメンテーションを除外するかどうか。デフォルトは `OFF` (`include`) です。

- `-DDISABLE_PSI_FILE=bool`

パフォーマンススキーマファイルインストールメンターションを除外するかどうか。デフォルトは **OFF** (include) です。

- **-DDISABLE_PSI_IDLE=bool**

パフォーマンススキーマアイドルインストールメンターションを除外するかどうか。デフォルトは **OFF** (include) です。

- **-DDISABLE_PSI_MEMORY=bool**

パフォーマンススキーマメモリーインストールメンターションを除外するかどうか。デフォルトは **OFF** (include) です。

- **-DDISABLE_PSI_METADATA=bool**

パフォーマンススキーマメタデータの計測を除外するかどうか。デフォルトは **OFF** (include) です。

- **-DDISABLE_PSI_MUTEX=bool**

パフォーマンススキーマミューテックスのインストールメンターションを除外するかどうか。デフォルトは **OFF** (include) です。

- **-DDISABLE_PSI_RWLOCK=bool**

パフォーマンススキーマ `rwlock` インストールメンターションを除外するかどうか。デフォルトは **OFF** (include) です。

- **-DDISABLE_PSI_SOCKET=bool**

パフォーマンススキーマソケットインストールメンターションを除外するかどうか。デフォルトは **OFF** (include) です。

- **-DDISABLE_PSI_SP=bool**

パフォーマンススキーマストアプログラムインストールメンターションを除外するかどうか。デフォルトは **OFF** (include) です。

- **-DDISABLE_PSI_STAGE=bool**

パフォーマンススキーマステージインストールメンターションを除外するかどうか。デフォルトは **OFF** (include) です。

- **-DDISABLE_PSI_STATEMENT=bool**

パフォーマンススキーマステートメントインストールメンターションを除外するかどうか。デフォルトは **OFF** (include) です。

- **-DDISABLE_PSI_STATEMENT_DIGEST=bool**

パフォーマンススキーマ `statement_digest` インストールメンターションを除外するかどうか。デフォルトは **OFF** (include) です。

- **-DDISABLE_PSI_TABLE=bool**

「パフォーマンススキーマ」テーブルインストールメンターションを除外するかどうか。デフォルトは **OFF** (include) です。

- **-DDISABLE_SHARED=bool**

ビルド共有ライブラリを無効にし、位置依存コードをコンパイルするかどうか。デフォルトは **OFF** (位置独立コードのコンパイル) です。

このオプションは未使用であり、MySQL 8.0.18 で削除されました。

- **-DDISABLE_PSI_PS=bool**

パフォーマンススキーマプリヘアドステートメントのインスタンスインストールメンテーションを除外します。デフォルトは **OFF** (include) です。

- **-DDISABLE_PSI_THREAD=bool**

パフォーマンススキーマスレッドインストールメンテーションを除外します。デフォルトは **OFF** (include) です。

ほかのインストールメンテーションはスレッドに依存しているため、インストールメンテーションなしで構築する場合にのみスレッドを無効にします。

- **-DDISABLE_PSI_TRANSACTION=bool**

パフォーマンススキーマのトランザクションインストールメンテーションを除外します。デフォルトは **OFF** (include) です。

- **-DDISABLE_PSI_DATA_LOCK=bool**

パフォーマンススキーマデータロックインストールメンテーションを除外します。デフォルトは **OFF** (include) です。

- **-DDISABLE_PSI_ERROR=bool**

パフォーマンススキーマサーバーエラーの計測を除外します。デフォルトは **OFF** (include) です。

- **-DDOWNLOAD_BOOST=bool**

Boost ライブラリをダウンロードするかどうか デフォルトは **OFF** です。

Boost の使用方法の詳細は、[WITH_BOOST](#) オプションを参照してください。

- **-DDOWNLOAD_BOOST_TIMEOUT=seconds**

Boost ライブラリをダウンロードするためのタイムアウト (秒)。デフォルトは 600 秒です。

Boost の使用方法の詳細は、[WITH_BOOST](#) オプションを参照してください。

- **-DENABLE_DOWNLOADS=bool**

オプションファイルをダウンロードするかどうか。たとえば、このオプションを有効にすると、**CMake** は、ユニットテストを実行するためにテストスイートで使用される Google テストディストリビューション、または GCS Java ラッパーの構築に必要な Ant と JUnit をダウンロードします。

- **-DENABLE_EXPERIMENTAL_SYSVARS=bool**

実験的な **InnoDB** システム変数を有効にするかどうか。試験的なシステム変数は、MySQL 開発に關与するものを対象としており、開発環境またはテスト環境でのみ使用する必要があり、将来の MySQL リリースでは予告なしに削除される可能性があります。実験システム変数については、MySQL ソースツリーの [/storage/innobase/handler/ha_innodb.cc](#) を参照してください。試験的なシステム変数は、「**PLUGIN_VAR_EXPERIMENTAL**」を検索することで識別できます。

- **-DENABLE_GCOV=bool**

gcov サポートをインクルードするかどうか (Linux のみ)。

- **-DENABLE_GPROF=bool**

gprof を有効にするかどうか (最適化された Linux ビルドのみ)。

- `-DENABLED_LOCAL_INFILE=bool`

このオプションは、MySQL クライアントライブラリのコンパイル済みのデフォルトの `LOCAL` 機能を制御します。したがって、明示的な配置を行わないクライアントでは、MySQL ビルド時に指定された `ENABLED_LOCAL_INFILE` 設定に従って、`LOCAL` 機能が無効または有効になります。

デフォルトでは、MySQL バイナリディストリビューションのクライアントライブラリは、`ENABLED_LOCAL_INFILE` を無効にしてコンパイルされます。ソースから MySQL をコンパイルする場合は、明示的な配置を行わないクライアントで `LOCAL` 機能をそれぞれ無効にするか有効にするかに基づいて、`ENABLED_LOCAL_INFILE` を無効または有効にして構成します。

`ENABLED_LOCAL_INFILE` は、クライアント側の `LOCAL` 機能のデフォルトを制御します。サーバーの場合、`local_infile` システム変数はサーバー側の `LOCAL` 機能を制御します。サーバーが `LOAD DATA LOCAL` ステートメントを明示的に拒否または許可するようにするには (構築時または実行時にクライアントプログラムおよびライブラリがどのように構成されているかに関係なく)、それぞれ `local_infile` を無効または有効にして `mysqld` を起動します。`local_infile` は実行時に設定することもできます。セクション6.1.6「`LOAD DATA LOCAL` のセキュリティ上の考慮事項」を参照してください。

- `-DENABLED_PROFILING=bool`

クエリープロファイリングコードを有効にするかどうか (`SHOW PROFILE` および `SHOW PROFILES` ステートメントで)。

- `-DFORCE_UNSUPPORTED_COMPILER=bool`

デフォルトでは、`CMake` はサポートされているコンパイラの最小バージョンをチェック: Visual Studio 2015 (Windows)、GCC 4.8 または Clang 3.4 (Linux)、Developer Studio 12.5 (Solaris サーバー)、Developer Studio 12.4 または GCC 4.8 (Solaris クライアントライブラリ)、Clang 3.6 (macOS)、Clang 3.4 (FreeBSD)。このチェックを無効にするには、`-DFORCE_UNSUPPORTED_COMPILER=ON` を使用します。

- `-DFPROFILE_GENERATE=bool`

プロファイルガイド付き最適化 (PGO) データを生成するかどうか。このオプションは、GCC を使用した PGO の実験に使用できます。`FPROFILE_GENERATE` および `FPROFILE_USE` の使用の詳細は、MySQL ソース配布の `cmake/fprofile.cmake` ファイルを参照してください。これらのオプションは GCC 8 および 9 でテストされています。

このオプションは MySQL 8.0.19 で追加されました。

- `-DFPROFILE_USE=bool`

プロファイルガイド付き最適化 (PGO) データを使用するかどうか。このオプションは、GCC を使用した PGO の実験に使用できます。`FPROFILE_GENERATE` および `FPROFILE_USE` の使用の詳細は、MySQL ソース配布の `cmake/fprofile.cmake` ファイルを参照してください。これらのオプションは GCC 8 および 9 でテストされています。

`FPROFILE_USE` を有効にすると、`WITH_LTO` も有効になります。

このオプションは MySQL 8.0.19 で追加されました。

- `-DIGNORE_AIO_CHECK=bool`

Linux で `-DBUILD_CONFIG=mysql_release` オプションが与えられた場合、デフォルトで `libaio` ライブラリがリンクされていなければなりません。`libaio` がない場合、またはインストールしない場合、`-DIGNORE_AIO_CHECK=1` を指定するとそのチェックを抑制できます。

- `-DMAX_INDEXES=num`

テーブル当たりのインデックスの最大数。デフォルトは 64 です。最大値は 255 です。64 より小さい値は無視され、デフォルトの 64 が使用されます。

- `-DMYSQL_MAINTAINER_MODE=bool`

MySQL 管理者固有の開発環境を有効にするかどうか。有効の場合、このオプションによりコンパイラの警告はエラーになります。

- `-DMUTEX_TYPE=type`

InnoDB で使用される mutex タイプ。次のようなオプションがあります。

- `event`: イベント相互排他ロックを使用します。これはデフォルト値であり、元の InnoDB mutex 実装です。
- `sys`: UNIX システムで POSIX mutex を使用します。使用可能な場合は、Windows で `CRITICAL_SECTION` objects を使用します。
- `futex`: 待機スレッドをスケジュールするには、条件変数のかわりに Linux futex を使用します。

- `-DMYSQLX_TCP_PORT=port_num`

X プラグイン が TCP/IP 接続をリスニングするポート番号。デフォルトは 33060 です。

この値は、サーバーの起動時に `mysqlx_port` システム変数を使用して設定できます。

- `-DMYSQLX_UNIX_ADDR=file_name`

サーバーが X プラグイン ソケット接続をリスニングする Unix ソケットファイルのパス。これは絶対パス名でなければなりません。デフォルトは `/tmp/mysqlx.sock` です。

この値は、サーバーの起動時に `mysqlx_port` システム変数を使用して設定できます。

- `-DMYSQL_PROJECT_NAME=name`

Windows または macOS の場合、プロジェクトファイル名に組み込むプロジェクト名。

- `-DMYSQL_TCP_PORT=port_num`

サーバーが TCP/IP 接続を待機するポート番号。デフォルトは 3306 です。

この値は、サーバー起動時に `--port` オプションで設定できます。

- `-DMYSQL_UNIX_ADDR=file_name`

サーバーがソケット接続を待機する Unix ソケットファイルのパス。これは絶対パス名でなければなりません。デフォルトは `/tmp/mysql.sock` です。

この値は、サーバー起動時に `--socket` オプションで設定できます。

- `-DOPTIMIZER_TRACE=bool`

オプティマイザのトレースをサポートするかどうか。「[MySQL Internals: Tracing the Optimizer](#)」を参照してください。

- `-DREPRODUCIBLE_BUILD=bool`

Linux システム上のビルドの場合、このオプションはビルドの場所と時間に関係なくビルド結果を作成するために特別な注意を払うかどうかを制御します。

このオプションは MySQL 8.0.11 で追加されました。MySQL 8.0.12 では、`RelWithDebInfo` ビルド用の `ON` にデフォルト設定されます。

- `-DUSE_LD_GOLD=bool`

GNU `gold` リンカーが使用可能で明示的に無効になっていない場合、`CMake` は構築プロセスを GNU `gold` リンカーにリンクします。このリンカーの使用を無効にするには、`-DUSE_LD_GOLD=OFF` オプションを指定します。

- `-DUSE_LD_LLD=bool`

構築プロセスが使用可能であり、明示的に無効になっていない場合、CMake はその構築プロセスを Clang 用の `llvm-ld` リンカーにリンクします。このリンカーの使用を無効にするには、`-DUSE_LD_LLD=OFF` オプションを指定します。

このオプションは MySQL 8.0.16 で追加されました。

- `-DWIN_DEBUG_NO_INLINE=bool`

Windows で関数のインライン化を無効にするかどうか。デフォルトは `off` (インライン化が有効) です。

- `-DWITH_ANT=path_name`

GCS Java ラッパーの構築時に必要な Ant へのパスを設定します。既存の `WITH_BOOST` CMake オプションと同様の方法で動作します。Ant tarball またはすでに解凍されたアーカイブが保存されているディレクトリのパスに `WITH_ANT` を設定します。 `WITH_ANT` が設定されていない場合、または特別な値 `system` を使用して設定されている場合、ビルドでは `$PATH` にバイナリ `ant` が存在すると想定されます。

- `-DWITH_ASAN=bool`

AddressSanitizer をサポートするコンパイラに対して有効にするかどうか。デフォルトはオフです。

- `-DWITH_ASAN_SCOPE=bool`

use-after-scope 検出に対して AddressSanitizer `-fsanitize-address-use-after-scope` Clang フラグを有効にするかどうか。デフォルトはオフです。このオプションを使用するには、`-DWITH_ASAN` も有効にする必要があります。

- `-DWITH_AUTHENTICATION_LDAP=bool`

LDAP 認証プラグインを構築できない場合にエラーを報告するかどうか:

- このオプションを無効 (デフォルト) にすると、必要なヘッダーファイルおよびライブラリが見つかった場合に LDAP プラグインが構築されます。そうでない場合は、CMake によってノートが表示されます。
- このオプションが有効な場合、必要なヘッダーファイルおよびライブラリが見つからないと、CMake でエラーが生成され、サーバーのビルドが妨げられます。

- `-DWITH_AUTHENTICATION_PAM=bool`

このプラグインを含むソースツリーの PAM 認証プラグインを構築するかどうか。(セクション 6.4.1.5 「PAM プラグイン認証」を参照してください。) このオプションが指定され、プラグインをコンパイルできない場合、ビルドは失敗します。

- `-DWITH_AWS_SDK=path_name`

Amazon Web Services ソフトウェア開発キットの場所。

- `-DWITH_BOOST=path_name`

MySQL をビルドするには Boost ライブラリが必要です。これらの CMake オプションを使用すると、ライブラリのソースの場所、およびそれを自動的にダウンロードするかどうかを制御できます:

- `-DWITH_BOOST=path_name` では、ブーストライブラリディレクトリの場所を指定します。 `BOOST_ROOT` または `WITH_BOOST` 環境変数を設定してブーストの場所を指定することもできます。

`-DWITH_BOOST=system` も許可され、コンパイルホストの標準の場所に正しいバージョンの Boost がインストールされていることを示します。この場合、MySQL ソース配布に含まれる任意のバージョンではなく、インストールされている Boost のバージョンが使用されます。

- `-DDOWNLOAD_BOOST=bool` は、Boost ソースが指定された場所に存在しない場合にダウンロードするかどうかを指定します。デフォルトは `OFF` です。
- Boost ライブラリをダウンロードするためのタイムアウトを秒単位で `-DDOWNLOAD_BOOST_TIMEOUT=seconds` に設定します。デフォルトは 600 秒です。

たとえば、通常、MySQL ソースツリーの `bld` サブディレクトリにオブジェクト出力を配置して MySQL をビルドする場合は、次のように Boost を使用してビルドできます:

```
mkdir bld
cd bld
cmake .. -DDOWNLOAD_BOOST=ON -DWITH_BOOST=$HOME/my_boost
```

これにより、Boost がホームディレクトリの下での `my_boost` ディレクトリにダウンロードされます。必要な Boost バージョンがすでに存在する場合、ダウンロードは行われません。必要な Boost バージョンが変更されると、新しいバージョンがダウンロードされます。

Boost がすでにローカルにインストールされていて、コンパイラが Boost ヘッダーファイルを単独で検出した場合は、前述の CMake オプションを指定する必要はありません。ただし、MySQL に必要な Boost のバージョンが変更され、ローカルにインストールされたバージョンがアップグレードされていない場合は、ビルドの問題が発生している可能性があります。CMake オプションを使用すると、ビルドが成功します。

指定した場所へのブーストダウンロードを許可する前述の設定では、必要なブーストバージョンが変更された場合、`bld` フォルダを削除して再作成し、`cmake` ステップを再度実行する必要があります。そうしないと、新しい Boost バージョンがダウンロードされず、コンパイルが失敗する可能性があります。

- `-DWITH_CLIENT_PROTOCOL_TRACING=bool`

クライアントライブラリにクライアント側プロトコルトレースフレームワークをビルドするかどうか。デフォルトでは、このオプションは有効です。

プロトコルトレースクライアントプラグインの記述の詳細は、[Writing Protocol Trace Plugins](#) を参照してください。

`WITH_TEST_TRACE_PLUGIN` オプションも参照してください。

- `-DWITH_CURL=curl_type`

`curl` ライブラリの場所。`curl_type` は、`system` (システム `curl` ライブラリを使用) または `curl` ライブラリへのパス名です。

- `-DWITH_DEBUG=bool`

デバッグサポートを含めるかどうか。

デバッグサポートを有効にして MySQL を構成することにより、サーバーを起動するときに `--debug="d.parser_debug"` オプションを使用できるようになります。これにより、SQL ステートメントの処理に使用される Bison パーサーが、パーサトレースをサーバーの標準エラー出力にダンプします。一般的に、この出力はエラーログに書き込まれます。

InnoDB ストレージエンジンの同期デバッグチェックは `UNIV_DEBUG` で定義され、`WITH_DEBUG` オプションを使用してデバッグサポートをコンパイルするときに使用できます。デバッグサポートがでコンパイルされている場合は、`innodb_sync_debug` 構成オプションを使用して、InnoDB 同期デバッグチェックを有効または無効にできます。

`WITH_DEBUG` を有効にすると、デバッグ同期も有効になります。この機能はテストとデバッグに使用されます。コンパイルされる場合、実行時には Debug Sync はデフォルトで無効です。有効にするには、`mysqld` を `--debug-sync-timeout=N` オプションを使用して起動します。ここで、`N` は 0 より大きいタイムアウト値です。(デフォルト値は 0 で、Debug Sync を無効にします。) `N` はそれぞれの同期ポイントのデフォルトのタイムアウトになります。

InnoDB ストレージエンジンの同期デバッグチェックは、`WITH_DEBUG` オプションを使用してデバッグサポートがコンパイルされている場合に使用できます。

Debug Sync 機能および同期点の使用法についての説明は、「[MySQL Internals: Test Synchronization](#)」を参照してください。

- `-DWITH_DEFAULT_FEATURE_SET=bool`

`cmake/build_configurations/feature_set.cmake` からのフラグを使用するかどうか。このオプションは MySQL 8.0.22 で削除されました。

- `-DWITH_EDITLINE=value`

どの `libedit/editline` ライブラリを使用するか。許可される値は、`bundled` (デフォルト) および `system` です。

- `-DWITH_GMOCK=path_name`

Google テストベースのユニットテストで使用する `googlemock` デイストリビューションへのパス。オプション値は、配布 Zip ファイルへのパスです。または、`WITH_GMOCK` 環境変数をパス名に設定します。CMake が GitHub からデイストリビューションをダウンロードできるように、`-DENABLE_DOWNLOADS=1` を使用することもできます。

Google テストベースのユニットテストを使用せずに (`WITH_GMOCK` を構成して) MySQL をビルドすると、CMake によってダウンロード方法を示すメッセージが表示されます。

- `-DWITH_ICU={icu_type|path_name}`

MySQL では、International Components for Unicode (ICU) を使用して正規表現操作をサポートしています。`WITH_ICU` オプションは、含める ICU サポートのタイプまたは使用する ICU インストールのパス名を示します。

- `icu_type` は、次のいずれかの値です:

- `bundled`: 配布にバンドルされている ICU ライブラリを使用します。これはデフォルトであり、Windows でサポートされている唯一のオプションです。

- `system`: システム ICU ライブラリを使用します。

- `path_name` は、使用する ICU インストールのパス名です。これは、CMake がシステムにインストールされている古いバージョンまたは正しくない ICU バージョンを検出して使用することを防ぐことができるため、`system` の `icu_type` 値を使用するよりも望ましい場合があります。(同じことを可能にする別の方法は、`WITH_ICU` を `system` に設定し、`CMAKE_PREFIX_PATH` オプションを `path_name` に設定することです。)

- `-DWITH_INNODB_EXTRA_DEBUG=bool`

追加の InnoDB デバッグサポートを含めるかどうか。

`WITH_INNODB_EXTRA_DEBUG` を有効にすると、追加の InnoDB デバッグチェックが有効になります。このオプションは、`WITH_DEBUG` が有効な場合にのみ有効にできます。

- `-DWITH_INNODB_MEMCACHED=bool`

`memcached` 共有ライブラリ (`libmemcached.so` および `innodb_engine.so`) を生成するかどうか。

- `-DWITH_JEMALLOC=bool`

`-ljemalloc` とリンクするかどうか。有効にすると、組み込み `malloc()`, `calloc()`, `realloc()` および `free()` ルーチンは無効になります。デフォルトは `OFF` です。

`WITH_JEMALLOC` と `WITH_TCMALLOC` は相互に排他的です。

このオプションは MySQL 8.0.16 で追加されました。

- `-DWITH_KEYRING_TEST=bool`

`keyring_file` プラグインに付属するテストプログラムをビルドするかどうか。デフォルトは `OFF` です。テストファイルのソースコードは、`plugin/keyring/keyring-test` ディレクトリにあります。

- `-DWITH_LIBEVENT=string`

どの `libevent` ライブラリを使用するか。許可される値は、`bundled` (デフォルト) および `system` です。MySQL 8.0.21 より前では、`system` を指定すると、システム `libevent` ライブラリが存在する場合はそれが使用され、それ以外

外の場合はエラーが発生します。MySQL 8.0.21 以降では、`system` が指定されていて、システム `libevent` ライブラリが見つからない場合、エラーが発生し、バンドルされている `libevent` は使用されません。

`libevent` ライブラリは、InnoDB memcached、X プラグイン、および MySQL Router に必要です。

- `-DWITH_LIBWRAP=bool`

`libwrap` (TCP ラッパー) サポートを含めるかどうか。

- `-DWITH_LOCK_ORDER=bool`

`LOCK_ORDER` ツールを有効にするかどうか。デフォルトでは、このオプションは無効になっており、サーバービルドにツールは含まれていません。ツールが有効な場合、`LOCK_ORDER` ツールは使用可能であり、[セクション 5.9.3 「LOCK_ORDER ツール」](#) で説明されているように使用できます。

注記

`WITH_LOCK_ORDER` オプションを有効にすると、MySQL ビルドに `flex` プログラムが必要になります。

このオプションは MySQL 8.0.17 で追加されました。

- `-DWITH_LSAN=bool`

AddressSanitizer を使用せずに LeakSanitizer を実行するかどうか。デフォルトは `OFF` です。

このオプションは MySQL 8.0.16 で追加されました。

- `-DWITH_LTO=bool`

コンパイラでサポートされている場合に、リンク時最適化を有効にするかどうか。 `FPROFILE_USE` が有効でないかぎり、デフォルトは `OFF` です。

このオプションは MySQL 8.0.13 で追加されました。

- `-DWITH_LZ4=lz4_type`

`WITH_LZ4` は、`zlib` サポートのソースを示します:

- `bundled`: ディストリビューションにバンドルされている `lz4` ライブラリを使用します。これはデフォルトです。
- `system`: システム `lz4` ライブラリを使用します。 `WITH_LZ4` がこの値に設定されている場合、 `lz4_decompress` ユーティリティはビルドされません。この場合、かわりに `system lz4` コマンドを使用できます。

- `-DWITH_LZMA=lzma_type`

含める LZMA ライブラリサポートのタイプ。 `lzma_type` は、次のいずれかの値です:

- `bundled`: ディストリビューションにバンドルされている LZMA ライブラリを使用します。これはデフォルトです。
- `system`: システム LZMA ライブラリを使用します。

このオプションは MySQL 8.0.16 で削除されました。

- `-DWITH_MECAB={disabled|system|path_name}`

このオプションを使用して、MeCab パーサーをコンパイルします。MeCab をデフォルトのインストールディレクトリにインストールした場合は、 `-DWITH_MECAB=system` を設定します。 `system` オプションは、ネイティブパッケージ管理ユーティリティを使用してソースまたはバイナリから実行される MeCab インストールに適用されます。MeCab をカスタムインストールディレクトリにインストールした場合は、MeCab インストールへのパスを指定します。たとえば、 `-DWITH_MECAB=/opt/mecab` です。 `system` オプションが機能しない場合は、すべての場合に MeCab インストールパスを指定する必要があります。

関連情報については、[セクション 12.10.9 「MeCab フルテキストパーサープラグイン」](#) を参照してください。

- `-DWITH_MSAN=bool`

MemorySanitizer をサポートするコンパイラに対して有効にするかどうか。デフォルトはオフです。

このオプションを有効にするには、MySQL にリンクされているすべてのライブラリもオプションを有効にしてコンパイルされている必要があります。

- `-DWITH_MSVCRT_DEBUG=bool`

Visual Studio CRT メモリーリークトレースを有効にするかどうか。デフォルトは `OFF` です。

- `-DWITH_MYSQLX=bool`

X プラグイン のサポート付きでビルドするかどうか。デフォルトの `ON`。第20章「ドキュメントストアとしての MySQL の使用」を参照してください。

- `-DWITH_NUMA=bool`

NUMA メモリー割当てポリシーを明示的に設定します。CMake では、現在のプラットフォームに NUMA サポートがあるかどうかに基づいて、デフォルトの `WITH_NUMA` 値が設定されます。NUMA がサポートされていないプラットフォームでは、CMake は次のように動作します:

- NUMA オプションなし (通常の場合) では、CMake は正常に続行され、この警告のみが生成されます: NUMA ライブラリがないか、必要なバージョンがありません
- `-DWITH_NUMA=ON` では、CMake はこのエラーで異常終了: NUMA ライブラリがないか、必要なバージョンがありません

- `-DWITH_PROTOBUF=protobuf_type`

使用するプロトコルバッファパッケージ。protobuf_type は、次のいずれかの値です:

- `bundled`: 配布にバンドルされているパッケージを使用します。これはデフォルトです。オプションで、`INSTALL_PRIV_LIBDIR` を使用して動的 Protobuf ライブラリディレクトリを変更します。
- `system`: システムにインストールされているパッケージを使用します。

その他の値は無視され、`bundled` にフォールバックされます。

- `-DWITH_RAPID=bool`

迅速な開発サイクルプラグインを構築するかどうか。有効にすると、これらのプラグインを含むビルドツリーに `rapid` ディレクトリが作成されます。無効にすると、ビルドツリーに `rapid` ディレクトリは作成されません。rapid ディレクトリがソースツリーから削除されないかぎり、デフォルトは `ON` です。この場合、デフォルトは `OFF` になります。

- `-DWITH_RAPIDJSON=rapidjson_type`

含める RapidJSON ライブラリサポートのタイプ。rapidjson_type は、次のいずれかの値です:

- `bundled`: ディストリビューションにバンドルされている RapidJSON ライブラリを使用します。これはデフォルトです。
- `system`: システム RapidJSON ライブラリを使用します。バージョン 1.1.0 以上が必要です。

このオプションは MySQL 8.0.13 で追加されました。

- `-DWITH_RE2=re2_type`

含める RE2 ライブラリサポートのタイプ。re2_type は、次のいずれかの値です:

- `bundled`: ディストリビューションにバンドルされている RE2 ライブラリを使用します。これはデフォルトです。
- `system`: システム RE2 ライブラリを使用します。

MySQL 8.0.18 の時点で、MySQL は RE2 ライブラリを使用しなくなり、このオプションは削除されました。

- `-DWITH_ROUTER=bool`

MySQL Router をビルドするかどうか。デフォルトは `ON` です。

このオプションは MySQL 8.0.16 で追加されました。

- `-DWITH_SSL={ssl_type|path_name}`

暗号化された接続、乱数生成のエントロピ、およびその他の暗号化関連の操作をサポートするには、SSL ライブラリを使用して MySQL を構築する必要があります。このオプションは、使用する SSL ライブラリを指定します。

- `ssl_type` には、次の値のいずれかを指定できます。

- `system`: システム OpenSSL ライブラリを使用します。これはデフォルトです。

macOS および Windows では、`system` を使用して、`path_name` を使用して CMake が起動されたときに手動でインストールされた OpenSSL ライブラリを指しているかのように構築するように MySQL を構成します。これは、システム SSL ライブラリがないためです。macOS では、`system` で検出できるように、`brew install openssl` は `/usr/local/opt/openssl` にインストールされます。Windows では、`%ProgramFiles%/OpenSSL`、`%ProgramFiles%/OpenSSL-Win32`、`%ProgramFiles%/OpenSSL-Win64`、`C:/OpenSSL`、`C:/OpenSSL-Win32` および `C:/OpenSSL-Win64` がチェックされます。

- `yes`: これは `system` のシノニムです。
- `path_name` は、使用する OpenSSL インストールのパス名です。これは、CMake がシステムにインストールされている古いバージョンまたは正しくない OpenSSL バージョンを検出して使用するのを防ぐことができるため、`system` の `ssl_type` 値を使用するよりも望ましい場合があります。(同じことを可能にする別の方法は、`WITH_SSL` を `system` に設定し、`CMAKE_PREFIX_PATH` オプションを `path_name` に設定することです。)

SSL ライブラリの構成の詳細は、を参照してください [セクション2.9.6「SSL ライブラリサポートの構成」](#)。

- `-DWITH_SYSTEMD=bool`

systemd サポートファイルのインストールを有効にするかどうか。デフォルトでは、このオプションは無効です。有効にすると、systemd サポートファイルがインストールされ、`mysqld_safe` や System V 初期化スクリプトなどのスクリプトはインストールされません。systemd を使用できないプラットフォームでは、`WITH_SYSTEMD` を有効にすると、CMake からエラーが発生します。

systemd の使用の詳細は、[セクション2.5.9「systemd を使用した MySQL Server の管理」](#) を参照してください。このセクションでは、`[mysqld_safe]` オプショングループで以前に指定されたオプションの指定についても説明します。systemd の使用時に `mysqld_safe` はインストールされないため、このようなオプションは別の方法で指定する必要があります。

- `-DWITH_SYSTEM_LIBS=bool`

このオプションは、明示的に設定されていない次の CMake オプションのいずれかの `system` 値を設定する「アンブレラ」オプションとして機能: `WITH_CURL`、`WITH_EDITLINE`、`WITH_ICU`、`WITH_LIBEVENT`、`WITH_LZ4`、`WITH_LZMA`、`WITH_PROTOBUF`、`WITH_RE2`、`WITH_SSL`、`WITH_ZLIB`、`WITH_ZSTD`。

- `-DWITH_SYSTEMD_DEBUG=bool`

systemd を使用して MySQL を実行するプラットフォームについて、追加の systemd デバッグ情報を生成するかどうか。デフォルトは `OFF` です。

このオプションは MySQL 8.0.22 で追加されました。

- `-DWITH_TCMALLOC=bool`

`-ltcmalloc` とリンクするかどうか。有効にすると、組み込み `malloc()`、`calloc()`、`realloc()` および `free()` ルーチンは無効になります。デフォルトは `OFF` です。

`WITH_TCMALLOC` と `WITH_JEMALLOC` は相互に排他的です。

このオプションは MySQL 8.0.22 で追加されました。

- `-DWITH_TEST_TRACE_PLUGIN=bool`

テストプロトコルトレースクライアントプラグインをビルドするかどうか ([Using the Test Protocol Trace Plugin](#) を参照)。デフォルトでは、このオプションは無効です。`WITH_CLIENT_PROTOCOL_TRACING` オプションが有効になっていないかぎり、このオプションを有効にしても効果はありません。MySQL が両方のオプションを有効にして構成されている場合、`libmysqlclient` クライアントライブラリはテストプロトコルトレースプラグインが組み込まれた状態で構築され、すべての標準 MySQL クライアントがプラグインをロードします。ただし、テストプラグインが有効になっていても、デフォルトでは何の効果もありません。プラグインの制御は、環境変数を使用して行います。[Using the Test Protocol Trace Plugin](#) を参照してください。

注記

独自のプロトコルトレースプラグインを使用する場合は、`WITH_TEST_TRACE_PLUGIN` オプションを有効にしないでください。このようなプラグインは一度に 1 つのみロードでき、別のプラグインをロードしようとするエラーが発生するためです。テストプロトコルトレースプラグインを有効にして MySQL をすでに構築し、その動作を確認している場合は、独自のプラグインを使用する前に MySQL を再構築する必要があります。

トレースプラグインの書き込みの詳細は、[Writing Protocol Trace Plugins](#) を参照してください。

- `-DWITH_TSAN=bool`

ThreadSanitizer をサポートするコンパイラに対して有効にするかどうか。デフォルトはオフです。

- `-DWITH_UBSAN=bool`

未定義の動作サニタイザをサポートするコンパイラに対して、サニタイザを有効にするかどうか。デフォルトはオフです。

- `-DWITH_UNIT_TESTS={ON|OFF}`

有効な場合は、ユニットテストを使用して MySQL をコンパイルします。サーバーがコンパイルされていない場合を除き、デフォルトは `ON` です。

- `-DWITH_UNIXODBC=1`

Connector/ODBC に関して、unixODBC サポートを有効にします。

- `-DWITH_VALGRIND=bool`

Valgrind ヘッダーファイルをコンパイルするかどうか。これにより、Valgrind API が MySQL コードに公開されます。デフォルトは `OFF` です。

Valgrind 対応のデバッグビルドを生成するには、`-DWITH_VALGRIND=1` は通常 `-DWITH_DEBUG=1` と組み合わせられます。[Building Debug Configurations](#) を参照してください。

- `-DWITH_ZLIB=zlib_type`

一部の機能では、サーバーが `COMPRESS()` および `UNCOMPRESS()` 関数などの圧縮ライブラリサポートでビルドされていること、およびクライアント/サーバープロトコルの圧縮を必要とします。 `WITH_ZLIB` は `zlib` サポートのソースを示します。

- `bundled`: 配布にバンドルされた `zlib` ライブラリを使用します。これはデフォルトです。

- `system`: システムの `zlib` ライブラリを使用します。 `WITH_ZLIB` がこの値に設定されている場合、 `zlib_decompress` ユーティリティはビルドされません。この場合、かわりに `system openssl zlib` コマンドを使用できます。

- `-DWITH_ZSTD=zstd_type`

`zstd` アルゴリズムを使用した接続圧縮 (セクション4.2.8「接続圧縮制御」を参照) では、 `zstd` ライブラリのサポートを使用してサーバーを構築する必要があります。 `WITH_ZSTD` は、 `zstd` サポートのソースを示します:

- `bundled`: ディストリビューションにバンドルされている `zstd` ライブラリを使用します。これはデフォルトです。

- `system`: システム `zstd` ライブラリを使用します。

このオプションは MySQL 8.0.18 で追加されました。

コンパイラフラグ

- `-DCMAKE_C_FLAGS="flags"`

C コンパイラのフラグ。

- `-DCMAKE_CXX_FLAGS="flags"`

C++ コンパイラのフラグ。

- `-DWITH_DEFAULT_COMPILER_OPTIONS=bool`

`cmake/build_configurations/compiler_options.cmake` からのフラグを使用するかどうか。

注記

すべての最適化フラグは、MySQL ビルドチームによって慎重に選択およびテストされています。オーバーライドすると予期しない結果になることがあります。自己責任において行なってください。

独自の C および C++ コンパイラフラグを指定するには、最適化に影響しないフラグの場合は `CMAKE_C_FLAGS` および `CMAKE_CXX_FLAGS` CMake オプションを使用します。

独自のコンパイラフラグを提供する場合、 `CMAKE_BUILD_TYPE` も指定するとよいでしょう。

たとえば、32 ビットリリースビルドを 64 ビット Linux マシンに作成するには次のようにします。

```
mkdir bld
cd bld
cmake .. -DCMAKE_C_FLAGS=-m32 \
  -DCMAKE_CXX_FLAGS=-m32 \
  -DCMAKE_BUILD_TYPE=RelWithDebInfo
```

最適化に影響するフラグ (`-Onumber`) をセットする場合は、 `CMAKE_C_FLAGS_build_type` および/または `CMAKE_CXX_FLAGS_build_type` オプションをセットする必要があります。ここで、 `build_type` は `CMAKE_BUILD_TYPE` 値に対応します。デフォルトのビルドタイプ (`RelWithDebInfo`) に異なる最適化を指定するには、 `CMAKE_C_FLAGS_RELWITHDEBINFO` および `CMAKE_CXX_FLAGS_RELWITHDEBINFO` オプションをセットします。たとえば、Linux で `-O3` とデバッグシンボルを使用してコンパイルするには、次のようにします。

```
cmake .. -DCMAKE_C_FLAGS_RELWITHDEBINFO="-O3 -g" \
  -DCMAKE_CXX_FLAGS_RELWITHDEBINFO="-O3 -g"
```

NDB Cluster をコンパイルするための CMake オプション

NDB Cluster サポートを使用して MySQL 8.0 ソースを構築する場合は、次のオプションを使用します。

- `-DMEMCACHED_HOME=dir_name`

`dir_name` によって示されるシステムディレクトリにインストールされている memcached (バージョン 1.6 以降) を使用してビルドを実行します。ビルドに使用されるこのインストールからのファイルは、memcached バイナリ、ヘッダーファイル、およびライブラリに加えて、`memcached_utilities` ライブラリおよびヘッダーファイル `engine_testapp.h` を含みます。

`ndbmemcache` を、バンドルの memcached ソースを使用してビルドする場合 (`WITH_BUNDLED_MEMCACHED` オプション)、このオプションはセットしないでください。すなわち、デフォルトでバンドルのソースが使用されます。

SASL 承認および `dtrace` サポートを提供するための追加の CMake オプションは外部ソースから `memcached` をコンパイルするときに使用できますが、NDB Cluster にバンドルされている `memcached` ソースではこれらのオプションは現在有効になっていません。

- `-NDB_UTILS_LINK_DYNAMIC={ON|OFF}`

`ndb_drop_table` などの NDB ユーティリティーを `ndbclient` と静的にリンクするか (`OFF`)、動的にリンクするか (`ON`) を制御します。`OFF` (静的リンク) がデフォルトです。通常、静的リンクは、`LD_LIBRARY_PATH` の問題を回避するため、または複数のバージョンの `ndbclient` がインストールされている場合に、これらを構築する場所で使用されます。このオプションは、Docker イメージを作成することを目的としており、ターゲット環境が正確に制御される可能性があり、イメージサイズを小さくすることが望ましい場合もあります。

NDB 8.0.22 に追加されました。

- `-DWITH_BUNDLED_LIBEVENT={ON|OFF}`

`ndbmemcached` サポートを使用して NDB Cluster を構築する場合は、NDB Cluster ソースに含まれている `libevent` を使用します。デフォルトで有効。`OFF` では、かわりにシステム `libevent` が使用されます。

- `-DWITH_BUNDLED_MEMCACHED={ON|OFF}`

NDB Cluster ソースツリーに含まれる memcached ソースを構築し、`ndbmemcache` エンジンの構築時に結果の memcached サーバーを使用します。この場合、`make install` は `memcached` バイナリをインストール `bin` ディレクトリに配置し、`ndbmemcache` エンジン共有ライブラリファイル `ndb_engine.so` をインストール `lib` ディレクトリに配置します。

このオプションはデフォルトで ON です。

- `-DWITH_CLASSPATH=path`

NDB Cluster Connector for Java を構築するためのクラスパスを設定します。デフォルトは空です。`-DWITH_NDB_JAVA=OFF` を使用する場合、このオプションは無視されます。

- `-DWITH_ERROR_INSERT={ON|OFF}`

NDB カーネルのエラーインジェクションを有効化します。テスト専用です。本番環境のバイナリのビルドに使用することは意図していません。デフォルトは `OFF` です。

- `-DWITH_NDBCLUSTER_STORAGE_ENGINE={ON|OFF}`

これは `WITH_NDBCLUSTER` のエイリアスです。

- `-DWITH_NDBCLUSTER={ON|OFF}`

`mysqld` で NDB (NDBCLUSTER) ストレージエンジンのサポートのビルドおよびリンク。デフォルトは `ON` です。

- `-DWITH_NDBMTD={ON|OFF}`

マルチスレッドデータノード実行可能ファイル `ndbmtd` を構築します。デフォルトは `ON` です。

- `-DWITH_NDB_BINLOG={ON|OFF}`

このオプションを使用して、`mysqld` ビルド内でデフォルトでバイナリロギングを有効にします。デフォルトで ON です。

- `-DWITH_NDB_DEBUG={ON|OFF}`

NDB Cluster バイナリのデバッグバージョンの構築を有効にします。デフォルトで OFF です。

- `-DWITH_NDB_JAVA={ON|OFF}`

ClusterJ を含む Java サポートを使用した NDB Cluster の構築を有効にします。

このオプションはデフォルトで ON です。NDB Cluster を Java サポート付きでコンパイルしない場合は、CMake の実行時に `-DWITH_NDB_JAVA=OFF` を指定して明示的に無効にする必要があります。そうしないと、Java が検出できない場合にビルドの構成が失敗します。

- `-DWITH_NDB_PORT=port`

この `port` をデフォルトで使用するよう構築された NDB Cluster 管理サーバー (`ndb_mgmd`) を生成します。このオプションがセットされていない場合、結果の管理サーバーはデフォルトでポート 1186 を使用しようとします。

- `-DWITH_NDB_TEST={ON|OFF}`

有効の場合、NDB API テストプログラムをインクルードします。デフォルトは OFF です。

- `-DWITH_PLUGIN_NDBCLUSTER={ON|OFF}`

`WITH_NDBCLUSTER` のエイリアス。

2.9.8 MySQL のコンパイルに関する問題

多くの問題の解決方法には再構成が含まれます。再構成を行う場合は、次に注意してください。

- CMake を以前に実行したあとで実行すると、以前の起動時に収集した情報を使用する場合があります。この情報は `CMakeCache.txt` に格納されています。CMake が起動すると、情報がまだ正しいことを前提として、そのファイルが検索され、その内容 (存在する場合) が読み取られます。この仮定は再構成した場合には無効です。
- CMake を実行するたびに、`make` を再実行して再コンパイルする必要があります。しかし、前のビルドの古いオブジェクトファイルが異なる構成オプションでコンパイルされている場合、それらを最初に削除する場合もあります。

古いオブジェクトファイルまたは構成情報が使用されることを予防するために、CMake を再実行する前に次のコマンドを実行します。

Unix の場合:

```
shell> make clean
shell> rm CMakeCache.txt
```

Windows の場合:

```
shell> devenv MySQL.sln /clean
shell> del CMakeCache.txt
```

ソースツリー外でビルドする場合、CMake を再実行する前にビルドディレクトリを削除して再作成します。ソースツリー外でのビルドに関する説明は、[Build MySQL Server を CMake でビルドする方法](#)を参照してください。

一部のシステムでは、システムインクルードファイルの違いにより警告が生じる場合があります。次のリストは、MySQL のコンパイル時にもっともよく生じることがよくわかっているその他の問題を記述しています。

- どの C および C++ コンパイラを使用するかを定義するために、`CC` および `CXX` 環境変数を使用できます。例:

```
shell> CC=gcc
shell> CXX=g++
shell> export CC CXX
```


独自の C および C++ コンパイラフラグを指定するには、`CMAKE_C_FLAGS` および `CMAKE_CXX_FLAGS` CMake オプションを使用します。 [コンパイラフラグ](#) を参照してください。

指定する必要がある可能性のあるフラグを確認するには、`mysql_config` を、`-cflags` および `--cxxflags` オプションを使用して起動します。

- コンパイル段階でどのコマンドが実行されるかを確認するには、CMake を使用して MySQL を構成したあと、単なる `make` ではなく `make VERBOSE=1` を実行します。
- コンパイルに失敗する場合は、`MYSQL_MAINTAINER_MODE` オプションが有効かどうかをチェックします。このモードでは、コンパイラの警告はエラーになるため、無効にすることによりコンパイルを続行できる場合があります。
- コンパイルが次のいずれかのエラーで失敗した場合、`make` のバージョンを GNU `make` にアップグレードする必要があります。

```
make: Fatal error in reader: Makefile, line 18:  
Badly formed macro assignment
```

または:

```
make: file `Makefile' line 18: Must be a separator (:
```

または:

```
pthread.h: No such file or directory
```

Solaris および FreeBSD は、`make` プログラムに問題が多いことが知られています。

GNU `make` 3.75 は動作が確認されています。

- `sql_yacc.cc` ファイルは `sql_yacc.yy` から生成されます。通常、MySQL には事前生成された `sql_yacc.cc` が付属しているため、ビルドプロセスでコピーを作成する必要はありません。しかし、それを再度作成する必要がある場合、次のエラーに遭遇する場合があります。

```
"sql_yacc.yy", line xxx fatal: default action causes potential...
```

これは `yacc` のバージョンに問題があることを意味しています。おそらく、代わりに `bison` (the GNU version of `yacc`) の最近のバージョンをインストールして使用する必要があります。

`bison` 1.75 以前のバージョンでは次のエラーがレポートされる場合があります。

```
sql_yacc.yy:#####: fatal error: maximum table size (32767) exceeded
```

テーブルの最大サイズを実際には超えていなくても、`bison` の旧バージョンのバグでエラーが発生します。

ツールの取得または更新の詳細は、[セクション2.9「ソースから MySQL をインストールする」](#) のシステム要件を参照してください。

2.9.9 MySQL の構成とサードパーティーツール

MySQL のソースから MySQL のバージョンを判断する必要があるサードパーティーツールは、トップレベルソースディレクトリの `VERSION` ファイルを読み取ることができます。このファイルには、バージョンの各部分が個別にリストされています。たとえば、バージョンが MySQL 8.0.4-rc の場合、ファイルは次のようになります:

```
MYSQL_VERSION_MAJOR=8  
MYSQL_VERSION_MINOR=0  
MYSQL_VERSION_PATCH=4  
MYSQL_VERSION_EXTRA=-rc
```

ソースが General Availability (GA) リリース用でない場合、`MYSQL_VERSION_EXTRA` 値は空ではありません。前述の例では、値は「候補者のリリース」に対応しています。

バージョンのコンポーネントから 5 桁の数字を構成するには、次の式を使用します。

```
MYSQL_VERSION_MAJOR*10000 + MYSQL_VERSION_MINOR*100 + MYSQL_VERSION_PATCH
```

2.9.10 MySQL Doxygen ドキュメントコンテンツの生成

MySQL ソースコードには、Doxygen を使用して記述された内部ドキュメントが含まれます。生成された Doxygen コンテンツは、<https://dev.mysql.com/doc/index-other.html> で入手できます。次の手順を使用して、このコンテンツを MySQL ソース配布からローカルに生成することもできます:

1. **doxygen** 1.8.11 以上をインストールします。配布は、<http://www.doxygen.nl/> で入手できます。

doxygen のインストール後、バージョン番号を確認します:

```
shell> doxygen --version
1.8.13
```

2. **PlantUML** をインストールします。

PlantUML を Windows (Windows 10 でテスト済) にインストールする場合、レジストリキーを作成するには、少なくとも管理者として実行する必要があります。管理者コンソールを開き、次のコマンドを実行します:

```
shell> java -jar path-to-plantuml.jar
```

このコマンドは GUI ウィンドウを開き、コンソールでエラーを返しません。

3. **PLANTUML_JAR_PATH** 環境を、PlantUML をインストールした場所に設定します。例:

```
shell> export PLANTUML_JAR_PATH=path-to-plantuml.jar
```

4. **Graphviz dot** コマンドをインストールします。

Graphviz のインストール後、**dot** の可用性を確認します。例:

```
shell> which dot
/usr/bin/dot

shell> dot -V
dot - graphviz version 2.28.0 (20130928.0220)
```

5. 場所を MySQL ソース配布の最上位ディレクトリに変更し、次の手順を実行します:

まず、**cmake** を実行します:

```
shell> cd your-mysql-source-directory
shell> mkdir bld
shell> cd bld
shell> cmake ..
```

次に、**doxygen** ドキュメントを生成します:

```
shell> make doxygen
```

エラーログを調べます。これは、最上位ディレクトリの **doxyerror.log** ファイルで使用できます。ビルドが正常に実行されたと仮定して、ブラウザを使用して生成された出力を表示します。例:

```
shell> firefox doxygen/html/index.html
```

2.10 インストール後のセットアップとテスト

このセクションでは、MySQL のインストール後に実行する必要があるタスクについて説明します:

- 必要に応じて、データディレクトリを初期化し、MySQL 付与テーブルを作成します。一部の MySQL インストール方法では、データディレクトリの初期化が自動的に実行される場合があります:
- MySQL Installer によって実行される Windows インストール操作。
- Oracle のサーバー RPM または Debian ディストリビューションを使用した Linux へのインストール。

- Debian Linux、Ubuntu Linux、Gentoo Linux など、多くのプラットフォームでネイティブパッケージングシステムを使用したインストール。
- DMG ディストリビューションを使用した macOS へのインストール。

他のプラットフォームおよびインストールタイプの場合は、データディレクトリを手動で初期化する必要があります。これには、Unix および Unix に似たシステムでの汎用バイナリおよびソースディストリビューションからのインストール、および Windows での ZIP アーカイブパッケージからのインストールが含まれます。その手順は、[セクション2.10.1「データディレクトリの初期化」](#)を参照してください。

- サーバーを起動し、アクセスできることを確認します。手順については、[セクション2.10.2「サーバーの起動」](#)および [セクション2.10.3「サーバーのテスト」](#)を参照してください。
- データディレクトリの初期化中にまだ実行されていない場合は、付与テーブルの初期 root アカウントにパスワードを割り当てます。パスワードにより、MySQL サーバーへの不正アクセスが防止されます。その手順は、[セクション2.10.4「初期 MySQL アカウントの保護」](#)を参照してください。
- オプションで、システムの起動および停止時にサーバーが自動的に起動および停止するように配置します。その手順は、[セクション2.10.5「MySQL を自動的に起動および停止する」](#)を参照してください。
- オプションで、タイムゾーンテーブルに移入して、名前付きタイムゾーンの認識を有効にします。その手順は、[セクション5.1.15「MySQL Server でのタイムゾーンのサポート」](#)を参照してください。

追加のユーザーアカウントを作成する準備ができれば、[セクション6.2「アクセス制御とアカウント管理」](#)で MySQL のアクセス制御システムおよびアカウント管理に関する情報を確認できます。

2.10.1 データディレクトリの初期化

MySQL のインストール後、`mysql` システムスキーマのテーブルを含め、データディレクトリを初期化する必要があります:

- 一部の MySQL インストール方法では、[セクション2.10「インストール後のセットアップとテスト」](#)で説明されているように、データディレクトリの初期化は自動的に行われます。
- その他のインストール方法では、データディレクトリを手動で初期化する必要があります。これには、Unix および Unix に似たシステムでの汎用バイナリおよびソースディストリビューションからのインストール、および Windows での ZIP アーカイブパッケージからのインストールが含まれます。

このセクションでは、データディレクトリの初期化が自動ではない MySQL のインストール方法で、データディレクトリを手動で初期化する方法について説明します。サーバーがアクセス可能で適切に動作しているかどうかをテストできるようにする推奨コマンドについては、[セクション2.10.3「サーバーのテスト」](#)を参照してください。

注記

MySQL 8.0 では、デフォルトの認証プラグインが `mysql_native_password` から `caching_sha2_password` に変更され、`'root'@'localhost'` 管理アカウントはデフォルトで `caching_sha2_password` を使用します。root アカウントで以前のデフォルトの認証プラグイン (`mysql_native_password`) を使用する場合は、[caching_sha2_password および root 管理アカウント](#)を参照してください。

- [データディレクトリの初期化の概要](#)
- [データディレクトリの初期化手順](#)
- [データディレクトリの初期化中のサーバーアクション](#)
- [初期化後の root パスワードの割当て](#)

データディレクトリの初期化の概要

ここに示す例では、サーバーは `mysql` ログインアカウントのユーザー ID で実行することを目的としています。アカウントが存在しない場合は作成するか ([mysql ユーザーおよびグループの作成](#)を参照)、サーバーの実行に使用する予定の別の既存のログインアカウントの名前を置き換えます。

1. 場所を MySQL インストールの最上位ディレクトリ (通常は `/usr/local/mysql`) に変更します (必要に応じてシステムのパス名を調整します):

```
cd /usr/local/mysql
```

このディレクトリ内には、サーバーを含む `bin` サブディレクトリ、クライアントプログラムおよびユーティリティプログラムなど、複数のファイルおよびサブディレクトリがあります。

2. `secure_file_priv` システム変数は、インポートおよびエクスポート操作を特定のディレクトリに制限します。その変数の値として場所を指定できるディレクトリを作成します:

```
mkdir mysql-files
```

ディレクトリユーザーおよびグループの所有権を `mysql` ユーザーおよび `mysql` グループに付与し、ディレクトリ権限を適切に設定します:

```
chown mysql:mysql mysql-files  
chmod 750 mysql-files
```

3. サーバーを使用してデータディレクトリを初期化します。これには、ユーザーがサーバーへの接続を許可される方法を決定する初期 MySQL 付与テーブルを含む `mysql` スキーマが含まれます。例:

```
bin/mysqld --initialize --user=mysql
```

コマンドに関する重要な情報、特に使用するコマンドオプションについては、[データディレクトリの初期化手順](#)を参照してください。サーバーが初期化を実行する方法の詳細は、[データディレクトリの初期化中のサーバーアクション](#)を参照してください。

通常、データディレクトリの初期化は、最初に MySQL をインストールした後にのみ行う必要があります。(既存のインストールへのアップグレードの場合は、かわりにアップグレード手順を実行します。[セクション 2.11 「MySQL のアップグレード」](#)を参照してください。)ただし、データディレクトリを初期化するコマンドは既存の `mysql` スキーマテーブルを上書きしないため、どのような状況でも安全に実行できます。

4. セキュアな接続の自動サポートを使用してサーバーをデプロイする場合は、`mysql_ssl_rsa_setup` ユーティリティを使用してデフォルトの SSL および RSA ファイルを作成します:

```
bin/mysql_ssl_rsa_setup
```

詳細は、[セクション 4.4.3 「mysql_ssl_rsa_setup — SSL/RSA ファイルの作成」](#)を参照してください。

5. オプションファイルがない場合、サーバーはデフォルト設定で起動します。([セクション 5.1.2 「サーバー構成のデフォルト値」](#)を参照してください。) MySQL サーバーが起動時に使用するオプションを明示的に指定するには、`/etc/my.cnf` や `/etc/mysql/my.cnf` などのオプションファイルに配置します。([セクション 4.2.2.2 「オプションファイルの使用」](#)を参照してください。) たとえば、オプションファイルを使用して `secure_file_priv` システム変数を設定できます。
6. システムブート時に手動で介入せずに MySQL が起動するように配置するには、[セクション 2.10.5 「MySQL を自動的に起動および停止する」](#)を参照してください。
7. データディレクトリの初期化では、`mysql` スキーマにタイムゾーンテーブルが作成されますが、タイムゾーンテーブルは移入されません。そのためには、[セクション 5.1.15 「MySQL Server でのタイムゾーンのサポート」](#)を参照してください。

データディレクトリの初期化手順

場所を MySQL インストールの最上位ディレクトリ (通常は `/usr/local/mysql`) に変更します (必要に応じてシステムのパス名を調整します):

```
cd /usr/local/mysql
```

データディレクトリを初期化するには、サーバーで `'root'@'localhost'` アカウントのランダムな初期パスワードを生成するか、パスワードなしでそのアカウントを作成するかに応じて、`--initialize` または `--initialize-insecure` オプションを指定して `mysqld` を起動します:

- 「デフォルトで保護」インストールに `--initialize` を使用します (ランダムな初期 `root` パスワードの生成を含む)。この場合、パスワードは期限切れとしてマークされるため、新しいパスワードを選択する必要があります。

- `--initialize-insecure` では、`root` パスワードは生成されません。これはセキュアではありません。サーバーを本番で使用する前に、適切なタイミングでアカウントにパスワードを割り当てることを想定しています。

新しい'`root`'@'`localhost`'パスワードを割り当てる手順については、[初期化後の root パスワードの割当て](#) を参照してください。

注記

サーバーは、すべてのメッセージ (初期パスワードを含む) を標準エラー出力に書き込みます。これはエラーログにリダイレクトされる場合があるため、画面にメッセージが表示されていないかどうかを確認してください。エラーログの場所などの詳細は、[セクション 5.4.2 「エラーログ」](#) を参照してください。

Windows では、`--console` オプションを使用してコンソールにメッセージを送信します。

Unix および Unix に似たシステムでは、後で実行するときにサーバーが読取りおよび書き込みアクセス権を持つように、データベースディレクトリおよびファイルを `mysql` ログインアカウントが所有することが重要です。これを確認するには、システム `root` アカウントから `mysqld` を起動し、次に示すように `--user` オプションを含めます:

```
bin/mysqld --initialize --user=mysql
bin/mysqld --initialize-insecure --user=mysql
```

または、`mysql` としてログインしているときに `mysqld` を実行します。この場合、コマンドから `--user` オプションを省略できます。

Windows の場合は、次のいずれかのコマンドを使用します:

```
bin\mysqld --initialize --console
bin\mysqld --initialize-insecure --console
```

注記

必要なシステムライブラリがない場合、データディレクトリの初期化が失敗することがあります。たとえば、次のようなエラーが表示される場合があります:

```
bin/mysqld: error while loading shared libraries:
libnuma.so.1: cannot open shared object file:
No such file or directory
```

この場合は、欠落しているライブラリを手動でインストールするか、システムパッケージマネージャを使用してインストールする必要があります。次に、データディレクトリの初期化コマンドを再試行します。

`mysqld` がインストールディレクトリまたはデータディレクトリの正しい場所を特定できない場合は、`--basedir` や `--datadir` などの他のオプションを指定する必要がある場合があります。次に例を示します (単一行にコマンドを入力します):

```
bin/mysqld --initialize --user=mysql
--basedir=/opt/mysql/mysql
--datadir=/opt/mysql/mysql/data
```

または、関連するオプション設定をオプションファイルに配置し、そのファイルの名前を `mysqld` に渡します。Unix および Unix に似たシステムでは、オプションファイル名が `/opt/mysql/mysql/etc/my.cnf` であるとし、ファイルに次の行を挿入します:

```
[mysqld]
basedir=/opt/mysql/mysql
datadir=/opt/mysql/mysql/data
```

次に、次のように `mysqld` を起動します (`--defaults-file` オプションを最初に指定して単一行にコマンドを入力します):

```
bin/mysqld --defaults-file=/opt/mysql/mysql/etc/my.cnf
--initialize --user=mysql
```

Windows では、`C:\my.ini` に次の行が含まれているとします:

```
[mysqld]
basedir=C:\Program Files\MySQL\MySQL Server 8.0
```

```
datadir=D:\MySQLdata
```

次に、次のように `mysqld` を起動します (`--defaults-file` オプションを最初に指定して単一行にコマンドを入力します):

```
bin\mysqld --defaults-file=C:\my.ini  
--initialize --console
```

データディレクトリの初期化中のサーバーアクション

注記

サーバーによって実行されるデータディレクトリの初期化シーケンスは、`mysql_secure_installation` および `mysql_ssl_rsa_setup` によって実行されるアクションの変わりにはなりません。 [セクション4.4.2「mysql_secure_installation — MySQL インストールのセキュリティー改善](#) および [セクション4.4.3「mysql_ssl_rsa_setup — SSL/RSA ファイルの作成](#) を参照してください。

`--initialize` または `--initialize-insecure` オプションを指定して起動すると、`mysqld` はデータディレクトリの初期化シーケンス中に次のアクションを実行します:

1. サーバーは、次のようにデータディレクトリの存在をチェックします:

- データディレクトリが存在しない場合は、サーバーによって作成されます。
- データディレクトリは存在するが空でない (つまり、ファイルまたはサブディレクトリが含まれている) 場合、サーバーはエラーメッセージを生成した後終了します:

```
[ERROR] --initialize specified but the data directory exists. Aborting.
```

この場合、データディレクトリを削除するか名前を変更して、再試行してください。

すべてのエントリにピリオド (.) で始まる名前がある場合、既存のデータディレクトリは空にできません。

2. サーバーは、データディレクトリ内に、データディクショナリテーブル、付与テーブル、タイムゾーンテーブル、サーバー側ヘルプテーブルなどの `mysql` システムスキーマとそのテーブルを作成します。 [セクション5.3「mysql システムスキーマ](#) を参照してください。
3. サーバーは、`InnoDB` テーブルの管理に必要な `system tablespace` および関連データ構造を初期化します。

注記

`mysqld` で `InnoDB system tablespace` を設定した後、テーブルスペースの特性を変更するには、新しい `instance` 全体を設定する必要があります。修飾変更には、システムテーブルスペースの最初のファイルのファイル名と `undo` ログの数が含まれます。デフォルト値を使用しない場合は、`mysqld` を実行する前に、`innodb_data_file_path` および `innodb_log_file_size` 構成パラメータの設定が `MySQL configuration file` に配置されていることを確認してください。また、`innodb_data_home_dir` および `innodb_log_group_home_dir` などの、`InnoDB` ファイルの作成および場所に影響するその他のパラメータを、必要に応じて指定してください。

これらのオプションが構成ファイルにあり、そのファイルが `MySQL` がデフォルトで読み取る場所がない場合は、`mysqld` の実行時に `--defaults-extra-file` オプションを使用してファイルの場所を指定します。

4. サーバーは、`'root'@'localhost'` スーパーユーザーアカウントおよびその他の予約済みアカウントを作成します ([セクション6.2.9「予約済みアカウント](#) を参照)。一部の予約済みアカウントはロックされており、クライアントでは使用できませんが、`'root'@'localhost'` は管理用であるため、パスワードを割り当てる必要があります。

`'root'@'localhost'` アカウントのパスワードに関するサーバーアクションは、その呼出し方法によって異なります:

- `--initialize` では、`--initialize-insecure` ではなくランダムパスワードが生成され、期限切れとしてマークされ、パスワードを表示するメッセージが書き込まれます:

```
[Warning] A temporary password is generated for root@localhost:  
iTag*AfrH5ej
```


- `--initialize-insecure` では (`--initialize-insecure` は `--initialize` を暗黙的に示すため、`--initialize` の有無にかかわらず)、サーバーはパスワードを生成したり期限切れとマークしたりせず、警告メッセージを書き込みます:

```
[Warning] root@localhost is created with an empty password ! Please
consider switching off the --initialize-insecure option.
```

新しい `'root'@'localhost'` パスワードを割り当てる手順については、[初期化後の root パスワードの割当て](#) を参照してください。

5. サーバーは、[HELP](#) ステートメントに使用されるサーバー側のヘルプテーブルに移入します ([セクション 13.8.3 「HELP ステートメント」](#) を参照)。サーバーはタイムゾーンテーブルを移入しません。これを手動で行うには、[セクション 5.1.15 「MySQL Server でのタイムゾーンのサポート」](#) を参照してください。
6. SQL ステートメントのファイルを指定するために `init_file` システム変数が指定された場合、サーバーはファイル内のステートメントを実行します。このオプションを使用すると、カスタムブートストラップシーケンスを実行できます。

サーバーがブートストラップモードで動作する場合、ファイルで許可されるステートメントを制限する一部の機能は使用できません。これには、アカウント管理 (`CREATE USER`、`GRANT` など)、レプリケーションおよびグローバルトランザクション識別子に関連するステートメントが含まれます。

7. サーバーが終了します。

初期化後の root パスワードの割当て

`--initialize` または `--initialize-insecure` でサーバーを起動してデータディレクトリを初期化した後、サーバーを通常どおり (つまり、これらのオプションのいずれも指定せずに) 起動し、`'root'@'localhost'` アカウントに新しいパスワードを割り当てます:

1. サーバーを起動します。その手順は、[セクション 2.10.2 「サーバーの起動」](#) を参照してください。
2. サーバーに接続します:

- `--initialize-insecure` ではなく `--initialize` を使用してデータディレクトリを初期化した場合は、`root` としてサーバーに接続します:

```
mysql -u root -p
```

次に、パスワードプロンプトで、サーバーが初期化シーケンス中に生成したランダムパスワードを入力します:

```
Enter password: (enter the random root password here)
```

このパスワードがわからない場合は、サーバーエラーログを参照してください。

- `--initialize-insecure` を使用してデータディレクトリを初期化した場合は、パスワードなしで `root` としてサーバーに接続します:

```
mysql -u root --skip-password
```

3. 接続後、`ALTER USER` ステートメントを使用して新しい `root` パスワードを割り当てます:

```
ALTER USER 'root'@'localhost' IDENTIFIED BY 'root-password';
```

[セクション 2.10.4 「初期 MySQL アカウントの保護」](#) も参照してください。

注記

ホスト `127.0.0.1` への接続を試行すると、通常は `localhost` アカウントに解決されます。ただし、サーバーが `skip_name_resolve` を有効にして実行されている場合、これは失敗します。これを行う場合は、接続を受け入れることができるアカウントが存在することを確認します。たとえば、`--host=127.0.0.1` または `--host>:::1` を使用して `root` として接続できるようにするには、次のアカウントを作成します:

```
CREATE USER 'root'@'127.0.0.1' IDENTIFIED BY 'root-password';
CREATE USER 'root'@':::1' IDENTIFIED BY 'root-password';
```

これらのステートメントは、[データディレクトリの初期化中のサーバーアクション](#) で説明されているように、`init_file` システム変数を使用して実行されるファイルに配置できます。

2.10.2 サーバーの起動

このセクションでは、Unix および Unix に似たシステムでサーバーを起動する方法について説明します。(Windows の場合は、[セクション2.3.4.5「サーバーをはじめて起動する」](#) を参照してください。) サーバーがアクセス可能で正しく動作しているかどうかをテストするために使用できるいくつかの推奨コマンドについては、[セクション2.10.3「サーバーのテスト」](#) を参照してください。

インストールに `mysqld_safe` が含まれる場合は、次のように MySQL サーバーを起動します:

```
shell> bin/mysqld_safe --user=mysql &
```

注記

RPM パッケージを使用して MySQL がインストールされている Linux システムでは、サーバーの起動と停止は `mysqld_safe` ではなく `systemd` を使用して管理され、`mysqld_safe` はインストールされません。[セクション2.5.9「systemd を使用した MySQL Server の管理」](#) を参照してください。

インストールに `systemd` サポートが含まれている場合は、次のようにサーバーを起動します:

```
shell> systemctl start mysqld
```

`mysqld` と異なる場合は、適切なサービス名 (SLES システム上の `mysql` など) に置き換えます。

MySQL Server を権限のない (非 `root`) ログインアカウントで起動することが重要です。これを確認するには、`root` として `mysqld_safe` を実行し、次に示すように `--user` オプションを含めます。それ以外の場合は、`mysql` としてログインしている間にプログラムを実行する必要があります。この場合、コマンドから `--user` オプションを省略できます。

MySQL を権限なしのユーザーとして実行する方法の詳細は、[セクション6.1.5「MySQL を通常ユーザーとして実行する方法」](#) を参照してください。

コマンドがただちに失敗して `mysqld ended` を出力する場合は、エラーログ (デフォルトではデータディレクトリ内の `host_name.err` ファイル) で情報を探してください。

サーバーが `mysql` スキーマ内の付与テーブルを起動または読み取るデータディレクトリにアクセスできない場合は、エラーログにメッセージが書き込まれます。このような問題は、このステップに進む前にデータディレクトリを初期化して付与テーブルを作成しない場合、または `--user` オプションを指定せずにデータディレクトリを初期化するコマンドを実行した場合に発生することがあります。`data` ディレクトリを削除し、`--user` オプションを指定してコマンドを実行します。

サーバーの起動時にほかの問題が発生した場合には、[セクション2.10.2.1「MySQL Server の起動時の問題のトラブルシューティング」](#) を参照してください。`mysqld_safe` の詳細は、[セクション4.3.2「mysqld_safe — MySQL サーバー起動スクリプト」](#) を参照してください。`systemd` のサポートの詳細は、[セクション2.5.9「systemd を使用した MySQL Server の管理」](#) を参照してください。

2.10.2.1 MySQL Server の起動時の問題のトラブルシューティング

このセクションでは、サーバーの起動に関する問題のトラブルシューティングの提案を提供します。Windows システムに関するその他の推奨事項については、[セクション2.3.5「Microsoft Windows MySQL Server インストールのトラブルシューティング」](#) を参照してください。

サーバーの起動時に問題がある場合、次を試してみます。

- **エラーログ** をチェックして、サーバーが起動しない原因を確認します。ログファイルはデータディレクトリ (通常、Windows では `C:\Program Files\MySQL\MySQL Server 8.0\data`、Unix/Linux バイナリ配布では `/usr/local/mysql/data`、Unix/Linux ソース配布では `/usr/local/var`) にあります。データディレクトリの `host_name.err` および `host_name.log` の形式のファイル名のファイルを探します。ここで、`host_name` はサーバーホストの名です。次にこれらのファイルの最後の数行を調べます。それらを表示するには `tail` を使用します。

```
shell> tail host_name.err
shell> tail host_name.log
```

- 使用しているストレージエンジンに必要なオプションがあれば指定します。 `my.cnf` ファイルを作成して、使用するエンジンの起動オプションを指定します。トランザクションテーブルをサポートするストレージエンジン (InnoDB、NDB) を使用する予定である場合、サーバーを起動する前に、それらが適切に構成されていることを確認します。InnoDB テーブルを使用している場合は、ガイドラインについては [セクション15.8「InnoDB の構成」](#)、オプションの構文については [セクション15.14「InnoDB の起動オプションおよびシステム変数」](#) を参照してください。

ストレージエンジンは、省略したオプションについてはデフォルト値を使用しますが、使用可能なオプションを確認し、デフォルト値がインストールに対して適切でないようなオプションがあれば明示的に値を指定することを推奨します。

- サーバーが、[データディレクトリ](#)を検索する場所を認識していることを確認してください。 `mysqld` サーバーはこのディレクトリを現在のディレクトリとして使用します。そこでデータベースを探し、ログファイルに書き込むことを想定しています。サーバーはデータディレクトリで `pid` (プロセス ID) ファイルも書き込みます。

デフォルトのデータディレクトリの場所は、サーバーのコンパイル時にハードコードされます。デフォルトのパス設定を確認するには、`mysqld` を `--verbose` オプションおよび `--help` オプションで起動します。データディレクトリがシステム上の別の場所にある場合は、その場所を、コマンド行またはオプションファイルで `--datadir` オプションを使用して `mysqld` または `mysqld_safe` に指定します。そうしないと、サーバーが正しく動作しません。`--datadir` オプションの代替として、MySQL がインストールされているベースディレクトリの場所を、`mysqld` に `--basedir` で指定できます。そうすれば、`mysqld` はそこで `data` ディレクトリを検索します。

パスオプションの指定の結果を知るには、`mysqld` をそれらのオプションで実行し次に `--verbose` および `--help` オプションを実行します。たとえば、`mysqld` がインストールされているディレクトリに場所を変更して次のコマンドを実行すると、`/usr/local` のベースディレクトリを使用してサーバーを起動した場合の影響が表示されます：

```
shell> ./mysqld --basedir=/usr/local --verbose --help
```

`--datadir` のようなオプションも同様に指定できますが、`--verbose` および `--help` は最後のオプションでなければなりません。

任意のパスを設定したあと、サーバーを `--verbose` および `--help` を使用しないで起動します。

`mysqld` が動作しているときは、次のコマンドを実行してどのパス設定が使用されているか確認できます。

```
shell> mysqladmin variables
```

または：

```
shell> mysqladmin -h host_name variables
```

`host_name` は MySQL サーバーのホスト名です。

- サーバーが、[データディレクトリ](#)にアクセスできることを確認してください。データディレクトリとそのコンテンツの所有権と許可は、サーバーがそれらを読み取って修正できるようなものでなければなりません。

`mysqld` の起動時に [Errcode 13 \(許可が却下されたことを意味します\)](#) を受け取る場合は、データディレクトリまたはそのコンテンツの特権が、サーバーアクセスを許可しないものであることを意味します。この場合、関連するファイルおよびディレクトリの権限を変更して、サーバーがそれらを使用する権限を持つようにします。サーバーを `root` から起動することもできますが、この場合セキュリティの問題が生じるため、避けるべきです。

データディレクトリの場所を変更し、データディレクトリとその内容の所有権をチェックして、サーバーがアクセスできることを確認します。たとえば、データディレクトリが `/usr/local/mysql/var` の場合は、次のコマンドを使用します。

```
shell> ls -la /usr/local/mysql/var
```

データディレクトリあるいはそのファイルまたはサブディレクトリが、サーバーを稼働するためのログインアカウントの所有でない場合、それらの所有権をそのアカウントに変更します。そのアカウントが `mysql` の場合、次のコマンドを使用します。

```
shell> chown -R mysql /usr/local/mysql/var
```

```
shell> chgrp -R mysql /usr/local/mysql/var
```

所有権が適切な場合でも、ファイルシステムのさまざまな部分へのアプリケーションのアクセスを管理するようなセキュリティソフトウェアがシステム上で実行されている場合、MySQL が起動に失敗することがあります。この場合は、そのソフトウェアを再構成して `mysqld` が通常の動作中に使用するディレクトリにアクセスできるようにします。

- サーバーが使用するネットワークインタフェースが利用できることを確認します。

次のいずれかのエラーが発生した場合、ほかのプログラム (おそらく別の `mysqld` サーバー) が、`mysqld` が使用する TCP/IP ポートあるいは Unix のソケットファイルを使用していることを意味します。

```
Can't start server: Bind on TCP/IP port: Address already in use
Can't start server: Bind on unix socket...
```

`ps` を使用して、別の `mysqld` サーバーが稼働しているかどうか判断します。その場合、`mysqld` を再度起動する前にサーバーをシャットダウンします。(別のサーバーが動作中で、複数のサーバーを稼働させる必要がある場合は、その方法に関する情報は、[セクション5.8「1つのマシン上での複数のMySQLインスタンスの実行」](#)にあります。)

別のサーバーが稼働していない場合は、コマンド `telnet your_host_name tcp_ip_port_number` を実行します。(デフォルトのMySQLポート番号は3306です。)次にEnterを数回押します。`telnet: Unable to connect to remote host: Connection refused`などのエラーメッセージが表示されない場合は、`mysqld` が使用しようとしているTCP/IPポートをほかのプログラムが使用しています。それがどのプログラムなのかを追跡して無効にするか、`--port` オプションを使用して、`mysqld` に別のポートで待機するよう指示します。この場合、TCP/IP を使用してサーバーに接続するときに、クライアントプログラムと同じデフォルト以外のポート番号を指定します。

ポートにアクセスできない別の理由に、ファイアウォールがその接続をブロックしている場合があります。その場合は、ファイアウォールの設定を、ポートへのアクセスを許可するように変更します。

サーバーは起動するがそれに接続できない場合は、`/etc/hosts` に次のようなエントリがあることを確認します。

```
127.0.0.1 localhost
```

- `mysqld` を起動できない場合は、`--debug` オプションを使用して、トレースファイルを作成して問題を発見してみてください。[セクション5.9.4「DBUGパッケージ」](#)を参照してください。

2.10.3 サーバーのテスト

データディレクトリを初期化し、サーバーを起動したら、いくつかの単純なテストを実行して、適切に動作することを確認します。このセクションでは、現在の場所がMySQLインストールディレクトリであり、ここで使用するMySQLプログラムを含む`bin`サブディレクトリがあることを前提としています。そうでない場合は、コマンドパス名を適宜調整します。

または、`bin` ディレクトリを `PATH` 環境変数設定に追加します。これにより、シェル(コマンドインタプリタ)はMySQLプログラムを適切に検索できるため、パス名ではなく名前のみを入力してプログラムを実行できます。[セクション4.2.9「環境変数の設定」](#)を参照してください。

`mysqladmin` を使用してサーバーが動作していることを確認します。次のコマンドは、サーバーの起動および接続を確認する簡単なテストを提供します。

```
shell> bin/mysqladmin version
shell> bin/mysqladmin variables
```

サーバーに接続できない場合は、`-u root` オプションを指定して `root` として接続します。`root` アカウントのパスワードをすでに割り当てている場合は、コマンドラインで `-p` を指定し、プロンプトが表示されたらパスワードを入力する必要があります。例:

```
shell> bin/mysqladmin -u root -p version
Enter password: (enter root password here)
```

`mysqladmin version` の出力は、プラットフォームおよびMySQLのバージョンによって多少異なりますが、次に示すものに類似します。

```
shell> bin/mysqladmin version
```

```
mysqladmin Ver 14.12 Distrib 8.0.29, for pc-linux-gnu on i686
...
Server version      8.0.29
Protocol version    10
Connection          Localhost via UNIX socket
UNIX socket         /var/lib/mysql/mysql.sock
Uptime:             14 days 5 hours 5 min 21 sec

Threads: 1  Questions: 366  Slow queries: 0
Opens: 0  Flush tables: 1  Open tables: 19
Queries per second avg: 0.000
```

`mysqladmin` のほかに機能を表示するには、それを `--help` オプションで起動します。

サーバーを停止できることを確認します (`root` アカウントにすでにパスワードがある場合は、`-p` オプションを含めま

```
shell> bin/mysqladmin -u root shutdown
```

サーバーを再度起動できることを確認します。これは、`mysqld_safe` を使用するか、あるいは `mysqld` を直接起動して行います。例:

```
shell> bin/mysqld_safe --user=mysql &
```

`mysqld_safe` が失敗する場合は、[セクション2.10.2.1「MySQL Serverの起動時の問題のトラブルシューティング」](#)を参照します。

簡単なテストをいくつか実行して、サーバーから情報を取り出せることを確認します。出力は次のようになります。

どのようなデータベースが存在するかを表示するには、`mysqlshow` を使用します。

```
shell> bin/mysqlshow
+-----+
| Databases |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
```

インストールされているデータベースのリストは異なる場合がありますが、常に `mysql` および `information_schema` 以上が含まれています。

データベース名を指定すると、`mysqlshow` はそのデータベース内のテーブルのリストを表示します。

```
shell> bin/mysqlshow mysql
Database: mysql
+-----+
| Tables |
+-----+
| columns_priv |
| component |
| db |
| default_roles |
| engine_cost |
| func |
| general_log |
| global_grants |
| gtid_executed |
| help_category |
| help_keyword |
| help_relation |
| help_topic |
| innodb_index_stats |
| innodb_table_stats |
| ndb_binlog_index |
| password_history |
| plugin |
| procs_priv |
```

```

proxies_priv |
role_edges |
server_cost |
servers |
slave_master_info |
slave_relay_log_info |
slave_worker_info |
slow_log |
tables_priv |
time_zone |
time_zone_leap_second |
time_zone_name |
time_zone_transition |
time_zone_transition_type |
user |
+-----+

```

mysql プログラムを使用して、mysql スキーマのテーブルから情報を選択します:

```

shell> bin/mysql -e "SELECT User, Host, plugin FROM mysql.user" mysql
+-----+-----+-----+
| User | Host | plugin |
+-----+-----+-----+
| root | localhost | caching_sha2_password |
+-----+-----+-----+

```

この時点で、サーバーは稼働しており、アクセスできます。初期アカウントにまだパスワードを割り当てていない場合にセキュリティを強化するには、[セクション2.10.4「初期 MySQL アカウントの保護」](#) の手順に従います。

mysql、mysqldadmin および mysqlshow の詳細は、[セクション4.5.1「mysql — MySQL コマンドラインクライアント」](#)、[セクション4.5.2「mysqldadmin — A MySQL Server 管理プログラム」](#) および [セクション4.5.7「mysqlshow — データベース、テーブル、およびカラム情報の表示」](#) を参照してください。

2.10.4 初期 MySQL アカウントの保護

MySQL のインストールプロセスには、MySQL アカウントを定義する mysql システムスキーマの付与テーブルを含む、データディレクトリの初期化が含まれます。詳細は、[セクション2.10.1「データディレクトリの初期化」](#) を参照してください。

このセクションでは、MySQL のインストール手順中に作成された初期 root アカウントにパスワードを割り当てる方法について説明します (まだ行っていない場合)。

注記

このセクションで説明するプロセスを実行する別の方法:

- Windows では、MySQL Installer を使用したインストール中にこのプロセスを実行できます ([セクション2.3.3「MySQL Installer for Windows」](#) を参照)。
- すべてのプラットフォームで、MySQL 配布には、MySQL インストールをセキュアにするプロセスの大部分を自動化するコマンド行ユーティリティー `mysql_secure_installation` が含まれます。
- すべてのプラットフォームで、MySQL Workbench を使用でき、ユーザーアカウントを管理できます ([第31章「MySQL Workbench」](#) を参照)。

次の状況では、初期アカウントにパスワードがすでに割り当てられている可能性があります:

- Windows では、MySQL Installer を使用してインストールを実行すると、パスワードを割り当てることはできません。
- macOS インストーラを使用したインストールでは、初期ランダムパスワードが生成されます。このパスワードは、インストーラによってダイアログボックスに表示されます。
- RPM パッケージを使用したインストールでは、サーバーエラーログに書き込まれる初期ランダムパスワードが生成されます。

- Debian パッケージを使用したインストールでは、パスワードを割り当てるオプションが提供されます。
- `mysqld --initialize` を使用して手動で実行されるデータディレクトリの初期化の場合、`mysqld` は初期ランダムパスワードを生成し、期限切れとマークして、サーバーエラーログに書き込みます。 [セクション2.10.1「データディレクトリの初期化」](#) を参照してください。

`mysql.user` 付与テーブルは、初期 MySQL ユーザーアカウントとそのアクセス権限を定義します。MySQL をインストールすると、すべての権限を持ち、何でも実行できる `'root'@'localhost'` スーパーユーザーアカウントのみが作成されます。 `root` アカウントのパスワードが空の場合、MySQL のインストールは保護されません: 誰でも `root` パスワードなしとして MySQL サーバーに接続でき、すべての権限を付与できます。

`'root'@'localhost'` アカウントの `mysql.proxies_priv` テーブルには、`"@"` の `PROXY` 権限 (すべてのユーザーおよびすべてのホスト) の付与を可能にする行もあります。これにより、`root` はプロキシユーザーを設定したり、プロキシユーザーを設定するための権限をほかのアカウントに委任したりできます。 [セクション6.2.18「プロキシユーザー」](#) を参照してください。

初期 MySQL `root` アカウントのパスワードを割り当てるには、次の手順を使用します。例の `root-password` を、使用するパスワードに置き換えます。

サーバーが実行されていない場合は起動します。その手順は、 [セクション2.10.2「サーバーの起動」](#) を参照してください。

初期 `root` アカウントには、パスワードがある場合とない場合があります。次の手順のいずれかを選択します:

- 期限切れの初期ランダムパスワードを持つ `root` アカウントが存在する場合は、そのパスワードを使用して `root` としてサーバーに接続し、新しいパスワードを選択します。これは、データディレクトリが手動で、またはインストール操作中にパスワードを指定できないインストーラを使用して `mysqld --initialize` を使用して初期化された場合です。パスワードは存在するため、サーバーへの接続に使用する必要があります。ただし、パスワードの有効期限が切れているため、新しいパスワードを選択する以外の目的でアカウントを使用することはできません。

1. 初期ランダムパスワードがわからない場合は、サーバーエラーログを参照してください。
2. パスワードを使用して `root` としてサーバーに接続します:

```
shell> mysql -u root -p
Enter password: (enter the random root password here)
```

3. ランダムパスワードを置き換える新しいパスワードを選択します:

```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'root-password';
```

- `root` アカウントは存在するがパスワードがない場合は、パスワードを使用せずに `root` としてサーバーに接続し、パスワードを割り当てます。これは、`mysqld --initialize-insecure` を使用してデータディレクトリを初期化した場合です。

1. パスワードを使用せずに `root` としてサーバーに接続します:

```
shell> mysql -u root --skip-password
```

2. パスワードを割り当てます:

```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'root-password';
```

`root` アカウントにパスワードを割り当てた後、そのアカウントを使用してサーバーに接続するたびにそのパスワードを指定する必要があります。たとえば、`mysql` クライアントを使用してサーバーに接続するには、次のコマンドを使用します:

```
shell> mysql -u root -p
Enter password: (enter root password here)
```

`mysqldadmin` でサーバーを停止するには、次のコマンドを使用します:

```
shell> mysqldadmin -u root -p shutdown
Enter password: (enter root password here)
```

注記

パスワード設定の詳細は、[セクション6.2.14「アカウントパスワードの割り当て」](#)を参照してください。root パスワードを設定したあとでそれを忘れた場合は、[セクションB.3.3.2「root のパスワードをリセットする方法」](#)を参照してください。

追加のアカウントをセットアップするには、[セクション6.2.8「アカウントの追加、権限の割当ておよびアカウントの削除」](#)を参照してください。

2.10.5 MySQL を自動的に起動および停止する

このセクションでは、MySQL サーバーを起動および停止する方法について説明します。

一般的には、`mysqld` サーバーは次のいずれかの方法で起動します。

- 直接 `mysqld` を呼び出します。これはどのプラットフォームでも機能します。
- Windows では、Windows の起動時に自動的に実行される MySQL サービスを設定できます。[セクション2.3.4.8「Windows のサービスとして MySQL を起動する」](#)を参照してください。
- Unix および Unix に似たシステムでは、`mysqld_safe` を起動して `mysqld` の適切なオプションを決定し、それらのオプションを使用して実行できます。[セクション4.3.2「mysqld_safe — MySQL サーバー起動スクリプト」](#)を参照してください。
- `systemd` をサポートする Linux システムでは、これを使用してサーバーを制御できます。[セクション2.5.9「systemd を使用した MySQL Server の管理」](#)を参照してください。
- System V 形式の実行ディレクトリ (つまり、`/etc/init.d` および実行レベル固有のディレクトリ) を使用するシステムでは、`mysql.server` を起動します。このスクリプトは、主にシステムの起動および停止時に使用されます。通常は、`mysql` という名前でインストールされます。`mysql.server` スクリプトは `mysqld_safe` を実行してサーバーを起動します。[セクション4.3.3「mysql.server — MySQL サーバー起動スクリプト」](#)を参照してください。
- macOS では、`launchd` デーモンをインストールして、システム起動時の MySQL の自動起動を有効にします。デーモンは、`mysqld_safe` を呼び出してサーバーを起動します。詳細は、[セクション2.4.3「MySQL 起動デーモンのインストールおよび使用」](#)を参照してください。MySQL プリファレンスペインでは、システムプリファレンスを使用して MySQL を起動および停止するためのコントロールも提供されます。[セクション2.4.4「MySQL Preference Pane のインストールと使用」](#)を参照してください。
- Solaris では、サービス管理フレームワーク (SMF) システムを使用して、MySQL の起動を開始および制御します。

`systemd`、`mysqld_safe` および `mysql.server` スクリプト、Solaris SMF、および macOS 起動項目 (または MySQL 設定ペイン) を使用すると、サーバーを手動で、またはシステムの起動時に自動的に起動できます。`systemd`、`mysql.server`、および起動項目を使用して、サーバーを停止することもできます。

次のテーブルに、サーバーおよび起動スクリプトがオプションファイルから読み取るオプショングループを示します。

表 2.15 MySQL 起動スクリプトおよびサポートされているサーバーオプショングループ

スクリプト	オプショングループ
<code>mysqld</code>	<code>[mysqld]</code> 、 <code>[server]</code> 、 <code>[mysqld-major_version]</code>
<code>mysqld_safe</code>	<code>[mysqld]</code> 、 <code>[server]</code> 、 <code>[mysqld_safe]</code>
<code>mysql.server</code>	<code>[mysqld]</code> 、 <code>[mysql.server]</code> 、 <code>[server]</code>

`[mysqld-major_version]` は、`[mysqld-5.7]` および `[mysqld-8.0]` のような名前のグループが、5.7.x、8.0.x、などのバージョンのサーバーによって読み取られることを意味します。この機能は、所定のリリースシリーズのサーバーによってのみ読み取られるオプションを指定するために使用できます。

下位互換性のため、`mysql.server` は `[mysql_server]` グループも読み取り、`mysqld_safe` は `[safe_mysqld]` グループも読み取ります。最新にするには、かわりに `[mysql.server]` および `[mysqld_safe]` グループを使用するようにオプションファイルを更新する必要があります。

MySQL 構成ファイルとその構造とコンテンツについては、[セクション4.2.2.2「オプションファイルの使用」](#)を参照してください。

2.11 MySQL のアップグレード

このセクションでは、MySQL インストールをアップグレードするステップについて説明します。

MySQL の同じリリースシリーズ内でのバグフィックスや、MySQL の主なリリース間の重大な機能を入手する場合など、アップグレードは一般的な手順です。この手順は、テストシステムでまず実行して、すべてが問題なく動作することを確認してから本番システムで実行します。

注記

次の説明では、管理権限を持つ MySQL アカウントを使用して実行する必要がある MySQL コマンドに、MySQL `root` ユーザーを指定するための `-u root` がコマンドラインに含まれています。`root` のパスワードを必要とするコマンドには、`-p` オプションも含まれます。`-p` の後にオプション値がないため、このようなコマンドではパスワードの入力を求められます。プロンプトが表示されたらパスワードを入力し、Enter キーを押します。

SQL ステートメントは、`mysql` コマンドラインクライアントを使用して実行できます (`root` として接続し、必要な権限があることを確認します)。

2.11.1 始める前に

アップグレードする前に、このセクションの情報を確認してください。推奨されるアクションを実行します。

- アップグレード中に発生する可能性があることを理解します。[セクション2.11.3「MySQL のアップグレードプロセスの内容」](#)を参照してください。
- バックアップを作成してデータを保護します。バックアップには、MySQL データディクショナリテーブルおよびシステムテーブルを含む `mysql` システムデータベースを含める必要があります。[セクション7.2「データベースバックアップ方法」](#)を参照してください。

重要

MySQL 8.0 から MySQL 5.7 へのダウングレード、または MySQL 8.0 リリースから以前の MySQL 8.0 リリースへのダウングレードはサポートされていません。サポートされている唯一の代替方法は、アップグレード前に作成したバックアップをリストアすることです。したがって、アップグレードプロセスを開始する前にデータをバックアップする必要があります。

- [セクション2.11.2「アップグレードパス」](#)を確認して、意図したアップグレードパスがサポートされていることを確認します。
- アップグレード前に認識しておく必要がある変更については、[セクション2.11.4「MySQL 8.0 での変更」](#)を確認します。一部の変更にはアクションが必要な場合があります。
- 非推奨となった機能および削除された機能については、[セクション1.3「MySQL 8.0 の新機能」](#)を確認してください。アップグレードでは、これらの機能のいずれかを使用する場合、それらの機能の変更が必要になることがあります。
- [セクション1.4「MySQL 8.0 で追加、非推奨または削除されたサーバーおよびステータスの変数とオプション」](#)を確認します。非推奨の変数または削除された変数を使用する場合、アップグレードで構成の変更が必要になることがあります。
- 修正、変更および新機能の詳細は、「[リリースノート](#)」を確認してください。
- レプリケーションを使用する場合は、[セクション17.5.3「レプリケーションセットアップをアップグレードする」](#)を確認します。
- アップグレード手順は、プラットフォームおよび初期インストールの実行方法によって異なります。現在の MySQL インストールに適用される手順を使用してください。

- Windows 以外のプラットフォームでのバイナリおよびパッケージベースのインストールについては、[セクション 2.11.6 「Unix/Linux での MySQL バイナリまたはパッケージベースのインストールのアップグレード」](#) を参照してください。

注記

サポートされている Linux ディストリビューションの場合、パッケージベースのインストールをアップグレードするには、MySQL ソフトウェアリポジトリ (MySQL Yum Repository、MySQL APT Repository、および MySQL SLES Repository) を使用することをお勧めします。

- MySQL Yum Repository を使用した Enterprise Linux プラットフォームまたは Fedora へのインストールについては、[セクション 2.11.7 「MySQL Yum リポジトリを使用する MySQL のアップグレード」](#) を参照してください。
- MySQL APT リポジトリを使用した Ubuntu へのインストールについては、[セクション 2.11.8 「MySQL APT リポジトリを使用する MySQL のアップグレード」](#) を参照してください。
- MySQL SLES リポジトリを使用した SLES へのインストールについては、[セクション 2.11.9 「MySQL SLES リポジトリを含む MySQL のアップグレード」](#) を参照してください。
- Docker を使用して実行されるインストールについては、[セクション 2.11.11 「MySQL の Docker インストールのアップグレード」](#) を参照してください。
- Windows へのインストールについては、[セクション 2.11.10 「Windows 上の MySQL をアップグレードする」](#) を参照してください。
- MySQL インストールに、インプレースアップグレード後の変換にかかる可能性のある大量のデータが含まれている場合、必要な変換およびそれらの実行に関連する作業を評価するためのテストインスタンスを作成すると便利な場合があります。テストインスタンスを作成するには、`mysql` データベースおよびデータのない他のデータベースを含む MySQL インスタンスのコピーを作成します。テストインスタンスでアップグレード手順を実行して、実際のデータ変換の実行に関連する作業を評価します。
- MySQL の新しいリリースをインストールまたはアップグレードする場合は、MySQL 言語インタフェースの再構築および再インストールをお勧めします。これは PHP `mysql` 拡張機能や Perl `DBD::mysql` モジュールなどの MySQL インタフェースに適用されます。

2.11.2 アップグレードパス

- MySQL 5.7 から 8.0 へのアップグレードがサポートされています。ただし、アップグレードは General Availability (GA) リリース間でのみサポートされます。MySQL 8.0 の場合、MySQL 5.7 GA リリース (5.7.9 以上) からアップグレードする必要があります。MySQL 5.7 の非 GA リリースからのアップグレードはサポートされていません。
- 次のバージョンにアップグレードする前に、最新リリースにアップグレードすることをお勧めします。たとえば、MySQL 8.0 にアップグレードする前に、最新の MySQL 5.7 リリースにアップグレードします。
- バージョンをスキップするアップグレードはサポートされていません。たとえば、MySQL 5.6 から 8.0 への直接アップグレードはサポートされていません。
- リリースシリーズが GA (General Availability) ステータスに達すると、リリースシリーズ内で (GA バージョン間で) アップグレードがサポートされます。たとえば、MySQL 8.0 からアップグレードします。x から 8.0 へ。y がサポートされています。(開発ステータス非 GA リリースを含むアップグレードはサポートされていません。) リリースのスキップもサポートされています。たとえば、MySQL 8.0 からアップグレードします。x から 8.0 へ。z がサポートされています。MySQL 8.0.11 は、MySQL 8.0 リリースシリーズ内の最初の GA ステータスリリースです。

2.11.3 MySQL のアップグレードプロセスの内容

新しいバージョンの MySQL をインストールするには、既存のインストールの次の部分をアップグレードする必要があります:

- `mysql` システムスキーマ。MySQL サーバーの実行時に必要な情報を格納するテーブルが含まれています ([セクション 5.3 「mysql システムスキーマ」](#) を参照)。`mysql` スキーマテーブルは、次の 2 つの広範なカテゴリに分類されます:

- データディクショナリテーブル:データベースオブジェクトメタデータを格納します。
- 他の操作目的で使用されるシステムテーブル (残りの非データディクショナリテーブル)。
- その他のスキーマ。一部は組込みであり、サーバーによって「owned」とみなされる場合があります。その他のスキーマには、次のものはありません:
 - パフォーマンススキーマ、[INFORMATION_SCHEMA](#)、[ndbinfo](#)、および [sys](#) スキーマ。
 - ユーザースキーマ。

アップグレードが必要になる可能性があるインストールの各部分には、次の 2 つの異なるバージョン番号が関連付けられています:

- データディクショナリのバージョン。これはデータディクショナリテーブルに適用されます。
- サーバーのバージョン (MySQL バージョンとも呼ばれます)。これは、他のスキーマ内のシステムテーブルおよびオブジェクトに適用されます。

どちらの場合も、既存の MySQL インストールに適用可能な実際のバージョンがデータディクショナリに格納され、現在予想されているバージョンが新しいバージョンの MySQL にコンパイルされます。実際のバージョンが現在予想されているバージョンより低い場合は、そのバージョンに関連付けられているインストールの一部を現在のバージョンにアップグレードする必要があります。両方のバージョンでアップグレードが必要であることが示されている場合は、まずデータディクショナリのアップグレードを実行する必要があります。

前述の 2 つの異なるバージョンの反射として、アップグレードは次の 2 つのステップで実行されます:

- ステップ 1: データディクショナリのアップグレード。

このステップでは、次のアップグレードを行います:

- [mysql](#) スキーマ内のデータディクショナリテーブル。実際のデータディクショナリのバージョンが現在予想されているバージョンより低い場合、サーバーは、更新された定義を含むデータディクショナリテーブルを作成し、永続化されたメタデータを新しいテーブルにコピーし、古いテーブルを新しいテーブルに原子的に置き換え、データディクショナリを再初期化します。
 - パフォーマンススキーマ、[INFORMATION_SCHEMA](#)、および [ndbinfo](#)。
- ステップ 2: サーバーのアップグレード。

このステップは、他のすべてのアップグレードタスクで構成されます。既存の MySQL インストールのサーバーバージョンが、新しくインストールされた MySQL バージョンより低い場合は、他のすべてをアップグレードする必要があります:

- [mysql](#) スキーマ内のシステムテーブル (残りの非データディクショナリテーブル)。
- [sys](#) スキーマ。
- ユーザースキーマ。

データディクショナリのアップグレード (ステップ 1) は、起動時に必要に応じてこのタスクを実行するサーバーの役割です。ただし、起動を妨げるオプションを指定して起動した場合は除きます。このオプションは、MySQL 8.0.16、MySQL 8.0.16 より前の [--no-dd-upgrade](#) の時点での [--upgrade=NONE](#) です。

データディクショナリは期限切れだが、サーバーがアップグレードできない場合、サーバーは実行されず、かわりにエラーで終了します。例:

```
[ERROR] [MY-013381] [Server] Server shutting down because upgrade is
required, yet prohibited by the command line option '--upgrade=NONE'.
[ERROR] [MY-010334] [Server] Failed to initialize DD Storage Engine
[ERROR] [MY-010020] [Server] Data Dictionary initialization failed.
```

ステップ 2 の職責に対するいくつかの変更は、MySQL 8.0.16 で発生しました:

- MySQL 8.0.16 より前は、`mysql_upgrade` によって、パフォーマンススキーマ、`INFORMATION_SCHEMA` およびステップ 2 で説明したオブジェクトがアップグレードされます。DBA は、サーバーの起動後に `mysql_upgrade` を手動で起動する必要があります。
- MySQL 8.0.16 の時点で、サーバーは以前に `mysql_upgrade` によって処理されたすべてのタスクを実行します。アップグレードは 2 ステップの操作のままですが、サーバーは両方を実行するため、プロセスが簡単になります。

アップグレード先の MySQL のバージョンに応じて、[インプレースアップグレード](#) および [論理アップグレード](#) の手順は、サーバーがすべてのアップグレードタスクを実行するか、サーバーの起動後に `mysql_upgrade` も起動する必要があるかを示します。

注記

サーバーはパフォーマンススキーマ、`INFORMATION_SCHEMA`、および MySQL 8.0.16 の時点でステップ 2 で説明されているオブジェクトをアップグレードするため、`mysql_upgrade` は不要であり、そのバージョンでは非推奨になっています。将来のバージョンの MySQL で削除される予定です。

ステップ 2 で発生する処理のほとんどは、MySQL 8.0.16 の前と時点で同じですが、特定の効果を得るために異なるコマンドオプションが必要になる場合があります。

MySQL 8.0.16 の時点で、`--upgrade` サーバーオプションは、サーバーが起動時に自動アップグレードを実行するかどうか、およびその方法を制御します:

- オプションを指定しない場合、または `--upgrade=AUTO` を指定した場合、サーバーは最新でないと判断したものをすべてアップグレードします (ステップ 1 および 2)。
- `--upgrade=NONE` では、サーバーは何もアップグレードしません (ステップ 1 および 2 はスキップします)。ただし、データディクショナリをアップグレードする必要がある場合はエラーで終了します。古いデータディクショナリを使用してサーバーを実行することはできません。サーバーはサーバーのアップグレードまたは終了を要求しません。
- `--upgrade=MINIMAL` を使用すると、サーバーは必要に応じてデータディクショナリ、パフォーマンススキーマおよび `INFORMATION_SCHEMA` をアップグレードします (ステップ 1)。このオプションを使用したアップグレード後は、レプリケーション内部が依存するシステムテーブルは更新されず、他の領域でも機能が低下する可能性があるため、Group Replication を起動できないことに注意してください。
- `--upgrade=FORCE` では、サーバーは必要に応じてデータディクショナリ、パフォーマンススキーマおよび `INFORMATION_SCHEMA` をアップグレードし (ステップ 1)、他のすべてのものを強制的にアップグレードします (ステップ 2)。サーバーはすべてのスキーマ内のすべてのオブジェクトをチェックするため、このオプションではサーバーの起動に時間がかかります。

`FORCE` は、必要ないとサーバーが判断した場合に、ステップ 2 のアクションを強制的に実行する場合に役立ちます。`FORCE` と `AUTO` の違いの 1 つは、`FORCE` では、ヘルプテーブルやタイムゾーンテーブルが欠落している場合は、サーバーによってシステムテーブル (ヘルプテーブルやタイムゾーンテーブルなど) が再作成されることです。

次のリストは、MySQL 8.0.16 より前のアップグレードコマンドと、MySQL 8.0.16 以上の同等のコマンドを示しています:

- 通常のアップグレードを実行します (必要に応じてステップ 1 および 2):
 - MySQL 8.0.16 より前: `mysqld` とそれに続く `mysql_upgrade`
 - MySQL 8.0.16 の時点: `mysqld`
- 必要に応じて、ステップ 1 のみを実行します:
 - MySQL 8.0.16 より前: ステップ 2 で説明したアップグレードタスクを除外して、ステップ 1 で説明したすべてのアップグレードタスクを実行することはできません。ただし、ユーザースキーマおよび `sys` スキーマのアップグレードは、`mysqld` の後に `--upgrade-system-tables` および `--skip-sys-schema` オプションを指定した `mysql_upgrade` を使用して回避できます。
 - MySQL 8.0.16 の時点: `mysqld --upgrade=MINIMAL`

- 必要に応じてステップ 1 を実行し、ステップ 2 を強制します:
 - MySQL 8.0.16 より前: `mysql` とそれに続く `mysql_upgrade --force`
 - MySQL 8.0.16 の時点: `mysql --upgrade=FORCE`

MySQL 8.0.16 より前は、特定の `mysql_upgrade` オプションが実行するアクションに影響します。次のテーブルに、同様の効果を得るために MySQL 8.0.16 の時点で使用するサーバー `--upgrade` オプションの値を示します。(指定された `--upgrade` オプション値には追加の効果がある場合があるため、これらは必ずしも完全に同等であるとはかぎりません。)

mysql_upgrade オプション	サーバーオプション
<code>--skip-sys-schema</code>	<code>--upgrade=NONE</code> または <code>--upgrade=MINIMAL</code>
<code>--upgrade-system-tables</code>	<code>--upgrade=NONE</code> または <code>--upgrade=MINIMAL</code>
<code>--force</code>	<code>--upgrade=FORCE</code>

アップグレードステップ 2 で発生する処理に関する追加のノート:

- ステップ 2 では、`sys` スキーマがインストールされていない場合はインストールし、インストールされていない場合は現在のバージョンにアップグレードします。`sys` スキーマは存在するが `version` ビューがない場合、その存在しないという前提で、ユーザーが作成したスキーマがあるとエラーが発生します:

```
A sys schema exists with no sys.version view. If
you have a user created sys schema, this must be renamed for the
upgrade to succeed.
```

この場合にアップグレードするには、まず既存の `sys` スキーマを削除するか、名前を変更します。その後、アップグレード手順を再度実行します。(場合によっては、ステップ 2 を強制的に実行する必要があります。)

`sys` スキーマチェックを防止するには:

- MySQL 8.0.16 の時点: `--upgrade=NONE` または `--upgrade=MINIMAL` オプションを使用してサーバーを起動します。
- MySQL 8.0.16 より前: `--skip-sys-schema` オプションを指定して `mysql_upgrade` を起動します。
- ステップ 2 では、必要に応じてすべてのユーザースキーマのすべてのテーブルを処理します。テーブルチェックの完了には時間がかかる場合があります。各テーブルはロックされるため、処理中にほかのセッションで使用することはできません。チェックと修復の処理には時間がかかることがあり、特に大きなテーブルでは長い時間を要する可能性があります。テーブルチェックでは、`CHECK TABLE` ステートメントの `FOR UPGRADE` オプションを使用します。このオプションに必要な内容の詳細は、[セクション 13.7.3.2 「CHECK TABLE ステートメント」](#) を参照してください。

テーブルチェックを防止するには:

- MySQL 8.0.16 の時点: `--upgrade=NONE` または `--upgrade=MINIMAL` オプションを使用してサーバーを起動します。
- MySQL 8.0.16 より前: `--upgrade-system-tables` オプションを指定して `mysql_upgrade` を起動します。

テーブルチェックを強制するには:

- MySQL 8.0.16 の時点: `--upgrade=FORCE` オプションを使用してサーバーを起動します。
- MySQL 8.0.16 より前: `--force` オプションを指定して `mysql_upgrade` を起動します。
- ステップ 2 では、MySQL のバージョン番号をデータディレクトリ内の `mysql_upgrade_info` という名前のファイルに保存します。

`mysql_upgrade_info` ファイルを無視して、次のことに関係なくチェックを実行するには:

- MySQL 8.0.16 の時点: `--upgrade=FORCE` オプションを使用してサーバーを起動します。

- MySQL 8.0.16 より前: `--force` オプションを指定して `mysql_upgrade` を起動します。

注記

`mysql_upgrade_info` ファイルは非推奨です。将来のバージョンの MySQL で削除される予定です。

- ステップ 2 では、チェックおよび修復されたすべてのテーブルに現在の MySQL バージョン番号を付けます。これにより、同じバージョンのサーバーで次のアップグレードチェックが行われるときに、特定のテーブルを再度チェックまたは修復する必要があるかどうかを判断できます。
- ステップ 2 では、システムテーブルをアップグレードして、現在の構造になるようにします。これは、サーバーと `mysql_upgrade` のどちらがこのステップを実行するかに関係なく当てはまります。ヘルプテーブルおよびタイムゾーンテーブルの内容に関して、`mysql_upgrade` はどちらのタイプのテーブルもロードしませんが、サーバーはヘルプテーブルをロードしますが、タイムゾーンテーブルはロードしません。(つまり、MySQL 8.0.16 より前では、サーバーはデータディレクトリの初期化時のみヘルプテーブルをロードします。MySQL 8.0.16 では、初期化およびアップグレード時にヘルプテーブルがロードされます。)タイムゾーンテーブルをロードする手順はプラットフォームによって異なり、DBA による決定が必要なため、自動的に実行できません。

2.11.4 MySQL 8.0 での変更

MySQL 8.0 にアップグレードする前に、このセクションで説明されている変更を確認して、現在の MySQL インストールおよびアプリケーションに適用される変更を特定します。推奨されるアクションを実行します。

互換性のない変更としてマークされた変更は、以前のバージョンの MySQL との互換性がないため、アップグレード前に注意する必要がある場合があります。弊社の目的はそれらの変更を避けることですが、各リリースの間の非適合性よりもさらに深刻な問題を修正するために必要である場合もあります。インストールに適用可能なアップグレードの問題に互換性がない場合は、説明に示されている手順に従ってください。

- [データディクショナリの変更](#)
- [優先認証プラグインとしての `caching_sha2_password`](#)
- [構成の変更](#)
- [サーバーの変更](#)
- [InnoDB の変更点](#)
- [SQL の変更](#)

データディクショナリの変更

MySQL Server 8.0 には、トランザクションテーブルのデータベースオブジェクトに関する情報を含むグローバルデータディクショナリが組み込まれています。以前の MySQL シリーズでは、ディクショナリデータはメタデータファイルおよび非トランザクションシステムテーブルに格納されていました。そのため、アップグレード手順では、特定の前提条件を確認して、インストールのアップグレード準備状況を確認する必要があります。詳細は、[セクション 2.11.5 「アップグレード用のインストールの準備」](#)を参照してください。データディクショナリ対応サーバーには、一般的な操作上の違いがいくつかあります。[セクション 14.7 「データディクショナリの使用方法の違い」](#)を参照してください。

優先認証プラグインとしての `caching_sha2_password`

`caching_sha2_password` および `sha256_password` 認証プラグインは、`mysql_native_password` プラグインよりもセキュアなパスワード暗号化を提供し、`caching_sha2_password` は `sha256_password` よりも優れたパフォーマンスを提供します。`caching_sha2_password` のこれらの優れたセキュリティおよびパフォーマンス特性のため、MySQL 8.0 が優先認証プラグインであり、`mysql_native_password` ではなくデフォルトの認証プラグインでもあります。この変更は、サーバーと `libmysqlclient` クライアントライブラリの両方に影響します:

- サーバーの場合、`default_authentication_plugin` システム変数のデフォルト値が `mysql_native_password` から `caching_sha2_password` に変更されます。

この変更は、MySQL 8.0 以上のインストールまたはアップグレード後に作成された新しいアカウントにのみ適用されます。アップグレードインストールにすでに存在するアカウントの場合、認証プラグインは変更されません。 `caching_sha2_password` に切り替える既存のユーザーは、 `ALTER USER` ステートメントを使用してこれを実行できます:

```
ALTER USER user
IDENTIFIED WITH caching_sha2_password
BY 'password';
```

- `libmysqlclient` ライブラリは、 `caching_sha2_password` を `mysql_native_password` ではなくデフォルトの認証プラグインとして扱います。

次の各セクションでは、 `caching_sha2_password` のより有望なロールの影響について説明します:

- [caching_sha2_password の互換性の問題および解決策](#)
- [caching_sha2_password-Compatible クライアントおよびコネクタ](#)
- [caching_sha2_password および root 管理アカウント](#)
- [caching_sha2_password とレプリケーション](#)

caching_sha2_password の互換性の問題および解決策

重要

MySQL インストールで 8.0 より前のクライアントを提供する必要があり、MySQL 8.0 以上へのアップグレード後に互換性の問題が発生した場合、これらの問題に対処して 8.0 より前の互換性をリストアする最も簡単な方法は、以前のデフォルトの認証プラグイン (`mysql_native_password`) に戻すようにサーバーを再構成することです。たとえば、サーバーオプションファイルで次の行を使用します:

```
[mysqld]
default_authentication_plugin=mysql_native_password
```

この設定により、インストールで使用されているクライアントおよびコネクタが `caching_sha2_password` を認識するようにアップグレードされるまで、8.0 より前のクライアントが 8.0 サーバーに接続できるようになります。ただし、この設定は、長期または永続的なソリューションとしてではなく、一時的なものとして表示する必要があります。これは、この設定を有効にして作成された新しいアカウントが、 `caching_sha2_password` によって提供される改善された認証セキュリティを予測するためです。

`caching_sha2_password` を使用すると、 `mysql_native_password` よりもセキュアなパスワードハッシュが提供されます (その結果、クライアント接続認証が改善されます)。ただし、既存の MySQL インストールに影響する可能性がある互換性の影響もあります:

- `caching_sha2_password` について知るために更新されていないクライアントおよびコネクタでは、 `caching_sha2_password` で認証されないアカウントを使用する場合でも、 `caching_sha2_password` でデフォルトの認証プラグインとして構成された MySQL 8.0 サーバーへの接続に問題が発生する可能性があります。この問題は、サーバーがデフォルトの認証プラグインの名前をクライアントに指定しているために発生します。クライアントまたはコネクタが、認識されないデフォルト認証プラグインを正常に処理しないクライアント/サーバープロトコル実装に基づいている場合、次のいずれかのエラーで失敗する可能性があります:

```
Authentication plugin 'caching_sha2_password' is not supported
```

```
Authentication plugin 'caching_sha2_password' cannot be loaded:
dlopen(/usr/local/mysql/lib/plugin/caching_sha2_password.so, 2):
image not found
```

```
Warning: mysqli_connect(): The server requested authentication
method unknown to the client [caching_sha2_password]
```

不明なデフォルト認証プラグインに対するサーバーからのリクエストを正常に処理するコネクタの記述の詳細は、[認証プラグインコネクタ - 書込みに関する考慮事項](#) を参照してください。

- `caching_sha2_password` で認証されるアカウントを使用するクライアントは、セキュアな接続 (TLS/SSL 資格証明、Unix ソケットファイルまたは共有メモリーを使用して TCP を使用して確立) または RSA キーペアを使用したパスワード交換をサポートする暗号化されていない接続を使用する必要があります。このセキュリティ要件は `mysql_native_password` には適用されないため、 `caching_sha2_password` への切替えには追加の構成が必要になる場合があります (セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」を参照)。ただし、MySQL 8.0 のクライアント接続では、デフォルトで TLS/SSL を使用することをお勧めします。そのため、そのプリファレンスにすでに準拠しているクライアントでは、追加の構成が必要ない場合があります。
- `caching_sha2_password` について知るために更新されていないクライアントおよびコネクタは、このプラグインが有効であると認識されないため、 `caching_sha2_password` で認証されるアカウントに接続できません。(これは、[認証プラグインクライアント/サーバーの互換性](#) で説明されているように、クライアント/サーバー認証プラグインの互換性要件が適用される方法の特定のインスタンスです。) この問題を回避するには、MySQL 8.0 以上から `libmysqlclient` に対してクライアントを再リンクするか、 `caching_sha2_password` を認識する更新済コネクタを取得します。
- `caching_sha2_password` は `libmysqlclient` クライアントライブラリのデフォルトの認証プラグインでもあるため、クライアントプログラムが `--default-auth=mysql_native_password` オプションで呼び出されないかぎり、MySQL 8.0 クライアントから `mysql_native_password` (以前のデフォルトの認証プラグイン) を使用するアカウントへの接続には、クライアント/サーバープロトコルで追加のラウンドトリップが必要になります。

8.0 MySQL より前のバージョンの `libmysqlclient` クライアントライブラリは、MySQL 8.0 サーバー (`caching_sha2_password` で認証されるアカウントを除く) に接続できます。つまり、 `libmysqlclient` に基づく 8.0 より前のクライアントも接続できる必要があります。例:

- `mysql` や `mysqladmin` などの標準 MySQL クライアントは `libmysqlclient` ベースです。
- Perl DBI 用の `DBD::mysql` ドライバは `libmysqlclient` ベースです。
- MySQL Connector/Python には、 `libmysqlclient` ベースの C 拡張モジュールがあります。これを使用するには、接続時に `use_pure=False` オプションを含めます。

既存の MySQL 8.0 インストールを MySQL 8.0.4 以上にアップグレードする場合、一部の古い `libmysqlclient` ベースのクライアントは、アップグレードによってインストールされた新しいクライアントライブラリを使用するため、動的にリンクされていれば「自動」をアップグレードできます。たとえば、Perl DBI 用の `DBD::mysql` ドライバで動的リンクが使用されている場合、MySQL 8.0.4 以上へのアップグレード後に `libmysqlclient` を使用できます。その結果は次のようになります:

- アップグレードの前に、 `DBD::mysql` を使用する DBI スクリプトは、 `caching_sha2_password` で認証されるアカウントを除き、MySQL 8.0 サーバーに接続できます。
- アップグレード後、同じスクリプトで `caching_sha2_password` アカウントも使用できるようになります。

ただし、8.0.4 より前の MySQL 8.0 インストールの `libmysqlclient` インスタンスにはバイナリ互換性があるため、前述の結果になります: どちらの場合も、共有ライブラリのメジャーバージョン番号 21 が使用されます。MySQL 5.7 以前から `libmysqlclient` にリンクされているクライアントの場合、バイナリ互換性のない異なるバージョン番号を持つ共有ライブラリにリンクします。この場合、MySQL 8.0 サーバーおよび `caching_sha2_password` アカウントとの完全な互換性を確保するために、8.0.4 以上の `libmysqlclient` に対してクライアントを再コンパイルする必要があります。

MySQL Connector/J 5.1 は 8.0.8 を介して MySQL 8.0 サーバーに接続できますが、 `caching_sha2_password` で認証されるアカウントは例外です。(Connector/J 8.0.9 以上が `caching_sha2_password` アカウントに接続するために必要です。)

`libmysqlclient` 以外のクライアント/サーバープロトコルの実装を使用するクライアントは、新しい認証プラグインを認識する新しいバージョンにアップグレードする必要がある場合があります。たとえば、PHP では、MySQL の接続性は通常、 `mysqlnd` に基づいており、 `caching_sha2_password` については現在認識されていません。更新されたバージョンの `mysqlnd` が使用可能になるまで、PHP クライアントが MySQL 8.0 に接続できるようにするには、前述のように、 `mysql_native_password` にデフォルトの認証プラグインとして戻すようにサーバーを再構成します。

クライアントまたはコネクタがデフォルトの認証プラグインを明示的に指定するオプションをサポートしている場合は、それを使用して `caching_sha2_password` 以外のプラグインに名前を付けます。例:

- 一部の MySQL クライアントは、 `--default-auth` オプションをサポートしています。(`mysql` や `mysqladmin` などの標準 MySQL クライアントはこのオプションをサポートしていますが、それを使用せずに 8.0 サーバーに正常に接続

できます。ただし、他のクライアントも同様のオプションをサポートする場合があります。その場合は、試行する価値があります。)

- `libmysqlclient` C API を使用するプログラムでは、`MYSQL_DEFAULT_AUTH` オプションを指定して `mysql_options()` 関数をコールできます。
- クライアント/サーバープロトコルのネイティブ Python 実装を使用する MySQL Connector/Python スクリプトでは、`auth_plugin` 接続オプションを指定できます。(または、`auth_plugin` を必要とせずに MySQL 8.0 サーバーに接続できる Connector/Python C 拡張機能を使用します。)

caching_sha2_password-Compatible クライアントおよびコネクタ

`caching_sha2_password` について知るために更新されたクライアントまたはコネクタが使用可能な場合は、`caching_sha2_password` をデフォルトの認証プラグインとして構成された MySQL 8.0 サーバーに接続するときに互換性を確保するための最善の方法です。

次のクライアントおよびコネクタは、`caching_sha2_password` をサポートするようにアップグレードされています:

- MySQL 8.0 (8.0.4 以上) の `libmysqlclient` クライアントライブラリ。 `mysql` や `mysqladmin` などの標準 MySQL クライアントは `libmysqlclient` ベースであるため、互換性もあります。
- MySQL 5.7 (5.7.23 以上) の `libmysqlclient` クライアントライブラリ。 `mysql` や `mysqladmin` などの標準 MySQL クライアントは `libmysqlclient` ベースであるため、互換性もあります。
- MySQL Connector/C++ 1.1.11 以上または 8.0.7 以上。
- MySQL Connector/J 8.0.9 以上。
- MySQL Connector/NET 8.0.10 以上 (クラシック MySQL プロトコルを使用)。
- MySQL Connector/Node.js 8.0.9 以上。
- PHP: X DevAPI PHP 拡張機能 (`mysql_xdevapi`) は、`caching_sha2_password` をサポートしています。

PHP: PDO_MySQL および `ext/mysql` 拡張機能は `caching_sha2_password` をサポートしていません。また、7.2.4 より前の 7.1.16 および PHP 7.2 より前の PHP バージョンで使用すると、`caching_sha2_password` が使用されていない場合でも `default_authentication_plugin=caching_sha2_password` との接続に失敗します。

caching_sha2_password および root 管理アカウント

MySQL 8.0 へのアップグレードの場合、`'root'@'localhost'` 管理アカウントのプラグインを含め、認証プラグインの既存のアカウントは変更されません。

MySQL 8.0 の新規インストールでは、(セクション 2.10.1 「データディレクトリの初期化」の手順を使用して) データディレクトリを初期化すると、`'root'@'localhost'` アカウントが作成され、そのアカウントはデフォルトで `caching_sha2_password` を使用します。したがって、データディレクトリの初期化後にサーバーに接続するには、`caching_sha2_password` をサポートするクライアントまたはコネクタを使用する必要があります。これを実行できるが、インストール後に `root` アカウントで `mysql_native_password` を使用する場合は、MySQL をインストールし、通常どおりにデータディレクトリを初期化します。次に、`root` としてサーバーに接続し、次のように `ALTER USER` を使用してアカウント認証プラグインおよびパスワードを変更します:

```
ALTER USER 'root'@'localhost'  
  IDENTIFIED WITH mysql_native_password  
  BY 'password';
```

使用するクライアントまたはコネクタが `caching_sha2_password` をまだサポートしていない場合は、アカウントが作成されるとすぐに `root` アカウントを `mysql_native_password` に関連付ける変更済データディレクトリ初期化プロシージャを使用できます。これを行うには、次のいずれかの方法を使用します:

- `--initialize` または `--initialize-insecure` とともに `--default-authentication-plugin=mysql_native_password` オプションを指定します。
- オプションファイルで `default_authentication_plugin` を `mysql_native_password` に設定し、`--initialize` または `--initialize-insecure` とともに `--defaults-file` オプションを使用してそのオプションファイルに名前を付けます。(この

場合、後続のサーバー起動にそのオプションファイルを引き続き使用すると、`default_authentication_plugin` 設定をオプションファイルから削除しないかぎり、`caching_sha2_password` ではなく `mysql_native_password` で新しいアカウントが作成されます。)

caching_sha2_password とレプリケーション

すべてのサーバーが MySQL 8.0.4 以上にアップグレードされているレプリケーションシナリオでは、ソースサーバーへのレプリカ接続で、`caching_sha2_password` で認証されるアカウントを使用できます。このような接続の場合、`caching_sha2_password` で認証されるアカウントを使用する他のクライアントと同じ要件が適用されます: セキュアな接続または RSA ベースのパスワード交換を使用します。

ソースレプリカレプリケーションのために `caching_sha2_password` アカウントに接続するには:

- 次の `CHANGE MASTER TO` オプションのいずれかを使用します:

```
MASTER_SSL = 1
GET_MASTER_PUBLIC_KEY = 1
MASTER_PUBLIC_KEY_PATH='path to RSA public key file'
```

- または、サーバーの起動時に必要なキーが指定されている場合は、RSA 公開キー関連オプションを使用できます。

グループレプリケーション用の `caching_sha2_password` アカウントに接続するには:

- OpenSSL を使用して構築された MySQL の場合は、次のいずれかのシステム変数を設定します:

```
SET GLOBAL group_replication_recovery_use_ssl = ON;
SET GLOBAL group_replication_recovery_get_public_key = 1;
SET GLOBAL group_replication_recovery_public_key_path = 'path to RSA public key file';
```

- または、サーバーの起動時に必要なキーが指定されている場合は、RSA 公開キー関連オプションを使用できます。

構成の変更

- 互換性のない変更: MySQL ストレージエンジンは独自のパーティショニングハンドラを提供する役割を果たし、MySQL サーバーは一般的なパーティショニングサポートを提供しなくなりました。MySQL 8.0 でサポートされているネイティブパーティショニングハンドラを提供するストレージエンジンは、`InnoDB` および `NDB` のみです。ほかのストレージエンジンを使用するパーティション化されたテーブルは、`InnoDB` または `NDB` に変換するか、サーバーをアップグレードする前にパーティション化を削除するために変更する必要があります。それ以外の場合は、あとで使用できません。

`MyISAM` テーブルの `InnoDB` への変換の詳細は、[セクション15.6.1.5「MyISAM から InnoDB へのテーブルの変換」](#)を参照してください。

このようなサポートがないストレージエンジンを使用してパーティション化されたテーブルになるテーブル作成ステートメントは、MySQL 8.0 のエラー (`ER_CHECK_NOT_IMPLEMENTED`) で失敗します。 `mysqldump` を使用して MySQL 5.7 (またはそれ以前) で作成されたダンプファイルから MySQL 8.0 サーバーにデータベースをインポートする場合は、パーティションテーブルを作成するすべてのステートメントで、パーティション化への参照を削除するか、ストレージエンジンを `InnoDB` として指定するか、デフォルトで `InnoDB` として設定できるようにして、サポートされていないストレージエンジンも指定しないようにする必要があります。

注記

[セクション2.11.5「アップグレード用のインストールの準備」](#) で提供されている手順では、MySQL 8.0 にアップグレードする前に変更する必要があるパーティションテーブルを識別する方法について説明します。

詳細は、[セクション24.6.2「ストレージエンジンに関連するパーティショニング制限」](#)を参照してください。

- 互換性のない変更: いくつかのサーバーエラーコードが使用されておらず、削除されています (リストについては、[MySQL 8.0 で削除された機能](#)を参照)。特にテストするアプリケーションは更新する必要があります。
- 重要な変更: デフォルトの文字セットが `latin1` から `utf8mb4` に変更されました。次のシステム変数が影響を受けません:

- `character_set_server` および `character_set_database` システム変数のデフォルト値が `latin1` から `utf8mb4` に変更されました。
- `collation_server` および `collation_database` システム変数のデフォルト値が `latin1_swedish_ci` から `utf8mb4_0900_ai_ci` に変更されました。

その結果、明示的な文字セットと照合順序が指定されていないかぎり、新しいオブジェクトのデフォルトの文字セットと照合順序は以前とは異なります。これには、データベースおよびその内部のオブジェクト (テーブル、ビュー、ストアドプログラムなど) が含まれます。以前のデフォルトが使用されていたと仮定すると、これらを保持する方法の 1 つは、`my.cnf` ファイル内の次の行を使用してサーバーを起動することです:

```
[mysqld]
character_set_server=latin1
collation_server=latin1_swedish_ci
```

レプリケートされた設定では、MySQL 5.7 から 8.0 にアップグレードする場合、アップグレードする前にデフォルトの文字セットを MySQL 5.7 で使用されている文字セットに戻すことをお勧めします。アップグレードの完了後、デフォルトの文字セットを `utf8mb4` に変更できます。

- 互換性のない変更: MySQL 8.0.11 の時点では、サーバーの初期化時に使用された設定とは異なる `lower_case_table_names` 設定でサーバーを起動することは禁止されています。様々なデータディクショナリテーブルのフィールドで使用される照合は、サーバーの初期化時に定義された `lower_case_table_names` 設定に基づいており、異なる設定でサーバーを再起動すると、識別子の順序付けおよび比較方法に矛盾が生じるため、制限が必要です。

サーバーの変更

- MySQL 8.0.11 では、ユーザーアカウントの非権限特性を変更するための `GRANT` ステートメントの使用、`NO_AUTO_CREATE_USER` SQL モード、`PASSWORD()` 関数、`old_passwords` システム変数など、アカウント管理に関連するいくつかの非推奨機能が削除されました。

これらの削除された機能を参照するステートメントの MySQL 5.7 から 8.0 へのレプリケーションによって、レプリケーションが失敗する可能性があります。MySQL 8.0 で削除された機能で説明されているように、削除された機能のいずれかを使用するアプリケーションを改訂して回避し、可能な場合は代替機能を使用する必要があります。

MySQL 8.0 での起動の失敗を回避するには、MySQL オプションファイルの `sql_mode` システム変数設定から `NO_AUTO_CREATE_USER` のインスタンスを削除します。

ストアドプログラム定義に `NO_AUTO_CREATE_USER` SQL モードを含むダンプファイルを MySQL 8.0 サーバーにロードすると、障害が発生します。MySQL 5.7.24 および MySQL 8.0.13 の時点で、`mysqldump` はストアドプログラム定義から `NO_AUTO_CREATE_USER` を削除します。以前のバージョンの `mysqldump` で作成されたダンプファイルは、`NO_AUTO_CREATE_USER` のインスタンスを削除するために手動で変更する必要があります。

- MySQL 8.0.11 では、これらの非推奨の互換性 SQL モードは削除されました: `DB2`, `MAXDB`, `MSSQL`, `MYSQL323`, `MYSQL40`, `ORACLE`, `POSTGRES`, `NO_FIELD_OPTIONS`, `NO_KEY_OPTIONS`, `NO_TABLE_OPTIONS`。これらを `sql_mode` システム変数に割り当てたり、`mysqldump --compatible` オプションの許可された値として使用したりすることはできなくなりました。

`MAXDB` を削除すると、`CREATE TABLE` または `ALTER TABLE` の `TIMESTAMP` データ型は `DATETIME` として扱われなくなります。

削除された SQL モードを参照するステートメントの MySQL 5.7 から 8.0 へのレプリケーションによって、レプリケーションが失敗する可能性があります。これには、現在の `sql_mode` 値に削除されたモードのいずれかが含まれているときに実行されるストアドプログラム (ストアドプロシージャとストアドファンクション、トリガーおよびイベント) の `CREATE` ステートメントのレプリケーションが含まれます。削除されたモードのいずれかを使用するアプリケーションは、それらを回避するために改訂する必要があります。

- MySQL 8.0.3 の時点では、空間データ型で `SRID` 属性を使用して、カラムに格納されている値の空間参照システム (SRS) を明示的に指定できます。セクション 11.4.1 「空間データ型」を参照してください。

明示的な `SRID` 属性を持つ空間カラムは、`SRID` 制限付きです: カラムはその ID を持つ値のみを取り、そのカラムの `SPATIAL` インデックスはオプティマイザによって使用される対象になります。オプティマイザは、`SRID` 属性の

ない空間カラムの **SPATIAL** インデックスを無視します。 [セクション8.3.3「SPATIAL インデックス最適化」](#) を参照してください。SRID 制限のない空間カラムの **SPATIAL** インデックスを最適化で考慮する場合は、そのような各カラムを変更する必要があります:

- カラム内のすべての値が同じ SRID を持つことを確認します。ジオメトリカラム `col_name` に含まれる SRID を確認するには、次のクエリーを使用します:

```
SELECT DISTINCT ST_SRID(col_name) FROM tbl_name;
```

クエリーで複数の行が返された場合、カラムには SRID の混在が含まれます。その場合は、その内容を変更して、すべての値が同じ SRID を持つようにします。

- 明示的な **SRID** 属性を持つようにカラムを再定義します。
- SPATIAL** インデックスを再作成します。
- 正確な操作を実行する関数の **ST_**接頭辞、または最小境界矩形に基づいて操作を実行する関数の **MBR** 接頭辞を実装する空間関数名前スペースの変更により、MySQL 8.0.0 でいくつかの空間関数が削除されました。生成されたカラム定義で削除された空間関数を使用すると、アップグレードが失敗する可能性があります。アップグレードする前に、削除された空間関数に対して `mysqlcheck --check-upgrade` を実行し、見つかったものを **ST_** または **MBR** の名前付き置換に置き換えます。削除された空間関数のリストは、[MySQL 8.0 で削除された機能](#) を参照してください。
- BACKUP_ADMIN** 権限は、MySQL 8.0.3 以上へのインプレースアップグレードの実行時に、**RELOAD** 権限を持つユーザーに自動的に付与されます。
- MySQL 8.0.13 からは、一時テーブルの処理方法における行ベースまたは混合レプリケーションモードとステートメントベースのレプリケーションモードの違いにより、実行時のバイナリロギング形式の切り替えに新しい制限があります。
 - セッションにオープン一時テーブルがある場合、`SET @@SESSION.binlog_format` は使用できません。
 - いずれかのレプリケーションチャンネルにオープン一時テーブルがある場合、`SET @@global.binlog_format` および `SET @@persist.binlog_format` は使用できません。`SET @@persist_only.binlog_format` は、レプリケーションチャンネルにオープン一時テーブルがある場合に許可されます。これは、**PERSIST** とは異なり、**PERSIST_ONLY** ではランタイムグローバルシステム変数値が変更されないためです。
 - レプリケーションチャンネルアプライヤが実行されている場合、`SET @@global.binlog_format` および `SET @@persist.binlog_format` は使用できません。これは、変更がレプリケーションチャンネルで有効になるのは、そのアプライヤが再起動されたときのみであり、そのときにレプリケーションチャンネルにオープン一時テーブルがある可能性があるためです。この動作は、以前よりも制限されています。`SET @@persist_only.binlog_format` は、レプリケーションチャンネルアプライヤが実行されている場合に許可されます。

InnoDB の変更点

- InnoDB** システムテーブルに基づく **INFORMATION_SCHEMA** ビューは、データディクショナリテーブルの内部システムビューに置き換えられました。影響を受ける **InnoDB INFORMATION_SCHEMA** ビューの名前が変更されました:

表 2.16 名前が変更された InnoDB 情報スキーマビュー

旧名称	新規名
<code>INNODB_SYS_COLUMNS</code>	<code>INNODB_COLUMNS</code>
<code>INNODB_SYS_DATAFILES</code>	<code>INNODB_DATAFILES</code>
<code>INNODB_SYS_FIELDS</code>	<code>INNODB_FIELDS</code>
<code>INNODB_SYS_FOREIGN</code>	<code>INNODB_FOREIGN</code>
<code>INNODB_SYS_FOREIGN_COLS</code>	<code>INNODB_FOREIGN_COLS</code>
<code>INNODB_SYS_INDEXES</code>	<code>INNODB_INDEXES</code>
<code>INNODB_SYS_TABLES</code>	<code>INNODB_TABLES</code>

旧名称	新規名
INNODB_SYS_TABLESPACES	INNODB_TABLESPACES
INNODB_SYS_TABLESTATS	INNODB_TABLESTATS
INNODB_SYS_VIRTUAL	INNODB_VIRTUAL

MySQL 8.0.3 以上にアップグレードした後、以前の `INNO_DB_INFORMATION_SCHEMA` ビュー名を参照するスクリプトを更新します。

- MySQL にバンドルされている「[zlib ライブラリ](#)」バージョンは、バージョン 1.2.3 からバージョン 1.2.11 に引き上げられました。

zlib 1.2.11 の `zlib compressBound()` 関数は、zlib version 1.2.3 の場合よりも一定のバイト長を圧縮するために必要なバッファサイズの見積りを少し大きく返します。`compressBound()` 関数は、圧縮 InnoDB テーブルの作成時または圧縮 InnoDB テーブルへの行の挿入および更新時に許可される最大行サイズを決定する InnoDB 関数によってコールされます。その結果、以前のリリースで成功した最大行サイズに非常に近い行サイズの `CREATE TABLE ... ROW_FORMAT=COMPRESSED`、`INSERT` および `UPDATE` 操作が失敗する可能性があります。この問題を回避するには、アップグレードの前に、MySQL 8.0 テストインスタンスで大きい行を含む圧縮 InnoDB テーブルの `CREATE TABLE` ステートメントをテストします。

- `--innodb-directories` 機能の導入により、絶対パスまたはデータディレクトリ外の場所で作成された file-per-table および一般的なテーブルスペースファイルの場所を `innodb_directories` 引数値に追加する必要があります。そうしないと、InnoDB はリカバリ中にこれらのファイルを検出できません。テーブルスペースファイルの場所を表示するには、`INFORMATION_SCHEMA.FILES` テーブルをクエリーします:

```
SELECT TABLESPACE_NAME, FILE_NAME FROM INFORMATION_SCHEMA.FILES \G
```

- undo ログはシステムテーブルスペースに存在できなくなりました。MySQL 8.0 では、undo ログはデフォルトで 2 つの undo テーブルスペースに存在します。詳細は、[セクション 15.6.3.4 「undo テーブルスペース」](#) を参照してください。

MySQL 5.7 から MySQL 8.0 にアップグレードする場合、MySQL 5.7 インスタンスに存在する undo テーブルスペースは削除され、2 つの新しいデフォルト undo テーブルスペースに置き換えられます。デフォルトの undo テーブルスペースは、`innodb_undo_directory` 変数で定義された場所に作成されます。`innodb_undo_directory` 変数が定義されていない場合、undo テーブルスペースはデータディレクトリに作成されます。MySQL 5.7 から MySQL へのアップグレード 8.0 では、MySQL 5.7 インスタンスの undo テーブルスペースが空であることを保証する低速な停止が必要であり、安全に削除できます。

以前の MySQL 8.0 リリースから MySQL 8.0.14 以降にアップグレードする場合、2 より大きい `innodb_undo_tablespaces` 設定の結果としてアップグレード前インスタンスに存在する undo テーブルスペースは、ユーザー定義 undo テーブルスペースとして扱われ、アップグレード後に `ALTER UNDO TABLESPACE` および `DROP UNDO TABLESPACE` 構文を使用してそれぞれ非アクティブ化および削除できます。MySQL 8.0 リリースシリーズ内のアップグレードでは、常に低速な停止が必要になるわけではありません。つまり、既存の undo テーブルスペースに undo ログが含まれている可能性があります。したがって、既存の undo テーブルスペースはアップグレードプロセスによって削除されません。

- 互換性のない変更: MySQL 8.0.17 では、`CREATE TABLESPACE ... ADD DATAFILE` 句で循環ディレクトリ参照は許可されません。たとえば、次のステートメントの循環ディレクトリ参照 (`././`) は使用できません:

```
CREATE TABLESPACE ts1 ADD DATAFILE ts1.ibd 'any_directory/./ts1.ibd';
```

Linux には制限の例外があり、前述のディレクトリがシンボリックリンクの場合は循環ディレクトリ参照が許可されます。たとえば、`any_directory` がシンボリックリンクの場合、前述の例のデータファイルパスは許可されます。(データファイルパスを `././` で始めることはできます。)

アップグレードの問題を回避するには、MySQL 8.0.17 以上にアップグレードする前に、テーブルスペースデータファイルパスから循環ディレクトリ参照を削除します。テーブルスペースパスを調べるには、`INFORMATION_SCHEMA.INNODB_DATAFILES` テーブルをクエリーします。

- MySQL 8.0.14 で導入された回帰のため、パーティションテーブルおよび `lower_case_table_names=1` を含むインスタンスで、MySQL 8.0.14 から MySQL 8.0.16 への MySQL 5.7 または MySQL 8.0 リリースより前の大/小文字を区別するファイルシステムでのインプレースアップグレードが失敗しました。失敗の原因は、パーティションテーブ

ルのファイル名に関連する大/小文字の不一致の問題でした。回帰を導入した修正が元に戻され、MySQL 8.0.14 より前の MySQL 5.7 または MySQL 8.0 リリースから MySQL 8.0.17 へのアップグレードが正常に機能するようになりました。ただし、回帰は MySQL 8.0.14、8.0.15 および 8.0.16 リリースには引き続き存在します。

パーティション化されたテーブルが存在し、`lower_case_table_names=1` がある場合、バイナリまたはパッケージを MySQL 8.0.17 にアップグレードした後にサーバーを起動すると、大/小文字を区別するファイルシステムでの MySQL 8.0.14、8.0.15 または 8.0.16 から MySQL 8.0.17 へのインプレースアップグレードが次のエラーで失敗します:

```
Upgrading from server version version_number with
partitioned tables and lower_case_table_names == 1 on a case sensitive file
system may cause issues, and is therefore prohibited. To upgrade anyway, restart
the new server version with the command line option 'upgrade=FORCE'. When
upgrade is completed, please execute 'RENAME TABLE part_table_name
TO new_table_name; RENAME TABLE new_table_name
TO part_table_name;' for each of the partitioned tables.
Please see the documentation for further information.
```

MySQL 8.0.17 へのアップグレード時にこのエラーが発生した場合は、次の回避策を実行します:

1. `--upgrade=force` を使用してサーバーを再起動し、アップグレード操作を強制的に続行します。
2. 小文字のパーティション名デリミタ (`#p#`または`#sp#`)を使用して、パーティションテーブルのファイル名を識別します:

```
mysql> SELECT FILE_NAME FROM INFORMATION_SCHEMA.FILES WHERE FILE_NAME LIKE '%#p#%' OR FILE_NAME LIKE '%#sp#%';
```

3. 識別されたファイルごとに、一時名を使用して関連するテーブルの名前を変更し、テーブルの名前を元の名前に戻します。

```
mysql> RENAME TABLE table_name TO temporary_table_name;
mysql> RENAME TABLE temporary_table_name TO table_name;
```

4. パーティションテーブルファイル名の小文字のパーティション名デリミタがないことを確認します (空の結果セットが返されます)。

```
mysql> SELECT FILE_NAME FROM INFORMATION_SCHEMA.FILES WHERE FILE_NAME LIKE '%#p#%' OR FILE_NAME LIKE '%#sp#%';
Empty set (0.00 sec)
```

5. 名前を変更した各テーブルで `ANALYZE TABLE` を実行して、`mysql.innodb_index_stats` テーブルおよび `mysql.innodb_table_stats` テーブルのオプティマイザ統計を更新します。

MySQL 8.0.14、8.0.15 および 8.0.16 リリースには回帰がまだ存在するため、MySQL 8.0.14、8.0.15 または 8.0.16 から MySQL 8.0.17 へのパーティションテーブルのインポートは、`lower_case_table_names=1` の大/小文字が区別されるファイルシステムではサポートされていません。これを試行すると、「「テーブルのテーブルスペースがありません」」エラーが発生します。

- MySQL では、テーブルパーティションのテーブルスペース名およびファイル名を作成する際にデリミタ文字列が使用されます。次に示すように、「`#p#`」デリミタ文字列はパーティション名の前に、「`#sp#`」デリミタ文字列はサブパーティション名の前にあります:

```
schema_name.table_name#p#partition_name#sp#subpartition_name
table_name#p#partition_name#sp#subpartition_name.ibd
```

従来、Linux などの大/小文字を区別するファイルシステムではデリミタ文字列は大文字 (`#P#`および`#SP#`)、Windows などの大/小文字を区別しないファイルシステムでは小文字 (`#p#`および`#sp#`) になっていました。MySQL 8.0.19 の時点では、デリミタ文字列はすべてのファイルシステムで小文字になります。この変更により、大/小文字を区別するファイルシステムと大/小文字を区別しないファイルシステムの間でデータディレクトリを移行する際の問題が回避されます。大文字のデリミタ文字列は使用されなくなりました。

また、ユーザー指定のパーティション名またはサブパーティション名 (大文字または小文字で指定可能) に基づいて生成されたパーティションテーブルスペース名およびファイル名は、大/小文字を区別しないように `lower_case_table_names` 設定に関係なく小文字で生成 (および内部的に格納) されるようになりました。たとえ

ば、テーブルパーティションが `PART_1` という名前で作成された場合、テーブルスペース名とファイル名は小文字で生成されます:

```
schema_name.table_name#p#part_1
table_name#p#part_1.ibd
```

アップグレード時に、MySQL は必要に応じてチェックおよび変更を行います:

- 小文字のデリミタとパーティション名を確認するために、ディスクおよびデータディクショナリのパーティションファイル名。
- 以前のバグ修正で発生した関連する問題のデータディクショナリ内のパーティションメタデータ。
- 以前のバグ修正で導入された関連する問題の `InnoDB` 統計データ。

テーブルスペースのインポート操作中に、小文字のデリミタとパーティション名を確認するために、必要に応じてディスク上のパーティションテーブルスペースファイル名がチェックおよび変更されます。

- MySQL 8.0.21 では、テーブルスペースデータファイルが不明なディレクトリに存在することが判明した場合、起動時または MySQL 5.7 からのアップグレード時にエラーログに警告が書き込まれます。既知のディレクトリは、`datadir`、`innodb_data_home_dir` および `innodb_directories` 変数で定義されているディレクトリです。ディレクトリを既知にするには、`innodb_directories` 設定に追加します。ディレクトリを既知にすると、リカバリ中にデータファイルを検出できるようになります。詳細は、[クラッシュリカバリ中のテーブルスペースの検出](#)を参照してください。

SQL の変更

- 互換性のない変更: MySQL 8.0.13 では、`GROUP BY` 句の非推奨の `ASC` 修飾子または `DESC` 修飾子は削除されています。以前に `GROUP BY` ソートに依存していたクエリーでは、以前の MySQL バージョンとは異なる結果が生成される場合があります。特定のソート順序を生成するには、`ORDER BY` 句を指定します。

`GROUP BY` 句に `ASC` または `DESC` 修飾子を使用する MySQL 8.0.12 以下のクエリーおよびストアドプログラム定義を修正する必要があります。そうしないと、MySQL 8.0.13 以上のレプリカサーバーにレプリケートされる可能性があるため、MySQL 8.0.13 以上へのアップグレードが失敗することがあります。

- MySQL 5.7 では予約されていなかった一部のキーワードが、MySQL 8.0 では予約されている場合があります。 [セクション9.3「キーワードと予約語」](#)を参照してください。これにより、以前に識別子として使用された単語が不正になる可能性があります。影響を受けるステートメントを修正するには、識別子の引用符を使用します。 [セクション9.2「スキーマオブジェクト名」](#)を参照してください。
- アップグレード後、目的の最適化戦略を達成するためにヒントがまだ必要であることを確認するために、アプリケーションコードで指定されたオプティマイザヒントをテストすることをお勧めします。オプティマイザの拡張により、特定のオプティマイザヒントが不要になる場合があります。場合によっては、不要なオプティマイザヒントに対応することもあります。
- 互換性のない変更: MySQL 5.7 では、`CONSTRAINT symbol` 句なしで `InnoDB` テーブルの `FOREIGN KEY` 定義を指定するか、`symbol` なしで `CONSTRAINT` キーワードを指定すると、`InnoDB` で生成された制約名が使用されます。この動作は MySQL 8.0 で変更され、`InnoDB` では生成された名前ではなく `FOREIGN KEY index_name` 値が使用されます。制約名はスキーマ (データベース) ごとに一意である必要があるため、スキーマごとに一意ではない外部キーインデックス名が原因でエラーが発生しました。このようなエラーを回避するために、MySQL 8.0.16 では新しい制約のネーミング動作が元に戻され、`InnoDB` では生成された制約名が再度使用されます。

`InnoDB` との一貫性のために、`CONSTRAINT symbol` 句が指定されていない場合、または `CONSTRAINT` キーワードが `symbol` なしで指定されている場合は、MySQL 8.0.16 以上に基づく `NDB` リリースで生成された制約名が使用されます。MySQL 5.7 以前の MySQL 8.0 リリースに基づく `NDB` リリースでは、`FOREIGN KEY index_name` 値が使用されていました。

前述の変更により、以前の外部キー制約のネーミング動作に依存するアプリケーションの非互換性が生じる場合があります。

2.11.5 アップグレード用のインストールの準備

最新の MySQL 8.0 リリースにアップグレードする前に、次に説明する事前チェックを実行して、現在の MySQL 5.7 または MySQL 8.0 サーバーインスタンスのアップグレード準備状況を確認します。 そうしないと、アップグレードプロセスが失敗する可能性があります。

ヒント

MySQL サーバーインスタンスのアップグレード準備ができているかどうかを確認できる [MySQL Shell upgrade checker utility](#) の使用を検討してください。 MySQL Server 8.0.11 から現在の MySQL Shell リリース番号と一致する MySQL Server リリース番号まで、アップグレード先のターゲット MySQL Server リリースを選択できます。 アップグレードチェックユーティリティは、指定されたターゲットリリースに関連する自動チェックを実行し、手動で行う必要がある関連チェックをさらにアドバイスします。 アップグレードチェッカーは、MySQL 5.7 および 8.0 のすべての GA リリースで機能します。 MySQL Shell のインストール手順は、[here](#) にあります。

予備チェック:

1. 次の問題は存在してはいけません:

- 廃止されたデータ型または関数を使用するテーブルは存在しない必要があります。

テーブルに 5.6.4 より前の形式 (`TIME`、`DATETIME` および `TIMESTAMP` カラムで小数秒精度がサポートされていない) の古い時間カラムが含まれている場合、MySQL 8.0 へのインプレースアップグレードはサポートされません。 テーブルで古い一時カラム形式を使用している場合は、MySQL 8.0 へのインプレースアップグレードを試行する前に、`REPAIR TABLE` を使用してそれらをアップグレードします。 詳細は、[MySQL 5.7 Reference Manual](#) の [Server Changes](#) を参照してください。

- 孤立した `.frm` ファイルは存在できません。
- トリガーには、欠落している定義者、空の定義者、または無効な作成コンテキスト (`SHOW TRIGGERS` または `INFORMATION_SCHEMA TRIGGERS` テーブルで表示される `character_set_client`、`collation_connection`、`Database Collation` 属性で示される) を指定できません。 このようなトリガーをダンプしてリストアし、問題を修正する必要があります。

これらの問題を確認するには、次のコマンドを実行します:

```
mysqlcheck -u root -p --all-databases --check-upgrade
```

`mysqlcheck` からエラーが報告された場合は、問題を修正します。

2. ネイティブパーティション化がサポートされていないストレージエンジンを使用するパーティションテーブルは存在しない必要があります。 このようなテーブルを識別するには、次のクエリーを実行します:

```
SELECT TABLE_SCHEMA, TABLE_NAME  
FROM INFORMATION_SCHEMA.TABLES  
WHERE ENGINE NOT IN ('innodb', 'ndbcluster')  
AND CREATE_OPTIONS LIKE '%partitioned%';
```

クエリーによってレポートされるテーブルは、`InnoDB` を使用するように変更するか、非パーティション化する必要があります。 テーブルストレージエンジンを `InnoDB` に変更するには、次のステートメントを実行します:

```
ALTER TABLE table_name ENGINE = INNODB;
```

`MyISAM` テーブルの `InnoDB` への変換の詳細は、[セクション15.6.1.5「MyISAM から InnoDB へのテーブルの変換」](#) を参照してください。

パーティションテーブルを非パーティションテーブルにするには、次のステートメントを実行します:

```
ALTER TABLE table_name REMOVE PARTITIONING;
```

3. 一部のキーワードは、以前に予約されていない MySQL 8.0 で予約されている場合があります。 [セクション 9.3「キーワードと予約語」](#) を参照してください。 これにより、以前に識別子として使用された単語が不正にな

る可能性があります。影響を受けるステートメントを修正するには、識別子の引用符を使用します。 [セクション 9.2「スキーマオブジェクト名」](#)を参照してください。

4. MySQL 5.7 `mysql` システムデータベースには、MySQL 8.0 データディクショナリで使用されるテーブルと同じ名前のテーブルは存在しない必要があります。これらの名前を持つテーブルを識別するには、次のクエリーを実行します:

```
SELECT TABLE_SCHEMA, TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE LOWER(TABLE_SCHEMA) = 'mysql'
and LOWER(TABLE_NAME) IN
(
'catalogs',
'character_sets',
'check_constraints',
'collations',
'column_statistics',
'column_type_elements',
'columns',
'dd_properties',
'events',
'foreign_key_column_usage',
'foreign_keys',
'index_column_usage',
'index_partitions',
'index_stats',
'indexes',
'parameter_type_elements',
'parameters',
'resource_groups',
'routines',
'schemata',
'st_spatial_reference_systems',
'table_partition_values',
'table_partitions',
'table_stats',
'tables',
'tablespace_files',
'tablespace_files',
'tablespace_files',
'triggers',
'view_routine_usage',
'view_table_usage'
);
```

クエリーによってレポートされたテーブルは、削除または名前変更する必要があります (`RENAME TABLE` を使用)。これには、影響を受けるテーブルを使用するアプリケーションへの変更が含まれる場合もあります。

5. 64 文字を超える外部キー制約名を持つテーブルは使用できません。制約名が長すぎるテーブルを識別するには、次のクエリーを使用します:

```
SELECT TABLE_SCHEMA, TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_NAME IN
(SELECT LEFT(SUBSTR(ID,INSTR(ID,'/')+1),
INSTR(SUBSTR(ID,INSTR(ID,'/')+1),'_ibfk_')-1)
FROM INFORMATION_SCHEMA.INNODB_SYS_FOREIGN
WHERE LENGTH(SUBSTR(ID,INSTR(ID,'/')+1))>64);
```

制約名が 64 文字を超えるテーブルの場合は、制約を削除し、64 文字を超えない制約名で追加しなおします (`ALTER TABLE` を使用)。

6. `sql_mode` システム変数で定義されている廃止された SQL モードは使用できません。廃止された SQL モードを使用しようとすると、MySQL 8.0 が起動しなくなります。廃止された SQL モードを使用するアプリケーションは、回避するために修正する必要があります。MySQL 8.0 で削除された SQL モードの詳細は、[サーバーの変更](#)を参照してください。
7. 明示的に定義されたカラム名が 64 文字を超えるビューは存在しない必要があります (MySQL 5.7 では、カラム名が 255 文字以内のビューが許可されていました)。アップグレードエラーを回避するには、アップグレード前にこのようなビューを変更する必要があります。現在、カラム名が 64 文字を超えるビューを識別する唯一の方法

は、[SHOW CREATE VIEW](#) を使用してビュー定義を検査することです。 [INFORMATION_SCHEMA.VIEWS](#) テーブルをクエリーすることで、ビュー定義を検査することもできます。

- 個々の [ENUM](#) または [SET](#) カラム要素の長さが 255 文字または 1020 バイトを超えるテーブルまたはストアードプロシージャは使用できません。MySQL 8.0 より前は、[ENUM](#) または [SET](#) のカラム要素の最大長は 64K でした。MySQL 8.0 では、個々の [ENUM](#) または [SET](#) カラム要素の最大文字長は 255 文字で、最大バイト長は 1020 バイトです。(1020 バイトの制限では、マルチバイト文字セットがサポートされます)。MySQL 8.0 にアップグレードする前に、新しい制限を超える [ENUM](#) または [SET](#) のカラム要素を変更します。 そうしないと、アップグレードはエラーで失敗します。

- MySQL 8.0.13 以上にアップグレードする前に、システムテーブルスペースおよび一般テーブルスペースを含む共有 [InnoDB](#) テーブルスペースにテーブルパーティションが存在していない必要があります。 [INFORMATION_SCHEMA](#) をクエリーして、共有テーブルスペースのテーブルパーティションを識別します:

MySQL 5.7 からアップグレードする場合は、次のクエリーを実行します:

```
SELECT DISTINCT NAME, SPACE, SPACE_TYPE FROM INFORMATION_SCHEMA.INNODB_SYS_TABLES
WHERE NAME LIKE '%#P#%' AND SPACE_TYPE NOT LIKE 'Single';
```

以前の MySQL 8.0 リリースからアップグレードする場合は、次のクエリーを実行します:

```
SELECT DISTINCT NAME, SPACE, SPACE_TYPE FROM INFORMATION_SCHEMA.INNODB_TABLES
WHERE NAME LIKE '%#P#%' AND SPACE_TYPE NOT LIKE 'Single';
```

[ALTER TABLE ... REORGANIZE PARTITION](#) を使用して、テーブルパーティションを共有テーブルスペースから [file-per-table](#) テーブルスペースに移動します:

```
ALTER TABLE table_name REORGANIZE PARTITION partition_name
INTO (partition_definition TABLESPACE=innodb_file_per_table);
```

- [GROUP BY](#) 句に [ASC](#) または [DESC](#) 修飾子を使用する MySQL 8.0.12 以下のクエリーおよびストアードプログラム定義は使用できません。 そうしないと、MySQL 8.0.13 以上のレプリカサーバーにレプリケートされる可能性があるため、MySQL 8.0.13 以上へのアップグレードが失敗することがあります。 追加の詳細については、[SQL の変更](#) を参照してください。

- MySQL 5.7 インストールでは、MySQL 8.0 でサポートされていない機能を使用しないでください。 ここでの変更はインストール固有である必要がありますが、次の例は検索する内容の種類を示しています:

一部のサーバー起動オプションおよびシステム変数は、MySQL 8.0 で削除されました。 [Features Removed in MySQL 8.0](#) および [Server and Status Variables and Options Added, Deprecated, or Removed in MySQL 8.0](#) を参照してください。 これらのいずれかを使用する場合、アップグレードには構成の変更が必要です。

例: データディクショナリはデータベースオブジェクトに関する情報を提供するため、サーバーはデータディレクトリ内のディレクトリ名をチェックしてデータベースを検索しなくなりました。 したがって、[--ignore-db-dir](#) オプションは無関係であり、削除されています。 これを処理するには、起動構成から [--ignore-db-dir](#) のインスタンスを削除します。 また、MySQL 8.0 にアップグレードする前に、名前付きデータディレクトリのサブディレクトリを削除または移動します。(または、8.0 サーバーでこれらのディレクトリをデータベースとしてデータディクショナリに追加し、[DROP DATABASE](#) を使用してこれらの各データベースを削除します。)

- アップグレード時に [lower_case_table_names](#) 設定を 1 に変更する場合は、アップグレード前にスキーマ名とテーブル名が小文字であることを確認してください。 そうしないと、スキーマ名またはテーブル名の英文字と小文字の不一致が原因で障害が発生する可能性があります。 次のクエリーを使用して、大文字を含むスキーマおよびテーブル名を確認できます:

```
mysql> SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME != LOWER(TABLE_NAME) AND TABLE_TYPE = 'BASE TABLE';
mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA WHERE SCHEMA_NAME != LOWER(SCHEMA_NAME);
```

MySQL 8.0.19 では、[lower_case_table_names=1](#) の場合、アップグレードプロセスによってテーブル名およびスキーマ名がチェックされ、すべての文字が小文字であることが確認されます。 テーブル名またはスキーマ名に大文字が含まれていることが判明した場合、アップグレードプロセスはエラーで失敗します。

注記

アップグレード時に [lower_case_table_names](#) 設定を変更することはお勧めしません。

前述のいずれかの問題が原因で MySQL 8.0 へのアップグレードが失敗した場合、サーバーはすべての変更をデータディレクトリに戻します。この場合は、すべての redo ログファイルを削除し、既存のデータディレクトリで MySQL 5.7 サーバーを再起動してエラーに対処します。redo ログファイル (`ib_logfile*`) は、デフォルトでは MySQL データディレクトリにあります。エラーが修正されたら、アップグレードを再試行する前に、(`innodb_fast_shutdown=0` を設定して) 低速な停止を実行します。

2.11.6 Unix/Linux での MySQL バイナリまたはパッケージベースのインストールのアップグレード

このセクションでは、Unix/Linux で MySQL バイナリおよびパッケージベースのインストールをアップグレードする方法について説明します。インプレースおよび論理アップグレード方法について説明します。

- [インプレースアップグレード](#)
- [論理アップグレード](#)
- [MySQL クラスタのアップグレード](#)

インプレースアップグレード

インプレースアップグレードには、古い MySQL サーバーの停止、古い MySQL バイナリまたはパッケージの新しいバイナリまたはパッケージへの置換、既存のデータディレクトリでの MySQL の再起動、およびアップグレードが必要な既存のインストールの残りの部分のアップグレードが含まれます。アップグレードが必要なものの詳細は、[セクション 2.11.3 「MySQL のアップグレードプロセスの内容」](#) を参照してください。

注記

最初に複数の RPM パッケージをインストールして作成したインストールをアップグレードする場合は、一部だけでなく、すべてのパッケージをアップグレードします。たとえば、以前にサーバーおよびクライアントの RPM をインストールした場合は、サーバーの RPM だけをアップグレードすることはしないでください。

一部の Linux プラットフォームでは、RPM または Debian パッケージからの MySQL インストールに、MySQL サーバーの起動と停止を管理するための `systemd` サポートが含まれています。これらのプラットフォームでは、`mysqld_safe` はインストールされません。このような場合は、次の手順で使用する方法ではなく、`systemd` を使用してサーバーを起動および停止します。[セクション 2.5.9 「systemd を使用した MySQL Server の管理」](#) を参照してください。

MySQL クラスタインストールへのアップグレードについては、[MySQL クラスタのアップグレード](#) も参照してください。

インプレースアップグレードを実行するには:

1. [セクション 2.11.1 「始める前に」](#) の情報を確認します。
2. [セクション 2.11.5 「アップグレード用のインストールの準備」](#) で事前チェックを完了して、インストールのアップグレード準備が整っていることを確認します。
3. XA トランザクションを `InnoDB` で使用する場合は、アップグレードの前に `XA RECOVER` を実行して、コミットされていない XA トランザクションを確認します。結果が返された場合は、`XA COMMIT` または `XA ROLLBACK` ステートメントを発行して XA トランザクションをコミットまたはロールバックします。
4. 暗号化された `InnoDB` テーブルスペースがある場合は、次のステートメントを実行してキーリングマスターキーをローテーションします:

```
ALTER INSTANCE ROTATE INNODB MASTER KEY;
```

5. `innodb_fast_shutdown` を 2 (コールドシャットダウン) に設定して構成した MySQL サーバーを通常実行する場合は、次のいずれかのステートメントを実行して、高速または低速なシャットダウンを実行するように構成します:

```
SET GLOBAL innodb_fast_shutdown = 1; -- fast shutdown
```



```
SET GLOBAL innodb_fast_shutdown = 0; -- slow shutdown
```

高速停止または低速停止では、**InnoDB** は undo ログおよびデータファイルを、リリース間でファイル形式が異なる場合に処理できる状態のままにします。

- 古い MySQL サーバーを停止します。例:

```
mysqladmin -u root -p shutdown
```

- MySQL バイナリまたはパッケージをアップグレードします。バイナリインストールをアップグレードする場合は、新しい MySQL バイナリ配布パッケージを解凍します。[配布の取得とアンパック](#)を参照してください。パッケージベースのインストールの場合は、新しいパッケージをインストールします。

- 既存のデータディレクトリを使用して、MySQL 8.0 サーバーを起動します。例:

```
mysqld_safe --user=mysql --datadir=/path/to/existing-datadir &
```

暗号化された **InnoDB** テーブルスペースがある場合は、`--early-plugin-load` オプションを使用してキーリングプラグインをロードします。

MySQL 8.0 サーバーを起動すると、データディクショナリテーブルが存在するかどうか自動的に検出されます。そうでない場合、サーバーはデータディレクトリにそれらを作成し、メタデータを移入してから、通常の起動シーケンスに進みます。このプロセス中、サーバーは、データベース、テーブルスペース、システムテーブルとユーザーテーブル、ビュー、ストアドプログラム (ストアドプロシージャと関数、トリガーおよびイベントスケジューライベント) など、すべてのデータベースオブジェクトのメタデータをアップグレードします。サーバーは、以前にメタデータ記憶域に使用されていたファイルも削除します。たとえば、MySQL 5.7 から MySQL 8.0 にアップグレードした後、テーブルに `.frm` ファイルがなくなった場合があります。

このステップが失敗すると、サーバーはすべての変更をデータディレクトリに戻します。この場合、すべての redo ログファイルを削除し、同じデータディレクトリで MySQL 5.7 サーバーを起動して、エラーの原因を修正する必要があります。次に、5.7 サーバーの別の低速シャットダウンを実行し、MySQL 8.0 サーバーを起動して再試行します。

- 前のステップでは、サーバーは必要に応じてデータディクショナリをアップグレードします。ここで、残りのアップグレード操作を実行する必要があります:

- MySQL 8.0.16 の時点では、サーバーは前のステップの一部としてこれを行い、新しい権限または機能を利用できるように、`mysql` システムデータベースで MySQL 5.7 と MySQL 8.0 の間で必要な変更を行います。また、パフォーマンススキーマ、`INFORMATION_SCHEMA` および `sys` データベースを MySQL 8.0 用に最新の状態にし、すべてのユーザーデータベースで現在のバージョンの MySQL との非互換性を調べます。
- MySQL 8.0.16 より前は、サーバーは前のステップのデータディクショナリのみをアップグレードします。MySQL 8.0 サーバーが正常に起動したら、`mysql_upgrade` を実行して残りのアップグレードタスクを実行します:

```
mysql_upgrade -u root -p
```

次に、MySQL サーバーを停止して再起動し、システムテーブルに対する変更が有効になるようにします。例:

```
mysqladmin -u root -p shutdown  
mysqld_safe --user=mysql --datadir=/path/to/existing-datadir &
```

(前のステップで)MySQL 8.0 サーバーを初めて起動すると、アップグレードされていないテーブルに関するメッセージがエラーログに記録される場合があります。`mysql_upgrade` が正常に実行された場合、サーバーの再起動時にこのようなメッセージは表示されません。

注記

アップグレードプロセスでは、タイムゾーンテーブルの内容はアップグレードされません。アップグレードの手順については、[セクション5.1.15「MySQL Server でのタイムゾーンのサポート」](#)を参照してください。

アップグレードプロセスで `mysql_upgrade` (MySQL 8.0.16 より前) が使用されている場合、このプロセスではヘルプテーブルの内容もアップグレードされません。この場合のアップグ

ロード手順については、[セクション5.1.17「サーバー側ヘルプのサポート」](#)を参照してください。

論理アップグレード

論理アップグレードには、[mysqldump](#) や [mysqlpump](#) などのバックアップまたはエクスポートユーティリティを使用した古い MySQL インスタンスからの SQL のエクスポート、新しい MySQL サーバーのインストール、および新しい MySQL インスタンスへの SQL の適用が含まれます。アップグレードが必要なものの詳細は、[セクション2.11.3「MySQL のアップグレードプロセスの内容」](#)を参照してください。

注記

一部の Linux プラットフォームでは、RPM または Debian パッケージからの MySQL インストールに、MySQL サーバーの起動と停止を管理するための `systemd` サポートが含まれています。これらのプラットフォームでは、`mysqld_safe` はインストールされません。このような場合は、次の手順で使用する方法ではなく、`systemd` を使用してサーバーを起動および停止します。[セクション2.5.9「systemd を使用した MySQL Server の管理」](#)を参照してください。

警告

以前の MySQL リリースから抽出された SQL を新しい MySQL リリースに適用すると、新しい機能、変更された機能、非推奨になった機能または削除された機能によって導入された非互換性が原因でエラーが発生する場合があります。したがって、以前の MySQL リリースから抽出された SQL では、論理アップグレードを有効にするために変更が必要になる場合があります。

最新の MySQL 8.0 リリースにアップグレードする前に非互換性を識別するには、[セクション2.11.5「アップグレード用のインストールの準備」](#)で説明されているステップを実行します。

論理アップグレードを実行するには:

1. [セクション2.11.1「始める前に」](#)の情報を確認します。
2. 以前の MySQL インストールから既存のデータをエクスポートします:

```
mysqldump -u root -p  
--add-drop-table --routines --events  
--all-databases --force > data-for-upgrade.sql
```

注記

データベースにストアードプログラムが含まれている場合は、`mysqldump` で `--routines` および `--events` オプションを使用します (前述を参照)。`--all-databases` オプションには、システムテーブルを保持する `mysql` データベースを含む、ダンプ内のすべてのデータベースが含まれます。

重要

生成されたカラムを含むテーブルがある場合は、MySQL 5.7.9 以上で提供される `mysqldump` ユーティリティを使用してダンプファイルを作成します。以前のリリースで提供されていた `mysqldump` ユーティリティでは、生成されたカラム定義に不正な構文が使用されています (Bug #20769542)。`INFORMATION_SCHEMA.COLUMNS` テーブルを使用して、生成されたカラムを持つテーブルを識別できます。

3. 古い MySQL サーバーを停止します。例:

```
mysqladmin -u root -p shutdown
```

4. MySQL 8.0 をインストールします。インストールの手順については、[第2章「MySQL のインストールとアップグレード」](#)を参照してください。

5. [セクション2.10.1「データディレクトリの初期化」](#)の説明に従って、新しいデータディレクトリを初期化します。例:

```
mysqld --initialize --datadir=/path/to/8.0-datadir
```

画面に表示された一時'`root@localhost`'パスワードをコピーするか、後で使用するためにエラーログに書き込みます。

6. 新しいデータディレクトリを使用して、MySQL 8.0 サーバーを起動します。例:

```
mysqld_safe --user=mysql --datadir=/path/to/8.0-datadir &
```

7. `root` パスワードをリセットします:

```
shell> mysql -u root -p  
Enter password: **** <- enter temporary root password
```

```
mysql> ALTER USER USER() IDENTIFIED BY 'your new password';
```

8. 以前に作成したダンプファイルを新しい MySQL サーバーにロードします。例:

```
mysql -u root -p --force < data-for-upgrade.sql
```

注記

ダンプファイルにシステムテーブルが含まれている場合、GTID がサーバー (`gtid_mode=ON`) で有効になっているときにダンプファイルをロードすることはお勧めしません。 `mysqldump` は、非トランザクション MyISAM ストレージエンジンを使用するシステムテーブルに対して DML 命令を発行します。GTID が有効になっている場合、この組み合わせは許可されません。GTID が有効になっているサーバーから GTID が有効になっている別のサーバーにダンプファイルをロードすると、異なるトランザクション識別子が生成されることにも注意してください。

9. 残りのアップグレード操作を実行します:

- MySQL 8.0.16 以上では、サーバーを停止し、`--upgrade=FORCE` オプションを使用して再起動して残りのアップグレードタスクを実行します:

```
mysqladmin -u root -p shutdown  
mysqld_safe --user=mysql --datadir=/path/to/8.0-datadir --upgrade=FORCE &
```

`--upgrade=FORCE` を使用して再起動すると、新しい権限または機能を利用できるように、`mysql` システムスキーマで MySQL 5.7 と MySQL 8.0 の間で必要な変更がサーバーによって行われます。また、パフォーマンススキーマ、`INFORMATION_SCHEMA`、および `sys` スキーマを MySQL 8.0 用に最新の状態にし、すべてのユーザースキーマで現在のバージョンの MySQL との非互換性を調べます。

- MySQL 8.0.16 より前は、`mysql_upgrade` を実行して残りのアップグレードタスクを実行します:

```
mysql_upgrade -u root -p
```

次に、MySQL サーバーを停止して再起動し、システムテーブルに対する変更が有効になるようにします。例:

```
mysqladmin -u root -p shutdown  
mysqld_safe --user=mysql --datadir=/path/to/8.0-datadir &
```

注記

アップグレードプロセスでは、タイムゾーンテーブルの内容はアップグレードされません。アップグレードの手順については、[セクション5.1.15「MySQL Server でのタイムゾーンのサポート」](#)を参照してください。

アップグレードプロセスで `mysql_upgrade` (MySQL 8.0.16 より前) が使用されている場合、このプロセスではヘルプテーブルの内容もアップグレードされません。この場合のアップグ

ロード手順については、[セクション5.1.17「サーバー側ヘルプのサポート」](#)を参照してください。

注記

MySQL 5.7 `mysql` スキーマを含むダンプファイルをロードすると、使用されなくなった2つのテーブルが再作成されます: `event` および `proc`。(対応する MySQL 8.0 テーブルは `events` および `routines` で、どちらもデータディクショナリテーブルであり、保護されています。) アップグレードが成功したことを確認したら、次の SQL ステートメントを実行して `event` テーブルおよび `proc` テーブルを削除できます:

```
DROP TABLE mysql.event;  
DROP TABLE mysql.proc;
```

MySQL クラスタのアップグレード

このセクションの情報は、MySQL クラスタをアップグレードする場合に使用するための、[インプレースアップグレード](#)で説明されているインプレースアップグレード手順に関連しています。

MySQL 8.0.16 では、MySQL クラスタのアップグレードは、通常の順序付きステップに従って定期的なローリングアップグレードとして実行できます:

1. MGM ノードのアップグレード。
2. データノードを一度に1つずつアップグレードします。
3. API ノードを一度に1つずつアップグレードします (MySQL サーバーを含む)。

データディクショナリのアップグレードとシステムテーブルのアップグレードは分離されているため、各ノードをアップグレードする方法は MySQL 8.0.16 の前とほとんど同じです。個々の `mysqld` をアップグレードするには、2つのステップがあります:

1. データディクショナリをインポートします。

`--upgrade=MINIMAL` オプションを使用して新しいサーバーを起動し、データディクショナリをアップグレードしますが、システムテーブルはアップグレードしません。これは基本的に、`mysql_upgrade` を起動せずにサーバーを起動する pre-MySQL 8.0.16 アクションと同じです。

このフェーズを完了するには、MySQL サーバーを `NDB` に接続する必要があります。 `NDB` テーブルまたは `NDBINFO` テーブルが存在し、サーバーがクラスタに接続できない場合は、次のエラーメッセージで終了します:

```
Failed to Populate DD tables.
```

2. システムテーブルをアップグレードします。

MySQL 8.0.16 より前は、DBA は `mysql_upgrade` クライアントを起動してシステムテーブルをアップグレードします。MySQL 8.0.16 の時点で、サーバーはこのアクションを実行: システムテーブルをアップグレードするには、`--upgrade=MINIMAL` オプションを指定せずに個々の `mysqld` を再起動します。

2.11.7 MySQL Yum リポジトリを使用する MySQL のアップグレード

サポートされている Yum ベースのプラットフォーム (リストについては、[セクション2.5.1「MySQL Yum リポジトリを使用して MySQL を Linux にインストールする」](#)を参照) では、MySQL のインプレースアップグレード (古いバージョンを置き換えてから、古いデータファイルを使用して新しいバージョンを実行) を MySQL Yum リポジトリで行えます。

メモ

- MySQL に対して更新を実行する前に、[セクション2.11「MySQL のアップグレード」](#)の手順に慎重に従ってください。そこで議論されている手順のうち、更新の前にデータベースのバックアップを取ることが特に重要です。
- 次の手順では、MySQL を MySQL Yum リポジトリとともにインストールしたか、「[MySQL Developer Zone MySQL ダウンロードページ](#)」から直接ダウンロードした

RPM パッケージを使用してインストールしたことを前提としています。そうでない場合は、[Replacing a Third-Party Distribution of MySQL Using the MySQL Yum Repository](#) の手順に従います。

ターゲットシリーズの選択

デフォルトでは、MySQL Yum リポジトリは、インストール時に選択したリリースシリーズの最新バージョンに MySQL を更新します (詳細は [リリースシリーズの選択](#) を参照)。つまり、5.7.x インストールは 8.0.x リリースに自動的に更新されません。別のリリースシリーズに更新するには、最初に選択したシリーズ (デフォルトまたは自分自身) のサブリポジトリを無効にし、ターゲットシリーズのサブリポジトリを有効にする必要があります。これを行うには、[リリースシリーズの選択](#) に示されている一般的な手順を参照してください。MySQL 5.7 から 8.0 にアップグレードするには、[リリースシリーズの選択](#) に示されているステップの `reverse` を実行し、MySQL 5.7 シリーズのサブリポジトリを無効にして、MySQL 8.0 シリーズに対して有効にします。

原則として、あるリリースシリーズから別のものにアップグレードするには、シリーズをスキップせずに次のシリーズに移動します。たとえば、現在 MySQL 5.6 を実行しており、8.0 にアップグレードする場合は、まず MySQL 5.7 にアップグレードしてから 8.0 にアップグレードします。

重要

MySQL 5.7 から 8.0 へのアップグレードに関する重要な情報は、[Upgrading from MySQL 5.7 to 8.0](#) を参照してください。

MySQL のアップグレード

dnf 対応でないプラットフォームの場合は、次のコマンドを使用して MySQL とそのコンポーネントをアップグレードします:

```
sudo yum update mysql-server
```

dnf 対応のプラットフォームの場合:

```
sudo dnf upgrade mysql-server
```

または、システム上のすべてを更新するように Yum に指示することで、MySQL を更新することもできます。これにはかなり時間がかかる可能性があります。dnf 対応でないプラットフォームの場合:

```
sudo yum update
```

dnf 対応のプラットフォームの場合:

```
sudo dnf upgrade
```

MySQL の再起動

MySQL Server は、Yum による更新後必ず再起動します。MySQL 8.0.16 より前は、サーバーの再起動後に `mysql_upgrade` を実行して、古いデータとアップグレードされたソフトウェア間の非互換性を確認し、場合によっては解決します。`mysql_upgrade` は、他の機能も実行します。詳細は、[セクション4.4.5「mysql_upgrade — MySQL テーブルのチェックとアップグレード」](#) を参照してください。MySQL 8.0.16 では、サーバーは以前に `mysql_upgrade` によって処理されたすべてのタスクを実行するため、このステップは必要ありません。

特定のコンポーネントのみを更新することもできます。次のコマンドを使用して、MySQL コンポーネントにインストールされているすべてのパッケージをリストします (dnf 対応システムの場合は、コマンド内の `yum` を `dnf` に置き換えます):

```
sudo yum list installed | grep "mysql"
```

選択したコンポーネントのパッケージ名を特定した後、次のコマンドを使用してパッケージを更新し、`package-name` をパッケージの名前に置き換えます。dnf 対応でないプラットフォームの場合:

```
sudo yum update package-name
```

dnf 対応プラットフォームの場合:


```
sudo dnf upgrade package-name
```

共有クライアントライブラリのアップグレード

Yum リポジトリを使用して MySQL を更新したあとも、古いバージョンの共有クライアントライブラリを使用してコンパイルされたアプリケーションは機能するはずですが、

アプリケーションを再コンパイルし、それらを更新されたライブラリに動的にリンクする場合:新しいライブラリと古いライブラリ間のシンボルのバージョンに違いまたは追加がある新しいバージョンの共有ライブラリ (たとえば、新しい標準 8.0 共有クライアントライブラリと、Linux 配布ソフトウェアリポジトリによってネイティブに出荷された共有ライブラリの古いバージョンまたは可変バージョンの共有ライブラリ、または他のソースから出荷されたアプリケーション) の場合、更新された新しい共有ライブラリを使用してコンパイルされたアプリケーションでは、それらの更新されたライブラリがアプリケーションがデプロイされているシステムで必要になります。予期したとおり、これらのライブラリが配置されていない場合、共有ライブラリを必要とするアプリケーションは失敗します。このため、共有ライブラリのパッケージは、必ず MySQL からそれらのシステムにデプロイしてください。これを行うには、MySQL Yum リポジトリをシステムに追加し ([MySQL Yum リポジトリの追加](#) を参照)、[追加の MySQL 製品およびコンポーネントの Yum でのインストール](#) に記載されている手順を使用して最新の共有ライブラリをインストールします。

2.11.8 MySQL APT リポジトリを使用する MySQL のアップグレード

Debian および Ubuntu プラットフォームで、MySQL とそのコンポーネントのインプレースアップグレードを実行するには、MySQL APT リポジトリを使用します。「[A Quick Guide to Using the MySQL APT Repository](#)」の、「[Upgrading MySQL with the MySQL APT Repository](#)」を参照してください。

2.11.9 MySQL SLES リポジトリを含む MySQL のアップグレード

SUSE Linux Enterprise Server (SLES) プラットフォームで、MySQL とそのコンポーネントのインプレースアップグレードを実行するには、MySQL SLES リポジトリを使用します。「[A MySQL SLES リポジトリの使用法のクイックガイド](#)」の「[MySQL SLES リポジトリを含む MySQL のアップグレード](#)」を参照してください。

2.11.10 Windows 上の MySQL をアップグレードする

Windows で MySQL をアップグレードするには、次の 2 つの方法があります:

- [Using MySQL Installer](#)
- [Using the Windows ZIP archive distribution](#)

選択する方法は、既存のインストールの実行方法によって異なります。先に進む前に、Windows に固有でない MySQL のアップグレードに関する追加情報を [セクション 2.11 「MySQL のアップグレード」](#) で確認してください。

注記

どちらの方法を選択しても、アップグレードを実行する前に必ず現在の MySQL インストールをバックアップしてください。[セクション 7.2 「データベースバックアップ方法」](#) を参照してください。

非 GA リリース間 (または非 GA リリースから GA リリースへ) のアップグレードはサポートされていません。GA 以外のリリースでは大幅な開発変更が行われ、サーバーの起動時に互換性の問題や問題が発生する可能性があります。

注記

MySQL Installer では、コミュニティリリースと商業リリース間のアップグレードはサポートされていません。このタイプのアップグレードが必要な場合は、[ZIP archive](#) アプローチを使用して実行します。

MySQL Installer を含む MySQL のアップグレード

MySQL Installer を使用したアップグレードの実行は、現在のサーバーインストールが実行され、アップグレードが現在のリリースシリーズ内にある場合に最適な方法です。MySQL Installer では、5.7 から 8.0 へのアップグレードなど、リリースシリーズ間のアップグレードはサポートされておらず、アップグレードを求めるアップグレードイン

ジケータも提供されていません。リリースシリーズ間のアップグレード手順については、[Windows ZIP ディストリビューションを使用した MySQL のアップグレード](#) を参照してください。

MySQL Installer を使用してアップグレードを実行するには:

1. MySQL Installer を起動します。
2. ダッシュボードから Catalog をクリックして、カタログへの最新の変更をダウンロードします。インストールされたサーバーは、ダッシュボードでサーバーのバージョン番号の横に矢印が表示されている場合にのみアップグレードできます。
3. 「アップグレード」をクリックします。新しいバージョンを持つすべての製品がリストに表示されます。

注記

MySQL Installer では、同じリリースシリーズのマイルストーンリリース (事前リリース) のサーバーアップグレードオプションの選択が解除されます。また、アップグレードがサポートされていないことを示す警告が表示され、続行のリスクが識別され、アップグレードを手動で実行するステップのサマリーが表示されます。サーバーのアップグレードを再選択して、自分のリスクで続行できます。

4. この時点で他の製品をアップグレードする予定がないかぎり、MySQL サーバー製品以外のすべての製品の選択を解除し、次へをクリックします。
5. Execute をクリックしてダウンロードを開始します。ダウンロードが終了したら、次へをクリックしてアップグレード操作を開始します。

MySQL 8.0.16 以上へのアップグレードでは、システムテーブルのアップグレードチェックおよびプロセスをスキップするオプションが表示される場合があります。このオプションの詳細は、[Important server upgrade conditions](#) を参照してください。

6. サーバーを構成します。

Windows ZIP ディストリビューションを使用した MySQL のアップグレード

Windows ZIP アーカイブ配布を使用してアップグレードを実行するには:

1. MySQL の最新の Windows ZIP アーカイブディストリビューションを <https://dev.mysql.com/downloads/> からダウンロードします。
2. サーバーが実行中の場合は停止します。サーバーをサービスとしてインストールしている場合は、コマンドプロンプトから次のコマンドでサービスを停止します。

```
C:\> SC STOP mysqlid_service_name
```

または、`NET STOP mysqlid_service_name` を使用します。

MySQL Server をサービスとして実行していない場合は、`mysqladmin` を使用して停止します。たとえば、MySQL 5.7 から 8.0 にアップグレードする前に、`mysqladmin` を MySQL 5.7 から次のように使用します。

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.7\bin\mysqladmin" -u root shutdown
```

注記

MySQL の `root` ユーザーアカウントにパスワードが設定されている場合は、`mysqladmin` を `-p` オプションを使って起動し、要求されたらパスワードを入力します。

3. ZIP アーカイブを抽出します。既存の MySQL インストール (通常は `C:\mysql` にあります) を上書きするか、`C:\mysql8` などの別のディレクトリにインストールできます。既存のインストールの上書きをお勧めしています。
4. サーバーを再起動します。たとえば、MySQL をサービスとして実行する場合は `SC START mysqlid_service_name` または `NET START mysqlid_service_name` コマンドを使用し、それ以外の場合は `mysqlid` を直接起動します。

- MySQL 8.0.16 より前は、管理者として `mysql_upgrade` を実行してテーブルを確認し、必要に応じて修復し、権限付与テーブルが変更されている場合は更新して、新しい機能を利用できるようにしてください。 [セクション 4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」](#) を参照してください。 MySQL 8.0.16 では、サーバーは以前に `mysql_upgrade` によって処理されたすべてのタスクを実行するため、このステップは必要ありません。
- エラーが発生する場合は [セクション 2.3.5 「Microsoft Windows MySQL Server インストールのトラブルシューティング」](#) を参照します。

2.11.11 MySQL の Docker インストールのアップグレード

MySQL の Docker インストールをアップグレードするには、[MySQL Server コンテナのアップグレード](#) を参照してください。

2.11.12 アップグレードのトラブルシューティング

- テーブルの `.frm` ファイルと InnoDB データディクショナリの間で MySQL 5.7 インスタンスのスキーマが一致しないと、MySQL 8.0 へのアップグレードが失敗する可能性があります。このような不一致は、`.frm` ファイルの破損が原因である可能性があります。この問題に対処するには、アップグレードを再試行する前に、影響を受けるテーブルをダンプしてリストアします。
- 新しい `mysqld` サーバーが起動しないなどの問題が発生した場合は、以前のインストールの古い `my.cnf` ファイルがないことを確認します。これは `--print-defaults` オプション (たとえば、`mysqld --print-defaults`) で確認できます。このコマンドがプログラム名以外の何かを表示した場合、サーバーあるいはクライアントオペレーションに影響を及ぼすアクティブな `my.cnf` ファイルがあることとなります。
- アップグレード後に、コンパイル済みのクライアントプログラムに [Commands out of sync](#) または予測外のコアダンプなどの問題が発生した場合は、プログラムのコンパイル時に、古いヘッダーファイルまたはライブラリファイルを使用した可能性があります。この場合、`mysql.h` ファイルおよび `libmysqlclient.a` ライブラリの日付をチェックして、それらが新しい MySQL 配布のものであることを確認します。そうでない場合には、プログラムを新しいヘッダーおよびライブラリで再度コンパイルします。ライブラリのメジャーバージョン番号が (たとえば、`libmysqlclient.so.20` から `libmysqlclient.so.21` に) 変更された場合、共有クライアントライブラリに対してコンパイルされたプログラムに再コンパイルが必要になることもあります。
- ユーザー定義関数 (UDF) を所定の名前で作成したあとで、MySQL を同じ名前の組み込み関数を実装した新しいバージョンにアップグレードした場合、その UDF はアクセスできなくなります。これを修正するには、[DROP FUNCTION](#) を使用して UDF を削除してから、[CREATE FUNCTION](#) を使用して競合しない別の名前で UDF を再作成します。新しいバージョンの MySQL が、既存のストアードファンクションと同名の組み込み関数を実装している場合にも、同じことが言えます。各種の関数への参照をサーバーが解釈する方法を記述したルールについては、[セクション 9.2.5 「関数名の構文解析と解決」](#) を参照してください。
- [セクション 2.11.5 「アップグレード用のインストールの準備」](#) で概説されている問題のいずれかが原因で MySQL 8.0 へのアップグレードが失敗した場合、サーバーはすべての変更をデータディレクトリに戻します。この場合は、すべての redo ログファイルを削除し、既存のデータディレクトリで MySQL 5.7 サーバーを再起動してエラーに対処します。redo ログファイル (`ib_logfile*`) は、デフォルトでは MySQL データディレクトリにあります。エラーが修正されたら、アップグレードを再試行する前に、(`innodb_fast_shutdown=0` を設定して) 低速な停止を実行します。

2.11.13 テーブルまたはインデックスの再作成または修復

このセクションでは、次の方法でテーブルまたはインデックスを再構築または修復する方法について説明します:

- MySQL によるデータ型または文字セットの処理方法の変更。たとえば、照合順序のエラーが修正され、照合順序を使用する文字カラムのインデックスを更新するためにテーブルの再構築が必要になる場合があります。
- [CHECK TABLE](#)、`mysqlcheck` または `mysql_upgrade` によって報告される必要なテーブルの修復またはアップグレード。

テーブルを再構築する方法には、次のものがあります:

- [ダンプおよびリロード方法](#)

- [ALTER TABLE メソッド](#)
- [REPAIR TABLE メソッド](#)

ダンプおよびリロード方法

バイナリ (インプレース) のアップグレードまたはダウングレード後に異なるバージョンの MySQL で処理できないためにテーブルを再構築する場合は、ダンプアンドリロード方法を使用する必要があります。アップグレードまたはダウングレードの前に、元のバージョンの MySQL を使用してテーブルをダンプします。次に、アップグレードまたはダウングレードのあとに、テーブルをリロードします。

インデックスを再構築する目的のためだけにダンプしてリロードする方法を使ってテーブルを再構築する場合は、アップグレードまたはダウングレードの前でもあとでもダンプを実行できます。その場合でも、リロードはあとで行う必要があります。

CHECK TABLE 操作でテーブルのアップグレードが必要であることが示されているために **InnoDB** テーブルを再構築する必要がある場合は、**mysqldump** を使用してダンプファイルを作成し、**mysql** を使用してファイルをリロードします。**CHECK TABLE** 操作で、破損があることが示されたり **InnoDB** が失敗したりする場合は、**innodb_force_recovery** オプションを使用して **InnoDB** を再起動する方法について、[セクション15.21.2「InnoDBのリカバリの強制的な実行」](#)を参照してください。**CHECK TABLE** が遭遇している問題のタイプを理解するには、[セクション13.7.3.2「CHECK TABLE ステートメント」](#)の **InnoDB** に関する注記を参照してください。

テーブルをダンプしてリロードすることによって再構築するには、**mysqldump** を使用してダンプファイルを作成し、**mysql** でファイルをリロードします。

```
mysqldump db_name t1 > dump.sql  
mysql db_name < dump.sql
```

単独のデータベース内のテーブルをすべて再構築する場合は、データベース名を、そのあとにテーブル名なしで指定します。

```
mysqldump db_name > dump.sql  
mysql db_name < dump.sql
```

すべてのデータベース内のすべてのテーブルをリロードするには、**--all-databases** オプションを使用してください。

```
mysqldump --all-databases > dump.sql  
mysql < dump.sql
```

ALTER TABLE メソッド

ALTER TABLE でテーブルを再構築する場合は、「null」変更を使用します。すなわち、すでに使用しているストレージエンジンを使用するように、テーブルを「変更」する **ALTER TABLE** ステートメントです。たとえば、**t1** が **InnoDB** テーブルである場合、次のステートメントを利用します。

```
ALTER TABLE t1 ENGINE = InnoDB;
```

ALTER TABLE ステートメントで指定するストレージエンジンがわからない場合は、**SHOW CREATE TABLE** を使用してテーブル定義を表示します。

REPAIR TABLE メソッド

REPAIR TABLE メソッドは、**MyISAM**、**ARCHIVE** および **CSV** テーブルにのみ適用できます。

テーブルチェック操作で破損があること、またはアップグレードが必要であることが示されている場合は、**REPAIR TABLE** を使用できます。たとえば、**MyISAM** テーブルを修復するには、次のステートメントを使用します。

```
REPAIR TABLE t1;
```

mysqlcheck --repair は、コマンド行で **REPAIR TABLE** ステートメントへのアクセスを提供します。**--databases** オプションまたは **--all-databases** オプションをそれぞれ使用して、特定のデータベースまたはすべてのデータベースのすべてのテーブルを修復できるため、テーブル修復の方法として、より便利な場合があります。

```
mysqlcheck --repair --databases db_name ...
```

```
mysqlcheck --repair --all-databases
```

2.11.14 MySQL データベースのほかのマシンへのコピー

データベースを異なるアーキテクチャー間で移動する必要がある場合、`mysqldump` を使用して SQL ステートメントを含むファイルを作成します。次にそのファイルを別のマシンに転送して `mysql` クライアントへの入力として扱います。

利用できるオプションを表示するには `mysqldump --help` を使用します。

注記

GTID がダンプ (`gtid_mode=ON`) を作成するサーバーで使用されている場合、デフォルトでは、`mysqldump` は `gtid_executed` セットの内容をダンプに含めて、これらを新しいマシンに転送します。この結果は、関係する MySQL Server のバージョンによって異なる場合があります。`mysqldump's --set-gtid-purged` オプションの説明を確認して、使用しているバージョンで何が起こるか、およびデフォルトの動作の結果が状況に適していない場合の動作の変更方法を確認します。

データベースを 2 つのマシンで間で移動するもっとも容易な (ただし、速くはない) 方法は、データベースを搭載したマシン上で次のコマンドを実行することです。

```
mysqladmin -h 'other_hostname' create db_name  
mysqldump db_name | mysql -h 'other_hostname' db_name
```

データベースをリモートマシンから速度の遅いネットワークを介してコピーするには、次のコマンドを使用できません。

```
mysqladmin create db_name  
mysqldump -h 'other_hostname' --compress db_name | mysql db_name
```

ダンプをファイルに保存して、そのファイルをターゲットマシンに転送し、そのファイルをそこのデータベースにロードすることもできます。たとえば、データベースをソースマシンの圧縮ファイルに次のようにダンプできます。

```
mysqldump --quick db_name | gzip > db_name.gz
```

データベースのコンテンツを含んだファイルをターゲットマシンに転送し、そこで次のコマンドを実行します。

```
mysqladmin create db_name  
gunzip < db_name.gz | mysql db_name
```

データベースの転送に `mysqldump` および `mysqlimport` を使用することもできます。大きなテーブルの場合、これは単に `mysqldump` を使用するよりも非常に速いです。次のコマンドで、`DUMPDIR` は `mysqldump` の出力の保存に使用されるディレクトリのフルパス名です。

最初に、その出力ファイルのディレクトリを作成してデータベースをダンプします。

```
mkdir DUMPDIR  
mysqldump --tab=DUMPDIR  
db_name
```

次に `DUMPDIR` ディレクトリのファイルをターゲットマシンの相当するディレクトリに転送して、そのファイルをその MySQL にロードします。

```
mysqladmin create db_name # create database  
cat DUMPDIR/*.sql | mysql db_name # create tables in database  
mysqlimport db_name  
DUMPDIR/*.txt # load data into tables
```

`mysql` データベースをコピーすることを忘れないでください。付与テーブルがそこに格納されているからです。新しいマシンで `mysql` データベースが用意できるまで、コマンドを MySQL `root` ユーザーとして実行しなければならない場合があります。

`mysql` データベースを新しいマシンにインポートしたら、`mysqladmin flush-privileges` を実行してサーバーに付与テーブルの情報をロードさせます。

2.12 MySQL のダウングレード

MySQL 8.0 から MySQL 5.7 へのダウングレード、または MySQL 8.0 リリースから以前の MySQL 8.0 リリースへのダウングレードはサポートされていません。サポートされている唯一の代替方法は、アップグレード前に作成したバックアップをリストアすることです。したがって、アップグレードプロセスを開始する前にデータをバックアップする必要があります。

2.13 Perl のインストールに関する注釈

PerlDBIモジュールはデータベースアクセスのための一般的なインタフェースを提供します。変更なしで多くの異なるデータベースエンジンで動作する DBI スクリプトを書くことができます。DBI を使用するには、DBI モジュール、および DataBase Driver (DBD) モジュールを、アクセスする各データベースサーバーのタイプごとにインストールする必要があります。MySQL の場合、このドライバは `DBD::mysql` モジュールです。

注記

MySQL の配布には Perl のサポートは含まれていません。必要なモジュールは、Unix の場合は <http://search.cpan.org> から、Windows では ActiveState ppm プログラムを使用して取得できます。次のセクションでその方法について説明します。

DBI/DBD インタフェースは Perl 5.6.0 を必要とし、5.6.1 以降が推奨されます。それより古いバージョンの Perl の場合、DBI は機能しません。`DBD::mysql` 4.009 以上を使用するようにしてください。それより前のバージョンは使用可能ですが、MySQL 8.0 の全機能をサポートしているわけではありません。

2.13.1 Unix に Perl をインストールする

MySQL の Perl のサポートには MySQL のクライアントプログラムサポート (ライブラリおよびヘッダーファイル) をインストールする必要があります。ほとんどのインストール方法で、必要なファイルがインストールされます。MySQL を RPM ファイルからインストールする場合、開発者 RPM もインストールしてください。クライアントプログラムはクライアント RPM にありますが、クライアントプログラムサポートは開発者 RPM にあります。

Perl のサポートに必要なファイルは、<http://search.cpan.org> の CPAN (Comprehensive Perl Archive Network) から入手できます。

Unix に Perl モジュールをインストールするには CPAN モジュールを使用するのがいちばん簡単です。例:

```
shell> perl -MCPAN -e shell
cpan> install DBI
cpan> install DBD::mysql
```

`DBD::mysql` インストールは多くのテストを実行します。これらのテストでは、デフォルトのユーザー名とパスワードを使用してローカルの MySQL サーバーに接続を試みます。(デフォルトのユーザー名は、Unix ではログイン名であり、Windows では ODBC です。デフォルトのパスワードは「パスワードなし」です)。サーバーにそれらの値で接続できない場合 (たとえば、アカウントにパスワードを設定している場合)、テストは失敗します。`force install DBD::mysql` を使用すると、失敗したテストを無視できます。

DBI には `Data::Dumper` モジュールが必要です。それはインストールされている場合があります。もしされていない場合、DBI をインストールする前にそれをインストールするようにしてください。

モジュールの配布を圧縮 tar アーカイブの形でダウンロードして、モジュールを手動でビルドすることもできます。たとえば、DBI 配布をアンパックしてビルドするには、次のような手順に従います。

1. 配布を現在のディレクトリにアンパックします。

```
shell> gunzip < DBI-VERSION.tar.gz | tar xvf -
```

このコマンドは、`DBI-VERSION` という名前のディレクトリを作成します。

2. アンパックした配布のトップレベルのディレクトリに場所を変更します。

```
shell> cd DBI-VERSION
```

3. 配布をビルドしてすべてをコンパイルします。

```
shell> perl Makefile.PL
shell> make
shell> make test
shell> make install
```

`make test` コマンドはモジュールが動作していることを確認するために重要です。 `DBD::mysql` のインストール中にそのコマンドを実行してインタフェースのコードを実行するには、MySQL サーバーが動作していなければなりません。 そうしないとそのテストは失敗します。

新しいリリースの MySQL をインストールする際は必ず `DBD::mysql` 配布を再構築して再インストールするのがいいでしょう。 これにより、MySQL クライアントライブラリの最新バージョンが正しくインストールされていることが保証されます。

Perl モジュールをシステムディレクトリにインストールするアクセス権がない場合、またはローカルの Perl をインストールする場合、次のリファレンスが有用でしょう。 <http://learn.perl.org/faq/perlfaq8.html#How-do-I-keep-my-own-module-library-directory->。

2.13.2 Windows に ActiveState Perl をインストールする

Windows 上で、MySQL `DBD` モジュールを ActiveState Perl でインストールするには次の手順に従います。

1. <http://www.activestate.com/Products/ActivePerl/> から ActiveState Perl を入手してインストールします。
2. コンソールウィンドウを開きます。
3. 必要に応じて `HTTP_proxy` 変数を設定します。 たとえば、次のような設定を試してみます。

```
C:\> set HTTP_proxy=my.proxy.com:3128
```

4. PPM プログラムを起動します。

```
C:\> C:\perl\bin\ppm.pl
```

5. まだ `DBI` をインストールしていない場合は、インストールします。

```
ppm> install DBI
```

6. これが成功したら、次のコマンドを実行します。

```
ppm> install DBD-mysql
```

この手順は、ActiveState Perl 5.6 以上で動作する必要があります。

手順が機能しない場合は、代わりに ODBC ドライバをインストールして ODBC から MySQL サーバーに接続するようにしてください。

```
use DBI;
$dbh= DBI->connect("DBI:ODBC:$dsn",$user,$password) ||
die "Got error $DBI::errstr when connecting to $dsn\n";
```

2.13.3 Perl DBI/DBD インタフェース使用の問題

Perl が `../mysql/mysql.so` モジュールを見つけることができない場合、問題はおそらく Perl が `libmysqlclient.so` 共有ライブラリの場所がわからないことです。 この問題は次の方法のいずれかで解決できるはずです。

- `libmysqlclient.so` をほかの共有ライブラリがある (おそらく `/usr/lib` あるいは `/lib`) ディレクトリにコピーします。
- `DBD::mysql` のコンパイルに使用される `-L` オプションを、`libmysqlclient.so` の実際の場所を反映するように変更します。
- Linux では、`libmysqlclient.so` があるディレクトリのパス名を `/etc/ld.so.conf` ファイルに追加できます。

- `libmysqlclient.so` があるディレクトリのパス名を `LD_RUN_PATH` 環境変数に追加します。システムの中には `LD_LIBRARY_PATH` を使用しているものもあります。

リンカーが見つけれられないほかのライブラリがある場合も、`-L` オプションを変更する必要がある場合があります。たとえば、`libc` が `/lib` にあり、リンクコマンドが `-L/usr/lib` を指定しているためにリンカーがそれを見つけれない場合、`-L` オプションを `-L/lib` に変更するかあるいは `-L/lib` を既存のリンクコマンドに追加します。

`DBD::mysql` から次のエラーが表示された場合、おそらく `gcc` (あるいは `gcc` でコンパイルされた旧バイナリ) を使用しているでしょう。

```
/usr/bin/perl: can't resolve symbol '__moddi3'  
/usr/bin/perl: can't resolve symbol '__divdi3'
```

`mysql.so` ライブラリがビルドされるときに、`-L/usr/lib/gcc-lib/... -lgcc` をリンクコマンドに追加します (Perl クライアントのコンパイル時に、`make` の出力で `mysql.so` を確認します)。`-L` オプションは、システム上の `libgcc.a` があるディレクトリのパス名を指定するようにしてください。

この問題の別の原因は Perl および MySQL が両方とも `gcc` でコンパイルされていない場合です。この場合、両方を `gcc` でコンパイルすることでこの不一致を解決できます。

第 3 章 チュートリアル

目次

3.1 サーバーへの接続とサーバーからの切断	265
3.2 クエリーの入力	266
3.3 データベースの作成と使用	269
3.3.1 データベースの作成と選択	270
3.3.2 テーブルの作成	270
3.3.3 テーブルへのデータのロード	272
3.3.4 テーブルからの情報の取り出し	273
3.4 データベースとテーブルに関する情報の取得	285
3.5 バッチモードでの MySQL の使用	286
3.6 一般的なクエリーの例	287
3.6.1 カラムの最大値	287
3.6.2 特定のカラムの最大値が格納されている行	288
3.6.3 グループごとのカラムの最大値	288
3.6.4 特定のカラムのグループごとの最大値が格納されている行	288
3.6.5 ユーザー定義の変数の使用	289
3.6.6 外部キーの使用	290
3.6.7 2 つのキーを使用した検索	291
3.6.8 日ごとの訪問数の計算	291
3.6.9 AUTO_INCREMENT の使用	292
3.7 Apache での MySQL の使用	294

この章では、`mysql` クライアントプログラムを使用して、簡単なデータベースを作成して使用方法を示すことで、MySQL のチュートリアルを提供します。`mysql` (「端末モニター」または単に「モニター」と呼ばれることもあります) は、MySQL Server への接続、クエリーの実行、および結果の表示を可能にするインタラクティブなプログラムです。`mysql` は、バッチモードでも使用できます。クエリーを前もってファイルに入れておき、`mysql` にファイルのコンテンツを実行するよう指示します。ここでは、`mysql` の両方の使用方法について説明します。

`mysql` で提供されているオプションのリストを表示するには、`--help` オプションを指定して `mysql` を起動します。

```
shell> mysql --help
```

この章では、`mysql` がマシンにインストールされていることと、接続可能な MySQL Server が存在することを前提にしています。そうでない場合は、MySQL 管理者に問い合わせてください。(ご自身が管理者の場合は、このドキュメントの第 5 章「MySQL サーバーの管理」などの関連部分を参照してください。)

この章では、データベースの設定と使用の手順全体について説明します。既存のデータベースへのアクセスのみに関心がある場合は、データベースおよびそのデータベースに含まれるテーブルの作成方法を説明するセクションをスキップできます。

この章はチュートリアルであるため、必然的に多くの詳細が省略されています。ここで説明されているトピックの詳細については、ドキュメントの関連セクションを参照してください。

3.1 サーバーへの接続とサーバーからの切断

サーバーに接続するには、通常、`mysql` の起動時に MySQL ユーザー名とパスワードを指定する必要があります。サーバーがログインしたマシン以外で実行されている場合は、ホスト名も指定する必要があります。接続に使用する接続パラメータ (使用するホスト、ユーザー名、およびパスワード) については、管理者に問い合わせてください。正しいパラメータがわかったら、次の方法で接続できます。

```
shell> mysql -h host -u user -p
Enter password: *****
```

`host` は MySQL サーバーが稼働しているホストの名前、`user` はユーザーの MySQL アカунトのユーザー名です。設定に応じて適切な値で置き換えてください。***** はユーザーのパスワードです。`mysql` で「Enter password:」というプロンプトが表示されたら入力してください。

それが機能すると、照会情報に続いて `mysql>` プロンプトが表示されるはずですが。

```
shell> mysql -h host -u user -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 25338 to server version: 8.0.29-standard

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

`mysql>` プロンプトに、`mysql` で SQL ステートメントを入力する準備ができていたことが示されます。

MySQL が稼働しているマシンと同じマシンにログインしている場合は、ホストを省略して次を使用できます。

```
shell> mysql -u user -p
```

ログイン時に次のようなエラーメッセージが表示される場合があります。 [ERROR 2002 \(HY000\): Can't connect to local MySQL server through socket '/tmp/mysql.sock' \(2\)](#)。これは、MySQL サーバーデーモン (Unix) またはサービス (Windows) が起動していないことを示しています。管理者に問い合わせるか、オペレーティングシステムに応じて第 2 章「MySQL のインストールとアップグレード」の適切なセクションを参照してください。

ログイン時によく発生するほかの問題については、[セクション B.3.2 「MySQL プログラム使用時の一般的なエラー」](#) を参照してください。

一部の MySQL インストールでは、ローカルホストで稼働しているサーバーに、ユーザーが匿名 (名前を指定しない) ユーザーとして接続できます。これに該当するマシンでは、オプションを何も指定せずに `mysql` を起動すると、そのサーバーに接続できるはずですが。

```
shell> mysql
```

接続に成功したら、`QUIT` (または `\q`) と `mysql>` プロンプトに入力することで、いつでも切断できます。

```
mysql> QUIT
Bye
```

Unix では、Control+D を押しても切断できます。

以降のセクションに示すほとんどの例では、サーバーに接続していることを前提にしています。このことは、`mysql>` プロンプトによって示されます。

3.2 クエリーの入力

前のセクションで説明したように、サーバーに接続していることを確認します。それだけでは操作するデータベースは選択されていませんが、それでかまいません。この時点では、テーブルの作成、テーブルへのデータのロード、テーブルからのデータの取り出しに今すぐ進むのではなく、クエリーの発行方法を学ぶことが重要です。このセクションでは、`mysql` の動作を理解するために試みることができるいくつかのクエリーを使用して、クエリーを入力する基本原則について説明します。

ここでは、サーバーにバージョン番号と現在の日付を尋ねる簡単なクエリーを示します。`mysql>` プロンプトに続けて、次に示すように入力してから Enter を押します。

```
mysql> SELECT VERSION(), CURRENT_DATE;
+-----+-----+
| VERSION() | CURRENT_DATE |
+-----+-----+
| 5.8.0-m17 | 2015-12-21   |
+-----+-----+
1 row in set (0.02 sec)
mysql>
```

このクエリーには、`mysql` に関するいくつかの事項が示されています。

- クエリーは通常、SQL ステートメントの後にセミコロンが続くもので構成されます。(セミコロンを省略できる例外もいくつかあります。前述の `QUIT` もその 1 つです。ほかのものについてはあとで説明します。)

- クエリーを発行すると、`mysql` はそのクエリーをサーバーに送信して実行し、結果を表示してから、別のクエリーの準備ができていることを示す別の `mysql>` プロンプトを出力します。
- `mysql` は、クエリーの出力を表形式 (行とカラム) で表示します。最初の行には、カラムのラベルが表示されます。以降の行はクエリーの結果です。通常、カラムラベルは、データベーステーブルからフェッチされるカラムの名前です。次の例のように、テーブルカラムではなく式の値を取得する場合、`mysql` ではカラムのラベルとして式自体が使用されます。
- `mysql` では、返された行数とクエリーの実行にかかった時間が表示されるため、サーバーの大まかなパフォーマンスがわかります。これらの値は、CPU 時間やマシン時間ではなく時計の時間で表されるため、また、サーバー負荷やネットワーク待機時間などの要因に影響されるため、正確ではありません。(簡略化のために、この章のほかの例では「rows in set」の行が省略されている場合もあります。)

キーワードは大文字でも小文字でも入力できます。次のクエリーは同等です。

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> SeLeCt vErSiOn(), current_DATE;
```

次に、別のクエリーを示します。これは、`mysql` を簡単な計算機として使用できることを示しています。

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
+-----+-----+
| SIN(PI()/4) | (4+1)*5 |
+-----+-----+
| 0.70710678118655 | 25 |
+-----+-----+
1 row in set (0.02 sec)
```

これまでに示したクエリーは、比較的短い、1 行のステートメントでした。1 行に複数のステートメントを入力することもできます。各ステートメントの末尾にはセミコロンを付加してください。

```
mysql> SELECT VERSION(); SELECT NOW();
+-----+
| VERSION() |
+-----+
| 8.0.13 |
+-----+
1 row in set (0.00 sec)

+-----+
| NOW() |
+-----+
| 2018-08-24 00:56:40 |
+-----+
1 row in set (0.00 sec)
```

クエリーのすべてを単一行で指定する必要はないため、複数の行を必要とする長いクエリーでも問題になりません。`mysql` では、入力行の終わりを探さず、終了セミコロンを探ことによって、ステートメントの終了位置が決定されます。(つまり、`mysql` は自由形式の入力を受け入れます。入力行を収集しますが、セミコロンが見つかるまでは実行しません。)

次に、単純な複数行ステートメントを示します。

```
mysql> SELECT
-> USER()
-> ,
-> CURRENT_DATE;
+-----+-----+
| USER() | CURRENT_DATE |
+-----+-----+
| jon@localhost | 2018-08-24 |
+-----+-----+
```

この例では、複数行にわたるクエリーの最初の行を入力したあと、プロンプトが `mysql>` から `->` に変化することに注意してください。これは、`mysql` がまだ完全なステートメントを検出しておらず、残りの入力を待機していることを示しています。プロンプトは貴重なフィードバックを提供してくれる味方です。そのフィードバックを使用すれば、`mysql` が何を待機しているのかを、常に認識することができます。

入力処理中のクエリーを実行しない場合は、`\c` と入力して取り消します:

```
mysql> SELECT
-> USER()
-> \c
mysql>
```

ここでもプロンプトに注目してください。 `\c` の入力後に `mysql>` に戻り、`mysql` が新しいクエリーの準備ができたことを示すフィードバックを提供します。

次の表に、表示される各プロンプトとそれらが示す `mysql` の状態をまとめます。

プロンプト	意味
<code>mysql></code>	新規クエリーの準備完了
<code>-></code>	複数行クエリーの次の行を待機しています
<code>'></code>	次の行を待機しており、一重引用符 (') で始まる文字列の完了を待機しています
<code>"></code>	次の行を待機し、二重引用符 (") で始まる文字列の完了を待機しています
<code>`></code>	次の行を待機しており、バックティック (`) で始まる識別子の完了を待機しています
<code>/*></code>	<code>/*</code> で始まったコメントが完了するまで次の行を待機しています

複数行のステートメントは、通常、単一行に対してクエリーを発行しようとしたが、終了セミコロンを忘れた場合に発生します。この場合、`mysql` は追加の入力を待機します。

```
mysql> SELECT USER()
->
```

これが起きた場合 (ステートメントを入力したのに `->` プロンプトの応答が示される) は、`mysql` がセミコロンを待機している可能性が高くなります。プロンプトの意味に気付かなければ、何をすべきか理解するまでに時間がかかるかもしれません。セミコロンを入力してステートメントを完了すると、`mysql` で実行されます。

```
mysql> SELECT USER()
-> ;
+-----+
| USER() |
+-----+
| jon@localhost |
+-----+
```

`'>` および `">` プロンプトは、文字列の収集中に発生します (MySQL が文字列の完了を待機しているという意味です)。MySQL では、'または"文字 ('hello'や"goodbye"など) で囲まれた文字列を記述でき、`mysql` では複数行にまたがる文字列を入力できます。 `'>` または `">` プロンプトが表示された場合は、'または"の引用符文字で始まる文字列を含む行を入力したが、その文字列を終了する一致する引用符をまだ入力していないことを意味します。このような場合、引用符の入力を忘れていたことがよくあります。例:

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND age < 30;
'>
```

この `SELECT` ステートメントを入力し、Enter を押して結果を待っても、何も起きません。このクエリーがなぜそれほど時間がかかるのかと不思議に思うのではなく、`'>` プロンプトで示される手がかりに注目してください。このプロンプトは、`mysql` が完了していない文字列の残りを待機していることを示しています。(このステートメントの間違いに気付きましたか。文字列 `'Smith` には、2 番目の一重引用符がありません。)

では、どうしますか。最も簡単なのは、クエリーを取り消すことです。ただし、この場合は単に `\c` と入力することはできません。なぜなら、`mysql` で収集中の文字列の一部とみなされるからです。代わりに、閉じる引用符を入力して (つまり、文字列が完了したことを `mysql` に認識させ)、次に `\c` と入力します。

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND age < 30;
'> \c
```

```
mysql>
```

プロンプトが `mysql>` に戻り、`mysql` で新しいクエリーの準備ができたことが示されます。

`>` プロンプトは `>` および `>` プロンプトと同様ですが、逆引用符を開始したがまだ終了していないことを示しています。

`>`、`>`、および `>` の各プロンプトが何を意味するかを理解することが重要です。なぜなら、誤って未終了の文字列を入力した場合、それ以降の入力行は `QUIT` を含む行も含めて、`mysql` に無視されるように見えるからです。これは、特に、現在のクエリーを取り消す前に終了引用符を指定する必要があることがわからない場合は、非常に混乱している可能性があります。

注記

この時点からの複数行のステートメントは、セカンダリ (`->` またはその他) プロンプトなしで記述されるため、ステートメントをコピーして自分で試すのが容易になります。

3.3 データベースの作成と使用

SQL ステートメントの入力方法がわかったら、データベースにアクセスできます。

何匹かのペット (`menagerie`、動物) を家で飼っているとします。これらのペットについてさまざまな種類の情報の記録をつける場合を考えます。これは、データを保持するためのテーブルを作成し、必要な情報をテーブルにロードすることで実現できます。次に、テーブルからデータを取り出すことで、ペットに関する各種の質問に回答できます。このセクションでは、次の操作の実行方法について説明します。

- データベースを作成する
- テーブルを作成する
- テーブルにデータをロードする
- さまざまな方法でテーブルからデータを取り出す
- 複数のテーブルを使用する

`menagerie` データベースは意図的に単純になっていますが、類似のデータベースを使用するような実際の状況を考えることは難しくありません。たとえば、これに似たデータベースを使用して農場主が家畜の記録をつけたり、獣医が患者の記録をつけたりできます。次のセクションで使用するクエリーおよびサンプルデータの一部を含む `menagerie` ディストリビューションは、MySQL の web サイトから取得できます。圧縮 `tar` ファイル形式および `Zip` 形式の両方で、<https://dev.mysql.com/doc/> で入手できます。

`SHOW` ステートメントを使用して、サーバーに現在どのようなデータベースが存在するかを調べます。

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| mysql |
| test |
| tmp |
+-----+
```

`mysql` データベースは、ユーザーのアクセス権限を記述します。`test` データベースは、多くの場合、ユーザーが各種の試行をするための作業スペースとして用意されています。

このステートメントで表示されるデータベースのリストは、マシンによって異なる場合があります。ユーザーが `SHOW DATABASES` 権限を持っていない場合、ユーザーにまったく権限のないデータベースは `SHOW DATABASES` で表示されません。セクション13.7.7.14「`SHOW DATABASES` ステートメント」を参照してください。

`test` データベースが存在する場合は、アクセスしてみます。

```
mysql> USE test
Database changed
```

QUIT と同様に、USE にもセミコロンは必要ありません。(これらのステートメントの末尾にセミコロンを付加しても問題ははありません。) USE ステートメントは、1 行に記述する必要があるという点でも特殊です。

test データベースへのアクセス権を持っている場合は以降の例に使用できますが、そのデータベース内に作成した内容は、アクセス権を持っているほかのユーザーによって削除される可能性があります。この理由から、自分専用のデータベースを使用する許可を MySQL 管理者に依頼するとよいでしょう。自分のデータベースに `menagerie` という名前を付けるとします。管理者は、次のようなステートメントを実行する必要があります：

```
mysql> GRANT ALL ON menagerie.* TO 'your_mysql_name'@'your_client_host';
```

ここで、`your_mysql_name` はユーザーに割り当てられた MySQL ユーザー名、`your_client_host` はサーバーに接続するホストです。

3.3.1 データベースの作成と選択

管理者がユーザーのアクセス権を設定する際にユーザー用のデータベースを作成した場合、ユーザーはそのデータベースを使用し始めることができます。そうでない場合は、ユーザー自身で作成する必要があります。

```
mysql> CREATE DATABASE menagerie;
```

Unix では、データベース名は大/小文字が区別されるため (SQL キーワードとは異なり)、データベースは常に `Menagerie`、`MENAGERIE` または他のバリエーションとしてではなく、`menagerie` として参照する必要があります。これはテーブル名にも当てはまります。(Windows の場合、この制限は適用されませんが、データベースとテーブルの表記は 1 つのクエリー内では統一する必要があります。ただし、さまざまな理由から、データベースの作成時に使用した表記を常に使用することをお勧めします。)

注記

データベースを作成しようとしたときに「`ERROR 1044 (42000): データベース 'menagerie' へのユーザー 'micah'@'localhost' のアクセスが拒否されました`」などのエラーが発生した場合は、そのために必要な権限がユーザーアカウントにないことを意味します。これについては管理者に問い合わせるか、[セクション 6.2 「アクセス制御とアカウント管理」](#) を参照してください。

データベースを作成しても、そのデータベースは選択されません。使用するには明示的に選択する必要があります。`menagerie` を現行のデータベースにするには、次のステートメントを使用します：

```
mysql> USE menagerie
Database changed
```

データベースの作成は 1 回だけ必要ですが、使用するには `mysql` セッションを開始するたびにデータベースを選択する必要があります。そのためには、例のように `USE` ステートメントを発行します。または、`mysql` を起動するときにコマンド行でデータベースを選択できます。必要な接続パラメータをすべて指定したあとに、データベースの名前を指定します。例：

```
shell> mysql -h host -u user -p menagerie
Enter password: *****
```

重要

このコマンド内の `menagerie` はパスワードではありません。`-p` オプションの後にコマンドラインでパスワードを指定する場合は、スペースを介さずに指定する必要があります (たとえば、`-p password` としてではなく `-ppassword` として指定します)。ただし、コマンド行にパスワードを入力することは、同じマシンにログインしているほかのユーザーに覗き見られるおそれがあるため、お勧めできません。

注記

`SELECT DATABASE()` を使用すると、現在どのデータベースが選択されているかをいつでも調べることができます。

3.3.2 テーブルの作成

データベースの作成は簡単な部分ですが、`SHOW TABLES` が示すとおり、この時点では空です。

```
mysql> SHOW TABLES;
Empty set (0.00 sec)
```

難しい部分は、データベースの構造、つまり、どのようなテーブルが必要で各テーブルにどのようなカラムを含めるかを決定することです。

各ペットの記録を格納するテーブルが必要です。このテーブルには、`pet` という名前を付けることができ、少なくとも各ペットの名前を含めるべきです。名前自体には深い意味はないため、このテーブルにはほかの情報も含めるとよいでしょう。たとえば、家族の複数のメンバーがペットを飼っている場合は、各ペットの所有者を記録することができます。種や性別などの基本的な説明も記録できます。

年齢はどうでしょうか。重要な情報ではありますが、データベースに格納するには適しません。年齢は時間の経過によって変化するため、記録を頻繁に更新する必要があります。代わりに、生年月日などの固定値を格納する方が適切です。そうしておけば、年齢が必要になったときに、現在の日付と生年月日の差として計算することができます。MySQL には日付演算を行う関数が用意されているため、これは難しくありません。年齢の代わりに生年月日を格納することには、ほかの利点もあります。

- たとえば、ペットの誕生日が近づいたらリマインダを生成するといったタスクにデータベースを使用できます。(このようなクエリーには意味がないと感じる場合、現在の週や月で誕生日のメッセージを送信する必要のあるクライアントを特定するためにビジネスデータベースを使用する場合と、コンピュータ支援の接客という点で同じ問題であることに注目してください。)
- 現在の日付ではなく、別の日付を基にして年齢を計算することもできます。たとえば、データベースに死亡日を格納すると、ペットが何歳で死んだかを簡単に計算できます。

`pet` テーブルに含めると役立つような情報はほかにもあるでしょうが、ここまで挙げた名前 (`name`)、所有者 (`owner`)、種 (`species`)、性別 (`sex`)、生年月日 (`birth`)、および死亡年月日 (`death`) で十分です。

`CREATE TABLE` ステートメントを使用して、テーブルのレイアウトを指定します。

```
mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),
species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
```

`name`、`owner`、および `species` の各カラムではカラム値の長さが変化するため、これらのカラムに `VARCHAR` を選択することは適切です。これらのカラム定義の長さは、すべて同じである必要はなく、`20` である必要もありません。通常は、`1` から `65535` までの範囲で、もっとも適切と思われる任意の長さを選択できます。選択が適切でなく、あとでより長いフィールドが必要になった場合は、MySQL の `ALTER TABLE` ステートメントを利用できます。

ペットのレコードで性別を表す値としては、`'m'` と `'f'`、または `'male'` と `'female'` など、いくつかの種類を選択できます。1文字の `'m'` と `'f'` を使用するのがもっとも簡単です。

`birth` カラムと `death` カラムに `DATE` データ型を使用することはかなり明白な選択です。

テーブルを作成したあとは、`SHOW TABLES` で何か出力されるはずですが。

```
mysql> SHOW TABLES;
+-----+
| Tables in menagerie |
+-----+
| pet                |
+-----+
```

想定どおりにテーブルが作成されたことを確認するには、`DESCRIBE` ステートメントを使用します。

```
mysql> DESCRIBE pet;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| name  | varchar(20) | YES  |     | NULL    |       |
| owner | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex   | char(1)   | YES  |     | NULL    |       |
| birth | date      | YES  |     | NULL    |       |
| death | date      | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+
```


`DESCRIBE` ステートメントは、テーブル内のカラムの名前や型を忘れた場合などに、いつでも使用できます。

MySQL のデータ型に関する詳細は、[第11章「データ型」](#)を参照してください。

3.3.3 テーブルへのデータのロード

テーブルを作成したら、データを移入する必要があります。これには、`LOAD DATA` ステートメントと `INSERT` ステートメントが役立ちます。

ペットのレコードを次のように記述できると仮定します。(MySQL では、'YYYY-MM-DD'形式の日付が想定されていることを確認します。これは、使用しているものとは異なる場合があります。)

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	
Claws	Gwen	cat	m	1994-03-17	
Buffy	Harold	dog	f	1989-05-13	
Fang	Benny	dog	m	1990-08-27	
Bowser	Diane	dog	m	1979-08-31	1995-07-29
Chirpy	Gwen	bird	f	1998-09-11	
Whistler	Gwen	bird		1997-12-09	
Slim	Benny	snake	m	1996-04-29	

空のテーブルから始めているため、データを移入する簡単な方法は、各ペットに対応する行を記述したテキストファイルを作成してから、1つのステートメントでそのファイルの内容をテーブルにロードすることです。

たとえば、テキストファイル `pet.txt` を作成し、1行に1レコードを記述します。値は、`CREATE TABLE` ステートメントに指定したカラムの順序に従い、タブで区切って指定します。性別が不明な場合やまだ生きているペットの死亡年月日など、不足している値には `NULL` 値を使用できます。テキストファイルでこれらを表現するには、`\N` (バックスラッシュと大文字の N) を使用します。たとえば、Whistler という鳥のレコードは次のようになります (値の間の空白は1つのタブ文字です)。

```
Whistler   Gwen   bird   \N   1997-12-09   \N
```

テキストファイル `pet.txt` を `pet` テーブルにロードするには、次のステートメントを使用します。

```
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet;
```

このファイルを Windows で作成した場合、作成に使用したエディタで `\r\n` が行ターミネータとして使用されているときは、代わりに次のステートメントを使用します。

```
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet
LINES TERMINATED BY '\r\n';
```

(macOS を実行している Apple マシンでは、`LINES TERMINATED BY '\r'`を使用する可能性があります。)

カラム値の区切り文字と行末マーカは、必要に応じて `LOAD DATA` ステートメントで明示的に指定できますが、デフォルトではタブと改行です。ステートメントで `pet.txt` ファイルを正しく読み取るにはこれで十分です。

ステートメントが失敗する場合、使用している MySQL インストールではローカルファイル機能がデフォルトで有効になっていない可能性があります。これを変更する方法については、[セクション6.1.6「LOAD DATA LOCAL のセキュリティ上の考慮事項」](#)を参照してください。

新しいレコードを1つずつ追加する場合は、`INSERT` ステートメントが役立ちます。もっとも単純な形式では、`CREATE TABLE` ステートメントに指定したカラムの順序に従って、各カラムの値を入力します。Diane が、「Puffball」という名前の新しいハムスターを手に入れたとします。次のように、`INSERT` ステートメントを使用して新しいレコードを追加できます。

```
mysql> INSERT INTO pet
VALUES ('Puffball','Diane','hamster','f','1999-03-30',NULL);
```

ここでは、文字値と日付値を、引用符付きの文字列で指定しています。また、`INSERT` では、不足している値を表す `NULL` を直接挿入することができます。`LOAD DATA` の場合のように `\N` を使用することはありません。

この例からわかるとおり、初期レコードをロードするために複数の `INSERT` ステートメントを使用すると、1つの `LOAD DATA` ステートメントを使用する場合よりもかなり多くの入力が必要になります。

3.3.4 テーブルからの情報の取り出し

テーブルから情報を取り出すには、`SELECT` ステートメントを使用します。このステートメントの一般的な形式は次のとおりです。

```
SELECT what_to_select
FROM which_table
WHERE conditions_to_satisfy;
```

`what_to_select` は取得する対象です。これには、カラムのリストか、「すべてのカラム」を表す `*` を指定できます。`which_table` は、データを取り出すテーブルです。`WHERE` 句はオプションです。指定する場合は、取得対象となる行の条件を1つまたは複数 `conditions_to_satisfy` に指定します。

3.3.4.1 すべてのデータの選択

`SELECT` のもっとも単純な形式では、テーブルのすべての内容が取り出されます。

```
mysql> SELECT * FROM pet;
+-----+-----+-----+-----+-----+-----+
| name  | owner | species | sex | birth      | death      |
+-----+-----+-----+-----+-----+-----+
| Fluffy | Harold | cat     | f   | 1993-02-04 | NULL       |
| Claws  | Gwen  | cat     | m   | 1994-03-17 | NULL       |
| Buffy  | Harold | dog     | f   | 1989-05-13 | NULL       |
| Fang   | Benny  | dog     | m   | 1990-08-27 | NULL       |
| Bowser | Diane  | dog     | m   | 1979-08-31 | 1995-07-29 |
| Chirpy | Gwen  | bird    | f   | 1998-09-11 | NULL       |
| Whistler | Gwen | bird    | NULL | 1997-12-09 | NULL       |
| Slim   | Benny  | snake   | m   | 1996-04-29 | NULL       |
| Puffball | Diane | hamster | f   | 1999-03-30 | NULL       |
+-----+-----+-----+-----+-----+-----+
```

この形式の `SELECT` では、「すべてのカラムの選択」の短縮形である `*` を使用します。これは、初期データセットとともにロードした後など、テーブル全体を確認する場合に便利です。たとえば、Bowserの生年月日が正しくないように気付いたとします。血統書の原本を確認すると、正しい生年は1979年ではなく1989年であるとわかりました。

これを修正するには、少なくとも2つの方法があります。

- ファイル `pet.txt` を編集して間違いを修正したあと、`DELETE` と `LOAD DATA` を使用してテーブルを空にしてからリロードします。

```
mysql> DELETE FROM pet;
mysql> LOAD DATA LOCAL INFILE 'pet.txt' INTO TABLE pet;
```

ただし、この方法では、Puffballのレコードも再度入力する必要があります。

- `UPDATE` ステートメントを使用して、間違っただけのレコードだけを修正します。

```
mysql> UPDATE pet SET birth = '1989-08-31' WHERE name = 'Bowser';
```

`UPDATE` は該当するレコードだけを変更するため、テーブルをリロードする必要はありません。

`SELECT *` がすべてのカラムを選択するという原則には例外があります。テーブルに非表示カラムが含まれている場合、`*` には非表示カラムは含まれません。詳細は、[セクション13.1.20.10「非表示カラム」](#)を参照してください。

3.3.4.2 特定の行の選択

前のセクションで説明したとおり、テーブル全体を取り出すことは簡単です。`SELECT` ステートメントから `WHERE` 句を省略するだけで済みます。ただし、特にテーブルが大きくなってくると、テーブル全体を取り出すことは通常あ

りません。通常は特定の質問に対する回答が必要であり、そのために、取得する情報に関していくつかの制約を指定します。ペットに関する質問に対して回答を得る選択クエリーをいくつか見てみましょう。

テーブルから特定の行だけを選択することができます。たとえば、Bowser の誕生日に加えた変更を確認するには、次のように Bowser のレコードを選択します。

```
mysql> SELECT * FROM pet WHERE name = 'Bowser';
+-----+-----+-----+-----+-----+
| name | owner | species | sex | birth   | death   |
+-----+-----+-----+-----+-----+
| Bowser | Diane | dog     | m   | 1989-08-31 | 1995-07-29 |
+-----+-----+-----+-----+-----+
```

出力から、1979 年ではなく 1989 年と正しく記録されていることが確認されます。

通常、文字列の比較では大文字と小文字が区別されないため、名前の指定には 'bowser' や 'BOWSER' などを使用できます。クエリーの結果は同じになります。

name だけでなく、任意のカラムに条件を指定できます。たとえば、1998 年以降に生まれたペットを調べるには、birth カラムを検査します。

```
mysql> SELECT * FROM pet WHERE birth >= '1998-1-1';
+-----+-----+-----+-----+-----+
| name | owner | species | sex | birth   | death   |
+-----+-----+-----+-----+-----+
| Chirpy | Gwen | bird   | f   | 1998-09-11 | NULL    |
| Puffball | Diane | hamster | f   | 1999-03-30 | NULL    |
+-----+-----+-----+-----+-----+
```

条件を組み合わせて、たとえば雌の犬を特定することができます。

```
mysql> SELECT * FROM pet WHERE species = 'dog' AND sex = 'f';
+-----+-----+-----+-----+-----+
| name | owner | species | sex | birth   | death   |
+-----+-----+-----+-----+-----+
| Buffy | Harold | dog     | f   | 1989-05-13 | NULL    |
+-----+-----+-----+-----+-----+
```

前のクエリーでは、AND 論理演算子が使用されています。OR 演算子もあります。

```
mysql> SELECT * FROM pet WHERE species = 'snake' OR species = 'bird';
+-----+-----+-----+-----+-----+
| name | owner | species | sex | birth   | death   |
+-----+-----+-----+-----+-----+
| Chirpy | Gwen | bird   | f   | 1998-09-11 | NULL    |
| Whistler | Gwen | bird   | NULL | 1997-12-09 | NULL    |
| Slim | Benny | snake  | m   | 1996-04-29 | NULL    |
+-----+-----+-----+-----+-----+
```

AND と OR は一緒に使用できますが、AND は OR よりも高い優先順位を持っています。両方の演算子を一緒に使用する場合は、括弧を使用して条件の組み合わせ方を明示的に示すことをお勧めします。

```
mysql> SELECT * FROM pet WHERE (species = 'cat' AND sex = 'm')
OR (species = 'dog' AND sex = 'f');
+-----+-----+-----+-----+-----+
| name | owner | species | sex | birth   | death   |
+-----+-----+-----+-----+-----+
| Claws | Gwen | cat     | m   | 1994-03-17 | NULL    |
| Buffy | Harold | dog     | f   | 1989-05-13 | NULL    |
+-----+-----+-----+-----+-----+
```

3.3.4.3 特定のカラムの選択

テーブルの行全体ではなく特定のカラムを取得するには、カラムの名前をカンマで区切って指定します。たとえば、ペットの生年月日を調べるには、name カラムと birth カラムを選択します。

```
mysql> SELECT name, birth FROM pet;
+-----+-----+
| name | birth   |
+-----+-----+
```

```
| Fluffy | 1993-02-04 |
| Claws | 1994-03-17 |
| Buffy | 1989-05-13 |
| Fang | 1990-08-27 |
| Bowser | 1989-08-31 |
| Chirpy | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim | 1996-04-29 |
| Puffball | 1999-03-30 |
+-----+-----+
```

ペットの所有者を調べるには、次のクエリーを使用します。

```
mysql> SELECT owner FROM pet;
+-----+
| owner |
+-----+
| Harold |
| Gwen |
| Harold |
| Benny |
| Diane |
| Gwen |
| Gwen |
| Benny |
| Diane |
+-----+
```

このクエリーは各レコードから `owner` カラムを取り出すだけなため、一部は複数回出現しています。出力を最小化するには、キーワード `DISTINCT` を追加して、一意の出カレコードをそれぞれ 1 回だけ取り出すようにします。

```
mysql> SELECT DISTINCT owner FROM pet;
+-----+
| owner |
+-----+
| Benny |
| Diane |
| Gwen |
| Harold |
+-----+
```

`WHERE` 句を使用して、行の選択とカラムの選択を組み合わせることができます。たとえば、犬と猫だけについて生年月日を調べるには、次のクエリーを使用します。

```
mysql> SELECT name, species, birth FROM pet
WHERE species = 'dog' OR species = 'cat';
+-----+-----+-----+
| name | species | birth |
+-----+-----+-----+
| Fluffy | cat | 1993-02-04 |
| Claws | cat | 1994-03-17 |
| Buffy | dog | 1989-05-13 |
| Fang | dog | 1990-08-27 |
| Bowser | dog | 1989-08-31 |
+-----+-----+-----+
```

3.3.4.4 行のソート

前の例で、結果の行の表示には特定の順序がないことに気付いたでしょう。多くの場合、クエリーの出力は、行を何らかの意味のある順序でソートすると確認しやすくなります。結果をソートするには、`ORDER BY` 句を使用します。

次に、ペットの生年月日を日付でソートしたものを示します。

```
mysql> SELECT name, birth FROM pet ORDER BY birth;
+-----+-----+
| name | birth |
+-----+-----+
| Buffy | 1989-05-13 |
| Bowser | 1989-08-31 |
| Fang | 1990-08-27 |
+-----+-----+
```

```
| Fluffy | 1993-02-04 |
| Claws | 1994-03-17 |
| Slim | 1996-04-29 |
| Whistler | 1997-12-09 |
| Chirpy | 1998-09-11 |
| Puffball | 1999-03-30 |
+-----+-----+
```

文字型のカラムでは、ソートはほかのすべての比較演算と同様に、通常大文字小文字の区別なしで実行されます。したがって、大文字と小文字の違いしかないカラムの場合、順序は未定義になります。カラムのソートで大文字と小文字を区別するには、`BINARY` を使用し、`ORDER BY BINARY col_name` のように指定します。

デフォルトのソート順序は昇順で、最小値が最初になります。逆順 (降順) でソートするには、ソートするカラムの名前に `DESC` キーワードを加えてください。

```
mysql> SELECT name, birth FROM pet ORDER BY birth DESC;
```

```
+-----+-----+
| name | birth |
+-----+-----+
| Puffball | 1999-03-30 |
| Chirpy | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim | 1996-04-29 |
| Claws | 1994-03-17 |
| Fluffy | 1993-02-04 |
| Fang | 1990-08-27 |
| Bowser | 1989-08-31 |
| Buffy | 1989-05-13 |
+-----+-----+
```

複数のカラムでソートでき、ソートの方向はカラムごとに変えることができます。たとえば、ペットの種類で昇順にソートしてから、同じ種類の中では生年月日で降順に (若い順に) ソートするには、次のクエリーを使用します。

```
mysql> SELECT name, species, birth FROM pet
ORDER BY species, birth DESC;
```

```
+-----+-----+-----+
| name | species | birth |
+-----+-----+-----+
| Chirpy | bird | 1998-09-11 |
| Whistler | bird | 1997-12-09 |
| Claws | cat | 1994-03-17 |
| Fluffy | cat | 1993-02-04 |
| Fang | dog | 1990-08-27 |
| Bowser | dog | 1989-08-31 |
| Buffy | dog | 1989-05-13 |
| Puffball | hamster | 1999-03-30 |
| Slim | snake | 1996-04-29 |
+-----+-----+-----+
```

`DESC` キーワードはその直前のカラム名 (`birth`) だけに適用されます。`species` カラムのソート順序には影響を与えません。

3.3.4.5 日付の計算

MySQL には、年齢の計算や日付の一部の抽出など、日付の計算に使用できる関数がいくつか用意されています。

それぞれのペットが何歳なのかを判別するには、`TIMESTAMPDIFF()` 関数を使用します。引数は、結果を表す単位と、差異を取る 2 つの日付です。次のクエリーでは、各ペットの生年月日、現在の日付、および年齢が表示されます。出力の最後のカラムラベルに意味を持たせるために、エイリアス (`age`) が使用されています。

```
mysql> SELECT name, birth, CURDATE(),
TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS age
FROM pet;
```

```
+-----+-----+-----+-----+
| name | birth | CURDATE() | age |
+-----+-----+-----+-----+
| Fluffy | 1993-02-04 | 2003-08-19 | 10 |
| Claws | 1994-03-17 | 2003-08-19 | 9 |
| Buffy | 1989-05-13 | 2003-08-19 | 14 |
| Fang | 1990-08-27 | 2003-08-19 | 12 |
| Bowser | 1989-08-31 | 2003-08-19 | 13 |
```



```
| Chirpy | 1998-09-11 | 2003-08-19 | 4 |
| Whistler | 1997-12-09 | 2003-08-19 | 5 |
| Slim | 1996-04-29 | 2003-08-19 | 7 |
| Puffball | 1999-03-30 | 2003-08-19 | 4 |
+-----+-----+-----+-----+
```

このクエリは正しく動作しますが、結果の行を何らかの順序で表示すると確認しやすくなるでしょう。そのためには、**ORDER BY name** 句を追加することで、出力を名前でソートできます。

```
mysql> SELECT name, birth, CURDATE(),
      TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS age
      FROM pet ORDER BY name;
```

```
+-----+-----+-----+-----+
| name | birth | CURDATE() | age |
+-----+-----+-----+
| Bowser | 1989-08-31 | 2003-08-19 | 13 |
| Buffy | 1989-05-13 | 2003-08-19 | 14 |
| Chirpy | 1998-09-11 | 2003-08-19 | 4 |
| Claws | 1994-03-17 | 2003-08-19 | 9 |
| Fang | 1990-08-27 | 2003-08-19 | 12 |
| Fluffy | 1993-02-04 | 2003-08-19 | 10 |
| Puffball | 1999-03-30 | 2003-08-19 | 4 |
| Slim | 1996-04-29 | 2003-08-19 | 7 |
| Whistler | 1997-12-09 | 2003-08-19 | 5 |
+-----+-----+-----+-----+
```

出力を **name** ではなく **age** でソートするには、異なる **ORDER BY** 句を使用します。

```
mysql> SELECT name, birth, CURDATE(),
      TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS age
      FROM pet ORDER BY age;
```

```
+-----+-----+-----+-----+
| name | birth | CURDATE() | age |
+-----+-----+-----+
| Chirpy | 1998-09-11 | 2003-08-19 | 4 |
| Puffball | 1999-03-30 | 2003-08-19 | 4 |
| Whistler | 1997-12-09 | 2003-08-19 | 5 |
| Slim | 1996-04-29 | 2003-08-19 | 7 |
| Claws | 1994-03-17 | 2003-08-19 | 9 |
| Fluffy | 1993-02-04 | 2003-08-19 | 10 |
| Fang | 1990-08-27 | 2003-08-19 | 12 |
| Bowser | 1989-08-31 | 2003-08-19 | 13 |
| Buffy | 1989-05-13 | 2003-08-19 | 14 |
+-----+-----+-----+-----+
```

類似のクエリを使用して、死んだペットの死亡時の年齢を求めることができます。どのペットなのかを判断するには、**death** 値が **NULL** かどうかを確認します。次に、**NULL** でない値について、**death** 値と **birth** 値の差を計算します。

```
mysql> SELECT name, birth, death,
      TIMESTAMPDIFF(YEAR,birth,death) AS age
      FROM pet WHERE death IS NOT NULL ORDER BY age;
```

```
+-----+-----+-----+-----+
| name | birth | death | age |
+-----+-----+-----+
| Bowser | 1989-08-31 | 1995-07-29 | 5 |
+-----+-----+-----+-----+
```

このクエリでは、**death <> NULL** ではなく **death IS NOT NULL** を使用します。**NULL** は通常の比較演算子を使用して比較することができない特殊な値であるためです。これについてはあとで説明します。[セクション3.3.4.6「NULL値の操作」](#)を参照してください。

来月誕生日を迎えるペットを調べるにはどうしますか。このような計算の場合、年と日は無関係で、**birth** カラムの月の部分を抽出するだけで済みます。MySQLには、**YEAR()**、**MONTH()**、**DAYOFMONTH()** など、日付の一部を抽出する関数がいくつか用意されています。ここでは **MONTH()** 関数が適しています。動作の仕組みを確認するために、**birth** と **MONTH(birth)** の両方の値を表示する単純なクエリを実行します。

```
mysql> SELECT name, birth, MONTH(birth) FROM pet;
```

```
+-----+-----+-----+
| name | birth | MONTH(birth) |
+-----+-----+-----+
```

```

| Fluffy | 1993-02-04 | 2 |
| Claws | 1994-03-17 | 3 |
| Buffy | 1989-05-13 | 5 |
| Fang | 1990-08-27 | 8 |
| Bowser | 1989-08-31 | 8 |
| Chirpy | 1998-09-11 | 9 |
| Whistler | 1997-12-09 | 12 |
| Slim | 1996-04-29 | 4 |
| Puffball | 1999-03-30 | 3 |
+-----+-----+-----+

```

来月誕生日を迎えるペットを見つけることも簡単です。今月は4月だとします。月の値は4であるため、5月(月5)に生まれたペットは次のように探すことができます。

```
mysql> SELECT name, birth FROM pet WHERE MONTH(birth) = 5;
```

```

+-----+-----+
| name | birth |
+-----+-----+
| Buffy | 1989-05-13 |
+-----+-----+

```

今月が12月の場合は多少複雑になります。月の番号(12)に単に1を加算して13月に生まれたペットを探すということはできません。そのような月は存在しないからです。代わりに、1月(月1)に生まれたペットを探します。

現在が何月であっても機能するクエリを記述すると、特定の月の番号を使用する必要がなくなります。`DATE_ADD()`を使用すると、所定の日付に時間間隔を加算できます。`CURDATE()`の値に1か月を加算してから、月の部分を`MONTH()`で抽出すると、誕生日を調べる月が得られます。

```
mysql> SELECT name, birth FROM pet
WHERE MONTH(birth) = MONTH(DATE_ADD(CURDATE(),INTERVAL 1 MONTH));
```

現在の月の値が12の場合はモジュロ関数(`MOD`)を適用して0に折り返してから、1を加算する方法でも、同じタスクを達成できます。

```
mysql> SELECT name, birth FROM pet
WHERE MONTH(birth) = MOD(MONTH(CURDATE()), 12) + 1;
```

`MONTH()`は1から12までの数値を返します。また、`MOD(something,12)`は0から11までの数値を返します。したがって、`MOD()`のあとで加算を行わないと、11月から1月に進んでしまいます。

計算に無効な日付が使用されている場合、計算は失敗し、警告が生成されます:

```

mysql> SELECT '2018-10-31' + INTERVAL 1 DAY;
+-----+
| '2018-10-31' + INTERVAL 1 DAY |
+-----+
| 2018-11-01 |
+-----+
mysql> SELECT '2018-10-32' + INTERVAL 1 DAY;
+-----+
| '2018-10-32' + INTERVAL 1 DAY |
+-----+
| NULL |
+-----+
mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Warning | 1292 | Incorrect datetime value: '2018-10-32' |
+-----+-----+-----+

```

3.3.4.6 NULL 値の操作

`NULL` 値に慣れるまでは驚くかもしれません。概念的には、`NULL`は「存在しない不明な値」を意味し、ほかの値とは多少異なる方法で扱われます。

`NULL`を調べるために、次に示すように`IS NULL`および`IS NOT NULL`演算子を使用します。

```
mysql> SELECT 1 IS NULL, 1 IS NOT NULL;
+-----+-----+

```

```
| 1 IS NULL | 1 IS NOT NULL |
+-----+-----+
| 0 | 1 |
+-----+-----+
```

=、<、または <> などの算術比較演算子を使用して NULL をテストすることはできません。これを自分で確認するために、次のクエリーを実行してみてください。

```
mysql> SELECT 1 = NULL, 1 <> NULL, 1 < NULL, 1 > NULL;
+-----+-----+-----+-----+
| 1 = NULL | 1 <> NULL | 1 < NULL | 1 > NULL |
+-----+-----+-----+-----+
| NULL | NULL | NULL | NULL |
+-----+-----+-----+-----+
```

NULL に関する算術比較は、結果もすべて NULL になるため、このような比較から意味のある結果を得ることはできません。

MySQL では、0 や NULL は false を意味し、それ以外はすべて true を意味します。ブール演算のデフォルトの真理値は 1 です。

NULL がこのように特殊な方法で扱われているため、前のセクションで、どの動物がもう生きていないかを判断するために、death <> NULL ではなく death IS NOT NULL を使用することが必要だったのです。

GROUP BY では、2 つの NULL 値は等しいとみなされます。

ORDER BY を実行する場合、NULL 値は ORDER BY ... ASC では最初に表示され、ORDER BY ... DESC では最後に表示されます。

NULL を操作するときによくある間違いは、NOT NULL と定義されたカラムにはゼロや空の文字列は挿入できないと想定することです。これらは実際に値ですが、一方 NULL は「値がない」ことを意味します。このことは、次に示すように IS [NOT] NULL を使用してとても簡単にテストできます。

```
mysql> SELECT 0 IS NULL, 0 IS NOT NULL, " IS NULL, " IS NOT NULL;
+-----+-----+-----+-----+
| 0 IS NULL | 0 IS NOT NULL | " IS NULL | " IS NOT NULL |
+-----+-----+-----+-----+
| 0 | 1 | 0 | 1 |
+-----+-----+-----+-----+
```

このように、ゼロや空の文字列は実際に NOT NULL であるため、NOT NULL カラムに挿入することができます。セクション B.3.4.3 「NULL 値に関する問題」を参照してください。

3.3.4.7 パターンマッチング

MySQL では、標準の SQL パターンマッチングに加え、vi、grep、sed などの Unix ユーティリティーで使用されるものに似た拡張正規表現に基づくパターンマッチング形式が提供されています。

SQL パターン一致を使用すると、_を使用して任意の単一文字を照合し、%を使用して任意の数の文字（ゼロ文字を含む）を照合できます。MySQL のデフォルトでは、SQL パターンでは大文字と小文字が区別されません。次にいくつかの例を示します。SQL パターンを使用する場合は、= または <> を使用しないでください。かわりに、LIKE または NOT LIKE の比較演算子を使用します。

bで始まる名前を探すには:

```
mysql> SELECT * FROM pet WHERE name LIKE 'b%';
+-----+-----+-----+-----+-----+-----+
| name | owner | species | sex | birth | death |
+-----+-----+-----+-----+-----+-----+
| Buffy | Harold | dog | f | 1989-05-13 | NULL |
| Bowser | Diane | dog | m | 1989-08-31 | 1995-07-29 |
+-----+-----+-----+-----+-----+-----+
```

fyで終わる名前を探すには:

```
mysql> SELECT * FROM pet WHERE name LIKE '%fy';
+-----+-----+-----+-----+-----+
| name | owner | species | sex | birth | death |
+-----+-----+-----+-----+-----+
```

```

+-----+-----+-----+-----+-----+
| Fluffy | Harold | cat   | f   | 1993-02-04 | NULL |
| Buffy  | Harold | dog   | f   | 1989-05-13 | NULL |
+-----+-----+-----+-----+-----+

```

wを含む名前を探すには:

```

mysql> SELECT * FROM pet WHERE name LIKE '%w%';
+-----+-----+-----+-----+-----+
| name  | owner | species | sex | birth      | death |
+-----+-----+-----+-----+-----+
| Claws | Gwen  | cat     | m   | 1994-03-17 | NULL  |
| Bowser | Diane | dog     | m   | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen | bird    | NULL | 1997-12-09 | NULL  |
+-----+-----+-----+-----+-----+

```

ちょうど 5 文字の名前を探すには、_パターン文字の 5 つのインスタンスを使用します。

```

mysql> SELECT * FROM pet WHERE name LIKE '_____';
+-----+-----+-----+-----+-----+
| name  | owner | species | sex | birth      | death |
+-----+-----+-----+-----+-----+
| Claws | Gwen  | cat     | m   | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f   | 1989-05-13 | NULL  |
+-----+-----+-----+-----+-----+

```

MySQL で提供されているもう 1 種類のパターンマッチングは、拡張正規表現を使用します。このタイプのパターンの一致をテストする場合は、`REGEXP_LIKE()` 関数 (または `REGEXP_LIKE()` のシノニムである `REGEXP` または `RLIKE` 演算子) を使用します。

次の表に、拡張正規表現の特徴の一部を示します。

- `.` は任意の 1 文字に一致します。
- 文字クラス`[...]`は、括弧内のいずれかの文字に一致します。たとえば、`[abc]`は `a`、`b` または `c` と一致します。文字の範囲に名前を付けるには、ダッシュを使用します。`[a-z]`は任意の文字に一致しますが、`[0-9]`は任意の数字に一致します。
- `*`は直前の文字の 0 個以上のインスタンスに一致します。たとえば、`x*` は任意の数の `x` 文字に一致し、`[0-9]*` は任意の数の数字に一致し、`*` は任意の数の任意の数の文字に一致します。
- 正規表現パターン一致は、パターンがテスト対象の値のいずれかと一致する場合に成功します。(これとは異なり、`LIKE` パターンマッチングは、パターンが値全体に一致する場合のみ成功です。)
- テストする値の先頭または末尾と一致するようにパターンをアンカーするには、パターンの先頭で `^` を使用するか、パターンの末尾で `$` を使用します。

拡張正規表現がどのように機能するかを示すために、前述の `LIKE` クエリーは `REGEXP_LIKE()` を使用するようにここにリライトされます。

`b` で始まる名前を検索するには、`^` を使用して名前の先頭と一致させます:

```

mysql> SELECT * FROM pet WHERE REGEXP_LIKE(name, '^b');
+-----+-----+-----+-----+-----+
| name  | owner | species | sex | birth      | death |
+-----+-----+-----+-----+-----+
| Buffy | Harold | dog     | f   | 1989-05-13 | NULL  |
| Bowser | Diane | dog     | m   | 1979-08-31 | 1995-07-29 |
+-----+-----+-----+-----+-----+

```

正規表現の比較を強制的に大/小文字を区別するには、大/小文字を区別する照合を使用するか、`BINARY` キーワードを使用して文字列のいずれかをバイナリ文字列にするか、`c` 一致制御文字を指定します。これらの各クエリーは、名前の先頭にある小文字の `b` のみに一致します:

```

SELECT * FROM pet WHERE REGEXP_LIKE(name, 'b' COLLATE utf8mb4_0900_as_cs);
SELECT * FROM pet WHERE REGEXP_LIKE(name, BINARY 'b');
SELECT * FROM pet WHERE REGEXP_LIKE(name, 'b', 'c');

```

`fy` で終わる名前を検索するには、`$` を使用して名前の末尾に一致させます:

```
mysql> SELECT * FROM pet WHERE REGEXP_LIKE(name, 'fy$');
+-----+-----+-----+-----+-----+
| name | owner | species | sex | birth | death |
+-----+-----+-----+-----+
| Fluffy | Harold | cat | f | 1993-02-04 | NULL |
| Buffy | Harold | dog | f | 1989-05-13 | NULL |
+-----+-----+-----+-----+
```

wを含む名前を探すには、次のクエリーを使用します。

```
mysql> SELECT * FROM pet WHERE REGEXP_LIKE(name, 'w');
+-----+-----+-----+-----+-----+
| name | owner | species | sex | birth | death |
+-----+-----+-----+-----+
| Claws | Gwen | cat | m | 1994-03-17 | NULL |
| Bowser | Diane | dog | m | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen | bird | NULL | 1997-12-09 | NULL |
+-----+-----+-----+-----+
```

正規表現パターンは値のどこかに出現した場合に一致するため、前のクエリーではワイルドカードをパターンの両側に配置して、SQL パターンの場合と同様に値全体に一致させる必要はありません。

正確に 5 文字を含む名前を検索するには、`^` と `$` を使用して名前の先頭と末尾を照合し、`.` の 5 つのインスタンスを次の間で照合します:

```
mysql> SELECT * FROM pet WHERE REGEXP_LIKE(name, '^.....$');
+-----+-----+-----+-----+-----+
| name | owner | species | sex | birth | death |
+-----+-----+-----+-----+
| Claws | Gwen | cat | m | 1994-03-17 | NULL |
| Buffy | Harold | dog | f | 1989-05-13 | NULL |
+-----+-----+-----+-----+
```

このクエリーは、`{n}` (「n 回繰り返し」) 演算子を使用して記述することもできます。

```
mysql> SELECT * FROM pet WHERE REGEXP_LIKE(name, '^.{5}$');
+-----+-----+-----+-----+-----+
| name | owner | species | sex | birth | death |
+-----+-----+-----+-----+
| Claws | Gwen | cat | m | 1994-03-17 | NULL |
| Buffy | Harold | dog | f | 1989-05-13 | NULL |
+-----+-----+-----+-----+
```

正規表現の構文の詳細は、[セクション12.8.2「正規表現」](#) を参照してください。

3.3.4.8 行のカウント

データベースは、「テーブルの中に、特定のタイプのデータがどの程度の頻度で現れるか」という質問に答えるために使用されることがよくあります。たとえば、何匹ペットを飼っているのか、それぞれの所有者が何匹のペットを所有しているかを調べたり、または動物に対してさまざまな個体数調査を実施したりすることがあるでしょう。

ペットの総数をカウントすることは、「pet テーブルには何行あるか」という質問と同等です。このテーブルにはペットごとに 1 つのレコードが存在するからです。COUNT(*) は行数をカウントするため、ペットの数をカウントするクエリーは次のようになります。

```
mysql> SELECT COUNT(*) FROM pet;
+-----+
| COUNT(*) |
+-----+
| 9 |
+-----+
```

前に、ペットの所有者の名前を取得しました。COUNT() を使用して、各所有者のペットの数を調べることができます。

```
mysql> SELECT owner, COUNT(*) FROM pet GROUP BY owner;
+-----+-----+
| owner | COUNT(*) |
+-----+-----+
| Benny | 2 |
+-----+
```



```
| Diane | 2 |
| Gwen | 3 |
| Harold | 2 |
+-----+
```

このクエリーは **GROUP BY** を使用して各 **owner** のすべてのレコードをグループ化しています。 **COUNT()** を **GROUP BY** とともに使用すると、さまざまなグループ化の下でデータの特徴を示すことができます。次の例では、ペットの個体数調査を実行するさまざまな方法を示します。

種ごとのペット数

```
mysql> SELECT species, COUNT(*) FROM pet GROUP BY species;
+-----+
| species | COUNT(*) |
+-----+
| bird   | 2 |
| cat    | 2 |
| dog    | 3 |
| hamster | 1 |
| snake  | 1 |
+-----+
```

性別ごとのペット数

```
mysql> SELECT sex, COUNT(*) FROM pet GROUP BY sex;
+-----+
| sex | COUNT(*) |
+-----+
| NULL | 1 |
| f | 4 |
| m | 4 |
+-----+
```

(この出力で、**NULL** は性別不明を示します。)

種と性別の組み合わせごとのペット数

```
mysql> SELECT species, sex, COUNT(*) FROM pet GROUP BY species, sex;
+-----+
| species | sex | COUNT(*) |
+-----+
| bird   | NULL | 1 |
| bird   | f | 1 |
| cat    | f | 1 |
| cat    | m | 1 |
| dog    | f | 1 |
| dog    | m | 2 |
| hamster | f | 1 |
| snake  | m | 1 |
+-----+
```

COUNT() を使用するときにはテーブル全体を取り出す必要はありません。たとえば、前のクエリーを犬と猫だけに対して実行する場合は、次のようになります。

```
mysql> SELECT species, sex, COUNT(*) FROM pet
  WHERE species = 'dog' OR species = 'cat'
  GROUP BY species, sex;
+-----+
| species | sex | COUNT(*) |
+-----+
| cat    | f | 1 |
| cat    | m | 1 |
| dog    | f | 1 |
| dog    | m | 2 |
+-----+
```

または、性別のわかっているペットについてのみ性別ごとのペット数を調べるには:

```
mysql> SELECT species, sex, COUNT(*) FROM pet
  WHERE sex IS NOT NULL
  GROUP BY species, sex;
+-----+
```

```

| species | sex | COUNT(*) |
+-----+-----+-----+
| bird   | f   | 1        |
| cat    | f   | 1        |
| cat    | m   | 1        |
| dog    | f   | 1        |
| dog    | m   | 2        |
| hamster| f   | 1        |
| snake  | m   | 1        |
+-----+-----+-----+

```

`COUNT()` 値に加え、選択するカラムを指定する場合は、それらのカラムを `GROUP BY` 句で指定する必要があります。そうでない場合は、次のようになります。

- `ONLY_FULL_GROUP_BY` SQL モードが有効である場合は、エラーが発生します。

```

mysql> SET sql_mode = 'ONLY_FULL_GROUP_BY';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT owner, COUNT(*) FROM pet;
ERROR 1140 (42000): In aggregated query without GROUP BY, expression
#1 of SELECT list contains nonaggregated column 'menagerie.pet.owner';
this is incompatible with sql_mode=only_full_group_by

```

- `ONLY_FULL_GROUP_BY` が有効になっていない場合、クエリーはすべての行を単一のグループとして扱うことで処理されますが、各名前付きカラムに選択された値は非決定的です。サーバーによって任意の行の値が自由に選択されます。

```

mysql> SET sql_mode = "";
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT owner, COUNT(*) FROM pet;
+-----+-----+
| owner | COUNT(*) |
+-----+-----+
| Harold | 8        |
+-----+-----+
1 row in set (0.00 sec)

```

セクション12.20.3「MySQLでのGROUP BYの処理」も参照してください。`COUNT(expr)`の動作および関連する最適化の詳細は、セクション12.20.1「集計関数の説明」を参照してください。

3.3.4.9 複数のテーブルの使用

`pet` テーブルはペットの記録を保持します。獣医の診察や出産といったペットの生涯におけるイベントなど、ペットに関するほかの情報を記録するには、別のテーブルが必要です。このテーブルはどのようなものにしたらよいでしょうか。次の情報を含める必要があります。

- 各イベントがどのペットに関するものを示すためのペット名。
- イベントがいつ発生したかを示す日付。
- イベントを説明するフィールド。
- イベントを分類できるようにする場合は、イベントタイプのフィールド。

これらを考慮すると、`event` テーブルの `CREATE TABLE` ステートメントは次のようになります。

```

mysql> CREATE TABLE event (name VARCHAR(20), date DATE,
type VARCHAR(15), remark VARCHAR(255));

```

`pet` テーブルの場合と同様に、初期レコードをロードするもっとも簡単な方法として、次の情報を記述したタブ区切りのテキストファイルを作成します。

name	date	type	remark
Fluffy	1995-05-15	litter	4 kittens, 3 female, 1 male
Buffy	1993-06-23	litter	5 puppies, 2 female, 3 male

name	date	type	remark
Buffy	1994-06-19	litter	3 puppies, 3 female
Chirpy	1999-03-21	vet	needed beak straightened
Slim	1997-08-03	vet	broken rib
Bowser	1991-10-12	kennel	
Fang	1991-10-12	kennel	
Fang	1998-08-28	birthday	Gave him a new chew toy
Claws	1998-03-17	birthday	Gave him a new flea collar
Whistler	1998-12-09	birthday	First birthday

次のようにレコードをロードします。

```
mysql> LOAD DATA LOCAL INFILE 'event.txt' INTO TABLE event;
```

`pet` テーブルで実行したクエリーから学んだことを基にすれば、原則は同じであるため `event` テーブルのレコードも取得できるはずですが、ただし、`event` テーブルだけでは質問に回答できない場合はどのようなときでしょうか。

各ペットの出産時の年齢を調べるとします。前に、2つの日付から年齢を計算する方法を学びました。ペットの出産日は `event` テーブルにあります。その日付での年齢を計算するには生年月日が必要で、それは `pet` テーブルにあります。したがって、このクエリーには両方のテーブルが必要です。

```
mysql> SELECT pet.name,
TIMESTAMPDIFF(YEAR,birth,date) AS age,
remark
FROM pet INNER JOIN event
ON pet.name = event.name
WHERE event.type = 'litter';
```

```
+-----+-----+-----+
| name | age | remark |
+-----+-----+-----+
| Fluffy | 2 | 4 kittens, 3 female, 1 male |
| Buffy | 4 | 5 puppies, 2 female, 3 male |
| Buffy | 5 | 3 puppies, 3 female |
+-----+-----+-----+
```

このクエリーには注目すべき点がいくつかあります。

- このクエリーは両方のテーブルから情報を取り出す必要があるため、`FROM` 句で2つのテーブルを結合しています。
- 複数のテーブルの情報を組み合わせる(結合する)場合、1つのテーブルのレコードとほかのテーブルのレコードがどのように対応するかを指定する必要があります。両方のテーブルに `name` カラムがあるため、これは簡単です。このクエリーは、`ON` 句を使用して、2つのテーブルのレコードを `name` 値に基づいて対応させています。

このクエリーは `INNER JOIN` を使用してテーブルを結合しています。`INNER JOIN` では、`ON` 句で指定された条件を両方のテーブルが満たす場合にかぎって、結果にテーブルの行が許可されます。この例では、`pet` テーブルの `name` カラムと `event` テーブルの `name` カラムが一致する必要があると `ON` 句で指定しています。一方のテーブルに名前が表示され、他方のテーブルには表示されない場合、`ON` 句の条件が失敗するため、結果に行は表示されません。

- `name` カラムは両方のテーブルにあるため、このカラムを参照するときはどちらのテーブルのものを明確に示す必要があります。そのためには、カラム名の前にテーブル名を付加します。

2つの異なるテーブルでなくても結合は実行できます。テーブル内のレコードをその同じテーブル内のほかのレコードと比較する場合に、テーブルをそれ自体に結合すると役立つことがあります。たとえば、ペットどうしの飼育ペアを検索するには、`pet` テーブルを自分自身と結合して、存命の雄と同一種の雌の候補ペアを生成します:

```
mysql> SELECT p1.name, p1.sex, p2.name, p2.sex, p1.species
FROM pet AS p1 INNER JOIN pet AS p2
ON p1.species = p2.species
AND p1.sex = 'f' AND p1.death IS NULL
AND p2.sex = 'm' AND p2.death IS NULL;
```

```
+-----+-----+-----+-----+
| name | sex | name | sex | species |
+-----+-----+-----+-----+
| Fluffy | f | Claws | m | cat |
| Buffy | f | Fang | m | dog |
+-----+-----+-----+-----+
```

このクエリーでは、テーブル名のエイリアスを指定してカラムを参照し、各カラムがテーブルのどちらのインスタンスに関連するかを必ず明確にしています。

3.4 データベースとテーブルに関する情報の取得

データベースやテーブルの名前を忘れた場合や、特定のテーブルの構造(カラムの名前など)を忘れた場合はどうしますか。MySQLでは、この問題に対処するために、サポートしているデータベースとテーブルについて情報を提供するステートメントがいくつか用意されています。

前出の [SHOW DATABASES](#) は、サーバーで管理されているデータベースのリストを表示します。現在どのデータベースが選択されているかを調べるには、[DATABASE\(\)](#) 関数を使用します。

```
mysql> SELECT DATABASE();
+-----+
| DATABASE() |
+-----+
| menagerie |
+-----+
```

まだどのデータベースも選択していない場合、結果は `NULL` になります。

デフォルトデータベースに含まれるテーブルを確認するには(たとえば、テーブルの名前がわからない場合)、次のステートメントを使用します:

```
mysql> SHOW TABLES;
+-----+
| Tables_in_menagerie |
+-----+
| event |
| pet |
+-----+
```

このステートメントで生成される出力のカラム名は常に `Tables_in_db_name` になります。ここで、`db_name` はデータベースの名前です。詳細については、[セクション13.7.7.39「SHOW TABLES ステートメント」](#)を参照してください。

テーブルの構造を知りたい場合は、[DESCRIBE](#) ステートメントが役に立ちます。このステートメントはテーブルの各カラムの情報を表示します。

```
mysql> DESCRIBE pet;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| name  | varchar(20) | YES | | NULL | |
| owner | varchar(20) | YES | | NULL | |
| species | varchar(20) | YES | | NULL | |
| sex   | char(1) | YES | | NULL | |
| birth | date | YES | | NULL | |
| death | date | YES | | NULL | |
+-----+-----+-----+-----+-----+
```

`Field` はカラム名、`Type` はそのカラムのデータ型、`NULL` はカラムに `NULL` 値を含められるかどうか、`Key` はカラムにインデックスが設定されているかどうか、`Default` はカラムのデフォルト値を示します。`Extra` には、カラムに関する特別な情報が表示されます:`AUTO_INCREMENT` オプションを使用してカラムが作成された場合、値は空ではなく `auto_increment` です。

`DESC` は `DESCRIBE` の省略形式です。詳細については、[セクション13.8.1「DESCRIBE ステートメント」](#)を参照してください。

既存のテーブルを作成するために必要な `CREATE TABLE` ステートメントを、`SHOW CREATE TABLE` ステートメントを使用して取得できます。[セクション13.7.7.10「SHOW CREATE TABLE ステートメント」](#)を参照してください。

テーブルにインデックスが設定されている場合は、`SHOW INDEX FROM tbl_name` でその情報を表示できます。このステートメントの詳細については、[セクション13.7.7.22「SHOW INDEX ステートメント」](#)を参照してください。

3.5 バッチモードでの MySQL の使用

前のセクションでは、`mysql` を対話形式で使用してステートメントを入力し、結果を表示しました。`mysql` をバッチモードで実行することもできます。これを行うには、実行するステートメントをファイルに入れてから、その入力をファイルから読み取るように `mysql` に指示します：

```
shell> mysql < batch-file
```

`mysql` を Windows で実行する場合に、ファイル内の一部の特殊文字によって問題が発生するときは、次のように実行できます。

```
C:\> mysql -e "source batch-file"
```

コマンド行で接続パラメータを指定する必要がある場合、コマンドは次のようになります。

```
shell> mysql -h host -u user -p < batch-file  
Enter password: *****
```

この方法で `mysql` を使用する場合は、スクリプトファイルを作成してから、そのスクリプトを実行することになります。

スクリプト内の一部のステートメントでエラーが発生してもスクリプトを続行する場合は、`--force` コマンド行オプションを使用します。

なぜスクリプトを使用するのでしょうか。いくつかの理由を次に示します。

- クエリーを繰り返し実行する場合 (毎日、毎週など)、スクリプトにすると、実行するたびに入力し直す必要がなくなります。
- 既存のクエリーのスクリプトファイルをコピーして編集することによって、類似の新しいクエリーを作成できます。
- バッチモードは、特に複数行のステートメントまたは一続きの複数ステートメントのクエリーを開発する場合にも役立ちます。間違いがあっても、すべてを入力し直す必要はありません。スクリプトを編集して間違いを修正してから、`mysql` で再度実行するだけで済みます。
- 多量の出力を生成するクエリーの場合、画面でスクロールアップする出力を見る代わりに、`pager` を介して出力できます。

```
shell> mysql < batch-file | more
```

- あとで処理できるように出力をファイルに取り込むことができます。

```
shell> mysql < batch-file > mysql.out
```

- スクリプトを他のユーザーに配布して、そのユーザーがステートメントを実行できるようにすることができます。
- `cron` ジョブからクエリーを実行する場合など、インタラクティブには使用できないことがあります。この場合はバッチモードを使用する必要があります。

`mysql` をバッチモードで実行したときのデフォルトの出力形式は、インタラクティブに使用した場合とは異なり、より簡潔になります。たとえば、`mysql` をインタラクティブに実行すると、`SELECT DISTINCT species FROM pet` の出力は次のようになります。

```
+-----+  
| species |  
+-----+  
| bird   |  
| cat    |  
| dog    |  
| hamster|  
| snake  |
```



```
+-----+
```

これに対し、バッチモードの出力は次のようになります。

```
species
bird
cat
dog
hamster
snake
```

バッチモードで、インタラクティブ出力形式のデータを取得するには、`mysql -t` を使用します。実行されるステートメントを出力にエコーするには、`mysql -v` を使用します。

`source` コマンドまたは `\.` コマンドを使用すると、`mysql` プロンプトからでもスクリプトを使用できます。

```
mysql> source filename;
mysql> \. filename
```

詳細については、[セクション4.5.1.5「テキストファイルから SQL ステートメントを実行する」](#)を参照してください。

3.6 一般的なクエリーの例

ここでは、MySQL に関する一般的な問題を解決する方法の例を示します。

一部の例では、テーブル `shop` を使用します。このテーブルには、業者 (ディーラー) の物品 (品番) ごとの価格が格納されます。各業者は物品ごとに 1 つの定価を付けていると仮定すると、(`article`, `dealer`) がレコードの主キーになります。

コマンド行ツール `mysql` を起動し、データベースを選択します。

```
shell> mysql your-database-name
```

サンプルテーブルを作成して移入するには、次のステートメントを使用します:

```
CREATE TABLE shop (
  article INT UNSIGNED DEFAULT '0000' NOT NULL,
  dealer CHAR(20) DEFAULT '' NOT NULL,
  price DECIMAL(16,2) DEFAULT '0.00' NOT NULL,
  PRIMARY KEY(article, dealer));
INSERT INTO shop VALUES
(1,'A',3.45),(1,'B',3.99),(2,'A',10.99),(3,'B',1.45),
(3,'C',1.69),(3,'D',1.25),(4,'D',19.95);
```

これらのステートメントを発行したあと、テーブルには次の内容が格納されています。

```
SELECT * FROM shop ORDER BY article;
```

```
+-----+-----+-----+
| article | dealer | price |
+-----+-----+-----+
| 1 | A | 3.45 |
| 1 | B | 3.99 |
| 2 | A | 10.99 |
| 3 | B | 1.45 |
| 3 | C | 1.69 |
| 3 | D | 1.25 |
| 4 | D | 19.95 |
+-----+-----+-----+
```

3.6.1 カラムの最大値

「もっとも大きい品番は?」

```
SELECT MAX(article) AS article FROM shop;
```

```
+-----+
| article |
```

```
+-----+
| 4 |
+-----+
```

3.6.2 特定のカラムの最大値が格納されている行

タスク: もっとも高価な物品の品番、業者、および価格を調べます。

これはサブクエリーを使用して簡単に実行できます。

```
SELECT article, dealer, price
FROM shop
WHERE price=(SELECT MAX(price) FROM shop);
```

```
+-----+-----+-----+
| article | dealer | price |
+-----+-----+-----+
| 0004 | D | 19.95 |
+-----+-----+-----+
```

ほかにも、[LEFT JOIN](#) を使用する方法や、すべての行を価格の降順でソートしてから MySQL 固有の [LIMIT](#) 句を使用して最初の行だけを取得する方法もあります。

```
SELECT s1.article, s1.dealer, s1.price
FROM shop s1
LEFT JOIN shop s2 ON s1.price < s2.price
WHERE s2.article IS NULL;
```

```
SELECT article, dealer, price
FROM shop
ORDER BY price DESC
LIMIT 1;
```

注記

最高価格のものが複数あり、価格が 19.95 の場合、[LIMIT](#) を使用した方法では、その中の 1 つしか取得できません。

3.6.3 グループごとのカラムの最大値

タスク: 物品ごとの最高値を調べます。

```
SELECT article, MAX(price) AS price
FROM shop
GROUP BY article
ORDER BY article;
```

```
+-----+-----+
| article | price |
+-----+-----+
| 0001 | 3.99 |
| 0002 | 10.99 |
| 0003 | 1.69 |
| 0004 | 19.95 |
+-----+-----+
```

3.6.4 特定のカラムのグループごとの最大値が格納されている行

タスク: 物品ごとに最高値を付けている業者 (複数可) を調べます。

この問題は、次のようなサブクエリーを使用して解決できます。

```
SELECT article, dealer, price
FROM shop s1
WHERE price=(SELECT MAX(s2.price)
              FROM shop s2
              WHERE s1.article = s2.article)
ORDER BY article;
```

```

+-----+-----+-----+
| article | dealer | price |
+-----+-----+-----+
| 0001 | B | 3.99 |
| 0002 | A | 10.99 |
| 0003 | C | 1.69 |
| 0004 | D | 19.95 |
+-----+-----+-----+

```

この例では相関サブクエリーを使用していますが、これは十分でない場合があります ([セクション13.2.11.7「相関サブクエリー」](#)を参照してください)。問題を解決する他の可能性があるのは、`FROM` 句、`LEFT JOIN` またはウィンドウ関数を使用した共通テーブル式で、関連のないサブクエリーを使用することです。

非相関サブクエリー

```

SELECT s1.article, dealer, s1.price
FROM shop s1
JOIN (
  SELECT article, MAX(price) AS price
  FROM shop
  GROUP BY article) AS s2
ON s1.article = s2.article AND s1.price = s2.price
ORDER BY article;

```

LEFT JOIN:

```

SELECT s1.article, s1.dealer, s1.price
FROM shop s1
LEFT JOIN shop s2 ON s1.article = s2.article AND s1.price < s2.price
WHERE s2.article IS NULL
ORDER BY s1.article;

```

`LEFT JOIN` は、`s1.price` が最大値である場合に、大きい値を持つ `s2.price` がないため、対応する `s2.article` 値が `NULL` であることに基いて動作します。 [セクション13.2.10.2「JOIN 句」](#)を参照してください。

ウィンドウ関数を使用した共通テーブル式:

```

WITH s1 AS (
  SELECT article, dealer, price,
         RANK() OVER (PARTITION BY article
                     ORDER BY price DESC)
         AS `Rank`
  FROM shop
)
SELECT article, dealer, price
FROM s1
WHERE `Rank` = 1
ORDER BY article;

```

3.6.5 ユーザー定義の変数の使用

MySQL ユーザー変数を使用すると、クライアント側で一時変数を使用せずに結果を記憶することができます。 ([セクション9.4「ユーザー定義変数」](#)を参照してください。)

たとえば、最高値および最安値が付けられている物品を取得するには、次を実行します。

```

mysql> SELECT @min_price:=MIN(price),@max_price:=MAX(price) FROM shop;
mysql> SELECT * FROM shop WHERE price=@min_price OR price=@max_price;
+-----+-----+-----+
| article | dealer | price |
+-----+-----+-----+
| 0003 | D | 1.25 |
| 0004 | D | 19.95 |
+-----+-----+-----+

```

注記

また、テーブルやカラムといったデータベースオブジェクトの名前をユーザー変数に格納してから、この変数を SQL ステートメントで使用することもできます。ただし、これにはブ

リペアドステートメントを使用する必要があります。詳しくは[セクション13.5「プリペアドステートメント」](#)をご覧ください。

3.6.6 外部キーの使用

MySQL では、InnoDB テーブルで外部キー制約の確認をサポートしています。第15章「[InnoDB ストレージエンジン](#)」および[セクション1.7.2.3「FOREIGN KEY 制約の違い」](#)を参照してください。

2つのテーブルを結合するだけの場合は、外部キー制約は必要ありません。InnoDB 以外のストレージエンジンの場合、カラムを定義するときに `REFERENCES tbl_name(col_name)` 句を使用できます。これは実際の効果はありませんが、現在定義しようとしているカラムが別のテーブルのカラムを参照する予定であるという自分のメモまたはコメントとして役立ちます。この構文を使用するときは、次の点を理解しておくことが非常に重要です。

- MySQL では、`col_name` が実際に `tbl_name` に存在するかどうか (または `tbl_name` 自体が存在するかどうか) を確認するチェックは実行されません。
- MySQL は、`tbl_name` に対してどのようなアクションも実行しません。たとえば、定義しようとしているテーブルの行に実行されたアクションに対応して行を削除することなどはありません。つまり、この構文にはどのような `ON DELETE` 動作や `ON UPDATE` 動作もありません。(`REFERENCES` 句の一部として `ON DELETE` 句や `ON UPDATE` 句を記述することはできますが、これらも無視されます。)
- この構文はカラムを作成します。どのようなインデックスやキーも作成しません。

このように作成したカラムを、次のように結合カラムとして使用できます。

```
CREATE TABLE person (  
  id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,  
  name CHAR(60) NOT NULL,  
  PRIMARY KEY (id)  
);  
  
CREATE TABLE shirt (  
  id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,  
  style ENUM('t-shirt', 'polo', 'dress') NOT NULL,  
  color ENUM('red', 'blue', 'orange', 'white', 'black') NOT NULL,  
  owner SMALLINT UNSIGNED NOT NULL REFERENCES person(id),  
  PRIMARY KEY (id)  
);  
  
INSERT INTO person VALUES (NULL, 'Antonio Paz');  
  
SELECT @last := LAST_INSERT_ID();  
  
INSERT INTO shirt VALUES  
(NULL, 'polo', 'blue', @last),  
(NULL, 'dress', 'white', @last),  
(NULL, 't-shirt', 'blue', @last);  
  
INSERT INTO person VALUES (NULL, 'Lilliana Angelovska');  
  
SELECT @last := LAST_INSERT_ID();  
  
INSERT INTO shirt VALUES  
(NULL, 'dress', 'orange', @last),  
(NULL, 'polo', 'red', @last),  
(NULL, 'dress', 'blue', @last),  
(NULL, 't-shirt', 'white', @last);  
  
SELECT * FROM person;  
+-----+  
| id | name          |  
+-----+  
| 1 | Antonio Paz   |  
| 2 | Lilliana Angelovska |  
+-----+  
  
SELECT * FROM shirt;  
+-----+-----+-----+  
| id | style | color | owner |  
+-----+-----+-----+-----+
```

```

| 1 | polo | blue | 1 |
| 2 | dress | white | 1 |
| 3 | t-shirt | blue | 1 |
| 4 | dress | orange | 2 |
| 5 | polo | red | 2 |
| 6 | dress | blue | 2 |
| 7 | t-shirt | white | 2 |
+-----+-----+-----+

```

```

SELECT s.* FROM person p INNER JOIN shirt s
  ON s.owner = p.id
 WHERE p.name LIKE 'Lilliana%'
  AND s.color <> 'white';

```

```

+-----+-----+-----+
| id | style | color | owner |
+-----+-----+-----+
| 4 | dress | orange | 2 |
| 5 | polo | red | 2 |
| 6 | dress | blue | 2 |
+-----+-----+-----+

```

この方法で使用する場合、[REFERENCES](#) 句は [SHOW CREATE TABLE](#) や [DESCRIBE](#) の出力に表示されません。

```

SHOW CREATE TABLE shirt\G
***** 1. row *****
Table: shirt
Create Table: CREATE TABLE `shirt` (
  `id` smallint(5) unsigned NOT NULL auto_increment,
  `style` enum('t-shirt','polo','dress') NOT NULL,
  `color` enum('red','blue','orange','white','black') NOT NULL,
  `owner` smallint(5) unsigned NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4

```

[REFERENCES](#) をこのようにカラム定義のコメントまたは「リマインダ」として使用方法は、[MyISAM](#) テーブルで機能します。

3.6.7 2つのキーを使用した検索

1つのキーを使用した [OR](#) の処理は、[AND](#) の処理と同様にかなり最適化されています。

注意が必要なのは、[OR](#) で結合された2つの異なるキーを使用して検索する場合です。

```

SELECT field1_index, field2_index FROM test_table
 WHERE field1_index = '1' OR field2_index = '1'

```

この場合は最適化されています。[セクション8.2.1.3「インデックスマージの最適化」](#)を参照してください。

2つの異なる [SELECT](#) ステートメントの出力を結合する [UNION](#) を使用することでも、この問題を効率的に解決できます。[セクション13.2.10.3「UNION 句」](#)を参照してください。

各 [SELECT](#) は1つのキーだけを検索するため、最適化できます。

```

SELECT field1_index, field2_index
  FROM test_table WHERE field1_index = '1'
 UNION
 SELECT field1_index, field2_index
  FROM test_table WHERE field2_index = '1';

```

3.6.8 日ごとの訪問数の計算

ビットグループ関数を使用して、あるユーザーが Web ページを訪問した月ごとの日数を計算する方法の例を次に示します。

```

CREATE TABLE t1 (year YEAR, month INT UNSIGNED,
  day INT UNSIGNED);
INSERT INTO t1 VALUES(2000,1,1),(2000,1,20),(2000,1,30),(2000,2,2),
  (2000,2,23),(2000,2,23);

```

このテーブルには、ユーザーがページを訪問した日付を表す年月日の値が格納されています。月ごとの訪問日数取得するには、次のクエリーを実行します。

```
SELECT year,month,BIT_COUNT(BIT_OR(1<<day)) AS days FROM t1
GROUP BY year,month;
```

次の結果が表示されます。

```
+----+-----+-----+
| year | month | days |
+----+-----+-----+
| 2000 | 1 | 3 |
| 2000 | 2 | 2 |
+----+-----+-----+
```

このクエリーでは、年と月の組み合わせに対して異なる日付が何回テーブルに出現するかを、自動的に重複エントリを除去することによって計算しています。

3.6.9 AUTO_INCREMENT の使用

AUTO_INCREMENT 属性を使用すると、新しい行に一意的識別子を生成できます。

```
CREATE TABLE animals (
  id MEDIUMINT NOT NULL AUTO_INCREMENT,
  name CHAR(30) NOT NULL,
  PRIMARY KEY (id)
);

INSERT INTO animals (name) VALUES
('dog'),('cat'),('penguin'),
('lax'),('whale'),('ostrich');

SELECT * FROM animals;
```

次の結果が表示されます。

```
+----+-----+
| id | name |
+----+-----+
| 1 | dog |
| 2 | cat |
| 3 | penguin |
| 4 | lax |
| 5 | whale |
| 6 | ostrich |
+----+-----+
```

AUTO_INCREMENT カラムには値が指定されなかったため、MySQL が自動的にシーケンス番号を割り当てました。**NO_AUTO_VALUE_ON_ZERO** SQL モードが有効になっていないかぎり、カラムに 0 を明示的に割り当てて順序番号を生成することもできます。例:

```
INSERT INTO animals (id,name) VALUES(0,'groundhog');
```

カラムが **NOT NULL** と宣言されている場合は、**NULL** 割り当ててシーケンス番号を生成することもできます。例:

```
INSERT INTO animals (id,name) VALUES(NULL,'squirrel');
```

他の値を **AUTO_INCREMENT** カラムに挿入すると、カラムはその値に設定され、次に自動的に生成される値が最大のカラム値から順番に続くように順序がリセットされます。例:

```
INSERT INTO animals (id,name) VALUES(100,'rabbit');
INSERT INTO animals (id,name) VALUES(NULL,'mouse');
SELECT * FROM animals;

+----+-----+
| id | name |
+----+-----+
| 1 | dog |
| 2 | cat |
| 3 | penguin |
| 4 | lax |
```



```

| 5 | whale |
| 6 | ostrich |
| 7 | groundhog |
| 8 | squirrel |
| 100 | rabbit |
| 101 | mouse |
+-----+-----+

```

既存の `AUTO_INCREMENT` カラム値を更新すると、`AUTO_INCREMENT` 順序もリセットされます。

`LAST_INSERT_ID()` SQL 関数または `mysql_insert_id()` C API 関数を使用して、自動的に生成された最新の `AUTO_INCREMENT` 値を取得できます。これらの関数は接続に固有の関数であるため、別の接続が同様に挿入を実行していても、戻り値は影響を受けません。

必要な最大順序値を保持するのに十分な大きさの `AUTO_INCREMENT` カラムには、最小の整数データ型を使用します。カラムがデータ型の上限値に到達すると、次にシーケンス番号を生成しようとしたときには失敗します。可能であれば、より広い範囲を可能にするために `UNSIGNED` 属性を使用します。たとえば、`TINYINT` を使用する場合、許可される最大のシーケンス番号は 127 です。`TINYINT UNSIGNED` の場合は最大値は 255 です。すべての整数型の範囲は、[セクション11.1.2「整数型\(真数値\) - INTEGER、INT、SMALLINT、TINYINT、MEDIUMINT、BIGINT」](#)を参照してください。

注記

複数行を同時に挿入する場合、`LAST_INSERT_ID()` と `mysql_insert_id()` は、実際には最初に挿入した行の `AUTO_INCREMENT` キーを返します。これにより、レプリケーションセットアップで複数行の挿入を別のサーバーで正しく再現できます。

1 以外の `AUTO_INCREMENT` 値で開始するには、次のように、その値を `CREATE TABLE` または `ALTER TABLE` でセットします。

```
mysql> ALTER TABLE tbl AUTO_INCREMENT = 100;
```

InnoDB の注意

InnoDB に固有の `AUTO_INCREMENT` の使用方法の詳細は、[セクション15.6.1.6「InnoDB での AUTO_INCREMENT 処理」](#)を参照してください。

MyISAM の注意

- MyISAM テーブルには、マルチカラムインデックス内のセカンダリカラムに `AUTO_INCREMENT` を指定することができます。この場合、`AUTO_INCREMENT` カラムに生成される値は、`MAX(auto_increment_column) + 1` `WHERE prefix=given-prefix` として計算されます。これは、データを順序付きのグループに分割する場合に便利です。

```

CREATE TABLE animals (
  grp ENUM('fish','mammal','bird') NOT NULL,
  id MEDIUMINT NOT NULL AUTO_INCREMENT,
  name CHAR(30) NOT NULL,
  PRIMARY KEY (grp,id)
) ENGINE=MyISAM;

INSERT INTO animals (grp,name) VALUES
('mammal','dog'),('mammal','cat'),
('bird','penguin'),('fish','lax'),('mammal','whale'),
('bird','ostrich');

SELECT * FROM animals ORDER BY grp,id;

```

次の結果が表示されます。

```

+-----+-----+
| grp | id | name |
+-----+-----+
| fish | 1 | lax   |
| mammal | 1 | dog   |
| mammal | 2 | cat   |
| mammal | 3 | whale |

```

```
| bird | 1 | penguin |
| bird | 2 | ostrich |
+-----+-----+
```

この場合 (`AUTO_INCREMENT` カラムがマルチカラムインデックスの一部として使用されている場合)、グループ内で最大の `AUTO_INCREMENT` 値を持つ行を削除すると、そのグループで同じ `AUTO_INCREMENT` 値が再使用されることになります。これは、通常は `AUTO_INCREMENT` 値が再使用されることのない `MyISAM` テーブルの場合にも発生します。

- `AUTO_INCREMENT` カラムが複合インデックスの一部である場合、MySQL は `AUTO_INCREMENT` カラムで始まるインデックスを使用してシーケンス値を生成します (ある場合)。たとえば、`animals` テーブルにインデックス `PRIMARY KEY (grp, id)` と `INDEX (id)` が含まれている場合、MySQL はシーケンス値の生成で `PRIMARY KEY` を無視します。その結果、テーブルには `grp` 値ごとに 1 つのシーケンスではなく、単一のシーケンスが含まれることになります。

参照情報

`AUTO_INCREMENT` に関する詳細の参照先を次に示します。

- カラムに `AUTO_INCREMENT` 属性を割り当てる方法: [セクション13.1.20「CREATE TABLE ステートメント」](#)、および [セクション13.1.9「ALTER TABLE ステートメント」](#)。
- `AUTO_INCREMENT` の、`NO_AUTO_VALUE_ON_ZERO` SQL モードによる動作の違い: [セクション5.1.11「サーバー SQL モード」](#)。
- `LAST_INSERT_ID()` 関数を使用して最新の `AUTO_INCREMENT` 値を見つける方法: [セクション12.16「情報関数」](#)。
- 使用する `AUTO_INCREMENT` 値の設定: [セクション5.1.8「サーバーシステム変数」](#)。
- [セクション15.6.1.6「InnoDB での AUTO_INCREMENT 処理」](#)
- `AUTO_INCREMENT` とレプリケーション: [セクション17.5.1.1「レプリケーションと AUTO_INCREMENT」](#)。
- レプリケーションに使用できる `AUTO_INCREMENT` 関連のサーバーシステム変数 (`auto_increment_increment` と `auto_increment_offset`): [セクション5.1.8「サーバーシステム変数」](#)。

3.7 Apache での MySQL の使用

MySQL データベースを使用してユーザーを認証し、ログファイルを MySQL のテーブルに書き込むプログラムがあります。

Apache 構成ファイルに次を追加することで、MySQL に簡単に読み込めるように Apache のロギング形式を変更することができます。

```
LogFormat \
    "%h" "%Y%m%d%H%M%S)t,%>s,"%b", "%{Content-Type}o", \
    "%U", "%{Referer}i", "%{User-Agent}i"
```

この形式のログファイルを MySQL にロードするには、次のようなステートメントを使用します。

```
LOAD DATA INFILE '/local/access_log' INTO TABLE tbl_name
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' ESCAPED BY '\\'
```

`LogFormat` 行がログファイルに書き込むデータに対応して、指定するテーブルのカラムを作成する必要があります。

第 4 章 MySQL プログラム

目次

4.1 MySQL プログラムの概要	296
4.2 MySQL プログラムの使用	299
4.2.1 MySQL プログラムの起動	299
4.2.2 プログラムオプションの指定	300
4.2.3 サーバーに接続するためのコマンドオプション	312
4.2.4 コマンドオプションを使用した MySQL Server への接続	319
4.2.5 URI 類似文字列またはキーと値のペアを使用したサーバーへの接続	322
4.2.6 DNS SRV レコードを使用したサーバーへの接続	327
4.2.7 接続トランスポートプロトコル	328
4.2.8 接続圧縮制御	330
4.2.9 環境変数の設定	333
4.3 サーバーおよびサーバーの起動プログラム	334
4.3.1 mysqld — MySQL サーバー	334
4.3.2 mysqld_safe — MySQL サーバー起動スクリプト	334
4.3.3 mysql.server — MySQL サーバー起動スクリプト	340
4.3.4 mysqld_multi — 複数の MySQL サーバーの管理	342
4.4 インストール関連プログラム	346
4.4.1 comp_err — MySQL エラーメッセージファイルのコンパイル	346
4.4.2 mysql_secure_installation — MySQL インストールのセキュリティー改善	347
4.4.3 mysql_ssl_rsa_setup — SSL/RSA ファイルの作成	350
4.4.4 mysql_tzinfo_to_sql — タイムゾーンテーブルのロード	353
4.4.5 mysql_upgrade — MySQL テーブルのチェックとアップグレード	353
4.5 クライアントプログラム	362
4.5.1 mysql — MySQL コマンドラインクライアント	362
4.5.2 mysqladmin — A MySQL Server 管理プログラム	393
4.5.3 mysqlcheck — テーブル保守プログラム	404
4.5.4 mysqldump — データベースバックアッププログラム	414
4.5.5 mysqlimport — データインポートプログラム	439
4.5.6 mysqlpump — データベースバックアッププログラム	447
4.5.7 mysqlsh — データベース、テーブル、およびカラム情報の表示	465
4.5.8 mysqlslap — ロードエミュレーションクライアント	472
4.6 管理およびユーティリティプログラム	483
4.6.1 ibd2sdi — InnoDB テーブルスペース SDI 抽出ユーティリティ	483
4.6.2 innochecksum — オフライン InnoDB ファイルチェックサムユーティリティ	486
4.6.3 myisam_ftdump — 全文インデックス情報の表示	491
4.6.4 myisamchk — MyISAM テーブルメンテナンスユーティリティ	492
4.6.5 myisamlog — MyISAM ログファイルの内容の表示	508
4.6.6 myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成	509
4.6.7 mysql_config_editor — MySQL 構成ユーティリティ	514
4.6.8 mysqlbinlog — バイナリログファイルを処理するためのユーティリティ	520
4.6.9 mysqldumpslow — スロークエリーログファイルの要約	544
4.7 プログラム開発ユーティリティ	546
4.7.1 mysql_config — クライアントのコンパイル用オプションの表示	546
4.7.2 my_print_defaults — オプションファイルからのオプションの表示	547
4.8 その他のプログラム	548
4.8.1 lz4_decompress — mysqlpump LZ4-Compressed 出力の解凍	548
4.8.2 perror — MySQL エラーメッセージ情報の表示	549
4.8.3 zlib_decompress — mysqlpump ZLIB 圧縮出力の解凍	549
4.9 環境変数	550
4.10 MySQL での Unix シグナル処理	552

この章では、Oracle Corporation が提供する MySQL コマンド行プログラムの簡単な概要を説明します。また、これらのプログラムを実行するときに、オプションを指定するための一般的な構文についても説明します。ほとんどのプ

プログラムには、その動作に固有のオプションがありますが、オプションの構文はそれらすべてに関して同様です。最後に、この章では個々のプログラムが認識するオプションを含め、詳細を説明します。

4.1 MySQL プログラムの概要

MySQL インストールには多くのさまざまなプログラムがあります。このセクションでは、それらの概要を簡単に説明します。あとのセクションでは、NDB Cluster プログラムを除いて、それぞれについてより詳細に説明します。各プログラムの説明は、その起動構文とサポートされるオプションを示しています。[セクション23.4「NDB Cluster プログラム」](#)では、NDB Cluster に固有のプログラムについて説明します。

ほとんどの MySQL 配布には、これらのプログラムが (プラットフォーム固有のプログラムを除き) すべて含まれます。(たとえば、サーバー起動スクリプトは Windows では使用されません。) 例外は、RPM 配布はより専門化されているということです。サーバー用に 1 つの RPM があり、クライアントプログラム用にもう 1 つ、などです。もし 1 つまたは複数のプログラムが欠けているようであれば、[第2章「MySQL のインストールとアップグレード」](#)を参照して、配布のタイプとその内容を確認してください。配布がすべてのプログラムを含んでおらず、追加のパッケージをインストールする必要がある場合もあります。

各 MySQL プログラムは多くのさまざまなオプションを取ります。ほとんどのプログラムには `--help` オプションがあり、プログラムのさまざまなオプションの説明を取得するために使用できます。たとえば、`mysql --help` を試してみてください。

MySQL プログラムのデフォルトのオプション値は、コマンド行のオプションまたはオプションファイルを指定して、オーバーライドできます。プログラムの起動と、プログラムオプションの指定に関する一般的な情報については、[セクション4.2「MySQL プログラムの使用」](#)を参照してください。

MySQL Server `mysqld` は、MySQL インストールのほとんどの作業を実行するメインプログラムです。サーバーには、サーバーの起動と停止を支援する、関係するいくつかのスクリプトが付随します。

- [mysqld](#)

SQL デーモン (すなわち MySQL Server)。クライアントは、サーバーに接続することでデータベースへアクセスするため、クライアントプログラムを使用するには、`mysqld` が稼働していなければなりません。[セクション4.3.1「mysqld — MySQL サーバー」](#)を参照してください。

- [mysqld_safe](#)

サーバー起動スクリプト。`mysqld_safe` は `mysqld` を起動しようとします。[セクション4.3.2「mysqld_safe — MySQL サーバー起動スクリプト」](#)を参照してください。

- [mysql.server](#)

サーバー起動スクリプト。このスクリプトは、特定の実行レベルのシステムサービスを起動するスクリプトを含む、System V スタイルの実行ディレクトリを使用するシステム上で使用されます。これは、MySQL サーバーを起動するために、`mysqld_safe` を呼び出します。[セクション4.3.3「mysql.server — MySQL サーバー起動スクリプト」](#)を参照してください。

- [mysqld_multi](#)

システムにインストールされている複数のサーバーの起動または停止を実行できるサーバー起動スクリプト。[セクション4.3.4「mysqld_multi — 複数の MySQL サーバーの管理」](#)を参照してください。

いくつかのプログラムは、MySQL のインストール中またはアップグレード中に、セットアップ操作を実行します。

- [comp_err](#)

このプログラムは MySQL のビルド/インストールプロセス中に使用されます。これは、エラーソースファイルからエラーメッセージをコンパイルします。[セクション4.4.1「comp_err — MySQL エラーメッセージファイルのコンパイル」](#)を参照してください。

- [mysql_secure_installation](#)

このプログラムにより、MySQL インストールのセキュリティーを改善できます。[セクション4.4.2「mysql_secure_installation — MySQL インストールのセキュリティー改善」](#)を参照してください。

- [mysql_ssl_rsa_setup](#)

このプログラムは、セキュアな接続をサポートするために必要な SSL 証明書およびキーファイルと RSA キーペアファイルを作成します (それらのファイルがない場合)。 [mysql_ssl_rsa_setup](#) によって作成されたファイルは、SSL または RSA を使用したセキュアな接続に使用できます。 [セクション4.4.3 「mysql_ssl_rsa_setup — SSL/RSA ファイルの作成」](#) を参照してください。

- [mysql_tzinfo_to_sql](#)

このプログラムは、ホストシステムの zoneinfo データベース (タイムゾーンを記述しているファイルセット) の内容を使用して、タイムゾーンテーブルを `mysql` データベースにロードします。 [セクション4.4.4 「mysql_tzinfo_to_sql — タイムゾーンテーブルのロード」](#) を参照してください。

- [mysql_upgrade](#)

MySQL 8.0.16 より前は、このプログラムは MySQL のアップグレード操作後に使用されます。新しいバージョンの MySQL で行われた変更で付与テーブルが更新され、テーブルの非互換性がチェックされ、必要に応じて修復されます。 [セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」](#) を参照してください。

MySQL 8.0.16 では、MySQL サーバーは以前に [mysql_upgrade](#) によって処理されたアップグレードタスクを実行します (詳細は、 [セクション2.11.3 「MySQL のアップグレードプロセスの内容」](#) を参照)。

MySQL サーバーに接続するクライアントプログラム。

- [mysql](#)

インタラクティブに SQL ステートメントを書き込む、またはバッチモードでファイルを使用してステートメントを実行するためのコマンド行ツールです。 [セクション4.5.1 「mysql — MySQL コマンドラインクライアント」](#) を参照してください。

- [mysqladmin](#)

データベースの作成と削除、付与テーブルのリロード、テーブルのディスクへのフラッシュ、およびログファイルの再オープン等、管理操作を実行するクライアント。 `mysqladmin` は、サーバーからバージョン、プロセス、およびステータス情報を取得するためにも使用できます。 [セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」](#) を参照してください。

- [mysqlcheck](#)

テーブルのチェック、修復、分析、および最適化を行うテーブルメンテナンスのクライアント。 [セクション4.5.3 「mysqlcheck — テーブル保守プログラム」](#) を参照してください。

- [mysqldump](#)

MySQL データベースを SQL、テキスト、または XML としてファイルにダンプするクライアント。 [セクション4.5.4 「mysqldump — データベースバックアッププログラム」](#) を参照してください。

- [mysqlimport](#)

`LOAD DATA` を使用してテキストファイルをそれぞれのテーブルにインポートするクライアント。 [セクション4.5.5 「mysqlimport — データインポートプログラム」](#) を参照してください。

- [mysqlpump](#)

MySQL データベースを SQL としてファイルにダンプするクライアント。 [セクション4.5.6 「mysqlpump — データベースバックアッププログラム」](#) を参照してください。

- [mysqlsh](#)

MySQL Shell は、MySQL Server の高度なクライアントおよびコードエディタです。 [MySQL Shell 8.0](#) を参照してください。 `mysql` と同様の提供される SQL 機能に加えて、MySQL Shell は JavaScript および Python のスクリプト機能を提供し、MySQL を操作するための API を備えています。X DevAPI を使用すると、リレーショナルデータとドキュメントデータの両方を操作できます。 [第20章 「ドキュメントストアとしての MySQL の使用」](#) を参照して

ください。AdminAPI を使用すると、InnoDB クラスタ を作業できます。[MySQL AdminAPI の使用](#) を参照してください。

- [mysqlshow](#)

データベース、テーブル、カラム、およびインデックスの情報を表示するクライアント。[セクション 4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」](#) を参照してください。

- [mysqlslap](#)

MySQL Server のクライアント負荷をエミュレートし、各ステージのタイミングをレポートするクライアント。複数のクライアントがサーバーにアクセスしているかのように作動します。[セクション 4.5.8 「mysqlslap — ロードエミュレーションクライアント」](#) を参照してください。

MySQL 管理プログラムおよびユーティリティープログラム:

- [innochecksum](#)

オフライン InnoDB オフラインファイルのチェックサムユーティリティー。[セクション 4.6.2 「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティー」](#) を参照してください。

- [myisam_ftdump](#)

MyISAM テーブル内の全文インデックスの情報を表示するユーティリティー。[セクション 4.6.3 「myisam_ftdump — 全文インデックス情報の表示」](#) を参照してください。

- [myisamchk](#)

MyISAM テーブルの記述、チェック、最適化、および修復を行うユーティリティー。[セクション 4.6.4 「myisamchk — MyISAM テーブルメンテナンスユーティリティー」](#) を参照してください。

- [myisamlog](#)

MyISAM ログファイルの内容を処理するユーティリティー。[セクション 4.6.5 「myisamlog — MyISAM ログファイルの内容の表示」](#) を参照してください。

- [myisampack](#)

MyISAM テーブルを圧縮してより小さい読み取り専用のテーブルを生成するユーティリティー。[セクション 4.6.6 「myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成」](#) を参照してください。

- [mysql_config_editor](#)

.mylogin.cnf という名前のセキュアで暗号化されたログインパスファイルに認証資格証明を格納できるユーティリティー。[セクション 4.6.7 「mysql_config_editor — MySQL 構成ユーティリティー」](#) を参照してください。

- [mysqlbinlog](#)

バイナリログからステートメントを読み取るためのユーティリティー。クラッシュ状態からリカバリするために、バイナリログファイルに含まれる実行ステートメントのログを使用することができます。[セクション 4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」](#) を参照してください。

- [mysqldumpslow](#)

スロークエリーログの内容を読み取って要約するユーティリティー。[セクション 4.6.9 「mysqldumpslow — スロークエリーログファイルの要約」](#) を参照してください。

MySQL プログラム開発ユーティリティー:

- [mysql_config](#)

MySQL プログラムをコンパイルするときに必要なオプション値を生成するシェルスクリプト。[セクション 4.7.1 「mysql_config — クライアントのコンパイル用オプションの表示」](#) を参照してください。

- [my_print_defaults](#)

オプションファイルのオプショングループにどのオプションがあるかを表示するユーティリティ。 [セクション 4.7.2 「my_print_defaults — オプションファイルからのオプションの表示」](#) を参照してください。

その他のユーティリティ:

- [lz4_decompress](#)

LZ4 圧縮を使用して作成された `mysqlpump` 出力を解凍するユーティリティ。 [セクション 4.8.1 「lz4_decompress — mysqlpump LZ4-Compressed 出力の解凍」](#) を参照してください。

- [perror](#)

システムあるいは MySQL エラーコードの意味を表示するユーティリティ。 [セクション 4.8.2 「perror — MySQL エラーメッセージ情報の表示」](#) を参照してください。

- [zlib_decompress](#)

ZLIB 圧縮を使用して作成された `mysqlpump` 出力を解凍するユーティリティ。 [セクション 4.8.3 「zlib_decompress — mysqlpump ZLIB 圧縮出力の解凍」](#) を参照してください。

Oracle Corporation は、[MySQL Workbench GUI](#) も提供します。これは、MySQL サーバーおよびデータベースの管理、クエリーの作成、実行、評価、およびほかのリレーショナルデータベース管理システムからのスキーマおよびデータを MySQL で使用するための移行を行うために使用されます。

MySQL クライアント/サーバーライブラリを使用してサーバーと通信する MySQL クライアントプログラムは、次の環境変数を使用します。

環境変数	意味
<code>MYSQL_UNIX_PORT</code>	<code>localhost</code> への接続に使用される、デフォルト Unix ソケットファイル
<code>MYSQL_TCP_PORT</code>	TCP/IP 接続に使用されるデフォルトのポート番号
<code>MYSQL_DEBUG</code>	デバッグ中のデバッグトレースオプション
<code>TMPDIR</code>	一時テーブルや一時ファイルが作成されるディレクトリ

MySQL プログラムが使用する環境変数の全リストについては、[セクション 4.9 「環境変数」](#) を参照してください。

4.2 MySQL プログラムの使用

4.2.1 MySQL プログラムの起動

MySQL プログラムをコマンド行から (つまり、シェルまたはコマンドプロンプトから) 起動するには、プログラム名に続けて、実行させたいことをプログラムに指示するために必要な任意のオプションまたはその他の引数を入力します。次のコマンドは、サンプルプログラムの起動を示しています。`shell>` は、コマンドインタプリタのプロンプトを表します。入力内容の一部ではありません。表示される具体的なプロンプトは、コマンドインタプリタによって異なります。通常のプロンプトは、`sh`、`ksh`、または `bash` では `$`、`cs` または `tcs` では `%`、Windows `command.com` または `cmd.exe` コマンドインタプリタでは `C:\>` です。

```
shell> mysql --user=root test
shell> mysqladmin extended-status variables
shell> mysqlshow --help
shell> mysqldump -u root personnel
```

単一ダッシュまたは二重ダッシュ (-, --) で始まる引数は、プログラムオプションを指定します。オプションは通常、プログラムが行うサーバーへの接続のタイプを示したり、動作モードに影響したりします。オプション構文については [セクション 4.2.2 「プログラムオプションの指定」](#) で説明します。

オプションではない引数 (先頭にダッシュのない引数) は、プログラムに追加情報を提供します。たとえば、`mysql` プログラムは最初のオプションではない引数をデータベース名として解釈するため、コマンド `mysql --user=root test` は `test` データベースを使用することを示します。

個々のプログラムについて説明しているこのあとのセクションでは、プログラムがどのオプションをサポートするかを示し、追加のオプションではない引数があればその意味を説明しています。

一部のオプションは、複数のプログラムで共通です。これらのうち、もっともよく使用されるのは、接続パラメータを指定する `--host` (または `-h`)、`--user` (または `-u`)、および `--password` (または `-p`) の各オプションです。これらは、MySQL サーバーが稼働しているホストの名前、MySQL アカウントのユーザー名とパスワードを示します。すべての MySQL クライアントプログラムは、これらのオプションを理解します。これらのオプションにより、どのサーバーに接続するか、およびそのサーバーで使用するアカウントを指定できます。その他の接続オプションには、TCP/IP ポート番号を指定する `--port` (または `-P`)、Unix の Unix ソケットファイル (または Windows の場合は名前付きパイプ名) を指定する `--socket` (または `-S`) があります。接続オプションを指定するオプションの詳細は、[セクション 4.2.4 「コマンドオプションを使用した MySQL Server への接続」](#) を参照してください。

MySQL プログラムを起動するために、プログラムがインストールされている `bin` ディレクトリのパス名を使用する必要があります。場合によっては、`bin` ディレクトリ以外のディレクトリから MySQL を起動しようとすると毎回「program not found」というエラーが表示される場合は、この場合に該当する可能性があります。MySQL プログラムをより使いやすくするために、MySQL の `bin` ディレクトリのパス名を `PATH` 環境変数設定に追加できます。それにより、プログラムのパス名全体ではなく名前のみを入力して実行できます。たとえば、`mysql` が `/usr/local/mysql/bin` にインストールされている場合、`mysql` と呼び出すことでプログラムを実行可能であり、`/usr/local/mysql/bin/mysql` と呼び出す必要はありません。

`PATH` 変数の設定する際の手順については、コマンドインタプリタのドキュメントを参照してください。環境変数を設定するための構文は、インタプリタ固有です。(これに関する説明は、[セクション 4.2.9 「環境変数の設定」](#) で述べられています。) `PATH` 設定を変更したら、設定が反映されるように、Windows の場合は新しいコンソールウィンドウを開き、Unix の場合はログインしなおします。

4.2.2 プログラムオプションの指定

MySQL プログラムに対してオプションを指定する方法はいくつかあります。

- コマンド行で、プログラム名に続けてオプションをリストします。これは、プログラムの特定の起動に適用されるオプションに共通です。
- プログラムが起動するときに読み取るオプションファイルにオプションをリストします。これは、プログラムを実行するとき毎回使用するオプションに共通です。
- 環境変数にオプションをリストします ([セクション 4.2.9 「環境変数の設定」](#) を参照してください)。この方法は、プログラムを実行するとき毎回適用するオプションに便利です。実際には、このためにはオプションファイルを使用する方が一般的ですが、環境変数が非常に役立つ状況については、[セクション 5.8.3 「Unix 上での複数の MySQL インスタンスの実行」](#) で説明しています。そこでは、このような変数を使用して、サーバーとクライアントプログラムに対して TCP/IP のポート番号および Unix ソケットファイルを指定する便利なテクニックについて説明しています。

オプションは順に処理されるため、あるオプションが複数回指定されている場合、最後のものが優先されます。次のコマンドでは、`mysql` は `localhost` で稼働しているサーバーに接続します。

```
mysql -h example.com -h localhost
```

1 つ例外があります。`mysqld` では、オプションファイルで指定されたユーザーがコマンド行でオーバーライドされるのを防ぐため、安全対策として `--user` オプションの最初のインスタンスが使用されます。

矛盾するオプションまたは関係するオプションが指定された場合、あとのオプションが先のオプションより優先されます。次のコマンドは `mysql` を「カラム名なし」モードで起動します。

```
mysql --column-names --skip-column-names
```

MySQL プログラムでは、最初に環境変数を調べ、次にオプションファイルを処理し、次にコマンドラインをチェックして、どのオプションを指定するかを決定します。後のオプションは前のオプションよりも優先されるため、処理順序は環境変数の優先順位が最も低く、コマンドラインオプションの優先順位が最も高いことを意味します。

サーバでは、1 つの例外が適用されます: データディレクトリ内の `mysqld-auto.cnf` オプションファイルは最後に処理されるため、コマンド行オプションよりも優先されます。

MySQL プログラムがオプションを処理する方法を利用して、プログラムのデフォルトオプション値をオプションファイルに指定できます。それにより、プログラムを実行するたびに入力するのを避けることができると同時に、必要に応じてコマンド行オプションを使用することにより、デフォルトをオーバーライドできます。

4.2.2.1 コマンド行でのオプションの使用

コマンド行で指定されるプログラムオプションは次のルールに従います。

- オプションはコマンド名のあとに指定します。
- オプション引数は、それがオプション名の短い形式か長い形式かによって、1つまたは2つのダッシュで始まります。多くのオプションには短い形式と長い形式の両方があります。たとえば、`-?` と `--help` は、MySQL プログラムに対してヘルプメッセージの表示を指示するオプションの短い形式と長い形式です。
- オプション名では大文字と小文字が区別されます。`-v` と `-V` はどちらも正当であり、意味が異なります。(これらは、`--verbose` および `--version` オプションに対応する短い形式です。)
- 一部のオプションは、オプション名に続いて値を取ります。たとえば、`-h localhost` または `--host=localhost` は、クライアントプログラムに対して MySQL サーバーホストを示します。オプション値は、プログラムに対して MySQL サーバーが稼働しているホストの名前を示します。
- 値を取る長いオプションでは、オプション名と値を `=` 記号で区切ります。値を取る短いオプションでは、オプション値はオプション文字の直後に続けるか、または間にスペースがあってもかまいません。`-localhost` と `-h localhost` は同等です。このルールの例外は、MySQL のパスワードを指定するオプションです。このオプションは、長い形式で `--password=pass_val` として、または `--password` として指定できます。後者の場合 (パスワード値が指定されていない場合)、プログラムは対話形式でパスワードの入力を求めます。パスワードオプションは、短い形式で `-ppass_val` または `-p` としても指定できます。ただし、短い形式では、パスワード値が指定されている場合、スペースの介在なしのオプション文字に従う必要があります: オプション文字のあとにスペースが続く場合、プログラムは、次の引数がパスワード値であると想定されるか、ほかの種類の引数であると想定されるかを判断する方法がありません。その結果、次の2つのコマンドは2つのまったく異なる意味を持ちます。

```
mysql -ptest
mysql -p test
```

最初のコマンドは、パスワード値 `test` を使用することを `mysql` に指示しますが、デフォルトデータベースの指定は行いません。2番目は、パスワード値を要求し、`test` をデフォルトデータベースとして使用することを `mysql` に指示します。

- オプション名の中では、ダッシュ (`-`) とアンダースコア (`_`) を同じ意味で使用できます。たとえば、`--skip-grant-tables` と `--skip_grant_tables` は同等です。(ただし、先頭のダッシュは下線で指定することはできません。)
- MySQL サーバーには、起動時にのみ指定できる特定のコマンドオプションと、起動時または実行時、あるいはその両方に設定できる一連のシステム変数があります。システム変数名はダッシュではなくアンダースコアを使用し、実行時に (`SET` ステートメントや `SELECT` ステートメントなどを使用して) 参照する場合は、アンダースコアを使用して記述する必要があります:

```
SET GLOBAL general_log = ON;
SELECT @@GLOBAL.general_log;
```

サーバーの起動時、システム変数の構文はコマンドオプションの構文と同じであるため、変数名内ではダッシュおよびアンダースコアを同じ意味で使用できます。たとえば、`--general_log=ON` と `--general-log=ON` は同等です。(これは、オプションファイル内に設定されたシステム変数にも当てはまります。)

- 数値を取るオプションの場合、 1024 、 1024^2 または 1024^3 の乗数を示すために、値に接尾辞 `K`、`M` または `G` を付けることができます。MySQL 8.0.14 の時点では、 1024^4 、 1024^5 または 1024^6 の乗数を示すために、`T`、`P` および `E` も使用できます。接尾辞文字は大文字または小文字にできます。

たとえば、次のコマンドは `mysqladmin` に対し、サーバーに 1024 回 ping を実行し、各 ping の間に 10 秒間スリープすることを指示します。

```
mysqladmin --count=1K --sleep=10 ping
```

- ファイル名をオプション値として指定する場合は、`~` Shell のメタ文字を使用しないでください。想定どおりに解釈されない場合があります。

スペースを含むオプション値をコマンド行で指定する場合は、引用符で囲まなければなりません。たとえば、`--execute` (または `-e`) オプションを `mysql` とともに使用して、セミコロンで区切られた SQL ステートメントをサーバーに渡すことができます。このオプションが使用されると、`mysql` はオプション値のステートメントを実行して終了します。ステートメントは引用符で囲む必要があります。例:

```
shell> mysql -u root -p -e "SELECT VERSION();SELECT NOW();"
Enter password: *****
+-----+
| VERSION() |
+-----+
| 8.0.19 |
+-----+
+-----+
| NOW() |
+-----+
| 2019-09-03 10:36:48 |
+-----+
shell>
```

注記

長い形式 (`--execute`) の後に等号 (=) が続きます。

ステートメント内で引用符で囲まれた値を使用するには、内部引用符をエスケープするか、ステートメント自体の引用符とは異なるタイプの引用符をステートメント内で使用する必要があります。一重引用符または二重引用符を使用できるかどうか、および引用符文字をエスケープするための構文は、コマンドプロセッサの機能により異なります。たとえば、コマンドプロセッサが一重引用符または二重引用符の使用をサポートする場合、ステートメントの周囲には二重引用符を使用し、ステートメント内の引用符で囲む値には一重引用符を使用できます。

4.2.2.2 オプションファイルの使用

ほとんどの MySQL プログラムは、オプションファイル (構成ファイルとも呼ばれる) から起動オプションを読み取ることができます。オプションファイルは、よく使用されるオプションを指定するための便利な方法を提供し、プログラムを実行するたびにコマンド行で入力する必要がなくなります。

プログラムがオプションファイルを読み取るかどうかを判断するには、`--help` オプションを使用してプログラムを呼び出します。(mysqld では、`--verbose` および `--help` を使用します。) プログラムがオプションファイルを読み取る場合は、どのファイルを探すのか、およびどのオプショングループを認識するのかが、ヘルプメッセージに示されます。

注記

`--no-defaults` オプションを使用して起動された MySQL プログラムは、`.mylogin.cnf` 以外のオプションファイルを読み取りません。

`persisted_globals_load` システム変数を無効にして起動されたサーバーは、`mysqld-auto.cnf` を読み取りません。

多くのオプションファイルはプレーンテキストファイルで、任意のテキストエディタを使用して作成されます。例外は次のとおりです:

- ログインパスオプションを含む `.mylogin.cnf` ファイル。これは、`mysql_config_editor` ユーティリティによって作成される暗号化ファイルです。セクション4.6.7「`mysql_config_editor` — MySQL 構成ユーティリティ」を参照してください。「ログインパス」は、特定のオプションのみを許可するオプショングループです: `host`, `user`, `password`, `port` および `socket`。クライアントプログラムは、`.mylogin.cnf` からどのログインパスを読み取るのかを、`--login-path` オプションを使用して指定します。

代替ログインパスファイル名を指定するには、`MYSQL_TEST_LOGIN_FILE` 環境変数を設定します。この変数は `mysql-test-run.pl` テストユーティリティが使用しますが、`mysql_config_editor` および `mysql`, `mysqldadmin`, などの MySQL クライアントによっても認識されます。

- データディレクトリ内の `mysqld-auto.cnf` ファイル。この JSON 形式ファイルには、永続化されたシステム変数設定が含まれています。これは、`SET PERSIST` または `SET PERSIST_ONLY` ステートメントの実行時にサーバーによって作成されます。セクション5.1.9.3「永続化されるシステム変数」を参照してください。`mysqld-auto.cnf` の管理はサーバーに残しておく必要があり、手動では実行しないでください。

- [オプションファイルの処理順序](#)
- [オプションファイル構文](#)
- [オプションファイルのインクルード](#)

オプションファイルの処理順序

MySQL は、次の説明に従ってオプションファイルを検索し、存在するものを読み取ります。使用するオプションファイルが存在しない場合は、前述の適切な方法を使用して作成します。

注記

NDB Cluster プログラムで使用されるオプションファイルについては、[セクション 23.3 「NDB Cluster の構成」](#) を参照してください。

Windows では、MySQL プログラムは、次のテーブルに示すファイルから起動オプションを指定された順序で読み取ります (最初にリストされたファイルが最初に読み取られ、後で読み取られたファイルが優先されます)。

表 4.1 Windows システムで読み取られるオプションファイル

ファイル名	目的
%WINDIR%\my.ini, %WINDIR%\my.cnf	グローバルオプション
C:\my.ini, C:\my.cnf	グローバルオプション
BASEDIR\my.ini, BASEDIR\my.cnf	グローバルオプション
defaults-extra-file	--defaults-extra-file で指定されたファイル (存在する場合)
%APPDATA%\MySQL\mylogin.cnf	ログインパスオプション (クライアントのみ)
DATADIR\mysqld-auto.cnf	SET PERSIST または SET PERSIST_ONLY で永続化されるシステム変数 (サーバーのみ)

前述のテーブルで、%WINDIR% は Windows ディレクトリの場所をテーブルしています。これは一般には C:\WINDOWS です。次のコマンドを使用して、WINDIR 環境変数の値から正確な場所を判断します:

```
C:\> echo %WINDIR%
```

%APPDATA% は、Windows アプリケーションデータディレクトリの値を示します。次のコマンドを使用して、APPDATA 環境変数の値から正確な場所を判断します:

```
C:\> echo %APPDATA%
```

BASEDIR は、MySQL ベースのインストールディレクトリを表します。MySQL 8.0 が MySQL Installer を使用してインストールされている場合、これは通常 C:\PROGRAMDIR\MySQL\MySQL 8.0 Server です。PROGRAMDIR はプログラムディレクトリ (英語バージョンの Windows では通常 Program Files) を表します。[セクション 2.3.3 「MySQL Installer for Windows」](#) を参照してください。

DATADIR は、MySQL データディレクトリを表します。mysqld-auto.cnf の検索に使用されるデフォルト値は、MySQL のコンパイル時に組み込まれたデータディレクトリの場所ですが、mysqld-auto.cnf の処理前に処理されるオプションファイルまたはコマンドラインオプションとして指定された --datadir によって変更できます。

Unix および Unix に似たシステムでは、MySQL プログラムは、次のテーブルに示すファイルから起動オプションを指定された順序で読み取ります (最初にリストされたファイルが最初に読み取られ、後で読み取られたファイルが優先されます)。

注記

Unix プラットフォームでは、MySQL はだれでも書き込める構成ファイルを無視します。これはセキュリティー対策として意図的なものです。

表 4.2 Unix および Unix-Like システムで読み取られるオプションファイル

ファイル名	目的
/etc/my.cnf	グローバルオプション
/etc/mysql/my.cnf	グローバルオプション
SYSCONFDIR/my.cnf	グローバルオプション
\$MYSQL_HOME/my.cnf	サーバー固有のオプション (サーバーのみ)

ファイル名	目的
defaults-extra-file	--defaults-extra-file で指定されたファイル (存在する場合)
~/my.cnf	ユーザー固有のオプション
~/mylogin.cnf	ユーザー固有のログインパスオプション (クライアントのみ)
DATADIR/mysqld-auto.cnf	SET PERSIST または SE PERSIST_ONLY で永続化されるシステム変数 (サーバーのみ)

前述のテーブルで、~ は現在のユーザーホームディレクトリ (\$HOME の値) をテーブルしています。

SYSCONFDIR は、MySQL がビルドされたときに SYSCONFDIR オプションとともに CMake に指定されたディレクトリを示します。デフォルトでは、これはコンパイル済みのインストールディレクトリの下にある etc ディレクトリです。

MYSQL_HOME はサーバー固有の my.cnf ファイルが存在するディレクトリへのパスを含む環境変数です。MYSQL_HOME が設定されていない場合に、mysqld_safe プログラムを使用してサーバーを起動すると、mysqld_safe によって BASEDIR(MySQL ベースのインストールディレクトリ) に設定されます。

DATADIR は、MySQL データディレクトリを表します。mysqld-auto.cnf の検索に使用されるデフォルト値は、MySQL のコンパイル時に組み込まれたデータディレクトリの場所ですが、mysqld-auto.cnf の処理前に処理されるオプションファイルまたはコマンドラインオプションとして指定された --datadir によって変更できます。

特定のオプションの複数のインスタンスが見つかった場合は、最後のインスタンスが優先されますが、1つの例外があります: mysqld の場合、--user オプションの first インスタンスは、オプションファイルで指定されたユーザーがコマンドラインでオーバーライドされないようにするためのセキュリティ対策として使用されます。

オプションファイル構文

次のオプションファイル構文の説明は、手動で編集するファイルに適用されます。これにより、mysql_config_editor を使用して作成され暗号化される .mylogin.cnf と、サーバーが JSON 形式で作成する mysqld-auto.cnf が除外されます。

MySQL プログラムを実行する際にコマンド行で指定できるすべての長いオプションは、オプションファイルでも指定できます。プログラムに対して使用可能なオプションのリストを取得するには、--help オプションを使用してそのプログラムを実行します。(mysqld では、--verbose および --help を使用します。)

オプションファイルでオプションを指定する構文は、コマンド行構文と同様です(セクション4.2.2.1「コマンド行でのオプションの使用」を参照してください)。ただしオプションファイルでは、先頭の2つのダッシュはオプション名から省略し、1行で1つのオプションのみを指定します。たとえば、コマンド行での --quick および --host=localhost は、オプションファイルでは独立した行にある quick および host=localhost として指定するようにしてください。--loose-opt_name 形式のオプションをオプションファイルで指定するには、loose-opt_name として作成します。

オプションファイルの空の行は無視されます。空でない行は次のいずれかの形式を取ることができます。

- #comment;comment

コメント行は#または;で始まります。# を使用するコメントは、行の途中で開始することもできます。

- [group]

group はオプションを設定するプログラムまたはグループの名前です。グループ行のあと、すべてのオプション設定行は、オプションファイルが終了するか、または別のグループ行が指定されるまで、名前を指定したグループに適用されます。オプショングループ名では、大文字と小文字は区別されません。

- opt_name

これは、コマンド行の --opt_name と同等です。

- opt_name=value

これは、コマンド行の --opt_name=value と同等です。オプションファイルでは、= 文字の周囲にスペースを置くことができます。これはコマンド行ではできません。オプションで、値を一重引用符または二重引用符で囲むことができます。これは、値に#コメント文字が含まれている場合に役立ちます。

先頭および末尾のスペースは、自動的にオプション名および値から削除されます。

オプション値にエスケープシーケンス**\b**、**\t**、**\n**、**\r**、****および**\\s**を使用して、バックスペース、タブ、改行、改行、バックスラッシュおよび空白文字を表すことができます。オプションファイルでは、次のエスケープルールが適用されます:

- バックスラッシュとそれに続く有効なエスケープシーケンス文字は、シーケンスで表される文字に変換されます。たとえば、**\\s** はスペースに変換されます。
- バックスラッシュの後に有効なエスケープシーケンス文字が続くことは変更されません。たとえば、**\\S** はそのままです。

前述のルールは、リテラルのバックスラッシュを****または**\\(有効なエスケープシーケンス文字が続く場合)**として指定できることを意味します。

オプションファイルにおけるエスケープシーケンスのルールは、SQL ステートメントにおける文字列リテラルのエスケープシーケンスのルールとは若干異なります。後者のコンテキストでは、「**x**」が有効なエスケープシーケンス文字でない場合、**\\x** は**\\x** ではなく「**x**」になります。セクション9.1.1「文字列リテラル」を参照してください。

オプションファイル値のエスケープのルールは、****をパス名区切り文字として使用する Windows パス名に特に関係します。Windows パス名の区切り文字は、エスケープシーケンス文字が続く場合は****と記述する必要があります。そうでない場合は、****または****として記述できます。または、**/**を Windows パス名で使用でき、****として扱われます。オプションファイルでベースディレクトリ **C:\Program Files\MySQL\MySQL Server 8.0** を指定するとします。これはいくつかの方法で実行できます。例:

```
basedir="C:\Program Files\MySQL\MySQL Server 8.0"
basedir="C:\\Program Files\\MySQL\\MySQL Server 8.0"
basedir="C:/Program Files/MySQL/MySQL Server 8.0"
basedir=C:\\Program\\Files\\MySQL\\MySQL\\sServer\\s8.0
```

オプショングループ名がプログラム名と同じである場合、グループ内のオプションは特にそのプログラムに適用されます。たとえば、**[mysqld]** グループおよび **[mysql]** グループは、それぞれ **mysqld** サーバーおよび **mysql** クライアントプログラムに適用されます。

[client] オプショングループは、MySQL ディストリビューションで提供されるすべてのクライアントプログラムによって読み取られます (**mysqld** によってではありません)。C API を使用するサードパーティのクライアントプログラムでオプションファイルを使用する方法を理解するには、**mysql_options()** の C API ドキュメントを参照してください。

[client] グループを使用すると、すべてのクライアントに適用するオプションを指定できます。たとえば、**[client]** は、サーバーに接続するためのパスワードの指定に使用する適切なグループです。(ただし、他のユーザーが自分のパスワードを検出できないように、オプションファイルに自分でのみアクセスできることを確認してください。)使用するすべてのクライアントプログラムが **[client]** グループを認識しないかぎり、オプションに置かないようにしてください。そのオプションを理解しないプログラムを実行しようとすると、そのプログラムはエラーメッセージを表示してから終了します。

より一般的なオプショングループを最初にリストし、より具体的なグループを後でリストします。たとえば、**[client]** グループはすべてのクライアントプログラムによって読み取られるのに対し、**[mysqldump]** グループは **mysqldump** によって読取り専用であるため、より一般的です。後で指定したオプションは、オプショングループを **[client]**、**[mysqldump]** の順に配置するため、**mysqldump** 固有のオプションで **[client]** オプションをオーバーライドできます。

一般的なグローバルオプションファイルを次に示します。

```
[client]
port=3306
socket=/tmp/mysql.sock

[mysqld]
port=3306
socket=/tmp/mysql.sock
key_buffer_size=16M
max_allowed_packet=128M

[mysqldump]
quick
```

一般的なユーザーオプションファイルを次に示します。

```
[client]
```

```
# The following password is sent to all standard MySQL clients
password="my password"
```

```
[mysql]
no-auto-rehash
connect_timeout=2
```

特定の MySQL リリースシリーズから `mysqld` サーバーによって読み取り専用になるオプショングループを作成するには、`[mysqld-5.7]`、`[mysqld-8.0]`などの名前のグループを使用します。次のグループは、8.0.x のバージョン番号を持つ MySQL サーバーでのみ `sql_mode` 設定を使用する必要があることを示しています：

```
[mysqld-8.0]
sql_mode=TRADITIONAL
```

オプションファイルのインクルード

オプションファイルで、`!include` ディレクティブを使用してほかのオプションファイルをインクルードしたり、`!includedir` を使用して特定のディレクトリでオプションファイルを検索したりできます。たとえば、`/home/mydir/myopt.cnf` ファイルをインクルードするには、次のディレクティブを使用します。

```
!include /home/mydir/myopt.cnf
```

`/home/mydir` ディレクトリを検索してそこで見つかったオプションファイルを読み取るには、次のディレクティブを使用します。

```
!includedir /home/mydir
```

MySQL では、ディレクトリ内のオプションファイルが読み取られる順序は保証されません。

注記

Unix オペレーティングシステムで `!includedir` ディレクティブを使用して検出およびインクルードするファイルには、`.cnf` で終わるファイル名が必要です。Windows においては、このディレクティブは `.ini` または `.cnf` 拡張子を持つファイルをチェックします。

インクルードされるオプションファイルの内容は、ほかのオプションファイルと同様に記述します。すなわち、オプションのグループを含み、それぞれの前にオプションが適用されるプログラムを示す `[group]` 行があるようにしてください。

インクルードされるファイルの処理中、現在のプログラムが検索するグループ内のオプションのみが使用されます。その他のグループは無視されます。`my.cnf` ファイルに次の行が含まれるとします。

```
!include /home/mydir/myopt.cnf
```

また、`/home/mydir/myopt.cnf` は次のようであるとします。

```
[mysqladmin]
force

[mysqld]
key_buffer_size=16M
```

`my.cnf` が `mysqld` によって処理される場合、`/home/mydir/myopt.cnf` 内の `[mysqld]` グループのみが使用されます。このファイルが `mysqladmin` によって処理される場合、`[mysqladmin]` グループのみが使用されます。このファイルがその他のプログラムによって処理される場合、`/home/mydir/myopt.cnf` のオプションは使用されません。

`!includedir` ディレクティブは同様に処理されますが、指名されたディレクトリ内のすべてのオプションファイルが読み取られる点が異なります。

オプションファイルに `!include` または `!includedir` ディレクティブが含まれている場合、それらのディレクティブによって指定されたファイルは、ファイル内のどこに指定されていても、オプションファイルが処理されるたびに処理されます。

包含ディレクティブが機能するには、ファイルパスを引用符で囲まず、エスケープシーケンスを使用しないでください。たとえば、`my.ini` で提供されている次のステートメントは、オプションファイル `myopts.ini` を読み取ります：

```
!include C:/ProgramData/MySQL/MySQL Server/myopts.ini
!include C:\ProgramData\MySQL\MySQL Server\myopts.ini
!include C:\\ProgramData\\MySQL\\MySQL Server\\myopts.ini
```

Windows では、`!include /path/to/extra.ini` がファイルの最後の行である場合、末尾に改行が追加されていることを確認してください。それ以外の場合、行は無視されます。

4.2.2.3 オプションファイルの処理に影響するコマンド行オプション

オプションファイルをサポートするほとんどの MySQL プログラムは、次のオプションを処理します。これらのオプションはオプションファイルの処理に影響するため、オプションファイルではなくコマンド行で指定する必要があります。これらのオプションはそれぞれ、正しく機能するためにはほかのオプションより前に指定する必要があります。ただし、次の例外があります。

- `--print-defaults` は、`--defaults-file`、`--defaults-extra-file` または `--login-path` の直後に使用できます。
- Windows では、サーバーが `--defaults-file` および `--install` オプションで起動される場合、`--install` が最初でなければなりません。セクション2.3.4.8「Windows のサービスとして MySQL を起動する」を参照してください。

ファイル名をオプション値として指定する場合、~ Shell のメタ文字は想定どおりに解釈されない可能性があるため、使用しないでください。

- `--defaults-extra-file=file_name`

このオプションファイルは、グローバルオプションファイルの後 (Unix の場合はユーザーオプションファイルの前、すべてのプラットフォームの場合はログインパスファイルの前) に読み取ります。(オプションファイルの使用順序の詳細は、セクション4.2.2.2「オプションファイルの使用」を参照してください。) ファイルが存在しないかアクセスできない場合、エラーが発生します。`file_name` は、フルパス名でなく相対パス名として指定された場合、現行ディレクトリを基準にして解釈されます。

このオプションを指定できる位置の制約については、このセクションの概要を参照してください。

- `--defaults-file=file_name`

指定されたオプションファイルのみを読み取ります。ファイルが存在しないかアクセスできない場合、エラーが発生します。`file_name` は、フルパス名でなく相対パス名として指定された場合、現行ディレクトリを基準にして解釈されます。

例外: `--defaults-file` でも、`mysqld` は `mysqld-auto.cnf` を読み取り、クライアントプログラムは `.mylogin.cnf` を読み取ります。

このオプションを指定できる位置の制約については、このセクションの概要を参照してください。

- `--defaults-group-suffix=str`

通常のオプショングループだけでなく、通常の名前に `str` のサフィクスが付いたグループも読み取ります。たとえば、`mysql` クライアントは通常 `[client]` グループおよび `[mysql]` グループを読み取ります。`--defaults-group-suffix=_other` オプションを指定した場合、`mysql` は `[client_other]` グループおよび `[mysql_other]` グループも読み取ります。

- `--login-path=name`

`.mylogin.cnf` ログインパスファイルの指定されたログインパスからオプションを読み取ります。「ログインパス」は、接続先の MySQL サーバーおよび認証に使用するアカウントを指定するオプションを含むオプショングループです。ログインパスファイルを作成または変更するには、`mysql_config_editor` ユーティリティを使用します。セクション4.6.7「`mysql_config_editor` — MySQL 構成ユーティリティー」を参照してください。

クライアントプログラムは、デフォルトで読み取られるオプショングループに加えて、指定されたログインパスに対応するオプショングループを読み取ります。次のコマンドについて考えてみます:

```
mysql --login-path=mypath
```

デフォルトでは、`mysql` クライアントは `[client]` および `[mysql]` オプショングループを読み取ります。そのため、示されているコマンドの場合、`mysql` は他のオプションファイルから `[client]` および `[mysql]` を読み取り、ログインパスファイルから `[client]`、`[mysql]` および `[mypath]` を読み取ります。

クライアントプログラムは、`--no-defaults` オプションが使用されている場合でもログインパスファイルを読み取ります。

代替ログインパスファイル名を指定するには、`MYSQL_TEST_LOGIN_FILE` 環境変数を設定します。

このオプションを指定できる位置の制約については、このセクションの概要を参照してください。

- `--no-defaults`

オプションファイルを読み取りません。オプションファイルから不明のオプションを読み取ることが原因でプログラムの起動に失敗する場合、`--no-defaults` を使用して、オプションを読み取らないようにすることができます。

ただし、クライアントプログラムは、`--no-defaults` が使用されている場合でも、`.mylogin.cnf` ログインパスファイルを読み取ります (存在する場合)。これにより、`--no-defaults` が存在する場合でも、コマンドラインよりも安全な方法でパスワードを指定できます。(`.mylogin.cnf` は `mysql_config_editor` ユーティリティによって作成されます。セクション4.6.7「`mysql_config_editor` — MySQL 構成ユーティリティー」を参照してください)。

- `--print-defaults`

プログラム名と、オプションファイルから受け取るすべてのオプションを出力します。パスワード値はマスクされます。

このオプションを指定できる位置の制約については、このセクションの概要を参照してください。

4.2.2.4 プログラムオプション修飾子

一部のオプションは「ブール値」で、オンまたはオフにできる動作を制御します。たとえば、`mysql` クライアントは、クエリーの結果の先頭にカラム名の行を表示するかどうかを決定するオプション `--column-names` をサポートします。デフォルトでは、このオプションは有効です。ただし、初期ヘッダーラインを想定せずデータのみを想定するような別のプログラムに `mysql` の出力を送信する場合などは、この機能を無効にするとよいでしょう。

カラム名を無効にするには、次のいずれかの形式を使用してオプションを指定します。

```
--disable-column-names
--skip-column-names
--column-names=0
```

`--disable` プリフィクスと `--skip` プリフィクス、および `=0` サフィクスはすべて同じ効果を持ち、オプションをオフにします。

オプションを「有効」にする形式は、次のうちのいずれかで指定できます。

```
--column-names
--enable-column-names
--column-names=1
```

値 `ON`, `TRUE`, `OFF` および `FALSE` もブールオプションで認識されます (大/小文字は区別されません)。

オプションに `--loose` プリフィクスがある場合、プログラムがオプションを認知できない場合にはエラーで終了せず、警告のみを発行します。

```
shell> mysql --loose-no-such-option
mysql: WARNING: unknown option '--loose-no-such-option'
```

`--loose` 接頭辞は、同じマシン上の MySQL の複数のインストールからプログラムを実行し、オプションファイルにオプションをリストする場合に役立ちます。プログラムのすべてのバージョンで認識されない可能性があるオプションは、`--loose` 接頭辞 (またはオプションファイル内の `loose`) を使用して指定できます。オプションを認識するバージョンのプログラムは通常どおりに処理し、認識しないバージョンは警告を発行して無視します。

`--maximum` 接頭辞は `mysqld` でのみ使用可能で、クライアントプログラムがセッションシステム変数を設定できる大きさに制限を設定できます。これを実行するには、`--maximum` プリフィクスを変数名とともに使用します。たとえば、`--maximum-max_heap_table_size=32M` では、クライアントがヒープテーブルのサイズ制限を 32M より大きくすることを防ぎます。

`--maximum` 接頭辞は、セッション値を持つシステム変数で使用するためのものです。グローバル値のみを持つシステム変数に適用すると、エラーが発生します。たとえば、`--maximum-back_log=200` の場合、サーバーは次のエラーを生成します:

```
Maximum value of 'back_log' cannot be set
```

4.2.2.5 プログラム変数の設定へのオプションの使用

多くの MySQL プログラムには内部変数があり、実行時に `SET` ステートメントを使用して設定できます。 [セクション 13.7.6.1「変数代入の SET 構文」](#) および [セクション 5.1.9「システム変数の使用」](#) を参照してください。

これらのプログラム変数のほとんどは、プログラムオプションの指定に適用されるのと同じ構文を使用して、サーバーの起動時にも設定できます。たとえば、`mysql` には通信バッファの最大サイズを制御する `max_allowed_packet` 変数があります。 `mysql` に対して `max_allowed_packet` 変数を 16M バイトの値にセットするには、次のコマンドのいずれかを使用してください。

```
mysql --max_allowed_packet=16777216
mysql --max_allowed_packet=16M
```

最初のコマンドは値をバイトで指定します。2 番目は値を M バイトで指定します。数値を取る変数の場合、 1024^1 、 1024^2 または 1024^3 の乗数を示すために、値に接尾辞 `K`、`M` または `G` を付けることができます。(たとえば、`max_allowed_packet` を設定するために使用される場合、サフィクスは K バイト、M バイトまたは G バイトの単位を示します。) MySQL 8.0.14 の時点では、 1024^4 、 1024^5 または 1024^6 の乗数を示すために、`T`、`P` および `E` も使用できます。接尾辞文字は大文字または小文字にできます。

オプションファイルでは、変数の設定は先頭のダッシュなしで指定されます。

```
[mysql]
max_allowed_packet=16777216
```

または:

```
[mysql]
max_allowed_packet=16M
```

必要に応じて、オプション名のアンダースコアをダッシュとして指定できます。次のオプショングループは同等です。どちらもサーバーのキーバッファを 512M バイトに設定します。

```
[mysqld]
key_buffer_size=512M
```

```
[mysqld]
key-buffer-size=512M
```

値乗数を指定する接尾辞は、プログラムの起動時に変数を設定する場合に使用できますが、実行時に `SET` で値を設定する場合には使用できません。一方、`SET` を使用すると、式を使用して変数の値を割り当てることができますが、サーバーの起動時に変数を設定するときには使用できません。たとえば、次の最初の行はプログラムの起動時に有効ですが、次の行は有効ではありません:

```
shell> mysql --max_allowed_packet=16M
shell> mysql --max_allowed_packet=16*1024*1024
```

逆に、実行時に次の 2 行目は有効ですが 1 行目は無効です。

```
mysql> SET GLOBAL max_allowed_packet=16M;
mysql> SET GLOBAL max_allowed_packet=16*1024*1024;
```

4.2.2.6 オプションのデフォルト、値を想定するオプション、および = 記号

慣例により、値を割り当てるオプションの長い形式は、次のように等号 (=) で記述されます。

```
mysql --host=tonfisk --user=jon
```

値を必要とする (つまり、デフォルト値を持たない) オプションの場合、等号は必要ないため、次も有効です:

```
mysql --host tonfisk --user jon
```

どちらの場合も、`mysql` クライアントは「tonfisk」という名前のホストで稼働している MySQL サーバーに、ユーザー名「jon」のアカウントを使用して接続しようとします。

この動作のため、値を想定するオプションに値を指定しない場合に、時として問題が生じることがあります。次の例を見てください。ユーザーがホスト `tonfisk` で稼働している MySQL サーバーに、ユーザー `jon` として接続します。

```
shell> mysql --host 85.224.35.45 --user jon
Welcome to the MySQL monitor. Commands end with ; or \g.
```



```
Your MySQL connection id is 3
Server version: 8.0.29 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT CURRENT_USER();
+-----+
| CURRENT_USER() |
+-----+
| jon@%          |
+-----+
1 row in set (0.00 sec)
```

これらのオプションの1つに対して必要な値を省略すると、次に示すようなエラーが生じます。

```
shell> mysql --host 85.224.35.45 --user
mysql: option '--user' requires an argument
```

この場合、`mysql` は、コマンド行で `--user` オプションのあとに何もなかったため、値を見つけれませんでした。ただし、使用される最後のオプションではないオプションの値を省略すると、想定外の別のエラーが生じます。

```
shell> mysql --host --user jon
ERROR 2005 (HY000): Unknown MySQL server host '--user' (1)
```

`mysql` はコマンド行で `--host` に続く任意の文字列をホスト名と想定するため、`--host --user` は `--host==user` と解釈され、クライアントは「`--user`」という名前のホストで稼働している MySQL サーバーに接続しようとします。

デフォルト値を持つオプションには、値を割り当てるときに常に等号が必要です。そうしないと、エラーが発生します。たとえば、MySQL サーバー `--log-error` オプションはデフォルト値 `host_name.err` を持ちます。ここで、`host_name` は MySQL が稼働しているホストの名前です。ホスト名が「`tonfisk`」であるコンピュータ上で MySQL を稼働しているとします。次のように `mysqld_safe` を呼び出した場合を考えます。

```
shell> mysqld_safe &
[1] 11699
shell> 080112 12:53:40 mysqld_safe Logging to '/usr/local/mysql/var/tonfisk.err'.
080112 12:53:40 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var
shell>
```

サーバーのシャットダウン後、次のように再起動します。

```
shell> mysqld_safe --log-error &
[1] 11699
shell> 080112 12:53:40 mysqld_safe Logging to '/usr/local/mysql/var/tonfisk.err'.
080112 12:53:40 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var
shell>
```

コマンド行で `--log-error` にはほかに何も続いておらず、独自のデフォルト値が供給されるため、結果は同じです。(& 文字は、オペレーティングシステムに対して MySQL をバックグラウンドで実行することを指示します。MySQL 自身はこれを無視します。) ここで、エラーを `my-errors.err` という名前のファイルに記録するとします。 `--log-error my-errors` でサーバーを起動しようとする可能性があります。これは次に示すように、意図した効果を持ちません。

```
shell> mysqld_safe --log-error my-errors &
[1] 31357
shell> 080111 22:53:31 mysqld_safe Logging to '/usr/local/mysql/var/tonfisk.err'.
080111 22:53:32 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var
080111 22:53:34 mysqld_safe mysqld from pid file /usr/local/mysql/var/tonfisk.pid ended

[1]+ Done                ./mysqld_safe --log-error my-errors
```

サーバーは `/usr/local/mysql/var/tonfisk.err` をエラーログとして起動しようとしたが、シャットダウンしました。このファイルの最後の数行を調べると理由がわかります。

```
shell> tail /usr/local/mysql/var/tonfisk.err
2013-09-24T15:36:22.278034Z 0 [ERROR] Too many arguments (first extra is 'my-errors').
2013-09-24T15:36:22.278059Z 0 [Note] Use --verbose --help to get a list of available options!
2013-09-24T15:36:22.278076Z 0 [ERROR] Aborting
2013-09-24T15:36:22.279704Z 0 [Note] InnoDB: Starting shutdown...
2013-09-24T15:36:23.777471Z 0 [Note] InnoDB: Shutdown completed; log sequence number 2319086
2013-09-24T15:36:23.780134Z 0 [Note] mysqld: Shutdown complete
```

`--log-error` オプションはデフォルト値を提供するため、次に示すように等号を使用して別の値を割り当てる必要があります:


```
shell> mysqld_safe --log-error=my-errors &
[1] 31437
shell> 080111 22:54:15 mysqld_safe Logging to '/usr/local/mysql/var/my-errors.err'.
080111 22:54:15 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var

shell>
```

今度はサーバーは正常に起動し、エラーをファイル `/usr/local/mysql/var/my-errors.err` にロギングしています。

オプションファイルでオプション値を指定する場合も、同様の問題が生じる可能性があります。たとえば、次の内容の `my.cnf` ファイルを考えます。

```
[mysql]

host
user
```

`mysql` クライアントがこのファイルを読み取ると、これらのエントリは `--host --user` または `--host=--user` と解釈され、次に示すような結果になります。

```
shell> mysql
ERROR 2005 (HY000): Unknown MySQL server host '--user' (1)
```

ただし、オプションファイルでは等号は想定されません。 `my.cnf` ファイルが次に示すようになっているとします。

```
[mysql]

user jon
```

この場合、`mysql` を起動しようとするると別のエラーになります。

```
shell> mysql
mysql: unknown option '--user jon'
```

`host=tonfisk` ではなく `host tonfisk` とオプションファイルに記述した場合も同様のエラーが生じます。かわりに、等号を使用する必要があります:

```
[mysql]

user=jon
```

これでログインが成功します。

```
shell> mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5
Server version: 8.0.29 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT USER();
+-----+
| USER() |
+-----+
| jon@localhost |
+-----+
1 row in set (0.00 sec)
```

これは、等号が不要なコマンドラインと同じ動作ではありません:

```
shell> mysql --user jon --host tonfisk
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 8.0.29 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT USER();
+-----+
| USER() |
+-----+
| jon@tonfisk |
```

```
+-----+  
1 row in set (0.00 sec)
```

オプションファイルに値のない値を必要とするオプションを指定すると、サーバーはエラーで中止されます。

4.2.3 サーバーに接続するためのコマンドオプション

このセクションでは、クライアントプログラムがサーバーへの接続を確立する方法、接続を暗号化するかどうか、および接続を圧縮するかどうかを制御する、ほとんどの MySQL クライアントプログラムでサポートされているオプションについて説明します。これらのオプションは、コマンド行またはオプションファイルで指定できます。

- [接続確立のコマンドオプション](#)
- [暗号化接続のコマンドオプション](#)
- [接続圧縮のコマンドオプション](#)

接続確立のコマンドオプション

このセクションでは、クライアントプログラムがサーバーへの接続を確立する方法を制御するオプションについて説明します。使用方法の詳細および例は、[セクション4.2.4「コマンドオプションを使用した MySQL Server への接続」](#)を参照してください。

表 4.3 「接続確立オプションサマリー」

オプション名	説明
--default-auth	使用する認証プラグイン
--host	MySQL サーバーがあるホスト
--password	サーバーに接続する際に使用するパスワード
--pipe	名前付きパイプを使用してサーバに接続する (Windows のみ)
--plugin-dir	プラグインがインストールされているディレクトリ
--port	接続用の TCP/IP ポート番号
--protocol	使用するトランスポートプロトコル
--shared-memory-base-name	共有メモリ-接続用の共有メモリ名 (Windows のみ)
--socket	使用する Unix ソケットファイルまたは Windows 名前付きパイプ
--user	サーバーへの接続時に使用する MySQL ユーザー名

- [--default-auth=plugin](#)

使用するクライアント側認証プラグインに関するヒント。 [セクション6.2.17「プラグブル認証」](#)を参照してください。

- [--host=host_name, -h host_name](#)

MySQL サーバーが実行されているホスト。 値には、ホスト名、IPv4 アドレスまたは IPv6 アドレスを指定できません。 デフォルト値は `localhost` です。

- [--password=\[pass_val\], -p\[pass_val\]](#)

サーバーへの接続に使用される MySQL アカウントのパスワード。 パスワード値はオプションです。 指定しない場合、プログラムによってプロンプトが表示されます。 指定する場合は、`--password=` または `-p` とそれに続くパスワードの間にスペースなしが存在する必要があります。 パスワードオプションを指定しない場合、デフォルトではパスワードは送信されません。

コマンド行でのパスワード指定は、セキュアでないと考えるべきです。 コマンド行でパスワードを指定しないようにするには、オプションファイルを使用します。 [セクション6.1.2.1「パスワードセキュリティのためのエンドユーザーガイドライン」](#)を参照してください。

パスワードがないこと、およびクライアントプログラムがパスワードを要求しないことを明示的に指定するには、`--skip-password` オプションを使用します。

- `--pipe, -W`

Windows で、名前付きパイプを使用してサーバーに接続します。このオプションは、ネームパイプ接続をサポートするために `named_pipe` システム変数を有効にしてサーバーを起動した場合にのみ適用されます。また、接続を行うユーザーは、`named_pipe_full_access_group` システム変数で指定された Windows グループのメンバーである必要があります。

- `--plugin-dir=dir_name`

プラグインを検索するディレクトリ。このオプションは、`--default-auth` オプションを使用して認証プラグインを指定しても、クライアントプログラムがそれを検出しない場合に指定します。セクション6.2.17「プラグイン認証」を参照してください。

- `--port=port_num, -P port_num`

TCP/IP 接続の場合、使用するポート番号。デフォルトのポート番号は 3306 です。

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

このオプションは、サーバーへの接続に使用するトランスポートプロトコルを明示的に指定します。これは、他の接続パラメータが通常、必要なプロトコル以外のプロトコルを使用する場合に便利です。たとえば、Unix では `localhost` への接続はデフォルトでは Unix ソケットファイルを使用して行われます。

```
mysql --host=localhost
```

かわりに TCP/IP トランスポートを強制的に使用するには、`--protocol` オプションを指定します:

```
mysql --host=localhost --protocol=TCP
```

次のテーブルに、許容される `--protocol` オプション値を示し、各値に適用可能なプラットフォームを示します。値では大文字と小文字は区別されません。

<code>--protocol</code> の値	使用されるトランスポートプロトコル	適用可能なプラットフォーム
TCP	ローカルサーバーまたはリモートサーバーへの TCP/IP トランスポート	すべて
SOCKET	ローカルサーバーへの Unix ソケットファイルトランスポート	Unix および Unix に似たシステム
PIPE	ローカルサーバーへの名前付きパイプトランスポート	Windows
MEMORY	ローカルサーバーへの共有メモリートランスポート	Windows

セクション4.2.7「接続トランスポートプロトコル」も参照してください

- `--shared-memory-base-name=name`

Windows の場合、共有メモリを使用してローカルサーバーに接続するために使用する共有メモリ名。デフォルト値は `MYSQL` です。共有メモリ名では大文字と小文字が区別されます。

このオプションは、共有メモリー接続をサポートするために `shared_memory` システム変数を有効にしてサーバーを起動した場合にのみ適用されます。

- `--socket=path, -S path`

Unix の場合、名前付きパイプを使用してローカルサーバーに接続するために使用する Unix ソケットファイルの名前。デフォルトの Unix ソケットファイル名は `/tmp/mysql.sock` です。

Windows の場合、ローカルサーバーへの接続に使用する名前付きパイプの名前。デフォルトの Windows パイプ名は `MySQL` です。パイプ名では大文字と小文字は区別されません。

Windows では、このオプションは、名前付きパイプ接続をサポートするために `named_pipe` システム変数を有効にしてサーバーを起動した場合にのみ適用されます。また、接続を行うユーザーは、`named_pipe_full_access_group` システム変数で指定された Windows グループのメンバーである必要があります。

- `--user=user_name, -u user_name`

サーバーへの接続に使用する MySQL アカウントのユーザー名。デフォルトのユーザー名は、Windows では `ODBC`、Unix では Unix のログイン名です。

暗号化接続のコマンドオプション

このセクションでは、サーバーへの暗号化された接続を使用するかどうか、証明書とキーファイルの名前、および暗号化された接続のサポートに関連するその他のパラメータを指定するクライアントプログラムのオプションについて説明します。推奨される使用例および接続が暗号化されているかどうかを確認する方法については、[セクション 6.3.1 「暗号化接続を使用するための MySQL の構成」](#) を参照してください。

注記

これらのオプションは、暗号化の対象となるトランスポートプロトコル (TCP/IP および Unix ソケットファイル接続) を使用する接続に対してのみ有効です。[セクション 4.2.7 「接続トランスポートプロトコル」](#) を参照してください

MySQL C API からの暗号化された接続の使用の詳細は、[Support for Encrypted Connections](#) を参照してください。

表 4.4 「接続暗号化オプションのサマリー」

オプション名	説明	導入
<code>--get-server-public-key</code>	サーバーから RSA 公開キーをリクエスト	
<code>--server-public-key-path</code>	RSA 公開鍵を含むファイルへのパス名	
<code>--ssl-ca</code>	信頼できる SSL 認証局のリストを含むファイル	
<code>--ssl-capath</code>	信頼できる SSL 認証局の証明書ファイルを含むディレクトリ	
<code>--ssl-cert</code>	X.509 証明書を含むファイル	
<code>--ssl-cipher</code>	接続の暗号化に許可される暗号	
<code>--ssl-crl</code>	証明書失効リストを含むファイル	
<code>--ssl-crlpath</code>	証明書失効リストファイルを含むディレクトリ	
<code>--ssl-fips-mode</code>	クライアント側で FIPS モードを有効にするかどうか	
<code>--ssl-key</code>	X.509 キーを含むファイル	
<code>--ssl-mode</code>	サーバーへの接続に必要なセキュリティ状態	
<code>--tls-ciphersuites</code>	暗号化された接続に許可される TLSv1.3 暗号スイート	8.0.16
<code>--tls-version</code>	暗号化された接続に許可される TLS プロトコル	

- `--get-server-public-key`

RSA キーペアベースのパスワード交換に必要な公開キーをサーバーにリクエストします。このオプションは、 `caching_sha2_password` 認証プラグインで認証されるクライアントに適用されます。そのプラグインの場合、サーバーは要求されないかぎり公開鍵を送信しません。このオプションは、そのプラグインで認証されないアカウントでは無視されます。クライアントがセキュアな接続を使用してサーバーに接続する場合と同様に、RSA ベースのパスワード交換を使用しない場合も無視されます。

`--server-public-key-path=file_name` が指定され、有効な公開キーファイルが指定されている場合は、`--get-server-public-key` よりも優先されます。

`caching_sha2_password` プラグインの詳細は、[セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#) を参照してください。

- `--server-public-key-path=file_name`

RSA キーペアベースのパスワード交換のためにサーバーが必要とする公開キーのクライアント側コピーを含む、PEM 形式のファイルへのパス名。このオプションは、 `sha256_password` または `caching_sha2_password` 認証プラグインで認証されるクライアントに適用されます。これらのプラグインのいずれかで認証されないアカウントでは、このオプションは無視されます。クライアントがセキュアな接続を使用してサーバーに接続する場合と同様に、RSA ベースのパスワード交換を使用しない場合も無視されます。

`--server-public-key-path=file_name` が指定され、有効な公開キーファイルが指定されている場合は、`--get-server-public-key` よりも優先されます。

このオプションは、MySQL が OpenSSL を使用してビルドされている場合のみ利用できます。

`sha256_password` および `caching_sha2_password` プラグインの詳細は、[セクション6.4.1.3「SHA-256 プラガブル認証」](#) および [セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#) を参照してください。

- `--ssl-ca=file_name`

PEM 形式の認証局 (CA) 証明書ファイルのパス名。このファイルには、信頼できる SSL 認証局のリストが含まれています。

サーバーへの暗号化された接続を確立するときにサーバー証明書を認証しないようにクライアントに指示するには、`--ssl-ca` も `--ssl-capath` も指定しません。サーバーは、クライアントアカウントに設定されている適用可能な要件に従ってクライアントを検証し、サーバー側で指定されている `ssl_ca` または `ssl_capath` システム変数値を引き続き使用します。

サーバーの CA ファイルを指定するには、 `ssl_ca` システム変数を設定します。

- `--ssl-capath=dir_name`

PEM 形式の信頼できる SSL 認証局 (CA) 証明書ファイルを含むディレクトリのパス名。

サーバーへの暗号化された接続を確立するときにサーバー証明書を認証しないようにクライアントに指示するには、`--ssl-ca` も `--ssl-capath` も指定しません。サーバーは、クライアントアカウントに設定されている適用可能な要件に従ってクライアントを検証し、サーバー側で指定されている `ssl_ca` または `ssl_capath` システム変数値を引き続き使用します。

サーバーの CA ディレクトリを指定するには、 `ssl_capath` システム変数を設定します。

- `--ssl-cert=file_name`

PEM 形式のクライアント SSL 公開キー証明書ファイルのパス名。

サーバー SSL 公開キー証明書ファイルを指定するには、 `ssl_cert` システム変数を設定します。

- `--ssl-cipher=cipher_list`

TLSv1.2までのTLS プロトコルを使用する接続に許可される暗号のリスト。リスト内の暗号がサポートされていない場合、これらの TLS プロトコルを使用する暗号化された接続は機能しません。

移植性を最大にするには、`cipher_list` をコロンで区切った 1 つ以上の暗号名のリストで指定するようにしてください。例:

```
--ssl-cipher=AES128-SHA  
--ssl-cipher=DHE-RSA-AES128-GCM-SHA256:AES128-SHA
```

OpenSSL は、<https://www.openssl.org/docs/manmaster/man1/ciphers.html> の OpenSSL ドキュメントで説明されている暗号を指定するための構文をサポートしています。

MySQL がサポートする暗号化暗号の詳細は、[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#) を参照してください。

サーバーの暗号を指定するには、`ssl_cipher` システム変数を設定します。

- `--ssl-crl=file_name`

PEM 形式の証明書失効リストを含むファイルのパス名。

`--ssl-crl` も `--ssl-crlpath` も指定されていない場合は、CA パスに証明書失効リストが含まれていても、CRL チェックが実行されません。

サーバーの失効リストファイルを指定するには、`ssl_crl` システム変数を設定します。

- `--ssl-crlpath=dir_name`

PEM 形式の証明書失効リストファイルを含むディレクトリのパス名。

`--ssl-crl` も `--ssl-crlpath` も指定されていない場合は、CA パスに証明書失効リストが含まれていても、CRL チェックが実行されません。

サーバーの失効リストディレクトリを指定するには、`ssl_crlpath` システム変数を設定します。

- `--ssl-fips-mode={OFF|ON|STRICT}`

クライアント側で FIPS モードを有効にするかどうかを制御します。`--ssl-fips-mode` オプションは、暗号化された接続の確立には使用されず、許可する暗号化操作に影響する点で、他の `--ssl-xxx` オプションとは異なります。[セクション6.8「FIPS のサポート」](#) を参照してください。

次の `--ssl-fips-mode` 値を使用できます:

- **OFF**: FIPS モードを無効にします。
- **ON**: FIPS モードを有効にします。
- **STRICT**: 「strict」 FIPS モードを有効にします。

注記

OpenSSL FIPS オブジェクトモジュールが使用できない場合、`--ssl-fips-mode` で許可される値は **OFF** のみです。この場合、`--ssl-fips-mode` を **ON** または **STRICT** に設定すると、クライアントは起動時に警告を生成し、FIPS 以外のモードで動作します。

サーバーの FIPS モードを指定するには、`ssl_fips_mode` システム変数を設定します。

- `--ssl-key=file_name`

PEM 形式のクライアント SSL 秘密キーファイルのパス名。セキュリティを向上させるには、RSA キーサイズが 2048 ビット以上の証明書を使用します。

鍵ファイルがパスフレーズで保護されている場合、クライアントプログラムはユーザーにパスフレーズの入力を求めます。パスワードは対話形式で指定する必要があり、ファイルには格納できません。パスフレーズが正しくない場合は、鍵を読み取ることができない場合と同様に、プログラムが続行されます。

サーバー SSL 秘密キーファイルを指定するには、`ssl_key` システム変数を設定します。

- `--ssl-mode=mode`

このオプションは、サーバーへの接続の目的のセキュリティー状態を指定します。これらのモード値は、厳密性の向上順に許容されます:

- **DISABLED**: 非暗号化接続を確立します。
- **PREFERRED**: サーバーが暗号化された接続をサポートしている場合は、暗号化された接続を確立します。暗号化された接続を確立できない場合は、暗号化されていない接続にフォールバックします。これは、`--ssl-mode` が指定されていない場合のデフォルトです。

Unix ソケットファイルを介した接続は、**PREFERRED** モードでは暗号化されません。Unix ソケットファイル接続の暗号化を強制するには、**REQUIRED** または厳密なモードを使用します。(ただし、ソケットファイルトランスポートはデフォルトでセキュアであるため、ソケットファイル接続を暗号化するとセキュアでなくなり、CPU 負荷が増加します。)

- **REQUIRED**: サーバーが暗号化された接続をサポートしている場合は、暗号化された接続を確立します。暗号化された接続を確立できない場合、接続は失敗します。
- **VERIFY_CA**: **REQUIRED** と似ていますが、構成された CA 証明書に対してサーバー認証局 (CA) 証明書を追加で検証します。一致する有効な CA 証明書が見つからない場合、接続は失敗します。
- **VERIFY_IDENTITY**: **VERIFY_CA** と同様ですが、サーバーがクライアントに送信する証明書のアイデンティティと照合して、クライアントがサーバーへの接続に使用するホスト名をチェックして、ホスト名のアイデンティティ検証を追加で実行します:
 - MySQL 8.0.12 では、クライアントが OpenSSL 1.0.2 以上を使用する場合、クライアントは接続に使用するホスト名がサーバー証明書のサブジェクト代替名の値または共通名の値と一致するかどうかを確認します。ホスト名のアイデンティティ検証は、ワイルドカードを使用して共通名を指定する証明書でも機能します。
 - それ以外の場合、クライアントは接続に使用するホスト名がサーバー証明書の共通名の値と一致するかどうかを確認します。

不一致がある場合、接続は失敗します。暗号化された接続の場合、このオプションは中間者攻撃を防ぐのに役立ちます。

注記

VERIFY_IDENTITY を使用したホスト名アイデンティティ検証は、サーバーによって自動的に作成された自己署名証明書、または `mysql_ssl_rsa_setup` を使用して手動で作成された自己署名証明書では機能しません ([セクション 6.3.3.1 「MySQL を使用した SSL](#)

「[および RSA 証明書とキーの作成](#)」を参照)。このような自己署名証明書には、共通名の値としてサーバー名は含まれません。

`--ssl-mode` オプションは、次のように CA 証明書オプションと対話します:

- `--ssl-mode` が明示的に設定されていない場合、`--ssl-ca` または `--ssl-capath` の使用は `--ssl-mode=VERIFY_CA` を意味します。
- `VERIFY_CA` または `VERIFY_IDENTITY` の `--ssl-mode` 値の場合、`--ssl-ca` または `--ssl-capath` も、サーバーで使用する CA 証明書と一致する CA 証明書を指定する必要があります。
- `VERIFY_CA` または `VERIFY_IDENTITY` 以外の値を持つ明示的な `--ssl-mode` オプションを明示的な `--ssl-ca` または `--ssl-capath` オプションとともに使用すると、CA 証明書オプションが指定されているにもかかわらず、サーバー証明書の検証が実行されないという警告が生成されます。

MySQL アカウントで暗号化された接続を使用する必要がある場合は、`CREATE USER` を使用して `REQUIRE SSL` 句でアカウントを作成するか、既存のアカウントに `ALTER USER` を使用して `REQUIRE SSL` 句を追加します。これにより、MySQL が暗号化された接続をサポートし、暗号化された接続を確立できないかぎり、アカウントを使用するクライアントによる接続試行が拒否されます。

`REQUIRE` 句では、他の暗号化関連のオプションを使用して、`REQUIRE SSL` より厳格なセキュリティ要件を適用できます。様々な `REQUIRE` オプションを使用して構成されたアカウントを使用して接続するクライアントで指定する必要があるコマンドオプションの詳細は、[CREATE USER SSL/TLS オプション](#) を参照してください。

- `--tls-ciphersuites=ciphersuite_list`

このオプションは、TLSv1.3 を使用する暗号化された接続に対してクライアントが許可する暗号スイートを指定します。値は、コロンで区切られたゼロ個以上の暗号スイート名のリストです。例:

```
mysql --tls-ciphersuites="suite1:suite2:suite3"
```

このオプションに指定できる暗号スイートは、MySQL のコンパイルに使用される SSL ライブラリによって異なります。このオプションが設定されていない場合、クライアントは暗号化方式群のデフォルトセットを許可します。このオプションが空の文字列に設定されている場合、暗号スイートは有効にならず、暗号化された接続を確立できません。詳細は、[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#)を参照してください。

このオプションは MySQL 8.0.16 で追加されました。

サーバーが許可する暗号化方式群を指定するには、`tls_ciphersuites` システム変数を設定します。

- `--tls-version=protocol_list`

このオプションは、クライアントが暗号化された接続に対して許可する TLS プロトコルを指定します。この値は、1 つまたは複数のコンマ区切りプロトコルバージョンのリストです。例:

```
mysql --tls-version="TLSv1.1,TLSv1.2"
```

このオプションに指定できるプロトコルは、MySQL のコンパイルに使用される SSL ライブラリによって異なります。「穴」をリストに残さないなど、許可されたプロトコルを選択する必要があります。たとえば、次の値には穴がありません:

```
--tls-version="TLSv1,TLSv1.1,TLSv1.2,TLSv1.3"  
--tls-version="TLSv1.1,TLSv1.2,TLSv1.3"  
--tls-version="TLSv1.2,TLSv1.3"  
--tls-version="TLSv1.3"
```

これらの値には穴があり、使用しないでください:

```
--tls-version="TLSv1,TLSv1.2"  
--tls-version="TLSv1.1,TLSv1.3"
```

詳細は、[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#)を参照してください。

サーバーが許可する TLS プロトコルを指定するには、`tls_version` システム変数を設定します。

接続圧縮のコマンドオプション

このセクションでは、クライアントプログラムがサーバーへの接続の圧縮の使用を制御できるようにするオプションについて説明します。使用方法の詳細および例は、[セクション4.2.8「接続圧縮制御」](#)を参照してください。

表 4.5 「接続圧縮オプションのサマリー」

オプション名	説明	導入	非推奨
<code>--compress</code>	クライアントとサーバー間で送信される情報をすべて圧縮		8.0.18
<code>--compression-algorithms</code>	サーバーへの接続に許可される圧縮アルゴリズム	8.0.18	
<code>--zstd-compression-level</code>	zstd 圧縮を使用するサーバーへの接続の圧縮レベル	8.0.18	

- `--compress, -C`

可能であれば、クライアントとサーバーの間で送信されるすべての情報を圧縮します。

MySQL 8.0.18 では、このオプションは非推奨です。MySQL の将来のバージョンで削除されることが予想されます。[レガシー接続圧縮の構成](#)を参照してください。

- `--compression-algorithms=value`

サーバーへの接続に許可される圧縮アルゴリズム。使用可能なアルゴリズムは、`protocol_compression_algorithms` システム変数の場合と同じです。デフォルト値は `uncompressed` です。

このオプションは MySQL 8.0.18 で追加されました。

- `--zstd-compression-level=level`

zstd 圧縮アルゴリズムを使用するサーバーへの接続に使用する圧縮レベル。許可されるレベルは 1 から 22 で、大きい値は圧縮レベルの増加を示します。デフォルトの zstd 圧縮レベルは 3 です。圧縮レベルの設定は、zstd 圧縮を使用しない接続には影響しません。

このオプションは MySQL 8.0.18 で追加されました。

4.2.4 コマンドオプションを使用した MySQL Server への接続

このセクションでは、コマンドラインオプションを使用して、`mysql` や `mysqldump` などのクライアントに対して MySQL サーバーへの接続を確立する方法を指定する方法について説明します。URI のような接続文字列またはキーと値のペアを使用して MySQL Shell などのクライアントに対して接続を確立する方法の詳細は、[セクション 4.2.5「URI 類似文字列またはキーと値のペアを使用したサーバーへの接続」](#)を参照してください。接続できない場合の追加情報は、[セクション6.2.21「MySQL への接続の問題のトラブルシューティング」](#)を参照してください。

クライアントプログラムが MySQL サーバーに接続するには、サーバーが実行されているホストの名前や MySQL アカウントのユーザー名とパスワードなど、適切な接続パラメータを使用する必要があります。各接続パラメータにはデフォルト値がありますが、コマンド行またはオプションファイルで指定されたプログラムオプションを使用して、必要に応じてデフォルト値をオーバーライドできます。

次の例は `mysql` クライアントプログラムを使用しますが、原則は `mysqldump`、`mysqladmin`、または `mysqlshow` など、ほかのクライアントにも適用されます。

このコマンドは、明示的な接続パラメータを指定せずに `mysql` を起動します:

```
mysql
```

パラメータオプションがないため、デフォルト値が適用されます。

- デフォルトのホスト名は `localhost` です。Unix では、後述するようにこれには特別な意味があります。

- デフォルトのユーザー名は、Windows では ODBC、Unix では Unix のログイン名です。
- `--password` も `-p` も指定されていないため、パスワードは送信されません。
- `mysql` では、最初のオプションではない引数はデフォルトデータベースの名前とみなされます。このような引数はないため、`mysql` はデフォルトのデータベースを選択しません。

ホスト名、ユーザー名およびパスワードを明示的に指定するには、コマンドラインで適切なオプションを指定します。デフォルトのデータベースを選択するには、`database-name` 引数を追加します。例:

```
mysql --host=localhost --user=myname --password=password mydb
mysql -h localhost -u myname -ppassword mydb
```

パスワードオプションについては、パスワード値はオプションです。

- `--password` または `-p` オプションを使用してパスワード値を指定する場合は、`--password=` または `-p` とそれに続くパスワードの間にスペースなしが存在する必要があります。
- `--password` または `-p` を使用するが、パスワード値を指定しない場合、クライアントプログラムはパスワードの入力を求めます。パスワードは入力時に表示されません。これは、システム上の他のユーザーが `ps` などのコマンドを実行してパスワード行を表示できるようにするコマンド行でパスワードを指定するよりも安全です。セクション 6.1.2.1「パスワードセキュリティのためのエンドユーザーガイドライン」を参照してください。
- パスワードがないこと、およびクライアントプログラムがパスワードを要求しないことを明示的に指定するには、`--skip-password` オプションを使用します。

前述したように、コマンドラインにパスワード値を含めるとセキュリティ上のリスクがあります。このリスクを回避するには、次のパスワード値なしで `--password` または `-p` オプションを指定します:

```
mysql --host=localhost --user=myname --password mydb
mysql -h localhost -u myname -p mydb
```

パスワード値を指定せずに `--password` または `-p` オプションを指定すると、クライアントプログラムはプロンプトを出力し、パスワードの入力を待機します。(これらの例では、`mydb` はその前のパスワードオプションとスペースで区切られているため、パスワードとは解釈されません。)

システムによっては、MySQL がパスワードを要求するために使用するライブラリルーチングが、自動的にパスワードを 8 文字に制限します。この制限は、MySQL ではなくシステムライブラリのプロパティです。内部的には、MySQL にはパスワードの長さに関する制限はありません。影響を受けるシステムの制限を回避するには、オプションファイルでパスワードを指定します (セクション 4.2.2.2「オプションファイルの使用」を参照)。別の回避策として、MySQL パスワードを 8 文字以下の値に変更しますが、パスワードが短いほど安全性が低くなる傾向があります。

クライアントプログラムは、次のように接続のタイプを決定します:

- ホストが指定されていないか、`localhost` である場合、ローカルホストへの接続が発生します:
 - Windows では、共有メモリ接続をサポートするために `shared_memory` システム変数を有効にしてサーバーを起動した場合、クライアントは共有メモリを使用して接続します。
 - Unix では、MySQL プログラムは、他のネットワークベースのプログラムと比較して予想とは異なる方法で、ホスト名 `localhost` を特別に処理: クライアントは Unix ソケットファイルを使用して接続します。ソケット名の指定には、`--socket` オプションまたは `MYSQL_UNIX_PORT` 環境変数を使用できます。
- Windows では、`host` が . (ピリオド) であるか、TCP/IP が有効でなく、`--socket` が指定されていないか、ホストが空の場合、`named_pipe` システム変数を有効にしてサーバーが起動され、名前付きパイプ接続がサポートされている場合、クライアントは名前付きパイプを使用して接続します。名前付きパイプ接続がサポートされていない場合、または接続を行うユーザーが `named_pipe_full_access_group` システム変数で指定された Windows グループのメンバーでない場合は、エラーが発生します。
- それ以外の場合、接続では TCP/IP が使用されます。

`--protocol` オプションを使用すると、ほかのオプションが通常は別のプロトコルを使用する場合でも、特定のトランスポートプロトコルを使用できます。つまり、`--protocol` はトランスポートプロトコルを明示的に指定し、`localhost` の場合でも前述のルールをオーバーライドします。

選択したトランスポートプロトコルに関連する接続オプションのみが使用または選択されます。その他の接続オプションは無視されます。たとえば、Unix 上の `--host=localhost` では、TCP/IP のポート番号を指定するために `--port` または `-P` オプションが指定されている場合でも、クライアントは Unix ソケットファイルを使用してローカルサーバーへの接続を試みます。

クライアントがローカルサーバーに TCP/IP 接続するには、`--host` または `-h` を使用して、[127.0.0.1](#) (`localhost` ではなく) のホスト名の値、またはローカルサーバーの IP アドレスまたは名前を指定します。`localhost` の場合でも、`--protocol=TCP` オプションを使用してトランスポートプロトコルを明示的に指定することもできます。例:

```
mysql --host=127.0.0.1
mysql --protocol=TCP
```

サーバーが IPv6 接続を受け入れるように構成されている場合、クライアントは `--host>:::1` を使用して IPv6 経由でローカルサーバーに接続できます。[セクション5.1.13「IPv6 サポート」](#) を参照してください。

Windows で、MySQL クライアントで名前付きパイプ接続を強制的に使用するには、`--pipe` または `--protocol=PIPE` オプションを指定するか、ホスト名として `.` (ピリオド) を指定します。名前付きパイプ接続をサポートするために `named_pipe` システム変数を有効にしてサーバーを起動しなかった場合、または接続を行うユーザーが `named_pipe_full_access_group` システム変数で指定された Windows グループのメンバーでない場合は、エラーが発生します。デフォルトのパイプ名を使用しない場合には `--socket` オプションを使用してパイプ名を指定します。

リモートサーバーへの接続では、TCP/IP が使用されます。次のコマンドは、デフォルトのポート番号 (3306) を使用して `remote.example.com` で動作するサーバーに接続します。

```
mysql --host=remote.example.com
```

ポート番号を明示的に指定するには、`--port` オプションまたは `-P` オプションを使用します。

```
mysql --host=remote.example.com --port=13306
```

ローカルサーバーへの接続にもポート番号を指定できます。ただし、前述のように、Unix 上の `localhost` への接続ではデフォルトでソケットファイルが使用されるため、前述のように TCP/IP 接続を強制しないかぎり、ポート番号を指定するオプションは無視されます。

次のコマンドでは、Unix ではプログラムはソケットファイルを使用し、`--port` オプションは無視されます。

```
mysql --port=13306 --host=localhost
```

ポート番号を使用するには、TCP/IP 接続を強制します。たとえば、次のいずれかの方法でプログラムを起動します:

```
mysql --port=13306 --host=127.0.0.1
mysql --port=13306 --protocol=TCP
```

クライアントプログラムがサーバーへの接続を確立する方法を制御するオプションの詳細は、[セクション4.2.3「サーバーに接続するためのコマンドオプション」](#) を参照してください。

クライアントプログラムを起動するたびに、コマンドラインで接続パラメータを入力せずに指定できます:

- オプションファイルの `[client]` セクションで接続パラメータを指定します。このファイルの関係セクションは次のようになります。

```
[client]
host=host_name
user=user_name
password=password
```

詳細は、[セクション4.2.2.2「オプションファイルの使用」](#) を参照してください。

- 一部の接続パラメータは、環境変数を使用して指定できます。例:
 - `mysql` のホストを指定するには、`MYSQL_HOST` を使用します。
 - Windows で MySQL ユーザー名を指定するには、`USER` を使用します。

サポートされている環境変数のリストは、[セクション4.9「環境変数」](#) を参照してください。

4.2.5 URI 類似文字列またはキーと値のペアを使用したサーバーへの接続

このセクションでは、URI のような接続文字列またはキーと値のペアを使用して、MySQL Shell などのクライアントに対して MySQL サーバーへの接続を確立する方法を指定する方法について説明します。mysql や mysqldump などのクライアント用のコマンドラインオプションを使用した接続の確立の詳細は、[セクション4.2.4「コマンドオプションを使用した MySQL Server への接続」](#)を参照してください。接続できない場合の追加情報は、[セクション6.2.21「MySQL への接続の問題のトラブルシューティング」](#)を参照してください。

注記

「URI-like」という用語は、[RFC 3986](#) で定義されている URI (uniform resource identifier) 構文に似ているが同一ではない接続文字列構文を示します。

次の MySQL クライアントは、URI のような接続文字列またはキーと値のペアを使用した MySQL サーバーへの接続をサポートしています:

- MySQL Shell
- X DevAPI を実装する MySQL コネクタ

このセクションでは、すべての有効な URI 類似文字列およびキーと値のペアの接続パラメータについて説明します。これらの多くは、コマンドラインオプションで指定したものと似ています:

- URI のような文字列で指定されたパラメータは、`myuser@example.com:3306/main-schema` などの構文を使用します。完全な構文については、[URI 類似の接続文字列を使用した接続](#)を参照してください。
- キーと値のペアで指定されたパラメータは、`{user:'myuser', host:'example.com', port:3306, schema:'main-schema'}` などの構文を使用します。完全な構文については、[キーと値のペアを使用した接続](#)を参照してください。

接続パラメータでは、大/小文字は区別されません。各パラメータは、指定した場合、一度のみ指定できます。パラメータを複数回指定すると、エラーが発生します。

このセクションの内容は次のとおりです:

- [ベース接続パラメータ](#)
- [追加の接続パラメータ](#)
- [URI 類似の接続文字列を使用した接続](#)
- [キーと値のペアを使用した接続](#)

ベース接続パラメータ

次の説明では、MySQL への接続を指定するときに使用できるパラメータについて説明します。これらのパラメータは、ベース URI のような構文に準拠する文字列 ([URI 類似の接続文字列を使用した接続](#)を参照) またはキーと値のペア ([キーと値のペアを使用した接続](#)を参照) を使用して指定できます。

- `scheme`: 使用するトランスポートプロトコル。X プロトコル 接続には `mysqlx` を、クラシック MySQL プロトコル 接続には `mysql` を使用します。プロトコルが指定されていない場合、サーバーはプロトコルの推測を試みます。DNS SRV をサポートするコネクタは、`mysqlx+srv` スキームを使用できます ([Connections Using DNS SRV Records](#)を参照)。
- `user`: 認証プロセスに提供する MySQL ユーザーアカウント。
- `password`: 認証プロセスに使用するパスワード。

警告

接続指定で明示的なパスワードを指定することはセキュアでないため、お勧めしません。後の説明では、パスワードの対話型プロンプトを表示する方法を示します。

- `host`: サーバーインスタンスが実行されているホスト。値には、ホスト名、IPv4 アドレスまたは IPv6 アドレスを指定できます。ホストが指定されていない場合、デフォルトは `localhost` です。

- **port**: ターゲット MySQL サーバーが接続をリスニングしている TCP/IP ネットワークポート。ポートが指定されていない場合、デフォルトは、X プロトコル 接続の場合は 33060、クラシック MySQL プロトコル 接続の場合は 3306 です。
- **socket**: Unix ソケットファイルへのパスまたは Windows 名前付きパイプの名前。値はローカルファイルパスです。URI のような文字列では、パーセントエンコーディングを使用するか、パスをカッコで囲んでエンコードする必要があります。カッコを使用すると、/ディレクトリセパレータ文字などの文字をパーセントエンコードする必要がなくなります。たとえば、Unix ソケット/tmp/mysql.sock を使用して root@localhost として接続するには、root@localhost?socket=%2Ftmp%2Fmysql.sock としてパーセントエンコーディングを使用するか、root@localhost?socket=(/tmp/mysql.sock) としてカッコを使用してパスを指定します。
- **schema**: 接続のデフォルトデータベース。データベースが指定されていない場合、接続にはデフォルトのデータベースはありません。

Unix での localhost の処理は、トランスポートプロトコルのタイプによって異なります。クラシック MySQL プロトコルを使用した接続は、他の MySQL クライアントと同様に localhost を処理します。つまり、localhost はソケットベースの接続用であると想定されます。X プロトコルを使用した接続の場合、localhost の動作は、IPv4 アドレス 127.0.0.1 などのループバックアドレスを表すと想定される点で異なります。

追加の接続パラメータ

?attribute=value を追加して URI のような文字列の属性として、またはキーと値のペアとして、接続のオプションを指定できます。次のオプションを使用できます。

- **ssl-mode**: 接続に必要なセキュリティ状態。次のモードを使用できます:

- DISABLED
- PREFERRED
- REQUIRED
- VERIFY_CA
- VERIFY_IDENTITY

これらのモードの詳細は、[暗号化接続のコマンドオプション](#) の --ssl-mode オプションの説明を参照してください。

- **ssl-ca**: PEM 形式の X.509 認証局ファイルへのパス。
- **ssl-capath**: PEM 形式の X.509 認証局ファイルを含むディレクトリへのパス。
- **ssl-cert**: PEM 形式の X.509 証明書ファイルへのパス。
- **ssl-cipher**: TLSv1.2 までの TLS プロトコルを使用する接続に使用する暗号。
- **ssl-crl**: PEM 形式の証明書失効リストを含むファイルへのパス。
- **ssl-crlpath**: PEM 形式の証明書失効リストファイルを含むディレクトリへのパス。
- **ssl-key**: PEM 形式の X.509 キーファイルへのパス。
- **tls-version**: クラシック MySQL プロトコル の暗号化された接続に許可される TLS プロトコル。このオプションは、MySQL Shell でのみサポートされます。tls-version (単数形) の値は、TLSv1.1, TLSv1.2 などのカンマ区切りリストです。詳細は、[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#)を参照してください。このオプションは、DISABLED に設定されていない ssl-mode オプションに依存します。
- **tls-versions**: 暗号化された X プロトコル 接続に許可される TLS プロトコル。tls-versions (plural) の値は、[TLSv1.2, TLSv1.3]などの配列です。詳細は、[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#)を参照してください。このオプションは、DISABLED に設定されていない ssl-mode オプションに依存します。
- **tls-ciphersuites**: 許可される TLS 暗号スイート。tls-ciphersuites の値は、「TLS 暗号スイート」にリストされている IANA 暗号スイート名のリストです。詳細は、[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#)

号」を参照してください。このオプションは、`DISABLED` に設定されていない `ssl-mode` オプションに依存します。

- `auth-method`: 接続に使用する認証方式。デフォルトは `AUTO` で、サーバーは推測を試みます。次のメソッドを使用できます:
 - `AUTO`
 - `MYSQL41`
 - `SHA256_MEMORY`
 - `FROM_CAPABILITIES`
 - `FALLBACK`
 - `PLAIN`

X プロトコル 接続の場合、構成された `auth-method` は、この一連の認証方式にオーバーライドされます:
`MYSQL41`, `SHA256_MEMORY`, `PLAIN`.

- `get-server-public-key`: RSA キーペアベースのパスワード交換に必要な公開キーをサーバーにリクエストします。SSL モード `DISABLED` を使用してクラシック MySQL プロトコル 経由で MySQL 8.0 サーバーに接続する場合に使用します。この場合、プロトコルを指定する必要があります。例:

```
mysql://user@localhost:3306?get-server-public-key=true
```

このオプションは、`caching_sha2_password` 認証プラグインで認証されるクライアントに適用されます。そのプラグインの場合、サーバーは要求されないかぎり公開鍵を送信しません。このオプションは、そのプラグインで認証されないアカウントでは無視されます。クライアントがセキュアな接続を使用してサーバーに接続する場合と同様に、RSA ベースのパスワード交換を使用しない場合も無視されます。

`server-public-key-path=file_name` が指定され、有効な公開キーファイルが指定されている場合は、`get-server-public-key` よりも優先されます。

`caching_sha2_password` プラグインの詳細は、[セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#) を参照してください。

- `server-public-key-path`: RSA キーペアベースのパスワード交換のためにサーバーが必要とする公開キーのクライアント側コピーを含む、PEM 形式のファイルへのパス名。SSL モード `DISABLED` を使用してクラシック MySQL プロトコル 経由で MySQL 8.0 サーバーに接続する場合に使用します。

このオプションは、`sha256_password` または `caching_sha2_password` 認証プラグインで認証されるクライアントに適用されます。これらのプラグインのいずれかで認証されないアカウントでは、このオプションは無視されます。クライアントがセキュアな接続を使用してサーバーに接続する場合と同様に、RSA ベースのパスワード交換を使用しない場合も無視されます。

`server-public-key-path=file_name` が指定され、有効な公開キーファイルが指定されている場合は、`get-server-public-key` よりも優先されます。

`sha256_password` および `caching_sha2_password` プラグインの詳細は、[セクション6.4.1.3「SHA-256 プラガブル認証」](#) および [セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#) を参照してください。

- `connect-timeout`: クライアント (MySQL Shell など) が応答しない MySQL サーバーへの接続の試行を停止するまで待機する秒数を構成するために使用される整数値。
- `compression`: このオプションは、接続の圧縮をリクエストまたは無効化します。MySQL 8.0.19 までは、クラシック MySQL プロトコル 接続に対してのみ動作し、MySQL 8.0.20 からは X プロトコル 接続に対しても動作します。
 - MySQL 8.0.19 まで、このオプションの値は、圧縮を有効にする `true` (または 1) と、圧縮を無効にするデフォルトの `false` (または 0) です。
 - MySQL 8.0.20 からは、このオプションの値は `required` で、圧縮をリクエストしてサーバーがサポートしていない場合は失敗し、`preferred` では圧縮をリクエストして圧縮されていない接続にフォールバックし、`disabled` で

は圧縮されていない接続をリクエストしていない場合は失敗します。preferred は X プロトコル 接続のデフォルトで、disabled は クラシック MySQL プロトコル 接続のデフォルトです。X プラグイン の接続圧縮制御の詳細は、[セクション20.5.5「X プラグイン での接続圧縮」](#) を参照してください。MySQL クライアントごとに、接続圧縮のサポートが異なることに注意してください。詳細は、クライアントのドキュメントを参照してください。

- **compression-algorithms** および **compression-level**: これらのオプションは、接続の圧縮をより詳細に制御するために MySQL Shell 8.0.20 以降で使用できます。これらを指定して、接続に使用される圧縮アルゴリズムと、そのアルゴリズムで使用される数値圧縮レベルを選択できます。compression のかわりに compression-algorithms を使用して、接続の圧縮をリクエストすることもできます。MySQL Shell の接続圧縮制御の詳細は、[圧縮接続の使用](#) を参照してください。
- **connection-attributes**: 接続時にアプリケーションプログラムがサーバーに渡すキーと値のペアを制御します。接続属性の一般情報は、[セクション27.12.9「パフォーマンススキーマ接続属性テーブル」](#) を参照してください。クライアントは通常、属性のデフォルトセットを定義します。これは無効または有効にできます。例:

```
mysqlx://user@host?connection-attributes
mysqlx://user@host?connection-attributes=true
mysqlx://user@host?connection-attributes=false
```

デフォルトの動作では、デフォルトの属性セットが送信されます。アプリケーションでは、デフォルト属性に加えて渡される属性を指定できます。追加の接続属性は、接続文字列の connection-attributes パラメータとして指定します。connection-attributes パラメータ値は空 (true の指定と同じ)、Boolean 値 (デフォルトの属性セットを有効または無効にする true または false) またはカンマで区切られたリストまたはゼロ以上の key=value 指定子 (デフォルトの属性セットに加えて送信される) である必要があります。リスト内では、欠落しているキー値は空の文字列として評価されます。その他の例:

```
mysqlx://user@host?connection-attributes=[attr1=val1,attr2,attr3=]
mysqlx://user@host?connection-attributes=[]
```

アプリケーション定義の属性名は内部属性用に予約されているため、_ で始めることはできません。

URI 類似の接続文字列を使用した接続

URI のような文字列を使用して、MySQL Server への接続を指定できます。このような文字列は、MySQL Shell で --uri コマンドオプション、MySQL Shell \connect コマンドおよび X DevAPI を実装する MySQL コネクタとともに使用できます。

注記

「URI-like」という用語は、[RFC 3986](#) で定義されている URI (uniform resource identifier) 構文に似ているが同一ではない接続文字列構文を示します。

URI のような接続文字列の構文は次のとおりです:

```
[scheme://][user[:[password]]@][host[:port]][/schema][?attribute1=value1&attribute2=value2...
```

重要

URI のような文字列の要素の予約文字には、パーセントエンコーディングを使用する必要があります。たとえば、@ 文字を含む文字列を指定する場合は、その文字を %40 に置き換える必要があります。IPv6 アドレスにゾーン ID を含める場合は、セパレータとして使用される % 文字を %25 に置き換える必要があります。

URI のような接続文字列で使用できるパラメータは、[ベース接続パラメータ](#) で説明されています。

MySQL Shell shell.parseUri() および shell.unparseUri() メソッドを使用して、URI のような接続文字列を分解およびアセンブルできます。URI のような接続文字列が指定されると、shell.parseUri() は、文字列内で見つかった各要素を含むディクショナリを返します。shell.unparseUri() は、URI コンポーネントおよび接続オプションのディクショナリを、MySQL Shell または X DevAPI を実装する MySQL コネクタで使用できる MySQL に接続するための有効な URI のような接続文字列に変換します。

URI のような文字列にパスワードが指定されていない場合 (これをお勧めします)、対話型クライアントはパスワードの入力を求めます。次の例は、ユーザー名 user_name を使用して URI のような文字列を指定する方法を示しています。いずれの場合も、パスワードの入力を求められます。

- ポート 33065 でリスニングしているローカルサーバーインスタンスへの X プロトコル 接続。

```
mysqlx://user_name@localhost:33065
```

- ポート 3333 でリスニングしているローカルサーバーインスタンスへの クラシック MySQL プロトコル 接続。

```
mysql://user_name@localhost:3333
```

- ホスト名、IPv4 アドレスおよび IPv6 アドレスを使用したリモートサーバーインスタンスへの X プロトコル 接続。

```
mysqlx://user_name@server.example.com/
mysqlx://user_name@198.51.100.14:123
mysqlx://user_name@[2001:db8:85a3:8d3:1319:8a2e:370:7348]
```

- パーセントエンコーディングまたはカッコを使用してパスが指定されたソケットを使用した X プロトコル 接続。

```
mysqlx://user_name@/path%2Fto%2Fsocket.sock
mysqlx://user_name@(/path/to/socket.sock)
```

- データベースを表すオプションのパスを指定できます。

```
# use 'world' as the default database
mysqlx://user_name@198.51.100.1/world

# use 'world_x' as the default database, encoding _ as %5F
mysqlx://user_name@198.51.100.2:33060/world%5Fx
```

- オプションのクエリーを指定できます。各クエリーは、`key=value` ペアまたは単一の `key` として指定された値で構成されます。複数の値を指定するには、`,` 文字で区切ります。 `key=value` と `key` の値を混在させることができません。値はリストタイプで、リスト値は外観順に並べられます。文字列はパーセントでエンコードするか、カッコで囲む必要があります。次は同等です。

```
ssluser@127.0.0.1?ssl-ca=%2Froot%2Fclientcert%2Fca-cert.pem\
&ssl-cert=%2Froot%2Fclientcert%2Fclient-cert.pem\
&ssl-key=%2Froot%2Fclientcert%2Fclient-key

ssluser@127.0.0.1?ssl-ca=(/root/clientcert/ca-cert.pem)\
&ssl-cert=(/root/clientcert/client-cert.pem)\
&ssl-key=(/root/clientcert/client-key)
```

- 暗号化された接続に使用する TLS バージョンおよび暗号スイートを指定するには:

```
mysql://user_name@198.51.100.2:3306/world%5Fx?\
tls-versions=[TLSv1.2,TLSv1.3]&tls-ciphersuites=[TLS_DHE_PSK_WITH_AES_128_\
GCM_SHA256, TLS_CHACHA20_POLY1305_SHA256]
```

前述の例では、接続にパスワードが必要であることを前提としています。対話型クライアントでは、ログインプロンプトで指定されたユーザーパスワードが要求されます。ユーザーアカウントにパスワードがない場合 (セキュアではなく推奨されません)、またはソケットピア資格証明認証が使用されている場合 (Unix ソケット接続など)、接続文字列にパスワードが指定されておらず、パスワードプロンプトが不要であることを明示的に指定する必要があります。これを行うには、文字列の `user_name` の後に `:` を配置しますが、その後にパスワードを指定しないでください。例:

```
mysqlx://user_name:@localhost
```

キーと値のペアを使用した接続

MySQL Shell および X DevAPI を実装する一部の MySQL コネクタでは、実装用の言語自然構造で提供されるキーと値のペアを使用して MySQL Server への接続を指定できます。たとえば、キーと値のペアを JavaScript の JSON オブジェクトとして、または Python のディクショナリとして使用して、接続パラメータを指定できます。キーと値のペアの提供方法に関係なく、概念は変わりません: このセクションで説明するキーには、接続の指定に使用される値を割り当てることができます。MySQL Shell `shell.connect()` メソッドまたは InnoDB クラスタ `dba.createCluster()` メソッド、および X DevAPI を実装する MySQL コネクタの一部でキーと値のペアを使用して接続を指定できます。

通常、キーと値のペアは `{文字と}` 文字で囲まれ、`,` 文字はキーと値のペア間のセパレーターとして使用されます。`:` 文字はキーと値の間で使用され、文字列は (`'` 文字を使用するなどして) 区切る必要があります。URI のような接続文字列とは異なり、文字列をパーセントエンコードする必要はありません。

キーと値のペアとして指定される接続の形式は次のとおりです:

```
{ key: value, key: value, ...}
```

接続のキーとして使用できるパラメータは、[ベース接続パラメータ](#) で説明されています。

キーと値のペアにパスワードが指定されていない場合 (これをお勧めします)、対話型クライアントはパスワードの入力を求めます。次の例は、ユーザー名が 'user_name' のキーと値のペアを使用して接続を指定する方法を示しています。いずれの場合も、パスワードの入力を求められます。

- ポート 33065 でリスニングしているローカルサーバーインスタンスへの X プロトコル 接続。

```
{user:'user_name', host:'localhost', port:33065}
```

- ポート 3333 でリスニングしているローカルサーバーインスタンスへの クラシック MySQL プロトコル 接続。

```
{user:'user_name', host:'localhost', port:3333}
```

- ホスト名、IPv4 アドレスおよび IPv6 アドレスを使用したりリモートサーバーインスタンスへの X プロトコル 接続。

```
{user:'user_name', host:'server.example.com'}
{user:'user_name', host:'198.51.100.14:123'}
{user:'user_name', host:['2001:db8:85a3:8d3:1319:8a2e:370:7348']}
```

- ソケットを使用した X プロトコル 接続。

```
{user:'user_name', socket:'/path/to/socket/file'}
```

- データベースを表すオプションのスキーマを指定できます。

```
{user:'user_name', host:'localhost', schema:'world'}
```

前述の例では、接続にパスワードが必要であることを前提としています。対話型クライアントでは、ログインプロンプトで指定されたユーザーパスワードが要求されます。ユーザーアカウントにパスワードがない (セキュリティ保護されておらず、推奨されていない) 場合、またはソケットピア資格証明認証が使用されている (Unix ソケット接続など) 場合は、パスワードが提供されず、パスワードプロンプトが不要であることを明示的に指定する必要があります。これを行うには、`password` キーの後に " を使用して空の文字列を指定します。例:

```
{user:'user_name', password:"", host:'localhost'}
```

4.2.6 DNS SRV レコードを使用したサーバーへの接続

ドメインネームシステム (DNS) では、SRV レコード (サービスローケーションレコード) は、クライアントがサービス、プロトコルおよびドメインを示す名前を指定できるようにするリソースレコードのタイプです。名前の DNS ルックアップは、必要なサービスを提供するドメイン内の複数の使用可能なサーバーの名前を含む応答を返します。リストされたサーバーの優先順位をレコードで定義する方法など、DNS SRV の詳細は、[RFC 2782](#) を参照してください。

MySQL では、サーバーへの接続に DNS SRV レコードの使用がサポートされています。DNS SRV ルックアップ結果を受信するクライアントは、DNS 管理者によって各ホストに割り当てられた優先度および重みに基づいて、リストされた各ホスト上の MySQL サーバーへの接続を優先順位に従って試行します。接続の失敗は、クライアントがどのサーバーにも接続できない場合にのみ発生します。

サーバーのクラスタなどの複数の MySQL インスタンスがアプリケーションに同じサービスを提供する場合、DNS SRV レコードを使用してフェイルオーバー、ロードバランシングおよびレプリケーションサービスを支援できます。アプリケーションが接続試行の候補サーバーのセットを直接管理するのは面倒であり、DNS SRV レコードには次の代替方法があります:

- DNS SRV レコードを使用すると、DNS 管理者は単一の DNS ドメインを複数のサーバーにマップできます。DNS SRV レコードは、サーバーが構成に対して追加または削除されたとき、またはサーバーのホスト名が変更されたときに、管理者が一元的に更新することもできます。
- DNS SRV レコードを集中管理することで、個々のクライアントが接続リクエストで可能な各ホストを識別したり、追加のソフトウェアコンポーネントで接続を処理したりする必要がなくなります。アプリケーションでは、サーバー情報自体を管理するかわりに、DNS SRV レコードを使用して候補 MySQL サーバーに関する情報を取得できます。

- DNS SRV レコードは接続プーリングと組み合わせて使用できます。この場合、DNS SRV レコードの現在のリストにないホストへの接続は、アイドル状態になるとプールから削除されます。

MySQL では、DNS SRV レコードを使用して、次のコンテキストでサーバーに接続できます：

- いくつかの MySQL コネクタには、DNS SRV サポートが実装されています。コネクタ固有のオプションを使用すると、X プロトコル 接続とクラシック MySQL プロトコル 接続の両方に対して DNS SRV レコード参照をリクエストできます。一般的な情報は、[Connections Using DNS SRV Records](#) を参照してください。詳細は、個々の MySQL コネクタのドキュメントを参照してください。
- C API には、`mysql_real_connect()` に似た `mysql_real_connect_dns_srv()` 関数が用意されていますが、引数リストには接続先の MySQL サーバーの特定のホストが指定されていない点が異なります。かわりに、サーバーのグループを指定する DNS SRV レコードを指定します。`mysql_real_connect_dns_srv()` を参照してください。
- `mysql` クライアントには、サーバーのグループを指定する DNS SRV レコードを示す `--dns-srv-name` オプションがあります。[セクション4.5.1「mysql — MySQL コマンドラインクライアント」](#) を参照してください。

DNS SRV 名は、サービス、プロトコルおよびドメインで構成され、それぞれにアンダースコアが付いたサービスおよびプロトコルがあります：

```
_service._protocol.domain
```

次の DNS SRV レコードは、クライアントが X プロトコル 接続を確立するために使用するなど、複数の候補サーバーを識別します：

Name	TTL	Class	Priority	Weight	Port	Target
_mysqlx._tcp.example.com.	86400	IN	SRV 0	5	33060	server1.example.com.
_mysqlx._tcp.example.com.	86400	IN	SRV 0	10	33060	server2.example.com.
_mysqlx._tcp.example.com.	86400	IN	SRV 10	5	33060	server3.example.com.
_mysqlx._tcp.example.com.	86400	IN	SRV 20	5	33060	server4.example.com.

ここで、`mysqlx` は X プロトコル サービスを示し、`tcp` は TCP プロトコルを示します。クライアントは、`_mysqlx._tcp.example.com` という名前を使用してこの DNS SRV レコードをリクエストできます。接続リクエストで名前を指定するための具体的な構文は、クライアントのタイプによって異なります。たとえば、クライアントは URI のような接続文字列内での名前の指定、またはキーと値のペアとしての名前の指定をサポートできます。

クラシックプロトコル接続用の DNS SRV レコードは、次のようになります：

Name	TTL	Class	Priority	Weight	Port	Target
_mysql._tcp.example.com.	86400	IN	SRV 0	5	3306	server1.example.com.
_mysql._tcp.example.com.	86400	IN	SRV 0	10	3306	server2.example.com.
_mysql._tcp.example.com.	86400	IN	SRV 10	5	3306	server3.example.com.
_mysql._tcp.example.com.	86400	IN	SRV 20	5	3306	server4.example.com.

ここで、`mysql` という名前はクラシック MySQL プロトコル サービスを示し、ポートは 33060 (デフォルトの X プロトコル ポート) ではなく 3306 (デフォルトのクラシック MySQL プロトコル ポート) です。

DNS SRV レコード参照を使用する場合、クライアントは通常、接続リクエストに次のルールを適用する必要があります (クライアント固有またはコネクタ固有の例外があります)：

- リクエストには、サービス名とプロトコル名の前にアンダースコアを付けて、完全な DNS SRV レコード名を指定する必要があります。
- リクエストに複数のホスト名を指定しないでください。
- リクエストにポート番号を指定しないでください。
- TCP 接続のみがサポートされています。Unix ソケットファイル、Windows 名前付きパイプおよび共有メモリーは使用できません。

X DevAPI での DNS SRV ベースの接続の使用の詳細は、[Connections Using DNS SRV Records](#) を参照してください。

4.2.7 接続トランスポートプロトコル

MySQL クライアントライブラリを使用するプログラム (`mysql` や `mysqldump` など) の場合、MySQL は複数のトランスポートプロトコルに基づいたサーバーへの接続をサポート：TCP/IP、Unix ソケットファイル、名前付きパイプおよび

び共有メモリー。このセクションでは、これらのプロトコルを選択する方法と、それらがどのように類似して異なるかについて説明します。

- [トランスポートプロトコルの選択](#)
- [ローカル接続およびリモート接続のトランスポートサポート](#)
- [localhost の解釈](#)
- [暗号化とセキュリティの特性](#)
- [接続圧縮](#)

トランスポートプロトコルの選択

特定の接続では、トランスポートプロトコルが明示的に指定されていない場合、暗黙的に決定されます。たとえば、localhost に接続すると、Unix および Unix に似たシステムでソケットファイル接続が行われ、それ以外の場合は 127.0.0.1 への TCP/IP 接続が行われます。追加情報については [セクション4.2.4「コマンドオプションを使用した MySQL Server への接続」](#) を参照してください。

プロトコルを明示的に指定するには、`--protocol` コマンドオプションを使用します。次のテーブルに、`--protocol` に許される値を示し、各値に適用可能なプラットフォームを示します。値では大文字と小文字は区別されません。

<code>--protocol</code> の値	使用されるトランスポートプロトコル	適用可能なプラットフォーム
TCP	TCP/IP	すべて
SOCKET	Unix ソケットファイル	Unix および Unix に似たシステム
PIPE	名前付きパイプ	Windows
MEMORY	共有メモリー	Windows

ローカル接続およびリモート接続のトランスポートサポート

TCP/IP トランスポートでは、ローカルまたはリモートの MySQL サーバーへの接続がサポートされます。

ソケットファイル、名前付きパイプおよび共有メモリートランスポートでは、ローカル MySQL サーバーへの接続のみがサポートされます。(名前付きパイプトランスポートではリモート接続は可能ですが、この機能は MySQL に実装されていません。)

localhost の解釈

トランスポートプロトコルを明示的に指定しない場合、localhost は次のように解釈されます：

- Unix および Unix に似たシステムでは、localhost に接続するとソケットファイル接続が行われます。
- それ以外の場合、localhost に接続すると、127.0.0.1 への TCP/IP 接続になります。

トランスポートプロトコルが明示的に指定されている場合、localhost はそのプロトコルに関して解釈されます。たとえば、`--protocol=TCP` では、localhost に接続すると、すべてのプラットフォームの 127.0.0.1 に TCP/IP 接続されます。

暗号化とセキュリティの特性

TCP/IP およびソケットファイルトランスポートは、[暗号化接続のコマンドオプション](#) で説明されているオプションを使用して TLS/SSL 暗号化の対象となります。名前付きパイプおよび共有メモリートランスポートは TLS/SSL 暗号化の対象になりません。

デフォルトで保護されているトランスポートプロトコルを介して行われた場合、接続はデフォルトで保護されます。それ以外の場合、TLS/SSL 暗号化の対象となるプロトコルでは、暗号化を使用して接続をセキュアにすることができます。

- TCP/IP 接続はデフォルトではセキュアではありませんが、セキュアにするために暗号化できます。

- ソケットファイル接続はデフォルトでセキュアです。暗号化することもできますが、ソケットファイル接続を暗号化するとセキュアでなくなり、CPU 負荷が増加します。
- 名前付きパイプ接続はデフォルトではセキュアではなく、セキュアにするために暗号化の対象にはなりません。ただし、`named_pipe_full_access_group` システム変数を使用して、名前付きパイプ接続の使用を許可する MySQL ユーザーを制御できます。
- 共有メモリー接続はデフォルトでセキュアです。

`require_secure_transport` システム変数が有効になっている場合、サーバーは何らかの形式のセキュアなトランスポートを使用する接続のみを許可します。前述の備考に従って、TLS/SSL を使用して暗号化された TCP/IP を使用する接続、ソケットファイルまたは共有メモリーはセキュアな接続です。TLS/SSL および名前付きパイプ接続を使用して暗号化されていない TCP/IP 接続はセキュアではありません。

暗号化された接続の必須としての構成も参照してください。

接続圧縮

すべてのトランスポートプロトコルは、クライアントとサーバー間のトラフィックで圧縮を使用する必要があります。特定の接続に圧縮と暗号化の両方が使用されている場合、圧縮は暗号化の前に行われます。詳細は、[セクション 4.2.8 「接続圧縮制御」](#) を参照してください。

4.2.8 接続圧縮制御

サーバーへの接続では、クライアントとサーバー間のトラフィックの圧縮を使用して、接続を介して送信されるバイト数を減らすことができます。デフォルトでは、接続は圧縮解除されますが、サーバーとクライアントが相互に許可された圧縮アルゴリズムに同意する場合は圧縮できます。

圧縮された接続はクライアント側で発生しますが、クライアント側とサーバー側の両方で CPU 負荷に影響します。これは、両側で圧縮および解凍操作が実行されるためです。圧縮を有効にするとパフォーマンスが低下するため、主にネットワーク帯域幅が低く、ネットワーク転送時間が圧縮および解凍操作のコストを支配し、結果セットが大きい場合に利点が生じます。

このセクションでは、使用可能な圧縮制御構成パラメータと、圧縮の使用を監視するために使用できる情報ソースについて説明します。クラシック MySQL プロトコル 接続に適用されます。

圧縮制御は、クライアントプログラムおよびソース/レプリカレプリケーションまたはグループレプリケーションに参加しているサーバーによるサーバーへの接続に適用されます。圧縮制御は、[FEDERATED](#) テーブルの接続には適用されません。次の説明では、コンテキストに特定の接続タイプが指定されていないかぎり、「クライアント接続」は圧縮がサポートされている任意のソースからのサーバーへの接続の短縮形です。

注記

MySQL Server インスタンスへの X プロトコル 接続では、MySQL 8.0.19 からの圧縮がサポートされますが、X プロトコル 接続の圧縮は、ここで説明するクラシック MySQL プロトコル 接続の圧縮とは独立して動作し、個別に制御されます。X プロトコル 接続圧縮の詳細は、[セクション 20.5.5 「X プラグイン での接続圧縮」](#) を参照してください。

- [接続圧縮の構成](#)
- [レガシー接続圧縮の構成](#)
- [接続圧縮の監視](#)

接続圧縮の構成

MySQL 8.0.18 では、次の構成パラメータを使用して接続圧縮を制御できます:

- `protocol_compression_algorithms` システム変数は、サーバーが受信接続に対して許可する圧縮アルゴリズムを構成します。
- `--compression-algorithms` および `--zstd-compression-level` のコマンドラインオプションは、これらのクライアントプログラムに許可される圧縮アルゴリズムおよび `zstd` 圧縮レベルを構成: `mysql`, `mysqldadmin`,

`mysqlbinlog`, `mysqlcheck`, `mysqldump`, `mysqlimport`, `mysqlpump`, `mysqlshow`, `mysqslap`, `mysqltest`, および `mysql_upgrade`。MySQL Shell には、8.0.20 リリースの次のコマンドラインオプションも用意されています。

- `mysql_options()` 関数の `MYSQL_OPT_COMPRESSION_ALGORITHMS` および `MYSQL_OPT_ZSTD_COMPRESSION_LEVEL` オプションでは、MySQL C API を使用するクライアントプログラムの許可された圧縮アルゴリズムおよび `zstd` 圧縮レベルを構成します。
- `CHANGE MASTER TO` ステートメントの `MASTER_COMPRESSION_ALGORITHMS` および `MASTER_ZSTD_COMPRESSION_LEVEL` オプションでは、ソース/レプリカレプリケーションに参加するレプリカサーバーの許可された圧縮アルゴリズムおよび `zstd` 圧縮レベルを構成します。MySQL 8.0.23 から、かわりにステートメント `CHANGE REPLICATION SOURCE TO` およびオプション `SOURCE_COMPRESSION_ALGORITHMS` と `SOURCE_ZSTD_COMPRESSION_LEVEL` を使用します。
- `group_replication_recovery_compression_algorithm` および `group_replication_recovery_zstd_compression_level` システム変数は、新しいメンバーがグループに参加してドナーに接続したときに、グループレプリケーションリカバリ接続に対して許可される圧縮アルゴリズムおよび `zstd` 圧縮レベルを構成します。

圧縮アルゴリズムの指定を可能にする構成パラメータは文字列値で、次の項目から任意の順序で選択された 1 つ以上のカンマ区切りの圧縮アルゴリズム名のリストを取ります (大/小文字は区別されません):

- `zlib`: `zlib` 圧縮アルゴリズムを使用する接続を許可します。
- `zstd`: `zstd` 圧縮アルゴリズム (`zstd 1.3`) を使用する接続を許可します。
- `uncompressed`: 圧縮解除された接続を許可します。

注記

`uncompressed` は、構成されている場合とされていない場合があるアルゴリズム名であるため、圧縮されていない接続を許可しないように MySQL を構成できます。

例:

- サーバーが受信接続に対して許可する圧縮アルゴリズムを構成するには、`protocol_compression_algorithms` システム変数を設定します。デフォルトでは、サーバーは使用可能なすべてのアルゴリズムを許可します。起動時にこの設定を明示的に構成するには、サーバー `my.cnf` ファイルで次の行を使用します:

```
[mysqld]
protocol_compression_algorithms=zlib,zstd,uncompressed
```

実行時に `protocol_compression_algorithms` システム変数を設定してその値に永続化するには、次のステートメントを使用します:

```
SET PERSIST protocol_compression_algorithms='zlib,zstd,uncompressed';
```

`SET PERSIST` は、実行中の MySQL インスタンスの値を設定します。また、値が保存され、その後のサーバーの再起動に引き継がれます。後続の再起動に引き継ぐことなく、実行中の MySQL インスタンスの値を変更するには、`PERSIST` ではなく `GLOBAL` キーワードを使用します。セクション 13.7.6.1 「変数代入の SET 構文」を参照してください。

- `zstd` 圧縮を使用する着信接続のみを許可するには、起動時に次のようにサーバーを構成します:

```
[mysqld]
protocol_compression_algorithms=zstd
```

または、実行時に変更を行うには:

```
SET PERSIST protocol_compression_algorithms='zstd';
```

- `mysql` クライアントが `zlib` または `uncompressed` 接続を開始できるようにするには、次のように呼び出します:

```
mysql --compression-algorithms=zlib,uncompressed
```

- `zlib` または `zstd` 接続を使用して、`zstd` 接続用に圧縮レベル 7 でソースに接続するようにレプリカを構成するには、`CHANGE REPLICATION SOURCE TO` ステートメント (MySQL 8.0.23 から) または `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 より前) を使用します:

```
CHANGE REPLICATION SOURCE TO
SOURCE_COMPRESSION_ALGORITHMS = 'zlib,zstd',
SOURCE_ZSTD_COMPRESSION_LEVEL = 7;
```

これは、[レガシー接続圧縮の構成](#) で説明されている理由で、`slave_compressed_protocol` システム変数が無効になっていることを前提としています。

接続を正常に設定するには、接続の両側が相互に許可された圧縮アルゴリズムに同意する必要があります。アルゴリズム - ネゴシエーションプロセスでは、`zlib`、`zstd`、`uncompressed` の順に使用しようとしています。両側で共通アルゴリズムが見つからない場合、接続の試行は失敗します。

どちらの側も圧縮アルゴリズムに同意する必要があるため、`uncompressed` は必ずしも許可されていないアルゴリズム値であるため、圧縮されていない接続へのフォールバックは必ずしも発生しません。たとえば、サーバーが `zstd` を許可するように構成され、クライアントが `zlib,uncompressed` を許可するように構成されている場合、クライアントはまったく接続できません。この場合、両側に共通のアルゴリズムがないため、接続の試行は失敗します。

`zstd` 圧縮レベルの指定を可能にする構成パラメータは、1 から 22 までの整数値を取り、大きい値は圧縮レベルの増加を示します。デフォルトの `zstd` 圧縮レベルは 3 です。圧縮レベルの設定は、`zstd` 圧縮を使用しない接続には影響しません。

構成可能な `zstd` 圧縮レベルを使用すると、ネットワークトラフィックの減少と CPU 負荷の増加の比較、およびネットワークトラフィックの増加と CPU 負荷の低下を選択できます。圧縮レベルを高くすると、ネットワークの輻輳が軽減されますが、CPU 負荷が増加すると、サーバーのパフォーマンスが低下する可能性があります。

レガシー接続圧縮の構成

MySQL 8.0.18 より前は、次の構成パラメータを使用して接続圧縮を制御できました:

- クライアントプログラムでは、サーバーへの接続に圧縮の使用を指定する `--compress` コマンドラインオプションがサポートされています。
- MySQL C API を使用するプログラムの場合、`mysql_options()` 関数の `MYSQL_OPT_COMPRESS` オプションを有効にすると、サーバーへの接続に圧縮を使用するように指定されます。
- ソース/レプリカレプリケーションの場合、`slave_compressed_protocol` システム変数を有効にすると、ソースへのレプリカ接続に圧縮の使用が指定されます。

いずれの場合も、圧縮の使用が指定されている場合、接続は `zlib` 圧縮アルゴリズムを使用します (両側で許可されている場合)。それ以外の場合は、圧縮されていない接続にフォールバックします。

MySQL 8.0.18 では、[接続圧縮の構成](#) で説明されている接続圧縮をより詳細に制御するために導入された追加の圧縮パラメータのため、前述の圧縮パラメータはレガシーパラメータになります。例外は MySQL Shell で、`--compress` コマンドラインオプションは現在のままで、圧縮アルゴリズムを選択せずに圧縮をリクエストするために使用できます。MySQL Shell の接続圧縮制御の詳細は、[圧縮接続の使用](#) を参照してください。

レガシー圧縮パラメータは、次のように新しいパラメータと対話し、そのセマンティクスが変更されます:

- レガシー `--compress` オプションの意味は、`--compression-algorithms` が指定されているかどうかによって異なります:
- `--compression-algorithms` が指定されていない場合、`--compress` は `zlib,uncompressed` のクライアント側アルゴリズムセットを指定することと同等です。
- `--compression-algorithms` が指定されている場合、`--compress` は `zlib` のアルゴリズムセットを指定することと同等であり、完全なクライアント側アルゴリズムセットは `zlib` の和集合に `--compression-algorithms` で指定されたアルゴリズムを加えたものです。たとえば、`--compress` と `--compression-algorithms=zlib,zstd` の両方で許可されるアルゴリズムセットは、`zlib` と `zlib,zstd(zlib,zstd)` の組合せです。`--compress` と `--compression-algorithms=zstd,uncompressed` の両方で許可されるアルゴリズムセットは、`zlib` と `zstd,uncompressed(zlib,zstd,uncompressed)` の組合せです。
- レガシー `MYSQL_OPT_COMPRESS` オプションと `mysql_options()` C API 関数の `MYSQL_OPT_COMPRESSION_ALGORITHMS` オプションの間では、同じタイプの相互作用が発生します。

- `slave_compressed_protocol` システム変数が有効になっている場合、`MASTER_COMPRESSION_ALGORITHMS` よりも優先され、ソースとレプリカの両方でそのアルゴリズムが許可されている場合は、ソースへの接続で `zlib` 圧縮が使用されます。 `slave_compressed_protocol` が無効な場合は、`MASTER_COMPRESSION_ALGORITHMS` の値が適用されます。

注記

レガシー圧縮制御パラメータは、MySQL 8.0.18 では非推奨です。MySQL の将来のバージョンでは削除される予定です。

接続圧縮の監視

`Compression` ステータス変数は `ON` または `OFF` で、現在の接続で圧縮が使用されるかどうかを示します。

`mysql` クライアントの `\status` コマンドは、現在の接続に対して圧縮が有効になっている場合に `Protocol: Compressed` を示す行を表示します。その行が存在しない場合、接続は圧縮解除されます。

8.0.14 では、MySQL Shell `\status` コマンドにより、`Disabled` または `Enabled` という `Compression:` 行が表示され、接続が圧縮されているかどうかを示されます。

MySQL 8.0.18 では、次の追加情報ソースを使用して接続圧縮を監視できます：

- クライアント接続に使用されている圧縮を監視するには、`Compression_algorithm` および `Compression_level` のステータス変数を使用します。現在の接続の場合、その値はそれぞれ圧縮アルゴリズムと圧縮レベルを示します。
- サーバーが着信接続を許可するように構成されている圧縮アルゴリズムを判別するには、`protocol_compression_algorithms` システム変数を確認します。
- ソース/レプリカレプリケーション接続の場合、構成済の圧縮アルゴリズムおよび圧縮レベルは複数のソースから使用できます：
 - パフォーマンススキーマ `replication_connection_configuration` テーブルには、`COMPRESSION_ALGORITHMS` および `ZSTD_COMPRESSION_LEVEL` カラムがあります。
 - `mysql.slave_master_info` システムテーブルには、`Master_compression_algorithms` カラムと `Master_zstd_compression_level` カラムがあります。 `master.info` ファイルが存在する場合は、それらの値の行も含まれます。

4.2.9 環境変数の設定

環境変数は、コマンドプロセッサの現在の起動に影響するようにコマンドプロンプトで設定したり、将来の起動に影響するように永続的に設定したりできます。変数を永続的に設定するには、起動ファイルで設定するか、またはこのためにシステムが提供するインタフェースを使用して設定できます。具体的な詳細は、コマンドインタプリタのドキュメントを参照してください。セクション4.9「環境変数」には、MySQL プログラムの動作に影響するすべての環境変数のリストがあります。

環境変数の値を指定するには、コマンドプロセッサに適した構文を使用します。たとえば Windows では、`USER` 変数を設定して MySQL アカウント名を指定できます。そうするには、次の構文を使用します。

```
SET USER=your_name
```

Unix での構文はシェルに依存します。 `MYSQL_TCP_PORT` 変数を使用して TCP/IP ポート番号を指定するとします。(`sh`、`ksh`、`bash`、`zsh`、などの) 典型的な構文は次のとおりです。

```
MYSQL_TCP_PORT=3306
export MYSQL_TCP_PORT
```

最初のコマンドが変数を設定し、その値が MySQL およびほかのプロセスにアクセス可能になるように、`export` コマンドがシェル環境に変数をエクスポートします。

`csh` および `tcsh` については、シェル変数を環境で使用可能にするために `setenv` を使用してください。

```
setenv MYSQL_TCP_PORT 3306
```


環境変数を設定するコマンドは、コマンドプロンプトで実行してただちに有効にできますが、その設定はログアウトするまでしか持続しません。ログインするたびに設定を有効にするには、システムが提供するインタフェースを使用するか、コマンドインタプリタが、起動時に毎回読み取る起動ファイルに適切なコマンドを配置します。

Windows では、システム コントロール パネル (詳細設定の下) で環境変数を設定できます。

Unix では、通常のシェル起動ファイルは `bash` では `.bashrc` または `.bash_profile`、または `tclsh` では `.tclshrc` です。

MySQL プログラムが `/usr/local/mysql/bin` にインストールされ、これらのプログラムを呼び出しやすくしたいとします。そのためには、`PATH` 環境変数の値をそのディレクトリを含むように設定します。たとえば、シェルが `bash` の場合、`.bashrc` ファイルに次の行を追加します。

```
PATH=${PATH}:/usr/local/mysql/bin
```

`bash` はログインシェルと非ログインシェルで異なる起動ファイルを使用するため、ログインシェルに対しては設定を `.bashrc`、非ログインシェルに対しては `.bash_profile` に追加して、いずれの場合にも確実に `PATH` が設定されるようにするとよいでしょう。

シェルが `tclsh` の場合、`.tclshrc` ファイルに次の行を追加します。

```
setenv PATH ${PATH}:/usr/local/mysql/bin
```

ホームディレクトリに適切な起動ファイルが存在しない場合、テキストエディタで作成します。

`PATH` 設定を変更したら、設定が反映されるように、Windows の場合は新しいコンソールウィンドウを開き、Unix の場合はログインしなおします。

4.3 サーバーおよびサーバーの起動プログラム

このセクションでは、MySQL サーバー `mysqld`、およびサーバーを起動するために使用されるいくつかのプログラムについて説明します。

4.3.1 `mysqld` — MySQL サーバー

`mysqld` (MySQL Server と呼ばれる) は、MySQL インストールでほとんどの作業を実行する単一のマルチスレッドプログラムです。追加プロセスは生成されません。MySQL サーバーは、データベースおよびテーブルを含む MySQL データディレクトリへのアクセスを管理します。データディレクトリは、ログファイルおよびステータスファイルなど、その他の情報のデフォルトの場所でもあります。

注記

一部のインストールパッケージには、`mysqld-debug` という名前のサーバーのデバッグバージョンが含まれています。デバッグサポート、メモリー割当てチェックおよびトレースファイルサポートのために、`mysqld` のかわりにこのバージョンを起動します ([セクション 5.9.1.2 「トレースファイルの作成」](#) を参照)。

MySQL サーバーが起動するとき、クライアントプログラムからのネットワーク接続を待機し、それらのクライアントに代わってデータベースへのアクセスを管理します。

`mysqld` プログラムには、起動時に指定できる多くのオプションがあります。すべてのオプションをリストするには、次のコマンドを実行します。

```
shell> mysqld --verbose --help
```

MySQL サーバーには、稼働時の動作に影響する一連のシステム変数もあります。システム変数はサーバーの起動時に設定でき、多くは実行時に変更して動的なサーバー再構成に影響を与えることができます。MySQL サーバーには、動作に関する情報を提供する一連のステータス変数もあります。これらのステータス変数をモニターして、実行時のパフォーマンス特性にアクセスできます。

MySQL サーバーのコマンドオプション、システム変数、およびステータス変数の詳細な説明は、[セクション 5.1 「MySQL Server」](#) を参照してください。MySQL のインストールおよび初期構成のセットアップについては、[第2章 「MySQL のインストールとアップグレード」](#) を参照してください。

4.3.2 `mysqld_safe` — MySQL サーバー起動スクリプト

Unix で `mysqld` サーバーを起動するには、`mysqld_safe` をお勧めします。`mysqld_safe` では、エラー発生時のサーバーの再起動やエラーログへの実行時情報のロギングなど、いくつかの安全機能が追加されています。エラーのロギングについては、このセクションで後ほど説明します。

注記

一部の Linux プラットフォームでは、RPM または Debian パッケージからの MySQL インストールに、MySQL サーバーの起動と停止を管理するための `systemd` サポートが含まれています。これらのプラットフォームでは、`mysqld_safe` は不要であるためインストールされません。詳細は、[セクション2.5.9「systemd を使用した MySQL Server の管理」](#)を参照してください。

`mysqld_safe` は、`mysqld` という名前の実行可能ファイルを起動しようとします。デフォルトの動作をオーバーライドして、起動するサーバーの名前を明示的に指定するには、`mysqld_safe` で `--mysqld` オプションまたは `--mysqld-version` オプションを指定します。`--ledir` オプションを使用して、`mysqld_safe` がサーバーを検索するディレクトリを指定することもできます。

`mysqld_safe` のオプションの多くは、`mysqld` のオプションと同じです。[セクション5.1.7「サーバーコマンドオプション」](#)を参照してください。

`mysqld_safe` が理解できないオプションは、コマンド行で指定された場合は `mysqld` に渡されますが、オプションファイルの `[mysqld_safe]` グループに指定された場合は無視されます。[セクション4.2.2.2「オプションファイルの使用」](#)を参照してください。

`mysqld_safe` は、オプションファイルの `[mysqld]`、`[server]`、`[mysqld_safe]` の各セクションからすべてのオプションを読み取ります。たとえば、次のような `[mysqld]` セクションを指定すると、`mysqld_safe` は `--log-error` オプションを検索して使用します:

```
[mysqld]
log-error=error.log
```

下位互換性のために、`mysqld_safe` は `[safe_mysqld]` セクションも読み取りますが、このようなセクションの名前を `[mysqld_safe]` に変更する必要があります。

`mysqld_safe` では、次のテーブルに示すように、コマンドラインおよびオプションファイルでオプションを使用できます。MySQL プログラムによって使用されるオプションファイルの詳細については、[セクション4.2.2.2「オプションファイルの使用」](#)を参照してください。

表 4.6 「mysqld_safe オプション」

オプション名	説明
<code>--basedir</code>	MySQL インストールディレクトリへのパス
<code>--core-file-size</code>	<code>mysqld</code> が作成できるべきコアファイルのサイズ
<code>--datadir</code>	データディレクトリへのパス
<code>--defaults-extra-file</code>	通常のオプションファイルに加えて、名前付きオプションファイルを読み取ります
<code>--defaults-file</code>	指名されたオプションファイルのみを読み取る
<code>--help</code>	ヘルプメッセージを表示して終了
<code>--ledir</code>	サーバーが置かれているディレクトリへのパス
<code>--log-error</code>	指定されたファイルにエラーログを書き込み
<code>--malloc-lib</code>	<code>mysqld</code> で使用する代替 <code>malloc</code> ライブラリ
<code>--mysqld</code>	起動するサーバープログラム名 (<code>ledir</code> ディレクトリの)
<code>--mysqld-safe-log-timestamps</code>	ロギングのタイムスタンプ書式
<code>--mysqld-version</code>	サーバープログラム名のサフィクス
<code>--nice</code>	サーバーのスケジュール設定の優先順位を設定するために <code>nice</code> プログラムを使用

オプション名	説明
<code>--no-defaults</code>	オプションファイルを読み取らない
<code>--open-files-limit</code>	mysqld が開くことができるファイル数
<code>--pid-file</code>	サーバープロセス ID ファイルのパス名
<code>--plugin-dir</code>	プラグインがインストールされているディレクトリ
<code>--port</code>	TCP/IP 接続を待機するポート番号
<code>--skip-kill-mysqld</code>	迷子の mysqld プロセスの kill を試行しない
<code>--skip-syslog</code>	syslog にエラーメッセージを書き込まず、エラーログファイルを使用
<code>--socket</code>	Unix ソケット接続を待機するソケットファイル
<code>--syslog</code>	syslog にエラーメッセージを書き込み
<code>--syslog-tag</code>	syslog に書き込まれるメッセージのタグサフィクス
<code>--timezone</code>	TZ タイムゾーン環境変数を指定された値に設定
<code>--user</code>	mysqld を、名前が <code>user_name</code> または数値のユーザー ID が <code>user_id</code> であるユーザーとして実行

- `--help`

ヘルプメッセージを表示して終了します。

- `--basedir=dir_name`

MySQL インストールディレクトリへのパス。

- `--core-file-size=size`

mysqld で作成できるべきコアファイルのサイズ。このオプション値は `ulimit -c` に渡されます。

注記

`innodb_buffer_pool_in_core_file` 変数を使用すると、それをサポートするオペレーティングシステム上のコアファイルのサイズを縮小できます。詳細は、[セクション15.8.3.7「コアファイルからのバッファープールページの除外」](#)を参照してください。

- `--datadir=dir_name`

データディレクトリへのパス。

- `--defaults-extra-file=file_name`

通常のオプションファイルに加えて、このオプションファイルを読み取ります。ファイルが存在しないか、アクセスできない場合、サーバーはエラーで終了します。`file_name` は、フルパス名ではなく相対パス名として指定された場合、カレントディレクトリを基準にして解釈されます。これを使用する場合は、コマンド行の最初のオプションでなければなりません。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--defaults-file=file_name`

指定されたオプションファイルのみ使用します。ファイルが存在しないか、アクセスできない場合、サーバーはエラーで終了します。`file_name` は、フルパス名ではなく相対パス名として指定された場合、カレントディレクトリを基準にして解釈されます。これを使用する場合は、コマンド行の最初のオプションでなければなりません。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--ledir=dir_name`

`mysqld_safe` がサーバーを検出できない場合、このオプションを使用してサーバーが置かれているディレクトリのパス名を示します。

このオプションはコマンド行でのみ受け入れられ、オプションファイルでは受け入れられません。systemd を使用するプラットフォームでは、`MYSQLD_OPTS` の値で値を指定できます。 [セクション2.5.9「systemd を使用した MySQL Server の管理」](#) を参照してください。

- `--log-error=file_name`

指定されたファイルにエラーログを書き込みます。 [セクション5.4.2「エラーログ」](#) を参照してください。

- `--mysqld-safe-log-timestamps`

このオプションは、`mysqld_safe` によって生成されるログ出力のタイムスタンプの書式を制御します。次のリストに、許可される値を示します。その他の値の場合、`mysqld_safe` は警告をログに記録し、UTC 形式を使用します。

- `UTC, utc`

ISO 8601 UTC 形式 (サーバーの `--log_timestamps=UTC` と同じ)。これはデフォルトです。

- `SYSTEM, system`

ISO 8601 のローカル時間書式 (サーバーの `--log_timestamps=SYSTEM` と同じ)。

- `HYPHEN, hyphen`

`mysqld_safe` for MySQL 5.6 と同様の `YY-MM-DD h:mm:ss` 形式。

- `LEGACY, legacy`

MySQL 5.6 より前の `mysqld_safe` と同様の `YYMMDD hh:mm:ss` 形式。

- `--malloc-lib=[lib_name]`

システムの `malloc()` ライブラリの代わりに、メモリー割り当てに使用されるライブラリの名前。オプション値は、ディレクトリ `/usr/lib`, `/usr/lib64`, `/usr/lib/i386-linux-gnu` または `/usr/lib/x86_64-linux-gnu` のいずれかである必要があります。

`--malloc-lib` オプションは、`mysqld` が起動するときに、ダイナミックリンクに影響を与え、ローダーがメモリー割り当てライブラリを検出できるように、`LD_PRELOAD` 環境値を変更します。

- このオプションが指定されない場合、または値なしで指定された場合 (`--malloc-lib=`)、`LD_PRELOAD` は変更されず、`tcmalloc` を使用する試みは行われません。

- MySQL 8.0.21 より前では、このオプションが `--malloc-lib=tcmalloc` として指定されている場合、`mysqld_safe` は `/usr/lib` で `tcmalloc` ライブラリを検索します。 `tmalloc` が検出された場合は、そのパス名が `LD_PRELOAD` 値の先頭に追加されて `mysqld` に渡されます。 `tcmalloc` が見つからない場合は、`mysqld_safe` はエラーで中止されます。

MySQL 8.0.21 の時点では、`tcmalloc` は `--malloc-lib` オプションに許可された値ではありません。

- オプションが `--malloc-lib=/path/to/some/library` と指定された場合、そのフルパスが `LD_PRELOAD` 値の先頭に追加されます。そのフルパスが、存在しないかまたは読み取り不能なファイルを指している場合は、`mysqld_safe` はエラーで中止されます。

- `mysqld_safe` がパス名を `LD_PRELOAD` に追加する場合は、その変数がすでに持っている、既存の値の先頭にパスを追加します。

注記

systemd を使用してサーバーを管理するシステムでは、`mysqld_safe` は使用できません。かわりに、`/etc/sysconfig/mysql` で `LD_PRELOAD` を設定して割当てライブラリを指定します。

Linux ユーザーは、次の行を `my.cnf` ファイルに追加することで、`/usr/lib` に `tcmalloc` パッケージがインストールされている任意のプラットフォームで `libtcmalloc_minimal.so` ライブラリを使用できます:

```
[mysqld_safe]
malloc-lib=tcmalloc
```

特定の `tcmalloc` ライブラリを使用するには、そのフルパス名を指定します。例:

```
[mysqld_safe]
malloc-lib=/opt/lib/libtcmalloc_minimal.so
```

- `--mysqld=prog_name`

`ledir` ディレクトリにある、起動するサーバープログラムの名前。MySQL のバイナリ配布を使用するが、バイナリ配布以外のデータディレクトリがある場合には、このオプションが必要です。 `mysqld_safe` がサーバーを検出できない場合は、`--ledir` オプションを使用してサーバーのあるディレクトリへのパス名を示します。

このオプションはコマンド行でのみ受け入れられ、オプションファイルでは受け入れられません。 `systemd` を使用するプラットフォームでは、`MYSQLD_OPTS` の値で値を指定できます。 [セクション2.5.9「systemdを使用したMySQL Serverの管理」](#) を参照してください。

- `--mysqld-version=suffix`

このオプションは `--mysqld` オプションと類似していますが、サーバーのプログラム名のサフィクスだけを指定します。ベース名は `mysqld` であると想定されます。たとえば、`--mysqld-version=debug` を使用すると、`mysqld_safe` は `ledir` ディレクトリの `mysqld-debug` プログラムを起動します。 `--mysqld-version` の引数が空の場合、`mysqld_safe` は `ledir` ディレクトリの `mysqld` を使用します。

このオプションはコマンド行でのみ受け入れられ、オプションファイルでは受け入れられません。 `systemd` を使用するプラットフォームでは、`MYSQLD_OPTS` の値で値を指定できます。 [セクション2.5.9「systemdを使用したMySQL Serverの管理」](#) を参照してください。

- `--nice=priority`

サーバーのスケジューリング優先順位を任意の値に設定するには、`nice` プログラムを使用します。

- `--no-defaults`

オプションファイルを読み取りません。オプションファイルから不明のオプションを読み取ることが原因でプログラムの起動に失敗する場合、`--no-defaults` を使用して、オプションを読み取らないようにすることができます。これを使用する場合は、コマンド行の最初のオプションでなければなりません。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--open-files-limit=count`

`mysqld` が開くことができるファイル数。オプション値は `ulimit -n` に渡されます。

注記

これが正しく機能するためには、`mysqld_safe` を `root` として起動する必要があります。

- `--pid-file=file_name`

`mysqld` がプロセス ID ファイルに使用するパス名。

- `--plugin-dir=dir_name`

プラグインディレクトリのパス名。

- `--port=port_num`

サーバーが TCP/IP 接続を待機するときに使用するポート番号。 `root` オペレーティングシステムユーザーがサーバーを起動しないかぎり、ポート番号は 1024 以上である必要があります。

- `--skip-kill-mysqld`

起動時に、未処理の `mysqld` プロセスの強制終了を試行しません。このオプションは、Linux のみで機能します。

- `--socket=path`

サーバーがローカル接続を待機するときに使用する Unix ソケットファイル。

- `--syslog, --skip-syslog`

`--syslog` は、`logger` プログラムをサポートするシステムで、エラーメッセージが `syslog` に送信されるようにします。`--skip-syslog` は、`syslog` の使用を抑制し、メッセージはエラーログファイルに書き込まれます。

`syslog` をエラーロギングに使用すると、すべてのログメッセージに `daemon.err` の機能/重大度が使用されます。

これらのオプションを使用した `mysqld` ロギングの制御は非推奨になりました。エラーログ出力をシステムログに書き込むには、[セクション5.4.2.8「システムログへのエラーロギング」](#) の手順を使用します。この機能を制御するには、`server log_syslog_facility` システム変数を使用します。

- `--syslog-tag=tag`

`syslog` へのロギングで、`mysqld_safe` および `mysqld` からのメッセージはそれぞれ `mysqld_safe` および `mysqld` 識別子を付けて書き込まれます。識別子のサフィクスを指定するには、`--syslog-tag=tag` を使用します。これにより、識別子は `mysqld_safe-tag` および `mysqld-tag` に変更されます。

このオプションを使用した `mysqld` ロギングの制御は非推奨になりました。代わりにサーバーの `log_syslog_tag` システム変数を使用してください。[セクション5.4.2.8「システムログへのエラーロギング」](#) を参照してください。

- `--timezone=timezone`

TZ タイムゾーン環境変数を、指定されたオプション値に設定します。正当なタイムゾーン指定形式は、オペレーティングシステムのドキュメントを参照してください。

- `--user={user_name|user_id}`

`mysqld` サーバーを、名前 `user_name` または数字ユーザー ID `user_id` を持つユーザーとして実行します。(このコンテキストでの「ユーザー」は、システムログインアカウントであり、付与テーブルにリストされている MySQL ユーザーではありません。)

`--defaults-file` または `--defaults-extra-file` オプションを指定して `mysqld_safe` を実行してオプションファイルに名前を付ける場合、オプションはコマンドラインで指定された最初のオプションである必要があります。そうでない場合、オプションファイルは使用されません。たとえば、次のコマンドは指定されたオプションファイルを使用しません:

```
mysql> mysqld_safe --port=port_num --defaults-file=file_name
```

代わりに、次のコマンドを使用します。

```
mysql> mysqld_safe --defaults-file=file_name --port=port_num
```

MySQL のソースまたはバイナリ配布は通常、サーバーを若干異なる場所にインストールしますが、`mysqld_safe` スクリプトは、どちらからインストールしたサーバーでも正常に立ち上げることができるように作成されています。([セクション2.1.5「インストールのレイアウト」](#) を参照してください。) `mysqld_safe` は、次の条件のいずれかが満たされていることを想定しています。

- サーバーとデータベースは作業ディレクトリ (`mysqld_safe` が呼び出されたディレクトリ) から相対的に検索できること。バイナリ配布の場合、`mysqld_safe` は作業ディレクトリの下 `bin` ディレクトリおよび `data` ディレクトリを検索します。ソース配布の場合、`libexec` ディレクトリおよび `var` ディレクトリを検索します。`mysqld_safe` を MySQL インストールディレクトリ (バイナリ配布の場合は `/usr/local/mysql`) から起動した場合には、この条件が満たされるはずですが。
- サーバーとデータベースが作業ディレクトリから相対的に検出できない場合は、`mysqld_safe` は絶対パス名での検索を試みます。通常の場合は `/usr/local/libexec` および `/usr/local/var` です。実際の場所は、配布のビルド時に構成される値から決定されます。構成時に指定された場所に MySQL がインストールされていれば、これは正しいはずですが。

`mysqld_safe` はサーバーおよびデータベースを作業ディレクトリから相対的に検索しようとするため、`mysqld_safe` を MySQL インストールディレクトリから起動するがぎり、MySQL のバイナリ配布は任意の場所にインストールできません。

```
shell> cd mysql_installation_directory
shell> bin/mysqld_safe &
```

`mysqld_safe` を MySQL インストールディレクトリから呼び出しても失敗する場合は、`--ledir` オプションおよび `--datadir` オプションを指定して、システムのサーバーとデータベースがあるディレクトリを指定します。

`mysqld_safe` は、`sleep` および `date` システムユーティリティを使用して、起動を試行した回数を秒単位で判別しようとします。これらのユーティリティが存在し、試行された起動数/秒が 5 より大きい場合、`mysqld_safe` は 1 フル秒待機してから再起動します。これは、連続して失敗する場合に過度に CPU を使用することを防ぐためのものです。(Bug #11761530、Bug #54035)

`mysqld_safe` を使用して `mysqld` を起動する場合、`mysqld_safe` は自身と `mysqld` からのエラー (および通知) メッセージが同じ出力先に送信されるよう手配します。

これらのメッセージの出力先を制御するための `mysqld_safe` オプションがいくつかあります。

- `--log-error=file_name`: エラーメッセージを指定されたエラーファイルに書き込みます。
- `--syslog: logger` プログラムをサポートするシステムで、`syslog` にエラーメッセージを書き込みます。
- `--skip-syslog`: エラーメッセージを `syslog` に書き込みません。メッセージは、デフォルトのエラーログファイル (データディレクトリの `host_name.err`)、または `--log-error` オプションが指定された場合は、指定されたファイルに書き込まれます。

これらのオプションが指定されていない場合は、デフォルトは `--skip-syslog` です。

`mysqld_safe` がメッセージを書き込む場合、通知はロギングの出力先 (`syslog` またはエラーログファイル) および `stdout` に送られます。エラーはロギングの出力先と `stderr` に送られます。

注記

`mysqld_safe` からの `mysqld` ロギングの制御は非推奨になりました。かわりに、サーバー固有の `syslog` サポートを使用してください。詳細は、[セクション5.4.2.8「システムログへのエラーロギング」](#)を参照してください。

4.3.3 mysql.server — MySQL サーバー起動スクリプト

Unix および Unix に似たシステム上の MySQL ディストリビューションには、`mysqld_safe` を使用して MySQL サーバーを起動する `mysql.server` というスクリプトが含まれています。System V スタイルの実行ディレクトリを使用してシステムサービスの起動および停止を行う、Linux および Solaris などのシステムで使用できます。また、macOS Startup Item for MySQL でも使用されます。

`mysql.server` は、MySQL ソースツリー内で使用されるスクリプト名です。インストールされた名前が異なる場合があります (たとえば、`mysqld` や `mysqld`)。次の説明では、システムに応じて `mysql.server` という名前を調整します。

注記

一部の Linux プラットフォームでは、RPM または Debian パッケージからの MySQL インストールに、MySQL サーバーの起動と停止を管理するための `systemd` サポートが含まれています。これらのプラットフォームでは、`mysql.server` および `mysqld_safe` は不要なためインストールされません。詳細は、[セクション2.5.9「systemd を使用した MySQL Server の管理」](#)を参照してください。

`mysql.server` スクリプトを使用してサーバーを手動で起動または停止するには、`start` または `stop` 引数を指定してコマンドラインから起動します:

```
shell> mysql.server start
shell> mysql.server stop
```


`mysql.server` は、場所を MySQL インストールディレクトリに変更し、`mysqld_safe` を起動します。特定のユーザーとしてサーバーを実行するには、このセクションで後述するように、グローバル`/etc/my.cnf` オプションファイルの`[mysqld]`グループに適切な `user` オプションを追加します。(非標準の場所に MySQL のバイナリ配布をインストールした場合は、`mysql.server` を編集する必要があります。`mysqld_safe` を実行する前に場所を適切なディレクトリに変更するように、それを修正します。これを行うと、将来 MySQL をアップグレードした場合に、変更されたバージョンの `mysql.server` が上書きされる可能性があります。再インストール可能な編集済バージョンのコピーを作成してください。)

`mysql.server stop` は、サーバーに信号を送って停止します。`mysqladmin shutdown` を実行してサーバーを手動で停止することもできます。

サーバー上で MySQL を自動的に起動および停止するには、`/etc/rc*` ファイル内の適切な場所に起動および停止コマンドを追加する必要があります:

- Linux サーバー RPM パッケージ (`MySQL-server-VERSION.rpm`) またはネイティブ Linux パッケージインストールを使用する場合、`mysql.server` スクリプトは `mysqld` または `mysql` という名前で`/etc/init.d` ディレクトリにインストールできます。Linux RPM パッケージに関する詳細は、[セクション2.5.4「Oracle の RPM パッケージを使用した Linux への MySQL のインストール」](#)を参照してください。
- MySQL をソース配布からインストールする場合、または `mysql.server` を自動的にインストールしないバイナリ配布形式を使用する場合は、スクリプトを手動でインストールできます。これは、MySQL インストールディレクトリの下に `support-files` ディレクトリまたは MySQL ソースツリーにあります。`mysql` という名前の`/etc/init.d` ディレクトリにスクリプトをコピーし、実行可能にします:

```
shell> cp mysql.server /etc/init.d/mysql
shell> chmod +x /etc/init.d/mysql
```

スクリプトをインストールしたあと、それを有効にしてシステムの起動時に実行するために必要なコマンドは、使用しているオペレーティングシステムによって異なります。Linux では、`chkconfig` を使用します。

```
shell> chkconfig --add mysql
```

一部の Linux システムでは、`mysql` スクリプトを完全に有効にするには次のコマンドも必要になる場合があります。

```
shell> chkconfig --level 345 mysql on
```

- FreeBSD では、起動スクリプトは通常 `/usr/local/etc/rc.d/` にあります。`mysql.server` スクリプトを`/usr/local/etc/rc.d/mysql.server.sh` としてインストールし、自動起動を有効にします。`rc(8)` のマニュアルページには、このディレクトリ内のスクリプトは、そのベース名が `*.sh` Shell ファイル名パターンと一致する場合にのみ実行されると記載されています。そのディレクトリの、その他のファイルあるいはディレクトリは警告なしで無視されます。
- 前述の設定の代案として、オペレーティングシステムの中には `/etc/rc.local` あるいは `/etc/init.d/boot.local` を使用して起動時に追加のサービスを起動するものもあります。この方法を使用して MySQL を起動するには、次のようなコマンドを適切な起動ファイルに追加します:

```
/bin/sh -c 'cd /usr/local/mysql; ./bin/mysqld_safe --user=mysql &'
```

- ほかのシステムの起動スクリプトのインストール方法についてはそのオペレーティングシステムのドキュメントをお読みください。

`mysql.server` は、オプションファイルの `[mysql.server]` セクションおよび `[mysqld]` セクションからオプションを読み取ります。下位互換性のために、`[mysql_server]`セクションも読み取りますが、最新の状態にするには、このようなセクションの名前を`[mysql.server]`に変更する必要があります。

`mysql.server` のオプションを、グローバル `/etc/my.cnf` ファイルに追加できます。一般的な `my.cnf` ファイルは次のようになります:

```
[mysqld]
datadir=/usr/local/mysql/var
socket=/var/tmp/mysql.sock
port=3306
user=mysql

[mysql.server]
basedir=/usr/local/mysql
```

`mysql.server` スクリプトでは、次のテーブルに示すオプションがサポートされます。指定する場合は、コマンド行ではなくオプションファイルに配置する必要があります。`mysql.server` は、`start` および `stop` のみをコマンド行の引数としてサポートしています。

表 4.7 「mysql.server オプション - ファイルオプション」

オプション名	説明	型
<code>basedir</code>	MySQL インストールディレクトリへのパス	ディレクトリ名
<code>datadir</code>	MySQL データディレクトリへのパス	ディレクトリ名
<code>pid-file</code>	サーバーがプロセス ID を書き込むファイル	ファイル名
<code>service-startup-timeout</code>	サーバーの起動を待機する時間	Integer

- `basedir=dir_name`

MySQL インストールディレクトリへのパス。

- `datadir=dir_name`

MySQL データディレクトリへのパス。

- `pid-file=file_name`

サーバーがプロセス ID を書き込むべきファイルのパス名。サーバーは、別のディレクトリを指定する絶対パス名が指定されないかぎり、データディレクトリ内にファイルを作成します。

このオプションを指定しない場合、`mysql.server` ではデフォルト値の `host_name.pid` が使用されます。`mysqld_safe` に渡された PID ファイルの値は、`[mysqld_safe]` オプションのファイルグループで指定された値をオーバーライドします。`mysql.server` は `[mysqld]` オプションファイルグループを読み取りますが、`[mysqld_safe]` グループは読み取りません。そのため、`[mysqld_safe]` グループと `[mysqld]` グループの両方に同じ `pid-file` 設定を配置することで、`mysqld_safe` を `mysql.server` から起動したときと手動で起動したときと同じ値を取得できます。

- `service-startup-timeout=seconds`

サーバーの起動を確認するために待機する秒数。サーバーがこの時間内に起動しない場合、`mysql.server` はエラーで終了します。デフォルト値は 900 です。0 の値は、起動をまったく待機しないことを意味します。負の値はいつまでも (タイムアウトなしで) 待機することを意味します。

4.3.4 mysqld_multi — 複数の MySQL サーバーの管理

`mysqld_multi` は、Unix ソケットファイルや TCP/IP ポートでの接続を待機する複数の `mysqld` プロセスを管理するためのものです。サーバーの起動または停止、現在のステータスのレポートを実行できます。

注記

一部の Linux プラットフォームでは、RPM または Debian パッケージからの MySQL インストールに、MySQL サーバーの起動と停止を管理するための `systemd` サポートが含まれています。これらのプラットフォームでは、`mysqld_multi` は不要であるためインストールされません。`systemd` を使用した複数の MySQL インスタンスの処理の詳細は、[セクション 2.5.9 「systemd を使用した MySQL Server の管理」](#) を参照してください。

`mysqld_multi` は `[mysqldN]` という名前のグループを `my.cnf` (または `--defaults-file` オプションで指定されたファイル) から検索します。N は任意の正の整数です。この数字は、次の説明ではオプショングループ番号、または `GNR` といいます。グループ番号は、それぞれのオプショングループを識別し、起動、停止、またはステータスレポート取得の対象となるサーバーを指定するために、`mysqld_multi` の引数として使用されます。これらのグループにリストされるオプションは、`mysqld` を起動するために `[mysqld]` グループで使用するものと同じです。(たとえば [セクション 2.10.5 「MySQL を自動的に起動および停止する」](#)などを参照してください。)ただし、複数のサーバーを使用する場合は、Unix ソケットファイルや TCP/IP ポート番号などに関してそれぞれが独自のオプション値を使用する必要があります。複数サーバーの環境で、どのオプションを一意とする必要があるのかについては、[セクション 5.8 「1 つのマシン上での複数の MySQL インスタンスの実行」](#)を参照してください。

`mysqld_multi` を呼び出すには、次の構文を使用します。

```
shell> mysqld_multi [options] {start|stop|reload|report} [GNR[,GNR] ...]
```

`start`, `stop`, `reload` (停止して再起動) および `report` は、実行する操作を示します。オプション名に続く `GNR` リストに従って、指定した操作を単一のサーバーまたは複数のサーバーで実行できます。リストがない場合は、`mysqld_multi` はオプションファイル内のすべてのサーバーに対して操作を実行します。

各 `GNR` 値は、オプショングループ番号またはグループ番号の範囲を表します。値は、オプションファイル内のグループ名の最後の数字とします。たとえば、`[mysqld17]` という名前のグループの `GNR` は `17` です。番号の範囲を指定するには、最初と最後の番号をダッシュで区切ります。`GNR` 値 `10-13` は、`[mysqld10]` から `[mysqld13]` のグループを表します。コマンド行で、複数のグループまたはグループの範囲を、カンマで区切って指定できます。`GNR` リストには空白文字 (スペースまたはタブ) を使用してはいけません。空白文字からあとにあるものはすべて無視されません。

次のコマンドは `[mysqld17]` というオプショングループを使用して単一のサーバーを起動します。

```
shell> mysqld_multi start 17
```

次のコマンドは、`[mysqld8]` および `[mysqld10]` から `[mysqld13]` までのオプショングループを使用して複数のサーバーを停止します。

```
shell> mysqld_multi stop 8,10-13
```

オプションファイルをセットアップする方法の例として、次のコマンドを使用します。

```
shell> mysqld_multi --example
```

`mysqld_multi` は次のようにオプションファイルを検索します。

- `--no-defaults` では、オプションファイルは読み取られません。
- `--defaults-file=file_name` では、指定されたファイルのみが読み取られます。
- それ以外の場合は、`--defaults-extra-file=file_name` オプションで指定されたファイル (指定されている場合) を含む、標準の場所リスト内のオプションファイルが読み取られます。(オプションが複数回指定された場合は、最後の値が使用されます。)

これらのオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

読み取られるオプションファイルでは、`[mysqld_multi]` および `[mysqldN]` オプショングループが検索されます。`[mysqld_multi]` グループは、`mysqld_multi` 自身へのオプションに使用できます。`[mysqldN]` グループは、`mysqld` の特定のインスタンスに渡されるオプションに使用できます。

`[mysqld]` グループまたは `[mysqld_safe]` グループは、`mysqld` または `mysqld_safe` のすべてのインスタンスによって読み取られる共通オプションに使用できます。`--defaults-file=file_name` オプションを指定して、そのインスタンスに別の構成ファイルを使用できます。この場合、そのファイルの `[mysqld]` または `[mysqld_safe]` グループがそのインスタンスに使用されます。

`mysqld_multi` は次のオプションをサポートします。

- `--help`
ヘルプメッセージを表示して終了します。
- `--example`
サンプルのオプションファイルを表示します。
- `--log=file_name`
ログファイルの名前を指定します。ファイルが存在する場合は、ログ出力はそこに追加されます。
- `--mysqladmin=prog_name`
サーバーの停止に使用する `mysqladmin` バイナリ。

- `--mysqld=prog_name`

使用する `mysqld` バイナリ。このオプションの値として `mysqld_safe` を指定することもできます。 `mysqld_safe` を使用してサーバーを起動する場合は、対応する `[mysqldN]` オプショングループに `mysqld` オプションまたは `ledir` オプションを含めることができます。これらのオプションは、`mysqld_safe` が起動するべきサーバーの名前と、サーバーがあるディレクトリのパス名を示します。(これらのオプションに関する説明は、[セクション 4.3.2 「mysqld_safe — MySQL サーバー起動スクリプト」](#)を参照してください。) 例:

```
[mysqld38]
mysqld = mysqld-debug
ledir = /opt/local/mysql/libexec
```

- `--no-log`

ログファイルではなく、`stdout` にログを出力します。デフォルトでは、出力はログファイルに送られます。

- `--password=password`

`mysqladmin` を呼び出すときに使う MySQL アカウントのパスワード。ほかの MySQL プログラムとは異なり、このオプションではパスワード値はオプションではありません。

- `--silent`

サイレントモード。警告を無効にします。

- `--tcp-ip`

Unix ソケットファイルではなく TCP/IP ポートを介して各 MySQL サーバーに接続します。(ソケットファイルがない場合でもサーバーは稼働している可能性があります、TCP/IP ポートからのみアクセスできます。) 接続はデフォルトでは Unix ソケットファイルを使用して行われます。このオプションは `stop` 操作と `report` 操作に影響しません。

- `--user=user_name`

`mysqladmin` を呼び出すときに使う MySQL アカウントのユーザー名。

- `--verbose`

より詳細になります。

- `--version`

バージョン情報を表示して終了します。

mysqld_multi に関する注意:

- **もっとも重要:** `mysqld_multi` を使用する前に、`mysqld` サーバーに渡されるオプションの意味と、なぜ独立した `mysqld` プロセスが必要なのかを確実に理解してください。同じデータディレクトリで複数の `mysqld` サーバーを使用することの危険性に注意してください。特別な意図がないかぎり、独立のデータディレクトリを使用してください。スレッドを使用するシステムでは、同じデータディレクトリで複数のサーバーを起動してもパフォーマンスは改善されません。[セクション5.8 「1つのマシン上での複数の MySQL インスタンスの実行」](#)を参照してください。

重要

各サーバーのデータディレクトリが、その特定の `mysqld` を開始した Unix アカウントから完全にアクセス可能であることを確認してください。特別な意図がないかぎり、Unix `root` アカウントをこれに使用しないでください。[セクション6.1.5 「MySQL を通常ユーザーとして実行する方法」](#)を参照してください。

- `mysqld` サーバーを (`mysqladmin` プログラムで) 停止するのに使用する MySQL アカウントが、各サーバーに対して同じユーザー名とパスワードを持つことを確認してください。また、そのアカウントには `SHUTDOWN` 権限があることも確かめてください。管理対象のサーバーの管理アカウントのユーザー名またはパスワードが異なる場合は、各サーバーに同じユーザー名とパスワードを持つアカウントを作成するとよいでしょう。たとえば、次のコマンドをそれぞれのサーバーで実行することにより、共通の `multi_admin` アカウントをセットアップできます。

```
shell> mysql -u root -S /tmp/mysql.sock -p
Enter password:
mysql> CREATE USER 'multi_admin'@'localhost' IDENTIFIED BY 'multipass';
mysql> GRANT SHUTDOWN ON *.* TO 'multi_admin'@'localhost';
```

セクション6.2「アクセス制御とアカウント管理」を参照してください。これは、それぞれの `mysqld` サーバーで行う必要があります。それぞれに接続する場合、接続パラメータを適切に変更します。アカウント名のホスト名部分では、`mysqld_multi` を実行するホストから `multi_admin` として接続できる必要があります。

- Unix ソケットファイルと TCP/IP ポート番号は、すべての `mysqld` で異なる必要があります。(または、ホストに複数のネットワークアドレスがある場合は、異なるサーバーが異なるインターフェースをリスニングするように `bind_address` システム変数を設定できます。)
- `mysqld_safe` を使用して `mysqld` を起動している場合 (たとえば `--mysqld=mysqld_safe`)、`--pid-file` オプションは非常に重要です。すべての `mysqld` が独自のプロセス ID ファイルを持っているべきです。`mysqld` ではなく、`mysqld_safe` を使用することの利点は、`mysqld_safe` は `mysqld` のプロセスをモニターして、`kill -9` を使用したシグナル送信や、セグメンテーション違反などその他の原因でプロセスが終了した場合に、再起動することです。
- `--user` オプションを `mysqld` に対して使用する場合がありますが、そのためには `mysqld_multi` スクリプトを Unix のスーパーユーザー (`root`) として実行する必要があります。このオプションがオプションファイルにあるかどうかは問題ではなく、もしスーパーユーザーではない人が、`mysqld` プロセスを自分の Unix アカウントで起動すると、警告が出ます。

次の例は、`mysqld_multi` とともに使用するオプションファイルの設定方法を示します。`mysqld` プログラムが起動または終了する順序は、オプションファイルで指定する順序によります。グループ番号は、切れ目のないシーケンスの形式にする必要はありません。例では、最初と 5 番目の `[mysqldN]` グループは意図的に省略しています。これは、オプションファイルで「ギャップ」があっても構わないことを示しています。これにより、柔軟性が高まります。

```
# This is an example of a my.cnf file for mysqld_multi.
# Usually this file is located in home dir ~/.my.cnf or /etc/my.cnf
```

```
[mysqld_multi]
mysqld = /usr/local/mysql/bin/mysqld_safe
mysqldadmin = /usr/local/mysql/bin/mysqldadmin
user = multi_admin
password = my_password

[mysqld2]
socket = /tmp/mysql.sock2
port = 3307
pid-file = /usr/local/mysql/data2/hostname.pid2
datadir = /usr/local/mysql/data2
language = /usr/local/mysql/share/mysql/english
user = unix_user1

[mysqld3]
mysqld = /path/to/mysqld_safe
ledir = /path/to/mysqld-binary/
mysqldadmin = /path/to/mysqldadmin
socket = /tmp/mysql.sock3
port = 3308
pid-file = /usr/local/mysql/data3/hostname.pid3
datadir = /usr/local/mysql/data3
language = /usr/local/mysql/share/mysql/swedish
user = unix_user2

[mysqld4]
socket = /tmp/mysql.sock4
port = 3309
pid-file = /usr/local/mysql/data4/hostname.pid4
datadir = /usr/local/mysql/data4
language = /usr/local/mysql/share/mysql/estonia
user = unix_user3

[mysqld6]
socket = /tmp/mysql.sock6
port = 3311
pid-file = /usr/local/mysql/data6/hostname.pid6
```

```
datadir = /usr/local/mysql/data6  
language = /usr/local/mysql/share/mysql/japanese  
user = unix_user4
```

セクション4.2.2.2「オプションファイルの使用」を参照してください。

4.4 インストール関連プログラム

このセクションのプログラムは、MySQL のインストールまたはアップグレードに使用されます。

4.4.1 comp_err — MySQL エラーメッセージファイルのコンパイル

`comp_err` は、`mysqld` がさまざまなエラーコードに対して表示するエラーメッセージを判断するために使用する、`errmsg.sys` ファイルを作成します。`comp_err` は、通常 MySQL のビルド時に自動的に実行されます。MySQL ソース配布のテキスト形式のエラー情報から `errmsg.sys` ファイルをコンパイルします:

- MySQL 8.0.19 では、エラー情報は `share` ディレクトリの `messages_to_error_log.txt` および `messages_to_clients.txt` ファイルから取得されます。

エラーメッセージの定義の詳細は、これらのファイル内のコメントおよび `errmsg_readme.txt` ファイルを参照してください。

- MySQL 8.0.19 より前は、エラー情報は `sql/share` ディレクトリの `errmsg-utf8.txt` ファイルから取得されていました。

`comp_err` では、`mysqld_error.h`、`mysqld_ename.h` および `mysqld_errmsg.h` ヘッダーファイルも生成されます。

`comp_err` は次のように呼び出します。

```
shell> comp_err [options]
```

`comp_err` は次のオプションをサポートします。

- `--help, -?`

ヘルプメッセージを表示して終了します。

- `--charset=dir_name, -C dir_name`

文字セットディレクトリ。デフォルトは `../sql/share/charsets` です。

- `--debug=debug_options, -# debug_options`

デバッグのログを書き込みます。一般的な `debug_options` 文字列は `d:t:O,file_name` です。デフォルトは `d:t:O,/tmp/comp_err.trace` です。

- `--debug-info, -T`

プログラムの終了時に、デバッグ情報を出力します。

- `--errmsg-file=file_name, -H file_name`

エラーメッセージファイルの名前。デフォルトは `mysqld_errmsg.h` です。このオプションは MySQL 8.0.18 で追加されました。

- `--header-file=file_name, -H file_name`

エラーヘッダーファイルの名前です。デフォルトは `mysqld_error.h` です。

- `--in-file=file_name, -F file_name`

入力ファイルの名前です。デフォルトは `../share/errmsg-utf8.txt` です。

このオプションは MySQL 8.0.19 で削除され、`--in-file-errlog` および `--in-file-toclient` オプションに置き換えられました。

- `--in-file-errlog=file_name, -e file_name`
エラーログに書き込まれるエラーメッセージを定義する入力ファイルの名前。デフォルトは `../share/messages_to_error_log.txt` です。
このオプションは MySQL 8.0.19 で追加されました。
- `--in-file-toclient=file_name, -c file_name`
クライアントに書き込まれるエラーメッセージを定義する入力ファイルの名前。デフォルトは `../share/messages_to_clients.txt` です。
このオプションは MySQL 8.0.19 で追加されました。
- `--name-file=file_name, -N file_name`
エラー名ファイルの名前です。デフォルトは `mysqld_errname.h` です。
- `--out-dir=dir_name, -D dir_name`
出力ベースディレクトリの名前です。デフォルトは `../sql/share/` です。
- `--out-file=file_name, -O file_name`
出力ファイルの名前です。デフォルトは `errmsg.sys` です。
- `--version, -V`
バージョン情報を表示して終了します。

4.4.2 mysql_secure_installation — MySQL インストールのセキュリティー改善

このプログラムにより、次の方法で MySQL インストールのセキュリティーを改善できます。

- `root` アカウントのパスワードを設定できます。
- ローカルホスト以外からアクセス可能な `root` アカウントを削除できます。
- 匿名ユーザーアカウントを削除できます。
- `test` データベース (デフォルトでは、匿名ユーザーであっても、すべてのユーザーがアクセスできます)、および `test_` で始まる名前を持つデータベースへのアクセスを許可する権限を削除できます。

`mysql_secure_installation` により、[セクション2.10.4「初期 MySQL アカウントの保護」](#)に述べられているのと同様のセキュリティーの推奨事項を実装しやすくなります。

通常の使用方法は、ローカル MySQL サーバーに接続することです。引数なしで `mysql_secure_installation` を呼び出します:

```
shell> mysql_secure_installation
```

実行されると、`mysql_secure_installation` により、実行するアクションを決定するよう求められます。

`validate_password` コンポーネントは、パスワード強度チェックに使用できます。プラグインがインストールされていない場合、`mysql_secure_installation` はプラグインをインストールするかどうかをユーザーに確認します。後で入力したパスワードは、プラグインを使用してチェックされます (有効になっている場合)。

`--host` や `--port` などの通常の MySQL クライアントオプションのほとんどは、コマンドラインおよびオプションファイルで使用できます。たとえば、ポート 3307 を使用して IPv6 経由でローカルサーバーに接続するには、次のコマンドを使用します:

```
shell> mysql_secure_installation --host::1 --port=3307
```

`mysql_secure_installation` では、次のオプションがサポートされています。これらのオプションは、コマンドラインまたはオプションファイルの `[mysql_secure_installation]` および `[client]` グループで指定できます。MySQL プログラムによって使用されるオプションファイルの詳細については、[セクション4.2.2.2「オプションファイルの使用」](#)を参照してください。

表 4.8 「mysql_secure_installation オプション」

オプション名	説明	導入
<code>--defaults-extra-file</code>	通常のオプションファイルに加えて、名前付きオプションファイルを読み取ります	
<code>--defaults-file</code>	指名されたオプションファイルのみを読み取る	
<code>--defaults-group-suffix</code>	オプショングループのサフィクス値	
<code>--help</code>	ヘルプメッセージを表示して終了	
<code>--host</code>	MySQL サーバーがあるホスト	
<code>--no-defaults</code>	オプションファイルを読み取らない	
<code>--password</code>	受け入れられましたが、常に見捨てられます。mysql_secure_installation が起動されるたびに、ユーザーはパスワードの入力を求められます	
<code>--port</code>	接続用の TCP/IP ポート番号	
<code>--print-defaults</code>	デフォルトオプションの印刷	
<code>--protocol</code>	使用するトランスポートプロトコル	
<code>--socket</code>	使用する Unix ソケットファイルまたは Windows 名前付きパイプ	
<code>--ssl-ca</code>	信頼できる SSL 認証局のリストを含むファイル	
<code>--ssl-capath</code>	信頼できる SSL 認証局の証明書ファイルを含むディレクトリ	
<code>--ssl-cert</code>	X.509 証明書を含むファイル	
<code>--ssl-cipher</code>	接続の暗号化に許可される暗号	
<code>--ssl-crl</code>	証明書失効リストを含むファイル	
<code>--ssl-crlpath</code>	証明書失効リストファイルを含むディレクトリ	
<code>--ssl-fips-mode</code>	クライアント側で FIPS モードを有効にするかどうか	
<code>--ssl-key</code>	X.509 キーを含むファイル	
<code>--tls-ciphersuites</code>	暗号化された接続に許可される TLSv1.3 暗号スイート	8.0.16
<code>--tls-version</code>	暗号化された接続に許可される TLS プロトコル	
<code>--use-default</code>	ユーザーの対話性なしで実行	
<code>--user</code>	サーバーへの接続時に使用する MySQL ユーザー名	

- `--help, -?`

ヘルプメッセージを表示して終了します。

- `--defaults-extra-file=file_name`

このオプションファイルは、グローバルオプションファイルのあとに読み取りますが、(UNIX では) ユーザーオプションファイルの前に読み取るようにしてください。ファイルが存在しないかアクセスできない場合、エラーが発生します。`file_name` は、フルパス名でなく相対パス名として指定された場合、現行ディレクトリを基準にして解釈されます。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--defaults-file=file_name`

指定されたオプションファイルのみ使用します。ファイルが存在しないかアクセスできない場合、エラーが発生します。`file_name` は、フルパス名でなく相対パス名として指定された場合、現行ディレクトリを基準にして解釈されます。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--defaults-group-suffix=str`

通常のオプショングループだけでなく、通常の名前に `str` のサフィクスが付いたグループも読み取ります。たとえば、`mysql_secure_installation` は通常、`[client]` および `[mysql_secure_installation]` グループを読み取ります。`--defaults-group-suffix=_other` オプションが指定されている場合、`mysql_secure_installation` は `[client_other]` および `[mysql_secure_installation_other]` グループも読み取ります。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--host=host_name, -h host_name`

指定されたホストの MySQL サーバーに接続します。

- `--no-defaults`

オプションファイルを読み取りません。オプションファイルから不明のオプションを読み取ることが原因でプログラムの起動に失敗する場合、`--no-defaults` を使用して、オプションを読み取らないようにすることができます。

例外として、`.mylogin.cnf` ファイルは、存在する場合はすべての場合に読み取られます。これにより、`--no-defaults` が使用された場合に、コマンド行よりも安全な方法でパスワードを指定できます。`.mylogin.cnf` は `mysql_config_editor` ユーティリティーによって作成されます。[セクション4.6.7「mysql_config_editor — MySQL 構成ユーティリティー」](#)を参照してください。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--password=password, -p password`

このオプションは受け入れられますが、無視されます。このオプションを使用するかどうかにかかわらず、`mysql_secure_installation` は常にユーザーにパスワードの入力を求めます。

- `--port=port_num, -P port_num`

TCP/IP 接続の場合、使用するポート番号。

- `--print-defaults`

プログラム名と、オプションファイルから受け取るすべてのオプションを出力します。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

サーバーへの接続に使用するトランスポートプロトコル。これは、他の接続パラメータが通常、必要なプロトコル以外のプロトコルを使用する場合に便利です。許可される値の詳細は、[セクション4.2.7「接続トランスポートプロトコル」](#)を参照してください。

- `--socket=path, -S path`

`localhost` への接続用に使用する、Unix ソケットファイル、または Windows では使用する名前付きパイプの名前。

Windows では、このオプションは、名前付きパイプ接続をサポートするために `named_pipe` システム変数を有効にしてサーバーを起動した場合にのみ適用されます。また、接続は、`named_pipe_full_access_group` システム変数で指定された Windows グループのメンバーである必要があります。

- `--ssl*`

`--ssl` で始まるオプションは、SSL を使用してサーバーに接続するかどうかを指定し、SSL 鍵および証明書を検索する場所を指定します。 [暗号化接続のコマンドオプション](#) を参照してください。

- `--ssl-fips-mode={OFF|ON|STRICT}`

クライアント側で FIPS モードを有効にするかどうかを制御します。 `--ssl-fips-mode` オプションは、暗号化された接続の確立には使用されず、許可する暗号化操作に影響する点で、他の `--ssl-xxx` オプションとは異なります。 [セクション6.8「FIPS のサポート」](#) を参照してください。

次の `--ssl-fips-mode` 値を使用できます:

- **OFF**: FIPS モードを無効にします。
- **ON**: FIPS モードを有効にします。
- **STRICT**: 「strict」 FIPS モードを有効にします。

注記

OpenSSL FIPS オブジェクトモジュールが使用できない場合、`--ssl-fips-mode` に許可される値は **OFF** のみです。この場合、`--ssl-fips-mode` を **ON** または **STRICT** に設定すると、クライアントは起動時に警告を生成し、FIPS 以外のモードで動作します。

- `--tls-ciphersuites=ciphersuite_list`

TLSv1.3 を使用する暗号化された接続に許可される暗号スイート。値は、コロンで区切られた 1 つ以上の暗号スイート名のリストです。このオプションに指定できる暗号スイートは、MySQL のコンパイルに使用される SSL ライブラリによって異なります。詳細は、[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#) を参照してください。

このオプションは MySQL 8.0.16 で追加されました。

- `--tls-version=protocol_list`

暗号化された接続に許可される TLS プロトコル。値は、1 つまたは複数のコンマ区切りプロトコル名のリストです。このオプションに指定できるプロトコルは、MySQL のコンパイルに使用される SSL ライブラリによって異なります。詳細は、[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#) を参照してください。

- `--use-default`

非対話形式で実行します。このオプションは、自動インストール操作に使用できます。

- `--user=user_name, -u user_name`

サーバーへの接続に使用する MySQL アカウントのユーザー名。

4.4.3 mysql_ssl_rsa_setup — SSL/RSA ファイルの作成

このプログラムは、SSL を使用したセキュアな接続をサポートするために必要な SSL 証明書およびキーファイルと RSA キーペアファイル、および暗号化されていない接続を介した RSA を使用したセキュアなパスワード交換を作成します (それらのファイルがない場合)。既存の SSL ファイルの有効期限が切れている場合、`mysql_ssl_rsa_setup` を使用して新しい SSL ファイルを作成することもできます。

注記

`mysql_ssl_rsa_setup` では `openssl` コマンドが使用されるため、その使用はマシンに OpenSSL がインストールされている必要があります。

OpenSSL を使用してコンパイルされた MySQL ディストリビューションの SSL および RSA ファイルを生成する別の方法は、サーバーで自動的に生成することです。 [セクション 6.3.3.1 「MySQL を使用した SSL および RSA 証明書とキーの作成」](#) を参照してください。

重要

`mysql_ssl_rsa_setup` では、必要なファイルの生成が容易になるため、SSL の使用によるバリアの削減に役立ちます。ただし、`mysql_ssl_rsa_setup` によって生成される証明書は自己署名付きであり、これはあまりセキュアではありません。`mysql_ssl_rsa_setup` によって作成されたファイルの使用経験がある場合は、登録された認証局から CA 証明書を取得することを検討してください。

次のように `mysql_ssl_rsa_setup` を起動します：

```
shell> mysql_ssl_rsa_setup [options]
```

一般的なオプションは、ファイルを作成する場所を指定する `--datadir` と、`mysql_ssl_rsa_setup` が実行する `openssl` コマンドを表示する `--verbose` です。

`mysql_ssl_rsa_setup` は、デフォルトのファイル名セットを使用して SSL および RSA ファイルの作成を試みます。これは次のように機能します：

1. `mysql_ssl_rsa_setup` は、`PATH` 環境変数で指定された場所で `openssl` バイナリをチェックします。`openssl` が見つからない場合、`mysql_ssl_rsa_setup` は何も行いません。`openssl` が存在する場合、`mysql_ssl_rsa_setup` は `--datadir` オプションで指定された MySQL データディレクトリ、または `--datadir` オプションが指定されていない場合はコンパイル済みデータディレクトリ内でデフォルトの SSL および RSA ファイルを検索します。

2. `mysql_ssl_rsa_setup` は、次の名前の SSL ファイルのデータディレクトリをチェックします：

```
ca.pem
server-cert.pem
server-key.pem
```

3. これらのファイルのいずれかが存在する場合、`mysql_ssl_rsa_setup` は SSL ファイルを作成しません。それ以外の場合は、`openssl` を起動して作成し、さらにいくつかの追加ファイルを作成します：

```
ca.pem          Self-signed CA certificate
ca-key.pem      CA private key
server-cert.pem Server certificate
server-key.pem  Server private key
client-cert.pem Client certificate
client-key.pem  Client private key
```

これらのファイルにより、SSL を使用したセキュアなクライアント接続が可能になります。[セクション 6.3.1 「暗号化接続を使用するための MySQL の構成」](#) を参照してください。

4. `mysql_ssl_rsa_setup` は、RSA ファイルのデータディレクトリを次の名前でチェックします：

```
private_key.pem Private member of private/public key pair
public_key.pem  Public member of private/public key pair
```

5. これらのファイルのいずれかが存在する場合、`mysql_ssl_rsa_setup` は RSA ファイルを作成しません。それ以外の場合は、`openssl` を起動して作成します。これらのファイルを使用すると、`sha256_password` または `caching_sha2_password` プラグインによって認証されたアカウントに対して RSA over unencrypted 接続を使用したセキュアなパスワード交換が可能になります。[セクション 6.4.1.3 「SHA-256 プラガブル認証」](#) および [セクション 6.4.1.2 「SHA-2 プラガブル認証のキャッシュ」](#) を参照してください。

`mysql_ssl_rsa_setup` によって作成されるファイルの特性の詳細は、[セクション 6.3.3.1 「MySQL を使用した SSL および RSA 証明書とキーの作成」](#) を参照してください。

起動時に、`--ssl` 以外の明示的な SSL オプションが指定されていない場合 (おそらく `ssl_cipher` とともに)、MySQL サーバーは `mysql_ssl_rsa_setup` によって作成された SSL ファイルを自動的に使用して SSL を有効にします。ファイルを明示的に指定する場合は、起動時に `--ssl-ca`、`--ssl-cert` および `--ssl-key` オプションを使用してクライアントを起動し、`ca.pem`、`server-cert.pem` および `server-key.pem` ファイルにそれぞれ名前を付けます。

また、明示的な RSA オプションが指定されていない場合、サーバーは `mysql_ssl_rsa_setup` によって作成された RSA ファイルを自動的に使用して RSA を有効にします。

サーバーで SSL が有効になっている場合、クライアントはデフォルトで SSL を使用して接続します。証明書ファイルとキーファイルを明示的に指定するには、`--ssl-ca`、`--ssl-cert` および `--ssl-key` オプションを使用して、`ca.pem`、`client-cert.pem` および `client-key.pem` ファイルにそれぞれ名前を付けます。ただし、`mysql_ssl_rsa_setup` ではデフォルトでこれらのファイルがデータディレクトリに作成されるため、最初に追加のクライアント設定が必要になる場合があります。データディレクトリのアクセス権は通常、MySQL サーバーを実行しているシステムアカウントへのアクセスのみを有効にするため、クライアントプログラムはそこにあるファイルを使用できません。ファイルを使用可能にするには、クライアントから読み取り可能な（書き込み可能ではない）ディレクトリにファイルをコピーします：

- ローカルクライアントの場合は、MySQL インストールディレクトリを使用できます。たとえば、データディレクトリがインストールディレクトリのサブディレクトリで、現在の場所がデータディレクトリである場合は、次のようにファイルをコピーできます：

```
cp ca.pem client-cert.pem client-key.pem ..
```

- リモートクライアントの場合、転送中に改ざんされないように、セキュアなチャンネルを使用してファイルを配布します。

MySQL インストールに使用される SSL ファイルの有効期限が切れている場合は、`mysql_ssl_rsa_setup` を使用して新しい SSL ファイルを作成できます：

- サーバーを停止します。
- 既存の SSL ファイルの名前を変更するか、削除します。最初にそれらのバックアップを作成することをお勧めします。（RSA ファイルは期限切れにならないため、削除する必要はありません。`mysql_ssl_rsa_setup` では、それらが存在し、上書きされないことを確認できます。）
- `--datadir` オプションを指定して `mysql_ssl_rsa_setup` を実行し、新しいファイルを作成する場所を指定します。
- サーバーを再起動します。

`mysql_ssl_rsa_setup` では、次のコマンドラインオプションがサポートされています。これらのオプションは、コマンドラインまたはオプションファイルの `[mysql_ssl_rsa_setup]` および `[mysqld]` グループで指定できます。MySQL プログラムによって使用されるオプションファイルの詳細については、[セクション4.2.2.2「オプションファイルの使用」](#)を参照してください。

表 4.9 「mysql_ssl_rsa_setup オプション」

オプション名	説明
<code>--datadir</code>	データディレクトリへのパス
<code>--help</code>	ヘルプメッセージを表示して終了
<code>--suffix</code>	X.509 証明書共通名属性の接尾辞
<code>--uid</code>	ファイル権限に使用する有効なユーザーの名前
<code>--verbose</code>	冗長モード
<code>--version</code>	バージョン情報を表示して終了

- `--help, ?`

ヘルプメッセージを表示して終了します。

- `--datadir=dir_name`

`mysql_ssl_rsa_setup` がデフォルトの SSL および RSA ファイルをチェックするディレクトリへのパス。ファイルがない場合は、そこにファイルを作成する必要があります。デフォルトは、コンパイルされたデータディレクトリです。

- `--suffix=str`

X.509 証明書の共通名属性の接尾辞。接尾辞の値は 17 文字に制限されています。デフォルトは、MySQL のバージョン番号に基づきます。

- `--uid=name, -v`

作成されたファイルの所有者である必要があるユーザーの名前。値はユーザー名であり、数値のユーザー ID ではありません。このオプションが指定されていない場合、`mysql_ssl_rsa_setup` によって作成されたファイルは、それを実行するユーザーが所有します。このオプションは、`chown()` システムコールをサポートするシステムで `root` としてプログラムを実行する場合にのみ有効です。

- `--verbose, -v`

冗長モード。プログラムの動作についてより多くの出力を生成します。たとえば、プログラムは実行する `openssl` コマンドを示し、いくつかのデフォルトファイルがすでに存在するため、SSL または RSA ファイルの作成をスキップするかどうかを示す出力を生成します。

- `--version, -V`

バージョン情報を表示して終了します。

4.4.4 mysql_tzinfo_to_sql — タイムゾーンテーブルのロード

`mysql_tzinfo_to_sql` プログラムは、`mysql` データベースに、タイムゾーンテーブルをロードします。zoneinfo データベース (タイムゾーンを記述するファイルのセット) があるシステムで使用します。このようなシステムの例には、Linux、FreeBSD、Solaris および macOS があります。これらのファイルの適切な場所の 1 つは `/usr/share/zoneinfo` ディレクトリです (Solaris では `/usr/share/lib/zoneinfo`)。zoneinfo データベースがないシステムの場合には、[セクション5.1.15「MySQL Server でのタイムゾーンのサポート」](#) で説明するダウンロード可能なパッケージを使用できます。

`mysql_tzinfo_to_sql` はいくつかの方法で呼び出せます。

```
shell> mysql_tzinfo_to_sql tz_dir
shell> mysql_tzinfo_to_sql tz_file tz_name
shell> mysql_tzinfo_to_sql --leap tz_file
```

最初の呼び出し構文は、zoneinfo ディレクトリのパス名を `mysql_tzinfo_to_sql` に渡し、出力を `mysql` プログラムに送信します。例:

```
shell> mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root mysql
```

`mysql_tzinfo_to_sql` は、システムのタイムゾーンファイルを読み取り、そのファイルから SQL ステートメントを生成します。`mysql` はこれらのステートメントを処理して、タイムゾーンテーブルをロードします。

2 番目の構文は、`mysql_tzinfo_to_sql` がタイムゾーン名 `tz_name` に対応する単一のタイムゾーンファイル `tz_file` をロードします。

```
shell> mysql_tzinfo_to_sql tz_file tz_name | mysql -u root mysql
```

うるう秒を考慮する必要がある場合は、3 番目の構文を使用して `mysql_tzinfo_to_sql` を呼び出します。これはうるう秒の情報を初期化します。`tz_file` はタイムゾーンファイルの名前です。

```
shell> mysql_tzinfo_to_sql --leap tz_file | mysql -u root mysql
```

`mysql_tzinfo_to_sql` の実行後、以前にキャッシュしたすべてのタイムゾーンデータを使用し続けないように、サーバーを再起動することをお勧めします。

4.4.5 mysql_upgrade — MySQL テーブルのチェックとアップグレード

注記

MySQL 8.0.16 では、MySQL サーバーは以前に `mysql_upgrade` によって処理されたアップグレードタスクを実行します (詳細は、[セクション2.11.3「MySQL のアップグレードプロセスの内容」](#) を参照)。したがって、`mysql_upgrade` は不要であり、そのバージョンで非推奨になりました。将来のバージョンの MySQL で削除される予定です。`mysql_upgrade` はアップグレードタスクを実行しなくなったため、無条件にステータス 0 で終了します。

MySQL をアップグレードするたびに、`mysql_upgrade` を実行する必要があります。これにより、アップグレードした MySQL サーバーとの非互換性が検索されます:

- 追加された可能性のある新しい権限または機能を利用できるように、`mysql` スキーマ内のシステムテーブルがアップグレードされます。
- パフォーマンススキーマ、`INFORMATION_SCHEMA`、および `sys` スキーマがアップグレードされます。
- ユーザースキーマを調べます。

`mysql_upgrade` は、テーブルに非互換性がある可能性が見つかった場合はテーブルのチェックを実行し、問題が検出された場合はテーブルの修復を試みます。テーブルを修復できない場合は、手動でテーブルを修復する方法について、[セクション2.11.13「テーブルまたはインデックスの再作成または修復」](#)を参照してください。

`mysql_upgrade` は MySQL サーバーと直接通信し、アップグレードの実行に必要な SQL ステートメントを送信しません。

注意

アップグレードを実行する前に、必ず現在の MySQL インストールをバックアップするようにしてください。[セクション7.2「データベースバックアップ方法」](#)を参照してください。

一部のアップグレードの非互換性では、MySQL インストールをアップグレードして `mysql_upgrade` を実行する前に特別な処理が必要になる場合があります。このような非互換性が、使用しているインストールに該当するかどうかの判断、およびその対処方法については、[セクション2.11「MySQL のアップグレード」](#)を参照してください。

次のように `mysql_upgrade` を使用します:

1. サーバーが実行されていることを確認します。
2. `mysql_upgrade` を起動して、`mysql` スキーマのシステムテーブルをアップグレードし、他のスキーマのテーブルをチェックおよび修復します:

```
shell> mysql_upgrade [options]
```

3. システムテーブルの変更を有効にするには、サーバーを停止して再起動します。

アップグレードする MySQL サーバーインスタンスが複数ある場合は、目的の各サーバーへの接続に適した接続パラメータを使用して `mysql_upgrade` を起動します。たとえば、ローカルホストで 3306 から 3308 までのポートでサーバーが稼働している場合、適切なポートに接続してそれぞれをアップグレードします。

```
shell> mysql_upgrade --protocol=tcp -P 3306 [other_options]
shell> mysql_upgrade --protocol=tcp -P 3307 [other_options]
shell> mysql_upgrade --protocol=tcp -P 3308 [other_options]
```

Unix でのローカルホスト接続では、`--protocol=tcp` オプションを使用すると、Unix ソケットファイルではなく TCP/IP を強制的に使用して接続が行われます。

デフォルトでは、`mysql_upgrade` は MySQL `root` ユーザーとして実行されます。`mysql_upgrade` の実行時に `root` パスワードの有効期限が切れた場合は、パスワードの有効期限が切れ、`mysql_upgrade` が失敗したことを示すメッセージが表示されます。これを修正するには、`root` パスワードをリセットして有効期限を解除し、`mysql_upgrade` を再度実行します。まず、`root` としてサーバーに接続します:

```
shell> mysql -u root -p
Enter password: **** <- enter root password here
```

`ALTER USER` を使用してパスワードをリセットします:

```
mysql> ALTER USER USER() IDENTIFIED BY 'root-password';
```

次に、`mysql` を終了し、`mysql_upgrade` を再度実行します:

```
shell> mysql_upgrade [options]
```

注記

特定のストレージエンジン (`MyISAM` など) を無効にするように `disabled_storage_engines` システム変数を設定してサーバーを実行すると、`mysql_upgrade` が次のようなエラーで失敗することがあります:

```
mysql_upgrade: [ERROR] 3161: Storage engine MyISAM is disabled
(Table creation is disallowed).
```

これを処理するには、[disabled_storage_engines](#) を無効にしてサーバーを再起動します。これで、`mysql_upgrade` を正常に実行できるようになります。その後、[disabled_storage_engines](#) を元の値に設定してサーバーを再起動します。

`--upgrade-system-tables` オプションを指定して起動しないかぎり、`mysql_upgrade` は必要に応じてすべてのユーザースキーマのすべてのテーブルを処理します。テーブルチェックの完了には時間がかかる場合があります。各テーブルはロックされるため、処理中にほかのセッションで使用することはできません。チェックと修復の処理には時間がかかることがあり、特に大きなテーブルでは長い時間を要する可能性があります。テーブルチェックでは、`CHECK TABLE` ステートメントの `FOR UPGRADE` オプションを使用します。このオプションに必要な内容の詳細は、[セクション13.7.3.2「CHECK TABLE ステートメント」](#) を参照してください。

`mysql_upgrade` は、チェックおよび修復されたすべてのテーブルを現在の MySQL バージョン番号でマークします。これにより、次回同じバージョンのサーバーで `mysql_upgrade` を実行するときに、特定のテーブルを再度チェックまたは修復する必要があるかどうかを判断できます。

`mysql_upgrade` は、MySQL のバージョン番号をデータディレクトリ内の `mysql_upgrade_info` という名前のファイルに保存します。これは、テーブルのチェックをスキップできるように、すべてのテーブルがこのリリースに関してチェック済みかどうかを迅速にチェックするために使用されます。このファイルを無視してとにかくチェックを実行するには、`--force` オプションを使用します。

注記

`mysql_upgrade_info` ファイルは非推奨です。将来のバージョンの MySQL で削除される予定です。

`mysql_upgrade` は `mysql.user` システムテーブルの行をチェックし、`plugin` カラムが空の行について、資格証明がそのプラグインと互換性のあるハッシュ形式を使用している場合は、そのカラムを `'mysql_native_password'` に設定します。4.1 より前のパスワードハッシュを持つ行は、手動でアップグレードする必要があります。

`mysql_upgrade` では、タイムゾーンテーブルまたはヘルプテーブルの内容はアップグレードされません。アップグレード手順については、[セクション5.1.15「MySQL Server でのタイムゾーンのサポート」](#) および [セクション5.1.17「サーバー側ヘルプのサポート」](#) を参照してください。

`--skip-sys-schema` オプションで呼び出されないかぎり、`mysql_upgrade` は `sys` スキーマがインストールされていない場合はインストールし、それ以外の場合は現在のバージョンにアップグレードします。`sys` スキーマは存在するが `version` ビューがない場合、その存在しないという前提で、ユーザーが作成したスキーマがあるとエラーが発生します:

```
A sys schema exists with no sys.version view. If
you have a user created sys schema, this must be renamed for the
upgrade to succeed.
```

この場合にアップグレードするには、まず既存の `sys` スキーマを削除するか、名前を変更します。

`mysql_upgrade` は次のオプションをサポートします。これらはコマンド行またはオプションファイルの `[mysql_upgrade]` グループおよび `[client]` グループで指定できます。MySQL プログラムによって使用されるオプションファイルの詳細については、[セクション4.2.2.2「オプションファイルの使用」](#) を参照してください。

表 4.10 「mysql_upgrade オプション」

オプション名	説明	導入	非推奨
<code>--bind-address</code>	指定されたネットワークインタフェースを使用して MySQL サーバーに接続		
<code>--character-sets-dir</code>	文字セットがインストールされているディレクトリ		
<code>--compress</code>	クライアントとサーバー間で送信される情報をすべて圧縮		8.0.18

オプション名	説明	導入	非推奨
<code>--compression-algorithms</code>	サーバーへの接続に許可される圧縮アルゴリズム	8.0.18	
<code>--debug</code>	デバッグログの書込み		
<code>--debug-check</code>	プログラムの終了時にデバッグ情報を出力		
<code>--debug-info</code>	プログラムの終了時に、デバッグ情報、メモリー、および CPU の統計を出力		
<code>--default-auth</code>	使用する認証プラグイン		
<code>--default-character-set</code>	デフォルト文字セットを指定		
<code>--defaults-extra-file</code>	通常のオプションファイルに加えて、名前付きオプションファイルを読み取ります		
<code>--defaults-file</code>	指名されたオプションファイルのみを読み取る		
<code>--defaults-group-suffix</code>	オプショングループのサフィクス値		
<code>--force</code>	mysql_upgrade が現在の MySQL バージョンに対してすでに実行されている場合でも実行を強制		
<code>--get-server-public-key</code>	サーバーから RSA 公開キーをリクエスト		
<code>--help</code>	ヘルプメッセージを表示して終了		
<code>--host</code>	MySQL サーバーがあるホスト		
<code>--login-path</code>	ログインパスオプションを .mylogin.cnf から読み取り		
<code>--max-allowed-packet</code>	サーバーとの間で送受信するパケットの最大長		
<code>--net-buffer-length</code>	TCP/IP とソケット通信のバッファサイズ		
<code>--no-defaults</code>	オプションファイルを読み取らない		
<code>--password</code>	サーバーに接続する際に使用するパスワード		
<code>--pipe</code>	名前付きパイプを使用してサーバに接続する (Windows のみ)		
<code>--plugin-dir</code>	プラグインがインストールされているディレクトリ		
<code>--port</code>	接続用の TCP/IP ポート番号		
<code>--print-defaults</code>	デフォルトオプションの印刷		

オプション名	説明	導入	非推奨
<code>--protocol</code>	使用するトランスポートプロトコル		
<code>--server-public-key-path</code>	RSA 公開鍵を含むファイルへのパス名		
<code>--shared-memory-base-name</code>	共有メモリー接続用の共有メモリー名 (Windows のみ)		
<code>--skip-sys-schema</code>	sys スキーマをインストールまたはアップグレードしない		
<code>--socket</code>	使用する Unix ソケットファイルまたは Windows 名前付きパイプ		
<code>--ssl-ca</code>	信頼できる SSL 認証局のリストを含むファイル		
<code>--ssl-capath</code>	信頼できる SSL 認証局の証明書ファイルを含むディレクトリ		
<code>--ssl-cert</code>	X.509 証明書を含むファイル		
<code>--ssl-cipher</code>	接続の暗号化に許可される暗号		
<code>--ssl-crl</code>	証明書失効リストを含むファイル		
<code>--ssl-crlpath</code>	証明書失効リストファイルを含むディレクトリ		
<code>--ssl-fips-mode</code>	クライアント側で FIPS モードを有効にするかどうか		
<code>--ssl-key</code>	X.509 キーを含むファイル		
<code>--ssl-mode</code>	サーバーへの接続に必要なセキュリティ状態		
<code>--tls-ciphersuites</code>	暗号化された接続に許可される TLSv1.3 暗号スイート	8.0.16	
<code>--tls-version</code>	暗号化された接続に許可される TLS プロトコル		
<code>--upgrade-system-tables</code>	システムテーブルのみを更新し、ユーザースキーマは更新しない		
<code>--user</code>	サーバーへの接続時に使用する MySQL ユーザー名		
<code>--verbose</code>	冗長モード		
<code>--version-check</code>	適切なサーバーバージョンをチェック		
<code>--write-binlog</code>	すべてのステートメントをバイナリログに書き込み		
<code>--zstd-compression-level</code>	zstd 圧縮を使用するサーバーへの接続の圧縮レベル	8.0.18	

- `--help`

短いヘルプメッセージを表示して終了します。

- `--bind-address=ip_address`

複数のネットワークインタフェースを持つコンピュータで、このオプションを使用して、MySQL サーバーへの接続に使用するインタフェースを選択します。

- `--character-sets-dir=dir_name`

文字セットがインストールされているディレクトリ。 [セクション10.15「文字セットの構成」](#) を参照してください。

- `--compress, -C`

可能であれば、クライアントとサーバーの間で送信されるすべての情報を圧縮します。 [セクション4.2.8「接続圧縮制御」](#) を参照してください。

MySQL 8.0.18 では、このオプションは非推奨です。MySQL の将来のバージョンで削除されることが予想されます。 [レガシー接続圧縮の構成](#) を参照してください。

- `--compression-algorithms=value`

サーバーへの接続に許可される圧縮アルゴリズム。使用可能なアルゴリズムは、`protocol_compression_algorithms` システム変数の場合と同じです。デフォルト値は `uncompressed` です。

詳細は、[セクション4.2.8「接続圧縮制御」](#) を参照してください。

このオプションは MySQL 8.0.18 で追加されました。

- `--debug=[debug_options], -# [debug_options]`

デバッグのログを書き込みます。一般的な `debug_options` 文字列は `d:t:o,file_name` です。デフォルトは `d:t:O,/tmp/mysql_upgrade.trace` です。

- `--debug-check`

プログラムの終了時に、デバッグ情報を出力します。

- `--debug-info, -T`

プログラムの終了時に、デバッグ情報とメモリーおよび CPU 使用率の統計を出力します。

- `--default-auth=plugin`

使用するクライアント側認証プラグインに関するヒント。 [セクション6.2.17「プラグイン認証」](#) を参照してください。

- `--default-character-set=charset_name`

`charset_name` をデフォルト文字セットとして使用します。 [セクション10.15「文字セットの構成」](#) を参照してください。

- `--defaults-extra-file=file_name`

このオプションファイルは、グローバルオプションファイルのあとに読み取りますが、(UNIX では) ユーザーオプションファイルの前に読み取るようにしてください。ファイルが存在しないかアクセスできない場合、エラーが発生します。`file_name` は、フルパス名でなく相対パス名として指定された場合、現行ディレクトリを基準にして解釈されます。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--defaults-file=file_name`

指定されたオプションファイルのみ使用します。ファイルが存在しないかアクセスできない場合、エラーが発生します。file_name は、フルパス名でなく相対パス名として指定された場合、現行ディレクトリを基準にして解釈されます。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--defaults-group-suffix=str`

通常のオプショングループだけでなく、通常の名前に str のサフィクスが付いたグループも読み取ります。たとえば、mysql_upgrade は通常 [client] グループおよび [mysql_upgrade] グループを読み取ります。--defaults-group-suffix=_other オプションを指定した場合、mysql_upgrade は [client_other] グループおよび [mysql_upgrade_other] グループも読み取ります。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--force`

mysql_upgrade_info ファイルを無視し、MySQL の現在のバージョンに対して mysql_upgrade を実行済みでも強制的に実行します。

- `--get-server-public-key`

RSA キーベースのパスワード交換に必要な公開キーをサーバーにリクエストします。このオプションは、`caching_sha2_password` 認証プラグインで認証されるクライアントに適用されます。そのプラグインの場合、サーバーは要求されないかぎり公開鍵を送信しません。このオプションは、そのプラグインで認証されないアカウントでは無視されます。クライアントがセキュアな接続を使用してサーバーに接続する場合と同様に、RSA ベースのパスワード交換を使用しない場合も無視されます。

`--server-public-key-path=file_name` が指定され、有効な公開キーファイルが指定されている場合は、`--get-server-public-key` よりも優先されます。

`caching_sha2_password` プラグインの詳細は、[セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#)を参照してください。

- `--host=host_name, -h host_name`

指定されたホストの MySQL サーバーに接続します。

- `--login-path=name`

`.mylogin.cnf` ログインパスファイルの指定されたログインパスからオプションを読み取ります。「ログインパス」は、接続先の MySQL サーバーおよび認証に使用するアカウントを指定するオプションを含むオプショングループです。ログインパスファイルを作成または変更するには、`mysql_config_editor` ユーティリティを使用します。[セクション4.6.7「mysql_config_editor — MySQL 構成ユーティリティー」](#)を参照してください。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--max-allowed-packet=value`

クライアント/サーバー通信のバッファの最大サイズ。デフォルトの値は 24M バイトです。最小値と最大値は 4KB と 2GB です。

- `--net-buffer-length=value`

クライアント/サーバー通信のバッファの初期サイズ。デフォルト値は 1MB - 1KB です。最小値と最大値は 4KB と 16MB です。

- `--no-defaults`

オプションファイルを読み取りません。オプションファイルから不明のオプションを読み取ることが原因でプログラムの起動に失敗する場合、`--no-defaults` を使用して、オプションを読み取らないようにすることができます。

例外として、`.mylogin.cnf` ファイルは、存在する場合はすべての場合に読み取られます。これにより、`--no-defaults` が使用されたとしても、コマンド行よりも安全な方法でパスワードを指定できます。(`.mylogin.cnf` は `mysql_config_editor` コーティリティーによって作成されます。 [セクション4.6.7「mysql_config_editor — MySQL 構成ユーティリティー」](#) を参照してください)。

このオプションおよびその他のオプションファイルオプションの詳細は、 [セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--password=[password], -p[password]`

サーバーへの接続に使用される MySQL アカウントのパスワード。パスワード値はオプションです。指定しない場合、`mysql_upgrade` によってプロンプトが表示されます。指定する場合は、`--password=` または `-p` とそれに続くパスワードの間にスペースなしが存在する必要があります。パスワードオプションを指定しない場合、デフォルトではパスワードは送信されません。

コマンド行でのパスワード指定は、セキュアでないと考えるべきです。コマンド行でパスワードを指定しないようにするには、オプションファイルを使用します。 [セクション6.1.2.1「パスワードセキュリティのためのエンドユーザーガイドライン」](#) を参照してください。

パスワードがなく、`mysql_upgrade` でパスワードの入力を求められないように明示的に指定するには、`--skip-password` オプションを使用します。

- `--pipe, -W`

Windows で、名前付きパイプを使用してサーバーに接続します。このオプションは、ネームパイプ接続をサポートするために `named_pipe` システム変数を有効にしてサーバーを起動した場合にのみ適用されます。また、接続を行うユーザーは、`named_pipe_full_access_group` システム変数で指定された Windows グループのメンバーである必要があります。

- `--plugin-dir=dir_name`

プラグインを検索するディレクトリ。このオプションは、`--default-auth` オプションを使用して認証プラグインを指定しても、`mysql_upgrade` がそれを検出しない場合に指定します。 [セクション6.2.17「プラグイン認証」](#) を参照してください。

- `--port=port_num, -P port_num`

TCP/IP 接続の場合、使用するポート番号。

- `--print-defaults`

プログラム名と、オプションファイルから受け取るすべてのオプションを出力します。

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

サーバーへの接続に使用するトランスポートプロトコル。これは、他の接続パラメータが通常、必要なプロトコル以外のプロトコルを使用する場合に便利です。許可される値の詳細は、 [セクション4.2.7「接続トランスポートプロトコル」](#) を参照してください。

- `--server-public-key-path=file_name`

RSA キーベースのパスワード交換のためにサーバーが必要とする公開キーのクライアント側コピーを含む、PEM 形式のファイルへのパス名。このオプションは、`sha256_password` または `caching_sha2_password` 認証プラグインで認証されるクライアントに適用されます。これらのプラグインのいずれかで認証されないアカウント

では、このオプションは無視されます。クライアントがセキュアな接続を使用してサーバーに接続する場合と同様に、RSA ベースのパスワード交換を使用しない場合も無視されます。

`--server-public-key-path=file_name` が指定され、有効な公開キーファイルが指定されている場合は、`--get-server-public-key` よりも優先されます。

`sha256_password` の場合、このオプションは、MySQL が OpenSSL を使用して構築された場合にのみ適用されません。

`sha256_password` および `caching_sha2_password` プラグインの詳細は、[セクション6.4.1.3「SHA-256 プラガブル認証」](#) および [セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#) を参照してください。

- `--shared-memory-base-name=name`

Windows の場合、共有メモリを使用してローカルサーバーに接続するために使用する共有メモリ名。デフォルト値は `MYSQL` です。共有メモリ名では大文字と小文字が区別されます。

このオプションは、共有メモリ接続をサポートするために `shared_memory` システム変数を有効にしてサーバーを起動した場合にのみ適用されます。

- `--skip-sys-schema`

デフォルトでは、`sys` スキーマがインストールされていない場合は `mysql_upgrade` によってインストールされ、それ以外の場合は現在のバージョンにアップグレードされます。`--skip-sys-schema` オプションは、この動作を抑止します。

- `--socket=path, -S path`

`localhost` への接続用に使用する、Unix ソケットファイル、または Windows では使用する名前付きパイプの名前。

Windows では、このオプションは、名前付きパイプ接続をサポートするために `named_pipe` システム変数を有効にしてサーバーを起動した場合にのみ適用されます。また、接続を行うユーザーは、`named_pipe_full_access_group` システム変数で指定された Windows グループのメンバーである必要があります。

- `--ssl*`

`--ssl` で始まるオプションは、SSL を使用してサーバーに接続するかどうかを指定し、SSL 鍵および証明書を検索する場所を指定します。[暗号化接続のコマンドオプション](#)を参照してください。

- `--ssl-fips-mode={OFF|ON|STRICT}`

クライアント側で FIPS モードを有効にするかどうかを制御します。`--ssl-fips-mode` オプションは、暗号化された接続の確立には使用されず、許可する暗号化操作に影響する点で、他の `--ssl-xxx` オプションとは異なります。[セクション6.8「FIPS のサポート」](#)を参照してください。

次の `--ssl-fips-mode` 値を使用できます:

- `OFF`: FIPS モードを無効にします。
- `ON`: FIPS モードを有効にします。
- `STRICT`: 「strict」 FIPS モードを有効にします。

注記

OpenSSL FIPS オブジェクトモジュールが使用できない場合、`--ssl-fips-mode` に許可される値は `OFF` のみです。この場合、`--ssl-fips-mode` を `ON` または `STRICT` に設定すると、クライアントは起動時に警告を生成し、FIPS 以外のモードで動作します。

- `--tls-ciphersuites=ciphersuite_list`

TLSv1.3 を使用する暗号化された接続に許可される暗号スイート。値は、コロンで区切られた 1 つ以上の暗号スイート名のリストです。このオプションに指定できる暗号スイートは、MySQL のコンパイルに使用される SSL ラ

イブラリによって異なります。詳細は、[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#)を参照してください。

このオプションは MySQL 8.0.16 で追加されました。

- `--tls-version=protocol_list`

暗号化された接続に許可される TLS プロトコル。値は、1 つまたは複数のコンマ区切りプロトコル名のリストです。このオプションに指定できるプロトコルは、MySQL のコンパイルに使用される SSL ライブラリによって異なります。詳細は、[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#)を参照してください。

- `--upgrade-system-tables, -s`

mysql スキーマのシステムテーブルのみをアップグレードし、ユーザースキーマはアップグレードしません。

- `--user=user_name, -u user_name`

サーバーへの接続に使用する MySQL アカウントのユーザー名。デフォルトのユーザー名は `root` です。

- `--verbose`

冗長モード。プログラムの動作についてより多くの情報を出力します。

- `--version-check, -k`

`mysql_upgrade` の接続先のサーバーのバージョンをチェックして、`mysql_upgrade` がビルドされたバージョンと同じであることを確認します。そうでない場合は `mysql_upgrade` は終了します。このオプションはデフォルトで有効にされています。`--skip-version-check` を使用して無効化してください。

- `--write-binlog`

デフォルトでは、`mysql_upgrade` によるバイナリロギングは無効になっています。アクションをバイナリログに書き込む場合は、`--write-binlog` を使用してプログラムを起動します。

グローバルトランザクション識別子 (GTID) を有効にして (`gtid_mode=ON`) サーバーを実行している場合は、`mysql_upgrade` によるバイナリロギングを有効にしないでください。

- `--zstd-compression-level=level`

`zstd` 圧縮アルゴリズムを使用するサーバーへの接続に使用する圧縮レベル。許可されるレベルは 1 から 22 で、大きい値は圧縮レベルの増加を示します。デフォルトの `zstd` 圧縮レベルは 3 です。圧縮レベルの設定は、`zstd` 圧縮を使用しない接続には影響しません。

詳細は、[セクション4.2.8「接続圧縮制御」](#)を参照してください。

このオプションは MySQL 8.0.18 で追加されました。

4.5 クライアントプログラム

このセクションでは、MySQL サーバーに接続するクライアントプログラムについて説明します。

4.5.1 mysql — MySQL コマンドラインクライアント

`mysql` は、入力行の編集機能を備えた簡単な SQL シェルです。インタラクティブおよび非インタラクティブでの使用をサポートします。インタラクティブで使用した場合、クエリー結果は ASCII 表形式で提示されます。非インタラクティブ (フィルタとしてなど) で使用する場合は、結果はタブ区切り形式で表示されます。出力形式は、コマンドオプションを使用して変更できます。

大きな結果セット用のメモリーが足りないことで問題が発生する場合は、`--quick` オプションを使用します。これにより `mysql` は、全結果セットを表示前に取得してメモリー内でバッファリングする代わりに、サーバーから 1 行ずつ結果を取得することを強制されます。これは、`mysql_store_result()` ではなく、クライアント/サーバーライブラリ内の `mysql_use_result()` C API 関数を使用して結果セットを返すことで実行されます。

注記

または、MySQL Shell は X DevAPI へのアクセスを提供します。詳細は、[MySQL Shell 8.0](#)を参照してください。

mysql は非常に簡単に使用できます。次のように、コマンドインタプリタのプロンプトから起動してください。

```
shell> mysql db_name
```

または:

```
shell> mysql --user=user_name --password db_name  
Enter password: your_password
```

次に SQL ステートメントを入力し、`;`、`\g`、または `\G` で終わらせ Enter を押します。

Control+C を入力すると、現在のステートメントがある場合は中断され、それ以外の場合は一部の入力行が取り消されます。

SQL ステートメントは次のように、スクリプトファイル (バッチファイル) で実行できます。

```
shell> mysql db_name < script.sql > output.tab
```

Unix では、mysql クライアントはインタラクティブに実行されたステートメントを履歴ファイルにログ記録します。[セクション4.5.1.3「mysql クライアントロギング」](#)を参照してください。

4.5.1.1 mysql クライアントオプション

mysql は次のオプションをサポートします。これらはコマンド行またはオプションファイルの `[mysql]` グループおよび `[client]` グループで指定できます。MySQL プログラムによって使用されるオプションファイルの詳細については、[セクション4.2.2.2「オプションファイルの使用」](#)を参照してください。

表 4.11 「mysql クライアントオプション」

オプション名	説明	導入	非推奨
<code>--auto-rehash</code>	自動リハッシュを有効化		
<code>--auto-vertical-output</code>	結果セットの垂直形式での表示を有効化		
<code>--batch</code>	履歴ファイルを使用しない		
<code>--binary-as-hex</code>	16 進数表記でバイナリ値を表示		
<code>--binary-mode</code>	<code>\n</code> の <code>\n</code> への変換および <code>\0</code> をクエリーの終端として処理することを無効化		
<code>--bind-address</code>	指定されたネットワークインタフェースを使用して MySQL サーバーに接続		
<code>--character-sets-dir</code>	文字セットがインストールされているディレクトリ		
<code>--column-names</code>	結果にカラム名を記述		
<code>--column-type-info</code>	結果セットのメタデータを表示		
<code>--comments</code>	サーバーに送信されたステートメント内のコメントを保持するか削除するか		
<code>--compress</code>	クライアントとサーバー間で送信される情報をすべて圧縮		8.0.18

オプション名	説明	導入	非推奨
<code>--compression-algorithms</code>	サーバーへの接続に許可される圧縮アルゴリズム	8.0.18	
<code>--connect-expired-password</code>	クライアントが期限切れパスワードのサンドボックスモードを処理できることをサーバーに指示		
<code>--connect-timeout</code>	接続タイムアウトまでの秒数		
<code>--database</code>	使用されるべきデータベース		
<code>--debug</code>	デバッグログを書き込みます。MySQL がデバッグサポート付きで構築された場合にのみサポートされます		
<code>--debug-check</code>	プログラムの終了時にデバッグ情報を出力		
<code>--debug-info</code>	プログラムの終了時に、デバッグ情報、メモリー、および CPU の統計を出力		
<code>--default-auth</code>	使用する認証プラグイン		
<code>--default-character-set</code>	デフォルト文字セットを指定		
<code>--defaults-extra-file</code>	通常のオプションファイルに加えて、名前付きオプションファイルを読み取ります		
<code>--defaults-file</code>	指名されたオプションファイルのみを読み取る		
<code>--defaults-group-suffix</code>	オプショングループのサフィクス値		
<code>--delimiter</code>	ステートメント区切り文字を設定		
<code>--dns-srv-name</code>	ホスト情報のための DNS SRV 参照の使用	8.0.22	
<code>--enable-cleartext-plugin</code>	平文の認証プラグインを有効化		
<code>--execute</code>	ステートメントを実行して終了		
<code>--force</code>	SQL エラーが発生しても続行		
<code>--get-server-public-key</code>	サーバーから RSA 公開キーをリクエスト		
<code>--help</code>	ヘルプメッセージを表示して終了		
<code>--histignore</code>	ロギングに関してどのステートメントを無視するかを指定するパターン		
<code>--host</code>	MySQL サーバーがあるホスト		

オプション名	説明	導入	非推奨
--html	HTML 出力を生成します		
--ignore-spaces	関数名のあとのスペースを無視		
--init-command	接続後に実行する SQL ステートメント		
--line-numbers	エラーの行番号を書き込み		
--load-data-local-dir	LOAD DATA LOCAL ステートメントで指定されたファイルのディレクトリ	8.0.21	
--local-infile	LOAD DATA の LOCAL 機能の有効化または無効化		
--login-path	ログインパスオプションを .mylogin.cnf から読み取り		
--max-allowed-packet	サーバーとの間で送受信するパケットの最大長		
--max-join-size	--safe-updates を使用する結合で、自動的に設定される行の制限		
--named-commands	名前付き mysql コマンドを有効化		
--net-buffer-length	TCP/IP とソケット通信のバッファサイズ		
--network-namespace	ネットワークネームスペースの指定	8.0.22	
--no-auto-rehash	自動リハッシュを無効化		
--no-beep	エラー時に音を発生させない		
--no-defaults	オプションファイルを読み取らない		
--one-database	コマンド行で指定されたデフォルトデータベースに対するステートメント以外を無視		
--pager	クエリー出力のページングに指定されたコマンドを使用		
--password	サーバーに接続する際に使用するパスワード		
--pipe	名前付きパイプを使用してサーバに接続する (Windows のみ)		
--plugin-dir	プラグインがインストールされているディレクトリ		
--port	接続用の TCP/IP ポート番号		
--print-defaults	デフォルトオプションの印刷		

オプション名	説明	導入	非推奨
<code>--prompt</code>	プロンプトを指定された形式に設定		
<code>--protocol</code>	使用するトランスポートプロトコル		
<code>--quick</code>	各クエリーの結果をキャッシュしない		
<code>--raw</code>	カラム値をエスケープの変換なしで書き込み		
<code>--reconnect</code>	サーバーとの接続が失われたとき、再接続を自動的に試行		
<code>--safe-updates, --i-am-a-dummy</code>	キー値を指定する UPDATE ステートメントおよび DELETE ステートメントのみを許可		
<code>--select-limit</code>	<code>--safe-updates</code> 使用時に自動的に設定される SELECT ステートメントの制限		
<code>--server-public-key-path</code>	RSA 公開鍵を含むファイルへのパス名		
<code>--shared-memory-base-name</code>	共有メモリー接続用の共有メモリー名 (Windows のみ)		
<code>--show-warnings</code>	各ステートメントのあとに警告があれば表示		
<code>--sigint-ignore</code>	SIGINT 信号を無視 (通常、Control+C を入力した結果)		
<code>--silent</code>	サイレントモード		
<code>--skip-auto-rehash</code>	自動リハッシュを無効化		
<code>--skip-column-names</code>	結果にカラム名を記述しない		
<code>--skip-line-numbers</code>	エラーの行番号を省略		
<code>--skip-named-commands</code>	名前付き mysql コマンドを無効化		
<code>--skip-pager</code>	ページングを無効化		
<code>--skip-reconnect</code>	再接続を無効化		
<code>--socket</code>	使用する Unix ソケットファイルまたは Windows 名前付きパイプ		
<code>--ssl-ca</code>	信頼できる SSL 認証局のリストを含むファイル		
<code>--ssl-capath</code>	信頼できる SSL 認証局の証明書ファイルを含むディレクトリ		
<code>--ssl-cert</code>	X.509 証明書を含むファイル		
<code>--ssl-cipher</code>	接続の暗号化に許可される暗号		

オプション名	説明	導入	非推奨
<code>--ssl-crl</code>	証明書失効リストを含むファイル		
<code>--ssl-crlpath</code>	証明書失効リストファイルを含むディレクトリ		
<code>--ssl-fips-mode</code>	クライアント側で FIPS モードを有効にするかどうか		
<code>--ssl-key</code>	X.509 キーを含むファイル		
<code>--ssl-mode</code>	サーバーへの接続に必要なセキュリティ状態		
<code>--syslog</code>	インタラクティブなステートメントをシステムログに記録		
<code>--table</code>	出力を表形式で表示		
<code>--tee</code>	出力のコピーを指定されたファイルに追加		
<code>--tls-ciphersuites</code>	暗号化された接続に許可される TLSv1.3 暗号スイート	8.0.16	
<code>--tls-version</code>	暗号化された接続に許可される TLS プロトコル		
<code>--unbuffered</code>	各クエリーのあとでバッファをフラッシュ		
<code>--user</code>	サーバーへの接続時に使用する MySQL ユーザー名		
<code>--verbose</code>	冗長モード		
<code>--version</code>	バージョン情報を表示して終了		
<code>--vertical</code>	クエリー出力行を垂直形式で出力 (カラム値ごとに 1 行)		
<code>--wait</code>	接続が確立できない場合、中止せずに待機してからリトライ		
<code>--xml</code>	XML 出力を生成		
<code>--zstd-compression-level</code>	zstd 圧縮を使用するサーバーへの接続の圧縮レベル	8.0.18	

- `--help, -?`

ヘルプメッセージを表示して終了します。

- `--auto-rehash`

自動リハッシュを有効化 このオプションはデフォルトでオンになっており、データベース、テーブル、およびカラムの名前補完が可能になります。リハッシュを無効にするには、`--disable-auto-rehash` を使用します。これによ

り、`mysql` の起動がより高速になりますが、名前補完を使用する場合は `rehash` コマンドまたはそのショートカット `#` を発行する必要があります。

名前を補完するには、最初の部分を入力して Tab を押します。名前があいまいでない場合、`mysql` がその名前を補完します。そうでない場合は、Tab をもう一度押して、これまでに入力した値で始まる、考えられる名前を表示できます。デフォルトのデータベースがない場合、補完は行われません。

注記

この機能では、`readline` ライブラリでコンパイルされた MySQL クライアントが必要で
す。通常、`readline` ライブラリは Windows では使用できません。

• `--auto-vertical-output`

結果セットが現在のウィンドウに対して大きすぎる場合には縦に表示され、そうでない場合は通常の表形式が使用されるようになります。(これは、`;` または `\G` で終了するステートメントに適用されます。)

• `--batch, -B`

カラム区切り文字としてタブを使用し、各行を新しい行に出力します。このオプションでは、`mysql` は履歴ファイルを使用しません。

バッチモードでは、表形式でない出力が生成され、特殊文字のエスケープ処理が行われます。raw モードを使用すれば、エスケープ処理を無効にできます。`--raw` オプションの説明を参照してください。

• `--binary-as-hex`

このオプションを指定すると、`mysql` は 16 進数表記 (`0xvalue`) を使用してバイナリデータを表示します。これは、全体的な出力表示形式が表形式、垂直形式、HTML 形式または XML 形式のいずれであるかに関係なく発生します。

`--binary-as-hex` を有効にすると、`CHAR()` や `UNHEX()` などの関数によって返されるものも含め、すべてのバイナリ文字列の表示に影響します。次の例では、`A` の ASCII コード (65 decimal、41 hexadecimal) を使用してこれを後退させます:

• `--binary-as-hex` 無効:

```
mysql> SELECT CHAR(0x41), UNHEX('41');
+-----+-----+
| CHAR(0x41) | UNHEX('41') |
+-----+-----+
| A         |              |
+-----+-----+
```

• `--binary-as-hex` 有効:

```
mysql> SELECT CHAR(0x41), UNHEX('41');
+-----+-----+
| CHAR(0x41) | UNHEX('41') |
+-----+-----+
| 0x41      | 0x41        |
+-----+-----+
```

`--binary-as-hex` が有効かどうかに関係なく文字列として表示されるようにバイナリ文字列を記述するには、次の手法を使用します:

• `CHAR()` 関数には、次の `USING charset` 句があります:

```
mysql> SELECT CHAR(0x41 USING utf8mb4);
+-----+
| CHAR(0x41 USING utf8mb4) |
+-----+
| A                          |
+-----+
```

• より一般的には、`CONVERT()` を使用して式を特定の文字セットに変換します:

```
mysql> SELECT CONVERT(UNHEX('41') USING utf8mb4);
```

```
+-----+
| CONVERT(UNHEX('41') USING utf8mb4) |
+-----+
| A |
+-----+
```

MySQL 8.0.19 では、`mysql` が対話モードで動作している場合、このオプションはデフォルトで有効になっています。また、オプションが暗黙的または明示的に有効になっている場合、`status` (または `\s`) コマンドからの出力には次の行が含まれます:

```
Binary data as: Hexadecimal
```

16 進表記法を無効にするには、`--skip-binary-as-hex` を使用

- `--binary-mode`

このオプションは、`BLOB` 値を含む可能性のある `mysqlbinlog` の出力を処理する場合に便利です。`mysql` は、デフォルトではステートメント文字列内の `\r\n` を `\n` に変換し、`\0` をステートメント終端記号と解釈します。`--binary-mode` ではこの両方の機能が無効になります。また、`charset` および `delimiter` を除くすべての `mysql` コマンドを非対話モードで無効にします (`mysql` にパイプされた入力または `source` コマンドを使用してロードされた入力の場合)。

- `--bind-address=ip_address`

複数のネットワークインタフェースを持つコンピュータで、このオプションを使用して、MySQL サーバーへの接続に使用するインタフェースを選択します。

- `--character-sets-dir=dir_name`

文字セットがインストールされているディレクトリ。 [セクション10.15「文字セットの構成」](#) を参照してください。

- `--column-names`

結果にカラム名を記述します。

- `--column-type-info`

結果セットのメタデータを表示します。この情報は、C API `MYSQL_FIELD` データ構造体の内容に対応します。 [C API Basic Data Structures](#) を参照してください。

- `--comments, -c`

サーバーに送信されるステートメント内のコメントを削除するか保持するか。デフォルトは `--skip-comments` (コメントの削除)、`--comments` で有効化 (コメントを保持) です。

注記

このオプションが指定されているかどうかに関係なく、`mysql` クライアントは常にオプションタイムアウトをサーバーに渡します。

コメントの削除は非推奨です。この機能と、将来の MySQL リリースで削除されることを制御するオプションが必要です。

- `--compress, -C`

可能であれば、クライアントとサーバーの間で送信されるすべての情報を圧縮します。 [セクション4.2.8「接続圧縮制御」](#) を参照してください。

MySQL 8.0.18 では、このオプションは非推奨です。将来のバージョンの MySQL で削除される予定です。 [レガシー接続圧縮の構成](#) を参照してください。

- `--compression-algorithms=value`

サーバーへの接続に許可される圧縮アルゴリズム。使用可能なアルゴリズムは、`protocol_compression_algorithms` システム変数の場合と同じです。デフォルト値は `uncompressed` です。

詳細は、[セクション4.2.8「接続圧縮制御」](#)を参照してください。

このオプションは MySQL 8.0.18 で追加されました。

- `--connect-expired-password`

接続に使用されるアカウントに期限切れのパスワードがある場合、クライアントがサンドボックスモードを処理できることをサーバーに示します。通常サーバーは、パスワードの期限の切れたアカウントを使用して接続を試みる非インタラクティブなクライアントを切断するため、これは非インタラクティブに `mysql` を起動する場合に便利です。([セクション6.2.16「期限切れパスワードのサーバー処理」](#)を参照してください。)

- `--connect-timeout=value`

接続タイムアウトまでの秒数。(デフォルト値は 0 です。)

- `--database=db_name, -D db_name`

使用するデータベース。これは主に、オプションファイルで便利です。

- `--debug=[debug_options], -# [debug_options]`

デバッグのログを書き込みます。一般的な `debug_options` 文字列は `d:t:o,file_name` です。デフォルトは `d:t:o/tmp/mysql.trace` です。

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `--debug-check`

プログラムの終了時に、デバッグ情報を出力します。

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `--debug-info, -T`

プログラムの終了時に、デバッグ情報とメモリーおよび CPU 使用率の統計を出力します。

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `--default-auth=plugin`

使用するクライアント側認証プラグインに関するヒント。 [セクション6.2.17「プラグブル認証」](#)を参照してください。

- `--default-character-set=charset_name`

クライアントおよび接続で、`charset_name` をデフォルト文字セットとして使用します。

このオプションは、オペレーティングシステムがある文字セットを使用し、`mysql` クライアントがデフォルトで別の文字セットを使用する場合に便利です。この場合、出力が正しくフォーマットされない可能性があります。このような問題は通常、このオプションを使用して、クライアントが代わりにシステムの文字セットを使用するように強制することで解決できます。

詳細は、[セクション10.4「接続文字セットおよび照合順序」](#)および[セクション10.15「文字セットの構成」](#)を参照してください。

- `--defaults-extra-file=file_name`

このオプションファイルは、グローバルオプションファイルのあとに読み取りますが、(UNIX では) ユーザーオプションファイルの前に読み取るようにしてください。ファイルが存在しないかアクセスできない場合、エラーが発生します。`file_name` は、フルパス名でなく相対パス名として指定された場合、現行ディレクトリを基準にして解釈されます。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--defaults-file=file_name`

指定されたオプションファイルのみ使用します。ファイルが存在しないかアクセスできない場合、エラーが発生します。`file_name` は、フルパス名でなく相対パス名として指定された場合、現行ディレクトリを基準にして解釈されます。

例外: `--defaults-file` でも、クライアントプログラムは `.mylogin.cnf` を読み取ります。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--defaults-group-suffix=str`

通常のオプショングループだけでなく、通常の名前に `str` のサフィクスが付いたグループも読み取ります。たとえば、`mysql` は通常 `[client]` グループおよび `[mysql]` グループを読み取ります。`--defaults-group-suffix=_other` オプションを指定した場合、`mysql` は `[client_other]` グループおよび `[mysql_other]` グループも読み取ります。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--delimiter=str`

ステートメント区切り文字を設定します。デフォルトはセミコロン (;) 文字です。

- `--disable-named-commands`

名前付きコマンドを無効化します。`*` 形式のみを使用します。または、セミコロン (;) で終わる行の先頭でのみ名前付きコマンドを使用します。`mysql` の起動時に、このオプションはデフォルトで有効になっています。ただし、このオプションを使用しても、ロング形式コマンドは最初の行から効果を発揮します。[セクション4.5.1.2「mysqlクライアントコマンド」](#)を参照してください。

- `--dns-srv-name=name`

MySQL サーバーへの接続の確立に使用する候補ホストを決定する DNS SRV レコードの名前を指定します。MySQL での DNS SRV サポートの詳細は、[セクション4.2.6「DNS SRV レコードを使用したサーバーへの接続」](#)を参照してください。

DNS が `example.com` ドメインの SRV 情報で構成されているとします:

Name	TTL	Class	Priority	Weight	Port	Target
<code>_mysql._tcp.example.com.</code>	86400	IN	SRV 0	5	3306	host1.example.com
<code>_mysql._tcp.example.com.</code>	86400	IN	SRV 0	10	3306	host2.example.com
<code>_mysql._tcp.example.com.</code>	86400	IN	SRV 10	5	3306	host3.example.com
<code>_mysql._tcp.example.com.</code>	86400	IN	SRV 20	5	3306	host4.example.com

その DNS SRV レコードを使用するには、次のように `mysql` を起動します:

```
mysql --dns-srv-name=_mysql._tcp.example.com
```

`mysql` は、接続が正常に確立されるまで、グループ内の各サーバーへの接続を試行します。接続の失敗は、サーバーへの接続を確立できない場合にのみ発生します。DNS SRV レコードの優先度と重みの値によって、サーバーの試行順序が決まります。

`--dns-srv-name` を使用して起動された場合、`mysql` は TCP 接続の確立のみを試みます。

`--dns-srv-name` オプションは、両方が指定されている場合、`--host` オプションよりも優先されます。`--dns-srv-name` では、接続確立で `mysql_real_connect()` ではなく `mysql_real_connect_dns_srv()` C API 関数が使用されます。ただし、後で `connect` コマンドが実行時に使用され、ホスト名引数が指定された場合、そのホスト名は `mysql` の起動時に指定された `--dns-srv-name` オプションよりも優先され、DNS SRV レコードが指定されます。

このオプションは MySQL 8.0.22 で追加されました。

- `--enable-cleartext-plugin`

`mysql_clear_password` 平文認証プラグインを有効にします。(セクション6.4.1.4「クライアント側クリアテキストプラグブル認証」を参照してください。)

- `--execute=statement, -e statement`

ステートメントを実行して、終了します。デフォルトの出力形式は、`--batch` で生成されるものと同様です。例については、セクション4.2.2.1「コマンド行でのオプションの使用」を参照してください。このオプションでは、`mysql` は履歴ファイルを使用しません。

- `--force, -f`

SQL エラーが発生しても続行します。

- `--get-server-public-key`

RSA キーペアベースのパスワード交換に必要な公開キーをサーバーにリクエストします。このオプションは、`caching_sha2_password` 認証プラグインで認証されるクライアントに適用されます。そのプラグインの場合、サーバーは要求されないかぎり公開鍵を送信しません。このオプションは、そのプラグインで認証されないアカウントでは無視されます。クライアントがセキュアな接続を使用してサーバーに接続する場合と同様に、RSA ベースのパスワード交換を使用しない場合も無視されます。

`--server-public-key-path=file_name` が指定され、有効な公開キーファイルが指定されている場合は、`--get-server-public-key` よりも優先されます。

`caching_sha2_password` プラグインの詳細は、セクション6.4.1.2「SHA-2 プラグブル認証のキャッシュ」を参照してください。

- `--histignore`

ロギングのために無視するステートメントを指定するコロンで区切られたパターンのリスト。これらのパターンはデフォルトのパターンリスト ("`*IDENTIFIED*:PASSWORD*`") に追加されます。このオプションに指定した値は、履歴ファイルに書き込まれるステートメントのロギング、および `--syslog` オプションが指定されている場合は `syslog` に影響します。詳細は、セクション4.5.1.3「mysql クライアントロギング」を参照してください。

- `--host=host_name, -h host_name`

指定されたホストの MySQL サーバーに接続します。

`--dns-srv-name` オプションは、両方が指定されている場合、`--host` オプションよりも優先されます。`--dns-srv-name` では、接続確立で `mysql_real_connect()` ではなく `mysql_real_connect_dns_srv()` C API 関数が使用されます。ただし、後で `connect` コマンドが実行時に使用され、ホスト名引数が指定された場合、そのホスト名は `mysql` の起動時に指定された `--dns-srv-name` オプションよりも優先され、DNS SRV レコードが指定されます。

- `--html, -H`

HTML 出力を生成します。

- `--ignore-spaces, -i`

関数名の後ろのスペースを無視します。この効果は、`IGNORE_SPACE` SQL モード(セクション5.1.11「サーバー SQL モード」を参照してください)に関する議論で説明されています。

- `--init-command=str`

サーバーへの接続後に実行する SQL ステートメント。 `auto-reconnect` が有効の場合、ステートメントは再接続が生じたあとに再度実行されます。

- `--line-numbers`

エラーの行番号を書き込みます。 `--skip-line-numbers` で無効にできます。

- `--load-data-local-dir=dir_name`

このオプションは、[LOAD DATA](#) 操作のクライアント側 [LOCAL](#) 機能に影響します。[LOAD DATA LOCAL](#) ステートメントで指定されたファイルを配置する必要があるディレクトリを指定します。`--load-data-local-dir` の効果は、[LOCAL](#) データロードが有効か無効かによって異なります:

- [LOCAL](#) データロードが MySQL クライアントライブラリでデフォルトで有効になっている場合、または `--local-infile[=1]` を指定して有効になっている場合、`--load-data-local-dir` オプションは無視されます。
- [LOCAL](#) のデータロードが無効になっている場合は、MySQL クライアントライブラリでデフォルトで無効になっているか、`--local-infile=0` を指定すると、`--load-data-local-dir` オプションが適用されます。

`--load-data-local-dir` が適用される場合、オプション値は、ローカルデータファイルを配置する必要があるディレクトリを指定します。ディレクトリパス名とロードされるファイルのパス名の比較では、基礎となるファイルシステムの大/小文字の区別に関係なく、大/小文字が区別されます。オプション値が空の文字列の場合、ディレクトリ名は指定されず、ローカルデータのロードにファイルは許可されません。

たとえば、`/my/local/data` ディレクトリにあるファイル以外のローカルデータロードを明示的に無効にするには、次のように `mysql` を起動します:

```
mysql --local-infile=0 --load-data-local-dir=/my/local/data
```

`--local-infile` と `--load-data-local-dir` の両方が指定されている場合、それらの指定順序は関係ありません。

`mysql` で [LOCAL](#) ロード操作を正常に使用するには、サーバーでローカルロードが許可されている必要もあります。[セクション6.1.6「LOAD DATA LOCAL のセキュリティ上の考慮事項」](#) を参照してください

`--load-data-local-dir` オプションが MySQL 8.0.21 に追加されました。

- `--local-infile[={0|1}]`

デフォルトでは、[LOAD DATA](#) の [LOCAL](#) 機能は、MySQL クライアントライブラリにコンパイルされたデフォルトで決定されます。[LOCAL](#) データロードを明示的に有効または無効にするには、`--local-infile` オプションを使用します。値を指定しない場合、このオプションは [LOCAL](#) データロードを有効にします。`--local-infile=0` または `--local-infile=1` として指定した場合、このオプションは [LOCAL](#) データロードを無効または有効にします。

[LOCAL](#) 機能が無効になっている場合は、`--load-data-local-dir` オプションを使用して、指定されたディレクトリにあるファイルの制限付きローカルロードを許可できます。

`mysql` で [LOCAL](#) ロード操作を正常に使用するには、サーバーでローカルロードが許可されている必要もあります。[セクション6.1.6「LOAD DATA LOCAL のセキュリティ上の考慮事項」](#) を参照してください

- `--login-path=name`

`.mylogin.cnf` ログインパスファイルの指定されたログインパスからオプションを読み取ります。「`「ログインパス」`」は、接続先の MySQL サーバーおよび認証に使用するアカウントを指定するオプションを含むオプショングループです。ログインパスファイルを作成または変更するには、`mysql_config_editor` ユーティリティを使用します。[セクション4.6.7「mysql_config_editor — MySQL 構成ユーティリティ」](#) を参照してください。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--max-allowed-packet=value`

クライアント/サーバー通信のバッファの最大サイズ。デフォルトは 16M バイト、最大は 1G バイトです。

- `--max-join-size=value`

`--safe-updates` 使用時の、自動的に設定される結合内の行の制限。(デフォルト値は 1,000,000 です。)

- `--named-commands, -G`

名前付き `mysql` コマンドを有効にします。ショート形式のコマンドのみではなく、ロング形式のコマンドが許可されます。たとえば、`quit` と `\q` は両方認識されます。名前付きコマンドを無効化するには、`--skip-named-commands` を使用します。[セクション4.5.1.2「mysql クライアントコマンド」](#) を参照してください。

- `--net-buffer-length=value`

TCP/IP とソケット通信のバッファサイズ。(デフォルト値は 16K バイトです。)

- `--network-namespace=name`

TCP/IP 接続に使用するネットワークネームスペース。省略すると、接続ではデフォルト (グローバル) のネームスペースが使用されます。ネットワークネームスペースの詳細は、[セクション5.1.14「ネットワークネームスペースのサポート」](#) を参照してください。

このオプションは MySQL 8.0.22 で追加されました。ネットワークネームスペースのサポートを実装するプラットフォームでのみ使用できます。

- `--no-auto-rehash, -A`

これは `--skip-auto-rehash` と同様の効果を持ちます。 `--auto-rehash` の説明を参照してください。

- `--no-beep, -b`

エラー時に音を発生させません。

- `--no-defaults`

オプションファイルを読み取りません。オプションファイルから不明のオプションを読み取ることが原因でプログラムの起動に失敗する場合、`--no-defaults` を使用して、オプションを読み取らないようにできます。

例外として、`.mylogin.cnf` ファイルは、存在する場合はすべての場合に読み取られます。これにより、`--no-defaults` が使用されたとしても、コマンド行よりも安全な方法でパスワードを指定できます。(`.mylogin.cnf` は `mysql_config_editor` ユーティリティーによって作成されます。 [セクション4.6.7「mysql_config_editor — MySQL 構成ユーティリティー」](#) を参照してください)。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--one-database, -o`

デフォルトデータベースがコマンド行で指定されたものである間に生じるステートメント以外を無視します。このオプションは初歩的なものであり、注意して使用してください。ステートメントのフィルタリングは `USE` ステートメントのみに基づいています。

最初、`mysql` は入力内のステートメントを実行します。これは、データベース `db_name` をコマンド行で指定することは `USE db_name` を入力の最初に挿入することと同等であるためです。次に、`USE` ステートメントを検出するたびに、`mysql` は、指名されたデータベースがコマンド行のものであるかによって、後続のステートメントを受け入れまたは拒否します。ステートメントの内容は重要ではありません。

`mysql` が次の一連のステートメントを処理するために起動されたとします。

```
DELETE FROM db2.t2;
USE db2;
DROP TABLE db1.t1;
CREATE TABLE db1.t1 (i INT);
USE db1;
INSERT INTO t1 (i) VALUES(1);
CREATE TABLE db2.t1 (j INT);
```

コマンド行が `mysql --force --one-database db1` の場合、`mysql` は入力を次のように処理します。

- `DELETE` ステートメントでは別のデータベースのテーブルが指名されていますが、デフォルトデータベースは `db1` であるため、このステートメントは実行されます。
- `DROP TABLE` ステートメントおよび `CREATE TABLE` ステートメントでは `db1` のテーブルを指名していますが、デフォルトデータベースが `db1` ではないため、これらのステートメントは実行されません。
- `CREATE TABLE` ステートメントでは別のデータベースのテーブルが指名されていますが、デフォルトデータベースは `db1` であるため、`INSERT` ステートメントおよび `CREATE TABLE` ステートメントは実行されます。

- `--pager[=command]`

クエリー出力のページングに、指定されたコマンドを使用します。このコマンドが省略された場合、デフォルトのページャーは `PAGER` 環境変数の値となります。有効なページャーは、`less`、`more`、`cat [> filename]`、などです。このオプションは Unix でインタラクティブモードの場合のみ機能します。ページングを無効化するには、`--skip-pager` を使用してください。[セクション4.5.1.2「mysql クライアントコマンド」](#)には出力のページングの詳細説明があります。

- `--password[=password]`, `-p[password]`

サーバーへの接続に使用される MySQL アカウントのパスワード。パスワード値はオプションです。指定しない場合、`mysql` によってプロンプトが表示されます。指定する場合は、`--password=` または `-p` とそれに続くパスワードの間にスペースなしが存在する必要があります。パスワードオプションを指定しない場合、デフォルトではパスワードは送信されません。

コマンド行でのパスワード指定は、セキュアでないと考えべきです。コマンド行でパスワードを指定しないようにするには、オプションファイルを使用します。[セクション6.1.2.1「パスワードセキュリティのためのエンドユーザーガイドライン」](#)を参照してください。

パスワードがなく、`mysql` でパスワードの入力を求められないように明示的に指定するには、`--skip-password` オプションを使用します。

- `--pipe`, `-W`

Windows で、名前付きパイプを使用してサーバーに接続します。このオプションは、ネームパイプ接続をサポートするために `named_pipe` システム変数を有効にしてサーバーを起動した場合にのみ適用されます。また、接続を行うユーザーは、`named_pipe_full_access_group` システム変数で指定された Windows グループのメンバーである必要があります。

- `--plugin-dir=dir_name`

プラグインを検索するディレクトリ。このオプションは、`--default-auth` オプションを使用して認証プラグインを指定しても、`mysql` がそれを検出しない場合に指定します。[セクション6.2.17「プラグイン認証」](#)を参照してください。

- `--port=port_num`, `-P port_num`

TCP/IP 接続の場合、使用するポート番号。

- `--print-defaults`

プログラム名と、オプションファイルから受け取るすべてのオプションを出力します。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--prompt=format_str`

プロンプトを指定された形式に設定します。デフォルトは `mysql>` です。プロンプト内に含めることができる特別なシーケンスは、[セクション4.5.1.2「mysql クライアントコマンド」](#)で説明されています。

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

サーバーへの接続に使用するトランスポートプロトコル。これは、他の接続パラメータが通常、必要なプロトコル以外のプロトコルを使用する場合に便利です。許可される値の詳細は、[セクション4.2.7「接続トランスポートプロトコル」](#)を参照してください。

- `--quick`, `-q`

各クエリー結果をキャッシュせず、各行を受信ししだい出力します。出力が遅延された場合、サーバーの速度が低下することがあります。このオプションでは、`mysql` は履歴ファイルを使用しません。

- `--raw, -r`

表形式の出力では、カラムを「枠で囲む」ことにより、1つのカラム値を別のカラム値と区別できます。表形式でない出力(バッチモードで生成される場合や `--batch` または `--silent` オプションを指定した場合など)では、簡単に識別できるように特殊文字が出力時にエスケープされます。改行、タブ、NUL、およびバックスラッシュはそれぞれ、`\n`、`\t`、`\0`、および `\\` と記述されます。`--raw` オプションを指定すると、この文字のエスケープ処理は無効になります。

次の例は、表形式と表形式でない出力の違い、および raw モードを使用してエスケープ処理を無効にした場合を示しています。

```
% mysql
mysql> SELECT CHAR(92);
+-----+
| CHAR(92) |
+-----+
|\ |
+-----+

% mysql -s
mysql> SELECT CHAR(92);
CHAR(92)
\\

% mysql -s -r
mysql> SELECT CHAR(92);
CHAR(92)
\
```

- `--reconnect`

サーバーとの接続が失われたとき、再接続を自動的に試行します。接続が失われるたびに一度再接続が試みられます。再接続動作を抑制するには、`--skip-reconnect` を使用します。

- `--safe-updates, --i-am-a-dummy, -U`

このオプションを有効にすると、`WHERE` 句または `LIMIT` 句でキーを使用しない `UPDATE` および `DELETE` ステートメントでエラーが発生します。また、大規模な結果セットを生成する(または生成すると見積もられる) `SELECT` ステートメントには制限があります。オプションファイルでこのオプションを設定した場合は、コマンドラインで `--skip-safe-updates` を使用してオーバーライドできます。このオプションについての詳細は、[セーフ更新モードの使用 \(--safe-updates\)](#) を参照してください。

- `--select-limit=value`

`--safe-updates` 使用時の、自動的に設定される `SELECT` ステートメントの制限。(デフォルト値は 1,000 です。)

- `--server-public-key-path=file_name`

RSA キーベースのパスワード交換のためにサーバーが必要とする公開キーのクライアント側コピーを含む、PEM 形式のファイルへのパス名。このオプションは、`sha256_password` または `caching_sha2_password` 認証プラグインで認証されるクライアントに適用されます。これらのプラグインのいずれかで認証されないアカウントでは、このオプションは無視されます。クライアントがセキュアな接続を使用してサーバーに接続する場合と同様に、RSA ベースのパスワード交換を使用しない場合も無視されます。

`--server-public-key-path=file_name` が指定され、有効な公開キーファイルが指定されている場合は、`--get-server-public-key` よりも優先されます。

このオプションは、MySQL が OpenSSL を使用してビルドされている場合のみ利用できます。

`sha256_password` および `caching_sha2_password` プラグインの詳細は、[セクション6.4.1.3「SHA-256 プラガブル認証」](#) および [セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#) を参照してください。

- `--shared-memory-base-name=name`

Windows の場合、共有メモリを使用してローカルサーバに接続するために使用する共有メモリ名。デフォルト値は `MYSQL` です。共有メモリー名では大文字と小文字が区別されます。

このオプションは、共有メモリー接続をサポートするために `shared_memory` システム変数を有効にしてサーバを起動した場合にのみ適用されます。

- `--show-warnings`

警告が存在する場合、各ステートメント後に表示させます。このオプションはインタラクティブとバッチモードにのみ対応しています。

- `--sigint-ignore`

`SIGINT` 信号を無視します (通常、Control+C を入力した結果)。

このオプションを指定せずに Control+C を入力すると、現在のステートメントがある場合は中断され、それ以外の場合は一部の入力行が取り消されます。

- `--silent, -s`

サイレントモード。出力の生成を少なくします。このオプションを複数回指定して、出力の生成をさらに少なくできます。

このオプションでは、表形式でない出力が生成され、特殊文字のエスケープ処理が行われます。raw モードを使用すれば、エスケープ処理を無効にできます。`--raw` オプションの説明を参照してください。

- `--skip-column-names, -N`

結果にカラム名を記述しません。

- `--skip-line-numbers, -L`

エラーの行番号を書き込みません。エラーメッセージを含む結果ファイルと比較する場合に便利です。

- `--socket=path, -S path`

`localhost` への接続用に使用する、Unix ソケットファイル、または Windows では使用する名前付きパイプの名前。

Windows では、このオプションは、名前付きパイプ接続をサポートするために `named_pipe` システム変数を有効にしてサーバを起動した場合にのみ適用されます。また、接続を行うユーザーは、`named_pipe_full_access_group` システム変数で指定された Windows グループのメンバーである必要があります。

- `--ssl*`

`--ssl` で始まるオプションは、SSL を使用してサーバに接続するかどうかを指定し、SSL 鍵および証明書を検索する場所を指定します。[暗号化接続のコマンドオプション](#)を参照してください。

- `--ssl-fips-mode={OFF|ON|STRICT}`

クライアント側で FIPS モードを有効にするかどうかを制御します。 `--ssl-fips-mode` オプションは、暗号化された接続の確立には使用されず、許可する暗号化操作に影響する点で、他の `--ssl-xxx` オプションとは異なります。 [セクション6.8「FIPS のサポート」](#) を参照してください。

次の `--ssl-fips-mode` 値を使用できます:

- **OFF**: FIPS モードを無効にします。
- **ON**: FIPS モードを有効にします。
- **STRICT**: 「strict」 FIPS モードを有効にします。

注記

OpenSSL FIPS オブジェクトモジュールが使用できない場合、 `--ssl-fips-mode` に許可される値は **OFF** のみです。 この場合、 `--ssl-fips-mode` を **ON** または **STRICT** に設定すると、クライアントは起動時に警告を生成し、FIPS 以外のモードで動作します。

- `--syslog, -j`

このオプションにより、 `mysql` は対話型のステートメントをシステムロギング機能に送信します。 Unix の場合は `syslog` で、Windows の場合は Windows イベントログです。 ログに記録されたメッセージが表示される宛先は、システムによって異なります。 Linux では、多くの場合、宛先は `/var/log/messages` ファイルです。

次に、 `--syslog` を使用して Linux で生成される出力のサンプルを示します。 この出力は読みやすくするためにフォーマットされており、ログに記録される各メッセージは実際には単一行を取ります。

```
Mar 7 12:39:25 myhost MysqlClient[20824]:  
SYSTEM_USER:'oscar', MYSQL_USER:'my_oscar', CONNECTION_ID:23,  
DB_SERVER:'127.0.0.1', DB:'-', QUERY:'USE test;'  
Mar 7 12:39:28 myhost MysqlClient[20824]:  
SYSTEM_USER:'oscar', MYSQL_USER:'my_oscar', CONNECTION_ID:23,  
DB_SERVER:'127.0.0.1', DB:'test', QUERY:'SHOW TABLES;'
```

詳細は、 [セクション4.5.1.3「mysql クライアントロギング」](#) を参照してください。

- `--table, -t`

出力を表形式で表示します。 インタラクティブに使用する場合はこれがデフォルトですが、バッチモードで表形式の出力を生成するのにも使用できます。

- `--tee=file_name`

出力のコピーを指定されたファイルに追加します。 このオプションはインタラクティブモードの場合のみ機能します。 [セクション4.5.1.2「mysql クライアントコマンド」](#) で、tee ファイルについて詳細に説明しています。

- `--tls-ciphersuites=ciphersuite_list`

TLSv1.3 を使用する暗号化された接続に許可される暗号スイート。 値は、コロンで区切られた 1 つ以上の暗号スイート名のリストです。 このオプションに指定できる暗号スイートは、MySQL のコンパイルに使用される SSL ライブラリによって異なります。 詳細は、 [セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#) を参照してください。

このオプションは MySQL 8.0.16 で追加されました。

- `--tls-version=protocol_list`

暗号化された接続に許可される TLS プロトコル。 値は、1 つまたは複数のコンマ区切りプロトコル名のリストです。 このオプションに指定できるプロトコルは、MySQL のコンパイルに使用される SSL ライブラリによって異なります。 詳細は、 [セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#) を参照してください。

- `--unbuffered, -n`

各クエリー後にバッファをフラッシュします。

- `--user=user_name, -u user_name`

サーバーへの接続に使用する MySQL アカウントのユーザー名。

- `--verbose, -v`

冗長モード。プログラムの動作についてより多くの出力を生成します。このオプションを複数回指定して、さらに多くの出力を生成できます。(たとえば、`-v -v -v` ではバッチモードでも表形式の出力を生成します。)

- `--version, -V`

バージョン情報を表示して終了します。

- `--vertical, -E`

クエリー出力行を縦に出力します(カラム値ごとに一行)。このオプションを使用しない場合、個々のステートメントを `\G` で終了させることにより、縦の出力を指定できます。

- `--wait, -w`

接続が確立できない場合、中止せずに待機してからリトライします。

- `--xml, -X`

XML 出力を生成

```
<field name="column_name">NULL</field>
```

`--xml` を `mysql` とともに使用した場合の出力は、`mysqldump --xml` の出力と一致します。詳細は、[セクション 4.5.4 「mysqldump — データベースバックアッププログラム」](#) を参照してください。

次に示すように、XML 出力は XML 名前空間も使用します。

```
shell> mysql --xml -uroot -e "SHOW VARIABLES LIKE 'version%'"
<?xml version="1.0"?>

<resultset statement="SHOW VARIABLES LIKE 'version%'" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<row>
<field name="Variable_name">version</field>
<field name="Value">5.0.40-debug</field>
</row>

<row>
<field name="Variable_name">version_comment</field>
<field name="Value">Source distribution</field>
</row>

<row>
<field name="Variable_name">version_compile_machine</field>
<field name="Value">i686</field>
</row>

<row>
<field name="Variable_name">version_compile_os</field>
<field name="Value">suse-linux-gnu</field>
</row>
</resultset>
```

- `--zstd-compression-level=level`

`zstd` 圧縮アルゴリズムを使用するサーバーへの接続に使用する圧縮レベル。許可されるレベルは 1 から 22 で、大きい値は圧縮レベルの増加を示します。デフォルトの `zstd` 圧縮レベルは 3 です。圧縮レベルの設定は、`zstd` 圧縮を使用しない接続には影響しません。

詳細は、[セクション 4.2.8 「接続圧縮制御」](#) を参照してください。

このオプションは MySQL 8.0.18 で追加されました。

4.5.1.2 mysql クライアントコマンド

`mysql` は、ユーザーが発行する各 SQL ステートメントを、実行のためサーバーに送信します。 `mysql` 自体が解釈するコマンドもあります。これらのコマンドのリストを表示するには、 `mysql>` プロンプトで `help` または `\h` と入力します。

```
mysql> help

List of all MySQL commands:
Note that all text commands must be first on line and end with ';'
?      (?) Synonym for 'help'.
clear  (lc) Clear the current input statement.
connect (lr) Reconnect to the server. Optional arguments are db and host.
delimiter (ld) Set statement delimiter.
edit   (le) Edit command with $EDITOR.
ego    (lG) Send command to mysql server, display result vertically.
exit   (lq) Exit mysql. Same as quit.
go     (lg) Send command to mysql server.
help   (lh) Display this help.
nopager (ln) Disable pager, print to stdout.
notee  (lt) Don't write into outfile.
pager  (lP) Set PAGER [to_pager]. Print the query results via PAGER.
print  (lp) Print current command.
prompt (lR) Change your mysql prompt.
quit   (lq) Quit mysql.
rehash (l#) Rebuild completion hash.
source (l.) Execute an SQL script file. Takes a file name as an argument.
status (ls) Get status information from the server.
system (l!) Execute a system shell command.
tee    (lT) Set outfile [to_outfile]. Append everything into given
        outfile.
use    (lu) Use another database. Takes database name as argument.
charset (lC) Switch to another charset. Might be needed for processing
        binlog with multi-byte charsets.
warnings (lW) Show warnings after every statement.
nowarning (lw) Don't show warnings after every statement.
resetconnection(lx) Clean session context.
query_attributes(l) Sets string parameters (name1 value1 name2 value2 ...)
for the next query to pick up.

For server side help, type 'help contents'
```

`--binary-mode` オプションを指定して `mysql` を起動すると、非対話モード (`mysql` にパイプされた入力または `source` コマンドを使用してロードされた入力の場合) の `charset` および `delimiter` を除き、すべての `mysql` コマンドが無効になります。

各コマンドにはそれぞれロング形式とショート形式があります。長い形式では大文字と小文字は区別されません。短い形式では区別されます。ロング形式にはオプションのセミコロン終端記号があとに続くこともありますが、ショート形式ではありません。

複数行の `/* ... */` コメント内でのショートフォームコマンドの使用はサポートされていません。ショートフォームコマンドは、オブジェクト定義に格納されている `/*+ ... */` オプティマイザヒントコメントと同様に、単一行の `/*! ... */` パージョンコメント内で機能します。オプティマイザヒントのコメントがオブジェクト定義に格納され、`mysql` でリロードされたときにダンプファイルによってそのようなコマンドが実行されることが懸念される場合は、`--binary-mode` オプションを指定して `mysql` を起動するか、`mysql` 以外のリロードクライアントを使用します。

- `help [arg]`, `\h [arg]`, `\? [arg]`, `? [arg]`

使用可能な `mysql` コマンドの一覧を示すヘルプメッセージを表示します。

`help` コマンドに引数を指定した場合、`mysql` は、サーバー側のヘルプにアクセスして MySQL リファレンスマニュアルの内容から検索するための文字列として引数を使用します。詳細は、[セクション4.5.1.4「mysql クライアントのサーバー側ヘルプ」](#)を参照してください。

- `charset charset_name, \C charset_name`

デフォルトの文字セットを変更し、[SET NAMES](#) ステートメントを発行します。これにより、自動再接続が有効になっている状態で `mysql` が稼働中の場合 (これは推奨されていません) に、クライアントとサーバーの間で文字セットの同期が保持されます。これは、指定された文字セットが再接続に使用されるためです。

- `clear, \c`

現在の入力をクリアします。これは、入力しているステートメントの実行を取りやめる場合に使用します。

- `connect [db_name [host_name]], \r [db_name [host_name]]`

サーバーに再接続します。オプションのデータベース名およびホスト名の引数を指定して、デフォルトのデータベースまたはサーバーが稼働しているホストを指定することもできます。省略した場合は、現在の値が使用されます。

`connect` コマンドでホスト名引数が指定されている場合、そのホストは `mysql` の起動時に指定された `--dns-srv-name` オプションよりも優先され、DNS SRV レコードが指定されます。

- `delimiter str, \d str`

`mysql` が SQL ステートメント間の区切り文字として解釈する文字列を変更します。デフォルトはセミコロン (;) 文字です。

区切り文字の文字列は、`delimiter` コマンド行で引用符ありまたはなしの引数として指定できます。単一引用符 (')、二重引用符 (")、または逆引用符 (``) 文字で囲むことができます。引用符で囲まれた文字列内に引用符を含めるには、別の引用符で文字列を囲むか、またはバックスラッシュ (\) 文字で引用符をエスケープ処理します。バックスラッシュは MySQL のエスケープ文字であるため、引用符で囲まれた文字列の外側では使用しないようにしてください。引用符で囲まれていない引数については、区切り文字は最初のスペースまたは行の最後まで読み取られます。引用符で囲まれた引数の場合、区切り文字はその行の対応する引用符まで読み取られます。

`mysql` は、区切り文字列のインスタンスがどこで検出されても (引用符で囲まれた文字列内を除いて)、ステートメントの区切り文字として解釈します。ほかの語に出現する可能性のある区切り文字を定義しないように注意してください。たとえば、デリミタを `X` として定義する場合、`INDEX` という語をステートメントで使用することはできません。`mysql` は、これを `INDE` として解釈し、その後にデリミタ `X` を付けます。

`mysql` によって認識される区切り文字がデフォルトの ; 以外の何かに設定されている場合、その文字のインスタンスは解釈されずにサーバーに送信されます。しかし、サーバー自体は引き続き ; をステートメントの区切り文字として解釈し、その解釈に従ってステートメントを処理します。サーバー側でのこの動作は、複数ステートメントの実行 ([Multiple Statement Execution Support](#) を参照してください) や、ストアードプロシージャおよび関数の本体、トリガー、およびイベントの解析 ([セクション25.1「ストアードプログラムの定義」](#) を参照してください) に効果があります。

- `edit, \e`

現在の入力ステートメントを編集します。`mysql` では、`EDITOR` および `VISUAL` 環境変数の値を確認して、どのエディタを使用するかを判断します。どちらの変数も設定されていない場合、デフォルトのエディタは `vi` です。

`edit` コマンドは Unix でのみ機能します。

- `ego, \G`

現在のステートメントを、実行するためにサーバーに送信し、結果を縦の形式で表示します。

- `exit, \q`

`mysql` を終了します。

- `go, \g`

現在のステートメントを、実行するためにサーバーに送信します。

- `nopager, \n`

出力のページングを無効にします。`pager` の説明を参照してください。

`nopager` コマンドは Unix でのみ機能します。

- `notee, \t`

`tee` ファイルへの出力コピーを無効にします。 `tee` の説明を参照してください。

- `nowarning, \w`

各ステートメントのあとの警告の表示を無効にします。

- `pager [command], \P [command]`

出力のページングを有効にします。 `mysql` を呼び出すときに `--pager` オプションを使用することで、`less`、`more`、またはその他の同様のプログラムなどの Unix プログラムを使って、インタラクティブモードでクエリー結果を参照または検索できます。 オプションで値を特定しない場合、`mysql` は `PAGER` 環境変数の値を確認し、ページャーをその値に設定します。 ページャー機能はインタラクティブモードの場合のみ機能します。

出力ページングは `pager` コマンドでインタラクティブに有効にでき、`nopager` で無効にできます。 このコマンドはオプションの引数を取ります。 指定された場合、ページングプログラムはそれに設定されます。 引数がない場合、ページャーはコマンド行で設定されたもの、またはページャーが指定されていない場合は `stdout` になります。

出力ページングは Unix 上でのみ機能します。 これは Windows では存在しない `popen()` 関数を使用するからです。 Windows では、クエリー出力の保存に `tee` オプションを代わりに使用できますが、これは場合によっては、出力のブラウズには `pager` ほど便利ではありません。

- `print, \p`

現在の入カステートメントを実行しないで出力します。

- `prompt [str], \R [str]`

`mysql` プロンプトを指定の文字列に再構成します。 プロンプトで使用できる特殊文字シーケンスについては、このセクションのあとの方で説明します。

引数なしで `prompt` コマンドを指定すると、`mysql` はプロンプトをデフォルトの `mysql>` にリセットします。

- `query_attributes name value [name value ...]`

サーバーに送信される次のクエリーに適用されるクエリー属性を定義します。 クエリー属性の目的および使用方法の詳細は、[セクション9.6「クエリー属性」](#)を参照してください。

`query_attributes` コマンドは、次のルールに従います:

- 属性名および値の書式および引用符のルールは、`delimiter` コマンドの場合と同じです。
 - このコマンドでは、最大 32 個の属性名/値のペアが許可されます。 名前と値の長さは最大 1024 文字です。 値を指定せずに名前を指定すると、エラーが発生します。
 - クエリーの実行前に複数の `query_attributes` コマンドが発行された場合は、最後のコマンドのみが適用されます。 クエリーの送信後、`mysql` は属性セットをクリアします。
 - 複数の属性が同じ名前前で定義されている場合、属性値を取得しようとすると未定義の結果になります。
 - 空の名前で定義された属性は名前前で取得できません。
 - `mysql` によるクエリーの実行中に再接続が発生した場合、`mysql` は再接続後に属性をリストアし、同じ属性を使用してクエリーを再度実行できるようにします。
- `quit, \q`

`mysql` を終了します。

- `rehash, \#`

ステートメントの入力中にデータベース、テーブル、およびカラムの名前補完を可能にする補完ハッシュを再構築します。(`--auto-rehash` オプションの説明を参照してください。)

- `resetconnection, \x`

接続をリセットしてセッションステートをクリアします。これには、`query_attributes` コマンドを使用して定義された現在のクエリー属性のクリアが含まれます。

接続をリセットすると、`mysql_change_user()` または自動再接続に似た効果がありますが、接続がクローズされて再オープンされず、再認証が行われない点が異なります。`mysql_change_user()` および `Automatic Reconnection Control` を参照してください。

この例では、`resetconnection` がセッションステートに保持されている値をクリアする方法を示します:

```
mysql> SELECT LAST_INSERT_ID(3);
+-----+
| LAST_INSERT_ID(3) |
+-----+
|          3 |
+-----+

mysql> SELECT LAST_INSERT_ID();
+-----+
| LAST_INSERT_ID() |
+-----+
|          3 |
+-----+

mysql> resetconnection;

mysql> SELECT LAST_INSERT_ID();
+-----+
| LAST_INSERT_ID() |
+-----+
|          0 |
+-----+
```

- `source file_name, \. file_name`

指定されたファイルを読み取り、その中に含まれているステートメントを実行します。Windows では、パス名区切り文字を `/` または `\\` として指定します。

引用符は、ファイル名自体の一部として使用されます。最良の結果を得るには、名前に空白文字を含めないでください。

- `status, \s`

使用中の接続とサーバーに関するステータス情報を表示します。`--safe-updates` を有効にして実行している場合、`status` はクエリーに影響する `mysql` 変数の値も出力します。

- `system command, \! command`

デフォルトのコマンドインタプリタを使って指定のコマンドを実行します。

MySQL 8.0.19 より前は、`system` コマンドは Unix でのみ機能します。8.0.19 では、Windows でも動作します。

- `tee [file_name], \T [file_name]`

`mysql` の呼び出し時に `--tee` オプションを使用することで、ステートメントとその出力をログに記録できます。画面上に表示されるデータはすべて指定されたファイルに追加されます。これはデバッグを行う際にも非常に便利です。`mysql` では、各ステートメントが終わって次のプロンプトが表示される直前に、結果をファイルにフラッシュします。`tee` 機能はインタラクティブモードの場合のみ機能します。

`tee` コマンドを使用すれば、この機能をインタラクティブに有効にできます。パラメータがない場合、以前のファイルが使用されます。`tee` ファイルを無効にするには、`notee` コマンドを使用します。`tee` を実行するとロギングが再度有効になります。

- `use db_name, \u db_name`

`db_name` をデフォルトデータベースとして使用します。

- `warnings, \W`

各ステートメントのあとの警告の表示を有効にします (存在する場合)。

`pager` コマンドのヒントを次に記します。

- これを使用してファイルに書き込むと、結果はファイルにのみ送られます。

```
mysql> pager cat > /tmp/log.txt
```

ページャーとして使用するプログラムのオプションを渡すこともできます。

```
mysql> pager less -n -i -S
```

- 前の例の、`-S` オプションに注意してください。幅の広いクエリー結果のブラウズの際に非常に便利です。非常に幅の広い結果セットは、画面上では読みにくい場合があります。`less` に対して `-S` オプションを指定すると、左右の方向キーを使用して横にスクロールできるため、結果セットが読みやすくなります。また、`-S` を `less` 内でインタラクティブに使用して、水平方向のブラウズモードをオン/オフにできます。詳細は、`less` マニュアルページを参照してください。

```
shell> man less
```

- `-F` および `-X` オプションを `less` で使用すると、出力が 1 画面に収まる場合にプログラムを終了させることができ、これはスクロールが不要なときに便利です。

```
mysql> pager less -n -i -S -F -X
```

- クエリー出力の取り扱いに関する非常に複雑なページャーコマンドを指定できます。

```
mysql> pager cat | tee /dr1/tmp/res.txt \  
| tee /dr2/tmp/res2.txt | less -n -i -S
```

この例では、コマンドはクエリーの結果を `/dr1` および `/dr2` にマウントされた 2 つの異なるファイルシステムの 2 つの異なるディレクトリ内の 2 つのファイルに送信し、さらに `less` を使用して結果を画面に表示します。

`tee` 関数と `pager` 関数を組み合わせることもできます。`tee` ファイルを有効にし、`pager` を `less` に設定してあれば、`less` プログラムを使って結果をブラウズしつつ、同時にすべてをファイルに追加できます。`pager` コマンドと一緒に使用する Unix `tee` と、`mysql` に組み込みの `tee` コマンドの違いは、組み込みの `tee` は Unix `tee` がなくても機能することです。また、組み込みの `tee` は画面に出力されるものすべてをログに記録しますが、`pager` と一緒に使用される Unix `tee` はそこまでログに記録しません。さらに、`tee` ファイルのロギングは `mysql` 内からインタラクティブにオン/オフできます。これは一部のクエリーのみをファイルにログするとき有効です。

`prompt` コマンドはデフォルトの `mysql>` プロンプトを再構成します。プロンプトを定義するための文字列には、次の特殊なシーケンスを含めることができます。

オプション	説明
<code>\C</code>	現在の接続識別子
<code>\c</code>	ステートメントを発行するたびにインクリメントするカウンタ
<code>\D</code>	現在の日付 (フルで)
<code>\d</code>	デフォルトデータベース
<code>\h</code>	サーバーホスト
<code>\l</code>	現在の区切り文字
<code>\m</code>	現在の時間の分
<code>\n</code>	改行文字
<code>\O</code>	3 文字の形式の現在の月 (Jan、Feb、...)
<code>\o</code>	数字形式の現在の月

オプション	説明
\P	am/pm
\p	現在の TCP/IP ポートまたはソケットファイル
\R	現在の時間、24 時間表記 (0-23)
\r	現在の時間、12 時間表記 (1-12)
\S	セミコロン
\s	現在の時間の秒
\t	タブ文字
\U	完全な user_name@host_name アカウント名
\u	ユーザー名
\v	サーバーバージョン
\w	3 文字の形式の現在の曜日 (Mon, Tue, ...)
\Y	現在の年 (4 桁)
\y	現在の年 (2 桁)
_	スペース
\	スペース (バックスラッシュのあとにスペースがあります)
\'	単一引用符
"	二重引用符
\\	リテラルの \ バックスラッシュ文字
\x	x (上記にないすべての「x」)

プロンプトはいくつかの方法でセットできます。

- 環境変数を使用します。MYSQL_PS1 環境変数をプロンプト文字列に設定できます。例:

```
shell> export MYSQL_PS1="(u@h) [d]> "
```

- コマンド行オプションを使用します。コマンド行で、--prompt オプションを mysql に設定できます。例:

```
shell> mysql --prompt="(u@h) [d]> "  
(user@host) [database]>
```

- オプションファイルを使用します。prompt オプションを、ホームディレクトリの /etc/my.cnf または .my.cnf ファイルなど、任意の MySQL オプションファイルの [mysql] グループに設定できます。例:

```
[mysql]  
prompt=(\u@\h) [\d]>\_
```

この例では、バックスラッシュが 2 つあることに注意してください。オプションファイルで prompt オプションを使用してプロンプトを設定する場合、特別なプロンプトオプションを使用するときはバックスラッシュを 2 つ使用することをお勧めします。許可されるプロンプトオプションのセットと、オプションファイルで認識される特殊なエスケープシーケンスのセットには、重複があります。(オプションファイルでのエスケープシーケンスに関するルールは [セクション 4.2.2.2 「オプションファイルの使用」](#) にリストされています。) 単一のバックスラッシュを使用している場合、この重複が問題となる可能性があります。たとえば、\s は現在の秒の値としてではなく、スペースとして解釈されます。次の例は、現在の時間を hh:mm:ss> 形式で含めるプロンプトをオプションファイル内に定義する方法を示しています:

```
[mysql]  
prompt="\r:\m:\s> "
```

- プロンプトをインタラクティブに設定します。prompt コマンド (または \R コマンド) を使用すると、プロンプトをインタラクティブに変更できます。例:

```
mysql> prompt (u@h) [d]>\_
```

```
PROMPT set to '(u@h) [d]>\'  
(user@host) [database]>  
(user@host) [database]> prompt  
Returning to default PROMPT of mysql>  
mysql>
```

4.5.1.3 mysql クライアントロギング

mysql クライアントは、対話形式で実行されるステートメントに対して次のタイプのロギングを実行できます:

- Unix では、mysql によってステートメントが履歴ファイルに書き込まれます。デフォルトでは、このファイルは `.mysql_history` という名前で、ユーザーのホームディレクトリにあります。別のファイルを指定する場合は、`MYSQL_HISTFILE` 環境変数値を設定します。
- すべてのプラットフォームで、`--syslog` オプションが指定されている場合、mysql はシステムロギング機能にステートメントを書き込みます。Unix の場合は `syslog` で、Windows の場合は Windows イベントログです。ログに記録されたメッセージが表示される宛先は、システムによって異なります。Linux では、多くの場合、宛先は `/var/log/messages` ファイルです。

次の説明では、すべてのロギングタイプに適用される特性について説明し、各ロギングタイプに固有の情報を提供します。

- [ロギングの方法](#)
- [履歴ファイルの制御](#)
- [syslog ロギング特性](#)

ロギングの方法

有効なロギング先ごとに、ステートメントのロギングは次のように行われます:

- ステートメントは、インタラクティブに実行された場合のみログに記録されます。ステートメントは、たとえばファイルまたはパイプから読み取られる場合はインタラクティブではありません。 `--batch` オプションまたは `--execute` オプションを使用することによって、ステートメントのロギングを抑制することもできます。
- ステートメントは、「ignore」リスト内のパターンのいずれかに一致する場合には無視され、ログに記録されません。このリストについてはあとで説明します。
- mysql は、無視されず空でない各ステートメント行を個別にログに記録します。
- 無視されないステートメントが複数の行にまたがる場合 (終端区切り文字を含まない)、mysql は行を連結して完全なステートメントを形成し、改行をスペースに対応付け、結果に区切り文字を付け加えてログに記録します。

その結果、複数行にまたがる入力ステートメントが 2 回ログに記録されることがあります。次の入力について考えます。

```
mysql> SELECT  
-> 'Today is'  
-> ,  
-> CURDATE()  
-> ;
```

この場合、mysql では「SELECT」、「Today is」、「,」、「CURDATE()」および「;」の行が読み取り時に記録されます。また、`SELECT\n'Today is'\n,\nCURDATE()` を `SELECT 'Today is', CURDATE()` に対応付けたあと、区切り文字を追加した完全なステートメントもログに記録します。したがって、ログに記録される出力には次の行がありません。

```
SELECT  
'Today is'  
,  
CURDATE()  
;  
SELECT 'Today is', CURDATE();
```

mysql では、「ignore」リスト内の任意のパターンに一致するロギング目的のステートメントは無視されます。デフォルトでは、パターンのリストは `"*IDENTIFIED*:PASSWORD*"` で、パスワードを参照するステートメントを無視します。パターン一致では、大/小文字は区別されません。パターン内では、特殊文字が 2 つあります。

- `?` は任意の 1 文字に一致します。
- `*` はゼロ個以上の文字の任意のシーケンスに一致します。

追加のパターンを指定するには、`--histignore` オプションを使用するか、または `MYSQL_HISTIGNORE` 環境変数を設定します。(両方を指定した場合はオプション値が優先されます。) この値は、コロンで区切られた 1 つ以上のパターンのリストである必要があり、デフォルトのパターンリストに追加されます。

コマンド行で指定されたパターンは、コマンドインタプリタで特殊な扱いを受けることを防ぐために、引用符で囲むかエスケープ処理を行う必要がある場合があります。たとえば、パスワードを参照するステートメントに加えて `UPDATE` ステートメントおよび `DELETE` ステートメントのロギングを抑制するには、`mysql` を次のように呼び出します。

```
shell> mysql --histignore="**UPDATE**DELETE**"
```

履歴ファイルの制御

`.mysql_history` ファイルには、パスワードを含む SQL ステートメントのテキストなどの機密情報が書き込まれる場合があるため、制限付きアクセスモードで保護するようにしてください。 [セクション6.1.2.1「パスワードセキュリティのためのエンドユーザーガイドライン」](#) を参照してください。 up-arrow キーを使用して履歴をリコールすると、ファイル内のステートメントに `mysql` クライアントからアクセスできます。 [インタラクティブ履歴の無効化](#) を参照してください。

履歴ファイルを維持しない場合は、`.mysql_history` が存在すればまずそれを削除します。次の手法のいずれかを使用してふたたび作成されないようにします。

- `MYSQL_HISTFILE` 環境変数を `/dev/null` に設定します。ログインするたびにこの設定が有効になるようにするには、これをシェルの起動ファイルに置きます。
- `.mysql_history` を `/dev/null` へのシンボリックリンクとして作成します。これは一度のみの実行で済みます。

```
shell> ln -s /dev/null $HOME/.mysql_history
```

syslog ロギング特性

`--syslog` オプションが指定されている場合、`mysql` は対話型のステートメントをシステムロギング機能に書き込みます。メッセージロギングには、次の特性があります。

ロギングは「information」レベルで行われます。これは、Unix/Linux `syslog` 機能の `syslog` および Windows イベントログの `EVENTLOG_INFORMATION_TYPE` の `LOG_INFO` 優先度に対応します。ロギング機能の構成については、システムのドキュメントを参照してください。

メッセージサイズは 1024 バイトに制限されています。

メッセージは、`MysqlClient` という識別子の後に次の値が続くもので構成されます：

- `SYSTEM_USER`
オペレーティングシステムのユーザー名 (ログイン名)。ユーザーが不明な場合は `--`。
- `MYSQL_USER`
MySQL ユーザー名 (`--user` オプションで指定) または `--` (ユーザーが不明な場合)。
- `CONNECTION_ID`:
クライアント接続識別子。これは、セッション内の `CONNECTION_ID()` 関数の値と同じです。
- `DB_SERVER`
サーバーホスト、またはホストが不明な場合は `--`。
- `DB`
デフォルトのデータベース、またはデータベースが選択されていない場合は `--`。

- QUERY

ログに記録されたステートメントのテキスト。

次に、`--syslog` を使用して Linux で生成される出力のサンプルを示します。この出力は読みやすくするためにフォーマットされており、ログに記録される各メッセージは実際には単一行を取ります。

```
Mar 7 12:39:25 myhost MysqlClient[20824]:
SYSTEM_USER:'oscar', MYSQL_USER:'my_oscar', CONNECTION_ID:23,
DB_SERVER:'127.0.0.1', DB:'-', QUERY:'USE test;'
Mar 7 12:39:28 myhost MysqlClient[20824]:
SYSTEM_USER:'oscar', MYSQL_USER:'my_oscar', CONNECTION_ID:23,
DB_SERVER:'127.0.0.1', DB:'test', QUERY:'SHOW TABLES;'
```

4.5.1.4 mysql クライアントのサーバー側ヘルプ

```
mysql> help search_string
```

`help` コマンドに引数を指定した場合、`mysql` は、サーバー側のヘルプにアクセスして MySQL リファレンスマニュアルの内容から検索するための文字列として引数を使用します。このコマンドを適切に操作するには、`mysql` データベース内のヘルプテーブルがヘルプトピック情報で初期化されていることが必要です ([セクション5.1.17「サーバー側ヘルプのサポート」](#)を参照してください)。

検索文字列に一致するものがない場合、検索は失敗に終わります。

```
mysql> help me
```

```
Nothing found
Please try to run 'help contents' for a list of all accessible topics
```

ヘルプカテゴリのリストを閲覧するには `help contents` を使用してください。

```
mysql> help contents
You asked for help about help category: "Contents"
For more information, type 'help <item>', where <item> is one of the
following categories:
  Account Management
  Administration
  Data Definition
  Data Manipulation
  Data Types
  Functions
  Functions and Modifiers for Use with GROUP BY
  Geographic Features
  Language Structure
  Plugins
  Storage Engines
  Stored Routines
  Table Maintenance
  Transactions
  Triggers
```

検索文字列に一致するものが複数ある場合は、`mysql` は一致するトピックのリストを表示します。

```
mysql> help logs
Many help items for your request exist.
To make a more specific request, please type 'help <item>',
where <item> is one of the following topics:
  SHOW
  SHOW BINARY LOGS
  SHOW ENGINE
  SHOW LOGS
```

トピックのヘルプエントリを閲覧するには、そのトピックを検索文字列として使用してください。

```
mysql> help show binary logs
Name: 'SHOW BINARY LOGS'
Description:
Syntax:
SHOW BINARY LOGS
SHOW MASTER LOGS
```


Lists the binary log files on the server. This statement is used as part of the procedure described in [purge-binary-logs], that shows how to determine which logs can be purged.

```
mysql> SHOW BINARY LOGS;
+-----+-----+-----+
| Log_name | File_size | Encrypted |
+-----+-----+-----+
| binlog.000015 | 724935 | Yes |
| binlog.000016 | 733481 | Yes |
+-----+-----+-----+
```

検索文字列には、ワイルドカード文字 `%` および `_` を含めることができます。これらは `LIKE` 演算子で実行されるパターンマッチング演算と同じ意味を持ちます。たとえば、`HELP rep%` は `rep` で始まるトピックのリストを返します。

```
mysql> HELP rep%
Many help items for your request exist.
To make a more specific request, please type 'help <item>',
where <item> is one of the following
topics:
REPAIR TABLE
REPEAT FUNCTION
REPEAT LOOP
REPLACE
REPLACE FUNCTION
```

4.5.1.5 テキストファイルから SQL ステートメントを実行する

`mysql` クライアントは、通常次のようにインタラクティブに使用されます。

```
shell> mysql db_name
```

しかし、SQL ステートメントをファイルに入れ、`mysql` にその入力をファイルから読み取るように指示することも可能です。そのためには、実行するステートメントを含む `text_file` を作成します。それから、次に示されるように `mysql` を起動します。

```
shell> mysql db_name < text_file
```

ファイルの最初のステートメントとして `USE db_name` ステートメントを配置する場合、コマンド行でデータベース名を指定する必要はありません。

```
shell> mysql < text_file
```

`mysql` がすでに稼働している場合は、`source` コマンドまたは `\.` コマンドを使用して SQL スクリプトファイルを実行できます。

```
mysql> source file_name
mysql> \. file_name
```

スクリプトでユーザーに進行状況を表示する場合があります。このためには、次のステートメントを挿入できます。

```
SELECT '<info_to_display>' AS '';
```

示されたステートメントは `<info_to_display>` を出力します。

また、`--verbose` オプションを付けて `mysql` を呼び出すこともでき、生成される結果の前に各ステートメントが表示されるようになります。

`mysql` は、入力ファイルの先頭にある Unicode バイト順マーク (BOM) 文字を無視します。以前は、それらを読み取ってサーバーに送信していたため、構文エラーが発生していました。BOM が存在しても、`mysql` はデフォルトの文字セットを変更しません。これを行うには、`--default-character-set=utf8` などのオプションを付けて `mysql` を呼び出します。

バッチモードの詳細は、[セクション3.5「バッチモードでの MySQL の使用」](#) を参照してください。

4.5.1.6 mysql クライアントのヒント

このセクションでは、`mysql` のより効果的な使用方法および `mysql` の操作動作について説明します。

- 入力行の編集
- インタラクティブ履歴の無効化
- Windows における Unicode のサポート
- クエリー結果を縦に表示する
- セーフ更新モードの使用 (--safe-updates)
- mysql の自動再接続を無効にする
- mysql クライアントパーサーとサーバーパーサー

入力行の編集

mysql は入力行の編集をサポートし、現在の入力行を修正したり以前の入力行を呼び出したりできます。たとえば、left-arrow キーと right-arrow キーは現在の入力行内で水平に移動し、up-arrow キーと down-arrow キーは以前に入力した行のセット内で上下に移動します。「バックスペース」でカーソルの前の文字を削除でき、新しい文字を入力するとカーソルの位置に挿入されます。行を入力するには、「Enter」を押します。

Windows では、編集キーシーケンスはコンソールウィンドウでコマンドの編集に関してサポートされているものと同じです。Unix では、キーシーケンスは mysql のビルドに使用された入力ライブラリ (たとえば、libedit または readline ライブラリ) に依存します。

libedit ライブラリおよび readline ライブラリのドキュメントは、オンラインで入手できます。所定の入力ライブラリで許可されるキーシーケンスのセットを変更するには、ライブラリ起動ファイルでキーバインドを定義します。これはホームディレクトリにあるファイルで、.editrc は libedit 用、.inputrc は readline 用です。

たとえば libedit では、Control+W は現在のカーソル位置の前にあるものをすべて削除し、Control+U は行全体を削除します。readline では、Control+W はカーソルの前の単語を削除し、Control+U は現在のカーソル位置の前にあるものをすべて削除します。mysql が libedit を使用してビルドされた場合は、これら 2 つのキーに関して readline の動作を好むユーザーは、.editrc ファイルに次の行を置くことができます (必要に応じてファイルを作成します)。

```
bind "^W" ed-delete-prev-word
bind "^U" vi-kill-line-prev
```

現在のキーバインディングのセットを表示するには、.editrc の最後に bind のみを示す行を一時的に配置します。mysql では、起動時にバインディングが表示されます。

インタラクティブ履歴の無効化

up-arrow キーを使用すると、現在および前のセッションから入力行をリコールできます。コンソールが共有されている場合、この動作は適切でない可能性があります。mysql では、ホストプラットフォームに応じて、対話型履歴の一部または全部の無効化をサポートしています。

Windows では、履歴はメモリに保存されます。Alt+F7 は、現在の履歴バッファのメモリーに格納されている入力行をすべて削除します。また、F7 で表示され、F9 でリコールされた入力行の前の連番のリストも削除されます。Alt+F7 で現在の履歴バッファを再移入した後に入力された新しい入力行。--syslog オプションを使用して mysql を起動した場合、バッファをクリアしても Windows イベントビューアへのロギングは妨げられません。コンソールウィンドウを閉じると、現在の履歴バッファもクリアされます。

Unix で対話型履歴を無効にするには、まず .mysql_history ファイルが存在する場合はそれを削除します (それ以外の場合は以前のエンタリがリコールされます)。次に、--histignore="" オプションを指定して mysql を起動し、すべての新しい入力行を無視します。リコール (およびロギング) 動作を再度有効にするには、オプションを指定せずに mysql を再起動します。

.mysql_history ファイルが作成されないようにし (履歴ファイルの制御 を参照)、--histignore="" を使用して mysql クライアントを起動すると、対話型履歴リコール機能が完全に無効になります。または、--histignore オプションを省略すると、現在のセッション中に入力された入力行をリコールできます。

Windows における Unicode のサポート

Windows には、コンソールからの読取りおよびコンソールへの書き込み用に UTF-16LE に基づく API が用意されています。Windows 用の mysql クライアントでは、これらの API を使用できます。Windows インストーラに

よって、MySQL メニューに [MySQL command line client - Unicode](#) という名前の項目が作成されます。この項目は、Unicode を使用して MySQL サーバーにコンソール経由で通信するように設定されたプロパティで `mysql` クライアントを呼び出します。

このサポートを手動で利用するためには、互換性のある Unicode フォントを使用するコンソール内で `mysql` を実行し、デフォルト文字セットをサーバーとの通信でサポートされる Unicode 文字セットに設定します。

1. コンソールウィンドウを開きます。
2. コンソールウィンドウプロパティに移動して「フォント」タブを選択し、Lucida Console またはその他の互換性のある Unicode フォントを選択します。コンソールウィンドウはデフォルトでは Unicode に不適切な DOS ラスターフォントを使用するため、これが重要です。
3. `mysql.exe` を `--default-character-set=utf8` (または `utf8mb4`) オプションで実行します。 `utf16le` はクライアント文字セットとして使用できない文字セットの 1 つであるため、このオプションが必要です。 [許可されていないクライアント文字セット](#) を参照してください。

これらの変更により、`mysql` は Windows API を使用して UTF-16LE を使用してコンソールと通信し、UTF-8 を使用してサーバーと通信します。(前述のメニュー項目は、フォントと文字セットを今説明したように設定します。)

`mysql` を起動するたびにこれらのステップを実行しなくてもいいように、`mysql.exe` を呼び出すショートカットを作成できます。このショートカットは、コンソールフォントを Lucida Console またはその他の互換性のある Unicode フォントに設定し、`--default-character-set=utf8` (または `utf8mb4`) オプションを `mysql.exe` に渡すようにしてください。

または、コンソールフォントの設定のみを行うショートカットを作成し、文字セットは `my.ini` ファイルの `[mysql]` グループで設定します。

```
[mysql]
default-character-set=utf8
```

クエリー結果を縦に表示する

クエリー結果の中には、縦表示の方が、通常の横向きの表形式よりもはるかに読みやすい場合があります。セミコロン`;`の代わりに `\G` でクエリーを終了することで、クエリーを縦に表示できます。たとえば、多くの場合、改行を含む長いテキスト値は縦の出力の方がはるかに読みやすくなります。

```
mysql> SELECT * FROM mails WHERE LENGTH(txt) < 300 LIMIT 300,1\G
***** 1. row *****
msg_nro: 3068
date: 2000-03-01 23:29:50
time_zone: +0200
mail_from: Jones
reply: jones@example.com
mail_to: "John Smith" <smith@example.com>
subj: UTF-8
txt: >>>> "John" == John Smith writes:

John> Hi. I think this is a good idea. Is anyone familiar
John> with UTF-8 or Unicode? Otherwise, I'll put this on my
John> TODO list and see what happens.

Yes, please do that.

Regards,
Jones
file: inbox-jani-1
hash: 190402944
1 row in set (0.09 sec)
```

セーフ更新モードの使用 (--safe-updates)

初心者にとって、使いやすい起動オプションは `--safe-updates` (または同じ効果のある `--i-am-a-dummy`) です。Safe-updates モードは、`UPDATE` または `DELETE` ステートメントを発行したが、変更する行を示す `WHERE` 句を忘れた場合に役立ちます。通常、このようなステートメントはテーブル内のすべての行を更新または削除します。`--safe-updates` では、行を識別するキー値または `LIMIT` 句 (あるいはその両方) を指定することによってのみ、行を変更でき

ます。これにより、事故を予防します。Safe-updates モードでは、非常に大きな結果セットを生成する (または生成すると見積もられる) `SELECT` ステートメントも制限されます。

`--safe-updates` オプションを使用すると、`mysql` は MySQL サーバーに接続するときに次のステートメントを実行して、`sql_safe_updates`、`sql_select_limit` および `max_join_size` システム変数のセッション値を設定します:

```
SET sql_safe_updates=1, sql_select_limit=1000, max_join_size=1000000;
```

`SET` ステートメントは、次のようにステートメントの処理に影響します:

- `sql_safe_updates` を有効にすると、`WHERE` 句でキー制約が指定されていない場合、または `LIMIT` 句 (あるいはその両方) が指定されていない場合に、`UPDATE` ステートメントと `DELETE` ステートメントでエラーが発生します。例:

```
UPDATE tbl_name SET not_key_column=val WHERE key_column=val;
```

```
UPDATE tbl_name SET not_key_column=val LIMIT 1;
```

- `sql_select_limit` を 1,000 に設定すると、ステートメントに `LIMIT` 句が含まれていないかぎり、すべての `SELECT` 結果セットが 1,000 行に制限されます。
- `max_join_size` を 1,000,000 に設定すると、サーバーが 1,000,000 を超える行の組合せを調べる必要があると見積もった場合に、複数テーブルの `SELECT` ステートメントでエラーが発生します。

1,000 および 1,000,000 以外の結果セット制限を指定するには、`mysql` の起動時に `--select-limit` および `--max-join-size` オプションを使用してデフォルトをオーバーライドします:

```
mysql --safe-updates --select-limit=500 --max-join-size=10000
```

オプティマイザがキーカラムでインデックスを使用しないことを決定した場合、`UPDATE` および `DELETE` ステートメントでは、`WHERE` 句で指定されたキーを使用してもセーフ更新モードでエラーが発生する可能性があります:

- メモリ使用量が `range_optimizer_max_mem_size` システム変数で許可されているメモリ使用量を超える場合、インデックスに対する範囲アクセスは使用できません。その後、オプティマイザはテーブルスキャンにフォールバックします。範囲最適化のためのメモリ使用の制限を参照してください。
- キー比較で型変換が必要な場合は、インデックスを使用できません (セクション 8.3.1 「MySQL のインデックスの使用の仕組み」を参照)。インデックス付けされた文字列カラム `c1` が、`WHERE c1 = 2222` を使用して数値と比較されるとします。このような比較では、文字列値が数値に変換され、オペランドが数値的に比較され (セクション 12.3 「式評価での型変換」を参照)、インデックスの使用が妨げられます。safe-updates モードが有効な場合は、エラーが発生します。

MySQL 8.0.13 の時点では、セーフ更新モードには次の動作も含まれます:

- `UPDATE` および `DELETE` ステートメントを使用した `EXPLAIN` では、セーフ更新エラーは生成されません。これにより、`EXPLAIN` と `SHOW WARNINGS` を使用してインデックスが使用されない理由を確認できます。これは、`range_optimizer_max_mem_size` 違反や型変換が発生し、`WHERE` 句でキーカラムが指定されていてもオプティマイザがインデックスを使用しない場合などに役立ちます。
- safe-updates エラーが発生すると、エラーメッセージには、失敗の理由に関する情報を提供するために生成された最初の診断が含まれます。たとえば、`range_optimizer_max_mem_size` 値を超えたか、型変換が発生したことを示すメッセージが表示される場合があり、そのいずれの場合もインデックスを使用できません。
- 複数テーブルの削除および更新の場合、いずれかのターゲットテーブルでテーブルスキャンが使用されている場合にのみ、安全な更新を有効にしてエラーが生成されます。

mysql の自動再接続を無効にする

ステートメントの送信中にサーバーとの接続が切断された場合、`mysql` クライアントはただちに自動的にサーバーに一度再接続してステートメントを再度送信しようとします。ただし、`mysql` が再接続に成功しても、最初の接続は終了し、前セッションのオブジェクトと設定は失われます。この中には、一時テーブル、自動コミットモード、およびユーザー定義変数やセッション変数が含まれます。また、現トランザクションはロールバックします。この動作は危険な場合があります。たとえば、次の例では、サーバーはユーザーの了解なしに、最初のステートメントと 2 番目のステートメントの間にシャットダウンして再起動させられています。

```
mysql> SET @a=1;
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO t VALUES(@a);
ERROR 2006: MySQL server has gone away
No connection. Trying to reconnect...
Connection id: 1
Current database: test

Query OK, 1 row affected (1.30 sec)

mysql> SELECT * FROM t;
+-----+
| a |
+-----+
| NULL |
+-----+
1 row in set (0.05 sec)
```

@a ユーザー変数は接続とともに失われ、再接続後は未定義です。接続が失われた際に、mysql がエラーで終了することが望ましい場合、mysql クライアントを `--skip-reconnect` オプションで起動できます。

自動再接続および再接続時の状態情報への影響の詳細は、[Automatic Reconnection Control](#)を参照してください。

mysql クライアントパーサーとサーバーパーサー

mysql クライアントは、サーバー側の `mysqld` サーバーで使用される完全なパーサーと重複しないパーサーをクライアント側で使用します。これにより、特定の構成の処理に違いが生じる場合があります。例:

- サーバーパーサーは、`ANSI_QUOTES` SQL モードが有効な場合、"文字で区切られた文字列をプレーン文字列ではなく識別子として扱います。

mysql クライアントパーサーでは、`ANSI_QUOTES` SQL モードは考慮されません。`ANSI_QUOTES` が有効かどうかに関係なく、"、'および`文字で区切られた文字列は同じように扱われます。

- `/*! ... */` および `/*+ ... */` コメント内で、mysql クライアントパーサーは短い形式の `mysql` コマンドを解釈します。これらのコマンドはサーバー側で意味を持たないため、サーバーパーサーはこれらを解釈しません。

mysql でコメント内のショートフォームコマンドを解釈しないことが望ましい場合、部分的な回避策は `--binary-mode` オプションを使用することです。これにより、`\C` および `\d` を除くすべての `mysql` コマンドが非対話モードで無効になります (`mysql` にパイプされた入力または `source` コマンドを使用してロードされた入力の場合)。

4.5.2 mysqladmin — A MySQL Server 管理プログラム

mysqladmin は管理操作を実行するためのクライアントです。サーバーの構成や現在のステータスの確認、データベースの作成および削除、およびその他の用途に使用できます。

mysqladmin は次のように起動します。

```
shell> mysqladmin [options] command [command-arg] [command [command-arg]] ...
```

mysqladmin は次のコマンドをサポートします。コマンドの中にはコマンド名のあとに引数を取るものもあります。

- `create db_name`

`db_name` という名前の新しいデータベースを作成します。

- `debug`

MySQL 8.0.20 より前は、デバッグ情報をエラーログに書き込むようにサーバーに指示していました。接続されたユーザーは `SUPER` 権限を持っている必要があります。この情報の形式と内容は変更されることがあります。

これにはイベントスケジューラの情報が含まれます。[セクション25.4.5「イベントスケジューラのステータス」](#)を参照してください。

- `drop db_name`

`db_name` という名前のデータベースとそのテーブルをすべて削除します。

- `extended-status`

サーバステータス変数とその値を表示します。

- `flush-hosts`

ホストキャッシュ内の情報をすべてフラッシュします。 [セクション5.1.12.3「DNS ルックアップとホストキャッシュ」](#)を参照してください。

- `flush-logs [log_type ...]`

ログをすべてフラッシュします。

`mysqldadmin flush-logs` コマンドでは、フラッシュするログを指定するために、オプションのログタイプを指定できます。 `flush-logs` コマンドの後に、次のログタイプのリストをスペースで区切って指定できます: `binary`, `engine`, `error`, `general`, `relay`, `slow`。これらは、`FLUSH LOGS` SQL ステートメントに指定できるログタイプに対応します。

- `flush-privileges`

付与テーブルをリロードします (`reload` と同じ)。

- `flush-status`

ステータス変数をクリアします。

- `flush-tables`

テーブルをすべてフラッシュします。

- `flush-threads`

スレッドキャッシュをフラッシュします。

- `kill id,id,...`

サーバスレッドを強制終了します。複数のスレッド ID 値を指定する場合、リストにはスペースが存在してはいけません。

他のユーザーに属するスレッドを強制終了するには、接続ユーザーに `CONNECTION_ADMIN` 権限 (または非推奨の `SUPER` 権限) が必要です。

- `password new_password`

新しいパスワードを設定します。これにより、`mysqldadmin` でサーバへの接続に使用するアカウントのパスワードが `new_password` に変更されます。したがって、次回、同じアカウントを使用して `mysqldadmin` (またはその他のクライアントプログラム) を起動するときに、新しいパスワードを指定する必要があります。

警告

`mysqldadmin` を使用してパスワードを設定する場合は、セキュアでないとみなす必要があります。一部のシステムでは、使用しているパスワードが、コマンド行を表示するためにほかのユーザーによって起動できる `ps` などのシステムステータスプログラムによって表示可能になります。MySQL クライアントは通常、クライアントの初期化シーケンス中にコマンド行パスワード引数をゼロで上書きします。ただし、まだ値が表示可能な短い期間があります。また、一部のシステムではこの上書きの方法には効果がなく、パスワードは `ps` から表示可能になったままになります。(SystemV Unix システムおよびおそらくほかのシステムでもこの問題の影響があります。)

`new_password` 値にコマンドインタプリタで特殊なスペースやその他の文字が含まれている場合は、引用符で囲む必要があります。Windows では、単一引用符ではなく二重引用符を必ず使用してください。単一引用符はパスワードから取り除かれず、むしろパスワードの一部として解釈されます。例:


```
shell> mysqladmin password "my new password"
```

新しいパスワードは、`password` コマンドの後に省略できます。この場合、`mysqladmin` はパスワード値を要求し、パスワードをコマンド行で指定するのを避けることができます。パスワード値は、`password` が `mysqladmin` コマンド行の最後のコマンドである場合にかぎって省略できます。そうでない場合、次の引数がパスワードとみなされます。

注意

サーバーが `--skip-grant-tables` オプションを使用して起動された場合は、このコマンドを使用しないでください。パスワードの変更は適用されません。これは、同じコマンド行で `password` コマンドの前に `flush-privileges` コマンドを指定して付与テーブルを再度有効にする場合にも当てはまります (フラッシュ処理は接続後に行われるため)。ただし、`mysqladmin flush-privileges` コマンドを実行して付与テーブルを再度有効にしてから、個別に `mysqladmin password` コマンドを実行してパスワードを変更することは可能です。

- `ping`

サーバーが使用可能かどうかをチェックします。サーバーが稼働中の場合は `mysqladmin` のリターンステータスは 0 になり、稼働していない場合は 1 になります。 `Access denied` のようなエラーの場合でも 0 となります。これは、サーバーは稼働しているが接続を拒否したことを意味しており、サーバーが稼働していない状態とは異なるからです。

- `processlist`

アクティブなサーバスレッドのリストを表示します。これは `SHOW PROCESSLIST` ステートメントの出力と同様です。 `--verbose` オプションが指定されている場合、出力は `SHOW FULL PROCESSLIST` の出力と同様です。(セクション13.7.7.29「`SHOW PROCESSLIST` ステートメント」を参照してください。)

- `reload`

付与テーブルをリロードします。

- `refresh`

全テーブルをフラッシュし、ログファイルを閉じて、開きます。

- `shutdown`

サーバーを停止します。

- `start-slave`

レプリカサーバーでレプリケーションを開始します。

- `status`

短いサーバーステータスメッセージを表示します。

- `stop-slave`

レプリカサーバーでレプリケーションを停止します。

- `variables`

サーバーシステム変数とその値を表示します。

- `version`

サーバーからのバージョン情報を表示します。

すべてのコマンドは一意的なプリフィクスに省略できます。例:

```
shell> mysqladmin proc stat
```

```
+-----+-----+-----+-----+-----+-----+
| Id | User | Host | db | Command | Time | State | Info |
+-----+-----+-----+-----+-----+-----+
| 51 | jones | localhost | | Query | 0 | | show processlist |
+-----+-----+-----+-----+-----+-----+
Uptime: 1473624 Threads: 1 Questions: 39487
Slow queries: 0 Opens: 541 Flush tables: 1
Open tables: 19 Queries per second avg: 0.0268
```

mysqladmin status コマンドの結果は次の値を表示します。

- **Uptime**

MySQL サーバーが稼働している秒数。

- **Threads**

アクティブスレッド (クライアント) の数です。

- **Questions**

サーバーが起動して以来クライアントから寄せられた質問 (クエリー) の数です。

- **Slow queries**

`long_query_time` 秒よりも時間を要したクエリーの数。 [セクション5.4.5 「スロークエリーログ」](#) を参照してください。

- **Opens**

サーバーによって開かれたテーブルの数。

- **Flush tables**

サーバーが実行した `flush*`、`refresh`、および `reload` コマンドの数です。

- **Open tables**

現在開いているテーブルの数。

Unix ソケットファイルを使用してローカルサーバーに接続する際に `mysqladmin shutdown` を実行した場合、`mysqladmin` はサーバーのプロセス ID ファイルが取り除かれるまで待ちます。これはサーバーが正しく停止したことを確認するためです。

`mysqladmin` は次のオプションをサポートします。これらはコマンド行またはオプションファイルの `[mysqladmin]` グループおよび `[client]` グループで指定できます。MySQL プログラムによって使用されるオプションファイルの詳細については、[セクション4.2.2.2 「オプションファイルの使用」](#) を参照してください。

表 4.12 「mysqladmin のオプション」

オプション名	説明	導入	非推奨
<code>--bind-address</code>	指定されたネットワークインタフェースを使用して MySQL サーバーに接続		
<code>--compress</code>	クライアントとサーバー間で送信される情報をすべて圧縮		8.0.18
<code>--compression-algorithms</code>	サーバーへの接続に許可される圧縮アルゴリズム	8.0.18	
<code>--connect-timeout</code>	接続タイムアウトまでの秒数		

オプション名	説明	導入	非推奨
<code>--count</code>	繰り返されるコマンド実行での反復回数		
<code>--debug</code>	デバッグログの書込み		
<code>--debug-check</code>	プログラムの終了時にデバッグ情報を出カ		
<code>--debug-info</code>	プログラムの終了時に、デバッグ情報、メモリー、および CPU の統計を出カ		
<code>--default-auth</code>	使用する認証プラグイン		
<code>--default-character-set</code>	デフォルト文字セットを指定		
<code>--defaults-extra-file</code>	通常のオプションファイルに加えて、名前付きオプションファイルを読み取ります		
<code>--defaults-file</code>	指名されたオプションファイルのみを読み取る		
<code>--defaults-group-suffix</code>	オプショングループのサフィクス値		
<code>--enable-cleartext-plugin</code>	平文の認証プラグインを有効化		
<code>--force</code>	SQL エラーが発生しても続行		
<code>--get-server-public-key</code>	サーバーから RSA 公開キーをリクエスト		
<code>--help</code>	ヘルプメッセージを表示して終了		
<code>--host</code>	MySQL サーバーがあるホスト		
<code>--login-path</code>	ログインパスオプションを <code>.mylogin.cnf</code> から読み取り		
<code>--no-beep</code>	エラー時に音を発生させない		
<code>--no-defaults</code>	オプションファイルを読み取らない		
<code>--password</code>	サーバーに接続する際に使用するパスワード		
<code>--pipe</code>	名前付きパイプを使用してサーバに接続する (Windows のみ)		
<code>--plugin-dir</code>	プラグインがインストールされているディレクトリ		
<code>--port</code>	接続用の TCP/IP ポート番号		
<code>--print-defaults</code>	デフォルトオプションの印刷		
<code>--protocol</code>	使用するトランスポートプロトコル		

オプション名	説明	導入	非推奨
<code>--relative</code>	<code>--sleep</code> オプションとともに使用された場合、現在値と以前の値の差異を表示		
<code>--server-public-key-path</code>	RSA 公開鍵を含むファイルへのパス名		
<code>--shared-memory-base-name</code>	共有メモリー接続用の共有メモリー名 (Windows のみ)		
<code>--show-warnings</code>	ステートメント実行後に警告を表示		
<code>--shutdown-timeout</code>	サーバーのシャットダウンを待機する最大秒数		
<code>--silent</code>	サイレントモード		
<code>--sleep</code>	コマンドを反復実行し、その間に delay 秒間スリープ		
<code>--socket</code>	使用する Unix ソケットファイルまたは Windows 名前付きパイプ		
<code>--ssl-ca</code>	信頼できる SSL 認証局のリストを含むファイル		
<code>--ssl-capath</code>	信頼できる SSL 認証局の証明書ファイルを含むディレクトリ		
<code>--ssl-cert</code>	X.509 証明書を含むファイル		
<code>--ssl-cipher</code>	接続の暗号化に許可される暗号		
<code>--ssl-crl</code>	証明書失効リストを含むファイル		
<code>--ssl-crlpath</code>	証明書失効リストファイルを含むディレクトリ		
<code>--ssl-fips-mode</code>	クライアント側で FIPS モードを有効にするかどうか		
<code>--ssl-key</code>	X.509 キーを含むファイル		
<code>--ssl-mode</code>	サーバーへの接続に必要なセキュリティ状態		
<code>--tls-ciphersuites</code>	暗号化された接続に許可される TLSv1.3 暗号スイート	8.0.16	
<code>--tls-version</code>	暗号化された接続に許可される TLS プロトコル		
<code>--user</code>	サーバーへの接続時に使用する MySQL ユーザー名		
<code>--verbose</code>	冗長モード		
<code>--version</code>	バージョン情報を表示して終了		
<code>--vertical</code>	クエリー出力行を垂直形式で出力 (カラム値ごとに 1 行)		

オプション名	説明	導入	非推奨
<code>--wait</code>	接続が確立できない場合、中止せずに待機してからリトライ		
<code>--zstd-compression-level</code>	zstd 圧縮を使用するサーバーへの接続の圧縮レベル	8.0.18	

- `--help, -?`
ヘルプメッセージを表示して終了します。
- `--bind-address=ip_address`
複数のネットワークインタフェースを持つコンピュータで、このオプションを使用して、MySQL サーバーへの接続に使用するインタフェースを選択します。
- `--character-sets-dir=dir_name`
文字セットがインストールされているディレクトリ。 [セクション10.15「文字セットの構成」](#)を参照してください。
- `--compress, -C`
可能であれば、クライアントとサーバーの間で送信されるすべての情報を圧縮します。 [セクション4.2.8「接続圧縮制御」](#)を参照してください。
MySQL 8.0.18 では、このオプションは非推奨です。MySQL の将来のバージョンで削除されることが予想されます。 [レガシー接続圧縮の構成](#)を参照してください。
- `--compression-algorithms=value`
サーバーへの接続に許可される圧縮アルゴリズム。使用可能なアルゴリズムは、`protocol_compression_algorithms` システム変数の場合と同じです。デフォルト値は `uncompressed` です。
詳細は、 [セクション4.2.8「接続圧縮制御」](#)を参照してください。
このオプションは MySQL 8.0.18 で追加されました。
- `--connect-timeout=value`
接続タイムアウトまでの最大秒数。デフォルト値は 43200 秒 (12 時間) です。
- `--count=N, -c N`
`--sleep` オプションが指定されている場合、繰り返し実行されるコマンドの反復実行の数。
- `--debug[=debug_options], -# [debug_options]`
デバッグのログを書き込みます。一般的な `debug_options` 文字列は `d:t:o,file_name` です。デフォルトは `d:t:o,/tmp/mysqldadmin.trace` です。
このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。
- `--debug-check`
プログラムの終了時に、デバッグ情報を出力します。
このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。
- `--debug-info`
プログラムの終了時に、デバッグ情報とメモリーおよび CPU 使用率の統計を出力します。

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `--default-auth=plugin`

使用するクライアント側認証プラグインに関するヒント。 [セクション6.2.17「プラグブル認証」](#) を参照してください。

- `--default-character-set=charset_name`

`charset_name` をデフォルト文字セットとして使用します。 [セクション10.15「文字セットの構成」](#) を参照してください。

- `--defaults-extra-file=file_name`

このオプションファイルは、グローバルオプションファイルのあとに読み取りますが、(UNIX では) ユーザーオプションファイルの前に読み取るようにしてください。ファイルが存在しないかアクセスできない場合、エラーが発生します。`file_name` は、フルパス名でなく相対パス名として指定された場合、現行ディレクトリを基準にして解釈されます。

このオプションおよびその他のオプションファイルオプションの詳細は、 [セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--defaults-file=file_name`

指定されたオプションファイルのみ使用します。ファイルが存在しないかアクセスできない場合、エラーが発生します。`file_name` は、フルパス名でなく相対パス名として指定された場合、現行ディレクトリを基準にして解釈されます。

例外: `--defaults-file` でも、クライアントプログラムは `.mylogin.cnf` を読み取ります。

このオプションおよびその他のオプションファイルオプションの詳細は、 [セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--defaults-group-suffix=str`

通常のオプショングループだけでなく、通常の名前に `str` のサフィクスが付いたグループも読み取ります。たとえば、`mysqldadmin` は通常 `[client]` グループおよび `[mysqldadmin]` グループを読み取ります。`--defaults-group-suffix=other` オプションを指定した場合、`mysqldadmin` は `[client_other]` グループおよび `[mysqldadmin_other]` グループも読み取ります。

このオプションおよびその他のオプションファイルオプションの詳細は、 [セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--enable-cleartext-plugin`

`mysql_clear_password` 平文認証プラグインを有効にします。([セクション6.4.1.4「クライアント側クリアテキストプラグブル認証」](#) を参照してください。)

- `--force, -f`

`drop db_name` コマンドの確認を求めません。複数のコマンドで、エラーが発生しても続けます。

- `--get-server-public-key`

RSA キーペアベースのパスワード交換に必要な公開キーをサーバーにリクエストします。このオプションは、`caching_sha2_password` 認証プラグインで認証されるクライアントに適用されます。そのプラグインの場合、サーバーは要求されないかぎり公開鍵を送信しません。このオプションは、そのプラグインで認証されないアカウントでは無視されます。クライアントがセキュアな接続を使用してサーバーに接続する場合と同様に、RSA ベースのパスワード交換を使用しない場合も無視されます。

`--server-public-key-path=file_name` が指定され、有効な公開キーファイルが指定されている場合は、`--get-server-public-key` よりも優先されます。

`caching_sha2_password` プラグインの詳細は、[セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#) を参照してください。

- `--host=host_name, -h host_name`

指定されたホストの MySQL サーバーに接続します。

- `--login-path=name`

`.mylogin.cnf` ログインパスファイルの指定されたログインパスからオプションを読み取ります。「ログインパス」は、接続先の MySQL サーバーおよび認証に使用するアカウントを指定するオプションを含むオプショングループです。ログインパスファイルを作成または変更するには、`mysql_config_editor` ユーティリティを使用します。[セクション4.6.7「mysql_config_editor — MySQL 構成ユーティリティ」](#) を参照してください。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--no-beep, -b`

サーバーへの接続に失敗するなどのエラーの際にデフォルトで鳴らされる警告音を抑制します。

- `--no-defaults`

オプションファイルを読み取りません。オプションファイルから不明のオプションを読み取ることが原因でプログラムの起動に失敗する場合、`--no-defaults` を使用して、オプションを読み取らないようにすることができます。

例外として、`.mylogin.cnf` ファイルは、存在する場合はすべての場合に読み取られます。これにより、`--no-defaults` が使用された場合でも、コマンド行よりも安全な方法でパスワードを指定できます。`(.mylogin.cnf` は `mysql_config_editor` ユーティリティによって作成されます。[セクション4.6.7「mysql_config_editor — MySQL 構成ユーティリティ」](#) を参照してください)。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--password[=password], -p[password]`

サーバーへの接続に使用される MySQL アカウントのパスワード。パスワード値はオプションです。指定しない場合、`mysqladmin` によってプロンプトが表示されます。指定する場合は、`--password=` または `-p` とそれに続くパスワードの間にスペースなしが存在する必要があります。パスワードオプションを指定しない場合、デフォルトではパスワードは送信されません。

コマンド行でのパスワード指定は、セキュアでないと考えるべきです。コマンド行でパスワードを指定しないようにするには、オプションファイルを使用します。[セクション6.1.2.1「パスワードセキュリティのためのエンドユーザーガイドライン」](#) を参照してください。

パスワードがなく、`mysqladmin` でパスワードの入力を求められないように明示的に指定するには、`--skip-password` オプションを使用します。

- `--pipe, -W`

Windows で、名前付きパイプを使用してサーバーに接続します。このオプションは、ネームパイプ接続をサポートするために `named_pipe` システム変数を有効にしてサーバーを起動した場合にのみ適用されます。また、接続を行うユーザーは、`named_pipe_full_access_group` システム変数で指定された Windows グループのメンバーである必要があります。

- `--plugin-dir=dir_name`

プラグインを検索するディレクトリ。このオプションは、`--default-auth` オプションを使用して認証プラグインを指定しても、`mysqladmin` がそれを検出しない場合に指定します。[セクション6.2.17「プラガブル認証」](#) を参照してください。

- `--port=port_num, -P port_num`

TCP/IP 接続の場合、使用するポート番号。

- `--print-defaults`

プログラム名と、オプションファイルから受け取るすべてのオプションを出力します。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

サーバーへの接続に使用するトランスポートプロトコル。これは、他の接続パラメータが通常、必要なプロトコル以外のプロトコルを使用する場合に便利です。許可される値の詳細は、[セクション4.2.7「接続トランスポートプロトコル」](#)を参照してください。

- `--relative, -r`

`--sleep` オプションとともに使用された場合、現在値と以前の値の差異を表示します。このオプションは `extended-status` コマンドとのみ機能します。

- `--server-public-key-path=file_name`

RSA キーベースのパスワード交換のためにサーバーが必要とする公開キーのクライアント側コピーを含む、PEM 形式のファイルへのパス名。このオプションは、`sha256_password` または `caching_sha2_password` 認証プラグインで認証されるクライアントに適用されます。これらのプラグインのいずれかで認証されないアカウントでは、このオプションは無視されます。クライアントがセキュアな接続を使用してサーバーに接続する場合と同様に、RSA ベースのパスワード交換を使用しない場合も無視されます。

`--server-public-key-path=file_name` が指定され、有効な公開キーファイルが指定されている場合は、`--get-server-public-key` よりも優先されます。

`sha256_password` の場合、このオプションは、MySQL が OpenSSL を使用して構築された場合にのみ適用されません。

`sha256_password` および `caching_sha2_password` プラグインの詳細は、[セクション6.4.1.3「SHA-256 プラガブル認証」](#) および [セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#)を参照してください。

- `--shared-memory-base-name=name`

Windows の場合、共有メモリを使用してローカルサーバーに接続するために使用する共有メモリ名。デフォルト値は `MYSQL` です。共有メモリ名では大文字と小文字が区別されます。

このオプションは、共有メモリ接続をサポートするために `shared_memory` システム変数を有効にしてサーバーを起動した場合にのみ適用されます。

- `--show-warnings`

サーバーに送信されたステートメントの実行によって発生した警告を表示します。

- `--shutdown-timeout=value`

サーバーのシャットダウンを待機する最大秒数。デフォルト値は 3600 秒 (1 時間) です。

- `--silent, -s`

サーバーとの接続が確立できない場合、警告なしで終了します。

- `--sleep=delay, -i delay`

コマンドを繰り返し実行し、その間 `delay` 秒間スリープします。`--count` オプションは反復回数を決定します。`--count` が指定されていない場合は、`mysqldadmin` は中断されるまでいつまでもコマンドを実行します。

- `--socket=path, -S path`

`localhost` への接続用に使用する、Unix ソケットファイル、または Windows では使用する名前付きパイプの名前。

Windows では、このオプションは、名前付きパイプ接続をサポートするために `named_pipe` システム変数を有効にしてサーバーを起動した場合にのみ適用されます。また、接続を行うユーザーは、`named_pipe_full_access_group` システム変数で指定された Windows グループのメンバーである必要があります。

- `--ssl*`

`--ssl` で始まるオプションは、SSL を使用してサーバーに接続するかどうかを指定し、SSL 鍵および証明書を検索する場所を指定します。暗号化接続のコマンドオプションを参照してください。

- `--ssl-fips-mode={OFF|ON|STRICT}`

クライアント側で FIPS モードを有効にするかどうかを制御します。`--ssl-fips-mode` オプションは、暗号化された接続の確立には使用されず、許可する暗号化操作に影響する点で、他の `--ssl-xxx` オプションとは異なります。セクション6.8「FIPS のサポート」を参照してください。

次の `--ssl-fips-mode` 値を使用できます:

- `OFF`: FIPS モードを無効にします。
- `ON`: FIPS モードを有効にします。
- `STRICT`: 「strict」 FIPS モードを有効にします。

注記

OpenSSL FIPS オブジェクトモジュールが使用できない場合、`--ssl-fips-mode` に許可される値は `OFF` のみです。この場合、`--ssl-fips-mode` を `ON` または `STRICT` に設定すると、クライアントは起動時に警告を生成し、FIPS 以外のモードで動作します。

- `--tls-ciphersuites=ciphersuite_list`

TLSv1.3 を使用する暗号化された接続に許可される暗号スイート。値は、コロンで区切られた 1 つ以上の暗号スイート名のリストです。このオプションに指定できる暗号スイートは、MySQL のコンパイルに使用される SSL ライブラリによって異なります。詳細は、セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」を参照してください。

このオプションは MySQL 8.0.16 で追加されました。

- `--tls-version=protocol_list`

暗号化された接続に許可される TLS プロトコル。値は、1 つまたは複数のコンマ区切りプロトコル名のリストです。このオプションに指定できるプロトコルは、MySQL のコンパイルに使用される SSL ライブラリによって異なります。詳細は、セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」を参照してください。

- `--user=user_name, -u user_name`

サーバーへの接続に使用する MySQL アカウントのユーザー名。

- `--verbose, -v`

冗長モード。プログラムの動作についてより多くの情報を出力します。

- `--version, -V`

バージョン情報を表示して終了します。

- `--vertical, -E`

出力を縦に出力します。これは `--relative` と同様ですが、出力を縦に出力します。

- `--wait[=count], -w[count]`

接続が確立できない場合、中止せずに待機してからリトライします。 `count` 値が指定されている場合、リトライ回数を示します。デフォルトは 1 回です。

- `--zstd-compression-level=level`

`zstd` 圧縮アルゴリズムを使用するサーバーへの接続に使用する圧縮レベル。許可されるレベルは 1 から 22 で、大きい値は圧縮レベルの増加を示します。デフォルトの `zstd` 圧縮レベルは 3 です。圧縮レベルの設定は、`zstd` 圧縮を使用しない接続には影響しません。

詳細は、[セクション4.2.8「接続圧縮制御」](#)を参照してください。

このオプションは MySQL 8.0.18 で追加されました。

4.5.3 mysqlcheck — テーブル保守プログラム

`mysqlcheck` クライアントでは、テーブルの保守 (テーブルの検査、修復、最適化、分析) を実行します。

各テーブルは処理中にロックされるため、ほかのセッションでは利用できません。ただし、検査操作ではテーブルは `READ` ロックでのみロックされます (`READ` ロックおよび `WRITE` ロックの詳細は、[セクション13.3.6「LOCK TABLES および UNLOCK TABLES ステートメント」](#)を参照してください)。テーブルの保守処理は、特に大きなテーブルでは長い時間を要する可能性があります。 `--databases` オプションまたは `--all-databases` オプションを使用して 1 つまたは複数のデータベースに含まれるすべてのテーブルを処理する場合は、`mysqlcheck` の呼び出しに長い時間がかかる可能性があります。(これは、テーブルの処理方法が同じであるためにテーブルチェックが必要であると判断された場合に、MySQL のアップグレード手順にも当てはまります。)

`mysqlcheck` は、`mysqld` サーバーの実行中に使用する必要があります。つまり、テーブルのメンテナンスを実行するためにサーバーを停止する必要はありません。

`mysqlcheck` は SQL ステートメント `CHECK TABLE`、`REPAIR TABLE`、`ANALYZE TABLE`、および `OPTIMIZE TABLE` をユーザーにとって便利な方法で使用します。実行する操作に対してどのステートメントを使用するかを決定し、実行のためサーバーにステートメントを送信します。各ステートメントがどのストレージエンジンと機能するかは、[セクション13.7.3「テーブル保守ステートメント」](#)のステートメントの説明を参照してください。

すべてのストレージエンジンが必ずしも 4 つの保守操作をすべてサポートしているわけではありません。そのような場合、エラーメッセージが表示されます。たとえば、`test.t` が `MEMORY` テーブルの場合、これをチェックしようとすると、次の結果が生成されます:

```
shell> mysqlcheck test t
test.t
note : The storage engine for the table doesn't support check
```

`mysqlcheck` がテーブルを修復できない場合、手動でテーブルを修復する方法については[セクション2.11.13「テーブルまたはインデックスの再作成または修復」](#)を参照してください。これは、たとえば、`CHECK TABLE` ではチェックできますが、`REPAIR TABLE` では修復できない `InnoDB` テーブルの場合です。

注意

テーブルの修復操作を実行する前に、テーブルのバックアップを作成することをお勧めします。状況によっては、この操作のためにデータ損失が発生することがあります。考えられる原因としては、ファイルシステムのエラーなどがありますがこれに限りません。

一般的に、`mysqlcheck` を起動するには 3 つの方法があります。

```
shell> mysqlcheck [options] db_name [tbl_name ...]
shell> mysqlcheck [options] --databases db_name ...
shell> mysqlcheck [options] --all-databases
```

`db_name` のあとにテーブルを指定しない場合、または `--databases` オプションまたは `--all-databases` オプションを使用している場合、データベース全体が検査されます。

ほかのクライアントプログラムに比べ、`mysqlcheck` は特別な機能があります。テーブル検査のデフォルト動作 (`--check`) はバイナリの名前を変更することで変えられます。テーブルをデフォルトで修復するツールが必要な場合、`mysqlrepair` という名前で `mysqlcheck` のコピーを作成するか、`mysqlrepair` という名前で `mysqlcheck` へのシンボリックリンクを作成してください。`mysqlrepair` を起動すれば、テーブルが修復されます。

次の表に示す名前は、mysqlcheck のデフォルト動作を変更するために使用できます。

コマンド	意味
mysqlrepair	デフォルトオプションは <code>--repair</code>
mysqanalyze	デフォルトオプションは <code>--analyze</code>
mysqloptimize	デフォルトオプションは <code>--optimize</code>

mysqlcheck は次のオプションをサポートします。これらはコマンド行またはオプションファイルの [mysqlcheck] グループおよび [client] グループで指定できます。MySQL プログラムによって使用されるオプションファイルの詳細については、[セクション4.2.2.2「オプションファイルの使用」](#)を参照してください。

表 4.13 「mysqlcheck のオプション」

オプション名	説明	導入	非推奨
<code>--all-databases</code>	データベース内のテーブルをすべて確認		
<code>--all-in-1</code>	各データベースに対して、そのデータベースのすべてのテーブルを指定する単一のステートメントを実行		
<code>--analyze</code>	テーブルを分析します		
<code>--auto-repair</code>	確認されたテーブルが破損していた場合、自動的に修復		
<code>--bind-address</code>	指定されたネットワークインタフェースを使用して MySQL サーバーに接続		
<code>--character-sets-dir</code>	文字セットがインストールされているディレクトリ		
<code>--check</code>	テーブルにエラーがないか確認		
<code>--check-only-changed</code>	最後に行われた検査以降に変更されたテーブルのみをチェック		
<code>--check-upgrade</code>	CHECK TABLE を FOR UPGRADE オプションで呼び出し		
<code>--compress</code>	クライアントとサーバー間で送信される情報をすべて圧縮		8.0.18
<code>--compression-algorithms</code>	サーバーへの接続に許可される圧縮アルゴリズム	8.0.18	
<code>--databases</code>	すべての引数をデータベース名として解釈		
<code>--debug</code>	デバッグログの書き込み		
<code>--debug-check</code>	プログラムの終了時にデバッグ情報を出力		
<code>--debug-info</code>	プログラムの終了時に、デバッグ情報、メモリー、および CPU の統計を出力		
<code>--default-auth</code>	使用する認証プラグイン		

オプション名	説明	導入	非推奨
<code>--default-character-set</code>	デフォルト文字セットを指定		
<code>--defaults-extra-file</code>	通常のオプションファイルに加えて、名前付きオプションファイルを読み取ります		
<code>--defaults-file</code>	指名されたオプションファイルのみを読み取る		
<code>--defaults-group-suffix</code>	オプショングループのサフィクス値		
<code>--enable-cleartext-plugin</code>	平文の認証プラグインを有効化		
<code>--extended</code>	テーブルをチェックして修復		
<code>--fast</code>	正しく閉じられていないテーブルのみを確認		
<code>--force</code>	SQL エラーが発生しても続行		
<code>--get-server-public-key</code>	サーバーから RSA 公開キーをリクエスト		
<code>--help</code>	ヘルプメッセージを表示して終了		
<code>--host</code>	MySQL サーバーがあるホスト		
<code>--login-path</code>	ログインパスオプションを <code>.mylogin.cnf</code> から読み取り		
<code>--medium-check</code>	<code>--extended</code> 操作よりも速いチェックを実行		
<code>--no-defaults</code>	オプションファイルを読み取らない		
<code>--optimize</code>	テーブルを最適化します		
<code>--password</code>	サーバーに接続する際に使用するパスワード		
<code>--pipe</code>	名前付きパイプを使用してサーバに接続する (Windows のみ)		
<code>--plugin-dir</code>	プラグインがインストールされているディレクトリ		
<code>--port</code>	接続用の TCP/IP ポート番号		
<code>--print-defaults</code>	デフォルトオプションの印刷		
<code>--protocol</code>	使用するトランスポートプロトコル		
<code>--quick</code>	最速のチェック方法		
<code>--repair</code>	一意ではない一意なキー以外のほぼすべてを修正できる修復を実行		

オプション名	説明	導入	非推奨
<code>--server-public-key-path</code>	RSA 公開鍵を含むファイルへのパス名		
<code>--shared-memory-base-name</code>	共有メモリー接続用の共有メモリー名 (Windows のみ)		
<code>--silent</code>	サイレントモード		
<code>--skip-database</code>	実行される操作からこのデータベースを除外		
<code>--socket</code>	使用する Unix ソケットファイルまたは Windows 名前付きパイプ		
<code>--ssl-ca</code>	信頼できる SSL 認証局のリストを含むファイル		
<code>--ssl-capath</code>	信頼できる SSL 認証局の証明書ファイルを含むディレクトリ		
<code>--ssl-cert</code>	X.509 証明書を含むファイル		
<code>--ssl-cipher</code>	接続の暗号化に許可される暗号		
<code>--ssl-crl</code>	証明書失効リストを含むファイル		
<code>--ssl-crlpath</code>	証明書失効リストファイルを含むディレクトリ		
<code>--ssl-fips-mode</code>	クライアント側で FIPS モードを有効にするかどうか		
<code>--ssl-key</code>	X.509 キーを含むファイル		
<code>--ssl-mode</code>	サーバーへの接続に必要なセキュリティ状態		
<code>--tables</code>	<code>--databases</code> オプションまたは <code>-B</code> オプションをオーバーライド		
<code>--tls-ciphersuites</code>	暗号化された接続に許可される TLSv1.3 暗号スイート	8.0.16	
<code>--tls-version</code>	暗号化された接続に許可される TLS プロトコル		
<code>--use-frm</code>	MyISAM テーブルの修復操作		
<code>--user</code>	サーバーへの接続時に使用する MySQL ユーザー名		
<code>--verbose</code>	冗長モード		
<code>--version</code>	バージョン情報を表示して終了		
<code>--write-binlog</code>	ANALYZE ステートメント、OPTIMIZE ステートメント、REPAIR ステートメントをバイナリログに記録。 <code>--skip-write-binlog</code> は、NO_WRITE_TO_BINLOG		

オプション名	説明	導入	非推奨
	をこれらのステートメントに追加します		
<code>--zstd-compression-level</code>	zstd 圧縮を使用するサーバーへの接続の圧縮レベル	8.0.18	

- `--help, -?`

ヘルプメッセージを表示して終了します。

- `--all-databases, -A`

データベース内のテーブルをすべて検査します。これは、`INFORMATION_SCHEMA` および `performance_schema` データベースがチェックされないことを除き、`--databases` オプションを使用し、コマンドラインですべてのデータベースに名前を付けることと同じです。これらは、`--databases` オプションを使用して明示的に名前を付けることでチェックできます。

- `--all-in-1, -1`

各テーブルに対してステートメントを発行する代わりに、各データベースに対して、そのデータベースから処理されるすべてのテーブルを指名する単一のステートメントを実行します。

- `--analyze, -a`

テーブルを分析します。

- `--auto-repair`

確認されたテーブルが壊れていた場合、自動的に修復します。必要な修復はすべてのテーブルが確認されたあとに実行されます。

- `--bind-address=ip_address`

複数のネットワークインターフェースを持つコンピュータで、このオプションを使用して、MySQL サーバーへの接続に使用するインターフェースを選択します。

- `--character-sets-dir=dir_name`

文字セットがインストールされているディレクトリ。セクション10.15「文字セットの構成」を参照してください。

- `--check, -c`

テーブルにエラーがないか確認します。これはデフォルトの動作です。

- `--check-only-changed, -C`

最後に行われた検査以降に変更されたテーブル、または適切に閉じられなかったテーブルのみを検査します。

- `--check-upgrade, -g`

`CHECK TABLE` を `FOR UPGRADE` オプションで呼び出し、サーバーの現在のバージョンとの互換性のないテーブルがあるか検査します。

- `--compress`

可能であれば、クライアントとサーバーの間で送信されるすべての情報を圧縮します。セクション4.2.8「接続圧縮制御」を参照してください。

MySQL 8.0.18 では、このオプションは非推奨です。MySQL の将来のバージョンで削除されることが予想されます。レガシー接続圧縮の構成を参照してください。

- `--compression-algorithms=value`

サーバーへの接続に許可される圧縮アルゴリズム。使用可能なアルゴリズムは、[protocol_compression_algorithms](#) システム変数の場合と同じです。デフォルト値は `uncompressed` です。

詳細は、[セクション4.2.8「接続圧縮制御」](#)を参照してください。

このオプションは MySQL 8.0.18 で追加されました。

- `--databases, -B`

指定されたデータベース内のテーブルをすべて処理します。通常、`mysqlcheck` では、コマンドラインの名引数はデータベース名として扱われ、後続の名前はテーブル名として扱われます。このオプションを使用すると、名前引数をすべてデータベース名として処理します。

- `--debug=[debug_options], -# [debug_options]`

デバッグのログを書き込みます。一般的な `debug_options` 文字列は `d:t:o,file_name` です。デフォルトは `d:t:o` です。

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `--debug-check`

プログラムの終了時に、デバッグ情報を出力します。

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `--debug-info`

プログラムの終了時に、デバッグ情報とメモリーおよび CPU 使用率の統計を出力します。

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `--default-character-set=charset_name`

`charset_name` をデフォルト文字セットとして使用します。[セクション10.15「文字セットの構成」](#)を参照してください。

- `--defaults-extra-file=file_name`

このオプションファイルは、グローバルオプションファイルのあとに読み取りますが、(UNIX では) ユーザーオプションファイルの前に読み取るようにしてください。ファイルが存在しないかアクセスできない場合、エラーが発生します。`file_name` は、フルパス名でなく相対パス名として指定された場合、現行ディレクトリを基準にして解釈されます。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--defaults-file=file_name`

指定されたオプションファイルのみ使用します。ファイルが存在しないかアクセスできない場合、エラーが発生します。`file_name` は、フルパス名でなく相対パス名として指定された場合、現行ディレクトリを基準にして解釈されます。

例外: `--defaults-file` でも、クライアントプログラムは `.mylogin.cnf` を読み取ります。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--defaults-group-suffix=str`

通常のオプショングループだけでなく、通常の名前に `str` のサフィクスが付いたグループも読み取ります。たとえば、`mysqlcheck` は通常 `[client]` グループおよび `[mysqlcheck]` グループを読み取ります。`--defaults-group-suffix=_other` オプションを指定した場合、`mysqlcheck` は `[client_other]` グループおよび `[mysqlcheck_other]` グループも読み取ります。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--extended, -e`

テーブルの検査にこのオプションを使用している場合、100% 整合性があることが保証されますが、時間がかかります。

このオプションを使用してテーブルを修復している場合、修復作業に時間がかかるだけでなく、大量のガベージ行を生成することもあります。

- `--default-auth=plugin`

使用するクライアント側認証プラグインに関するヒント。[セクション6.2.17「プラグブル認証」](#) を参照してください。

- `--enable-cleartext-plugin`

`mysql_clear_password` 平文認証プラグインを有効にします。([セクション6.4.1.4「クライアント側クリアテキストプラグブル認証」](#) を参照してください。)

- `--fast, -F`

正しく閉じられていないテーブルのみを検査します。

- `--force, -f`

SQL エラーが発生しても続行します。

- `--get-server-public-key`

RSA キーペアベースのパスワード交換に必要な公開キーをサーバーにリクエストします。このオプションは、`caching_sha2_password` 認証プラグインで認証されるクライアントに適用されます。そのプラグインの場合、サーバーは要求されないかぎり公開鍵を送信しません。このオプションは、そのプラグインで認証されないアカウントでは無視されます。クライアントがセキュアな接続を使用してサーバーに接続する場合と同様に、RSA ベースのパスワード交換を使用しない場合も無視されます。

`--server-public-key-path=file_name` が指定され、有効な公開キーファイルが指定されている場合は、`--get-server-public-key` よりも優先されます。

`caching_sha2_password` プラグインの詳細は、[セクション6.4.1.2「SHA-2 プラグブル認証のキャッシュ」](#) を参照してください。

- `--host=host_name, -h host_name`

指定されたホストの MySQL サーバーに接続します。

- `--login-path=name`

`.mylogin.cnf` ログインパスファイルの指定されたログインパスからオプションを読み取ります。「`「ログインパス」`」は、接続先の MySQL サーバーおよび認証に使用するアカウントを指定するオプションを含むオプショングループです。ログインパスファイルを作成または変更するには、`mysql_config_editor` ユーティリティを使用します。[セクション4.6.7「mysql_config_editor — MySQL 構成ユーティリティー」](#) を参照してください。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--medium-check, -m`

`--extended` 操作よりも高速な検査を実行します。これはすべてのエラーの 99.99% のみを確認し、ほとんどの場合はこれで十分でしょう。

- `--no-defaults`

オプションファイルを読み取りません。オプションファイルから不明のオプションを読み取ることが原因でプログラムの起動に失敗する場合、`--no-defaults` を使用して、オプションを読み取らないようにできます。

例外として、`.mylogin.cnf` ファイルは、存在する場合はすべての場合に読み取られます。これにより、`--no-defaults` が使用された場合にも、コマンド行よりも安全な方法でパスワードを指定できます。`(.mylogin.cnf` は `mysql_config_editor` ユーティリティーによって作成されます。 [セクション4.6.7「mysql_config_editor — MySQL 構成ユーティリティー」](#) を参照してください)。

このオプションおよびその他のオプションファイルオプションの詳細は、 [セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--optimize, -o`

テーブルを最適化します。

- `--password[=password], -p[password]`

サーバーへの接続に使用される MySQL アカウントのパスワード。パスワード値はオプションです。指定しない場合、`mysqlcheck` によってプロンプトが表示されます。指定する場合は、`--password=` または `-p` とそれに続くパスワードの間にスペースなしが存在する必要があります。パスワードオプションを指定しない場合、デフォルトではパスワードは送信されません。

コマンド行でのパスワード指定は、セキュアでないと考えるべきです。コマンド行でパスワードを指定しないようにするには、オプションファイルを使用します。 [セクション6.1.2.1「パスワードセキュリティのためのエンドユーザーガイドライン」](#) を参照してください。

パスワードがなく、`mysqlcheck` でパスワードの入力を求められないように明示的に指定するには、`--skip-password` オプションを使用します。

- `--pipe, -W`

Windows で、名前付きパイプを使用してサーバーに接続します。このオプションは、ネームパイプ接続をサポートするために `named_pipe` システム変数を有効にしてサーバーを起動した場合にのみ適用されます。また、接続を行うユーザーは、`named_pipe_full_access_group` システム変数で指定された Windows グループのメンバーである必要があります。

- `--plugin-dir=dir_name`

プラグインを検索するディレクトリ。このオプションは、`--default-auth` オプションを使用して認証プラグインを指定しても、`mysqlcheck` がそれを検出しない場合に指定します。 [セクション6.2.17「プラグイン認証」](#) を参照してください。

- `--port=port_num, -P port_num`

TCP/IP 接続の場合、使用するポート番号。

- `--print-defaults`

プログラム名と、オプションファイルから受け取るすべてのオプションを出力します。

このオプションおよびその他のオプションファイルオプションの詳細は、 [セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

サーバーへの接続に使用するトランスポートプロトコル。これは、他の接続パラメータが通常、必要なプロトコル以外のプロトコルを使用する場合に便利です。許可される値の詳細は、 [セクション4.2.7「接続トランスポートプロトコル」](#) を参照してください。

- `--quick, -q`

このオプションを使用してテーブルを検査している場合、正しくないリンクを検査するために行のスキャンを行いません。これが最速の検査方法です。

このオプションを使用してテーブルを修復している場合、インデックスツリーのみの修復を試みます。これが最速の修復方法です。

- `--repair, -r`

一意ではないユニークキー以外のすべてを修正できる修復を実行します。

- `--server-public-key-path=file_name`

RSA キーペアベースのパスワード交換のためにサーバーが必要とする公開キーのクライアント側コピーを含む、PEM 形式のファイルへのパス名。このオプションは、`sha256_password` または `caching_sha2_password` 認証プラグインで認証されるクライアントに適用されます。これらのプラグインのいずれかで認証されないアカウントでは、このオプションは無視されます。クライアントがセキュアな接続を使用してサーバーに接続する場合と同様に、RSA ベースのパスワード交換を使用しない場合も無視されます。

`--server-public-key-path=file_name` が指定され、有効な公開キーファイルが指定されている場合は、`--get-server-public-key` よりも優先されます。

`sha256_password` の場合、このオプションは、MySQL が OpenSSL を使用して構築された場合にのみ適用されません。

`sha256_password` および `caching_sha2_password` プラグインの詳細は、[セクション6.4.1.3「SHA-256 プラガブル認証」](#) および [セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#) を参照してください。

- `--shared-memory-base-name=name`

Windows の場合、共有メモリを使用してローカルサーバーに接続するために使用する共有メモリ名。デフォルト値は `MYSQL` です。共有メモリー名では大文字と小文字が区別されます。

このオプションは、共有メモリー接続をサポートするために `shared_memory` システム変数を有効にしてサーバーを起動した場合にのみ適用されます。

- `--silent, -s`

サイレントモード。エラーメッセージのみを出力します。

- `--skip-database=db_name`

`mysqlcheck` によって実行される操作には、名前付きデータベース (大/小文字を区別) を含めないでください。

- `--socket=path, -S path`

`localhost` への接続用に使用する、Unix ソケットファイル、または Windows では使用する名前付きパイプの名前。

Windows では、このオプションは、名前付きパイプ接続をサポートするために `named_pipe` システム変数を有効にしてサーバーを起動した場合にのみ適用されます。また、接続を行うユーザーは、`named_pipe_full_access_group` システム変数で指定された Windows グループのメンバーである必要があります。

- `--ssl*`

`--ssl` で始まるオプションは、SSL を使用してサーバーに接続するかどうかを指定し、SSL 鍵および証明書を検索する場所を指定します。 [暗号化接続のコマンドオプション](#) を参照してください。

- `--ssl-fips-mode={OFF|ON|STRICT}`

クライアント側で FIPS モードを有効にするかどうかを制御します。 `--ssl-fips-mode` オプションは、暗号化された接続の確立には使用されず、許可する暗号化操作に影響する点で、他の `--ssl-xxx` オプションとは異なります。 [セクション6.8「FIPS のサポート」](#) を参照してください。

次の `--ssl-fips-mode` 値を使用できます:

- **OFF**: FIPS モードを無効にします。
- **ON**: FIPS モードを有効にします。
- **STRICT**: 「strict」 FIPS モードを有効にします。

注記

OpenSSL FIPS オブジェクトモジュールが使用できない場合、`--ssl-fips-mode` に許可される値は **OFF** のみです。この場合、`--ssl-fips-mode` を **ON** または **STRICT** に設定すると、クライアントは起動時に警告を生成し、FIPS 以外のモードで動作します。

- `--tables`

`--databases` オプションまたは `-B` オプションをおオーバーライドします。オプションに続くすべての名前引数はテーブル名とみなされます。

- `--tls-ciphersuites=ciphersuite_list`

TLSv1.3 を使用する暗号化された接続に許可される暗号スイート。値は、コロンで区切られた 1 つ以上の暗号スイート名のリストです。このオプションに指定できる暗号スイートは、MySQL のコンパイルに使用される SSL ライブラリによって異なります。詳細は、[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#) を参照してください。

このオプションは MySQL 8.0.16 で追加されました。

- `--tls-version=protocol_list`

暗号化された接続に許可される TLS プロトコル。値は、1 つまたは複数のコンマ区切りプロトコル名のリストです。このオプションに指定できるプロトコルは、MySQL のコンパイルに使用される SSL ライブラリによって異なります。詳細は、[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#) を参照してください。

- `--use-frm`

MyISAM テーブルに対する修復操作の場合は、`.MYI` ヘッダーが破損していてもテーブルを修復できるように、データディクショナリからテーブル構造を取得します。

- `--user=user_name, -u user_name`

サーバーへの接続に使用する MySQL アカウントのユーザー名。

- `--verbose, -v`

冗長モード。プログラム処理のさまざまな段階についての情報を出力します。

- `--version, -V`

バージョン情報を表示して終了します。

- `--write-binlog`

このオプションはデフォルトで有効で、`mysqlcheck` によって生成される `ANALYZE TABLE`、`OPTIMIZE TABLE`、および `REPAIR TABLE` の各ステートメントがバイナリログに書き込まれます。`--skip-write-binlog` を使用すると、ステートメントに `NO_WRITE_TO_BINLOG` が追加され、ログに記録されなくなります。これらのステートメントをレプリカに送信したり、バイナリログをバックアップからの回復に使用するときを実行したりしない場合は、`--skip-write-binlog` を使用します。

- `--zstd-compression-level=level`

`zstd` 圧縮アルゴリズムを使用するサーバーへの接続に使用する圧縮レベル。許可されるレベルは 1 から 22 で、大きい値は圧縮レベルの増加を示します。デフォルトの `zstd` 圧縮レベルは 3 です。圧縮レベルの設定は、`zstd` 圧縮を使用しない接続には影響しません。

詳細は、[セクション4.2.8「接続圧縮制御」](#) を参照してください。

このオプションは MySQL 8.0.18 で追加されました。

4.5.4 mysqldump — データベースバックアッププログラム

`mysqldump` クライアントユーティリティは [logical backups](#) を実行し、元のデータベースオブジェクト定義およびテーブルデータを再現するために実行できる一連の SQL ステートメントを生成します。別の SQL サーバーにバックアップまたは転送するために、1 つ以上の MySQL データベースをダンプします。`mysqldump` コマンドは、CSV、その他の区切り文字で区切られたテキスト、または XML 形式でも出力を生成できます。

ヒント

複数のスレッド、ファイル圧縮、進捗情報の表示、および Oracle Cloud Infrastructure Object Storage ストリーミングや MySQL データベースサービス 互換性チェックおよび変更などのクラウド機能で並列ダンプを提供する [MySQL Shell dump utilities](#) の使用を検討してください。ダンプは、[MySQL Shell load dump utilities](#) を使用して MySQL Server インスタンスまたは MySQL データベースサービス DB システムに簡単にインポートできます。MySQL Shell のインストール手順は、[here](#) にあります。

- [パフォーマンスおよびスケーラビリティに関する考慮事項](#)
- [起動構文](#)
- [オプション構文 - アルファベット順のサマリー](#)
- [接続オプション](#)
- [オプションファイルオプション](#)
- [DDL オプション](#)
- [デバッグオプション](#)
- [ヘルプオプション](#)
- [国際化オプション](#)
- [レプリケーションオプション](#)
- [形式オプション](#)
- [フィルタリングオプション](#)
- [パフォーマンスオプション](#)
- [トランザクションオプション](#)
- [オプショングループ](#)
- [例](#)
- [制約](#)

`mysqldump` には、ダンプされたテーブルに対する `SELECT` 以上の権限、ダンプされたビューに対する `SHOW VIEW`、ダンプされたトリガーに対する `TRIGGER`、`--single-transaction` オプションが使用されていない場合 `LOCK TABLES`、および (MySQL 8.0.21 時点で)`--no-tablespaces` オプションが使用されなければ、`PROCESS` が必要です。オプションの説明に示すように、一部のオプションではその他の権限が必要な場合があります。

ダンプファイルをリロードするには、ダンプファイルに含まれているステートメントを実行するために必要な権限 (これらのステートメントによって作成されたオブジェクトに対する適切な `CREATE` 権限など) が必要です。

`mysqldump` 出力には、データベースの照合順序を変更する `ALTER DATABASE` ステートメントを含めることができます。これらは、ストアプログラムをダンプする際に文字のエンコードを維持するために使用できます。このようなステートメントを含むダンプファイルをリロードするには、影響されるデータベースに対する `ALTER` 権限が必要です。

注記

Windows で出力リダイレクトを使用して PowerShell を使用して作成されたダンプは、UTF-16 エンコーディングを持つファイルを作成します:

```
shell> mysqldump [options] > dump.sql
```

ただし、UTF-16 は接続文字セットとして許可されていないため ([許可されていないクライアント文字セット](#) を参照)、ダンプファイルを正しくロードできません。この問題を回避するには、ASCII 形式で出力を作成する `--result-file` オプションを使用します:

```
shell> mysqldump [options] --result-file=dump.sql
```

パフォーマンスおよびスケーラビリティに関する考慮事項

`mysqldump` の利点には、リストアする前に出力を表示して編集もできるという便利さと柔軟性があります。開発およびデータベース管理用にデータベースのクローンを作成したり、テスト用に既存のデータベースとわずかに異なるデータベースを作成したりできます。大量のデータのバックアップのための、高速でスケーラブルなソリューションを意図したものではありません。データサイズが大量の場合、バックアップのステップにかかる時間が妥当だとしても、SQL ステートメントの再現には、挿入やインデックスの作成などのディスク I/O が含まれるため、データのリストアに非常に長い時間がかかることがあります。

大規模なバックアップとリストアでは、データファイルを高速でリストアできる元の形式でコピーする、[物理バックアップ](#)の方が適切です。

- テーブルが主に [InnoDB](#) テーブルである場合、または [InnoDB](#) テーブルと [MyISAM](#) テーブルが混在する場合は、MySQL Enterprise Backup 製品の `mysqlbackup` コマンドを使用することを検討してください。(Enterprise サブスクリプションの一部として含まれています。) これにより、最低限の中断でもっともパフォーマンスのよい [InnoDB](#) のバックアップを実行できます。また、[MyISAM](#) およびその他のストレージエンジンからのテーブルもバックアップでき、さまざまなバックアップシナリオに対応するための便利なオプションを多数提供します。[セクション 30.2 「MySQL Enterprise Backup の概要」](#) を参照してください。

`mysqldump` は、テーブルの内容を 1 行ずつ取得してダンプすることも、ダンプする前にテーブルからすべての内容を取得して、メモリーにバッファリングすることもできます。大きなテーブルをダンプしている場合、メモリーへのバッファリングが問題になる場合があります。テーブルを 1 行ずつダンプする場合、`--quick` オプションを使用してください (または `--opt` を指定すれば `--quick` が有効になります)。 `--opt` オプションは (したがって `--quick` も) デフォルトで有効なため、メモリーへのバッファリングを有効にするには、`--skip-quick` を使用します。

最近のバージョンの `mysqldump` を使用して、非常に古い MySQL サーバーにリロードされるダンプを生成する場合は、`--opt` オプションまたは `--extended-insert` オプションの代わりに `--skip-opt` オプションを使用します。

`mysqldump` の詳細は、[セクション 7.4 「バックアップへの mysqldump の使用」](#) を参照してください。

起動構文

次に示すように、一般に `mysqldump` を使用するには、1 つまたは複数のテーブルのセットのダンプ、1 つまたは複数の完全なデータベースのセット、または MySQL サーバー全体の 3 つの方法があります。

```
shell> mysqldump [options] db_name [tbl_name ...]
shell> mysqldump [options] --databases db_name ...
shell> mysqldump [options] --all-databases
```

データベース全体をダンプするには、`db_name` に続けてテーブルを指名しないか、または `--databases` オプションまたは `--all-databases` オプションを使用します。

使用しているバージョンの `mysqldump` がサポートするオプションのリストを表示するには、コマンド `mysqldump --help` を発行します。

オプション構文 - アルファベット順のサマリー

`mysqldump` は次のオプションをサポートします。これらはコマンド行またはオプションファイルの `[mysqldump]` グループおよび `[client]` グループで指定できます。MySQL プログラムによって使用されるオプションファイルの詳細については、[セクション 4.2.2.2 「オプションファイルの使用」](#) を参照してください。

表 4.14 「mysqldump のオプション」

オプション名	説明	導入	非推奨
<code>--add-drop-database</code>	DROP DATABASE ステートメントを各 CREATE DATABASE ステートメントの前に追加		
<code>--add-drop-table</code>	各 CREATE TABLE ステートメントの前に DROP TABLE ステートメントを追加		
<code>--add-drop-trigger</code>	DROP TRIGGER ステートメントを各 CREATE TRIGGER ステートメントの前に追加		
<code>--add-locks</code>	LOCK TABLES と UNLOCK TABLES ステートメントで各テーブルダンプを囲む		
<code>--all-databases</code>	すべてのデータベース内のすべてのテーブルをダンプ		
<code>--allow-keywords</code>	キーワードであるカラム名の作成を許可		
<code>--apply-slave-statements</code>	CHANGE MASTER ステートメントの前に STOP SLAVE を含め、START SLAVE を出力の最後に含める		
<code>--bind-address</code>	指定されたネットワークインタフェースを使用して MySQL サーバーに接続		
<code>--character-sets-dir</code>	文字セットがインストールされているディレクトリ		
<code>--column-statistics</code>	ANALYZE TABLE ステートメントを記述して統計ヒストグラムを生成		
<code>--comments</code>	ダンプファイルへのコメントの追加		
<code>--compact</code>	よりコンパクトな出力を生成		
<code>--compatible</code>	古い MySQL サーバーやほかのデータベースシステムとの互換性がより高い出力を生成		
<code>--complete-insert</code>	カラム名を含む完全な INSERT ステートメントを使用		
<code>--compress</code>	クライアントとサーバー間で送信される情報をすべて圧縮		8.0.18
<code>--compression-algorithms</code>	サーバーへの接続に許可される圧縮アルゴリズム	8.0.18	
<code>--create-options</code>	すべての MySQL に固有なテーブルオプションを		

オプション名	説明	導入	非推奨
	CREATE TABLE ステートメントに含める		
<code>--databases</code>	すべての名前引数をデータベース名として解釈		
<code>--debug</code>	デバッグログの書込み		
<code>--debug-check</code>	プログラムの終了時にデバッグ情報を出力		
<code>--debug-info</code>	プログラムの終了時に、デバッグ情報、メモリー、および CPU の統計を出力		
<code>--default-auth</code>	使用する認証プラグイン		
<code>--default-character-set</code>	デフォルト文字セットを指定		
<code>--defaults-extra-file</code>	通常のオプションファイルに加えて、名前付きオプションファイルを読み取ります		
<code>--defaults-file</code>	指名されたオプションファイルのみを読み取る		
<code>--defaults-group-suffix</code>	オプショングループのサフィクス値		
<code>--delete-master-logs</code>	マスターレプリケーションサーバーで、ダンプ操作の実行後にバイナリログを削除		
<code>--disable-keys</code>	テーブルごとに、INSERT ステートメントを無効化してキーを有効化するステートメントで囲みます		
<code>--dump-date</code>	<code>--comments</code> が指定された場合、ダンプ日を "Dump completed on" コメントとして含める		
<code>--dump-slave</code>	スレーブのマスターのバイナリログ座標をリストする CHANGE MASTER ステートメントを含める		
<code>--enable-cleartext-plugin</code>	平文の認証プラグインを有効化		
<code>--events</code>	ダンプされたデータベースからのイベントのダンプ		
<code>--extended-insert</code>	複数行 INSERT 構文の使用		
<code>--fields-enclosed-by</code>	このオプションは <code>--tab</code> オプションとともに使用され、LOAD DATA の対応する句と同じ意味を持ちます		
<code>--fields-escaped-by</code>	このオプションは <code>--tab</code> オプションとともに使用され、LOAD DATA の対応する句と同じ意味を持ちます		

オプション名	説明	導入	非推奨
<code>--fields-optionally-enclosed-by</code>	このオプションは <code>--tab</code> オプションとともに使用され、LOAD DATA の対応する句と同じ意味を持ちます		
<code>--fields-terminated-by</code>	このオプションは <code>--tab</code> オプションとともに使用され、LOAD DATA の対応する句と同じ意味を持ちます		
<code>--flush-logs</code>	ダンプを開始する前に MySQL サーバーログファイルをフラッシュ		
<code>--flush-privileges</code>	mysql データベースのダンプ後に FLUSH PRIVILEGES ステートメントを発行		
<code>--force</code>	テーブルダンプの最中に SQL エラーが発生しても続行します		
<code>--get-server-public-key</code>	サーバーから RSA 公開キーをリクエスト		
<code>--help</code>	ヘルプメッセージを表示して終了		
<code>--hex-blob</code>	16 進数表記法を使用したバイナリカラムのダンプ		
<code>--host</code>	MySQL サーバーがあるホスト		
<code>--ignore-error</code>	指定されたエラーを無視		
<code>--ignore-table</code>	指定されたテーブルをダンプしない		
<code>--include-master-host-port</code>	<code>--dump-slave</code> とともに生成された CHANGE MASTER ステートメントに MASTER_HOST/MASTER_PORT オプションを含める		
<code>--insert-ignore</code>	INSERT ステートメントではなく INSERT IGNORE を書き込みます		
<code>--lines-terminated-by</code>	このオプションは <code>--tab</code> オプションとともに使用され、LOAD DATA の対応する句と同じ意味を持ちます		
<code>--lock-all-tables</code>	データベース内のテーブルをすべてロック		
<code>--lock-tables</code>	テーブルをダンプする前にすべてロック		
<code>--log-error</code>	指定されたファイルに警告およびエラーを追加		
<code>--login-path</code>	ログインパスオプションを <code>.mylogin.cnf</code> から読み取り		

オプション名	説明	導入	非推奨
<code>--master-data</code>	バイナリログファイルの名前と場所を出力に書き込む		
<code>--max-allowed-packet</code>	サーバーとの間で送受信するバケットの最大長		
<code>--net-buffer-length</code>	TCP/IP とソケット通信のバッファサイズ		
<code>--network-timeout</code>	ネットワークタイムアウトを増やして、より大きなテーブルダンプを許可		
<code>--no-autocommit</code>	ダンプされたテーブルごとに、INSERT ステートメントを SET autocommit = 0 ステートメントと COMMIT ステートメントで囲む		
<code>--no-create-db</code>	CREATE DATABASE ステートメントを記述しないでください		
<code>--no-create-info</code>	各ダンプされたテーブルを再作成する CREATE TABLE ステートメントを書き出さない		
<code>--no-data</code>	テーブルの内容をダンプしない		
<code>--no-defaults</code>	オプションファイルを読み取らない		
<code>--no-set-names</code>	<code>--skip-set-charset</code> と同じ		
<code>--no-tablespaces</code>	CREATE LOGFILE GROUP ステートメントおよび CREATE TABLESPACE ステートメントを出力に書き出さない		
<code>--opt</code>	<code>--add-drop-table</code> <code>--add-locks</code> <code>--create-options</code> <code>--disable-keys</code> <code>--extended-insert</code> <code>--lock-tables</code> <code>--quick</code> <code>--set-charset</code> の短縮形		
<code>--order-by-primary</code>	各テーブルの行を、主キーまたは最初の一意のインデックスでソートしてダンプ		
<code>--password</code>	サーバーに接続する際に使用するパスワード		
<code>--pipe</code>	名前付きパイプを使用してサーバに接続する (Windows のみ)		
<code>--plugin-dir</code>	プラグインがインストールされているディレクトリ		
<code>--port</code>	接続用の TCP/IP ポート番号		
<code>--print-defaults</code>	デフォルトオプションの印刷		

オプション名	説明	導入	非推奨
<code>--protocol</code>	使用するトランスポートプロトコル		
<code>--quick</code>	サーバーからのテーブルについて、一度に 1 行ずつ取得		
<code>--quote-names</code>	識別子を逆引用符文字で囲む		
<code>--replace</code>	INSERT ステートメントではなく REPLACE ステートメントを書き出す		
<code>--result-file</code>	指定されたファイルに出力		
<code>--routines</code>	ダンプされたデータベースからストアドルーチン (プロシージャとファンクション) をダンプ		
<code>--server-public-key-path</code>	RSA 公開鍵を含むファイルへのパス名		
<code>--set-charset</code>	SET NAMES default_character_set を出力に追加		
<code>--set-gtid-purged</code>	SET @@GLOBAL.GTID_PURGED を出力に追加するかどうか		
<code>--shared-memory-base-name</code>	共有メモリー接続用の共有メモリー名 (Windows のみ)		
<code>--show-create-skip-secondary-engine</code>	CREATE TABLE ステートメントから SECONDARY ENGINE 句を除外	8.0.18	
<code>--single-transaction</code>	サーバーからデータをダンプする前に BEGIN SQL ステートメントを発行してください		
<code>--skip-add-drop-table</code>	DROP TABLE ステートメントを CREATE TABLE ステートメントの前に追加しない		
<code>--skip-add-locks</code>	ロックを追加しない		
<code>--skip-comments</code>	ダンプファイルにコメントを追加しない		
<code>--skip-compact</code>	よりコンパクトな出力を生成しない		
<code>--skip-disable-keys</code>	キーを無効にしない		
<code>--skip-extended-insert</code>	extended-insert をオフにする		
<code>--skip-opt</code>	--opt で設定されたオプションをオフにします		
<code>--skip-quick</code>	サーバーからのテーブルについて、一度に 1 行ずつ取得しない		

オプション名	説明	導入	非推奨
<code>--skip-quote-names</code>	識別子を引用符で囲まない		
<code>--skip-set-charset</code>	SET NAMES ステートメントを記述しないでください		
<code>--skip-triggers</code>	トリガーをダンプしない		
<code>--skip-tz-utc</code>	tz-utc をオフにする		
<code>--socket</code>	使用する Unix ソケットファイルまたは Windows 名前付きパイプ		
<code>--ssl-ca</code>	信頼できる SSL 認証局のリストを含むファイル		
<code>--ssl-capath</code>	信頼できる SSL 認証局の証明書ファイルを含むディレクトリ		
<code>--ssl-cert</code>	X.509 証明書を含むファイル		
<code>--ssl-cipher</code>	接続の暗号化に許可される暗号		
<code>--ssl-crl</code>	証明書失効リストを含むファイル		
<code>--ssl-crlpath</code>	証明書失効リストファイルを含むディレクトリ		
<code>--ssl-fips-mode</code>	クライアント側で FIPS モードを有効にするかどうか		
<code>--ssl-key</code>	X.509 キーを含むファイル		
<code>--ssl-mode</code>	サーバーへの接続に必要なセキュリティ状態		
<code>--tab</code>	タブ区切りのデータファイルを生成		
<code>--tables</code>	<code>--databases</code> または <code>-B</code> オプションのオーバーライド		
<code>--tls-ciphersuites</code>	暗号化された接続に許可される TLSv1.3 暗号スイート	8.0.16	
<code>--tls-version</code>	暗号化された接続に許可される TLS プロトコル		
<code>--triggers</code>	ダンプされた各テーブルについて、トリガーをダンプする		
<code>--tz-utc</code>	SET TIME_ZONE='+00:00'をダンプファイルに追加		
<code>--user</code>	サーバーへの接続時に使用する MySQL ユーザー名		
<code>--verbose</code>	冗長モード		
<code>--version</code>	バージョン情報を表示して終了		
<code>--where</code>	指定された WHERE 条件で選択された行のみダンプ		

オプション名	説明	導入	非推奨
<code>--xml</code>	XML 出力を生成		
<code>--zstd-compression-level</code>	zstd 圧縮を使用するサーバーへの接続の圧縮レベル	8.0.18	

接続オプション

`mysqldump` コマンドは MySQL サーバーにログインして情報を抽出します。次のオプションは、同じマシンまたはリモートシステム上の MySQL サーバーに接続する方法を指定します。

- `--bind-address=ip_address`

複数のネットワークインタフェースを持つコンピュータで、このオプションを使用して、MySQL サーバーへの接続に使用するインタフェースを選択します。

- `--compress, -C`

可能であれば、クライアントとサーバーの間で送信されるすべての情報を圧縮します。 [セクション4.2.8「接続圧縮制御」](#)を参照してください。

MySQL 8.0.18 では、このオプションは非推奨です。MySQL の将来のバージョンで削除されることが予想されます。 [レガシー接続圧縮の構成](#)を参照してください。

- `--compression-algorithms=value`

サーバーへの接続に許可される圧縮アルゴリズム。使用可能なアルゴリズムは、`protocol_compression_algorithms` システム変数の場合と同じです。デフォルト値は `uncompressed` です。

詳細は、 [セクション4.2.8「接続圧縮制御」](#)を参照してください。

このオプションは MySQL 8.0.18 で追加されました。

- `--default-auth=plugin`

使用するクライアント側認証プラグインに関するヒント。 [セクション6.2.17「プラグブル認証」](#)を参照してください。

- `--enable-cleartext-plugin`

`mysql_clear_password` 平文認証プラグインを有効にします。([セクション6.4.1.4「クライアント側クリアテキストプラグブル認証」](#)を参照してください。)

- `--get-server-public-key`

RSA キーベースのパスワード交換に必要な公開キーをサーバーにリクエストします。このオプションは、`caching_sha2_password` 認証プラグインで認証されるクライアントに適用されます。そのプラグインの場合、サーバーは要求されないかぎり公開鍵を送信しません。このオプションは、そのプラグインで認証されないアカウントでは無視されます。クライアントがセキュアな接続を使用してサーバーに接続する場合と同様に、RSA ベースのパスワード交換を使用しない場合も無視されます。

`--server-public-key-path=file_name` が指定され、有効な公開キーファイルが指定されている場合は、`--get-server-public-key` よりも優先されます。

`caching_sha2_password` プラグインの詳細は、 [セクション6.4.1.2「SHA-2 プラグブル認証のキャッシュ」](#)を参照してください。

- `--host=host_name, -h host_name`

与えられたホスト上の MySQL サーバーからデータをダンプします。デフォルトホストは `localhost` です。

- `--login-path=name`

`.mylogin.cnf` ログインパスファイルの指定されたログインパスからオプションを読み取ります。「ログインパス」は、接続先の MySQL サーバーおよび認証に使用するアカウントを指定するオプションを含むオプショングループです。ログインパスファイルを作成または変更するには、`mysql_config_editor` ユーティリティを使用します。セクション4.6.7「`mysql_config_editor` — MySQL 構成ユーティリティー」を参照してください。

このオプションおよびその他のオプションファイルオプションの詳細は、セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」を参照してください。

- `--password[=password], -p[password]`

サーバーへの接続に使用される MySQL アカウントのパスワード。パスワード値はオプションです。指定しない場合、`mysqldump` によってプロンプトが表示されます。指定する場合は、`--password=` または `-p` とそれに続くパスワードの間にスペースなしが存在する必要があります。パスワードオプションを指定しない場合、デフォルトではパスワードは送信されません。

コマンド行でのパスワード指定は、セキュアでないと考えべきです。コマンド行でパスワードを指定しないようにするには、オプションファイルを使用します。セクション6.1.2.1「パスワードセキュリティのためのエンドユーザーガイドライン」を参照してください。

パスワードがなく、`mysqldump` でパスワードの入力を求められないように明示的に指定するには、`--skip-password` オプションを使用します。

- `--pipe, -W`

Windows で、名前付きパイプを使用してサーバーに接続します。このオプションは、ネームパイプ接続をサポートするために `named_pipe` システム変数を有効にしてサーバーを起動した場合にのみ適用されます。また、接続を行うユーザーは、`named_pipe_full_access_group` システム変数で指定された Windows グループのメンバーである必要があります。

- `--plugin-dir=dir_name`

プラグインを検索するディレクトリ。このオプションは、`--default-auth` オプションを使用して認証プラグインを指定しても、`mysqldump` がそれを検出しない場合に指定します。セクション6.2.17「プラグブル認証」を参照してください。

- `--port=port_num, -P port_num`

TCP/IP 接続の場合、使用するポート番号。

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

サーバーへの接続に使用するトランスポートプロトコル。これは、他の接続パラメータが通常、必要なプロトコル以外のプロトコルを使用する場合に便利です。許可される値の詳細は、セクション4.2.7「接続トランスポートプロトコル」を参照してください。

- `--server-public-key-path=file_name`

RSA キーベースのパスワード交換のためにサーバーが必要とする公開キーのクライアント側コピーを含む、PEM 形式のファイルへのパス名。このオプションは、`sha256_password` または `caching_sha2_password` 認証プラグインで認証されるクライアントに適用されます。これらのプラグインのいずれかで認証されないアカウントでは、このオプションは無視されます。クライアントがセキュアな接続を使用してサーバーに接続する場合と同様に、RSA ベースのパスワード交換を使用しない場合も無視されます。

`--server-public-key-path=file_name` が指定され、有効な公開キーファイルが指定されている場合は、`--get-server-public-key` よりも優先されます。

`sha256_password` の場合、このオプションは、MySQL が OpenSSL を使用して構築された場合にのみ適用されます。

`sha256_password` および `caching_sha2_password` プラグインの詳細は、セクション6.4.1.3「SHA-256 プラグブル認証」およびセクション6.4.1.2「SHA-2 プラグブル認証のキャッシュ」を参照してください。

- `--socket=path, -S path`

localhost への接続用に使用する、Unix ソケットファイル、または Windows では使用する名前付きパイプの名前。

Windows では、このオプションは、名前付きパイプ接続をサポートするために `named_pipe` システム変数を有効にしてサーバーを起動した場合にのみ適用されます。また、接続を行うユーザーは、`named_pipe_full_access_group` システム変数で指定された Windows グループのメンバーである必要があります。

- `--ssl*`

`--ssl` で始まるオプションは、SSL を使用してサーバーに接続するかどうかを指定し、SSL 鍵および証明書を検索する場所を指定します。暗号化接続のコマンドオプションを参照してください。

- `--ssl-fips-mode={OFF|ON|STRICT}`

クライアント側で FIPS モードを有効にするかどうかを制御します。 `--ssl-fips-mode` オプションは、暗号化された接続の確立には使用されず、許可する暗号化操作に影響する点で、他の `--ssl-xxx` オプションとは異なります。 [セクション6.8「FIPS のサポート」](#) を参照してください。

次の `--ssl-fips-mode` 値を使用できます:

- `OFF`: FIPS モードを無効にします。
- `ON`: FIPS モードを有効にします。
- `STRICT`: 「strict」 FIPS モードを有効にします。

注記

OpenSSL FIPS オブジェクトモジュールが使用できない場合、`--ssl-fips-mode` に許可される値は `OFF` のみです。この場合、`--ssl-fips-mode` を `ON` または `STRICT` に設定すると、クライアントは起動時に警告を生成し、FIPS 以外のモードで動作します。

- `--tls-ciphersuites=ciphersuite_list`

TLSv1.3 を使用する暗号化された接続に許可される暗号スイート。値は、コロンで区切られた 1 つ以上の暗号スイート名のリストです。このオプションに指定できる暗号スイートは、MySQL のコンパイルに使用される SSL ライブラリによって異なります。詳細は、[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#) を参照してください。

このオプションは MySQL 8.0.16 で追加されました。

- `--tls-version=protocol_list`

暗号化された接続に許可される TLS プロトコル。値は、1 つまたは複数のコンマ区切りプロトコル名のリストです。このオプションに指定できるプロトコルは、MySQL のコンパイルに使用される SSL ライブラリによって異なります。詳細は、[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#) を参照してください。

- `--user=user_name, -u user_name`

サーバーへの接続に使用する MySQL アカウントのユーザー名。

- `--zstd-compression-level=level`

`zstd` 圧縮アルゴリズムを使用するサーバーへの接続に使用する圧縮レベル。許可されるレベルは 1 から 22 で、大きい値は圧縮レベルの増加を示します。デフォルトの `zstd` 圧縮レベルは 3 です。圧縮レベルの設定は、`zstd` 圧縮を使用しない接続には影響しません。

詳細は、[セクション4.2.8「接続圧縮制御」](#) を参照してください。

このオプションは MySQL 8.0.18 で追加されました。

オプションファイルオプション

これらのオプションは、どのオプションファイルを読み取るかを制御するために使用されます。

- `--defaults-extra-file=file_name`

このオプションファイルは、グローバルオプションファイルのあとに読み取りますが、(UNIX では) ユーザーオプションファイルの前に読み取るようにしてください。ファイルが存在しないかアクセスできない場合、エラーが発生します。`file_name` は、フルパス名でなく相対パス名として指定された場合、現行ディレクトリを基準にして解釈されます。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--defaults-file=file_name`

指定されたオプションファイルのみ使用します。ファイルが存在しないかアクセスできない場合、エラーが発生します。`file_name` は、フルパス名でなく相対パス名として指定された場合、現行ディレクトリを基準にして解釈されます。

例外: `--defaults-file` でも、クライアントプログラムは `.mylogin.cnf` を読み取ります。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--defaults-group-suffix=str`

通常のオプショングループだけでなく、通常の名前に `str` のサフィクスが付いたグループも読み取ります。たとえば、`mysqldump` は通常 `[client]` グループおよび `[mysqldump]` グループを読み取ります。`--defaults-group-suffix=_other` オプションを指定した場合、`mysqldump` は `[client_other]` グループおよび `[mysqldump_other]` グループも読み取ります。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--no-defaults`

オプションファイルを読み取りません。オプションファイルから不明のオプションを読み取ることが原因でプログラムの起動に失敗する場合、`--no-defaults` を使用して、オプションを読み取らないようにできます。

例外として、`.mylogin.cnf` ファイルは、存在する場合はすべての場合に読み取られます。これにより、`--no-defaults` が使用された場合にも、コマンド行よりも安全な方法でパスワードを指定できます。(`.mylogin.cnf` は `mysql_config_editor` ユーティリティーによって作成されます。[セクション4.6.7「mysql_config_editor — MySQL 構成ユーティリティー」](#)を参照してください)。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--print-defaults`

プログラム名と、オプションファイルから受け取るすべてのオプションを出力します。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

DDL オプション

`mysqldump` の使用シナリオには、新しい MySQL インスタンス全体 (データベーステーブルを含む) のセットアップ、および既存のインスタンス内部のデータを既存のデータベースおよびテーブルで置換することが含まれます。次のオプションを使用すると、ダンプファイル内にさまざまな DDL ステートメントをエンコードすることによって、ダンプをリストアする際に何を削除し何をセットアップするのかが指定できます。

- `--add-drop-database`

各 `CREATE DATABASE` ステートメントの前に `DROP DATABASE` ステートメントを記述します。通常このオプションは、`--all-databases` オプションまたは `--databases` オプションとともに使用されます。これらのオプションのいずれかを指定しないと `CREATE DATABASE` ステートメントが書き込まれないからです。

注記

MySQL 8.0 では、`mysql` スキーマはエンドユーザーが削除できないシステムスキーマとみなされます。`--add-drop-database` が `--all-databases` または `--databases` とともに使用され、ダンプするスキーマのリストに `mysql` が含まれている場合、ダンプファイルにはダンプファイルのリロード時にエラーを引き起こす `DROP DATABASE `mysql`` ステートメントが含まれます。

かわりに、`--add-drop-database` を使用するには、ダンプするスキーマのリストとともに `--databases` を使用します。このリストには `mysql` は含まれません。

• `--add-drop-table`

各 `CREATE TABLE` ステートメントの前に `DROP TABLE` ステートメントを記述します。

• `--add-drop-trigger`

各 `CREATE TRIGGER` ステートメントの前に `DROP TRIGGER` ステートメントを記述します。

• `--all-tablespaces, -Y`

NDB テーブルが使用するテーブルスペースを作成するために必要なすべての SQL ステートメントをテーブルダンプに追加します。そうしないと、この情報は `mysqldump` の出力には含まれません。このオプションは現在、「NDB Cluster」テーブルにのみ関連しています。

• `--no-create-db, -n`

`--databases` または `--all-databases` オプションが指定されている場合は、出力に含まれる `CREATE DATABASE` ステートメントを抑制します。

• `--no-create-info, -t`

ダンプされた各テーブルを作成する `CREATE TABLE` ステートメントを記述しないでください。

注記

このオプションでは、ログファイルグループまたはテーブルスペースを作成するステートメントは `mysqldump` 出力から除外されませんが、この目的には `--no-tablespaces` オプションを使用できます。

• `--no-tablespaces, -y`

このオプションは、`mysqldump` の出力内のすべての `CREATE LOGFILE GROUP` ステートメントおよび `CREATE TABLESPACE` ステートメントを抑制します。

• `--replace`

`INSERT` ステートメントではなく `REPLACE` ステートメントを書き込みます。

デバッグオプション

次のオプションは、デバッグ情報を出力したり、ダンプファイルにデバッグ情報をエンコードしたり、または潜在的な問題にかかわらずダンプ操作を続行させたりします。

• `--allow-keywords`

キーワードであるカラム名の作成を許可します。これは各カラム名にテーブル名のプリフィクスを用いることで機能します。

• `--comments, -i`

プログラムバージョン、サーバーバージョン、およびホストなどの追加情報をダンプファイルに書き込みます。このオプションはデフォルトで有効となっています。この追加情報を抑制するには、`--skip-comments` を使用してください。

- `--debug[=debug_options], -# [debug_options]`

デバッグのログを書き込みます。一般的な `debug_options` 文字列は `d:t:o,file_name` です。デフォルト値は `d:t:o/tmp/mysqldump.trace` です。

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `--debug-check`

プログラムの終了時に、デバッグ情報を出力します。

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `--debug-info`

プログラムの終了時に、デバッグ情報とメモリーおよび CPU 使用率の統計を出力します。

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `--dump-date`

`--comments` オプションが指定された場合、`mysqldump` はダンプの最後に次の形式でコメントを生成します。

```
-- Dump completed on DATE
```

ただし、別のときに取られたダンプファイルが、日付以外のデータがまったく同じでも日付のために異なって見えます。`--dump-date` および `--skip-dump-date` は、コメントに日付を追加するかどうかを制御します。デフォルトは `--dump-date` (日付をコメントに含める) です。`--skip-dump-date` は日付の出力を抑制します。

- `--force, -f`

すべてのエラーを無視します。テーブルダンプ中に SQL エラーが発生した場合でも続行します。

このオプションの使い方の 1 つとして、削除されたテーブルをビュー定義が参照するために無効になっているビューを検出したときにも、`mysqldump` が実行を続けるようにすることです。`--force` を指定しないと、`mysqldump` はエラーメッセージで終了します。`--force` を使用すると、`mysqldump` はエラーメッセージを出力しますが、さらにビュー定義を含む SQL コメントをダンプ出力に書き込み、実行を継続します。

特定のエラーを無視するために `--ignore-error` オプションも指定されている場合は、`--force` が優先されます。

- `--log-error=file_name`

警告およびエラーを、指名されたファイルに追加することによってログに記録します。デフォルトでは、ロギングを行いません。

- `--skip-comments`

`--comments` オプションの説明を参照してください。

- `--verbose, -v`

冗長モード。プログラムの動作についてより多くの情報を出力します。

ヘルプオプション

次のオプションは、`mysqldump` コマンド自身に関する情報を表示します。

- `--help, -?`

ヘルプメッセージを表示して終了します。

- `--version, -V`

バージョン情報を表示して終了します。

国際化オプション

次のオプションは、`mysqldump` コマンドが各国語の設定で文字データを表現する方法を変更します。

- `--character-sets-dir=dir_name`

文字セットがインストールされているディレクトリ。 [セクション10.15「文字セットの構成」](#) を参照してください。

- `--default-character-set=charset_name`

`charset_name` をデフォルト文字セットとして使用します。 [セクション10.15「文字セットの構成」](#) を参照してください。文字セットが指定されていない場合、`mysqldump` は `utf8` を使用します。

- `--no-set-names, -N`

`--set-charset` 設定をオフにします。 `--skip-set-charset` を指定するのと同様です。

- `--set-charset`

`SET NAMES default_character_set` を出力に書き込みます。このオプションはデフォルトで有効となっています。`SET NAMES` ステートメントを抑制するには、`--skip-set-charset` を使用してください。

レプリケーションオプション

`mysqldump` コマンドは、レプリケーション構成のレプリカサーバーに空のインスタンスまたはデータを含むインスタンスを作成するために頻繁に使用されます。次のオプションは、レプリケーションソースサーバーおよびレプリカ上のデータのダンプと復元に適用されます。

- `--apply-slave-statements`

`--dump-slave` オプションを使用して生成されたレプリカダンプの場合は、バイナリログ座標を持つステートメントの前に `STOP REPLICATION | SLAVE` ステートメントを追加し、出力の最後に `START REPLICATION | SLAVE` ステートメントを追加します。

- `--delete-master-logs`

レプリケーションソースサーバーで、ダンプ操作の実行後に `PURGE BINARY LOGS` ステートメントをサーバーに送信して、バイナリログを削除します。このオプションには、`RELOAD` 権限と、そのステートメントを実行するのに十分な権限が必要です。このオプションは自動的に `--master-data` を有効にします。

- `--dump-slave[=value]`

このオプションは、`--master-data` と似ていますが、ダンプされたサーバーと同じソースを持つレプリカとして別のサーバーを設定するために使用できるダンプファイルを生成するためにレプリカサーバーをダンプするために使用される点が異なります。ダンプ出力には、ダンプされたレプリカソースのバイナリログ座標(ファイル名と位置)を示す `CHANGE REPLICATION SOURCE TO` ステートメント(MySQL 8.0.23 の場合)または `CHANGE MASTER TO` ステートメント(MySQL 8.0.23 の場合)が含まれます。`CHANGE REPLICATION SOURCE TO` ステートメントは、`SHOW REPLICATION | SLAVE STATUS` 出力から `Relay_Master_Log_File` および `Exec_Master_Log_Pos` の値を読み取り、`SOURCE_LOG_FILE` および `SOURCE_LOG_POS` にそれぞれ使用します。これらは、レプリカがレプリケートを開始するレプリケーションソースサーバーの座標です。

注記

実行されたりレログからの一連のトランザクションに一貫性がないと、間違った位置が使用される可能性があります。詳しくは[セクション17.5.1.34「レプリケーションとトランザクションの非一貫性」](#)をご覧ください。

`--dump-slave` では、`--master-data` オプションと同様に、ダンプされたサーバーの座標ではなくソースの座標が使用されます。また、このオプションを指定すると、`--master-data` オプションがオーバーライドされ(使用されている場合)、事実上無視されます。

警告

ダンプを適用するサーバーで `gtid_mode=ON` および `MASTER_AUTOPOSITION=1` を使用する場合は、このオプションを使用しないでください。

オプション値は、`--master-data` の場合と同様に処理されます。値を設定しないか、1 に設定すると、(MySQL 8.0.23 の) `CHANGE REPLICATION SOURCE TO` ステートメントまたは (MySQL 8.0.23 の前の) `CHANGE MASTER TO` ステートメントがダンプに書き込まれます。2 に設定すると、ステートメントは書き込まれますが、SQL コメントに含まれます。他のオプションの有効化または無効化、およびロックの処理方法に関しては、`--master-data` と同じ効果があります。

このオプションにより、`mysqldump` はダンプの前にレプリケーション SQL スレッドを停止し、その後再起動します。

`--dump-slave` は、情報を取得するために `SHOW REPLICAS | SLAVE STATUS` ステートメントをサーバーに送信するため、そのステートメントを実行するのに十分な権限が必要です。

`--dump-slave` とともに、`--apply-slave-statements` オプションおよび `--include-master-host-port` オプションも使用できます。

- `--include-master-host-port`

`CHANGE REPLICATION SOURCE TO` ステートメント (MySQL 8.0.23 から) または `--dump-slave` オプション付きで生成されたレプリカダンプの `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 の前) の場合は、ホスト名に `SOURCE_HOST | MASTER_HOST` および `SOURCE_PORT | MASTER_PORT` オプション、およびレプリカソースの TCP/IP ポート番号を追加します。

- `--master-data[=value]`

このオプションを使用して、レプリケーションソースサーバーをダンプし、別のサーバーをソースのレプリカとして設定するために使用できるダンプファイルを生成します。ダンプ出力には、ダンプされたサーバーのバイナリログ座標 (ファイル名と位置) を示す `CHANGE REPLICATION SOURCE TO` ステートメント (MySQL 8.0.23 の場合) または `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 の場合) が含まれます。これらは、ダンプファイルがレプリカにロードされた後にレプリカがレプリケートを開始するレプリケーションソースサーバーの座標です。

オプション値が 2 の場合、`CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO` ステートメントは SQL コメントとして書き込まれるため、情報提供のみを目的としており、ダンプファイルがリロードされても効果はありません。オプション値が 1 の場合、ステートメントはコメントとしては書き込まれず、ダンプファイルがリロードされるときに実行されます。オプション値が指定されていない場合は、デフォルト値は 1 です。

`--master-data` は、情報を取得するために `SHOW MASTER STATUS` ステートメントをサーバーに送信するため、そのステートメントを実行するのに十分な権限が必要です。このオプションには `RELOAD` 権限も必要であり、バイナリログを有効にする必要があります。

`--master-data` オプションは自動的に `--lock-tables` をオフにします。また、`--single-transaction` も指定されていない場合は、`--lock-all-tables` をオンにします。その場合、ダンプの最初のわずかな時間のみグローバル読み取りロックが取得されます (`--single-transaction` の説明を参照してください)。どの場合でも、ログに対するアクションはすべてダンプと同時に発生します。

`--dump-slave` オプションを使用してソースの既存のレプリカをダンプすることでレプリカを設定することもできます。このオプションは `--master-data` をオーバーライドし、両方のオプションを使用すると無視されます。

- `--set-gtid-purged=value`

このオプションは GTID ベースのレプリケーション (`gtid_mode=ON`) を使用するサーバー用です。ダンプ出力への `SET @@GLOBAL.gtid_purged` ステートメントのインクルードを制御し、ダンプファイルがリロードされるサーバー上の `gtid_purged` の値を更新して、ソースサーバーの `gtid_executed` システム変数から GTID セットを追加します。`gtid_purged` は、サーバーに適用されたが、サーバー上のバイナリログファイルには存在しないすべてのトランザクションの GTID を保持します。したがって、`mysqldump` はソースサーバーで実行されたトランザクションの GTID を追加するため、ターゲットサーバーはこれらのトランザクションを適用済として記録しますが、バイナリログには記録しません。`--set-gtid-purged` は `SET @@SESSION.sql_log_bin=0` ステートメントのインクルードも制御します。これにより、ダンプファイルのリロード中にバイナリロギングが無効になります。このステートメント

は、トランザクションの元の GTID が使用されるように、新しい GTID が生成され、実行時にダンプファイル内のトランザクションに割り当てられないようにします。

`--set-gtid-purged` オプションを設定しない場合、デフォルトでは、バックアップするサーバーで GTID が有効になっており、`gtid_executed` システム変数のグローバル値の GTID のセットが空でない場合に、`SET @@GLOBAL.gtid_purged` ステートメントがダンプ出力に含まれます。GTID がサーバーで有効になっている場合は、`SET @@SESSION.sql_log_bin=0` ステートメントも含まれます。

MySQL 5.6 および 5.7 では、`gtid_executed` および `gtid_purged` が空の場合、`gtid_purged` の値を指定された GTID セットに置き換えることができます。MySQL 8.0 から、`gtid_purged` の値を指定された GTID セットに置き換えることも、プラス記号 (+) をステートメントに追加して、`gtid_purged` によってすでに保持されている GTID セットに指定された GTID セットを追加することもできます。`mysqldump` の `SET @@GLOBAL.gtid_purged` ステートメントには、MySQL 8.0 からのリリースでダンプファイルがリプレイされるときに有効になるバージョンコメントにプラス記号 (+) が含まれています。つまり、これらのリリースでは、ダンプファイルから設定された GTID が既存の `gtid_purged` 値に追加されます。MySQL 5.6 および 5.7 の場合、`gtid_purged` の値はダンプファイルの GTID セットに置き換えられます。これは、`gtid_executed` が空のセットである場合 (レプリケーションが以前に開始されていない場合、またはレプリケーションが GTID を以前に使用していなかった場合) にのみ発生する可能性があります。`SET @@GLOBAL.gtid_purged` ステートメントの動作の詳細は、ダンプファイルがリプレイされるリリースの `gtid_purged` の説明を参照してください。

`SET @@GLOBAL.gtid_purged` ステートメントの `mysqldump` に含まれる値には、サーバー上の `gtid_executed` セット内のすべてのトランザクションの GTID (データベースの抑制された部分を変更したトランザクションや、部分ダンプに含まれていなかったサーバー上のその他のデータベースも含む) が含まれることに注意してください。これは、ダンプファイルがリプレイされるサーバーで `gtid_purged` 値が更新された後、ターゲットサーバー上のデータに関連しない GTID が存在することを意味します。ターゲットサーバーでこれ以上ダンプファイルをリプレイしない場合、余分な GTID によってサーバーの将来の操作で問題が発生することはありませんが、レプリケーショントポロジ内の異なるサーバー上の GTID セットを比較またはリコンサイルすることは困難になります。同じ GTID (同じオリジンサーバーからの別の部分ダンプなど) を含むターゲットサーバーでさらにダンプファイルをリプレイすると、2 番目のダンプファイル内の `SET @@GLOBAL.gtid_purged` ステートメントは失敗します。この場合は、ダンプファイルをリプレイする前にステートメントを手動で削除するか、ステートメントなしでダンプファイルを出力します。

注記

MySQL 5.6 および 5.7 では、ダンプファイルにシステムテーブルが含まれている場合、GTID がサーバー (`gtid_mode=ON`) で有効になっているときにダンプファイルをロードすることはお勧めしません。`mysqldump` は、非トランザクション MyISAM ストレージエンジンを使用するシステムテーブルに対して DML 命令を発行します。GTID が有効になっている場合、この組み合わせは許可されません。

ターゲットサーバーで `SET @@GLOBAL.gtid_purged` ステートメントに目的の結果が得られない場合は、出力からステートメントを除外するか、(MySQL 8.0.17) ステートメントを含めて自動的にアクションされないようにコメントアウトできます。ステートメントを含めることもできますが、必要な結果を得るには、ダンプファイルを手動で編集します。

`--set-gtid-purged` オプションに使用可能な値は次のとおりです:

- AUTO** デフォルト値。バックアップするサーバーで GTID が有効になっており、`gtid_executed` が空でない場合は、`gtid_executed` からの GTID セットを含む `SET @@GLOBAL.gtid_purged` が出力に追加されます。GTID が有効な場合、`SET @@SESSION.sql_log_bin=0` が出力に追加されます。GTID がサーバーで有効になっていない場合、ステートメントは出力に追加されません。
- OFF** `SET @@GLOBAL.gtid_purged` は出力に追加されず、`SET @@SESSION.sql_log_bin=0` は出力に追加されません。GTID が使用されていないサーバーの場合は、このオプションまたは **AUTO** を使用します。GTID が使用されているサーバーでは、必要な GTID セットがターゲットサーバーの `gtid_purged` にすでに存在し、変更しないことが確実な場合、または欠落している GTID を手動で識別して追加する予定の場合にのみ、このオプションを使用します。
- ON** バックアップするサーバーで GTID が有効になっている場合、(`gtid_executed` が空でないかぎり) `SET @@GLOBAL.gtid_purged` が出力に追加され、`SET @@SESSION.sql_log_bin=0` が出力に追加されます。このオプションを設定しても GTID がサーバーで有効になっていない場合は、**工**

ラーが発生します。GTID が使用されているサーバーの場合は、`gtid_executed` の GTID がターゲットサーバーで必要ないことが確実でないかぎり、このオプションまたは `AUTO` を使用します。

COMMENTED MySQL 8.0.17 から入手できます。バックアップしているサーバーで GTID が有効になっている場合、(`gtid_executed` が空でないかぎり) `SET @@GLOBAL.gtid_purged` が出力に追加されますが、コメントアウトされます。つまり、`gtid_executed` の値は出力で使用できますが、ダンプファイルがリロードされてもアクションは自動的に実行されません。 `SET @@SESSION.sql_log_bin=0` が出力に追加され、コメントアウトされません。 **COMMENTED** を使用すると、`gtid_executed` セットの使用を手動または自動化で制御できます。たとえば、アクティブなデータベースがすでに異なる別のサーバーにデータを移行する場合は、これを行うことをお勧めします。

形式オプション

次のオプションは、ダンプファイル全体またはダンプファイル内のある種のデータの提示方法を指定します。また、ある種のオプションの情報をダンプファイルに書き込むかどうかを制御します。

- `--compact`

よりコンパクトな出力を生成します。このオプションは、`--skip-add-drop-table`、`--skip-add-locks`、`--skip-comments`、`--skip-disable-keys`、および `--skip-set-charset` オプションを有効にします。

- `--compatible=name`

古い MySQL サーバーやほかのデータベースシステムとの互換性がより高い出力を生成します。このオプションに指定できる値は `ansi` のみです。これは、サーバー SQL モードを設定するための対応するオプションと同じ意味を持ちます。 [セクション5.1.11「サーバー SQL モード」](#) を参照してください。

- `--complete-insert, -c`

カラム名を含む、完全な `INSERT` ステートメントを使用します。

- `--create-options`

MySQL 固有のテーブルオプションを `CREATE TABLE` ステートメントに含めます。

- `--fields-terminated-by=...`, `--fields-enclosed-by=...`, `--fields-optionally-enclosed-by=...`, `--fields-escaped-by=...`

これらのオプションは `--tab` オプションとともに使用され、`LOAD DATA` の対応する `FIELDS` 句と同じ意味を持ちます。 [セクション13.2.7「LOAD DATA ステートメント」](#) を参照してください。

- `--hex-blob`

16 進表記を使用してバイナリカラムをダンプします (たとえば、`'abc'` は `0x616263` となります)。影響を受けるデータ型は、`binary character set` で使用する場合、`BINARY`、`VARBINARY`、`BLOB` 型、`BIT`、すべての空間データ型およびその他の非バイナリデータ型です。

- `--lines-terminated-by=...`

このオプションは `--tab` オプションとともに使用され、`LOAD DATA` の対応する `LINES` 句と同じ意味を持ちます。 [セクション13.2.7「LOAD DATA ステートメント」](#) を参照してください。

- `--quote-names, -Q`

識別子 (データベース、テーブル、およびカラム名など) を ` 文字で囲みます。 `ANSI_QUOTES` SQL モードが有効な場合、識別子は " 文字で囲まれます。このオプションはデフォルトで有効となっています。 `--skip-quote-names` で無効にできますが、このオプションは `--compatible` のような `--quote-names` を有効にする可能性のあるオプションのあとに指定するようにしてください。

- `--result-file=file_name, -r file_name`

指定されたファイルに出力を転送します。ダンプの生成中にエラーが発生しても、結果ファイルが作成され以前の内容は上書きされます。

このオプションは、改行\n 文字が\r\n キャリッジリターン/改行シーケンスに変換されないようにするために、Windows で使用する必要があります。

- `--show-create-skip-secondary-engine=value`

`CREATE TABLE` ステートメントから `SECONDARY ENGINE` 句を除外します。これを行うには、ダンプ操作中に `show_create_table_skip_secondary_engine` システム変数を有効にします。または、`mysqldump` を使用する前に `show_create_table_skip_secondary_engine` システム変数を有効にすることもできます。

このオプションは MySQL 8.0.18 で追加されました。 `show_create_table_skip_secondary_engine` 変数をサポートしていない MySQL 8.0.18 より前のリリースで `--show-create-skip-secondary-engine` オプションを使用して `mysqldump` 操作を試行すると、エラーが発生します。

- `--tab=dir_name, -T dir_name`

タブ区切りのテキスト形式データファイルを生成します。 `mysqldump` は、各ダンプテーブルに対して、テーブルを作成する `CREATE TABLE` ステートメントを含む `tbl_name.sql` ファイルを作成し、サーバーはそのデータを含む `tbl_name.txt` ファイルに書き込みます。オプション値はファイルを書き込むディレクトリです。

注記

このオプションは、`mysqldump` が `mysqld` サーバーと同じマシンで動作している場合のみ使用するようにしてください。サーバーは指定したディレクトリに `*.txt` ファイルを作成するため、ディレクトリはサーバーによって書き込み可能である必要があり、使用する MySQL アカウントには `FILE` 権限が必要です。 `mysqldump` は同じディレクトリに `*.sql` を作成するため、システムログインアカウントによって書き込み可能である必要があります。

デフォルトでは、`.txt` データファイルはカラム値の間にタブ文字、各行の最後に改行を使用する形式になります。この形式は、`--fields-xxx` オプションおよび `--lines-terminated-by` オプションを使用して明示的に指定できます。

カラム値は、`--default-character-set` オプションで指定された文字セットに変換されます。

- `--tz-utc`

このオプションにより、異なるタイムゾーンのサーバー間で `TIMESTAMP` カラムをダンプしてリロードできるようになります。 `mysqldump` はその接続タイムゾーンを UTC に設定し、`SET TIME_ZONE='+00:00'` をダンプファイルに追加します。このオプションを使用しないと、`TIMESTAMP` カラムはダンプ元およびリロード先のサーバーのローカルタイムゾーンでダンプおよびリロードが実行され、サーバーが異なるタイムゾーンにある場合、値が変更されます。 `--tz-utc` は、サマータイムによる変更からも保護します。 `--tz-utc` はデフォルトで有効です。無効にするには、`--skip-tz-utc` を使用します。

- `--xml, -X`

ダンプ出力および整形 XML を書き出します。

`NULL`、`'NULL'`、および空の値: このオプションで生成される出力では、`column_name` という名前のカラムに関して、`NULL` 値、空の文字列、および文字列値 `'NULL'` は次のように互いに区別されます。

値:	XML 表現:
<code>NULL</code> (不明な値)	<code><field name="column_name" xsi:nil="true" /></code>
<code>"</code> (空の文字列)	<code><field name="column_name"></field></code>
<code>'NULL'</code> (文字列値)	<code><field name="column_name">NULL</field></code>

`mysql` クライアントを `--xml` オプションを使用して実行した場合の出力も、前記のルールに従います。(セクション 4.5.1.1 「`mysql` クライアントオプション」を参照してください。)

`mysqldump` からの XML 出力には、次に示すように XML 名前空間が含まれます。

```
shell> mysqldump --xml -u root world City
<?xml version="1.0"?>
```

```
<mysqldump xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<database name="world">
<table_structure name="City">
<field Field="ID" Type="int(11)" Null="NO" Key="PRI" Extra="auto_increment" />
<field Field="Name" Type="char(35)" Null="NO" Key="" Default="" Extra="" />
<field Field="CountryCode" Type="char(3)" Null="NO" Key="" Default="" Extra="" />
<field Field="District" Type="char(20)" Null="NO" Key="" Default="" Extra="" />
<field Field="Population" Type="int(11)" Null="NO" Key="" Default="0" Extra="" />
<key Table="City" Non_unique="0" Key_name="PRIMARY" Seq_in_index="1" Column_name="ID"
Collation="A" Cardinality="4079" Null="" Index_type="BTREE" Comment="" />
<options Name="City" Engine="MyISAM" Version="10" Row_format="Fixed" Rows="4079"
Avg_row_length="67" Data_length="273293" Max_data_length="18858823439613951"
Index_length="43008" Data_free="0" Auto_increment="4080"
Create_time="2007-03-31 01:47:01" Update_time="2007-03-31 01:47:02"
Collation="latin1_swedish_ci" Create_options="" Comment="" />
</table_structure>
<table_data name="City">
<row>
<field name="ID">1</field>
<field name="Name">Kabul</field>
<field name="CountryCode">AFG</field>
<field name="District">Kabul</field>
<field name="Population">1780000</field>
</row>
...
<row>
<field name="ID">4079</field>
<field name="Name">Rafah</field>
<field name="CountryCode">PSE</field>
<field name="District">Rafah</field>
<field name="Population">92020</field>
</row>
</table_data>
</database>
</mysqldump>
```

フィルタリングオプション

次のオプションは、どのような種類のスキーマオブジェクトがダンプファイルに書き出されるかを、トリガーまたはイベントなどのカテゴリによって制御したり、たとえばダンプするデータベースおよびテーブルを選択して名前によって制御したり、または **WHERE** 句を使用してテーブルデータから行をフィルタリングして制御したりできます。

- **--all-databases, -A**

すべてのデータベース内のすべてのテーブルをダンプします。これは、コマンド行で **--databases** オプションを使用してすべてのデータベース名を指定するのと同じです。

注記

--all-databases との非互換性の詳細は、**--add-drop-database** の説明を参照してください。

MySQL 8.0 より前では、**mysqldump** および **mysqlpump** の **--routines** および **--events** オプションは、**--all-databases** オプションの使用時にストアドルーチンおよびイベントを含める必要はありませんでした: ダンプには **mysql** システムデータベースが含まれていたため、ストアドルーチンおよびイベント定義を含む **mysql.proc** および **mysql.event** テーブルも含まれていました。MySQL 8.0 では、**mysql.event** テーブルおよび **mysql.proc** テーブルは使用されません。対応するオブジェクトの定義はデータディクショナリテーブルに格納されますが、これらのテーブルはダンプされません。 **--all-databases** を使用して作成されたダンプにストアドルーチンおよびイベントを含めるには、**--routines** および **--events** オプションを明示的に使用します。

- **--databases, -B**

複数のデータベースをダンプします。通常、**mysqldump** は、コマンド行の最初の名前引数をデータベース名として、それに続く名前をテーブル名として処理します。このオプションを使用すると、名前引数をすべてデータベース名として処理します。出力には、各新しいデータベースの前に **CREATE DATABASE** ステートメントおよび **USE** ステートメントが含まれます。

このオプションは、`performance_schema` データベースのダンプに使用できます。通常、`--all-databases` オプションでもダンプされません。(`--skip-lock-tables` オプションも使用してください。)

注記

`--databases` との非互換性の詳細は、`--add-drop-database` の説明を参照してください。

- `--events, -E`

ダンプされるデータベースのイベントスケジューライベントを出力に含めます。このオプションには、これらのデータベースに対する `EVENT` 権限が必要です。

`--events` を使用して生成される出力には、イベントを作成するための `CREATE EVENT` ステートメントが含まれています。

- `--ignore-error=error[,error]...`

指定されたエラーを無視 オプション値は、`mysqldump` の実行中に無視するエラーを指定するカンマ区切りのエラー番号のリストです。すべてのエラーを無視するために `--force` オプションも指定されている場合は、`--force` が優先されます。

- `--ignore-table=db_name.tbl_name`

指定されたテーブルをダンプしません。これはデータベース名とテーブル名を両方指定する必要があります。複数のテーブルを無視するには、このオプションを複数回使用してください。このオプションを使用してビューを無視することもできます。

- `--no-data, -d`

テーブルの行情報を書き出しません (つまり、テーブルの内容をダンプしません)。これは、テーブルの `CREATE TABLE` ステートメントのみをダンプする場合に便利です (たとえば、ダンプファイルをロードしてテーブルの空のコピーを作成する場合など)。

- `--routines, -R`

ダンプされるデータベースのストアドルーチン (プロシージャおよび関数) を出力に含めます。このオプションには、グローバル `SELECT` 権限が必要です。

`--routines` を使用して生成される出力には、ルーチンを作成するための `CREATE PROCEDURE` および `CREATE FUNCTION` ステートメントが含まれています。

- `--tables`

`--databases` オプションまたは `-B` オプションをオーバーライドします。`mysqldump` は、このオプションに続く名前の引数をすべてテーブル名とみなします。

- `--triggers`

ダンプされる各テーブルのトリガーを出力に含めます。このオプションはデフォルトで有効です。`--skip-triggers` を使用して無効にします。

テーブルトリガーをダンプできるようにするには、そのテーブルに対する `TRIGGER` 権限が必要です。

複数のトリガーが許可されます。`mysqldump` は、ダンプファイルがリロードされたときにトリガーが同じアクティブ化順序で作成されるように、トリガーをアクティブ化順にダンプします。ただし、`mysqldump` ダンプファイルに、トリガーイベントとアクション時間が同じテーブルに対する複数のトリガーが含まれている場合、複数のトリガーをサポートしていない古いサーバーにダンプファイルをロードしようとするエラーが発生します。(回避策については、[Downgrade Notes](#) を参照してください。古いサーバーと互換性を持つようにトリガーを変換できます。)

- `--where='where_condition', -w 'where_condition'`

指定された `WHERE` 条件で選択される行のみダンプします。条件が、スペースまたはユーザーのコマンドインタプリタにとって特別なその他の文字を含んでいる場合、条件を引用符で囲まなければなりません。

例:

```
--where="user='jimf'"
-w"userid>1"
-w"userid<1"
```

パフォーマンスオプション

次のオプションは、特にリストア操作のパフォーマンスにもっとも重要です。大規模なデータセットでは、リストア操作 (ダンプファイル内の `INSERT` ステートメントの処理) がもっとも時間のかかる部分です。データを迅速にリストアすることが緊急である場合、事前にステージを計画してパフォーマンスをテストします。時間単位で測定されたリストア時間の場合、`InnoDB` のみおよび混合使用のデータベース用の `MySQL Enterprise Backup` など、代替のバックアップおよびリストアソリューションを使用することをお勧めします。

パフォーマンスは、主にダンプ操作に関して、[トランザクションオプション](#)にも影響されます。

- `--column-statistics`

ダンプファイルのリロード時にダンプされたテーブルのヒストグラム統計を生成するために、出力に `ANALYZE TABLE` ステートメントを追加します。大規模なテーブルのヒストグラム生成には時間がかかる場合があるため、このオプションはデフォルトで無効になっています。

- `--disable-keys, -K`

テーブルごとに、`INSERT` ステートメントを `/*!40000 ALTER TABLE tbl_name DISABLE KEYS */;` ステートメントと `/*!40000 ALTER TABLE tbl_name ENABLE KEYS */;` ステートメントで囲みます。これにより、行がすべて挿入されたあとにインデックスが作成されるため、ダンプファイルのロードが高速になります。このオプションは、`MyISAM` テーブルの一意でないインデックスにのみ効果があります。

- `--extended-insert, -e`

複数の `VALUES` リストを含む複数行構文を使用して `INSERT` ステートメントを記述します。これにより、ダンプファイルのサイズが小さくなり、ファイルがリロードされる際の挿入が高速化されます。

- `--insert-ignore`

`INSERT` ステートメントではなく、`INSERT IGNORE` ステートメントを書き出します。

- `--max-allowed-packet=value`

クライアント/サーバー通信のバッファの最大サイズ。デフォルトは 24M バイト、最大は 1G バイトです。

- `--net-buffer-length=value`

クライアント/サーバー通信のバッファの初期サイズ。(`--extended-insert` または `--opt` オプションと同様に) 複数行の `INSERT` ステートメントを作成する場合、`mysqldump` は `--net-buffer-length` バイトまでの長さの行を作成します。この変数を増やす場合は、MySQL サーバーの `net_buffer_length` システム変数の値がこの値以上であることを確認してください。

- `--network-timeout, -M`

`--max-allowed-packet` をその最大値に設定し、ネットワークの読取りおよび書き込みタイムアウトを大きな値に設定して、大きなテーブルをダンプできるようにします。このオプションはデフォルトで有効となっています。無効にするには、`--skip-network-timeout` を使用します。

- `--opt`

このオプションはデフォルトで有効で、`--add-drop-table --add-locks --create-options --disable-keys --extended-insert --lock-tables --quick --set-charset` の組み合わせの短縮形です。高速ダンプ操作が可能になり、MySQL サーバーに迅速にリロードできるダンプファイルを生成します。

`--opt` オプションはデフォルトで有効であるため、いくつかのデフォルト設定をオフにする場合のみ、この逆の `--skip-opt` を指定します。`--opt` に影響されるオプションのサブセットを選択的に有効または無効にする方法は、[mysqldump オプショングループ](#) の説明を参照してください。

- `--quick, -q`

このオプションは大規模なテーブルのダンプに便利です。これは `mysqldump` に対して、テーブルのすべての行のセットを取得して、書き出す前にメモリーにバッファリングするのではなく、サーバーから 1 行ずつ行を取得することを強制します。

- `--skip-opt`

`--opt` オプションの説明を参照してください。

トランザクションオプション

次のオプションは、エクスポートされるデータの信頼性と一貫性のために、ダンプ操作のパフォーマンスを犠牲にします。

- `--add-locks`

`LOCK TABLES` ステートメントと `UNLOCK TABLES` ステートメントで各テーブルダンプを囲みます。これにより、ダンプファイルをリロードする際の挿入の速度が向上します。 [セクション8.2.5.1「INSERT ステートメントの最適化」](#) を参照してください。

- `--flush-logs, -F`

ダンプを始める前に MySQL サーバーログファイルをフラッシュします。このオプションには `RELOAD` 権限が必要です。このオプションを `--all-databases` オプションと組み合わせて使用すると、ログはダンプされるデータベースごとにフラッシュされます。例外は、`--lock-all-tables`、`--master-data` または `--single-transaction` の使用時です: この場合、すべてのテーブルが `FLUSH TABLES WITH READ LOCK` によってロックされた時点に対応して、ログは一度のみフラッシュされます。ダンプとログのフラッシュを正確に同時に実行するには、`--flush-logs` を `--lock-all-tables`、`--master-data`、または `--single-transaction` とともに使用するようしてください。

- `--flush-privileges`

`mysql` データベースのダンプ後に、ダンプ出力に `FLUSH PRIVILEGES` ステートメントを追加します。ダンプに `mysql` データベースおよび `mysql` データベース内のデータに依存するその他のすべてのデータベースが含まれている場合には、正しいリストのために必ずこのオプションを使用するようしてください。

ダンプファイルには `FLUSH PRIVILEGES` ステートメントが含まれているため、ファイルをリロードするには、そのステートメントを実行するのに十分な権限が必要です。

注記

古いバージョンから MySQL 5.7 以上にアップグレードする場合は、`--flush-privileges` を使用しないでください。この場合のアップグレード手順については、[セクション2.11.4「MySQL 8.0 での変更」](#) を参照してください。

- `--lock-all-tables, -x`

データベース内のテーブルをすべてロックします。これは全ダンプの期間、グローバル読み取りロックを取得することで達成されます。このオプションにより、`--single-transaction` および `--lock-tables` は自動的にオフになります。

- `--lock-tables, -l`

ダンプされる各データベースに対して、ダンプするすべてのテーブルをダンプ前にロックします。MyISAM テーブルの場合には、並列挿入を許可するため、テーブルは `READ LOCAL` でロックされます。InnoDB などのトランザクションテーブルの場合は、`--single-transaction` はテーブルをロックする必要がまったくないため、`--lock-tables` よりはるかに適したオプションです。

`--lock-tables` は各データベースに対して個別にテーブルをロックするため、このオプションではダンプファイル内のテーブルがデータベース間で論理的に一貫していることは保証されません。異なるデータベース内のテーブルは完全に異なる状態でダンプされることがあります。

`--opt` など、一部のオプションは `--lock-tables` を自動的に有効にします。これをオーバーライドするには、`--skip-lock-tables` をオプションリストの最後に使用します。

- `--no-autocommit`

ダンプされるテーブルごとに、`INSERT` ステートメントを `SET autocommit = 0` ステートメントと `COMMIT` ステートメントで囲みます。

- `--order-by-primary`

各テーブルの行を、主キーまたは最初の一意のインデックス (このようなインデックスが存在する場合) でソートしてダンプします。これは、`InnoDB` テーブルにロードされる `MyISAM` テーブルをダンプする場合に便利ですが、ダンプ操作にかかる時間がかなり長くなります。

- `--shared-memory-base-name=name`

Windows の場合、共有メモリを使用してローカルサーバに接続するために使用する共有メモリ名。デフォルト値は `MYSQL` です。共有メモリ名では大文字と小文字が区別されます。

このオプションは、共有メモリ接続をサポートするために `shared_memory` システム変数を有効にしてサーバを起動した場合にのみ適用されます。

- `--single-transaction`

このオプションは、データのダンプ前に、トランザクション分離モードを `REPEATABLE READ` に設定し、`START TRANSACTION` SQL ステートメントをサーバに送信します。これは、`InnoDB` などのトランザクションテーブルの場合にかぎって便利です。その場合、アプリケーションをブロックすることなく、`START TRANSACTION` が発行された時点のデータベースの一貫した状態をダンプするからです。

このオプションを使用する場合、一貫した状態でダンプされるのは `InnoDB` テーブルのみだということに留意してください。たとえば、このオプションの使用中にダンプされた `MyISAM` テーブルまたは `MEMORY` テーブルは状態が変化する可能性があります。

`--single-transaction` ダンプの処理中、ダンプファイルが正当である (テーブルの内容とバイナリログ座標が正しい) ことを保証するために、ほかの接続で `ALTER TABLE`、`CREATE TABLE`、`DROP TABLE`、`RENAME TABLE`、`TRUNCATE TABLE` ステートメントを使用しないようにしてください。一貫性読み取りはこれらのステートメントから分離されないため、ダンプされるテーブルでこれらを使用すると、`mysqldump` によって実行され、テーブルの内容を取得する `SELECT` が、正しくない内容を取得したり失敗したりすることがあります。

`--single-transaction` オプションおよび `--lock-tables` オプションは相互に排他的です。これは、保留中のトランザクションが `LOCK TABLES` により暗黙的にコミットされるためです。

大規模なテーブルをダンプするには、`--single-transaction` オプションを `--quick` オプションと組み合わせてください。

オプショングループ

- `--opt` オプションは、高速なダンプ操作を実行するために協働するいくつかの設定をオンにします。`--opt` はデフォルトでオンであるため、これらの設定はすべてデフォルトでオンです。したがって、`--opt` を指定することは、あるとしてもまれです。代わりに、`--skip-opt` を指定してこれらの設定をグループとしてオフにし、そのあと、コマンド行で関連するオプションを指定して特定の設定を再度有効にできます。

- `--compact` オプションは、オプションのステートメントおよびコメントが出力に現れるかどうかを制御するいくつかの設定をオフにします。この場合も、このオプションに、特定の設定を再度有効にするその他のオプションを続けたり、`--skip-compact` の形式を使用してすべての設定をオンにしたりできます。

グループオプションの一部を選択的に効果を有効または無効にする場合、オプションは前から後ろへの順で処理されるため、順序が重要です。たとえば、`--disable-keys --lock-tables --skip-opt` では意図している効果を得られません。`--skip-opt` だけの場合と同じになります。

例

データベース全体のバックアップを作成するには:

```
shell> mysqldump db_name > backup-file.sql
```

ダンプファイルをサーバにロードするには:

```
shell> mysql db_name < backup-file.sql
```

ダンプファイルをリロードする別の方法:

```
shell> mysql -e "source /path-to-backup/backup-file.sql" db_name
```

`mysqldump` は、1 つの MySQL サーバーから別のサーバーにデータをコピーすることでデータベースを移入するのに非常に便利です。

```
shell> mysqldump --opt db_name | mysql --host=remote_host -C db_name
```

複数のデータベースを 1 つのコマンドでダンプできます。

```
shell> mysqldump --databases db_name1 [db_name2 ...] > my_databases.sql
```

すべてのデータベースをダンプするには、`--all-databases` オプションを使用します。

```
shell> mysqldump --all-databases > all_databases.sql
```

InnoDB テーブルに関して、`mysqldump` はオンラインバックアップの作成方法を提供します。

```
shell> mysqldump --all-databases --master-data --single-transaction > all_databases.sql
```

このバックアップでは、ダンプの最初で (`FLUSH TABLES WITH READ LOCK` を使用して) すべてのテーブルに対するグローバル読み取りロックが取得されます。このロックが取得されるとすぐに、バイナリログの座標が読み取られ、ロックが解除されます。`FLUSH` ステートメントが発行されたときに長い更新ステートメントが実行中の場合、MySQL サーバーはそれらのステートメントが終わるまで停止する可能性があります。そのあと、ダンプはロックがなくなり、テーブルの読み取りと書き込みを妨げることはなくなります。MySQL サーバーが受信する更新ステートメントが (実行時間の点で) 短い場合、更新の数が多くても最初のロック時間はさほど気にならないはずです。

ポイントインタイムリカバリ (または「ロールフォワード」、これは古いバックアップをリストアし、そのバックアップ後に発生した変更を再現する必要がある場合) は、バイナリログを交替させる ([セクション5.4.4「バイナリログ」](#)を参照してください) か、または少なくともダンプが対応しているバイナリログ座標を知っていると便利な場合があります。

```
shell> mysqldump --all-databases --master-data=2 > all_databases.sql
```

または:

```
shell> mysqldump --all-databases --flush-logs --master-data=2  
> all_databases.sql
```

`--master-data` オプションおよび `--single-transaction` オプションは同時に使用できます。これは、テーブルが InnoDB ストレージエンジンを使用して保存されている場合に、ポイントインタイムリカバリの前に使用するのに適したオンラインバックアップを作成する便利な方法を提供します。

バックアップ作成の詳細は、[セクション7.2「データベースバックアップ方法」](#)と[セクション7.3「バックアップおよびリカバリ戦略の例」](#)を参照してください。

- いくつかの機能を除いて `--opt` の効果を選択するには、除く各機能に対して `--skip` オプションを選択します。拡張挿入およびメモリーバッファリングを無効にするには、`--opt --skip-extended-insert --skip-quick` を使用します。(`--opt` はデフォルトでオンであるため、実際には `--skip-extended-insert --skip-quick` で十分です。)
- インデックスの無効化とテーブルのロックを除くすべての機能に関して `--opt` を反転するには、`--skip-opt --disable-keys --lock-tables` を使用します。

制約

`mysqldump` は、デフォルトでは `performance_schema` または `sys` スキーマをダンプしません。これらのいずれかをダンプするには、コマンドラインで明示的に名前を付けます。`--databases` オプションでも指定できます。`performance_schema` の場合は、`--skip-lock-tables` オプションも使用します。

`mysqldump` は、`INFORMATION_SCHEMA` スキーマをダンプしません。

`mysqldump` は、`InnoDB CREATE TABLESPACE` ステートメントをダンプしません。

`mysqldump` は NDB Cluster `ndbinfo` 情報データベースをダンプしません。

mysqldump には、mysql データベースのダンプ用に `general_log` テーブルおよび `slow_query_log` テーブルを再作成するステートメントが含まれています。ログテーブルの内容はダンプされません。

権限が不十分なためビューのバックアップに問題が生じる場合は、[セクション25.9「ビューの制約」](#)の回避策を参照してください。

4.5.5 mysqlimport — データインポートプログラム

mysqlimport クライアントは、LOAD DATA SQL ステートメントへのコマンドラインインタフェースを提供します。mysqlimport のほとんどのオプションは、LOAD DATA 構文の句に直接対応しています。[セクション13.2.7「LOAD DATA ステートメント」](#)を参照してください。

mysqlimport は次のように起動します。

```
shell> mysqlimport [options] db_name textfile1 [textfile2 ...]
```

mysqlimport は、コマンド行で指定されたテキストファイルごとに、ファイル名の拡張子があればそれを取り除き、その結果をもとに、ファイル内容をインポートするテーブルの名前を決定します。たとえば、`patient.txt`、`patient.text`、および `patient` と名付けられたファイルはすべて `patient` と名付けられたテーブルにインポートされます。

mysqlimport は次のオプションをサポートします。これらはコマンド行またはオプションファイルの `[mysqlimport]` グループおよび `[client]` グループで指定できます。MySQL プログラムによって使用されるオプションファイルの詳細については、[セクション4.2.2.2「オプションファイルの使用」](#)を参照してください。

表 4.15 「mysqlimport のオプション」

オプション名	説明	導入	非推奨
<code>--bind-address</code>	指定されたネットワークインタフェースを使用して MySQL サーバーに接続		
<code>--columns</code>	このオプションはカンマによって分けられたカラム名のリストを値とします		
<code>--compress</code>	クライアントとサーバー間で送信される情報をすべて圧縮		8.0.18
<code>--compression-algorithms</code>	サーバーへの接続に許可される圧縮アルゴリズム	8.0.18	
<code>--debug</code>	デバッグログの書込み		
<code>--debug-check</code>	プログラムの終了時にデバッグ情報を出力		
<code>--debug-info</code>	プログラムの終了時に、デバッグ情報、メモリー、および CPU の統計を出力		
<code>--default-auth</code>	使用する認証プラグイン		
<code>--default-character-set</code>	デフォルト文字セットを指定		
<code>--defaults-extra-file</code>	通常のオプションファイルに加えて、名前付きオプションファイルを読み取ります		
<code>--defaults-file</code>	指名されたオプションファイルのみを読み取る		
<code>--defaults-group-suffix</code>	オプショングループのサフィクス値		

オプション名	説明	導入	非推奨
<code>--delete</code>	テキストファイルをインポートする前にテーブルを空にする		
<code>--enable-cleartext-plugin</code>	平文の認証プラグインを有効化		
<code>--fields-enclosed-by</code>	このオプションの意味は LOAD DATA の対応する句と同じです		
<code>--fields-escaped-by</code>	このオプションの意味は LOAD DATA の対応する句と同じです		
<code>--fields-optionally-enclosed-by</code>	このオプションの意味は LOAD DATA の対応する句と同じです		
<code>--fields-terminated-by</code>	このオプションの意味は LOAD DATA の対応する句と同じです		
<code>--force</code>	SQL エラーが発生しても続行		
<code>--get-server-public-key</code>	サーバーから RSA 公開キーをリクエスト		
<code>--help</code>	ヘルプメッセージを表示して終了		
<code>--host</code>	MySQL サーバーがあるホスト		
<code>--ignore</code>	<code>--replace</code> オプションの説明を参照		
<code>--ignore-lines</code>	データファイルの最初の N 行を無視		
<code>--lines-terminated-by</code>	このオプションの意味は LOAD DATA の対応する句と同じです		
<code>--local</code>	クライアントホストから入力ファイルをローカルで読み込む		
<code>--lock-tables</code>	テキストファイルを処理する前に、すべてのテーブルを書き込みロック		
<code>--login-path</code>	ログインパスオプションを <code>.mylogin.cnf</code> から読み取り		
<code>--low-priority</code>	テーブルのロード時に <code>LOW_PRIORITY</code> を使用します		
<code>--no-defaults</code>	オプションファイルを読み取らない		
<code>--password</code>	サーバーに接続する際に使用するパスワード		

オプション名	説明	導入	非推奨
<code>--pipe</code>	名前付きパイプを使用してサーバに接続する (Windows のみ)		
<code>--plugin-dir</code>	プラグインがインストールされているディレクトリ		
<code>--port</code>	接続用の TCP/IP ポート番号		
<code>--print-defaults</code>	デフォルトオプションの印刷		
<code>--protocol</code>	使用するトランスポートプロトコル		
<code>--replace</code>	<code>--replace</code> および <code>--ignore</code> オプションは、一意のキー値に関して既存の行と重複する入力行の処理を制御		
<code>--server-public-key-path</code>	RSA 公開鍵を含むファイルへのパス名		
<code>--shared-memory-base-name</code>	共有メモリー接続用の共有メモリー名 (Windows のみ)		
<code>--silent</code>	エラーが発生したときのみ出力を生成		
<code>--socket</code>	使用する Unix ソケットファイルまたは Windows 名前付きパイプ		
<code>--ssl-ca</code>	信頼できる SSL 認証局のリストを含むファイル		
<code>--ssl-capath</code>	信頼できる SSL 認証局の証明書ファイルを含むディレクトリ		
<code>--ssl-cert</code>	X.509 証明書を含むファイル		
<code>--ssl-cipher</code>	接続の暗号化に許可される暗号		
<code>--ssl-crl</code>	証明書失効リストを含むファイル		
<code>--ssl-crlpath</code>	証明書失効リストファイルを含むディレクトリ		
<code>--ssl-fips-mode</code>	クライアント側で FIPS モードを有効にするかどうか		
<code>--ssl-key</code>	X.509 キーを含むファイル		
<code>--ssl-mode</code>	サーバーへの接続に必要なセキュリティ状態		
<code>--tls-ciphersuites</code>	暗号化された接続に許可される TLSv1.3 暗号スイート	8.0.16	
<code>--tls-version</code>	暗号化された接続に許可される TLS プロトコル		
<code>--use-threads</code>	並列ファイルロードのスレッド数		

オプション名	説明	導入	非推奨
<code>--user</code>	サーバーへの接続時に使用する MySQL ユーザー名		
<code>--verbose</code>	冗長モード		
<code>--version</code>	バージョン情報を表示して終了		
<code>--zstd-compression-level</code>	zstd 圧縮を使用するサーバーへの接続の圧縮レベル	8.0.18	

- `--help, -?`

ヘルプメッセージを表示して終了します。

- `--bind-address=ip_address`

複数のネットワークインタフェースを持つコンピュータで、このオプションを使用して、MySQL サーバーへの接続に使用するインタフェースを選択します。

- `--character-sets-dir=dir_name`

文字セットがインストールされているディレクトリ。 [セクション10.15「文字セットの構成」](#) を参照してください。

- `--columns=column_list, -c column_list`

このオプションは、カンマ区切りのカラム名のリストをその値として使用します。カラム名の順序は、データファイルのカラムとテーブルのカラムを対応付ける方法を示しています。

- `--compress, -C`

可能であれば、クライアントとサーバーの間で送信されるすべての情報を圧縮します。 [セクション4.2.8「接続圧縮制御」](#) を参照してください。

MySQL 8.0.18 では、このオプションは非推奨です。MySQL の将来のバージョンで削除されることが予想されます。 [レガシー接続圧縮の構成](#) を参照してください。

- `--compression-algorithms=value`

サーバーへの接続に許可される圧縮アルゴリズム。使用可能なアルゴリズムは、`protocol_compression_algorithms` システム変数の場合と同じです。デフォルト値は `uncompressed` です。

詳細は、[セクション4.2.8「接続圧縮制御」](#) を参照してください。

このオプションは MySQL 8.0.18 で追加されました。

- `--debug=[debug_options], -# [debug_options]`

デバッグのログを書き込みます。一般的な `debug_options` 文字列は `d:t:o,file_name` です。デフォルトは `d:t:o` です。

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `--debug-check`

プログラムの終了時に、デバッグ情報を出力します。

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `--debug-info`

プログラムの終了時に、デバッグ情報とメモリーおよび CPU 使用率の統計を出力します。

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `--default-character-set=charset_name`

`charset_name` をデフォルト文字セットとして使用します。 [セクション10.15「文字セットの構成」](#) を参照してください。

- `--default-auth=plugin`

使用するクライアント側認証プラグインに関するヒント。 [セクション6.2.17「プラグブル認証」](#) を参照してください。

- `--defaults-extra-file=file_name`

このオプションファイルは、グローバルオプションファイルのあとに読み取りますが、(UNIX では) ユーザーオプションファイルの前に読み取るようにしてください。ファイルが存在しないかアクセスできない場合、エラーが発生します。`file_name` は、フルパス名でなく相対パス名として指定された場合、現行ディレクトリを基準にして解釈されます。

このオプションおよびその他のオプションファイルオプションの詳細は、 [セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--defaults-file=file_name`

指定されたオプションファイルのみ使用します。ファイルが存在しないかアクセスできない場合、エラーが発生します。`file_name` は、フルパス名でなく相対パス名として指定された場合、現行ディレクトリを基準にして解釈されます。

例外: `--defaults-file` でも、クライアントプログラムは `.mylogin.cnf` を読み取ります。

このオプションおよびその他のオプションファイルオプションの詳細は、 [セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--defaults-group-suffix=str`

通常のオプショングループだけでなく、通常の名前に `str` のサフィクスが付いたグループも読み取ります。たとえば、`mysqlimport` は通常 `[client]` グループおよび `[mysqlimport]` グループを読み取ります。`--defaults-group-suffix=_other` オプションを指定した場合、`mysqlimport` は `[client_other]` グループおよび `[mysqlimport_other]` グループも読み取ります。

このオプションおよびその他のオプションファイルオプションの詳細は、 [セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--delete, -D`

テキストファイルをインポートする前にテーブルを空にします。

- `--enable-cleartext-plugin`

`mysql_clear_password` 平文認証プラグインを有効にします。([セクション6.4.1.4「クライアント側クリアテキストプラグブル認証」](#) を参照してください。)

- `--fields-terminated-by=..., --fields-enclosed-by=..., --fields-optionally-enclosed-by=..., --fields-escaped-by=...`

これらのオプションは、`LOAD DATA` の対応する句と同じ意味を持ちます。 [セクション13.2.7「LOAD DATA ステートメント」](#) を参照してください。

- `--force, -f`

エラーを無視します。たとえば、テキストファイルのテーブルが存在しない場合、残ったファイルの処理を続行します。`--force` がない場合は、テーブルが存在しないと `mysqlimport` は終了します。

- `--get-server-public-key`

RSA キーペアベースのパスワード交換に必要な公開キーをサーバーにリクエストします。このオプションは、 `caching_sha2_password` 認証プラグインで認証されるクライアントに適用されます。そのプラグインの場合、サーバーは要求されないかぎり公開鍵を送信しません。このオプションは、そのプラグインで認証されないアカウントでは無視されます。クライアントがセキュアな接続を使用してサーバーに接続する場合と同様に、RSA ベースのパスワード交換を使用しない場合も無視されます。

`--server-public-key-path=file_name` が指定され、有効な公開キーファイルが指定されている場合は、 `--get-server-public-key` よりも優先されます。

`caching_sha2_password` プラグインの詳細は、[セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#) を参照してください。

- `--host=host_name, -h host_name`

指定されたホスト上の MySQL サーバーにデータをインポートします。デフォルトホストは `localhost` です。

- `--ignore, -i`

`--replace` オプションの説明を参照してください。

- `--ignore-lines=N`

データファイルの最初の `N` 行を無視します。

- `--lines-terminated-by=...`

このオプションの意味は、 `LOAD DATA` の対応する句と同じです。たとえば、復帰改行と改行のペアで終了する行を持つ Windows ファイルをインポートする場合、 `--lines-terminated-by="\r\n "` を使用してください。(コマンドインタプリタのエスケープ処理の規則によってはバックスラッシュを 2 つ使用しなければならない場合があります。) [セクション13.2.7「LOAD DATA ステートメント」](#) を参照してください。

- `--local, -L`

デフォルトでは、ファイルはサーバーホスト上のサーバーによって読み取られます。このオプションを指定すると、 `mysqlimport` はクライアントホスト上の入力ファイルをローカルに読み取ります。

`mysqlimport` で `LOCAL` ロード操作を正常に使用するには、サーバーでローカルロードが許可されている必要もあります。[セクション6.1.6「LOAD DATA LOCAL のセキュリティー上の考慮事項」](#) を参照してください

- `--lock-tables, -l`

テキストファイルを処理する前に、すべてのテーブルを書き込み用にロックします。これにより、すべてのテーブルがサーバー上で同期していることが保証されます。

- `--login-path=name`

`.mylogin.cnf` ログインパスファイルの指定されたログインパスからオプションを読み取ります。「ログインパス」は、接続先の MySQL サーバーおよび認証に使用するアカウントを指定するオプションを含むオプショングループです。ログインパスファイルを作成または変更するには、 `mysql_config_editor` ユーティリティを使用します。[セクション4.6.7「mysql_config_editor — MySQL 構成ユーティリティー」](#) を参照してください。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--low-priority`

テーブルのロード時に `LOW_PRIORITY` を使用します。これは、テーブルレベルロックのみを使用するストレージエンジン (`MyISAM` 、 `MEMORY` 、および `MERGE`) にのみ影響を与えます。

- `--no-defaults`

オプションファイルを読み取りません。オプションファイルから不明のオプションを読み取ることが原因でプログラムの起動に失敗する場合、 `--no-defaults` を使用して、オプションを読み取らないようにできます。

例外として、`.mylogin.cnf` ファイルは、存在する場合はすべての場合に読み取られます。これにより、`--no-defaults` が使用された場合にも、コマンド行よりも安全な方法でパスワードを指定できます。(`.mylogin.cnf` は `mysql_config_editor` ユーティリティーによって作成されます。 [セクション4.6.7「mysql_config_editor — MySQL 構成ユーティリティー」](#) を参照してください)。

このオプションおよびその他のオプションファイルオプションの詳細は、 [セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--password[=password], -p[password]`

サーバーへの接続に使用される MySQL アカウントのパスワード。パスワード値はオプションです。指定しない場合、`mysqlimport` によってプロンプトが表示されます。指定する場合は、`--password=` または `-p` とそれに続くパスワードの間にスペースなしが存在する必要があります。パスワードオプションを指定しない場合、デフォルトではパスワードは送信されません。

コマンド行でのパスワード指定は、セキュアでないと考えるべきです。コマンド行でパスワードを指定しないようにするには、オプションファイルを使用します。 [セクション6.1.2.1「パスワードセキュリティのためのエンドユーザーガイドライン」](#) を参照してください。

パスワードがなく、`mysqlimport` でパスワードの入力を求められないように明示的に指定するには、`--skip-password` オプションを使用します。

- `--pipe, -W`

Windows で、名前付きパイプを使用してサーバーに接続します。このオプションは、ネームパイプ接続をサポートするために `named_pipe` システム変数を有効にしてサーバーを起動した場合にのみ適用されます。また、接続を行うユーザーは、`named_pipe_full_access_group` システム変数で指定された Windows グループのメンバーである必要があります。

- `--plugin-dir=dir_name`

プラグインを検索するディレクトリ。このオプションは、`--default-auth` オプションを使用して認証プラグインを指定しても、`mysqlimport` がそれを検出しない場合に指定します。 [セクション6.2.17「プラグイン認証」](#) を参照してください。

- `--port=port_num, -P port_num`

TCP/IP 接続の場合、使用するポート番号。

- `--print-defaults`

プログラム名と、オプションファイルから受け取るすべてのオプションを出力します。

このオプションおよびその他のオプションファイルオプションの詳細は、 [セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

サーバーへの接続に使用するトランスポートプロトコル。これは、他の接続パラメータが通常、必要なプロトコル以外のプロトコルを使用する場合に便利です。許可される値の詳細は、 [セクション4.2.7「接続トランスポートプロトコル」](#) を参照してください。

- `--replace, -r`

`--replace` オプションおよび `--ignore` オプションは、既存の行の一意のキー値を重複させるような入力行の処理を制御します。`--replace` を指定した場合、同じ一意のキー値を持つ既存の行は新しい行で置換されます。`--ignore` を指定した場合、既存の行の一意のキー値を重複させる入力行はスキップされます。どちらのオプションも指定しなかった場合、重複するキー値が検出されるとエラーが発生し、残りのテキストファイルは無視されます。

- `--server-public-key-path=file_name`

RSA キーペアベースのパスワード交換のためにサーバーが必要とする公開キーのクライアント側コピーを含む、PEM 形式のファイルへのパス名。このオプションは、`sha256_password` または `caching_sha2_password` 認証

プラグインで認証されるクライアントに適用されます。これらのプラグインのいずれかで認証されないアカウントでは、このオプションは無視されます。クライアントがセキュアな接続を使用してサーバーに接続する場合と同様に、RSA ベースのパスワード交換を使用しない場合も無視されます。

`--server-public-key-path=file_name` が指定され、有効な公開キーファイルが指定されている場合は、`--get-server-public-key` よりも優先されます。

`sha256_password` の場合、このオプションは、MySQL が OpenSSL を使用して構築された場合にのみ適用されません。

`sha256_password` および `caching_sha2_password` プラグインの詳細は、[セクション6.4.1.3「SHA-256 プラガブル認証」](#) および [セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#) を参照してください。

- `--shared-memory-base-name=name`

Windows の場合、共有メモリを使用してローカルサーバーに接続するために使用する共有メモリ名。デフォルト値は `MYSQL` です。共有メモリ名では大文字と小文字が区別されます。

このオプションは、共有メモリ接続をサポートするために `shared_memory` システム変数を有効にしてサーバーを起動した場合にのみ適用されます。

- `--silent, -s`

サイレントモード。エラーが発生したときのみ出力を生成します。

- `--socket=path, -S path`

`localhost` への接続用に使用する、Unix ソケットファイル、または Windows では使用する名前付きパイプの名前。

Windows では、このオプションは、名前付きパイプ接続をサポートするために `named_pipe` システム変数を有効にしてサーバーを起動した場合にのみ適用されます。また、接続を行うユーザーは、`named_pipe_full_access_group` システム変数で指定された Windows グループのメンバーである必要があります。

- `--ssl*`

`--ssl` で始まるオプションは、SSL を使用してサーバーに接続するかどうかを指定し、SSL 鍵および証明書を検索する場所を指定します。[暗号化接続のコマンドオプション](#)を参照してください。

- `--ssl-fips-mode={OFF|ON|STRICT}`

クライアント側で FIPS モードを有効にするかどうかを制御します。`--ssl-fips-mode` オプションは、暗号化された接続の確立には使用されず、許可する暗号化操作に影響する点で、他の `--ssl-xxx` オプションとは異なります。[セクション6.8「FIPS のサポート」](#)を参照してください。

次の `--ssl-fips-mode` 値を使用できます:

- `OFF`: FIPS モードを無効にします。
- `ON`: FIPS モードを有効にします。
- `STRICT`: 「strict」 FIPS モードを有効にします。

注記

OpenSSL FIPS オブジェクトモジュールが使用できない場合、`--ssl-fips-mode` に許可される値は `OFF` のみです。この場合、`--ssl-fips-mode` を `ON` または `STRICT` に設定すると、クライアントは起動時に警告を生成し、FIPS 以外のモードで動作します。

- `--tls-ciphersuites=ciphersuite_list`

TLSv1.3 を使用する暗号化された接続に許可される暗号スイート。値は、コロンで区切られた 1 つ以上の暗号スイート名のリストです。このオプションに指定できる暗号スイートは、MySQL のコンパイルに使用される SSL ライブラリによって異なります。詳細は、[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#)を参照してください。

このオプションは MySQL 8.0.16 で追加されました。

- `--tls-version=protocol_list`

暗号化された接続に許可される TLS プロトコル。値は、1 つまたは複数のコンマ区切りプロトコル名のリストです。このオプションに指定できるプロトコルは、MySQL のコンパイルに使用される SSL ライブラリによって異なります。詳細は、[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#)を参照してください。

- `--user=user_name, -u user_name`

サーバーへの接続に使用する MySQL アカウントのユーザー名。

- `--use-threads=N`

N 個のスレッドを使用して複数のファイルを並行してロードします。

- `--verbose, -v`

冗長モード。プログラムの動作についてより多くの情報を出力します。

- `--version, -V`

バージョン情報を表示して終了します。

- `--zstd-compression-level=level`

`zstd` 圧縮アルゴリズムを使用するサーバーへの接続に使用する圧縮レベル。許可されるレベルは 1 から 22 で、大きい値は圧縮レベルの増加を示します。デフォルトの `zstd` 圧縮レベルは 3 です。圧縮レベルの設定は、`zstd` 圧縮を使用しない接続には影響しません。

詳細は、[セクション4.2.8「接続圧縮制御」](#)を参照してください。

このオプションは MySQL 8.0.18 で追加されました。

`mysqlimport` の使用方法を表すサンプルのセッションを次に示します。

```
shell> mysql -e 'CREATE TABLE impptest(id INT, n VARCHAR(30))' test
shell> ed
a
100 Max Sydow
101 Count Dracula
.
w impptest.txt
32
q
shell> od -c impptest.txt
0000000 1 0 0 \t M a x   S y d o w \n 1 0
0000020 1 \t C o u n t   D r a c u l a \n
0000040
shell> mysqlimport --local test impptest.txt
test.impptest: Records: 2 Deleted: 0 Skipped: 0 Warnings: 0
shell> mysql -e 'SELECT * FROM impptest' test
+----+-----+
| id | n      |
+----+-----+
| 100 | Max Sydow |
| 101 | Count Dracula |
+----+-----+
```

4.5.6 mysqldump — データベースバックアッププログラム

- [mysqldump の起動構文](#)
- [mysqldump オプションのサマリー](#)
- [mysqldump オプションの説明](#)
- [mysqldump オブジェクトの選択](#)

- [mysqldump パラレル処理](#)
- [mysqldump の制限事項](#)

`mysqldump` クライアントユーティリティは [logical backups](#) を実行し、元のデータベースオブジェクト定義およびテーブルデータを再現するために実行できる一連の SQL ステートメントを生成します。別の SQL サーバーにバックアップまたは転送するために、1 つ以上の MySQL データベースをダンプします。

ヒント

複数のスレッド、ファイル圧縮、進捗情報の表示、および Oracle Cloud Infrastructure Object Storage ストリーミングや MySQL データベースサービス 互換性チェックおよび変更などのクラウド機能で並列ダンプを提供する [MySQL Shell dump utilities](#) の使用を検討してください。ダンプは、[MySQL Shell load dump utilities](#) を使用して MySQL Server インスタンスまたは MySQL データベースサービス DB システムに簡単にインポートできます。MySQL Shell のインストール手順は、[here](#) にあります。

`mysqldump` の機能は次のとおりです:

- ダンププロセスを高速化するための、データベースおよびデータベース内のオブジェクトのパラレル処理
- ダンプするデータベースおよびデータベースオブジェクト (テーブル、ストアードプログラム、ユーザーアカウント) の制御の向上
- `mysql` システムデータベースへの挿入としてではなく、アカウント管理ステートメント (`CREATE USER`、`GRANT`) としてのユーザーアカウントのダンプ
- 圧縮出力を作成する機能
- 進捗インジケータ (値は推定)
- ダンプファイルのリロードでは、行の挿入後にインデックスを追加することで、`InnoDB` テーブルのセカンダリインデックスの作成が高速化されます

注記

`mysqldump` は、MySQL 5.7 で導入された MySQL 機能を使用するため、MySQL 5.7 以上で使用することを前提としています。

`mysqldump` には、ダンプされたテーブル、ダンプされたビュー用の `SHOW VIEW`、ダンプされたトリガー用の `TRIGGER`、および `--single-transaction` オプションが使用されていない場合は `LOCK TABLES` に対する少なくとも `SELECT` 権限が必要です。ユーザー定義をダンプするには、`mysql` システムデータベースに対する `SELECT` 権限が必要です。オプションの説明に示すように、一部のオプションではその他の権限が必要な場合があります。

ダンプファイルのリロードするには、ダンプファイルに含まれているステートメントを実行するために必要な権限 (これらのステートメントによって作成されたオブジェクトに対する適切な `CREATE` 権限など) が必要です。

注記

Windows で出力ディレクトリを使用して PowerShell を使用して作成されたダンプは、UTF-16 エンコーディングを持つファイルを作成します:

```
shell> mysqldump [options] > dump.sql
```

ただし、UTF-16 は接続文字セットとして許可されていないため ([セクション10.4「接続文字セットおよび照合順序」](#) を参照)、ダンプファイルを正しくロードできません。この問題を回避するには、ASCII 形式で出力を作成する `--result-file` オプションを使用します:

```
shell> mysqldump [options] --result-file=dump.sql
```

mysqldump の起動構文

デフォルトでは、`mysqldump` はすべてのデータベースをダンプします ([mysqldump の制限事項](#) に記載されている特定の例外を除く)。この動作を明示的に指定するには、`--all-databases` オプションを使用します:

```
shell> mysqldump --all-databases
```


単一のデータベースまたはそのデータベース内の特定のテーブルをダンプするには、コマンドラインでデータベースに名前を付け、オプションでテーブル名を指定します:

```
shell> mysqldump db_name
shell> mysqldump db_name tbl_name1 tbl_name2 ...
```

すべての名前引数をデータベース名として扱うには、`--databases` オプションを使用します:

```
shell> mysqldump --databases db_name1 db_name2 ...
```

デフォルトでは、付与テーブルを含む `mysql` システムデータベースをダンプしても、`mysqldump` はユーザーアカウント定義をダンプしません。付与テーブルの内容を論理定義として `CREATE USER` および `GRANT` ステートメントの形式でダンプするには、`--users` オプションを使用して、すべてのデータベースダンプを抑制します:

```
shell> mysqldump --exclude-databases=% --users
```

前述のコマンドの `%` は、`--exclude-databases` オプションのすべてのデータベース名に一致するワイルドカードです。

`mysqldump` では、データベース、テーブル、ストアプログラムおよびユーザー定義を含めるか除外するためのいくつかのオプションがサポートされています。[mysqldump オブジェクトの選択](#)を参照してください。

ダンプファイルをリロードするには、ダンプファイルに含まれるステートメントを実行します。たとえば、`mysql` クライアントを使用します:

```
shell> mysqldump [options] > dump.sql
shell> mysql < dump.sql
```

次の説明では、その他の `mysqldump` の使用例を示します。

`mysqldump` でサポートされているオプションのリストを表示するには、コマンド `mysqldump --help` を発行します。

mysqldump オプションのサマリー

`mysqldump` では、次のオプションがサポートされています。これらのオプションは、コマンドラインまたはオプションファイルの `[mysqldump]` および `[client]` グループで指定できます。(MySQL 8.0.20 より前は、`mysqldump` は `[mysqldump]` ではなく `[mysql_dump]` グループを読み取りました。8.0.20 では、`[mysql_dump]` は引き続き受け入れられますが、非推奨になりました。) MySQL プログラムによって使用されるオプションファイルの詳細については、[セクション4.2.2.2「オプションファイルの使用」](#)を参照してください。

表 4.16 「mysqldump のオプション」

オプション名	説明	導入	非推奨
<code>--add-drop-database</code>	DROP DATABASE ステートメントを各 CREATE DATABASE ステートメントの前に追加		
<code>--add-drop-table</code>	各 CREATE TABLE ステートメントの前に DROP TABLE ステートメントを追加		
<code>--add-drop-user</code>	各 CREATE USER ステートメントの前に DROP USER ステートメントを追加		
<code>--add-locks</code>	LOCK TABLES と UNLOCK TABLES ステートメントで各テーブルダンプを囲む		
<code>--all-databases</code>	すべてのデータベースのダンプ		
<code>--bind-address</code>	指定されたネットワークインタフェースを使用して MySQL サーバーに接続		

オプション名	説明	導入	非推奨
<code>--character-sets-dir</code>	文字セットがインストールされているディレクトリ		
<code>--column-statistics</code>	ANALYZE TABLE ステートメントを記述して統計ヒストグラムを生成		
<code>--complete-insert</code>	カラム名を含む完全な INSERT ステートメントを使用		
<code>--compress</code>	クライアントとサーバー間で送信される情報をすべて圧縮		8.0.18
<code>--compress-output</code>	出力圧縮アルゴリズム		
<code>--compression-algorithms</code>	サーバーへの接続に許可される圧縮アルゴリズム	8.0.18	
<code>--databases</code>	すべての名前引数をデータベース名として解釈		
<code>--debug</code>	デバッグログの書込み		
<code>--debug-check</code>	プログラムの終了時にデバッグ情報を出力		
<code>--debug-info</code>	プログラムの終了時に、デバッグ情報、メモリー、および CPU の統計を出力		
<code>--default-auth</code>	使用する認証プラグイン		
<code>--default-character-set</code>	デフォルト文字セットを指定		
<code>--default-parallelism</code>	パラレル処理のデフォルトのスレッド数		
<code>--defaults-extra-file</code>	通常のオプションファイルに加えて、名前付きオプションファイルを読み取ります		
<code>--defaults-file</code>	指名されたオプションファイルのみを読み取る		
<code>--defaults-group-suffix</code>	オプショングループのサフィクス値		
<code>--defer-table-indexes</code>	リロードの場合は、テーブルの行をロードするまでインデックス作成を延期		
<code>--events</code>	ダンプされたデータベースからのイベントのダンプ		
<code>--exclude-databases</code>	ダンプから除外するデータベース		
<code>--exclude-events</code>	ダンプから除外するイベント		
<code>--exclude-routines</code>	ダンプから除外するルーチン		
<code>--exclude-tables</code>	ダンプから除外するテーブル		

オプション名	説明	導入	非推奨
<code>--exclude-triggers</code>	ダンプから除外するトリガー		
<code>--exclude-users</code>	ダンプから除外するユーザー		
<code>--extended-insert</code>	複数行 INSERT 構文の使用		
<code>--get-server-public-key</code>	サーバーから RSA 公開キーをリクエスト		
<code>--help</code>	ヘルプメッセージを表示して終了		
<code>--hex-blob</code>	16 進数表記法を使用したバイナリカラムのダンプ		
<code>--host</code>	MySQL サーバーがあるホスト		
<code>--include-databases</code>	ダンプに含めるデータベース		
<code>--include-events</code>	ダンプに含めるイベント		
<code>--include-routines</code>	ダンプに含めるルーチン		
<code>--include-tables</code>	ダンプに含めるテーブル		
<code>--include-triggers</code>	ダンプに含めるトリガー		
<code>--include-users</code>	ダンプに含めるユーザー		
<code>--insert-ignore</code>	INSERT ステートメントではなく INSERT IGNORE を書き込みます		
<code>--log-error-file</code>	指定されたファイルに警告およびエラーを追加		
<code>--login-path</code>	ログインパスオプションを .mylogin.cnf から読み取り		
<code>--max-allowed-packet</code>	サーバーとの間で送受信するパケットの最大長		
<code>--net-buffer-length</code>	TCP/IP とソケット通信のバッファサイズ		
<code>--no-create-db</code>	CREATE DATABASE ステートメントを記述しないでください		
<code>--no-create-info</code>	各ダンプされたテーブルを再作成する CREATE TABLE ステートメントを書き出さない		
<code>--no-defaults</code>	オプションファイルを読み取らない		
<code>--parallel-schemas</code>	スキーマ処理の並列性の指定		
<code>--password</code>	サーバーに接続する際に使用するパスワード		
<code>--plugin-dir</code>	プラグインがインストールされているディレクトリ		

オプション名	説明	導入	非推奨
<code>--port</code>	接続用の TCP/IP ポート番号		
<code>--print-defaults</code>	デフォルトオプションの印刷		
<code>--protocol</code>	使用するトランスポートプロトコル		
<code>--replace</code>	INSERT ステートメントではなく REPLACE ステートメントを書き出す		
<code>--result-file</code>	指定されたファイルに出力		
<code>--routines</code>	ダンプされたデータベースからストアドルーチン (プロシージャとファンクション) をダンプ		
<code>--server-public-key-path</code>	RSA 公開鍵を含むファイルへのパス名		
<code>--set-charset</code>	SET NAMES default_character_set を出力に追加		
<code>--set-gtid-purged</code>	SET @@GLOBAL.GTID_PURGED を出力に追加するかどうか		
<code>--single-transaction</code>	単一トランザクション内のテーブルのダンプ		
<code>--skip-definer</code>	VIEW およびストアドプログラム CREATE ステートメントから DEFINER および SQL SECURITY 句を省略		
<code>--skip-dump-rows</code>	テーブルの行をダンプしない		
<code>--socket</code>	使用する Unix ソケットファイルまたは Windows 名前付きパイプ		
<code>--ssl-ca</code>	信頼できる SSL 認証局のリストを含むファイル		
<code>--ssl-capath</code>	信頼できる SSL 認証局の証明書ファイルを含むディレクトリ		
<code>--ssl-cert</code>	X.509 証明書を含むファイル		
<code>--ssl-cipher</code>	接続の暗号化に許可される暗号		
<code>--ssl-crl</code>	証明書失効リストを含むファイル		
<code>--ssl-crlpath</code>	証明書失効リストファイルを含むディレクトリ		
<code>--ssl-fips-mode</code>	クライアント側で FIPS モードを有効にするかどうか		

オプション名	説明	導入	非推奨
<code>--ssl-key</code>	X.509 キーを含むファイル		
<code>--ssl-mode</code>	サーバーへの接続に必要なセキュリティ状態		
<code>--tls-ciphersuites</code>	暗号化された接続に許可される TLSv1.3 暗号スイート	8.0.16	
<code>--tls-version</code>	暗号化された接続に許可される TLS プロトコル		
<code>--triggers</code>	ダンプされた各テーブルについて、トリガーをダンプする		
<code>--tz-utc</code>	SET TIME_ZONE='+00:00'をダンプファイルに追加		
<code>--user</code>	サーバーへの接続時に使用する MySQL ユーザー名		
<code>--users</code>	ユーザーアカウントのダンプ		
<code>--version</code>	バージョン情報を表示して終了		
<code>--watch-progress</code>	進捗インジケータの表示		
<code>--zstd-compression-level</code>	zstd 圧縮を使用するサーバーへの接続の圧縮レベル	8.0.18	

mysqlpump オプションの説明

- `--help, -?`

ヘルプメッセージを表示して終了します。

- `--add-drop-database`

各 `CREATE DATABASE` ステートメントの前に `DROP DATABASE` ステートメントを記述します。

注記

MySQL 8.0 では、`mysql` スキーマはエンドユーザーが削除できないシステムスキーマとみなされます。`--add-drop-database` が `--all-databases` または `--databases` とともに使用され、ダンプするスキーマのリストに `mysql` が含まれている場合、ダンプファイルにはダンプファイルのリロード時にエラーを引き起こす `DROP DATABASE `mysql`` ステートメントが含まれます。

かわりに、`--add-drop-database` を使用するには、ダンプするスキーマのリストとともに `--databases` を使用します。このリストには `mysql` は含まれません。

- `--add-drop-table`

各 `CREATE TABLE` ステートメントの前に `DROP TABLE` ステートメントを記述します。

- `--add-drop-user`

各 `CREATE USER` ステートメントの前に `DROP USER` ステートメントを記述します。

- `--add-locks`

`LOCK TABLES` ステートメントと `UNLOCK TABLES` ステートメントで各テーブルダンプを囲みます。これにより、ダンプファイルのリロードする際の挿入の速度が向上します。 [セクション8.2.5.1「INSERT ステートメントの最適化」](#) を参照してください。

異なるテーブルからの `INSERT` ステートメントをインターリーブでき、あるテーブルに対する挿入の終了後に `UNLOCK TABLES` が挿入が残っているテーブルのロックを解放する可能性があるため、このオプションは並列処理では機能しません。

`--add-locks` と `--single-transaction` は相互に排他的です。

- `--all-databases, -A`

すべてのデータベースをダンプします ([mysqldump の制限事項](#) に記載されている特定の例外を除く)。これは、他に明示的に指定されていない場合のデフォルトの動作です。

`--all-databases` と `--databases` は相互に排他的です。

注記

`--all-databases` との非互換性の詳細は、`--add-drop-database` の説明を参照してください。

MySQL 8.0 より前では、`mysqldump` および `mysqldump` の `--routines` および `--events` オプションは、`--all-databases` オプションの使用時にストアルーチンおよびイベントを含める必要はありませんでした: ダンプには `mysql` システムデータベースが含まれていたため、ストアルーチンおよびイベント定義を含む `mysql.proc` および `mysql.event` テーブルも含まれていました。MySQL 8.0 では、`mysql.event` テーブルおよび `mysql.proc` テーブルは使用されません。対応するオブジェクトの定義はデータディクショナリテーブルに格納されますが、これらのテーブルはダンプされません。 `--all-databases` を使用して作成されたダンプにストアルーチンおよびイベントを含めるには、`--routines` および `--events` オプションを明示的に使用します。

- `--bind-address=ip_address`

複数のネットワークインターフェースを持つコンピュータで、このオプションを使用して、MySQL サーバーへの接続に使用するインターフェースを選択します。

- `--character-sets-dir=path`

文字セットがインストールされているディレクトリ。 [セクション10.15「文字セットの構成」](#) を参照してください。

- `--column-statistics`

ダンプファイルのリロード時にダンプされたテーブルのヒストグラム統計を生成するために、出力に `ANALYZE TABLE` ステートメントを追加します。大規模なテーブルのヒストグラム生成には時間がかかる場合があるため、このオプションはデフォルトで無効になっています。

- `--complete-insert`

カラム名を含む完全な `INSERT` ステートメントを記述します。

- `--compress, -C`

可能であれば、クライアントとサーバーの間で送信されるすべての情報を圧縮します。 [セクション4.2.8「接続圧縮制御」](#) を参照してください。

MySQL 8.0.18 では、このオプションは非推奨です。MySQL の将来のバージョンで削除されることが予想されます。 [レガシー接続圧縮の構成](#) を参照してください。

- `--compress-output=algorithm`

デフォルトでは、`mysqldump` は出力を圧縮しません。このオプションは、指定されたアルゴリズムを使用して出力圧縮を指定します。許可されるアルゴリズムは、`LZ4` および `ZLIB` です。

圧縮された出力を解凍するには、適切なユーティリティが必要です。`lz4` および `openssl zlib` システムコマンドを使用できない場合、MySQL の配布には `lz4_decompress` および `zlib_decompress` ユーティリティが含まれ、`mysqldump` の出力は `--compress-output=LZ4` および `--compress-output=ZLIB` オプションを使用して圧縮されています。詳細は、[セクション4.8.1「lz4_decompress — mysqldump LZ4-Compressed 出力の解凍」](#) および [セクション4.8.3「zlib_decompress — mysqldump ZLIB 圧縮出力の解凍」](#) を参照してください。

- `--compression-algorithms=value`

サーバーへの接続に許可される圧縮アルゴリズム。使用可能なアルゴリズムは、`protocol_compression_algorithms` システム変数の場合と同じです。デフォルト値は `uncompressed` です。

詳細は、[セクション4.2.8「接続圧縮制御」](#) を参照してください。

このオプションは MySQL 8.0.18 で追加されました。

- `--databases, -B`

通常、`mysqldump` では、コマンドラインの名引数はデータベース名として扱われ、後続の名前はテーブル名として扱われます。このオプションを使用すると、名前引数をすべてデータベース名として処理します。`CREATE DATABASE` ステートメントは、各新規データベースの前に出力に含まれます。

`--all-databases` と `--databases` は相互に排他的です。

注記

`--databases` との非互換性の詳細は、`--add-drop-database` の説明を参照してください。

- `--debug[=debug_options], -# [debug_options]`

デバッグのログを書き込みます。一般的な `debug_options` 文字列は `d:t:o,file_name` です。デフォルトは `d:t:O,/tmp/mysqldump.trace` です。

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `--debug-check`

プログラムの終了時に、デバッグ情報を出力します。

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `--debug-info, -T`

プログラムの終了時に、デバッグ情報とメモリーおよび CPU 使用率の統計を出力します。

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `--default-auth=plugin`

使用するクライアント側認証プラグインに関するヒント。[セクション6.2.17「プラグブル認証」](#) を参照してください。

- `--default-character-set=charset_name`

`charset_name` をデフォルト文字セットとして使用します。[セクション10.15「文字セットの構成」](#) を参照してください。文字セットが指定されていない場合、`mysqldump` は `utf8` を使用します。

- `--default-parallelism=N`

各パラレル処理キューのデフォルトのスレッド数。デフォルトは 2 です。

`--parallel-schemas` オプションは並列性にも影響し、デフォルトのスレッド数をオーバーライドするために使用できます。詳細は、[mysqldump パラレル処理](#) を参照してください。

`--default-parallelism=0` および `--parallel-schemas` オプションがない場合、`mysqldump` はシングルスレッドプロセスとして実行され、キューは作成されません。

並列性を有効にすると、異なるデータベースからの出力をインターリーブできます。

- `--defaults-extra-file=file_name`

このオプションファイルは、グローバルオプションファイルのあとに読み取りますが、(UNIX では) ユーザーオプションファイルの前に読み取るようにしてください。ファイルが存在しないかアクセスできない場合、エラーが発生します。`file_name` は、フルパス名でなく相対パス名として指定された場合、現行ディレクトリを基準にして解釈されます。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--defaults-file=file_name`

指定されたオプションファイルのみ使用します。ファイルが存在しないかアクセスできない場合、エラーが発生します。`file_name` は、フルパス名でなく相対パス名として指定された場合、現行ディレクトリを基準にして解釈されます。

例外: `--defaults-file` でも、クライアントプログラムは `.mylogin.cnf` を読み取ります。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--defaults-group-suffix=str`

通常のオプショングループだけでなく、通常の名前に `str` のサフィクスが付いたグループも読み取ります。たとえば、`mysqldump` は通常、`[client]` および `[mysqldump]` グループを読み取ります。`--defaults-group-suffix=_other` オプションが指定されている場合、`mysqldump` は `[client_other]` および `[mysqldump_other]` グループも読み取ります。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--defer-table-indexes`

ダンプ出力で、行がロードされるまで各テーブルのインデックス作成を遅延します。これはすべてのストレージエンジンで機能しますが、`InnoDB` の場合はセカンダリインデックスにのみ適用されます。

このオプションはデフォルトで有効になっています。無効にするには `--skip-defer-table-indexes` を使用します。

- `--events`

ダンプされるデータベースのイベントスケジューライベントを出力に含めます。イベントダンプには、これらのデータベースに対する `EVENT` 権限が必要です。

`--events` を使用して生成される出力には、イベントを作成するための `CREATE EVENT` ステートメントが含まれています。

このオプションはデフォルトで有効になっています。無効にするには `--skip-events` を使用します。

- `--exclude-databases=db_list`

カンマ区切りのデータベース名のリストである `db_list` 内のデータベースはダンプしないでください。このオプションの複数のインスタンスは加算的です。詳細は、[mysqldump オブジェクトの選択](#) を参照してください。

- `--exclude-events=event_list`

カンマ区切りのイベント名のリストである `event_list` 内のデータベースはダンプしないでください。このオプションの複数のインスタンスは加算的です。詳細は、[mysqumpump オブジェクトの選択](#)を参照してください。

- `--exclude-routines=routine_list`

カンマ区切りのルーチン (ストアドプロシージャまたはファンクション) 名のリストである `routine_list` 内のイベントはダンプしないでください。このオプションの複数のインスタンスは加算的です。詳細は、[mysqumpump オブジェクトの選択](#)を参照してください。

- `--exclude-tables=table_list`

カンマ区切りのテーブル名のリストである `table_list` 内のテーブルはダンプしないでください。このオプションの複数のインスタンスは加算的です。詳細は、[mysqumpump オブジェクトの選択](#)を参照してください。

- `--exclude-triggers=trigger_list`

`trigger_list` でトリガーをダンプしないでください。これは、カンマ区切りのトリガー名のリストです。このオプションの複数のインスタンスは加算的です。詳細は、[mysqumpump オブジェクトの選択](#)を参照してください。

- `--exclude-users=user_list`

カンマ区切りのアカウント名のリストである `user_list` のユーザーアカウントはダンプしないでください。このオプションの複数のインスタンスは加算的です。詳細は、[mysqumpump オブジェクトの選択](#)を参照してください。

- `--extended-insert=N`

複数の `VALUES` リストを含む複数行構文を使用して `INSERT` ステートメントを記述します。これにより、ダンプファイルのサイズが小さくなり、ファイルがリロードされる際の挿入が高速化されます。

オプション値は、各 `INSERT` ステートメントに含める行数を示します。デフォルトは 250 です。値 1 は、テーブルの行ごとに 1 つの `INSERT` ステートメントを生成します。

- `--get-server-public-key`

RSA キーベースのパスワード交換に必要な公開キーをサーバーにリクエストします。このオプションは、`caching_sha2_password` 認証プラグインで認証されるクライアントに適用されます。そのプラグインの場合、サーバーは要求されないかぎり公開鍵を送信しません。このオプションは、そのプラグインで認証されないアカウントでは無視されます。クライアントがセキュアな接続を使用してサーバーに接続する場合と同様に、RSA ベースのパスワード交換を使用しない場合も無視されます。

`--server-public-key-path=file_name` が指定され、有効な公開キーファイルが指定されている場合は、`--get-server-public-key` よりも優先されます。

`caching_sha2_password` プラグインの詳細は、[セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#)を参照してください。

- `--hex-blob`

16 進表記を使用してバイナリカラムをダンプします (たとえば、`'abc'`は `0x616263` となります)。影響を受けるデータ型は、`binary character set` で使用する場合、`BINARY`、`VARBINARY`、`BLOB` 型、`BIT`、すべての空間データ型およびその他の非バイナリデータ型です。

- `--host=host_name, -h host_name`

与えられたホスト上の MySQL サーバーからデータをダンプします。

- `--include-databases=db_list`

`db_list` でデータベースをダンプします。これは、カンマ区切りのデータベース名のリストです。ダンプには、指定したデータベース内のすべてのオブジェクトが含まれます。このオプションの複数のインスタンスは加算的です。詳細は、[mysqumpump オブジェクトの選択](#)を参照してください。

- `--include-events=event_list`

`event_list` でイベントをダンプします。これは、カンマ区切りのイベント名のリストです。このオプションの複数のインスタンスは加算的です。詳細は、[mysqldump オブジェクトの選択](#)を参照してください。

- `--include-routines=routine_list`

`routine_list` でルーチンをダンプします。これは、コンマ区切りのルーチン (ストアドプロシージャまたはストアドファンクション) 名のリストです。このオプションの複数のインスタンスは加算的です。詳細は、[mysqldump オブジェクトの選択](#)を参照してください。

- `--include-tables=table_list`

カンマ区切りのテーブル名のリストである `table_list` のテーブルをダンプします。このオプションの複数のインスタンスは加算的です。詳細は、[mysqldump オブジェクトの選択](#)を参照してください。

- `--include-triggers=trigger_list`

`trigger_list` でトリガーをダンプします。これは、カンマ区切りのトリガー名のリストです。このオプションの複数のインスタンスは加算的です。詳細は、[mysqldump オブジェクトの選択](#)を参照してください。

- `--include-users=user_list`

`user_list` でユーザーアカウントをダンプします。これは、カンマ区切りのユーザー名のリストです。このオプションの複数のインスタンスは加算的です。詳細は、[mysqldump オブジェクトの選択](#)を参照してください。

- `--insert-ignore`

`INSERT` ステートメントではなく、`INSERT IGNORE` ステートメントを書き出します。

- `--log-error-file=file_name`

警告およびエラーを、指名されたファイルに追加することによってログに記録します。このオプションを指定しない場合、`mysqldump` は警告およびエラーを標準エラー出力に書き込みます。

- `--login-path=name`

`.mylogin.cnf` ログインパスファイルの指定されたログインパスからオプションを読み取ります。「ログインパス」は、接続先の MySQL サーバーおよび認証に使用するアカウントを指定するオプションを含むオプショングループです。ログインパスファイルを作成または変更するには、[mysql_config_editor](#) ユーティリティを使用します。[セクション4.6.7「mysql_config_editor — MySQL 構成ユーティリティー」](#)を参照してください。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--max-allowed-packet=N`

クライアント/サーバー通信用のバッファの最大サイズ。デフォルトは 24M バイト、最大は 1G バイトです。

- `--net-buffer-length=N`

クライアント/サーバー通信用のバッファの初期サイズ。(`--extended-insert` オプションと同様に) 複数行の `INSERT` ステートメントを作成する場合、`mysqldump` は `N` バイトまでの行を作成します。このオプションを使用して値を増やす場合は、MySQL サーバーの `net_buffer_length` システム変数の値がこの値以上であることを確認してください。

- `--no-create-db`

それ以外の場合は出力に含まれる可能性のある `CREATE DATABASE` ステートメントを抑制します。

- `--no-create-info, -t`

ダンプされた各テーブルを作成する `CREATE TABLE` ステートメントを記述しないでください。

- `--no-defaults`

オプションファイルを読み取りません。オプションファイルから不明のオプションを読み取ることが原因でプログラムの起動に失敗する場合、`--no-defaults` を使用して、オプションを読み取らないようにすることができます。

例外として、`.mylogin.cnf` ファイルは、存在する場合はすべての場合に読み取られます。これにより、`--no-defaults` が使用された場合に、コマンド行よりも安全な方法でパスワードを指定できます。`.mylogin.cnf` は `mysql_config_editor` ユーティリティーによって作成されます。セクション4.6.7「`mysql_config_editor` — MySQL 構成ユーティリティー」を参照してください。

このオプションおよびその他のオプションファイルオプションの詳細は、セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」を参照してください。

- `--parallel-schemas=[N:]db_list`

`db_list` でデータベースを処理するためのキューを作成します。これは、カンマ区切りのデータベース名のリストです。N が指定されている場合、キューは N スレッドを使用します。N が指定されていない場合、`--default-parallelism` オプションによってキュースレッドの数が決定されます。

このオプションの複数のインスタンスでは、複数のキューが作成されます。また、`mysqumpump` では、どの `--parallel-schemas` オプションでも指定されていないデータベースに使用するデフォルトキューが作成され、コマンドオプションでユーザー定義が選択されている場合はユーザー定義がダンプされます。詳細は、`mysqumpump` [パラレル処理](#) を参照してください。

- `--password[=password], -p[password]`

サーバーへの接続に使用される MySQL アカウントのパスワード。パスワード値はオプションです。指定しない場合、`mysqumpump` によってプロンプトが表示されます。指定する場合は、`--password=` または `-p` とそれに続くパスワードの間にスペースなしが存在する必要があります。パスワードオプションを指定しない場合、デフォルトではパスワードは送信されません。

コマンド行でのパスワード指定は、セキュアでないと考えべきです。コマンド行でパスワードを指定しないようにするには、オプションファイルを使用します。セクション6.1.2.1「パスワードセキュリティのためのエンドユーザーガイドライン」を参照してください。

パスワードがなく、`mysqumpump` でパスワードの入力を求められないように明示的に指定するには、`--skip-password` オプションを使用します。

- `--plugin-dir=dir_name`

プラグインを検索するディレクトリ。このオプションは、`--default-auth` オプションを使用して認証プラグインを指定しても、`mysqumpump` がそれを検出しない場合に指定します。セクション6.2.17「[プラグイン認証](#)」を参照してください。

- `--port=port_num, -P port_num`

TCP/IP 接続の場合、使用するポート番号。

- `--print-defaults`

プログラム名と、オプションファイルから受け取るすべてのオプションを出力します。

このオプションおよびその他のオプションファイルオプションの詳細は、セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」を参照してください。

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

サーバーへの接続に使用するトランスポートプロトコル。これは、他の接続パラメータが通常、必要なプロトコル以外のプロトコルを使用する場合に便利です。許可される値の詳細は、セクション4.2.7「[接続トランスポートプロトコル](#)」を参照してください。

- `--replace`

INSERT ステートメントではなく REPLACE ステートメントを書き込みます。

- `--result-file=file_name`

指定されたファイルに出力を転送します。ダンプの生成中にエラーが発生しても、結果ファイルが作成され以前の内容は上書きされます。

このオプションは、改行 `\n` 文字が `\r\n` キャリッジリターン/改行シーケンスに変換されないようにするために、Windows で使用する必要があります。

- `--routines`

ダンプされるデータベースのストアドルーチン (プロシージャおよび関数) を出力に含めます。このオプションには、グローバル `SELECT` 権限が必要です。

`--routines` を使用して生成される出力には、ルーチンを作成するための `CREATE PROCEDURE` および `CREATE FUNCTION` ステートメントが含まれています。

このオプションはデフォルトで有効になっています。無効にするには `--skip-routines` を使用します。

- `--server-public-key-path=file_name`

RSA キーベースのパスワード交換のためにサーバーが必要とする公開キーのクライアント側コピーを含む、PEM 形式のファイルへのパス名。このオプションは、`sha256_password` または `caching_sha2_password` 認証プラグインで認証されるクライアントに適用されます。これらのプラグインのいずれかで認証されないアカウントでは、このオプションは無視されます。クライアントがセキュアな接続を使用してサーバーに接続する場合と同様に、RSA ベースのパスワード交換を使用しない場合も無視されます。

`--server-public-key-path=file_name` が指定され、有効な公開キーファイルが指定されている場合は、`--get-server-public-key` よりも優先されます。

`sha256_password` の場合、このオプションは、MySQL が OpenSSL を使用して構築された場合にのみ適用されません。

`sha256_password` および `caching_sha2_password` プラグインの詳細は、[セクション6.4.1.3「SHA-256 プラガブル認証」](#) および [セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#) を参照してください。

- `--set-charset`

`SET NAMES default_character_set` を出力に書き込みます。

このオプションはデフォルトで有効となっています。これを無効にして `SET NAMES` ステートメントを抑制するには、`--skip-set-charset` を使用します。

- `--set-gtid-purged=value`

このオプションを使用すると、`SET @@GLOBAL.gtid_purged` ステートメントを出力に追加するかどうかを指定することで、ダンプファイルに書き込まれるグローバルトランザクション ID (GTID) 情報を制御できます。このオプションを使用すると、ダンプファイルのリロード中にバイナリロギングを無効にするステートメントが出力に書き込まれることもあります。

次の表は、許可されるオプション値を示しています。デフォルト値は `AUTO` です。

値	意味
<code>OFF</code>	出力に <code>SET</code> ステートメントを追加しません。
<code>ON</code>	出力に <code>SET</code> ステートメントを追加します。サーバーで GTID が有効になっていない場合は、エラーが発生します。
<code>AUTO</code>	サーバーで GTID が有効になっている場合に、出力に <code>SET</code> ステートメントを追加します。

`--set-gtid-purged` オプションは、ダンプファイルがリロードされる時にバイナリロギングに次の影響を与えます:

- `--set-gtid-purged=OFF: SET @@SESSION.SQL_LOG_BIN=0;` は出力に追加されません。

- `--set-gtid-purged=ON: SET @@SESSION.SQL_LOG_BIN=0;`が出力に追加されます。
- `--set-gtid-purged=AUTO:` バックアップするサーバーで GTID が有効になっている場合 (つまり、`AUTO` が `ON` と評価される場合)、`SET @@SESSION.SQL_LOG_BIN=0;`が出力に追加されます。

- `--single-transaction`

このオプションは、データのダンプ前に、トランザクション分離モードを `REPEATABLE READ` に設定し、`START TRANSACTION SQL` ステートメントをサーバーに送信します。これは、`InnoDB` などのトランザクションテーブルの場合にかぎって便利です。その場合、アプリケーションをブロックすることなく、`START TRANSACTION` が発行された時点のデータベースの一貫した状態をダンプするからです。

このオプションを使用する場合、一貫した状態でダンプされるのは `InnoDB` テーブルのみだということに留意してください。たとえば、このオプションの使用中にダンプされた `MyISAM` テーブルまたは `MEMORY` テーブルは状態が変化する可能性があります。

`--single-transaction` ダンプの処理中、ダンプファイルが正当である (テーブルの内容とバイナリログ座標が正しい) ことを保証するために、ほかの接続で `ALTER TABLE`、`CREATE TABLE`、`DROP TABLE`、`RENAME TABLE`、`TRUNCATE TABLE` ステートメントを使用しないようにしてください。一貫性読み取りはこれらのステートメントから分離されないため、ダンプされるテーブルでこれらを使用すると、`mysqldump` によって実行される `SELECT` がテーブルの内容を取得して誤った内容を取得したり、失敗したりする可能性があります。

`--add-locks` と `--single-transaction` は相互に排他的です。

- `--skip-definer`

ビューおよびストアードプログラムの `CREATE` ステートメントから `DEFINER` 句および `SQL SECURITY` 句を省略します。ダンプファイルをリロードすると、デフォルトの `DEFINER` および `SQL SECURITY` 値を使用するオブジェクトが作成されます。セクション25.6「ストアードオブジェクトのアクセス制御」を参照してください。

- `--skip-dump-rows, -d`

テーブルの行をダンプしません。

- `--socket=path, -S path`

`localhost` への接続用に使用する、Unix ソケットファイル、または Windows では使用する名前付きパイプの名前。

Windows では、このオプションは、名前付きパイプ接続をサポートするために `named_pipe` システム変数を有効にしてサーバーを起動した場合にのみ適用されます。また、接続を行うユーザーは、`named_pipe_full_access_group` システム変数で指定された Windows グループのメンバーである必要があります。

- `--ssl*`

`--ssl` で始まるオプションは、SSL を使用してサーバーに接続するかどうかを指定し、SSL 鍵および証明書を検索する場所を指定します。暗号化接続のコマンドオプションを参照してください。

- `--ssl-fips-mode={OFF|ON|STRICT}`

クライアント側で FIPS モードを有効にするかどうかを制御します。 `--ssl-fips-mode` オプションは、暗号化された接続の確立には使用されず、許可する暗号化操作に影響する点で、他の `--ssl-xxx` オプションとは異なります。 [セクション6.8「FIPS のサポート」](#) を参照してください。

次の `--ssl-fips-mode` 値を使用できます:

- **OFF**: FIPS モードを無効にします。
- **ON**: FIPS モードを有効にします。
- **STRICT**: 「strict」 FIPS モードを有効にします。

注記

OpenSSL FIPS オブジェクトモジュールが使用できない場合、 `--ssl-fips-mode` に許可される値は **OFF** のみです。 この場合、 `--ssl-fips-mode` を **ON** または **STRICT** に設定すると、クライアントは起動時に警告を生成し、FIPS 以外のモードで動作します。

- `--tls-ciphersuites=ciphersuite_list`

TLSv1.3 を使用する暗号化された接続に許可される暗号スイート。 値は、コロンで区切られた 1 つ以上の暗号スイート名のリストです。 このオプションに指定できる暗号スイートは、MySQL のコンパイルに使用される SSL ライブラリによって異なります。 詳細は、 [セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#) を参照してください。

このオプションは MySQL 8.0.16 で追加されました。

- `--tls-version=protocol_list`

暗号化された接続に許可される TLS プロトコル。 値は、1 つまたは複数のコンマ区切りプロトコル名のリストです。 このオプションに指定できるプロトコルは、MySQL のコンパイルに使用される SSL ライブラリによって異なります。 詳細は、 [セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#) を参照してください。

- `--triggers`

ダンプされる各テーブルのトリガーを出力に含めます。

このオプションはデフォルトで有効になっています。 無効にするには `--skip-triggers` を使用します。

- `--tz-utc`

このオプションを使用すると、異なるタイムゾーンのサーバー間で **TIMESTAMP** カラムをダンプおよびリロードできます。 `mysqldump` は、接続タイムゾーンを UTC に設定し、 `SET TIME_ZONE='+00:00'` をダンプファイルに追加します。 このオプションを指定しない場合、ソースサーバーと宛先サーバーにローカルなタイムゾーンで **TIMESTAMP** カラムがダンプおよびリロードされ、サーバーが異なるタイムゾーンにある場合は値が変更される可能性があります。 `--tz-utc` は、夏時間による変更からも保護します。

このオプションはデフォルトで有効になっています。 無効にするには `--skip-tz-utc` を使用します。

- `--user=user_name, -u user_name`

サーバーへの接続に使用する MySQL アカウントのユーザー名。

- `--users`

CREATE USER および **GRANT** ステートメントの形式でユーザーアカウントを論理定義としてダンプします。

ユーザー定義は、`mysql` システムデータベースの付与テーブルに格納されます。 デフォルトでは、`mysqldump` は付与テーブルを `mysql` データベースダンプに含めません。 付与テーブルの内容を論理定義としてダンプするには、 `--users` オプションを使用して、すべてのデータベースダンプを抑制します:

```
shell> mysqldump --exclude-databases=% --users
```

- `--version, -V`

バージョン情報を表示して終了します。

- `--watch-progress`

テーブル、行およびその他のオブジェクトの完了数と合計数に関する情報を提供する進捗インジケータを定期的に表示します。

このオプションはデフォルトで有効になっています。無効にするには `--skip-watch-progress` を使用します。

- `--zstd-compression-level=level`

`zstd` 圧縮アルゴリズムを使用するサーバーへの接続に使用する圧縮レベル。許可されるレベルは 1 から 22 で、大きい値は圧縮レベルの増加を示します。デフォルトの `zstd` 圧縮レベルは 3 です。圧縮レベルの設定は、`zstd` 圧縮を使用しない接続には影響しません。

詳細は、[セクション4.2.8「接続圧縮制御」](#)を参照してください。

このオプションは MySQL 8.0.18 で追加されました。

mysqldump オブジェクトの選択

`mysqldump` には、複数のオブジェクトタイプのフィルタリングを可能にし、ダンプするオブジェクトを柔軟に制御できる包含および除外オプションのセットがあります:

- `--include-databases` および `--exclude-databases` は、データベースおよびその中のすべてのオブジェクトに適用されます。
- `--include-tables` および `--exclude-tables` はテーブルに適用されます。これらのオプションは、トリガー固有のオプションが指定されていないかぎり、テーブルに関連付けられたトリガーにも影響します。
- `--include-triggers` および `--exclude-triggers` はトリガーに適用されます。
- `--include-routines` および `--exclude-routines` は、ストアドプロシージャおよびストアドファンクションに適用されます。ルーチンオプションがストアドプロシージャ名に一致する場合は、同じ名前のストアドファンクションにも一致します。
- `--include-events` および `--exclude-events` は、イベントスケジューライベントに適用されます。
- `--include-users` および `--exclude-users` はユーザーアカウントに適用されます。

包含または除外オプションは複数回指定できます。効果は加法的です。これらのオプションの順序は関係ありません。

各包含および除外オプションの値は、適切なオブジェクトタイプのカンマ区切りの名前リストです。例:

```
--exclude-databases=test,world
--include-tables=customer,invoice
```

オブジェクト名にはワイルドカード文字を使用できます:

- `%` は、ゼロ文字以上の任意のシーケンスに一致します。
- `_` は、任意の単一文字に一致します。

たとえば、`--include-tables=t%,__tmp` は、`t` で始まるすべてのテーブル名と、`tmp` で終わるすべての 5 文字のテーブル名を照合します。

ユーザーの場合、ホスト部分なしで指定された名前は、`%` の暗黙的なホストで解釈されます。たとえば、`u1` と `u1@%` は同等です。これは、MySQL で一般的に適用される同値化と同じです ([セクション6.2.4「アカウント名の指定」](#)を参照)。

包含オプションと除外オプションは、次のように相互作用します:

- デフォルトでは、包含オプションも除外オプションも指定せずに、`mysqldump` はすべてのデータベースをダンプします (`mysqldump` の [制限事項](#) に記載されている特定の例外を除く)。
- 除外オプションが指定されていない場合は、`include` として指定されたオブジェクトのみがダンプされます。
- 包含オプションがない場合に除外オプションを指定すると、`excluded` という名前のオブジェクトを除くすべてのオブジェクトがダンプされます。
- 包含オプションと除外オプションが指定されている場合、除外されたオブジェクトと含まれているオブジェクトの名前が指定されていないオブジェクトはすべてダンプされません。他のすべてのオブジェクトはダンプされます。

複数のデータベースがダンプされている場合は、オブジェクト名をデータベース名で修飾することで、特定のデータベース内のテーブル、トリガー、およびルーチンに名前を付けることができます。次のコマンドは、データベース `db1` および `db2` をダンプしますが、テーブル `db1.t1` および `db2.t2` は除外します:

```
shell> mysqldump --include-databases=db1,db2 --exclude-tables=db1.t1,db2.t2
```

次のオプションは、ダンプするデータベースを指定する別の方法を提供します:

- `--all-databases` オプションは、すべてのデータベースをダンプします (`mysqldump` の [制限事項](#) に記載されている特定の例外を除く)。これは、オブジェクトオプションをまったく指定しないことと同等です (デフォルトの `mysqldump` アクションでは、すべてをダンプします)。
`--include-databases=%` は `--all-databases` に似ていますが、`--all-databases` の例外であるデータベースも含めて、すべてのデータベースをダンプ対象として選択します。
- `--databases` オプションを使用すると、`mysqldump` はすべての名前引数をダンプするデータベースの名前として扱います。これは、同じデータベースを指定する `--include-databases` オプションと同等です。

mysqldump パラレル処理

`mysqldump` では、並列性を使用して同時処理を実現できます。データベース間の同時実行性 (複数のデータベースを同時にダンプする場合) およびデータベース内の同時実行性 (特定のデータベースから複数のオブジェクトを同時にダンプする場合) を選択できます。

デフォルトでは、`mysqldump` は 2 つのスレッドで 1 つのキューを設定します。追加のキューを作成し、デフォルトキューを含む各キューに割り当てられるスレッドの数を制御できます:

- `--default-parallelism=N` では、各キューに使用されるデフォルトのスレッド数を指定します。このオプションがない場合、`N` は 2 です。

デフォルトキューでは、常にデフォルトのスレッド数が使用されます。その他のキューでは、特に指定しないかぎり、デフォルトのスレッド数が使用されます。

- `--parallel-schemas=[N:]db_list` は、`db_list` で指定されたデータベースをダンプするための処理キューを設定し、オプションでキューが使用するスレッド数を指定します。`db_list` は、カンマ区切りのデータベース名のリストです。オプション引数が `N:` で始まる場合、キューは `N` スレッドを使用します。それ以外の場合は、`--default-parallelism` オプションによってキュースレッドの数が決まります。

`--parallel-schemas` オプションの複数のインスタンスでは、複数のキューが作成されます。

データベースリスト内の名前には、フィルタリングオプションでサポートされているのと同じ `%` および `_` ワイルドカード文字を含めることができます ([mysqldump オブジェクトの選択](#) を参照)。

`mysqldump` は、`--parallel-schemas` オプションで明示的に指定されていないデータベースを処理したり、コマンドオプションで選択された場合にユーザー定義をダンプしたりするために、デフォルトキューを使用します。

通常、複数のキューでは、`mysqldump` はキューで処理されるデータベースのセット間で並列性を使用して、複数のデータベースを同時にダンプします。複数のスレッドを使用するキューの場合、`mysqldump` はデータベース内で並列性を使用して、特定のデータベースから複数のオブジェクトを同時にダンプします。例外が発生する可能性があります。たとえば、`mysqldump` は、データベース内のオブジェクトのサーバーリストからキューを取得している間、キューをブロックする場合があります。

並列性を有効にすると、異なるデータベースからの出力をインターリーブできます。たとえば、並列でダンプされた複数のテーブルからの `INSERT` ステートメントはインターリーブできます。ステートメントは特定の順序で書き込ま

れません。出力ステートメントはオブジェクト名をデータベース名で修飾するか、必要に応じて `USE` ステートメントの前に付けるため、再ロードには影響しません。

並列度の粒度は単一のデータベースオブジェクトです。たとえば、単一のテーブルを複数のスレッドを使用してパラレルにダンプすることはできません。

例:

```
shell> mysqlpump --parallel-schemas=db1,db2 --parallel-schemas=db3
```

`mysqlpump` は、`db1` および `db2` を処理するキュー、`db3` を処理する別のキュー、および他のすべてのデータベースを処理するデフォルトキューを設定します。すべてのキューは 2 つのスレッドを使用します。

```
shell> mysqlpump --parallel-schemas=db1,db2 --parallel-schemas=db3
--default-parallelism=4
```

これは前の例と同じですが、すべてのキューが 4 つのスレッドを使用する点が異なります。

```
shell> mysqlpump --parallel-schemas=5:db1,db2 --parallel-schemas=3:db3
```

`db1` および `db2` のキューは 5 つのスレッドを使用し、`db3` のキューは 3 つのスレッドを使用し、デフォルトキューは 2 つのスレッドのデフォルトを使用します。

特殊なケースとして、`--default-parallelism=0` で `--parallel-schemas` オプションを指定しない場合、`mysqlpump` はシングルスレッドプロセスとして実行され、キューは作成されません。

mysqlpump の制限事項

デフォルトでは、`mysqlpump` は `performance_schema`、`ndbinfo` または `sys` スキーマをダンプしません。これらのいずれかをダンプするには、コマンドラインで明示的に名前を付けます。 `--databases` または `--include-databases` オプションを使用して名前を付けることもできます。

`mysqlpump` は、`INFORMATION_SCHEMA` スキーマをダンプしません。

`mysqlpump` は、`InnoDB CREATE TABLESPACE` ステートメントをダンプしません。

`mysqlpump` は、`CREATE USER` および `GRANT` ステートメントを使用して、ユーザーアカウントを論理形式でダンプします (`--include-users` または `--users` オプションを使用する場合など)。このため、`mysql` システムデータベースのダンプには、デフォルトでユーザー定義を含む付与テーブルは含まれません: `user`, `db`, `tables_priv`, `columns_priv`, `procs_priv` または `proxies_priv`。付与テーブルをダンプするには、`mysql` データベースに名前を付け、その後にテーブル名を付けます:

```
shell> mysqlpump mysql user db ...
```

4.5.7 mysqlshow — データベース、テーブル、およびカラム情報の表示

`mysqlshow` クライアントは、どのデータベース、そのテーブル、またはテーブルのカラムまたはインデックスが存在するかを迅速に確認するために使用できます。

`mysqlshow` は複数の SQL `SHOW` ステートメントに対してコマンド行インタフェースを提供します。 [セクション 13.7.7 「SHOW ステートメント」](#) を参照してください。それらステートメントを直接使用することで同じ情報を得られます。たとえば、`mysql` クライアントプログラムからそれらを発行できます。

`mysqlshow` は次のように起動します。

```
shell> mysqlshow [options] [db_name [tbl_name [col_name]]]
```

- データベースを指定しないと、データベース名のリストが表示されます。
- テーブルを指定しないと、データベース内のすべての一致するテーブルが表示されます。
- カラムを指定しないと、テーブル内のすべての一致するカラムとカラムの型が表示されます。

出力は、ユーザーが何らかの権限を所持しているデータベース、テーブル、またはカラムの名前のみを表示します。

最後の引数にシェルまたは SQL ワイルドカード文字 (`*`, `?`, `%` または `_`) が含まれている場合は、ワイルドカードに一致する名前のみが表示されます。データベース名にアンダースコアが含まれている場合、適切なテーブルまたはカラム

のリストを取得するには、それらをバックスラッシュでエスケープする必要があります (Unix シェルによっては 2 つ 必要があります)。* および ? の文字は、SQL % および _ のワイルドカード文字に変換されます。これは名前に _ を含むテーブルのカラムを表示しようとした際に問題を引き起こす場合があります。なぜなら、この場合 `mysqlshow` はパターンに一致するテーブル名のみを表示するからです。これは、コマンド行上で最後に % を別個の引数として追加することで簡単に修正できます。

`mysqlshow` は次のオプションをサポートします。これらはコマンド行またはオプションファイルの `[mysqlshow]` グループおよび `[client]` グループで指定できます。MySQL プログラムによって使用されるオプションファイルの詳細については、[セクション4.2.2.2「オプションファイルの使用」](#)を参照してください。

表 4.17 「mysqlshow のオプション」

オプション名	説明	導入	非推奨
<code>--bind-address</code>	指定されたネットワークインタフェースを使用して MySQL サーバーに接続		
<code>--compress</code>	クライアントとサーバー間で送信される情報をすべて圧縮		8.0.18
<code>--compression-algorithms</code>	サーバーへの接続に許可される圧縮アルゴリズム	8.0.18	
<code>--count</code>	テーブルごとの行の数を表示		
<code>--debug</code>	デバッグログの書込み		
<code>--debug-check</code>	プログラムの終了時にデバッグ情報を出力		
<code>--debug-info</code>	プログラムの終了時に、デバッグ情報、メモリー、および CPU の統計を出力		
<code>--default-auth</code>	使用する認証プラグイン		
<code>--default-character-set</code>	デフォルト文字セットを指定		
<code>--defaults-extra-file</code>	通常のオプションファイルに加えて、名前付きオプションファイルを読み取ります		
<code>--defaults-file</code>	指名されたオプションファイルのみを読み取る		
<code>--defaults-group-suffix</code>	オプショングループのサフィクス値		
<code>--enable-cleartext-plugin</code>	平文の認証プラグインを有効化		
<code>--get-server-public-key</code>	サーバーから RSA 公開キーをリクエスト		
<code>--help</code>	ヘルプメッセージを表示して終了		
<code>--host</code>	MySQL サーバーがあるホスト		
<code>--keys</code>	テーブルインデックスを表示します		
<code>--login-path</code>	ログインパスオプションを <code>.mylogin.cnf</code> から読み取り		

オプション名	説明	導入	非推奨
<code>--no-defaults</code>	オプションファイルを読み取らない		
<code>--password</code>	サーバーに接続する際に使用するパスワード		
<code>--pipe</code>	名前付きパイプを使用してサーバに接続する (Windows のみ)		
<code>--plugin-dir</code>	プラグインがインストールされているディレクトリ		
<code>--port</code>	接続用の TCP/IP ポート番号		
<code>--print-defaults</code>	デフォルトオプションの印刷		
<code>--protocol</code>	使用するトランスポートプロトコル		
<code>--server-public-key-path</code>	RSA 公開鍵を含むファイルへのパス名		
<code>--shared-memory-base-name</code>	共有メモリー接続用の共有メモリー名 (Windows のみ)		
<code>--show-table-type</code>	テーブルのタイプを示すカラムを表示		
<code>--socket</code>	使用する Unix ソケットファイルまたは Windows 名前付きパイプ		
<code>--ssl-ca</code>	信頼できる SSL 認証局のリストを含むファイル		
<code>--ssl-capath</code>	信頼できる SSL 認証局の証明書ファイルを含むディレクトリ		
<code>--ssl-cert</code>	X.509 証明書を含むファイル		
<code>--ssl-cipher</code>	接続の暗号化に許可される暗号		
<code>--ssl-crl</code>	証明書失効リストを含むファイル		
<code>--ssl-crlpath</code>	証明書失効リストファイルを含むディレクトリ		
<code>--ssl-fips-mode</code>	クライアント側で FIPS モードを有効にするかどうか		
<code>--ssl-key</code>	X.509 キーを含むファイル		
<code>--ssl-mode</code>	サーバーへの接続に必要なセキュリティ状態		
<code>--status</code>	各テーブルの追加情報を表示		
<code>--tls-ciphersuites</code>	暗号化された接続に許可される TLSv1.3 暗号スイート	8.0.16	
<code>--tls-version</code>	暗号化された接続に許可される TLS プロトコル		

オプション名	説明	導入	非推奨
<code>--user</code>	サーバーへの接続時に使用する MySQL ユーザー名		
<code>--verbose</code>	冗長モード		
<code>--version</code>	バージョン情報を表示して終了		
<code>--zstd-compression-level</code>	zstd 圧縮を使用するサーバーへの接続の圧縮レベル	8.0.18	

- `--help, -?`

ヘルプメッセージを表示して終了します。

- `--bind-address=ip_address`

複数のネットワークインターフェースを持つコンピュータで、このオプションを使用して、MySQL サーバーへの接続に使用するインターフェースを選択します。

- `--character-sets-dir=dir_name`

文字セットがインストールされているディレクトリ。 [セクション10.15「文字セットの構成」](#) を参照してください。

- `--compress, -C`

可能であれば、クライアントとサーバーの間で送信されるすべての情報を圧縮します。 [セクション4.2.8「接続圧縮制御」](#) を参照してください。

MySQL 8.0.18 では、このオプションは非推奨です。MySQL の将来のバージョンで削除されることが予想されます。 [レガシー接続圧縮の構成](#) を参照してください。

- `--compression-algorithms=value`

サーバーへの接続に許可される圧縮アルゴリズム。使用可能なアルゴリズムは、`protocol_compression_algorithms` システム変数の場合と同じです。デフォルト値は `uncompressed` です。

詳細は、[セクション4.2.8「接続圧縮制御」](#) を参照してください。

このオプションは MySQL 8.0.18 で追加されました。

- `--count`

テーブルごとの行の数を表示します。これは `MyISAM` でないテーブルでは、遅い場合があります。

- `--debug=[debug_options], -# [debug_options]`

デバッグのログを書き込みます。一般的な `debug_options` 文字列は `d:t:o,file_name` です。デフォルトは `d:t:o` です。

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `--debug-check`

プログラムの終了時に、デバッグ情報を出力します。

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `--debug-info`

プログラムの終了時に、デバッグ情報とメモリーおよび CPU 使用率の統計を出力します。

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `--default-character-set=charset_name`

`charset_name` をデフォルト文字セットとして使用します。 [セクション10.15「文字セットの構成」](#) を参照してください。

- `--default-auth=plugin`

使用するクライアント側認証プラグインに関するヒント。 [セクション6.2.17「プラグブル認証」](#) を参照してください。

- `--defaults-extra-file=file_name`

このオプションファイルは、グローバルオプションファイルのあとに読み取りますが、(UNIX では) ユーザーオプションファイルの前に読み取るようにしてください。ファイルが存在しないかアクセスできない場合、エラーが発生します。`file_name` は、フルパス名でなく相対パス名として指定された場合、現行ディレクトリを基準にして解釈されます。

このオプションおよびその他のオプションファイルオプションの詳細は、 [セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--defaults-file=file_name`

指定されたオプションファイルのみ使用します。ファイルが存在しないかアクセスできない場合、エラーが発生します。`file_name` は、フルパス名でなく相対パス名として指定された場合、現行ディレクトリを基準にして解釈されます。

例外: `--defaults-file` でも、クライアントプログラムは `.mylogin.cnf` を読み取ります。

このオプションおよびその他のオプションファイルオプションの詳細は、 [セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--defaults-group-suffix=str`

通常のオプショングループだけでなく、通常の名前に `str` のサフィクスが付いたグループも読み取ります。たとえば、`mysqlshow` は通常 `[client]` グループおよび `[mysqlshow]` グループを読み取ります。`--defaults-group-suffix=_other` オプションを指定した場合、`mysqlshow` は `[client_other]` グループおよび `[mysqlshow_other]` グループも読み取ります。

このオプションおよびその他のオプションファイルオプションの詳細は、 [セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--enable-cleartext-plugin`

`mysql_clear_password` 平文認証プラグインを有効にします。([セクション6.4.1.4「クライアント側クリアテキストプラグブル認証」](#) を参照してください。)

- `--get-server-public-key`

キーペアベースのパスワード交換に使用する RSA 公開キーをサーバーにリクエストします。このオプションは、 `caching_sha2_password` 認証プラグインで認証されるアカウントを使用してサーバーに接続するクライアントに適用されます。このようなアカウントによる接続の場合、サーバーは要求されないかぎり公開鍵をクライアントに送信しません。このオプションは、そのプラグインで認証されないアカウントでは無視されます。クライアントがセキュアな接続を使用してサーバーに接続する場合と同様に、RSA ベースのパスワード交換が不要な場合も無視されます。

`--server-public-key-path=file_name` が指定され、有効な公開キーファイルが指定されている場合は、`--get-server-public-key` よりも優先されます。

`caching_sha2_password` プラグインの詳細は、 [セクション6.4.1.2「SHA-2 プラグブル認証のキャッシュ」](#) を参照してください。

- `--host=host_name, -h host_name`

指定されたホストの MySQL サーバーに接続します。

- `--keys, -k`

テーブルインデックスを表示します。

- `--login-path=name`

`.mylogin.cnf` ログインパスファイルの指定されたログインパスからオプションを読み取ります。「ログインパス」は、接続先の MySQL サーバーおよび認証に使用するアカウントを指定するオプションを含むオプショングループです。ログインパスファイルを作成または変更するには、`mysql_config_editor` コーティリティーを使用します。セクション4.6.7「`mysql_config_editor` — MySQL 構成ユーティリティー」を参照してください。

このオプションおよびその他のオプションファイルオプションの詳細は、セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」を参照してください。

- `--no-defaults`

オプションファイルを読み取りません。オプションファイルから不明のオプションを読み取ることが原因でプログラムの起動に失敗する場合、`--no-defaults` を使用して、オプションを読み取らないようにできます。

例外として、`.mylogin.cnf` ファイルは、存在する場合はすべての場合に読み取られます。これにより、`--no-defaults` が使用された場合にも、コマンド行よりも安全な方法でパスワードを指定できます。`.mylogin.cnf` は `mysql_config_editor` コーティリティーによって作成されます。セクション4.6.7「`mysql_config_editor` — MySQL 構成ユーティリティー」を参照してください。

このオプションおよびその他のオプションファイルオプションの詳細は、セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」を参照してください。

- `--password[=password], -p[password]`

サーバーへの接続に使用される MySQL アカウントのパスワード。パスワード値はオプションです。指定しない場合、`mysqlshow` によってプロンプトが表示されます。指定する場合は、`--password=` または `-p` とそれに続くパスワードの間にスペースなしが存在する必要があります。パスワードオプションを指定しない場合、デフォルトではパスワードは送信されません。

コマンド行でのパスワード指定は、セキュアでないと考えるべきです。コマンド行でパスワードを指定しないようにするには、オプションファイルを使用します。セクション6.1.2.1「パスワードセキュリティのためのエンドユーザーガイドライン」を参照してください。

パスワードがなく、`mysqlshow` でパスワードの入力を求められないように明示的に指定するには、`--skip-password` オプションを使用します。

- `--pipe, -W`

Windows で、名前付きパイプを使用してサーバーに接続します。このオプションは、ネームパイプ接続をサポートするために `named_pipe` システム変数を有効にしてサーバーを起動した場合にのみ適用されます。また、接続を行うユーザーは、`named_pipe_full_access_group` システム変数で指定された Windows グループのメンバーである必要があります。

- `--plugin-dir=dir_name`

プラグインを検索するディレクトリ。このオプションは、`--default-auth` オプションを使用して認証プラグインを指定しても、`mysqlshow` がそれを検出しない場合に指定します。セクション6.2.17「プラグイン認証」を参照してください。

- `--port=port_num, -P port_num`

TCP/IP 接続の場合、使用するポート番号。

- `--print-defaults`

プログラム名と、オプションファイルから受け取るすべてのオプションを出力します。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

サーバーへの接続に使用するトランスポートプロトコル。これは、他の接続パラメータが通常、必要なプロトコル以外のプロトコルを使用する場合に便利です。許可される値の詳細は、[セクション4.2.7「接続トランスポートプロトコル」](#)を参照してください。

- `--server-public-key-path=file_name`

RSA キーベースのパスワード交換のためにサーバーが必要とする公開キーのクライアント側コピーを含む、PEM 形式のファイルへのパス名。このオプションは、`sha256_password` または `caching_sha2_password` 認証プラグインで認証されるクライアントに適用されます。これらのプラグインのいずれかで認証されないアカウントでは、このオプションは無視されます。クライアントがセキュアな接続を使用してサーバーに接続する場合と同様に、RSA ベースのパスワード交換を使用しない場合も無視されます。

`--server-public-key-path=file_name` が指定され、有効な公開キーファイルが指定されている場合は、`--get-server-public-key` よりも優先されます。

`sha256_password` の場合、このオプションは、MySQL が OpenSSL を使用して構築された場合にのみ適用されます。

`sha256_password` および `caching_sha2_password` プラグインの詳細は、[セクション6.4.1.3「SHA-256 プラガブル認証」](#) および [セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#)を参照してください。

- `--shared-memory-base-name=name`

Windows の場合、共有メモリを使用してローカルサーバーに接続するために使用する共有メモリ名。デフォルト値は `MYSQL` です。共有メモリー名では大文字と小文字が区別されます。

このオプションは、共有メモリー接続をサポートするために `shared_memory` システム変数を有効にしてサーバーを起動した場合にのみ適用されます。

- `--show-table-type, -t`

`SHOW FULL TABLES` と同様に、テーブルの型を示すカラムを表示します。型は `BASE TABLE` または `VIEW` です。

- `--socket=path, -S path`

`localhost` への接続用に使用する、Unix ソケットファイル、または Windows では使用する名前付きパイプの名前。

Windows では、このオプションは、名前付きパイプ接続をサポートするために `named_pipe` システム変数を有効にしてサーバーを起動した場合にのみ適用されます。また、接続を行うユーザーは、`named_pipe_full_access_group` システム変数で指定された Windows グループのメンバーである必要があります。

- `--ssl*`

`--ssl` で始まるオプションは、SSL を使用してサーバーに接続するかどうかを指定し、SSL 鍵および証明書を検索する場所を指定します。[暗号化接続のコマンドオプション](#)を参照してください。

- `--ssl-fips-mode={OFF|ON|STRICT}`

クライアント側で FIPS モードを有効にするかどうかを制御します。`--ssl-fips-mode` オプションは、暗号化された接続の確立には使用されず、許可する暗号化操作に影響する点で、他の `--ssl-xxx` オプションとは異なります。[セクション6.8「FIPS のサポート」](#)を参照してください。

次の `--ssl-fips-mode` 値を使用できます:

- `OFF`: FIPS モードを無効にします。
- `ON`: FIPS モードを有効にします。

- **STRICT**: 「strict」 FIPS モードを有効にします。

注記

OpenSSL FIPS オブジェクトモジュールが使用できない場合、`--ssl-fips-mode` に許可される値は `OFF` のみです。この場合、`--ssl-fips-mode` を `ON` または `STRICT` に設定すると、クライアントは起動時に警告を生成し、FIPS 以外のモードで動作します。

- `--status, -i`

各テーブルの追加情報を表示します。

- `--tls-ciphersuites=ciphersuite_list`

TLSv1.3 を使用する暗号化された接続に許可される暗号スイート。値は、コロンで区切られた 1 つ以上の暗号スイート名のリストです。このオプションに指定できる暗号スイートは、MySQL のコンパイルに使用される SSL ライブラリによって異なります。詳細は、[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#)を参照してください。

このオプションは MySQL 8.0.16 で追加されました。

- `--tls-version=protocol_list`

暗号化された接続に許可される TLS プロトコル。値は、1 つまたは複数のコンマ区切りプロトコル名のリストです。このオプションに指定できるプロトコルは、MySQL のコンパイルに使用される SSL ライブラリによって異なります。詳細は、[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#)を参照してください。

- `--user=user_name, -u user_name`

サーバーへの接続に使用する MySQL アカウントのユーザー名。

- `--verbose, -v`

冗長モード。プログラムの動作についてより多くの情報を出力します。このオプションは情報量を増加させるために複数回使用できます。

- `--version, -V`

バージョン情報を表示して終了します。

- `--zstd-compression-level=level`

`zstd` 圧縮アルゴリズムを使用するサーバーへの接続に使用する圧縮レベル。許可されるレベルは 1 から 22 で、大きい値は圧縮レベルの増加を示します。デフォルトの `zstd` 圧縮レベルは 3 です。圧縮レベルの設定は、`zstd` 圧縮を使用しない接続には影響しません。

詳細は、[セクション4.2.8「接続圧縮制御」](#)を参照してください。

このオプションは MySQL 8.0.18 で追加されました。

4.5.8 mysqlslap — ロードエミュレーションクライアント

`mysqlslap` は MySQL サーバーのクライアント負荷をエミュレートし、各段階のタイミングをレポートする診断プログラムです。複数のクライアントがサーバーにアクセスしているかのように作動します。

`mysqlslap` は次のように起動します。

```
shell> mysqlslap [options]
```

`--create` または `--query` などのオプションを使用すると、SQL ステートメントを含む文字列やステートメントを含むファイルを指定できます。ファイルを指定した場合、デフォルトでは各行に 1 つのステートメントを含んでいなければなりません。(つまり、暗黙的なステートメント区切り文字は改行文字です。)異なる区切り文字を指定するには、`--delimiter` オプションを使用します。これにより、複数行にわたるステートメントを指定したり、1 行に複数のス

コメントを配置したりできます。ファイルにコメントを含めることはできません。mysqlslap はそれらを理解しません。

mysqlslap は次の 3 段階で実行されます。

1. テストに使用するスキーマ、テーブル、およびオプションでストアプログラムまたはデータを作成します。この段階では、1 つのクライアント接続を使用します。
2. 負荷テストを実行します。この段階では、多数のクライアント接続を使用できます。
3. クリーンアップ (接続の解除、指定した場合はテーブルの削除) を実行します。この段階では、1 つのクライアント接続を使用します。

例:

50 台のクライアントがクエリーを実行し、それぞれ 200 の選択を行うような、create および query SQL ステートメントを提供します (コマンドは単一行に入力します)。

```
mysqlslap --delimiter=";"
--create="CREATE TABLE a (b int);INSERT INTO a VALUES (23)"
--query="SELECT * FROM a" --concurrency=50 --iterations=200
```

mysqlslap に、2 つの INT カラムと 3 つの VARCHAR カラムから成るテーブルを含む query SQL ステートメントを構築させます。5 台のクライアントを使ってそれぞれ 20 回ずつクエリーを実行します。テーブルを作成したり、データを挿入したりしないでください (直前のテストのスキーマとデータを使用します)。

```
mysqlslap --concurrency=5 --iterations=20
--number-int-cols=2 --number-char-cols=3
--auto-generate-sql
```

プログラムに、指定のファイルから create、insert、および query SQL ステートメントをロードするように指示します。この場合の create.sql ファイルには、';' で区切られた複数のテーブル作成ステートメントと ';' で区切られた複数の挿入ステートメントが含まれています。--query ファイルには、';' で区切られた複数のクエリーを含める必要があります。5 台のクライアントを使用して、すべての load ステートメントを実行してから、query ファイル内のすべてのクエリーを実行します (それぞれ 5 回ずつ)。

```
mysqlslap --concurrency=5
--iterations=5 --query=query.sql --create=create.sql
--delimiter=";"
```

mysqlslap は次のオプションをサポートします。これらはコマンド行またはオプションファイルの [mysqlslap] グループおよび [client] グループで指定できます。MySQL プログラムによって使用されるオプションファイルの詳細については、セクション 4.2.2.2 「オプションファイルの使用」を参照してください。

表 4.18 「mysqlslap のオプション」

オプション名	説明	導入	非推奨
<code>--auto-generate-sql</code>	SQL ステートメントがファイルおよびコマンドオプションを使用して指定されない場合、自動的に生成		
<code>--auto-generate-sql-add-autoincrement</code>	AUTO_INCREMENT カラムを自動生成されたテーブルに追加		
<code>--auto-generate-sql-execute-number</code>	自動的に生成するクエリーの数を指定		
<code>--auto-generate-sql-guid-primary</code>	自動生成されたテーブルに GUID ベースの主キーを追加		
<code>--auto-generate-sql-load-type</code>	テストの負荷タイプを指定します		

オプション名	説明	導入	非推奨
<code>--auto-generate-sql-secondary-indexes</code>	自動生成されたテーブルに追加するセカンダリインデックスの数を指定		
<code>--auto-generate-sql-unique-query-number</code>	自動テスト用に生成する異なるクエリーの数		
<code>--auto-generate-sql-unique-write-number</code>	<code>--auto-generate-sql-write-number</code> 用に生成する異なるクエリーの数		
<code>--auto-generate-sql-write-number</code>	各スレッドで実行する行挿入の回数		
<code>--commit</code>	コミットの前に実行するステートメントの数		
<code>--compress</code>	クライアントとサーバー間で送信される情報をすべて圧縮		8.0.18
<code>--compression-algorithms</code>	サーバーへの接続に許可される圧縮アルゴリズム	8.0.18	
<code>--concurrency</code>	SELECT ステートメントを発行する際、シミュレートするクライアントの数		
<code>--create</code>	テーブルの作成に使用するステートメントを含むファイルまたは文字列		
<code>--create-schema</code>	テストを実行するスキーマ		
<code>--csv</code>	カンマ区切りの値の形式で出力を生成		
<code>--debug</code>	デバッグログの書込み		
<code>--debug-check</code>	プログラムの終了時にデバッグ情報を出力		
<code>--debug-info</code>	プログラムの終了時に、デバッグ情報、メモリー、および CPU の統計を出力		
<code>--default-auth</code>	使用する認証プラグイン		
<code>--defaults-extra-file</code>	通常のオプションファイルに加えて、名前付きオプションファイルを読み取ります		
<code>--defaults-file</code>	指名されたオプションファイルのみを読み取る		
<code>--defaults-group-suffix</code>	オプショングループのサフィクス値		
<code>--delimiter</code>	SQL ステートメントで使用する区切り文字		
<code>--detach</code>	N 個のステートメントが終わるごとに各接続を切り離す (閉じてからふたたび開く)		
<code>--enable-cleartext-plugin</code>	平文の認証プラグインを有効化		

オプション名	説明	導入	非推奨
<code>--engine</code>	テーブルの作成に使用するストレージエンジン		
<code>--get-server-public-key</code>	サーバーから RSA 公開キーをリクエスト		
<code>--help</code>	ヘルプメッセージを表示して終了		
<code>--host</code>	MySQL サーバーがあるホスト		
<code>--iterations</code>	実行するテストの回数		
<code>--login-path</code>	ログインパスオプションを <code>.mylogin.cnf</code> から読み取り		
<code>--no-defaults</code>	オプションファイルを読み取らない		
<code>--no-drop</code>	テスト実行中に作成されたスキーマを削除しない		
<code>--number-char-cols</code>	<code>--auto-generate-sql</code> が指定された場合に使用する VARCHAR カラムの数		
<code>--number-int-cols</code>	<code>--auto-generate-sql</code> が指定された場合に使用する INT カラムの数		
<code>--number-of-queries</code>	各クライアントのクエリー数をおよその数に制限		
<code>--only-print</code>	データベースに接続しない。 <code>mysqlslap</code> が実行したであろう内容を出力するのみ		
<code>--password</code>	サーバーに接続する際に使用するパスワード		
<code>--pipe</code>	名前付きパイプを使用してサーバに接続する (Windows のみ)		
<code>--plugin-dir</code>	プラグインがインストールされているディレクトリ		
<code>--port</code>	接続用の TCP/IP ポート番号		
<code>--post-query</code>	テスト完了後に実行するステートメントを含むファイルまたは文字列		
<code>--post-system</code>	テスト完了後に <code>system()</code> を使用して実行する文字列		
<code>--pre-query</code>	テストの実施前に実行するステートメントを含むファイルまたは文字列		
<code>--pre-system</code>	テストの実施前に <code>system()</code> を使用して実行する文字列		
<code>--print-defaults</code>	デフォルトオプションの印刷		

オプション名	説明	導入	非推奨
<code>--protocol</code>	使用するトランスポートプロトコル		
<code>--query</code>	データ取得のために使用する SELECT ステートメントを含むファイルまたは文字列		
<code>--server-public-key-path</code>	RSA 公開鍵を含むファイルへのパス名		
<code>--shared-memory-base-name</code>	共有メモリー接続用の共有メモリー名 (Windows のみ)		
<code>--silent</code>	サイレントモード		
<code>--socket</code>	使用する Unix ソケットファイルまたは Windows 名前付きパイプ		
<code>--sql-mode</code>	クライアントセッションの SQL モードを設定		
<code>--ssl-ca</code>	信頼できる SSL 認証局のリストを含むファイル		
<code>--ssl-capath</code>	信頼できる SSL 認証局の証明書ファイルを含むディレクトリ		
<code>--ssl-cert</code>	X.509 証明書を含むファイル		
<code>--ssl-cipher</code>	接続の暗号化に許可される暗号		
<code>--ssl-crl</code>	証明書失効リストを含むファイル		
<code>--ssl-crlpath</code>	証明書失効リストファイルを含むディレクトリ		
<code>--ssl-fips-mode</code>	クライアント側で FIPS モードを有効にするかどうか		
<code>--ssl-key</code>	X.509 キーを含むファイル		
<code>--ssl-mode</code>	サーバーへの接続に必要なセキュリティ状態		
<code>--tls-ciphersuites</code>	暗号化された接続に許可される TLSv1.3 暗号スイート	8.0.16	
<code>--tls-version</code>	暗号化された接続に許可される TLS プロトコル		
<code>--user</code>	サーバーへの接続時に使用する MySQL ユーザー名		
<code>--verbose</code>	冗長モード		
<code>--version</code>	バージョン情報を表示して終了		
<code>--zstd-compression-level</code>	zstd 圧縮を使用するサーバーへの接続の圧縮レベル	8.0.18	

- `--help, -?`
ヘルプメッセージを表示して終了します。
- `--auto-generate-sql, -a`
SQL ステートメントがファイルおよびコマンドオプションを使用して指定されない場合、自動的に生成します。
- `--auto-generate-sql-add-autoincrement`
`AUTO_INCREMENT` カラムを自動生成されたテーブルに追加します。
- `--auto-generate-sql-execute-number=N`
自動的に生成するクエリーの数を指定します。
- `--auto-generate-sql-guid-primary`
自動生成されたテーブルに GUID ベースの主キーを追加します。
- `--auto-generate-sql-load-type=type`
テストの負荷タイプを指定します。許可される値は、`read` (テーブルのスキャン)、`write` (テーブルに挿入)、`key` (主キーの読み取り)、`update` (主キーの更新)、または `mixed` (挿入とスキャンして選択が半分ずつ) です。デフォルトは `mixed` です。
- `--auto-generate-sql-secondary-indexes=N`
自動生成されたテーブルに追加するセカンダリインデックスの数を指定します。デフォルトでは、何も追加されません。
- `--auto-generate-sql-unique-query-number=N`
自動テスト用に生成する異なるクエリーの数。たとえば、1000 回の選択を行う `key` テストを実施する場合、このオプションの値を 1000 にして一意のクエリーを 1000 個実行することも、値を 50 にして異なるクエリーを 50 回行うこともできます。デフォルトは 10 です。
- `--auto-generate-sql-unique-write-number=N`
`--auto-generate-sql-write-number` 用に生成する異なるクエリーの数を指定します。デフォルトは 10 です。
- `--auto-generate-sql-write-number=N`
実行する行挿入の数。デフォルトは 100 です。
- `--commit=N`
コミットの前に実行するステートメントの数。デフォルトは 0 (コミットは行われません) です。
- `--compress, -C`
可能であれば、クライアントとサーバーの間で送信されるすべての情報を圧縮します。 [セクション4.2.8「接続圧縮制御」](#)を参照してください。
MySQL 8.0.18 では、このオプションは非推奨です。MySQL の将来のバージョンで削除されることが予想されます。 [レガシー接続圧縮の構成](#)を参照してください。
- `--compression-algorithms=value`
サーバーへの接続に許可される圧縮アルゴリズム。使用可能なアルゴリズムは、`protocol_compression_algorithms` システム変数の場合と同じです。デフォルト値は `uncompressed` です。
詳細は、 [セクション4.2.8「接続圧縮制御」](#)を参照してください。
このオプションは MySQL 8.0.18 で追加されました。
- `--concurrency=N, -c N`

シミュレートするパラレルクライアントの数。

- `--create=value`

テーブルの作成に使用するステートメントを含むファイルまたは文字列。

- `--create-schema=value`

テストを実行するスキーマ。

注記

`--auto-generate-sql` オプションも指定されている場合、`mysqlslap` はテスト実行の最後にスキーマを削除します。これを避けるには、`--no-drop` オプションも使用します。

- `--csv=[file_name]`

カンマ区切りの値の形式で出力を生成します。出力は指定されたファイルか、ファイルが指定されていない場合標準出力に送られます。

- `--debug=[debug_options], -# [debug_options]`

デバッグのログを書き込みます。一般的な `debug_options` 文字列は `d:t:o,file_name` です。デフォルトは `d:t:o/tmp/mysqlslap.trace` です。

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `--debug-check`

プログラムの終了時に、デバッグ情報を出力します。

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `--debug-info, -T`

プログラムの終了時に、デバッグ情報とメモリーおよび CPU 使用率の統計を出力します。

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `--default-auth=plugin`

使用するクライアント側認証プラグインに関するヒント。 [セクション6.2.17「プラグブル認証」](#) を参照してください。

- `--defaults-extra-file=file_name`

このオプションファイルは、グローバルオプションファイルのあとに読み取りますが、(UNIX では) ユーザーオプションファイルの前に読み取るようにしてください。ファイルが存在しないかアクセスできない場合、エラーが発生します。`file_name` は、フルパス名でなく相対パス名として指定された場合、現行ディレクトリを基準にして解釈されます。

このオプションおよびその他のオプションファイルオプションの詳細は、 [セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--defaults-file=file_name`

指定されたオプションファイルのみ使用します。ファイルが存在しないかアクセスできない場合、エラーが発生します。`file_name` は、フルパス名でなく相対パス名として指定された場合、現行ディレクトリを基準にして解釈されます。

例外: `--defaults-file` でも、クライアントプログラムは `.mylogin.cnf` を読み取ります。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--defaults-group-suffix=str`

通常のオプショングループだけでなく、通常の名前に `str` のサフィクスが付いたグループも読み取ります。たとえば、`mysqlslap` は通常 `[client]` グループおよび `[mysqlslap]` グループを読み取ります。`--defaults-group-suffix=_other` オプションを指定した場合、`mysqlslap` は `[client_other]` グループおよび `[mysqlslap_other]` グループも読み取ります。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--delimiter=str, -F str`

ファイルまたはコマンドオプションを使用して提供される SQL ステートメントで使用される区切り文字。

- `--detach=N`

`N` 個のステートメントごとに各接続を切り離します (閉じてからふたたび開きます)。デフォルトは 0 (接続は切り離されません) です。

- `--enable-cleartext-plugin`

`mysql_clear_password` 平文認証プラグインを有効にします。([セクション6.4.1.4「クライアント側クリアテキストプラグイン認証」](#)を参照してください。)

- `--engine=engine_name, -e engine_name`

テーブルの作成に使用するストレージエンジンを指定します。

- `--get-server-public-key`

キーペアベースのパスワード交換に使用する RSA 公開キーをサーバーにリクエストします。このオプションは、`caching_sha2_password` 認証プラグインで認証されるアカウントを使用してサーバーに接続するクライアントに適用されます。このようなアカウントによる接続の場合、サーバーは要求されないがざり公開鍵をクライアントに送信しません。このオプションは、そのプラグインで認証されないアカウントでは無視されます。クライアントがセキュアな接続を使用してサーバーに接続する場合と同様に、RSA ベースのパスワード交換が不要な場合も無視されます。

`--server-public-key-path=file_name` が指定され、有効な公開キーファイルが指定されている場合は、`--get-server-public-key` よりも優先されます。

`caching_sha2_password` プラグインの詳細は、[セクション6.4.1.2「SHA-2 プラグイン認証のキャッシュ」](#)を参照してください。

- `--host=host_name, -h host_name`

指定されたホストの MySQL サーバーに接続します。

- `--iterations=N, -i N`

実行するテストの回数。

- `--login-path=name`

`.mylogin.cnf` ログインパスファイルの指定されたログインパスからオプションを読み取ります。「ログインパス」は、接続先の MySQL サーバーおよび認証に使用するアカウントを指定するオプションを含むオプショングループです。ログインパスファイルを作成または変更するには、`mysql_config_editor` ユーティリティを使用します。[セクション4.6.7「mysql_config_editor — MySQL 構成ユーティリティ」](#)を参照してください。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--no-drop`

`mysqlslap` がテスト実行中に作成するスキーマをドロップしないようにします。

- `--no-defaults`

オプションファイルを読み取りません。オプションファイルから不明のオプションを読み取ることが原因でプログラムの起動に失敗する場合、`--no-defaults` を使用して、オプションを読み取らないようにできます。

例外として、`.mylogin.cnf` ファイルは、存在する場合はすべての場合に読み取られます。これにより、`--no-defaults` が使用された場合にも、コマンド行よりも安全な方法でパスワードを指定できます。(`.mylogin.cnf` は `mysql_config_editor` ユーティリティーによって作成されます。 [セクション4.6.7「mysql_config_editor — MySQL 構成ユーティリティー」](#) を参照してください)。

このオプションおよびその他のオプションファイルオプションの詳細は、 [セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--number-char-cols=N, -x N`

`--auto-generate-sql` が指定されている場合に使用する `VARCHAR` カラムの数。

- `--number-int-cols=N, -y N`

`--auto-generate-sql` が指定されている場合に使用する `INT` カラムの数。

- `--number-of-queries=N`

各クライアントのクエリー数をおよその数に制限します。クエリーのカウントには、ステートメント区切り文字が考慮されます。たとえば、`mysqlslap` を次のように起動する場合、`;` 区切り文字が認識され、クエリー文字列の各インスタンスは 2 つのクエリーとカウントされます。その結果、(10 ではなく) 5 つの行が挿入されます。

```
shell> mysqlslap --delimiter=";" --number-of-queries=10
--query="use test;insert into t values(null)"
```

- `--only-print`

データベースには接続しません。`mysqlslap` は、実行したであろう内容を出力するだけです。

- `--password[=password], -p[password]`

サーバーへの接続に使用される MySQL アカウントのパスワード。パスワード値はオプションです。指定しない場合、`mysqlslap` によってプロンプトが表示されます。指定する場合は、`--password=` または `-p` とそれに続くパスワードの間にスペースなしが存在する必要があります。パスワードオプションを指定しない場合、デフォルトではパスワードは送信されません。

コマンド行でのパスワード指定は、セキュアでないと考えるべきです。コマンド行でパスワードを指定しないようにするには、オプションファイルを使用します。 [セクション6.1.2.1「パスワードセキュリティのためのエンドユーザーガイドライン」](#) を参照してください。

パスワードがなく、`mysqlslap` でパスワードの入力を求められないように明示的に指定するには、`--skip-password` オプションを使用します。

- `--pipe, -W`

Windows で、名前付きパイプを使用してサーバーに接続します。このオプションは、ネームパイプ接続をサポートするために `named_pipe` システム変数を有効にしてサーバーを起動した場合にのみ適用されます。また、接続を行うユーザーは、`named_pipe_full_access_group` システム変数で指定された Windows グループのメンバーである必要があります。

- `--plugin-dir=dir_name`

プラグインを検索するディレクトリ。このオプションは、`--default-auth` オプションを使用して認証プラグインを指定しても、`mysqlslap` がそれを見出さない場合に指定します。 [セクション6.2.17「プラグイン認証」](#) を参照してください。

- `--port=port_num, -P port_num`
TCP/IP 接続の場合、使用するポート番号。
- `--post-query=value`
テスト完了後に実行するステートメントを含むファイルまたは文字列。この実行は、タイミングの目的ではカウントされません。
- `--post-system=str`
テスト完了後に `system()` を使用して実行する文字列。この実行は、タイミングの目的ではカウントされません。
- `--pre-query=value`
テストの実行前に実行するステートメントを含むファイルまたは文字列。この実行は、タイミングの目的ではカウントされません。
- `--pre-system=str`
テストの実行前に `system()` を使用して実行する文字列。この実行は、タイミングの目的ではカウントされません。
- `--print-defaults`
プログラム名と、オプションファイルから受け取るすべてのオプションを出力します。
このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。
- `--protocol={TCP|SOCKET|PIPE|MEMORY}`
サーバーへの接続に使用するトランスポートプロトコル。これは、他の接続パラメータが通常、必要なプロトコル以外のプロトコルを使用する場合に便利です。許可される値の詳細は、[セクション4.2.7「接続トランスポートプロトコル」](#)を参照してください。
- `--query=value, -q value`
データ取得のため使用する `SELECT` ステートメントを含むファイルまたは文字列。
- `--server-public-key-path=file_name`
RSA キーベースのパスワード交換のためにサーバーが必要とする公開キーのクライアント側コピーを含む、PEM 形式のファイルへのパス名。このオプションは、`sha256_password` または `caching_sha2_password` 認証プラグインで認証されるクライアントに適用されます。これらのプラグインのいずれかで認証されないアカウントでは、このオプションは無視されます。クライアントがセキュアな接続を使用してサーバーに接続する場合と同様に、RSA ベースのパスワード交換を使用しない場合も無視されます。
`--server-public-key-path=file_name` が指定され、有効な公開キーファイルが指定されている場合は、`--get-server-public-key` よりも優先されます。
`sha256_password` の場合、このオプションは、MySQL が OpenSSL を使用して構築された場合にのみ適用されません。
`sha256_password` および `caching_sha2_password` プラグインの詳細は、[セクション6.4.1.3「SHA-256 プラガブル認証」](#) および [セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#)を参照してください。
- `--shared-memory-base-name=name`
Windows の場合、共有メモリを使用してローカルサーバーに接続するために使用する共有メモリ名。デフォルト値は `MYSQL` です。共有メモリ名では大文字と小文字が区別されます。
このオプションは、共有メモリ接続をサポートするために `shared_memory` システム変数を有効にしてサーバーを起動した場合にのみ適用されます。
- `--silent, -s`

サイレントモード。出力はありません。

- `--socket=path, -S path`

`localhost` への接続用に使用する、Unix ソケットファイル、または Windows では使用する名前付きパイプの名前。

Windows では、このオプションは、名前付きパイプ接続をサポートするために `named_pipe` システム変数を有効にしてサーバーを起動した場合にのみ適用されます。また、接続を行うユーザーは、`named_pipe_full_access_group` システム変数で指定された Windows グループのメンバーである必要があります。

- `--sql-mode=mode`

クライアントセッションの SQL モードを設定します。

- `--ssl*`

`--ssl` で始まるオプションは、SSL を使用してサーバーに接続するかどうかを指定し、SSL 鍵および証明書を検索する場所を指定します。 [暗号化接続のコマンドオプション](#) を参照してください。

- `--ssl-fips-mode={OFF|ON|STRICT}`

クライアント側で FIPS モードを有効にするかどうかを制御します。 `--ssl-fips-mode` オプションは、暗号化された接続の確立には使用されず、許可する暗号化操作に影響する点で、他の `--ssl-xxx` オプションとは異なります。 [セクション6.8「FIPS のサポート」](#) を参照してください。

次の `--ssl-fips-mode` 値を使用できます:

- `OFF`: FIPS モードを無効にします。
- `ON`: FIPS モードを有効にします。
- `STRICT`: 「strict」 FIPS モードを有効にします。

注記

OpenSSL FIPS オブジェクトモジュールが使用できない場合、`--ssl-fips-mode` に許可される値は `OFF` のみです。この場合、`--ssl-fips-mode` を `ON` または `STRICT` に設定すると、クライアントは起動時に警告を生成し、FIPS 以外のモードで動作します。

- `--tls-ciphersuites=ciphersuite_list`

TLSv1.3 を使用する暗号化された接続に許可される暗号スイート。値は、コロンで区切られた 1 つ以上の暗号スイート名のリストです。このオプションに指定できる暗号スイートは、MySQL のコンパイルに使用される SSL ライブラリによって異なります。詳細は、[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#) を参照してください。

このオプションは MySQL 8.0.16 で追加されました。

- `--tls-version=protocol_list`

暗号化された接続に許可される TLS プロトコル。値は、1 つまたは複数のコンマ区切りプロトコル名のリストです。このオプションに指定できるプロトコルは、MySQL のコンパイルに使用される SSL ライブラリによって異なります。詳細は、[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#) を参照してください。

- `--user=user_name, -u user_name`

サーバーへの接続に使用する MySQL アカウントのユーザー名。

- `--verbose, -v`

冗長モード。プログラムの動作についてより多くの情報を出力します。このオプションは情報量を増加させるために複数回使用できます。

- `--version, -V`

バージョン情報を表示して終了します。

- `--zstd-compression-level=level`

`zstd` 圧縮アルゴリズムを使用するサーバーへの接続に使用する圧縮レベル。許可されるレベルは 1 から 22 で、大きい値は圧縮レベルの増加を示します。デフォルトの `zstd` 圧縮レベルは 3 です。圧縮レベルの設定は、`zstd` 圧縮を使用しない接続には影響しません。

詳細は、[セクション4.2.8「接続圧縮制御」](#)を参照してください。

このオプションは MySQL 8.0.18 で追加されました。

4.6 管理およびユーティリティプログラム

このセクションでは、管理プログラムおよびその他のユーティリティー操作を実行するプログラムについて説明します。

4.6.1 `ibd2sdi` — InnoDB テーブルスペース SDI 抽出ユーティリティ

`ibd2sdi` は、InnoDB テーブルスペースファイルから [serialized dictionary information](#) (SDI) を抽出するためのユーティリティです。SDI データは、すべての永続 InnoDB テーブルスペースファイルに存在します。

`ibd2sdi` は、[file-per-table](#) テーブルスペースファイル (`*.ibd` ファイル)、[general tablespace](#) ファイル (`*.ibd` ファイル)、[system tablespace](#) ファイル (`ibdata*` ファイル) およびデータディクショナリテーブルスペース (`mysql.ibd`) で実行できます。一時テーブルスペースまたは undo テーブルスペースでの使用はサポートされていません。

`ibd2sdi` は、実行時またはサーバーのオフライン中に使用できます。SDI に関連する [DDL](#) 操作、[ROLLBACK](#) 操作および undo ログのパージ操作中に、`ibd2sdi` がテーブルスペースに格納されている SDI データの読取りに失敗する短い間隔がある場合があります。

`ibd2sdi` は、指定されたテーブルスペースから SDI のコミットされていない読取りを実行します。redo ログおよび undo ログはアクセスされません。

次のように `ibd2sdi` ユーティリティを起動します:

```
shell> ibd2sdi [options] file_name1 [file_name2 file_name3 ...]
```

`ibd2sdi` は、InnoDB システムテーブルスペースのような複数ファイルのテーブルスペースをサポートしていますが、一度に複数のテーブルスペースで実行することはできません。複数ファイルテーブルスペースの場合は、各ファイルを指定します:

```
shell> ibd2sdi ibdata1 ibdata2
```

複数ファイルテーブルスペースのファイルは、ページ番号の昇順で指定する必要があります。2 つの連続するファイルのスペース ID が同じ場合、後のファイルは前のファイルの最後のページ番号 + 1 で始まる必要があります。

`ibd2sdi` は、SDI (id、type および data フィールドを含む) を [JSON](#) 形式で出力します。

`ibd2sdi` オプション

`ibd2sdi` では、次のオプションがサポートされます:

- `--help, -h`

コマンドラインヘルプを表示します。

```
shell> ibd2sdi --help
Usage: ./ibd2sdi [-v] [-c <strict-check>] [-d <dump file name>] [-n] filename1 [filenames]
See http://dev.mysql.com/doc/refman/8.0/en/ibd2sdi.html for usage hints.
-h, --help      Display this help and exit.
-v, --version   Display version information and exit.
-#, --debug[=name] Output debug log. See
                http://dev.mysql.com/doc/refman/8.0/en/dbug-package.html
-d, --dump-file=name
```

```

Dump the tablespace SDI into the file passed by user.
Without the filename, it will default to stdout
-s, --skip-data Skip retrieving data from SDI records. Retrieve only id
and type.
-i, --id=# Retrieve the SDI record matching the id passed by user.
-t, --type=# Retrieve the SDI records matching the type passed by
user.
-c, --strict-check=name
Specify the strict checksum algorithm by the user.
Allowed values are innodb, crc32, none.
-n, --no-check Ignore the checksum verification.
-p, --pretty Pretty format the SDI output. If false, SDI would be not
human readable but it will be of less size
(Default to on; use --skip-pretty to disable.)

```

Variables (--variable-name=value)
and boolean options {FALSE|TRUE} Value (after reading options)

debug	(No default value)
dump-file	(No default value)
skip-data	FALSE
id	0
type	0
strict-check	crc32
no-check	FALSE
pretty	TRUE

- `--version, -v`

MySQL のバージョン情報を表示します。

```

shell> ibd2sdi --version
ibd2sdi Ver 8.0.3-dmr for Linux on x86_64 (Source distribution)

```

- `--debug=[debug_options], -# [debug_options]`

デバッグログを出力します。デバッグオプションについては、[セクション5.9.4「DEBUG パッケージ」](#)を参照してください。

```

shell> ibd2sdi --debug=d:t/tmp/ibd2sdi.trace

```

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `--dump-file=, -d`

シリアル化されたディクショナリ情報 (SDI) を指定されたダンプファイルにダンプします。ダンプファイルが指定されていない場合、テーブルスペース SDI は `stdout` にダンプされます。

```

shell> ibd2sdi --dump-file=file_name ../data/test/t1.ibd

```

- `--skip-data, -s`

シリアライズされたディクショナリ情報 (SDI) からの `data` フィールド値の取得をスキップし、SDI レコードの主キーである `id` および `type` フィールド値のみを取得します。

```

shell> ibd2sdi --skip-data ../data/test/t1.ibd
["ibd2sdi"
,
{
  "type": 1,
  "id": 330
},
{
  "type": 2,
  "id": 7
}
]

```

- `--id=#, -i #`

指定されたテーブルまたはテーブルスペースオブジェクト ID に一致するシリアル化ディクショナリ情報 (SDI) を取得します。オブジェクト ID はオブジェクトタイプに対して一意です。テーブルおよびテーブルスペースのオブジェクト ID は、`mysql.tables` および `mysql.tablespaces` データディクショナリテーブルの `id` カラムにもあります。データディクショナリテーブルの詳細は、[セクション14.1「データディクショナリスキーマ」](#) を参照してください。

```
shell> ibd2sdi --id=7 ../data/test/t1.ibd
["ibd2sdi"
,
{
  "type": 2,
  "id": 7,
  "object": {
    {
      "mysql_version_id": 80003,
      "dd_version": 80003,
      "sdi_version": 1,
      "dd_object_type": "Tablespace",
      "dd_object": {
        "name": "test/t1",
        "comment": "",
        "options": "",
        "se_private_data": "flags=16417;id=2;server_version=80003;space_version=1;",
        "engine": "InnoDB",
        "files": [
          {
            "ordinal_position": 1,
            "filename": "../data/test/t1.ibd",
            "se_private_data": "id=2;"
          }
        ]
      }
    }
  }
}
]
```

- `--type=#, -t #`

指定されたオブジェクト型に一致するシリアル化ディクショナリ情報 (SDI) を取得します。SDI は、テーブル (タイプ = 1) およびテーブルスペース (タイプ = 2) オブジェクトに提供されます。

```
shell> ibd2sdi --type=2 ../data/test/t1.ibd
["ibd2sdi"
,
{
  "type": 2,
  "id": 7,
  "object": {
    {
      "mysql_version_id": 80003,
      "dd_version": 80003,
      "sdi_version": 1,
      "dd_object_type": "Tablespace",
      "dd_object": {
        "name": "test/t1",
        "comment": "",
        "options": "",
        "se_private_data": "flags=16417;id=2;server_version=80003;space_version=1;",
        "engine": "InnoDB",
        "files": [
          {
            "ordinal_position": 1,
            "filename": "../data/test/t1.ibd",
            "se_private_data": "id=2;"
          }
        ]
      }
    }
  }
}
]
```

- `--strict-check, -c`

読み取られるページのチェックサムを検証するための厳密なチェックサムアルゴリズムを指定します。オプションには、`innodb`、`crc32` および `none` があります。

この例では、厳密なバージョンの `innodb` チェックサムアルゴリズムが指定されています:

```
shell> ibd2sdi --strict-check=innodb ../data/test/t1.ibd
```

この例では、厳密なバージョンの `crc32` チェックサムアルゴリズムが指定されています:

```
shell> ibd2sdi -c crc32 ../data/test/t1.ibd
```

`--strict-check` オプションを指定しない場合、非厳密な `innodb`、`crc32` および `none` チェックサムに対して検証が実行されます。

- `--no-check, -n`

読み取られるページのチェックサム検証をスキップします。

```
shell> ibd2sdi --no-check ../data/test/t1.ibd
```

- `--pretty, -p`

SDI データを JSON プリティ印刷形式で出力します。デフォルトで有効。無効にすると、SDI は人間が読めるようになりませんが、サイズは小さくなります。 `--skip-pretty` を使用して無効にします。

```
shell> ibd2sdi --skip-pretty ../data/test/t1.ibd
```

4.6.2 innochecksum — オフライン InnoDB ファイルチェックサムユーティリティー

`innochecksum` は、InnoDB ファイルのチェックサムを出力します。このツールは InnoDB テーブルスペースファイルを読み取って各ページのチェックサムを計算し、計算されたチェックサムを保存されているチェックサムと比較して不一致をレポートします。不一致はページが破損していることを示します。元は、停電後にテーブルスペースの完全性の検証を迅速化するために開発されましたが、ファイルコピーのあとにも使用できます。チェックサムの不一致により、InnoDB は意図的に実行中のサーバーを停止するため、本番中のサーバーで破損したページが発生するのを待つのではなく、このツールを使用することをお勧めします。

`innochecksum` は、サーバーがすでにオープンしているテーブルスペースファイルには使用できません。このようなファイルに関しては、`CHECK TABLE` を使用してテーブルスペース内のテーブルをチェックするとよいでしょう。サーバーがすでにオープンしているテーブルスペースで `innochecksum` を実行しようとする、`「ファイルをロックできません」` エラーが発生します。

チェックサムの不一致が見つかった場合は、バックアップからテーブルスペースをリストアするか、サーバーを起動し、`mysqldump` を使用してテーブルスペース内のテーブルのバックアップを作成します。

`innochecksum` は次のように起動します。

```
shell> innochecksum [options] file_name
```

innochecksum のオプション

`innochecksum` は次のオプションをサポートします。ページ番号を参照するオプションについては、数字はゼロベースです。

- `--help, -?`

コマンドラインヘルプを表示します。使用例:

```
shell> innochecksum --help
```

- `--info, -l`

`--help` と同義です。コマンドラインヘルプを表示します。使用例:

```
shell> innochecksum --info
```

- `--version, -V`

バージョン情報を表示します。使用例:

```
shell> innochecksum --version
```

- `--verbose, -v`

冗長モード。5 秒ごとに進行状況インジケータをログファイルに出力します。進捗インジケータを出力するには、`--log option` を使用してログファイルを指定する必要があります。`verbose` モードを有効にするには、次を実行します:

```
shell> innochecksum --verbose
```

冗長モードをオフにするには、次のコマンドを実行します:

```
shell> innochecksum --verbose=FALSE
```

`--verbose` オプションと `--log` オプションは同時に指定できます。例:

```
shell> innochecksum --verbose --log=/var/lib/mysql/test/logtest.txt
```

ログファイルで進捗インジケータ情報を検索するには、次の検索を実行します:

```
shell> cat ./logtest.txt | grep -i "okay"
```

ログファイルの進捗インジケータ情報は、次のように表示されます:

```
page 1663 okay: 2.863% done
page 8447 okay: 14.537% done
page 13695 okay: 23.568% done
page 18815 okay: 32.379% done
page 23039 okay: 39.648% done
page 28351 okay: 48.789% done
page 33023 okay: 56.828% done
page 37951 okay: 65.308% done
page 44095 okay: 75.881% done
page 49407 okay: 85.022% done
page 54463 okay: 93.722% done
...
```

- `--count, -c`

ファイル内のページ数を出力して終了します。使用例:

```
shell> innochecksum --count ../data/test/tab1.ibd
```

- `--start-page=num, -s num`

このページ番号から開始します。使用例:

```
shell> innochecksum --start-page=600 ../data/test/tab1.ibd
```

または

```
shell> innochecksum -s 600 ../data/test/tab1.ibd
```

- `--end-page=num, -e num`

このページ番号で終了します。使用例:

```
shell> innochecksum --end-page=700 ../data/test/tab1.ibd
```

または

```
shell> innochecksum -p 700 ../data/test/tab1.ibd
```

- `--page=num, -p num`

このページ番号のみをチェックします。使用例:

```
shell> innochecksum --page=701 ../data/test/tab1.ibd
```

- `--strict-check, -C`

厳密なチェックサムアルゴリズムを指定します。オプションには、`innodb`、`crc32` および `none` があります。

この例では、`innodb` チェックサムアルゴリズムが指定されています:

```
shell> innochecksum --strict-check=innodb ../data/test/tab1.ibd
```

この例では、`crc32` チェックサムアルゴリズムが指定されています:

```
shell> innochecksum -C crc32 ../data/test/tab1.ibd
```

次の条件が適用されます。

- `--strict-check` オプションを指定しない場合、`innochecksum` は `innodb`、`crc32` および `none` に対して検証を行います。
- `none` オプションを指定すると、`none` によって生成されたチェックサムのみが許可されます。
- `innodb` オプションを指定すると、`innodb` によって生成されたチェックサムのみが許可されます。
- `crc32` オプションを指定すると、`crc32` によって生成されたチェックサムのみが許可されます。
- `--no-check, -n`

チェックサムを書き換えるときにチェックサム検証を無視します。このオプションは、`innochecksum --write` オプションと一緒にのみ使用できます。`--write` オプションを指定しない場合、`innochecksum` は終了します。

この例では、`innodb` チェックサムを書き換えて、無効なチェックサムを置き換えます:

```
shell> innochecksum --no-check --write innodb ../data/test/tab1.ibd
```

- `--allow-mismatches, -a`

`innochecksum` が終了するまでに許可されるチェックサムの不一致の最大数。デフォルト設定は 0 です。 $N \geq 0$ を含む `--allow-mismatches=N` の場合、 N の不一致が許可され、`innochecksum` は $N+1$ で終了します。`--allow-mismatches` が 0 に設定されている場合、`innochecksum` は最初のチェックサムの不一致で終了します。

この例では、既存の `innodb` チェックサムが書き換えられ、`--allow-mismatches` が 1 に設定されます。

```
shell> innochecksum --allow-mismatches=1 --write innodb ../data/test/tab1.ibd
```

`--allow-mismatches` を 1 に設定すると、1000 ページのファイルで 600 ページ目と 700 ページ目に不一致がある場合、0-599 ページと 601-699 ページ目のチェックサムが更新されます。`--allow-mismatches` が 1 に設定されているため、チェックサムは最初の不一致を許容し、2 番目の不一致で終了します。ページ 600 およびページ 700-999 は変更されません。

- `--write=name, -w num`

チェックサムを書き換えます。無効なチェックサムを書き換える場合は、`--no-check` オプションを `--write` オプションとともに使用する必要があります。`--no-check` オプションは、無効なチェックサムの検証を無視するように

`innochecksum` に指示します。現在のチェックサムが有効な場合は、`--no-check` オプションを指定する必要はありません。

`--write` オプションを使用する場合は、アルゴリズムを指定する必要があります。`--write` オプションに使用可能な値は次のとおりです:

- `innodb`: InnoDB の元のアルゴリズムを使用してソフトウェアで計算されるチェックサム。
- `crc32`: `crc32` アルゴリズムを使用して計算されるチェックサムで、ハードウェア支援を使用して実行される可能性があります。
- `none`: 定数。

`--write` オプションは、ページ全体をディスクに書き換えます。新しいチェックサムが既存のチェックサムと同じである場合、I/O を最小化するために新しいチェックサムはディスクに書き込まれません。

`--write` オプションを使用すると、`innochecksum` は排他ロックを取得します。

この例では、`tab1.ibd` の `crc32` チェックサムが書き込まれます:

```
shell> innochecksum -w crc32 ../data/test/tab1.ibd
```

この例では、`crc32` チェックサムを書き換えて、無効な `crc32` チェックサムを置き換えます:

```
shell> innochecksum --no-check --write crc32 ../data/test/tab1.ibd
```

- `--page-type-summary, -S`

テーブルスペース内の各ページタイプの数を表示します。使用例:

```
shell> innochecksum --page-type-summary ../data/test/tab1.ibd
```

`--page-type-summary` の出力例:

```
File:../data/test/tab1.ibd
=====PAGE TYPE SUMMARY=====
#PAGE_COUNT PAGE_TYPE
=====
  2   Index page
  0   Undo log page
  1   Inode page
  0   Insert buffer free list page
  2   Freshly allocated page
  1   Insert buffer bitmap
  0   System page
  0   Transaction system page
  1   File Space Header
  0   Extent descriptor page
  0   BLOB page
  0   Compressed BLOB page
  0   Other type of page
=====
Additional information:
Undo page type: 0 insert, 0 update, 0 other
Undo page state: 0 active, 0 cached, 0 to_free, 0 to_purge, 0 prepared, 0 other
```

- `--page-type-dump, -D`

テーブルスペース内の各ページのページタイプ情報を `stderr` または `stdout` にダンプします。使用例:

```
shell> innochecksum --page-type-dump=tmp/a.txt ../data/test/tab1.ibd
```

- `--log, -l`

`innochecksum` ツールのログ出力。ログファイル名を指定する必要があります。ログ出力には、各テーブルスペースのチェックサム値が含まれます。圧縮されていないテーブルの場合は、LSN 値も提供されます。`--log` は、以前のリリースで使用可能だった `--debug` オプションに置き換わります。使用例:

```
shell> innochecksum --log=tmp/log.txt ../data/test/tab1.ibd
```

または

```
shell> innochecksum -l /tmp/log.txt ../data/test/tab1.ibd
```

- - オプション。

標準入力から読み取る - オプションを指定します。「標準からの読み取り」が必要なときに - オプションが欠落している場合、innochecksum は「-」オプションが省略されたことを示す innochecksum 使用状況情報を出力します。使用例:

```
shell> cat t1.ibd | innochecksum -
```

この例では、innochecksum は、元の t1.ibd ファイルを変更せずに crc32 チェックサムアルゴリズムを a.ibd に書き込みます。

```
shell> cat t1.ibd | innochecksum --write=crc32 -> a.ibd
```

複数のユーザー定義テーブルスペースファイルに対する innochecksum の実行

次の例は、複数のユーザー定義テーブルスペースファイル (.ibd ファイル) で innochecksum を実行する方法を示しています。

「test」データベース内のすべてのテーブルスペース (.ibd) ファイルに対して innochecksum を実行します:

```
shell> innochecksum ./data/test/*.ibd
```

「t」で始まるファイル名を持つすべてのテーブルスペースファイル (.ibd ファイル) に対して innochecksum を実行します:

```
shell> innochecksum ./data/test/t*.ibd
```

data ディレクトリ内のすべてのテーブルスペースファイル (.ibd ファイル) に対して innochecksum を実行します:

```
shell> innochecksum ./data/*.ibd
```

注記

cmd.exe などの Windows シェルは glob パターン展開をサポートしていないため、複数のユーザー定義テーブルスペースファイルでの innochecksum の実行は Windows オペレーティングシステムではサポートされていません。Windows システムでは、ユーザー定義のテーブルスペースファイルごとに innochecksum を個別に実行する必要があります。例:

```
cmd> innochecksum.exe t1.ibd
cmd> innochecksum.exe t2.ibd
cmd> innochecksum.exe t3.ibd
```

複数のシステムテーブルスペースファイルに対する innochecksum の実行

デフォルトでは、InnoDB システムテーブルスペースファイル (ibdata1) は 1 つのみですが、innodb_data_file_path オプションを使用してシステムテーブルスペースに複数のファイルを定義できます。次の例では、system テーブルスペースの 3 つのファイルが innodb_data_file_path オプションを使用して定義されます: ibdata1、ibdata2 および ibdata3。

```
shell> ./bin/mysqld --no-defaults --innodb-data-file-path="ibdata1:10M;ibdata2:10M;ibdata3:10M:autoextend"
```

3 つのファイル (ibdata1、ibdata2 および ibdata3) は、1 つの論理システムテーブルスペースを形成します。単一の論理システムテーブルスペースを形成する複数のファイルに対して innochecksum を実行するには、innochecksum に、標準入力からテーブルスペースファイルを読み取るための - オプションが必要です。これは、複数のファイルを連結して単一のファイルを作成することと同等です。前述の例では、次の innochecksum コマンドが使用されます:

```
shell> cat ibdata* | innochecksum -
```

「-」オプションの詳細は、innochecksum オプションの情報を参照してください。

注記

同じテーブルスペース内の複数のファイルでの `innochecksum` の実行は、Windows オペレーティングシステムではサポートされていません。これは、`cmd.exe` などの Windows シェルは `glob` パターン展開をサポートしていないためです。Windows システムでは、`innochecksum` はシステムテーブルスペースファイルごとに個別に実行する必要があります。例:

```
cmd> innochecksum.exe ibdata1
cmd> innochecksum.exe ibdata2
cmd> innochecksum.exe ibdata3
```

4.6.3 mysiam_ftdump — 全文インデックス情報の表示

`mysiam_ftdump` は MyISAM テーブル内の `FULLTEXT` インデックスに関する情報を表示します。MyISAM インデックスファイルを直接読み取るため、テーブルのあるサーバーホスト上で実行する必要があります。サーバーが稼働中の場合は、`mysiam_ftdump` を使用する前に、必ず最初に `FLUSH TABLES` ステートメントを発行してください。

`mysiam_ftdump` はインデックス全体をスキャンしてダンプします。これは特に高速ではありません。一方、単語の分布の変更頻度は高くないため、頻繁に実行する必要はありません。

`mysiam_ftdump` は次のように起動します。

```
shell> mysiam_ftdump [options] tbl_name index_num
```

`tbl_name` 引数は MyISAM テーブルの名前であるべきです。インデックスファイル (`.MYI` サフィックスの付いたファイル) を指名することでテーブルを指定することもできます。テーブルファイルがあるディレクトリで `mysiam_ftdump` を起動しない場合は、テーブルまたはインデックスのファイル名の前に、テーブルのデータベースディレクトリのパス名を指定する必要があります。インデックス番号は 0 から始まります。

例: `test` データベースに、次の定義を持つ `mytexttable` という名前のテーブルが含まれるとします。

```
CREATE TABLE mytexttable
(
  id INT NOT NULL,
  txt TEXT NOT NULL,
  PRIMARY KEY (id),
  FULLTEXT (txt)
) ENGINE=MyISAM;
```

`id` のインデックスはインデックス 0 で、`txt` の `FULLTEXT` インデックスはインデックス 1 です。作業ディレクトリが `test` データベースディレクトリの場合は、`mysiam_ftdump` を次のように起動します。

```
shell> mysiam_ftdump mytexttable 1
```

`test` データベースディレクトリへのパス名が `/usr/local/mysql/data/test` の場合、そのパス名を指定してテーブル名引数を指定することもできます。これは、データベースディレクトリ内で `mysiam_ftdump` を起動しない場合役に立ちます。

```
shell> mysiam_ftdump /usr/local/mysql/data/test/mytexttable 1
```

`mysiam_ftdump` を使用して、Unix のようなシステムで発生する頻度の順にインデックスエントリのリストを生成できます:

```
shell> mysiam_ftdump -c mytexttable 1 | sort -r
```

Windows の場合は、次を使用します:

```
shell> mysiam_ftdump -c mytexttable 1 | sort /R
```

`mysiam_ftdump` は次のオプションをサポートします。

- `--help, -h -?`
ヘルプメッセージを表示して終了します。
- `--count, -c`
1 語当たりの統計 (カウントとグローバルな重み) を計算します。

- `--dump, -d`

データオフセットや語の重みを含むインデックスをダンプします。

- `--length, -l`

長さの分布をレポートします。

- `--stats, -s`

グローバルインデックス統計をレポートします。ほかの操作が指定されていない場合、これがデフォルトの操作です。

- `--verbose, -v`

冗長モード。プログラムの動作についてより多くの情報を出力します。

4.6.4 myisamchk — MyISAM テーブルメンテナンスユーティリティ

`myisamchk` ユーティリティは、データベーステーブルに関する情報を取得したり、データベーステーブルのチェック、修復、または最適化を実行したりします。`myisamchk` は `MyISAM` テーブル (データおよびインデックスの保存のための `.MYD` および `.MYI` ファイルを持つテーブル) に機能します。

さらに、`CHECK TABLE` および `REPAIR TABLE` ステートメントを使用して、`MyISAM` テーブルをチェックして修復することもできます。 [セクション13.7.3.2「CHECK TABLE ステートメント」](#) および [セクション13.7.3.5「REPAIR TABLE ステートメント」](#) を参照してください。

`myisamchk` をパーティション化されたテーブルに対して使用することはサポートされていません。

注意

テーブルの修復操作を実行する前に、テーブルのバックアップを作成することをお勧めします。状況によっては、この操作のためにデータ損失が発生することがあります。考えられる原因としては、ファイルシステムのエラーなどがありますがこれに限りません。

`myisamchk` は次のように起動します。

```
shell> myisamchk [options] tbl_name ...
```

`options` は `myisamchk` に実行させる内容を指定します。次のセクションで、これらについて説明します。また、`myisamchk --help` を起動することでオプションのリストを取得できます。

オプションを指定しないと、`myisamchk` はデフォルトの操作としてユーザーのテーブルをチェックします。詳細な情報を取得したり、`myisamchk` に修正アクションを取らせたりするには、次の議論で説明されているようにオプションを指定してください。

`tbl_name` は、チェックまたは修復するデータベーステーブルです。データベースディレクトリ以外で `myisamchk` を起動する場合は、データベースディレクトリへのパスを指定する必要があります。これは、`myisamchk` にはデータベースディレクトリの場所がまったくわからないからです。実際には、`myisamchk` は作業対象のファイルがデータベースディレクトリにあるかどうかは考慮しません。データベーステーブルに対応するファイルをほかの場所へコピーして、そこでそれらのファイルに対してリカバリ操作を行うことができます。

必要に応じて、`myisamchk` コマンド行で複数のテーブルを指定できます。インデックスファイル (`.MYI` サフィックスの付いたファイル) を指名することでテーブルを指定することもできます。これにより、パターン `*.MYI` を使用して、ディレクトリ内のすべてのテーブルを指定することも可能になります。たとえば、データベースディレクトリ内に居る場合、次のようにそのディレクトリ内のすべての `MyISAM` テーブルをチェックできます。

```
shell> myisamchk *.MYI
```

データベースディレクトリ以外の場所からは、ディレクトリへのパスを指定することですべてのテーブルをチェックできます。

```
shell> myisamchk /path/to/database_dir/*.MYI
```

MySQL データディレクトリへのパスにワイルドカードを指定することで、すべてのデータベースのすべてのテーブルをチェックすることもできます。

```
shell> myisamchk /path/to/datadir/*/*.MYI
```

すべての MyISAM テーブルをチェックするお勧めの方法は:

```
shell> myisamchk --silent --fast /path/to/datadir/*/*.MYI
```

すべての MyISAM テーブルをチェックし、破損しているものを修復する場合は、次のコマンドを使用できます。

```
shell> myisamchk --silent --force --fast --update-state \
--key_buffer_size=64M --myisam_sort_buffer_size=64M \
--read_buffer_size=1M --write_buffer_size=1M \
/path/to/datadir/*/*.MYI
```

このコマンドは 64M バイト以上の空きがあることが前提です。myisamchk とメモリーの割り当ての詳細は、[セクション4.6.4.6「myisamchk メモリー使用量」](#)を参照してください。

myisamchk の使用に関する詳細は、[セクション7.6「MyISAM テーブルの保守とクラッシュリカバリ」](#)を参照してください。

重要

myisamchk の実行中にほかのプログラムがテーブルを使用しないことを、保証する必要があります。そのためのもっとも効果的な方法は、myisamchk の実行中に MySQL サーバーをシャットダウンするか、または myisamchk が対象とするすべてのテーブルをロックする方法です。

そうしないと、myisamchk を起動したとき、次のエラーが表示されることがあります。

```
warning: clients are using or haven't closed the table properly
```

これは、別のプログラム (mysqld サーバーなど) がテーブルを更新し、そのファイルをまだ閉じていないか、ファイルを適切に閉じずに異常終了したテーブルをチェックしようとしていることを意味します。この場合、1 つまたは複数の MyISAM テーブルが破損することがあります。

mysqld が稼働している場合、FLUSH TABLES を使用して、メモリーにバッファリングされているテーブルに加えられた変更があれば、それをフラッシュするように命令する必要があります。そのあと、myisamchk の実行中にほかのプログラムがテーブルを使用しないことを、保証する必要があります。

ただし、この問題を回避するもっとも簡単な方法は、myisamchk ではなく CHECK TABLE を使用してテーブルをチェックする方法です。[セクション13.7.3.2「CHECK TABLE ステートメント」](#)を参照してください。

myisamchk は次のオプションをサポートします。これらはコマンド行または任意のオプションファイルの [myisamchk] グループで指定できます。MySQL プログラムによって使用されるオプションファイルの詳細については、[セクション4.2.2.2「オプションファイルの使用」](#)を参照してください。

表 4.19 「myisamchk のオプション」

オプション名	説明
<code>--analyze</code>	キー値の分布を分析
<code>--backup</code>	.MYD ファイルのバックアップを file_name-time.BAK として作成
<code>--block-search</code>	指定されたオフセットのブロックが属するレコードを検索
<code>--check</code>	テーブルにエラーがないか確認
<code>--check-only-changed</code>	最後に行われた検査以降に変更されたテーブルのみをチェック
<code>--correct-checksum</code>	テーブルのチェックサム情報を修正
<code>--data-file-length</code>	データファイルの最大長 (データファイルがいっぱいになったとき再作成する場合)

オプション名	説明
<code>--debug</code>	デバッグログの書込み
<code>--decode_bits</code>	Decode_bits
<code>--defaults-extra-file</code>	通常のオプションファイルに加えて、名前付きオプションファイルを読み取ります
<code>--defaults-file</code>	指名されたオプションファイルのみを読み取る
<code>--defaults-group-suffix</code>	オプショングループのサフィクス値
<code>--description</code>	テーブルの説明情報を出力
<code>--extend-check</code>	データファイルからすべての行をリカバリする修復を試みる、非常に徹底したテーブルチェックを実行または修復を実行
<code>--fast</code>	正しく閉じられていないテーブルのみをチェック
<code>--force</code>	myisamchk がエラーをテーブル内で発見した場合、自動的に修復オペレーションを実行する。
<code>--force</code>	古い一時ファイルを上書き。 <code>-r</code> オプションまたは <code>-o</code> オプションとともに使用
<code>--ft_max_word_len</code>	FULLTEXT インデックスの単語の最大長
<code>--ft_min_word_len</code>	FULLTEXT インデックスの単語の最小長
<code>--ft_stopword_file</code>	組み込みのリストではなくこのファイルからのストップワードを使用
<code>--HELP</code>	ヘルプメッセージを表示して終了
<code>--help</code>	ヘルプメッセージを表示して終了
<code>--information</code>	チェックされたテーブルの統計を出力
<code>--key_buffer_size</code>	MyISAM テーブルのインデックスブロックに使用するバッファのサイズ
<code>--keys-used</code>	どのインデックスを更新するかを示すビット値
<code>--max-record-length</code>	myisamchk が記憶するためのメモリーを確保できない場合、指定された長さを超える行をスキップ
<code>--medium-check</code>	<code>--extend-check</code> 操作よりも速いチェックを実行
<code>--myisam_block_size</code>	MyISAM インデックスページに使用するブロックサイズ。
<code>--myisam_sort_buffer_size</code>	REPAIR 実行時のインデックスのソート、または CREATE INDEX か ALTER TABLE によるインデックスの作成の際に割り当てられるバッファ
<code>--no-defaults</code>	オプションファイルを読み取らない
<code>--parallel-recover</code>	<code>-r</code> および <code>-n</code> と同じテクニックを使用するが、異なるスレッドを使用してすべてのキーを並行して作成 (β)
<code>--print-defaults</code>	デフォルトオプションの印刷
<code>--quick</code>	データファイルを変更しないことで、修復のスピードを向上
<code>--read_buffer_size</code>	順次スキャンを実行する各スレッドは、スキャンする各テーブルについてこのサイズのバッファを割り当て
<code>--read-only</code>	テーブルを検査済みとマークしません
<code>--recover</code>	一意ではない一意なキー以外のすべてを修復できる修復を実行

オプション名	説明
<code>--safe-recover</code>	すべての行を順に読み取り、検出された行に基づいてすべてのインデックスツリーを更新する古いリカバリ方法を使用して修復
<code>--set-auto-increment</code>	新しいレコードが指定された値で開始するように AUTO_INCREMENT ナンバリングを強制
<code>--set-collation</code>	テーブルインデックスのソートに使用する照合順序を指定
<code>--silent</code>	サイレントモード
<code>--sort_buffer_size</code>	REPAIR 実行時のインデックスのソート、または CREATE INDEX か ALTER TABLE によるインデックスの作成の際に割り当てられるバッファ
<code>--sort-index</code>	インデックスツリーブロックを高いものから低いものへの順にソート
<code>--sort_key_blocks</code>	sort_key_blocks
<code>--sort-records</code>	特定のインデックスに基づいてレコードをソート
<code>--sort-recover</code>	一時ファイルのサイズが非常に大きくなっても、キーの解決にソートを使用することを myisamchk に強制
<code>--stats_method</code>	MyISAM インデックス統計コレクションコードでの NULL の取り扱い方法を指定
<code>--tmpdir</code>	一時ファイルの格納に使用されるディレクトリ
<code>--unpack</code>	myisampack でバックされたテーブルをアンバック
<code>--update-state</code>	情報を .MYI ファイルに保存し、いつテーブルがチェックされたか、およびテーブルがクラッシュしたかどうかをチェックします。
<code>--verbose</code>	冗長モード
<code>--version</code>	バージョン情報を表示して終了
<code>--write_buffer_size</code>	書き込みバッファサイズ

4.6.4.1 myisamchk の一般オプション

このセクションで紹介されているオプションは `myisamchk` によって実行されるすべてのテーブルメンテナンス操作に使用できます。このセクション以降のセクションは、テーブルのチェックまたは修復などの特定の操作のみに関するオプションを説明します。

- `--help, -?`

ヘルプメッセージを表示して終了します。オプションは操作の種類によってグループ化されています。

- `--HELP, -H`

ヘルプメッセージを表示して終了します。オプションは単独のリストに提示されます。

- `--debug=debug_options, -# debug_options`

デバッグのログを書き込みます。一般的な `debug_options` 文字列は `d:t:o,file_name` です。デフォルトは `d:t:o/tmp/myisamchk.trace` です。

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `--defaults-extra-file=file_name`

このオプションファイルは、グローバルオプションファイルのあとに読み取りますが、(UNIX では) ユーザーオプションファイルの前に読み取るようにしてください。ファイルが存在しないかアクセスできない場合、エラーが発

生じます。file_name は、フルパス名でなく相対パス名として指定された場合、現行ディレクトリを基準にして解釈されます。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--defaults-file=file_name`

指定されたオプションファイルのみ使用します。ファイルが存在しないかアクセスできない場合、エラーが発生します。file_name は、フルパス名でなく相対パス名として指定された場合、現行ディレクトリを基準にして解釈されます。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--defaults-group-suffix=str`

通常のオプショングループだけでなく、通常の名前に str のサフィクスが付いたグループも読み取ります。たとえば、myisamchk は通常 [myisamchk] グループを読み取ります。--defaults-group-suffix=_other オプションを指定した場合、myisamchk は [myisamchk_other] グループも読み取ります。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--no-defaults`

オプションファイルを読み取りません。オプションファイルから不明のオプションを読み取ることが原因でプログラムの起動に失敗する場合、--no-defaults を使用して、オプションを読み取らないようにできます。

例外として、.mylogin.cnf ファイルは、存在する場合はすべての場合に読み取られます。これにより、--no-defaults が使用された場合にも、コマンド行よりも安全な方法でパスワードを指定できます。(mylogin.cnf は mysql_config_editor ユーティリティによって作成されます。[セクション4.6.7「mysql_config_editor — MySQL 構成ユーティリティ」](#)を参照してください)。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--print-defaults`

プログラム名と、オプションファイルから受け取るすべてのオプションを出力します。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--silent, -s`

サイレントモード。エラーが発生したときのみ出力を書き出します。-s は myisamchk を非常にサイレントにするために 2 回 (-ss) 使用できます。

- `--verbose, -v`

冗長モード。プログラムの動作についてより多くの情報を出力します。これは -d と -e とともに使用できます。さらに多くの出力を得るためには、-v を複数回 (-vv, -vvv) 使用します。

- `--version, -V`

バージョン情報を表示して終了します。

- `--wait, -w`

テーブルがロックされている場合に、エラーで終了する代わりに、テーブルがロック解除されるまで待機してから続行します。外部ロックを無効にした状態で mysqld を実行している場合、テーブルをロックできるのはもう 1 つの myisamchk コマンドのみです。

--var_name=value 構文を使用すれば、次の変数も設定できます。

変数	デフォルト値
<code>decode_bits</code>	9
<code>ft_max_word_len</code>	バージョンに依存
<code>ft_min_word_len</code>	4
<code>ft_stopword_file</code>	組み込みのリスト
<code>key_buffer_size</code>	523264
<code>myisam_block_size</code>	1024
<code>myisam_sort_key_blocks</code>	16
<code>read_buffer_size</code>	262136
<code>sort_buffer_size</code>	2097144
<code>sort_key_blocks</code>	16
<code>stats_method</code>	<code>nulls_unequal</code>
<code>write_buffer_size</code>	262136

可能な `myisamchk` 変数とデフォルト値は `myisamchk --help` で確認できます。

`myisam_sort_buffer_size` は、キーをソートしてキーを修復するときに使用されます。これは、`--recover` を使用する場合の通常のケースです。`sort_buffer_size` は、`myisam_sort_buffer_size` の非推奨シノニムです。

`key_buffer_size` は `--extend-check` でテーブルをチェックするとき、またはテーブルに (通常の挿入を実行する場合のように) 1 行ずつキーを入力することでキーを修復する際に使用されます。キーバッファを通しての修復は次の場合に使用されます。

- `--safe-recover` を使用する。
- キーのソートに必要な一時ファイルが、直接キーファイルを作成する場合と比較して倍以上の大きさとなる。これは `CHAR`、`VARCHAR`、または `TEXT` カラムに大きなキー値が与えられている場合によくあります。これは、ソート操作では処理中に完全なキー値を保存する必要があるからです。一時スペースに余裕があり、ソートによって修復することを `myisamchk` に強制できる場合、`--sort-recover` オプションを使用できます。

キーバッファを使用する修復はソートを使用するよりもはるかにディスクの使用量が少ないですが、速度も落ちます。

より高速に修復を行いたい場合は、`key_buffer_size` 変数および `myisam_sort_buffer_size` 変数を、使用可能なメモリの約 25% に設定します。両方の変数が同時に使用されることはないので、両方の変数に大きい値を設定できます。

`myisam_block_size` はインデックスブロックに使用されるサイズです。

`stats_method` は、`--analyze` オプションを指定した場合に、インデックス統計の集計で `NULL` 値がどう扱われるかに影響します。`myisam_stats_method` システム変数のような働きをします。詳細は、[セクション5.1.8「サーバシステム変数」](#)と[セクション8.3.8「InnoDB および MyISAM インデックス統計コレクション」](#)に含まれる `myisam_stats_method` の説明を参照してください。

`ft_min_word_len` および `ft_max_word_len` は、MyISAM テーブルの `FULLTEXT` インデックスの語長を示します。`ft_stopword_file` はストップワードファイルを指名します。次の状況ではこれらをセットする必要があります。

`myisamchk` を使用してテーブルインデックスを変更する操作を実行する場合 (たとえば修復や分析)、特に指定しなければ、`FULLTEXT` インデックスは、最小および最大の語長とストップワードファイルにデフォルトの全文パラメータ値を使用して再構築されます。これにより、クエリーに失敗する可能性があります。

この問題は、これらのパラメータがサーバーでのみ認識されていることが原因で発生します。`MyISAM` インデックスファイルには格納されていません。サーバー内の最小および最大の語長またはストップワードファイルを変更した場合に問題を回避するには、`mysqld` で使用するのと同じ `ft_min_word_len`、`ft_max_word_len` と `ft_stopword_file` の値を `myisamchk` に指定してください。たとえば、単語の最小長を 3 に設定した場合は、次のように `myisamchk` を使用してテーブルを修復できます。

```
shell> mysamchk --recover --ft_min_word_len=3 tbl_name.MYI
```

`mysamchk` とサーバーが全文パラメータに同じ値を確実に使用するには、オプションファイルの `[mysqld]` と `[mysamchk]` セクションの両方にそれぞれを置いてください。

```
[mysqld]
ft_min_word_len=3

[mysamchk]
ft_min_word_len=3
```

`mysamchk` を使用する代わりに、`REPAIR TABLE`、`ANALYZE TABLE`、`OPTIMIZE TABLE`、または `ALTER TABLE` を使用できます。これらのステートメントは、適切に使用される全文パラメータ値が認識されているサーバーで実行されます。

4.6.4.2 mysamchk のチェックオプション

`mysamchk` はテーブルチェック操作に次のオプションをサポートします。

- `--check, -c`

テーブルにエラーがないか確認します。明示的に操作のタイプを選択しない場合、これがデフォルトの操作です。

- `--check-only-changed, -C`

最後に行われた検査以降に変更されたテーブルのみをチェックします。

- `--extend-check, -e`

テーブルチェックを非常に詳細に行います。テーブルにインデックスが多数ある場合、これには非常に時間がかかります。このオプションは極端な場合のみに使用するべきです。通常、`mysamchk` または `mysamchk --medium-check` を使用してテーブル内のエラーの有無を確認できます。

`--extend-check` を使用し、メモリー容量が十分な場合、`key_buffer_size` 値を大きくセットすれば、修復操作のスピードを上げることができます。

このオプションの説明は、テーブル修復オプションも参照してください。

出力形式の説明は、[セクション4.6.4.5「mysamchkによるテーブル情報の取得」](#)を参照してください。

- `--fast, -F`

正しく閉じられていないテーブルのみを検査します。

- `--force, -f`

`mysamchk` がテーブル内にエラーを発見した場合、自動的に修復操作を実行します。修復タイプは `--recover` または `-r` オプションで指定されるものと同じです。

- `--information, -i`

チェックされたテーブルの統計を出力します。

- `--medium-check, -m`

`--extend-check` 操作よりも高速なチェックを実行します。これはすべてのエラーの 99.99% のみを確認し、ほとんどの場合はこれで十分でしょう。

- `--read-only, -T`

テーブルを検査済みとマークしません。これは、`mysqld` が外部ロックが無効の状態稼働している場合など、ロックを使用しないほかのアプリケーションが使用しているテーブルをチェックするために `mysamchk` を使用する場合に便利です。

- `--update-state, -U`

情報を `.MYI` ファイルに保存して、テーブルがチェックされた時期およびテーブルがクラッシュしたかどうかを示します。 `--check-only-changed` オプションの利便性を最大限に引き出すためにはこれを使用すべきですが、`mysqld` サーバーがテーブルを使用し、かつ外部ロックを無効にした状態で起動している場合は、このオプションを使用しないでください。

4.6.4.3 myisamchk の修復オプション

`myisamchk` は、テーブルの修復操作 (`--recover` オプションまたは `--safe-recover` オプションなどのオプションが指定された場合に実行される操作) のために次のオプションをサポートします。

- `--backup, -B`

`.MYD` ファイルのバックアップを `file_name-time.BAK` として作成します。

- `--character-sets-dir=dir_name`

文字セットがインストールされているディレクトリ。 [セクション10.15「文字セットの構成」](#) を参照してください。

- `--correct-checksum`

テーブルのチェックサム情報を修正します。

- `--data-file-length=len, -D len`

データファイルの最大長 (データファイルが「いっぱい」になったとき再作成する場合)。

- `--extend-check, -e`

データファイルからできるかぎりの行をリカバリしようとする修復を実行します。これは通常、ガベージ行も大量に検出します。このオプションは、切羽詰まった状況でなければ使用しないでください。

このオプションの説明は、テーブルチェックオプションも参照してください。

出力形式の説明は、 [セクション4.6.4.5「myisamchk によるテーブル情報の取得」](#) を参照してください。

- `--force, -f`

中止せず、古い中間ファイル (`tbl_name.TMD` のような名前のファイル) を上書きします。

- `--keys-used=val, -k val`

`myisamchk` の場合、オプション値は更新するインデックスを示すビット値です。オプション値の各バイナリビットがテーブルインデックスに対応します。最初のインデックスはビット 0 です。オプション値が 0 の場合、すべてのインデックスの更新が無効になります。より高速な挿入を行うために使用できます。無効化されたインデックスは `myisamchk -r` を使用して再度有効化できます。

- `--no-symlinks, -l`

シンボリックリンクをたどりません。通常、`myisamchk` はシンボリックリンクが示すテーブルを修復します。MySQL 4.0 以降のバージョンでは修復操作中にシンボリックリンクを削除しないため、このオプションは 4.0 には存在しません。

- `--max-record-length=len`

`myisamchk` が指定された長さより大きい行を保存するためにメモリーを割り当てられない場合、行をスキップします。

- `--parallel-recover, -p`

`-r` および `-n` と同じテクニックを使用しますが、異なるスレッドを使用してすべてのキーを並行して作成します。これはベータ品質のコードです。自己責任でご使用ください。

- `--quick, -q`

データファイルではなくインデックスファイルのみを変更することで、より高速な修復を実現します。このオプションを2回指定することで、重複キーの場合に `myisamchk` を使用して強制的に元のデータファイルを変更させることができます。

- `--recover, -r`

一意ではない一意なキー以外のすべてを修復できる修復を実行します (これは MyISAM テーブルではきわめてまれなエラーです)。テーブルをリカバリするには、このオプションをまず試してください。 `--safe-recover` は、`myisamchk` が、`--recover` を使用してテーブルをリカバリできないとレポートする場合のみ使用するようになっています。(非常にまれではありますが、`--recover` が失敗した場合、データファイルはそのままです。)

メモリー容量に余裕がある場合、`myisam_sort_buffer_size` の値を増やすようにしてください。

- `--safe-recover, -o`

すべての行を順に読み取り、検出された行に基づいてすべてのインデックスツリーを更新する古いリカバリ方法を使用して修復します。この方法は `--recover` よりも格段に遅くなりますが、`--recover` では対応できないいくつかのまれなケースに対応できます。また、このリカバリ方法は `--recover` よりもはるかに少ないディスクスペースを使用します。通常、まず `--recover` を使用して修復し、`--recover` が失敗した場合にかぎって `--safe-recover` を使用するようになっています。

メモリー容量に余裕がある場合、`key_buffer_size` の値を増やすとよいでしょう。

- `--set-collation=name`

テーブルインデックスのソートに使用する照合順序を指定します。照合順序名の初めの部位が文字セット名を示しています。

- `--sort-recover, -n`

一時ファイルのサイズが非常に大きくなっても、キーの解決にソートを使用することを `myisamchk` に強制します。

- `--tmpdir=dir_name, -t dir_name`

一時ファイルの保存に使用するディレクトリのパス。設定しない場合、`myisamchk` は `TMPDIR` 環境変数の値を使用します。`--tmpdir` に、一時ファイルの作成にラウンドロビン方式で順に使用する、ディレクトリパスのリストを設定できます。ディレクトリ名の間のカンマ文字は、Unix ではコロン (:)、Windows ではセミコロン (;) です。

- `--unpack, -u`

`myisampack` でパックされたテーブルをアンパックします。

4.6.4.4 その他の myisamchk オプション

`myisamchk` はテーブルチェックおよび修復以外のアクションのために、次のオプションをサポートします。

- `--analyze, -a`

キー値の分布を分析します。これは、結合オプティマイザが、テーブルを結合する順番と、それが使用するインデックスをより適切に選択できるようにすることで、結合パフォーマンスを向上させます。キー分布に関する情報を取得するには、`myisamchk --description --verbose tbl_name` コマンドまたは `SHOW INDEX FROM tbl_name` ステートメントを使用します。

- `--block-search=offset, -b offset`

指定されたオフセットのブロックが属するレコードを検索します。

- `--description, -d`

テーブルの説明情報を出力します。`--verbose` オプションを一回または二回使用すると、追加情報が生成されます。セクション4.6.4.5「`myisamchk` によるテーブル情報の取得」を参照してください。

- `--set-auto-increment[=value], -A[value]`

新しい行に対する `AUTO_INCREMENT` の番号付けを、指定した値 (または、`AUTO_INCREMENT` 値がこの値と同じであるレコードが存在する場合は、それより大きい値) で開始するように強制します。 `value` が指定されていない場合は、新しいレコードの `AUTO_INCREMENT` の数字は現在テーブル内にあるもっとも大きい値 +1 で開始します。

- `--sort-index, -S`

インデックスツリーブロックを降順でソートします。これはシークを最適化し、インデックスを使用するテーブルスキャンを高速化します。

- `--sort-records=N, -R N`

特定のインデックスに基づいてレコードをソートします。これにより、データが大幅に局所に集中化されるため、このインデックスを使用する、範囲に基づいた `SELECT` または `ORDER BY` 操作が高速化する可能性があります。(テーブルのソートにはじめてこのオプションを使用する場合、かなり遅い場合があります。) テーブルのインデックス番号を決定するには、`myisamchk` が認識するのと同じ順序でテーブルのインデックスを表示する `SHOW INDEX` を使用してください。インデックスの番号は 1 から始まります。

キーがパックされていない場合 (`PACK_KEYS=0`)、長さが同じであるため、`myisamchk` がレコードをソートして移動する際、インデックスのレコードオフセットを上書きするだけです。キーがパックされている場合 (`PACK_KEYS=1`)、`myisamchk` はまずキーブロックをアンパックし、次にインデックスを再作成してキーブロックをパックする必要があります。(この場合、各インデックスのオフセットを更新するよりも、インデックスを再作成する方が早くなります。)

4.6.4.5 myisamchk によるテーブル情報の取得

`MyISAM` テーブル情報またはそれに関する統計を取得するには、次に示すコマンドを使用します。これらのコマンドの出力は、このセクションのあとの方で説明します。

- `myisamchk -d tbl_name`

`myisamchk` を「describe モード」で実行し、テーブル情報を生成します。外部ロックが無効になっている MySQL サーバーを起動した場合には、`myisamchk` は、実行中に更新があったテーブルに対してエラーをレポートすることがあります。ただし、describe モードでは `myisamchk` はテーブルを変更しないため、データが破損される危険はありません。

- `myisamchk -dv tbl_name`

`-v` を追加すると、`myisamchk` は詳細モードで実行され、テーブルについてより多くの情報を生成します。`-v` をもう一度使用するとさらに多くの情報が生成されます。

- `myisamchk -eis tbl_name`

テーブルからのもっとも重要な情報のみを表示します。この操作はテーブル全体を読み取る必要があるため、時間がかかります。

- `myisamchk -eiv tbl_name`

これは `-eis` と同様ですが、進行中の処理が表示されます。

`tbl_name` 引数は、[セクション4.6.4「myisamchk — MyISAM テーブルメンテナンスユーティリティ」](#)で説明するように、`MyISAM` テーブルの名前が、またはそのインデックスファイル名のいずれかです。複数の `tbl_name` 引数を指定できます。

`person` という名前のテーブルの構造が、次のようになっているとします。(あとに示す `myisamchk` からの出力例で、一部の値がより小さく、出力形式に適合しやすいように、`MAX_ROWS` テーブルオプションが含まれます。)

```
CREATE TABLE person
(
  id      INT NOT NULL AUTO_INCREMENT,
  last_name VARCHAR(20) NOT NULL,
  first_name VARCHAR(20) NOT NULL,
  birth   DATE,
```

```
death DATE,
PRIMARY KEY (id),
INDEX (last_name, first_name),
INDEX (birth)
) MAX_ROWS = 1000000 ENGINE=MYISAM;
```

また、テーブルのデータファイルおよびインデックスファイルのサイズは次のようになっています。

```
-rw-rw---- 1 mysql mysql 9347072 Aug 19 11:47 person.MYD
-rw-rw---- 1 mysql mysql 6066176 Aug 19 11:47 person.MYI
```

myisamchk -dvv の出力例:

```
MyISAM file:      person
Record format:    Packed
Character set:    utf8mb4_0900_ai_ci (255)
File-version:     1
Creation time:    2017-03-30 21:21:30
Status:           checked,analyzed,optimized keys,sorted index pages
Auto increment key: 1 Last value: 306688
Data records:     306688 Deleted blocks: 0
Datafile parts:   306688 Deleted data: 0
Datafile pointer (bytes): 4 Keyfile pointer (bytes): 3
Datafile length: 9347072 Keyfile length: 6066176
Max datafile length: 4294967294 Max keyfile length: 17179868159
Recordlength:    54
```

table description:

Key	Start	Len	Index	Type	Rec/key	Root	Blocksize
1	2	4	unique	long	1	1024	
2	6	80	multip.	varchar prefix	0	1024	
	87	80		varchar	0		
3	168	3	multip.	uint24 NULL	0	1024	

Field Start Length Nullpos Nullbit Type

Field	Start	Length	Nullpos	Nullbit	Type
1	1	1			
2	2	4			no zeros
3	6	81			varchar
4	87	81			varchar
5	168	3	1	1	no zeros
6	171	3	1	2	no zeros

myisamchk が生成する情報のタイプについて次に説明します。「Keyfile」とはインデックスファイルのことです。「レコード」と「行」、「フィールド」と「カラム」は、それぞれシノニムです。

テーブル情報の最初の部分には次の値が含まれます。

- **MyISAM file**

MyISAM (インデックス) ファイルの名前。

- **Record format**

テーブルの行を保存するために使用される形式。前述の例では **Fixed length** を使用しています。ほかの値には、**Compressed** および **Packed** があります。(Packed は **SHOW TABLE STATUS** レポートで **Dynamic** とレポートされるものに対応します。)

- **Character set**

テーブルのデフォルト文字セット。

- **File-version**

MyISAM 形式のバージョン。常に 1。

- **Creation time**

いつデータファイルが作成されたか。

- **Recover time**

インデックスファイル/データファイルが最後にいつ再構成されたか。

- **Status**

テーブルのステータスフラグ。可能な値は `crashed`、`open`、`changed`、`analyzed`、`optimized keys`、および `sorted index pages` です。

- **Auto increment key、Last value**

テーブルの `AUTO_INCREMENT` カラムに関連付けられたキー番号、およびこのカラムに対して直前に生成された値。そのようなカラムがない場合はこのフィールドは表示されません。

- **Data records**

テーブル内の行数。

- **Deleted blocks**

削除されたブロックで、スペースがまだ予約されているものの数。テーブルを最適化してこのスペースを最小化できます。 [セクション7.6.4「MyISAM テーブルの最適化」](#) を参照してください。

- **Datafile parts**

動的行フォーマットでは、これはデータブロックの数を示します。断片化レコードがない最適化されたテーブルでは、これは [データレコード](#) と同じです。

- **Deleted data**

未使用の削除されたデータのバイト数。テーブルを最適化してこのスペースを最小化できます。 [セクション7.6.4「MyISAM テーブルの最適化」](#) を参照してください。

- **Datafile pointer**

データファイルポインタのサイズ (バイト単位)。通常は、2、3、4、または 5 バイトです。ほとんどのテーブルは 2 バイトで対応できますが、これはまだ MySQL では制御できません。固定テーブルでは、これは行のアドレスです。動的テーブルでは、これはバイトアドレスです。

- **Keyfile pointer**

インデックスファイルポインタのサイズ (バイト単位)。通常は、1、2、または 3 バイトです。ほとんどのテーブルは 2 バイトで対応できますが、これは MySQL によって自動的に計算されます。常にブロックアドレスです。

- **Max datafile length**

テーブルデータファイルがどこまで長くなれるか (バイト単位)。

- **Max keyfile length**

テーブルインデックスファイルがどこまで長くなれるか (バイト単位)。

- **Recordlength**

各行が使用するスペース (バイト単位)。

出力の `table description` の部分にはテーブルのすべてのキーのリストが含まれます。 `myisamchk` は、それぞれのキーについて低レベル情報を表示します。

- **Key**

このキーの番号。この値は、キーの最初のカラムに関してのみ表示されます。この値が欠落している場合、行は複数カラムキーの 2 番目以降のカラムに対応します。例に示したテーブルでは、2 番目のインデックスについて 2 つの `table description` 行があります。これは、2 つの部分で構成されるマルチパートインデックスであることを示しています。

- **Start**

インデックスのこの部分が行の中のどこで始まるか。

- **Len**

インデックスのこの部分の長さ。パックされた番号では、これは常にカラム全体の長さであるはずですが、文字列では、文字列カラムのプリフィクスにインデックスを付けることができるため、インデックスされるカラム全体の長さより短い場合があります。マルチパートキーの合計の長さは、すべてのキーパートの **Len** 値の合計です。

- **Index**

キー値がインデックス内に複数回存在できるかどうか。可能な値は **unique** または **multip** (multiple) です。

- **Type**

インデックスのこの部分のデータ型。 **packed**、**stripped**、または **empty** のいずれかの値の **MyISAM** データ型です。

- **Root**

ルートインデックスブロックのアドレス。

- **Blocksize**

各インデックスブロックのサイズ。デフォルトでは 1024 ですが、この値は MySQL がソースからビルドされる場合はコンパイル時に変更できます。

- **Rec/key**

これはオプティマイザによって使用される統計値です。このインデックスで、値当たりにいくつの行があるかを示します。一意のインデックスでは値は常に 1 です。これは、テーブルのロード (または大きな変更) のあとに **myisamchk -a** で更新される場合があります。まったく更新されない場合はデフォルト値の 30 が指定されます。

出力の最後の部分は、各カラムの情報を示します。

- **Field**

カラム番号。

- **Start**

テーブルの行の中でのカラムのバイト位置。

- **Length**

カラムの長さ (バイト単位)。

- **Nullpos、Nullbit**

NULL を取るカラムでは、**MyISAM** は **NULL** 値をバイト内のフラグとして保存します。Null にできるカラムがいくつあるかによって、このために使用されるバイトが 1 またはそれ以上ある場合があります。 **Nullpos** 値および **Nullbit** 値が空でない場合は、カラムが **NULL** かどうかを示すフラグが、どのバイトおよびビットに含まれるかを示します。

NULL フラグを保存するために使用される位置とバイト数は、フィールド 1 の行に示されます。 **person** テーブルには 5 つのカラムしかないのに **Field** 行が 6 個あるのはこのためです。

- **Type**

データ型。この値は次のいずれかのディスクリプタを含むことがあります。

- **constant**

すべての行は同じ値を持っています。

- **no endspace**

エンドスペースを保存しません。

- `no endspace, not_always`

エンドスペースを保存せず、またすべての値にエンドスペース圧縮を行いません。

- `no endspace, no empty`

エンドスペースを保存しません。空の値を保存しません。

- `table-lookup`

カラムは `ENUM` に変換されました。

- `zerofill(N)`

値の中の最上位の `N` バイトは常に 0 であり、保存されません。

- `no zeros`

ゼロを保存しません。

- `always zero`

ゼロ値は 1 ビットを使用して保存されます。

- `Huff tree`

カラムに関連しているハフマンツリーの数。

- `Bits`

ハフマンツリーで使用されているビット数。

`Huff tree` フィールドおよび `Bits` フィールドは、テーブルが `myisampack` で圧縮されている場合にのみ表示されます。この情報の例は、[セクション4.6.6「myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成」](#)を参照してください。

myisamchk -eiv の出力例

```
Checking MyISAM file: person
Data records: 306688 Deleted blocks: 0
- check file-size
- check record delete-chain
No recordlinks
- check key delete-chain
block_size 1024:
- check index reference
- check data record references index: 1
Key: 1: Keyblocks used: 98% Packed: 0% Max levels: 3
- check data record references index: 2
Key: 2: Keyblocks used: 99% Packed: 97% Max levels: 3
- check data record references index: 3
Key: 3: Keyblocks used: 98% Packed: -14% Max levels: 3
Total: Keyblocks used: 98% Packed: 89%

- check records and index references
*** LOTS OF ROW NUMBERS DELETED ***

Records: 306688 M.recordlength: 25 Packed: 83%
Recordspace used: 97% Empty space: 2% Blocks/Record: 1.00
Record blocks: 306688 Delete blocks: 0
Record data: 7934464 Deleted data: 0
Lost space: 256512 Linkdata: 1156096

User time 43.08, System time 1.68
Maximum resident set size 0, Integral resident set size 0
Non-physical pagefaults 0, Physical pagefaults 0, Swaps 0
```

```
Blocks in 0 out 7, Messages in 0 out 0, Signals 0  
Voluntary context switches 0, Involuntary context switches 0  
Maximum memory usage: 1046926 bytes (1023k)
```

myisamchk -eiv の出力には、次の情報が含まれます。

- **Data records**

テーブル内の行数。

- **Deleted blocks**

削除されたブロックで、スペースがまだ予約されているものの数。テーブルを最適化してこのスペースを最小化できます。 [セクション7.6.4「MyISAM テーブルの最適化」](#)を参照してください。

- **Key**

キー番号。

- **Keyblocks used**

キーブロックの何パーセントが使用されているか。テーブルが myisamchk で再構成されたばかりの場合は、値が非常に高く (理論上の最大値に非常に近く) になります。

- **Packed**

MySQL は、共通のサフィクスを持つキー値のパックを試行します。これは、**CHAR** カラムおよび **VARCHAR** カラムのインデックスにのみ使用できます。左端に同様の部分を持つインデックス付きの長い文字列では、使用されるスペースをこれによって大幅に削減できる場合があります。前記の例では 2 番目のキーは長さが 40 バイトで、97% のスペース削減が実現されています。

- **Max levels**

このキーの B ツリーの深さ。キー値の長い大規模なテーブルでは値が大きくなります。

- **Records**

テーブル内の行数。

- **M.recordlength**

平均行長。固定長の行を持つテーブルではすべての行が同じ長さであるため、これは正確な行の長さです。

- **Packed**

MySQL は文字列の最後からスペースを削除します。 **Packed** 値は、これを行うことによって達成された削減のパーセンテージを示します。

- **Recordspace used**

データファイルの何パーセントが使用されているか。

- **Empty space**

データファイルの何パーセントが未使用か。

- **Blocks/Record**

行当たりの平均ブロック数 (すなわち、断片化された行がいくつのリンクで構成されるか)。固定形式のテーブルではこの値は常に 1.0 です。この値はできるだけ 1.0 に近くなるようにしてください。大きくなりすぎた場合は、テーブルを再構成できます。 [セクション7.6.4「MyISAM テーブルの最適化」](#)を参照してください。

- **Recordblocks**

使用されているブロック (リンク) の数。固定形式のテーブルでは、これは行数と同じです。

- [Deleteblocks](#)

削除されたブロック (リンク) の数。

- [Recorddata](#)

データファイル内の使用されているバイト数。

- [Deleted data](#)

データファイル内の削除された (未使用の) バイト数。

- [Lost space](#)

行がより短い長さに更新されると、一部のスペースが失われます。これは、そのような損失の合計 (バイト単位) です。

- [Linkdata](#)

動的テーブル形式が使用される場合、行の断片はポインタ (それぞれ 4 から 7 バイト) でリンクされます。 [Linkdata](#) は、このようなポインタすべてが使用しているストレージ量の合計です。

4.6.4.6 myisamchk メモリー使用量

[myisamchk](#) を実行する際、メモリー割り当ては重要です。 [myisamchk](#) はメモリー関係の変数の設定を超えてメモリーを使用することはありません。 [myisamchk](#) を非常に大きなテーブルで使用する場合、まずどのくらいのメモリーを使用するか決定する必要があります。デフォルトでは、修復に 3M バイト程度しか使用しないように設定されています。より大きな値を使用することで、 [myisamchk](#) の動作速度を上げることができます。たとえば、512M バイトを超える RAM が使用可能な場合は、(ほかに指定するオプションに加えて) 次のようなオプションを使用できます。

```
shell> myisamchk --myisam_sort_buffer_size=256M \
--key_buffer_size=512M \
--read_buffer_size=64M \
--write_buffer_size=64M ...
```

おそらくほとんどの場合には `--myisam_sort_buffer_size=16M` を使用すれば十分です。

[myisamchk](#) は `TMPDIR` 内の一時ファイルを使用することに注意してください。 `TMPDIR` がメモリーファイルシステムを指している場合、メモリー不足エラーが容易におきる可能性があります。これが発生した場合は、 `--tmpdir=dir_name` オプションを指定して [myisamchk](#) を実行し、より多くの領域を持つファイルシステム上にあるディレクトリを指定します。

修復操作を実行する場合、 [myisamchk](#) はディスクスペースも大量に必要とします。

- データファイルのサイズの 2 倍 (元のファイルとコピー)。 `--quick` で修復を行う場合、スペースは必要ありません。この場合、再作成されるのはインデックスファイルのみです。コピーはオリジナルと同じディレクトリ内に作成されるため、このスペースはオリジナルのデータファイルと同じファイルシステム上で使用可能でなければなりません。
- 古いインデックスファイルを置換する新しいもの用のスペース。古いインデックスファイルは修復操作の最初に切り捨てられるため、通常このスペースは無視します。このスペースは、オリジナルのデータファイルと同じファイルシステム上で使用可能でなければなりません。
- `--recover` または `--sort-recover` を使用する場合 (ただし `--safe-recover` を使用する場合を除く)、ソートのためのスペースがディスク上に必要です。この領域は、 (`TMPDIR` または `--tmpdir=dir_name` で指定された) 一時ディレクトリに割り当てられます。次の式は必要なスペースの量を求めます。

```
(largest_key + row_pointer_length) * number_of_rows * 2
```

キー長および `row_pointer_length` は [myisamchk -dv tbl_name](#) で確認できます ([セクション4.6.4.5 「myisamchk によるテーブル情報の取得」](#)を参照してください)。 `row_pointer_length` 値および `number_of_rows` 値は、テーブル情報内の `Datafile pointer` 値および `Data records` 値です。 `largest_key` 値を判断するには、テーブル情報内の `Key` 行を確認します。 `Len` カラムは各キー部分のバイト数を示します。マルチカラムインデックスでは、キーサイズはすべてのキー部分の `Len` 値の合計です。

修復中にディスクスペースの問題がある場合は、`--recover`の代わりに `--safe-recover` を使用してみてください。

4.6.5 myisamlog — MyISAM ログファイルの内容の表示

`myisamlog` は MyISAM ログファイルの内容を処理します。このようなファイルを作成するには、サーバーを `--log-isam=log_file` オプションで起動します。

`myisamlog` は次のように起動します。

```
shell> myisamlog [options] [file_name [tbl_name] ...]
```

デフォルトの操作は更新 (`-u`) です。リカバリが実行された場合 (`-r`)、すべての書き込み、および更新や削除が実行され、エラーは数えられるだけです。 `log_file` 引数が指定されない場合は、デフォルトのログファイル名は `myisam.log` です。コマンド行でテーブルが指名された場合は、そのテーブルのみが更新されます。

`myisamlog` は次のオプションをサポートします。

- `-?, -l`
ヘルプメッセージを表示して終了します。
- `-c N`
`N` 個のコマンドのみ実行します。
- `-f N`
開かれているファイルの最大数を指定します。
- `-F filepath/`
末尾にスラッシュを付けてファイルパスを指定します。
- `-i`
終了する前に追加情報を表示します。
- `-o offset`
開始オフセットを指定します。
- `-p N`
パスから `N` 個のコンポーネントを取り除きます。
- `-r`
リカバリ操作を実行します。
- `-R record_pos_file record_pos`
レコード位置ファイルとレコード位置を指定します。
- `-u`
更新操作を実行します。
- `-v`
冗長モード。プログラムの動作についてより多くの情報を出力します。このオプションを複数回指定して、さらに多くの出力を生成できます。
- `-w write_file`
書き込みファイルを指定します。

- -V

バージョン情報を表示します。

4.6.6 myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成

`myisampack` ユーティリティーは MyISAM テーブルを圧縮します。`myisampack` は、テーブルの各カラムを独立して圧縮することによって機能します。通常、`myisampack` はデータファイルを 40% から 70% バックします。

テーブルがあとで使用される場合、カラムの解凍に必要な情報をサーバーがメモリー内に読み取ります。これにより、個々の行をアクセスする際のパフォーマンスが大幅に向上します。これは、圧縮解除しなければならないのは 1 つの行のみであるためです。

MySQL は、圧縮されたテーブルでメモリーのマッピングを行う場合に、可能であれば `mmap()` を使用します。`mmap()` が機能しない場合、MySQL は普通のファイル読み取り/書き込み操作に戻ります。

次の点に注意してください。

- `mysqld` サーバーが外部ロックが無効化された状態で起動された場合、バック処理の最中にサーバーによってテーブルが更新される可能性がある場合は、`myisampack` の起動は推奨されません。サーバーが停止している状態でテーブルを圧縮するのもっとも安全です。
- テーブルは、バック後に読み取り専用になります。通常これは意図されたものです (CD 内のバックされたテーブルにアクセスする場合など)。
- `myisampack` はパーティション化されたテーブルをサポートしません。

`myisampack` は次のように起動します。

```
shell> myisampack [options] file_name ...
```

各ファイル名引数はインデックス (.MYI) ファイルの名前にしてください。データベースディレクトリ内にいない場合、ファイルへのパスを指定するようにしてください。MYI 拡張子は省略可能です。

`myisampack` でテーブルを圧縮した後、`myisamchk -rq` を使用してインデックスを再構築します。セクション 4.6.4 「`myisamchk` — MyISAM テーブルメンテナンスユーティリティー」。

`myisampack` は次のオプションをサポートします。また、オプションファイルを読み取り、セクション 4.2.2.3 「オプションファイルの処理に影響するコマンド行オプション」に説明されている、それらを処理するためのオプションもサポートします。

- `--help, -?`

ヘルプメッセージを表示して終了します。

- `--backup, -b`

`tbl_name.OLD` という名前を使用して、各テーブルのデータファイルのバックアップを作成します。

- `--character-sets-dir=dir_name`

文字セットがインストールされているディレクトリ。セクション 10.15 「文字セットの構成」を参照してください。

- `--debug=[debug_options], -# [debug_options]`

デバッグのログを書き込みます。一般的な `debug_options` 文字列は `d:t:o,file_name` です。デフォルトは `d:t:o` です。

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合のみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `--force, -f`

パックされたテーブルが元のテーブルより大きくなる場合や、以前に `myisampack` を呼び出した際の間中ファイルが存在する場合でも、パックされたテーブルを生成します (`myisampack` はテーブルの圧縮中に、`tbl_name.TMD` という名前の中間ファイルをデータベースディレクトリに作成します。`myisampack` を強制終了した場合、`.TMD` ファイルは削除されないことがあります。) 通常、`myisampack` は `tbl_name.TMD` が存在することを検出すると、エラーで終了します。 `--force` を使用すると、`myisampack` は必ずテーブルをパックします。

- `--join=big_tbl_name, -j big_tbl_name`

コマンド行で指名されたすべてのテーブルを、単一のパックされたテーブル `big_tbl_name` に結合します。結合されるすべてのテーブルは、必ずまったく同一の構造でなければなりません (同一カラム名、型、同一インデックスなど)。

結合操作の前に、`big_tbl_name` が存在していなければなりません。コマンド行で指名され `big_tbl_name` にマージされるすべてのソーステーブルが存在していなければなりません。ソースのテーブルは結合のために読み取られますが、変更はされません。

- `--silent, -s`

サイレントモード。エラーが発生したときのみ出力を書き出します。

- `--test, -t`

実際にテーブルをパックせず、パックのテストのみを実行します。

- `--tmpdir=dir_name, -T dir_name`

指名されたディレクトリを、`myisampack` が一時ファイルを作成する場所として使用します。

- `--verbose, -v`

冗長モード。パック操作の進行状況とその結果に関する情報を書き込みます。

- `--version, -V`

バージョン情報を表示して終了します。

- `--wait, -w`

テーブルが使用中の場合、待機してから再度試みます。`mysqld` サーバーが外部ロックが無効化された状態で起動された場合、パック処理の最中にサーバーによってテーブルが更新される可能性がある場合は、`myisampack` の起動は推奨されません。

次のコマンドシーケンスは典型的なテーブル圧縮セッションを表しています。

```
shell> ls -l station.*
-rw-rw-r-- 1 jones  my   994128 Apr 17 19:00 station.MYD
-rw-rw-r-- 1 jones  my   53248  Apr 17 19:00 station.MYI

shell> myisamchk -dvv station

MyISAM file:  station
Isam-version: 2
Creation time: 1996-03-13 10:08:58
Recover time: 1997-02-02 3:06:43
Data records: 1192 Deleted blocks: 0
Datafile parts: 1192 Deleted data: 0
Datafile pointer (bytes): 2 Keyfile pointer (bytes): 2
Max datafile length: 54657023 Max keyfile length: 33554431
Recordlength: 834
Record format: Fixed length

table description:
Key Start Len Index Type Root Blocksize Rec/key
1 2 4 unique unsigned long 1024 1024 1
2 32 30 multip. text 10240 1024 1
```

```
Field Start Length Type
1 1 1
2 2 4
3 6 4
4 10 1
5 11 20
6 31 1
7 32 30
8 62 35
9 97 35
10 132 35
11 167 4
12 171 16
13 187 35
14 222 4
15 226 16
16 242 20
17 262 20
18 282 20
19 302 30
20 332 4
21 336 4
22 340 1
23 341 8
24 349 8
25 357 8
26 365 2
27 367 2
28 369 4
29 373 4
30 377 1
31 378 2
32 380 8
33 388 4
34 392 4
35 396 4
36 400 4
37 404 1
38 405 4
39 409 4
40 413 4
41 417 4
42 421 4
43 425 4
44 429 20
45 449 30
46 479 1
47 480 1
48 481 79
49 560 79
50 639 79
51 718 79
52 797 8
53 805 1
54 806 1
55 807 20
56 827 4
57 831 4
```

```
shell> myisampack station.MYI
Compressing station.MYI: (1192 records)
- Calculating statistics

normal: 20 empty-space: 16 empty-zero: 12 empty-fill: 11
pre-space: 0 end-space: 12 table-lookups: 5 zero: 7
Original trees: 57 After join: 17
- Compressing file
87.14%
Remember to run myisamchk -rq on compressed tables

shell> myisamchk -rq station
- check record delete-chain
- recovering (with sort) MyISAM-table 'station'
Data records: 1192
- Fixing index 1
```

```

- Fixing index 2

shell> mysqladmin -uroot flush-tables

shell> ls -l station.*
-rw-rw-r-- 1 jones  my   127874 Apr 17 19:00 station.MYD
-rw-rw-r-- 1 jones  my   55296 Apr 17 19:04 station.MYI

shell> myisamchk -dvv station

MyISAM file:  station
Isam-version:  2
Creation time: 1996-03-13 10:08:58
Recover time:  1997-04-17 19:04:26
Data records:   1192 Deleted blocks:    0
Datafile parts: 1192 Deleted data:    0
Datafile pointer (bytes): 3 Keyfile pointer (bytes): 1
Max datafile length: 16777215 Max keyfile length: 131071
Recordlength:   834
Record format: Compressed

table description:
Key Start Len Index  Type          Root Blocksize Rec/key
1  2  4  unique unsigned long  10240  1024    1
2 32 30  multip. text          54272  1024    1

Field Start Length Type          Huff tree Bits
1  1  1  constant          1  0
2  2  4  zerofill(1)       2  9
3  6  4  no zeros, zerofill(1) 2  9
4 10  1                3  9
5 11 20  table-lookup      4  0
6 31  1                3  9
7 32 30  no endspace, not_always 5  9
8 62 35  no endspace, not_always, no empty 6  9
9 97 35  no empty          7  9
10 132 35 no endspace, not_always, no empty 6  9
11 167 4  zerofill(1)       2  9
12 171 16  no endspace, not_always, no empty 5  9
13 187 35  no endspace, not_always, no empty 6  9
14 222 4  zerofill(1)       2  9
15 226 16  no endspace, not_always, no empty 5  9
16 242 20  no endspace, not_always 8  9
17 262 20  no endspace, no empty 8  9
18 282 20  no endspace, no empty 5  9
19 302 30  no endspace, no empty 6  9
20 332 4  always zero       2  9
21 336 4  always zero       2  9
22 340 1                3  9
23 341 8  table-lookup      9  0
24 349 8  table-lookup     10  0
25 357 8  always zero       2  9
26 365 2                2  9
27 367 2  no zeros, zerofill(1) 2  9
28 369 4  no zeros, zerofill(1) 2  9
29 373 4  table-lookup     11  0
30 377 1                3  9
31 378 2  no zeros, zerofill(1) 2  9
32 380 8  no zeros          2  9
33 388 4  always zero       2  9
34 392 4  table-lookup     12  0
35 396 4  no zeros, zerofill(1) 13  9
36 400 4  no zeros, zerofill(1) 2  9
37 404 1                2  9
38 405 4  no zeros          2  9
39 409 4  always zero       2  9
40 413 4  no zeros          2  9
41 417 4  always zero       2  9
42 421 4  no zeros          2  9
43 425 4  always zero       2  9
44 429 20  no empty          3  9
45 449 30  no empty          3  9
46 479 1                14  4
47 480 1                14  4
48 481 79  no endspace, no empty 15  9

```

49	560	79	no empty	2	9
50	639	79	no empty	2	9
51	718	79	no endspace	16	9
52	797	8	no empty	2	9
53	805	1		17	1
54	806	1		3	9
55	807	20	no empty	3	9
56	827	4	no zeros, zerofill(2)	2	9
57	831	4	no zeros, zerofill(1)	2	9

myisampack は次の種類の情報を表示します。

- **normal**

追加のパックが使用されていないカラムの数。

- **empty-space**

スペースのみの値を含むカラムの数。これらは 1 ビットを占めます。

- **empty-zero**

バイナリのゼロのみの値を含むカラムの数。これらは 1 ビットを占めます。

- **empty-fill**

それぞれの型のバイト範囲をすべて占領しない整数カラムの数。これらはより小さい型に変更されます。たとえば、BIGINT カラム (8 バイト) のすべての値が -128 から 127 の範囲内にある場合は、このカラムを TINYINT カラム (1 バイト) として格納できます。

- **pre-space**

先頭にスペースが付いて保存されている 10 進数カラムの数。この場合、各値は先頭のスペースの数のカウントを含んでいます。

- **end-space**

後続のスペースを多く含むカラムの数。この場合、各値は後続のスペースの数のカウントを含んでいます。

- **table-lookup**

カラムには少数の異なる値のみがあり、ハフマン圧縮の前に ENUM に変換されました。

- **zero**

すべての値がゼロのカラムの数。

- **Original trees**

もともとのハフマンツリーの数です。

- **After join**

ヘッダースペースを節約するため、ツリーの結合のあとに残った異なるハフマンツリーの数。

テーブルの圧縮後、myisamchk -dvv が表示する Field 行には、各カラムに関する追加情報が含まれます。

- **Type**

データ型。この値は次のいずれかのディスクリプタを含むことがあります。

- **constant**

すべての行は同じ値を持っています。

- **no endspace**

エンドスペースを保存しません。

- `no endspace, not_always`

エンドスペースを保存せず、またすべての値にエンドスペース圧縮を行いません。

- `no endspace, no empty`

エンドスペースを保存しません。空の値を保存しません。

- `table-lookup`

カラムは `ENUM` に変換されました。

- `zerofill(N)`

値の中の最上位の `N` バイトは常に 0 であり、保存されません。

- `no zeros`

ゼロを保存しません。

- `always zero`

ゼロ値は 1 ビットを使用して保存されます。

- `Huff tree`

カラムに関連しているハフマンツリーの数。

- `Bits`

ハフマンツリーで使用されているビット数。

`mysampack` を実行した後、`mysamchk` を使用してインデックスを再作成します。このとき、MySQL オプティマイザの動作効率化を図るために、インデックスブロックのソートと統計の作成を行うことができます。

```
shell> mysamchk -rq --sort-index --analyze tbl_name.MYI
```

バックされたテーブルを MySQL データベースディレクトリにインストールしたあと、`mysqld` が新しいテーブルを使用することを強制するため、`mysqladmin flush-tables` を実行するようにしてください。

バックされたテーブルをアンパックするには、`mysamchk` に対して `--unpack` オプションを使用してください。

4.6.7 mysql_config_editor — MySQL 構成ユーティリティ

`mysql_config_editor` ユーティリティでは、`.mylogin.cnf` という名前の不明瞭化されたログインパスファイルに認証資格証明を格納できます。ファイルの場所は、Windows では `%APPDATA%\MySQL` ディレクトリ、非 Windows システムでは現在のユーザーのホームディレクトリです。このファイルは、MySQL Server に接続するための認証情報を取得するために、MySQL クライアントプログラムによってあとで読み取ることができます。

`.mylogin.cnf` ログインパスファイルの不明瞭化された形式は、他のオプションファイルと同様に、オプショングループで構成されます。`.mylogin.cnf` の各オプショングループは、特定のオプションのみを許可するグループである「ログインパス、」と呼ばれます: `host`, `user`, `password`, `port` および `socket`。ログインパスオプショングループは、接続先の MySQL サーバーおよび認証に使用するアカウントを指定する一連のオプションと考えてください。不明瞭化された例を次に示します:

```
[client]
user = mydefaultname
password = mydefaultpass
host = 127.0.0.1
[mypath]
user = myothername
```



```
password = myotherpass
host = localhost
```

クライアントプログラムを起動してサーバーに接続すると、クライアントは `.mylogin.cnf` を他のオプションファイルとともに使用します。その優先順位はほかのオプションファイルより高くなりますが、クライアントのコマンド行で明示的に指定されたオプションよりは低くなります。オプションファイルが使用される順序の詳細は、[セクション 4.2.2.2「オプションファイルの使用」](#)を参照してください。

代替ログインパスファイル名を指定するには、`MYSQL_TEST_LOGIN_FILE` 環境変数を設定します。この変数は、`mysql_config_editor`、標準の MySQL クライアント (`mysql`、`mysqladmin` など) および `mysql-test-run.pl` テストユーティリティによって認識されます。

プログラムでは、次のようにログインパスファイルのグループが使用されます:

- 使用するログインパスを明示的に示す `--login-path=name` オプションを指定しない場合、`mysql_config_editor` はデフォルトで `client` ログインパスを操作します。
- `--login-path` オプションを指定しない場合、クライアントプログラムは、ほかのオプションファイルから読み取られたものと同じオプショングループをログインパスファイルから読み取ります。次のコマンドについて考えてみます:

```
shell> mysql
```

デフォルトでは、`mysql` クライアントは他のオプションファイルから `[client]` および `[mysql]` グループを読み取るため、ログインパスファイルからも読み取ります。

- `--login-path` オプションを使用すると、クライアントプログラムはさらにログインパスファイルから名前付きログインパスを読み取ります。ほかのオプションファイルから読み取られたオプショングループは同じままです。次のコマンドについて考えてみます:

```
shell> mysql --login-path=myspath
```

`mysql` クライアントは、他のオプションファイルから `[client]` および `[mysql]` を読み取り、ログインパスファイルから `[client]`、`[mysql]` および `[myspath]` を読み取ります。

- クライアントプログラムは、`--no-defaults` オプションが使用されている場合でもログインパスファイルを読み取ります。これにより、`--no-defaults` が存在する場合でも、コマンド行よりも安全な方法でパスワードを指定できます。

`mysql_config_editor` は、`.mylogin.cnf` ファイルを不明瞭化してクリアテキストとして読み取れないようにし、クライアントプログラムによって不明瞭化されていない場合はその内容がメモリー内でのみ使用されるようにします。このようにして、パスワードをクリアテキスト以外の形式でファイルに格納し、後でコマンドラインまたは環境変数で公開しなくても使用できます。`mysql_config_editor` には、ログインパスファイルの内容を表示するための `print` コマンドが用意されていますが、この場合でも、他のユーザーが表示できるようにパスワード値がマスクされます。

`mysql_config_editor` で使用される不明瞭化により、`.mylogin.cnf` でパスワードがクリアテキストとして表示されなくなり、誤ったパスワードの公開を防止することでセキュリティ対策が提供されます。たとえば、通常の不明瞭化されていない `my.cnf` オプションファイルを画面に表示すると、そこに含まれるパスワードはすべてのユーザーに表示されません。`.mylogin.cnf` では、これは `true` ではありませんが、使用される不明瞭化によって決定された攻撃者が検出されない可能性があるため、再利用不可能とみなすべきではありません。マシンでファイルにアクセスするためのシステム管理権限を取得できるユーザーは、なんらかの労力で `.mylogin.cnf` ファイルを不明瞭化解除できます。

ログインパスファイルは、現在のユーザーが読取りおよび書き込み可能であり、他のユーザーはアクセスできない必要があります。それ以外の場合、`mysql_config_editor` では無視され、クライアントプログラムでも使用されません。

`mysql_config_editor` は次のように起動します。

```
shell> mysql_config_editor [program_options] command [command_options]
```

ログインパスファイルが存在しない場合は、`mysql_config_editor` によって作成されます。

コマンド引数は次のとおりです:

- `program_options` は、一般的な `mysql_config_editor` オプションで構成されています。

- `command` は、`.mylogin.cnf` ログインパスファイルに対して実行するアクションを示します。たとえば、`set` はログインパスをファイルに書き出し、`remove` はログインパスを削除し、`print` はログインパスの内容を表示します。
- `command_options` は、ログインパス名やログインパスで使用する値など、コマンドに固有の追加オプションを示します。

プログラム引数のセット内でのコマンド名の位置は重要です。たとえば、次のコマンド行は同じ引数を持ちますが、結果は異なります。

```
shell> mysql_config_editor --help set
shell> mysql_config_editor set --help
```

最初のコマンドラインでは、`mysql_config_editor` の一般的なヘルプメッセージが表示され、`set` コマンドは無視されます。2 目目のコマンドラインには、`set` コマンドに固有のヘルプメッセージが表示されます。

デフォルトの接続パラメータを定義する `client` ログインパスと、ホスト `remote.example.com` の MySQL サーバーに接続するための `remote` という追加のログインパスを確立するとします。次のようにログインします:

- デフォルトでは、`localuser` および `localpass` のユーザー名とパスワードを持つローカルサーバー
- ユーザー名とパスワードが `remoteuser` および `remotepass` のリモートサーバーへ

`.mylogin.cnf` ファイルにログインパスを設定するには、次の `set` コマンドを使用します。各コマンドを単一行に入力し、プロンプトが表示されたら適切なパスワードを入力します:

```
shell> mysql_config_editor set --login-path=client
--host=localhost --user=localuser --password
Enter password: enter password "localpass" here
shell> mysql_config_editor set --login-path=remote
--host=remote.example.com --user=remoteuser --password
Enter password: enter password "remotepass" here
```

`mysql_config_editor` ではデフォルトで `client` ログインパスが使用されるため、最初のコマンドから `--login-path=client` オプションを省略しても効果を変更できません。

`mysql_config_editor` が `.mylogin.cnf` ファイルに書き込む内容を確認するには、`print` コマンドを使用します:

```
shell> mysql_config_editor print --all
[client]
user = localuser
password = *****
host = localhost
[remote]
user = remoteuser
password = *****
host = remote.example.com
```

`print` コマンドは、各ログインパスを行のセットとして表示します。各セットには最初にグループヘッダー (角かっこ内にログインパス名を示します) があり、ログインパスのオプション値がそれに続きます。パスワード値はマスクされ、クリアテキストとして表示されません。

すべてのログインパスを表示するために `--all` を指定しない場合、または名前付きログインパスを表示するために `--login-path=name` を指定した場合、`print` コマンドによって `client` ログインパスがデフォルトで表示されます (存在する場合)。

前述の例に示すように、ログインパスファイルには複数のログインパスを含めることができます。このようにして、`mysql_config_editor` では、異なる MySQL サーバーに接続するため、または異なるアカウントを使用して特定のサーバーに接続するために、複数の「パーソナリティ」を簡単に設定できます。これらはすべて、あとでクライアントプログラムを起動するときに `--login-path` オプションを使用して名前を選択できます。たとえば、リモートサーバーに接続するには、次のコマンドを使用します:

```
shell> mysql --login-path=remote
```

ここで、`mysql` は、他のオプションファイルから `[client]` および `[mysql]` オプショングループを読み取り、ログインパスファイルから `[client]`、`[mysql]` および `[remote]` グループを読み取ります。

ローカルサーバーに接続するには、次のコマンドを使用します:

```
shell> mysql --login-path=client
```

`mysql` はデフォルトで `client` および `mysql` のログインパスを読み取るため、`--login-path` オプションでは何も追加されません。このコマンドは次のコマンドと同等です:

```
shell> mysql
```

ログインパスファイルから読み取られたオプションは、ほかのオプションファイルから読み取られたオプションよりも優先されます。後でログインパスファイルに表示されるログインパスグループから読み取られるオプションは、前にファイルに表示されたグループから読み取られるオプションよりも優先されます。

`mysql_config_editor` では、ログインパスが作成順にログインパスファイルに追加されるため、最初により一般的なログインパスを作成し、後でより具体的なパスを作成する必要があります。ファイル内でログインパスを移動する必要がある場合は、ログインパスを削除してから再作成し、最後に追加できます。たとえば、`client` ログインパスはすべてのクライアントプログラムによって読み取られるのに対し、`mysqldump` ログインパスは `mysqldump` によって読み取り専用であるため、より一般的です。後で指定したオプションは、前に指定したオプションをオーバーライドするため、`mysqldump` の順序でログインパスを配置すると、`mysqldump` 固有のオプションで `client` オプションをオーバーライドできます。

`mysql_config_editor` で `set` コマンドを使用してログインパスを作成する場合、使用可能なすべてのオプション値 (ホスト名、ユーザー名、パスワード、ポート、ソケット) を指定する必要はありません。指定した値のみがパスに書き出されます。欠落している値があとで必要になった場合は、クライアントパスを起動して MySQL サーバーに接続するときに、オプションファイルまたはコマンド行で指定できます。コマンド行で指定されたオプションは、ログインパスファイルまたはその他のオプションファイルで指定されたオプションより優先されます。たとえば、`remote` ログインパスの資格証明がホスト `remote2.example.com` にも適用される場合は、そのホスト上のサーバーに次のように接続します:

```
shell> mysql --login-path=remote --host=remote2.example.com
```

mysql_config_editor の一般オプション

`mysql_config_editor` では、次の一般的なオプションがサポートされています。これらは、コマンドラインで指定された任意のコマンドの前に使用できます。コマンド固有のオプションの詳細は、[mysql_config_editor のコマンドおよびコマンド固有のオプション](#) を参照してください。

表 4.20 「mysql_config_editor の一般オプション」

オプション名	説明
<code>--debug</code>	デバッグログの書込み
<code>--help</code>	ヘルプメッセージを表示して終了
<code>--verbose</code>	冗長モード
<code>--version</code>	バージョン情報を表示して終了

- `--help, -?`

一般的なヘルプメッセージを表示して終了します。

コマンド固有のヘルプメッセージを表示するには、次のように `mysql_config_editor` を起動します (`command` は `help` 以外のコマンドです):

```
shell> mysql_config_editor command --help
```

- `--debug[=debug_options], -# debug_options`

デバッグのログを書き込みます。一般的な `debug_options` 文字列は `d:t:o,file_name` です。デフォルトは `d:t:o,/tmp/mysql_config_editor.trace` です。

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合のみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `--verbose, -v`

冗長モード。プログラムの動作についてより多くの情報を出力します。このオプションは、操作が期待する効果を生じない場合に、問題を診断するのに役立ちます。

- `--version, -V`

バージョン情報を表示して終了します。

mysql_config_editor のコマンドおよびコマンド固有のオプション

このセクションでは、許可される `mysql_config_editor` コマンドについて説明し、各コマンドについて、コマンドラインのコマンド名の後に許可されるコマンド固有のオプションについて説明します。

また、`mysql_config_editor` は、任意のコマンドの前に使用できる一般的なオプションをサポートしています。これらのオプションの詳細は、[mysql_config_editor の一般オプション](#) を参照してください。

`mysql_config_editor` は次のコマンドをサポートします。

- `help`

一般的なヘルプメッセージを表示して終了します。このコマンドには、次のオプションはありません。

コマンド固有のヘルプメッセージを表示するには、次のように `mysql_config_editor` を起動します (`command` は `help` 以外のコマンドです):

```
shell> mysql_config_editor command --help
```

- `print [options]`

パスワードが `*****` として表示されることを除き、ログインパスワードの内容を不明瞭化された形式で出力します。

ログインパスが指定されていない場合、デフォルトのログインパス名は `client` です。 `--all` および `--login-path` の両方が指定されている場合、`--all` が優先されます。

`print` コマンドでは、コマンド名の後に次のオプションを指定できます:

- `--help, -?`

`print` コマンドのヘルプメッセージを表示して終了します。

一般的なヘルプメッセージを表示するには、`mysql_config_editor --help` を使用します。

- `--all`

ログインパスワード内のすべてのログインパスの内容を出力します。

- `--login-path=name, -G name`

指定されたログインパスの内容を出力します。

- `remove [options]`

ログインパスワードからログインパスを削除するか、ログインパスからオプションを削除してログインパスを変更します。

このコマンドは、`--host`、`--password`、`--port`、`--socket` および `--user` オプションで指定されたオプションのみをログインパスから削除します。これらのオプションのいずれも指定しない場合、`remove` はログインパス全体を削除します。たとえば、このコマンドは、`mypath` ログインパス全体ではなく、`mypath` ログインパスから `user` オプションのみを削除します:

```
shell> mysql_config_editor remove --login-path=mypath --user
```

このコマンドは、`mypath` ログインパス全体を削除します:

```
shell> mysql_config_editor remove --login-path=mypath
```

`remove` コマンドでは、コマンド名の後に次のオプションを指定できます:

- `--help, -?`

`remove` コマンドのヘルプメッセージを表示して終了します。

一般的なヘルプメッセージを表示するには、`mysql_config_editor --help` を使用します。

- `--host, -h`

ホスト名をログインパスから削除します。

- `--login-path=name, -G name`

削除または変更するログインパス。このオプションが指定されていない場合、デフォルトのログインパス名は `client` です。

- `--password, -p`

パスワードをログインパスから削除します。

- `--port, -P`

TCP/IP ポート番号をログインパスから削除します。

- `--socket, -S`

Unix ソケットファイル名をログインパスから削除します。

- `--user, -u`

ユーザー名をログインパスから削除します。

- `--warn, -w`

コマンドがデフォルトのログインパス (`client`) を削除しようとし、`--login-path=client` が指定されていない場合に、ユーザーに警告して確認を求めます。このオプションはデフォルトで有効です。無効にするには、`--skip-warn` を使用します。

- `reset [options]`

ログインパスファイルの内容を空にします。

`reset` コマンドでは、コマンド名の後に次のオプションを指定できます:

- `--help, -?`

`reset` コマンドのヘルプメッセージを表示して終了します。

一般的なヘルプメッセージを表示するには、`mysql_config_editor --help` を使用します。

- `set [options]`

ログインパスファイルにログインパスを書き込みます。

このコマンドは、`--host`、`--password`、`--port`、`--socket` および `--user` オプションで指定されたオプションのみをログインパスに書き込みます。これらのオプションがいずれも指定されない場合は、`mysql_config_editor` はログインパスを空のグループとして書き出します。

`set` コマンドでは、コマンド名の後に次のオプションを指定できます:

- `--help, -?`

`set` コマンドのヘルプメッセージを表示して終了します。

一般的なヘルプメッセージを表示するには、`mysql_config_editor --help` を使用します。

- `--host=host_name, -h host_name`

ログインパスに書き出すホスト名。

- `--login-path=name, -G name`

作成するログインパス。このオプションが指定されていない場合、デフォルトのログインパス名は `client` です。

- `--password, -p`

ログインパスに書き出すパスワードを要求します。`mysql_config_editor` にプロンプトが表示されたら、パスワードを入力して Enter を押します。ほかのユーザーがパスワードを見るのを防ぐため、`mysql_config_editor` はエコーしません。

空のパスワードを指定するには、パスワードプロンプトで Enter を押します。ログインパスファイルに書き込まれるログインパスには、次のような行が含まれます:

```
password =
```

- `--port=port_num, -P port_num`

ログインパスに書き出す TCP/IP ポート番号。

- `--socket=file_name, -S file_name`

ログインパスに書き込む Unix ソケットファイル名。

- `--user=user_name, -u user_name`

ログインパスに書き出すユーザー名。

- `--warn, -w`

コマンドが既存のログインパスを上書きしようとした場合に、ユーザーに警告して確認を求めます。このオプションはデフォルトで有効です。無効にするには、`--skip-warn` を使用します。

4.6.8 mysqlbinlog — バイナリログファイルを処理するためのユーティリティ

サーバーのバイナリログは、データベースの内容に対する変更を記述する「イベント」を含むファイルで構成されます。サーバーはこれらのファイルをバイナリ形式で書き出します。内容をテキスト形式で表示するには、`mysqlbinlog` ユーティリティを使用します。また、`mysqlbinlog` を使用して、複製設定で複製サーバーによって書き込まれたリレーログファイルの内容を表示することもできます。これは、リレーログの形式がバイナリログと同じであるためです。バイナリログおよびリレーログは、[セクション5.4.4「バイナリログ」](#)および[セクション17.2.4「リレーログおよびレプリケーションメタデータリポジトリ」](#)でさらに説明します。

`mysqlbinlog` は次のように起動します。

```
shell> mysqlbinlog [options] log_file ...
```

たとえば `binlog.000003` という名前のバイナリログファイルの内容を表示するには、このコマンドを使用してください。

```
shell> mysqlbinlog binlog.000003
```

出力には、`binlog.000003` に含まれるイベントが含まれます。ステートメントベースのロギングでは、イベント情報には SQL ステートメント、それが実行されたサーバーの ID、ステートメントが実行されたタイムスタンプ、かかった時間などが含まれます。行ベースのロギングでは、イベントは SQL ステートメントではなく行の変更を示します。ロギングモードの詳細は[セクション17.2.1「レプリケーション形式」](#)を参照してください。

イベントには、その前に追加情報を提供するヘッダーコメントがあります。例:

```
# at 141
```



```
#100309 9:28:36 server id 123 end_log_pos 245
Query thread_id=3350 exec_time=11 error_code=0
```

最初の行で、`at` に続く数字はバイナリログファイル内でのイベントのファイルオフセット、つまり開始位置を示します。

2行目は、イベントが発生したサーバー上でステートメントがいつ開始されたかを示す日付と時間で始まります。レプリケーションの場合、このタイムスタンプはレプリカサーバーに伝播されます。`server id` は、イベントが発生したサーバーの `server_id` 値です。`end_log_pos` は、次のイベントの開始位置 (つまり、現在のイベントの終了位置 + 1) を示します。`thread_id` は、イベントを実行したスレッドを示します。`exec_time` は、レプリケーションソースサーバーでイベントの実行に費やされた時間です。レプリカでは、レプリカの終了実行時間からソースでの開始実行時間を差し引いた時間の差です。この違いは、ソースからどの程度遅れているレプリケーションかを示すインジケータとして機能します。`error_code` は、イベントの実行結果を示します。ゼロはエラーが発生しなかったことを意味します。

注記

イベントグループを使用する場合は、イベントのファイルオフセットのグループ化およびイベントのコメントのグループ化ができます。これらのグループ化イベントをブランクファイルオフセットと間違えないでください。

`mysqlbinlog` からの出力は、(たとえばそれを `mysql` への入力として使用することによって) 再実行し、ログ内のステートメントをやり直せます。これは、予期しないサーバー終了後のリカバリ操作に役立ちます。ほかの使用例は、このセクションのあとの方の説明および [セクション7.5「Point-in-Time \(増分\) リカバリ」](#) を参照してください。`mysqlbinlog` で使用される内部使用 `BINLOG` ステートメントを実行するには、`BINLOG_ADMIN` 権限 (または非推奨の `SUPER` 権限) か、`REPLICATION_APPLIER` 権限と各ログイベントを実行するための適切な権限が必要です。

`mysqlbinlog` を使用してバイナリログファイルを直接読み取り、ローカル MySQL サーバーに適用できます。`--read-from-remote-server` オプションを使用して、リモートサーバーからバイナリログを読み取ることもできます。リモートバイナリログを読み取るために、接続パラメータオプションを指定してサーバーへの接続方法を示すことができます。これらのオプションは `--host`、`--password`、`--port`、`--protocol`、`--socket`、および `--user` です。

バイナリログファイルが暗号化されている場合 (MySQL 8.0.14 以降で実行可能)、`mysqlbinlog` はそれらを直接読み取ることができませんが、`--read-from-remote-server` オプションを使用してサーバーから読み取ることができます。バイナリログファイルは、サーバーの `binlog_encryption` システム変数が `ON` に設定されている場合に暗号化されます。`SHOW BINARY LOGS` ステートメントは、特定のバイナリログファイルが暗号化されているか暗号化されていないかを示します。暗号化されたバイナリログファイルと暗号化されていないバイナリログファイルは、暗号化されたログファイル (`0xFD62696E`) のファイルヘッダーの先頭にあるマジック番号を使用して区別することもできます。これは、暗号化されていないログファイル (`0xFE62696E`) に使用されるものとは異なります。暗号化されたバイナリログファイルを直接読み取ろうとしたが、古いバージョンの `mysqlbinlog` がそのファイルをバイナリログファイルとして認識しない場合、MySQL 8.0.14 から適切なエラーが返されることに注意してください。バイナリログの暗号化の詳細は、[セクション17.3.2「バイナリログファイルとリレーログファイルの暗号化」](#) を参照してください。

バイナリログトランザクションペイロードが圧縮されている場合 (MySQL 8.0.20 以降で実行可能)、そのリリースの `mysqlbinlog` バージョンでは、トランザクションペイロードを自動的に解凍してデコードし、圧縮解除されたイベントと同様に出力します。古いバージョンの `mysqlbinlog` では、圧縮されたトランザクションペイロードを読み取ることができません。サーバー `binlog_transaction_compression` システム変数が `ON` に設定されている場合、トランザクションペイロードは圧縮され、単一のイベント (`Transaction_payload_event`) としてサーバーバイナリログファイルに書き込まれます。`--verbose` オプションを使用すると、`mysqlbinlog` によって、使用される圧縮アルゴリズム、最初に受信された圧縮済ペイロードサイズ、および解凍後の結果のペイロードサイズを示すコメントが追加されます。

注記

圧縮されたトランザクションペイロードの一部である個々のイベントに対して `mysqlbinlog` が示す終了位置 (`end_log_pos`) は、元の圧縮されたペイロードの終了位置と同じです。したがって、複数の解凍されたイベントの終了位置を同じにすることができます。

`mysqlbinlog` 独自の接続圧縮は、トランザクションペイロードがすでに圧縮されている場合は少なくなります。圧縮されていないトランザクションおよびヘッダーでは引き続き動作します。

バイナリログのトランザクション圧縮の詳細は、[セクション5.4.4.5「バイナリログトランザクション圧縮」](#) を参照してください。

mysqlbinlog を大規模なバイナリログに対して実行する場合は、結果のファイルのためにファイルシステムに十分なスペースがあるように注意してください。mysqlbinlog が一時ファイル用に使用するディレクトリを構成するには、TMPDIR 環境変数を使用します。

mysqlbinlog は、SQL ステートメントを実行する前に pseudo_slave_mode の値を true に設定します。このシステム変数は、XA トランザクションの処理、original_commit_timestamp レプリケーション遅延タイムスタンプと original_server_version システム変数、およびサポートされていない SQL モードに影響します。

mysqlbinlog は次のオプションをサポートします。これらはコマンド行またはオプションファイルの [mysqlbinlog] グループおよび [client] グループで指定できます。MySQL プログラムによって使用されるオプションファイルの詳細については、セクション4.2.2.2「オプションファイルの使用」を参照してください。

表 4.21 「mysqlbinlog のオプション」

オプション名	説明	導入	非推奨
--base64-output	バイナリログのエントリを base-64 エンコードで出力		
--bind-address	指定されたネットワークインタフェースを使用して MySQL サーバーに接続		
--binlog-row-event-max-size	バイナリログの最大イベントサイズ		
--character-sets-dir	文字セットがインストールされているディレクトリ		
--compress	クライアントとサーバー間で送信される情報をすべて圧縮	8.0.17	8.0.18
--compression-algorithms	サーバーへの接続に許可される圧縮アルゴリズム	8.0.18	
--connection-server-id	テストとデバッグに使用。適用されるデフォルト値およびその他の事項については、テキストを参照してください		
--database	このデータベースのみのエントリをリスト		
--debug	デバッグログの書込み		
--debug-check	プログラムの終了時にデバッグ情報を出力		
--debug-info	プログラムの終了時に、デバッグ情報、メモリー、および CPU の統計を出力		
--default-auth	使用する認証プラグイン		
--defaults-extra-file	通常のオプションファイルに加えて、名前付きオプションファイルを読み取ります		
--defaults-file	指名されたオプションファイルのみを読み取る		
--defaults-group-suffix	オプショングループのサフィクス値		
--disable-log-bin	バイナリロギングを無効化		
--exclude-gtids	提供された GTID セットのグループを表示しない		

オプション名	説明	導入	非推奨
<code>--force-if-open</code>	バイナリログファイルが開いているか適切にクローズしていない場合でも読み取る		
<code>--force-read</code>	mysqlbinlog が認識しないバイナリログイベントを読み取った場合、警告を出力		
<code>--get-server-public-key</code>	サーバーから RSA 公開キーをリクエスト		
<code>--help</code>	ヘルプメッセージを表示して終了		
<code>--hexdump</code>	コメント内にログの 16 進ダンプを表示		
<code>--host</code>	MySQL サーバーがあるホスト		
<code>--idempotent</code>	サーバーが、このセッションからのバイナリログの更新を処理する間のみ、べき乗モードを使用		
<code>--include-gtids</code>	提供された GTID セットのグループのみを表示		
<code>--local-load</code>	指定されたディレクトリに LOAD DATA のローカル一時ファイルを準備		
<code>--login-path</code>	ログインパスオプションを .mylogin.cnf から読み取り		
<code>--no-defaults</code>	オプションファイルを読み取らない		
<code>--offset</code>	ログの最初の N エントリをスキップ		
<code>--password</code>	サーバーに接続する際に使用するパスワード		
<code>--plugin-dir</code>	プラグインがインストールされているディレクトリ		
<code>--port</code>	接続用の TCP/IP ポート番号		
<code>--print-defaults</code>	デフォルトオプションの印刷		
<code>--print-table-metadata</code>	テーブルメタデータの印刷		
<code>--protocol</code>	使用するトランスポートプロトコル		
<code>--raw</code>	イベントを生の (バイナリ) 形式で出力ファイルに書き込み		
<code>--read-from-remote-master</code>	バイナリログをローカルログファイルからではなく MySQL マスターから読み取り		

オプション名	説明	導入	非推奨
<code>--read-from-remote-server</code>	バイナリログをローカルログファイルからではなく MySQL サーバーから読み取り		
<code>--require-row-format</code>	行ベースのバイナリロギング形式が必要	8.0.19	
<code>--result-file</code>	指定されたファイルに出力を送信		
<code>--rewrite-db</code>	生ベースの形式で書き込まれたログから再現する場合の、データベースの書き換えルールを作成。複数回使用できます		
<code>--server-id</code>	指定されたサーバー ID を持つサーバーによって作成されたイベントのみを抽出		
<code>--server-id-bits</code>	サーバー ID ビットが最大値未満に設定されている mysqld によってログが書き込まれた場合に、バイナリログのサーバー ID を解釈する方法を、mysqlbinlog に対して指定。MySQL Cluster バージョンの mysqlbinlog でのみサポート		
<code>--server-public-key-path</code>	RSA 公開鍵を含むファイルへのパス名		
<code>--set-charset</code>	SET NAMES charset_name ステートメントを出力に追加		
<code>--shared-memory-base-name</code>	共有メモリー接続用の共有メモリー名 (Windows のみ)		
<code>--short-form</code>	ログに含まれるステートメントのみを表示		
<code>--skip-gtids</code>	GTID を出力しない。これは、GTID を含むバイナリログからのダンプファイルを書き込む場合に使用します		
<code>--socket</code>	使用する Unix ソケットファイルまたは Windows 名前付きパイプ		
<code>--ssl-ca</code>	信頼できる SSL 認証局のリストを含むファイル		
<code>--ssl-capath</code>	信頼できる SSL 認証局の証明書ファイルを含むディレクトリ		
<code>--ssl-cert</code>	X.509 証明書を含むファイル		
<code>--ssl-cipher</code>	接続の暗号化に許可される暗号		

オプション名	説明	導入	非推奨
<code>--ssl-crl</code>	証明書失効リストを含むファイル		
<code>--ssl-crlpath</code>	証明書失効リストファイルを含むディレクトリ		
<code>--ssl-fips-mode</code>	クライアント側で FIPS モードを有効にするかどうか		
<code>--ssl-key</code>	X.509 キーを含むファイル		
<code>--ssl-mode</code>	サーバーへの接続に必要なセキュリティ状態		
<code>--start-datetime</code>	タイムスタンプが <code>datetime</code> 引数と同じかそれよりあとの最初のイベントからバイナリログを読み取り		
<code>--start-position</code>	引数以上の位置を持つ最初のイベントからバイナリログをデコード		
<code>--stop-datetime</code>	タイムスタンプが <code>datetime</code> 引数と同じかそれより大きい最初のイベントでバイナリログの読み取りを停止		
<code>--stop-never</code>	最後のバイナリログファイルの読み取り後、サーバーとの接続を維持		
<code>--stop-never-slave-server-id</code>	サーバーへの接続時にレポートするスレーブサーバー ID		
<code>--stop-position</code>	引数以上の位置を持つ最初のイベントでバイナリログのデコードを停止		
<code>--tls-ciphersuites</code>	暗号化された接続に許可される TLSv1.3 暗号スイート	8.0.16	
<code>--tls-version</code>	暗号化された接続に許可される TLS プロトコル		
<code>--to-last-log</code>	MySQL サーバーから要求されたバイナリログの最後で停止せず、最後のバイナリログまで続けて出力		
<code>--user</code>	サーバーへの接続時に使用する MySQL ユーザー名		
<code>--verbose</code>	行イベントを SQL ステートメントとして再構築		
<code>--verify-binlog-checksum</code>	バイナリログのチェックサムを検証		
<code>--version</code>	バージョン情報を表示して終了		
<code>--zstd-compression-level</code>	zstd 圧縮を使用するサーバーへの接続の圧縮レベル	8.0.18	

- `--help, -?`

ヘルプメッセージを表示して終了します。

- `--base64-output=value`

このオプションは、イベントをいつ **BINLOG** ステートメントを使用して、base-64 文字列としてエンコードして表示するべきかを決定します。このオプションには次の許容値があります (大/小文字は区別されません):

- **AUTO** (「自動」) または **UNSPEC** (「未指定」) では、必要なときに (すなわち、形式記述イベントおよび行イベント) 自動的に **BINLOG** ステートメントを表示します。 `--base64-output` オプションが指定されない場合は、効果は `--base64-output=AUTO` と同じです。

注記

自動的な **BINLOG** 表示は、`mysqlbinlog` の出力を使用してバイナリログファイルの内容を再実行する場合には、唯一の安全な動作です。その他のオプション値はデバッグまたはテスト専用です。これらのオプションで生成される出力には、すべてのイベントが実行可能な形式で含まれるわけではないからです。

- **NEVER** を使用すると **BINLOG** ステートメントは表示されなくなります。 `mysqlbinlog` は、**BINLOG** を使用して表示しなければならない行イベントが検出された場合にはエラーで終了します。
- **DECODE-ROWS** は、`--verbose` オプションも指定することによって、行イベントをコメント付きの SQL ステートメントとしてデコードおよび表示することをユーザーが意図していることを、`mysqlbinlog` に指定します。 **NEVER** と同様に、**DECODE-ROWS** は **BINLOG** ステートメントの表示を抑制しますが、**NEVER** とは異なり、行イベントが検出されてもエラーで終了しません。

`--base64-output` および `--verbose` の行イベント出力への影響を示す例は、[セクション4.6.8.2「mysqlbinlog 行イベントの表示」](#)を参照してください。

- `--bind-address=ip_address`

複数のネットワークインタフェースを持つコンピュータで、このオプションを使用して、MySQL サーバーへの接続に使用するインタフェースを選択します。

- `--binlog-row-event-max-size=N`

コマンド行形式	<code>--binlog-row-event-max-size=#</code>
型	数値
デフォルト値	4294967040
最小値	256
最大値	18446744073709547520

行ベースのバイナリログイベントの最大サイズをバイト単位で指定します。可能であれば、行はこのサイズより小さいイベントにグループ化されます。値は 256 の倍数であるべきです。デフォルトは 4G バイトです。

- `--character-sets-dir=dir_name`

文字セットがインストールされているディレクトリ。 [セクション10.15「文字セットの構成」](#)を参照してください。

- `--compress`

可能であれば、クライアントとサーバーの間で送信されるすべての情報を圧縮します。 [セクション4.2.8「接続圧縮制御」](#)を参照してください。

このオプションは MySQL 8.0.17 で追加されました。MySQL 8.0.18 では非推奨です。MySQL の将来のバージョンで削除されることが予想されます。 [レガシー接続圧縮の構成](#)を参照してください。

- `--compression-algorithms=value`

サーバーへの接続に許可される圧縮アルゴリズム。使用可能なアルゴリズムは、`protocol_compression_algorithms` システム変数の場合と同じです。デフォルト値は `uncompressed` です。

詳細は、[セクション4.2.8「接続圧縮制御」](#)を参照してください。

このオプションは MySQL 8.0.18 で追加されました。

- `--connection-server-id=server_id`

`--connection-server-id` は、`mysqlbinlog` がサーバーへの接続時にレポートするサーバー ID を指定します。これは、レプリカサーバーまたは別の `mysqlbinlog` プロセスの ID との競合を回避するために使用できます。

`--read-from-remote-server` オプションが指定されている場合、`mysqlbinlog` はサーバー ID 0 を報告します。これは、最後のログファイルを送信した後に切断するようにサーバーに指示します (非ブロッキング動作)。サーバーへの接続を維持するために `--stop-never` オプションも指定されている場合、`mysqlbinlog` はデフォルトで 0 の代わりに 1 のサーバー ID を報告し、必要に応じて `--connection-server-id` を使用してそのサーバー ID を置き換えることができます。[セクション4.6.8.4「mysqlbinlog サーバー ID の指定」](#)を参照してください。

- `--database=db_name, -d db_name`

このオプションを使用すると、`mysqlbinlog` は、`USE` によって `db_name` がデフォルトデータベースとして選択されている間に発生するバイナリログ (ローカルログのみ) からのエントリを出力するようになります。

`mysqlbinlog` の `--database` オプションは、`mysqld` の `--binlog-do-db` オプションと同様ですが、指定できるのは 1 つのデータベースのみです。`--database` を複数回指定すると、最後のインスタンスのみが使用されます。

このオプションの影響は、ステートメントベースまたは行ベースのロギング形式のどちらが使用されているかによって異なります。これは、`--binlog-do-db` の影響がステートメントベースまたは行ベースのいずれのロギングが使用されているかによって異なるのと同じです。

ステートメントベースのロギング。 `--database` オプションは次のように機能します。

- `db_name` がデフォルトデータベースである間、ステートメントは `db_name` または別のデータベースのテーブルを修正する場合でも出力されます。
- `db_name` がデフォルトデータベースとして選択されていない場合は、`db_name` のテーブルを修正する場合でもステートメントは出力されません。
- `CREATE DATABASE`、`ALTER DATABASE`、および `DROP DATABASE` は例外です。ステートメントを出力するかどうかを判断するときには、作成、変更、または削除されたデータベースがデフォルトのデータベースであるとみなされます。

これらのステートメントを実行することによって、ステートメントベースのロギングを使用してバイナリログが作成されたとします。

```
INSERT INTO test.t1 (i) VALUES(100);
INSERT INTO db2.t2 (j) VALUES(200);
USE test;
INSERT INTO test.t1 (i) VALUES(101);
INSERT INTO t1 (i) VALUES(102);
INSERT INTO db2.t2 (j) VALUES(201);
USE db2;
INSERT INTO test.t1 (i) VALUES(103);
INSERT INTO db2.t2 (j) VALUES(202);
INSERT INTO t2 (j) VALUES(203);
```

デフォルトデータベースがないため、`mysqlbinlog --database=test` は最初の 2 つの `INSERT` ステートメントを出力しません。`USE test` に続く 3 つの `INSERT` ステートメントは出力しますが、`USE db2` に続く 3 つの `INSERT` ステートメントは出力しません。

デフォルトデータベースがないため、`mysqlbinlog --database=db2` は最初の 2 つの `INSERT` ステートメントを出力しません。`USE test` に続く 3 つの `INSERT` ステートメントは出力しませんが、`USE db2` に続く 3 つの `INSERT` ステートメントは出力します。

行ベースのロギング。 `mysqlbinlog` は、`db_name` に属するテーブルを変更するエントリのみを出力します。これにはデフォルトのデータベースには影響しません。今説明したバイナリログが、ステートメントベースのロギングではなく行ベースのロギングを使用して作成されたとします。`mysqlbinlog --database=test` は、`USE` が発行されたか、またはデフォルトのデータベースが何かにかかわらず、テストデータベースの `t1` を変更するエントリのみを出力します。

サーバーが、`binlog_format` が `MIXED` に設定された状態で稼働していて、`mysqlbinlog` を `--database` オプションで使えるようにする場合、変更されるテーブルが `USE` で選択されたデータベースであることを保証する必要があります。(特に、クロスデータベースの更新は使用しないようにしてください。)

`--rewrite-db` オプションとともに使用すると、最初に `--rewrite-db` オプションが適用され、リライトされたデータベース名を使用して `--database` オプションが適用されます。オプションが指定されている順序に違いはありません。

- `--debug[=debug_options], -# [debug_options]`

デバッグのログを書き込みます。一般的な `debug_options` 文字列は `d:t:o,file_name` です。デフォルトは `d:t:o,/tmp/mysqlbinlog.trace` です。

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `--debug-check`

プログラムの終了時に、デバッグ情報を出力します。

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `--debug-info`

プログラムの終了時に、デバッグ情報とメモリーおよび CPU 使用率の統計を出力します。

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `--default-auth=plugin`

使用するクライアント側認証プラグインに関するヒント。 [セクション6.2.17「プラグブル認証」](#) を参照してください。

- `--defaults-extra-file=file_name`

このオプションファイルは、グローバルオプションファイルのあとに読み取りますが、(UNIX では) ユーザーオプションファイルの前に読み取るようにしてください。ファイルが存在しないかアクセスできない場合、エラーが発生します。`file_name` は、フルパス名でなく相対パス名として指定された場合、現行ディレクトリを基準にして解釈されます。

このオプションおよびその他のオプションファイルオプションの詳細は、 [セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--defaults-file=file_name`

指定されたオプションファイルのみ使用します。ファイルが存在しないかアクセスできない場合、エラーが発生します。`file_name` は、フルパス名でなく相対パス名として指定された場合、現行ディレクトリを基準にして解釈されます。

例外: `--defaults-file` でも、クライアントプログラムは `.mylogin.cnf` を読み取ります。

このオプションおよびその他のオプションファイルオプションの詳細は、 [セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--defaults-group-suffix=str`

通常のオプショングループだけでなく、通常の名前に `str` のサフィクスが付いたグループも読み取ります。たとえば、`mysqlbinlog` は通常 `[client]` グループおよび `[mysqlbinlog]` グループを読み取ります。 `--defaults-group-suffix=_other` オプションを指定した場合、`mysqlbinlog` は `[client_other]` グループおよび `[mysqlbinlog_other]` グループも読み取ります。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--disable-log-bin, -D`

バイナリロギングを無効化します。これは、`--to-last-log` オプションを使用して同じ MySQL サーバーに対して出力を送信している場合、無限ループを回避するのに便利です。このオプションは、ログに記録したステートメントの重複を回避するために、予期しない終了後にリストアする場合にも役立ちます。

このオプションを使用すると、`mysqlbinlog` の出力に `SET sql_log_bin = 0` ステートメントが含まれ、残りの出力のバイナリロギングが無効になります。 `sql_log_bin` システム変数のセッション値の操作は制限付き操作であるため、このオプションを使用するには、制限付きセッション変数を設定するのに十分な権限が必要です。[セクション5.1.9.1「システム変数権限」](#) を参照してください。

- `--exclude-gtids=gtid_set`

`gtid_set` にリストされたグループを表示しません。

- `--force-if-open, -F`

バイナリログファイルが開いているか適切に閉じられていない場合でも読み取ります。

- `--force-read, -f`

このオプションでは、`mysqlbinlog` が認識できないバイナリログイベントを読み取った場合に、警告を出力し、イベントを無視して続行します。このオプションを使用しない場合は、`mysqlbinlog` はそのようなイベントを読み取ることを停止します。

- `--get-server-public-key`

RSA キーベースのパスワード交換に必要な公開キーをサーバーにリクエストします。このオプションは、`cached_sha2_password` 認証プラグインで認証されるクライアントに適用されます。そのプラグインの場合、サーバーは要求されないかぎり公開鍵を送信しません。このオプションは、そのプラグインで認証されないアカウントでは無視されます。クライアントがセキュアな接続を使用してサーバーに接続する場合と同様に、RSA ベースのパスワード交換を使用しない場合も無視されます。

`--server-public-key-path=file_name` が指定され、有効な公開キーファイルが指定されている場合は、`--get-server-public-key` よりも優先されます。

`cached_sha2_password` プラグインの詳細は、[セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#) を参照してください。

- `--hexdump, -H`

[セクション4.6.8.1「mysqlbinlog 16 進ダンプ形式」](#) に説明されているように、ログの 16 進ダンプをコメントに表示します。16 進出力はアプリケーションのデバッグに便利な場合があります。

- `--host=host_name, -h host_name`

指定されたホストの MySQL サーバーからバイナリログを取得します。

- `--idempotent`

更新の処理中に多重呼出し不変モードを使用するように MySQL Server に指示します。これにより、更新の処理中に現在のセッションでサーバーが検出した重複キーまたはキーが見つからないエラーが抑制されます。このオプションは、必要に応じて、またはログが参照するすべてのデータが含まれていない可能性がある MySQL Server にバイナリログをリプレイする必要がある場合に便利です。

このオプションの効果の範囲には、現在の `mysqlbinlog` クライアントおよびセッションのみが含まれます。

- `--include-gtids=gtid_set`

`gtid_set` にリストされたグループのみを表示します。

- `--local-load=dir_name, -l dir_name`

`LOAD DATA` ステートメントに対応するデータロード操作の場合、`mysqlbinlog` はバイナリログイベントからファイルを抽出し、一時的なファイルとしてローカルファイルシステムに書き込み、`LOAD DATA LOCAL` ステートメントを書き込んでファイルをロードします。デフォルトでは、`mysqlbinlog` はこれらの一時ファイルをオペレーティングシステム固有のディレクトリに書き込みます。`--local-load` オプションを使用すると、`mysqlbinlog` がローカル一時ファイルを準備するディレクトリを明示的に指定できます。

他のプロセスはデフォルトのシステム固有のディレクトリにファイルを書き込むことができるため、`mysqlbinlog` に `--local-load` オプションを指定してデータファイルに別のディレクトリを指定し、`mysqlbinlog` からの出力を処理するときに `mysql` に `--load-data-local-dir` オプションを指定して同じディレクトリを指定することをお勧めします。例:

```
mysqlbinlog --local-load=/my/local/data ...
| mysql --load-data-local-dir=/my/local/data ...
```

重要

これらの一時ファイルは、`mysqlbinlog` およびその他のどの MySQL プログラムによっても自動的に削除されません。

- `--login-path=name`

`.mylogin.cnf` ログインパスファイルの指定されたログインパスからオプションを読み取ります。「ログインパス」は、接続先の MySQL サーバーおよび認証に使用するアカウントを指定するオプションを含むオプショングループです。ログインパスファイルを作成または変更するには、`mysql_config_editor` ユーティリティを使用します。セクション4.6.7「`mysql_config_editor` — MySQL 構成ユーティリティ」を参照してください。

このオプションおよびその他のオプションファイルオプションの詳細は、セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」を参照してください。

- `--no-defaults`

オプションファイルを読み取りません。オプションファイルから不明のオプションを読み取ることが原因でプログラムの起動に失敗する場合、`--no-defaults` を使用して、オプションを読み取らないようにできます。

例外として、`.mylogin.cnf` ファイルは、存在する場合はすべての場合に読み取られます。これにより、`--no-defaults` が使用された場合にも、コマンド行よりも安全な方法でパスワードを指定できます。(`.mylogin.cnf` は `mysql_config_editor` ユーティリティによって作成されます。セクション4.6.7「`mysql_config_editor` — MySQL 構成ユーティリティ」を参照してください)。

このオプションおよびその他のオプションファイルオプションの詳細は、セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」を参照してください。

- `--offset=N, -o N`

ログの最初の `N` 個のエントリをスキップします。

- `--open-files-limit=N`

予約するオープンファイルディスクリプタの数を指定します。

- `--password[=password], -p[password]`

サーバーへの接続に使用される MySQL アカウントのパスワード。パスワード値はオプションです。指定しない場合、`mysqlbinlog` によってプロンプトが表示されます。指定する場合は、`--password=` または `-p` とそれに続くパス

ワードの間にスペースなしが存在する必要があります。パスワードオプションを指定しない場合、デフォルトではパスワードは送信されません。

コマンド行でのパスワード指定は、セキュアでないと考えるべきです。コマンド行でパスワードを指定しないようにするには、オプションファイルを使用します。 [セクション6.1.2.1「パスワードセキュリティのためのエンドユーザーガイドライン」](#)を参照してください。

パスワードがなく、`mysqlbinlog` でパスワードの入力を求められないように明示的に指定するには、`--skip-password` オプションを使用します。

- `--plugin-dir=dir_name`

プラグインを検索するディレクトリ。このオプションは、`--default-auth` オプションを使用して認証プラグインを指定しても、`mysqlbinlog` がそれを見出さない場合に指定します。 [セクション6.2.17「プラグイン認証」](#)を参照してください。

- `--port=port_num, -P port_num`

リモートサーバーへの接続に使用する TCP/IP ポート番号。

- `--print-defaults`

プログラム名と、オプションファイルから受け取るすべてのオプションを出力します。

このオプションおよびその他のオプションファイルオプションの詳細は、 [セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--print-table-metadata`

バイナリログからテーブル関連のメタデータを出力します。 `binlog-row-metadata` を使用して、ログに記録されるテーブル関連のメタデータバイナリの量を構成します。

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

サーバーへの接続に使用するトランスポートプロトコル。これは、他の接続パラメータが通常、必要なプロトコル以外のプロトコルを使用する場合に便利です。許可される値の詳細は、 [セクション4.2.7「接続トランスポートプロトコル」](#)を参照してください。

- `--raw`

デフォルトでは、`mysqlbinlog` はバイナリログファイルを読み取り、イベントをテキスト形式で書き出します。 `--raw` オプションは `mysqlbinlog` に対して、元のバイナリ形式で書き出すことを指示します。ファイルはサーバーから要求されるため、これを使用するには、`--read-from-remote-server` も使用する必要があります。`mysqlbinlog` はサーバーから読み取った各ファイルに対して1つの出力ファイルを書き出します。`--raw` オプションは、サーバーのバイナリログのバックアップを作成するために使用できます。`--stop-never` オプションを使用すると、`mysqlbinlog` はサーバーに接続されたままになるため、バックアップは「ライブ」です。デフォルトでは、出力ファイルは現在のディレクトリに元のログファイルと同じ名前を書き出されます。出力ファイル名は `--result-file` オプションを使用して変更できます。詳細は、 [セクション4.6.8.3「バイナリログファイルのバックアップのためのmysqlbinlogの使用」](#)を参照してください。

- `--read-from-remote-master=type`

`COM_BINLOG_DUMP` コマンドまたは `COM_BINLOG_DUMP_GTID` コマンドで、オプション値をそれぞれ `BINLOG-DUMP-NON-GTIDS` または `BINLOG-DUMP-GTIDS` に設定して、MySQL サーバーからバイナリログを読み取ります。`--read-from-remote-master=BINLOG-DUMP-GTIDS` が `--exclude-gtids` と組み合わされている場合、不要なネットワークトラフィックを回避するために、ソースでトランザクションをフィルタで除外できます。

接続パラメータオプションは、このオプションまたは `--read-from-remote-server` オプションとともに使用します。これらのオプションは `--host`、`--password`、`--port`、`--protocol`、`--socket`、および `--user` です。どちらのリモートオプションも指定しない場合、接続パラメータオプションは無視されます。

このオプションを使用するには、`REPLICATION SLAVE` 権限が必要です。

- `--read-from-remote-server, -R`

バイナリログをローカルログファイルからではなく MySQL サーバーから読み取ります。このオプションでは、リモートサーバーが稼働していることが必要です。リモートサーバー上のバイナリログファイルに対してのみ機能します。リレーログファイルには機能しません。

接続パラメータオプションは、このオプションまたは `--read-from-remote-master` オプションとともに使用します。これらのオプションは `--host`、`--password`、`--port`、`--protocol`、`--socket`、および `--user` です。どちらのリモートオプションも指定しない場合、接続パラメータオプションは無視されます。

このオプションを使用するには、`REPLICATION SLAVE` 権限が必要です。

このオプションは、`--read-from-remote-master=BINLOG-DUMP-NON-GTIDS` に似ています。

- `--result-file=name, -r name`

`--raw` オプションがない場合は、このオプションは `mysqlbinlog` がテキスト出力を書き出すファイルを示します。 `--raw` がある場合は、`mysqlbinlog` はサーバーから転送される各ログファイルに対して 1 つのバイナリ出力ファイルを、デフォルトでは現在のディレクトリに元のログファイルと同じ名前で作成して書き出します。この場合、`--result-file` オプションの値は出力ファイル名を変更するプリフィクスとして処理されます。

- `--require-row-format`

イベントの行ベースのバイナリロギング形式が必要です。このオプションは、`mysqlbinlog` 出力に行ベースのレプリケーションイベントを強制します。このオプションを使用して生成されたイベントのストリームは、(MySQL 8.0.23 の) `CHANGE REPLICATION SOURCE TO` ステートメントまたは `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 の前) の `REQUIRE_ROW_FORMAT` オプションを使用して保護されたレプリケーションチャンネルによって受け入れられます。 `binlog_format=ROW` は、バイナリログが書き込まれたサーバーに設定する必要があります。このオプションを指定すると、`LOAD DATA INFILE` 命令、一時テーブルの作成または削除、`INTVAR`、`RAND` または `USER_VAR` イベント、DML トランザクション内の行ベース以外のイベントなど、`REQUIRE_ROW_FORMAT` 制限の下で許可されていないイベントが発生した場合、`mysqlbinlog` はエラーメッセージで停止します。また、`mysqlbinlog` は、制限を適用するためその出力の開始時に、出力が実行されたが `SET @@session.pseudo_thread_id` ステートメントがプリントされない場合、`SET @@session.require_row_format` ステートメントをプリントします。

このオプションは MySQL 8.0.19 で追加されました。

- `--rewrite-db='from_name->to_name'`

行ベースまたはステートメントベースのログから読み取る場合は、すべての `from_name` を `to_name` にリライトします。リライトは、ステートメントベースのログの行、行ベースのログおよび `USE` 句に対して実行されます。

警告

このオプションを使用する場合、テーブル名がデータベース名で修飾されているステートメントは、新しい名前を使用するようにリライトされません。

このオプションの値として使用されるリライトルールは、前述のように `'from_name->to_name'` という形式の文字列であり、このため引用符で囲む必要があります。

複数のリライトルールを使用するには、次に示すように、オプションを複数回指定します:

```
shell> mysqlbinlog --rewrite-db='dbcurent->dbold' --rewrite-db='dbtest->dbcurent' \  
binlog.00001 > /tmp/statements.sql
```

`--database` オプションとともに使用すると、最初に `--rewrite-db` オプションが適用され、リライトされたデータベース名を使用して `--database` オプションが適用されます。オプションが指定されている順序に違いはありません。

これは、たとえば、`mysqlbinlog` が `--rewrite-db='mydb->yourdb' --database=yourdb` で起動された場合、データベース `mydb` および `yourdb` 内のテーブルに対するすべての更新が出力に含まれることを意味します。一方、`--rewrite-db='mydb->yourdb' --database=mydb` で起動された場合、`mysqlbinlog` はステートメントをまったく出力しません: `mydb` へのすべての更新は、`--database` オプションを適用する前に `yourdb` への更新として最初にリライトされるため、`--database=mydb` に一致する更新は残っていません。

- `--server-id=id`

指定されたサーバー ID を持つサーバーによって作成されたイベントのみを表示します。

- `--server-id-bits=N`

サーバーを特定するために、`server_id` の最初の `N` ビットのみを使用します。 `server-id-bits` が 32 未満にセットされ、ユーザーデータが最上位ビットに保存される `mysql` によってバイナリログが書き出された場合、`--server-id-bits` を 32 にセットして `mysqlbinlog` を実行するとこのデータを表示できます。

このオプションは、NDB Cluster ディストリビューションで提供される `mysqlbinlog` のバージョン、または NDB Cluster サポートで構築されたバージョンでのみサポートされます。

- `--server-public-key-path=file_name`

RSA キーペアベースのパスワード交換のためにサーバーが必要とする公開キーのクライアント側コピーを含む、PEM 形式のファイルへのパス名。このオプションは、`sha256_password` または `caching_sha2_password` 認証プラグインで認証されるクライアントに適用されます。これらのプラグインのいずれかで認証されないアカウントでは、このオプションは無視されます。クライアントがセキュアな接続を使用してサーバーに接続する場合と同様に、RSA ベースのパスワード交換を使用しない場合も無視されます。

`--server-public-key-path=file_name` が指定され、有効な公開キーファイルが指定されている場合は、`--get-server-public-key` よりも優先されます。

`sha256_password` の場合、このオプションは、MySQL が OpenSSL を使用して構築された場合にのみ適用されます。

`sha256_password` および `caching_sha2_password` プラグインの詳細は、[セクション6.4.1.3「SHA-256 プラガブル認証」](#) および [セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#) を参照してください。

- `--set-charset=charset_name`

`SET NAMES charset_name` ステートメントを出力に追加して、ログファイルの処理に使用される文字セットを指定します。

- `--shared-memory-base-name=name`

Windows の場合、共有メモリを使用してローカルサーバーに接続するために使用する共有メモリ名。デフォルト値は `MYSQL` です。共有メモリ名では大文字と小文字が区別されます。

このオプションは、共有メモリ接続をサポートするために `shared_memory` システム変数を有効にしてサーバーを起動した場合にのみ適用されます。

- `--short-form, -s`

追加情報および行ベースのイベントなしで、ログに含まれるステートメントのみを表示します。これはテスト専用で、本番環境のシステムで使用するべきではありません。これは非推奨であり、将来のリリースで削除される予定です。

- `--skip-gtids[=(true|false)]`

出力に GTID を表示しません。これは、次の例に示すように GTID を含む 1 つまたは複数のバイナリログからダンプファイルに書き出す場合に必要です。

```
shell> mysqlbinlog --skip-gtids binlog.000001 > /tmp/dump.sql
shell> mysqlbinlog --skip-gtids binlog.000002 >> /tmp/dump.sql
shell> mysql -u root -p -e "source /tmp/dump.sql"
```

そうでない場合には、このオプションを本番環境で使用することは、通常推奨されません。

- `--socket=path, -S path`

`localhost` への接続用に使用する、Unix ソケットファイル、または Windows では使用する名前付きパイプの名前。

Windows では、このオプションは、名前付きパイプ接続をサポートするために `named_pipe` システム変数を有効にしてサーバーを起動した場合にのみ適用されます。また、接続を行うユーザーは、`named_pipe_full_access_group` システム変数で指定された Windows グループのメンバーである必要があります。

- `--ssl*`

`--ssl` で始まるオプションは、SSL を使用してサーバーに接続するかどうかを指定し、SSL 鍵および証明書を検索する場所を指定します。暗号化接続のコマンドオプションを参照してください。

- `--ssl-fips-mode={OFF|ON|STRICT}`

クライアント側で FIPS モードを有効にするかどうかを制御します。`--ssl-fips-mode` オプションは、暗号化された接続の確立には使用されず、許可する暗号化操作に影響する点で、他の `--ssl-xxx` オプションとは異なります。セクション 6.8 「FIPS のサポート」を参照してください。

次の `--ssl-fips-mode` 値を使用できます:

- **OFF**: FIPS モードを無効にします。
- **ON**: FIPS モードを有効にします。
- **STRICT**: 「strict」 FIPS モードを有効にします。

注記

OpenSSL FIPS オブジェクトモジュールが使用できない場合、`--ssl-fips-mode` に許可される値は **OFF** のみです。この場合、`--ssl-fips-mode` を **ON** または **STRICT** に設定すると、クライアントは起動時に警告を生成し、FIPS 以外のモードで動作します。

- `--start-datetime=datetime`

`datetime` 引数と同じかそれより遅いタイムスタンプを持つ最初のイベントから、バイナリログの読み取りを始めます。`datetime` 値は、`mysqlbinlog` を実行するマシンのローカルタイムゾーンに相対的です。値は **DATETIME** または **TIMESTAMP** データ型に受け付けられる形式にしてください。例:

```
shell> mysqlbinlog --start-datetime="2005-12-25 11:25:56" binlog.000003
```

このオプションはポイントインタイムリカバリに便利です。セクション 7.5 「Point-in-Time (増分) リカバリ」を参照してください。

- `--start-position=N, -j N`

ログ位置 `N` でバイナリログのデコードを開始します。出力には、位置 `N` 以降で開始するイベントも含まれます。位置はログファイル内のバイトポイントであり、イベントカウンタではありません。有用な出力を生成するには、イベントの開始位置を指す必要があります。このオプションはコマンド行で最初に指名されたログファイルに適用されます。

このオプションはポイントインタイムリカバリに便利です。セクション 7.5 「Point-in-Time (増分) リカバリ」を参照してください。

- `--stop-datetime=datetime`

`datetime` 引数と同じかそれより遅いタイムスタンプを持つ最初のイベントで、バイナリログの読み取りを終了します。`datetime` 値の詳細については、`--start-datetime` オプションの説明を参照してください。

このオプションはポイントインタイムリカバリに便利です。セクション 7.5 「Point-in-Time (増分) リカバリ」を参照してください。

- `--stop-never`

このオプションは `--read-from-remote-server` とともに使用されます。 `mysqlbinlog` に対して、サーバーに接続したままの状態を保つことを指示します。 そうしないと、 `mysqlbinlog` は最後のログファイルがサーバーから転送された時点で終了します。 `--stop-never` は暗黙的に `--to-last-log` を指定するため、コマンド行で指名する必要があるのは転送される最初のログファイルのみです。

`--stop-never` は一般的に、ライブバイナリログバックアップを作成するために `--raw` とともに使用されますが、 `--raw` なしで使用して、サーバーがログイベントを生成するに従ってそれらを継続的にテキスト表示することも可能です。

`--stop-never` では、デフォルトで、 `mysqlbinlog` はサーバーへの接続時にサーバー ID 1 を報告します。 `--connection-server-id` を使用して、レポートする代替 ID を明示的に指定します。 これは、レプリカサーバーまたは別の `mysqlbinlog` プロセスの ID との競合を回避するために使用できます。 [セクション4.6.8.4「mysqlbinlog サーバー ID の指定」](#) を参照してください。

- `--stop-never-slave-server-id=id`

このオプションは非推奨です。 将来のリリースで削除される予定です。 かわりに `--connection-server-id` オプションを使用して、レポートする `mysqlbinlog` のサーバー ID を指定します。

- `--stop-position=N`

ログ位置 `N` 以降で開始するイベントを出力から除外して、ログ位置 `N` でバイナリログのデコードを停止します。 位置はログファイル内のバイトポイントであり、イベントカウンタではありません。 出力に含める最後のイベントの開始位置の後の位置を指す必要があります。 位置 `N` の前に開始し、位置以降に終了するイベントが、最後に処理されるイベントです。 このオプションは、コマンド行で最後に指名されたログファイルに適用されます。

このオプションはポイントインタイムリカバリに便利です。 [セクション7.5「Point-in-Time \(増分\) リカバリ」](#) を参照してください。

- `--tls-ciphersuites=ciphersuite_list`

TLSv1.3 を使用する暗号化された接続に許可される暗号スイート。 値は、コロンで区切られた 1 つ以上の暗号スイート名のリストです。 このオプションに指定できる暗号スイートは、MySQL のコンパイルに使用される SSL ライブラリによって異なります。 詳細は、 [セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#) を参照してください。

このオプションは MySQL 8.0.16 で追加されました。

- `--tls-version=protocol_list`

暗号化された接続に許可される TLS プロトコル。 値は、1 つまたは複数のコンマ区切りプロトコル名のリストです。 このオプションに指定できるプロトコルは、MySQL のコンパイルに使用される SSL ライブラリによって異なります。 詳細は、 [セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#) を参照してください。

- `--to-last-log, -t`

MySQL サーバーから要求されたバイナリログの最後で終了せず、最後のバイナリログの最後まで続けて出力します。 同じ MySQL サーバーに出力を送信した場合、無限ループになる場合があります。 このオプションには `--read-from-remote-server` が必要です。

- `--user=user_name, -u user_name`

リモートサーバーへの接続時に使用する MySQL アカウントのユーザー名。

- `--verbose, -v`

行イベントを再構築し、該当する場合はテーブルパーティション情報とともにコメント付き SQL ステートメントとして表示します。 このオプションを (`"-vv"` または `"--verbose --verbose"` のいずれかを渡して) 2 回指定すると、出

カには、カラムのデータ型と一部のメタデータを示すコメント、および `binlog_rows_query_log_events` システム変数が `TRUE` に設定されている場合の行クエリーログイベントなどの情報ログイベントが含まれます。

`--base64-output` および `--verbose` の行イベント出力への影響を示す例は、[セクション4.6.8.2「mysqlbinlog 行イベントの表示」](#)を参照してください。

- `--verify-binlog-checksum, -c`

バイナリログファイルのチェックサムを検証します。

- `--version, -V`

バージョン情報を表示して終了します。

このオプションを使用するときに表示される `mysqlbinlog` のバージョン番号は、3.4 です。

- `--zstd-compression-level=level`

`zstd` 圧縮アルゴリズムを使用するサーバーへの接続に使用する圧縮レベル。許可されるレベルは 1 から 22 で、大きい値は圧縮レベルの増加を示します。デフォルトの `zstd` 圧縮レベルは 3 です。圧縮レベルの設定は、`zstd` 圧縮を使用しない接続には影響しません。

詳細は、[セクション4.2.8「接続圧縮制御」](#)を参照してください。

このオプションは MySQL 8.0.18 で追加されました。

`mysqlbinlog` の出力を `mysql` クライアントにパイプして、バイナリログに含まれるイベントを実行できます。この方法は、古いバックアップがある場合に予期しない終了からリカバリするために使用されます ([セクション7.5「Point-in-Time \(増分\) リカバリ」](#)を参照)。例:

```
shell> mysqlbinlog binlog.000001 | mysql -u root -p
```

または:

```
shell> mysqlbinlog binlog.[0-9]* | mysql -u root -p
```

`mysqlbinlog` が生成したステートメントに `BLOB` 値が含まれる可能性がある場合、`mysql` がそれらを処理するときに問題が生じることがあります。この場合は、`mysql` を `--binary-mode` オプションで起動します。

ステートメントログをまず変更する必要がある場合 (たとえば、何らかの理由で実行しないステートメントを削除するなど) は、代わりに `mysqlbinlog` の出力をテキストファイルにリダイレクトすることもできます。ファイルの編集後、`mysql` プログラムへの入力として使用することによって、そこに含まれるステートメントを実行します。

```
shell> mysqlbinlog binlog.000001 > tmpfile
shell> ... edit tmpfile ...
shell> mysql -u root -p < tmpfile
```

`mysqlbinlog` は、`--start-position` オプションで起動された場合、バイナリログ内のオフセットが指定された位置以上のイベントのみを表示します (指定された位置は 1 つのイベントの開始位置に一致していなければなりません)。指定された日付と時間を持つイベントを検出したときに停止および起動するオプションもあります。これにより、`--stop-datetime` オプションを使用してポイントインタイムリカバリが実行できます (たとえば、「データベースを今日の 10:30 am 現在の状態までロールバックする」といったことが可能になります)。

複数のファイルの処理. MySQL サーバーに実行する複数のバイナリログがある場合、安全な方法は、サーバーへの 1 つの接続を使用して、それらすべてを処理することです。これは、安全でない可能性があることを示す例です。

```
shell> mysqlbinlog binlog.000001 | mysql -u root -p # DANGER!!
shell> mysqlbinlog binlog.000002 | mysql -u root -p # DANGER!!
```

サーバーに対してこのように複数の接続を使用してバイナリログを処理する場合、最初のログファイルに `CREATE TEMPORARY TABLE` ステートメントが含まれており、2 番目のログには一時テーブルを使用するステートメントが含まれていると、問題が発生します。最初の `mysql` プロセスが終了すると、サーバーは一時テーブルを削除します。2 番目の `mysql` プロセスでテーブルの使用を試みると、サーバーは「不明なテーブル」と報告します。

このような問題を回避するには、1 つの `mysql` プロセスを使用して、処理するすべてのバイナリログの内容を実行します。これはそれを実行する 1 つの方法です。

```
shell> mysqlbinlog binlog.000001 binlog.000002 | mysql -u root -p
```

もう 1 つのアプローチは、すべてのログを 1 つのファイルに書き込み、次にそのファイルを処理することです。

```
shell> mysqlbinlog binlog.000001 > /tmp/statements.sql
shell> mysqlbinlog binlog.000002 >> /tmp/statements.sql
shell> mysql -u root -p -e "source /tmp/statements.sql"
```

MySQL 8.0.12 から、シェルパイプを使用して、ストリーム入力として複数のバイナリログファイルを `mysqlbinlog` に提供することもできます。圧縮バイナリログファイルのアーカイブは、解凍して `mysqlbinlog` に直接提供できます。この例では、`binlog-files_1.gz` に処理用の複数のバイナリログファイルが含まれています。パイプラインは、`binlog-files_1.gz` の内容を抽出し、バイナリログファイルを標準入力として `mysqlbinlog` にパイプし、`mysqlbinlog` の出力を `mysql` クライアントにパイプして実行します。

```
shell> gzip -cd binlog-files_1.gz | ./mysqlbinlog - | ./mysql -uroot -p
```

複数のアーカイブファイルを指定できます。次に例を示します：

```
shell> gzip -cd binlog-files_1.gz binlog-files_2.gz | ./mysqlbinlog - | ./mysql -uroot -p
```

ストリーム入力の場合、`mysqlbinlog` はこのオプションを適用する最後のログファイルを識別できないため、`--stop-position` を使用しないでください。

LOAD DATA 操作。 `mysqlbinlog` では、元のデータファイルを使用せずに `LOAD DATA` 操作を再現する出力を生成できます。`mysqlbinlog` はデータを一時ファイルにコピーし、そのファイルを参照する `LOAD DATA LOCAL` ステートメントを書き込みます。これらのファイルが書き出されるディレクトリのデフォルトの場所はシステムごとに異なります。ディレクトリを明示的に指定するには、`--local-load` オプションを使用します。

`mysqlbinlog` は `LOAD DATA` ステートメントを `LOAD DATA LOCAL` ステートメントに変換する (つまり、`LOCAL` を追加する) ため、ステートメントの処理に使用するクライアントとサーバーの両方で `LOCAL` 機能を有効にして構成する必要があります。セクション6.1.6「`LOAD DATA LOCAL` のセキュリティ上の考慮事項」を参照してください。

警告

`LOAD DATA LOCAL` ステートメント用に作成された一時ファイルは、これらのステートメントをユーザーが実際に実行するまで必要なため、自動的に削除されません。ステートメントログが必要なくなった時点で、ユーザーが一時ファイルを削除するようにしてください。これらのファイルは一時ファイルディレクトリに存在し、`original_file_name-##` といった名前が付いています。

4.6.8.1 mysqlbinlog 16 進ダンプ形式

`--hexdump` オプションを使用すると、`mysqlbinlog` はバイナリログの内容の 16 進ダンプを生成するようになります。

```
shell> mysqlbinlog --hexdump source-bin.000001
```

16 進出力は、`#` で始まるコメント行で構成され、前のコマンドの出力は次のようになります。

```
/*!40019 SET @@SESSION.max_insert_delayed_threads=0*/;
/*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
# at 4
#051024 17:24:13 server id 1 end_log_pos 98
# Position Timestamp Type Master ID Size Master Pos Flags
# 00000004 9d fc 5c 43 0f 01 00 00 00 5e 00 00 00 62 00 00 00 00 00
# 00000017 04 00 35 2e 30 2e 31 35 2d 64 65 62 75 67 2d 6c |.5.0.15.debug.|]
# 00000027 6f 67 00 00 00 00 00 00 00 00 00 00 00 00 00 |log.....|]
# 00000037 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|]
# 00000047 00 00 00 00 9d fc 5c 43 13 38 0d 00 08 00 12 00 |.....C.8.....|]
# 00000057 04 04 04 04 12 00 00 4b 00 04 1a |.....K...|]
# Start: binlog v 4, server v 5.0.15-debug-log created 051024 17:24:13
# at startup
ROLLBACK;
```

現在、16 進出力には次のリストの要素が含まれます。この形式は変更される場合があります。バイナリログの形式の詳細は、「[MySQL Internals: The Binary Log](#)」を参照してください。

- **Position:** ログファイル内のバイト位置。
- **Timestamp:** イベントのタイムスタンプ。示した例では、`'9d fc 5c 43'` は `'051024 17:24:13'` の 16 進表現です。

- **Type:** イベントタイプコード。
- **Master ID:** イベントを作成したレプリケーションソースサーバーのサーバー ID。
- **Size:** イベントのサイズをバイトで表しています。
- **Master Pos:** 元のソースバイナリログファイル内の次のイベントの位置。
- **Flags:** イベントフラグ値。

4.6.8.2 mysqlbinlog 行イベントの表示

次の例は、データの変更を指定する行イベントを `mysqlbinlog` が表示する方法を説明しています。これらは `WRITE_ROWS_EVENT`、`UPDATE_ROWS_EVENT`、および `DELETE_ROWS_EVENT` タイプコードを持つイベントに対応します。 `--base64-output=DECODE-ROWS` オプションおよび `--verbose` オプションを、行イベント出力に影響を与えるために使用できます。

サーバーが行ベースのバイナリロギングを使用しており、次のステートメントのシーケンスを実行するとします。

```
CREATE TABLE t
(
  id INT NOT NULL,
  name VARCHAR(20) NOT NULL,
  date DATE NULL
) ENGINE = InnoDB;

START TRANSACTION;
INSERT INTO t VALUES(1, 'apple', NULL);
UPDATE t SET name = 'pear', date = '2009-01-01' WHERE id = 1;
DELETE FROM t WHERE id = 1;
COMMIT;
```

デフォルトでは、`mysqlbinlog` は行イベントを、`BINLOG` ステートメントを使用して base-64 文字列としてエンコードして表示します。無関係な行を省略すると、前のステートメントシーケンスによって生成される行イベントの出力は次のようになります。

```
shell> mysqlbinlog log_file
...
# at 218
#080828 15:03:08 server id 1  end_log_pos 258  Write_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAAALAAAANoAAAAABEAAAAAAAAABHRic3QAAXQAAwMPCgIUAAQ=
fAS3SBcBAAAAKAAAAIBAAAQABEAAAAAAAAEAA//8AQAAAAVhcHBsZQ==
/*!*';
...
# at 302
#080828 15:03:08 server id 1  end_log_pos 356  Update_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAAALAAAAC4BAAAAABEAAAAAAAAABHRic3QAAXQAAwMPCgIUAAQ=
fAS3SBgBAAAAngAAAGQBAQAABEAAAAAAAAEAA////AEAAAAFYXBwbGX4AQAAARwZWFyYlI=
/*!*';
...
# at 400
#080828 15:03:08 server id 1  end_log_pos 442  Delete_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAAALAAAABAAAAABEAAAAAAAAABHRic3QAAXQAAwMPCgIUAAQ=
fAS3SBkBAAAAkgAAALoBAAAQABEAAAAAAAAEAA//4AQAAARwZWFyYlI=
/*!*';
```

行イベントを「擬似 SQL」ステートメントの形式のコメントして表示するには、`mysqlbinlog` を `--verbose` オプションまたは `-v` オプションで実行します。この出力レベルには、テーブルパーティション情報も表示されます (該当する場合)。出力には、`###` で始まる行が含まれます:

```
shell> mysqlbinlog -v log_file
...
# at 218
#080828 15:03:08 server id 1  end_log_pos 258  Write_rows: table id 17 flags: STMT_END_F
```



```

BINLOG '
fAS3SBMBAAAAALAAAANoAAAAABEAAAAAAAAABHRic3QAAXQAAwMPCgIUAAQ=
fAS3SBcBAAAAKAAAAAIBAAQABEAAAAAAAAEAA//8AQAAAVhchBsZQ==
/*!*/;
### INSERT INTO test.t
### SET
### @1=1
### @2='apple'
### @3=NULL
...
# at 302
#080828 15:03:08 server id 1 end_log_pos 356 Update_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAAALAAAC4BAAAAABEAAAAAAAAABHRic3QAAXQAAwMPCgIUAAQ=
fAS3SBgBAAAAANgAAAGQBAAQABEAAAAAAAAEAA///AEAAAAFYXBwbGX4AQAAAAARwZWFyIblIP
/*!*/;
### UPDATE test.t
### WHERE
### @1=1
### @2='apple'
### @3=NULL
### SET
### @1=1
### @2='pear'
### @3='2009:01:01'
...
# at 400
#080828 15:03:08 server id 1 end_log_pos 442 Delete_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAAALAAAABAAAAABEAAAAAAAAABHRic3QAAXQAAwMPCgIUAAQ=
fAS3SBkBAAAAKGAALoBAAQABEAAAAAAAAEAA//4AQAAAAARwZWFyIblIP
/*!*/;
### DELETE FROM test.t
### WHERE
### @1=1
### @2='pear'
### @3='2009:01:01'

```

--verbose または -v を 2 回指定すると、各カラムのデータ型と一部のメタデータ、および [binlog_rows_query_log_events](#) システム変数が TRUE に設定されている場合は行クエリーログイベントなどの情報ログイベントも表示されます。出力には、各カラムの変更後に追加のコメントが含まれます:

```

shell> mysqlbinlog -vv log_file
...
# at 218
#080828 15:03:08 server id 1 end_log_pos 258 Write_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAAALAAAANoAAAAABEAAAAAAAAABHRic3QAAXQAAwMPCgIUAAQ=
fAS3SBcBAAAAKAAAAAIBAAQABEAAAAAAAAEAA//8AQAAAVhchBsZQ==
/*!*/;
### INSERT INTO test.t
### SET
### @1=1 /* INT meta=0 nullable=0 is_null=0 */
### @2='apple' /* VARSTRING(20) meta=20 nullable=0 is_null=0 */
### @3=NULL /* VARSTRING(20) meta=0 nullable=1 is_null=1 */
...
# at 302
#080828 15:03:08 server id 1 end_log_pos 356 Update_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAAALAAAC4BAAAAABEAAAAAAAAABHRic3QAAXQAAwMPCgIUAAQ=
fAS3SBgBAAAAANgAAAGQBAAQABEAAAAAAAAEAA///AEAAAAFYXBwbGX4AQAAAAARwZWFyIblIP
/*!*/;
### UPDATE test.t
### WHERE
### @1=1 /* INT meta=0 nullable=0 is_null=0 */
### @2='apple' /* VARSTRING(20) meta=20 nullable=0 is_null=0 */
### @3=NULL /* VARSTRING(20) meta=0 nullable=1 is_null=1 */
### SET
### @1=1 /* INT meta=0 nullable=0 is_null=0 */
### @2='pear' /* VARSTRING(20) meta=20 nullable=0 is_null=0 */

```

```
### @3='2009:01:01' /* DATE meta=0 nullable=1 is_null=0 */
...
# at 400
#080828 15:03:08 server id 1 end_log_pos 442 Delete_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAAALAAAAJABAAAAABEAAAAAAAAABHRic3QAAXQAAMPCgIUAAQ=
fAS3SBkBAAAAKgAAALoBAAAQABEAAAAAAAAEAA/4AQAAAAArwZWFyIbIP
/*!*/;
### DELETE FROM test.t
### WHERE
### @1=1 /* INT meta=0 nullable=0 is_null=0 */
### @2='pear' /* VARSTRING(20) meta=20 nullable=0 is_null=0 */
### @3='2009:01:01' /* DATE meta=0 nullable=1 is_null=0 */
```

`--base64-output=DECODE-ROWS` オプションを使用して、行イベントに対する `BINLOG` ステートメントを抑制するように `mysqlbinlog` に指示できます。これは `--base64-output=NEVER` と同様ですが、行イベントが検出された場合にエラーで終了しません。 `--base64-output=DECODE-ROWS` および `--verbose` の組み合わせにより、行イベントを SQL ステートメントとしてのみ表示する便利な方法が提供されます。

```
shell> mysqlbinlog -v --base64-output=DECODE-ROWS log_file
...
# at 218
#080828 15:03:08 server id 1 end_log_pos 258 Write_rows: table id 17 flags: STMT_END_F
### INSERT INTO test.t
### SET
### @1=1
### @2='apple'
### @3=NULL
...
# at 302
#080828 15:03:08 server id 1 end_log_pos 356 Update_rows: table id 17 flags: STMT_END_F
### UPDATE test.t
### WHERE
### @1=1
### @2='apple'
### @3=NULL
### SET
### @1=1
### @2='pear'
### @3='2009:01:01'
...
# at 400
#080828 15:03:08 server id 1 end_log_pos 442 Delete_rows: table id 17 flags: STMT_END_F
### DELETE FROM test.t
### WHERE
### @1=1
### @2='pear'
### @3='2009:01:01'
```

注記

`mysqlbinlog` の出力を再実行する予定である場合は、`BINLOG` ステートメントを抑制するべきではありません。

`--verbose` が行イベントに関して生成した SQL ステートメントは、対応する `BINLOG` ステートメントよりもはるかに読みやすくなります。ただし、イベントを生成した元の SQL ステートメントと正確には対応しません。次の制約が適用されます。

- 元のカラム名は失われて `@N` に置換されます。 `N` はカラム番号です。
- バイナリログでは文字セット情報は利用できません。これは文字列カラムの表示に影響します。
- 対応するバイナリ文字列型と非バイナリ文字列型の間に区別はありません (`BINARY` と `CHAR`、`VARBINARY` と `VARCHAR`、`BLOB` と `TEXT`)。出力では固定長文字列に対して `STRING`、可変長文字列に対して `VARSTRING` データ型が使用されます。
- マルチバイト文字セットでは、文字当たりの最大のバイト数はバイナリログには現れないため、文字列型の長さは文字数ではなくバイト数で表示されます。たとえば、`STRING(4)` は、次のいずれかのカラムタイプの値のデータ型として使用されます:

```
CHAR(4) CHARACTER SET latin1
CHAR(2) CHARACTER SET ucs2
```

- `UPDATE_ROWS_EVENT` 型のイベントのストレージフォーマットのため、`UPDATE` ステートメントは `WHERE` 句が `SET` 句の前に表示されます。

行イベントを適切に解釈するには、バイナリログの最初にある形式の記述の情報が必要です。mysqlbinlog はログの残りの部分に行イベントが含まれるかどうかは前もってわからないため、デフォルトでは出力の最初の部分に `BINLOG` ステートメントを使用して形式記述イベントを表示します。

バイナリログに `BINLOG` ステートメントを必要とするイベントが含まれないことがわかっている (つまり行イベントがない) 場合は、`--base64-output=NEVER` オプションを使用してこのヘッダーが書き込まれるのを回避できます。

4.6.8.3 バイナリログファイルのバックアップのための mysqlbinlog の使用

デフォルトでは、mysqlbinlog はバイナリログファイルを読み取り、その内容をテキスト形式で表示します。これにより、ファイル内のイベントが調べやすくなり、(たとえば出力を mysql への入力として使用して) それらを再実行できます。mysqlbinlog はローカルファイルシステムから直接ログファイルを読み取るか、または `--read-from-remote-server` オプションで、サーバーに接続してそのサーバーからバイナリログの内容を要求できます。mysqlbinlog はテキスト出力を標準出力、または `--result-file=file_name` オプションが指定された場合はその値で指名されるファイルに書き出すことができます。

- mysqlbinlog バックアップ機能
- mysqlbinlog のバックアップオプション
- 静的バックアップとライブバックアップ
- 出力ファイルの命名
- 例: mysqldump と mysqlbinlog を合わせてバックアップとリストアを行う
- mysqlbinlog バックアップの制限事項

mysqlbinlog バックアップ機能

mysqlbinlog では、バイナリログファイルを読み取り、同じコンテンツを含む新しいファイル (テキスト形式ではなくバイナリ形式) を書き込むことができます。この機能により、バイナリログを容易に元の形式でバックアップできます。mysqlbinlog は、ログファイルのセットのバックアップを実行して最後のファイルの最後に到達したときに停止して、静的バックアップを作成できます。また、最後のログファイルの最後に到達したときにサーバーとの接続を保ち、新しいイベントが生成されるたびにコピーを継続して、連続的な (「ライブ」) バックアップを作成することもできます。連続的なバックアップ操作では、mysqlbinlog は接続が終了するまで (たとえばサーバーが終了するときなど)、または mysqlbinlog が強制的に終了させられるまで実行されます。接続が終了すると、mysqlbinlog はレプリカサーバーとは異なり、接続を待機して再試行しません。サーバーの再起動後にライブバックアップを続行するには、mysqlbinlog も再起動する必要があります。

重要

mysqlbinlog では、暗号化されたバイナリログファイルと暗号化されていないバイナリログファイルの両方をバックアップできます。ただし、mysqlbinlog を使用して生成された暗号化バイナリログファイルのコピーは、暗号化されていない形式で格納されます。

mysqlbinlog のバックアップオプション

バイナリログバックアップには、mysqlbinlog を少なくとも 2 つのオプションを使用して起動する必要があります。

- `--read-from-remote-server` (または `-R`) オプションは mysqlbinlog に、サーバーに接続してバイナリログを要求するよう指示します。(これは、レプリケーションソースサーバーに接続するレプリカサーバーに似ています。)
- `--raw` オプションは mysqlbinlog に、テキスト出力ではなく生の (バイナリ) 出力を書き出すように指示します。

`--read-from-remote-server` オプションに加えて、次のオプションを指定するのが一般的です。`--host` はサーバーが稼働している場所を示し、`--user` および `--password` などの接続オプションも必要な場合があります。

`--raw` とともに使用すると便利なオプションがほかにいくつかあります。

- `--stop-never`: 最後のログファイルの最後に到達したあと、サーバーとの接続を維持して新しいイベントの読み取りを続行します。
- `--connection-server-id=id`: `mysqlbinlog` がサーバーへの接続時にレポートするサーバー ID。 `--stop-never` を使用する場合、レポートされるデフォルトのサーバー ID は 1 です。これにより、レプリカサーバーまたは別の `mysqlbinlog` プロセスの ID と競合する場合は、 `--connection-server-id` を使用して別のサーバー ID を指定します。 [セクション 4.6.8.4 「mysqlbinlog サーバー ID の指定」](#) を参照してください。
- `--result-file`: あとで説明するように、出力ファイル名のプリフィクス。

静的バックアップとライブバックアップ

`mysqlbinlog` でサーバーのバイナリログファイルのバックアップを行うには、サーバーに実際に存在するファイル名を指定する必要があります。名前がわからない場合は、サーバーに接続して `SHOW BINARY LOGS` ステートメントを使用して現在の名前を表示します。このステートメントによって次の出力が生成されるとします。

```
mysql> SHOW BINARY LOGS;
+-----+-----+
| Log_name | File_size | Encrypted |
+-----+-----+
| binlog.000130 | 27459 | No |
| binlog.000131 | 13719 | No |
| binlog.000132 | 43268 | No |
+-----+-----+
```

この情報により、`mysqlbinlog` を使用して次のようにバイナリログを現在のディレクトリにバックアップできます (コマンドは 1 行に 1 つずつ入力します)。

- `binlog.000130` から `binlog.000132` までの静的バックアップを行うには、次のいずれかのコマンドを使用します。

```
mysqlbinlog --read-from-remote-server --host=host_name --raw
binlog.000130 binlog.000131 binlog.000132

mysqlbinlog --read-from-remote-server --host=host_name --raw
--to-last-log binlog.000130
```

最初のコマンドはすべてのファイル名を明示的に指定します。2 番目は最初のファイルのみを指名し、`--to-last-log` を使用して最後まで読み取ります。これらのコマンドの違いは、`mysqlbinlog` が `binlog.000132` の最後に到達する前にサーバーが `binlog.000133` を開く場合、最初のコマンドはそれを読み取りませんが、2 番目のコマンドは読み取ります。

- `mysqlbinlog` が `binlog.000130` から既存のログファイルのコピーを開始し、その後接続を維持してサーバーが新しいイベントを生成するにつれてそれらをコピーするライブバックアップを行うには:

```
mysqlbinlog --read-from-remote-server --host=host_name --raw
--stop-never binlog.000130
```

`--stop-never` を使用すると、`--to-last-log` オプションは暗示されているため、最後のログファイルまで読み取るようにこのオプションを指定する必要はありません。

出力ファイルの命名

`--raw` を使用しないと、`mysqlbinlog` はテキスト出力を生成し、`--result-file` オプションが与えられた場合は、すべての出力が書き出される単一のファイルの名前を指定します。`--raw` を使用すると、`mysqlbinlog` はサーバーから転送される各ログファイルに対して 1 つのバイナリ出力ファイルを書き出します。デフォルトでは、`mysqlbinlog` はファイルを現在のディレクトリに元のログファイルと同じ名前で書き出します。出力ファイル名を変更するには、`--result-file` オプションを使用します。`--raw` も指定されている場合、`--result-file` オプション値は出力ファイル名を変更するプリフィクスとして処理されます。

サーバーに現在 `binlog.000999` 以上の名前のバイナリログファイルがあるとします。`mysqlbinlog --raw` を使用してファイルのバックアップを行う場合、`--result-file` オプションは次の表に示すように出力ファイル名を生成します。`--result-file` の値をディレクトリパスで始まるようにすることで、ファイルを特定のディレクトリに書き出すことができます。`--result-file` の値がディレクトリ名のみで構成されている場合は、その値はパス名区切り文字で終わっていません。出力ファイルが存在する場合は上書きされます。

<code>--result-file</code> オプション	出力ファイル名
<code>--result-file=x</code>	<code>xbinlog.000999</code> 以上
<code>--result-file=/tmp/</code>	<code>/tmp/binlog.000999</code> 以上
<code>--result-file=/tmp/x</code>	<code>/tmp/xbinlog.000999</code> 以上

例: mysqldump と mysqlbinlog を合わせてバックアップとリストアを行う

次の例は、`mysqldump` と `mysqlbinlog` を一緒に使用してサーバーのデータおよびバイナリログのバックアップを取る方法、およびデータの損失が生じた場合にバックアップを使用してサーバーのリストアを行う方法を示す簡単なシナリオを説明しています。例では、サーバーはホスト `host_name` 上で稼働し、最初のバイナリログファイルの名前は `binlog.000999` であるとしています。コマンドは 1 行に 1 つずつ入力します。

`mysqlbinlog` を使用してバイナリログのバックアップを継続的に作成します。

```
mysqlbinlog --read-from-remote-server --host=host_name --raw
--stop-never binlog.000999
```

`mysqldump` を使用してサーバーのデータのスナップショットとしてダンプファイルを作成します。`--all-databases`、`--events`、および `--routines` を使用してすべてのデータのバックアップを行い、`--master-data=2` を使用して現在のバイナリログ座標をダンプファイルに含めます。

```
mysqldump --host=host_name --all-databases --events --routines --master-data=2 > dump_file
```

必要に応じて `mysqldump` コマンドを定期的に行って、新しいスナップショットを作成します。

データ損失が発生した場合（たとえば、サーバーが予期せず終了した場合）、最新のダンプファイルを使用してデータをリストアします：

```
mysql --host=host_name -u root -p < dump_file
```

次にバイナリログバックアップを使用して、ダンプファイル内にリストされている座標よりあとで書き出されたイベントを再実行します。ファイル内の座標が次のようになっています。

```
-- CHANGE MASTER TO MASTER_LOG_FILE='binlog.001002', MASTER_LOG_POS=27284;
```

直前にバックアップされたログファイルが `binlog.001004` という名前の場合、次のようにロギングイベントを再実行します。

```
mysqlbinlog --start-position=27284 binlog.001002 binlog.001003 binlog.001004
| mysql --host=host_name -u root -p
```

リストア操作をより容易に実行するために、または MySQL でリモート `root` アクセスが許可されない場合は、バックアップファイル（ダンプファイルおよびバイナリログファイル）をサーバーホストにコピーする方が簡単な場合もあります。

mysqlbinlog バックアップの制限事項

`mysqlbinlog` でのバイナリログバックアップには、次の制限事項があります：

- 接続が失われた場合（サーバーの再起動が発生した場合やネットワークが停止した場合など）、`mysqlbinlog` は MySQL サーバーに自動的に再接続しません。
- バックアップの遅延は、レプリカサーバーの遅延と似ています。

4.6.8.4 mysqlbinlog サーバー ID の指定

`mysqlbinlog` は、`--read-from-remote-server` オプションで呼び出された場合、MySQL サーバーに接続し、自分を証明するためにサーバー ID を指定し、サーバーからバイナリログファイルを要求します。`mysqlbinlog` を使用してサーバーからログファイルを要求するには、いくつかの方法があります。

- 一連のファイルを名前を明示的に示して指定します。`mysqlbinlog` は、各ファイルに接続して `Binlog dump` コマンドを発行します。サーバーはファイルを送信して切断します。ファイルごとに 1 つの接続があります。

- 開始ファイルと `--to-last-log` を指定します。 `mysqlbinlog` はすべてのファイルに接続して `Binlog dump` コマンドを発行します。サーバーはすべてのファイルを送信して切断します。
- 開始ファイルと `--stop-never` を指定します (これは暗黙的に `--to-last-log` を示します)。 `mysqlbinlog` はすべてのファイルに対して接続して `Binlog dump` コマンドを発行します。サーバーはすべてのファイルを送信しますが、最後のファイルの送信後に切断しません。

`--read-from-remote-server` のみを使用すると、 `mysqlbinlog` は 0 をサーバー ID として使用します。これは、要求された最後のログファイルの送信後に切断することをサーバーに指示します。

`--read-from-remote-server` および `--stop-never` を使用すると、 `mysqlbinlog` はゼロ以外のサーバー ID を使用して接続するため、サーバーは最後のログファイルの送信後に切断しません。サーバー ID はデフォルトで 1 ですが、これは `--connection-server-id` で変更できます。

したがって、ファイルを要求する最初の 2 つの方法では、 `mysqlbinlog` がサーバー ID の 0 を指定するためサーバーは切断します。 `--stop-never` が指定された場合は、 `mysqlbinlog` はゼロ以外のサーバー ID を指定するため切断しません。

4.6.9 mysqldumpslow — スロークエリーログファイルの要約

MySQL スロークエリーログには、実行に時間がかかるクエリーに関する情報が含まれています (セクション 5.4.5 「スロークエリーログ」を参照)。 `mysqldumpslow` は、MySQL スロークエリーログファイルを解析し、その内容を要約します。

通常、 `mysqldumpslow` は数字の特定の値および文字列データ値以外が同様のクエリーをグループ化します。サマリーの出力を表示する際、これらの値を N および 'S' に「抽象化」します。値の抽象化動作を変更するには、 `-a` および `-n` オプションを使用します。

`mysqldumpslow` は次のように起動します。

```
shell> mysqldumpslow [options] [log_file ...]
```

使用例:

```
shell> mysqldumpslow
```

```
Reading mysql slow query log from /usr/local/mysql/data/mysqld80-slow.log
Count: 1 Time=4.32s (4s) Lock=0.00s (0s) Rows=0.0 (0), root[root]@localhost
insert into t2 select * from t1
```

```
Count: 3 Time=2.53s (7s) Lock=0.00s (0s) Rows=0.0 (0), root[root]@localhost
insert into t2 select * from t1 limit N
```

```
Count: 3 Time=2.13s (6s) Lock=0.00s (0s) Rows=0.0 (0), root[root]@localhost
insert into t1 select * from t1
```

`mysqldumpslow` は次のオプションをサポートします。

表 4.22 「mysqldumpslow のオプション」

オプション名	説明
<code>-a</code>	すべての数字を N に、文字列を 'S' に抽象化しません
<code>-n</code>	少なくとも指定された桁数の数字を抽象化
<code>--debug</code>	デバッグ情報を書き込み
<code>-g</code>	パターンに一致するステートメントのみを考慮
<code>--help</code>	ヘルプメッセージを表示して終了
<code>-h</code>	ログファイル名内のサーバーのホスト名
<code>-i</code>	サーバーインスタンスの名前
<code>-l</code>	合計時間からロック時間を減算しない
<code>-r</code>	ソート順序を逆転
<code>-s</code>	出力のソート方法

オプション名	説明
<code>-t</code>	最初から指定された数だけのクエリーのみ表示
<code>--verbose</code>	冗長モード

- `--help`

ヘルプメッセージを表示して終了します。

- `-a`

すべての数字を `N` に、文字列を 'S' に抽象化しません。

- `--debug, -d`

デバッグモードで実行します。

このオプションは、MySQL が `WITH_DEBUG` を使用して構築された場合にのみ使用できます。Oracle によって提供される MySQL リリースバイナリは、このオプションを使用して構築されません。

- `-g pattern`

(`grep` 形式の) パターンに一致するクエリーのみを考慮します。

- `-h host_name`

*`-slow.log` ファイル名の MySQL サーバーのホスト名。値にはワイルドカードを含めることができます。デフォルトは * (すべて一致) です。

- `-i name`

サーバーインスタンス名 (`mysql.server` 起動スクリプトを使用している場合)。

- `-l`

合計時間からロック時間を減算しません。

- `-n N`

少なくとも `N` 桁の数字を名前に抽象化します。

- `-r`

ソート順序を逆転します。

- `-s sort_type`

出力のソート方法。 `sort_type` の値は次のリストから選択するようにしてください。

- `t, at`: クエリー時間または平均クエリー時間でソート
- `l, al`: ロック時間または平均ロック時間でソート
- `r, ar`: 送信行数または平均送信行数でソート
- `c`: カウントでソート

デフォルトでは、`mysqldumpslow` は平均クエリー時間でソートします (`-s at` と同等)。

- `-t N`

出力内の最初の `N` 個のクエリーのみを表示します。

- `--verbose, -v`

冗長モード。プログラムの動作についてより多くの情報を出力します。

4.7 プログラム開発ユーティリティ

このセクションでは、MySQL プログラムを開発する際に有用であると思われるいくつかのユーティリティについて説明します。

シェルスクリプトで `my_print_defaults` プログラムを使用して、オプションファイルを解析し、所定のプログラムがどのオプションを使用するかを表示できます。次の例は、`[client]` グループおよび `[mysql]` グループで検出されたオプションを表示するように指示された場合に、`my_print_defaults` が生成するであろう出力を示しています。

```
shell> my_print_defaults client mysql
--port=3306
--socket=/tmp/mysql.sock
--no-auto-rehash
```

開発者へのメモ: オプションファイルの処理は、単にコマンド行引数の前に適切なグループ内のすべてのオプションを処理することによって、C クライアントライブラリに実装されています。これは、複数回指定されたオプションの最後のインスタンスを使用するプログラムではうまく機能します。複数指定されたオプションをこの方法で処理するが、オプションファイルを読み取らない C プログラムまたは C++ プログラムがある場合は、その機能を与えるために 2 行のみを追加する必要があります。標準 MySQL クライアントの任意のソースコードをチェックして、その方法を確認します。

MySQL へのほかのいくつかの言語インタフェースは C クライアントライブラリに基づいており、そのうちのいくつかはオプションファイルの内容にアクセスする方法を提供します。これらには、Perl および Python が含まれます。詳細は、使用するインタフェースのドキュメントを参照してください。

4.7.1 mysql_config — クライアントのコンパイル用オプションの表示

`mysql_config` は、MySQL クライアントをコンパイルして MySQL に接続するのに有用な情報を提供します。シェルスクリプトであるため、Unix および Unix 類似システムでのみ使用可能です。

注記

`pkg-config` を `mysql_config` のかわりに使用して、MySQL アプリケーションのコンパイルに必要なコンパイラフラグやリンクライブラリなどの情報を取得できます。詳細は、[Building C API Client Programs Using pkg-config](#) を参照してください。

`mysql_config` は次のオプションをサポートします。

- `--cflags`

`libmysqlclient` ライブラリのコンパイルに使用される、インクルードファイルを検索するための C コンパイラフラグおよび重要なコンパイラフラグおよび定義。返されるオプションは、ライブラリが作成されたときに使用された特定のコンパイラに結びついており、ユーザー自身のコンパイラ設定ではクラッシュする場合があります。インクルードパスのみを含むより移植性の高いオプションには、`--include` を使用します。

- `--cxxflags`

`--cflags` と同様ですが、C++ コンパイラフラグ用です。

- `--include`

MySQL インクルードファイルを検出するためのコンパイラオプション。

- `--libs`

MySQL クライアントライブラリにリンクするために必要なライブラリおよびオプション。

- `--libs_r`

スレッドセーフな MySQL クライアントライブラリにリンクするために必要なライブラリおよびオプション。In MySQL 8.0 ではすべてのクライアントライブラリはスレッドセーフであるため、このオプションを使用する必要はありません。すべての場合に `--libs` オプションを使用できます。

- `--plugindir`

MySQL の構成時に定義される、デフォルトのプラグインディレクトリパス名。

- `--port`

MySQL の構成時に定義される、デフォルトの TCP/IP ポート番号。

- `--socket`

MySQL の構成時に定義される、デフォルトの Unix ソケットファイル。

- `--variable=var_name`

指定された構成変数の値を表示します。許可される `var_name` 値は、`pkgincludedir` (ヘッダーファイルディレクトリ)、`pkglibdir` (ライブラリディレクトリ) および `plugindir` (プラグインディレクトリ) です。

- `--version`

MySQL 配布のバージョン番号。

`mysql_config` をオプションなしで呼び出すと、サポートされるすべてのオプションおよびそれらの値のリストが表示されます。

```
shell> mysql_config
Usage: /usr/local/mysql/bin/mysql_config [options]
Options:
--cflags      [-I/usr/local/mysql/include/mysql -mcpu=pentiumpro]
--cxxflags    [-I/usr/local/mysql/include/mysql -mcpu=pentiumpro]
--include     [-I/usr/local/mysql/include/mysql]
--libs        [-L/usr/local/mysql/lib/mysql -lmysqlclient
              -lpthread -lm -lrt -lssl -lcrypto -ldl]
--libs_r      [-L/usr/local/mysql/lib/mysql -lmysqlclient_r
              -lpthread -lm -lrt -lssl -lcrypto -ldl]
--plugindir   [/usr/local/mysql/lib/plugin]
--socket      [/tmp/mysql.sock]
--port        [3306]
--version     [5.8.0-m17]
--variable=VAR VAR is one of:
              pkgincludedir [/usr/local/mysql/include]
              pkglibdir     [/usr/local/mysql/lib]
              plugindir     [/usr/local/mysql/lib/plugin]
```

バックティックを使用してコマンドライン内で `mysql_config` を使用し、特定のオプションに対して生成される出力を含めることができます。たとえば、MySQL クライアントプログラムのコンパイルおよびリンクを行うには、`mysql_config` を次のように使用します。

```
gcc -c `mysql_config --cflags` progname.c
gcc -o progname progname.o `mysql_config --libs`
```

4.7.2 my_print_defaults — オプションファイルからのオプションの表示

`my_print_defaults` はオプションファイルのオプショングループ内にあるオプションを表示します。出力には、指定されたオプショングループを読み取るプログラムによって使用されるオプションが示されます。たとえば、`mysqlcheck` プログラムは `[mysqlcheck]` および `[client]` のオプショングループを読み取ります。標準オプションファイル内のこれらのグループに存在するオプションを確認するには、`my_print_defaults` を次のように起動します。

```
shell> my_print_defaults mysqlcheck client
--user=myusername
--password=password
--host=localhost
```

出力には、コマンド行で指定される形式のオプションが 1 行につき 1 つ含まれます。

`my_print_defaults` は次のオプションをサポートします。

- `--help, -?`

ヘルプメッセージを表示して終了します。

- `--config-file=file_name, --defaults-file=file_name, -c file_name`

指定されたオプションファイルのみを読み取ります。

- `--debug=debug_options, -# debug_options`

デバッグのログを書き込みます。一般的な `debug_options` 文字列は `d:t:o,file_name` です。デフォルトは `d:t:o,tmp/my_print_defaults.trace` です。

- `--defaults-extra-file=file_name, --extra-file=file_name, -e file_name`

このオプションファイルは、グローバルオプションファイルのあとに読み取りますが、(UNIX では) ユーザーオプションファイルの前に読み取るようにしてください。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--defaults-group-suffix=suffix, -g suffix`

コマンド行で指名されたグループのほかに、指定されたサフィクスのあるグループを読み取ります。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--login-path=name, -l name`

`.mylogin.cnf` ログインパスファイルの指定されたログインパスからオプションを読み取ります。「ログインパス」は、接続先の MySQL サーバーおよび認証に使用するアカウントを指定するオプションを含むオプショングループです。ログインパスファイルを作成または変更するには、[mysql_config_editor](#) ユーティリティを使用します。[セクション4.6.7「mysql_config_editor — MySQL 構成ユーティリティー」](#)を参照してください。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--no-defaults, -n`

空の文字列を返します。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--show, -s`

`my_print_defaults` は、デフォルトでパスワードをマスクします。パスワードをクリアテキストで表示するには、このオプションを使用します。

- `--verbose, -v`

冗長モード。プログラムの動作についてより多くの情報を出力します。

- `--version, -V`

バージョン情報を表示して終了します。

4.8 その他のプログラム

4.8.1 lz4_decompress — mysqlpump LZ4-Compressed 出力の解凍

`lz4_decompress` ユーティリティは、LZ4 圧縮を使用して作成された `mysqlpump` 出力を解凍します。

注記

MySQL が `-DWITH_LZ4=system` オプションで構成されている場合、`lz4_decompress` は構築されません。この場合、かわりに `system lz4` コマンドを使用できます。

次のように `lz4_decompress` を起動します:

```
shell> lz4_decompress input_file output_file
```

例:

```
shell> mysqlpump --compress-output=LZ4 > dump.lz4  
shell> lz4_decompress dump.lz4 dump.txt
```

ヘルプメッセージを表示するには、引数を指定せずに `lz4_decompress` を起動します。

`mysqlpump` ZLIB 圧縮出力を解凍するには、`zlib_decompress` を使用します。 [セクション4.8.3「zlib_decompress — mysqlpump ZLIB 圧縮出力の解凍」](#) を参照してください。

4.8.2 perror — MySQL エラーメッセージ情報の表示

`perror` に、MySQL またはオペレーティングシステムのエラーコードのエラーメッセージが表示されます。 `perror` は次のように起動します。

```
shell> perror [options] errorcode ...
```

`perror` は、引数を柔軟に理解しようとします。たとえば、`ER_WRONG_VALUE_FOR_VAR` エラーの場合、`perror` はこれらの引数のいずれかを認識: `1231`、`001231`、`MY-1231`、`MY-001231` または `ER_WRONG_VALUE_FOR_VAR`。

```
shell> perror 1231  
MySQL error code MY-001231 (ER_WRONG_VALUE_FOR_VAR): Variable '%-.64s'  
can't be set to the value of '%-.200s'
```

エラー番号が、MySQL とオペレーティングシステムのエラーが重複する範囲内にある場合、`perror` では両方のエラーメッセージが表示されます:

```
shell> perror 1 13  
OS error code 1: Operation not permitted  
MySQL error code MY-000001: Can't create/write to file '%s' (OS errno %d - %s)  
OS error code 13: Permission denied  
MySQL error code MY-000013: Can't get stat of '%s' (OS errno %d - %s)
```

MySQL クラスタエラーコードのエラーメッセージを取得するには、`ndb_perror` ユーティリティを使用します。

システムエラーメッセージの意味は、オペレーティングシステムによって異なります。異なるオペレーティングシステムでは、所定のエラーコードの意味が異なる場合があります。

`perror` は次のオプションをサポートします。

- `--help`, `--info`, `-l`, `-?`

ヘルプメッセージを表示して終了します。

- `--ndb`

MySQL Cluster エラーコードのエラーメッセージを出力します。

このオプションは MySQL 8.0.13 で削除されました。かわりに `ndb_perror` ユーティリティを使用してください。

- `--silent`, `-s`

サイレントモード。エラーメッセージのみ出力します。

- `--verbose`, `-v`

冗長モード。エラーコードおよびメッセージを出力します。これはデフォルトの動作です。

- `--version`, `-V`

バージョン情報を表示して終了します。

4.8.3 zlib_decompress — mysqlpump ZLIB 圧縮出力の解凍

`zlib_decompress` コーティリティは、ZLIB 圧縮を使用して作成された `mysqlpump` 出力を解凍します。

注記

MySQL が `-DWITH_ZLIB=system` オプションで構成されている場合、`zlib_decompress` は構築されません。この場合、かわりに `system openssl zlib` コマンドを使用できます。

次のように `zlib_decompress` を起動します:

```
shell> zlib_decompress input_file output_file
```

例:

```
shell> mysqlpump --compress-output=ZLIB > dump.zlib
shell> zlib_decompress dump.zlib dump.txt
```

ヘルプメッセージを表示するには、引数を指定せずに `zlib_decompress` を起動します。

`mysqlpump` LZ4-compressed 出力を解凍するには、`lz4_decompress` を使用します。 [セクション 4.8.1 「lz4_decompress — mysqlpump LZ4-Compressed 出力の解凍」](#) を参照してください。

4.9 環境変数

このセクションでは、MySQL によって直接的または間接的に使用される環境変数を示します。これらの多くは本ドキュメントの別の場所にもあります。

コマンド行のオプションは、オプションファイルおよび環境変数で指定された値よりも優先され、オプションファイルの値は、環境変数の値よりも優先されます。多くの場合、MySQL の動作を変更するには、環境変数ではなくオプションファイルを使用する方が好ましいといえます。 [セクション 4.2.2.2 「オプションファイルの使用」](#) を参照してください。

変数	説明
<code>AUTHENTICATION_LDAP_CLIENT_LOG</code>	クライアント側の LDAP 認証ロギングレベル。
<code>AUTHENTICATION_PAM_LOG</code>	PAM 認証プラグインのデバッグロギング設定。
<code>CC</code>	C コンパイラの名前 (CMake 実行用)。
<code>CXX</code>	C++ コンパイラの名前 (CMake 実行用)。
<code>CC</code>	C コンパイラの名前 (CMake 実行用)。
<code>DBI_USER</code>	Perl DBI のデフォルトユーザー名。
<code>DBI_TRACE</code>	Perl DBI のトレースオプション。
<code>HOME</code>	<code>mysql</code> 履歴ファイルのデフォルトのパスは <code>\$HOME/.mysql_history</code> です。
<code>LD_RUN_PATH</code>	<code>libmysqlclient.so</code> のロケーション指定に使用。
<code>LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN</code>	<code>mysql_clear_password</code> 認証プラグインを有効化。 セクション 6.4.1.4 「クライアント側クリアテキストプラグイン認証」 を参照してください。
<code>LIBMYSQL_PLUGIN_DIR</code>	クライアントプラグインを検索するディレクトリ。
<code>LIBMYSQL_PLUGINS</code>	プリロードするクライアントプラグイン。
<code>MYSQL_DEBUG</code>	デバッグ中のデバッグトレースオプション。
<code>MYSQL_GROUP_SUFFIX</code>	オプショングループのサフィックスの値 (<code>--defaults-group-suffix</code> などの指定)。
<code>MYSQL_HISTFILE</code>	<code>mysql</code> 履歴ファイルへのパス。この変数を設定すると、その値は <code>\$HOME/.mysql_history</code> のデフォルトをオーバーライドします。
<code>MYSQL_HISTIGNORE</code>	<code>mysql</code> が <code>\$HOME/.mysql_history</code> または <code>syslog</code> (<code>--syslog</code> が指定されている場合) に記録しないステートメントを指定するパターン。

変数	説明
MYSQL_HOME	サーバー固有の <code>my.cnf</code> ファイルが存在するディレクトリへのパス。
MYSQL_HOST	<code>mysql</code> コマンド行クライアントが使用するデフォルトのホスト名。
MYSQL_OPENSSL_UDF_DH_BITS_THRESHOLD	<code>create_dh_parameters()</code> のキーの最大長。 セクション 6.6.2 「MySQL Enterprise Encryption の使用方法と例」 を参照してください。
MYSQL_OPENSSL_UDF_DSA_BITS_THRESHOLD	<code>create_asymmetric_priv_key()</code> の DSA キーの最大長。 セクション 6.6.2 「MySQL Enterprise Encryption の使用方法と例」 を参照してください。
MYSQL_OPENSSL_UDF_RSA_BITS_THRESHOLD	<code>create_asymmetric_priv_key()</code> の RSA キーの最大長。 セクション 6.6.2 「MySQL Enterprise Encryption の使用方法と例」 を参照してください。
MYSQL_PS1	<code>mysql</code> コマンド行クライアントで使用するコマンドプロンプト。
MYSQL_PWD	<code>mysqld</code> に接続する際のデフォルトのパスワード。この使用はセキュアではありません。表のあとにある注釈を参照してください。
MYSQL_TCP_PORT	デフォルトの TCP/IP ポート番号。
MYSQL_TEST_LOGIN_FILE	<code>.mylogin.cnf</code> ログインパスファイルの名前。
MYSQL_TEST_TRACE_CRASH	テストプロトコルトレースプラグインがクライアントをクラッシュさせるかどうか。表のあとにある注釈を参照してください。
MYSQL_TEST_TRACE_DEBUG	テストプロトコルトレースプラグインが出力を生成するかどうか。表のあとにある注釈を参照してください。
MYSQL_UNIX_PORT	<code>localhost</code> への接続に使用される、デフォルトの Unix ソケットファイル名。
MYSQLX_TCP_PORT	X プラグイン のデフォルトの TCP/IP ポート番号。
MYSQLX_UNIX_PORT	X プラグイン のデフォルトの Unix ソケットファイル名。 <code>localhost</code> への接続に使用されます。
NOTIFY_SOCKET	<code>systemd</code> と通信するために <code>mysqld</code> で使用されるソケット。
PATH	シェルが MySQL プログラムの検索に使用します。
PKG_CONFIG_PATH	<code>mysqlclient.pc pkg-config</code> ファイルの場所。表のあとにある注釈を参照してください。
TMPDIR	一時ファイルが作成されるディレクトリ。
TZ	ローカルタイムゾーンに設定するようにしてください。 セクション B.3.3.7 「タイムゾーンの問題」 を参照してください。
UMASK	ファイルを作成する際のユーザーファイル作成モード。表のあとにある注釈を参照してください。
UMASK_DIR	ディレクトリを作成する際のユーザーディレクトリ作成モード。表のあとにある注釈を参照してください。
USER	<code>mysqld</code> に接続する際の Windows のデフォルトのユーザー名。

`mysql` の履歴ファイルの詳細は、 [セクション 4.5.1.3 「mysql クライアントロギング」](#) を参照してください。

`MYSQL_PWD` を使用して MySQL パスワードを指定する場合は、極めてセキュアではありませんとみなす必要があります。使用しないでください。 `ps` のバージョンによっては、実行プロセスの環境を表示するオプションがあります。

一部のシステムで、`MYSQL_PWD` を設定した場合、パスワードは `ps` を実行するすべてのユーザーに公開されます。そのようなバージョンの `ps` を持たないシステムであっても、ユーザーが処理環境を調査できるほかの方法がないと想定することは賢明ではありません。

`MYSQL_PWD` は、MySQL 8.0 では非推奨です。将来のバージョンの MySQL では削除される予定です。

`MYSQL_TEST_LOGIN_FILE` は、ログインパスファイル (`mysql_config_editor` によって作成されたファイル) のパス名です。設定されていない場合、デフォルト値は Windows では `%APPDATA%\MySQL\mylogin.cnf` ディレクトリ、Windows 以外のシステムでは `$HOME/.mylogin.cnf` です。セクション 4.6.7 「`mysql_config_editor` — MySQL 構成ユーティリティー」を参照してください。

`MYSQL_TEST_TRACE_DEBUG` および `MYSQL_TEST_TRACE_CRASH` 変数は、MySQL がプラグインを有効にして構築されている場合、テストプロトコルトレースクライアントプラグインを制御します。詳細は、[Using the Test Protocol Trace Plugin](#) を参照してください。

`UMASK` および `UMASK_DIR` のデフォルト値は、それぞれ `0640` および `0750` です。MySQL では、`UMASK` または `UMASK_DIR` の値がゼロで始まる場合、その値は 8 進数と見なされます。たとえば、`0600` オクタルは `384` 桁であるため、`UMASK=0600` の設定は `UMASK=384` と同等です。

`UMASK` 変数および `UMASK_DIR` 変数は、その名前にもかかわらず、マスクではなくモードとして使用されます。

- `UMASK` が設定されている場合、`mysqld` は (`$UMASK | 0600`) をファイル作成のモードとして使用し、新しく作成されるファイルのモードは `0600` から `0666` の範囲になります (すべて 8 進数の値)。
- `UMASK_DIR` が設定されている場合、`mysqld` は (`$UMASK_DIR | 0700`) をディレクトリ作成のベースモードとして使用し、次に `~(~$UMASK & 0666)` との AND が取られます。そのため新しく作成されるファイルのモードは `0700` から `0777` の範囲になります (すべて 8 進数の値)。AND 演算によってディレクトリモードから読み取り/書き込み権が削除されることがありますが、実行権が削除されることはありません。

セクション B.3.3.1 「ファイル権限の問題」も参照してください。

MySQL プログラムの構築に `pkg-config` を使用する場合は、`PKG_CONFIG_PATH` の設定が必要になることがあります。[Building C API Client Programs Using pkg-config](#) を参照してください。

4.10 MySQL での Unix シグナル処理

Unix および Unix に似たシステムでは、プロセスは、`root` システムアカウントまたはプロセスを所有するシステムアカウントによって送信されるシグナルの受信者になります。シグナルは、`kill` コマンドを使用して送信できます。一部のコマンドインタプリタは、特定のキーシーケンスを `Control+C` などのシグナルに関連付けて、`SIGINT` シグナルを送信します。このセクションでは、MySQL サーバーおよびクライアントプログラムがシグナルに応答する方法について説明します。

- シグナルへのサーバー応答
- シグナルに対するクライアントのレスポンス

シグナルへのサーバー応答

`mysqld` は、次のようにシグナルに応答します:

- `SIGTERM` によってサーバーはシャットダウンします。これは、サーバーに接続せずに `SHUTDOWN` ステートメントを実行する場合と似ています (停止するには、`SHUTDOWN` 権限を持つアカウントが必要です)。
- `SIGHUP` によって、サーバーは付与テーブルをリロードし、テーブル、ログ、スレッドキャッシュ、およびホストキャッシュをフラッシュします。これらのアクションは、`FLUSH` ステートメントのさまざまな形式に似ています。シグナルを送信すると、サーバーに接続せずにフラッシュ操作を実行できるようになります。これには、それらの操作に十分な権限を持つ MySQL アカウントが必要です。MySQL 8.0.20 より前のサーバーでは、次の形式のステータスレポートもエラーログに書き込まれます:

```
Status information:
```

```
Current dir: /var/mysql/data/  
Running threads: 4 Stack size: 262144
```

```
Current locks:
lock: 0x7f742c02c0e0:

lock: 0x2cee2a20:
:
lock: 0x207a080:

Key caches:
default
Buffer_size: 8388608
Block_size: 1024
Division_limit: 100
Age_limit: 300
blocks used: 4
not flushed: 0
w_requests: 0
writes: 0
r_requests: 8
reads: 4

handler status:
read_key: 13
read_next: 4
read_rnd: 0
read_first: 13
write: 1
delete: 0
update: 0

Table status:
Opened tables: 121
Open tables: 114
Open files: 18
Open streams: 0

Memory status:
<malloc version="1">
<heap nr="0">
<sizes>
<size from="17" to="32" total="32" count="1"/>
<size from="33" to="48" total="96" count="2"/>
<size from="33" to="33" total="33" count="1"/>
<size from="97" to="97" total="6014" count="62"/>
<size from="113" to="113" total="904" count="8"/>
<size from="193" to="193" total="193" count="1"/>
<size from="241" to="241" total="241" count="1"/>
<size from="609" to="609" total="609" count="1"/>
<size from="16369" to="16369" total="49107" count="3"/>
<size from="24529" to="24529" total="98116" count="4"/>
<size from="32689" to="32689" total="32689" count="1"/>
<unsorted from="241" to="7505" total="7746" count="2"/>
</sizes>
<total type="fast" count="3" size="128"/>
<total type="rest" count="84" size="195652"/>
<system type="current" size="690774016"/>
<system type="max" size="690774016"/>
<aspace type="total" size="690774016"/>
<aspace type="mprotect" size="690774016"/>
</heap>
:
<total type="fast" count="85" size="5520"/>
<total type="rest" count="116" size="316820"/>
<total type="mmap" count="82" size="939954176"/>
<system type="current" size="695717888"/>
<system type="max" size="695717888"/>
<aspace type="total" size="695717888"/>
<aspace type="mprotect" size="695717888"/>
</malloc>

Events status:
LLA = Last Locked At  LUA = Last Unlocked At
WOC = Waiting On Condition  DL = Data Locked

Event scheduler status:
State : INITIALIZED
```

```
Thread id : 0
LLA      : n/a:0
LUA      : n/a:0
WOC      : NO
Workers  : 0
Executed : 0
Data locked: NO

Event queue status:
Element count : 0
Data locked   : NO
Attempting lock : NO
LLA          : init_queue:95
LUA          : init_queue:103
WOC          : NO
Next activation : never
```

- MySQL 8.0.19 の時点で、[SIGUSR1](#) により、サーバーはエラーログ、一般クエリーログおよびスロークエリーログをフラッシュします。[SIGUSR1](#) の使用方法の 1 つは、サーバーに接続せずにログローテーションを実装することです。サーバーに接続するには、これらの操作に十分な権限を持つ MySQL アカウントが必要です。ログローテーションの詳細は、[セクション5.4.6「サーバーログの保守」](#) を参照してください。

[SIGUSR1](#) へのサーバーレスポンスは、[SIGHUP](#) へのレスポンスのサブセットであり、スレッドおよびホストキャッシュのフラッシュやエラーログへのステータスレポートの書込みなど、他の [SIGHUP](#) 効果なしで特定のログをフラッシュする、より多くの「軽量」シグナルとして [SIGUSR1](#) を使用できます。

- [SIGINT](#) は通常、サーバーによって無視されます。`--gdb` オプションを使用してサーバーを起動すると、デバッグ目的で [SIGINT](#) 用の割り込みハンドラがインストールされます。[セクション5.9.1.4「gdbでのmysqldのデバッグ」](#) を参照してください。

シグナルに対するクライアントのレスポンス

MySQL クライアントプログラムは、次のようにシグナルに応答します:

- `mysql` クライアントは、[SIGINT](#) (通常は Control+C と入力した結果) を命令として解釈して、現在のステートメントがある場合は中断し、それ以外の場合は一部の入力行を取り消します。この動作は、`--sigint-ignore` オプションを使用して無効にし、[SIGINT](#) シグナルを無視できます。
- MySQL クライアントライブラリを使用するクライアントプログラムは、デフォルトで [SIGPIPE](#) シグナルをブロックします。次のバリエーションが可能です:
 - クライアントは独自の [SIGPIPE](#) ハンドラをインストールして、デフォルトの動作をオーバーライドできます。[Writing C API Threaded Client Programs](#) を参照してください。
 - クライアントは、接続時に `mysql_real_connect()` に `CLIENT_IGNORE_SIGPIPE` オプションを指定することで、[SIGPIPE](#) ハンドラのインストールを防止できます。[mysql_real_connect\(\)](#) を参照してください。

第 5 章 MySQL サーバーの管理

目次

5.1 MySQL Server	556
5.1.1 サーバーの構成	556
5.1.2 サーバー構成のデフォルト値	557
5.1.3 サーバー構成の検証	558
5.1.4 サーバーオプション、システム変数およびステータス変数リファレンス	559
5.1.5 サーバーシステム変数リファレンス	604
5.1.6 サーバーステータス変数リファレンス	628
5.1.7 サーバーコマンドオプション	644
5.1.8 サーバーシステム変数	669
5.1.9 システム変数の使用	806
5.1.10 サーバーステータス変数	834
5.1.11 サーバー SQL モード	854
5.1.12 接続管理	864
5.1.13 IPv6 サポート	871
5.1.14 ネットワークネームスペースのサポート	875
5.1.15 MySQL Server でのタイムゾーンのサポート	880
5.1.16 リソースグループ	885
5.1.17 サーバー側ヘルプのサポート	889
5.1.18 クライアントセッション状態の変更のサーバートラッキング	890
5.1.19 サーバーの停止プロセス	892
5.2 MySQL データディレクトリ	894
5.3 mysql システムスキーマ	895
5.4 MySQL Server ログ	900
5.4.1 一般クエリーログおよびスロークエリーログの出力先の選択	900
5.4.2 エラーログ	903
5.4.3 一般クエリーログ	921
5.4.4 バイナリログ	922
5.4.5 スロークエリーログ	937
5.4.6 サーバーログの保守	940
5.5 MySQL のコンポーネント	942
5.5.1 コンポーネントのインストールおよびアンインストール	942
5.5.2 コンポーネント情報の取得	943
5.5.3 エラーログコンポーネント	943
5.5.4 クエリー属性コンポーネント	945
5.6 MySQL Server プラグイン	945
5.6.1 プラグインのインストールおよびアンインストール	946
5.6.2 サーバープラグイン情報の取得	950
5.6.3 MySQL Enterprise Thread Pool	951
5.6.4 リライタクエリーリライトプラグイン	958
5.6.5 ddl_rewriter プラグイン	966
5.6.6 バージョントークン	968
5.6.7 クローンプラグイン	978
5.6.8 MySQL プラグインサービス	1000
5.7 MySQL Server のユーザー定義関数	1007
5.7.1 ユーザー定義関数のインストールおよびアンインストール	1008
5.7.2 ユーザー定義関数情報の取得	1009
5.8 1 つのマシン上での複数の MySQL インスタンスの実行	1009
5.8.1 複数のデータディレクトリのセットアップ	1011
5.8.2 Windows 上での複数の MySQL インスタンスの実行	1012
5.8.3 Unix 上での複数の MySQL インスタンスの実行	1014
5.8.4 複数サーバー環境でのクライアントプログラムの使用	1015
5.9 MySQL のデバッグ	1016
5.9.1 MySQL サーバーのデバッグ	1016

5.9.2 MySQL クライアントのデバッグ	1022
5.9.3 LOCK_ORDER ツール	1022
5.9.4 DBUG パッケージ	1027

MySQL サーバー (`mysqld`) は、MySQL インストールの中核を担うメインプログラムです。この章では、MySQL サーバーの概要を説明し、一般的なサーバー管理を取り上げます。

- サーバーの構成
- データディレクトリ (特に `mysql` システムスキーマ)
- サーバーログファイル
- 単一マシン上の複数のサーバーの管理

管理に関するトピックの追加情報は、次も参照してください。

- [第6章「セキュリティ」](#)
- [第7章「バックアップとリカバリ」](#)
- [第17章「レプリケーション」](#)

5.1 MySQL Server

`mysqld` は MySQL Server です。次の説明では、これらの MySQL Server の構成トピックについて扱います。

- サーバーがサポートする起動オプション。コマンド行、構成ファイル、あるいは両方でこれらのオプションを指定できます。
- サーバースystem変数。これらの変数は、起動オプションの現在の状態および値を反映したもので、変数の一部はサーバーの実行中に変更できます。
- サーバーステータス変数。これらの変数には、ランタイム動作についてのカウンタおよび統計が含まれています。
- サーバー SQL モードの設定方法。この設定は、たとえば別のデータベースシステムからのコードとの互換性を保ったり、特定の状況についてのエラー処理を制御したりするために、SQL の構文およびセマンティクスの特定の側面を変更します。
- サーバーがクライアント接続を管理する方法。
- IPv6 およびネットワークネームスペースのサポートの構成および使用。
- タイムゾーンサポートの構成および使用。
- リソースグループの使用。
- サーバー側のヘルプ機能。
- クライアントセッション状態の変更を有効にするために提供されている機能。
- サーバーのシャットダウンプロセス。テーブルのタイプ (トランザクションまたは非トランザクション) やレプリケーションを使用するかどうかに応じて、パフォーマンスおよび信頼性についての考慮事項があります。

MySQL 8.0 で追加、非推奨または削除された MySQL サーバ変数およびオプションのリストは、[セクション 1.4「MySQL 8.0 で追加、非推奨または削除されたサーバーおよびステータスの変数とオプション」](#) を参照してください。

注記

すべてのストレージエンジンが、すべての MySQL Server のバイナリおよび構成によってサポートされているわけではありません。MySQL Server のインストール環境がサポートしているストレージエンジンを判別するための方法を見つけるには、[セクション 13.7.7.16「SHOW ENGINES ステートメント」](#) を参照してください。

5.1.1 サーバーの構成

MySQL サーバーである `mysqld` には、起動時にその操作を構成するために設定できる多くのコマンドオプションおよびシステム変数があります。サーバーで使用されるデフォルトのコマンドオプションとシステム変数の値を確認するには、次のコマンドを実行します:

```
shell> mysqld --verbose --help
```

このコマンドは、すべての `mysqld` オプションと構成可能なシステム変数のリストを生成します。その出力には、デフォルトのオプションと変数の値が含まれ、次のようになります:

```
abort-slave-event-count      0
allow-suspicious-udfs       FALSE
archive                       ON
auto-increment-increment    1
auto-increment-offset       1
autocommit                   TRUE
automatic-sp-privileges     TRUE
avoid-temporal-upgrade      FALSE
back-log                      80
basedir                       /home/jon/bin/mysql-8.0/
...
tmpdir                       /tmp
transaction-alloc-block-size 8192
transaction-isolation        REPEATABLE-READ
transaction-prealloc-size    4096
transaction-read-only        FALSE
transaction-write-set-extraction OFF
updatable-views-with-limit   YES
validate-user-plugins        TRUE
verbose                       TRUE
wait-timeout                  28800
```

サーバーの実行時に実際に使用される現在のシステム変数値を確認するには、接続して次のステートメントを実行します:

```
mysql> SHOW VARIABLES;
```

実行中のサーバーの統計およびステータスインジケータを表示するには、次のステートメントを実行します:

```
mysql> SHOW STATUS;
```

システム変数およびステータス情報は、`mysqladmin` コマンドを使用しても入手できます:

```
shell> mysqladmin variables
shell> mysqladmin extended-status
```

すべてのコマンドオプション、システム変数およびステータス変数の詳細は、次のセクションを参照してください:

- [セクション5.1.7「サーバーコマンドオプション」](#)
- [セクション5.1.8「サーバーシステム変数」](#)
- [セクション5.1.10「サーバーステータス変数」](#)

パフォーマンススキーマからより詳細なモニタリング情報を入手できます。[第27章「MySQL パフォーマンススキーマ」](#)を参照してください。また、MySQL `sys` スキーマは、パフォーマンススキーマによって収集されたデータへの便利なアクセスを提供する一連のオブジェクトです。[第28章「MySQL sys スキーマ」](#)を参照してください。

コマンド行で `mysqld` または `mysqld_safe` のオプションを指定する場合、そのサーバーの呼び出しに対してのみ有効です。サーバーの実行のためにオプションを使用するには、それをオプションファイルに入れます。[セクション4.2.2.2「オプションファイルの使用」](#)を参照してください。

5.1.2 サーバー構成のデフォルト値

MySQL Server には多くの操作パラメータがあり、これらはコマンド行オプションまたは構成ファイル (オプションファイル) を使用して、サーバー起動時に変更できます。多くのパラメータを実行時に変更することもできます。起動時または実行時にパラメータを設定することに関する一般的な指示については、[セクション5.1.7「サーバーコマンドオプション」](#) および [セクション5.1.8「サーバーシステム変数」](#) を参照してください。

Windows では、MySQL Installer がユーザーと対話し、`my.ini` という名前のファイルをデフォルトオプションファイルとして基本インストールディレクトリに作成します。

注記

Windows では、`.ini` または `.cnf` というオプションのファイル拡張子が表示されない場合もあります。

インストールプロセスが完了したら、いつでもデフォルトのオプションファイルを編集して、サーバーで使用されるパラメータを変更できます。たとえば、行の先頭の `#` 文字によってコメント化されたファイル内のパラメータ設定を使用するには、`#` を削除し、必要に応じてパラメータ値を変更します。設定を無効にするには、行の先頭に `#` を追加するか、削除します。

Windows 以外のプラットフォームでは、サーバーのインストールまたはデータディレクトリの初期化プロセス中にデフォルトのオプションファイルは作成されません。セクション4.2.2.2「オプションファイルの使用」の指示に従って、オプションファイルを作成します。オプションファイルがない場合、サーバーはデフォルト設定で起動するだけです。これらの設定を確認する方法については、セクション5.1.2「サーバー構成のデフォルト値」を参照してください。

オプションファイルの形式および構文についての追加情報は、セクション4.2.2.2「オプションファイルの使用」を参照してください。

5.1.3 サーバー構成の検証

MySQL 8.0.16 の時点で、MySQL Server は `--validate-config` オプションをサポートしており、サーバーを通常の動作モードで実行せずに、起動構成の問題をチェックできます：

```
mysqld --validate-config
```

エラーが見つからない場合、サーバーは終了コード 0 で終了します。エラーが見つかった場合、サーバーは診断メッセージを表示し、終了コード 1 で終了します。例：

```
shell> mysqld --validate-config --no-such-option
2018-11-05T17:50:12.738919Z 0 [ERROR] [MY-000068] [Server] unknown
option '--no-such-option'.
2018-11-05T17:50:12.738962Z 0 [ERROR] [MY-010119] [Server] Aborting
```

エラーが検出されるとすぐにサーバーは終了します。追加のチェックを実行するには、最初の問題を修正し、`--validate-config` でサーバーを再度実行します。

前述の例では、`--validate-config` を使用するとエラーメッセージが表示され、サーバーの終了コードは 1 です。警告および情報メッセージは、`log_error_verbosity` の値によっては表示されることもありますが、即時検証の終了や終了コード 1 は生成されません。たとえば、このコマンドでは複数の警告が生成され、その両方が表示されます。ただし、エラーは発生しないため、終了コードは 0 になります：

```
shell> mysqld --validate-config --log_error_verbosity=2
--read-only=s --transaction_read_only=s
2018-11-05T15:43:18.445863Z 0 [Warning] [MY-000076] [Server] option
'read_only': boolean value 's' was not recognized. Set to OFF.
2018-11-05T15:43:18.445882Z 0 [Warning] [MY-000076] [Server] option
'transaction-read-only': boolean value 's' was not recognized. Set to OFF.
```

このコマンドでは同じ警告が生成されますが、エラーも生成されるため、エラーメッセージが警告とともに表示され、終了コードは 1 です：

```
shell> mysqld --validate-config --log_error_verbosity=2
--no-such-option --read-only=s --transaction_read_only=s
2018-11-05T15:43:53.152886Z 0 [Warning] [MY-000076] [Server] option
'read_only': boolean value 's' was not recognized. Set to OFF.
2018-11-05T15:43:53.152913Z 0 [Warning] [MY-000076] [Server] option
'transaction-read-only': boolean value 's' was not recognized. Set to OFF.
2018-11-05T15:43:53.164889Z 0 [ERROR] [MY-000068] [Server] unknown
option '--no-such-option'.
2018-11-05T15:43:53.165053Z 0 [ERROR] [MY-010119] [Server] Aborting
```

`--validate-config` オプションの範囲は、通常の起動プロセスを実行せずにサーバーが実行できる構成チェックに制限されます。そのため、構成チェックではストレージエンジンやその他のプラグインやコンポーネントなどは初期化されず、初期化されていないサブシステムに関連付けられたオプションも検証されません。

`--validate-config` はいつでも使用できますが、アップグレード後に特に役立ちます。古いサーバーで以前に使用されていたオプションが、アップグレードされたサーバーで非推奨または廃止とみなされているかどうかを確認する場合に役立ちます。たとえば、`tx_read_only` システム変数は MySQL 5.7 で非推奨になり、8.0 で削除されました。MySQL 5.7 サーバーが `my.cnf` ファイルでそのシステム変数を使用して実行され、MySQL 8.0 にアップグレードされたとしたら、`--validate-config` でアップグレードされたサーバーを実行して構成を確認すると、次の結果が生成されます：

```
shell> mysqld --validate-config
2018-11-05T10:40:02.712141Z 0 [ERROR] [MY-000067] [Server] unknown variable
'tx_read_only=ON'.
2018-11-05T10:40:02.712178Z 0 [ERROR] [MY-010119] [Server] Aborting
```

`--validate-config` を `--defaults-file` オプションとともに使用すると、特定のファイル内のオプションのみを検証できます：

```
shell> mysqld --defaults-file=./my.cnf-test --validate-config
2018-11-05T10:40:02.712141Z 0 [ERROR] [MY-000067] [Server] unknown variable
'tx_read_only=ON'.
2018-11-05T10:40:02.712178Z 0 [ERROR] [MY-010119] [Server] Aborting
```

`--defaults-file` を指定する場合は、コマンドラインの最初のオプションにする必要があることに注意してください。(オプションの順序を逆にして前述の例を実行すると、`--defaults-file` 自体が不明であることを示すメッセージが生成されます。)

5.1.4 サーバーオプション、システム変数およびステータス変数リファレンス

次のテーブルに、`mysqld` 内で適用可能なすべてのコマンドラインオプション、システム変数およびステータス変数を示します。

このテーブルは、コマンド行オプション (Cmd-line)、構成ファイルで有効なオプション (Option file)、サーバーシステム変数 (System Var)、およびステータス変数 (Status var) を 1 つの統合リストに示し、各オプションまたは変数が有効な場所を示しています。コマンド行またはオプションファイルで設定されたサーバーオプションが、対応するシステム変数の名前と異なる場合、変数名は対応するオプションのすぐ下に表示されます。システム変数およびステータス変数の場合、変数のスコープ (Var Scope) は Global、Session、またはその両方です。オプションおよび変数の設定および使用の詳細は、対応するアイテムの説明を参照してください。必要に応じて、アイテムに関する詳細情報へのダイレクトリンクが提供されます。

NDB Cluster に固有のこのテーブルのバージョンについては、[セクション23.3.2.5「NDB Cluster mysqld オプションおよび変数のリファレンス」](#)を参照してください。

表 5.1 「コマンドラインオプション、システム変数およびステータス変数サマリー」

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
abort-slave-event-count	はい	はい				
Aborted_clients				はい	グローバル	いいえ
Aborted_connects				はい	グローバル	いいえ
Acl_cache_items_count				はい	グローバル	いいえ
activate_all_roles_on_login	はい	はい	はい		グローバル	はい
admin_address	はい	はい	はい		グローバル	いいえ
admin_port	はい	はい	はい		グローバル	いいえ
admin-ssl	はい	はい				
admin_ssl_ca	はい	はい	はい		グローバル	はい
admin_ssl_capath	はい	はい	はい		グローバル	はい
admin_ssl_cert	はい	はい	はい		グローバル	はい
admin_ssl_cipher	はい	はい	はい		グローバル	はい
admin_ssl_crl	はい	はい	はい		グローバル	はい
admin_ssl_crlpath	はい	はい	はい		グローバル	はい

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
admin_ssl_key	はい	はい	はい		グローバル	はい
admin_tls_ciphersuites	はい	はい	はい		グローバル	はい
admin_tls_version	はい	はい	はい		グローバル	はい
allow-suspicious-udfs	はい	はい				
ansi	はい	はい				
audit-log	はい	はい				
audit_log_buffer_size	はい	はい	はい		グローバル	いいえ
audit_log_compression	はい	はい	はい		グローバル	いいえ
audit_log_connection_policy	はい	はい	はい		グローバル	はい
audit_log_current_session			はい		両方	いいえ
Audit_log_current_size				はい	グローバル	いいえ
audit_log_encryption	はい	はい	はい		グローバル	いいえ
Audit_log_event_max_drop_size				はい	グローバル	いいえ
Audit_log_events				はい	グローバル	いいえ
Audit_log_events_filtered				はい	グローバル	いいえ
Audit_log_events_lost				はい	グローバル	いいえ
Audit_log_events_written				はい	グローバル	いいえ
audit_log_excluded_accounts	はい	はい	はい		グローバル	はい
audit_log_file	はい	はい	はい		グローバル	いいえ
audit_log_filter_id			はい		両方	いいえ
audit_log_flush			はい		グローバル	はい
audit_log_format	はい	はい	はい		グローバル	いいえ
audit_log_included_accounts	はい	はい	はい		グローバル	はい
audit_log_password_history_keep_days	はい	はい	はい		グローバル	はい
audit_log_policy	はい	はい	はい		グローバル	いいえ
audit_log_prune_bonds	はい	はい	はい		グローバル	はい
audit_log_read_buffer_size	はい	はい	はい		異なる	異なる
audit_log_rotate_size	はい	はい	はい		グローバル	はい
audit_log_statement_policy	はい	はい	はい		グローバル	はい
audit_log_strategy	はい	はい	はい		グローバル	いいえ
Audit_log_total_size				はい	グローバル	いいえ
Audit_log_write_waits				はい	グローバル	いいえ
authentication_id_key_sasl_auth_method	はい	はい	はい		グローバル	はい
authentication_id_key_sasl_bind_base_dn	はい	はい	はい		グローバル	はい
authentication_id_key_sasl_bind_root_dn	はい	はい	はい		グローバル	はい
authentication_id_key_sasl_bind_root_pwd	はい	はい	はい		グローバル	はい
authentication_id_key_sasl_ca_path	はい	はい	はい		グローバル	はい
authentication_id_key_sasl_group_search_attr	はい	はい	はい		グローバル	はい
authentication_id_key_sasl_group_search_filter	はい	はい	はい		グローバル	はい

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
authentication_ldapsasl_init_pool_size	はい	はい	はい		グローバル	はい
authentication_ldapsasl_log_status	はい	はい	はい		グローバル	はい
authentication_ldapsasl_max_pool_size	はい	はい	はい		グローバル	はい
authentication_ldapsasl_referral	はい	はい	はい		グローバル	はい
authentication_ldapsasl_server_host	はい	はい	はい		グローバル	はい
authentication_ldapsasl_server_port	はい	はい	はい		グローバル	はい
Authenticationldap_sasl_supported_methods				はい	グローバル	いいえ
authentication_ldapsasl_tls	はい	はい	はい		グローバル	はい
authentication_ldapsasl_user_search_attr	はい	はい	はい		グローバル	はい
authentication_ldapsimple_auth_method_name	はい	はい	はい		グローバル	はい
authentication_ldapsimple_bind_target_dn	はい	はい	はい		グローバル	はい
authentication_ldapsimple_bind_target_dn	はい	はい	はい		グローバル	はい
authentication_ldapsimple_bind_target_pwd	はい	はい	はい		グローバル	はい
authentication_ldapsimple_ca_path	はい	はい	はい		グローバル	はい
authentication_ldapsimple_group_search_attr	はい	はい	はい		グローバル	はい
authentication_ldapsimple_group_search_filter	はい	はい	はい		グローバル	はい
authentication_ldapsimple_init_pool_size	はい	はい	はい		グローバル	はい
authentication_ldapsimple_log_status	はい	はい	はい		グローバル	はい
authentication_ldapsimple_max_pool_size	はい	はい	はい		グローバル	はい
authentication_ldapsimple_referral	はい	はい	はい		グローバル	はい
authentication_ldapsimple_server_host	はい	はい	はい		グローバル	はい
authentication_ldapsimple_server_port	はい	はい	はい		グローバル	はい
authentication_ldapsimple_tls	はい	はい	はい		グローバル	はい
authentication_ldapsimple_user_search_attr	はい	はい	はい		グローバル	はい
authentication_wildows_log_level	はい	はい	はい		グローバル	いいえ
authentication_wildows_use_principal_name	はい	はい	はい		グローバル	いいえ
auto_generate_certificate	はい	はい	はい		グローバル	いいえ
auto_increment_increment	はい	はい	はい		両方	はい
auto_increment_offset	はい	はい	はい		両方	はい
autocommit	はい	はい	はい		両方	はい
automatic_sp_privileges	はい	はい	はい		グローバル	はい
avoid_temporal_upgrade	はい	はい	はい		グローバル	はい
back_log	はい	はい	はい		グローバル	いいえ
basedir	はい	はい	はい		グローバル	いいえ
big_tables	はい	はい	はい		両方	はい
bind_address	はい	はい	はい		グローバル	いいえ
Binlog_cache_disk_use				はい	グローバル	いいえ
binlog_cache_size	はい	はい	はい		グローバル	はい
Binlog_cache_use				はい	グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
binlog-checksum	はい	はい				
binlog_checksum	はい	はい	はい		グローバル	はい
binlog_direct_nontransactional_updates	はい	はい	はい		両方	はい
binlog-do-db	はい	はい				
binlog_encryption	はい	はい	はい		グローバル	はい
binlog_error_action	はい	はい	はい		グローバル	はい
binlog_expire_logs_seconds	はい	はい	はい		グローバル	はい
binlog_format	はい	はい	はい		両方	はい
binlog_group_commit_sync_delay	はい	はい	はい		グローバル	はい
binlog_group_commit_sync_no_delay_count	はい	はい	はい		グローバル	はい
binlog_gtid_simple_recovery	はい	はい	はい		グローバル	いいえ
binlog-ignore-db	はい	はい				
binlog_max_flush_queue_time	はい	はい	はい		グローバル	はい
binlog_order_commits	はい	はい	はい		グローバル	はい
binlog_rotate_encryption_master_key_at_startup	はい	はい	はい		グローバル	いいえ
binlog_row_event_max_size	はい	はい	はい		グローバル	いいえ
binlog_row_image	はい	はい	はい		両方	はい
binlog_row_metadata	はい	はい	はい		グローバル	はい
binlog_row_value_options	はい	はい	はい		両方	はい
binlog_rows_query_log_events	はい	はい	はい		両方	はい
Binlog_stmt_cache_disk_use				はい	グローバル	いいえ
binlog_stmt_cache_size	はい	はい	はい		グローバル	はい
Binlog_stmt_cache_use				はい	グローバル	いいえ
binlog_transaction_compression	はい	はい	はい		グローバル	はい
binlog_transaction_compression_level	はい	はい	はい		グローバル	はい
binlog_transaction_dependency_history_size	はい	はい	はい		グローバル	はい
binlog_transaction_dependency_tracking	はい	はい	はい		グローバル	はい
block_encryption_mode	はい	はい	はい		両方	はい
bulk_insert_buffer_size	はい	はい	はい		両方	はい
Bytes_received				はい	両方	いいえ
Bytes_sent				はい	両方	いいえ
caching_sha2_password_auto_generate_rsa_keys	はい	はい	はい		グローバル	いいえ
caching_sha2_password_digest_rounds	はい	はい	はい		グローバル	いいえ
caching_sha2_password_private_key_path	はい	はい	はい		グローバル	いいえ
caching_sha2_password_public_key_path	はい	はい	はい		グローバル	いいえ
Caching_sha2_password_rsa_public_key				はい	グローバル	いいえ
character_set_client			はい		両方	はい

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
character-set-client-handshake	はい	はい				
character_set_connection			はい		両方	はい
character_set_database (note 1)			はい		両方	はい
character_set_filesystem	はい	はい	はい		両方	はい
character_set_results			はい		両方	はい
character_set_server	はい	はい	はい		両方	はい
character_set_system			はい		グローバル	いいえ
character_sets_dir	はい	はい	はい		グローバル	いいえ
check_proxy_usage	はい	はい	はい		グローバル	はい
chroot	はい	はい				
clone_autotune_currency	はい	はい	はい		グローバル	はい
clone_buffer_size	はい	はい	はい		グローバル	はい
clone_ddl_timeout	はい	はい	はい		グローバル	はい
clone_enable_compression	はい	はい	はい		グローバル	はい
clone_max_concurrency	はい	はい	はい		グローバル	はい
clone_max_data_width	はい	はい	はい		グローバル	はい
clone_max_network_bandwidth	はい	はい	はい		グローバル	はい
clone_ssl_ca	はい	はい	はい		グローバル	はい
clone_ssl_cert	はい	はい	はい		グローバル	はい
clone_ssl_key	はい	はい	はい		グローバル	はい
clone_valid_domains	はい	はい	はい		グローバル	はい
collation_connection			はい		両方	はい
collation_database (note 1)			はい		両方	はい
collation_server	はい	はい	はい		両方	はい
Com_admin_commands				はい	両方	いいえ
Com_alter_db				はい	両方	いいえ
Com_alter_event				はい	両方	いいえ
Com_alter_function				はい	両方	いいえ
Com_alter_procedure				はい	両方	いいえ
Com_alter_resource_group				はい	グローバル	いいえ
Com_alter_server				はい	両方	いいえ
Com_alter_table				はい	両方	いいえ
Com_alter_tablespace				はい	両方	いいえ
Com_alter_user				はい	両方	いいえ
Com_alter_user_default_role				はい	グローバル	いいえ
Com_analyze				はい	両方	いいえ
Com_assign_to_keycache				はい	両方	いいえ

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
Com_begin				はい	両方	いいえ
Com_binlog				はい	両方	いいえ
Com_call_procedure				はい	両方	いいえ
Com_change_db				はい	両方	いいえ
Com_change_master				はい	両方	いいえ
Com_change_repl_filter				はい	両方	いいえ
Com_check				はい	両方	いいえ
Com_checksum				はい	両方	いいえ
Com_clone				はい	グローバル	いいえ
Com_commit				はい	両方	いいえ
Com_create_db				はい	両方	いいえ
Com_create_event				はい	両方	いいえ
Com_create_function				はい	両方	いいえ
Com_create_index				はい	両方	いいえ
Com_create_procedure				はい	両方	いいえ
Com_create_resource_group				はい	グローバル	いいえ
Com_create_role				はい	グローバル	いいえ
Com_create_server				はい	両方	いいえ
Com_create_table				はい	両方	いいえ
Com_create_trigger				はい	両方	いいえ
Com_create_udf				はい	両方	いいえ
Com_create_user				はい	両方	いいえ
Com_create_view				はい	両方	いいえ
Com_dealloc_sql				はい	両方	いいえ
Com_delete				はい	両方	いいえ
Com_delete_multi				はい	両方	いいえ
Com_do				はい	両方	いいえ
Com_drop_db				はい	両方	いいえ
Com_drop_event				はい	両方	いいえ
Com_drop_function				はい	両方	いいえ
Com_drop_index				はい	両方	いいえ
Com_drop_procedure				はい	両方	いいえ
Com_drop_resource_group				はい	グローバル	いいえ
Com_drop_role				はい	グローバル	いいえ
Com_drop_server				はい	両方	いいえ
Com_drop_table				はい	両方	いいえ
Com_drop_trigger				はい	両方	いいえ
Com_drop_user				はい	両方	いいえ
Com_drop_view				はい	両方	いいえ
Com_empty_query				はい	両方	いいえ

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
Com_execute_sql				はい	両方	いいえ
Com_explain_other				はい	両方	いいえ
Com_flush				はい	両方	いいえ
Com_get_diagnostics				はい	両方	いいえ
Com_grant				はい	両方	いいえ
Com_grant_roles				はい	グローバル	いいえ
Com_group_replication_start				はい	グローバル	いいえ
Com_group_replication_stop				はい	グローバル	いいえ
Com_ha_close				はい	両方	いいえ
Com_ha_open				はい	両方	いいえ
Com_ha_read				はい	両方	いいえ
Com_help				はい	両方	いいえ
Com_insert				はい	両方	いいえ
Com_insert_select				はい	両方	いいえ
Com_install_component				はい	グローバル	いいえ
Com_install_plugin				はい	両方	いいえ
Com_kill				はい	両方	いいえ
Com_load				はい	両方	いいえ
Com_lock_tables				はい	両方	いいえ
Com_optimize				はい	両方	いいえ
Com_preload_keys				はい	両方	いいえ
Com_prepare_sql				はい	両方	いいえ
Com_purge				はい	両方	いいえ
Com_purge_before_date				はい	両方	いいえ
Com_release_savepoint				はい	両方	いいえ
Com_rename_table				はい	両方	いいえ
Com_rename_user				はい	両方	いいえ
Com_repair				はい	両方	いいえ
Com_replace				はい	両方	いいえ
Com_replace_select				はい	両方	いいえ
Com_replica_start				はい	両方	いいえ
Com_replica_stop				はい	両方	いいえ
Com_reset				はい	両方	いいえ
Com_resignal				はい	両方	いいえ
Com_restart				はい	両方	いいえ
Com_revoke				はい	両方	いいえ
Com_revoke_all				はい	両方	いいえ
Com_revoke_roles				はい	グローバル	いいえ
Com_rollback				はい	両方	いいえ
Com_rollback_to_savepoint				はい	両方	いいえ

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
Com_savepoint				はい	両方	いいえ
Com_select				はい	両方	いいえ
Com_set_option				はい	両方	いいえ
Com_set_resource_group				はい	グローバル	いいえ
Com_set_role				はい	グローバル	いいえ
Com_show_authors				はい	両方	いいえ
Com_show_binlog_events				はい	両方	いいえ
Com_show_binlogs				はい	両方	いいえ
Com_show_charsets				はい	両方	いいえ
Com_show_collations				はい	両方	いいえ
Com_show_contributors				はい	両方	いいえ
Com_show_create_db				はい	両方	いいえ
Com_show_create_event				はい	両方	いいえ
Com_show_create_func				はい	両方	いいえ
Com_show_create_proc				はい	両方	いいえ
Com_show_create_table				はい	両方	いいえ
Com_show_create_trigger				はい	両方	いいえ
Com_show_create_user				はい	両方	いいえ
Com_show_databases				はい	両方	いいえ
Com_show_engine_logs				はい	両方	いいえ
Com_show_engine_mutex				はい	両方	いいえ
Com_show_engine_status				はい	両方	いいえ
Com_show_errors				はい	両方	いいえ
Com_show_events				はい	両方	いいえ
Com_show_fields				はい	両方	いいえ
Com_show_function_code				はい	両方	いいえ
Com_show_function_status				はい	両方	いいえ
Com_show_grants				はい	両方	いいえ
Com_show_keys				はい	両方	いいえ
Com_show_master_status				はい	両方	いいえ
Com_show_ndb_status				はい	両方	いいえ
Com_show_open_tables				はい	両方	いいえ
Com_show_plugins				はい	両方	いいえ
Com_show_privileges				はい	両方	いいえ
Com_show_procedure_code				はい	両方	いいえ
Com_show_procedure_status				はい	両方	いいえ
Com_show_processlist				はい	両方	いいえ
Com_show_profile				はい	両方	いいえ
Com_show_profiles				はい	両方	いいえ
Com_show_relaylog_events				はい	両方	いいえ

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
Com_show_replica_status				はい	両方	いいえ
Com_show_replicas				はい	両方	いいえ
Com_show_slave_hosts				はい	両方	いいえ
Com_show_slave_status				はい	両方	いいえ
Com_show_status				はい	両方	いいえ
Com_show_storage_engines				はい	両方	いいえ
Com_show_table_status				はい	両方	いいえ
Com_show_tables				はい	両方	いいえ
Com_show_triggers				はい	両方	いいえ
Com_show_variables				はい	両方	いいえ
Com_show_warnings				はい	両方	いいえ
Com_shutdown				はい	両方	いいえ
Com_signal				はい	両方	いいえ
Com_slave_start				はい	両方	いいえ
Com_slave_stop				はい	両方	いいえ
Com_stmt_close				はい	両方	いいえ
Com_stmt_execute				はい	両方	いいえ
Com_stmt_fetch				はい	両方	いいえ
Com_stmt_prepare				はい	両方	いいえ
Com_stmt_reprepare				はい	両方	いいえ
Com_stmt_reset				はい	両方	いいえ
Com_stmt_send_long_data				はい	両方	いいえ
Com_truncate				はい	両方	いいえ
Com_uninstall_component				はい	グローバル	いいえ
Com_uninstall_plugin				はい	両方	いいえ
Com_unlock_tables				はい	両方	いいえ
Com_update				はい	両方	いいえ
Com_update_multi				はい	両方	いいえ
Com_xa_commit				はい	両方	いいえ
Com_xa_end				はい	両方	いいえ
Com_xa_prepare				はい	両方	いいえ
Com_xa_recover				はい	両方	いいえ
Com_xa_rollback				はい	両方	いいえ
Com_xa_start				はい	両方	いいえ
completion_type	はい	はい	はい		両方	はい
Compression				はい	セッション	いいえ
Compression_algorithm				はい	グローバル	いいえ
Compression_level				はい	グローバル	いいえ
concurrent_insert	はい	はい	はい		グローバル	はい
connect_timeout	はい	はい	はい		グローバル	はい

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
Connection_control_delay_generated				はい	グローバル	いいえ
connection_control_failed_connection_threshold	はい	はい	はい		グローバル	はい
connection_control_max_connection_delay	はい	はい	はい		グローバル	はい
connection_control_min_connection_delay	はい	はい	はい		グローバル	はい
Connection_errors_accept				はい	グローバル	いいえ
Connection_errors_internal				はい	グローバル	いいえ
Connection_errors_max_connections				はい	グローバル	いいえ
Connection_errors_peer_address				はい	グローバル	いいえ
Connection_errors_select				はい	グローバル	いいえ
Connection_errors_tcpwrap				はい	グローバル	いいえ
Connections				はい	グローバル	いいえ
console	はい	はい				
core-file	はい	はい				
core_file			はい		グローバル	いいえ
create_admin_listen_thread	はい	はい	はい		グローバル	いいえ
Created_tmp_disk_tables				はい	両方	いいえ
Created_tmp_files				はい	グローバル	いいえ
Created_tmp_tables				はい	両方	いいえ
cte_max_recursion_depth	はい	はい	はい		両方	はい
Current_tls_ca				はい	グローバル	いいえ
Current_tls_capath				はい	グローバル	いいえ
Current_tls_cert				はい	グローバル	いいえ
Current_tls_cipher				はい	グローバル	いいえ
Current_tls_ciphersuites				はい	グローバル	いいえ
Current_tls_crl				はい	グローバル	いいえ
Current_tls_crlpath				はい	グローバル	いいえ
Current_tls_key				はい	グローバル	いいえ
Current_tls_version				はい	グローバル	いいえ
daemon_memcached_enable_binlog	はい	はい	はい		グローバル	いいえ
daemon_memcached_engine_lib_path	はい	はい	はい		グローバル	いいえ
daemon_memcached_engine_lib_path	はい	はい	はい		グローバル	いいえ
daemon_memcached_option	はい	はい	はい		グローバル	いいえ
daemon_memcached_r_batch_size	はい	はい	はい		グローバル	いいえ
daemon_memcached_w_batch_size	はい	はい	はい		グローバル	いいえ
daemonize	はい	はい				
datadir	はい	はい	はい		グローバル	いいえ
ddl-rewriter	はい	はい				
debug	はい	はい	はい		両方	はい
debug_sync			はい		セッション	はい

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
debug-sync-timeout	はい	はい				
default_authentication_plugin	はい	はい	はい		グローバル	いいえ
default_collation_for_utf8mb4			はい		両方	はい
default_password_lifetime	はい	はい	はい		グローバル	はい
default_storage_engine	はい	はい	はい		両方	はい
default_table_definition	はい	はい	はい		両方	はい
default-time-zone	はい	はい				
default_tmp_storage_engine	はい	はい	はい		両方	はい
default_week_format	はい	はい	はい		両方	はい
defaults-extra-file	はい					
defaults-file	はい					
defaults-group-suffix	はい					
delay_key_write	はい	はい	はい		グローバル	はい
Delayed_errors				はい	グローバル	いいえ
delayed_insert_threads	はい	はい	はい		グローバル	はい
Delayed_insert_threads				はい	グローバル	いいえ
delayed_insert_timeout	はい	はい	はい		グローバル	はい
delayed_queue_size	はい	はい	はい		グローバル	はい
Delayed_writes				はい	グローバル	いいえ
disabled_storage_engines	はい	はい	はい		グローバル	いいえ
disconnect_on_expired_password	はい	はい	はい		グローバル	いいえ
disconnect-slave-event-count	はい	はい				
div_precision_increment	はい	はい	はい		両方	はい
dragnet.log_error_filter_rules	はい	はい	はい		グローバル	はい
dragnet.Status				はい	グローバル	いいえ
early-plugin-load	はい	はい				
end_markers_in_json	はい	はい	はい		両方	はい
enforce_gtid_consistency	はい	はい	はい		グローバル	はい
eq_range_index_limit	はい	はい	はい		両方	はい
error_count			はい		セッション	いいえ
Error_log_buffered_bytes				はい	グローバル	いいえ
Error_log_buffered_events				はい	グローバル	いいえ
Error_log_expired_events				はい	グローバル	いいえ
Error_log_latest_write				はい	グローバル	いいえ
event_scheduler	はい	はい	はい		グローバル	はい

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
exit-info	はい	はい				
expire_logs_days	はい	はい	はい		グローバル	はい
explicit_defaults_for_timestamp	はい	はい	はい		両方	はい
external-locking	はい	はい				
- 変数: skip_external_locking						
external_user			はい		セッション	いいえ
federated	はい	はい				
Firewall_access_denied				はい	グローバル	いいえ
Firewall_access_granted				はい	グローバル	いいえ
Firewall_cached_entries				はい	グローバル	いいえ
flush	はい	はい	はい		グローバル	はい
Flush_commands				はい	グローバル	いいえ
flush_time	はい	はい	はい		グローバル	はい
foreign_key_checks			はい		両方	はい
ft_boolean_syntax	はい	はい	はい		グローバル	はい
ft_max_word_len	はい	はい	はい		グローバル	いいえ
ft_min_word_len	はい	はい	はい		グローバル	いいえ
ft_query_expansion_limit	はい	はい	はい		グローバル	いいえ
ft_stopword_file	はい	はい	はい		グローバル	いいえ
gdb	はい	はい				
general_log	はい	はい	はい		グローバル	はい
general_log_file	はい	はい	はい		グローバル	はい
generated_random_password_length	はい	はい	はい		両方	はい
group_concat_max_len	はい	はい	はい		両方	はい
group_replicationadvertise_recovery_endpoints	はい	はい	はい		グローバル	はい
group_replicationallow_local_low_version_join	はい	はい	はい		グローバル	はい
group_replicationauto_increment_increment	はい	はい	はい		グローバル	はい
group_replicationauto_rejoin_tries	はい	はい	はい		グローバル	はい
group_replicationbootstrap_group	はい	はい	はい		グローバル	はい
group_replicationclone_threshold	はい	はい	はい		グローバル	はい
group_replicationcommunication_debug_options	はい	はい	はい		グローバル	はい
group_replicationcommunication_max_message_size	はい	はい	はい		グローバル	はい
group_replicationcomponents_timeout	はい	はい	はい		グローバル	はい
group_replicationcompression_threshold	はい	はい	はい		グローバル	はい
group_replicationconsistency	はい	はい	はい		両方	はい
group_replicationforce_update_everywhere_checks	はい	はい	はい		グローバル	はい
group_replicationexit_state_action	はい	はい	はい		グローバル	はい
group_replicationflow_control_apply_threshold	はい	はい	はい		グローバル	はい
group_replicationflow_control_cancel_threshold	はい	はい	はい		グローバル	はい

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
group_replication_start_on_boot	はい	はい	はい		グローバル	はい
group_replication_source	はい	はい	はい		グローバル	はい
group_replication_transaction_size_limit	はい	はい	はい		グローバル	はい
group_replication_reachable_max_retry_timeout	はい	はい	はい		グローバル	はい
gtid_executed			はい		異なる	いいえ
gtid_executed_compression_period	はい	はい	はい		グローバル	はい
gtid_mode	はい	はい	はい		グローバル	はい
gtid_next			はい		セッション	はい
gtid_owned			はい		両方	いいえ
gtid_purged			はい		グローバル	はい
Handler_commit				はい	両方	いいえ
Handler_delete				はい	両方	いいえ
Handler_discover				はい	両方	いいえ
Handler_external_lock				はい	両方	いいえ
Handler_mrr_init				はい	両方	いいえ
Handler_prepare				はい	両方	いいえ
Handler_read_first				はい	両方	いいえ
Handler_read_key				はい	両方	いいえ
Handler_read_last				はい	両方	いいえ
Handler_read_next				はい	両方	いいえ
Handler_read_prev				はい	両方	いいえ
Handler_read_rnd				はい	両方	いいえ
Handler_read_rnd_next				はい	両方	いいえ
Handler_rollback				はい	両方	いいえ
Handler_savepoint				はい	両方	いいえ
Handler_savepoint_rollback				はい	両方	いいえ
Handler_update				はい	両方	いいえ
Handler_write				はい	両方	いいえ
have_compress			はい		グローバル	いいえ
have_dynamic_loading			はい		グローバル	いいえ
have_geometry			はい		グローバル	いいえ
have_openssl			はい		グローバル	いいえ
have_profiling			はい		グローバル	いいえ
have_query_cache			はい		グローバル	いいえ
have_rtree_keys			はい		グローバル	いいえ
have_ssl			はい		グローバル	いいえ
have_statement_timeout			はい		グローバル	いいえ
have_symlink			はい		グローバル	いいえ
help	はい	はい				
histogram_generation_max_memory	はい	はい	はい		両方	はい

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
host_cache_size	はい	はい	はい		グローバル	はい
hostname			はい		グローバル	いいえ
identity			はい		セッション	はい
immediate_server_version			はい		セッション	はい
information_schema_stats_expiry	はい	はい	はい		両方	はい
init_connect	はい	はい	はい		グローバル	はい
init_file	はい	はい	はい		グローバル	いいえ
init_slave	はい	はい	はい		グローバル	はい
initialize	はい	はい				
initialize-insecure	はい	はい				
innodb	はい	はい				
innodb_adaptive_flushing	はい	はい	はい		グローバル	はい
innodb_adaptive_flushing_lwm	はい	はい	はい		グローバル	はい
innodb_adaptive_hash_index	はい	はい	はい		グローバル	はい
innodb_adaptive_hash_index_parts	はい	はい	はい		グローバル	いいえ
innodb_adaptive_max_sleep_delay	はい	はい	はい		グローバル	はい
innodb_api_bk_commit_interval	はい	はい	はい		グローバル	はい
innodb_api_disable_rowlock	はい	はい	はい		グローバル	いいえ
innodb_api_enable_binlog	はい	はい	はい		グローバル	いいえ
innodb_api_enable_mdlog	はい	はい	はい		グローバル	いいえ
innodb_api_trx_level	はい	はい	はい		グローバル	はい
innodb_autoextend_increment	はい	はい	はい		グローバル	はい
innodb_autoinc_lock_mode	はい	はい	はい		グローバル	いいえ
innodb_background_drop_list_empty	はい	はい	はい		グローバル	はい
Innodb_buffer_pool_bytes_data				はい	グローバル	いいえ
Innodb_buffer_pool_bytes_dirty				はい	グローバル	いいえ
innodb_buffer_pool_chunk_size	はい	はい	はい		グローバル	いいえ
innodb_buffer_pool_debug	はい	はい	はい		グローバル	いいえ
innodb_buffer_pool_dump_at_shutdown	はい	はい	はい		グローバル	はい
innodb_buffer_pool_dump_now	はい	はい	はい		グローバル	はい
innodb_buffer_pool_dump_pct	はい	はい	はい		グローバル	はい
Innodb_buffer_pool_dump_status				はい	グローバル	いいえ
innodb_buffer_pool_filename	はい	はい	はい		グローバル	はい
innodb_buffer_pool_in_core_file	はい	はい	はい		グローバル	はい
innodb_buffer_pool_instances	はい	はい	はい		グローバル	いいえ
innodb_buffer_pool_load_abort	はい	はい	はい		グローバル	はい
innodb_buffer_pool_load_at_startup	はい	はい	はい		グローバル	いいえ
innodb_buffer_pool_load_now	はい	はい	はい		グローバル	はい
Innodb_buffer_pool_load_status				はい	グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
InnoDB_buffer_pool_pages_data				はい	グローバル	いいえ
InnoDB_buffer_pool_pages_dirty				はい	グローバル	いいえ
InnoDB_buffer_pool_pages_flushed				はい	グローバル	いいえ
InnoDB_buffer_pool_pages_free				はい	グローバル	いいえ
InnoDB_buffer_pool_pages_latched				はい	グローバル	いいえ
InnoDB_buffer_pool_pages_misc				はい	グローバル	いいえ
InnoDB_buffer_pool_pages_total				はい	グローバル	いいえ
InnoDB_buffer_pool_read_ahead				はい	グローバル	いいえ
InnoDB_buffer_pool_read_ahead_evicted				はい	グローバル	いいえ
InnoDB_buffer_pool_read_ahead_rnd				はい	グローバル	いいえ
InnoDB_buffer_pool_read_requests				はい	グローバル	いいえ
InnoDB_buffer_pool_reads				はい	グローバル	いいえ
InnoDB_buffer_pool_resize_status				はい	グローバル	いいえ
innodb_buffer_pool_size	はい	はい	はい		グローバル	はい
InnoDB_buffer_pool_wait_free				はい	グローバル	いいえ
InnoDB_buffer_pool_write_requests				はい	グローバル	いいえ
innodb_change_buffer_max_size	はい	はい	はい		グローバル	はい
innodb_change_buffering	はい	はい	はい		グローバル	はい
innodb_change_buffering_debug	はい	はい	はい		グローバル	はい
innodb_checkpoint_disabled	はい	はい	はい		グローバル	はい
innodb_checksum_algorithm	はい	はい	はい		グローバル	はい
innodb_cmp_per_index_enabled	はい	はい	はい		グローバル	はい
innodb_commit_order	はい	はい	はい		グローバル	はい
innodb_compression_debug	はい	はい	はい		グローバル	はい
innodb_compression_failure_threshold_pct	はい	はい	はい		グローバル	はい
innodb_compression_level	はい	はい	はい		グローバル	はい
innodb_compression_pad_pct_max	はい	はい	はい		グローバル	はい
innodb_concurrency_tickets	はい	はい	はい		グローバル	はい
innodb_data_file_path	はい	はい	はい		グローバル	いいえ
InnoDB_data_fsyncs				はい	グローバル	いいえ
innodb_data_home_dir	はい	はい	はい		グローバル	いいえ
InnoDB_data_pending_fsyncs				はい	グローバル	いいえ
InnoDB_data_pending_reads				はい	グローバル	いいえ
InnoDB_data_pending_writes				はい	グローバル	いいえ
InnoDB_data_read				はい	グローバル	いいえ
InnoDB_data_reads				はい	グローバル	いいえ
InnoDB_data_writes				はい	グローバル	いいえ
InnoDB_data_written				はい	グローバル	いいえ
InnoDB_dblwr_pages_written				はい	グローバル	いいえ
InnoDB_dblwr_writes				はい	グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
innodb_ddl_log_checksum_reset_debug	はい	はい	はい		グローバル	はい
innodb_deadlock_detect	はい	はい	はい		グローバル	はい
innodb_dedicated_server	はい	はい	はい		グローバル	いいえ
innodb_default_row_format	はい	はい	はい		グローバル	はい
innodb_directories	はい	はい	はい		グローバル	いいえ
innodb_disable_sort_file_cache	はい	はい	はい		グローバル	はい
innodb_doublewrite	はい	はい	はい		グローバル	いいえ
innodb_doublewrite_batch_size	はい	はい	はい		グローバル	いいえ
innodb_doublewrite_dir	はい	はい	はい		グローバル	いいえ
innodb_doublewrite_files	はい	はい	はい		グローバル	いいえ
innodb_doublewrite_pages	はい	はい	はい		グローバル	いいえ
innodb_extend_archive_initialize	はい	はい	はい		グローバル	はい
innodb_fast_shutdown	はい	はい	はい		グローバル	はい
innodb_file_make_page_dirty_debug	はい	はい	はい		グローバル	はい
innodb_file_per_table	はい	はい	はい		グローバル	はい
innodb_fill_factor	はい	はい	はい		グローバル	はい
innodb_flush_log_timeout	はい	はい	はい		グローバル	はい
innodb_flush_log_trx_commit	はい	はい	はい		グローバル	はい
innodb_flush_method	はい	はい	はい		グローバル	いいえ
innodb_flush_neighbors	はい	はい	はい		グローバル	はい
innodb_flush_sync	はい	はい	はい		グローバル	はい
innodb_flushing_loops	はい	はい	はい		グローバル	はい
innodb_force_load_corrupted	はい	はい	はい		グローバル	いいえ
innodb_force_recovery	はい	はい	はい		グローバル	いいえ
innodb_fsync_threshold	はい	はい	はい		グローバル	はい
innodb_ft_aux_table			はい		グローバル	はい
innodb_ft_cache_size	はい	はい	はい		グローバル	いいえ
innodb_ft_enable_nag_print	はい	はい	はい		グローバル	はい
innodb_ft_enable_stopword	はい	はい	はい		両方	はい
innodb_ft_max_token_size	はい	はい	はい		グローバル	いいえ
innodb_ft_min_token_size	はい	はい	はい		グローバル	いいえ
innodb_ft_num_word_optimize	はい	はい	はい		グローバル	はい
innodb_ft_result_cache_limit	はい	はい	はい		グローバル	はい
innodb_ft_server_stopword_table	はい	はい	はい		グローバル	はい
innodb_ft_sort_pll_degree	はい	はい	はい		グローバル	いいえ
innodb_ft_total_cache_size	はい	はい	はい		グローバル	いいえ
innodb_ft_user_stopword_table	はい	はい	はい		両方	はい
InnoDB_have_atomic_builtins				はい	グローバル	いいえ
innodb_idle_flush_pct	はい	はい	はい		グローバル	はい
innodb_io_capacity	はい	はい	はい		グローバル	はい

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
innodb_io_capacity_max	はい	はい	はい		グローバル	はい
innodb_limit_optimizations	はい	はい	はい		グローバル	はい
innodb_lock_wait_timeout	はい	はい	はい		両方	はい
innodb_log_buffer_size	はい	はい	はい		グローバル	異なる
innodb_log_checkpoint_fuzzy_nowait	はい	はい	はい		グローバル	はい
innodb_log_checkpoint_nowait	はい	はい	はい		グローバル	はい
innodb_log_checkpoint_syncs	はい	はい	はい		グローバル	はい
innodb_log_compressed_pages	はい	はい	はい		グローバル	はい
innodb_log_file_size	はい	はい	はい		グローバル	いいえ
innodb_log_files_in_group	はい	はい	はい		グローバル	いいえ
innodb_log_group_home_dir	はい	はい	はい		グローバル	いいえ
innodb_log_spin_low_abs_lwm	はい	はい	はい		グローバル	はい
innodb_log_spin_low_pct_hwm	はい	はい	はい		グローバル	はい
innodb_log_wait_flush_spin_hwm	はい	はい	はい		グローバル	はい
Innodb_log_waits				はい	グローバル	いいえ
innodb_log_write_block_size	はい	はい	はい		グローバル	はい
Innodb_log_write_requests				はい	グローバル	いいえ
innodb_log_write_block_size	はい	はい	はい		グローバル	はい
Innodb_log_writes				はい	グローバル	いいえ
innodb_lru_scan_depth	はい	はい	はい		グローバル	はい
innodb_max_dirty_pages_pct	はい	はい	はい		グローバル	はい
innodb_max_dirty_pages_pct_lwm	はい	はい	はい		グローバル	はい
innodb_max_purge_lag	はい	はい	はい		グローバル	はい
innodb_max_purge_lag_delay	はい	はい	はい		グローバル	はい
innodb_max_undo_log_size	はい	はい	はい		グローバル	はい
innodb_merge_threads_hold_set_all_data	はい	はい	はい		グローバル	はい
innodb_monitor_disable	はい	はい	はい		グローバル	はい
innodb_monitor_enable	はい	はい	はい		グローバル	はい
innodb_monitor_reset	はい	はい	はい		グローバル	はい
innodb_monitor_reset_all	はい	はい	はい		グローバル	はい
Innodb_num_open_files				はい	グローバル	いいえ
innodb_numa_interleave	はい	はい	はい		グローバル	いいえ
innodb_old_block_size_pct	はい	はい	はい		グローバル	はい
innodb_old_block_size_time	はい	はい	はい		グローバル	はい
innodb_online_alter_log_max_size	はい	はい	はい		グローバル	はい
innodb_open_files_in_memory	はい	はい	はい		グローバル	いいえ
innodb_optimize_text_only	はい	はい	はい		グローバル	はい
Innodb_os_log_fsyncs				はい	グローバル	いいえ
Innodb_os_log_pending_fsyncs				はい	グローバル	いいえ
Innodb_os_log_pending_writes				はい	グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
Innodb_os_log_written				はい	グローバル	いいえ
innodb_page_cleaners	はい	はい	はい		グローバル	いいえ
Innodb_page_size				はい	グローバル	いいえ
innodb_page_size	はい	はい	はい		グローバル	いいえ
Innodb_pages_created				はい	グローバル	いいえ
Innodb_pages_read				はい	グローバル	いいえ
Innodb_pages_written				はい	グローバル	いいえ
innodb_parallel_read_threads	はい	はい	はい		セッション	はい
innodb_print_all_deadlocks	はい	はい	はい		グローバル	はい
innodb_print_ddllogs	はい	はい	はい		グローバル	はい
innodb_purge_batch_size	はい	はい	はい		グローバル	はい
innodb_purge_reuse_oldest_truncate_frequency	はい	はい	はい		グローバル	はい
innodb_purge_threads	はい	はい	はい		グローバル	いいえ
innodb_random_read_ahead	はい	はい	はい		グローバル	はい
innodb_read_ahead_threshold	はい	はい	はい		グローバル	はい
innodb_read_io_threads	はい	はい	はい		グローバル	いいえ
innodb_read_only	はい	はい	はい		グローバル	いいえ
innodb_redo_log_archive_dirs	はい	はい	はい		グローバル	はい
Innodb_redo_log_enabled				はい	グローバル	いいえ
innodb_redo_log_encrypt	はい	はい	はい		グローバル	はい
innodb_replication_delay	はい	はい	はい		グローバル	はい
innodb_rollback_timeout	はい	はい	はい		グローバル	いいえ
innodb_rollback_segments	はい	はい	はい		グローバル	はい
Innodb_row_lock_current_waits				はい	グローバル	いいえ
Innodb_row_lock_time				はい	グローバル	いいえ
Innodb_row_lock_time_avg				はい	グローバル	いいえ
Innodb_row_lock_time_max				はい	グローバル	いいえ
Innodb_row_lock_waits				はい	グローバル	いいえ
Innodb_rows_deleted				はい	グローバル	いいえ
Innodb_rows_inserted				はい	グローバル	いいえ
Innodb_rows_read				はい	グローバル	いいえ
Innodb_rows_updated				はい	グローバル	いいえ
innodb_saved_page_number_debug	はい	はい	はい		グローバル	はい
innodb_sort_buffer_size	はい	はい	はい		グローバル	いいえ
innodb_spin_wait_delay	はい	はい	はい		グローバル	はい
innodb_spin_wait_reuse_multiplier	はい	はい	はい		グローバル	はい
innodb_stats_auto_recalc	はい	はい	はい		グローバル	はい
innodb_stats_include_delete_marked	はい	はい	はい		グローバル	はい
innodb_stats_metadata	はい	はい	はい		グローバル	はい
innodb_stats_on_metadata	はい	はい	はい		グローバル	はい

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
innodb_stats_persistent	はい	はい	はい		グローバル	はい
innodb_stats_persistent_sample_pages	はい	はい	はい		グローバル	はい
innodb_stats_transient_sample_pages	はい	はい	はい		グローバル	はい
innodb-status-file	はい	はい				
innodb_status_output	はい	はい	はい		グローバル	はい
innodb_status_output_locks	はい	はい	はい		グローバル	はい
innodb_strict_mode	はい	はい	はい		両方	はい
innodb_sync_array_size	はい	はい	はい		グローバル	いいえ
innodb_sync_debug	はい	はい	はい		グローバル	いいえ
innodb_sync_spin_loops	はい	はい	はい		グローバル	はい
Innodb_system_rows_deleted				はい	グローバル	いいえ
Innodb_system_rows_inserted				はい	グローバル	いいえ
Innodb_system_rows_read				はい	グローバル	いいえ
innodb_table_locks	はい	はい	はい		両方	はい
innodb_temp_data_home_path	はい	はい	はい		グローバル	いいえ
innodb_temp_tablespace_dir	はい	はい	はい		グローバル	いいえ
innodb_thread_concurrency	はい	はい	はい		グローバル	はい
innodb_thread_stack_delay	はい	はい	はい		グローバル	はい
innodb_tmpdir	はい	はい	はい		両方	はい
Innodb_truncated_status_writes				はい	グローバル	いいえ
innodb_trx_purge_new_update_order	はい	はい	はい		グローバル	はい
innodb_trx_rseg_slots_debug	はい	はい	はい		グローバル	はい
innodb_undo_directory	はい	はい	はい		グローバル	いいえ
innodb_undo_log_encrypt	はい	はい	はい		グローバル	はい
innodb_undo_log_encrypt_innodb	はい	はい	はい		グローバル	はい
innodb_undo_tablespace	はい	はい	はい		グローバル	異なる
Innodb_undo_tablespace_active				はい	グローバル	いいえ
Innodb_undo_tablespace_explicit				はい	グローバル	いいえ
Innodb_undo_tablespace_implicit				はい	グローバル	いいえ
Innodb_undo_tablespace_total				はい	グローバル	いいえ
innodb_use_native_aio	はい	はい	はい		グローバル	いいえ
innodb_validate_tablespace_paths	はい	はい	はい		グローバル	いいえ
innodb_version			はい		グローバル	いいえ
innodb_write_io_threads	はい	はい	はい		グローバル	いいえ
insert_id			はい		セッション	はい
install	はい					
install-manual	はい					
interactive_timeout	はい	はい	はい		両方	はい
internal_tmp_disk_storage_engine	はい	はい	はい		グローバル	はい

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
internal_tmp_mem_size	はい	はい	はい		両方	はい
join_buffer_size	はい	はい	はい		両方	はい
keep_files_on_create	はい	はい	はい		両方	はい
Key_blocks_not_flushed				はい	グローバル	いいえ
Key_blocks_unused				はい	グローバル	いいえ
Key_blocks_used				はい	グローバル	いいえ
key_buffer_size	はい	はい	はい		グローバル	はい
key_cache_age_threshold	はい	はい	はい		グローバル	はい
key_cache_block_size	はい	はい	はい		グローバル	はい
key_cache_divisor_limit	はい	はい	はい		グローバル	はい
Key_read_requests				はい	グローバル	いいえ
Key_reads				はい	グローバル	いいえ
Key_write_requests				はい	グローバル	いいえ
Key_writes				はい	グローバル	いいえ
keyring_aws_cmek	はい	はい	はい		グローバル	はい
keyring_aws_conf	はい	はい	はい		グローバル	いいえ
keyring_aws_data	はい	はい	はい		グローバル	いいえ
keyring_aws_regid	はい	はい	はい		グローバル	はい
keyring_encrypted_data	はい	はい	はい		グローバル	はい
keyring_encrypted_password	はい	はい	はい		グローバル	はい
keyring_file_data	はい	はい	はい		グローバル	はい
keyring_hashicorp_auth_path	はい	はい	はい		グローバル	はい
keyring_hashicorp_ca_path	はい	はい	はい		グローバル	はい
keyring_hashicorp_caching	はい	はい	はい		グローバル	はい
keyring_hashicorp_commit_auth_path			はい		グローバル	いいえ
keyring_hashicorp_commit_ca_path			はい		グローバル	いいえ
keyring_hashicorp_commit_caching			はい		グローバル	いいえ
keyring_hashicorp_commit_role_id			はい		グローバル	いいえ
keyring_hashicorp_commit_server_url			はい		グローバル	いいえ
keyring_hashicorp_commit_store_path			はい		グローバル	いいえ
keyring_hashicorp_role_id	はい	はい	はい		グローバル	はい
keyring_hashicorp_secret_id	はい	はい	はい		グローバル	はい
keyring_hashicorp_server_url	はい	はい	はい		グローバル	はい
keyring_hashicorp_store_path	はい	はい	はい		グローバル	はい
keyring-migration-destination	はい	はい				
keyring-migration-host	はい	はい				
keyring-migration-password	はい	はい				

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
keyring-migration-port	はい	はい				
keyring-migration-socket	はい	はい				
keyring-migration-source	はい	はい				
keyring-migration-user	はい	はい				
keyring_oci_certificate	はい	はい	はい		グローバル	いいえ
keyring_oci_compartment	はい	はい	はい		グローバル	いいえ
keyring_oci_encryption_endpoint	はい	はい	はい		グローバル	いいえ
keyring_oci_key_id	はい	はい	はい		グローバル	いいえ
keyring_oci_key_fingerprint	はい	はい	はい		グローバル	いいえ
keyring_oci_management_endpoint	はい	はい	はい		グローバル	いいえ
keyring_oci_master_key	はい	はい	はい		グローバル	いいえ
keyring_oci_secret_endpoint	はい	はい	はい		グローバル	いいえ
keyring_oci_tenant_id	はい	はい	はい		グローバル	いいえ
keyring_oci_user	はい	はい	はい		グローバル	いいえ
keyring_oci_vault_endpoint	はい	はい	はい		グローバル	いいえ
keyring_oci_virtual_vault	はい	はい	はい		グローバル	いいえ
keyring_okv_conf_id	はい	はい	はい		グローバル	はい
keyring_operations			はい		グローバル	はい
language	はい	はい	はい		グローバル	いいえ
large_files_support			はい		グローバル	いいえ
large_page_size			はい		グローバル	いいえ
large_pages	はい	はい	はい		グローバル	いいえ
last_insert_id			はい		セッション	はい
Last_query_cost				はい	セッション	いいえ
Last_query_partial_plans				はい	セッション	いいえ
lc_messages	はい	はい	はい		両方	はい
lc_messages_dir	はい	はい	はい		グローバル	いいえ
lc_time_names	はい	はい	はい		両方	はい
license			はい		グローバル	いいえ
local_infile	はい	はい	はい		グローバル	はい
local-service	はい					
lock_order	はい	はい	はい		グローバル	いいえ
lock_order_debug_missing_arc	はい	はい	はい		グローバル	いいえ
lock_order_debug_missing_key	はい	はい	はい		グローバル	いいえ
lock_order_debug_missing_unlock	はい	はい	はい		グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
lock_order_dependencies	はい	はい	はい		グローバル	いいえ
lock_order_extra_dependencies	はい	はい	はい		グローバル	いいえ
lock_order_output_directory	はい	はい	はい		グローバル	いいえ
lock_order_print	はい	はい	はい		グローバル	いいえ
lock_order_trace	はい	はい	はい		グローバル	いいえ
lock_order_trace_using_arc	はい	はい	はい		グローバル	いいえ
lock_order_trace_using_key	はい	はい	はい		グローバル	いいえ
lock_order_trace_using_unlock	はい	はい	はい		グローバル	いいえ
lock_wait_timeout	はい	はい	はい		両方	はい
Locked_connects				はい	グローバル	いいえ
locked_in_memory			はい		グローバル	いいえ
log-bin	はい	はい				
log_bin			はい		グローバル	いいえ
log_bin_basename			はい		グローバル	いいえ
log_bin_index	はい	はい	はい		グローバル	いいえ
log_bin_trust_function_creators	はい	はい	はい		グローバル	はい
log_bin_use_v1_row_events	はい	はい	はい		グローバル	いいえ
log_error	はい	はい	はい		グローバル	いいえ
log_error_service	はい	はい	はい		グローバル	はい
log_error_suppression_list	はい	はい	はい		グローバル	はい
log_error_verbosity	はい	はい	はい		グローバル	はい
log-isam	はい	はい				
log_output	はい	はい	はい		グローバル	はい
log_queries_not_using_indexes	はい	はい	はい		グローバル	はい
log_raw	はい	はい	はい		グローバル	はい
log-short-format	はい	はい				
log_slave_updates	はい	はい	はい		グローバル	いいえ
log_slow_admin_statements	はい	はい	はい		グローバル	はい
log_slow_extra	はい	はい	はい		グローバル	はい
log_slow_slave_statements	はい	はい	はい		グローバル	はい
log_statements_use_safe_for_binlog	はい	はい	はい		グローバル	はい
log_syslog	はい	はい	はい		グローバル	はい
log_syslog_facility	はい	はい	はい		グローバル	はい
log_syslog_include_pid	はい	はい	はい		グローバル	はい
log_syslog_tag	はい	はい	はい		グローバル	はい
log-tc	はい	はい				
log-tc-size	はい	はい				
log_throttle_queries_not_using_indexes	はい	はい	はい		グローバル	はい
log_timestamps	はい	はい	はい		グローバル	はい

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
long_query_time	はい	はい	はい		両方	はい
low_priority_updates	はい	はい	はい		両方	はい
lower_case_file_system			はい		グローバル	いいえ
lower_case_table_names	はい	はい	はい		グローバル	いいえ
mandatory_roles	はい	はい	はい		グローバル	はい
master-info-file	はい	はい				
master_info_repository	はい	はい	はい		グローバル	はい
master-retry-count	はい	はい				
master_verify_checksum	はい	はい	はい		グローバル	はい
max_allowed_packet	はい	はい	はい		両方	はい
max_binlog_cache_size	はい	はい	はい		グローバル	はい
max-binlog-dump-events	はい	はい				
max_binlog_size	はい	はい	はい		グローバル	はい
max_binlog_stmt_cache_size	はい	はい	はい		グローバル	はい
max_connect_error	はい	はい	はい		グローバル	はい
max_connections	はい	はい	はい		グローバル	はい
max_delayed_threads	はい	はい	はい		両方	はい
max_digest_length	はい	はい	はい		グローバル	いいえ
max_error_count	はい	はい	はい		両方	はい
max_execution_time	はい	はい	はい		両方	はい
Max_execution_time_exceeded				はい	両方	いいえ
Max_execution_time_set				はい	両方	いいえ
Max_execution_time_set_failed				はい	両方	いいえ
max_heap_table_size	はい	はい	はい		両方	はい
max_insert_delayed_threads			はい		両方	はい
max_join_size	はい	はい	はい		両方	はい
max_length_for_sort_data	はい	はい	はい		両方	はい
max_points_in_geometry	はい	はい	はい		両方	はい
max_prepared_stmt_count	はい	はい	はい		グローバル	はい
max_relay_log_size	はい	はい	はい		グローバル	はい
max_seeks_for_key	はい	はい	はい		両方	はい
max_sort_length	はい	はい	はい		両方	はい
max_sp_recursion_depth	はい	はい	はい		両方	はい
Max_used_connections				はい	グローバル	いいえ
Max_used_connections_time				はい	グローバル	いいえ
max_user_connections	はい	はい	はい		両方	はい
max_write_lock_duration	はい	はい	はい		グローバル	はい
mecab_charset				はい	グローバル	いいえ
mecab_rc_file	はい	はい	はい		グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
memlock	はい	はい				
- 変数: locked_in_memory						
metadata_locks_size	はい	はい	はい		グローバル	いいえ
metadata_locks_instances	はい	はい	はい		グローバル	いいえ
min_examined_row_limit	はい	はい	はい		両方	はい
mysam-block-size	はい	はい				
mysam_data_pointer_size	はい	はい	はい		グローバル	はい
mysam_max_sort_size	はい	はい	はい		グローバル	はい
mysam_mmap_size	はい	はい	はい		グローバル	いいえ
mysam_recover_options	はい	はい	はい		グローバル	いいえ
mysam_repair_threads	はい	はい	はい		両方	はい
mysam_sort_buffer_size	はい	はい	はい		両方	はい
mysam_stats_method	はい	はい	はい		両方	はい
mysam_use_mmap	はい	はい	はい		グローバル	はい
mysql_firewall_mode	はい	はい	はい		グローバル	はい
mysql_firewall_trace	はい	はい	はい		グローバル	はい
mysql_native_password_proxy_users	はい	はい	はい		グローバル	はい
mysqlx	はい	はい				
Mysqlx_aborted_clients				はい	グローバル	いいえ
Mysqlx_address				はい	グローバル	いいえ
mysqlx_bind_address	はい	はい	はい		グローバル	いいえ
Mysqlx_bytes_received				はい	両方	いいえ
Mysqlx_bytes_received_compressed_payload				はい	両方	いいえ
Mysqlx_bytes_received_uncompressed_frame				はい	両方	いいえ
Mysqlx_bytes_sent				はい	両方	いいえ
Mysqlx_bytes_sent_compressed_payload				はい	両方	いいえ
Mysqlx_bytes_sent_uncompressed_frame				はい	両方	いいえ
Mysqlx_compression_algorithm				はい	セッション	いいえ
mysqlx_compression_algorithms	はい	はい	はい		グローバル	はい
Mysqlx_compression_level				はい	セッション	いいえ
mysqlx_connect_timeout	はい	はい	はい		グローバル	はい
Mysqlx_connection_accept_errors				はい	両方	いいえ
Mysqlx_connection_errors				はい	両方	いいえ
Mysqlx_connections_accepted				はい	グローバル	いいえ
Mysqlx_connections_closed				はい	グローバル	いいえ
Mysqlx_connections_rejected				はい	グローバル	いいえ
Mysqlx_crud_create_view				はい	両方	いいえ
Mysqlx_crud_delete				はい	両方	いいえ
Mysqlx_crud_drop_view				はい	両方	いいえ

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
Mysqlx_crud_find				はい	両方	いいえ
Mysqlx_crud_insert				はい	両方	いいえ
Mysqlx_crud_modify_view				はい	両方	いいえ
Mysqlx_crud_update				はい	両方	いいえ
Mysqlx_cursor_close				はい	両方	いいえ
Mysqlx_cursor_fetch				はい	両方	いいえ
Mysqlx_cursor_open				はい	両方	いいえ
mysqlx_deflate_default_compression_level	はい	はい	はい		グローバル	はい
mysqlx_deflate_client_compression_level	はい	はい	はい		グローバル	はい
mysqlx_document_id_unique_prefix	はい	はい	はい		グローバル	はい
mysqlx_enable_hidden_notice	はい	はい	はい		グローバル	はい
Mysqlx_errors_sent				はい	両方	いいえ
Mysqlx_errors_unknown_message_type				はい	両方	いいえ
Mysqlx_expect_close				はい	両方	いいえ
Mysqlx_expect_open				はい	両方	いいえ
mysqlx_idle_worker_thread_timeout	はい	はい	はい		グローバル	はい
Mysqlx_init_error				はい	両方	いいえ
mysqlx_interactive_timeout	はい	はい	はい		グローバル	はい
mysqlx_lz4_default_compression_level	はい	はい	はい		グローバル	はい
mysqlx_lz4_max_client_compression_level	はい	はい	はい		グローバル	はい
mysqlx_max_allowed_packet	はい	はい	はい		グローバル	はい
mysqlx_max_connections	はい	はい	はい		グローバル	はい
Mysqlx_messages_sent				はい	両方	いいえ
mysqlx_min_worker_threads	はい	はい	はい		グローバル	はい
Mysqlx_notice_global_sent				はい	両方	いいえ
Mysqlx_notice_other_sent				はい	両方	いいえ
Mysqlx_notice_warning_sent				はい	両方	いいえ
Mysqlx_notified_by_group_replication				はい	両方	いいえ
Mysqlx_port				はい	グローバル	いいえ
mysqlx_port	はい	はい	はい		グローバル	いいえ
mysqlx_port_operation_timeout	はい	はい	はい		グローバル	いいえ
Mysqlx_prep_deallocate				はい	両方	いいえ
Mysqlx_prep_execute				はい	両方	いいえ
Mysqlx_prep_prepare				はい	両方	いいえ
mysqlx_read_timeout	はい	はい	はい		セッション	はい
Mysqlx_rows_sent				はい	両方	いいえ
Mysqlx_sessions				はい	グローバル	いいえ
Mysqlx_sessions_accepted				はい	グローバル	いいえ
Mysqlx_sessions_closed				はい	グローバル	いいえ
Mysqlx_sessions_fatal_error				はい	グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
Mysqlx_sessions_killed				はい	グローバル	いいえ
Mysqlx_sessions_rejected				はい	グローバル	いいえ
Mysqlx_socket				はい	グローバル	いいえ
mysqlx_socket	はい	はい	はい		グローバル	いいえ
Mysqlx_ssl_accept_renegotiates				はい	グローバル	いいえ
Mysqlx_ssl_accepts				はい	グローバル	いいえ
Mysqlx_ssl_active				はい	両方	いいえ
mysqlx_ssl_ca	はい	はい	はい		グローバル	いいえ
mysqlx_ssl_capabilities	はい	はい	はい		グローバル	いいえ
mysqlx_ssl_cert	はい	はい	はい		グローバル	いいえ
Mysqlx_ssl_cipher				はい	両方	いいえ
mysqlx_ssl_cipher_list	はい	はい	はい		グローバル	いいえ
Mysqlx_ssl_cipher_list				はい	両方	いいえ
mysqlx_ssl_crl	はい	はい	はい		グローバル	いいえ
mysqlx_ssl_crlpath	はい	はい	はい		グローバル	いいえ
Mysqlx_ssl_ctx_verify_depth				はい	両方	いいえ
Mysqlx_ssl_ctx_verify_mode				はい	両方	いいえ
Mysqlx_ssl_finished_accepts				はい	グローバル	いいえ
mysqlx_ssl_key	はい	はい	はい		グローバル	いいえ
Mysqlx_ssl_server_not_after				はい	グローバル	いいえ
Mysqlx_ssl_server_not_before				はい	グローバル	いいえ
Mysqlx_ssl_verify_depth				はい	グローバル	いいえ
Mysqlx_ssl_verify_mode				はい	グローバル	いいえ
Mysqlx_ssl_version				はい	両方	いいえ
Mysqlx_stmt_create_collection				はい	両方	いいえ
Mysqlx_stmt_create_collection_index				はい	両方	いいえ
Mysqlx_stmt_disable_notices				はい	両方	いいえ
Mysqlx_stmt_drop_collection				はい	両方	いいえ
Mysqlx_stmt_drop_collection_index				はい	両方	いいえ
Mysqlx_stmt_enable_notices				はい	両方	いいえ
Mysqlx_stmt_ensure_collection				はい	両方	いいえ
Mysqlx_stmt_execute_mysqlx				はい	両方	いいえ
Mysqlx_stmt_execute_sql				はい	両方	いいえ
Mysqlx_stmt_execute_xplugin				はい	両方	いいえ
Mysqlx_stmt_get_collection_options				はい	両方	いいえ
Mysqlx_stmt_kill_client				はい	両方	いいえ
Mysqlx_stmt_list_clients				はい	両方	いいえ
Mysqlx_stmt_list_notices				はい	両方	いいえ
Mysqlx_stmt_list_objects				はい	両方	いいえ
Mysqlx_stmt_modify_collection_options				はい	両方	いいえ

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
Mysqlx_stmt_ping				はい	両方	いいえ
mysqlx_wait_timeout	はい	はい	はい		セッション	はい
Mysqlx_worker_threads				はい	グローバル	いいえ
Mysqlx_worker_threads_active				はい	グローバル	いいえ
mysqlx_write_timeout	はい	はい	はい		セッション	はい
mysqlx_zstd_deflate_compression_level	はい	はい	はい		グローバル	はい
mysqlx_zstd_max_inflight_compression_level	はい	はい	はい		グローバル	はい
named_pipe	はい	はい	はい		グローバル	いいえ
named_pipe_full_access_group	はい	はい	はい		グローバル	いいえ
ndb_allow_copying	はい	はい	はい		両方	はい
Ndb_api_adaptive_send_deferred_count				はい	グローバル	いいえ
Ndb_api_adaptive_send_deferred_count_replica				はい	グローバル	いいえ
Ndb_api_adaptive_send_deferred_count_session				はい	グローバル	いいえ
Ndb_api_adaptive_send_deferred_count_slave				はい	グローバル	いいえ
Ndb_api_adaptive_send_forced_count				はい	グローバル	いいえ
Ndb_api_adaptive_send_forced_count_replica				はい	グローバル	いいえ
Ndb_api_adaptive_send_forced_count_session				はい	グローバル	いいえ
Ndb_api_adaptive_send_forced_count_slave				はい	グローバル	いいえ
Ndb_api_adaptive_send_unforced_count				はい	グローバル	いいえ
Ndb_api_adaptive_send_unforced_count_replica				はい	グローバル	いいえ
Ndb_api_adaptive_send_unforced_count_session				はい	グローバル	いいえ
Ndb_api_adaptive_send_unforced_count_slave				はい	グローバル	いいえ
Ndb_api_bytes_received_count				はい	グローバル	いいえ
Ndb_api_bytes_received_count_replica				はい	グローバル	いいえ
Ndb_api_bytes_received_count_session				はい	セッション	いいえ
Ndb_api_bytes_received_count_slave				はい	グローバル	いいえ
Ndb_api_bytes_sent_count				はい	グローバル	いいえ
Ndb_api_bytes_sent_count_replica				はい	グローバル	いいえ
Ndb_api_bytes_sent_count_session				はい	セッション	いいえ
Ndb_api_bytes_sent_count_slave				はい	グローバル	いいえ
Ndb_api_event_bytes_count				はい	グローバル	いいえ
Ndb_api_event_bytes_count_injector				はい	グローバル	いいえ
Ndb_api_event_data_count				はい	グローバル	いいえ
Ndb_api_event_data_count_injector				はい	グローバル	いいえ
Ndb_api_event_nondata_count				はい	グローバル	いいえ
Ndb_api_event_nondata_count_injector				はい	グローバル	いいえ
Ndb_api_pk_op_count				はい	グローバル	いいえ
Ndb_api_pk_op_count_replica				はい	グローバル	いいえ
Ndb_api_pk_op_count_session				はい	セッション	いいえ
Ndb_api_pk_op_count_slave				はい	グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
Ndb_api_pruned_scan_count				はい	グローバル	いいえ
Ndb_api_pruned_scan_count_replica				はい	グローバル	いいえ
Ndb_api_pruned_scan_count_session				はい	セッション	いいえ
Ndb_api_pruned_scan_count_slave				はい	グローバル	いいえ
Ndb_api_range_scan_count				はい	グローバル	いいえ
Ndb_api_range_scan_count_replica				はい	グローバル	いいえ
Ndb_api_range_scan_count_session				はい	セッション	いいえ
Ndb_api_range_scan_count_slave				はい	グローバル	いいえ
Ndb_api_read_row_count				はい	グローバル	いいえ
Ndb_api_read_row_count_replica				はい	グローバル	いいえ
Ndb_api_read_row_count_session				はい	セッション	いいえ
Ndb_api_read_row_count_slave				はい	グローバル	いいえ
Ndb_api_scan_batch_count				はい	グローバル	いいえ
Ndb_api_scan_batch_count_replica				はい	グローバル	いいえ
Ndb_api_scan_batch_count_session				はい	セッション	いいえ
Ndb_api_scan_batch_count_slave				はい	グローバル	いいえ
Ndb_api_table_scan_count				はい	グローバル	いいえ
Ndb_api_table_scan_count_replica				はい	グローバル	いいえ
Ndb_api_table_scan_count_session				はい	セッション	いいえ
Ndb_api_table_scan_count_slave				はい	グローバル	いいえ
Ndb_api_trans_abort_count				はい	グローバル	いいえ
Ndb_api_trans_abort_count_replica				はい	グローバル	いいえ
Ndb_api_trans_abort_count_session				はい	セッション	いいえ
Ndb_api_trans_abort_count_slave				はい	グローバル	いいえ
Ndb_api_trans_close_count				はい	グローバル	いいえ
Ndb_api_trans_close_count_replica				はい	グローバル	いいえ
Ndb_api_trans_close_count_session				はい	セッション	いいえ
Ndb_api_trans_close_count_slave				はい	グローバル	いいえ
Ndb_api_trans_commit_count				はい	グローバル	いいえ
Ndb_api_trans_commit_count_replica				はい	グローバル	いいえ
Ndb_api_trans_commit_count_session				はい	セッション	いいえ
Ndb_api_trans_commit_count_slave				はい	グローバル	いいえ
Ndb_api_trans_local_read_row_count				はい	グローバル	いいえ
Ndb_api_trans_local_read_row_count_replica				はい	グローバル	いいえ
Ndb_api_trans_local_read_row_count_session				はい	セッション	いいえ
Ndb_api_trans_local_read_row_count_slave				はい	グローバル	いいえ
Ndb_api_trans_start_count				はい	グローバル	いいえ
Ndb_api_trans_start_count_replica				はい	グローバル	いいえ
Ndb_api_trans_start_count_session				はい	セッション	いいえ
Ndb_api_trans_start_count_slave				はい	グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
Ndb_api_uk_op_count				はい	グローバル	いいえ
Ndb_api_uk_op_count_replica				はい	グローバル	いいえ
Ndb_api_uk_op_count_session				はい	セッション	いいえ
Ndb_api_uk_op_count_slave				はい	グローバル	いいえ
Ndb_api_wait_exec_complete_count				はい	グローバル	いいえ
Ndb_api_wait_exec_complete_count_replica				はい	グローバル	いいえ
Ndb_api_wait_exec_complete_count_session				はい	セッション	いいえ
Ndb_api_wait_exec_complete_count_slave				はい	グローバル	いいえ
Ndb_api_wait_meta_request_count				はい	グローバル	いいえ
Ndb_api_wait_meta_request_count_replica				はい	グローバル	いいえ
Ndb_api_wait_meta_request_count_session				はい	セッション	いいえ
Ndb_api_wait_meta_request_count_slave				はい	グローバル	いいえ
Ndb_api_wait_nanos_count				はい	グローバル	いいえ
Ndb_api_wait_nanos_count_replica				はい	グローバル	いいえ
Ndb_api_wait_nanos_count_session				はい	セッション	いいえ
Ndb_api_wait_nanos_count_slave				はい	グローバル	いいえ
Ndb_api_wait_scan_result_count				はい	グローバル	いいえ
Ndb_api_wait_scan_result_count_replica				はい	グローバル	いいえ
Ndb_api_wait_scan_result_count_session				はい	セッション	いいえ
Ndb_api_wait_scan_result_count_slave				はい	グローバル	いいえ
ndb_autoincrement_prefetch_sz	はい	はい	はい		両方	はい
ndb_batch_size	はい	はい	はい		グローバル	いいえ
ndb_blob_read_block_size	はい	はい	はい		両方	はい
ndb_blob_write_block_size	はい	はい	はい		両方	はい
ndb_cache_check_time	はい	はい	はい		グローバル	はい
ndb_clear_apply_time	はい		はい		グローバル	はい
ndb_cluster_connection_pool	はい	はい	はい		グローバル	いいえ
ndb_cluster_connection_pool_node_id	はい	はい	はい		グローバル	いいえ
Ndb_cluster_node_id				はい	グローバル	いいえ
Ndb_config_from_host				はい	両方	いいえ
Ndb_config_from_port				はい	両方	いいえ
Ndb_config_generation				はい	グローバル	いいえ
Ndb_conflict_fn_epoch				はい	グローバル	いいえ
Ndb_conflict_fn_epoch_trans				はい	グローバル	いいえ
Ndb_conflict_fn_epoch2				はい	グローバル	いいえ
Ndb_conflict_fn_epoch2_trans				はい	グローバル	いいえ
Ndb_conflict_fn_max				はい	グローバル	いいえ
Ndb_conflict_fn_old				はい	グローバル	いいえ
Ndb_conflict_last_conflict_epoch			はい		グローバル	いいえ
Ndb_conflict_last_stable_epoch				はい	グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
Ndb_conflict_reflected_op_discard_count				はい	グローバル	いいえ
Ndb_conflict_reflected_op_prepare_count				はい	グローバル	いいえ
Ndb_conflict_refresh_op_count				はい	グローバル	いいえ
ndb_conflict_role	はい	はい	はい		グローバル	はい
Ndb_conflict_trans_conflict_commit_count				はい	グローバル	いいえ
Ndb_conflict_trans_detect_iter_count				はい	グローバル	いいえ
Ndb_conflict_trans_reject_count				はい	グローバル	いいえ
Ndb_conflict_trans_row_conflict_count				はい	グローバル	いいえ
Ndb_conflict_trans_row_reject_count				はい	グローバル	いいえ
ndb-connectstring	はい	はい				
ndb_data_node_neighbour	はい	はい	はい		グローバル	はい
ndb_dbg_check_files	はい	はい	はい		両方	はい
ndb_default_column_format	はい	はい	はい		グローバル	はい
ndb_default_column_format	はい	はい	はい		グローバル	はい
ndb_deferred_constraints	はい	はい	はい		両方	はい
ndb_deferred_constraints	はい	はい	はい		両方	はい
ndb_distribution	はい	はい	はい		グローバル	はい
ndb_distribution	はい	はい	はい		グローバル	はい
Ndb_epoch_delete_delete_count				はい	グローバル	いいえ
ndb_eventbuffer_size_percent	はい	はい	はい		グローバル	はい
ndb_eventbuffer_max_alloc	はい	はい	はい		グローバル	はい
Ndb_execute_count				はい	グローバル	いいえ
ndb_extra_logging	はい	はい	はい		グローバル	はい
ndb_force_send	はい	はい	はい		両方	はい
ndb_fully_replicated	はい	はい	はい		両方	はい
ndb_index_stat_enable	はい	はい	はい		両方	はい
ndb_index_stat_optimize	はい	はい	はい		両方	はい
ndb_join_pushdown			はい		両方	はい
Ndb_last_commit_epoch_server				はい	グローバル	いいえ
Ndb_last_commit_epoch_session				はい	セッション	いいえ
ndb_log_apply_status	はい	はい	はい		グローバル	いいえ
ndb_log_apply_status	はい	はい	はい		グローバル	いいえ
ndb_log_bin	はい		はい		両方	はい
ndb_log_binlog_index	はい		はい		グローバル	はい
ndb_log_empty_epochs	はい	はい	はい		グローバル	はい
ndb_log_empty_epochs	はい	はい	はい		グローバル	はい
ndb_log_empty_update	はい	はい	はい		グローバル	はい
ndb_log_empty_update	はい	はい	はい		グローバル	はい
ndb_log_exclusive_leads	はい	はい	はい		両方	はい

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
ndb_log_exclusive_reads	はい	はい	はい		両方	はい
ndb_log_fail_terminate	はい	はい	はい		グローバル	いいえ
ndb_log_orig	はい	はい	はい		グローバル	いいえ
ndb_log_orig	はい	はい	はい		グローバル	いいえ
ndb_log_transaction_id	はい	はい	はい		グローバル	いいえ
ndb_log_transaction_id			はい		グローバル	いいえ
ndb_log_update_write	はい	はい	はい		グローバル	はい
ndb_log_update_format	はい	はい	はい		グローバル	はい
ndb_log_updated_only	はい	はい	はい		グローバル	はい
Ndb_metadata_blacklist_size				はい	グローバル	いいえ
ndb_metadata_checksum	はい	はい	はい		グローバル	はい
ndb_metadata_checksum_interval	はい	はい	はい		グローバル	はい
Ndb_metadata_detected_count				はい	グローバル	いいえ
Ndb_metadata_excluded_count				はい	グローバル	いいえ
ndb_metadata_sync			はい		グローバル	はい
Ndb_metadata_synced_count				はい	グローバル	いいえ
ndb-mgmd-host	はい	はい				
ndb_nodeid	はい	はい		はい	グローバル	いいえ
Ndb_number_of_data_nodes				はい	グローバル	いいえ
ndb_optimization_delay	はい	はい	はい		グローバル	はい
ndb_optimized_node_selection	はい	はい	はい		グローバル	いいえ
Ndb_pruned_scan_count				はい	グローバル	いいえ
Ndb_pushed_queries_defined				はい	グローバル	いいえ
Ndb_pushed_queries_dropped				はい	グローバル	いいえ
Ndb_pushed_queries_executed				はい	グローバル	いいえ
Ndb_pushed_reads				はい	グローバル	いいえ
ndb_read_backup	はい	はい	はい		グローバル	はい
ndb_recv_thread_activation_threshold	はい	はい	はい		グローバル	はい
ndb_recv_thread_mask	はい	はい	はい		グローバル	はい
Ndb_replica_max_replicated_epoch			はい		グローバル	いいえ
ndb_report_thread_log_epoch	はい	はい	はい		グローバル	はい
ndb_report_thread_log_mem_usage	はい	はい	はい		グローバル	はい
ndb_row_checksum			はい		両方	はい
Ndb_scan_count				はい	グローバル	いいえ
ndb_schema_disk_wait_timeout	はい	はい	はい		グローバル	はい
ndb_schema_disk_timeout	はい	はい	はい		グローバル	いいえ
ndb_schema_disk_timeout	はい	はい	はい		グローバル	いいえ
ndb_schema_disk_upgrade_allowed	はい	はい	はい		グローバル	いいえ
ndb_show_foreign_key_mock_tables	はい	はい	はい		グローバル	はい
ndb_slave_conflict_resolution	はい	はい	はい		グローバル	はい

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
Ndb_slave_max_replicated_epoch			はい		グローバル	いいえ
Ndb_system_name			はい		グローバル	いいえ
ndb_table_no_logging			はい		セッション	はい
ndb_table_temporary			はい		セッション	はい
Ndb_trans_hint_count_session				はい	両方	いいえ
ndb-transid-mysql-connection-map	はい					
ndb_use_copying_alter_table			はい		両方	いいえ
ndb_use_exact_count			はい		両方	はい
ndb_use_transactions	はい	はい	はい		両方	はい
ndb_version			はい		グローバル	いいえ
ndb_version_string			はい		グローバル	いいえ
ndb_wait_connect	はい	はい	はい		グローバル	いいえ
ndb_wait_setup	はい	はい	はい		グローバル	いいえ
ndbcluster	はい	はい				
ndbinfo	はい					
ndbinfo_database			はい		グローバル	いいえ
ndbinfo_max_bytes	はい		はい		両方	はい
ndbinfo_max_rows	はい		はい		両方	はい
ndbinfo_offline			はい		グローバル	はい
ndbinfo_show_hidden	はい		はい		両方	はい
ndbinfo_table_prefix			はい		グローバル	いいえ
ndbinfo_version			はい		グローバル	いいえ
net_buffer_length	はい	はい	はい		両方	はい
net_read_timeout	はい	はい	はい		両方	はい
net_retry_count	はい	はい	はい		両方	はい
net_write_timeout	はい	はい	はい		両方	はい
new	はい	はい	はい		両方	はい
ngram_token_size	はい	はい	はい		グローバル	いいえ
no-dd-upgrade	はい	はい				
no-defaults	はい					
no-monitor	はい	はい				
Not_flushed_delayed_rows				はい	グローバル	いいえ
offline_mode	はい	はい	はい		グローバル	はい
old	はい	はい	はい		グローバル	いいえ
old_alter_table	はい	はい	はい		両方	はい
old-style-user-limits	はい	はい				
Ongoing_anonymous_gtid_violating_transaction_count				はい	グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
Ongoing_anonymous_transaction_count				はい	グローバル	いいえ
Ongoing_automatic_gtid_violating_transaction_count				はい	グローバル	いいえ
Open_files				はい	グローバル	いいえ
open_files_limit	はい	はい	はい		グローバル	いいえ
Open_streams				はい	グローバル	いいえ
Open_table_definitions				はい	グローバル	いいえ
Open_tables				はい	両方	いいえ
Opened_files				はい	グローバル	いいえ
Opened_table_definitions				はい	両方	いいえ
Opened_tables				はい	両方	いいえ
optimizer_prune_level	はい	はい	はい		両方	はい
optimizer_search_depth	はい	はい	はい		両方	はい
optimizer_switch	はい	はい	はい		両方	はい
optimizer_trace	はい	はい	はい		両方	はい
optimizer_trace_features	はい	はい	はい		両方	はい
optimizer_trace_level	はい	はい	はい		両方	はい
optimizer_trace_max_mem_size	はい	はい	はい		両方	はい
optimizer_trace_options	はい	はい	はい		両方	はい
original_commit_timestamp			はい		セッション	はい
original_server_version			はい		セッション	はい
parser_max_memory_size	はい	はい	はい		両方	はい
partial_revokes	はい	はい	はい		グローバル	はい
password_history	はい	はい	はい		グローバル	はい
password_require_current	はい	はい	はい		グローバル	はい
password_reuse_interval	はい	はい	はい		グローバル	はい
performance_schema	はい	はい	はい		グローバル	いいえ
Performance_schema_accounts_lost				はい	グローバル	いいえ
performance_schema_accounts_size	はい	はい	はい		グローバル	いいえ
Performance_schema_cond_classes_lost				はい	グローバル	いいえ
Performance_schema_cond_instances_lost				はい	グローバル	いいえ
performance-schema-consumer-events-stages-current	はい	はい				
performance-schema-consumer-events-stages-history	はい	はい				
performance-schema-consumer-	はい	はい				

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
events-stages-history-long						
performance-schema-consumer-events-statements-current	はい	はい				
performance-schema-consumer-events-statements-history	はい	はい				
performance-schema-consumer-events-statements-history-long	はい	はい				
performance-schema-consumer-events-transactions-current	はい	はい				
performance-schema-consumer-events-transactions-history	はい	はい				
performance-schema-consumer-events-transactions-history-long	はい	はい				
performance-schema-consumer-events-waits-current	はい	はい				
performance-schema-consumer-events-waits-history	はい	はい				
performance-schema-consumer-events-waits-history-long	はい	はい				

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
performance-schema-consumer-global-instrumentation	はい	はい				
performance-schema-consumer-statements-digest	はい	はい				
performance-schema-consumer-thread-instrumentation	はい	はい				
Performance_schema_digest_lost				はい	グローバル	いいえ
performance_schema_digests_size	はい	はい	はい		グローバル	いいえ
performance_schema_error_size	はい	はい	はい		グローバル	いいえ
performance_schema_events_stages_history_long	はい	はい	はい		グローバル	いいえ
performance_schema_events_stages_history_size	はい	はい	はい		グローバル	いいえ
performance_schema_events_stages_history_size	はい	はい	はい		グローバル	いいえ
performance_schema_events_stages_history_size	はい	はい	はい		グローバル	いいえ
performance_schema_events_transactions_history_long	はい	はい	はい		グローバル	いいえ
performance_schema_events_transactions_history_size	はい	はい	はい		グローバル	いいえ
performance_schema_events_transactions_history_size	はい	はい	はい		グローバル	いいえ
performance_schema_events_waits_history_long	はい	はい	はい		グローバル	いいえ
performance_schema_events_waits_history_size	はい	はい	はい		グローバル	いいえ
Performance_schema_file_classes_lost				はい	グローバル	いいえ
Performance_schema_file_handles_lost				はい	グローバル	いいえ
Performance_schema_file_instances_lost				はい	グローバル	いいえ
Performance_schema_hosts_lost				はい	グローバル	いいえ
performance_schema_hosts_size	はい	はい	はい		グローバル	いいえ
Performance_schema_index_statistics_lost				はい	グローバル	いいえ
performance-schema-instrument	はい	はい				
Performance_schema_locker_lost				はい	グローバル	いいえ
performance_schema_max_conditional_instances	はい	はい	はい		グローバル	いいえ
performance_schema_max_conditional_instances	はい	はい	はい		グローバル	いいえ
performance_schema_max_digest_length	はい	はい	はい		グローバル	いいえ
performance_schema_max_digest_sample_age	はい	はい	はい		グローバル	はい
performance_schema_max_file_classes	はい	はい	はい		グローバル	いいえ
performance_schema_max_file_handles	はい	はい	はい		グローバル	いいえ
performance_schema_max_file_instances	はい	はい	はい		グローバル	いいえ
performance_schema_max_index_statistics	はい	はい	はい		グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
performance_schema_max_memory_classes	はい	はい	はい		グローバル	いいえ
performance_schema_max_metadata_locks	はい	はい	はい		グローバル	いいえ
performance_schema_max_mutex_classes	はい	はい	はい		グローバル	いいえ
performance_schema_max_mutex_instances	はい	はい	はい		グローバル	いいえ
performance_schema_max_prepared_statements_instances	はい	はい	はい		グローバル	いいえ
performance_schema_max_program_instances	はい	はい	はい		グローバル	いいえ
performance_schema_max_rwlock_classes	はい	はい	はい		グローバル	いいえ
performance_schema_max_rwlock_instances	はい	はい	はい		グローバル	いいえ
performance_schema_max_socket_classes	はい	はい	はい		グローバル	いいえ
performance_schema_max_socket_instances	はい	はい	はい		グローバル	いいえ
performance_schema_max_sql_text_length	はい	はい	はい		グローバル	いいえ
performance_schema_max_stage_classes	はい	はい	はい		グローバル	いいえ
performance_schema_max_statement_classes	はい	はい	はい		グローバル	いいえ
performance_schema_max_statement_stack	はい	はい	はい		グローバル	いいえ
performance_schema_max_table_handles	はい	はい	はい		グローバル	いいえ
performance_schema_max_table_instances	はい	はい	はい		グローバル	いいえ
performance_schema_max_table_lock_stat	はい	はい	はい		グローバル	いいえ
performance_schema_max_thread_classes	はい	はい	はい		グローバル	いいえ
performance_schema_max_thread_instances	はい	はい	はい		グローバル	いいえ
Performance_schema_memory_classes_lost				はい	グローバル	いいえ
Performance_schema_metadata_lock_lost				はい	グローバル	いいえ
Performance_schema_mutex_classes_lost				はい	グローバル	いいえ
Performance_schema_mutex_instances_lost				はい	グローバル	いいえ
Performance_schema_nested_statement_lost				はい	グローバル	いいえ
Performance_schema_prepared_statements_lost				はい	グローバル	いいえ
Performance_schema_program_lost				はい	グローバル	いいえ
Performance_schema_rwlock_classes_lost				はい	グローバル	いいえ
Performance_schema_rwlock_instances_lost				はい	グローバル	いいえ
Performance_schema_session_connect_attrs_longest_seen				はい	グローバル	いいえ
Performance_schema_session_connect_attrs_lost				はい	グローバル	いいえ
performance_schema_session_connect_attrs_size	はい	はい	はい		グローバル	いいえ
performance_schema_setup_actor_size	はい	はい	はい		グローバル	いいえ
performance_schema_setup_object_size	はい	はい	はい		グローバル	いいえ
performance_schema_show_procedure_list	はい	はい	はい		グローバル	はい
Performance_schema_socket_classes_lost				はい	グローバル	いいえ
Performance_schema_socket_instances_lost				はい	グローバル	いいえ
Performance_schema_stage_classes_lost				はい	グローバル	いいえ
Performance_schema_statement_classes_lost				はい	グローバル	いいえ
Performance_schema_table_handles_lost				はい	グローバル	いいえ
Performance_schema_table_instances_lost				はい	グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
Performance_schema_table_lock_stat_lost				はい	グローバル	いいえ
Performance_schema_thread_classes_lost				はい	グローバル	いいえ
Performance_schema_thread_instances_lost				はい	グローバル	いいえ
Performance_schema_users_lost				はい	グローバル	いいえ
performance_schema_users_size	はい	はい	はい		グローバル	いいえ
persist_only_admin_509_subject	はい	はい	はい		グローバル	いいえ
persisted_globals_load	はい	はい	はい		グローバル	いいえ
pid_file	はい	はい	はい		グローバル	いいえ
plugin_dir	はい	はい	はい		グローバル	いいえ
plugin_load	はい	はい	はい		グローバル	いいえ
plugin_load_add	はい	はい	はい		グローバル	いいえ
plugin-xxx	はい	はい				
port	はい	はい	はい		グローバル	いいえ
port-open-timeout	はい	はい				
preload_buffer_size	はい	はい	はい		両方	はい
Prepared_stmt_count				はい	グローバル	いいえ
print-defaults	はい					
print_identified_with_as_hex	はい	はい	はい		両方	はい
profiling			はい		両方	はい
profiling_history_size	はい	はい	はい		両方	はい
protocol_compression_algorithms	はい	はい	はい		グローバル	はい
protocol_version			はい		グローバル	いいえ
proxy_user			はい		セッション	いいえ
pseudo_slave_mode			はい		セッション	はい
pseudo_thread_id			はい		セッション	はい
Queries				はい	両方	いいえ
query_alloc_block_size	はい	はい	はい		両方	はい
query_prealloc_size	はい	はい	はい		両方	はい
Questions				はい	両方	いいえ
rand_seed1			はい		セッション	はい
rand_seed2			はい		セッション	はい
range_alloc_block_size	はい	はい	はい		両方	はい
range_optimizer_max_mem_size	はい	はい	はい		両方	はい
rbr_exec_mode			はい		両方	はい
read_buffer_size	はい	はい	はい		両方	はい
read_only	はい	はい	はい		グローバル	はい
read_rnd_buffer_size	はい	はい	はい		両方	はい
regexp_stack_limit	はい	はい	はい		グローバル	はい
regexp_time_limit	はい	はい	はい		グローバル	はい

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
relay_log	はい	はい	はい		グローバル	いいえ
relay_log_basename			はい		グローバル	いいえ
relay_log_index	はい	はい	はい		グローバル	いいえ
relay_log_info_file	はい	はい	はい		グローバル	いいえ
relay_log_info_repository	はい	はい	はい		グローバル	はい
relay_log_purge	はい	はい	はい		グローバル	はい
relay_log_recovery	はい	はい	はい		グローバル	いいえ
relay_log_space_limit	はい	はい	はい		グローバル	いいえ
remove	はい					
replicate-do-db	はい	はい				
replicate-do-table	はい	はい				
replicate-ignore-db	はい	はい				
replicate-ignore-table	はい	はい				
replicate-rewrite-db	はい	はい				
replicate-same-server-id	はい	はい				
replicate-wild-do-table	はい	はい				
replicate-wild-ignore-table	はい	はい				
replication_optimize_for_static_plugins	はい	はい	はい		グローバル	はい
replication_send_logs_observes_only	はい	はい	はい		グローバル	はい
report_host	はい	はい	はい		グローバル	いいえ
report_password	はい	はい	はい		グローバル	いいえ
report_port	はい	はい	はい		グローバル	いいえ
report_user	はい	はい	はい		グローバル	いいえ
require_row_format			はい		セッション	はい
require_secure_transport	はい	はい	はい		グローバル	はい
resultset_metadata			はい		セッション	はい
rewriter_enabled			はい		グローバル	はい
Rewriter_number_loaded_rules				はい	グローバル	いいえ
Rewriter_number_reloads				はい	グローバル	いいえ
Rewriter_number_rewritten_queries				はい	グローバル	いいえ
Rewriter_reload_error				はい	グローバル	いいえ
rewriter_verbose			はい		グローバル	はい
rpl_read_size	はい	はい	はい		グローバル	はい
Rpl_semi_sync_master_clients				はい	グローバル	いいえ
rpl_semi_sync_master_enabled	はい	はい	はい		グローバル	はい

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
Rpl_semi_sync_master_net_avg_wait_time				はい	グローバル	いいえ
Rpl_semi_sync_master_net_wait_time				はい	グローバル	いいえ
Rpl_semi_sync_master_net_waits				はい	グローバル	いいえ
Rpl_semi_sync_master_no_times				はい	グローバル	いいえ
Rpl_semi_sync_master_no_tx				はい	グローバル	いいえ
Rpl_semi_sync_master_status				はい	グローバル	いいえ
Rpl_semi_sync_master_timefunc_failures				はい	グローバル	いいえ
rpl_semi_sync_master_timeout	はい	はい	はい		グローバル	はい
rpl_semi_sync_master_trace_level	はい	はい	はい		グローバル	はい
Rpl_semi_sync_master_tx_avg_wait_time				はい	グローバル	いいえ
Rpl_semi_sync_master_tx_wait_time				はい	グローバル	いいえ
Rpl_semi_sync_master_tx_waits				はい	グローバル	いいえ
rpl_semi_sync_master_wait_for_slave_count	はい	はい	はい		グローバル	はい
rpl_semi_sync_master_wait_no_slave	はい	はい	はい		グローバル	はい
rpl_semi_sync_master_wait_point	はい	はい	はい		グローバル	はい
Rpl_semi_sync_master_wait_pos_backtraverse				はい	グローバル	いいえ
Rpl_semi_sync_master_wait_sessions				はい	グローバル	いいえ
Rpl_semi_sync_master_yes_tx				はい	グローバル	いいえ
rpl_semi_sync_slave_enabled	はい	はい	はい		グローバル	はい
Rpl_semi_sync_slave_status				はい	グローバル	いいえ
rpl_semi_sync_slave_trace_level	はい	はい	はい		グローバル	はい
rpl_stop_slave_timeout	はい	はい	はい		グローバル	はい
Rsa_public_key				はい	グローバル	いいえ
safe-user-create	はい	はい				
schema_definition_cache	はい	はい	はい		グローバル	はい
secondary_engine_cost_threshold			はい		セッション	はい
Secondary_engine_execution_count				はい	両方	いいえ
secure_file_priv	はい	はい	はい		グローバル	いいえ
Select_full_join				はい	両方	いいえ
Select_full_range_join				はい	両方	いいえ
select_into_buffer_size	はい	はい	はい		両方	はい
select_into_disk_cache	はい	はい	はい		両方	はい
select_into_disk_cache_delay	はい	はい	はい		両方	はい
Select_range				はい	両方	いいえ
Select_range_check				はい	両方	いいえ
Select_scan				はい	両方	いいえ
server_id	はい	はい	はい		グローバル	はい
server_id_bits	はい	はい	はい		グローバル	いいえ
server_uuid			はい		グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
session_track_gtid	はい	はい	はい		両方	はい
session_track_schema	はい	はい	はい		両方	はい
session_track_state_change	はい	はい	はい		両方	はい
session_track_system_variables	はい	はい	はい		両方	はい
session_track_table_action_info	はい	はい	はい		両方	はい
sha256_password_auto_generate_keys	はい	はい	はい		グローバル	いいえ
sha256_password_private_key_path	はい	はい	はい		グローバル	いいえ
sha256_password_proxy_users	はい	はい	はい		グローバル	はい
sha256_password_public_key_path	はい	はい	はい		グローバル	いいえ
shared_memory	はい	はい	はい		グローバル	いいえ
shared_memory_base_name	はい	はい	はい		グローバル	いいえ
show_create_table_skip_secondary_engine	はい	はい	はい		セッション	はい
show_create_table_verbosity	はい	はい	はい		両方	はい
show_old_temporals	はい	はい	はい		両方	はい
show-slave-auth-info	はい	はい				
skip-character-set-client-handshake	はい	はい				
skip_external_locking	はい	はい	はい		グローバル	いいえ
skip-grant-tables	はい	はい				
skip-host-cache	はい	はい				
skip_name_resolve	はい	はい	はい		グローバル	いいえ
skip-ndbcluster	はい	はい				
skip_networking	はい	はい	はい		グローバル	いいえ
skip-new	はい	はい				
skip_show_databases	はい	はい	はい		グローバル	いいえ
skip-slave-start	はい	はい				
skip-ssl	はい	はい				
skip-stack-trace	はい	はい				
slave_allow_batching	はい	はい	はい		グローバル	はい
slave_checkpoint_group	はい	はい	はい		グローバル	はい
slave_checkpoint_period	はい	はい	はい		グローバル	はい
slave_compressed_protocol	はい	はい	はい		グローバル	はい
slave_exec_mode	はい	はい	はい		グローバル	はい
slave_load_tmpdir	はい	はい	はい		グローバル	いいえ
slave_max_allowed_packet	はい	はい	はい		グローバル	はい
slave_net_timeout	はい	はい	はい		グローバル	はい
Slave_open_temp_tables				はい	グローバル	いいえ
slave_parallel_type	はい	はい	はい		グローバル	はい

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
slave_parallel_workers	はい	はい	はい		グローバル	はい
slave_pending_jobs_size_max	はい	はい	はい		グローバル	はい
slave_preserve_commit_order	はい	はい	はい		グローバル	はい
Slave_rows_last_search_algorithm_used				はい	グローバル	いいえ
slave_rows_search_algorithms	はい	はい	はい		グローバル	はい
slave_skip_errors	はい	はい	はい		グローバル	いいえ
slave-sql-verify-checksum	はい	はい				
slave_sql_verify_checksum	はい	はい	はい		グローバル	はい
slave_transaction_retries	はい	はい	はい		グローバル	はい
slave_type_conversions	はい	はい	はい		グローバル	はい
Slow_launch_threads				はい	両方	いいえ
slow_launch_time	はい	はい	はい		グローバル	はい
Slow_queries				はい	両方	いいえ
slow_query_log	はい	はい	はい		グローバル	はい
slow_query_log_file	はい	はい	はい		グローバル	はい
slow-start-timeout	はい	はい				
socket	はい	はい	はい		グローバル	いいえ
sort_buffer_size	はい	はい	はい		両方	はい
Sort_merge_passes				はい	両方	いいえ
Sort_range				はい	両方	いいえ
Sort_rows				はい	両方	いいえ
Sort_scan				はい	両方	いいえ
sporadic-binlog-dump-fail	はい	はい				
sql_auto_is_null			はい		両方	はい
sql_big_selects			はい		両方	はい
sql_buffer_result			はい		両方	はい
sql_log_bin			はい		セッション	はい
sql_log_off			はい		両方	はい
sql_mode	はい	はい	はい		両方	はい
sql_notes			はい		両方	はい
sql_quote_show_create			はい		両方	はい
sql_require_primary_key	はい	はい	はい		両方	はい
sql_safe_updates			はい		両方	はい
sql_select_limit			はい		両方	はい
sql_slave_skip_counter			はい		グローバル	はい
sql_warnings			はい		両方	はい
ssl	はい	はい				

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
Ssl_accept_renegotiates				はい	グローバル	いいえ
Ssl_accepts				はい	グローバル	いいえ
ssl_ca	はい	はい	はい		グローバル	異なる
Ssl_callback_cache_hits				はい	グローバル	いいえ
ssl_capath	はい	はい	はい		グローバル	異なる
ssl_cert	はい	はい	はい		グローバル	異なる
Ssl_cipher				はい	両方	いいえ
ssl_cipher	はい	はい	はい		グローバル	異なる
Ssl_cipher_list				はい	両方	いいえ
Ssl_client_connects				はい	グローバル	いいえ
Ssl_connect_renegotiates				はい	グローバル	いいえ
ssl_crl	はい	はい	はい		グローバル	異なる
ssl_crlpath	はい	はい	はい		グローバル	異なる
Ssl_ctx_verify_depth				はい	グローバル	いいえ
Ssl_ctx_verify_mode				はい	グローバル	いいえ
Ssl_default_timeout				はい	両方	いいえ
Ssl_finished_accepts				はい	グローバル	いいえ
Ssl_finished_connects				はい	グローバル	いいえ
ssl_fips_mode	はい	はい	はい		グローバル	はい
ssl_key	はい	はい	はい		グローバル	異なる
Ssl_server_not_after				はい	両方	いいえ
Ssl_server_not_before				はい	両方	いいえ
Ssl_session_cache_hits				はい	グローバル	いいえ
Ssl_session_cache_misses				はい	グローバル	いいえ
Ssl_session_cache_mode				はい	グローバル	いいえ
Ssl_session_cache_overflows				はい	グローバル	いいえ
Ssl_session_cache_size				はい	グローバル	いいえ
Ssl_session_cache_timeouts				はい	グローバル	いいえ
Ssl_sessions_reused				はい	両方	いいえ
Ssl_used_session_cache_entries				はい	グローバル	いいえ
Ssl_verify_depth				はい	両方	いいえ
Ssl_verify_mode				はい	両方	いいえ
Ssl_version				はい	両方	いいえ
standalone	はい	はい				
stored_program_name	はい	はい	はい		グローバル	はい
stored_program_name_definition_cache	はい	はい	はい		グローバル	はい
super-large-pages	はい	はい				
super_read_only	はい	はい	はい		グローバル	はい
symbolic-links	はい	はい				

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
sync_binlog	はい	はい	はい		グローバル	はい
sync_master_info	はい	はい	はい		グローバル	はい
sync_relay_log	はい	はい	はい		グローバル	はい
sync_relay_log_info	はい	はい	はい		グローバル	はい
sysdate-is-now	はい	はい				
syseventlog.facility	はい	はい	はい		グローバル	はい
syseventlog.include_pid	はい	はい	はい		グローバル	はい
syseventlog.tag	はい	はい	はい		グローバル	はい
system_time_zone			はい		グローバル	いいえ
table_definition_cache	はい	はい	はい		グローバル	はい
table_encryption_privilege_check	はい	はい	はい		グローバル	はい
Table_locks_immediate				はい	グローバル	いいえ
Table_locks_waited				はい	グローバル	いいえ
table_open_cache	はい	はい	はい		グローバル	はい
Table_open_cache_hits				はい	両方	いいえ
table_open_cache_instances	はい	はい	はい		グローバル	いいえ
Table_open_cache_misses				はい	両方	いいえ
Table_open_cache_overflows				はい	両方	いいえ
tablespace_definition_cache	はい	はい	はい		グローバル	はい
tc-heuristic-recover	はい	はい				
Tc_log_max_pages_used				はい	グローバル	いいえ
Tc_log_page_size				はい	グローバル	いいえ
Tc_log_page_waits				はい	グローバル	いいえ
temptable_max_size	はい	はい	はい		グローバル	はい
temptable_max_size_per_thread	はい	はい	はい		グローバル	はい
temptable_use_map	はい	はい	はい		グローバル	はい
thread_cache_size	はい	はい	はい		グローバル	はい
thread_handling	はい	はい	はい		グローバル	いいえ
thread_pool_algorithm	はい	はい	はい		グローバル	いいえ
thread_pool_high_priority_connection_rollback	はい	はい	はい		両方	はい
thread_pool_max_active_query_threads_per_connection	はい	はい	はい		グローバル	はい
thread_pool_max_used_threads	はい	はい	はい		グローバル	はい
thread_pool_priority_rollback_timer	はい	はい	はい		両方	はい
thread_pool_size	はい	はい	はい		グローバル	いいえ
thread_pool_stall_limit	はい	はい	はい		グローバル	はい
thread_stack	はい	はい	はい		グローバル	いいえ
Threads_cached				はい	グローバル	いいえ
Threads_connected				はい	グローバル	いいえ
Threads_created				はい	グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
Threads_running				はい	グローバル	いいえ
time_zone			はい		両方	はい
timestamp			はい		セッション	はい
tls_ciphersuites	はい	はい	はい		グローバル	はい
tls_version	はい	はい	はい		グローバル	異なる
tmp_table_size	はい	はい	はい		両方	はい
tmpdir	はい	はい	はい		グローバル	いいえ
transaction_alloc_block_size	はい	はい	はい		両方	はい
transaction_allow_batching			はい		セッション	はい
transaction_isolation	はい	はい	はい		両方	はい
transaction_preallocate_size	はい	はい	はい		両方	はい
transaction_read_only	はい	はい	はい		両方	はい
transaction_write_set_extraction	はい	はい	はい		両方	はい
unique_checks			はい		両方	はい
updatable_views_limit	はい	はい	はい		両方	はい
upgrade	はい	はい				
Uptime				はい	グローバル	いいえ
Uptime_since_flush_status				はい	グローバル	いいえ
use_secondary_engine			はい		セッション	はい
user	はい	はい				
validate-config	はい	はい				
validate-password	はい	はい				
validate_password_check_user_names	はい	はい	はい		グローバル	はい
validate_password_dictionary_file	はい	はい	はい		グローバル	はい
validate_password_dictionary_file_last_parsed				はい	グローバル	いいえ
validate_password_dictionary_file_words_count				はい	グローバル	いいえ
validate_password_length	はい	はい	はい		グローバル	はい
validate_password_mixed_case_count	はい	はい	はい		グローバル	はい
validate_password_number_count	はい	はい	はい		グローバル	はい
validate_password_policy	はい	はい	はい		グローバル	はい
validate_password_special_char_count	はい	はい	はい		グローバル	はい
validate_password_check_user_names	はい	はい	はい		グローバル	はい
validate_password_dictionary_file	はい	はい	はい		グローバル	はい
validate_password_dictionary_file_last_parsed				はい	グローバル	いいえ
validate_password_dictionary_file_words_count				はい	グローバル	いいえ
validate_password_length	はい	はい	はい		グローバル	はい
validate_password_mixed_case_count	はい	はい	はい		グローバル	はい
validate_password_number_count	はい	はい	はい		グローバル	はい
validate_password_policy	はい	はい	はい		グローバル	はい

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
validate_password_special_char_count	はい	はい	はい		グローバル	はい
validate_user_plugins	はい	はい	はい		グローバル	いいえ
verbose	はい	はい				
version			はい		グローバル	いいえ
version_comment			はい		グローバル	いいえ
version_compile_machine			はい		グローバル	いいえ
version_compile_os			はい		グローバル	いいえ
version_compile_zlib			はい		グローバル	いいえ
version_tokens_session	はい	はい	はい		両方	はい
version_tokens_session_number	はい	はい	はい		両方	いいえ
wait_timeout	はい	はい	はい		両方	はい
warning_count			はい		セッション	いいえ
windowing_use_high_precision	はい	はい	はい		両方	はい

Notes:

1. このオプションは動的ですが、サーバーによってのみ設定する必要があります。この変数は手動で設定しないでください。

5.1.5 サーバーシステム変数リファレンス

次のテーブルに、`mysqld` 内で適用可能なすべてのシステム変数を示します。

このテーブルは、コマンド行オプション (Cmd-line)、構成ファイルで有効なオプション (Option file)、サーバーシステム変数 (System Var)、およびステータス変数 (Status var) を 1 つの統合リストに示し、各オプションまたは変数が有効な場所を示しています。コマンド行またはオプションファイルで設定されたサーバーオプションが、対応するシステム変数の名前と異なる場合、変数名は対応するオプションのすぐ下に表示されます。変数 (Var スコープ) のスコープは、グローバル、セッション、またはその両方です。変数の設定および使用の詳細は、対応するアイテムの説明を参照してください。必要に応じて、アイテムに関する詳細情報へのダイレクトリンクが提供されます。

表 5.2 「システム変数サマリー」

名前	コマンド行	オプションファイル	システム変数	変数スコープ	動的
activate_all_roles_on_login	はい	はい	はい	グローバル	はい
admin_address	はい	はい	はい	グローバル	いいえ
admin_port	はい	はい	はい	グローバル	いいえ
admin_ssl_ca	はい	はい	はい	グローバル	はい
admin_ssl_cpath	はい	はい	はい	グローバル	はい
admin_ssl_cert	はい	はい	はい	グローバル	はい
admin_ssl_cipher	はい	はい	はい	グローバル	はい
admin_ssl_crl	はい	はい	はい	グローバル	はい
admin_ssl_crlpath	はい	はい	はい	グローバル	はい
admin_ssl_key	はい	はい	はい	グローバル	はい
admin_tls_ciphersuites	はい	はい	はい	グローバル	はい
admin_tls_version	はい	はい	はい	グローバル	はい
audit_log_buffer_size	はい	はい	はい	グローバル	いいえ
audit_log_compression	はい	はい	はい	グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	変数スコープ	動的
audit_log_connection_policy	はい	はい	はい	グローバル	はい
audit_log_current_session			はい	両方	いいえ
audit_log_encryption	はい	はい	はい	グローバル	いいえ
audit_log_exclude_accounts	はい	はい	はい	グローバル	はい
audit_log_file	はい	はい	はい	グローバル	いいえ
audit_log_filter_id			はい	両方	いいえ
audit_log_flush			はい	グローバル	はい
audit_log_format	はい	はい	はい	グローバル	いいえ
audit_log_include_accounts	はい	はい	はい	グローバル	はい
audit_log_password_history_keep_days	はい	はい	はい	グローバル	はい
audit_log_policy	はい	はい	はい	グローバル	いいえ
audit_log_prune_sessions	はい	はい	はい	グローバル	はい
audit_log_read_buffer_size	はい	はい	はい	異なる	異なる
audit_log_rotate_on_size	はい	はい	はい	グローバル	はい
audit_log_statement_policy	はい	はい	はい	グローバル	はい
audit_log_strategy	はい	はい	はい	グローバル	いいえ
authentication_Idap_auth_method	はい	はい	はい	グローバル	はい
authentication_Idap_bind_base_dn	はい	はい	はい	グローバル	はい
authentication_Idap_bind_root_dn	はい	はい	はい	グローバル	はい
authentication_Idap_bind_root_pwd	はい	はい	はい	グローバル	はい
authentication_Idap_ca_path	はい	はい	はい	グローバル	はい
authentication_Idap_group_search_attr	はい	はい	はい	グローバル	はい
authentication_Idap_group_search_filter	はい	はい	はい	グローバル	はい
authentication_Idap_init_pool_size	はい	はい	はい	グローバル	はい
authentication_Idap_log_status	はい	はい	はい	グローバル	はい
authentication_Idap_max_pool_size	はい	はい	はい	グローバル	はい
authentication_Idap_referral	はい	はい	はい	グローバル	はい
authentication_Idap_server_host	はい	はい	はい	グローバル	はい
authentication_Idap_server_port	はい	はい	はい	グローバル	はい
authentication_Idap_tls	はい	はい	はい	グローバル	はい
authentication_Idap_user_search_attr	はい	はい	はい	グローバル	はい
authentication_Idap_auth_method_name	はい	はい	はい	グローバル	はい
authentication_Idap_bind_base_dn	はい	はい	はい	グローバル	はい
authentication_Idap_bind_root_dn	はい	はい	はい	グローバル	はい
authentication_Idap_bind_root_pwd	はい	はい	はい	グローバル	はい
authentication_Idap_ca_path	はい	はい	はい	グローバル	はい
authentication_Idap_group_search_attr	はい	はい	はい	グローバル	はい
authentication_Idap_group_search_filter	はい	はい	はい	グローバル	はい
authentication_Idap_init_pool_size	はい	はい	はい	グローバル	はい
authentication_Idap_log_status	はい	はい	はい	グローバル	はい

名前	コマンド行	オプションファイル	システム変数	変数スコープ	動的
authentication_ldap_simple_max_pool_size	はい	はい	はい	グローバル	はい
authentication_ldap_simple_referral	はい	はい	はい	グローバル	はい
authentication_ldap_simple_server_host	はい	はい	はい	グローバル	はい
authentication_ldap_simple_server_port	はい	はい	はい	グローバル	はい
authentication_ldap_simple_tls	はい	はい	はい	グローバル	はい
authentication_ldap_simple_user_search_attr	はい	はい	はい	グローバル	はい
authentication_windows_log_level	はい	はい	はい	グローバル	いいえ
authentication_windows_use_principal_name	はい	はい	はい	グローバル	いいえ
auto_generate_certificate	はい	はい	はい	グローバル	いいえ
auto_increment_increment	はい	はい	はい	両方	はい
auto_increment_offset	はい	はい	はい	両方	はい
autocommit	はい	はい	はい	両方	はい
automatic_sp_privileges	はい	はい	はい	グローバル	はい
avoid_temporal_upgrade	はい	はい	はい	グローバル	はい
back_log	はい	はい	はい	グローバル	いいえ
basedir	はい	はい	はい	グローバル	いいえ
big_tables	はい	はい	はい	両方	はい
bind_address	はい	はい	はい	グローバル	いいえ
binlog_cache_size	はい	はい	はい	グローバル	はい
binlog_checksum	はい	はい	はい	グローバル	はい
binlog_direct_non_transactional_updates	はい	はい	はい	両方	はい
binlog_encryption	はい	はい	はい	グローバル	はい
binlog_error_action	はい	はい	はい	グローバル	はい
binlog_expire_logs_seconds	はい	はい	はい	グローバル	はい
binlog_format	はい	はい	はい	両方	はい
binlog_group_commit_sync_delay	はい	はい	はい	グローバル	はい
binlog_group_commit_sync_no_delay_count	はい	はい	はい	グローバル	はい
binlog_gtid_simple_recovery	はい	はい	はい	グローバル	いいえ
binlog_max_flush_queue_time	はい	はい	はい	グローバル	はい
binlog_order_commits	はい	はい	はい	グローバル	はい
binlog_rotate_encryption_master_key_at_startup	はい	はい	はい	グローバル	いいえ
binlog_row_event_max_size	はい	はい	はい	グローバル	いいえ
binlog_row_image	はい	はい	はい	両方	はい
binlog_row_metadata	はい	はい	はい	グローバル	はい
binlog_row_value_options	はい	はい	はい	両方	はい
binlog_rows_query_log_events	はい	はい	はい	両方	はい
binlog_stmt_cache_size	はい	はい	はい	グローバル	はい
binlog_transaction_compression	はい	はい	はい	グローバル	はい
binlog_transaction_compression_level	はい	はい	はい	グローバル	はい
binlog_transaction_dependency_history_size	はい	はい	はい	グローバル	はい

名前	コマンド行	オプションファイル	システム変数	変数スコープ	動的
binlog_transaction_dependency_tracking	はい	はい	はい	グローバル	はい
block_encryption_mode	はい	はい	はい	両方	はい
bulk_insert_buffer_size	はい	はい	はい	両方	はい
caching_sha2_password_auto_generate_rsa_keys	はい	はい	はい	グローバル	いいえ
caching_sha2_password_digest_rounds	はい	はい	はい	グローバル	いいえ
caching_sha2_password_private_key_path	はい	はい	はい	グローバル	いいえ
caching_sha2_password_public_key_path	はい	はい	はい	グローバル	いいえ
character_set_client			はい	両方	はい
character_set_connection			はい	両方	はい
character_set_database (note 1)			はい	両方	はい
character_set_filesyschm	はい	はい	はい	両方	はい
character_set_results			はい	両方	はい
character_set_server	はい	はい	はい	両方	はい
character_set_system			はい	グローバル	いいえ
character_sets_dir	はい	はい	はい	グローバル	いいえ
check_proxy_users	はい	はい	はい	グローバル	はい
clone_autotune_connection	はい	はい	はい	グローバル	はい
clone_buffer_size	はい	はい	はい	グローバル	はい
clone_ddl_timeout	はい	はい	はい	グローバル	はい
clone_enable_compression	はい	はい	はい	グローバル	はい
clone_max_concurrent_connections	はい	はい	はい	グローバル	はい
clone_max_data_buffer_width	はい	はい	はい	グローバル	はい
clone_max_network_buffer_size	はい	はい	はい	グローバル	はい
clone_ssl_ca	はい	はい	はい	グローバル	はい
clone_ssl_cert	はい	はい	はい	グローバル	はい
clone_ssl_key	はい	はい	はい	グローバル	はい
clone_valid_donor_connections	はい	はい	はい	グローバル	はい
collation_connection			はい	両方	はい
collation_database (note 1)			はい	両方	はい
collation_server	はい	はい	はい	両方	はい
completion_type	はい	はい	はい	両方	はい
concurrent_insert	はい	はい	はい	グローバル	はい
connect_timeout	はい	はい	はい	グローバル	はい
connection_control_max_connections_per_thread	はい	はい	はい	グローバル	はい
connection_control_max_connection_delay	はい	はい	はい	グローバル	はい
connection_control_max_connection_delay_per_thread	はい	はい	はい	グローバル	はい
core_file			はい	グローバル	いいえ
create_admin_listen_thread	はい	はい	はい	グローバル	いいえ
cte_max_recursion_depth	はい	はい	はい	両方	はい

名前	コマンド行	オプションファイル	システム変数	変数スコープ	動的
daemon_memcached_enable_binlog	はい	はい	はい	グローバル	いいえ
daemon_memcached_engine_lib_name	はい	はい	はい	グローバル	いいえ
daemon_memcached_engine_lib_path	はい	はい	はい	グローバル	いいえ
daemon_memcached_option	はい	はい	はい	グローバル	いいえ
daemon_memcached_batch_size	はい	はい	はい	グローバル	いいえ
daemon_memcached_w_batch_size	はい	はい	はい	グローバル	いいえ
datadir	はい	はい	はい	グローバル	いいえ
debug	はい	はい	はい	両方	はい
debug_sync			はい	セッション	はい
default_authentication_plugin	はい	はい	はい	グローバル	いいえ
default_collation_for_utf8mb4			はい	両方	はい
default_password_lifetime	はい	はい	はい	グローバル	はい
default_storage_engine	はい	はい	はい	両方	はい
default_table_encryption	はい	はい	はい	両方	はい
default_tmp_storage_engine	はい	はい	はい	両方	はい
default_week_format	はい	はい	はい	両方	はい
delay_key_write	はい	はい	はい	グローバル	はい
delayed_insert_limit	はい	はい	はい	グローバル	はい
delayed_insert_timeout	はい	はい	はい	グローバル	はい
delayed_queue_size	はい	はい	はい	グローバル	はい
disabled_storage_engines	はい	はい	はい	グローバル	いいえ
disconnect_on_expired_password	はい	はい	はい	グローバル	いいえ
div_precision_increment	はい	はい	はい	両方	はい
dragnet.log_error_files	はい	はい	はい	グローバル	はい
end_markers_in_json	はい	はい	はい	両方	はい
enforce_gtid_consistency	はい	はい	はい	グローバル	はい
eq_range_index_div_limit	はい	はい	はい	両方	はい
error_count			はい	セッション	いいえ
event_scheduler	はい	はい	はい	グローバル	はい
expire_logs_days	はい	はい	はい	グローバル	はい
explicit_defaults_for_timestamp	はい	はい	はい	両方	はい
external_user			はい	セッション	いいえ
flush	はい	はい	はい	グローバル	はい
flush_time	はい	はい	はい	グローバル	はい
foreign_key_checks			はい	両方	はい
ft_boolean_syntax	はい	はい	はい	グローバル	はい
ft_max_word_len	はい	はい	はい	グローバル	いいえ
ft_min_word_len	はい	はい	はい	グローバル	いいえ
ft_query_expansion_limit	はい	はい	はい	グローバル	いいえ
ft_stopword_file	はい	はい	はい	グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	変数スコープ	動的
general_log	はい	はい	はい	グローバル	はい
general_log_file	はい	はい	はい	グローバル	はい
generated_random_password_length	はい	はい	はい	両方	はい
group_concat_max_len	はい	はい	はい	両方	はい
group_replication_allow_ambiguous_recovery_error_points	はい	はい	はい	グローバル	はい
group_replication_allow_local_lower_version_to_join	はい	はい	はい	グローバル	はい
group_replication_allow_increment_increment	はい	はい	はい	グローバル	はい
group_replication_allow_rejoin_tries	はい	はい	はい	グローバル	はい
group_replication_bootstrap_group	はい	はい	はい	グローバル	はい
group_replication_checkpoint_threshold	はい	はい	はい	グローバル	はい
group_replication_communication_debug_options	はい	はい	はい	グローバル	はい
group_replication_communication_max_message_size	はい	はい	はい	グローバル	はい
group_replication_components_stop_timeout	はい	はい	はい	グローバル	はい
group_replication_compression_threshold	はい	はい	はい	グローバル	はい
group_replication_consistency	はい	はい	はい	両方	はい
group_replication_enable_update_everywhere_checks	はい	はい	はい	グローバル	はい
group_replication_enable_state_action	はい	はい	はい	グローバル	はい
group_replication_force_control_applier_threshold	はい	はい	はい	グローバル	はい
group_replication_force_control_certifier_threshold	はい	はい	はい	グローバル	はい
group_replication_force_control_hold_percent	はい	はい	はい	グローバル	はい
group_replication_force_control_max_control_quota	はい	はい	はい	グローバル	はい
group_replication_force_control_member_quota_percent	はい	はい	はい	グローバル	はい
group_replication_force_control_min_quota	はい	はい	はい	グローバル	はい
group_replication_force_control_min_recovery_quota	はい	はい	はい	グローバル	はい
group_replication_force_control_mode	はい	はい	はい	グローバル	はい
group_replication_force_control_period	はい	はい	はい	グローバル	はい
group_replication_force_control_release_percent	はい	はい	はい	グローバル	はい
group_replication_force_members	はい	はい	はい	グローバル	はい
group_replication_group_name	はい	はい	はい	グローバル	はい
group_replication_group_seeds	はい	はい	はい	グローバル	はい
group_replication_group_assignment_block_size	はい	はい	はい	グローバル	はい
group_replication_ip_blacklist	はい	はい	はい	グローバル	はい
group_replication_ip_whitelist	はい	はい	はい	グローバル	はい
group_replication_log_address	はい	はい	はい	グローバル	はい
group_replication_member_expel_timeout	はい	はい	はい	グローバル	はい
group_replication_member_weight	はい	はい	はい	グローバル	はい
group_replication_message_cache_size	はい	はい	はい	グローバル	はい
group_replication_pin_loops	はい	はい	はい	グローバル	はい
group_replication_read_very_complete_algorithm	はい	はい	はい	グローバル	はい
group_replication_read_very_compression_algorithm	はい	はい	はい	グローバル	はい

名前	コマンド行	オプションファイル	システム変数	変数スコープ	動的
group_replication_read_only_get_public_key	はい	はい	はい	グローバル	はい
group_replication_read_only_public_key	はい	はい	はい	グローバル	はい
group_replication_read_only_reconnect_interval	はい	はい	はい	グローバル	はい
group_replication_read_only_retry_count	はい	はい	はい	グローバル	はい
group_replication_read_only_ssl_ca	はい	はい	はい	グローバル	はい
group_replication_read_only_ssl_capath	はい	はい	はい	グローバル	はい
group_replication_read_only_ssl_cert	はい	はい	はい	グローバル	はい
group_replication_read_only_ssl_cipher	はい	はい	はい	グローバル	はい
group_replication_read_only_ssl_crl	はい	はい	はい	グローバル	はい
group_replication_read_only_ssl_crlpath	はい	はい	はい	グローバル	はい
group_replication_read_only_ssl_key	はい	はい	はい	グローバル	はい
group_replication_read_only_ssl_verify_server_cert	はい	はい	はい	グローバル	はい
group_replication_read_only_tls_ciphersuites	はい	はい	はい	グローバル	はい
group_replication_read_only_tls_version	はい	はい	はい	グローバル	はい
group_replication_read_only_use_ssl	はい	はい	はい	グローバル	はい
group_replication_read_only_zstd_compression_level	はい	はい	はい	グローバル	はい
group_replication_single_primary_mode	はい	はい	はい	グローバル	はい
group_replication_slave_mode	はい	はい	はい	グローバル	はい
group_replication_slave_on_boot	はい	はい	はい	グローバル	はい
group_replication_transaction_source	はい	はい	はい	グローバル	はい
group_replication_transaction_size_limit	はい	はい	はい	グローバル	はい
group_replication_unreachable_majority_timeout	はい	はい	はい	グローバル	はい
gtid_executed			はい	異なる	いいえ
gtid_executed_compression_period	はい	はい	はい	グローバル	はい
gtid_mode	はい	はい	はい	グローバル	はい
gtid_next			はい	セッション	はい
gtid_owned			はい	両方	いいえ
gtid_purged			はい	グローバル	はい
have_compress			はい	グローバル	いいえ
have_dynamic_loading			はい	グローバル	いいえ
have_geometry			はい	グローバル	いいえ
have_openssl			はい	グローバル	いいえ
have_profiling			はい	グローバル	いいえ
have_query_cache			はい	グローバル	いいえ
have_rtree_keys			はい	グローバル	いいえ
have_ssl			はい	グローバル	いいえ
have_statement_timeout			はい	グローバル	いいえ
have_symlink			はい	グローバル	いいえ
histogram_generation_max_mem_size	はい	はい	はい	両方	はい
host_cache_size	はい	はい	はい	グローバル	はい

名前	コマンド行	オプションファイル	システム変数	変数スコープ	動的
hostname			はい	グローバル	いいえ
identity			はい	セッション	はい
immediate_server_version			はい	セッション	はい
information_schema_stats_expiry	はい	はい	はい	両方	はい
init_connect	はい	はい	はい	グローバル	はい
init_file	はい	はい	はい	グローバル	いいえ
init_slave	はい	はい	はい	グローバル	はい
innodb_adaptive_flushing	はい	はい	はい	グローバル	はい
innodb_adaptive_flushing_lwm	はい	はい	はい	グローバル	はい
innodb_adaptive_hash_index	はい	はい	はい	グローバル	はい
innodb_adaptive_hash_index_parts	はい	はい	はい	グローバル	いいえ
innodb_adaptive_max_sleep_delay	はい	はい	はい	グローバル	はい
innodb_api_bk_commit_interval	はい	はい	はい	グローバル	はい
innodb_api_disable_rowlock	はい	はい	はい	グローバル	いいえ
innodb_api_enable_log	はい	はい	はい	グローバル	いいえ
innodb_api_enable_mutex	はい	はい	はい	グローバル	いいえ
innodb_api_trx_level	はい	はい	はい	グローバル	はい
innodb_autoextend_increment	はい	はい	はい	グローバル	はい
innodb_autoinc_lock_mode	はい	はい	はい	グローバル	いいえ
innodb_background_flush_list_empty	はい	はい	はい	グローバル	はい
innodb_buffer_pool_chunk_size	はい	はい	はい	グローバル	いいえ
innodb_buffer_pool_dump	はい	はい	はい	グローバル	いいえ
innodb_buffer_pool_dump_at_shutdown	はい	はい	はい	グローバル	はい
innodb_buffer_pool_dump_now	はい	はい	はい	グローバル	はい
innodb_buffer_pool_dump_pct	はい	はい	はい	グローバル	はい
innodb_buffer_pool_filename	はい	はい	はい	グローバル	はい
innodb_buffer_pool_heartbeat_file	はい	はい	はい	グローバル	はい
innodb_buffer_pool_instances	はい	はい	はい	グローバル	いいえ
innodb_buffer_pool_instances_abort	はい	はい	はい	グローバル	はい
innodb_buffer_pool_instances_at_startup	はい	はい	はい	グローバル	いいえ
innodb_buffer_pool_instances_now	はい	はい	はい	グローバル	はい
innodb_buffer_pool_load	はい	はい	はい	グローバル	はい
innodb_change_buffer_max_size	はい	はい	はい	グローバル	はい
innodb_change_buffering	はい	はい	はい	グローバル	はい
innodb_change_buffering_debug	はい	はい	はい	グローバル	はい
innodb_checkpoint_disabled	はい	はい	はい	グローバル	はい
innodb_checksum_algorithm	はい	はい	はい	グローバル	はい
innodb_cmp_per_index_enabled	はい	はい	はい	グローバル	はい
innodb_commit_concurrency	はい	はい	はい	グローバル	はい
innodb_compress_debug	はい	はい	はい	グローバル	はい

名前	コマンド行	オプションファイル	システム変数	変数スコープ	動的
innodb_compression_failure_threshold	はい	はい	はい	グローバル	はい
innodb_compression_level	はい	はい	はい	グローバル	はい
innodb_compression_pct_max	はい	はい	はい	グローバル	はい
innodb_concurrency_tickets	はい	はい	はい	グローバル	はい
innodb_data_file_path	はい	はい	はい	グローバル	いいえ
innodb_data_home_dir	はい	はい	はい	グローバル	いいえ
innodb_ddl_log_crash_reset_debug	はい	はい	はい	グローバル	はい
innodb_deadlock_detect	はい	はい	はい	グローバル	はい
innodb_dedicated_server	はい	はい	はい	グローバル	いいえ
innodb_default_row_format	はい	はい	はい	グローバル	はい
innodb_directories	はい	はい	はい	グローバル	いいえ
innodb_disable_sort_file_cache	はい	はい	はい	グローバル	はい
innodb_doublewrite	はい	はい	はい	グローバル	いいえ
innodb_doublewrite_batch_size	はい	はい	はい	グローバル	いいえ
innodb_doublewrite_batch	はい	はい	はい	グローバル	いいえ
innodb_doublewrite_files	はい	はい	はい	グローバル	いいえ
innodb_doublewrite_pages	はい	はい	はい	グローバル	いいえ
innodb_extend_and_initialize	はい	はい	はい	グローバル	はい
innodb_fast_shutdown	はい	はい	はい	グローバル	はい
innodb_fil_make_page_dirty_debug	はい	はい	はい	グローバル	はい
innodb_file_per_table	はい	はい	はい	グローバル	はい
innodb_fill_factor	はい	はい	はい	グローバル	はい
innodb_flush_log_at_timeout	はい	はい	はい	グローバル	はい
innodb_flush_log_at_trx_commit	はい	はい	はい	グローバル	はい
innodb_flush_method	はい	はい	はい	グローバル	いいえ
innodb_flush_neighbors	はい	はい	はい	グローバル	はい
innodb_flush_sync	はい	はい	はい	グローバル	はい
innodb_flushing_avg_loops	はい	はい	はい	グローバル	はい
innodb_force_load_terminated	はい	はい	はい	グローバル	いいえ
innodb_force_recovery	はい	はい	はい	グローバル	いいえ
innodb_fsync_threshold	はい	はい	はい	グローバル	はい
innodb_ft_aux_table			はい	グローバル	はい
innodb_ft_cache_size	はい	はい	はい	グローバル	いいえ
innodb_ft_enable_debug_print	はい	はい	はい	グローバル	はい
innodb_ft_enable_stopword	はい	はい	はい	両方	はい
innodb_ft_max_token_size	はい	はい	はい	グローバル	いいえ
innodb_ft_min_token_size	はい	はい	はい	グローバル	いいえ
innodb_ft_num_word_optimize	はい	はい	はい	グローバル	はい
innodb_ft_result_cache_limit	はい	はい	はい	グローバル	はい
innodb_ft_server_stopword_table	はい	はい	はい	グローバル	はい

名前	コマンド行	オプションファイル	システム変数	変数スコープ	動的
innodb_ft_sort_pll_degree	はい	はい	はい	グローバル	いいえ
innodb_ft_total_cache_size	はい	はい	はい	グローバル	いいえ
innodb_ft_user_stopwords_table	はい	はい	はい	両方	はい
innodb_idle_flush_pct	はい	はい	はい	グローバル	はい
innodb_io_capacity	はい	はい	はい	グローバル	はい
innodb_io_capacity_max	はい	はい	はい	グローバル	はい
innodb_limit_optimizations	はい	はい	はい	グローバル	はい
innodb_lock_wait_timeout	はい	はい	はい	両方	はい
innodb_log_buffer_size	はい	はい	はい	グローバル	異なる
innodb_log_checkpoint_fuzzy_nowait	はい	はい	はい	グローバル	はい
innodb_log_checkpoint_nowait	はい	はい	はい	グローバル	はい
innodb_log_checks_only	はい	はい	はい	グローバル	はい
innodb_log_compressed_pages	はい	はい	はい	グローバル	はい
innodb_log_file_size	はい	はい	はい	グローバル	いいえ
innodb_log_files_in_group	はい	はい	はい	グローバル	いいえ
innodb_log_group_home_dir	はい	はい	はい	グローバル	いいえ
innodb_log_spin_capacity_lwm	はい	はい	はい	グローバル	はい
innodb_log_spin_capacity_hwm	はい	はい	はい	グローバル	はい
innodb_log_wait_for_flush_spin_hwm	はい	はい	はい	グローバル	はい
innodb_log_write_ahead_size	はい	はい	はい	グローバル	はい
innodb_log_writer_threads	はい	はい	はい	グローバル	はい
innodb_lru_scan_depth	はい	はい	はい	グローバル	はい
innodb_max_dirty_pages_pct	はい	はい	はい	グローバル	はい
innodb_max_dirty_pages_pct_lwm	はい	はい	はい	グローバル	はい
innodb_max_purge_age	はい	はい	はい	グローバル	はい
innodb_max_purge_age_delay	はい	はい	はい	グローバル	はい
innodb_max_undo_log_size	はい	はい	はい	グローバル	はい
innodb_merge_threads_set_all_debug	はい	はい	はい	グローバル	はい
innodb_monitor_disable	はい	はい	はい	グローバル	はい
innodb_monitor_enable	はい	はい	はい	グローバル	はい
innodb_monitor_reset	はい	はい	はい	グローバル	はい
innodb_monitor_reset_all	はい	はい	はい	グローバル	はい
innodb_numa_interleave	はい	はい	はい	グローバル	いいえ
innodb_old_blocks	はい	はい	はい	グローバル	はい
innodb_old_blocks_time	はい	はい	はい	グローバル	はい
innodb_online_alter_log_max_size	はい	はい	はい	グローバル	はい
innodb_open_files	はい	はい	はい	グローバル	いいえ
innodb_optimize_fulltext_only	はい	はい	はい	グローバル	はい
innodb_page_cleaners	はい	はい	はい	グローバル	いいえ
innodb_page_size	はい	はい	はい	グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	変数スコープ	動的	
innodb_parallel_readahead	はい	reads	はい	はい	セッション	はい
innodb_print_all_deadlocks	はい	locks	はい	はい	グローバル	はい
innodb_print_ddl_logs	はい		はい	はい	グローバル	はい
innodb_purge_batch_size	はい	size	はい	はい	グローバル	はい
innodb_purge_rseg_undo_frequency	はい	undo_frequency	はい	はい	グローバル	はい
innodb_purge_threads	はい		はい	はい	グローバル	いいえ
innodb_random_read_ahead	はい	ahead	はい	はい	グローバル	はい
innodb_read_ahead_threshold	はい	threshold	はい	はい	グローバル	はい
innodb_read_io_threads	はい	threads	はい	はい	グローバル	いいえ
innodb_read_only	はい		はい	はい	グローバル	いいえ
innodb_redo_log_archive_dirs	はい	archive_dirs	はい	はい	グローバル	はい
innodb_redo_log_encrypt	はい	encrypt	はい	はい	グローバル	はい
innodb_replication_delay	はい	delay	はい	はい	グローバル	はい
innodb_rollback_on_timeout	はい	timeout	はい	はい	グローバル	いいえ
innodb_rollback_segments	はい	segments	はい	はい	グローバル	はい
innodb_saved_page_checksum_debug	はい	checksum_debug	はい	はい	グローバル	はい
innodb_sort_buffer_size	はい	size	はい	はい	グローバル	いいえ
innodb_spin_wait_delay	はい	delay	はい	はい	グローバル	はい
innodb_spin_wait_parallel_multiplier	はい	parallel_multiplier	はい	はい	グローバル	はい
innodb_stats_auto_recalc	はい	recalc	はい	はい	グローバル	はい
innodb_stats_include_delete_marked	はい	delete_marked	はい	はい	グローバル	はい
innodb_stats_method	はい		はい	はい	グローバル	はい
innodb_stats_on_metadata	はい	metadata	はい	はい	グローバル	はい
innodb_stats_persisted	はい		はい	はい	グローバル	はい
innodb_stats_persistent_sample_pages	はい	sample_pages	はい	はい	グローバル	はい
innodb_stats_transient_sample_pages	はい	sample_pages	はい	はい	グローバル	はい
innodb_status_output	はい		はい	はい	グローバル	はい
innodb_status_output_locks	はい	locks	はい	はい	グローバル	はい
innodb_strict_mode	はい		はい	はい	両方	はい
innodb_sync_array_size	はい	size	はい	はい	グローバル	いいえ
innodb_sync_debug	はい		はい	はい	グローバル	いいえ
innodb_sync_spin_delay	はい	delay	はい	はい	グローバル	はい
innodb_table_locks	はい		はい	はい	両方	はい
innodb_temp_data_home_path	はい	home_path	はい	はい	グローバル	いいえ
innodb_temp_tablespace_dir	はい	spaces_dir	はい	はい	グローバル	いいえ
innodb_thread_concurrency	はい	concurrency	はい	はい	グローバル	はい
innodb_thread_sleep_delay	はい	delay	はい	はい	グローバル	はい
innodb_tmpdir	はい		はい	はい	両方	はい
innodb_trx_purge_verbose_update_only_debug	はい	update_only_debug	はい	はい	グローバル	はい
innodb_trx_rseg_nocache_debug	はい	nocache_debug	はい	はい	グローバル	はい

名前	コマンド行	オプションファイル	システム変数	変数スコープ	動的
innodb_undo_directory	はい	はい	はい	グローバル	いいえ
innodb_undo_log_encrypt	はい	はい	はい	グローバル	はい
innodb_undo_log_timestamp	はい	はい	はい	グローバル	はい
innodb_undo_tablespace	はい	はい	はい	グローバル	異なる
innodb_use_native_aio	はい	はい	はい	グローバル	いいえ
innodb_validate_tablespaces	はい	はい	はい	グローバル	いいえ
innodb_version			はい	グローバル	いいえ
innodb_write_io_threads	はい	はい	はい	グローバル	いいえ
insert_id			はい	セッション	はい
interactive_timeout	はい	はい	はい	両方	はい
internal_tmp_disk_storage_engine	はい	はい	はい	グローバル	はい
internal_tmp_mem_storage_engine	はい	はい	はい	両方	はい
join_buffer_size	はい	はい	はい	両方	はい
keep_files_on_create	はい	はい	はい	両方	はい
key_buffer_size	はい	はい	はい	グローバル	はい
key_cache_age_threshold	はい	はい	はい	グローバル	はい
key_cache_block_size	はい	はい	はい	グローバル	はい
key_cache_division_limit	はい	はい	はい	グローバル	はい
keyring_aws_cmk_id	はい	はい	はい	グローバル	はい
keyring_aws_conf_file	はい	はい	はい	グローバル	いいえ
keyring_aws_data_file	はい	はい	はい	グローバル	いいえ
keyring_aws_region	はい	はい	はい	グローバル	はい
keyring_encrypted_data_file	はい	はい	はい	グローバル	はい
keyring_encrypted_password_file	はい	はい	はい	グローバル	はい
keyring_file_data	はい	はい	はい	グローバル	はい
keyring_hashicorp_auth_path	はい	はい	はい	グローバル	はい
keyring_hashicorp_ca_path	はい	はい	はい	グローバル	はい
keyring_hashicorp_cert_path	はい	はい	はい	グローバル	はい
keyring_hashicorp_commit_auth_path			はい	グローバル	いいえ
keyring_hashicorp_commit_ca_path			はい	グローバル	いいえ
keyring_hashicorp_commit_caching			はい	グローバル	いいえ
keyring_hashicorp_commit_role_id			はい	グローバル	いいえ
keyring_hashicorp_commit_server_url			はい	グローバル	いいえ
keyring_hashicorp_commit_store_path			はい	グローバル	いいえ
keyring_hashicorp_role_id	はい	はい	はい	グローバル	はい
keyring_hashicorp_secret_id	はい	はい	はい	グローバル	はい
keyring_hashicorp_server_url	はい	はい	はい	グローバル	はい
keyring_hashicorp_store_path	はい	はい	はい	グローバル	はい
keyring_oci_ca_certificate	はい	はい	はい	グローバル	いいえ
keyring_oci_compartment	はい	はい	はい	グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	変数スコープ	動的
keyring_oci_encrypt	はい endpoint	はい	はい	グローバル	いいえ
keyring_oci_key_file	はい	はい	はい	グローバル	いいえ
keyring_oci_key_finger	はい print	はい	はい	グローバル	いいえ
keyring_oci_managed	はい endpoint	はい	はい	グローバル	いいえ
keyring_oci_master	はい	はい	はい	グローバル	いいえ
keyring_oci_secrets	はい endpoint	はい	はい	グローバル	いいえ
keyring_oci_tenancy	はい	はい	はい	グローバル	いいえ
keyring_oci_user	はい	はい	はい	グローバル	いいえ
keyring_oci_vaults	はい endpoint	はい	はい	グローバル	いいえ
keyring_oci_virtual	はい	はい	はい	グローバル	いいえ
keyring_okv_conf_dir	はい	はい	はい	グローバル	はい
keyring_operations			はい	グローバル	はい
language	はい	はい	はい	グローバル	いいえ
large_files_support			はい	グローバル	いいえ
large_page_size			はい	グローバル	いいえ
large_pages	はい	はい	はい	グローバル	いいえ
last_insert_id			はい	セッション	はい
lc_messages	はい	はい	はい	両方	はい
lc_messages_dir	はい	はい	はい	グローバル	いいえ
lc_time_names	はい	はい	はい	両方	はい
license			はい	グローバル	いいえ
local_infile	はい	はい	はい	グローバル	はい
lock_order	はい	はい	はい	グローバル	いいえ
lock_order_debug	はい	はい	はい	グローバル	いいえ
lock_order_debug_mis	はい signing_arc	はい	はい	グローバル	いいえ
lock_order_debug_mis	はい signing_key	はい	はい	グローバル	いいえ
lock_order_debug_mis	はい signing_unlock	はい	はい	グローバル	いいえ
lock_order_dependencies	はい	はい	はい	グローバル	いいえ
lock_order_extra_dependencies	はい	はい	はい	グローバル	いいえ
lock_order_output_directory	はい	はい	はい	グローバル	いいえ
lock_order_print_text	はい	はい	はい	グローバル	いいえ
lock_order_trace_log	はい	はい	はい	グローバル	いいえ
lock_order_trace_mis	はい signing_arc	はい	はい	グローバル	いいえ
lock_order_trace_mis	はい signing_key	はい	はい	グローバル	いいえ
lock_order_trace_mis	はい signing_unlock	はい	はい	グローバル	いいえ
lock_wait_timeout	はい	はい	はい	両方	はい
locked_in_memory			はい	グローバル	いいえ
log_bin			はい	グローバル	いいえ
log_bin_basename			はい	グローバル	いいえ
log_bin_index	はい	はい	はい	グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	変数スコープ	動的
log_bin_trust_function_creators	はい	はい	はい	グローバル	はい
log_bin_use_v1_row_events	はい	はい	はい	グローバル	いいえ
log_error	はい	はい	はい	グローバル	いいえ
log_error_services	はい	はい	はい	グローバル	はい
log_error_suppress_errors	はい	はい	はい	グローバル	はい
log_error_verbosity	はい	はい	はい	グローバル	はい
log_output	はい	はい	はい	グローバル	はい
log_queries_not_using_indexes	はい	はい	はい	グローバル	はい
log_raw	はい	はい	はい	グローバル	はい
log_slave_updates	はい	はい	はい	グローバル	いいえ
log_slow_admin_statements	はい	はい	はい	グローバル	はい
log_slow_extra	はい	はい	はい	グローバル	はい
log_slow_slave_statements	はい	はい	はい	グローバル	はい
log_statements_unsafe_for_binlog	はい	はい	はい	グローバル	はい
log_syslog	はい	はい	はい	グローバル	はい
log_syslog_facility	はい	はい	はい	グローバル	はい
log_syslog_include_pid	はい	はい	はい	グローバル	はい
log_syslog_tag	はい	はい	はい	グローバル	はい
log_throttle_queries_not_using_indexes	はい	はい	はい	グローバル	はい
log_timestamps	はい	はい	はい	グローバル	はい
long_query_time	はい	はい	はい	両方	はい
low_priority_updates	はい	はい	はい	両方	はい
lower_case_file_system			はい	グローバル	いいえ
lower_case_table_names	はい	はい	はい	グローバル	いいえ
mandatory_roles	はい	はい	はい	グローバル	はい
master_info_repository	はい	はい	はい	グローバル	はい
master_verify_checksums	はい	はい	はい	グローバル	はい
max_allowed_packet_size	はい	はい	はい	両方	はい
max_binlog_cache_size	はい	はい	はい	グローバル	はい
max_binlog_size	はい	はい	はい	グローバル	はい
max_binlog_stmt_cache_size	はい	はい	はい	グローバル	はい
max_connect_errors	はい	はい	はい	グローバル	はい
max_connections	はい	はい	はい	グローバル	はい
max_delayed_threads	はい	はい	はい	両方	はい
max_digest_length	はい	はい	はい	グローバル	いいえ
max_error_count	はい	はい	はい	両方	はい
max_execution_time	はい	はい	はい	両方	はい
max_heap_table_size	はい	はい	はい	両方	はい
max_insert_delayed_threads			はい	両方	はい
max_join_size	はい	はい	はい	両方	はい

名前	コマンド行	オプションファイル	システム変数	変数スコープ	動的
max_length_for_sort_data	はい	はい	はい	両方	はい
max_points_in_geometry	はい	はい	はい	両方	はい
max_prepared_stmt_count	はい	はい	はい	グローバル	はい
max_relay_log_size	はい	はい	はい	グローバル	はい
max_seeks_for_key	はい	はい	はい	両方	はい
max_sort_length	はい	はい	はい	両方	はい
max_sp_recursion_depth	はい	はい	はい	両方	はい
max_user_connections	はい	はい	はい	両方	はい
max_write_lock_count	はい	はい	はい	グローバル	はい
mecab_rc_file	はい	はい	はい	グローバル	いいえ
metadata_locks_cache_size	はい	はい	はい	グローバル	いいえ
metadata_locks_hash_instances	はい	はい	はい	グローバル	いいえ
min_examined_row_limit	はい	はい	はい	両方	はい
mysam_data_point_size	はい	はい	はい	グローバル	はい
mysam_max_sort_file_size	はい	はい	はい	グローバル	はい
mysam_mmap_size	はい	はい	はい	グローバル	いいえ
mysam_recover_options	はい	はい	はい	グローバル	いいえ
mysam_repair_threads	はい	はい	はい	両方	はい
mysam_sort_buffer_size	はい	はい	はい	両方	はい
mysam_stats_method	はい	はい	はい	両方	はい
mysam_use_mmap	はい	はい	はい	グローバル	はい
mysql_firewall_mode	はい	はい	はい	グローバル	はい
mysql_firewall_trace	はい	はい	はい	グローバル	はい
mysql_native_password_proxy_users	はい	はい	はい	グローバル	はい
mysqlx_bind_address	はい	はい	はい	グローバル	いいえ
mysqlx_compression_algorithms	はい	はい	はい	グローバル	はい
mysqlx_connect_timeout	はい	はい	はい	グローバル	はい
mysqlx_deflate_default_compression_level	はい	はい	はい	グローバル	はい
mysqlx_deflate_max_client_compression_level	はい	はい	はい	グローバル	はい
mysqlx_document_id_unique_prefix	はい	はい	はい	グローバル	はい
mysqlx_enable_helptext	はい	はい	はい	グローバル	はい
mysqlx_idle_worker_thread_timeout	はい	はい	はい	グローバル	はい
mysqlx_interactive_timeout	はい	はい	はい	グローバル	はい
mysqlx_lz4_default_compression_level	はい	はい	はい	グローバル	はい
mysqlx_lz4_max_client_compression_level	はい	はい	はい	グローバル	はい
mysqlx_max_allowed_packet	はい	はい	はい	グローバル	はい
mysqlx_max_connections	はい	はい	はい	グローバル	はい
mysqlx_min_worker_threads	はい	はい	はい	グローバル	はい
mysqlx_port	はい	はい	はい	グローバル	いいえ
mysqlx_port_timeout	はい	はい	はい	グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	変数スコープ	動的
mysqlx_read_timeout	はい	はい	はい	セッション	はい
mysqlx_socket	はい	はい	はい	グローバル	いいえ
mysqlx_ssl_ca	はい	はい	はい	グローバル	いいえ
mysqlx_ssl_capath	はい	はい	はい	グローバル	いいえ
mysqlx_ssl_cert	はい	はい	はい	グローバル	いいえ
mysqlx_ssl_cipher	はい	はい	はい	グローバル	いいえ
mysqlx_ssl_crl	はい	はい	はい	グローバル	いいえ
mysqlx_ssl_crlpath	はい	はい	はい	グローバル	いいえ
mysqlx_ssl_key	はい	はい	はい	グローバル	いいえ
mysqlx_wait_timeout	はい	はい	はい	セッション	はい
mysqlx_write_timeout	はい	はい	はい	セッション	はい
mysqlx_zstd_default_compression_level	はい	はい	はい	グローバル	はい
mysqlx_zstd_max_compression_level	はい	はい	はい	グローバル	はい
named_pipe	はい	はい	はい	グローバル	いいえ
named_pipe_full_access_group	はい	はい	はい	グローバル	いいえ
ndb_allow_copying_to_table	はい	はい	はい	両方	はい
ndb_autoincrement_increment	はい	はい	はい	両方	はい
ndb_batch_size	はい	はい	はい	グローバル	いいえ
ndb_blob_read_batch_size	はい	はい	はい	両方	はい
ndb_blob_write_batch_size	はい	はい	はい	両方	はい
ndb_cache_check_time	はい	はい	はい	グローバル	はい
ndb_clear_apply_status	はい		はい	グローバル	はい
ndb_cluster_connect_timeout	はい	はい	はい	グローバル	いいえ
ndb_cluster_connect_timeout_nodeids	はい	はい	はい	グローバル	いいえ
Ndb_conflict_last_conflict_epoch			はい	グローバル	いいえ
ndb_conflict_role	はい	はい	はい	グローバル	はい
ndb_data_node_neighbour	はい	はい	はい	グローバル	はい
ndb_dbg_check_shades	はい	はい	はい	両方	はい
ndb_default_column_format	はい	はい	はい	グローバル	はい
ndb_default_column_format	はい	はい	はい	グローバル	はい
ndb_deferred_constraints	はい	はい	はい	両方	はい
ndb_deferred_constraints	はい	はい	はい	両方	はい
ndb_distribution	はい	はい	はい	グローバル	はい
ndb_distribution	はい	はい	はい	グローバル	はい
ndb_eventbuffer_frequency_percent	はい	はい	はい	グローバル	はい
ndb_eventbuffer_max_size	はい	はい	はい	グローバル	はい
ndb_extra_logging	はい	はい	はい	グローバル	はい
ndb_force_send	はい	はい	はい	両方	はい
ndb_fully_replicated	はい	はい	はい	両方	はい
ndb_index_stat_enabled	はい	はい	はい	両方	はい

名前	コマンド行	オプションファイル	システム変数	変数スコープ	動的
ndb_index_stat_optimize	はい	はい	はい	両方	はい
ndb_join_pushdown			はい	両方	はい
ndb_log_apply_status	はい	はい	はい	グローバル	いいえ
ndb_log_apply_status	はい	はい	はい	グローバル	いいえ
ndb_log_bin	はい		はい	両方	はい
ndb_log_binlog_index	はい		はい	グローバル	はい
ndb_log_empty_epochs	はい	はい	はい	グローバル	はい
ndb_log_empty_epochs	はい	はい	はい	グローバル	はい
ndb_log_empty_updates	はい	はい	はい	グローバル	はい
ndb_log_empty_updates	はい	はい	はい	グローバル	はい
ndb_log_exclusive_reads	はい	はい	はい	両方	はい
ndb_log_exclusive_reads	はい	はい	はい	両方	はい
ndb_log_fail_terminate	はい	はい	はい	グローバル	いいえ
ndb_log_orig	はい	はい	はい	グローバル	いいえ
ndb_log_orig	はい	はい	はい	グローバル	いいえ
ndb_log_transaction_id	はい	はい	はい	グローバル	いいえ
ndb_log_transaction_id			はい	グローバル	いいえ
ndb_log_update_async	はい	はい	はい	グローバル	はい
ndb_log_update_minimal	はい	はい	はい	グローバル	はい
ndb_log_updated_operations	はい	はい	はい	グローバル	はい
ndb_metadata_checksum	はい	はい	はい	グローバル	はい
ndb_metadata_checksum_interval	はい	はい	はい	グローバル	はい
ndb_metadata_sync			はい	グローバル	はい
ndb_optimization_delay	はい	はい	はい	グローバル	はい
ndb_optimized_node_selection	はい	はい	はい	グローバル	いいえ
ndb_read_backup	はい	はい	はい	グローバル	はい
ndb_recv_thread_activation_threshold	はい	はい	はい	グローバル	はい
ndb_recv_thread_control_mask	はい	はい	はい	グローバル	はい
Ndb_replica_max_replicated_epoch			はい	グローバル	いいえ
ndb_report_threshold_log_epoch_slip	はい	はい	はい	グローバル	はい
ndb_report_threshold_log_mem_usage	はい	はい	はい	グローバル	はい
ndb_row_checksum			はい	両方	はい
ndb_schema_dist_lock_wait_timeout	はい	はい	はい	グローバル	はい
ndb_schema_dist_lock_timeout	はい	はい	はい	グローバル	いいえ
ndb_schema_dist_lock_timeout	はい	はい	はい	グローバル	いいえ
ndb_schema_dist_update_allowed	はい	はい	はい	グローバル	いいえ
ndb_show_foreign_keys_mock_tables	はい	はい	はい	グローバル	はい
ndb_slave_conflict_resolution	はい	はい	はい	グローバル	はい
Ndb_slave_max_replicated_epoch			はい	グローバル	いいえ
Ndb_system_name			はい	グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	変数スコープ	動的
ndb_table_no_logging			はい	セッション	はい
ndb_table_temporary			はい	セッション	はい
ndb_use_copying_alter_table			はい	両方	いいえ
ndb_use_exact_count			はい	両方	はい
ndb_use_transactions	はい	はい	はい	両方	はい
ndb_version			はい	グローバル	いいえ
ndb_version_string			はい	グローバル	いいえ
ndb_wait_connected	はい	はい	はい	グローバル	いいえ
ndb_wait_setup	はい	はい	はい	グローバル	いいえ
ndbinfo_database			はい	グローバル	いいえ
ndbinfo_max_bytes	はい		はい	両方	はい
ndbinfo_max_rows	はい		はい	両方	はい
ndbinfo_offline			はい	グローバル	はい
ndbinfo_show_hidden	はい		はい	両方	はい
ndbinfo_table_prefix			はい	グローバル	いいえ
ndbinfo_version			はい	グローバル	いいえ
net_buffer_length	はい	はい	はい	両方	はい
net_read_timeout	はい	はい	はい	両方	はい
net_retry_count	はい	はい	はい	両方	はい
net_write_timeout	はい	はい	はい	両方	はい
new	はい	はい	はい	両方	はい
ngram_token_size	はい	はい	はい	グローバル	いいえ
offline_mode	はい	はい	はい	グローバル	はい
old	はい	はい	はい	グローバル	いいえ
old_alter_table	はい	はい	はい	両方	はい
open_files_limit	はい	はい	はい	グローバル	いいえ
optimizer_prune_level	はい	はい	はい	両方	はい
optimizer_search_depth	はい	はい	はい	両方	はい
optimizer_switch	はい	はい	はい	両方	はい
optimizer_trace	はい	はい	はい	両方	はい
optimizer_trace_features	はい	はい	はい	両方	はい
optimizer_trace_limit	はい	はい	はい	両方	はい
optimizer_trace_max_mem_size	はい	はい	はい	両方	はい
optimizer_trace_offs	はい	はい	はい	両方	はい
original_commit_timestamp			はい	セッション	はい
original_server_version			はい	セッション	はい
parser_max_mem_size	はい	はい	はい	両方	はい
partial_revokes	はい	はい	はい	グローバル	はい
password_history	はい	はい	はい	グローバル	はい
password_require_content	はい	はい	はい	グローバル	はい

名前	コマンド行	オプションファイル	システム変数	変数スコープ	動的
password_reuse_interval	はい	はい	はい	グローバル	はい
performance_schema	はい	はい	はい	グローバル	いいえ
performance_schema_accounts_size	はい	はい	はい	グローバル	いいえ
performance_schema_digests_size	はい	はい	はい	グローバル	いいえ
performance_schema_error_size	はい	はい	はい	グローバル	いいえ
performance_schema_events_stages_history_long_size	はい	はい	はい	グローバル	いいえ
performance_schema_events_stages_history_size	はい	はい	はい	グローバル	いいえ
performance_schema_events_statements_history_long_size	はい	はい	はい	グローバル	いいえ
performance_schema_events_statements_history_size	はい	はい	はい	グローバル	いいえ
performance_schema_events_transactions_history_long_size	はい	はい	はい	グローバル	いいえ
performance_schema_events_transactions_history_size	はい	はい	はい	グローバル	いいえ
performance_schema_events_waits_history_long_size	はい	はい	はい	グローバル	いいえ
performance_schema_events_waits_history_size	はい	はい	はい	グローバル	いいえ
performance_schema_hosts_size	はい	はい	はい	グローバル	いいえ
performance_schema_max_cond_classes	はい	はい	はい	グローバル	いいえ
performance_schema_max_cond_instances	はい	はい	はい	グローバル	いいえ
performance_schema_max_digest_length	はい	はい	はい	グローバル	いいえ
performance_schema_max_digest_sample_age	はい	はい	はい	グローバル	はい
performance_schema_max_file_classes	はい	はい	はい	グローバル	いいえ
performance_schema_max_file_handles	はい	はい	はい	グローバル	いいえ
performance_schema_max_file_instances	はい	はい	はい	グローバル	いいえ
performance_schema_max_index_statistics	はい	はい	はい	グローバル	いいえ
performance_schema_max_memory_classes	はい	はい	はい	グローバル	いいえ
performance_schema_max_metadata_instances	はい	はい	はい	グローバル	いいえ
performance_schema_max_mutex_classes	はい	はい	はい	グローバル	いいえ
performance_schema_max_mutex_instances	はい	はい	はい	グローバル	いいえ
performance_schema_max_prepared_statements_instances	はい	はい	はい	グローバル	いいえ
performance_schema_max_program_instances	はい	はい	はい	グローバル	いいえ
performance_schema_max_rwlock_classes	はい	はい	はい	グローバル	いいえ
performance_schema_max_rwlock_instances	はい	はい	はい	グローバル	いいえ
performance_schema_max_socket_classes	はい	はい	はい	グローバル	いいえ
performance_schema_max_socket_instances	はい	はい	はい	グローバル	いいえ
performance_schema_max_sql_text_length	はい	はい	はい	グローバル	いいえ
performance_schema_max_stage_classes	はい	はい	はい	グローバル	いいえ
performance_schema_max_statement_classes	はい	はい	はい	グローバル	いいえ
performance_schema_max_statement_stack	はい	はい	はい	グローバル	いいえ
performance_schema_max_table_handles	はい	はい	はい	グローバル	いいえ
performance_schema_max_table_instances	はい	はい	はい	グローバル	いいえ
performance_schema_max_table_locks	はい	はい	はい	グローバル	いいえ
performance_schema_max_thread_classes	はい	はい	はい	グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	変数スコープ	動的
performance_schema_max_thread_instances	はい	はい	はい	グローバル	いいえ
performance_schema_session_connect_attrs_size	はい	はい	はい	グローバル	いいえ
performance_schema_setup_actors_size	はい	はい	はい	グローバル	いいえ
performance_schema_setup_objects_size	はい	はい	はい	グローバル	いいえ
performance_schema_show_processlist	はい	はい	はい	グローバル	はい
performance_schema_users_size	はい	はい	はい	グローバル	いいえ
persist_only_administrators	はい	はい	はい	グローバル	いいえ
persisted_globals_load	はい	はい	はい	グローバル	いいえ
pid_file	はい	はい	はい	グローバル	いいえ
plugin_dir	はい	はい	はい	グローバル	いいえ
plugin_load	はい	はい	はい	グローバル	いいえ
plugin_load_add	はい	はい	はい	グローバル	いいえ
port	はい	はい	はい	グローバル	いいえ
preload_buffer_size	はい	はい	はい	両方	はい
print_identified_with_hex	はい	はい	はい	両方	はい
profiling			はい	両方	はい
profiling_history_size	はい	はい	はい	両方	はい
protocol_compression_algorithms	はい	はい	はい	グローバル	はい
protocol_version			はい	グローバル	いいえ
proxy_user			はい	セッション	いいえ
pseudo_slave_mode			はい	セッション	はい
pseudo_thread_id			はい	セッション	はい
query_alloc_block_size	はい	はい	はい	両方	はい
query_prealloc_size	はい	はい	はい	両方	はい
rand_seed1			はい	セッション	はい
rand_seed2			はい	セッション	はい
range_alloc_block_size	はい	はい	はい	両方	はい
range_optimizer_max_mem_size	はい	はい	はい	両方	はい
rbr_exec_mode			はい	両方	はい
read_buffer_size	はい	はい	はい	両方	はい
read_only	はい	はい	はい	グローバル	はい
read_rnd_buffer_size	はい	はい	はい	両方	はい
regexp_stack_limit	はい	はい	はい	グローバル	はい
regexp_time_limit	はい	はい	はい	グローバル	はい
relay_log	はい	はい	はい	グローバル	いいえ
relay_log_basename			はい	グローバル	いいえ
relay_log_index	はい	はい	はい	グローバル	いいえ
relay_log_info_file	はい	はい	はい	グローバル	いいえ
relay_log_info_repository	はい	はい	はい	グローバル	はい
relay_log_purge	はい	はい	はい	グローバル	はい

名前	コマンド行	オプションファイル	システム変数	変数スコープ	動的
relay_log_recovery	はい	はい	はい	グローバル	いいえ
relay_log_space_limit	はい	はい	はい	グローバル	いいえ
replication_optimize_binlog	はい	はい	はい	グローバル	はい
replication_sender_heartbeat_commit_timeout	はい	はい	はい	グローバル	はい
report_host	はい	はい	はい	グローバル	いいえ
report_password	はい	はい	はい	グローバル	いいえ
report_port	はい	はい	はい	グローバル	いいえ
report_user	はい	はい	はい	グローバル	いいえ
require_row_format			はい	セッション	はい
require_secure_transport	はい	はい	はい	グローバル	はい
resultset_metadata			はい	セッション	はい
rewriter_enabled			はい	グローバル	はい
rewriter_verbose			はい	グローバル	はい
rpl_read_size	はい	はい	はい	グローバル	はい
rpl_semi_sync_master_enabled	はい	はい	はい	グローバル	はい
rpl_semi_sync_master_timeout	はい	はい	はい	グローバル	はい
rpl_semi_sync_master_trace_level	はい	はい	はい	グローバル	はい
rpl_semi_sync_master_wait_for_slave_close	はい	はい	はい	グローバル	はい
rpl_semi_sync_master_wait_no_slave	はい	はい	はい	グローバル	はい
rpl_semi_sync_master_wait_point	はい	はい	はい	グローバル	はい
rpl_semi_sync_slave_enabled	はい	はい	はい	グローバル	はい
rpl_semi_sync_slave_trace_level	はい	はい	はい	グローバル	はい
rpl_stop_slave_timeout	はい	はい	はい	グローバル	はい
schema_definition_cache	はい	はい	はい	グローバル	はい
secondary_engine_cost_threshold			はい	セッション	はい
secure_file_priv	はい	はい	はい	グローバル	いいえ
select_into_buffer_size	はい	はい	はい	両方	はい
select_into_disk_sync	はい	はい	はい	両方	はい
select_into_disk_sync_delay	はい	はい	はい	両方	はい
server_id	はい	はい	はい	グローバル	はい
server_id_bits	はい	はい	はい	グローバル	いいえ
server_uuid			はい	グローバル	いいえ
session_track_gtids	はい	はい	はい	両方	はい
session_track_schema	はい	はい	はい	両方	はい
session_track_state_change	はい	はい	はい	両方	はい
session_track_system_variables	はい	はい	はい	両方	はい
session_track_transaction_info	はい	はい	はい	両方	はい
sha256_password_generate_rsa_key	はい	はい	はい	グローバル	いいえ
sha256_password_private_key_path	はい	はい	はい	グローバル	いいえ
sha256_password_rsa_key_users	はい	はい	はい	グローバル	はい

名前	コマンド行	オプションファイル	システム変数	変数スコープ	動的
sha256_password_plugin_key_path	はい	はい	はい	グローバル	いいえ
shared_memory	はい	はい	はい	グローバル	いいえ
shared_memory_base_name	はい	はい	はい	グローバル	いいえ
show_create_table_skip_secondary_engine	はい	はい	はい	セッション	はい
show_create_table_skip_innodb	はい	はい	はい	両方	はい
show_old_temporals	はい	はい	はい	両方	はい
skip_external_locking	はい	はい	はい	グローバル	いいえ
skip_name_resolve	はい	はい	はい	グローバル	いいえ
skip_networking	はい	はい	はい	グローバル	いいえ
skip_show_databases	はい	はい	はい	グローバル	いいえ
slave_allow_batching	はい	はい	はい	グローバル	はい
slave_checkpoint_group	はい	はい	はい	グローバル	はい
slave_checkpoint_period	はい	はい	はい	グローバル	はい
slave_compressed_protocol	はい	はい	はい	グローバル	はい
slave_exec_mode	はい	はい	はい	グローバル	はい
slave_load_tmpdir	はい	はい	はい	グローバル	いいえ
slave_max_allowed_packet	はい	はい	はい	グローバル	はい
slave_net_timeout	はい	はい	はい	グローバル	はい
slave_parallel_type	はい	はい	はい	グローバル	はい
slave_parallel_workers	はい	はい	はい	グローバル	はい
slave_pending_jobs_size_max	はい	はい	はい	グローバル	はい
slave_preserve_commit_order	はい	はい	はい	グローバル	はい
slave_rows_search_algorithms	はい	はい	はい	グローバル	はい
slave_skip_errors	はい	はい	はい	グローバル	いいえ
slave_sql_verify_checksum	はい	はい	はい	グローバル	はい
slave_transaction_retries	はい	はい	はい	グローバル	はい
slave_type_converters	はい	はい	はい	グローバル	はい
slow_launch_time	はい	はい	はい	グローバル	はい
slow_query_log	はい	はい	はい	グローバル	はい
slow_query_log_file	はい	はい	はい	グローバル	はい
socket	はい	はい	はい	グローバル	いいえ
sort_buffer_size	はい	はい	はい	両方	はい
sql_auto_is_null			はい	両方	はい
sql_big_selects			はい	両方	はい
sql_buffer_result			はい	両方	はい
sql_log_bin			はい	セッション	はい
sql_log_off			はい	両方	はい
sql_mode	はい	はい	はい	両方	はい
sql_notes			はい	両方	はい
sql_quote_show_create			はい	両方	はい

名前	コマンド行	オプションファイル	システム変数	変数スコープ	動的
sql_require_primary_key	はい	はい	はい	両方	はい
sql_safe_updates			はい	両方	はい
sql_select_limit			はい	両方	はい
sql_slave_skip_counter			はい	グローバル	はい
sql_warnings			はい	両方	はい
ssl_ca	はい	はい	はい	グローバル	異なる
ssl_capath	はい	はい	はい	グローバル	異なる
ssl_cert	はい	はい	はい	グローバル	異なる
ssl_cipher	はい	はい	はい	グローバル	異なる
ssl_crl	はい	はい	はい	グローバル	異なる
ssl_crlpath	はい	はい	はい	グローバル	異なる
ssl_fips_mode	はい	はい	はい	グローバル	はい
ssl_key	はい	はい	はい	グローバル	異なる
stored_program_cache	はい	はい	はい	グローバル	はい
stored_program_definition_cache	はい	はい	はい	グローバル	はい
super_read_only	はい	はい	はい	グローバル	はい
sync_binlog	はい	はい	はい	グローバル	はい
sync_master_info	はい	はい	はい	グローバル	はい
sync_relay_log	はい	はい	はい	グローバル	はい
sync_relay_log_info	はい	はい	はい	グローバル	はい
syseventlog.facility	はい	はい	はい	グローバル	はい
syseventlog.include_binlog	はい	はい	はい	グローバル	はい
syseventlog.tag	はい	はい	はい	グローバル	はい
system_time_zone			はい	グローバル	いいえ
table_definition_cache	はい	はい	はい	グローバル	はい
table_encryption_privilege_check	はい	はい	はい	グローバル	はい
table_open_cache	はい	はい	はい	グローバル	はい
table_open_cache_instances	はい	はい	はい	グローバル	いいえ
tablespace_definition_cache	はい	はい	はい	グローバル	はい
temptable_max_memory	はい	はい	はい	グローバル	はい
temptable_max_ram_size	はい	はい	はい	グローバル	はい
temptable_use_memory	はい	はい	はい	グローバル	はい
thread_cache_size	はい	はい	はい	グローバル	はい
thread_handling	はい	はい	はい	グローバル	いいえ
thread_pool_algorithm	はい	はい	はい	グローバル	いいえ
thread_pool_high_priority_connection	はい	はい	はい	両方	はい
thread_pool_max_active_query_threads	はい	はい	はい	グローバル	はい
thread_pool_max_user_connections	はい	はい	はい	グローバル	はい
thread_pool_priority_timer	はい	はい	はい	両方	はい
thread_pool_size	はい	はい	はい	グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	変数スコープ	動的
thread_pool_stall_limit	はい	はい	はい	グローバル	はい
thread_stack	はい	はい	はい	グローバル	いいえ
time_zone			はい	両方	はい
timestamp			はい	セッション	はい
tls_ciphersuites	はい	はい	はい	グローバル	はい
tls_version	はい	はい	はい	グローバル	異なる
tmp_table_size	はい	はい	はい	両方	はい
tmpdir	はい	はい	はい	グローバル	いいえ
transaction_alloc_block_size	はい	はい	はい	両方	はい
transaction_allow_batching			はい	セッション	はい
transaction_isolation	はい	はい	はい	両方	はい
transaction_prealloc_size	はい	はい	はい	両方	はい
transaction_read_only	はい	はい	はい	両方	はい
transaction_write_set_extraction	はい	はい	はい	両方	はい
unique_checks			はい	両方	はい
updatable_views_with_limit	はい	はい	はい	両方	はい
use_secondary_engine			はい	セッション	はい
validate_password_check_user_name	はい	はい	はい	グローバル	はい
validate_password_dictionary_file	はい	はい	はい	グローバル	はい
validate_password_length	はい	はい	はい	グローバル	はい
validate_password_lower_case_count	はい	はい	はい	グローバル	はい
validate_password_number_count	はい	はい	はい	グローバル	はい
validate_password_policy	はい	はい	はい	グローバル	はい
validate_password_special_char_count	はい	はい	はい	グローバル	はい
validate_password_check_user_name	はい	はい	はい	グローバル	はい
validate_password_dictionary_file	はい	はい	はい	グローバル	はい
validate_password_length	はい	はい	はい	グローバル	はい
validate_password_lower_case_count	はい	はい	はい	グローバル	はい
validate_password_number_count	はい	はい	はい	グローバル	はい
validate_password_policy	はい	はい	はい	グローバル	はい
validate_password_special_char_count	はい	はい	はい	グローバル	はい
validate_user_plugins	はい	はい	はい	グローバル	いいえ
version			はい	グローバル	いいえ
version_comment			はい	グローバル	いいえ
version_compile_machine			はい	グローバル	いいえ
version_compile_os			はい	グローバル	いいえ
version_compile_zlib			はい	グローバル	いいえ
version_tokens_session	はい	はい	はい	両方	はい
version_tokens_session_number	はい	はい	はい	両方	いいえ
wait_timeout	はい	はい	はい	両方	はい

名前	コマンド行	オプションファイル	システム変数	変数スコープ	動的
warning_count			はい	セッション	いいえ
windowing_use_high_precision	はい	はい	はい	両方	はい

Notes:

1. このオプションは動的ですが、サーバーによってのみ設定する必要があります。この変数は手動で設定しないでください。

5.1.6 サーバーステータス変数リファレンス

次のテーブルに、`mysqld` 内で適用可能なすべてのステータス変数を示します。

このテーブルは、各変数のデータ型とスコープを示しています。最後の列は、各変数のスコープがグローバル、セッション、またはその両方のいずれであるかを示します。変数の設定および使用の詳細は、対応するアイテムの説明を参照してください。必要に応じて、アイテムに関する詳細情報へのダイレクトリンクが提供されます。

表 5.3 「ステータス変数サマリー」

変数名	変数型	変数スコープ
Aborted_clients	Integer	グローバル
Aborted_connects	Integer	グローバル
Acl_cache_items_count	Integer	グローバル
Audit_log_current_size	Integer	グローバル
Audit_log_event_max_drop_size	Integer	グローバル
Audit_log_events	Integer	グローバル
Audit_log_events_filtered	Integer	グローバル
Audit_log_events_lost	Integer	グローバル
Audit_log_events_written	Integer	グローバル
Audit_log_total_size	Integer	グローバル
Audit_log_write_waits	Integer	グローバル
Authentication_ldap_sasl_supported_mechanisms	文字列	グローバル
Binlog_cache_disk_use	Integer	グローバル
Binlog_cache_use	Integer	グローバル
Binlog_stmt_cache_disk_use	Integer	グローバル
Binlog_stmt_cache_use	Integer	グローバル
Bytes_received	Integer	両方
Bytes_sent	Integer	両方
Caching_sha2_password_rsa_public_key	文字列	グローバル
Com_admin_commands	Integer	両方
Com_alter_db	Integer	両方
Com_alter_event	Integer	両方
Com_alter_function	Integer	両方
Com_alter_procedure	Integer	両方
Com_alter_resource_group	Integer	グローバル
Com_alter_server	Integer	両方
Com_alter_table	Integer	両方

変数名	変数型	変数スコープ
Com_alter_tablespace	Integer	両方
Com_alter_user	Integer	両方
Com_alter_user_default_role	Integer	グローバル
Com_analyze	Integer	両方
Com_assign_to_keycache	Integer	両方
Com_begin	Integer	両方
Com_binlog	Integer	両方
Com_call_procedure	Integer	両方
Com_change_db	Integer	両方
Com_change_master	Integer	両方
Com_change_repl_filter	Integer	両方
Com_check	Integer	両方
Com_checksum	Integer	両方
Com_clone	Integer	グローバル
Com_commit	Integer	両方
Com_create_db	Integer	両方
Com_create_event	Integer	両方
Com_create_function	Integer	両方
Com_create_index	Integer	両方
Com_create_procedure	Integer	両方
Com_create_resource_group	Integer	グローバル
Com_create_role	Integer	グローバル
Com_create_server	Integer	両方
Com_create_table	Integer	両方
Com_create_trigger	Integer	両方
Com_create_udf	Integer	両方
Com_create_user	Integer	両方
Com_create_view	Integer	両方
Com_dealloc_sql	Integer	両方
Com_delete	Integer	両方
Com_delete_multi	Integer	両方
Com_do	Integer	両方
Com_drop_db	Integer	両方
Com_drop_event	Integer	両方
Com_drop_function	Integer	両方
Com_drop_index	Integer	両方
Com_drop_procedure	Integer	両方
Com_drop_resource_group	Integer	グローバル
Com_drop_role	Integer	グローバル
Com_drop_server	Integer	両方
Com_drop_table	Integer	両方

変数名	変数型	変数スコープ
Com_drop_trigger	Integer	両方
Com_drop_user	Integer	両方
Com_drop_view	Integer	両方
Com_empty_query	Integer	両方
Com_execute_sql	Integer	両方
Com_explain_other	Integer	両方
Com_flush	Integer	両方
Com_get_diagnostics	Integer	両方
Com_grant	Integer	両方
Com_grant_roles	Integer	グローバル
Com_group_replication_start	Integer	グローバル
Com_group_replication_stop	Integer	グローバル
Com_ha_close	Integer	両方
Com_ha_open	Integer	両方
Com_ha_read	Integer	両方
Com_help	Integer	両方
Com_insert	Integer	両方
Com_insert_select	Integer	両方
Com_install_component	Integer	グローバル
Com_install_plugin	Integer	両方
Com_kill	Integer	両方
Com_load	Integer	両方
Com_lock_tables	Integer	両方
Com_optimize	Integer	両方
Com_preload_keys	Integer	両方
Com_prepare_sql	Integer	両方
Com_purge	Integer	両方
Com_purge_before_date	Integer	両方
Com_release_savepoint	Integer	両方
Com_rename_table	Integer	両方
Com_rename_user	Integer	両方
Com_repair	Integer	両方
Com_replace	Integer	両方
Com_replace_select	Integer	両方
Com_replica_start	Integer	両方
Com_replica_stop	Integer	両方
Com_reset	Integer	両方
Com_resignal	Integer	両方
Com_restart	Integer	両方
Com_revoke	Integer	両方
Com_revoke_all	Integer	両方

変数名	変数型	変数スコープ
Com_revoke_roles	Integer	グローバル
Com_rollback	Integer	両方
Com_rollback_to_savepoint	Integer	両方
Com_savepoint	Integer	両方
Com_select	Integer	両方
Com_set_option	Integer	両方
Com_set_resource_group	Integer	グローバル
Com_set_role	Integer	グローバル
Com_show_authors	Integer	両方
Com_show_binlog_events	Integer	両方
Com_show_binlogs	Integer	両方
Com_show_charsets	Integer	両方
Com_show_collations	Integer	両方
Com_show_contributors	Integer	両方
Com_show_create_db	Integer	両方
Com_show_create_event	Integer	両方
Com_show_create_func	Integer	両方
Com_show_create_proc	Integer	両方
Com_show_create_table	Integer	両方
Com_show_create_trigger	Integer	両方
Com_show_create_user	Integer	両方
Com_show_databases	Integer	両方
Com_show_engine_logs	Integer	両方
Com_show_engine_mutex	Integer	両方
Com_show_engine_status	Integer	両方
Com_show_errors	Integer	両方
Com_show_events	Integer	両方
Com_show_fields	Integer	両方
Com_show_function_code	Integer	両方
Com_show_function_status	Integer	両方
Com_show_grants	Integer	両方
Com_show_keys	Integer	両方
Com_show_master_status	Integer	両方
Com_show_ndb_status	Integer	両方
Com_show_open_tables	Integer	両方
Com_show_plugins	Integer	両方
Com_show_privileges	Integer	両方
Com_show_procedure_code	Integer	両方
Com_show_procedure_status	Integer	両方
Com_show_processlist	Integer	両方
Com_show_profile	Integer	両方

変数名	変数型	変数スコープ
Com_show_profiles	Integer	両方
Com_show_relaylog_events	Integer	両方
Com_show_replica_status	Integer	両方
Com_show_replicas	Integer	両方
Com_show_slave_hosts	Integer	両方
Com_show_slave_status	Integer	両方
Com_show_status	Integer	両方
Com_show_storage_engines	Integer	両方
Com_show_table_status	Integer	両方
Com_show_tables	Integer	両方
Com_show_triggers	Integer	両方
Com_show_variables	Integer	両方
Com_show_warnings	Integer	両方
Com_shutdown	Integer	両方
Com_signal	Integer	両方
Com_slave_start	Integer	両方
Com_slave_stop	Integer	両方
Com_stmt_close	Integer	両方
Com_stmt_execute	Integer	両方
Com_stmt_fetch	Integer	両方
Com_stmt_prepare	Integer	両方
Com_stmt_reprepare	Integer	両方
Com_stmt_reset	Integer	両方
Com_stmt_send_long_data	Integer	両方
Com_truncate	Integer	両方
Com_uninstall_component	Integer	グローバル
Com_uninstall_plugin	Integer	両方
Com_unlock_tables	Integer	両方
Com_update	Integer	両方
Com_update_multi	Integer	両方
Com_xa_commit	Integer	両方
Com_xa_end	Integer	両方
Com_xa_prepare	Integer	両方
Com_xa_recover	Integer	両方
Com_xa_rollback	Integer	両方
Com_xa_start	Integer	両方
Compression	Integer	セッション
Compression_algorithm	文字列	グローバル
Compression_level	Integer	グローバル
Connection_control_delay_generated	Integer	グローバル
Connection_errors_accept	Integer	グローバル

変数名	変数型	変数スコープ
Connection_errors_internal	Integer	グローバル
Connection_errors_max_connections	Integer	グローバル
Connection_errors_peer_address	Integer	グローバル
Connection_errors_select	Integer	グローバル
Connection_errors_tcpwrap	Integer	グローバル
Connections	Integer	グローバル
Created_tmp_disk_tables	Integer	両方
Created_tmp_files	Integer	グローバル
Created_tmp_tables	Integer	両方
Current_tls_ca	ファイル名	グローバル
Current_tls_capath	ディレクトリ名	グローバル
Current_tls_cert	ファイル名	グローバル
Current_tls_cipher	文字列	グローバル
Current_tls_ciphersuites	文字列	グローバル
Current_tls_crl	ファイル名	グローバル
Current_tls_crlpath	ディレクトリ名	グローバル
Current_tls_key	ファイル名	グローバル
Current_tls_version	文字列	グローバル
Delayed_errors	Integer	グローバル
Delayed_insert_threads	Integer	グローバル
Delayed_writes	Integer	グローバル
dragnet.Status	文字列	グローバル
Error_log_buffered_bytes	Integer	グローバル
Error_log_buffered_events	Integer	グローバル
Error_log_expired_events	Integer	グローバル
Error_log_latest_write	Integer	グローバル
Firewall_access_denied	Integer	グローバル
Firewall_access_granted	Integer	グローバル
Firewall_cached_entries	Integer	グローバル
Flush_commands	Integer	グローバル
group_replication_primary_member	文字列	グローバル
Handler_commit	Integer	両方
Handler_delete	Integer	両方
Handler_discover	Integer	両方
Handler_external_lock	Integer	両方
Handler_mrr_init	Integer	両方
Handler_prepare	Integer	両方
Handler_read_first	Integer	両方
Handler_read_key	Integer	両方
Handler_read_last	Integer	両方
Handler_read_next	Integer	両方

変数名	変数型	変数スコープ
Handler_read_prev	Integer	両方
Handler_read_rnd	Integer	両方
Handler_read_rnd_next	Integer	両方
Handler_rollback	Integer	両方
Handler_savepoint	Integer	両方
Handler_savepoint_rollback	Integer	両方
Handler_update	Integer	両方
Handler_write	Integer	両方
Innodb_buffer_pool_bytes_data	Integer	グローバル
Innodb_buffer_pool_bytes_dirty	Integer	グローバル
Innodb_buffer_pool_dump_status	文字列	グローバル
Innodb_buffer_pool_load_status	文字列	グローバル
Innodb_buffer_pool_pages_data	Integer	グローバル
Innodb_buffer_pool_pages_dirty	Integer	グローバル
Innodb_buffer_pool_pages_flushed	Integer	グローバル
Innodb_buffer_pool_pages_free	Integer	グローバル
Innodb_buffer_pool_pages_latched	Integer	グローバル
Innodb_buffer_pool_pages_misc	Integer	グローバル
Innodb_buffer_pool_pages_total	Integer	グローバル
Innodb_buffer_pool_read_ahead	Integer	グローバル
Innodb_buffer_pool_read_ahead_evicted	Integer	グローバル
Innodb_buffer_pool_read_ahead_rnd	Integer	グローバル
Innodb_buffer_pool_read_requests	Integer	グローバル
Innodb_buffer_pool_reads	Integer	グローバル
Innodb_buffer_pool_resize_status	文字列	グローバル
Innodb_buffer_pool_wait_free	Integer	グローバル
Innodb_buffer_pool_write_requests	Integer	グローバル
Innodb_data_fsyncs	Integer	グローバル
Innodb_data_pending_fsyncs	Integer	グローバル
Innodb_data_pending_reads	Integer	グローバル
Innodb_data_pending_writes	Integer	グローバル
Innodb_data_read	Integer	グローバル
Innodb_data_reads	Integer	グローバル
Innodb_data_writes	Integer	グローバル
Innodb_data_written	Integer	グローバル
Innodb_dblwr_pages_written	Integer	グローバル
Innodb_dblwr_writes	Integer	グローバル
Innodb_have_atomic_builtins	Integer	グローバル
Innodb_log_waits	Integer	グローバル
Innodb_log_write_requests	Integer	グローバル
Innodb_log_writes	Integer	グローバル

変数名	変数型	変数スコープ
InnoDB_num_open_files	Integer	グローバル
InnoDB_os_log_fsyncls	Integer	グローバル
InnoDB_os_log_pending_fsyncls	Integer	グローバル
InnoDB_os_log_pending_writes	Integer	グローバル
InnoDB_os_log_written	Integer	グローバル
InnoDB_page_size	Integer	グローバル
InnoDB_pages_created	Integer	グローバル
InnoDB_pages_read	Integer	グローバル
InnoDB_pages_written	Integer	グローバル
InnoDB_redo_log_enabled	Boolean	グローバル
InnoDB_row_lock_current_waits	Integer	グローバル
InnoDB_row_lock_time	Integer	グローバル
InnoDB_row_lock_time_avg	Integer	グローバル
InnoDB_row_lock_time_max	Integer	グローバル
InnoDB_row_lock_waits	Integer	グローバル
InnoDB_rows_deleted	Integer	グローバル
InnoDB_rows_inserted	Integer	グローバル
InnoDB_rows_read	Integer	グローバル
InnoDB_rows_updated	Integer	グローバル
InnoDB_system_rows_deleted	Integer	グローバル
InnoDB_system_rows_inserted	Integer	グローバル
InnoDB_system_rows_read	Integer	グローバル
InnoDB_truncated_status_writes	Integer	グローバル
InnoDB_undo_tablespaces_active	Integer	グローバル
InnoDB_undo_tablespaces_explicit	Integer	グローバル
InnoDB_undo_tablespaces_implicit	Integer	グローバル
InnoDB_undo_tablespaces_total	Integer	グローバル
Key_blocks_not_flushed	Integer	グローバル
Key_blocks_unused	Integer	グローバル
Key_blocks_used	Integer	グローバル
Key_read_requests	Integer	グローバル
Key_reads	Integer	グローバル
Key_write_requests	Integer	グローバル
Key_writes	Integer	グローバル
Last_query_cost	数値	セッション
Last_query_partial_plans	Integer	セッション
Locked_connects	Integer	グローバル
Max_execution_time_exceeded	Integer	両方
Max_execution_time_set	Integer	両方
Max_execution_time_set_failed	Integer	両方
Max_used_connections	Integer	グローバル

変数名	変数型	変数スコープ
Max_used_connections_time	日付時間	グローバル
mecab_charset	文字列	グローバル
Mysqlx_aborted_clients	Integer	グローバル
Mysqlx_address	文字列	グローバル
Mysqlx_bytes_received	Integer	両方
Mysqlx_bytes_received_compressed_payload	Integer	両方
Mysqlx_bytes_received_uncompressed_payload	Integer	両方
Mysqlx_bytes_sent	Integer	両方
Mysqlx_bytes_sent_compressed_payload	Integer	両方
Mysqlx_bytes_sent_uncompressed_payload	Integer	両方
Mysqlx_compression_algorithm	文字列	セッション
Mysqlx_compression_level	文字列	セッション
Mysqlx_connection_accept_errors	Integer	両方
Mysqlx_connection_errors	Integer	両方
Mysqlx_connections_accepted	Integer	グローバル
Mysqlx_connections_closed	Integer	グローバル
Mysqlx_connections_rejected	Integer	グローバル
Mysqlx_crud_create_view	Integer	両方
Mysqlx_crud_delete	Integer	両方
Mysqlx_crud_drop_view	Integer	両方
Mysqlx_crud_find	Integer	両方
Mysqlx_crud_insert	Integer	両方
Mysqlx_crud_modify_view	Integer	両方
Mysqlx_crud_update	Integer	両方
Mysqlx_cursor_close	Integer	両方
Mysqlx_cursor_fetch	Integer	両方
Mysqlx_cursor_open	Integer	両方
Mysqlx_errors_sent	Integer	両方
Mysqlx_errors_unknown_message_type	Integer	両方
Mysqlx_expect_close	Integer	両方
Mysqlx_expect_open	Integer	両方
Mysqlx_init_error	Integer	両方
Mysqlx_messages_sent	Integer	両方
Mysqlx_notice_global_sent	Integer	両方
Mysqlx_notice_other_sent	Integer	両方
Mysqlx_notice_warning_sent	Integer	両方
Mysqlx_notified_by_group_replication	Integer	両方
Mysqlx_port	文字列	グローバル
Mysqlx_prep_deallocate	Integer	両方
Mysqlx_prep_execute	Integer	両方
Mysqlx_prep_prepare	Integer	両方

変数名	変数型	変数スコープ
Mysqlx_rows_sent	Integer	両方
Mysqlx_sessions	Integer	グローバル
Mysqlx_sessions_accepted	Integer	グローバル
Mysqlx_sessions_closed	Integer	グローバル
Mysqlx_sessions_fatal_error	Integer	グローバル
Mysqlx_sessions_killed	Integer	グローバル
Mysqlx_sessions_rejected	Integer	グローバル
Mysqlx_socket	文字列	グローバル
Mysqlx_ssl_accept_renegotiates	Integer	グローバル
Mysqlx_ssl_accepts	Integer	グローバル
Mysqlx_ssl_active	Integer	両方
Mysqlx_ssl_cipher	Integer	両方
Mysqlx_ssl_cipher_list	Integer	両方
Mysqlx_ssl_ctx_verify_depth	Integer	両方
Mysqlx_ssl_ctx_verify_mode	Integer	両方
Mysqlx_ssl_finished_accepts	Integer	グローバル
Mysqlx_ssl_server_not_after	Integer	グローバル
Mysqlx_ssl_server_not_before	Integer	グローバル
Mysqlx_ssl_verify_depth	Integer	グローバル
Mysqlx_ssl_verify_mode	Integer	グローバル
Mysqlx_ssl_version	Integer	両方
Mysqlx_stmt_create_collection	Integer	両方
Mysqlx_stmt_create_collection_index	Integer	両方
Mysqlx_stmt_disable_notices	Integer	両方
Mysqlx_stmt_drop_collection	Integer	両方
Mysqlx_stmt_drop_collection_index	Integer	両方
Mysqlx_stmt_enable_notices	Integer	両方
Mysqlx_stmt_ensure_collection	文字列	両方
Mysqlx_stmt_execute_mysqlx	Integer	両方
Mysqlx_stmt_execute_sql	Integer	両方
Mysqlx_stmt_execute_xplugin	Integer	両方
Mysqlx_stmt_get_collection_options	Integer	両方
Mysqlx_stmt_kill_client	Integer	両方
Mysqlx_stmt_list_clients	Integer	両方
Mysqlx_stmt_list_notices	Integer	両方
Mysqlx_stmt_list_objects	Integer	両方
Mysqlx_stmt_modify_collection_options	Integer	両方
Mysqlx_stmt_ping	Integer	両方
Mysqlx_worker_threads	Integer	グローバル
Mysqlx_worker_threads_active	Integer	グローバル
Ndb_api_adaptive_send_deferred_count	Integer	グローバル

変数名	変数型	変数スコープ
Ndb_api_adaptive_send_deferred_count	Integer	グローバル
Ndb_api_adaptive_send_deferred_count_replica	Integer	グローバル
Ndb_api_adaptive_send_deferred_count_session	Integer	グローバル
Ndb_api_adaptive_send_deferred_count_slave	Integer	グローバル
Ndb_api_adaptive_send_forced_count	Integer	グローバル
Ndb_api_adaptive_send_forced_count_replica	Integer	グローバル
Ndb_api_adaptive_send_forced_count_session	Integer	グローバル
Ndb_api_adaptive_send_forced_count_slave	Integer	グローバル
Ndb_api_adaptive_send_unforced_count	Integer	グローバル
Ndb_api_adaptive_send_unforced_count_replica	Integer	グローバル
Ndb_api_adaptive_send_unforced_count_session	Integer	グローバル
Ndb_api_adaptive_send_unforced_count_slave	Integer	グローバル
Ndb_api_bytes_received_count	Integer	グローバル
Ndb_api_bytes_received_count_replica	Integer	グローバル
Ndb_api_bytes_received_count_session	Integer	セッション
Ndb_api_bytes_received_count_slave	Integer	グローバル
Ndb_api_bytes_sent_count	Integer	グローバル
Ndb_api_bytes_sent_count_replica	Integer	グローバル
Ndb_api_bytes_sent_count_session	Integer	セッション
Ndb_api_bytes_sent_count_slave	Integer	グローバル
Ndb_api_event_bytes_count	Integer	グローバル
Ndb_api_event_bytes_count_injector	Integer	グローバル
Ndb_api_event_data_count	Integer	グローバル
Ndb_api_event_data_count_injector	Integer	グローバル
Ndb_api_event_nodata_count	Integer	グローバル
Ndb_api_event_nodata_count_injector	Integer	グローバル
Ndb_api_pk_op_count	Integer	グローバル
Ndb_api_pk_op_count_replica	Integer	グローバル
Ndb_api_pk_op_count_session	Integer	セッション
Ndb_api_pk_op_count_slave	Integer	グローバル
Ndb_api_pruned_scan_count	Integer	グローバル
Ndb_api_pruned_scan_count_replica	Integer	グローバル
Ndb_api_pruned_scan_count_session	Integer	セッション
Ndb_api_pruned_scan_count_slave	Integer	グローバル
Ndb_api_range_scan_count	Integer	グローバル
Ndb_api_range_scan_count_replica	Integer	グローバル
Ndb_api_range_scan_count_session	Integer	セッション
Ndb_api_range_scan_count_slave	Integer	グローバル
Ndb_api_read_row_count	Integer	グローバル
Ndb_api_read_row_count_replica	Integer	グローバル
Ndb_api_read_row_count_session	Integer	セッション
Ndb_api_read_row_count_slave	Integer	グローバル

変数名	変数型	変数スコープ
Ndb_api_scan_batch_count	Integer	グローバル
Ndb_api_scan_batch_count_replica	Integer	グローバル
Ndb_api_scan_batch_count_session	Integer	セッション
Ndb_api_scan_batch_count_slave	Integer	グローバル
Ndb_api_table_scan_count	Integer	グローバル
Ndb_api_table_scan_count_replica	Integer	グローバル
Ndb_api_table_scan_count_session	Integer	セッション
Ndb_api_table_scan_count_slave	Integer	グローバル
Ndb_api_trans_abort_count	Integer	グローバル
Ndb_api_trans_abort_count_replica	Integer	グローバル
Ndb_api_trans_abort_count_session	Integer	セッション
Ndb_api_trans_abort_count_slave	Integer	グローバル
Ndb_api_trans_close_count	Integer	グローバル
Ndb_api_trans_close_count_replica	Integer	グローバル
Ndb_api_trans_close_count_session	Integer	セッション
Ndb_api_trans_close_count_slave	Integer	グローバル
Ndb_api_trans_commit_count	Integer	グローバル
Ndb_api_trans_commit_count_replica	Integer	グローバル
Ndb_api_trans_commit_count_session	Integer	セッション
Ndb_api_trans_commit_count_slave	Integer	グローバル
Ndb_api_trans_local_read_row_count	Integer	グローバル
Ndb_api_trans_local_read_row_count_replica	Integer	グローバル
Ndb_api_trans_local_read_row_count_session	Integer	セッション
Ndb_api_trans_local_read_row_count_slave	Integer	グローバル
Ndb_api_trans_start_count	Integer	グローバル
Ndb_api_trans_start_count_replica	Integer	グローバル
Ndb_api_trans_start_count_session	Integer	セッション
Ndb_api_trans_start_count_slave	Integer	グローバル
Ndb_api_uk_op_count	Integer	グローバル
Ndb_api_uk_op_count_replica	Integer	グローバル
Ndb_api_uk_op_count_session	Integer	セッション
Ndb_api_uk_op_count_slave	Integer	グローバル
Ndb_api_wait_exec_complete_count	Integer	グローバル
Ndb_api_wait_exec_complete_count_replica	Integer	グローバル
Ndb_api_wait_exec_complete_count_session	Integer	セッション
Ndb_api_wait_exec_complete_count_slave	Integer	グローバル
Ndb_api_wait_meta_request_count	Integer	グローバル
Ndb_api_wait_meta_request_count_replica	Integer	グローバル
Ndb_api_wait_meta_request_count_session	Integer	セッション
Ndb_api_wait_meta_request_count_slave	Integer	グローバル
Ndb_api_wait_nanos_count	Integer	グローバル

変数名	変数型	変数スコープ
Ndb_api_wait_nanos_count_replica	Integer	グローバル
Ndb_api_wait_nanos_count_session	Integer	セッション
Ndb_api_wait_nanos_count_slave	Integer	グローバル
Ndb_api_wait_scan_result_count	Integer	グローバル
Ndb_api_wait_scan_result_count_replica	Integer	グローバル
Ndb_api_wait_scan_result_count_session	Integer	セッション
Ndb_api_wait_scan_result_count_slave	Integer	グローバル
Ndb_cluster_node_id	Integer	グローバル
Ndb_config_from_host	Integer	両方
Ndb_config_from_port	Integer	両方
Ndb_config_generation	Integer	グローバル
Ndb_conflict_fn_epoch	Integer	グローバル
Ndb_conflict_fn_epoch_trans	Integer	グローバル
Ndb_conflict_fn_epoch2	Integer	グローバル
Ndb_conflict_fn_epoch2_trans	Integer	グローバル
Ndb_conflict_fn_max	Integer	グローバル
Ndb_conflict_fn_old	Integer	グローバル
Ndb_conflict_last_stable_epoch	Integer	グローバル
Ndb_conflict_reflected_op_discard_count	Integer	グローバル
Ndb_conflict_reflected_op_prepare_count	Integer	グローバル
Ndb_conflict_refresh_op_count	Integer	グローバル
Ndb_conflict_trans_conflict_commit_count	Integer	グローバル
Ndb_conflict_trans_detect_iter_count	Integer	グローバル
Ndb_conflict_trans_reject_count	Integer	グローバル
Ndb_conflict_trans_row_conflict_count	Integer	グローバル
Ndb_conflict_trans_row_reject_count	Integer	グローバル
Ndb_epoch_delete_delete_count	Integer	グローバル
Ndb_execute_count	Integer	グローバル
Ndb_last_commit_epoch_server	Integer	グローバル
Ndb_last_commit_epoch_session	Integer	セッション
Ndb_metadata_blacklist_size	Integer	グローバル
Ndb_metadata_detected_count	Integer	グローバル
Ndb_metadata_excluded_count	Integer	グローバル
Ndb_metadata_synced_count	Integer	グローバル
Ndb_cluster_node_id	Integer	グローバル
Ndb_number_of_data_nodes	Integer	グローバル
Ndb_pruned_scan_count	Integer	グローバル
Ndb_pushed_queries_defined	Integer	グローバル
Ndb_pushed_queries_dropped	Integer	グローバル
Ndb_pushed_queries_executed	Integer	グローバル
Ndb_pushed_reads	Integer	グローバル

変数名	変数型	変数スコープ
Ndb_scan_count	Integer	グローバル
Ndb_trans_hint_count_session	Integer	両方
Not_flushed_delayed_rows	Integer	グローバル
Ongoing_anonymous_gtid_violating_transactions_count	Integer	グローバル
Ongoing_anonymous_transaction_count	Integer	グローバル
Ongoing_automatic_gtid_violating_transactions_count	Integer	グローバル
Open_files	Integer	グローバル
Open_streams	Integer	グローバル
Open_table_definitions	Integer	グローバル
Open_tables	Integer	両方
Opened_files	Integer	グローバル
Opened_table_definitions	Integer	両方
Opened_tables	Integer	両方
Performance_schema_accounts_lost	Integer	グローバル
Performance_schema_cond_classes_lost	Integer	グローバル
Performance_schema_cond_instances_lost	Integer	グローバル
Performance_schema_digest_lost	Integer	グローバル
Performance_schema_file_classes_lost	Integer	グローバル
Performance_schema_file_handles_lost	Integer	グローバル
Performance_schema_file_instances_lost	Integer	グローバル
Performance_schema_hosts_lost	Integer	グローバル
Performance_schema_index_stat_lost	Integer	グローバル
Performance_schema_locker_lost	Integer	グローバル
Performance_schema_memory_classes_lost	Integer	グローバル
Performance_schema_metadata_lock_lost	Integer	グローバル
Performance_schema_mutex_classes_lost	Integer	グローバル
Performance_schema_mutex_instances_lost	Integer	グローバル
Performance_schema_nested_statement_instances_lost	Integer	グローバル
Performance_schema_prepared_statement_instances_lost	Integer	グローバル
Performance_schema_program_lost	Integer	グローバル
Performance_schema_rwlock_classes_lost	Integer	グローバル
Performance_schema_rwlock_instances_lost	Integer	グローバル
Performance_schema_session_connect_attrs_longest_seen	Integer	グローバル
Performance_schema_session_connect_attrs_lost	Integer	グローバル
Performance_schema_socket_classes_lost	Integer	グローバル
Performance_schema_socket_instances_lost	Integer	グローバル
Performance_schema_stage_classes_lost	Integer	グローバル
Performance_schema_statement_classes_lost	Integer	グローバル
Performance_schema_table_handles_lost	Integer	グローバル
Performance_schema_table_instances_lost	Integer	グローバル
Performance_schema_table_lock_stat_lost	Integer	グローバル

変数名	変数型	変数スコープ
Performance_schema_thread_classes	Integer	グローバル
Performance_schema_thread_instances	Integer	グローバル
Performance_schema_users_lost	Integer	グローバル
Prepared_stmt_count	Integer	グローバル
Queries	Integer	両方
Questions	Integer	両方
Rewriter_number_loaded_rules	Integer	グローバル
Rewriter_number_reloads	Integer	グローバル
Rewriter_number_rewritten_queries	Integer	グローバル
Rewriter_reload_error	Boolean	グローバル
Rpl_semi_sync_master_clients	Integer	グローバル
Rpl_semi_sync_master_net_avg_wait_time	Integer	グローバル
Rpl_semi_sync_master_net_wait_time	Integer	グローバル
Rpl_semi_sync_master_net_waits	Integer	グローバル
Rpl_semi_sync_master_no_times	Integer	グローバル
Rpl_semi_sync_master_no_tx	Integer	グローバル
Rpl_semi_sync_master_status	Boolean	グローバル
Rpl_semi_sync_master_timefunc_failures	Integer	グローバル
Rpl_semi_sync_master_tx_avg_wait_time	Integer	グローバル
Rpl_semi_sync_master_tx_wait_time	Integer	グローバル
Rpl_semi_sync_master_tx_waits	Integer	グローバル
Rpl_semi_sync_master_wait_pos_backlog	Integer	グローバル
Rpl_semi_sync_master_wait_sessions	Integer	グローバル
Rpl_semi_sync_master_yes_tx	Integer	グローバル
Rpl_semi_sync_slave_status	Boolean	グローバル
Rsa_public_key	文字列	グローバル
Secondary_engine_execution_count	Integer	両方
Select_full_join	Integer	両方
Select_full_range_join	Integer	両方
Select_range	Integer	両方
Select_range_check	Integer	両方
Select_scan	Integer	両方
Slave_open_temp_tables	Integer	グローバル
Slave_rows_last_search_algorithm_used	文字列	グローバル
Slow_launch_threads	Integer	両方
Slow_queries	Integer	両方
Sort_merge_passes	Integer	両方
Sort_range	Integer	両方
Sort_rows	Integer	両方
Sort_scan	Integer	両方
Ssl_accept_renegotiates	Integer	グローバル

変数名	変数型	変数スコープ
Ssl_accepts	Integer	グローバル
Ssl_callback_cache_hits	Integer	グローバル
Ssl_cipher	文字列	両方
Ssl_cipher_list	文字列	両方
Ssl_client_connects	Integer	グローバル
Ssl_connect_renegotiates	Integer	グローバル
Ssl_ctx_verify_depth	Integer	グローバル
Ssl_ctx_verify_mode	Integer	グローバル
Ssl_default_timeout	Integer	両方
Ssl_finished_accepts	Integer	グローバル
Ssl_finished_connects	Integer	グローバル
Ssl_server_not_after	Integer	両方
Ssl_server_not_before	Integer	両方
Ssl_session_cache_hits	Integer	グローバル
Ssl_session_cache_misses	Integer	グローバル
Ssl_session_cache_mode	文字列	グローバル
Ssl_session_cache_overflows	Integer	グローバル
Ssl_session_cache_size	Integer	グローバル
Ssl_session_cache_timeouts	Integer	グローバル
Ssl_sessions_reused	Integer	両方
Ssl_used_session_cache_entries	Integer	グローバル
Ssl_verify_depth	Integer	両方
Ssl_verify_mode	Integer	両方
Ssl_version	文字列	両方
Table_locks_immediate	Integer	グローバル
Table_locks_waited	Integer	グローバル
Table_open_cache_hits	Integer	両方
Table_open_cache_misses	Integer	両方
Table_open_cache_overflows	Integer	両方
Tc_log_max_pages_used	Integer	グローバル
Tc_log_page_size	Integer	グローバル
Tc_log_page_waits	Integer	グローバル
Threads_cached	Integer	グローバル
Threads_connected	Integer	グローバル
Threads_created	Integer	グローバル
Threads_running	Integer	グローバル
Uptime	Integer	グローバル
Uptime_since_flush_status	Integer	グローバル
validate_password_dictionary_file_last_parsed	日付時間	グローバル
validate_password_dictionary_file_words_checked	Integer	グローバル
validate_password_dictionary_file_last_parsed	日付時間	グローバル

変数名	変数型	変数スコープ
<code>validate_password.dictionary_file_words</code>	<code>integer</code>	グローバル

5.1.7 サーバーコマンドオプション

`mysqld` サーバーを起動するときに、[セクション4.2.2「プログラムオプションの指定」](#)に記載されているいずれかの方法で、プログラムオプションを指定できます。もっとも一般的な方法は、オプションファイルまたはコマンド行でオプションを提供するやり方です。ただし、ほとんどの場合では、サーバーが毎回実行するときサーバーが必ず同じオプションを使用します。これを確実に行う最適な方法は、オプションファイルにオプションを一覧表示することです。[セクション4.2.2.2「オプションファイルの使用」](#)を参照してください。このセクションではオプションファイルの形式および構文についても説明します。

`mysqld` は `[mysqld]` および `[server]` グループからオプションを読み取ります。`mysqld_safe` は `[mysqld]`、`[server]`、`[mysqld_safe]`、および `[safe_mysqld]` グループからオプションを読み取ります。`mysql.server` は `[mysqld]` および `[mysql.server]` グループからオプションを読み取ります。

`mysqld` には多くのコマンドオプションがあります。簡単なサマリーを表示するには、次のコマンドを実行します：

```
mysqld --help
```

完全なリストを表示するには、次のコマンドを使用します：

```
mysqld --verbose --help
```

リスト内の一部の項目は、実際にはサーバーの起動時に設定できるシステム変数です。これらは、[SHOW VARIABLES](#) ステートメントを使用して実行時に表示できます。前述の `mysqld` コマンドで表示される一部の項目は、[SHOW VARIABLES](#) 出力には表示されません。これは、これらがシステム変数ではなくオプションであるためです。

次に、もっとも一般的なサーバーオプションの一部を示します。その他のオプションは、ほかのセクションに記載されています。

- セキュリティーに影響するオプション。[セクション6.1.4「セキュリティ関連の mysqld オプションおよび変数」](#)を参照してください。
- SSL 関連オプション。[暗号化接続のコマンドオプション](#)を参照してください。
- バイナリログ制御オプション。[セクション5.4.4「バイナリログ」](#)を参照してください。
- レプリケーション関連オプション。[セクション17.1.6「レプリケーションおよびバイナリロギングのオプションと変数」](#)を参照してください。
- プラガブルストレージエンジンなどのプラグインをロードするためのオプション。[セクション5.6.1「プラグインのインストールおよびアンインストール」](#)を参照してください。
- 特定のストレージエンジンに固有のオプション。[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)および[セクション16.2.1「MyISAM 起動オプション」](#)を参照してください。

一部のオプションはバッファまたはキャッシュのサイズを制御します。所定のバッファについて、サーバーは内部データ構造を割り当てる必要がある場合もあります。これらの構造は、バッファに割り当てられた合計メモリーから割り当てられ、必要なスペースの量はプラットフォームに依存することがあります。つまり、バッファサイズを制御するオプションに値を割り当てたとき、実際に使用可能なスペースの量が、割り当てられた値と異なる場合もあることを意味します。一部の 경우에는、この量は割り当てられた値より少ないこともあります。サーバーが値を上方に調整することもできます。たとえば、最小値が 1024 のオプションに値 0 を割り当てると、サーバーは値を 1024 に設定します。

バッファサイズ、長さ、およびスタックサイズの値は、別途指定しないかぎりバイト単位で指定されます。

一部のオプションはファイル名の値を取ります。別途指定しないかぎり、値が相対パス名であれば、デフォルトのファイルの場所はデータディレクトリです。場所を明示的に指定するには、絶対パス名を使用します。たとえばデータディレクトリが `/var/mysql/data` だとします。相対パス名としてファイル値オプションを指定すると、`/var/mysql/data` の下に配置されます。値が絶対パス名である場合、その場所はパス名によって指定されます。

オプションとして変数名を使用して、サーバーの起動時にサーバーシステム変数の値を設定することもできます。サーバーシステム変数に値を割り当てるには、`--var_name=value` という形式のオプションを使用します。たとえば、`--sort_buffer_size=384M` は `sort_buffer_size` 変数を 384MB の値に設定します。

変数に値を割り当てると、特定の範囲内にとどまるように MySQL によって値が自動的に修正されたり、特定の値のみが許可されている場合は最も近い許容値に値が調整されたりすることがあります。

SET ステートメントを使用してシステム変数を実行時に設定できる最大値を制限するには、サーバー起動時に `--maximum-var_name=value` 形式のオプションを使用して、この最大値を指定します。

ほとんどのシステム変数の値は、SET ステートメントを使用して実行時に変更できます。セクション13.7.6.1「変数代入の SET 構文」を参照してください。

セクション5.1.8「サーバーシステム変数」では、すべての変数についての詳細な説明と、サーバーの起動時および実行時にそれらを設定するための追加情報を提供します。システム変数の変更の詳細は、セクション5.1.1「サーバーの構成」を参照してください。

- `--help, -?`

コマンド行形式	<code>--help</code>
---------	---------------------

短いヘルプメッセージを表示して終了します。詳細メッセージを表示するには、`--verbose` および `--help` の両方のオプションを使用します。

- `--admin-ssl, --skip-admin-ssl`

コマンド行形式	<code>--admin-ssl[={OFF ON}]</code>
導入	8.0.21
型	Boolean
デフォルト値	ON

`--admin-ssl` オプションは `--ssl` オプションと似ていますが、メイン接続インタフェースではなく管理接続インタフェースに適用される点が異なります。これらのインタフェースの詳細は、セクション5.1.12.1「接続インタフェース」を参照してください。

`--admin-ssl` オプションは、サーバーが管理インタフェースで暗号化された接続を許可するが不要としないことを指定します。このオプションはデフォルトで有効となっています。

`--admin-ssl` は、否定形式で `--skip-admin-ssl` またはシノニム (`--admin-ssl=OFF`、`--disable-admin-ssl`) として指定できます。この場合、このオプションは、`admin_ssl_XXX` および `admin_ssl_XXX` システム変数の設定に関係なく、サーバーが暗号化された接続を許可しないことを指定します。

`--admin-ssl` オプションは、管理インタフェースが暗号化された接続をサポートしているかどうかにかかわらずサーバーの起動時にのみ有効です。これは無視され、実行時の `ALTER INSTANCE RELOAD TLS` の操作には影響しません。たとえば、`--admin-ssl=OFF` を使用して、暗号化された接続を無効にして管理インタフェースを起動し、TLS を再構成して `ALTER INSTANCE RELOAD TLS FOR CHANNEL mysql_admin` を実行し、実行時に暗号化された接続を有効にできます。

connection-encryption サポートの構成に関する一般情報は、セクション6.3.1「暗号化接続を使用するための MySQL の構成」を参照してください。この説明はメイン接続インタフェース用に記述されていますが、パラメータ名は管理接続インタフェース用に似ています。サーバー側で `admin_ssl_cert` および `admin_ssl_key` システム変数を設定し、クライアント側で `--ssl-ca` (または `--ssl-capath`) オプションを設定することを検討してください。管理インタフェースの詳細は、暗号化された接続に対する管理インタフェースのサポートを参照してください。

- `--allow-suspicious-udfs`

コマンド行形式	<code>--allow-suspicious-udfs[={OFF ON}]</code>
型	Boolean
デフォルト値	OFF

このオプションは、メイン関数に `xxx` 記号のみを持つユーザー定義関数をロードできるかどうかを制御します。デフォルトでは、このオプションはオフで、少なくとも 1 つの補助記号を持つ UDF のみをロードできます。これにより、正当な UDF を含むもの以外の共有オブジェクトファイルから関数をロードしないようにします。 [Loadable Function Security Precautions](#) を参照してください。

- `--ansi`

コマンド行形式	<code>--ansi</code>
---------	---------------------

MySQL 構文の代わりに標準 (ANSI) SQL 構文を使用します。サーバー SQL モードをさらに正確に制御するには、代わりに `--sql-mode` オプションを使用します。 [セクション 1.7 「MySQL の標準への準拠」](#) および [セクション 5.1.11 「サーバー SQL モード」](#) を参照してください。

- `--basedir=dir_name, -b dir_name`

コマンド行形式	<code>--basedir=dir_name</code>
システム変数	<code>basedir</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ディレクトリ名
デフォルト値	parent of mysqld installation directory

MySQL インストールディレクトリへのパス。このオプションは、`basedir` システム変数を設定します。

サーバー実行可能ファイルは、起動時に独自のフルパス名を決定し、それが配置されているディレクトリの親をデフォルトの `basedir` 値として使用します。これにより、サーバーは、エラーメッセージを含む `share` ディレクトリなどのサーバー関連情報を検索するときに、その `basedir` を使用できます。

- `--character-set-client-handshake`

コマンド行形式	<code>--character-set-client-handshake[={OFF ON}]</code>
型	Boolean
デフォルト値	ON

クライアントによって送信された文字セット情報を無視しません。クライアント情報を無視して、サーバーのデフォルトの文字セットを使用するには、`--skip-character-set-client-handshake` を使用します。これにより、MySQL は MySQL 4.0. のように動作します。

- `--chroot=dir_name, -r dir_name`

コマンド行形式	<code>--chroot=dir_name</code>
型	ディレクトリ名

`chroot()` のシステムコールを使用して、`mysqld` サーバーを起動中にクローズ環境にします。これは推奨されるセキュリティ対策です。このオプションを使用すると、`LOAD DATA` および `SELECT ... INTO OUTFILE` が多少制限されます。

- `--console`

コマンド行形式	<code>--console</code>
プラットフォーム固有	Windows

(Windows のみ。) デフォルトのエラーログの出力先をコンソールにします。これは、デフォルトの宛先に基づく独自の出力先のログシンクに影響します。 [セクション 5.4.2 「エラーログ」](#) を参照してください。このオプションを使用した場合、`mysqld` はコンソールウィンドウを閉じません。

両方が指定されている場合、`--console` は `--log-error` よりも優先されます。

- `--core-file`

コマンド行形式	<code>--core-file[={OFF ON}]</code>
型	Boolean
デフォルト値	OFF

`mysqld` が異常終了した場合にコアファイルを作成します。コアファイルの名前および場所はシステムに依存します。Linux の場合、`core.pid` という名前のコアファイルがプロセスの現在の作業ディレクトリに書き込まれ、これは `mysqld` のデータディレクトリです。`pid` はサーバープロセスのプロセス ID を表します。macOS では、`core.pid` というコアファイルが `/cores` ディレクトリに書き込まれます。Solaris の場合、`coreadm` コマンドを使用して、コアファイルの書き込み先と名前を指定する方法を指定します。

一部のシステムでコアファイルを取得するには、`mysqld_safe` に `--core-file-size` オプションを指定する必要もあります。[セクション4.3.2「mysqld_safe — MySQL サーバー起動スクリプト」](#) を参照してください。Solaris などの一部のシステムでは、`--user` オプションも使用しないとコアファイルを取得できません。追加の制限または制約がある場合もあります。たとえば、サーバーを起動する前に `ulimit -c unlimited` を実行することが必要な場合もあります。システムのドキュメントを参照してください。

`innodb_buffer_pool_in_core_file` 変数を使用すると、それをサポートするオペレーティングシステム上のコアファイルのサイズを縮小できます。詳細は、[セクション15.8.3.7「コアファイルからのバッファープールページの除外」](#) を参照してください。

- `--daemonize, -D`

コマンド行形式	<code>--daemonize[={OFF ON}]</code>
型	Boolean
デフォルト値	OFF

このオプションを使用すると、サーバーは従来のフォーキングデーモンとして実行され、`systemd` を使用してプロセス制御を行うオペレーティングシステムと連携できます。詳細は、[セクション2.5.9「systemd を使用した MySQL Server の管理」](#) を参照してください。

`--daemonize` は、`--initialize` および `--initialize-insecure` と相互に排他的です。

サーバーが `--daemonize` オプションを使用して起動され、tty デバイスに接続されていない場合は、エラー出力をデフォルトのログファイルに送るために、明示的なロギングオプションがないときに `--log-error=""` のデフォルトのエラーロギングオプションが使用されます。

`-D` は、`--daemonize` のシノニムです。

- `--datadir=dir_name, -h dir_name`

コマンド行形式	<code>--datadir=dir_name</code>
システム変数	<code>datadir</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ディレクトリ名

MySQL サーバーデータディレクトリへのパス。このオプションは、`datadir` システム変数を設定します。その変数の説明を参照してください。

- `--debug[=debug_options], -# [debug_options]`

コマンド行形式	<code>--debug[=debug_options]</code>
---------	--------------------------------------

システム変数	<code>debug</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値 (Unix)	<code>d:t:i:o,/tmp/mysqlq.trace</code>
デフォルト値 (Windows)	<code>d:t:i:O,\mysqlq.trace</code>

MySQL が `-DWITH_DEBUG=1 CMake` オプションを使用して構成されている場合、このオプションを使用して `mysqld` が実行しているトレースファイルを取得できます。一般的な `debug_options` 文字列は `d:t:o,file_name` です。デフォルトは、UNIX の場合は `d:t:i:o,/tmp/mysqlq.trace`、Windows の場合は `d:t:i:O,\mysqlq.trace` です。

`-DWITH_DEBUG=1` を使用して MySQL にデバッグサポートを構成することにより、サーバーを起動するときに `--debug="d,parser_debug"` オプションを使用できるようになります。これにより、SQL ステートメントの処理に使用される Bison パーサーが、パーサートレースをサーバーの標準エラー出力にダンプします。一般的に、この出力はエラーログに書き込まれます。

このオプションは複数回指定されることがあります。+ または - で開始される値が以前の値に加算または減算されます。たとえば、`--debug=T --debug=+P` と指定すると、値は `P:T` に設定されます。

詳細については、[セクション5.9.4「DEBUG パッケージ」](#)を参照してください。

- `--debug-sync-timeout[=N]`

コマンド行形式	<code>--debug-sync-timeout[=#]</code>
型	Integer

テストおよびデバッグのための Debug Sync 機能が有効かどうかを制御します。デバッグ同期を使用するには、MySQL が `-DENABLE_DEBUG_SYNC=1 CMake` オプションで構成されている必要があります ([セクション2.9.7「MySQL ソース構成オプション」](#)を参照)。Debug Sync がコンパイルされていない場合、このオプションは使用できません。オプション値は秒単位のタイムアウトです。デフォルト値は 0 で、Debug Sync を無効にします。これを有効にするには、0 より大きい値を指定してください。この値は、個々の同期点についてのデフォルトのタイムアウトになります。オプションが値なしで指定された場合、タイムアウトは 300 秒に設定されます。

Debug Sync 機能および同期点の使用方法についての説明は、「[MySQL Internals: Test Synchronization](#)」を参照してください。

- `--default-time-zone=timezone`

コマンド行形式	<code>--default-time-zone=name</code>
型	文字列

デフォルトのサーバータイムゾーンを設定します。このオプションは、グローバルな `time_zone` システム変数を設定します。このオプションを指定しない場合、デフォルトのタイムゾーンは、(`system_time_zone` システム変数の値によって指定される) システムのタイムゾーンと同一になります。

- `--defaults-extra-file=file_name`

このオプションファイルは、グローバルオプションファイルのあとに読み取りますが、(UNIX では) ユーザーオプションファイルの前に読み取るようにしてください。ファイルが存在しないかアクセスできない場合、エラーが発生します。`file_name` は、フルパス名でなく相対パス名として指定された場合、現行ディレクトリを基準にして解釈されます。これを使用する場合は、コマンド行の最初のオプションでなければなりません。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--defaults-file=file_name`

指定されたオプションファイルのみを読み取ります。ファイルが存在しないかアクセスできない場合、エラーが発生します。`file_name` は、フルパス名でなく相対パス名として指定された場合、現行ディレクトリを基準にして解釈されます。

例外: `--defaults-file` でも、`mysqld` は `mysqld-auto.cnf` を読み取ります。

注記

`--defaults-file` および `--install` (または `--install-manual`) オプションを使用してサーバーを起動する場合を除き、これはコマンド行の最初のオプションである必要があります。`--install` (または `--install-manual`) が最初に必要です。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--defaults-group-suffix=str`

通常のオプショングループだけでなく、通常の名前に `str` のサフィクスが付いたグループも読み取ります。たとえば、`mysqld` は通常 `[mysqld]` グループを読み取ります。`--defaults-group-suffix=_other` オプションを指定した場合、`mysqld` は `[mysqld_other]` グループも読み取ります。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--early-plugin-load=plugin_list`

コマンド行形式	<code>--early-plugin-load=plugin_list</code>
型	文字列
デフォルト値	empty string

このオプションは、必須の組み込みプラグインをロードする前、およびストレージエンジンを初期化する前に、どのプラグインをロードするかをサーバーに指示します。複数の `--early-plugin-load` オプションを指定した場合は、最後のオプションのみが使用されます。

オプション値は、`name = plugin_library` および `plugin_library` の値をセミコロンで区切ったリストです。各 `name` はロードするプラグインの名前で、`plugin_library` はプラグインコードを含むライブラリファイルの名前です。プラグイン名を前に付けずにプラグインライブラリを指定した場合、サーバーはライブラリ内のすべてのプラグインを

ロードします。サーバーは、`plugin_dir` システム変数で指定されたディレクトリ内でプラグインライブラリファイルを検索します。

たとえば、`myplug1` および `myplug2` という名前のプラグインに `myplug1.so` および `myplug2.so` というライブラリファイルがある場合は、このオプションを使用して初期プラグインロードを実行します:

```
shell> mysqld --early-plugin-load="myplug1=myplug1.so;myplug2=myplug2.so"
```

一部のコマンドインタプリタではセミコロン (;) が特殊文字として解釈されるため、引数値の前後に引用符が使用されます。(たとえば UNIX シェルでは、これはコマンド終端記号として扱われます。)

指定された各プラグインは、`mysqld` の単一の呼出しに対してのみ早期にロードされます。再起動後、`--early-plugin-load` を再度使用しないかぎり、プラグインは早期にロードされません。

サーバーが `--initialize` または `--initialize-insecure` を使用して起動された場合、`--early-plugin-load` で指定されたプラグインはロードされません。

サーバーが `--help` で実行されている場合、`--early-plugin-load` で指定されたプラグインがロードされますが、初期化されません。この動作により、プラグインオプションがヘルプメッセージに確実に表示されます。

デフォルトの `--early-plugin-load` 値は空です。`keyring_file` プラグインをロードするには、空でない値で明示的な `--early-plugin-load` オプションを使用する必要があります。

InnoDB のテーブルスペース暗号化機能は、暗号化キー管理のために `keyring_file` プラグインに依存しており、暗号化されたテーブルの InnoDB リカバリを容易にするために、ストレージエンジンを初期化する前に `keyring_file` プラグインをロードする必要があります。起動時に `keyring_file` プラグインをロードする管理者は、適切な空でないオプション値 (Unix や Unix のようなシステムでは `keyring_file.so`、Windows では `keyring_file.dll`) を使用する必要があります。

InnoDB テーブルスペースの暗号化の詳細は、[セクション15.13「InnoDB 保存データ暗号化」](#) を参照してください。プラグインのロードに関する一般情報については、[セクション5.6.1「プラグインのインストールおよびアンインストール」](#) を参照してください。

- `--exit-info[=flags], -T [flags]`

コマンド行形式	<code>--exit-info[=flags]</code>
型	Integer

これは、`mysqld` サーバーのデバッグに使用できる様々なフラグのビットマスクです。このオプションを使用するには、完全にこのオプションを理解していることが必要です。

- `--external-locking`

コマンド行形式	<code>--external-locking[={OFF ON}]</code>
型	Boolean
デフォルト値	OFF

デフォルトで無効になっている外部ロック (システムロック) を有効にします。`lockd` が完全には機能しないシステム (Linux など) でこのオプションを使用すると、`mysqld` でデッドロックが発生しやすくなります。

外部ロックを明示的に無効にするには、`--skip-external-locking` を使用します。

外部ロックは MyISAM テーブルアクセスにのみ影響します。使用できるまたはできない状況も含めた詳細情報については、[セクション8.11.5「外部ロック」](#) を参照してください。

- `--flush`

コマンド行形式	<code>--flush[={OFF ON}]</code>
システム変数	<code>flush</code>
スコープ	グローバル

動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

各 SQL ステートメント後にすべての変更内容をディスクにフラッシュ (同期) します。通常、MySQL では各 SQL ステートメントの終了後にのみすべての変更内容をディスクに書き込み、ディスクへの同期はオペレーティングシステムが処理します。 [セクションB.3.3.3「MySQL が繰り返しクラッシュする場合の対処方法」](#) を参照してください。

注記

--flush が指定されている場合、flush_time の値は関係なく、flush_time を変更してもフラッシュ動作には影響しません。

- --gdb

コマンド行形式	--gdb[={OFF ON}]
型	Boolean
デフォルト値	OFF

SIGINT 用の割り込みハンドラ (ブレークポイントを設定するための ^C を使用して mysqld を停止するために必要) をインストールし、スタックトレースおよびコアファイルの処理を無効にします。 [セクション5.9.1.4「gdb での mysqld のデバッグ」](#) を参照してください。

Windows では、このオプションにより、RESTART ステートメントの実装に使用される分岐も抑制されます: フォーキングを使用すると、一方のプロセスを他方のプロセスのモニターとして動作させることができます。このモニターはサーバーとして機能します。ただし、フォークすると、デバッグのためにアタッチするサーバープロセスの決定が困難になるため、--gdb でサーバーを起動するとフォーキングが抑制されます。このオプションで起動されたサーバーの場合、RESTART は単に終了し、再起動しません。

デバッグ以外の設定では、--no-monitor を使用してモニタープロセスの分岐を抑制できます。

- --initialize, -l

コマンド行形式	--initialize[={OFF ON}]
型	Boolean
デフォルト値	OFF

このオプションは、データディレクトリを作成し、mysql システムスキーマにテーブルを移入することで、MySQL インストールを初期化するために使用します。詳細は、 [セクション2.10.1「データディレクトリの初期化」](#) を参照してください。

--initialize を使用してサーバーを起動すると、init_file システム変数で指定されたファイルで許可されるステートメントを制限する一部の機能が使用できなくなります。詳細は、その変数の説明を参照してください。また、disabled_storage_engines システム変数は効果がありません。

--initialize とともに使用する場合、--ndbcluster オプションは無視されます。

--initialize は、--daemonize と相互に排他的です。

-l は、--initialize のシノニムです。

- --initialize-insecure

コマンド行形式	--initialize-insecure[={OFF ON}]
型	Boolean

デフォルト値	OFF
--------	-----

このオプションは、データディレクトリを作成し、`mysql` システムスキーマにテーブルを移入することで、MySQL インストールを初期化するために使用します。このオプションは、`--initialize` を意味します。詳細は、そのオプションの説明および [セクション2.10.1「データディレクトリの初期化」](#) を参照してください。

`--initialize-insecure` は、`--daemonize` と相互に排他的です。

- `--innodb-xxx`

InnoDB ストレージエンジンのオプションを設定します。InnoDB オプションは、[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#) にリストされています。

- `--install [service_name]`

コマンド行形式	<code>--install [service_name]</code>
プラットフォーム固有	Windows

(Windows のみ) Windows の起動時に自動的に開始する Windows サービスとしてサーバーをインストールします。`service_name` の値が指定されない場合、デフォルトサービス名は `MySQL` です。詳細については、[セクション2.3.4.8「Windows のサービスとして MySQL を起動する」](#) を参照してください。

注記

`--defaults-file` オプションおよび `--install` オプションを使用してサーバーを起動する場合、`--install` を先にする必要があります。

- `--install-manual [service_name]`

コマンド行形式	<code>--install-manual [service_name]</code>
プラットフォーム固有	Windows

(Windows のみ) 手動で開始する必要がある Windows サービスとしてサーバーをインストールします。Windows の起動中に自動的に開始されません。`service_name` の値が指定されない場合、デフォルトサービス名は `MySQL` です。詳細については、[セクション2.3.4.8「Windows のサービスとして MySQL を起動する」](#) を参照してください。

注記

`--defaults-file` オプションおよび `--install-manual` オプションを使用してサーバーを起動する場合、`--install-manual` を先にする必要があります。

- `--language=lang_name, -L lang_name`

コマンド行形式	<code>--language=name</code>
非推奨	はい; use <code>lc-messages-dir</code> instead
システム変数	<code>language</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ディレクトリ名

デフォルト値	<code>/usr/local/mysql/share/mysql/english/</code>
--------	--

エラーメッセージに使用する言語。lang_name は、言語名として指定するか、言語ファイルがインストールされているディレクトリへのフルパス名として指定できます。 [セクション10.12「エラーメッセージ言語の設定」](#) を参照してください。

非推奨になった (`--language` ではなく、`--lc-messages-dir` および `--lc-messages` を使用する必要があります。また、`--lc-messages-dir` のシノニムとして処理されます)。 `--language` オプションは、将来の MySQL リリースで削除される予定です。

- `--large-pages`

コマンド行形式	<code>--large-pages[={OFF ON}]</code>
システム変数	<code>large_pages</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
プラットフォーム固有	Linux
型	Boolean
デフォルト値	OFF

ハードウェアまたはオペレーティングシステムのアーキテクチャーによっては、デフォルト (通常は 4K バイト) よりも大きいメモリーページをサポートしています。このサポートの実際の実装は、ベースとなるハードウェアやオペレーティングシステムに依存します。大量のメモリーアクセスがあるアプリケーションの場合、大きいページを使用して、トランスレーションルックアサイドバッファ (TLB; Translation Lookaside Buffer) のミスが減ることによってパフォーマンスが改善される可能性があります。

MySQL では、ラージページサポート (Linux では HugeTLB と呼ばれる) の Linux 実装がサポートされています。 [セクション8.12.3.2「ラージページのサポートの有効化」](#) を参照してください。大きいページの Solaris サポートについては、`--super-large-pages` オプションの説明を参照してください。

`--large-pages` はデフォルトで無効になっています。

- `--lc-messages=locale_name`

コマンド行形式	<code>--lc-messages=name</code>
システム変数	<code>lc_messages</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	<code>en_US</code>

エラーメッセージに使用するロケール。デフォルトは `en_US` です。サーバーは引数を言語名に変換し、これを `--lc-messages-dir` の値と組み合わせてエラーメッセージファイルの場所を生成します。 [セクション10.12「エラーメッセージ言語の設定」](#) を参照してください。

- `--lc-messages-dir=dir_name`

コマンド行形式	<code>--lc-messages-dir=dir_name</code>
システム変数	<code>lc_messages_dir</code>
スコープ	グローバル
動的	いいえ

SET_VAR ヒントの適用	いいえ
型	ディレクトリ名

エラーメッセージが配置されているディレクトリ。サーバーはこの値を `--lc-messages` の値と一緒に使用して、エラーメッセージファイルの場所を生成します。 [セクション10.12「エラーメッセージ言語の設定」](#) を参照してください。

- `--local-service`

コマンド行形式	<code>--local-service</code>
---------	------------------------------

(Windows のみ) サービス名のあとに `--local-service` オプションが指定されると、システム権限が制限された `LocalService` の Windows アカウントを使用してサーバーが実行されます。 `--defaults-file` および `--local-service` の両方がサービス名のあとに指定される場合、どのような順序でもかまいません。 [セクション2.3.4.8「Windows のサービスとして MySQL を起動する」](#) を参照してください。

- `--log-error[=file_name]`

コマンド行形式	<code>--log-error[=file_name]</code>
システム変数	<code>log_error</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ファイル名

デフォルトのエラーログの保存先を指定されたファイルに設定します。これは、デフォルトの宛先に基づく独自の出力先のログシンクに影響します。 [セクション5.4.2「エラーログ」](#) を参照してください。

オプションでファイル名が指定されていない場合、Unix および Unix に似たシステムのデフォルトのエラーログの保存先は、データディレクトリ内の `host_name.err` という名前のファイルです。 `--pid-file` オプションが指定されていないかぎり、Windows のデフォルトの宛先は同じです。その場合、ファイル名は PID ファイルベース名に接尾辞 `.err` を付けたものになります。

オプションでファイル名を指定する場合、デフォルトの宛先はそのファイル (名前に接尾辞がない場合は `.err` 接尾辞が追加されたもの) で、別の場所を指定する絶対パス名が指定されていないかぎり、データディレクトリの下にあります。

エラーログ出力をエラーログファイルにリダイレクトできない場合は、エラーが発生し、起動に失敗します。

Windows では、`--console` は `--log-error` よりも優先されます (両方が指定されている場合)。この場合、デフォルトのエラーログの保存先は、ファイルではなくコンソールです。

- `--log-isam[=file_name]`

コマンド行形式	<code>--log-isam[=file_name]</code>
型	ファイル名

MyISAM のすべての変更内容をこのファイルに記録します (**MyISAM** をデバッグするときだけ使用します)。

- `--log-raw`

コマンド行形式	<code>--log-raw[={OFF ON}]</code>
システム変数 (≥ 8.0.19)	<code>log_raw</code>
スコープ (≥ 8.0.19)	グローバル
動的 (≥ 8.0.19)	はい
SET_VAR ヒントの適用 (≥ 8.0.19)	いいえ

型	Boolean
デフォルト値	OFF

一般クエリーログ、スロークエリーログ、およびバイナリログに書き込まれた特定のステートメント内のパスワードは、文字どおりプレーンテキストで発生しないようにサーバーによって書き換えられます。一般クエリーログについてのパスワードの書き換えは、`--log-raw` オプションでサーバーを起動することによって抑制できます。このオプションは、サーバーによって受け取られるステートメントの正確なテキストを表示する際の診断目的で役立つ場合がありますが、セキュリティ上の理由で本番用途では推奨されません。

クエリーリライトプラグインがインストールされている場合、`--log-raw` オプションは次のようにステートメントのロギングに影響します:

- `--log-raw` が不在の場合、サーバーはクエリーリライトプラグインによって返されたステートメントをログに記録します。これは、受け取ったステートメントとは異なる場合があります。
- `--log-raw` では、サーバーは元のステートメントを受信したとおりにログに記録します。

詳細は、[セクション6.1.2.3「パスワードおよびロギング」](#)を参照してください。

- `--log-short-format`

コマンド行形式	<code>--log-short-format[={OFF ON}]</code>
型	Boolean
デフォルト値	OFF

スロークエリーログがアクティブ化されている場合は、ログに記録される情報が少なくなります。

- `--log-tc=file_name`

コマンド行形式	<code>--log-tc=file_name</code>
型	ファイル名
デフォルト値	<code>tc.log</code>

メモリーマップ済みのトランザクションコーディネータログファイルの名前 (バイナリログが無効のときに複数のストレージエンジンに影響するXAトランザクション用)。デフォルト名は `tc.log` です。フルパス名が指定されない場合、ファイルはデータディレクトリの下に作成されます。このオプションは使用されません。

- `--log-tc-size=size`

コマンド行形式	<code>--log-tc-size=#</code>
型	Integer
デフォルト値	<code>6 * page size</code>
最小値	<code>6 * page size</code>
最大値 (64 ビットプラットフォーム)	<code>18446744073709551615</code>
最大値 (32 ビットプラットフォーム)	<code>4294967295</code>

メモリーマップ済みのトランザクションコーディネータログのバイト単位のサイズ。デフォルト値と最小値はページサイズの6倍で、値はページサイズの倍数である必要があります。

- `--memlock`

コマンド行形式	<code>--memlock[={OFF ON}]</code>
型	Boolean

デフォルト値	OFF
--------	-----

メモリー内の `mysqld` プロセスをロックします。このオプションは、オペレーティングシステムによって `mysqld` がディスクへのスワップを実行するという問題がある場合に役立つことがあります。

`--memlock` は、`mlockall()` システムコールをサポートするシステムで動作します。これには、Solaris、2.4 以上のカーネルを使用するほとんどの Linux ディストリビューション、およびその他の Unix システムが含まれます。Linux システムの場合、`mlockall()` (およびこのオプション) が、システムの `mman.h` ファイルで定義されているかどうかを次のようにして確認することによって、これがサポートされているかどうかを識別できます。

```
shell> grep mlockall /usr/include/sys/mman.h
```

`mlockall()` がサポートされている場合、前のコマンドの出力に、次のように表示されます。

```
extern int mlockall (int __flags) __THROW;
```

重要

このオプションを使用する場合、サーバーを `root` として実行することが必要な場合もあり、これはセキュリティ上の理由から通常はよい考えではありません。 [セクション 6.1.5 「MySQL を通常ユーザーとして実行する方法」](#) を参照してください。

Linux およびおそらくその他のシステムでは、`limits.conf` ファイルを変更することによって、サーバーを `root` として実行しないで済みます。 [セクション 8.12.3.2 「ラージページのサポートの有効化」](#) の `memlock` 制限に関するメモを参照してください。

`mlockall()` システムコールをサポートしていないシステムでは、このオプションを使用しないでください。使用すると、`mysqld` を起動しようとするときにすぐに終了する可能性が高くなります。

- `--myisam-block-size=N`

コマンド行形式	<code>--myisam-block-size=#</code>
型	Integer
デフォルト値	1024
最小値	1024
最大値	16384

MyISAM インデックスページに使用するブロックサイズ。

- `--no-defaults`

オプションファイルを読み取りません。オプションファイルから不明のオプションを読み取ることが原因でプログラムの起動に失敗する場合、`--no-defaults` を使用して、オプションを読み取らないようにできます。これを使用する場合は、コマンド行の最初のオプションでなければなりません。

このオプションおよびその他のオプションファイルオプションの詳細は、 [セクション 4.2.2.3 「オプションファイルの処理に影響するコマンド行オプション」](#) を参照してください。

- `--no-dd-upgrade`

コマンド行形式	<code>--no-dd-upgrade[={OFF ON}]</code>
非推奨	8.0.16
型	Boolean

デフォルト値	OFF
--------	-----

注記

このオプションは、MySQL 8.0.16 では非推奨です。データディクショナリおよびサーバーのアップグレード動作をより細かく制御できる `--upgrade` オプションに置き換えられています。

MySQL サーバーの起動プロセス中にデータディクショナリテーブルが自動的にアップグレードされないようにします。このオプションは通常、既存のインストールを新しい MySQL バージョンにインプレースアップグレードした後に MySQL サーバーを起動するときに使用されます。これにはデータディクショナリテーブル定義の変更が含まれる場合があります。

`--no-dd-upgrade` が指定されていて、サーバーが予期されるバージョンのデータディクショナリがデータディクショナリ自体に格納されているバージョンと異なることを検出した場合、データディクショナリのアップグレードが禁止されていることを示すエラーで起動が失敗

```
[ERROR] [MY-011091] [Server] Data dictionary upgrade prohibited by the
command line option '--no_dd_upgrade'.
[ERROR] [MY-010020] [Server] Data Dictionary initialization failed.
```

通常の起動時に、データディクショナリテーブル定義をアップグレードする必要があるかどうかを判断するために、サーバーのデータディクショナリのバージョンがデータディクショナリに格納されているバージョンと比較されます。アップグレードが必要でサポートされている場合、サーバーは、更新された定義を含むデータディクショナリテーブルを作成し、永続化されたメタデータを新しいテーブルにコピーし、古いテーブルを新しいテーブルに原子的に置き換え、データディクショナリを再初期化します。アップグレードが不要な場合、データディクショナリテーブルを更新せずに起動が続行されます。

- `--no-monitor`

コマンド行形式	<code>--no-monitor[={OFF ON}]</code>
導入	8.0.12
プラットフォーム固有	Windows
型	Boolean
デフォルト値	OFF

(Windows のみ) このオプションは、`RESTART` ステートメントの実装に使用される分岐を抑制: フォーキングを使用すると、一方のプロセスを他方のプロセスのモニターとして動作させることができます。このモニターはサーバーとして機能します。このオプションで起動されたサーバーの場合、`RESTART` は単に終了し、再起動しません。

`--no-monitor` は、MySQL 8.0.12 より前は使用できません。回避策として `--gdb` オプションを使用できます。

- `--old-style-user-limits`

コマンド行形式	<code>--old-style-user-limits[={OFF ON}]</code>
型	Boolean
デフォルト値	OFF

古いスタイルのユーザー制限を有効にします。(MySQL 5.0.3 以前では、アカウントリソースは、`user` テーブルのアカウント行単位ではなく、ユーザーが接続したホストごとに別々にカウントされていました。) [セクション 6.2.20 「アカウントリソース制限の設定」](#) を参照してください。

- `--performance-schema-xxx`

パフォーマンススキーマオプションを構成します。詳細は、[セクション 27.14 「パフォーマンススキーマコマンドオプション」](#) を参照してください。

- `--plugin-load=plugin_list`

コマンド行形式	<code>--plugin-load=plugin_list</code>
システム変数	<code>plugin_load</code>
スコープ	グローバル
動的	いいえ
<code>SET_VAR</code> ヒントの適用	いいえ
型	文字列

このオプションは、指定されたプラグインを起動時にロードするようサーバーに指示します。複数の `--plugin-load` オプションが指定された場合、最後のオプションのみが使用されます。ロードする追加のプラグインは、`--plugin-load-add` オプションを使用して指定できます。

オプション値は、`name = plugin_library` および `plugin_library` の値をセミコロンで区切ったリストです。各 `name` はロードするプラグインの名前で、`plugin_library` はプラグインコードを含むライブラリファイルの名前です。プラグイン名を前に付けずにプラグインライブラリを指定した場合、サーバーはライブラリ内のすべてのプラグインをロードします。サーバーは、`plugin_dir` システム変数で指定されたディレクトリ内でプラグインライブラリファイルを検索します。

たとえば、`myplug1` および `myplug2` という名前のプラグインに `myplug1.so` および `myplug2.so` というライブラリファイルがある場合は、このオプションを使用して初期プラグインロードを実行します:

```
shell> mysqld --plugin-load="myplug1=myplug1.so;myplug2=myplug2.so"
```

セミコロン (;) は一部のコマンドインタプリタで特殊文字として解釈されるため、ここでは引数値の前後に引用符が使用されます。(たとえば UNIX シェルでは、これはコマンド終端記号として扱われます。)

各名前付きプラグインは、`mysqld` の単一の呼出しに対してのみロードされます。再起動後、`--plugin-load` をふたたび使用しないかぎり、プラグインはロードされません。これは `INSTALL PLUGIN` とは対照的で、こちらは `mysql.plugins` テーブルに項目を追加することで、サーバーが通常起動するたびにプラグインがロードされます。

通常の起動シーケンスでは、サーバーは `mysql.plugins` システムテーブルを読み取ることによって、ロードするプラグインを決定します。サーバーが `--skip-grant-tables` オプションで起動された場合、`mysql.plugins` テーブルに登録されているプラグインはロードされず、使用できません。`--plugin-load` を使用すると、`--skip-grant-tables` が指定されている場合でもプラグインをロードできます。`--plugin-load` を使用すると、実行時にロードできないプラグインを起動時にロードすることもできます。

プラグインのロードについての追加情報は、[セクション5.6.1「プラグインのインストールおよびアンインストール」](#)を参照してください。

- `--plugin-load-add=plugin_list`

コマンド行形式	<code>--plugin-load-add=plugin_list</code>
システム変数	<code>plugin_load_add</code>
スコープ	グローバル
動的	いいえ
<code>SET_VAR</code> ヒントの適用	いいえ

型	文字列
---	-----

このオプションは `--plugin-load` オプションを補完します。`--plugin-load-add` は、起動時にロードされるプラグインのセットに 1 つまたは複数のプラグインを追加します。引数の形式は `--plugin-load` と同じです。`--plugin-load-add` は、大量のプラグインのセットを、長くて扱いにくい単一の `--plugin-load` 引数として指定しないようにできます。

`--plugin-load-add` は `--plugin-load` がなくても指定できますが、`--plugin-load` はロードするプラグインのセットをリセットするため、`--plugin-load` の前に出現するすべての `--plugin-load-add` は効果がありません。つまり、次のオプションの場合、

```
--plugin-load=x --plugin-load-add=y
```

上記は次のオプションと同等です。

```
--plugin-load="x;y"
```

ただし、次のオプションの場合、

```
--plugin-load-add=y --plugin-load=x
```

上記は次のオプションと同等です。

```
--plugin-load=x
```

プラグインのロードについての追加情報は、[セクション 5.6.1「プラグインのインストールおよびアンインストール」](#) を参照してください。

- `--plugin-xxx`

サーバープラグインに関するオプションを指定します。たとえば、多くのストレージエンジンはプラグインとして構築でき、そのようなエンジンに対してそれらのオプションを `--plugin` プリフィクスで指定できます。したがって、InnoDB の `--innodb-file-per-table` オプションは `--plugin-innodb-file-per-table` として指定できます。

有効または無効にできるブールオプションの場合、`--skip` プリフィクスおよびその他の代替形式もサポートされます ([セクション 4.2.2.4「プログラムオプション修飾子」](#) を参照してください)。たとえば、`--skip-plugin-innodb-file-per-table` は `innodb-file-per-table` を無効にします。

`--plugin` プリフィクスの理由として、組み込みサーバーオプションとの名前競合がある場合に、あいまいさを排除してプラグインオプションを指定できるということがあります。たとえば、プラグイン「sql」に名前を指定し「mode」オプションを実装するプラグインライターは、オプション名が `--sql-mode` となることがあり、同じ名前の組み込みオプションと競合します。そのような場合、競合する名前への参照は、組み込みオプション側として解決されます。あいまいさを避けるために、ユーザーはプラグインオプションを `--plugin-sql-mode` として指定できません。あいまいさの問題を避けるために、プラグインオプションに `--plugin` プリフィクスを使用することを推奨します。

- `--port=port_num, -P port_num`

コマンド行形式	<code>--port=port_num</code>
システム変数	<code>port</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	3306
最小値	0

最大値	65535
-----	-------

TCP/IP 接続を listen するとき使用するポート番号。Unix および Unix に似たシステムでは、サーバーが root オペレーティングシステムユーザーによって起動されないかぎり、ポート番号は 1024 以上である必要があります。このオプションを 0 に設定すると、デフォルト値が使用されます。

- `--port-open-timeout=num`

コマンド行形式	<code>--port-open-timeout=#</code>
型	Integer
デフォルト値	0

一部のシステムでは、サーバーが停止すると、TCP/IP ポートがただちに利用できなくなることがあります。その後すぐにサーバーを再起動した場合、サーバーがポートをふたたびオープンしようとして失敗することがあります。このオプションは、TCP/IP ポートを開くことができない場合、TCP/IP ポートが開放されるまでサーバーが待機する秒数を指示します。デフォルトでは待機しません。

- `--print-defaults`

プログラム名と、オプションファイルから受け取るすべてのオプションを出力します。パスワード値はマスクされます。これは、`--defaults-file` または `--defaults-extra-file` の直後に使用できることを除き、コマンドラインで最初のオプションにする必要があります。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--remove [service_name]`

コマンド行形式	<code>--remove [service_name]</code>
プラットフォーム固有	Windows

(Windows のみ) MySQL Windows サービスを削除します。 `service_name` の値が指定されない場合、デフォルトサービス名は `MySQL` です。詳細については、[セクション2.3.4.8「Windows のサービスとして MySQL を起動する」](#)を参照してください。

- `--safe-user-create`

コマンド行形式	<code>--safe-user-create[={OFF ON}]</code>
型	Boolean
デフォルト値	OFF

このオプションが有効になっている場合、`mysql.user` システムテーブルまたはテーブル内のカラムに対する `INSERT` 権限がないかぎり、ユーザーは `GRANT` ステートメントを使用して新しい MySQL ユーザーを作成できません。あるユーザーが新規ユーザーを作成する能力を持ち、そのユーザーが付与する権利を持つ権限を、新規ユーザーが持つようにするには、そのユーザーに次の権限を付与します。

```
GRANT INSERT(user) ON mysql.user TO 'user_name'@'host_name';
```

これで、ユーザーは権限カラムを直接変更できませんが、`GRANT` ステートメントを使用してほかのユーザーに権限を与えることができるようになります。

- `--skip-grant-tables`

コマンド行形式	<code>--skip-grant-tables[={OFF ON}]</code>
型	Boolean

デフォルト値	OFF
--------	-----

このオプションは、サーバーの起動順序に影響します:

- `--skip-grant-tables` により、サーバーは `mysql` システムスキーマ内の付与テーブルを読み取らないため、権限システムをまったく使用せずに起動します。これにより、すべてのユーザーがサーバーすべてのデータベースへの無制限アクセスにアクセスできるようになります。

`--skip-grant-tables` を使用してサーバーを起動すると認証チェックが無効になるため、この場合、サーバーは `skip_networking` を有効にしてリモート接続も無効にします。

`--skip-grant-tables` で起動されたサーバーが実行時に付与テーブルをロードするようにするには、次の方法で権限フラッシュ操作を実行します:

- サーバーへの接続後に MySQL `FLUSH PRIVILEGES` ステートメントを発行します。
- コマンドラインから `mysqladmin flush-privileges` または `mysqladmin reload` コマンドを実行します。

権限のフラッシュは、起動後に他のアクションが実行された結果として暗黙的に発生する可能性もあるため、サーバーは付与テーブルの使用を開始します。たとえば、起動シーケンス中にアップグレードを実行すると、サーバーは権限をフラッシュします。

- `--skip-grant-tables` では、ログイン失敗追跡および一時アカウントロックが無効になります。これらの機能は付与テーブルに依存するためです。 [セクション6.2.15「パスワード管理」](#) を参照してください。
- `--skip-grant-tables` では、データディクショナリまたは `mysql` システムスキーマに登録されている他の特定のオブジェクトがサーバーによってロードされません:
 - `CREATE EVENT` を使用してインストールされ、`events` データディクショナリテーブルに登録されたスケジュール済イベント。
 - `INSTALL PLUGIN` を使用してインストールされ、`mysql.plugin` システムテーブルに登録されたプラグイン。

`--skip-grant-tables` を使用していてもプラグインがロードされるようにするには、`--plugin-load` または `--plugin-load-add` オプションを使用します。
 - `CREATE FUNCTION` を使用してインストールされ、`mysql.func` システムテーブルに登録されているユーザー定義関数 (UDF)。

`--skip-grant-tables` は、コンポーネントの起動時にロードを抑制しません。
- `--skip-grant-tables` では、`disabled_storage_engines` システム変数は無効になります。
- `--skip-host-cache`

コマンド行形式	<code>--skip-host-cache</code>
---------	--------------------------------

名前と IP の解決を高速化するために内部ホストキャッシュの使用を無効にします。キャッシュを無効にすると、サーバーはクライアントが接続するたびに DNS ルックアップを実行します。

`--skip-host-cache` の使用は `host_cache_size` システム変数を 0 に設定することに似ていますが、`host_cache_size` の方が柔軟性が高く、これはサーバー起動時だけでなく実行時にもホストキャッシュのサイズを変更したり有効化または無効化したりするために使用できるためです。

`--skip-host-cache` を使用してサーバーを起動しても、`host_cache_size` の値に対する実行時の変更は妨げられませんが、このような変更は効果がなく、`host_cache_size` が 0 より大きい値に設定されていてもキャッシュは再度有効になりません。

ホストキャッシュの動作の詳細は、 [セクション5.1.12.3「DNS ルックアップとホストキャッシュ」](#) を参照してください。

- `--skip-innodb`

InnoDB ストレージエンジンを無効にします。この場合、デフォルトのストレージエンジンは InnoDB であるため、`--default-storage-engine` および `--default-tmp-storage-engine` を使用して永続テーブルと TEMPORARY テーブルの両方のデフォルトをほかのエンジンに設定しないかぎり、サーバーは起動しません。

InnoDB ストレージエンジンを無効にすることはできず、`--skip-innodb` オプションは非推奨であり、効果はありません。これを使用すると警告が出ます。このオプションは、将来の MySQL リリースで削除される予定です。

- `--skip-new`

コマンド行形式	<code>--skip-new</code>
---------	-------------------------

このオプションは、安全でない可能性のある新しい動作を無効にします (考慮する必要があります)。これらの設定になります: `delay_key_write=OFF`, `concurrent_insert=NEVER`, `automatic_sp_privileges=OFF`。また、`OPTIMIZE TABLE` がサポートされていないストレージエンジンの `ALTER TABLE` にも `OPTIMIZE TABLE` がマップされます。

- `--skip-show-database`

コマンド行形式	<code>--skip-show-database</code>
システム変数	<code>skip_show_database</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ

このオプションは、`SHOW DATABASES` ステートメントを使用することが許可されているユーザーを制御する `skip_show_database` システム変数を設定します。セクション 5.1.8 「サーバーシステム変数」を参照してください。

- `--skip-stack-trace`

コマンド行形式	<code>--skip-stack-trace</code>
---------	---------------------------------

スタックトレースを書き込みません。このオプションは、デバッグで `mysqld` を実行するときに役立ちます。一部のシステムでは、コアファイルを取得するために、このオプションの使用が必要になることもあります。セクション 5.9 「MySQL のデバッグ」を参照してください。

- `--slow-start-timeout=timeout`

コマンド行形式	<code>--slow-start-timeout=#</code>
型	Integer
デフォルト値	15000

このオプションは、Windows サービスコントロールマネージャーのサービス開始タイムアウトを制御します。この値は、起動時に Windows サービスを強制終了する前に、サービスコントロールマネージャーが待機する最大のミリ秒数です。デフォルト値は 15000 (15 秒) です。MySQL サービスの開始に時間がかかりすぎる場合、この値を増やすことが必要なこともあります。値 0 は、タイムアウトがないことを意味します。

- `--socket=path`

コマンド行形式	<code>--socket={file_name pipe_name}</code>
システム変数	<code>socket</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

デフォルト値 (Windows)	MySQL
デフォルト値 (その他)	/tmp/mysql.sock

UNIX の場合、このオプションは、ローカル接続用の listen を行うときに使用する UNIX ソケットファイルを指定します。デフォルト値は `/tmp/mysql.sock` です。このオプションが指定された場合、別のディレクトリを指定する絶対パス名が指定されないかぎり、サーバーはデータディレクトリにファイルを作成します。Windows の場合、このオプションは、名前付きパイプを使用する、ローカル接続用の listen を行うときに使用するパイプ名を指定します。デフォルト値は `MySQL` です (大/小文字は区別されません)。

- `--sql-mode=value[,value[,value...]]`

コマンド行形式	<code>--sql-mode=name</code>
システム変数	<code>sql_mode</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Set
デフォルト値	<code>ONLY_FULL_GROUP_BY STRICT_TRANS_TABLES NO_ZERO_IN_DATE NO_ZERO_DATE ERROR_FOR_DIVISION_BY_ZERO NO_ENGINE_SUBSTITUTION</code>
有効な値	<code>ALLOW_INVALID_DATES ANSI_QUOTES ERROR_FOR_DIVISION_BY_ZERO HIGH_NOT_PRECEDENCE IGNORE_SPACE NO_AUTO_VALUE_ON_ZERO NO_BACKSLASH_ESCAPES NO_DIR_IN_CREATE NO_ENGINE_SUBSTITUTION NO_UNSIGNED_SUBTRACTION NO_ZERO_DATE NO_ZERO_IN_DATE ONLY_FULL_GROUP_BY PAD_CHAR_TO_FULL_LENGTH PIPES_AS_CONCAT REAL_AS_FLOAT STRICT_ALL_TABLES STRICT_TRANS_TABLES</code>

TIME_TRUNCATE_FRACTIONAL

SQL モードを設定します。 [セクション5.1.11「サーバー SQL モード」](#) を参照してください。

注記

MySQL インストールプログラムはインストールプロセス中に SQL モードを構成することがあります。

SQL モードがデフォルトまたは期待されているモードと異なる場合、サーバーが起動時に読み取るオプションファイル内の設定を確認してください。

- `--ssl, --skip-ssl`

コマンド行形式	<code>--ssl[={OFF ON}]</code>
無効化	<code>skip-ssl</code>
型	Boolean
デフォルト値	ON

`--ssl` オプションは、サーバーが暗号化された接続を許可するが必要としないことを指定します。このオプションはデフォルトで有効となっています。

`--ssl` は、否定形式で `--skip-ssl` またはシノニム (`--ssl=OFF`、`--disable-ssl`) として指定できます。この場合、このオプションは、`tls_XXX` および `ssl_XXX` システム変数の設定に関係なく、サーバーが暗号化された接続を許可しないことを指定します。

`--ssl` オプションは、サーバーの起動時に、サーバーが暗号化された接続をサポートしているかどうかのみに有効です。これは無視され、実行時の `ALTER INSTANCE RELOAD TLS` の操作には影響しません。たとえば、`--ssl=OFF` を使用して、暗号化された接続を無効にしてサーバーを起動し、TLS を再構成して `ALTER INSTANCE RELOAD TLS` を実行し、実行時に暗号化された接続を有効にできます。

クライアントが SSL を使用して接続できるかどうかの構成、および SSL キーと証明書の検索場所の指定の詳細は、[セクション6.3.1「暗号化接続を使用するための MySQL の構成」](#) を参照してください。ここでは、証明書とキーファイルの自動生成および自動検出のためのサーバー機能についても説明します。サーバー側で `ssl_cert` および `ssl_key` システム変数を設定し、クライアント側で `--ssl-ca` (または `--ssl-capath`) オプションを設定することを検討してください。

- `--standalone`

コマンド行形式	<code>--standalone</code>
プラットフォーム固有	Windows

Windows でのみ使用可能で、MySQL Server にサービスとして実行しないよう指示します。

- `--super-large-pages`

コマンド行形式	<code>--super-large-pages[={OFF ON}]</code>
プラットフォーム固有	Solaris
型	Boolean
デフォルト値	OFF

MySQL での標準的な大規模ページの使用では、サポートされる最大サイズである 4M バイトまでの使用が試行されます。Solaris では「超大規模ページ」機能により 256M バイトまでのページの使用が可能です。この機能は最新の SPARC プラットフォームで使用できます。これは `--super-large-pages` または `--skip-super-large-pages` オプションを使用して有効または無効にできます。

- `--symbolic-links`, `--skip-symbolic-links`

コマンド行形式	<code>--symbolic-links[={OFF ON}]</code>
非推奨	はい
型	Boolean
デフォルト値	OFF

シンボリックリンクサポートを有効または無効にします。Unix でシンボリックリンクを有効にすると、`CREATE TABLE` ステートメントの `INDEX DIRECTORY` または `DATA DIRECTORY` オプションを使用して、`MyISAM` インデックスファイルまたはデータファイルを別のディレクトリにリンクできます。テーブルを削除したり名前変更したりすると、そのシンボリックリンクが指定するファイルも削除されたり名前が変更されたりします。[セクション 8.12.2.2 「Unix 上の MyISAM へのシンボリックリンクの使用」](#) を参照してください。

注記

シンボリックリンクのサポートは、それを制御する `--symbolic-links` オプションとともに非推奨になりました。将来のバージョンの MySQL で削除される予定です。また、このオプションはデフォルトで無効になっています。関連する `have_symlink` システム変数も非推奨になりました。将来のバージョンの MySQL で削除される予定です。

このオプションは Windows では意味がありません。

- `--sysdate-is-now`

コマンド行形式	<code>--sysdate-is-now[={OFF ON}]</code>
型	Boolean
デフォルト値	OFF

デフォルトの `SYSDATE()` は、この関数があるステートメントの実行が開始された時間ではなく、この関数が実行された時間を返します。これは `NOW()` の動作と異なります。このオプションにより、`SYSDATE()` が `NOW()` のシノニムになります。バイナリロギングおよびレプリケーションに対する意味については、[セクション 12.7 「日および時間関数」](#) の `SYSDATE()` および [セクション 5.1.8 「サーバーシステム変数」](#) の `SET TIMESTAMPTO` についての説明を参照してください。

- `--tc-heuristic-recover={COMMIT|ROLLBACK}`

コマンド行形式	<code>--tc-heuristic-recover=name</code>
型	列挙
デフォルト値	OFF
有効な値	OFF COMMIT ROLLBACK

手動ヒューリスティックリカバリで使用する決定。

`--tc-heuristic-recover` オプションを指定すると、手動ヒューリスティックリカバリが成功したかどうかに関係なく、サーバーは終了します。

2 フェーズコミットが可能なストレージエンジンが複数あるシステムでは、`ROLLBACK` オプションは安全ではなく、次のエラーで回復が停止します:

```
[ERROR] --tc-heuristic-recover rollback
strategy is not safe on systems with more than one 2-phase-commit-capable
storage engine. Aborting crash recovery.
```

- `--transaction-isolation=level`

コマンド行形式	<code>--transaction-isolation=name</code>
システム変数	<code>transaction_isolation</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	REPEATABLE-READ
有効な値	READ-UNCOMMITTED READ-COMMITTED REPEATABLE-READ SERIALIZABLE

デフォルトのトランザクション分離レベルを設定します。level 値は、`READ-UNCOMMITTED`、`READ-COMMITTED`、`REPEATABLE-READ`、または `SERIALIZABLE` に設定できます。セクション13.3.7「`SET TRANSACTION ステートメント`」を参照してください。

デフォルトのトランザクション分離レベルは、`SET TRANSACTION` ステートメントを使用するか、`transaction_isolation` システム変数を設定して、実行時に設定することもできます。

- `--transaction-read-only`

コマンド行形式	<code>--transaction-read-only[={OFF ON}]</code>
システム変数	<code>transaction_read_only</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

デフォルトのトランザクションアクセスモードを設定します。デフォルトでは読み取り専用モードが無効化されているため、モードは読み取り/書き込みです。

実行時にデフォルトのトランザクションアクセスモードを設定するには、`SET TRANSACTION` ステートメントを使用するか、`transaction_read_only` システム変数を設定します。セクション13.3.7「`SET TRANSACTION ステートメント`」を参照してください。

- `--tmpdir=dir_name, -t dir_name`

コマンド行形式	<code>--tmpdir=dir_name</code>
システム変数	<code>tmpdir</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ディレクトリ名

一時ファイルを作成するために使用するディレクトリのパス。これは、小さすぎて一時テーブルを保持できないパーティション上にデフォルトの `/tmp` ディレクトリがある場合に役立つことがあります。このオプション

は、ラウンドロビン方式で使用されるいくつかのパスを受け入れます。パスは、Unix ではコロン文字 (:) で区切り、Windows ではセミコロン文字 (;) で区切る必要があります。

`--tmpdir` は、メモリーベースのファイルシステム上のディレクトリや、サーバーホストの再起動時にクリアされるディレクトリなど、非永続的な場所にすることができます。MySQL サーバーがレプリカとして機能しており、`--tmpdir` に非永続的な場所を使用している場合は、`slave_load_tmpdir` システム変数を使用してレプリカに別の一時ディレクトリを設定することを検討してください。レプリカの場合、`LOAD DATA` ステートメントのレプリケートに使用される一時ファイルはこのディレクトリに格納されるため、永続的な場所ではマシンの再起動後も存続できますが、一時ファイルが削除されている場合は、再起動後もレプリケーションを続行できるようになりました。

一時ファイルのストレージ位置に関しては、[セクションB.3.3.5「MySQL が一時ファイルを格納する場所」](#)を参照してください。

- `--upgrade=value`

コマンド行形式	<code>--upgrade=value</code>
導入	8.0.16
型	列挙
デフォルト値	AUTO
有効な値	AUTO NONE MINIMAL

FORCE

このオプションは、サーバーが起動時に自動アップグレードを実行するかどうか、およびその方法を制御します。自動アップグレードには、次の2つのステップが含まれます:

- ステップ 1: データディクショナリのアップグレード。

このステップでは、次のアップグレードを行います:

- `mysql` スキーマ内のデータディクショナリテーブル。実際のデータディクショナリのバージョンが現在予想されているバージョンより低い場合、サーバーはデータディクショナリをアップグレードします。実行できない場合、または実行できない場合、サーバーは実行できません。
 - パフォーマンススキーマと `INFORMATION_SCHEMA`。
- ステップ 2: サーバーのアップグレード。

このステップは、他のすべてのアップグレードタスクで構成されます。既存のインストールデータの MySQL バージョンがサーバーの予想より低い場合は、アップグレードする必要があります:

- `mysql` スキーマ内のシステムテーブル (残りの非データディクショナリテーブル)。
- `sys` スキーマ。
- ユーザースキーマ。

アップグレードステップ 1 および 2 の詳細は、[セクション2.11.3「MySQL のアップグレードプロセスの内容」](#) を参照してください。

次の `--upgrade` オプション値を使用できます:

- `AUTO`

サーバーは、最新でないことが判明したものの自動アップグレードを実行します (ステップ 1 および 2)。これは、`--upgrade` が明示的に指定されていない場合のデフォルトのアクションです。

- `NONE`

サーバーは、起動プロセス中に自動アップグレードステップを実行しません (ステップ 1 および 2 はスキップします)。このオプション値はデータディクショナリのアップグレードを妨げるため、データディクショナリが期限切れであることが判明した場合、サーバーはエラーで終了します:

```
[ERROR] [MY-013381] [Server] Server shutting down because upgrade is
required, yet prohibited by the command line option '--upgrade=NONE'.
[ERROR] [MY-010334] [Server] Failed to initialize DD Storage Engine
[ERROR] [MY-010020] [Server] Data Dictionary initialization failed.
```

- `MINIMAL`

サーバーは、必要に応じてデータディクショナリ、パフォーマンススキーマおよび `INFORMATION_SCHEMA` をアップグレードします (ステップ 1)。このオプションを使用したアップグレード後は、レプリケーション内部が依存するシステムテーブルは更新されず、他の領域でも機能が低下する可能性があるため、Group Replication を起動できないことに注意してください。

- `FORCE`

サーバーは、必要に応じてデータディクショナリ、パフォーマンススキーマおよび `INFORMATION_SCHEMA` をアップグレードします (ステップ 1)。また、サーバーは他のすべてのものを強制的にアップグレードします (ス

トップ 2)。サーバーはすべてのスキーマ内のすべてのオブジェクトをチェックするため、このオプションではサーバーの起動に時間がかかります。

FORCE は、必要ないとサーバーが判断した場合に、ステップ 2 のアクションを強制的に実行する場合に役立ちます。たとえば、システムテーブルが欠落しているか破損して修復を強制する必要があると考えられる場合があります。

次のテーブルは、各オプション値に対してサーバーが実行するアクションをまとめたものです。

オプション値	サーバーはステップ 1 を実行しますか。	サーバーはステップ 2 を実行しますか。
AUTO	必要に応じて	必要に応じて
NONE	いいえ	いいえ
MINIMAL	必要に応じて	いいえ
FORCE	必要に応じて	はい

- `--user={user_name|user_id}, -u {user_name|user_id}`

コマンド行形式	<code>--user=name</code>
型	文字列

`mysqld` サーバーを、名前 `user_name` または数字ユーザー ID `user_id` を持つユーザーとして実行します。(このコンテキストでの「ユーザー」は、システムログインアカウントであり、付与テーブルにリストされている MySQL ユーザーではありません。)

`mysqld` を `root` として起動する場合、このオプションは必須です。サーバーは起動シーケンス中にそのユーザー ID を変更し、`root` ではなく特定のユーザーでこれを実行します。セクション 6.1.1 「セキュリティガイドライン」を参照してください。

セキュリティホールを回避するため、つまりユーザーが `--user=root` オプションを `my.cnf` ファイルに追加することが原因で、サーバーが `root` として稼働できないようにするために、`mysqld` で最初に指定した `--user` オプションだけを使用し、複数の `--user` オプションがあった場合に警告を生成します。`/etc/my.cnf` および `$MYSQL_HOME/my.cnf` 内のオプションは、コマンド行のオプションより先に処理することになるため、`--user` オプションを `/etc/my.cnf` に含めた上で、`root` 以外の値を指定することを推奨します。`/etc/my.cnf` 内のオプションがほかの `--user` オプションよりも先に検出されることになるので、サーバーは確実に `root` 以外のユーザーとして実行することになり、別の `--user` オプションが検出されると警告を出します。

- `--validate-config`

コマンド行形式	<code>--validate-config[={OFF ON}]</code>
導入	8.0.16
型	Boolean
デフォルト値	<code>OFF</code>

サーバーの起動構成を検証します。エラーが見つからない場合、サーバーは終了コード 0 で終了します。エラーが見つかった場合、サーバーは診断メッセージを表示し、終了コード 1 で終了します。警告および情報メッセージは、`log_error_verbosity` の値によっては表示されることもあります。即時検証の終了や終了コード 1 は生成されません。詳細は、セクション 5.1.3 「サーバー構成の検証」を参照してください。

- `--verbose, -v`

詳細なヘルプを得るには、このオプションを `--help` オプションと一緒に使用します。

- `--version, -V`

バージョン情報を表示して終了します。

5.1.8 サーバーシステム変数

MySQL サーバーは、その操作を構成する多くのシステム変数を保持します。各システム変数にはデフォルト値があります。システム変数は、コマンド行のオプションを使用するか、オプションファイルでサーバー起動時に設定できます。これらのほとんどは、実行時に `SET` ステートメントを使用して動的に変更できます。これにより、サーバーを停止して再起動しなくても、サーバーの操作を変更できます。式でシステム変数値を使用することもできます。

グローバルシステム変数のランタイム値を設定するには、通常、`SYSTEM_VARIABLES_ADMIN` 権限 (または非推奨の `SUPER` 権限) が必要です。セッションシステムランタイム変数値を設定する場合、通常は特別な権限は必要なく、すべてのユーザーが実行できますが、例外があります。詳細は、[セクション 5.1.9.1 「システム変数権限」](#) を参照してください。

システム変数の名前と値を表示するにはいくつかの方法があります。

- サーバーが使用する値を、コンパイル済みのデフォルト値と、そのサーバーが読み取るオプションファイルに基づいて表示するには、次のコマンドを使用します:

```
mysql --verbose --help
```

- コンパイルされたデフォルト値のみに基づいてサーバーが使用する値を表示し、オプションファイルの設定を無視するには、次のコマンドを使用します:

```
mysql --no-defaults --verbose --help
```

- 実行中のサーバーで使用されている現在の値を表示するには、`SHOW VARIABLES` ステートメントまたはパフォーマンススキーマシステム変数テーブルを使用します。[セクション 27.12.14 「パフォーマンススキーマシステム変数テーブル」](#) を参照してください。

このセクションでは各システム変数について説明します。システム変数サマリーテーブルについては、[セクション 5.1.5 「サーバーシステム変数リファレンス」](#) を参照してください。システム変数の操作の詳細は、[セクション 5.1.9 「システム変数の使用」](#) を参照してください。

追加のシステム変数情報については、次のセクションを参照してください。

- [セクション 5.1.9 「システム変数の使用」](#) では、システム変数値の設定および表示の構文について説明します。
- [セクション 5.1.9.2 「動的システム変数」](#) では、実行時に設定できる変数を一覧表示しています。
- システム変数の調整に関する情報は、[セクション 5.1.1 「サーバーの構成」](#) を参照してください。
- [セクション 15.14 「InnoDB の起動オプションおよびシステム変数」](#) では、InnoDB システム変数を一覧表示しています。
- [NDB Cluster システム変数](#) では、NDB Cluster に固有のシステム変数を一覧表示します。
- レプリケーションに固有のサーバーシステム変数については、[セクション 17.1.6 「レプリケーションおよびバイナリロギングのオプションと変数」](#) を参照してください。

注記

次の変数説明の一部では、変数を「有効にする」または「無効にする」ことについて述べています。これらの変数は `SET` ステートメントを `ON` または `1` に設定すると有効になり、あるいは `OFF` または `0` に設定すると無効になります。ブール変数は、起動時に値 `ON`、`TRUE`、`OFF`、`FALSE` (大/小文字の区別なし)、および `1` と `0` に設定できます。[セクション 4.2.2.4 「プログラムオプション修飾子」](#) を参照してください。

一部のシステム変数はバッファまたはキャッシュのサイズを制御します。所定のバッファについて、サーバーは内部データ構造を割り当てる必要がある場合もあります。これらの構造は、バッファに割り当てられた合計メモリから割り当てられ、必要なスペースの量はプラットフォームに依存することがあります。つまり、バッファサイズを制御するシステム変数に値を割り当てたとき、実際に使用可能なスペースの量が、割り当てられた値と異なる場合もあることを意味します。一部の 경우에는、この量は割り当てられた値より少ないこともあります。サーバーが値を上方に調整することもできます。たとえば、最小値が 1024 の変数に値 0 を割り当てると、サーバーは値を 1024 に設定します。

バッファサイズ、長さ、およびスタックサイズの値は、別途指定しないかぎりバイト単位で指定されます。

一部のシステム変数はファイル名の値を取ります。別途指定しないかぎり、値が相対パス名であれば、デフォルトのファイルの場所はデータディレクトリです。場所を明示的に指定するには、絶対パス名を使用します。たとえばデー

タディレクトリが `/var/mysql/data` だとします。ファイル値変数を相対パス名として指定すると、`/var/mysql/data` の下に配置されます。値が絶対パス名である場合、その場所はパス名によって指定されます。

- [activate_all_roles_on_login](#)

コマンド行形式	<code>--activate-all-roles-on-login[={OFF ON}]</code>
システム変数	activate_all_roles_on_login
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

ユーザーがサーバーにログインしたときに、付与されたすべてのロールの自動アクティブ化を有効にするかどうか:

- [activate_all_roles_on_login](#) が有効な場合、サーバーはログイン時に各アカウントに付与されたすべてのロールをアクティブ化します。これは、[SET DEFAULT ROLE](#) で指定されたデフォルトのロールよりも優先されます。
- [activate_all_roles_on_login](#) が無効になっている場合、サーバーはログイン時に [SET DEFAULT ROLE](#) で指定されたデフォルトのロール (存在する場合) をアクティブ化します。

付与されるロールには、ユーザーに明示的に付与されるロールと、[mandatory_roles](#) システム変数値で指定されるロールが含まれます。

[activate_all_roles_on_login](#) は、ログイン時、および定義者コンテキストで実行されるストアプログラムおよびビューの実行開始時にのみ適用されます。セッション内のアクティブなロールを変更するには、[SET ROLE](#) を使用します。ストアプログラムのアクティブなロールを変更するには、プログラム本体で [SET ROLE](#) を実行する必要があります。

- [admin_address](#)

コマンド行形式	<code>--admin-address=addr</code>
導入	8.0.14
システム変数	admin_address
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

管理ネットワークインタフェースで TCP/IP 接続をリスニングする IP アドレス ([セクション5.1.12.1「接続インタフェース」](#) を参照)。デフォルトの [admin_address](#) 値はありません。この変数が起動時に指定されない場合、サーバーは管理インタフェースを維持しません。サーバーには、通常の (非管理) クライアント TCP/IP 接続を構成するための [bind_address](#) システム変数もあります。 [セクション5.1.12.1「接続インタフェース」](#) を参照してください。

[admin_address](#) が指定されている場合、その値は次の要件を満たす必要があります:

- 値は、単一の IPv4 アドレス、IPv6 アドレスまたはホスト名である必要があります。
- 値にワイルドカードアドレス書式 (*、0.0.0.0 または ::) は指定できません。
- MySQL 8.0.22 の時点では、値にネットワークネームスペース指定子が含まれる場合があります。

IP アドレスは、IPv4 または IPv6 アドレスとして指定できます。値がホスト名の場合、サーバーは名前を IP アドレスに解決し、そのアドレスにバインドします。ホスト名が複数の IP アドレスに解決される場合、サーバーは最初の IPv4 アドレス (存在する場合) または最初の IPv6 アドレスを使用します。

サーバーはさまざまなタイプのアドレスを次のように処理します。

- アドレスが IPv4 にマップ済みのアドレスの場合、サーバーは IPv4 または IPv6 のいずれかの形式で、そのアドレスの TCP/IP 接続を受け入れます。たとえば、サーバーが `::ffff:127.0.0.1` にバインドされている場合、クライアントは `--host=127.0.0.1` または `--host>::ffff:127.0.0.1` のいずれかを使用して接続できます。
- アドレスが「通常の」 IPv4 または IPv6 アドレスの場合 (`127.0.0.1` や `::1` など)、サーバーはその IPv4 または IPv6 アドレスについてのみ TCP/IP 接続を受け入れます。

アドレスのネットワークネームスペースの指定には、次のルールが適用されます:

- ネットワークネームスペースは、IP アドレスまたはホスト名に指定できます。
- ワイルドカード IP アドレスにはネットワークネームスペースを指定できません。
- 指定されたアドレスでは、ネットワーク名前空間はオプションです。指定する場合は、アドレスの直後に `/ns` 接尾辞として指定する必要があります。
- `/ns` 接尾辞のないアドレスは、ホストシステムのグローバルネームスペースを使用します。したがって、グローバルネームスペースがデフォルトです。
- `/ns` 接尾辞の付いたアドレスは、`ns` という名前のネームスペースを使用します。
- ホストシステムはネットワークネームスペースをサポートしている必要があり、各名前付きネームスペースは事前に設定されている必要があります。存在しないネームスペースに名前を付けると、エラーが発生します。

ネットワークネームスペースの詳細は、[セクション5.1.14「ネットワークネームスペースのサポート」](#)を参照してください。

アドレスへのバインドが失敗した場合、サーバーはエラーを生成し、起動しません。

`admin_address` システム変数は、サーバーを通常のクライアント接続のアドレスにバインドする `bind_address` システム変数と似ていますが、次の点が異なります:

- `bind_address` では、複数のアドレスが許可されます。`admin_address` では、単一のアドレスが許可されます。
 - `bind_address` では、ワイルドカードアドレスが許可されます。`admin_address` はそうではありません。
- `admin_port`

コマンド行形式	<code>--admin-port=port_num</code>
導入	8.0.14
システム変数	<code>admin_port</code>
スコープ	グローバル
動的	いいえ
<code>SET_VAR</code> ヒントの適用	いいえ
型	Integer
デフォルト値	33062
最小値	0
最大値	65535

管理ネットワークインタフェースでの接続に使用する TCP/IP ポート番号 ([セクション5.1.12.1「接続インタフェース」](#)を参照)。この変数を 0 に設定すると、デフォルト値が使用されます。

`admin_address` が指定されていない場合、サーバーは管理ネットワークインタフェースを保持しないため、`admin_port` を設定しても効果はありません。

- `admin_ssl_ca`

コマンド行形式	<code>--admin-ssl-ca=file_name</code>
---------	---------------------------------------

導入	8.0.21
システム変数	admin_ssl_ca
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	NULL

[admin_ssl_ca](#) システム変数は [ssl_ca](#) と似ていますが、メイン接続インタフェースではなく管理接続インタフェースに適用される点が異なります。管理インタフェースの暗号化サポートの構成の詳細は、[暗号化された接続に対する管理インタフェースのサポート](#) を参照してください。

- [admin_ssl_capath](#)

コマンド行形式	<code>--admin-ssl-capath=dir_name</code>
導入	8.0.21
システム変数	admin_ssl_capath
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	ディレクトリ名
デフォルト値	NULL

[admin_ssl_capath](#) システム変数は [ssl_capath](#) と似ていますが、メイン接続インタフェースではなく管理接続インタフェースに適用される点が異なります。管理インタフェースの暗号化サポートの構成の詳細は、[暗号化された接続に対する管理インタフェースのサポート](#) を参照してください。

- [admin_ssl_cert](#)

コマンド行形式	<code>--admin-ssl-cert=file_name</code>
導入	8.0.21
システム変数	admin_ssl_cert
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	NULL

[admin_ssl_cert](#) システム変数は [ssl_cert](#) と似ていますが、メイン接続インタフェースではなく管理接続インタフェースに適用される点が異なります。管理インタフェースの暗号化サポートの構成の詳細は、[暗号化された接続に対する管理インタフェースのサポート](#) を参照してください。

- [admin_ssl_cipher](#)

コマンド行形式	<code>--admin-ssl-cipher=name</code>
導入	8.0.21
システム変数	admin_ssl_cipher
スコープ	グローバル
動的	はい

SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	NULL

`admin_ssl_cipher` システム変数は `ssl_cipher` と似ていますが、メイン接続インターフェースではなく管理接続インターフェースに適用される点が異なります。管理インターフェースの暗号化サポートの構成の詳細は、[暗号化された接続に対する管理インターフェースのサポート](#) を参照してください。

- `admin_ssl_crl`

コマンド行形式	<code>--admin-ssl-crl=file_name</code>
導入	8.0.21
システム変数	<code>admin_ssl_crl</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	NULL

`admin_ssl_crl` システム変数は `ssl_crl` と似ていますが、メイン接続インターフェースではなく管理接続インターフェースに適用される点が異なります。管理インターフェースの暗号化サポートの構成の詳細は、[暗号化された接続に対する管理インターフェースのサポート](#) を参照してください。

- `admin_ssl_crlpath`

コマンド行形式	<code>--admin-ssl-crlpath=dir_name</code>
導入	8.0.21
システム変数	<code>admin_ssl_crlpath</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	ディレクトリ名
デフォルト値	NULL

`admin_ssl_crlpath` システム変数は `ssl_crlpath` と似ていますが、メイン接続インターフェースではなく管理接続インターフェースに適用される点が異なります。管理インターフェースの暗号化サポートの構成の詳細は、[暗号化された接続に対する管理インターフェースのサポート](#) を参照してください。

- `admin_ssl_key`

コマンド行形式	<code>--admin-ssl-key=file_name</code>
導入	8.0.21
システム変数	<code>admin_ssl_key</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	ファイル名

デフォルト値	NULL
--------	------

`admin_ssl_key` システム変数は `ssl_key` と似ていますが、メイン接続インターフェースではなく管理接続インターフェースに適用される点が異なります。管理インターフェースの暗号化サポートの構成の詳細は、[暗号化された接続に対する管理インターフェースのサポート](#) を参照してください。

- `admin_tls_ciphersuites`

コマンド行形式	<code>--admin-tls-ciphersuites=ciphersuite_list</code>
導入	8.0.21
システム変数	<code>admin_tls_ciphersuites</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	NULL

`admin_tls_ciphersuites` システム変数は `tls_ciphersuites` と似ていますが、メイン接続インターフェースではなく管理接続インターフェースに適用される点が異なります。管理インターフェースの暗号化サポートの構成の詳細は、[暗号化された接続に対する管理インターフェースのサポート](#) を参照してください。

- `admin_tls_version`

コマンド行形式	<code>--admin-tls-version=protocol_list</code>
導入	8.0.21
システム変数	<code>admin_tls_version</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	TLSv1,TLSv1.1,TLSv1.2,TLSv1.3 (OpenSSL 1.1.1 以上) TLSv1,TLSv1.1,TLSv1.2 (otherwise)

`admin_tls_version` システム変数は `tls_version` と似ていますが、メイン接続インターフェースではなく管理接続インターフェースに適用される点が異なります。管理インターフェースの暗号化サポートの構成の詳細は、[暗号化された接続に対する管理インターフェースのサポート](#) を参照してください。

- `authentication_windows_log_level`

コマンド行形式	<code>--authentication-windows-log-level=#</code>
システム変数	<code>authentication_windows_log_level</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	2
最小値	0

最大値	4
-----	---

この変数は、[authentication_windows](#) Windows 認証プラグインが使用可能で、デバッグコードが有効な場合のみ使用できます。 [セクション6.4.1.6「Windows プラガブル認証」](#)を参照してください。

この変数は、Windows 認証プラグインのロギングレベルを設定します。 次の表は、許可される値を示しています。

値	説明
0	ロギングなし
1	エラーメッセージのみログに記録します
2	レベル 1 メッセージおよび警告メッセージをログに記録します
3	レベル 2 メッセージおよび情報メモをログに記録します
4	レベル 3 メッセージおよびデバッグメッセージをログに記録します

- [authentication_windows_use_principal_name](#)

コマンド行形式	<code>--authentication-windows-use-principal-name[={OFF ON}]</code>
システム変数	authentication_windows_use_principal_name
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

この変数は、[authentication_windows](#) Windows 認証プラグインが使用可能な場合のみ使用できます。 [セクション6.4.1.6「Windows プラガブル認証」](#)を参照してください。

`InitSecurityContext()` 関数を使用して認証するクライアントは、接続するサービスを識別する文字列を提供する必要があります (`targetName`)。MySQL は、サーバーが実行するアカウントの主体名 (UPN) を使用します。UPN は `user_id@computer_name` という形式で、使用される場所に登録される必要はありません。この UPN は、認証ハンドシェイクの最初にサーバーによって送信されます。

この変数は、サーバーが初期チャレンジで UPN を送信するかどうかを制御します。デフォルトでは、変数は有効になっています。セキュリティ上の理由から、サーバーアカウント名をクリアテキストとしてクライアントに送信しないように無効にできます。変数が無効な場合、サーバーは最初のチャレンジで常に `0x00` バイトを送信し、クライアントは `targetName` を指定せず、結果として NTLM 認証が使用されます。

サーバーが UPN の取得に失敗した場合 (主に Kerberos 認証をサポートしていない環境で発生)、UPN はサーバーによって送信されず、NTLM 認証が使用されます。

- [自動コミット](#)

コマンド行形式	<code>--autocommit[={OFF ON}]</code>
システム変数	autocommit
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean

デフォルト値	ON
--------	----

自動コミットモード。1に設定された場合、テーブルへのすべての変更はすぐに有効になります。0に設定した場合、`COMMIT`を使用してトランザクションを受け入れるか、`ROLLBACK`でトランザクションをキャンセルする必要があります。`autocommit`が0で、これを1に変更した場合、MySQLはオープン中のすべてのトランザクションの自動的な`COMMIT`を実行します。トランザクションを始める別の方法は、`START TRANSACTION`または`BEGIN`ステートメントを利用する方法です。[セクション13.3.1「START TRANSACTION、COMMIT および ROLLBACK ステートメント」](#)を参照してください。

デフォルトでは、クライアント接続は`autocommit`を1に設定して開始されます。デフォルト0でクライアントを開始させるには、`--autocommit=0` オプションを使用してサーバーを開始することによって、グローバルな`autocommit`値を設定します。オプションファイルを使用して変数を設定するには、次の行を含めます。

```
[mysqld]
autocommit=0
```

- [automatic_sp_privileges](#)

コマンド行形式	<code>--automatic-sp-privileges[={OFF ON}]</code>
システム変数	<code>automatic_sp_privileges</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

この変数の値が1(デフォルト)のとき、ユーザーがルーチンを実行して変更したりドロップしたりできない場合、サーバーは自動的に`EXECUTE`および`ALTER ROUTINE`の権限をストアルーチンの作成者に付与します。(ルーチンをドロップするには`ALTER ROUTINE`権限が必要です。)ルーチンがドロップされると、サーバーはそれらの権限を作成者から自動的にドロップします。`automatic_sp_privileges`が0の場合、サーバーはこれらの権限を自動的に追加またはドロップしません。

ルーチンの作成者は、ルーチンの`CREATE`ステートメントを実行するために使用されるアカウントです。これは、ルーチン定義で`DEFINER`として名前が指定されているアカウントと同じでないことがあります。

`--skip-new`を使用して`mysqld`を起動すると、`automatic_sp_privileges`は`OFF`に設定されます。

[セクション25.2.2「ストアルーチンとMySQL権限」](#)も参照してください。

- [auto_generate_certs](#)

コマンド行形式	<code>--auto-generate-certs[={OFF ON}]</code>
システム変数	<code>auto_generate_certs</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

この変数は、サーバーがSSLキーおよび証明書ファイルをデータディレクトリに自動生成するかどうかを制御します(まだ存在しない場合)。

起動時に、`auto_generate_certs`システム変数が有効で、`--ssl`以外のSSLオプションが指定されておらず、サーバー側のSSLファイルがデータディレクトリから欠落している場合、サーバーはサーバー側およびクライアント側のSSL証明書とキーファイルをデータディレクトリに自動的に生成します。これらのファイルにより、SSLを使

用したセキュアなクライアント接続が可能になります。セクション6.3.1「暗号化接続を使用するための MySQL の構成」を参照してください。

ファイル名や特性など、SSL ファイルの自動生成の詳細は、セクション6.3.3.1「MySQL を使用した SSL および RSA 証明書とキーの作成」を参照してください

`sha256_password_auto_generate_rsa_keys` および `caching_sha2_password_auto_generate_rsa_keys` システム変数は関連していますが、暗号化されていない RSA 接続を使用したセキュアなパスワード交換に必要な RSA キーペアファイルの自動生成を制御します。

- `avoid_temporal_upgrade`

コマンド行形式	<code>--avoid-temporal-upgrade[={OFF ON}]</code>
非推奨	はい
システム変数	<code>avoid_temporal_upgrade</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

この変数は、ALTER TABLE が、5.6.4 より前の形式 (TIME、DATETIME および TIMESTAMP カラムで小数秒精度をサポートしない) で検出された一時カラムを暗黙的にアップグレードするかどうかを制御します。このようなカラムをアップグレードするには、テーブルの再構築が必要です。これにより、操作に適用される可能性のある高速変更が使用されなくなります。

この変数はデフォルトでは無効になっています。これを有効にすると、ALTER TABLE は時間的カラムを再構築しないため、可能な高速変更を利用できます。

この変数は非推奨です。将来の MySQL リリースで削除される予定です。

- `back_log`

コマンド行形式	<code>--back-log=#</code>
システム変数	<code>back_log</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動サイズ設定を示します。このリテラル値を割り当てないでください)
最小値	1
最大値	65535

MySQL で保持できる未処理の接続リクエストの数。これはメインの MySQL スレッドが非常に短時間で非常に多くの接続リクエストを受け取る場合に効果をあらわします。次に、メインスレッドが接続を検査し新規スレッドを開始するまで (非常に短いですが) 少し時間がかかります。back_log 値は、MySQL が新規リクエストへの回答を一時的に停止するまでの短い時間に、スタック可能なリクエストの数を示します。短い時間に大量の接続が予想される場合にかぎり、これを増加する必要があります。

つまり、この値は着信 TCP/IP 接続の listen キューのサイズです。使用しているオペレーティングシステムには、このキューのサイズについて独自の制限があります。UNIX listen() システムコールのマニュアルページに、詳細情報

報があります。この変数の最大値については OS のドキュメントを確認してください。[back_log](#) をオペレーティングシステムの制限を超える設定はできません。

デフォルト値は [max_connections](#) の値です。これにより、許可されたバックログを最大許容接続数に調整できません。

- [basedir](#)

コマンド行形式	<code>--basedir=dir_name</code>
システム変数	basedir
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ディレクトリ名
デフォルト値	parent of mysqld installation directory

MySQL インストールベースディレクトリへのパス。

- [big_tables](#)

コマンド行形式	<code>--big-tables[={OFF ON}]</code>
システム変数	big_tables
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

有効にすると、サーバーはすべての一時テーブルをメモリーではなくディスクに格納します。これにより、大規模な一時テーブルを必要とする `SELECT` 操作のほとんどの `The table tbl_name is full` エラーが回避されますが、インメモリーテーブルで十分なクエリーが遅くなります。

新しい接続のデフォルト値は `OFF` です (インメモリー一時テーブルを使用)。通常、この変数を有効にする必要はありません。インメモリー internal 一時テーブルが `TempTable` ストレージエンジンによって管理され (デフォルト)、`TempTable` ストレージエンジンが占有できるメモリーの最大量を超えると、`TempTable` ストレージエンジンはディスク上の一時ファイルへのデータの格納を開始します。インメモリー一時テーブルが `MEMORY` ストレージエンジンによって管理される場合、インメモリーテーブルは必要に応じてディスクベースのテーブルに自動的に変換されます。詳細は、[セクション8.4.4「MySQL での内部一時テーブルの使用」](#)を参照してください。

- [bind_address](#)

コマンド行形式	<code>--bind-address=addr</code>
システム変数	bind_address
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	*

MySQL サーバーは、1 つ以上のネットワークソケットで TCP/IP 接続をリスニングします。各ソケットは 1 つのアドレスにバインドされますが、1 つのアドレスを複数のネットワークインタフェースにマップできます。サーバーが TCP/IP 接続をリスニングする方法を指定するには、サーバーの起動時に [bind_address](#) システム変数を設定しま

す。サーバーには、専用インタフェースでの管理接続を可能にする `admin_address` システム変数もあります。 [セクション5.1.12.1「接続インタフェース」](#)を参照してください。

`bind_address` が指定されている場合、その値は次の要件を満たす必要があります：

- MySQL 8.0.13 より前では、`bind_address` は単一のアドレス値を受け入れます。この値には、単一のワイルドカード以外の IP アドレスまたはホスト名、あるいは複数のネットワークインタフェース (`*`、`0.0.0.0` または `::`) でのリスニングを許可するワイルドカードアドレス形式のいずれかを指定できます。
- MySQL 8.0.13 では、`bind_address` は前述の単一の値またはカンマ区切り値のリストを受け入れます。変数が複数の値のリストを指定する場合、各値は単一のワイルドカード以外の IP アドレス (IPv4 または IPv6) またはホスト名を指定する必要があります。ワイルドカードアドレス書式 (`*`、`0.0.0.0` または `::`) は、値リストでは使用できません。
- MySQL 8.0.22 の時点では、アドレスにネットワーク名前空間指定子を含めることができます。

IP アドレスは、IPv4 または IPv6 アドレスとして指定できます。ホスト名である値の場合、サーバーは名前を IP アドレスに解決し、そのアドレスにバインドします。ホスト名が複数の IP アドレスに解決される場合、サーバーは最初の IPv4 アドレス (存在する場合) または最初の IPv6 アドレスを使用します。

サーバーはさまざまなタイプのアドレスを次のように処理します。

- アドレスが `*` の場合、サーバーはすべてのサーバーホスト IPv4 インタフェースで TCP/IP 接続を受け入れ、サーバーホストが IPv6 をサポートしている場合はすべての IPv6 インタフェースで TCP/IP 接続を受け入れます。すべてのサーバーインタフェース上の IPv4 および IPv6 の両方の接続を許可するには、このアドレスを使用します。この値がデフォルトです。変数で複数の値のリストが指定されている場合、この値は許可されません。
- アドレスが `0.0.0.0` の場合、サーバーはすべてのサーバーホスト IPv4 インタフェース上の TCP/IP 接続を受け入れます。変数で複数の値のリストが指定されている場合、この値は許可されません。
- アドレスが `::` の場合、サーバーはすべてのサーバーホスト IPv4 および IPv6 インタフェース上の TCP/IP 接続を受け入れます。変数で複数の値のリストが指定されている場合、この値は許可されません。
- アドレスが IPv4 にマップ済みのアドレスの場合、サーバーは IPv4 または IPv6 のいずれかの形式で、そのアドレスの TCP/IP 接続を受け入れます。たとえば、サーバーが `::ffff:127.0.0.1` にバインドされている場合、クライアントは `--host=127.0.0.1` または `--host>::ffff:127.0.0.1` のいずれかを使用して接続できます。
- アドレスが「通常の」IPv4 または IPv6 アドレスの場合 (`127.0.0.1` や `::1` など)、サーバーはその IPv4 または IPv6 アドレスについてのみ TCP/IP 接続を受け入れます。

アドレスのネットワークネームスペースの指定には、次のルールが適用されます：

- ネットワークネームスペースは、IP アドレスまたはホスト名に指定できます。
- ワイルドカード IP アドレスにはネットワークネームスペースを指定できません。
- 指定されたアドレスでは、ネットワーク名前空間はオプションです。指定する場合は、アドレスの直後に `/ns` 接尾辞として指定する必要があります。
- `/ns` 接尾辞のないアドレスは、ホストシステムのグローバルネームスペースを使用します。したがって、グローバルネームスペースがデフォルトです。
- `/ns` 接尾辞の付いたアドレスは、`ns` という名前のネームスペースを使用します。
- ホストシステムはネットワークネームスペースをサポートしている必要があります、各名前付きネームスペースは事前に設定されている必要があります。存在しないネームスペースに名前を付けると、エラーが発生します。

- 変数値に複数のアドレスが指定されている場合は、グローバルネームスペース、名前付きネームスペースまたはその組合せにアドレスを含めることができます。

ネットワークネームスペースの詳細は、[セクション5.1.14「ネットワークネームスペースのサポート」](#)を参照してください。

いずれかのアドレスへのバインドが失敗した場合、サーバーはエラーを生成し、起動しません。

例:

- `bind_address=*`

サーバーは、* ワイルドカードで指定されたすべての IPv4 または IPv6 アドレスでリスニングします。

- `bind_address=198.51.100.20`

サーバーは、`198.51.100.20` IPv4 アドレスでのみリスニングします。

- `bind_address=198.51.100.20,2001:db8:0:f101::1`

サーバーは、`198.51.100.20` IPv4 アドレスおよび `2001:db8:0:f101::1` IPv6 アドレスでリスニングします。

- `bind_address=198.51.100.20,*`

`bind_address` が複数の値のリストに名前を付ける場合、ワイルドカードアドレスは許可されないため、エラーが発生します。

- `bind_address=198.51.100.20/red,2001:db8:0:f101::1/blue,192.0.2.50`

サーバーは、`red` 名前空間の `198.51.100.20` IPv4 アドレス、`blue` 名前空間の `2001:db8:0:f101::1` IPv6 アドレス、およびグローバル名前空間の `192.0.2.50` IPv4 アドレスで待機します。

`bind_address` が単一の値 (ワイルドカードまたは非ワイルドカード) に名前を付けると、サーバーは単一のソケットでリスニングし、ワイルドカードアドレスは複数のネットワークインタフェースにバインドできます。`bind_address` が複数の値のリストに名前を付けると、サーバーは値ごとに 1 つのソケットをリスニングし、各ソケットは単一のネットワークインタフェースにバインドされます。ソケットの数は、指定された値の数と線形です。オペレーティングシステムの connection-acceptance 効率によっては、長い値のリストで TCP/IP 接続を受け入れるためのパフォーマンスペナルティが発生する場合があります。

リスニングソケットおよびネットワークネームスペースファイルにはファイル記述子が割り当てられるため、`open_files_limit` システム変数を増やす必要がある場合があります。

サーバーを特定のアドレスにバインドする場合は、そのアドレスへの接続に使用できる管理権限を持つアカウントが `mysql.user` システムテーブルに含まれていることを確認してください。そうしないと、サーバーを停止できません。たとえば、サーバーを * にバインドしている場合、すべての既存のアカウントを使用して接続できます。ただし、サーバーを `::1` にバインドしている場合、そのアドレスの接続のみ受け入れます。この場合、`'root'@'::1'` アカウントが `mysql.user` テーブルに存在することをまず確認して、サーバーに接続してシャットダウンできることをたしかめます。

- `block_encryption_mode`

コマンド行形式	<code>--block-encryption-mode=#</code>
システム変数	<code>block_encryption_mode</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列

デフォルト値	aes-128-ecb
--------	-------------

この変数は、AES などのブロックベースのアルゴリズムのブロック暗号化モードを制御します。これは `AES_ENCRYPT()` および `AES_DECRYPT()` の暗号化に影響します。

`block_encryption_mode` は `aes-keylen-mode` 形式の値を取り、ここで `keylen` はビット単位の鍵の長さ、`mode` は暗号化モードです。値では大文字と小文字は区別されません。許可される `keylen` 値は 128、192、および 256 です。許可される `mode` 値は、`ECB`、`CBC`、`CFB1`、`CFB8`、`CFB128` および `OFB` です。

たとえば次のステートメントでは、AES 暗号化機能が 256 ビットの鍵の長さおよび CBC モードを使用します。

```
SET block_encryption_mode = 'aes-256-cbc';
```

サポートされない鍵の長さや SSL ライブラリがサポートしないモードを含む値に `block_encryption_mode` を設定しようとすると、エラーが発生します。

- `bulk_insert_buffer_size`

コマンド行形式	<code>--bulk-insert-buffer-size=#</code>
システム変数	<code>bulk_insert_buffer_size</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Integer
デフォルト値	8388608
最小値	0
最大値 (64 ビットプラットフォーム)	18446744073709551615
最大値 (32 ビットプラットフォーム)	4294967295

MyISAM では、空でないテーブルにデータを追加する際に、`INSERT ... SELECT`、`INSERT ... VALUES (...), (...), ...` および `LOAD DATA` のバルク挿入を高速化するために、特別なツリー形式のキャッシュを使用します。この変数は、スレッドあたりのバイト単位のキャッシュツリーのサイズを制限します。これを 0 に設定すると、この最適化が無効になります。デフォルトの値は 8M バイトです。

MySQL 8.0.14 では、このシステム変数のセッション値の設定は制限された操作です。セッションユーザーには、制限付きセッション変数を設定するのに十分な権限が必要です。セクション 5.1.9.1 「システム変数権限」を参照してください。

- `caching_sha2_password_digest_rounds`

コマンド行形式	<code>--caching-sha2-password-digest-rounds=#</code>
導入	8.0.24
システム変数	<code>caching_sha2_password_digest_rounds</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	5000
最小値	5000

最大値	4095000
-----	---------

パスワード記憶域用の `caching_sha2_password` 認証プラグインで使用されるハッシュラウンドの数。

ハッシュラウンドの数をデフォルト値より大きくすると、増加量に関連するパフォーマンスペナルティが発生します:

- `caching_sha2_password` プラグインを使用するアカウントを作成しても、アカウントが作成されるクライアントセッションには影響しませんが、サーバーはハッシュラウンドを実行して操作を完了する必要があります。
 - アカウントを使用するクライアント接続の場合、サーバーはハッシュラウンドを実行し、結果をキャッシュに保存する必要があります。その結果、最初のクライアント接続のログイン時間は長くなりますが、後続の接続のログイン時間は長くなりません。この動作は、サーバーを再起動するたびに発生します。
- `caching_sha2_password_auto_generate_rsa_keys`

コマンド行形式	<code>--caching-sha2-password-auto-generate-rsa-keys[={OFF ON}]</code>
システム変数	<code>caching_sha2_password_auto_generate_rsa_keys</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

サーバーはこの変数を使用して、RSA 秘密/公開鍵ペアファイルがまだ存在しない場合に、それらをデータディレクトリに自動生成するかどうかを決定します。

これらの条件がすべて満たされている場合、サーバーは起動時に RSA 秘密キー/公開キーのペアファイルデータをディレクトリに自動的に生成: `sha256_password_auto_generate_rsa_keys` または `caching_sha2_password_auto_generate_rsa_keys` システム変数が有効になっており、RSA オプションが指定されていません。RSA ファイルがデータディレクトリにありません。これらのキーペアファイルを使用すると、`sha256_password` または `caching_sha2_password` プラグインによって認証されたアカウントに対して、暗号化されていない RSA 接続を使用したセキュアなパスワード交換が可能になります。[セクション6.4.1.3「SHA-256 プラガブル認証」](#) および [セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#) を参照してください。

RSA ファイルの自動生成の詳細 (ファイル名や特性など) は、[セクション6.3.3.1「MySQL を使用した SSL および RSA 証明書とキーの作成」](#) を参照してください

`auto_generate_certs` システム変数は関連していますが、SSL を使用したセキュアな接続に必要な SSL 証明書およびキーファイルの自動生成を制御します。

- `caching_sha2_password_private_key_path`

コマンド行形式	<code>--caching-sha2-password-private-key-path=file_name</code>
システム変数	<code>caching_sha2_password_private_key_path</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ファイル名

デフォルト値	<code>private_key.pem</code>
--------	------------------------------

この変数は、 `caching_sha2_password` 認証プラグインの RSA 秘密キーファイルのパス名を指定します。ファイル名が相対パスとして指定された場合、サーバーのデータディレクトリを基準として解釈されます。ファイルは PEM 形式である必要があります。

重要

このファイルは秘密鍵を格納しているため、MySQL Server のみがファイルを読み取りできるようにファイルのアクセスモードを制限します。

`caching_sha2_password` の詳細は、[セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#) を参照してください。

- `caching_sha2_password_public_key_path`

コマンド行形式	<code>--caching-sha2-password-public-key-path=file_name</code>
システム変数	<code>caching_sha2_password_public_key_path</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	<code>public_key.pem</code>

この変数は、 `caching_sha2_password` 認証プラグインの RSA 公開キーファイルのパス名を指定します。ファイル名が相対パスとして指定された場合、サーバーのデータディレクトリを基準として解釈されます。ファイルは PEM 形式である必要があります。

クライアントが RSA 公開キーをリクエストする方法など、 `caching_sha2_password` の詳細は、[セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#) を参照してください。

- `character_set_client`

システム変数	<code>character_set_client</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	<code>utf8mb4</code>

クライアントから到達するステートメントの文字セット。この変数のセッション値は、クライアントがサーバーに接続するときにクライアントによってリクエストされる文字セットを使用して設定されます。(多くのクライアントは、この文字セットを明示的に指定するための `--default-character-set` オプションをサポートします。[セクション10.4「接続文字セットおよび照合順序」](#) も参照してください。) クライアントがリクエストする値が不明または利用できないか、サーバーがクライアントリクエストを無視するように構成されている場合、セッション値を設定するよう変数のグローバル値が使用されます。

- クライアントがリクエストする文字セットがサーバーで認識されない場合。たとえば、日本語に対応したクライアントが、`sjis` サポートを構成されていないサーバーに接続するときに `sjis` をリクエストする場合があります。
- クライアントの MySQL バージョンが MySQL 4.1 よりも古い場合、文字セットをリクエストしない場合。

- `mysqld` が `--skip-character-set-client-handshake` オプションを使用して開始された場合、これによってクライアント文字セット構成が無視されます。これによって MySQL 4.0 の動作が再現されるため、すべてのクライアントをアップグレードしないでサーバーをアップグレードする場合に便利です。

一部の文字セットは、クライアントの文字セットとして使用できません。これらを `character_set_client` 値として使用しようとする、エラーが発生します。許可されていないクライアント文字セットを参照してください。

- `character_set_connection`

システム変数	<code>character_set_connection</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	<code>utf8mb4</code>

文字セットイントロデューサなしで指定されたリテラルおよび数値から文字列への変換に使用される文字セット。イントロデューサについては、[セクション10.3.8「文字セットイントロデューサ」](#)を参照してください。

- `character_set_database`

システム変数	<code>character_set_database</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	<code>utf8mb4</code>
脚注	このオプションは動的ですが、サーバーによってのみ設定する必要があります。この変数は手動で設定しないでください。

デフォルトデータベースで使用される文字セット。デフォルトのデータベースが変更されるたびに、サーバーはこの変数を設定します。デフォルトデータベースが存在しない場合、変数は `character_set_server` と同じ値になります。

MySQL 8.0.14 では、このシステム変数のセッション値の設定は制限された操作です。セッションユーザーには、制限付きセッション変数を設定するのに十分な権限が必要です。[セクション5.1.9.1「システム変数権限」](#)を参照してください。

グローバル `character_set_database` および `collation_database` システム変数は非推奨です。将来のバージョンの MySQL で削除される予定です。

セッション `character_set_database` および `collation_database` システム変数への値の割当ては非推奨になり、割当てによって警告が生成されます。MySQL の将来のバージョンでは、セッション変数にアクセスしてデフォルトデータベースのデータベース文字セットおよび照合を決定できるように、セッション変数が読み取り専用になる（およびエラーを生成するために割り当てられる）ことを想定します。

- `character_set_filesystem`

コマンド行形式	<code>--character-set-filesystem=name</code>
システム変数	<code>character_set_filesystem</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ

型	文字列
デフォルト値	binary

ファイルシステムの文字セット。この変数は、[LOAD DATA](#) ステートメント、[SELECT ... INTO OUTFILE](#) ステートメント、[LOAD_FILE\(\)](#) 関数など、ファイル名を参照する文字列リテラルを解釈するために使用されます。このようなファイル名は、ファイルを開くよう試行する前に [character_set_client](#) から [character_set_filesystem](#) に変換されます。デフォルト値は [binary](#) で、変換が行われなかったことを意味します。マルチバイトファイル名が許可されるシステムについては、別の値が適切な場合もあります。たとえば、システムが UTF-8 を使用してファイル名を表す場合は、[character_set_filesystem](#) を 'utf8mb4' に設定します。

MySQL 8.0.14 では、このシステム変数のセッション値の設定は制限された操作です。セッションユーザーには、制限付きセッション変数を設定するのに十分な権限が必要です。[セクション5.1.9.1「システム変数権限」](#)を参照してください。

- [character_set_results](#)

システム変数	character_set_results
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	utf8mb4

クエリ結果をクライアントに返すために使用される文字セット。これには、カラム値、結果メタデータ (カラム名など)、エラーメッセージなどの結果データが含まれます。

- [character_set_server](#)

コマンド行形式	<code>--character-set-server=name</code>
システム変数	character_set_server
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	utf8mb4

サーバーのデフォルトの文字セット。[セクション10.15「文字セットの構成」](#)を参照してください。この変数を設定する場合は、文字セットの照合順序を指定するように [collation_server](#) も設定する必要があります。

- [character_set_system](#)

システム変数	character_set_system
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	utf8

識別子を格納するためにサーバーで使用される文字セット。この値は常に [utf8](#) です。

- [character_sets_dir](#)

コマンド行形式	<code>--character-sets-dir=dir_name</code>
---------	--

システム変数	character_sets_dir
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ディレクトリ名

文字セットがインストールされているディレクトリ。 [セクション10.15「文字セットの構成」](#) を参照してください。

- [check_proxy_users](#)

コマンド行形式	<code>--check-proxy-users[={OFF ON}]</code>
システム変数	check_proxy_users
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

一部の認証プラグインは、プロキシユーザーマッピングを実装しています (PAM や Windows 認証プラグインなど)。その他の認証プラグインは、デフォルトではプロキシユーザーをサポートしていません。その中には、付与されたプロキシ権限に従って、MySQL サーバー自体がプロキシユーザーをマップするようにリクエストできるものがあります: [mysql_native_password](#)、[sha256_password](#)。

[check_proxy_users](#) システム変数が有効になっている場合、サーバーは、このようなリクエストを行う認証プラグインに対してプロキシユーザーマッピングを実行します。ただし、サーバープロキシユーザーマッピングのサポートを利用するには、プラグイン固有のシステム変数を有効にする必要がある場合もあります:

- [mysql_native_password](#) プラグインの場合は、[mysql_native_password_proxy_users](#) を有効にします。
- [sha256_password](#) プラグインの場合は、[sha256_password_proxy_users](#) を有効にします。

ユーザープロキシの詳細は、[セクション6.2.18「プロキシユーザー」](#) を参照してください。

- [collation_connection](#)

システム変数	collation_connection
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列

接続文字セットの照合順序。[collation_connection](#) は、リテラル文字列の比較に重要です。カラム値と文字列を比較する場合、[collation_connection](#) は関係ありません。これは、カラムには照合優先度の高い独自の照合があるためです ([セクション10.8.4「式での照合の強制性」](#) を参照)。

- [collation_database](#)

システム変数	collation_database
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列

デフォルト値	utf8mb4_0900_ai_ci
脚注	このオプションは動的ですが、サーバーによってのみ設定する必要があります。この変数は手動で設定しないでください。

デフォルトデータベースで使用される照合。デフォルトのデータベースが変更されるたびに、サーバーはこの変数を設定します。デフォルトデータベースが存在しない場合、変数は `collation_server` と同じ値になります。

MySQL 8.0.18 では、このシステム変数のセッション値の設定は制限付き操作ではなくなりました。

グローバル `character_set_database` および `collation_database` システム変数は非推奨です。将来のバージョンの MySQL で削除される予定です。

セッション `character_set_database` および `collation_database` システム変数への値の割当ては非推奨になり、割当てによって警告が生成されます。MySQL の将来のバージョンでは、セッション変数にアクセスしてデフォルトデータベースのデータベース文字セットおよび照合を決定できるように、セッション変数が読み取り専用 (およびエラーを生成する割当て) になることを想定しています。

- `collation_server`

コマンド行形式	<code>--collation-server=name</code>
システム変数	<code>collation_server</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	utf8mb4_0900_ai_ci

サーバーのデフォルトの照合順序。セクション10.15「文字セットの構成」を参照してください。

- `completion_type`

コマンド行形式	<code>--completion-type=#</code>
システム変数	<code>completion_type</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	NO_CHAIN
有効な値	NO_CHAIN CHAIN RELEASE 0 1

2

トランザクション完了タイプ。この変数は、次の表に示す値を取ることができます。変数は、名前の値が対応する整数値のいずれかを使用して割り当てることができます。

値	説明
NO_CHAIN (または 0)	COMMIT および ROLLBACK は影響されません。これはデフォルト値です。
CHAIN (または 1)	COMMIT および ROLLBACK は、それぞれ COMMIT AND CHAIN および ROLLBACK AND CHAIN と同等です。(終了したばかりのトランザクションと同じ分離レベルで新規トランザクションがすぐに開始します。)
RELEASE (または 2)	COMMIT および ROLLBACK は、それぞれ COMMIT RELEASE および ROLLBACK RELEASE と同等です。(サーバーはトランザクションの終了後に切断されません。)

completion_type は、START TRANSACTION または BEGIN で開始されて COMMIT または ROLLBACK で終了するトランザクションに影響します。これは、セクション13.3.3「暗黙的なコミットを発生させるステートメント」に一覧表示されているステートメントの実行から生じる暗黙的なコミットに適用されません。また、XA COMMIT や XA ROLLBACK に対して、あるいは autocommit=1 の場合にも適用されません。

- concurrent_insert

コマンド行形式	--concurrent-insert[=value]
システム変数	concurrent_insert
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	AUTO
有効な値	NEVER AUTO ALWAYS 0 1 2

AUTO (デフォルト) の場合、MySQL ではデータファイルの中間に空きブロックがない MyISAM テーブルに対して INSERT および SELECT ステートメントを同時に実行することが許可されます。

この変数は、次の表に示す値を取ることができます。変数は、名前の値が対応する整数値のいずれかを使用して割り当てることができます。

値	説明
NEVER (または 0)	同時挿入を無効にします
AUTO (または 1)	(デフォルト) 空きブロックがない MyISAM テーブルについての同時挿入を有効にします

値	説明
ALWAYS (または 2)	空きブロックがあるテーブルであっても、すべての MyISAM テーブルについての同時挿入を有効にします。途中に空きブロックのあるテーブルが別のスレッドによって使用されている場合は、新しい行がテーブルの最後に挿入されます。そうでない場合は、MySQL は正常な書き込みロックを取得し、行を空きブロックに挿入します。

--skip-new を使用して mysqld を起動すると、concurrent_insert は NEVER に設定されます。

セクション8.11.3「同時挿入」も参照してください。

- connect_timeout

コマンド行形式	--connect-timeout=#
システム変数	connect_timeout
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	10
最小値	2
最大値	31536000

mysqld サーバーがハンドシェイクエラーを返すまでに接続パケットを待つ秒数。デフォルトは 10 秒です。

「Lost connection to MySQL server at 'XXX', system error: errno」という形式のエラーがクライアントで頻繁に発生する場合、connect_timeout 値を増やすと役立つことがあります。

- core_file

システム変数	core_file
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

サーバーが予期せず終了した場合にコアファイルを書き込むかどうか。この変数は --core-file オプションによって設定されます。

- create_admin_listener_thread

コマンド行形式	--create-admin-listener-thread[={OFF ON}]
導入	8.0.14
システム変数	create_admin_listener_thread
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean

デフォルト値	OFF
--------	-----

管理ネットワークインタフェースでクライアント接続専用のリスニングスレッドを使用するかどうか ([セクション 5.1.12.1「接続インタフェース」](#) を参照)。デフォルトは **OFF** です。つまり、メインインタフェース上の通常の接続のマネージャスレッドは、管理インタフェースの接続も処理します。

プラットフォームタイプやワークロードなどの要因によっては、この変数の一方の設定により、他方の設定よりもパフォーマンスが向上する場合があります。

`admin_address` が指定されていない場合、サーバーは管理ネットワークインタフェースを保持しないため、`create_admin_listener_thread` を設定しても効果はありません。

- `cte_max_recursion_depth`

コマンド行形式	<code>--cte-max-recursion-depth=#</code>
システム変数	<code>cte_max_recursion_depth</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1000
最小値	0
最大値	4294967295

共通テーブル式 (CTE) の最大再帰深度。サーバーは、この変数の値よりも多くのレベルを繰り返す CTE の実行を終了します。詳細は、[共通テーブル式の再帰の制限](#) を参照してください。

- `datadir`

コマンド行形式	<code>--datadir=dir_name</code>
システム変数	<code>datadir</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ディレクトリ名

MySQL サーバーデータディレクトリへのパス。相対パスは、現在のディレクトリに関して解決されます。サーバーを自動的に (つまり、事前に現在のディレクトリを認識できないコンテキストで) 起動する場合は、`datadir` 値を絶対パスとして指定することをお勧めします。

- `debug`

コマンド行形式	<code>--debug[=debug_options]</code>
システム変数	<code>debug</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値 (Unix)	<code>d:t:i:o,/tmp/mysqld.trace</code>

デフォルト値 (Windows)

d:t:i:O,\mysqld.trace

この変数は現在のデバッグ設定を指定します。これはデバッグサポートを使用して構築されたサーバーについてのみ使用できます。初期値は、サーバー起動時に指定された `--debug` オプションのインスタンスの値から取得されます。グローバル値とセッション値は、実行時に設定できます。

このシステム変数のセッション値の設定は制限された操作です。セッションユーザーには、制限付きセッション変数を設定するのに十分な権限が必要です。 [セクション5.1.9.1「システム変数権限」](#)を参照してください。

+ または - で始まる値を割り当てると、値は現在の値に追加されたり現在の値から削除されたりします。

```
mysql> SET debug = 'T';
mysql> SELECT @@debug;
+-----+
| @@debug |
+-----+
| T      |
+-----+

mysql> SET debug = '+P';
mysql> SELECT @@debug;
+-----+
| @@debug |
+-----+
| P:T    |
+-----+

mysql> SET debug = '-P';
mysql> SELECT @@debug;
+-----+
| @@debug |
+-----+
| T      |
+-----+
```

詳細については、 [セクション5.9.4「DEBUG パッケージ」](#)を参照してください。

- `debug_sync`

システム変数	<code>debug_sync</code>
スコープ	セッション
動的	はい
<code>SET_VAR</code> ヒントの適用	いいえ
型	文字列

この変数は、Debug Sync 機能へのユーザーインターフェースです。デバッグ同期を使用するには、MySQL が `DENABLE_DEBUG_SYNC=1 CMake` オプションで構成されている必要があります ([セクション2.9.7「MySQL ソース構成オプション」](#)を参照)。Debug Sync がコンパイルされていない場合、このシステム変数は使用できません。

グローバル変数値は読み取り専用で、この機能が有効かどうかを示します。デフォルトでは、Debug Sync は無効化されており、`debug_sync` の値は `OFF` です。サーバーが `--debug-sync-timeout=N` で開始した場合 (ここで、`N` は 0 より大きいタイムアウト値)、Debug Sync は有効化され、`debug_sync` の値は `ON - current signal` の後にシグナル名が続いたものになります。また、`N` は個々の同期点についてのデフォルトのタイムアウトになります。

セッション値は任意のユーザーが読み取ることができ、グローバル変数と同じ値を持ちます。セッション値は、同期ポイントを制御するように設定できます。

このシステム変数のセッション値の設定は制限された操作です。セッションユーザーには、制限付きセッション変数を設定するのに十分な権限が必要です。 [セクション5.1.9.1「システム変数権限」](#)を参照してください。

Debug Sync 機能および同期点の使用方法についての説明は、「[MySQL Internals: Test Synchronization](#)」を参照してください。

- [default_authentication_plugin](#)

コマンド行形式	<code>--default-authentication-plugin=plugin_name</code>
システム変数	default_authentication_plugin
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	<code>cached_sha2_password</code>
有効な値	<code>mysql_native_password</code> <code>sha256_password</code> <code>cached_sha2_password</code>

デフォルトの認証プラグイン 次の値を使用できます:

- `mysql_native_password`: MySQL ネイティブパスワードを使用します。 [セクション6.4.1.1「ネイティブプラグイン認証」](#) を参照してください。
- `sha256_password`: SHA-256 パスワードを使用します。 [セクション6.4.1.3「SHA-256 プラグイン認証」](#) を参照してください。
- `cached_sha2_password`: SHA-256 パスワードを使用します。 [セクション6.4.1.2「SHA-2 プラグイン認証のキャッシュ」](#) を参照してください。

注記

MySQL 8.0 では、`cached_sha2_password` が `mysql_native_password` ではなくデフォルトの認証プラグインです。 サーバー操作に対するこの変更の影響、およびサーバーとクライアントおよびコネクタとの互換性の詳細は、 [優先認証プラグインとしての `cached_sha2_password`](#) を参照してください。

`default_authentication_plugin` の値は、サーバー操作の次の側面に影響します:

- 認証プラグインを明示的に指定しない `CREATE USER` および `GRANT` ステートメントによって作成された新しいアカウントにサーバーが割り当てる認証プラグインを決定します。
- 次のステートメントで作成されたアカウントの場合、サーバーはそのアカウントをデフォルトの認証プラグインに関連付け、そのプラグインの必要に応じてハッシュされた特定のパスワードをアカウントに割り当てます:

```
CREATE USER ... IDENTIFIED BY 'cleartext password';
```

- [default_collation_for_utf8mb4](#)

システム変数	default_collation_for_utf8mb4
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
有効な値	<code>utf8mb4_0900_ai_ci</code> <code>utf8mb4_general_ci</code>

レプリケーションによる内部使用用。 このシステム変数は、`utf8mb4` 文字セットのデフォルトの照合順序に設定されます。 変数の値はソースからレプリカにレプリケートされるため、レプリカは、`utf8mb4` のデフォルトの照合順序が異なるソースからのデータを正しく処理できます。 この変数は主に、MySQL 5.7 以前のレプリケーションソー

スサーバーから MySQL 8.0 複製サーバーへのレプリケーション、または MySQL 5.7 プライマリノードと 1 つ以上の MySQL 8.0 セカンダリノードを使用したグループレプリケーションをサポートすることを目的としています。MySQL 5.7 の `utf8mb4` のデフォルトの照合は `utf8mb4_general_ci` ですが、MySQL 8.0 の `utf8mb4_0900_ai_ci` です。この変数は MySQL 8.0 より前のリリースには存在しないため、レプリカが変数の値を受信しない場合、ソースは以前のリリースのものと同みなされ、値は以前のデフォルトの照合 `utf8mb4_general_ci` に設定されます。

MySQL 8.0.18 では、このシステム変数のセッション値の設定は制限付き操作ではなくなりました。

デフォルトの `utf8mb4` 照合は、次のステートメントで使用されます:

- `SHOW COLLATION` および `SHOW CHARACTER SET`。
- `COLLATION` 句のない `CHARACTER SET utf8mb4` 句を持つ `CREATE TABLE` および `ALTER TABLE` (テーブルの文字セットまたはカラムの文字セット用)。
- `COLLATION` 句のない `CHARACTER SET utf8mb4` 句を持つ `CREATE DATABASE` および `ALTER DATABASE`。
- `COLLATE` 句のない `_utf8mb4'some text'` 形式の文字列リテラルを含むステートメント。
- `default_password_lifetime`

コマンド行形式	<code>--default-password-lifetime=#</code>
システム変数	<code>default_password_lifetime</code>
スコープ	グローバル
動的	はい
<code>SET_VAR</code> ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	65535

この変数は、グローバル自動パスワード有効期限ポリシーを定義します。デフォルトの `default_password_lifetime` 値は 0 で、パスワードの自動期限切れは無効になります。 `default_password_lifetime` の値が正の整数の `N` の場合は、許可されているパスワードの存続期間を示します。パスワードは `N` 日ごとに変更する必要があります。

グローバルパスワード有効期限ポリシーは、`CREATE USER` および `ALTER USER` ステートメントのパスワード有効期限オプションを使用して、個々のアカウントに対して必要に応じてオーバーライドできます。 [セクション 6.2.15 「パスワード管理」](#) を参照してください。

- `default_storage_engine`

コマンド行形式	<code>--default-storage-engine=name</code>
システム変数	<code>default_storage_engine</code>
スコープ	グローバル、セッション
動的	はい
<code>SET_VAR</code> ヒントの適用	いいえ
型	列挙

デフォルト値	InnoDB
--------	--------

テーブルのデフォルトのストレージエンジン。第16章「代替ストレージエンジン」を参照してください。この変数は、永続テーブルのストレージエンジンのみを設定します。TEMPORARY テーブルについてストレージエンジンを設定するには、`default_tmp_storage_engine` システム変数を設定します。

使用可能かつ有効化できるストレージエンジンを表示するには、`SHOW ENGINES` ステートメントまたはクエリー `INFORMATION_SCHEMA ENGINES` テーブルを参照してください。

サーバーの起動時にデフォルトのストレージエンジンを無効にする場合は、永続テーブルと TEMPORARY テーブルの両方のデフォルトエンジンを別のエンジンに設定する必要があります。そうしないと、サーバーは起動しません。

- `default_table_encryption`

コマンド行形式	<code>--default-table-encryption[={OFF ON}]</code>
導入	8.0.16
システム変数	<code>default_table_encryption</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Boolean
デフォルト値	OFF

ENCRYPTION 句を指定せずにスキーマおよび一般テーブルスペースを作成するときに適用されるデフォルトの暗号化設定を定義します。

`default_table_encryption` 変数は、ユーザーが作成したスキーマおよび一般テーブルスペースにのみ適用できます。mysql システムテーブルスペースの暗号化は制御されません。

`default_table_encryption` のランタイム値を設定するには、`SYSTEM_VARIABLES_ADMIN` および `TABLE_ENCRYPTION_ADMIN` 権限、または非推奨の `SUPER` 権限が必要です。

`default_table_encryption` は、`SET PERSIST` および `SET PERSIST_ONLY` 構文をサポートしています。セクション 5.1.9.3「永続化されるシステム変数」を参照してください。

詳細は、スキーマおよび一般テーブルスペースの暗号化デフォルトの定義を参照してください。

- `default_tmp_storage_engine`

コマンド行形式	<code>--default-tmp-storage-engine=name</code>
システム変数	<code>default_tmp_storage_engine</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	列挙
デフォルト値	InnoDB

TEMPORARY テーブルのデフォルトストレージエンジン (`CREATE TEMPORARY TABLE` で作成されたもの)。永続的なテーブルについてのストレージエンジンを設定するには、`default_storage_engine` システム変数を設定します。可能な値については、その変数の説明も参照してください。

サーバーの起動時にデフォルトのストレージエンジンを無効にする場合は、永続テーブルと TEMPORARY テーブルの両方のデフォルトエンジンを別のエンジンに設定する必要があります。そうしないと、サーバーは起動しません。

- [default_week_format](#)

コマンド行形式	<code>--default-week-format=#</code>
システム変数	<code>default_week_format</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	7

`WEEK()` 関数について使用するデフォルトモード値。 [セクション12.7「日付および時間関数」](#) を参照してください。

- [delay_key_write](#)

コマンド行形式	<code>--delay-key-write[={OFF ON ALL}]</code>
システム変数	<code>delay_key_write</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	ON
有効な値	ON OFF ALL

この変数は、遅延鍵書き込みの使用方法を指定します。これは `MyISAM` テーブルにのみ適用されます。鍵の書き込みが遅延すると、書き込み間で鍵バッファがフラッシュされません。 [セクション16.2.1「MyISAM 起動オプション」](#) も参照してください。

この変数には、`CREATE TABLE` ステートメントで使用できる `DELAY_KEY_WRITE` テーブルオプションの処理に影響する次のいずれかの値を指定できます。

オプション	説明
OFF	<code>DELAY_KEY_WRITE</code> は無視されます。
ON	MySQL は <code>CREATE TABLE</code> ステートメントに指定される <code>DELAY_KEY_WRITE</code> オプションを優先します。これはデフォルト値です。

オプション	説明
ALL	新しくオープンしたすべてのテーブルは、 DELAY_KEY_WRITE オプションを有効にして作成された場合と同様に処理されます。

注記

この変数を [ALL](#) に設定した場合、[MyISAM](#) テーブルの使用中に、このテーブルを別のプログラム内 (別の MySQL Server または [myisamchk](#) など) から使用しないでください。これを行うと、インデックスが破損します。

テーブルの [DELAY_KEY_WRITE](#) を有効にした場合、インデックス更新のたびにそのテーブルのキーバッファがフラッシュされるのではなく、テーブルが閉じたときだけフラッシュされます。これにより、キーの書き込みが大幅に高速化されますが、この機能を使用する場合は、[myisam_recover_options](#) システム変数を設定してサーバーを起動することで、すべての [MyISAM](#) テーブルの自動チェックを追加する必要があります ([myisam_recover_options='BACKUP,FORCE'](#) など)。 [セクション5.1.8「サーバーシステム変数」](#) および [セクション16.2.1「MyISAM 起動オプション」](#) を参照してください。

`--skip-new` を使用して `mysqld` を起動すると、`delay_key_write` は `OFF` に設定されます。

警告

`--external-locking` で外部ロックを有効にした場合、キーの遅延書き込みを使用するテーブルについてのインデックス破損に対する保護はありません。

• [delayed_insert_limit](#)

コマンド行形式	<code>--delayed-insert-limit=#</code>
非推奨	はい
システム変数	delayed_insert_limit
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	100
最小値	1
最大値 (64 ビットプラットフォーム)	18446744073709551615
最大値 (32 ビットプラットフォーム)	4294967295

このシステム変数は非推奨 ([DELAYED](#) 挿入はサポートされていないため) であり、将来のリリースで削除される予定です。

• [delayed_insert_timeout](#)

コマンド行形式	<code>--delayed-insert-timeout=#</code>
非推奨	はい
システム変数	delayed_insert_timeout
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer

デフォルト値	300
--------	-----

このシステム変数は非推奨 (`DELAYED` 挿入はサポートされていないため) であり、将来のリリースで削除される予定です。

- `delayed_queue_size`

コマンド行形式	<code>--delayed-queue-size=#</code>
非推奨	はい
システム変数	<code>delayed_queue_size</code>
スコープ	グローバル
動的	はい
<code>SET_VAR</code> ヒントの適用	いいえ
型	Integer
デフォルト値	1000
最小値	1
最大値 (64 ビットプラットフォーム)	18446744073709551615
最大値 (32 ビットプラットフォーム)	4294967295

このシステム変数は非推奨 (`DELAYED` 挿入はサポートされていないため) であり、将来のリリースで削除される予定です。

- `disabled_storage_engines`

コマンド行形式	<code>--disabled-storage-engines=engine[,engine]...</code>
システム変数	<code>disabled_storage_engines</code>
スコープ	グローバル
動的	いいえ
<code>SET_VAR</code> ヒントの適用	いいえ
型	文字列
デフォルト値	<code>empty string</code>

この変数は、テーブルまたはテーブルスペースの作成に使用できないストレージエンジンを示します。たとえば、新しい `MyISAM` または `FEDERATED` テーブルが作成されないようにするには、サーバーオプションファイルで次の行を使用してサーバーを起動します:

```
[mysqld]
disabled_storage_engines="MyISAM,FEDERATED"
```

デフォルトでは、`disabled_storage_engines` は空ですが (エンジンは無効化されていません)、1 つ以上のエンジンのカンマ区切りリストに設定できます (大文字と小文字は区別されません)。値に指定されたエンジンは、`CREATE TABLE` または `CREATE TABLESPACE` を使用したテーブルまたはテーブルスペースの作成には使用できません。また、`ALTER TABLE ... ENGINE` または `ALTER TABLESPACE ... ENGINE` とともに使用して、既存のテーブルまたはテーブルスペースの記憶域エンジンを変更することもできません。これを試行すると、`ER_DISABLED_STORAGE_ENGINE` エラーが発生します。

`disabled_storage_engines` では、`CREATE INDEX`、`TRUNCATE TABLE`、`ANALYZE TABLE`、`DROP TABLE` や `DROP TABLESPACE` などの既存のテーブルに対する他の DDL ステートメントは制限されません。これにより、無効なエンジンを使用する既存のテーブルまたはテーブルスペースを、`ALTER TABLE ... ENGINE permitted_engine` などの方法で許可されたエンジンに移行できるように、スムーズな遷移が可能になります。

`default_storage_engine` または `default_tmp_storage_engine` システム変数を無効になっているストレージエンジンに設定できます。これにより、アプリケーションは誤って動作または失敗する可能性があります。無効化され

たエンジンを使用するアプリケーションを識別して変更できるように開発環境で有用な手法である可能性があります。

これらのオプションのいずれかを使用してサーバーを起動した場合、`disabled_storage_engines` は無効になり、効果はありません: `--initialize`, `--initialize-insecure`, `--skip-grant-tables`。

注記

`disabled_storage_engines` を設定すると、`mysql_upgrade` で問題が発生する可能性があります。詳細は、[セクション4.4.5「mysql_upgrade — MySQL テーブルのチェックとアップグレード」](#)を参照してください。

- [disconnect_on_expired_password](#)

コマンド行形式	<code>--disconnect-on-expired-password[={OFF ON}]</code>
システム変数	<code>disconnect_on_expired_password</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

この変数は、期限切れのパスワードを持つクライアントをサーバーが処理する方法を制御します。

- クライアントが期限切れのパスワードを処理できることを示している場合、`disconnect_on_expired_password` の値は関係ありません。サーバーはクライアントが接続することを許可しますが、クライアントをサンドボックスモードに設定します。
- クライアントが期限切れのパスワードを処理できないことを示していない場合、サーバーは `disconnect_on_expired_password` の値に従ってクライアントを処理します:
 - `disconnect_on_expired_password`: が有効な場合、サーバーはクライアントを切断します。
 - `disconnect_on_expired_password`: が無効な場合、サーバーはクライアントの接続を許可しますが、クライアントをサンドボックスモードに設定します。

期限切れパスワードに関するクライアントとサーバーの対話の設定の詳細については、[セクション6.2.16「期限切れパスワードのサーバー処理」](#)を参照してください。

- [div_precision_increment](#)

コマンド行形式	<code>--div-precision-increment=#</code>
システム変数	<code>div_precision_increment</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Integer
デフォルト値	4
最小値	0
最大値	30

この変数は、/ 演算子で実行される除算の結果のスケールを増やす桁数を示します。デフォルト値は 4 です。最小値および最大値は、それぞれ 0 および 30 です。次の例は、デフォルト値を増やした効果を説明したものです。

```
mysql> SELECT 1/7;
```

```
+-----+
```

```

| 1/7 |
+-----+
| 0.1429 |
+-----+
mysql> SET div_precision_increment = 12;
mysql> SELECT 1/7;
+-----+
| 1/7 |
+-----+
| 0.142857142857 |
+-----+

```

- [dragnet.log_error_filter_rules](#)

コマンド行形式	--dragnet.log-error-filter-rules=value
システム変数	dragnet.log_error_filter_rules
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	IF prio>=INFORMATION THEN drop. IF EXISTS source_line THEN unset source_line.

[log_filter_dragnet](#) エラーログフィルタコンポーネントの操作を制御するフィルタルール。 [log_filter_dragnet](#) がインストールされていない場合、 [dragnet.log_error_filter_rules](#) は使用できません。 [log_filter_dragnet](#) がインストールされているが有効になっていない場合、 [dragnet.log_error_filter_rules](#) への変更は無効です。

デフォルト値の効果は、 [log_error_verbosity=2](#) の設定を使用して [log_sink_internal](#) フィルタによって実行されるフィルタリングに似ています。

MySQL 8.0.12 の時点では、 [dragnet.Status](#) ステータス変数を参照して、 [dragnet.log_error_filter_rules](#) への最新の割当ての結果を判断できます。

MySQL 8.0.12 より前は、実行時に [dragnet.log_error_filter_rules](#) への割当てが成功すると、新しい値を確認するノートが生成されます:

```

mysql> SET GLOBAL dragnet.log_error_filter_rules = 'IF prio <> 0 THEN unset prio.';
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Note
Code: 4569
Message: filter configuration accepted:
        SET @@GLOBAL.dragnet.log_error_filter_rules=
        'IF prio!=ERROR THEN unset prio.';

```

[SHOW WARNINGS](#) によって表示される値は、ルールセットが正常に解析されて内部形式にコンパイルされた後の「decompiled」正規表現を示します。意味上、この正規の形式は [dragnet.log_error_filter_rules](#) に割り当てられた値と同じですが、前述の例に示すように、割り当てられた値と正規の値にはいくつかの違いがあります:

- <> 演算子が != に変更されます。
- 数値の優先度 0 は、対応する優先度記号 [ERROR](#) に変更されます。
- オプションのスペースが削除されます。

詳細については、[セクション5.4.2.4「エラーログフィルタリングのタイプ」](#)、および[セクション5.5.3「エラーログコンポーネント」](#)を参照してください。

- [end_markers_in_json](#)

コマンド行形式	--end-markers-in-json[={OFF ON}]
---------	----------------------------------

システム変数	end_markers_in_json
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Boolean
デフォルト値	OFF

オプティマイザ JSON 出力がエンドマーカを追加するかどうか。 「MySQL 内部: [end_markers_in_json システム変数](#)」を参照してください。

- [eq_range_index_dive_limit](#)

コマンド行形式	<code>--eq-range-index-dive-limit=#</code>
システム変数	eq_range_index_dive_limit
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Integer
デフォルト値	200
最小値	0
最大値	4294967295

この変数は、オプティマイザが限定する行数を推定するときに、インデックスダイブの使用からインデックス統計の使用に切り換える場合の等価比較条件内の等価範囲の数を指定します。これは次に示す同等のいずれかの形式を持つ式の評価に適用され、このときオプティマイザは一意でないインデックスを使用して [col_name](#) 値を参照します。

```
col_name IN(val1, ..., valN)
col_name = val1 OR ... OR col_name = valN
```

どちらの場合も、式に N 個の等価範囲が含まれています。オプティマイザはインデックスダイブまたはインデックス統計を使用すると行の推定を実行できます。[eq_range_index_dive_limit](#) が 0 より大きい場合、[eq_range_index_dive_limit](#) 以上の等価範囲があれば、オプティマイザはインデックスダイブの代わりに既存のインデックス統計を使用します。したがって、 N 個までの等価範囲に対してインデックスダイブの使用を可能にするには、[eq_range_index_dive_limit](#) を $N + 1$ に設定します。インデックス統計の使用を無効にし、 N に関係なく常にインデックスダイブを使用するには、[eq_range_index_dive_limit](#) を 0 に設定します。

詳細については、[複数値比較の等価範囲の最適化](#)を参照してください。

最適な推定を行うためにテーブルインデックス統計を更新するには、[ANALYZE TABLE](#) を使用します。

- [error_count](#)

メッセージを生成した最後のステートメントから発生したエラーの数。この変数は読み取り専用です。[セクション 13.7.7.17 「SHOW ERRORS ステートメント」](#)を参照してください。

- [event_scheduler](#)

コマンド行形式	<code>--event-scheduler[=value]</code>
システム変数	event_scheduler
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙

デフォルト値	ON
有効な値	ON OFF DISABLED

この変数は、イベントスケジューラを有効または無効にし、起動または停止します。使用可能なステータス値は、ON、OFF および DISABLED で、デフォルトは OFF です。イベントスケジューラの有効化 OFF は、ステータスを DISABLED に設定する必要があるイベントスケジューラを無効にすることと同じではありません。この変数とそのイベントスケジューラ操作への影響の詳細は、[セクション25.4.2「イベントスケジューラの構成」](#)を参照してください

- [explicit_defaults_for_timestamp](#)

コマンド行形式	--explicit-defaults-for-timestamp[={OFF ON}]
非推奨	はい
システム変数	explicit_defaults_for_timestamp
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

このシステム変数は、サーバーがデフォルト値に対して特定の非標準動作を有効にするかどうか、および `TIMESTAMP` カラムで `NULL` 値の処理を有効にするかどうかを決定します。デフォルトでは、[explicit_defaults_for_timestamp](#) は有効になっており、非標準動作は無効になっています。[explicit_defaults_for_timestamp](#) を無効にすると、警告が表示されます。

MySQL 8.0.18 では、このシステム変数のセッション値の設定は制限付き操作ではなくなりました。

[explicit_defaults_for_timestamp](#) が無効になっている場合、サーバーは非標準動作を有効にし、次のように `TIMESTAMP` カラムを処理します:

- `NULL` 属性で明示的に宣言されていない `TIMESTAMP` カラムは、`NOT NULL` 属性で自動的に宣言されます。このようなカラムに `NULL` の値を割り当てることができ、カラムを現在のタイムスタンプに設定します。例外: MySQL 8.0.22 では、`TIMESTAMP NOT NULL` として宣言された生成されたカラムに `NULL` を挿入しようとすると、エラーで拒否されます。
- `NULL` 属性、明示的な `DEFAULT` または `ON UPDATE` 属性で明示的に宣言されていない場合、テーブルの最初の `TIMESTAMP` カラムは `DEFAULT CURRENT_TIMESTAMP` および `ON UPDATE CURRENT_TIMESTAMP` 属性で自動的に宣言されます。
- 最初のカラムに続く `TIMESTAMP` カラムは、`NULL` 属性または明示的な `DEFAULT` 属性で明示的に宣言されていない場合、`DEFAULT '0000-00-00 00:00:00'` (「ゼロ」タイムスタンプ) として自動的に宣言されます。そのようなカラムに対して明示的な値を指定しない挿入された行については、カラムに `'0000-00-00 00:00:00'` が自動的に割り当てられて、警告は発生しません。

厳密な SQL モードと `NO_ZERO_DATE` SQL モードのどちらが有効になっているかによって、デフォルト値の `'0000-00-00 00:00:00'` が無効になる場合があります。 `TRADITIONAL` SQL モードには、厳密モードおよび

`NO_ZERO_DATE` が含まれることに注意してください。 [セクション5.1.11「サーバー SQL モード」](#) を参照してください。

前述の非標準の動作は非推奨です。将来の MySQL リリースで削除される予定です。

`explicit_defaults_for_timestamp` が有効になっている場合、サーバーは非標準動作を無効にし、次のように `TIMESTAMP` カラムを処理します:

- `TIMESTAMP` カラムに `NULL` の値を割り当てて現在のタイムスタンプに設定することはできません。現在のタイムスタンプを割り当てるには、カラムを `CURRENT_TIMESTAMP` または `NOW()` などのシノニムに設定します。
- `NOT NULL` 属性で明示的に宣言されていない `TIMESTAMP` カラムは、`NULL` 属性で自動的に宣言され、`NULL` 値を許可します。このようなカラムに `NULL` の値を割り当てると、現在のタイムスタンプではなく `NULL` に設定されます。
- `NOT NULL` 属性で宣言された `TIMESTAMP` カラムでは、`NULL` 値は許可されません。このようなカラムに `NULL` を指定する挿入の場合、単一行挿入のエラー、厳密な SQL モードが有効になっている場合、または厳密な SQL モードが無効になっている複数行挿入の場合、`'0000-00-00 00:00:00'` が挿入されます。いずれの場合も、カラムに `NULL` の値が割り当てられず、現在のタイムスタンプに設定されます。
- 明示的な `DEFAULT` 属性を使用せずに `NOT NULL` 属性で明示的に宣言された `TIMESTAMP` カラムは、デフォルト値を持たないものとして扱われます。そのようなカラムについて明示的な値を指定しない挿入された行の場合、結果は SQL モードによって異なります。厳密な SQL モードが有効である場合、エラーが発生します。厳密な SQL モードが無効になっていない場合、カラムは暗黙的なデフォルトの `'0000-00-00 00:00:00'` で宣言され、警告が発生します。これは、MySQL が `DATETIME` などのほかの時間型を処理する方法に類似しています。
- `DEFAULT CURRENT_TIMESTAMP` または `ON UPDATE CURRENT_TIMESTAMP` 属性で `TIMESTAMP` カラムが自動的に宣言されることはありません。これらの属性は、明示的に指定する必要があります。
- テーブルの最初の `TIMESTAMP` カラムは、最初のカラムに続く `TIMESTAMP` カラムとは異なる方法で処理されることはありません。

サーバーの起動時に `explicit_defaults_for_timestamp` が無効になっている場合、次の警告がエラーログに表示されます:

```
[Warning] TIMESTAMP with implicit DEFAULT value is deprecated.
Please use --explicit_defaults_for_timestamp server option (see
documentation for more details).
```

警告に示されているように、非推奨の非標準動作を無効にするには、サーバーの起動時に `explicit_defaults_for_timestamp` システム変数を有効にします。

注記

`explicit_defaults_for_timestamp` 自体は非推奨です。これは、将来の MySQL リリースで削除される非推奨の `TIMESTAMP` 動作の制御を許可することのみを目的としているためです。これらの動作の削除が発生した場合は、`explicit_defaults_for_timestamp` も削除されることを想定してください。

追加情報については [セクション11.2.5「TIMESTAMP および DATETIME の自動初期化および更新機能」](#) を参照してください。

- `external_user`

システム変数	<code>external_user</code>
スコープ	セッション
動的	いいえ
<code>SET_VAR</code> ヒントの適用	いいえ

型	文字列
---	-----

クライアントを認証するために使用されるプラグインによって設定された、認証プロセス中に使用される外部ユーザー名。ネイティブ (組み込み型) の MySQL 認証や、プラグインで値が設定されない場合、この変数は `NULL` です。 [セクション6.2.18「プロキシユーザー」](#) を参照してください。

- フラッシュ

コマンド行形式	<code>--flush[={OFF ON}]</code>
システム変数	<code>flush</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

ON の場合、各 SQL ステートメントのあとでサーバーはすべての変更をデスクにフラッシュ (同期) します。通常、MySQL では各 SQL ステートメントの終了後にのみすべての変更内容をディスクに書き込み、ディスクへの同期はオペレーティングシステムが処理します。 [セクションB.3.3.3「MySQL が繰り返しクラッシュする場合の対処方法」](#) を参照してください。 `--flush` オプションで `mysqld` を起動した場合、この変数は ON に設定されます。

注記

`flush` が有効な場合、`flush_time` の値は関係なく、`flush_time` を変更してもフラッシュ動作には影響しません。

- `flush_time`

コマンド行形式	<code>--flush-time=#</code>
システム変数	<code>flush_time</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0

これがゼロ以外の値に設定されると、すべてのテーブルは `flush_time` 秒ごとに閉じられて、リソースが解放され、フラッシュされていないデータがディスクへ同期されます。このオプションは、リソースが非常に限定されたシステムでのみ使用することを推奨します。

注記

`flush` が有効な場合、`flush_time` の値は関係なく、`flush_time` を変更してもフラッシュ動作には影響しません。

- `foreign_key_checks`

システム変数	<code>foreign_key_checks</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Boolean

デフォルト値	ON
--------	----

1 (デフォルト) に設定すると、外部キー制約がチェックされます。0 に設定すると、外部キー制約は無視されますが、いくつかの例外があります。削除されたテーブルを再作成する場合、テーブル定義がテーブルを参照する外部キー制約に準拠していないと、エラーが返されます。同様に、外部キー定義の形式が正しくない場合、**ALTER TABLE** 操作はエラーを返します。詳細は、[セクション 13.1.20.5 「FOREIGN KEY の制約」](#) を参照してください。

この変数を設定すると、**NDB** テーブルには **InnoDB** テーブルと同じ効果があります。一般的に、通常の操作中はこの設定を有効にしたままにすることで、[参照整合性](#) を強制します。外部キーチェックを無効にすると、親/子関係に必要な順序とは異なる順序で **InnoDB** テーブルをリロードする場合に役立ちます。[セクション 13.1.20.5 「FOREIGN KEY の制約」](#) を参照してください。

foreign_key_checks を 0 に設定すると、データ定義ステートメントにも影響します。**DROP SCHEMA** は、スキーマの外部のテーブルによって参照されている外部キーを持つテーブルをスキーマが含む場合であってもスキーマをドロップし、**DROP TABLE** は、別のテーブルによって参照されている外部キーを持つテーブルをドロップします。

注記

foreign_key_checks を 1 に設定すると、既存のテーブルデータのスキャンがトリガーされません。したがって、**foreign_key_checks = 0** の間にテーブルに追加された行の一貫性は検証されません。

外部キー制約に必要なインデックスの削除は、**foreign_key_checks=0** でも許可されています。インデックスを削除する前に、外部キー制約を削除する必要があります。

• [ft_boolean_syntax](#)

コマンド行形式	--ft-boolean-syntax=name
システム変数	ft_boolean_syntax
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	+ -><()~*:"'&

IN BOOLEAN MODE を使用して実行されるブール全文検索によってサポートされる演算子のリスト。[セクション 12.10.2 「ブール全文検索」](#) を参照してください。

デフォルトの変数値は '+ -><()~*:"'&|' です。値を変更するルールは次のようになります。

- 演算子の機能は、文字列内の位置によって決定されます。
- 置換する値は 14 文字である必要があります。
- 各文字は、英数字以外の ASCII 文字である必要があります。
- 1 番目または 2 番目の文字がスペースである必要があります。
- 位置 11 および 12 にある句を引用する演算子を除き、重複は許可されません。これらの 2 つの文字は同じである必要はありませんが、同じであってもよいのはこれら 2 つだけです。
- ポジション 10、13 および 14 (デフォルトでは:、& および|に設定されます) は、将来の拡張用に予約されています。

• [ft_max_word_len](#)

コマンド行形式	--ft-max-word-len=#
システム変数	ft_max_word_len

スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
最小値	10

MyISAM FULLTEXT インデックスに含めることができる最大の単語の長さ。

注記

この変数を変更したあと、MyISAM テーブルの FULLTEXT インデックスを再構築する必要があります。REPAIR TABLE tbl_name QUICK を使用します。

• ft_min_word_len

コマンド行形式	--ft-min-word-len=#
システム変数	ft_min_word_len
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	4
最小値	1

MyISAM FULLTEXT インデックスに含めることができる最小の単語の長さ。

注記

この変数を変更したあと、MyISAM テーブルの FULLTEXT インデックスを再構築する必要があります。REPAIR TABLE tbl_name QUICK を使用します。

• ft_query_expansion_limit

コマンド行形式	--ft-query-expansion-limit=#
システム変数	ft_query_expansion_limit
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	20
最小値	0
最大値	1000

WITH QUERY EXPANSION を使用して実行する全文検索で使用する最上位の一致の数。

• ft_stopword_file

コマンド行形式	--ft-stopword-file=file_name
システム変数	ft_stopword_file
スコープ	グローバル
動的	いいえ

SET_VAR ヒントの適用	いいえ
型	ファイル名

MyISAM テーブルの全文検索について、ストップワードのリストの読み取り元ファイル。サーバーは、別のディレクトリを指定する絶対パス名が指定されないかぎり、データディレクトリ内のファイルを検索します。ファイル内のすべての単語が使用され、コメントは受け付けられません。デフォルトでは、ストップワードの組み込みリストが使用されます (`storage/myisam/ft_static.c` ファイルに定義されています)。この変数を空の文字列 ("") に設定すると、ストップワードフィルタリングが無効になります。 [セクション12.10.4「全文ストップワード」](#) も参照してください。

注記

この変数またはストップワードファイルの内容を変更したあと、MyISAM テーブルの FULLTEXT インデックスを再構築する必要があります。 `REPAIR TABLE tbl_name QUICK` を使用します。

• [general_log](#)

コマンド行形式	<code>--general-log[={OFF ON}]</code>
システム変数	general_log
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

一般クエリーログを有効にするかどうか。値が 0 (または OFF) の場合はログを無効にし、1 (または ON) の場合はログを有効にします。ログ出力先は `log_output` システム変数によって制御され、この値を NONE にした場合はログが有効になっていてもログエントリは書き込まれません。

• [general_log_file](#)

コマンド行形式	<code>--general-log-file=file_name</code>
システム変数	general_log_file
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	<code>host_name.log</code>

一般クエリーログファイルの名前。デフォルト値は `host_name.log` ですが、初期値は `--general_log_file` オプションを使用すると変更できます。

• [generated_random_password_length](#)

コマンド行形式	<code>--generated-random-password-length=#</code>
導入	8.0.18
システム変数	generated_random_password_length
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer

デフォルト値	20
最小値	5
最大値	255

`CREATE USER`、`ALTER USER` および `SET PASSWORD` ステートメントに対して生成されるランダムパスワードで許可される最大文字数。詳細は、[ランダムパスワード生成](#)を参照してください。

- [group_concat_max_len](#)

コマンド行形式	<code>--group-concat-max-len=#</code>
システム変数	group_concat_max_len
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Integer
デフォルト値	1024
最小値	4
最大値 (64 ビットプラットフォーム)	18446744073709551615
最大値 (32 ビットプラットフォーム)	4294967295

`GROUP_CONCAT()` 関数について許可されるバイト単位の最大の結果の長さ。デフォルトは 1024 です。

- [have_compress](#)

`zlib` 圧縮ライブラリがサーバーで利用できる場合は `YES`、そうでない場合は `NO`。利用できない場合、`COMPRESS()` および `UNCOMPRESS()` 関数は使用できません。

- [have_dynamic_loading](#)

`mysqld` がプラグインの動的ロードをサポートする場合は `YES`、そうでない場合は `NO`。値が `NO` の場合、`--plugin-load` などのオプションを使用してサーバー起動時にプラグインをロードしたり、`INSTALL PLUGIN` ステートメントを使用して実行時にプラグインをロードしたりすることはできません。

- [have_geometry](#)

サーバーが空間データ型をサポートする場合は `YES`、そうでない場合は `NO`。

- [have_openssl](#)

この変数は、[have_ssl](#) のシノニムです。

- [have_profiling](#)

ステートメントプロファイリング機能が存在する場合は `YES`、そうでない場合は `NO`。存在する場合、この機能を有効または無効にするかが `profiling` システム変数によって制御されます。[セクション13.7.7.31「SHOW PROFILES ステートメント」](#)を参照してください。

この変数は非推奨であり、将来の MySQL リリースで削除される予定です。

- [have_query_cache](#)

クエリーキャッシュは MySQL 8.0.3 で削除されました。[have_query_cache](#) は非推奨であり、常に `NO` の値を持ち、将来の MySQL リリースで削除される予定です。

- [have_rtree_keys](#)

`RTREE` インデックスを利用できる場合は `YES`、そうでない場合は `NO`。(これらは `MyISAM` テーブル内の空間インデックスで使用されます。)

- [have_ssl](#)

システム変数	have_ssl
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列
有効な値	YES (SSL サポートが使用可能) DISABLED (SSL サポートはサーバーにコンパイルされましたが、有効にするために必要なオプションでサーバーが起動されませんでした)

YES (mysqld が SSL 接続をサポートしている場合)、DISABLED(サーバーが SSL サポート付きでコンパイルされているが、適切な connection-encryption オプションで起動されていない場合)。詳細は、[セクション2.9.6「SSL ライブラリサポートの構成」](#)を参照してください。

- [have_statement_timeout](#)

システム変数	have_statement_timeout
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean

ステートメントの実行タイムアウト機能が使用可能かどうか ([ステートメント実行時オプティマイザヒント](#) を参照)。この機能で使用するバックグラウンドスレッドを初期化できなかった場合、値は NO になります。

- [have_symlink](#)

シンボリックリンクサポートを有効化している場合は YES、そうでない場合は NO。これは、[DATA DIRECTORY](#) および [INDEX DIRECTORY](#) テーブルオプションをサポートするために Unix で必要です。 `--skip-symbolic-links` オプションを指定してサーバーが開始された場合、この値は [DISABLED](#) です。

この変数は Windows では意味がありません。

注記

シンボリックリンクのサポートは、それを制御する `--symbolic-links` オプションとともに非推奨になりました。これらは MySQL の将来のバージョンで削除される予定です。また、このオプションはデフォルトで無効になっています。関連する [have_symlink](#) システム変数も非推奨であり、将来のバージョンの MySQL で削除される予定です。

- [histogram_generation_max_mem_size](#)

コマンド行形式	<code>--histogram-generation-max-mem-size=#</code>
システム変数	histogram_generation_max_mem_size
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	20000000
最小値	1000000
最大値 (64 ビットプラットフォーム)	18446744073709551615

最大値 (32 ビットプラットフォーム)	4294967295
----------------------	------------

ヒストグラム統計の生成に使用可能なメモリの最大量。 [セクション8.9.6「オプティマイザ統計」](#) および [セクション13.7.3.1「ANALYZE TABLE ステートメント」](#) を参照してください。

このシステム変数のセッション値の設定は制限された操作です。セッションユーザーには、制限付きセッション変数を設定するのに十分な権限が必要です。 [セクション5.1.9.1「システム変数権限」](#) を参照してください。

- [host_cache_size](#)

コマンド行形式	<code>--host-cache-size=#</code>
システム変数	host_cache_size
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動サイズ設定を示します。このリテラル値を割り当てないでください)
最小値	0
最大値	65536

MySQL サーバーは、クライアントホスト名および IP アドレス情報を含むインメモリーホストキャッシュを保持し、ドメインネームシステム (DNS) のルックアップを回避するために使用されます。 [セクション5.1.12.3「DNS ルックアップとホストキャッシュ」](#) を参照してください。

[host_cache_size](#) 変数は、ホストキャッシュのサイズと、キャッシュの内容を公開するパフォーマンススキーマ [host_cache](#) テーブルのサイズを制御します。 [host_cache_size](#) を設定すると、次の効果があります:

- サイズを 0 に設定すると、ホストキャッシュが無効になります。キャッシュを無効にすると、サーバーはクライアントが接続するたびに DNS ルックアップを実行します。
- 実行時にサイズを変更すると、暗黙的なホストキャッシュフラッシュ操作が実行され、ホストキャッシュのクリア、[host_cache](#) テーブルの切捨ておよびブロックされたホストのブロック解除が行われます。

デフォルト値は 128 に自動サイズ設定され、500 までの [max_connections](#) の値には 1 が加算され、[max_connections](#) 値の 500 を超えるすべての増分には 1 が加算され、制限は 2000 に制限されます。

`--skip-host-cache` オプションの使用は、[host_cache_size](#) システム変数を 0 に設定するのと似ていますが、[host_cache_size](#) は、サーバーの起動時だけでなく、実行時にホストキャッシュのサイズ変更、有効化、および無効化にも使用できるため、より柔軟です。 `--skip-host-cache` を使用してサーバーを起動しても、[host_cache_size](#) の値に対する実行時の変更は妨げられませんが、このような変更は効果がなく、[host_cache_size](#) が 0 より大きい値に設定されていてもキャッシュは再度有効になりません。

- [hostname](#)

システム変数	hostname
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

サーバーは起動時に、この変数をサーバーホスト名に設定します。RFC 1034 によると、MySQL 8.0.17 での最大長は 255 文字で、その前は 60 文字です。

- identity

この変数は `last_insert_id` 変数のシノニムです。これはほかのデータベースシステムとの互換性のために存在します。この値は `SELECT @@identity` で読み取ることができ、`SET identity` で設定できます。

- init_connect

コマンド行形式	<code>--init-connect=name</code>
システム変数	<code>init_connect</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列

接続する各クライアントに対してサーバーによって実行される文字列。文字列は 1 つ以上の SQL ステートメントで構成され、セミコロン文字で区切られます。

`CONNECTION_ADMIN` 権限 (または非推奨の `SUPER` 権限) を持つユーザーの場合、`init_connect` のコンテンツは実行されません。これを行うのは、`init_connect` の値が誤っていても、すべてのクライアントの接続を妨げないようにするためです。たとえば、値に含まれているステートメントが構文エラーを含むため、クライアント接続が失敗することがあります。`CONNECTION_ADMIN` または `SUPER` 権限を持つユーザーに対して `init_connect` を実行しないと、接続をオープンして `init_connect` 値を修正できます。

パスワードが期限切れのクライアントユーザーの場合、`init_connect` の実行はスキップされます。これは、このようなユーザーは任意のステートメントを実行できないため、`init_connect` の実行が失敗し、クライアントが接続できなくなるためです。`init_connect` の実行をスキップすると、ユーザーは接続してパスワードを変更できます。

サーバーは、`init_connect` の値のステートメントによって生成されたすべての結果セットを破棄します。

- information_schema_stats_expiry

コマンド行形式	<code>--information-schema-stats-expiry=#</code>
システム変数	<code>information_schema_stats_expiry</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	86400
最小値	0
最大値	31536000

一部の `INFORMATION_SCHEMA` テーブルには、テーブル統計を提供するカラムが含まれています:

```
STATISTICS.CARDINALITY
TABLES.AUTO_INCREMENT
TABLES.AVG_ROW_LENGTH
TABLES.CHECKSUM
TABLES.CHECK_TIME
TABLES.CREATE_TIME
TABLES.DATA_FREE
TABLES.DATA_LENGTH
TABLES.INDEX_LENGTH
TABLES.MAX_DATA_LENGTH
TABLES.TABLE_ROWS
```

TABLES.UPDATE_TIME

これらのカラムは、動的テーブルメタデータ、つまりテーブルの内容の変更に応じて変更される情報を表します。

デフォルトでは、MySQL は、カラムのクエリー時に `mysql.index_stats` および `mysql.table_stats` ディクショナリテーブルからこれらのカラムのキャッシュされた値を取得します。これは、ストレージエンジンから統計を直接取得するよりも効率的です。キャッシュされた統計が使用できないか、期限切れになっている場合、MySQL はストレージエンジンから最新の統計を取得し、`mysql.index_stats` および `mysql.table_stats` ディクショナリテーブルにキャッシュします。後続のクエリーでは、キャッシュされた統計が期限切れになるまで、キャッシュされた統計が取得されます。

`information_schema_stats_expiry` セッション変数は、キャッシュされた統計が期限切れになるまでの期間を定義します。デフォルトは 86400 秒 (24 時間) ですが、期間は 1 年まで延長できます。

特定のテーブルのキャッシュされた値をいつでも更新するには、`ANALYZE TABLE` を使用します。

常に最新の統計をストレージエンジンから直接取得し、キャッシュされた値をバイパスするには、`information_schema_stats_expiry` を 0 に設定します。

次の場合、統計カラムのクエリーでは、`mysql.index_stats` および `mysql.table_stats` ディクショナリテーブルの統計は格納または更新されません:

- キャッシュされた統計が失効していない場合。
- `information_schema_stats_expiry` が 0 に設定されている場合。
- サーバーが `read_only`, `super_read_only`, `transaction_read_only` または `innodb_read_only` モードで起動されたとき。
- クエリーでパフォーマンススキーマデータもフェッチされる場合。

`information_schema_stats_expiry` はセッション変数であり、各クライアントセッションで独自の有効期限値を定義できます。ストレージエンジンから取得され、あるセッションによってキャッシュされた統計は、ほかのセッションで使用できます。

関連情報については、[セクション8.2.3「INFORMATION_SCHEMA クエリーの最適化」](#)を参照してください。

- `init_file`

コマンド行形式	<code>--init-file=file_name</code>
システム変数	<code>init_file</code>
スコープ	グローバル
動的	いいえ
<code>SET_VAR</code> ヒントの適用	いいえ
型	ファイル名

指定した場合、この変数は、起動プロセス中に読み取られて実行される SQL ステートメントを含むファイルに名前を付けます。MySQL 8.0.18 より前では、各ステートメントは単一行に記述する必要があり、コメントを含めることはできません。MySQL 8.0.18 の時点では、ファイル内のステートメントの許容形式は、次の構成をサポートするように拡張されています:

- `delimiter ;`:ステートメントデリミタを;文字に設定します。
- `delimiter $$`:ステートメントデリミタを \$\$ 文字シーケンスに設定します。
- 現在のデリミタで区切られた、同じ行の複数のステートメント。
- Multiple-line statements.
- #文字から行の末尾までのコメント。

- -- 順序から行の末尾までのコメント。
- /* 順序から次の */順序への C 形式のコメント (複数行にわたるコメントを含む)。
- 一重引用符 (') または二重引用符 (") 文字で囲まれた複数行の文字列リテラル。

サーバーが `--initialize` または `--initialize-insecure` オプションを使用して起動された場合、それはブートスタップモードで動作し、ファイルで許可されるステートメントを制限する一部の機能は使用できません。これには、アカウント管理 (`CREATE USER`、`GRANT` など)、レプリケーションおよびグローバルトランザクション識別子に関連するステートメントが含まれます。セクション17.1.3「グローバルトランザクション識別子を使用したレプリケーション」を参照してください。

MySQL 8.0.17 では、サーバーの起動時に作成されたスレッドは、データディクショナリの作成、アップグレード手順の実行、システムテーブルの作成などのタスクに使用されます。安定した予測可能な環境を確保するために、これらのスレッドは、`sql_mode`、`character_set_server`、`collation_server`、`completion_type`、`explicit_defaults_for_timestamp` や `default_table_encryption` などの一部のシステム変数のサーバー組み込みデフォルトを使用して実行されます。

これらのスレッドは、サーバーの起動時に `init_file` で指定されたファイル内のステートメントを実行するためにも使用されるため、このようなステートメントは、これらのシステム変数のサーバーに組み込まれたデフォルト値で実行されます。

- [innodb_xxx](#)

InnoDB システム変数は、セクション15.14「InnoDB の起動オプションおよびシステム変数」にリストされています。これらの変数は、InnoDB テーブルの記憶域、メモリー使用および I/O パターンの多くの側面を制御し、InnoDB がデフォルトの記憶域エンジンであることが特に重要です。

- [insert_id](#)

`AUTO_INCREMENT` 値を挿入するときに、後に続く `INSERT` または `ALTER TABLE` ステートメントによって使用される値。これは主にバイナリログと一緒に使用されます。

- [interactive_timeout](#)

コマンド行形式	<code>--interactive-timeout=#</code>
システム変数	interactive_timeout
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	28800
最小値	1

サーバーが対話型の接続で、対話型の接続を閉じる前にアクティビティーを待機する秒数。対話型クライアントは、`mysql_real_connect()` で `CLIENT_INTERACTIVE` オプションを使用するクライアントと定義されます。[wait_timeout](#)も参照してください。

- [internal_tmp_disk_storage_engine](#)

コマンド行形式	<code>--internal-tmp-disk-storage-engine=#</code>
削除	8.0.16
システム変数	internal_tmp_disk_storage_engine
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ

型	列挙
デフォルト値	INNODB
有効な値	MYISAM INNODB

重要

MySQL 8.0.16 以降では、ディスク上の内部一時テーブルは常に InnoDB ストレージエンジンを使用します。MySQL 8.0.16 の時点で、この変数は削除されているため、サポートされなくなりました。

MySQL 8.0.16 より前は、この変数によって、ディスク上の内部一時テーブルに使用されるストレージエンジンが決まります ([オンディスク内部一時テーブルのストレージエンジン](#) を参照)。許可される値は、MYISAM および INNODB (デフォルト) です。

- [internal_tmp_mem_storage_engine](#)

コマンド行形式	<code>--internal-tmp-mem-storage-engine=#</code>
システム変数	internal_tmp_mem_storage_engine
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	列挙
デフォルト値	TempTable
有効な値	TempTable MEMORY

インメモリ内部一時テーブルのストレージエンジン ([セクション 8.4.4 「MySQL での内部一時テーブルの使用」](#) を参照)。許可される値は、TempTable (デフォルト) および MEMORY です。

[optimizer](#) は、インメモリ内部一時テーブル用に [internal_tmp_mem_storage_engine](#) によって定義されたストレージエンジンを使用します。

- [join_buffer_size](#)

コマンド行形式	<code>--join-buffer-size=#</code>
システム変数	join_buffer_size
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Integer
デフォルト値	262144
最小値	128
最大値 (Windows)	4294967295
最大値 (その他, 64 ビットプラットフォーム)	18446744073709547520
最大値 (その他, 32 ビットプラットフォーム)	4294967295

単純インデックススキャン、範囲インデックススキャン、およびインデックスを使用しないため完全テーブルスキャンを実行する結合について、使用されるバッファの最小サイズ。通常の場合、高速な結合を得るための最適な方法は、インデックスを追加することです。インデックスを追加できない場合、より高速な完全結合を得るために、[join_buffer_size](#) の値を大きくします。2つのテーブル間の完全結合 1つに対して 1つの結合バッファが割り

当てられます。インデックスが使用されない複数テーブル間の複雑な結合については、複数の結合バッファが必要になることもあります。

ブロックネストループアルゴリズムまたはバッチキーアクセスアルゴリズムを使用しないかぎり、一致する各行を保持するために必要な大きさを超えるバッファを設定することはできず、すべての結合で少なくとも最小サイズが割り当てられるため、この変数をグローバルに大きな値に設定する際は注意が必要です。グローバル設定を小さく保ち、大きな結合を実行しているセッションでのみセッション設定を大きな値に変更するか、`SET_VAR` オプティマイザヒントを使用してクエリーごとに設定を変更することをお勧めします(セクション8.9.3「[オプティマイザヒント](#)」を参照)。メモリーを使用するほとんどのクエリーによって必要なサイズよりもグローバルサイズを大きくすると、メモリー割り当て時間が原因でパフォーマンスが著しく低下することがあります。

Block Nested-Loop を使用すると、最初のテーブルのすべての行から必要なすべてのカラムが結合バッファに格納される時点まで、大きな結合バッファが有効になります。これはクエリーによって異なります。最適なサイズは、最初のテーブルのすべての行を保持するよりも小さくなる場合があります。

バッチ化されたキーアクセスを使用する場合、`join_buffer_size` の値によって、ストレージエンジンへのリクエストごとにキーのバッチの大きさが定義されます。バッファが大きいくほど、結合操作の右側のテーブルへの順次アクセスが多くなり、パフォーマンスが大幅に向上する可能性があります。

デフォルトは 256KB です。`join_buffer_size` で許可される最大の設定値は 4G バイト -1 です。64 ビットプラットフォームの場合は大きい値が許可されます (64 ビットの Windows の場合は例外で、大きい値は 4G バイト -1 に切り捨てられて警告が出ます)。

結合バッファリングについての追加情報は、[セクション8.2.1.7「Nested Loop 結合アルゴリズム」](#)を参照してください。バッチキーアクセスについては、[セクション8.2.1.12「Block Nested Loop 結合と Batched Key Access 結合」](#)を参照してください。

- [keep_files_on_create](#)

コマンド行形式	<code>--keep-files-on-create[={OFF ON}]</code>
システム変数	<code>keep_files_on_create</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

MyISAM テーブルが `DATA DIRECTORY` オプションなしで作成される場合、`.MYD` ファイルがデータベースディレクトリ内に作成されます。デフォルトでは、MyISAM が既存の `.MYD` ファイルを検出した場合、そのファイルを上書きします。 `INDEX DIRECTORY` オプションを指定せずに作成されたテーブルについて、`.MYI` ファイルに同じことが当てはまります。この動作を抑制するには、`keep_files_on_create` 変数を ON (1) に設定します。この場合、MyISAM は既存のファイルを上書きせず、かわりにエラーを返します。デフォルト値は OFF (0) です。

MyISAM テーブルが `DATA DIRECTORY` または `INDEX DIRECTORY` オプションを使用して作成され、既存の `.MYD` または `.MYI` ファイルが見つかった場合、MyISAM は常にエラーを返します。指定したディレクトリ内のファイルは上書きされません。

- [key_buffer_size](#)

コマンド行形式	<code>--key-buffer-size=#</code>
システム変数	<code>key_buffer_size</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	8388608

最小値	8
最大値 (64 ビットプラットフォーム)	OS_PER_PROCESS_LIMIT
最大値 (32 ビットプラットフォーム)	4294967295

MyISAM テーブルのインデックスブロックはバッファリングされ、すべてのスレッドで共有されます。[key_buffer_size](#) は、インデックスブロックに使用されるバッファのサイズです。キーバッファはキーキャッシュとしても知られています。

32 ビットプラットフォームでは、[key_buffer_size](#) に許可される最大の設定値は 4G バイト - 1 です。64 ビットプラットフォームでは、さらに大きい値が許可されます。実質的な最大サイズは、使用可能な物理 RAM や、オペレーティングシステムまたはハードウェアプラットフォームによって課されるプロセスごとの RAM 制限によって、もっと小さいことがあります。この変数の値は、リクエストされるメモリーの量を示します。サーバーは内部的に、この量までのできるだけ多くのメモリーを割り当てますが、実際の割り当てがもっと少なくなることもあります。

値を増やすことで、すべての読み取りおよび複数の書き込みのインデックス処理を改善できます。システムの主な機能が **MyISAM** ストレージエンジンを使用して MySQL を実行する場合、マシンの合計メモリーの 25% がこの変数の許容可能な値です。ただし、値を大きくしすぎると (マシンの合計メモリーの 50% 超)、システムのページングが始まってきわめて低速になることがあります。これは MySQL がデータ読み取りのためのファイルシステムキャッシュの実行をオペレーティングシステムに依存しているため、ファイルシステムキャッシュのためにいくらかの空きを残しておく必要があります。また、**MyISAM** に追加して使用するほかのストレージエンジンのメモリー要件も考慮します。

多くの行の同時書き込みなどスピードを高めるには、[LOCK TABLES](#) を使用します。[セクション8.2.5.1「INSERT ステートメントの最適化」](#)を参照してください。

キーバッファのパフォーマンスを確認するために、[SHOW STATUS](#) ステートメントを発行し、[Key_read_requests](#)、[Key_reads](#)、[Key_write_requests](#)、および [Key_writes](#) のステータス変数を調べることができます。([セクション13.7.7「SHOW ステートメント」](#) を参照してください。) [Key_reads/Key_read_requests](#) の比率は通常は 0.01 より小さくなります。操作がほとんど更新と削除だけの場合は [Key_writes/Key_write_requests](#) の比率は 1 に近くなりますが、同時に多くの行に影響を与える更新を行う場合や、[DELAY_KEY_WRITE](#) テーブルオプションを使用する場合はもっと小さくなる場合があります。

使用中のキーバッファの部分は、[key_buffer_size](#) に加えて、[Key_blocks_unused](#) ステータス変数と、[key_cache_block_size](#) システム変数から利用可能なバッファブロックサイズを使用して決定できます。

```
1 - ((Key_blocks_unused * key_cache_block_size) / key_buffer_size)
```

キーバッファ内の一部のスペースは、管理構造の内部で割り当てられるため、この値は概算です。これらの構造についてのオーバーヘッドの量に影響する要素には、ブロックサイズおよびポインタサイズがあります。ブロックサイズが増加すると、オーバーヘッドで失われるキーバッファのパーセントが減少する傾向にあります。ブロックが大きくなると、読み取り操作の数が少なくなりますが (読み取りあたりで取得されるキーが増えるため)、検査されないキーの読み取りが逆に増加します (ブロック内の一部のキーがクエリーに関連していない場合)。

MyISAM の複数キーキャッシュを作成できます。グループとしてではなく個別の各キャッシュに対して 4G バイトのサイズ制限が適用されます。[セクション8.10.2「MyISAM キーキャッシュ」](#)を参照してください。

- [key_cache_age_threshold](#)

コマンド行形式	--key-cache-age-threshold=#
システム変数	key_cache_age_threshold
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	300
最小値	100

最大値 (64 ビットプラットフォーム)	18446744073709551615
最大値 (32 ビットプラットフォーム)	4294967295

この値は、キーキャッシュのホットサブリストからウォームサブリストへのバッファの格下げを制御します。値が低いと格下げが早く行われます。最小値は 100 です。デフォルト値は 300 です。 [セクション8.10.2「MyISAM キーキャッシュ」](#) を参照してください。

- [key_cache_block_size](#)

コマンド行形式	<code>--key-cache-block-size=#</code>
システム変数	key_cache_block_size
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1024
最小値	512
最大値	16384

キーキャッシュ内のバイト単位のブロックのサイズ。デフォルト値は 1024 です。 [セクション8.10.2「MyISAM キーキャッシュ」](#) を参照してください。

- [key_cache_division_limit](#)

コマンド行形式	<code>--key-cache-division-limit=#</code>
システム変数	key_cache_division_limit
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	100
最小値	1
最大値	100

キーキャッシュバッファリストのホットサブリストとウォームサブリストの間の分割点。値は、ウォームサブリスト用に使用するバッファリストのパーセントです。許可される値の範囲は 1 から 100 です。デフォルト値は 100 です。 [セクション8.10.2「MyISAM キーキャッシュ」](#) を参照してください。

- [large_files_support](#)

システム変数	large_files_support
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean

大きなファイルをサポートするオプションで `mysqld` をコンパイルしているかどうか。

- [large_pages](#)

コマンド行形式	<code>--large-pages[={OFF ON}]</code>	717
---------	---------------------------------------	-----

システム変数	large_pages
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
プラットフォーム固有	Linux
型	Boolean
デフォルト値	OFF

大規模ページサポートが (`--large-pages` オプションで) 有効になっているかどうか。 [セクション8.12.3.2「ラージページのサポートの有効化」](#)を参照してください。

- [large_page_size](#)

システム変数	large_page_size
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0

大規模ページサポートが有効化されている場合、これはメモリーページのサイズを示します。ラージメモリーページは Linux でのみサポートされます。他のプラットフォームでは、この変数の値は常に 0 です。 [セクション 8.12.3.2「ラージページのサポートの有効化」](#)を参照してください。

- [last_insert_id](#)

`LAST_INSERT_ID()` から返される値。これは、テーブルを更新するステートメント内で `LAST_INSERT_ID()` を使用するときバイナリログ内に格納されます。この変数を設定しても、`mysql_insert_id()` C API 関数によって返される値は更新されません。

- [lc_messages](#)

コマンド行形式	<code>--lc-messages=name</code>
システム変数	lc_messages
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	en_US

エラーメッセージに使用するロケール。デフォルトは `en_US` です。サーバーは引数を言語名に変換し、これを `lc_messages_dir` の値と組み合わせてエラーメッセージファイルの場所を生成します。 [セクション10.12「エラーメッセージ言語の設定」](#)を参照してください。

- [lc_messages_dir](#)

コマンド行形式	<code>--lc-messages-dir=dir_name</code>
システム変数	lc_messages_dir
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ

型	ディレクトリ名
---	---------

エラーメッセージが配置されているディレクトリ。サーバーはこの値を `lc_messages` の値と一緒に使用して、エラーメッセージファイルの場所を生成します。 [セクション10.12「エラーメッセージ言語の設定」](#) を参照してください。

- [lc_time_names](#)

コマンド行形式	<code>--lc-time-names=value</code>
システム変数	<code>lc_time_names</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列

この変数は、日および月の名前と略語を表示するために使用する言語を制御するロケールを指定します。この変数は `DATE_FORMAT()`、`DAYNAME()`、および `MONTHNAME()` 関数の出力に影響を与えます。ロケール名は、`'ja_JP'` や `'pt_BR'` などの POSIX 規格の値です。システムのロケール設定に関係なく、デフォルト値は `'en_US'` です。詳細については、[セクション10.16「MySQL Server のロケールサポート」](#) を参照してください。

- [license](#)

システム変数	<code>license</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	GPL

サーバーが持つライセンスのタイプ。

- [local_infile](#)

コマンド行形式	<code>--local-infile[={OFF ON}]</code>
システム変数	<code>local_infile</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

この変数は、`LOAD DATA` ステートメントのサーバー側 `LOCAL` 機能を制御します。`local_infile` の設定に応じて、サーバーはクライアント側で `LOCAL` が有効になっているクライアントによるローカルデータロードを拒否または許可します。

サーバーが `LOAD DATA LOCAL` ステートメントを明示的に拒否または許可するようにするには (構築時または実行時にクライアントプログラムおよびライブラリがどのように構成されているかに関係なく)、それぞれ `local_infile` を無効または有効にして `mysqld` を起動します。`local_infile` は実行時に設定することもできます。詳細は、[セクション6.1.6「LOAD DATA LOCAL のセキュリティ上の考慮事項」](#) を参照してください。

- [lock_wait_timeout](#)

コマンド行形式	<code>--lock-wait-timeout=#</code>	719
---------	------------------------------------	-----

システム変数	lock_wait_timeout
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Integer
デフォルト値	31536000
最小値	1
最大値	31536000

この変数は、メタデータロックを取得するための試行のタイムアウトを秒単位で指定します。許可される値の範囲は 1 から 31536000 (1 年) です。デフォルトは 31536000 です。

このタイムアウトは、メタデータロックを使用するすべてのステートメントに適用されます。これらには、テーブル、ビュー、ストアドプロシージャ、ストアドファンクションの DML 操作および DDL 操作のほかに、[LOCK TABLES](#)、[FLUSH TABLES WITH READ LOCK](#)、および [HANDLER](#) ステートメントが含まれます。

このタイムアウトは、[GRANT](#) または [REVOKE](#) ステートメントやテーブルロギングステートメントによって変更される付与テーブルなど、[mysql](#) データベース内のシステムテーブルへの暗黙的なアクセスには適用されません。タイムアウトは、[SELECT](#) や [UPDATE](#) などによって直接アクセスされるシステムテーブルに適用されます。

タイムアウト値は、メタデータロック試行ごとに別々に適用されます。ある特定のステートメントが複数のロックを必要とする場合もあるため、タイムアウトエラーを報告する前に、ステートメントが [lock_wait_timeout](#) 値よりも長くブロックする可能性もあります。ロックタイムアウトが発生すると、[ER_LOCK_WAIT_TIMEOUT](#) が報告されます。

[lock_wait_timeout](#) では、[LOCK INSTANCE FOR BACKUP](#) ステートメントがロックを放棄するまで待機する時間も定義されます。

- [locked_in_memory](#)

システム変数	locked_in_memory
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ

[mysqld](#) が [--memlock](#) によってメモリー内でロックされたかどうか。

- [log_error](#)

コマンド行形式	--log-error[=file_name]
システム変数	log_error
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ファイル名

デフォルトのエラーログの保存先。宛先がコンソールの場合、値は [stderr](#) です。それ以外の場合、宛先はファイルで、[log_error](#) 値はファイル名です。 [セクション5.4.2「エラーログ」](#) を参照してください。

- [log_error_services](#)

コマンド行形式	--log-error-services=value
システム変数	log_error_services

スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	log_filter_internal; log_sink_internal

エラーロギングを有効にするコンポーネント。変数には、0、1 または多数の要素を含むリストを含めることができます。後者の場合、要素はセミコロンまたは (MySQL 8.0.12 の時点で) カンマで区切り、オプションでスペースを続けることができます。指定された設定にセミコロンとカンマの両方のセパレータを使用することはできません。サーバーはリストされた順序でコンポーネントを実行するため、コンポーネントの順序は重要です。log_error_services 値に指定されたロード可能な (組込みではない) コンポーネントは、最初に **INSTALL COMPONENT** とともにインストールする必要があります。詳細は、[セクション5.4.2.1「エラーログ構成」](#)を参照してください。

- [log_error_suppression_list](#)

コマンド行形式	--log-error-suppression-list=value
導入	8.0.13
システム変数	log_error_suppression_list
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	empty string

[log_error_suppression_list](#) システム変数は、エラーログを対象としたイベントに適用され、**WARNING** または **INFORMATION** の優先度で発生した場合に抑制するイベントを指定します。たとえば、特定のタイプの警告が頻繁に発生するが関心がないためにエラーログで望ましくない「ノイズ」とみなされる場合は、抑止できます。この変数は、デフォルトで有効になっている [log_filter_internal](#) エラーログフィルタコンポーネントによって実行されるフィルタリングに影響します ([セクション5.5.3「エラーログコンポーネント」](#)を参照)。log_filter_internal が無効になっている場合、log_error_suppression_list は影響を与えません。

[log_error_suppression_list](#) 値には、抑制しない場合は空の文字列、抑制するエラーコードを示すカンマ区切り値のリストを指定できます。エラーコードはシンボリックまたは数値形式で指定できます。数値コードは、MY- 接頭辞の有無にかかわらず指定できます。数値部分の先頭のゼロは重要ではありません。許可されているコード形式の例:

```
ER_SERVER_SHUTDOWN_COMPLETE
MY-000031
000031
MY-31
31
```

読みやすく移植性を高めるために、数値よりもシンボリック値をお勧めします。許可されるエラー記号およびエラー番号の詳細は、[MySQL 8.0 Error Message Reference](#) を参照してください。

[log_error_suppression_list](#) の効果は、[log_error_verbosity](#) の効果と結合されます。追加情報については [セクション5.4.2.5「優先度ベースのエラーログのフィルタリング \(log_filter_internal\)」](#)を参照してください。

- [log_error_verbosity](#)

コマンド行形式	--log-error-verbosity=#
システム変数	log_error_verbosity
スコープ	グローバル
動的	はい

SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	2
最小値	1
最大値	3

`log_error_verbosity` システム変数は、エラーログを対象としたイベントを処理するための冗長性を指定します。この変数は、デフォルトで有効になっている `log_filter_internal` エラーログフィルタコンポーネントによって実行されるフィルタリングに影響します (セクション5.5.3「エラーログコンポーネント」を参照)。`log_filter_internal` が無効になっている場合、`log_error_verbosity` は影響を与えません。

エラーログを対象としたイベントの優先度は、`ERROR`、`WARNING` または `INFORMATION` です。`log_error_verbosity` は、次のテーブルに示すように、ログに書き込まれるメッセージに対して許可する優先度に基づいて冗長性を制御します。

log_error_verbosity 値	許可されたメッセージの優先度
1	<code>ERROR</code>
2	<code>ERROR</code> , <code>WARNING</code>
3	<code>ERROR</code> , <code>WARNING</code> , <code>INFORMATION</code>

`SYSTEM` の優先度もあります。`log_error_verbosity` の値に関係なく、エラー以外の状況に関するシステムメッセージがエラーログに出力されます。これらのメッセージには、起動メッセージと停止メッセージ、および設定に対する重要な変更が含まれます。

`log_error_verbosity` の効果は、`log_error_suppression_list` の効果と結合されます。追加情報については [セクション 5.4.2.5「優先度ベースのエラーログのフィルタリング \(log_filter_internal\)」](#) を参照してください。

- `log_output`

コマンド行形式	<code>--log-output=name</code>
システム変数	<code>log_output</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Set
デフォルト値	<code>FILE</code>
有効な値	<code>TABLE</code> <code>FILE</code> <code>NONE</code>

一般クエリーログおよびスロークエリーログ出力の宛先。値は、`TABLE`、`FILE` および `NONE` から選択されたカンマ区切りの単語のリストです。`TABLE` は、`mysql` システムスキーマ内の `general_log` および `slow_log` テーブルへのロギングを選択します。`FILE` は、ログファイルへのロギングを選択します。`NONE` はロギングを無効にします。値に `NONE` が存在する場合は、存在する他の単語よりも優先されます。`TABLE` と `FILE` の両方を指定して、両方のログ出力先を選択できます。

この変数は、ログ出力先を選択しますが、ログ出力は有効にしません。これを行うには、`general_log` および `slow_query_log` システム変数を有効にします。`FILE` ロギングの場合、`general_log_file` および `slow_query_log_file` システム変数によってログファイルの場所が決まります。詳細については、[セクション5.4.1「一般クエリーログおよびスロークエリーログの出力先の選択」](#) を参照してください。

- [log_queries_not_using_indexes](#)

コマンド行形式	<code>--log-queries-not-using-indexes[={OFF ON}]</code>
システム変数	log_queries_not_using_indexes
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

スロークエリーログを有効にしてこの変数を有効にすると、すべての行を取得すると予想されるクエリーがログに記録されます。 [セクション5.4.5「スロークエリーログ」](#)を参照してください。このオプションは、インデックスが使用されないことを必ずしも意味するわけではありません。たとえば、フルインデックススキャンを使用するクエリーはインデックスを使用しますが、インデックスは行数を制限しないため、クエリーはログに記録されます。

- [log_raw](#)

コマンド行形式	<code>--log-raw[={OFF ON}]</code>
システム変数 (≥ 8.0.19)	log_raw
スコープ (≥ 8.0.19)	グローバル
動的 (≥ 8.0.19)	はい
SET_VAR ヒントの適用 (≥ 8.0.19)	いいえ
型	Boolean
デフォルト値	OFF

[log_raw](#) システム変数は、最初は `--log-raw` オプションの値に設定されます。詳細は、そのオプションの説明を参照してください。システム変数を実行時に設定して、パスワードマスキング動作を変更することもできます。

- [log_slow_admin_statements](#)

コマンド行形式	<code>--log-slow-admin-statements[={OFF ON}]</code>
システム変数	log_slow_admin_statements
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

スロークエリーログに書き込まれるステートメントにスロー管理ステートメントを含めます。管理ステートメントには、[ALTER TABLE](#)、[ANALYZE TABLE](#)、[CHECK TABLE](#)、[CREATE INDEX](#)、[DROP INDEX](#)、[OPTIMIZE TABLE](#)、および [REPAIR TABLE](#) が含まれます。

- [log_slow_extra](#)

コマンド行形式	<code>--log-slow-extra[={OFF ON}]</code>
導入	8.0.14
システム変数	log_slow_extra
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ

型	Boolean
デフォルト値	OFF

スロークエリーログが有効で、出力先に `FILE` が含まれている場合、サーバーはスローステートメントに関する情報を提供する追加のフィールドをログファイル行に書き込みます。セクション5.4.5「スロークエリーログ」を参照してください。TABLE 出力は影響を受けません。

- `log_syslog`

コマンド行形式	<code>--log-syslog[={OFF ON}]</code>
非推奨	はい (removed in 8.0.13)
システム変数	<code>log_syslog</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON (システムログへのエラーロギングが有効な場合)

MySQL 8.0 より前では、この変数はシステムログ (Windows の場合はイベントログ、Unix および Unix に似たシステムの場合は `syslog`) にエラーロギングを実行するかどうかを制御していました。

MySQL 8.0 では、`log_sink_syseventlog` ログコンポーネントによってシステムログへのエラーロギングが実装されるため (セクション5.4.2.8「システムログへのエラーロギング」を参照)、`log_error_services` システム変数にこのタイプのロギングを追加することで有効にできます。`log_syslog` が削除されます。(MySQL 8.0.13 より前は、`log_syslog` は存在していましたが、非推奨であり、効果はありません。)

- `log_syslog_facility`

コマンド行形式	<code>--log-syslog-facility=value</code>
削除	8.0.13
システム変数	<code>log_syslog_facility</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	<code>daemon</code>

この変数は MySQL 8.0.13 で削除され、`syseventlog.facility` に置き換えられました。

- `log_syslog_include_pid`

コマンド行形式	<code>--log-syslog-include-pid[={OFF ON}]</code>
削除	8.0.13
システム変数	<code>log_syslog_include_pid</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

この変数は MySQL 8.0.13 で削除され、`syseventlog.include_pid` に置き換えられました。

- [log_syslog_tag](#)

コマンド行形式	<code>--log-syslog-tag=tag</code>
削除	8.0.13
システム変数	log_syslog_tag
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	empty string

この変数は MySQL 8.0.13 で削除され、[syseventlog.tag](#) に置き換えられました。

- [log_timestamps](#)

コマンド行形式	<code>--log-timestamps=#</code>
システム変数	log_timestamps
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	UTC
有効な値	UTC SYSTEM

この変数は、エラーログに書き込まれるメッセージのタイムスタンプ、およびファイルに書き込まれる一般的なクエリーログメッセージとスロークエリーログメッセージのタイムゾーンを制御します。一般クエリーログおよびテーブル ([mysql.general_log](#)、[mysql.slow_log](#)) に書き込まれるスロークエリーログメッセージのタイムゾーンには影響しません。これらのテーブルから取得された行は、[CONVERT_TZ\(\)](#) を使用するか、セッションの [time_zone](#) システム変数を設定することで、ローカルシステムのタイムゾーンから任意のタイムゾーンに変換できます。

許可される [log_timestamps](#) 値は、UTC (デフォルト) および SYSTEM (ローカルシステムのタイムゾーン) です。

タイムスタンプは ISO 8601 / RFC 3339 形式を使用して書き込まれます: YYYY-MM-DDThh:mm:ss.uuuuuu に Zulu 時間 (UTC) または ±hh:mm (UTC からのオフセット) を示す Z の末尾の値を加えたもの。

- [log_throttle_queries_not_using_indexes](#)

コマンド行形式	<code>--log-throttle-queries-not-using-indexes=#</code>
システム変数	log_throttle_queries_not_using_indexes
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0

[log_queries_not_using_indexes](#) が有効な場合、[log_throttle_queries_not_using_indexes](#) 変数は、スロークエリーログに書き込み可能な分あたりのクエリー数を制限します。値 0 (デフォルト) は「制限なし」を意味します。詳細は、[セクション5.4.5「スロークエリーログ」](#)を参照してください。

- [long_query_time](#)

コマンド行形式	--long-query-time=#
システム変数	long_query_time
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	数値
デフォルト値	10
最小値	0

クエリーの時間がこの秒数よりかかると、サーバーは [Slow_queries](#) ステータス変数を増やします。スロークエリーログが有効な場合、クエリーはスロークエリーログファイルに記録されます。この値は CPU 時間でなくリアルタイムで測定されるため、負荷の軽いシステムでしきい値を下回るクエリーが、負荷の重いシステムではしきい値を超える場合もあります。[long_query_time](#) の最小値およびデフォルト値は、それぞれ 0 および 10 です。値はマイクロ秒の精度まで指定できます。[セクション5.4.5「スロークエリーログ」](#)を参照してください。

この変数の値を小さくすると、長時間実行とみなされるステートメントが増えるため、スロークエリーログに必要な領域が増えます。非常に小さい値 (1 秒未満) の場合、ログは非常に大きくなる可能性があります。長時間実行とみなされるステートメントの数を増やすと、特にグループレプリケーションが有効になっている場合に、MySQL Enterprise Monitor の「長時間実行プロセスの数が多すぎます」アラートに対して誤検出が発生する可能性もあります。これらの理由から、非常に小さい値はテスト環境でのみ使用するか、本番環境では短期間のみ使用する必要があります。

- [low_priority_updates](#)

コマンド行形式	--low-priority-updates[={OFF ON}]
システム変数	low_priority_updates
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

1 に設定された場合、すべての [INSERT](#)、[UPDATE](#)、[DELETE](#)、および [LOCK TABLE WRITE](#) ステートメントは、影響を受けるテーブルでの保留中の [SELECT](#) または [LOCK TABLE READ](#) がなくなるまで待機します。[{INSERT | REPLACE | DELETE | UPDATE} LOW_PRIORITY ...](#) を使用して同じ効果を得て、クエリーの優先度を下げることができます。この変数は、テーブルレベルのロック ([MyISAM](#)、[MEMORY](#)、[MERGE](#) など) のみを使用するストレージエンジンにのみ影響します。[セクション8.11.2「テーブルロックの問題」](#)を参照してください。

- [lower_case_file_system](#)

システム変数	lower_case_file_system
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean

この変数は、データディレクトリが配置されているファイルシステムでのファイル名の大文字小文字の区別を示します。[OFF](#) では、ファイル名は大/小文字が区別され、[ON](#) では大/小文字が区別されません。この変数は、ファイルシステム属性を反映するため読み取り専用で、変数を設定してもファイルシステムに影響しません。

- [lower_case_table_names](#)

コマンド行形式	<code>--lower-case-table-names[=#]</code>
システム変数	<code>lower_case_table_names</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	2

0 に設定すると、テーブル名は指定したとおりに格納され、比較では大/小文字が区別されます。1 に設定すると、テーブル名はディスクに小文字で格納され、比較では大文字と小文字は区別されません。2 に設定されると、テーブル名は指定したとおりに格納されますが、小文字で比較されます。このオプションはデータベース名やテーブルエイリアスにも適用されます。追加の詳細については、[セクション9.2.3「識別子の小文字と大文字の区別」](#)を参照してください。

Windows では、デフォルト値は 1 です。macOS では、デフォルト値は 2 です。Linux では、値 2 はサポートされていません。かわりに、サーバーは値を強制的に 0 にします。

データディレクトリが大/小文字を区別しないファイルシステム (Windows や macOS など) に存在するシステムで MySQL を実行している場合は、`lower_case_table_names` を 0 に設定しないでください。これはサポートされていない組合せであり、間違った `tbl_name` 文字ケースで `INSERT INTO ... SELECT ... FROM tbl_name` 操作を実行するとハング状態になる可能性があります。MyISAM では、異なる大文字と小文字を使用してテーブル名にアクセスすると、インデックスが破損する可能性があります。

大/小文字を区別しないファイルシステムで `--lower_case_table_names=0` を使用してサーバーを起動しようとすると、エラーメッセージが出力され、サーバーが終了します。

InnoDB テーブルを使用する場合、名前を強制的に小文字に変換するために、すべてのプラットフォームでこの値を 1 に設定します。

この変数の設定は、大文字と小文字の区別に関してレプリケーションフィルタリングオプションの動作に影響します。詳細は、[セクション17.2.5「サーバーがレプリケーションフィルタリングルールをどのように評価するか」](#)を参照してください。

サーバーの初期化時に使用された設定とは異なる `lower_case_table_names` 設定でサーバーを起動することは禁止されています。様々なデータディクショナリテーブルのフィールドで使用される照合は、サーバーの初期化時に定義された設定によって決定され、異なる設定でサーバーを再起動すると、識別子の順序付けおよび比較方法に矛盾が生じるため、制限が必要です。

したがって、サーバーを初期化する前に、`lower_case_table_names` を目的の設定に構成する必要があります。ほとんどの場合、MySQL サーバーを初めて起動する前に、MySQL オプションファイルで `lower_case_table_names` を構成する必要があります。ただし、Debian および Ubuntu での APT インストールの場合、サーバーは初期化され、事前にオプションファイルで設定を構成する機会はありません。そのため、`lower_case_table_names` を有効にするには、APT を使用して MySQL をインストールする前に `debconf-set-selections` ユーティリティを使用する必要があります。これを行うには、APT を使用して MySQL をインストールする前に、次のコマンドを実行します:

```
shell> sudo debconf-set-selections <<< "mysql-server mysql-server/lowercase-table-names select Enabled
```

注記

`debconf-set-selections` を使用して `lower_case_table_names` を有効にする機能が MySQL 8.0.17 に追加されました。 `lower_case_table_names` を有効にすると、値は 1 に設定されます。

- [mandatory_roles](#)

コマンド行形式	<code>--mandatory-roles=value</code>
システム変数	<code>mandatory_roles</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	<code>empty string</code>

サーバーが必須として扱う必要があるロール。実際には、これらのロールはすべてのユーザーに自動的に付与されますが、`mandatory_roles` を設定しても実際にはユーザーアカウントは変更されず、付与されたロールは `mysql.role_edges` システムテーブルに表示されません。

変数値は、ロール名のカンマ区切りリストです。例:

```
SET PERSIST mandatory_roles = 'role1`@`%`,`role2`,`role3,role4@localhost';
```

`mandatory_roles` のランタイム値を設定するには、グローバルシステム変数のランタイム値を設定するために通常必要な `SYSTEM_VARIABLES_ADMIN` 権限 (または非推奨の `SUPER` 権限) に加えて、`ROLE_ADMIN` 権限が必要です。

ロール名は、`user_name@host_name` 形式のユーザー部分とホスト部分で構成されます。ホスト部分を省略すると、デフォルトで `%` に設定されます。追加情報については [セクション6.2.5「ロール名の指定」](#) を参照してください。

`mandatory_roles` 値は文字列であるため、引用符で囲まれている場合は、引用符で囲まれた文字列内での引用に許可された方法でユーザー名とホスト名を記述する必要があります。

`mandatory_roles` の値で指定されたロールは、`REVOKE` で取り消すことも、`DROP ROLE` または `DROP USER` で削除することもできません。

セッションがデフォルトでシステムセッションにならないようにするには、`SYSTEM_USER` 権限を持つロールを `mandatory_roles` システム変数の値にリストできません:

- `SYSTEM_USER` 権限を持つロールが起動時に `mandatory_roles` に割り当てられた場合、サーバーはエラーログにメッセージを書き込み、終了します。
- `SYSTEM_USER` 権限を持つロールが実行時に `mandatory_roles` に割り当てられた場合、エラーが発生し、`mandatory_roles` 値は変更されません。

明示的に付与されたロールと同様に、必須ロールはアクティブ化されるまで有効になりません ([ロールのアクティブ化](#) を参照)。ログイン時に、`activate_all_roles_on_login` システム変数が有効になっている場合は付与されているすべてのロールに対してロールのアクティブ化が行われ、それ以外の場合はデフォルトロールとして設定されているロールに対してロールのアクティブ化が行われます。実行時に、`SET ROLE` によってロールがアクティブ化されません。

`mandatory_roles` に割り当てられたときに存在しないが、後で作成されるロールでは、特別な処理を必須とみなす必要がある場合があります。詳細は、[必須ロールの定義](#) を参照してください。

`SHOW GRANTS` では、[セクション13.7.7.21「SHOW GRANTS ステートメント」](#) で説明されているルールに従って必須ロールが表示されます。

- [max_allowed_packet](#)

コマンド行形式	<code>--max-allowed-packet=#</code>
システム変数	<code>max_allowed_packet</code>
スコープ	グローバル、セッション

動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	67108864
最小値	1024
最大値	1073741824

1つのパケット、生成された文字列または中間文字列、または `mysql_stmt_send_long_data()` C API 関数によって送信されたすべてのパラメータの最大サイズ。デフォルトは 64M バイトです。

パケットメッセージバッファは `net_buffer_length` バイトに初期化されますが、必要に応じて `max_allowed_packet` バイトまで大きくできます。この値はデフォルトでは小さいため、大きい (正しくない可能性がある) パケットをキャッチできません。

大きい BLOB カラムまたは長い文字列を使用している場合、この値を大きくする必要があります。使用する最大の BLOB と同じ大きさにしてください。 `max_allowed_packet` のプロトコル制限は 1G バイトです。値は 1024 の倍数にします。倍数でない場合、もっとも近い倍数に切り下げられます。

`max_allowed_packet` 変数の値を変更することによってメッセージバッファサイズを変更するとき、クライアントプログラムでそれが可能である場合は、クライアント側のバッファサイズも変更します。クライアントライブラリに組み込まれるデフォルトの `max_allowed_packet` 値は 1G バイトですが、個々のクライアントプログラムはこれをオーバーライドできます。たとえば、`mysql` および `mysqldump` のデフォルトは、それぞれ 16M バイトおよび 24M バイトです。また、コマンド行またはオプションファイル内で `max_allowed_packet` を設定することによって、クライアント側の値を変更することもできます。

この変数のセッションの値は、読み取り専用です。クライアントは、セッション値と同じバイト数まで受信できません。ただし、サーバーは現在のグローバル `max_allowed_packet` 値を超えるバイト数をクライアントに送信しません。(クライアントの接続後にグローバル値が変更された場合、グローバル値はセッション値より小さくなる可能性があります。)

- `max_connect_errors`

コマンド行形式	<code>--max-connect-errors=#</code>
システム変数	<code>max_connect_errors</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	100
最小値	1
最大値 (64 ビットプラットフォーム)	18446744073709551615
最大値 (32 ビットプラットフォーム)	4294967295

正常に接続されずにホストからの `max_connect_errors` の連続した接続リクエストが中断されると、サーバーはそのホストをそれ以降の接続からブロックします。前回の接続が中断された後、`max_connect_errors` の試行回数より少ない時間内にホストからの接続が正常に確立された場合、ホストのエラー数はゼロにクリアされます。ブロックされたホストのブロックを解除するには、ホストキャッシュをフラッシュします。[ホストキャッシュのフラッシュ](#) を参照してください。

- `max_connections`

コマンド行形式	<code>--max-connections=#</code>
システム変数	<code>max_connections</code>
スコープ	グローバル

動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	151
最小値	1
最大値	100000

許可される最大のクライアントの同時接続数。詳細は、[セクション5.1.12.1「接続インターフェース」](#)を参照してください。

- [max_delayed_threads](#)

コマンド行形式	<code>--max-delayed-threads=#</code>
非推奨	はい
システム変数	max_delayed_threads
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	20
最小値	0
最大値	16384

このシステム変数は非推奨です (`DELAYED` の挿入はサポートされていないため)。将来のリリースで削除される予定です。

- [max_digest_length](#)

コマンド行形式	<code>--max-digest-length=#</code>
システム変数	max_digest_length
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1024
最小値	0
最大値	1048576

正規化されたステートメントダイジェストの計算用にセッションごとに予約されているメモリの最大バイト数。ダイジェスト計算中にその量の領域が使用されると、切捨てが発生: 解析されたステートメントからのそれ以上のトークンは収集されず、ダイジェスト値になりません。解析されたトークンのバイト数が同じ正規化されたステートメントダイジェストを生成し、比較された場合、またはダイジェスト統計のために集計された場合にのみ、ステートメントが同じであるとみなされます。

[max_digest_length](#) 値を小さくするとメモリ使用量は減少しますが、末尾のみが異なる場合は、より多くのステートメントのダイジェスト値が区別できなくなります。この値を大きくすると、より長いステートメントを区別できますが、特に多数の同時セッションを含むワークロード (サーバーはセッションごとに [max_digest_length](#) バイトを割り当てます) では、メモリ使用量が増加します。

パーサーは、このシステム変数を、計算する正規化されたステートメントダイジェストの最大長の制限として使用します。パフォーマンススキーマは、ステートメントダイジェストを追跡する場合、格納するダイジェストの最大

長の制限として `performance_schema_max_digest_length`. システム変数を使用してダイジェスト値のコピーを作成します。したがって、`performance_schema_max_digest_length` が `max_digest_length` より小さい場合、パフォーマンススキーマに格納されているダイジェスト値は、元のダイジェスト値と比較して切り捨てられます。

ステートメントダイジェストの詳細については、[セクション27.10「パフォーマンススキーマのステートメントダイジェストとサンプリング」](#)を参照してください。

- `max_error_count`

コマンド行形式	<code>--max-error-count=#</code>
システム変数	<code>max_error_count</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Integer
デフォルト値	1024
最小値	0
最大値	65535

`SHOW ERRORS` および `SHOW WARNINGS` ステートメントで表示するために格納されるエラー、警告および情報メッセージの最大数。これは診断領域内の条件領域の数と同じで、`GET DIAGNOSTICS` によって調査できる条件数と同じです。

- `max_execution_time`

コマンド行形式	<code>--max-execution-time=#</code>
システム変数	<code>max_execution_time</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Integer
デフォルト値	0

`SELECT` ステートメントの実行タイムアウト (ミリ秒)。値が 0 の場合、タイムアウトは有効になりません。

`max_execution_time` は次のように適用されます:

- グローバル `max_execution_time` 値は、新しい接続のセッション値のデフォルトを提供します。セッション値は、`MAX_EXECUTION_TIME(N)` オプティマイザヒントを含まない、または `N` が 0 のセッション内で実行される `SELECT` 実行に適用されます。
- `max_execution_time` は、読み取り専用の `SELECT` ステートメントに適用されます。読み取り専用でないステートメントは、副作用としてデータを変更するストアドファンクションを呼び出すステートメントです。
- ストアドプログラムの `SELECT` ステートメントでは、`max_execution_time` は無視されます。

- `max_heap_table_size`

コマンド行形式	<code>--max-heap-table-size=#</code>
システム変数	<code>max_heap_table_size</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい

型	Integer
デフォルト値	16777216
最小値	16384
最大値 (64 ビットプラットフォーム)	1844674407370954752
最大値 (32 ビットプラットフォーム)	4294967295

この変数は、ユーザーが作成した `MEMORY` テーブルの増加が許可される最大サイズを設定します。この変数の値は `MEMORY` テーブルの `MAX_ROWS` 値を計算するために使用されます。この変数を設定しても、既存の `MEMORY` テーブルに影響しませんが、`CREATE TABLE` などのステートメントでテーブルを再作成したり、`ALTER TABLE` または `TRUNCATE TABLE` でテーブルを変更したりした場合は影響します。サーバーを再起動しても、既存の `MEMORY` テーブルの最大サイズがグローバルの `max_heap_table_size` 値に設定されます。

この変数は、内部インメモリーテーブルのサイズを制限するために `tmp_table_size` と一緒に使用されることもあります。セクション8.4.4「MySQLでの内部一時テーブルの使用」を参照してください。

`max_heap_table_size` は複製されません。詳しくは、セクション17.5.1.21「レプリケーションと `MEMORY` テーブル」およびセクション17.5.1.39「レプリケーションと変数」を参照してください。

- `max_insert_delayed_threads`

非推奨	はい
システム変数	<code>max_insert_delayed_threads</code>
スコープ	グローバル、セッション
動的	はい
<code>SET_VAR</code> ヒントの適用	いいえ
型	Integer

この変数は、`max_delayed_threads` のシノニムです。

このシステム変数は非推奨です (`DELAYED` の挿入はサポートされていないため)。将来のリリースで削除される予定です。

- `max_join_size`

コマンド行形式	<code>--max-join-size=#</code>
システム変数	<code>max_join_size</code>
スコープ	グローバル、セッション
動的	はい
<code>SET_VAR</code> ヒントの適用	はい
型	Integer
デフォルト値	18446744073709551615
最小値	1
最大値	18446744073709551615

検査が必要となる行数 (単一テーブルステートメントの場合) または行の組み合わせの数 (複数テーブルステートメントの場合) が、`max_join_size` をおそらく超えるか、ディスクシークが `max_join_size` 回を超えて実行される可能性があるステートメントを許可しません。この値を設定することで、キーが適切に使用されず長い時間がかかりそうなステートメントをキャッチできます。ユーザーが、`WHERE` 句のない結合、長い時間がかかる結合、または数百万行を返す結合を実行する傾向がある場合にこれを設定します。詳細は、セーフ更新モードの使用 (`--safe-updates`) を参照してください。

この変数を `DEFAULT` 以外の値に設定すると、`sql_big_selects` の値が 0 にリセットされます。 `sql_big_selects` 値を再設定すると、`max_join_size` 変数は無視されます。

- [max_length_for_sort_data](#)

コマンド行形式	<code>--max-length-for-sort-data=#</code>
非推奨	8.0.20
システム変数	max_length_for_sort_data
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Integer
デフォルト値	4096
最小値	4
最大値	8388608

この変数は、オプティマイザの変更によって廃止され、効果がないため、MySQL 8.0.20 では非推奨になりました。以前は、使用する `filesort` アルゴリズムを決定するインデックス値のサイズに対するカットオフとして機能していました。 [セクション8.2.1.16「ORDER BY の最適化」](#) を参照してください。

- [max_points_in_geometry](#)

コマンド行形式	<code>--max-points-in-geometry=#</code>
システム変数	max_points_in_geometry
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Integer
デフォルト値	65536
最小値	3
最大値	1048576

`ST_Buffer_Strategy()` 関数に対する `points_per_circle` 引数の最大値。

- [max_prepared_stmt_count](#)

コマンド行形式	<code>--max-prepared-stmt-count=#</code>
システム変数	max_prepared_stmt_count
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	16382
最小値	0
最大値 (≥ 8.0.18)	4194304
最大値 (≤ 8.0.17)	1048576

この変数は、サーバー内のプリペアドステートメントの総数を制限します。これは、大量のステートメントを作成することによってサーバーの実行するメモリーを不足させることに基づくサービス妨害攻撃の可能性がある環境で使用できます。値が現在のプリペアドステートメントの数より低く設定された場合、既存のステートメントは影響を受けずに使用できますが、現在の数が制限を下回るまで新しいステートメントを作成できません。値を 0 に設定すると、プリペアドステートメントが無効になります。

- [max_seeks_for_key](#)

コマンド行形式	<code>--max-seeks-for-key=#</code>
システム変数	max_seeks_for_key
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Integer
デフォルト値 (Windows)	4294967295
デフォルト値 (その他, 64 ビットプラットフォーム)	18446744073709551615
デフォルト値 (その他, 32 ビットプラットフォーム)	4294967295
最小値	1
最大値 (Windows)	4294967295
最大値 (その他, 64 ビットプラットフォーム)	18446744073709551615
最大値 (その他, 32 ビットプラットフォーム)	4294967295

キーに基づいて行を参照するとき、推定されるシークの最大数を制限します。MySQL オプティマイザは、インデックスをスキャンすることによってテーブル内で一致する行を検索するとき、インデックスの実際のカーディナリティーに関係なく、この数を超えるキーシークは不要であると推定します ([セクション13.7.7.22「SHOW INDEX ステートメント」](#)を参照してください)。これを低い値 (100 など) に設定することで、MySQL でテーブルスキャンよりもインデックスを優先するように強制できます。

- [max_sort_length](#)

コマンド行形式	<code>--max-sort-length=#</code>
システム変数	max_sort_length
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Integer
デフォルト値	1024
最小値	4
最大値	8388608

データ値をソートするときに使用するバイト数。サーバーは、各値の最初の [max_sort_length](#) バイトのみを使用し、残りは無視します。したがって、最初の [max_sort_length](#) バイトの後にのみ異なる値は、[GROUP BY](#)、[ORDER BY](#) および [DISTINCT](#) 操作で等しいと比較されます。

[max_sort_length](#) の値を増やすには、[sort_buffer_size](#) の値も増やす必要があります。詳細は、[セクション 8.2.1.16「ORDER BY の最適化」](#)を参照してください

- [max_sp_recursion_depth](#)

コマンド行形式	<code>--max-sp-recursion-depth[=#]</code>
システム変数	max_sp_recursion_depth
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer

デフォルト値	0
最大値	255

任意のストアードプロシージャを再帰的に呼び出すことができる回数。このオプションのデフォルト値は 0 で、これはストアードプロシージャの再帰を完全に無効化します。最大値は 255 です。

ストアードプロシージャの再帰により、スレッドスタック領域の要求が増加します。[max_sp_recursion_depth](#) の値を増やした場合、サーバー起動時に [thread_stack](#) の値を増やすことによってスレッドスタックサイズを増やすことが必要な場合もあります。

- [max_user_connections](#)

コマンド行形式	<code>--max-user-connections=#</code>
システム変数	max_user_connections
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	4294967295

任意の MySQL ユーザーアカウントに許可された最大同時接続数。値 0 (デフォルト) は「制限なし」を意味します。

この変数は、サーバー起動時または実行時に設定できるグローバル値を持ちます。また、現在のセッションに関連付けられたアカウントに適用される、実際の同時接続制限を示す読み取り専用のセッション値も持ちます。セッション値は次のように初期化されます。

- ユーザーアカウントの [MAX_USER_CONNECTIONS](#) リソース制限がゼロでない場合、セッション [max_user_connections](#) の値はその制限値に設定されます。
- そうでない場合、セッション [max_user_connections](#) の値はグローバル値に設定されます。

アカウントのリソース制限は、[CREATE USER](#) ステートメントまたは [ALTER USER](#) ステートメントを使用して指定します。[セクション6.2.20「アカウントリソース制限の設定」](#)を参照してください。

- [max_write_lock_count](#)

コマンド行形式	<code>--max-write-lock-count=#</code>
システム変数	max_write_lock_count
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値 (Windows)	4294967295
デフォルト値 (その他, 64 ビットプラットフォーム)	18446744073709551615
デフォルト値 (その他, 32 ビットプラットフォーム)	4294967295
最小値	1
最大値 (Windows)	4294967295
最大値 (その他, 64 ビットプラットフォーム)	18446744073709551615
最大値 (その他, 32 ビットプラットフォーム)	4294967295

この大きさの書き込みロックのあと、保留中の読み取りロックリクエストの処理を間で許可します。書き込みロック要求の優先順位は、読み取りロック要求よりも高くなります。ただし、`max_write_lock_count`がある程度低い値(たとえば、10)に設定されている場合、読み取りロック要求がすでに10個の書き込みロック要求を優先して渡されていれば、保留中の書き込みロック要求よりも読み取りロック要求が優先されることがあります。通常、`max_write_lock_count`のデフォルト値は非常に大きいため、この動作は発生しません。

- `mecab_rc_file`

コマンド行形式	<code>--mecab-rc-file=file_name</code>
システム変数	<code>mecab_rc_file</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ファイル名

`mecab_rc_file` オプションは、MeCab 全文パーサーの設定時に使用されます。

`mecab_rc_file` オプションは、MeCab の構成ファイルである `mecabrc` 構成ファイルへのパスを定義します。このオプションは読み取り専用で、起動時にのみ設定できます。MeCab を初期化するには、`mecabrc` 構成ファイルが必要です。

MeCab 全文パーサーの詳細は、[セクション12.10.9「MeCab フルテキストパーサープラグイン」](#) を参照してください。

MeCab `mecabrc` 構成ファイルで指定できるオプションの詳細は、「[Google 開発者](#)」サイトの「[MeCab ドキュメント](#)」を参照してください。

- `metadata_locks_cache_size`

コマンド行形式	<code>--metadata-locks-cache-size=#</code>
非推奨	はい (removed in 8.0.13)
システム変数	<code>metadata_locks_cache_size</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1024
最小値	1
最大値	1048576

このシステム変数は、MySQL 8.0.13 で削除されました。

- `metadata_locks_hash_instances`

コマンド行形式	<code>--metadata-locks-hash-instances=#</code>
非推奨	はい (removed in 8.0.13)
システム変数	<code>metadata_locks_hash_instances</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer

デフォルト値	8
最小値	1
最大値	1024

このシステム変数は、MySQL 8.0.13 で削除されました。

- [min_examined_row_limit](#)

コマンド行形式	<code>--min-examined-row-limit=#</code>
システム変数	min_examined_row_limit
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値 (64 ビットプラットフォーム)	18446744073709551615
最大値 (32 ビットプラットフォーム)	4294967295

これよりも少ない行数を検査するクエリーは、スロークエリーログに記録されません。

- [myisam_data_pointer_size](#)

コマンド行形式	<code>--myisam-data-pointer-size=#</code>
システム変数	myisam_data_pointer_size
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	6
最小値	2
最大値	7

`MAX_ROWS` オプションが指定されていない場合に `MyISAM` テーブルの `CREATE TABLE` によって使用されるバイト単位のデフォルトポインタサイズ。この値を 2 より小さくしたり 7 より大きくしたりすることはできません。デフォルト値は 6 です。 [セクション B.3.2.10 「テーブルが満杯です」](#) を参照してください。

- [myisam_max_sort_file_size](#)

コマンド行形式	<code>--myisam-max-sort-file-size=#</code>
システム変数	myisam_max_sort_file_size
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値 (Windows)	2146435072
デフォルト値 (その他, 64 ビットプラットフォーム)	9223372036853727232
デフォルト値 (その他, 32 ビットプラットフォーム)	2147483648

最大値 (Windows)	2146435072
最大値 (その他, 64 ビットプラットフォーム)	9223372036853727232
最大値 (その他, 32 ビットプラットフォーム)	2147483648

MyISAM インデックスの再作成時 (**REPAIR TABLE**、**ALTER TABLE** または **LOAD DATA**) に MySQL で使用できる一時ファイルの最大サイズ。ファイルサイズがこの値より大きい場合、さらに低速なキーキャッシュを代わりに使用してインデックスが作成されます。値はバイト単位で指定されます。

MyISAM インデックスファイルがこのサイズを超えて、ディスクスペースが使用できる場合、この値を大きくするとパフォーマンスが向上することがあります。このスペースは、元のインデックスファイルが配置されているディレクトリを含むファイルシステム内で利用する必要があります。

- [myisam_mmap_size](#)

コマンド行形式	<code>--myisam-mmap-size=#</code>
システム変数	<code>myisam_mmap_size</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値 (64 ビットプラットフォーム)	18446744073709551615
デフォルト値 (32 ビットプラットフォーム)	4294967295
最小値	7
最大値 (64 ビットプラットフォーム)	18446744073709551615
最大値 (32 ビットプラットフォーム)	4294967295

圧縮された **MyISAM** ファイルのメモリーマッピングに使用する最大のメモリー量。圧縮された **MyISAM** テーブルが多く使用される場合、この値を減らすことで、メモリースワッピングの問題が生じるおそれを低下できます。

- [myisam_recover_options](#)

コマンド行形式	<code>--myisam-recover-options[=list]</code>
システム変数	<code>myisam_recover_options</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	OFF
有効な値	OFF DEFAULT BACKUP FORCE QUICK

MyISAM のストレージエンジンのリカバリモードを設定します。変数値は、**OFF**、**DEFAULT**、**BACKUP**、**FORCE** または **QUICK** の値の任意の組合せです。複数の値を指定する場合、値をカンマで区切ります。サーバー起動時に値なしで変数を指定することは **DEFAULT** を指定することと同じで、明示的な値を""に指定するとリカバリが無効になります (**OFF** の値と同じ)。リカバリが有効な場合、`mysqld` は **MyISAM** テーブルをオープンするたび、テーブルがクラッシュしたというマークが付いているか、テーブルが正しくクローズしなかったかどうかをチェックしま

す。(最後のオプションは外部ロックを無効にして実行している場合のみ機能します。)このような場合、`mysqld` はテーブル上でチェックを実行します。テーブルが破損していた場合、`mysqld` は修復を試みます。

次のオプションは修復の動作方法に影響します。

オプション	説明
OFF	リカバリなし。
DEFAULT	バックアップ、強制、クイックチェックを行わないリカバリ。
BACKUP	データファイルがリカバリ中に変更された場合、 <code>tbl_name.MYD</code> ファイルのバックアップを <code>tbl_name-datetime.BAK</code> として保存します。
FORCE	<code>.MYD</code> ファイルから複数のレコードがなくなる場合でもリカバリを実行します。
QUICK	削除ブロックがない場合、テーブルの行をチェックしません。

サーバーがテーブルを自動的に修復する前に、サーバーは修復に関するメモをエラーログに書き込みます。ユーザーが介入せずにほとんどの問題をリカバリできるようにするには、`BACKUP,FORCE` オプションを使用します。これにより、一部の行が削除される場合でもテーブルの修復を強制しますが、古いデータファイルをバックアップとして保持しているため、何が発生したかをあとで検査できます。

[セクション16.2.1「MyISAM 起動オプション」](#)を参照してください。

- `myisam_repair_threads`

コマンド行形式	<code>--myisam-repair-threads=#</code>
システム変数	<code>myisam_repair_threads</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1
最小値	1
最大値 (64 ビットプラットフォーム)	18446744073709551615
最大値 (32 ビットプラットフォーム)	4294967295

この値が 1 より大きい場合、`MyISAM` テーブルインデックスは `Repair by sorting` プロセス中に並列で作成されます (各インデックスはインデックス独自のスレッド内)。デフォルト値は 1 です。

注記

マルチスレッド修復はまだ beta-quality コードです。

- `myisam_sort_buffer_size`

コマンド行形式	<code>--myisam-sort-buffer-size=#</code>
システム変数	<code>myisam_sort_buffer_size</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer

デフォルト値	8388608
最小値	4096
最大値 (Windows, 64 ビットプラットフォーム)	18446744073709551615
最大値 (Windows, 32 ビットプラットフォーム)	4294967295
最大値 (その他, 64 ビットプラットフォーム)	18446744073709551615
最大値 (その他, 32 ビットプラットフォーム)	4294967295

REPAIR TABLE 中に MyISAM インデックスをソートするときや、CREATE INDEX または ALTER TABLE を使用してインデックスを作成するときに割り当てられるバッファのサイズ。

- [myisam_stats_method](#)

コマンド行形式	--myisam-stats-method=name
システム変数	myisam_stats_method
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	nonnull_unequal
有効な値	nonnull_equal nonnull_unequal nonnull_ignored

MyISAM テーブルのインデックス値の分布に関する統計を収集するときに、サーバーが NULL 値を扱う方法。この変数は、nonnull_equal、nonnull_unequal、および nonnull_ignored の 3 つの値を指定できます。nonnull_equal の場合、すべての NULL インデックス値を同等として扱い、NULL 値の数とサイズが同等の単一値グループを生成します。nonnull_unequal の場合、NULL 値同士を同等として扱わず、それぞれの NULL はサイズが 1 の別個のグループを生成します。nonnull_ignored の場合、NULL 値は無視されます。

テーブル統計を生成するために使用する方法は、[セクション 8.3.8 「InnoDB および MyISAM インデックス統計コレクション」](#)に記載されているように、オプティマイザがクエリー実行のためのインデックスを選択する方法に影響を与えます。

- [myisam_use_mmap](#)

コマンド行形式	--myisam-use-mmap[={OFF ON}]
システム変数	myisam_use_mmap
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

MyISAM テーブルの読み取りおよび書き込みでメモリーマッピングを使用します。

- [mysql_native_password_proxy_users](#)

コマンド行形式	--mysql-native-password-proxy-users[={OFF ON}]
システム変数	mysql_native_password_proxy_users
スコープ	グローバル

動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

この変数は、`mysql_native_password` 組み込み認証プラグインがプロキシユーザーをサポートするかどうかを制御します。`check_proxy_users` システム変数が有効になっていないかぎり、効果はありません。ユーザープロキシの詳細は、[セクション6.2.18「プロキシユーザー」](#)を参照してください。

- [named_pipe](#)

コマンド行形式	<code>--named-pipe[={OFF ON}]</code>
システム変数	named_pipe
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
プラットフォーム固有	Windows
型	Boolean
デフォルト値	OFF

(Windows のみ。) サーバーが名前付きパイプでの接続をサポートしているかどうかを指定します。

- [named_pipe_full_access_group](#)

コマンド行形式	<code>--named-pipe-full-access-group=value</code>
導入	8.0.14
システム変数	named_pipe_full_access_group
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
プラットフォーム固有	Windows
型	文字列
デフォルト値	<code>*everyone*</code>
有効な値	<code>*everyone*</code> <code>empty string</code>

(Windows のみ。) MySQL サーバーによって作成された名前付きパイプ上のクライアントに付与されるアクセス制御は、[named_pipe](#) システム変数が名前付きパイプ接続をサポートできるようになっている場合に、正常に通信するために必要な最小値に設定されます。新しい MySQL クライアントソフトウェアでは、追加の構成なしで名前付きパイプ接続を開くことができますが、古いクライアントソフトウェアでは、名前付きパイプ接続を開くために完全なアクセスが必要になる場合があります。

この変数は、古い名前付きパイプクライアントを使用するための十分なアクセス権が MySQL サーバーによってメンバーに付与されている Windows ローカルグループの名前を設定します。初期状態では、この値はデフォルトで `*everyone*` に設定されており、古いクライアントがアップグレードされるまで、Windows 上の Everyone グループのユーザーは古いクライアントを引き続き使用できます。一方、値を空の文字列に設定すると、名前付きパイプへのフルアクセス権が Windows ユーザーに付与されなくなります。デフォルト値の `*everyone*` では、言語に依存しない方法で Windows の Everyone グループを参照できます。

新しい Windows ローカルグループ名 ([mysql_old_client_users](#) など) を Windows で作成し、古いクライアントソフトウェアへのアクセスが絶対に必要な場合にこの変数のデフォルト値を置き換えるために使用することが理想的で

す。この場合、グループのメンバーシップをできるだけ少ないユーザーに制限し、クライアントソフトウェアのアップグレード時にグループからユーザーを削除します。古い名前付きパイプクライアントを使用して MySQL への接続を開こうとするグループの非メンバーは、ユーザーが Windows 管理者によってグループに追加され、ログアウトしてログインするまでアクセスが拒否されます (Windows で必要)。

- [net_buffer_length](#)

コマンド行形式	<code>--net-buffer-length=#</code>
システム変数	net_buffer_length
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	16384
最小値	1024
最大値	1048576

各クライアントスレッドは、接続バッファおよび結果バッファに関連付けられています。両者は [net_buffer_length](#) で与えられたサイズで開始されますが、必要に応じて、[max_allowed_packet](#) バイトまで動的に拡大できます。結果バッファは、各 SQL ステートメントのあとで [net_buffer_length](#) に縮小されます。

この変数は通常は変更しませんが、メモリーが非常に少ない場合、クライアントによって送信される予想されるステートメントの長さに設定できます。ステートメントがこの長さを超えた場合、接続バッファは自動的に拡大されます。[net_buffer_length](#) の最大値は 1M バイトに設定できます。

この変数のセッションの値は、読み取り専用です。

- [net_read_timeout](#)

コマンド行形式	<code>--net-read-timeout=#</code>
システム変数	net_read_timeout
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	30
最小値	1

読み取りを中止する前に接続からのデータを待機する秒数。サーバーがクライアントからの読み込みを行うとき、[net_read_timeout](#) は中止するタイミングを制御するタイムアウト値です。サーバーがクライアントに書き込みを行うとき、[net_write_timeout](#) は中止するタイミングを制御するタイムアウト値です。[slave_net_timeout](#)も参照してください。

- [net_retry_count](#)

コマンド行形式	<code>--net-retry-count=#</code>
システム変数	net_retry_count
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	10

最小値	1
最大値 (64 ビットプラットフォーム)	18446744073709551615
最大値 (32 ビットプラットフォーム)	4294967295

通信ポートでの読み取りまたは書き込みが中断された場合、停止するまでこの回数だけ再試行します。FreeBSD では内部の中断がすべてのスレッドに送信されるため、この値をきわめて高く設定するようにしてください。

- [net_write_timeout](#)

コマンド行形式	<code>--net-write-timeout=#</code>
システム変数	net_write_timeout
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	60
最小値	1

書き込みを中止する前にブロックが接続に書き込まれるのを待機する秒数。 [net_read_timeout](#) も参照してください。

- [new](#)

コマンド行形式	<code>--new[={OFF ON}]</code>
システム変数	new
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
無効化	skip-new
型	Boolean
デフォルト値	OFF

この変数は、一部の 4.1 の動作をオンにするために MySQL 4.0 で使用されており、下位互換性のために保持されています。その値は常に OFF です。

NDB Cluster でこの変数を ON に設定すると、KEY または LINEAR KEY 以外のパーティショニングタイプを NDB テーブルとともに使用できるようになります。この機能は試験的なものであり、本番ではサポートされていません。追加情報については [ユーザー定義のパーティション分割と NDB ストレージエンジン \(NDB Cluster\)](#) を参照してください。

- [ngram_token_size](#)

コマンド行形式	<code>--ngram-token-size=#</code>
システム変数	ngram_token_size
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	2
最小値	1

最大値	10
-----	----

n-gram 全文パーサーの n-gram トークンサイズを定義します。 `ngram_token_size` オプションは読み取り専用で、起動時にのみ変更できます。デフォルト値は 2 (bigram) です。最大値は 10 です。

この変数の構成方法の詳細は、[セクション12.10.8「ngram 全文パーサー」](#) を参照してください。

- `offline_mode`

コマンド行形式	<code>--offline-mode[={OFF ON}]</code>
システム変数	<code>offline_mode</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

サーバーが「「オフラインモード」」内にあるかどうか。次の特性があります：

- `CONNECTION_ADMIN` 権限 (または非推奨の `SUPER` 権限) を持たない接続クライアントユーザーは、次のリクエストで切断され、適切なエラーが発生します。切断には、実行中のステートメントの終了およびロックの解放が含まれます。また、このようなクライアントは新しい接続を開始できず、適切なエラーを受け取ります。
- `CONNECTION_ADMIN` または `SUPER` 権限を持つ接続クライアントユーザーは切断されず、新しい接続を開始してサーバーを管理できます。
- レプリケーションスレッドは、サーバーへのデータの適用を継続できます。

オフラインモードを制御できるのは、`SYSTEM_VARIABLES_ADMIN` または `SUPER` 権限を持つユーザーのみです。サーバーをオフラインモードにするには、`offline_mode` システム変数の値を `OFF` から `ON` に変更します。通常の操作を再開するには、`offline_mode` を `ON` から `OFF` に変更します。オフラインモードでは、アクセスを拒否されたクライアントは `ER_SERVER_OFFLINE_MODE` エラーを受け取ります。

- `old`

コマンド行形式	<code>--old[={OFF ON}]</code>
システム変数	<code>old</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

`old` は互換性変数です。これはデフォルトでは無効化されていますが、以前のバージョンに存在した動作にサーバーを戻すために、起動時に有効にできます。

`old` を有効にすると、インデックスヒントのデフォルトの有効範囲が MySQL 5.1.17 より前に使用されていた有効範囲に変更されます。つまり、`FOR` 句を使用しないインデックスヒントは、インデックスが行の取得に使用する方法についてのみ適用され、`ORDER BY` 句または `GROUP BY` 句の解決には適用されません。([セクション8.9.4「インデックスヒント」](#) を参照してください。) レプリケーションのセットアップでこれを有効にする場合は注意してください。ステートメントベースのバイナリロギングでは、ソースとレプリカのモードが異なると、レプリケーションエラーが発生する可能性があります。

- `old_alter_table`

コマンド行形式	<code>--old-alter-table[={OFF ON}]</code>
---------	---

システム変数	old_alter_table
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

この変数を有効にすると、サーバーは [ALTER TABLE](#) 操作を処理する最適化された方法を使用しません。一時テーブルの使用に戻り、データのコピー後、MySQL 5.0 以前で使用されていたように、一時テーブルを元のテーブルの名前に変更します。[ALTER TABLE](#) の操作について詳しくは、[セクション13.1.9「ALTER TABLE ステートメント」](#)を参照してください。

[old_alter_table=ON](#) を使用した [ALTER TABLE ... DROP PARTITION](#) は、パーティションテーブルを再構築し、削除されたパーティションから互換性のある [PARTITION ... VALUES](#) 定義を持つ別のパーティションにデータを移動しようとします。別のパーティションに移動できないデータは削除されます。以前のリリースでは、[old_alter_table=ON](#) を使用した [ALTER TABLE ... DROP PARTITION](#) は、パーティションに格納されているデータを削除し、パーティションを削除していました。

- [open_files_limit](#)

コマンド行形式	<code>--open-files-limit=#</code>
システム変数	open_files_limit
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	5000, with possible adjustment
最小値	0
最大値	platform dependent

オペレーティングシステムから [mysqld](#) で使用可能なファイル記述子の数:

- 起動時に、[mysqld](#) は、この変数を直接設定するか、[mysqld_safe](#) の `--open-files-limit` オプションを使用して、リクエストされた値を使用して、`setrlimit()` でディスクリプタを予約します。[mysqld](#) でエラー `Too many open files` が生成された場合は、[open_files_limit](#) 値を増やしてみてください。内部的には、この変数の最大値は符号なし整数の最大値ですが、実際の最大値はプラットフォームに依存します。
- 実行時、[open_files_limit](#) の値は、オペレーティングシステムによって実際に [mysqld](#) に許可されるファイル記述子の数を示します。これは、起動時に要求される値とは異なる場合があります。起動時にリクエストされたファイル記述子の数を割り当てることができない場合、[mysqld](#) はエラーログに警告を書き込みます。

実際の [open_files_limit](#) の値は、システム起動時に指定された値 (ある場合) と、[max_connections](#) および [table_open_cache](#) の値に基づき、次の式を使用します。

- $10 + \text{max_connections} + (\text{table_open_cache} * 2)$
- $\text{max_connections} * 5$
- MySQL 8.0.19 以上: オペレーティングシステムの制限。

- MySQL 8.0.19 より前:

- オペレーティングシステムの制限 (その制限が正で、Infinity ではない場合)。
- オペレーティングシステムの制限が Infinity の場合: 起動時に指定された場合は `open_files_limit` 値、指定されていない場合は 5000。

サーバーは、それらの値の最大値を使用してファイル記述子の数を取得しようとします。この値は、符号なし整数の最大値に制限されます。そのような数の記述子を取得できない場合、サーバーはシステムが許可する数だけ取得しようとします。

MySQL がオープンファイルの数を変更できないシステムでは、有効な値は 0 です。

Unix では、`ulimit -n` コマンドで表示される値より大きい値を設定することはできません。

- `optimizer_prune_level`

コマンド行形式	<code>--optimizer-prune-level=#</code>
システム変数	<code>optimizer_prune_level</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Integer
デフォルト値	1
最小値	0
最大値	1

見込みのない部分的プランをオプティマイザ検索スペースから削除するために、クエリー最適化中に適用される経験則を制御します。値 0 は、オプティマイザが網羅的な検索を実行できるよう経験則を無効にします。値 1 は、中間プランによって取得された行の数に基づいて、オプティマイザにプランを削除させます。

- `optimizer_search_depth`

コマンド行形式	<code>--optimizer-search-depth=#</code>
システム変数	<code>optimizer_search_depth</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Integer
デフォルト値	62
最小値	0
最大値	62

クエリーオプティマイザによって実行される検索の最大の深さ。クエリー内の関係の数より値が大きいと、適切なクエリー計画が得られますが、クエリーの実行計画の生成に時間がかかります。クエリー内の関係の数より値が小さいと、実行プランがすばやく返されますが、結果のプランがまったく最適にならないことがあります。0 に設定された場合、システムは合理的な値を自動的に選択します。

- `optimizer_switch`

コマンド行形式	<code>--optimizer-switch=value</code>
システム変数	<code>optimizer_switch</code>
スコープ	グローバル、セッション

動的	はい
SET_VAR ヒントの適用	はい
型	Set
有効な値 (≥ 8.0.22)	batched_key_access={on off} block_nested_loop={on off} condition_fanout_filter={on off} derived_condition_pushdown={on off} derived_merge={on off} duplicateweedout={on off} engine_condition_pushdown={on off} firstmatch={on off} hash_join={on off} index_condition_pushdown={on off} index_merge={on off} index_merge_intersection={on off} index_merge_sort_union={on off} index_merge_union={on off} loosescan={on off} materialization={on off} mrr={on off} mrr_cost_based={on off} prefer_ordering_index={on off} semijoin={on off} skip_scan={on off} subquery_materialization_cost_based={on off} use_index_extensions={on off} use_invisible_indexes={on off}
有効な値 (≥ 8.0.21)	batched_key_access={on off} block_nested_loop={on off} condition_fanout_filter={on off} derived_merge={on off} duplicateweedout={on off} engine_condition_pushdown={on off}

	<p> firstmatch={on off} hash_join={on off} index_condition_pushdown={on off} index_merge={on off} index_merge_intersection={on off} index_merge_sort_union={on off} index_merge_union={on off} loosescan={on off} materialization={on off} mrr={on off} mrr_cost_based={on off} prefer_ordering_index={on off} semijoin={on off} skip_scan={on off} subquery_materialization_cost_based={on off} use_index_extensions={on off} use_invisible_indexes={on off} </p>
有効な値 (≥ 8.0.18)	<p> batched_key_access={on off} block_nested_loop={on off} condition_fanout_filter={on off} derived_merge={on off} duplicateweedout={on off} engine_condition_pushdown={on off} firstmatch={on off} hash_join={on off} index_condition_pushdown={on off} index_merge={on off} index_merge_intersection={on off} index_merge_sort_union={on off} index_merge_union={on off} loosescan={on off} materialization={on off} </p>

	mrr={on off} mrr_cost_based={on off} semijoin={on off} skip_scan={on off} subquery_materialization_cost_based={on off} use_index_extensions={on off} use_invisible_indexes={on off}
有効な値 (≥ 8.0.13)	batched_key_access={on off} block_nested_loop={on off} condition_fanout_filter={on off} derived_merge={on off} duplicateweedout={on off} engine_condition_pushdown={on off} firstmatch={on off} index_condition_pushdown={on off} index_merge={on off} index_merge_intersection={on off} index_merge_sort_union={on off} index_merge_union={on off} loosescan={on off} materialization={on off} mrr={on off} mrr_cost_based={on off} semijoin={on off} skip_scan={on off} subquery_materialization_cost_based={on off} use_index_extensions={on off} use_invisible_indexes={on off}
有効な値 (≤ 8.0.12)	batched_key_access={on off} block_nested_loop={on off} condition_fanout_filter={on off} derived_merge={on off} duplicateweedout={on off}

```

engine_condition_pushdown={on|off}
firstmatch={on|off}
index_condition_pushdown={on|off}
index_merge={on|off}
index_merge_intersection={on|off}
index_merge_sort_union={on|off}
index_merge_union={on|off}
loosescan={on|off}
materialization={on|off}
mrr={on|off}
mrr_cost_based={on|off}
semijoin={on|off}
subquery_materialization_cost_based={on|off}
use_index_extensions={on|off}
use_invisible_indexes={on|off}

```

`optimizer_switch` システム変数を使用するとオプティマイザの動作を制御できます。この変数の値はフラグのセットで、各フラグは対応するオプティマイザの動作の有効または無効を示す `on` または `off` を値を持ちます。この変数はグローバル値およびセッション値を持ち、実行時に変更できます。グローバル値のデフォルトはサーバーの起動時に設定できます。

オプティマイザの現在のフラグセットを表示するには、変数値を選択します。

```

mysql> SELECT @@optimizer_switch\G
***** 1. row *****
@@optimizer_switch: index_merge=on,index_merge_union=on,
index_merge_sort_union=on,index_merge_intersection=on,
engine_condition_pushdown=on,index_condition_pushdown=on,
mrr=on,mrr_cost_based=on,block_nested_loop=on,
batched_key_access=off,materialization=on,semijoin=on,
loosescan=on,firstmatch=on,duplicateweedout=on,
subquery_materialization_cost_based=on,
use_index_extensions=on,condition_fanout_filter=on,
derived_merge=on,use_invisible_indexes=off,skip_scan=on,
hash_join=on,subquery_to_derived=off,
prefer_ordering_index=on,hypergraph_optimizer=off,
derived_condition_pushdown=on

```

この変数の構文と、制御するオプティマイザの動作の詳細については、[セクション8.9.2「切り替え可能な最適化」](#)を参照してください。

- `optimizer_trace`

コマンド行形式	<code>--optimizer-trace=value</code>
システム変数	<code>optimizer_trace</code>
スコープ	グローバル、セッション
動的	はい
<code>SET_VAR</code> ヒントの適用	いいえ

型	文字列
---	-----

この変数はオプティマイザのトレースを制御します。詳細については、「[MySQL Internals: Tracing the Optimizer](#)」を参照してください。

- optimizer_trace_features

コマンド行形式	<code>--optimizer-trace-features=value</code>
システム変数	<code>optimizer_trace_features</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列

この変数は選択されたオプティマイザトレース機能を有効または無効にします。詳細については、「[MySQL Internals: Tracing the Optimizer](#)」を参照してください。

- optimizer_trace_limit

コマンド行形式	<code>--optimizer-trace-limit=#</code>
システム変数	<code>optimizer_trace_limit</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1

表示するオプティマイザトレースの最大数。詳細については、「[MySQL Internals: Tracing the Optimizer](#)」を参照してください。

- optimizer_trace_max_mem_size

コマンド行形式	<code>--optimizer-trace-max-mem-size=#</code>
システム変数	<code>optimizer_trace_max_mem_size</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Integer
デフォルト値	1048576

格納されるオプティマイザトレースの最大累積サイズ。詳細については、「[MySQL Internals: Tracing the Optimizer](#)」を参照してください。

- optimizer_trace_offset

コマンド行形式	<code>--optimizer-trace-offset=#</code>
システム変数	<code>optimizer_trace_offset</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer

デフォルト値	-1
--------	----

表示するオプティマイザトレースのオフセット。詳細については、「[MySQL Internals: Tracing the Optimizer](#)」を参照してください。

- [performance_schema_xxx](#)

パフォーマンススキーマのシステム変数は、[セクション27.15「パフォーマンススキーマシステム変数」](#)にリストされています。これらの変数は、パフォーマンススキーマ操作を構成するために使用されることもあります。

- [parser_max_mem_size](#)

コマンド行形式	<code>--parser-max-mem-size=#</code>
システム変数	parser_max_mem_size
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値 (64 ビットプラットフォーム)	18446744073709551615
デフォルト値 (32 ビットプラットフォーム)	4294967295
最小値	10000000
最大値 (64 ビットプラットフォーム)	18446744073709551615
最大値 (32 ビットプラットフォーム)	4294967295

パーサーで使用可能なメモリの最大量。デフォルト値では、使用可能なメモリーに制限はありません。この値を減らして、長い SQL ステートメントまたは複雑な SQL ステートメントの解析によって生じるメモリー不足の状況から保護できます。

- [partial_revokes](#)

コマンド行形式	<code>--partial-revokes[={OFF ON}]</code>
導入	8.0.16
システム変数	partial_revokes
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF (部分的な取消しが存在しない場合) ON (部分的な取消しが存在する場合)

この変数を有効にすると、権限を部分的に取り消すことができます。具体的には、グローバルレベルの権限を持つユーザーの場合、[partial_revokes](#) では、特定のスキーマの権限を取り消しながら、他のスキーマの権限をそのままにすることができます。たとえば、グローバル UPDATE 権限を持つユーザーは、[mysql](#) システムスキーマに対するこの権限の実行を制限できます。(または、別の方法で、ユーザーは [mysql](#) スキーマを除くすべてのスキーマに対して UPDATE 権限を実行できます。) この意味では、ユーザーのグローバル UPDATE 権限は部分的に取り消されます。

一度有効にすると、アカウントに権限制限がある場合は [partial_revokes](#) を無効にできません。そのようなアカウントが存在する場合、[partial_revokes](#) の無効化は失敗します:

- 起動時に [partial_revokes](#) を無効にしようとする、サーバーはエラーメッセージをログに記録し、[partial_revokes](#) を有効にします。

- 実行時に `partial_revokes` を無効にしようとすると、エラーが発生し、`partial_revokes` 値は変更されません。

この場合に `partial_revokes` を無効にするには、まず権限を再付与するか、アカウントを削除して、権限が部分的に取り消された各アカウントを変更します。

部分的な取消しを削除する手順などの詳細は、[セクション6.2.12「部分取消しを使用した権限の制限」](#) を参照してください。

- `password_history`

コマンド行形式	<code>--password-history=#</code>
システム変数	<code>password_history</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	4294967295

この変数は、必要な最小パスワード変更数に基づいて以前のパスワードの再利用を制御するグローバルポリシーを定義します。以前に使用されたアカウントパスワードの場合、この変数は、パスワードを再利用する前に発生する必要がある後続のアカウントパスワード変更の数を示します。値が 0 (デフォルト) の場合、パスワード変更の数に基づく再利用制限はありません。

この変数への変更は、`PASSWORD HISTORY DEFAULT` オプションで定義されたすべてのアカウントにただちに適用されます。

`CREATE USER` ステートメントおよび `ALTER USER` ステートメントの `PASSWORD HISTORY` オプションを使用して、個々のアカウントに対して必要に応じて変更のグローバル数のパスワード再利用ポリシーをオーバーライドできます。[セクション6.2.15「パスワード管理」](#) を参照してください。

- `password_require_current`

コマンド行形式	<code>--password-require-current[={OFF ON}]</code>
導入	8.0.13
システム変数	<code>password_require_current</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

この変数は、アカウントパスワードを変更しようとする際に、置換する現在のパスワードを指定する必要があるかどうかを制御するグローバルポリシーを定義します。

この変数への変更は、`PASSWORD REQUIRE CURRENT DEFAULT` オプションで定義されたすべてのアカウントにただちに適用されます。

グローバル検証必須ポリシーは、`CREATE USER` ステートメントおよび `ALTER USER` ステートメントの `PASSWORD REQUIRE` オプションを使用して、個々のアカウントに対して必要に応じてオーバーライドできます。[セクション6.2.15「パスワード管理」](#) を参照してください。

- [password_reuse_interval](#)

コマンド行形式	<code>--password-reuse-interval=#</code>
システム変数	password_reuse_interval
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	4294967295

この変数は、経過時間に基づいて以前のパスワードの再利用を制御するグローバルポリシーを定義します。以前に使用したアカウントパスワードの場合、この変数は、パスワードを再利用するまでに経過する必要がある日数を示します。値が 0 (デフォルト) の場合、経過時間に基づく再利用制限はありません。

この変数への変更は、[PASSWORD REUSE INTERVAL DEFAULT](#) オプションで定義されたすべてのアカウントにただちに適用されます。

グローバル経過時間パスワード再利用ポリシーは、[CREATE USER](#) ステートメントおよび [ALTER USER](#) ステートメントの [PASSWORD REUSE INTERVAL](#) オプションを使用して、個々のアカウントに対して必要に応じてオーバーライドできます。[セクション6.2.15「パスワード管理」](#)を参照してください。

- [persisted_globals_load](#)

コマンド行形式	<code>--persisted-globals-load[={OFF ON}]</code>
システム変数	persisted_globals_load
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

永続化された構成設定をデータディレクトリの `mysqld-auto.cnf` ファイルからロードするかどうか。サーバーは通常、このファイルを起動時にほかのすべてのオプションファイルのあとに処理します ([セクション4.2.2.2「オプションファイルの使用」](#)を参照)。[persisted_globals_load](#) を無効にすると、サーバーの起動シーケンスで `mysqld-auto.cnf` がスキップされます。

`mysqld-auto.cnf` の内容を変更するには、[SET PERSIST](#)、[SET PERSIST_ONLY](#) および [RESET PERSIST](#) ステートメントを使用します。[セクション5.1.9.3「永続化されるシステム変数」](#)を参照してください。

- [persist_only_admin_x509_subject](#)

コマンド行形式	<code>--persist-only-admin-x509-subject=string</code>
導入	8.0.14
システム変数	persist_only_admin_x509_subject
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

デフォルト値	empty string
--------	--------------

`SET PERSIST` および `SET PERSIST_ONLY` を使用すると、システム変数をデータディレクトリ内の `mysqld-auto.cnf` オプションファイルに永続化できます ([セクション13.7.6.1「変数代入の SET 構文」](#) を参照)。システム変数を永続化すると、後続のサーバーの再起動に影響する実行時構成の変更が可能になります。これは、MySQL サーバーのホストオプションファイルに直接アクセスする必要がないリモート管理に便利です。ただし、一部のシステム変数は永続的でないか、特定の制限条件下でのみ永続化できます。

`persist_only_admin_x509_subject` システム変数は、ユーザーが永続的に制限されたシステム変数を永続化できるようにするために必要な SSL 証明書の X.509 サブジェクト値を指定します。デフォルト値は空の文字列で、サブジェクトチェックを無効にして、永続的に制限されたシステム変数をユーザーが永続化できないようにします。

`persist_only_admin_x509_subject` が空でない場合、暗号化された接続を使用してサーバーに接続し、指定されたサブジェクト値で SSL 証明書を提供するユーザーは、`SET PERSIST_ONLY` を使用して永続制限付きシステム変数を永続化できます。永続制限付きシステム変数および `persist_only_admin_x509_subject` を有効にするように MySQL を構成する手順の詳細は、[セクション5.1.9.4「永続的で永続的に制限されないシステム変数」](#) を参照してください。

- `pid_file`

コマンド行形式	<code>--pid-file=file_name</code>
システム変数	<code>pid_file</code>
スコープ	グローバル
動的	いいえ
<code>SET_VAR</code> ヒントの適用	いいえ
型	ファイル名

サーバーがプロセス ID を書き込むファイルのパス名。サーバーは、別のディレクトリを指定する絶対パス名が指定されないかぎり、データディレクトリ内にファイルを作成します。この変数を指定する場合は、値を指定する必要があります。この変数を指定しない場合、MySQL はデフォルト値の `host_name.pid` を使用します。`host_name` はホストマシンの名前です。

プロセス ID ファイルは、`mysqld_safe` などの他のプログラムでサーバープロセス ID を決定するために使用されます。Windows では、この変数はデフォルトのエラーログファイル名にも影響します。[セクション5.4.2「エラーログ」](#) を参照してください。

- `plugin_dir`

コマンド行形式	<code>--plugin-dir=dir_name</code>
システム変数	<code>plugin_dir</code>
スコープ	グローバル
動的	いいえ
<code>SET_VAR</code> ヒントの適用	いいえ
型	ディレクトリ名
デフォルト値	<code>BASEDIR/lib/plugin</code>

プラグインディレクトリのパス名。

プラグインディレクトリがサーバーによって書き込み可能な場合、ユーザーは `SELECT ... INTO DUMPFILE` を使用して、ディレクトリ内のファイルに実行可能コードを書き込むことができます。これを回避するには、`plugin_dir` をサーバーに対して読み取り専用にするか、`SELECT` 書き込みを安全に行うことができるディレクトリに `secure_file_priv` を設定します。

- `port`

コマンド行形式	<code>--port=port_num</code>
---------	------------------------------

システム変数	port
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	3306
最小値	0
最大値	65535

サーバーが TCP/IP 接続を listen するポートの数。この変数は、`--port` オプションで設定できます。

- [preload_buffer_size](#)

コマンド行形式	<code>--preload-buffer-size=#</code>
システム変数	preload_buffer_size
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	32768
最小値	1024
最大値	1073741824

インデックスをプリロードしたときに割り当てられるバッファのサイズ。

- [print_identified_with_as_hex](#)

コマンド行形式	<code>--print-identified-with-as-hex[={OFF ON}]</code>
導入	8.0.17
システム変数	print_identified_with_as_hex
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

`SHOW CREATE USER` からの出力の `IDENTIFIED WITH` 句に表示されるパスワードハッシュ値には、端末表示やその他の環境に悪影響を与える印刷不可能な文字が含まれている可能性があります。[print_identified_with_as_hex](#) を有効にすると、`SHOW CREATE USER` では、通常の文字列リテラルではなく 16 進数文字列などのハッシュ値が表示されます。印刷できない文字を含まないハッシュ値は、この変数が有効になっていても、通常の文字列リテラルとして表示されます。

- [profiling](#)

0 または `OFF` (デフォルト) に設定した場合、ステートメントのプロファイリングは無効になります。1 または `ON` に設定した場合、ステートメントのプロファイリングは有効になり、`SHOW PROFILE` および `SHOW PROFILES` ステートメントはプロファイリング情報へのアクセスを提供します。[セクション13.7.7.31「SHOW PROFILES ステートメント」](#)を参照してください。

この変数は非推奨です。将来の MySQL リリースで削除される予定です。

- [profiling_history_size](#)

`profiling` が有効な場合にプロファイリング情報を保持する対象となるステートメントの数。デフォルト値は 15 です。最大値は 100 です。値を 0 に設定すると、プロファイリングは実質的に無効になります。 [セクション 13.7.7.31 「SHOW PROFILES ステートメント」](#) を参照してください。

この変数は非推奨です。将来の MySQL リリースで削除される予定です。

- `protocol_compression_algorithms`

コマンド行形式	<code>--protocol-compression-algorithms=value</code>
導入	8.0.18
システム変数	<code>protocol_compression_algorithms</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Set
デフォルト値	<code>zlib,zstd,uncompressed</code>
有効な値	<code>zlib</code> <code>zstd</code> <code>uncompressed</code>

サーバーが受信接続に対して許可する圧縮アルゴリズム。これには、クライアントプログラムおよびソース/レプリカレプリケーションまたはグループレプリケーションに参加しているサーバーによる接続が含まれます。圧縮は `FEDERATED` テーブルの接続には適用されません。

`protocol_compression_algorithms` は、X プロトコルの接続圧縮を制御しません。この動作の詳細は、 [セクション 20.5.5 「X プラグイン での接続圧縮」](#) を参照してください。

変数値は、次の項目から任意の順序でカンマ区切りの圧縮アルゴリズム名のリストです (大/小文字は区別されません):

- `zlib`: `zlib` 圧縮アルゴリズムを使用する接続を許可します。
- `zstd`: `zstd` 圧縮アルゴリズム (`zstd 1.3`) を使用する接続を許可します。
- `uncompressed`: 圧縮解除された接続を許可します。このアルゴリズム名が `protocol_compression_algorithms` 値に含まれていない場合、サーバーは圧縮されていない接続を許可しません。値に指定されている他のアルゴリズムを使用する圧縮接続のみが許可され、圧縮されていない接続へのフォールバックはありません。

`zlib,zstd,uncompressed` のデフォルト値は、サーバーがすべての圧縮アルゴリズムを許可することを示します。

詳細は、 [セクション 4.2.8 「接続圧縮制御」](#) を参照してください。

- `protocol_version`

システム変数	<code>protocol_version</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer

MySQL Server によって使用されるクライアント/サーバープロトコルのバージョン。

- `proxy_user`

システム変数	<code>proxy_user</code>
スコープ	セッション
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

現在のクライアントが別のユーザーのプロキシの場合、この変数はプロキシユーザーのアカウント名です。そうでない場合、この変数は `NULL` です。セクション6.2.18「プロキシユーザー」を参照してください。

- `pseudo_slave_mode`

システム変数	<code>pseudo_slave_mode</code>
スコープ	セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean

このシステム変数は、内部サーバー用です。`pseudo_slave_mode` は、現在処理しているサーバーより古いサーバーまたは新しいサーバーで発生したトランザクションの正しい処理を支援します。`mysqlbinlog` は、SQL ステートメントを実行する前に `pseudo_slave_mode` の値を `true` に設定します。

このシステム変数のセッション値の設定は制限された操作です。セッションユーザーには、`REPLICATION_APPLIER` 権限 (セクション17.3.3「レプリケーション権限チェック」を参照) または制限付きセッション変数の設定に十分な権限 (セクション5.1.9.1「システム変数権限」を参照) が必要です。ただし、この変数はユーザーが設定するためのものではなく、レプリケーションインフラストラクチャによって自動的に設定されることに注意してください。

`pseudo_slave_mode` は、準備された XA トランザクションの処理に次の影響を与えます。これらのトランザクションは、処理セッション (デフォルトでは、`XA START` を発行するセッション) に対して連結または連結解除できます:

- `true` で、処理セッションが内部使用 `BINLOG` ステートメントを実行した場合、XA トランザクションは、`XA PREPARE` までのトランザクションの最初の部分が終了するとすぐにセッションから自動的にデタッチされるため、`XA RECOVER_ADMIN` 権限を持つセッションでコミットまたはロールバックできます。
- `false` の場合、XA トランザクションは、そのセッションが存続しているかぎり処理セッションにアタッチされたままになり、その間、他のセッションはトランザクションをコミットできません。準備されたトランザクションは、セッションが切断されるか、サーバーが再起動した場合にのみデタッチされます。

`pseudo_slave_mode` は、`original_commit_timestamp` レプリケーション遅延タイムスタンプおよび `original_server_version` システム変数に次の影響を与えます:

- `true` の場合、`original_commit_timestamp` または `original_server_version` を明示的に設定しないトランザクションは別の不明なサーバーで発生しているとみなされるため、値 0(不明) がタイムスタンプとシステム変数の両方に割り当てられます。
- `false` の場合、`original_commit_timestamp` または `original_server_version` を明示的に設定しないトランザクションは現在のサーバーで発生しているとみなされるため、現在のタイムスタンプと現在のサーバーバージョンがタイムスタンプとシステム変数に割り当てられます。

MySQL 8.0.14 以降では、`pseudo_slave_mode` は、サポートされていない (削除または不明な) SQL モードを設定するステートメントの処理に次の影響を与えます:

- `true` の場合、サーバーはサポートされていないモードを無視し、警告を生成します。
- `false` の場合、サーバーは `ER_UNSUPPORTED_SQL_MODE` でステートメントを拒否します。

- [pseudo_thread_id](#)

システム変数	pseudo_thread_id
スコープ	セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer

この変数は内部サーバーで使用します。

警告

[pseudo_thread_id](#) システム変数のセッション値を変更すると、[CONNECTION_ID\(\)](#) 関数によって返される値が変更されます。

MySQL 8.0.14 では、このシステム変数のセッション値の設定は制限された操作です。セッションユーザーには、制限付きセッション変数を設定するのに十分な権限が必要です。 [セクション5.1.9.1「システム変数権限」](#)を参照してください。

- [query_alloc_block_size](#)

コマンド行形式	<code>--query-alloc-block-size=#</code>
システム変数	query_alloc_block_size
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	8192
最小値	1024
最大値	4294967295
単位	bytes
ブロックサイズ	1024

ステートメントの解析および実行中に作成されたオブジェクトに割り当てられるメモリーブロックの割当てサイズ(バイト)。メモリーのフラグメント化について問題がある場合、このパラメータを増やすと役立つ場合があります。

- [query_prealloc_size](#)

コマンド行形式	<code>--query-prealloc-size=#</code>
システム変数	query_prealloc_size
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	8192
最小値	8192
最大値 (64 ビットプラットフォーム)	18446744073709551615
最大値 (32 ビットプラットフォーム)	4294967295

ブロックサイズ	1024
---------	------

ステートメントの解析および実行に使用される永続バッファのサイズ (バイト単位)。このバッファは、ステートメント間で解放されません。複雑なクエリーを発行する場合、`query_prealloc_size` の値を大きくすると、クエリー実行操作時にサーバーがメモリー割り当てを実行する必要性が低くなるため、パフォーマンスの向上に役立つ場合があります。

- [rand_seed1](#)

システム変数	rand_seed1
スコープ	セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer

[rand_seed1](#) および [rand_seed2](#) 変数は、セッション変数としてのみ存在し、設定はできますが読み取ることはできません。変数は `SHOW VARIABLES` の出力に表示されますが、その値は表示されません。

これらの変数の目的は、`RAND()` 関数のレプリケーションをサポートすることです。`RAND()` を起動するステートメントの場合、ソースはレプリカに 2 つの値を渡し、そこでランダム番号ジェネレータのシードに使用されます。レプリカは、これらの値を使用して、レプリカ上の `RAND()` がソースと同じ値を生成するように、[rand_seed1](#) および [rand_seed2](#) のセッション変数を設定します。

- [rand_seed2](#)

[rand_seed1](#) の説明を参照してください。

- [range_alloc_block_size](#)

コマンド行形式	<code>--range-alloc-block-size=#</code>
システム変数	range_alloc_block_size
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Integer
デフォルト値	4096
最小値	4096
最大値 (64 ビットプラットフォーム)	18446744073709547520
最大値	4294967295
ブロックサイズ	1024

範囲最適化の実行時に割り当てられるブロックのサイズ (バイト単位)。

- [range_optimizer_max_mem_size](#)

コマンド行形式	<code>--range-optimizer-max-mem-size=#</code>
システム変数	range_optimizer_max_mem_size
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	8388608

最小値	0
最大値	18446744073709551615

範囲最適化のメモリ消費の制限。値 0 は「制限なし」を表します。最適化によって考慮される実行計画で範囲アクセス方法が使用されているが、最適化はこの方法に必要なメモリ量が制限を超えると見積もった場合、計画を破棄し、他の計画を考慮します。詳細は、[範囲最適化のためのメモリ使用の制限](#)を参照してください。

- [rbr_exec_mode](#)

システム変数	rbr_exec_mode
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	STRICT
有効な値	IDEMPOTENT STRICT

[mysqlbinlog](#) による内部使用。この変数は、サーバーを IDEMPOTENT モードと STRICT モードの間で切り替えます。IDEMPOTENT モードでは、[mysqlbinlog](#) によって生成された BINLOG ステートメントで重複キーおよびキーのないエラーが抑制されます。このモードは、既存のデータと競合する原因となるサーバー上で行ベースのバイナリログを再生する場合に役立ちます。[mysqlbinlog](#) では、出力に次のように記述して `--idempotent` オプションを指定すると、このモードが設定されます:

```
SET SESSION RBR_EXEC_MODE=IDEMPOTENT;
```

MySQL 8.0.18 では、このシステム変数のセッション値の設定は制限付き操作ではなくなりました。

- [read_buffer_size](#)

コマンド行形式	<code>--read-buffer-size=#</code>
システム変数	read_buffer_size
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Integer
デフォルト値	131072
最小値	8192
最大値	2147479552

MyISAM テーブルの順次スキャンを実行する各スレッドは、スキャンする各テーブルにこのサイズ (バイト単位) のバッファを割り当てます。多くの順次スキャンを実行する場合、この値を増やした方がよい場合もあり、デフォルトは 131072 です。この変数の値は 4K バイトの倍数にしてください。4KB の倍数ではない値に設定すると、その値は 4KB の最も近い倍数に切り捨てられます。

このオプションは、すべてのストレージエンジンの次のコンテキストでも使用されます:

- `ORDER BY` で行をソートするとき、インデックスを一時ファイル (一時テーブルではない) にキャッシュする場合。
- パーティションに一括挿入する場合。

- ネストされたクエリーの結果をキャッシュする場合。

`read_buffer_size` は、他のストレージエンジン固有の方法でも使用されます: `MEMORY` テーブルのメモリーブロックサイズを決定します。

MySQL 8.0.22 以降、`SELECT INTO DUMPFILE` および `SELECT INTO OUTFILE` ステートメントの実行時に使用されるバッファの `read_buffer_size` の値のかわりに `select_into_buffer_size` の値が使用されます。

さまざまな操作中でのメモリー使用についての詳細は、[セクション8.12.3.1「MySQL のメモリーの使用方法」](#)を参照してください。

- `read_only`

コマンド行形式	<code>--read-only[={OFF ON}]</code>
システム変数	<code>read_only</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

`read_only` システム変数が有効になっている場合、サーバーは、`CONNECTION_ADMIN` 権限 (または非推奨の `SUPER` 権限) を持つユーザー以外のクライアント更新を許可しません。この変数はデフォルトでは無効になっています。

サーバーは `super_read_only` システム変数 (デフォルトでは無効) もサポートしていますが、これには次の効果があります:

- `super_read_only` が有効になっている場合、`CONNECTION_ADMIN` または `SUPER` 権限を持つユーザーであっても、サーバーはクライアントの更新を禁止します。
- `super_read_only` を `ON` に設定すると、`read_only` は暗黙的に `ON` に強制されます。
- `read_only` を `OFF` に設定すると、`super_read_only` は暗黙的に `OFF` に強制されます。

`read_only` が有効な場合でも、サーバーは次の操作を許可します:

- サーバーがレプリカの場合、レプリケーションスレッドによって実行される更新。レプリケーション設定では、レプリカサーバー上の `read_only` を有効にして、レプリカがクライアントからではなくレプリケーションソースサーバーからの更新のみを受け入れるようにすると便利です。
- 現在のバイナリログファイルに存在しない実行されたトランザクションの GTID を格納するシステムテーブル `mysql.gtid_executed` に書き込みます。
- `ANALYZE TABLE` ステートメントまたは `OPTIMIZE TABLE` ステートメントの使用。読取り専用モードの目的は、テーブルの構造または内容の変更を防ぐことです。分析および最適化は、そのような変更の条件を備えていません。これは、たとえば、読取り専用レプリカに対する整合性チェックを `mysqlcheck --all-databases --analyze` で実行できることを意味します。
- `TEMPORARY` テーブルに対する操作。
- ログテーブル (`mysql.general_log` および `mysql.slow_log`) に挿入します。[セクション5.4.1「一般クエリーログおよびスロークエリーログの出力先の選択」](#)を参照してください。

- `UPDATE` または `TRUNCATE TABLE` 操作などの「パフォーマンススキーマ」テーブルの更新。

レプリケーションソースサーバー上の `read_only` への変更は、複製サーバーにレプリケートされません。この値は、ソースの設定とは無関係にレプリカサーバーに設定できます。

`read_only` の有効化の試行 (`super_read_only` の有効化による暗黙的な試行を含む) には、次の条件が適用されます:

- 試行は失敗し、(`LOCK TABLES` で取得した) 明示的なロックがあるか、保留中のトランザクションがある場合はエラーが発生します。
 - ロックが解放されてステートメントおよびトランザクションが終了するまで、他のクライアントに進行中のステートメント、アクティブな `LOCK TABLES WRITE` または進行中のコミットがある間、試行はブロックされます。`read_only` の有効化の試行が保留されているとき、ほかのクライアントによるテーブルロックあるいはトランザクションの開始のリクエストもまた `read_only` が設定されるまでブロックされます。
 - メタデータロックを保持するアクティブなトランザクションがある場合、そのトランザクションが終了するまで試行はブロックされます。
 - グローバル読み取りロック (`FLUSH TABLES WITH READ LOCK` で取得) にはテーブルロックが含まれていないため、`read_only` を有効化できます。
- `read_rnd_buffer_size`

コマンド行形式	<code>--read-rnd-buffer-size=#</code>
システム変数	<code>read_rnd_buffer_size</code>
スコープ	グローバル、セッション
動的	はい
<code>SET_VAR</code> ヒントの適用	はい
型	Integer
デフォルト値	262144
最小値	1
最大値	2147483647

この変数は、`MyISAM` テーブルからの読み取り、ストレージエンジン、および `Multi-Range Read` の最適化のために使用されます。

キーソート操作のあとで、`MyISAM` テーブルの行をソート順に読み取るとき、ディスクシークを回避するため行はこのバッファから読み取られます。 [セクション8.2.1.16「ORDER BY の最適化」](#) を参照してください。この変数を大きい値に設定すると、`ORDER BY` のパフォーマンスを大幅に向上できます。ただし、これは各クライアントに割り当てられるバッファであるため、グローバル変数を大きい値に設定しないでください。代わりに、大規模なクエリーを実行する必要があるクライアント内からのみセッション変数を変更します。

さまざまな操作中でのメモリー使用についての詳細は、 [セクション8.12.3.1「MySQL のメモリーの使用方法」](#) を参照してください。 `Multi-Range Read` の最適化については、 [セクション8.2.1.11「Multi-Range Read の最適化」](#) を参照してください。

- `regexp_stack_limit`

コマンド行形式	<code>--regexp-stack-limit=#</code>
システム変数	<code>regexp_stack_limit</code>
スコープ	グローバル
動的	はい
<code>SET_VAR</code> ヒントの適用	いいえ
型	Integer
デフォルト値	8000000

最小値	0
最大値	2147483647

REGEXP_LIKE() および同様の関数によって実行される正規表現照合操作に使用される内部スタックの最大使用可能メモリー (バイト)。セクション12.8.2「正規表現」を参照してください。

- [regexp_time_limit](#)

コマンド行形式	--regexp-time-limit=#
システム変数	regexp_time_limit
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	32
最小値	0
最大値	2147483647

REGEXP_LIKE() および同様の関数によって実行される正規表現照合操作の時間制限 (セクション12.8.2「正規表現」を参照)。この制限は、照合エンジンによって実行されるステップの最大許容数として表されるため、実行時間に間接的にのみ影響します。通常はミリ秒の順序で表示されます。

- [require_row_format](#)

導入	8.0.19
システム変数	require_row_format
スコープ	セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

この変数は、レプリケーションおよび [mysqlbinlog](#) による内部サーバーの使用に使用されます。セッションで実行される DML イベントは、行ベースのバイナリロギング形式でエンコードされたイベントのみに制限され、一時テーブルは作成できません。制限に従わないクエリーは失敗します。

このシステム変数のセッション値を ON に設定する場合、権限は必要ありません。このシステム変数のセッション値を OFF に設定することは制限された操作であり、セッションユーザーには制限されたセッション変数を設定するのに十分な権限が必要です。セクション5.1.9.1「システム変数権限」を参照してください。

- [require_secure_transport](#)

コマンド行形式	--require-secure-transport[={OFF ON}]
システム変数	require_secure_transport
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean

デフォルト値	OFF
--------	-----

なんらかの形式のセキュアトランスポートを使用するためにサーバーへのクライアント接続が必要かどうか。この変数が有効な場合、サーバーは TLS/SSL を使用して暗号化された TCP/IP 接続、またはソケットファイル (Unix の場合) または共有メモリー (Windows の場合) を使用する接続のみを許可します。サーバーはセキュアでない接続試行を拒否し、`ER_SECURE_TRANSPORT_REQUIRED` エラーで失敗します。

この機能は、優先されるアカウントごとの SSL 要件を補完します。たとえば、アカウントが `REQUIRE SSL` で定義されている場合、`require_secure_transport` を有効にしても、そのアカウントを使用して Unix ソケットファイルを使用して接続することはできません。

サーバーでセキュアなトランスポートを使用できない場合があります。たとえば、Windows 上のサーバーは、SSL 証明書またはキーファイルを指定せずに `shared_memory` システム変数を無効にして起動した場合、セキュアなトランスポートをサポートしません。これらの条件下で、起動時に `require_secure_transport` を有効にしようとする、サーバーはエラーログにメッセージを書き込み、終了します。実行時に変数を有効にしようとする、`ER_NO_SECURE_TRANSPORTS_CONFIGURED` エラーで失敗します。

[暗号化された接続の必須としての構成](#)も参照してください。

- [resultset_metadata](#)

システム変数	resultset_metadata
スコープ	セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	FULL
有効な値	FULL NONE

メタデータ転送がオプションの接続の場合、クライアントは `resultset_metadata` システム変数を設定して、サーバーが結果セットのメタデータを返すかどうかを制御します。許可される値は、`FULL` (すべてのメタデータを返します。これがデフォルトです) および `NONE` (メタデータを返しません) です。

メタデータ以外の接続の場合、`resultset_metadata` を `NONE` に設定するとエラーが発生します。

結果セットのメタデータ転送の管理の詳細は、[Optional Result Set Metadata](#) を参照してください。

- [secondary_engine_cost_threshold](#)

導入	8.0.16
システム変数	secondary_engine_cost_threshold
スコープ	セッション
動的	はい
SET_VAR ヒントの適用	はい
型	数値
デフォルト値	100000.000000
最小値	0
最大値	<code>DBL_MAX</code> (maximum double value)

セカンダリエンジンへのクエリーオフロードのオプティマイザコストしきい値。

HeatWave で使用します。[MySQL HeatWave User Guide](#) を参照してください。

- [schema_definition_cache](#)

コマンド行形式	--schema-definition-cache=#
システム変数	schema_definition_cache
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	256
最小値	256
最大値	524288

ディクショナリオブジェクトキャッシュに保持できるスキーマ定義オブジェクト (使用済と未使用の両方) の数の制限を定義します。

未使用のスキーマ定義オブジェクトは、使用中の数が [schema_definition_cache](#) で定義されている容量より少ない場合にのみディクショナリオブジェクトキャッシュに保持されます。

0 の設定は、スキーマ定義オブジェクトが使用中にディクショナリオブジェクトキャッシュにのみ保持されることを意味します。

詳細は、[セクション14.4「ディクショナリオブジェクトキャッシュ」](#)を参照してください。

- [secure_file_priv](#)

コマンド行形式	--secure-file-priv=dir_name
システム変数	secure_file_priv
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	platform specific
有効な値	empty string dirname NULL

この変数は、[LOAD DATA](#) ステートメント、[SELECT ... INTO OUTFILE](#) ステートメントおよび [LOAD_FILE\(\)](#) 関数によって実行される操作など、データインポートおよびエクスポート操作の影響を制限するために使用されます。これらの操作は、[FILE](#) 権限を持つユーザーにのみ許可されます。

[secure_file_priv](#) は、次のように設定できます:

- 空の場合、変数は無効です。これはセキュアな設定ではありません。
- ディレクトリの名前に設定すると、サーバーはインポートおよびエクスポート操作をそのディレクトリ内のファイルでのみ機能するように制限します。ディレクトリが存在する必要があります。サーバーは作成しません。

- `NULL` に設定されている場合、サーバーはインポートおよびエクスポート操作を無効にします。

次のテーブルに示すように、デフォルト値はプラットフォーム固有であり、`INSTALL_LAYOUT` CMake オプションの値によって異なります。ソースからビルドする場合にデフォルトの `secure_file_priv` 値を明示的に指定するには、`INSTALL_SECURE_FILE_PRIVDIR` CMake オプションを使用します。

<code>INSTALL_LAYOUT</code> 値	デフォルトの <code>secure_file_priv</code> 値
<code>STANDALONE</code>	empty
<code>DEB, RPM, SVR4</code>	<code>/var/lib/mysql-files</code>
それ以外の場合	<code>CMAKE_INSTALL_PREFIX</code> 値の下の <code>mysql-files</code>

サーバーは起動時に `secure_file_priv` の値をチェックし、値がセキュアでない場合はエラーログに警告を書き込みます。`NULL` 以外の値は、空の場合、値がデータディレクトリまたはそのサブディレクトリである場合、あるいはすべてのユーザーがアクセスできるディレクトリである場合、セキュアでないといみなされます。`secure_file_priv` が存在しないパスに設定されている場合、サーバーはエラーログにエラーメッセージを書き込み、終了します。

- `select_into_buffer_size`

コマンド行形式	<code>--select-into-buffer-size=#</code>
導入	8.0.22
システム変数	<code>select_into_buffer_size</code>
スコープ	グローバル、セッション
動的	はい
<code>SET_VAR</code> ヒントの適用	はい
型	Integer
デフォルト値	131072
最小値	8192
最大値	2147479552
単位	bytes

`SELECT INTO OUTFILE` または `SELECT INTO DUMPFILE` を使用して、バックアップの作成、データ移行またはその他の目的でデータを 1 つ以上のファイルにダンプする場合、書き込みをバッファリングしてから、ディスクまたは他のストレージデバイスへの大量の書き込み I/O アクティビティのバーストをトリガーし、待機時間の影響を受けやすい他のクエリーを停止することがよくあります。この変数を使用すると、ストレージデバイスにデータを書き込むために使用されるバッファのサイズを制御して、バッファの同期が発生するタイミングを決定できるため、記述した種類の書き込みストールが発生しないようにできます。

`select_into_buffer_size` は、`read_buffer_size` に設定された値をオーバーライドします。(`select_into_buffer_size` と `read_buffer_size` のデフォルト値、最大値および最小値は同じです。) `select_into_disk_sync_delay` を使用して、後で同期が発生するたびに監視されるタイムアウトを設定することもできます。

- `select_into_disk_sync`

コマンド行形式	<code>--select-into-disk-sync={ON OFF}</code>
導入	8.0.22
システム変数	<code>select_into_disk_sync</code>
スコープ	グローバル、セッション
動的	はい
<code>SET_VAR</code> ヒントの適用	はい
型	Boolean
デフォルト値	OFF

有効な値	OFF ON
------	-----------

ON で設定すると、`select_into_buffer_size` を使用して長時間実行される `SELECT INTO OUTFILE` ステートメントまたは `SELECT INTO DUMPFILE` ステートメントによる出力ファイルへの書き込みのバッファ同期が有効になります。

- `select_into_disk_sync_delay`

コマンド行形式	<code>--select-into-disk-sync-delay=#</code>
導入	8.0.22
システム変数	<code>select_into_disk_sync_delay</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Integer
デフォルト値	0
最小値	0
最大値	31536000
単位	milliseconds

長時間実行される `SELECT INTO OUTFILE` ステートメントまたは `SELECT INTO DUMPFILE` ステートメントによる出力ファイルへの書き込みのバッファ同期が `select_into_disk_sync` で有効になっている場合、この変数は同期後のオプションの遅延 (ミリ秒) を設定します。0 (デフォルト) は遅延がないことを意味します。

- `session_track_gtids`

コマンド行形式	<code>--session-track-gtids=value</code>
システム変数	<code>session_track_gtids</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	OFF
有効な値	OFF OWN_GTID ALL_GTIDS

サーバーが GTID をクライアントに返すかどうかを制御し、クライアントが GTID を使用してサーバーの状態を追跡できるようにします。変数値に応じて、各トランザクションの実行の終了時に、サーバーの GTID が取得され、確認の一環としてクライアントに返されます。`session_track_gtids` に使用可能な値は次のとおりです:

- **OFF:** サーバーは GTID をクライアントに返しません。これはデフォルトです。
- **OWN_GTID:** サーバーは、最後の確認応答以降に現在のセッションでこのクライアントによって正常にコミットされたすべてのトランザクションの GTID を返します。通常、これは最後にコミットされたトランザクションの単一 GTID ですが、単一のクライアント要求によって複数のトランザクションが発生した場合、サーバーは関連する GTID をすべて含む GTID セットを返します。
- **ALL_GTIDS:** サーバーは、トランザクションが正常にコミットされた時点で読み取る `gtid_executed` システム変数のグローバル値を返します。この GTID セットには、コミットされたばかりのトランザクションの GTID だけで

なく、任意のクライアントによってサーバー上でコミットされたすべてのトランザクションが含まれ、現在確認されているトランザクションがコミットされた時点以降にコミットされたトランザクションを含めることができません。

`session_track_gtids` はトランザクションコンテキスト内で設定できません。

セッションステートトラッキングの詳細は、[セクション5.1.18「クライアントセッション状態の変更のサーバートラッキング」](#)を参照してください。

- `session_track_schema`

コマンド行形式	<code>--session-track-schema[={OFF ON}]</code>
システム変数	<code>session_track_schema</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

現在のセッション内でデフォルトスキーマ (データベース) が設定されたときにサーバーが追跡し、スキーマ名を使用可能にするようクライアントに通知するかどうかを制御します。

スキーマ名トラッカが有効な場合、新しいスキーマ名が古いスキーマ名と同じであっても、デフォルトスキーマが設定されるたびに名前通知が発生します。

セッションステートトラッキングの詳細は、[セクション5.1.18「クライアントセッション状態の変更のサーバートラッキング」](#)を参照してください。

- `session_track_state_change`

コマンド行形式	<code>--session-track-state-change[={OFF ON}]</code>
システム変数	<code>session_track_state_change</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

現在のセッションの状態に対する変更をサーバーが追跡し、状態の変更が発生したときにクライアントに通知するかどうかを制御します。クライアントセッションステートの次の属性について変更をレポートできます:

- デフォルトスキーマ (データベース)。
- システム変数のセッション固有の値。
- ユーザー定義変数。
- 一時テーブル
- プリベアドステートメント。

セッション状態トラッカが有効になっている場合、新しい属性値が古い属性値と同じであっても、追跡対象のセッション属性を含む変更ごとに通知が発生します。たとえば、ユーザー定義変数を現在の値に設定すると、通知されます。

`session_track_state_change` 変数は、変更が発生した場合の通知のみを制御し、変更内容は制御しません。たとえば、状態変更通知は、デフォルトスキーマが設定されている場合や追跡されているセッションシステ

ム変数が割り当てられている場合に発生しますが、通知にはスキーマ名または変数値は含まれません。スキーマ名またはセッションシステム変数の値の通知を受信するには、それぞれ [session_track_schema](#) または [session_track_system_variables](#) システム変数を使用します。

注記

[session_track_state_change](#) 自体に値を割り当てることは状態変更とはみなされず、そのようにはレポートされません。ただし、その名前が [session_track_system_variables](#) の値にリストされている場合、その名前への割当てによって新しい値が通知されます。

セッションステートトラッキングの詳細は、[セクション5.1.18「クライアントセッション状態の変更のサーバートラッキング」](#) を参照してください。

- [session_track_system_variables](#)

コマンド行形式	<code>--session-track-system-variables=#</code>
システム変数	session_track_system_variables
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	time_zone , autocommit , character_set_client , character_set_results , character_set_connection

サーバーがセッションシステム変数への割当てを追跡し、割り当てられた各変数の名前と値をクライアントに通知するかどうかを制御します。変数値は、割当てを追跡する変数のカンマ区切りリストです。デフォルトでは、通知は [time_zone](#), [autocommit](#), [character_set_client](#), [character_set_results](#) および [character_set_connection](#) に対して有効になっています。(後者の3つの変数は、[SET NAMES](#) の影響を受ける変数です。)

* という特別な値を指定すると、サーバーはすべてのセッション変数への割当てを追跡します。指定する場合、この値は特定のシステム変数名なしで単独で指定する必要があります。

セッション変数割当ての通知を無効にするには、[session_track_system_variables](#) を空の文字列に設定します。

セッションシステム変数の追跡が有効になっている場合、新しい値が古い値と同じであっても、追跡対象のセッション変数へのすべての割当てに対して通知が行われます。

セッションステートトラッキングの詳細は、[セクション5.1.18「クライアントセッション状態の変更のサーバートラッキング」](#) を参照してください。

- [session_track_transaction_info](#)

コマンド行形式	<code>--session-track-transaction-info=value</code>
システム変数	session_track_transaction_info
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	OFF
有効な値	OFF STATE

CHARACTERISTICS

サーバーが現在のセッション内のトランザクションの状態と特性を追跡し、この情報を使用可能にするようクライアントに通知するかどうかを制御します。次の `session_track_transaction_info` 値を使用できます:

- **OFF**: トランザクション状態トラッキングを無効にします。これはデフォルトです。
- **STATE**: 特性トラッキングなしでトランザクション状態トラッキングを有効にします。状態トラッキングを使用すると、クライアントは、トランザクションが進行中かどうか、およびトランザクションをロールバックせずに別のセッションに移動できるかどうかを判断できます。
- **CHARACTERISTICS**: 特性トラッキングを含むトランザクション状態トラッキングを有効にします。特性トラッキングを使用すると、クライアントは、元のセッションと同じ特性を持つように、別のセッションでトランザクションを再開する方法を決定できます。この目的に関連する特性は次のとおりです:

```
ISOLATION LEVEL
READ ONLY
READ WRITE
WITH CONSISTENT SNAPSHOT
```

クライアントがトランザクションを別のセッションに安全に再配置するには、トランザクションの状態だけでなく、トランザクションの特性も追跡する必要があります。また、クライアントは、`transaction_isolation` および `transaction_read_only` システム変数を追跡して、セッションのデフォルトを正しく決定する必要があります。(これらの変数を追跡するには、`session_track_system_variables` システム変数の値にリストします。)

セッションステートトラッキングの詳細は、[セクション5.1.18「クライアントセッション状態の変更のサーバートラッキング」](#)を参照してください。

- [sha256_password_auto_generate_rsa_keys](#)

コマンド行形式	<code>--sha256-password-auto-generate-rsa-keys[={OFF ON}]</code>
システム変数	<code>sha256_password_auto_generate_rsa_keys</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

サーバーはこの変数を使用して、RSA 秘密/公開鍵ペアファイルがまだ存在しない場合に、それらをデータディレクトリに自動生成するかどうかを決定します。

これらの条件がすべて満たされている場合、サーバーは起動時に RSA 秘密キー/公開キーのペアファイルをデータディレクトリに自動的に生成: `sha256_password_auto_generate_rsa_keys` または `cached_sha2_password_auto_generate_rsa_keys` システム変数があり、RSA オプションが指定されていません。RSA ファイルがデータディレクトリにありません。これらのキーペアファイルを使用すると、`sha256_password` または `cached_sha2_password` プラグインによって認証されたアカウントに対して、暗号化されていない RSA 接続を使用したセキュアなパスワード交換が可能になります。[セクション6.4.1.3「SHA-256 プラガブル認証」](#) および [セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#)を参照してください。

RSA ファイルの自動生成の詳細(ファイル名や特性など)は、[セクション6.3.3.1「MySQL を使用した SSL および RSA 証明書とキーの作成」](#)を参照してください

`auto_generate_certs` システム変数は関連していますが、SSL を使用したセキュアな接続に必要な SSL 証明書およびキーファイルの自動生成を制御します。

- [sha256_password_private_key_path](#)

コマンド行形式	<code>--sha256-password-private-key-path=file_name</code>
---------	---

システム変数	sha256_password_private_key_path
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	private_key.pem

この変数の値は、[sha256_password](#) 認証プラグインの RSA 秘密キーファイルのパス名です。ファイル名が相対パスとして指定された場合、サーバーのデータディレクトリを基準として解釈されます。ファイルは PEM 形式である必要があります。

重要

このファイルは秘密鍵を格納しているため、MySQL Server のみがファイルを読み取りできるようにファイルのアクセスモードを制限します。

[sha256_password](#) については、[セクション6.4.1.3「SHA-256 プラガブル認証」](#)を参照してください。

- [sha256_password_proxy_users](#)

コマンド行形式	<code>--sha256-password-proxy-users[={OFF ON}]</code>
システム変数	sha256_password_proxy_users
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

この変数は、[sha256_password](#) 組込み認証プラグインがプロキシユーザーをサポートするかどうかを制御します。[check_proxy_users](#) システム変数が有効になっていないかぎり、効果はありません。ユーザープロキシの詳細は、[セクション6.2.18「プロキシユーザー」](#)を参照してください。

- [sha256_password_public_key_path](#)

コマンド行形式	<code>--sha256-password-public-key-path=file_name</code>
システム変数	sha256_password_public_key_path
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	public_key.pem

この変数の値は、[sha256_password](#) 認証プラグインの RSA 公開キーファイルのパス名です。ファイル名が相対パスとして指定された場合、サーバーのデータディレクトリを基準として解釈されます。ファイルは PEM 形式である必要があります。このファイルは公開鍵を格納しているため、クライアントユーザーに対してコピーを自由に配布できます。(RSA パスワード暗号化を使用してサーバーに接続するときに公開鍵を明示的に指定するクライアントは、サーバーで使用されるものと同じ公開鍵を使用する必要があります。)

クライアントが RSA 公開キーを指定する方法など、[sha256_password](#) の詳細は、[セクション6.4.1.3「SHA-256 プラガブル認証」](#)を参照してください。

- shared_memory

コマンド行形式	--shared-memory[={OFF ON}]
システム変数	shared_memory
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
プラットフォーム固有	Windows
型	Boolean
デフォルト値	OFF

(Windows のみ。) サーバーが共有メモリー接続を許可するかどうか。

- shared_memory_base_name

コマンド行形式	--shared-memory-base-name=name
システム変数	shared_memory_base_name
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
プラットフォーム固有	Windows
型	文字列
デフォルト値	MYSQL

(Windows のみ。) 共有メモリー接続に使用する共有メモリーの名前。これは、単一の物理マシン上で複数の MySQL インスタンスを実行する場合に便利です。デフォルト名は **MYSQL** です。名前では大文字と小文字が区別されます。

この変数は、共有メモリー接続をサポートするために **shared_memory** システム変数を有効にしてサーバーを起動した場合にのみ適用されます。

- show_create_table_skip_secondary_engine

コマンド行形式	--show-create-table-skip-secondary-engine[={OFF ON}]
導入	8.0.18
システム変数	show_create_table_skip_secondary_engine
スコープ	セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Boolean

デフォルト値	OFF
--------	-----

`show_create_table_skip_secondary_engine` を有効にすると、`SECONDARY ENGINE` 句が `SHOW CREATE TABLE` 出力および `mysqldump` ユーティリティによってダンプされた `CREATE TABLE` ステートメントから除外されません。

`mysqldump` には `--show-create-skip-secondary-engine` オプションが用意されています。指定すると、ダンプ操作中に `show_create_table_skip_secondary_engine` システム変数が有効になります。

`show_create_table_skip_secondary_engine` 変数をサポートしていない MySQL 8.0.18 より前のリリースで `--show-create-skip-secondary-engine` オプションを使用して `mysqldump` 操作を試行すると、エラーが発生します。

HeatWave で使用します。MySQL HeatWave User Guide を参照してください。

- `show_create_table_verbosity`

コマンド行形式	<code>--show-create-table-verbosity[={OFF ON}]</code>
システム変数	<code>show_create_table_verbosity</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

行フォーマットがデフォルトフォーマットの場合、`SHOW CREATE TABLE` では通常、`ROW_FORMAT` テーブルオプションは表示されません。この変数を有効にすると、`SHOW CREATE TABLE` では、デフォルトフォーマットであるかどうかに関係なく、`ROW_FORMAT` が表示されます。

- `show_old_temporals`

コマンド行形式	<code>--show-old-temporals[={OFF ON}]</code>
非推奨	はい
システム変数	<code>show_old_temporals</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

`SHOW CREATE TABLE` 出力に、5.6.4 より前の形式で検出された時間的カラムにフラグを付けるコメントが含まれるかどうか (`TIME`、`DATETIME` および `TIMESTAMP` カラムで小数秒精度はサポートされません)。この変数はデフォルトでは無効になっています。有効な場合、`SHOW CREATE TABLE` 出力は次のようになります:

```
CREATE TABLE `mytbl` (
  `ts` timestamp /* 5.5 binary format */ NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `dt` datetime /* 5.5 binary format */ DEFAULT NULL,
  `t` time /* 5.5 binary format */ DEFAULT NULL
) DEFAULT CHARSET=utf8mb4
```

`INFORMATION_SCHEMA.COLUMNS` テーブルの `COLUMN_TYPE` カラムの出力も同様に影響を受けます。

この変数は非推奨です。将来の MySQL リリースで削除される予定です。

- `skip_external_locking`

コマンド行形式	<code>--skip-external-locking[={OFF ON}]</code>
---------	---

システム変数	skip_external_locking
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

これは、`mysqld` が外部ロック (システムロック) を使用する場合は `OFF` で、外部ロックが無効な場合は `ON` です。これは、`MyISAM` テーブルアクセスにのみ影響します。

この変数は、`--external-locking` または `--skip-external-locking` オプションによって設定されます。外部ロックはデフォルトで無効になっています。

外部ロックは `MyISAM` テーブルアクセスにのみ影響します。使用できるまたはできない状況も含めた詳細情報については、[セクション8.11.5「外部ロック」](#)を参照してください。

- [skip_name_resolve](#)

コマンド行形式	<code>--skip-name-resolve[={OFF ON}]</code>
システム変数	skip_name_resolve
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

クライアント接続のチェック時にホスト名を解決するかどうか。この変数が `OFF` の場合、`mysqld` はクライアント接続のチェック時にホスト名を解決します。 `ON` の場合、`mysqld` は IP 番号のみを使用します。この場合、付与テーブル内のすべての `Host` カラム値は IP アドレスである必要があります。 [セクション5.1.12.3「DNS ルックアップとホストキャッシュ」](#)を参照してください。

システムのネットワーク構成およびアカウントの `Host` 値によっては、クライアントは `--host=127.0.0.1` や `--host>:::1` などの明示的な `--host` オプションを使用して接続する必要がある場合があります。

ホスト `127.0.0.1` への接続を試みると、通常 `localhost` アカウントに解決します。ただし、サーバーが `skip_name_resolve` を有効にして実行されている場合、これは失敗します。これを行う場合は、接続を受け入れることができるアカウントが存在することを確認します。たとえば、`--host=127.0.0.1` または `--host>:::1` を使用して `root` として接続できるようにするには、次のアカウントを作成します:

```
CREATE USER 'root'@'127.0.0.1' IDENTIFIED BY 'root-password';
CREATE USER 'root'@':::1' IDENTIFIED BY 'root-password';
```

- [skip_networking](#)

コマンド行形式	<code>--skip-networking[={OFF ON}]</code>
システム変数	skip_networking
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

この変数は、サーバーが TCP/IP 接続を許可するかどうかを制御します。デフォルトでは無効になっています (TCP 接続を許可します)。有効な場合、サーバーはローカル (TCP/IP 以外) 接続のみを許可し、`mysqld` とのすべての対

話は、名前付きパイプ、共有メモリー (Windows の場合) または Unix ソケットファイル (Unix の場合) を使用して行う必要があります。このオプションは、ローカルクライアントのみが許可されているシステムで強く推奨します。[セクション5.1.12.3「DNS ルックアップとホストキャッシュ」](#)を参照してください。

`--skip-grant-tables` を使用してサーバーを起動すると認証チェックが無効になるため、この場合、サーバーは `skip_networking` を有効にしてリモート接続も無効にします。

- `skip_show_database`

コマンド行形式	<code>--skip-show-database</code>
システム変数	<code>skip_show_database</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ

これは、`SHOW DATABASES` 権限を持っていないユーザーが `SHOW DATABASES` ステートメントを使用することを防ぎます。ほかのユーザーに属するデータベースをユーザーが表示できることに不安がある場合に、セキュリティを高めることができます。この効果は `SHOW DATABASES` 権限によって異なります。変数の値が `ON` の場合、`SHOW DATABASES` ステートメントは `SHOW DATABASES` 権限を持つユーザーにのみ許可され、ステートメントはすべてのデータベース名を表示します。値が `OFF` の場合、`SHOW DATABASES` はすべてのユーザーに許可されますが、ユーザーが `SHOW DATABASES` またはほかの権限を持つデータベースの名前のみが表示されます。

注意

静的グローバル権限はすべてのデータベースに対する権限とみなされるため、静的グローバル権限を使用すると、ユーザーは、部分的な取消しによってデータベースレベルで制限されているデータベースを除き、`SHOW DATABASES` を使用するか、`INFORMATION_SCHEMA` の `SCHEMATA` テーブルを調べることで、すべてのデータベース名を表示できます。

- `slow_launch_time`

コマンド行形式	<code>--slow-launch-time=#</code>
システム変数	<code>slow_launch_time</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	2

スレッドを作成する時間がこの秒数より長くなると、サーバーは `Slow_launch_threads` ステータス変数を増やします。

- `slow_query_log`

コマンド行形式	<code>--slow-query-log[={OFF ON}]</code>
システム変数	<code>slow_query_log</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean

デフォルト値	OFF
--------	-----

スロークエリーログを有効にするかどうか。値が 0 (または OFF) の場合はログを無効にし、1 (または ON) の場合はログを有効にします。ログ出力先は `log_output` システム変数によって制御され、この値を NONE にした場合はログが有効になっていてもログエントリは書き込まれません。

「スロー」の程度は、`long_query_time` 変数の値によって決定されます。セクション5.4.5「スロークエリーログ」を参照してください。

- [slow_query_log_file](#)

コマンド行形式	<code>--slow-query-log-file=file_name</code>
システム変数	slow_query_log_file
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	<code>host_name-slow.log</code>

スロークエリーログファイルの名前。デフォルト値は `host_name-slow.log` ですが、初期値は `--slow_query_log_file` オプションを使用すると変更できます。

- [socket](#)

コマンド行形式	<code>--socket={file_name pipe_name}</code>
システム変数	socket
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値 (Windows)	MySQL
デフォルト値 (その他)	<code>/tmp/mysql.sock</code>

Unix プラットフォームでは、この変数は、ローカルクライアント接続に使用されるソケットファイルの名前です。デフォルトは `/tmp/mysql.sock` です。(一部の配布形式ではディレクトリが異なる場合があります、たとえば RPM の場合は `/var/lib/mysql` です。)

Windows では、この変数は、ローカルクライアント接続に使用される名前付きパイプの名前です。デフォルト値は MySQL です (大/小文字は区別されません)。

- [sort_buffer_size](#)

コマンド行形式	<code>--sort-buffer-size=#</code>
システム変数	sort_buffer_size
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Integer
デフォルト値	262144
最小値	32768
最大値 (Windows)	4294967295

最大値 (その他, 64 ビットプラットフォーム)	18446744073709551615
最大値 (その他, 32 ビットプラットフォーム)	4294967295

ソートを実行する必要がある各セッションは、このサイズのバッファを割り当てます。sort_buffer_size はどのストレージエンジンにも固有ではなく、最適化の一般的な方法で適用されます。少なくとも、sort_buffer_size 値はソートバッファ内の 15 個のタプルを収容できる十分な大きさである必要があります。また、max_sort_length の値を増やすには、sort_buffer_size の値を増やす必要がある場合があります。詳細は、[セクション8.2.1.16「ORDER BY の最適化」](#)を参照してください

SHOW GLOBAL STATUS の出力に表示される秒あたりの Sort_merge_passes の数が多い場合、sort_buffer_size 値を増やすことで、クエリー最適化またはインデックスの改善によって改善できない ORDER BY または GROUP BY 操作を高速化することを検討できます。

オプティマイザは、必要な領域の量を計算しようとはしますが、制限まで割り当てることができません。必要以上に大きく設定すると、ソートを実行するほとんどのクエリーがグローバルに遅くなります。これはセッション設定として増やし、かつ大きいサイズを必要とするセッションに制限することを推奨します。Linux の場合、256K バイトおよび 2M バイトのしきい値があり、それより大きい値にするとメモリー割り当てが著しく低速になるため、これらのいずれかの値より低くすることを検討してください。実験して、ワークロードに最適な値を見つけてください。[セクションB.3.3.5「MySQL が一時ファイルを格納する場所」](#)を参照してください。

許可される sort_buffer_size の最大の設定値は 4G バイト - 1 です。64 ビットプラットフォームの場合は大きい値が許可されます (64 ビットの Windows の場合は例外で、大きい値は 4G バイト - 1 に切り捨てられて警告が出ます)。

- [sql_auto_is_null](#)

システム変数	sql_auto_is_null
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Boolean
デフォルト値	OFF

この変数が有効な場合、自動的に生成された AUTO_INCREMENT 値を正常に挿入するステートメントの後に、次の形式のステートメントを発行してその値を検索できます:

```
SELECT * FROM tbl_name WHERE auto_col IS NULL
```

ステートメントが行を返す場合、返される値は LAST_INSERT_ID() 関数を呼び出した場合と同じです。複数行の挿入後の戻り値などについての詳細は、[セクション12.16「情報関数」](#)を参照してください。AUTO_INCREMENT 値を正常に挿入できなかった場合、SELECT ステートメントは行を返しません。

IS NULL 比較を使用して AUTO_INCREMENT 値を取得する動作は、Access などの一部の ODBC プログラムによって使用されます。Obtaining Auto-Increment Values を参照してください。この動作は、sql_auto_is_null を OFF に設定することで無効にできます。

MySQL 8.0.16 より前は、WHERE auto_col IS NULL から WHERE auto_col = LAST_INSERT_ID() への変換はステートメントの実行時にのみ実行されていたため、実行中の sql_auto_is_null の値によってクエリーが変換されたかどうか判断されました。MySQL 8.0.16 以降では、ステートメントの準備中に変換が実行されます。

sql_auto_is_null のデフォルト値は OFF です。

- [sql_big_selects](#)

システム変数	sql_big_selects
スコープ	グローバル、セッション
動的	はい

SET_VAR ヒントの適用	はい
型	Boolean
デフォルト値	ON

OFF に設定されている場合、MySQL は、実行に非常に時間がかかる可能性が高い **SELECT** ステートメント (つまり、オプティマイザが調査対象の行数が `max_join_size` の値を超えていると見積もるステートメント) を中断します。これは、推奨されない **WHERE** ステートメントが発行されたときに便利です。新しい接続のデフォルト値は **ON** で、すべての **SELECT** ステートメントが許可されます。

`max_join_size` システム変数を **DEFAULT** 以外の値に設定すると、`sql_big_selects` は **OFF** に設定されます。

- `sql_buffer_result`

システム変数	<code>sql_buffer_result</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Boolean
デフォルト値	OFF

有効にすると、`sql_buffer_result` は **SELECT** ステートメントの結果を一時テーブルに強制的に配置します。これは、MySQL でテーブルロックを早期に解放するのに役立ち、クライアントに結果を送信するのに長い時間がかかる場合に適していることがあります。デフォルト値は **OFF** です。

- `sql_log_off`

システム変数	<code>sql_log_off</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF
有効な値	OFF (ロギングの有効化) ON (ロギングの無効化)

この変数は、一般クエリログへのロギングを現在のセッションで無効にするかどうかを制御します (一般クエリログ自体が有効になっていると仮定します)。デフォルト値は **OFF** です (つまり、ロギングを有効にします)。現在のセッションの一般クエリログを無効または有効にするには、セッション `sql_log_off` 変数を **ON** または **OFF** に設定します。

このシステム変数のセッション値の設定は制限された操作です。セッションユーザーには、制限付きセッション変数を設定するのに十分な権限が必要です。セクション5.1.9.1「システム変数権限」を参照してください。

- `sql_mode`

コマンド行形式	<code>--sql-mode=name</code>
システム変数	<code>sql_mode</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Set

デフォルト値	ONLY_FULL_GROUP_BY STRICT_TRANS_TABLES NO_ZERO_IN_DATE NO_ZERO_DATE ERROR_FOR_DIVISION_BY_ZERO NO_ENGINE_SUBSTITUTION
有効な値	ALLOW_INVALID_DATES ANSI_QUOTES ERROR_FOR_DIVISION_BY_ZERO HIGH_NOT_PRECEDENCE IGNORE_SPACE NO_AUTO_VALUE_ON_ZERO NO_BACKSLASH_ESCAPES NO_DIR_IN_CREATE NO_ENGINE_SUBSTITUTION NO_UNSIGNED_SUBTRACTION NO_ZERO_DATE NO_ZERO_IN_DATE ONLY_FULL_GROUP_BY PAD_CHAR_TO_FULL_LENGTH PIPES_AS_CONCAT REAL_AS_FLOAT STRICT_ALL_TABLES STRICT_TRANS_TABLES TIME_TRUNCATE_FRACTIONAL

現在のサーバー SQL モードで、動的に設定できます。詳細は、[セクション5.1.11「サーバー SQL モード」](#)を参照してください。

注記

MySQL インストールプログラムはインストールプロセス中に SQL モードを構成することがあります。

SQL モードがデフォルトまたは期待されているモードと異なる場合、サーバーが起動時に読み取るオプションファイル内の設定を確認してください。

• [sql_notes](#)

システム変数	sql_notes
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean

デフォルト値	ON
--------	----

有効 (デフォルト) にすると、**Note** レベルの診断によって `warning_count` が増分され、サーバーによって記録されます。無効にした場合、**Note** 診断では `warning_count` が増分されず、サーバーでは記録されません。`mysqldump` には、ダンプファイルをリロードしてもリロード操作の整合性に影響しないイベントの警告が生成されないように、この変数を無効にする出力が含まれています。

- [sql_quote_show_create](#)

システム変数	sql_quote_show_create
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

有効 (デフォルト) にすると、サーバーは **SHOW CREATE TABLE** および **SHOW CREATE DATABASE** ステートメントの識別子を引用符で囲みます。使用不可の場合、見積りは使用不可になります。このオプションはデフォルトで有効化されているため、引用が必要な識別子に対してレプリケーションが機能します。[セクション 13.7.7.10 「SHOW CREATE TABLE ステートメント」](#) および [セクション 13.7.7.6 「SHOW CREATE DATABASE ステートメント」](#) を参照してください。

- [sql_require_primary_key](#)

コマンド行形式	<code>--sql-require-primary-key[={OFF ON}]</code>
導入	8.0.13
システム変数	sql_require_primary_key
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Boolean
デフォルト値	OFF

新しいテーブルを作成するステートメントまたは既存のテーブルの構造を変更するステートメントが、テーブルに主キーがあるという要件を強制するかどうか。

このシステム変数のセッション値の設定は制限された操作です。セッションユーザーには、制限付きセッション変数を設定するのに十分な権限が必要です。[セクション 5.1.9.1 「システム変数権限」](#) を参照してください。

この変数を有効にすると、テーブルに主キーがない場合に発生する可能性のある行ベースのレプリケーションでのパフォーマンスの問題を回避できます。テーブルに主キーがなく、更新または削除によって複数の行が変更されるとします。レプリケーションソースサーバーでは、この操作は単一のテーブルスキャンを使用して実行できますが、行ベースのレプリケーションを使用してレプリケートすると、レプリカで変更される行ごとにテーブルスキャンが行われます。主キーの場合、これらのテーブルスキャンは行われません。

`sql_require_primary_key` は実テーブルと **TEMPORARY** テーブルの両方に適用され、その値に対する変更はレプリカサーバーにレプリケートされます。MySQL 8.0.18 では、レプリケーションに参加できるストレージエンジンにのみ適用されます。

有効にすると、`sql_require_primary_key` には次の効果があります:

- 主キーのない新しいテーブルを作成しようとすると、エラーで失敗します。これには、**CREATE TABLE ... LIKE** が含まれます。**CREATE TABLE** 部分に主キー定義が含まれていないかぎり、**CREATE TABLE ... SELECT** も含まれます。

- 既存のテーブルから主キーを削除しようとする、エラーで失敗します。ただし、同じ `ALTER TABLE` ステートメントで主キーを削除して主キーを追加することはできません。

テーブルに `UNIQUE NOT NULL` インデックスも含まれている場合でも、主キーの削除は失敗します。

- 主キーのないテーブルをインポートしようとする、エラーで失敗します。

`CHANGE REPLICATION SOURCE TO` ステートメント (MySQL 8.0.23 の場合) または `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 の場合) の `REQUIRE_TABLE_PRIMARY_KEY_CHECK` オプションを使用すると、レプリカは主キーチェック用の独自のポリシーを選択できます。レプリケーションチャンネルのオプションが `ON` に設定されている場合、レプリカはレプリケーション操作で常に `sql_require_primary_key` システム変数に値 `ON` を使用し、主キーが必要です。このオプションが `OFF` に設定されている場合、レプリカはレプリケーション操作で `sql_require_primary_key` システム変数に常に値 `OFF` を使用するため、ソースで必要な場合でも主キーは必要ありません。`REQUIRE_TABLE_PRIMARY_KEY_CHECK` オプションが `STREAM` (デフォルト) に設定されている場合、レプリカは各トランザクションのソースからレプリケートされた値を使用します。`REQUIRE_TABLE_PRIMARY_KEY_CHECK` オプションの `STREAM` 設定では、レプリケーションチャンネルに権限チェックが使用されている場合、`PRIVILEGE_CHECKS_USER` アカウントには、`sql_require_primary_key` システム変数のセッション値を設定できるように、制限されたセッション変数を設定するのに十分な権限が必要です。`ON` または `OFF` の設定では、アカウントにこれらの権限は必要ありません。詳細は、[セクション17.3.3「レプリケーション権限チェック」](#)を参照してください。

- `sql_safe_updates`

システム変数	<code>sql_safe_updates</code>
スコープ	グローバル、セッション
動的	はい
<code>SET_VAR</code> ヒントの適用	はい
型	Boolean
デフォルト値	<code>OFF</code>

この変数を有効にすると、`WHERE` 句または `LIMIT` 句でキーを使用しない `UPDATE` および `DELETE` ステートメントでエラーが発生します。これにより、キーが正しく使用されず、多くの行が変更または削除される可能性がある `UPDATE` および `DELETE` ステートメントを捕捉できます。デフォルト値は `OFF` です。

mysql クライアントの場合は、`--safe-updates` オプションを使用して `sql_safe_updates` を有効にできます。詳細は、[セーフ更新モードの使用 \(--safe-updates\)](#)を参照してください。

- `sql_select_limit`

システム変数	<code>sql_select_limit</code>
スコープ	グローバル、セッション
動的	はい
<code>SET_VAR</code> ヒントの適用	はい
型	Integer

`SELECT` ステートメントから返される最大行数。詳細は、[セーフ更新モードの使用 \(--safe-updates\)](#)を参照してください。

新規接続についてのデフォルト値は、サーバーがテーブルあたりで許可する最大行数です。標準的なデフォルト値は $(2^{32})-1$ または $(2^{64})-1$ です。制限を変更した場合、デフォルト値は `DEFAULT` の値を割り当てることでリストアできます。

`SELECT` に `LIMIT` 句がある場合、`LIMIT` が `sql_select_limit` の値に優先されます。

- `sql_warnings`

システム変数	<code>sql_warnings</code>
--------	---------------------------

スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

この変数は、警告が発生する場合に、単一行の `INSERT` ステートメントが情報文字列を生成するかどうかを制御します。デフォルトは `OFF` です。情報文字列を生成するには、値を `ON` に設定します。

- `ssl_ca`

コマンド行形式	<code>--ssl-ca=file_name</code>
システム変数	<code>ssl_ca</code>
スコープ	グローバル
動的 (≥ 8.0.16)	はい
動的 (≤ 8.0.15)	いいえ
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	<code>NULL</code>

PEM 形式の認証局 (CA) 証明書ファイルのパス名。このファイルには、信頼できる SSL 認証局のリストが含まれています。

MySQL 8.0.16 では、この変数は動的であり、サーバーが新しい接続に使用する TSL コンテキストに影響を与えるように実行時に変更できます。サーバー側のランタイム構成および暗号化された接続の監視を参照してください。MySQL 8.0.16 より前は、この変数はサーバーの起動時にのみ設定できます。

- `ssl_capath`

コマンド行形式	<code>--ssl-capath=dir_name</code>
システム変数	<code>ssl_capath</code>
スコープ	グローバル
動的 (≥ 8.0.16)	はい
動的 (≤ 8.0.15)	いいえ
SET_VAR ヒントの適用	いいえ
型	ディレクトリ名
デフォルト値	<code>NULL</code>

PEM 形式の信頼できる SSL 認証局 (CA) 証明書ファイルを含むディレクトリのパス名。

MySQL 8.0.16 では、この変数は動的であり、サーバーが新しい接続に使用する TSL コンテキストに影響を与えるように実行時に変更できます。サーバー側のランタイム構成および暗号化された接続の監視を参照してください。MySQL 8.0.16 より前は、この変数はサーバーの起動時にのみ設定できます。

- `ssl_cert`

コマンド行形式	<code>--ssl-cert=file_name</code>
システム変数	<code>ssl_cert</code>
スコープ	グローバル
動的 (≥ 8.0.16)	はい
動的 (≤ 8.0.15)	いいえ

SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	NULL

PEM 形式のサーバー SSL 公開キー証明書ファイルのパス名。

制限付き暗号または暗号カテゴリを使用する証明書に `ssl_cert` が設定された状態でサーバーを起動すると、サーバーは暗号化された接続のサポートを無効にして起動します。暗号制限の詳細は、[接続暗号構成](#) を参照してください。

MySQL 8.0.16 では、この変数は動的であり、サーバーが新しい接続に使用する TSL コンテキストに影響を与えるように実行時に変更できます。[サーバー側のランタイム構成および暗号化された接続の監視](#) を参照してください。MySQL 8.0.16 より前は、この変数はサーバーの起動時にのみ設定できます。

- `ssl_cipher`

コマンド行形式	<code>--ssl-cipher=name</code>
システム変数	<code>ssl_cipher</code>
スコープ	グローバル
動的 (≥ 8.0.16)	はい
動的 (≤ 8.0.15)	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	NULL

TLSv1.2までのTLSプロトコルを使用する接続に許可される暗号のリスト。リスト内の暗号がサポートされていない場合、これらのTLSプロトコルを使用する暗号化された接続は機能しません。

移植性を最大限に高めるために、暗号リストはコロンで区切られた1つ以上の暗号名のリストである必要があります。例:

```
[mysqld]
ssl_cipher="AES128-SHA"
ssl_cipher="DHE-RSA-AES128-GCM-SHA256:AES128-SHA"
```

OpenSSL は、<https://www.openssl.org/docs/manmaster/man1/ciphers.html> の OpenSSL ドキュメントで説明されている暗号を指定するための構文をサポートしています。

MySQL がサポートする暗号化暗号の詳細は、[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#) を参照してください。

MySQL 8.0.16 では、この変数は動的であり、サーバーが新しい接続に使用する TSL コンテキストに影響を与えるように実行時に変更できます。[サーバー側のランタイム構成および暗号化された接続の監視](#) を参照してください。MySQL 8.0.16 より前は、この変数はサーバーの起動時にのみ設定できます。

- `ssl_crl`

コマンド行形式	<code>--ssl-crl=file_name</code>
システム変数	<code>ssl_crl</code>
スコープ	グローバル
動的 (≥ 8.0.16)	はい
動的 (≤ 8.0.15)	いいえ
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	NULL

PEM 形式の証明書失効リストを含むファイルのパス名。

MySQL 8.0.16 では、この変数は動的であり、サーバーが新しい接続に使用する TSL コンテキストに影響を与えるように実行時に変更できます。サーバー側のランタイム構成および暗号化された接続の監視を参照してください。MySQL 8.0.16 より前は、この変数はサーバーの起動時にのみ設定できます。

- [ssl_crlpath](#)

コマンド行形式	<code>--ssl-crlpath=dir_name</code>
システム変数	ssl_crlpath
スコープ	グローバル
動的 (≥ 8.0.16)	はい
動的 (≤ 8.0.15)	いいえ
SET_VAR ヒントの適用	いいえ
型	ディレクトリ名
デフォルト値	NULL

PEM 形式の証明書失効リストファイルを含むディレクトリのパス。

MySQL 8.0.16 では、この変数は動的であり、サーバーが新しい接続に使用する TSL コンテキストに影響を与えるように実行時に変更できます。サーバー側のランタイム構成および暗号化された接続の監視を参照してください。MySQL 8.0.16 より前は、この変数はサーバーの起動時にのみ設定できます。

- [ssl_fips_mode](#)

コマンド行形式	<code>--ssl-fips-mode={OFF ON STRICT}</code>
システム変数	ssl_fips_mode
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	OFF
有効な値	OFF (または 0) ON (または 1) STRICT (または 2)

サーバー側で FIPS モードを有効にするかどうかを制御します。ssl_fips_mode システム変数は、暗号化された接続をサーバーが許可するかどうかの制御には使用されず、許可される暗号化操作に影響する点で、他の ssl_xxx システム変数とは異なります。セクション6.8「FIPS のサポート」を参照してください。

次の ssl_fips_mode 値を使用できます:

- **OFF** (or 0): FIPS モードを無効にします。
- **ON** (or 1): FIPS モードを有効にします。
- **STRICT** (or 2): 「strict」 FIPS モードを有効にします。

注記

OpenSSL FIPS オブジェクトモジュールが使用できない場合、ssl_fips_mode に許可される値は OFF のみです。この場合、起動時に ssl_fips_mode を ON または STRICT に設定すると、サーバーでエラーメッセージが生成されて終了します。

- [ssl_key](#)

コマンド行形式	<code>--ssl-key=file_name</code>
システム変数	<code>ssl_key</code>
スコープ	グローバル
動的 (≥ 8.0.16)	はい
動的 (≤ 8.0.15)	いいえ
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	NULL

PEM 形式のサーバー SSL 秘密キーファイルのパス名。セキュリティを向上させるには、RSA キーサイズが 2048 ビット以上の証明書を使用します。

鍵ファイルがパスワードで保護されている場合、サーバーはユーザーにパスワードの入力を求めます。パスワードは対話形式で指定する必要があり、ファイルには格納できません。パスワードが正しくない場合は、鍵を読み取ることができない場合と同様に、プログラムが続行されます。

MySQL 8.0.16 では、この変数は動的であり、サーバーが新しい接続に使用する TSL コンテキストに影響を与えるように実行時に変更できます。サーバー側のランタイム構成および暗号化された接続の監視を参照してください。MySQL 8.0.16 より前は、この変数はサーバーの起動時のみ設定できます。

- [stored_program_cache](#)

コマンド行形式	<code>--stored-program-cache=#</code>
システム変数	<code>stored_program_cache</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	256
最小値	16
最大値	524288

接続あたりでキャッシュされるストアードルーチンの数について、上側のソフトリミットを設定します。この変数の値は、ストアードプロシージャおよびストアードファンクションで、MySQL Server によって維持される 2 つのキャッシュそれぞれに保持されるストアードルーチンの数に関して指定します。

ストアードルーチンが実行されると、ルーチン内の先頭または最上位レベルのステートメントが解析される前に、このキャッシュサイズが検査されます。同じタイプのルーチン (どちらが実行されているかによってストアードプロシージャまたはストアードファンクション) の数が、この変数によって指定される制限を超える場合、対応するキャッシュがフラッシュされ、キャッシュされたオブジェクトに対して以前割り当てられていたメモリーが解放されます。これにより、ストアードルーチン間に依存関係がある場合でも、キャッシュを安全にフラッシュできます。

ストアードプロシージャおよびストアードファンクションキャッシュは、[dictionary object cache](#) のストアードプログラム定義キャッシュパーティションと並行して存在します。ストアードプロシージャおよびストアードファンクションキャッシュは接続ごとに行われますが、ストアードプログラム定義キャッシュは共有されます。ストアードプロシージャキャッシュとストアードファンクションキャッシュにオブジェクトが存在するかどうかは、ストアードプログラム定義キャッシュにオブジェクトが存在するかどうかに依存しません。その逆も同様です。詳細は、[セクション 14.4 「ディクショナリオブジェクトキャッシュ」](#) を参照してください。

- [stored_program_definition_cache](#)

コマンド行形式	<code>--stored-program-definition-cache=#</code>
---------	--

システム変数	stored_program_definition_cache
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	256
最小値	256
最大値	524288

ディクショナリオブジェクトキャッシュに保持できるストアプログラム定義オブジェクト (使用済と未使用の両方) の数の制限を定義します。

未使用のストアプログラム定義オブジェクトは、使用中の数が [stored_program_definition_cache](#) で定義された容量より少ない場合にのみディクショナリオブジェクトキャッシュに保持されます。

0 に設定すると、ストアプログラム定義オブジェクトは、使用中はディクショナリオブジェクトキャッシュにのみ保持されます。

ストアプログラム定義キャッシュパーティションは、[stored_program_cache](#) オプションを使用して構成されたストアプロシージャおよびストアファンクションキャッシュと並行して存在します。

[stored_program_cache](#) オプションは、接続ごとにキャッシュされるストアプロシージャまたはファンクションの数に弱い上限を設定し、接続がストアプロシージャまたはファンクションを実行するたびに制限がチェックされます。一方、ストアプログラム定義キャッシュパーティションは、他の目的でストアプログラム定義オブジェクトを格納する共有キャッシュです。ストアプログラム定義キャッシュパーティションにオブジェクトが存在しても、ストアプロシージャキャッシュまたはストアファンクションキャッシュにオブジェクトが存在するかどうかには依存しません。その逆も同様です。

関連情報については、[セクション14.4「ディクショナリオブジェクトキャッシュ」](#)を参照してください。

- [super_read_only](#)

コマンド行形式	<code>--super-read-only[={OFF ON}]</code>
システム変数	super_read_only
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

[read_only](#) システム変数が有効になっている場合、サーバーは、[CONNECTION_ADMIN](#) 権限 (または非推奨の [SUPER](#) 権限) を持つユーザー以外のクライアント更新を許可しません。 [super_read_only](#) システム変数も有効になっている場合、サーバーは [CONNECTION_ADMIN](#) または [SUPER](#) を持つユーザーからのクライアント更新を禁止します。 読取り専用モードの説明および [read_only](#) と [super_read_only](#) の相互作用の詳細は、[read_only](#) システム変数の説明を参照してください。

[super_read_only](#) が有効になっているときにクライアントの更新を防止するには、[CREATE FUNCTION](#) (UDF のインストール)、[INSTALL PLUGIN](#)、[INSTALL COMPONENT](#) など、必ずしも更新ではないように見える操作を含めます。これらの操作は、[mysql](#) システムスキーマのテーブルへの変更を伴うため禁止されています。

レプリケーションソースサーバー上の [super_read_only](#) への変更は、複製サーバーにレプリケートされません。この値は、ソースの設定とは無関係にレプリカに設定できます。

- [syseventlog.facility](#)

コマンド行形式	<code>--syseventlog.facility=value</code>
導入	8.0.13
システム変数	syseventlog.facility
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	<code>daemon</code>

`syslog` に書き込まれるエラーログ出力の機能 (メッセージを送信しているプログラムのタイプ)。この変数は、`log_sink_syseventlog` エラーログコンポーネントがインストールされていないかぎり使用できません。 [セクション5.4.2.8「システムログへのエラーロギング」](#) を参照してください。

許可される値はオペレーティングシステムによって異なります。システムの `syslog` ドキュメントを参照してください。

この変数は Windows には存在しません。

- [syseventlog.include_pid](#)

コマンド行形式	<code>--syseventlog.include-pid[={OFF ON}]</code>
導入	8.0.13
システム変数	syseventlog.include_pid
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	<code>ON</code>

`syslog` に書き込まれるエラーログ出力の各行にサーバープロセス ID を含めるかどうか。この変数は、`log_sink_syseventlog` エラーログコンポーネントがインストールされていないかぎり使用できません。 [セクション5.4.2.8「システムログへのエラーロギング」](#) を参照してください。

この変数は Windows には存在しません。

- [syseventlog.tag](#)

コマンド行形式	<code>--syseventlog.tag=tag</code>
導入	8.0.13
システム変数	syseventlog.tag
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列

デフォルト値	empty string
--------	--------------

`syslog` または Windows イベントログに書き込まれるエラーログ出力でサーバー識別子に追加されるタグ。この変数は、`log_sink_syseventlog` エラーログコンポーネントがインストールされていないかぎり使用できません。セクション5.4.2.8「システムログへのエラーロギング」を参照してください。

デフォルトでは、タグは設定されていないため、サーバー識別子は単に Windows では MySQL で、他のプラットフォームでは `mysqld` です。tag のタグ値を指定すると、先頭にハイフンが付いたサーバー識別子にタグ値が追加され、`syslog` 識別子 `mysqld-tag` (または Windows の場合は `MySQL-tag`) が生成されます。

Windows で、まだ存在しないタグを使用するには、管理者権限を持つアカウントからサーバーを実行して、タグのレジストリエントリの作成を許可する必要があります。タグがすでに存在する場合、昇格された権限は必要ありません。

- `system_time_zone`

システム変数	<code>system_time_zone</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

サーバーシステムタイムゾーン。サーバーは実行を開始するとき、マシンのデフォルトからタイムゾーン設定を継承し、サーバーを実行するために使用されるアカウントの環境または起動スクリプトによって変更されることがあります。値は `system_time_zone` を設定するために使用されます。通常、タイムゾーンは `TZ` 環境変数で指定されます。または `mysqld_safe` スクリプトの `--timezone` オプションを使用しても指定できます。

`system_time_zone` 変数は `time_zone` と異なります。これらは同じ値になることもありますが、後者の変数は、接続する各クライアントのタイムゾーンを初期化するために使用されます。セクション5.1.15「MySQL Server でのタイムゾーンのサポート」を参照してください。

- `table_definition_cache`

コマンド行形式	<code>--table-definition-cache=#</code>
システム変数	<code>table_definition_cache</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動サイズ設定を示します。このリテラル値を割り当てないでください)
最小値	400
最大値	524288

定義キャッシュに格納できるテーブル定義の数。多数のテーブルを使用する場合、大きいテーブル定義キャッシュを作成して、テーブルを開くことを高速化できます。標準のテーブルキャッシュと異なり、テーブル定義キャッシュは占有スペースが少なくファイルディスクリプタを使用しません。最小値は 400 です。デフォルト値は次の式に基づき、2000 までに制限されています。

```
MIN(400 + table_open_cache / 2, 2000)
```

InnoDB の場合、`table_definition_cache` は、InnoDB データディクショナリキャッシュ内の開いているテーブルインスタンスの数のソフト制限として機能します。開いているテーブルインスタンスの数が `table_definition_cache` 設定を超えた場合、LRU メカニズムはエビクション用のテーブルインスタンスにマークを付け、最終的にデータディクショナリキャッシュから削除されます。この制限は、次回サーバー開始までに使用頻度が低いテーブルインスタンスをキャッシュするために大量のメモリーが使用されるような状況に対処するのに役立ちます。外部キー関係を持

つ親テーブルインスタンスおよび子テーブルインスタンスは LRU リストに配置されず、メモリーから削除されないため、キャッシュされたメタデータを持つテーブルインスタンスの数は、`table_definition_cache` で定義された制限を超える可能性があります。

さらに、`table_definition_cache` は、一度に開くことができる、InnoDB file-per-table テーブルスペースの数のソフト制限を定義し、これは `innodb_open_files` によっても制御されます。`table_definition_cache` および `innodb_open_files` の両方が設定される場合、高い方の設定値が使用されます。どちらの変数も設定されない場合、デフォルト値が高い `table_definition_cache` が使用されます。オープンテーブルスペースファイルハンドルの数が、`table_definition_cache` または `innodb_open_files` によって定義された制限を超える場合、LRU メカニズムは、テーブルスペースファイル LRU リストを検索して、完全にフラッシュされて現在延長されていないファイルを探します。この処理は、新しいテーブルスペースが開くたびに実行されます。「非アクティブな」テーブルスペースがない場合、テーブルスペースファイルはクローズされません。

テーブル定義キャッシュは、`dictionary object cache` のテーブル定義キャッシュパーティションと並行して存在します。どちらのキャッシュにもテーブル定義が格納されますが、MySQL サーバーの様々な部分に対応します。一方のキャッシュ内のオブジェクトは、他方のキャッシュ内の存在オブジェクトに依存しません。詳細は、[セクション 14.4 「ディクショナリオブジェクトキャッシュ」](#) を参照してください。

- `table_encryption_privilege_check`

コマンド行形式	<code>--table-encryption-privilege-check[={OFF ON}]</code>
導入	8.0.16
システム変数	<code>table_encryption_privilege_check</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

`default_table_encryption` 設定とは異なる暗号化を使用してスキーマまたは一般テーブルスペースを作成または変更する場合、またはデフォルトのスキーマ暗号化とは異なる暗号化設定を使用してテーブルを作成または変更する場合に発生する `TABLE_ENCRYPTION_ADMIN` 権限チェックを制御します。このチェックはデフォルトで無効になっています。

実行時に `table_encryption_privilege_check` を設定するには、`SUPER` 権限が必要です。

`table_encryption_privilege_check` は、`SET PERSIST` および `SET PERSIST_ONLY` 構文をサポートしています。[セクション 5.1.9.3 「永続化されるシステム変数」](#) を参照してください。

詳細は、[スキーマおよび一般テーブルスペースの暗号化デフォルトの定義](#) を参照してください。

- `table_open_cache`

コマンド行形式	<code>--table-open-cache=#</code>
システム変数	<code>table_open_cache</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	4000
最小値	1
最大値	524288

すべてのスレッドについて開いているテーブルの数。この値を大きくすると、`mysqld` が要求するファイルディスクリプタの数が増加します。`Opened_tables` ステータス変数を検査して、テーブルキャッシュを増やす必要があ

るかどうかを確認できます。 [セクション5.1.10「サーバーステータス変数」](#)を参照してください。 `Opened_tables` の値が大きく、 `FLUSH TABLES` をあまり使用しない場合 (すべてのテーブルのクローズおよび再オープン of 強制のみを実行する)、 `table_open_cache` 変数の値を増やします。 テーブルキャッシュに関する詳細は、 [セクション8.4.3.1「MySQLでのテーブルのオープンとクローズの方法」](#)を参照してください。

- [table_open_cache_instances](#)

コマンド行形式	<code>--table-open-cache-instances=#</code>
システム変数	table_open_cache_instances
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	16
最小値	1
最大値	64

オープンテーブルキャッシュインスタンスの数。セッション間の競合を減少させることでスケーラビリティを改善するために、開いているテーブルキャッシュを、サイズが `table_open_cache` / `table_open_cache_instances` のいくつかの小さいキャッシュインスタンスにパーティション化できます。DML ステートメントでは、セッションはインスタンスにアクセスするために、1つのインスタンスのみロックする必要があります。このセグメントキャッシュは複数インスタンスにわたってアクセスし、多くのセッションがテーブルにアクセスする場合にキャッシュを使用する演算の高いパフォーマンスが可能になります。(DDL ステートメントでは引き続きキャッシュ全体のロックが必要ですが、そのようなステートメントは DML ステートメントよりも頻度がずっと低くなります。)

通常 16 以上のコアを使用するシステムでは、8 または 16 の値が推奨されます。

- [tablespace_definition_cache](#)

コマンド行形式	<code>--tablespace-definition-cache=#</code>
システム変数	tablespace_definition_cache
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	256
最小値	256
最大値	524288

ディクショナリオブジェクトキャッシュに保持できるテーブルスペース定義オブジェクト (使用済と未使用の両方) の数の制限を定義します。

未使用のテーブルスペース定義オブジェクトは、使用中の数が `tablespace_definition_cache` で定義されている容量より少ない場合にのみディクショナリオブジェクトキャッシュに保持されます。

0 を設定すると、テーブルスペース定義オブジェクトは、使用中はディクショナリオブジェクトキャッシュにのみ保持されます。

詳細は、 [セクション14.4「ディクショナリオブジェクトキャッシュ」](#)を参照してください。

- [temptable_max_mmap](#)

コマンド行形式	<code>--temptable-max-mmap=#</code>
導入	8.0.23

システム変数	temptable_max_mmap
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1073741824
最小値	0
最大値	2 ⁶⁴ -1

TempTable ストレージエンジンがディスク上の InnoDB 内部一時テーブルへのデータの格納を開始する前に、メモリーマップされた一時ファイルから割り当てることができるメモリーの最大量 (バイト単位) を定義します。0 に設定すると、メモリーマップされた一時ファイルからのメモリーの割り当てが無効になります。詳細は、[セクション 8.4.4 「MySQL での内部一時テーブルの使用」](#) を参照してください。

- [temptable_max_ram](#)

コマンド行形式	<code>--temptable-max-ram=#</code>
システム変数	temptable_max_ram
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1073741824
最小値	2097152
最大値	2 ⁶⁴ -1

TempTable ストレージエンジンがディスクへのデータの格納を開始する前に占有できるメモリーの最大量を定義します。デフォルト値は 1073741824 バイト (1GiB) です。詳細は、[セクション 8.4.4 「MySQL での内部一時テーブルの使用」](#) を参照してください。

- [temptable_use_mmap](#)

コマンド行形式	<code>--temptable-use-mmap[={OFF ON}]</code>
導入	8.0.16
システム変数	temptable_use_mmap
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

TempTable ストレージエンジンが占有しているメモリー量が [temptable_max_ram](#) 変数で定義されている制限を超えた場合に、TempTable ストレージエンジンがメモリーマップされた一時ファイルとして内部インメモリー一時テーブルの領域を割り当てかどうかを定義します。[temptable_use_mmap](#) が無効になっている場合、TempTable ストレージエンジンは代わりに InnoDB ディスク上の内部一時テーブルを使用します。詳細は、[セクション 8.4.4 「MySQL での内部一時テーブルの使用」](#) を参照してください。

- [thread_cache_size](#)

コマンド行形式	<code>--thread-cache-size=#</code>
---------	------------------------------------

システム変数	thread_cache_size
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動サイズ設定を示します。このリテラル値を割り当てないでください)
最小値	0
最大値	16384

サーバーが再使用のためにキャッシュするスレッドの数。クライアントが接続を切断したとき、スレッド数が [thread_cache_size](#) より少なければ、クライアントのスレッドはキャッシュに配置されます。スレッドのリクエストは、可能であれば、キャッシュからのスレッドを再使用することによって満たされ、キャッシュが空の場合のみ新しいスレッドが作成されます。多くの新しい接続がある場合、この変数を増やしてパフォーマンスを向上できます。スレッドの実装が適切な場合、通常はパフォーマンスが著しく改善されることはありません。ただし、1秒あたり数百件の接続がサーバーで見られる場合、通常は [thread_cache_size](#) を十分に高く設定すると、ほとんどの新しい接続でキャッシュされたスレッドを使用できます。ステータス変数 [Connections](#) と [Threads_created](#) の差異を調査することで、スレッドキャッシュの効率性を確認できます。詳細は、[セクション5.1.10「サーバーステータス変数」](#)を参照してください。

デフォルト値は次の式に基づいており、上限は 100 に制限されています。

```
8 + (max_connections / 100)
```

- [thread_handling](#)

コマンド行形式	<code>--thread-handling=name</code>
システム変数	thread_handling
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	<code>one-thread-per-connection</code>
有効な値	<code>no-threads</code> <code>one-thread-per-connection</code> <code>loaded-dynamically</code>

接続スレッドのサーバーによって使用されるスレッド処理モデル。使用可能な値は、`no-threads` (サーバーは単一のスレッドを使用して1つの接続を処理)、`one-thread-per-connection` (サーバーは各クライアント接続を処理するために1つのスレッドを使用)、および `loaded-dynamically` (スレッドプールプラグインによって初期化時に設定) です。`no-threads` は、Linux でのデバッグに役立ちます。[セクション5.9「MySQL のデバッグ」](#)を参照してください。

- [thread_pool_algorithm](#)

コマンド行形式	<code>--thread-pool-algorithm=#</code>
システム変数	thread_pool_algorithm
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer

デフォルト値	0
最小値	0
最大値	1

この変数は、スレッドプールプラグインが使用するアルゴリズムを制御します。

- 値 0 (デフォルト) では、並列性の低い保守的なアルゴリズムが使用されます。これはもっとも良く検査されていて、非常に良好な結果を生成することが知られています。
- 値 1 では並列性が高まり、より積極的なアルゴリズムが使用されます。このアルゴリズムは、最適なスレッドカウントでパフォーマンスが 5 - 10% 高まりますが、接続数が増えるにつれてパフォーマンスが低下することが知られています。この使用は実験的であり、サポートされないものとみなすようにしてください。

この変数は、スレッドプールプラグインが有効な場合にのみ使用できます。 [セクション5.6.3「MySQL Enterprise Thread Pool」](#) を参照してください。

- [thread_pool_high_priority_connection](#)

コマンド行形式	<code>--thread-pool-high-priority-connection=#</code>
システム変数	thread_pool_high_priority_connection
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	1

この変数は、実行前の新規ステートメントのキューイングに影響します。値が 0 (false、デフォルト) の場合、ステートメントのキューイングでは優先度の低いキューと優先度の高いキューの両方が使用されます。値が 1 (true) の場合、キューに入れられるステートメントは常に優先度の高いキューに入ります。

この変数は、スレッドプールプラグインが有効な場合にのみ使用できます。 [セクション5.6.3「MySQL Enterprise Thread Pool」](#) を参照してください。

- [thread_pool_max_active_query_threads](#)

コマンド行形式	<code>--thread-pool-max-active-query-threads</code>
導入	8.0.19
システム変数	thread_pool_max_active_query_threads
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	512

グループ当たりのアクティブな (実行中の) クエリースレッドの最大許容数。値が 0 の場合、スレッドプールプラグインは使用可能な最大数のスレッドを使用します。

この変数は、スレッドプールプラグインが有効な場合にのみ使用できます。 [セクション5.6.3「MySQL Enterprise Thread Pool」](#) を参照してください。

- `thread_pool_max_unused_threads`

コマンド行形式	<code>--thread-pool-max-unused-threads=#</code>
システム変数	<code>thread_pool_max_unused_threads</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	4096

スレッドプール内で許可される最大の未使用スレッド数。この変数により、スリープ状態のスレッドによって使用されるメモリーの量を制限できます。

値 0 (デフォルト) は、スリープ状態のスレッドの数を制限しないことを意味します。値 N は、 N が 0 より大きい場合、1 つのコンシューマスレッドと、 $N-1$ 個の予約スレッドを意味します。この状況で、スレッドがスリープ状態に入ろうとしたが、スリープ状態のスレッド数がすでに最大値に到達している場合、スレッドはスリープ状態に入らずに存在します。

スリープ状態のスレッドは、コンシューマスレッドまたは予約スレッドのいずれかとしてスリープ状態になります。スレッドプールでは、1 つのスレッドがスリープ状態のコンシューマスレッドであることを許可します。スレッドがスリープ状態になり、既存のコンシューマスレッドがない場合は、コンシューマスレッドとしてスリープします。スレッドをウェイクアップさせる必要があるとき、コンシューマスレッドが存在すれば、そのコンシューマスレッドが選択されます。ウェイクアップするコンシューマスレッドがない場合にのみ予約スレッドが選択されます。

この変数は、スレッドプールプラグインが有効な場合にのみ使用できます。 [セクション 5.6.3 「MySQL Enterprise Thread Pool」](#) を参照してください。

- `thread_pool_prio_kickup_timer`

コマンド行形式	<code>--thread-pool-prio-kickup-timer=#</code>
システム変数	<code>thread_pool_prio_kickup_timer</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1000
最小値	0
最大値	4294967294

この変数は、優先度が低いキューで実行を待機するステートメントに影響します。この値は、待機中のステートメントが優先度の高いキューに移されるまでのミリ秒数です。デフォルトは 1000 (1 秒) です。

この変数は、スレッドプールプラグインが有効な場合にのみ使用できます。 [セクション 5.6.3 「MySQL Enterprise Thread Pool」](#) を参照してください。

- `thread_pool_size`

コマンド行形式	<code>--thread-pool-size=#</code>
システム変数	<code>thread_pool_size</code>
スコープ	グローバル

動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	16
最小値	1
最大値 (≥ 8.0.19)	512
最大値 (≤ 8.0.18)	64

スレッドプール内のスレッドグループの数。これはスレッドプールのパフォーマンスを制御するもっとも重要なパラメータです。同時に実行できるステートメントの数に影響します。許容値の範囲外の値を指定すると、スレッドプールプラグインはロードされず、サーバーはエラーログにメッセージを書き込みます。

この変数は、スレッドプールプラグインが有効な場合にのみ使用できます。 [セクション5.6.3「MySQL Enterprise Thread Pool」](#)を参照してください。

- [thread_pool_stall_limit](#)

コマンド行形式	--thread-pool-stall-limit=#
システム変数	thread_pool_stall_limit
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	6
最小値	4
最大値	600

この変数はステートメントの実行に影響します。この値は、ステートメントが実行を開始したあと、ステートメントが停滞していると定義される前に終了する時間量で、その時点で、スレッドプールはスレッドグループは別のステートメントの実行の開始を許可します。値は 10 ミリ秒単位で測定されるため、デフォルトの 6 は 60ms を意味します。待機の値が短いと、スレッドはよりすみやかに開始できます。短い値はデッドロック状況を回避により適しています。長い待機の値は、長時間実行するステートメントを含むワークロードで有用で、現在のステートメントの実行時に多数の新しいステートメントが開始しないようにします。

この変数は、スレッドプールプラグインが有効な場合にのみ使用できます。 [セクション5.6.3「MySQL Enterprise Thread Pool」](#)を参照してください。

- [thread_stack](#)

コマンド行形式	--thread-stack=#
システム変数	thread_stack
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値 (64 ビットプラットフォーム)	286720
デフォルト値 (32 ビットプラットフォーム)	221184
最小値	131072
最大値 (64 ビットプラットフォーム)	18446744073709551615
最大値 (32 ビットプラットフォーム)	4294967295

ブロックサイズ	1024
---------	------

各スレッドのスタックサイズ。デフォルトは、通常の動作に十分な大きさです。スレッドスタックサイズが小さすぎると、サーバーで処理できる SQL ステートメントの複雑さ、ストアードプロシージャの再帰の深さなど、メモリーを大量に消費する処理が制限されます。

- [time_zone](#)

システム変数	time_zone
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用 (≥ 8.0.17)	はい
SET_VAR ヒントの適用 (≤ 8.0.16)	いいえ
型	文字列
デフォルト値	SYSTEM
最小値 (≥ 8.0.19)	-14:00
最小値 (≤ 8.0.18)	-12:59
最大値 (≥ 8.0.19)	+14:00
最大値 (≤ 8.0.18)	+13:00

現在のタイムゾーン。この変数は、接続する各クライアントのタイムゾーンを初期化するために使用されます。デフォルトでは、この初期値は 'SYSTEM' です (「[system_time_zone](#) の値を使用する」ことを意味します)。この値はサーバー起動時に `--default-time-zone` オプションで明示的に指定できます。 [セクション5.1.15「MySQL Serverでのタイムゾーンのサポート」](#) を参照してください。

注記

SYSTEM に設定されている場合、タイムゾーン計算を必要とするすべての MySQL 関数コールは、システムライブラリコールを実行して現在のシステムタイムゾーンを決定します。このコールはグローバル mutex によって保護される可能性があるため、競合が発生します。

- [timestamp](#)

システム変数	timestamp
スコープ	セッション
動的	はい
SET_VAR ヒントの適用	はい
型	数値
デフォルト値	UNIX_TIMESTAMP()
最小値	1

最大値	2147483647
-----	------------

このクライアントの時間を設定します。これは、バイナリログを使用して行を復元する場合に元のタイムスタンプを取得するために使用されます。`timestamp_value` は、Unix エポックタイムスタンプ ('YYYY-MM-DD hh:mm:ss' 形式の値ではなく、`UNIX_TIMESTAMP()` によって返される値など) または `DEFAULT` である必要があります。

`timestamp` を定数値に設定すると、ふたたび変更されるまでその値が保持されます。`timestamp` を `DEFAULT` に設定すると、その値はアクセスを受けた時点での現在の日付および時間になります。

`timestamp` は、値にマイクロ秒部分が含まれているため、`BIGINT` ではなく `DOUBLE` です。最大値は、`TIMESTAMP` データ型の場合と同じように、'2038-01-19 03:14:07' UTC に対応します。

`SET timestamp` は `NOW()` によって戻された値に影響を及ぼしますが、`SYSDATE()` によって戻された値には影響しません。つまり、バイナリログのタイムスタンプ設定は、`SYSDATE()` の呼び出しに影響しないことを意味します。`--sysdate-is-now` オプションを指定してサーバーを起動すると、`SYSDATE()` を `NOW()` のシノニムにできます。この場合、`SET timestamp` は両方の機能に影響します。

- `tls_ciphersuites`

コマンド行形式	<code>--tls-ciphersuites=ciphersuite_list</code>
導入	8.0.16
システム変数	<code>tls_ciphersuites</code>
スコープ	グローバル
動的	はい
<code>SET_VAR</code> ヒントの適用	いいえ
型	文字列
デフォルト値	<code>NULL</code>

TLSv1.3 を使用する暗号化された接続に対してサーバーが許可する暗号スイート。値は、コロンで区切られたゼロ個以上の暗号スイート名のリストです。

この変数に指定できる暗号スイートは、MySQL のコンパイルに使用される SSL ライブラリによって異なります。この変数が設定されていない場合、デフォルト値は `NULL` です。つまり、サーバーは暗号スイートのデフォルトセットを許可します。変数が空の文字列に設定されている場合、暗号スイートは有効にならず、暗号化された接続を確立できません。詳細は、[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#)を参照してください。

- `tls_version`

コマンド行形式	<code>--tls-version=protocol_list</code>
システム変数	<code>tls_version</code>
スコープ	グローバル
動的 (≥ 8.0.16)	はい
動的 (≤ 8.0.15)	いいえ
<code>SET_VAR</code> ヒントの適用	いいえ
型	文字列
デフォルト値 (≥ 8.0.16)	<code>TLSv1,TLSv1.1,TLSv1.2,TLSv1.3</code> (OpenSSL 1.1.1 以上) <code>TLSv1,TLSv1.1,TLSv1.2</code> (otherwise)
デフォルト値 (≤ 8.0.15)	<code>TLSv1,TLSv1.1,TLSv1.2</code>

暗号化された接続に対してサーバーが許可するプロトコル。値は、1 つまたは複数のコンマ区切りプロトコル名のリストです。この変数に指定できるプロトコルは、MySQL のコンパイルに使用される SSL ライブラリによって異

なります。「穴」をリストに残さないなど、許可されたプロトコルを選択する必要があります。詳細は、[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#)を参照してください。

MySQL 8.0.16 では、この変数は動的であり、サーバーが新しい接続に使用する TSL コンテキストに影響を与えるように実行時に変更できます。[サーバー側のランタイム構成および暗号化された接続の監視](#)を参照してください。MySQL 8.0.16 より前は、この変数はサーバーの起動時にのみ設定できます。

- [tmp_table_size](#)

コマンド行形式	<code>--tmp-table-size=#</code>
システム変数	tmp_table_size
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Integer
デフォルト値	16777216
最小値	1024
最大値	18446744073709551615

内部インメモリーの一時テーブルの最大サイズ。この変数はユーザーが作成した MEMORY テーブルには適用されません。

実際の制限は、[tmp_table_size](#) および [max_heap_table_size](#) の小さい方です。インメモリー一時テーブルが制限を超えると、MySQL はそれをディスク上の一時テーブルに自動的に変換します。

多数の高度な GROUP BY クエリーを実行する場合にメモリーが多くあるときは、[tmp_table_size](#) (さらに必要に応じて [max_heap_table_size](#)) の値を増やします。

作成されたディスク上の内部一時テーブルの数と作成された内部一時テーブルの合計数を比較するには、[Created_tmp_disk_tables](#) と [Created_tmp_tables](#) の値を比較します。

[セクション8.4.4「MySQL での内部一時テーブルの使用」](#)も参照してください。

- [tmpdir](#)

コマンド行形式	<code>--tmpdir=dir_name</code>
システム変数	tmpdir
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ディレクトリ名

一時ファイルを作成するために使用するディレクトリのパス。これは、小さすぎて一時テーブルを保持できないパーティション上にデフォルトの `/tmp` ディレクトリがある場合に役立つことがあります。この変数は、ラウンドロビン方式で使われるいくつかのパスのリストとして設定できます。パスは、Unix ではコロン文字 (:) で区切り、Windows ではセミコロン文字 (;) で区切る必要があります。

[tmpdir](#) は、メモリーベースのファイルシステム上のディレクトリや、サーバーホストの再起動時にクリアされるディレクトリなど、非永続的な場所にすることができます。MySQL サーバーがレプリカとして機能しており、[tmpdir](#) に非永続的な場所を使用している場合は、[slave_load_tmpdir](#) 変数を使用してレプリカに別の一時ディレクトリを設定することを検討してください。レプリカの場合、LOAD DATA ステートメントのレプリケートに使用される一時ファイルはこのディレクトリに格納されるため、永続的な場所ではマシンの再起動後も存続できますが、一時ファイルが削除されている場合は、再起動後もレプリケーションを続行できるようになりました。

一時ファイルのストレージ位置に関しては、[セクションB.3.3.5「MySQL が一時ファイルを格納する場所」](#)を参照してください。

- [transaction_alloc_block_size](#)

コマンド行形式	<code>--transaction-alloc-block-size=#</code>
システム変数	transaction_alloc_block_size
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	8192
最小値	1024
最大値	131072
ブロックサイズ	1024

メモリーを必要とするトランザクションごとのメモリープールを増やす、バイト単位の増加量。
[transaction_prealloc_size](#) の説明を参照してください。

- [transaction_isolation](#)

コマンド行形式	<code>--transaction-isolation=name</code>
システム変数	transaction_isolation
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	REPEATABLE-READ
有効な値	READ-UNCOMMITTED READ-COMMITTED REPEATABLE-READ SERIALIZABLE

トランザクション分離レベル。デフォルトは [REPEATABLE-READ](#) です。

トランザクション分離レベルには 3 つのスコープがあります: グローバル、セッションおよび次のトランザクション。この 3 スコープの実装は、あとで説明するように、いくつかの非標準のアイソレーションレベルの割り当てセマンティクスにつながります。

起動時にグローバルトランザクション分離レベルを設定するには、`--transaction-isolation` サーバーオプションを使用します。

実行時には、`SET` ステートメントを使用して [transaction_isolation](#) システム変数に値を割り当てるか、`SET TRANSACTION` ステートメントを使用して間接的に分離レベルを設定できます。 [transaction_isolation](#) をスパー

スを含む分離レベル名に直接設定する場合は、スペースをダッシュで置き換えて名前を引用符で囲む必要があります。たとえば、次の **SET** ステートメントを使用してグローバル値を設定します:

```
SET GLOBAL transaction_isolation = 'READ-COMMITTED';
```

グローバル `transaction_isolation` 値を設定すると、後続のすべてのセッションの分離レベルが設定されます。既存のセッションは影響を受けません。

セッションまたは次のレベルの `transaction_isolation` 値を設定するには、**SET** ステートメントを使用します。ほとんどのセッションシステム変数では、次のステートメントは値を設定する同等の方法です:

```
SET @@SESSION.var_name = value;
SET SESSION var_name = value;
SET var_name = value;
SET @@var_name = value;
```

前述のように、トランザクション分離レベルには、グローバルスコープとセッションスコープに加えて、次のトランザクションスコープがあります。次のトランザクションスコープを設定できるようにするために、セッションシステム変数値を割り当てるための **SET** 構文には、`transaction_isolation` の非標準セマンティクスがあります:

- セッション分離レベルを設定するには、次の構文のいずれかを使用します:

```
SET @@SESSION.transaction_isolation = value;
SET SESSION transaction_isolation = value;
SET transaction_isolation = value;
```

これらの構文ごとに、次のセマンティクスが適用されます:

- セッション内で実行される後続のすべてのトランザクションの分離レベルを設定します。
- トランザクション内で許可されますが、現在進行中のトランザクションには影響しません。
- トランザクション間で実行する場合は、次のトランザクション分離レベルを設定する前述のステートメントをオーバーライドします。
- SET SESSION TRANSACTION ISOLATION LEVEL** (`SESSION` キーワードを使用) に対応します。
- 次のトランザクション分離レベルを設定するには、次の構文を使用します:

```
SET @@transaction_isolation = value;
```

その構文には、次のセマンティクスが適用されます:

- 分離レベルは、セッション内で次に実行される単一トランザクションに対してのみ設定します。
- 後続のトランザクションは、セッション分離レベルに戻ります。
- トランザクション内では許可されません。
- SET TRANSACTION ISOLATION LEVEL** (`SESSION` キーワードなし) に対応します。

SET TRANSACTION および `transaction_isolation` システム変数との関係の詳細は、[セクション13.3.7「SET TRANSACTION ステートメント」](#) を参照してください。

- `transaction_prealloc_size`

コマンド行形式	<code>--transaction-prealloc-size=#</code>
システム変数	<code>transaction_prealloc_size</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer

デフォルト値	4096
最小値	1024
最大値	131072
ブロックサイズ	1024

トランザクションに関するさまざまな割り当てでメモリの取得元となる、トランザクションごとのメモリープールが存在します。プールのバイト単位の初期サイズは `transaction_prealloc_size` です。利用できるメモリーが不足しているためプールから十分に行えない各割り当てについて、プールは `transaction_alloc_block_size` バイトだけ増加されます。トランザクションが終了すると、プールは `transaction_prealloc_size` バイトに切り捨てられます。

単一トランザクション内のすべてのステートメントを含めるように `transaction_prealloc_size` を十分に大きくすると、多数の `malloc()` コールを避けることができます。

- `transaction_read_only`

コマンド行形式	<code>--transaction-read-only[={OFF ON}]</code>
システム変数	<code>transaction_read_only</code>
スコープ	グローバル、セッション
動的	はい
<code>SET_VAR</code> ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

トランザクションアクセスモード。値は、`OFF` (読取り/書き込み、デフォルト) または `ON` (読取り専用) です。

トランザクションアクセスモードには 3 つのスコープがあります: グローバル、セッションおよび次のトランザクション。この 3 スコープの実装は、あとで説明するように、いくつかの非標準のアクセスモード割り当てセマンティクスにつながります。

起動時にグローバルトランザクションアクセスモードを設定するには、`--transaction-read-only` サーバーオプションを使用します。

実行時には、`SET` ステートメントを使用して `transaction_read_only` システム変数に値を割り当てるか、`SET TRANSACTION` ステートメントを使用して間接的にアクセスモードを設定できます。たとえば、次の `SET` ステートメントを使用してグローバル値を設定します:

```
SET GLOBAL transaction_read_only = ON;
```

グローバル `transaction_read_only` 値を設定すると、後続のすべてのセッションのアクセスモードが設定されます。既存のセッションは影響を受けません。

セッションまたは次のレベルの `transaction_read_only` 値を設定するには、`SET` ステートメントを使用します。ほとんどのセッションシステム変数では、次のステートメントは値を設定する同等の方法です:

```
SET @@SESSION.var_name = value;
SET SESSION var_name = value;
SET var_name = value;
SET @@var_name = value;
```

前述のように、トランザクションアクセスモードには、グローバルスコープとセッションスコープに加えて、次のトランザクションスコープがあります。次のトランザクションスコープを設定できるようにするために、セッションシステム変数値を割り当てるための `SET` 構文には、`transaction_read_only` の非標準セマンティクスがあります。

- セッションアクセスモードを設定するには、次の構文のいずれかを使用します:

```
SET @@SESSION.transaction_read_only = value;
SET SESSION transaction_read_only = value;
```



```
SET transaction_read_only = value;
```

これらの構文ごとに、次のセマンティクスが適用されます:

- セッション内で実行される後続のすべてのトランザクションのアクセスモードを設定します。
- トランザクション内で許可されますが、現在進行中のトランザクションには影響しません。
- トランザクション間で実行する場合は、次のトランザクションアクセスモードを設定する前述のステートメントをオーバーライドします。
- `SET SESSION TRANSACTION {READ WRITE | READ ONLY}` (SESSION キーワードを使用) に対応します。
- 次のトランザクションアクセスモードを設定するには、次の構文を使用します:

```
SET @@transaction_read_only = value;
```

その構文には、次のセマンティクスが適用されます:

- セッション内で次に実行される単一トランザクションのアクセスモードのみを設定します。
- 後続のトランザクションは、セッションアクセスモードに戻ります。
- トランザクション内では許可されません。
- `SET TRANSACTION {READ WRITE | READ ONLY}` (SESSION キーワードなし) に対応します。

`SET TRANSACTION` および `transaction_read_only` システム変数との関係の詳細は、[セクション13.3.7「SET TRANSACTION ステートメント」](#) を参照してください。

- [unique_checks](#)

システム変数	unique_checks
スコープ	グローバル、セッション
動的	はい
<code>SET_VAR</code> ヒントの適用	はい
型	Boolean
デフォルト値	ON

1 に設定した場合 (デフォルト)、InnoDB テーブルのセカンダリインデックスの一貫性チェックが行われます。0 に設定した場合、ストレージエンジンでは、重複したキーが入力データに存在しないことが想定されます。一意性違反がデータにないことが確実にわかっている場合、これを 0 に設定して InnoDB への大きいテーブルのインポートを高速化できます。

この変数を 0 に設定しても、require ストレージエンジンは重複キーを無視しません。エンジンは引き続き、重複キーの存在を検査し、検出された場合に重複キーエラーが生成されます。

- [updatable_views_with_limit](#)

コマンド行形式	<code>--updatable-views-with-limit[={OFF ON}]</code>
システム変数	updatable_views_with_limit
スコープ	グローバル、セッション
動的	はい
<code>SET_VAR</code> ヒントの適用	はい
型	Boolean

デフォルト値	1
--------	---

この変数は、基礎テーブルで定義した主キーのすべてのカラムがビューに含まれていない場合に、更新ステートメントに `LIMIT` 句が含まれているとき、そのビューの更新を行えるかどうかを制御します。(このような更新は GUI ツールによって頻繁に生成されます。)更新は `UPDATE` または `DELETE` ステートメントのことです。ここでの主キーとは `PRIMARY KEY` か、カラムに `NULL` を含むことができない `UNIQUE` インデックスです。

変数は 2 つの値に設定できます。

- 1 または `YES`: 警告のみ発行します (エラーメッセージではない)。これはデフォルト値です。
- 0 または `NO`: 更新を禁止します。
- [use_secondary_engine](#)

導入	8.0.13
システム変数	use_secondary_engine
スコープ	セッション
動的	はい
<code>SET_VAR</code> ヒントの適用	はい
型	列挙
デフォルト値	<code>ON</code>
有効な値	<code>OFF</code> <code>ON</code> <code>FORCED</code>

将来使用します。

セカンダリエンジンを使用してクエリーを実行するかどうか。

HeatWave で使用します。 [MySQL HeatWave User Guide](#) を参照してください。

- [validate_password.xxx](#)

`validate_password` コンポーネントは、`validate_password.xxx` という形式の名前を持つ一連のシステム変数を実装します。これらの変数は、そのコンポーネントによるパスワードテストに影響します。 [セクション 6.4.3.2 「パスワード検証オプションおよび変数」](#) を参照してください。

- [validate_user_plugins](#)

コマンド行形式	<code>--validate-user-plugins[={OFF ON}]</code>
システム変数	validate_user_plugins
スコープ	グローバル
動的	いいえ
<code>SET_VAR</code> ヒントの適用	いいえ
型	Boolean
デフォルト値	<code>ON</code>

この変数が有効な場合 (デフォルト)、サーバーは各ユーザーアカウントを検査し、アカウントが使用できなくなる条件が検出された場合に警告を生成します。

- アカウントが、ロードされていない認証プラグインを必要としている。

- アカウントには `sha256_password` または `caching_sha2_password` 認証プラグインが必要ですが、プラグインで必要な SSL も RSA も有効にせずにサーバーが起動されました。

`validate_user_plugins` を有効にすると、サーバー初期化および `FLUSH PRIVILEGES` の速度が低下します。追加の検査が必要ない場合、この変数を起動時に無効化するとパフォーマンス低下を防ぐことができます。

- `version`

サーバーのバージョン番号。この値には、サーバーのビルドまたは構成情報を示す接尾辞が含まれることもあります。`-debug` は、デバッグサポートを有効にしてサーバーが構築されたことを示します。

- `version_comment`

システム変数	version_comment
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

CMake 構成プログラムには、MySQL の構築時にコメントを指定できる `COMPILATION_COMMENT_SERVER` オプションがあります。この変数は、そのコメントの値を格納します。(MySQL 8.0.14 より前は、`version_comment` は `COMPILATION_COMMENT` オプションによって設定されていました。) [セクション2.9.7「MySQL ソース構成オプション」](#) を参照してください。

- `version_compile_machine`

システム変数	version_compile_machine
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

サーバーバイナリのタイプ。

- `version_compile_os`

システム変数	version_compile_os
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

MySQL が構築されているオペレーティングシステムのタイプ。

- `version_compile_zlib`

システム変数	version_compile_zlib
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

コンパイルされた `zlib` ライブラリのバージョン。

- [wait_timeout](#)

コマンド行形式	<code>--wait-timeout=#</code>
システム変数	<code>wait_timeout</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	28800
最小値	1
最大値 (Windows)	2147483
最大値 (その他)	31536000

非インタラクティブな接続を閉じる前に、サーバーがその接続上でアクティビティを待機する秒数。

スレッド開始時に、セッションの `wait_timeout` 値は、`wait_timeout` グローバル値または `interactive_timeout` グローバル値で初期化されますが、いずれになるかはクライアントのタイプ (`mysql_real_connect()` に対する `CLIENT_INTERACTIVE` 接続オプションによって定義される) によって決まります。 `interactive_timeout` も参照してください。

- [warning_count](#)

メッセージを生成した最後のステートメントから得られたエラー、警告、および注意の数。この変数は読み取り専用です。 [セクション13.7.7.42「SHOW WARNINGS ステートメント」](#) を参照してください。

- [windowing_use_high_precision](#)

コマンド行形式	<code>--windowing-use-high-precision[={OFF ON}]</code>
システム変数	<code>windowing_use_high_precision</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Boolean
デフォルト値	ON

精度を失わずにウィンドウ操作を計算するかどうか。 [セクション8.2.1.21「ウィンドウ機能最適化」](#) を参照してください。

5.1.9 システム変数の使用

MySQL サーバーは、その操作を構成する多くのシステム変数を保持します。 [セクション5.1.8「サーバーシステム変数」](#) では、これらの変数の意味について説明します。各システム変数にはデフォルト値があります。システム変数は、コマンド行のオプションを使用するか、オプションファイルでサーバー起動時に設定できます。これらのほとんどは、`SET` ステートメントを使用してサーバーの実行中に動的に変更でき、これによりサーバーを停止して再起動することなくサーバーの動作を変更できます。式でシステム変数値を使用することもできます。

多くのシステム変数が組み込まれています。システム変数は、サーバープラグインまたはコンポーネントによってインストールすることもできます:

- サーバープラグインによって実装されたシステム変数は、プラグインのインストール時に公開され、プラグイン名で始まる名前を持ちます。たとえば、`audit_log` プラグインは `audit_log_policy` という名前のシステム変数を実装します。
- コンポーネントによって実装されたシステム変数は、コンポーネントのインストール時に公開され、コンポーネント固有の接頭辞で始まる名前を持ちます。たとえば、`log_filter_dragnet` エラーログフィルタコン

ポーネントは、`log_error_filter_rules` というシステム変数を実装します。このシステム変数のフルネームは `dragnet.log_error_filter_rules` です。この変数を参照するには、フルネームを使用します。

システム変数が存在するスコープは 2 つあります。グローバル変数は、サーバーの操作全体に影響します。セッション変数は、個々のクライアント接続の操作に影響します。所定のシステム変数は、グローバル値とセッション値の両方を持つことができます。グローバルシステム変数とセッションシステム変数は、次のように関連しています。

- サーバーが起動すると、各グローバル変数がデフォルト値に初期化されます。これらのデフォルトは、コマンド行で指定されるオプションまたはオプションファイルで変更できます。(セクション4.2.2「プログラムオプションの指定」を参照してください。)
- サーバーは、接続する各クライアントのセッション変数のセットも保持しています。クライアントのセッション変数は、対応するグローバル変数の現在の値を使用して、接続時に初期化されます。たとえば、クライアント SQL モードは、クライアントがグローバル `sql_mode` 値の値に接続するときに初期化されるセッション `sql_mode` 値によって制御されます。

一部のシステム変数では、対応するグローバル値からセッション値が初期化されません。初期化されている場合は、変数の説明に示されます。

システム変数値は、コマンド行のオプションまたはオプションファイルを使用すると、サーバー起動時にグローバルに設定できます。起動時のシステム変数の構文はコマンドオプションの構文と同じであるため、変数名内ではダッシュおよびアンダースコアを同じ意味で使用できます。たとえば、`--general_log=ON` と `--general-log=ON` は同等です。

起動オプションを使用して、数値を取る値を設定するとき、値には 1024 、 1024^2 または 1024^3 の倍数を示す、**K**、**M**、または **G** のサフィクス (大文字あるは小文字) を付けて指定でき、それぞれがキロバイト、メガバイト、またはギガバイトの単位を示します。MySQL 8.0.14 の時点では、 1024^4 、 1024^5 または 1024^6 の乗数を示すために、**T**、**P** および **E** も使用できます。したがって、次のコマンドは、16 メガバイトの `InnoDB` ログファイルサイズと最大パケットサイズを 1 ギガバイトにしてサーバーを起動します:

```
mysqld --innodb-log-file-size=16M --max-allowed-packet=1G
```

オプションファイル内で、これらの変数は次のように設定されます。

```
[mysqld]
innodb_log_file_size=16M
max_allowed_packet=1G
```

サフィクスの大文字、小文字の区別は問わず、`16M` と `16m`、`1G` と `1g` を同等とします。

`SET` ステートメントを使用してシステム変数を実行時に設定できる最大値を制限するには、サーバー起動時に `--maximum-var_name=value` 形式のオプションを使用して、この最大値を指定します。たとえば、実行時に `innodb_log_file_size` の値が 32MB を超えないようにするには、`--maximum-innodb-log-file-size=32M` オプションを使用します。

多くのシステム変数は動的であり、実行時に `SET` ステートメントを使用して変更できます。リストについては、セクション5.1.9.2「動的システム変数」を参照してください。 `SET` を使用してシステム変数を変更するには、その前に修飾子を付けて名前を参照します (オプション)。実行時に、システム変数名はダッシュではなくアンダースコアを使用して記述する必要があります。次の例は、この構文を簡単に示しています:

- グローバルシステム変数を設定します:

```
SET GLOBAL max_connections = 1000;
SET @@GLOBAL.max_connections = 1000;
```

- グローバルシステム変数を `mysqld-auto.cnf` ファイルに永続化 (およびランタイム値を設定) します:

```
SET PERSIST max_connections = 1000;
SET @@PERSIST.max_connections = 1000;
```

- グローバルシステム変数を (ランタイム値を設定せずに) `mysqld-auto.cnf` ファイルに永続化します:

```
SET PERSIST_ONLY back_log = 1000;
SET @@PERSIST_ONLY.back_log = 1000;
```

- セッションシステム変数を設定します:

```
SET SESSION sql_mode = 'TRADITIONAL';
SET @@SESSION.sql_mode = 'TRADITIONAL';
SET @@sql_mode = 'TRADITIONAL';
```

SET 構文の詳細は、[セクション13.7.6.1「変数代入の SET 構文」](#)を参照してください。システム変数を設定および永続化するための権限要件の詳細は、[セクション5.1.9.1「システム変数権限」](#)を参照してください

値乗数を指定するサフィクスは、サーバーの起動時に変数を設定するときに使用できますが、実行時に SET で値を設定するためには使用できません。一方、SET を使用すると、式を使用して変数の値を割り当てることができますが、サーバーの起動時に変数を設定するときには使用できません。たとえば、サーバーの起動時に次の 1 行目は有効ですが 2 行目は無効です。

```
shell> mysql --max_allowed_packet=16M
shell> mysql --max_allowed_packet=16*1024*1024
```

逆に、実行時に次の 2 行目は有効ですが 1 行目は無効です。

```
mysql> SET GLOBAL max_allowed_packet=16M;
mysql> SET GLOBAL max_allowed_packet=16*1024*1024;
```

注記

一部のシステム変数は、SET ステートメントで ON または 1 に設定することで有効化され、OFF または 0 に設定することで無効化されます。ただし、このような変数をコマンドラインまたはオプションファイルで設定するには、1 または 0 に設定する必要があります。ON または OFF に設定しても機能しません。たとえば、コマンド行において、--delay_key_write=1 は機能しますが、--delay_key_write=ON は機能しません。

システム変数名と値を表示するには、SHOW VARIABLES ステートメントを使用します。

```
mysql> SHOW VARIABLES;
+-----+-----+
| Variable_name | Value |
+-----+-----+
| auto_increment_increment | 1 |
| auto_increment_offset | 1 |
| automatic_sp_privileges | ON |
| back_log | 151 |
| basedir | /home/mysql/ |
| binlog_cache_size | 32768 |
| bulk_insert_buffer_size | 8388608 |
| character_set_client | utf8 |
| character_set_connection | utf8 |
| character_set_database | utf8mb4 |
| character_set_filesystem | binary |
| character_set_results | utf8 |
| character_set_server | utf8mb4 |
| character_set_system | utf8 |
| character_sets_dir | /home/mysql/share/mysqlCharsets/ |
| collation_connection | utf8_general_ci |
| collation_database | utf8mb4_0900_ai_ci |
| collation_server | utf8mb4_0900_ai_ci |
| ... | ... |
| innodb_autoextend_increment | 8 |
| innodb_buffer_pool_size | 8388608 |
| innodb_commit_concurrency | 0 |
| innodb_concurrency_tickets | 500 |
| innodb_data_file_path | ibdata1:10M:autoextend |
| innodb_data_home_dir | |
| ... | ... |
| version | 8.0.1-dmr-log |
| version_comment | Source distribution |
| version_compile_machine | i686 |
| version_compile_os | suse-linux |
| wait_timeout | 28800 |
+-----+-----+
```

LIKE 句では、ステートメントはパターンに一致する変数のみを表示します。特定の変数名を取得するには、LIKE 句を次のように使用します。


```
SHOW VARIABLES LIKE 'max_join_size';
SHOW SESSION VARIABLES LIKE 'max_join_size';
```

名前がパターンと一致する変数のリストを取得するには、**LIKE** 句の中で **%** のワイルドカード文字を使用します。

```
SHOW VARIABLES LIKE '%size%';
SHOW GLOBAL VARIABLES LIKE '%size%';
```

ワイルドカード文字は、照合されるパターン内のどの場所でも利用できます。厳密に言えば、**_**は単一の文字と一致するワイルドカードであるため、文字どおりに一致するように**_**としてエスケープする必要があります。実際には、これはほとんど必要ありません。

SHOW VARIABLES で **GLOBAL** および **SESSION** をいずれも指定しない場合、MySQL は **SESSION** 値を返します。

GLOBAL のみの変数を設定するときに **GLOBAL** キーワードを要求するが、取得時には要求しない理由は、将来の問題を回避するためです：

- **GLOBAL** 変数と同じ名前の **SESSION** 変数を削除しようとしたが、グローバル変数を変更するのに十分な権限を持つクライアントが、独自のセッションの **SESSION** 変数だけでなく、誤って **GLOBAL** 変数を変更する可能性があります。
- **GLOBAL** 変数と同じ名前でも **SESSION** 変数を追加した場合、**GLOBAL** 変数を変更しようとしているクライアントでは、独自の **SESSION** 変数のみが変更されている可能性があります。

5.1.9.1 システム変数権限

システム変数には、サーバー操作全体に影響するグローバル値、現在のセッションのみに影響するセッション値、またはその両方を指定できます：

- 動的システム変数の場合、**SET** ステートメントを使用してグローバルまたはセッションのランタイム値 (あるいはその両方) を変更し、現在のサーバーインスタンスの操作に影響を与えることができます。(動的変数の詳細は、[セクション5.1.9.2「動的システム変数」](#)を参照してください。)
- 特定のグローバルシステム変数では、**SET** を使用してその値をデータディレクトリ内の `mysqld-auto.cnf` ファイルに永続化し、後続の起動のためのサーバー操作に影響を与えることができます。(システム変数および `mysqld-auto.cnf` ファイルの永続化の詳細は、[セクション5.1.9.3「永続化されるシステム変数」](#)を参照してください。)
- 永続化されたグローバルシステム変数の場合、**RESET PERSIST** を使用してその値を `mysqld-auto.cnf` から削除し、後続の起動のサーバー操作に影響を与えることができます。

このセクションでは、実行時にシステム変数に値を割り当てる操作に必要な権限について説明します。これには、ランタイム値に影響する操作と、値を永続化する操作が含まれます。

グローバルシステム変数を設定するには、適切なキーワードを指定した **SET** ステートメントを使用します。次の権限が適用されます：

- グローバルシステム変数のランタイム値を設定するには、**SYSTEM_VARIABLES_ADMIN** 権限 (または非推奨の **SUPER** 権限) を必要とする **SET GLOBAL** ステートメントを使用します。
- グローバルシステム変数を `mysqld-auto.cnf` ファイルに永続化 (およびランタイム値を設定) するには、**SYSTEM_VARIABLES_ADMIN** または **SUPER** 権限を必要とする **SET PERSIST** ステートメントを使用します。
- グローバルシステム変数を `mysqld-auto.cnf` ファイルに永続化するには (ランタイム値を設定せず)、**SYSTEM_VARIABLES_ADMIN** および **PERSIST_RO_VARIABLES_ADMIN** 権限を必要とする **SET PERSIST_ONLY** ステートメントを使用します。**SET PERSIST_ONLY** は、動的システム変数と読み取り専用システム変数の両方に使用できますが、**SET PERSIST** を使用できない読み取り専用変数を永続化する場合に特に便利です。
- 一部のグローバルシステム変数は永続的に制限されます ([セクション5.1.9.4「永続的で永続的に制限されないシステム変数」](#)を参照)。これらの変数を永続化するには、前述の権限を必要とする **SET PERSIST_ONLY** ステートメントを使用します。また、暗号化された接続を使用してサーバーに接続し、`persist_only_admin_x509_subject` システム変数で指定されたサブジェクト値で SSL 証明書を指定する必要があります。

永続化されたグローバルシステム変数を `mysqld-auto.cnf` ファイルから削除するには、**RESET PERSIST** ステートメントを使用します。次の権限が適用されます：

- 動的システム変数の場合、`RESET PERSIST` には `SYSTEM_VARIABLES_ADMIN` または `SUPER` 権限が必要です。
- 読み取り専用システム変数の場合、`RESET PERSIST` には `SYSTEM_VARIABLES_ADMIN` および `PERSIST_RO_VARIABLES_ADMIN` 権限が必要です。
- 永続制限変数の場合、`RESET PERSIST` では、特定の SSL 証明書を使用して作成されたサーバーへの暗号化された接続は必要ありません。

グローバルシステム変数に前述の権限要件に対する例外がある場合、変数の説明にそれらの例外が示されます。例として、追加の権限を必要とする `default_table_encryption` や `mandatory_roles` があります。これらの追加権限は、グローバルランタイム値を設定する操作には適用されますが、値を永続化する操作には適用されません。

セッションシステム変数のランタイム値を設定するには、`SET SESSION` ステートメントを使用します。グローバルランタイム値の設定とは対照的に、セッションランタイム値の設定には通常、特別な権限は必要なく、すべてのユーザーが現在のセッションに影響を与えることができます。一部のシステム変数では、セッション値を設定すると現在のセッションの外部に影響を与える可能性があるため、特殊な権限を持つユーザーのみが実行できる制限付き操作です:

- MySQL 8.0.14 では、必要な権限は `SESSION_VARIABLES_ADMIN` です。

注記

`SYSTEM_VARIABLES_ADMIN` または `SUPER` を持つユーザーは、効果的に `SESSION_VARIABLES_ADMIN` を意味するため、`SESSION_VARIABLES_ADMIN` を明示的に付与する必要はありません。

- MySQL 8.0.14 より前は、必要な権限は `SYSTEM_VARIABLES_ADMIN` または `SUPER` です。

セッションシステム変数が制限されている場合、変数の説明にその制限が示されます。たとえば、`binlog_format` や `sql_log_bin` などです。これらの変数のセッション値を設定すると、現在のセッションのバイナリロギングに影響しますが、サーバーアプリケーションおよびバックアップの整合性にも大きな影響を与える可能性があります。

`SESSION_VARIABLES_ADMIN` を使用すると、管理者は、制限付きセッションシステム変数を変更できるようにするために、以前に `SYSTEM_VARIABLES_ADMIN` または `SUPER` を付与されたユーザーの権限フットプリントを最小限に抑えることができます。管理者が、制限付きセッションシステム変数を設定する機能を付与するために、次のロールを作成したとします:

```
CREATE ROLE set_session_sysvars;  
GRANT SYSTEM_VARIABLES_ADMIN ON *.* TO set_session_sysvars;
```

`set_session_sysvars` ロールを付与されたユーザー（およびそのロールがアクティブなユーザー）は、制限付きセッションシステム変数を設定できます。ただし、そのユーザーはグローバルシステム変数を設定することもできますが、これは望ましくない場合があります。

`SYSTEM_VARIABLES_ADMIN` のかわりに `SESSION_VARIABLES_ADMIN` を使用するようにロールを変更することで、ロール権限を制限されたセッションシステム変数を設定する機能まで減らすことができます。ロールを変更するには、次のステートメントを使用します:

```
GRANT SESSION_VARIABLES_ADMIN ON *.* TO set_session_sysvars;  
REVOKE SYSTEM_VARIABLES_ADMIN ON *.* FROM set_session_sysvars;
```

ロールを変更するとすぐに有効になります: `set_session_sysvars` ロールを付与されたアカウントには `SYSTEM_VARIABLES_ADMIN` がなく、その権限が明示的に付与されないうりグローバルシステム変数を設定できません。同様の `GRANT/REVOKE` 順序は、ロールによってではなく、`SYSTEM_VARIABLES_ADMIN` が直接付与されたアカウントに適用できます。

5.1.9.2 動的システム変数

多くのサーバーシステム変数は動的であり、実行時に設定できます。セクション13.7.6.1「変数代入の SET 構文」を参照してください。システム変数を設定するための権限要件の詳細は、セクション5.1.9.1「システム変数権限」を参照してください

次のテーブルに、`mysqld` 内で適用可能なすべての動的システム変数を示します。

このテーブルは、各変数のデータ型とスコープを示しています。最後のカラムは、各変数のスコープがグローバル、セッション、またはその両方のいずれであるかを示します。変数の設定および使用の詳細は、対応するアイテムの説明を参照してください。必要に応じて、アイテムに関する詳細情報へのダイレクトリンクが提供されます。

「string」型の変数は文字列値を取ります。「numeric」型の変数は数値を取ります。「boolean」型の変数は、0、1、ON、OFF に設定できます。「enumeration」と記載されている変数は、通常はその変数に対して使用可能ないずれかの値に設定しますが、目的とする列挙値に相当する数値も設定できます。列挙されたシステム変数について、最初の列挙値は 0 になります。これは、最初の列挙値が 1 に対応するテーブルのカラムに使用される ENUM データ型とは異なります。

表 5.4 「動的システム変数サマリー」

変数名	変数型	変数スコープ
activate_all_roles_on_login	Boolean	グローバル
admin_ssl_ca	ファイル名	グローバル
admin_ssl_capath	ディレクトリ名	グローバル
admin_ssl_cert	ファイル名	グローバル
admin_ssl_cipher	文字列	グローバル
admin_ssl_crl	ファイル名	グローバル
admin_ssl_crlpath	ディレクトリ名	グローバル
admin_ssl_key	ファイル名	グローバル
admin_tls_ciphersuites	文字列	グローバル
admin_tls_version	文字列	グローバル
audit_log_connection_policy	列挙	グローバル
audit_log_exclude_accounts	文字列	グローバル
audit_log_flush	Boolean	グローバル
audit_log_include_accounts	文字列	グローバル
audit_log_password_history_keep_days	Integer	グローバル
audit_log_prune_seconds	Integer	グローバル
audit_log_read_buffer_size	Integer	異なる
audit_log_rotate_on_size	Integer	グローバル
audit_log_statement_policy	列挙	グローバル
authentication_ldap_sasl_auth_method	文字列	グローバル
authentication_ldap_sasl_bind_base_dn	文字列	グローバル
authentication_ldap_sasl_bind_root_dn	文字列	グローバル
authentication_ldap_sasl_bind_root_pwd	文字列	グローバル
authentication_ldap_sasl_ca_path	文字列	グローバル
authentication_ldap_sasl_group_search	文字列	グローバル
authentication_ldap_sasl_group_search	文字列	グローバル
authentication_ldap_sasl_init_pool_size	Integer	グローバル
authentication_ldap_sasl_log_status	Integer	グローバル
authentication_ldap_sasl_max_pool_size	Integer	グローバル
authentication_ldap_sasl_referral	Boolean	グローバル
authentication_ldap_sasl_server_host	文字列	グローバル
authentication_ldap_sasl_server_port	Integer	グローバル
authentication_ldap_sasl_tls	Boolean	グローバル

変数名	変数型	変数スコープ
authentication_ldap_sasl_user_search_base	文字列	グローバル
authentication_ldap_simple_auth_method	文字列	グローバル
authentication_ldap_simple_bind_base	文字列	グローバル
authentication_ldap_simple_bind_root_dn	文字列	グローバル
authentication_ldap_simple_bind_root_password	文字列	グローバル
authentication_ldap_simple_ca_path	文字列	グローバル
authentication_ldap_simple_group_search_base	文字列	グローバル
authentication_ldap_simple_group_search_filter	文字列	グローバル
authentication_ldap_simple_init_pool_size	Integer	グローバル
authentication_ldap_simple_log_status	Integer	グローバル
authentication_ldap_simple_max_pool_size	Integer	グローバル
authentication_ldap_simple_referral	Boolean	グローバル
authentication_ldap_simple_server_host	文字列	グローバル
authentication_ldap_simple_server_port	Integer	グローバル
authentication_ldap_simple_tls	Boolean	グローバル
authentication_ldap_simple_user_search_base	文字列	グローバル
auto_increment_increment	Integer	両方
auto_increment_offset	Integer	両方
autocommit	Boolean	両方
automatic_sp_privileges	Boolean	グローバル
avoid_temporal_upgrade	Boolean	グローバル
big_tables	Boolean	両方
binlog_cache_size	Integer	グローバル
binlog_checksum	文字列	グローバル
binlog_direct_non_transactional_updates	Boolean	両方
binlog_encryption	Boolean	グローバル
binlog_error_action	列挙	グローバル
binlog_expire_logs_seconds	Integer	グローバル
binlog_format	列挙	両方
binlog_group_commit_sync_delay	Integer	グローバル
binlog_group_commit_sync_no_delay_count	Integer	グローバル
binlog_max_flush_queue_time	Integer	グローバル
binlog_order_commits	Boolean	グローバル
binlog_row_image	列挙	両方
binlog_row_metadata	列挙	グローバル
binlog_row_value_options	Set	両方
binlog_rows_query_log_events	Boolean	両方
binlog_stmt_cache_size	Integer	グローバル
binlog_transaction_compression	Boolean	グローバル
binlog_transaction_compression_level	Integer	グローバル
binlog_transaction_dependency_history_size	Integer	グローバル

変数名	変数型	変数スコープ
binlog_transaction_dependency_tracking	列挙	グローバル
block_encryption_mode	文字列	両方
bulk_insert_buffer_size	Integer	両方
character_set_client	文字列	両方
character_set_connection	文字列	両方
character_set_database	文字列	両方
character_set_filesystem	文字列	両方
character_set_results	文字列	両方
character_set_server	文字列	両方
check_proxy_users	Boolean	グローバル
clone_autotune_concurrency	Boolean	グローバル
clone_buffer_size	Integer	グローバル
clone_ddl_timeout	Integer	グローバル
clone_enable_compression	Boolean	グローバル
clone_max_concurrency	Integer	グローバル
clone_max_data_bandwidth	Integer	グローバル
clone_max_network_bandwidth	Integer	グローバル
clone_ssl_ca	ファイル名	グローバル
clone_ssl_cert	ファイル名	グローバル
clone_ssl_key	ファイル名	グローバル
clone_valid_donor_list	文字列	グローバル
collation_connection	文字列	両方
collation_database	文字列	両方
collation_server	文字列	両方
completion_type	列挙	両方
concurrent_insert	列挙	グローバル
connect_timeout	Integer	グローバル
connection_control_failed_connections_threshold	Integer	グローバル
connection_control_max_connection_delay	Integer	グローバル
connection_control_min_connection_delay	Integer	グローバル
cte_max_recursion_depth	Integer	両方
debug	文字列	両方
debug_sync	文字列	セッション
default_collation_for_utf8mb4	列挙	両方
default_password_lifetime	Integer	グローバル
default_storage_engine	列挙	両方
default_table_encryption	Boolean	両方
default_tmp_storage_engine	列挙	両方
default_week_format	Integer	両方
delay_key_write	列挙	グローバル
delayed_insert_limit	Integer	グローバル

変数名	変数型	変数スコープ
delayed_insert_timeout	Integer	グローバル
delayed_queue_size	Integer	グローバル
div_precision_increment	Integer	両方
dragnet.log_error_filter_rules	文字列	グローバル
end_markers_in_json	Boolean	両方
enforce_gtid_consistency	列挙	グローバル
eq_range_index_dive_limit	Integer	両方
event_scheduler	列挙	グローバル
expire_logs_days	Integer	グローバル
explicit_defaults_for_timestamp	Boolean	両方
flush	Boolean	グローバル
flush_time	Integer	グローバル
foreign_key_checks	Boolean	両方
ft_boolean_syntax	文字列	グローバル
general_log	Boolean	グローバル
general_log_file	ファイル名	グローバル
generated_random_password_length	Integer	両方
group_concat_max_len	Integer	両方
group_replication_advertise_recovery_ev	文字列	グローバル
group_replication_allow_local_lower_ver	Boolean	グローバル
group_replication_auto_increment_incre	Integer	グローバル
group_replication_autorejoin_tries	Integer	グローバル
group_replication_bootstrap_group	Boolean	グローバル
group_replication_clone_threshold	Integer	グローバル
group_replication_communication_debug	文字列	グローバル
group_replication_communication_max	Integer	グローバル
group_replication_components_stop_tim	Integer	グローバル
group_replication_compression_threshol	Integer	グローバル
group_replication_consistency	列挙	両方
group_replication_enforce_update_ever	Boolean	グローバル
group_replication_exit_state_action	列挙	グローバル
group_replication_flow_control_applier	Integer	グローバル
group_replication_flow_control_certifier	Integer	グローバル
group_replication_flow_control_hold_per	Integer	グローバル
group_replication_flow_control_max_con	Integer	グローバル
group_replication_flow_control_member	Integer	グローバル
group_replication_flow_control_min_quo	Integer	グローバル
group_replication_flow_control_min_rec	Integer	グローバル
group_replication_flow_control_mode	列挙	グローバル
group_replication_flow_control_period	Integer	グローバル
group_replication_flow_control_release	Integer	グローバル

変数名	変数型	変数スコープ
group_replication_force_members	文字列	グローバル
group_replication_group_name	文字列	グローバル
group_replication_group_seeds	文字列	グローバル
group_replication_gtid_assignment_block_increment	Integer	グローバル
group_replication_ip_allowlist	文字列	グローバル
group_replication_ip_whitelist	文字列	グローバル
group_replication_local_address	文字列	グローバル
group_replication_member_expel_timeout	Integer	グローバル
group_replication_member_weight	Integer	グローバル
group_replication_message_cache_size	Integer	グローバル
group_replication_poll_spin_loops	Integer	グローバル
group_replication_recovery_complete_at	列挙	グローバル
group_replication_recovery_compression_algorithm	Set Algorithm	グローバル
group_replication_recovery_get_public_key	Boolean	グローバル
group_replication_recovery_public_key_path	ファイル名	グローバル
group_replication_recovery_reconnect_timeout	Integer	グローバル
group_replication_recovery_retry_count	Integer	グローバル
group_replication_recovery_ssl_ca	文字列	グローバル
group_replication_recovery_ssl_capath	文字列	グローバル
group_replication_recovery_ssl_cert	文字列	グローバル
group_replication_recovery_ssl_cipher	文字列	グローバル
group_replication_recovery_ssl_crl	ファイル名	グローバル
group_replication_recovery_ssl_crlpath	ディレクトリ名	グローバル
group_replication_recovery_ssl_key	文字列	グローバル
group_replication_recovery_ssl_verify_mode	Boolean	グローバル
group_replication_recovery_tls_cipher_suite	文字列	グローバル
group_replication_recovery_tls_version	文字列	グローバル
group_replication_recovery_use_ssl	Boolean	グローバル
group_replication_recovery_zstd_compression_level	Integer	グローバル
group_replication_single_primary_mode	Boolean	グローバル
group_replication_ssl_mode	列挙	グローバル
group_replication_start_on_boot	Boolean	グローバル
group_replication_tls_source	列挙	グローバル
group_replication_transaction_size_limit	Integer	グローバル
group_replication_unreachable_majority_timeout	Integer	グローバル
gtid_executed_compression_period	Integer	グローバル
gtid_mode	列挙	グローバル
gtid_next	列挙	セッション
gtid_purged	文字列	グローバル
histogram_generation_max_mem_size	Integer	両方
host_cache_size	Integer	グローバル

変数名	変数型	変数スコープ
identity	Integer	セッション
immediate_server_version	Integer	セッション
information_schema_stats_expiry	Integer	両方
init_connect	文字列	グローバル
init_slave	文字列	グローバル
innodb_adaptive_flushing	Boolean	グローバル
innodb_adaptive_flushing_lwm	Integer	グローバル
innodb_adaptive_hash_index	Boolean	グローバル
innodb_adaptive_max_sleep_delay	Integer	グローバル
innodb_api_bk_commit_interval	Integer	グローバル
innodb_api_trx_level	Integer	グローバル
innodb_autoextend_increment	Integer	グローバル
innodb_background_drop_list_empty	Boolean	グローバル
innodb_buffer_pool_dump_at_shutdown	Boolean	グローバル
innodb_buffer_pool_dump_now	Boolean	グローバル
innodb_buffer_pool_dump_pct	Integer	グローバル
innodb_buffer_pool_filename	ファイル名	グローバル
innodb_buffer_pool_in_core_file	Boolean	グローバル
innodb_buffer_pool_load_abort	Boolean	グローバル
innodb_buffer_pool_load_now	Boolean	グローバル
innodb_buffer_pool_size	Integer	グローバル
innodb_change_buffer_max_size	Integer	グローバル
innodb_change_buffering	列挙	グローバル
innodb_change_buffering_debug	Integer	グローバル
innodb_checkpoint_disabled	Boolean	グローバル
innodb_checksum_algorithm	列挙	グローバル
innodb_cmp_per_index_enabled	Boolean	グローバル
innodb_commit_concurrency	Integer	グローバル
innodb_compress_debug	列挙	グローバル
innodb_compression_failure_threshold	Integer	グローバル
innodb_compression_level	Integer	グローバル
innodb_compression_pad_pct_max	Integer	グローバル
innodb_concurrency_tickets	Integer	グローバル
innodb_ddl_log_crash_reset_debug	Boolean	グローバル
innodb_deadlock_detect	Boolean	グローバル
innodb_default_row_format	列挙	グローバル
innodb_disable_sort_file_cache	Boolean	グローバル
innodb_extend_and_initialize	Boolean	グローバル
innodb_fast_shutdown	Integer	グローバル
innodb_fil_make_page_dirty_debug	Integer	グローバル
innodb_file_per_table	Boolean	グローバル

変数名	変数型	変数スコープ
innodb_fill_factor	Integer	グローバル
innodb_flush_log_at_timeout	Integer	グローバル
innodb_flush_log_at_trx_commit	列挙	グローバル
innodb_flush_neighbors	列挙	グローバル
innodb_flush_sync	Boolean	グローバル
innodb_flushing_avg_loops	Integer	グローバル
innodb_fsync_threshold	Integer	グローバル
innodb_ft_aux_table	文字列	グローバル
innodb_ft_enable_diag_print	Boolean	グローバル
innodb_ft_enable_stopword	Boolean	両方
innodb_ft_num_word_optimize	Integer	グローバル
innodb_ft_result_cache_limit	Integer	グローバル
innodb_ft_server_stopword_table	文字列	グローバル
innodb_ft_user_stopword_table	文字列	両方
innodb_idle_flush_pct	Integer	グローバル
innodb_io_capacity	Integer	グローバル
innodb_io_capacity_max	Integer	グローバル
innodb_limit_optimistic_insert_debug	Integer	グローバル
innodb_lock_wait_timeout	Integer	両方
innodb_log_buffer_size	Integer	グローバル
innodb_log_checkpoint_fuzzy_now	Boolean	グローバル
innodb_log_checkpoint_now	Boolean	グローバル
innodb_log_checksums	Boolean	グローバル
innodb_log_compressed_pages	Boolean	グローバル
innodb_log_spin_cpu_abs_lwm	Integer	グローバル
innodb_log_spin_cpu_pct_hwm	Integer	グローバル
innodb_log_wait_for_flush_spin_hwm	Integer	グローバル
innodb_log_write_ahead_size	Integer	グローバル
innodb_log_writer_threads	Boolean	グローバル
innodb_lru_scan_depth	Integer	グローバル
innodb_max_dirty_pages_pct	数値	グローバル
innodb_max_dirty_pages_pct_lwm	数値	グローバル
innodb_max_purge_lag	Integer	グローバル
innodb_max_purge_lag_delay	Integer	グローバル
innodb_max_undo_log_size	Integer	グローバル
innodb_merge_threshold_set_all_debug	Integer	グローバル
innodb_monitor_disable	文字列	グローバル
innodb_monitor_enable	文字列	グローバル
innodb_monitor_reset	列挙	グローバル
innodb_monitor_reset_all	列挙	グローバル
innodb_old_blocks_pct	Integer	グローバル

変数名	変数型	変数スコープ
innodb_old_blocks_time	Integer	グローバル
innodb_online_alter_log_max_size	Integer	グローバル
innodb_optimize_fulltext_only	Boolean	グローバル
innodb_parallel_read_threads	Integer	セッション
innodb_print_all_deadlocks	Boolean	グローバル
innodb_print_ddl_logs	Boolean	グローバル
innodb_purge_batch_size	Integer	グローバル
innodb_purge_rseg_truncate_frequency	Integer	グローバル
innodb_random_read_ahead	Boolean	グローバル
innodb_read_ahead_threshold	Integer	グローバル
innodb_redo_log_archive_dirs	文字列	グローバル
innodb_redo_log_encrypt	Boolean	グローバル
innodb_replication_delay	Integer	グローバル
innodb_rollback_segments	Integer	グローバル
innodb_saved_page_number_debug	Integer	グローバル
innodb_spin_wait_delay	Integer	グローバル
innodb_spin_wait_pause_multiplier	Integer	グローバル
innodb_stats_auto_recalc	Boolean	グローバル
innodb_stats_include_delete_marked	Boolean	グローバル
innodb_stats_method	列挙	グローバル
innodb_stats_on_metadata	Boolean	グローバル
innodb_stats_persistent	Boolean	グローバル
innodb_stats_persistent_sample_pages	Integer	グローバル
innodb_stats_transient_sample_pages	Integer	グローバル
innodb_status_output	Boolean	グローバル
innodb_status_output_locks	Boolean	グローバル
innodb_strict_mode	Boolean	両方
innodb_sync_spin_loops	Integer	グローバル
innodb_table_locks	Boolean	両方
innodb_thread_concurrency	Integer	グローバル
innodb_thread_sleep_delay	Integer	グローバル
innodb_tmpdir	ディレクトリ名	両方
innodb_trx_purge_view_update_only_debug	Boolean	グローバル
innodb_trx_rseg_n_slots_debug	Integer	グローバル
innodb_undo_log_encrypt	Boolean	グローバル
innodb_undo_log_truncate	Boolean	グローバル
innodb_undo_tablespaces	Integer	グローバル
insert_id	Integer	セッション
interactive_timeout	Integer	両方
internal_tmp_disk_storage_engine	列挙	グローバル
internal_tmp_mem_storage_engine	列挙	両方

変数名	変数型	変数スコープ
join_buffer_size	Integer	両方
keep_files_on_create	Boolean	両方
key_buffer_size	Integer	グローバル
key_cache_age_threshold	Integer	グローバル
key_cache_block_size	Integer	グローバル
key_cache_division_limit	Integer	グローバル
keyring_aws_cmek_id	文字列	グローバル
keyring_aws_region	列挙	グローバル
keyring_encrypted_file_data	ファイル名	グローバル
keyring_encrypted_file_password	文字列	グローバル
keyring_file_data	ファイル名	グローバル
keyring_hashicorp_auth_path	文字列	グローバル
keyring_hashicorp_ca_path	ファイル名	グローバル
keyring_hashicorp_caching	Boolean	グローバル
keyring_hashicorp_role_id	文字列	グローバル
keyring_hashicorp_secret_id	文字列	グローバル
keyring_hashicorp_server_url	文字列	グローバル
keyring_hashicorp_store_path	文字列	グローバル
keyring_okv_conf_dir	ディレクトリ名	グローバル
keyring_operations	Boolean	グローバル
last_insert_id	Integer	セッション
lc_messages	文字列	両方
lc_time_names	文字列	両方
local_infile	Boolean	グローバル
lock_wait_timeout	Integer	両方
log_bin_trust_function_creators	Boolean	グローバル
log_error_services	文字列	グローバル
log_error_suppression_list	文字列	グローバル
log_error_verbosity	Integer	グローバル
log_output	Set	グローバル
log_queries_not_using_indexes	Boolean	グローバル
log_raw	Boolean	グローバル
log_slow_admin_statements	Boolean	グローバル
log_slow_extra	Boolean	グローバル
log_slow_slave_statements	Boolean	グローバル
log_statements_unsafe_for_binlog	Boolean	グローバル
log_syslog	Boolean	グローバル
log_syslog_facility	文字列	グローバル
log_syslog_include_pid	Boolean	グローバル
log_syslog_tag	文字列	グローバル
log_throttle_queries_not_using_indexes	Integer	グローバル

変数名	変数型	変数スコープ
log_timestamps	列挙	グローバル
long_query_time	数値	両方
low_priority_updates	Boolean	両方
mandatory_roles	文字列	グローバル
master_info_repository	文字列	グローバル
master_verify_checksum	Boolean	グローバル
max_allowed_packet	Integer	両方
max_binlog_cache_size	Integer	グローバル
max_binlog_size	Integer	グローバル
max_binlog_stmt_cache_size	Integer	グローバル
max_connect_errors	Integer	グローバル
max_connections	Integer	グローバル
max_delayed_threads	Integer	両方
max_error_count	Integer	両方
max_execution_time	Integer	両方
max_heap_table_size	Integer	両方
max_insert_delayed_threads	Integer	両方
max_join_size	Integer	両方
max_length_for_sort_data	Integer	両方
max_points_in_geometry	Integer	両方
max_prepared_stmt_count	Integer	グローバル
max_relay_log_size	Integer	グローバル
max_seeks_for_key	Integer	両方
max_sort_length	Integer	両方
max_sp_recursion_depth	Integer	両方
max_user_connections	Integer	両方
max_write_lock_count	Integer	グローバル
min_examined_row_limit	Integer	両方
myisam_data_pointer_size	Integer	グローバル
myisam_max_sort_file_size	Integer	グローバル
myisam_repair_threads	Integer	両方
myisam_sort_buffer_size	Integer	両方
myisam_stats_method	列挙	両方
myisam_use_mmap	Boolean	グローバル
mysql_firewall_mode	Boolean	グローバル
mysql_firewall_trace	Boolean	グローバル
mysql_native_password_proxy_users	Boolean	グローバル
mysqlx_compression_algorithms	Set	グローバル
mysqlx_connect_timeout	Integer	グローバル
mysqlx_deflate_default_compression_level	Integer	グローバル
mysqlx_deflate_max_client_compression_level	Integer	グローバル

変数名	変数型	変数スコープ
mysqlx_document_id_unique_prefix	Integer	グローバル
mysqlx_enable_hello_notice	Boolean	グローバル
mysqlx_idle_worker_thread_timeout	Integer	グローバル
mysqlx_interactive_timeout	Integer	グローバル
mysqlx_lz4_default_compression_level	Integer	グローバル
mysqlx_lz4_max_client_compression_level	Integer	グローバル
mysqlx_max_allowed_packet	Integer	グローバル
mysqlx_max_connections	Integer	グローバル
mysqlx_min_worker_threads	Integer	グローバル
mysqlx_read_timeout	Integer	セッション
mysqlx_wait_timeout	Integer	セッション
mysqlx_write_timeout	Integer	セッション
mysqlx_zstd_default_compression_level	Integer	グローバル
mysqlx_zstd_max_client_compression_level	Integer	グローバル
ndb_allow_copying_alter_table	Boolean	両方
ndb_autoincrement_prefetch_sz	Integer	両方
ndb_blob_read_batch_bytes	Integer	両方
ndb_blob_write_batch_bytes	Integer	両方
ndb_cache_check_time	Integer	グローバル
ndb_clear_apply_status	Boolean	グローバル
ndb_conflict_role	列挙	グローバル
ndb_data_node_neighbour	Integer	グローバル
ndb_dbg_check_shares	Integer	両方
ndb_default_column_format	列挙	グローバル
ndb_default_column_format	列挙	グローバル
ndb_deferred_constraints	Integer	両方
ndb_deferred_constraints	Integer	両方
ndb_distribution	列挙	グローバル
ndb_distribution	列挙	グローバル
ndb_eventbuffer_free_percent	Integer	グローバル
ndb_eventbuffer_max_alloc	Integer	グローバル
ndb_extra_logging	Integer	グローバル
ndb_force_send	Boolean	両方
ndb_fully_replicated	Boolean	両方
ndb_index_stat_enable	Boolean	両方
ndb_index_stat_option	文字列	両方
ndb_join_pushdown	Boolean	両方
ndb_log_bin	Boolean	両方
ndb_log_binlog_index	Boolean	グローバル
ndb_log_empty_epochs	Boolean	グローバル
ndb_log_empty_epochs	Boolean	グローバル

変数名	変数型	変数スコープ
ndb_log_empty_update	Boolean	グローバル
ndb_log_empty_update	Boolean	グローバル
ndb_log_exclusive_reads	Boolean	両方
ndb_log_exclusive_reads	Boolean	両方
ndb_log_update_as_write	Boolean	グローバル
ndb_log_update_minimal	Boolean	グローバル
ndb_log_updated_only	Boolean	グローバル
ndb_metadata_check	Boolean	グローバル
ndb_metadata_check_interval	Integer	グローバル
ndb_metadata_sync	Boolean	グローバル
ndb_optimization_delay	Integer	グローバル
ndb_read_backup	Boolean	グローバル
ndb_recv_thread_activation_threshold	Integer	グローバル
ndb_recv_thread_cpu_mask	ビットマップ	グローバル
ndb_report_thresh_binlog_epoch_slip	Integer	グローバル
ndb_report_thresh_binlog_mem_usage	Integer	グローバル
ndb_row_checksum	Integer	両方
ndb_schema_dist_lock_wait_timeout	Integer	グローバル
ndb_show_foreign_key_mock_tables	Boolean	グローバル
ndb_slave_conflict_role	列挙	グローバル
ndb_table_no_logging	Boolean	セッション
ndb_table_temporary	Boolean	セッション
ndb_use_exact_count	Boolean	両方
ndb_use_transactions	Boolean	両方
ndbinfo_max_bytes	Integer	両方
ndbinfo_max_rows	Integer	両方
ndbinfo_offline	Boolean	グローバル
ndbinfo_show_hidden	Boolean	両方
net_buffer_length	Integer	両方
net_read_timeout	Integer	両方
net_retry_count	Integer	両方
net_write_timeout	Integer	両方
new	Boolean	両方
offline_mode	Boolean	グローバル
old_alter_table	Boolean	両方
optimizer_prune_level	Integer	両方
optimizer_search_depth	Integer	両方
optimizer_switch	Set	両方
optimizer_trace	文字列	両方
optimizer_trace_features	文字列	両方
optimizer_trace_limit	Integer	両方

変数名	変数型	変数スコープ
optimizer_trace_max_mem_size	Integer	両方
optimizer_trace_offset	Integer	両方
original_commit_timestamp	数値	セッション
original_server_version	Integer	セッション
parser_max_mem_size	Integer	両方
partial_revokes	Boolean	グローバル
password_history	Integer	グローバル
password_require_current	Boolean	グローバル
password_reuse_interval	Integer	グローバル
performance_schema_max_digest_sample_size	Integer	グローバル
performance_schema_show_processlist	Boolean	グローバル
preload_buffer_size	Integer	両方
print_identified_with_as_hex	Boolean	両方
profiling	Boolean	両方
profiling_history_size	Integer	両方
protocol_compression_algorithms	Set	グローバル
pseudo_slave_mode	Boolean	セッション
pseudo_thread_id	Integer	セッション
query_alloc_block_size	Integer	両方
query_prealloc_size	Integer	両方
rand_seed1	Integer	セッション
rand_seed2	Integer	セッション
range_alloc_block_size	Integer	両方
range_optimizer_max_mem_size	Integer	両方
rbr_exec_mode	列挙	両方
read_buffer_size	Integer	両方
read_only	Boolean	グローバル
read_rnd_buffer_size	Integer	両方
regexp_stack_limit	Integer	グローバル
regexp_time_limit	Integer	グローバル
relay_log_info_repository	文字列	グローバル
relay_log_purge	Boolean	グローバル
replication_optimize_for_static_plugin_configuration	Boolean	グローバル
replication_sender_observe_commit_only	Boolean	グローバル
require_row_format	Boolean	セッション
require_secure_transport	Boolean	グローバル
resultset_metadata	列挙	セッション
rewriter_enabled	Boolean	グローバル
rewriter_verbose	Integer	グローバル
rpl_read_size	Integer	グローバル
rpl_semi_sync_master_enabled	Boolean	グローバル

変数名	変数型	変数スコープ
rpl_semi_sync_master_timeout	Integer	グローバル
rpl_semi_sync_master_trace_level	Integer	グローバル
rpl_semi_sync_master_wait_for_slave_completion	Integer	グローバル
rpl_semi_sync_master_wait_no_slave	Boolean	グローバル
rpl_semi_sync_master_wait_point	列挙	グローバル
rpl_semi_sync_slave_enabled	Boolean	グローバル
rpl_semi_sync_slave_trace_level	Integer	グローバル
rpl_stop_slave_timeout	Integer	グローバル
schema_definition_cache	Integer	グローバル
secondary_engine_cost_threshold	数値	セッション
select_into_buffer_size	Integer	両方
select_into_disk_sync	Boolean	両方
select_into_disk_sync_delay	Integer	両方
server_id	Integer	グローバル
session_track_gtid	列挙	両方
session_track_schema	Boolean	両方
session_track_state_change	Boolean	両方
session_track_system_variables	文字列	両方
session_track_transaction_info	列挙	両方
sha256_password_proxy_users	Boolean	グローバル
show_create_table_skip_secondary_engine	Boolean	セッション
show_create_table_verbosity	Boolean	両方
show_old_temporals	Boolean	両方
slave_allow_batching	Boolean	グローバル
slave_checkpoint_group	Integer	グローバル
slave_checkpoint_period	Integer	グローバル
slave_compressed_protocol	Boolean	グローバル
slave_exec_mode	列挙	グローバル
slave_max_allowed_packet	Integer	グローバル
slave_net_timeout	Integer	グローバル
slave_parallel_type	列挙	グローバル
slave_parallel_workers	Integer	グローバル
slave_pending_jobs_size_max	Integer	グローバル
slave_preserve_commit_order	Boolean	グローバル
slave_rows_search_algorithms	Set	グローバル
slave_sql_verify_checksum	Boolean	グローバル
slave_transaction_retries	Integer	グローバル
slave_type_conversions	Set	グローバル
slow_launch_time	Integer	グローバル
slow_query_log	Boolean	グローバル
slow_query_log_file	ファイル名	グローバル

変数名	変数型	変数スコープ
sort_buffer_size	Integer	両方
sql_auto_is_null	Boolean	両方
sql_big_selects	Boolean	両方
sql_buffer_result	Boolean	両方
sql_log_bin	Boolean	セッション
sql_log_off	Boolean	両方
sql_mode	Set	両方
sql_notes	Boolean	両方
sql_quote_show_create	Boolean	両方
sql_require_primary_key	Boolean	両方
sql_safe_updates	Boolean	両方
sql_select_limit	Integer	両方
sql_slave_skip_counter	Integer	グローバル
sql_warnings	Boolean	両方
ssl_ca	ファイル名	グローバル
ssl_capath	ディレクトリ名	グローバル
ssl_cert	ファイル名	グローバル
ssl_cipher	文字列	グローバル
ssl_crl	ファイル名	グローバル
ssl_crlpath	ディレクトリ名	グローバル
ssl_fips_mode	列挙	グローバル
ssl_key	ファイル名	グローバル
stored_program_cache	Integer	グローバル
stored_program_definition_cache	Integer	グローバル
super_read_only	Boolean	グローバル
sync_binlog	Integer	グローバル
sync_master_info	Integer	グローバル
sync_relay_log	Integer	グローバル
sync_relay_log_info	Integer	グローバル
syseventlog.facility	文字列	グローバル
syseventlog.include_pid	Boolean	グローバル
syseventlog.tag	文字列	グローバル
table_definition_cache	Integer	グローバル
table_encryption_privilege_check	Boolean	グローバル
table_open_cache	Integer	グローバル
tablespace_definition_cache	Integer	グローバル
temptable_max_mmap	Integer	グローバル
temptable_max_ram	Integer	グローバル
temptable_use_mmap	Boolean	グローバル
thread_cache_size	Integer	グローバル
thread_pool_high_priority_connection	Integer	両方

変数名	変数型	変数スコープ
thread_pool_max_active_query_threads	Integer	グローバル
thread_pool_max_unused_threads	Integer	グローバル
thread_pool_prio_kickup_timer	Integer	両方
thread_pool_stall_limit	Integer	グローバル
time_zone	文字列	両方
timestamp	数値	セッション
tls_ciphersuites	文字列	グローバル
tls_version	文字列	グローバル
tmp_table_size	Integer	両方
transaction_alloc_block_size	Integer	両方
transaction_allow_batching	Boolean	セッション
transaction_isolation	列挙	両方
transaction_prealloc_size	Integer	両方
transaction_read_only	Boolean	両方
transaction_write_set_extraction	列挙	両方
unique_checks	Boolean	両方
updatable_views_with_limit	Boolean	両方
use_secondary_engine	列挙	セッション
validate_password_check_user_name	Boolean	グローバル
validate_password_dictionary_file	ファイル名	グローバル
validate_password_length	Integer	グローバル
validate_password_mixed_case_count	Integer	グローバル
validate_password_number_count	Integer	グローバル
validate_password_policy	列挙	グローバル
validate_password_special_char_count	Integer	グローバル
validate_password.check_user_name	Boolean	グローバル
validate_password.dictionary_file	ファイル名	グローバル
validate_password.length	Integer	グローバル
validate_password.mixed_case_count	Integer	グローバル
validate_password.number_count	Integer	グローバル
validate_password.policy	列挙	グローバル
validate_password.special_char_count	Integer	グローバル
version_tokens_session	文字列	両方
wait_timeout	Integer	両方
windowing_use_high_precision	Boolean	両方

5.1.9.3 永続化されるシステム変数

MySQL サーバーは、その操作を構成するシステム変数を保持します。システム変数には、サーバー操作全体に影響するグローバル値、現在のセッションに影響するセッション値、またはその両方を指定できます。多くのシステム変数は動的であり、`SET` ステートメントを使用して実行時に変更し、現在のサーバーインスタンスの操作に影響を与えることができます。`SET` を使用して、特定のグローバルシステム変数をデータディレクトリ内の `mysqld-auto.cnf` ファイルに永続化し、後続の起動のためのサーバー操作に影響を与えることもできます。`RESET PERSIST` は、永続化された設定を `mysqld-auto.cnf` から削除します。

次の説明では、システム変数の永続化の側面について説明します：

- [永続システム変数の概要](#)
- [システム変数を永続化するための構文](#)
- [永続化されたシステム変数に関する情報の取得](#)
- [mysqld-auto.cnf ファイルのフォーマットおよびサーバー処理](#)

永続システム変数の概要

実行時にグローバルシステム変数を永続化する機能により、サーバーの起動後も保持されるサーバー構成が可能になります。多くのシステム変数は、起動時に `my.cnf` オプションファイルから設定することも、実行時に `SET` ステートメントを使用して設定することもできますが、サーバーを構成するには、サーバーホストへのログインアクセスが必要になるか、実行時またはリモートでサーバーを永続的に構成する機能を提供しません：

- オプションファイルを変更するには、そのファイルに直接アクセスする必要があります。これには、MySQL サーバーホストへのログインアクセスが必要です。これは必ずしも便利ではありません。
- `SET GLOBAL` を使用したシステム変数の変更は、ローカルまたはリモートホストから実行されるクライアントから実行できるランタイム機能ですが、変更は現在実行中のサーバーインスタンスにのみ影響します。設定は永続的ではなく、後続のサーバー起動には引き継がれません。

オプションファイルを編集するか、`SET GLOBAL` を使用して達成可能なものを超えてサーバー構成の管理機能を強化するために、MySQL には、システム変数設定をデータディレクトリ内の `mysqld-auto.cnf` ファイルに永続化する `SET` 構文のバリエーションが用意されています。例：

```
SET PERSIST max_connections = 1000;
SET @@PERSIST.max_connections = 1000;

SET PERSIST_ONLY back_log = 100;
SET @@PERSIST_ONLY.back_log = 100;
```

MySQL には、`mysqld-auto.cnf` から永続システム変数を削除するための `RESET PERSIST` ステートメントも用意されています。

システム変数の永続化によって実行されるサーバー構成には、次の特性があります：

- 永続化された設定は実行時に行われます。
- 永続的な設定は永続的です。これらはサーバーの再起動後に適用されます。
- 永続的な設定は、ローカルクライアントまたはリモートホストから接続するクライアントから行うことができます。これにより、中央クライアントホストから複数の MySQL サーバーをリモートで構成する際に便利です。
- システム変数を永続化するには、MySQL サーバーホストへのログインアクセス権や、オプションファイルへのファイルシステムアクセス権は必要ありません。設定を永続化する機能は、MySQL 権限システムを使用して制御されます。 [セクション 5.1.9.1「システム変数権限」](#) を参照してください。
- 十分な権限を持つ管理者は、システム変数を永続化してサーバーを再構成し、`RESTART` ステートメントを実行して変更された設定をすぐにサーバーで使用できます。
- 永続化された設定では、エラーに関するフィードバックがすぐに提供されます。手動で入力した設定のエラーは、後で検出される可能性があります。構文エラーのある設定は成功せず、サーバー構成を変更しないため、システム変数を永続化する `SET` ステートメントを使用すると、不正な設定の可能性が回避されます。

システム変数を永続化するための構文

システム変数の永続化には、次の `SET` 構文オプションを使用できます：

- グローバルシステム変数をデータディレクトリ内の `mysqld-auto.cnf` オプションファイルに永続化するには、変数名の前に `PERSIST` キーワードまたは `@@PERSIST.` 修飾子を付けます：

```
SET PERSIST max_connections = 1000;
SET @@PERSIST.max_connections = 1000;
```

`SET GLOBAL` と同様に、`SET PERSIST` はグローバル変数のランタイム値を設定しますが、変数設定も `mysqld-auto.cnf` ファイルに書き込みます (既存の変数設定がある場合は置き換えます)。

- グローバル変数のランタイム値を設定せずにグローバルシステム変数を `mysqld-auto.cnf` ファイルに永続化するには、変数名の前に `PERSIST_ONLY` キーワードまたは `@@PERSIST_ONLY` 修飾子を付けます:

```
SET PERSIST_ONLY back_log = 1000;  
SET @@PERSIST_ONLY.back_log = 1000;
```

`PERSIST` と同様に、`PERSIST_ONLY` は変数設定を `mysqld-auto.cnf` に書き込みます。ただし、`PERSIST` とは異なり、`PERSIST_ONLY` はグローバル変数のランタイム値を変更しません。これにより、`PERSIST_ONLY` は、サーバーの起動時にのみ設定できる読み取り専用システム変数の構成に適しています。

`SET` の詳細は、[セクション13.7.6.1「変数代入の SET 構文」](#) を参照してください。

永続化されたシステム変数を削除するには、次の `RESET PERSIST` 構文オプションを使用できます:

- `mysqld-auto.cnf` からすべての永続変数を削除するには、システム変数に名前を付けずに `RESET PERSIST` を使用します:

```
RESET PERSIST;
```

- `mysqld-auto.cnf` から特定の永続変数を削除するには、ステートメントで名前を付けます:

```
RESET PERSIST system_var_name;
```

これには、プラグインが現在インストールされていない場合でも、プラグインシステム変数が含まれます。変数がファイルに存在しない場合は、エラーが発生します。

- `mysqld-auto.cnf` から特定の永続変数を削除し、ファイルに変数が存在しない場合にエラーではなく警告を生成するには、前の構文に `IF EXISTS` 句を追加します:

```
RESET PERSIST IF EXISTS system_var_name;
```

`RESET PERSIST` の詳細は、[セクション13.7.8.7「RESET PERSIST ステートメント」](#) を参照してください。

`SET` を使用してグローバルシステム変数を `DEFAULT` の値またはリテラルのデフォルト値に永続化すると、変数のデフォルト値が割り当てられ、変数の設定が `mysqld-auto.cnf` に追加されます。ファイルから変数を削除するには、`RESET PERSIST` を使用します。

一部のシステム変数は永続化できません。[セクション5.1.9.4「永続的で永続的に制限されないシステム変数」](#) を参照してください。

プラグインによって実装されたシステム変数は、`SET` ステートメントの実行時にプラグインがインストールされた場合に永続化できます。永続化されたプラグイン変数の割り当ては、プラグインがまだインストールされている場合、それ以降のサーバーの再起動で有効になります。プラグインがインストールされなくなった場合、サーバーが `mysqld-auto.cnf` ファイルを読み取るときにプラグイン変数は存在しません。この場合、サーバーはエラーログに警告を書き込み、続行します:

```
currently unknown variable 'var_name'  
was read from the persisted config file
```

永続化されたシステム変数に関する情報の取得

パフォーマンススキーマ `persisted_variables` テーブルは、`mysqld-auto.cnf` ファイルへの SQL インタフェースを提供し、`SELECT` ステートメントを使用して実行時にその内容を検査できるようにします。[セクション27.12.14.1「パフォーマンススキーマ persisted_variables テーブル」](#) を参照してください。

パフォーマンススキーマ `variables_info` テーブルには、各システム変数が最後に設定された時期とユーザーを示す情報が含まれています。[セクション27.12.14.2「パフォーマンススキーマ variables_info テーブル」](#) を参照してください。

テーブルの内容は `mysqld-auto.cnf` ファイルの内容に対応しているため、`RESET PERSIST` は `persisted_variables` テーブルの内容に影響します。一方、`RESET PERSIST` では変数値は変更されないため、サーバーが再起動されるまで `variables_info` テーブルの内容には影響しません。

mysqld-auto.cnf ファイルのフォーマットおよびサーバー処理

`mysqld-auto.cnf` ファイルでは、次のような JSON 形式が使用されます (読みやすくするために少し再フォーマットされています):

```
{
  "Version": 1,
  "mysql_server": {
    "max_connections": {
      "Value": "152",
      "Metadata": {
        "Timestamp": 1519921341372531,
        "User": "root",
        "Host": "localhost"
      }
    },
  },
  "transaction_isolation": {
    "Value": "READ-COMMITTED",
    "Metadata": {
      "Timestamp": 1519921553880520,
      "User": "root",
      "Host": "localhost"
    }
  },
  "mysql_server_static_options": {
    "innodb_api_enable_mdll": {
      "Value": "0",
      "Metadata": {
        "Timestamp": 1519922873467872,
        "User": "root",
        "Host": "localhost"
      }
    },
  },
  "log_slave_updates": {
    "Value": "1",
    "Metadata": {
      "Timestamp": 1519925628441588,
      "User": "root",
      "Host": "localhost"
    }
  },
}
}
```

起動時に、サーバーは他のすべてのオプションファイルの後に `mysqld-auto.cnf` ファイルを処理します (セクション 4.2.2.2 「オプションファイルの使用」 を参照)。サーバーは、ファイルの内容を次のように処理します:

- `persisted_globals_load` システム変数が無効な場合、サーバーは `mysqld-auto.cnf` ファイルを無視します。
- `SET PERSIST_ONLY` を使用して永続化された読取り専用変数のみが `"mysql_server_static_options"` セクションに表示されます。このセクション内にあるすべての変数はコマンド行に追加され、ほかのコマンド行オプションで処理されます。
- 残りのすべての永続変数は、サーバーがクライアント接続のリスニングを開始する直前に、後で `SET GLOBAL` ステートメントと同等のものを実行することによって設定されます。したがって、これらの設定は、起動プロセスが遅くなるまで有効になりません。これは、特定のシステム変数には適していない可能性があります。このような変数は、`mysqld-auto.cnf` ではなく `my.cnf` で設定することをお勧めします。

`mysqld-auto.cnf` ファイルの管理はサーバーに残す必要があります。ファイルの操作は、手動ではなく、`SET` および `RESET PERSIST` ステートメントを使用してのみ実行する必要があります:

- ファイルを削除すると、次のサーバー起動時にすべての永続設定が失われます。(これは、これらの設定を使用せずにサーバーを再構成する場合に許可されます。) ファイル自体を削除せずにファイル内のすべての設定を削除するには、次のステートメントを使用します:

```
RESET PERSIST;
```

- ファイルを手動で変更すると、サーバーの起動時に解析エラーが発生する場合があります。この場合、サーバーはエラーを報告して終了します。この問題が発生した場合は、`persisted_globals_load` システム変数を無効にする

か、`--no-defaults` オプションを指定してサーバーを起動します。または、`mysqld-auto.cnf` ファイルを削除します。ただし、前述のように、このファイルを削除すると、すべての永続設定が失われます。

5.1.9.4 永続的で永続的に制限されないシステム変数

`SET PERSIST` および `SET PERSIST_ONLY` を使用すると、グローバルシステム変数をデータディレクトリ内の `mysqld-auto.cnf` オプションファイルに永続化できます ([セクション13.7.6.1「変数代入の SET 構文」](#) を参照)。ただし、すべてのシステム変数を永続化できるわけではありません。または、特定の制限条件下でのみ永続化できるわけではありません。システム変数が永続的または永続的に制限されない理由を次に示します：

- セッションシステム変数は永続化できません。セッション変数はサーバーの起動時に設定できないため、永続化する理由はありません。
- グローバルシステム変数には、サーバーホストへの直接アクセス権を持つユーザーのみが設定できるような機密データが含まれる場合があります。
- グローバルシステム変数は読み取り専用の場合があります (つまり、サーバーによってのみ設定されます)。この場合、サーバーの起動時でも実行時でも、ユーザーが設定することはできません。
- グローバルシステム変数は、内部使用のみを目的としている場合があります。

永続的でないシステム変数は、どのような状況でも永続化できません。MySQL 8.0.14 の時点では、永続制限付きシステム変数は `SET PERSIST_ONLY` で永続化できますが、次の条件を満たすユーザーのみが永続化できます：

- `persist_only_admin_x509_subject` システム変数は、SSL 証明書の X.509 サブジェクト値に設定されます。
- ユーザーは暗号化された接続を使用してサーバーに接続し、指定されたサブジェクト値で SSL 証明書を提供します。
- ユーザーには、`SET PERSIST_ONLY` を使用するための十分な権限があります ([セクション5.1.9.1「システム変数権限」](#) を参照)。

たとえば、`protocol_version` は読み取り専用であり、サーバーによってのみ設定されるため、どのような状況でも永続化できません。一方、`bind_address` は永続的に制限されるため、前述の条件を満たすユーザーが設定できます。

次のシステム変数は永続性がありません。このリストは、進行中の開発によって変更される可能性があります。

```
audit_log_current_session
audit_log_filter_id
caching_sha2_password_digest_rounds
character_set_system
core_file
have_statement_timeout
have_symlink
hostname
innodb_version
keyring_hashicorp_auth_path
keyring_hashicorp_ca_path
keyring_hashicorp_caching
keyring_hashicorp_commit_auth_path
keyring_hashicorp_commit_ca_path
keyring_hashicorp_commit_caching
keyring_hashicorp_commit_role_id
keyring_hashicorp_commit_server_url
keyring_hashicorp_commit_store_path
keyring_hashicorp_role_id
keyring_hashicorp_secret_id
keyring_hashicorp_server_url
keyring_hashicorp_store_path
large_files_support
large_page_size
license
locked_in_memory
log_bin
log_bin_basename
log_bin_index
lower_case_file_system
ndb_version
```

```
ndb_version_string
persist_only_admin_x509_subject
persisted_globals_load
protocol_version
relay_log_basename
relay_log_index
server_uuid
skip_external_locking
system_time_zone
version_comment
version_compile_machine
version_compile_os
version_compile_zlib
```

永続制限付きシステム変数は、読取り専用であり、コマンドラインまたはオプションファイル ([persist_only_admin_x509_subject](#) および [persisted_globals_load](#) 以外) で設定できます。このリストは、進行中の開発によって変更される可能性があります。

```
audit_log_file
audit_log_format
auto_generate_certs
basedir
bind_address
caching_sha2_password_auto_generate_rsa_keys
caching_sha2_password_private_key_path
caching_sha2_password_public_key_path
character_sets_dir
daemon_memcached_engine_lib_name
daemon_memcached_engine_lib_path
daemon_memcached_option
datadir
default_authentication_plugin
ft_stopword_file
init_file
innodb_buffer_pool_load_at_startup
innodb_data_file_path
innodb_data_home_dir
innodb_dedicated_server
innodb_directories
innodb_force_load_corrupted
innodb_log_group_home_dir
innodb_page_size
innodb_read_only
innodb_temp_data_file_path
innodb_temp_tablespaces_dir
innodb_undo_directory
innodb_undo_tablespaces
keyring_encrypted_file_data
keyring_encrypted_file_password
lc_messages_dir
log_error
mecab_rc_file
named_pipe
pid_file
plugin_dir
port
relay_log
relay_log_info_file
secure_file_priv
sha256_password_auto_generate_rsa_keys
sha256_password_private_key_path
sha256_password_public_key_path
shared_memory
shared_memory_base_name
skip_networking
slave_load_tmpdir
socket
ssl_ca
ssl_capath
ssl_cert
ssl_crl
ssl_crlpath
ssl_key
tmpdir
```

```
version_tokens_session_number
```

永続的に制限されたシステム変数を保持できるようにサーバーを構成するには、次の手順を使用します:

1. MySQL が暗号化された接続をサポートするように構成されていることを確認します。 [セクション6.3.1「暗号化接続を使用するためのMySQLの構成」](#)を参照してください。
2. 永続的に制限されたシステム変数を保持する機能を示す SSL 証明書の X.509 サブジェクト値を指定し、そのサブジェクトを持つ証明書を生成します。 [セクション6.3.3「SSL および RSA 証明書とキーの作成」](#)を参照してください。
3. `persist_only_admin_x509_subject` を指定されたサブジェクト値に設定してサーバーを起動します。たとえば、サーバーの `my.cnf` ファイルに次の行を挿入します:

```
[mysqld]
persist_only_admin_x509_subject="subject-value"
```

Subject 値の形式は、`CREATE USER ... REQUIRE SUBJECT` で使用されるものと同じです。 [セクション13.7.1.3「CREATE USER ステートメント」](#)を参照してください。

`persist_only_admin_x509_subject` 自体は実行時に永続化できないため、このステップは MySQL サーバーホストで直接実行する必要があります。

4. サーバーを再起動します。
5. 指定された Subject 値を持つ SSL 証明書を、永続的に制限されたシステム変数の永続化を許可されるユーザーに配布します。

`myclient-cert.pem` が、永続的に制限されたシステム変数を保持できるクライアントで使用される SSL 証明書であるとして、`openssl` コマンドを使用して、証明書の内容を表示します:

```
shell> openssl x509 -text -in myclient-cert.pem
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 2 (0x2)
    Signature Algorithm: md5WithRSAEncryption
    Issuer: C=US, ST=IL, L=Chicago, O=MyOrg, OU=CA, CN=MyCN
    Validity
      Not Before: Oct 18 17:03:03 2018 GMT
      Not After : Oct 15 17:03:03 2028 GMT
    Subject: C=US, ST=IL, L=Chicago, O=MyOrg, OU=client, CN=MyCN
  ...
```

`openssl` 出力には、証明書サブジェクトの値が次のように表示されます:

```
C=US, ST=IL, L=Chicago, O=MyOrg, OU=client, CN=MyCN
```

MySQL のサブジェクトを指定するには、次の形式を使用します:

```
/C=US/ST=IL/L=Chicago/O=MyOrg/OU=client/CN=MyCN
```

サブジェクト値を使用してサーバー `my.cnf` ファイルを構成します:

```
[mysqld]
persist_only_admin_x509_subject="/C=US/ST=IL/L=Chicago/O=MyOrg/OU=client/CN=MyCN"
```

新しい構成を有効にするには、サーバーを再起動します。

SSL 証明書 (およびその他の関連する SSL ファイル) を適切なユーザーに配布します。このようなユーザーは、暗号化された接続を確立するために必要な証明書およびその他の SSL オプションを使用してサーバーに接続します。

X.509 を使用するには、クライアントは接続する `--ssl-key` および `--ssl-cert` オプションを指定する必要があります。サーバーによって提供される公開証明書を検証できるように、`--ssl-ca` も指定することをお勧めしますが、必須ではありません。例:

```
shell> mysql --ssl-key=myclient-key.pem --ssl-cert=myclient-cert.pem --ssl-ca=mycacert.pem
```


ユーザーに `SET PERSIST_ONLY` を使用するための十分な権限があると仮定すると、永続的に制限されたシステム変数は次のように永続化できます:

```
mysql> SET PERSIST_ONLY socket = '/tmp/mysql.sock';  
Query OK, 0 rows affected (0.00 sec)
```

永続的に制限されたシステム変数を保持できるようにサーバーが構成されていない場合、またはユーザーがその機能に必要な条件を満たしていない場合は、エラーが発生します:

```
mysql> SET PERSIST_ONLY socket = '/tmp/mysql.sock';  
ERROR 1238 (HY000): Variable 'socket' is a non persistent read only variable
```

5.1.9.5 構造化システム変数

構造化システム変数は 2 つの点で通常のシステム変数と異なります。

- この値は、密接に関連すると考えられるサーバーパラメータを指定するコンポーネントを持つ構造です。
- あるタイプの構造化変数に複数のインスタンスがある場合もあります。それぞれが異なる名前を持ち、サーバーによって保持される異なるリソースを参照します。

MySQL では、キーキャッシュの操作を制御するパラメータを指定する構造化変数タイプがサポートされます。キーキャッシュ構造化変数には次のコンポーネントがあります。

- `key_buffer_size`
- `key_cache_block_size`
- `key_cache_division_limit`
- `key_cache_age_threshold`

このセクションでは、構造化変数を参照するための構文について説明します。キーキャッシュ変数は構文の例で使用されますが、キーキャッシュの操作方法についての具体的な詳細は、[セクション 8.10.2「MyISAM キーキャッシュ」](#)に記載されています。

構造化変数インスタンスのコンポーネントを参照するには、`instance_name.component_name` 形式の複合名を使用できます。例:

```
hot_cache.key_buffer_size  
hot_cache.key_cache_block_size  
cold_cache.key_cache_block_size
```

それぞれの構造化システム変数には、`default` という名前のインスタンスが常に事前定義されます。インスタンス名を付けずに構造化変数のコンポーネントを参照した場合、`default` インスタンスが使用されます。つまり、`default.key_buffer_size` および `key_buffer_size` は両方とも同じシステム変数を指します。

構造化変数インスタンスおよびコンポーネントは次の命名ルールに従います。

- あるタイプの構造化変数について、それぞれのインスタンスは、そのタイプの変数の範囲内で一意の名前を持つ必要があります。ただし、インスタンス名は構造化変数タイプをまたいで一意である必要はありません。たとえば、各構造化変数には `default` という名前のインスタンスがあるため、`default` は変数タイプをまたいで一意ではありません。
- 各構造化変数タイプのコンポーネントの名前は、すべてのシステム変数名で一意である必要があります。このようにならない場合 (つまり、2 つの異なる構造化変数のタイプがコンポーネントメンバー名を共有する場合)、インスタンス名によって修飾されないメンバー名への参照に使用するデフォルトの構造化変数が明確でなくなります。
- 構造化変数インスタンス名が引用符で囲まれていない識別子として有効でない場合、逆引用符を使用した、引用符で囲まれた識別子としてこれを指定します。たとえば、`hot-cache` は有効ではありませんが、`'hot-cache'` は有効です。
- `global`、`session`、`local` は有効なインスタンス名ではありません。これにより、構造化されていないシステム変数を参照するための `@@GLOBAL.var_name` などの表記法との競合を回避できます。

現時点では、構造化変数タイプはキーキャッシュのものだけであるため、最初の 2 つのルールが違反される可能性はありません。これらのルールでは、将来、他のタイプの構造化変数が作成される場合に重要性が高くなる可能性があります。

1 つの例外はありますが、単純な変数名を指定できるあらゆるコンテキストで、複合名を使用すると構造化変数コンポーネントを参照できます。たとえば、コマンド行オプションを使用すると、構造化変数に値を割り当てることができます。

```
shell> mysqld --hot_cache.key_buffer_size=64K
```

オプションファイルでは、次の構文を使用します。

```
[mysqld]
hot_cache.key_buffer_size=64K
```

このオプションでサーバーを起動する場合、デフォルトサイズが 8M バイトのデフォルトのキーキャッシュに加えて、サイズが 64K バイトの `hot_cache` という名前のキーキャッシュが作成されます。

次のようにサーバーを開始したとします。

```
shell> mysqld --key_buffer_size=256K \
--extra_cache.key_buffer_size=128K \
--extra_cache.key_cache_block_size=2048
```

この場合、サーバーはデフォルトキーキャッシュのサイズを 256K バイトに設定します。(--`default.key_buffer_size=256K` と記述することもできます)。さらに、このサーバーは、128K バイトのサイズを持つ `extra_cache` という名前の 2 番目のキーキャッシュを作成し、テーブルインデックスブロックのキャッシュ用のブロッックバッファのサイズを 2048 バイトに設定します。

次の例では、サイズの比が 3:1:1 である 3 つの異なるキーキャッシュを指定してサーバーを開始します。

```
shell> mysqld --key_buffer_size=6M \
--hot_cache.key_buffer_size=2M \
--cold_cache.key_buffer_size=2M
```

構造化変数値は実行時にも設定および取得できます。たとえば、`hot_cache` という名前のキーキャッシュを 10M バイトのサイズに設定するには、次のステートメントのどちらかを使用します。

```
mysql> SET GLOBAL hot_cache.key_buffer_size = 10*1024*1024;
mysql> SET @@GLOBAL.hot_cache.key_buffer_size = 10*1024*1024;
```

キャッシュサイズを取得するには、次のようにします。

```
mysql> SELECT @@GLOBAL.hot_cache.key_buffer_size;
```

ただし、次のステートメントは機能しません。この変数は複合名として解釈されず、`LIKE` のパターンマッチング操作の単純文字列として解釈されます。

```
mysql> SHOW GLOBAL VARIABLES LIKE 'hot_cache.key_buffer_size';
```

これは、単純な変数名を指定できるすべての場所で構造化変数を使用できる例外です。

5.1.10 サーバーステータス変数

MySQL サーバーは、その操作に関する情報を提供する多くのステータス変数を保持します。これらの変数およびその値は、`SHOW [GLOBAL | SESSION] STATUS` ステートメントを使用して表示できます ([セクション 13.7.7.37 「SHOW STATUS ステートメント」](#) を参照してください)。オプションの `GLOBAL` キーワードはすべての接続にわたって値を集計し、`SESSION` は現在の接続についての値を表示します。

```
mysql> SHOW GLOBAL STATUS;
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Aborted_clients | 0 |
| Aborted_connects | 0 |
| Bytes_received | 155372598 |
```

```

Bytes_sent      | 1176560426 |
...
Connections     | 30023      |
Created_tmp_disk_tables | 0          |
Created_tmp_files   | 3          |
Created_tmp_tables | 2          |
...
Threads_created  | 217        |
Threads_running   | 88         |
Uptime           | 1389872    |
+-----+-----+

```

多くのステータス変数は、`FLUSH STATUS` ステートメントで 0 にリセットされます。

このセクションでは、各ステータス変数について説明します。ステータス変数サマリーについては、[セクション 5.1.6 「サーバーステータス変数リファレンス」](#) を参照してください。NDB Cluster に固有のステータス変数については、[NDB Cluster ステータス変数](#) を参照してください。

ステータス変数には次の意味があります。

- [Aborted_clients](#)

クライアントが接続を適切に閉じることなく終了したため中止された接続の数。 [セクション B.3.2.9 「通信エラーおよび中止された接続」](#) を参照してください。

- [Aborted_connects](#)

MySQL Server への接続に失敗した試行数。 [セクション B.3.2.9 「通信エラーおよび中止された接続」](#) を参照してください。

接続に関する追加情報については、`Connection_errors_xxx` ステータス変数および `host_cache` テーブルを確認してください。

- [Authentication_ldap_sasl_supported_methods](#)

SASL LDAP 認証を実装する `authentication_ldap_sasl` プラグインは複数の認証方式をサポートしていますが、ホストシステムの構成によっては、すべてが使用できるわけではありません。`Authentication_ldap_sasl_supported_methods` 変数は、サポートされているメソッドの検出機能を提供します。その値は、空白で区切られたサポートされているメソッド名で構成される文字列です。例: "SCRAM-SHA 1 SCRAM-SHA-256 GSSAPI"

この変数は MySQL 8.0.21 で追加されました。

- [Binlog_cache_disk_use](#)

一時バイナリログキャッシュを使用したか、`binlog_cache_size` の値を超えたため、一時ファイルを使用してトランザクションからのステートメントを保管したトランザクション数。

バイナリログトランザクションキャッシュがディスクに書き込まれた非トランザクションステートメントの数は、`Binlog_stmt_cache_disk_use` ステータス変数で別途追跡されます。

- [Acl_cache_items_count](#)

キャッシュされた権限オブジェクトの数。各オブジェクトは、ユーザーとそのアクティブなロールの権限の組合せです。

- [Binlog_cache_use](#)

バイナリログキャッシュを使用したトランザクション数。

- [Binlog_stmt_cache_disk_use](#)

バイナリログステートメントキャッシュを使用したか、`binlog_stmt_cache_size` の値を超えたため、一時ファイルを使用してこれらのステートメントを保管した、非トランザクションステートメントの数。

- [Binlog_stmt_cache_use](#)

バイナリログステートメントキャッシュを使用した非トランザクションステートメントの数。

- `Bytes_received`

すべてのクライアントから受信したバイト数。

- `Bytes_sent`

すべてのクライアントに送信されたバイト数。

- `Caching_sha2_password_rsa_public_key`

RSA キーペアベースのパスワード交換のために `caching_sha2_password` 認証プラグインによって使用される公開キー。この値は、サーバーが `caching_sha2_password_private_key_path` および `caching_sha2_password_public_key_path` システム変数で指定されたファイル内の秘密キーと公開キーを正常に初期化した場合にのみ空になります。`Caching_sha2_password_rsa_public_key` の値は後者のファイルから取得されます。

- `Com_xxx`

`Com_xxx` ステートメントカウンタ変数は、それぞれの `xxx` ステートメントが実行された回数を示します。ステートメントのタイプごとにステータス変数が 1 つあります。たとえば、`Com_delete` および `Com_update` はそれぞれ `DELETE` および `UPDATE` ステートメントをカウントします。`Com_delete_multi` および `Com_update_multi` は類似していますが、複数テーブル構文を使用する `DELETE` および `UPDATE` ステートメントに適用されます。

プリペアドステートメントの引数が不明であったり、実行中にエラーが発生した場合でも、すべての `Com_stmt_xxx` 変数が増加します。つまり、これらの値は発行されたリクエスト数に対応し、正常に完了したリクエスト数に対応しません。たとえば、ステータス変数はサーバーの起動ごとに初期化され、再起動後も保持されないため、`RESTART` および `SHUTDOWN` ステートメントを追跡する `Com_restart` および `Com_shutdown` 変数の値は通常ゼロですが、`RESTART` または `SHUTDOWN` ステートメントが実行されたが失敗した場合はゼロ以外にすることができます。

`Com_stmt_xxx` ステータス変数は次のとおりです。

- `Com_stmt_prepare`

- `Com_stmt_execute`

- `Com_stmt_fetch`

- `Com_stmt_send_long_data`

- `Com_stmt_reset`

- `Com_stmt_close`

これらの変数は、プリペアドステートメントコマンドを表します。これらの名前は、ネットワーク層で使われる `COM_xxx` コマンドセットを示します。つまり、`mysql_stmt_prepare()` や `mysql_stmt_execute()` などのプリペアドステートメントの API コールを実行すると、これらの値は増加します。ただし、`Com_stmt_prepare`、`Com_stmt_execute`、および `Com_stmt_close` も、`PREPARE`、`EXECUTE`、または `DEALLOCATE PREPARE` に対してそれぞれ増加します。さらに、古いステートメントカウンタ変数の値 `Com_prepare_sql`、`Com_execute_sql`、および `Com_dealloc_sql` は、`PREPARE`、`EXECUTE`、および `DEALLOCATE PREPARE` ステートメントに対して増加します。`Com_stmt_fetch` はカーソルからフェッチしたときに発行されるネットワーク往復の合計回数のことです。

`Com_stmt_reprepare` は、サーバーによってステートメントが自動的に再準備された回数を示します。たとえば、ステートメントによって参照されるテーブルまたはビューに対するメタデータ変更の後です。再作成操作は `Com_stmt_reprepare` および `Com_stmt_prepare` を増加させます。

`Com_explain_other` は、実行された `EXPLAIN FOR CONNECTION` ステートメントの数を示します。[セクション 8.8.4 「名前付き接続の実行計画情報の取得」](#) を参照してください。

`Com_change_repl_filter` は、実行された `CHANGE REPLICATION FILTER` ステートメントの数を示します。

- [Compression](#)

クライアント接続で、クライアント/サーバプロトコルの圧縮を使用するかどうか。

MySQL 8.0.18 では、このステータス変数は非推奨です。将来のバージョンの MySQL で削除される予定です。 [レガシー接続圧縮の構成](#) を参照してください。

- [Compression_algorithm](#)

サーバへの現在の接続に使用される圧縮アルゴリズムの名前。値には、[protocol_compression_algorithms](#) システム変数の値で許可されている任意のアルゴリズムを指定できます。たとえば、接続で圧縮が使用されていない場合、値は `uncompressed` になり、接続で `zlib` アルゴリズムが使用されている場合は `zlib` になります。

詳細は、[セクション4.2.8「接続圧縮制御」](#) を参照してください。

この変数は MySQL 8.0.18 で追加されました。

- [Compression_level](#)

サーバへの現在の接続に使用される圧縮レベル。この値は、`zlib` 接続の場合は 6 (デフォルトの `zlib` アルゴリズム圧縮レベル)、`zstd` 接続の場合は 1 から 22、`uncompressed` 接続の場合は 0 です。

詳細は、[セクション4.2.8「接続圧縮制御」](#) を参照してください。

この変数は MySQL 8.0.18 で追加されました。

- [Connection_errors_xxx](#)

これらの変数は、クライアント接続プロセス中に発生したエラーについての情報を提供します。これらはグローバル専用で、すべてのホストからの接続全体で集計したエラー数を表します。これらの変数は、ホストキャッシュによって説明されないエラーを追跡し ([セクション5.1.12.3「DNS ルックアップとホストキャッシュ」](#) を参照してください)、たとえば、TCP 接続に関連付けられないエラーや、接続プロセスのきわめて早期に (IP アドレスが既知となる前も含めて) 発生するエラー、または特定の IP アドレスに固有でない (メモリー不足の状況などの) エラーなどです。

- [Connection_errors_accept](#)

リスニングポートでの `accept()` への呼び出し中に発生したエラーの数。

- [Connection_errors_internal](#)

新しいスレッドの開始のエラーやメモリー不足状況など、サーバの内部エラーが原因で拒否された接続の数。

- [Connection_errors_max_connections](#)

サーバの `max_connections` 制限に到達したため拒否された接続の数。

- [Connection_errors_peer_address](#)

クライアント IP アドレスへの接続の検索中に発生したエラーの数。

- [Connection_errors_select](#)

リスニングポートでの `select()` または `poll()` への呼び出し中に発生したエラーの数。(この操作に失敗したことは、クライアント接続が拒否されたことを必ずしも意味しません。)

- [Connection_errors_tcpwrap](#)

`libwrap` ライブラリによって拒否された接続の数。

- [Connections](#)

MySQL Server への (成功またはそれ以外の) 接続の試行数。

- [Created_tmp_disk_tables](#)

ステートメントの実行中にサーバーによって作成された、ディスク上の内部一時テーブルの数。

作成されたディスク上の内部一時テーブルの数と作成された内部一時テーブルの合計数を比較するには、[Created_tmp_disk_tables](#) と [Created_tmp_tables](#) の値を比較します。

注記

既知の制限のため、[Created_tmp_disk_tables](#) ではメモリーマップファイルに作成されたディスク上の一時テーブルはカウントされません。デフォルトでは、TempTable ストレージエンジンオーバーフローメカニズムは、メモリーマップされたファイルに内部一時テーブルを作成します。この動作は、デフォルトで有効になっている [temptable_use_mmap](#) 変数によって制御されます。

[セクション8.4.4「MySQL での内部一時テーブルの使用」](#) も参照してください。

- [Created_tmp_files](#)

[mysqld](#) が生成した一時ファイルの数。

- [Created_tmp_tables](#)

ステートメントの実行中にサーバーによって作成された、内部一時テーブルの数。

作成されたディスク上の内部一時テーブルの数と作成された内部一時テーブルの合計数を比較するには、[Created_tmp_disk_tables](#) と [Created_tmp_tables](#) の値を比較します。

[セクション8.4.4「MySQL での内部一時テーブルの使用」](#) も参照してください。

[SHOW STATUS](#) ステートメントを呼び出すたびに内部一時テーブルが使用され、グローバルの [Created_tmp_tables](#) 値が増加します。

- [Current_tls_ca](#)

サーバーが新しい接続に使用する SSL コンテキスト内のアクティブな [ssl_ca](#) 値。システム変数が変更されたが、コンテキスト関連のシステム変数値から SSL コンテキストを再構成し、対応するステータス変数を更新するために [ALTER INSTANCE RELOAD TLS](#) が実行されていない場合、このコンテキスト値は現在の [ssl_ca](#) システム変数値と異なる可能性があります。(この潜在的な値の違いは、対応する各コンテキスト関連のシステム変数とステータス変数のペアに適用されます。サーバー側のランタイム構成および暗号化された接続の監視を参照してください。)

この変数は MySQL 8.0.16 で追加されました。

MySQL 8.0.21 の時点では、[Current_tls_xxx](#) ステータス変数の値はパフォーマンススキーマ [tls_channel_status](#) テーブルからも使用できます。[セクション27.12.19.11「tls_channel_status テーブル」](#) を参照してください。

- [Current_tls_capath](#)

サーバーが新しい接続に使用する TSL コンテキスト内のアクティブな [ssl_capath](#) 値。このステータス変数とそれに対応するシステム変数の関係に関するノートは、[Current_tls_ca](#) の説明を参照してください。

この変数は MySQL 8.0.16 で追加されました。

- [Current_tls_cert](#)

サーバーが新しい接続に使用する TSL コンテキスト内のアクティブな [ssl_cert](#) 値。このステータス変数とそれに対応するシステム変数の関係に関するノートは、[Current_tls_ca](#) の説明を参照してください。

この変数は MySQL 8.0.16 で追加されました。

- [Current_tls_cipher](#)

サーバーが新しい接続に使用する TSL コンテキスト内のアクティブな [ssl_cipher](#) 値。このステータス変数とそれに対応するシステム変数の関係に関するノートは、[Current_tls_ca](#) の説明を参照してください。

この変数は MySQL 8.0.16 で追加されました。

- [Current_tls_ciphersuites](#)

サーバーが新しい接続に使用する TSL コンテキスト内のアクティブな [tls_ciphersuites](#) 値。このステータス変数とそれに対応するシステム変数の関係に関するノートは、[Current_tls_ca](#) の説明を参照してください。

この変数は MySQL 8.0.16 で追加されました。

- [Current_tls_crl](#)

サーバーが新しい接続に使用する TSL コンテキスト内のアクティブな [ssl_crl](#) 値。このステータス変数とそれに対応するシステム変数の関係に関するノートは、[Current_tls_ca](#) の説明を参照してください。

この変数は MySQL 8.0.16 で追加されました。

- [Current_tls_crlpath](#)

サーバーが新しい接続に使用する TSL コンテキスト内のアクティブな [ssl_crlpath](#) 値。このステータス変数とそれに対応するシステム変数の関係に関するノートは、[Current_tls_ca](#) の説明を参照してください。

この変数は MySQL 8.0.16 で追加されました。

- [Current_tls_key](#)

サーバーが新しい接続に使用する TSL コンテキスト内のアクティブな [ssl_key](#) 値。このステータス変数とそれに対応するシステム変数の関係に関するノートは、[Current_tls_ca](#) の説明を参照してください。

この変数は MySQL 8.0.16 で追加されました。

- [Current_tls_version](#)

サーバーが新しい接続に使用する TSL コンテキスト内のアクティブな [tls_version](#) 値。このステータス変数とそれに対応するシステム変数の関係に関するノートは、[Current_tls_ca](#) の説明を参照してください。

この変数は MySQL 8.0.16 で追加されました。

- [Delayed_errors](#)

このステータス変数は非推奨です ([DELAYED](#) の挿入はサポートされていないため)。将来のリリースで削除される予定です。

- [Delayed_insert_threads](#)

このステータス変数は非推奨です ([DELAYED](#) の挿入はサポートされていないため)。将来のリリースで削除される予定です。

- [Delayed_writes](#)

このステータス変数は非推奨です ([DELAYED](#) の挿入はサポートされていないため)。将来のリリースで削除される予定です。

- [dragnet.Status](#)

[dragnet.log_error_filter_rules](#) システム変数への最新の代入の結果。このような代入が行われていない場合は空です。

この変数は MySQL 8.0.12 で追加されました。

- [Error_log_buffered_bytes](#)

パフォーマンススキーマ [error_log](#) テーブルで現在使用されているバイト数。たとえば、新しいイベントが古いイベントを破棄するまで収まらないが、新しいイベントが古いイベントよりも小さい場合は、値を減らすことができます。

この変数は MySQL 8.0.22 で追加されました。

- [Error_log_buffered_events](#)

パフォーマンススキーマ `error_log` テーブルに現在存在するイベントの数。 `Error_log_buffered_bytes` と同様に、値を減らすことができます。

この変数は MySQL 8.0.22 で追加されました。

- `Error_log_expired_events`

新しいイベント用の領域を確保するためにパフォーマンススキーマ `error_log` テーブルから破棄されたイベントの数。

この変数は MySQL 8.0.22 で追加されました。

- `Error_log_latest_write`

パフォーマンススキーマ `error_log` テーブルへの最後の書き込みの時間。

この変数は MySQL 8.0.22 で追加されました。

- `Flush_commands`

ユーザーが `FLUSH TABLES` ステートメントを実行したか、内部のサーバー動作が原因で、サーバーがテーブルをフラッシュする回数。これは `COM_REFRESH` パケットの受信によっても増加します。これは `Com_flush` とは対照的で、`FLUSH TABLES`、`FLUSH LOGS` などのいずれかの `FLUSH` ステートメントが実行された回数を示しません。

- `group_replication_primary_member`

グループがシングルプライマリモードで動作している場合、プライマリメンバー UUID を表示します。グループがマルチプライマリモードで動作している場合は、空の文字列が表示されます。

`group_replication_primary_member` ステータス変数は非推奨であり、将来のバージョンで削除される予定です。

- `Handler_commit`

内部 `COMMIT` ステートメントの数。

- `Handler_delete`

テーブルから行が削除された回数。

- `Handler_external_lock`

サーバーは `external_lock()` 関数への呼び出しごとにこの変数を増加し、この呼び出しは通常、テーブルインスタンスへのアクセスの最初と最後に発生します。ストレージエンジンによって相違がある場合があります。この変数は、たとえばパーティション化されたテーブルにアクセスするステートメントについて、ロックが発生する前に削除されたパーティション数を検出するために使用されます。ステートメントについてカウンタがいくら増加したかを確認し、2 を減算し (テーブルそのものに対する 2 件の呼び出し)、2 で除算して、ロックされたパーティション数を取得します。

- `Handler_mrr_init`

サーバーがテーブルへのアクセスでストレージエンジン独自の Multi-Range Read 実装を使用する回数。

- `Handler_prepare`

2 フェーズコミット操作の準備フェーズのカウンタ。

- `Handler_read_first`

インデックスの最初のエントリが読み取られた回数。この値が大きい場合は、サーバーが大量の全インデックススキャンを実行していることを推奨します (`col1` がインデックス付けされていると仮定した `SELECT col1 FROM foo` など)。

- `Handler_read_key`

キーに基づいて行を読み取るリクエスト数。この値が高いことは、クエリーに対してテーブルが適切にインデックス付けされていることのよい目安になります。

- [Handler_read_last](#)

インデックスの最後のキーを読み取るリクエスト数。ORDER BY では、サーバーは最初のキーリクエストの後にいくつかの次のキーリクエストを発行しますが、ORDER BY DESC では、最後のキーリクエストの後にいくつかの前のキーリクエストを発行します。

- [Handler_read_next](#)

キー順で次の行の読み取りリクエスト数。この値は、範囲制約を持つインデックスカラムにクエリーを実行するか、インデックススキャンを実行する場合に増加します。

- [Handler_read_prev](#)

キー順で前の行の読み取りリクエスト数。この読み取り方法は、ORDER BY ... DESC を最適化するために主に使用されます。

- [Handler_read_rnd](#)

固定された位置に基づいた行読み取りリクエスト数。この値は、結果のソートが必要となる多くのクエリーを実行する場合に高くなります。MySQL がテーブル全体をスキャンする必要がある多くのクエリーが存在する可能性があるか、キーが適切に使用されない結合があります。

- [Handler_read_rnd_next](#)

データファイル内で次の行の読み取りリクエスト数。多くのテーブルスキャンを実行すると、この値は高くなります。一般的に、これはテーブルが正しくインデックス付けされていないか、作成したインデックスを利用するようにクエリーが記述されていないことを示します。

- [Handler_rollback](#)

ロールバック操作を実行するためのストレージエンジンのリクエスト数。

- [Handler_savepoint](#)

セーブポイントを配置するためのストレージエンジンへのリクエスト数。

- [Handler_savepoint_rollback](#)

セーブポイントをロールバックするためのストレージエンジンへのリクエスト数。

- [Handler_update](#)

テーブルの行更新リクエスト数。

- [Handler_write](#)

テーブルへの行挿入リクエスト数。

- [InnoDB_buffer_pool_dump_status](#)

InnoDB バッファプールに保持されるページを記録するための操作の進捗状況で、innodb_buffer_pool_dump_at_shutdown または innodb_buffer_pool_dump_now の設定によってトリガーされます。

関連する情報と例については、[セクション15.8.3.6「バッファプールの状態の保存と復元」](#)を参照してください。

- [InnoDB_buffer_pool_load_status](#)

以前の時点のものに対応するページのセットを読み取ることによって、InnoDB バッファプールをウォームアップする操作の進捗状況で、innodb_buffer_pool_load_at_startup または innodb_buffer_pool_load_now の設定によってトリガーされます。操作によってもたらされるオーバーヘッドが多すぎる場合、innodb_buffer_pool_load_abort を設定すると取り消しできます。

関連する情報と例については、[セクション15.8.3.6「バッファープールの状態の保存と復元」](#)を参照してください。

- `InnoDB_buffer_pool_bytes_data`

データを含む InnoDB バッファープール内のバイトの総数。ダーティーページとクリーンページの両方が含まれます。圧縮テーブルによってバッファープールが異なるサイズのページを保持する場合には、`InnoDB_buffer_pool_pages_data` を使用するよりも正確なメモリ使用量を計算するために使用します。

- `InnoDB_buffer_pool_pages_data`

データを含む InnoDB バッファープール内のページ数。ダーティーページとクリーンページの両方が含まれます。`compressed tables` を使用している場合、レポートされる `InnoDB_buffer_pool_pages_data` 値が `InnoDB_buffer_pool_pages_total` より大きい可能性があります (Bug #59550)。

- `InnoDB_buffer_pool_bytes_dirty`

InnoDB バッファープール内のダーティーページに保持されている現在の合計バイト数。圧縮テーブルによってバッファープールが異なるサイズのページを保持する場合には、`InnoDB_buffer_pool_pages_dirty` を使用するよりも正確なメモリ使用量を計算するために使用します。

- `InnoDB_buffer_pool_pages_dirty`

InnoDB バッファープール内のダーティーページの現在の数。

- `InnoDB_buffer_pool_pages_flushed`

InnoDB バッファープールからページをフラッシュするためのリクエスト数。

- `InnoDB_buffer_pool_pages_free`

InnoDB バッファープール内の空きページの数。

- `InnoDB_buffer_pool_pages_latched`

InnoDB バッファープール内のラッチされたページの数。これらは現在読み取りまたは書き込み中であるか、ほかの何らかの理由でフラッシュまたは削除できないページです。この変数の計算にはコストがかかるため、UNIV_DEBUG システムがサーバー構築時に定義される場合のみ利用できます。

- `InnoDB_buffer_pool_pages_misc`

行ロックやアダプティブハッシュインデックスなど、管理オーバーヘッドに割り当てられているためビジー状態になっている、InnoDB バッファープール内のページ数。この値は `InnoDB_buffer_pool_pages_total` - `InnoDB_buffer_pool_pages_free` - `InnoDB_buffer_pool_pages_data` として計算することもできます。`compressed tables` を使用している場合、`InnoDB_buffer_pool_pages_misc` で範囲外の値が報告されることがあります (Bug #59550)。

- `InnoDB_buffer_pool_pages_total`

InnoDB バッファープールの合計サイズ (ページ単位)。`compressed tables` を使用している場合、レポートされる `InnoDB_buffer_pool_pages_data` 値が `InnoDB_buffer_pool_pages_total` より大きい可能性があります (Bug #59550)

- `InnoDB_buffer_pool_read_ahead`

先読みバックグラウンドスレッドによって InnoDB バッファープールに読み取られたページ数。

- `InnoDB_buffer_pool_read_ahead_evicted`

クエリーによってアクセスされずにあとで消去された先読みバックグラウンドスレッドによって InnoDB バッファープールに読み取られたページ数。

- `InnoDB_buffer_pool_read_ahead_rnd`

InnoDB によって開始された「random」先読みの数。これは、クエリーがテーブルの大部分をランダムな順序でスキャンする場合に発生します。

- `InnoDB_buffer_pool_read_requests`

論理読み取りリクエスト数。

- `InnoDB_buffer_pool_reads`

InnoDB がバッファプールから満たすことができず、ディスクから直接読み取る必要があった論理読み取りの数。

- `InnoDB_buffer_pool_resize_status`

InnoDB `buffer pool` を動的にサイズ変更する操作のステータスで、`innodb_buffer_pool_size` パラメータを動的に設定することによってトリガーされます。`innodb_buffer_pool_size` パラメータは動的で、サーバーを再起動せずにバッファプールのサイズを変更できます。関連情報については、[オンラインでの InnoDB バッファプールサイズの構成](#)を参照してください。

- `InnoDB_buffer_pool_wait_free`

通常は、InnoDB バッファプールへの書き込みは、バックグラウンドで行われます。InnoDB がページを読み取るか作成する必要があって、クリーンページが利用できない場合、InnoDB は一部の**ダーティーページ**を最初にフラッシュし、その操作の完了まで待機します。このカウンタはこれらの待機のインスタンスをカウントします。`innodb_buffer_pool_size` が適切に設定されていれば、この値は小さくなります。

- `InnoDB_buffer_pool_write_requests`

InnoDB バッファプールに対して実行される書き込みの数。

- `InnoDB_data_fsyncs`

これまでの `fsync()` 操作数。`fsync()` 呼び出しの頻度は `innodb_flush_method` 構成オプションの設定に影響されません。

- `InnoDB_data_pending_fsyncs`

現在保留中の `fsync()` 操作の数。`fsync()` 呼び出しの頻度は `innodb_flush_method` 構成オプションの設定に影響されます。

- `InnoDB_data_pending_reads`

現在保留中の読み取りの数。

- `InnoDB_data_pending_writes`

現在保留中の書き込み数。

- `InnoDB_data_read`

サーバーの起動後に読み取られたデータ量 (バイト単位)。

- `InnoDB_data_reads`

データ読取り (OS ファイル読取り) の合計数。

- `InnoDB_data_writes`

データ書き込みの合計数。

- `InnoDB_data_written`

これまでに書き込まれたデータ量 (バイト単位)。

- `InnoDB_dblwr_pages_written`

ダブル書き込みバッファに書き込まれたページ数。 [セクション15.11.1「InnoDB ディスク I/O」](#)を参照してください。

- `InnoDB_dblwr_writes`

実行されたダブル書き込み操作の数。 [セクション15.11.1「InnoDB ディスク I/O」](#) を参照してください。

- [InnoDB_have_atomic_builtins](#)

サーバーが [アトミック命令](#) で構築されたかどうかを示します。

- [InnoDB_log_waits](#)

The number of times that the [ログバッファ](#) が小さすぎるため、続行する前に [フラッシュ](#) するために [待機](#) が必要だった回数。

- [InnoDB_log_write_requests](#)

InnoDB [Redo ログ](#) の書き込みリクエストの数。

- [InnoDB_log_writes](#)

InnoDB [Redo ログ](#) ファイルへの物理書き込みの数。

- [InnoDB_num_open_files](#)

InnoDB で現在開いたままになっているファイルの数。

- [InnoDB_os_log_fsyncs](#)

InnoDB [Redo ログ](#) ファイルに対して実行される [fsync\(\)](#) 書き込みの数。

- [InnoDB_os_log_pending_fsyncs](#)

InnoDB [Redo ログ](#) ファイルに対する保留中の [fsync\(\)](#) 操作の数。

- [InnoDB_os_log_pending_writes](#)

InnoDB [Redo ログ](#) ファイルに対する保留中の書き込み数。

- [InnoDB_os_log_written](#)

InnoDB [Redo ログ](#) ファイルに書き込まれたバイト数。

- [InnoDB_page_size](#)

InnoDB のページサイズ (デフォルトは 16K バイト)。ページには多くの値がカウントされ、ページサイズは簡単にバイトに換算できます。

- [InnoDB_pages_created](#)

InnoDB テーブルの操作によって作成されるページ数。

- [InnoDB_pages_read](#)

InnoDB テーブルに対する操作によって [InnoDB バッファプール](#) から読み取られたページ数。

- [InnoDB_pages_written](#)

InnoDB テーブルの操作によって書き込まれるページ数。

- [InnoDB_redo_log_enabled](#)

redo ログが有効か無効か。MySQL 8.0.21 で導入されました。

[redo ログの無効化](#) を参照してください。

- [InnoDB_row_lock_current_waits](#)

InnoDB テーブルの操作によって現在待機中の [行ロック](#) の数。

- `InnoDB_row_lock_time`
InnoDB テーブルの行ロックの取得に要した合計時間 (ミリ秒単位)。
- `InnoDB_row_lock_time_avg`
InnoDB テーブルの行ロックの取得に要した平均時間 (ミリ秒)。
- `InnoDB_row_lock_time_max`
InnoDB テーブルの行ロックの取得に要した最大時間 (ミリ秒)。
- `InnoDB_row_lock_waits`
InnoDB テーブル上の操作が行ロックを待機した回数。
- `InnoDB_rows_deleted`
InnoDB テーブルから削除された行数。
- `InnoDB_rows_inserted`
InnoDB テーブルに挿入された行数。
- `InnoDB_rows_read`
InnoDB テーブルから読み取られた行数。
- `InnoDB_rows_updated`
InnoDB テーブル内で更新された行数。
- `InnoDB_system_rows_deleted`
システム作成スキーマに属する InnoDB テーブルから削除された行数。
- `InnoDB_system_rows_inserted`
システム作成スキーマに属する InnoDB テーブルに挿入された行数。
- `InnoDB_system_rows_read`
システム作成スキーマに属する InnoDB テーブルから読み取られた行数。
- `InnoDB_truncated_status_writes`
SHOW ENGINE INNODB STATUS ステートメントからの出力が切り捨てられた回数。
- `InnoDB_undo_tablespaces_active`
アクティブな undo テーブルスペースの数。暗黙的 undo テーブルスペース (InnoDB 作成) と明示的 undo テーブルスペース (ユーザー作成) の両方が含まれます。undo テーブルスペースの詳細は、[セクション15.6.3.4「undo テーブルスペース」](#) を参照してください。
- `InnoDB_undo_tablespaces_explicit`
ユーザー作成 undo テーブルスペースの数。undo テーブルスペースの詳細は、[セクション15.6.3.4「undo テーブルスペース」](#) を参照してください。
- `InnoDB_undo_tablespaces_implicit`
InnoDB によって作成された undo テーブルスペースの数。MySQL インスタンスの初期化時に、InnoDB によって 2 つのデフォルト undo テーブルスペースが作成されます。undo テーブルスペースの詳細は、[セクション15.6.3.4「undo テーブルスペース」](#) を参照してください。
- `InnoDB_undo_tablespaces_total`

undo テーブルスペースの合計数。アクティブおよび非アクティブな暗黙的な undo テーブルスペース (InnoDB 作成) と明示的な undo テーブルスペース (ユーザー作成) の両方が含まれます。undo テーブルスペースの詳細は、[セクション15.6.3.4「undo テーブルスペース」](#) を参照してください。

- [Key_blocks_not_flushed](#)

変更されたがまだディスクにフラッシュされていない MyISAM キーキャッシュ内のキーブロック数。

- [Key_blocks_unused](#)

MyISAM キーキャッシュ内の未使用ブロック数。この値を使用して、使用中のキーキャッシュの量を判別できます。[セクション5.1.8「サーバースystem変数」](#) の [key_buffer_size](#) に関する説明を参照してください。

- [Key_blocks_used](#)

MyISAM キーキャッシュ内の使用済みブロック数。この値は、一度に使用された今までの最大ブロック数を示す高位境界値です。

- [Key_read_requests](#)

MyISAM キーキャッシュからキーブロックを読み取るリクエスト数。

- [Key_reads](#)

ディスクから MyISAM キーキャッシュへのキーブロックの物理的な読み取りの数。Key_reads が大きい場合、key_buffer_size の値が小さすぎる可能性があります。キャッシュミス率は $\text{Key_reads} / \text{Key_read_requests}$ と計算できます。

- [Key_write_requests](#)

MyISAM キーキャッシュにキーブロックを書き込むリクエスト数。

- [Key_writes](#)

MyISAM キーキャッシュからディスクへのキーブロックの物理的な書き込みの数。

- [Last_query_cost](#)

クエリーオプティマイザによって計算された、最後にコンパイルされたクエリーの合計コスト。これは同じクエリーに対して異なるクエリー計画のコストを比較するために役立ちます。デフォルト値の 0 は、クエリーがまだコンパイルされていないことを意味します。デフォルト値は 0 です。Last_query_cost はセッションスコープを持ちます。

MySQL 8.0.16 以降では、この変数は、複数のクエリーブロックを持つクエリーのコストを表示し、各クエリーブロックのコスト見積りを合計して、キャッシュ不可能なサブクエリーが実行される回数を見積もり、それらのクエリーブロックのコストにサブクエリー実行の数を乗算します。(Bug #92766、Bug #28786951) MySQL 8.0.16 より前は、Last_query_cost は単純な「フラット」クエリーに対してのみ正確に計算されましたが、サブクエリーや UNION を含むクエリーなどの複雑なクエリーに対しては計算されませんでした。(後者の場合、値は 0 に設定されています。)

- [Last_query_partial_plans](#)

クエリーオプティマイザが前のクエリーの実行計画の構築で実行した反復数。Last_query_cost はセッションスコープを持ちます。

- [Locked_connects](#)

ロックされたユーザーアカウントへの接続試行回数。アカウントのロックおよびロック解除の詳細は、[セクション6.2.19「アカウントロック」](#) を参照してください。

- [Max_execution_time_exceeded](#)

実行タイムアウトを超えた SELECT ステートメントの数。

- [Max_execution_time_set](#)

ゼロ以外の実行タイムアウトが設定された [SELECT](#) ステートメントの数。これには、ゼロ以外の [MAX_EXECUTION_TIME](#) オプティマイザヒントを含むステートメント、およびそのようなヒントを含まないが [max_execution_time](#) システム変数で示されるタイムアウトがゼロ以外の場合に実行されるステートメントが含まれます。
- [Max_execution_time_set_failed](#)

実行タイムアウトを設定しようとして失敗した [SELECT](#) ステートメントの数。
- [Max_used_connections](#)

サーバーが開始されてから同時に使用された接続の最大数。
- [Max_used_connections_time](#)

[Max_used_connections](#) が現在の値に達した時刻。
- [Not_flushed_delayed_rows](#)

このステータス変数は非推奨です ([DELAYED](#) の挿入はサポートされていないため)。将来のリリースで削除される予定です。
- [mecab_charset](#)

MeCab 全文パーサープラグインで現在使用されている文字セット。関連情報については、[セクション 12.10.9 「MeCab フルテキストパーサープラグイン」](#) を参照してください。
- [Ongoing_anonymous_transaction_count](#)

匿名としてマークされた進行中のトランザクションの数を表示します。これを使用すると、これ以上のトランザクションが処理を待機していないことを確認できます。
- [Ongoing_anonymous_gtid_violating_transaction_count](#)

このステータス変数は、デバッグビルドでのみ使用できます。 [gtid_next=ANONYMOUS](#) を使用し、GTID 整合性に違反する進行中のトランザクションの数を表示します。
- [Ongoing_automatic_gtid_violating_transaction_count](#)

このステータス変数は、デバッグビルドでのみ使用できます。 [gtid_next=AUTOMATIC](#) を使用し、GTID 整合性に違反する進行中のトランザクションの数を表示します。
- [Open_files](#)

開いているファイルの数。このカウントにはサーバーによって開いた通常のファイルが含まれます。ソケットやパイプなどのほかのタイプのファイルは含まれません。またこのカウントには、サーバーレベルに実行を依頼するのではなく、ストレージエンジンがそれら独自の内部関数を使用して開いたファイルは含まれません。
- [Open_streams](#)

開いているストリーム数 (主にロギングに使用)。
- [Open_table_definitions](#)

キャッシュされたテーブル定義の数。
- [Open_tables](#)

開いているテーブルの数。
- [Opened_files](#)

[my_open\(\)](#) ([mysys](#) ライブラリ関数) によって開いたファイルの数。この関数を使用せずにファイルを開くサーバーの一部は、カウントを増加させません。

- [Opened_table_definitions](#)

キャッシュされたテーブル定義の数。

- [Opened_tables](#)

開いているテーブル数。 [Opened_tables](#) の値が大きい場合、 [table_open_cache](#) の値が小さいすぎる可能性があります。

- [Performance_schema_xxx](#)

パフォーマンススキーマのステータス変数は、 [セクション27.16「パフォーマンススキーマステータス変数」](#) にリストされています。これらの変数は、メモリー制約のためロードまたは作成できないにインストールメンテーションについての情報を提供します。

- [Prepared_stmt_count](#)

現在のプリアドステートメントの数。(ステートメントの最大数は、 [max_prepared_stmt_count](#) システム変数によって指定されます。)

- [Queries](#)

サーバーによって実行されたステートメントの数。この変数は [Questions](#) 変数と異なり、ストアードプロシージャー内で実行されるステートメントを含みます。 [COM_PING](#) または [COM_STATISTICS](#) コマンドをカウントしません。

このセクションの最初の説明では、このステートメントカウントステータス変数をこのようなほかの変数に関連付ける方法を示します。

- [質問](#)

サーバーによって実行されたステートメントの数。これは [Queries](#) 変数とは異なり、クライアントによってサーバーに送信されたステートメントのみを含み、ストアードプロシージャー内で実行されたステートメントは含みません。この変数は、 [COM_PING](#)、 [COM_STATISTICS](#)、 [COM_STMT_PREPARE](#)、 [COM_STMT_CLOSE](#)、または [COM_STMT_RESET](#) コマンドをカウントしません。

このセクションの最初の説明では、このステートメントカウントステータス変数をこのようなほかの変数に関連付ける方法を示します。

- [Rpl_semi_sync_master_clients](#)

準同期レプリカの数。

この変数は、ソース側の準同期レプリケーションプラグインがインストールされている場合にのみ使用できます。

- [Rpl_semi_sync_master_net_avg_wait_time](#)

ソースがレプリカ応答を待機した平均時間(マイクロ秒)。この変数は常に 0 であり、非推奨です。将来のバージョンで削除される予定です。

この変数は、ソース側の準同期レプリケーションプラグインがインストールされている場合にのみ使用できます。

- [Rpl_semi_sync_master_net_wait_time](#)

ソースがレプリカ応答を待機した合計時間(マイクロ秒)。この変数は常に 0 であり、非推奨です。将来のバージョンで削除される予定です。

この変数は、ソース側の準同期レプリケーションプラグインがインストールされている場合にのみ使用できます。

- [Rpl_semi_sync_master_net_waits](#)

ソースがレプリカ応答を待機した合計回数。

この変数は、ソース側の準同期レプリケーションプラグインがインストールされている場合にのみ使用できます。

- [Rpl_semi_sync_master_no_times](#)

ソースが準同期レプリケーションをオフにした回数。

この変数は、ソース側の準同期レプリケーションプラグインがインストールされている場合にのみ使用できます。

- [Rpl_semi_sync_master_no_tx](#)

レプリカによって正常に確認されなかったコミットの数。

この変数は、ソース側の準同期レプリケーションプラグインがインストールされている場合にのみ使用できます。

- [Rpl_semi_sync_master_status](#)

準同期レプリケーションが現在ソースで動作しているかどうか。プラグインが有効で、コミット認証が発生した場合、この値は **ON** です。これは、プラグインが有効になっていないか、コミット確認タイムアウトのためにソースが非同期レプリケーションにフォールバックした場合の **OFF** です。

この変数は、ソース側の準同期レプリケーションプラグインがインストールされている場合にのみ使用できます。

- [Rpl_semi_sync_master_timefunc_failures](#)

[gettimeofday\(\)](#) などの時間関数のコール時にソースが失敗した回数。

この変数は、ソース側の準同期レプリケーションプラグインがインストールされている場合にのみ使用できます。

- [Rpl_semi_sync_master_tx_avg_wait_time](#)

ソースが各トランザクションを待機した平均時間 (マイクロ秒)。

この変数は、ソース側の準同期レプリケーションプラグインがインストールされている場合にのみ使用できます。

- [Rpl_semi_sync_master_tx_wait_time](#)

ソースがトランザクションを待機した合計時間 (マイクロ秒)。

この変数は、ソース側の準同期レプリケーションプラグインがインストールされている場合にのみ使用できます。

- [Rpl_semi_sync_master_tx_waits](#)

ソースがトランザクションを待機した合計回数。

この変数は、ソース側の準同期レプリケーションプラグインがインストールされている場合にのみ使用できます。

- [Rpl_semi_sync_master_wait_pos_backtraverse](#)

バイナリ座標が以前に待機したイベントより小さいイベントをソースが待機した合計回数。これは、トランザクションが応答の待機を開始した順序が、バイナリロギイベントが書き込まれた順序と異なる場合に発生することがあります。

この変数は、ソース側の準同期レプリケーションプラグインがインストールされている場合にのみ使用できます。

- [Rpl_semi_sync_master_wait_sessions](#)

レプリカ応答を現在待機しているセッションの数。

この変数は、ソース側の準同期レプリケーションプラグインがインストールされている場合にのみ使用できます。

- [Rpl_semi_sync_master_yes_tx](#)

レプリカによって正常に確認されたコミットの数。

この変数は、ソース側の準同期レプリケーションプラグインがインストールされている場合にのみ使用できます。

- [Rpl_semi_sync_slave_status](#)

準同期レプリケーションが現在レプリカで動作しているかどうか。これは、プラグインが有効化され、レプリケーション I/O スレッドが実行されている場合は **ON**、それ以外の場合は **OFF** です。

この変数は、レプリカ側の準同期レプリケーションプラグインがインストールされている場合にのみ使用できません。

- [Rsa_public_key](#)

この変数の値は、RSA キーペアベースのパスワード交換用の `sha256_password` 認証プラグインで使用される公開キーです。この値は、`sha256_password_private_key_path` および `sha256_password_public_key_path` システム変数によって名前が指定されるファイル内の秘密鍵および公開鍵をサーバーが正常に初期化した場合のみ空ではありません。`Rsa_public_key` の値は後者のファイルから得られます。

`sha256_password` については、[セクション6.4.1.3「SHA-256 プラガブル認証」](#)を参照してください。

- [Secondary_engine_execution_count](#)

セカンダリエンジンにオフロードされたクエリーの数。この変数は MySQL 8.0.13 で追加されました。

HeatWave で使用します。[MySQL HeatWave User Guide](#) を参照してください。

- [Select_full_join](#)

インデックスを使用しないためテーブルスキャンを実行する結合の数。この値が 0 でない場合、テーブルのインデックスを慎重に検査してください。

- [Select_full_range_join](#)

参照テーブル上で範囲検索を使用した結合の数。

- [Select_range](#)

最初のテーブルの範囲が使用された結合の数。値がきわめて大きい場合でも、これは通常重大な問題ではありません。

- [Select_range_check](#)

各行のあとにキーの使用法がチェックされるキーなしの結合数。これが 0 でない場合、テーブルのインデックスを慎重に検査してください。

- [Select_scan](#)

最初のテーブルのフルスキャンが実行された結合の数。

- [Slave_open_temp_tables](#)

レプリケーション SQL スレッドが現在オープンしている一時テーブルの数。値がゼロより大きい場合は、レプリカを安全に停止できません。[セクション17.5.1.31「レプリケーションと一時テーブル」](#)を参照してください。この変数は、all レプリケーションチャンネルのオープン一時テーブルの合計数をレポートします。

- [Slave_rows_last_search_algorithm_used](#)

このレプリカが行ベースレプリケーション用の行を検索するために最後に使用した検索アルゴリズム。結果には、チャンネルで最後に実行されたトランザクションの検索アルゴリズムとして、レプリカがインデックス、テーブルスキャンまたはハッシュを使用したかが表示されます。

使用される方法は、`slave_rows_search_algorithms` システム変数の設定、および関連するテーブルで使用可能なキーによって異なります。

この変数は、MySQL のデバッグビルドでのみ使用できます。

- [Slow_launch_threads](#)

作成に要した時間が `slow_launch_time` 秒を超えたスレッドの数。

- [Slow_queries](#)

[long_query_time](#) 秒よりも時間を要したクエリーの数。このカウンタは、スロークエリーログが有効かどうかに関係なく増加します。このログについては、[セクション5.4.5「スロークエリーログ」](#)を参照してください。

- [Sort_merge_passes](#)

ソートアルゴリズムが実行する必要があったマージパスの数。この値が大きい場合、[sort_buffer_size](#) システム変数を増やすことを検討してください。

- [Sort_range](#)

範囲を使用して実行されたソートの数。

- [Sort_rows](#)

ソートされた行の数。

- [Sort_scan](#)

テーブルをスキャンすることで実行されたソートの数。

- [Ssl_accept_renegotiates](#)

接続を確立するために必要なネゴシエーションの数。

- [Ssl_accepts](#)

受け入れられた SSL 接続の数。

- [Ssl_callback_cache_hits](#)

コールバックキャッシュのヒット数。

- [Ssl_cipher](#)

現在の暗号化暗号 (暗号化されていない接続の場合は空)。

- [Ssl_cipher_list](#)

利用可能な SSL 暗号のリスト (非 SSL 接続の場合は空)。MySQL が TLSv1.3 をサポートしている場合、この値には使用可能な TLSv1.3 暗号スイートが含まれます。[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#)を参照してください。

- [Ssl_client_connects](#)

SSL 対応アプリケーションソースサーバーへの SSL 接続試行の数。

- [Ssl_connect_renegotiates](#)

SSL 対応アプリケーションソースサーバーへの接続を確立するために必要なネゴシエーションの数。

- [Ssl_ctx_verify_depth](#)

SSL コンテキスト検証の深さ (テストされるチェーン内の証明書数)。

- [Ssl_ctx_verify_mode](#)

SSL コンテキスト検証モード。

- [Ssl_default_timeout](#)

デフォルトの SSL タイムアウト。

- [Ssl_finished_accepts](#)

サーバーへの正常な SSL 接続数。

- [Ssl_finished_connects](#)

SSL 対応レプリケーションソースサーバーへの正常なレプリカ接続の数。

- [Ssl_server_not_after](#)

SSL 証明書が有効な最終日。SSL 証明書の有効期限情報を確認するには、次のステートメントを使用します:

```
mysql> SHOW STATUS LIKE 'Ssl_server_not%';
+-----+-----+
| Variable_name | Value                               |
+-----+-----+
| Ssl_server_not_after | Apr 28 14:16:39 2025 GMT |
| Ssl_server_not_before | May 1 14:16:39 2015 GMT |
+-----+-----+
```

- [Ssl_server_not_before](#)

SSL 証明書が有効な最初の日。

- [Ssl_session_cache_hits](#)

SSL セッションキャッシュのヒット数。

- [Ssl_session_cache_misses](#)

SSL セッションキャッシュのミス数。

- [Ssl_session_cache_mode](#)

SSL セッションキャッシュモード。

- [Ssl_session_cache_overflows](#)

SSL セッションキャッシュのオーバーフロー数。

- [Ssl_session_cache_size](#)

SSL セッションキャッシュサイズ。

- [Ssl_session_cache_timeouts](#)

SSL セッションキャッシュのタイムアウト数。

- [Ssl_sessions_reused](#)

キャッシュから再使用された SSL 接続の数。

- [Ssl_used_session_cache_entries](#)

使用された SSL セッションキャッシュエントリの数。

- [Ssl_verify_depth](#)

レプリケーション SSL 接続の検証の深さ。

- [Ssl_verify_mode](#)

SSL を使用する接続のためにサーバーで使用される検証モード。値はビットマスクです。ビットは [openssl/ssl.h](#) ヘッダーファイルで定義されます:

```
# define SSL_VERIFY_NONE          0x00
# define SSL_VERIFY_PEER         0x01
# define SSL_VERIFY_FAIL_IF_NO_PEER_CERT 0x02
# define SSL_VERIFY_CLIENT_ONCE  0x04
```

`SSL_VERIFY_PEER` は、サーバーがクライアント証明書を要求することを示します。クライアントが提供する場合、サーバーは検証を実行し、検証が成功した場合にのみ続行します。`SSL_VERIFY_CLIENT_ONCE` は、クライアント証明書のリクエストが最初のハンドシェイクでのみ実行されることを示します。

- `Ssl_version`

接続の SSL プロトコルバージョン (TLSv1 など)。接続が暗号化されていない場合、値は空です。

- `Table_locks_immediate`

テーブルロックのリクエストが即座に付与された回数。

- `Table_locks_waited`

テーブルロックのリクエストが即座に付与されず、待機が必要だった回数。これが高く、パフォーマンスに問題がある場合、最初にクエリーを最適化し、次に 1 つ以上のテーブルを分割するか、レプリケーションを使用してください。

- `Table_open_cache_hits`

開いたテーブルのキャッシュルックアップのヒット数。

- `Table_open_cache_misses`

開いたテーブルのキャッシュルックアップのミス数。

- `Table_open_cache_overflows`

開いたテーブルのキャッシュのオーバーフロー数。これはテーブルが開くか閉じたあとにキャッシュインスタンスが未使用のエントリを持ち、インスタンスのサイズが `table_open_cache / table_open_cache_instances` より大きい場合の回数です。

- `Tc_log_max_pages_used`

この変数は、`mysqld` が内部の XA トランザクションのリカバリのためのトランザクションコーディネータとしての役割を果たすとき、`mysqld` によって使用されるログのメモリーマップ実装に対して、サーバーが起動してからログに使用された最大のページ数を示します。`Tc_log_max_pages_used` と `Tc_log_page_size` の積が常にログサイズよりも極端に小さい場合、そのサイズが必要以上に大きいため削減できます。(このサイズは `--log-tc-size` オプションで指定できます。この変数は未使用です: バイナリログベースのリカバリでは不要であり、2 フェーズのコミットが可能で XA トランザクションをサポートするストレージエンジンの数が複数の場合を除き、メモリーマップされたリカバリログ方式は使用されません。(InnoDB のみが該当するエンジンです。))

- `Tc_log_page_size`

XA リカバリログのメモリーマップ実装に使用されるページサイズ。デフォルト値は `getpagesize()` を使用して決定されます。この変数は、`Tc_log_max_pages_used` で説明されているのと同じ理由で未使用です。

- `Tc_log_page_waits`

この変数は、リカバリログのメモリーマップ実装で、サーバーがトランザクションをコミットできず、ログ内の空きページを待機する必要がある場合に毎回増加します。この値が大きい場合、(`--log-tc-size` オプションで) ログサイズを増加した方がよい場合もあります。この変数は、バイナリログベースのリカバリで、2 フェーズコミットが進行中のためバイナリログをクローズできない場合に毎回増加します。(クローズ操作は、このようなトランザクションがすべて終了するまで待機します。)

- `Threads_cached`

スレッドキャッシュ内のスレッド数。

- `Threads_connected`

現在開いている接続の数。

- `Threads_created`

接続を処理するために作成されたスレッドの数。 `Threads_created` が大きい場合、 `thread_cache_size` の値を大きくした方がよい場合もあります。 キャッシュミス率は `Threads_created/Connections` として計算できます。

- [Threads_running](#)

スリープ状態ではないスレッド数。

- [Uptime](#)

サーバーが作動している秒数。

- [Uptime_since_flush_status](#)

最新の `FLUSH STATUS` ステートメントの秒数。

5.1.11 サーバー SQL モード

MySQL Server は異なる SQL モードで動作でき、 `sql_mode` システム変数の値に応じて異なるクライアントにこれらの異なるモードを適用できます。 DBA はサイトサーバーの動作要件に一致するグローバル SQL モードを設定でき、各アプリケーションはアプリケーションのセッション SQL モードをアプリケーション独自の要件に設定できます。

モードは MySQL がサポートする SQL 構文と、MySQL が実行するデータ検証に影響します。 これにより、MySQL をさまざまな環境で使用したり、MySQL をほかのデータベースサーバーと一緒に使用したりすることが、さらに容易になります。

- [SQL モードの設定](#)

- [もっとも重要な SQL モード](#)

- [SQL モードの完全なリスト](#)

- [組み合わせ SQL モード](#)

- [厳密な SQL モード](#)

- [IGNORE キーワードと厳密な SQL モードの比較](#)

MySQL のサーバー SQL モードについてのよくある質問に対する回答は、 [セクションA.3「MySQL 8.0 FAQ: サーバー SQL モード」](#) を参照してください。

InnoDB テーブルを操作するとき、 `innodb_strict_mode` システム変数についても考慮してください。 これによって、InnoDB テーブルの追加のエラー検査が可能になります。

SQL モードの設定

MySQL 8.0 のデフォルトの SQL モードには、次のモードが含まれます: `ONLY_FULL_GROUP_BY`, `STRICT_TRANS_TABLES`, `NO_ZERO_IN_DATE`, `NO_ZERO_DATE`, `ERROR_FOR_DIVISION_BY_ZERO` および `NO_ENGINE_SUBSTITUTION`。

サーバー起動時に SQL モードを設定するには、コマンド行で `--sql-mode="modes"` オプションを使用するか、 `my.cnf` (Unix オペレーティングシステム) または `my.ini` (Windows) などのオプションファイル内で `sql-mode="modes"` を使用します。 `modes` は、カンマで区切られるさまざまなモードのリストです。 SQL モードを明示的にクリアするには、コマンド行で `--sql-mode=""` を使用するかオプションファイル内で `sql-mode=""` を使用して、SQL モードを空の文字列に設定します。

注記

MySQL インストールプログラムはインストールプロセス中に SQL モードを構成することがあります。

SQL モードがデフォルトまたは期待されているモードと異なる場合、サーバーが起動時に読み取るオプションファイル内の設定を確認してください。

SQL モードを実行時に変更するには、 `SET` ステートメントを使用して、グローバルまたはセッションの `sql_mode` システム変数を設定します。

```
SET GLOBAL sql_mode = 'modes';  
SET SESSION sql_mode = 'modes';
```

GLOBAL 変数を設定するには、[SYSTEM_VARIABLES_ADMIN](#) 権限 (または非推奨の [SUPER](#) 権限) が必要であり、その時点から接続するすべてのクライアントの操作に影響します。SESSION 変数を設定すると、現在のクライアントにのみ影響します。すべてのクライアントは、自分のセッションの `sql_mode` 値をいつでも変更できます。

現在のグローバルまたはセッションの `sql_mode` 設定を確認するには、その値を選択します:

```
SELECT @@GLOBAL.sql_mode;  
SELECT @@SESSION.sql_mode;
```

重要

SQL モードおよびユーザー定義のパーティショニング。パーティション化されたテーブルを作成してデータを挿入したあとでサーバー SQL モードを変更すると、このようなテーブルの動作が大きく変更される可能性があり、データが失われたり破損したりすることがあります。ユーザー定義のパーティショニングを使用したテーブルを作成したら、SQL モードを変更しないことを強くお勧めします。

パーティションテーブルをレプリケートする場合、ソースとレプリカで異なる SQL モードを使用すると、問題が発生する可能性もあります。最良の結果を得るには、ソースとレプリカで常に同じサーバー SQL モードを使用する必要があります。

詳細は、[セクション24.6「パーティショニングの制約と制限」](#)を参照してください。

もっとも重要な SQL モード

次に、多くの場合でもっとも重要な `sql_mode` 値を示します。

- [ANSI](#)

このモードは、構文および動作が標準の SQL にさらに緊密に準拠するように変更します。これは、このセクションの末尾にリストされている、特殊な[組み合わせモード](#)の1つです。

- [STRICT_TRANS_TABLES](#)

値を指定したとおりにトランザクションテーブルに挿入できない場合、ステートメントを中止します。非トランザクションテーブルの場合、値が単一行ステートメントで発生するか、複数行ステートメントの先頭行で発生した場合、ステートメントを中止します。詳細については、このセクションのあとの方で説明します。

- [TRADITIONAL](#)

MySQL を「従来型の」SQL データベースシステムのように動作させます。このモードを簡単に説明すると、カラムに不正な値を挿入したときに「警告ではなくエラーを返し」ます。これは、このセクションの末尾にリストされている、特殊な[組み合わせモード](#)の1つです。

注記

TRADITIONAL モードを有効にすると、エラーが発生するとすぐに `INSERT` または `UPDATE` が中断されます。非トランザクションストレージエンジンを使用している場合、エラーの前に行われたデータ変更はロールバックされず、「一部完了」が更新される可能性があるため、これは望ましくない可能性があります。

このマニュアルの「厳密モード」とは、[STRICT_TRANS_TABLES](#) または [STRICT_ALL_TABLES](#) のいずれかあるいは両方が有効なモードを意味します。

SQL モードの完全なリスト

次のリストは、サポートされるすべての SQL モードについて説明しています。

- [ALLOW_INVALID_DATES](#)

日付の完全な検査を実行しません。月が 1 から 12 までの範囲にあることと、日が 1 から 31 までの範囲にあることのみ検査します。これは、3 つの異なるフィールドで年、月および日を取得し、日付検証なしでユーザーが挿入した内容を正確に格納する Web アプリケーションに役立ちます。このモードは `DATE` および `DATETIME` カラムに適用されます。`TIMESTAMP` カラムは有効な日付が常に必要なため、このカラムには適用されません。

`ALLOW_INVALID_DATES` が無効になっている場合、サーバーでは月と日の値が有効である必要があり、それぞれ 1 から 12 および 1 から 31 の範囲内にあるだけではありません。厳密モードが無効になっていると、`'2004-04-31'` のような無効な日付は `'0000-00-00'` に変換され、警告メッセージが表示されます。厳密モードが有効なときは、無効な日付によってエラーが発生します。このような日付を許可するには、`ALLOW_INVALID_DATES` を有効にします。

• `ANSI_QUOTES`

"は、文字列引用符文字としてではなく、識別子引用符文字（引用符文字など）として扱います。このモードを有効にして、識別子を引用するために ` を引き続き使用できます。`ANSI_QUOTES` が有効な場合、リテラル文字列は識別子として解釈されるため、二重引用符を使用して引用符を使用することはできません。

• `ERROR_FOR_DIVISION_BY_ZERO`

`ERROR_FOR_DIVISION_BY_ZERO` モードは、`MOD(N,0)` を含むゼロ除算の処理に影響します。データ変更操作 (`INSERT`、`UPDATE`) の場合、この効果は厳密 SQL モードが有効であるかどうかにもよります。

- このモードが有効でない場合、ゼロによる除算は `NULL` を挿入し、警告は生成されません。
- このモードが有効な場合、ゼロによる除算は `NULL` を挿入し、警告が生成されます。
- このモードおよび厳密モードが有効な場合、ゼロによる除算はエラーを生成しますが、`IGNORE` も指定されている場合は例外です。`INSERT IGNORE` および `UPDATE IGNORE` の場合、ゼロによる除算は `NULL` を挿入し、警告が生成されます。

`SELECT` の場合、ゼロによる除算は `NULL` を返します。`ERROR_FOR_DIVISION_BY_ZERO` を有効にすると、厳密モードが有効かどうかに関係なく警告も生成されます。

`ERROR_FOR_DIVISION_BY_ZERO` は非推奨です。`ERROR_FOR_DIVISION_BY_ZERO` は厳密モードの一部ではありませんが、厳密モードとともに使用する必要があり、デフォルトで有効になっています。厳密モードも有効にせずに `ERROR_FOR_DIVISION_BY_ZERO` が有効になっている場合、またはその逆の場合は、警告が発生します。

`ERROR_FOR_DIVISION_BY_ZERO` は非推奨であるため、将来の MySQL リリースでは、個別のモード名として削除され、厳密な SQL モードの影響に含まれることが予想されます。

• `HIGH_NOT_PRECEDENCE`

`NOT` 演算子の存在によって、`NOT a BETWEEN b AND c` のような式は `NOT (a BETWEEN b AND c)` として構文解析されます。一部の古い MySQL バージョンでは、この式は `(NOT a) BETWEEN b AND c` として構文解析されます。優先順位を高める以前の動作は、`HIGH_NOT_PRECEDENCE` の SQL モードを有効にすることによって取得できます。

```
mysql> SET sql_mode = "";
mysql> SELECT NOT 1 BETWEEN -5 AND 5;
-> 0
mysql> SET sql_mode = 'HIGH_NOT_PRECEDENCE';
mysql> SELECT NOT 1 BETWEEN -5 AND 5;
-> 1
```

• `IGNORE_SPACE`

関数名と（文字の間にスペースを許可します。これにより、組み込み関数名が予約語として扱われます。その結果、関数名と同じ識別子は、[セクション9.2「スキーマオブジェクト名」](#)に記載されているように引用符で囲む必要があります。たとえば、`COUNT()` 関数があるため、次のステートメントで `count` をテーブル名として使用すると、エラーが発生します。

```
mysql> CREATE TABLE count (i INT);
ERROR 1064 (42000): You have an error in your SQL syntax
```


テーブル名を引用符で囲んでください。

```
mysql> CREATE TABLE `count` (i INT);  
Query OK, 0 rows affected (0.00 sec)
```

IGNORE_SPACE SQL モードは、ユーザー定義関数またはストアドファンクションではなく、組み込み関数に適用されます。IGNORE_SPACE が有効かどうかにかかわらず、UDF またはストアドファンクション名のあとにスペースを入れることが常に許可されます。

IGNORE_SPACE に関する詳細は、[セクション9.2.5「関数名の構文解析と解決」](#)を参照してください。

- **NO_AUTO_VALUE_ON_ZERO**

NO_AUTO_VALUE_ON_ZERO は AUTO_INCREMENT カラムの処理に影響します。通常は、NULL または 0 をカラムに挿入することによって、カラムの次のシーケンス番号を生成します。NO_AUTO_VALUE_ON_ZERO は 0 のこの動作を抑制するため、NULL のみが次のシーケンス番号を生成します。

このモードは、テーブルの AUTO_INCREMENT カラムに 0 が格納されている場合に便利ことがあります。(ただし、0 を格納することは、推奨される方法ではありません。)たとえば、mysqldump でテーブルをダンプして、テーブルをリロードする場合、MySQL は通常、0 という値を検出すると、新たなシーケンス番号を生成するため、その結果、ダンプされたものとは異なる内容を持つテーブルになります。ダンプファイルをリロードする前に NO_AUTO_VALUE_ON_ZERO を有効にすると、この問題は解決します。このため、mysqldump の出力には、NO_AUTO_VALUE_ON_ZERO を有効にするステートメントが自動的に含まれます。

- **NO_BACKSLASH_ESCAPES**

文字列および識別子内のエスケープ文字としてのバックスラッシュ文字 (\) の使用を無効にします。このモードを有効にすると、バックスラッシュはほかの文字のように通常の文字になります。

- **NO_DIR_IN_CREATE**

テーブルを作成するとき、INDEX DIRECTORY および DATA DIRECTORY ディレクティブをすべて無視します。このオプションは、レプリカサーバーで役立ちます。

- **NO_ENGINE_SUBSTITUTION**

CREATE TABLE または ALTER TABLE などのステートメントが無効またはコンパイルされていないストレージエンジンを指定したとき、デフォルトのストレージエンジンの自動置換を制御します。

デフォルトでは、NO_ENGINE_SUBSTITUTION は有効です。

ストレージエンジンは実行時にプラグブルであるため、利用できないエンジンも同様に扱われます。

NO_ENGINE_SUBSTITUTION を無効にすると、CREATE TABLE については、目的のエンジンが利用できない場合にデフォルトエンジンが使用されて警告が発生します。ALTER TABLE では、警告が発生してテーブルは変更されません。

NO_ENGINE_SUBSTITUTION を有効にすると、目的のエンジンが利用できない場合にエラーが発生し、テーブルは作成または変更されません。

- **NO_UNSIGNED_SUBTRACTION**

一方が UNSIGNED 型のときに 2 つの整数値の間で減算を行うと、デフォルトでは符号なしの結果が生成されます。それ以外の場合は、エラーが発生します:

```
mysql> SET sql_mode = "  
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT CAST(0 AS UNSIGNED) - 1;  
ERROR 1690 (22003): BIGINT UNSIGNED value is out of range in '(cast(0 as unsigned) - 1)'
```

NO_UNSIGNED_SUBTRACTION SQL モードが有効な場合は、結果は負になります。

```
mysql> SET sql_mode = 'NO_UNSIGNED_SUBTRACTION';  
mysql> SELECT CAST(0 AS UNSIGNED) - 1;
```

```

+-----+
| CAST(0 AS UNSIGNED) - 1 |
+-----+
|          -1 |
+-----+

```

このような演算の結果を使用して `UNSIGNED` 整数カラムが更新されると、結果はそのカラム型の最大値に切り落とされます。`NO_UNSIGNED_SUBTRACTION` が有効になっている場合は、0 に切り落とされます。厳密な SQL モードを有効にすると、エラーが発生し、カラムは変更されません。

`NO_UNSIGNED_SUBTRACTION` が有効な場合、いずれかのオペランドが符号なしであっても、減算の結果は符号付きになります。たとえば、テーブル `t1` のカラム `c2` のタイプと、テーブル `t2` のカラム `c2` のタイプを比較します。

```

mysql> SET sql_mode="";
mysql> CREATE TABLE test (c1 BIGINT UNSIGNED NOT NULL);
mysql> CREATE TABLE t1 SELECT c1 - 1 AS c2 FROM test;
mysql> DESCRIBE t1;
+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| c2    | bigint(21) unsigned | NO   |     | 0       |       |
+-----+-----+-----+-----+-----+

mysql> SET sql_mode="NO_UNSIGNED_SUBTRACTION";
mysql> CREATE TABLE t2 SELECT c1 - 1 AS c2 FROM test;
mysql> DESCRIBE t2;
+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| c2    | bigint(21)   | NO   |     | 0       |       |
+-----+-----+-----+-----+-----+

```

つまり、`BIGINT UNSIGNED` はすべてのコンテキストで 100% 使用できるわけではありません。 [セクション 12.11 「キャスト関数と演算子」](#) を参照してください。

• `NO_ZERO_DATE`

`NO_ZERO_DATE` モードは、サーバーが `'0000-00-00'` を有効な日付として許可するかどうかに影響します。この影響は、厳密な SQL モードが有効かどうかにも依存します。

- このモードが有効でない場合、`'0000-00-00'` は許可され、挿入によって警告が生成されません。
- このモードが有効な場合、`'0000-00-00'` は許可され、挿入によって警告が生成されます。
- このモードおよび厳密モードが有効な場合、`IGNORE` も指定されている場合を除き、`'0000-00-00'` は許可されず、挿入によってエラーが生成されます。`INSERT IGNORE` および `UPDATE IGNORE` の場合、`'0000-00-00'` は許可され、挿入によって警告が生成されます。

`NO_ZERO_DATE` は非推奨です。`NO_ZERO_DATE` は厳密モードの一部ではありませんが、厳密モードとともに使用する必要があり、デフォルトで有効になっています。厳密モードも有効にせずに `NO_ZERO_DATE` が有効になっている場合、またはその逆の場合は、警告が発生します。

`NO_ZERO_DATE` は非推奨であるため、将来の MySQL リリースでは、個別のモード名として削除され、厳密な SQL モードの影響に含まれることが予想されます。

• `NO_ZERO_IN_DATE`

`NO_ZERO_IN_DATE` モードは、年の部分は非ゼロであるが月または日の部分が 0 である日付をサーバーが許可するかどうかに影響します。(このモードは `'2010-00-01'` や `'2010-01-00'` などの日付に影響しますが、`'0000-00-00'` に

は影響しません。サーバーが '0000-00-00' を許可するかどうかを制御するには、`NO_ZERO_DATE` モードを使用してください。) `NO_ZERO_IN_DATE` の影響は、厳密 SQL モードが有効かどうかにも依存します。

- このモードが有効でない場合、ゼロ部分を含む日付は許可され、挿入によって警告が生成されません。
- このモードが有効な場合、ゼロ部分を含む日付は '0000-00-00' として挿入され、警告が生成されます。
- このモードおよび厳密モードが有効な場合は、`IGNORE` も指定されている場合を除き、ゼロ部分を含む日付は許可されず、挿入によってエラーが生成されます。`INSERT IGNORE` および `UPDATE IGNORE` の場合、ゼロ部分を含む日付は '0000-00-00' として挿入され、警告が生成されます。

`NO_ZERO_IN_DATE` は非推奨です。`NO_ZERO_IN_DATE` は厳密モードの一部ではありませんが、厳密モードとともに使用する必要があり、デフォルトで有効になっています。厳密モードも有効にせずに `NO_ZERO_IN_DATE` が有効になっている場合、またはその逆の場合は、警告が発生します。

`NO_ZERO_IN_DATE` は非推奨であるため、将来の MySQL リリースでは、個別のモード名として削除され、厳密な SQL モードの影響に含まれることが予想されます。

• `ONLY_FULL_GROUP_BY`

選択リスト、`HAVING` 条件または `ORDER BY` リストが、`GROUP BY` 句で指定されておらず、機能的に `GROUP BY` カラムに依存しない (一意に決定される) 非集計カラムを参照するクエリーを拒否します。

標準 SQL に対する MySQL 拡張機能を使用すると、`HAVING` 句内で選択リスト内のエイリアス式を参照できます。`HAVING` 句は、`ONLY_FULL_GROUP_BY` が有効かどうかに関係なく、エイリアスを参照できます。

その他の説明と例については、[セクション12.20.3「MySQL での GROUP BY の処理」](#) を参照してください。

• `PAD_CHAR_TO_FULL_LENGTH`

デフォルトでは、末尾のスペースは、取得時に `CHAR` カラム値から削除されます。`PAD_CHAR_TO_FULL_LENGTH` が有効な場合、削除は行われず、取得された `CHAR` 値は完全な長さになるまでパディングされます。このモードは `VARCHAR` カラムには適用されず、この場合、末尾のスペースは取得時に保持されます。

注記

MySQL 8.0.13 では、`PAD_CHAR_TO_FULL_LENGTH` は非推奨です。MySQL の将来のバージョンで削除されることが予想されます。

```
mysql> CREATE TABLE t1 (c1 CHAR(10));
Query OK, 0 rows affected (0.37 sec)

mysql> INSERT INTO t1 (c1) VALUES('xy');
Query OK, 1 row affected (0.01 sec)

mysql> SET sql_mode = "";
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT c1, CHAR_LENGTH(c1) FROM t1;
+-----+-----+
| c1 | CHAR_LENGTH(c1) |
+-----+-----+
| xy |          2 |
+-----+-----+
1 row in set (0.00 sec)

mysql> SET sql_mode = 'PAD_CHAR_TO_FULL_LENGTH';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT c1, CHAR_LENGTH(c1) FROM t1;
+-----+-----+
| c1 | CHAR_LENGTH(c1) |
+-----+-----+
| xy |          10 |
+-----+-----+
```

```
1 row in set (0.00 sec)
```

- [PIPES_AS_CONCAT](#)

`||` を、`OR` のシノニムとしてではなく (`CONCAT()`) と同様に) 文字列連結演算子として扱います。

- [REAL_AS_FLOAT](#)

`REAL` を `FLOAT` のシノニムとして扱います。デフォルトでは、MySQL は `REAL` を `DOUBLE` のシノニムとして扱います。

- [STRICT_ALL_TABLES](#)

すべてのストレージエンジンに対して厳密な SQL モードを有効にします。無効なデータ値は拒否されます。詳細は、[厳密な SQL モード](#)を参照してください。

- [STRICT_TRANS_TABLES](#)

トランザクションストレージエンジンに対して、および可能な場合は非トランザクションストレージエンジンに対して、厳密な SQL モードを有効にします。詳細は、[厳密な SQL モード](#)を参照してください。

- [TIME_TRUNCATE_FRACTIONAL](#)

小数秒部分を持つ `TIME`、`DATE` または `TIMESTAMP` 値を、同じタイプで小数桁数が少ないカラムに挿入するときに、端数処理または切捨てが行われるかどうかを制御します。デフォルトの動作では、丸めが使用されます。このモードが有効な場合は、かわりに切捨てが行われます。次の一連のステートメントは、違いを示しています:

```
CREATE TABLE t (id INT, tval TIME(1));
SET sql_mode="";
INSERT INTO t (id, tval) VALUES(1, 1.55);
SET sql_mode="TIME_TRUNCATE_FRACTIONAL";
INSERT INTO t (id, tval) VALUES(2, 1.55);
```

結果のテーブルの内容は次のようになります。最初の値は端数処理の対象となり、次に切捨ての対象となります:

```
mysql> SELECT id, tval FROM t ORDER BY id;
+----+-----+
| id | tval   |
+----+-----+
|  1 | 00:00:01.6 |
|  2 | 00:00:01.5 |
+----+-----+
```

[セクション11.2.6「時間値での小数秒」](#)も参照してください。

組み合わせ SQL モード

次の特殊なモードは、前リストのモード値の組み合わせを表す省略表現として提供されています。

- [ANSI](#)

`REAL_AS_FLOAT`、`PIPES_AS_CONCAT`、`ANSI_QUOTES`、`IGNORE_SPACE` および `ONLY_FULL_GROUP_BY` と同等です。

また `ANSI` モードは、外部参照 `S(outer_ref)` を持つ設定関数 `S` が、外部参照が解決される外部クエリー内で集約できない場合のクエリーに、サーバーがエラーを返します。このようなクエリーを次に示します。

```
SELECT * FROM t1 WHERE t1.a IN (SELECT MAX(t1.b) FROM t2 WHERE ...);
```

ここで、`MAX(t1.b)` はそのクエリーの `WHERE` 句に指定されているため、外部クエリーで集約できません。標準的な SQL では、この状況ではエラーになります。ANSI モードが有効でない場合、サーバーはそのようなクエリー内の `S(outer_ref)` を、`S(const)` を解釈する同じ方法で扱います。

[セクション1.7「MySQL の標準への準拠」](#)を参照してください。

- [TRADITIONAL](#)

TRADITIONAL は、STRICT_TRANS_TABLES, STRICT_ALL_TABLES, NO_ZERO_IN_DATE, NO_ZERO_DATE, ERROR_FOR_DIVISION_BY_ZERO および NO_ENGINE_SUBSTITUTION と同等です。

厳密な SQL モード

厳密モードは、MySQL が INSERT や UPDATE などのデータ変更ステートメントで無効な値または欠落した値を処理する方法を制御します。値はいくつかの理由で無効になることがあります。たとえば、カラムに対して正しくないデータ型を持っていたり、範囲外であったりすることがあります。値の欠落が発生するのは、挿入される新しい行の非 NULL カラムに値が含まれておらず、そのカラムに明示的な DEFAULT 句が定義されていない場合です。(NULL カラムの場合、値が欠落しているときは NULL が挿入されます。) 厳密モードは、CREATE TABLE などの DDL ステートメントにも影響します。

厳密モードが有効でない場合、MySQL は無効または欠落した値に対して調整された値を挿入し、警告を生成します (セクション13.7.7.42「SHOW WARNINGS ステートメント」を参照してください)。厳密モードでは、INSERT IGNORE または UPDATE IGNORE を使用すると、この動作を実行できます。

データを変更しない SELECT などのステートメントの場合、厳密モードでは無効な値はエラーでなく警告を生成しません。

厳密モードでは、最大キー長を超えるキーを作成しようとするときエラーが発生します。厳密モードが有効になっていない場合、これにより警告が発生し、キーが最大キー長に切り捨てられます。

厳密モードは、外部キー制約が検査されるかどうかに影響されません。foreign_key_checks を検査に使用できます。(セクション5.1.8「サーバーシステム変数」を参照してください。)

厳密な SQL モードは、STRICT_ALL_TABLES または STRICT_TRANS_TABLES のいずれかが有効な場合に有効になりますが、これらのモードの影響はいくらか異なります。

- トランザクションテーブルの場合、STRICT_ALL_TABLES または STRICT_TRANS_TABLES のいずれかが有効なとき、データ変更ステートメント内の無効な値または欠落した値に対してエラーが発生します。ステートメントは中止されてロールバックされます。
- 非トランザクションテーブルの場合、挿入または更新される最初の行に不適切な値があるとき、どちらのモードでも動作は同じになり、ステートメントが中止されて、テーブルはそのまま変更されません。ステートメントが複数行を挿入または変更し、2 行目以降に不適切な値がある場合、どちらの厳密モードが有効になっているかによって結果は異なります。
- STRICT_ALL_TABLES では、MySQL はエラーを返し、残りの行を無視します。ただし、それより前の行が挿入または更新されているため、結果は部分更新となります。これを防ぐには、テーブルを変更することなく中止できる単一行ステートメントを使用します。
- STRICT_TRANS_TABLES では、MySQL は無効な値をカラムについてのもっとも近い有効な値に変換し、調整された値を挿入します。値が欠落している場合、MySQL はカラムデータ型の暗黙のデフォルト値を挿入します。いずれの状況でも MySQL はエラーでなく警告を生成し、ステートメントの処理を続行します。暗黙的なデフォルトについては、セクション11.6「データ型デフォルト値」に記載されています。

厳密モードは、日付のゼロ、ゼロ日付およびゼロによる除算の処理に次のように影響します:

- 厳密モードは、MOD(N,0) を含むゼロによる除算の処理に影響します:

データ変更操作 (INSERT、UPDATE):

- 厳密モードが有効になっていない場合、ゼロ除算によって NULL が挿入され、警告は生成されません。
- 厳密モードが有効な場合、IGNORE も指定されていないかぎり、ゼロによる除算でエラーが発生します。INSERT IGNORE および UPDATE IGNORE の場合、ゼロによる除算は NULL を挿入し、警告が生成されます。

SELECT の場合、ゼロによる除算は NULL を返します。厳密モードを有効にすると、警告も生成されます。

- 厳密モードは、サーバーが'0000-00-00'を有効な日付として許可するかどうかに影響します:

- 厳密モードが有効になっていない場合、'0000-00-00'は許可され、挿入によって警告は生成されません。

- 厳密モードが有効な場合、`IGNORE` も指定されていないかぎり、`'0000-00-00'`は許可されず、挿入によってエラーが生成されます。`INSERT IGNORE` および `UPDATE IGNORE` の場合、`'0000-00-00'` は許可され、挿入によって警告が生成されます。
- 厳密モードは、年の部分がゼロ以外で月または日の部分が 0 (`'2010-00-01'`や`'2010-01-00'`などの日付) である日付をサーバーが許可するかどうかに影響します:
 - 厳密モードが有効になっていない場合、ゼロ部分の日付は許可され、挿入によって警告は生成されません。
 - 厳密モードが有効な場合、`IGNORE` も指定されていないかぎり、ゼロ部分の日付は許可されず、挿入によってエラーが生成されます。`INSERT IGNORE` および `UPDATE IGNORE` の場合、ゼロ部分の日付は`'0000-00-00'` (`IGNORE` で有効とみなされます) として挿入され、警告が生成されます。

`IGNORE` に関する厳密モードの詳細は、[IGNORE キーワードと厳密な SQL モードの比較](#) を参照してください。

厳密モードは、`ERROR_FOR_DIVISION_BY_ZERO`、`NO_ZERO_DATE` および `NO_ZERO_IN_DATE` モードと組み合わせた日付でのゼロ、ゼロ日付およびゼロによる除算の処理に影響します。

IGNORE キーワードと厳密な SQL モードの比較

このセクションでは、`IGNORE` キーワード (エラーを警告にダウングレード) および厳密な SQL モード (警告をエラーにアップグレード) のステートメント実行への影響を比較します。影響を受けるステートメントと、それらが適用されるエラーについて説明します。

次のテーブルに、デフォルトでエラーが生成された場合と警告が生成された場合のステートメントの動作のサマリー比較を示します。デフォルトでエラーが生成されるのは、`NOT NULL` カラムへの `NULL` の挿入などです。デフォルトでは、誤ったデータ型の値をカラムに挿入するという警告が生成されます (文字列`'abc'`を整数カラムに挿入するなど)。

操作モード	ステートメントのデフォルトが Error の場合	ステートメントのデフォルトが警告の場合
<code>IGNORE</code> または厳密な SQL モードなし	エラー	警告
<code>IGNORE</code> を使用	警告	警告 (<code>IGNORE</code> または厳密な SQL モードを使用しない場合と同じ)
厳密な SQL モード	エラー (<code>IGNORE</code> または厳密な SQL モードを使用しない場合と同じ)	エラー
<code>IGNORE</code> および厳密な SQL モード	警告	警告

テーブルから描画する結論の 1 つは、`IGNORE` キーワードと厳密な SQL モードの両方が有効な場合に、`IGNORE` が優先されることです。つまり、`IGNORE` と厳密な SQL モードはエラー処理に反対の影響を与えると考えられますが、一緒に使用しても取り消されません。

- [IGNORE がステートメントの実行に与える影響](#)
- [ステートメントの実行に対する厳密な SQL モードの影響](#)

IGNORE がステートメントの実行に与える影響

MySQL のいくつかのステートメントでは、オプションの `IGNORE` キーワードがサポートされます。このキーワードを使用すると、サーバーは特定のタイプのエラーをダウングレードし、かわりに警告を生成します。複数行のステートメントの場合、`IGNORE` はステートメントを中断するのではなく、次の行にスキップします。(無視できないエラーの場合、`IGNORE` キーワードに関係なくエラーが発生します。)

例: テーブル `t` に主キーカラム `i` がある場合、複数の行に同じ値の `i` を挿入しようとする、通常、重複キーエラーが発生します:

```
mysql> INSERT INTO t (i) VALUES(1),(1);
```



```
ERROR 1062 (23000): Duplicate entry '1' for key 't.PRIMARY'
```

IGNORE では、重複キーを含む行は挿入されませんが、エラーのかわりに警告が発生します:

```
mysql> INSERT IGNORE INTO t (i) VALUES(1),(1);
Query OK, 1 row affected, 1 warning (0.01 sec)
Records: 2 Duplicates: 1 Warnings: 1

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Warning | 1062 | Duplicate entry '1' for key 't.PRIMARY' |
+-----+-----+-----+
1 row in set (0.00 sec)
```

次のステートメントは、**IGNORE** キーワードをサポートしています:

- **CREATE TABLE ... SELECT: IGNORE** は、ステートメントの **CREATE TABLE** または **SELECT** 部分には適用されませんが、**SELECT** によって生成された行のテーブルへの挿入には適用されます。一意キー値の既存の行を複製する行は破棄されます。
- **DELETE: IGNORE** では、行の削除プロセス中に MySQL でエラーが無視されます。
- **INSERT: IGNORE** では、一意キー値の既存の行を複製する行は破棄されます。データ変換エラーの原因となる値に設定された行は、かわりに最も近い有効な値に設定されます。

特定の値に一致するパーティションが見つからないパーティションテーブルの場合、**IGNORE** では、一致しない値を含む行に対して挿入操作が暗黙的に失敗します。

- **LOAD DATA, LOAD XML: IGNORE** では、一意キー値の既存の行を複製する行は破棄されます。
- **UPDATE: IGNORE** では、一意キー値で重複キーの競合が発生した行は更新されません。データ変換エラーの原因になる値に更新された行は、代わりに、もっとも近い有効な値に更新されます。

IGNORE キーワードは、無視できる次のエラーに適用されます:

```
ER_BAD_NULL_ERROR
ER_DUP_ENTRY
ER_DUP_ENTRY_WITH_KEY_NAME
ER_DUP_KEY
ER_NO_PARTITION_FOR_GIVEN_VALUE
ER_NO_PARTITION_FOR_GIVEN_VALUE_SILENT
ER_NO_REFERENCED_ROW_2
ER_ROW_DOES_NOT_MATCH_GIVEN_PARTITION_SET
ER_ROW_IS_REFERENCED_2
ER_SUBQUERY_NO_1_ROW
ER_VIEW_CHECK_FAILED
```

ステートメントの実行に対する厳密な SQL モードの影響

MySQL Server は異なる SQL モードで動作でき、**sql_mode** システム変数の値に応じて異なるクライアントにこれらの異なるモードを適用できます。「strict」SQL モードでは、サーバーは特定の警告をエラーにアップグレードしません。

たとえば、非厳密 SQL モードでは、文字列'abc'を整数カラムに挿入すると、値が 0 に変換され、警告が表示されません:

```
mysql> SET sql_mode = "";
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t (i) VALUES('abc');
Query OK, 1 row affected, 1 warning (0.01 sec)

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
```

```
+-----+-----+-----+
| Warning | 1366 | Incorrect integer value: 'abc' for column 'i' at row 1 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

厳密な SQL モードでは、無効な値は次のエラーで拒否されます:

```
mysql> SET sql_mode = 'STRICT_ALL_TABLES';
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t (i) VALUES('abc');
ERROR 1366 (HY000): Incorrect integer value: 'abc' for column 'i' at row 1
```

`sql_mode` システム変数の可能な設定の詳細は、[セクション5.1.11「サーバー SQL モード」](#) を参照してください。

厳密な SQL モードは、一部の値が範囲外であるか、無効な行がテーブルに挿入またはテーブルから削除される可能性がある場合に、次のステートメントに適用されます:

- ALTER TABLE
- CREATE TABLE
- CREATE TABLE ... SELECT
- DELETE (単一テーブルと複数テーブルの両方)
- INSERT
- LOAD DATA
- LOAD XML
- SELECT SLEEP()
- UPDATE (単一テーブルと複数テーブルの両方)

ストアプログラム内では、厳密モードが有効なときにプログラムが定義されていた場合、リストされている型の個々のステートメントは厳密な SQL モードで実行されます。

厳密な SQL モードは、入力値が無効または欠落しているエラーのクラスを表す次のエラーに適用されます。カラムのデータ型が間違っているか、値が範囲外である可能性がある場合、値は無効です。挿入する新しい行の定義に明示的な `DEFAULT` 句がない `NOT NULL` カラムの値が含まれていない場合、値は欠落しています。

```
ER_BAD_NULL_ERROR
ER_CUT_VALUE_GROUP_CONCAT
ER_DATA_TOO_LONG
ER_DATETIME_FUNCTION_OVERFLOW
ER_DIVISION_BY_ZERO
ER_INVALID_ARGUMENT_FOR_LOGARITHM
ER_NO_DEFAULT_FOR_FIELD
ER_NO_DEFAULT_FOR_VIEW_FIELD
ER_TOO_LONG_KEY
ER_TRUNCATED_WRONG_VALUE
ER_TRUNCATED_WRONG_VALUE_FOR_FIELD
ER_WARN_DATA_OUT_OF_RANGE
ER_WARN_NULL_TO_NOTNULL
ER_WARN_TOO_FEW_RECORDS
ER_WRONG_ARGUMENTS
ER_WRONG_VALUE_FOR_TYPE
WARN_DATA_TRUNCATED
```

注記

継続的な MySQL 開発では新しいエラーが定義されるため、前述のリストにないエラーがあり、厳密な SQL モードが適用される場合があります。

5.1.12 接続管理

このセクションでは、MySQL Server が接続を管理する方法について説明します。これには、使用可能な接続インターフェース、サーバーが接続ハンドラスレッドを使用する方法、管理接続インターフェースの詳細、および DNS ルックアップの管理が含まれます。

5.1.12.1 接続インターフェース

このセクションでは、MySQL サーバーがクライアント接続を管理する方法について説明します。

- [ネットワークインターフェースと接続マネージャスレッド](#)
- [クライアント接続スレッド管理](#)
- [接続ボリューム管理](#)

ネットワークインターフェースと接続マネージャスレッド

サーバーは、複数のネットワークインターフェースでクライアント接続をリスニングできます。接続マネージャスレッドは、サーバーがリスニングするネットワークインターフェース上のクライアント接続リクエストを処理します：

- どのプラットフォームでも、1つのマネージャスレッドが TCP/IP 接続要求を処理します。
- Unix では、同じマネージャスレッドが Unix ソケットファイルの接続リクエストも処理します。
- Windows では、1つのマネージャスレッドが共有メモリ接続要求を処理し、もう1つのマネージャスレッドが名前付きパイプ接続要求を処理します。
- すべてのプラットフォームで、追加のネットワークインターフェースを有効にして、管理 TCP/IP 接続リクエストを受け入れることができます。このインターフェースは、「普通」 TCP/IP リクエストを処理するマネージャスレッド、または別のスレッドを使用できます。

サーバーは、待機していないインターフェースを処理するためのスレッドを作成しません。たとえば、Windows サーバーで名前付きパイプ接続のサポートが有効になっていない場合、これらの接続を処理するスレッドは作成されません。

個々のサーバープラグインまたはコンポーネントは、独自の接続インターフェースを実装できます：

- X プラグインを使用すると、MySQL Server は X プロトコルを使用してクライアントと通信できます。 [セクション 20.5 「X プラグイン」](#) を参照してください。

クライアント接続スレッド管理

接続マネージャスレッドは、各クライアント接続を、その接続の認証および要求を処理する専用スレッドに関連付けます。マネージャスレッドは、必要に応じて新しいスレッドを作成しますが、まずスレッドキャッシュを調べて接続に使用できるスレッドが含まれているかどうかを確認することによって、それを回避することを試みます。接続が終了すると、スレッドキャッシュが満杯でない場合は、そのスレッドがスレッドキャッシュに返されます。

この接続スレッドモデルでは、現在接続しているクライアントと同数のスレッドが存在し、多数の接続を処理するためにサーバーのワークロードを拡大する必要がある場合にはいくつかの欠点があります。たとえば、スレッドの作成と破棄の負荷が大きくなります。また、各スレッドにスタック領域などのサーバーリソースとカーネルリソースが必要になります。多数の同時接続に対応するには、スレッドあたりのスタックサイズは小さく保つ必要があり、それが小さくなりすぎるか、またはサーバーで大量のメモリーを消費することになる状況につながります。ほかのリソースを使い果たす可能性もあり、スケジューリングのオーバーヘッドがかなり大きくなる可能性があります。

MySQL Enterprise Edition には、オーバーヘッドを削減してパフォーマンスを向上させるために設計された代替スレッド処理モデルを提供するスレッドプールプラグインが含まれています。これは、多数のクライアント接続のステートメント実行スレッドを効率的に管理して、サーバーのパフォーマンスを向上させるスレッドプールを実装します。 [セクション 5.6.3 「MySQL Enterprise Thread Pool」](#) を参照してください。

クライアント接続を処理するスレッドをサーバーがどのように管理するかを制御し、モニターするには、いくつかのシステム変数とステータス変数が関連します。([セクション 5.1.8 「サーバーシステム変数」](#) および [セクション 5.1.10 「サーバーステータス変数」](#) を参照してください。)

- `thread_cache_size` システム変数はスレッドキャッシュサイズを決定します。デフォルトでは、サーバーは起動時に値のサイズを自動設定しますが、このデフォルトをオーバーライドするように明示的に設定できます。値

0 を指定すると、キャッシュが無効になり、新しい接続ごとにスレッドが設定され、接続の終了時に破棄されます。N の非アクティブな接続スレッドをキャッシュできるようにするには、サーバーの起動時または実行時に `thread_cache_size` を N に設定します。関連付けられていたクライアント接続が終了すると、接続スレッドは非アクティブになります。

- キャッシュ内のスレッド数と、スレッドをキャッシュから取得できなかったために作成されたスレッド数を監視するには、`Threads_cached` および `Threads_created` のステータス変数を確認します。
- スレッドスタックが小さすぎると、サーバーが処理できる SQL ステートメントの複雑さ、ストアドプロシージャの再帰深度およびその他のメモリー消費アクションが制限されます。スレッドごとに N バイトのスタックサイズを設定するには、`thread_stack` を N に設定してサーバーを起動します。

接続ポリューム管理

サーバーが同時に接続できるクライアントの最大数を制御するには、サーバーの起動時または実行時に `max_connections` システム変数を設定します。より多くのクライアントが同時に接続を試みる場合、サーバーが処理するように構成されているため、`max_connections` を増やす必要があります ([セクション B.3.2.5 「接続が多すぎます」](#) を参照)。 `max_connections` の制限に達したためにサーバーが接続を拒否した場合、サーバーは `Connection_errors_max_connections` ステータス変数を増分します。

`mysqld` では、実際には `max_connections` + 1 クライアント接続が許可されます。追加の接続は、`CONNECTION_ADMIN` 権限 (または非推奨の `SUPER` 権限) を持つアカウントで使用するために予約されています。通常のユーザー (必要ないユーザー) ではなく管理者に権限を付与することで、管理者はサーバーに接続し、権限のないクライアントの最大数が接続されている場合でも `SHOW PROCESLIST` を使用して問題を診断できます。 [セクション 13.7.7.29 「SHOW PROCESLIST ステートメント」](#) を参照してください。

MySQL 8.0.14 の時点では、専用の IP アドレスおよびポートを使用して設定できる管理ネットワークインタフェース上の管理接続もサーバーで許可されます。 [セクション 5.1.12.2 「管理接続管理」](#) を参照してください。

Group Replication プラグインは、内部セッションを使用して MySQL Server と対話し、SQL API 操作を実行します。MySQL 8.0.18 へのリリースでは、これらのセッションは `max_connections` サーバースystem変数で指定されたクライアント接続制限にカウントされます。これらのリリースでは、グループレプリケーションの起動時または操作の実行試行時にサーバーが `max_connections` 制限に達した場合、操作は失敗し、グループレプリケーションまたはサーバー自体が停止する可能性があります。MySQL 8.0.19 からは、グループレプリケーションの内部セッションはクライアント接続とは別に処理されるため、`max_connections` の制限にはカウントされず、サーバーがこの制限に達しても拒否されません。

MySQL でサポートされるクライアント接続の最大数 (つまり、`max_connections` を設定できる最大値) は、いくつかの要因によって異なります:

- 指定されたプラットフォーム上のスレッドライブラリの品質。
- 使用可能な RAM の量。
- RAM の量は接続ごとに使用されます。
- 各接続のワークロード。
- 目的のレスポンス時間。
- 使用可能なファイル記述子の数。

Linux または Solaris では、使用可能な RAM が多数あり、それぞれのワークロードが少ない場合、またはレスポンス時間のターゲットが要求されない場合、少なくとも 500 から 1000 の同時接続と 10,000 の接続を定期的にサポートできる必要があります。

`max_connections` 値を増やすと、`mysqld` に必要なファイル記述子の数が増えます。必要な数のディスクリプタが利用できない場合、サーバーは `max_connections` の値を削減します。ファイル記述子の制限に関するコメントについては、[セクション 8.4.3.1 「MySQL でのテーブルのオープンとクローズの方法」](#) を参照してください。

`open_files_limit` システム変数の増加が必要になる場合がありますが、これには、MySQL で使用できるファイル記述子の数に関するオペレーティングシステム制限の引上げが必要になることもあります。制限値を増やすことができるか

どうか、およびその実行方法については、使用するオペレーティングシステムのドキュメントを参照してください。[セクションB.3.2.16「ファイルが見つからず同様のエラーが発生しました」](#)も参照してください。

5.1.12.2 管理接続管理

[接続ボリューム管理](#)で説明されているように、通常の接続に使用されるインタフェースで `max_connections` 接続がすでに確立されている場合でも管理操作を実行できるように、MySQL サーバーは `CONNECTION_ADMIN` 権限 (または非推奨の `SUPER` 権限) を持つユーザーに対して単一の管理接続を許可します。

また、MySQL 8.0.14 の時点では、次の項で説明するように、サーバーは管理接続専用の TCP/IP ポートを許可します。

- [管理インタフェースの特性](#)
- [暗号化された接続に対する管理インタフェースのサポート](#)

管理インタフェースの特性

管理接続インタフェースには、次の特性があります:

- サーバーは、`admin_address` システム変数が起動時にその IP アドレスを示すように設定されている場合にのみ、インタフェースを有効にします。`admin_address` が設定されていない場合、サーバーは管理インタフェースを維持しません。
- `admin_port` システム変数は、インタフェースの TCP/IP ポート番号 (デフォルトは 33062) を指定します。
- 管理接続の数に制限はありません。
- 接続は、`SERVICE_CONNECTION_ADMIN` 権限を持つユーザーにのみ許可されます。
- `create_admin_listener_thread` システム変数を使用すると、DBA は起動時に管理インタフェースに独自のスレッドがあるかどうかを選択できます。デフォルトは `OFF` です。つまり、メインインタフェース上の通常の接続のマネージャスレッドは、管理インタフェースの接続も処理します。

サーバー `my.cnf` ファイルの次の行は、ループバックインタフェースの管理インタフェースを有効にし、ポート番号 33064 (つまり、デフォルトとは異なるポート) を使用するように構成します:

```
[mysqld]
admin_address=127.0.0.1
admin_port=33064
```

MySQL クライアントプログラムは、適切な接続パラメータを指定して、メインインタフェースまたは管理インタフェースに接続します。ローカルホストで実行されているサーバーが、メインインタフェースおよび管理インタフェースに 3306 および 33062 のデフォルトの TCP/IP ポート番号を使用している場合、次のコマンドはこれらのインタフェースに接続します:

```
mysql --protocol=TCP --port=3306
mysql --protocol=TCP --port=33062
```

暗号化された接続に対する管理インタフェースのサポート

MySQL 8.0.21 より前の管理インタフェースでは、メインインタフェースに適用される `connection-encryption` 構成を使用した暗号化された接続がサポートされていました。MySQL 8.0.21 の時点では、管理インタフェースには暗号化された接続用の独自の構成パラメータがあります。これらはメインインタフェースパラメータに対応していますが、管理インタフェースの暗号化された接続を独立して構成できます:

- `--admin-ssl` オプションは `--ssl` オプションと似ていますが、メインインタフェースではなく管理インタフェースの暗号化された接続を有効または無効にします。
- `admin_tls_xxx` および `admin_ssl_xxx` システム変数は、`tls_xxx` および `ssl_xxx` システム変数と似ていますが、メインインタフェースではなく管理インタフェースの TLS コンテキストを構成します。

`connection-encryption` サポートの構成に関する一般情報は、[セクション6.3.1「暗号化接続を使用するための MySQL の構成」](#)を参照してください。この説明はメイン接続インタフェース用に記述されていますが、パラメータ名は管理接続インタフェース用に似ています。次のノートでは、管理インタフェースに固有の情報を提供します。

管理インターフェースの TLS 構成は、次の規則に従います:

- `--admin-ssl` が有効な場合 (デフォルト)、管理インターフェースは暗号化された接続をサポートします。インターフェース上の接続の場合、適用可能な TLS コンテキストは、デフォルト以外の管理 TLS パラメータが構成されているかどうかによって異なります:
- すべての管理 TLS パラメータにデフォルト値がある場合、管理インターフェースはメインインターフェースと同じ TLS コンテキストを使用します。
- 管理 TLS パラメータにデフォルト以外の値がある場合、管理インターフェースは独自のパラメータで定義された TLS コンテキストを使用します。(これは、`admin_tls_xxx` または `admin_ssl_xxx` システム変数がデフォルトとは異なる値に設定されている場合です。) これらのパラメータから有効な TLS コンテキストを作成できない場合、管理インターフェースはメインインターフェース TLS コンテキストにフォールバックします。
- `--admin-ssl` が無効になっている場合、管理インターフェースへの暗号化された接続は無効になります。(これは、`--admin-ssl` の無効化が優先されるため、管理 TLS パラメータにデフォルト以外の値がある場合でも当てはまりません。)

例:

- サーバー `my.cnf` ファイル内のこの構成は管理インターフェースを有効にしますが、そのインターフェースに固有の TLS パラメータは設定しません:

```
[mysqld]
admin_address=127.0.0.1
```

その結果、管理インターフェースは暗号化された接続をサポートしますが (管理インターフェースが有効な場合、暗号化はデフォルトでサポートされるため)、メインインターフェース TLS コンテキストを使用します。クライアントは、管理インターフェースに接続するときに、メインインターフェースでの通常の接続と同じ証明書およびキーファイルを使用する必要があります。次に例を示します (単一行にコマンドを入力します):

```
mysql --protocol=TCP --port=33062
--ssl-ca=ca.pem
--ssl-cert=client-cert.pem
--ssl-key=client-key.pem
```

- このサーバー構成は、管理インターフェースを有効にし、そのインターフェースに固有の TLS 証明書およびキーファイルパラメータを設定します:

```
[mysqld]
admin_address=127.0.0.1
admin_ssl_ca=admin-ca.pem
admin_ssl_cert=admin-server-cert.pem
admin_ssl_key=admin-server-key.pem
```

その結果、管理インターフェースは独自の TLS コンテキストを使用して暗号化された接続をサポートします。クライアントは、管理インターフェースに接続するときに、そのインターフェースに固有の証明書およびキーファイルを使用する必要があります。次に例を示します (単一行にコマンドを入力します):

```
mysql --protocol=TCP --port=33062
--ssl-ca=admin-ca.pem
--ssl-cert=admin-client-cert.pem
--ssl-key=admin-client-key.pem
```

- このサーバー構成は、管理インターフェースを有効にしますが、暗号化された接続を無効にします:

```
[mysqld]
admin-ssl=OFF
admin_address=127.0.0.1
```

この場合、構成で `admin_ssl_ca` などの管理 TLS パラメータも設定すると、`admin-ssl=OFF` が優先されるため、これらのパラメータ設定は無効になります。

5.1.12.3 DNS ルックアップとホストキャッシュ

MySQL サーバーは、クライアントに関する情報を含むインメモリーホストキャッシュを保持: IP アドレス、ホスト名およびエラー情報。パフォーマンススキーマ `host_cache` テーブルは、`SELECT` ステートメントを使用して検査でき

るように、ホストキャッシュの内容を公開します。これは、接続の問題の原因の診断に役立つことがあります。 [セクション27.12.19.5「host_cache テーブル」](#)を参照してください。

次のセクションでは、ホストキャッシュの動作、およびキャッシュの構成方法やモニター方法などのその他のトピックについて説明します。

- [ホストキャッシュ操作](#)
- [ホストキャッシュの構成](#)
- [ホストキャッシュの監視](#)
- [ホストキャッシュのフラッシュ](#)
- [ブロックされたホストの処理](#)

ホストキャッシュ操作

サーバーは、localhost 以外の TCP 接続にのみホストキャッシュを使用します。ループバックインタフェースアドレス (127.0.0.1 や ::1 など) を使用して確立された TCP 接続、または Unix ソケットファイル、名前付きパイプまたは共有メモリーを使用して確立された接続には、キャッシュは使用されません。

サーバーはいくつかの目的でホストキャッシュを使用します。

- IP からホスト名へのルックアップの結果をキャッシュすることで、サーバーはクライアント接続ごとにドメインネームシステム (DNS) ルックアップを行わないようにします。代わりに、特定のホストに対して、そのホストからの最初の接続でのみルックアップを実行する必要があります。
- キャッシュには、クライアント接続プロセス中に発生したエラーに関する情報が含まれます。一部のエラーは「ブロッキング」とみなされます。成功した接続がない特定のホストから、これらの多くが連続して発生している場合、サーバーはそのホストからのその後の接続をブロックします。max_connect_errors システム変数は、ブロックが発生するまでの連続エラーの許容数を決定します。

適用可能な新しいクライアント接続ごとに、サーバーはクライアント IP アドレスを使用して、クライアントホスト名がホストキャッシュ内にあるかどうかを確認します。その場合、ホストがブロックされているかどうかに応じて、サーバーは接続リクエストの処理を拒否または続行します。ホストがキャッシュ内がない場合、サーバーはホスト名の解決を試みます。まず、それは IP アドレスをホスト名に解決し、そのホスト名を再度 IP アドレスに解決します。次に、その結果と元の IP アドレスを比較して、それらが同じであることを確認します。サーバーはこの操作の結果に関する情報をホストキャッシュに格納します。キャッシュがいっぱいである場合、直近で使用されていないエントリが破棄されます。

サーバーは、gethostbyaddr() および gethostbyname() システムコールを使用してホスト名解決を実行します。

サーバーは次のようにホストキャッシュ内のエントリを処理します。

1. 最初の TCP クライアント接続が特定の IP アドレスからサーバーに到達すると、クライアント IP、ホスト名およびクライアント検索検証フラグを記録するための新しいキャッシュエントリが作成されます。最初に、ホスト名が NULL に設定され、フラグは false になります。このエントリは、同じ発信元 IP からの後続のクライアント TCP 接続にも使用されます。
2. クライアント IP エントリの検証フラグが false の場合、サーバーは IP-to-host name-to-IP DNS 解決を試みます。それが成功した場合、ホスト名が解決されたホスト名で更新され、検証フラグが true に設定されます。解決が成功しない場合、とられるアクションは、エラーが永続的か一時的かによって異なります。永続的なエラーの場合、ホスト名は NULL のままになり、検証フラグは true に設定されます。一時的なエラーの場合、ホスト名と検証フラグは変更されないままになります。(この場合、クライアントが次回この IP から接続したときに、別の DNS 解決が試行されます。)
3. 特定の IP アドレスからの着信クライアント接続の処理中にエラーが発生した場合、サーバーはその IP のエントリ内の対応するエラーカウンタを更新します。記録されたエラーの説明については、[セクション 27.12.19.5「host_cache テーブル」](#)を参照してください。

ブロックされたホストのブロックを解除するには、ホストキャッシュをフラッシュします。[ブロックされたホストの処理](#)を参照してください。

ほかのホストからのアクティビティーが発生した場合でも、ブロックされたホストがホストキャッシュをフラッシュせずにブロック解除される可能性があります:

- キャッシュに存在しないクライアント IP から接続が到着したときにキャッシュがいっぱいになった場合、サーバーは新しいエントリ用の領域を確保するために、最も最近使用されていないキャッシュエントリを破棄します。
- 破棄されたエントリがブロックされたホストのものである場合、そのホストのブロックが解除されます。

一部の接続エラーは TCP 接続に関連付けられないか、接続プロセスのきわめて早期に (IP アドレスも判明する前に) 発生するか、または特定の IP アドレスに固有ではありません (メモリー不足の状況など)。これらのエラーについては、[Connection_errors_xxx](#) ステータス変数をチェックしてください ([セクション5.1.10「サーバーステータス変数」](#)を参照してください)。

ホストキャッシュの構成

ホストキャッシュはデフォルトで有効になっています。 `host_cache_size` システム変数は、キャッシュの内容を公開するパフォーマンススキーマ `host_cache` テーブルのサイズだけでなく、そのサイズも制御します。キャッシュサイズはサーバーの起動時に設定でき、実行時に変更できます。たとえば、起動時にサイズを 100 に設定するには、サーバー `my.cnf` ファイルに次の行を入力します:

```
[mysqld]
host_cache_size=200
```

実行時にサイズを 300 に変更するには、次のようにします:

```
SET GLOBAL host_cache_size=300;
```

サーバーの起動時または実行時に `host_cache_size` を 0 に設定すると、ホストキャッシュが無効になります。キャッシュを無効にすると、サーバーはクライアントが接続するたびに DNS ルックアップを実行します。

実行時にキャッシュサイズを変更すると、ホストキャッシュをクリアし、`host_cache` テーブルを切り捨て、ブロックされたホストをブロック解除する暗黙的なホストキャッシュフラッシュ操作が発生します。[ホストキャッシュのフラッシュ](#)を参照してください。

`--skip-host-cache` オプションの使用は、`host_cache_size` システム変数を 0 に設定するのと似ていますが、`host_cache_size` は、サーバーの起動時だけでなく、実行時にホストキャッシュのサイズ変更、有効化、および無効化にも使用できるため、より柔軟です。`--skip-host-cache` を使用してサーバーを起動しても、`host_cache_size` の値に対する実行時の変更は妨げられませんが、このような変更は効果がなく、`host_cache_size` が 0 より大きい値に設定されていてもキャッシュは再度有効になりません。

DNS ホスト名検索を無効にするには、`skip_name_resolve` システム変数を有効にしてサーバーを起動します。この場合、サーバーは IP アドレスのみを使用し、ホスト名を使用しないで、接続しているホストを MySQL 付与テーブル内の行と照合します。IP アドレスを使用してそれらのテーブルに指定されたアカウントのみを使用できます。(クライアント IP アドレスを指定するアカウントが存在しない場合、クライアントは接続できない可能性があります。)

非常に遅い DNS および多数のホストがある場合は、`skip_name_resolve` で DNS ルックアップを無効にするか、`host_cache_size` の値を増やしてホストキャッシュを大きくすることで、パフォーマンスを向上させることができます。

TCP/IP 接続を完全に禁止するには、`skip_networking` システム変数を有効にしてサーバーを起動します。

ホストのブロックが発生する前に、連続する接続エラーの許容数を調整するには、`max_connect_errors` システム変数を設定します。たとえば、起動時に値を設定するには、サーバーの `my.cnf` ファイルに次の行を入力します:

```
[mysqld]
max_connect_errors=10000
```

実行時に値を変更するには、次のようにします:

```
SET GLOBAL max_connect_errors=10000;
```

ホストキャッシュの監視

パフォーマンススキーマ `host_cache` テーブルは、ホストキャッシュの内容を公開します。このテーブルは、接続の問題の原因の診断に役立つ可能性のある `SELECT` ステートメントを使用して調べることができます。このテーブルの詳細は、[セクション27.12.19.5「host_cache テーブル」](#)を参照してください。

ホストキャッシュのフラッシュ

次の条件下では、ホストキャッシュをフラッシュすることをお勧めします:

- クライアントホストの一部が IP アドレスを変更します。
- 正当なホストからの接続に対して `Host 'host_name' is blocked` というエラーメッセージが表示されます。([ブロックされたホストの処理](#) を参照してください。)

ホストキャッシュをフラッシュすると、次の効果があります:

- インメモリホストキャッシュをクリアします。
- キャッシュの内容を公開するパフォーマンススキーマ `host_cache` テーブルからすべての行が削除されます。
- ブロックされたホストのブロックを解除します。これにより、これらのホストからの以降の接続試行が可能になります。

ホストキャッシュをフラッシュするには、次のいずれかの方法を使用します:

- `host_cache_size` システム変数の値を変更します。これには、`SYSTEM_VARIABLES_ADMIN` 権限 (または非推奨の `SUPER` 権限) が必要です。
- パフォーマンススキーマ `host_cache` テーブルを切り捨てる `TRUNCATE TABLE` ステートメントを実行します。これには、テーブルに対する `DROP` 権限が必要です。
- `FLUSH HOSTS` ステートメントを実行します。これには、`RELOAD` 権限が必須です。
- `mysqladmin flush-hosts` コマンドを実行します。これには、パフォーマンススキーマ `host_cache` テーブルに対する `DROP` 権限または `RELOAD` 権限が必要です。

ブロックされたホストの処理

サーバーはホストキャッシュを使用して、クライアント接続プロセス中に発生したエラーを追跡します。次のエラーが発生する場合は、`mysqld` が途中で中断された多数の接続要求を特定のホストから受け取ったことを意味します。

```
Host 'host_name' is blocked because of many connection errors.  
Unblock with 'mysqladmin flush-hosts'
```

`max_connect_errors` システム変数の値によって、ホストをブロックする前にサーバーが許可する連続した中断された接続リクエストの数が決まります。正常に接続されずに `max_connect_errors` がリクエストに失敗した後、サーバーはなんらかの問題がある (たとえば、だれかがブレークしようとしている) とみなし、それ以降の接続リクエストからホストをブロックします。

ブロックされたホストのブロックを解除するには、ホストキャッシュをフラッシュします。 [ホストキャッシュのフラッシュ](#) を参照してください。

または、エラーメッセージが表示されないようにするには、 [ホストキャッシュの構成](#) の説明に従って `max_connect_errors` を設定します。 `max_connect_errors` のデフォルト値は 100 です。 `max_connect_errors` を大きい値に増やすと、ホストがしきい値に達してブロックされる可能性が低くなります。ただし、 `Host 'host_name' is blocked` エラーメッセージが表示された場合は、まずブロックされたホストからの TCP/IP 接続に問題がないことを確認します。ネットワークに問題がある場合、 `max_connect_errors` の値を増やすことは適切ではありません。

5.1.13 IPv6 サポート

MySQL での IPv6 のサポートには次の機能があります。

- MySQL Server は、IPv6 を介して接続するクライアントからの TCP/IP 接続を受け入れることができます。たとえば、次のコマンドは、ローカルホスト上の MySQL Server に IPv6 を介して接続します。

```
shell> mysql -h ::1
```

この機能を使用するには、2 つのことが満たされている必要があります。

- システムが IPv6 をサポートするように構成されている必要があります。 [セクション5.1.13.1「IPv6 用のシステムサポートの確認」](#) を参照してください。
- デフォルトの MySQL サーバー構成では、IPv4 接続に加えて IPv6 接続が許可されます。デフォルトの構成を変更するには、`bind_address` システム変数を適切な値に設定してサーバーを起動します。 [セクション5.1.8「サーバーシステム変数」](#) を参照してください。
- MySQL アカウント名には、IPv6 を介してサーバーに接続するクライアントの権限を DBA が指定できるようにするために、IPv6 アドレスが許可されます。 [セクション6.2.4「アカウント名の指定」](#) を参照してください。IPv6 アドレスは、`CREATE USER`、`GRANT`、`REVOKE` などのステートメント内のアカウント名に指定できます。例:

```
mysql> CREATE USER 'bill'@ ':::1' IDENTIFIED BY 'secret';
mysql> GRANT SELECT ON mydb.* TO 'bill'@ ':::1';
```

- IPv6 関数は、文字列と内部形式の IPv6 アドレス形式の間の変換が可能で、値が有効な IPv6 アドレスを表現しているかどうかを検査できます。たとえば、`INET6_ATON()` および `INET6_NTOA()` は `INET_ATON()` および `INET_NTOA()` に類似していますが、IPv4 アドレスに加えて IPv6 アドレスも処理します。 [セクション12.24「その他の関数」](#) を参照してください。
- MySQL 8.0.14 から、グループレプリケーショングループメンバーは、グループ内の通信に IPv6 アドレスを使用できます。グループには、IPv6 を使用するメンバーと IPv4 を使用するメンバーを混在させることができます。 [セクション18.4.5「IPv6 および IPv6 と IPv4 の混合グループのサポート」](#) を参照してください。

以降のセクションでは、クライアントが IPv6 を介してサーバーに接続できるようにするために、MySQL をセットアップする方法について説明します。

5.1.13.1 IPv6 用のシステムサポートの確認

MySQL Server が IPv6 接続を受け入れるには、サーバーホスト上のオペレーティングシステムが IPv6 をサポートしている必要があります。このことが当てはまるかどうかを判別する簡単なテストとして、次のコマンドを試してみてください。

```
shell> ping6 ::1
16 bytes from ::1, icmp_seq=0 hlim=64 time=0.171 ms
16 bytes from ::1, icmp_seq=1 hlim=64 time=0.077 ms
...
```

システムのネットワークインタフェースの説明を生成するには、`ifconfig -a` を呼び出して、出力に IPv6 がないか探します。

ホストが IPv6 をサポートしない場合、システムのドキュメントを調べて IPv6 を有効にするための手順を確認します。既存のネットワークインタフェースの再構成後 IPv6 アドレスの追加のみが必要な場合もあります。また、IPv6 オプションを有効にしてカーネルを再構築するなどの大がかりな変更が必要な場合もあります。

IPv6 さまざまなプラットフォームで設定するとき、次のリンクが役立つことがあります。

- [Windows](#)
- [Gentoo Linux](#)
- [Ubuntu Linux](#)
- [Linux \(汎用\)](#)
- [macOS](#)

5.1.13.2 IPv6 接続を許可するための MySQL Server の構成

MySQL サーバーは、1 つ以上のネットワークソケットで TCP/IP 接続をリスニングします。各ソケットは 1 つのアドレスにバインドされますが、1 つのアドレスを複数のネットワークインタフェースにマップできます。

サーバー起動時に `bind_address` システム変数を設定して、サーバーインスタンスが受け入れる TCP/IP 接続を指定します。MySQL 8.0.13 では、IPv6 アドレス、IPv4 アドレス、および IPv6 アドレスまたは IPv4 アドレスに解決されるホスト名の任意の組合せなど、このオプションに複数の値を指定できます。または、複数のネットワークインタ

フェースでのリスニングを許可するワイルドカードアドレス形式のいずれかを指定できます。* の値 (デフォルト) または:: の値を指定すると、すべてのサーバーホストの IPv4 および IPv6 インタフェースで IPv4 と IPv6 の両方の接続が許可されます。詳細は、[セクション5.1.8「サーバーシステム変数」](#) の `bind_address` の説明を参照してください。

5.1.13.3 IPv6 ローカルホストアドレスを使用した接続

次の手順では、::1 のローカルホストアドレスを使用してローカルサーバーに接続するクライアントによる IPv6 接続を許可するために、MySQL を構成する方法を示します。ここに示す手順は、システムが IPv6 をサポートしていることを想定しています。

1. 適切な `bind_address` 設定を使用して MySQL サーバーを起動し、IPv6 接続の受入れを許可します。たとえば、次の行をサーバーオプションファイルに入れて、サーバーを再起動します。

```
[mysqld]
bind_address = *
```

`bind_address` の値として *(または::) を指定すると、すべてのサーバーホストの IPv4 および IPv6 インタフェースで IPv4 と IPv6 の両方の接続が許可されます。サーバーを特定のアドレスリストにバインドする場合は、`bind_address` にカンマ区切りの値リストを指定することで、MySQL 8.0.13 の時点でこれを実行できます。この例では、IPv4 と IPv6 の両方のローカルホストアドレスを指定します:

```
[mysqld]
bind_address = 127.0.0.1:::1
```

詳細は、[セクション5.1.8「サーバーシステム変数」](#) の `bind_address` の説明を参照してください。

2. 管理者としてサーバーに接続し、::1 ローカル IPv6 ホストアドレスから接続できるローカルユーザーのアカウントを作成します:

```
mysql> CREATE USER 'ipv6user'@ ':::1' IDENTIFIED BY 'ipv6pass';
```

アカウント名で許可される IPv6 アドレスの構文については、[セクション6.2.4「アカウント名の指定」](#) を参照してください。CREATE USER ステートメントに加えて、特定の権限をアカウントに付与する GRANT ステートメントを発行できます。ただし、この手順の残りのステップには不要です。

3. mysql クライアントを呼び出し、新しいアカウントを使用するサーバーに接続します。

```
shell> mysql -h ::1 -u ipv6user -pipv6pass
```

4. 接続情報を表示する単純なステートメントを試してみます。

```
mysql> STATUS
...
Connection: ::1 via TCP/IP
...

mysql> SELECT CURRENT_USER(), @@bind_address;
+-----+-----+
| CURRENT_USER() | @@bind_address |
+-----+-----+
| ipv6user@::1 | :::1           |
+-----+-----+
```

5.1.13.4 IPv6 非ローカルホストアドレスを使用した接続

次の手順では、リモートクライアントによる IPv6 接続を許可するために MySQL を構成する方法を示します。これは前に示したローカルクライアントについての手順と似ていますが、サーバーとクライアントホストが別個であり、それぞれがローカル以外の独自の IPv6 アドレスを持ちます。この例では次のアドレスが使用されます。

```
Server host: 2001:db8:0:f101::1
Client host: 2001:db8:0:f101::2
```

これらのアドレスは、文書化目的で IANA によって推奨されるルーティングできないアドレス範囲から選択され、ローカルネットワークでの試験向けには十分です。ローカルネットワーク外部のクライアントから IPv6 接続を受け入れるには、サーバーホストが公開アドレスを持つ必要があります。ネットワークプロバイダによって IPv6 アドレスが割り当てられている場合、それを使用できます。そうでない場合、アドレスを取得するための別の方法は、IPv6 ブローカを使用する方法で、[セクション5.1.13.5「ブローカからの IPv6 アドレスの入手」](#) を参照してください。

1. 適切な `bind_address` 設定を使用して MySQL サーバーを起動し、IPv6 接続の受入れを許可します。たとえば、次の行をサーバーオプションファイルに入れて、サーバーを再起動します。

```
[mysqld]
bind_address = *
```

`bind_address` の値として *(または::) を指定すると、すべてのサーバーホストの IPv4 および IPv6 インタフェースで IPv4 と IPv6 の両方の接続が許可されます。サーバーを特定のアドレスリストにバインドする場合は、`bind_address` にカンマ区切りの値リストを指定することで、MySQL 8.0.13 の時点でこれを実行できます。この例では、IPv4 アドレスと必要なサーバーホストの IPv6 アドレスを指定します:

```
[mysqld]
bind_address = 198.51.100.20,2001:db8:0:f101::1
```

詳細は、[セクション5.1.8「サーバーシステム変数」](#) の `bind_address` の説明を参照してください。

2. サーバーホスト (`2001:db8:0:f101::1`) で、クライアントホスト (`2001:db8:0:f101::2`) から接続できるユーザーのアカウントを作成します:

```
mysql> CREATE USER 'remoteipv6user'@'2001:db8:0:f101::2' IDENTIFIED BY 'remoteipv6pass';
```

3. クライアントホスト (`2001:db8:0:f101::2`) 上で、`mysql` クライアントを呼び出し、新しいアカウントを使用するサーバーに接続します。

```
shell> mysql -h 2001:db8:0:f101::1 -u remoteipv6user -premoveipv6pass
```

4. 接続情報を表示する単純なステートメントを試してみます。

```
mysql> STATUS
...
Connection: 2001:db8:0:f101::1 via TCP/IP
...

mysql> SELECT CURRENT_USER(), @@bind_address;
+-----+-----+
| CURRENT_USER()          | @@bind_address |
+-----+-----+
| remoteipv6user@2001:db8:0:f101::2 | ::             |
+-----+-----+
```

5.1.13.5 ブローカからの IPv6 アドレスの入手

システムが IPv6 を介してローカルネットワークの外部と通信できるようにするための公開 IPv6 アドレスがない場合、IPv6 ブローカから入手できます。[Wikipedia IPv6 Tunnel Broker page](#) には、いくつかのブローカとその特徴 (静的アドレスおよびサポートされるルーティングプロトコルを提供するかどうかなど) をリストします。

ブローカ提供の IPv6 アドレスを使用するようにサーバーホストを構成した後、適切な `bind_address` 設定を使用して MySQL サーバーを起動し、サーバーが IPv6 接続を受け入れることを許可します。*(または::) を `bind_address` 値として指定するか、ブローカによって提供される特定の IPv6 アドレスにサーバーをバインドできます。詳細は、[セクション5.1.8「サーバーシステム変数」](#) の `bind_address` の説明を参照してください。

ブローカが動的アドレスを割り当てると、次回ブローカに接続したときにシステムに指定されたアドレスが変更される可能性があることに注意してください。その場合、ユーザーが作成した、元のアドレスを指定するアカウントが無効になります。特定のアドレスにバインドするが、このアドレス変更の問題を回避するために、ブローカに静的 IPv6 アドレスを配置できる場合があります。

次の例は、Freenet6 をブローカとして使用し、[gogonet](#) IPv6 クライアントパッケージを Gentoo Linux 上で使用方法を示します。

1. 次の URL にアクセスしてサインアップし、Freenet6 でアカウントを作成します:

```
http://gogonet.gogo6.com
```

2. アカウントを作成したあと、この URL に移動してサインインし、IPv6 ブローカ用のユーザー ID およびパスワードを作成します。

```
http://gogonet.gogo6.com/page/freenet6-registration
```


3. `root` として、`gogoc` をインストールします。

```
shell> emerge gogoc
```

4. `/etc/gogoc/gogoc.conf` を編集して `userid` および `password` の値を設定します。例:

```
userid=gogouser  
passwd=gogopass
```

5. `gogoc` を開始します。

```
shell> /etc/init.d/gogoc start
```

システムが起動するたびに `gogoc` を開始するには、次のコマンドを実行します。

```
shell> rc-update add gogoc default
```

6. `ping6` を使用してホストに `ping` を実行します。

```
shell> ping6 ipv6.google.com
```

7. IPv6 アドレスを表示するには、次のようにします。

```
shell> ifconfig tun
```

5.1.14 ネットワークネームスペースのサポート

ネットワークネームスペースは、ホストシステムからのネットワークスタックの論理コピーです。ネットワークネームスペースは、コンテナまたは仮想環境の設定に役立ちます。各ネームスペースには、独自の IP アドレス、ネットワークインタフェース、ルーティングテーブルなどがあります。デフォルトまたはグローバル名前空間は、ホストシステムの物理インタフェースが存在する名前空間です。

ネームスペース固有のアドレス空間は、MySQL 接続がネームスペース間をまたがる場合に問題になる可能性があります。たとえば、コンテナまたは仮想ネットワークで実行されている MySQL インスタンスのネットワークアドレス空間は、ホストマシンのアドレス空間とは異なる場合があります。これにより、同じマシン上で実行されているクライアントとサーバーであっても、MySQL サーバーに表示されるネームスペース内のアドレスからのクライアント接続などの現象が発生する可能性があります。両方のプロセスが IP アドレスが `203.0.113.10` のホストで実行されているが、異なるネームスペースを使用しているとします。接続によって次のような結果が生成される場合があります:

```
shell> mysql --user=admin --host=203.0.113.10 --protocol=tcp  
  
mysql> SELECT USER();  
+-----+  
| USER() |  
+-----+  
| admin@198.51.100.2 |  
+-----+
```

この場合、予想される `USER()` 値は `admin@203.0.113.10` です。このような動作により、接続元のアドレスが表示されない場合に、アカウント権限を正しく割り当てるのが困難になる可能性があります。

この問題に対処するために、MySQL では、TCP/IP 接続に使用するネットワークネームスペースを指定できるため、接続の両方のエンドポイントが合意した共通アドレス空間を使用します。

MySQL 8.0.22 以上では、それらを実装するプラットフォームでネットワークネームスペースがサポートされます。MySQL 内のサポートは、次のものに適用されます:

- MySQL サーバー、`mysqld`。
- X プラグイン。
- `mysql` クライアントおよび `mysqlxtest` テストスイートクライアント。(他のクライアントはサポートされていません。これらは、接続先のサーバーのネットワークネームスペース内から呼び出す必要があります。)
- 通常のレプリケーション。(グループレプリケーションはサポートされていません。)

次の各セクションでは、MySQL でネットワーク名前空間を使用する方法について説明します:

- [ホストシステムの前提条件](#)
- [MySQL 構成](#)
- [ネットワーク名前空間のモニタリング](#)

ホストシステムの前提条件

MySQL でネットワーク名前空間のサポートを使用する前に、次のホストシステムの前提条件を満たす必要があります:

- ホストオペレーティングシステムは、ネットワーク名前空間をサポートする必要があります。(たとえば、Linux。)
- MySQL で使用されるネットワーク名前空間は、最初にホストシステムに作成する必要があります。
- ネットワーク名前空間をサポートするには、システム管理者がホスト名解決を構成する必要があります。

注記

既知の制限は、MySQL 内では、ネットワーク名前空間固有のホストファイルで指定された名前に対してホスト名解決が機能しないことです。たとえば、`red` 名前空間のホスト名のアドレスが `/etc/netns/red/hosts` ファイルで指定されている場合、名前へのバインドはサーバー側とクライアント側の両方で失敗します。回避策は、ホスト名ではなく IP アドレスを使用することです。

- システム管理者は、ネットワーク名前空間 (`mysqld`, `mysql`, `mysqlxtest`) をサポートする MySQL バイナリに対して `CAP_SYS_ADMIN` オペレーティングシステム権限を有効にする必要があります。

重要

`CAP_SYS_ADMIN` の有効化は、名前空間の設定に加えてプロセスが他の権限アクションを実行できるようにするため、セキュリティに依存する操作です。効果の詳細は、<https://man7.org/linux/man-pages/man7/capabilities.7.html> を参照してください。

`CAP_SYS_ADMIN` はシステム管理者が明示的に有効にする必要があるため、MySQL バイナリではデフォルトでネットワーク名前空間のサポートは有効になっていません。システム管理者は、有効にする前に、`CAP_SYS_ADMIN` で MySQL プロセスを実行した場合のセキュリティの影響を評価する必要があります。

次の例の手順では、`red` および `blue` という名前のネットワーク名前空間を設定します。選択する名前は、ホストシステムのネットワークアドレスおよびインタフェースとは異なる場合があります。

ここに示すコマンドは、`root` オペレーティングシステムユーザーとして、または各コマンドの前に `sudo` を付けて起動します。たとえば、`root` 以外の場合に `ip` または `setcap` コマンドを起動するには、`sudo ip` または `sudo setcap` を使用します。

ネットワーク名前空間を構成するには、`ip` コマンドを使用します。一部の操作では、`ip` コマンドは特定のネットワーク名前空間 (すでに存在している必要があります) 内で実行する必要があります。このような場合は、次のようにコマンドを開始します:

```
ip netns exec namespace_name
```

たとえば、次のコマンドは `red` ネットワーク名前空間内で実行され、ループバックインタフェースを起動します:

```
ip netns exec red ip link set lo up
```

ネットワーク名前空間と独自のループバックインタフェース間のリンクとして使用される独自の仮想イーサネットデバイスを持つ、`red` および `blue` という名前のネットワーク名前空間を追加するには:

```
ip netns add red
```

```
ip link add veth-red type veth peer name vpeer-red
ip link set vpeer-red netns red
ip addr add 192.0.2.1/24 dev veth-red
ip link set veth-red up
ip netns exec red ip addr add 192.0.2.2/24 dev vpeer-red
ip netns exec red ip link set vpeer-red up
ip netns exec red ip link set lo up

ip netns add blue
ip link add veth-blue type veth peer name vpeer-blue
ip link set vpeer-blue netns blue
ip addr add 198.51.100.1/24 dev veth-blue
ip link set veth-blue up
ip netns exec blue ip addr add 198.51.100.2/24 dev vpeer-blue
ip netns exec blue ip link set vpeer-blue up
ip netns exec blue ip link set lo up

# if you want to enable inter-subnet routing...
sysctl net.ipv4.ip_forward=1
ip netns exec red ip route add default via 192.0.2.1
ip netns exec blue ip route add default via 198.51.100.1
```

ネームスペース間のリンクのダイアグラムは次のようになります:

```
red      global      blue
192.0.2.2 <=> 192.0.2.1
(vpeer-red) (veth-red)

          198.51.100.1 <=> 198.51.100.2
          (veth-blue) (vpeer-blue)
```

存在するネームスペースおよびリンクを確認するには:

```
ip netns list
ip link list
```

グローバルおよび名前付きネームスペースのルーティングテーブルを表示するには:

```
ip route show
ip netns exec red ip route show
ip netns exec blue ip route show
```

red および **blue** のリンクとネームスペースを削除するには:

```
ip link del veth-red
ip link del veth-blue

ip netns del red
ip netns del blue

sysctl net.ipv4.ip_forward=0
```

ネットワーク名前空間のサポートを含む MySQL バイナリが実際に名前空間を使用できるようにするには、それらに **CAP_SYS_ADMIN** 機能を付与する必要があります。次の **setcap** コマンドは、MySQL バイナリを含むディレクトリの場所を変更したことを前提としています (必要に応じてシステムのパス名を調整します):

```
cd /usr/local/mysql/bin
```

CAP_SYS_ADMIN 機能を適切なバイナリに付与するには:

```
setcap cap_sys_admin+ep ./mysqld
setcap cap_sys_admin+ep ./mysql
setcap cap_sys_admin+ep ./mysqlxtest
```

CAP_SYS_ADMIN の機能を確認するには:

```
shell> getcap ./mysqld ./mysql ./mysqlxtest
./mysqld = cap_sys_admin+ep
./mysql = cap_sys_admin+ep
./mysqlxtest = cap_sys_admin+ep
```

`CAP_SYS_ADMIN` 機能を削除するには:

```
setcap -r /mysqld  
setcap -r /mysql  
setcap -r /mysqlxtest
```

重要

以前に `setcap` を適用したバイナリを再インストールする場合は、`setcap` を再度使用する必要があります。たとえば、MySQL のインプレースアップグレードを実行する場合、`CAP_SYS_ADMIN` 機能の付与に再度失敗すると、ネームスペース関連の障害が発生します。名前付きネームスペースを持つアドレスにバインドしようとする、サーバーは次のエラーで失敗します:

```
[ERROR] [MY-013408] [Server] setns() failed with error 'Operation not permitted'
```

`--network-namespace` オプションを指定して起動されたクライアントは、次のように失敗します:

```
ERROR: Network namespace error: Operation not permitted
```

MySQL 構成

前述のホストシステムの前提条件が満たされている場合、MySQL では、接続のリスニング (インバウンド) 側のサーバー側ネームスペースおよび接続のアウトバウンド側のクライアント側ネームスペースを構成できます。

サーバー側では、`bind_address`、`admin_address` および `mysqlx_bind_address` システム変数に、着信接続をリスニングする特定の IP アドレスまたはホスト名に使用するネットワークネームスペースを指定するための拡張構文があります。アドレスのネームスペースを指定するには、スラッシュとネームスペース名を追加します。たとえば、サーバー `my.cnf` ファイルには次の行が含まれる場合があります:

```
[mysqld]  
bind_address = 127.0.1.1,192.0.2.2/red,198.51.100.2/blue  
admin_address = 102.0.2.2/red  
mysqlx_bind_address = 102.0.2.2/red
```

次のルールが適用されます:

- ネットワークネームスペースは、IP アドレスまたはホスト名に指定できます。
- ワイルドカード IP アドレスにはネットワークネームスペースを指定できません。
- 指定されたアドレスでは、ネットワーク名前空間はオプションです。指定する場合は、アドレスの直後に `/ns` 接尾辞として指定する必要があります。
- `/ns` 接尾辞のないアドレスは、ホストシステムのグローバルネームスペースを使用します。したがって、グローバルネームスペースがデフォルトです。
- `/ns` 接尾辞の付いたアドレスは、`ns` という名前のネームスペースを使用します。
- ホストシステムはネットワークネームスペースをサポートしている必要があり、各名前付きネームスペースは事前に設定されている必要があります。存在しないネームスペースに名前を付けると、エラーが発生します。
- `bind_address` および (MySQL 8.0.21) `mysqlx_bind_address` は、カンマ区切りの複数のアドレスのリストを受け入れます。変数値は、グローバルネームスペース、名前付きネームスペースまたはその組合せでアドレスを指定できません。

ネームスペースを使用しようとしてサーバーの起動中にエラーが発生した場合、サーバーは起動しません。プラグインの初期化中に `X` プラグイン でエラーが発生し、どのアドレスにもバインドできない場合、プラグインはその初期化シーケンスに失敗し、サーバーはそれをロードしません。

クライアント側では、ネットワークネームスペースを次のコンテキストで指定できます:

- `mysql` クライアントおよび `mysqlxtest` テストスイートクライアントの場合は、`--network-namespace` オプションを使用します。例:

```
mysql --host=192.0.2.2 --network-namespace=red
```

`--network-namespace` オプションを省略すると、接続ではデフォルト (グローバル) の名前空間が使用されません。

- レプリカサーバーからソースサーバーへのレプリケーション接続には、`CHANGE REPLICATION SOURCE TO` ステートメント (MySQL 8.0.23) または `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 より前) を使用し、`NETWORK_NAMESPACE` オプションを指定します。例:

```
CHANGE REPLICATION SOURCE TO
SOURCE_HOST = '192.0.2.2',
NETWORK_NAMESPACE = 'red';
```

`NETWORK_NAMESPACE` オプションを省略すると、レプリケーション接続はデフォルト (グローバル) 名前空間を使用します。

次の例では、グローバル、`red` および `blue` 名前空間で接続をリスニングする MySQL サーバーを設定し、`red` および `blue` 名前空間から接続するアカウントを構成する方法を示します。[ホストシステムの前提条件](#) に示すように、`red` および `blue` 名前空間がすでに作成されていることを前提としています。

- 複数の名前空間のアドレスをリスニングするようにサーバーを構成します。サーバー `my.cnf` ファイルに次の行を入力し、サーバーを起動します:

```
[mysqld]
bind_address = 127.0.1.1,192.0.2.2/red,198.51.100.2/blue
```

この値は、グローバル名前空間でループバックアドレス `127.0.0.1` をリスニングし、`red` 名前空間でアドレス `192.0.2.2` をリスニングし、`blue` 名前空間でアドレス `198.51.100.2` をリスニングするようサーバーに指示します。

- グローバル名前空間のサーバーに接続し、各名前付き名前空間のアドレス空間のアドレスから接続する権限を持つアカウントを作成します:

```
shell> mysql -u root -h 127.0.0.1 -p
Enter password: root_password

mysql> CREATE USER 'red_user'@'192.0.2.2'
IDENTIFIED BY 'red_user_password';
mysql> CREATE USER 'blue_user'@'198.51.100.2'
IDENTIFIED BY 'blue_user_password';
```

- 各名前付き名前空間のサーバーに接続できることを確認します:

```
shell> mysql -u red_user -h 192.0.2.2 --network-namespace=red -p
Enter password: red_user_password
```

```
mysql> SELECT USER();
+-----+
| USER() |
+-----+
| red_user@192.0.2.2 |
+-----+
```

```
shell> mysql -u blue_user -h 198.51.100.2 --network-namespace=blue -p
Enter password: blue_user_password
```

```
mysql> SELECT USER();
+-----+
| USER() |
+-----+
| blue_user@198.51.100.2 |
+-----+
```

注記

DNS がアドレスを対応するホスト名に解決できるように構成されており、サーバーが `skip_name_resolve` システム変数を有効にして実行されていない場合、`USER()` とは異なる結果が表示され、IP アドレスではなくホスト名を含む値が返されることがあります。

`--network-namespace` オプションを指定せずに `mysql` を起動して、接続試行が成功したかどうか、成功した場合は `USER()` 値がどのように影響を受けるかを確認することもできます。

ネットワークネームスペースのモニタリング

レプリケーションモニタリングのために、これらの情報ソースには、接続に適用可能なネットワークネームスペースを表示するカラムがあります:

- パフォーマンススキーマの `replication_connection_configuration` テーブル。 [セクション 27.12.11.1 「replication_connection_configuration テーブル」](#) を参照してください。
- レプリカサーバー接続メタデータリポジトリ。 [セクション 17.2.4.2 「レプリケーションメタデータリポジトリ」](#) を参照してください。
- `SHOW REPLICA | SLAVE STATUS` ステートメント。 [セクション 13.7.7.35 「SHOW REPLICA | SLAVE STATUS ステートメント」](#) を参照してください。

5.1.15 MySQL Server でのタイムゾーンのサポート

このセクションでは、MySQL で保持されるタイムゾーン設定、名前付き時間サポートに必要なシステムテーブルのロード方法、タイムゾーンの変更を最新の状態に保つ方法、およびうるう秒のサポートを有効にする方法について説明します。

MySQL 8.0.19 以降、タイムゾーンオフセットは挿入された日時値に対してもサポートされます。詳細は、[セクション 11.2.2 「DATE、DATETIME、および TIMESTAMP 型」](#) を参照してください。

- [タイムゾーン変数](#)
- [タイムゾーンテーブルへの移入](#)
- [タイムゾーンの変更による現在の時間の維持](#)
- [タイムゾーンのうるう秒のサポート](#)

レプリケーション設定のタイムゾーン設定の詳細は、[セクション 17.5.1.14 「レプリケーションとシステム関数」](#) および [セクション 17.5.1.33 「レプリケーションとタイムゾーン」](#) を参照してください。

タイムゾーン変数

MySQL Server では、複数のタイムゾーン設定が保持されます:

- システムタイムゾーン。サーバーは起動時に、ホストマシンのタイムゾーンを自動的に判別し、それを使用して `system_time_zone` システム変数を設定しようとします。その後、この値は変更しません。

起動時に MySQL Server のシステムタイムゾーンを明示的に指定するには、`mysqld` を起動する前に `TZ` 環境変数を設定します。`mysqld_safe` を使用してサーバーを起動する場合は、その `--timezone` オプションを使用してシステムのタイムゾーンを設定することもできます。`TZ` および `--timezone` に許可される値は、システムによって異なります。許容可能な値を確認するには、オペレーティングシステムのドキュメントを参照してください。

- サーバーの現在のタイムゾーン。`time_zone` グローバルシステム変数は、サーバーが現在動作しているタイムゾーンを示します。`time_zone` の初期値は `'SYSTEM'` で、サーバーのタイムゾーンがシステムのタイムゾーンと同じであることを示します。

注記

`SYSTEM` に設定されている場合、タイムゾーン計算を必要とするすべての MySQL 関数コールは、システムライブラリコールを実行して現在のシステムタイムゾーンを決定します。このコールはグローバル mutex によって保護される可能性があるため、競合が発生します。

初期グローバルサーバーのタイムゾーン値は、起動時にコマンド行で `--default-time-zone` オプションを使用して明示的に指定するか、オプションファイルで次の行を使用できます:

```
default-time-zone='timezone'
```


`SYSTEM_VARIABLES_ADMIN` 権限 (または非推奨の `SUPER` 権限) がある場合は、次のステートメントを使用して、実行時にグローバルサーバーのタイムゾーン値を設定できます:

```
SET GLOBAL time_zone = timezone;
```

- セッションごとのタイムゾーン。接続する各クライアントには、セッションの `time_zone` 変数で指定された独自のセッションタイムゾーン設定があります。最初、セッション変数は、`time_zone` グローバル変数から値を取得しますが、クライアントは次のステートメントを使用して、それぞれのタイムゾーンを変更できます。

```
SET time_zone = timezone;
```

セッションのタイムゾーン設定は、ゾーン依存の時間値の表示および格納に影響します。これには、`NOW()` や `CURTIME()` などの関数で表示される値や、`TIMESTAMP` カラムに保存し、そこから読み出す値も含まれます。`TIMESTAMP` カラムの値は、格納のためにセッションタイムゾーンから UTC に、取得のために UTC からセッションタイムゾーンに変換されます。

セッションのタイムゾーン設定は、`UTC_TIMESTAMP()` などの関数や `DATE`、`TIME`、`DATETIME` カラムの値によって表示される値には影響しません。また、これらのデータ型の値も UTC で格納されません。タイムゾーンは、`TIMESTAMP` 値から変換するときのみ適用されます。`DATE`、`TIME`、または `DATETIME` 値に対してロケール固有の演算を実行する場合、これらの値を UTC に変換し、演算を実行してから、元に変換し直します。

現在のグローバルおよびセッションのタイムゾーン値は、次のように取得できます:

```
SELECT @@GLOBAL.time_zone, @@SESSION.time_zone;
```

`timezone` 値はいくつかの形式で指定できますが、大文字と小文字は区別されません:

- 値 `'SYSTEM'` として、サーバーのタイムゾーンがシステムのタイムゾーンと同じであることを示します。
- `'+10:00'`、`'-6:00'`、`'+05:30'` などの `+` または `-` の接頭辞が付いた、`[H]H:MM` 形式の UTC からのオフセットを示す文字列として。必要に応じて、先頭のゼロを 10 未満の時間値に使用できます。このような場合、MySQL では、値を格納および取得するときに先頭のゼロが付加されます。MySQL は、`'-00:00'` または `'-0:00'` を `'+00:00'` に変換します。

MySQL 8.0.19 より前は、この値は `'-12:59'` から `'+13:00'` までの範囲である必要がありました。MySQL 8.0.19 以降、許可される範囲は `'-14:00'` から `'+14:00'` までです。
- `'Europe/Helsinki'`、`'US/Eastern'`、`'MET'` や `'UTC'` などの名前付きタイムゾーンとして。

注記

名前付きのタイムゾーンは、`mysql` データベース内のタイムゾーン情報テーブルが作成され移入されている場合のみ使用できます。それ以外の場合は、名前付きタイムゾーンを使用するとエラーになります:

```
mysql> SET time_zone = 'UTC';  
ERROR 1298 (HY000): Unknown or incorrect time zone: 'UTC'
```

タイムゾーンテーブルへの移入

タイムゾーン情報を格納するために、`mysql` システムスキーマにはいくつかのテーブルが存在します ([セクション 5.3 「mysql システムスキーマ」](#) を参照)。MySQL のインストール手順では、タイムゾーンテーブルは作成されますが、ロードされません。これを手動で行うには、次の手順を使用します。

注記

情報は変更することがあるので、タイムゾーン情報のロードは必ずしも 1 回だけの操作とはかぎりません。このような変更が起きた場合、古いルールを使用したアプリケーションは旧式になり、MySQL Server で使用されている情報を最新の状態に維持するために、タイムゾーンテーブルをリロードする必要が生じることがあります。[タイムゾーンの変更による現在の時間の維持](#) を参照してください。

システムに独自の `zoneinfo` データベース (タイムゾーンを記述する一連のファイル) がある場合は、`mysql_tzinfo_to_sql` プログラムを使用してタイムゾーンテーブルをロードします。このようなシステムの例

には、Linux、macOS、FreeBSD および Solaris があります。これらのファイルの 1 つの適切な場所は `/usr/share/zoneinfo` ディレクトリです。システムに zoneinfo データベースがない場合は、このセクションの後半で説明するように、ダウンロード可能なパッケージを使用できます。

コマンド行からタイムゾーンテーブルをロードするには、zoneinfo ディレクトリのパス名を `mysql_tzinfo_to_sql` に渡し、その出力を `mysql` プログラムに送信します。例:

```
mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root -p mysql
```

ここに示す `mysql` コマンドは、`mysql` システムスキーマのテーブルを変更する権限を持つ `root` などのアカウントを使用してサーバーに接続することを前提としています。必要に応じて接続パラメータを調整します。

`mysql_tzinfo_to_sql` は、システムのタイムゾーンファイルを読み取り、そのファイルから SQL ステートメントを生成します。`mysql` はこれらのステートメントを処理して、タイムゾーンテーブルをロードします。

`mysql_tzinfo_to_sql` を使用して、単一のタイムゾーンファイルをロードしたり、うるう秒の情報を生成することもできます:

- タイムゾーン名 `tz_name` に対応した単一のタイムゾーンファイル `tz_file` をロードするには、次のように `mysql_tzinfo_to_sql` を呼び出します。

```
mysql_tzinfo_to_sql tz_file tz_name | mysql -u root -p mysql
```

このアプローチでは、サーバーが認識する名前付きゾーンごとに、個別のコマンドを実行してタイムゾーンファイルをロードする必要があります。

- タイムゾーンでうるう秒を考慮する必要がある場合は、次のようにうるう秒の情報を初期化します (`tz_file` はタイムゾーンファイルの名前です):

```
mysql_tzinfo_to_sql --leap tz_file | mysql -u root -p mysql
```

`mysql_tzinfo_to_sql` の実行後、以前にキャッシュされたタイムゾーンデータが引き続き使用されないようにサーバーを再起動します。

システムに zoneinfo データベース (Windows など) がない場合は、MySQL Developer Zone でダウンロード可能な SQL ステートメントを含むパッケージを使用できます:

```
https://dev.mysql.com/downloads/timezones.html
```

警告

システムに zoneinfo データベースがある場合は、ダウンロード可能なタイムゾーンパッケージを使用しないでください。代わりに、`mysql_tzinfo_to_sql` ユーティリティを使用してください。そうしないと、MySQL とシステム上のほかのアプリケーション間で日時処理に違いが生じることがあります。

ダウンロードした SQL ステートメントタイムゾーンパッケージを使用するには、パッケージを解凍してから、解凍したファイルの内容をタイムゾーンテーブルにロードします:

```
mysql -u root -p mysql < file_name
```

次に、サーバーを再起動します。

警告

MyISAM テーブルを含むダウンロード可能なタイムゾーンパッケージは使用しないでください。これは、古い MySQL バージョンを対象としています。MySQL では、タイムゾーンテーブルに **InnoDB** が使用されるようになりました。これらを **MyISAM** テーブルに置換しようとすると、問題が発生します。

タイムゾーンの変更による現在の時間の維持

タイムゾーンルールが変更されると、古いルールを使用するアプリケーションは期限切れになります。現在の時間に維持するには、システムが現在のタイムゾーン情報を使用していることを確認する必要があります。MySQL では、最新の状態に保つために複数の要因を考慮する必要があります:

- オペレーティングシステムの時間は、そのタイムゾーンが **SYSTEM** に設定されている場合、MySQL Server が時間に使用する値に影響します。オペレーティングシステムが最新のタイムゾーン情報を使用していることを確認します。ほとんどのオペレーティングシステムでは、最新の更新またはサービスパックによってシステムは時間の変更に対応できます。オペレーティングシステムのベンダーの web サイトで、時間の変更に対処する更新を確認します。
- システムの `/etc/localtime` タイムゾーンファイルを、`mysqld` の起動時に有効なものとは異なるルールを使用するバージョンに置き換える場合は、更新されたルールを使用するように `mysqld` を再起動します。そうしないと、システムが時間を変更したときに `mysqld` が認識されないことがあります。
- MySQL で名前付きタイムゾーンを使用する場合は、`mysql` データベースのタイムゾーンテーブルが最新であることを確認してください。
 - システムに独自の `zoneinfo` データベースがある場合は、`zoneinfo` データベースが更新されるたびに MySQL タイムゾーンテーブルをリロードします。
 - システムに独自の `zoneinfo` データベースがない場合、MySQL Developer Zone で更新がないか調べます。新しい更新が使用可能になったら、それをダウンロードし、それを使用して現在のタイムゾーンテーブルのコンテンツを置き換えます。

両方の方法の手順については、[タイムゾーンテーブルへの移入](#)を参照してください。`mysqld` では、検索するタイムゾーン情報がキャッシュされるため、タイムゾーンテーブルの更新後に `mysqld` を再起動して、古いタイムゾーンデータが引き続き提供されないようにします。

サーバーのタイムゾーン設定として使用するか、独自のタイムゾーンを設定するクライアントが使用するために、名前付きタイムゾーンが使用できるかどうか不確かな場合は、タイムゾーンテーブルが空かどうかを調べてください。次のクエリは、タイムゾーン名を含むテーブルに行があるかどうかを判断します。

```
mysql> SELECT COUNT(*) FROM mysql.time_zone_name;
+-----+
| COUNT(*) |
+-----+
|      0 |
+-----+
```

カウントがゼロの場合、テーブルが空であることを示します。この場合、現在名前付きタイムゾーンを使用しているアプリケーションはなく、テーブルを更新する必要はありません (名前付きタイムゾーンのサポートを有効にする場合を除く)。カウントがゼロより大きい場合、テーブルは空ではなく、その内容が名前付きタイムゾーンのサポートに使用できることを示します。この場合、名前付きタイムゾーンを使用するアプリケーションが正しいクエリ結果を取得できるように、必ずタイムゾーンテーブルをリロードしてください。

サマータイムのルール変更に対して MySQL インストールが正しく更新されているかどうかを確認するには、次のようなテストを使用します。この例では、3月11日午前2時に米国で行われる2007年DSTの1時間の更新に適切な値を使用しています。

テストでは、次のクエリが使用されます:

```
SELECT
  CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central') AS time1,
  CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central') AS time2;
```

2つの時間値は、DST変更が行われる時間を示し、名前付きタイムゾーンの使用には、タイムゾーンテーブルを使用する必要があります。結果では、両方のクエリで同じ結果が返されることが期待されます (「米国/中央」タイムゾーンの同等の値に変換された入力時間)。

タイムゾーンテーブルを更新する前に、次のような誤った結果が表示されます:

```
+-----+-----+
| time1          | time2          |
+-----+-----+
| 2007-03-11 01:00:00 | 2007-03-11 02:00:00 |
+-----+-----+
```

テーブルの更新後に、正しい結果が表示されます。

```
+-----+-----+
```

time1	time2
2007-03-11 01:00:00	2007-03-11 01:00:00

タイムゾーンのうるう秒のサポート

うるう秒値は、:59:59 で終わる時間部分を使用して返されます。これは、NOW() などの関数が、うるう秒の間、2、3 秒連続して同じ値を返すことがあることを意味します。:59:60 または :59:61 で終わる時間部分を持つリテラル時間値が無効と見なされることに変わりはありません。

うるう秒の 1 秒前に **TIMESTAMP** 値を検索する必要がある場合、'YYYY-MM-DD hh:mm:ss' 値との比較を使用すると異常な結果が得られる可能性があります。この点について次の例で説明します。セッションタイムゾーンが UTC に変更されるため、内部 **TIMESTAMP** 値 (UTC) と表示される値 (タイムゾーン修正が適用されている値) に違いはありません。

```
mysql> CREATE TABLE t1 (
  a INT,
  ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (ts)
);
Query OK, 0 rows affected (0.01 sec)

mysql> -- change to UTC
mysql> SET time_zone = '+00:00';
Query OK, 0 rows affected (0.00 sec)

mysql> -- Simulate NOW() = '2008-12-31 23:59:59'
mysql> SET timestamp = 1230767999;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t1 (a) VALUES (1);
Query OK, 1 row affected (0.00 sec)

mysql> -- Simulate NOW() = '2008-12-31 23:59:60'
mysql> SET timestamp = 1230768000;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t1 (a) VALUES (2);
Query OK, 1 row affected (0.00 sec)

mysql> -- values differ internally but display the same
mysql> SELECT a, ts, UNIX_TIMESTAMP(ts) FROM t1;
+----+-----+-----+
| a  | ts          | UNIX_TIMESTAMP(ts) |
+----+-----+-----+
| 1  | 2008-12-31 23:59:59 | 1230767999 |
| 2  | 2008-12-31 23:59:59 | 1230768000 |
+----+-----+-----+
2 rows in set (0.00 sec)

mysql> -- only the non-leap value matches
mysql> SELECT * FROM t1 WHERE ts = '2008-12-31 23:59:59';
+----+-----+
| a  | ts          |
+----+-----+
| 1  | 2008-12-31 23:59:59 |
+----+-----+
1 row in set (0.00 sec)

mysql> -- the leap value with seconds=60 is invalid
mysql> SELECT * FROM t1 WHERE ts = '2008-12-31 23:59:60';
Empty set, 2 warnings (0.00 sec)
```

これを回避するには、うるう秒の修正が適用されているカラムに実際に格納されている UTC 値に基づく比較を使用できます:

```
mysql> -- selecting using UNIX_TIMESTAMP value return leap value
mysql> SELECT * FROM t1 WHERE UNIX_TIMESTAMP(ts) = 1230768000;
+----+-----+
| a  | ts          |
+----+-----+
```

```
+-----+
| 2 | 2008-12-31 23:59:59 |
+-----+
1 row in set (0.00 sec)
```

5.1.16 リソースグループ

MySQL では、リソースグループの作成および管理がサポートされており、グループで使用可能なリソースに従ってスレッドが実行されるように、サーバー内で実行されているスレッドを特定のグループに割り当てることができます。グループ属性を使用すると、そのリソースを制御して、グループ内のスレッドによるリソース消費を有効にしたり制限したりすることができます。DBA は、様々なワークロードに応じてこれらの属性を変更できます。

>現在、CPU 時間は管理可能なリソースであり、CPU コア、ハイパースレッド、ハードウェアスレッドなどを含む用語として「**仮想 CPU**」の概念で表現されています。サーバーは起動時に使用可能な仮想 CPU の数を決定し、適切な権限を持つデータベース管理者はこれらの CPU をリソースグループに関連付け、スレッドをグループに割り当てることができます。

たとえば、高優先度で実行する必要のないバッチジョブの実行を管理するために、DBA は **Batch** リソースグループを作成し、サーバーのビジー状態に応じて優先度を上下に調整できます。(おそらく、グループに割り当てられたバッチジョブは、日中は低い優先度で実行し、夜間は高い優先度で実行する必要があります。) DBA は、グループで使用可能な CPU のセットを調整することもできます。グループを有効または無効にして、スレッドを割り当てることができるかどうかを制御できます。

次の各セクションでは、MySQL でのリソースグループの使用について説明します:

- [リソースグループ要素](#)
- [リソースグループ属性](#)
- [リソースグループ管理](#)
- [リソースグループのレプリケーション](#)
- [リソースグループの制限](#)

重要

一部のプラットフォームまたは MySQL サーバー構成では、リソースグループが使用できないか、制限があります。特に、一部のインストール方法では、Linux システムで手動のステップが必要になる場合があります。詳細は、[リソースグループの制限](#)を参照してください。

リソースグループ要素

これらの機能は、MySQL でリソースグループを管理するための SQL インタフェースを提供します:

- SQL ステートメントを使用すると、リソースグループを作成、変更および削除したり、リソースグループにスレッドを割り当てることができます。最適化ヒントを使用すると、個々のステートメントをリソースグループに割り当てることができます。
- リソースグループ権限は、リソースグループ操作を実行できるユーザーを制御します。
- [INFORMATION_SCHEMA.RESOURCE_GROUPS](#) テーブルはリソースグループ定義に関する情報を公開し、パフォーマンススキーマ [threads](#) テーブルには各スレッドのリソースグループ割り当てが表示されます。
- ステータス変数は、各管理 SQL ステートメントの実行数を提供します。

リソースグループ属性

リソースグループには、グループを定義する属性があります。すべての属性は、グループ作成時に設定できます。一部の属性は作成時に固定されますが、それ以降はいつでも変更できます。

これらの属性はリソースグループの作成時に定義され、変更できません:

- 各グループには名前があります。リソースグループ名はテーブル名やカラム名などの識別子であり、特殊文字が含まれているか予約語でないかぎり、SQL ステートメントで引用符で囲む必要はありません。グループ名は大/小文字が区別されず、64 文字以内にする必要があります。
- 各グループには、**SYSTEM** または **USER** のいずれかのタイプがあります。リソースグループタイプは、あとで説明するように、グループに割り当て可能な優先順位の値の範囲に影響します。この属性と許可される優先順位の違いを使用すると、CPU リソースの競合をユーザースレッドから保護するためにシステムスレッドを識別できます。

システムスレッドとユーザースレッドは、パフォーマンススキーマ `threads` テーブルに一覧表示されているバックグラウンドスレッドとフォアグラウンドスレッドに対応しています。

これらの属性はリソースグループの作成時に定義され、後でいつでも変更できます：

- CPU アフィニティは、リソースグループが使用できる仮想 CPU のセットです。アフィニティには、使用可能な CPU の空でない任意のサブセットを指定できます。グループにアフィニティがない場合は、使用可能なすべての CPU を使用できます。
- スレッド優先度は、リソースグループに割り当てられたスレッドの実行優先度です。優先度の値の範囲は -20 (最も高い優先度) から 19 (最も低い優先度) です。システムグループとユーザーグループの両方で、デフォルトの優先度は 0 です。

システムグループはユーザーグループよりも高い優先度を許可されるため、ユーザースレッドの優先度がシステムスレッドより高くなることはありません：

- システムリソースグループの場合、許可される優先順位の範囲は -20 から 0 です。
- ユーザーリソースグループの場合、許可される優先順位の範囲は 0 から 19 です。
- 各グループを有効または無効にして、管理者がスレッド割当てを制御できるようにします。スレッドは、有効なグループにのみ割り当てることができます。

リソースグループ管理

デフォルトでは、それぞれ `SYS_default` および `USR_default` という名前のシステムグループとユーザーグループがあります。これらのデフォルトグループは削除できず、その属性は変更できません。各デフォルトグループには、CPU アフィニティと優先順位 0 はありません。

新しく作成されたシステムスレッドとユーザースレッドは、それぞれ `SYS_default` グループと `USR_default` グループに割り当てられます。

ユーザー定義のリソースグループの場合、すべての属性はグループの作成時に割り当てられます。グループを作成した後は、`name` 属性と `type` 属性を除き、その属性を変更できます。

ユーザー定義リソースグループを作成および管理するには、次の SQL ステートメントを使用します：

- `CREATE RESOURCE GROUP` により、新しいグループが作成されます。 [セクション 13.7.2.2 「CREATE RESOURCE GROUP ステートメント」](#) を参照してください。
- `ALTER RESOURCE GROUP` は、既存のグループを変更します。 [セクション 13.7.2.1 「ALTER RESOURCE GROUP ステートメント」](#) を参照してください。
- `DROP RESOURCE GROUP` は、既存のグループを削除します。 [セクション 13.7.2.3 「DROP RESOURCE GROUP ステートメント」](#) を参照してください。

これらのステートメントには `RESOURCE_GROUP_ADMIN` 権限が必要です。

リソースグループ割当てを管理するには、次の機能を使用します：

- `SET RESOURCE GROUP` は、スレッドをグループに割り当てます。 [セクション 13.7.2.4 「SET RESOURCE GROUP ステートメント」](#) を参照してください。
- `RESOURCE_GROUP` オプティマイザヒントは、個々のステートメントをグループに割り当てます。 [セクション 8.9.3 「オプティマイザヒント」](#) を参照してください。

これらの操作には、`RESOURCE_GROUP_ADMIN` または `RESOURCE_GROUP_USER` 権限が必要です。

リソースグループ定義は、サーバーの再起動後もグループが保持されるように、`resource_groups` データディクショナリテーブルに格納されます。`resource_groups` はデータディクショナリの一部であるため、ユーザーは直接アクセスできません。リソースグループ情報は、データディクショナリテーブルのビューとして実装される `INFORMATION_SCHEMA.RESOURCE_GROUPS` テーブルを使用して使用できます。[セクション 26.26 「INFORMATION_SCHEMA RESOURCE_GROUPS テーブル」](#) を参照してください。

最初、`RESOURCE_GROUPS` テーブルには、デフォルトグループを説明する次の行があります:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.RESOURCE_GROUPS;
+-----+-----+-----+-----+-----+-----+
***** 1. row *****
RESOURCE_GROUP_NAME: USR_default
RESOURCE_GROUP_TYPE: USER
RESOURCE_GROUP_ENABLED: 1
VCPU_IDS: 0-3
THREAD_PRIORITY: 0
***** 2. row *****
RESOURCE_GROUP_NAME: SYS_default
RESOURCE_GROUP_TYPE: SYSTEM
RESOURCE_GROUP_ENABLED: 1
VCPU_IDS: 0-3
THREAD_PRIORITY: 0
```

`THREAD_PRIORITY` の値は 0 で、デフォルトの優先度を示します。`VCPU_IDS` の値は、使用可能なすべての CPU で構成される範囲を示します。デフォルトグループの場合、表示される値は、MySQL サーバーが実行されているシステムによって異なります。

前述の説明では、高優先度で実行する必要がないバッチジョブの実行を管理するために、`Batch` という名前のリソースグループを含むシナリオについて説明しました。このようなグループを作成するには、次のようなステートメントを使用します:

```
CREATE RESOURCE GROUP Batch
TYPE = USER
VCPU = 2-3      -- assumes a system with at least 4 CPUs
THREAD_PRIORITY = 10;
```

リソースグループが予想どおりに作成されたことを確認するには、`RESOURCE_GROUPS` テーブルを確認します:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.RESOURCE_GROUPS
WHERE RESOURCE_GROUP_NAME = 'Batch';
+-----+-----+-----+-----+-----+-----+
***** 1. row *****
RESOURCE_GROUP_NAME: Batch
RESOURCE_GROUP_TYPE: USER
RESOURCE_GROUP_ENABLED: 1
VCPU_IDS: 2-3
THREAD_PRIORITY: 10
```

`THREAD_PRIORITY` 値が 10 ではなく 0 の場合は、プラットフォームまたはシステム構成によってリソースグループの機能が制限されているかどうかを確認します。[リソースグループの制限](#) を参照してください。

`Batch` グループにスレッドを割り当てるには、次のようにします:

```
SET RESOURCE GROUP Batch FOR thread_id;
```

その後、指定されたスレッド内のステートメントは、`Batch` グループリソースで実行されます。

セッション所有の現在のスレッドが `Batch` グループ内に存在する必要がある場合は、セッション内で次のステートメントを実行します:

```
SET RESOURCE GROUP Batch;
```

その後、セッション内のステートメントは `Batch` グループリソースを使用して実行されます。

`Batch` グループを使用して単一のステートメントを実行するには、`RESOURCE_GROUP` オプティマイザヒントを使用します:

```
INSERT /*+ RESOURCE_GROUP(Batch) */ INTO t2 VALUES(2);
```

Batch グループに割り当てられたスレッドはそのリソースで実行され、必要に応じて変更できます:

- システムの負荷が高い場合は、グループに割り当てられている CPU の数を減らすか、その優先度を低くするか、または (図に示すように) 両方を行います:

```
ALTER RESOURCE GROUP Batch
VCPU = 3
THREAD_PRIORITY = 19;
```

- システムの負荷が軽い場合は、グループに割り当てられている CPU の数を増やすか、その優先度を上げるか、または (図に示すように) 両方を行います:

```
ALTER RESOURCE GROUP Batch
VCPU = 0-3
THREAD_PRIORITY = 0;
```

リソースグループのレプリケーション

リソースグループ管理は、リソースグループが発生したサーバーに対してローカルです。リソースグループの SQL ステートメントおよび `resource_groups` データディクショナリテーブルに対する変更はバイナリログに書き込まれず、レプリケートされません。

リソースグループの制限

一部のプラットフォームまたは MySQL サーバー構成では、リソースグループは使用できないか、制限があります:

- スレッドプールプラグインがインストールされている場合、リソースグループは使用できません。
- リソースグループは、CPU をスレッドにバインドする API を提供しない macOS では使用できません。
- FreeBSD および Solaris では、リソースグループスレッドの優先順位は無視されます。(事実上、すべてのスレッドは優先度 0 で実行されます。) 優先度を変更しようとすると、警告が表示されます:

```
mysql> ALTER RESOURCE GROUP abc THREAD_PRIORITY = 10;
Query OK, 0 rows affected, 1 warning (0.18 sec)
```

```
mysql> SHOW WARNINGS;
```

```
+-----+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+-----+
| Warning | 4560 | Attribute thread_priority is ignored (using default value). |
+-----+-----+-----+-----+
```

- Linux では、`CAP_SYS_NICE` 機能が設定されていないかぎり、リソースグループのスレッド優先順位は無視されます。`CAP_SYS_NICE` 機能をプロセスに付与すると、一定範囲の権限が有効になります。完全なリストは、<http://man7.org/linux/man-pages/man7/capabilities.7.html> を参照してください。この機能を有効にする場合は注意してください。

systemd およびカーネルによるアンビエント機能 (Linux 4.3 以降) のサポートを使用する Linux プラットフォームでは、`CAP_SYS_NICE` 機能を有効にするには、MySQL サービスファイルを変更し、`mysqld` バイナリを変更しないでおくことをお勧めします。MySQL のサービスファイルを調整するには、この手順を使用します:

- ご使用のプラットフォームに適したコマンドを実行します:

- Oracle Linux、Red Hat および Fedora システム:

```
shell> sudo systemctl edit mysqld
```

- SUSE、Ubuntu および Debian システム:

```
shell> sudo systemctl edit mysql
```

- エディタを使用して、次のテキストをサービスファイルに追加します:

```
[Service]
AmbientCapabilities=CAP_SYS_NICE
```

- MySQL サービスを再起動します。

前述のように `CAP_SYS_NICE` 機能を有効にできない場合は、`mysqld` 実行可能ファイルへのパス名を指定して、`setcap` コマンドを使用して手動で設定できます (`sudo` アクセスが必要です)。`getcap` を使用して機能を確認できます。例:

```
shell> sudo setcap cap_sys_nice+ep /path/to/mysqld
shell> getcap /path/to/mysqld
/path/to/mysqld = cap_sys_nice+ep
```

安全対策として、`mysqld` バイナリの実行を `root` ユーザーおよび `mysql` グループメンバーシップを持つユーザーに制限します:

```
shell> sudo chown root:mysql /path/to/mysqld
shell> sudo chmod 0750 /path/to/mysqld
```

重要

`setcap` を手動で使用する必要がある場合は、再インストールのたびに実行する必要があります。

- Windows では、スレッドは 5 つのスレッド優先度レベルのいずれかで実行されます。-20 から 19 のリソースグループスレッド優先順位の範囲は、次のテーブルに示すレベルにマップされます。

表 5.5 Windows でのリソースグループスレッドの優先順位

優先度範囲	Windows 優先順位レベル
-20 から -10	THREAD_PRIORITY_HIGHEST
-9 から -1	THREAD_PRIORITY_ABOVE_NORMAL
0	THREAD_PRIORITY_NORMAL
1 から 10 まで	THREAD_PRIORITY_BELOW_NORMAL
11 から 19 まで	THREAD_PRIORITY_LOWEST

5.1.17 サーバー側ヘルプのサポート

MySQL Server は、MySQL リファレンスマニュアル ([セクション13.8.3「HELP ステートメント」](#) を参照) から情報を返す `HELP` ステートメントをサポートしています。この情報は、`mysql` スキーマの複数のテーブルに格納されます ([セクション5.3「mysql システムスキーマ」](#) を参照)。`HELP` ステートメントを正しく操作するには、これらのヘルプテーブルを初期化する必要があります。

Unix でバイナリまたはソース配布を使用した MySQL の新規インストールの場合、データディレクトリを初期化するとヘルプテーブルのコンテンツの初期化が行われます ([セクション2.10.1「データディレクトリの初期化」](#) を参照)。Linux の RPM 配布または Windows のバイナリ配布の場合、コンテンツの初期化は MySQL インストールプロセスの一部として実行されます。

バイナリ配布を使用した MySQL アップグレードの場合、`help-table` コンテンツは MySQL 8.0.16 の時点でサーバーによって自動的にアップグレードされます。MySQL 8.0.16 より前は、コンテンツは自動的にアップグレードされませんが、手動でアップグレードできます。`share` または `share/mysql` ディレクトリ内から `fill_help_tables.sql` ファイルを見つけます。場所をそのディレクトリに変更し、`mysql` クライアントを使用してファイルを次のように処理します。

```
mysql -u root -p mysql < fill_help_tables.sql
```

ここに示すコマンドは、`mysql` スキーマのテーブルを変更する権限を持つ `root` などのアカウントを使用してサーバーに接続することを前提としています。必要に応じて接続パラメータを調整します。

MySQL 8.0.16 より前は、Git および MySQL 開発ソースツリーを使用している場合、ソースツリーには `fill_help_tables.sql` の「stub」バージョンのみが含まれていました。非スタブコピーを取得するには、ソースまたはバイナリ配布からコピーを使用します。

注記

各 MySQL シリーズには独自のシリーズ固有のリファレンスマニュアルがあるため、ヘルプテーブルのコンテンツもシリーズ固有です。ヘルプテーブルの内容は MySQL シリーズと

一致する必要があるため、これはレプリケーションに影響します。MySQL 8.0 ヘルプコンテンツを MySQL 8.0 レプリケーションサーバーにロードする場合、そのコンテンツを別の MySQL シリーズからレプリカサーバーにレプリケートし、そのコンテンツが適切でないという意味がありません。このため、レプリケーションシナリオで個々のサーバーをアップグレードする場合は、前述の手順を使用して各サーバーヘルプテーブルをアップグレードする必要があります。(手動によるヘルプコンテンツのアップグレードは、8.0.16 より前のバージョンのレプリケーションサーバーでのみ必要です。前述の手順で説明したように、コンテンツアップグレードは MySQL 8.0.16 の時点で自動的に行われます。)

5.1.18 クライアントセッション状態の変更のサーバートラッキング

MySQL サーバーは、いくつかのセッション状態トラッカを実装します。クライアントは、これらのトラッカがセッションステートに対する変更の通知を受信できるようにすることができます。

トラッカメカニズムの 1 つの用途は、MySQL コネクタおよびクライアントアプリケーションが、あるサーバーから別のサーバーへのセッション移行を許可するためにセッションコンテキストが使用可能かどうかを判断する手段を提供することです。(ロードバランシングされた環境でセッションを変更するには、切替えが可能かどうかを判断する際に考慮する必要があるセッション状態があるかどうかを検出する必要があります。)

トラッカメカニズムの別の用途は、あるセッションから別のセッションにトランザクションを移動できるタイミングをアプリケーションが把握できるようにすることです。トランザクション状態トラッキングにより、これが有効になります。これは、ビジー状態のサーバーから負荷の低いサーバーにトランザクションを移動するアプリケーションで役立ちます。たとえば、クライアント接続プールを管理するロードバランシングコネクタは、プール内の使用可能なセッション間でトランザクションを移動できます。

ただし、セッション切替えは任意の時点では実行できません。読取りまたは書込みが行われたトランザクションの途中でセッションがある場合、別のセッションに切り替えると、元のセッションでトランザクションがロールバックされます。セッション切替えは、トランザクション内で読取りまたは書込みがまだ実行されていない場合にのみ実行する必要があります。

トランザクションを適切に切り替える場合の例を次に示します:

- `START TRANSACTION` の直後
- `COMMIT AND CHAIN` の後

トランザクションの状態を把握するだけでなく、トランザクションが別のセッションに移動された場合にも同じ特性を使用できるように、トランザクションの特性を把握しておくことが役立ちます。この目的に関連する特性は次のとおりです:

```
READ ONLY  
READ WRITE  
ISOLATION LEVEL  
WITH CONSISTENT SNAPSHOT
```

前述のセッション切替えアクティビティをサポートするために、次のタイプのクライアントセッション状態情報の通知を使用できます:

- クライアントセッションステートの次の属性に対する変更:
 - デフォルトスキーマ (データベース)。
 - システム変数のセッション固有の値。
 - ユーザー定義変数。
 - 一時テーブル
 - プリベアドステートメント。

`session_track_state_change` システム変数は、このトラッカを制御します。

- デフォルトのスキーマ名への変更。 `session_track_schema` システム変数は、このトラッカを制御します。

- システム変数のセッション値の変更。 [session_track_system_variables](#) システム変数は、このトラッカを制御します。
- 使用可能 GTID。 [session_track_gtids](#) システム変数は、このトラッカを制御します。
- トランザクションの状態と特性に関する情報。 [session_track_transaction_info](#) システム変数は、このトラッカを制御します。

トラッカ関連のシステム変数の詳細は、[セクション5.1.8「サーバーシステム変数」](#)を参照してください。これらのシステム変数は、発生する変更通知を制御できますが、通知情報にアクセスする方法は提供しません。通知は、セッション状態の変更を検出できるように OK パケットにトラッカ情報を含む MySQL クライアント/サーバープロトコルで発生します。クライアントアプリケーションがサーバーから返された OK パケットから状態変更情報を抽出できるようにするために、MySQL C API には次の関数のペアが用意されています:

- [mysql_session_track_get_first\(\)](#) は、サーバーから受信した状態変更情報の最初の部分をフェッチします。[mysql_session_track_get_first\(\)](#)を参照してください。
- [mysql_session_track_get_next\(\)](#) は、サーバーから受信した残りの状態変更情報をフェッチします。[mysql_session_track_get_first\(\)](#) が正常にコールされた後、成功を返すかぎり、この関数を繰り返しコールします。[mysql_session_track_get_next\(\)](#)を参照してください。

`mysqltest` プログラムには、セッショントラッカ通知が発生するかどうかを制御する [disable_session_track_info](#) および [enable_session_track_info](#) コマンドがあります。これらのコマンドを使用して、SQL ステートメントによって生成される通知をコマンドラインから確認できます。ファイル `testscript` に次の `mysqltest` スクリプトが含まれているとします:

```
DROP TABLE IF EXISTS test.t1;
CREATE TABLE test.t1 (i INT, f FLOAT);
--enable_session_track_info
SET @@SESSION.session_track_schema=ON;
SET @@SESSION.session_track_system_variables="*";
SET @@SESSION.session_track_state_change=ON;
USE information_schema;
SET NAMES 'utf8mb4';
SET @@SESSION.session_track_transaction_info='CHARACTERISTICS';
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET TRANSACTION READ WRITE;
START TRANSACTION;
SELECT 1;
INSERT INTO test.t1 () VALUES();
INSERT INTO test.t1 () VALUES(1, RAND());
COMMIT;
```

次のようにスクリプトを実行して、有効なトラッカによって提供される情報を確認します。様々なトラッカについて `mysqltest` によって表示される `Tracker:`情報の詳細は、[mysql_session_track_get_first\(\)](#)を参照してください。

```
shell> mysqltest < testscript
DROP TABLE IF EXISTS test.t1;
CREATE TABLE test.t1 (i INT, f FLOAT);
SET @@SESSION.session_track_schema=ON;
SET @@SESSION.session_track_system_variables="*";
-- Tracker : SESSION_TRACK_SYSTEM_VARIABLES
-- session_track_system_variables
-- *

SET @@SESSION.session_track_state_change=ON;
-- Tracker : SESSION_TRACK_SYSTEM_VARIABLES
-- session_track_state_change
-- ON

USE information_schema;
-- Tracker : SESSION_TRACK_SCHEMA
-- information_schema

-- Tracker : SESSION_TRACK_STATE_CHANGE
-- 1

SET NAMES 'utf8mb4';
-- Tracker : SESSION_TRACK_SYSTEM_VARIABLES
-- character_set_client
```

```
-- utf8mb4
-- character_set_connection
-- utf8mb4
-- character_set_results
-- utf8mb4

-- Tracker : SESSION_TRACK_STATE_CHANGE
-- 1

SET @@SESSION.session_track_transaction_info='CHARACTERISTICS';
-- Tracker : SESSION_TRACK_SYSTEM_VARIABLES
-- session_track_transaction_info
-- CHARACTERISTICS

-- Tracker : SESSION_TRACK_STATE_CHANGE
-- 1

-- Tracker : SESSION_TRACK_TRANSACTION_CHARACTERISTICS
--

-- Tracker : SESSION_TRACK_TRANSACTION_STATE
-- _____

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
-- Tracker : SESSION_TRACK_TRANSACTION_CHARACTERISTICS
-- SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

SET TRANSACTION READ WRITE;
-- Tracker : SESSION_TRACK_TRANSACTION_CHARACTERISTICS
-- SET TRANSACTION ISOLATION LEVEL SERIALIZABLE; SET TRANSACTION READ WRITE;

START TRANSACTION;
-- Tracker : SESSION_TRACK_TRANSACTION_CHARACTERISTICS
-- SET TRANSACTION ISOLATION LEVEL SERIALIZABLE; START TRANSACTION READ WRITE;

-- Tracker : SESSION_TRACK_TRANSACTION_STATE
-- T_____

SELECT 1;
1
1
-- Tracker : SESSION_TRACK_TRANSACTION_STATE
-- T____S_

INSERT INTO test.t1 () VALUES();
-- Tracker : SESSION_TRACK_TRANSACTION_STATE
-- T__W_S_

INSERT INTO test.t1 () VALUES(1, RAND());
-- Tracker : SESSION_TRACK_TRANSACTION_STATE
-- T__WsS_

COMMIT;
-- Tracker : SESSION_TRACK_TRANSACTION_CHARACTERISTICS
--

-- Tracker : SESSION_TRACK_TRANSACTION_STATE
-- _____

ok
```

START TRANSACTION ステートメントの前に、次のトランザクションの分離レベルおよびアクセスモード特性を設定する 2 つの **SET TRANSACTION** ステートメントが実行されます。**SESSION_TRACK_TRANSACTION_CHARACTERISTICS** 値は、次に設定されたトランザクション値を示します。

トランザクションを終了する **COMMIT** ステートメントに続いて、**SESSION_TRACK_TRANSACTION_CHARACTERISTICS** 値が空として報告されます。これは、トランザクションの開始前に設定された次のトランザクション特性がリセットされ、セッションのデフォルトが適用されることを示します。これらのセッションデフォルトに対する変更を追跡するには、**transaction_isolation** および **transaction_read_only** システム変数のセッション値を追跡します。

5.1.19 サーバーの停止プロセス

サーバーのシャットダウンプロセスは、次のように実行されます。

1. シャットダウンプロセスが開始されます。

これはいくつかの方法で開始できます。たとえば、`SHUTDOWN` 権限を持つユーザーは `mysqladmin shutdown` コマンドを実行できます。`mysqladmin` は、MySQL によってサポートされているすべてのプラットフォーム上で使用できます。オペレーティングシステムに固有のほかのシャットダウン開始方式も使用可能で、Unix ではサーバーが `SIGTERM` シグナルを受け取るとサーバーがシャットダウンします。Windows 上でサービスとして実行中のサーバーは、サービスマネージャーからシャットダウンを指示されるとシャットダウンします。

2. サーバーは必要に応じてシャットダウンスレッドを作成します。

シャットダウンが開始された方法によっては、シャットダウンプロセスを処理するためのスレッドをサーバーが作成することがあります。シャットダウンがクライアントによってリクエストされた場合、シャットダウンスレッドが作成されます。シャットダウンが `SIGTERM` を受信したことによるものである場合、シグナルスレッドそれ自体がシャットダウンを処理したり、処理を行うための別のスレッドを作成したりすることがあります。サーバーがシャットダウンスレッドを作成しようとしたが作成できない場合(メモリーが不足する場合など)、診断メッセージが発行されてエラーログに示されます。

```
Error: Can't create thread to kill server
```

3. サーバーは新しい接続の受け入れを停止します。

シャットダウン中に新しいアクティビティーが開始されるのを防ぐために、サーバーは通常、接続を `listen` するネットワークインタフェースのハンドラを閉じて新しいクライアント接続の受け入れを停止します。接続には TCP/IP ポート、Unix ソケットファイル、Windows 名前付きパイプ、および Windows の共有メモリーがあります。

4. サーバーは現在のアクティビティーを終了します。

クライアント接続に関連付けられている各スレッドについて、サーバーはクライアントへの接続を切断し、スレッドに強制終了のマークを付けます。スレッドは、そのようなマークを付けられたことが通知されると終了します。アイドル状態の接続のスレッドは、ただちに終了します。ステートメントを現在処理中のスレッドは、その状態を定期的に検査するため、終了するのに時間がかかります。スレッド終了についての追加情報については、[MyISAM テーブルで強制終了された REPAIR TABLE または OPTIMIZE TABLE 操作に関する指示](#)については、[セクション13.7.8.4「KILL ステートメント」](#)を参照してください。

オープン中のトランザクションがあるスレッドでは、トランザクションがロールバックされます。スレッドが非トランザクションテーブルを更新している場合、複数行の `UPDATE` や `INSERT` などの操作は完了前に終了する可能性があるため、テーブルを部分的に更新したままにしておくことができます。

サーバーがレプリケーションソースサーバーの場合、現在接続されているレプリカに関連付けられているスレッドはほかのクライアントスレッドと同様に扱われます。つまり、それぞれに強制終了のマークを付け、その状態を次回検査するときに終了します。

サーバーがレプリカサーバーの場合、クライアントスレッドを強制終了としてマークする前に、レプリケーション I/O および SQL スレッドがアクティブであれば停止します。SQL スレッドは、(レプリケーションの問題を起こすことを防ぐために)現在のステートメントを完了することが可能で、そのあとで終了します。SQL スレッドがこの時点でトランザクションの途中である場合、サーバーは現在のレプリケーションイベントグループ(ある場合)が実行を完了するか、ユーザーが `KILL QUERY` または `KILL CONNECTION` ステートメントを発行するまで待機します。[セクション13.4.2.10「STOP SLAVE | REPLICA ステートメント」](#)も参照してください。非トランザクションステートメントはロールバックできないため、クラッシュに対する安全性を持つレプリケーションを保証するにはトランザクションテーブルのみを使用してください。

注記

レプリカのクラッシュの安全性を保証するには、`--relay-log-recovery` を有効にしてレプリカを実行する必要があります。

[セクション17.2.4「リレーログおよびレプリケーションメタデータリポジトリ」](#)も参照してください。)

5. サーバーはシャットダウンするかストレージエンジンを閉じます。

この段階で、サーバーはテーブルキャッシュをフラッシュして、オープン中のすべてのテーブルを閉じます。

各ストレージエンジンは、管理するテーブルに必要なすべての動作を実行します。InnoDB はバッファプールをディスクにフラッシュし (innodb_fast_shutdown が 2 である場合を除く)、現在の LSN をテーブルスペースに書き込み、その独自の内部スレッドを終了します。MyISAM は、テーブルの保留中のインデックス書き込みをフラッシュします。

6. サーバーが終了します。

管理プロセスに情報を提供するために、サーバーは次のリストに示す終了コードのいずれかを返します。カッコ内のフレーズは、systemd がサーバーの管理に使用されるプラットフォームについて、systemd がコードに回答して実行するアクションを示します。

- 0 = 正常終了 (再起動は行われません)
- 1 = 正常に終了しませんでした (再起動は行われません)
- 2 = 正常に終了しませんでした (再起動完了)

5.2 MySQL データディレクトリ

MySQL サーバーによって管理される情報は、データディレクトリと呼ばれるディレクトリに格納されます。次のリストでは、データディレクトリで通常検出される項目について簡単に説明し、追加情報の相互参照を示します:

- データディレクトリのサブディレクトリ。データディレクトリの各サブディレクトリはデータベースディレクトリで、サーバーによって管理されるデータベースに対応しています。すべての MySQL インストールには、特定の標準データベースがあります:
 - `mysql` ディレクトリは、MySQL サーバーの実行時に必要な情報を含む `mysql` システムスキーマに対応しています。このデータベースには、データディクショナリテーブルおよびシステムテーブルが含まれます。セクション 5.3 「`mysql` システムスキーマ」を参照してください。
 - `performance_schema` ディレクトリは、実行時にサーバーの内部実行を検査するために使用される情報を提供するパフォーマンススキーマに対応しています。第27章 「MySQL パフォーマンススキーマ」を参照してください。
 - `sys` ディレクトリは、パフォーマンススキーマ情報をより簡単に解釈できるように一連のオブジェクトを提供する `sys` スキーマに対応しています。第28章 「MySQL `sys` スキーマ」を参照してください。
 - `ndbinfo` ディレクトリは、NDB Cluster に固有の情報を格納する `ndbinfo` データベースに対応します (NDB Cluster を含むように構築されたインストールにのみ存在します)。セクション 23.5.14 「`ndbinfo`: NDB Cluster 情報データベース」を参照してください。

他のサブディレクトリは、ユーザーまたはアプリケーションによって作成されたデータベースに対応します。

注記

`INFORMATION_SCHEMA` は標準データベースですが、その実装では対応するデータベースディレクトリは使用されません。

- サーバーによって書き込まれるログファイル。セクション 5.4 「MySQL Server ログ」を参照してください。
- InnoDB のテーブルスペースおよびログファイル。第15章 「InnoDB ストレージエンジン」を参照してください。
- デフォルト/自動生成された SSL および RSA 証明書とキーファイル。セクション 6.3.3 「SSL および RSA 証明書とキーの作成」を参照してください。
- サーバードキュメント ID ファイル (サーバーの実行中)。
- 永続化されたグローバルシステム変数設定を格納する `mysqld-auto.cnf` ファイル。セクション 13.7.6.1 「変数代入の SET 構文」を参照してください。

前述のリストの一部のアイテムは、サーバーを再構成することで別の場所に再配置できます。また、`--datadir` オプションを使用すると、データディレクトリ自体の場所を変更できます。特定の MySQL インストールについて、サーバー構成をチェックして、アイテムが移動されたかどうかを確認します。

5.3 mysql システムスキーマ

`mysql` スキーマはシステムスキーマです。これには、MySQL サーバーの実行時に必要な情報を格納するテーブルが含まれます。広範なカテゴリ化とは、`mysql` スキーマに、データベースオブジェクトメタデータを格納するデータディクショナリテーブルと、他の操作目的で使用されるシステムテーブルが含まれていることです。次の説明では、システムテーブルのセットをさらに小さなカテゴリに分割します。

- [データディクショナリテーブル](#)
- [システムテーブルの付与](#)
- [オブジェクト情報システムテーブル](#)
- [ログシステムテーブル](#)
- [サーバー側ヘルプシステムのテーブル](#)
- [タイムゾーンシステムテーブル](#)
- [レプリケーションシステムテーブル](#)
- [オプティマイザシステムテーブル](#)
- [その他のシステムテーブル](#)

このセクションの残りの部分では、各カテゴリのテーブルを、追加情報の相互参照とともに列挙します。データディクショナリテーブルおよびシステムテーブルは、特に指定がない限り、`InnoDB` ストレージエンジンを使用します。

`mysql` システムテーブルおよびデータディクショナリテーブルは、MySQL データディレクトリの `mysql.ibd` という名前の単一の `InnoDB` テーブルスペースファイルに存在します。以前は、これらのテーブルは `mysql` データベースディレクトリの個々のテーブルスペースファイルに作成されていました。

保存データ暗号化は、`mysql` システムスキーマテーブルスペースに対して有効にできます。詳細は、[セクション 15.13「InnoDB 保存データ暗号化」](#)を参照してください。

データディクショナリテーブル

これらのテーブルは、データベースオブジェクトに関するメタデータを含むデータディクショナリで構成されます。追加情報については [第14章「MySQL データディクショナリ」](#)を参照してください。

重要

データディクショナリは、MySQL 8.0 で新しく追加されました。データディクショナリ対応サーバーには、以前の MySQL リリースと比較した一般的な操作上の違いがいくつかあります。詳細は、[セクション14.7「データディクショナリの使用方法の違い」](#)を参照してください。また、MySQL 5.7 から MySQL 8.0 へのアップグレードの場合、アップグレード手順は以前の MySQL リリースと多少異なり、特定の前提条件を確認してインストールのアップグレード準備状況を確認する必要があります。詳細は、[セクション2.11「MySQL のアップグレード」](#) (特に [セクション2.11.5「アップグレード用のインストールの準備」](#)) を参照してください。

- `catalogs`: カタログ情報。
- `character_sets`: 使用可能な文字セットに関する情報。
- `check_constraints`: テーブルに定義されている `CHECK` 制約に関する情報。 [セクション13.1.20.6「CHECK 制約」](#)を参照してください。
- `collations`: 各文字セットの照合順序に関する情報。

- `column_statistics`: カラム値のヒストグラム統計。 [セクション8.9.6「オブティマイザ統計」](#)を参照してください。
- `column_type_elements`: カラムで使用される型に関する情報。
- `columns`: テーブルのカラムに関する情報。
- `dd_properties`: バージョンなどのデータディクショナリのプロパティを識別するテーブル。 サーバーはこれを使用して、データディクショナリを新しいバージョンにアップグレードする必要があるかどうかを判断します。
- `events`: イベントスケジューラのイベントに関する情報。 [セクション25.4「イベントスケジューラの使用」](#)を参照してください。 `--skip-grant-tables` オプションを指定してサーバーを起動すると、イベントスケジューラは無効になり、テーブルに登録されているイベントは実行されません。 [セクション25.4.2「イベントスケジューラの構成」](#)を参照してください。
- `foreign_keys`, `foreign_key_column_usage`: 外部キーに関する情報。
- `index_column_usage`: インデックスで使用されるカラムに関する情報。
- `index_partitions`: インデックスで使用されるパーティションに関する情報。
- `index_stats`: `ANALYZE TABLE` の実行時に生成される動的インデックス統計の格納に使用されます。
- `indexes`: テーブルインデックスに関する情報。
- `innodb_ddl_log`: クラッシュセーフな DDL 操作の DDL ログを格納します。
- `parameter_type_elements`: ストアドプロシージャとストアドファンクションのパラメータ、およびストアドファンクションの戻り値に関する情報。
- `parameters`: ストアドプロシージャおよびストアドファンクションに関する情報。 [セクション25.2「ストアドルーチンの使用」](#)を参照してください。
- `resource_groups`: リソースグループに関する情報。 [セクション5.1.16「リソースグループ」](#)を参照してください。
- `routines`: ストアドプロシージャおよびストアドファンクションに関する情報。 [セクション25.2「ストアドルーチンの使用」](#)を参照してください。
- `schemata`: `schemata` に関する情報。 MySQL では、スキーマはデータベースであるため、このテーブルはデータベースに関する情報を提供します。
- `st_spatial_reference_systems`: 空間データに使用可能な空間参照システムに関する情報。
- `table_partition_values`: テーブルパーティションで使用される値に関する情報。
- `table_partitions`: テーブルで使用されるパーティションに関する情報。
- `table_stats`: `ANALYZE TABLE` の実行時に生成される動的テーブル統計に関する情報。
- `tables`: データベース内のテーブルに関する情報。
- `tablespace_files`: テーブルスペースで使用されるファイルに関する情報。
- `tablespaces`: アクティブなテーブルスペースに関する情報。
- `triggers`: トリガーに関する情報。
- `view_routine_usage`: ビューとビューで使用されるストアドファンクション間の依存性に関する情報。
- `view_table_usage`: ビューとその基礎となるテーブル間の依存性を追跡するために使用されます。

データディクショナリテーブルは非表示です。 `SELECT` では読み取ることができず、`SHOW TABLES` の出力には表示されず、`INFORMATION_SCHEMA.TABLES` テーブルにもリストされません。 ただし、ほとんどの場合、クエリー可能な対応する `INFORMATION_SCHEMA` テーブルがあります。 概念的には、`INFORMATION_SCHEMA` は、MySQL がデータディクショナリメタデータを公開するためのビューを提供します。 たとえば、`mysql.schemata` テーブルから直接選択することはできません:

```
mysql> SELECT * FROM mysql.schemata;
ERROR 3554 (HY000): Access to data dictionary table 'mysql.schemata' is rejected.
```

かわりに、対応する `INFORMATION_SCHEMA` テーブルからその情報を選択します:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.SCHEMATA\G
***** 1. row *****
      CATALOG_NAME: def
      SCHEMA_NAME: mysql
DEFAULT_CHARACTER_SET_NAME: utf8mb4
DEFAULT_COLLATION_NAME: utf8mb4_0900_ai_ci
      SQL_PATH: NULL
      DEFAULT_ENCRYPTION: NO
***** 2. row *****
      CATALOG_NAME: def
      SCHEMA_NAME: information_schema
DEFAULT_CHARACTER_SET_NAME: utf8
DEFAULT_COLLATION_NAME: utf8_general_ci
      SQL_PATH: NULL
      DEFAULT_ENCRYPTION: NO
...

```

`mysql.indexes` に正確に対応する `INFORMATION_SCHEMA` テーブルはありませんが、`INFORMATION_SCHEMA.STATISTICS` には同じ情報の多くが含まれています。

現時点では、`mysql.foreign_keys`、`mysql.foreign_key_column_usage` に正確に対応する `INFORMATION_SCHEMA` テーブルはありません。外部キー情報を取得する標準の SQL 方法は、`INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS` および `KEY_COLUMN_USAGE` テーブルを使用することです。これらのテーブルは、`foreign_keys`、`foreign_key_column_usage`、およびその他のデータディクショナリテーブルのビューとして実装されるようになりました。

MySQL 8.0 より前の一部のシステムテーブルはデータディクショナリテーブルに置き換えられ、`mysql` システムスキーマには存在しなくなりました:

- `events` データディクショナリテーブルは、MySQL 8.0 より前の `event` テーブルよりも優先されます。
- `parameters` と `routines` のデータディクショナリテーブルは、MySQL 8.0 より前の `proc` テーブルよりも優先されま

システムテーブルの付与

これらのシステムテーブルには、ユーザーアカウントおよびそのアカウントが保持する権限に関する付与情報が含まれています。これらのテーブルの構造、内容および目的の詳細は、[セクション6.2.3「付与テーブル」](#)を参照してください。

MySQL 8.0 の時点では、付与テーブルは `InnoDB` (トランザクション) テーブルです。以前は、これらは `MyISAM` (非トランザクション) テーブルでした。権限テーブルストレージエンジンの変更は、`CREATE USER` や `GRANT` などのアカウント管理ステートメントの動作に付随する MySQL 8.0 の変更の基礎となります。以前は、複数のユーザーを指定したアカウント管理ステートメントは、一部のユーザーでは成功し、他のユーザーでは失敗していました。ステートメントはトランザクションになり、すべての名前付きユーザーに対して成功するか、ロールバックされ、エラーが発生した場合は何の効果もありません。

注記

MySQL が古いバージョンからアップグレードされたものの、付与テーブルが `MyISAM` から `InnoDB` にアップグレードされていない場合、サーバーはそれらを読み取り専用とみなし、アカウント管理ステートメントはエラーを生成します。アップグレードの手順については、[セクション2.11「MySQL のアップグレード」](#)を参照してください。

- `user`: ユーザーアカウント、グローバル権限およびその他の非権限カラム。
- `global_grants`: ユーザーへの動的グローバル権限の割当て。[静的権限と動的権限](#)を参照してください。
- `db`: Database-level privileges.
- `tables_priv`: Table-level privileges.

- `columns_priv`: Column-level privileges.
- `procs_priv`: ストアドプロシージャおよびファンクション権限。
- `proxies_priv`: Proxy-user privileges.
- `default_roles`: このテーブルは、ユーザーが `SET ROLE DEFAULT` に接続して認証または実行した後にアクティブ化されるデフォルトのロールを示しています。
- `role_edges`: このテーブルは、ロールのサブグラフのエッジを示しています。
特定の `user` テーブルの行がユーザーアカウントまたはロールを参照する場合があります。サーバーは、認証 ID 間の関係に関する情報を `role_edges` テーブルで調べることで、行がユーザーアカウント、ロールまたはその両方を表すかどうかを調べることができます。
- `password_history`: パスワード変更に関する情報。

オブジェクト情報システムテーブル

次のシステムテーブルには、コンポーネント、ユーザー定義関数、およびサーバー側プラグインに関する情報が含まれています:

- `component`: `INSTALL COMPONENT` を使用してインストールされたサーバーコンポーネントのレジストリ。このテーブルにリストされているコンポーネントは、サーバーの起動シーケンス中にローダーサービスによってインストールされます。 [セクション5.5.1「コンポーネントのインストールおよびアンインストール」](#) を参照してください。
- `func`: `CREATE FUNCTION` を使用してインストールされたユーザー定義関数 (UDF) のレジストリ。通常の起動シーケンスでは、サーバーはこのテーブルに登録されている UDF をロードします。 `--skip-grant-tables` オプションを使用してサーバーを起動した場合、テーブルに登録されている UDF はロードされず、使用できません。 [セクション5.7.1「ユーザー定義関数のインストールおよびアンインストール」](#) を参照してください。

注記

`mysql.func` システムテーブルと同様に、パフォーマンススキーマ `user_defined_functions` テーブルには、`CREATE FUNCTION` を使用してインストールされた UDF が一覧表示されます。 `mysql.func` テーブルとは異なり、`user_defined_functions` テーブルには、サーバーコンポーネントまたはプラグインによって自動的にインストールされる UDF もリストされます。この違いにより、どの UDF がインストールされているかを `mysql.func` より `user_defined_functions` で確認することをお勧めします。 [セクション27.12.19.12「user_defined_functions テーブル」](#) を参照してください。

- `plugin`: `INSTALL PLUGIN` を使用してインストールされたサーバー側プラグインのレジストリ。通常の起動シーケンスでは、サーバーはこのテーブルに登録されているプラグインをロードします。サーバーが `--skip-grant-tables` オプションで起動された場合、テーブルに登録されているプラグインはロードされず、使用できません。 [セクション5.6.1「プラグインのインストールおよびアンインストール」](#) を参照してください。

ログシステムテーブル

サーバーは、ロギングに次のシステムテーブルを使用します:

- `general_log`: 一般クエリーログテーブル。
- `slow_log`: スロークエリーログテーブル。

ログテーブルは `CSV` ストレージエンジンを使用します。

詳細は、[セクション5.4「MySQL Server ログ」](#) を参照してください。

サーバー側ヘルプシステムのテーブル

次のシステムテーブルには、サーバー側のヘルプ情報が含まれています:

- `help_category`: ヘルプカテゴリに関する情報。
- `help_keyword`: ヘルプトピックに関連付けられたキーワード。
- `help_relation`: ヘルプキーワードとトピック間のマッピング。
- `help_topic`: ヘルプトピックの内容。

詳細は、[セクション5.1.17「サーバー側ヘルプのサポート」](#)を参照してください。

タイムゾーンシステムテーブル

次のシステムテーブルには、タイムゾーン情報が含まれています:

- `time_zone`: タイムゾーン ID、およびうるう秒を使用するかどうか。
- `time_zone_leap_second`: うるう秒が発生したとき。
- `time_zone_name`: タイムゾーン ID と名前のマッピング。
- `time_zone_transition`, `time_zone_transition_type`: タイムゾーンの説明。

詳細は、[セクション5.1.15「MySQL Server でのタイムゾーンのサポート」](#)を参照してください。

レプリケーションシステムテーブル

サーバーは、次のシステムテーブルを使用してレプリケーションをサポートします:

- `gtid_executed`: GTID 値を格納するためのテーブル。 [mysql.gtid_executed テーブル](#)を参照してください。
- `ndb_binlog_index`: NDB Cluster レプリケーションのバイナリログ情報。このテーブルは、サーバーが NDBCLUSTER サポートで構築されている場合にのみ作成されます。 [セクション23.6.4「NDB Cluster レプリケーションスキーマおよびテーブル」](#)を参照してください。
- `slave_master_info`, `slave_relay_log_info`, `slave_worker_info`: 複製サーバーに複製情報を格納するために使用されます。 [セクション17.2.4「リレーログおよびレプリケーションメタデータリポジトリ」](#)を参照してください。

リストされているすべてのテーブルは、InnoDB ストレージエンジンを使用します。

オプティマイザシステムテーブル

これらのシステムテーブルはオプティマイザで使用されます:

- `innodb_index_stats`, `innodb_table_stats`: InnoDB 永続オプティマイザ統計に使用されます。 [セクション15.8.10.1「永続的オプティマイザ統計のパラメータの構成」](#)を参照してください。
- `server_cost`, `engine_cost`: オプティマイザのコストモデルでは、クエリーの実行中に発生する操作に関するコスト見積り情報を含むテーブルが使用されます。`server_cost` には、一般的なサーバー操作のオプティマイザコストの見積りが含まれています。`engine_cost` には、特定のストレージエンジンに固有の操作の見積りが含まれています。 [セクション8.9.5「オプティマイザコストモデル」](#)を参照してください。

その他のシステムテーブル

その他のシステムテーブルは、前述のカテゴリに適合しません:

- `audit_log_filter`, `audit_log_user`: MySQL Enterprise Audit がインストールされている場合、これらのテーブルは監査ログフィルタ定義およびユーザーアカウントの永続記憶域を提供します。 [監査ログテーブル](#)を参照してください。
- `firewall_group_allowlist`, `firewall_groups`, `firewall_membership`, `firewall_users`, `firewall_whitelist`: MySQL Enterprise Firewall がインストールされている場合、これらのテーブルはファイアウォールで使用される情報の永続記憶域を提供します。 [セクション6.4.7「MySQL Enterprise Firewall」](#)を参照してください。

- [servers: FEDERATED](#) ストレージエンジンによって使用されます。 [セクション16.8.2.2「CREATE SERVER を使用した FEDERATED テーブルの作成」](#) を参照してください。
- [innodb_dynamic_metadata](#): InnoDB ストレージエンジンが、自動インクリメントカウンタ値やインデックスツリー破損フラグなどの高速変更テーブルメタデータを格納するために使用します。 InnoDB システムテーブルスペースに存在していたデータディクショナリバッファテーブルを置き換えます。

5.4 MySQL Server ログ

MySQL Server には、実行中のアクティビティを検出するのに役立ついくつかのログがあります。

ログのタイプ	ログに書き込まれる情報
エラーログ	<code>mysqld</code> の起動、実行、および停止で発生した問題
一般クエリーログ	確立されたクライアント接続およびクライアントから受け取ったステートメント
バイナリログ	データを変更するステートメント (レプリケーションにも使用される)
リレーログ	レプリケーションソースサーバーから受信したデータ変更
スロークエリーログ	実行するのに <code>long_query_time</code> 秒よりも時間を要したクエリー
DDL ログ (メタデータログ)	DDL ステートメントによって実行されたメタデータ操作

デフォルトでは、Windows 上のエラーログを除いてログは有効化されていません。(DDL ログは必要な場合に常に作成され、ユーザーが構成可能なオプションはありません。 [The DDL Log](#) を参照してください。) このあとに続くログに固有のセクションでは、ロギングを有効にするためのサーバーオプションに関する情報を提供します。

デフォルトでは、サーバーは有効化されたすべてのログに対してデータディレクトリ内にファイルを書き込みます。ログをフラッシュすることによって、サーバーがログファイルを閉じて再オープンする (または新しいログファイルに切り替える) ことを強制的に実行できます。ログのフラッシュは、`FLUSH LOGS` ステートメントを発行したり、`mysqldadmin` に `flush-logs` または `refresh` 引数を指定して実行したり、`mysqldump` に `--flush-logs` または `--master-data` オプションを指定して実行したりしたときに実行されます。 [セクション13.7.8.3「FLUSH ステートメント」](#)、[セクション4.5.2「mysqldadmin — A MySQL Server 管理プログラム」](#)、および [セクション4.5.4「mysqldump — データベースバックアッププログラム」](#) を参照してください。さらに、バイナリログは、サイズが `max_binlog_size` システム変数の値に達するとフラッシュされます。

一般クエリーログおよびスロークエリーログを実行時に制御することができます。ロギングを有効化または無効化したり、ログファイル名を変更したりできます。一般クエリーエントリおよびスロークエリーエントリを、ログテーブル、ログファイル、または両方に書き込むようにサーバーに指示することができます。詳細については、[セクション5.4.1「一般クエリーログおよびスロークエリーログの出力先の選択」](#)、[セクション5.4.3「一般クエリーログ」](#)、および [セクション5.4.5「スロークエリーログ」](#) を参照してください。

リレーログはレプリカでのみ使用され、レプリカでも行う必要があるレプリケーションソースサーバーからのデータ変更を保持します。リレーログの内容および構成については、[セクション17.2.4.1「リレーログ」](#) を参照してください。

古いログファイルの有効期限などのログの保守操作についての情報は、[セクション5.4.6「サーバーログの保守」](#) を参照してください。

ログのセキュリティ保護についての情報は、[セクション6.1.2.3「パスワードおよびロギング」](#) を参照してください。

5.4.1 一般クエリーログおよびスロークエリーログの出力先の選択

MySQL Server では、一般クエリーログおよびスロークエリーログが有効になっている場合は、それらのログに書き込まれる出力先を柔軟に制御できます。ログエントリに使用可能な宛先は、ログファイル、または `mysql` システムデータベース内の `general_log` テーブルと `slow_log` テーブルです。ファイル出力またはテーブル出力、あるいはその両方を選択できます。

- [サーバー起動時のログ制御](#)
- [実行時のログ制御](#)
- [ログテーブルの利点と特性](#)

サーバー起動時のログ制御

`log_output` システム変数は、ログ出力の出力先を指定します。この変数を単独で設定してもログは有効になりません。個別に有効にする必要があります。

- 起動時に `log_output` が指定されていない場合、デフォルトのロギング先は `FILE` です。
- 起動時に `log_output` が指定された場合、その値は `TABLE` (テーブルに記録)、`FILE` (ファイルに記録) または `NONE` (テーブルまたはファイルに記録しない) から選択されたカンマ区切りの単語のリストです。`NONE` がある場合は、ほかの指定子よりも優先されます。

`general_log` システム変数は、選択されたログ出力先についての一般クエリログへのロギングを制御します。サーバー起動時に指定された場合、`general_log` は、ログを有効化または無効化するためのオプション引数 1 または 0 を取ります。ファイルロギングについて、デフォルト以外のファイル名を指定するには、`general_log_file` 変数を設定します。同様に、`slow_query_log` 変数は、選択された出力先についてのスロークエリログへのロギングを制御し、`slow_query_log_file` の設定は、ファイルロギングのためのファイル名を指定します。いずれかのログが有効化された場合、サーバーは対応するログファイルを開き、ログファイルに起動メッセージを書き込みます。ただし、`FILE` ログの出力先が選択されないかぎり、ファイルに対するそれ以上のクエリーのロギングは実行されません。

例:

- 一般クエリログエントリをログテーブルおよびログファイルに書き込むには、`--log_output=TABLE,FILE` を使用してログの宛先と `--general_log` の両方を選択し、一般クエリログを有効にします。
- 一般クエリログエントリとスロークエリログエントリのみをログテーブルに書き込むには、`--log_output=TABLE` を使用してログの宛先としてテーブルを選択し、`--general_log` と `--slow_query_log` を使用して両方のログを有効にします。
- スロークエリログエントリをログファイルにのみ書き込むには、`--log_output=FILE` を使用してログの宛先としてファイルを選択し、`--slow_query_log` を使用してスロークエリログを有効にします。この場合、デフォルトのログ保存先は `FILE` であるため、`log_output` 設定を省略できます。

実行時のログ制御

ログテーブルおよびファイルに関連付けられたシステム変数によって、ロギングへの実行時制御が可能になります。

- `log_output` 変数は、現在のロギング先を示します。出力先を変更するために、これを実行時に変更できます。
- `general_log` および `slow_query_log` 変数は、一般クエリログとスロークエリログが有効 (`ON`) か無効 (`OFF`) かを示します。これらの変数を実行時に設定して、ログを有効化するかどうかを制御することができます。
- `general_log_file` および `slow_query_log_file` 変数は、一般クエリログおよびスロークエリログファイルの名前を示します。これらの変数をサーバー起動時または実行時に設定して、ログファイルの名前を変更することができます。
- 現在のセッションの一般クエリログを無効または有効にするには、セッション `sql_log_off` 変数を `ON` または `OFF` に設定します。(これは、一般クエリログ自体が有効になっていることを前提としています。)

ログテーブルの利点と特性

ログ出力用のテーブルを使用することには、次の利点があります。

- ログエントリが標準形式を持ちます。ログテーブルの現在の構造を表示するには、次のステートメントを使用します。

```
SHOW CREATE TABLE mysql.general_log;
```

```
SHOW CREATE TABLE mysql.slow_log;
```

- ログ内容に SQL ステートメントを使用してアクセスできます。これにより、特定の基準を満たすログエントリのみを選択するクエリを使用することができます。たとえば、特定のクライアントに関連したログ内容を選択するには (そのクライアントからの問題のあるクエリを特定するために役立つことがあります)、ログファイルよりもログテーブルを使用して行う方が簡単です。
- サーバーに接続してクエリを発行できるすべてのクライアントを介して、ログにリモートからアクセスできます (クライアントが適切なログテーブル権限を持つ場合)。サーバーホストにログインしてファイルシステムに直接アクセスする必要はありません。

ログテーブルの実装には次の特徴があります。

- 一般的に、ログテーブルの主な目的は、サーバーのランタイム実行を観察するユーザーにインタフェースを提供することで、サーバーのランタイム実行を妨げません。
- **CREATE TABLE**、**ALTER TABLE**、および **DROP TABLE** はログテーブル上での有効な操作です。**ALTER TABLE** および **DROP TABLE** の場合、ログテーブルは使用中であってはならず、あとで説明するように無効にする必要があります。
- デフォルトでは、ログテーブルは、カンマ区切り値形式でデータを書き込む **CSV** ストレージエンジンを使用します。ログテーブルデータを含む **CSV** ファイルにアクセスするユーザーの場合、CSV 入力を処理できるスプレッドシートなどのほかのプログラムにファイルを簡単にインポートできます。

ログテーブルは、**MyISAM** ストレージエンジンを使用するように変更することができます。使用中のログテーブルを変更するために、**ALTER TABLE** を使用することはできません。ログを最初に無効にする必要があります。**CSV** または **MyISAM** 以外のすべてのエンジンは、ログテーブルについて適正ではありません。

ログテーブルおよび「開いているファイルが多すぎます」エラー。ログの宛先として **TABLE** を選択し、ログテーブルが **CSV** ストレージエンジンを使用している場合、一般クエリログまたはスロークエリログを実行時に繰り返し無効にして有効にすると、**CSV** ファイル用に多数のオープンファイル記述子が生成され、「開いているファイルが多すぎます」エラーが発生する可能性があります。この問題を回避するには、**FLUSH TABLES** を実行するか、**open_files_limit** の値が **table_open_cache_instances** の値より大きいことを確認します。

- ログテーブルを変更 (または削除) できるようにロギングを無効化するには、次の方法を使用することができます。この例では一般クエリログを使用しており、スロークエリログについての手順も類似していますが、**slow_log** テーブルおよび **slow_query_log** システム変数を使用します。

```
SET @old_log_state = @@GLOBAL.general_log;
SET GLOBAL general_log = 'OFF';
ALTER TABLE mysql.general_log ENGINE = MyISAM;
SET GLOBAL general_log = @old_log_state;
```

- **TRUNCATE TABLE** は、ログテーブル上での有効な操作です。ログエントリを期限切れにするために使用できます。
- **RENAME TABLE** は、ログテーブル上での有効な操作です。次の方法を使用して、(たとえばログローテーションを実行するために) ログテーブルを原子的に名前変更できます。

```
USE mysql;
DROP TABLE IF EXISTS general_log2;
CREATE TABLE general_log2 LIKE general_log;
RENAME TABLE general_log TO general_log_backup, general_log2 TO general_log;
```

- **CHECK TABLE** は、ログテーブル上での有効な操作です。
- **LOCK TABLES** をログテーブル上で使用することはできません。
- **INSERT**、**DELETE**、および **UPDATE** をログテーブル上で使用することはできません。これらの操作は、サーバー自体の内部でのみ許可されます。
- **FLUSH TABLES WITH READ LOCK** および **read_only** システム変数の状態は、ログテーブルには影響しません。サーバーは常にログテーブルに書き込むことができます。
- ログテーブルに書き込まれたエントリはバイナリログに書き込まれないため、複製に複製されません。

- ログテーブルまたはログファイルをフラッシュするには、[FLUSH TABLES](#) または [FLUSH LOGS](#) をそれぞれ使用します。
- ログテーブルのパーティション化は許可されません。
- `mysqldump` ダンプには、ダンプファイルのリロード後に欠落しないように、これらのテーブルを再作成するステートメントが含まれています。ログテーブルの内容はダンプされません。

5.4.2 エラーログ

このセクションでは、エラーログに診断メッセージをロギングするように MySQL サーバーを構成する方法について説明します。エラーメッセージの文字セットおよび言語の選択の詳細は、[セクション10.6「エラーメッセージ文字セット」](#) および [セクション10.12「エラーメッセージ言語の設定」](#) を参照してください。

エラーログには、`mysqld` の起動時間と停止時間の記録が含まれます。また、サーバーの起動および停止中、およびサーバーの実行中に発生するエラー、警告、ノートなどの診断メッセージも含まれます。たとえば、テーブルを自動的にチェックまたは修復する必要があることに `mysqld` が気付いた場合、エラーログにメッセージが書き込まれます。

エラーログの構成によっては、エラーメッセージがパフォーマンススキーマ `error_log` テーブルに移入され、ログに SQL インタフェースが提供され、その内容のクエリーが可能になる場合もあります。[セクション27.12.19.1「error_log テーブル」](#) を参照してください。

一部のオペレーティングシステムでは、`mysqld` が異常終了すると、エラーログにスタックトレースが含まれます。トレースを使用して、`mysqld` が終了した場所を判別できます。[セクション5.9「MySQL のデバッグ」](#) を参照してください。

`mysqld` の起動に使用した場合、`mysqld_safe` はエラーログにメッセージを書き込むことがあります。たとえば、`mysqld_safe` は異常な `mysqld` の終了に気付いた場合、`mysqld` を再起動し、`mysqld restarted` メッセージをエラーログに書き込みます。

次の各セクションでは、エラーロギングの構成について説明します。

5.4.2.1 エラーログ構成

MySQL 8.0 では、エラーロギングは [セクション5.5「MySQL のコンポーネント」](#) で説明されている MySQL コンポーネントアーキテクチャを使用します。エラーログサブシステムは、ログイベントのフィルタリングおよび書込みを実行するコンポーネントと、目的のロギング結果を得るために有効にするコンポーネントを構成するシステム変数で構成されます。

このセクションでは、エラーロギングのコンポーネントを選択する方法について説明します。ログフィルタに固有の手順は、[セクション5.4.2.4「エラーログフィルタリングのタイプ」](#) を参照してください。JSON およびシステムログシンクに固有の手順は、[セクション5.4.2.7「JSON 形式でのエラーロギング」](#) および [セクション5.4.2.8「システムログへのエラーロギング」](#) を参照してください。使用可能なすべてのログコンポーネントの詳細は、[セクション5.5.3「エラーログコンポーネント」](#) を参照してください。

コンポーネントベースのエラーロギングには、次の機能があります:

- ログイベントをフィルタコンポーネントでフィルタして、書込み可能な情報に影響を与えることができます。
- ログイベントは、シンク (ライター) コンポーネントによって出力されます。複数のシンクコンポーネントを有効にして、エラーログ出力を複数の宛先に書き込むことができます。
- 組み込みフィルタとシンクコンポーネントを組み合わせ、デフォルトのエラーログ形式を実装します。
- ロード可能なシンクにより、JSON 形式でのロギングが可能になります。
- ロード可能なシンクにより、システムログへのロギングが可能になります。
- システム変数は、有効にするログコンポーネントおよび各コンポーネントの動作を制御します。

`log_error_services` システム変数は、エラーロギングを有効にするログコンポーネントを制御します。変数には、0、1 または多数の要素を含むリストを含めることができます。後者の場合、要素はセミコロンまたは (MySQL

8.0.12 の時点で) カンマで区切り、オプションでスペースを続けることができます。指定された設定にセミicolonとカンマの両方のセパレータを使用することはできません。サーバーはリストされた順序でコンポーネントを実行するため、コンポーネントの順序は重要です。

デフォルトでは、`log_error_services` の値は次のとおりです:

```
mysql> SELECT @@GLOBAL.log_error_services;
+-----+
| @@GLOBAL.log_error_services |
+-----+
| log_filter_internal; log_sink_internal |
+-----+
```

この値は、ログイベントが最初に `log_filter_internal` フィルタコンポーネントを通過し、次に `log_sink_internal` シンクコンポーネントを通過することを示します。どちらも組み込まれています。フィルタは、`log_error_services` 値の後半で指定されたコンポーネントに表示されるログイベントを変更します。シンクは、ログイベントの宛先です。通常、シンクはログイベントを特定の形式のログメッセージに処理し、これらのメッセージをファイルやシステムログなどの関連出力に書き込みます。

注記

`log_error_services` 値の最終コンポーネントをフィルタにすることはできません。イベントに対する変更は出力に影響しないため、これはエラーです:

```
mysql> SET GLOBAL log_error_services = 'log_filter_internal';
ERROR 1231 (42000): Variable 'log_error_services' can't be set to the value
of 'log_filter_internal'
```

問題を修正するには、値の最後にシンクを含めます:

```
mysql> SET GLOBAL log_error_services = 'log_filter_internal; log_sink_internal';
Query OK, 0 rows affected (0.00 sec)
```

`log_filter_internal` と `log_sink_internal` の組合せにより、デフォルトのエラーログのフィルタリングおよび出力動作が実装されます。これらのコンポーネントのアクションは、他のサーバーオプションおよびシステム変数の影響を受けます:

- 出力先は、`--log-error` オプション (および Windows、`--pid-file` および `--console` の場合) によって決まります。これらは、コンソールまたはファイルにエラーメッセージを書き込むかどうか、およびファイルに書き込む場合はエラーログファイル名を決定します。 [セクション5.4.2.2「デフォルトのエラーログ保存先の構成」](#) を参照してください。
- `log_error_verbosity` および `log_error_suppression_list` システム変数は、`log_filter_internal` が許可または抑制するログイベントのタイプに影響します。 [セクション5.4.2.5「優先度ベースのエラーログのフィルタリング \(log_filter_internal\)」](#) を参照してください。

エラーロギングに使用されるログコンポーネントのセットを変更するには、必要に応じてコンポーネントをロードし、コンポーネント固有の構成を実行して、`log_error_services` 値を変更します。ログコンポーネントの追加または削除には、次の制約があります:

- 有効なコンポーネントのリストにログコンポーネントを追加するには:
 - `INSTALL COMPONENT` を使用してコンポーネントをロードします (組み込みまたはすでにロードされている場合を除く)。
 - コンポーネントの初期化を成功させるために設定する必要があるシステム変数がコンポーネントで公開されている場合は、それらの変数に適切な値を割り当てます。
 - コンポーネントを `log_error_services` 値にリストして有効にします。
- コンポーネントを `log_error_services` 値で許可するには、既知である必要があります。コンポーネントは、組み込まれている場合、またはロード可能で `INSTALL COMPONENT` を使用してロードされている場合に認識されます。サーバーの起動時に不明なコンポーネントの名前を指定しようとすると、`log_error_services` がデフォルト値に設定されます。実行時に不明なコンポーネントを指定しようとするとエラーが発生し、`log_error_services` 値は変更されません。

- ログコンポーネントを無効にするには、`log_error_services` 値から削除します。次に、コンポーネントがロード可能で、アンロードする場合は、`UNINSTALL COMPONENT` を使用します。

`UNINSTALL COMPONENT` を使用して、まだ `log_error_services` 値に指定されているロード可能コンポーネントをアンロードしようとする、エラーが発生します。

たとえば、デフォルトのシンク (`log_sink_internal`) のかわりにシステムログシンク (`log_sink_syseventlog`) を使用するには、まずシンクコンポーネントをロードしてから、`log_error_services` 値を変更します:

```
INSTALL COMPONENT 'file://component_log_sink_syseventlog';
SET GLOBAL log_error_services = 'log_filter_internal; log_sink_syseventlog';
```

注記

`INSTALL COMPONENT` を使用してログコンポーネントをロードするために使用する URN は、`file://component` という接頭辞が付いたコンポーネント名です。たとえば、`log_sink_syseventlog` コンポーネントの場合、対応する URN は `file://component_log_sink_syseventlog` です。

複数のログシンクを構成して、複数の宛先に出力を送信できます。デフォルトシンクに加えて (ではなく) システムログシンクを有効にするには、`log_error_services` 値を次のように設定します:

```
SET GLOBAL log_error_services = 'log_filter_internal; log_sink_internal; log_sink_syseventlog';
```

デフォルトのシンクのみを使用してシステムログシンクをアンロードするには、次のステートメントを実行します:

```
SET GLOBAL log_error_services = 'log_filter_internal; log_sink_internal';
UNINSTALL COMPONENT 'file://component_log_sink_syseventlog';
```

サーバーの起動ごとに有効になるようにログコンポーネントを構成するには、次の手順を使用します:

1. コンポーネントがロード可能な場合は、`INSTALL COMPONENT` を使用して実行時にロードします。コンポーネントをロードすると、それが `mysql.component` システムテーブルに登録され、後続の起動時にサーバーによって自動的にロードされます。
2. コンポーネント名を含むように、起動時に `log_error_services` 値を設定します。サーバー `my.cnf` ファイルで値を設定するか、`SET PERSIST` を使用して、実行中の MySQL インスタンスの値を設定し、後続のサーバーの再起動に使用する値も保存します。[セクション13.7.6.1「変数代入の SET 構文」](#) を参照してください。`my.cnf` で設定された値は、次の再起動時に有効になります。`SET PERSIST` を使用して設定された値は、すぐに有効になり、その後の再起動に対して有効になります。

サーバーの起動ごとに、組込みログフィルタおよびシンク (`log_filter_internal`、`log_sink_internal`) に加えて JSON ログシンク (`log_sink_json`) を使用するように構成するとします。JSON シンクがロードされていない場合は、最初にロードします:

```
INSTALL COMPONENT 'file://component_log_sink_json';
```

次に、サーバーの起動時に有効になるように `log_error_services` を設定します。これは、`my.cnf` で設定できます:

```
[mysqld]
log_error_services='log_filter_internal; log_sink_internal; log_sink_json'
```

または、`SET PERSIST` を使用して設定できます:

```
SET PERSIST log_error_services = 'log_filter_internal; log_sink_internal; log_sink_json';
```

`log_error_services` で指定されたコンポーネントの順序は、特にフィルタおよびシンクの相対的な順序に関して重要です。次の `log_error_services` 値について考えてみます:

```
log_filter_internal; log_sink_1; log_sink_2
```

この場合、ログイベントは組込みフィルタ、最初のシンク、次に 2 番目のシンクに渡されます。どちらのシンクも、フィルタリングされたログイベントを受信します。

これを次の `log_error_services` 値と比較します:

```
log_sink_1; log_filter_internal; log_sink_2
```

この場合、ログイベントは最初のシンク、次に組み込みフィルタ、次に 2 番目のシンクに渡されます。最初のシンクはフィルタリングされていないイベントを受信します。2 番目のシンクは、フィルタリングされたイベントを受信します。すべてのログイベントのメッセージを含むログと、ログイベントのサブセットのメッセージのみを含む別のログが必要な場合は、この方法でエラーロギングを構成できます。

注記

有効になっているログコンポーネントにパフォーマンススキーマのサポートを提供するシンクが含まれている場合、エラーログに書き込まれたイベントもパフォーマンススキーマ `error_log` テーブルに書き込まれます。これにより、SQL クエリーを使用したエラーログの内容の調査が可能になります。現在、従来の形式の `log_sink_internal` および JSON 形式の `log_sink_json` シンクでは、この機能がサポートされています。 [セクション 27.12.19.1 「error_log テーブル」](#) を参照してください。

5.4.2.2 デフォルトのエラーログ保存先の構成

このセクションでは、デフォルトのエラーログの宛先 (コンソールまたは名前付きファイル) を構成するサーバーオプションについて説明します。また、デフォルトの宛先に基づいて独自の出力先を構成するログシンクコンポーネントも示します。

この説明では、「console」は標準エラー出力である `stderr` を意味します。標準エラー出力が別の出力先にリダイレクトされていないかぎり、これは端末またはコンソールウィンドウです。

サーバーは、Windows システムと Unix システムでデフォルトのエラーログの保存先を決定するオプションを多少解釈します。ご使用のプラットフォームに適した情報を使用して宛先を構成してください。サーバーは、デフォルトのエラーログ宛先オプションを解釈した後、デフォルトの宛先を示すように `log_error` システム変数を設定します。これは、複数のログシンクコンポーネントがエラーメッセージを書き込む場所に影響します。次の各セクションでは、これらのトピックについて説明します。

- [Windows のデフォルトのエラーログの保存先](#)
- [Unix および Unix-Like システムでのデフォルトのエラーログの保存先](#)
- [デフォルトのエラーログ保存先がログシンクに与える影響](#)

Windows のデフォルトのエラーログの保存先

Windows では、`mysqld` は `--log-error`、`--pid-file` および `--console` オプションを使用して、デフォルトのエラーログの保存先がコンソールかファイルか、およびファイルの場合はファイル名を決定します:

- `--console` が指定されている場合、デフォルトの宛先はコンソールです。(`--console` は、両方が指定されている場合は `--log-error` よりも優先され、`--log-error` に関する次の項目は適用されません。)
- `--log-error` が指定されていない場合、またはファイルに名前を付けずに指定されている場合、`--pid-file` オプションが指定されていないかぎり、デフォルトの宛先はデータディレクトリ内の `host_name.err` という名前のファイルになります。その場合、ファイル名は PID ファイルベース名に接尾辞 `.err` を付けたものになります。
- ファイルに名前を付けるために `--log-error` が指定されている場合、デフォルトの宛先はそのファイルです (名前に接尾辞がない場合は、`.err` 接尾辞が追加されます)。別の場所を指定する絶対パス名が指定されていないかぎり、ファイルの場所はデータディレクトリの下にあります。

デフォルトのエラーログの保存先がコンソールの場合、サーバーは `log_error` システム変数を `stderr` に設定します。それ以外の場合、デフォルトの宛先はファイルで、サーバーは `log_error` をファイル名に設定します。

Unix および Unix-Like システムでのデフォルトのエラーログの保存先

Unix および Unix に似たシステムでは、`mysqld` は `--log-error` オプションを使用して、デフォルトのエラーログの保存先がコンソールかファイルか、およびファイルの場合はファイル名を決定します:

- `--log-error` が指定されていない場合、デフォルトの宛先はコンソールです。
- ファイルに名前を付けずに `--log-error` を指定した場合、デフォルトの宛先はデータディレクトリ内の `host_name.err` という名前のファイルになります。

- ファイルに名前を付けるために `--log-error` が指定されている場合、デフォルトの宛先はそのファイルです (名前に接尾辞がない場合は、`.err` 接尾辞が追加されます)。別の場所を指定する絶対パス名が指定されていないかぎり、ファイルの場所はデータディレクトリの下にあります。
- `[mysqld]`、`[server]` または `[mysqld_safe]` セクションのオプションファイルに `--log-error` が指定されている場合、`mysqld_safe` を使用してサーバーを起動するシステムでは、`mysqld_safe` はそのオプションを検索して使用し、`mysqld` に渡します。

注記

Yum または APT パッケージのインストールでは、サーバー構成ファイルで `log-error=/var/log/mysqld.log` などのオプションを使用して、`/var/log` の下にエラーログファイルの場所を構成するのが一般的です。オプションからパス名を削除すると、データディレクトリ内の `host_name.err` ファイルが使用されます。

デフォルトのエラーログの保存先がコンソールの場合、サーバーは `log_error` システム変数を `stderr` に設定します。それ以外の場合、デフォルトの宛先はファイルで、サーバーは `log_error` をファイル名に設定します。

デフォルトのエラーログ保存先がログシンクに与える影響

サーバーは、エラーログの宛先の構成オプションを解釈した後、デフォルトのエラーログの宛先を示すように `log_error` システム変数を設定します。ログシンクコンポーネントは、`log_error` 値に基づいて独自の出力先を決定するか、`log_error` とは無関係に宛先を決定できます。

`log_error` が `stderr` の場合、デフォルトのエラーログの保存先はコンソールであり、出力先をデフォルトの保存先にするログシンクもコンソールに書き込みます:

- `log_sink_internal`、`log_sink_json`、`log_sink_test`: これらのシンクはコンソールに書き込みます。これは、複数回有効にできる `log_sink_json` などのシンクにも当てはまります。すべてのインスタンスがコンソールに書き込みます。
- `log_sink_syseventlog`: このシンクは、`log_error` の値に関係なく、システムログに書き込みます。

`log_error` が `stderr` でない場合、デフォルトのエラーログの保存先はファイルで、`log_error` はファイル名を示します。出力先のベースとなるログシンクは、そのファイル名に基づいてデフォルトの宛先ベース出力ファイルのネーミングを行います。(シンクは正確にその名前を使用することも、そのようなバリエーションを使用することもできます。) `log_error` 値が `file_name` であるとし、次に、ログシンクは次のような名前を使用します:

- `log_sink_internal`、`log_sink_test`: これらのシンクは `file_name` に書き込みます。
- `log_sink_json`: `log_error_services` 値で指定されたこのシンクの後続インスタンスは、`file_name` という名前のファイルと番号付き `.NN.json` 接尾辞に書き込まれます: `file_name.00.json`、`file_name.01.json` など。
- `log_sink_syseventlog`: このシンクは、`log_error` の値に関係なく、システムログに書き込みます。

5.4.2.3 エラーイベントフィールド

エラーログを対象としたエラーイベントには一連のフィールドが含まれ、各フィールドはキーと値のペアで構成されます。イベントフィールドは、コア、オプションまたはユーザー定義に分類できます:

- コアフィールドは、エラーイベントに対して自動的に設定されます。ただし、すべてのタイプのフィールドと同様に、ログフィルタによってコアフィールドが設定解除される可能性があるため、イベント処理中のイベントでのその存在は保証されません。これが発生した場合、そのフィルタ内の後続の処理およびフィルタの後に実行されるコンポーネント (ログシンクなど) によってフィールドが見つかりません。
- オプションフィールドは通常は存在しませんが、特定のイベントタイプに存在する可能性があります。オプションのフィールドが存在する場合は、必要に応じて追加のイベント情報が表示されます。
- ユーザー定義フィールドは、コアフィールドまたはオプションフィールドとして定義されていない名前を持つ任意のフィールドです。ユーザー定義フィールドは、ログフィルタによって作成されるまで存在しません。

前述の説明で示されているように、特定のフィールドは、最初の場所に存在しなかったか、フィルタによって破棄されたため、イベント処理中に存在しない可能性があります。ログシンクの場合、フィールド休暇欠勤の効果はシンク固有です。たとえば、シンクでログメッセージからフィールドを省略したり、フィールドが欠落していることを示し

たり、デフォルトを置き換えることができます。疑いがある場合はテスト: フィールドの設定を解除するフィルタを使用してから、ログシンクで何が行われるかを確認します。

次の各セクションでは、コアおよびオプションのエラーイベントフィールドについて説明します。個々のログフィルタコンポーネントについては、これらのフィールドにフィルタ固有の追加の考慮事項があるか、ここにリストされていないユーザー定義フィールドがフィルタによって追加される可能性があります。詳細は、特定のフィルタのドキュメントを参照してください。

- [コアエラーイベントフィールド](#)
- [オプションのエラーイベントフィールド](#)

コアエラーイベントフィールド

次のエラーイベントフィールドはコアフィールドです:

- [time](#)
マイクロ秒精度のイベントタイムスタンプ。
- [msg](#)
イベントメッセージ文字列。
- [prio](#)

システム、エラー、警告またはノート/情報イベントを示すイベント優先度。このフィールドは、[syslog](#) の重大度に対応します。次のテーブルに、使用可能な優先度レベルを示します。

イベントタイプ	数値の優先度
システムイベント	0
エラーイベント	1
警告イベント	2
ノート/情報イベント	3

[prio](#) 値は数値です。エラーイベントには、優先度を文字列として表すオプションの [label](#) フィールドが含まれる場合もあります。たとえば、[prio](#) 値が 2 のイベントの [label](#) 値は 'Warning' になります。

フィルタコンポーネントには、優先度に基づいてエラーイベントを含めるか削除できますが、システムイベントは必須であり、削除できません。

一般に、メッセージの優先度は次のように決定されます:

状況またはイベントはアクション可能ですか。

- はい: 状況またはイベントは無視できますか。
 - はい: 優先度は警告です。
 - いいえ: 優先度はエラーです。
- いいえ: 状況またはイベントは必須ですか。
 - はい: 優先度は system です。
 - いいえ: 優先度はノート/情報です。
- [err_code](#)
数値としてのイベントエラーコード (1022 など)。
- [err_symbol](#)

文字列としてのイベントエラー記号 ('ER_DUP_KEY'など)。

- [SQL_state](#)

文字列としてのイベント SQLSTATE 値 ('23000'など)。

- [subsystem](#)

イベントが発生したサブシステム。使用可能な値は、[InnoDB](#) (InnoDB ストレージエンジン)、[Repl](#) (レプリケーションサブシステム)、[Server](#) (それ以外) です。

オプションのエラーイベントフィールド

オプションのエラーイベントフィールドは、次のカテゴリに分類されます:

- オペレーティングシステムによって通知されたエラーやエラーラベルなど、エラーに関する追加情報:

- [OS_errno](#)

- オペレーティングシステムのエラー番号。

- [OS_errmsg](#)

- オペレーティングシステムのエラーメッセージ。

- [label](#)

- [prio](#) 値に対応するラベルを文字列として指定します。

- イベントが発生したクライアントの識別:

- [user](#)

- クライアントユーザー。

- [host](#)

- クライアントホスト。

- [thread](#)

- エラーイベントの生成を担当する [mysqld](#) 内のスレッドの ID。この ID は、サーバーのどの部分がイベントを生成したかを示し、一般クエリーログおよびスロークエリーログメッセージ (接続スレッド ID を含む) と一貫性があります。

- [query_id](#)

- クエリー ID。

- デバッグ情報:

- [source_file](#)

- イベントが発生したソースファイル。先頭にパスはありません。

- [source_line](#)

- イベントが発生したソースファイル内の行。

- [function](#)

- イベントが発生した関数。

- [component](#)

イベントが発生したコンポーネントまたはプラグイン。

5.4.2.4 エラーログフィルタリングのタイプ

エラーログ構成には、通常、1つのログフィルタコンポーネントと1つ以上のログシンクコンポーネントが含まれます。エラーログのフィルタリングの場合、MySQL には次のコンポーネントが用意されています:

- `log_filter_internal`: このフィルタコンポーネントは、`log_error_verbosity` および `log_error_suppression_list` システム変数と組み合わせて、ログイベントの優先度およびエラーコードに基づくエラーログのフィルタリングを提供します。`log_filter_internal` は組み込みであり、デフォルトで有効になっています。 [セクション5.4.2.5「優先度ベースのエラーログのフィルタリング \(log_filter_internal\)」](#) を参照してください。
- `log_filter_dragnet`: このフィルタコンポーネントは、`dragnet.log_error_filter_rules` システム変数と組み合わせて、ユーザー指定のルールに基づくエラーログフィルタリングを提供します。 [セクション5.4.2.6「ルールベースのエラーログのフィルタリング \(log_filter_dragnet\)」](#) を参照してください。

5.4.2.5 優先度ベースのエラーログのフィルタリング (log_filter_internal)

`log_filter_internal` ログフィルタコンポーネントは、エラーイベントの優先度およびエラーコードに基づいた単純な形式のログフィルタリングを実装します。`log_filter_internal` がエラーログ用のエラー、警告および情報イベントを許可または抑制する方法に影響を与えるには、`log_error_verbosity` および `log_error_suppression_list` システム変数を設定します。

`log_filter_internal` は組み込みで、デフォルトで有効になっています。このフィルタが無効になっている場合、`log_error_verbosity` および `log_error_suppression_list` は効果がないため、必要に応じて、代わりに別のフィルタサービスを使用してフィルタリングを実行する必要があります(たとえば、`log_filter_dragnet` を使用している場合は個々のフィルタルールを使用します)。フィルタ構成の詳細は、[セクション5.4.2.1「エラーログ構成」](#) を参照してください。

- [冗長性フィルタリング](#)
- [抑制リストのフィルタリング](#)
- [冗長性と抑制リストの相互作用](#)

冗長性フィルタリング

エラーログを対象としたイベントの優先度は、`ERROR`、`WARNING` または `INFORMATION` です。

`log_error_verbosity` システム変数は、次のテーブルに示すように、ログに書き込まれるメッセージに対して許可する優先度に基づいて冗長性を制御します。

log_error_verbosity 値	許可されたメッセージの優先度
1	<code>ERROR</code>
2	<code>ERROR</code> , <code>WARNING</code>
3	<code>ERROR</code> , <code>WARNING</code> , <code>INFORMATION</code>

`log_error_verbosity` が 2 以上の場合、サーバーはステートメントベースのロギングに安全でないステートメントに関するメッセージをログに記録します。値が 3 の場合、サーバーは中断された接続および新しい接続試行のアクセス拒否エラーをログに記録します。 [セクションB.3.2.9「通信エラーおよび中止された接続」](#) を参照してください。

レプリケーションを使用する場合は、ネットワーク障害や再接続に関するメッセージなど、発生していることに関する詳細情報を取得するために、2 以上の `log_error_verbosity` 値をお勧めします。

レプリカで `log_error_verbosity` が 2 以上の場合、レプリカはエラーログにメッセージを出力して、ジョブを開始するバイナリログおよびリレーログ座標、別のリレーログへの切替え時、切断後の再接続時など、そのステータスに関する情報を提供します。

冗長性フィルタリングの対象ではない `SYSTEM` のメッセージ優先度もあります。`log_error_verbosity` の値に関係なく、エラー以外の状況に関するシステムメッセージがエラーログに出力されます。これらのメッセージには、起動メッセージと停止メッセージ、および設定に対する重要な変更が含まれます。

MySQL エラーログでは、システムメッセージに「System」というラベルが付けられます。他のログシンクは同じ規則に従っている場合と従っていない場合があります。結果のログでは、「ノット」や「情報」などの情報優先度レベルに使用されるラベルがシステムメッセージに割り当てられる場合があります。メッセージのラベル付けに基づいてロギングに追加のフィルタリングまたはリダイレクトを適用した場合、システムメッセージはフィルタをオーバーライドしませんが、他のメッセージと同じ方法で処理されます。

抑制リストのフィルタリング

`log_error_suppression_list` システム変数は、エラーログを対象としたイベントに適用され、**WARNING** または **INFORMATION** の優先度で発生した場合に抑制するイベントを指定します。たとえば、特定のタイプの警告が頻繁に発生するが関心がないためにエラーログで望ましくない「ノイズ」とみなされる場合は、抑止できません。`log_error_suppression_list` では、優先度が **ERROR** または **SYSTEM** のメッセージは抑制されません。

`log_error_suppression_list` 値には、抑制しない場合は空の文字列、抑制するエラーコードを示すカンマ区切り値のリストを指定できます。エラーコードはシンボリックまたは数値形式で指定できます。数値コードは、**MY-** 接頭辞の有無にかかわらず指定できます。数値部分の先頭のゼロは重要ではありません。許可されているコード形式の例:

```
ER_SERVER_SHUTDOWN_COMPLETE
MY-000031
000031
MY-31
31
```

読みやすく移植性を高めるために、シンボリック値は数値よりも優先されます。

抑制されるコードはシンボリックまたは数値形式で表現できますが、各コードの数値は許可された範囲内である必要があります:

- 1 から 999: サーバーおよびクライアントによって使用されるグローバルエラーコード。
- 10000 以上: エラーログに書き込まれるサーバーエラーコード (クライアントには送信されません)。

また、指定した各エラーコードは、実際には MySQL で使用する必要があります。許可された範囲内または許可された範囲内にはないが、MySQL で使用されていないコードを指定しようとすると、エラーが発生し、`log_error_suppression_list` 値は変更されません。

エラーコードの範囲と、各範囲内で定義されているエラー記号および番号の詳細は、[セクションB.1「エラーメッセージのソースと要素」](#) および [MySQL 8.0 Error Message Reference](#) を参照してください。

サーバーは、特定のエラーコードのメッセージを異なる優先度で生成できるため、`log_error_suppression_list` にリストされているエラーコードに関連付けられたメッセージの抑制は、その優先度に依存します。変数の値が `'ER_PARSER_TRACE,MY-010001,10002'` であるとします。`log_error_suppression_list` は、これらのコードのメッセージに次の影響を与えます:

- **WARNING** または **INFORMATION** の優先度で生成されたメッセージは抑制されます。
- **ERROR** または **SYSTEM** の優先度で生成されたメッセージは抑制されません。

冗長性と抑制リストの相互作用

`log_error_verbosity` の効果は、`log_error_suppression_list` の効果と結合されます。次の設定でサーバーを起動したとします:

```
[mysqld]
log_error_verbosity=2 # error and warning messages only
log_error_suppression_list='ER_PARSER_TRACE,MY-010001,10002'
```

この場合、`log_error_verbosity` では、優先度が **ERROR** または **WARNING** のメッセージが許可され、優先度が **INFORMATION** のメッセージが破棄されます。破棄されないメッセージのうち、`log_error_suppression_list` は、優先度が **WARNING** のメッセージおよび指定されたエラーコードを破棄します。

注記

例に示されている `log_error_verbosity` 値 2 もデフォルト値であるため、この変数の **INFORMATION** メッセージへの影響は、明示的な設定なしで、デフォルトで説明されている

とおりです。 `log_error_suppression_list` が `INFORMATION` 優先度のメッセージに影響を与えるようにするには、`log_error_verbosity` を 3 に設定する必要があります。

次の設定でサーバーが起動したとします:

```
[mysqld]
log_error_verbosity=1 # error messages only
```

この場合、`log_error_verbosity` では優先度が `ERROR` のメッセージが許可され、優先度が `WARNING` または `INFORMATION` のメッセージが破棄されます。抑制される可能性のあるすべてのエラーコードが `log_error_verbosity` 設定のためにすでに破棄されているため、`log_error_suppression_list` の設定は効果がありません。

5.4.2.6 ルールベースのエラーログのフィルタリング (log_filter_dragnet)

`log_filter_dragnet` のログフィルタコンポーネントを使用すると、ユーザー定義ルールに基づいてログをフィルタリングできます。

`log_filter_dragnet` フィルタを有効にするには、まずフィルタコンポーネントをロードしてから、`log_error_services` 値を変更します。次の例では、組込みログシンクと組み合わせて `log_filter_dragnet` を有効にします:

```
INSTALL COMPONENT 'file://component_log_filter_dragnet';
SET GLOBAL log_error_services = 'log_filter_dragnet; log_sink_internal';
```

`log_error_services` がサーバーの起動時に有効になるように設定するには、[セクション5.4.2.1「エラーログ構成」](#)の手順を使用します。これらの手順は、ほかのエラーロギングシステム変数にも適用されます。

`log_filter_dragnet` が有効になっている場合は、`dragnet.log_error_filter_rules` システム変数を設定してフィルタルールを定義します。ルールセットはゼロ以上のルールで構成され、各ルールはピリオド (.) 文字で終了する IF ステートメントです。変数値が空 (ゼロルール) の場合、フィルタリングは行われません。

例 1 このルールセットは情報イベントを削除し、他のイベントの場合は `source_line` フィールドを削除します:

```
SET GLOBAL dragnet.log_error_filter_rules =
'IF prio>=INFORMATION THEN drop. IF EXISTS source_line THEN unset source_line.;
```

この効果は、`log_error_verbosity=2` の設定を使用して `log_sink_internal` フィルタによって実行されるフィルタリングに似ています。

読みやすくするために、ルールを別々の行にリストすることをお勧めします。例:

```
SET GLOBAL dragnet.log_error_filter_rules = '
  IF prio>=INFORMATION THEN drop.
  IF EXISTS source_line THEN unset source_line.
';
```

例 2: このルールは、情報イベントを 60 秒当たり 1 つ以下に制限します:

```
SET GLOBAL dragnet.log_error_filter_rules =
'IF prio>=INFORMATION THEN throttle 1/60.;
```

必要に応じてフィルタリング構成を設定したら、`SET GLOBAL` ではなく `SET PERSIST` を使用して `dragnet.log_error_filter_rules` を割り当て、サーバーの再起動後も設定を保持することを検討してください。または、サーバーオプションファイルに設定を追加します。

フィルタリング言語の使用を停止するには、まずエラーロギングコンポーネントのセットから削除します。通常、これはフィルタコンポーネントではなく別のフィルタコンポーネントを使用することを意味します。例:

```
SET GLOBAL log_error_services = 'log_filter_internal; log_sink_internal';
```

ここでも、`SET GLOBAL` ではなく `SET PERSIST` を使用して、サーバーの再起動後も設定を保持することを検討してください。

次に、フィルタ `log_filter_dragnet` コンポーネントをアンインストールします:

```
UNINSTALL COMPONENT 'file://component_log_filter_dragnet';
```

次の各セクションでは、`log_filter_dragnet` 操作の側面について詳細に説明します:

- [log_filter_dragnet Rule Language の文法](#)
- [log_filter_dragnet ルールのアクション](#)
- [log_filter_dragnet ルールのフィールドリファレンス](#)

log_filter_dragnet Rule Language の文法

次の文法では、`log_filter_dragnet` フィルタールールの言語を定義します。各ルールは、ピリオド (.) 文字で終了する IF ステートメントです。言語では大文字と小文字は区別されません。

```
rule:
  IF condition THEN action
  [ELSEIF condition THEN action] ...
  [ELSE action]
  .

condition: {
  field comparator value
  | [NOT] EXISTS field
  | condition {AND | OR} condition
}

action: {
  drop
  | throttle {count | count / window_size}
  | set field [:= | =] value
  | unset [field]
}

field: {
  core_field
  | optional_field
  | user_defined_field
}

core_field: {
  time
  | msg
  | prio
  | err_code
  | err_symbol
  | SQL_state
  | subsystem
}

optional_field: {
  OS_errno
  | OS_errmsg
  | label
  | user
  | host
  | thread
  | query_id
  | source_file
  | source_line
  | function
  | component
}

user_defined_field:
  sequence of characters in [a-zA-Z0-9_] class

comparator: {== | != | <> | >= | => | <= | =< | < | >}

value: {
  string_literal
  | integer_literal
  | float_literal
  | error_symbol
  | priority
}

count: integer_literal
```

```

window_size: integer_literal

string_literal:
  sequence of characters quoted as '...' or "..."

integer_literal:
  sequence of characters in [0-9] class

float_literal:
  integer_literal[.integer_literal]

error_symbol:
  valid MySQL error symbol such as ER_ACCESS_DENIED_ERROR or ER_STARTUP

priority: {
  ERROR
  | WARNING
  | INFORMATION
}

```

単純条件は、フィールドと値またはテストフィールドの存在を比較します。より複雑な条件を作成するには、**AND** および **OR** 演算子を使用します。両方の演算子の優先順位は同じで、左から右に評価されます。

文字列内の文字をエスケープするには、その前にバックスラッシュ (\) を付けます。バックスラッシュ自体または文字列引用符文字を含めるにはバックスラッシュが必要です。他の文字の場合はオプションです。

便宜上、**log_filter_dragnet** では、特定のフィールドとの比較にシンボリック名がサポートされています。読みやすく移植性を高めるために、数値よりシンボリック値をお薦めします (該当する場合)。

- イベント優先度の値 1、2 および 3 は、**ERROR**、**WARNING** および **INFORMATION** として指定できます。優先度記号は、**prio** フィールドとの比較でのみ認識されます。これらの比較は同等です:

```

IF prio == INFORMATION THEN ...
IF prio == 3 THEN ...

```

- エラーコードは、数値形式または対応するエラー記号として指定できます。たとえば、**ER_STARTUP** はエラー **1408** のシンボリック名であるため、次の比較は同等です:

```

IF err_code == ER_STARTUP THEN ...
IF err_code == 1408 THEN ...

```

エラー記号は、**err_code** フィールドおよびユーザー定義フィールドとの比較でのみ認識されます。

特定のエラーコード番号に対応するエラー記号を検索するには、次のいずれかの方法を使用します:

- Server Error Message Reference** でサーバーエラーのリストを確認します。
- perrow** コマンドを使用します。エラー番号引数を指定すると、**perrow** ではエラーに関する情報 (その記号を含む) が表示されます。

エラー番号を持つルールセットが次のようになっているとします:

```

IF err_code == 10927 OR err_code == 10914 THEN drop.
IF err_code == 1131 THEN drop.

```

perrow を使用して、エラー記号を確認します:

```

shell> perrow 10927 10914 1131
MySQL error code MY-010927 (ER_ACCESS_DENIED_FOR_USER_ACCOUNT_LOCKED):
Access denied for user '%-.48s'@'%-.64s'. Account is locked.
MySQL error code MY-010914 (ER_ABORTING_USER_CONNECTION):
Aborted connection %u to db: '%-.192s' user: '%-.48s' host:
'%.64s' (%-.64s).
MySQL error code MY-001131 (ER_PASSWORD_ANONYMOUS_USER):
You are using MySQL as an anonymous user and anonymous users
are not allowed to change passwords

```

エラー記号を数値に置換すると、ルールセットは次のようになります:

```

IF err_code == ER_ACCESS_DENIED_FOR_USER_ACCOUNT_LOCKED

```

```
OR err_code == ER_ABORTING_USER_CONNECTION THEN drop.
IF err_code == ER_PASSWORD_ANONYMOUS_USER THEN drop.
```

シンボリック名は、文字列フィールドと比較するために引用符で囲まれた文字列として指定できますが、そのような場合、名前は特別な意味を持たない文字列であり、`log_filter_dragnet` はそれらに対応する数値に解決しません。また、誤字は検出されない場合がありますが、サーバーで認識されない引用符で囲まれていないシンボルを使用しようとすると、`SET` でエラーがすぐに発生します。

log_filter_dragnet ルールのアクション

`log_filter_dragnet` では、フィルタルールで次のアクションがサポートされます:

- **drop**: 現在のログイベントを削除します (ログに記録しないでください)。
- **throttle**: レート制限を適用して、特定の条件に一致するイベントのログの冗長性を減らします。引数は、`count` または `count/window_size` の形式でレートを示します。`count` 値は、時間ウィンドウごとにログに記録するイベント発生の許容数を示します。`window_size` 値は秒単位の時間ウィンドウです。省略した場合、デフォルトのウィンドウは 60 秒です。どちらの値も整数リテラルである必要があります。

このルールは、プラグインシャットダウンメッセージを 60 秒当たり 5 回に抑制します:

```
IF err_code == ER_PLUGIN_SHUTTING_DOWN_PLUGIN THEN throttle 5.
```

このルールは、エラーおよび警告を時間当たり 1000 回、情報メッセージを時間当たり 100 回に抑制します:

```
IF prio <= INFORMATION THEN throttle 1000/3600 ELSE throttle 100/3600.
```

- **set**: フィールドに値を割り当てます (フィールドがまだ存在しない場合は存在させます)。後続のルールでは、フィールド名に対する `EXISTS` のテストは `true` で、新しい値は比較条件によってテストできます。
- **unset**: フィールドを破棄します。後続のルールでは、フィールド名に対する `EXISTS` のテストは `false` で、フィールドと任意の値の比較は `false` です。

条件が 1 つのフィールド名のみを参照する特殊なケースでは、`unset` に続くフィールド名はオプションであり、`unset` は名前付きフィールドを破棄します。これらのルールは同等です:

```
IF myfield == 2 THEN unset myfield.
IF myfield == 2 THEN unset.
```

log_filter_dragnet ルールのフィールドリファレンス

`log_filter_dragnet` ルールでは、エラーイベントのコア、オプションおよびユーザー定義フィールドへの参照がサポートされています。

- [コアフィールドリファレンス](#)
- [オプションのフィールドリファレンス](#)
- [ユーザー定義フィールドリファレンス](#)

コアフィールドリファレンス

`log_filter_dragnet Rule Language` の文法の `log_filter_dragnet` 文法では、フィルタルールが認識するコアフィールドに名前が付けられます。これらのフィールドの一般的な説明は、よく理解していることを前提とした [セクション 5.4.2.3 「エラーイベントフィールド」](#) を参照してください。次の備考は、`log_filter_dragnet` ルール内で使用されるコアフィールド参照に特に関連するため、追加情報のみを提供します。

- **prio**

エラー、警告またはノート/情報イベントを示すイベント優先度。比較では、各優先度をシンボリック優先度名または整数リテラルとして指定できます。優先度記号は、`prio` フィールドとの比較でのみ認識されます。これらの比較は同等です:

```
IF prio == INFORMATION THEN ...
```

```
IF prio == 3 THEN ...
```

次のテーブルに、許可される優先度レベルを示します。

イベントタイプ	優先度記号	数値の優先度
エラーイベント	ERROR	1
警告イベント	WARNING	2
ノート/情報イベント	INFORMATION	3

SYSTEM のメッセージ優先度もありますが、システムメッセージはフィルタ処理できず、常にエラーログに書き込まれます。

優先度の値は、高い優先度の値が低いという原則に従います。その逆も同様です。優先度の値は、最も重大なイベント (エラー) の場合は 1 から始まり、優先度の低いイベントの場合は増加します。たとえば、優先度が警告より低いイベントを破棄するには、**WARNING** より高い優先度値をテストします:

```
IF prio > WARNING THEN drop.
```

次の例は、`log_filter_internal` フィルタで許可される各 `log_error_verbosity` 値と同様の効果を得るための `log_filter_dragnet` ルールを示しています:

- エラーのみ (`log_error_verbosity=1`):

```
IF prio > ERROR THEN drop.
```

- エラーおよび警告 (`log_error_verbosity=2`):

```
IF prio > WARNING THEN drop.
```

- エラー、警告およびノート (`log_error_verbosity=3`):

```
IF prio > INFORMATION THEN drop.
```

INFORMATION より大きい `prio` 値がないため、このルールは実際には省略できるため、事実上何も削除されません。

- `err_code`

数値のイベントエラーコード。比較では、テストする値をシンボリックエラー名または整数リテラルとして指定できます。エラー記号は、`err_code` フィールドおよびユーザー定義フィールドとの比較でのみ認識されます。これらの比較は同等です:

```
IF err_code == ER_ACCESS_DENIED_ERROR THEN ...
IF err_code == 1045 THEN ...
```

- `err_symbol`

文字列としてのイベントエラー記号 ('`ER_DUP_KEY`' など)。`log_filter_dragnet` では、文字列として指定された比較値が同等の数値エラーコードに解決されないため、`err_symbol` 値は、フィルタルール比較で使用するよりもログ出力で特定の行を識別することを目的としています。(そのためには、引用符で囲まれていないシンボルを使用してエラーを指定する必要があります。)

オプションのフィールドリファレンス

`log_filter_dragnet Rule Language` の文法 の `log_filter_dragnet` 文法では、フィルタルールで認識されるオプションのフィールドに名前が付けられます。これらのフィールドの一般的な説明は、よく理解していることを前提とした [セクション5.4.2.3「エラーイベントフィールド」](#) を参照してください。次の備考は、`log_filter_dragnet` ルール内で使用されるオプションのフィールド参照に特に関連するため、追加情報のみを提供します。

- `label`

`prio` 値に対応するラベルを文字列として指定します。フィルタルールでは、カスタムラベルをサポートするログシグのラベルを変更できます。`log_filter_dragnet` では、文字列として指定された比較値が同等の数値優先度に解決

されないため、`label` の値は、フィルタルール比較で使用するよりもログ出力で特定の行を識別することを目的としています。

- `source_file`

イベントが発生したソースファイル。先頭にパスはありません。たとえば、`sql/gis/distance.cc` ファイルをテストするには、次のように比較を記述します:

```
IF source_file == "distance.cc" THEN ...
```

ユーザー定義フィールドドリファレンス

コアまたはオプションのフィールド名として認識されない `log_filter_dragnet` フィルタルール内のフィールド名は、ユーザー定義フィールドを参照するために使用されます。

5.4.2.7 JSON 形式でのエラーロギング

このセクションでは、組み込みフィルタ、`log_filter_internal` および JSON シンク、`log_sink_json` を使用してエラーロギングを構成し、即時に有効にする方法と、その後のサーバー起動について説明します。エラーロギングの構成に関する一般情報は、[セクション5.4.2.1「エラーログ構成」](#) を参照してください。

JSON シンクを有効にするには、まずシンクコンポーネントをロードしてから、`log_error_services` 値を変更します:

```
INSTALL COMPONENT 'file://component_log_sink_json';
SET PERSIST log_error_services = 'log_filter_internal; log_sink_json';
```

`log_error_services` がサーバーの起動時に有効になるように設定するには、[セクション5.4.2.1「エラーログ構成」](#) の手順を使用します。これらの手順は、ほかのエラーロギングシステム変数にも適用されます。

`log_error_services` 値で `log_sink_json` に複数回名前を付けることができます。たとえば、フィルタ処理されていないイベントをあるインスタンスで書き込み、フィルタ処理されたイベントを別のインスタンスで書き込むには、`log_error_services` を次のように設定します:

```
SET PERSIST log_error_services = 'log_sink_json; log_filter_internal; log_sink_json';
```

JSON シンクは、`log_error` システム変数で指定されたデフォルトのエラーログ宛先に基づいて出力先を決定します。`log_error` がファイルに名前を付ける場合、JSON シンクはそのファイル名と番号付き `.NN.json` 接尾辞に基づいて出力ファイルのネーミングを行い、`NN` は `00` から始まります。たとえば、`log_error` が `file_name` の場合、`log_error_services` 値で指定された `log_sink_json` の連続するインスタンスは `file_name.00.json`、`file_name.01.json` などに書き込まれます。

`log_error` が `stderr` の場合、JSON シンクはコンソールに書き込みます。`log_sink_json` の名前が `log_error_services` 値で複数回指定されている場合は、コンソールに書き込まれるため、役に立たない可能性があります。

5.4.2.8 システムログへのエラーロギング

`mysqld` では、エラーログをシステムログ (Windows の場合はイベントログ、Unix および Unix に似たシステムの場合は `syslog`) に書き込むことができます。

このセクションでは、組み込みフィルタ、`log_filter_internal` およびシステムログシンク、`log_sink_syseventlog` を使用してエラーロギングを構成し、即時に有効にする方法と、その後のサーバー起動について説明します。エラーロギングの構成に関する一般情報は、[セクション5.4.2.1「エラーログ構成」](#) を参照してください。

システムログシンクを有効にするには、まずシンクコンポーネントをロードしてから、`log_error_services` 値を変更します:

```
INSTALL COMPONENT 'file://component_log_sink_syseventlog';
SET PERSIST log_error_services = 'log_filter_internal; log_sink_syseventlog';
```

`log_error_services` がサーバーの起動時に有効になるように設定するには、[セクション5.4.2.1「エラーログ構成」](#) の手順を使用します。これらの手順は、ほかのエラーロギングシステム変数にも適用されます。

注記

MySQL 8.0 構成では、システムログへのエラーロギングを明示的に有効にする必要があります。これは、システムログへのエラーロギングが Windows でデフォルトで有効になってお

り、すべてのプラットフォームでコンポーネントのロードを必要としない MySQL 5.7 以前とは異なります。

システムログへのエラーロギングには、追加のシステム構成が必要な場合があります。使用しているプラットフォームのシステムログドキュメントを参照してください。

Windows では、アプリケーションログ内のイベントログに書き込まれるエラーメッセージには次の特性があります:

- **Error**、**Warning** および **Note** としてマークされたエントリはイベントログに書き込まれますが、個々のストレージエンジンからの情報ステートメントなどのメッセージには書き込まれません。
- イベントログエントリのソースは、MySQL (`syseventlog.tag` が `tag` として定義されている場合は `MySQL-tag`) です。

Unix および Unix に似たシステムでは、システムログへのロギングに `syslog` が使用されます。次のシステム変数は、`syslog` メッセージに影響します:

- `syseventlog.facility`: `syslog` メッセージのデフォルト機能は `daemon` です。別のファシリティを指定するには、この変数を設定します。
- `syseventlog.include_pid`: `syslog` 出力の各行にサーバプロセス ID を含めるかどうか。
- `syseventlog.tag`: この変数は、`syslog` メッセージのサーバ識別子 (`mysqld`) に追加するタグを定義します。定義すると、先頭にハイフンが付いたタグが識別子に追加されます。

注記

MySQL 8.0.13 より前は、`syseventlog.xxx` 変数ではなく `log_syslog_facility`、`log_syslog_include_pid` および `log_syslog_tag` システム変数を使用してください。

MySQL では、起動、停止、設定に対する重要な変更など、エラー以外の状況に関する重要なシステムメッセージにカスタムラベル「System」が使用されます。カスタムラベルをサポートしていないログ (Windows の場合はイベントログ、Unix および Unix に似たシステムの場合は `syslog`) では、情報の優先度レベルに使用されるラベルがシステムメッセージに割り当てられます。ただし、これらのメッセージは、MySQL `log_error_verbosity` 設定で通常は情報レベルのメッセージが除外されている場合でもログに出力されます。

ログシンクがこの方法で「System」ではなく「情報」のラベルにフォールバックする必要があり、ログイベントが MySQL サーバーの外部でさらに処理される場合 (たとえば、`syslog` 構成によってフィルタ処理または転送される場合)、これらのイベントは、セカンダリアプリケーションによって、「System」優先度ではなく「情報」優先度としてデフォルトで処理されることがあります。

5.4.2.9 エラーログ出力形式

各エラーログシンク (ライター) コンポーネントには、宛先へのメッセージの書き込みに使用される特性出力形式がありますが、他の要因がメッセージの内容に影響する場合があります:

- ログシンクで使用可能な情報。シンクコンポーネントの実行前に実行されたログフィルタコンポーネントによってログイベントフィールドが削除された場合、そのフィールドは書き込みできません。ログのフィルタリングの詳細は、[セクション5.4.2.4「エラーログフィルタリングのタイプ」](#)を参照してください。
- ログシンクに関連する情報。すべてのシンクがエラーイベントで使用可能なすべてのフィールドを書き込むわけではありません。
- システム変数はログシンクに影響を与える可能性があります。[エラーログ形式に影響するシステム変数](#)を参照してください。

エラーイベントのフィールドの名前と説明は、[セクション5.4.2.3「エラーイベントフィールド」](#)を参照してください。すべてのログシンクについて、エラーログメッセージに含まれるスレッド ID は、メッセージの書き込みを担当する `mysqld` 内のスレッドのスレッド ID です。この ID は、サーバーのどの部分がメッセージを生成したかを示し、接続スレッド ID を含む一般クエリログおよびスロークエリログメッセージと整合性があります。

- [log_sink_internal](#) 出力形式

- [log_sink_json 出力形式](#)
- [log_sink_syseventlog 出力形式](#)
- [初期起動時のロギング出力形式](#)
- [エラーログ形式に影響するシステム変数](#)

log_sink_internal 出力形式

内部ログシンクは、従来のエラーログ出力を生成します。例:

```
2020-08-06T14:25:02.835618Z 0 [Note] [MY-012487] [InnoDB] DDL log recovery : begin
2020-08-06T14:25:02.936146Z 0 [Warning] [MY-010068] [Server] CA certificate /var/mysql/sslinfo/cacert.pem is self signed.
2020-08-06T14:25:02.963127Z 0 [Note] [MY-010253] [Server] IPv6 is available.
2020-08-06T14:25:03.109022Z 5 [Note] [MY-010051] [Server] Event Scheduler: scheduler thread started with id 5
```

従来の形式のメッセージには、次のフィールドがあります:

```
time thread [label] [err_code] [subsystem] msg
```

[および]の角カッコ文字は、メッセージ形式のリテラル文字です。フィールドがオプションであることを示すものではありません。

label 値は、**prio** エラーイベント優先度フィールドの文字列形式に対応します。

[err_code]および[subsystem]フィールドが MySQL 8.0 に追加されました。古いサーバーによって生成されたログから欠落しています。ログパーサーは、これらのフィールドを、それらを含めるために最近サーバーによって書き込まれたログに対してのみ存在するメッセージテキストの一部として処理できます。パーサーは、[err_code]インジケータの **err_code** 部分を数値ではなく文字列値として扱う必要があります。これは、**MY-012487** や **MY-010051** などの値に数値以外の文字が含まれているためです。

log_sink_json 出力形式

JSON 形式のログシンクは、キーと値のペアを含む JSON オブジェクトとしてメッセージを生成します。例:

```
{
  "prio": 3,
  "err_code": 10051,
  "source_line": 561,
  "source_file": "event_scheduler.cc",
  "function": "run",
  "msg": "Event Scheduler: scheduler thread started with id 5",
  "time": "2020-08-06T14:25:03.109022Z",
  "ts": 1596724012005,
  "thread": 5,
  "err_symbol": "ER_SCHEDULER_STARTED",
  "SQL_state": "HY000",
  "subsystem": "Server",
  "buffered": 1596723903109022,
  "label": "Note"
}
```

表示されるメッセージは、読みやすくするために再フォーマットされています。エラーログに書き込まれたイベントは、行ごとに 1 つのメッセージが表示されます。

ts (タイムスタンプ) キーは MySQL 8.0.20 で追加され、JSON 形式のログシンクに固有です。この値は、エポック ('1970-01-01 00:00:00' UTC) からのミリ秒数を示す整数です。

ts と **buffered** の値は Unix のタイムスタンプ値であり、**FROM_UNIXTIME()** と適切な除数を使用して変換できます:

```
mysql> SET time_zone = '+00:00';
mysql> SELECT FROM_UNIXTIME(1596724012005/1000.0);
+-----+
| FROM_UNIXTIME(1596724012005/1000.0) |
+-----+
| 2020-08-06 14:26:52.0050           |
```

```

+-----+
mysql> SELECT FROM_UNIXTIME(1596723903109022/1000000.0);
+-----+
| FROM_UNIXTIME(1596723903109022/1000000.0) |
+-----+
| 2020-08-06 14:25:03.1090 |
+-----+

```

log_sink_syseventlog 出力形式

システムログシンクは、ローカルプラットフォームで使用されるシステムログ形式に準拠した出力を生成します。

初期起動時のロギング出力形式

サーバーは、起動オプションが処理される前、つまり `log_error_verbosity` や `log_timestamps` システム変数の値などのエラーログ設定がわかっている前、および使用されるログコンポーネントがわかっている前に、いくつかのエラーログメッセージを生成します。サーバーは、起動プロセスの初期段階で生成されるエラーログメッセージを次のように処理します:

- MySQL 8.0.14 より前は、サーバーはデフォルトのタイムスタンプ、書式および冗長性レベルでメッセージを生成し、それらをバッファします。起動オプションが処理され、エラーログ構成がわかった後、サーバーはバッファされたメッセージをフラッシュします。これらの早期メッセージはデフォルトのログ構成を使用するため、起動オプションで指定されたものとは異なる場合があります。また、初期メッセージはデフォルト以外のログシンクにフラッシュされません。たとえば、JSON シンクへのロギングには、JSON 形式ではないため、これらの早期メッセージは含まれません。
- MySQL 8.0.14 では、サーバーは書式設定されたログメッセージではなくロギイベントをバッファします。これにより、設定がわかった後に構成設定をイベントに遡及的に適用でき、フラッシュされたメッセージではデフォルトではなく構成済の設定が使用されます。また、メッセージは、デフォルトのシンクだけでなく、構成されているすべてのシンクにフラッシュされます。

ログ構成がわかる前に致命的エラーが発生し、サーバーを終了する必要がある場合、サーバーはロギングのデフォルトを使用してバッファ済メッセージをフォーマットし、失われないようにします。致命的エラーは発生しませんが、起動オプションを処理する前に起動が過度に遅い場合、サーバーはロギングのデフォルトを使用してバッファされたメッセージを定期的にフォーマットおよびフラッシュし、応答しないようにします。この動作は、デフォルトが使用されるという点で 8.0.14 より前の動作に似ていますが、例外的な条件が発生した場合にメッセージを失うよりも推奨されます。

エラーログ形式に影響するシステム変数

`log_timestamps` システム変数は、エラーログ (および一般的なエラーログファイルとスロークエリログファイル) に書き込まれるメッセージのタイムスタンプのタイムゾーンを制御します。サーバーは、ログシンクに到達する前に `log_timestamps` をエラーイベントに適用するため、すべてのシンクからのエラーメッセージ出力に影響します。

許可される `log_timestamps` 値は、`UTC` (デフォルト) および `SYSTEM` (ローカルシステムのタイムゾーン) です。タイムスタンプは ISO 8601 / RFC 3339 形式を使用して書き込まれます: `YYYY-MM-DDThh:mm:ss.uuuuu` に Zulu 時間 (UTC) または `±hh:mm` (UTC に対するローカルシステムのタイムゾーン調整を示すオフセット) を示す `Z` の末尾の値を加えたもの。例:

```

2020-08-07T15:02:00.832521Z      (UTC)
2020-08-07T10:02:00.832521-05:00 (SYSTEM)

```

5.4.2.10 エラーログファイルのフラッシュおよび名前変更

`FLUSH ERROR LOGS` または `FLUSH LOGS` 文、あるいは `mysqladmin flush-logs` コマンドを使用してエラーログをフラッシュすると、サーバーは書き込まれているエラーログファイルを閉じて再度開きます。エラーログファイルの名前を変更するには、フラッシュする前に手動で変更します。ログをフラッシュすると、元のファイル名の新しいファイルが開きます。たとえば、ログファイル名が `host_name.err` の場合、次のコマンドを使用してファイル名を変更し、新しいファイルを作成します:

```

mv host_name.err host_name.err-old
mysqladmin flush-logs
mv host_name.err-old backup-directory

```

Windows では、`mv` の代わりに `rename` を使用してください。

エラーログファイルの場所がサーバーによって書込み可能でない場合、ログフラッシュ操作は新しいログファイルの作成に失敗します。たとえば、Linux では、サーバーはエラーログを `/var/log/mysqld.log` ファイルに書き込むことができます。このファイルでは、`/var/log` ディレクトリは `root` によって所有され、`mysqld` によって書込み可能ではありません。このケースの処理の詳細は、[セクション5.4.6「サーバーログの保守」](#) を参照してください。

サーバーが指定されたエラーログファイルに書き込んでいない場合、エラーログがフラッシュされてもエラーログファイルの名前は変更されません。

5.4.3 一般クエリーログ

一般クエリーログは、`mysqld` の実行内容の一般的な記録です。サーバーは、クライアントが接続または接続解除したときに情報をこのログに書き込み、クライアントから受け取った各 SQL ステートメントをログに記録します。一般クエリーログは、クライアント側でエラーが疑われるとき、クライアントが `mysqld` に送信した内容を正確に知りたい場合に非常に役立つことがあります。

クライアントが接続するタイミングを示す各行には、`using connection_type` も含まれ、接続の確立に使用されるプロトコルを示します。`connection_type` は、`TCP/IP` (SSL なしで確立された TCP/IP 接続)、`SSL/TLS` (SSL で確立された TCP/IP 接続)、`Socket` (Unix ソケットファイル接続)、`Named Pipe` (Windows 名前付きパイプ接続)、`Shared Memory` (Windows 共有メモリー接続) のいずれかです。

`mysqld` は、ステートメントを受け取った順にクエリーログに書き込みますが、ステートメントが実行された順番とは異なることがあります。このロギング順序はバイナリログの順序とは対照的で、バイナリログの場合、ステートメントはそれが実行されたあと、ロックがリリースされる前に書き込まれます。さらに、クエリーログはデータを選択するだけのステートメントを格納することもあり、そのようなステートメントはバイナリログには一切書き込まれません。

レプリケーションソースサーバーでステートメントベースのバイナリロギングを使用する場合、その複製によって受信されたステートメントは、各複製のクエリーログに書き込まれます。クライアントが `mysqlbinlog` ユーティリティを使用してイベントを読み取り、サーバーに渡すと、ステートメントはソースのクエリーログに書き込まれます。

ただし、行ベースのバイナリロギングを使用する場合、更新は SQL ステートメントではなく行の変更として送信されるため、`binlog_format` が `ROW` の場合、これらのステートメントはクエリーログに書き込まれません。使用されるステートメントによっては、この変数が `MIXED` に設定された場合、所定の更新がクエリーログに書き込まれないこともあります。詳細については、[セクション17.2.1.1「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」](#) を参照してください。

デフォルトでは、一般クエリーログは無効になっています。初期の一般クエリーログ状態を明示的に指定するには、`--general_log={0|1}` を使用します。引数を指定しないか、引数が 1 の場合、`--general_log` によってログが有効になります。引数が 0 の場合、このオプションによってログが無効になります。ログファイル名を指定するには、`--general_log_file=file_name` を使用します。ログの宛先を指定するには、`log_output` システム変数 ([セクション5.4.1「一般クエリーログおよびスロークエリーログの出力先の選択」](#) を参照) を使用します。

注記

TABLE ログの保存先を指定する場合は、**ログテーブルおよび「開いているファイルが多すぎます」エラー** を参照してください。

一般クエリーログファイルの名前を指定しない場合、デフォルト名は `host_name.log` です。サーバーは、別のディレクトリを指定する絶対パス名が指定されないかぎり、データディレクトリ内にファイルを作成します。

実行時に一般クエリーログを無効化または有効化したり、ログファイル名を変更したりするには、グローバルな `general_log` および `general_log_file` システム変数を使用します。`general_log` を 0 (または `OFF`) に設定するとログが無効化され、1 (または `ON`) にすると有効化されます。ログファイルの名前を指定するには、`general_log_file` を指定します。ログファイルがすでに開いている場合、ログファイルが閉じて新しいファイルが開きます。

一般クエリーログが有効になっている場合、サーバーは `log_output` システム変数で指定された宛先に出力を書き込みます。ログを有効にすると、サーバーはログファイルを開き、ログファイルに起動メッセージを書き込みます。ただし、`FILE` ログの出力先が選択されないかぎり、ファイルに対するそれ以上のクエリーのロギングは実行されません。

出力先が `NONE` の場合、一般ログが有効な場合であってもサーバーはクエリーを書き込みません。ログ出力先の値に `FILE` が含まれていない場合、ログファイル名を設定してもロギングへの影響はありません。

サーバー再起動およびログフラッシュを行っても、新しい一般クエリーログファイルは生成されません (ただし、フラッシュではファイルが閉じて再オープンします)。ファイルを名前変更して新しいファイルを作成するには、次のコマンドを使用します。

```
shell> mv host_name.log host_name-old.log
shell> mysqladmin flush-logs
shell> mv host_name-old.log backup-directory
```

Windows では、`mv` の代わりに `rename` を使用してください。

また、ログを無効にすることによって、実行時に一般クエリーログファイルを名前変更することができます。

```
SET GLOBAL general_log = 'OFF';
```

ログを無効にして、ログファイルの名前を外部で (たとえば、コマンドラインから) 変更します。そのあと、ログをふたたび有効にします。

```
SET GLOBAL general_log = 'ON';
```

この方法はすべてのプラットフォームで動作し、サーバー再起動を必要としません。

現在のセッションの一般クエリーロギングを無効または有効にするには、セッション `sql_log_off` 変数を `ON` または `OFF` に設定します。(これは、一般クエリーログ自体が有効になっていることを前提としています。)

一般クエリーログに書き込まれたステートメント内のパスワードは、文字どおりプレーンテキストで発生しないようにサーバーによって書き換えられます。一般クエリーログについてのパスワードの書き換えは、`--log-raw` オプションでサーバーを起動することによって抑制できます。このオプションは、サーバーによって受け取られるステートメントの正確なテキストを表示する際の診断目的で役立つ場合がありますが、セキュリティ上の理由で本番用途では推奨されません。セクション6.1.2.3「パスワードおよびロギング」も参照してください。

パスワードの書き換えの影響は、解析できないステートメント (構文エラーなど) はパスワードがないことがわかっていないため、一般クエリーログに書き込まれないことです。エラーが発生したステートメントを含むすべてのステートメントのロギングが必要なユースケースでは、`--log-raw` オプションを使用する必要があります。これにより、パスワードのリライトもバイパスされることに注意してください。

パスワードの書き換えは、プレーンテキストパスワードが必要な場合にのみ行われます。パスワードハッシュ値を想定する構文を持つステートメントの場合、書き換えは行われません。このような構文に対してプレーンテキストパスワードが誤って指定された場合、パスワードはリライトなしで指定されたとおりにログに記録されます。

`log_timestamps` システム変数は、一般クエリーログファイル (およびスロークエリーログファイルとエラーログ) に書き込まれるメッセージのタイムスタンプのタイムゾーンを制御します。一般クエリーログおよびログテーブルに書き込まれるスロークエリーログメッセージのタイムゾーンには影響しませんが、これらのテーブルから取得された行は、`CONVERT_TZ()` を使用するが、セッションの `time_zone` システム変数を設定することによって、ローカルシステムのタイムゾーンから任意のタイムゾーンに変換できます。

5.4.4 バイナリログ

バイナリログには、テーブル作成操作やテーブルデータへの変更などのデータベース変更を記述する「イベント」が格納されます。また、行ベースのロギングが使用される場合を除き、(一致する行のない `DELETE` などの) 潜在的に変更を行おうとしたステートメントについてのイベントも格納されます。バイナリログには、データを更新した各ステートメントに要した時間に関する情報も格納されます。バイナリログには 2 つの重要な目的があります。

- レプリケーションの場合、レプリケーションソースサーバー上のバイナリログは、レプリカに送信されるデータ変更のレコードを提供します。ソースは、バイナリログに含まれている情報をレプリカに送信します。このレプリカは、それらのトランザクションを再現して、ソースで行われたものと同じデータ変更を行います。セクション17.2「レプリケーションの実装」を参照してください。
- ある特定のデータリカバリ操作には、バイナリログの使用が必要です。バックアップがリストアされたあと、バックアップが実行されたあとに記録されたバイナリログ内のイベントが再実行されます。これらのイベントは、データベースをバックアップのポイントから最新の状態に持って行きます。セクション7.5「Point-in-Time (増分) リカバリ」を参照してください。

バイナリログは、データを変更しない `SELECT` や `SHOW` などのステートメントでは使用されません。(問題となるクエリーを特定するなどのために) すべてのステートメントをログに記録するには、一般クエリーログを使用します。[セクション5.4.3「一般クエリーログ」](#)を参照してください。

バイナリロギングを有効にしてサーバーを実行すると、パフォーマンスがいくらか低下します。ただし、レプリケーションをセットアップでき、リストア操作に対応できるというバイナリログの利点は、一般的にこのパフォーマンスの減少よりも重要です。

バイナリログは、予期しない停止に対して回復性があります。完全なイベントまたはトランザクションのみがログに記録されたり、または読み戻されたりします。

バイナリログに書き込まれたステートメント内のパスワードは、文字どおりプレーンテキストで発生しないようにサーバーによって書き換えられます。[セクション6.1.2.3「パスワードおよびロギング」](#)も参照してください。

MySQL 8.0.14 からは、バイナリログファイルとリレーログファイルを暗号化できるため、これらのファイルとそれらに含まれる潜在的機密データを、外部の攻撃者による誤用から保護したり、格納されているオペレーティングシステムのユーザーによる不正な表示から保護したりできます。MySQL サーバーで暗号化を有効にするには、`binlog_encryption` システム変数を `ON` に設定します。詳細は、[セクション17.3.2「バイナリログファイルとリレーログファイルの暗号化」](#)を参照してください。

次の説明では、バイナリロギングの操作に影響する一部のサーバーオプションおよび変数について記述します。完全なリストについては、[セクション17.1.6.4「バイナリロギングのオプションと変数」](#)を参照してください。

バイナリロギングはデフォルトで有効になっています (`log_bin` システム変数は `ON` に設定されています)。ただし、バイナリロギングがデフォルトで無効になっているが、`--log-bin` オプションを指定して有効にできる場合は、`mysqld` を使用して、`--initialize` または `--initialize-insecure` オプションを指定してデータディレクトリを手動で呼び出してデータディレクトリを初期化することは例外です。

バイナリロギングを無効にするには、起動時に `--skip-log-bin` または `--disable-log-bin` オプションを指定できます。これらのオプションのいずれかが指定され、`--log-bin` も指定されている場合は、後で指定するオプションが優先されません。

`--log-slave-updates` および `--slave-preserve-commit-order` オプションにはバイナリロギングが必要です。バイナリロギングを無効にする場合は、これらのオプションを省略するか、`--log-slave-updates=OFF` および `--skip-slave-preserve-commit-order` を指定します。`--skip-log-bin` または `--disable-log-bin` が指定されている場合、MySQL はこれらのオプションをデフォルトで無効にします。`--log-slave-updates` または `--slave-preserve-commit-order` を `--skip-log-bin` または `--disable-log-bin` とともに指定すると、警告またはエラーメッセージが発行されます。

バイナリログファイルのベース名を指定するには、`--log-bin=[base_name]` オプションを使用します。`--log-bin` オプションを指定しない場合、MySQL はバイナリログファイルのデフォルトのベース名として `binlog` を使用します。以前のリリースとの互換性のために、文字列なしまたは空の文字列を指定して `--log-bin` オプションを指定した場合、ベース名はホストマシンの名前を使用して `host_name-bin` にデフォルト設定されます。ホスト名が変更された場合でも、同じバイナリログファイル名を簡単に使用できるように、ベース名を指定することをお勧めします ([セクション B.3.7「MySQL の既知の問題」](#)を参照)。ログ名に拡張子を指定した場合 (`--log-bin=base_name.extension` など)、拡張子は暗黙的に削除されて無視されます。

`mysqld` は、バイナリログベース名に数値拡張子を付加してバイナリログファイル名を生成します。数値はサーバーが新しいログファイルを作成するたびに増加し、順序付きの一連のファイルが作成されます。サーバーは、次のいずれかのイベントが発生するたびにシリーズに新しいファイルを作成します:

- サーバーが起動または再起動されます
- サーバーはログをフラッシュします。
- 現在のログファイルのサイズが `max_binlog_size` に達しました。

大きなトランザクションを使用する場合、トランザクションがひとまとまりでファイルに書き込まれて、複数のファイルに分割されないため、バイナリログファイルが `max_binlog_size` を超えることがあります。

使用されたバイナリログファイルを追跡するために、`mysqld` はバイナリログファイルの名前を含むバイナリログインデックスファイルも作成します。デフォルトでは、これはバイナリログファイルと同じベース名を持ち、拡張子は `.index` です。バイナリログインデックスファイルの名前は、`--log-bin-index=[file_name]` オプションを使用して変更できます。`mysqld` の動作中にこのファイルを手動で変更しないでください。変更すると、`mysqld` を混乱させることになります。

「バイナリログファイル」という用語は一般的に、データベースイベントを格納する、番号付けされた個々のファイルを指します。「バイナリログ」という用語は、番号付けされたバイナリログファイルとインデックスファイルのセットをひとまとめにしたものを指します。

バイナリログファイルとバイナリログインデックスファイルのデフォルトの場所は、データディレクトリです。ベース名に先頭の絶対パス名を追加して別のディレクトリを指定することで、`--log-bin` オプションを使用して別の場所を指定できます。サーバーは、使用されたバイナリログファイルを追跡するバイナリログインデックスファイルからエントリを読み取るときに、エントリに相対パスが含まれているかどうかを確認します。その場合、パスの相対部分は、`--log-bin` オプションを使用して設定された絶対パスに置き換えられます。バイナリログインデックスファイルに記録された絶対パスは変更されません。このような場合は、新しいパスを使用できるようにインデックスファイルを手動で編集する必要があります。バイナリログファイルのベース名と指定されたパスは、`log_bin_basename` システム変数として使用できます。

MySQL 5.7 では、バイナリロギングが有効になっているときにサーバー ID を指定する必要がありました。そうしないと、サーバーが起動しません。MySQL 8.0 では、`server_id` システム変数はデフォルトで 1 に設定されています。バイナリロギングが有効になっている場合、このデフォルト ID でサーバーを起動できますが、`server_id` システム変数を使用してサーバー ID を明示的に指定しないと、情報メッセージが発行されます。レプリケーショントポロジで 사용되는サーバーの場合、サーバーごとにゼロ以外の一意のサーバー ID を指定する必要があります。

制限付きセッションシステム変数を設定するのに十分な権限を持つクライアント ([セクション 5.1.9.1 「システム変数権限」](#) を参照) は、`SET sql_log_bin=OFF` ステートメントを使用して独自のステートメントのバイナリロギングを無効にできます。

デフォルトでは、サーバーはイベント自体だけでなくイベントの長さもログに記録し、イベントが正しく書き込まれたことを検証するためにこれを使用します。また、`binlog_checksum` システム変数を設定することによって、サーバーがイベントのチェックサムを書き込むようにすることもできます。バイナリログから読み取る場合、ソースはデフォルトでイベント長を使用しますが、使用可能な場合は、`master_verify_checksum` システム変数を有効にすることによってチェックサムを使用するようにできます。レプリカ上のレプリケーション I/O スレッドは、ソースから受信したイベントも検証します。`slave_sql_verify_checksum` システム変数を有効にすることで、レプリケーション SQL スレッドがリレーログから読み取るときにチェックサムを使用できるようにすることができます。

バイナリログに記録されるイベントの形式は、バイナリロギング形式に依存します。3 つの形式タイプがサポートされています: 行ベースロギング、ステートメントベースロギングおよび混合ベースロギング。使用されるバイナリロギング形式は、MySQL のバージョンに依存します。ロギング形式の一般的な説明については、[セクション 5.4.4.1 「バイナリロギング形式」](#) を参照してください。バイナリログの形式についての詳細な説明は、「[MySQL Internals: The Binary Log](#)」を参照してください。

サーバーは、`--binlog-do-db` および `--binlog-ignore-db` オプションを評価する際、それが `--replicate-do-db` および `--replicate-ignore-db` オプションを評価する場合と同じ方法で行います。これを行う方法については、[セクション 17.2.5.1 「データベースレベルレプリケーションオプションおよびバイナリロギングオプションの評価」](#) を参照してください。

レプリカは、デフォルトで有効になっている `log_slave_updates` システム変数を使用して起動されます。つまり、レプリカは、ソースから受信したデータ変更を自身のバイナリログに書き込みます。この設定が機能するには、バイナリログが有効になっている必要があります ([セクション 17.1.6.3 「Replica Server のオプションと変数」](#) を参照)。この設定により、レプリカを他のレプリカのソースとして機能させることができます。

`RESET MASTER` ステートメントですべてのバイナリログファイルを削除したり、`PURGE BINARY LOGS` でそのサブセットを削除したりすることができます。[セクション 13.7.8.6 「RESET ステートメント」](#) および [セクション 13.4.1.1 「PURGE BINARY LOGS ステートメント」](#) を参照してください。

レプリケーションを使用している場合は、レプリカがまだそれらを使用する必要がないことを確認するまで、ソース上の古いバイナリログファイルを削除しないでください。たとえば、レプリカが 3 日を超えて実行されることがない場合、1 日に `mysqladmin flush-logs` をソースで実行してから、3 日以上経過したログを削除できます。ファイルを削除することができますが、`PURGE BINARY LOGS` を使用することが推奨され、この操作によってバイナリログインデックスファイルも安全に更新されます (さらに日付引数を使用できます)。[セクション 13.4.1.1 「PURGE BINARY LOGS ステートメント」](#) を参照してください。

`mysqlbinlog` ユーティリティを使用して、バイナリログファイルの内容を表示できます。これはリカバリ操作のためにログ内のステートメントを再処理するときに役立ちます。たとえば、次のようにしてバイナリログから MySQL Server を更新できます。

```
shell> mysqlbinlog log_file | mysql -h server_name
```

`mysqlbinlog` は、バイナリログファイルと同じ形式で書き込まれるため、レプリカ上のリレーログファイルの内容を表示するためにも使用できます。 `mysqlbinlog` ユーティリティーとその使用方法についての詳細は、[セクション 4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」](#) を参照してください。バイナリログおよびリカバリ操作の詳細については、[セクション 7.5 「Point-in-Time \(増分\) リカバリ」](#) を参照してください。

バイナリロギングは、ステートメントまたはトランザクションの完了後すぐに行われますが、すべてのロックがリリースされるかコミットが実行されるよりも前になります。これにより、ログがコミット順に記録されることが保証されます。

非トランザクションテーブルへの更新は、実行後すぐにバイナリログに格納されます。

コミットなしのトランザクション内では、`InnoDB` テーブルなどのトランザクションテーブルを変更するすべての更新 (`UPDATE`、`DELETE`、`INSERT`) は、サーバーによって `COMMIT` ステートメントが受け取られるまでキャッシュされます。その時点で、`COMMIT` が実行される前に `mysqld` はトランザクション全体をバイナリログに書き込みます。

非トランザクションテーブルへの変更はロールバックできません。ロールバックされるトランザクションに非トランザクションテーブルへの変更が含まれている場合は、非トランザクションテーブルへの変更が確実にレプリケーションされるようにするために、最後に `ROLLBACK` ステートメントを使用してトランザクション全体がログに記録されます。

トランザクションを処理するスレッドが開始すると、スレッドは `binlog_cache_size` のバッファをバッファーステートメントに割り当てます。ステートメントがこれより大きい場合、スレッドはトランザクションを格納する一時ファイルを開きます。スレッドが終了すると、一時ファイルは削除されます。MySQL 8.0.17 から、バイナリログの暗号化がサーバー上でアクティブな場合、一時ファイルは暗号化されます。

`Binlog_cache_use` ステータス変数は、ステートメントを格納するために、このバッファ (および場合によっては一時ファイル) を使用したトランザクションの数を表示します。 `Binlog_cache_disk_use` ステータス変数は、それらのトランザクションのうち、実際に一時ファイルを使用する必要があったものの数を表示します。これらの 2 つの変数は、一時ファイルの使用を避けるために十分な値になるよう `binlog_cache_size` を調整するために使用することができます。

`max_binlog_cache_size` システム変数 (デフォルトは最大値の 4G バイト) を使用して、複数ステートメントのトランザクションをキャッシュするために使用する合計サイズを制限することができます。トランザクションがこのバイト数より大きくなると、失敗してロールバックします。最小値は 4096 です。

バイナリログおよび行ベースのロギングを使用している場合、並列挿入は `CREATE ... SELECT` または `INSERT ... SELECT` ステートメントの一般的な挿入に変換されます。これは、バックアップ操作中にログを適用することでテーブルの正確なコピーを確実に再作成できるようにするために行われます。ステートメントベースのロギングを使用している場合、元のステートメントがログに書き込まれます。

バイナリログ形式には、バックアップからのリカバリに影響する可能性があるいくつかの既知の制約があります。 [セクション 17.5.1 「レプリケーションの機能と問題」](#) を参照してください。

ストアードプログラムのバイナリロギングは、[セクション 25.7 「ストアードプログラムバイナリロギング」](#) で説明しているように行われます。

MySQL 8.0 のバイナリログ形式は、レプリケーションの機能拡張により以前のバージョンの MySQL とは異なることに注意してください。 [セクション 17.5.2 「MySQL バージョン間のレプリケーション互換性」](#) を参照してください。

サーバーがバイナリログへの書き込み、バイナリログファイルのフラッシュ、またはバイナリログのディスクへの同期を実行できない場合、レプリケーションソースサーバー上のバイナリログに一貫性がなくなる可能性があり、レプリカはソースとの同期を失う可能性があります。 `binlog_error_action` システム変数は、バイナリログでこのタイプのエラーが発生した場合に実行されるアクションを制御します。

- デフォルト設定の `ABORT_SERVER` では、サーバーはバイナリロギングを停止してシャットダウンします。この時点で、エラーの原因を特定して修正できます。再起動時に、予期しないサーバーが停止した場合と同様にリカバリが続行されます ([セクション 17.4.2 「レプリカの予期しない停止の処理」](#) を参照)。
- `IGNORE_ERROR` の設定では、古いバージョンの MySQL との下位互換性が提供されます。この設定では、サーバーは進行中のトランザクションを続行し、エラーをログに記録してからバイナリロギングを停止しますが、更新の実行は続行します。この時点で、エラーの原因を特定して修正できます。バイナリロギングを再開するには、`log_bin` を再度有効にする必要があります。これにはサーバーの再起動が必要です。このオプションは、下位互換性が必須で、バイナリログがこの MySQL サーバーインスタンスで必須でない場合にのみ使用します。たとえ

ば、サーバーの断続的な監査またはデバッグにのみバイナリログを使用し、サーバーからのレプリケーションやポイントインタイムリストア操作にはバイナリログを使用しないことがあります。

デフォルトでは、バイナリログは各書き込み (`sync_binlog=1`) でディスクに同期されます。 `sync_binlog` が有効になっておらず、(MySQL サーバーだけでなく) オペレーティングシステムまたはマシンがクラッシュした場合は、バイナリログの最後のステートメントが失われる可能性があります。これを回避するには、`sync_binlog` システム変数を有効にして、すべての `N` コミットグループの後にバイナリログをディスクに同期します。 [セクション5.1.8「サーバーシステム変数」](#) を参照してください。 `sync_binlog` の最も安全な値は 1 (デフォルト) ですが、これも最も遅くなります。

以前の MySQL リリースでは、`sync_binlog` が 1 に設定されていても、クラッシュが発生した場合、テーブルの内容とバイナリログの内容の間に不整合が発生する可能性があります。たとえば、`InnoDB` テーブルを使用していて、MySQL サーバーが `COMMIT` ステートメントを処理する場合、多くの準備済みトランザクションをバイナリログに順番に書き込み、バイナリログを同期してから、トランザクションを `InnoDB` にコミットします。これらの操作の間にサーバーが予期せず終了した場合、トランザクションは再起動時に `InnoDB` によってロールバックされますが、バイナリログにはまだ存在します。このような問題は、XA トランザクションでの双方向コミットの `InnoDB` サポートを有効にすることで、以前のリリースで解決されました。8.0.0 以上では、XA トランザクションでの 2 フェーズコミットの `InnoDB` サポートは常に有効です。

XA トランザクションでの双方向コミットの `InnoDB` サポートにより、バイナリログファイルと `InnoDB` データファイルが確実に同期化されます。ただし、トランザクションをコミットする前にバイナリログと `InnoDB` ログをディスクに同期するように MySQL サーバーを構成する必要があります。 `InnoDB` ログはデフォルトで同期化され、`sync_binlog=1` によってバイナリログが確実に同期化されます。 XA トランザクションおよび `sync_binlog=1` での双方向コミットに対する暗黙的な `InnoDB` サポートの効果は、クラッシュ後の再起動時に、トランザクションのロールバックを実行した後、MySQL サーバーが最新のバイナリログファイルをスキャンしてトランザクション `xid` 値を収集し、バイナリログファイル内の最後の有効な位置を計算することです。次に、MySQL サーバーは、バイナリログに正常に書き込まれた準備済みトランザクションを完了し、バイナリログを最後の有効な位置に切り捨てるように `InnoDB` に指示します。これにより、バイナリログに `InnoDB` テーブルの正確なデータが反映されるため、レプリカはロールバックされたステートメントを受信しないため、ソースとの同期が維持されます。

バイナリログが、必要な長さよりも短いということを、MySQL Server がクラッシュリカバリ中に検出した場合、正常にコミットされた `InnoDB` トランザクションが、少なくとも 1 つバイナリログから欠落していることを示しています。これは `sync_binlog=1` の場合は発生するはずがなく、ディスクまたはファイルシステムは、リクエストされた場合は (されない場合もあります) 実際の同期を実行するため、サーバーは「`The binary log file_name is shorter than its expected size`」というエラーメッセージを出力します。この場合、このバイナリログは正しくなく、レプリケーションはソースデータの新しいスナップショットから再開する必要があります。

次のシステム変数のセッション値はバイナリログに書き込まれ、バイナリログの解析時にレプリカによって適用されます:

- `sql_mode` (`NO_DIR_IN_CREATE` モードがレプリケーションされない場合を除きます。 [セクション17.5.1.39「レプリケーションと変数」](#) を参照してください)
- `foreign_key_checks`
- `unique_checks`
- `character_set_client`
- `collation_connection`
- `collation_database`
- `collation_server`
- `sql_auto_is_null`

5.4.4.1 バイナリロギング形式

サーバーは、いくつかのロギング形式を使用して情報をバイナリログに記録します:

- MySQL のレプリケーション機能は、元はソースからレプリカへの SQL ステートメントの伝播に基づいていました。これはステートメントベースのロギングと呼ばれます。 `--binlog-format=STATEMENT` を指定してサーバーを起動することによって、この形式を使用できます。

- 行ベースロギング (デフォルト) では、ソースは個々のテーブル行がどのように影響を受けるかを示すイベントをバイナリログに書き込みます。--binlog-format=ROW を指定してサーバーを起動することによって、サーバーが行ベースのロギングを使用するようにすることができます。
- 3 番目のオプションである混合形式ロギングも選択できます。混合形式ロギングの場合、デフォルトではステートメントベースのロギングが使用されますが、次に示すような特定の状況ではロギングモードが自動的に行ベースに切り替わります。--binlog-format=MIXED オプションを指定して `mysqld` を開始することによって、MySQL に混合形式ロギングを使用させることができます。

ロギング形式は、使用されているストレージエンジンによって設定または制限される可能性があります。これは、異なるストレージエンジンを使用しているソースとレプリカ間で特定のステートメントをレプリケートするときの問題を排除するのに役立ちます。

ステートメントベースのレプリケーションでは、非決定的なステートメントのレプリケーションに関して問題があることがあります。所定のステートメントがステートメントベースのレプリケーションについて安全かどうかを判断するために、MySQL は、ステートメントベースのロギングを使用してステートメントをレプリケーションできることを保証できるかどうかを判断します。MySQL がこれを保証できない場合、潜在的に信頼できないステートメントにマークを付け、次の警告を発行します。Statement may not be safe to log in statement format.

これらの問題は、代わりに MySQL の行ベースのレプリケーションを使用することで回避できます。

5.4.4.2 バイナリログ形式の設定

MySQL Server を --binlog-format=type で起動することによって、バイナリロギング形式を明示的に選択することができます。type については次の値がサポートされます。

- STATEMENT の場合、ロギングはステートメントに基づきます。
- ROW の場合、ロギングは行に基づきます。これはデフォルトです。
- MIXED の場合、ロギングは混合形式を使用します。

ロギング形式は実行時に切り替えることもできますが、このセクションの後半で説明するように、これを実行できない状況が多数あることに注意してください。binlog_format システム変数のグローバル値を設定して、変更後に接続するクライアントの形式を指定します:

```
mysql> SET GLOBAL binlog_format = 'STATEMENT';
mysql> SET GLOBAL binlog_format = 'ROW';
mysql> SET GLOBAL binlog_format = 'MIXED';
```

個別クライアントは binlog_format のセッション値を設定することによって、クライアント自身のステートメントについてのロギング形式を制御することができます。

```
mysql> SET SESSION binlog_format = 'STATEMENT';
mysql> SET SESSION binlog_format = 'ROW';
mysql> SET SESSION binlog_format = 'MIXED';
```

グローバル binlog_format 値を変更するには、グローバルシステム変数を設定するのに十分な権限が必要です。セッションの binlog_format 値を変更するには、制限付きセッションシステム変数を設定するのに十分な権限が必要です。セクション5.1.9.1「システム変数権限」を参照してください。

クライアントがセッションごとにバイナリロギングを設定することには、いくつかの理由があります。

- 多くの小さい変更をデータベースに行うセッションでは、行ベースのロギングを使用した方がよい場合があります。
- WHERE 句の多くの行に一致する更新を実行するセッションでは、ステートメントベースのロギングを使用する場合があります。これは、少数のステートメントを多数の行よりも効率的に記録できるためです。
- 一部のステートメントでは、ソースで多くの実行時間が必要ですが、変更されるのは少数の行のみです。したがって、行ベースのロギングを使用してそれらの行をレプリケーションする方が有益な場合があります。

レプリケーション形式を実行時に切り替えることができない例外もあります。

- レプリケーション形式は、ストアドファンクションまたはトリガー内から変更できません。

- **NDB** ストレージエンジンが有効な場合
- セッションにオープン一時テーブルがある場合、セッション (`SET @@SESSION.binlog_format`) のレプリケーション形式は変更できません。
- いずれかのレプリケーションチャンネルにオープン一時テーブルがある場合、レプリケーション形式はグローバルに変更できません (`SET @@GLOBAL.binlog_format` または `SET @@PERSIST.binlog_format`)。
- レプリケーションチャンネルアプライヤスレッドが現在実行されている場合、レプリケーション形式はグローバルに変更できません (`SET @@GLOBAL.binlog_format` または `SET @@PERSIST.binlog_format`)。

これらのいずれかの場合 (または現在のレプリケーション形式を設定しようとする場合) にレプリケーション形式を切り替えようとする、エラーになります。ただし、`PERSIST_ONLY` (`SET @@PERSIST_ONLY.binlog_format`) を使用してレプリケーション形式をいつでも変更できます。これは、このアクションによってランタイムグローバルシステム変数の値が変更されず、サーバーの再起動後にのみ有効になるためです。

一時テーブルが存在する場合、実行時にレプリケーション形式を切り替えることはお勧めしません。一時テーブルはステートメントベースレプリケーションの使用時にのみログに記録されますが、行ベースレプリケーションと混在レプリケーションではログに記録されないためです。

レプリケーションの進行中にレプリケーション形式を切り替えると、問題が発生する可能性もあります。それぞれの MySQL Server は、サーバー独自のバイナリロギング形式のみを設定できます (`binlog_format` がグローバルスコープまたはセッションスコープのいずれかで設定される場合にも当てはまります)。つまり、レプリケーションソースサーバーでロギング形式を変更しても、レプリカはそのロギング形式を一致させることはできません。`STATEMENT` モードを使用する場合、`binlog_format` システム変数はレプリケートされません。`MIXED` または `ROW` ロギングモードを使用している場合、レプリケートされますが、レプリカでは無視されます。

レプリカは、`ROW` ロギング形式で受信したバイナリログエントリを、独自のバイナリログで使用するために `STATEMENT` 形式に変換できません。そのため、ソースの場合、レプリカは `ROW` または `MIXED` 形式を使用する必要があります。レプリケーションが `STATEMENT` 形式のレプリカに進行中にソースのバイナリロギング形式を `STATEMENT` から `ROW` または `MIXED` に変更すると、レプリケーションが「行イベント実行中のエラー: 'ステートメントを実行できません: ステートメントは行形式で BINLOG_FORMAT=STATEMENT であるため、バイナリログに書き込めません。」などのエラーで失敗することがあります。ソースがまだ `MIXED` または `ROW` 形式を使用している場合にレプリカのバイナリロギング形式を `STATEMENT` 形式に変更すると、同じタイプのレプリケーションも失敗します。フォーマットを安全に変更するには、レプリケーションを停止し、ソースとレプリカの両方で同じ変更が行われていることを確認する必要があります。

`InnoDB` テーブルを使用中で、トランザクション分離レベルが `READ COMMITTED` または `READ UNCOMMITTED` の場合、行ベースのロギングのみを使用することができます。ロギング形式を `STATEMENT` に変更することは可能ですが、`InnoDB` は挿入を実行できないため、実行時にこれを行うと、非常に速くエラーが発生します。

バイナリログ形式を `ROW` に設定すると、多くの変更は行ベースの形式を使用してバイナリログに書き込まれます。しかし、一部の変更ではステートメントベース形式が使用されます。たとえば、`CREATE TABLE`、`ALTER TABLE`、`DROP TABLE` などのすべての DDL (データ定義言語) ステートメントがこれに該当します。

行ベースのバイナリロギングを使用する場合、`binlog_row_event_max_size` システム変数とそれに対応する起動オプション `--binlog-row-event-max-size` は、行イベントの最大サイズに弱い制限を設定します。デフォルト値は 8192 バイトで、値はサーバーの起動時にのみ変更できます。可能であれば、バイナリログに格納されている行は、この設定の値を超えないサイズのイベントにグループ化されます。イベントを分割できない場合は、最大サイズを超えることができます。

`--binlog-row-event-max-size` オプションは、行ベースのレプリケーションが可能なサーバーで使用できます。行は、このオプションの値を超えないバイト単位のサイズを持つチャンクとして、バイナリログに格納されます。この値は 256 の倍数である必要があります。デフォルト値は 8192 です。

警告

レプリケーションにステートメントベースのロギングを使用する場合、データ変更が非決定的であるようにステートメントが設計されていると、ソースとレプリカのデータが異なる可能性があります。つまり、クエリーオプティマイザに残されます。一般的に、これはレプリケーションの領域外であっても適切なやり方ではありません。この問題についての詳細な説明は、[セクション B.3.7 「MySQL の既知の問題」](#) を参照してください。

5.4.4.3 混合形式のバイナリロギング形式

MIXED のロギング形式で実行すると、サーバーは次の条件のときにステートメントベースのロギングから行ベースのロギングに自動的に切り替わります。

- 関数に `UUID()` が含まれているとき。
- `AUTO_INCREMENT` カラムを含む 1 つ以上のテーブルが更新され、トリガーまたはストアドファンクションが呼び出されたとき。ほかのすべての安全でないステートメントのように、`binlog_format = STATEMENT` の場合にこれによって警告が生成されます。

詳細については、[セクション 17.5.1.1 「レプリケーションと AUTO_INCREMENT」](#) を参照してください。

- ビューの本体が行ベースのレプリケーションを必要とするときに、ビューを作成するステートメントもそれを使用するとき。たとえば、ビューを作成するステートメントが `UUID()` 関数を使用するときに発生します。
- UDF の呼び出しが含まれるとき。
- `FOUND_ROWS()` または `ROW_COUNT()` が使用されるとき。(Bug #12092、Bug #30244)
- `USER()`、`CURRENT_USER()`、または `CURRENT_USER` が使用されるとき。(Bug #28086)
- 関係するテーブルの 1 つが `mysql` データベース内のログテーブルのとき。
- `LOAD_FILE()` 関数が使用されるとき。(Bug #39701)
- ステートメントが 1 つ以上のシステム変数を参照するとき。(Bug #31168)

例外。 次のシステム変数がセッションスコープ (のみ) で使用された場合、ロギング形式の切り替えは発生しません。

- `auto_increment_increment`
- `auto_increment_offset`
- `character_set_client`
- `character_set_connection`
- `character_set_database`
- `character_set_server`
- `collation_connection`
- `collation_database`
- `collation_server`
- `foreign_key_checks`
- `identity`
- `last_insert_id`
- `lc_time_names`
- `pseudo_thread_id`
- `sql_auto_is_null`
- `time_zone`
- `timestamp`
- `unique_checks`

システム変数スコープを決定することについては、[セクション5.1.9「システム変数の使用」](#)を参照してください。

レプリケーションが `sql_mode` を処理する方法については、[セクション17.5.1.39「レプリケーションと変数」](#)を参照してください。

以前のリリースでは、混合バイナリロギング形式が使用されていたときに、ステートメントが行ごとにログに記録され、ステートメントを実行したセッションに一時テーブルがある場合、そのセッションで使用されているすべての一時テーブルが削除されるまで、後続のすべてのステートメントは安全でないものとして扱われ、行ベース形式でログに記録されていました。MySQL 8.0 の時点では、一時テーブルに対する操作は混合バイナリロギング形式で記録されず、セッション内に一時テーブルが存在しても、ステートメントごとに使用されるロギングモードには影響しません。

注記

行ベースのロギングを使用して記述されるべきステートメントをステートメントベースのロギングを使用して実行しようとする、警告が生成されます。警告は、クライアント (`SHOW WARNINGS` の出力内) および `mysql` エラーログの両方に表示されます。そのようなステートメントが実行されるごとに警告が `SHOW WARNINGS` テーブルに追加されます。ただし、ログがいっぱいになるのを防ぐために、各クライアントセッションについて警告を生成した最初のステートメントのみがエラーログに書き込まれます。

前述の判断のほかに、テーブル内の情報が更新されるときに使用されるロギング形式が、個々のエンジンによって決定される場合もあります。個々のエンジンのロギング機能は、次のように定義することができます。

- エンジンが行ベースのロギングをサポートする場合、そのエンジンは行ロギング対応といえます。
- エンジンがステートメントベースのロギングをサポートする場合、そのエンジンはステートメントロギング対応といえます。

ある特定のストレージエンジンは、いずれかまたは両方のロギング形式をサポートできます。次の表に、各エンジンによってサポートされる形式を示します。

ストレージエンジン	行ロギングのサポート	ステートメントロギングのサポート
ARCHIVE	はい	はい
BLACKHOLE	はい	はい
CSV	はい	はい
EXAMPLE	はい	いいえ
FEDERATED	はい	はい
HEAP	はい	はい
InnoDB	はい	トランザクション分離レベルが REPEATABLE READ または SERIALIZABLE の場合は「はい」、それ以外の場合は「いいえ」。
MyISAM	はい	はい
MERGE	はい	はい
NDB	はい	いいえ

ステートメントをログに記録するかどうか、および使用されるロギングモードは、ステートメントのタイプ (安全、安全でない、またはバイナリの注入)、バイナリロギング形式 ([STATEMENT](#)、[ROW](#)、または [MIXED](#))、およびストレージエンジンのロギング機能 (ステートメント対応、行対応、両方、またはいずれか) に従って決定されます。(バイナリインジェクションとは、[ROW](#) 形式を使用してログに記録する必要がある変更のロギングのことをいいます。)

ステートメントがログに記録されるときに警告を出す場合と出さない場合があります。失敗したステートメントはログに記録されませんが、ログにエラーが生成されます。これを次のデシジョンテーブルに示します。Type、binlog_format、SLC および RLC のカラムは条件の概要を示し、エラー / 警告およびログイン名のカラムは対

応するアクションを表します。SLC 「「statement-logging 対応」」を表し、RLC は 「「行ロギング対応」」を表します。

型	binlog_format	SLC	RLC	エラーまたは警告	ロギング形式
*	*	いいえ	いいえ	Error: Cannot execute statement: 行ロギングにもステートメントロギングにも対応していないエンジンが少なくとも1つあるためバイナリロギングは不可能です。	-
安全	STATEMENT	はい	いいえ	-	STATEMENT
安全	MIXED	はい	いいえ	-	STATEMENT
安全	ROW	はい	いいえ	Error: Cannot execute statement: BINLOG_FORMAT = ROW であり、少なくとも1つのテーブルが、行ベースのロギングに対応しないストレージエンジンを使用しているため、バイナリロギングは不可能です。	-
安全でない	STATEMENT	はい	いいえ	Warning: Unsafe statement binlogged in statement format: BINLOG_FORMAT = STATEMENT であるため。	STATEMENT
安全でない	MIXED	はい	いいえ	Error: Cannot execute statement: BINLOG_FORMAT = MIXED であっても、ストレージエンジンがステートメントベースのロギングに限定されている場合、安全でないステートメントのバイナリロギングは不可能です。	-
安全でない	ROW	はい	いいえ	Error: Cannot execute statement: BINLOG_FORMAT = ROW であり、少なくとも1つ	-

型	binlog_format	SLC	RLC	エラーまたは警告	ログ形式
				のテーブルが、行ベースのロギングに対応しないストレージエンジンを使用しているため、バイナリロギングは不可能です。	
行インジェクション	STATEMENT	はい	いいえ	Error: Cannot execute row injection: 少なくとも1つのテーブルが、行ベースのロギングに対応していないストレージエンジンを使用しているため、バイナリロギングは不可能です。	-
行インジェクション	MIXED	はい	いいえ	Error: Cannot execute row injection: 少なくとも1つのテーブルが、行ベースのロギングに対応していないストレージエンジンを使用しているため、バイナリロギングは不可能です。	-
行インジェクション	ROW	はい	いいえ	Error: Cannot execute row injection: 少なくとも1つのテーブルが、行ベースのロギングに対応していないストレージエンジンを使用しているため、バイナリロギングは不可能です。	-
安全	STATEMENT	いいえ	はい	Error: Cannot execute statement: BINLOG_FORMAT = STATEMENT であり、少なくとも1つのテーブルが、ステートメントベースのロギングに対応しないストレージエンジンを使用しているため、バイナリロギングは不可能です。	-

型	binlog_format	SLC	RLC	エラーまたは警告	ロギング形式
安全	MIXED	いいえ	はい	-	ROW
安全	ROW	いいえ	はい	-	ROW
安全でない	STATEMENT	いいえ	はい	Error: Cannot execute statement: BINLOG_FORMAT = STATEMENT であり、少なくとも 1 つのテーブルが、ステートメントベースのロギングに対応しないストレージエンジンを使用しているため、バイナリロギングは不可能です。	-
安全でない	MIXED	いいえ	はい	-	ROW
安全でない	ROW	いいえ	はい	-	ROW
行インジェクション	STATEMENT	いいえ	はい	Error: Cannot execute row injection: BINLOG_FORMAT = STATEMENT のため、バイナリロギングは不可能です。	-
行インジェクション	MIXED	いいえ	はい	-	ROW
行インジェクション	ROW	いいえ	はい	-	ROW
安全	STATEMENT	はい	はい	-	STATEMENT
安全	MIXED	はい	はい	-	STATEMENT
安全	ROW	はい	はい	-	ROW
安全でない	STATEMENT	はい	はい	Warning: Unsafe statement binlogged in statement format: BINLOG_FORMAT = STATEMENT であるため。	STATEMENT
安全でない	MIXED	はい	はい	-	ROW
安全でない	ROW	はい	はい	-	ROW
行インジェクション	STATEMENT	はい	はい	Error: Cannot execute row injection: BINLOG_FORMAT = STATEMENT のため、バイナリロギングは不可能です。	-

型	binlog_format	SLC	RLC	エラーまたは警告	ロギング形式
行インジェクション	MIXED	はい	はい	-	ROW
行インジェクション	ROW	はい	はい	-	ROW

決定によって警告が生成される場合、標準の MySQL 警告が生成されます (警告は `SHOW WARNINGS` を使用して確認できます)。この情報は `mysqld` エラーログにも書き込まれます。ログがいっぱいになるのを防ぐために、エラーは各クライアント接続のエラー発生ごとに 1 つだけログに記録されます。ログメッセージには試行された SQL ステートメントが含められます。

レプリカに警告を表示するように設定された `log_error_verbosity` がある場合、レプリカはメッセージをエラーログに出力して、ジョブを開始するバイナリログとリレーログの座標、別のリレーログへの切替え時、切断後の再接続時、ステートメントベースのロギングに安全でないステートメントなどのステータスに関する情報を提供します。

5.4.4.4 mysql データベーステーブルへの変更に対するロギング形式

`mysql` データベース内の付与テーブルの内容は、直接的に (`INSERT` や `DELETE` などを使用して) または間接的に (`GRANT` や `CREATE USER` などを使用して) 変更することができます。 `mysql` データベーステーブルに影響するステートメントは、次のルールを使用してバイナリログに書き込まれます。

- `mysql` データベーステーブル内のデータを直接変更するデータ操作ステートメントは `binlog_format` システム変数の設定に従ってログに記録されます。これは、`INSERT`、`UPDATE`、`DELETE`、`REPLACE`、`DO`、`LOAD DATA`、`SELECT` や `TRUNCATE TABLE` などのステートメントに関連します。
- `mysql` データベースを間接的に変更するステートメントは、`binlog_format` の値にかかわらずステートメントとしてログに記録されます。これが関係するステートメントは、`GRANT`、`REVOKE`、`SET PASSWORD`、`RENAME USER`、`CREATE` (`CREATE TABLE ... SELECT` を除くすべての形式)、`ALTER` (すべての形式)、および `DROP` (すべての形式) などです。

`CREATE TABLE ... SELECT` はデータ定義とデータ操作の組み合わせです。 `CREATE TABLE` 部分はステートメント形式を使用してログに記録され、`SELECT` 部分は `binlog_format` の値に従ってログに記録されます。

5.4.4.5 バイナリログトランザクション圧縮

MySQL 8.0.20 から、MySQL サーバーインスタンスでバイナリログトランザクション圧縮を有効にできます。バイナリログのトランザクション圧縮が有効になっている場合、トランザクションペイロードは `zstd` アルゴリズムを使用して圧縮され、単一のイベント (`Transaction_payload_event`) としてサーバーのバイナリログファイルに書き込まれます。圧縮トランザクションペイロードは、レプリケーションストリームでレプリカ、他のグループレプリケーショングループメンバー、または `mysqlbinlog` などのクライアントに送信されている間、圧縮状態のままです。これらは受信側スレッドによって解凍されず、圧縮された状態のままリレーログに書き込まれます。したがって、バイナリログトランザクション圧縮では、トランザクションの作成者と受信者 (およびそのバックアップ) の両方に記憶領域が節約され、トランザクションがサーバーインスタンス間で送信されるときにネットワーク帯域幅が節約されます。

圧縮されたトランザクションペイロードは、それに含まれる個々のイベントを検査する必要がある場合に解凍されます。たとえば、受信者に含まれるイベントを適用するために、`Transaction_payload_event` はアプライヤスレッドによって解凍されます。解凍は、リカバリ中、`mysqlbinlog` によるトランザクションのリプレイ時、および `SHOW BINLOG EVENTS` ステートメントと `SHOW RELAYLOG EVENTS` ステートメントによっても実行されます。

`binlog_transaction_compression` システム変数 (デフォルトは `OFF`) を使用して、MySQL サーバーインスタンスでバイナリログトランザクション圧縮を有効にできます。 `binlog_transaction_compression_level_zstd` システム変数を使用して、圧縮に使用される `zstd` アルゴリズムのレベルを設定することもできます。この値は、圧縮作業を 1 (最小作業量) から 22 (最大作業量) の範囲で決定します。圧縮レベルが高くなるにつれて、圧縮率が高くなり、トランザクションペイロードに必要なストレージ領域およびネットワーク帯域幅が削減されます。ただし、データ圧縮に必要な労力も増加し、元のサーバーでは時間と CPU およびメモリーリソースがかかります。圧縮作業の増加には、圧縮率の増加と線形関係はありません。

次のタイプのイベントはバイナリログトランザクション圧縮から除外されるため、常に非圧縮でバイナリログに書き込まれます:

- トランザクションの GTID に関連するイベント (匿名 GTID イベントを含む)。
- 変更イベントやハートビートイベントの表示など、その他のタイプの制御イベント。
- インシデントイベントおよびそれを含むトランザクション全体。
- 非トランザクションイベントおよびそれらを含むトランザクション全体。非トランザクションストレージエンジンとトランザクションストレージエンジンが混在するトランザクションでは、ペイロードは圧縮されません。
- ステートメントベースのバイナリロギングを使用してログに記録されるイベント。バイナリログトランザクションの圧縮は、行ベースのバイナリロギング形式にのみ適用されます。

バイナリログの暗号化は、圧縮されたトランザクションを含むバイナリログファイルで使用できます。

バイナリログトランザクション圧縮が有効な場合の動作

ペイロードが圧縮されたトランザクションは、他のトランザクションと同様にロールバックでき、通常のフィルタリングオプションを使用してレプリカでフィルタ処理で除外することもできます。バイナリログトランザクション圧縮は XA トランザクションに適用できます。

バイナリログのトランザクション圧縮が有効になっている場合、サーバーの `max_allowed_packet` および `slave_max_allowed_packet` の制限は引き続き適用され、`Transaction_payload_event` の圧縮サイズにイベントヘッダーに使用されるバイト数を加えて測定されます。圧縮トランザクションペイロードは、バイナリログトランザクション圧縮が使用されていない場合と同様に、個々のパケットで送信されるトランザクションの各イベントではなく、単一のパケットとして送信されることに注意してください。

マルチスレッドワーカーの場合、各トランザクション (GTID イベントおよび `Transaction_payload_event` を含む) はワーカースレッドに割り当てられます。ワーカースレッドはトランザクションペイロードを解凍し、個別のイベントを 1 つずつ適用します。`Transaction_payload_event` 内でイベントの適用中にエラーが検出された場合、完全なトランザクションは失敗したとして座標系にレポートされます。`slave_parallel_type` が `DATABASE` に設定されている場合、トランザクションがスケジューラされる前に、トランザクションの影響を受けるすべてのデータベースがマップされます。バイナリログトランザクション圧縮を `DATABASE` ポリシーとともに使用すると、イベントごとにマップおよびスケジューラされる圧縮されていないトランザクションと比較して並列度を減らすことができます。

準同期レプリケーション (セクション 17.4.10 「準同期レプリケーション」 を参照) の場合、レプリカは完全な `Transaction_payload_event` を受信したときにトランザクションを確認します。

バイナリログチェックサムが有効になっている場合 (デフォルト)、レプリケーションソースサーバーは、圧縮されたトランザクションペイロード内の個々のイベントのチェックサムを書き込みません。代わりに、完全な `Transaction_payload_event` に対してチェックサムが書き込まれ、GTID に関連するイベントなど、圧縮されなかったすべてのイベントに対して個別のチェックサムが書き込まれます。

`SHOW BINLOG EVENTS` および `SHOW RELAYLOG EVENTS` ステートメントの場合、`Transaction_payload_event` は最初に単一のユニットとして印刷され、次に開梱され、その内部の各イベントが印刷されます。

`UNTIL` 句、`MASTER_POS_WAIT()` および `sql_slave_skip_counter` を使用した `START REPLICA | SLAVE` など、イベントの終了位置を参照する操作の場合、圧縮トランザクションペイロード (`Transaction_payload_event`) の終了位置を指定する必要があります。`sql_slave_skip_counter` を使用してイベントをスキップする場合、圧縮されたトランザクションペイロードは単一のカウンタ値としてカウントされるため、その内部のすべてのイベントは単位としてスキップされます。

圧縮トランザクションペイロードと非圧縮トランザクションペイロードの組合せ

バイナリログトランザクション圧縮をサポートする MySQL Server リリースでは、圧縮されたトランザクションペイロードと圧縮されていないトランザクションペイロードの混在を処理できます。

- バイナリログトランザクション圧縮に関連するシステム変数は、すべてのグループレプリケーショングループメンバーで同じように設定する必要はなく、レプリケーショントポロジのソースからレプリカにレプリケートされません。バイナリログトランザクション圧縮がバイナリログを持つ各 MySQL Server インスタンスに適しているかどうかを判断できます。
- トランザクション圧縮がサーバーで有効になっている場合、圧縮はそのサーバーで発生した将来のトランザクションには適用されませんが、圧縮されたトランザクションペイロードは引き続き処理および表示できます。

- `binlog_transaction_compression` のセッション値を設定して個々のセッションにトランザクション圧縮が指定されている場合、バイナリログには圧縮されたトランザクションペイロードと圧縮されていないトランザクションペイロードを混在させることができます。

レプリケーショントポロジ内のソースとそのレプリカの両方でバイナリログトランザクション圧縮が有効になっている場合、レプリカは圧縮トランザクションペイロードを受信し、それらをリレーログに圧縮して書き込みます。トランザクションペイロードを解凍してトランザクションを適用し、バイナリログへの書き込みのために適用した後で再度圧縮します。ダウンストリームレプリカは、圧縮されたトランザクションペイロードを受信します。

レプリケーショントポロジのソースでバイナリログトランザクション圧縮が有効になっていてもレプリカで有効になっていない場合、レプリカは圧縮されたトランザクションペイロードを受信し、それをリレーログに書き込みます。トランザクションペイロードを解凍してトランザクションを適用し、圧縮解除したペイロードがある場合は独自のバイナリログに書き込みます。ダウンストリームレプリカは、圧縮されていないトランザクションペイロードを受信します。

レプリケーショントポロジのソースでバイナリログトランザクション圧縮が有効になっていないが、そのレプリカでバイナリログがある場合、トランザクションペイロードは適用後に圧縮され、圧縮されたトランザクションペイロードがバイナリログに書き込まれます。ダウンストリームレプリカは、圧縮されたトランザクションペイロードを受信します。

MySQL サーバーインスタンスにバイナリログがない場合、MySQL 8.0.20 からのリリースであれば、`binlog_transaction_compression` の値に関係なく、圧縮されたトランザクションペイロードを受信、処理および表示できます。このようなサーバーインスタンスによって受信された圧縮トランザクションペイロードは、圧縮状態でリレーログに書き込まれるため、レプリケーショントポロジ内の他のサーバーによって実行された圧縮から間接的に利点があります。

MySQL 8.0.20 より前のリリースのレプリカは、バイナリログトランザクション圧縮が有効になっているソースからレプリケートできません。MySQL 8.0.20 以上のレプリカは、バイナリログのトランザクション圧縮をサポートしていない以前のリリースのソースからレプリケートでき、独自のバイナリログに書き込むときに、そのソースから受信したトランザクションに対して独自の圧縮を実行できます。

バイナリログのトランザクション圧縮のモニタリング

「パフォーマンススキーマ」テーブル `binary_log_transaction_compression_stats` を使用して、バイナリログトランザクション圧縮の影響を監視できます。統計には、監視対象期間のデータ圧縮率が含まれ、サーバー上の最後のトランザクションに対する圧縮の影響を表示することもできます。統計をリセットするには、テーブルを切り捨てます。バイナリログおよびリレーログの統計は分割されるため、各ログタイプの圧縮の影響を確認できます。これらの統計を生成するには、MySQL サーバーインスタンスにバイナリログが必要です。

「パフォーマンススキーマ」テーブル `events_stages_current` は、トランザクションがトランザクションペイロードの解凍または圧縮のステージにあるかどうかを示し、このステージの進行状況を表示します。圧縮は、トランザクションがコミットされる直前に、バイナリログトランザクション圧縮 (インシデントイベントなど) からトランザクションを除外するイベントがファイナライズされた取得キャッシュにない場合に、トランザクションを処理するワーカースレッドによって実行されます。解凍が必要な場合は、ペイロードから一度に 1 つのイベントに対して実行されます。

`--verbose` オプションを指定した `mysqlbinlog` には、圧縮されたトランザクションペイロードの圧縮サイズと圧縮されていないサイズ、および使用された圧縮アルゴリズムを示すコメントが含まれます。

`CHANGE REPLICATION SOURCE TO` ステートメント (MySQL 8.0.23 から) の `SOURCE_COMPRESSION_ALGORITHMS` | `MASTER_COMPRESSION_ALGORITHMS` および `SOURCE_ZSTD_COMPRESSION_LEVEL` | `MASTER_ZSTD_COMPRESSION_LEVEL` オプション、または `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 より前)、または非推奨の `slave_compressed_protocol` システム変数を使用して、レプリケーション接続のプロトコルレベルで接続圧縮を有効にできます。接続圧縮も有効になっているシステムでバイナリログのトランザクション圧縮を有効にすると、圧縮されたトランザクションペイロードをさらに圧縮する機会がほとんどない可能性があるため、接続圧縮の影響が軽減されます。ただし、接続圧縮は、圧縮されていないイベントおよびメッセージヘッダーで引き続き動作できます。ネットワーク帯域幅だけでなく記憶領域も節約する必要がある場合は、バイナリログトランザクション圧縮を接続圧縮と組み合わせて有効にできます。レプリケーション接続の接続圧縮の詳細は、[セクション4.2.8「接続圧縮制御」](#) を参照してください。

グループレプリケーションの場合、圧縮は `group_replication_compression_threshold` システム変数で設定されたしきい値を超えるメッセージに対してデフォルトで有効になっています。`group_replication_recovery_compression_algorithm` および `group_replication_recovery_zstd_compression_level` システ

ム変数を使用して、ドナーのバイナリログからの状態転送によって分散回復のために送信されるメッセージの圧縮を構成することもできます。バイナリログトランザクション圧縮が構成されているシステムでバイナリログトランザクション圧縮を有効にしても、グループレプリケーションメッセージ圧縮は圧縮されていないイベントおよびメッセージヘッダーでは引き続き動作できますが、その影響は軽減されます。グループレプリケーションのメッセージ圧縮の詳細は、[セクション18.6.3「メッセージ圧縮」](#)を参照してください。

5.4.5 スロークエリーログ

スロークエリーログは、実行に `long_query_time` 秒を超える時間がかかり、少なくとも `min_examined_row_limit` 行を検査する必要がある SQL ステートメントで構成されます。スロークエリーログは、実行に長い時間がかかっているため最適化の候補となるクエリーを見つけるために使用できます。ただし、長いスロークエリーログの調査には時間がかかる場合があります。これを簡単にするために、`mysqldumpslow` コマンドを使用してスロークエリーログファイルを処理し、その内容を要約できます。[セクション4.6.9「mysqldumpslow — スロークエリーログファイルの要約」](#)を参照してください。

初期ロックを取得する時間は実行時間として計算されません。`mysqld` がスロークエリーログにステートメントを書き込むのは、ステートメントが実行されて、すべてのロックが解放されたあとであるため、ログの順序が実行順と異なる場合があります。

- [スロークエリーログパラメータ](#)
- [スロークエリーログの内容](#)

スロークエリーログパラメータ

`long_query_time` の最小値およびデフォルト値は、それぞれ 0 および 10 です。値はマイクロ秒の精度まで指定できます。

デフォルトでは、管理ステートメントはログに記録されず、参照にインデックスを使用しないクエリーも記録されません。あとで説明するように、この動作は `log_slow_admin_statements` および `log_queries_not_using_indexes` を使用して変更することができます。

デフォルトでは、スロークエリーログは無効になっています。初期のスロークエリーログ状態を明示的に指定するには、`--slow_query_log[={0|1}]` を使用します。引数を指定しないか、引数が 1 の場合、`--slow_query_log` によってログが有効になります。引数が 0 の場合、このオプションによってログが無効になります。ログファイル名を指定するには、`--slow_query_log_file=file_name` を使用します。ログの宛先を指定するには、`log_output` システム変数 ([セクション5.4.1「一般クエリーログおよびスロークエリーログの出力先の選択」](#)を参照) を使用します。

注記

TABLE ログの保存先を指定する場合は、[ログテーブルおよび「開いているファイルが多すぎます」エラー](#)を参照してください。

スロークエリーログファイルの名前を指定しない場合、デフォルト名は `host_name-slow.log` です。サーバーは、別のディレクトリを指定する絶対パス名が指定されないかぎり、データディレクトリ内にファイルを作成します。

実行時にスロークエリーログを無効化または有効化したり、ログファイル名を変更したりするには、グローバルな `slow_query_log` および `slow_query_log_file` システム変数を使用します。`slow_query_log` を 0 に設定してログを無効にするか、1 に設定してログを有効にします。ログファイルの名前を指定するには、`slow_query_log_file` を指定します。ログファイルがすでに開いている場合、ログファイルが閉じて新しいファイルが開きます。

`--log-short-format` オプションを使用すると、サーバーはスロークエリーログに書き込む情報が少なくなります。

スロークエリーログにスロークエリーステートメントを含めるには、`log_slow_admin_statements` システム変数を有効にします。管理ステートメントには、`ALTER TABLE`、`ANALYZE TABLE`、`CHECK TABLE`、`CREATE INDEX`、`DROP INDEX`、`OPTIMIZE TABLE`、および `REPAIR TABLE` が含まれます。

スロークエリーログに書き込まれるステートメントに、行参照についてインデックスを使用しないクエリーを含めるには、`log_queries_not_using_indexes` システム変数を有効にします。(この変数が有効になっていても、テーブルの行数が 2 行未満のためにインデックスの存在からメリットが得られないクエリーはログに記録されません。)

インデックスを使用しないクエリーがログに記録されると、スロークエリーログが迅速に増大する可能性があります。`log_throttle_queries_not_using_indexes` システム変数を設定することによって、これらのクエリーに速度制限を

課することが可能です。デフォルトでは、この変数は 0 で、制限がないことを意味します。正の値を指定すると、インデックスを使用しないクエリーのロギングについて分あたりの制限が課されます。そのような最初のクエリーによって 60 秒間のウィンドウが開き、その期間内でサーバーはクエリーを所定の制限までログに記録し、そのあと、追加のクエリーを抑制します。ウィンドウが終了したときに抑制されたクエリーが存在する場合、サーバーはクエリーが存在した数と、それらに要した集計時間とを示すサマリーをログに記録します。インデックスを使用しない次のクエリーをサーバーがログに記録するとき、別の 60 秒間のウィンドウが開始されます。

サーバーは、スロークエリーログにクエリーを書き込むかどうかを判断するために、制御パラメータを次の順序で使用します。

1. クエリーは管理ステートメントでないが、`log_slow_admin_statements` が有効になっている必要がある。
2. クエリーに少なくとも `long_query_time` 秒かかっているが、`log_queries_not_using_indexes` が有効であって、クエリーは行参照にインデックスを使用していない。
3. クエリーは少なくとも `min_examined_row_limit` 行を検査している必要がある。
4. クエリーは、`log_throttle_queries_not_using_indexes` 設定によって抑制されてはならない。

`log_timestamps` システム変数は、スロークエリーログファイル（および一般クエリーログファイルとエラーログ）に書き込まれるメッセージのタイムスタンプのタイムゾーンを制御します。一般クエリーログおよびログテーブルに書き込まれるスロークエリーログメッセージのタイムゾーンには影響しませんが、これらのテーブルから取得された行は、`CONVERT_TZ()` を使用するか、セッションの `time_zone` システム変数を設定することによって、ローカルシステムのタイムゾーンから任意のタイムゾーンに変換できます。

デフォルトでは、レプリカはレプリケートされたクエリーをスロークエリーログに書き込みません。これを変更するには、`log_slow_slave_statements` システム変数を有効にします。行ベースのレプリケーションが使用されている (`binlog_format=ROW`) 場合、`log_slow_slave_statements` は効果がないことに注意してください。クエリーがレプリカのスロークエリーログに追加されるのは、バイナリログにステートメント形式で記録されている場合、つまり `binlog_format=STATEMENT` が設定されている場合、または `binlog_format=MIXED` が設定されていてステートメントがステートメント形式で記録されている場合だけです。`binlog_format=MIXED` の設定時に行形式でログに記録されるスロークエリー、または `binlog_format=ROW` の設定時にログに記録されるスロークエリーは、`log_slow_slave_statements` が有効な場合でもレプリカのスロークエリーログに追加されません。

スロークエリーログの内容

スロークエリーログが有効になっている場合、サーバーは `log_output` システム変数で指定された宛先に出力を書き込みます。ログを有効にすると、サーバーはログファイルを開き、ログファイルに起動メッセージを書き込みます。ただし、`FILE` ログの出力先が選択されないかぎり、ファイルに対するそれ以上のクエリーのロギングは実行されません。出力先が `NONE` の場合、スロークエリーログが有効な場合であってもサーバーはクエリーを書き込みません。`FILE` が出力先として選択されていない場合、ログファイル名を設定してもロギングには影響しません。

スロークエリーログが有効で、`FILE` が出力先として選択されている場合、ログに書き込まれる各ステートメントの前には#文字で始まり、次のフィールドが含まれます（すべてのフィールドが単一行にあります）:

- `Query_time: duration`

ステートメントの実行時間 (秒)。

- `Lock_time: duration`

ロックを取得する時間 (秒)。

- `Rows_sent: N`

クライアントに送信された行数。

- `Rows_examined:`

サーバーレイヤーによって検査された行数 (ストレージエンジン内部の処理はカウントされません)。

`log_slow_extra` システム変数 (MySQL 8.0.14 の時点で使用可能) を有効にすると、サーバーは、前述のフィールドに加えて次の追加フィールドを `FILE` 出力に書き込みます (`TABLE` 出力には影響しません)。一部のフィールドの説明で

は、ステータス変数名を参照します。詳細は、ステータス変数の説明を参照してください。ただし、スロークエリールログでは、カウンタはステートメントごとの値で、セッションごとの累積値ではありません。

- `Thread_id`: ID
ステートメントスレッド識別子。
- `Errno`: `error_number`
ステートメントのエラー番号。エラーが発生しなかった場合は 0。
- `Killed`: N
ステートメントが終了した場合、理由を示すエラー番号。ステートメントが正常に終了した場合は 0。
- `Bytes_received`: N
ステートメントの `Bytes_received` 値。
- `Bytes_sent`: N
ステートメントの `Bytes_sent` 値。
- `Read_first`: N
ステートメントの `Handler_read_first` 値。
- `Read_last`: N
ステートメントの `Handler_read_last` 値。
- `Read_key`: N
ステートメントの `Handler_read_key` 値。
- `Read_next`: N
ステートメントの `Handler_read_next` 値。
- `Read_prev`: N
ステートメントの `Handler_read_prev` 値。
- `Read_rnd`: N
ステートメントの `Handler_read_rnd` 値。
- `Read_rnd_next`: N
ステートメントの `Handler_read_rnd_next` 値。
- `Sort_merge_passes`: N
ステートメントの `Sort_merge_passes` 値。
- `Sort_range_count`: N
ステートメントの `Sort_range` 値。
- `Sort_rows`: N
ステートメントの `Sort_rows` 値。
- `Sort_scan_count`: N
ステートメントの `Sort_scan` 値。

- **Created_tmp_disk_tables:** N
ステートメントの `Created_tmp_disk_tables` 値。
- **Created_tmp_tables:** N
ステートメントの `Created_tmp_tables` 値。
- **Start:** timestamp
ステートメントの実行開始時間。
- **End:** timestamp
ステートメントの実行終了時間。

指定されたスロークエリーログファイルには、`log_slow_extra` を有効にすることによって追加された追加フィールドの有無にかかわらず、行が混在する場合があります。ログファイルアナライザは、行にフィールド数別の追加フィールドが含まれているかどうかを判断できます。

スロークエリーログファイルに書き込まれる各ステートメントの前には、タイムスタンプを含む `SET` ステートメントが付きます。MySQL 8.0.14 の時点では、タイムスタンプは低速なステートメントの実行が開始された時間を示します。8.0.14 より前は、タイムスタンプは低速なステートメントが記録された時間 (ステートメントの実行が終了した後) に発生) を示していました。

スロークエリーログに書き込まれたステートメント内のパスワードは、文字どおりプレーンテキストで発生しないようにサーバーによって書き換えられます。 [セクション6.1.2.3「パスワードおよびロギング」](#) を参照してください。

5.4.6 サーバーログの保守

[セクション5.4「MySQL Server ログ」](#) で説明したように、MySQL Server は実行中のアクティビティーの内容を確認するのに役立ついくつかの異なるログファイルを作成することができます。ただし、多くのディスクスペースを占有しすぎないようにするために、これらのファイルを定期的にクリーンアップする必要があります。

ロギングを有効にして MySQL を使用しているとき、古いログファイルをときどきバックアップおよび削除して、新しいファイルへのロギングを開始するよう MySQL に指示することが必要な場合があります。 [セクション7.2「データベースバックアップ方法」](#) を参照してください。

Linux (Red Hat) インストールでは、`mysql-log-rotate` スクリプトを使用してログのメンテナンスを行うことができます。RPM 配布から MySQL をインストールした場合、このスクリプトは自動的にインストールされているはずですが、レプリケーション用にバイナリログを使用している場合、このスクリプトには注意が必要です。バイナリログの内容がすべてのレプリカによって処理されていることが確実になるまで、バイナリログを削除しないでください。

ほかのシステムでは、ログファイルを処理するための、`cron` (またはその同等物) で開始する短いスクリプトを自分でインストールする必要があります。

バイナリログファイルは、サーバーのバイナリログの有効期限後に自動的に削除されます。ファイルの削除は、起動時およびバイナリログのフラッシュ時に実行できます。デフォルトのバイナリログの有効期限は 30 日です。別の有効期限を指定するには、`binlog_expire_logs_seconds` システム変数を使用します。レプリケーションを使用している場合は、レプリカがソースより遅れる可能性のある最大時間以下の有効期限を指定する必要があります。バイナリログをオンデマンドで削除するには、`PURGE BINARY LOGS` ステートメントを使用します ([セクション13.4.1.1「PURGE BINARY LOGS ステートメント」](#) を参照してください)。

MySQL で新しいログファイルの使用を強制的に開始するには、ログをフラッシュします。ログのフラッシュは、`FLUSH LOGS` ステートメント、`mysqladmin flush-logs`、`mysqladmin refresh`、`mysqldump --flush-logs` または `mysqldump --master-data` コマンドを実行すると発生します。 [セクション13.7.8.3「FLUSH ステートメント」](#)、[セクション4.5.2「mysqladmin — A MySQL Server 管理プログラム」](#)、および [セクション4.5.4「mysqldump — データベースバックアッププログラム」](#) を参照してください。また、現在のバイナリログファイルサイズが `max_binlog_size` システム変数の値に達すると、サーバーはバイナリログを自動的にフラッシュします。

`FLUSH LOGS` は、個々のログの選択的なフラッシュを可能にするためのオプションの修飾子をサポートします (`FLUSH BINARY LOGS` など)。 [セクション13.7.8.3「FLUSH ステートメント」](#) を参照してください。

ログフラッシュ操作には、次の効果があります：

- ・バイナリロギングが有効化されている場合、サーバーは現在のバイナリログファイルを閉じ、新しいログファイルを次のシーケンス番号で開きます。
- ・ログファイルへの一般クエリーロギングまたはスロークエリーロギングが有効になっている場合、サーバーはログファイルを閉じてから再度開きます。
- ・エラーログがファイルに書き込まれるようにするためにサーバーが `--log-error` オプションで開始されている場合、サーバーはログファイルを閉じて再オープンします。

ログフラッシュステートメントまたはコマンドを実行するには、`RELOAD` 権限を持つアカウントを使用してサーバーに接続する必要があります。Unix および Unix に似たシステムでは、ログをフラッシュする別の方法は、サーバーにシグナルを送信することです。これは、`root` またはサーバープロセスを所有するアカウントによって実行できます。(セクション4.10「MySQLでのUnixシグナル処理」を参照してください。)シグナルを使用すると、サーバーに接続せずにログフラッシュを実行できます：

- ・`SIGHUP` シグナルはすべてのログをフラッシュします。ただし、`SIGHUP` には、望ましくないログフラッシュ以外の効果があります。
- ・MySQL 8.0.19 の時点で、`SIGUSR1` により、サーバーはエラーログ、一般クエリーログおよびスロークエリーログをフラッシュします。これらのログのみをフラッシュする場合は、`SIGUSR1` を、ログに関連しない `SIGHUP` 効果を持たないより多くの「軽量」シグナルとして使用できます。

前述のように、バイナリログをフラッシュすると新しいバイナリログファイルが作成されますが、一般クエリーログ、スロークエリーログ、またはエラーログをフラッシュすると、ログファイルが閉じて再度開きます。後者のログの場合、Unix で新しいログファイルが作成されるようにするには、まず現在のログファイルの名前を変更してからフラッシュします。フラッシュ時に、サーバーは元の名前で新しいログファイルを開きます。たとえば、一般クエリーログ、スロークエリーログ、およびエラーログファイルの名前が `mysql.log`、`mysql-slow.log`、および `err.log` である場合は、コマンド行から次のような一連のコマンドを使用できます：

```
cd mysql-data-directory
mv mysql.log mysql.log.old
mv mysql-slow.log mysql-slow.log.old
mv err.log err.log.old
mysqladmin flush-logs
```

Windows では、`mv` の代わりに `rename` を使用してください。

この時点で、`mysql.log.old`、`mysql-slow.log.old` および `err.log.old` のバックアップを作成し、ディスクから削除できます。

実行時に一般クエリーログまたはスロークエリーログの名前を変更するには、まずサーバーに接続し、ログを無効にします：

```
SET GLOBAL general_log = 'OFF';
SET GLOBAL slow_query_log = 'OFF';
```

ログを無効にして、ログファイルの名前を外部 (たとえば、コマンドラインから) に変更します。次に、ログをふたたび有効にします。

```
SET GLOBAL general_log = 'ON';
SET GLOBAL slow_query_log = 'ON';
```

この方法はすべてのプラットフォームで動作し、サーバー再起動を必要としません。

注記

外部でファイルの名前を変更した後にサーバーが特定のログファイルを再作成するには、ファイルの場所がサーバーによって書き込み可能である必要があります。これは常に当てはまるわけではありません。たとえば、Linux では、サーバーはエラーログを `/var/log/mysql.log` として書き込むことができます。ここで、`/var/log` は `root` によって所有され、`mysqld` によって書き込み可能ではありません。この場合、ログフラッシュ操作は新しいログファイルの作成に失敗します。

この状況に対処するには、元のログファイルの名前を変更した後、適切な所有権を持つ新しいログファイルを手動で作成する必要があります。たとえば、次のコマンドを `root` として実行します:

```
mv /var/log/mysqld.log /var/log/mysqld.log.old
install -omysql -gmysql -m0644 /dev/null /var/log/mysqld.log
```

5.5 MySQL のコンポーネント

MySQL Server には、サーバー機能を拡張するためのコンポーネントベースのインフラストラクチャが含まれています。コンポーネントは、サーバーおよびその他のコンポーネントで使用可能なサービスを提供します。(サービスの使用に関しては、サーバーは他のコンポーネントと同等のコンポーネントです。)コンポーネントは、提供するサービスを介してのみ相互作用します。

MySQL ディストリビューションには、サーバー拡張を実装するいくつかのコンポーネントが含まれています:

- エラーロギングを構成するコンポーネント。 [セクション5.4.2「エラーログ」](#) および [セクション5.5.3「エラーログコンポーネント」](#) を参照してください。
- パスワードを確認するためのコンポーネント。 [セクション6.4.3「パスワード検証コンポーネント」](#) を参照してください。
- アプリケーションが独自のメッセージイベントを監査ログに追加できるようにするコンポーネント。 [セクション6.4.6「監査メッセージコンポーネント」](#) を参照してください。
- クエリー属性にアクセスするためのユーザー定義関数を実装するコンポーネント。 [セクション9.6「クエリー属性」](#) を参照してください。

コンポーネントによって実装されたシステム変数およびステータス変数は、コンポーネントのインストール時に公開され、コンポーネント固有の接頭辞で始まる名前を持ちます。たとえば、`log_filter_dragonet` エラーログフィルタコンポーネントは、`log_error_filter_rules` というシステム変数を実装します。このシステム変数のフルネームは `dragonet.log_error_filter_rules` です。この変数を参照するには、フルネームを使用します。

次の各セクションでは、コンポーネントをインストールおよびアンインストールする方法と、実行時にどのコンポーネントをインストールするかを決定し、それらに関する情報を取得する方法について説明します。

コンポーネントの内部実装の詳細は、<https://dev.mysql.com/doc/index-other.html> で入手可能な MySQL Server Doxygen のドキュメントを参照してください。たとえば、独自のコンポーネントを記述する場合、この情報はコンポーネントの動作を理解するために重要です。

5.5.1 コンポーネントのインストールおよびアンインストール

コンポーネントは、使用する前にサーバーにロードする必要があります。MySQL は、実行時の手動コンポーネントロードおよびサーバー起動時の自動ロードをサポートしています。

コンポーネントがロードされている間、[セクション5.5.2「コンポーネント情報の取得」](#) で説明されているように、コンポーネントに関する情報を使用できます。

`INSTALL COMPONENT` および `UNINSTALL COMPONENT` SQL ステートメントを使用すると、コンポーネントのロードおよびアンロードが可能になります。例:

```
INSTALL COMPONENT 'file://component_validate_password';
UNINSTALL COMPONENT 'file://component_validate_password';
```

ローダーサービスは、コンポーネントのロードおよびアンロードを処理し、ロードされたコンポーネントを `mysql.component` システムテーブルに登録します。

コンポーネント操作の SQL ステートメントは、次のようにサーバー操作および `mysql.component` システムテーブルに影響します:

- `INSTALL COMPONENT` は、コンポーネントをサーバーにロードします。コンポーネントがすぐにアクティブになります。ローダーサービスでは、ロードされたコンポーネントも `mysql.component` システムテーブルに登録されます。その後のサーバーの再起動では、ローダーサービスは、起動シーケンス中に `mysql.component` にリストされて

いるコンポーネントをすべてロードします。これは、サーバーが `--skip-grant-tables` オプションを使用して起動された場合でも発生します。

- **UNINSTALL COMPONENT** によってコンポーネントが非アクティブ化され、サーバーからアンロードされます。ローダーサービスは、`mysql.component` システムテーブルからコンポーネントの登録を解除して、その後の再起動のためにサーバーが起動シーケンス中にコンポーネントをロードしないようにします。

サーバープラグインの対応する **INSTALL PLUGIN** ステートメントと比較すると、コンポーネントの **INSTALL COMPONENT** ステートメントには、コンポーネントに名前を付けるためのプラットフォーム固有のファイル名接尾辞を知る必要がないという大きな利点があります。つまり、特定の **INSTALL COMPONENT** ステートメントをプラットフォーム間で均一に実行できます。

コンポーネントをインストールすると、関連するユーザー定義関数 (UDF) も自動的にインストールされる場合があります。その場合、アンインストール時にコンポーネントによってそれらの UDF も自動的にアンインストールされます。

5.5.2 コンポーネント情報の取得

`mysql.component` システムテーブルには、現在ロードされているコンポーネントに関する情報と、**INSTALL COMPONENT** を使用して登録されたコンポーネントが表示されます。インストールされているコンポーネントを確認するには、次のステートメントを使用します：

```
SELECT * FROM mysql.component;
```

5.5.3 エラーログコンポーネント

このセクションでは、個々のエラーログコンポーネントの特性について説明します。エラーロギングの構成に関する一般情報は、[セクション5.4.2「エラーログ」](#)を参照してください。

ログコンポーネントには、フィルタまたはシンクを指定できます：

- フィルタは、ログイベントを処理し、イベントフィールドを追加、削除または変更したり、イベント全体を削除します。結果のイベントは、有効なコンポーネントのリスト内の次のログコンポーネントに渡されます。
- シンクは、ログイベントの宛先 (ライター) です。通常、シンクはログイベントを特定の形式のログメッセージに処理し、これらのメッセージをファイルやシステムログなどの関連出力に書き込みます。シンクは、パフォーマンススキーマ `error_log` テーブルに書き込むこともできます。[セクション27.12.19.1「error_log テーブル」](#)を参照してください。イベントは、有効なコンポーネントのリスト内の次のログコンポーネントに未変更で渡されます (つまり、シンクはイベントをフォーマットして出力メッセージを生成しますが、イベントは次のコンポーネントに内部的に渡されるため変更されません)。

`log_error_services` システム変数値には、有効なログコンポーネントがリストされます。リストに指定されていないコンポーネントは無効になります。

次の各セクションでは、コンポーネントタイプ別にグループ化された個々のログコンポーネントについて説明します：

- [エラーログコンポーネントのフィルタ](#)
- [シンクエラーログコンポーネント](#)

コンポーネントの説明には、次のタイプの情報が含まれます：

- コンポーネント名と目的。
- コンポーネントが組み込まれているか、ロードする必要があるか。ロード可能なコンポーネントの場合、説明には、**INSTALL COMPONENT** および **UNINSTALL COMPONENT** ステートメントを使用してコンポーネントをロードおよびアンロードするために使用する URN を指定します。
- コンポーネントを `log_error_services` 値に複数回リストできるかどうか。
- シンクコンポーネントの場合、コンポーネントが出力を書き込む宛先。
- シンクコンポーネントの場合、パフォーマンススキーマ `error_log` テーブルへのインタフェースをサポートするかどうか。

エラーログコンポーネントのフィルタ

エラーログフィルタコンポーネントは、エラーログイベントのフィルタリングを実装します。フィルタコンポーネントが有効になっていない場合、フィルタリングは行われません。

有効なフィルタコンポーネントは、`log_error_services` 値の後半にリストされているコンポーネントのログイベントにのみ影響します。特に、フィルタコンポーネントより前の `log_error_services` にリストされているログシンクコンポーネントでは、ログイベントのフィルタリングは行われません。

log_filter_internal コンポーネント

- 目的: `log_error_verbosity` および `log_error_suppression_list` システム変数と組み合わせて、ログイベントの優先度およびエラーコードに基づいてフィルタリングを実装します。 [セクション5.4.2.5「優先度ベースのエラーログのフィルタリング \(log_filter_internal\)」](#) を参照してください。
- URN: このコンポーネントは組み込まれており、使用する前に `INSTALL COMPONENT` とともにロードする必要はありません。
- 複数使用の許可: No.

`log_filter_internal` が無効になっている場合、`log_error_verbosity` および `log_error_suppression_list` は無効になります。

log_filter_dragnet コンポーネント

- 目的: `dragnet.log_error_filter_rules` システム変数設定で定義されたルールに基づいてフィルタリングを実装します。 [セクション5.4.2.6「ルールベースのエラーログのフィルタリング \(log_filter_dragnet\)」](#) を参照してください。
- URN: `file://component_log_filter_dragnet`
- 複数使用の許可: No.

シンクエラーログコンポーネント

エラーログシンクコンポーネントは、エラーログ出力を実装するライターです。シンクコンポーネントが有効になっていない場合、ログ出力は発生しません。

シンクコンポーネントの説明の中には、デフォルトのエラーログの宛先を参照するものがあります。これはコンソールまたはファイルであり、 [セクション5.4.2.2「デフォルトのエラーログ保存先の構成」](#) で説明されているように、`log_error` システム変数の値によって示されます。

log_sink_internal コンポーネント

- 目的: 従来のエラーログメッセージ出力形式を実装します。
- URN: このコンポーネントは組み込まれており、使用する前に `INSTALL COMPONENT` とともにロードする必要はありません。
- 複数使用の許可: No.
- 出力先: デフォルトのエラーログの宛先に書き込みます。
- パフォーマンススキーマのサポート: `error_log` テーブルに書き込みます。以前のサーバーインスタンスによって作成されたエラーログファイルを読み取るためのパーサーを提供します。

log_sink_json コンポーネント

- 目的: JSON 形式のエラーロギングを実装します。 [セクション5.4.2.7「JSON 形式でのエラーロギング」](#) を参照してください。
- URN: `file://component_log_sink_json`
- 複数使用の許可: Yes.

- 出力先: このシンクは、`log_error` システム変数で指定されたデフォルトのエラーログの宛先に基づいて出力先を決定します:
 - `log_error` がファイルに名前を付ける場合、シンクはそのファイル名に加えて、`NN` が 00 から始まる番号付き `.NN.json` 接尾辞に基づいて出力ファイルの名前を付けます。たとえば、`log_error` が `file_name` の場合、`log_error_services` 値で指定された `log_sink_json` の連続するインスタンスは `file_name.00.json`、`file_name.01.json` など書き込まれます。
 - `log_error` が `stderr` の場合、シンクはコンソールに書き込みます。`log_sink_json` の名前が `log_error_services` 値で複数回指定されている場合は、コンソールに書き込まれるため、役に立たない可能性があります。
- パフォーマンススキーマのサポート: `error_log` テーブルに書き込みます。以前のサーバーインスタンスによって作成されたエラーログファイルを読み取るためのパーサーを提供します。

log_sink_syseventlog コンポーネント

- 目的: エラーロギングをシステムログに実装します。これは、Windows ではイベントログ、Unix および Unix に似たシステムでは `syslog` です。[セクション5.4.2.8「システムログへのエラーロギング」](#)を参照してください。
- URN: `file://component_log_sink_syseventlog`
- 複数使用の許可: No.
- 出力先: システムログに書き込みます。デフォルトのエラーログの保存先は使用しません。
- パフォーマンススキーマのサポート: `error_log` テーブルには書き込まれません。以前のサーバーインスタンスによって作成されたエラーログファイルを読み取るパーサーを提供しません。

log_sink_test コンポーネント

- 目的: 本番での使用ではなく、テストケースの記述での内部使用を目的としています。
- URN: `file://component_log_sink_test`

複数使用が許可されているかどうかや、出力先が `log_sink_test` に指定されていないかどうかなどのシンクプロパティ。これは、前述のように内部使用のためです。そのため、その動作はいつでも変更される可能性があります。

5.5.4 クエリー属性コンポーネント

MySQL 8.0.23 の時点では、コンポーネントサービスはクエリー属性へのアクセスを提供します ([セクション9.6「クエリー属性」](#)を参照)。`query_attributes` コンポーネントは、このサービスを使用して、SQL ステートメント内のクエリー属性へのアクセスを提供します。

- 目的: 属性名引数を取り、属性値を文字列として返す `mysql_query_attribute_string()` ユーザー定義関数、または属性が存在しない場合は `NULL` を実装します。
- URN: `file://component_query_attributes`

`query_attributes` で使用されるのと同じクエリー属性コンポーネントサービスを組み込む開発者は、MySQL ソース配布の `mysql_query_attributes.h` ファイルを参照する必要があります。

5.6 MySQL Server プラグイン

MySQL は、サーバープラグインの作成を可能にするプラグイン API をサポートしています。プラグインはサーバー起動時にロードしたり、実行時にサーバーを再起動せずにロードおよびアンロードしたりできます。このインターフェイスでサポートされているプラグインには、ストレージエンジン、`INFORMATION_SCHEMA` テーブル、全文パーサープラグイン、およびサーバー拡張機能が含まれます。

MySQL ディストリビューションには、サーバー拡張機能を実装する複数のプラグインが含まれています:

- クライアントによる MySQL Server への接続試行を認証するためのプラグイン。プラグインは、複数の認証プロトコルで使用できます。[セクション6.2.17「プラグイン認証」](#)を参照してください。

- 接続制御プラグイン。これにより、管理者は、クライアント接続試行が連続して失敗した後、増加する遅延を導入できます。 [セクション6.4.2「Connection-Control プラグイン」](#)を参照してください。
- パスワード検証プラグインは、パスワード強度ポリシーを実装し、潜在的なパスワードの強度を評価します。 [セクション6.4.3「パスワード検証コンポーネント」](#)を参照してください。
- 準同期レプリケーションプラグインは、レプリケーション機能へのインタフェースを実装します。このインタフェースを使用すると、少なくとも1つのレプリカが各トランザクションに応答しているかぎり、ソースを続行できます。 [セクション17.4.10「準同期レプリケーション」](#)を参照してください。
- グループレプリケーションを使用すると、MySQL サーバーインスタンスのグループ全体で可用性の高い分散 MySQL サービスを作成でき、データ整合性、競合検出および解決、グループメンバーシップサービスがすべて組み込まれています。 [第18章「グループレプリケーション」](#)を参照してください。
- MySQL Enterprise Edition には、多数のクライアント接続のステートメント実行スレッドを効率的に管理することによってサーバーのパフォーマンスを向上させるために、接続スレッドを管理するスレッドプールプラグインが含まれています。 [セクション5.6.3「MySQL Enterprise Thread Pool」](#)を参照してください。
- MySQL Enterprise Edition には、接続およびクエリーアクティビティを監視およびロギングするための監査プラグインが含まれています。 [セクション6.4.5「MySQL Enterprise Audit」](#)を参照してください。
- MySQL Enterprise Edition には、アプリケーションレベルのファイアウォールを実装するファイアウォールプラグインが含まれており、データベース管理者は、受け入れられたステートメントパターンの許可リストに対する照合に基づいて SQL ステートメントの実行を許可または拒否できます。 [セクション6.4.7「MySQL Enterprise Firewall」](#)を参照してください。
- クエリーリライトプラグインは、MySQL Server によって受信されたステートメントを調べ、サーバーが実行する前にリライトする可能性があります。 [セクション5.6.4「リライタクエリーリライトプラグイン」](#) および [セクション5.6.5「ddl_rewriter プラグイン」](#)を参照してください。
- バージョントークンを使用すると、アプリケーションが不正または古いデータへのアクセスを防ぐために使用できるサーバートークンを作成して同期できます。バージョントークンは、`version_tokens` プラグインと一連のユーザー定義関数を実装するプラグインライブラリに基づいています。 [セクション5.6.6「バージョントークン」](#)を参照してください。
- キーリングプラグインは、機密情報のためのセキュアなストレージを提供します。 [セクション6.4.4「MySQL キーリング」](#)を参照してください。
- X プラグイン は、ドキュメントストアとして機能できるように MySQL Server を拡張します。X プラグイン を実行すると、MySQL の ACID 準拠の記憶域機能をドキュメントストアとして公開するように設計された X プロトコル を使用して、MySQL Server がクライアントと通信できるようになります。 [セクション20.5「X プラグイン」](#)を参照してください。
- クローンを使用すると、ローカルまたはリモートの MySQL サーバーインスタンスから InnoDB データをクローニングできます。 [セクション5.6.7「クローンプラグイン」](#)を参照してください。
- テストフレームワークプラグインは、サーバーサービスをテストします。これらのプラグインの詳細は、<https://dev.mysql.com/doc/index-other.html> で入手可能な MySQL Server Doxygen ドキュメントの「プラグインサービスのテスト」用プラグインに関するセクションを参照してください。

次の各セクションでは、プラグインをインストールおよびアンインストールする方法と、実行時にインストールされるプラグインを決定し、それらに関する情報を取得する方法について説明します。プラグインの作成の詳細は、[The MySQL Plugin API](#) を参照してください。

5.6.1 プラグインのインストールおよびアンインストール

サーバープラグインは、使用する前にサーバーにロードする必要があります。MySQL は、サーバーの起動時および実行時のプラグインのロードをサポートしています。また、起動時にロードされたプラグインのアクティブ化状態を制御し、実行時にアンロードすることもできます。

プラグインがロードされている間は、[セクション5.6.2「サーバープラグイン情報の取得」](#) で説明されているように、プラグインに関する情報を入手できます。

- [プラグインのインストール](#)
- [プラグインのアクティブ化状態の制御](#)
- [プラグインのアンインストール](#)
- [プラグインおよびユーザー定義関数](#)

プラグインのインストール

サーバープラグインを使用する前に、次のいずれかの方法を使用してインストールする必要があります。説明で、`plugin_name` は `innodb`、`csv`、`validate_password` などのプラグイン名を表します。

- [組み込みプラグイン](#)
- `mysql.plugin` システムテーブルに登録されているプラグイン
- コマンドラインオプションで指定されたプラグイン
- `INSTALL PLUGIN` ステートメントでインストールされるプラグイン

組み込みプラグイン

組み込みプラグインは、サーバーによって自動的に認識されます。デフォルトでは、サーバーは起動時にプラグインを有効にします。一部の組み込みプラグインでは、これを `--plugin_name[=activation_state]` オプションで変更できます。

mysql.plugin システムテーブルに登録されているプラグイン

`mysql.plugin` システムテーブルは、プラグインのレジストリとして機能します (登録する必要のない組み込みプラグイン以外)。通常の起動シーケンスでは、サーバーはテーブルに登録されているプラグインをロードします。デフォルトでは、`mysql.plugin` テーブルからロードされたプラグインの場合、サーバーはプラグインも有効にします。これは、`--plugin_name[=activation_state]` オプションで変更できます。

サーバーが `--skip-grant-tables` オプションで起動された場合、`mysql.plugin` テーブルに登録されているプラグインはロードされず、使用できません。

コマンドラインオプションで指定されたプラグイン

プラグインライブラリファイルにあるプラグインは、サーバーの起動時に `--plugin-load`、`--plugin-load-add`、または `--early-plugin-load` オプションを使用してロードできます。通常、起動時にロードされたプラグインの場合、サーバーはプラグインも有効にします。これは、`--plugin_name[=activation_state]` オプションで変更できます。

`--plugin-load` および `--plugin-load-add` オプションは、組み込みプラグインおよびストレージエンジンがサーバーの起動シーケンス中に初期化されたあとにプラグインをロードします。`--early-plugin-load` オプションは、組み込みプラグインおよびストレージエンジンを初期化する前に使用可能である必要があるプラグインをロードするために使用されます。

各プラグインロードオプションの値は、`name = plugin_library` および `plugin_library` の値をセミコロンで区切ったリストです。各 `name` はロードするプラグインの名前で、`plugin_library` はプラグインコードを含むライブラリファイルの名前です。プラグイン名を前に付けずにプラグインライブラリを指定した場合、サーバーはライブラリ内のすべてのプラグインをロードします。サーバーは、`plugin_dir` システム変数で指定されたディレクトリ内でプラグインライブラリファイルを検索します。

プラグインロードオプションでは、プラグインは `mysql.plugin` テーブルに登録されません。その後の再起動では、`--plugin-load`、`--plugin-load-add` または `--early-plugin-load` が再度指定された場合のみ、サーバーはプラグインを再度ロードします。つまり、このオプションを指定すると、単一サーバーの起動のために保持されるワンタイムプラグインのインストール操作が生成されます。

`--plugin-load`、`--plugin-load-add` および `--early-plugin-load` を使用すると、`--skip-grant-tables` が指定されている場合でもプラグインをロードできます (これにより、サーバーは `mysql.plugin` テーブルを無視します)。`--plugin-load`、`--plugin-load-add` および `--early-plugin-load` を使用すると、実行時にロードできないプラグインを起動時にロードすることもできます。

`--plugin-load-add` オプションは、`--plugin-load` オプションを補完します:

- `--plugin-load` の各インスタンスは、起動時にロードするプラグインのセットをリセットしますが、`--plugin-load-add` は、現在のセットをリセットせずに、ロードするプラグインのセットにプラグインを追加します。したがって、`--plugin-load` の複数のインスタンスが指定されている場合は、最後のインスタンスのみが有効になります。`--plugin-load-add` の複数のインスタンスでは、それらはすべて有効になります。
- 引数の形式は `--plugin-load` の場合と同じですが、`--plugin-load-add` の複数のインスタンスを使用して、多数のプラグインを単一の長いアンワイルディ `--plugin-load` 引数として指定しないようにできます。
- `--plugin-load-add` は `--plugin-load` がない場合でも指定できますが、`--plugin-load` がロードするプラグインのセットをリセットするため、`--plugin-load` より前に表示されていた `--plugin-load-add` のインスタンスは影響を受けません。

たとえば、次のオプションがあります:

```
--plugin-load=x --plugin-load-add=y
```

次のオプションと同等です:

```
--plugin-load-add=x --plugin-load-add=y
```

また、次のオプションと同等です:

```
--plugin-load="x;y"
```

ただし、次のオプションの場合、

```
--plugin-load-add=y --plugin-load=x
```

上記は次のオプションと同等です。

```
--plugin-load=x
```

INSTALL PLUGIN ステートメントでインストールされるプラグイン

プラグインライブラリファイルにあるプラグインは、実行時に `INSTALL PLUGIN` ステートメントを使用してロードできます。このステートメントはさらにプラグインを `mysql.plugin` テーブルに登録し、その後再起動してサーバーからロードします。このため、`INSTALL PLUGIN` には、`mysql.plugin` テーブルに対する `INSERT` 権限が必要です。

プラグインライブラリファイルのベース名は、プラットフォームによって異なります。一般的な接尾辞は、`.so` for Unix および Unix のようなシステム、`.dll` for Windows です。

例: `--plugin-load-add` オプションは、サーバーの起動時にプラグインをインストールします。 `somepluglib.so` という名前のプラグインライブラリファイルから `myplugin` という名前のプラグインをインストールするには、`my.cnf` ファイルで次の行を使用します:

```
[mysqld]
plugin-load-add=myplugin=somepluglib.so
```

この場合、プラグインは `mysql.plugin` に登録されません。`--plugin-load-add` オプションを指定せずにサーバーを再起動すると、起動時にプラグインがロードされません。

一方、`INSTALL PLUGIN` ステートメントでは、サーバーは起動時にライブラリファイルからプラグインコードをロードします。

```
INSTALL PLUGIN myplugin SONAME 'somepluglib.so';
```

`INSTALL PLUGIN` では、「permanent」プラグインの登録も行われます: プラグインは、その後の再起動時にサーバーがロードするように、`mysql.plugin` テーブルにリストされます。

多くのプラグインは、サーバーの起動時または実行時のいずれかにロードできます。ただし、プラグインがサーバーの起動時にロードおよび初期化する必要があるように設計されている場合は、`INSTALL PLUGIN` を使用して実行時にロードしようとするエラーが発生します:

```
mysql> INSTALL PLUGIN myplugin SONAME 'somepluglib.so';
```

```
ERROR 1721 (HY000): Plugin 'myplugin' is marked as not dynamically installable. You have to stop the server to install it.
```

この場合、`--plugin-load`、`--plugin-load-add` または `--early-plugin-load` を使用する必要があります。

プラグインの名前が `mysql.plugin` テーブルで `--plugin-load`、`--plugin-load-add`、または `--early-plugin-load` オプションと (`INSTALL PLUGIN` ステートメントの以前の結果として) の両方を使用して指定されている場合、サーバーは起動しますが、これらのメッセージはエラーログに書き込まれます:

```
[ERROR] Function 'plugin_name' already exists  
[Warning] Couldn't load plugin named 'plugin_name'  
with soname 'plugin_object_file'.
```

プラグインのアクティブ化状態の制御

プラグインの起動時にサーバーがプラグインについて知っている場合 (たとえば、プラグインに `--plugin-load-add` オプションを使用して名前が付けられているか、`mysql.plugin` テーブルに登録されているため)、サーバーはデフォルトでプラグインをロードして有効にします。このようなプラグインのアクティブ化状態は、`--plugin_name[=activation_state]` 起動オプションを使用して制御できます。ここで、`plugin_name` は、`innodb`、`csv`、`validate_password` など、影響を受けるプラグインの名前です。ほかのオプションと同じように、オプション名のダッシュとアンダースコアは交換可能です。また、アクティブ化状態の値では大文字と小文字は区別されません。たとえば、`--my_plugin=ON` と `--my-plugin=on` は同等です。

- `--plugin_name=OFF`

プラグインを無効にするようサーバーに指示します。これは、`mysql_native_password` などの特定の組み込みプラグインでは不可能な場合があります。

- `--plugin_name[=ON]`

プラグインを有効にするようサーバーに指示します。(値を付けずにオプションを `--plugin_name` と指定しても効果は同じです。) プラグインが初期化に失敗した場合、サーバーはプラグインを無効にして実行します。

- `--plugin_name=FORCE`

プラグインを有効にするようサーバーに指示しますが、プラグインの初期化が失敗した場合、サーバーは開始しません。つまり、このオプションはプラグインを有効にしてサーバーを実行するか、何もしないかのいずれかを強制します。

- `--plugin_name=FORCE_PLUS_PERMANENT`

`FORCE` と似ていますが、さらにプラグインが実行時にアンロードされないようにします。ユーザーが `UNINSTALL PLUGIN` を使用してこの操作を実行しようとすると、エラーが発生します。

プラグインアクティブ化状態は、`INFORMATION_SCHEMA.PLUGINS` テーブルの `LOAD_OPTION` カラムに表示されます。

`CSV`、`BLACKHOLE` および `ARCHIVE` が組み込みのプラグナブルなストレージエンジンであり、サーバーが起動時にそれらをロードする必要があるとします。条件は次のとおりです: `CSV` の初期化が失敗した場合、サーバーの実行が許可され、`BLACKHOLE` の初期化が成功する必要があり、`ARCHIVE` を無効にする必要があります。これを満たすには、オプションファイル内で次の行を使用します。

```
[mysqld]  
csv=ON  
blackhole=FORCE  
archive=OFF
```

`--enable-plugin_name` オプションの形式は、`--plugin_name=ON` のシノニムです。 `--disable-plugin_name` および `--skip-plugin_name` のオプション形式は、`--plugin_name=OFF` のシノニムです。

プラグインが `OFF` で明示的に無効化されているか、`ON` で有効化されていても初期化に失敗したために暗黙的に無効化されている場合、プラグインを変更する必要があるサーバー操作の側面。たとえば、プラグインがストレージエンジンを実装している場合、ストレージエンジンの既存のテーブルにアクセスできなくなり、ストレージエンジンの新しいテーブルを作成しようとすると、代わりに `NO_ENGINE_SUBSTITUTION` SQL モードが有効になってエラーが発生しないかぎり、デフォルトのストレージエンジンを使用するテーブルになります。

プラグインの無効化は、ほかのオプションの調整を要することもあります。たとえば、`--skip-innodb` を使用して InnoDB を無効にするためにサーバーを起動する場合、起動時に他の `innodb_xxx` オプションも省略する必要がある可能性があります。また、InnoDB はデフォルトのストレージエンジンであるため、`--default_storage_engine` で別の使用可能なストレージエンジンを指定しないかぎり起動できません。`--default_tmp_storage_engine` も設定する必要があります。

プラグインのアンインストール

実行時に、`UNINSTALL PLUGIN` ステートメントはサーバーに認識されているプラグインを無効にしてアンインストールします。このステートメントは、プラグインをアンロードし、`mysql.plugin` システムテーブルに登録されている場合はそこから削除します。このため、`UNINSTALL PLUGIN` ステートメントには、`mysql.plugin` テーブルに対する `DELETE` 権限が必要です。プラグインがテーブルに登録されなくなったため、サーバーはその後の再起動時にプラグインをロードしません。

`UNINSTALL PLUGIN` では、プラグインが `INSTALL PLUGIN` を使用して実行時にロードされたか、プラグインロードオプションを使用して起動時にロードされたかに関係なく、次の条件に従ってプラグインをアンロードできます:

- サーバーに組み込まれているプラグインをアンロードできません。これらは `INFORMATION_SCHEMA.PLUGINS` または `SHOW PLUGINS` からの出力で、ライブラリ名が `NULL` のプラグインとして識別できます。
- 実行時にプラグインをアンロードしないようにする `--plugin_name=FORCE_PLUS_PERMANENT` を使用すると、サーバーが開始された目的のプラグインをアンロードできません。これらは `INFORMATION_SCHEMA.PLUGINS` テーブルの `LOAD_OPTION` カラムから識別できます。

サーバーの起動時にプラグインロードオプションを使用して現在ロードされているプラグインをアンインストールするには、この手順を使用します。

1. プラグインに関連するオプションおよびシステム変数を `my.cnf` ファイルから削除します。プラグインシステム変数が `mysqld-auto.cnf` ファイルに永続化されていた場合は、削除するプラグインごとに `RESET PERSIST var_name` を使用して削除します。
2. サーバーを再起動します。
3. プラグインは通常、起動時にプラグインロードオプションを使用するか、実行時に `INSTALL PLUGIN` とともにインストールされますが、両方を使用することはできません。ただし、ある時点で `INSTALL PLUGIN` も使用されている場合は、プラグインのオプションを `my.cnf` ファイルから削除するだけではアンインストールできない可能性があります。`INFORMATION_SCHEMA.PLUGINS` または `SHOW PLUGINS` からの出力にプラグインがまだ表示される場合は、`UNINSTALL PLUGIN` を使用して `mysql.plugin` テーブルからプラグインを削除します。その後、サーバーを再起動します。

プラグインおよびユーザー定義関数

プラグインをインストールすると、関連するユーザー定義関数 (UDF) も自動的にインストールされる場合があります。その場合、アンインストール時にプラグインによってそれらの UDF も自動的にアンインストールされます。

5.6.2 サーバープラグイン情報の取得

サーバーにインストールされているプラグインを調べるにはいくつかの方法があります。

- `INFORMATION_SCHEMA.PLUGINS` テーブルには、ロードされているプラグインの行が含まれています。`PLUGIN_LIBRARY` 値が `NULL` のプラグインは組み込み型であり、アンロードできません。

```
mysql> SELECT * FROM INFORMATION_SCHEMA.PLUGINS\G
***** 1. row *****
  PLUGIN_NAME: binlog
  PLUGIN_VERSION: 1.0
  PLUGIN_STATUS: ACTIVE
  PLUGIN_TYPE: STORAGE ENGINE
  PLUGIN_TYPE_VERSION: 50158.0
  PLUGIN_LIBRARY: NULL
  PLUGIN_LIBRARY_VERSION: NULL
  PLUGIN_AUTHOR: Oracle Corporation
  PLUGIN_DESCRIPTION: This is a pseudo storage engine to represent the binlog in a transaction
  PLUGIN_LICENSE: GPL
  LOAD_OPTION: FORCE
```

```

...
***** 10. row *****
  PLUGIN_NAME: InnoDB
  PLUGIN_VERSION: 1.0
  PLUGIN_STATUS: ACTIVE
  PLUGIN_TYPE: STORAGE ENGINE
  PLUGIN_TYPE_VERSION: 50158.0
  PLUGIN_LIBRARY: ha_innodb_plugin.so
  PLUGIN_LIBRARY_VERSION: 1.0
  PLUGIN_AUTHOR: Oracle Corporation
  PLUGIN_DESCRIPTION: Supports transactions, row-level locking,
                      and foreign keys
  PLUGIN_LICENSE: GPL
  LOAD_OPTION: ON
...

```

- **SHOW PLUGINS** ステートメントは、ロードされている各プラグインの行を表示します。Library 値が NULL のプラグインは組み込み型であり、アンロードできません。

```

mysql> SHOW PLUGINS\G
***** 1. row *****
  Name: binlog
  Status: ACTIVE
  Type: STORAGE ENGINE
  Library: NULL
  License: GPL
...
***** 10. row *****
  Name: InnoDB
  Status: ACTIVE
  Type: STORAGE ENGINE
  Library: ha_innodb_plugin.so
  License: GPL
...

```

- `mysql.plugin` テーブルには、**INSTALL PLUGIN** で登録されたプラグインを表示します。このテーブルにはプラグイン名およびライブラリファイル名のみが含まれているため、**PLUGINS** テーブルまたは **SHOW PLUGINS** ステートメントほどの多くの情報は提供されません。

5.6.3 MySQL Enterprise Thread Pool

注記

MySQL Enterprise Thread Pool は、商用製品である MySQL Enterprise Edition に含まれる拡張機能です。商用製品の詳細は、<https://www.mysql.com/products/> を参照してください。

MySQL Enterprise Edition には、サーバープラグインを使用して実装される MySQL Enterprise Thread Pool が含まれています。MySQL サーバーのデフォルトのスレッド処理モデルでは、クライアント接続ごとに 1 つのスレッドを使用してステートメントが実行されます。より多くのクライアントがサーバーに接続してステートメントを実行すると、全体的なパフォーマンスが低下します。スレッドプールプラグインは、オーバーヘッドを軽減し、パフォーマンスを向上するように設計されている代替のスレッド処理モデルを提供します。このプラグインは、多数のクライアント接続に対してステートメント実行スレッドを効率的に管理することによってサーバーのパフォーマンスを向上させるスレッドプールを実装します。

スレッドプールは、接続ごとに 1 つのスレッドを使用するモデルのいくつかの問題に対処します:

- スレッドが多すぎると、高度な並列実行ワークロードで CPU キャッシュがほとんど役に立たなくなります。スレッドプールはスレッドスタックの再利用を促進し、CPU キャッシュのフットプリントを最小にします。
- 並列で実行しているスレッド数が多すぎると、コンテキストスイッチングのオーバーヘッドが高くなります。これにより、オペレーティングシステムスケジューラへの課題も示されます。スレッドプールは、アクティブスレッドの数を制御して、それが処理可能で、MySQL を実行しているサーバーホストに適切なレベルで MySQL サーバー内の並列性を維持します。
- 並列で実行するトランザクションが多すぎると、リソースの競合が増加します。InnoDB では、これにより中央の相互排他ロックの保持に費やされる時間が多くなります。スレッドプールは、あまり多く並列で実行しないように、トランザクションが開始するタイミングを制御します。

追加のリソース

[セクションA.15「MySQL 8.0 FAQ: MySQL Enterprise Thread Pool」](#)

5.6.3.1 スレッドプール要素

MySQL Enterprise Thread Pool は、次の要素で構成されます:

- プラグインライブラリファイルは、スレッドプールコードのプラグインと、スレッドプール操作に関する情報を提供するいくつかの関連モニタリングテーブルを実装します:
- MySQL 8.0.14 では、監視テーブルは「パフォーマンススキーマ」テーブルです。[セクション27.12.16「パフォーマンススキーマスレッドプールテーブル」](#)を参照してください。
- MySQL 8.0.14 より前は、監視テーブルは `INFORMATION_SCHEMA` テーブルです。[セクション26.52「INFORMATION_SCHEMA スレッドプールテーブル」](#)を参照してください。

`INFORMATION_SCHEMA` テーブルは非推奨になりました。将来のバージョンの MySQL で削除される予定です。アプリケーションは、`INFORMATION_SCHEMA` テーブルから「パフォーマンススキーマ」テーブルに移行する必要があります。たとえば、アプリケーションで次のクエリーを使用するとします:

```
SELECT * FROM INFORMATION_SCHEMA.TP_THREAD_STATE;
```

アプリケーションでは、かわりに次のクエリーを使用する必要があります:

```
SELECT * FROM performance_schema.tp_thread_state;
```

注記

すべてのモニタリングテーブルをロードしない場合、一部またはすべての MySQL Enterprise Monitor スレッドプールグラフが空になる可能性があります。

スレッドプールの仕組みの詳細については、[セクション5.6.3.3「スレッドプール操作」](#)を参照してください。

- いくつかのシステム変数がスレッドプールに関連しています。`thread_handling` システム変数は、サーバーがスレッドプールプラグインを正常にロードしたときに、`loaded-dynamically` の値になります。

その他の関連するシステム変数はスレッドプールプラグインによって実装され、有効になっていないかぎり使用できません。これらの変数の使用の詳細は、[セクション5.6.3.3「スレッドプール操作」](#) および [セクション5.6.3.4「スレッドプールのチューニング」](#)を参照してください。

- パフォーマンススキーマには、スレッドプールに関する情報を公開するインストゥルメントがあり、操作パフォーマンスの調査に使用できます。識別するには、次のクエリーを使用します:

```
SELECT * FROM performance_schema.setup_instruments  
WHERE NAME LIKE '%thread_pool%';
```

詳細については、[第27章「MySQL パフォーマンススキーマ」](#)を参照してください。

5.6.3.2 スレッドプールのインストール

このセクションでは、MySQL Enterprise Thread Pool のインストール方法について説明します。プラグインのインストールについての一般的な情報は、[セクション5.6.1「プラグインのインストールおよびアンインストール」](#)を参照してください。

サーバーで使用できるようにするには、プラグインライブラリファイルを MySQL プラグインディレクトリ (`plugin_dir` システム変数で指定されたディレクトリ) に配置する必要があります。必要に応じて、サーバーの起動時に `plugin_dir` の値を設定してプラグインディレクトリの場所を構成します。

プラグインライブラリファイルのベース名は `thread_pool` です。ファイル名の接尾辞は、プラットフォームごとに異なります (たとえば、`.so` for Unix and Unix-like systems, `.dll` for Windows)。

- [MySQL 8.0.14 でのスレッドプールのインストール](#)

- [MySQL 8.0.14 より前のスレッドプールのインストール](#)

MySQL 8.0.14 でのスレッドプールのインストール

MySQL 8.0.14 以上では、スレッドプールのモニタリングテーブルは、スレッドプールプラグインとともにロードおよびアンロードされる「パフォーマンススキーマ」テーブルです。 [INFORMATION_SCHEMA](#) バージョンのテーブルは非推奨ですが、引き続き使用できます。これらは、[MySQL 8.0.14 より前のスレッドプールのインストール](#) の手順に従ってインストールされます。

スレッドプール機能を有効にするには、`--plugin-load-add` オプションを指定してサーバーを起動し、プラグインをロードします。これを行うには、サーバー `my.cnf` ファイルに次の行を配置し、必要に応じてプラットフォームの `.so` 接尾辞を調整します:

```
[mysqld]
plugin-load-add=thread_pool.so
```

プラグインのインストールを確認するには、`INFORMATION_SCHEMA.PLUGINS` テーブルを調べるか、`SHOW PLUGINS` ステートメントを使用します ([セクション5.6.2「サーバープラグイン情報の取得」](#)を参照)。例:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
FROM INFORMATION_SCHEMA.PLUGINS
WHERE PLUGIN_NAME LIKE 'thread%';
+-----+-----+
| PLUGIN_NAME | PLUGIN_STATUS |
+-----+-----+
| thread_pool | ACTIVE       |
+-----+-----+
```

パフォーマンススキーマのモニタリングテーブルが使用可能であることを確認するには、`INFORMATION_SCHEMA.TABLES` テーブルを調べるか、`SHOW TABLES` ステートメントを使用します。例:

```
mysql> SELECT TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'performance_schema'
AND TABLE_NAME LIKE 'tp%';
+-----+
| TABLE_NAME |
+-----+
| tp_thread_group_state |
| tp_thread_group_stats |
| tp_thread_state |
+-----+
```

サーバーはスレッドプールプラグインを正常にロードすると、`thread_handling` システム変数を `loaded-dynamically` に設定します。

プラグインの初期化に失敗した場合は、サーバーエラーログで診断メッセージを確認してください。

MySQL 8.0.14 より前のスレッドプールのインストール

MySQL 8.0.14 より前では、スレッドプールモニタリングテーブルはスレッドプールプラグインとは別のプラグインであり、個別にインストールできます。

スレッドプール機能を有効にするには、`--plugin-load-add` オプションを指定してサーバーを起動し、使用するプラグインをロードします。たとえば、プラグインライブラリファイルのみを指定した場合、サーバーはそれに含まれるすべてのプラグイン (スレッドプールプラグインとすべての `INFORMATION_SCHEMA` テーブル) をロードします。これを行うには、サーバー `my.cnf` ファイルに次の行を配置し、必要に応じてプラットフォームの `.so` 接尾辞を調整します:

```
[mysqld]
plugin-load-add=thread_pool.so
```

それは、個別にスレッドプールプラグインを指定して、それらをすべてロードするのと同様です。

```
[mysqld]
plugin-load-add=thread_pool=thread_pool.so
```

```
plugin-load-add=tp_thread_state=thread_pool.so
plugin-load-add=tp_thread_group_state=thread_pool.so
plugin-load-add=tp_thread_group_stats=thread_pool.so
```

必要な場合、ライブラリアイファイルから個々のプラグインをロードできます。スレッドプールプラグインをロードするが、`INFORMATION_SCHEMA` テーブルはロードしない場合、次のようなオプションを使用します。

```
[mysqld]
plugin-load-add=thread_pool=thread_pool.so
```

スレッドプールプラグインと `TP_THREAD_STATE INFORMATION_SCHEMA` テーブルのみをロードするには、次のようなオプションを使用します：

```
[mysqld]
plugin-load-add=thread_pool=thread_pool.so
plugin-load-add=tp_thread_state=thread_pool.so
```

プラグインのインストールを確認するには、`INFORMATION_SCHEMA.PLUGINS` テーブルを調べるか、`SHOW PLUGINS` ステートメントを使用します (セクション5.6.2「サーバープラグイン情報の取得」を参照)。例：

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
FROM INFORMATION_SCHEMA.PLUGINS
WHERE PLUGIN_NAME LIKE 'thread%' OR PLUGIN_NAME LIKE 'tp%';
+-----+-----+
| PLUGIN_NAME      | PLUGIN_STATUS |
+-----+-----+
| thread_pool      | ACTIVE        |
| TP_THREAD_STATE  | ACTIVE        |
| TP_THREAD_GROUP_STATE | ACTIVE        |
| TP_THREAD_GROUP_STATS | ACTIVE        |
+-----+-----+
```

サーバーはスレッドプールプラグインを正常にロードすると、`thread_handling` システム変数を `loaded-dynamically` に設定します。

プラグインの初期化に失敗した場合は、サーバーエラーログで診断メッセージを確認してください。

5.6.3.3 スレッドプール操作

スレッドプールは、それぞれクライアント接続のセットを管理するいくつかのスレッドグループから構成されます。接続が確立されると、スレッドプールはラウンドロビン方式でそれらをスレッドグループに割り当てます。

スレッドプールは、その操作の構成に使用できるシステム変数を公開します：

- `thread_pool_algorithm`: スケジューリングに使用する並列性アルゴリズム。
- `thread_pool_high_priority_connection`: セッションのステートメント実行のスケジューリング方法。
- `thread_pool_max_active_query_threads`: 許可するグループ当たりのアクティブスレッド数。
- `thread_pool_max_unused_threads`: 許可するスリープ中のスレッド数。
- `thread_pool_prio_kickup_timer`: スレッドプールが、実行を待機しているステートメントを低優先度キューから高優先度キューに移動するまでの時間。
- `thread_pool_size`: スレッドプール内のスレッドグループの数。これはスレッドプールのパフォーマンスを制御するもっとも重要なパラメータです。
- `thread_pool_stall_limit`: 実行中のステートメントが停滞しているとみなされるまでの時間。

スレッドグループの数を構成するには、`thread_pool_size` システム変数を使用します。グループのデフォルトの数は 16 です。この変数の設定のガイドラインについては、セクション5.6.3.4「スレッドプールのチューニング」を参照してください。

グループあたりのスレッドの最大数は 4096 (または 1 つのスレッドが内部で使用される一部のシステムでは 4095) です。

スレッドプールは接続とスレッドを区別するため、接続と、それらの接続から受信したステートメントを実行するスレッド間に固定の関係はありません。これは、特定のスレッドがその接続からすべてのステートメントを実行するうちに、あるスレッドを1つの接続に関連付けるデフォルトのスレッド処理モデルとは異なります。

デフォルトでは、スレッドプールは各グループで一度に最大1つのスレッドを実行しようとしませんが、最高のパフォーマンスを得るために一時的に実行できるスレッドが増える場合があります:

- 各スレッドグループには、グループに割り当てられた接続からの着信ステートメントを待機するリスナースレッドがあります。ステートメントが到着すると、スレッドグループはその実行をただちに開始するか、あとで実行するためにキューに入れます。
- 即時の実行は、ステートメントが受信した唯一のもので、キューに入れられていたり、現在実行していたりするステートメントがない場合に行われます。
- キューイングは、ステートメントの実行をすぐに開始できない場合に行われます。
- 即時実行が発生すると、リスナースレッドによって実行されます。(つまり、グループ内に一時的に待機しているスレッドがなくなります。)ステートメントがすぐに終了すると、実行中のスレッドがステートメントの待機に戻ります。そうでない場合、スレッドプールはステートメントを停滞中とみなし、別のスレッド(必要に応じて作成して)をリスナースレッドとして開始します。スレッドグループが停滞中のステートメントによってブロックされないように、スレッドプールには、スレッドグループ状態を定期的にモニターするバックグラウンドスレッドがあります。

待機中のスレッドを使用して、ただちに開始できるステートメントを実行することによって、ステートメントがすぐに終了した場合、追加のスレッドを作成する必要はありません。これにより、同時スレッド数が少ない場合に、もっとも効率的な実行が可能になります。

スレッドプールプラグインが開始すると、それはグループあたり1つのスレッド(リスナースレッド)に加えてバックグラウンドスレッドを作成します。ステートメントを実行するための必要に応じて、追加のスレッドが作成されます。

- `thread_pool_stall_limit` システム変数の値は、先述の項目の「すぐに終了する」の意味を決定します。スレッドが停滞中とみなされるまでのデフォルトの時間は60ミリ秒ですが、6秒まで設定できます。このパラメータは、サーバーのワークロードに適切なバランスがとれるように構成できます。待機の値が短いと、スレッドはよりすみやかに開始できます。短い値はデッドロック状況を回避により適しています。長い待機の値は、長時間実行するステートメントを含むワークロードで有用で、現在のステートメントの実行時に多数の新しいステートメントが開始しないようにします。
- `thread_pool_max_active_query_threads` が0の場合、グループ当たりのアクティブスレッドの最大数を決定するために説明したように、デフォルトのアルゴリズムが適用されます。デフォルトのアルゴリズムでは、中断されたスレッドが考慮され、よりアクティブなスレッドが一時的に許可される場合があります。`thread_pool_max_active_query_threads` が0より大きい場合、グループ当たりのアクティブスレッド数に制限が設定されます。
- スレッドプールは、同時の短時間実行ステートメントの数を制限することに焦点を合わせています。実行中のステートメントが停滞時間に達する前に、ほかのステートメントの実行の開始を妨げます。ステートメントが停滞時間を過ぎて実行している場合、それは続行が許可されますが、ほかのステートメントの開始は妨げられなくなります。このように、スレッドプールは、各スレッドグループに、複数の長時間実行ステートメントがあっても、複数の短時間実行ステートメントがないように努めます。必要な待機時間に対する制限がないため、長時間実行ステートメントによって、ほかのステートメントの実行が妨げられることは望ましくありません。たとえば、レプリケーションソースサーバーでは、バイナリロギングイベントをレプリカに送信しているスレッドは事実上永久に実行されません。
- ステートメントはディスク I/O 操作またはユーザーレベルロック(行ロックまたはテーブルロック)を検出するとブロックされます。ブロックによって、スレッドグループは使用されなくなることがあるため、スレッドプールがこのグループで新しいスレッドをただちに開始して、別のステートメントを実行できるようにするため、スレッドプールへのコールバックがあります。ブロックされたスレッドが返されると、スレッドプールはそれをすぐに再開することを許可します。
- 優先度が高いキューと優先度が低いキューの2つのキューがあります。トランザクションの最初のステートメントは優先度が低いキューに入ります。トランザクションの後続のステートメントは、トランザクションが進行中(そのステートメントが実行を開始している)場合、優先度が高いキューに入れられ、そうでない場合は優先度が低い

キューに入れられます。キューの割り当ては、`thread_pool_high_priority_connection` システム変数を有効にすることによって影響を受けることがあります。これにより、セッションのすべてのキューに入れられているステートメントが優先度の高いキューに入れられます。

非トランザクションストレージエンジンまたは `autocommit` が有効にされている場合のトランザクションエンジンのステートメントは、この場合に各ステートメントがトランザクションであるため、優先度の低いステートメントとして扱われます。そのため、`InnoDB` テーブルと `MyISAM` テーブルに対するステートメントを組み合わせると、`autocommit` が有効にされていないかぎり、スレッドプールは `MyISAM` に対するステートメントより、`InnoDB` に対するステートメントを優先します。`autocommit` を有効にすると、すべてのステートメントの優先度が低くなります。

- スレッドグループが実行のためにキューに入れられているステートメントを選択する場合、まず優先度の高いキューを調べて、次に優先度が低いキューを調べます。ステートメントが見つかった場合、そのキューからそれが削除され、実行が開始されます。
- ステートメントが優先度の低いキューに長くとどまりすぎた場合、スレッドプールは優先度の高いキューに移動します。`thread_pool_prio_kickup_timer` システム変数の値は、移動までの時間を制御します。スレッドグループごとに、10ms 当たり最大 1 つのステートメント (100/秒) が優先度の低いキューから優先度の高いキューに移動されません。
- スレッドプールは、CPU キャッシュの使用を大幅に効率化するために、もっともアクティブなスレッドを再利用します。これは、パフォーマンスに大きな影響を与える小さな調整です。
- スレッドがユーザー接続からステートメントを実行している間、パフォーマンススキーマインストゥルメンテーションは、ユーザー接続にスレッドアクティビティを報告します。それ以外の場合、パフォーマンススキーマはアクティビティをスレッドプールに報告します。

これは、スレッドグループがステートメントを実行するために複数のスレッドを開始している状況の例です。

- 1 つのスレッドがステートメントの実行を開始しますが、長時間実行しているため、停滞中とみなされます。スレッドグループは、最初のスレッドがまだ実行中であっても、別のスレッドに別のステートメントの実行の開始を許可します。
- 1 つのスレッドがステートメントの実行を開始し、その後ブロックされ、このことをスレッドプールにレポートします。スレッドグループは、別のスレッドに別のステートメントの実行の開始を許可します。
- 1 つのスレッドがステートメントの実行を開始し、ブロックされましたが、スレッドプールのコールバックによってインストゥルメントされたコードでブロックが発生していないため、ブロックされたことをレポートしません。この場合、スレッドはスレッドグループにまだ実行中であるように見えます。ステートメントが停滞中とみなされるほどブロックが長く続いた場合、グループは、別のスレッドに別のステートメントの実行の開始を許可しません。

スレッドプールは、増加する接続全体に拡張できるように設計されています。さらに、アクティブに実行しているステートメントの数を制限することから発生する可能性のあるデッドロックを回避するようにも設計されています。スレッドプールにレポートしないスレッドは、ほかのステートメントの実行を妨げないため、スレッドプールのデッドロックを引き起こすことは重要です。そのようなステートメントの例を次に示します。

- 長時間実行ステートメント。これらによって、すべてのリソースがほんの少数のステートメントで使用されることになり、ほかのすべてのステートメントのサーバーへのアクセスを妨げる可能性があります。
- バイナリログを読み取り、それをレプリカに送信するバイナリログダンプスレッド。これは、きわめて長い時間実行する長時間実行「ステートメント」の一種であり、ほかのステートメントの実行を妨げないはずですが。
- MySQL Server またはストレージエンジンによって、スレッドプールにレポートされていない、行ロック、テーブルロック、またはほかの何らかのブロックアクティビティでブロックされたステートメント。

どの場合も、デッドロックを避けるため、スレッドグループが別のステートメントの実行の開始を許可できるように、ステートメントがすぐに完了しない場合、停滞中カテゴリに移動されます。この設計により、スレッドが長時間実行するか、ブロックされた場合に、スレッドプールはスレッドを停滞中カテゴリに移動し、ステートメントの実行の残りの間、ほかのステートメントの実行を妨げません。

発生する可能性のあるスレッドの最大数は、`max_connections` と `thread_pool_size` の合計です。これは、すべての接続が実行モードにあり、グループあたりに追加のステートメントを待機する 1 つの追加スレッドが作成される状況で発生する可能性があります。これは必ずしも頻繁に発生する状態ではありませんが、理論的には可能性があります。

5.6.3.4 スレッドプールのチューニング

このセクションでは、秒あたりのトランザクション数などのメトリックを使用して測定された、最高のパフォーマンスを得るためのスレッドプールシステム変数の設定に関するガイドラインを提供します。

`thread_pool_size` はスレッドプールのパフォーマンスを制御するもっとも重要なパラメータです。それはサーバーの起動時にも設定できます。スレッドプールのテストにおける経験では、次のように示されます。

- プライマリストレージエンジンが `InnoDB` である場合、最適な `thread_pool_size` 設定は、16 から 36 の間になる可能性があり、もっとも一般的な最適な値は 24 から 36 になる傾向があります。36 を超える設定が最適であった状況はありませんでした。16 未満の値が最適である特殊なケースがある場合もあります。

DBT2 や Sysbench などのワークロードの場合、`InnoDB` の最適な値は通常 36 くらいであるようです。著しく書き込みの多いワークロードでは、最適な設定はもっと少ない可能性があります。

- プライマリストレージエンジンが `MyISAM` である場合、`thread_pool_size` 設定はかなり小さくする必要があります。最適なパフォーマンスは、多くの場合、4 から 8 の値で確認できます。値を大きくすると、パフォーマンスにややマイナスでも劇的な影響を与える傾向はありません。

もう 1 つのシステム変数 `thread_pool_stall_limit` はブロックされたステートメントと長時間実行ステートメントの処理に重要です。MySQL Server をブロックするすべての呼び出しがスレッドプールにレポートされる場合、実行スレッドがブロックされるといつでもわかります。ただし、これは常には当てはまらないことがあります。たとえば、ブロックはスレッドプールコールバックによってインストールされていないコードで発生する可能性があります。そのような場合、スレッドプールはブロックされているように見えるスレッドを識別する必要があります。これは、`thread_pool_stall_limit` システム変数を使用してチューニングできるタイムアウトを使用して行われます。この値は 10ms 単位で測定されます。このパラメータにより、サーバーは完全にブロックされることはありません。`thread_pool_stall_limit` の値は、デッドロックされたサーバーのリスクを回避するため、6 秒の上限があります。

`thread_pool_stall_limit` により、スレッドプールは長時間実行ステートメントを処理することもできます。長期間実行するステートメントがスレッドグループをブロックすることを許可された場合、グループに割り当てられるその他のすべての接続はブロックされ、長期間実行するステートメントが完了するまで実行を開始できません。最悪の場合、これには数時間または数日かかることもあります。

`thread_pool_stall_limit` の値は、その値より長く実行するステートメントが停滞中とみなされるように選択する必要があります。停滞中のステートメントは、追加のコンテキストスイッチと場合によっては追加のスレッド作成が必要であるため、大量の追加のオーバーヘッドを生成します。一方、`thread_pool_stall_limit` パラメータの設定が高すぎるということは、長時間実行されるステートメントが必要以上に長く実行されるステートメントをブロックすることを意味します。待機の値が短いと、スレッドはよりすみやかに開始できます。短い値はデッドロック状況を回避により適しています。長い待機の値は、長時間実行するステートメントを含むワークロードで有用で、現在のステートメントの実行時に多数の新しいステートメントが開始しないようにします。

サーバーに負荷がかかっている場合でも、サーバーはステートメントの 99.9% が 100 ミリ秒以内に完了するワークロードを実行しており、残りのステートメントが 100 ミリ秒から 2 時間の間でまったく均等に分散してかかるものとします。この場合、`thread_pool_stall_limit` を 10 (10× 10ms = 100ms) に設定すると意味があります。デフォルト値の 6 (60ms) は、主に非常に単純なステートメントを実行するサーバーに適しています。

`thread_pool_stall_limit` パラメータは、サーバーのワークロードに対して適切なバランスをとることができるように、実行時に変更できます。`tp_thread_group_stats` テーブルが有効になっていると仮定すると、次のクエリーを使用して、停止した実行済のステートメントの割合を判断できます:

```
SELECT SUM(STALLED_QUERIES_EXECUTED) / SUM(QUERIES_EXECUTED)
FROM performance_schema.tp_thread_group_stats;
```

この数値は可能なかぎり小さくする必要があります。ステートメントの停滞の可能性を削減するには、`thread_pool_stall_limit` の値を増やします。

ステートメントが到着したときに、それが実際に実行を開始するまで、遅延できる最大の時間はどれくらいですか。次の条件が当てはまるとします。

- 優先度が低いキューに 200 ステートメントが入れられています。
- 優先度が高いキューに 10 ステートメントが入れられています。
- `thread_pool_prio_kickup_timer` は 10000 (10 秒) に設定されています。

- `thread_pool_stall_limit` は 100 (1 秒) に設定されています。

最悪の場合、10 個の優先度の高いステートメントは長時間実行し続ける 10 個のトランザクションを表します。したがって、最悪の場合、ステートメントは常に実行を待機しているステートメントが含まれているため、優先度の高いキューに移動できません。10 秒後、新しいステートメントは優先度の高いキューに移動される資格を得ます。ただし、それが移動される前に、その前のすべてのステートメントも移動される必要があります。優先度の高いキューに移動されるのは、1 秒あたり最大 100 ステートメントであるため、これはさらに 2 秒かかる可能性があります。ステートメントが優先度の高いキューに到達したときに、多くの長時間実行ステートメントがその前にある可能性があります。最悪の場合、これらのすべてが停止し、次のステートメントが優先度の高いキューから取得される前に各ステートメントに 1 秒が必要になります。したがって、このシナリオでは、新しいステートメントの実行が開始されるまで 222 秒かかります。

この例では、アプリケーションの最悪のケースを示しています。その処理方法はアプリケーションに依存します。アプリケーションの応答時間に対する要件が高い場合、おそらくそれ自体で高いレベルでユーザーを制限するはずですが。そうでない場合は、スレッドプール構成パラメータを使用して、何らかの最大待機時間を設定できます。

5.6.4 リライタクエリーリライトプラグイン

MySQL は、サーバーが SQL ステートメントを実行する前に、サーバーが受け取った SQL ステートメントを調査して変更できるクエリーリライトプラグインをサポートしています。 [Query Rewrite Plugins](#) を参照してください。

MySQL ディストリビューションには、[Rewriter](#) という名前の解析後クエリーリライトプラグインと、プラグインおよびその関連要素をインストールするためのスクリプトが含まれています。これらの要素は連携して、ステートメントのリライト機能を提供します:

- [Rewriter](#) という名前のサーバー側プラグインはステートメントを検査し、リライト規則のメモリー内キャッシュに基づいてステートメントを書き換えることができます。
- 次のステートメントはリライトの対象となります:
 - MySQL 8.0.12 の時点: [SELECT](#), [INSERT](#), [REPLACE](#), [UPDATE](#) および [DELETE](#)。
 - MySQL 8.0.12 より前: [SELECT](#) only.

スタンドアロンステートメントおよびプリペアドステートメントはリライトの対象となります。ビュー定義またはストアードプログラム内で発生するステートメントは、書き換えの対象にはなりません。

- [Rewriter](#) プラグインは、[rewrite_rules](#) という名前のテーブルを含む [query_rewrite](#) という名前のデータベースを使用します。このテーブルは、ステートメントを書き換えるかどうかを決定するためにプラグインが使用するルールの永続ストレージを提供します。ユーザーは、このテーブルに格納されているルールのセットを変更することによって、プラグインと通信します。プラグインは、テーブル行の [message](#) カラムを設定することによってユーザーと通信します。
- [query_rewrite](#) データベースには、ルールテーブルの内容をプラグインにロードする [flush_rewrite_rules\(\)](#) というストアードプロシージャが含まれています。
- [flush_rewrite_rules\(\)](#) ストアドプロシージャでは、[load_rewrite_rules\(\)](#) という名前のユーザー定義関数が使用されません。
- [Rewriter](#) プラグインは、実行時の操作情報を提供するプラグイン構成およびステータス変数を有効にするシステム変数を公開します。

次の各セクションでは、[Rewriter](#) プラグインのインストールおよび使用方法について説明し、関連する要素のリファレンス情報を提供します。

5.6.4.1 リライタのクエリーリライトプラグインのインストールまたはアンインストール

注記

インストールされている場合、[Rewriter](#) プラグインは無効になっていても多少のオーバーヘッドを伴います。このオーバーヘッドを回避するには、使用する予定がないがぎり、プラグインをインストールしないでください。

Rewriter クエリーリライトプラグインをインストールまたはアンインストールするには、MySQL インストールの `share` ディレクトリにある適切なスクリプトを選択します:

- `install_rewriter.sql`: **Rewriter** プラグインとその関連要素をインストールするには、このスクリプトを選択します。
- `uninstall_rewriter.sql`: **Rewriter** プラグインとその関連要素をアンインストールするには、このスクリプトを選択します。

選択したスクリプトを次のように実行します:

```
shell> mysql -u root -p < install_rewriter.sql
Enter password: (enter root password here)
```

この例では、`install_rewriter.sql` インストールスクリプトを使用します。プラグインをアンインストールする場合は、`uninstall_rewriter.sql` に置き換えます。

インストールスクリプトを実行すると、プラグインがインストールされて有効になります。これを確認するには、サーバーに接続して次のステートメントを実行します:

```
mysql> SHOW GLOBAL VARIABLES LIKE 'rewriter_enabled';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| rewriter_enabled | ON |
+-----+-----+
```

使用手順については、[セクション5.6.4.2「リライタクエリーリライトプラグインの使用」](#)を参照してください。参照情報については、[セクション5.6.4.3「リライタクエリーリライトプラグインリファレンス」](#)を参照してください。

5.6.4.2 リライタクエリーリライトプラグインの使用

プラグインを有効または無効にするには、`rewriter_enabled` システム変数を有効または無効にします。デフォルトでは、**Rewriter** プラグインはインストール時に有効になります ([セクション5.6.4.1「リライタのクエリーリライトプラグインのインストールまたはアンインストール」](#)を参照)。プラグインの初期状態を明示的に設定するには、サーバーの起動時に変数を設定します。たとえば、オプションファイルでプラグインを有効にするには、次の行を使用します:

```
[mysqld]
rewriter_enabled=ON
```

実行時にプラグインを有効または無効にすることもできます:

```
SET GLOBAL rewriter_enabled = ON;
SET GLOBAL rewriter_enabled = OFF;
```

Rewriter プラグインが有効になっていると仮定すると、サーバーによって受信されたリライト可能な各ステートメントを調査し、場合によっては変更します。プラグインは、`query_rewrite` データベースの `rewrite_rules` テーブルからロードされるリライトルールのインメモリーキャッシュに基づいてステートメントをリライトするかどうかを決定します。

次のステートメントはリライトの対象となります:

- MySQL 8.0.12 の時点: `SELECT`, `INSERT`, `REPLACE`, `UPDATE` および `DELETE`。
- MySQL 8.0.12 より前: `SELECT` only.

スタンドアロンステートメントおよびプリペアドステートメントはリライトの対象となります。ビュー定義またはストアドプログラム内で発生するステートメントは、書き換えの対象にはなりません。

- [リライトルールの追加](#)
- [ステートメント照合の仕組み](#)
- [プリペアドステートメントのリライト](#)
- [リライタプラグインの操作情報](#)

- リライクエリーリライトプラグインでの文字セットの使用

リライトルールの追加

Rewriter プラグインのルールを追加するには、`rewrite_rules` テーブルに行を追加し、`flush_rewrite_rules()` ストアドプロシージャを起動してテーブルからプラグインにルールをロードします。次の例では、単一のリテラル値を選択するステートメントを照合する単純なルールを作成します:

```
INSERT INTO query_rewrite.rewrite_rules (pattern, replacement)
VALUES('SELECT ?', 'SELECT ? + 1');
```

結果のテーブルの内容は次のようになります:

```
mysql> SELECT * FROM query_rewrite.rewrite_rules\G
***** 1. row *****
      id: 1
      pattern: SELECT ?
      pattern_database: NULL
      replacement: SELECT ? + 1
      enabled: YES
      message: NULL
      pattern_digest: NULL
      normalized_pattern: NULL
```

このルールは、照合する `SELECT` ステートメントを示すパターンテンプレートと、照合ステートメントのリライト方法を示す置換テンプレートを指定します。ただし、`rewrite_rules` テーブルにルールを追加するだけでは、**Rewriter** プラグインでルールを使用できません。テーブルの内容をプラグインインメモリーキャッシュにロードするには、`flush_rewrite_rules()` を起動する必要があります:

```
mysql> CALL query_rewrite.flush_rewrite_rules();
```

ヒント

リライトルールが正しく機能していないように見える場合は、`flush_rewrite_rules()` をコールしてルールテーブルをリロードしたことを確認してください。

プラグインは、ルールテーブルから各ルールを読み取るときに、パターンおよびダイジェストハッシュ値から正規化された(ステートメントダイジェスト)フォームを計算し、それらを使用して `normalized_pattern` および `pattern_digest` カラムを更新します:

```
mysql> SELECT * FROM query_rewrite.rewrite_rules\G
***** 1. row *****
      id: 1
      pattern: SELECT ?
      pattern_database: NULL
      replacement: SELECT ? + 1
      enabled: YES
      message: NULL
      pattern_digest: d1b44b0c19af710b5a679907e284acd2ddc285201794bc69a2389d77baedddae
      normalized_pattern: select ?
```

ステートメントダイジェスト、正規化されたステートメント、およびダイジェストハッシュ値については、[セクション27.10「パフォーマンススキーマのステートメントダイジェストとサンプリング」](#)を参照してください。

なんらかのエラーが原因でルールをロードできない場合、`flush_rewrite_rules()` をコールするとエラーが発生します:

```
mysql> CALL query_rewrite.flush_rewrite_rules();
ERROR 1644 (45000): Loading of some rule(s) failed.
```

これが発生すると、プラグインはルール行の `message` カラムにエラーメッセージを書き込み、問題を伝達します。`NULL message` 以外のカラム値を持つ行の `rewrite_rules` テーブルをチェックして、どのような問題が存在するかを確認します。

パターンは、プリアドステートメントと同じ構文を使用します([セクション13.5.1「PREPARE ステートメント」](#)を参照)。パターンテンプレート内では、`?` 文字はデータ値と一致するパラメータマーカースとして機能します。パラメータマーカースは、SQL キーワードや識別子などではなく、データ値を指定するべき場所にしか使用できません。`?` 文字を引用符で囲まないでください。

パターンと同様に、置換には ? 文字を含めることができます。パターンテンプレートに一致するステートメントの場合、プラグインはそれを書き換え、置換内の ? パラメータマーカを、パターン内の対応するマーカに一致するデータ値を使用して置き換えます。結果は完全なステートメントの文字列になります。プラグインはサーバーに解析を要求し、書き換えられたステートメントの表現として結果をサーバーに返します。

ルールを追加してロードした後、ステートメントがルールパターンと一致するかどうかに応じてリライトが行われるかどうかを確認します:

```
mysql> SELECT PI();
+-----+
| PI() |
+-----+
| 3.141593 |
+-----+
1 row in set (0.01 sec)
```

```
mysql> SELECT 10;
+-----+
| 10 + 1 |
+-----+
| 11 |
+-----+
1 row in set, 1 warning (0.00 sec)
```

最初の **SELECT** ステートメントではリライトは行われませんが、次のステートメントでは行われます。2 番目のステートメントは、**Rewriter** プラグインがステートメントを書き換えると警告メッセージを生成することを示しています。メッセージを表示するには、**SHOW WARNINGS** を使用します:

```
mysql> SHOW WARNINGS\SIG
***** 1. row *****
Level: Note
Code: 1105
Message: Query 'SELECT 10' rewritten to 'SELECT 10 + 1' by a query rewrite plugin
```

ステートメントを同じタイプのステートメントにリライトする必要はありません。次の例では、**DELETE** ステートメントを **UPDATE** ステートメントにリライトするルールをロードします:

```
INSERT INTO query_rewrite.rewrite_rules (pattern, replacement)
VALUES('DELETE FROM db1.t1 WHERE col = ?',
      'UPDATE db1.t1 SET col = NULL WHERE col = ?');
CALL query_rewrite.flush_rewrite_rules();
```

既存のルールを有効または無効にするには、その **enabled** カラムを変更し、プラグインにテーブルをリロードします。ルール 1 を無効にするには:

```
UPDATE query_rewrite.rewrite_rules SET enabled = 'NO' WHERE id = 1;
CALL query_rewrite.flush_rewrite_rules();
```

これにより、ルールをテーブルから削除せずに非アクティブ化できます。

ルール 1 を再度有効にするには:

```
UPDATE query_rewrite.rewrite_rules SET enabled = 'YES' WHERE id = 1;
CALL query_rewrite.flush_rewrite_rules();
```

rewrite_rules テーブルには、**Rewriter** がデータベース名で修飾されていないテーブル名の照合に使用する **pattern_database** カラムが含まれています:

- ステートメントの修飾テーブル名は、対応するデータベース名とテーブル名が同一の場合、パターンの修飾名と一致します。
- ステートメント内の修飾されていないテーブル名は、デフォルトのデータベースが **pattern_database** と同じで、テーブル名が同一の場合にのみ、パターン内の修飾されていない名前と一致します。

appdb.users という名前のテーブルに **id** という名前のカラムがあり、アプリケーションが次のいずれかの形式のクエリーを使用してテーブルから行を選択するとします。2 番目の形式は、**appdb** がデフォルトデータベースの場合にのみ使用できます:

```
SELECT * FROM users WHERE appdb.id = id_value;
```

```
SELECT * FROM users WHERE id = id_value;
```

また、`id` カラムの名前が `user_id` に変更されたとします (場合によっては、テーブルを変更して別のタイプの ID を追加する必要があり、`id` カラムが表す ID のタイプをより具体的に指定する必要があります)。

この変更は、アプリケーションが `WHERE` 句で `id` ではなく `user_id` を参照する必要があることを意味します。ただし、生成される `SELECT` クエリーを変更するために書き込めない古いアプリケーションがある場合は、正しく機能しなくなります。`Rewriter` プラグインはこの問題を解決できます。テーブル名を修飾するかどうかに関係なくステートメントを照合およびリライトするには、次の 2 つのルールを追加してルールテーブルをリロードします:

```
INSERT INTO query_rewrite.rewrite_rules
(pattern, replacement) VALUES(
'SELECT * FROM appdb.users WHERE id = ?',
'SELECT * FROM appdb.users WHERE user_id = ?'
);
INSERT INTO query_rewrite.rewrite_rules
(pattern, replacement, pattern_database) VALUES(
'SELECT * FROM users WHERE id = ?',
'SELECT * FROM users WHERE user_id = ?',
'appdb'
);
CALL query_rewrite.flush_rewrite_rules();
```

`Rewriter` では、最初のルールを使用して、修飾テーブル名を使用するステートメントを照合します。デフォルトデータベースが `appdb` (`pattern_database` の値) の場合のみ、秒を使用して、修飾されていない名前を使用したステートメントを照合します。

ステートメント照合の仕組み

`Rewriter` プラグインは、ステートメントダイジェストとダイジェストハッシュ値を使用して、着信ステートメントを段階的なリライトルールと照合します。`max_digest_length` システム変数は、ステートメントダイジェストの計算に使用されるバッファのサイズを決定します。値が大きいくほど、長いステートメントを区別するダイジェストの計算が可能になります。値が小さいほどメモリ使用量は少なくなりますが、同じダイジェスト値と競合する長いステートメントの可能性が高くなります。

プラグインは、次のように各ステートメントをリライト規則と照合します:

1. ステートメントダイジェストハッシュ値を計算し、ルールダイジェストハッシュ値と比較します。これは誤検出の対象ですが、迅速な拒否テストとして機能します。
2. ステートメントダイジェストハッシュ値がパターンダイジェストハッシュ値と一致する場合は、ステートメントの正規化された (ステートメントダイジェスト) 形式を一致ルールパターンの正規化された形式と照合します。
3. 正規化されたステートメントがルールと一致する場合は、ステートメントのリテラル値とパターンを比較します。パターン内の `?` 文字は、ステートメント内の任意のリテラル値と一致します。ステートメントがステートメントを準備する場合、パターン内の `?` もステートメント内の `?` と一致します。それ以外の場合、対応するリテラルは同じである必要があります。

複数のルールがステートメントに一致する場合は、プラグインがステートメントを書き換えるために使用する非決定的です。

パターンに置換より多くのマーカーが含まれている場合、プラグインは余分なデータ値を破棄します。パターンに含まれるマーカーが置換より少ない場合は、エラーになります。プラグインは、ルールテーブルがロードされたときにこれに気づき、問題を伝えるためにルール行の `message` カラムにエラーメッセージを書き込み、`Rewriter_reload_error` ステータス変数を `ON` に設定します。

プリペアドステートメントのリライト

プリペアドステートメントは、後で実行されるのではなく、解析時 (つまり準備時) にリライトされます。

プリペアドステートメントは、パラメータマーカーとして `?` 文字を含むことができるという点で、プリペアドステートメントと異なります。プリペアドステートメントの `?` と一致させるには、`Rewriter` パターンの同じ場所に `?` が含まれている必要があります。リライトルールに次のパターンがあるとします:

```
SELECT ?, 3
```

次のテーブルに、いくつかの準備済 `SELECT` ステートメントと、ルールパターンがそれらに一致するかどうかを示します。

プリペアドステートメント	パターンがステートメントと一致するかどうか
<code>PREPARE s AS 'SELECT 3, 3'</code>	はい
<code>PREPARE s AS 'SELECT ?, 3'</code>	はい
<code>PREPARE s AS 'SELECT 3, ?'</code>	いいえ
<code>PREPARE s AS 'SELECT ?, ?'</code>	いいえ

リライタプラグインの操作情報

`Rewriter` プラグインは、いくつかのステータス変数を使用して、その操作に関する情報を使用可能にします:

```
mysql> SHOW GLOBAL STATUS LIKE 'Rewriter%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Rewriter_number_loaded_rules | 1 |
| Rewriter_number_reloads | 5 |
| Rewriter_number_rewritten_queries | 1 |
| Rewriter_reload_error | ON |
+-----+-----+
```

これらの変数の説明については、[リライタのクエリーリライトプラグインステータス変数](#)を参照してください。

`flush_rewrite_rules()` ストアドプロシージャをコールしてルールテーブルをロードすると、一部のルールでエラーが発生した場合、`CALL` ステートメントによってエラーが生成され、プラグインによって `Rewriter_reload_error` ステータス変数が `ON` に設定されます:

```
mysql> CALL query_rewrite.flush_rewrite_rules();
ERROR 1644 (45000): Loading of some rule(s) failed.

mysql> SHOW GLOBAL STATUS LIKE 'Rewriter_reload_error';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Rewriter_reload_error | ON |
+-----+-----+
```

この場合、`rewrite_rules` テーブルで `NULL` 以外の `message` カラム値を持つ行をチェックして、どのような問題が存在するかを確認します。

リライタプラグインでの文字セットの使用

`rewrite_rules` テーブルが `Rewriter` プラグインにロードされると、プラグインは `character_set_client` システム変数の現在のグローバル値を使用してステートメントを解釈します。グローバル `character_set_client` 値が後で変更された場合、ルールテーブルをリロードする必要があります。

クライアントは、ルールテーブルがロードされたときのグローバル値と同一のセッション `character_set_client` 値を持つ必要があります。そうしないと、そのクライアントに対してルール照合が機能しません。

5.6.4.3 リライタクエリーリライトプラグインリファレンス

次の説明は、`Rewriter` クエリーリライトプラグインに関連付けられたこれらの要素への参照として機能します:

- `query_rewrite` データベースの `Rewriter` ルールテーブル
- `Rewriter` プロシージャおよび関数
- `Rewriter` のシステム変数とステータス変数

リライタクエリーリライトプラグインルールテーブル

`query_rewrite` データベースの `rewrite_rules` テーブルは、`Rewriter` プラグインがステートメントをリライトするかどうかを決定するために使用するルールの永続記憶域を提供します。

ユーザーは、このテーブルに格納されているルールのセットを変更することによって、プラグインと通信します。プラグインは、テーブル `message` カラムを設定することによって、ユーザーに情報を伝達します。

注記

ルールテーブルは、`flush_rewrite_rules` ストアドプロシージャによってプラグインにロードされます。最新のテーブルの変更後にプロシージャがコールされていないかぎり、テーブルの内容がプラグインが使用しているルールのセットに対応しているとはかぎりません。

`rewrite_rules` テーブルには、次のカラムがあります:

- `id`

ルール ID。このカラムはテーブルの主キーです。この ID を使用して、任意のルールを一意に識別できます。

- `pattern`

ルールが一致するステートメントのパターンを示すテンプレート。? を使用して、データ値と一致するパラメータマーカ―を表します。

- `pattern_database`

ステートメントの未修飾のテーブル名を照合するために使用されるデータベース。ステートメントの修飾テーブル名は、対応するデータベース名とテーブル名が同一の場合、パターンの修飾名と一致します。ステートメント内の修飾されていないテーブル名は、デフォルトのデータベースが `pattern_database` と同じで、テーブル名が同一の場合にのみ、パターン内の修飾されていない名前と一致します。

- `replacement`

`pattern` のカラム値と一致するステートメントのリライト方法を示すテンプレート。? を使用して、データ値と一致するパラメータマーカ―を表します。リライトされたステートメントでは、`pattern` の対応するマーカ―に一致するデータ値を使用して、`replacement` の ? パラメータマーカ―がプラグインによって置換されます。

- `enabled`

ルールが有効かどうか。(flush_rewrite_rules() ストアドプロシージャの起動によって実行される) ロード操作では、このカラムが `YES` の場合にのみ、ルールがテーブルから `Rewriter` インメモリキャッシュにロードされます。

このカラムでは、ルールを削除せずに非アクティブ化できます: カラムを `YES` 以外の値に設定し、プラグインにテーブルをリロードします。

- `message`

プラグインは、ユーザーとの通信にこのカラムを使用します。ルールテーブルがメモリーにロードされたときにエラーが発生しない場合、プラグインは `message` カラムを `NULL` に設定します。 `NULL` 以外の値はエラーを示し、カラムの内容はエラーメッセージです。エラーは、次の状況で発生する可能性があります:

- パターンまたは置換のいずれかが、構文エラーを生成する不適切な SQL ステートメントです。
- 置換には、パターンより多くの ? パラメータマーカ―が含まれています。

ロードエラーが発生した場合、プラグインは `Rewriter_reload_error` ステータス変数も `ON` に設定します。

- `pattern_digest`

このカラムは、デバッグおよび診断に使用されます。ルールテーブルがメモリーにロードされたときにカラムが存在する場合、プラグインはそれをパターンダイジェストで更新します。このカラムは、一部のステートメントのリライトに失敗した理由を判断しようとする場合に役立つことがあります。

- `normalized_pattern`

このカラムは、デバッグおよび診断に使用されます。ルールテーブルがメモリーにロードされたときにカラムが存在する場合、プラグインは正規化された形式のパターンで更新します。このカラムは、一部のステートメントのリライトに失敗した理由を判断しようとする場合に役立つことがあります。

リライタのクエリーリライトプロシージャおよび関数

Rewriter プラグイン操作では、ルールテーブルをインメモリーキャッシュにロードするストアドプロシージャとヘルパーユーザー定義関数 (UDF) を使用します。通常の操作では、ユーザーはストアドプロシージャのみを起動します。UDF は、ユーザーが直接呼び出すのではなく、ストアドプロシージャによって呼び出されることを意図しています。

- [flush_rewrite_rules\(\)](#)

このストアドプロシージャは、[load_rewrite_rules\(\)](#) UDF を使用して、[rewrite_rules](#) テーブルの内容を **Rewriter** インメモリーキャッシュにロードします。

[flush_rewrite_rules\(\)](#) のコールは、**COMMIT** を暗黙的に意味します。

ルールテーブルを変更した後にこのプロシージャを起動して、プラグインが新しいテーブルの内容からキャッシュを更新するようにします。エラーが発生した場合、プラグインはテーブル内の適切なルール行の [message](#) カラムを設定し、[Rewriter_reload_error](#) ステータス変数を **ON** に設定します。

- [load_rewrite_rules\(\)](#)

この UDF は、[flush_rewrite_rules\(\)](#) ストアドプロシージャで使用されるヘルパールーチンです。

リライタのクエリーリライトプラグインのシステム変数

Rewriter クエリーリライトプラグインでは、次のシステム変数がサポートされます。これらの変数は、プラグインがインストールされている場合にのみ使用できます ([セクション5.6.4.1「リライタのクエリーリライトプラグインのインストールまたはアンインストール」](#) を参照)。

- [rewriter_enabled](#)

システム変数	rewriter_enabled
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

Rewriter クエリーリライトプラグインが有効かどうか。

- [rewriter_verbose](#)

システム変数	rewriter_verbose
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer

内部使用。

リライタのクエリーリライトプラグインステータス変数

Rewriter クエリーリライトプラグインでは、次のステータス変数がサポートされます。これらの変数は、プラグインがインストールされている場合にのみ使用できます ([セクション5.6.4.1「リライタのクエリーリライトプラグインのインストールまたはアンインストール」](#) を参照)。

- [Rewriter_number_loaded_rules](#)

Rewriter プラグインで使用するために [rewrite_rules](#) テーブルからメモリーに正常にロードされたリライトプラグインのリライトルールの数。

- [Rewriter_number_reloads](#)

[Rewriter](#) プラグインで使用されるインメモリーキャッシュに [rewrite_rules](#) テーブルがロードされた回数。

- [Rewriter_number_rewritten_queries](#)

[Rewriter](#) クエリーリライトプラグインがロードされてからリライトされたクエリーの数。

- [Rewriter_reload_error](#)

[Rewriter](#) プラグインで使用されるインメモリーキャッシュに [rewrite_rules](#) テーブルが最後にロードされたときにエラーが発生したかどうか。値が **OFF** の場合、エラーは発生していません。値が **ON** の場合、エラーが発生しました。[rewrite_rules](#) テーブルの [message](#) カラムでエラーメッセージを確認してください。

5.6.5 ddl_rewriter プラグイン

MySQL 8.0.16 以上には、サーバーが受け取る [CREATE TABLE](#) ステートメントを解析および実行する前に変更する [ddl_rewriter](#) プラグインが含まれています。プラグインは、[ENCRYPTION](#)、[DATA DIRECTORY](#) および [INDEX DIRECTORY](#) 句を削除します。これらは、暗号化されているか、テーブルがデータディレクトリ外に格納されているデータベースから作成された SQL ダンプファイルからテーブルをリストアする場合に役立ちます。たとえば、このようなダンプファイルを暗号化されていないインスタンス、またはデータディレクトリ外のパスにアクセスできない環境にリストアできます。

[ddl_rewriter](#) プラグインを使用する前に、[セクション5.6.5.1「ddl_rewriter のインストールまたはアンインストール」](#)に記載されている手順に従ってインストールします。

[ddl_rewriter](#) は、解析前にサーバーが受信した SQL ステートメントを調べ、次の条件に従ってリライトします:

- [ddl_rewriter](#) では、[CREATE TABLE](#) ステートメントのみが考慮され、入力行の先頭またはプリバードステートメントのテキストの先頭で発生するスタンドアロンステートメントである場合にのみ考慮されます。[ddl_rewriter](#) では、ストアドプログラム定義内の [CREATE TABLE](#) ステートメントは考慮されません。ステートメントは複数の行にまたがることができます。
- リライトの対象となるステートメント内では、次の句のインスタンスがリライトされ、各インスタンスが単一の領域に置き換えられます:
 - [ENCRYPTION](#)
 - [DATA DIRECTORY](#) (テーブルおよびパーティションレベル)
 - [INDEX DIRECTORY](#) (テーブルおよびパーティションレベル)
- リライトは大文字と小文字に依存しません。

[ddl_rewriter](#) がステートメントをリライトすると、警告が生成されます:

```
mysql> CREATE TABLE t (i INT) DATA DIRECTORY 'var/mysql/data';
Query OK, 0 rows affected, 1 warning (0.03 sec)

mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Note
Code: 1105
Message: Query 'CREATE TABLE t (i INT) DATA DIRECTORY 'var/mysql/data'
         rewritten to 'CREATE TABLE t (i INT) ' by a query rewrite plugin
1 row in set (0.00 sec)
```

一般クエリーログまたはバイナリログが有効になっている場合、サーバーは、[ddl_rewriter](#) による書換え後のステートメントに書き込みます。

インストールされると、[ddl_rewriter](#) はプラグインメモリーの使用状況を追跡するためにパフォーマンススキーマ [memory/rewriter/ddl_rewriter](#) インストゥルメントを公開します。[セクション27.12.18.10「メモリーサマリーテーブル」](#)を参照してください

5.6.5.1 ddl_rewriter のインストールまたはアンインストール

このセクションでは、`ddl_rewriter` プラグインをインストールまたはアンインストールする方法について説明します。プラグインのインストールについての一般的な情報は、[セクション5.6.1「プラグインのインストールおよびアンインストール」](#)を参照してください。

注記

インストールされている場合、`ddl_rewriter` プラグインは、無効になっていても最小限のオーバーヘッドを伴います。このオーバーヘッドを回避するには、そのオーバーヘッドを使用する期間にのみ `ddl_rewriter` をインストールします。

主なユースケースはダンプファイルからリストアされたステートメントの変更であるため、一般的な使用パターンは次のとおりです: 1) プラグインをインストールし、2) ダンプファイルをリストアし、3) プラグインをアンインストールします。

サーバーで使用できるようにするには、プラグインライブラリファイルを MySQL プラグインディレクトリ (`plugin_dir` システム変数で指定されたディレクトリ) に配置する必要があります。必要に応じて、サーバーの起動時に `plugin_dir` の値を設定してプラグインディレクトリの場所を構成します。

プラグインライブラリファイルのベース名は `ddl_rewriter` です。ファイル名の接尾辞は、プラットフォームごとに異なります (たとえば、`.so` for Unix and Unix-like systems, `.dll` for Windows)。

`ddl_rewriter` プラグインをインストールするには、必要に応じてプラットフォームの `.so` 接尾辞を調整して、`INSTALL PLUGIN` ステートメントを使用します:

```
INSTALL PLUGIN ddl_rewriter SONAME 'ddl_rewriter.so';
```

プラグインのインストールを確認するには、`INFORMATION_SCHEMA.PLUGINS` テーブルを調べるか、`SHOW PLUGINS` ステートメントを使用します ([セクション5.6.2「サーバープラグイン情報の取得」](#)を参照)。例:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS, PLUGIN_TYPE
FROM INFORMATION_SCHEMA.PLUGINS
WHERE PLUGIN_NAME LIKE 'ddl%';
+-----+-----+-----+
| PLUGIN_NAME | PLUGIN_STATUS | PLUGIN_TYPE |
+-----+-----+-----+
| ddl_rewriter | ACTIVE        | AUDIT       |
+-----+-----+-----+
```

前述の結果に示すように、`ddl_rewriter` は監査プラグインとして実装されます。

プラグインの初期化に失敗した場合は、サーバーエラーログで診断メッセージを確認してください。

前述のようにインストールすると、`ddl_rewriter` はアンインストールされるまでインストールされたままになります。削除するには、`UNINSTALL PLUGIN` を使用します:

```
UNINSTALL PLUGIN ddl_rewriter;
```

`ddl_rewriter` がインストールされている場合は、後続のサーバー起動に `--ddl-rewriter` オプションを使用して、`ddl_rewriter` プラグインのアクティブ化を制御できます。たとえば、プラグインが実行時に有効にならないようにするには、このオプションを使用します:

```
[mysqld]
ddl-rewriter=OFF
```

5.6.5.2 ddl_rewriter プラグインオプション

このセクションでは、`ddl_rewriter` プラグインの操作を制御するコマンドオプションについて説明します。起動時に指定された値が正しくない場合、`ddl_rewriter` プラグインが正しく初期化されず、サーバーがロードしない可能性があります。

`ddl_rewriter` プラグインのアクティブ化を制御するには、このオプションを使用します:

- `--ddl-rewriter[=value]`

コマンド行形式	<code>--ddl-rewriter[=value]</code>
導入	8.0.16

型	列挙
デフォルト値	ON
有効な値	ON OFF FORCE FORCE_PLUS_PERMANENT

このオプションは、サーバーが起動時に `ddl_rewriter` プラグインをロードする方法を制御します。プラグインが以前に `INSTALL PLUGIN` に登録されているか、`--plugin-load` または `--plugin-load-add` にロードされている場合にのみ使用できます。セクション5.6.5.1「`ddl_rewriter` のインストールまたはアンインストール」を参照してください。

セクション5.6.1「プラグインのインストールおよびアンインストール」で説明したように、オプションの値は、プラグインのロードオプションに指定可能な値のいずれかである必要があります。たとえば、`--ddl-rewriter=OFF` はサーバーの起動時にプラグインを無効にします。

5.6.6 バージョントークン

MySQL には、バージョントークンが含まれています。バージョントークンは、アプリケーションが不正または古いデータへのアクセスを防ぐために使用できるサーバートークンの作成および同期を可能にする機能です。

バージョントークンインタフェースには、次の特性があります：

- バージョントークンは、キーまたは識別子として機能する名前と値で構成されるペアです。
- バージョントークンはロックできます。アプリケーションはトークンロックを使用して、トークンが使用中で変更できないことを他の協調アプリケーションに示すことができます。
- バージョントークンリストは、サーバーごとに確立されます (たとえば、サーバー割当てまたは操作状態を指定するため)。また、サーバーと通信するアプリケーションは、サーバーの状態を示す独自のトークンリストを登録できます。アプリケーションによってサーバーに送信された SQL ステートメントが必要な状態でない場合は、エラーが発生します。これは、SQL ステートメントを受信するために必要な状態で別のサーバーを検索する必要があることをアプリケーションに伝えるシグナルです。

次の各セクションでは、バージョントークンの要素について説明し、それをインストールおよび使用方法について説明し、その要素のリファレンス情報を提供します。

5.6.6.1 バージョントークン要素

バージョントークンは、次の要素を実装するプラグインライブラリに基づいています：

- `version_tokens` という名前のサーバー側プラグインは、サーバーに関連付けられたバージョントークンのリストを保持し、ステートメント実行イベントの通知をサブスクライブします。`version_tokens` プラグインは、`audit plugin API` を使用してクライアントからの受信ステートメントをモニターし、各クライアントセッション固有のバージョントークンリストをサーバーのバージョントークンリストと照合します。一致がある場合、プラグインはステートメントを通過させ、サーバーはそのステートメントの処理を続行します。それ以外の場合、プラグインはクライアントにエラーを返し、ステートメントは失敗します。
- 一連のユーザー定義関数 (UDF) は、プラグインによって保持されるサーバーバージョントークンのリストを操作および検査するための SQL レベルの API を提供します。任意のバージョントークン UDF をコールするには、`VERSION_TOKEN_ADMIN` 権限 (または非推奨の `SUPER` 権限) が必要です。
- `version_tokens` プラグインがロードされると、`VERSION_TOKEN_ADMIN` 動的権限が定義されます。この権限は UDF のユーザーに付与できます。
- システム変数を使用すると、クライアントは必要なサーバー状態を登録するバージョントークンのリストを指定できます。クライアントがステートメントを送信したときにサーバーの状態が異なる場合、クライアントはエラーを受け取ります。

5.6.6.2 バージョントークンのインストールまたはアンインストール

注記

インストールされている場合、バージョントークンには多少のオーバーヘッドが伴います。このオーバーヘッドを回避するには、使用する予定がないかぎりインストールしないでください。

このセクションでは、プラグインおよびユーザー定義関数 (UDF) を含むプラグインライブラリファイルに実装されているバージョントークンをインストールまたはアンインストールする方法について説明します。プラグインおよび UDF のインストールまたはアンインストールの一般情報は、[セクション5.6.1「プラグインのインストールおよびアンインストール」](#) および [セクション5.7.1「ユーザー定義関数のインストールおよびアンインストール」](#) を参照してください。

サーバーで使用できるようにするには、プラグインライブラリファイルを MySQL プラグインディレクトリ (`plugin_dir` システム変数で指定されたディレクトリ) に配置する必要があります。必要に応じて、サーバーの起動時に `plugin_dir` の値を設定してプラグインディレクトリの場所を構成します。

プラグインライブラリファイルのベース名は `version_tokens` です。ファイル名の接尾辞は、プラットフォームごとに異なります (たとえば、`.so` for Unix and Unix-like systems, `.dll` for Windows)。

バージョントークンプラグインおよび UDF をインストールするには、[INSTALL PLUGIN](#) および [CREATE FUNCTION](#) ステートメントを使用し、必要に応じてプラットフォームの `.so` 接尾辞を調整します:

```
INSTALL PLUGIN version_tokens SONAME 'version_token.so';
CREATE FUNCTION version_tokens_set RETURNS STRING
  SONAME 'version_token.so';
CREATE FUNCTION version_tokens_show RETURNS STRING
  SONAME 'version_token.so';
CREATE FUNCTION version_tokens_edit RETURNS STRING
  SONAME 'version_token.so';
CREATE FUNCTION version_tokens_delete RETURNS STRING
  SONAME 'version_token.so';
CREATE FUNCTION version_tokens_lock_shared RETURNS INT
  SONAME 'version_token.so';
CREATE FUNCTION version_tokens_lock_exclusive RETURNS INT
  SONAME 'version_token.so';
CREATE FUNCTION version_tokens_unlock RETURNS INT
  SONAME 'version_token.so';
```

サーバーバージョントークンリストを管理するには UDF をインストールする必要がありますが、UDF が正しく動作しないため、プラグインもインストールする必要があります。

レプリケーションソースサーバーでプラグインと UDF が使用されている場合は、レプリケーションの問題を回避するために、それらをすべてのレプリカサーバーにインストールします。

前述のようにインストールすると、プラグインと UDF はアンインストールされるまでインストールされたままになります。これらを削除するには、[UNINSTALL PLUGIN](#) および [DROP FUNCTION](#) ステートメントを使用します:

```
UNINSTALL PLUGIN version_tokens;
DROP FUNCTION version_tokens_set;
DROP FUNCTION version_tokens_show;
DROP FUNCTION version_tokens_edit;
DROP FUNCTION version_tokens_delete;
DROP FUNCTION version_tokens_lock_shared;
DROP FUNCTION version_tokens_lock_exclusive;
DROP FUNCTION version_tokens_unlock;
```

5.6.6.3 バージョントークンの使用

バージョントークンを使用する前に、[セクション5.6.6.2「バージョントークンのインストールまたはアンインストール」](#) に記載されている手順に従ってインストールします。

バージョントークンが役立つシナリオは、MySQL サーバーのコレクションにアクセスするシステムですが、ロードバランシングの目的でそれらを監視し、負荷の変更に応じてサーバー割当てを調整することで管理する必要があります。このようなシステムは、次の要素で構成されます:

- 管理対象の MySQL サーバーのコレクション。

- サーバーと通信し、それらを高可用性グループに編成する管理アプリケーションまたは管理アプリケーション。グループは異なる目的で使用され、各グループ内のサーバーには異なる割当てがある場合があります。特定のグループ内のサーバーの割当ては、いつでも変更できます。
- サーバーにアクセスしてデータを取得および更新し、割り当てられた目的に応じてサーバーを選択するクライアントアプリケーション。たとえば、クライアントは読み取り専用サーバーに更新を送信しないでください。

バージョントークンを使用すると、クライアントが割当てについてサーバーに繰り返しクエリーすることなく、割当てに従ってサーバーアクセスを管理できます：

- 管理アプリケーションはサーバー割当てを実行し、その割当てを反映するために各サーバーにバージョントークンを確立します。アプリケーションはこの情報をキャッシュして、集中アクセスポイントを提供します。

ある時点で、管理アプリケーションがサーバー割当てを変更する必要がある場合（たとえば、書き込みを許可から読み取り専用に変更する場合）、サーバーバージョントークンリストを変更し、そのキャッシュを更新します。

- パフォーマンスを向上させるために、クライアントアプリケーションは管理アプリケーションからキャッシュ情報を取得するため、各ステートメントのサーバー割当てに関する情報を取得する必要がなくなります。クライアントは、発行するステートメントのタイプ（読み取りと書き込みなど）に基づいて、適切なサーバーを選択して接続します。
- さらに、クライアントはサーバーにクライアント固有のバージョントークンを送信して、サーバーに必要な割当てを登録します。クライアントからサーバーに送信されたステートメントごとに、サーバーは独自のトークンリストをクライアントトークンリストと比較します。サーバートークンリストに同じ値を持つクライアントトークンリストに存在するすべてのトークンが含まれている場合、一致があり、サーバーはステートメントを実行します。

一方、管理アプリケーションによってサーバー割当てとそのバージョントークンリストが変更された可能性があります。この場合、新しいサーバー割当てがクライアント要件と互換性がなくなる可能性があります。サーバーとクライアントトークンリストの間でトークンの不一致が発生し、サーバーはステートメントへの応答としてエラーを返します。これは、管理アプリケーションキャッシュからバージョントークン情報をリフレッシュし、通信する新しいサーバーを選択するようクライアントに指示します。

バージョントークンエラーを検出して新しいサーバーを選択するためのクライアント側ロジックは、様々な方法で実装できます：

- クライアントは、すべてのバージョントークン登録、不一致検出および接続切替え自体を処理できます。
- これらのアクションのロジックは、クライアントと MySQL サーバー間の接続を管理するコネクタに実装できます。このようなコネクタは、不一致エラー検出およびステートメントの再送信自体を処理したり、アプリケーションにエラーを渡してアプリケーションに残し、ステートメントを再送信したりする場合があります。

次の例は、より具体的な形式で前述の説明を示しています。

特定のサーバーでバージョントークンが初期化されると、サーバーバージョントークンリストは空になります。トークンリストのメンテナンスは、ユーザー定義関数 (UDF) をコールすることで実行されます。任意のバージョントークン UDF をコールするには、`VERSION_TOKEN_ADMIN` 権限（または非推奨の `SUPER` 権限）が必要であるため、トークンリストの変更は、その権限を持つ管理アプリケーションまたは管理アプリケーションによって実行される必要があります。

管理アプリケーションが、従業員および製品データベース（それぞれ `emp` および `prod` という名前）にアクセスするためにクライアントによってクエリーされる一連のサーバーと通信するとします。すべてのサーバーでデータ取得ステートメントの処理が許可されていますが、一部のサーバーでのみデータベースの更新が許可されています。これをデータベース固有のベースで処理するために、管理アプリケーションは各サーバーにバージョントークンのリストを確立します。特定のサーバーのトークンリストでは、トークン名はデータベース名を表し、トークン値は、データベースを読み取り専用で使用する必要があるかどうか、または読み取りおよび書き込みが可能かどうかに応じて `read` または `write` です。

クライアントアプリケーションは、システム変数を設定して、サーバーが一致する必要があるバージョントークンのリストを登録します。変数の設定はクライアント固有のベースで行われるため、クライアントごとに異なる要件を登録できます。デフォルトでは、クライアントトークンリストは空で、任意のサーバートークンリストに一致します。クライアントがトークンリストを空でない値に設定すると、サーバーバージョンのトークンリストに応じて、照合が成功または失敗する場合があります。

サーバーのバージョントークンリストを定義するために、管理アプリケーションは `version_tokens_set()` UDF を呼び出します。(後で説明するように、トークンリストを変更および表示するための UDF もあります。) たとえば、アプリケーションは次のステートメントを 3 つのサーバーのグループに送信します:

サーバー 1:

```
mysql> SELECT version_tokens_set('emp=read;prod=read');
+-----+
| version_tokens_set('emp=read;prod=read') |
+-----+
| 2 version tokens set.                    |
+-----+
```

サーバー 2:

```
mysql> SELECT version_tokens_set('emp=write;prod=read');
+-----+
| version_tokens_set('emp=write;prod=read') |
+-----+
| 2 version tokens set.                    |
+-----+
```

サーバー 3:

```
mysql> SELECT version_tokens_set('emp=read;prod=write');
+-----+
| version_tokens_set('emp=read;prod=write') |
+-----+
| 2 version tokens set.                    |
+-----+
```

いずれの場合も、トークンリストはセミコロンで区切られた `name=value` ペアのリストとして指定されます。生成されるトークンリストの値は、次のサーバー割当てになります:

- いずれかのサーバーがいずれかのデータベースの読取りを受け入れます。
- サーバー 2 のみが `emp` データベースの更新を受け入れます。
- `prod` データベースの更新を受け入れるのはサーバー 3 のみです。

管理アプリケーションは、各サーバーにバージョントークンリストを割り当てるだけでなく、サーバー割当てを反映するキャッシュも保持します。

サーバーと通信する前に、クライアントアプリケーションは管理アプリケーションに接続し、サーバー割当てに関する情報を取得します。次に、クライアントはそれらの割り当てに基づいてサーバーを選択します。クライアントが `emp` データベースで読取りと書込みの両方を実行するとします。前述の割当てに基づいて、サーバー 2 のみが修飾されます。クライアントはサーバー 2 に接続し、その `version_tokens_session` システム変数を設定してサーバー要件を登録します:

```
mysql> SET @@SESSION.version_tokens_session = 'emp=write';
```

クライアントからサーバー 2 に送信される後続のステートメントの場合、サーバーは独自のバージョントークンリストをクライアントリストと比較して、それらが一致するかどうかを確認します。その場合、ステートメントは正常に実行されます:

```
mysql> UPDATE emp.employee SET salary = salary * 1.1 WHERE id = 4981;
Query OK, 1 row affected (0.07 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT last_name, first_name FROM emp.employee WHERE id = 4981;
+-----+-----+
| last_name | first_name |
+-----+-----+
| Smith    | Abe       |
+-----+-----+
1 row in set (0.01 sec)
```

サーバとクライアントのバージョントークンリストの不一致は、次の 2 つの方法で発生します:

- `version_tokens_session` 値のトークン名がサーバートークンリストに存在しません。この場合、`ER_VTOKEN_PLUGIN_TOKEN_NOT_FOUND` エラーが発生します。
- `version_tokens_session` 値のトークン値は、サーバートークンリストの対応するトークンの値とは異なります。この場合、`ER_VTOKEN_PLUGIN_TOKEN_MISMATCH` エラーが発生します。

サーバー 2 の割り当てが変更されないかぎり、クライアントはそれを読み取りおよび書き込みに引き続き使用します。ただし、管理アプリケーションがサーバー割当てを変更して、`emp` データベースの書き込みをサーバー 2 ではなくサーバー 1 に送信する必要があるとします。これを行うには、`version_tokens_edit()` を使用して、2 つのサーバーの `emp` トークン値を変更します (また、サーバー割当てのキャッシュを更新します):

サーバー 1:

```
mysql> SELECT version_tokens_edit('emp=write');
+-----+
| version_tokens_edit('emp=write') |
+-----+
| 1 version tokens updated.         |
+-----+
```

サーバー 2:

```
mysql> SELECT version_tokens_edit('emp=read');
+-----+
| version_tokens_edit('emp=read') |
+-----+
| 1 version tokens updated.         |
+-----+
```

`version_tokens_edit()` では、サーバートークンリスト内の名前付きトークンが変更され、他のトークンは変更されません。

クライアントが次回サーバー 2 にステートメントを送信するときに、独自のトークンリストがサーバーのトークンリストと一致なくなり、エラーが発生します:

```
mysql> UPDATE emp.employee SET salary = salary * 1.1 WHERE id = 4982;
ERROR 3136 (42000): Version token mismatch for emp. Correct value read
```

この場合、クライアントは管理アプリケーションに連絡して、サーバー割当てに関する更新情報を取得し、新しいサーバーを選択して、失敗したステートメントを新しいサーバーに送信する必要があります。

注記

各クライアントは、特定のサーバーに登録されているトークンリストに従ってステートメントのみを送信することで、バージョントークンと連携する必要があります。たとえば、クライアントが `'emp=read'` のトークンリストに登録した場合、クライアントが `emp` データベースの更新を送信できないようにするためのものがバージョントークンにありません。クライアント自体がそのような処理を再試行する必要があります。

クライアントから受信したステートメントごとに、サーバーは次のように暗黙的にロックを使用します:

- クライアントトークンリストで指定されたトークン (`version_tokens_session` 値) ごとに共有ロックを取得
- サーバーとクライアントトークンリストの比較を実行
- ステートメントを実行するか、比較結果に応じてエラーを生成
- ロックの解除

サーバーは共有ロックを使用するため、ブロックせずに複数のセッションの比較を行うことができますが、サーバートークンリスト内の同じ名前のトークンを操作する前に排他ロックを取得しようとするセッションのトークンに対する変更を防止します。

前述の例では、バージョントークンプラグインライブラリに含まれているユーザー定義の一部のみを使用していますが、他にもあります。UDF のセットでは、バージョントークンのサーバーリストを操作および検査できます。UDF の別のセットでは、バージョントークンをロックおよびロック解除できます。

次の UDF を使用すると、バージョントークンのサーバーリストを作成、変更、削除および検査できます:

- `version_tokens_set()` では、現在のリストが完全に置換され、新しいリストが割り当てられます。引数は、セミコロンで区切られた `name=value` ペアのリストです。
- `version_tokens_edit()` では、現在のリストを部分的に変更できます。新しいトークンを追加したり、既存のトークンの値を変更できます。引数は、セミコロンで区切られた `name=value` ペアのリストです。
- `version_tokens_delete()` は、現在のリストからトークンを削除します。引数は、セミコロンで区切られたトークン名のリストです。
- `version_tokens_show()` に現在のトークンリストが表示されます。引数は取りません。

これらの各関数は、成功した場合、発生したアクションを示すバイナリ文字列を返します。次の例では、サーバートークンリストを確立し、新しいトークンを追加して変更し、一部のトークンを削除して、結果のトークンリストを表示します:

```
mysql> SELECT version_tokens_set('tok1=a;tok2=b');
+-----+
| version_tokens_set('tok1=a;tok2=b') |
+-----+
| 2 version tokens set.                |
+-----+
mysql> SELECT version_tokens_edit('tok3=c');
+-----+
| version_tokens_edit('tok3=c') |
+-----+
| 1 version tokens updated.      |
+-----+
mysql> SELECT version_tokens_delete('tok2;tok1');
+-----+
| version_tokens_delete('tok2;tok1') |
+-----+
| 2 version tokens deleted.        |
+-----+
mysql> SELECT version_tokens_show();
+-----+
| version_tokens_show() |
+-----+
| tok3=c;                |
+-----+
```

トークンリストの形式が正しくない場合、警告が発生します:

```
mysql> SELECT version_tokens_set('tok1=a; =c');
+-----+
| version_tokens_set('tok1=a; =c') |
+-----+
| 1 version tokens set.            |
+-----+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
***** 1. row *****
Level: Warning
Code: 42000
Message: Invalid version token pair encountered. The list provided
is only partially updated.
1 row in set (0.00 sec)
```

前述のように、バージョントークンは、セミコロンで区切られた `name=value` ペアのリストを使用して定義されません。 `version_tokens_set()` の次の呼出しについて考えてみます:

```
mysql> SELECT version_tokens_set('tok1=b;;; tok2= a = b ; tok1 = 1\2 3"4')
+-----+
| version_tokens_set('tok1=b;;; tok2= a = b ; tok1 = 1\2 3"4') |
+-----+
| 3 version tokens set.                                          |
+-----+
```

バージョントークンは、引数を次のように解釈します:

- 名前と値の前後の空白は無視されます。名前と値の中に空白を含めることができます。(値のない名前のリストを取る `version_tokens_delete()` の場合、名前の前後の空白は無視されます。)
- 引用メカニズムはありません。
- トークンの順序は重要ではありませんが、トークンリストに特定のトークン名の複数のインスタンスが含まれている場合は、最後の値が以前の値より優先されます。

これらのルールを考慮すると、前述の `version_tokens_set()` コールの結果、2つのトークンを含むトークンリストが生成されます: `tok1` の値は `1'2 3'4` で、`tok2` の値は `a = b` です。これを確認するには、`version_tokens_show()` をコールします:

```
mysql> SELECT version_tokens_show();
+-----+
| version_tokens_show() |
+-----+
| tok2=a = b;tok1=1'2 3'4; |
+-----+
```

トークンリストに2つのトークンが含まれている場合、`version_tokens_set()` が値 `3 version tokens set` を返したのはなぜですか。これは、元のトークンリストに `tok1` の2つの定義が含まれ、2つ目の定義が最初の定義に置き換わったために発生しました。

バージョントークンのトークン操作 UDF は、トークン名と値に次の制約を設定します:

- トークン名に `=` または ; 文字を含めることはできず、最大長は 64 文字です。
- トークン値に ; 文字を含めることはできません。値の長さは、`max_allowed_packet` システム変数の値によって制約されます。
- バージョントークンはトークン名と値をバイナリ文字列として扱うため、比較では大文字と小文字が区別されません。

バージョントークンには、トークンをロックおよびロック解除できる UDF のセットも含まれています:

- `version_tokens_lock_exclusive()` は排他的バージョントークンロックを取得します。1つ以上のロック名とタイムアウト値のリストを取ります。
- `version_tokens_lock_shared()` は、共有バージョンのトークンロックを取得します。1つ以上のロック名とタイムアウト値のリストを取ります。
- `version_tokens_unlock()` は、バージョントークンロック (排他的および共有) を解放します。引数は取りません。

各ロック関数は、成功した場合はゼロ以外を返します。それ以外の場合は、次のエラーが発生します:

```
mysql> SELECT version_tokens_lock_shared('lock1', 'lock2', 0);
+-----+
| version_tokens_lock_shared('lock1', 'lock2', 0) |
+-----+
| 1 |
+-----+

mysql> SELECT version_tokens_lock_shared(NULL, 0);
ERROR 3131 (42000): Incorrect locking service lock name '(null)'.
```

バージョントークンロック機能を使用したロックは警告です。アプリケーションは協調する必要があります。

存在しないトークン名をロックできます。トークンは作成されません。

注記

バージョントークンロック関数は、[セクション5.6.8.1「ロックサービス」](#)で説明されているロックサービスに基づいているため、共有ロックと排他ロックのセマンティクスは同じです。(バージョントークンは、ロックサービス UDF インタフェースではなく、サーバーに組み込まれたロックサービスルーチンを使用するため、バージョントークンを使用するためにこれらの UDF をインストールする必要はありません。)バージョントークンによって取得されたロックは、`version_token_locks` のロックサービス名前空間を使用します。ロックサービスのロックはパフォーマンススキーマを使用してモニターできるため、これはバー

ジョイントトークンロックにも当てはまります。詳細は、[ロックサービスの監視](#)を参照してください。

バージョントークンロック関数の場合、トークン名引数は指定したとおりに正確に使用されます。前後の空白は無視されず、`=`および文字は許可されます。これは、バージョントークンが単にロック対象のトークン名をロックサービスにそのまま渡すためです。

5.6.6.4 バージョントークン参照

次の説明は、これらのバージョントークン要素への参照として機能します:

- [バージョントークン関数](#)
- [バージョントークンシステム変数](#)

バージョントークン関数

バージョントークンプラグインライブラリには、複数のユーザー定義関数が含まれています。UDF のセットでは、バージョントークンのサーバーリストを操作および検査できます。UDF の別のセットでは、バージョントークンをロックおよびロック解除できます。バージョントークン UDF を起動するには、`VERSION_TOKEN_ADMIN` 権限 (または非推奨の `SUPER` 権限) が必要です。

次の UDF を使用すると、バージョントークンのサーバーリストを作成、変更、削除および検査できます。`name_list` および `token_list` の引数 (空白の処理を含む) の解釈は、[セクション5.6.6.3「バージョントークンの使用」](#)で説明されているように行われます。ここでは、トークンを指定するための構文とその他の例について詳しく説明します。

- [version_tokens_delete\(name_list\)](#)

`name_list` 引数を使用してバージョントークンのサーバーリストからトークンを削除し、操作の結果を示すバイナリ文字列を返します。`name_list` は、削除するバージョントークン名のセミコロン区切りリストです。

```
mysql> SELECT version_tokens_delete('tok1;tok3');
+-----+
| version_tokens_delete('tok1;tok3') |
+-----+
| 2 version tokens deleted.          |
+-----+
```

`NULL` の引数は空の文字列として扱われ、トークンリストには影響しません。

`version_tokens_delete()` は、引数に指定されたトークンが存在する場合、それらを削除します。(存在しないトークンを削除してもエラーになりません。) リストに含まれるトークンを知らずにトークンリストを完全にクリアするには、`NULL` またはトークンを含まない文字列を `version_tokens_set()` に渡します:

```
mysql> SELECT version_tokens_set(NULL);
+-----+
| version_tokens_set(NULL) |
+-----+
| Version tokens list cleared. |
+-----+
mysql> SELECT version_tokens_set("");
+-----+
| version_tokens_set("") |
+-----+
| Version tokens list cleared. |
+-----+
```

- [version_tokens_edit\(token_list\)](#)

`token_list` 引数を使用してバージョントークンのサーバーリストを変更し、操作の結果を示すバイナリ文字列を返します。`token_list` は、定義する各トークンの名前とその値を指定する `name=value` ペアのセミコロン区切りリストです。トークンが存在する場合、その値は指定された値で更新されます。トークンが存在しない場合は、指定された値で作成されます。引数が `NULL` であるか、トークンを含まない文字列である場合、トークンリストは変更されません。

```
mysql> SELECT version_tokens_set('tok1=value1;tok2=value2');
+-----+
```

```
| version_tokens_set('tok1=value1;tok2=value2') |
+-----+
| 2 version tokens set. |
+-----+
mysql> SELECT version_tokens_edit('tok2=new_value2;tok3=new_value3');
+-----+
| version_tokens_edit('tok2=new_value2;tok3=new_value3') |
+-----+
| 2 version tokens updated. |
+-----+
```

- `version_tokens_set(token_list)`

バージョントークンのサーバーリストを `token_list` 引数で定義されたトークンに置き換え、操作の結果を示すバイナリ文字列を返します。`token_list` は、定義する各トークンの名前とその値を指定する `name=value` ペアのセミコロン区切りリストです。引数が `NULL` であるか、トークンを含まない文字列である場合、トークンリストはクリアされます。

```
mysql> SELECT version_tokens_set('tok1=value1;tok2=value2');
+-----+
| version_tokens_set('tok1=value1;tok2=value2') |
+-----+
| 2 version tokens set. |
+-----+
```

- `version_tokens_show()`

バージョントークンのサーバーリストを、`name=value` ペアのセミコロン区切りリストを含むバイナリ文字列として返します。

```
mysql> SELECT version_tokens_show();
+-----+
| version_tokens_show() |
+-----+
| tok2=value2;tok1=value1; |
+-----+
```

次の UDF を使用すると、バージョントークンをロックおよびロック解除できます:

- `version_tokens_lock_exclusive(token_name[, token_name] ..., timeout)`

指定されたタイムアウト値内にロックが取得されない場合、名前で文字列として指定された 1 つ以上のバージョントークンの排他ロックを取得し、エラーでタイムアウトします。

```
mysql> SELECT version_tokens_lock_exclusive('lock1', 'lock2', 10);
+-----+
| version_tokens_lock_exclusive('lock1', 'lock2', 10) |
+-----+
| 1 |
+-----+
```

- `version_tokens_lock_shared(token_name[, token_name] ..., timeout)`

指定されたタイムアウト値内にロックが取得されない場合、名前で文字列として指定された 1 つ以上のバージョントークンの共有ロックを取得し、エラーでタイムアウトします。

```
mysql> SELECT version_tokens_lock_shared('lock1', 'lock2', 10);
+-----+
| version_tokens_lock_shared('lock1', 'lock2', 10) |
+-----+
| 1 |
+-----+
```

- `version_tokens_unlock()`

`version_tokens_lock_exclusive()` および `version_tokens_lock_shared()` を使用して、現在のセッション内で取得されたすべてのロックを解放します。

```
mysql> SELECT version_tokens_unlock();
+-----+
```



```
| version_tokens_unlock() |
+-----+
|          1 |
+-----+
```

ロック関数は、次の特性を共有します:

- 成功の場合、戻り値はゼロ以外です。それ以外の場合は、エラーが発生します。
- トークン名は文字列です。
- サーバートークンリストを操作する UDF の引数処理とは対照的に、トークン名引数を囲む空白は無視されず、= および;文字は許可されます。
- 存在しないトークン名をロックできます。トークンは作成されません。
- タイムアウト値は、エラーでタイムアウトするまでにロックの取得を待機する時間 (秒) を表す負でない整数です。タイムアウトが 0 の場合、待機はなく、ロックをすぐに取得できないと、関数はエラーを生成します。
- バージョントークンのロック関数は、[セクション5.6.8.1「ロックサービス」](#) で説明されているロックサービスに基づいています。

バージョントークンシステム変数

バージョントークンでは、次のシステム変数がサポートされます。これらの変数は、バージョントークンプラグインがインストールされていないかぎり使用できません ([セクション5.6.6.2「バージョントークンのインストールまたはアンインストール」](#) を参照)。

システム変数:

- [version_tokens_session](#)

コマンド行形式	<code>--version-tokens-session=value</code>
システム変数	version_tokens_session
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	NULL

この変数のセッション値は、クライアントバージョントークンリストを指定し、クライアントセッションがサーババージョンのトークンリストを持つ必要があるトークンを示します。

[version_tokens_session](#) 変数が NULL (デフォルト) であるか、値が空の場合、サーババージョンのトークンリストが一致します。(実際には、空の値を指定すると、一致する要件が無効になります。)

[version_tokens_session](#) 変数に空でない値が含まれている場合、その値とサーババージョンのトークンリストが一致しないと、セッションがサーバに送信するステートメントでエラーが発生します。次の条件下で不一致が発生します:

- [version_tokens_session](#) 値のトークン名がサーバートークンリストに存在しません。この場合、[ER_VTOKEN_PLUGIN_TOKEN_NOT_FOUND](#) エラーが発生します。
- [version_tokens_session](#) 値のトークン値は、サーバートークンリストの対応するトークンの値とは異なります。この場合、[ER_VTOKEN_PLUGIN_TOKEN_MISMATCH](#) エラーが発生します。

サーババージョンのトークンリストに [version_tokens_session](#) 値に指定されていないトークンが含まれていても、不一致ではありません。

管理アプリケーションでサーバートークンリストが次のように設定されているとします:

```
mysql> SELECT version_tokens_set('tok1=a;tok2=b;tok3=c');
```

```

+-----+
| version_tokens_set('tok1=a;tok2=b;tok3=c') |
+-----+
| 3 version tokens set. |
+-----+

```

クライアントは、`version_tokens_session` 値を設定して、サーバーが一致する必要があるトークンを登録します。次に、クライアントによって送信される後続のステートメントごとに、サーバーはそのトークンリストをクライアントの `version_tokens_session` 値と照合してチェックし、不一致がある場合はエラーを生成します:

```

mysql> SET @@SESSION.version_tokens_session = 'tok1=a;tok2=b';
mysql> SELECT 1;
+---+
| 1 |
+---+
| 1 |
+---+

mysql> SET @@SESSION.version_tokens_session = 'tok1=b';
mysql> SELECT 1;
ERROR 3136 (42000): Version token mismatch for tok1. Correct value a

```

クライアントトークン `tok1` および `tok2` がサーバートークンリストに存在し、各トークンの値がサーバーリストに同じであるため、最初の `SELECT` は成功します。 `tok1` はサーバートークンリストに存在しますが、クライアントで指定された値とは異なるため、2 番目の `SELECT` は失敗します。

この時点で、サーバートークンリストが再度一致するように変更されないかぎり、クライアントによって送信されたステートメントは失敗します。管理アプリケーションがサーバートークンリストを次のように変更するとします:

```

mysql> SELECT version_tokens_edit('tok1=b');
+-----+
| version_tokens_edit('tok1=b') |
+-----+
| 1 version tokens updated. |
+-----+

mysql> SELECT version_tokens_show();
+-----+
| version_tokens_show() |
+-----+
| tok3=c;tok1=b;tok2=b; |
+-----+

```

これで、クライアントの `version_tokens_session` 値がサーバートークンリストと一致し、クライアントがステートメントを再度正常に実行できるようになります:

```

mysql> SELECT 1;
+---+
| 1 |
+---+
| 1 |
+---+

```

- `version_tokens_session_number`

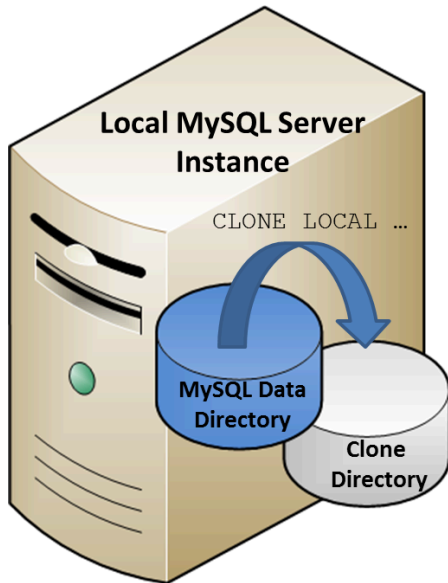
コマンド行形式	<code>--version-tokens-session-number=#</code>
システム変数	<code>version_tokens_session_number</code>
スコープ	グローバル、セッション
動的	いいえ
<code>SET_VAR</code> ヒントの適用	いいえ
型	Integer
デフォルト値	0

この変数は内部で使用されます。

5.6.7 クローンプラグイン

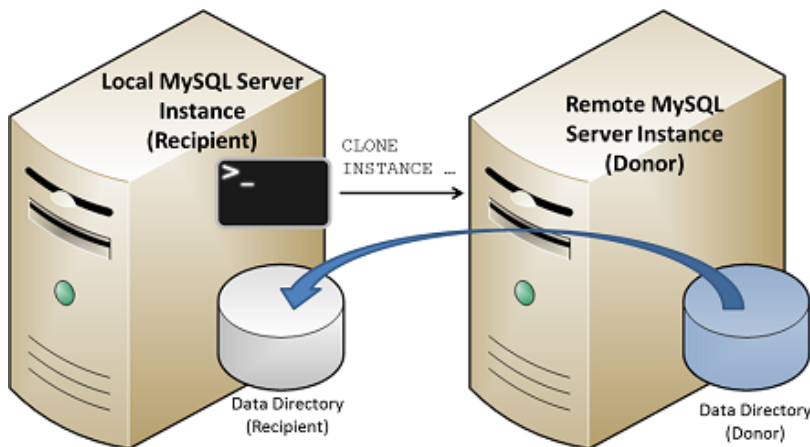
クローンプラグインを使用すると、ローカルまたはリモートの MySQL サーバーインスタンスからデータをクローニングできます。クローンデータは、スキーマ、テーブル、テーブルスペースおよびデータディクショナリメタデータを含む、InnoDB に格納されているデータの物理スナップショットです。クローンデータは完全に機能するデータディレクトリで構成され、MySQL サーバープロビジョニングにクローンプラグインを使用できます。

図 5.1 ローカルクローニング操作



ローカルクローニング操作では、クローニング操作が開始された MySQL サーバーインスタンスから、MySQL サーバーインスタンスが実行されているのと同じサーバーまたはノード上のディレクトリにデータをクローニングします。

図 5.2 リモートクローニング操作



リモートクローニング操作には、クローニング操作が開始されるローカル MySQL サーバーインスタンス (「受信者」) と、ソースデータが配置されるリモート MySQL サーバーインスタンス (「ドナー」) が含まれます。受信者でリモートクローニング操作が開始されると、クローニングされたデータがドナーから受信者にネットワーク経由で転送されます。デフォルトでは、リモートクローニング操作によって受信者データディレクトリ内のデータが削除され、クローニングされたデータに置き換えられます。必要に応じて、既存のデータを削除しないように、受信者の別のディレクトリにデータをクローニングできます。

リモートクローニング操作と比較して、ローカルクローニング操作によってクローニングされるデータに違いはありません。両方の操作で同じデータがクローニングされます。

クローンプラグインはレプリケーションをサポートします。クローニング操作では、クローニングデータに加えて、ドナーからレプリケーション座標が抽出および転送され、受信者に適用されるため、グループレプリケーションメンバーおよびレプリカのプロビジョニングにクローンプラグインを使用できます。プロビジョニングにクローンプラグインを使用すると、多数のトランザクションをレプリケートするよりもはるかに高速かつ効率的になります ([セクション5.6.7.6「レプリケーション用のクローニング」](#)を参照)。グループレプリケーションメンバーは、シードメンバーからグループデータを取得する最も効率的な方法をメンバーが自動的に選択できるように、代替のリカバリ方法としてクローンプラグインを使用するように構成することもできます。詳細は、[セクション18.4.3.2「分散リカバリのためのクローニング」](#)を参照してください。

クローンプラグインは、暗号化およびページ圧縮されたデータのクローニングをサポートします。 [セクション5.6.7.4「暗号化データのクローニング」](#) および [セクション5.6.7.5「圧縮データのクローニング」](#) を参照してください。

クローンプラグインは、使用する前にインストールする必要があります。インストールの手順については、 [セクション5.6.7.1「クローンプラグインのインストール」](#) を参照してください。クローニングの手順は、 [セクション5.6.7.2「ローカルでのデータのクローニング」](#) および [セクション5.6.7.3「リモートデータのクローニング」](#) を参照してください。

クローニング操作を監視するために、「パフォーマンススキーマ」テーブルおよびインストールメンテーションが用意されています。 [セクション5.6.7.9「クローニング操作の監視」](#) を参照してください。

5.6.7.1 クローンプラグインのインストール

このセクションでは、クローンプラグインをインストールおよび構成する方法について説明します。リモートクローニング操作の場合、クローンプラグインをドナーおよび受信者の MySQL サーバーインスタンスにインストールする必要があります。

プラグインのインストールまたはアンインストールに関する一般的な情報は、 [セクション5.6.1「プラグインのインストールおよびアンインストール」](#) を参照してください。

サーバーで使用できるようにするには、プラグインライブラリファイルを MySQL プラグインディレクトリ (`plugin_dir` システム変数で指定されたディレクトリ) に配置する必要があります。必要に応じて、サーバーの起動時に `plugin_dir` の値を設定して、プラグインディレクトリの場所をサーバーに通知します。

プラグインライブラリファイルのベース名は `mysql_clone.so` です。ファイル名の接尾辞は、プラットフォームによって異なります (たとえば、`.so` for Unix and Unix-like systems, `.dll` for Windows)。

サーバーの起動時にプラグインをロードするには、`--plugin-load-add` オプションを使用して、プラグインを含むライブラリファイルに名前を付けます。このプラグインのロード方式では、サーバーを起動するたびにオプションを指定する必要があります。たとえば、`my.cnf` ファイルに次の行を入力し、必要に応じてプラットフォームの `.so` 接尾辞を調整します:

```
[mysqld]
plugin-load-add=mysql_clone.so
```

`my.cnf` を変更したら、新しい設定を有効にするためにサーバーを再起動します。

注記

以前の MySQL バージョンからのアップグレード中にサーバーを再起動する場合、`--plugin-load-add` オプションを使用してクローンプラグインをロードすることはできません。たとえば、バイナリまたはパッケージを MySQL 5.7 から MySQL 8.0 にアップグレードした後、`plugin-load-add=mysql_clone.so` を使用してサーバーを再起動しようとする、このエラーが発生: `[ERROR][MY-013238] [サーバー]プラグイン'clone'のインストール中にエラーが発生しました: アップグレード中にインストールできません。回避策は、plugin-load-add=mysql_clone.so でサーバーを起動する前にサーバーをアップグレードすることです。`

または、実行時にプラグインをロードするには、次のステートメントを使用して、必要に応じてプラットフォームの `.so` 接尾辞を調整します:

```
INSTALL PLUGIN clone SONAME 'mysql_clone.so';
```

`INSTALL PLUGIN` はプラグインをロードし、それを `mysql.plugins` システムテーブルに登録して、`--plugin-load-add` を必要とせずに後続の通常のサーバー起動ごとにプラグインをロードします。

プラグインのインストールを確認するには、`INFORMATION_SCHEMA.PLUGINS` テーブルを調べるか、`SHOW PLUGINS` ステートメントを使用します (セクション5.6.2「サーバープラグイン情報の取得」を参照)。例:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME = 'clone';
+-----+-----+
| PLUGIN_NAME | PLUGIN_STATUS |
+-----+-----+
| clone       | ACTIVE        |
+-----+-----+
```

プラグインの初期化に失敗した場合は、サーバーエラーログでクローンまたはプラグイン関連の診断メッセージを確認します。

プラグインが以前に `INSTALL PLUGIN` に登録されているか、`--plugin-load-add` にロードされている場合は、サーバーの起動時に `--clone` オプションを使用して、プラグインのアクティブ化状態を制御できます。たとえば、起動時にプラグインをロードし、実行時にプラグインが削除されないようにするには、次のオプションを使用します:

```
[mysql]
plugin-load-add=mysql_clone.so
clone=FORCE_PLUS_PERMANENT
```

クローンプラグインなしでサーバーが実行されないようにするには、`FORCE` または `FORCE_PLUS_PERMANENT` の値を指定して `--clone` を使用し、プラグインが正常に初期化されない場合にサーバーの起動が強制的に失敗するようにします。

プラグインのアクティブ化状態の詳細は、[プラグインのアクティブ化状態の制御](#) を参照してください。

5.6.7.2 ローカルでのデータのクローニング

クローンプラグインは、データをローカルにクローニングするための次の構文をサポートしています。つまり、ローカル MySQL データディレクトリから、MySQL サーバーインスタンスが実行されている同じサーバーまたはノード上の別のディレクトリにデータをクローニングします:

```
CLONE LOCAL DATA DIRECTORY [=] 'clone_dir';
```

`CLONE` 構文を使用するには、クローンプラグインをインストールする必要があります。インストールの手順については、[セクション5.6.7.1「クローンプラグインのインストール」](#) を参照してください。

`CLONE LOCAL DATA DIRECTORY` ステートメントを実行するには、`BACKUP_ADMIN` 権限が必要です。

```
mysql> GRANT BACKUP_ADMIN ON *.* TO 'clone_user';
```

ここで、`clone_user` はクローニング操作を実行する MySQL ユーザーです。クローニング操作の実行を選択するユーザーは、`*.*` に対する `BACKUP_ADMIN` 権限を持つ任意の MySQL ユーザーです。

次の例は、ローカルでのデータのクローニングを示しています:

```
mysql> CLONE LOCAL DATA DIRECTORY = '/path/to/clone_dir';
```

ここで、`/path/to/clone_dir` は、データのクローニング先のローカルディレクトリのフルパスです。絶対パスが必要であり、指定したディレクトリ (「`clone_dir`」) が存在してはいけませんが、指定したパスは存在するパスである必要があります。MySQL サーバーには、ディレクトリの作成に必要な書込みアクセス権が必要です。

注記

ローカルクローニング操作では、データディレクトリの外部にあるユーザー作成のテーブルまたはテーブルスペースのクローニングはサポートされていません。このようなテーブルまたはテーブルスペースをクローニングしようとすると、次のエラーが発生: `ERROR 1086 (HY000): ファイル'/path/to/tablespace_name.ibd'はすでに存在`。ソーステーブルスペースと同じパスを持つテーブルスペースをクローニングすると競合が発生するため、禁止されます。

他のすべてのユーザー作成の `InnoDB` テーブルおよびテーブルスペース、`InnoDB` システムテーブルスペース、redo ログおよび undo テーブルスペースは、指定したディレクトリにクローニングされます。

必要に応じて、クローニング操作の完了後にクローンディレクトリで MySQL サーバーを起動できます。

```
shell> mysqld_safe --datadir=clone_dir
```

ここで、`clone_dir` はデータがクローニングされたディレクトリです。

クローニング操作のステータスおよび進行状況の監視の詳細は、[セクション5.6.7.9「クローニング操作の監視」](#)を参照してください。

5.6.7.3 リモートデータのクローニング

クローンプラグインでは、リモートデータをクローニングするための次の構文がサポートされています。つまり、リモート MySQL サーバーインスタンス (ドナー) からクローニング操作が開始された MySQL インスタンス (受信者) にデータをクローニングして転送します。

```
CLONE INSTANCE FROM 'user'@'host':port  
IDENTIFIED BY 'password'  
[DATA DIRECTORY [=] 'clone_dir']  
[REQUIRE [NO] SSL];
```

ここでは:

- `user` は、ドナー MySQL サーバーインスタンス上のクローンユーザーです。
- `password` は `user` のパスワードです。
- `host` は、ドナー MySQL サーバーインスタンスの `hostname` アドレスです。インターネットプロトコルバージョン 6 (IPv6) アドレス形式はサポートされていません。かわりに、IPv6 アドレスのエイリアスを使用できます。IPv4 アドレスはそのまま使用できます。
- `port` は、ドナー MySQL サーバーインスタンスの `port` 番号です。 (`mysqlx_port` で指定された X プロトコル ポートはサポートされていません。MySQL Router を介したドナー MySQL サーバーインスタンスへの接続もサポートされていません。)
- `DATA DIRECTORY [=] 'clone_dir'` は、クローニングするデータの受信者上のディレクトリを指定するために使用するオプションの句です。このオプションは、受信者データディレクトリ内の既存のデータを削除しない場合に使用します。絶対パスが必要であり、ディレクトリが存在していない必要があります。MySQL サーバーには、ディレクトリの作成に必要な書き込みアクセス権が必要です。

オプションの `DATA DIRECTORY [=] 'clone_dir'` 句を使用しない場合、クローニング操作では、受信者データディレクトリ内の既存のデータが削除され、クローニングされたデータに置き換えられ、その後サーバーが自動的に再起動されます。

- `[REQUIRE [NO] SSL]` は、クローニングされたデータをネットワーク経由で転送するときに、暗号化された接続を使用するかどうかを明示的に指定します。明示的な指定が満たされない場合は、エラーが返されます。SSL 句が指定されていない場合、クローンはデフォルトで暗号化された接続を確立しようとし、セキュアな接続試行が失敗した場合はセキュアでない接続にフォールバックします。暗号化データをクローニングする場合は、この句が指定されているかどうかに関係なく、セキュアな接続が必要です。詳細は、[クローニング用の暗号化された接続の構成](#)を参照してください。

注記

デフォルトでは、ドナー MySQL サーバーインスタンスのデータディレクトリに存在するユーザー作成の `InnoDB` テーブルおよびテーブルスペースは、受信者 MySQL サーバーインスタンスのデータディレクトリにクローニングされます。 `DATA DIRECTORY [=] 'clone_dir'` 句を指定すると、指定したディレクトリにクローニングされます。

ドナー MySQL サーバーインスタンスのデータディレクトリ外にあるユーザー作成の `InnoDB` テーブルおよびテーブルスペースは、受信者 MySQL サーバーインスタンスの同じパスにクローニングされます。 テーブルまたはテーブルスペースがすでに存在する場合は、エラーが報告されます。

デフォルトでは、`InnoDB` システムテーブルスペース、redo ログおよび undo テーブルスペースは、ドナーに構成されているのと同じ場所にクローニングされます (それぞ

れ、`innodb_data_home_dir` および `innodb_data_file_path`、`innodb_log_group_home_dir` および `innodb_undo_directory` で定義されています)。 `DATA DIRECTORY [=] 'clone_dir'` 句が指定されている場合、これらのテーブルスペースおよびログは指定されたディレクトリにクローニングされます。

リモートクローニングの前提条件

クローニング操作を実行するには、ドナーと受信者の両方の MySQL サーバーインスタンスでクローンプラグインがアクティブである必要があります。インストールの手順については、[セクション5.6.7.1「クローンプラグインのインストール」](#)を参照してください。

クローニング操作 (「クローンユーザー」) を実行するには、ドナーおよび受信者の MySQL ユーザーが必要です。

- ドナーでは、クローンユーザーには、ドナーからデータにアクセスして転送し、クローニング操作中に DDL をブロックするための `BACKUP_ADMIN` 権限が必要です。
- 受信者では、クローンユーザーに、受信者データの置換、クローニング操作中の DDL のブロック、およびサーバーの自動再起動のための `CLONE_ADMIN` 権限が必要です。 `CLONE_ADMIN` 権限には、`BACKUP_ADMIN` および `SHUTDOWN` 権限が暗黙的に含まれます。

クローンユーザーを作成し、必要な権限を付与する手順は、この前提条件情報に続くリモートクローニングの例に含まれています。

`CLONE INSTANCE` ステートメントの実行時には、次の前提条件がチェックされます:

- ドナーと受信者の MySQL サーバーバージョンは同じである必要があります。クローンプラグインは MySQL 8.0.17 以上でサポートされます。

```
mysql> SHOW VARIABLES LIKE 'version';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| version      | 8.0.17 |
+-----+-----+
```

- ドナーおよび受信者の MySQL サーバーインスタンスは、同じオペレーティングシステムおよびプラットフォーム上で実行する必要があります。たとえば、ドナーインスタンスが Linux 64-bit プラットフォームで実行されている場合、受信者インスタンスもそのプラットフォームで実行される必要があります。オペレーティングシステムのプラットフォームを決定する方法の詳細は、オペレーティングシステムのドキュメントを参照してください。
- 受信者には、クローンデータ用の十分なディスク領域が必要です。デフォルトでは、ドナーデータをクローニングする前に受信者データが削除されるため、ドナーデータに十分な領域のみが必要です。 `DATA DIRECTORY` 句を使用して名前付きディレクトリにクローニングする場合は、既存の受信者データおよびクローンデータ用の十分なディスク領域が必要です。データのサイズを見積もるには、ファイルシステムのデータディレクトリサイズと、データディレクトリの外部にあるテーブルスペースのサイズを確認します。ドナーでデータサイズを見積もる場合は、`InnoDB` データのみがクローニングされることに注意してください。データをほかのストレージエンジンに格納する場合は、それに応じてデータサイズの見積りを調整します。
- `InnoDB` では、データディレクトリ外に一部のテーブルスペースタイプを作成できます。ドナー MySQL サーバーインスタンスにデータディレクトリの外部に存在するテーブルスペースがある場合、クローニング操作はそれらのテーブルスペースにアクセスする必要があります。 `INFORMATION_SCHEMA.FILES` テーブルをクエリーして、データディレクトリの外部にあるテーブルスペースを識別できます。データディレクトリの外部にあるファイルには、データディレクトリ以外のディレクトリへの完全修飾パスがあります。

```
mysql> SELECT FILE_NAME FROM INFORMATION_SCHEMA.FILES;
```

- ドナー上でアクティブなプラグイン (キーリングプラグインを含む) も、受信者上でアクティブである必要があります。アクティブなプラグインを識別するには、`SHOW PLUGINS` ステートメントを発行するか、`INFORMATION_SCHEMA.PLUGINS` テーブルをクエリーします。
- ドナーと受信者は、同じ MySQL サーバー文字セットと照合順序を持つ必要があります。MySQL サーバーの文字セットおよび照合順序の構成の詳細は、[セクション10.15「文字セットの構成」](#)を参照してください。

- ドナーと受信者には、同じ `innodb_page_size` および `innodb_data_file_path` 設定が必要です。ドナーと受信者の `innodb_data_file_path` 設定では、同数のデータファイルを同等のサイズで指定する必要があります。変数の設定は、`SHOW VARIABLES` 構文を使用して確認できます。

```
mysql> SHOW VARIABLES LIKE 'innodb_page_size';
mysql> SHOW VARIABLES LIKE 'innodb_data_file_path';
```

- 暗号化されたデータまたはページ圧縮されたデータをクローニングする場合、ドナーと受信者のファイルシステムのブロックサイズは同じである必要があります。ページ圧縮データの場合、受信者でホールパンチが発生するには、受信者ファイルシステムがスパースファイルとホールパンチをサポートしている必要があります。これらの機能と、それらを使用するテーブルおよびテーブルスペースの識別方法の詳細は、[セクション5.6.7.4「暗号化データのクローニング」](#) および [セクション5.6.7.5「圧縮データのクローニング」](#) を参照してください。ファイルシステムのブロックサイズを確認するには、オペレーティングシステムのドキュメントを参照してください。
- 暗号化されたデータをクローニングする場合は、セキュアな接続が必要です。[クローニング用の暗号化された接続の構成](#)を参照してください。
- 受信者の `clone_valid_donor_list` 設定には、ドナー MySQL サーバーインスタンスのホストアドレスを含める必要があります。有効なドナーリストのホストからのみデータをクローニングできます。この変数を構成するには、`SYSTEM_VARIABLES_ADMIN` 権限を持つ MySQL ユーザーが必要です。`clone_valid_donor_list` 変数の設定手順は、このセクションの後のリモートクローニングの例で説明します。`SHOW VARIABLES` 構文を使用して、`clone_valid_donor_list` 設定を確認できます。

```
mysql> SHOW VARIABLES LIKE 'clone_valid_donor_list';
```

- 他のクローニング操作は実行しないでください。一度に許可されるのは単一のクローニング操作のみです。クローン操作が実行されているかどうかを確認するには、`clone_status` テーブルをクエリーします。[パフォーマンススキーマクローンテーブルを使用したクローニング操作のモニタリング](#)を参照してください。
- クローンプラグインは、1M バイトのパケットとメタデータでデータを転送します。したがって、ドナーおよび受信者の MySQL サーバーインスタンスに必要な `max_allowed_packet` の最小値は 2MB です。2MB 未満の `max_allowed_packet` 値はエラーになります。次のクエリーを使用して、`max_allowed_packet` 設定を確認します:

```
mysql> SHOW VARIABLES LIKE 'max_allowed_packet';
```

次の前提条件も適用されます:

- ドナーの undo テーブルスペースファイル名は一意である必要があります。データが受信者にクローニングされると、undo テーブルスペースは、ドナー上の場所に関係なく、受信者の `innodb_undo_directory` の場所または `DATA DIRECTORY [=] 'clone_dir'` 句で指定されたディレクトリ (使用されている場合) にクローニングされます。このため、ドナーでの undo テーブルスペースファイル名の重複は許可されません。MySQL 8.0.18 では、クローニング操作中に重複する undo テーブルスペースファイル名が検出されると、エラーがレポートされます。MySQL 8.0.18 より前は、同じファイル名の undo テーブルスペースをクローニングすると、受信者で undo テーブルスペースファイルが上書きされる可能性があります。

ドナーで undo テーブルスペースのファイル名を表示して一意であることを確認するには、`INFORMATION_SCHEMA.FILES` にクエリーします:

```
mysql> SELECT TABLESPACE_NAME, FILE_NAME FROM INFORMATION_SCHEMA.FILES
WHERE FILE_TYPE LIKE 'UNDO LOG';
```

undo テーブルスペースファイルの削除および追加の詳細は、[セクション15.6.3.4「undo テーブルスペース」](#) を参照してください。

- デフォルトでは、受信者 MySQL サーバーインスタンスは、データのクローニング後に自動的に再起動 (停止および起動) されます。自動再起動を実行するには、受信者で監視プロセスを使用してサーバーの停止を検出する必要があります。それ以外の場合、クローニング操作は、データのクローニング後に次のエラーで停止し、受信者の MySQL サーバーインスタンスが停止します:

```
ERROR 3707 (HY000): Restart server failed (mysqld is not managed by supervisor process).
```

このエラーはクローニングの失敗を示しません。つまり、受信者の MySQL サーバーインスタンスは、データのクローニング後に手動で再起動する必要があります。サーバーを手動で起動した後、受信者の MySQL サーバーインスタンスに接続し、パフォーマンススキーマクローンテーブルをチェックして、クローニング操作が正常

に完了したことを確認できます (パフォーマンススキーマクローンテーブルを使用したクローニング操作のモニタリングを参照。) `RESTART` ステートメントのモニタリングプロセス要件は同じです。詳細は、[セクション 13.7.8.8 「RESTART ステートメント」](#) を参照してください。この場合、自動再起動は実行されないため、`DATA DIRECTORY` 句を使用して名前付きディレクトリにクローニングする場合、この要件は適用されません。

- リモートクローニング操作の様々な側面を制御する変数がいくつかあります。リモートクローニング操作を実行する前に、変数を確認し、使用しているコンピューティング環境に合わせて必要に応じて設定を調整します。クローン変数は、クローニング操作が実行される受信者 MySQL サーバーインスタンスに設定されます。[セクション 5.6.7.12 「クローンシステム変数」](#) を参照してください。

リモートデータのクローニング

次の例は、リモートデータのクローニングを示しています。デフォルトでは、リモートクローニング操作により、受信者データディレクトリ内のデータが削除され、クローニングされたデータに置き換えられ、後で MySQL サーバーが再起動されます。

この例では、リモートクローニングの前提条件を満たしていることを前提としています。[リモートクローニングの前提条件](#)を参照してください。

1. 管理ユーザーアカウントでドナー MySQL サーバーインスタンスにログインします。

- a. `BACKUP_ADMIN` 権限を持つクローンユーザーを作成します。

```
mysql> CREATE USER 'donor_clone_user'@'example.donor.host.com' IDENTIFIED BY 'password';
mysql> GRANT BACKUP_ADMIN on *.* to 'donor_clone_user'@'example.donor.host.com';
```

- b. クローンプラグインをインストールします:

```
mysql> INSTALL PLUGIN clone SONAME 'mysql_clone.so';
```

2. 管理ユーザーアカウントで受信者 MySQL サーバーインスタンスにログインします。

- a. `CLONE_ADMIN` 権限を持つクローンユーザーを作成します。

```
mysql> CREATE USER 'recipient_clone_user'@'example.recipient.host.com' IDENTIFIED BY 'password';
mysql> GRANT CLONE_ADMIN on *.* to 'recipient_clone_user'@'example.recipient.host.com';
```

- b. クローンプラグインをインストールします:

```
mysql> INSTALL PLUGIN clone SONAME 'mysql_clone.so';
```

- c. ドナー MySQL サーバーインスタンスのホストアドレスを `clone_valid_donor_list` 変数設定に追加します。

```
mysql> SET GLOBAL clone_valid_donor_list = 'example.donor.host.com:3306';
```

3. 以前に作成したクローンユーザー (`recipient_clone_user'@'example.recipient.host.com`) として受信者 MySQL サーバーインスタンスにログオンし、`CLONE INSTANCE` ステートメントを実行します。

```
mysql> CLONE INSTANCE FROM 'donor_clone_user'@'example.donor.host.com':3306
IDENTIFIED BY 'password';
```

データがクローニングされると、受信者の MySQL サーバーインスタンスが自動的に再起動されます。

クローニング操作のステータスおよび進行状況の監視の詳細は、[セクション 5.6.7.9 「クローニング操作の監視」](#) を参照してください。

名前付きディレクトリへのクローニング

デフォルトでは、リモートクローニング操作によって受信者データディレクトリ内のデータが削除され、クローニングされたデータに置き換えられます。名前付きディレクトリにクローニングすることで、受信者データディレクトリから既存のデータを削除しないようにできます。

名前付きディレクトリにクローニングする手順は、[リモートデータのクローニング](#) で説明されている手順と同じですが、例外があります: `CLONE INSTANCE` ステートメントには、`DATA DIRECTORY` 句を含める必要があります。例:

```
mysql> CLONE INSTANCE FROM 'user'@'example.donor.host.com':3306
```

```
IDENTIFIED BY 'password'  
DATA DIRECTORY = '/path/to/clone_dir';
```

絶対パスが必要であり、ディレクトリが存在していない必要があります。MySQL サーバーには、ディレクトリの作成に必要な書込みアクセス権が必要です。

指定されたディレクトリにクローニングする場合、受信者の MySQL サーバーインスタンスは、データのクローニング後に自動的に再起動されません。指定したディレクトリで MySQL サーバーを再起動する場合は、手動で再起動する必要があります。

```
shell> mysqld_safe --datadir=/path/to/clone_dir
```

ここで、`/path/to/clone_dir` は受信者の指定されたディレクトリへのパスです。

クローニング用の暗号化された接続の構成

リモートクローニング操作の暗号化された接続を構成して、ネットワーク経由でクローニングされるデータを保護できます。暗号化されたデータをクローニングする場合は、デフォルトで暗号化された接続が必要です ([セクション 5.6.7.4 「暗号化データのクローニング」](#) を参照。)

次の手順では、暗号化された接続を使用するように受信者 MySQL サーバーインスタンスを構成する方法について説明します。ドナー MySQL サーバーインスタンスは、暗号化された接続を使用するようにすでに構成されていることを前提としています。そうでない場合は、サーバー側の構成手順について [セクション 6.3.1 「暗号化接続を使用するための MySQL の構成」](#) を参照してください。

暗号化された接続を使用するように受信者 MySQL サーバーインスタンスを構成するには:

1. ドナー MySQL サーバーインスタンスのクライアント証明書およびキーファイルを受信者ホストで使用できるようにします。セキュアなチャネルを使用して受信者ホストにファイルを配布するか、受信者ホストにアクセス可能なマウントされたパーティションにファイルを配置します。使用可能にするクライアント証明書およびキーファイルには、次のものがあります:

- `ca.pem`

自己署名認証局 (CA) ファイル。

- `client-cert.pem`

クライアント公開キー証明書ファイル。

- `client-key.pem`

クライアント秘密キーファイル。

2. 受信者 MySQL サーバーインスタンスで次の SSL オプションを構成します。

- `clone_ssl_ca`

自己署名認証局 (CA) ファイルへのパスを指定します。

- `clone_ssl_cert`

クライアント公開キー証明書ファイルへのパスを指定します。

- `clone_ssl_key`

クライアント秘密キーファイルへのパスを指定します。

例:

```
clone_ssl_ca=/path/to/ca.pem  
clone_ssl_cert=/path/to/client-cert.pem  
clone_ssl_key=/path/to/client-key.pem
```

3. 暗号化された接続を使用する必要がある場合は、受信者に対して `CLONE` ステートメントを発行するときに `REQUIRE SSL` 句を含めます。

```
mysql> CLONE INSTANCE FROM 'user'@'example.donor.host.com':3306
IDENTIFIED BY 'password'
DATA DIRECTORY = '/path/to/clone_dir'
REQUIRE SSL;
```

SSL 句が指定されていない場合、クローンプラグインはデフォルトで暗号化された接続を確立しようとし、暗号化された接続試行が失敗した場合は暗号化されていない接続にフォールバックします。

注記

暗号化データをクローニングする場合は、`REQUIRE SSL` 句が指定されているかどうかに関係なく、デフォルトで暗号化された接続が必要です。暗号化されたデータをクローニングしようとする、`REQUIRE NO SSL` を使用するとエラーが発生します。

5.6.7.4 暗号化データのクローニング

暗号化データのクローニングがサポートされています。次の要件が適用されます:

- リモートデータをクローニングして、暗号化されていないテーブルスペースをネットワーク経由で安全に転送するには、セキュアな接続が必要です。テーブルスペースは、トランスポート前にドナーで復号化され、受信者マスターキーを使用して受信者で再暗号化されます。暗号化された接続が使用できない場合、または `CLONE INSTANCE` ステートメントで `REQUIRE NO SSL` 句が使用されている場合は、エラーが報告されます。クローニング用の暗号化された接続の構成の詳細は、[クローニング用の暗号化された接続の構成](#) を参照してください。
- ローカル管理キーリングを使用するローカルデータディレクトリにデータをクローニングする場合は、クローンディレクトリで MySQL サーバーを起動するときに同じキーリングを使用する必要があります。
- ローカル管理キーリングを使用するリモートデータディレクトリ (受信者ディレクトリ) にデータをクローニングする場合、クローニングされたディレクトリで MySQL サーバーを起動するときに受信者キーリングを使用する必要があります。

注記

クローニング操作の進行中は、`innodb_redo_log_encrypt` および `innodb_undo_log_encrypt` の変数設定を変更できません。

データ暗号化機能の詳細は、[セクション 15.13 「InnoDB 保存データ暗号化」](#) を参照してください。

5.6.7.5 圧縮データのクローニング

ページ圧縮データのクローニングがサポートされています。リモートデータをクローニングする場合は、次の要件が適用されます:

- 受信者でホールパンチが発生するには、受信者ファイルシステムでスパースファイルとホールパンチがサポートされている必要があります。
- ドナーと受信者のファイルシステムのブロックサイズは同じである必要があります。ファイルシステムのブロックサイズが異なる場合は、次のようなエラーがレポートされます: `ERROR 3868 (HY000): クローン構成 FS ブロックサイズ: ドナー値: 114688 は受信者の値と異なります: 4096`。

ページ圧縮機能の詳細は、[セクション 15.9.2 「InnoDB ページ圧縮」](#) を参照してください。

5.6.7.6 レプリケーション用のクローニング

クローンプラグインはレプリケーションをサポートします。クローニング操作では、クローニングデータに加えて、ドナーからレプリケーション座標が抽出されて受信者に転送されるため、グループレプリケーションメンバーおよびレプリカのプロビジョニングにクローンプラグインを使用できます。プロビジョニングにクローンプラグインを使用すると、多数のトランザクションをレプリケートするよりもはるかに高速かつ効率的になります。

グループレプリケーションメンバーは、分散リカバリのオプションとしてクローンプラグインを使用するように構成することもできます。この場合、メンバーを結合すると、既存のグループメンバーからグループデータを取得する最も効率的な方法が自動的に選択されます。詳細は、[セクション 18.4.3.2 「分散リカバリのためのクローニング」](#) を参照してください。

クローニング操作中に、バイナリログの位置 (ファイル名、オフセット) と `gtid_executed` GTID セットの両方が抽出され、ドナー MySQL サーバーインスタンスから受信者に転送されます。このデータを使用すると、レプリケーションストリーム内の一貫した位置でレプリケーションを開始できます。ファイルに保持されているバイナリログおよびリレーログは、ドナーから受信者にコピーされません。レプリケーションを開始するには、受信者がドナーをキャッチアップするために必要なバイナリログを、データがクローニングされてからレプリケーションが開始されるまでパージしないでください。必要なバイナリログが使用できない場合は、レプリケーションハンドシェイクエラーが報告されます。したがって、クローニングされたインスタンスは、必要なバイナリログがパージされたり、新しいメンバーが大幅に遅れたりしないように、過剰な遅延なしでレプリケーショングループに追加する必要があり、リカバリ時間が長くなります。

- クローニングされた MySQL サーバーインスタンスで次のクエリを発行して、受信者に転送されたバイナリログの位置を確認します:

```
mysql> SELECT BINLOG_FILE, BINLOG_POSITION FROM performance_schema.clone_status;
```

- クローニングされた MySQL サーバーインスタンスで次のクエリを発行して、受信者に転送された `gtid_executed` GTID セットを確認します:

```
mysql> SELECT @@GLOBAL.GTID_EXECUTED;
```

MySQL 8.0 のデフォルトでは、レプリケーションメタデータリポジトリは、クローニング操作中にドナーから受信者にコピーされるテーブルに保持されます。レプリケーションメタデータリポジトリには、クローニング操作後にレプリケーションを正しく再開するために使用できるレプリケーション関連の構成設定が保持されます。

- MySQL 8.0.17 および 8.0.18 では、`mysql.slave_master_info` テーブル (接続メタデータリポジトリ) のみがコピーされます。
- MySQL 8.0.19 から、`mysql.slave_relay_log_info` (アプライヤメタデータリポジトリ) テーブルおよび `mysql.slave_worker_info` (アプライワーメタデータリポジトリ) テーブルもコピーされます。

各テーブルに含まれる内容のリストは、[セクション17.2.4.2「レプリケーションメタデータリポジトリ」](#) を参照してください。設定 `master_info_repository=FILE` および `relay_log_info_repository=FILE` がサーバーで使用されている場合 (MySQL 8.0 ではデフォルトではなく、非推奨です)、レプリケーションメタデータリポジトリはクローニングされず、`TABLE` が設定されている場合にのみクローニングされます。

レプリケーション用にクローニングするには、次のステップを実行します:

- グループレプリケーション用の新しいグループメンバーの場合は、[セクション18.2.1.6「グループへのインスタンスの追加」](#) の手順に従って、まずグループレプリケーション用の MySQL Server インスタンスを構成します。また、[セクション18.4.3.2「分散リカバリのためのクローニング」](#) で説明されているクローニングの前提条件も設定します。結合メンバーに対して `START GROUP_REPLICATION` を発行すると、クローニング操作はグループレプリケーションによって自動的に管理されるため、操作を手動で実行する必要はなく、結合メンバーに対してこれ以上の設定ステップを実行する必要もありません。
- ソース/レプリカ MySQL レプリケーショントポロジ内のレプリカの場合、まずドナー MySQL サーバーインスタンスから受信者に手動でデータをクローニングします。ドナーは、レプリケーショントポロジのソースまたはレプリカである必要があります。クローニングの手順は、[セクション5.6.7.3「リモートデータのクローニング」](#) を参照してください。
- クローニング操作が正常に完了した後、ドナーに存在する受信者 MySQL サーバーインスタンスで同じレプリケーションチャンネルを使用する場合は、ソース/レプリカ MySQL レプリケーショントポロジでレプリケーションを自動的に再開できるかどうか、および手動で設定する必要があるかどうかを確認します。
 - GTID ベースのレプリケーションでは、受信者が `gtid_mode=ON` を使用して構成され、`gtid_mode=ON`、`ON_PERMISSIVE` または `OFF_PERMISSIVE` を使用してドナーからクローニングされた場合、ドナーからの `gtid_executed` GTID セットが受信者に適用されます。トポロジ内にすでに存在するレプリカから受信者がクローニングされている場合、GTID 自動配置を使用する受信者のレプリケーションチャンネルは、チャンネルの起動時にクローニング操作後にレプリケーションを自動的に再開できます。これらの同じチャンネルを使用するだけの場合は、手動設定を実行する必要はありません。
 - バイナリログファイルの位置ベースのレプリケーションでは、受信者が MySQL 8.0.17 または 8.0.18 にいる場合、ドナーからのバイナリログの位置は受信者に適用されず、パフォーマンススキーマ `clone_status` テーブルにのみ記録されます。したがって、バイナリログファイルの位置ベースのレプリケーションを使用する受信者

のレプリケーションチャンネルは、クローニング操作後にレプリケーションを再開するように手動で設定する必要があります。サーバーの起動時にレプリケーションを自動的に開始するようにこれらのチャンネルが構成されていないことを確認します。これらのチャンネルにはまだバイナリログの位置がなく、最初からレプリケーションを開始しようとしていないためです。

- バイナリログファイルの位置ベースのレプリケーションでは、受信者が MySQL 8.0.19 以上の場合、ドナーからのバイナリログの位置が受信者に適用されます。バイナリログファイルの位置ベースのレプリケーションを使用する受信者のレプリケーションチャンネルは、複製されたリレーログ情報を使用してリレーログリカバリプロセスの実行を自動的に試みてから、レプリケーションを再開します。シングルスレッドレプリカ (`slave_parallel_workers` が 0 に設定されている) の場合、他の問題がなければリレーログリカバリは成功し、チャンネルはそれ以上の設定なしでレプリケーションを再開できます。マルチスレッドレプリカ (`slave_parallel_workers` が 0 より大きい) の場合、通常は自動的に完了できないため、リレーログリカバリが失敗する可能性があります。この場合、エラーメッセージが発行され、チャンネルを手動で設定する必要があります。
4. クローニングされたレプリケーションチャンネルを手動で設定する必要がある場合、または受信者で異なるレプリケーションチャンネルを使用する必要がある場合は、次の手順で、受信者 MySQL サーバーインスタンスをレプリケーショントポロジに追加するためのサマリーおよび省略された例を示します。レプリケーション設定に適用される詳細な手順も参照してください。
- GTID ベースのトランザクションをレプリケーションデータソースとして使用する MySQL レプリケーショントポロジに受信者 MySQL サーバーインスタンスを追加するには、[セクション 17.1.3.4 「GTID を使用したレプリケーションのセットアップ」](#) の手順に従って、必要に応じてインスタンスを構成します。次の省略例に示すように、インスタンスのレプリケーションチャンネルを追加します。(MySQL 8.0.23 の) `CHANGE REPLICATION SOURCE TO` ステートメントまたは (MySQL 8.0.23 の前の) `CHANGE MASTER TO` ステートメントで、ソースのホストアドレスおよびポート番号を定義し、`SOURCE_AUTO_POSITION | MASTER_AUTO_POSITION` オプションを有効にする必要があります:

```
mysql> CHANGE MASTER TO MASTER_HOST = 'source_host_name', MASTER_PORT = source_port_num,
...
MASTER_AUTO_POSITION = 1,
FOR CHANNEL 'setup_channel';
mysql> START SLAVE USER = 'user_name' PASSWORD = 'password' FOR CHANNEL 'setup_channel';

Or from MySQL 8.0.22 and 8.0.23:

mysql> CHANGE SOURCE TO SOURCE_HOST = 'source_host_name', SOURCE_PORT = source_port_num,
...
SOURCE_AUTO_POSITION = 1,
FOR CHANNEL 'setup_channel';
mysql> START REPLICA USER = 'user_name' PASSWORD = 'password' FOR CHANNEL 'setup_channel';
```

- バイナリログファイルの位置ベースのレプリケーションを使用する MySQL レプリケーショントポロジに受信者 MySQL サーバーインスタンスを追加するには、[セクション 17.1.2 「バイナリログファイルの位置ベースのレプリケーションの設定」](#) の手順に従って、必要に応じてインスタンスを構成します。クローニング操作中に受信者に転送されたバイナリログ位置を使用して、次の省略例に示すように、インスタンスのレプリケーションチャンネルを追加します:

```
mysql> SELECT BINLOG_FILE, BINLOG_POSITION FROM performance_schema.clone_status;
mysql> CHANGE MASTER TO MASTER_HOST = 'source_host_name', MASTER_PORT = source_port_num,
...
MASTER_LOG_FILE = 'source_log_name',
MASTER_LOG_POS = source_log_pos,
FOR CHANNEL 'setup_channel';
mysql> START SLAVE USER = 'user_name' PASSWORD = 'password' FOR CHANNEL 'setup_channel';

Or from MySQL 8.0.22 and 8.0.23:

mysql> SELECT BINLOG_FILE, BINLOG_POSITION FROM performance_schema.clone_status;
mysql> CHANGE SOURCE TO SOURCE_HOST = 'source_host_name', SOURCE_PORT = source_port_num,
...
SOURCE_LOG_FILE = 'source_log_name',
SOURCE_LOG_POS = source_log_pos,
FOR CHANNEL 'setup_channel';
mysql> START REPLICA USER = 'user_name' PASSWORD = 'password' FOR CHANNEL 'setup_channel';
```

5.6.7.7 クローニング操作中に作成されるディレクトリおよびファイル

データがクローニングされると、内部使用のために次のディレクトリおよびファイルが作成されます。これらは変更しないでください。

- `#clone`: クローニング操作で使用される内部クローンファイルが含まれます。データのクローニング先のディレクトリに作成されます。
- `#ib_archive`: クローニング操作中にドナーにアーカイブされた、内部的にアーカイブされたログファイルが含まれます。
- `*.#clone` ファイル: 既存のデータディレクトリがリモートクローニング操作に置き換えられている間に受信者に作成された一時データファイル。

5.6.7.8 リモートクローニング操作の失敗処理

このセクションでは、クローニング操作の様々な段階での障害処理について説明します。

1. 前提条件がチェックされます ([リモートクローニングの前提条件](#) を参照)。
 - 前提条件チェック中に障害が発生した場合、`CLONE INSTANCE` 操作によってエラーが報告されます。
2. DDL 操作をブロックするためにバックアップロックが取得されます。
 - クローニング操作で、`clone_ddl_timeout` 変数で指定された時間制限内に DDL ロックを取得できない場合は、エラーが報告されます。
3. 受信者のユーザー作成データ (スキーマ、テーブル、テーブルスペース) およびバイナリログは、データが受信者データディレクトリにクローニングされる前に削除されます。
 - リモートクローニング操作中にユーザーが作成したデータが受信者から削除されると、受信者データディレクトリ内の既存のデータは保存されず、障害が発生した場合に失われる可能性があります。受信者で置き換えるデータが重要な場合は、リモートクローニング操作を開始する前にバックアップを作成する必要があります。

情報提供のために、データの削除がいつ開始および終了するかを指定する警告がサーバーエラーログに出力されます:

```
[Warning] [MY-013453] [InnoDB] Clone removing all user data for provisioning:
Started...

[Warning] [MY-013453] [InnoDB] Clone removing all user data for provisioning:
Finished
```

データの削除中に障害が発生した場合は、クローニング操作の前に存在していたスキーマ、テーブルおよびテーブルスペースの一部が受信者に残される可能性があります。クローニング操作の実行中または障害の発生後は、サーバーは常に一貫性のある状態になります。

4. データはドナーからクローニングされます。ユーザー作成データ、ディクショナリメタデータおよびその他のシステムデータがクローニングされます。
 - データのクローニング中に障害が発生した場合、クローニング操作はロールバックされ、クローニングされたすべてのデータが削除されます。この段階では、受信者の既存のデータも削除され、受信者にはユーザーデータが残されません。
 - このシナリオが発生した場合は、失敗の原因を修正してクローニング操作を再実行するか、クローニング操作を忘れてクローニング操作の前に作成したバックアップから受信者データをリストアできます。
5. サーバーは自動的に再起動されます (名前付きディレクトリにクローニングしないリモートクローニング操作に適用されます)。起動時には、一般的なサーバー起動タスクが実行されます。
 - サーバーの自動再起動が失敗した場合は、サーバーを手動で再起動してクローニング操作を完了できます。

クローニング操作中にネットワークエラーが発生した場合、エラーが 5 分以内に解決されると操作は再開されます。それ以外の場合、操作は中止され、エラーが返されます。

5.6.7.9 クローニング操作の監視

このセクションでは、クローニング操作を監視するためのオプションについて説明します。

- パフォーマンススキーマクローンテーブルを使用したクローニング操作のモニタリング
- パフォーマンススキーマステージイベントを使用したクローニング操作の監視
- パフォーマンススキーマクローンの計測を使用したクローニング操作のモニタリング
- Com_clone ステータス変数

パフォーマンススキーマクローンテーブルを使用したクローニング操作のモニタリング

データの量やデータ転送に関連するその他の要因によっては、クローニング操作の完了に時間がかかる場合があります。clone_status および clone_progress 「パフォーマンススキーマ」テーブルを使用して、受信者 MySQL サーバーインスタンスでのクローニング操作のステータスおよび進行状況を監視できます。

注記

clone_status および clone_progress 「パフォーマンススキーマ」テーブルは、受信者 MySQL サーバーインスタンスでのみクローニング操作を監視するために使用できます。ドナー MySQL サーバーインスタンスでクローニング操作を監視するには、パフォーマンススキーマステージイベントを使用したクローニング操作の監視の説明に従ってクローンステージイベントを使用します。

- clone_status テーブルには、現在または最後に実行されたクローニング操作の状態が表示されます。クローン操作には、4 つの状態があります: Not Started, In Progress, Completed および Failed。
- clone_progress テーブルには、現在または最後に実行されたクローン操作の進行状況情報がステージ別に表示されます。クローニング操作のステージには、DROP DATA, FILE COPY, PAGE_COPY, REDO_COPY, FILE_SYNC, RESTART および RECOVERY が含まれます。

パフォーマンススキーマクローンテーブルにアクセスするには、パフォーマンススキーマに対する SELECT および EXECUTE 権限が必要です。

クローニング操作の状態を確認するには:

- 受信者の MySQL サーバーインスタンスに接続します。
- clone_status テーブルをクエリーします:

```
mysql> SELECT STATE FROM performance_schema.clone_status;
+-----+
| STATE |
+-----+
| Completed |
+-----+
```

クローニング操作中に障害が発生した場合は、clone_status テーブルにエラー情報をクエリーすることができます:

```
mysql> SELECT STATE, ERROR_NO, ERROR_MESSAGE FROM performance_schema.clone_status;
+-----+-----+-----+
| STATE | ERROR_NO | ERROR_MESSAGE |
+-----+-----+-----+
| Failed | xxx | "xxxxxxxxxxxx" |
+-----+-----+-----+
```

クローニング操作の各ステージの詳細を確認するには:

- 受信者の MySQL サーバーインスタンスに接続します。
- clone_progress テーブルをクエリーします。たとえば、次のクエリーでは、クローニング操作の各ステージの状態および終了時間データが提供されます:

```
mysql> SELECT STAGE, STATE, END_TIME FROM performance_schema.clone_progress;
+-----+-----+-----+
| stage | state | end_time |
+-----+-----+-----+
| DROP DATA | Completed | 2019-01-27 22:45:43.141261 |
| FILE COPY | Completed | 2019-01-27 22:45:44.457572 |
| PAGE COPY | Completed | 2019-01-27 22:45:44.577330 |
| REDO COPY | Completed | 2019-01-27 22:45:44.679570 |
```

```

| FILE SYNC | Completed | 2019-01-27 22:45:44.918547 |
| RESTART   | Completed | 2019-01-27 22:45:48.583565 |
| RECOVERY  | Completed | 2019-01-27 22:45:49.626595 |
+-----+-----+-----+

```

監視可能なその他のクローンステータスおよび進捗データポイントについては、[セクション27.12.17「パフォーマンススキーマクローンテーブル」](#)を参照してください。

パフォーマンススキーマステージイベントを使用したクローニング操作の監視

データの量やデータ転送に関連するその他の要因によっては、クローニング操作の完了に時間がかかる場合があります。クローニング操作の進行状況を監視するには、3つのステージイベントがあります。各ステージイベントでは、[WORK_COMPLETED](#) および [WORK_ESTIMATED](#) の値がレポートされます。レポートされた値は、操作の進行に応じて改訂されます。

クローニング操作を監視する方法は、ドナーまたは受信者の MySQL サーバーインスタンスで使用できます。

発生順に、クローニング操作ステージイベントは次のとおりです：

- [stage/innodb/clone \(file copy\)](#): クローニング操作のファイルコピーフェーズの進行状況を示します。[WORK_ESTIMATED](#) および [WORK_COMPLETED](#) ユニットはファイルチャンクです。転送されるファイルの数はファイルコピーフェーズの開始時に認識され、チャンクの数にはファイルの数に基づいて推定されます。[WORK_ESTIMATED](#) は、推定ファイルチャンクの数に設定されます。[WORK_COMPLETED](#) は、各チャンクの送信後に更新されます。
- [stage/innodb/clone \(page copy\)](#): クローニング操作のページコピーフェーズの進行状況を示します。[WORK_ESTIMATED](#) および [WORK_COMPLETED](#) ユニットはページです。ファイルのコピーフェーズが完了すると、転送されるページ数がわかり、[WORK_ESTIMATED](#) はこの値に設定されます。[WORK_COMPLETED](#) は、各ページの送信後に更新されます。
- [stage/innodb/clone \(redo copy\)](#): クローニング操作の redo コピーフェーズの進行状況を示します。[WORK_ESTIMATED](#) および [WORK_COMPLETED](#) ユニットは redo チャンクです。ページコピーフェーズが完了すると、転送される redo チャンクの数があり、[WORK_ESTIMATED](#) はこの値に設定されます。[WORK_COMPLETED](#) は、各チャンクの送信後に更新されます。

次の例は、クローニング操作を監視するために [stage/innodb/clone%](#) イベントインストゥルメントおよび関連コンシューマテーブルを有効にする方法を示しています。パフォーマンススキーマステージイベントインストゥルメントおよび関連コンシューマについては、[セクション27.12.5「パフォーマンススキーマステージイベントテーブル」](#)を参照してください。

1. [stage/innodb/clone%](#) インストゥルメントを有効にします：

```

mysql> UPDATE performance_schema.setup_instruments SET ENABLED = 'YES'
      WHERE NAME LIKE 'stage/innodb/clone%';

```

2. ステージイベントコンシューマテーブル ([events_stages_current](#)、[events_stages_history](#) および [events_stages_history_long](#) を含む) を有効にします。

```

mysql> UPDATE performance_schema.setup_consumers SET ENABLED = 'YES'
      WHERE NAME LIKE '%stages%';

```

3. クローニング操作を実行します。この例では、ローカルデータディレクトリが [cloned_dir](#) という名前のディレクトリにクローニングされます。

```

mysql> CLONE LOCAL DATA DIRECTORY = '/path/to/cloned_dir';

```

4. パフォーマンススキーマ [events_stages_current](#) テーブルをクエリーして、クローニング操作の進行状況を確認します。表示されるステージイベントは、進行中のクローニングフェーズによって異なります。[WORK_COMPLETED](#) カラムには、完了した作業が表示されます。[WORK_ESTIMATED](#) カラムには、必要な作業の合計が表示されます。

```

mysql> SELECT EVENT_NAME, WORK_COMPLETED, WORK_ESTIMATED FROM performance_schema.events_stages_current
      WHERE EVENT_NAME LIKE 'stage/innodb/clone%';

```

```

+-----+-----+-----+
| EVENT_NAME          | WORK_COMPLETED | WORK_ESTIMATED |
+-----+-----+-----+
| stage/innodb/clone (redo copy) |          1 |          1 |

```

クローニング操作が終了すると、`events_stages_current` テーブルは空のセットを返します。この場合、`events_stages_history` テーブルをチェックして、完了した操作のイベントデータを表示できます。例:

```
mysql> SELECT EVENT_NAME, WORK_COMPLETED, WORK_ESTIMATED FROM events_stages_history
WHERE EVENT_NAME LIKE 'stage/innodb/clone%';
```

EVENT_NAME	WORK_COMPLETED	WORK_ESTIMATED
stage/innodb/clone (file copy)	301	301
stage/innodb/clone (page copy)	0	0
stage/innodb/clone (redo copy)	1	1

パフォーマンススキーマクローンの計測を使用したクローニング操作のモニタリング

[Performance Schema](#) は、クローン操作の高度なパフォーマンス監視のためのインストゥルメンテーションを提供します。使用可能なクローンインストゥルメンテーションを表示するには、次のクエリーを発行します:

```
mysql> SELECT * FROM performance_schema.setup_instruments
WHERE NAME LIKE WHERE NAME LIKE '%clone%';
```

NAME	ENABLED
wait/synch/mutex/innodb/clone_snapshot_mutex	NO
wait/synch/mutex/innodb/clone_sys_mutex	NO
wait/synch/mutex/innodb/clone_task_mutex	NO
wait/io/file/innodb/innodb_clone_file	YES
stage/innodb/clone (file copy)	YES
stage/innodb/clone (redo copy)	YES
stage/innodb/clone (page copy)	YES
statement/abstract/clone	YES
statement/clone/local	YES
statement/clone/client	YES
statement/clone/server	YES
memory/innodb/clone	YES
memory/clone/data	YES

待機インストゥルメント

パフォーマンススキーマ待機インストゥルメントは、時間がかかるイベントを追跡します。クローン待機イベントインストゥルメントには次のものがあります:

- [wait/synch/mutex/innodb/clone_snapshot_mutex](#): クローンスナップショット mutex の待機イベントを追跡します。これは、(ドナーと受信者の) 動的スナップショットオブジェクトへのアクセスを複数のクローンスレッド間で同期します。
- [wait/synch/mutex/innodb/clone_sys_mutex](#): クローン sys mutex の待機イベントを追跡します。MySQL サーバーインスタンスには、クローンシステムオブジェクトが 1 つあります。この mutex は、ドナーと受信者のクローンシステムオブジェクトへのアクセスを同期します。クローンスレッド、その他のフォアグラウンドおよびバックグラウンドスレッドによって取得されます。
- [wait/synch/mutex/innodb/clone_task_mutex](#): クローンタスク管理に使用されるクローンタスク mutex の待機イベントを追跡します。clone_task_mutex はクローンスレッドによって取得されます。
- [wait/io/file/innodb/innodb_clone_file](#): クローンが動作するファイルのすべての I/O 待機操作を追跡します。

InnoDB mutex 待機の監視の詳細は、[セクション15.16.2「パフォーマンススキーマを使用した InnoDB Mutex 待機のモニタリング」](#)を参照してください。待機イベントの一般的な監視の詳細は、[セクション27.12.4「パフォーマンススキーマ待機イベントテーブル」](#)を参照してください。

証書のステージング

パフォーマンススキーマのステージイベントは、ステートメント実行プロセス中に発生する手順を追跡します。クローンステージイベントインストゥルメントには次のものがあります:

- [stage/innodb/clone \(file copy\)](#): クローニング操作のファイルコピーフェーズの進行状況を示します。

- [stage/innodb/clone \(redo copy\)](#): クローニング操作の redo コピーフェーズの進行状況を示します。
- [stage/innodb/clone \(page copy\)](#): クローニング操作のページコピーフェーズの進行状況を示します。

ステージイベントを使用したクローニング操作の監視の詳細は、[パフォーマンススキーマステージイベントを使用したクローニング操作の監視](#)を参照してください。ステージイベントの監視の一般情報は、[セクション27.12.5「パフォーマンススキーマステージイベントテーブル」](#)を参照してください。

ステートメント証書

パフォーマンススキーマのステートメントイベントは、ステートメントの実行を追跡します。クローン操作が開始されると、clone ステートメントインストゥルメントによって追跡される様々なステートメントタイプが平行で実行される場合があります。これらのステートメントイベントは、パフォーマンススキーマステートメントイベントテーブルで監視できます。実行されるステートメントの数は、[clone_max_concurrency](#) および [clone_autotune_concurrency](#) の設定によって異なります。

ステートメントイベントインストゥルメントのクローニングには、次のものが含まれます:

- [statement/abstract/clone](#): ローカル、クライアントまたはサーバー操作タイプとして分類される前に、クローン操作のステートメントイベントを追跡します。
- [statement/clone/local](#): ローカルクローン操作のクローンステートメントイベントを追跡します。[CLONE LOCAL](#) ステートメントの実行時に生成されます。
- [statement/clone/client](#): 受信者 MySQL サーバーインスタンスで発生するリモートクローニングステートメントイベントを追跡します。受信者で [CLONE INSTANCE](#) ステートメントを実行すると生成されます。
- [statement/clone/server](#): ドナー MySQL サーバーインスタンスで発生するリモートクローニングステートメントイベントを追跡します。受信者で [CLONE INSTANCE](#) ステートメントを実行すると生成されます。

パフォーマンススキーマのステートメントイベントのモニタリングについては、[セクション27.12.6「パフォーマンススキーマステートメントイベントテーブル」](#)を参照してください。

メモリーインストゥルメント

パフォーマンススキーマメモリーインストゥルメントはメモリー使用状況を追跡します。クローンメモリー使用量インストゥルメントには次のものがあります:

- [memory/innodb/clone](#): 動的スナップショット用に InnoDB によって割り当てられたメモリーを追跡します。
- [memory/clone/data](#): クローン操作中にクローンプラグインによって割り当てられたメモリーを追跡します。

パフォーマンススキーマを使用したメモリー使用量のモニタリングについては、[セクション27.12.18.10「メモリーサマリーテーブル」](#)を参照してください。

Com_clone ステータス変数

[Com_clone](#) ステータス変数は、[CLONE](#) ステートメントの実行回数を提供します。

詳細は、[セクション5.1.10「サーバーステータス変数」](#)の [Com_xxx](#) ステートメントカウンタ変数に関する説明を参照してください。

5.6.7.10 クローニング操作の停止

必要に応じて、[KILL QUERY processlist_id](#) ステートメントを使用してクローニング操作を停止できます。

受信者 MySQL サーバーインスタンスでは、[clone_status](#) テーブルの [PID](#) カラムからクローニング操作のプロセスリクエスト識別子 (PID) を取得できます。

```
mysql> SELECT * FROM performance_schema.clone_status\G
***** 1. row *****
      ID: 1
      PID: 8
      STATE: In Progress
      BEGIN_TIME: 2019-07-15 11:58:36.767
      END_TIME: NULL
      SOURCE: LOCAL INSTANCE
```



```

DESTINATION: /path/to/clone_dir/
ERROR_NO: 0
ERROR_MESSAGE:
BINLOG_FILE:
BINLOG_POSITION: 0
GTID_EXECUTED:

```

プロセスリスト識別子は、`INFORMATION_SCHEMA PROCESLIST` テーブルの `ID` カラム、`SHOW PROCESLIST` 出力の `Id` カラム、またはパフォーマンススキーマ `threads` テーブルの `PROCESLIST_ID` カラムから取得することもできます。PID 情報を取得するこれらの方法は、ドナーまたは受信者の MySQL サーバーインスタンスで使用できません。

5.6.7.11 クローンシステム変数リファレンス

表 5.6 「クローンシステム変数リファレンス」

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
<code>clone_autotune_concurrency</code>	はい	はい	はい		グローバル	はい
<code>clone_buffer_size</code>	はい	はい	はい		グローバル	はい
<code>clone_ddl_timeout</code>	はい	はい	はい		グローバル	はい
<code>clone_enable_compression</code>	はい	はい	はい		グローバル	はい
<code>clone_max_concurrency</code>	はい	はい	はい		グローバル	はい
<code>clone_max_data_size</code>	はい	はい	はい		グローバル	はい
<code>clone_max_network_bandwidth</code>	はい	はい	はい		グローバル	はい
<code>clone_ssl_ca</code>	はい	はい	はい		グローバル	はい
<code>clone_ssl_cert</code>	はい	はい	はい		グローバル	はい
<code>clone_ssl_key</code>	はい	はい	はい		グローバル	はい
<code>clone_valid_donors</code>	はい	はい	はい		グローバル	はい

5.6.7.12 クローンシステム変数

このセクションでは、クローンプラグインの操作を制御するシステム変数について説明します。起動時に指定された値が正しくない場合、クローンプラグインが正しく初期化されず、サーバーがロードしない可能性があります。この場合、サーバーは他のクローン設定を認識しないため、エラーメッセージを生成することもあります。

各システム変数にはデフォルト値があります。システム変数は、コマンド行のオプションを使用するか、オプションファイルでサーバー起動時に設定できます。これらは実行時に `SET` ステートメントを使用して動的に変更できます。これにより、サーバーを停止して再起動しなくても、サーバーの操作を変更できます。

グローバルシステム変数のランタイム値を設定するには、通常、`SYSTEM_VARIABLES_ADMIN` 権限 (または非推奨の `SUPER` 権限) が必要です。詳細は、[セクション 5.1.9.1 「システム変数権限」](#) を参照してください。

クローン変数は、クローニング操作が実行される受信者 MySQL サーバーインスタンスで構成されます。

- `clone_autotune_concurrency`

コマンド行形式	<code>--clone-autotune-concurrency</code>
導入	8.0.17
システム変数	<code>clone_autotune_concurrency</code>
スコープ	グローバル
動的	はい
<code>SET_VAR</code> ヒントの適用	いいえ
型	Boolean
デフォルト値	<code>ON</code>

`clone_autotune_concurrency` が有効になっている場合 (デフォルト)、リモートクローニング操作の追加スレッドが動的に生成され、データ転送速度が最適化されます。この設定は、受信者の MySQL サーバーインスタンスにのみ適用できます。

クローニング操作中、スレッド数は現在のスレッド数の倍精度のターゲットに対して増分的に増加します。データ転送速度への影響は、増分ごとに評価されます。プロセスは、次のルールに従って続行または停止します:

- 増分増加によってデータ転送速度が 5% を超えると、プロセスは停止します。
- ターゲットの 25% に達した後で 5% 以上の改善があった場合、プロセスは続行されます。それ以外の場合、プロセスは停止します。
- ターゲットの 50% に達した後で 10% 以上の改善があった場合、プロセスは続行されます。それ以外の場合、プロセスは停止します。
- ターゲットへの到達後に 25% 以上の改善があった場合、プロセスは現在のスレッド数の倍精度の新しいターゲットに進みます。それ以外の場合、プロセスは停止します。

自動チューニングプロセスでは、スレッド数の減少はサポートされていません。

`clone_max_concurrency` 変数は、生成できるスレッドの最大数を定義します。

`clone_autotune_concurrency` が無効になっている場合、`clone_max_concurrency` はリモートクローニング操作に生成されるスレッドの数を定義します。

- `clone_buffer_size`

コマンド行形式	<code>--clone-buffer-size</code>
導入	8.0.17
システム変数	<code>clone_buffer_size</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	4194304
最小値	1048576
最大値	268435456

ローカルクローニング操作中にデータを転送するときに使用される中間バッファのサイズを定義します。この設定は、リモートクローニング操作には適用されません。デフォルト値は 4 メビバイト (MiB) です。バッファサイズを大きくすると、I/O デバイスドライバでデータをパラレルにフェッチできるため、クローニングのパフォーマンスを向上させることができます。

- `clone_ddl_timeout`

コマンド行形式	<code>--clone-ddl-timeout</code>
導入	8.0.17
システム変数	<code>clone_ddl_timeout</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	300

最小値	0
最大値	2592000

クローニング操作の実行時にバックアップロックを待機する時間(秒)。この設定は、ドナーと受信者の両方の MySQL サーバーインスタンスに適用されます。クローニング操作は DDL 操作と同時に実行できません。ドナーおよび受信者の MySQL サーバーインスタンスでは、バックアップロックが必要です。クローニング操作は、現在の DDL 操作が終了するまで待機します。バックアップロックが取得されると、DDL 操作はクローニング操作が終了するまで待機する必要があります。値 0 は、クローニング操作でバックアップロックが取得されないことを意味します。この場合、DDL 操作が同時に試行されると、クローニング操作はエラーで失敗します。

- [clone_enable_compression](#)

コマンド行形式	<code>--clone-enable-compression</code>
導入	8.0.17
システム変数	clone_enable_compression
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

リモートクローニング操作中のネットワークレイヤーでのデータ圧縮を有効にします。圧縮により、ネットワーク帯域幅が節約されますが、CPU のコストがかかります。圧縮を有効にすると、データ転送速度が向上する場合があります。この設定は、受信者の MySQL サーバーインスタンスにのみ適用されます。

- [clone_max_concurrency](#)

コマンド行形式	<code>--clone-max-concurrency</code>
導入	8.0.17
システム変数	clone_max_concurrency
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	16
最小値	1
最大値	128

リモートクローニング操作の最大同時スレッド数を定義します。デフォルト値は 16 です。スレッドの数が多いほどクローニングのパフォーマンスは向上しますが、同時に許可されるクライアント接続の数も減少するため、既存のクライアント接続のパフォーマンスに影響する可能性があります。この設定は、受信者の MySQL サーバーインスタンスにのみ適用されます。

[clone_autotune_concurrency](#) が有効な場合(デフォルト)、[clone_max_concurrency](#) はリモートクローニング操作用に動的に生成できるスレッドの最大数です。[clone_autotune_concurrency](#) が無効になっている場合、[clone_max_concurrency](#) はリモートクローニング操作用に生成されるスレッドの数を定義します。

リモートクローニング操作には、スレッド当たり 1 メビバイト (MiB) の最小データ転送速度をお勧めします。リモートクローニング操作のデータ転送速度は、[clone_max_data_bandwidth](#) 変数によって制御されます。

- [clone_max_data_bandwidth](#)

コマンド行形式	<code>--clone-max-data-bandwidth</code>	997
---------	---	-----

導入	8.0.17
システム変数	clone_max_data_bandwidth
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	1048576

リモートクローニング操作の最大データ転送速度をメガバイト (MiB)/秒で定義します。この変数は、クローニング操作のパフォーマンスへの影響を管理するのに役立ちます。制限は、ドナーディスクの I/O 帯域幅が飽和してパフォーマンスに影響を与える場合のみ設定してください。値 0 は、クローニング操作を可能な限り高いデータ転送速度で実行できる「unlimited」を意味します。この設定は、受信者の MySQL サーバーインスタンスにのみ適用できます。

最小データ転送速度は、スレッドあたり 1 MiB/秒です。たとえば、8 つのスレッドがある場合、最小転送速度は 8 MiB/秒です。[clone_max_concurrency](#) 変数は、リモートクローニング操作に生成されるスレッドの最大数を制御します。

[clone_max_data_bandwidth](#) で指定されたリクエストされたデータ転送速度は、[performance_schema.clone_progress](#) テーブルの [DATA_SPEED](#) カラムで報告された実際のデータ転送速度とは異なる場合があります。クローニング操作が目的のデータ転送速度に達しておらず、使用可能な帯域幅がある場合は、受信者およびドナーでの I/O の使用状況を確認します。使用率の低い帯域幅がある場合、I/O が次にボトルネックになる可能性があります。

- [clone_max_network_bandwidth](#)

コマンド行形式	<code>--clone-max-network-bandwidth</code>
導入	8.0.17
システム変数	clone_max_network_bandwidth
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	1048576

リモートクローニング操作の最大およそのネットワーク転送速度をメガバイト (MiB)/秒で指定します。この変数は、ネットワーク帯域幅に対するクローニング操作のパフォーマンスへの影響を管理するために使用できます。ネットワーク帯域幅が飽和状態で、ドナーインスタンスのパフォーマンスに影響する場合にのみ設定する必要があります。値 0 は、ネットワーク経由で可能な限り高いデータ転送速度でクローニングできる「unlimited」を意味し、最高のパフォーマンスを提供します。この設定は、受信者の MySQL サーバーインスタンスにのみ適用できます。

- [clone_ssl_ca](#)

コマンド行形式	<code>--clone-ssl-ca=file_name</code>
導入	8.0.14
システム変数	clone_ssl_ca
スコープ	グローバル

動的	はい
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	empty string

認証局 (CA) ファイルへのパスを指定します。リモートクローニング操作用に暗号化された接続を構成するために使用します。この設定は受信者に対して構成され、ドナーへの接続時に使用されます。

- [clone_ssl_cert](#)

コマンド行形式	<code>--clone-ssl-cert=file_name</code>
導入	8.0.14
システム変数	clone_ssl_cert
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	empty string

公開キー証明書へのパスを指定します。リモートクローニング操作用に暗号化された接続を構成するために使用します。この設定は受信者に対して構成され、ドナーへの接続時に使用されます。

- [clone_ssl_key](#)

コマンド行形式	<code>--clone-ssl-key=file_name</code>
導入	8.0.14
システム変数	clone_ssl_key
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	empty string

秘密キーファイルのパスを指定します。リモートクローニング操作用に暗号化された接続を構成するために使用します。この設定は受信者に対して構成され、ドナーへの接続時に使用されます。

- [clone_valid_donor_list](#)

コマンド行形式	<code>--clone-valid-donor-list=value</code>
導入	8.0.17
システム変数	clone_valid_donor_list
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	NULL

リモートクローニング操作の有効なドナーホストアドレスを定義します。この設定は、受信者の MySQL サーバーインスタンスに適用されます。カンマ区切りの値リストは、次の形式で使用できます: 「HOST1:PORT1,HOST2:PORT2,HOST3:PORT3」。スペースは使用できません。

`clone_valid_donor_list` 変数は、クローンデータのソースを制御することで、セキュリティレイヤーを追加します。`clone_valid_donor_list` の構成に必要な権限は、リモートクローニング操作の実行に必要な権限とは異なります。リモートクローニング操作では、これらの権限を別のロールに割り当てることができます。`clone_valid_donor_list` を構成するには `SYSTEM_VARIABLES_ADMIN` 権限が必要ですが、リモートクローニング操作を実行するには `CLONE_ADMIN` 権限が必要です。

インターネットプロトコルバージョン 6 (IPv6) アドレス形式はサポートされていません。インターネットプロトコルバージョン 6 (IPv6) アドレス形式はサポートされていません。かわりに、IPv6 アドレスのエイリアスを使用できます。IPv4 アドレスはそのまま使用できます。

5.6.7.13 クローンプラグインの制限事項

クローンプラグインには、次の制限があります:

- `TRUNCATE TABLE` を含む DDL は、クローニング操作中は許可されません。この制限は、データソースを選択するときに考慮する必要があります。回避策は、データのクローニング中にブロックされる DDL 操作に対応できる専用のドナーインスタンスを使用することです。同時 DML が許可されます。
- 別の MySQL サーバーのバージョンまたはリリースからインスタンスをクローニングすることはできません。ドナーと受信者の MySQL サーバーのバージョンとリリースは完全に同じである必要があります。たとえば、MySQL 5.7 と MySQL 8.0 の間、または MySQL 8.0.19 と MySQL 8.0.20 の間でクローニングすることはできません。クローンプラグインは、MySQL 8.0.17 以上でのみサポートされます。
- 一度にクローニングできるのは単一の MySQL インスタンスのみです。単一のクローニング操作での複数の MySQL インスタンスのクローニングはサポートされていません。
- `mysqlx_port` で指定された X プロトコル ポートは、リモートクローニング操作ではサポートされていません (`CLONE INSTANCE` ステートメントでドナー MySQL サーバーインスタンスのポート番号を指定する場合)。
- クローンプラグインは、MySQL サーバー構成のクローニングをサポートしていません。受信者の MySQL サーバーインスタンスは、永続化されたシステム変数設定を含め、その構成を保持します ([セクション 5.1.9.3 「永続化されるシステム変数」](#) を参照。)
- クローンプラグインはバイナリログのクローニングをサポートしていません。
- クローンプラグインは、`InnoDB` に格納されているデータのみをクローニングします。その他のストレージエンジンデータはクローニングされません。`sys` スキーマを含む任意のスキーマに格納されている `MyISAM` および `CSV` テーブルは、空のテーブルとしてクローニングされます。
- MySQL Router を介したドナー MySQL サーバーインスタンスへの接続はサポートされていません。
- ローカルクローニング操作では、絶対パスで作成された一般的なテーブルスペースのクローニングはサポートされていません。ソーステーブルスペースファイルと同じパスを持つクローニングされたテーブルスペースファイルは、競合の原因となります。

5.6.8 MySQL プラグインサービス

MySQL サーバープラグインは、サーバー「`プラグインサービス。`」にアクセスできます プラグインサービスインタフェースは、プラグインが呼び出すことができるサーバー機能を公開することによって、プラグイン API を補完します。プラグインサービスの記述に関する開発者情報は、[MySQL Services for Plugins](#) を参照してください。次の各セクションでは、SQL および C 言語レベルで使用可能なプラグインサービスについて説明します。

5.6.8.1 ロックサービス

MySQL デイストリビューションは、次の 2 つのレベルでアクセスできるロックインタフェースを提供します:

- SQL レベルでは、サービスルーチンへのコールにマップされる一連のユーザー定義関数 (UDF) として。
- C 言語インタフェースとして、サーバープラグインまたはユーザー定義関数からプラグインサービスとして呼び出すことができます。

プラグインサービスの一般情報は、[セクション 5.6.8 「MySQL プラグインサービス」](#) を参照してください。ユーザー定義関数の一般情報は、[Adding a Loadable Function](#) を参照してください。

ロックインタフェースには、次の特性があります:

- ロックには 3 つの属性があります: 名前空間、ロック名、およびロックモードをロックします:
 - ロックは、ネームスペースとロック名の組合せによって識別されます。ネームスペースを使用すると、別々のネームスペースにロックを作成することで、異なるアプリケーションで同じロック名を競合せずに使用できます。たとえば、アプリケーション A および B がそれぞれ `ns1` および `ns2` のネームスペースを使用する場合、各アプリケーションは他のアプリケーションと干渉することなく、`lock1` および `lock2` のロック名を使用できます。
 - ロックモードは読取りまたは書込みのいずれかです。読取りロックは共有されます: セッションに特定のロック識別子に対する読取りロックがある場合、他のセッションは同じ識別子に対する読取りロックを取得できます。書込みロックは排他的です: セッションに特定のロック識別子に対する書込みロックがある場合、他のセッションは同じ識別子に対する読取りまたは書込みロックを取得できません。
- ネームスペースおよびロック名は、`NULL` 以外で空ではなく、最大 64 文字である必要があります。`NULL`、空の文字列または 64 文字を超える文字列として指定されたネームスペースまたはロック名は、`ER_LOCKING_SERVICE_WRONG_NAME` エラーになります。
- ロックインタフェースは名前空間とロック名をバイナリ文字列として扱うため、比較では大文字と小文字が区別されます。
- ロックインタフェースは、ロックを取得してロックを解放する関数を提供します。これらの関数を呼び出すために特別な権限は必要ありません。権限チェックは、コール側アプリケーションの役割を果たします。
- すぐに使用できない場合は、ロックを待機できます。ロック取得コールには、ロックの取得を待機する秒数を示す整数のタイムアウト値が必要です。ロックの取得に成功せずにタイムアウトに達すると、`ER_LOCKING_SERVICE_TIMEOUT` エラーが発生します。タイムアウトが 0 の場合、待機はなく、ロックをすぐに取得できない場合はエラーが発生します。
- ロックインタフェースは、異なるセッションでのロック取得コール間のデッドロックを検出します。この場合、ロックサービスは呼出し側を選択し、そのロック取得リクエストを `ER_LOCKING_SERVICE_DEADLOCK` エラーで終了します。このエラーによってトランザクションがロールバックされることはありません。デッドロックの場合にセッションを選択するには、ロックサービスは、書込みロックを保持するセッションよりも読取りロックを保持するセッションを優先します。
- セッションでは、単一のロック問合せコールを使用して複数のロックを取得できます。特定のコールについて、ロック取得はアトミックです: すべてのロックが取得されると、コールは成功します。ロックの取得に失敗した場合、呼出しはロックを取得せず、通常は `ER_LOCKING_SERVICE_TIMEOUT` または `ER_LOCKING_SERVICE_DEADLOCK` エラーで失敗します。
- セッションは、同じロック識別子 (ネームスペースとロック名の組合せ) に対して複数のロックを取得できます。これらのロックインスタンスは、読み取りロック、書き込みロック、またはその両方を組み合わせて使用できます。
- セッション内で取得されたロックは、`release-locks` 関数を呼び出すことによって明示的に解放されるか、セッションの終了時に暗黙的に解放されます (通常または異常)。トランザクションがコミットまたはロールバックされても、ロックは解放されません。
- セッション内では、解放された特定のネームスペースに対するすべてのロックがまとめて解放されます。

ロックサービスによって提供されるインタフェースは、`GET_LOCK()` および関連する SQL 関数によって提供されるインタフェースとは異なります (セクション 12.15 「ロック関数」を参照)。たとえば、`GET_LOCK()` はネームスペースを実装せず、排他ロックのみを提供し、個別の読取りおよび書込みロックは提供しません。

ロックサービス C インタフェース

このセクションでは、ロックサービスの C 言語インタフェースの使用法について説明します。かわりに UDF インタフェースを使用するには、[ロックサービス UDF インタフェース](#) を参照してください。ロックサービスインタフェースの一般的な特性は、[セクション 5.6.8.1 「ロックサービス」](#) を参照してください。プラグインサービスの一般情報は、[セクション 5.6.8 「MySQL プラグインサービス」](#) を参照してください。

ロックサービスを使用するソースファイルには、次のヘッダーファイルを含める必要があります:

```
#include <mysql/service_locking.h>
```

1 つ以上のロックを取得するには、次の関数を呼び出します:

```
int mysql_acquire_locking_service_locks(MYSQL_THD opaque_thd,
    const char* lock_namespace,
    const char**lock_names,
    size_t lock_num,
    enum enum_locking_service_lock_type lock_type,
    unsigned long lock_timeout);
```

引数の意味は次のとおりです:

- `opaque_thd`: スレッドハンドル。 `NULL` として指定した場合は、現在のスレッドのハンドルが使用されます。
- `lock_namespace`: ロックネームスペースを示す `NULL` で終わる文字列。
- `lock_names`: 取得するロックの名前を提供する `NULL` 終了文字列の配列。
- `lock_num`: `lock_names` 配列内の名前の数。
- `lock_type`: 読取りロックまたは書き込みロックを取得するためのロックモード (`LOCKING_SERVICE_READ` または `LOCKING_SERVICE_WRITE`)。
- `lock_timeout`: ロックの取得を待機してから中止する整数の秒数。

特定のネームスペースに対して取得したロックを解放するには、次の関数をコールします:

```
int mysql_release_locking_service_locks(MYSQL_THD opaque_thd,
    const char* lock_namespace);
```

引数の意味は次のとおりです:

- `opaque_thd`: スレッドハンドル。 `NULL` として指定した場合は、現在のスレッドのハンドルが使用されます。
- `lock_namespace`: ロックネームスペースを示す `NULL` で終わる文字列。

ロックサービスによって取得または待機されたロックは、パフォーマンススキーマを使用して SQL レベルで監視できます。詳細は、[ロックサービスの監視](#)を参照してください。

ロックサービス UDF インタフェース

このセクションでは、ロックサービスのユーザー定義関数 (UDF) インタフェースの使用方法について説明します。代わりに C 言語インタフェースを使用するには、[ロックサービス C インタフェース](#)を参照してください。ロックサービスインタフェースの一般的な特性は、[セクション5.6.8.1「ロックサービス」](#)を参照してください。ユーザー定義関数の一般情報は、[Adding a Loadable Function](#)を参照してください。

- [UDF ロックインターフェイスのインストールまたはアンインストール](#)
- [UDF ロックインターフェイスの使用](#)
- [ロックサービスの監視](#)
- [ロックサービス UDF インタフェースリファレンス](#)

UDF ロックインターフェイスのインストールまたはアンインストール

[ロックサービス C インタフェース](#)で説明されているロックサービスルーチンはサーバーに組み込まれているため、インストールする必要はありません。サービスルーチンへのコールにマップされるユーザー定義関数 (UDF) にも同じことは当てはまりません: UDF は使用する前にインストールする必要があります。このセクションでは、その方法について説明します。UDF のインストールに関する一般情報については、[セクション5.7.1「ユーザー定義関数のインストールおよびアンインストール」](#)を参照してください。

ロックサービス UDF は、`plugin_dir` システム変数で指定されたディレクトリにあるプラグインライブラリファイルに実装されます。ファイルベース名は `locking_service` です。ファイル名の接尾辞は、プラットフォームごとに異なります (たとえば、`.so` for Unix and Unix-like systems, `.dll` for Windows)。

ロックサービス UDF をインストールするには、`CREATE FUNCTION` ステートメントを使用して、プラットフォームの `.so` 接尾辞を必要に応じて調整します:

```
CREATE FUNCTION service_get_read_locks RETURNS INT
SONAME 'locking_service.so';
```

```
CREATE FUNCTION service_get_write_locks RETURNS INT
SONAME 'locking_service.so';
CREATE FUNCTION service_release_locks RETURNS INT
SONAME 'locking_service.so';
```

UDF がソースレプリケーションサーバーで使用されている場合は、レプリケーションの問題を回避するために、それらをすべてのレプリカサーバーにインストールします。

いったんインストールされると、UDF はアンインストールされるまでインストールされたままです。これらを削除するには、[DROP FUNCTION](#) ステートメントを使用します:

```
DROP FUNCTION service_get_read_locks;
DROP FUNCTION service_get_write_locks;
DROP FUNCTION service_release_locks;
```

UDF ロックインターフェイスの使用

ロックサービス UDF を使用する前に、[UDF ロックインターフェイスのインストールまたはアンインストール](#) で提供されている手順に従ってそれらをインストールします。

1 つ以上の読み取りロックを取得するには、次の関数を呼び出します:

```
mysql> SELECT service_get_read_locks('myspace', 'rlock1', 'rlock2', 10);
+-----+
| service_get_read_locks('myspace', 'rlock1', 'rlock2', 10) |
+-----+
| 1 |
+-----+
```

最初の引数はロックネームスペースです。最後の引数は、ロックの取得を待機する秒数を示す整数のタイムアウトです。間の引数はロック名です。

前述の例では、関数はロック識別子が `(myspace, rlock1)` および `(myspace, rlock2)` のロックを取得します。

読み取りロックではなく書き込みロックを取得するには、次の関数を呼び出します:

```
mysql> SELECT service_get_write_locks('myspace', 'wlock1', 'wlock2', 10);
+-----+
| service_get_write_locks('myspace', 'wlock1', 'wlock2', 10) |
+-----+
| 1 |
+-----+
```

この場合、ロック識別子は `(myspace, wlock1)` および `(myspace, wlock2)` です。

ネームスペースのすべてのロックを解除するには、次の関数を使用します:

```
mysql> SELECT service_release_locks('myspace');
+-----+
| service_release_locks('myspace') |
+-----+
| 1 |
+-----+
```

各ロック関数は、成功した場合はゼロ以外を返します。関数が失敗すると、エラーが発生します。たとえば、ロック名は空にできないため、次のエラーが発生します:

```
mysql> SELECT service_get_read_locks('myspace', '', 10);
ERROR 3131 (42000): Incorrect locking service lock name ''
```

セッションは、同じロック識別子に対して複数のロックを取得できます。別のセッションに識別子の書き込みロックがない限り、セッションは任意の数の読み取りロックまたは書き込みロックを取得できます。識別子に対する各ロックリクエストは、新しいロックを取得します。次のステートメントは、同じ識別子を持つ 3 つの書き込みロックを取得してから、同じ識別子に対して 3 つの読み取りロックを取得します:

```
SELECT service_get_write_locks('ns', 'lock1', 'lock1', 'lock1', 0);
SELECT service_get_read_locks('ns', 'lock1', 'lock1', 'lock1', 0);
```

この時点でパフォーマンススキーマ `metadata_locks` テーブルを調べると、同じ `(ns, lock1)` 識別子を持つ 6 つの個別ロックがセッションに保持されていることがわかります。(詳細は、[ロックサービスの監視](#) を参照してください。)

セッションは (`ns`, `lock1`) で少なくとも 1 つの書き込みロックを保持するため、他のセッションは読取りまたは書き込みのいずれのロックも取得できません。セッションが識別子の読取りロックのみを保持している場合、他のセッションはその識別子の読取りロックを取得できますが、書き込みロックは取得できません。

単一ロック取得コールのロックはアトミックに取得されますが、アトミック性はコール間で保持されません。したがって、`service_get_write_locks()` が結果セットの行ごとに 1 回コールされる次のようなステートメントの場合、アトミック性は個々のコールに対して保持されますが、ステートメント全体に対しては保持されません:

```
SELECT service_get_write_locks('ns', 'lock1', 'lock2', 0) FROM t1 WHERE ... ;
```

注意

ロックサービスは、指定されたロック識別子に対する成功したリクエストごとに個別のロックを返すため、単一のステートメントが多数のロックを取得する可能性があります。例:

```
INSERT INTO ... SELECT service_get_write_locks('ns', t1.col_name, 0) FROM t1;
```

これらのタイプのステートメントには、特定の悪影響がある場合があります。たとえば、ステートメントが途中で失敗してロールバックされた場合、障害ポイントまで取得されたロックは引き続き存在します。目的が、挿入された行と取得されたロックの間に対応するものである場合、その目的は満たされません。また、ロックが特定の順序で付与されることが重要な場合は、オプティマイザが選択する実行計画によって結果セットの順序が異なる可能性があることに注意してください。このような理由から、アプリケーションをステートメントごとに単一のロック取得コールに制限することをお勧めします。

ロックサービスの監視

ロックサービスは MySQL Server メタデータロックフレームワークを使用して実装されるため、パフォーマンススキーマ `metadata_locks` テーブルを検査して、取得または待機したロックサービスロックをモニターします。

まず、メタデータロックインストゥルメントを有効にします:

```
mysql> UPDATE performance_schema.setup_instruments SET ENABLED = 'YES'
-> WHERE NAME = 'wait/lock/metadata/sql/mdl';
```

次に、いくつかのロックを取得し、`metadata_locks` テーブルの内容を確認します:

```
mysql> SELECT service_get_write_locks('mynamespace', 'lock1', 0);
+-----+
| service_get_write_locks('mynamespace', 'lock1', 0) |
+-----+
| 1 |
+-----+
mysql> SELECT service_get_read_locks('mynamespace', 'lock2', 0);
+-----+
| service_get_read_locks('mynamespace', 'lock2', 0) |
+-----+
| 1 |
+-----+
mysql> SELECT OBJECT_TYPE, OBJECT_SCHEMA, OBJECT_NAME, LOCK_TYPE, LOCK_STATUS
-> FROM performance_schema.metadata_locks
-> WHERE OBJECT_TYPE = 'LOCKING SERVICE'\G
***** 1. row *****
OBJECT_TYPE: LOCKING SERVICE
OBJECT_SCHEMA: mynamespace
OBJECT_NAME: lock1
LOCK_TYPE: EXCLUSIVE
LOCK_STATUS: GRANTED
***** 2. row *****
OBJECT_TYPE: LOCKING SERVICE
OBJECT_SCHEMA: mynamespace
OBJECT_NAME: lock2
LOCK_TYPE: SHARED
LOCK_STATUS: GRANTED
```

ロックサービスロックの `OBJECT_TYPE` 値は `LOCKING SERVICE` です。これは、たとえば、`USER LEVEL LOCK` の `OBJECT_TYPE` を持つ `GET_LOCK()` 関数で取得されたロックとは異なります。

ロックネームスペース、名前およびモードは、`OBJECT_SCHEMA`、`OBJECT_NAME` および `LOCK_TYPE` カラムに表示されます。読取りロックおよび書き込みロックには、それぞれ `SHARED` および `EXCLUSIVE` の `LOCK_TYPE` 値があります。

`LOCK_STATUS` 値は、取得したロックの場合は `GRANTED`、待機中のロックの場合は `PENDING` です。あるセッションが書き込みロックを保持していて、別のセッションが同じ識別子を持つロックを取得しようとする、と、`PENDING` が表示されます。

ロックサービス UDF インタフェースリファレンス

ロックサービスへの SQL インタフェースは、このセクションで説明するユーザー定義関数を実装します。使用例については、[UDF ロックインターフェイスの使用](#) を参照してください。

これらの関数は、次の特性を共有します：

- 成功の場合、戻り値はゼロ以外です。それ以外の場合は、エラーが発生します。
- ネームスペースおよびロック名は、`NULL` 以外で空ではなく、最大 64 文字である必要があります。
- タイムアウト値は、エラーが発生するまでロックの取得を待機する秒数を示す整数である必要があります。タイムアウトが 0 の場合、待機はなく、ロックをすぐに取得できないと、関数はエラーを生成します。

次のロックサービス UDF を使用できます：

- `service_get_read_locks(namespace, lock_name[, lock_name] ..., timeout)`

指定されたタイムアウト値内にロックが取得されない場合、指定されたロック名を使用して、指定された名前空間内の 1 つ以上の読み取り (共有) ロックを取得し、エラーでタイムアウトします。

- `service_get_write_locks(namespace, lock_name[, lock_name] ..., timeout)`

指定されたタイムアウト値内にロックが取得されない場合、指定されたロック名を使用して、指定されたネームスペース内の 1 つ以上の書き込み (排他) ロックを取得し、エラーでタイムアウトします。

- `service_release_locks(namespace)`

指定されたネームスペースについて、`service_get_read_locks()` および `service_get_write_locks()` を使用して現在のセッション内で取得されたすべてのロックを解放します。

ネームスペースにロックがない場合は、エラーではありません。

5.6.8.2 キーリングサービス

MySQL Server は、内部コンポーネントおよびプラグインが機密情報を安全に格納して後で取得できるようにするキーリングサービスをサポートしています。MySQL ディストリビューションは、次の 2 つのレベルでアクセスできるキーリングインタフェースを提供します：

- SQL レベルでは、サービスルーチンへのコールにマップされる一連のユーザー定義関数 (UDF) として。
- C 言語インタフェースとして、サーバープラグインまたはユーザー定義関数からプラグインサービスとして呼び出すことができます。

このセクションでは、キーリングサービス関数を使用して、MySQL キーリングキーストアのキーを格納、取得および削除する方法について説明します。UDF を使用する SQL インタフェースについては、[セクション 6.4.4.10 「汎用キーリングキー管理関数」](#) を参照してください。一般的な鍵リング情報については、[セクション 6.4.4 「MySQL キーリング」](#) を参照してください。

キーリングサービスは、有効になっているベースとなるキーリングプラグインがあれば、それを使用します。キーリングプラグインが有効になっていない場合、キーリングサービスの呼び出しは失敗します。

キーストア内の「record」は、データ (キー自体) と、キーへのアクセスに使用される一意の識別子で構成されます。識別子には、次の 2 つの部分があります：

- `key_id`: キー ID またはキー名。mysql_ で始まる `key_id` 値は、MySQL Server によって予約されています。

- `user_id`: セッションの実効ユーザー ID。ユーザーコンテキストがない場合、この値は `NULL` にすることができます。値は実際には「user」である必要はありません。意味はアプリケーションによって異なります。

キーリング UDF インタフェースを実装する関数は、`CURRENT_USER()` の値を `user_id` 値としてキーリングサービス関数に渡します。

鍵リングサービス関数には、次のような共通の特性があります:

- 各関数は、成功した場合は 0、失敗した場合は 1 を返します。
- `key_id` 引数と `user_id` 引数は、キーリング内のどのキーを使用するかを示す一意の組合せを形成します。
- `key_type` 引数は、暗号化方法や使用目的など、キーに関する追加情報を提供します。
- キーリングサービス関数は、キー ID、ユーザー名、タイプおよび値をバイナリ文字列として扱うため、比較では大文字と小文字が区別されます。たとえば、`MyKey` と `mykey` の ID は異なるキーを参照します。

次のキーリングサービス関数を使用できます:

- `my_key_fetch()`

キーリングからキーを不明瞭化し、そのタイプとともに取得します。この関数は、戻されたキーおよびキータイプの格納に使用されるバッファにメモリーを割り当てます。コール元は、不要になったメモリーをゼロまたは不明瞭化してから解放する必要があります。

構文:

```
bool my_key_fetch(const char *key_id, const char **key_type,
                 const char* user_id, void **key, size_t *key_len)
```

引数:

- `key_id`, `user_id`: ペアとしての `NULL` で終了する文字列は、フェッチするキーを示す一意の識別子を形成します。
- `key_type`: バッファポインタのアドレス。この関数は、(キーの追加時に格納された) キーに関する追加情報を提供する `NULL` で終わる文字列へのポインタを格納します。
- `key`: バッファポインタのアドレス。この関数は、フェッチされたキーデータを含むバッファへのポインタを格納します。
- `key_len`: 関数が `*key` バッファのサイズをバイト単位で格納する変数のアドレス。

戻り値:

成功した場合は 0、失敗した場合は 1 を返します。

- `my_key_generate()`

指定されたタイプと長さの新しいランダムキーを生成し、キーリングに格納します。キーの長さは `key_len` で、`key_id` および `user_id` から形成された識別子に関連付けられます。型と長さの値は、基礎となるキーリングプラグインでサポートされている値と一致している必要があります。セクション6.4.4.8「サポートされているキーリングキーのタイプと長さ」を参照してください。

構文:

```
bool my_key_generate(const char *key_id, const char *key_type,
                    const char *user_id, size_t key_len)
```

引数:

- `key_id`, `user_id`: ペアとしての `NULL` で終了する文字列は、生成されるキーの一意的識別子を形成します。
- `key_type`: キーに関する追加情報を提供する `NULL` で終わる文字列。
- `key_len`: 生成されるキーのサイズ (バイト単位)。

戻り値:

成功した場合は 0、失敗した場合は 1 を返します。

- [my_key_remove\(\)](#)

キーリングからキーを削除します。

構文:

```
bool my_key_remove(const char *key_id, const char* user_id)
```

引数:

- [key_id](#), [user_id](#): ペアとしての NULL で終了する文字列は、削除されるキーの一意的識別子を形成します。

戻り値:

成功した場合は 0、失敗した場合は 1 を返します。

- [my_key_store\(\)](#)

キーリングにキーを不明瞭化して格納します。

構文:

```
bool my_key_store(const char *key_id, const char *key_type,  
                 const char* user_id, void *key, size_t key_len)
```

引数:

- [key_id](#), [user_id](#): ペアとしての NULL 終了文字列は、格納されるキーの一意的識別子を形成します。
- [key_type](#): キーに関する追加情報を提供する NULL で終わる文字列。
- [key](#): 格納されるキーデータを含むバッファ。
- [key_len](#): [key](#) バッファのサイズ (バイト単位)。

戻り値:

成功した場合は 0、失敗した場合は 1 を返します。

5.7 MySQL Server のユーザー定義関数

MySQL Server を使用すると、ユーザー定義関数 (UDF) を作成およびロードしてサーバー機能を拡張できます。サーバー機能は、UDF を使用して全体または一部に実装できます。また、独自の UDF を記述することもできます。

MySQL ディストリビューションには、いくつかのサーバー機能を実装する (実装に役立つ) UDF が含まれています:

- グループレプリケーションを使用すると、MySQL サーバーインスタンスのグループ全体で可用性の高い分散 MySQL サービスを作成でき、データ整合性、競合検出および解決、グループメンバーシップサービスがすべて組み込まれています。 [第18章「グループレプリケーション」](#) を参照してください。
- MySQL Enterprise Edition には、OpenSSL ライブラリに基づいて暗号化操作を実行する UDF が含まれます。 [セクション6.6「MySQL Enterprise Encryption」](#) を参照してください。
- MySQL Enterprise Edition には、マスキングおよび識別解除操作を実行するための SQL レベル API を提供する UDF が含まれています。 [セクション6.5.1「MySQL Enterprise Data Masking and De-Identification 要素」](#) を参照してください。
- MySQL Enterprise Edition には、接続およびクエリーアクティビティの監視およびロギング用の監査ロギングが含まれています。 [セクション6.4.5「MySQL Enterprise Audit」](#) および [セクション6.4.6「監査メッセージコンポーネント」](#) を参照してください。

- MySQL Enterprise Edition には、アプリケーションレベルのファイアウォールを実装するファイアウォール機能が含まれており、データベース管理者は、受け入れられたステートメントパターンの許可リストに対する照合に基づいて SQL ステートメントの実行を許可または拒否できます。 [セクション6.4.7「MySQL Enterprise Firewall」](#) を参照してください。
- クエリーリライタは、MySQL Server によって受信されたステートメントを調べ、サーバーが実行する前にリライトする可能性があります。 [セクション5.6.4「リライタクエリーリライトプラグイン」](#) を参照してください。
- バージョントークンを使用すると、アプリケーションが不正または古いデータへのアクセスを防ぐために使用できるサーバートークンを作成して同期できます。 [セクション5.6.6「バージョントークン」](#) を参照してください。
- MySQL キーリングは、機密情報用のセキュアな記憶域を提供します。 [セクション6.4.4「MySQL キーリング」](#) を参照してください。
- ロックサービスは、アプリケーションで使用するロックインタフェースを提供します。 [セクション5.6.8.1「ロックサービス」](#) を参照してください。
- クエリー属性にアクセスするための UDF。 [セクション9.6「クエリー属性」](#) を参照してください。

次の各セクションでは、UDF をインストールおよびアンインストールする方法と、実行時にどの UDF をインストールするかを決定し、それらに関する情報を取得する方法について説明します。ユーザー定義関数を示すテーブルについては、[セクション12.2「ユーザー定義関数参照」](#) を参照してください。UDF の書込みの詳細は、[Adding Functions to MySQL](#) を参照してください。

UDF は、UDF を直接インストールするのではなく、UDF を実装するコンポーネントをインストールすることによってインストールされる場合があります。特定の UDF の詳細は、UDF を含むサーバー機能のインストール手順を参照してください。

5.7.1 ユーザー定義関数のインストールおよびアンインストール

ユーザー定義関数 (UDF) は、使用する前にサーバーにロードする必要があります。MySQL では、実行時の手動 UDF ロードおよびサーバー起動時の自動ロードがサポートされています。

UDF がロードされている間、UDF に関する情報は [セクション5.7.2「ユーザー定義関数情報の取得」](#) で説明されているように使用できます。

- [ユーザー定義関数のインストール](#)
- [ユーザー定義関数のアンインストール](#)
- [ユーザー定義関数の再インストールまたはアップグレード](#)

ユーザー定義関数のインストール

UDF を手動でロードするには、`CREATE FUNCTION` ステートメントを使用します。例:

```
CREATE FUNCTION metaphor  
RETURNS STRING  
SONAME 'udf_example.so';
```

UDF ファイルのベース名は、プラットフォームによって異なります。一般的な接尾辞は、`.so` for Unix および Unix のようなシステム、`.dll` for Windows です。

`CREATE FUNCTION` には、次の効果があります:

- UDF がサーバーにロードされ、すぐに使用できるようになります。
- UDF が `mysql.func` システムテーブルに登録され、サーバーの再起動後も保持されます。このため、`CREATE FUNCTION` には、`mysql` システムデータベースに対する `INSERT` 権限が必要です。
- UDF は、インストールされている UDF に関する実行時情報を提供するパフォーマンススキーマ `user_defined_functions` テーブルに追加されます。 [セクション5.7.2「ユーザー定義関数情報の取得」](#) を参照してください。

UDF の自動ロードは、通常のサーバー起動シーケンス中に発生します:

- `mysql.func` テーブルに登録されている UDF がインストールされます。
- 起動時にインストールされるコンポーネントまたはプラグインは、関連 UDF を自動的にインストールする場合があります。
- UDF の自動インストールでは、UDF がパフォーマンススキーマ `user_defined_functions` テーブルに追加され、インストールされた UDF に関する実行時情報が提供されます。

`--skip-grant-tables` オプションを使用してサーバーを起動した場合、`mysql.func` テーブルに登録されている UDF はロードされず、使用できません。これは、コンポーネントまたはプラグインによって自動的にインストールされる UDF には適用されません。

ユーザー定義関数のアンインストール

UDF を削除するには、`DROP FUNCTION` ステートメントを使用します。例:

```
DROP FUNCTION metaphor;
```

`DROP FUNCTION` には、次の効果があります:

- UDF をアンロードして使用できないようにします。
- `mysql.func` システムテーブルから UDF が削除されます。このため、`DROP FUNCTION` には、`mysql` システムデータベースに対する `DELETE` 権限が必要です。UDF が `mysql.func` テーブルに登録されなくなったため、サーバーはその後の再起動時に UDF をロードしません。
- インストールされている UDF に関する実行時情報を提供する UDF がパフォーマンススキーマ `user_defined_functions` テーブルから削除されます。

`DROP FUNCTION` を使用して、`CREATE FUNCTION` を使用するのではなく、コンポーネントまたはプラグインによって自動的にインストールされる UDF を削除することはできません。このような UDF は、UDF をインストールしたコンポーネントまたはプラグインがアンインストールされると、自動的に削除されます。

ユーザー定義関数の再インストールまたはアップグレード

UDF に関連付けられた共有ライブラリを再インストールまたはアップグレードするには、`DROP FUNCTION` ステートメントを発行し、共有ライブラリをアップグレードしてから、`CREATE FUNCTION` ステートメントを発行します。最初に共有ライブラリをアップグレードしてから `DROP FUNCTION` を使用すると、サーバーが予期せず停止する可能性があります。

5.7.2 ユーザー定義関数情報の取得

パフォーマンススキーマ `user_defined_functions` テーブルには、現在インストールされているユーザー定義関数 (UDF) に関する情報が含まれています:

```
SELECT * FROM performance_schema.user_defined_functions;
```

`mysql.func` システムテーブルには、インストールされている UDF もリストされますが、`CREATE FUNCTION` を使用してインストールされている UDF のみがリストされます。`user_defined_functions` テーブルには、`CREATE FUNCTION` を使用してインストールされた UDF と、コンポーネントまたはプラグインによって自動的にインストールされた UDF がリストされます。この違いにより、どの UDF がインストールされているかを `mysql.func` より `user_defined_functions` で確認することをお勧めします。[セクション 27.12.19.12 「user_defined_functions テーブル」](#) を参照してください。

5.8 1 つのマシン上での複数の MySQL インスタンスの実行

状況によっては、MySQL の複数インスタンスを単一マシン上で実行する場合があります。既存の本番設定をそのままにして、新しい MySQL リリースをテストすることもできます。または、ユーザーが自分で管理する異なる `mysqld` サーバーへのアクセス権を別々のユーザーに与える場合があります。(たとえば、ユーザーは独立した MySQL インストールを異なるカスタム用に提供するインターネットサービスプロバイダである場合があります。)

インスタンスごとに異なる MySQL Server バイナリを使用したり、複数のインスタンスに対して同じバイナリを使用したり、この 2 つの方法を組み合わせたりすることが可能です。たとえば、MySQL 5.7 と MySQL 8.0 からそれぞれ

サーバーを実行し、異なるバージョンによって所定のワークロードがどのように処理されるかを確認することもできます。または、現在の本番バージョンの複数インスタンスを実行し、それぞれが異なるデータベースのセットを管理する場合もあります。

別個のサーバーバイナリを使用するかどうかにかかわらず、実行する各インスタンスは、いくつかの操作パラメータについて一意の値を使用して構成される必要があります。これにより、インスタンス間で競合するおそれなくなります。パラメータは、コマンド行、オプションファイル、または環境変数の設定によって設定できます。 [セクション 4.2.2「プログラムオプションの指定」](#) を参照してください。所定のインスタンスによって使用される値を表示するには、インスタンスに接続して、`SHOW VARIABLES` ステートメントを実行します。

MySQL インスタンスによって管理される主なりソースは、データディレクトリです。各インスタンスは異なるデータディレクトリを使用する必要があり、その場所は `--datadir=dir_name` オプションを使用して指定されます。各インスタンスをインスタンス独自のデータディレクトリで構成する方法と、構成を行わないことの危険についての警告は、 [セクション 5.8.1「複数のデータディレクトリのセットアップ」](#) を参照してください。

異なるデータディレクトリを使用することに加えて、いくつかのほかのオプションは、各サーバーインスタンスについて異なる値を持つ必要があります。

- `--port=port_num`

`--port` は、TCP/IP 接続のポート番号を制御します。または、ホストに複数のネットワークアドレスがある場合は、各サーバーが異なるアドレスをリスニングするように `bind_address` システム変数を設定できます。

- `--socket={file_name|pipe_name}`

`--socket` は、Unix 上の Unix ソケットファイルパスまたは Windows 上の名前付きパイプ名を制御します。Windows の場合、名前付きパイプ接続を許可するように構成されたサーバーについてのみ、個別のパイプ名を指定することが必要です。

- `--shared-memory-base-name=name`

このオプションは Windows でのみ使用されます。これは、クライアントが共有メモリーを使用して接続できるようにするために、Windows サーバーによって使用される共有メモリー名を指定します。共有メモリー接続を許可するように構成されたサーバーについてのみ、個別の共有メモリー名を指定することが必要です。

- `--pid-file=file_name`

このオプションは、サーバーがプロセス ID を書き込むファイルのパス名を示します。

次のログファイルオプションを使用した場合、これらの値はサーバーごとに異なっている必要があります。

- `--general_log_file=file_name`
- `--log-bin[=file_name]`
- `--slow_query_log_file=file_name`
- `--log-error[=file_name]`

ログファイルオプションについての詳細な説明は、 [セクション 5.4「MySQL Server ログ」](#) を参照してください。

パフォーマンスを高めるには、次のオプションをサーバーごとに異なるやり方で指定して、いくつかの物理ディスクに負荷を分散させることができます。

- `--tmpdir=dir_name`

異なる一時ディレクトリを作成すると、特定の一時ファイルを作成した MySQL Server を判別しやすくなります。

異なる場所に複数の MySQL インストールがある場合は、 `--basedir=dir_name` オプションを使用して各インストールのベースディレクトリを指定できます。これにより、各インスタンスは自動的に異なるデータディレクトリ、ログファイル、および PID ファイルを使用します。この理由は、これらの各パラメータのデフォルトが、基本ディレクトリに対して相対的に指定されるためです。この場合、指定する必要があるほかのオプションは、 `--socket` および `--port` オプションのみです。 `tar` ファイルバイナリ配布を使用して、異なるバージョンの MySQL をインストールするとします。これらは別の場所にインストールされるため、各インストールについてのサーバーを、対応する基本ディレクトリの下でコマンド `bin/mysqld_safe` を使用して開始することができます。 `mysqld_safe` によって、 `mysqld` に渡さ

れる適切な `--basedir` オプションが決定され、`--socket` および `--port` オプションのみを `mysqld_safe` に指定する必要があります。

あとのセクションで説明するように、適切なコマンドオプションを指定するか、環境変数を設定することによって、追加のサーバーを開始することができます。ただし、複数のサーバーをより永続的に実行する必要がある場合は、オプションファイルを使用して、サーバーに一意となる必要があるオプション値を各サーバーに指定する方が簡単です。`--defaults-file` オプションは、このために役立ちます。

5.8.1 複数のデータディレクトリのセットアップ

マシン上の各 MySQL インスタンスには、独自のデータディレクトリを持たせるようにします。場所は、`--datadir=dir_name` オプションを使用して指定します。

新しいインスタンス用のデータディレクトリをセットアップするには、さまざまな方法があります。

- 新しいデータディレクトリを作成する。
- 既存のデータディレクトリをコピーする。

次に、それぞれの方法について詳しく説明します。

警告

通常、2つのサーバーが同じデータベース内のデータを更新するようにするべきではありません。このようにすると、使用しているオペレーティングシステムが、障害のないシステムロックをサポートしない場合に、好ましくない意外な結果が生じるおそれがあります。(この警告にもかかわらず) 同じデータディレクトリを使用する複数のサーバーを実行し、これらのロギングが有効な場合、適切なオプションを使用して、各サーバーについて一意となるログファイル名を指定する必要があります。そうでなければ、サーバーは同じファイルにログを記録しようとします。

前述の予防策が遵守されたとしても、この種類のセットアップは `MyISAM` および `MERGE` テーブルでのみ機能し、ほかのストレージエンジンでは機能しません。さらに、データディレクトリを複数のサーバー間で共有することに対するこの警告は、NFS 環境では常に該当します。複数の MySQL Server が共通のデータディレクトリに NFS 経由でアクセスできるようにすることは、非常によくない方法です。主な問題は、NFS が速度のボトルネックになることです。これはそのように使用するためのものではありません。NFS のもう 1 つのリスクは、2つ以上のサーバーが相互に干渉しないような方法を考案する必要があるということです。通常、NFS ファイルロックは `lockd` デーモンで処理されますが、現在のところ、どのような状況でも 100% の信頼性でロックを実行できるプラットフォームは存在しません。

新規データディレクトリの作成

この方法では、データディレクトリは MySQL を最初にインストールしたときと同じ状態であり、MySQL アカウントのデフォルトセットを持ち、ユーザーデータはありません。

Unix では、データディレクトリを初期化します。セクション 2.10 「インストール後のセットアップとテスト」を参照してください。

Windows では、データディレクトリは MySQL 配布に含まれています。

- Windows 用の MySQL Zip アーカイブ配布には、未変更のままのデータディレクトリが格納されています。そのような配布を一時的な場所に解凍し、新規インスタンスをセットアップする場所にその `data` ディレクトリをコピーします。
- Windows MSI パッケージインストーラは、インストールされたサーバーが使用するデータディレクトリを作成および設定しますが、インストールディレクトリの下に `data` という名前の初期「template」データディレクトリも作成します。MSI パッケージを使用してインストールが実行されたあと、テンプレートデータディレクトリをコピーして、追加の MySQL インスタンスをセットアップすることができます。

既存のデータディレクトリのコピー

この方法を使用すると、データディレクトリ内に存在するすべての MySQL アカウントまたはユーザーデータは新しいデータディレクトリに引き継がれます。

1. データディレクトリを使用する既存の MySQL インスタンスを停止します。これは、保留中の変更をインスタンスがディスクにフラッシュするクリーンシャットダウンである必要があります。
2. 新規データディレクトリがあるべき場所にデータディレクトリをコピーします。
3. 既存のインスタンスによって使用されるオプションファイル `my.cnf` または `my.ini` をコピーします。これは新規インスタンスの基礎となります。
4. 元のデータディレクトリを参照するすべてのパス名が新規データディレクトリを参照するように、新規オプションファイルを変更します。さらに、インスタンスごとに一意でなければならない TCP/IP ポート番号やログファイルなど、ほかのオプションも変更します。インスタンスごとに一意でなければならないパラメータのリストについては、[セクション5.8「1つのマシン上での複数の MySQL インスタンスの実行」](#)を参照してください。
5. 新規インスタンスを起動し、新規オプションファイルを使用するよう指示します。

5.8.2 Windows 上での複数の MySQL インスタンスの実行

Windows 上で複数のサーバーを実行するには、適切な操作パラメータを付けてコマンド行からそれらを手動で起動するか、複数のサーバーを Windows サービスとしてインストールしてそのように実行することによって行うことができます。MySQL をコマンド行から実行するか、サービスとして実行するための一般的な手順については、[セクション2.3「Microsoft Windows に MySQL をインストールする」](#)に記載されています。次のセクションでは、サーバーごとに一意でなければならないデータディレクトリなどのオプションに対して異なる値を指定して、各サーバーを起動する方法について説明します。これらのオプションは、[セクション5.8「1つのマシン上での複数の MySQL インスタンスの実行」](#)にリストされています。

5.8.2.1 Windows コマンド行での複数の MySQL インスタンスの起動

単一の MySQL Server をコマンド行から手動で起動するための手順は[セクション2.3.4.6「Windows のコマンド行からの MySQL の起動」](#)に記載されています。複数のサーバーをこの方法で起動する場合、コマンド行またはオプションファイルで適切なオプションを指定することができます。オプションをオプションファイルに配置する方が便利ですが、各サーバーが確実にそれ独自のオプションのセットを取得するようにする必要があります。これを行うには、サーバーごとにオプションファイルを作成し、サーバーを実行するときに `--defaults-file` オプションを使用してファイル名をサーバーに指示します。

`C:\mydata1` のデータディレクトリを使用してポート 3307 で `mysqld` のインスタンスを実行し、`C:\mydata2` のデータディレクトリを使用してポート 3308 で別のインスタンスを実行するとします。次の手順を使用します。

1. 各データディレクトリが存在し、付与テーブルを格納する `mysql` データベースの独自のコピーも含まれていることを確認します。
2. 2つのオプションファイルを作成します。たとえば、次のような `C:\my-opts1.cnf` という名前のファイルを作成します。

```
[mysqld]
datadir = C:/mydata1
port = 3307
```

そして、次のような `C:\my-opts2.cnf` という名前の 2 番目のファイルを作成します。

```
[mysqld]
datadir = C:/mydata2
port = 3308
```

3. `--defaults-file` オプションを使用して、サーバー独自のオプションファイルを使用して各サーバーを起動します。

```
C:\> C:\mysql\bin\mysqld --defaults-file=C:\my-opts1.cnf
C:\> C:\mysql\bin\mysqld --defaults-file=C:\my-opts2.cnf
```

各サーバーはフォアグラウンドで起動するため (サーバーが後で終了するまで新しいプロンプトは表示されません)、これら 2 つのコマンドを別々のコンソールウィンドウで発行する必要があります。

サーバーをシャットダウンするには、適切なポート番号を使用して各サーバーに接続します。


```
C:\> C:\mysql\bin\mysqldadmin --port=3307 --host=127.0.0.1 --user=root --password shutdown
C:\> C:\mysql\bin\mysqldadmin --port=3308 --host=127.0.0.1 --user=root --password shutdown
```

上述のように構成されたサーバーは、クライアントが TCP/IP 経由で接続することを許可します。Windows のバージョンで名前付きパイプがサポートされていて、名前付きパイプ接続も許可する場合は、名前付きパイプを有効にするオプションを指定し、その名前を指定します。名前付きパイプ接続をサポートする各サーバーは、一意のパイプ名を使用する必要があります。たとえば、`C:\my-opts1.cnf` ファイルは次のように修正されることがあります。

```
[mysqld]
datadir = C:/mydata1
port = 3307
enable-named-pipe
socket = mypipe1
```

2 番目のサーバーで使用する `C:\my-opts2.cnf` も同様に修正します。そのあと、前に説明したようにサーバーを起動します。

共有メモリー接続を許可するサーバーについても同様の手順が適用されます。このような接続を有効にするには、`shared_memory` システム変数を有効にしてサーバーを起動し、`shared_memory_base_name` システム変数を設定して各サーバーに一意の共有メモリー名を指定します。

5.8.2.2 Windows サービスとして複数の MySQL インスタンスの起動

Windows 上では、MySQL Server は Windows サービスとして実行することができます。単一の MySQL サービスをインストール、制御、および削除する手順は、[セクション2.3.4.8「Windows のサービスとして MySQL を起動する」](#)に記載されています。

複数の MySQL サービスをセットアップするには、各インスタンスが、インスタンスごとに一意でなければならないほかのパラメータを使用することに加えて、異なるサービス名を使用するする必要があります。

次の手順について、`mysqld` サーバーを、`C:\mysql-5.5.9` および `C:\mysql-8.0.29` にそれぞれインストールされている 2 つの異なるバージョンの MySQL から実行するとします。(5.5.9 を本番サーバーとして実行しているが、8.0.29 を使用したテストも実行したい場合、このような状況になることがあります。)

MySQL を Windows サービスとしてインストールするには、`--install` または `--install-manual` オプションを使用します。これらのオプションについては、[セクション2.3.4.8「Windows のサービスとして MySQL を起動する」](#)を参照してください。

前述の説明によれば、複数のサービスをセットアップするにはいくつかの方法があります。次の手順では、いくつかの例を説明します。これらのいずれかを試行する前に、既存の MySQL サービスがあればシャットダウンして削除してください。

- 方法 1: いずれかの標準オプションファイル内ですべてのサービスのオプションを指定します。これを行うには、サーバーごとに異なるサービス名を使用します。5.5.9 `mysqld` をサービス名 `mysqld1` で実行し、8.0.29 `mysqld` をサービス名 `mysqld2` で実行するとします。この場合、`[mysqld1]` グループを 5.5.9 に対して使用し、`[mysqld2]` グループを 8.0.29 に対して使用することができます。たとえば、`C:\my.cnf` を次のようにセットアップすることができます。

```
# options for mysqld1 service
[mysqld1]
basedir = C:/mysql-5.5.9
port = 3307
enable-named-pipe
socket = mypipe1

# options for mysqld2 service
[mysqld2]
basedir = C:/mysql-8.0.29
port = 3308
enable-named-pipe
socket = mypipe2
```

各サービスに対して Windows が正しい実行可能プログラムを登録するようにするために、完全なサーバーパス名を使用して、サービスを次のようにインストールします。

```
C:\> C:\mysql-5.5.9\bin\mysqld --install mysqld1
C:\> C:\mysql-8.0.29\bin\mysqld --install mysqld2
```

サービスを開始するには、サービスマネージャ、`NET START` または `SC START` を適切なサービス名とともに使用します:

```
C:\> SC START mysqlq1
C:\> SC START mysqlq2
```

サービスを停止するには、サービスマネージャを使用するか、適切なサービス名を指定して `NET STOP` または `SC STOP` を使用します:

```
C:\> SC STOP mysqlq1
C:\> SC STOP mysqlq2
```

- 方法 2: 各サーバーのオプションを別々のファイルに指定し、サービスをインストールするときに `--defaults-file` を使用して、使用するファイルを各サーバーに指示します。この場合、それぞれのファイルで `[mysqld]` グループを使用してオプションをリストするようにします。

この方法を使用する場合、5.5.9 `mysqld` のオプションを指定するには、次のようなファイル `C:\my-opts1.cnf` を作成します。

```
[mysqld]
basedir = C:/mysql-5.5.9
port = 3307
enable-named-pipe
socket = mypipe1
```

8.0.29 `mysqld` については、次のようなファイル `C:\my-opts2.cnf` を作成します。

```
[mysqld]
basedir = C:/mysql-8.0.29
port = 3308
enable-named-pipe
socket = mypipe2
```

次のようにしてサービスをインストールします (各コマンドを単一行に入力します)。

```
C:\> C:\mysql-5.5.9\bin\mysqld --install mysqlq1
--defaults-file=C:\my-opts1.cnf
C:\> C:\mysql-8.0.29\bin\mysqld --install mysqlq2
--defaults-file=C:\my-opts2.cnf
```

MySQL Server をサービスとしてインストールし、`--defaults-file` オプションを使用する場合、サービス名がオプションより前になければなりません。

サービスをインストールしたあと、前の例と同じ方法でサービスを起動および停止します。

複数のサービスを削除するには、それぞれに `SC DELETE mysqlq_service_name` を使用します。または、`--remove` オプションの後にサービス名を指定して、それぞれに `mysqld --remove` を使用します。サービス名がデフォルト (MySQL) の場合は、`mysqld --remove` の使用時に省略できます。

5.8.3 Unix 上での複数の MySQL インスタンスの実行

注記

ここでの説明では、`mysqld_safe` を使用して MySQL の複数のインスタンスを起動します。RPM ディストリビューションを使用した MySQL インストールの場合、サーバーの起動と停止は、複数の Linux プラットフォーム上の `systemd` によって管理されます。これらのプラットフォームでは、`mysqld_safe` は不要であるためインストールされません。`systemd` を使用した複数の MySQL インスタンスの処理の詳細は、[セクション 2.5.9 「systemd を使用した MySQL Server の管理」](#) を参照してください。

Unix 上で複数の MySQL インスタンスを実行するための 1 つの方法は、デフォルトの TCP/IP ポートおよび Unix ソケットファイルが異なる別々のサーバーをコンパイルして、それぞれのサーバーが別々のネットワークインタフェースを待機するようにすることです。インストールごとに異なる基本ディレクトリ内にコンパイルすることで、コンパイル済みのデータディレクトリ、ログファイル、および PID ファイルの場所がサーバーごとに自動的に別々になります。

デフォルトの TCP/IP ポート番号 (3306) および Unix ソケットファイル (`/tmp/mysql.sock`) に対して既存の 5.7 サーバーが構成されていると仮定します。別の操作パラメータを持つ新しい 8.0.29 サーバーを構成するには、次のような `CMake` コマンドを使用します。

```
shell> cmake . -DMYSQL_TCP_PORT=port_number \  
-DMYSQL_UNIX_ADDR=file_name \  
-DCMAKE_INSTALL_PREFIX=/usr/local/mysql-8.0.29
```

ここで、`port_number` および `file_name` は、デフォルトの TCP/IP ポート番号および Unix ソケットファイルパス名と異なっている必要があり、`CMAKE_INSTALL_PREFIX` 値は、既存の MySQL インストールが存在する場所とは異なるインストールディレクトリを指定します。

MySQL Server が所定のポート番号を待機している場合、次のコマンドを使用して、基本ディレクトリおよび Unix ソケットファイル名などのいくつかの重要な構成可能変数に対して MySQL Server が使用中の操作パラメータを検出できます。

```
shell> mysqladmin --host=host_name --port=port_number variables
```

このコマンドによって表示される情報を使用すれば、追加のサーバーを構成するときに使用しないオプション値を見分けることができます。

ホスト名として `localhost` を指定した場合、`mysqladmin` では TCP/IP ではなく Unix ソケットファイルがデフォルトで使用されます。トランスポートプロトコルを明示的に指定するには、`--protocol={TCP|SOCKET|PIPE|MEMORY}` オプションを使用します。

異なる Unix ソケットファイルおよび TCP/IP ポート番号を使用して起動するためだけの理由で新しい MySQL Server をコンパイルする必要はありません。同じサーバーバイナリを使用し、実行時に異なるパラメータ値を使用してそれぞれのバイナリの起動を開始することも可能です。これを行う 1 つの方法は、コマンド行オプションを使用する方法です。

```
shell> mysqld_safe --socket=file_name --port=port_number
```

別のサーバーを起動するには、異なる `--socket` および `--port` オプションの値を指定し、`--datadir=dir_name` オプションを `mysqld_safe` に渡して、サーバーが別のデータディレクトリを使用するようにします。

または、サーバーごとのオプションを別々のオプションファイルに配置し、適切なオプションファイルへのパスを指定する `--defaults-file` オプションを使用して各サーバーを起動します。たとえば、2 つのサーバーインスタンスのオプションファイルの名前が `/usr/local/mysql/my.cnf` および `/usr/local/mysql/my.cnf2` の場合、次のようなコマンドでサーバーを起動します。

```
shell> mysqld_safe --defaults-file=/usr/local/mysql/my.cnf  
shell> mysqld_safe --defaults-file=/usr/local/mysql/my.cnf2
```

同様な効果を得るための別の方法は、環境変数を使用して Unix ソケットファイル名および TCP/IP ポート番号を設定する方法です。

```
shell> MYSQL_UNIX_PORT=/tmp/mysqld-new.sock  
shell> MYSQL_TCP_PORT=3307  
shell> export MYSQL_UNIX_PORT MYSQL_TCP_PORT  
shell> bin/mysqld --initialize --user=mysql  
shell> mysqld_safe --datadir=/path/to/datadir &
```

これはテスト用に使用するための 2 番目のサーバーを起動する簡単な方法です。この方法の利点は、同じシェルから起動するすべてのクライアントプログラムに対して環境変数設定が適用されるということです。したがって、これらのクライアントに対する接続は 2 番目のサーバーに自動的に送信されます。

MySQL プログラムに影響を及ぼすために使用できるほかの環境変数のリストは、[セクション4.9「環境変数」](#)に記載されています。

Unix の場合、複数のサーバーを起動する別の方法として、`mysqld_multi` スクリプトがあります。[セクション4.3.4「mysqld_multi — 複数の MySQL サーバーの管理」](#)を参照してください。

5.8.4 複数サーバー環境でのクライアントプログラムの使用

クライアント内にコンパイルされたものとは異なるネットワークインタフェースを待機している MySQL Server に対してクライアントプログラムを使用して接続するために、次のいずれかの方法を使用することができます。

- クライアントを起動する際、`--host=host_name --port=port_number` を指定することによって TCP/IP を使用してリモートサーバーに接続するか、`--host=127.0.0.1 --port=port_number` を指定することによって TCP/IP を使用してローカルサーバーに接続するか、`--host=localhost --socket=file_name` を指定することによって Unix ソケットファイルまたは Windows 名前付きパイプを使用してローカルサーバーに接続します。
- クライアントを起動する際、`--protocol=TCP` を指定することによって TCP/IP を使用して接続するか、`--protocol=SOCKET` を指定することによって Unix ソケットファイルを使用して接続するか、`--protocol=PIPE` を指定することによって名前付きパイプを使用して接続するか、`--protocol=MEMORY` を指定することによって共有メモリーを使用して接続します。TCP/IP 接続では、`--host` オプションと `--port` オプションも指定することが必要な場合もあります。ほかの接続タイプでは、`--socket` オプションを指定して Unix ソケットファイルまたは Windows 名前付きパイプ名を指定したり、`--shared-memory-base-name` オプションで共有メモリー名を指定したりすることが必要になることもあります。共有メモリー接続は Windows でのみサポートされます。
- Unix の場合、`MYSQL_UNIX_PORT` および `MYSQL_TCP_PORT` 環境変数を設定して、Unix ソケットファイルおよび TCP/IP ポート番号を指示してからクライアントを起動します。通常、特定のソケットファイルまたはポート番号を使用する場合、これらの環境変数を設定するためのコマンドを `.login` ファイルに配置して、ログインするたびにこれらが適用されるようにすることができます。[セクション4.9「環境変数」](#)を参照してください。
- オプションファイルの`[client]`グループ内のデフォルトの Unix ソケットファイルと TCP/IP ポート番号を指定します。たとえば、Windows の `C:\my.cnf` や、Unix のホームディレクトリにある `.my.cnf` ファイルを使用できます。[セクション4.2.2.2「オプションファイルの使用」](#)を参照してください。
- C プログラムでは、ソケットファイルまたはポート番号の引数を `mysql_real_connect()` の呼び出しで指定できます。また、`mysql_options()` を呼び出して、プログラムにオプションファイルを読み取らせることもできます。[C API Basic Function Descriptions](#)を参照してください。
- Perl の `DBD::mysql` モジュールを使用している場合、MySQL オプションファイルからオプションを読み取ることができます。例:

```
$dsn = "DBI:mysql:test:mysql_read_default_group=client;"
      ".mysql_read_default_file=/usr/local/mysql/data/my.cnf";
$dbh = DBI->connect($dsn, $user, $password);
```

[セクション29.9「MySQL Perl API」](#)を参照してください。

ほかのプログラミングインタフェースでも、オプションファイルの読み取りのための同様の機能を利用できることがあります。

5.9 MySQL のデバッグ

このセクションでは、MySQL での問題の追跡に役立つデバッグ手法について説明します。

5.9.1 MySQL サーバーのデバッグ

MySQL の非常に新しい機能を使用している場合は、`--skip-new` オプションを指定して `mysqld` の実行を試みることができます (これにより、安全でない可能性のある新しい機能がすべて無効になります)。[セクションB.3.3.3「MySQL が繰り返しクラッシュする場合の対処方法」](#)を参照してください。

`mysqld` を起動しない場合は、設定を妨げる `my.cnf` ファイルがないことを確認してください。`my.cnf` の引数をチェックするには `mysqld --print-defaults` を使用します。`mysqld --no-defaults ...` を指定して起動するとそれらの引数は使用されません。

`mysqld` が CPU やメモリーを使い尽くしてしまうようになった場合、または「ハングアップ」する場合は、`mysqldadmin processlist status` を使用すると、ユーザーが長時間かかるクエリーを実行しているかどうかを確認できます。新しいクライアントが接続できないときにパフォーマンスの問題または問題が発生した場合は、いくつかのウィンドウで `mysqldadmin -i10 processlist status` を実行することをお勧めします。

`mysqldadmin debug` コマンドは、使用されているロック、使用されているメモリー、およびクエリーの使用状況に関する情報を MySQL ログファイルにダンプします。これは一部の問題の解決に役立つことがあります。このコマンドは、デバッグのために MySQL をコンパイルしていない場合でも有用な情報を提供します。

一部のテーブルが徐々に遅くなるという問題がある場合は、`OPTIMIZE TABLE` または `myisamchk` を使用してテーブルを最適化することを試みてください。[第5章「MySQL サーバーの管理」](#)を参照してください。また、遅いクエリーを `EXPLAIN` でチェックしてください。

また、使用している環境に特有である可能性がある問題については、このマニュアルの OS 固有のセクションをお読みください。 [セクション2.1「一般的なインストールガイド」](#) を参照してください。

5.9.1.1 デバッグのための MySQL のコンパイル

非常に特定された問題がある場合は、MySQL のデバッグを試みることができます。これを実行するには、`-DWITH_DEBUG=1` オプションを指定して MySQL を構成する必要があります。 `mysqld --help` を実行すると、MySQL がデバッグ付きでコンパイルされたかどうかを確認できます。 `--debug` フラグがオプションとともに示される場合は、デバッグが有効にされています。この場合は、`mysqladmin ver` でも `mysqld` バージョンが `mysql ... --debug` として示されます。

`mysqld` に `-DWITH_DEBUG=1` CMake オプションを指定して構成するとクラッシュして停止する場合は、MySQL 内のコンパイラのバグまたはタイミングのバグが検出された可能性があります。この場合は、`-DWITH_DEBUG=1` を使用せずに、`CMAKE_C_FLAGS` および `CMAKE_CXX_FLAGS` CMake オプションを使用して `-g` の追加を試みることができます。 `mysqld` が異常終了した場合は、少なくとも `gdb` を使用してそれに接続するか、コアファイルに対して `gdb` を使用して、発生した事象を確認できます。

MySQL をデバッグ用に構成すると、`mysqld` のヘルスをモニターする多くの追加の安全チェック機能が自動的に有効になります。「予期しない」現象が検出された場合、`stderr` にエントリが書き込まれ、これは `mysqld_safe` によってエラーログに送られます。これは、MySQL に予期しない問題があり、ソース配布を使用している場合、最初にデバッグ用に MySQL を構成する必要があることも意味します。バグを見つけたと思われる場合は、[セクション1.6「質問またはバグをレポートする方法」](#) の手順を使用してください。

Windows の MySQL 配布では、`mysqld.exe` はデフォルトでトレースファイルをサポートするようにコンパイルされています。

5.9.1.2 トレースファイルの作成

`mysqld` サーバーが起動しないかクラッシュしやすい場合は、トレースファイルを作成して問題を見つけることができます。

これを行うには、デバッグサポート付きでコンパイルされている `mysqld` がある必要があります。これは `mysqld -V` を実行することによって確認できます。バージョン番号が `-debug` で終わっている場合は、トレースファイルのサポート付きでコンパイルされています。(Windows では、デバッグサーバーの名前は `mysqld` ではなく `mysqld-debug` になります。)

Unix の場合は `/tmp/mysqld.trace`、Windows の場合は `\mysqld.trace` にあるトレースログを使用して、`mysqld` サーバーを起動します。

```
shell> mysqld --debug
```

Windows では、`mysqld` をサービスとして起動しないようにするために、`--standalone` フラグも使用してください。コンソールウィンドウで、次のコマンドを使用します。

```
C:\> mysqld-debug --debug --standalone
```

このあと、2つ目のコンソールウィンドウで `mysql.exe` コマンド行ツールを使用して、問題を再現できます。 `mysqld` サーバーを停止するには、`mysqladmin shutdown` を使用します。

トレースファイルは非常に大きくなる場合があります。より小さいトレースファイルが生成されるようにするには、次のようなデバッグオプションを使用できます。

```
mysqld --debug=d,info,error,query,general,where:O,/tmp/mysqld.trace
```

これにより、もっとも関心があるタグの情報のみがトレースファイルに出力されます。

バグをファイルする場合は、トレースファイルの行のみをバグレポートに追加して、問題が発生している可能性がある場所を示してください。間違った場所が見つからない場合は、バグレポートを開き、トレースファイル全体をレポートにアップロードして、MySQL 開発者が確認できるようにします。その手順は、[セクション1.6「質問またはバグをレポートする方法」](#) を参照してください。

トレースファイルは、Fred Fish が作成した DBUG パッケージによって生成されます。 [セクション5.9.4「DBUG パッケージ」](#) を参照してください。

5.9.1.3 WER と PDB を使用した Windows クラッシュダンプの作成

プログラムデータベースファイル (接尾辞 `pdb`) は、MySQL の ZIP アーカイブデバッグバイナリ & テストスイートディストリビューションに含まれています。これらのファイルには、問題が発生したときに MySQL インストール環境をデバッグするための情報が含まれています。これは、標準 MSI または Zip ファイルとは別のダウンロードです。

注記

PDB ファイルは、ZIP アーカイブデバッグバイナリおよびテストスイートというラベルの付いた別のファイルで使用できます。

PDB ファイルには、より詳細なトレースファイルおよびダンプファイルを作成できる、`mysqld` およびその他のツールについての詳細な情報が含まれています。これらを `WinDbg` または `Visual Studio` とともに使用して、`mysqld` をデバッグできます。

PDB ファイルについては、[Microsoft サポート技術情報の記事 121366](#) を参照してください。使用可能なデバッグオプションについては、[Windows のデバッグツール](#) を参照してください。

`WinDbg` を使用するには、完全な Windows ドライバキット (WDK) をインストールするか、スタンドアロンバージョンをインストールします。

重要

`.exe` ファイルと `.pdb` ファイルは完全に一致している必要があります (バージョン番号と MySQL サーバーエディションの両方)。一致していない場合、`WinDBG` はシンボルのロード中に苦情します。

1. ミニダンプ `mysqld.dmp` を生成するには、`my.ini` の `[mysqld]` セクションで `core-file` オプションを有効にします。これらの変更を行った後、MySQL サーバーを再起動します。
2. `c:\symbols` など、生成されたファイルを格納するディレクトリを作成
3. 検索 GUI またはコマンドラインを使用して、`windbg.exe` 実行可能ファイルへのパスを確認します。次に例を示します: `dir /s /b windbg.exe` -- 一般的なデフォルトは `C:\Program Files\Debugging Tools for Windows (x64)\windbg.exe` です
4. `mysqld.exe`, `mysqld.pdb`, `mysqld.dmp` へのパスおよびソースコードを指定して、`windbg.exe` を起動します。または、`WinDbg` GUI から各パスを渡します。例:

```
windbg.exe -i "C:\mysql-8.0.29-win64\bin\"^
-z "C:\mysql-8.0.29-win64\data\mysqld.dmp"^
-srcpath "E:\ade\mysql_archives\8.0\8.0.29\mysql-8.0.29"^
-y "C:\mysql-8.0.29-win64\bin;SRV*c:\symbols*http://msdl.microsoft.com/download/symbols"^
-v -n -c "!analyze -vvvv"
```

注記

^ の文字と改行は Windows コマンドラインプロセッサによって削除されるため、空白はそのままにしてください。

5.9.1.4 gdb での mysqld のデバッグ

ほとんどのシステムでは、`mysqld` がクラッシュした場合に詳細な情報を取得するために、`gdb` から `mysqld` を起動できます。

Linux の一部の古い `gdb` バージョンでは、`mysqld` のスレッドをデバッグできるようにするには、`run --one-thread` を使用する必要があります。この場合、一度にアクティブにできるのは 1 つのスレッドのみです。

`gdb` で `mysqld` を実行すると、NPTL スレッド (Linux の新しいスレッドライブラリ) に起因する問題が発生することがあります。次のような現象が発生します。

- `mysqld` が起動中 (「接続準備完了」と出力される前) にハングアップする。
- `mysqld` が `pthread_mutex_lock()` または `pthread_mutex_unlock()` の呼び出し中にクラッシュする。

この場合、`gdb` を起動する前に、シェルで次の環境変数を設定してください。

```
LD_ASSUME_KERNEL=2.4.1
export LD_ASSUME_KERNEL
```

`gdb` で `mysqld` を実行するときは、`--skip-stack-trace` を使用してスタックトレースを無効にし、`gdb` 内でセグメンテーション違反を捕捉できるようにする必要があります。

`mysqld` の `--gdb` オプションを使用して、`SIGINT` の割り込みハンドラ (^C でブレークポイントを設定するために `mysqld` を停止するために必要) をインストールし、スタックトレースおよびコアファイル処理を無効にします。

`gdb` では古いスレッドのメモリーが解放されないため、すべての時間に多数の新しい接続を行う場合、`gdb` で MySQL をデバッグすることは非常に困難です。この問題を回避するには、`thread_cache_size` を `max_connections + 1` に等しい値に設定して `mysqld` を起動します。ほとんどの場合、`--thread_cache_size=5` を使用するだけでもかなり改善されます。

`SIGSEGV` シグナルが発生して `mysqld` が異常終了したときに Linux でコアダンプを取得する場合は、`--core-file` オプションを指定して `mysqld` を起動します。このコアファイルは、`mysqld` が異常終了した理由を見つけるために役立つことがあるバックトレースを作成するために使用できます。

```
shell> gdb mysqld core
gdb> backtrace full
gdb> quit
```

[セクションB.3.3.3「MySQL が繰り返しクラッシュする場合の対処方法」](#)を参照してください。

Linux で `gdb` を使用している場合は、次の情報を含む `.gdb` ファイルを現在のディレクトリにインストールする必要があります：

```
set print sevenbit off
handle SIGUSR1 nostop noprint
handle SIGUSR2 nostop noprint
handle SIGWAITING nostop noprint
handle SIGLWP nostop noprint
handle SIGPIPE nostop
handle SIGALRM nostop
handle SIGHUP nostop
handle SIGTERM nostop noprint
```

`mysqld` をデバッグする方法の例を次に示します。

```
shell> gdb /usr/local/libexec/mysqld
gdb> run
...
backtrace full # Do this when mysqld crashes
```

上記の出力をバグレポートに含め、[セクション1.6「質問またはバグをレポートする方法」](#)の手順を使用してバグレポートを提出できます。

`mysqld` がハングアップした場合は、`strace`、`/usr/proc/bin/pstack` などのシステムツールを使用して、`mysqld` がハングアップした場所を調査できます。

```
strace /tmp/log libexec/mysqld
```

Perl の `DBI` インタフェースを使用している場合は、`trace` メソッドを使用するか、`DBI_TRACE` 環境変数を設定することによって、デバッグ情報をオンにできます。

5.9.1.5 スタックトレースの使用

オペレーティングシステムによっては、`mysqld` が予期せずに異常終了した場合に、エラーログにスタックトレースが含まれています。これを使用して、`mysqld` が異常終了した場所 (および多くの場合その理由) を見つけることができます。[セクション5.4.2「エラーログ」](#)を参照してください。スタックトレースを取得するには、`-fomit-frame-pointer` オプションを `gcc` に指定して `mysqld` をコンパイルしないでください。[セクション5.9.1.1「デバッグのためのMySQLのコンパイル」](#)を参照してください。

エラーログのスタックトレースは次のよう出力されます。

```
mysqld got signal 11;
Attempting backtrace. You can use the following information
```

```
to find out where mysqld died. If you see no messages after
this, something went terribly wrong...
```

```
stack_bottom = 0x41fd0110 thread_stack 0x40000
mysqld(my_print_stacktrace+0x32)[0x9da402]
mysqld(handle_segfault+0x28a)[0x6648e9]
/lib/libpthread.so.0[0x7f1a5af00f0]
/lib/libc.so.6(strcmp+0x2)[0x7f1a5a10f0f2]
mysqld(_Z21check_change_passwordP3THDPKcS2_Pcj+0x7c)[0x7412cb]
mysqld(_ZN16set_var_password5checkEP3THD+0xd0)[0x688354]
mysqld(_Z17sql_set_variablesP3THDP4Listl12set_var_baseE+0x68)[0x688494]
mysqld(_Z21mysql_execute_commandP3THD+0x41a0)[0x67a170]
mysqld(_Z11mysql_parseP3THDPKcjPS2_+0x282)[0x67f0ad]
mysqld(_Z16dispatch_command19enum_server_commandP3THDPcj+0xbb7)[0x67fdf8]
mysqld(_Z10do_commandP3THD+0x24d)[0x6811b6]
mysqld(handle_one_connection+0x11c)[0x66e05e]
```

トレースの関数名の解決に失敗した場合、トレースに格納される情報が少なくなります。

```
mysqld got signal 11;
Attempting backtrace. You can use the following information
to find out where mysqld died. If you see no messages after
this, something went terribly wrong...
```

```
stack_bottom = 0x41fd0110 thread_stack 0x40000
[0x9da402]
[0x6648e9]
[0x7f1a5af00f0]
[0x7f1a5a10f0f2]
[0x7412cb]
[0x688354]
[0x688494]
[0x67a170]
[0x67f0ad]
[0x67fdf8]
[0x6811b6]
[0x66e05e]
```

新しいバージョンの `glibc` スタックトレース関数では、オブジェクトへの相対アドレスも出力されます。 `glibc` ベースのシステム (Linux) では、プラグイン内の予期しない終了のトレースは次のようになります:

```
plugin/auth/auth_test_plugin.so(+0x9a6)[0x7ff4d11c29a6]
```

相対アドレス (+0x9a6) をファイル名および行番号に変換するには、次のコマンドを使用します。

```
shell> addr2line -fie auth_test_plugin.so 0x9a6
auth_test_plugin
mysql-trunk/plugin/auth/test_plugin.c:65
```

`addr2line` ユーティリティーは Linux の `binutils` パッケージの一部です。

Solaris でも手順は同様です。Solaris の `printstack()` では、相対アドレスがすでに出力されています。

```
plugin/auth/auth_test_plugin.so:0x1510
```

これを変換するには、次のコマンドを使用します。

```
shell> gaddr2line -fie auth_test_plugin.so 0x1510
mysql-trunk/plugin/auth/test_plugin.c:88
```

Windows では、アドレス、関数名、および行がすでに出力されています。

```
000007FEF07E10A4 auth_test_plugin.dll!auth_test_plugin()[test_plugin.c:72]
```

5.9.1.6 mysqld でのエラーの原因を見つけるためのサーバーログの使用

一般クエリログを有効にして `mysqld` を起動する前に、`myisamchk` を使用してすべてのテーブルをチェックしてください。第5章「MySQL サーバーの管理」を参照してください。

`mysqld` が異常終了またはハングアップする場合は、一般クエリログを有効にして `mysqld` を起動してください。セクション 5.4.3 「一般クエリログ」を参照してください。 `mysqld` がふたたび異常終了したら、ログファイルの最後の部分を調査して、`mysqld` が強制終了されたクエリを見つけることができます。

デフォルトの一般クエリログファイルを使用した場合、ログはデータベースディレクトリに `host_name.log` として格納されます。ほとんどの場合、`mysqld` が強制終了されたのはログファイル内の最後のクエリですが、可能であれば、`mysqld` を再起動して、見つかったクエリを `mysql` コマンド行ツールから実行することによって、このことを検証してください。これが機能する場合は、完了しなかった複雑なクエリもすべてテストする必要があります。

また、長い時間がかかるすべての `SELECT` ステートメントに対して `EXPLAIN` コマンドを試すことで、`mysqld` がインデックスを適切に使用していることを確認できます。セクション13.8.2「`EXPLAIN` ステートメント」を参照してください。

実行に長い時間がかかるクエリを見つけるには、スロークエリログを有効にして `mysqld` を起動します。セクション5.4.5「スロークエリログ」を参照してください。

エラーログ (通常は `host_name.err` という名前のファイル) にテキスト `mysqld restarted` が見つかった場合、`mysqld` が失敗する原因となるクエリが見つかった可能性があります。これが発生した場合、`myisamchk` を使用してすべてのテーブルをチェックし (第5章「MySQL サーバーの管理」を参照してください)、MySQL ログファイル内のクエリをテストして、失敗するかどうかを確認します。そのようなクエリが見つかった場合は、まず最新バージョンの MySQL にアップグレードすることを試してください。問題が解決しない場合は、バグを報告してください。セクション1.6「質問またはバグをレポートする方法」を参照してください。

`myisam_recover_options` システム変数を設定して `mysqld` を起動した場合、`MyISAM` テーブルが正しくクローズされていないまたはクラッシュとマークされていれば、MySQL によって自動的にチェックされ、修復が試行されます。これが発生した場合、MySQL は `hostname.err` ファイルに「警告: テーブル ... をチェックしています」と書き込み、テーブルを修復する必要がある場合は、「警告: テーブルを修復しています」がそのあとに書き込まれます。これらのエラーを多数受け取り、その直前に予期しない `mysqld` の停止がなかった場合は、何らかの問題があるため、さらに調査する必要があります。セクション5.1.7「サーバーコマンドオプション」を参照してください。

サーバーは、`MyISAM` テーブルの破損を検出すると、ソースファイルの名前や行番号、テーブルにアクセスするスレッドのリストなどの追加情報をエラーログに書き込みます。たとえば、「`thread_id=1 からエラーを受け取りました。mi_dynrec.c:368`」です。これは、バグレポートに含めると役に立つ情報です。

`mysqld` が予期せず異常終了することは良い兆候ではありませんが、この場合は `Checking table...` メッセージを調査するのではなく、`mysqld` が異常終了した原因を見つけるようにしてください。

5.9.1.7 テーブルが破損した場合のテストケースの作成

次の手順は、`MyISAM` テーブルに適用されます。`InnoDB` テーブルの破損が発生した場合に実行するステップの詳細は、セクション1.6「質問またはバグをレポートする方法」を参照してください。

破損した `MyISAM` テーブルが発生した場合、または一部の `UPDATE` ステートメントの後に `mysqld` が常に失敗した場合は、次の手順を実行して問題が再現可能かどうかをテストできます:

1. `mysqldadmin shutdown` で MySQL デーモンを停止します。
2. 修復によって問題が発生する可能性が非常に低いケースから保護するために、テーブルのバックアップを作成します。
3. `myisamchk -s database/*.MYI` を使用してすべてのテーブルをチェックします。破損したテーブルを `myisamchk -r database/table.MYI` で修復します。
4. テーブルの 2 番目のバックアップを作成します。
5. より多くの領域が必要な場合は、MySQL データディレクトリから古いログファイルを削除 (または移動) します。
6. バイナリログを有効にして `mysqld` を起動します。`mysqld` がクラッシュするステートメントを見つける場合は、一般クエリログも有効にしてサーバーを起動するようにしてください。セクション5.4.3「一般クエリログ」およびセクション5.4.4「バイナリログ」を参照してください。
7. クラッシュしたテーブルを取得したら、`mysqld` サーバーを停止します。
8. バックアップをリストアします。
9. バイナリログを有効にせずに、`mysqld` サーバーを再起動します。
10. `mysqlbinlog binary-log-file | mysql` を使用してステートメントを再実行します。バイナリログは、`hostname-bin.NNNNNN` という名前で MySQL データベースディレクトリに保存されます。

11. テーブルが再度破損した場合、または前述のコマンドで `mysqld` が停止する可能性がある場合は、再現可能なバグが見つかりました。 [セクション1.6「質問またはバグをレポートする方法」](#) の手順を使用して、テーブルおよびバイナリログをバグデータベースに FTP で送信してください。サポートのお客様の場合は、MySQL カスタマサポートセンター (<https://www.mysql.com/support/>) を使用して MySQL チームにその問題を通知し、可能な限り早く修正してもらうことができます。

5.9.2 MySQL クライアントのデバッグ

統合デバッグパッケージを使用して MySQL クライアントをデバッグできるようにするには、`-DWITH_DEBUG=1` を指定して MySQL を構成します。 [セクション2.9.7「MySQL ソース構成オプション」](#) を参照してください。

クライアントを実行する前に、`MYSQL_DEBUG` 環境変数を設定します。

```
shell> MYSQL_DEBUG=d:t:O/tmp/client.trace
shell> export MYSQL_DEBUG
```

これにより、クライアントは `/tmp/client.trace` にトレースファイルを生成します。

独自のクライアントコードに問題がある場合は、動作することがわかっているクライアントを使用してサーバーに接続し、クエリーを実行してください。これを行うには、`mysql` をデバッグモードで実行します (デバッグを有効にして MySQL をコンパイルしたことを想定しています)。

```
shell> mysql --debug=d:t:O/tmp/client.trace
```

これにより、バグレポートをメール送信するときに役立つ情報が得られます。 [セクション1.6「質問またはバグをレポートする方法」](#) を参照してください。

クライアントが「正しい」ように見えるコードでクラッシュしている場合は、`mysql.h` インクルードファイルが MySQL のライブラリファイルと一致していることを確認してください。非常によくある間違いは、古い MySQL インストール環境にある古い `mysql.h` ファイルを新しい MySQL ライブラリとともに使用していることです。

5.9.3 LOCK_ORDER ツール

MySQL サーバーは、相互排他ロック、`rwlocks` (`prlocks` および `sxlocks` を含む)、条件、ファイルなど、多数の内部ロックおよびロック関連のプリミティブを使用するマルチスレッドアプリケーションです。サーバー内では、ロック関連オブジェクトのセットは、パフォーマンス向上のための新機能およびコードリファクタの実装によって変更されます。ロックプリミティブを使用するマルチスレッドアプリケーションと同様に、複数のロックが一度に保持されている場合、実行中にデッドロックが発生するリスクが常にあります。MySQL の場合、デッドロックの影響は致命的であり、サービスが完全に失われます。

MySQL 8.0.17 では、ロック取得デッドロックの検出およびランタイム実行が解放されていることの強制を有効にするために、MySQL は `LOCK_ORDER` ツールをサポートしています。これにより、ロック順序の依存性グラフをサーバー設計の一部として定義し、サーバー実行時チェックを使用して、ロック取得が非循環であり、実行パスがグラフに準拠していることを確認できます。

このセクションでは、`LOCK_ORDER` ツールの使用方法について説明しますが、基本レベルでのみ説明します。詳細は、<https://dev.mysql.com/doc/index-other.html> で入手可能な MySQL Server Doxygen ドキュメントの「ロック順序」のセクションを参照してください。

`LOCK_ORDER` ツールは、本番用ではなく、サーバーのデバッグを目的としています。

`LOCK_ORDER` ツールを使用するには、この手順に従います:

1. ソースから MySQL をビルドし、ビルドに `LOCK_ORDER` ツールが含まれるように `-DWITH_LOCK_ORDER=ON` `CMake` オプションを使用して構成します。

注記

`WITH_LOCK_ORDER` オプションを有効にすると、MySQL ビルドに `flex` プログラムが必要になります。

2. `LOCK_ORDER` ツールを有効にしてサーバーを実行するには、サーバーの起動時に `lock_order` システム変数を有効にします。 `LOCK_ORDER` 構成には、他にもいくつかのシステム変数を使用できます。

3. MySQL テストスイート操作の場合、`mysql-test-run.pl` には、テストケースの実行中に LOCK_ORDER ツールを有効にするかどうかを制御する `--lock-order` オプションがあります。

次に説明するシステム変数は、MySQL が LOCK_ORDER ツールを含むように構築されていることを前提として、LOCK_ORDER ツールの構成操作を行います。主な変数は `lock_order` で、実行時に LOCK_ORDER ツールを有効にするかどうかを示します:

- `lock_order` が無効 (デフォルト) の場合、他の LOCK_ORDER システム変数は無効になりません。
- `lock_order` が有効になっている場合、他のシステム変数によって、有効にする LOCK_ORDER 機能が構成されません。

注記

一般に、`--lock-order` オプションを指定して `mysql-test-run.pl` を実行し、`mysql-test-run.pl` で LOCK_ORDER システム変数を適切な値に設定することで、LOCK_ORDER ツールを構成することを目的としています。

すべての LOCK_ORDER システム変数は、サーバー起動時に設定する必要があります。実行時には、その値は表示されますが変更できません。

一部のシステム変数は、`lock_order_debug_loop` や `lock_order_trace_loop` などのペアで存在します。このようなペアでは、変数が関連付けられている条件が発生すると、変数は次のように区別されます:

- `_debug` 変数が有効な場合は、デバッグアサーションが発生します。
- `_trace` 変数が有効な場合は、ログにエラーが出力されます。

表 5.7 「LOCK_ORDER システム変数サマリー」

変数名	変数型	変数スコープ
<code>lock_order</code>	Boolean	グローバル
<code>lock_order_debug_loop</code>	Boolean	グローバル
<code>lock_order_debug_missing_arc</code>	Boolean	グローバル
<code>lock_order_debug_missing_key</code>	Boolean	グローバル
<code>lock_order_debug_missing_unlock</code>	Boolean	グローバル
<code>lock_order_dependencies</code>	ファイル名	グローバル
<code>lock_order_extra_dependencies</code>	ファイル名	グローバル
<code>lock_order_output_directory</code>	ディレクトリ名	グローバル
<code>lock_order_print_txt</code>	Boolean	グローバル
<code>lock_order_trace_loop</code>	Boolean	グローバル
<code>lock_order_trace_missing_arc</code>	Boolean	グローバル
<code>lock_order_trace_missing_key</code>	Boolean	グローバル
<code>lock_order_trace_missing_unlock</code>	Boolean	グローバル

- `lock_order`

コマンド行形式	<code>--lock-order[={OFF ON}]</code>
導入	8.0.17
システム変数	<code>lock_order</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean

デフォルト値	OFF
--------	-----

実行時に LOCK_ORDER ツールを有効にするかどうか。lock_order が無効 (デフォルト) の場合、他の LOCK_ORDER システム変数は無効になりません。lock_order が有効になっている場合、他のシステム変数によって、有効にする LOCK_ORDER 機能が構成されます。

lock_order が有効な場合、ロック順序グラフで宣言されていないロック問合せ順序がサーバーで検出されると、エラーが発生します。

- lock_order_debug_loop

コマンド行形式	--lock-order-debug-loop[={OFF ON}]
導入	8.0.17
システム変数	lock_order_debug_loop
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

ロック順序グラフでループとしてフラグ付けされた依存性が検出されたときに、LOCK_ORDER ツールがデバッグアサーションの失敗の原因となるかどうか。

- lock_order_debug_missing_arc

コマンド行形式	--lock-order-debug-missing-arc[={OFF ON}]
導入	8.0.17
システム変数	lock_order_debug_missing_arc
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

ロック順序グラフで宣言されていない依存性が検出されたときに、LOCK_ORDER ツールがデバッグアサーションの失敗を引き起こすかどうか。

- lock_order_debug_missing_key

コマンド行形式	--lock-order-debug-missing-key[={OFF ON}]
導入	8.0.17
システム変数	lock_order_debug_missing_key
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

パフォーマンススキーマで適切に計測されていないオブジェクトが検出されたときに、LOCK_ORDER ツールがデバッグアサーションの失敗を引き起こすかどうか。

- lock_order_debug_missing_unlock

コマンド行形式	<code>--lock-order-debug-missing-unlock[={OFF ON}]</code>
導入	8.0.17
システム変数	lock_order_debug_missing_unlock
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

LOCK_ORDER ツールが、まだ保持されている間に破棄されたロックを検出したときにデバッグアサーションの失敗を引き起こすかどうか。

- [lock_order_dependencies](#)

コマンド行形式	<code>--lock-order-dependencies=file_name</code>
導入	8.0.17
システム変数	lock_order_dependencies
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	empty string

サーバーのロック順序依存性グラフを定義する `lock_order_dependencies.txt` ファイルへのパス。

依存関係は指定できません。この場合、空の依存性グラフが使用されます。

- [lock_order_extra_dependencies](#)

コマンド行形式	<code>--lock-order-extra-dependencies=file_name</code>
導入	8.0.17
システム変数	lock_order_extra_dependencies
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	empty string

ロック順序依存性グラフの追加の依存性を含むファイルへのパス。これは、サードパーティコードの動作を説明する追加の依存関係を使用して、`lock_order_dependencies.txt` ファイルで定義されているプライマリサーバー依存性グラフを修正する場合に役立ちます。(かわりに、`lock_order_dependencies.txt` 自体を変更することをお勧めしません。)

この変数が設定されていない場合、セカンダリファイルは使用されません。

- [lock_order_output_directory](#)

コマンド行形式	<code>--lock-order-output-directory=dir_name</code>
導入	8.0.17
システム変数	lock_order_output_directory

スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ディレクトリ名
デフォルト値	empty string

LOCK_ORDER ツールがログを書き込むディレクトリ。この変数が設定されていない場合、デフォルトは現在のディレクトリです。

- lock_order_print_txt

コマンド行形式	--lock-order-print-txt[={OFF ON}]
導入	8.0.17
システム変数	lock_order_print_txt
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

LOCK_ORDER ツールがロック順グラフ分析を実行し、テキストレポートを出力するかどうか。このレポートには、検出されたロック取得サイクルが含まれます。

- lock_order_trace_loop

コマンド行形式	--lock-order-trace-loop[={OFF ON}]
導入	8.0.17
システム変数	lock_order_trace_loop
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

ロック順序グラフでループとしてフラグ付けされた依存性が検出されたときに、LOCK_ORDER ツールがトレースをログファイルに出力するかどうか。

- lock_order_trace_missing_arc

コマンド行形式	--lock-order-trace-missing-arc[={OFF ON}]
導入	8.0.17
システム変数	lock_order_trace_missing_arc
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

ロック順序グラフで宣言されていない依存性が検出されたときに、LOCK_ORDER ツールがトレースをログファイルに出力するかどうか。

- [lock_order_trace_missing_key](#)

コマンド行形式	<code>--lock-order-trace-missing-key[={OFF ON}]</code>
導入	8.0.17
システム変数	lock_order_trace_missing_key
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

パフォーマンススキーマで適切に計測されていないオブジェクトが検出されたときに、LOCK_ORDER ツールがトレースをログファイルに出力するかどうか。

- [lock_order_trace_missing_unlock](#)

コマンド行形式	<code>--lock-order-trace-missing-unlock[={OFF ON}]</code>
導入	8.0.17
システム変数	lock_order_trace_missing_unlock
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

まだ保持されている間に破棄されたロックが検出された場合に、LOCK_ORDER ツールがトレースをログファイルに出力するかどうか。

5.9.4 DBUG パッケージ

MySQL サーバーおよびほとんどの MySQL クライアントは、もともと Fred Fish によって作成された DBUG パッケージとともにコンパイルされます。MySQL をデバッグ用に構成した場合は、このパッケージによって、プログラムが実行している内容のトレースファイルを取得できるようになります。[セクション5.9.1.2「トレースファイルの作成」](#)を参照してください。

このセクションでは、デバッグサポート付きでビルドされた MySQL プログラムのコマンド行のデバッグオプションに指定できる引数値をまとめています。

DBUG パッケージは、`--debug[=debug_options]` または `-# [debug_options]` オプションを指定してプログラムを起動することによって使用できます。`--debug` または `-#` オプションを指定して、`debug_options` 値を指定しない場合、ほとんどの MySQL プログラムではデフォルト値が使用されます。サーバーのデフォルトは、Unix の場合は `d:t:i:o,/tmp/mysqld.trace`、Windows の場合は `d:t:i:O,\mysqld.trace` です。このデフォルトには次のような効果があります。

- `d`: すべてのデバッグマクロの出力を有効にします
- `t`: 関数の呼び出しおよび終了をトレースします
- `i`: 出力行に PID を追加します
- `o,/tmp/mysqld.trace`、`O,\mysqld.trace`: デバッグ出力ファイルを設定します

ほとんどのクライアントプログラムでは、プラットフォームにかかわらず、デフォルトの `debug_options` 値である `d:t:o,/tmp/program_name.trace` が使用されます。

シェルのコマンド行で指定されることがある、デバッグ制御文字列のいくつかの例を次に示します。

```
--debug=d:t
--debug=d:f,main,subr1:F:L:t,20
--debug=d,input,output,files:n
--debug=d:ti:O,\mysql.trace
```

mysqld の場合は、`debug` システム変数を設定することによって、DBUG 設定を実行時に変更することもできます。この変数にはグローバル値とセッション値があります。

```
mysql> SET GLOBAL debug = 'debug_options';
mysql> SET SESSION debug = 'debug_options';
```

グローバル `debug` 値を変更するには、グローバルシステム変数を設定するのに十分な権限が必要です。セッションの `debug` 値を変更するには、制限付きセッションシステム変数を設定するのに十分な権限が必要です。[セクション 5.1.9.1 「システム変数権限」](#) を参照してください。

`debug_options` 値は、コロンで区切られた一連のフィールドです。

```
field_1:field_2:...:field_N
```

値内の各フィールドは、必須フラグ文字で構成され、オプションで + または - 文字が前に付き、オプションで修飾子のカンマ区切りリストが続きます:

```
[+]-flag[,modifier,modifier,...,modifier]
```

次の表は、許可されるフラグ文字を示しています。認識されないフラグ文字は暗黙のうちに無視されます。

フラグ	説明
d	DBUG_XXX マクロからの現在の状態に関する出力を有効にします。キーワードのリストがあとに続くことがあり、そのキーワードを使用する DBUG マクロの出力のみが有効になります。キーワードのリストが空の場合は、すべてのマクロの出力が有効になります。 MySQL では、一般的に有効にされるデバッグマクロのキーワードは、 <code>enter</code> 、 <code>exit</code> 、 <code>error</code> 、 <code>warning</code> 、 <code>info</code> 、および <code>loop</code> です。
D	各デバッガの出力行のあとに待機します。引数は 0.1 秒単位の待機時間であり、マシンの能力の影響を受けません。たとえば、 <code>D,20</code> は 2 秒の待機を指定します。
f	デバッグ、トレース、およびプロファイリングの対象を指定された関数のリストに制限します。空のリストの場合はすべての関数が有効になります。適切な <code>d</code> フラグまたは <code>t</code> フラグを指定する必要があり、それらのフラグが有効な場合にのみ、このフラグはそれらのフラグのアクションを制限します。
F	デバッグ出力またはトレース出力の各行にソースファイル名を示します。
i	デバッグ出力またはトレース出力の各行に PID またはスレッド ID でプロセスを示します。
L	デバッグ出力またはトレース出力の各行にソースファイルの行番号を示します。
n	デバッグ出力またはトレース出力の各行に現在の関数のネストの深さを出力します。
N	デバッグ出力の各行に番号を付けます。
o	デバッガの出力ストリームを指定されたファイルにリダイレクトします。デフォルトの出力先は <code>stderr</code> です。
O	<code>o</code> と似ていますが、ファイルは書き込みごとに実際にはフラッシュされます。必要な場合、ファイルが書き込みごとに閉じられてふたたび開きます。

フラグ	説明
p	デバッグアクションを指定されたプロセスに限定します。デバッグアクションが実行されるためには、プロセスが <code>DEBUG_PROCESS</code> マクロで識別され、リスト内のプロセスと一致する必要があります。
P	デバッグ出力またはトレース出力の各行に現在のプロセス名を出力します。
r	新しい状態をプッシュするときに、前の状態の関数のネストレベルを継承しません。出力を左マージンから開始する場合に便利です。
S	<code>_sanity()</code> から 0 以外が返されるまで、デバッグされる各関数で関数 <code>_sanity(_file_,_line_)</code> を実行します。
t	関数の呼び出し/終了のトレース行を有効にします。最大のトレースレベルを示す数値を指定するリスト (修飾子が 1 つだけ含まれています) があとに続く場合があり、それを超えるとデバッグマクロまたはトレースマクロの出力は行われません。デフォルトはコンパイル時のオプションです。

フラグの前の + 文字や - 文字、およびフラグの後ろに続く修飾子のリストは、`d`、`f` などのフラグ文字に対して使用して、該当するすべての修飾子または一部の修飾子に対してデバッグ操作を有効にできます。

- フラグの前に + または - がいない場合、フラグ値は指定された修飾子リストのとおり設定されます。
- フラグの前に + または - がある場合は、リスト内の修飾子が現在の修飾子リストに対して追加または削除されます。

次の例は、`d` フラグでのこの動作を示しています。`d` のリストが空の場合は、すべてのデバッグマクロの出力が有効になります。リストが空でない場合は、リスト内のマクロキーワードの出力のみが有効になります。

次のステートメントでは、指定されたとおりに `d` 値が修飾子リストに設定されます。

```
mysql> SET debug = 'd';
mysql> SELECT @@debug;
+-----+
| @@debug |
+-----+
| d       |
+-----+
mysql> SET debug = 'd,error,warning';
mysql> SELECT @@debug;
+-----+
| @@debug      |
+-----+
| d,error,warning |
+-----+
```

フラグの前の + または - は、現在の `d` 値に対して追加または削除を行います。

```
mysql> SET debug = '+d,loop';
mysql> SELECT @@debug;
+-----+
| @@debug      |
+-----+
| d,error,warning,loop |
+-----+

mysql> SET debug = '-d,error,loop';
mysql> SELECT @@debug;
+-----+
| @@debug      |
+-----+
| d,warning    |
+-----+
```

「すべてのマクロが有効な状態」に対して追加した場合は、何も変更されません。

```
mysql> SET debug = 'd';
mysql> SELECT @@debug;
+-----+
| @@debug |
+-----+
| d       |
+-----+

mysql> SET debug = '+d,loop';
mysql> SELECT @@debug;
+-----+
| @@debug |
+-----+
| d       |
+-----+
```

有効なすべてのマクロを無効にすると、**d** フラグは完全に無効になります。

```
mysql> SET debug = 'd,error,loop';
mysql> SELECT @@debug;
+-----+
| @@debug |
+-----+
| d,error,loop |
+-----+

mysql> SET debug = '-d,error,loop';
mysql> SELECT @@debug;
+-----+
| @@debug |
+-----+
|         |
+-----+
```


第 6 章 セキュリティー

目次

6.1 一般的なセキュリティの問題	1032
6.1.1 セキュリティーガイドライン	1032
6.1.2 パスワードをセキュアな状態にする	1034
6.1.3 攻撃者に対する MySQL のセキュアな状態の維持	1036
6.1.4 セキュリティー関連の mysqld オプションおよび変数	1038
6.1.5 MySQL を通常ユーザーとして実行する方法	1038
6.1.6 LOAD DATA LOCAL のセキュリティ上の考慮事項	1039
6.1.7 クライアントプログラミングのセキュリティガイドライン	1042
6.2 アクセス制御とアカウント管理	1043
6.2.1 アカウントのユーザー名とパスワード	1044
6.2.2 MySQL で提供される権限	1045
6.2.3 付与テーブル	1062
6.2.4 アカウント名の指定	1070
6.2.5 ロール名の指定	1072
6.2.6 アクセス制御、ステージ 1: 接続の検証	1073
6.2.7 アクセス制御、ステージ 2: リクエストの確認	1076
6.2.8 アカウントの追加、権限の割当ておよびアカウントの削除	1077
6.2.9 予約済アカウント	1081
6.2.10 ロールの使用	1081
6.2.11 アカウントカテゴリ	1087
6.2.12 部分取消しを使用した権限の制限	1091
6.2.13 権限変更が有効化される時期	1096
6.2.14 アカウントパスワードの割り当て	1097
6.2.15 パスワード管理	1098
6.2.16 期限切れパスワードのサーバー処理	1108
6.2.17 プラガブル認証	1110
6.2.18 プロキシユーザー	1115
6.2.19 アカウントロック	1122
6.2.20 アカウントリソース制限の設定	1122
6.2.21 MySQL への接続の問題のトラブルシューティング	1124
6.2.22 SQL ベースのアカウントアクティビティ監査	1128
6.3 暗号化された接続の使用	1129
6.3.1 暗号化接続を使用するための MySQL の構成	1130
6.3.2 暗号化された接続 TLS プロトコルおよび暗号	1136
6.3.3 SSL および RSA 証明書とキーの作成	1142
6.3.4 SSH を使用した Windows から MySQL へのリモート接続	1150
6.4 セキュリティコンポーネントおよびプラグイン	1151
6.4.1 認証プラグイン	1151
6.4.2 Connection-Control プラグイン	1215
6.4.3 パスワード検証コンポーネント	1221
6.4.4 MySQL キーリング	1231
6.4.5 MySQL Enterprise Audit	1279
6.4.6 監査メッセージコンポーネント	1346
6.4.7 MySQL Enterprise Firewall	1348
6.5 MySQL Enterprise Data Masking and De-Identification	1371
6.5.1 MySQL Enterprise Data Masking and De-Identification 要素	1373
6.5.2 MySQL Enterprise Data Masking and De-Identification のインストールまたはアンインストール	1373
6.5.3 MySQL Enterprise Data Masking and De-Identification の使用	1374
6.5.4 MySQL Enterprise Data Masking and De-Identification ユーザー定義関数リファレンス	1379
6.5.5 MySQL Enterprise Data Masking and De-Identification ユーザー定義関数の説明	1380
6.6 MySQL Enterprise Encryption	1388
6.6.1 MySQL Enterprise Encryption のインストール	1389
6.6.2 MySQL Enterprise Encryption の使用方法と例	1389

6.6.3 MySQL Enterprise Encryption ユーザー定義関数リファレンス	1391
6.6.4 MySQL Enterprise Encryption ユーザー定義関数の説明	1392
6.7 SELinux	1395
6.7.1 SELinux が有効かどうかの確認	1396
6.7.2 SELinux モードの変更	1396
6.7.3 MySQL Server SELinux ポリシー	1397
6.7.4 SELinux ファイルコンテキスト	1397
6.7.5 SELinux TCP ポートコンテキスト	1398
6.7.6 SELinux のトラブルシューティング	1400
6.8 FIPS のサポート	1401

MySQL インストール内のセキュリティについて検討するときは、可能性のあるさまざまなトピックについて考慮し、それらが MySQL サーバーおよび関連するアプリケーションのセキュリティに及ぼす影響について考慮するようにしてください。

- セキュリティーに影響する一般的な要因。これらには、適切なパスワードの選択、不要な権限をユーザーに付与しないこと、SQL インジェクションおよびデータ損失を防ぐことによるアプリケーションセキュリティの確保などが含まれます。 [セクション6.1「一般的なセキュリティの問題」](#)を参照してください。
- インストール自体のセキュリティ。インストールにおけるデータファイル、ログファイル、およびすべてのアプリケーションファイルを保護することで、許可のない人物による読み取りまたは書き込みができないようにします。詳細については、 [セクション2.10「インストール後のセットアップとテスト」](#)を参照してください。
- データベースおよびデータベース内で使用中のビューおよびストアードプログラムへのアクセス権限を付与されたユーザーおよびデータベースを含む、データベースシステム自体の内部におけるアクセス制御およびセキュリティ。詳細は、 [セクション6.2「アクセス制御とアカウント管理」](#)を参照してください。
- セキュリティー関連のプラグインによって提供される機能。 [セクション6.4「セキュリティコンポーネントおよびプラグイン」](#)を参照してください。
- MySQL およびシステムのネットワークセキュリティ。セキュリティは個々のユーザーに対する権限付与に関係しますが、MySQL が、MySQL サーバーホスト上でローカルからのみ利用できるか、限定されたほかのホストのセットについてのみ利用できるように MySQL を制限したい場合もあります。
- データベースファイル、構成、およびログファイルの十分かつ適切なバックアップを用意します。また、リカバリソリューションを用意するようにし、バックアップから情報を正しくリカバリできることをテストしてください。 [第7章「バックアップとリカバリ」](#)を参照してください。

6.1 一般的なセキュリティの問題

このセクションでは、留意すべき一般的なセキュリティの問題と、攻撃または悪用に対して MySQL インストールをさらにセキュアな状態にするために実行可能なアクションについて説明します。ユーザーアカウントのセットアップおよびデータベースアクセスの検査のために MySQL が使用するアクセス制御システムについての詳細は、 [セクション2.10「インストール後のセットアップとテスト」](#)を参照してください。

MySQL Server のセキュリティの問題について、よくある質問のうちのいくつかに対する回答は、 [セクション A.9「MySQL 8.0 FAQ: セキュリティー」](#)を参照してください。

6.1.1 セキュリティーガイドライン

インターネットに接続したコンピュータ上で MySQL を使用するすべてのユーザーは、セキュリティに関するもっとも一般的な間違いを回避するために、このセクションを読むようにしてください。

セキュリティについて検討する際、該当するすべての種類の攻撃(盗聴、改変、プレイバック、およびサービス妨害)から、(MySQL サーバーだけでなく)サーバーホスト全体を完全に保護することを考慮する必要があります。ここでは可用性およびフォールトトレランスのすべての側面について扱うことはしません。

MySQL では、ユーザーが実行を試行できるすべての接続、クエリー、およびその他の操作に対して、アクセス制御リスト(ACL)に基づくセキュリティが使用されています。また、MySQL クライアントとサーバーの間で SSL に対応した接続のサポートもあります。ここで説明されている多くの概念は MySQL に固有のものではなく、同じような一般的な考え方は、ほぼすべてのアプリケーションに該当します。

MySQL を実行するときは、次のガイドラインに従ってください。

- `mysql` システムデータベースの `user` テーブルへのアクセス権を (MySQL `root` アカウントを除く) ユーザーに付与しないでください。これはきわめて重要です。
- MySQL アクセス権限システムのしくみについて学習してください ([セクション6.2「アクセス制御とアカウント管理」](#)を参照してください)。MySQL へのアクセスを制御するには、`GRANT` および `REVOKE` ステートメントを使用します。必要以上の権限を付与しないでください。すべてのホストに権限を付与してはいけません。

チェックリスト:

- `mysql -u root` を試してください。パスワードを尋ねられずにサーバーへの接続に成功する場合、すべてのユーザーが、完全な権限を持つ MySQL `root` ユーザーとして MySQL サーバーに接続できます。`root` パスワードの設定に関する情報に特に注意して、MySQL インストール手順を見直してください。[セクション2.10.4「初期MySQLアカウントの保護」](#)を参照してください。
- `SHOW GRANTS` ステートメントを使用して、どのアカウントが何にアクセスできるかをチェックします。`REVOKE` ステートメントを使用して、不要な権限を削除します。
- 平文パスワードをデータベースに保管しないでください。コンピュータのセキュリティが損なわれた場合、侵入者はすべてのパスワードのリストを取得して使用することができます。かわりに、`SHA2()` またはその他の一方方向ハッシュ関数を使用してハッシュ値を格納します。

レインボーテーブルを使用したパスワードのリカバリを避けるために、これらの関数をプレーンテキストパスワードに使用しないようにしてください。代わりに、ソルトとして使用する何らかの文字列を選択して、`hash(hash(パスワード)+ソルト)` 値を使用してください。

- 辞書からパスワードを選択しないでください。パスワードを解読する特殊なプログラムが存在します。「xfish98」のようなパスワードでさえも、非常に悪いものです。同じ「fish」という単語を、標準 QWERTY キーボードでキー 1 個分左にずらしてタイプした「duag98」の方が、ずっと優れています。別の方法は、文の各単語の先頭文字を取ったパスワードを使用することです (たとえば、「Four score and seven years ago」からは「Fsasya」というパスワードができます)。パスワードは覚えやすく入力も簡単ですが、文を知らない人は推測が困難です。この事例で、数字を示す単語をさらに数値に置き換えて「4 score and 7 years ago」という句を作成し、「4sa7ya」というさらに推測困難なパスワードを得ることができます。
- ファイアウォールに投資します。これにより、あらゆる種類のソフトウェアの悪用のうち、少なくとも 50% から保護されます。MySQL をファイアウォールの背後または非武装地帯 (DMZ) に配置します。

チェックリスト:

- `nmap` などのツールを使用して、インターネットから自分のポートをスキャンしてみてください。MySQL はデフォルトでポート 3306 を使用します。このポートは信頼できないホストからアクセス可能であってはなりません。MySQL ポートが開いているかどうかを検査する簡単な方法として、いずれかのリモートマシンから次のコマンドを試行します。ここで、`server_host` は MySQL サーバーが実行しているホストのホスト名または IP アドレスです。

```
shell> telnet server_host 3306
```

`telnet` がハングするか、接続が拒否されれば、ポートはブロックされており、これは期待どおりの結果です。接続を取得して、何らかの文字化けた文字が得られた場合、ポートは開いているため、ポートを開いたままにしておく十分な理由が実際にある場合を除き、ファイアウォールまたはルーターで閉じるようにしてください。

- MySQL にアクセスするアプリケーションは、ユーザーから入力されるすべてのデータを信頼しないようにし、適切な防御的プログラミング技術を使用して記述するようにします。[セクション6.1.7「クライアントプログラミングのセキュリティガイドライン」](#)を参照してください。
- プレーンの (暗号化されていない) データをインターネット経由で送信しないでください。この情報は、情報を傍受する時間と能力を持ち、自身の目的のために情報を使用するすべての人物からアクセスできます。代わりに、SSL や SSH などの暗号化されたプロトコルを使用してください。MySQL は、内部 SSL 接続をサポートします。別の技術として、SSH ポートフォワーディングを使用して、暗号化された (および圧縮された) 通信用トンネルを作成する方法があります。
- `tcpdump` や `strings` などのユーティリティーの使用法について学習します。ほとんどの場合、次のようなコマンドを発行することによって、MySQL データストリームが暗号化されていない状態であるかどうかを検査できます。

```
shell> tcpdump -l -i eth0 -w - src or dst port 3306 | strings
```

これは Linux で動作するほか、ほかのシステムでもわずかな変更を行うことで動作するはずです。

警告

平文データが表示されない場合、これは情報が実際に暗号化されていることを必ずしも意味しているわけではありません。セキュリティを強化する必要がある場合、セキュリティの専門家に相談してください。

6.1.2 パスワードをセキュアな状態にする

パスワードは MySQL のいくつかのコンテキストで現れます。次のセクションでは、エンドユーザーおよび管理者がこれらのパスワードをセキュアな状態にし、公開しないようにするためのガイドラインを提供します。また、[validate_password](#) プラグインを使用して、許容可能なパスワードにポリシーを適用できます。[セクション 6.4.3 「パスワード検証コンポーネント」](#) を参照してください。

6.1.2.1 パスワードセキュリティのためのエンドユーザーガイドライン

MySQL ユーザーは、パスワードをセキュアな状態にするために次のガイドラインを使用することをお勧めします。

クライアントプログラムを実行して MySQL サーバーに接続する場合、ほかのユーザーからの検出によって公開されるような方法でパスワードを指定することはお勧めできません。クライアントプログラムを実行するときにパスワードを指定するために使用できる方法と、それぞれの方法のリスクの評価について、次の一覧で示します。簡単に言えば、もっとも安全な方法は、クライアントプログラムがパスワードを求めるプロンプトを出すようにするか、適切に保護されたオプションファイルにパスワードを指定する方法です。

- `mysql_config_editor` ユーティリティを使用すると、`.mylogin.cnf` という名前の暗号化されたログインパスワードファイルに認証資格証明を格納できます。このファイルは、MySQL Server に接続するための認証情報を取得するために、MySQL クライアントプログラムによってあとで読み取ることができます。[セクション 4.6.7 「mysql_config_editor — MySQL 構成ユーティリティ」](#) を参照してください。
- コマンドラインで `--password=password` または `-ppassword` オプションを使用します。例:

```
shell> mysql -u francis -pfrank db_name
```

警告

これは便利ですがセキュアではありません。一部のシステムでは、使用しているパスワードが、コマンド行を表示するためにほかのユーザーによって起動できる `ps` などのシステムステータスプログラムによって表示可能になります。MySQL クライアントは通常、クライアントの初期化シーケンス中にコマンド行パスワード引数をゼロで上書きします。ただし、まだ値が表示可能な短い期間があります。また、一部のシステムではこの上書きの方法には効果がなく、パスワードは `ps` から表示可能になったままになります。(SystemV Unix システムおよびおそらくほかのシステムでもこの問題の影響があります。)

ターミナルウィンドウのタイトルバーに現在のコマンドを表示するようにオペレーティング環境がセットアップされている場合、コマンドがウィンドウのコンテンツ領域からスクロールアウトされて表示されなくなっても、コマンドが実行中であるかぎりパスワードが表示されたままになります。

- パスワード値を指定せずに、コマンドラインで `--password` または `-p` オプションを使用します。この場合、クライアントプログラムはパスワードを対話的に要求します。

```
shell> mysql -u francis -p db_name
Enter password: *****
```

- * 文字はパスワードを入力したことを示しています。パスワードは入力時に表示されません。

この方法でパスワードを入力する方が、コマンド行でパスワードを指定するよりもセキュアです。これは、パスワードがほかのユーザーに表示されないためです。ただし、このパスワード入力方法は、対話的に実行するプログラムについてのみ適しています。非対話的に実行するスクリプトからクライアントを呼び出す場合、キーボードからパスワードを入力する機会はありません。一部のシステムでは、スクリプトの 1 行目 (誤って) パスワードとして読み取られて解釈されることすらあります。

- パスワードをオプションファイルに格納します。たとえば Unix の場合、ホームディレクトリの `.my.cnf` ファイルの `[client]` セクションにパスワードを一覧表示することができます。

```
[client]
password=password
```

パスワードを安全に保持するには、自分以外のすべてのユーザーからファイルにアクセス可能にしてはいけません。このようにするには、ファイルのアクセスモードを `400` または `600` に設定します。例:

```
shell> chmod 600 .my.cnf
```

パスワードを格納する特定のオプションファイルをコマンド行から指定するには、`--defaults-file=file_name` オプションを使用します。ここで `file_name` はファイルへのフルパス名です。例:

```
shell> mysql --defaults-file=/home/francis/mysql-opts
```

[セクション4.2.2.2「オプションファイルの使用」](#)には、オプションファイルについてさらに詳しく記載されています。

Unix の場合、`mysql` クライアントは実行済みステートメントを履歴ファイルに書き込みます ([セクション4.5.1.3「mysql クライアントロギング」](#)を参照してください)。デフォルトでは、このファイルは `.mysql_history` という名前で、ユーザーのホームディレクトリに作成されます。パスワードは、`CREATE USER` や `ALTER USER` などの SQL ステートメントにプレーンテキストで記述できるため、これらのステートメントを使用すると、履歴ファイルに記録されます。このファイルを安全に保持するには、以前 `.my.cnf` ファイルについて説明したのと同じ方法である制限アクセスモードを使用します。

コマンドインタプリタで履歴が保持されている場合、コマンドが保存されるすべてのファイルには、コマンドラインで入力された MySQL パスワードが含まれます。たとえば、`bash` は `~/.bash_history` を使用します。そのようなすべてのファイルは、制限アクセスモードにするようにしてください。

6.1.2.2 パスワードセキュリティについての管理者ガイドライン

データベース管理者は、パスワードをセキュアな状態にするための次のガイドラインを使用するようにしてください。

MySQL は、ユーザーアカウントのパスワードを `mysql.user` システムテーブルに格納します。このテーブルへのアクセス権を、管理者以外のすべてのアカウントに決して付与しないでください。

アカウントパスワードは期限切れとなることがあり、ユーザーはそれらをリセットする必要があります。 [セクション6.2.15「パスワード管理」](#) および [セクション6.2.16「期限切れパスワードのサーバー処理」](#) を参照してください。

`validate_password` プラグインを使用して、許容可能なパスワードについてのポリシーを強制することができます。 [セクション6.4.3「パスワード検証コンポーネント」](#) を参照してください。

プラグインディレクトリ (`plugin_dir` システム変数の値) を変更するためのアクセス権を持つユーザー、またはプラグインディレクトリの場所を指定する `my.cnf` ファイルは、プラグインを置き換えたり、認証プラグインを含むプラグインによって提供される機能を変更したりできます。

パスワードが書き込まれる可能性があるログファイルなどのファイルを保護するようにしてください。 [セクション6.1.2.3「パスワードおよびロギング」](#) を参照してください。

6.1.2.3 パスワードおよびロギング

パスワードは、`CREATE USER`、`GRANT`、`SET PASSWORD` などの SQL ステートメントでプレーンテキストとして記述できます。このようなステートメントが書き込まれたとおりに MySQL サーバーによってログに記録された場合、それらの中のパスワードは、ログへのアクセス権を持つすべてのユーザーに表示されます。

ステートメントロギングでは、次のステートメントのクリアテキストとしてパスワードを書き込まないようにします:

```
CREATE USER ... IDENTIFIED BY ...
ALTER USER ... IDENTIFIED BY ...
SET PASSWORD ...
START SLAVE ... PASSWORD = ...
START REPLICA ... PASSWORD = ...
```



```
CREATE SERVER ... OPTIONS(... PASSWORD ...)  
ALTER SERVER ... OPTIONS(... PASSWORD ...)
```

これらのステートメントのパスワードは、一般クエリーログ、スロークエリーログ、およびバイナリログに書き込まれたステートメントテキストに文字どおりに表示されないように書き換えられます。ほかのステートメントについては書き換えが適用されません。特に、リテラルパスワードを参照する `mysql.user` システムテーブルの `INSERT` ステートメントまたは `UPDATE` ステートメントはそのままログに記録されるため、このようなステートメントは避けてください。(付与テーブルを直接変更することはお勧めしません。)

一般クエリーログの場合、パスワードの書き換えは、`--log-raw` オプションを使用してサーバーを起動することによって抑制することができます。セキュリティ上の理由から、このオプションは本番での使用にはお薦めできません。診断のために、サーバーで受信されたステートメントの正確なテキストを確認すると役立つ場合があります。

デフォルトでは、監査ログプラグインによって生成される監査ログファイルの内容は暗号化されず、SQL ステートメントのテキストなどの機密情報が含まれることがあります。セキュリティ上の理由から、監査ログファイルは、MySQL サーバーおよびログを表示する正当な理由を持つユーザーのみがアクセスできるディレクトリに書き込む必要があります。[セクション6.4.5.3「MySQL Enterprise Audit のセキュリティに関する考慮事項」](#)を参照してください。

クエリーリライトプラグインがインストールされている場合は、サーバーが受信したステートメントがリライトされることがあります ([Query Rewrite Plugins](#) を参照)。この場合、`--log-raw` オプションは次のようにステートメントロギングに影響します:

- `--log-raw` がない場合、サーバーはクエリーリライトプラグインによって返されたステートメントをログに記録します。これは、受け取ったステートメントとは異なる場合があります。
- `--log-raw` では、サーバーは元のステートメントを受信したとおりにログに記録します。

パスワードの書き換えの影響は、解析できないステートメント(構文エラーなど)はパスワードがないことがわかっていないため、一般クエリーログに書き込まれないことです。エラーが発生したステートメントを含むすべてのステートメントのロギングが必要なユースケースでは、`--log-raw` オプションを使用する必要があります。これにより、パスワードのリライトもバイパスされることに注意してください。

パスワードの書き換えは、プレーンテキストパスワードが必要な場合にのみ行われます。パスワードハッシュ値を想定する構文を持つステートメントの場合、書き換えは行われません。このような構文に対してプレーンテキストパスワードが誤って指定された場合、パスワードはリライトなしで指定されたとおりにログに記録されます。

保証されていない公開からログファイルを保護するには、サーバーおよびデータベース管理者へのアクセスを制限するディレクトリ内でログファイルを見つけます。サーバーが `mysql` データベース内のテーブルにログを記録する場合は、それらのテーブルへのアクセス権をデータベース管理者にのみ付与します。

レプリカは、レプリケーションソースサーバーのパスワードを接続メタデータリポジトリ(デフォルトでは、`slave_master_info` という名前の `mysql` データベースのテーブル)に格納します。接続メタデータリポジトリのデータディレクトリでのファイルの使用は非推奨になりましたが、引き続き可能です([セクション17.2.4「リレーログおよびレプリケーションメタデータリポジトリ」](#)を参照)。データベース管理者のみが接続メタデータリポジトリにアクセスできることを確認します。接続メタデータリポジトリにパスワードを格納するかわりに、`START REPLICASlave` ステートメントまたは `START GROUP_REPLICATION` ステートメントを使用して、ソースに接続するための資格証明を指定することもできます。

制限付きアクセスモードを使用して、パスワードを含むログテーブルまたはログファイルを含むデータベースバックアップを保護します。

6.1.3 攻撃者に対する MySQL のセキュアな状態の維持

MySQL サーバーに接続するときは、パスワードを使用するようにしてください。パスワードはクリアテキストとして接続経由で送信されません。

ほかのすべての情報はテキストとして送信され、接続を観察できるすべてのユーザーによって読み取ることができません。クライアントとサーバーの間の接続が、信頼できないネットワークを介して行われ、そのことに不安がある場合、圧縮されたプロトコルを使用して、トラフィックの解読をさらに困難にすることができます。また、MySQL の内部 SSL サポートを使用して、接続をさらにセキュアな状態にすることもできます。[セクション6.3「暗号化された接続の使用」](#)を参照してください。または SSH を使用して、MySQL サーバーと MySQL クライアントの間で暗号化

された TCP/IP 接続を実現します。オープンソース SSH クライアントは <http://www.openssh.org/> から見つけることができ、オープンソースと商用の SSH クライアントの比較は http://en.wikipedia.org/wiki/Comparison_of_SSH_clients にあります。

MySQL システムをセキュアな状態にするには、次の推奨事項についてよく検討するようにしてください。

- すべての MySQL アカウントがパスワードを持つことを要求します。クライアントプログラムは、それを実行中の人物の ID を必ずしも認識していません。クライアント/サーバーアプリケーションでは、ユーザーがクライアントプログラムに任意のユーザー名を指定できることが一般的です。たとえば、`other_user` にパスワードがない場合、`mysql` プログラムを `mysql -u other_user db_name` として呼び出すことによって、すべてのユーザーがこのプログラムを使用してほかのユーザーとして接続することができます。すべてのアカウントにパスワードがある場合、ほかのユーザーのアカウントを使用した接続は、もっと難しくなります。

パスワードの設定方法についての説明は、[セクション6.2.14「アカウントパスワードの割り当て」](#)を参照してください。

- データベースディレクトリ内の読み取りまたは書き込み権限を持つ Unix ユーザーアカウントのみが、`mysqld` の実行に使用されるアカウントであるようにしてください。
- MySQL サーバーを Unix `root` ユーザーとして絶対に実行しないでください。これを行うと、`FILE` 権限を持つすべてのユーザーが、`root` としてサーバーにファイルを作成させることができるため (`~root/.bashrc` など)、非常に危険です。これを防ぐために、`mysqld` は `--user=root` オプションを使用して明示的に指定された場合を除き、`root` として実行することを拒否します。

`mysqld` は、権限のない普通のユーザーとしても実行できます (また、そのように実行するべきです)。`mysql` という名前の別の Unix アカウントを作成して、すべてをさらにセキュアな状態にすることができます。このアカウントは、MySQL の管理にのみ使用してください。`mysqld` を別の Unix ユーザーとして開始するには、サーバーオプションを指定した `my.cnf` オプションファイルの `[mysqld]` グループ内のユーザー名を指定する `user` オプションを追加します。例:

```
[mysqld]
user=mysqld
```

これにより、サーバーを手動で起動した場合も、`mysqld_safe` または `mysql.server` を使用して起動した場合も、指定のユーザーでサーバーが起動します。詳細は、[セクション6.1.5「MySQL を通常ユーザーとして実行する方法」](#)を参照してください。

`root` 以外の Unix ユーザーとして `mysqld` を実行しても、`user` テーブル内の `root` ユーザー名を変更する必要があるということを意味するわけではありません。MySQL アカウントのユーザー名は、Unix アカウントのユーザー名とは何の関係もありません。

- 管理者以外のユーザーに `FILE` 権限を付与しないでください。この権限を持つすべてのユーザーは、`mysqld` デモンの権限で、ファイルシステムのあらゆる場所のファイルに書き込むことができます。これは、権限テーブルを実装するファイルを格納するサーバーのデータディレクトリを含みます。`FILE` 権限の操作をもう少し安全にするために、`SELECT ... INTO OUTFILE` で生成されたファイルは既存のファイルを上書きせず、すべてのユーザーによって書き込み可能になります。

`FILE` 権限は、すべてのユーザーが読み取り可能であるか、サーバーを実行している Unix ユーザーがアクセスできる、すべてのファイルを読み取る場合にも使用できます。この権限を使用して、すべてのファイルをデータベーステーブルに読み取ることができます。これは不正使用される可能性があり、たとえば `LOAD DATA` を使用して `/etc/passwd` をテーブルにロードし、次に `SELECT` を使用してこれを表示することができます。

ファイルを読み取りおよび書き込みできる場所を制限するには、`secure_file_priv` システムを特定のディレクトリに設定します。[セクション5.1.8「サーバーシステム変数」](#)を参照してください。

- バイナリログファイルとリレーログファイルを暗号化します。暗号化を使用すると、これらのファイルおよびそのファイルに含まれる機密データを、外部の攻撃者による誤用や、ファイルが格納されているオペレーティングシステムのユーザーによる不正な表示から保護できます。MySQL サーバーで暗号化を有効にするには、`binlog_encryption` システム変数を `ON` に設定します。詳細は、[セクション17.3.2「バイナリログファイルとリレーログファイルの暗号化」](#)を参照してください。
- 管理者以外のユーザーに `PROCESS` または `SUPER` 権限を付与しないでください。`mysqldadmin processlist` および `SHOW PROCESSLIST` の出力には、現在実行されているステートメントのテキストが表示されるため、サーバープ

ロケストの表示を許可されているユーザーは、ほかのユーザーによって発行されたステートメントを表示できません。

mysqld は、[CONNECTION_ADMIN](#) または [SUPER](#) 権限を持つユーザーに対して追加の接続を予約するため、MySQL [root](#) ユーザーは、通常の接続がすべて使用されている場合でも、ログインしてサーバーアクティビティを確認できます。

[SUPER](#) 権限は、クライアント接続を終了したり、システム変数の値を変更することによってサーバー操作を変更したり、レプリケーションサーバーを制御したりするために使用することができます。

- テーブルへのシンボリックリンクを許可しないでください。(この機能は `--skip-symbolic-links` オプションで無効にできます。) このことは、[mysqld](#) を [root](#) として実行する場合に特に重要です。これは、サーバーのデータディレクトリへの書き込みアクセス権限があるすべてのユーザーは、システムのすべてのファイルを削除できることになるためです。[セクション8.12.2.2「Unix 上の MyISAM へのシンボリックリンクの使用」](#)を参照してください。
- ストアドプログラムおよびビューは、[セクション25.6「ストアドオブジェクトのアクセス制御」](#)に記載されているセキュリティガイドラインを使用して記述するようにしてください。
- DNS を信頼していない場合、付与テーブル内でホスト名の代わりに IP アドレスを使用するようにしてください。いずれの場合も、ワイルドカードを含むホスト名の値を使用して付与テーブルエントリを作成することについては、十分に注意するようにしてください。
- 単一アカウントに対して許可される接続数を制限するには、[mysqld](#) の `max_user_connections` 変数を設定することによってこれを実行できます。[CREATE USER](#) および [ALTER USER](#) ステートメントでは、アカウントに許可されるサーバー使用の範囲を制限するためのリソース制御オプションもサポートされています。[セクション13.7.1.3「CREATE USER ステートメント」](#) および [セクション13.7.1.1「ALTER USER ステートメント」](#)を参照してください。
- プラグインディレクトリがサーバーによって書き込み可能な場合、ユーザーは `SELECT ... INTO DUMPFILE` を使用して、ディレクトリ内のファイルに実行可能コードを書き込むことができます。これを回避するには、`plugin_dir` をサーバーに対して読取り専用にするか、`SELECT` 書き込みを安全に行うことができるディレクトリに `secure_file_priv` を設定します。

6.1.4 セキュリティー関連の mysqld オプションおよび変数

次の表は、セキュリティに影響する [mysqld](#) オプションおよびシステム変数を示します。これらの個別の説明は、[セクション5.1.7「サーバーコマンドオプション」](#) および [セクション5.1.8「サーバーシステム変数」](#)を参照してください。

表 6.1 「セキュリティオプションおよび変数サマリー」

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
allow-suspicious-udfs	はい	はい				
automatic_sp_privileges	はい	はい	はい		グローバル	はい
chroot	はい	はい				
local_infile	はい	はい	はい		グローバル	はい
safe-user-create	はい	はい				
secure_file_priv	はい	はい	はい		グローバル	いいえ
skip-grant-tables	はい	はい				
skip_name_resolve	はい	はい	はい		グローバル	いいえ
skip_networking	はい	はい	はい		グローバル	いいえ
skip_show_database	はい	はい	はい		グローバル	いいえ

6.1.5 MySQL を通常ユーザーとして実行する方法

Windows 上では、通常のユーザーアカウントを使用して Windows サービスとしてサーバーを実行できます。

Linux では、MySQL リポジトリまたは RPM パッケージを使用してインストールを実行する場合、ローカルの `mysql` オペレーティングシステムユーザーが MySQL サーバー `mysqld` を起動する必要があります。別のオペレーティングシステムユーザーによる開始は、MySQL リポジトリの一部として含まれる `init` スクリプトではサポートされていません。

Unix (`tar.gz` パッケージを使用して実行されるインストールの場合は Linux) では、MySQL サーバー `mysqld` を起動して任意のユーザーが実行できます。しかし、セキュリティ上の理由から、Unix `root` ユーザーとしてサーバーを実行することは避けてください。`mysqld` を変更し、権限のない普通の Unix ユーザー `user_name` として実行するには、次のことを行う必要があります。

1. サーバーが稼働していれば、サーバーを停止します (`mysqldadmin shutdown` を使用します)。
2. データベースディレクトリとファイルを変更して、`user_name` がそのファイルの読み取りおよび書き込みを行う権限を与えます (この操作は Unix `root` ユーザーとして行う必要があります)。

```
shell> chown -R user_name /path/to/mysql/datadir
```

これを行わないと、サーバーは `user_name` として実行されているときにデータベースまたはテーブルにアクセスできません。

MySQL データディレクトリ内のディレクトリまたはファイルがシンボリックリンクの場合、`chown -R` がシンボリックリンク先を参照しないことがあります。そうでない場合は、これらのリンクをたどり、それらが指し示しているディレクトリおよびファイルを変更する必要があります。

3. `user_name` というユーザーでサーバーを起動します。別の方法として、Unix `root` ユーザーとして `mysqld` を起動し、`--user=user_name` オプションを使用することもできます。`mysqld` が起動し、接続を受け入れる前に Unix ユーザー `user_name` として実行されるように切り替えます。
4. システム起動時に指定されたユーザーとしてサーバーを起動するには、`/etc/my.cnf` オプションファイルまたはサーバーのデータディレクトリに格納されている `my.cnf` オプションファイルの `[mysqld]` グループに、`user` オプションを追加することによってユーザー名を指定します。例:

```
[mysqld]  
user=user_name
```

Unix マシン自体が保護されていない場合は、権限付与テーブルで MySQL `root` アカウントにパスワードを割り当てる必要があります。これをしないと、そのマシンのログインアカウントを持つすべてのユーザーが、`--user=root` オプションを使用して `mysql` クライアントを実行でき、あらゆる操作を行うことができます。(すべての場合に MySQL アカウントにパスワードを割り当てることはよい考えですが、ほかのログインアカウントがサーバーホスト上に存在する場合は特に重要です。) [セクション2.10.4「初期 MySQL アカウントの保護」](#)を参照してください。

6.1.6 LOAD DATA LOCAL のセキュリティ上の考慮事項

`LOAD DATA` ステートメントは、データファイルをテーブルにロードします。このステートメントは、サーバーホスト上のファイル、または `LOCAL` キーワードが指定されている場合はクライアントホスト上のファイルをロードできます。

`LOCAL` バージョンの `LOAD DATA` には、セキュリティ上の潜在的な問題があります:

- `LOAD DATA LOCAL` は SQL ステートメントであるため、解析はサーバー側で行われ、クライアントホストからサーバーホストへのファイルの転送は MySQL サーバーによって開始され、クライアントにステートメントで指定されたファイルが通知されます。理論上、バッチが適用されたサーバーは、ステートメントで指定されたファイルではなく、選択したサーバーのファイルを転送するようにクライアントプログラムに指示できます。そのようなサーバーは、クライアントユーザーが読み取りアクセス権を持つクライアントホスト上のすべてのファイルにアクセスできます。(バッチが適用されたサーバーは、実際には `LOAD DATA LOCAL` だけでなく任意のステートメントへのファイル転送リクエストで応答する可能性があるため、より基本的な問題は、クライアントが信頼できないサーバーに接続しないことです。)
- クライアントが Web サーバーから接続している Web 環境では、ユーザーは `LOAD DATA LOCAL` を使用して、Web サーバープロセスが読み取りアクセス権を持つファイルを読み取ることができます (ユーザーが SQL サー

バーに対して任意のステートメントを実行できると想定しています)。この環境では、MySQL サーバーに関するクライアントは実際には Web サーバーであり、Web サーバーに接続するユーザーによって実行されるリモートプログラムではありません。

信頼できないサーバーへの接続を回避するために、クライアントはセキュアな接続を確立し、`--ssl-mode=VERIFY_IDENTITY` オプションと適切な CA 証明書を使用して接続することでサーバー識別情報を確認できません。

LOAD DATA の問題を回避するには、クライアント側で適切な予防措置が講じられていないかぎり、クライアントは LOCAL を使用しないようにしてください。

ローカルデータロードを制御するために、MySQL ではこの機能を有効または無効にできます。また、MySQL 8.0.21 の時点では、MySQL を使用すると、クライアントはローカルデータロード操作を指定されたディレクトリにあるファイルに制限できます。

- [ローカルデータロード機能の有効化または無効化](#)
- [ローカルデータロードに許可されるファイルの制限](#)

ローカルデータロード機能の有効化または無効化

管理者およびアプリケーションは、ローカルデータロードを許可するかどうかを次のように構成できます：

- サーバー側：
 - `local_infile` システム変数は、サーバー側の LOCAL 機能を制御します。`local_infile` の設定に応じて、サーバーはローカルデータロードをリクエストするクライアントによるローカルデータロードを拒否または許可します。
 - デフォルトでは、`local_infile` は無効です。サーバーが LOAD DATA LOCAL ステートメントを明示的に拒否または許可するには (構築時または実行時にクライアントプログラムおよびライブラリがどのように構成されているかに関係なく)、`local_infile` を無効または有効にして `mysqld` を起動します。`local_infile` は実行時に設定することもできます。
- クライアント側：
 - `ENABLED_LOCAL_INFILE` CMake オプションは、MySQL クライアントライブラリのコンパイル済みのデフォルトの LOCAL 機能を制御します ([セクション 2.9.7 「MySQL ソース構成オプション」](#) を参照)。したがって、明示的な配置を行わないクライアントでは、MySQL ビルド時に指定された `ENABLED_LOCAL_INFILE` 設定に従って、LOCAL 機能が無効または有効になります。
 - デフォルトでは、MySQL バイナリディストリビューションのクライアントライブラリは、`ENABLED_LOCAL_INFILE` を無効にしてコンパイルされます。ソースから MySQL をコンパイルする場合は、明示的な配置を行わないクライアントで LOCAL 機能を無効にするか有効にするかに基づいて、`ENABLED_LOCAL_INFILE` を無効または有効にして構成します。
 - C API を使用するクライアントプログラムの場合、ローカルデータロード機能は、MySQL クライアントライブラリにコンパイルされるデフォルトで決定されます。明示的に有効または無効にするには、`mysql_options()` C API 関数を呼び出して、`MYSQL_OPT_LOCAL_INFILE` オプションを無効または有効にします。[mysql_options\(\)](#) を参照してください。
 - `mysql` クライアントの場合、ローカルデータロード機能は、MySQL クライアントライブラリにコンパイルされたデフォルトで決定されます。明示的に無効または有効にするには、`--local-infile=0` または `--local-infile=[1]` オプションを使用します。
 - `mysqlimport` クライアントの場合、デフォルトではローカルデータロードは使用されません。明示的に無効または有効にするには、`--local=0` または `--local=[1]` オプションを使用します。
 - Perl スクリプトまたはオプションファイルから `[client]` グループを読み取る他のプログラムで LOAD DATA LOCAL を使用する場合は、そのグループに `local-infile` オプション設定を追加できます。このオプションを認識しないプログラムの問題を回避するには、`loose-` 接頭辞を使用して指定します：

```
[client]
loose-local-infile=0
```


または

```
[client]
loose-local-infile=1
```

- いずれの場合も、クライアントによる **LOCAL** ロード操作を正常に使用するには、サーバーでローカルロードが許可されている必要があります。

サーバー側またはクライアント側で **LOCAL** 機能が無効になっている場合、**LOAD DATA LOCAL** ステートメントを発行しようとするクライアントは次のエラーメッセージを受け取ります:

```
ERROR 3950 (42000): Loading local data is disabled; this must be
enabled on both the client and server side
```

ローカルデータロードに許可されるファイルの制限

MySQL 8.0.21 の時点で、MySQL クライアントライブラリを使用すると、クライアントアプリケーションはローカルデータロード操作を指定されたディレクトリにあるファイルに制限できます。特定の MySQL クライアントプログラムは、この機能を利用します。

C API を使用するクライアントプログラムは、`mysql_options()` C API 関数の `MYSQL_OPT_LOCAL_INFILE` および `MYSQL_OPT_LOAD_DATA_LOCAL_DIR` オプションを使用して、データロードを許可するファイルを制御できます (`mysql_options()` を参照)。

`MYSQL_OPT_LOAD_DATA_LOCAL_DIR` の効果は、**LOCAL** データロードが有効か無効かによって異なります:

- **LOCAL** データロードが MySQL クライアントライブラリでデフォルトで有効になっているか、`MYSQL_OPT_LOCAL_INFILE` を明示的に有効にしても、`MYSQL_OPT_LOAD_DATA_LOCAL_DIR` オプションは無効です。
- **LOCAL** データロードが無効になっている場合は、デフォルトで MySQL クライアントライブラリで無効になっているか、`MYSQL_OPT_LOCAL_INFILE` を明示的に無効にすることによって、`MYSQL_OPT_LOAD_DATA_LOCAL_DIR` オプションを使用して、ローカルにロードされるファイルに許可されるディレクトリを指定できます。この場合、**LOCAL** データのロードは許可されますが、指定されたディレクトリにあるファイルに制限されます。`MYSQL_OPT_LOAD_DATA_LOCAL_DIR` 値の解釈は次のとおりです:
 - 値が `NULL` ポインタ (デフォルト) の場合、ディレクトリ名は指定されず、**LOCAL** データのロードにファイルは許可されません。
 - 値がディレクトリパス名の場合、**LOCAL** データのロードは許可されますが、指定されたディレクトリにあるファイルに制限されます。ディレクトリパス名とロードされるファイルのパス名の比較では、基礎となるファイルシステムの大/小文字の区別に関係なく、大/小文字が区別されます。

MySQL クライアントプログラムでは、前述の `mysql_options()` オプションを次のように使用します:

- `mysql` クライアントには、ディレクトリパスまたは空の文字列を取る `--load-data-local-dir` オプションがあります。`mysql` は、オプション値を使用して `MYSQL_OPT_LOAD_DATA_LOCAL_DIR` オプションを設定します (値を `NULL` ポインタに設定する空の文字列を使用)。`--load-data-local-dir` の効果は、**LOCAL** データロードが有効かどうかによって異なります:
 - **LOCAL** データロードが MySQL クライアントライブラリでデフォルトで有効になっている場合、または `--local-infile[=1]` を指定して有効になっている場合、`--load-data-local-dir` オプションは無視されます。
 - **LOCAL** のデータロードが無効になっている場合は、MySQL クライアントライブラリでデフォルトで無効になっているか、`--local-infile=0` を指定すると、`--load-data-local-dir` オプションが適用されます。
- `--load-data-local-dir` が適用される場合、オプション値は、ローカルデータファイルを配置する必要があるディレクトリを指定します。ディレクトリパス名とロードされるファイルのパス名の比較では、基礎となるファイルシステムの大/小文字の区別に関係なく、大/小文字が区別されます。オプション値が空の文字列の場合、ディレクトリ名は指定されず、ローカルデータのロードにファイルは許可されません。
- `mysqlimport` は、ファイルを含むディレクトリが許可されたローカルロードディレクトリになるように、処理するファイルごとに `MYSQL_OPT_LOAD_DATA_LOCAL_DIR` を設定します。

- **LOAD DATA** ステートメントに対応するデータロード操作の場合、`mysqlbinlog` はバイナリログイベントからファイルを抽出し、一時的なファイルとしてローカルファイルシステムに書き込み、**LOAD DATA LOCAL** ステートメントを書き込んでファイルをロードします。デフォルトでは、`mysqlbinlog` はこれらの一時ファイルをオペレーティングシステム固有のディレクトリに書き込みます。`--local-load` オプションを使用すると、`mysqlbinlog` がローカル一時ファイルを準備するディレクトリを明示的に指定できます。

他のプロセスはデフォルトのシステム固有のディレクトリにファイルを書き込むことができるため、`mysqlbinlog` に `--local-load` オプションを指定してデータファイルに別のディレクトリを指定し、`mysqlbinlog` からの出力を処理するときに `mysql` に `--load-data-local-dir` オプションを指定して同じディレクトリを指定することをお勧めします。

6.1.7 クライアントプログラミングのセキュリティーガイドライン

MySQL にアクセスするアプリケーションは、ユーザーによって入力されるあらゆるデータを信頼しないようにしてください。ユーザーは Web フォーム、URL、または構築されたあらゆるアプリケーションに特殊文字またはエスケープ文字のシーケンスを入力することによってコードを欺くことを試すことができます。ユーザーが `;` **DROP DATABASE mysql;** のような入力を行なっても、アプリケーションがセキュアな状態に保たれるようにしてください。これは極端な例ですが、同様の技術を使用するハッカーに備えていない場合、結果として大規模なセキュリティーリークおよびデータ損失が発生することがあります。

よくある過ちは、文字列データ値のみ保護することです。数値データも忘れずに検査してください。ユーザーが `234` という値を入力したとき、アプリケーションが `SELECT * FROM table WHERE ID=234` のようなクエリを生成する場合、ユーザーは `234 OR 1=1` という値を入力して、`SELECT * FROM table WHERE ID=234 OR 1=1` というクエリをアプリケーションに生成させることができます。その結果、サーバーはテーブル内のすべての行を取得します。これはすべての行を公開し、サーバーに過剰な負荷がかかります。この種類の攻撃から保護するためのもっとも簡単な方法は、数値定数を囲む単一引用符を使用して、`SELECT * FROM table WHERE ID='234'` とする方法です。ユーザーが余分の情報を入力すると、その情報はすべて文字列の一部となります。数値コンテキストでは、MySQL は自動的にこの文字列を数値に変換し、あとに続く数値以外のすべての文字を取り除きます。

データベースに格納されているデータが公開されているもののみであれば、保護は不要だと思われることもあります。これは正しくありません。データベース内のどの行も表示が許可されている場合であっても、サービス妨害攻撃(前のパラグラフの技術に基づいた、サーバーにリソースを浪費させるものなど)から保護する必要があります。そうしない場合、サーバーは正当なユーザーに対して応答不能になります。

チェックリスト:

- 厳密な SQL モードを有効にして、サーバーが受け入れるデータ値の制限を厳しくするようサーバーに指示します。[セクション5.1.11「サーバー SQL モード」](#)を参照してください。
- すべての Web フォームに一重引用符および二重引用符 ('および") を入力してみてください。何らかの種類の MySQL エラーが出る場合、すぐに問題を調査してください。
- `%22 (%)`、`%23 (#)` および `%27 (')` を追加して、動的 URL の変更を試みます。
- 前の例で示した文字を使用して、動的 URL のデータ型を数値型から文字型に変更してみます。これらの攻撃や類似の攻撃に対してアプリケーションが安全であるようにしてください。
- 数値フィールドに、数値でなく文字、スペース、および特殊記号を入力してみます。アプリケーションはこれらを MySQL に渡す前にこれらを削除するか、またはエラーを生成します。検査済みでない値を MySQL に渡すことは非常に危険です。
- データを MySQL に渡す前にデータのサイズを検査してください。
- アプリケーションがデータベースに接続するときは、管理目的で使用するものとは異なるユーザー名を使用するようにしてください。アプリケーションに不要なアクセス権限を付与しないでください。

多くのアプリケーションプログラミングインタフェースには、データ値の特殊文字をエスケープする手段が備わっています。適切に使用すれば、これにより、意図とは異なる効果を持つステートメントをアプリケーションに生成させる値を、アプリケーションユーザーが入力できないようにすることができます。

- MySQL C API: `mysql_real_escape_string_quote()` API コールを使用します。
- MySQL++: クエリーストリームに対して `escape` および `quote` 修飾子を使用してください。

- PHP: `mysqli` または `pdo_mysql` 拡張子を使用し、古い `ext/mysql` 拡張子を使用しないでください。推奨される API は、プレースホルダを持つプリペアドステートメントのほかに、改善された MySQL 認証プロトコルおよびパスワードをサポートします。 [MySQL and PHP](#) も参照してください。

古い `ext/mysql` 拡張機能を使用する必要がある場合、エスケープには `mysql_escape_string()` または `addslashes()` ではなく `mysql_real_escape_string_quote()` 関数を使用します。これは、`mysql_real_escape_string_quote()` のみが文字セット対応であるためです。マルチバイト文字セットを使用している場合は、他の関数を「バイパス」にできます (無効)。

- Perl DBI: プレースホルダまたは `quote()` メソッドを使用してください。
- Ruby DBI: プレースホルダまたは `quote()` メソッドを使用してください。
- Java JDBC: `PreparedStatement` オブジェクトおよびプレースホルダを使用してください。

ほかのプログラミングインタフェースも似たような機能を持っている場合があります。

6.2 アクセス制御とアカウント管理

MySQL では、クライアントユーザーがサーバーに接続し、サーバーによって管理されるデータにアクセスできるようにするアカウントを作成できます。MySQL 権限システムの主な役割は、特定のホストから接続するユーザーを認証すること、およびそのユーザーを、`SELECT`、`INSERT`、`UPDATE`、`DELETE` などのデータベースにおける権限に関連付けることです。追加機能には、管理操作の権限を付与する機能が含まれます。

接続できるユーザーを制御するために、各アカウントにパスワードなどの認証資格証明を割り当てることができません。MySQL アカウントへのユーザーインタフェースは、`CREATE USER`、`GRANT`、`REVOKE` などの SQL ステートメントで構成されます。 [セクション13.7.1「アカウント管理ステートメント」](#) を参照してください。

MySQL 権限システムによって、すべてのユーザーは自分に許可された操作のみ実行可能になります。ユーザーとして MySQL サーバーに接続すると、ユーザーの ID は、接続元のホストおよび指定したユーザー名によって決定されます。接続後にリクエストを発行すると、システムは、ユーザー ID とユーザーが行う操作に応じて権限を付与します。

MySQL ではホスト名とユーザー名の両方を考慮に入れてユーザーを特定しますが、これは、特定のユーザー名がすべてのホストで同一人物に属すると想定する根拠がないためです。たとえば、`office.example.com` から接続したユーザー `joe` は、`home.example.com` から接続した `joe` と同一人物とは限りません。MySQL では、たまたま同じ名前を持った異なるホスト上のユーザーを識別できるようにすることによってこれを処理します。つまり、`office.example.com` からの `joe` による接続に対して 1 つの権限セットを付与し、`home.example.com` からの `joe` による接続に対して別の権限セットを提供することができます。特定のアカウントが持つ権限を表示するには、`SHOW GRANTS` ステートメントを使用します。例:

```
SHOW GRANTS FOR 'joe'@office.example.com';
SHOW GRANTS FOR 'joe'@home.example.com';
```

内部的に、サーバーは `mysql` システムデータベースの付与テーブルに特権情報を格納します。MySQL サーバーはこれらのテーブルの内容を起動時にメモリーに読み取り、付与テーブルのインメモリーコピーに基づいてアクセス制御を決定します。

サーバーに接続するクライアントプログラムを実行するとき、MySQL アクセス制御には 2 つのステージがあります。

ステージ 1: サーバーは、ユーザーの ID および正しいパスワードを指定することによって ID を検証できるかどうかに基づいて、接続を受け入れるか拒否します。

ステージ 2: 接続できる場合、サーバーはユーザーが発行する各ステートメントを検査して、ステートメントを実行するだけの十分な権限をユーザーが持っているかどうかを判別します。たとえば、データベースのテーブルからレコードを選択したり、データベースのテーブルを削除したりしようとするとき、サーバーはユーザーにそのテーブルの `SELECT` 権限があるかどうか、またはデータベースの `DROP` 権限があるかどうかを検証します。

各ステージで発生する動作の詳細な説明については、[セクション6.2.6「アクセス制御、ステージ 1: 接続の検証」](#) および [セクション6.2.7「アクセス制御、ステージ 2: リクエストの確認」](#) を参照してください。権限に関連した問題の診断についての支援情報は、[セクション6.2.21「MySQL への接続の問題のトラブルシューティング」](#) を参照してください。

ユーザーの接続中に (ユーザー自身または別のどれかによって) 権限が変更された場合、それらの変更は、ユーザーが発行する次のステートメントで必ずしもすぐに有効になるわけではありません。サーバーが付与テーブルをリロードする条件についての詳細は、[セクション6.2.13「権限変更が有効化される時期」](#)を参照してください。

MySQL 権限システムでは実行できないこともあります。

- 特定ユーザーのアクセスを拒否するように明示的に指定することはできません。つまり、ユーザーを明示的に突き合わせて接続を拒否することはできません。
- ユーザーがデータベースのテーブルを作成または削除できるが、データベース自体の作成または削除はできないような権限をユーザーが持つように指定することはできません。
- パスワードはアカウントに対してグローバルに適用されます。データベース、テーブル、ルーチンなどの特定のオブジェクトにパスワードを関連付けることはできません。

6.2.1 アカウントのユーザー名とパスワード

MySQL は、[mysql](#) システムデータベースの `user` テーブルにアカウントを格納します。アカウントは、ユーザー名およびユーザーがサーバーに接続できるクライアントホスト (複数の場合あり) に関して定義されます。`user` テーブルでのアカウントの表示については、[セクション6.2.3「付与テーブル」](#)を参照してください。

アカウントは、パスワードなどの認証資格証明を持つこともできます。資格証明は、アカウント認証プラグインによって処理されます。MySQL は、複数の認証プラグインをサポートしています。これらの中には組み込みの認証方式を使用するものもあれば、外部の認証方式を使用した認証を有効にするものもあります。[セクション6.2.17「プラグイン認証」](#)を参照してください。

MySQL でのユーザー名とパスワードの使用法とオペレーティングシステムの使用法には、いくつかの違いがあります:

- MySQL で認証目的に使用されるユーザー名と、Windows または Unix で使用されるユーザー名 (ログイン名) とには、まったく関係がありません。Unix では、ほとんどの MySQL クライアントがデフォルトで、現在の Unix ユーザー名を MySQL ユーザー名として使用してログインを試みますが、これは便宜上の目的に過ぎません。クライアントプログラムでは、`-u` または `--user` オプションを使用して任意のユーザー名を指定することが許可されているため、簡単にデフォルトをオーバーライドできます。つまり、すべての MySQL アカウントにパスワードがないかぎり、誰でも任意のユーザー名を使用してサーバーへの接続を試みることができるため、どのような方法でもデータベースを保護することはできません。パスワードのないアカウントのユーザー名を指定したユーザーは、サーバーに正常に接続できます。
- MySQL ユーザー名の長さは最大 32 文字です。オペレーティングシステムのユーザー名の最大長が異なる場合があります。

警告

MySQL ユーザー名の長さ制限は MySQL サーバーおよびクライアントでハードコードされており、[mysql](#) データベース機能しない内のテーブルの定義を変更して回避しようとしています。

[mysql](#) データベースのテーブルの構造は、[セクション2.11「MySQL のアップグレード」](#)で説明されているプロシージャを使用しないかぎり、どのような方法でも変更しないでください。MySQL システムテーブルを他の方法で再定義しようとすると、未定義でサポートされていない動作が発生します。これらの変更の結果として誤った形式となった行を、サーバーは随意で無視します。

- 組み込み認証方式を使用するアカウントのクライアント接続を認証するために、サーバーは `user` テーブルに格納されているパスワードを使用します。これらのパスワードは、オペレーティングシステムにログインするためのパスワードとは異なります。Windows または Unix マシンにログインする際に使用される「外部」パスワードと、そのマシン上の MySQL サーバーにアクセスする際に使用されるパスワードとの間には、必要な関係はありません。

サーバーがほかのプラグインを使用してクライアントを認証する場合、プラグインが実装する認証方式は、`user` テーブルに格納されているパスワードを使用する場合と使用しない場合があります。この場合、MySQL サーバーへの認証を行う際に、外部パスワードも使用される可能性があります。

- `user` テーブルに格納されているパスワードは、プラグイン固有のアルゴリズムを使用して暗号化されます。

- ユーザー名とパスワードに ASCII 文字のみが含まれている場合は、文字セットの設定に関係なくサーバーに接続できます。ユーザー名またはパスワードに ASCII 以外の文字が含まれている場合に接続を有効にするには、クライアントアプリケーションで、`MYSQL_SET_CHARSET_NAME` オプションおよび適切な文字セット名を引数に指定して `mysql_options()` C API 関数をコールする必要があります。これにより、指定された文字セットを使用した認証が実行されます。そうしないと、サーバーのデフォルト文字セットが認証のデフォルトのエンコーディングと同じでないかぎり、認証は失敗します。

標準の MySQL クライアントプログラムでは、先ほど説明したように、`mysql_options()` が呼び出される `--default-character-set` オプションがサポートされています。さらに、[セクション10.4「接続文字セットおよび照合順序」](#)で説明したように、文字セットの自動検出もサポートされています。C API に基づいていないコネクタを使用するプログラムでは、`mysql_options()` と同等のものがコネクタで提供されている場合があり、それを代わりに使用できません。コネクタのドキュメントを確認してください。

前述の注は、クライアントの文字セットとして許可されていない `ucs2`、`utf16`、および `utf32` には適用されません。

MySQL のインストールプロセスでは、[セクション2.10.4「初期 MySQL アカウントの保護」](#)で説明されているように、付与テーブルに初期 `root` アカウントが移入されます。このアカウントにパスワードを割り当てる方法についても説明します。その後、通常は `CREATE USER`、`DROP USER`、`GRANT` や `REVOKE` などのステートメントを使用して、MySQL アカウントを設定、変更および削除します。[セクション6.2.8「アカウントの追加、権限の割当ておよびアカウントの削除」](#)および[セクション13.7.1「アカウント管理ステートメント」](#)を参照してください。

コマンドラインクライアントを使用して MySQL サーバーに接続するには、必要に応じて、使用するアカウントのユーザー名とパスワードのオプションを指定します:

```
shell> mysql --user=finley --password db_name
```

短いオプションを好む場合は、コマンドは次のようになります。

```
shell> mysql -u finley -p db_name
```

コマンドラインで `--password` または `-p` オプションの後にパスワード値を省略すると (図のように)、クライアントからパスワードの入力を求められます。または、コマンドラインでパスワードを指定できます:

```
shell> mysql --user=finley --password=password db_name
shell> mysql -u finley -ppassword db_name
```

`-p` オプションを使用する場合、`-p` と次のパスワード値の間にスペースなしが存在する必要があります。

コマンド行でのパスワード指定は、セキュアでないと考えべきです。[セクション6.1.2.1「パスワードセキュリティのためのエンドユーザーガイドライン」](#)を参照してください。コマンド行でパスワードを指定しないようにするには、オプションファイルまたはログインパスファイルを使用します。[セクション4.2.2.2「オプションファイルの使用」](#)および[セクション4.6.7「mysql_config_editor — MySQL 構成ユーティリティ」](#)を参照してください。

ユーザー名、パスワード、およびその他の接続パラメータの指定に関する追加情報については、[セクション4.2.4「コマンドオプションを使用した MySQL Server への接続」](#)を参照してください。

6.2.2 MySQL で提供される権限

MySQL アカウントに付与される権限によって、アカウントが実行できる操作が決まります。MySQL の権限は、適用されるコンテキストと操作のレベルによって異なります:

- 管理権限によって、ユーザーは MySQL サーバーの動作を管理できます。これらの権限は特定のデータベースに固有でないため、グローバルです。
- データベース権限は、データベースおよびデータベース内のすべてのオブジェクトに適用されます。これらの権限は、特定のデータベースに付与したり、すべてのデータベースに適用されるようにグローバルに付与したりすることができます。
- テーブル、インデックス、ビュー、ストアドルーチンなどのデータベースオブジェクトに対する権限は、データベース内の特定のオブジェクト、データベース内の特定のタイプのすべてのオブジェクト (データベース内のすべてのテーブルなど)、またはすべてのデータベース内の特定のタイプのすべてのオブジェクトに対してグローバルに付与できます。

権限は、静的 (サーバーに組み込まれている) か動的 (実行時に定義される) によっても異なります。権限が静的であるか動的であるかは、ユーザーアカウントおよびロールに付与される可用性に影響します。静的権限と動的権限の違いの詳細は、[静的権限と動的権限](#) を参照してください。)

アカウント権限に関する情報は、mysql システムデータベースの付与テーブルに格納されます。これらのテーブルの構造および内容の説明は、[セクション6.2.3「付与テーブル」](#) を参照してください。MySQL サーバーは、起動時に付与テーブルの内容をメモリーに読み取り、[セクション6.2.13「権限変更が有効化される時期」](#) に示されている状況下でそれらをリロードします。サーバーは、付与テーブルのメモリー内コピーに基づいてアクセス制御の決定を行います。

重要

一部の MySQL リリースでは、新しい権限または機能を追加するために付与テーブルに変更が導入されています。新しい機能を実際に利用できるようにするには、MySQL をアップグレードするたびに付与テーブルを現在の構造に更新します。[セクション2.11「MySQL のアップグレード」](#) を参照してください。

次の各セクションでは、使用可能な権限の概要、各権限の詳細な説明および使用上のガイドラインを示します。

- [使用可能な権限のサマリー](#)
- [静的権限の説明](#)
- [動的権限の説明](#)
- [権限付与のガイドライン](#)
- [静的権限と動的権限](#)
- [SUPER から動的権限へのアカウントの移行](#)

使用可能な権限のサマリー

次のテーブルに、GRANT および REVOKE ステートメントで使用される静的権限名と、付与テーブルの各権限に関連付けられたカラム名および権限が適用されるコンテキストを示します。

表 6.2 GRANT および REVOKE に許可される静的権限

権限	付与テーブルカラム	コンテキスト
ALL [PRIVILEGES]	「すべての権限」のシノニム	サーバー管理
ALTER	Alter_priv	テーブル
ALTER ROUTINE	Alter_routine_priv	ストアドルーチン
CREATE	Create_priv	データベース、テーブルまたはインデックス
CREATE ROLE	Create_role_priv	サーバー管理
CREATE ROUTINE	Create_routine_priv	ストアドルーチン
CREATE TABLESPACE	Create_tablespace_priv	サーバー管理
CREATE TEMPORARY TABLES	Create_tmp_table_priv	テーブル
CREATE USER	Create_user_priv	サーバー管理
CREATE VIEW	Create_view_priv	ビュー
DELETE	Delete_priv	テーブル
DROP	Drop_priv	データベース、テーブルまたはビュー
DROP ROLE	Drop_role_priv	サーバー管理
EVENT	Event_priv	データベース
EXECUTE	Execute_priv	ストアドルーチン
FILE	File_priv	サーバーホストでのファイルアクセス

権限	付与テーブルカラム	コンテキスト
GRANT OPTION	Grant_priv	データベース、テーブル、またはストアドルーチン
INDEX	Index_priv	テーブル
INSERT	Insert_priv	テーブルまたはカラム
LOCK TABLES	Lock_tables_priv	データベース
PROCESS	Process_priv	サーバー管理
PROXY	proxies_priv テーブルを参照	サーバー管理
REFERENCES	References_priv	データベースまたはテーブル
RELOAD	Reload_priv	サーバー管理
REPLICATION CLIENT	Repl_client_priv	サーバー管理
REPLICATION SLAVE	Repl_slave_priv	サーバー管理
SELECT	Select_priv	テーブルまたはカラム
SHOW DATABASES	Show_db_priv	サーバー管理
SHOW VIEW	Show_view_priv	ビュー
SHUTDOWN	Shutdown_priv	サーバー管理
SUPER	Super_priv	サーバー管理
TRIGGER	Trigger_priv	テーブル
UPDATE	Update_priv	テーブルまたはカラム
USAGE	「権限なし」のシノニムです	サーバー管理

次のテーブルに、GRANT および REVOKE ステートメントで使用される動的権限名と、その権限が適用されるコンテキストを示します。

表 6.3 GRANT および REVOKE に許可される動的権限

権限	コンテキスト
APPLICATION_PASSWORD_ADMIN	デュアルパスワード管理
AUDIT_ADMIN	監査ログの管理
BACKUP_ADMIN	バックアップ管理
BINLOG_ADMIN	バックアップとレプリケーションの管理
BINLOG_ENCRYPTION_ADMIN	バックアップとレプリケーションの管理
CLONE_ADMIN	クローン管理
CONNECTION_ADMIN	サーバー管理
ENCRYPTION_KEY_ADMIN	サーバー管理
FIREWALL_ADMIN	ファイアウォール管理
FIREWALL_USER	ファイアウォール管理
FLUSH_OPTIMIZER_COSTS	サーバー管理
FLUSH_STATUS	サーバー管理
FLUSH_TABLES	サーバー管理
FLUSH_USER_RESOURCES	サーバー管理
GROUP_REPLICATION_ADMIN	レプリケーション管理
INNODB_REDO_LOG_ARCHIVE	redo ログアーカイブ管理
NDB_STORED_USER	NDB Cluster
PERSIST_RO_VARIABLES_ADMIN	サーバー管理

権限	コンテキスト
REPLICATION_APPLIER	レプリケーションチャンネル用の PRIVILEGE_CHECKS_USER
REPLICATION_SLAVE_ADMIN	レプリケーション管理
RESOURCE_GROUP_ADMIN	リソースグループ管理
RESOURCE_GROUP_USER	リソースグループ管理
ROLE_ADMIN	サーバー管理
SESSION_VARIABLES_ADMIN	サーバー管理
SET_USER_ID	サーバー管理
SHOW_ROUTINE	サーバー管理
SYSTEM_USER	サーバー管理
SYSTEM_VARIABLES_ADMIN	サーバー管理
TABLE_ENCRYPTION_ADMIN	サーバー管理
VERSION_TOKEN_ADMIN	サーバー管理
XA_RECOVER_ADMIN	サーバー管理

静的権限の説明

静的権限は、実行時に定義される動的権限とは対照的に、サーバーに組み込まれます。次のリストでは、MySQL で使用可能な各静的権限について説明します。

特定の SQL ステートメントには、ここに示されているよりも具体的な権限要件がある場合もあります。そのような場合、該当するステートメントの説明で詳細を示します。

- [ALL, ALL PRIVILEGES](#)

これらの権限指定子は、「特定の権限レベルで使用可能なすべての権限」 ([GRANT OPTION](#) を除く) の短縮形です。たとえば、グローバルレベルまたはテーブルレベルで [ALL](#) を付与すると、すべてのグローバル権限またはすべてのテーブルレベル権限がそれぞれ付与されます。

- [ALTER](#)

[ALTER TABLE](#) ステートメントを使用してテーブルの構造を変更できます。[ALTER TABLE](#) には [CREATE](#) および [INSERT](#) 権限も必要です。テーブルの名前を変更するには、古いテーブルで [ALTER](#) および [DROP](#) を、新しいテーブルで [CREATE](#) および [INSERT](#) を実行する必要があります。

- [ALTER ROUTINE](#)

ストアドルーチン (ストアドプロシージャおよびストアドファンクション) を変更または削除するステートメントの使用を有効にします。権限が付与されるスコープ内にあり、ユーザーがルーチン [DEFINER](#) として指定されたユーザーではないルーチンの場合、ルーチン定義以外のルーチンプロパティへのアクセスも有効になります。

- [CREATE](#)

新しいデータベースおよびテーブルを作成するステートメントの使用を有効にします。

- [CREATE ROLE](#)

[CREATE ROLE](#) ステートメントの使用を有効にします。([CREATE USER](#) 権限により、[CREATE ROLE](#) ステートメントの使用も可能になります。) [セクション6.2.10「ロールの使用」](#) を参照してください。

[CREATE ROLE](#) および [DROP ROLE](#) 権限は、アカウントの作成および削除にのみ使用できるため、[CREATE USER](#) ほど強力ではありません。これらは、[CREATE USER](#) でアカウント属性を変更したり、アカウントの名前を変更できるため、使用できません。[ユーザーとロールの互換性](#) を参照してください。

- [CREATE ROUTINE](#)

ストアドルーチン (ストアドプロシージャとストアドファンクション) を作成するステートメントの使用を有効にします。権限が付与されるスコープ内にあり、ユーザーがルーチン **DEFINER** として指定されたユーザーではないルーチンの場合、ルーチン定義以外のルーチンプロパティへのアクセスも有効になります。

- **CREATE TABLESPACE**

テーブルスペースおよびログファイルグループを作成、変更または削除するステートメントを使用可能にします。

- **CREATE TEMPORARY TABLES**

CREATE TEMPORARY TABLE ステートメントを使用した一時テーブルの作成を有効にします。

セッションが一時テーブルを作成したあと、サーバーはそのテーブルに対するそれ以上の権限チェックを実行しません。セッションの作成によって、**DROP TABLE**、**INSERT**、**UPDATE**、**SELECT** などのあらゆる操作をテーブル上で実行できます。詳細は、[セクション13.1.20.2「CREATE TEMPORARY TABLE ステートメント」](#)を参照してください。

- **CREATE USER**

ALTER USER、**CREATE ROLE**、**CREATE USER**、**DROP ROLE**、**DROP USER**、**RENAME USER** および **REVOKE ALL PRIVILEGES** ステートメントの使用を有効にします。

- **CREATE VIEW**

CREATE VIEW ステートメントの使用を有効にします。

- **DELETE**

データベース内のテーブルから行を削除できます。

- **DROP**

既存のデータベース、テーブルおよびビューを削除するステートメントを使用可能にします。パーティションテーブルで **ALTER TABLE ... DROP PARTITION** ステートメントを使用するには、**DROP** 権限が必要です。**DROP** 権限は **TRUNCATE TABLE** のためにも必要です。

- **DROP ROLE**

DROP ROLE ステートメントの使用を有効にします。(**CREATE USER** 権限により、**DROP ROLE** ステートメントの使用も可能になります。) [セクション6.2.10「ロールの使用」](#)を参照してください。

CREATE ROLE および **DROP ROLE** 権限は、アカウントの作成および削除にのみ使用できるため、**CREATE USER** ほど強力ではありません。これらは、**CREATE USER** でアカウント属性を変更したり、アカウントの名前を変更できるため、使用できません。[ユーザーとロールの互換性](#)を参照してください。

- **EVENT**

イベントスケジューラのイベントを作成、変更、削除、または表示するステートメントの使用を有効にします。

- **EXECUTE**

ストアドルーチン (ストアドプロシージャとストアドファンクション) を実行するステートメントの使用を有効にします。権限が付与されるスコープ内にあり、ユーザーがルーチン **DEFINER** として指定されたユーザーではないルーチンの場合、ルーチン定義以外のルーチンプロパティへのアクセスも有効になります。

- **FILE**

次の操作およびサーバーの動作に影響します:

- **LOAD DATA** および **SELECT ... INTO OUTFILE** ステートメントと **LOAD_FILE()** 関数を使用して、サーバーホスト上のファイルの読み取りおよび書き込みを有効にします。**FILE** 権限を持つユーザーは、すべてのユーザーから読み取り可能であるか、MySQL サーバーによって読み取り可能なサーバーホスト上のすべてのファイルを読み取ることができません。(このことは暗黙的に、データベースディレクトリ内のあらゆるファイルにサーバーからアクセスできるため、ユーザーはそれらのすべてのファイルを読み取ることができるとを意味します。)

- MySQL サーバーが書き込みアクセス権を持つ任意のディレクトリに新しいファイルを作成できます。これは、権限テーブルを実装するファイルを格納するサーバーのデータディレクトリを含みます。
- `CREATE TABLE` ステートメントに `DATA DIRECTORY` または `INDEX DIRECTORY` テーブルオプションを使用できるようにします。

セキュリティ対策として、サーバーは既存のファイルを上書きしません。

ファイルの読取りおよび書き込みが可能な場所を制限するには、`secure_file_priv` システム変数を特定のディレクトリに設定します。セクション5.1.8「サーバーシステム変数」を参照してください。

- **GRANT OPTION**

自分が所有する権限を他のユーザーに付与したり、他のユーザーから取り消すことができます。

- **INDEX**

インデックスを作成または削除するステートメントの使用を有効にします。INDEX は既存のテーブルに適用されます。テーブルに対する `CREATE` 権限を持つ場合、`CREATE TABLE` ステートメントにインデックス定義を含めることも可能です。

- **INSERT**

データベースのテーブルに行を挿入できます。 `ANALYZE TABLE`、`OPTIMIZE TABLE`、`REPAIR TABLE` などのテーブル保守に関するステートメントにも、`INSERT` 権限が必要です。

- **LOCK TABLES**

明示的な `LOCK TABLES` ステートメントを使用して、`SELECT` 権限を持つテーブルをロックできます。これには、他のセッションによるロックされたテーブルの読取りを防止する書き込みロックの使用が含まれます。

- **PROCESS**

`PROCESS` 権限は、サーバー内で実行されているスレッドに関する情報 (セッションによって実行されているステートメントに関する情報) へのアクセスを制御します。 `SHOW PROCESSLIST` ステートメント、`mysqladmin processlist` コマンド、`INFORMATION_SCHEMA.PROCESSLIST` テーブル、およびパフォーマンススキーマ `processlist` テーブルを使用して使用可能なスレッド情報には、次のようにアクセスできます:

- `PROCESS` 権限を持つユーザーは、他のユーザーに属するスレッドも含め、すべてのスレッドに関する情報にアクセスできます。
- `PROCESS` 権限がない場合、非匿名ユーザーは自分のスレッドに関する情報にアクセスできますが、他のユーザーのスレッドにはアクセスできません。また、匿名ユーザーはスレッド情報にアクセスできません。

注記

パフォーマンススキーマ `threads` テーブルはスレッド情報も提供しますが、テーブルアクセスは別の特権モデルを使用します。セクション27.12.19.10「スレッドテーブル」を参照してください。

`PROCESS` 権限では、`SHOW ENGINE` ステートメントの使用、`INFORMATION_SCHEMA.InnoDB` テーブル (`INNODB_` で始まる名前を持つテーブル) へのアクセスおよび、(MySQL 8.0.21 の時点で) `INFORMATION_SCHEMA.FILES` テーブルへのアクセスも可能です。

- **PROXY**

あるユーザーが別のユーザーになりすましたり、別のユーザーとして知られるようになります。セクション6.2.18「プロキシユーザー」を参照してください。

- **REFERENCES**

外部キー制約を作成するには、親テーブルに対する `REFERENCES` 権限が必要です。

- **RELOAD**

RELOAD では、次の操作が可能です:

- FLUSH ステートメントの使用。
- FLUSH 操作と同等の `mysqladmin` コマンドの使用: `flush-hosts`, `flush-logs`, `flush-privileges`, `flush-status`, `flush-tables`, `flush-threads`, `refresh` および `reload`。

`reload` コマンドは、付与テーブルをメモリーにリロードするようにサーバーに指示します。`flush-privileges` は `reload` のシノニムです。`refresh` コマンドは、ログファイルを閉じて再オープンし、すべてのテーブルをフラッシュします。ほかの `flush-xxx` コマンドは `refresh` に類似した機能を実行しますが、より具体的であるため一部の状況では好ましい場合があります。たとえば、ログファイルだけをフラッシュするときは、`refresh` よりも `flush-logs` を選択することをお勧めします。

- 様々な FLUSH 操作を実行する `mysqldump` オプションの使用: `--flush-logs` および `--master-data`。
- RESET MASTER および RESET REPLICA | SLAVE ステートメントの使用。

• REPLICATION CLIENT

SHOW MASTER STATUS、SHOW REPLICA | SLAVE STATUS および SHOW BINARY LOGS ステートメントの使用を有効にします。レプリカがレプリケーションソースサーバーとして現在のサーバーに接続するために使用するアカウントにこの権限を付与します。

• REPLICATION SLAVE

SHOW REPLICAS | SHOW SLAVE HOSTS、SHOW RELAYLOG EVENTS および SHOW BINLOG EVENTS ステートメントを使用して、レプリケーションソースサーバー上のデータベースに対して行われた更新をアカウントがリクエストできるようにします。この権限は、`mysqlbinlog` オプションの `--read-from-remote-server (-R)` および `--read-from-remote-master` を使用する場合にも必要です。レプリカがレプリケーションソースサーバーとして現在のサーバーに接続するために使用するアカウントにこの権限を付与します。

• SELECT

データベース内のテーブルから行を選択できます。SELECT ステートメントには、実際にテーブルにアクセスする場合にのみ SELECT 権限が必要です。一部の SELECT ステートメントはテーブルにアクセスしないため、あらゆるデータベースへのアクセス権がなくても実行できます。たとえば、テーブルを参照しない式を評価するための単純な計算機として SELECT を使用することができます。

```
SELECT 1+1;  
SELECT PI()*2;
```

SELECT 権限は、カラム値を読み取るほかのステートメントについても必要です。たとえば、UPDATE ステートメントの代入 `col_name=expr` の右辺で参照されるカラムや、DELETE または UPDATE ステートメントの WHERE 句で指定されるカラムについて、SELECT が必要です。

EXPLAIN で使用されるテーブルまたはビュー (ビュー定義の基礎となるテーブルを含む) には、SELECT 権限が必要です。

• SHOW DATABASES

SHOW DATABASE ステートメントを発行して、アカウントがデータベース名を表示できるようにします。この権限を持たないアカウントには、アカウントが一部の権限を持つデータベースしか表示されず、サーバーが `--skip-show-database` オプションで起動されている場合はステートメントを一切使用できません。

注意

静的グローバル権限はすべてのデータベースに対する権限とみなされるため、静的グローバル権限を使用すると、ユーザーは、部分的な取消しによってデータベースレベルで制限されているデータベースを除き、SHOW DATABASES を使用するが、INFORMATION_SCHEMA の SCHEMATA テーブルを調べることで、すべてのデータベース名を表示できます。

• SHOW VIEW

`SHOW CREATE VIEW` ステートメントの使用を有効にします。この権限は、`EXPLAIN` で使用されるビューにも必要です。

- `SHUTDOWN`

`SHUTDOWN` および `RESTART` ステートメント、`mysqladmin shutdown` コマンドおよび `mysql_shutdown()` C API 関数の使用を有効にします。

- SUPER

SUPER は強力な遠隔リカバリ権限であるため、軽く付与しないでください。アカウントで **SUPER** 操作のサブセットのみを実行する必要がある場合は、かわりに 1 つ以上の動的権限を付与することで、必要な権限セットを実現できる場合があります、それぞれがより制限された機能を構成します。 [動的権限の説明](#) を参照してください。

注記

SUPER は非推奨であり、将来のバージョンの MySQL で削除される予定です。 **SUPER** から動的権限へのアカウントの移行を参照してください。

SUPER は、次の操作およびサーバーの動作に影響します:

- 実行時のシステム変数の変更を有効にします:
 - **SET GLOBAL** および **SET PERSIST** を使用して、グローバルシステム変数に対するサーバー構成の変更を有効にします。
対応する動的権限は **SYSTEM_VARIABLES_ADMIN** です。
 - 特別な権限を必要とする制限付きセッションシステム変数の設定を有効にします。
対応する動的権限は **SESSION_VARIABLES_ADMIN** です。
[セクション5.1.9.1「システム変数権限」](#) も参照してください。
- グローバルトランザクション特性の変更を有効にします ([セクション13.3.7「SET TRANSACTION ステートメント」](#) を参照)。
対応する動的権限は **SYSTEM_VARIABLES_ADMIN** です。
- グループレプリケーションを含む、レプリケーションを開始および停止するアカウントを有効にします。
対応する動的権限は、通常のレプリケーションの場合は **REPLICATION_SLAVE_ADMIN**、グループレプリケーションの場合は **GROUP_REPLICATION_ADMIN** です。
- (MySQL 8.0.23 の) **CHANGE REPLICATION SOURCE TO** ステートメント、(MySQL 8.0.23 の前の) **CHANGE MASTER TO** ステートメントおよび **CHANGE REPLICATION FILTER** ステートメントを使用できます。
対応する動的権限は **REPLICATION_SLAVE_ADMIN** です。
- **PURGE BINARY LOGS** および **BINLOG** ステートメントを使用してバイナリログ制御を有効にします。
対応する動的権限は **BINLOG_ADMIN** です。
- ビューまたはストアードプログラムの実行時に有効な承認 ID を設定できるようにします。この権限を持つユーザーは、ビューまたはストアードプログラムの **DEFINER** 属性で任意のアカウントを指定できます。
対応する動的権限は **SET_USER_ID** です。
- **CREATE SERVER**、**ALTER SERVER** および **DROP SERVER** ステートメントの使用を有効にします。
- **mysqladmin debug** コマンドの使用を有効にします。
- **InnoDB** 暗号化キーのローテーションを有効にします。
対応する動的権限は **ENCRYPTION_KEY_ADMIN** です。
- バージョントークンのユーザー定義関数の実行を有効にします。
対応する動的権限は **VERSION_TOKEN_ADMIN** です。
- ロールの付与と取消し、**GRANT** ステートメントの **WITH ADMIN OPTION** 句の使用、および **ROLES_GRAPHML()** 関数からの結果の空でない **<graphml>** 要素コンテンツを有効にします。

対応する動的権限は `ROLE_ADMIN` です。

- `SUPER` 以外のアカウントに対して許可されていないクライアント接続の制御を有効にします:
 - `KILL` ステートメントまたは `mysqladmin kill` コマンドを使用して、他のアカウントに属するスレッドを強制終了できます。(アカウントは常に独自のスレッドを強制終了できます。)
 - サーバーは、`SUPER` クライアントの接続時に `init_connect` システム変数の内容を実行しません。
 - サーバーは、`max_connections` システム変数で構成された接続制限に達した場合でも、`SUPER` クライアントからの接続を受け入れます。
 - オフラインモード (`offline_mode` が有効) のサーバーは、次のクライアントリクエスト時に `SUPER` クライアント接続を終了せず、`SUPER` クライアントからの新しい接続を受け入れます。
 - `read_only` システム変数が有効な場合でも、更新を実行できます。これは、明示的なテーブル更新、およびテーブルを暗黙的に更新する `GRANT` や `REVOKE` などのアカウント管理ステートメントの使用に適用されません。

前述の接続制御操作に対応する動的権限は、`CONNECTION_ADMIN` です。

[セクション25.7「ストアドプログラムバイナリロギング」](#) で説明されているように、バイナリロギングが有効になっている場合は、ストアドファンクションを作成または変更するために `SUPER` 権限が必要になることもあります。

- `TRIGGER`

トリガー操作を有効にします。テーブルのトリガーを作成、削除、実行または表示するには、そのテーブルに対するこの権限が必要です。

トリガーが (トリガーに関連付けられたテーブルに対して `INSERT`、`UPDATE` または `DELETE` ステートメントを実行する権限を持つユーザーによって) アクティブ化された場合、トリガーの実行には、トリガーを定義したユーザーがそのテーブルに対する `TRIGGER` 権限を持っている必要があります。

- `UPDATE`

データベース内のテーブルで行を更新できます。

- `USAGE`

この権限指定子は、「権限なし」を表します。`GRANT` でグローバルレベルで使用され、権限リスト内の特定のアカウント権限に名前を付けずに `WITH GRANT OPTION` などの句を指定します。`SHOW GRANTS` には、アカウントに権限レベルの権限がないことを示す `USAGE` が表示されます。

動的権限の説明

サーバーに組み込まれている静的権限とは対照的に、動的権限は実行時に定義されます。次のリストでは、MySQL で使用可能な各動的権限について説明します。

ほとんどの動的権限は、サーバーの起動時に定義されます。その他は、権限の説明に示されているように、特定のコンポーネントまたはプラグインによって定義されます。このような場合、権限を定義するコンポーネントまたはプラグインが有効になっていないかぎり、権限は使用できません。

特定の SQL ステートメントには、ここに示されているよりも具体的な権限要件がある場合もあります。そのような場合、該当するステートメントの説明で詳細を示します。

- `APPLICATION_PASSWORD_ADMIN` (MySQL 8.0.14 で追加)

デュアルパスワード機能の場合、この権限により、自分のアカウントに適用される `ALTER USER` ステートメントおよび `SET PASSWORD` ステートメントに `RETAIN CURRENT PASSWORD` 句および `DISCARD OLD PASSWORD` 句を使用できます。ほとんどのユーザーは 1 つのパスワードのみを必要とするため、自分のセカンダリパスワードを操作するにはこの権限が必要です。

アカウントがすべてのアカウントのセカンダリパスワードの操作を許可される場合は、`APPLICATION_PASSWORD_ADMIN` ではなく `CREATE USER` 権限を付与する必要があります。

デュアルパスワードの使用の詳細は、[セクション6.2.15「パスワード管理」](#) を参照してください。

- `AUDIT_ADMIN`

監査ログ構成を有効にします。この権限は、`audit_log` プラグインによって定義されます。[セクション6.4.5「MySQL Enterprise Audit」](#) を参照してください。

- `BACKUP_ADMIN`

`LOCK INSTANCE FOR BACKUP` ステートメントの実行およびパフォーマンススキーマ `log_status` テーブルへのアクセスを有効にします。

注記

`BACKUP_ADMIN` の他に、`log_status` テーブルに対する `SELECT` 権限もアクセスに必要です。

以前のバージョンから MySQL 8.0 へのインプレースアップグレードを実行すると、`RELOAD` 権限を持つユーザーに `BACKUP_ADMIN` 権限が自動的に付与されます。

- `BINLOG_ADMIN`

`PURGE BINARY LOGS` および `BINLOG` ステートメントを使用してバイナリログ制御を有効にします。

- `BINLOG_ENCRYPTION_ADMIN`

バイナリログファイルおよびリレーログファイルの暗号化をアクティブまたは非アクティブにするシステム変数 `binlog_encryption` の設定を有効にします。この機能は、`BINLOG_ADMIN`、`SYSTEM_VARIABLES_ADMIN` または `SESSION_VARIABLES_ADMIN` 権限では提供されません。サーバーの再起動時にバイナリログマスターキーを自動的にローテーションする関連システム変数 `binlog_rotate_encryption_master_key_at_startup` には、この権限は必要ありません。

- `CLONE_ADMIN`

`CLONE` ステートメントの実行を有効にします。`BACKUP_ADMIN` および `SHUTDOWN` 権限が含まれます。

- `CONNECTION_ADMIN`

`KILL` ステートメントまたは `mysqladmin kill` コマンドを使用して、他のアカウントに属するスレッドを強制終了できます。(アカウントは常に独自のスレッドを強制終了できます。)

クライアント接続に関連するシステム変数の設定、またはクライアント接続に関連する制限の回避を有効にします。`CONNECTION_ADMIN` は、次のシステム変数の影響に適用されます:

- `init_connect`: サーバーは、`CONNECTION_ADMIN` クライアントの接続時に `init_connect` システム変数の内容を実行しません。
- `max_connections`: サーバーは、`max_connections` システム変数で構成された接続制限に達した場合でも、`CONNECTION_ADMIN` クライアントからの接続を受け入れます。
- `offline_mode`: オフラインモード (`offline_mode` が有効) のサーバーは、次のクライアントリクエスト時に `CONNECTION_ADMIN` クライアント接続を終了せず、`CONNECTION_ADMIN` クライアントからの新しい接続を受け入れます。
- `read_only`: `read_only` システム変数が有効な場合でも、更新を実行できます。これは、明示的なテーブル更新、およびテーブルを暗黙的に更新する `GRANT` や `REVOKE` などのアカウント管理ステートメントの使用に適用されます。

- `ENCRYPTION_KEY_ADMIN`

InnoDB 暗号化キーのローテーションを有効にします。

- **FIREWALL_ADMIN**

ユーザーは、任意のユーザーのファイアウォールルールを管理できます。この権限は、**MYSQL_FIREWALL** プラグインによって定義されます。セクション6.4.7「MySQL Enterprise Firewall」を参照してください。

- **FIREWALL_USER**

ユーザーが独自のファイアウォールルールを更新できるようにします。この権限は、**MYSQL_FIREWALL** プラグインによって定義されます。セクション6.4.7「MySQL Enterprise Firewall」を参照してください。

- **FLUSH_OPTIMIZER_COSTS** (MySQL 8.0.23 で追加)

FLUSH OPTIMIZER_COSTS ステートメントの使用を有効にします。

- **FLUSH_STATUS** (MySQL 8.0.23 で追加)

FLUSH STATUS ステートメントの使用を有効にします。

- **FLUSH_TABLES** (MySQL 8.0.23 で追加)

FLUSH TABLES ステートメントの使用を有効にします。

- **FLUSH_USER_RESOURCES** (MySQL 8.0.23 で追加)

FLUSH USER_RESOURCES ステートメントの使用を有効にします。

- **GROUP_REPLICATION_ADMIN**

START GROUP REPLICATION および **STOP GROUP REPLICATION** ステートメントを使用してグループレプリケーションを開始および停止し、**group_replication_consistency** システム変数のグローバル設定を変更し、**group_replication_set_write_concurrency()** および **group_replication_set_communication_protocol()** UDF を使用できるようにします。レプリケーショングループのメンバーであるサーバーの管理に使用されるアカウントにこの権限を付与します。

- **INNODB_REDO_LOG_ARCHIVE**

アカウントが redo ログアーカイブをアクティブ化および非アクティブ化できるようにします。

- **INNODB_REDO_LOG_ENABLE**

ALTER INSTANCE {ENABLE|DISABLE} INNODB REDO_LOG ステートメントを使用して redo ロギングを有効または無効にします。MySQL 8.0.21 で導入されました。

[redo ロギングの無効化](#)を参照してください。

- **NDB_STORED_USER**

特定の NDB Cluster に参加するとすぐに、ユーザーまたは役割とその権限をすべての NDB 対応 MySQL サーバー間で共有および同期できるようにします。この権限は、NDB ストレージエンジンが有効になっている場合にのみ使用できます。

指定されたユーザーまたはロールに対して行われた権限の変更または取消しは、接続されているすべての MySQL サーバー (SQL ノード) とただちに同期化されます。異なる SQL ノードから発生した権限に影響する複数のステートメントが、すべての SQL ノードで同じ順序で実行されることは保証されないことに注意してください。このため、すべてのユーザー管理は単一の指定された SQL ノードから実行することを強くお勧めします。

NDB_STORED_USER はグローバル権限であり、**ON *.*** を使用して付与または取り消す必要があります。この権限に他のスコープを設定しようとすると、エラーになります。この権限は、ほとんどのアプリケーションユーザーおよび管理ユーザーに付与できますが、**mysql.session@localhost** や **mysql.infoschema@localhost** などのシステム予約アカウントには付与できません。

NDB_STORED_USER 権限を付与されたユーザーは、この権限を持つロールと同様に、NDB に格納されます (したがって、すべての SQL ノードで共有されます)。NDB_STORED_USER を持つロールのみを付与されたユーザーは、NDB には格納されません。各 NDB 格納ユーザーには、権限を明示的に付与する必要があります。

NDB でのこの動作の詳細は、[セクション23.5.12「NDB_STORED_USER での分散 MySQL 権限」](#)を参照してください。

NDB_STORED_USER 権限は NDB 8.0.18 以降で使用できます。

- [PERSIST_RO_VARIABLES_ADMIN](#)

[SYSTEM_VARIABLES_ADMIN](#) も持っているユーザーの場合、[PERSIST_RO_VARIABLES_ADMIN](#) を使用すると、[SET PERSIST_ONLY](#) を使用してグローバルシステム変数をデータディレクトリの `mysqld-auto.cnf` オプションファイルに永続化できます。このステートメントは [SET PERSIST](#) と似ていますが、ランタイムグローバルシステム変数の値は変更されません。これにより、[SET PERSIST_ONLY](#) は、サーバーの起動時にのみ設定できる読み取り専用システム変数の構成に適しています。

[セクション5.1.9.1「システム変数権限」](#)も参照してください。

- [REPLICATION_APPLIER](#)

アカウントをレプリケーションチャンネルの [PRIVILEGE_CHECKS_USER](#) として機能させ、`mysqlbinlog` 出力で [BINLOG](#) ステートメントを実行できるようにします。[CHANGE REPLICATION SOURCE TO](#) (MySQL 8.0.23 から) または [CHANGE MASTER TO](#) (MySQL 8.0.23 より前) を使用して割り当てられたアカウントにこの権限を付与して、レプリケーションチャンネルのセキュリティコンテキストを提供し、それらのチャンネルでレプリケーションエラーを処理します。[REPLICATION_APPLIER](#) 権限と同様に、レプリケーションチャンネルで受信したトランザクションまたは `mysqlbinlog` 出力に含まれるトランザクションを実行するために必要な権限をアカウントに付与して、影響を受けるテーブルを更新する必要があります。詳細は、[セクション17.3.3「レプリケーション権限チェック」](#)を参照してください。

- [REPLICATION_SLAVE_ADMIN](#)

アカウントがレプリケーションソースサーバーに接続し、[START REPLICA | SLAVE](#) および [STOP REPLICA | SLAVE](#) ステートメントを使用してレプリケーションを開始および停止し、[CHANGE REPLICATION SOURCE TO](#) ステートメント (MySQL 8.0.23 から) または [CHANGE MASTER TO](#) ステートメント (MySQL 8.0.23 より前) および [CHANGE REPLICATION FILTER](#) ステートメントを使用します。レプリカがレプリケーションソースサーバーとして現在のサーバーに接続するために使用するアカウントにこの権限を付与します。この権限はグループレプリケーションには適用されません。そのためには [GROUP_REPLICATION_ADMIN](#) を使用します。

- [RESOURCE_GROUP_ADMIN](#)

リソースグループの作成、変更および削除、およびリソースグループへのスレッドとステートメントの割り当てで構成されるリソースグループ管理を有効にします。この権限を持つユーザーは、リソースグループに関連する操作を実行できます。

- [RESOURCE_GROUP_USER](#)

リソースグループへのスレッドとステートメントの割り当てを有効にします。この権限を持つユーザーは、[SET RESOURCE_GROUP](#) ステートメントおよび [RESOURCE_GROUP](#) オプティマイザヒントを使用できます。

- [ROLE_ADMIN](#)

ロールの付与と取消し、[GRANT](#) ステートメントの [WITH ADMIN OPTION](#) 句の使用、および [ROLES_GRAPHML\(\)](#) 関数からの結果の空でない `<graphml>` 要素コンテンツを有効にします。`mandatory_roles` システム変数の値を設定するために必要です。

- [SERVICE_CONNECTION_ADMIN](#)

管理接続のみを許可するネットワークインタフェースへの接続を有効にします ([セクション5.1.12.1「接続インタフェース」](#)を参照)。

- [SESSION_VARIABLES_ADMIN](#) (MySQL 8.0.14 で追加)

ほとんどのシステム変数では、セッション値の設定に特別な権限は必要なく、すべてのユーザーが現在のセッションに影響を与えることができます。一部のシステム変数では、セッション値を設定すると、現在のセッションの外部に影響を与える可能性があるため、操作が制限されます。これらの場合、[SESSION_VARIABLES_ADMIN](#) 権限により、ユーザーはセッション値を設定できます。

システム変数が制限されていて、セッション値を設定するために特別な権限が必要な場合、変数の説明にその制限が示されます。例として、`binlog_format`、`sql_log_bin` および `sql_log_off` があります。

`SESSION_VARIABLES_ADMIN` が追加された MySQL 8.0.14 より前は、`SYSTEM_VARIABLES_ADMIN` または `SUPER` 権限を持つユーザーのみが制限付きセッションシステム変数を設定できます。

`SESSION_VARIABLES_ADMIN` 権限は、`SYSTEM_VARIABLES_ADMIN` および `SUPER` 権限のサブセットです。これらの権限のいずれかを持つユーザーは、制限付きセッション変数の設定も許可され、効果的には `SESSION_VARIABLES_ADMIN` を意味するため、`SESSION_VARIABLES_ADMIN` を明示的に付与する必要はありません。

[セクション5.1.9.1「システム変数権限」](#) も参照してください。

- `SET_USER_ID`

ビューまたはストアプログラムの実行時に有効な承認 ID を設定できるようにします。この権限を持つユーザーは、ビューまたはストアプログラムの `DEFINER` 属性として任意のアカウントを指定できます。

MySQL 8.0.22 では、`SET_USER_ID` を使用して、(おそらく誤って) ストアドオブジェクトが孤立したり、現在孤立しているストアオブジェクトを採用する原因となる操作を防ぐために設計されたセキュリティチェックをオーバーライドすることもできます。詳細は、[孤立したストアオブジェクト](#) を参照してください。

- `SHOW_ROUTINE` (MySQL 8.0.20 で追加)

ユーザーがルーチン `DEFINER` として指定されていないストアルーチン (ストアプロシージャおよびストアファンクション) の定義およびプロパティにアクセスできるようにします。このアクセスには、次のものが含まれます:

- `INFORMATION_SCHEMA.ROUTINES` テーブルの内容。
- `SHOW CREATE FUNCTION` および `SHOW CREATE PROCEDURE` ステートメント。
- `SHOW FUNCTION CODE` および `SHOW PROCEDURE CODE` ステートメント。
- `SHOW FUNCTION STATUS` および `SHOW PROCEDURE STATUS` ステートメント。

MySQL 8.0.20 より前では、ユーザーが定義しなかったルーチンの定義にアクセスするには、グローバルな `SELECT` 権限が必要です。これは非常に広範です。8.0.20 の時点では、代わりに、ルーチン定義へのアクセスを許可するより制限されたスコープを持つ特権として `SHOW_ROUTINE` を付与できます。(つまり、管理者は、それを必要としないユーザーからグローバル `SELECT` を廃棄し、かわりに `SHOW_ROUTINE` を付与できます。) これにより、アカウントは広範囲の権限を必要とせずにストアルーチンをバックアップできます。

- `SYSTEM_USER` (MySQL 8.0.16 で追加)

`SYSTEM_USER` 権限は、システムユーザーを通常のユーザーと区別します:

- `SYSTEM_USER` 権限を持つユーザーはシステムユーザーです。
- `SYSTEM_USER` 権限を持たないユーザーは通常のユーザーです。

`SYSTEM_USER` 権限は、特定のユーザーが他の権限を適用できるアカウント、およびそのユーザーが他のアカウントから保護されているかどうかに影響します:

- システムユーザーは、システムアカウントと通常アカウントの両方を変更できます。つまり、通常アカウントに対して特定の操作を実行する適切な権限を持つユーザーは、システムアカウントに対しても操作を実行するよ

うに `SYSTEM_USER` を所有することで有効になります。システムアカウントは、通常のユーザーではなく、適切な権限を持つシステムユーザーのみが変更できます。

- 適切な権限を持つ通常のユーザーは通常のアカウントを変更できますが、システムアカウントは変更できません。通常のアカウントは、適切な権限を持つシステムユーザーと通常のユーザーの両方が変更できます。

詳細は、[セクション6.2.11「アカウントカテゴリ」](#)を参照してください。

`SYSTEM_USER` 権限によってシステムアカウントに付与される通常のアカウントによる変更からの保護は、`mysql` システムスキーマに対する権限を持つ通常のアカウントには適用されないため、そのスキーマの付与テーブルを直接変更できます。完全保護のために、`mysql` スキーマ権限を通常のアカウントに付与しないでください。[通常アカウントによる操作からのシステムアカウントの保護](#)を参照してください。

• `SYSTEM_VARIABLES_ADMIN`

次の操作およびサーバーの動作に影響します:

- 実行時のシステム変数の変更を有効にします:
 - `SET GLOBAL` および `SET PERSIST` を使用して、グローバルシステム変数に対するサーバー構成の変更を有効にします。
 - ユーザーが `PERSIST_RO_VARIABLES_ADMIN` も持っている場合、`SET PERSIST_ONLY` を使用してグローバルシステム変数に対するサーバー構成の変更を有効にします。
 - 特別な権限を必要とする制限付きセッションシステム変数の設定を有効にします。実際には、`SYSTEM_VARIABLES_ADMIN` は、`SESSION_VARIABLES_ADMIN` を明示的に付与せずに `SESSION_VARIABLES_ADMIN` を暗黙的に意味します。

[セクション5.1.9.1「システム変数権限」](#)も参照してください。

- グローバルトランザクション特性の変更を有効にします ([セクション13.3.7「SET TRANSACTION ステートメント」](#)を参照)。

• `TABLE_ENCRYPTION_ADMIN` (MySQL 8.0.16 で追加)

`table_encryption_privilege_check` が有効な場合に、ユーザーがデフォルトの暗号化設定をオーバーライドできるようにします。[スキーマおよび一般テーブルスペースの暗号化デフォルトの定義](#)を参照してください。

• `VERSION_TOKEN_ADMIN`

バージョントークンのユーザー定義関数の実行を有効にします。この権限は、`version_tokens` プラグインによって定義されます。[セクション5.6.6「バージョントークン」](#)を参照してください。

• `XA_RECOVER_ADMIN`

`XA RECOVER` ステートメントの実行を有効にします。[セクション13.3.8.1「XA トランザクション SQL ステートメント」](#)を参照してください。

MySQL 8.0 より前は、すべてのユーザーが `XA RECOVER` ステートメントを実行して、未処理の準備済 XA トランザクションの XID 値を検出できたため、XA トランザクションを開始したユーザー以外のユーザーによる XA トランザクションのコミットまたはロールバックが発生する可能性があります。MySQL 8.0 では、`XA RECOVER` は `XA_RECOVER_ADMIN` 権限を持つユーザーにのみ許可されます。これは、それを必要とする管理ユーザーにのみ付与されることが予想されます。たとえば、XA アプリケーションがクラッシュし、ロールバックできるようにアプリケーションによって開始された未処理のトランザクションを検索する必要がある場合などです。この権限要件により、ユーザーは自分以外の未処理の準備済 XA トランザクションの XID 値を検出できなくなります。XA トランザクションを開始したユーザーが XID を認識しているため、XA トランザクションの通常のコミットまたはロールバックには影響しません。

権限付与のガイドライン

アカウントには必要な権限のみを付与することをお勧めします。`FILE` 権限と管理権限の付与については十分に注意するようにしてください。

- **FILE** を使用すると、MySQL サーバーがサーバーホスト上で読み取ることができるファイルをデータベーステーブルに読み込むことができます。これにはすべてのユーザーが読み取り可能なすべてのファイルと、サーバーのデータディレクトリ内のファイルが含まれます。そのあと、**SELECT** を使用してそのテーブルにアクセスし、テーブルの内容をクライアントホストに送信することができます。
- **GRANT OPTION** を使用すると、ユーザーは他のユーザーに権限を付与できます。異なる権限を持つ 2 人のユーザーが **GRANT OPTION** 権限を持っていれば、権限を組み合わせることができます。
- **ALTER** を使用すると、テーブルの名前を変更して権限システムを再変換できます。
- **SHUTDOWN** では、サーバーを終了することで、他のユーザーに対するサービスを完全に拒否できます。
- **PROCESS** を使用すると、パスワードを設定または変更するステートメントを含む、現在実行中のステートメントのプレーンテキストを表示できます。
- **SUPER** を使用すると、他のセッションを終了したり、サーバーの動作方法を変更できます。
- **mysql** システムデータベース自体に付与された権限を使用して、パスワードおよびその他のアクセス権限情報を変更できます:
 - パスワードは暗号化されて保管されているため、悪意のあるユーザーは単純にそのパスワードを見てプレーンテキストパスワードを知ることはできません。ただし、**mysql.user** システムテーブルの **authentication_string** カラムへの書き込みアクセス権を持つユーザーは、アカウントパスワードを変更し、そのアカウントを使用して MySQL サーバーに接続できます。
 - **mysql** システムデータベースに付与された **INSERT** または **UPDATE** を使用すると、ユーザーはそれぞれ権限を追加したり、既存の権限を変更できます。
 - **mysql** システムデータベース用の **DROP** を使用すると、ユーザーはリモート権限テーブルまたはデータベース自体を使用できます。

静的権限と動的権限

MySQL では、静的権限および動的権限がサポートされています。

- 静的権限はサーバーに組み込まれています。これらは常にユーザーアカウントに付与でき、登録解除できません。
- 動的権限は、実行時に登録および登録解除できます。これは可用性に影響: 登録されていない動的権限は付与できません。

たとえば、**SELECT** および **INSERT** 権限は静的で常に使用可能ですが、動的権限は、それを実装するコンポーネントが有効になっている場合にのみ使用可能になります。

このセクションの残りの部分では、動的権限が MySQL でどのように機能するかについて説明します。この説明では、「コンポーネント」という用語を使用しますが、プラグインにも同様に適用されます。

注記

サーバー管理者は、動的権限を定義するサーバーコンポーネントに注意する必要があります。MySQL ディストリビューションの場合、動的権限を定義するコンポーネントのドキュメントには、これらの権限が記載されています。

サードパーティコンポーネントでも動的権限を定義できます。管理者は、これらの権限を理解し、サーバー操作が競合または危険にさらされる可能性のあるコンポーネントをインストールしないでください。たとえば、両方で同じ名前の権限が定義されている場合、あるコンポーネントが別のコンポーネントと競合します。コンポーネント開発者は、コンポーネント名に基づいて接頭辞を持つ権限名を選択することで、この発生の可能性を減らすことができます。

サーバーは、登録された動的権限のセットをメモリー内に内部的に保持します。サーバーの停止時に登録解除が発生します。

通常、動的権限を定義するコンポーネントは、インストール時に初期化シーケンス中にそれらを登録します。アンインストール時に、コンポーネントは登録されている動的権限を登録解除しません。(これは現在の演習であり、要件ではありません。つまり、コンポーネントは、登録した権限でいつでも登録解除できますが、できません。)

すでに登録されている動的権限を登録しようとしても、警告やエラーは発生しません。次の一連のステートメントについて考えてみます:

```
INSTALL COMPONENT 'my_component';
UNINSTALL COMPONENT 'my_component';
INSTALL COMPONENT 'my_component';
```

最初の `INSTALL COMPONENT` ステートメントでは、コンポーネント `my_component` によって定義された権限は登録されますが、`UNINSTALL COMPONENT` では登録解除されません。2 番目の `INSTALL COMPONENT` ステートメントでは、登録するコンポーネント権限はすでに登録されていますが、警告やエラーは発生しません。

動的権限はグローバルレベルでのみ適用されます。サーバーは、ユーザーアカウントへの動的権限の現在の割当てに関する情報を `mysql.global_grants` システムテーブルに格納します:

- サーバーは、(`--skip-grant-tables` オプションが指定されていないかぎり) サーバーの起動時に `global_grants` で指定された権限を自動的に登録します。
- `GRANT` および `REVOKE` ステートメントによって、`global_grants` の内容が変更されます。
- `global_grants` にリストされている動的権限割当ては永続的です。サーバーの停止時には削除されません。

例: 次のステートメントは、レプリカに対するレプリケーション (グループレプリケーションを含む) の制御およびシステム変数の変更に必要な権限をユーザー `u1` に付与します:

```
GRANT REPLICATION_SLAVE_ADMIN, GROUP_REPLICATION_ADMIN, BINLOG_ADMIN
ON *.* TO 'u1'@'localhost';
```

付与された動的権限は、`SHOW GRANTS` ステートメントおよび `INFORMATION_SCHEMA.USER_PRIVILEGES` テーブルからの出力に表示されます。

グローバルレベルの `GRANT` および `REVOKE` の場合、静的として認識されない名前付き権限は、現在登録されている動的権限のセットに対してチェックされ、見つかった場合は付与されます。それ以外の場合は、不明な権限識別子を示すエラーが発生します。

`GRANT` および `REVOKE` の場合、グローバルレベルの `ALL [PRIVILEGES]` の意味には、すべての静的グローバル権限と、現在登録されているすべての動的権限が含まれます:

- グローバルレベルの `GRANT ALL` では、すべての静的グローバル権限および現在登録されているすべての動的権限が付与されます。 `GRANT` ステートメントの実行後に登録された動的権限は、どのアカウントにも遡及的に付与されません。
- グローバルレベルの `REVOKE ALL` は、付与されているすべての静的グローバル権限および付与されているすべての動的権限を取り消します。

`FLUSH PRIVILEGES` ステートメントは、動的権限割当てのために `global_grants` テーブルを読み取り、そこで見つかった未登録の権限を登録します。

MySQL Server によって提供される動的権限および MySQL ディストリビューションに含まれるコンポーネントの詳細は、[セクション6.2.2「MySQL で提供される権限」](#) を参照してください。

SUPER から動的権限へのアカウントの移行

MySQL 8.0 では、以前に `SUPER` 権限を必要としていた多くの操作も、より限定された有効範囲の動的権限に関連付けられています。(これらの権限の詳細は、[セクション6.2.2「MySQL で提供される権限」](#) を参照してください。) このような各操作は、`SUPER` ではなく、関連付けられた動的権限を付与することで、アカウントに対して許可できます。この変更により、DBA は `SUPER` の付与を回避し、ユーザー権限を許可されている操作により厳密に調整できるため、セキュリティが向上します。 `SUPER` は非推奨になりました。将来のバージョンの MySQL で削除される予定です。

`SUPER` の削除が発生すると、`SUPER` を付与されたアカウントが適切な動的権限に移行されないかぎり、以前は `SUPER` を必要としていた操作は失敗します。次の手順を使用して、`SUPER` を削除する前にアカウントの準備が整うようにその目標を達成します:

1. 次のクエリーを実行して、`SUPER` が付与されているアカウントを識別します:

```
SELECT GRANTEE FROM INFORMATION_SCHEMA.USER_PRIVILEGES
WHERE PRIVILEGE_TYPE = 'SUPER';
```

2. 前述のクエリーで識別されたアカウントごとに、**SUPER**が必要な操作を決定します。次に、これらの操作に対応する動的権限を付与し、**SUPER**を取り消します。

たとえば、バイナリログのパージおよびシステム変数の変更で 'u1'@'localhost' で **SUPER**が必要な場合、次のステートメントによってアカウントに必要な変更が加えられます:

```
GRANT BINLOG_ADMIN, SYSTEM_VARIABLES_ADMIN ON *.* TO 'u1'@'localhost';
REVOKE SUPER ON *.* FROM 'u1'@'localhost';
```

適用可能なすべてのアカウントを変更した後、最初のステップの **INFORMATION_SCHEMA** クエリーで空の結果セットが生成されます。

6.2.3 付与テーブル

mysql システムデータベースには、ユーザーアカウントおよびそのアカウントが保持する権限に関する情報を含む複数の付与テーブルが含まれています。このセクションでは、これらのテーブルについて説明します。システムデータベース内の他のテーブルの詳細は、[セクション5.3「mysql システムスキーマ」](#)を参照してください。

ここでは付与テーブルの基本構造と、サーバーがクライアントと対話するときに付与テーブルの内容をどのように使用するかについて説明します。ただし、通常は付与テーブルを直接変更しません。**CREATE USER**、**GRANT**、**REVOKE**などのアカウント管理ステートメントを使用してアカウントを設定し、各アカウントで使用可能な権限を制御すると、変更が間接的に行われます。[セクション13.7.1「アカウント管理ステートメント」](#)を参照してください。このようなステートメントを使用してアカウント操作を実行すると、サーバーはユーザーの代わりに付与テーブルを変更します。

注記

INSERT、**UPDATE**、**DELETE**などのステートメントを使用して付与テーブルを直接変更することはお薦めできません。独自のリスクで実行してください。これらの変更の結果として誤った形式となった行を、サーバーは随意で無視します。

付与テーブルを変更する操作の場合、サーバーはテーブルが予期された構造を持っているかどうかをチェックし、持っていない場合はエラーを生成します。テーブルを必要な構造に更新するには、MySQLのアップグレード手順を実行します。[セクション2.11「MySQLのアップグレード」](#)を参照してください。

- [付与テーブルの概要](#)
- [ユーザーおよび DB 付与テーブル](#)
- [tables_priv および columns_priv の付与テーブル](#)
- [procs_priv 付与テーブル](#)
- [proxies_priv 付与テーブル](#)
- [global_grants 付与テーブル](#)
- [default_roles 付与テーブル](#)
- [role_edges 付与テーブル](#)
- [password_history 付与テーブル](#)
- [付与テーブルのスコープカラムのプロパティ](#)
- [テーブル権限の付与のカラムプロパティ](#)
- [テーブル同時実行性の付与](#)

付与テーブルの概要

次の **mysql** データベーステーブルには付与情報が格納されています。

- **user**: ユーザーアカウント、静的グローバル権限およびその他の非権限カラム。

- `global_grants`: 動的グローバル権限。
- `db`: Database-level privileges.
- `tables_priv`: Table-level privileges.
- `columns_priv`: Column-level privileges.
- `procs_priv`: ストアドプロシージャおよびファンクション権限。
- `proxies_priv`: Proxy-user privileges.
- `default_roles`: デフォルトのユーザーロール。
- `role_edges`: ロールサブグラフのエッジ。
- `password_history`: パスワード変更履歴。

静的グローバル権限と動的グローバル権限の違いの詳細は、[静的権限と動的権限](#) を参照してください。)

MySQL 8.0 では、付与テーブルは `InnoDB` ストレージエンジンを使用し、トランザクションです。MySQL 8.0 より前は、付与テーブルは `MyISAM` ストレージエンジンを使用しており、非トランザクションでした。付与テーブルストレージエンジンのこの変更により、`CREATE USER` や `GRANT` などのアカウント管理ステートメントの動作に付随する変更が可能になります。以前は、複数のユーザーを指定したアカウント管理ステートメントは、一部のユーザーでは成功し、他のユーザーでは失敗していました。現在は、各ステートメントはトランザクションに対応し、指定されたすべてのユーザーに対して成功するか、エラーが発生した場合はロールバックされて何の影響も与えなくなりました。

各付与テーブルにはスコープカラムと権限カラムがあります。

- 有効範囲カラムは、テーブルの各行の有効範囲、つまり行が適用されるコンテキストを決定します。たとえば、`Host` と `User` の値が `'h1.example.net'` および `'bob'` の `user` テーブルの行は、ユーザー名 `bob` を指定するクライアントによってホスト `h1.example.net` からサーバーに対して行われる認証接続に適用されます。同様に、`Host`、`User` および `Db` カラムの `'h1.example.net'`、`'bob'` および `'reports'` の値を含む `db` テーブルの行は、`bob` がホスト `h1.example.net` から接続して `reports` データベースにアクセスする場合に適用されます。`tables_priv` テーブルおよび `columns_priv` テーブルには、それぞれの行に適用されるテーブルまたはテーブルとカラムの組み合わせを示すスコープカラムがあります。`procs_priv` スコープカラムは、それぞれの行に適用されるストアドルーチンを示します。
- 権限カラムは、テーブルの行で付与される権限、つまり実行が許可される操作を示します。サーバーは、さまざまな付与テーブル内の情報を結合して、ユーザー権限の完全な説明を形成します。[セクション 6.2.7 「アクセス制御、ステージ 2: リクエストの確認」](#) では、このルールについて説明します。

また、権限付与テーブルには、スコープまたは権限評価以外の目的で使用されるカラムが含まれる場合があります。

サーバーは次の方法で付与テーブルを使用します。

- `user` テーブルのスコープカラムは、入接続を拒否または許可するかを決定します。許可された接続の場合、`user` テーブルで付与された権限は、ユーザーの静的グローバル権限を示します。このテーブルで付与される権限は、サーバー上の all データベースに適用されます。

注意

静的グローバル権限はすべてのデータベースに対する権限とみなされるため、静的グローバル権限を使用すると、ユーザーは、部分的な取消しによってデータベースレベルで制限されているデータベースを除き、`SHOW DATABASES` を使用するか、`INFORMATION_SCHEMA` の `SCHEMATA` テーブルを調べることで、すべてのデータベース名を表示できます。

- `global_grants` テーブルには、ユーザーアカウントへの動的グローバル権限の現在の割当てがリストされます。行ごとに、有効範囲カラムによって、権限カラムで指定された権限を持つユーザーが決定されます。
- `db` テーブルのスコープカラムは、どのユーザーがどのホストからどのデータベースにアクセスできるかを決定します。権限カラムによって、許可される操作が決まります。データベースレベルで付与される権限は、データベースのほかテーブルやストアドプログラムなどのデータベース内のすべてのオブジェクトに適用されます。

- `tables_priv` および `columns_priv` テーブルは `db` テーブルと似ていますが、これらはさらに粒度が細かく、データベースレベルでなくテーブルレベルおよびカラムレベルに適用されます。テーブルレベルで付与される権限は、テーブルおよびそのすべてのカラムに適用されます。カラムレベルで付与される権限は、特定のカラムにのみ適用されます。
- `procs_priv` テーブルはストアドルーチン (ストアドプロシージャとストアドファンクション) に適用されます。ルーチンレベルで付与される権限は、単一のプロシージャまたは関数にのみ適用されます。
- `proxies_priv` テーブルには、他のユーザーのプロキシとして機能できるユーザーと、ユーザーが他のユーザーに `PROXY` 権限を付与できるかどうかを示されます。
- `default_roles` テーブルと `role_edges` テーブルには、ロール関係に関する情報が含まれています。
- `password_history` テーブルでは、パスワードの再利用に関する制限を有効にするために、以前に選択したパスワードが保持されます。 [セクション6.2.15「パスワード管理」](#) を参照してください。

サーバーは、起動時に付与テーブルの内容をメモリーに読み取ります。 `FLUSH PRIVILEGES` ステートメントを発行するか、 `mysqladmin flush-privileges` または `mysqladmin reload` コマンドを実行することによって、テーブルをリロードするようサーバーに指示することができます。付与テーブルへの変更は、 [セクション6.2.13「権限変更が有効化される時期」](#) で示すように反映されます。

アカウントを変更する場合は、変更が意図したとおりの効果を持つことを確認することをお勧めします。特定のアカウントの権限を確認するには、 `SHOW GRANTS` ステートメントを使用します。たとえば、ユーザー名およびホスト名の値がそれぞれ `bob` および `pc84.example.com` のアカウントに付与された権限を調べるには、次のステートメントを使用します。

```
SHOW GRANTS FOR 'bob'@'pc84.example.com';
```

アカウントの非特権プロパティーを表示するには、 `SHOW CREATE USER` を使用します:

```
SHOW CREATE USER 'bob'@'pc84.example.com';
```

ユーザーおよび DB 付与テーブル

サーバーは `mysql` データベース内の `user` および `db` テーブルを、アクセス制御のステージ 1 とステージ 2 の両方で使用します ([セクション6.2「アクセス制御とアカウント管理」](#) を参照してください)。 `user` と `db` のテーブルのカラムをここで示します。

表 6.4 `user` テーブルおよび `db` テーブルのカラム

テーブル名	<code>user</code>	<code>db</code>
スコープカラム	<code>Host</code>	<code>Host</code>
	<code>User</code>	<code>Db</code>
権限カラム	<code>Select_priv</code>	<code>Select_priv</code>
	<code>Insert_priv</code>	<code>Insert_priv</code>
	<code>Update_priv</code>	<code>Update_priv</code>
	<code>Delete_priv</code>	<code>Delete_priv</code>
	<code>Index_priv</code>	<code>Index_priv</code>
	<code>Alter_priv</code>	<code>Alter_priv</code>
	<code>Create_priv</code>	<code>Create_priv</code>
	<code>Drop_priv</code>	<code>Drop_priv</code>
	<code>Grant_priv</code>	<code>Grant_priv</code>
	<code>Create_view_priv</code>	<code>Create_view_priv</code>
	<code>Show_view_priv</code>	<code>Show_view_priv</code>
	<code>Create_routine_priv</code>	<code>Create_routine_priv</code>

付与テーブル

テーブル名	user	db
	Alter_routine_priv	Alter_routine_priv
	Execute_priv	Execute_priv
	Trigger_priv	Trigger_priv
	Event_priv	Event_priv
	Create_tmp_table_priv	Create_tmp_table_priv
	Lock_tables_priv	Lock_tables_priv
	References_priv	References_priv
	Reload_priv	
	Shutdown_priv	
	Process_priv	
	File_priv	
	Show_db_priv	
	Super_priv	
	Repl_slave_priv	
	Repl_client_priv	
	Create_user_priv	
	Create_tablespace_priv	
	Create_role_priv	
	Drop_role_priv	
セキュリティーカラム	ssl_type	
	ssl_cipher	
	x509_issuer	
	x509_subject	
	plugin	
	authentication_string	
	password_expired	
	password_last_changed	
	password_lifetime	
	account_locked	
	Password_reuse_history	
	Password_reuse_time	
	Password_require_current	
	User_attributes	
リソース制御カラム	max_questions	
	max_updates	
	max_connections	
	max_user_connections	

user テーブルの plugin カラムおよび authentication_string カラムには、認証プラグインおよび資格証明情報が格納されます。

サーバーは、アカウント行の plugin カラムで指定されたプラグインを使用して、アカウントの接続試行を認証します。

`plugin` カラムは空にできません。起動時および実行時に、`FLUSH PRIVILEGES` が実行されると、サーバーは `user` テーブルの行をチェックします。 `plugin` カラムが空の行の場合、サーバーは次の形式のエラーログに警告を書き込みます:

```
[Warning] User entry 'user_name'@'host_name' has an empty plugin value. The user will be ignored and no one can login with this user anymore.
```

プラグインがないアカウントにプラグインを割り当てるには、`ALTER USER` ステートメントを使用します。

`password_expired` カラムを使用すると、DBA はアカウントパスワードを期限切れにでき、ユーザーはパスワードをリセットする必要があります。デフォルトの `password_expired` 値は 'N' ですが、`ALTER USER` ステートメントを使用して 'Y' に設定できます。アカウントパスワードが期限切れになると、ユーザーが新しいアカウントパスワードを確立するために `ALTER USER` ステートメントを発行するまで、サーバーへの後続の接続でアカウントによって実行されるすべての操作でエラーが発生します。

注記

期限切れのパスワードは、現在の値に設定することで「reset」で使用できますが、適切なポリシーとして、別のパスワードを選択することをお勧めします。DBA は、適切なパスワード再利用ポリシーを確立することで、非キューを強制できます。パスワード再利用ポリシーを参照してください。

`password_last_changed` は、パスワードが最後に変更された日時を示す `TIMESTAMP` カラムです。この値は、MySQL 組み込み認証プラグイン (`mysql_native_password`、`sha256_password` または `caching_sha2_password`) を使用するアカウントに対してのみ `NULL` 以外です。他のアカウント (外部認証システムを使用して認証されたアカウントなど) の場合、値は `NULL` です。

`password_last_changed` は、`CREATE USER`、`ALTER USER` および `SET PASSWORD` ステートメント、およびアカウントの作成やアカウントパスワードの変更を行う `GRANT` ステートメントによって更新されます。

`password_lifetime` は、アカウントパスワードの存続期間を日数で示します。パスワードが存続期間を過ぎた場合 (`password_last_changed` カラムを使用して評価)、クライアントがアカウントを使用して接続すると、サーバーはパスワードの有効期限が切れたとみなします。ゼロより大きい `N` の値は、パスワードを `N` 日ごとに変更する必要があることを意味します。値 `0` を指定すると、自動パスワード有効期限が無効になります。値が `NULL` (デフォルト) の場合は、`default_password_lifetime` システム変数で定義されているグローバル有効期限ポリシーが適用されます。

`account_locked` は、アカウントがロックされているかどうかを示します (セクション6.2.19「アカウントロック」を参照)。

`Password_reuse_history` は、アカウントの `PASSWORD HISTORY` オプションの値、またはデフォルト履歴の `NULL` の値です。

`Password_reuse_time` は、アカウントの `PASSWORD REUSE INTERVAL` オプション、またはデフォルト間隔の `NULL` の値です。

`Password_require_current` (MySQL 8.0.13 で追加) は、次のテーブルに示すように、アカウントの `PASSWORD REQUIRE` オプションの値に対応します。

表 6.5 許可される `Password_require_current` 値

<code>Password_require_current</code> 値	対応する <code>PASSWORD REQUIRE</code> オプション
'Y'	<code>PASSWORD REQUIRE CURRENT</code>
'N'	<code>PASSWORD REQUIRE CURRENT OPTIONAL</code>
<code>NULL</code>	<code>PASSWORD REQUIRE CURRENT DEFAULT</code>

`User_attributes` (MySQL 8.0.14 で追加) は、他のカラムに格納されていないアカウント属性を格納する JSON 形式のカラムです:

- `additional_password`: セカンダリパスワード (存在する場合)。デュアルパスワードのサポートを参照してください。

- **Restrictions**: 制限リスト (ある場合)。制限は、部分失効操作によって追加されます。属性値は、それぞれ制限されたデータベースの名前とそれに適用可能な制限を示す **Database** および **Restrictions** キーを持つ要素の配列です ([セクション6.2.12「部分取消しを使用した権限の制限」](#)を参照)。
- **Password_locking**: 失敗したログインロックおよび一時アカウントロックの条件 (存在する場合)([失敗したログインロックと一時アカウントロック](#)を参照)。**Password_locking** 属性は、**CREATE USER** および **ALTER USER** ステートメントの **FAILED_LOGIN_ATTEMPTS** および **PASSWORD_LOCK_TIME** オプションに従って更新されます。属性値は、アカウントに指定されているようなオプションの値を示す **failed_login_attempts** および **password_lock_time_days** キーを含むハッシュです。キーがない場合、その値は暗黙的に 0 になります。キー値が暗黙的または明示的に 0 の場合、対応する機能は無効になります。この属性は、MySQL 8.0.19 で追加されました。

適用される属性がない場合、**User_attributes** は **NULL** です。

例: セカンダリパスワードを持ち、部分的に取り消されたデータベース権限を持つアカウントの列挙値には、**additional_password** および **Restrictions** 属性が含まれます:

```
mysql> SELECT User_attributes FROM mysql.User WHERE User = 'u\G
***** 1. row *****
User_attributes: {"Restrictions":
  [{"Database": "mysql", "Privileges": ["SELECT"]}],
  "additional_password": "hashed_credentials"}
```

存在する属性を確認するには、**JSON_KEYS()** 関数を使用します:

```
SELECT User, Host, JSON_KEYS(User_attributes)
FROM mysql.user WHERE User_attributes IS NOT NULL;
```

Restrictions などの特定の属性を抽出するには、次のようにします:

```
SELECT User, Host, User_attributes->'$.Restrictions'
FROM mysql.user WHERE User_attributes->'$.Restrictions' <> '';
```

tables_priv および columns_priv の付与テーブル

アクセス制御の第 2 段階で、サーバーはリクエスト検証を実行して、各クライアントが発行するリクエストごとに十分な権限を持っていることを確認します。**user** および **db** 付与テーブルに加えて、テーブルに関するリクエストの場合、サーバーは **tables_priv** および **columns_priv** テーブルを参照することもあります。後者のテーブルは、テーブルレベルおよびカラムレベルでの細かい権限制御を提供します。これらには、次の表に示すカラムがあります。

表 6.6 tables_priv テーブルおよび columns_priv テーブルのカラム

テーブル名	tables_priv	columns_priv
スコープカラム	Host	Host
	Db	Db
	User	User
	Table_name	Table_name
		Column_name
権限カラム	Table_priv	Column_priv
	Column_priv	
その他のカラム	Timestamp	Timestamp
	Grantor	

Timestamp カラムと **Grantor** カラムは、それぞれ現在のタイムスタンプと **CURRENT_USER** 値に設定されますが、それ以外の場合は使用されません。

procs_priv 付与テーブル

ストアドルーチンに関するリクエストを検証するために、サーバーは **procs_priv** テーブルを参照することがあり、このテーブルには次の表に示すカラムがあります。

表 6.7 procs_priv テーブルのカラム

テーブル名	procs_priv
スコープカラム	Host
	Db
	User
	Routine_name
	Routine_type
権限カラム	Proc_priv
その他のカラム	Timestamp
	Grantor

`Routine_type` カラムは、'FUNCTION' または 'PROCEDURE' の値を持つ ENUM カラムであり、その行が示すルーチンのタイプを指します。このカラムにより、同じ名前を持つ関数とプロシージャに別々に権限を付与することができます。

`Timestamp` カラムおよび `Grantor` カラムは未使用です。

proxies_priv 付与テーブル

`proxies_priv` テーブルには、プロキシアカウントに関する情報が記録されます。これには次のカラムがあります。

- `Host`, `User`: プロキシアカウント (プロキシアカウントに対する PROXY 権限を持つアカウント)。
- `Proxied_host`, `Proxied_user`: プロキシされたアカウント。
- `Grantor`, `Timestamp`: Unused.
- `With_grant`: プロキシアカウントが PROXY 権限を他のアカウントに付与できるかどうか。

アカウントが PROXY 権限を他のアカウントに付与できるようにするには、`With_grant` が 1 に設定され、`Proxied_host` および `Proxied_user` が権限を付与できるアカウントを示す行が `proxies_priv` テーブルに含まれている必要があります。たとえば、MySQL のインストール時に作成された 'root'@'localhost' アカウントの `proxies_priv` テーブルには、"@" の PROXY 権限 (すべてのユーザーおよびすべてのホスト) の付与を可能にする行があります。これにより、`root` はプロキシユーザーを設定したり、プロキシユーザーを設定するための権限をほかのアカウントに委任したりできます。 [セクション6.2.18「プロキシユーザー」](#) を参照してください。

global_grants 付与テーブル

`global_grants` テーブルには、ユーザーアカウントへの動的グローバル権限の現在の割当てがリストされます。テーブルには次のカラムがあります。

- `USER`, `HOST`: 権限が付与されるアカウントのユーザー名およびホスト名。
- `PRIV`: 権限名。
- `WITH_GRANT_OPTION`: アカウントが他のアカウントに権限を付与できるかどうか。

default_roles 付与テーブル

`default_roles` テーブルには、デフォルトのユーザーロールがリストされます。これには次のカラムがあります。

- `HOST`, `USER`: デフォルトロールが適用されるアカウントまたはロール。
- `DEFAULT_ROLE_HOST`, `DEFAULT_ROLE_USER`: デフォルトロール。

role_edges 付与テーブル

`role_edges` テーブルには、ロールサブグラフのエッジがリストされます。これには次のカラムがあります。

- `FROM_HOST`, `FROM_USER`: ロールが付与されているアカウント。
- `TO_HOST`, `TO_USER`: アカウントに付与されるロール。
- `WITH_ADMIN_OPTION`: アカウントが `WITH ADMIN OPTION` を使用して他のアカウントにロールを付与したり、他のアカウントからロールを取り消すことができるかどうか。

password_history 付与テーブル

`password_history` テーブルには、パスワード変更に関する情報が含まれています。これには次のカラムがあります。

- `Host`, `User`: パスワード変更が発生したアカウント。
- `Password_timestamp`: パスワード変更が発生した時刻。
- `Password`: 新しいパスワードハッシュ値。

`password_history` テーブルには、MySQL がアカウントパスワード履歴の長さで再利用間隔の両方に対してチェックを実行できるように、アカウントごとに空でない十分な数のパスワードが蓄積されます。両方の制限外のエントリの自動ブルーニングは、パスワード変更の試行が発生したときに行われます。

注記

空のパスワードはパスワード履歴にカウントされず、いつでも再利用される可能性があります。

アカウントの名前が変更されると、一致するようにそのエントリの名前が変更されます。アカウントが削除されるか、その認証プラグインが変更されると、そのエントリは削除されます。

付与テーブルのスコープカラムのプロパティ

付与テーブルのスコープカラムには文字列が格納されています。それぞれのデフォルト値は空の文字列です。次のテーブルに、各カラムで許可される文字数を示します。

表 6.8 付与テーブルの有効範囲カラム長

カラム名	最大許容文字数
<code>Host</code> , <code>Proxied_host</code>	255 (MySQL 8.0.17 より前の 60)
<code>User</code> , <code>Proxied_user</code>	32
<code>Db</code>	64
<code>Table_name</code>	64
<code>Column_name</code>	64
<code>Routine_name</code>	64

`Host` および `Proxied_host` の値は、権限付与テーブルに格納される前に小文字に変換されます。

アクセスチェックの目的で、`User`, `Proxied_user`, `authentication_string`, `Db` と `Table_name` の値の比較では大/小文字が区別されます。`Host`, `Proxied_host`, `Column_name` と `Routine_name` の値の比較では、大/小文字は区別されません。

テーブル権限の付与のカラムプロパティ

`user` テーブルおよび `db` テーブルでは、`ENUM('N','Y') DEFAULT 'N'` として宣言された個別のカラムに各権限がリストされます。つまり、各権限は無効または有効にすることができ、デフォルトは無効です。

`tables_priv`, `columns_priv` および `procs_priv` の各テーブルでは、権限カラムを `SET` カラムとして宣言します。これらのカラムの値は、テーブルによって制御されるあらゆる組み合わせの権限を含むことができます。カラム値にリストされている権限のみが入力されます。

表 6.9 Set タイプ権限のカラム値

テーブル名	カラム名	可能な Set 要素
tables_priv	Table_priv	'Select', 'Insert', 'Update', 'Delete', 'Create', 'Drop', 'Grant', 'References', 'Index', 'Alter', 'Create View', 'Show view', 'Trigger'
tables_priv	Column_priv	'Select', 'Insert', 'Update', 'References'
columns_priv	Column_priv	'Select', 'Insert', 'Update', 'References'
procs_priv	Proc_priv	'Execute', 'Alter Routine', 'Grant'

RELOAD、SHUTDOWN、SYSTEM_VARIABLES_ADMIN などの管理権限を指定するのは、`user` テーブルおよび `global_grants` テーブルのみです。管理操作はサーバー自体での操作であって、データベース固有でないため、これらの権限をほかの付与テーブルにリストする理由はありません。したがって、サーバーは、ユーザーが管理操作を実行できるかどうかを判断するために、`user` および `global_grants` テーブルのみを参照する必要があります。

FILE 権限も、`user` テーブルでのみ指定されます。このような管理権限ではありませんが、サーバーホスト上のファイルの読取りまたは書き込みを行うユーザー権限は、アクセスされるデータベースから独立しています。

テーブル同時実行性の付与

MySQL 8.0.22 では、MySQL 付与テーブルに対する同時 DML 操作および DDL 操作を許可するために、MySQL 付与テーブルに対して以前に行ロックを取得した読取り操作は、非ロック読取りとして実行されます。MySQL 付与テーブルで非ロック読取りとして実行される操作には、次のものがあります：

- 任意のトランザクション分離レベルを使用して、結合リストおよびサブクエリー (`SELECT ... FOR SHARE` ステートメントを含む) を介して付与テーブルからデータを読み取る `SELECT` ステートメントおよびその他の読取り専用ステートメント。
- 任意のトランザクション分離レベルを使用して (結合リストまたはサブクエリーを介して) 付与テーブルからデータを読み取り、変更を伴わない DML 操作。

付与テーブルからデータを読み取る時に行ロックを取得しなくなったステートメントは、ステートメントベースレプリケーションの使用中に実行された場合に警告を報告します。

使用時 `-binlog_format=mixed`、付与テーブルからデータを読み取る DML 操作は、混合モードのレプリケーションで操作を安全にするために、行イベントとしてバイナリログに書き込まれます。

付与テーブルからデータを読み取る `SELECT ... FOR SHARE` ステートメントは警告を報告します。`FOR SHARE` 句を使用する場合、読取りロックは付与テーブルではサポートされません。

付与テーブルからデータを読み取り、`SERIALIZABLE` 分離レベルを使用して実行される DML 操作では、警告がレポートされます。`SERIALIZABLE` 分離レベルの使用時に通常取得される読取りロックは、付与テーブルではサポートされていません。

6.2.4 アカウント名の指定

MySQL アカウント名は、ユーザー名とホスト名で構成されます。これにより、異なるホストから接続する同じユーザー名を持つユーザーに対して個別のアカウントを作成できます。このセクションでは、特別な値やワイルドカードルールなど、アカウント名の構文について説明します。

ほとんどの点で、アカウント名は MySQL のロール名と似ていますが、[セクション6.2.5「ロール名の指定」](#) で説明されているいくつかの違いがあります。

アカウント名は、`CREATE USER`、`GRANT`、`SET PASSWORD` などの SQL ステートメントに表示され、次のルールに従います：

- アカウント名の構文は `'user_name'@'host_name'` です。
- `@'host_name'` 部分はオプションです。ユーザー名のみで構成されるアカウント名は、`'user_name'@'%'` と同等です。たとえば、`'me'` は `'me'@'%'` と同等です。

- ユーザー名およびホスト名は、それらが引用符なしの識別子として有効な場合、引用する必要はありません。 `user_name` 文字列に特殊文字 (スペースや `-` など) が含まれている場合、または `host_name` 文字列に特殊文字やワイルドカード文字 (`.` や `%` など) が含まれている場合は、引用符を使用する必要があります。たとえば、アカウント名 `'test-user'@'% .com'` では、ユーザー名とホスト名の両方の部分に引用符が必要です。
- バックティック (```)、一重引用符 (`'`) または二重引用符 (`"`) を使用して、ユーザー名とホスト名を識別子または文字列として引用符で囲みます。文字列および識別子として引用符で囲む方法のガイドラインについては、[セクション 9.1.1 「文字列リテラル」](#) および [セクション 9.2 「スキーマオブジェクト名」](#) を参照してください。
- ユーザー名およびホスト名の部分が引用符で囲まれる場合、別々に囲んでください。つまり、`'me@localhost'` ではなく `'me'@'localhost'` と記述します。後者は、実際には `'me@localhost'@'%'` と同等です。
- `CURRENT_USER` または `CURRENT_USER()` 関数への参照は、現行クライアントのユーザー名およびホスト名の文字を指定することと同等です。

MySQL では、ユーザー名とホスト名の部分に別々のカラムを使用して、アカウント名が `mysql` システムデータベースの付与テーブルに格納されます:

- `user` テーブルにはアカウントごとに 1 行が格納されます。 `User` および `Host` カラムはユーザー名およびホスト名を格納します。このテーブルには、アカウントが持つグローバル権限も指定されます。
- ほかの付与テーブルには、データベースおよびデータベース内のオブジェクトに対してアカウントが持つ権限が示されます。これらのテーブルにはアカウント名を格納するための `User` および `Host` カラムがあります。これらのテーブルの各行は、同じ `User` および `Host` の値を持つ `user` テーブル内のアカウントに関連付けられています。
- アクセスチェックの目的で、ユーザー値の比較では大/小文字が区別されます。ホスト値の比較では、大文字と小文字は区別されません。

付与テーブルに格納されているユーザー名とホスト名のプロパティ (最大長など) の詳細は、[付与テーブルのスコープカラムのプロパティ](#) を参照してください。

ユーザー名およびホスト名は、次に示すような特殊な値が使用されたりワイルドカード規則が適用されたりします。

アカウント名のユーザー名部分は、受信接続試行のユーザー名と文字どおり一致する空白以外の値、または任意のユーザー名と一致する空白の値 (空の文字列) です。ブランクのユーザー名を持つアカウントは匿名ユーザーです。SQL ステートメントで匿名ユーザーを指定するには、`"@'localhost'"` のように引用符で囲んだ空のユーザー名部分を使用します。

アカウント名のホスト名部分は多くの形式を持つことができ、ワイルドカードが許可されます。

- ホスト値はホスト名または IP アドレス (IPv4 または IPv6) とすることができ、`'localhost'` という名前はローカルホストを示します。IP アドレス `'127.0.0.1'` は IPv4 ループバックインタフェースを示します。IP アドレス `:::1'` は、IPv6 ループバックインタフェースを示します。
- `%` および `_` のワイルドカード文字は、ホスト名または IP アドレスの値に使用できます。これらは `LIKE` 演算子で実行されるパターンマッチング演算と同じ意味を持ちます。たとえば、`'%'` のホスト値は任意のホスト名に一致しますが、`'%.mysql.com'` の値は `mysql.com` ドメインの任意のホストに一致します。`'198.51.100.%'` は、1 つの 98.51.100 クラス C ネットワーク内の任意のホストに一致します。

IP ワイルドカード値はホスト値で許可されているため (たとえば、サブネット上のすべてのホストに一致する `'198.51.100.%'` など)、ホストに `198.51.100.somewhere.com` という名前を付けてこの機能を利用しようとする可能性があります。このような試行を行うために、MySQL では、数字とドットで始まるホスト名の照合は実行されません。たとえば、ホストの名前が `1.2.example.com` の場合、その名前はアカウント名のホスト部分と一致しません。IP ワイルドカード値は、ホスト名でなく IP アドレスのみと一致することができます。

- IPv4 アドレスとして指定されたホスト値の場合、ネットマスクを指定して、ネットワーク番号に使用するアドレスビット数を示すことができます。ネットマスク表記は IPv6 アドレスについては使用できません。

構文は `host_ip/netmask` です。例:

```
CREATE USER 'david'@'198.51.100.0/255.255.255.0';
```

これにより `david` は、次の条件が `true` となる IP アドレス `client_ip` を持つすべてのクライアントホストから接続できます。


```
client_ip & netmask = host_ip
```

つまり、次のような `CREATE USER` ステートメントがあるとして。

```
client_ip & 255.255.255.0 = 198.51.100.0
```

この条件を満たす IP アドレスの範囲は、`198.51.100.0` から `198.51.100.255` です。

ネットマスクは通常、1 に設定されたビットから始まり、0 に設定されたビットが続きます。例:

- `198.0.0.0/255.0.0.0`: 198 のクラス A ネットワーク上のすべてのホスト
- `198.51.0.0/255.255.0.0`: 198.51 のクラス B ネットワーク上のすべてのホスト
- `198.51.100.0/255.255.255.0`: 198.51.100 のクラス C ネットワーク上のすべてのホスト
- `198.51.100.1`: この特定の IP アドレスを持つホストのみ
- MySQL 8.0.23 では、IPv4 アドレスとして指定されたホスト値は、`198.51.100.44/24` などの CIDR 表記法を使用して書き込むことができます。

サーバーは、クライアントホスト名または IP アドレス用のシステム DNS リゾルバによって返された値を使用して、アカウント名のホスト値の突き合わせをクライアントホストに対して実行します。アカウントのホスト値がネットマスク表記法を使用して指定されている場合を除き、サーバーは、IP アドレスとして指定されたアカウントのホスト値に対しても、この比較を文字列一致として実行します。つまり、DNS によって使用されるのと同じ形式でアカウントホスト値を指定しなければならないということを意味します。留意すべき問題の例を次に示します。

- ローカルネットワーク上のホストの完全修飾名が `host1.example.com` だとします。DNS がこのホストの名前参照を `host1.example.com` として返す場合、アカウントホスト値にこの名前を使用します。DNS が `host1` のみを返す場合は、かわりに `host1` を使用します。
- DNS が特定のホストの IP アドレスを `198.51.100.2` として返す場合、`198.51.100.2` のアカウントホスト値と一致しますが、`198.051.100.2` とは一致しません。同様に、`198.51.100.%` のようなアカウントホストパターンに一致しますが、`198.051.100.%` には一致しません。

このような問題を回避するには、DNS がホスト名とアドレスを返す形式を確認することをお勧めします。MySQL アカウント名には、同じ形式の値を使用します。

6.2.5 ロール名の指定

MySQL のロール名は、権限の名前付きコレクションであるロールを参照します。ロールの使用例は、[セクション 6.2.10 「ロールの使用」](#) を参照してください。

ロール名の構文とセマンティクスは、アカウント名と似ています。[セクション 6.2.4 「アカウント名の指定」](#) を参照してください。付与テーブルに格納されているように、[付与テーブルのスコープカラムのプロパティ](#) で説明されているアカウント名と同じプロパティを持ちます。

ロール名は、次の点でアカウント名と異なります:

- ロール名のユーザー部分は空白にできません。したがって、「匿名ユーザー」の概念に類似した「匿名ロール」はありません。
- アカウント名の場合、ロール名のホスト部分を省略すると、`'%'` のホスト部分になります。ただし、アカウント名の `'%'` とは異なり、ロール名の `'%'` のホスト部分にはワイルドカードプロパティはありません。たとえば、ロール名として使用される名前 `'me'@'%'` の場合、ホスト部分 (`'%'`) はリテラル値であり、「任意のホスト」一致プロパティはありません。
- ロール名のホスト部分のネットマスク表記に意味はありません。
- アカウント名は、複数のコンテキストで `CURRENT_USER()` にすることができます。ロール名がではありません。

`mysql.user` システムテーブルの行は、アカウントとロールの両方として機能できます。この場合、プロパティに一致する特別なユーザー名またはホスト名は、名前がロール名として使用されるコンテキストには適用されません。た

たとえば、次のステートメントは、`myrole` のユーザー部分およびホスト名を持つすべてのロールを使用して現在のセッションロールを設定することを想定して実行できません:

```
SET ROLE 'myrole'@'%';
```

かわりに、このステートメントはセッションのアクティブロールを `'myrole'@'%'` という名前のロールに設定します。

このため、多くの場合、ロール名はユーザー名部分のみを使用して指定され、ホスト名部分は暗黙的に `'%'` になります。`'%'` 以外のホスト部分でロールを指定すると、特定のホストからの接続を許可されたユーザーアカウントとしてロールとして機能する名前を作成する場合に役立ちます。

6.2.6 アクセス制御、ステージ 1: 接続の検証

MySQL サーバーに接続しようとする時、サーバーは次の条件に基づいて接続を受け入れるか拒否します:

- アイデンティティ、および適切な資格証明を指定して検証できるかどうか。
- アカウントがロックされているかロック解除されているか。

サーバーはまず資格証明をチェックし、次にアカウントのロック状態をチェックします。いずれかのステップで障害が発生すると、サーバーはユーザーへのアクセスを完全に拒否します。それ以外の場合、サーバーは接続を受け入れてステージ 2 に進み、リクエストを待機します。

サーバーは、`user` テーブルのカラムを使用してアイデンティティと資格証明のチェックを実行し、次の条件を満たしている場合にのみ接続を受け入れます:

- クライアントホスト名およびユーザー名は、一部の `user` テーブルの行の `Host` および `User` のカラムと一致します。`Host` および `User` の許容値を制御するルールについては、[セクション 6.2.4 「アカウント名の指定」](#) を参照してください。
- クライアントは、`authentication_string` カラムで示されるように、行で指定された資格証明 (パスワードなど) を提供します。資格証明は、`plugin` カラムで指定された認証プラグインを使用して解釈されます。
- 行は、アカウントがロック解除されていることを示します。ロック状態は `account_locked` カラムに記録されます。このカラムの値は `'N'` である必要があります。アカウントのロックは、`CREATE USER` ステートメントまたは `ALTER USER` ステートメントを使用して設定または変更できます。

ユーザーの ID は 2 つの部分の情報に基づきます。

- MySQL ユーザー名
- 接続元のクライアントホスト

`User` カラム値が空白でない場合、入接続のユーザー名は正確に一致する必要があります。`User` 値が空白の場合、これはすべてのユーザー名と一致します。入接続と一致する `user` テーブル行のユーザー名が空白である場合、ユーザーはクライアントが実際に指定した名前を持つユーザーでなく、名前のない匿名ユーザーとみなされます。つまり、接続期間中の (つまりステージ 2 での) 今後のすべてのアクセスチェックで空白のユーザー名が使用されることを意味します。

`authentication_string` カラムは空白にできます。これはワイルドカードではなく、あらゆるパスワードが一致するという意味ではありません。これは、ユーザーはパスワードを指定せずに接続しなければならないことを意味します。クライアントを認証するプラグインによって実装される認証方法は、`authentication_string` カラムのパスワードを使用する場合と使用しない場合があります。この場合、MySQL サーバーへの認証を行う際に、外部パスワードも使用される可能性があります。

`user` テーブルの `authentication_string` カラムに格納されている空白以外のパスワード値は暗号化されます。MySQL では、パスワードはクリアテキストとして格納されず、すべてのユーザーに表示されます。かわりに、接続しようとしているユーザーが指定したパスワードは暗号化されます (アカウント認証プラグインによって実装されたパスワードハッシュ方式を使用)。そのあと、暗号化パスワードは、接続プロセス中にパスワードが正しいかどうかをチェックするときに使用されます。これは、暗号化パスワードが接続を介してやりとりされずに実行されます。[セクション 6.2.1 「アカウントのユーザー名とパスワード」](#) を参照してください。

MySQL から見ると、暗号化パスワードが実際のパスワードであるため、暗号化パスワードへのアクセス権限をすべてのユーザーに付与しないようにしてください。特に、「mysql システムデータベース内のテーブルへの読取りアクセス権を非管理ユーザーに付与しない」です。

次のテーブルに、`user` テーブルの `User` 値と `Host` 値の様々な組合せが着信接続にどのように適用されるかを示します。

User 値	Host 値	許容される接続
'fred'	'h1.example.net'	fred, h1.example.net からの接続
"	'h1.example.net'	h1.example.net から接続する任意のユーザー
'fred'	'%'	任意のホストから接続する fred
"	'%'	任意のホストから接続する任意のユーザー
'fred'	'%.example.net'	fred, example.net ドメインの任意のホストからの接続
'fred'	'x.example.%'	fred, x.example.net, x.example.com, x.example.edu からの接続など。これはおそらく役に立ちません
'fred'	'198.51.100.177'	IP アドレス 198.51.100.177 のホストから接続する fred
'fred'	'198.51.100.%'	198.51.100 のクラス C サブネットの任意のホストから接続する fred
'fred'	'198.51.100.0/255.255.255.0'	前の例と同じ

入接続のクライアントホスト名およびユーザー名が `user` テーブルの複数行と一致することもあります。前述の一連の例は、これを示しています: 表示されるエントリのいくつかは、fred による h1.example.net からの接続と一致します。

複数の一致が可能な場合、サーバーはいずれを使用するかを決定する必要があります。この問題は、次のように解決されます。

- サーバーが `user` テーブルをメモリーに読み取るとき、行を毎回ソートします。
- クライアントが接続しようとする、サーバーは行をソート順に参照します。
- サーバーは、クライアントホスト名およびユーザー名が一致した最初の行を使用します。

サーバーは、特定の `Host` 値が最も多い行から順に並べ替えるソートルールを使用します:

- リテラル IP アドレスとホスト名は最も具体的です。
- MySQL 8.0.23 より前は、リテラル IP アドレスの特異性はネットマスクがあるかどうかの影響を受けないため、198.51.100.13 と 198.51.100.0/255.255.255.0 は同等とみなされます。MySQL 8.0.23 の時点では、ホスト部分に IP アドレスを持つアカウントには次のような特異性があります:

- ホスト部分が IP アドレスとして指定されているアカウント:

```
CREATE USER 'user_name'@'127.0.0.1';  
CREATE USER 'user_name'@'198.51.100.44';
```

- CIDR 表記を使用して IP アドレスとして指定されたホスト部分を持つアカウント:

```
CREATE USER 'user_name'@'192.0.2.21/8';  
CREATE USER 'user_name'@'198.51.100.44/16';
```

- ホスト部分が IP アドレスとしてサブネットマスクとともに指定されているアカウント:

```
CREATE USER 'user_name'@'192.0.2.0/255.255.255.0';  
CREATE USER 'user_name'@'198.51.0.0/255.255.0.0';
```

- パターン '%' は「任意のホスト」を意味するため、具体性をもっとも低くなります。
- 空の文字列 "" も「任意のホスト」を意味しますが、 '%' のあとにソートされます。

非 TCP (ソケットファイル、名前付きパイプおよび共有メモリ) 接続は、ローカル接続として扱われ、そのようなカウントがある場合は `localhost` のホスト部分と照合され、そうでない場合は `localhost` と一致するワイルドカードを持つホスト部分と照合されます (たとえば、`local%`, `l%`, `%`)。

同じ `Host` 値を持つ行は、最も固有の `User` 値から順に並べられます。空白の `User` 値は「「任意のユーザー」」を意味し、最も限定的ではないため、同じ `Host` 値を持つ行の場合、匿名でないユーザーは匿名ユーザーの前にソートされます。

`Host` と `User` の値が等しい行の場合、順序は非決定的です。

`user` テーブルが次の内容であると仮定して、これがどのように作用するかを説明します。

```
+-----+-----+
| Host   | User   | ...
+-----+-----+
| %      | root   | ...
| %      | jeffrey | ...
| localhost | root   | ...
| localhost |       | ...
+-----+-----+
```

サーバーがテーブルをメモリーに読み取るとき、サーバーは前に記載したルールを使用して行をソートします。ソート後の結果は次のようになります。

```
+-----+-----+
| Host   | User   | ...
+-----+-----+
| localhost | root   | ...
| localhost |       | ...
| %      | jeffrey | ...
| %      | root   | ...
+-----+-----+
```

クライアントが接続しようとする時、サーバーはソート済みの行を参照し、見つかった最初の一致を使用します。`jeffrey` による `localhost` からの接続の場合、テーブルの 2 つの行が一致し、すなわち `Host` および `User` 値が `localhost` および "" であるものと、値が '%' および `jeffrey` であるものが一致します。ソート順では `localhost` 行が最初になるため、サーバーはこの行を使用します。

次に別の例を示します。 `user` テーブルが次のようになっていると仮定します。

```
+-----+-----+
| Host   | User   | ...
+-----+-----+
| %      | jeffrey | ...
| h1.example.net |       | ...
+-----+-----+
```

ソート済みテーブルは次のようになります。

```
+-----+-----+
| Host   | User   | ...
+-----+-----+
| h1.example.net |       | ...
| %      | jeffrey | ...
+-----+-----+
```

最初の行は `h1.example.net` の任意のユーザーによる接続と一致し、2 番目の行は任意のホストの `jeffrey` による接続と一致します。

注記

よくある誤解として、ある特定のユーザー名についてサーバーが接続に対する一致を検出しようとしたとき、そのユーザーの名前を明示的に指定するすべての行が最初に使用されるという認識があります。これは正しくありません。前述の例は、`jeffrey` による

`h1.example.net` からの接続が、`User` カラム値として `'jeffrey'` を含む行ではなく、ユーザー名のない行によって最初に照合されることを示しています。その結果、`jeffrey` は接続するときユーザー名を指定したにもかかわらず、匿名ユーザーとして認証されます。

サーバーに接続できても権限が期待したものとは異なる場合、おそらくほかのアカウントとして認証されています。サーバーがユーザーの認識に使用したアカウントを見つけるには、`CURRENT_USER()` 関数を使用します。(セクション 12.16 「情報関数」を参照してください。) これは、一致する `user` テーブル行の `User` および `Host` 値を示す、`user_name@host_name` 形式の値を返します。たとえば、`jeffrey` が接続して、次のクエリを発行したとします。

```
mysql> SELECT CURRENT_USER();
+-----+
| CURRENT_USER() |
+-----+
| @localhost |
+-----+
```

ここで表示された結果は、一致した `user` テーブル行の `User` カラム値が空白であることを示しています。つまり、サーバーは `jeffrey` を匿名ユーザーとして扱っています。

認証の問題を診断するための別の方法は、`user` テーブルを出力し、テーブルを手作業でソートして、最初の一致が行われた行を確認する方法です。

6.2.7 アクセス制御、ステージ 2: リクエストの確認

サーバーは接続を受け入れると、アクセス制御のステージ 2 に入ります。接続を介して発行するリクエストごとに、実行する操作がサーバーによって決定され、権限が十分かどうかチェックされます。ここで、付与テーブルの権限カラムが役立ちます。これらの権限は、`user`、`global_grants`、`db`、`tables_priv`、`columns_priv` テーブルまたは `procs_priv` テーブルのいずれかから取得できます。(各付与テーブルに存在するカラムを一覧表示するセクション 6.2.3 「付与テーブル」を参照すると役立つ場合があります。)

`user` テーブルおよび `global_grants` テーブルは、グローバル権限を付与します。特定のアカウントのこれらのテーブルの行は、デフォルトデータベースが何であるかに関係なく、グローバルに適用されるアカウント権限を示します。たとえば、`user` テーブルで `DELETE` 権限が付与されている場合、サーバーホスト上の任意のデータベース内の任意のテーブルから行を削除できます。`user` テーブルの権限は、データベース管理者などの権限を必要とするユーザーに対してのみ付与するのが賢明です。他のユーザーの場合は、`user` テーブルのすべての権限を `'N'` に設定したままにし、より具体的なレベルでのみ権限を付与します (特定のデータベース、テーブル、カラムまたはルーチンの場合)。データベース権限をグローバルに付与することもできますが、部分的な取消しを使用して、特定のデータベースでの実行を制限します (セクション 6.2.12 「部分取消しを使用した権限の制限」を参照)。

`db` テーブルは、データベース固有の権限を付与します。このテーブルのスコープカラムの値は、次の形式を取ることができます。

- 空白の `User` 値は匿名ユーザーに一致します。空白以外の値は文字どおりに一致し、ユーザー名にワイルドカードはありません。
- ワイルドカード文字 `%` および `_` は、`Host` カラムおよび `Db` カラムで使用できます。これらは `LIKE` 演算子で実行されるパターンマッチング演算と同じ意味を持ちます。権限を付与するときにいずれかの文字を文字どおりに使用する場合、文字をバックスラッシュでエスケープする必要があります。たとえば、アンダースコア文字 (`_`) をデータベース名の一部として含めるには、`GRANT` ステートメントで `_` として指定します。
- `'%'` または空白の `Host` 値は、「任意のホスト」を意味します。
- `'%'` または空白の `Db` 値は、「任意のデータベース」を意味します。

サーバーは `user` テーブルを読み取るのと同時に、`db` テーブルをメモリーに読み取ってソートします。サーバーは、`Host`、`Db`、および `User` スコープカラムに基づき `db` テーブルをソートします。`user` テーブルと同様に、ソートでは最も固有の値が最初に、最も固有の値が最後に配置され、サーバーで一致する行が検索されると、最初に検出された一致が使用されます。

`tables_priv`、`columns_priv`、および `procs_priv` テーブルは、テーブル固有、カラム固有、およびルーチン固有の権限を付与します。これらのテーブルのスコープカラムの値は、次の形式を取ることができます。

- ワイルドカード文字 `%` および `_` を `Host` カラムで使用できます。これらは `LIKE` 演算子で実行されるパターンマッチング演算と同じ意味を持ちます。
- `'%'` またはブランクの `Host` 値は、「任意のホスト」を意味します。
- `Db`、`Table_name`、`Column_name`、および `Routine_name` カラムは、ワイルドカードを含めたり、ブランクにしたりすることができません。

サーバーは、`Host`、`Db`、および `User` カラムに基づき、`tables_priv`、`columns_priv`、および `procs_priv` テーブルをソートします。これは `db` テーブルのソートと同様ですが、`Host` カラムのみワイルドカードを含めることができるため、よりシンプルです。

サーバーはソート済みテーブルを使用して、サーバーが受け取るリクエストを検証します。`SHUTDOWN` や `RELOAD` などの管理権限を必要とするリクエストの場合、サーバーは `user` テーブルと `global_privilege` テーブルのみをチェックします。これらは管理権限を指定する唯一のテーブルであるためです。サーバーは、それらのテーブル内のアカウントの行がリクエストされた操作を許可し、それ以外の場合はアクセスを拒否する場合にアクセス権を付与します。たとえば、ユーザーが `mysqladmin shutdown` を実行したいが、`user` テーブル行によって `SHUTDOWN` 権限がユーザーに付与されていない場合、サーバーは `db` テーブルさえもチェックせずにアクセスを拒否します。(後者のテーブルには `Shutdown_priv` カラムが含まれていないため、チェックする必要はありません。)

データベース関連のリクエスト (`INSERT`、`UPDATE` など) の場合、サーバーはまず `user` テーブルの行のユーザーグローバル権限をチェックします (部分的な取消しによる権制限は少なくなります)。リクエストされた操作が行で許可されている場合、アクセス権限が付与されます。`user` テーブルのグローバル権限が不十分な場合、サーバーは `db` テーブルからユーザーデータベース固有の権限を判別します:

- サーバーは、`db` テーブルを参照し `Host`、`Db`、および `User` カラムの一致がないかチェックします。
- `Host` および `User` カラムは、接続するユーザーのホスト名および MySQL ユーザー名に突き合わせされます。
- `Db` カラムは、ユーザーがアクセスしようとしているデータベースに突き合わせされます。
- `Host` および `User` に該当する行がない場合、アクセスは拒否されます。

`db` テーブルの行によって付与されたデータベース固有の権限を確認すると、サーバーは、`user` テーブルによって付与されたグローバル権限にそれらを追加します。その結果、リクエストされた操作が許可される場合、アクセス権限が付与されます。そうでない場合、サーバーは続けて `tables_priv` および `columns_priv` テーブル内でユーザーのテーブル権限およびカラム権限をチェックし、これらをユーザーの権限に追加し、結果に基づいてアクセスを許可または拒否します。ストアドルーチン操作の場合、サーバーは `tables_priv` および `columns_priv` の代わりに `procs_priv` テーブルを使用します。

ブール条件によって表現すると、ユーザーの権限を計算する方法についての前述の説明は、次のように要約できます。

```
global privileges
OR database privileges
OR table privileges
OR column privileges
OR routine privileges
```

グローバル権限が最初にリクエストされた操作に対して不十分であることがわかった場合、サーバーはこれらの権限を後でデータベース、テーブルおよびカラムの権限に追加する理由は明らかではありません。この理由は、1つのリクエストが複数のタイプの権限を必要とすることがあるためです。たとえば、`INSERT INTO ... SELECT` ステートメントを実行する場合、`INSERT` および `SELECT` 権限が両方必要です。`user` テーブルの行が一方の権限をグローバルに付与し、`db` テーブルの行が他方の権限を関連データベース専用で付与するような権限の場合があります。この場合、リクエストを実行するために必要な権限を持っていますが、サーバーはグローバル権限またはデータベース権限のみからそれを通知できません。結合された権限に基づいてアクセス制御を決定する必要があります。

6.2.8 アカウントの追加、権限の割当ておよびアカウントの削除

MySQL アカウントを管理するには、その目的の SQL ステートメントを使用します:

- `CREATE USER` および `DROP USER` は、アカウントを作成および削除します。
- `GRANT` および `REVOKE` は、アカウントに対する権限の割当ておよび取消しを行います。

- `SHOW GRANTS` にアカウント権限の割当てが表示されます。

アカウント管理ステートメントを使用すると、サーバーは基礎となる付与テーブルに適切な変更を加えます。これらについては、[セクション6.2.3「付与テーブル」](#)で説明します。

注記

`INSERT`、`UPDATE`、`DELETE`などのステートメントを使用して付与テーブルを直接変更することはお薦めできません。独自のリスクで実行してください。これらの変更の結果として誤った形式となった行を、サーバーは随意で無視します。

付与テーブルを変更する操作の場合、サーバーはテーブルが予期された構造を持っているかどうかをチェックし、持っていない場合はエラーを生成します。テーブルを必要な構造に更新するには、MySQLのアップグレード手順を実行します。[セクション2.11「MySQLのアップグレード」](#)を参照してください。

アカウントを作成するためのもう1つのオプションは、GUI ツール MySQL Workbench を使用する方法です。また、いくつかのサードパーティプログラムには、MySQL アカウント管理用の機能が用意されています。[phpMyAdmin](#) はそのようなプログラムです。

このセクションでは、次のトピックについて説明します。

- [アカウントの作成および権限の付与](#)
- [アカウントの権限およびプロパティの確認](#)
- [アカウント権限の取消し](#)
- [アカウントの削除](#)

ここで説明するステートメントの詳細は、[セクション13.7.1「アカウント管理ステートメント」](#)を参照してください。

アカウントの作成および権限の付与

次の例では、`mysql` クライアントプログラムを使用して、新しいアカウントを設定する方法を示します。これらの例では、MySQL `root` アカウントに `CREATE USER` 権限と、他のアカウントに付与されているすべての権限があることを前提としています。

コマンドラインで、MySQL `root` ユーザーとしてサーバーに接続し、パスワードプロンプトで適切なパスワードを指定します:

```
shell> mysql -u root -p
Enter password: (enter root password here)
```

サーバーに接続した後、新しいアカウントを追加できます。次の例では、`CREATE USER` および `GRANT` ステートメントを使用して4つのアカウントを設定します ('password'が表示されている場合は、適切なパスワードを置き換えます):

```
CREATE USER 'finley'@'localhost'
  IDENTIFIED BY 'password';
GRANT ALL
  ON *.*
  TO 'finley'@'localhost'
  WITH GRANT OPTION;

CREATE USER 'finley'@'%example.com'
  IDENTIFIED BY 'password';
GRANT ALL
  ON *.*
  TO 'finley'@'%example.com'
  WITH GRANT OPTION;

CREATE USER 'admin'@'localhost'
  IDENTIFIED BY 'password';
```



```
GRANT RELOAD,PROCESS
ON *.*
TO 'admin'@'localhost';

CREATE USER 'dummy'@'localhost';
```

これらのステートメントによって作成されるアカウントには、次のプロパティがあります:

- 2つのアカウントのユーザー名は `finley` です。どちらも、すべてを実行するための完全なグローバル権限を持つスーパーユーザーアカウントです。'`finley`'@'`localhost`'アカウントは、ローカルホストから接続する場合にのみ使用できます。'`finley`'@'`%example.com`'アカウントは、ホスト部分で'`%`'ワイルドカードを使用するため、`example.com` ドメイン内の任意のホストからの接続に使用できます。

'`finley`'@'`localhost`'アカウントは、`localhost` の匿名ユーザーアカウントがある場合に必要です。'`finley`'@'`localhost`'アカウントがない場合、その匿名ユーザーアカウントは、`finley` がローカルホストから接続し、`finley` が匿名ユーザーとして処理されるときに優先されます。これは、匿名ユーザーアカウントには'`finley`'@'`%`'アカウントよりも具体的な `Host` カラム値があるため、`user` テーブルのソート順が早いからです。(`user` テーブルのソートの詳細は、[セクション6.2.6「アクセス制御、ステージ 1: 接続の検証」](#) を参照してください。)

- '`admin`'@'`localhost`'アカウントは、ローカルホストから接続するために `admin` でのみ使用できます。 `RELOAD` および `PROCESS` のグローバル管理権限が付与されます。これらの権限を持つ `admin` ユーザーは、`mysqladmin reload`、`mysqladmin refresh`、`mysqladmin flush-xxx` コマンド、および `mysqladmin processlist` を実行できます。任意のデータベースにアクセスするための権限は付与されません。このような権限は、`GRANT` ステートメントを使用して追加できます。
- '`dummy`'@'`localhost`'アカウントにパスワードがありません (セキュアでないため、お薦めしません)。このアカウントは、ローカルホストから接続する際のみ使用できます。権限は付与されません。 `GRANT` ステートメントを使用してアカウントに特定の権限を付与することを前提としています。

前の例では、グローバルレベルで権限を付与しています。次の例では、3つのアカウントを作成し、下位レベル、つまりデータベース内の特定のデータベースまたはオブジェクトへのアクセス権を付与します。各アカウントのユーザー名は `custom` ですが、ホスト名の部分は異なります:

```
CREATE USER 'custom'@'localhost'
  IDENTIFIED BY 'password';
GRANT ALL
  ON bankaccount.*
  TO 'custom'@'localhost';

CREATE USER 'custom'@'host47.example.com'
  IDENTIFIED BY 'password';
GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
  ON expenses.*
  TO 'custom'@'host47.example.com';

CREATE USER 'custom'@'%example.com'
  IDENTIFIED BY 'password';
GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
  ON customer.addresses
  TO 'custom'@'%example.com';
```

3つのアカウントは、次のように使用できます。

- '`custom`'@'`localhost`'アカウントには、`bankaccount` データベースにアクセスするためのすべてのデータベースレベルの権限があります。アカウントを使用してサーバーに接続できるのは、ローカルホストからのみです。
- '`custom`'@'`host47.example.com`'アカウントには、`expenses` データベースにアクセスするための特定のデータベースレベル権限があります。このアカウントは、ホスト `host47.example.com` からのみサーバーに接続するために使用できます。
- '`custom`'@'`%example.com`'アカウントには、`example.com` ドメイン内の任意のホストから `customer` データベース内の `addresses` テーブルにアクセスするための特定のテーブルレベルの権限があります。アカウント名のホスト部分に `%` ワイルドカード文字が使用されているため、アカウントを使用してドメイン内のすべてのマシンからサーバーに接続できます。

アカウントの権限およびプロパティの確認

アカウントの権限を表示するには、[SHOW GRANTS](#) を使用します:

```
mysql> SHOW GRANTS FOR 'admin'@'localhost';
+-----+
| Grants for admin@localhost |
+-----+
| GRANT RELOAD, PROCESS ON *.* TO 'admin'@'localhost' |
+-----+
```

アカウントの非特権プロパティを表示するには、[SHOW CREATE USER](#) を使用します:

```
mysql> SET print_identified_with_as_hex = ON;
mysql> SHOW CREATE USER 'admin'@'localhost'\G
***** 1. row *****
CREATE USER for admin@localhost: CREATE USER 'admin'@'localhost'
IDENTIFIED WITH 'caching_sha2_password'
AS 0x24412430303524301D0E17054E2241362B1419313C3E44326F294133734B30792F436E77764270373039612E32445250786D43594F45354532324B616979
REQUIRE NONE PASSWORD EXPIRE DEFAULT ACCOUNT UNLOCK
PASSWORD HISTORY DEFAULT
PASSWORD REUSE INTERVAL DEFAULT
PASSWORD REQUIRE CURRENT DEFAULT
```

[print_identified_with_as_hex](#) システム変数 (MySQL 8.0.17 で使用可能) を有効にすると、[SHOW CREATE USER](#) では、印刷できない文字を含むハッシュ値が、通常の文字列リテラルとしてではなく 16 進数文字列として表示されます。

アカウント権限の取消し

アカウント権限を取り消すには、[REVOKE](#) ステートメントを使用します。権限は、異なるレベルで付与できると同様に、異なるレベルで取り消すことができます。

グローバル権限を取り消します:

```
REVOKE ALL
ON *.*
FROM 'finley'@'%.example.com';

REVOKE RELOAD
ON *.*
FROM 'admin'@'localhost';
```

データベースレベルの権限を取り消します:

```
REVOKE CREATE,DROP
ON expenses.*
FROM 'custom'@'host47.example.com';
```

テーブルレベルの権限を取り消します:

```
REVOKE INSERT,UPDATE,DELETE
ON customer.addresses
FROM 'custom'@'%.example.com';
```

権限取消しの影響を確認するには、[SHOW GRANTS](#) を使用します:

```
mysql> SHOW GRANTS FOR 'admin'@'localhost';
+-----+
| Grants for admin@localhost |
+-----+
| GRANT PROCESS ON *.* TO 'admin'@'localhost' |
+-----+
```

アカウントの削除

アカウントを削除するには、[DROP USER](#) ステートメントを使用します。たとえば、以前に作成したアカウントの一部を削除するには、次のようにします:

```
DROP USER 'finley'@'localhost';
```

```
DROP USER 'finley'@'%example.com';
DROP USER 'admin'@'localhost';
DROP USER 'dummy'@'localhost';
```

6.2.9 予約済アカウント

MySQL のインストールプロセスの一部として、データディレクトリの初期化があります ([セクション2.10.1「データディレクトリの初期化」](#) を参照)。データディレクトリの初期化中に、MySQL は予約済とみなす必要があるユーザーアカウントを作成します:

- `'root'@'localhost'`: 管理目的で使用されます。このアカウントにはすべての権限があり、システムアカウントであり、任意の操作を実行できます。

厳密に言うと、このアカウント名は予約されていません。一部のインストールでは、よく知られた名前を持つ特権の高いアカウントが公開されないように、`root` アカウントの名前が別の名前に変更されます。
- `'mysql.sys'@'localhost'`: `sys` スキーマオブジェクトの `DEFINER` として使用されます。`mysql.sys` アカウントを使用すると、DBA が `root` アカウントの名前を変更または削除した場合に発生する問題を回避できます。このアカウントは、クライアント接続に使用できないようにロックされています。
- `'mysql.session'@'localhost'`: サーバーにアクセスするためにプラグインによって内部的に使用されます。このアカウントは、クライアント接続に使用できないようにロックされています。アカウントはシステムアカウントです。
- `'mysql.infoschema'@'localhost'`: `INFORMATION_SCHEMA` ビューの `DEFINER` として使用されます。`mysql.infoschema` アカウントを使用すると、DBA が `root` アカウントの名前を変更または削除した場合に発生する問題を回避できます。このアカウントは、クライアント接続に使用できないようにロックされています。

6.2.10 ロールの使用

MySQL ロールは、権限の名前付きコレクションです。ユーザーアカウントと同様に、ロールには付与された権限と取り消された権限があります。

ユーザーアカウントには、各ロールに関連付けられた権限をアカウントに付与するロールを付与できます。これにより、アカウントへの一連の権限の割当てが可能になり、必要な権限割当てを概念化して実装するために、個々の権限を付与するかわりに便利です。

次のリストに、MySQL が提供するロール管理機能の概要を示します:

- `CREATE ROLE` および `DROP ROLE` は、ロールを作成および削除します。
- `GRANT` および `REVOKE` は、ユーザーアカウントおよびロールから権限を取り消す権限を割り当てます。
- `SHOW GRANTS` には、ユーザーアカウントおよびロールの権限およびロールの割当てが表示されます。
- `SET DEFAULT ROLE` では、デフォルトでアクティブなアカウントロールを指定します。
- `SET ROLE` は、現在のセッション内のアクティブなロールを変更します。
- `CURRENT_ROLE()` 関数は、現在のセッション内のアクティブなロールを表示します。
- `mandatory_roles` および `activate_all_roles_on_login` システム変数を使用すると、ユーザーがサーバーにログインするときに、必須ロールを定義し、付与されたロールを自動的にアクティブ化できます。

個々のロール操作ステートメント (それらの使用に必要な権限を含む) の詳細は、[セクション13.7.1「アカウント管理ステートメント」](#) を参照してください。次の説明では、ロールの使用例を示します。特に指定がない限り、ここに示す SQL ステートメントは、`root` アカウントなどの十分な管理権限を持つ MySQL アカウントを使用して実行する必要があります。

- [ロールの作成およびロールへの権限の付与](#)
- [必須ロールの定義](#)
- [ロール権限の確認](#)

- [ロールのアクティブ化](#)
- [ロールまたはロール権限の取消し](#)
- [ロールの削除](#)
- [ユーザーとロールの互換性](#)

ロールの作成およびロールへの権限の付与

次のシナリオを考えてみます:

- アプリケーションは、`app_db` という名前のデータベースを使用します。
- アプリケーションに関連付けられている場合、アプリケーションを作成および保守する開発者、およびアプリケーションと対話するユーザーのアカウントが存在する可能性があります。
- 開発者には、データベースへの完全なアクセス権が必要です。一部のユーザーには読取りアクセスのみが必要で、その他のユーザーには読取り/書き込みアクセスが必要です。

多くのユーザーアカウントに権限を個別に付与しないようにするには、必要な権限セットの名前としてロールを作成します。これにより、適切なロールを付与することで、必要な権限をユーザーアカウントに簡単に付与できます。

ロールを作成するには、`CREATE ROLE` ステートメントを使用します:

```
CREATE ROLE 'app_developer', 'app_read', 'app_write';
```

ロール名はユーザーアカウント名とよく似ており、`'user_name'@'host_name'`形式のユーザー部分とホスト部分で構成されます。ホスト部分を省略すると、デフォルトで`'%'`に設定されます。ユーザー部分とホスト部分は、`-` や `%` などの特殊文字が含まれていないかぎり、引用符で囲むことはできません。アカウント名とは異なり、ロール名のユーザー部分は空白にできません。追加情報については [セクション6.2.5「ロール名の指定」](#) を参照してください。

ロールに権限を割り当てるには、ユーザーアカウントに権限を割り当てる場合と同じ構文を使用して `GRANT` ステートメントを実行します:

```
GRANT ALL ON app_db.* TO 'app_developer';
GRANT SELECT ON app_db.* TO 'app_read';
GRANT INSERT, UPDATE, DELETE ON app_db.* TO 'app_write';
```

最初に、1つの開発者アカウント、読取り専用アクセスを必要とする2つのユーザーアカウント、および読取り/書き込みアクセスを必要とする1つのユーザーアカウントが必要であるとします。 `CREATE USER` を使用してアカウントを作成します:

```
CREATE USER 'dev1'@'localhost' IDENTIFIED BY 'dev1pass';
CREATE USER 'read_user1'@'localhost' IDENTIFIED BY 'read_user1pass';
CREATE USER 'read_user2'@'localhost' IDENTIFIED BY 'read_user2pass';
CREATE USER 'rw_user1'@'localhost' IDENTIFIED BY 'rw_user1pass';
```

各ユーザーアカウントに必要な権限を割り当てるには、前述と同じ形式の `GRANT` ステートメントを使用できますが、ユーザーごとに個別の権限を列挙する必要があります。かわりに、権限ではなくロールの付与を許可する代替の `GRANT` 構文を使用します:

```
GRANT 'app_developer' TO 'dev1'@'localhost';
GRANT 'app_read' TO 'read_user1'@'localhost', 'read_user2'@'localhost';
GRANT 'app_read', 'app_write' TO 'rw_user1'@'localhost';
```

`rw_user1` アカウントの `GRANT` ステートメントは、読取りおよび書き込みロールを付与します。このロールを組み合わせると、必要な読取りおよび書き込み権限を提供します。

アカウントにロールを付与するための `GRANT` 構文は、権限を付与するための構文とは異なります: 権限を割り当てる `ON` 句はありますが、ロールを割り当てる `ON` 句はありません。構文は異なるため、同じステートメントで権限とロールの割当てを混在させることはできません。(権限とロールの両方をアカウントに割り当てることはできますが、付与する内容に適した構文を持つ個別の `GRANT` ステートメントを使用する必要があります。) MySQL 8.0.16 では、匿名ユーザーにロールを付与できません。

作成されたロールはロックされ、パスワードがなく、デフォルトの認証プラグインが割り当てられます。(これらのロール属性は、後で `ALTER USER` ステートメントを使用して、グローバル `CREATE USER` 権限を持つユーザーが変更できます。)

ロック中は、ロールを使用してサーバーに対する認証を行うことはできません。ロックが解除されている場合は、ロールを使用して認証できます。これは、ロールとユーザーの両方が認可識別子であり、それらを区別することはほとんどなく共通であるためです。[ユーザーとロールの互換性](#)も参照してください。

必須ロールの定義

`mandatory_roles` システム変数の値でロールに名前を付けることで、ロールを必須として指定できます。サーバーは必須ロールをすべてのユーザーに付与されたものとして処理するため、アカウントに明示的に付与する必要はありません。

サーバーの起動時に必須ロールを指定するには、サーバーの `my.cnf` ファイルで `mandatory_roles` を定義します:

```
[mysqld]
mandatory_roles='role1,role2@localhost,r3@%.example.com'
```

実行時に `mandatory_roles` を設定および永続化するには、次のようなステートメントを使用します:

```
SET PERSIST mandatory_roles = 'role1,role2@localhost,r3@%.example.com';
```

`SET PERSIST` は、実行中の MySQL インスタンスの値を設定します。また、値が保存され、その後のサーバーの再起動に引き継がれます。後続の再起動に引き継ぐことなく、実行中の MySQL インスタンスの値を変更するには、`PERSIST` ではなく `GLOBAL` キーワードを使用します。[セクション13.7.6.1「変数代入の SET 構文」](#)を参照してください。

`mandatory_roles` を設定するには、グローバルシステム変数を設定するために通常必要な `SYSTEM_VARIABLES_ADMIN` 権限 (または非推奨の `SUPER` 権限) に加えて、`ROLE_ADMIN` 権限が必要です。

明示的に付与されたロールと同様に、必須ロールはアクティブ化されるまで有効になりません ([ロールのアクティブ化](#)を参照)。ログイン時に、`activate_all_roles_on_login` システム変数が有効になっている場合は付与されているすべてのロールに対して、それ以外の場合はデフォルトロールとして設定されているロールに対してロールのアクティブ化が行われます。実行時に、`SET ROLE` によってロールがアクティブ化されます。

`mandatory_roles` の値で指定されたロールは、`REVOKE` で取り消すことも、`DROP ROLE` または `DROP USER` で削除することもできません。

セッションがデフォルトでシステムセッションにならないようにするには、`SYSTEM_USER` 権限を持つロールを `mandatory_roles` システム変数の値にリストできません:

- `SYSTEM_USER` 権限を持つロールが起動時に `mandatory_roles` に割り当てられた場合、サーバーはエラーログにメッセージを書き込み、終了します。
- `SYSTEM_USER` 権限を持つロールが実行時に `mandatory_roles` に割り当てられた場合、エラーが発生し、`mandatory_roles` 値は変更されません。

`mandatory_roles` で指定されたロールが `mysql.user` システムテーブルに存在しない場合、そのロールはユーザーに付与されません。サーバーは、ユーザーに対してロールのアクティブ化を試行しても、存在しないロールを必須として処理せず、エラーログに警告を書き込みます。ロールが後で作成されて有効になった場合は、`FLUSH PRIVILEGES` でサーバーに必須として処理させる必要がある場合があります。

`SHOW GRANTS` では、[セクション13.7.7.21「SHOW GRANTS ステートメント」](#)で説明されているルールに従って必須ロールが表示されます。

ロール権限の確認

アカウントに割り当てられた権限を確認するには、`SHOW GRANTS` を使用します。例:

```
mysql> SHOW GRANTS FOR 'dev1'@'localhost';
+-----+
| Grants for dev1@localhost |
+-----+
```

```
| GRANT USAGE ON *.* TO `dev1`@`localhost` |
| GRANT `app_developer`@`%` TO `dev1`@`localhost` |
+-----+
```

ただし、「展開」なしで付与された各ロールは、そのロールが表す権限に表示されます。ロール権限も表示するには、権限を表示する付与されたロールを指定する `USING` 句を追加します:

```
mysql> SHOW GRANTS FOR 'dev1'@'localhost' USING 'app_developer';
+-----+
| Grants for dev1@localhost |
+-----+
| GRANT USAGE ON *.* TO `dev1`@`localhost` |
| GRANT ALL PRIVILEGES ON `app_db`.* TO `dev1`@`localhost` |
| GRANT `app_developer`@`%` TO `dev1`@`localhost` |
+-----+
```

他のタイプのユーザーを同様に検証します:

```
mysql> SHOW GRANTS FOR 'read_user1'@'localhost' USING 'app_read';
+-----+
| Grants for read_user1@localhost |
+-----+
| GRANT USAGE ON *.* TO `read_user1`@`localhost` |
| GRANT SELECT ON `app_db`.* TO `read_user1`@`localhost` |
| GRANT `app_read`@`%` TO `read_user1`@`localhost` |
+-----+
mysql> SHOW GRANTS FOR 'rw_user1'@'localhost' USING 'app_read', 'app_write';
+-----+
| Grants for rw_user1@localhost |
+-----+
| GRANT USAGE ON *.* TO `rw_user1`@`localhost` |
| GRANT SELECT, INSERT, UPDATE, DELETE ON `app_db`.* TO `rw_user1`@`localhost` |
| GRANT `app_read`@`%`,`app_write`@`%` TO `rw_user1`@`localhost` |
+-----+
```

`SHOW GRANTS` では、[セクション13.7.7.21「SHOW GRANTS ステートメント」](#) で説明されているルールに従って必須ロールが表示されます。

ロールのアクティブ化

ユーザーアカウントに付与されるロールは、アカウントセッション内でアクティブまたは非アクティブにできます。付与されたロールがセッション内でアクティブな場合は、その権限が適用されます。それ以外の場合は適用されません。現在のセッション内でアクティブなロールを判別するには、`CURRENT_ROLE()` 関数を使用します。

デフォルトでは、アカウントにロールを付与したり、`mandatory_roles` システム変数値に名前を付けたりしても、アカウントセッション内でロールが自動的にアクティブになることはありません。たとえば、前述の説明では `rw_user1` ロールがアクティブ化されていないため、`rw_user1` としてサーバーに接続し、`CURRENT_ROLE()` 関数を起動すると、結果は `NONE` (アクティブなロールなし) になります:

```
mysql> SELECT CURRENT_ROLE();
+-----+
| CURRENT_ROLE() |
+-----+
| NONE |
+-----+
```

ユーザーがサーバーに接続して認証するたびにアクティブにするロールを指定するには、`SET DEFAULT ROLE` を使用します。以前に作成した各アカウントに割り当てられたすべてのロールにデフォルトを設定するには、次のステートメントを使用します:

```
SET DEFAULT ROLE ALL TO
`dev1`@`localhost`,
`read_user1`@`localhost`,
`read_user2`@`localhost`,
`rw_user1`@`localhost`;
```

`rw_user1` として接続すると、`CURRENT_ROLE()` の初期値に新しいデフォルトのロール割当てが反映されます:

```
mysql> SELECT CURRENT_ROLE();
+-----+
```



```
| CURRENT_ROLE() |
+-----+
| 'app_read'@'%', 'app_write'@'%` |
+-----+
```

ユーザーがサーバーに接続したときに、明示的に付与されたロールと必須ロールがすべて自動的にアクティブ化されるようにするには、`activate_all_roles_on_login` システム変数を有効にします。デフォルトでは、自動ロールアクティブ化は無効になっています。

セッション内で、ユーザーは `SET ROLE` を実行してアクティブなロールのセットを変更できます。たとえば、`rw_user1` の場合は次のようになります：

```
mysql> SET ROLE NONE; SELECT CURRENT_ROLE();
+-----+
| CURRENT_ROLE() |
+-----+
| NONE           |
+-----+
mysql> SET ROLE ALL EXCEPT 'app_write'; SELECT CURRENT_ROLE();
+-----+
| CURRENT_ROLE() |
+-----+
| 'app_read'@'%` |
+-----+
mysql> SET ROLE DEFAULT; SELECT CURRENT_ROLE();
+-----+
| CURRENT_ROLE() |
+-----+
| 'app_read'@'%', 'app_write'@'%` |
+-----+
```

最初の `SET ROLE` ステートメントは、すべてのロールを非アクティブ化します。次に、`rw_user1` を事実上読み取り専用にします。3 目はデフォルトのロールをリストアします。

ストアードプログラムおよびビューオブジェクトの有効なユーザーは、`DEFINER` および `SQL SECURITY` 属性の影響を受けます。これらの属性は、実行者コンテキストまたは定義者コンテキストのどちらで実行されるかを決定します ([セクション25.6「ストアードオブジェクトのアクセス制御」](#) を参照)：

- 起動側コンテキストで実行されるストアードプログラムおよびビューオブジェクトは、現在のセッション内でアクティブなロールで実行されます。
- 定義者コンテキストで実行されるストアードプログラムおよびビューオブジェクトは、`DEFINER` 属性で指定されたユーザーのデフォルトロールで実行されます。`activate_all_roles_on_login` が有効な場合、このようなオブジェクトは、`DEFINER` ユーザーに付与されたすべてのロール (必須ロールを含む) で実行されます。ストアードプログラムでは、デフォルトとは異なるロールで実行する必要がある場合、プログラム本体は `SET ROLE` を実行して必要なロールをアクティブ化する必要があります。

ロールまたはロール権限の取消し

ロールは、アカウントに付与できるのと同様に、アカウントから取り消すことができます：

```
REVOKE role FROM user;
```

`mandatory_roles` システム変数値で指定されたロールは取り消すことができません。

`REVOKE` をロールに適用して、付与された権限を変更することもできます。これは、ロール自体だけでなく、そのロールを付与されたアカウントにも影響します。すべてのアプリケーションユーザーを一時的に読み取り専用にします。これを行うには、`REVOKE` を使用して、`app_write` ロールから変更権限を取り消します：

```
REVOKE INSERT, UPDATE, DELETE ON app_db.* FROM 'app_write';
```

これが発生すると、`SHOW GRANTS` を使用して表示できるように、権限のないロールのままになります (これは、このステートメントをユーザーだけでなくロールで使用できることを示しています)：

```
mysql> SHOW GRANTS FOR 'app_write';
+-----+
| Grants for app_write@% |
+-----+
```

```
| GRANT USAGE ON *.* TO `app_write`@`%` |
+-----+
+-----+
```

ロールから権限を取り消すと、変更されたロールが割り当てられているユーザーの権限に影響するため、`rw_user1`にはテーブルの変更権限がなくなりました (`INSERT`、`UPDATE` および `DELETE` は存在しません):

```
mysql> SHOW GRANTS FOR `rw_user1`@`localhost`
        USING `app_read`, `app_write`;
+-----+
| Grants for rw_user1@localhost |
+-----+
| GRANT USAGE ON *.* TO `rw_user1`@`localhost` |
| GRANT SELECT ON `app_db`.* TO `rw_user1`@`localhost` |
| GRANT `app_read`@`%`,`app_write`@`%` TO `rw_user1`@`localhost` |
+-----+
```

実際には、`rw_user1`の読取り/書込みユーザーが読取り専用ユーザーになりました。これは、`app_write`ロールが付与されている他のアカウントに対しても発生し、ロールの使用によって個々のアカウントの権限を変更する必要がなくなる方法を示します。

変更権限をロールにリストアするには、変更権限を再度付与します:

```
GRANT INSERT, UPDATE, DELETE ON app_db.* TO `app_write`;
```

`app_write`ロールを付与された他のアカウントと同様に、`rw_user1`にも変更権限があります。

ロールの削除

ロールを削除するには、`DROP ROLE`を使用します:

```
DROP ROLE `app_read`, `app_write`;
```

ロールを削除すると、そのロールが付与されたすべてのアカウントからロールが取り消されます。

`mandatory_roles`システム変数値で指定されたロールは削除できません。

ユーザーとロールの互換性

以前に `SHOW GRANTS` についてヒントされていたように、ユーザーアカウントまたはロールに対する権限が表示されるため、アカウントおよびロールは同じ意味で使用できます。

ロールとユーザーの違いの1つは、`CREATE ROLE`ではデフォルトでロックされる認可識別子が作成されるのに対し、`CREATE USER`ではデフォルトでロック解除される認可識別子が作成されることです。ただし、適切な権限を持つユーザーは、作成後にロールまたはユーザーをロックまたはロック解除できるため、区別は不変ではありません。

データベース管理者が、特定の認可識別子をロールにする必要があるというプリファレンスを持っている場合は、名前スキームを使用してこの意図を伝達できます。たとえば、ロールにするすべての認可識別子に `r_`接頭辞を使用でき、それ以外には使用できません。

ロールとユーザーの別の違いは、それらの管理に使用できる権限にあります:

- `CREATE ROLE` および `DROP ROLE` 権限では、それぞれ `CREATE ROLE` および `DROP ROLE` ステートメントのみを使用できます。
- `CREATE USER` 権限を使用すると、`ALTER USER`、`CREATE ROLE`、`CREATE USER`、`DROP ROLE`、`DROP USER`、`RENAME USER` ステートメントおよび `REVOKE ALL PRIVILEGES` ステートメントを使用できます。

したがって、`CREATE ROLE` および `DROP ROLE` 権限は `CREATE USER` ほど強力ではなく、ロールの作成および削除のみを許可し、より一般的なアカウント操作を実行しないユーザーに付与できます。

ユーザーおよびロールの権限および互換性に関して、ユーザーアカウントをロールのように扱い、そのアカウントを別のユーザーまたはロールに付与できます。その結果、他のユーザーまたはロールにアカウント権限およびロールが付与されます。

次の一連のステートメントは、ユーザーにユーザー、ユーザーにロール、ロールにユーザーまたはロールを付与できることを示しています:

```
CREATE USER 'u1';
CREATE ROLE 'r1';
GRANT SELECT ON db1.* TO 'u1';
GRANT SELECT ON db2.* TO 'r1';
CREATE USER 'u2';
CREATE ROLE 'r2';
GRANT 'u1', 'r1' TO 'u2';
GRANT 'u1', 'r1' TO 'r2';
```

いずれの場合も、付与されたオブジェクトに関連付けられた権限が権限受領者オブジェクトに付与されます。これらのステートメントを実行すると、ユーザー (**u1**) およびロール (**r1**) から各 **u2** および **r2** に権限が付与されます:

```
mysql> SHOW GRANTS FOR 'u2' USING 'u1', 'r1';
+-----+
| Grants for u2@% |
+-----+
| GRANT USAGE ON *.* TO `u2`@`%` |
| GRANT SELECT ON `db1`.* TO `u2`@`%` |
| GRANT SELECT ON `db2`.* TO `u2`@`%` |
| GRANT `u1`@`%`,`r1`@`%` TO `u2`@`%` |
+-----+
mysql> SHOW GRANTS FOR 'r2' USING 'u1', 'r1';
+-----+
| Grants for r2@% |
+-----+
| GRANT USAGE ON *.* TO `r2`@`%` |
| GRANT SELECT ON `db1`.* TO `r2`@`%` |
| GRANT SELECT ON `db2`.* TO `r2`@`%` |
| GRANT `u1`@`%`,`r1`@`%` TO `r2`@`%` |
+-----+
```

前述の例は単なる説明ですが、ユーザーアカウントとロールの互換性には、次のような実用的なアプリケーションがあります: レガシーアプリケーション開発プロジェクトが MySQL でロールが追加される前に開始されたため、プロジェクトに関連付けられているすべてのユーザーアカウントに権限が直接付与されているとします (ロールの付与によって権限が付与されるのではなく)。これらのアカウントの 1 つは、次のように最初に権限を付与された開発者アカウントです:

```
CREATE USER 'old_app_dev'@'localhost' IDENTIFIED BY 'old_app_devpass';
GRANT ALL ON old_app.* TO 'old_app_dev'@'localhost';
```

この開発者がプロジェクトを終了した場合は、権限を別のユーザーに割り当てるか、開発アクティビティが展開されている場合は複数のユーザーに割り当てる必要があります。次に、この問題を処理する方法をいくつか示します:

- **ロールを使用しない:** 元の開発者が使用できないようにアカウントパスワードを変更し、かわりに新しい開発者がそのアカウントを使用するようにします:

```
ALTER USER 'old_app_dev'@'localhost' IDENTIFIED BY 'new_password';
```

- **ロールの使用:** アカウントをロックして、誰もがそのアカウントを使用してサーバーに接続できないようにします:

```
ALTER USER 'old_app_dev'@'localhost' ACCOUNT LOCK;
```

次に、アカウントをロールとして扱います。プロジェクトの新規開発者ごとに、新規アカウントを作成し、元の開発者アカウントを付与します:

```
CREATE USER 'new_app_dev1'@'localhost' IDENTIFIED BY 'new_password';
GRANT 'old_app_dev'@'localhost' TO 'new_app_dev1'@'localhost';
```

その結果、元の開発者アカウント権限が新しいアカウントに割り当てられます。

6.2.11 アカウントカテゴリ

MySQL 8.0.16 では、MySQL に、**SYSTEM_USER** 権限に基づいてユーザーアカウントカテゴリの概念が組み込まれています。

- [システムアカウントと通常アカウント](#)
- [SYSTEM_USER 権限の影響を受ける操作](#)

- システムおよび通常のセッション
- 通常アカウントによる操作からのシステムアカウントの保護

システムアカウントと通常アカウント

MySQL には、ユーザーアカウントカテゴリの概念が組み込まれており、システムユーザーと通常のユーザーは `SYSTEM_USER` 権限を持っているかどうかによって区別されます:

- `SYSTEM_USER` 権限を持つユーザーはシステムユーザーです。
- `SYSTEM_USER` 権限を持たないユーザーは通常のユーザーです。

`SYSTEM_USER` 権限は、特定のユーザーが他の権限を適用できるアカウント、およびそのユーザーが他のアカウントから保護されているかどうかに影響します:

- システムユーザーは、システムアカウントと通常アカウントの両方を変更できます。つまり、通常アカウントに対して特定の操作を実行する適切な権限を持つユーザーは、システムアカウントに対しても操作を実行するように `SYSTEM_USER` を所有することで有効になります。システムアカウントは、通常のユーザーではなく、適切な権限を持つシステムユーザーのみが変更できます。
- 適切な権限を持つ通常のユーザーは通常アカウントを変更できますが、システムアカウントは変更できません。通常アカウントは、適切な権限を持つシステムユーザーと通常のユーザーの両方が変更できます。

ユーザーが通常アカウントに対して特定の操作を実行するための適切な権限を持っている場合、`SYSTEM_USER` を使用すると、ユーザーはシステムアカウントに対しても操作を実行できます。`SYSTEM_USER` は他の権限を意味しないため、特定のアカウント操作を実行する機能は、他の必要な権限を所有するための述語のままです。たとえば、ユーザーが `SELECT` および `UPDATE` 権限を通常アカウントに付与できる場合、`SYSTEM_USER` では、ユーザーは `SELECT` および `UPDATE` をシステムアカウントに付与することもできます。

システムアカウントと通常アカウントの区別により、`SYSTEM_USER` 権限を持つアカウントを権限を持たないアカウントから保護することで、特定のアカウント管理の問題をより適切に制御できます。たとえば、`CREATE USER` 権限では、新しいアカウントの作成のみでなく、既存のアカウントの変更および削除も可能です。システムユーザーの概念がない場合、`CREATE USER` 権限を持つユーザーは、`root` アカウントを含む既存のアカウントを変更または削除できます。システムユーザーの概念により、システムユーザーのみが行うことができるように、`root` アカウント (それ自体はシステムアカウント) への変更を制限できます。`CREATE USER` 権限を持つ通常のユーザーは、既存のアカウントを変更または削除できますが、通常アカウントのみです。

SYSTEM_USER 権限の影響を受ける操作

`SYSTEM_USER` 権限は、次の操作に影響します:

- アカウント操作。

アカウント操作には、アカウントの作成と削除、権限の付与と取消し、資格証明や認証プラグインなどのアカウント認証特性の変更、およびパスワード有効期限ポリシーなどの他のアカウント特性の変更が含まれます。

`CREATE USER` や `GRANT` などのアカウント管理ステートメントを使用してシステムアカウントを操作するには、`SYSTEM_USER` 権限が必要です。この方法でアカウントがシステムアカウントに変更できないようにするには、`SYSTEM_USER` 権限を付与しないで、通常アカウントにします。(ただし、システムアカウントを通常アカウントから完全に保護するには、`mysql` システムスキーマの変更権限を通常アカウントから源泉徴収する必要があります。通常アカウントによる操作からのシステムアカウントの保護を参照してください。)

- 現在のセッションおよびその中で実行されているステートメントを強制終了します。

`SYSTEM_USER` 権限で実行されているセッションまたはステートメントを強制終了するには、他の必要な権限 (`CONNECTION_ADMIN` または非推奨の `SUPER` 権限) に加えて、自分のセッションに `SYSTEM_USER` 権限が必要です。

MySQL 8.0.16 より前は、セッションまたはステートメントを強制終了するには `CONNECTION_ADMIN` 権限 (または非推奨の `SUPER` 権限) で十分です。

- ストアドオブジェクトの `DEFINER` 属性を設定します。

ストアオブジェクトの `DEFINER` 属性を `SYSTEM_USER` 権限を持つアカウントに設定するには、他の必要な権限 (`SET_USER_ID` または非推奨の `SUPER` 権限) に加えて、`SYSTEM_USER` 権限が必要です。

MySQL 8.0.16 より前は、`SET_USER_ID` 権限 (または非推奨の `SUPER` 権限) を使用して、ストアオブジェクトの `DEFINER` 値を指定するだけで十分です。

- 必須ロールの指定。

`SYSTEM_USER` 権限を持つロールは、`mandatory_roles` システム変数の値にリストできません。

MySQL 8.0.16 より前は、`mandatory_roles` に任意のロールをリストできます。

システムおよび通常のセッション

サーバー内で実行されているセッションは、システムユーザーと通常のユーザーの区別と同様に、システムセッションまたは通常のセッションと区別されます:

- `SYSTEM_USER` 権限を持つセッションはシステムセッションです。
- `SYSTEM_USER` 権限を持たないセッションは通常のセッションです。

通常のセッションは、通常のユーザーに許可されている操作のみを実行できます。システムセッションは、システムユーザーにのみ許可される操作を追加で実行できます。

セッションによって所有される権限は、基礎となるアカウントに直接付与される権限と、セッション内で現在アクティブなすべてのロールに付与される権限です。したがって、そのアカウントに `SYSTEM_USER` 権限が直接付与されているか、セッションで `SYSTEM_USER` 権限を持つロールがアクティブ化されているため、セッションはシステムセッションである可能性があります。セッション内でアクティブでないアカウントに付与されたロールは、セッション権限には影響しません。

ロールをアクティブ化および非アクティブ化すると、セッションが所有する権限が変更される可能性があるため、セッションは通常のセッションからシステムセッションに、またはその逆に変更される可能性があります。セッションが `SYSTEM_USER` 権限を持つロールをアクティブ化または非アクティブ化すると、通常のセッションとシステムセッションの間の適切な変更は、そのセッションに対してのみ即時に行われます:

- 通常のセッションが `SYSTEM_USER` 権限を持つロールをアクティブ化すると、そのセッションはシステムセッションになります。
- システムセッションが `SYSTEM_USER` 権限を持つロールを非アクティブ化した場合、`SYSTEM_USER` 権限を持つ他のロールがアクティブなままでないかぎり、そのセッションは通常のセッションになります。

これらの操作は、既存のセッションには影響しません:

- `SYSTEM_USER` 権限がアカウントに対して付与または取り消された場合、そのアカウントの既存のセッションは通常のセッションとシステムセッションの間で変更されません。付与または取消し操作は、アカウントによる後続の接続のセッションにのみ影響します。
- セッション内で呼び出されたストアオブジェクトによって実行されるステートメントは、オブジェクトの `DEFINER` 属性がシステムアカウントを指定している場合でも、親セッションのシステムステータスまたは通常のステータスで実行されます。

ロールのアクティブ化はセッションにのみ影響し、アカウントには影響しないため、`SYSTEM_USER` 権限を持つロールを通常のアカウントに付与しても、そのアカウントは通常のユーザーから保護されません。このロールは、ロールがアクティブ化されているアカウントのセッションのみを保護し、通常のセッションによる強制終了からのみセッションを保護します。

通常アカウントによる操作からのシステムアカウントの保護

アカウント操作には、アカウントの作成と削除、権限の付与と取消し、資格証明や認証プラグインなどのアカウント認証特性の変更、およびパスワード有効期限ポリシーなどの他のアカウント特性の変更が含まれます。

アカウント操作は、次の 2 つの方法で実行できます:

- `CREATE USER` や `GRANT` などのアカウント管理ステートメントを使用します。これが推奨される方法です。
- `INSERT` や `UPDATE` などのステートメントを使用して権限テーブルを直接変更します。この方法はお薦めしませんが、付与テーブルを含む `mysql` システムスキーマに対する適切な権限を持つユーザーにはお薦めします。

特定のアカウントによる変更からシステムアカウントを完全に保護するには、通常のアカウントにし、`mysql` スキーマに対する変更権限を付与しないでください:

- `account-management` ステートメントを使用してシステムアカウントを操作するには、`SYSTEM_USER` 権限が必要です。この方法でアカウントがシステムアカウントを変更できないようにするには、`SYSTEM_USER` を付与しないで、通常のアカウントにします。これには、アカウントに付与されたロールへの `SYSTEM_USER` の付与は含まれません。
- `mysql` スキーマの権限を使用すると、変更するアカウントが通常のアカウントであっても、権限付与テーブルを直接変更することでシステムアカウントを操作できます。通常のアカウントによるシステムアカウントの不正な直接変更を制限するには、`mysql` スキーマの変更権限をアカウント (またはアカウントに付与されたロール) に付与しないでください。通常のアカウントに、すべてのスキーマに適用されるグローバル権限が必要な場合は、部分的な取消しを使用して課される権限制限を使用して、`mysql` スキーマの変更を防止できます。セクション6.2.12「部分取消しを使用した権限の制限」を参照してください。

注記

`SYSTEM_USER` 権限を源泉徴収することで、アカウントがシステムアカウントを変更できなくなりますが、通常のアカウントは変更できなくなりますが、`mysql` スキーマ権限を源泉徴収することで、アカウントは通常のアカウントと同様にシステムアカウントを変更できなくなりますが、前述のように、直接付与テーブルの変更はお薦めしないため、これは問題ではありません。

すべてのスキーマに対するすべての権限を持つユーザー `u1` を作成するとします。ただし、`u1` はシステムアカウントを変更できない通常のユーザーである必要があります。 `partial_revokes` システム変数が有効になっている場合は、次のように `u1` を構成します:

```
CREATE USER u1 IDENTIFIED BY 'password';

GRANT ALL ON *.* TO u1 WITH GRANT OPTION;
-- GRANT ALL includes SYSTEM_USER, so at this point
-- u1 can manipulate system or regular accounts

REVOKE SYSTEM_USER ON *.* FROM u1;
-- Revoking SYSTEM_USER makes u1 a regular user;
-- now u1 can use account-management statements
-- to manipulate only regular accounts

REVOKE ALL ON mysql.* FROM u1;
-- This partial revoke prevents u1 from directly
-- modifying grant tables to manipulate accounts
```

アカウントによるすべての `mysql` システムスキーマへのアクセスを防止するには、次に示すように、`mysql` スキーマに対するすべての権限を取り消します。読取り専用アクセスなどの部分的な `mysql` スキーマアクセスを許可することもできます。次の例では、`SELECT`、`INSERT`、`UPDATE` および `DELETE` 権限を持つアカウントをすべてのスキーマに対してグローバルに作成しますが、`mysql` スキーマに対しては `SELECT` のみを作成します:

```
CREATE USER u2 IDENTIFIED BY 'password';
GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO u2;
REVOKE INSERT, UPDATE, DELETE ON mysql.* FROM u2;
```

また、すべての `mysql` スキーマ権限を取り消し、特定の `mysql` テーブルまたはカラムへのアクセス権を付与することもできます。これは、`mysql` で部分的な取消しを使用しても実行できます。次のステートメントは、`mysql` スキーマ内の `u1` への読取り専用アクセスを有効にしますが、`user` テーブルの `db` テーブル、`Host` カラムおよび `User` カラムに対してのみ有効にします:

```
CREATE USER u3 IDENTIFIED BY 'password';
GRANT ALL ON *.* TO u3;
REVOKE ALL ON mysql.* FROM u3;
GRANT SELECT ON mysql.db TO u3;
GRANT SELECT(Host,User) ON mysql.user TO u3;
```


6.2.12 部分取消しを使用した権限の制限

MySQL 8.0.16 より前は、特定のスキーマを除き、グローバルに適用される権限を付与することはできません。MySQL 8.0.16 では、これは `partial_revokes` システム変数が有効な場合に可能です。具体的には、グローバルレベルの権限を持つユーザーの場合、`partial_revokes` では、特定のスキーマの権限を取り消しながら、他のスキーマの権限をそのままにすることができます。したがって、権限の制限は、グローバル権限を持つが、特定のスキーマへのアクセスを許可しないアカウントの管理に役立つ場合があります。たとえば、`mysql` システムスキーマ内のテーブルを除く任意のテーブルの変更をアカウントに許可できます。

- [部分失効の使用](#)
- [部分的な取消しと明示的なスキーマ付与](#)
- [部分失効の無効化](#)
- [部分的な取消しとレプリケーション](#)

注記

簡潔にするために、ここに示す `CREATE USER` ステートメントにはパスワードは含まれていません。本番で使用する場合は、常にアカウントパスワードを割り当てます。

部分失効の使用

`partial_revokes` システム変数は、アカウントに権限制限を設定できるかどうかを制御します。デフォルトでは、`partial_revokes` は無効になっており、グローバル権限を部分的に取り消そうとするとエラーが発生します：

```
mysql> CREATE USER u1;
mysql> GRANT SELECT, INSERT ON *.* TO u1;
mysql> REVOKE INSERT ON world.* FROM u1;
ERROR 1141 (42000): There is no such grant defined for user 'u1' on host '%'
```

`REVOKE` 操作を許可するには、`partial_revokes` を有効にします：

```
SET PERSIST partial_revokes = ON;
```

`SET PERSIST` は、実行中の MySQL インスタンスの値を設定します。また、値が保存され、その後のサーバーの再起動に引き継がれます。後続の再起動に引き継ぐことなく、実行中の MySQL インスタンスの値を変更するには、`PERSIST` ではなく `GLOBAL` キーワードを使用します。セクション13.7.6.1「変数代入の `SET` 構文」を参照してください。

`partial_revokes` が有効になっている場合、部分的な取消しは成功します：

```
mysql> REVOKE INSERT ON world.* FROM u1;
mysql> SHOW GRANTS FOR u1;
+-----+
| Grants for u1@% |
+-----+
| GRANT SELECT, INSERT ON *.* TO `u1`@`%` |
| REVOKE INSERT ON `world`.* FROM `u1`@`%` |
+-----+
```

`SHOW GRANTS` の出力には、部分的な取消しが `REVOKE` ステートメントとしてリストされます。この結果は、`world` スキーマのテーブルに対して `INSERT` を実行できないことを除き、`u1` にグローバルな `SELECT` および `INSERT` 権限があることを示しています。つまり、`u1` による `world` テーブルへのアクセスは読取り専用です。

サーバーは、部分的な取消しによって実装された権限制限を `mysql.user` システムテーブルに記録します。アカウントに部分的な失効がある場合、その `User_attributes` カラム値には `Restrictions` 属性があります：

```
mysql> SELECT User, Host, User_attributes->'$.Restrictions'
FROM mysql.user WHERE User_attributes->'$.Restrictions' <> '';
+-----+
| User | Host | User_attributes->'$.Restrictions' |
+-----+
| u1 | % | [{"Database": "world", "Privileges": ["INSERT"]}]] |
+-----+
```

注記

部分的な取消しは任意のスキーマに適用できますが、`mysql` システムスキーマに対する権限制限は、通常のアカウントがシステムアカウントを変更できないようにする戦略の一環として特に役立ちます。通常アカウントによる操作からのシステムアカウントの保護を参照してください。

部分的な取消し操作には、次の条件があります:

- 部分的な取消しでは、スキーマに文字どおりの名前を付ける必要があります。 `%` または `_SQL` ワイルドカード文字 (`myschema%` など) を含むスキーマ名は使用できません。
- 部分的な取消しを使用して、存在しないスキーマに制限を設定できますが、取り消す権限がグローバルに付与されている場合のみです。権限がグローバルに付与されていない場合、存在しないスキーマに対して権限を取り消すとエラーが発生します。
- 部分的な取消しはスキーマレベルでのみ適用されます。部分取消しは、グローバルにのみ適用される権限 (`FILE`、`BINLOG_ADMIN` など)、またはテーブル、カラムまたはルーチン権限には使用できません。

前述のように、スキーマレベルの権限の部分的な取消しは、`SHOW GRANTS` 出力に `REVOKE` ステートメントとして表示されます。これは、`SHOW GRANTS` が「「プレーン」」スキーマレベルの権限を表す方法とは異なります:

- 付与されると、スキーマレベルの権限は、出力内の独自の `GRANT` ステートメントによって表されます:

```
mysql> CREATE USER u1;
mysql> GRANT UPDATE ON mysql.* TO u1;
mysql> GRANT DELETE ON world.* TO u1;
mysql> SHOW GRANTS FOR u1;
+-----+
| Grants for u1@%          |
+-----+
| GRANT USAGE ON *.* TO `u1`@`%` |
| GRANT UPDATE ON `mysql`.* TO `u1`@`%` |
| GRANT DELETE ON `world`.* TO `u1`@`%` |
+-----+
```

- 取り消すと、スキーマレベルの権限は単に出力から消えます。これらは `REVOKE` ステートメントとしては表示されません:

```
mysql> REVOKE UPDATE ON mysql.* FROM u1;
mysql> REVOKE DELETE ON world.* FROM u1;
mysql> SHOW GRANTS FOR u1;
+-----+
| Grants for u1@%          |
+-----+
| GRANT USAGE ON *.* TO `u1`@`%` |
+-----+
```

ユーザーが権限を付与すると、権限受領者がその権限を持たない権限をすでに持っていないかぎり、権限受領者はその権限を継承します。次の 2 人のユーザーについて考えてみます。いずれかのユーザーにグローバル `SELECT` 権限があります:

```
CREATE USER u1, u2;
GRANT SELECT ON *.* TO u2;
```

管理ユーザー `admin` に、グローバルだが部分的に取り消された `SELECT` 権限があるとします:

```
mysql> CREATE USER admin;
mysql> GRANT SELECT ON *.* TO admin WITH GRANT OPTION;
mysql> REVOKE SELECT ON mysql.* FROM admin;
mysql> SHOW GRANTS FOR admin;
+-----+
| Grants for admin@%      |
+-----+
| GRANT SELECT ON *.* TO `admin`@`%` WITH GRANT OPTION |
| REVOKE SELECT ON `mysql`.* FROM `admin`@`%`          |
+-----+
```

`admin` によって `SELECT` が `u1` および `u2` にグローバルに付与される場合、結果はユーザーごとに異なります:

- `admin` が `SELECT` 権限を持たない `u1` に `SELECT` をグローバルに付与する場合、`u1` は `admin` 権限制限を継承します:

```
mysql> GRANT SELECT ON *.* TO u1;
mysql> SHOW GRANTS FOR u1;
+-----+
| Grants for u1@% |
+-----+
| GRANT SELECT ON *.* TO `u1`@`%` |
| REVOKE SELECT ON `mysql`.* FROM `u1`@`%` |
+-----+
```

- 一方、`u2` は制限なしでグローバル `SELECT` 権限をすでに保持しています。 `GRANT` は、権限受領者の既存の権限にのみ追加でき、権限を減らすことはできないため、`admin` が `SELECT` を `u2` にグローバルに付与する場合、`u2` は `admin` の制限を継承しません:

```
mysql> GRANT SELECT ON *.* TO u2;
mysql> SHOW GRANTS FOR u2;
+-----+
| Grants for u2@% |
+-----+
| GRANT SELECT ON *.* TO `u2`@`%` |
+-----+
```

`GRANT` ステートメントに `AS user` 句が含まれている場合、適用される権限制限は、ステートメントを実行するユーザーではなく、句で指定されたユーザー/ロールの組合せに対する権限制限です。 `AS` 句の詳細は、[セクション 13.7.1.6 「GRANT ステートメント」](#) を参照してください。

アカウントに付与される新しい権限の制限は、そのアカウントの既存の制限に追加されます:

```
mysql> CREATE USER u1;
mysql> GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO u1;
mysql> REVOKE INSERT ON mysql.* FROM u1;
mysql> SHOW GRANTS FOR u1;
+-----+
| Grants for u1@% |
+-----+
| GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO `u1`@`%` |
| REVOKE INSERT ON `mysql`.* FROM `u1`@`%` |
+-----+
mysql> REVOKE DELETE, UPDATE ON db2.* FROM u1;
mysql> SHOW GRANTS FOR u1;
+-----+
| Grants for u1@% |
+-----+
| GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO `u1`@`%` |
| REVOKE UPDATE, DELETE ON `db2`.* FROM `u1`@`%` |
| REVOKE INSERT ON `mysql`.* FROM `u1`@`%` |
+-----+
```

権限制限の集計は、権限が明示的に明示的に取り消された場合 (前述のとおり)、およびステートメントを実行するユーザーまたは `AS user` 句に指定されたユーザーから制限が暗黙的に継承された場合の両方に適用されます。

アカウントにスキーマに対する権限制限がある場合:

- アカウントは、制限付きスキーマまたはその中のオブジェクトに対する権限を他のアカウントに付与することはできません。
- 制限のない別のアカウントは、制限付きスキーマまたはその中のオブジェクトの制限付きアカウントに権限を付与できます。 無制限のユーザーが次のステートメントを実行するとします:

```
CREATE USER u1;
GRANT SELECT, INSERT, UPDATE ON *.* TO u1;
REVOKE SELECT, INSERT, UPDATE ON mysql.* FROM u1;
GRANT SELECT ON mysql.user TO u1; -- grant table privilege
GRANT SELECT(Host,User) ON mysql.db TO u1; -- grant column privileges
```

結果のアカウントには次の権限があり、制限付きスキーマ内で制限付き操作を実行できます:

```
mysql> SHOW GRANTS FOR u1;
```

```
+-----+
| Grants for u1@% |
+-----+
| GRANT SELECT, INSERT, UPDATE ON *.* TO `u1`@`%` |
| REVOKE SELECT, INSERT, UPDATE ON `mysql`.* FROM `u1`@`%` |
| GRANT SELECT ('Host', `User`) ON `mysql`.`db` TO `u1`@`%` |
| GRANT SELECT ON `mysql`.`user` TO `u1`@`%` |
+-----+
```

アカウントにグローバル権限の制限がある場合、その制限は次のいずれかのアクションによって削除されます:

- 権限に対する制限のないアカウントによるアカウントへの権限のグローバルな付与。
- スキーマレベルでの権限の付与。
- 権限をグローバルに取り消します。

複数の権限をグローバルに保持しているが、**INSERT**、**UPDATE** および **DELETE** に制限があるユーザー **u1** について考えてみます:

```
mysql> CREATE USER u1;
mysql> GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO u1;
mysql> REVOKE INSERT, UPDATE, DELETE ON mysql.* FROM u1;
mysql> SHOW GRANTS FOR u1;
+-----+
| Grants for u1@% |
+-----+
| GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO `u1`@`%` |
| REVOKE INSERT, UPDATE, DELETE ON `mysql`.* FROM `u1`@`%` |
+-----+
```

制限のないアカウントから **u1** にグローバルに権限を付与すると、権限制限が削除されます。たとえば、**INSERT** の制限を削除するには:

```
mysql> GRANT INSERT ON *.* TO u1;
mysql> SHOW GRANTS FOR u1;
+-----+
| Grants for u1@% |
+-----+
| GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO `u1`@`%` |
| REVOKE UPDATE, DELETE ON `mysql`.* FROM `u1`@`%` |
+-----+
```

スキーマレベルで **u1** に権限を付与すると、権限の制限がなくなります。たとえば、**UPDATE** の制限を削除するには:

```
mysql> GRANT UPDATE ON mysql.* TO u1;
mysql> SHOW GRANTS FOR u1;
+-----+
| Grants for u1@% |
+-----+
| GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO `u1`@`%` |
| REVOKE DELETE ON `mysql`.* FROM `u1`@`%` |
+-----+
```

グローバル権限を取り消すと、その権限に対する制限も含めて権限が削除されます。たとえば、**DELETE** の制限を削除するには、次のようにします (すべての **DELETE** アクセスを削除します):

```
mysql> REVOKE DELETE ON *.* FROM u1;
mysql> SHOW GRANTS FOR u1;
+-----+
| Grants for u1@% |
+-----+
| GRANT SELECT, INSERT, UPDATE ON *.* TO `u1`@`%` |
+-----+
```

アカウントにグローバルレベルとスキーマレベルの両方の権限がある場合、部分的な取消しを有効にするには、スキーマレベルで 2 回取り消す必要があります。 **u1** に次の権限があり、**INSERT** がグローバルおよび **world** スキーマの両方で保持されているとします:

```
mysql> CREATE USER u1;
```

```
mysql> GRANT SELECT, INSERT ON *.* TO u1;
mysql> GRANT INSERT ON world.* TO u1;
mysql> SHOW GRANTS FOR u1;
+-----+
| Grants for u1@% |
+-----+
| GRANT SELECT, INSERT ON *.* TO `u1`@`%` |
| GRANT INSERT ON `world`.* TO `u1`@`%` |
+-----+
```

`world` で `INSERT` を取り消すと、スキーマレベルの権限が取り消されます (`SHOW GRANTS` にはスキーマレベルの `GRANT` ステートメントは表示されなくなります):

```
mysql> REVOKE INSERT ON world.* FROM u1;
mysql> SHOW GRANTS FOR u1;
+-----+
| Grants for u1@% |
+-----+
| GRANT SELECT, INSERT ON *.* TO `u1`@`%` |
+-----+
```

`world` で `INSERT` を再度取り消すと、グローバル権限の部分的な取消しが実行されます (`SHOW GRANTS` にはスキーマレベルの `REVOKE` ステートメントが含まれるようになりました):

```
mysql> REVOKE INSERT ON world.* FROM u1;
mysql> SHOW GRANTS FOR u1;
+-----+
| Grants for u1@% |
+-----+
| GRANT SELECT, INSERT ON *.* TO `u1`@`%` |
| REVOKE INSERT ON `world`.* FROM `u1`@`%` |
+-----+
```

部分的な取消しと明示的なスキーマ付与

一部のスキーマのアカウントへのアクセスを提供し、他のスキーマのアカウントへのアクセスを提供するために、部分的な取消しでは、グローバル権限を付与せずにスキーマレベルのアクセス権を明示的に付与する方法の代替方法が提供されます。2つのアプローチには、長所と短所があります。

グローバル権限ではなく、スキーマレベルの権限を付与します:

- 新規スキーマの追加: デフォルトでは、スキーマには既存のアカウントからアクセスできません。スキーマにアクセスできるアカウントの場合、DBA はスキーマレベルのアクセス権を付与する必要があります。
- 新規アカウントの追加: DBA は、アカウントがアクセスできるスキーマごとにスキーマレベルのアクセス権を付与する必要があります。

部分的な取消しと組み合わせたグローバル権限の付与:

- 新規スキーマの追加: スキーマは、グローバル権限を持つ既存のアカウントからアクセスできます。スキーマにアクセスできないアカウントの場合、DBA は部分的な取消しを追加する必要があります。
- 新規アカウントの追加: DBA は、グローバル権限に加えて、各制限付きスキーマに対する部分的な取消しを付与する必要があります。

明示的なスキーマレベルの付与を使用するアプローチは、アクセスがいくつかのスキーマに制限されているアカウントでは便利です。部分的な取消しを使用するアプローチは、少数を除くすべてのスキーマへの広範なアクセス権を持つアカウントではより便利です。

部分失効の無効化

一度有効にすると、アカウントに権限制限がある場合は `partial_revokes` を無効にできません。そのようなアカウントが存在する場合、`partial_revokes` の無効化は失敗します:

- 起動時に `partial_revokes` を無効にしようとすると、サーバーはエラーメッセージをログに記録し、`partial_revokes` を有効にします。

- 実行時に `partial_revokes` を無効にしようとすると、エラーが発生し、`partial_revokes` 値は変更されません。

制限が存在する場合に `partial_revokes` を無効にするには、最初に制限を削除する必要があります:

- 部分的な失効があるアカウントを判別します:

```
SELECT User, Host, User_attributes->>$.Restrictions'  
FROM mysql.user WHERE User_attributes->>$.Restrictions' <> '';
```

- このようなアカウントごとに、その権限制限を削除します。前のステップで、アカウント `u1` に次の制限があるとします:

```
[{"Database": "world", "Privileges": ["INSERT", "DELETE"]}
```

制限の削除は様々な方法で実行できます:

- 制限なしで権限をグローバルに付与します:

```
GRANT INSERT, DELETE ON *.* TO u1;
```

- スキーマレベルで権限を付与します:

```
GRANT INSERT, DELETE ON world.* TO u1;
```

- 権限をグローバルに取り消します (不要になった場合):

```
REVOKE INSERT, DELETE ON *.* FROM u1;
```

- アカウント自体を削除します (不要になった場合):

```
DROP USER u1;
```

すべての権限制限を削除した後、部分的な取消しを無効にできます:

```
SET PERSIST partial_revokes = OFF;
```

部分的な取消しとレプリケーション

レプリケーションシナリオでは、`partial_revokes` が任意のホストで有効になっている場合、すべてのホストで有効にする必要があります。それ以外の場合、グローバル権限を部分的に取り消す `REVOKE` ステートメントは、レプリケーションが発生するすべてのホストで同じ効果がないため、レプリケーションの不整合またはエラーが発生する可能性があります。

6.2.13 権限変更が有効化される時期

`--skip-grant-tables` オプションを指定せずに `mysqld` サーバーを起動すると、起動シーケンス中にすべての付与テーブルの内容がメモリーに読み込まれます。インメモリーテーブルは、その時点でアクセス制御に有効になります。

`account-management` ステートメントを使用して付与テーブルを間接的に変更すると、サーバーはこれらの変更に応じ、付与テーブルをすぐにメモリーに再度ロードします。アカウント管理ステートメントについては、[セクション 13.7.1 「アカウント管理ステートメント」](#) を参照してください。たとえば、`GRANT`、`REVOKE`、`SET PASSWORD` や `RENAME USER` などです。

`INSERT`、`UPDATE`、`DELETE` (非推奨) などのステートメントを使用して付与テーブルを直接変更した場合、その変更は、テーブルをリロードするか再起動するようにサーバーに指示するまで、権限チェックには影響しません。したがって、付与テーブルを直接変更してもリロードを忘れた場合、サーバーを再起動するまで変更には影響なしが含まれます。このため、変更したのに違いが現れないことを不思議に思うことがあるかもしれません。

付与テーブルをリロードするようサーバーに指示するには、フラッシュ権限操作を実行します。これは、`FLUSH PRIVILEGES` ステートメントを発行するか、`mysqladmin flush-privileges` または `mysqladmin reload` コマンドを実行することによって行うことができます。

付与テーブルのリロードは、次のように既存の各クライアントセッションの権限に影響します:

- テーブルおよびカラムの権限の変更は、クライアントの次のリクエストで有効になります。

- データベース権限の変更は、クライアントが次回 `USE db_name` ステートメントを実行したときに有効になります。

注記

クライアントアプリケーションはデータベース名をキャッシュできます。そのため、実際に別のデータベースに変更しないと、この効果が表示されない場合があります。

- 接続済みクライアントについてのグローバルな権限およびパスワードは影響されません。これらの変更は、後続の接続のセッションでのみ有効になります。

セッション内のアクティブなロールのセットに対する変更は、そのセッションに対してのみ即時に有効になります。`SET ROLE` ステートメントは、セッションロールのアクティブ化および非アクティブ化を実行します ([セクション 13.7.1.11 「SET ROLE ステートメント」](#) を参照)。

サーバーが `--skip-grant-tables` オプションを指定して起動された場合、サーバーは付与テーブルを読み取ったりアクセス制御を実装したりしません。すべてのユーザーがセキュアでないに接続し、任意の操作を実行できます。そのように起動されたサーバーが、テーブルを読み取ってアクセスチェックを有効にするようにするには、権限をフラッシュします。

6.2.14 アカウントパスワードの割り当て

MySQL サーバーに接続するクライアントに必要な証明書には、パスワードを含めることができます。このセクションでは、MySQL アカウントにパスワードを割り当てる方法について説明します。

MySQL は、`mysql` システムデータベースの `user` テーブルに資格証明を格納します。パスワードを割り当てたり変更したりする操作は、`CREATE USER` 権限を持つユーザー、または `mysql` データベースに対する権限 (新しいアカウントを作成するための `INSERT` 権限、既存のアカウントを変更するための `UPDATE` 権限) にのみ許可されます。`read_only` システム変数が有効になっている場合、`CREATE USER` や `ALTER USER` などのアカウント変更ステートメントを使用するには、`CONNECTION_ADMIN` 権限 (または非推奨の `SUPER` 権限) も必要です。

ここでは、最も一般的なパスワード割り当てステートメントの構文のみをまとめています。その他の可能性の詳細は、[セクション 13.7.1.3 「CREATE USER ステートメント」](#)、[セクション 13.7.1.1 「ALTER USER ステートメント」](#) および [セクション 13.7.1.10 「SET PASSWORD ステートメント」](#) を参照してください。

MySQL は、プラグインを使用してクライアント認証を実行します。[セクション 6.2.17 「プラグイン認証」](#) を参照してください。パスワード割り当てステートメントでは、アカウントに関連付けられた認証プラグインは、指定されたクリプトパスワードに必要なハッシュを実行します。これにより、MySQL では、パスワードを `mysql.user` システムテーブルに格納する前に不明瞭化できます。ここで説明するステートメントでは、MySQL は指定されたパスワードを自動的にハッシュします。`CREATE USER` および `ALTER USER` には、ハッシュ値を文字どおりに指定できる構文もあります。詳細は、これらのステートメントの説明を参照してください。

新しいアカウントの作成時にパスワードを割り当てるには、`CREATE USER` を使用して `IDENTIFIED BY` 句を含めません:

```
CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'password';
```

`CREATE USER` は、アカウント認証プラグインを指定するための構文もサポートしています。[セクション 13.7.1.3 「CREATE USER ステートメント」](#) を参照してください。

既存のアカウントのパスワードを割り当てたり変更したりするには、`IDENTIFIED BY` 句を含む `ALTER USER` ステートメントを使用します:

```
ALTER USER 'jeffrey'@'localhost' IDENTIFIED BY 'password';
```

匿名ユーザーとして接続していない場合は、自分のアカウントに文字どおりに名前を付けずに自分のパスワードを変更できます:

```
ALTER USER USER() IDENTIFIED BY 'password';
```

コマンドラインからアカウントパスワードを変更するには、`mysqladmin` コマンドを使用します:

```
mysqladmin -u user_name -h host_name password "password"
```

このコマンドでパスワードを設定するアカウントは、`User` カラムに `user_name` が、`Host` カラムにクライアントホスト接続元が一致する `mysql.user` システムテーブルの行を持つアカウントです。

警告

`mysqladmin` を使用してパスワードを設定する場合は、セキュアでないとみなす必要があります。一部のシステムでは、使用しているパスワードが、コマンド行を表示するためにほかのユーザーによって起動できる `ps` などのシステムステータスプログラムによって表示可能になります。MySQL クライアントは通常、クライアントの初期化シーケンス中にコマンド行パスワード引数をゼロで上書きします。ただし、まだ値が表示可能な短い期間があります。また、一部のシステムではこの上書きの方法には効果がなく、パスワードは `ps` から表示可能になったままになります。(SystemV Unix システムおよびおそらくほかのシステムでもこの問題の影響があります。)

MySQL レプリケーションを使用している場合は、現在、`CHANGE REPLICATION SOURCE TO` ステートメント (MySQL 8.0.23 の場合) または `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 の場合) の一部としてレプリカによって使用されるパスワードの長さが事実上 32 文字に制限されていることに注意してください。パスワードが長い場合は、余分な文字が切り捨てられます。これは通常、MySQL Server によって課される制限によるものではなく、MySQL レプリケーションに固有の問題です。

6.2.15 パスワード管理

MySQL は、次のパスワード管理機能をサポートしています:

- パスワードの有効期限。パスワードを定期的に変更する必要があります。
- 古いパスワードが再度選択されないようにするためのパスワード再利用の制限。
- パスワードの検証。パスワードの変更を要求するには、置換する現在のパスワードも指定します。
- デュアルパスワード。クライアントがプライマリパスワードまたはセカンダリパスワードを使用して接続できるようにします。
- 強力なパスワードを要求するためのパスワード強度評価。
- ランダムパスワード生成。明示的な管理者指定のリテラルパスワードを要求するかわりに使用します。
- パスワードの失敗を追跡して、連続するパスワードログインの失敗が多すぎる場合に一時アカウントロックを有効にします。

次の各セクションでは、`validate_password` コンポーネントを使用して実装され、[セクション6.4.3「パスワード検証コンポーネント」](#) で説明されているパスワード強度評価を除き、これらの機能について説明します。

- [内部資格証明記憶域と外部資格証明記憶域](#)
- [パスワード有効期限ポリシー](#)
- [パスワード再利用ポリシー](#)
- [パスワード検証必須ポリシー](#)
- [デュアルパスワードのサポート](#)
- [ランダムパスワード生成](#)
- [失敗したログイントラッキングと一時アカウントロック](#)

重要

MySQL は、`mysql` システムデータベースのテーブルを使用してパスワード管理機能を実装します。MySQL を以前のバージョンからアップグレードする場合、システムテーブルが最新でない可能性があります。その場合、サーバーは起動プロセス中に次のようなメッセージをエラーログに書き込みます (正確な数値は異なる場合があります):

```
[ERROR] Column count of mysql.user is wrong. Expected
49, found 47. The table is probably corrupted
[Warning] ACL table mysql.password_history missing.
Some operations may fail.
```

この問題を修正するには、MySQL のアップグレード手順を実行します。 [セクション 2.11 「MySQL のアップグレード」](#) を参照してください。これが完了するまで、パスワードは変更できませんは次のようになります。

内部資格証明記憶域と外部資格証明記憶域

一部の認証プラグインでは、アカウント資格証明が `mysql.user` システムテーブルの MySQL に内部的に格納されます:

- [mysql_native_password](#)
- [caching_sha2_password](#)
- [sha256_password](#)

ここで説明するほとんどのパスワード管理機能は、MySQL 自体で処理される内部資格証明記憶域に基づいているため、このセクションのほとんどの説明はこのような認証プラグインに適用されます。その他の認証プラグインは、MySQL の外部にアカウント資格証明を格納します。外部資格証明システムに対して認証を実行するプラグインを使用するアカウントの場合、パスワード管理もそのシステムに対して外部で処理する必要があります。

例外は、内部資格証明記憶域を使用するアカウントのみでなく、失敗したログイン追跡および一時アカウントロックのオプションがすべてのアカウントに適用されることです。これは、MySQL では、内部資格証明記憶域を使用するか外部資格証明記憶域を使用するかに関係なく、任意のアカウントのログイン試行のステータスを評価できるためです。

個々の認証プラグインの詳細は、[セクション 6.4.1 「認証プラグイン」](#) を参照してください。

パスワード有効期限ポリシー

MySQL を使用すると、データベース管理者はアカウントパスワードを手動で期限切れにしたり、自動パスワード期限切れのポリシーを設定できます。有効期限ポリシーはグローバルに設定でき、個々のアカウントは、グローバルポリシーに従うか、特定のアカウントごとの動作でグローバルポリシーをオーバーライドするように設定できます。

アカウントパスワードを手動で期限切れにするには、`ALTER USER` ステートメントを使用します:

```
ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE;
```

この操作は、`mysql.user` システムテーブルの対応する行でパスワードを期限切れとマークします。

ポリシーに従ったパスワードの有効期限は自動的に設定され、特定のアカウントのパスワード有効期限は、最新のパスワード変更の日時から評価されます。`mysql.user` システムテーブルには、各アカウントのパスワードが最後に変更された時間が示され、その有効期間が許容される存続期間を超えると、サーバーはクライアント接続時にパスワードを期限切れとして自動的に処理します。これは、明示的な手動パスワード有効期限なしで機能します。

自動パスワード失効ポリシーをグローバルに確立するには、`default_password_lifetime` システム変数を使用します。デフォルト値は 0 で、自動パスワード有効期限は無効になります。`default_password_lifetime` の値が正の整数の `N` である場合、パスワードを `N` 日ごとに変更する必要があるように、許可されたパスワードの存続期間を示します。

例:

- パスワードの存続期間が約 6 か月であるグローバルポリシーを確立するには、サーバー `my.cnf` ファイルで次の行を使用してサーバーを起動します:

```
[mysqld]
default_password_lifetime=180
```

- パスワードが期限切れにならないようにグローバルポリシーを設定するには、`default_password_lifetime` を 0 に設定します:

```
[mysqld]
```

```
default_password_lifetime=0
```

- `default_password_lifetime` は、実行時に設定および永続化することもできます:

```
SET PERSIST default_password_lifetime = 180;  
SET PERSIST default_password_lifetime = 0;
```

`SET PERSIST` は、実行中の MySQL インスタンスの値を設定します。また、後続のサーバー再起動に引き継ぐための値も保存されます。[セクション13.7.6.1「変数代入の SET 構文」](#)を参照してください。後続の再起動に引き継ぐことなく、実行中の MySQL インスタンスの値を変更するには、`PERSIST` ではなく `GLOBAL` キーワードを使用します。

グローバルパスワード失効ポリシーは、オーバーライドするように設定されていないすべてのアカウントに適用されます。個々のアカウントのポリシーを設定するには、`CREATE USER` および `ALTER USER` ステートメントの `PASSWORD EXPIRE` オプションを使用します。[セクション13.7.1.3「CREATE USER ステートメント」](#)および[セクション13.7.1.1「ALTER USER ステートメント」](#)を参照してください。

アカウント固有のステートメントの例:

- 90 日ごとにパスワードを変更する必要があります:

```
CREATE USER 'jeffrey'@'localhost' PASSWORD EXPIRE INTERVAL 90 DAY;  
ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE INTERVAL 90 DAY;
```

この有効期限オプションは、ステートメントで指定されたすべてのアカウントのグローバルポリシーをオーバーライドします。

- パスワードの有効期限の無効化:

```
CREATE USER 'jeffrey'@'localhost' PASSWORD EXPIRE NEVER;  
ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE NEVER;
```

この有効期限オプションは、ステートメントで指定されたすべてのアカウントのグローバルポリシーをオーバーライドします。

- ステートメントで指定されたすべてのアカウントのグローバルな有効期限ポリシーに従います:

```
CREATE USER 'jeffrey'@'localhost' PASSWORD EXPIRE DEFAULT;  
ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE DEFAULT;
```

クライアントが正常に接続すると、サーバーはアカウントパスワードの有効期限が切れているかどうかを判断します:

- サーバーは、パスワードが手動で期限切れになっているかどうかを確認します。
- それ以外の場合、サーバーは、自動パスワード有効期限ポリシーに従って、パスワードの有効期間が許可された持続期間を超えているかどうかを確認します。その場合、サーバーはパスワードの有効期限が切れたとみなします。

パスワードの有効期限が切れた場合(手動が自動かに関係なく)、サーバーはクライアントを切断するか、許可されている操作を制限します([セクション6.2.16「期限切れパスワードのサーバー処理」](#)を参照)。制限付きクライアントによって実行される操作は、ユーザーが新しいアカウントパスワードを確立するまでエラーになります:

```
mysql> SELECT 1;  
ERROR 1820 (HY000): You must reset your password using ALTER USER  
statement before executing this statement.
```

```
mysql> ALTER USER USER() IDENTIFIED BY 'password';  
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> SELECT 1;  
+---+  
| 1 |  
+---+  
| 1 |  
+---+  
1 row in set (0.00 sec)
```

クライアントがパスワードをリセットすると、サーバーはセッションの通常のアクセスと、そのアカウントを使用する後続の接続をリストアします。管理ユーザーがアカウントパスワードをリセットすることもできますが、そのアカ

ワントの既存の制限付きセッションは制限されたままになります。アカウントを使用するクライアントは、ステートメントを正常に実行する前に切断して再接続する必要があります。

注記

期限切れのパスワードは、現在の値に設定することで「reset」で使用できますが、適切なポリシーとして、別のパスワードを選択することをお勧めします。DBAは、適切なパスワード再利用ポリシーを確立することで、非キューを強制できます。[パスワード再利用ポリシー](#)を参照してください。

パスワード再利用ポリシー

MySQLでは、以前のパスワードの再利用に制限を適用できます。再利用制限は、パスワード変更の数、経過時間、またはその両方に基づいて設定できます。再利用ポリシーはグローバルに確立でき、個々のアカウントはグローバルポリシーに従うように設定することも、特定のアカウントごとの動作でグローバルポリシーをオーバーライドするように設定することもできます。

アカウントのパスワード履歴は、過去に割り当てられたパスワードで構成されます。MySQLでは、新しいパスワードがこの履歴から選択されないように制限できます：

- アカウントがパスワード変更数に基づいて制限されている場合、指定された数の最新のパスワードから新しいパスワードを選択することはできません。たとえば、パスワード変更の最小数が3に設定されている場合、新しいパスワードは最新の3つのパスワードと同じにできません。
- 経過時間に基づいてアカウントが制限されている場合、履歴内の指定した日数より新しいパスワードから新しいパスワードを選択することはできません。たとえば、パスワードの再利用間隔が60に設定されている場合、新しいパスワードは過去60日以内に選択されたパスワードの中にあってはなりません。

注記

空のパスワードはパスワード履歴にカウントされず、いつでも再利用される可能性があります。

パスワード再利用ポリシーをグローバルに確立するには、`password_history` および `password_reuse_interval` システム変数を使用します。

例:

- 365日より新しい過去6つのパスワードまたはパスワードの再利用を禁止するには、サーバーの `my.cnf` ファイルに次の行を挿入します:

```
[mysqld]
password_history=6
password_reuse_interval=365
```

- 実行時に変数を設定して永続化するには、次のようなステートメントを使用します:

```
SET PERSIST password_history = 6;
SET PERSIST password_reuse_interval = 365;
```

`SET PERSIST` は、実行中の MySQL インスタンスの値を設定します。また、後続のサーバー再起動に引き継ぐための値も保存されます。[セクション13.7.6.1「変数代入のSET構文」](#)を参照してください。後続の再起動に引き継ぐことなく、実行中の MySQL インスタンスの値を変更するには、`PERSIST` ではなく `GLOBAL` キーワードを使用します。

グローバルパスワード再利用ポリシーは、オーバーライドするように設定されていないすべてのアカウントに適用されます。個々のアカウントのポリシーを設定するには、`CREATE USER` および `ALTER USER` ステートメントの `PASSWORD HISTORY` および `PASSWORD REUSE INTERVAL` オプションを使用します。[セクション13.7.1.3「CREATE USER ステートメント」](#) および [セクション13.7.1.1「ALTER USER ステートメント」](#)を参照してください。

アカウント固有のステートメントの例:

- 再利用を許可する前に、少なくとも5つのパスワード変更が必要です:


```
CREATE USER 'jeffrey'@'localhost' PASSWORD HISTORY 5;  
ALTER USER 'jeffrey'@'localhost' PASSWORD HISTORY 5;
```

この履歴長オプションは、ステートメントで指定されたすべてのアカウントのグローバルポリシーをオーバーライドします。

- 再利用を許可する前に 365 日以上経過する必要があります:

```
CREATE USER 'jeffrey'@'localhost' PASSWORD REUSE INTERVAL 365 DAY;  
ALTER USER 'jeffrey'@'localhost' PASSWORD REUSE INTERVAL 365 DAY;
```

この time-elapsed オプションは、ステートメントで指定されたすべてのアカウントのグローバルポリシーをオーバーライドします。

- 両方のタイプの再利用制限を組み合わせるには、[PASSWORD HISTORY](#) と [PASSWORD REUSE INTERVAL](#) を組み合わせて使用します:

```
CREATE USER 'jeffrey'@'localhost'  
PASSWORD HISTORY 5  
PASSWORD REUSE INTERVAL 365 DAY;  
ALTER USER 'jeffrey'@'localhost'  
PASSWORD HISTORY 5  
PASSWORD REUSE INTERVAL 365 DAY;
```

これらのオプションは、ステートメントで指定されたすべてのアカウントのグローバルポリシー再利用制限をオーバーライドします。

- 両方のタイプの再利用制限について、グローバルポリシーに従います:

```
CREATE USER 'jeffrey'@'localhost'  
PASSWORD HISTORY DEFAULT  
PASSWORD REUSE INTERVAL DEFAULT;  
ALTER USER 'jeffrey'@'localhost'  
PASSWORD HISTORY DEFAULT  
PASSWORD REUSE INTERVAL DEFAULT;
```

パスワード検証必須ポリシー

MySQL 8.0.13 の時点では、置換する現在のパスワードを指定して、アカウントパスワードの変更の試行を検証する必要があります。これにより、DBA は、現在のパスワードを知らなくてもユーザーがパスワードを変更できないようにできます。このような変更は、たとえば、あるユーザーがログアウトせずに一時的に端末セッションから離れると、悪質なユーザーが元のユーザー MySQL パスワードを変更するためにセッションを使用した場合に発生する可能性があります。これには、次のような不適切な結果が生じる可能性があります:

- 管理者がアカウントパスワードをリセットするまで、元のユーザーは MySQL にアクセスできなくなります。
- パスワードのリセットが発生するまで、悪質なユーザーは無害なユーザー変更の資格証明を使用して MySQL にアクセスできます。

パスワード検証ポリシーはグローバルに確立でき、個々のアカウントは、グローバルポリシーに従うか、特定のアカウントごとの動作でグローバルポリシーをオーバーライドするように設定できます。

アカウントごとに、その `mysql.user` 行は、パスワード変更試行のために現在のパスワードの検証を必要とするアカウント固有の設定があるかどうかを示します。この設定は、[CREATE USER](#) および [ALTER USER](#) ステートメントの [PASSWORD REQUIRE](#) オプションによって確立されます:

- アカウント設定が [PASSWORD REQUIRE CURRENT](#) の場合、パスワード変更では現在のパスワードを指定する必要があります。
- アカウント設定が [PASSWORD REQUIRE CURRENT OPTIONAL](#) の場合、パスワードの変更では現在のパスワードを指定する必要はありません。
- アカウント設定が [PASSWORD REQUIRE CURRENT DEFAULT](#) の場合、`password_require_current` システム変数によってアカウントの検証必須ポリシーが決定されます:
 - `password_require_current` が有効な場合、パスワード変更では現在のパスワードを指定する必要があります。

- `password_require_current` が無効になっている場合、パスワードの変更では現在のパスワードを指定する必要はありません。

つまり、アカウント設定が `PASSWORD REQUIRE CURRENT DEFAULT` でない場合、アカウント設定は `password_require_current` システム変数によって設定されたグローバルポリシーよりも優先されます。それ以外の場合、アカウントは `password_require_current` 設定に従います。

デフォルトでは、パスワード検証はオプションです: `password_require_current` が無効になり、`PASSWORD REQUIRE` オプションなしで作成されたアカウントは `PASSWORD REQUIRE CURRENT DEFAULT` にデフォルト設定されます。

次のテーブルは、アカウントごとの設定が `password_require_current` システム変数値とどのように相互作用して、アカウントパスワードの検証が必要なポリシーを決定するかを示しています。

表 6.10 パスワード検証ポリシー

アカウントごとの設定	<code>password_require_current</code> システム変数	パスワードの変更には現在のパスワードが必要ですか。
<code>PASSWORD REQUIRE CURRENT</code>	OFF	はい
<code>PASSWORD REQUIRE CURRENT</code>	ON	はい
<code>PASSWORD REQUIRE CURRENT OPTIONAL</code>	OFF	いいえ
<code>PASSWORD REQUIRE CURRENT OPTIONAL</code>	ON	いいえ
<code>PASSWORD REQUIRE CURRENT DEFAULT</code>	OFF	いいえ
<code>PASSWORD REQUIRE CURRENT DEFAULT</code>	ON	はい

注記

特権ユーザーは、検証必須ポリシーに関係なく、現在のパスワードを指定せずに任意のアカウントパスワードを変更できます。特権ユーザーは、`mysql` システムデータベースに対するグローバル `CREATE USER` 権限または `UPDATE` 権限を持つユーザーです。

パスワード検証ポリシーをグローバルに確立するには、`password_require_current` システム変数を使用します。デフォルト値は `OFF` であるため、アカウントパスワードの変更で現在のパスワードを指定する必要はありません。

例:

- パスワード変更で現在のパスワードを指定する必要があるグローバルポリシーを確立するには、サーバー `my.cnf` ファイルで次の行を使用してサーバーを起動します:

```
[mysqld]
password_require_current=ON
```

- 実行時に `password_require_current` を設定および永続化するには、次のいずれかのステートメントを使用します:

```
SET PERSIST password_require_current = ON;
SET PERSIST password_require_current = OFF;
```

`SET PERSIST` は、実行中の MySQL インスタンスの値を設定します。また、後続のサーバー再起動に引き継ぐための値も保存されます。セクション13.7.6.1「変数代入の SET 構文」を参照してください。後続の再起動に引き継ぐことなく、実行中の MySQL インスタンスの値を変更するには、`PERSIST` ではなく `GLOBAL` キーワードを使用します。

グローバルパスワード検証必須ポリシーは、オーバーライドするように設定されていないすべてのアカウントに適用されます。個々のアカウントのポリシーを設定するには、`CREATE USER` および `ALTER USER` ステートメントの `PASSWORD REQUIRE` オプションを使用します。セクション13.7.1.3「CREATE USER ステートメント」およびセクション13.7.1.1「ALTER USER ステートメント」を参照してください。

アカウント固有のステートメントの例:

- パスワードの変更で現在のパスワードを指定する必要があります:

```
CREATE USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT;  
ALTER USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT;
```

この検証オプションは、ステートメントで指定されたすべてのアカウントのグローバルポリシーをオーバーライドします。

- パスワードの変更で現在のパスワードを指定する必要はありません (現在のパスワードを指定する必要があります):

```
CREATE USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT OPTIONAL;  
ALTER USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT OPTIONAL;
```

この検証オプションは、ステートメントで指定されたすべてのアカウントのグローバルポリシーをオーバーライドします。

- ステートメントで指定されたすべてのアカウントのグローバルパスワード検証必須ポリシーに従います:

```
CREATE USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT DEFAULT;  
ALTER USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT DEFAULT;
```

ユーザーが **ALTER USER** または **SET PASSWORD** ステートメントを使用してパスワードを変更すると、現在のパスワードの検証が行われます。この例では、**SET PASSWORD** よりも優先される **ALTER USER** を使用していますが、ここで説明する原則は両方のステートメントで同じです。

password-change ステートメントでは、置換する現在のパスワードを **REPLACE** 句で指定します。例:

- 現在のユーザーパスワードを変更します:

```
ALTER USER USER() IDENTIFIED BY 'auth_string' REPLACE 'current_auth_string';
```

- 名前付きユーザーのパスワードを変更します:

```
ALTER USER 'jeffrey'@'localhost'  
IDENTIFIED BY 'auth_string'  
REPLACE 'current_auth_string';
```

- 名前付きユーザー認証プラグインおよびパスワードを変更します:

```
ALTER USER 'jeffrey'@'localhost'  
IDENTIFIED WITH caching_sha2_password BY 'auth_string'  
REPLACE 'current_auth_string';
```

REPLACE 句は次のように機能します:

- 現在のパスワードを指定するためにアカウントのパスワード変更が必要な場合は、変更しようとしているユーザーが実際に現在のパスワードを知っていることを確認するために、**REPLACE** を指定する必要があります。
- アカウントのパスワード変更で現在のパスワードを指定する必要がない場合、**REPLACE** はオプションです。
- **REPLACE** が指定されている場合は、正しい現在のパスワードを指定する必要があります。そうしないと、エラーが発生します。これは、**REPLACE** がオプションの場合でも当てはまります。
- **REPLACE** は、現在のユーザーのアカウントパスワードを変更する場合にのみ指定できます。(つまり、前述の例では、現在のユーザーが **jeffrey** でないかぎり、**jeffrey** のアカウントを明示的に指定するステートメントは失敗します。) これは、特権ユーザーが別のユーザーに対して変更を試みた場合でも当てはまりますが、そのようなユーザーは **REPLACE** を指定せずに任意のパスワードを変更できます。
- クリアテキストパスワードが書き込まれないように、バイナリログから **REPLACE** が省略されています。

デュアルパスワードのサポート

MySQL 8.0.14 の時点では、ユーザーアカウントは、プライマリパスワードおよびセカンダリパスワードとして指定されたデュアルパスワードを持つことができます。デュアルパスワード機能により、次のようなシナリオで資格証明の変更をシームレスに実行できます:

- システムには多数の MySQL サーバーがあり、レプリケーションが含まれる可能性があります。
- 複数のアプリケーションが異なる MySQL サーバーに接続します。
- 定期的な資格証明の変更は、アプリケーションがサーバーに接続するために使用するアカウントに対して行う必要があります。

アカウントが単一のパスワードのみを許可されている場合に、前述のタイプのシナリオで資格証明の変更を実行する方法を検討してください。この場合、アカウントパスワードが変更されてすべてのサーバーに伝播されるタイミング、およびアカウントを使用するすべてのアプリケーションが新しいパスワードを使用するように更新されるタイミングには、緊密な連携が必要です。このプロセスには、サーバーまたはアプリケーションを使用できない停止時間が含まれる場合があります。

デュアルパスワードを使用すると、資格証明の変更をフェーズでより簡単に行うことができ、密接な連携や停止時間なしで行うことができます:

1. 影響を受けるアカウントごとに、現在のパスワードをセカンダリパスワードとして保持して、サーバーに新しいプライマリパスワードを設定します。これにより、サーバーは各アカウントのプライマリパスワードまたはセカンダリパスワードを認識できますが、アプリケーションは以前と同じパスワード (現在はセカンダリパスワード) を使用して引き続きサーバーに接続できます。
2. パスワード変更がすべてのサーバーに伝播されたら、影響を受けるアカウントを使用するアプリケーションを変更し、アカウントプライマリパスワードを使用して接続します。
3. すべてのアプリケーションがセカンダリパスワードからプライマリパスワードに移行されると、セカンダリパスワードは不要になり、破棄できます。この変更がすべてのサーバーに伝播された後は、各アカウントのプライマリパスワードのみを使用して接続できます。資格証明の変更が完了しました。

MySQL には、セカンダリパスワードを保存および破棄する構文を使用したデュアルパスワード機能が実装されています:

- `ALTER USER` および `SET PASSWORD` ステートメントの `RETAIN CURRENT PASSWORD` 句では、新しいプライマリパスワードを割り当てるときに、アカウントの現在のパスワードがセカンダリパスワードとして保存されます。
- `ALTER USER` の `DISCARD OLD PASSWORD` 句では、アカウントセカンダリパスワードが破棄され、プライマリパスワードのみが残されます。

前述の資格証明変更シナリオでは、アプリケーションがサーバーに接続するために `'appuser1'@'host1.example.com'` という名前のアカウントを使用し、アカウントパスワードを `'password_a'` から `'password_b'` に変更するとします。

この資格証明の変更を実行するには、次のように `ALTER USER` を使用します:

1. レプリカではない各サーバーで、`'password_b'` を新しい `appuser1` プライマリパスワードとして確立し、現在のパスワードをセカンダリパスワードとして保持します:

```
ALTER USER 'appuser1'@'host1.example.com'  
IDENTIFIED BY 'password_b'  
RETAIN CURRENT PASSWORD;
```

2. パスワード変更がシステム全体ですべてのレプリカにレプリケートされるのを待ちます。
3. `'password_a'` ではなく `'password_b'` のパスワードを使用してサーバーに接続するように、`appuser1` アカウントを使用する各アプリケーションを変更します。
4. この時点で、セカンダリパスワードは不要になりました。レプリカではない各サーバーで、セカンダリパスワードを破棄します:

```
ALTER USER 'appuser1'@'host1.example.com'  
DISCARD OLD PASSWORD;
```

5. 廃棄パスワードの変更がすべてのレプリカにレプリケートされると、資格証明の変更が完了します。

`RETAIN CURRENT PASSWORD` 句および `DISCARD OLD PASSWORD` 句には、次の効果があります:

- **RETAIN CURRENT PASSWORD** は、アカウントの現在のパスワードをセカンダリパスワードとして保持し、既存のセカンダリパスワードを置き換えます。新しいパスワードはプライマリパスワードになりますが、クライアントはアカウントを使用して、プライマリパスワードまたはセカンダリパスワードのいずれかを使用してサーバーに接続できます。(例外: **ALTER USER** ステートメントまたは **SET PASSWORD** ステートメントで指定された新しいパスワードが空の場合、**RETAIN CURRENT PASSWORD** が指定されていてもセカンダリパスワードも空になります。)
- プライマリパスワードが空のアカウントに **RETAIN CURRENT PASSWORD** を指定すると、ステートメントは失敗します。
- アカウントにセカンダリパスワードがあり、**RETAIN CURRENT PASSWORD** を指定せずにプライマリパスワードを変更した場合、セカンダリパスワードは変更されません。
- **ALTER USER** の場合、アカウントに割り当てられた認証プラグインを変更すると、セカンダリパスワードは破棄されます。認証プラグインを変更し、**RETAIN CURRENT PASSWORD** も指定すると、ステートメントは失敗します。
- **ALTER USER** では、セカンダリパスワードが存在する場合、**DISCARD OLD PASSWORD** は破棄します。アカウントはプライマリパスワードのみを保持し、クライアントはプライマリパスワードのみを使用してサーバーに接続するためにアカウントを使用できます。

セカンダリパスワードを変更するステートメントには、次の権限が必要です:

- 自分のアカウントに適用される **ALTER USER** および **SET PASSWORD** ステートメントに **RETAIN CURRENT PASSWORD** または **DISCARD OLD PASSWORD** 句を使用するには、**APPLICATION_PASSWORD_ADMIN** 権限が必要です。ほとんどのユーザーは 1 つのパスワードのみを必要とするため、自分のセカンダリパスワードを操作するには権限が必要です。
- アカウントがすべてのアカウントのセカンダリパスワードの操作を許可される場合は、**APPLICATION_PASSWORD_ADMIN** ではなく **CREATE USER** 権限を付与する必要があります。

ランダムパスワード生成

MySQL 8.0.18 では、**CREATE USER**、**ALTER USER** および **SET PASSWORD** ステートメントに、明示的に管理者指定のリテラルパスワードを要求するかわりに、ユーザーアカウントのランダムパスワードを生成する機能があります。構文の詳細は、各ステートメントの説明を参照してください。このセクションでは、生成されるランダムパスワードに共通する特性について説明します。

デフォルトでは、生成されるランダムパスワードの長さは 20 文字です。この長さは、5 から 255 の範囲の **generated_random_password_length** システム変数によって制御されます。

ステートメントがランダムなパスワードを生成するアカウントごとに、このステートメントは、アカウント認証プラグイン用に適切にハッシュされたパスワードを **mysql.user** システムテーブルに格納します。このステートメントは、ステートメントを実行するユーザーまたはアプリケーションが使用できるように、結果セットの行にクリアテキストのパスワードも返します。結果セットカラムの名前は **user**、**host** および **generated password** で、**mysql.user** システムテーブルの影響を受ける行を識別するユーザー名とホスト名の値、およびクリアテキストで生成されたパスワードを示します。

```
mysql> CREATE USER
'u1'@'localhost' IDENTIFIED BY RANDOM PASSWORD,
'u2'@'%example.com' IDENTIFIED BY RANDOM PASSWORD,
'u3'@'%org' IDENTIFIED BY RANDOM PASSWORD;
+-----+-----+
| user | host      | generated password |
+-----+-----+
| u1   | localhost | BA;42VpXqQ@i+y{&TDFF |
| u2   | %example.com | YX5>XRAJRP@>sn9azmD4 |
| u3   | %org      | ;GfD44I,)C}PI/6)4TwZ |
+-----+-----+
mysql> ALTER USER
'u1'@'localhost' IDENTIFIED BY RANDOM PASSWORD,
'u2'@'%example.com' IDENTIFIED BY RANDOM PASSWORD;
+-----+-----+
| user | host      | generated password |
+-----+-----+
```

```

| u1 | localhost | yhXBrBp.;Y6abBje_UWr |
| u2 | %example.com | >M-vmjp9DTY6}hkp,RcC |
+-----+-----+
mysql> SET PASSWORD FOR 'u3'@'%org' TO RANDOM;
+-----+-----+
| user | host | generated password |
+-----+-----+
| u3 | %org | o(.oNn)d;FC<vJIDg9M |
+-----+-----+

```

アカウントのランダムパスワードを生成する `CREATE USER`、`ALTER USER` または `SET PASSWORD` ステートメントは、`IDENTIFIED WITH auth_plugin AS 'auth_string'` 句を含む `CREATE USER` または `ALTER USER` ステートメントとしてバイナリログに書き込まれます。`auth_plugin` はアカウント認証プラグインで、`'auth_string'` はアカウントのハッシュパスワード値です。

`validate_password` コンポーネントがインストールされている場合、実装されているポリシーは生成されたパスワードに影響しません。(パスワード検証の目的は、人間がより適切なパスワードを作成できるようにすることです。)

失敗したログインラッキングと一時アカウントロック

MySQL 8.0.19 の時点では、管理者は、連続するログイン失敗が多すぎると一時アカウントロックが発生するようにユーザーアカウントを構成できます。

このコンテキストの「「ログイン失敗」」は、接続試行中にクライアントが正しいパスワードを指定できないことを意味します。不明なユーザーまたはネットワークの問題などの理由で接続に失敗することは含まれません。デュアルパスワードを持つアカウント ([デュアルパスワードのサポート](#) を参照) の場合、どちらのアカウントパスワードも正しいものとしてカウントされます。

必要なログイン失敗の数とロック時間は、`CREATE USER` および `ALTER USER` ステートメントの `FAILED_LOGIN_ATTEMPTS` および `PASSWORD_LOCK_TIME` オプションを使用してアカウントごとに構成できます。例:

```

CREATE USER 'u1'@'localhost' IDENTIFIED BY 'password'
  FAILED_LOGIN_ATTEMPTS 3 PASSWORD_LOCK_TIME 3;

ALTER USER 'u2'@'localhost'
  FAILED_LOGIN_ATTEMPTS 4 PASSWORD_LOCK_TIME UNBOUNDED;

```

連続したログイン失敗が多すぎる場合、クライアントは次のようなエラーを受け取ります:

```

ERROR 3957 (HY000): Access denied for user 'user'.
Account is blocked for D day(s) (R day(s) remaining)
due to N consecutive failed logins.

```

次のオプションを使用します:

- `FAILED_LOGIN_ATTEMPTS N`

このオプションは、不正なパスワードを指定するアカウントログイン試行を追跡するかどうかを示します。`N` の数には、一時的なアカウントロックを引き起こす連続した不正なパスワードの数を指定します。

- `PASSWORD_LOCK_TIME {N | UNBOUNDED}`

このオプションは、連続して何回もログインしようとする間違ったパスワードが提供された後に、アカウントをロックする期間を示します。値は、アカウントがロックされたままになる日数を指定する `N` の数値、またはアカウントが一時的にロックされた状態になるとその状態の期間が無制限になり、アカウントがロック解除されるまで終了しないことを指定する `UNBOUNDED` です。ロック解除が行われる条件については、後で説明します。

各オプションに許可される `N` の値は、0 から 32767 の範囲です。値 0 を指定すると、オプションが無効になります。

ログイン失敗ラッキングと一時アカウントロックには、次の特性があります:

- 失敗したログイン追跡および一時ロックをアカウントに対して実行するには、その `FAILED_LOGIN_ATTEMPTS` オプションと `PASSWORD_LOCK_TIME` オプションの両方をゼロ以外にする必要があります。

- `CREATE USER` では、`FAILED_LOGIN_ATTEMPTS` または `PASSWORD_LOCK_TIME` が指定されていない場合、ステートメントで指定されたすべてのアカウントの暗黙的なデフォルト値は 0 です。これは、失敗したログイン、トラッキングおよび一時アカウントロックが無効であることを意味します。(これらの暗黙的なデフォルトは、失敗したログイン追跡の導入前に作成されたアカウントにも適用されます。)
- `ALTER USER` では、`FAILED_LOGIN_ATTEMPTS` または `PASSWORD_LOCK_TIME` が指定されていない場合、ステートメントで指定されたすべてのアカウントの値は変更されません。
- 一時アカウントロックを実行するには、パスワード障害が連続して発生する必要があります。失敗したログインの `FAILED_LOGIN_ATTEMPTS` 値に達する前に正常にログインすると、失敗カウントがリセットされます。たとえば、`FAILED_LOGIN_ATTEMPTS` が 4 で、3 つの連続したパスワード障害が発生した場合、ロックを開始するにはさらに障害が発生する必要があります。ただし、次のログインが成功した場合は、ロックに 4 つの連続した失敗が再度必要になるように、アカウントの失敗ログインカウントがリセットされます。
- 一時ロックが開始されると、ロック期間が経過するか、次の説明に示す `account-reset` メソッドのいずれかによってアカウントのロックが解除されるまで、正しいパスワードを使用してもログインは成功しません。

サーバーは、付与テーブルを読み取るときに、失敗したログイン追跡が有効かどうか、現在アカウントが一時的にロックされているかどうか、ロックが開始された場合はロックが開始された場合、およびアカウントがロックされていない場合に一時的なロックが発生するまでの失敗数に関する状態情報をアカウントごとに初期化します。

アカウント状態情報をリセットできます。つまり、失敗したログイン数がリセットされ、アカウントが現在一時的にロックされている場合はロック解除されます。アカウントのリセットは、すべてのアカウントに対してグローバルにすることも、アカウントごとに行うこともできます:

- すべてのアカウントのグローバルリセットは、次のいずれかの状況で発生します:
 - サーバーの再起動。
 - `FLUSH PRIVILEGES` の実行。(`--skip-grant-tables` を使用してサーバーを起動すると、付与テーブルが読み取られず、失敗したログイン追跡が無効になります。この場合、`FLUSH PRIVILEGES` を最初に行うと、すべてのアカウントのリセットに加えて、サーバーが付与テーブルを読み取り、失敗したログイン追跡を有効にします。)
- アカウントごとのリセットは、次のいずれかの状況で発生します:
 - アカウントのログインに成功しました。
 - ロック期間が経過します。この場合、ログイン失敗回数は次のログイン試行時にリセットされます。
 - `FAILED_LOGIN_ATTEMPTS` または `PASSWORD_LOCK_TIME` (あるいはその両方) を任意の値 (現在のオプション値を含む) に設定するアカウントに対する `ALTER USER` ステートメントの実行、またはアカウントに対する `ALTER USER ... UNLOCK` ステートメントの実行。

アカウントの他の `ALTER USER` ステートメントは、現在の失敗ログイン数またはロック状態には影響しません。

失敗したログイン、トラッキングは、資格証明のチェックに使用されるログインアカウントに関連付けられます。ユーザープロキシが使用されている場合、プロキシユーザーではなくプロキシユーザーに対してトラッキングが行われます。つまり、トラッキングは、`CURRENT_USER()` で示されるアカウントではなく、`USER()` で示されるアカウントに関連付けられます。プロキシユーザーとプロキシユーザーの区別の詳細は、[セクション 6.2.18 「プロキシユーザー」](#) を参照してください。

6.2.16 期限切れパスワードのサーバー処理

MySQL にはパスワードの有効期限機能が用意されており、データベース管理者はパスワードのリセットをユーザーに要求できます。パスワードは手動で期限切れにすることも、自動期限切れのポリシーに基づいて期限切れにすることもできます ([セクション 6.2.15 「パスワード管理」](#) を参照)。

`ALTER USER` ステートメントは、アカウントパスワードの有効期限を有効にします。例:

```
ALTER USER 'myuser'@'localhost' PASSWORD EXPIRE;
```


期限切れのパスワードを持つアカウントを使用する接続ごとに、サーバーはクライアントを切断するか、クライアントを「サンドボックスモード、」に制限します。「サンドボックスモード、」では、クライアントは期限切れのパスワードのリセットに必要な操作のみを実行できます。サーバーによって実行されるアクションは、後で説明するように、クライアントとサーバーの両方の設定によって異なります。

サーバーがクライアントを切断すると、`ER_MUST_CHANGE_PASSWORD_LOGIN` エラーが返されます。

```
shell> mysql -u myuser -p
Password: *****
ERROR 1862 (HY000): Your password has expired. To log in you must
change it using a client that supports expired passwords.
```

サーバーがクライアントをサンドボックスモードに制限する場合、クライアントセッション内では次の操作が許可されます:

- クライアントは、`ALTER USER` または `SET PASSWORD` を使用してアカウントパスワードをリセットできます。これが完了すると、サーバーはセッションおよびアカウントを使用する後続の接続の通常のアクセスをリストアします。

注記

期限切れのパスワードは、現在の値に設定することで「reset」で使用できますが、適切なポリシーとして、別のパスワードを選択することをお勧めします。DBAは、適切なパスワード再利用ポリシーを確立することで、非キューを強制できます。[パスワード再利用ポリシー](#)を参照してください。

- クライアントは `SET` ステートメントを使用できます。

セッション内で許可されていない操作の場合、サーバーは `ER_MUST_CHANGE_PASSWORD` エラーを返します。

```
mysql> USE performance_schema;
ERROR 1820 (HY000): You must reset your password using ALTER USER
statement before executing this statement.

mysql> SELECT 1;
ERROR 1820 (HY000): You must reset your password using ALTER USER
statement before executing this statement.
```

このような呼出しはデフォルトでサンドボックスモードになるため、通常は `mysql` クライアントの対話型呼出しで行われます。通常の機能を再開するには、新しいパスワードを選択します。

`mysql` クライアントの非対話型呼出し (バッチモードなど) の場合、パスワードが期限切れになると、サーバーは通常クライアントを切断します。(サンドボックスモードで許可されたステートメントを使用して) パスワードを変更できるように、非対話型の `mysql` 呼出しが接続されたままにするには、`mysql` コマンドに `--connect-expired-password` オプションを追加します。

前述のように、サーバーが期限切れパスワードクライアントを切断するか、サンドボックスモードに制限するかは、クライアントとサーバーの設定の組合せによって決まります。次の説明では、関連する設定と、それらがどのように相互作用するのかについて記述します。

注記

この説明は、パスワードが期限切れのアカウントにのみ適用されます。クライアントが期限切れでないパスワードを使用して接続すれば、サーバーはクライアントを通常どおりに処理します。

クライアント側では、特定のクライアントが期限切れパスワードに対してサンドボックスモードを処理できるかどうかを示します。C クライアントライブラリを使用するクライアントの場合、これを実行するための方法が2つあります。

- 接続前に `MYSQL_OPT_CAN_HANDLE_EXPIRED_PASSWORDS` フラグを `mysql_options()` に渡します。

```
bool arg = 1;
mysql_options(mysql,
    MYSQL_OPT_CAN_HANDLE_EXPIRED_PASSWORDS,
```

```
&arg);
```

これは `mysql` クライアント内で使用される手法であり、対話形式または `--connect-expired-password` オプションを使用して `MYSQL_OPT_CAN_HANDLE_EXPIRED_PASSWORDS` を呼び出すことができます。

- 接続時に `CLIENT_CAN_HANDLE_EXPIRED_PASSWORDS` フラグを `mysql_real_connect()` に渡します:

```
MYSQL mysql;
mysql_init(&mysql);
if (!mysql_real_connect(&mysql,
    host, user, password, db,
    port, unix_socket,
    CLIENT_CAN_HANDLE_EXPIRED_PASSWORDS))
{
    ... handle error ...
}
```

その他の MySQL コネクタには、サンドボックスモードを処理する準備ができていないことを示す独自の規則が用意されています。目的のコネクタのドキュメントを参照してください。

サーバー側では、クライアントが期限切れパスワードを処理できることを示している場合、サーバーはサンドボックスモードに移行します。

クライアントが期限切れパスワードを処理できない場合 (または、そのように示すことができない古いバージョンのクライアントライブラリを使用している場合)、サーバーのアクションは `disconnect_on_expired_password` システム変数の値によって異なります。

- `disconnect_on_expired_password` が有効になっている (デフォルト) 場合、サーバーはクライアントを切断し、`ER_MUST_CHANGE_PASSWORD_LOGIN` エラーを返します。
- `disconnect_on_expired_password` が無効になっている場合、サーバーはクライアントをサンドボックスモードに移行します。

6.2.17 プラグブル認証

クライアントが MySQL サーバーに接続すると、サーバーはクライアントおよびクライアントホストによって指定されたユーザー名を使用して、`mysql.user` システムテーブルから適切なアカウント行を選択します。次に、サーバーはクライアントを認証し、どの認証プラグインがクライアントに適用されるかをアカウント行から決定します:

- サーバーがプラグインを検出できない場合は、エラーが発生し、接続の試行は拒否されます。
- それ以外の場合、サーバーはそのプラグインを呼び出してユーザーを認証し、ユーザーが正しいパスワードを指定して接続を許可されているかどうかを示すステータスをサーバーに返します。

プラグブル認証により、次の重要な機能が有効になります:

- 認証方式の選択。 プラグブル認証を使用すると、DBA は個々の MySQL アカウントに使用される認証方式を簡単に選択および変更できます。
- 外部認証。 プラグブル認証を使用すると、クライアントは、`mysql.user` システムテーブル以外の場所に資格証明を格納する認証方法に適した資格証明を使用して MySQL サーバーに接続できます。たとえば、PAM、Windows のログオン ID、LDAP、Kerberos などの外部認証方式を使用するプラグインを作成できます。
- プロキシユーザー。 ユーザーが接続を許可されている場合、認証プラグインは接続ユーザーの名前とは異なるユーザー名をサーバーに返して、接続ユーザーが別のユーザー (プロキシユーザー) のプロキシであることを示すことができます。接続が継続している間、プロキシユーザーはアクセス制御のためにプロキシユーザーの権限を持つものとして扱われます。実際に、あるユーザーは別のユーザーを偽装します。詳細については、[セクション 6.2.18「プロキシユーザー」](#)を参照してください。

注記

`--skip-grant-tables` オプションを付けてサーバーを起動した場合、サーバーはクライアント認証を実行せず、任意のクライアントが接続することを許可するため、認証プラグインはロードされたとしても使用されません。これはセキュアではないため、`--skip-grant-tables`

オプションを使用してサーバーを起動すると、`skip_networking` を有効にしてリモート接続も無効になります。

- [使用可能な認証プラグイン](#)
- [認証プラグインの使用](#)
- [認証プラグインクライアント/サーバーの互換性](#)
- [認証プラグインコネクタ - 書込みに関する考慮事項](#)
- [プラグブルな認証の制約](#)

使用可能な認証プラグイン

MySQL 8.0 には次の認証プラグインが用意されています:

- ネイティブ認証を実行するプラグイン。つまり、MySQL でプラグブル認証が導入される前から使用されているパスワードハッシュ方式に基づく認証です。 `mysql_native_password` プラグインは、このネイティブパスワードハッシュ方式に基づいて認証を実装します。 [セクション6.4.1.1「ネイティブプラグブル認証」](#)を参照してください。
- SHA-256 パスワードハッシュを使用して認証を実行するプラグイン。これは、ネイティブ認証で実現できるよりも強力な暗号化です。 [セクション6.4.1.3「SHA-256 プラグブル認証」](#) および [セクション6.4.1.2「SHA-2 プラグブル認証のキャッシュ」](#)を参照してください。
- ハッシュ化または暗号化を行わずに、サーバーにパスワードを送信するクライアント側のプラグイン。このプラグインは、クライアントユーザーが指定したパスワードに正確にアクセスする必要があるサーバー側プラグインと組み合わせて使用されます。 [セクション6.4.1.4「クライアント側クリアテキストプラグブル認証」](#)を参照してください。
- PAM (Pluggable Authentication Modules) を使用して外部認証を実行し、MySQL Server が PAM を使用して MySQL ユーザーを認証できるようにするプラグイン。このプラグインでは、プロキシユーザーもサポートされています。 [セクション6.4.1.5「PAM プラグブル認証」](#)を参照してください。
- Windows で外部認証を実行するプラグイン。これを使用すると、MySQL サーバーがネイティブの Windows サービスを使用して、クライアント接続を認証できます。Windows にログインしたユーザーは、追加のパスワードを指定せずに、自分の環境内の情報に基づいて MySQL クライアントプログラムからサーバーに接続できます。このプラグインでは、プロキシユーザーもサポートされています。 [セクション6.4.1.6「Windows プラグブル認証」](#)を参照してください。
- LDAP (Lightweight Directory Access Protocol) を使用して認証を実行し、X.500 などのディレクトリサービスにアクセスして MySQL ユーザーを認証するプラグイン。これらのプラグインは、プロキシユーザーもサポートしています。 [セクション6.4.1.7「LDAP プラグブル認証」](#)を参照してください。
- それを使用するアカウントへのすべてのクライアント接続を妨げるプラグイン。このプラグインのユースケースには、直接ログインを許可しないが、通常のユーザーに権限を公開せずに昇格された権限を持つストアドプログラムおよびビューを実行できる必要があるプロキシアカウントおよびアカウントを介してのみアクセスされるプロキシアカウントが含まれます。 [セクション6.4.1.8「ログインなしのプラグブル認証」](#)を参照してください。
- Unix ソケットファイルを使用してローカルホストから接続するクライアントを認証するプラグイン。 [セクション6.4.1.9「ソケットピア資格証明プラグブル認証」](#)を参照してください。
- アカウント資格証明をチェックし、成功または失敗をサーバーエラーログに記録するテストプラグイン。このプラグインは、テストおよび開発のために、認証プラグインを作成する方法を示す例として使用されます。 [セクション6.4.1.10「プラグブル認証のテスト」](#)を参照してください。

注記

プラグブル認証の使用に対する現在の制約 (どのコネクタがどのプラグインをサポートしているのかなど) については、 [プラグブルな認証の制約](#)を参照してください。

サードパーティー製コネクタの開発者は、コネクタがプラグブル認証機能を活用できる範囲と、より準拠させるために実行する手順を確認するために、そのセクションを読むべきです。

独自の認証プラグインを作成することに関心がある場合は、[Writing Authentication Plugins](#)を参照してください。

認証プラグインの使用

このセクションでは、認証プラグインをインストールおよび使用するための一般的な手順を示します。特定のプラグインに固有の手順については、[セクション6.4.1「認証プラグイン」](#)でそのプラグインについて説明しているセクションを参照してください。

通常、プラグブル認証では、サーバー側とクライアント側で対応するプラグインのペアが使用されるため、次のような特定の認証方法を使用します:

- 必要に応じて、適切なプラグインを含むプラグインライブラリをインストールします。サーバーがサーバーホストを使用してクライアント接続を認証できるように、サーバー側プラグインを含むライブラリをインストールします。同様に、クライアントホストごとに、クライアントプログラムで使用するクライアント側プラグインを含むライブラリをインストールします。組込みの認証プラグインをインストールする必要はありません。
- 作成する MySQL アカウントごとに、認証に使用する適切なサーバー側プラグインを指定します。アカウントがデフォルトの認証プラグインを使用する場合、`account-creation` ステートメントでプラグインを明示的に指定する必要はありません。`default_authentication_plugin` システム変数は、デフォルトの認証プラグインを構成します。
- クライアントが接続すると、サーバー側プラグインはクライアントプログラムに認証に使用するクライアント側プラグインを通知します。

アカウントがサーバーとクライアントプログラムの両方のデフォルトの認証方式を使用している場合、サーバーはクライアント側のプラグインが使用するクライアントと通信する必要はなく、クライアント/サーバーのネゴシエーションでラウンドトリップを回避できます。

`mysql` や `mysqladmin` などの標準 MySQL クライアントの場合、`--default-auth=plugin_name` オプションは、プログラムが使用できるクライアント側プラグインに関するヒントとしてコマンド行で指定できますが、ユーザーアカウントに関連付けられたサーバー側プラグインが異なるクライアント側プラグインを必要とする場合、サーバーはこれをオーバーライドします。

クライアントプログラムがクライアント側プラグインライブラリファイルを見つけられない場合は、プラグインライブラリディレクトリの場所を示す `--plugin-dir=dir_name` オプションを指定します。

認証プラグインクライアント/サーバーの互換性

プラグブル認証を使用すると、MySQL アカウントの認証方式を柔軟に選択できますが、クライアントとサーバー間の認証プラグインの非互換性のためにクライアント接続を確立できない場合があります。

特定のサーバー上の特定のアカウントへのクライアント接続を成功させるための一般的な互換性の原則は、クライアントとサーバーの両方がアカウントに必要な認証 `method` をサポートしている必要があることです。認証方式は認証プラグインによって実装されるため、クライアントとサーバーは両方ともアカウントに必要な認証 `plugin` をサポートする必要があります。

認証プラグインの非互換性は様々な方法で発生する可能性があります。例:

- 5.7.22 以下の MySQL 5.7 クライアントを使用して、`caching_sha2_password` で認証される MySQL 8.0 サーバーアカウントに接続します。MySQL 8.0 で導入されたプラグインが 5.7 クライアントで認識されないため、これは失敗します。(この問題は、`caching_sha2_password` クライアント側サポートが MySQL クライアントライブラリおよびクライアントプログラムに追加されたときに、5.7.23 の時点で MySQL 5.7 で対処されます。)
- MySQL 5.5 クライアントを使用して、`sha256_password` で認証される MySQL 5.6 サーバーアカウントに接続します。MySQL 5.6 で導入されたプラグインが 5.5 クライアントで認識されないため、これは失敗します。
- MySQL 5.7 クライアントを使用して、`mysql_old_password` で認証する 5.7 より前のサーバーアカウントに接続します。これは、複数の理由で失敗します。まず、このような接続には `--secure-auth=0` が必要ですが、これはサポートされなくなりました。サポートされていても、5.7 クライアントは MySQL 5.7 で削除されたため、プラグインを認識しません。
- Community ディストリビューションの MySQL 5.7 クライアントを使用して、エンタープライズ専用 LDAP 認証プラグインのいずれかを使用して認証する MySQL 5.7 Enterprise サーバーアカウントに接続します。Community クライアントに Enterprise プラグインへのアクセス権がないため、これは失敗します。

一般に、これらの互換性の問題は、同じ MySQL ディストリビューションのクライアントとサーバー間で接続が確立された場合には発生しません。異なる MySQL シリーズのクライアントとサーバーの間で接続が行われると、問題が発生する可能性があります。これらの問題は、MySQL で新しい認証プラグインが導入されたとき、または古い認証プラグインが削除されたときに、開発プロセスに固有です。非互換性の可能性を最小限に抑えるには、サーバー、クライアントおよびコネクタを適時に定期的にアップグレードします。

認証プラグインコネクタ - 書込みに関する考慮事項

MySQL クライアント/サーバープロトコルの様々な実装が存在します。 `libmysqlclient` C API クライアントライブラリは実装の一例です。一部の MySQL コネクタ (通常は C で記述されていないコネクタ) では、独自の実装が提供されます。ただし、すべてのプロトコル実装が同じ方法でプラグイン認証を処理するわけではありません。このセクションでは、プロトコル実装者が考慮する必要がある認証の問題について説明します。

クライアント/サーバープロトコルでは、サーバーはデフォルトとみなす認証プラグインをクライアントに接続するように指示します。クライアントが使用するプロトコル実装がデフォルトのプラグインをロードしようとし、そのプラグインがクライアント側に存在しない場合、ロード操作は失敗します。これは、デフォルトのプラグインが、クライアントが接続しようとしているアカウントに実際に必要なプラグインではない場合、不要な障害です。

クライアント/サーバープロトコルの実装にデフォルトの認証プラグインの独自の概念がなく、サーバーによって指定されたデフォルトのプラグインを常にロードしようとすると、そのプラグインが使用できない場合はエラーで失敗します。

この問題を回避するには、クライアントで使用されるプロトコル実装に独自のデフォルトプラグインがあり、最初の選択肢として使用する必要があります (または、サーバーで指定されたデフォルトプラグインのロードに失敗した場合は、このデフォルトにフォールバックします)。例:

- MySQL 5.7 では、 `libmysqlclient` は `mysql_native_password` または `mysql_options()` の `MYSQL_DEFAULT_AUTH` オプションで指定されたプラグインのいずれかをデフォルトの選択として使用します。
- 5.7 クライアントが 8.0 サーバーに接続しようとすると、サーバーは `caching_sha2_password` をデフォルトの認証プラグインとして指定しますが、クライアントは `mysql_native_password` または `MYSQL_DEFAULT_AUTH` を介して指定された資格証明の詳細を送信します。
- クライアントがサーバーによって指定されたプラグインをロードするのは変更プラグインリクエストの場合のみですが、その場合はユーザーアカウントに応じて任意のプラグインを使用できます。この場合、クライアントはプラグインのロードを試みる必要があり、そのプラグインが使用できない場合、エラーはオプションではありません。

プラグブルな認証の制約

このセクションの最初の部分では、 [セクション6.2.17「プラグブル認証」](#) で説明しているプラグブルな認証フレームワークの適用基準に関する一般的な制約について説明します。2 番目の部分では、サードパーティーコネクタ開発者が、コネクタがプラグブルな認証機能を利用できる範囲と、対応性を高めるために行うステップについて判断する方法について説明します。

ここで使用する「ネイティブ認証」という用語は、 `mysql.user` システムテーブルに格納されているパスワードに対する認証を指します。これは、プラグブルな認証が実装される前に古い MySQL Server で提供されていたものと同じ認証方法です。「Windows ネイティブ認証」とは、Windows ネイティブ認証プラグイン (「Windows プラグイン」と略します) で実装された、すでに Windows にログインしているユーザーの資格証明を使用した認証を示します。

- [一般的なプラグブルな認証の制約](#)
- [プラグブルな認証とサードパーティーコネクタ](#)

一般的なプラグブルな認証の制約

- Connector/C++: このコネクタを使用するクライアントは、ネイティブ認証を使用するアカウントを介してのみサーバーに接続できます。

例外: コネクタは、 `libmysqlclient` に (静的ではなく) 動的にリンクするように構築された場合にプラグブルな認証をサポートし、最新バージョンの `libmysqlclient` がインストールされている場合、またはコネクタが最新の

`libmysqlclient` に対してリンクするようにソースから再コンパイルされている場合にそのバージョンをロードします。

サーバーからのデフォルトのサーバー側認証プラグインに関する情報を処理するコネクタの記述については、[認証プラグインコネクタ - 書込みに関する考慮事項](#)を参照してください。

- Connector/NET: Connector/NET を使用するクライアントは、ネイティブ認証または Windows ネイティブ認証を使用するアカウントを介してサーバーに接続できます。
- Connector/PHP: このコネクタを使用するクライアントは、PHP 用の MySQL ネイティブドライバ (`mysqlnd`) を使用してコンパイルされている場合、ネイティブ認証を使用するアカウントを通じてのみサーバーに接続できます。
- Windows ネイティブ認証: Windows プラグインを使用するアカウントを通じた接続は、Windows Domain セットアップを必要とします。これがない場合、NTLM 認証が使用され、ローカル接続だけが可能になります。つまり、クライアントとサーバーを同じコンピュータ上で実行する必要があります。
- プロキシユーザー: プロキシユーザーサポートは、プロキシユーザー機能を実装するプラグイン (つまり、接続しているユーザーの名前と異なるユーザー名を返す場合があるプラグイン) で認証されたアカウントを通じて、クライアントが接続できる範囲まで利用できます。たとえば、PAM および Windows プラグインはプロキシユーザーをサポートしています。 `mysql_native_password` および `sha256_password` 認証プラグインは、デフォルトではプロキシユーザーをサポートしていませんが、これを行うように構成できます。 [プロキシユーザーマッピングのサーバーサポート](#) を参照してください。
- レプリケーション: レプリカは、ネイティブ認証を使用してレプリケーションユーザーアカウントを採用するだけでなく、必要なクライアント側プラグインが使用可能な場合は、非ネイティブ認証を使用するレプリケーションユーザーアカウントを介して接続することもできます。プラグインは、`libmysqlclient` に組み込まれている場合、デフォルトで利用できます。それ以外の場合は、レプリカ `plugin_dir` システム変数で指定されたディレクトリのレプリカ側にプラグインをインストールする必要があります。
- FEDERATED テーブル: FEDERATED テーブルは、ネイティブ認証を使用するリモートサーバー上のアカウントを通じてのみリモートテーブルにアクセスできます。

プラグブルな認証とサードパーティコネクタ

サードパーティコネクタ開発者は、次のガイドラインを使用して、プラグブルな認証機能を利用するためのコネクタの準備と、対応性を高めるために行うステップについて判断できます。

- 変更が行われていない既存のコネクタは、ネイティブ認証を使用し、このコネクタを使用するクライアントは、ネイティブ認証を使用するアカウントを通じてのみサーバーに接続できます。ただし、最新バージョンのサーバーに対してコネクタをテストして、このような接続が引き続き問題なく機能することを検証する必要があります。

例外: コネクタは、(静的ではなく) 動的に `libmysqlclient` にリンクしている場合に、変更せずにプラグブルな認証を処理でき、最新バージョンの `libmysqlclient` がインストールされている場合に、このバージョンをロードします。

- プラグブルな認証機能を利用するには、`libmysqlclient` ベースのコネクタを、最新バージョンの `libmysqlclient` に対して再リンクする必要があります。これにより、コネクタは、現在 `libmysqlclient` に組み込まれているクライアント側のプラグイン (PAM 認証に必要な平文プラグインや Windows ネイティブ認証に必要な Windows プラグインなど) を必要とするアカウントを通じた接続をサポートできるようになります。現在の `libmysqlclient` とのリンクによっても、コネクタは、デフォルトの MySQL プラグインディレクトリ (通常、ローカルサーバーの `plugin_dir` システム変数のデフォルト値で指名されたディレクトリ) にインストールされたクライアント側にアクセスできるようになります。

コネクタが動的に `libmysqlclient` にリンクする場合、より新しいバージョンの `libmysqlclient` がクライアントホストにインストールされていることと、コネクタが実行時にそれをロードすることを確認する必要があります。

- コネクタが特定の認証方式をサポートする別の方法は、クライアント/サーバープロトコルに直接実装することです。Connector/NET は、このアプローチを使用して Windows ネイティブ認証をサポートします。
- コネクタが、デフォルトのプラグインディレクトリとは異なるディレクトリから、クライアント側のプラグインをロードできる必要がある場合、クライアントユーザーがそのディレクトリを指定するための手段を実装する必要があります。この候補としては、コネクタがディレクトリ名を取得できるコマンド行オプションまたは環境変数などがあります。 `mysql` や `mysqldadmin` などの標準 MySQL クライアントプログラムは、`--plugin-dir` オプションを実装します。 [C API Client Plugin Interface](#) も参照してください。

- コネクタでのプロキシユーザーのサポートは、このセクションで前述したように、コネクタがサポートする認証方式がプロキシユーザーを許可するかどうかによって異なります。

6.2.18 プロキシユーザー

MySQL サーバーは、認証プラグインを使用してクライアント接続を認証します。特定の接続を認証するプラグインは、特権チェックのために接続 (外部) ユーザーを別のユーザーとして扱うようにリクエストする場合があります。これにより、外部ユーザーを 2 人目のユーザーのプロキシにできます。つまり、2 人目のユーザーの権限を引き受けることができます:

- 外部ユーザーは「「プロキシユーザー」」(別のユーザーとして偽装または認識できるユーザー) です。
- 2 番目のユーザーは「「プロキシユーザー」」(プロキシユーザーがアイデンティティと権限を引き受けることができるユーザー) です。

このセクションでは、プロキシユーザー機能の動作について説明します。認証プラグインに関する一般的な情報については、[セクション6.2.17「プラグイン認証」](#)を参照してください。特定のプラグインの詳細は、[セクション6.4.1「認証プラグイン」](#)を参照してください。プロキシユーザーをサポートする認証プラグインの記述の詳細は、[Implementing Proxy User Support in Authentication Plugins](#)を参照してください。

- [プロキシユーザーサポートの要件](#)
- [単純なプロキシユーザーの例](#)
- [プロキシアカウントへの直接ログインの防止](#)
- [PROXY 権限の付与および取消し](#)
- [デフォルトのプロキシユーザー](#)
- [デフォルトのプロキシユーザーと匿名ユーザーの競合](#)
- [プロキシユーザーマッピングのサーバーサポート](#)
- [プロキシユーザーのシステム変数](#)

注記

プロキシによって得られる管理上の利点の 1 つは、DBA が一連の権限を持つ単一のアカウントを設定し、それらの各ユーザーに権限を個別に割り当てることなく、複数のプロキシユーザーがそれらの権限を持つことができることです。プロキシユーザーのかわりに、DBA は、ユーザーを特定の名前付き権限セットにマップするための適切な方法をロールが提供する場合があります。各ユーザーには、特定の単一のロールを付与して、実際には適切な権限セットを付与できます。[セクション6.2.10「ロールの使用」](#)を参照してください。

プロキシユーザーサポートの要件

特定の認証プラグインに対してプロキシを実行するには、次の条件を満たす必要があります:

- プロキシは、プラグイン自体またはプラグインのかわりに MySQL サーバーによってサポートされている必要があります。後者の場合、サーバーサポートを明示的に有効にする必要がある場合があります。[プロキシユーザーマッピングのサーバーサポート](#)を参照してください。
- プラグインによって認証されるように外部プロキシユーザーのアカウントを設定する必要があります。`CREATE USER` ステートメントを使用してアカウントを認証プラグインに関連付けるか、`ALTER USER` を使用してそのプラグインを変更します。
- プロキシユーザーのアカウントが存在し、プロキシユーザーが想定する権限が付与されている必要があります。これには、`CREATE USER` および `GRANT` ステートメントを使用します。
- 通常、プロキシユーザーは、プロキシシナリオでのみ使用でき、直接ログインには使用できないように構成されません。

- プロキシユーザーアカウントには、プロキシ設定されたアカウントに対する **PROXY** 権限が必要です。これを行うには、**GRANT** ステートメントを使用します。
- プロキシアカウントに接続しているクライアントをプロキシユーザーとして処理するには、認証プラグインがクライアントユーザー名とは異なるユーザー名を返して、プロキシユーザーが想定する権限を定義するプロキシアカウントのユーザー名を示す必要があります。

または、サーバーによってプロキシマッピングが提供されるプラグインの場合、プロキシユーザーはプロキシユーザーが保持する **PROXY** 権限から決定されます。

プロキシメカニズムでは、外部クライアントユーザー名のみをプロキシユーザー名にマップできます。ホスト名をマッピングするためのプロビジョニングはありません:

- クライアントがサーバーに接続すると、サーバーはクライアントプログラムによって渡されたユーザー名とクライアントの接続元のホストに基づいて適切なアカウントを決定します。
- そのアカウントがプロキシアカウントである場合、サーバーは、認証プラグインによって返されたユーザー名とプロキシアカウントのホスト名を使用してプロキシアカウントの一致を検索することで、適切なプロキシアカウントを決定しようとします。プロキシ設定されたアカウントのホスト名は無視されます。

単純なプロキシユーザーの例

次のアカウント定義について考えてみます:

```
-- create proxy account
CREATE USER 'employee_ext'@'localhost'
  IDENTIFIED WITH my_auth_plugin
  AS 'my_auth_string';

-- create proxied account and grant its privileges;
-- use mysql_no_login plugin to prevent direct login
CREATE USER 'employee'@'localhost'
  IDENTIFIED WITH mysql_no_login;
GRANT ALL
  ON employees.*
  TO 'employee'@'localhost';

-- grant to proxy account the
-- PROXY privilege for proxied account
GRANT PROXY
  ON 'employee'@'localhost'
  TO 'employee_ext'@'localhost';
```

クライアントがローカルホストから **employee_ext** として接続すると、MySQL は **my_auth_plugin** という名前のプラグインを使用して認証を実行します。 **my_auth_plugin** が、'my_auth_string'の内容に基づいて、外部認証システムを参照することで、**employee** のユーザー名をサーバーに返すとしてします。 **employee** という名前は **employee_ext** とは異なるため、**employee** を返すことは、権限チェックの目的で **employee_ext** 外部ユーザーを **employee** ローカルユーザーとして処理するためのリクエストとして機能します。

この場合、**employee_ext** はプロキシユーザーで、**employee** はプロキシユーザーです。

サーバーは、**employee_ext** (プロキシユーザー) が **employee** (プロキシユーザー) に対する **PROXY** 権限を持っているかどうかを確認することで、**employee_ext** ユーザーに対して **employee** のプロキシ認証が可能であることを検証します。この権限が付与されていない場合は、エラーが発生します。それ以外の場合、**employee_ext** は **employee** の権限を引き継ぎます。サーバーは、**employee_ext** によってクライアントセッション中に実行されたステートメントを、**employee** に付与された権限に対してチェックします。この場合、**employee_ext** は **employees** データベースのテーブルにアクセスできます。

プロキシ設定されたアカウント **employee** は、**mysql_no_login** 認証プラグインを使用して、クライアントがアカウントを使用して直接ログインできないようにします。(これは、プラグインがインストールされていることを前提としています。手順については、[セクション6.4.1.8「ログインなしのプラグイン認証」](#)を参照してください。)プロキシ設定されたアカウントを直接使用しないように保護する別の方法については、[プロキシアカウントへの直接ログインの防止](#)を参照してください。

プロキシが発生すると、**USER()** および **CURRENT_USER()** の機能を使用して、接続ユーザー (プロキシユーザー) と、現在のセッション中に権限が適用されるアカウント (プロキシユーザー) の違いを確認できます。先ほど説明した例では、これらの関数は次の値を返します。

```
mysql> SELECT USER(), CURRENT_USER();
+-----+-----+
| USER()          | CURRENT_USER() |
+-----+-----+
| employee_ext@localhost | employee@localhost |
+-----+-----+
```

プロキシユーザーアカウントを作成する `CREATE USER` ステートメントでは、プロキシサポート認証プラグインを指定する `IDENTIFIED WITH` 句のあとに、オプションで、ユーザーの接続時にサーバーがプラグインに渡す文字列を指定する `AS 'auth_string'` 句が続きます。存在する場合、この文字列は、プロキシ (外部) クライアントユーザー名をプロキシユーザー名にマップする方法をプラグインが決定するのに役立つ情報を提供します。`AS` 句が必要かどうかは、プラグインごとに異なります。その場合、認証文字列の形式は、プラグインがそれをどのように使用するかによって異なります。許可される認証文字列の値については、特定のプラグインに関するドキュメントを参照してください。

プロキシアカウントへの直接ログインの防止

プロキシアカウントは通常、プロキシアカウントによってのみ使用されます。つまり、クライアントはプロキシアカウントを使用して接続し、適切なプロキシユーザーの権限にマップされて引き受けます。

プロキシ設定されたアカウントを直接使用できないようにするには、複数の方法があります：

- アカウントを `mysql_no_login` 認証プラグインに関連付けます。この場合、アカウントはどのような状況でも直接ログインには使用できません。これは、プラグインがインストールされていることを前提としています。その手順は、[セクション6.4.1.8「ログインなしのプラグイン認証」](#)を参照してください。
- アカウントの作成時に `ACCOUNT LOCK` オプションを含めます。[セクション13.7.1.3「CREATE USER ステートメント」](#)を参照してください。この方法では、アカウントが後でロック解除された場合にパスワードなしでアクセスできないように、パスワードも含めます。(`validate_password` コンポーネントが有効な場合、アカウントがロックされていても、パスワードなしのアカウントの作成は許可されません。[セクション6.4.3「パスワード検証コンポーネント」](#)を参照してください。)
- パスワードを使用してアカウントを作成しますが、他のユーザーにはパスワードを通知しません。アカウントのパスワードを誰にも知らない場合、クライアントはそれを使用して MySQL サーバーに直接接続できません。

PROXY 権限の付与および取消し

外部ユーザーが別のユーザーとして接続し、別のユーザーの権限を持つことができるようにするには、`PROXY` 権限が必要です。この権限を付与するには、`GRANT` ステートメントを使用します。例：

```
GRANT PROXY ON 'proxied_user' TO 'proxy_user';
```

このステートメントは、`mysql.proxies_priv` 付与テーブルに行を作成します。

接続時に、`proxy_user` は有効な外部認証 MySQL ユーザーを表し、`proxied_user` は有効なローカル認証ユーザーを表す必要があります。そうでない場合、接続は失敗します。

対応する `REVOKE` 構文は次のとおりです。

```
REVOKE PROXY ON 'proxied_user' FROM 'proxy_user';
```

MySQL `GRANT` および `REVOKE` 構文の拡張機能は、通常どおりに動作します。例：

```
-- grant PROXY to multiple accounts
GRANT PROXY ON 'a' TO 'b', 'c', 'd';

-- revoke PROXY from multiple accounts
REVOKE PROXY ON 'a' FROM 'b', 'c', 'd';

-- grant PROXY to an account and enable the account to grant
-- PROXY to the proxied account
GRANT PROXY ON 'a' TO 'd' WITH GRANT OPTION;

-- grant PROXY to default proxy account
```

```
GRANT PROXY ON 'a' TO '@';
```

次のような場合に、**PROXY** 権限を付与できます。

- proxied_user に対する **GRANT PROXY ... WITH GRANT OPTION** を持つユーザーによる。
- 自分で proxied_user による: アカウント名のユーザー名とホスト名の両方の部分で、**USER()** の値が **CURRENT_USER()** および proxied_user と完全に一致する必要があります。

MySQL のインストール中に作成された最初の root アカウントには、"**@**"(すべてのユーザーおよびすべてのホスト) に対する **PROXY ... WITH GRANT OPTION** 権限があります。これにより、root はプロキシユーザーを設定したり、プロキシユーザーを設定するための権限をほかのアカウントに委任したりできます。たとえば、root は次の操作を実行できます。

```
CREATE USER 'admin'@'localhost'  
  IDENTIFIED BY 'admin_password';  
GRANT PROXY  
  ON '@'  
  TO 'admin'@'localhost'  
  WITH GRANT OPTION;
```

これらのステートメントによって、すべての **GRANT PROXY** マッピングを管理できる admin ユーザーが作成されます。たとえば、admin は次の操作を実行できます。

```
GRANT PROXY ON sally TO joe;
```

デフォルトのプロキシユーザー

一部またはすべてのユーザーが特定の認証プラグインを使用して接続する必要があることを指定するには、空のユーザー名とホスト名 ("**@**") で「空白」MySQL アカウントを作成し、それをそのプラグインに関連付けて、プラグインが実際の認証済ユーザー名を返すようにします(空白のユーザーと異なる場合)。LDAP 認証を実装し、接続ユーザーを開発者またはマネージャアカウントにマップする **ldap_auth** という名前のプラグインが存在するとします。これらのアカウントに対するユーザーのプロキシを設定するには、次のステートメントを使用します:

```
-- create default proxy account  
CREATE USER '@'  
  IDENTIFIED WITH ldap_auth  
  AS 'O=Oracle, OU=MySQL';  
  
-- create proxied accounts; use  
-- mysql_no_login plugin to prevent direct login  
CREATE USER 'developer'@'localhost'  
  IDENTIFIED WITH mysql_no_login;  
CREATE USER 'manager'@'localhost'  
  IDENTIFIED WITH mysql_no_login;  
  
-- grant to default proxy account the  
-- PROXY privilege for proxied accounts  
GRANT PROXY  
  ON 'manager'@'localhost'  
  TO '@';  
GRANT PROXY  
  ON 'developer'@'localhost'  
  TO '@';
```

ここで、クライアントが次のように接続するとします:

```
shell> mysql --user=myuser --password ...  
Enter password: myuser_password
```

サーバーは MySQL ユーザーとして定義された myuser を検出しませんが、クライアントユーザー名およびホスト名と一致する空白のユーザーアカウント ("**@**") があるため、サーバーはそのアカウントに対してクライアントを認証します。サーバーは **ldap_auth** 認証プラグインを呼び出し、myuser および myuser_password をユーザー名とパスワードとして渡します。

myuser_password が myuser の正しいパスワードではないことが LDAP ディレクトリで ldap_auth プラグインによって検出された場合、認証は失敗し、サーバーは接続を拒否します。

パスワードが正しく、`ldap_auth` によって `myuser` が開発者であることが検出された場合は、MySQL サーバーに `myuser` ではなく、`developer` というユーザー名が返されます。 `myuser` のクライアントユーザー名とは異なるユーザー名を返すと、`myuser` をプロキシとして扱う必要があることがサーバーに通知されます。サーバーは、"`@`"が `developer` として認証できることを検証し ("`@`"にはそれを行う `PROXY` 権限があるため)、接続を受け入れます。セッションは、`developer` プロキシユーザーの権限を持つ `myuser` に進みます。(これらの権限は、DBA が `GRANT` ステートメントを使用して設定するべきですが、表示されません。) `USER()` および `CURRENT_USER()` 関数は、次の値を返します。

```
mysql> SELECT USER(), CURRENT_USER();
+-----+-----+
| USER() | CURRENT_USER() |
+-----+-----+
| myuser@localhost | developer@localhost |
+-----+-----+
```

かわりに、LDAP ディレクトリで `myuser` がマネージャであることが検出されると、ユーザー名として `manager` が返され、セッションは `manager` プロキシユーザーの権限を持つ `myuser` で続行されます。

```
mysql> SELECT USER(), CURRENT_USER();
+-----+-----+
| USER() | CURRENT_USER() |
+-----+-----+
| myuser@localhost | manager@localhost |
+-----+-----+
```

単純にするために、外部認証はマルチレベルで実行できません。前述の例では、`developer` の証明書も、`manager` の証明書も考慮されません。ただし、クライアントが `developer` または `manager` アカウントとして直接接続および認証しようとする場合は、これらのプロキシアカウントを直接ログインから保護する必要があります ([プロキシアカウントへの直接ログインの防止](#) を参照)。

デフォルトのプロキシユーザーと匿名ユーザーの競合

デフォルトのプロキシユーザーを作成する場合は、デフォルトのプロキシユーザーよりも優先される他の既存の「「いずれかのユーザーに一致」」アカウントを確認します。これは、そのユーザーが意図したとおりに作業できない可能性があるためです。

前述の説明で、デフォルトのプロキシユーザーアカウントのホスト部分には、任意のホストと一致する"`@`"があります。デフォルトのプロキシユーザーを設定する場合は、ホスト部分に同じユーザー部分と"`%`"を持つ非プロキシアカウントが存在するかどうかにも注意して確認してください。これは、"`%`"も任意のホストに一致しますが、サーバーがアカウント行を内部的にソートするために使用するルールによって"`@`"よりも優先されるためです ([セクション6.2.6「アクセス制御、ステージ 1: 接続の検証」](#) を参照)。

MySQL のインストールに、次の 2 つのアカウントが含まれていると仮定します。

```
-- create default proxy account
CREATE USER "@"
  IDENTIFIED WITH some_plugin
  AS 'some_auth_string';
-- create anonymous account
CREATE USER "@%"
  IDENTIFIED BY 'anon_user_password';
```

最初のアカウント ("`@`") はデフォルトのプロキシユーザーとして使用され、それ以外の場合はより特定のアカウントと一致しないユーザーの接続を認証するために使用されます。2 つ目のアカウント ("`@%`") は匿名ユーザーアカウントで、たとえば、独自のアカウントを持たないユーザーが匿名で接続できるようにするために作成された可能性があります。

両方のアカウントに同じユーザー部分 ("`@`") があり、これは任意のユーザーに一致します。各アカウントには、任意のホストと一致するホスト部分があります。ただし、一致ルールは"`@`"の前に"`%`"のホストをソートするため、接続試行のアカウント照合には優先度があります。他の特定のアカウントと一致しないアカウントの場合、サーバーは"`@`" (デフォルトのプロキシユーザー) ではなく、"`@%`" (匿名ユーザー) に対して認証を試みます。その結果、デフォルトのプロキシアカウントは使用されません。

この問題を回避するには、次のいずれかの方法を使用します:

- 匿名アカウントを削除して、デフォルトのプロキシユーザーと競合しないようにします。
- 匿名ユーザーの前に一致する特定のデフォルトプロキシユーザーを使用します。たとえば、`localhost` プロキシ接続のみを許可するには、`"@'localhost'`を使用します:

```
CREATE USER "@'localhost'  
  IDENTIFIED WITH some_plugin  
  AS 'some_auth_string';
```

また、`GRANT PROXY` ステートメントを変更して、`"@"`ではなく`"@'localhost'`をプロキシユーザーとして指定します。

この戦略により、`localhost` からの匿名ユーザー接続が防止されることに注意してください。

- 匿名のデフォルトアカウントではなく、名前付きのデフォルトアカウントを使用します。この方法の例については、`authentication_windows` プラグインを使用する手順を参照してください。セクション6.4.1.6「Windows プラグイン認証」を参照してください。
- 複数のプロキシユーザーを作成します。1つはローカル接続用、もう1つは「その他すべて」用です(リモート接続)。これは、特にローカルユーザーがリモートユーザーとは異なる権限を持つ必要がある場合に役立ちます。

プロキシユーザーを作成します:

```
-- create proxy user for local connections  
CREATE USER "@'localhost'  
  IDENTIFIED WITH some_plugin  
  AS 'some_auth_string';  
-- create proxy user for remote connections  
CREATE USER "@'%'  
  IDENTIFIED WITH some_plugin  
  AS 'some_auth_string';
```

プロキシユーザーを作成します:

```
-- create proxied user for local connections  
CREATE USER 'developer'@'localhost'  
  IDENTIFIED WITH mysql_no_login;  
-- create proxied user for remote connections  
CREATE USER 'developer'@'%'  
  IDENTIFIED WITH mysql_no_login;
```

各プロキシアカウントに、対応するプロキシアカウントの `PROXY` 権限を付与します:

```
GRANT PROXY  
  ON 'developer'@'localhost'  
  TO "@'localhost';  
GRANT PROXY  
  ON 'developer'@'%'  
  TO "@'%';
```

最後に、ローカルおよびリモートのプロキシユーザーに適切な権限を付与します(表示されていません)。

`some_plugin/some_auth_string`の組合せによって、`some_plugin` がクライアントユーザー名を `developer` にマップするとします。ローカル接続は、`'developer'@'localhost'`プロキシユーザーにマップされる`"@'localhost'`プロキシユーザーと一致します。リモート接続は、`'developer'@'%'`プロキシユーザーにマップされる`"@'%'`プロキシユーザーと一致します。

プロキシユーザーマッピングのサーバーサポート

一部の認証プラグインは、プロキシユーザーマッピングを実装しています(PAM や Windows 認証プラグインなど)。その他の認証プラグインは、デフォルトではプロキシユーザーをサポートしていません。その中には、付与されたプロキシ権限に従って、MySQL サーバー自体がプロキシユーザーをマップするようにリクエストできるものがあります:`mysql_native_password`、`sha256_password`。`check_proxy_users` システム変数が有効になっている場合、サーバーは、このようなリクエストを行う認証プラグインに対してプロキシユーザーマッピングを実行します:

- デフォルトでは、`check_proxy_users` は無効になっているため、サーバーはプロキシユーザーのサーバーサポートをリクエストする認証プラグインに対してもプロキシユーザーマッピングを実行しません。

- `check_proxy_users` が有効になっている場合は、サーバープロキシユーザーマッピングのサポートを利用するために、プラグイン固有のシステム変数を有効にする必要がある場合もあります:
 - `mysql_native_password` プラグインの場合は、`mysql_native_password_proxy_users` を有効にします。
 - `sha256_password` プラグインの場合は、`sha256_password_proxy_users` を有効にします。

たとえば、前述のすべての機能を有効にするには、`my.cnf` ファイルで次の行を使用してサーバーを起動します:

```
[mysqld]
check_proxy_users=ON
mysql_native_password_proxy_users=ON
sha256_password_proxy_users=ON
```

関連するシステム変数が有効になっている場合は、`CREATE USER` を使用して通常どおりにプロキシユーザーを作成し、プロキシユーザーとして扱われる単一の他のアカウントに `PROXY` 権限を付与します。サーバーは、プロキシユーザーに対する正常な接続リクエストを受信すると、そのユーザーに `PROXY` 権限があることを確認し、それを使用して適切なプロキシユーザーを決定します。

```
-- create proxy account
CREATE USER 'proxy_user'@'localhost'
  IDENTIFIED WITH mysql_native_password
  BY 'password';

-- create proxied account and grant its privileges;
-- use mysql_no_login plugin to prevent direct login
CREATE USER 'proxied_user'@'localhost'
  IDENTIFIED WITH mysql_no_login;
-- grant privileges to proxied account
GRANT ...
  ON ...
  TO 'proxied_user'@'localhost';

-- grant to proxy account the
-- PROXY privilege for proxied account
GRANT PROXY
  ON 'proxied_user'@'localhost'
  TO 'proxy_user'@'localhost';
```

プロキシアカウントを使用するには、名前とパスワードを使用してサーバーに接続します:

```
shell> mysql -u proxy_user -p
Enter password: (enter proxy_user password here)
```

認証に成功すると、サーバーは `proxy_user` に `proxied_user` に対する `PROXY` 権限があることを検出し、セッションは `proxied_user` の権限を持つ `proxy_user` に進みます。

サーバーによって実行されるプロキシユーザーマッピングには、次の制限事項があります:

- 関連付けられた `PROXY` 権限が付与されている場合でも、サーバーは匿名ユーザーとの間でプロキシを行いません。
- 単一のアカウントに複数のプロキシアカウントのプロキシ権限が付与されている場合、サーバープロキシユーザーマッピングは非決定的です。したがって、複数のプロキシアカウントに対する単一アカウントのプロキシ権限への付与はお勧めしません。

プロキシユーザーのシステム変数

次の2つのシステム変数は、プロキシのログインプロセスをトレースする際に役立ちます。

- `proxy_user`: プロキシ処理が使用されていない場合、この値は `NULL` です。それ以外の場合は、プロキシユーザーのアカウントを示します。たとえば、クライアントが"@"プロキシアカウントを介して認証する場合、この変数は次のように設定されます:

```
mysql> SELECT @@proxy_user;
+-----+
| @@proxy_user |
+-----+
```

```
|"@" |  
+-----+
```

- `external_user`: 認証プラグインは外部ユーザーを使用して、MySQL サーバーへの認証を行うことがあります。たとえば、Windows のネイティブ認証を使用するときは、Windows の API を使用して認証するプラグインに、ログイン ID を渡す必要がありません。ただし、認証には引き続き Windows ユーザー ID が使用されます。このプラグインは、読み取り専用のセッション変数 `external_user` を使用して、この外部ユーザー ID (または最初の 512 UTF-8 バイト) をサーバーに返すことがあります。プラグインがこの変数を設定しない場合、その値は `NULL` です。

6.2.19 アカウントロック

MySQL では、`CREATE USER` および `ALTER USER` ステートメントの `ACCOUNT LOCK` 句および `ACCOUNT UNLOCK` 句を使用したユーザーアカウントのロックおよびロック解除がサポートされています:

- これらの句を `CREATE USER` とともに使用すると、新しいアカウントの初期ロック状態を指定できます。どちらの句もない場合、アカウントはロック解除された状態で作成されます。

`validate_password` コンポーネントが有効な場合、アカウントがロックされていても、パスワードなしでのアカウントの作成は許可されません。 [セクション6.4.3「パスワード検証コンポーネント」](#) を参照してください。

- これらの句を `ALTER USER` とともに使用すると、既存のアカウントの新しいロック状態を指定できます。どちらの句もない場合、アカウントのロック状態は変更されません。

MySQL 8.0.19 の時点では、ログインの失敗回数が多すぎるために一時的にロックされているステートメントで指定されたアカウントは、`ALTER USER ... UNLOCK` によってロック解除されます。 [セクション6.2.15「パスワード管理」](#) を参照してください。

アカウントのロック状態は、`mysql.user` システムテーブルの `account_locked` カラムに記録されます。 `SHOW CREATE USER` からの出力には、アカウントがロックされているかロック解除されているかが示されます。

クライアントがロックされたアカウントに接続しようとすると、試行は失敗します。サーバーは、ロックされたアカウントへの接続試行回数を示す `Locked_connects` ステータス変数を増分し、`ER_ACCOUNT_HAS_BEEN_LOCKED` エラーを返して、エラーログにメッセージを書き込みます:

```
Access denied for user 'user_name'@'host_name'.  
Account is locked.
```

アカウントをロックしても、ロックされたアカウントのアイデンティティを想定するプロキシユーザーを使用して接続できることには影響しません。また、ロックされたアカウントを指定する `DEFINER` 属性を持つストアードプログラムまたはビューを実行する機能にも影響しません。つまり、プロキシ設定されたアカウント、ストアードプログラムまたはビューを使用する機能は、アカウントをロックしても影響を受けません。

アカウントロック機能は、`mysql.user` システムテーブルに `account_locked` カラムが存在するかどうかによって異なります。5.7.6 より古い MySQL バージョンからのアップグレードの場合は、MySQL のアップグレード手順を実行して、このカラムが存在することを確認します。 [セクション2.11「MySQL のアップグレード」](#) を参照してください。 `account_locked` カラムがないアップグレードされていないインストールの場合、サーバーはすべてのアカウントをロック解除済として扱い、`ACCOUNT LOCK` 句または `ACCOUNT UNLOCK` 句を使用するとエラーが発生します。

6.2.20 アカウントリソース制限の設定

MySQL サーバーリソースのクライアント使用を制限する方法の 1 つは、グローバル `max_user_connections` システム変数をゼロ以外の値に設定することです。これにより、任意の特定のアカウントで実行できる同時接続の数が制限されますが、一度クライアントが接続したら、実行内容には制限が課されません。さらに、`max_user_connections` を設定しても、各アカウントの管理は有効になりません。どちらのタイプの制御も、MySQL 管理者にとって重要です。

このような問題に対処するために、MySQL では、次のサーバーリソースの使用に関する個々のアカウントの制限を許可しています:

- アカウントが発行できるクエリーの数/時間
- アカウントが発行できる更新の数/時間

- アカウントが 1 時間ごとにサーバーに接続できる回数
- アカウントによるサーバーへの同時接続の数

クライアントがクエリー制限に対してカウントを発行できるステートメント。更新制限に対して、データベースまたはテーブルを変更するステートメントのみがカウントされます。

このコンテキストの「account」は、`mysql.user` システムテーブルの行に対応します。つまり、接続に適用される `user` テーブル行内の `User` および `Host` 値に対して、接続が評価されます。たとえば、アカウント `'usera'@'%example.com'` は、`example.com` ドメイン内の任意のホストから接続することを `usera` に許可するために、`usera` および `%example.com` の `User` および `Host` 値を持つ `user` テーブル内の行に対応しています。この場合、このような接続ではすべて同じアカウントが使用されるため、サーバーは、`usera` による `example.com` ドメイン内の任意のホストからのすべての接続に、この行のリソース制限をまとめて適用します。

MySQL 5.0 よりも前では、ユーザーの接続元である実際のホストに対して、「アカウント」が評価されていました。この古いアカウントイング方法は、`--old-style-user-limits` オプションを使用してサーバーを起動することで選択できます。この場合、`usera` が `host1.example.com` と `host2.example.com` から同時に接続すると、サーバーは各接続に個別にアカウントリソースの制限を適用します。`usera` が `host1.example.com` から再度接続すると、サーバーはそのホストからの既存の接続とともに、その接続に対する制限を適用します。

アカウントの作成時にアカウントのリソース制限を確立するには、`CREATE USER` ステートメントを使用します。既存のアカウントの制限を変更するには、`ALTER USER` を使用します。制限される各リソースの名前を指定する `WITH` 句を指定します。各制限のデフォルト値は、ゼロ (制限なし) です。たとえば、制限された方法でのみ `customer` データベースにアクセスできる新しいアカウントを作成するには、次のようなステートメントを発行します。

```
mysql> CREATE USER 'francis'@'localhost' IDENTIFIED BY 'frank'  
-> WITH MAX_QUERIES_PER_HOUR 20  
-> MAX_UPDATES_PER_HOUR 10  
-> MAX_CONNECTIONS_PER_HOUR 5  
-> MAX_USER_CONNECTIONS 2;
```

制限タイプの名前をすべて `WITH` 句に指定する必要はありませんが、名前を指定したものは任意の順序で表示できます。1 時間ごとの各制限の値は、1 時間当たりの回数を表す整数にするようにしてください。`MAX_USER_CONNECTIONS` では、制限はアカウントによる同時接続の最大数を表す整数です。この制限がゼロに設定されている場合は、グローバルな `max_user_connections` システム変数の値によって同時接続の数が決定されます。`max_user_connections` もゼロである場合は、アカウントに制限がありません。

既存のアカウントの制限を変更するには、`ALTER USER` ステートメントを使用します。次のステートメントは、`francis` に対するクエリー制限を 100 に変更します。

```
mysql> ALTER USER 'francis'@'localhost' WITH MAX_QUERIES_PER_HOUR 100;
```

このステートメントは、指定された制限値のみを変更し、それ以外のアカウントは未変更のままにします。

制限を削除するには、その値をゼロに設定します。たとえば、`francis` が接続できる 1 時間当たりの回数に対する制限を削除するには、次のステートメントを使用します。

```
mysql> ALTER USER 'francis'@'localhost' WITH MAX_CONNECTIONS_PER_HOUR 0;
```

すでに説明したように、アカウントに対する同時接続の制限は、`MAX_USER_CONNECTIONS` 制限および `max_user_connections` システム変数によって決定されます。グローバル `max_user_connections` 値が 10 で、3 つのアカウントに次のように個別のリソース制限が指定されているとします:

```
ALTER USER 'user1'@'localhost' WITH MAX_USER_CONNECTIONS 0;  
ALTER USER 'user2'@'localhost' WITH MAX_USER_CONNECTIONS 5;  
ALTER USER 'user3'@'localhost' WITH MAX_USER_CONNECTIONS 20;
```

`user1` の接続制限は、`MAX_USER_CONNECTIONS` 制限がゼロであるため、10 (グローバル `max_user_connections` 値) です。`user2` および `user3` には、ゼロ以外の `MAX_USER_CONNECTIONS` 制限があるため、それぞれ接続制限は 5 および 20 です。

サーバーはアカウントに対応する `user` テーブル行に、アカウントに対するリソース制限を格納します。`max_questions`、`max_updates`、または `max_connections` カラムには、1 時間当たりの制限が格納され、`max_user_connections` カラムには、`MAX_USER_CONNECTIONS` の制限が格納されます。(セクション 6.2.3 「付与テーブル」を参照してください。)

任意のアカウントによるリソースのいずれかの使用に対してゼロ以外の制限が設定されている場合は、リソース使用のカウンタが発生します。

サーバーが実行されると、各アカウントがリソースを使用する回数がカウンタされます。アカウントが過去 1 時間以内の接続数の制限に達した場合、サーバーはその時間が稼働するまでアカウントのそれ以上の接続を拒否します。同様に、アカウントがクエリーまたは更新の数の制限に達した場合、サーバーは時間が稼働するまでそれ以上のクエリーまたは更新を拒否します。このような場合、サーバーは適切なエラーメッセージを発行します。

リソースカウンタは、クライアントごとではなく、アカウントごとに行われます。たとえば、アカウントのクエリー制限が 50 である場合は、サーバーへの 2 つの同時クライアント接続を作成しても、制限を 100 に増加できません。両方の接続で発行されたクエリーは、まとめてカウンタされます。

現在の 1 時間ごとのリソース使用のカウンタは、すべてのアカウントに対してグローバルにリセットすることも、特定のアカウントごとに個別にリセットすることもできます。

- すべてのアカウントに対して現在のカウンタをゼロにリセットするには、`FLUSH USER_RESOURCES` ステートメント発行します。また、(たとえば、`FLUSH PRIVILEGES` ステートメントまたは `mysqladmin reload` コマンドを使用して) 付与テーブルを再ロードして、カウンタをリセットすることもできます。
- 個々のアカウントのカウンタは、制限のいずれかを再度設定することでゼロにリセットできます。アカウントに現在割り当てられている値と等しい制限値を指定します。

時間ごとのカウンタのリセットは、`MAX_USER_CONNECTIONS` の制限には影響しません。

すべてのカウンタは、サーバーの起動時にゼロから始まります。カウンタはサーバーの再起動を引き継ぎません。

`MAX_USER_CONNECTIONS` の制限では、アカウントが許可されている接続の最大数を現在開いている場合に、エッジケースが発生する可能性があります。接続が発生する時点までにサーバーで切断が完全に処理されていない場合に、切断後にすぐに接続すると、エラー (`ER_TOO_MANY_USER_CONNECTIONS` または `ER_USER_LIMIT_REACHED`) が発生する可能性があります。サーバーが切断処理を終了すると、別の接続が許可されます。

6.2.21 MySQL への接続の問題のトラブルシューティング

MySQL サーバーへの接続を試行したときに問題が発生した場合に問題を修正するために実行できる一連のアクションについて、次の項目で説明します。

- サーバーが実行中であることを確認します。そうでない場合、クライアントは接続できません。たとえば、サーバーに接続しようとして次のいずれかのようなメッセージで失敗した場合、サーバーが実行中でないことが 1 つの原因であることがあります。

```
shell> mysql
ERROR 2003: Can't connect to MySQL server on 'host_name' (111)
shell> mysql
ERROR 2002: Can't connect to local MySQL server through socket
'/tmp/mysql.sock' (111)
```

- サーバーは実行しているが、サーバーが待機しているのと異なる TCP/IP ポート、名前付きパイプ、または Unix ソケットファイルを使用して接続しようとしている場合もあります。これを修正するには、クライアントプログラムを呼び出すときに、適切なポート番号を指すように `--port` オプションを指定するか、`--socket` で適切な名前付きパイプまたは Unix ソケットファイルを指定します。ソケットファイルがある場所を見つけるには、次のコマンドを使用できます。

```
shell> netstat -ln | grep mysql
```

- サーバーがネットワーク接続を無視するように構成されていないこと、または (リモート側から接続しようとする場合に) サーバーのネットワークインタフェース上でローカル側でのみ待機するように構成されていないことを確認します。 `skip_networking` システム変数を有効にしてサーバーを起動した場合、TCP/IP 接続は受け入れられません。 `bind_address` システム変数を `127.0.0.1` に設定してサーバーを起動した場合、サーバーはループバックインタフェースでローカルでのみ TCP/IP 接続をリスニングし、リモート接続を受け入れません。
- ファイアウォールが MySQL へのアクセスをブロックしていないか確認します。ファイアウォールは、実行中のアプリケーションまたは MySQL によって通信用に使用されるポート番号 (デフォルトは 3306) を基準として構成され

ることがあります。Linux または Unix の場合、IP テーブル (または同様の機能の) 構成を調べてポートがブロックされていないことを確認します。Windows では、ZoneAlarm や Windows ファイアウォールなどのアプリケーションを、MySQL ポートをブロックしないように構成する必要がある場合があります。

- 付与テーブルが適切にセットアップされており、サーバーがこれをアクセス制御に使用できるようになっていることが必要です。一部の配布タイプ (Windows でのバイナリ配布、Linux での RPM および DEB 配布など) では、インストールプロセスによって、付与テーブルを含む `mysql` システムデータベースを含む MySQL データディレクトリが初期化されます。これを行わない配布の場合は、データディレクトリを手動で初期化する必要があります。詳細は、[セクション2.10「インストール後のセットアップとテスト」](#)を参照してください。

付与テーブルの初期化が必要かどうかを判別するには、データディレクトリの下にある `mysql` ディレクトリを参照します。(通常、データディレクトリは `data` または `var` という名前で、MySQL のインストールディレクトリの下にあります。) `mysql` データベースディレクトリに `user.MYD` という名前のファイルがあることを確認してください。そうでない場合は、データディレクトリを初期化します。これを行ってサーバーを起動すると、サーバーに接続できるようになります。

- 新規インストールの後、パスワードを使用せずに `root` としてサーバーにログオンしようとする、次のエラーメッセージが表示されることがあります。

```
shell> mysql -u root
ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: NO)
```

これは、インストール時に `root` パスワードがすでに割り当てられており、指定する必要があることを意味します。パスワードが割り当てられている様々な方法、およびパスワードの検索方法については、[セクション2.10.4「初期 MySQL アカウントの保護」](#)を参照してください。`root` パスワードをリセットする必要がある場合は、[セクションB.3.3.2「root のパスワードをリセットする方法」](#)の手順を参照してください。パスワードを検出またはリセットした後、`--password` (または `-p`) オプションを使用して `root` として再度ログオンします:

```
shell> mysql -u root -p
Enter password:
```

ただし、`mysqld --initialize-insecure` を使用して MySQL を初期化した場合、サーバーはパスワードを使用せずに `root` として接続できます (詳細は [セクション2.10.1「データディレクトリの初期化」](#)を参照)。これはセキュリティ上のリスクであるため、`root` アカウントのパスワードを設定する必要があります。手順は、[セクション2.10.4「初期 MySQL アカウントの保護」](#)を参照してください。

- 既存の MySQL インストールを新しいバージョンに更新した場合、MySQL のアップグレード手順を実行しましたか。行っていない場合は実行します。付与テーブルの構造は、新機能が追加されるときにしばしば変更されるため、アップグレードしたあとは常に、テーブルの構造が最新であることを確認することをお勧めします。その手順は、[セクション2.11「MySQL のアップグレード」](#)を参照してください。
- クライアントプログラムが接続しようとしたときに次のエラーメッセージを受け取る場合、サーバーはクライアントが生成可能なものよりも新しい形式のパスワードを予期していることを意味します。

```
shell> mysql
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

- クライアントプログラムは、オプションファイルまたは環境変数で指定された接続パラメータを使用することに注意してください。コマンド行に接続パラメータを指定しないときにクライアントプログラムが間違ったデフォルトの接続パラメータを送信していると思われる場合、該当するオプションファイルおよび環境を確認してください。たとえば、オプションなしでクライアントを実行するときに `Access denied` を受け取る場合、いずれかのオプションファイルで古いパスワードを指定していないか確認してください。

`--no-defaults` を指定してクライアントプログラムを呼び出すことによって、オプションファイルの使用をクライアントプログラムによって抑制することができます。例:

```
shell> mysqladmin --no-defaults -u root version
```

クライアントが使用するオプションファイルの一覧は、[セクション4.2.2.2「オプションファイルの使用」](#)にあります。環境変数の一覧は、[セクション4.9「環境変数」](#)にあります。

- 次のエラーが出る場合、誤った `root` パスワードを使用していることを示しています。

```
shell> mysqladmin -u root -pxxxx ver
```

```
Access denied for user 'root'@'localhost' (using password: YES)
```

パスワードを指定していないのに前述のエラーが発生する場合、いずれかのオプションファイルに間違ったパスワードがリストされていることを意味します。前述の項目で説明したように、`--no-defaults` オプションを試してみてください。

パスワードの変更に関する情報は、[セクション6.2.14「アカウントパスワードの割り当て」](#)を参照してください。

`root` パスワードを紛失したか忘れた場合、[セクションB.3.3.2「root のパスワードをリセットする方法」](#)を参照してください。

- `localhost` はローカルホスト名のシノニムで、ホストを明示的に指定しない場合にクライアントが接続を試行するデフォルトホストでもあります。

`--host=127.0.0.1` オプションを使用して、サーバーホストに明示的に名前を付けることができます。これにより、ローカル `mysqld` サーバーへの TCP/IP 接続が発生します。また、ローカルホストの実際のホスト名を使用する `--host` オプションを指定することによって TCP/IP を使用することもできます。この場合、サーバーと同じホスト上でクライアントプログラムを実行していても、ホスト名がサーバーホスト上の `user` テーブル行に指定されていなければなりません。

- `Access denied` というエラーメッセージは、ログインしようとしているユーザー名、接続を試行しているクライアントホスト、およびパスワードを使用したかどうかを通知します。通常では、エラーメッセージ内で指定されたホスト名およびユーザー名に正確に一致する 1 つの行を `user` テーブル内に持つようにします。たとえば、`using password: NO` というメッセージを含むエラーメッセージを受け取る場合、パスワードなしでログインしようとしたことを意味します。
- `mysql -u user_name` を使用してデータベースに接続しようとしたときに `Access denied` エラーを受け取った場合、`user` テーブルにおそらく問題があります。これをチェックするには、`mysql -u root mysql` を実行し、次の SQL ステートメントを発行します。

```
SELECT * FROM user;
```

この結果には、クライアントのホスト名および使用中の MySQL ユーザー名に一致する `Host` および `User` カラムを持つ行が含まれているはずですが。

- MySQL サーバーを実行しているホストではないホストから接続しようとして次のエラーが発生する場合、クライアントホストと一致する `Host` 値を持つ行が `user` テーブルにないということを意味しています。

```
Host ... is not allowed to connect to this MySQL server
```

これは、接続しようとするときに使用するクライアントホスト名およびユーザー名の組み合わせに対するアカウントをセットアップすることによって修正できます。

接続元のマシンの IP アドレスまたはホスト名がわからない場合、`Host` カラム値が `'%'` の行を `user` テーブル内に作成するようにします。そして、そのクライアントマシンから接続しようとしたあとで、`SELECT USER()` クエリーを使用して、実際にどのように接続したかを確認します。そのあと、`user` テーブル行の `'%'` を、ログに表示されている実際のホスト名に変更します。そうしない場合、特定のユーザー名について任意のホストからの接続が可能になるため、システムはセキュアでない状態のままになります。

Linux では、このエラーが発生する可能性がある別の理由として、使用中のバージョンとは異なるバージョンの `glibc` ライブラリでコンパイルされたバイナリ MySQL バージョンを使用しているということがあります。この場合、オペレーティングシステムまたは `glibc` をアップグレードするか、MySQL のソース配布バージョンをダウンロードして自分でコンパイルします。ソース RPM のコンパイルおよびインストールは通常簡単であるため、これは大きな問題ではありません。

- 接続しようとしたときにホスト名を指定したが、ホスト名が非表示または IP アドレスとなっているエラーメッセージを受け取った場合、MySQL サーバーはクライアントホストの IP アドレスを名前に解決しようとしたときにエラーを受け取ったことを意味します。

```
shell> mysqladmin -u root -pxxxx -h some_hostname ver  
Access denied for user 'root'@' (using password: YES)
```

`root` として接続しようとして次のエラーを受け取った場合、`User` カラム値が `'root'` の行が `user` テーブルになく、`mysqld` がクライアントに対してホスト名を解決できないことを意味します。


```
Access denied for user ''@'unknown'
```

これらのエラーは DNS の問題を示しています。これを修正するには、`mysqladmin flush-hosts` を実行して内部 DNS ホストキャッシュをリセットします。セクション5.1.12.3「DNS ルックアップとホストキャッシュ」を参照してください。

いくつかの永続的な解決策を次に示します。

- DNS サーバーの問題を判別して修正します。
- MySQL 付与テーブルにホスト名の代わりに IP アドレスを指定します。
- クライアントマシン名に対するエントリを、Unix の場合は `/etc/hosts` に、Windows の場合は `\windows\hosts` に配置します。
- `skip_name_resolve` システム変数を有効にして `mysqld` を起動します。
- `mysqld` を `--skip-host-cache` オプションで起動します。
- Unix で、サーバーとクライアントを同じマシンで実行している場合、`localhost` に接続します。`localhost` への接続の場合、MySQL プログラムは、クライアントが TCP/IP 接続を確立するための接続パラメータが指定されていないかぎり、Unix ソケットファイルを使用してローカルサーバーへの接続を試みます。詳細は、セクション4.2.4「コマンドオプションを使用した MySQL Server への接続」を参照してください。
- Windows で、サーバーとクライアントを同じマシンで実行していて、サーバーが名前付きパイプ接続をサポートしている場合、ホスト名 `.` (ピリオド) に接続します。`.` への接続には、TCP/IP ではなく名前付きパイプが使用されます。
- `mysql -u root` は動作するが、`mysql -h your_hostname -u root` によって `Access denied (your_hostname` はローカルホストの実際のホスト名) が生成される場合、`user` テーブルにホストの正しい名前がない可能性があります。このときよくある問題として、`user` テーブル行の `Host` 値は、修飾されていないホスト名を指定しているが、システムの名前解決ルーチンは完全修飾ドメイン名を返すということ (またはその逆) があります。たとえば、`user` テーブルにホスト `'pluto'` を含む行があり、ホスト名が `'pluto.example.com'` であることを DNS が MySQL に通知した場合、その行は機能しません。ホストの IP アドレスを `Host` カラム値として含む行を `user` テーブルに追加してみます。(または、ワイルドカードを含む `Host` 値 (`'pluto.%'` など) を使用して、`user` テーブルに行を追加できます。ただし、`%` で終わる `Host` 値を使用することは安全でないため推奨されません。)
- `mysql -u user_name` は動作するが、`mysql -u user_name some_db` は動作しない場合、`some_db` という名前のデータベースに対する特定のユーザーへのアクセス権を付与していません。
- `mysql -u user_name` をサーバーホスト上で実行したときに動作するが、`mysql -h host_name -u user_name` をリモートクライアントホスト上で実行したときに動作しない場合、特定のユーザー名についてリモートホストからサーバーへのアクセスを有効にしていません。
- `Access denied` を取得する理由がわからない場合は、ワイルドカード (`'%'` または `'_'` 文字を含む行) を含む `Host` 値を持つすべての行を `user` テーブルから削除します。非常に一般的なエラーは、`Host = '%'` および `User = 'some_user'` を使用して新しい行を挿入することです。これにより、`localhost` を指定して同じマシンから接続できると考えられます。これが機能しない理由は、デフォルトの権限に `Host = 'localhost'` および `User = ''` の行が含まれているためです。その行には `'%'` よりも具体的な `Host` 値 `'localhost'` があるため、`localhost` から接続するときに新しい行よりも優先して使用されます。正しい手順は、`Host = 'localhost'` および `User = 'some_user'` を使用して 2 行目を挿入するか、`Host = 'localhost'` および `User = ''` を使用して行を削除することです。行を削除した後は、必ず `FLUSH PRIVILEGES` ステートメントを発行して付与テーブルをリロードしてください。セクション6.2.6「アクセス制御、ステージ 1: 接続の検証」も参照してください。
- MySQL サーバーに接続できても、`SELECT ... INTO OUTFILE` または `LOAD DATA` ステートメントを発行するたびに `Access denied` メッセージが表示される場合、`user` テーブルの行で `FILE` 権限が有効になっていません。
- 付与テーブルを直接 (たとえば、`INSERT`、`UPDATE`、または `DELETE` ステートメントを使用して) 変更し、変更が無視されたように思われる場合、サーバーに権限テーブルをリロードさせるために `FLUSH PRIVILEGES` ステートメントまたは `mysqladmin flush-privileges` コマンドを実行する必要があることを覚えておいてください。そうしない場合、サーバーが次回再起動するまで変更の影響はありません。 `UPDATE` ステートメントを使用して `root` のパ

パスワードを変更した後は、権限をフラッシュするまで新しいパスワードを指定する必要はありません。これは、パスワードを変更するまでサーバーが認識しないためです。

- セッションの最中に権限が変更されたと思われる場合、MySQL 管理者によって権限が変更された可能性があります。付与テーブルのリロードは、新しいクライアント接続に影響しますが、[セクション6.2.13「権限変更が有効化される時期」](#)に示すように既存の接続にも影響します。
- Perl、PHP、Python または ODBC プログラムでアクセスの問題が発生した場合は、`mysql -u user_name db_name` または `mysql -u user_name -ppassword db_name` を使用してサーバーに接続してみてください。mysql クライアントを使用して接続できる場合、問題はアクセス権限ではなくプログラムにあります。(`-p` とパスワードの間に空白はありません。 `--password=password` 構文を使用してパスワードを指定することもできます。 `-p` または `--password` オプションを使用してパスワード値を指定しない場合、MySQL はパスワードを要求します。)
- テスト目的で、`--skip-grant-tables` オプションを指定して `mysqld` サーバーを起動します。これにより、MySQL 付与テーブルを変更でき、`SHOW GRANTS` ステートメントを使用して、変更による望ましい影響があるかどうかを確認することができます。変更が完了したら、`mysqladmin flush-privileges` を実行して、権限をリロードするよう `mysqld` サーバーに指示します。これにより、サーバーを停止して再起動することなく新しい付与テーブルの内容の使用を開始することができます。
- すべてが失敗する場合、`mysqld` サーバーをデバッグオプション (`--debug=d,general,query` など) で起動します。これによって、発行された各コマンドに関する情報のほかに、試行された接続についてのホストおよびユーザー情報が出力されます。[セクション5.9.4「DEBUG パッケージ」](#)を参照してください。
- MySQL 付与テーブルにその他の問題があり、[MySQL Community Slack](#) で確認する場合は、常に MySQL 付与テーブルのダンプを提供してください。`mysqldump mysql` コマンドでテーブルをダンプできます。バグレポートを提出するには、[セクション1.6「質問またはバグをレポートする方法」](#)の説明を参照してください。`mysqldump` を実行するには `--skip-grant-tables` を指定して `mysqld` を再起動することが必要な場合もあります。

6.2.22 SQL ベースのアカウントアクティビティ監査

アプリケーションは次のガイドラインを使用することで、データベースアクティビティを MySQL アカウントに関連付ける SQL ベースの監査を実行できます。

MySQL アカウントは、`mysql.user` システムテーブルの行に対応します。クライアントが正常に接続すると、サーバーはこのテーブル内の特定の行にアクセスするクライアントを認証します。この行の `User` および `Host` カラムの値は、アカウントを一意に識別し、アカウント名が SQL ステートメントに書き込まれる `'user_name'@'host_name'` 形式に対応します。

クライアントを認証する際に使用されるアカウントによって、クライアントが持っている権限が特定されます。通常、`CURRENT_USER()` 関数を呼び出すと、このアカウントがどのクライアントユーザー用であるかを特定できます。その値は、アカウントの `user` テーブル行の `User` および `Host` カラムで構成されています。

ただし、`CURRENT_USER()` 値がクライアントユーザーではなく、別のアカウントに対応するという状況もあります。これは、権限チェックがクライアントのアカウントに基づいて実行されないコンテキストで発生します。

- [SQL SECURITY DEFINER](#) 特性を使用して定義されたストアードルーチン (プロシージャおよび関数)
- [SQL SECURITY DEFINER](#) 特性を使用して定義されたビュー
- トリガーとイベント

このようなコンテキストでは、権限チェックは `DEFINER` アカウントと照合して実行され、`CURRENT_USER()` はそのアカウントを参照し、ストアードルーチンまたはビューを呼び出したクライアント、またはトリガーをアクティブにしたクライアントのアカウントは参照しません。クライアントおよびクライアントの接続元ホストによって指定された実際のユーザー名を示す値を返す `USER()` 関数を呼び出すと、呼び出し元のユーザーを特定できます。ただし、`USER()` の値にはワイルドカードが含まれない一方で、(`CURRENT_USER()` によって返される) アカウントの値にはユーザー名およびホスト名のワイルドカードが含まれる可能性があるため、この値は必ずしも、`user` テーブル内のアカウントに直接対応するとはかぎりません。

たとえば、空白のユーザー名は任意のユーザーに一致するため、`'@localhost'` のアカウントを使用すると、クライアントは任意のユーザー名を持つローカルホストから匿名ユーザーとして接続できます。この場合、クライアントがローカルホストから `user1` として接続すると、`USER()` と `CURRENT_USER()` は異なる値を返します:

```
mysql> SELECT USER(), CURRENT_USER();
+-----+-----+
| USER() | CURRENT_USER() |
+-----+-----+
| user1@localhost | @localhost |
+-----+-----+
```

アカウントのホスト名部分にワイルドカードを含めることもできます。ホスト名に '%' または '_' パターン文字が含まれているか、ネットマスク表記法を使用している場合、アカウントは複数のホストから接続しているクライアントに使用でき、`CURRENT_USER()` 値はどちらかを示しません。たとえば、アカウント `'user2'@'%example.com'` を使用すると、`user2` が `example.com` ドメイン内の任意のホストから接続できます。`user2` が `remote.example.com` から接続すると、`USER()` と `CURRENT_USER()` は別々の値を返します。

```
mysql> SELECT USER(), CURRENT_USER();
+-----+-----+
| USER() | CURRENT_USER() |
+-----+-----+
| user2@remote.example.com | user2@%example.com |
+-----+-----+
```

アプリケーションがユーザーを監査するために `USER()` を呼び出す必要があるが (たとえば、トリガー内から監査を実行する場合)、`USER()` の値を `user` テーブル内のアカウントに関連付けることができる必要がある場合は、アカウントの `User` または `Host` カラムにワイルドカードが含まれることを回避する必要があります。特に、`User` を (匿名のユーザーアカウントが作成される) 空にすることは許可しないでください。また、`Host` の値に、パターン文字またはネットマスク表記を使用することも許可しないでください。すべてのアカウントには、空でない `User` 値とリテラルの `Host` 値を含める必要があります。

前述の例に関しては、ワイルドカードが使用されないように `'@localhost'` および `'user2'@'%example.com'` アカウントを変更するようにしてください。

```
RENAME USER '@localhost' TO 'user1'@'localhost';
RENAME USER 'user2'@'%example.com' TO 'user2'@'remote.example.com';
```

`user2` が `example.com` ドメイン内の複数のホストから接続できる必要がある場合は、ホストごとに個別のアカウントにするべきです。

`CURRENT_USER()` または `USER()` の値からユーザー名またはホスト名の部分を抽出するには、`SUBSTRING_INDEX()` 関数を使用します。

```
mysql> SELECT SUBSTRING_INDEX(CURRENT_USER(), '@', 1);
+-----+
| SUBSTRING_INDEX(CURRENT_USER(), '@', 1) |
+-----+
| user1 |
+-----+

mysql> SELECT SUBSTRING_INDEX(CURRENT_USER(), '@', -1);
+-----+
| SUBSTRING_INDEX(CURRENT_USER(), '@', -1) |
+-----+
| localhost |
+-----+
```

6.3 暗号化された接続の使用

MySQL クライアントとサーバー間の暗号化されていない接続では、ネットワークへのアクセス権を持つユーザーがすべてのトラフィックを監視し、クライアントとサーバー間で送受信されるデータを検査できます。

セキュアな方法でネットワーク経由で情報を移動する必要がある場合、暗号化されていない接続は受け入れられません。あらゆる種類のデータを読み取れないようにするには、暗号化を使用します。暗号化されたメッセージの順序を変更したり、データを2回再生したりするなどの、多くの種類の既知の攻撃に対抗するために、暗号化アルゴリズムには、セキュリティ要素を含める必要があります。

MySQL では、TLS (トランスポート層セキュリティ) プロトコルを使用したクライアントとサーバー間の暗号化された接続をサポートしています。TLS は SSL (Secure Sockets Layer) と呼ばれることもありますが、暗号化が弱いいため、MySQL は実際には暗号化された接続に SSL プロトコルを使用しません ([セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#) を参照)。

TLS は暗号化アルゴリズムを使用して、パブリックネットワーク経由で受信したデータを確実に信頼できるようにします。データの変更、損失またはリプレイを検出するメカニズムがあります。TLS には、X.509 標準を使用したアイデンティティ検証を提供するアルゴリズムも組み込まれています。

X.509 インターネット上のユーザーを識別できるようにします。基本的な用語には、必要とするすべてのユーザーに電子証明書を割り当てる「認証局」(CA)と呼ばれるエンティティーがいくつか存在するはずですが、証明書は、2つの暗号化鍵(公開鍵と秘密鍵)を持つ非対称の暗号化アルゴリズムに依存しています。証明書の所有者は、別のパーティにアイデンティティ証明として証明書を提示できます。証明書は、所有者の公開鍵で構成されます。この公開キーを使用して暗号化されたデータは、証明書の所有者によって保持されている対応する秘密キーを使用してのみ復号化できます。

MySQL での暗号化された接続のサポートは、OpenSSL を使用して提供されます。OpenSSL でサポートされている暗号化プロトコルおよび暗号の詳細は、[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#)を参照してください。

注記

MySQL 8.0.11 から 8.0.17 に、OpenSSL のかわりに wolfSSL を使用して MySQL をコンパイルできました。MySQL 8.0.18 では、wolfSSL のサポートは削除され、すべての MySQL ビルドで OpenSSL が使用されます。

デフォルトでは、サーバーが暗号化された接続をサポートしている場合、MySQL プログラムは暗号化を使用して接続を試み、暗号化された接続を確立できない場合は暗号化されていない接続にフォールバックします。暗号化された接続の使用に影響するオプションの詳細は、[セクション6.3.1「暗号化接続を使用するための MySQL の構成」](#)および[暗号化接続のコマンドオプション](#)を参照してください。

MySQL は接続ごとに暗号化を実行し、特定のユーザーの暗号化の使用はオプションまたは必須にできます。これにより、個々のアプリケーションの要件に従って、暗号化された接続または暗号化されていない接続を選択できます。暗号化された接続の使用をユーザーに要求する方法の詳細は、[セクション13.7.1.3「CREATE USER ステートメント」](#)の CREATE USER ステートメントの REQUIRE 句に関する説明を参照してください。[セクション5.1.8「サーバーシステム変数」](#)の require_secure_transport システム変数の説明も参照してください。

暗号化された接続は、ソースサーバーとレプリカサーバー間で使用できます。[セクション17.3.1「暗号化接続を使用するためのレプリケーションの設定」](#)を参照してください。

MySQL C API からの暗号化された接続の使用の詳細は、[Support for Encrypted Connections](#) を参照してください。

SSH 接続内から MySQL サーバーホストに暗号化を使用して接続することもできます。例については、[セクション6.3.4「SSH を使用した Windows から MySQL へのリモート接続」](#)を参照してください。

6.3.1 暗号化接続を使用するための MySQL の構成

暗号化された接続を使用するかどうかを示し、適切な証明書およびキーファイルを指定するために、いくつかの構成パラメータを使用できます。このセクションでは、暗号化された接続のためのサーバーおよびクライアントの構成に関する一般的なガイダンスを示します:

- [暗号化された接続のサーバー側の起動構成](#)
- [サーバー側のランタイム構成および暗号化された接続の監視](#)
- [暗号化された接続のクライアント側の構成](#)
- [暗号化された接続の必須としての構成](#)

暗号化された接続は次のコンテキストでも使用できます:

- ソースおよびレプリカレプリケーションサーバー間。[セクション17.3.1「暗号化接続を使用するためのレプリケーションの設定」](#)を参照してください。
- グループレプリケーションサーバー間。[セクション18.5.2「Secure Socket Layer \(SSL\) を使用したグループ通信接続の保護」](#)を参照してください。
- MySQL C API に基づくクライアントプログラム。[Support for Encrypted Connections](#)を参照してください。

必要な証明書およびキーファイルを作成する手順は、[セクション6.3.3「SSL および RSA 証明書とキーの作成」](#) で入手できます。

暗号化された接続のサーバー側の起動構成

サーバー側では、`--ssl` オプションは、サーバーが暗号化された接続を許可するが必要としないことを指定します。このオプションはデフォルトで有効になっているため、明示的に指定する必要はありません。

クライアントが暗号化された接続を使用して接続することを要求するには、`require_secure_transport` システム変数を有効にします。暗号化された接続の必須としての構成を参照してください。

サーバー側の次のシステム変数は、クライアントに暗号化された接続の確立を許可するときにサーバーが使用する証明書および鍵ファイルを指定します:

- `ssl_ca`: 認証局 (CA) 証明書ファイルのパス名。(`ssl_capath` は類似していますが、CA 証明書ファイルのディレクトリのパス名を指定します。)
- `ssl_cert`: サーバー公開キー証明書ファイルのパス名。この証明書はクライアントに送信し、クライアントが持っている CA 証明書に対して認証できます。
- `ssl_key`: サーバー秘密キーファイルのパス名。

たとえば、サーバーで暗号化された接続を有効にするには、`my.cnf` ファイルの次の行でサーバーを起動し、必要に応じてファイル名を変更します:

```
[mysqld]
ssl_ca=ca.pem
ssl_cert=server-cert.pem
ssl_key=server-key.pem
```

暗号化された接続を使用するためにクライアントが必要であることを指定するには、`require_secure_transport` システム変数を有効にします:

```
[mysqld]
ssl_ca=ca.pem
ssl_cert=server-cert.pem
ssl_key=server-key.pem
require_secure_transport=ON
```

各証明書およびキーシステム変数は、PEM 形式のファイルに名前を付けます。必要な証明書およびキーファイルを作成する必要がある場合は、[セクション6.3.3「SSL および RSA 証明書とキーの作成」](#) を参照してください。OpenSSL を使用してコンパイルされた MySQL サーバーは、起動時に欠落している証明書およびキーファイルを自動的に生成できます。[セクション6.3.3.1「MySQL を使用した SSL および RSA 証明書とキーの作成」](#) を参照してください。または、MySQL ソース配布がある場合は、その `mysql-test/std_data` ディレクトリにあるデモンストレーション証明書およびキーファイルを使用して設定をテストできます。

サーバーは証明書と鍵ファイルの自動検出を実行します。暗号化された接続を構成するために、`--ssl` 以外の明示的な `encrypted-connection` オプションが (`ssl_cipher` とともに) 指定されていない場合、サーバーは起動時に自動的に `encrypted-connection` サポートを有効にしようとします:

- サーバーは、`ca.pem`、`server-cert.pem` および `server-key.pem` という名前の有効な証明書およびキーファイルをデータディレクトリで検出すると、クライアントによる暗号化された接続のサポートを有効にします。(ファイルは自動的に生成されている必要はありません。これらの名前を持ち、有効であることが重要です。)
- 有効な証明書およびキーファイルがデータディレクトリに見つからない場合、サーバーは実行を続行しますが、暗号化された接続はサポートしません。

サーバーが自動的に暗号化された接続サポートを有効にすると、エラーログにノートが書き込まれます。CA 証明書が自己署名証明書であることがサーバーで検出されると、エラーログに警告が書き込まれます。(サーバーによって自動的に作成された場合、または `mysql_ssl_rsa_setup` を使用して手動で作成された場合、証明書は自己署名されません。)

MySQL には、サーバー側の暗号化された接続制御用の次のシステム変数も用意されています:

- `ssl_cipher`: 接続の暗号化に許可される暗号のリスト。

- `ssl_crl`: 証明書失効リストを含むファイルのパス名。(`ssl_crlpath` は類似していますが、証明書失効リストファイルのディレクトリのパス名を指定します。)
- `tls_version`, `tls_ciphersuites`: サーバーが暗号化された接続に対して許可する暗号化プロトコルおよび暗号スイート。[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#)を参照してください。たとえば、クライアントがセキュアでないプロトコルを使用できないように `tls_version` を設定できます。

サーバーがサーバー側の暗号化された接続制御用のシステム変数から有効な TLS コンテキストを作成できない場合、サーバーは暗号化された接続をサポートしません。

サーバー側のランタイム構成および暗号化された接続の監視

MySQL 8.0.16 より前は、暗号化された接続サポートを構成する `tls_xxx` および `ssl_xxx` システム変数は、サーバーの起動時にのみ設定できます。したがって、これらのシステム変数は、サーバーがすべての新しい接続に使用する TLS コンテキストを決定します。

MySQL 8.0.16 の時点では、`tls_xxx` および `ssl_xxx` システム変数は動的であり、起動時だけでなく実行時に設定できます。`SET GLOBAL` を使用して変更した場合、新しい値はサーバーが再起動するまで適用されません。`SET PERSIST` を使用して変更した場合、新しい値も後続のサーバーの再起動に引き継がれます。[セクション13.7.6.1「変数代入の SET 構文」](#)を参照してください。ただし、このセクションの後半で説明するように、これらの変数を実行時に変更しても、新しい接続の TLS コンテキストにすぐには影響しません。

TLS コンテキスト関連のシステム変数に対する実行時の変更を可能にする MySQL 8.0.16 の変更に加えて、サーバーは、新しい接続に使用される実際の TLS コンテキストに対する実行時の更新を可能にします。この機能は、SSL 証明書が期限切れになるまで実行されている MySQL サーバーを再起動しないようにする場合などに役立ちます。

初期 TLS コンテキストを作成するために、サーバーは起動時にコンテキスト関連のシステム変数が持つ値を使用します。コンテキスト値を公開するために、サーバーは対応するステータス変数のセットも初期化します。次のテーブルに、TLS コンテキストを定義するシステム変数と、現在アクティブなコンテキスト値を公開する対応するステータス変数を示します。

表 6.11 サーバーメイン接続インタフェース TLS コンテキストのシステム変数およびステータス変数

システム変数名	対応するステータス変数名
<code>ssl_ca</code>	<code>Current_tls_ca</code>
<code>ssl_capath</code>	<code>Current_tls_capath</code>
<code>ssl_cert</code>	<code>Current_tls_cert</code>
<code>ssl_cipher</code>	<code>Current_tls_cipher</code>
<code>ssl_crl</code>	<code>Current_tls_crl</code>
<code>ssl_crlpath</code>	<code>Current_tls_crlpath</code>
<code>ssl_key</code>	<code>Current_tls_key</code>
<code>tls_ciphersuites</code>	<code>Current_tls_ciphersuites</code>
<code>tls_version</code>	<code>Current_tls_version</code>

実行時に TLS コンテキストを再構成するには、次の手順を使用します:

1. 変更する必要がある TLS コンテキスト関連のシステム変数については、それらを新しい値に設定します。
2. `ALTER INSTANCE RELOAD TLS` を実行します。このステートメントは、TLS コンテキスト関連のシステム変数の現在の値からアクティブな TLS コンテキストを再構成します。また、新しいアクティブなコンテキスト値を反映するようにコンテキスト関連のステータス変数を設定します。ステートメントには `CONNECTION_ADMIN` 権限が必要です。
3. `ALTER INSTANCE RELOAD TLS` の実行後に確立された新しい接続は、新しい TLS コンテキストを使用します。既存の接続は影響を受けません。既存の接続を終了する場合は、`KILL` ステートメントを使用します。

システム変数とステータス変数の各ペアのメンバーは、再構成手順の動作によって一時的に異なる値を持つ場合があります:

- [ALTER INSTANCE RELOAD TLS](#) より前のシステム変数を変更しても、TLS コンテキストは変更されません。この時点では、これらの変更は新しい接続には影響せず、対応するコンテキスト関連のシステム変数とステータス変数の値が異なる場合があります。これにより、個々のシステム変数に必要な変更を加え、すべてのシステム変数の変更が行われた後に、[ALTER INSTANCE RELOAD TLS](#) を使用してアクティブな TLS コンテキストを原子的に更新できます。
- [ALTER INSTANCE RELOAD TLS](#) の後、対応するシステム変数とステータス変数の値は同じになります。これは、次にシステム変数を変更されるまで当てはまります。

場合によっては、システム変数を変更せずに TLS コンテキストを再構成するには、[ALTER INSTANCE RELOAD TLS](#) 自体で十分です。[ssl_cert](#) によって指定されたファイル内の証明書の有効期限が切れているとします。既存のファイルの内容を期限切れでない証明書に置き換え、[ALTER INSTANCE RELOAD TLS](#) を実行して新しいファイルの内容を読み取り、新しい接続に使用するだけで十分です。

MySQL 8.0.21 の時点では、サーバーは管理接続インタフェースの独立した接続暗号化構成を実装します。[暗号化された接続に対する管理インタフェースのサポート](#)を参照してください。また、TLS コンテキストをリロードするチャンネル (インタフェース) を指定できる [FOR CHANNEL](#) 句を使用して、[ALTER INSTANCE RELOAD TLS](#) が拡張されます。[セクション13.1.5「ALTER INSTANCE ステートメント」](#)を参照してください。管理インタフェース TLS コンテキストを公開するステータス変数はありませんが、パフォーマンススキーマ [tls_channel_status](#) テーブルはメインインタフェースと管理インタフェースの両方の TLS プロパティを公開します。[セクション27.12.19.11「tls_channel_status テーブル」](#)を参照してください。

メインインタフェース TLS コンテキストを更新すると、次の効果があります:

- 更新により、メイン接続インタフェース上の新しい接続に使用される TLS コンテキストが変更されます。
- この更新では、インタフェースにデフォルト以外の TLS パラメータ値が構成されていないがぎり、管理インタフェース上の新しい接続に使用される TLS コンテキストも変更されます。
- この更新は、ほかの有効なサーバープラグインまたはグループレプリケーションや X プラグイン などのコンポーネントで使用される TLS コンテキストには影響しません:
 - サーバー TLS コンテキスト関連のシステム変数から設定を取得する Group Replication グループ通信接続にメインインタフェース再構成を適用するには、[STOP GROUP_REPLICATION](#) の後に [START GROUP_REPLICATION](#) を実行して Group Replication を停止および再起動する必要があります。
 - X プラグイン は、[セクション20.5.3「X プラグイン での暗号化接続の使用」](#) で説明されているように、プラグインの初期化時に TLS コンテキストを初期化します。その後、このコンテキストは変更されません。

デフォルトでは、[RELOAD TLS](#) アクションはエラーでロールバックされ、構成値で新しい TLS コンテキストの作成が許可されていない場合は影響しません。以前のコンテキスト値は、引き続き新しい接続に使用されます。オプションの [NO ROLLBACK ON ERROR](#) 句が指定され、新しいコンテキストを作成できない場合、ロールバックは発生しません。代わりに、警告が生成され、ステートメントが適用されるインタフェース上の新しい接続の暗号化が無効になります。

接続インタフェースで暗号化された接続を有効または無効にするオプションは、起動時にのみ有効になります。たとえば、[--ssl](#) および [--admin-ssl](#) オプションは、メインインタフェースと管理インタフェースが暗号化された接続をサポートしているかどうかにかかわらず、起動時にのみ影響します。このようなオプションは無視され、実行時の [ALTER INSTANCE RELOAD TLS](#) の操作には影響しません。たとえば、[--ssl=OFF](#) を使用して、メインインタフェースで暗号化された接続を無効にしてサーバーを起動し、TLS を再構成して [ALTER INSTANCE RELOAD TLS](#) を実行し、実行時に暗号化された接続を有効にできます。

暗号化された接続のクライアント側の構成

暗号化された接続の確立に関連するクライアントオプションの完全なリストは、[暗号化接続のコマンドオプション](#) を参照してください。

デフォルトでは、サーバーが暗号化された接続をサポートしている場合、MySQL クライアントプログラムは暗号化された接続を確立しようと、[--ssl-mode](#) オプションを使用してさらに制御できます:

- [--ssl-mode](#) オプションがない場合、クライアントは暗号化を使用して接続を試み、暗号化された接続を確立できない場合は暗号化されていない接続にフォールバックします。これは、明示的な [--ssl-mode=PREFERRED](#) オプションでの動作でもあります。

- `--ssl-mode=REQUIRED` では、クライアントは暗号化された接続を必要とし、確立できない場合は失敗します。
- `--ssl-mode=DISABLED` では、クライアントは暗号化されていない接続を使用します。
- `--ssl-mode=VERIFY_CA` または `--ssl-mode=VERIFY_IDENTITY` では、クライアントは暗号化された接続を必要とし、サーバー CA 証明書および (`VERIFY_IDENTITY` を使用して) 証明書内のサーバーホスト名に対しても検証を行います。

サーバー側で `require_secure_transport` システム変数が有効になっていて、サーバーが暗号化された接続を要求する場合、暗号化されていない接続を確立しようとするると失敗します。暗号化された接続の必須としての構成を参照してください。

クライアント側の次のオプションは、クライアントがサーバーへの暗号化された接続を確立するとき使用する証明書および鍵ファイルを識別します。これらは、サーバー側で使用される `ssl_ca`、`ssl_cert` および `ssl_key` システム変数に似ていますが、`--ssl-cert` および `--ssl-key` はクライアントの公開キーと秘密キーを識別します:

- `--ssl-ca`: 認証局 (CA) 証明書ファイルのパス名。このオプションを使用する場合は、サーバーで使用されるものと同じ証明書を指定する必要があります。(`--ssl-capath` は類似していますが、CA 証明書ファイルのディレクトリのパス名を指定します。)
- `--ssl-cert`: クライアント公開キー証明書ファイルのパス名。
- `--ssl-key`: クライアント秘密キーファイルのパス名。

デフォルトの暗号化で提供されるものと比較してセキュリティを強化するために、クライアントはサーバーで使用されるものと一致する CA 証明書を提供し、ホスト名のアイデンティティ検証を有効にできます。このようにして、サーバーとクライアントは同じ CA 証明書に信頼を配置し、クライアントは接続先のホストが意図したものであることを確認します:

- CA 証明書を指定するには、`--ssl-ca` (または `--ssl-capath`) を使用し、`--ssl-mode=VERIFY_CA` を指定します。
- ホスト名アイデンティティ検証も有効にするには、`--ssl-mode=VERIFY_CA` ではなく `--ssl-mode=VERIFY_IDENTITY` を使用します。

注記

`VERIFY_IDENTITY` を使用したホスト名アイデンティティ検証は、サーバーによって自動的に作成された自己署名証明書、または `mysql_ssl_rsa_setup` を使用して手動で作成された自己署名証明書では機能しません (セクション6.3.3.1「MySQL を使用した SSL および RSA 証明書とキーの作成」を参照)。このような自己署名証明書には、共通名の値としてサーバー名は含まれません。

MySQL 8.0.12 より前では、ホスト名のアイデンティティ検証は、ワイルドカードを使用して共通名を指定する証明書でも機能しません。これは、その名前がサーバー名と比較されるためです。

MySQL には、クライアント側の SSL 制御用に次のオプションも用意されています:

- `--ssl-cipher`: 接続の暗号化に許可される暗号のリスト。
- `--ssl-crl`: 証明書失効リストを含むファイルのパス名。(`--ssl-crlpath` は類似していますが、証明書失効リストファイルのディレクトリのパス名を指定します。)
- `--tls-version`, `--tls-ciphersuites`: 許可されている暗号化プロトコルおよび暗号スイート。セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」を参照してください。

クライアントで使用される MySQL アカウントの暗号化要件によっては、クライアントは、暗号化を使用して MySQL サーバーに接続するための特定のオプションを指定する必要がある場合があります。

特別な暗号化要件がないアカウント、または `REQUIRE SSL` 句を含む `CREATE USER` ステートメントを使用して作成されたアカウントを使用して接続するとします。サーバーが暗号化された接続をサポートしていると仮定すると、クライアントは `--ssl-mode` オプションなしで暗号化を使用して、または明示的な `--ssl-mode=PREFERRED` オプションを使用して接続できます:

```
mysql
```

または:

```
mysql --ssl-mode=PREFERRED
```

REQUIRE SSL 句を使用して作成されたアカウントでは、暗号化された接続を確立できない場合、接続の試行は失敗します。特別な暗号化要件のないアカウントの場合、暗号化された接続を確立できないと、暗号化されていない接続にフォールバックしようとします。暗号化された接続を取得できない場合にフォールバックおよび失敗を防ぐには、次のように接続します:

```
mysql --ssl-mode=REQUIRED
```

アカウントのセキュリティ要件が厳しい場合は、暗号化された接続を確立するために他のオプションを指定する必要があります:

- **REQUIRE X509** 句を使用して作成されたアカウントの場合、クライアントは少なくとも **--ssl-cert** および **--ssl-key** を指定する必要があります。また、サーバーによって提供される公開証明書を検証できるように、**--ssl-ca** (または **--ssl-capath**) をお勧めします。次に例を示します (単一行にコマンドを入力します):

```
mysql --ssl-ca=ca.pem
--ssl-cert=client-cert.pem
--ssl-key=client-key.pem
```

- **REQUIRE ISSUER** または **REQUIRE SUBJECT** 句を使用して作成されたアカウントの場合、暗号化要件は **REQUIRE X509** の場合と同じですが、証明書はそれぞれアカウント定義で指定された問題またはサブジェクトと一致する必要があります。

REQUIRE 句の詳細は、[セクション13.7.1.3「CREATE USER ステートメント」](#) を参照してください。

暗号化の使用を防止し、他の **--ssl-xxx** オプションをオーバーライドするには、**--ssl-mode=DISABLED** を使用してクライアントプログラムを起動します:

```
mysql --ssl-mode=DISABLED
```

サーバーとの現在の接続で暗号化が使用されているかどうかを確認するには、**Ssl_cipher** ステータス変数のセッション値を確認します。値が空の場合、接続は暗号化されません。それ以外の場合、接続は暗号化され、値は暗号化暗号を示します。例:

```
mysql> SHOW SESSION STATUS LIKE 'Ssl_cipher';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Ssl_cipher    | DHE-RSA-AES128-GCM-SHA256 |
+-----+-----+
```

mysql クライアントの場合は、代わりに **STATUS** または **\s** コマンドを使用して、**SSL** 行をチェックすることもできます。

```
mysql> \s
...
SSL: Not in use
...
```

または:

```
mysql> \s
...
SSL: Cipher in use is DHE-RSA-AES128-GCM-SHA256
...
```

暗号化された接続の必須としての構成

一部の MySQL デプロイメントでは、暗号化された接続を使用することが望ましいだけでなく、(規制要件を満たすためなどに) 必須である場合があります。このセクションでは、これを可能にする構成設定について説明します。次のレベルの制御を使用できます:

- クライアントが暗号化された接続を使用して接続することを要求するようにサーバーを構成できます。

- 個々のクライアントプログラムを起動して、サーバーで許可されているが暗号化が必要ない場合でも、暗号化された接続を要求できます。
- 暗号化された接続でのみ使用できるように個々の MySQL アカウントを構成できます。

クライアントが暗号化された接続を使用して接続することを要求するには、`require_secure_transport` システム変数を有効にします。たとえば、サーバー `my.cnf` ファイルに次の行を挿入します:

```
[mysqld]
require_secure_transport=ON
```

または、実行時に値を設定して永続化するには、次のステートメントを使用します:

```
SET PERSIST require_secure_transport=ON;
```

`SET PERSIST` は、実行中の MySQL インスタンスの値を設定します。また、値が保存され、それ以降のサーバーの再起動にも使用されます。[セクション13.7.6.1「変数代入の SET 構文」](#)を参照してください。

`require_secure_transport` を有効にすると、なんらかの形式のセキュアなトランスポートを使用するためにサーバーへのクライアント接続が必要になり、SSL を使用する TCP/IP 接続、またはソケットファイル (Unix の場合) または共有メモリー (Windows の場合) を使用する接続のみがサーバーで許可されます。サーバーはセキュアでない接続試行を拒否し、`ER_SECURE_TRANSPORT_REQUIRED` エラーで失敗します。

サーバーで暗号化が必要かどうかにかかわらず暗号化された接続を必要とするようにクライアントプログラムを起動するには、`REQUIRED`、`VERIFY_CA` または `VERIFY_IDENTITY` の `--ssl-mode` オプション値を使用します。例:

```
mysql --ssl-mode=REQUIRED
mysqldump --ssl-mode=VERIFY_CA
mysqladmin --ssl-mode=VERIFY_IDENTITY
```

暗号化された接続でのみ使用できるように MySQL アカウントを構成するには、アカウントを作成する `CREATE USER` ステートメントに `REQUIRE` 句を含め、その句に必要な暗号化特性を指定します。たとえば、暗号化された接続と有効な X.509 証明書の使用を要求するには、`REQUIRE X509` を使用します:

```
CREATE USER 'jeffrey'@'localhost' REQUIRE X509;
```

`REQUIRE` 句の詳細は、[セクション13.7.1.3「CREATE USER ステートメント」](#)を参照してください。

暗号化要件のない既存のアカウントを変更するには、`ALTER USER` ステートメントを使用します。

6.3.2 暗号化された接続 TLS プロトコルおよび暗号

MySQL では、複数の TLS プロトコルおよび暗号がサポートされており、暗号化された接続を許可するプロトコルおよび暗号を構成できます。現在のセッションで使用されているプロトコルおよび暗号を判別することもできます。

- [サポートされている接続 TLS プロトコル](#)
- [接続 TLS プロトコル構成](#)
- [接続暗号構成](#)
- [接続 TLS プロトコルネゴシエーション](#)
- [現在のクライアントセッション TLS プロトコルおよび暗号のモニタリング](#)

サポートされている接続 TLS プロトコル

MySQL では、TLSv1、TLSv1.1、TLSv1.2 および TLSv1.3 プロトコルを使用して暗号化された接続がサポートされており、安全性の低いものから安全性の高いものの順にリストされています。接続に対して実際に許可されるプロトコルのセットは、複数の要因に従います:

- MySQL configuration. 許可された TLS プロトコルは、サーバー側とクライアント側の両方で、サポートされている TLS プロトコルのサブセットのみを含むように構成できます。両側の設定には、共通のプロトコルが少なくとも 1 つ含まれている必要があります。含まれていないと、接続を試行しても使用するプロトコルをネゴシエートできません。詳細は、[接続 TLS プロトコルネゴシエーション](#)を参照してください。

- システム全体のホスト構成。ホストシステムでは特定の TLS プロトコルのみが許可される場合があります。つまり、MySQL 自体で許可されている場合でも、MySQL 接続では許可されていないプロトコルを使用できません。
- MySQL 構成では TLSv1、TLSv1.1 および TLSv1.2 が許可されていますが、ホストシステム構成では TLSv1.2 以上を使用する接続のみが許可されているとします。この場合、TLSv1 または TLSv1.1 を使用する MySQL 接続は、ホストシステムで許可されていないため、MySQL がそれらを許可するように構成されていても確立できません。
- MySQL 構成で TLSv1、TLSv1.1 および TLSv1.2 が許可されているが、ホストシステム構成では TLSv1.3 以上を使用する接続のみが許可されている場合は、MySQL で許可されているプロトコルがホストシステムで許可されていないため、MySQL 接続を確立できません。

この問題の回避策は次のとおりです：

- システム全体のホスト構成を変更して、追加の TLS プロトコルを許可します。手順については、オペレーティングシステムのドキュメントを参照してください。たとえば、TLS プロトコルを TLSv1.2 以上に制限するために、システムに次の行を含む `/etc/ssl/openssl.cnf` ファイルがあるとします：

```
[system_default_sect]
MinProtocol = TLSv1.2
```

値を低いプロトコルバージョンまたは `None` に変更すると、システムがより許可されます。この回避策には、低い（安全性の低い）プロトコルを許可するというデメリットがあり、セキュリティに悪影響を与える可能性があります。

- ホストシステムの TLS 構成を変更できない場合や変更したくない場合は、ホストシステムで許可されているより高い（よりセキュアな）TLS プロトコルを使用するように MySQL アプリケーションを変更します。これは、より低いプロトコルバージョンのみをサポートする古いバージョンの MySQL では不可能な場合があります。たとえば、TLSv1 は MySQL 5.6.46 より前にサポートされていた唯一のプロトコルであるため、5.6.46 より前のサーバーへの接続の試行は、クライアントがより高いプロトコルバージョンをサポートするより新しい MySQL バージョンからのものであっても失敗します。このような場合は、追加の TLS バージョンをサポートする MySQL のバージョンへのアップグレードが必要になることがあります。
- SSL ライブラリ。SSL ライブラリが特定のプロトコルをサポートしていない場合、MySQL もサポートせず、そのプロトコルを指定する次の説明の一部も適用されません。

注記

TLSv1.3 プロトコルのサポートは、MySQL 8.0.16 (Group Replication コンポーネントの MySQL 8.0.18 時点) で使用できます。また、TLSv1.3 を使用するには、MySQL サーバーとクライアントアプリケーションの両方を OpenSSL 1.1.1 以上を使用してコンパイルする必要があります。

接続 TLS プロトコル構成

サーバー側では、`tls_version` システム変数の値によって、MySQL サーバーが暗号化された接続に対して許可する TLS プロトコルが決まります。`tls_version` の値は、クライアントからの接続、このサーバーインスタンスがソースである通常のソースレプリカレプリケーション接続、グループレプリケーショングループ通信接続、およびこのサーバーインスタンスがドナーであるグループレプリケーション分散リカバリ接続に適用されます。変数値は、このリストの 1 つ以上のカンマ区切りプロトコルバージョンのリストです（大/小文字は区別されません）：TLSv1、TLSv1.1、TLSv1.2、および TLSv1.3（使用可能な場合）。デフォルトでは、この変数は、MySQL のコンパイルに使用される SSL ライブラリでサポートされているすべてのプロトコルをリストします。実行時に `tls_version` の値を確認するには、次のステートメントを使用します：

```
mysql> SHOW GLOBAL VARIABLES LIKE 'tls_version';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| tls_version   | TLSv1,TLSv1.1,TLSv1.2 |
+-----+-----+
```

`tls_version` の値を変更するには、サーバーの起動時に設定します。たとえば、TLSv1.1 または TLSv1.2 プロトコルを使用するが、セキュアでない TLSv1 プロトコルを使用する接続を禁止するには、サーバー `my.cnf` ファイルで次の行を使用します：


```
[mysqld]
tls_version=TLSv1.1,TLSv1.2
```

さらに限定的にして TLSv1.2 接続のみを許可するには、次のように `tls_version` を設定します:

```
[mysqld]
tls_version=TLSv1.2
```

MySQL 8.0.16 では、`tls_version` を実行時に変更することもできます。サーバー側のランタイム構成および暗号化された接続の監視を参照してください。

クライアント側では、`--tls-version` オプションは、クライアントプログラムがサーバーへの接続を許可する TLS プロトコルを指定します。オプション値の形式は、前述の `tls_version` システム変数 (プロトコルバージョンのカンマ区切りリスト) と同じです。

このサーバーインスタンスがレプリカであるソース/レプリカレプリケーション接続の場合、`CHANGE REPLICATION SOURCE TO` ステートメント (MySQL 8.0.23 より前) または `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 より前) の `SOURCE_TLS_VERSION` | `MASTER_TLS_VERSION` オプションで、レプリカがソースへの接続を許可する TLS プロトコルを指定します。オプション値の形式は、前述の `tls_version` システム変数の場合と同じです。セクション 17.3.1 「暗号化接続を使用するためのレプリケーションの設定」を参照してください。

`SOURCE_TLS_VERSION` | `MASTER_TLS_VERSION` に指定できるプロトコルは、SSL ライブラリによって異なります。このオプションは、サーバーの `tls_version` 値に依存せず、影響を受けません。たとえば、レプリカとして機能するサーバーは、`tls_version` を TLSv1.3 に設定して、TLSv1.3 を使用する受信接続のみを許可するように構成できますが、`SOURCE_TLS_VERSION` | `MASTER_TLS_VERSION` を TLSv1.2 に設定して、ソースへの送信レプリカ接続に TLSv1.2 のみを許可するように構成することもできます。

このサーバーインスタンスが分散リカバリを開始する結合メンバー (つまり、クライアント) である Group Replication 分散リカバリ接続の場合、`group_replication_recovery_tls_version` システム変数は、クライアントで許可されるプロトコルを指定します。このオプションは、このサーバーインスタンスがドナーである場合に適用されるサーバー `tls_version` 値に依存せず、影響を受けません。グループレプリケーションサーバーは、通常、そのグループメンバーシップの過程でドナーとしても参加メンバーとしても分散リカバリに参加するため、これら両方のシステム変数を設定する必要があります。セクション 18.5.2 「Secure Socket Layer (SSL) を使用したグループ通信接続の保護」を参照してください。

TLS プロトコル構成は、接続 TLS プロトコルネゴシエーションで説明されているように、特定の接続が使用するプロトコルに影響します。

「穴」をリストに残さないなど、許可されたプロトコルを選択する必要があります。たとえば、次のサーバー構成値には穴がありません:

```
tls_version=TLSv1,TLSv1.1,TLSv1.2,TLSv1.3
tls_version=TLSv1.1,TLSv1.2,TLSv1.3
tls_version=TLSv1.2,TLSv1.3
tls_version=TLSv1.3
```

これらの値には穴があり、使用しないでください:

```
tls_version=TLSv1,TLSv1.2 (TLSv1.1 is missing)
tls_version=TLSv1.1,TLSv1.3 (TLSv1.2 is missing)
```

穴の禁止は、クライアントやレプリカなどの他の構成コンテキストでも適用されます。

許可されたプロトコルのリストは空にできません。TLS バージョンパラメータを空の文字列に設定すると、暗号化された接続を確立できません:

- `tls_version`: サーバーは暗号化された着信接続を許可しません。
- `--tls-version`: クライアントは、サーバーへの暗号化された送信接続を許可しません。
- `MASTER_TLS_VERSION`: レプリカでは、ソースへの暗号化された送信接続は許可されません。

接続暗号構成

暗号のデフォルトセットは暗号化された接続に適用され、これは許可された暗号を明示的に構成することでオーバーライドできます。接続の確立時には、接続の両側で共通の暗号が許可されている必要があります。許可されていない

場合、接続は失敗します。SSL ライブラリは、両側に共通の許可された暗号のうち、優先度が最も高い指定された証明書でサポートされている暗号を選択します。

TLSv1.2 までの TLS プロトコルを使用する暗号化された接続に適用可能な暗号を指定するには:

- サーバー側で `ssl_cipher` システム変数を設定し、クライアントプログラムに `--ssl-cipher` オプションを使用します。
- このサーバーインスタンスがソースである通常のソース/レプリカレプリケーション接続の場合は、`ssl_cipher` システム変数を設定します。このサーバーインスタンスがレプリカである場合、`CHANGE REPLICATION SOURCE TO` ステートメント (MySQL 8.0.23) または `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 より前) に `SOURCE_SSL_CIPHER | MASTER_SSL_CIPHER` オプションを使用します。セクション 17.3.1 「暗号化接続を使用するためのレプリケーションの設定」を参照してください。
- グループレプリケーショングループメンバーの場合、グループレプリケーショングループ通信接続、およびこのサーバーインスタンスがドナーであるグループレプリケーション分散リカバリ接続の場合は、`ssl_cipher` システム変数を設定します。このサーバーインスタンスが参加メンバーであるグループレプリケーション分散リカバリ接続の場合は、`group_replication_recovery_ssl_cipher` システム変数を使用します。セクション 18.5.2 「Secure Socket Layer (SSL) を使用したグループ通信接続の保護」を参照してください。

TLSv1.3 を使用する暗号化された接続の場合、OpenSSL 1.1.1 以上では次の暗号スイートがサポートされており、最初の 3 つはデフォルトで有効になっています:

```
TLS_AES_128_GCM_SHA256
TLS_AES_256_GCM_SHA384
TLS_CHACHA20_POLY1305_SHA256
TLS_AES_128_CCM_SHA256
TLS_AES_128_CCM_8_SHA256
```

許可された TLSv1.3 暗号スイートを明示的に構成するには、次のパラメータを設定します。いずれの場合も、構成値はコロンで区切られたゼロ個以上の暗号スイート名のリストです。

- サーバー側では、`tls_ciphersuites` システム変数を使用します。この変数が設定されていない場合、デフォルト値は `NULL` です。つまり、サーバーは暗号スイートのデフォルトセットを許可します。変数が空の文字列に設定されている場合、暗号スイートは有効にならず、暗号化された接続を確立できません。
- クライアント側では、`--tls-ciphersuites` オプションを使用します。このオプションが設定されていない場合、クライアントは暗号化方式群のデフォルトセットを許可します。このオプションが空の文字列に設定されている場合、暗号スイートは有効にならず、暗号化された接続を確立できません。
- このサーバーインスタンスがソースである通常のソース/レプリカレプリケーション接続には、`tls_ciphersuites` システム変数を使用します。このサーバーインスタンスがレプリカである場合、`CHANGE REPLICATION SOURCE TO` ステートメント (MySQL 8.0.23) または `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 より前) に `SOURCE_TLS_CIPHERSUITES | MASTER_TLS_CIPHERSUITES` オプションを使用します。セクション 17.3.1 「暗号化接続を使用するためのレプリケーションの設定」を参照してください。
- グループレプリケーショングループメンバーの場合、グループレプリケーショングループ通信接続、およびこのサーバーインスタンスがドナーであるグループレプリケーション分散リカバリ接続には、`tls_ciphersuites` システム変数を使用します。このサーバーインスタンスが参加メンバーであるグループレプリケーション分散リカバリ接続の場合は、`group_replication_recovery_tls_ciphersuites` システム変数を使用します。セクション 18.5.2 「Secure Socket Layer (SSL) を使用したグループ通信接続の保護」を参照してください。

注記

暗号スイートのサポートは MySQL 8.0.16 で使用できますが、MySQL サーバーとクライアントアプリケーションの両方が OpenSSL 1.1.1 以上を使用してコンパイルされている必要があります。

MySQL 8.0.16 から 8.0.18 では、`group_replication_recovery_tls_ciphersuites` システム変数および `CHANGE REPLICATION SOURCE TO` ステートメント (MySQL 8.0.23 から) の `SOURCE_TLS_CIPHERSUITES | MASTER_TLS_CIPHERSUITES` オプション、または `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 より前) は使用できません。これらのリリースでは、TLSv1.3 がソース/レプリカレプリケーション接続に使用されている場合、または分散リカバリ (MySQL 8.0.18 からサポートされている) のグループレプリケーションで使用されている場合、レプリケーションソースまたはグループレプリケーションドナーの

サーバーは、デフォルトで有効になっている TLSv1.3 暗号スイートの使用を許可する必要があります。MySQL 8.0.19 から、オプションを使用して、必要に応じてデフォルト以外の暗号スイートのみを含む任意の暗号スイートのクライアントサポートを構成できます。

特定の暗号は、TLS プロトコルネゴシエーションプロセスに影響する特定の TLS プロトコルでのみ機能します。 [接続 TLS プロトコルネゴシエーション](#) を参照してください。

特定のサーバーがサポートする暗号を判別するには、`Ssl_cipher_list` ステータス変数のセッション値を確認します：

```
SHOW SESSION STATUS LIKE 'Ssl_cipher_list';
```

`Ssl_cipher_list` ステータス変数には、使用可能な SSL 暗号がリストされます (非 SSL 接続の場合は空)。MySQL が TLSv1.3 をサポートしている場合、この値には使用可能な TLSv1.3 暗号スイートが含まれます。

TLS.v 1.3 を使用する暗号化された接続の場合、MySQL は SSL ライブラリのデフォルトの暗号化スイートリストを使用します。

TLSv1.2 を介して TLS プロトコルを使用する暗号化された接続の場合、MySQL は次のデフォルトの暗号リストを SSL ライブラリに渡します。

```
ECDHE-ECDSA-AES128-GCM-SHA256
ECDHE-ECDSA-AES256-GCM-SHA384
ECDHE-RSA-AES128-GCM-SHA256
ECDHE-RSA-AES256-GCM-SHA384
ECDHE-ECDSA-AES128-SHA256
ECDHE-RSA-AES128-SHA256
ECDHE-ECDSA-AES256-SHA384
ECDHE-RSA-AES256-SHA384
DHE-RSA-AES128-GCM-SHA256
DHE-DSS-AES128-GCM-SHA256
DHE-RSA-AES128-SHA256
DHE-DSS-AES128-SHA256
DHE-DSS-AES256-GCM-SHA384
DHE-RSA-AES256-SHA256
DHE-DSS-AES256-SHA256
ECDHE-RSA-AES128-SHA
ECDHE-ECDSA-AES128-SHA
ECDHE-RSA-AES256-SHA
ECDHE-ECDSA-AES256-SHA
DHE-DSS-AES128-SHA
DHE-RSA-AES128-SHA
TLS_DHE_DSS_WITH_AES_256_CBC_SHA
DHE-RSA-AES256-SHA
AES128-GCM-SHA256
DH-DSS-AES128-GCM-SHA256
ECDH-ECDSA-AES128-GCM-SHA256
AES256-GCM-SHA384
DH-DSS-AES256-GCM-SHA384
ECDH-ECDSA-AES256-GCM-SHA384
AES128-SHA256
DH-DSS-AES128-SHA256
ECDH-ECDSA-AES128-SHA256
AES256-SHA256
DH-DSS-AES256-SHA256
ECDH-ECDSA-AES256-SHA384
AES128-SHA
DH-DSS-AES128-SHA
ECDH-ECDSA-AES128-SHA
AES256-SHA
DH-DSS-AES256-SHA
ECDH-ECDSA-AES256-SHA
DHE-RSA-AES256-GCM-SHA384
DH-RSA-AES128-GCM-SHA256
ECDH-RSA-AES128-GCM-SHA256
DH-RSA-AES256-GCM-SHA384
ECDH-RSA-AES256-GCM-SHA384
DH-RSA-AES128-SHA256
ECDH-RSA-AES128-SHA256
DH-RSA-AES256-SHA256
ECDH-RSA-AES256-SHA384
ECDHE-RSA-AES128-SHA
ECDHE-ECDSA-AES128-SHA
```

```
ECDHE-RSA-AES256-SHA
ECDHE-ECDSA-AES256-SHA
DHE-DSS-AES128-SHA
DHE-RSA-AES128-SHA
TLS_DHE_DSS_WITH_AES_256_CBC_SHA
DHE-RSA-AES256-SHA
AES128-SHA
DH-DSS-AES128-SHA
ECDH-ECDSA-AES128-SHA
AES256-SHA
DH-DSS-AES256-SHA
ECDH-ECDSA-AES256-SHA
DH-RSA-AES128-SHA
ECDH-RSA-AES128-SHA
DH-RSA-AES256-SHA
ECDH-RSA-AES256-SHA
DES-CBC3-SHA
```

次の暗号制限が適用されます:

- 次の暗号は永続的に制限されています:

```
!DHE-DSS-DES-CBC3-SHA
!DHE-RSA-DES-CBC3-SHA
!ECDH-RSA-DES-CBC3-SHA
!ECDH-ECDSA-DES-CBC3-SHA
!ECDHE-RSA-DES-CBC3-SHA
!ECDHE-ECDSA-DES-CBC3-SHA
```

- 次のカテゴリの暗号は永続的に制限されています:

```
!aNULL
!eNULL
!EXPORT
!LOW
!MD5
!DES
!RC2
!RC4
!PSK
!SSLv3
```

前述の制限付き暗号または暗号カテゴリのいずれかを使用する証明書に `ssl_cert` システム変数を設定してサーバーを起動すると、サーバーは暗号化された接続のサポートを無効にして起動します。

接続 TLS プロトコルネゴシエーション

MySQL での接続試行では、プロトコル互換暗号が両側で使用可能な両側で使用可能な最上位 TLS プロトコルバージョンの使用がネゴシエーションされます。ネゴシエーションプロセスは、サーバーとクライアントのコンパイルに使用される SSL ライブラリ、TLS プロトコルと暗号化暗号構成、使用されるキーサイズなどの要因によって異なります:

- 接続を正常に試行するには、サーバーとクライアントの TLS プロトコル構成で一部のプロトコルを共通に許可する必要があります。
- 同様に、サーバーとクライアントの暗号化の構成では、一部の暗号を共通に許可する必要があります。特定の暗号は特定の TLS プロトコルでのみ機能するため、互換性のある暗号がないかぎり、ネゴシエーションプロセスで使用可能なプロトコルは選択されません。
- TLSv1.3 が使用可能な場合は、可能であれば使用されます。(つまり、サーバーとクライアントの両方の構成で TLSv1.3 が許可されている必要があり、両方で一部の TLSv1.3 互換暗号も許可されている必要があります。) それ以外の場合、MySQL は、可能であれば TLSv1.2 などを使用して、使用可能なプロトコルのリストを続行します。ネゴシエーションは、よりセキュアなプロトコルからよりセキュアでないプロトコルに進みます。ネゴシエーション順序は、プロトコルが構成されている順序とは無関係です。たとえば、ネゴシエーション順序は、`tls_version` の値が `TLSv1,TLSv1.1,TLSv1.2,TLSv1.3` であるか `TLSv1.3,TLSv1.2,TLSv1.1,TLSv1` であるかに関係なく同じです。
- TLSv1.2 は、512 ビット以下のキーサイズのすべての暗号で機能するわけではありません。このようなキーでこのプロトコルを使用するには、サーバー側で `ssl_cipher` システム変数を設定するか、`--ssl-cipher` クライアントオプションを使用して暗号名を明示的に指定します:

```
AES128-SHA
AES128-SHA256
AES256-SHA
AES256-SHA256
CAMELLIA128-SHA
CAMELLIA256-SHA
DES-CBC3-SHA
DHE-RSA-AES256-SHA
RC4-MD5
RC4-SHA
SEED-SHA
```

- セキュリティを向上させるには、RSA キーサイズが 2048 ビット以上の証明書を使用します。

サーバーとクライアントに共通の許可されたプロトコルがなく、共通のプロトコル互換暗号がある場合、サーバーは接続リクエストを終了します。例:

- サーバーが `tls_version=TLSv1.1,TLSv1.2` で構成されている場合:
 - `--tls-version=TLSv1` を使用して起動されたクライアント、および TLSv1 のみをサポートする古いクライアントの接続試行は失敗します。
 - 同様に、`MASTER_TLS_VERSION = 'TLSv1'` を使用して構成されたレプリカ、および TLSv1 のみをサポートする古いレプリカの接続試行も失敗します。
- サーバーが `tls_version=TLSv1` で構成されているか、TLSv1 のみをサポートする古いサーバーである場合:
 - `--tls-version=TLSv1.1,TLSv1.2` で起動されたクライアントの接続試行が失敗します。
 - 同様に、`MASTER_TLS_VERSION = 'TLSv1.1,TLSv1.2'` で構成されたレプリカの接続試行も失敗します。

MySQL では、サポートするプロトコルのリストを指定できます。このリストは、基礎となる SSL ライブラリに直接渡され、最終的には、提供されたリストから実際に有効にするプロトコルがそのライブラリになります。SSL ライブラリによる処理方法の詳細は、MySQL ソースコードおよび OpenSSL `SSL_CTX_new()` のドキュメントを参照してください。

現在のクライアントセッション TLS プロトコルおよび暗号のモニタリング

現在のクライアントセッションが使用する暗号化 TLS プロトコルと暗号を判別するには、`Ssl_version` および `Ssl_cipher` ステータス変数のセッション値を確認します:

```
mysql> SELECT * FROM performance_schema.session_status
WHERE VARIABLE_NAME IN ('Ssl_version','Ssl_cipher');
+-----+-----+
| VARIABLE_NAME | VARIABLE_VALUE |
+-----+-----+
| Ssl_cipher    | DHE-RSA-AES128-GCM-SHA256 |
| Ssl_version   | TLSv1.2        |
+-----+-----+
```

接続が暗号化されていない場合、両方の変数の値は空になります。

6.3.3 SSL および RSA 証明書とキーの作成

次の説明では、MySQL での SSL および RSA サポートに必要なファイルの作成方法について説明します。ファイルの作成は、MySQL 自体が提供する機能を使用するか、`openssl` コマンドを直接起動することで実行できます。

SSL 証明書およびキーファイルを使用すると、MySQL で SSL を使用した暗号化された接続をサポートできます。セクション6.3.1「暗号化接続を使用するための MySQL の構成」を参照してください。

RSA キーファイルを使用すると、MySQL は、`sha256_password` または `caching_sha2_password` プラグインによって認証されたアカウントに対して、暗号化されていない接続を介したセキュアなパスワード交換をサポートできます。セクション6.4.1.3「SHA-256 プラガブル認証」およびセクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」を参照してください。

6.3.3.1 MySQL を使用した SSL および RSA 証明書とキーの作成

MySQL では、SSL を使用した暗号化された接続をサポートするために必要な SSL 証明書およびキーファイルと RSA キーペアファイル、および暗号化されていない接続を介した RSA を使用したセキュアなパスワード交換 (これらのファイルがない場合) を作成するために、次の方法が提供されます:

- MySQL ディストリビューションの場合、サーバーは起動時にこれらのファイルを自動生成できます。
- ユーザーは `mysql_ssl_rsa_setup` ユーティリティを手動で起動できます。
- RPM パッケージや DEB パッケージなどの一部の配布タイプでは、データディレクトリの初期化中に `mysql_ssl_rsa_setup` の起動が発生します。この場合、`openssl` コマンドが使用可能であれば、OpenSSL を使用して MySQL ディストリビューションをコンパイルする必要はありません。

重要

サーバーの自動生成および `mysql_ssl_rsa_setup` は、必要なファイルを簡単に生成できるようにすることで、SSL を使用したバリアの低減に役立ちます。ただし、これらの方法で生成された証明書は自己署名されているため、あまりセキュアでない可能性があります。このようなファイルの使用経験がある場合は、登録された認証局から証明書/キーデータを取得することを検討してください。

- [SSL および RSA ファイルの自動生成](#)
- [mysql_ssl_rsa_setup を使用した SSL および RSA ファイルの手動生成](#)
- [SSL および RSA ファイルの特性](#)

SSL および RSA ファイルの自動生成

OpenSSL を使用してコンパイルされた MySQL ディストリビューションの場合、MySQL サーバーには、起動時に欠落している SSL および RSA ファイルを自動的に生成する機能があります。 `auto_generate_certs`、`sha256_password_auto_generate_rsa_keys` および `caching_sha2_password_auto_generate_rsa_keys` システム変数は、これらのファイルの自動生成を制御します。これらの変数はデフォルトで有効になっています。これらは起動時に有効化して検査できますが、実行時には設定できません。

起動時に、`auto_generate_certs` システム変数が有効で、`--ssl` 以外の SSL オプションが指定されておらず、サーバー側の SSL ファイルがデータディレクトリから欠落している場合、サーバーはサーバー側およびクライアント側の SSL 証明書とキーファイルをデータディレクトリに自動的に生成します。これらのファイルにより、SSL を使用した暗号化されたクライアント接続が可能になります。 [セクション 6.3.1 「暗号化接続を使用するための MySQL の構成」](#) を参照してください。

1. サーバーは、次の名前の SSL ファイルのデータディレクトリをチェックします:

```
ca.pem
server-cert.pem
server-key.pem
```

2. これらのファイルのいずれかが存在する場合、サーバーは SSL ファイルを作成しません。それ以外の場合は、それらに加えて追加のファイルが作成されます:

```
ca.pem           Self-signed CA certificate
ca-key.pem       CA private key
server-cert.pem  Server certificate
server-key.pem   Server private key
client-cert.pem  Client certificate
client-key.pem   Client private key
```

3. サーバーが SSL ファイルを自動生成する場合は、`ca.pem`、`server-cert.pem` および `server-key.pem` ファイルの名前を使用して、対応するシステム変数 (`ssl_ca`、`ssl_cert`、`ssl_key`) を設定します。

これらの条件がすべて満たされている場合、サーバーは起動時に RSA 秘密キー/公開キーのペアファイルをデータディレクトリに自動的に生成: `sha256_password_auto_generate_rsa_keys` または `caching_sha2_password_auto_generate_rsa_keys` システム変数が有効になっており、RSA オプションが指定

されていません。RSA ファイルがデータディレクトリにありません。これらのキーペアファイルを使用すると、`sha256_password` または `caching_sha2_password` プラグインによって認証されたアカウントに対して、暗号化されていない RSA 接続を使用したセキュアなパスワード交換が可能になります。[セクション6.4.1.3「SHA-256 プラグブル認証」](#) および [セクション6.4.1.2「SHA-2 プラグブル認証のキャッシュ」](#) を参照してください。

1. サーバーは、RSA ファイルのデータディレクトリを次の名前でチェックします:

```
private_key.pem Private member of private/public key pair
public_key.pem Public member of private/public key pair
```

2. これらのファイルのいずれかが存在する場合、サーバーは RSA ファイルを作成しません。それ以外の場合は、作成されます。
3. サーバーは RSA ファイルを自動生成する場合、その名前を使用して、対応するシステム変数 (`sha256_password_private_key_path` および `sha256_password_public_key_path`、`caching_sha2_password_private_key_path` および `caching_sha2_password_public_key_path`) を設定します。

mysql_ssl_rsa_setup を使用した SSL および RSA ファイルの手動生成

MySQL デイストリビューションには、SSL および RSA ファイルを生成するために手動で起動できる `mysql_ssl_rsa_setup` ユーティリティが含まれています。このユーティリティはすべての MySQL デイストリビューションに含まれていますが、`openssl` コマンドを使用できる必要があります。使用手順については、[セクション4.4.3「mysql_ssl_rsa_setup — SSL/RSA ファイルの作成」](#) を参照してください。

SSL および RSA ファイルの特性

サーバーまたは `mysql_ssl_rsa_setup` の起動によって自動的に作成される SSL および RSA ファイルには、次の特性があります:

- SSL および RSA キーのサイズは 2048 ビットです。
- SSL CA 証明書は自己署名されています。
- SSL サーバーおよびクライアント証明書は、`sha256WithRSAEncryption` 署名アルゴリズムを使用して CA 証明書およびキーで署名されます。
- SSL 証明書は、適切な証明書タイプ (CA、サーバー、クライアント) で次の共通名 (CN) 値を使用します:

```
ca.pem: MySQL_Server_suffix_Auto_Generated_CA_Certificate
server-cert.pem: MySQL_Server_suffix_Auto_Generated_Server_Certificate
client-cert.pem: MySQL_Server_suffix_Auto_Generated_Client_Certificate
```

`suffix` の値は、MySQL のバージョン番号に基づきます。 `mysql_ssl_rsa_setup` によって生成されたファイルの場合、`--suffix` オプションを使用して接尾辞を明示的に指定できます。

サーバーによって生成されたファイルの場合、CN 値が 64 文字を超えると、名前の `_suffix` 部分は省略されます。

- SSL ファイルには、国 (C)、都道府県 (ST)、組織 (O)、組織単位名 (OU) および電子メールアドレスの空白値があります。
- サーバーまたは `mysql_ssl_rsa_setup` によって作成された SSL ファイルは、生成後 10 年間有効です。
- RSA ファイルは期限切れになりません。
- SSL ファイルには、証明書とキーのペアごとに異なるシリアル番号があります (CA の場合は 1、サーバーの場合は 2、クライアントの場合は 3)。
- サーバーによって自動的に作成されたファイルは、サーバーを実行するアカウントによって所有されます。 `mysql_ssl_rsa_setup` を使用して作成されたファイルは、そのプログラムを起動したユーザーが所有します。これは、プログラムが `root` によって起動され、ファイルを所有するユーザーを指定する `--uid` オプションが指定されている場合に、`chown()` システムコールをサポートするシステムで変更できます。
- Unix および Unix に似たシステムでは、ファイルアクセスモードは、証明書ファイルの場合は 644 (ワールド読取り可能) で、キーファイルの場合は 600 (つまり、サーバーを実行するアカウントによってのみアクセス可能) です。

SSL 証明書の内容を確認するには (たとえば、有効な日付の範囲を確認するには)、`openssl` を直接起動します:

```
openssl x509 -text -in ca.pem
openssl x509 -text -in server-cert.pem
openssl x509 -text -in client-cert.pem
```

次の SQL ステートメントを使用して SSL 証明書の有効期限情報を確認することもできます:

```
mysql> SHOW STATUS LIKE 'Ssl_server_not%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Ssl_server_not_after | Apr 28 14:16:39 2027 GMT |
| Ssl_server_not_before | May 1 14:16:39 2017 GMT |
+-----+-----+
```

6.3.3.2 openssl を使用した SSL 証明書およびキーの作成

このセクションでは、`openssl` コマンドを使用して、MySQL サーバーおよびクライアントで使用する SSL 証明書およびキーファイルを設定する方法について説明します。1 番目の例は、コマンド行から使用する場合などの単純化された手順を示しています。2 番目では、より詳細なものを含むスクリプトを示します。最初の 2 つの例は、Unix で使用するためのものであり、どちらの例でも OpenSSL の一部である `openssl` コマンドが使用されます。3 番目の例では、Windows 上で SSL ファイルを設定する方法について説明します。

注記

ここで説明する手順よりも SSL に必要なファイルを生成する方が簡単です: サーバーで自動生成するか、`mysql_ssl_rsa_setup` プログラムを使用します。 [セクション 6.3.3.1 「MySQL を使用した SSL および RSA 証明書とキーの作成」](#) を参照してください。

重要

証明書および鍵ファイルを生成する際にどの方法を使用するのには関係なく、サーバーおよびクライアントの証明書と鍵で使用される Common Name の値はそれぞれ、CA 証明書で使用されている Common Name の値と異なる必要があります。 そうしないと、OpenSSL を使用してコンパイルされたサーバーで証明書およびキーファイルが機能しません。 この場合の一般的なエラーは、次のとおりです。

```
ERROR 2026 (HY000): SSL connection error:
error:00000001:lib(0):func(0):reason(1)
```

- [例 1: Unix でコマンド行から SSL ファイルを作成する](#)
- [例 2: Unix でスクリプトを使用して SSL ファイルを作成する](#)
- [例 3: Windows で SSL ファイルを作成する](#)

例 1: Unix でコマンド行から SSL ファイルを作成する

次の例には、MySQL サーバーおよびクライアントの証明書および鍵ファイルを作成するためのコマンドセットを示します。 `openssl` コマンドでは、複数のプロンプトに回答する必要があります。すべてのプロンプトに対して Enter キーを押せば、テストファイルを生成できます。本番環境用のファイルを生成するには、空でない回答を提供するようにしてください。

```
# Create clean environment
rm -rf newcerts
mkdir newcerts && cd newcerts

# Create CA certificate
openssl genrsa 2048 > ca-key.pem
openssl req -new -x509 -nodes -days 3600 \
  -key ca-key.pem -out ca.pem

# Create server certificate, remove passphrase, and sign it
# server-cert.pem = public key, server-key.pem = private key
openssl req -newkey rsa:2048 -days 3600 \
  -nodes -keyout server-key.pem -out server-req.pem
openssl rsa -in server-key.pem -out server-key.pem
```

```
openssl x509 -req -in server-req.pem -days 3600 \  
-CA ca.pem -CAkey ca-key.pem -set_serial 01 -out server-cert.pem
```

```
# Create client certificate, remove passphrase, and sign it  
# client-cert.pem = public key, client-key.pem = private key  
openssl req -newkey rsa:2048 -days 3600 \  
-nodes -keyout client-key.pem -out client-req.pem  
openssl rsa -in client-key.pem -out client-key.pem  
openssl x509 -req -in client-req.pem -days 3600 \  
-CA ca.pem -CAkey ca-key.pem -set_serial 01 -out client-cert.pem
```

証明書が生成されたら、それらを確認します。

```
openssl verify -CAfile ca.pem server-cert.pem client-cert.pem
```

次のようなレスポンスが表示されます:

```
server-cert.pem: OK  
client-cert.pem: OK
```

証明書の内容を表示するには (たとえば、証明書が有効な日付の範囲を確認するには)、次のように `openssl` を起動します:

```
openssl x509 -text -in ca.pem  
openssl x509 -text -in server-cert.pem  
openssl x509 -text -in client-cert.pem
```

この時点で、次のようなファイルセットを使用できます。

- `ca.pem`: これを使用して、サーバー側で `ssl_ca` システム変数を設定し、クライアント側で `--ssl-ca` オプションを設定します。(CA 証明書を使用する場合は、両側で同じものを指定する必要があります。)
- `server-cert.pem`, `server-key.pem`: これらを使用して、サーバー側で `ssl_cert` および `ssl_key` システム変数を設定します。
- `client-cert.pem`, `client-key.pem`: これらは、クライアント側の `--ssl-cert` および `--ssl-key` オプションの引数として使用します。

その他の使用手順については、[セクション6.3.1「暗号化接続を使用するための MySQL の構成」](#) を参照してください。

例 2: Unix でスクリプトを使用して SSL ファイルを作成する

次に、MySQL に SSL 証明書および鍵ファイルを設定する方法を示したサンプルスクリプトを示します。スクリプトの実行後、[セクション6.3.1「暗号化接続を使用するための MySQL の構成」](#) の説明に従って SSL 接続用のファイルを使用します。

```
DIR=`pwd`/openssl  
PRIV=$DIR/private  
  
mkdir $DIR $PRIV $DIR/newcerts  
cp /usr/share/ssl/openssl.cnf $DIR  
replace ./demoCA $DIR -- $DIR/openssl.cnf  
  
# Create necessary files: $database, $serial and $new_certs_dir  
# directory (optional)  
  
touch $DIR/index.txt  
echo "01" > $DIR/serial  
  
#  
# Generation of Certificate Authority(CA)  
#  
  
openssl req -new -x509 -keyout $PRIV/cakey.pem -out $DIR/ca.pem \  
-days 3600 -config $DIR/openssl.cnf  
  
# Sample output:  
# Using configuration from /home/jones/openssl/openssl.cnf  
# Generating a 1024 bit RSA private key  
# .....++++++
```

```
# .....+++++
# writing new private key to '/home/jones/openssl/private/cakey.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# ----
# You are about to be asked to enter information to be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# ----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL admin
# Email Address []:

#
# Create server request and key
#
openssl req -new -keyout $DIR/server-key.pem -out \
  $DIR/server-req.pem -days 3600 -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/jones/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# ..+++++
# .....+++++
# writing new private key to '/home/jones/openssl/server-key.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# ----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# ----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL server
# Email Address []:
#
# Please enter the following 'extra' attributes
# to be sent with your certificate request
# A challenge password []:
# An optional company name []:

#
# Remove the passphrase from the key
#
openssl rsa -in $DIR/server-key.pem -out $DIR/server-key.pem

#
# Sign server cert
#
openssl ca -cert $DIR/ca.pem -policy policy_anything \
  -out $DIR/server-cert.pem -config $DIR/openssl.cnf \
  -infiles $DIR/server-req.pem

# Sample output:
# Using configuration from /home/jones/openssl/openssl.cnf
# Enter PEM pass phrase:
# Check that the request matches the signature
# Signature ok
# The Subjects Distinguished Name is as follows
```

```
# countryName      :PRINTABLE:'FI'
# organizationName :PRINTABLE:'MySQL AB'
# commonName       :PRINTABLE:'MySQL admin'
# Certificate is to be certified until Sep 13 14:22:46 2003 GMT
# (365 days)
# Sign the certificate? [y/n]:y
#
#
# 1 out of 1 certificate requests certified, commit? [y/n]y
# Write out database with 1 new entries
# Data Base Updated

#
# Create client request and key
#
openssl req -new -keyout $DIR/client-key.pem -out \
  $DIR/client-req.pem -days 3600 -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/jones/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# .....++++++
# .....++++++
# writing new private key to '/home/jones/openssl/client-key.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# ----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# ----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL user
# Email Address []:
#
# Please enter the following 'extra' attributes
# to be sent with your certificate request
# A challenge password []:
# An optional company name []:

#
# Remove the passphrase from the key
#
openssl rsa -in $DIR/client-key.pem -out $DIR/client-key.pem

#
# Sign client cert
#
openssl ca -cert $DIR/ca.pem -policy policy_anything \
  -out $DIR/client-cert.pem -config $DIR/openssl.cnf \
  -infiles $DIR/client-req.pem

# Sample output:
# Using configuration from /home/jones/openssl/openssl.cnf
# Enter PEM pass phrase:
# Check that the request matches the signature
# Signature ok
# The Subjects Distinguished Name is as follows
# countryName      :PRINTABLE:'FI'
# organizationName :PRINTABLE:'MySQL AB'
# commonName       :PRINTABLE:'MySQL user'
# Certificate is to be certified until Sep 13 16:45:17 2003 GMT
# (365 days)
# Sign the certificate? [y/n]:y
#
#
```

```
# 1 out of 1 certificate requests certified, commit? [y/n]y
# Write out database with 1 new entries
# Data Base Updated

#
# Create a my.cnf file that you can use to test the certificates
#

cat <<EOF > $DIR/my.cnf
[client]
ssl-ca=$DIR/ca.pem
ssl-cert=$DIR/client-cert.pem
ssl-key=$DIR/client-key.pem
[mysqld]
ssl_ca=$DIR/ca.pem
ssl_cert=$DIR/server-cert.pem
ssl_key=$DIR/server-key.pem
EOF
```

例 3: Windows で SSL ファイルを作成する

Windows 用の OpenSSL がシステムにインストールされていない場合は、それをダウンロードします。次に、使用可能なパッケージの概要を示します。

<http://www.slproweb.com/products/Win32OpenSSL.html>

アーキテクチャー (32 ビットまたは 64 ビット) に応じて、Win32 OpenSSL Light または Win64 OpenSSL Light パッケージを選択します。デフォルトのインストール場所は、ダウンロードしたパッケージに応じて C:\OpenSSL-Win32 または C:\OpenSSL-Win64 です。次の手順では、デフォルトの場所が C:\OpenSSL-Win32 であることが前提となっています。64 ビットのパッケージを使用している場合は、必要に応じて、これを変更します。

設定中に「...critical component is missing: Microsoft Visual C++ 2008 Redistributables」というメッセージが発生した場合は、設定を取り消し、アーキテクチャー (32 ビットまたは 64 ビット) に応じて、次のパッケージのいずれかをダウンロードします。

- Visual C++ 2008 Redistributables (x86) (次の場所で入手可能):

<http://www.microsoft.com/downloads/details.aspx?familyid=9B2DA534-3E03-4391-8A4D-074B9F2BC1BF>

- Visual C++ 2008 Redistributables (x64) (次の場所で入手可能):

<http://www.microsoft.com/downloads/details.aspx?familyid=bd2a6171-e2d6-4230-b809-9a8d7548c1b6>

追加のパッケージをインストールしたら、OpenSSL の設定手順を再開します。

インストール時に、インストールパスとしてデフォルトの C:\OpenSSL-Win32 のままにして、デフォルトの「Copy OpenSSL DLL files to the Windows system directory」オプションも選択されたままにします。

インストールが完了したら、C:\OpenSSL-Win32\bin をサーバーの Windows システムパス変数に追加します (Windows のバージョンによっては、次のパス設定手順が若干異なる場合があります):

1. Windows デスクトップで、「マイコンピュータ」アイコンを右クリックして「プロパティ」を選択します。
2. 表示された「システムのプロパティ」メニューから「詳細設定」タブを選択し、「環境変数」ボタンをクリックします。
3. 「システム変数」で「パス」を選択してから、「編集」ボタンをクリックします。「システム変数の編集」のダイアログが表示されます。
4. 末尾に「;C:\OpenSSL-Win32\bin」を追加します (セミコロンに注意してください)。
5. 「OK」を 3 回押します。
6. 新しいコマンドコンソール (Start>Run>cmd.exe) を開いて、OpenSSL が使用可能であることを確認することで、OpenSSL が Path 変数に正しく統合されたことをチェックします。

Microsoft Windows [Version ...]
Copyright (c) 2006 Microsoft Corporation. All rights reserved.

```
C:\Windows\system32>cd \  
  
C:\>openssl  
OpenSSL> exit <<< If you see the OpenSSL prompt, installation was successful.  
  
C:\>
```

OpenSSL がインストールされたら、(このセクションの前半で示した) 例 1 と同様の手順を次のように変更して使用します。

- 次のように Unix コマンドを変更します。

```
# Create clean environment  
rm -rf newcerts  
mkdir newcerts && cd newcerts
```

Windows では、代わりに次のコマンドを使用します。

```
# Create clean environment  
md c:\newcerts  
cd c:\newcerts
```

- コマンド行の末尾に「\」文字が表示されたら、この「\」文字を削除し、コマンド行をすべて 1 行で入力する必要があります。

証明書およびキーファイルを生成した後、それらを SSL 接続に使用するには、[セクション6.3.1「暗号化接続を使用するための MySQL の構成」](#) を参照してください。

6.3.3.3 openssl を使用した RSA キーの作成

このセクションでは、`openssl` コマンドを使用して RSA キーファイルを設定する方法について説明します。RSA キーファイルを使用すると、MySQL は、`sha256_password` および `caching_sha2_password` プラグインによって認証されたアカウントの暗号化されていない接続を介したセキュアなパスワード交換をサポートできます。

注記

RSA に必要なファイルを生成する方法は、ここで説明する手順よりも簡単です: サーバーで自動生成するか、`mysql_ssl_rsa_setup` プログラムを使用します。 [セクション 6.3.3.1「MySQL を使用した SSL および RSA 証明書とキーの作成」](#) を参照してください。

RSA 秘密キーと公開キーのペアのファイルを作成するには、MySQL サーバーの実行に使用されたシステムアカウントにログインして、そのアカウントがファイルを所有するように、次のコマンドを実行します:

```
openssl genrsa -out private_key.pem 2048  
openssl rsa -in private_key.pem -pubout -out public_key.pem
```

これらのコマンドでは、2,048 ビットの鍵が作成されます。より強力なキーを作成するには、大きい値を使用します。

次に、鍵ファイルのアクセスモードを設定します。秘密鍵は、サーバーからのみ読み取り可能にするべきです。一方で、公開鍵は、クライアントユーザーに自由に配布できます。

```
chmod 400 private_key.pem  
chmod 444 public_key.pem
```

6.3.4 SSH を使用した Windows から MySQL へのリモート接続

このセクションでは、SSH を使用してリモート MySQL サーバーへの暗号化された接続を取得する方法について説明します。この情報は、David Carlson <dcarlson@mplcomm.com> によって提供されました。

1. Windows マシン上に SSH クライアントをインストールします。SSH クライアントの比較については、http://en.wikipedia.org/wiki/Comparison_of_SSH_clients を参照してください。
2. Windows SSH クライアントを起動します。 `Host_Name = yourmysqlserver_URL_or_IP` を設定します。サーバーにログインする `userid=your_userid` を設定します。この `userid` 値は、MySQL アカウントのユーザー名と同じでない可能性があります。

3. ポートフォワーディングを設定します。リモート転送 (`local_port: 3306`、`remote_host: yourmysqservername_or_ip`、`remote_port: 3306` を設定します) とローカル転送 (`port: 3306`、`host: localhost`、`remote port: 3306` を設定します) のいずれかを実行します。
4. すべてを保存します。保存しない場合は、次回再実行する必要があります。
5. 作成した SSH セッションを使用して、サーバーにログインします。
6. Windows マシン上で、いくつかの ODBC アプリケーション (Access など) を起動します。
7. 通常と同じ方法で、Windows で新しいファイルを作成し、ODBC ドライバを使用して MySQL へのリンクを作成します。ただし、MySQL ホストサーバーでは、`yourmysqservername` ではなく、`localhost` に入力します。

この時点で、MySQL への ODBC 接続が SSH を使用して暗号化されているはずですが。

6.4 セキュリティコンポーネントおよびプラグイン

MySQL には、セキュリティ機能を実装するいくつかのコンポーネントおよびプラグインが含まれています:

- クライアントによる MySQL Server への接続試行を認証するためのプラグイン。プラグインは、複数の認証プロトコルで使用できます。認証プロセスの概要は、[セクション6.2.17「プラグブル認証」](#)を参照してください。特定の認証プラグインの特性については、[セクション6.4.1「認証プラグイン」](#)を参照してください。
- パスワード強度ポリシーを実装し、潜在的なパスワードの強度を評価するためのパスワード検証コンポーネント。[セクション6.4.3「パスワード検証コンポーネント」](#)を参照してください。
- 機密情報用のセキュアなストレージを提供するキーリングプラグイン。[セクション6.4.4「MySQL キーリング」](#)を参照してください。
- (MySQL Enterprise Edition のみ) サーバープラグインを使用して実装された MySQL Enterprise Audit は、オープン MySQL 監査 API を使用して、特定の MySQL サーバーで実行された接続およびクエリーアクティビティの標準のポリシーベースの監視およびロギングを有効にします。MySQL Enterprise Audit は、Oracle 監査仕様を満たすように設計されており、内部および外部の規制ガイドラインによって管理されるアプリケーションに対して、すぐに使用できる監査およびコンプライアンスソリューションを提供します。[セクション6.4.5「MySQL Enterprise Audit」](#)を参照してください。
- ユーザー定義関数を使用すると、アプリケーションで独自のメッセージイベントを監査ログに追加できます。[セクション6.4.6「監査メッセージコンポーネント」](#)を参照してください。
- (MySQL Enterprise Edition のみ) MySQL Enterprise Firewall は、データベース管理者が受け入れられたステートメントパターンのリストに対する照合に基づいて SQL ステートメントの実行を許可または拒否できるようにするアプリケーションレベルのファイアウォールです。これにより、SQL インジェクションなどの攻撃や、正当なクエリーワークロード特性の外部でアプリケーションを使用することで、アプリケーションを利用しようとする攻撃に対して MySQL Server を強化できます。[セクション6.4.7「MySQL Enterprise Firewall」](#)を参照してください。
- (MySQL Enterprise Edition のみ) プラグインおよび一連のユーザー定義関数を含むプラグインライブラリとして実装される MySQL Enterprise Data Masking and De-Identification。データマスキングでは、実際の値を置換で置換することで機密情報が非表示になります。MySQL Enterprise Data Masking and De-Identification 関数を使用すると、不明瞭化 (識別特性の削除)、フォーマットされたランダムデータの生成、データの置換または置換など、いくつかの方法を使用して既存のデータをマスキングできます。[セクション6.5「MySQL Enterprise Data Masking and De-Identification」](#)を参照してください。

6.4.1 認証プラグイン

次の各セクションでは、MySQL で使用可能なプラグブル認証方式と、これらの方式を実装するプラグインについて説明します。認証プロセスの概要は、[セクション6.2.17「プラグブル認証」](#)を参照してください。

デフォルトのプラグインは、`default_authentication_plugin` システム変数の値で示されます。

6.4.1.1 ネイティブプラグブル認証

MySQL には、ネイティブ認証 (プラグブル認証の導入前から使用されていたパスワードハッシュ方式に基づく認証) を実装する `mysql_native_password` プラグインが含まれています。

次の表に、サーバー側とクライアント側のプラグイン名を示します。

表 6.12 ネイティブパスワード認証用のプラグインおよびライブラリ名

プラグインまたはファイル	プラグインまたはファイル名
サーバー側プラグイン	<code>mysql_native_password</code>
クライアント側プラグイン	<code>mysql_native_password</code>
ライブラリファイル	なし (プラグインは組み込み済み)

次の各セクションでは、ネイティブプラグイン認証に固有のインストールおよび使用方法について説明します:

- [ネイティブプラグイン認証のインストール](#)
- [ネイティブプラグイン認証の使用](#)

MySQL のプラグイン認証に関する一般的な情報については、[セクション6.2.17「プラグイン認証」](#)を参照してください。

ネイティブプラグイン認証のインストール

`mysql_native_password` プラグインは、サーバーおよびクライアントフォームに存在します:

- サーバー側のプラグインはサーバーに組み込まれているため、明示的にロードする必要はなく、アンロードしても無効にすることができません。
- クライアント側プラグインは `libmysqlclient` クライアントライブラリに組み込まれており、`libmysqlclient` に対してリンクされているすべてのプログラムで使用できます。

ネイティブプラグイン認証の使用

MySQL クライアントプログラムでは、デフォルトで `mysql_native_password` が使用されます。 `--default-auth` オプションは、プログラムがどのクライアント側プラグインを使用できるかに関するヒントとして使用できます:

```
shell> mysql --default-auth=mysql_native_password ...
```

6.4.1.2 SHA-2 プラグイン認証のキャッシュ

MySQL には、ユーザーアカウントパスワードの SHA-256 ハッシングを実装する次の 2 つの認証プラグインが用意されています:

- `sha256_password`: 基本 SHA-256 認証を実装します。
- `caching_sha2_password`: SHA-256 認証 (`sha256_password` など) を実装しますが、パフォーマンスを向上させるためにサーバー側でキャッシュを使用し、適用性を高めるための追加機能を備えています。

このセクションでは、キャッシュ SHA-2 認証プラグインについて説明します。元の基本 (非キャッシュ) プラグインの詳細は、[セクション6.4.1.3「SHA-256 プラグイン認証」](#)を参照してください。

重要

MySQL 8.0 では、`caching_sha2_password` が `mysql_native_password` ではなくデフォルトの認証プラグインです。サーバー操作に対するこの変更の影響、およびサーバーとクライアントおよびコネクタとの互換性の詳細は、[優先認証プラグインとしての `caching_sha2_password`](#) を参照してください。

重要

`caching_sha2_password` プラグインで認証されるアカウントを使用してサーバーに接続するには、このセクションで後述するように、RSA キーペアを使用したパスワード交換をサポートするセキュアな接続または暗号化されていない接続を使用する必要があります。どちらの方法でも、`caching_sha2_password` プラグインは MySQL 暗号化機能を使用します。[セクション6.3「暗号化された接続の使用」](#)を参照してください。

注記

`sha256_password` という名前の「sha256」は、プラグインが暗号化に使用する 256 ビットのダイジェスト長を表します。`caching_sha2_password` という名前では、「sha2」はより一般的に暗号化アルゴリズムの SHA-2 クラスを指し、256 ビット暗号化は 1 つのインスタンスです。後者の名前を選択すると、プラグイン名を変更せずに、可能性のあるダイジェスト長を将来拡張するための領域が残されます。

`caching_sha2_password` プラグインには、`sha256_password` と比較して次の利点があります:

- サーバー側では、インメモリーキャッシュを使用すると、再接続時に以前に接続したユーザーの再認証を高速化できます。
- RSA ベースのパスワード交換は、MySQL がリンクされている SSL ライブラリに関係なく使用できます。
- Unix ソケットファイルおよび共有メモリープロトコルを使用するクライアント接続のサポートが提供されます。

次の表に、サーバー側とクライアント側のプラグイン名を示します。

表 6.13 SHA-2 認証用のプラグインおよびライブラリ名

プラグインまたはファイル	プラグインまたはファイル名
サーバー側プラグイン	<code>caching_sha2_password</code>
クライアント側プラグイン	<code>caching_sha2_password</code>
ライブラリファイル	なし (プラグインは組み込み済み)

次の各セクションでは、SHA-2 プラガブル認証のキャッシュに固有のインストールおよび使用方法について説明します:

- [SHA-2 プラガブル認証のインストール](#)
- [SHA-2 プラガブル認証の使用](#)
- [SHA-2 プラガブル認証のキャッシュ操作](#)

MySQL のプラガブル認証に関する一般的な情報については、[セクション 6.2.17 「プラガブル認証」](#) を参照してください。

SHA-2 プラガブル認証のインストール

`caching_sha2_password` プラグインは、サーバーおよびクライアントフォームに存在します:

- サーバー側のプラグインはサーバーに組み込まれているため、明示的にロードする必要はなく、アンロードしても無効にすることができません。
- クライアント側プラグインは `libmysqlclient` クライアントライブラリに組み込まれており、`libmysqlclient` に対してリンクされているすべてのプログラムで使用できます。

サーバー側プラグインは、`sha2_cache_cleaner` 監査プラグインをヘルパーとして使用して、パスワードキャッシュ管理を実行します。`caching_sha2_password` と同様に、`sha2_cache_cleaner` は組み込まれており、インストールする必要はありません。

SHA-2 プラガブル認証の使用

SHA-256 パスワードハッシュ用の `caching_sha2_password` プラグインを使用するアカウントを設定するには、次のステートメントを使用します。ここで、`password` は目的のアカウントパスワードです:

```
CREATE USER 'sha2user'@'localhost'
IDENTIFIED WITH caching_sha2_password BY 'password';
```

サーバーは `caching_sha2_password` プラグインをアカウントに割り当て、それを使用して SHA-256 を使用してパスワードを暗号化し、`mysql.user` システムテーブルの `plugin` および `authentication_string` カラムにそれらの値を格納します。

前述の手順では、`caching_sha2_password` がデフォルトの認証プラグインであると想定していません。`caching_sha2_password` がデフォルトの認証プラグインである場合は、より単純な `CREATE USER` 構文を使用できます。

デフォルトの認証プラグインを `caching_sha2_password` に設定してサーバーを起動するには、サーバーオプションファイルに次の行を入力します:

```
[mysqld]
default_authentication_plugin=caching_sha2_password
```

これにより、`caching_sha2_password` プラグインが新しいアカウントにデフォルトで使用されます。その結果、プラグインに明示的に名前を付けずに、アカウントを作成してそのパスワードを設定できます:

```
CREATE USER 'sha2user'@'localhost' IDENTIFIED BY 'password';
```

`default_authentication_plugin` を `caching_sha2_password` に設定した場合の別の結果は、アカウントの作成に他のプラグインを使用するには、そのプラグインを明示的に指定する必要があります。たとえば、`mysql_native_password` プラグインを使用するには、次のステートメントを使用します:

```
CREATE USER 'nativeuser'@'localhost'
IDENTIFIED WITH mysql_native_password BY 'password';
```

`caching_sha2_password` は、セキュアなトランスポートを介した接続をサポートしています。このセクションで後述する RSA の構成手順に従うと、暗号化されていない RSA 接続を介した暗号化されたパスワード交換もサポートされます。RSA サポートには、次の特性があります:

- サーバー側では、RSA 秘密キーペアファイルと公開キーペアファイルに 2 つのシステム変数が指定されます: `caching_sha2_password_private_key_path` および `caching_sha2_password_public_key_path`。使用するキーファイルの名前がシステム変数のデフォルト値と異なる場合、データベース管理者はサーバーの起動時にこれらの変数を設定する必要があります。
- サーバーは、`caching_sha2_password_auto_generate_rsa_keys` システム変数を使用して RSA キーペアファイルを自動的に生成するかどうかを決定します。 [セクション6.3.3「SSL および RSA 証明書とキーの作成」](#)を参照してください。
- `Caching_sha2_password_rsa_public_key` ステータス変数には、`caching_sha2_password` 認証プラグインで使用される RSA 公開キーの値が表示されます。
- RSA 公開鍵を所有しているクライアントは、あとで説明するように、接続プロセス中にサーバーと RSA 鍵ペアベースのパスワード交換を実行できます。
- `caching_sha2_password` および RSA 鍵ペアベースのパスワード交換で認証されるアカウントによる接続の場合、サーバーはデフォルトで RSA 公開鍵をクライアントに送信しません。クライアントは、必要な公開キーのクライアント側コピーを使用するか、サーバーから公開キーをリクエストできます。

公開キーの信頼できるローカルコピーを使用すると、クライアントはクライアント/サーバープロトコルでラウンドトリップを回避でき、サーバーから公開キーをリクエストするよりも安全です。一方、サーバーから公開キーをリクエストする方が便利で(クライアント側ファイルの管理は不要)、セキュアなネットワーク環境で受け入れられる場合があります。

- コマンドラインクライアントの場合は、`--server-public-key-path` オプションを使用して RSA 公開キーファイルを指定します。 `--get-server-public-key` オプションを使用して、サーバーから公開キーをリクエストします。次のプログラムは、2 つのオプションをサポートしています: `mysql`, `mysqlsh`, `mysqladmin`, `mysqlbinlog`, `mysqlcheck`, `mysqldump`, `mysqlimport`, `mysqlpump`, `mysqlshow`, `mysqlslap`, `mysqltest`, `mysql_upgrade`。
- C API を使用するプログラムの場合は、`mysql_options()` をコールして、`MYSQL_SERVER_PUBLIC_KEY` オプションとファイル名を渡して RSA 公開キーファイルを指定するか、`MYSQL_OPT_GET_SERVER_PUBLIC_KEY` オプションを渡してサーバーから公開キーをリクエストします。
- For replicas, use the `CHANGE REPLICATION SOURCE TO` statement (from MySQL 8.0.23) or `CHANGE MASTER TO` statement (before MySQL 8.0.23) with the `SOURCE_PUBLIC_KEY_PATH` | `MASTER_PUBLIC_KEY_PATH` option to specify the RSA public key file, or the `GET_SOURCE_PUBLIC_KEY` | `GET_MASTER_PUBLIC_KEY` option to request the public key from the source. グループレプリケーションの場合、`group_replication_recovery_public_key_path` および `group_replication_recovery_get_public_key` システム変数は同じ目的で使用されます。

いずれの場合も、有効な公開キーファイルを指定するオプションが指定されている場合は、サーバーから公開キーをリクエストするオプションよりも優先されます。

`caching_sha2_password` プラグインを使用するクライアントの場合、サーバーへの接続時にパスワードがクリアテキストとして公開されることはありません。パスワード転送の方法は、セキュアな接続と RSA 暗号化のどちらを使用するかによって異なります:

- 接続がセキュアな場合、RSA キーペアは不要であり、使用されません。これは、TLS を使用して暗号化された TCP 接続と、Unix ソケットファイルおよび共有メモリー接続に適用されます。パスワードはクリアテキストとして送信されますが、接続がセキュアであるためスヌープできません。
- 接続がセキュアでない場合は、RSA キーペアが使用されます。これは、TLS および名前付きパイプ接続なしで暗号化されない TCP 接続に適用されます。RSA は、パスワードのスヌーピングを防ぐために、クライアントとサーバー間のパスワード交換にのみ使用されます。サーバーは、暗号化されたパスワードを受信すると復号化します。繰り返し攻撃を防ぐために、スクランブルが暗号化で使用されます。

クライアント接続プロセス中にパスワード交換に RSA キーペアを使用できるようにするには、次の手順を使用します:

1. [セクション6.3.3「SSL および RSA 証明書とキーの作成」](#) の手順を使用して RSA 秘密キーと公開キーのペアのファイルを作成します。
2. 秘密キーファイルと公開キーファイルがデータディレクトリにあり、 `private_key.pem` および `public_key.pem` (`caching_sha2_password_private_key_path` および `caching_sha2_password_public_key_path` システム変数のデフォルト値) という名前である場合、サーバーはそれらを起動時に自動的に使用します。

それ以外の場合、キーファイルに明示的に名前を付けるには、システム変数をサーバーオプションファイルのキーファイル名に設定します。ファイルがサーバーデータディレクトリにある場合、ファイルのフルパス名を指定する必要はありません:

```
[mysqld]
caching_sha2_password_private_key_path=myprivkey.pem
caching_sha2_password_public_key_path=myspubkey.pem
```

キーファイルがデータディレクトリに配置されていない場合、またはキーファイルの場所をシステム変数値で明示的にする場合は、フルパス名を使用します:

```
[mysqld]
caching_sha2_password_private_key_path=/usr/local/mysql/myprivkey.pem
caching_sha2_password_public_key_path=/usr/local/mysql/mypubkey.pem
```

3. パスワード生成時に `caching_sha2_password` で使用されるハッシュ丸めの数を変更する場合は、 `caching_sha2_password_digest_rounds` システム変数を設定します。例:

```
[mysqld]
caching_sha2_password_digest_rounds=10000
```

4. サーバーを再起動してから接続し、 `Caching_sha2_password_rsa_public_key` ステータス変数値を確認します。実際に表示される値は、次に示す値とは異なりますが、空でない必要があります:

```
mysql> SHOW STATUS LIKE 'Caching_sha2_password_rsa_public_key'G
***** 1. row *****
Variable_name: Caching_sha2_password_rsa_public_key
Value: -----BEGIN PUBLIC KEY-----
MIGfMA0GCsQGS1b3DQEBAQUAA4GNADCBiQKBgQDO9nRUDd+KvSZgY7cNBZMnpwX6
MvE1PbJFXO7u18nJ9lwc99Du/E7lw6CVXw7VKrXPehbVQUzGyUNkf45Nz/ckaaJa
aLgJ0BCIDmNVnyU54OT/1lcs2xiyfaDMe8fCJ64ZwTnKbY2gk11MjUAB5Ogd5kJ
g8aV7EtKwyhHb0c30QIDAQAB
-----END PUBLIC KEY-----
```

値が空の場合は、鍵ファイルに関するいくつかの問題がサーバーで見つかっています。エラーログをチェックして、診断情報を確認してください。

RSA キーファイルを使用してサーバーを構成した後、 `caching_sha2_password` プラグインで認証されるアカウントには、これらのキーファイルを使用してサーバーに接続するオプションがあります。前述のように、このようなアカウ

ントでは、セキュアな接続 (RSA が使用されない場合) または RSA を使用してパスワード交換を実行する暗号化されていない接続のいずれかを使用できます。暗号化されていない接続が使用されるとします。例:

```
shell> mysql --ssl-mode=DISABLED -u sha2user -p
Enter password: password
```

sha2user によるこの接続試行の場合、サーバーは `caching_sha2_password` が適切な認証プラグインであると判断し、それを呼び出します (CREATE USER 時に指定されたプラグインであるため)。プラグインは、接続が暗号化されていないことを検出したため、RSA 暗号化を使用してパスワードを送信する必要があります。ただし、サーバーは公開鍵をクライアントに送信せず、クライアントは公開鍵を提供しなかったため、パスワードを暗号化できず、接続に失敗します:

```
ERROR 2061 (HY000): Authentication plugin 'caching_sha2_password'
reported error: Authentication requires secure connection.
```

RSA 公開鍵をサーバーにリクエストするには、`--get-server-public-key` オプションを指定します:

```
shell> mysql --ssl-mode=DISABLED -u sha2user -p --get-server-public-key
Enter password: password
```

この場合、サーバーは RSA 公開鍵をクライアントに送信し、クライアントはこの鍵を使用してパスワードを暗号化し、その結果をサーバーに返します。プラグインは、サーバー側で RSA 秘密キーを使用してパスワードを復号化し、パスワードが正しいかどうかに基づいて接続を受け入れるか拒否します。

または、サーバーに必要な RSA 公開鍵のローカルコピーを含むファイルがクライアントにある場合は、`--server-public-key-path` オプションを使用してファイルを指定できます:

```
shell> mysql --ssl-mode=DISABLED -u sha2user -p --server-public-key-path=file_name
Enter password: password
```

この場合、クライアントは公開鍵を使用してパスワードを暗号化し、結果をサーバーに返します。プラグインは、サーバー側で RSA 秘密キーを使用してパスワードを復号化し、パスワードが正しいかどうかに基づいて接続を受け入れるか拒否します。

`--server-public-key-path` オプションで指定されたファイル内の公開鍵値

は、`caching_sha2_password_public_key_path` システム変数で指定されたサーバー側ファイル内のキー値と同じである必要があります。鍵ファイルに有効な公開鍵値が含まれているが、その値が正しくない場合は、アクセス拒否のエラーが発生します。鍵ファイルに有効な公開鍵が含まれていない場合は、その鍵をクライアントプログラムで使用できません。

クライアントユーザーは、次の 2 つの方法で RSA 公開鍵を取得できます:

- データベース管理者は、公開鍵ファイルのコピーを提供できます。
- 他の方法でサーバーに接続できるクライアントユーザーは、`SHOW STATUS LIKE 'Caching_sha2_password_rsa_public_key'` ステートメントを使用して、返されたキー値をファイルに保存できます。

SHA-2 プラガブル認証のキャッシュ操作

サーバー側では、`caching_sha2_password` プラグインはメモリー内キャッシュを使用して、以前に接続したクライアントの認証を高速化します。エントリは、`account-name/password-hash` のペアで構成されます。キャッシュは次のように機能します:

1. クライアントが接続すると、`caching_sha2_password` はクライアントとパスワードが一部のキャッシュエントリと一致するかどうかを確認します。その場合、認証は成功します。
2. 一致するキャッシュエントリがない場合、プラグインは `mysql.user` システムテーブルの資格証明と照合してクライアントの検証を試みます。これが成功すると、`caching_sha2_password` はクライアントのエントリをハッシュに追加します。それ以外の場合、認証は失敗し、接続は拒否されます。

このように、クライアントが最初に接続すると、`mysql.user` システムテーブルに対する認証が行われます。その後、クライアントが接続すると、キャッシュに対する認証が高速になります。

エントリの追加以外のパスワードキャッシュ操作は、`caching_sha2_password` の代わりに次のアクションを実行する `sha2_cache_cleaner` 監査プラグインによって処理されます:

- 名前が変更または削除されたアカウント、または資格証明または認証プラグインが変更されたアカウントのキャッシュエントリがクリアされます。
- `FLUSH PRIVILEGES` ステートメントの実行時にキャッシュを空にします。
- サーバーの停止時にキャッシュを空にします。(これは、サーバーの再起動後もキャッシュが永続しないことを意味します。)

キャッシュのクリア操作は、後続のクライアント接続の認証要件に影響します。ユーザーアカウントごとに、次のいずれかの操作後のユーザーの最初のクライアント接続では、セキュアな接続 (TLS 資格証明、Unix ソケットファイルまたは共有メモリーを使用して TCP を使用して確立) または RSA キーペアベースのパスワード交換を使用する必要があります:

- アカウントの作成後。
- アカウントのパスワード変更後。
- アカウントの `RENAME USER` の後。
- `FLUSH PRIVILEGES` の後。

`FLUSH PRIVILEGES` はキャッシュ全体をクリアし、`caching_sha2_password` プラグインを使用するすべてのアカウントに影響します。その他の操作では、特定のキャッシュエントリがクリアされ、操作の一部であるアカウントにのみ影響します。

ユーザーが正常に認証されると、アカウントに影響する別のキャッシュクリアイベントが発生するまで、アカウントはキャッシュに入力され、後続の接続にセキュアな接続または RSA キーペアは必要ありません。(キャッシュを使用できる場合、サーバーはクリアテキストパスワード転送を使用せず、セキュアな接続を必要としないチャレンジレスポンスメカニズムを使用します。)

6.4.1.3 SHA-256 プラガブル認証

MySQL には、ユーザーアカウントパスワードの SHA-256 ハッシングを実装する次の 2 つの認証プラグインが用意されています:

- `sha256_password`: 基本 SHA-256 認証を実装します。
- `caching_sha2_password`: SHA-256 認証 (`sha256_password` など) を実装しますが、パフォーマンスを向上させるためにサーバー側でキャッシュを使用し、適用性を高めるための追加機能を備えています。

このセクションでは、元の非キャッシュ SHA-2 認証プラグインについて説明します。キャッシングプラグインの詳細は、[セクション 6.4.1.2 「SHA-2 プラガブル認証のキャッシュ」](#) を参照してください。

重要

MySQL 8.0 では、`caching_sha2_password` が `mysql_native_password` ではなくデフォルトの認証プラグインです。サーバー操作に対するこの変更の影響、およびサーバーとクライアントおよびコネクタとの互換性の詳細は、[優先認証プラグインとしての `caching_sha2_password`](#) を参照してください。

`caching_sha2_password` は MySQL 8.0 のデフォルトの認証プラグインであり、`sha256_password` 認証プラグインの機能のスーパーセットを提供するため、`sha256_password` は非推奨になりました。将来のバージョンの MySQL で削除される予定です。`sha256_password` を使用して認証する MySQL アカウントは、かわりに `caching_sha2_password` を使用するように移行する必要があります。

重要

`sha256_password` プラグインで認証されるアカウントを使用してサーバーに接続するには、このセクションで後述するように、RSA キーペアを使用したパスワード交換をサポートする TLS 接続または暗号化されていない接続を使用する必要があります。どちらの方法でも、`sha256_password` プラグインは MySQL 暗号化機能を使用します。[セクション 6.3 「暗号化された接続の使用」](#) を参照してください。

注記

`sha256_password` という名前の「sha256」は、プラグインが暗号化に使用する 256 ビットのダイジェスト長を表します。`caching_sha2_password` という名前では、「sha2」はより一般的に暗号化アルゴリズムの SHA-2 クラスを指し、256 ビット暗号化は 1 つのインスタンスです。後者の名前を選択すると、プラグイン名を変更せずに、可能性のあるダイジェスト長を将来拡張するための領域が残されます。

次の表に、サーバー側とクライアント側のプラグイン名を示します。

表 6.14 SHA-256 認証用のプラグインおよびライブラリ名

プラグインまたはファイル	プラグインまたはファイル名
サーバー側プラグイン	<code>sha256_password</code>
クライアント側プラグイン	<code>sha256_password</code>
ライブラリファイル	なし (プラグインは組み込み済み)

次の各セクションでは、SHA-256 プラガブル認証に固有のインストールおよび使用方法について説明します：

- [SHA-256 プラガブル認証のインストール](#)
- [SHA-256 プラガブル認証の使用](#)

MySQL のプラガブル認証に関する一般的な情報については、[セクション6.2.17「プラガブル認証」](#)を参照してください。

SHA-256 プラガブル認証のインストール

`sha256_password` プラグインは、サーバーおよびクライアントフォームに存在します：

- サーバー側のプラグインはサーバーに組み込まれているため、明示的にロードする必要はなく、アンロードしても無効にすることができません。
- クライアント側プラグインは `libmysqlclient` クライアントライブラリに組み込まれており、`libmysqlclient` に対してリンクされているすべてのプログラムで使用できます。

SHA-256 プラガブル認証の使用

SHA-256 パスワードハッシュ用の `sha256_password` プラグインを使用するアカウントを設定するには、次のステートメントを使用します。ここで、`password` は目的のアカウントパスワードです：

```
CREATE USER 'sha256user'@'localhost'
IDENTIFIED WITH sha256_password BY 'password';
```

サーバーは `sha256_password` プラグインをアカウントに割り当て、それを使用して SHA-256 を使用してパスワードを暗号化し、`mysql.user` システムテーブルの `plugin` および `authentication_string` カラムにそれらの値を格納します。

前述の手順では、`sha256_password` がデフォルトの認証プラグインであると想定していません。`sha256_password` がデフォルトの認証プラグインである場合は、より単純な `CREATE USER` 構文を使用できます。

デフォルトの認証プラグインを `sha256_password` に設定してサーバーを起動するには、サーバーオプションファイルに次の行を入力します：

```
[mysqld]
default_authentication_plugin=sha256_password
```

これにより、`sha256_password` プラグインが新しいアカウントにデフォルトで使用されます。その結果、プラグインに明示的に名前を付けずに、アカウントを作成してそのパスワードを設定できます：

```
CREATE USER 'sha256user'@'localhost' IDENTIFIED BY 'password';
```

`default_authentication_plugin` を `sha256_password` に設定した場合の別の結果は、アカウントの作成に他のプラグインを使用するには、そのプラグインを明示的に指定する必要があります。たとえば、`mysql_native_password` プラグインを使用するには、次のステートメントを使用します：

```
CREATE USER 'nativeuser'@'localhost'  
IDENTIFIED WITH mysql_native_password BY 'password';
```

`sha256_password` は、セキュアなトランスポートを介した接続をサポートしています。`sha256_password` では、MySQL が OpenSSL を使用してコンパイルされ、接続先の MySQL サーバーが RSA をサポートするように構成されている場合 (このセクションの後半で説明する RSA 構成手順を使用)、RSA over unencrypted 接続を使用した暗号化パスワード交換もサポートされます。

RSA サポートには、次の特性があります:

- サーバー側では、RSA 秘密キーペアファイルと公開キーペアファイルに 2 つのシステム変数が指定されます: `sha256_password_private_key_path` および `sha256_password_public_key_path`。使用するキーファイルの名前がシステム変数のデフォルト値と異なる場合、データベース管理者はサーバーの起動時にこれらの変数を設定する必要があります。
- サーバーは、`sha256_password_auto_generate_rsa_keys` システム変数を使用して RSA キーペアファイルを自動的に生成するかどうかを決定します。 [セクション6.3.3「SSL および RSA 証明書とキーの作成」](#) を参照してください。
- `Rsa_public_key` ステータス変数には、`sha256_password` 認証プラグインで使用される RSA 公開キーの値が表示されます。
- RSA 公開鍵を所有しているクライアントは、あとで説明するように、接続プロセス中にサーバーと RSA 鍵ペアベースのパスワード交換を実行できます。
- `sha256_password` および RSA 公開キーペアベースのパスワード交換で認証されるアカウントによる接続の場合、サーバーは必要に応じて RSA 公開キーをクライアントに送信します。ただし、公開鍵のコピーがクライアントホストで使用可能な場合、クライアントはそれを使用してラウンドトリップをクライアント/サーバープロトコルに保存できます:
 - これらのコマンドラインクライアントでは、`--server-public-key-path` オプションを使用して RSA 公開キーファイルを指定: `mysql`, `mysqladmin`, `mysqlbinlog`, `mysqlcheck`, `mysqldump`, `mysqlimport`, `mysqlpump`, `mysqlshow`, `mysqlslap`, `mysqltest`, `mysql_upgrade`。
 - C API を使用するプログラムの場合、`mysql_options()` をコールし、`MYSQL_SERVER_PUBLIC_KEY` オプションとファイル名を渡して RSA 公開キーファイルを指定します。
 - レプリカの場合は、`SOURCE_PUBLIC_KEY_PATH | MASTER_PUBLIC_KEY_PATH` オプションを指定して (MySQL 8.0.23 の) `CHANGE REPLICATION SOURCE TO` ステートメントまたは (MySQL 8.0.23 の前の) `CHANGE MASTER TO` ステートメントを使用し、RSA 公開キーファイルを指定します。Group Replication の場合、`group_replication_recovery_get_public_key` システム変数は同じ目的で機能します。

`sha256_password` プラグインを使用するクライアントでは、サーバーへの接続時にパスワードがクリアテキストとして公開されません。パスワード転送の方法は、セキュアな接続と RSA 暗号化のどちらを使用するかによって異なります:

- 接続がセキュアな場合、RSA キーペアは不要であり、使用されません。これは TLS を使用して暗号化された接続に適用されます。パスワードはクリアテキストとして送信されますが、接続がセキュアであるためスヌープできません。

注記

`caching_sha2_password` とは異なり、共有メモリートランスポートはデフォルトでセキュアですが、`sha256_password` プラグインは共有メモリー接続をセキュアとして扱いません。

- 接続がセキュアでなく、RSA キーペアが使用可能な場合、接続は暗号化されません。これは TLS を使用して暗号化されていない接続に適用されます。RSA は、パスワードのスヌーピングを防ぐために、クライアントとサーバー間のパスワード交換にのみ使用されます。サーバーは、暗号化されたパスワードを受信すると復号化します。繰り返しの攻撃を防ぐために、スクランブルが暗号化で使用されます。
- セキュアな接続が使用されておらず、RSA 暗号化が使用できない場合、クリアテキストとして公開されていないパスワードを送信できないため、接続の試行は失敗します。

注記

RSA パスワード暗号化を `sha256_password` で使用するには、クライアントとサーバーの両方が、いずれか一方のみでなく、OpenSSL を使用してコンパイルされている必要があります。

MySQL が OpenSSL を使用してコンパイルされている場合は、次の手順を使用して、クライアント接続プロセス中にパスワード交換に RSA キーペアを使用できるようにします:

1. [セクション6.3.3「SSL および RSA 証明書とキーの作成」](#) の手順を使用して RSA 秘密キーと公開キーのペアのファイルを作成します。
2. 秘密キーファイルと公開キーファイルがデータディレクトリにあり、`private_key.pem` および `public_key.pem` (`sha256_password_private_key_path` および `sha256_password_public_key_path` システム変数のデフォルト値) という名前である場合、サーバーはそれらを起動時に自動的に使用します。

それ以外の場合、キーファイルに明示的に名前を付けるには、システム変数をサーバーオプションファイルのキーファイル名に設定します。ファイルがサーバーデータディレクトリにある場合、ファイルのフルパス名を指定する必要はありません:

```
[mysqld]
sha256_password_private_key_path=myprivkey.pem
sha256_password_public_key_path=myspubkey.pem
```

キーファイルがデータディレクトリに配置されていない場合、またはキーファイルの場所をシステム変数値で明示的にする場合は、フルパス名を使用します:

```
[mysqld]
sha256_password_private_key_path=/usr/local/mysql/myprivkey.pem
sha256_password_public_key_path=/usr/local/mysql/mypubkey.pem
```

3. サーバーを再起動してから、それに接続し、`Rsa_public_key` ステータス変数の値をチェックします。実際に表示される値は、次に示す値とは異なりますが、空でない必要があります:

```
mysql> SHOW STATUS LIKE 'Rsa_public_key'
***** 1. row *****
Variable_name: Rsa_public_key
Value: -----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDO9nRUDd+KvSZgY7cNBZMNPwX6
MvE1PbJFXO7u18nJ9Iwc99Du/E7lw6CVXw7VKrXPeHbVQUzGyUNkf45Nz/ckaaJa
aLgJOBCLDmNVnyU54OT/1lcs2xiyfaDMe8fCJ64ZwTnKbY2gk11MjUAB5Ogd5KJ
g8aV7EtKwyhHb0c30QIDAQAB
-----END PUBLIC KEY-----
```

値が空の場合は、鍵ファイルに関するいくつかの問題がサーバーで見つかっています。エラーログをチェックして、診断情報を確認してください。

RSA キーファイルを使用してサーバーを構成した後、`sha256_password` プラグインで認証されるアカウントには、これらのキーファイルを使用してサーバーに接続するオプションがあります。前述のように、このようなアカウントでは、セキュアな接続 (RSA が使用されない場合) または RSA を使用してパスワード交換を実行する暗号化されていない接続のいずれかを使用できます。暗号化されていない接続が使用されるとします。例:

```
shell> mysql --ssl-mode=DISABLED -u sha256user -p
Enter password: password
```

`sha256user` によるこの接続試行の場合、サーバーは `sha256_password` が適切な認証プラグインであると判断し、それを呼び出します (`CREATE USER` 時に指定されたプラグインであるため)。プラグインは、接続が暗号化されていないことを検出したため、RSA 暗号化を使用してパスワードを送信する必要があります。この場合、プラグインは RSA 公開鍵をクライアントに送信し、クライアントはこの鍵を使用してパスワードを暗号化し、その結果をサーバーに返します。プラグインは、サーバー側で RSA 秘密キーを使用してパスワードを復号化し、パスワードが正しいかどうかに基づいて接続を受け入れるか拒否します。

サーバーは、必要に応じて RSA 公開鍵をクライアントに送信します。ただし、サーバーに必要な RSA 公開キーのローカルコピーを含むファイルがクライアントにある場合は、`--server-public-key-path` オプションを使用してファイルを指定できます:

```
shell> mysql --ssl-mode=DISABLED -u sha256user -p --server-public-key-path=file_name
```

Enter password: `password`

`--server-public-key-path` オプションで指定されたファイル内の公開鍵値は、`sha256_password_public_key_path` システム変数で指定されたサーバー側のファイル内の鍵値と同じにしてください。鍵ファイルに有効な公開鍵値が含まれているが、その値が正しくない場合は、アクセス拒否のエラーが発生します。鍵ファイルに有効な公開鍵が含まれていない場合は、その鍵をクライアントプログラムで使用できません。この場合、`sha256_password` プラグインは、`--server-public-key-path` オプションが指定されていないかのように公開鍵をクライアントに送信します。

クライアントユーザーは、次の 2 つの方法で RSA 公開キーを取得できます：

- データベース管理者は、公開鍵ファイルのコピーを提供できます。
- その他の方法でサーバーに接続できるクライアントユーザーは、`SHOW STATUS LIKE 'Rsa_public_key'` ステートメントを使用し、返された鍵値をファイル内に保存できます。

6.4.1.4 クライアント側クリアテキストプラグブル認証

クライアント側の認証プラグインを使用すると、クライアントはハッシュや暗号化を行わずにクリアテキストでサーバーにパスワードを送信できます。このプラグインは、MySQL クライアントライブラリに組み込まれています。

次の表に、プラグイン名を示します。

表 6.15 クリアテキスト認証用のプラグインおよびライブラリ名

プラグインまたはファイル	プラグインまたはファイル名
サーバー側プラグイン	なし (説明を参照してください)
クライアント側プラグイン	<code>mysql_clear_password</code>
ライブラリファイル	なし (プラグインは組み込み済みです)

クライアント側の認証プラグインの多くは、パスワードをサーバーに送信する前にハッシュ化または暗号化を実行します。これにより、クライアントはクリアテキストとしてパスワードを送信しないようにできます。

クライアント側で入力されたパスワードをサーバーが受信する必要がある認証スキームでは、ハッシュまたは暗号化を実行できません。このような場合、クライアント側の `mysql_clear_password` プラグインが使用され、クライアントはクリアテキストとしてパスワードをサーバーに送信できます。対応するサーバー側のプラグインはありません。かわりに、クリアテキストパスワードを必要とするサーバー側プラグインと組み合わせて、`mysql_clear_password` をクライアント側で使用できます。(PAM および単純な LDAP 認証プラグインなどがあります。[セクション 6.4.1.5 「PAM プラグブル認証」](#) および [セクション 6.4.1.7 「LDAP プラグブル認証」](#) を参照してください。)

次の説明では、クリアテキストプラグブル認証に固有の使用方法について説明します。MySQL のプラグブル認証に関する一般的な情報については、[セクション 6.2.17 「プラグブル認証」](#) を参照してください。

注記

クリアテキストでパスワードを送信すると、一部の構成でセキュリティ上の問題が発生する可能性があります。パスワードが傍受される可能性がある場合に問題を回避するには、クライアントはパスワードが保護される方式を使用して、MySQL サーバーに接続するようにしてください。SSL ([セクション 6.3 「暗号化された接続の使用」](#) を参照)、IPsec、またはプライベートネットワークでも発生する可能性があります。

`mysql_clear_password` プラグインを誤って使用する可能性を低くするには、MySQL クライアントで明示的に有効にする必要があります。これはいくつかの方法で実行できます：

- `LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN` 環境変数を `1`、`Y`、または `y` で始まる値に設定します。これにより、すべてのクライアント接続でプラグインが有効になります。
- `mysql`、`mysqladmin`、`mysqlcheck`、`mysqldump`、`mysqlshow` および `mysqlslap` クライアントプログラムは、プラグインを呼び出しごとに有効にする `--enable-clear-text-plugin` オプションをサポートしています。
- `mysql_options()` C API 関数では、接続するたびにプラグインを有効にする `MYSQL_ENABLE_CLEARTEXT_PLUGIN` オプションがサポートされています。また、クライアントライブラリによって読み取られるオプショングループ内に `enable-clear-text-plugin` を含めると、`libmysqlclient` を使用し、オプションファイルを読み取る任意のプログラムでプラグインを有効にすることができます。

6.4.1.5 PAM プラガブル認証

注記

PAM プラガブル認証は、商用製品である MySQL Enterprise Edition に含まれる拡張機能です。商用製品の詳細は、<https://www.mysql.com/products/> を参照してください。

MySQL Enterprise Edition は、MySQL Server が PAM (Pluggable Authentication Module) を使用して MySQL ユーザーを認証できるようにする認証方式をサポートしています。PAM を使用すると、システムは標準インタフェースを使用して、従来の Unix パスワードや LDAP ディレクトリなどのさまざまな種類の認証方式にアクセスできます。

PAM プラガブル認証は、次の機能を提供します:

- 外部認証: PAM 認証を使用すると、MySQL Server は、MySQL 付与テーブルの外部で定義され、PAM でサポートされている方法を使用して認証するユーザーからの接続を受け入れることができます。
- プロキシユーザーのサポート: PAM 認証は、外部ユーザーがメンバーになっている PAM グループと指定された認証文字列に基づいて、クライアントプログラムによって渡された外部ユーザー名とは異なるユーザー名を MySQL に返すことができます。つまり、このプラグインは、外部 PAM で認証されたユーザーが持つべき権限を定義する MySQL ユーザーを返すことができます。たとえば、`joe` というオペレーティングシステムユーザーは、`developer` という MySQL ユーザーに接続して権限を持つことができます。

PAM プラガブル認証は、Linux および macOS でテストされています。

次の表には、プラグインおよびライブラリファイルの名前を示します。ファイル名のサフィクスは、システムによって異なる場合があります。ファイルは、`plugin_dir` システム変数で指定されたディレクトリに配置する必要があります。インストールに関する情報については、[PAM プラガブル認証のインストール](#) を参照してください。

表 6.16 PAM 認証用のプラグインおよびライブラリ名

プラグインまたはファイル	プラグインまたはファイル名
サーバー側プラグイン	<code>authentication_pam</code>
クライアント側プラグイン	<code>mysql_clear_password</code>
ライブラリファイル	<code>authentication_pam.so</code>

サーバー側 PAM プラグインと通信するクライアント側 `mysql_clear_password` クリアテキストプラグインは、`libmysqlclient` クライアントライブラリに組み込まれており、コミュニティー配布を含むすべての配布に含まれます。すべての MySQL ディストリビューションにクライアント側のクリアテキストプラグインを含めると、任意のディストリビューションのクライアントが、サーバー側 PAM プラグインがロードされているサーバーに接続できるようになります。

次の各セクションでは、PAM プラガブル認証に固有のインストールおよび使用方法について説明します:

- [MySQL ユーザーの PAM 認証の仕組み](#)
- [PAM プラガブル認証のインストール](#)
- [PAM プラガブル認証のアンインストール](#)
- [PAM プラガブル認証の使用](#)
- [プロキシユーザーを使用しない PAM Unix パスワード認証](#)
- [プロキシユーザーを使用しない PAM LDAP 認証](#)
- [プロキシユーザーとグループマッピングを使用した PAM Unix パスワード認証](#)
- [Unix パスワードストアへの PAM 認証アクセス](#)
- [PAM 認証のデバッグ](#)

MySQL のプラガブル認証に関する一般的な情報については、[セクション6.2.17「プラガブル認証」](#) を参照してください。`mysql_clear_password` プラグインの詳細は、[セクション6.4.1.4「クライアント側クリアテキストプラガブル認](#)

証」を参照してください。プロキシユーザーについては、[セクション6.2.18「プロキシユーザー」](#)を参照してください。

MySQL ユーザーの PAM 認証の仕組み

このセクションでは、MySQL と PAM が連携して MySQL ユーザーを認証する方法の概要について説明します。特定の PAM サービスを使用するように MySQL アカウントを設定する方法を示す例については、[PAM プラガブル認証の使用](#)を参照してください。

- クライアントプログラムとサーバーは通信し、クライアントはサーバーにクライアントユーザー名 (デフォルトではオペレーティングシステムのユーザー名) とパスワードを送信します:
 - クライアントユーザー名は外部ユーザー名です。
 - PAM サーバー側認証プラグインを使用するアカウントの場合、対応するクライアント側プラグインは `mysql_clear_password` です。このクライアント側プラグインはパスワードハッシュを実行せず、その結果、クライアントはパスワードをクリアテキストとしてサーバーに送信します。
- サーバーは、外部ユーザー名とクライアントの接続元のホストに基づいて、一致する MySQL アカウントを検索します。PAM プラグインは、MySQL Server によって渡された情報 (ユーザー名、ホスト名、パスワード、認証文字列など) を使用します。PAM を使用して認証する MySQL アカウントを定義する場合、認証文字列には次のものが含まれます:
 - PAM サービス名。システム管理者が特定のアプリケーションの認証方式を参照するために使用できる名前です。単一のデータベースサーバーインスタンスに複数のアプリケーションを関連付けることができるため、サービス名の選択は SQL アプリケーション開発者に任せられます。
 - オプションで、プロキシを使用する場合は、PAM グループから MySQL ユーザー名へのマッピング。
- プラグインは、認証文字列で指定された PAM サービスを使用してユーザー資格証明を確認し、'`Authentication succeeded, Username is user_name`'または'`Authentication failed`'を返します。パスワードは、PAM サービスで使われるパスワードストアに適している必要があります。例:
 - 従来の Unix パスワードの場合、サービスは `/etc/shadow` ファイルに格納されているパスワードを検索します。
 - LDAP の場合、サービスは LDAP ディレクトリに格納されているパスワードを検索します。資格証明チェックが失敗すると、サーバーは接続を拒否します。
- それ以外の場合、認証文字列はプロキシが発生するかどうかを示します。文字列に PAM グループマッピングが含まれていない場合、プロキシは発生しません。この場合、MySQL ユーザー名は外部ユーザー名と同じです。
- それ以外の場合、プロキシは PAM グループマッピングに基づいて示され、MySQL ユーザー名はマッピングリスト内の最初に一致するグループに基づいて決定されます。「PAM グループ」の意味は PAM サービスによって異なります。例:
 - 従来の Unix パスワードの場合、グループは `/etc/group` ファイルで定義された Unix グループで、`/etc/security/group.conf` などのファイル内の追加 PAM 情報を補足する可能性があります。
 - LDAP の場合、グループは LDAP ディレクトリで定義された LDAP グループです。

プロキシユーザー (外部ユーザー) がプロキシ設定された MySQL ユーザー名に対する `PROXY` 権限を持っている場合、プロキシ設定は、プロキシ設定されたユーザーがプロキシ設定されたユーザーの権限を引き受けた状態で行われます。

PAM プラガブル認証のインストール

このセクションでは、PAM 認証プラグインをインストールする方法について説明します。プラグインのインストールについての一般的な情報は、[セクション5.6.1「プラグインのインストールおよびアンインストール」](#)を参照してください。

サーバーで使用できるようにするには、プラグインライブラリファイルを MySQL プラグインディレクトリ (`plugin_dir` システム変数で指定されたディレクトリ) に配置する必要があります。必要に応じて、サーバーの起動時に `plugin_dir` の値を設定してプラグインディレクトリの場所を構成します。

プラグインライブラリファイルのベース名は `authentication_pam` です。ファイル名の接尾辞は、プラットフォームごとに異なります (たとえば、`.so` for Unix and Unix-like systems, `.dll` for Windows)。

サーバーの起動時にプラグインをロードするには、`--plugin-load-add` オプションを使用して、プラグインを含むライブラリファイルに名前を付けます。このプラグインのロード方式では、サーバーを起動するたびにオプションを指定する必要があります。たとえば、サーバー `my.cnf` ファイルに次の行を入力し、必要に応じてプラットフォームの `.so` 接尾辞を調整します:

```
[mysqld]
plugin-load-add=authentication_pam.so
```

`my.cnf` を変更したら、新しい設定を有効にするためにサーバーを再起動します。

または、実行時にプラグインをロードするには、次のステートメントを使用して、必要に応じてプラットフォームの `.so` 接尾辞を調整します:

```
INSTALL PLUGIN authentication_pam SONAME 'authentication_pam.so';
```

`INSTALL PLUGIN` はプラグインをただちにロードし、`mysql.plugins` システムテーブルにも登録して、`--plugin-load-add` を必要とせずに後続の通常の起動のたびにサーバーがプラグインをロードするようにします。

プラグインのインストールを確認するには、`INFORMATION_SCHEMA.PLUGINS` テーブルを調べるか、`SHOW PLUGINS` ステートメントを使用します (セクション5.6.2「サーバープラグイン情報の取得」を参照)。例:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
        FROM INFORMATION_SCHEMA.PLUGINS
        WHERE PLUGIN_NAME LIKE '%pam%';
+-----+-----+
| PLUGIN_NAME | PLUGIN_STATUS |
+-----+-----+
| authentication_pam | ACTIVE |
+-----+-----+
```

プラグインの初期化に失敗した場合は、サーバーエラーログで診断メッセージを確認してください。

MySQL アカウントを PAM プラグインに関連付けるには、[PAM プラガブル認証の使用](#) を参照してください。

PAM プラガブル認証のアンインストール

PAM 認証プラグインのアンインストールに使用される方法は、インストール方法によって異なります:

- `--plugin-load-add` オプションを使用してサーバーの起動時にプラグインをインストールした場合は、オプションなしでサーバーを再起動します。
- `INSTALL PLUGIN` ステートメントを使用して実行時にプラグインをインストールした場合、サーバーの再起動後もインストールされたままになります。アンインストールするには、`UNINSTALL PLUGIN` を使用します:

```
UNINSTALL PLUGIN authentication_pam;
```

PAM プラガブル認証の使用

このセクションでは、PAM 認証プラグインを使用して MySQL クライアントプログラムからサーバーに接続する一般的な用語について説明します。次のセクションでは、PAM 認証を特定の方法で使用する手順について説明します。[PAM プラガブル認証のインストール](#) で説明されているように、サーバーがサーバー側 PAM プラグインを有効にして実行されていることを前提としています。

`CREATE USER` ステートメントの `IDENTIFIED WITH` 句で PAM 認証プラグインを参照するには、`authentication_pam` という名前を使用します。例:

```
CREATE USER user
IDENTIFIED WITH authentication_pam
AS 'auth_string';
```

認証文字列には、次のタイプの情報が指定されます。

- PAM サービス名 ([MySQL ユーザーの PAM 認証の仕組み](#) を参照)。次の説明の例では、従来の Unix パスワードを使用した認証に `mysql-unix` のサービス名を使用し、LDAP を使用した認証に `mysql-ldap` を使用します。

- プロキシサポートのために、PAM は、クライアントプログラムがサーバーに接続するときに渡される外部ユーザー名以外の MySQL ユーザー名を PAM モジュールがサーバーに返す方法を提供します。認証文字列を使用して、外部ユーザー名から MySQL ユーザー名へのマッピングを制御します。プロキシユーザーの機能を活用するには、この種類のマッピングを認証文字列に含める必要があります。

たとえば、アカウントが `mysql-unix` PAM サービス名を使用し、`root` および `users` PAM グループのオペレーティングシステムユーザーを `developer` および `data_entry` MySQL ユーザーにそれぞれマップする必要がある場合は、次のようなステートメントを使用します:

```
CREATE USER user
  IDENTIFIED WITH authentication_pam
  AS 'mysql-unix, root=developer, users=data_entry';
```

PAM 認証プラグインでの認証文字列の構文は、次のようなルールに従っています。

- 文字列は PAM サービス名で構成され、オプションで PAM グループマッピングリストが続きます。PAM グループマッピングリストは、PAM グループ名と MySQL ユーザー名を指定する 1 つ以上のキーワードと値のペアで構成されます:

```
pam_service_name[.pam_group_name=mysql_user_name]...
```

プラグインは、アカウントを使用する各接続試行の認証文字列を解析します。オーバーヘッドを最小限に抑えるには、できるだけ文字列を短く保ちます。

- 各 `pam_group_name=mysql_user_name` ペアの前にカンマを付ける必要があります。
- 二重引用符で囲まれていない先頭および末尾の空白文字は、無視されます。
- 引用符で囲まれていない `pam_service_name`、`pam_group_name` および `mysql_user_name` の値には、等号、カンマまたは空白以外の任意の値を含めることができます。
- `pam_service_name`、`pam_group_name` または `mysql_user_name` の値が二重引用符で囲まれている場合、引用符の間のすべてが値の一部になります。たとえば、値に空白文字が含まれている場合は、これが必要です。二重引用符およびバックスラッシュ (\) を除くすべての文字は有効です。どちらかの文字を含めるには、バックスラッシュを使用してエスケープします。

プラグインが外部ユーザー名 (クライアントから渡された名前) を正常に認証すると、認証文字列内で PAM グループマッピングリストが検索され、存在する場合は、それを使用して、外部ユーザーがメンバーになっている PAM グループに基づいて別の MySQL ユーザー名が MySQL サーバーに返されます:

- 認証文字列に PAM グループマッピングリストが含まれていない場合、プラグインは外部名を返します。
- 認証文字列に PAM グループマッピングリストが含まれている場合、プラグインはリスト内の各 `pam_group_name=mysql_user_name` ペアを左から右に検査し、認証されたユーザーに割り当てられたグループの non-MySQL ディレクトリ内で `pam_group_name` 値の一致を見つけようとし、見つかった最初の一致について `mysql_user_name` を返します。どの PAM グループにも一致するものが見つからない場合、プラグインは外部名を返します。プラグインは、ディレクトリ内のグループを検索できない場合、PAM グループマッピングリストを無視して外部名を返します。

次のセクションでは、PAM 認証プラグインを使用するいくつかの認証シナリオを設定する方法について説明します。

- プロキシユーザーなし。ここでは、ログイン名とパスワードをチェックする際のみ PAM が使用されます。MySQL Server への接続を許可されたすべての外部ユーザーには、PAM 認証を使用するように定義された一致する MySQL アカウントが必要です。('`user_name`@'`host_name`' の MySQL アカウントが外部ユーザーと一致するには、`user_name` が外部ユーザー名であり、`host_name` がクライアントの接続元のホストと一致する必要があります。) PAM でサポートされているさまざまな方式で、認証を実行できます。後で、従来の Unix パスワードおよび LDAP のパスワードを使用してクライアント資格証明を認証する方法について説明します。

PAM 認証は、プロキシユーザーまたは PAM グループを介して行われない場合、MySQL ユーザー名がオペレーティングシステムユーザー名と同じである必要があります。MySQL ユーザー名は 32 文字に制限され ([セクション 6.2.3 「付与テーブル」](#) を参照)、PAM 非プロキシ認証は最大 32 文字の名前を持つ Unix アカウントに制限されません。

- プロキシユーザーのみ (PAM グループマッピングを使用)。このシナリオでは、異なる権限セットを定義する 1 つ以上の MySQL アカウントを作成します。(理想的には、これらのアカウントを使用して誰も接続しないでくださ

い。)次に、PAM を介して認証するデフォルトユーザーを定義します。PAM は、なんらかのマッピングスキーム (通常、ユーザーがメンバーになっている外部 PAM グループに基づく) を使用して、すべての外部ユーザー名を、権限セットを保持する少数の MySQL アカウントにマップします。クライアントユーザー名として外部ユーザー名を接続および指定するクライアントは、いずれかの MySQL アカウントにマップされ、その権限を使用します。ここでは、従来の Unix パスワードを使用してこれを設定する方法を示しますが、LDAP などのほかの PAM 方法を代わりに使用することもできます。

これらのシナリオには、次のバリエーションがあります:

- 一部のユーザーは (プロキシを使用せずに) 直接ログインできますが、他のユーザーはプロキシアカウントを介して接続する必要があります。
- PAM 認証アカウント間で異なる PAM サービス名を使用することで、ある PAM 認証方法を一部のユーザーに使用し、別の方法を他のユーザーに使用できます。たとえば、一部のユーザーには `mysql-unix` PAM サービスを使用し、その他のユーザーには `mysql-ldap` を使用できます。

この例は、次のことが前提となっています。システムが異なる方法で設定されている場合は、多少の調整が必要になることもあります。

- ログイン名とパスワードは、それぞれ `antonio` と `antonio_password` です。これらを、認証するユーザーに対応するように変更します。
- PAM 構成ディレクトリは `/etc/pam.d` です。
- PAM サービス名は、認証方式 (この説明では `mysql-unix` または `mysql-ldap`) に対応します。特定の PAM サービスを使用するには、PAM 構成ディレクトリに同じ名前の PAM ファイルを設定する必要があります (ファイルが存在しない場合は作成します)。また、PAM サービスを使用して認証するアカウントの場合は、`CREATE USER` ステートメントの認証文字列に PAM サービスを指定する必要があります。

PAM 認証プラグインは、サーバーの起動環境で `AUTHENTICATION_PAM_LOG` 環境値が設定されているかどうかを初期化時にチェックします。その場合、プラグインを使用すると、標準出力への診断メッセージのロギングが有効になります。サーバーの起動方法によっては、コンソールまたはエラーログにメッセージが表示される場合があります。これらのメッセージは、プラグインが認証を実行するときに発生する PAM 関連の問題のデバッグに役立ちます。詳細は、[PAM 認証のデバッグ](#) を参照してください。

プロキシユーザーを使用しない PAM Unix パスワード認証

この認証シナリオでは、PAM を使用して、プロキシを使用せずに、オペレーティングシステムユーザー名および Unix パスワードに関して定義された外部ユーザーをチェックします。MySQL Server への接続を許可されたすべての外部ユーザーには、従来の Unix パスワードストアを介した PAM 認証を使用するように定義された、一致する MySQL アカウントが必要です。

注記

従来の Unix パスワードは、`/etc/shadow` ファイルを使用してチェックされます。このファイルに関連して発生する可能性のある問題の詳細は、[Unix パスワードストアへの PAM 認証アクセス](#) を参照してください。

- Unix 認証で、ユーザー名が `antonio` でパスワードが `antonio_password` のオペレーティングシステムへのログインが許可されていることを確認します。
- `/etc/pam.d/mysql-unix` という名前の `mysql-unix` PAM サービスファイルを作成して、従来の Unix パスワードを使用して MySQL 接続を認証するように PAM を設定します。ファイルの内容はシステムに依存するため、`/etc/pam.d` ディレクトリ内の既存のログイン関連ファイルをチェックして、それらがどのように表示されるかを確認します。Linux では、`mysql-unix` ファイルは次のようになります:

```
#%PAM-1.0
auth    include    password-auth
account include    password-auth
```

macOS の場合は、`password-auth` ではなく `login` を使用します。

PAM ファイル形式は、一部のシステムで異なる場合があります。たとえば、Ubuntu およびその他の Debian ベースのシステムでは、かわりに次のファイルコンテンツを使用します:

```
@include common-auth
@include common-account
@include common-session-noninteractive
```

- オペレーティングシステムユーザー名と同じユーザー名で MySQL アカウントを作成し、PAM プラグインおよび `mysql-unix` PAM サービスを使用して認証するように定義します:

```
CREATE USER 'antonio'@'localhost'
IDENTIFIED WITH authentication_pam
AS 'mysql-unix';
GRANT ALL PRIVILEGES
ON mydb.*
TO 'antonio'@'localhost';
```

ここで、認証文字列には PAM サービス名 (`mysql-unix`) のみが含まれており、これによって Unix パスワードが認証されます。

- `mysql` コマンドラインクライアントを使用して、`antonio` として MySQL サーバーに接続します。例:

```
shell> mysql --user=antonio --password --enable-cleartext-plugin
Enter password: antonio_password
```

サーバーは接続を許可する必要があるため、次のクエリーは次のような出力を返します:

```
mysql> SELECT USER(), CURRENT_USER(), @@proxy_user;
+-----+-----+-----+
| USER()          | CURRENT_USER() | @@proxy_user |
+-----+-----+-----+
| antonio@localhost | antonio@localhost | NULL          |
+-----+-----+-----+
```

これは、`antonio` MySQL ユーザーに付与された権限を持つように `antonio` オペレーティングシステムユーザーが認証され、プロキシが発生していないことを示しています。

注記

クライアント側の `mysql_clear_password` 認証プラグインでは、パスワードはそのまま残されるため、クライアントプログラムはクリアテキストとして MySQL サーバーに送信します。これにより、パスワードをそのまま PAM に渡すことができます。サーバー側 PAM ライブラリを使用するにはクリアテキストのパスワードが必要ですが、一部の構成でセキュリティの問題が発生する可能性があります。これらのメジャーにより、リスクが最小限に抑えられます:

- `mysql_clear_password` プラグインを誤って使用する可能性を低くするには、MySQL クライアントで明示的に有効にする必要があります (たとえば、`--enable-cleartext-plugin` オプションを使用)。セクション6.4.1.4「クライアント側クリアテキストプラグブル認証」を参照してください。
- `mysql_clear_password` プラグインを有効にしてパスワードの公開を回避するには、MySQL クライアントは暗号化された接続を使用して MySQL サーバーに接続する必要があります。セクション6.3.1「暗号化接続を使用するための MySQL の構成」を参照してください。

プロキシユーザーを使用しない PAM LDAP 認証

この認証シナリオでは、PAM を使用して、プロキシを使用せずに、オペレーティングシステムのユーザー名および LDAP パスワードに関して定義された外部ユーザーをチェックします。MySQL Server への接続を許可されたすべての外部ユーザーには、LDAP を介した PAM 認証を使用するように定義された一致する MySQL アカウントが必要です。

MySQL で PAM LDAP プラグブル認証を使用するには、次の前提条件を満たす必要があります:

- PAM LDAP サービスが通信するには、LDAP サーバーが使用可能である必要があります。
- MySQL によって認証される LDAP ユーザーは、LDAP サーバーによって管理されるディレクトリに存在する必要があります。

注記

MySQL ユーザー認証に LDAP を使用する別の方法は、LDAP 固有の認証プラグインを使用することです。 [セクション6.4.1.7「LDAP プラガブル認証」](#) を参照してください。

PAM LDAP 認証用の MySQL を次のように構成します:

1. Unix 認証で、ユーザー名が `antonio` でパスワードが `antonio_password` のオペレーティングシステムへのログインが許可されていることを確認します。
2. `/etc/pam.d/mysql-ldap` という名前の `mysql-ldap` PAM サービスファイルを作成して、LDAP を使用して MySQL 接続を認証するように PAM を設定します。ファイルの内容はシステムに依存するため、`/etc/pam.d` ディレクトリ内の既存のログイン関連ファイルをチェックして、それらがどのように表示されるかを確認します。Linux では、`mysql-ldap` ファイルは次のようになります:

```
#%PAM-1.0
auth    required pam_ldap.so
account required pam_ldap.so
```

PAM オブジェクトファイルのサフィクスがシステム上の `.so` と異なる場合は、正しいサフィクスに置き換えてください。

PAM ファイル形式は、一部のシステムで異なる場合があります。

3. オペレーティングシステムユーザー名と同じユーザー名で MySQL アカウントを作成し、PAM プラグインおよび `mysql-ldap` PAM サービスを使用して認証するように定義します:

```
CREATE USER 'antonio'@'localhost'
IDENTIFIED WITH authentication_pam
AS 'mysql-ldap';
GRANT ALL PRIVILEGES
ON mydb.*
TO 'antonio'@'localhost';
```

ここで、認証文字列には PAM サービス名 `mysql-ldap` のみが含まれ、LDAP を使用して認証されます。

4. サーバーへの接続は、[プロキシユーザーを使用しない PAM Unix パスワード認証](#) で説明されている接続と同じです。

プロキシユーザーとグループマッピングを使用した PAM Unix パスワード認証

ここで説明する認証スキームでは、プロキシと PAM グループのマッピングを使用して、PAM を使用して認証する接続 MySQL ユーザーを、異なる特権セットを定義するほかの MySQL アカウントにマップします。ユーザーは、権限を定義するアカウントを使用して直接接続しません。かわりに、PAM を使用して認証されたデフォルトのプロキシアカウントを介して接続し、すべての外部ユーザーが権限を保持する MySQL アカウントにマップされるようにします。プロキシアカウントを使用して接続するユーザーは、外部ユーザーに許可されるデータベース操作を決定する権限である、これらの MySQL アカウントのいずれかにマップされます。

ここに示す手順では、Unix パスワード認証が使用されます。代わりに LDAP を使用するには、前半で示した[プロキシユーザーを使用しない PAM LDAP 認証](#)の手順を参照してください。

注記

従来の Unix パスワードは、`/etc/shadow` ファイルを使用してチェックされます。このファイルに関連して発生する可能性のある問題の詳細は、[Unix パスワードストアへの PAM 認証アクセス](#) を参照してください。

1. Unix 認証で、ユーザー名が `antonio` でパスワードが `antonio_password` のオペレーティングシステムへのログインが許可されていることを確認します。
2. `antonio` が `root` または `users` PAM グループのメンバーであることを確認します。
3. `/etc/pam.d/mysql-unix` という名前のファイルを作成して、オペレーティングシステムユーザーを介して `mysql-unix` PAM サービスを認証するように PAM を設定します。ファイルの内容はシステムに依存するため、`/etc/pam.d`

ディレクトリ内の既存のログイン関連ファイルをチェックして、それらがどのように表示されるかを確認します。Linux では、`mysql-unix` ファイルは次のようになります:

```
##PAM-1.0
auth    include    password-auth
account include    password-auth
```

macOS の場合は、`password-auth` ではなく `login` を使用します。

PAM ファイル形式は、一部のシステムで異なる場合があります。たとえば、Ubuntu およびその他の Debian ベースのシステムでは、かわりに次のファイルコンテンツを使用します:

```
@include common-auth
@include common-account
@include common-session-noninteractive
```

- 外部 PAM ユーザーをプロキシ設定されたアカウントにマップするデフォルトプロキシユーザー ("`@`") を作成します:

```
CREATE USER "@"
IDENTIFIED WITH authentication_pam
AS 'mysql-unix, root=developer, users=data_entry';
```

ここで、認証文字列には PAM サービス名 (`mysql-unix`) が含まれており、Unix パスワードを認証します。また、認証文字列は、`root` および `users` PAM グループ内の外部ユーザーを、それぞれ `developer` および `data_entry` MySQL ユーザー名にマップします。

プロキシユーザーを設定するときは、PAM サービス名のあとに PAM グループマッピングリストが必要です。そうしないと、プラグインは、外部ユーザー名から適切なプロキシ MySQL ユーザー名へのマッピングの実行方法を認識できません。

注記

MySQL インストールに匿名ユーザーが含まれている場合、デフォルトのプロキシユーザーと競合する可能性があります。この問題とその対処方法の詳細は、[デフォルトのプロキシユーザーと匿名ユーザーの競合](#) を参照してください。

- プロキシ設定されたアカウントを作成し、各アカウントに次の権限を付与します:

```
CREATE USER 'developer'@'localhost'
IDENTIFIED WITH mysql_no_login;
CREATE USER 'data_entry'@'localhost'
IDENTIFIED WITH mysql_no_login;

GRANT ALL PRIVILEGES
ON mydevdb.*
TO 'developer'@'localhost';
GRANT ALL PRIVILEGES
ON mydb.*
TO 'data_entry'@'localhost';
```

プロキシ設定されたアカウントは、`mysql_no_login` 認証プラグインを使用して、クライアントがアカウントを使用して MySQL サーバーに直接ログインできないようにします。代わりに、PAM を使用して認証するユーザーは、PAM グループに基づいてプロキシによって `developer` または `data_entry` アカウントを使用することが期待されます。(これは、プラグインがインストールされていることを前提としています。手順については、[セクション 6.4.1.8 「ログインなしのプラグブル認証」](#) を参照してください。) プロキシ設定されたアカウントを直接使用しないように保護する別の方法については、[プロキシアカウントへの直接ログインの防止](#) を参照してください。

- プロキシされた各アカウントの `PROXY` 権限をプロキシアカウントに付与します:

```
GRANT PROXY
ON 'developer'@'localhost'
TO "@";
GRANT PROXY
ON 'data_entry'@'localhost'
TO "@";
```

- `mysql` コマンドラインクライアントを使用して、`antonio` として MySQL サーバーに接続します。

```
shell> mysql --user=antonio --password --enable-cleartext-plugin
Enter password: antonio_password
```

サーバーは、デフォルトの"@"プロキシアカウントを使用して接続を認証します。結果として生成される `antonio` の権限は、`antonio` がメンバーになっている PAM グループによって異なります。`antonio` が `root` PAM グループのメンバーである場合、PAM プラグインは、`developer` MySQL ユーザー名に `root` をマップし、その名前をサーバーに返します。サーバーは、"@が `developer` に対する `PROXY` 権限を持っていることを確認し、接続を許可します。次のクエリーは、次に示すような出力を返します:

```
mysql> SELECT USER(), CURRENT_USER(), @@proxy_user;
+-----+-----+-----+
| USER()      | CURRENT_USER() | @@proxy_user |
+-----+-----+-----+
| antonio@localhost | developer@localhost | "@"          |
+-----+-----+-----+
```

これは、`antonio` オペレーティングシステムユーザーが `developer` MySQL ユーザーに付与された権限を持つように認証され、プロキシがデフォルトのプロキシアカウントを介して行われることを示しています。

`antonio` が `root` PAM グループのメンバーではなく、`users` PAM グループのメンバーである場合、同様のプロセスが発生しますが、プラグインは `user` PAM グループメンバーシップを `data_entry` MySQL ユーザー名にマップし、その名前をサーバーに返します:

```
mysql> SELECT USER(), CURRENT_USER(), @@proxy_user;
+-----+-----+-----+
| USER()      | CURRENT_USER() | @@proxy_user |
+-----+-----+-----+
| antonio@localhost | data_entry@localhost | "@"          |
+-----+-----+-----+
```

これは、`antonio` オペレーティングシステムユーザーが `data_entry` MySQL ユーザーの権限を持つように認証され、そのプロキシがデフォルトのプロキシアカウントを介して行われることを示しています。

注記

クライアント側の `mysql_clear_password` 認証プラグインでは、パスワードはそのまま残されるため、クライアントプログラムはクリアテキストとして MySQL サーバーに送信します。これにより、パスワードをそのまま PAM に渡すことができます。サーバー側 PAM ライブラリを使用するにはクリアテキストのパスワードが必要ですが、一部の構成でセキュリティの問題が発生する可能性があります。これらのメジャーにより、リスクが最小限に抑えられます:

- `mysql_clear_password` プラグインを誤って使用する可能性を低くするには、MySQL クライアントで明示的に有効にする必要があります (たとえば、`--enable-cleartext-plugin` オプションを使用)。セクション 6.4.1.4 「クライアント側クリアテキストプラグイン認証」を参照してください。
- `mysql_clear_password` プラグインを有効にしてパスワードの公開を回避するには、MySQL クライアントは暗号化された接続を使用して MySQL サーバーに接続する必要があります。セクション 6.3.1 「暗号化接続を使用するための MySQL の構成」を参照してください。

Unix パスワードストアへの PAM 認証アクセス

一部のシステムでは、Unix 認証は、通常はアクセス権限が制限されているファイルである `/etc/shadow` などのパスワードストアを使用します。これにより、MySQL PAM ベースの認証が失敗する可能性があります。残念ながら、PAM 実装では、「パスワードが一致しません」との「パスワードを確認できませんでした」の区別 (たとえば、`/etc/shadow` を読み取れないため) は許可されていません。PAM 認証に Unix パスワードストアを使用している場合は、次のいずれかの方法を使用して MySQL からパスワードストアへのアクセスを有効にできます:

- MySQL サーバーが `mysql` オペレーティングシステムアカウントから実行されている場合は、`/etc/shadow` アクセス権を持つ `shadow` グループにそのアカウントを配置します:
 1. `/etc/group` で `shadow` グループを作成します。

2. `mysql` オペレーティングシステムユーザーを `/etc/group` の `shadow` グループに追加します。
3. `/etc/group` を `shadow` グループに割り当て、グループの読取り権限を有効にします:

```
chgrp shadow /etc/shadow
chmod g+r /etc/shadow
```

4. MySQL Server を再起動します。

- `pam_unix` モジュールおよび `unix_chkpwd` ユーティリティを使用している場合は、次のようにパスワードストアへのアクセスを有効にします:

```
chmod u-s /usr/sbin/unix_chkpwd
setcap cap_dac_read_search+ep /usr/sbin/unix_chkpwd
```

プラットフォームに応じて、`unix_chkpwd` へのパスを調整します。

PAM 認証のデバッグ

PAM 認証プラグインは、初期化時に `AUTHENTICATION_PAM_LOG` 環境の値が設定されているかどうかをチェックします (値は問題ありません)。その場合、プラグインを使用すると、標準出力への診断メッセージのロギングが有効になります。これらのメッセージは、プラグインが認証を実行するときに発生する PAM 関連の問題のデバッグに役立つ場合があります。

一部のメッセージには、PAM プラグインソースファイルと行番号への参照が含まれています。これを使用すると、プラグインアクションをそれが発生するコード内の場所に、より緊密に関連付けることができます。

接続障害をデバッグし、接続試行中に何が起きているかを判断する別の手法は、PAM 認証を構成してすべての接続を許可し、システムログファイルを確認することです。この方法は temporary ベースでのみ使用し、本番サーバーでは使用しないでください。

`/etc/pam.d/mysql-any-password` という PAM サービスファイルを次の内容で構成します (一部のシステムでは形式が異なる場合があります):

```
##%PAM-1.0
auth    required pam_permit.so
account required pam_permit.so
```

PAM プラグインを使用するアカウントを作成し、`mysql-any-password` PAM サービスに名前を付けます:

```
CREATE USER 'testuser'@'localhost'
  IDENTIFIED WITH authentication_pam
  AS 'mysql-any-password';
```

`mysql-any-password` サービスファイルを使用すると、不正なパスワードの場合でも、認証試行で `true` が返されます。認証の試行が失敗した場合は、構成の問題が MySQL 側にあることを示します。それ以外の場合は、オペレーティングシステム/PAM 側で問題が発生します。何が起きているかを確認するには、`/var/log/secure`、`/var/log/audit.log`、`/var/log/syslog` や `/var/log/messages` などのシステムログファイルを確認します。

問題点を特定したら、`mysql-any-password` PAM サービスファイルを削除して `any-password` アクセスを無効にします。

6.4.1.6 Windows プラガブル認証

注記

Windows プラガブル認証は、商用製品である MySQL Enterprise Edition に含まれている拡張機能です。商用製品の詳細は、<https://www.mysql.com/products/> を参照してください。

MySQL Enterprise Edition for Windows は、Windows で外部認証を実行する認証方式をサポートしているため、MySQL Server はネイティブ Windows サービスを使用してクライアント接続を認証できます。Windows にログインしたユーザーは、追加のパスワードを指定せずに、自分の環境内の情報に基づいて MySQL クライアントプログラムからサーバーに接続できます。

クライアントとサーバーは、認証ハンドシェイクでデータパケットを交換します。この交換の結果として、サーバーは Windows OS 内のクライアントのアイデンティティーを表すセキュリティコンテキストオブジェクトを作成します。このアイデンティティーには、クライアントアカウントの名前が含まれています。Windows プラガブル認証では、クライアントの ID を使用して、クライアントが特定のアカウントであるかグループのメンバーであることを確認します。デフォルトでは、認証のネゴシエーションに Kerberos が使用されます。Kerberos が使用できない場合は、NTLM が使用されます。

Windows プラガブル認証には、次の機能があります：

- 外部認証: Windows 認証を使用すると、MySQL Server は、Windows にログインした MySQL 付与テーブルの外部で定義されたユーザーからの接続を受け入れることができます。
- プロキシユーザーのサポート: Windows 認証は、クライアントプログラムによって渡された外部ユーザー名とは異なるユーザー名を MySQL に返すことができます。つまり、このプラグインは、外部の Windows で認証されたユーザーが持つべき権限を定義する MySQL ユーザーを返すことができます。たとえば、`joe` という名前の Windows ユーザーは、`developer` という名前の MySQL ユーザーに接続して権限を持つことができます。

次の表には、プラグインおよびライブラリファイルの名前を示します。ファイルは、`plugin_dir` システム変数で指定されたディレクトリに配置する必要があります。

表 6.17 Windows 認証用のプラグインとライブラリの名前

プラグインまたはファイル	プラグインまたはファイル名
サーバー側プラグイン	<code>authentication_windows</code>
クライアント側プラグイン	<code>authentication_windows_client</code>
ライブラリファイル	<code>authentication_windows.dll</code>

ライブラリファイルには、サーバー側のプラグインのみが含まれています。クライアント側のプラグインは、`libmysqlclient` クライアントライブラリに組み込まれています。

サーバー側 Windows 認証プラグインは、MySQL Enterprise Edition にのみ含まれています。MySQL コミュニティー配布には含まれていません。クライアント側のプラグインは、コミュニティ配布を含むすべての配布に含まれています。これにより、任意の配布から、サーバー側のプラグインがロードされたサーバーに接続することがクライアントに許可されます。

Windows 認証プラグインは、MySQL 8.0 でサポートされている任意のバージョンの Windows でサポートされています (<https://www.mysql.com/support/supportedplatforms/database.html>を参照してください)。

次の各セクションでは、Windows プラガブル認証に固有のインストールおよび使用方法について説明します：

- [Windows プラガブル認証のインストール](#)
- [Windows プラガブル認証のアンインストール](#)
- [Windows プラガブル認証の使用](#)

MySQL のプラガブル認証に関する一般的な情報については、[セクション6.2.17「プラガブル認証」](#)を参照してください。プロキシユーザーについては、[セクション6.2.18「プロキシユーザー」](#)を参照してください。

Windows プラガブル認証のインストール

このセクションでは、Windows 認証プラグインをインストールする方法について説明します。プラグインのインストールについての一般的な情報は、[セクション5.6.1「プラグインのインストールおよびアンインストール」](#)を参照してください。

サーバーで使用できるようにするには、プラグインライブラリファイルを MySQL プラグインディレクトリ (`plugin_dir` システム変数で指定されたディレクトリ) に配置する必要があります。必要に応じて、サーバーの起動時に `plugin_dir` の値を設定してプラグインディレクトリの場所を構成します。

サーバーの起動時にプラグインをロードするには、`--plugin-load-add` オプションを使用して、プラグインを含むライブラリファイルに名前を付けます。このプラグインのロード方式では、サーバーを起動するたびにオプションを指定する必要があります。たとえば、サーバー `my.cnf` ファイルに次の行を挿入します：

```
[mysqld]
plugin-load-add=authentication_windows.dll
```

`my.cnf` を変更したら、新しい設定を有効にするためにサーバーを再起動します。

または、実行時にプラグインをロードするには、次のステートメントを使用します:

```
INSTALL PLUGIN authentication_windows SONAME 'authentication_windows.dll';
```

`INSTALL PLUGIN` はプラグインをただちにロードし、`mysql.plugins` システムテーブルにも登録して、`--plugin-load-add` を必要とせずに後続の通常の起動のたびにサーバーがプラグインをロードするようにします。

プラグインのインストールを確認するには、`INFORMATION_SCHEMA.PLUGINS` テーブルを調べるか、`SHOW PLUGINS` ステートメントを使用します (セクション5.6.2「サーバープラグイン情報の取得」を参照)。例:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
FROM INFORMATION_SCHEMA.PLUGINS
WHERE PLUGIN_NAME LIKE '%windows%';
+-----+-----+
| PLUGIN_NAME | PLUGIN_STATUS |
+-----+-----+
| authentication_windows | ACTIVE |
+-----+-----+
```

プラグインの初期化に失敗した場合は、サーバーエラーログで診断メッセージを確認してください。

MySQL アカウントを Windows 認証プラグインに関連付けるには、[Windows プラガブル認証の使用](#) を参照してください。追加のプラグイン制御は、`authentication_windows_use_principal_name` および `authentication_windows_log_level` システム変数によって提供されます。セクション5.1.8「サーバーシステム変数」を参照してください。

Windows プラガブル認証のアンインストール

Windows 認証プラグインのアンインストールに使用する方法は、インストール方法によって異なります:

- `--plugin-load-add` オプションを使用してサーバーの起動時にプラグインをインストールした場合は、オプションなしでサーバーを再起動します。
- `INSTALL PLUGIN` ステートメントを使用して実行時にプラグインをインストールした場合、サーバーの再起動後もインストールされたままになります。アンインストールするには、`UNINSTALL PLUGIN` を使用します:

```
UNINSTALL PLUGIN authentication_windows;
```

また、Windows プラグイン関連のシステム変数を設定する起動オプションも削除します。

Windows プラガブル認証の使用

Windows 認証プラグインでは、Windows にログインしたユーザーが追加のパスワードを指定しなくても、MySQL サーバーに接続できるように、MySQL アカウントの使用がサポートされています。[Windows プラガブル認証のインストール](#) で説明されているように、サーバーがサーバー側プラグインを有効にして実行されていることを前提としています。DBA がサーバー側のプラグインを有効にして、それを使用するようにアカウントを設定すると、クライアントは自分の側でその他の設定を行う必要なしで、これらのアカウントを使用して接続できます。

`CREATE USER` ステートメントの `IDENTIFIED WITH` 句で Windows 認証プラグインを参照するには、`authentication_windows` という名前を使用します。`Rafal` と `Tasha` という Windows ユーザー、および `Administrators` または `Power Users` グループ内の任意のユーザーが MySQL への接続が許可されるべきであると仮定します。このように設定するには、Windows プラグインを使用して認証する `sql_admin` という名前の MySQL アカウントを作成します。

```
CREATE USER sql_admin
IDENTIFIED WITH authentication_windows
AS 'Rafal, Tasha, Administrators, "Power Users";
```

プラグイン名は `authentication_windows` です。AS キーワードに続く文字列は、認証文字列です。`Rafal` または `Tasha` という名前の Windows ユーザー、および `Administrators` または `Power Users` グループ内の任意の Windows

ユーザーが MySQL ユーザー `sql_admin` として、サーバーへの認証が許可されるように指定されます。後者のグループ名には空白文字が含まれているため、二重引用符で囲む必要があります。

`sql_admin` アカウントを作成したあとは、Windows にログインしたユーザーはそのアカウントを使用して、サーバーへの接続を試みることができます。

```
C:\> mysql --user=sql_admin
```

ここでは、パスワードは必要ありません。`authentication_windows` プラグインは Windows のセキュリティ API を使用して、接続中の Windows ユーザーをチェックします。そのユーザーの名前が `Rafal` または `Tasha` であるか、`Administrators` または `Power Users` グループのメンバーである場合、サーバーはアクセス権を付与し、クライアントは `sql_admin` として認証され、`sql_admin` アカウントに付与されている権限を持ちます。それ以外の場合、サーバーはアクセスを拒否します。

Windows 認証プラグインでの認証文字列の構文は、次のようなルールに従っています。

- 文字列は、カンマで区切られた 1 つ以上のユーザーマッピングで構成されます。
- 各ユーザーマッピングによって、Windows ユーザー名またはグループ名が MySQL ユーザー名に関連付けられます。

```
win_user_or_group_name=mysql_user_name
win_user_or_group_name
```

後者の構文では、`mysql_user_name` 値が指定されていない場合、暗黙的な値は `CREATE USER` ステートメントによって作成された MySQL ユーザーです。したがって、次のステートメントは同等です。

```
CREATE USER sql_admin
IDENTIFIED WITH authentication_windows
AS 'Rafal, Tasha, Administrators, "Power Users"';

CREATE USER sql_admin
IDENTIFIED WITH authentication_windows
AS 'Rafal=sql_admin, Tasha=sql_admin, Administrators=sql_admin,
"Power Users"=sql_admin';
```

- バックスラッシュは MySQL 文字列のエスケープ文字であるため、値の各バックスラッシュ文字 (`\`) は二重にする必要があります。
- 二重引用符で囲まれていない先頭および末尾の空白文字は、無視されます。
- 引用符で囲まれていない `win_user_or_group_name` および `mysql_user_name` の値には、等号、カンマまたは空白以外の任意の値を含めることができます。
- `win_user_or_group_name` または `mysql_user_name` (あるいはその両方) の値が二重引用符で囲まっている場合、引用符の間のすべてが値の一部になります。たとえば、名前に空白文字が含まれている場合は、これが必要です。二重引用符およびバックスラッシュを除く、二重引用符内のすべての文字が有効です。どちらかの文字を含めるには、バックスラッシュを使用してエスケープします。
- `win_user_or_group_name` 値では、Windows 主体 (ローカルまたはドメイン内) 用の従来の構文が使用されます。例 (バックスラッシュを二重にすることに注意してください):

```
domain\user
.\user
domain\group
.\group
BUILTIN\WellKnownGroup
```

クライアントを認証するためにサーバーから呼び出されると、プラグインは認証文字列を左から右へとスキャンして、Windows ユーザーとのユーザーまたはグループの一致があるかどうかを確認します。一致する場合、プラグインは対応する `mysql_user_name` を MySQL サーバーに返します。一致がない場合は、認証に失敗します。

ユーザー名の一致は、グループ名の一致よりも優先されます。`win_user` という名前の Windows ユーザーが `win_group` のメンバーであり、認証文字列が次のとおりであると仮定します。

```
'win_group = sql_user1, win_user = sql_user2'
```


`win_user` が MySQL サーバーに接続すると、`win_group` と `win_user` の両方への一致があります。グループが認証文字列の最初に一覧表示されますが、より具体的なユーザーの一致がグループの一致よりも優先されるため、プラグインは `sql_user2` としてユーザーを認証します。

サーバーが実行されているものと同じコンピュータからの接続では、Windows 認証は常に機能します。コンピュータ間の接続では、両方のコンピュータを Windows Active Directory に登録する必要があります。同じ Windows ドメイン内にある場合は、ドメイン名を指定する必要はありません。次の例に示すように、別のドメインからの接続を許可することもできます。

```
CREATE USER sql_accounting
IDENTIFIED WITH authentication_windows
AS 'SomeDomain\Accounting';
```

ここで、`SomeDomain` は別のドメインの名前です。バックスラッシュ文字は文字列内の MySQL エスケープ文字であるため、二重に入力されています。

MySQL では、クライアントは 1 つのアカウントを使用して MySQL サーバーに接続して認証できるが、接続されると別のアカウントの権限を持つというプロキシユーザーの概念がサポートされています ([セクション6.2.18「プロキシユーザー」](#)を参照してください)。次のように、Windows ユーザーは単一のユーザー名を使用して接続するが、Windows ユーザー名およびグループ名に基づいて特定の MySQL アカウント上にマップされると仮定します。

- `local_user` および `MyDomain\domain_user` というローカルおよびドメインの Windows ユーザーは、`local_wlad` MySQL アカウントにマップするべきです。
- `MyDomain\Developers` ドメイングループ内のユーザーは、`local_dev` MySQL アカウントにマップするべきです。
- ローカルマシンの管理者は、`local_admin` MySQL アカウントにマップするべきです。

このように設定するには、接続先の Windows ユーザーのプロキシアカウントを作成し、ユーザーとグループが適切な MySQL アカウント (`local_wlad`、`local_dev`、`local_admin`) にマップされるように、このアカウントを構成します。さらに、実行する必要がある操作に適した権限を MySQL アカウントに付与します。次の手順では、プロキシアカウントとして `win_proxy` が使用され、プロキシ対象アカウントとして `local_wlad`、`local_dev`、および `local_admin` が使用されています。

1. プロキシ MySQL アカウントを作成します。

```
CREATE USER win_proxy
IDENTIFIED WITH authentication_windows
AS 'local_user = local_wlad,
    MyDomain\domain_user = local_wlad,
    MyDomain\Developers = local_dev,
    BUILTIN\Administrators = local_admin';
```

2. プロキシ処理が動作するには、プロキシ対象アカウントが存在する必要があるため、次のように作成します。

```
CREATE USER local_wlad
IDENTIFIED WITH mysql_no_login;
CREATE USER local_dev
IDENTIFIED WITH mysql_no_login;
CREATE USER local_admin
IDENTIFIED WITH mysql_no_login;
```

プロキシ設定されたアカウントは、`mysql_no_login` 認証プラグインを使用して、クライアントがアカウントを使用して MySQL サーバーに直接ログインできないようにします。かわりに、Windows を使用して認証するユーザーは、`win_proxy` プロキシアカウントを使用する必要があります。(これは、プラグインがインストールされていることを前提としています。手順については、[セクション6.4.1.8「ログインなしのプラグイン認証」](#)を参照してください。) プロキシ設定されたアカウントを直接使用しないように保護する別の方法については、[プロキシアカウントへの直接ログインの防止](#)を参照してください。

また、各プロキシアカウントに MySQL アクセスに必要な権限を付与する `GRANT` ステートメント (表示されていません) も実行する必要があります。

3. プロキシされた各アカウントの `PROXY` 権限をプロキシアカウントに付与します:

```
GRANT PROXY ON local_wlad TO win_proxy;
GRANT PROXY ON local_dev TO win_proxy;
```

```
GRANT PROXY ON local_admin TO win_proxy;
```

これで、Windows ユーザー `local_user` および `MyDomain\domain_user` は `win_proxy` として MySQL サーバーに接続でき、認証時に認証文字列で指定されたアカウント (この場合は `local_wlad`) の権限を持つようになります。 `win_proxy` として接続する `MyDomain\Developers` グループ内のユーザーは、`local_dev` アカウントの権限を持っています。 `BUILTIN\Administrators` グループ内のユーザーは、`local_admin` アカウントの権限を持っています。

独自の MySQL アカウントを持たないすべての Windows ユーザーがプロキシアカウントを経由するように認証を構成するには、前述の手順で `win_proxy` のデフォルトのプロキシアカウント ("`@`") に置き換えます。デフォルトのプロキシアカウントの詳細は、[セクション6.2.18「プロキシユーザー」](#) を参照してください。

注記

MySQL インストールに匿名ユーザーが含まれている場合、デフォルトのプロキシユーザーと競合する可能性があります。この問題とその対処方法の詳細は、[デフォルトのプロキシユーザーと匿名ユーザーの競合](#) を参照してください。

Connector/NET 6.4.4 以上の Connector/NET 接続文字列で Windows 認証プラグインを使用するには、[Connector/NET Authentication](#) を参照してください。

6.4.1.7 LDAP プラガブル認証

注記

LDAP プラガブル認証は、商用製品である MySQL Enterprise Edition に含まれる拡張機能です。商用製品の詳細は、<https://www.mysql.com/products/> を参照してください。

MySQL Enterprise Edition は、MySQL Server が LDAP (Lightweight Directory Access Protocol) を使用して X.500 などのディレクトリサービスにアクセスすることで MySQL ユーザーを認証できるようにする認証方式をサポートしています。MySQL は、LDAP を使用してユーザー、資格証明およびグループ情報をフェッチします。

LDAP プラガブル認証は、次の機能を提供します:

- 外部認証: LDAP 認証を使用すると、MySQL Server は LDAP ディレクトリ内の MySQL 付与テーブルの外部で定義されたユーザーからの接続を受け入れることができます。
- プロキシユーザーのサポート: LDAP 認証は、外部ユーザーがメンバーになっている LDAP グループに基づいて、クライアントプログラムによって渡される外部ユーザー名とは異なるユーザー名を MySQL に返すことができます。つまり、LDAP プラグインは、外部 LDAP 認証ユーザーが持つべき権限を定義する MySQL ユーザーを返すことができます。たとえば、`joe` の LDAP グループが `developer` の場合、`joe` という名前の LDAP ユーザーは `developer` という名前の MySQL ユーザーに接続して権限を持つことができます。
- セキュリティ: TLS を使用すると、LDAP サーバーへの接続をセキュリティー保護できます。

次のテーブルに、単純および SASL ベースの LDAP 認証用のプラグインおよびライブラリファイル名を示します。ファイル名のサフィクスは、システムによって異なる場合があります。ファイルは、`plugin_dir` システム変数で指定されたディレクトリに配置する必要があります。

表 6.18 簡易 LDAP 認証のプラグインおよびライブラリ名

プラグインまたはファイル	プラグインまたはファイル名
サーバー側のプラグイン名	<code>authentication_ldap_simple</code>
クライアント側のプラグイン名	<code>mysql_clear_password</code>
ライブラリファイル名	<code>authentication_ldap_simple.so</code>

表 6.19 SASL ベースの LDAP 認証用のプラグインおよびライブラリ名

プラグインまたはファイル	プラグインまたはファイル名
サーバー側のプラグイン名	<code>authentication_ldap_sasl</code>

プラグインまたはファイル	プラグインまたはファイル名
クライアント側のプラグイン名	<code>authentication_ldap_sasl_client</code>
ライブラリファイル名	<code>authentication_ldap_sasl.so</code> , <code>authentication_ldap_sasl_client.so</code>

ライブラリファイルには、`authentication_ldap_XXX` 認証プラグインのみが含まれます。クライアント側の `mysql_clear_password` プラグインは、`libmysqlclient` クライアントライブラリに組み込まれています。

各サーバー側 LDAP プラグインは、特定のクライアント側プラグインで動作します:

- サーバー側 `authentication_ldap_simple` プラグインは、単純な LDAP 認証を実行します。このプラグインを使用するアカウントによる接続の場合、クライアントプログラムはクライアント側の `mysql_clear_password` プラグインを使用します。このプラグインは、パスワードをクリアテキストとしてサーバーに送信します。パスワードのハッシュ化または暗号化は使用されないため、パスワードの公開を防ぐために、MySQL クライアントとサーバー間のセキュアな接続をお勧めします。
- サーバー側の `authentication_ldap_sasl` プラグインは、SASL ベースの LDAP 認証を実行します。このプラグインを使用するアカウントによる接続の場合、クライアントプログラムはクライアント側の `authentication_ldap_sasl_client` プラグインを使用します。クライアント側およびサーバー側 SASL LDAP プラグインは、SASL メッセージを使用して LDAP プロトコル内での資格証明のセキュアな転送を行い、MySQL クライアントとサーバー間でクリアテキストパスワードが送信されないようにします。

次の各セクションでは、LDAP プラガブル認証に固有のインストールおよび使用方法について説明します:

- LDAP プラガブル認証の前提条件
- MySQL ユーザーの LDAP 認証の仕組み
- LDAP プラガブル認証のインストール
- LDAP プラガブル認証のアンインストール
- LDAP プラガブル認証および `ldap.conf`
- LDAP プラガブル認証の使用
- 簡易 LDAP 認証
- SASL ベースの LDAP 認証
- プロキシを使用した LDAP 認証
- LDAP 認証グループプリファレンスとマッピングの指定
- LDAP 認証ユーザー DN 接尾辞
- LDAP 認証方式
- GSSAPI/Kerberos 認証方式
- LDAP 検索照会

MySQL のプラガブル認証に関する一般的な情報については、[セクション6.2.17「プラガブル認証」](#)を参照してください。`mysql_clear_password` プラグインの詳細は、[セクション6.4.1.4「クライアント側クリアテキストプラガブル認証」](#)を参照してください。プロキシユーザーについては、[セクション6.2.18「プロキシユーザー」](#)を参照してください。

注記

システムが PAM をサポートし、PAM 認証方式として LDAP を許可する場合、MySQL ユーザー認証に LDAP を使用する別の方法は、サーバー側の `authentication_pam` プラグインを使用することです。[セクション6.4.1.5「PAM プラガブル認証」](#)を参照してください。

LDAP プラガブル認証の前提条件

MySQL で LDAP プラガブル認証を使用するには、次の前提条件を満たす必要があります：

- LDAP 認証プラグインが通信するには、LDAP サーバーが使用可能である必要があります。
- MySQL によって認証される LDAP ユーザーは、LDAP サーバーによって管理されるディレクトリに存在する必要があります。
- LDAP クライアントライブラリは、サーバー側の [authentication_ldap_sasl](#) または [authentication_ldap_simple](#) プラグインが使用されているシステムで使用可能である必要があります。現在サポートされているライブラリは、Windows ネイティブ LDAP ライブラリ、または Windows 以外のシステムの OpenLDAP ライブラリです。
- SASL ベースの LDAP 認証を使用するには：
 - LDAP サーバーは SASL サーバーと通信するように構成する必要があります。
 - SASL クライアントライブラリは、クライアント側の [authentication_ldap_sasl_client](#) プラグインが使用されているシステムで使用可能である必要があります。現在サポートされているライブラリは Cyrus SASL ライブラリのみです。
 - 特定の SASL 認証方式を使用するには、その方式に必要なその他のサービスが使用可能である必要があります。たとえば、GSSAPI/Kerberos を使用するには、GSSAPI ライブラリおよび Kerberos サービスが使用可能である必要があります。

MySQL ユーザーの LDAP 認証の仕組み

このセクションでは、MySQL と LDAP が連携して MySQL ユーザーを認証する方法の概要を示します。特定の LDAP 認証プラグインを使用するように MySQL アカウントを設定する方法を示す例は、[LDAP プラガブル認証の使用](#) を参照してください。LDAP プラグインで使用可能な認証方式については、[LDAP 認証方式](#) を参照してください。

クライアントは MySQL サーバーに接続し、MySQL クライアントユーザー名とパスワードを指定します：

- 単純な LDAP 認証の場合、クライアント側およびサーバー側のプラグインはパスワードをクリアテキストとして通信します。パスワードの公開を防ぐために、MySQL クライアントとサーバー間のセキュアな接続をお勧めします。
- SASL ベースの LDAP 認証の場合、クライアント側とサーバー側のプラグインは、MySQL クライアントとサーバー間でクリアテキストのパスワードを送信しないようにします。たとえば、プラグインは SASL メッセージを使用して、LDAP プロトコル内での資格証明のセキュアな転送を行うことができます。GSSAPI 認証方式の場合、クライアント側およびサーバー側のプラグインは、LDAP メッセージを直接使用せずに Kerberos を使用してセキュアに通信します。

クライアントユーザー名とホスト名が MySQL アカウントと一致しない場合、接続は拒否されます。

一致する MySQL アカウントがある場合は、LDAP に対する認証が行われます。LDAP サーバーは、ユーザーと一致するエントリを検索し、LDAP パスワードに対してエントリを認証します：

- MySQL アカウントが LDAP ユーザー識別名 (DN) を指定している場合、LDAP 認証はその値とクライアントによって提供された LDAP パスワードを使用します。(LDAP ユーザー DN を MySQL アカウントに関連付けるには、アカウントを作成する [CREATE USER](#) ステートメントに認証文字列を指定する [BY](#) 句を含めます。)
- MySQL アカウントに LDAP ユーザー DN が指定されていない場合、LDAP 認証ではクライアントから提供されたユーザー名と LDAP パスワードが使用されます。この場合、認証プラグインはまずルート DN とパスワードを資格証明として使用して LDAP サーバーにバインドし、クライアントユーザー名に基づいてユーザー DN を検索してから、LDAP パスワードに対してそのユーザー DN を認証します。ルート DN およびパスワードが正しくない値に設定されているか、空 (設定されていない) で LDAP サーバーが匿名接続を許可していない場合、ルート資格証明を使用したこのバインドは失敗します。

LDAP サーバーが一致または複数の一致を検出しなかった場合、認証は失敗し、クライアント接続は拒否されます。

LDAP サーバーが単一の一致を検出した場合、LDAP 認証は成功し (パスワードが正しいと想定)、LDAP サーバーは LDAP エントリを返し、認証プラグインはそのエントリに基づいて認証されたユーザーの名前を決定します：

- LDAP エントリにグループ属性 (デフォルトでは `cn` 属性) がある場合、プラグインはその値を認証されたユーザー名として返します。
- LDAP エントリに `group` 属性がない場合、認証プラグインは、認証されたユーザー名としてクライアントユーザー名を返します。

MySQL サーバーは、クライアントユーザー名と認証済ユーザー名を比較して、クライアントセッションでプロキシが発生するかどうかを判断します:

- 名前が同じ場合、プロキシは発生しません: クライアントユーザー名と一致する MySQL アカウントが権限チェックに使用されます。
- 名前が異なる場合、プロキシが発生: MySQL は、認証されたユーザー名と一致するアカウントを検索します。そのアカウントはプロキシユーザーになり、権限チェックに使用されます。クライアントユーザー名に一致する MySQL アカウントは、外部プロキシユーザーとして扱われます。

LDAP プラガブル認証のインストール

このセクションでは、LDAP 認証プラグインをインストールする方法について説明します。プラグインのインストールについての一般的な情報は、[セクション5.6.1「プラグインのインストールおよびアンインストール」](#)を参照してください。

サーバーで使用できるようにするには、プラグインライブラリファイルを MySQL プラグインディレクトリ (`plugin_dir` システム変数で指定されたディレクトリ) に配置する必要があります。必要に応じて、サーバーの起動時に `plugin_dir` の値を設定してプラグインディレクトリの場所を構成します。

サーバー側のプラグインライブラリファイルのベース名は、[authentication_ldap_simple](#) および [authentication_ldap_sasl](#) です。ファイル名の接尾辞は、プラットフォームごとに異なります (たとえば、`.so` for Unix and Unix-like systems, `.dll` for Windows)。

サーバーの起動時にプラグインをロードするには、`--plugin-load-add` オプションを使用して、プラグインを含むライブラリファイルに名前を付けます。このプラグインのロード方法では、サーバーが起動するたびにオプションを指定する必要があります。また、構成するプラグイン提供のシステム変数の値を指定します。

各サーバー側 LDAP プラグインは、その操作の構成を可能にする一連のシステム変数を公開します。これらのほとんどはオプションですが、LDAP バインド操作の LDAP サーバーホストを指定する変数 (プラグインが接続先を認識できるようにするため)、およびベース識別名を設定する必要があります (検索の範囲を制限してより高速な検索を取得するため)。すべての LDAP システム変数の詳細は、[セクション6.4.1.11「プラガブル認証システム変数」](#)を参照してください。

プラグインをロードし、LDAP バインド操作の LDAP サーバーホストおよびベース識別名を設定するには、`my.cnf` ファイルに次のような行を入力し、必要に応じてプラットフォームの `.so` 接尾辞を調整します:

```
[mysqld]
plugin-load-add=authentication_ldap_simple.so
authentication_ldap_simple_server_host=127.0.0.1
authentication_ldap_simple_bind_base_dn="dc=example,dc=com"
plugin-load-add=authentication_ldap_sasl.so
authentication_ldap_sasl_server_host=127.0.0.1
authentication_ldap_sasl_bind_base_dn="dc=example,dc=com"
```

`my.cnf` を変更したら、新しい設定を有効にするためにサーバーを再起動します。

または、実行時にプラグインをロードするには、次のステートメントを使用して、プラットフォームの `.so` 接尾辞を必要に応じて調整します:

```
INSTALL PLUGIN authentication_ldap_simple
SONAME 'authentication_ldap_simple.so';
INSTALL PLUGIN authentication_ldap_sasl
SONAME 'authentication_ldap_sasl.so';
```

`INSTALL PLUGIN` はプラグインをただちにロードし、`mysql.plugins` システムテーブルにも登録して、`--plugin-load-add` を必要とせずに後続の通常の起動のたびにサーバーがプラグインをロードするようにします。

実行時にプラグインをインストールすると、そのシステム変数が使用可能になり、それらの設定を `my.cnf` ファイルに追加して、その後の再起動のためにプラグインを構成できます。例:


```
[mysqld]
authentication_ldap_simple_server_host=127.0.0.1
authentication_ldap_simple_bind_base_dn="dc=example,dc=com"
authentication_ldap_sasl_server_host=127.0.0.1
authentication_ldap_sasl_bind_base_dn="dc=example,dc=com"
```

`my.cnf` を変更したら、新しい設定を有効にするためにサーバーを再起動します。

または、実行時に値を設定して永続化するには、次のステートメントを使用します:

```
SET PERSIST authentication_ldap_simple_server_host='127.0.0.1';
SET PERSIST authentication_ldap_simple_bind_base_dn='dc=example,dc=com';
SET PERSIST authentication_ldap_sasl_server_host='127.0.0.1';
SET PERSIST authentication_ldap_sasl_bind_base_dn='dc=example,dc=com';
```

`SET PERSIST` は、実行中の MySQL インスタンスの値を設定します。また、値が保存され、その後のサーバーの再起動に引き継がれます。後続の再起動に引き継ぐことなく、実行中の MySQL インスタンスの値を変更するには、`PERSIST` ではなく `GLOBAL` キーワードを使用します。セクション13.7.6.1「変数代入の SET 構文」を参照してください。

プラグインのインストールを確認するには、`INFORMATION_SCHEMA.PLUGINS` テーブルを調べるか、`SHOW PLUGINS` ステートメントを使用します (セクション5.6.2「サーバープラグイン情報の取得」を参照)。例:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
FROM INFORMATION_SCHEMA.PLUGINS
WHERE PLUGIN_NAME LIKE '%ldap%';
+-----+-----+
| PLUGIN_NAME | PLUGIN_STATUS |
+-----+-----+
| authentication_ldap_sasl | ACTIVE |
| authentication_ldap_simple | ACTIVE |
+-----+-----+
```

プラグインの初期化に失敗した場合は、サーバーエラーログで診断メッセージを確認してください。

MySQL アカウントを LDAP プラグインに関連付けるには、[LDAP プラガブル認証の使用](#) を参照してください。

SELinux の追加ノート

SELinux が有効になっている EL6 または EL を実行しているシステムでは、MySQL LDAP プラグインが LDAP サービスと通信できるように SELinux ポリシーを変更する必要があります:

1. 次の内容のファイル `mysqldap.te` を作成します:

```
module mysqldap 1.0;

require {
    type ldap_port_t;
    type mysqld_t;
    class tcp_socket name_connect;
}

#===== mysqld_t =====
allow mysqld_t ldap_port_t:tcp_socket name_connect;
```

2. セキュリティポリシーモジュールをバイナリ表現にコンパイルします:

```
checkmodule -M -m mysqldap.te -o mysqldap.mod
```

3. SELinux ポリシーモジュールパッケージを作成します:

```
semodule_package -m mysqldap.mod -o mysqldap.pp
```

4. モジュールパッケージをインストールします:

```
semodule -i mysqldap.pp
```

5. SELinux ポリシーが変更されたら、MySQL サーバーを再起動します:


```
service mysqld restart
```

LDAP プラガブル認証のアンインストール

LDAP 認証プラグインのアンインストールに使用する方法は、インストール方法によって異なります:

- `--plugin-load-add` オプションを使用してサーバーの起動時にプラグインをインストールした場合は、それらのオプションを指定せずにサーバーを再起動します。
- `INSTALL PLUGIN` を使用して実行時にプラグインをインストールした場合、それらはサーバーの再起動後もインストールされたままです。これらをアンインストールするには、`UNINSTALL PLUGIN` を使用します:

```
UNINSTALL PLUGIN authentication_ldap_simple;  
UNINSTALL PLUGIN authentication_ldap_sasl;
```

また、LDAP プラグイン関連のシステム変数を設定する起動オプションを `my.cnf` ファイルから削除します。 `SET PERSIST` を使用して LDAP システム変数を永続化した場合は、`RESET PERSIST` を使用して設定を削除します。

LDAP プラガブル認証および `ldap.conf`

OpenLDAP を使用するインストールの場合、`ldap.conf` ファイルは LDAP クライアントのグローバルデフォルトを提供します。このファイルでオプションを設定して、LDAP 認証プラグインを含む LDAP クライアントに影響を与えることができます。OpenLDAP では、次の優先順位で構成オプションが使用されます:

- LDAP クライアントによって指定された構成。
- `ldap.conf` ファイルで指定された構成。このファイルの使用を無効にするには、`LDAPNOINIT` 環境変数を設定します。
- OpenLDAP ライブラリの組み込みデフォルト。

ライブラリのデフォルト値または `ldap.conf` 値で適切なオプション値が得られない場合は、LDAP 認証プラグインで関連する変数を設定して LDAP 構成に直接影響を与えることができます。たとえば、LDAP プラグインは、次のようなパラメータの `ldap.conf` をオーバーライドできます:

- TLS 構成: TLS を有効にして CA 構成を制御するためにシステム変数を使用できます。単純な LDAP 認証の場合は `authentication_ldap_simple_tls` および `authentication_ldap_simple_ca_path` など、および、SASL LDAP 認証の場合は `authentication_ldap_sasl_tls` および `authentication_ldap_sasl_ca_path` などです。
- LDAP 参照。 [LDAP 検索照会](#) を参照してください。

`ldap.conf` の詳細は、`ldap.conf(5)` のマニュアルページを参照してください。

LDAP プラガブル認証の使用

このセクションでは、MySQL アカウントが LDAP プラガブル認証を使用して MySQL サーバーに接続できるようにする方法について説明します。 [LDAP プラガブル認証のインストール](#) で説明されているように、サーバーが適切なサーバー側プラグインを有効にして実行されており、適切なクライアント側プラグインがクライアントホストで使用可能であることを前提としています。

このセクションでは、LDAP の構成または管理については説明しません。これらのトピックを理解していることを前提としています。

2 つのサーバー側 LDAP プラグインは、それぞれ特定のクライアント側プラグインで動作します:

- サーバー側 `authentication_ldap_simple` プラグインは、単純な LDAP 認証を実行します。このプラグインを使用するアカウントによる接続の場合、クライアントプログラムはクライアント側の `mysql_clear_password` プラグインを使用します。このプラグインは、パスワードをクリアテキストとしてサーバーに送信します。パスワードのハッシュ化または暗号化は使用されないため、パスワードの公開を防ぐために、MySQL クライアントとサーバー間のセキュアな接続をお勧めします。
- サーバー側の `authentication_ldap_sasl` プラグインは、SASL ベースの LDAP 認証を実行します。このプラグインを使用するアカウントによる接続の場合、クライアントプログラムはクライアント側の

`authentication_ldap_sasl_client` プラグインを使用します。クライアント側およびサーバー側 SASL LDAP プラグインは、SASL メッセージを使用して LDAP プロトコル内での資格証明のセキュアな転送を行い、MySQL クライアントとサーバー間でクリアテキストパスワードが送信されないようにします。

MySQL ユーザーの LDAP 認証の全体的な要件:

- 認証されるユーザーごとに LDAP ディレクトリエントリが必要です。
- サーバー側の LDAP 認証プラグインを指定し、オプションで関連する LDAP ユーザー識別名 (DN) を指定する MySQL ユーザーアカウントが必要です。(LDAP ユーザー DN を MySQL アカウントに関連付けるには、アカウントを作成する `CREATE USER` ステートメントに `BY` 句を含めます。) アカウント名に LDAP 文字列がない場合、LDAP 認証はクライアントによって指定されたユーザー名を使用して LDAP エントリを検索します。
- クライアントプログラムは、MySQL アカウントが使用するサーバー側認証プラグインに適した接続方法を使用して接続します。LDAP 認証の場合、接続には MySQL ユーザー名と LDAP パスワードが必要です。また、サーバー側の `authentication_ldap_simple` プラグインを使用するアカウントの場合は、`--enable-clear-text-plugin` オプションを指定してクライアントプログラムを呼び出し、クライアント側の `mysql_clear_password` プラグインを有効にします。

ここでの手順は、次のシナリオを前提としています:

- MySQL ユーザー `betsy` および `boris` は、それぞれ `betsy_ldap` および `boris_ldap` の LDAP エントリに対して認証を行います。(MySQL と LDAP のユーザー名が異なる必要はありません。この説明で異なる名前を使用すると、操作コンテキストが MySQL か LDAP かを明確にするのに役立ちます。)
- LDAP エントリは、`uid` 属性を使用してユーザー名を指定します。これは LDAP サーバーによって異なる場合があります。一部の LDAP サーバーでは、`uid` ではなく `cn` 属性をユーザー名に使用します。属性を変更するには、`authentication_ldap_simple_user_search_attr` または `authentication_ldap_sasl_user_search_attr` システム変数を適切に変更します。
- これらの LDAP エントリは、各ユーザーを一意に識別する識別名値を提供するために、LDAP サーバーによって管理されるディレクトリで使用できます:

```
uid=betsy_ldap,ou=People,dc=example,dc=com
uid=boris_ldap,ou=People,dc=example,dc=com
```

- MySQL アカウントを作成する `CREATE USER` ステートメントは、MySQL アカウントの認証対象となる LDAP エントリを示すために、`BY` 句で LDAP ユーザーを指定します。

LDAP 認証を使用するアカウントを設定する手順は、使用するサーバー側 LDAP プラグインによって異なります。次の各セクションでは、いくつかの使用シナリオについて説明します。

簡易 LDAP 認証

単純な LDAP 認証用に MySQL アカウントを構成するには、`CREATE USER` ステートメントで `authentication_ldap_simple` プラグインを指定し、オプションで LDAP ユーザー識別名 (DN) を指定します:

```
CREATE USER user
IDENTIFIED WITH authentication_ldap_simple
[BY 'LDAP user DN'];
```

MySQL ユーザー `betsy` の LDAP ディレクトリに次のエントリがあるとします:

```
uid=betsy_ldap,ou=People,dc=example,dc=com
```

その後、`betsy` の MySQL アカウントを作成するステートメントは次のようになります:

```
CREATE USER 'betsy'@'localhost'
IDENTIFIED WITH authentication_ldap_simple
AS 'uid=betsy_ldap,ou=People,dc=example,dc=com';
```

`BY` 句で指定された認証文字列に LDAP パスワードが含まれていません。これは、接続時にクライアントユーザーが指定する必要があります。

クライアントは、MySQL ユーザー名と LDAP パスワードを指定し、クライアント側の `mysql_clear_password` プラグインを有効にすることによって、MySQL サーバーに接続します:

```
shell> mysql --user=betsy --password --enable-cleartext-plugin
Enter password: betsy_password (betsy_ldap LDAP password)
```

注記

クライアント側の `mysql_clear_password` 認証プラグインでは、パスワードはそのまま残されるため、クライアントプログラムはクリアテキストとして MySQL サーバーに送信します。これにより、パスワードをそのまま LDAP サーバーに渡すことができます。SASL なしでサーバー側 LDAP ライブラリを使用するにはクリアテキストパスワードが必要ですが、一部の構成でセキュリティの問題が発生する可能性があります。これらのメジャーにより、リスクが最小限に抑えられます:

- `mysql_clear_password` プラグインを誤って使用する可能性を低くするには、MySQL クライアントで明示的に有効にする必要があります (たとえば、`--enable-cleartext-plugin` オプションを使用)。セクション6.4.1.4「クライアント側クリアテキストプラグイン認証」を参照してください。
- `mysql_clear_password` プラグインを有効にしてパスワードの公開を回避するには、MySQL クライアントは暗号化された接続を使用して MySQL サーバーに接続する必要があります。セクション6.3.1「暗号化接続を使用するための MySQL の構成」を参照してください。

認証プロセスは次のように実行されます:

1. クライアント側プラグインは、`betsy` および `betsy_password` をクライアントユーザー名および LDAP パスワードとして MySQL サーバーに送信します。
2. 接続試行は'`betsy`'@'`localhost`'アカウントと一致します。サーバー側 LDAP プラグインは、このアカウントに、LDAP ユーザー DN を指定するための'`uid=betsy_ldap,ou=People,dc=example,dc=com`'の認証文字列があることを検出します。プラグインは、この文字列と LDAP パスワードを LDAP サーバーに送信します。
3. LDAP サーバーは `betsy_ldap` の LDAP エントリを検出し、パスワードが一致するため、LDAP 認証は成功します。
4. LDAP エントリにはグループ属性がないため、サーバー側プラグインは認証されたユーザーとしてクライアントユーザー名 (`betsy`) を返します。これはクライアントによって提供されるユーザー名と同じであるため、プロキシは発生せず、クライアントセッションは権限チェックに'`betsy`'@'`localhost`'アカウントを使用します。

一致する LDAP エントリにグループ属性が含まれている場合、その属性値は認証されたユーザー名になり、値が `betsy` と異なる場合はプロキシが発生します。group 属性の使用例については、[プロキシを使用した LDAP 認証](#) を参照してください。

`betsy_ldap` LDAP 識別名を指定する `BY` 句が `CREATE USER` ステートメントに含まれていない場合、認証の試行ではクライアントによって提供されたユーザー名 (この場合は `betsy`) が使用されます。`betsy` の LDAP エントリがない場合、認証は失敗します。

SASL ベースの LDAP 認証

SASL LDAP 認証用に MySQL アカウントを構成するには、`CREATE USER` ステートメントで `authentication_ldap_sasl` プラグインを指定し、オプションで LDAP ユーザー識別名 (DN) を指定します:

```
CREATE USER user
  IDENTIFIED WITH authentication_ldap_sasl
  [BY 'LDAP user DN'];
```

MySQL ユーザー `boris` の LDAP ディレクトリに次のエントリがあるとします:

```
uid=boris_ldap,ou=People,dc=example,dc=com
```

その後、`boris` の MySQL アカウントを作成するステートメントは次のようになります:

```
CREATE USER 'boris'@'localhost'
  IDENTIFIED WITH authentication_ldap_sasl
  AS 'uid=boris_ldap,ou=People,dc=example,dc=com';
```

BY 句で指定された認証文字列に LDAP パスワードが含まれていません。これは、接続時にクライアントユーザーが指定する必要があります。

クライアントは、MySQL ユーザー名と LDAP パスワードを指定して MySQL サーバーに接続します:

```
shell> mysql --user=boris --password
Enter password: boris_password (boris_ldap LDAP password)
```

サーバー側 `authentication_ldap_sasl` プラグインの場合、クライアントはクライアント側 `authentication_ldap_sasl_client` プラグインを使用します。クライアントプログラムがクライアント側プラグインを見つけられない場合は、プラグインライブラリファイルがインストールされているディレクトリの名前を指定する `--plugin-dir` オプションを指定します。

`boris` の認証プロセスは、クライアント側とサーバー側の SASL LDAP プラグインが SASL メッセージを使用して LDAP プロトコル内の資格証明をセキュアに送信し、MySQL クライアントとサーバー間のクリアテキストパスワードの送信を回避する点を除き、単純な LDAP 認証を使用した `betsy` の場合と同様です。

プロキシを使用した LDAP 認証

LDAP 認証プラグインはプロキシをサポートしているため、ユーザーはあるユーザーとして MySQL サーバーに接続できますが、別のユーザーの権限を引き受けます。このセクションでは、LDAP プラグインの基本的なプロキシサポートについて説明します。LDAP プラグインは、グループプリファレンスおよびプロキシユーザーマッピングの指定もサポートしています。[LDAP 認証グループプリファレンスとマッピングの指定](#) を参照してください。

ここで説明するプロキシ実装は、LDAP を使用して認証する MySQL ユーザーを、様々な権限セットを定義する他の MySQL アカウントにマップするための LDAP グループ属性値の使用に基づいています。ユーザーは、権限を定義するアカウントを使用して直接接続しません。かわりに、すべての外部ロギンが権限を保持するプロキシ MySQL アカウントにマップされるように、LDAP で認証されたデフォルトのプロキシアカウントを介して接続します。プロキシアカウントを使用して接続するユーザーは、プロキシ設定された MySQL アカウントのいずれか (外部ユーザーに許可されるデータベース操作を決定する権限) にマップされます。

ここでの手順は、次のシナリオを前提としています:

- LDAP エントリは、`uid` 属性と `cn` 属性を使用して、それぞれユーザー名とグループ値を指定します。異なるユーザーおよびグループ属性名を使用するには、適切なプラグイン固有のシステム変数を設定します:
- `authentication_ldap_simple` プラグインの場合: `authentication_ldap_simple_user_search_attr` および `authentication_ldap_simple_group_search_attr` を設定します。
- `authentication_ldap_sasl` プラグインの場合: `authentication_ldap_sasl_user_search_attr` および `authentication_ldap_sasl_group_search_attr` を設定します。
- これらの LDAP エントリは、各ユーザーを一意に識別する識別名値を提供するために、LDAP サーバーによって管理されるディレクトリで使用できます:

```
uid=basha,ou=People,dc=example,dc=com,cn=accounting
uid=basil,ou=People,dc=example,dc=com,cn=front_office
```

接続時に、グループ属性値は認証されたユーザー名になるため、`accounting` および `front_office` プロキシアカウントに名前が付けられます。

- この例では、SASL LDAP 認証を使用することを前提としています。単純な LDAP 認証に対して適切な調整を行います。

デフォルトのプロキシ MySQL アカウントを作成します:

```
CREATE USER '@'@%'
IDENTIFIED WITH authentication_ldap_sasl;
```

プロキシアカウント定義には、LDAP ユーザー DN を指定する `AS 'auth_string'` 句がありません。したがって、次のようになります:

- クライアントが接続すると、クライアントユーザー名が検索対象の LDAP ユーザー名になります。
- 一致する LDAP エントリには、クライアントが保持する必要がある権限を定義するプロキシ設定された MySQL アカウントを指定するグループ属性が含まれている必要があります。

注記

MySQL インストールに匿名ユーザーが含まれている場合、デフォルトのプロキシユーザーと競合する可能性があります。この問題とその対処方法の詳細は、[デフォルトのプロキシユーザーと匿名ユーザーの競合](#) を参照してください。

プロキシ設定されたアカウントを作成し、各アカウントに次の権限を付与します:

```
CREATE USER 'accounting'@'localhost'
  IDENTIFIED WITH mysql_no_login;
CREATE USER 'front_office'@'localhost'
  IDENTIFIED WITH mysql_no_login;

GRANT ALL PRIVILEGES
  ON accountingdb.*
  TO 'accounting'@'localhost';
GRANT ALL PRIVILEGES
  ON frontdb.*
  TO 'front_office'@'localhost';
```

プロキシ設定されたアカウントは、[mysql_no_login](#) 認証プラグインを使用して、クライアントがアカウントを使用して MySQL サーバーに直接ログインできないようにします。かわりに、LDAP を使用して認証するユーザーは、デフォルトの"@%"プロキシアカウントを使用する必要があります。(これは、[mysql_no_login](#) プラグインがインストールされていることを前提としています。手順については、[セクション6.4.1.8「ログインなしのプラグイン認証」](#) を参照してください。)プロキシ設定されたアカウントを直接使用しないように保護する別の方法については、[プロキシアカウントへの直接ログインの防止](#) を参照してください。

プロキシされた各アカウントの [PROXY](#) 権限をプロキシアカウントに付与します:

```
GRANT PROXY
  ON 'accounting'@'localhost'
  TO "@%";
GRANT PROXY
  ON 'front_office'@'localhost'
  TO "@%";
```

[mysql](#) コマンドラインクライアントを使用して、[basha](#) として MySQL サーバーに接続します。

```
shell> mysql --user=basha --password
Enter password: basha_password (basha LDAP password)
```

認証は次のように行われます:

1. サーバーは、クライアントユーザー [basha](#) のデフォルトの"@%"プロキシアカウントを使用して接続を認証します。
2. 一致する LDAP エントリは次のとおりです:

```
uid=basha,ou=People,dc=example,dc=com,cn=accounting
```
3. 一致する LDAP エントリにはグループ属性 [cn=accounting](#) があるため、[accounting](#) が認証済プロキシユーザーになります。
4. 認証されたユーザーはクライアントユーザー名 [basha](#) とは異なりますが、その結果、[basha](#) は [accounting](#) のプロキシとして扱われ、[basha](#) はプロキシされた [accounting](#) アカウントの権限を引き継ぎます。次のクエリーは、次に示すような出力を返します:

```
mysql> SELECT USER(), CURRENT_USER(), @@proxy_user;
+-----+-----+-----+
| USER() | CURRENT_USER() | @@proxy_user |
+-----+-----+-----+
| basha@localhost | accounting@localhost | "@%" |
+-----+-----+-----+
```

これは、[basha](#) がプロキシ設定された [accounting](#) MySQL アカウントに付与された権限を使用し、プロキシ設定がデフォルトのプロキシユーザーアカウントを介して行われることを示しています。

かわりに、[basil](#) として接続します:


```
shell> mysql --user=basil --password
Enter password: basil_password (basil LDAP password)
```

basil の認証プロセスは、前述の basha の認証プロセスと似ています：

1. サーバーは、クライアントユーザー basil のデフォルトの"@'%プロキシアカウントを使用して接続を認証します。
2. 一致する LDAP エントリは次のとおりです：

```
uid=basil,ou=People,dc=example,dc=com,cn=front_office
```

3. 一致する LDAP エントリにはグループ属性 cn=front_office があるため、front_office が認証済プロキシユーザーになります。
4. 認証されたユーザーはクライアントユーザー名 basil とは異なりますが、その結果、basil は front_office のプロキシとして扱われ、basil はプロキシされた front_office アカウントの権限を引き継ぎます。次のクエリーは、次に示すような出力を返します：

```
mysql> SELECT USER(), CURRENT_USER(), @@proxy_user;
+-----+-----+-----+
| USER()      | CURRENT_USER()  | @@proxy_user |
+-----+-----+-----+
| basil@localhost | front_office@localhost | "@'%      |
+-----+-----+-----+
```

これは、basil がプロキシ設定された front_office MySQL アカウントに付与された権限を使用し、プロキシ設定がデフォルトのプロキシユーザーアカウントを介して行われることを示しています。

LDAP 認証グループプリファレンスとマッピングの指定

プロキシを使用した LDAP 認証で説明されているように、基本的な LDAP 認証プロキシは、LDAP サーバーから返された最初のグループ名を MySQL プロキシユーザーアカウント名としてプラグインが使用する原則によって機能します。この単純な機能では、LDAP サーバーが複数のグループ名を返す場合、またはプロキシユーザー名としてグループ名以外の名前を指定する場合、使用するグループ名に関するプリファレンスを指定できません。

MySQL 8.0.14 では、LDAP 認証を使用する MySQL アカウントの場合、認証文字列で次の情報を指定して、プロキシの柔軟性を高めることができます：

- プラグインが LDAP サーバーから返されたグループと一致するリスト内の最初のグループ名を使用するように、優先順位の高いグループのリスト。
- グループ名からプロキシ設定されたユーザー名へのマッピング。これにより、一致したグループ名は、プロキシ設定されたユーザーとして使用するために指定された名前を提供できます。これにより、プロキシユーザーとしてグループ名を使用するかわりに使用できます。

次の MySQL プロキシアカウント定義について考えてみます：

```
CREATE USER "@'%
IDENTIFIED WITH authentication_ldap_sasl
AS '+ou=People,dc=example,dc=com#grp1=usera,grp2,grp3=userc';
```

認証文字列には、+ 文字が接頭辞として付いたユーザー DN 接尾辞 ou=People,dc=example,dc=com があります。したがって、LDAP 認証ユーザー DN 接尾辞で説明されているように、完全なユーザー DN は、指定されたユーザー DN 接尾辞と、uid 属性としてのクライアントユーザー名から構成されます。

認証文字列の残りの部分は、グループプリファレンスおよびマッピング情報の開始を示す#で始まります。認証文字列のこの部分には、グループ名が grp1, grp2, grp3 の順序でリストされます。LDAP プラグインは、そのリストを LDAP サーバーから返されたグループ名のセットと比較し、リストの順序で返された名前との一致を探します。プラグインは最初の一致を使用します。一致するものがない場合、認証は失敗します。

LDAP サーバーがグループ grp3, grp2 および grp7 を返すとします。LDAP プラグインは、LDAP サーバーによって返される最初のグループではなくても、一致する認証文字列の最初のグループであるため、grp2 を使用します。LDAP サーバーが grp4, grp2 および grp1 を返す場合、grp2 も一致していても、プラグインは grp1 を使用します。grp1 は、認証文字列の前半にリストされているため、grp2 よりも優先されます。

プラグインは、一致するグループ名を検出すると、そのグループ名から MySQL プロキシユーザー名へのマッピングを実行します (存在する場合)。プロキシアカウントの例では、マッピングは次のように行われます:

- 一致するグループ名が `grp1` または `grp3` の場合、それらは認証文字列内でユーザー名 `usera` および `userc` にそれぞれ関連付けられます。プラグインは、対応する関連ユーザー名をプロキシユーザー名として使用します。
- 一致するグループ名が `grp2` の場合、認証文字列に関連付けられたユーザー名はありません。プラグインは、プロキシユーザー名として `grp2` を使用します。

LDAP サーバーが DN 形式でグループを返す場合、LDAP プラグインはグループ DN を解析してグループ名を抽出します。

LDAP グループプリファレンスおよびマッピング情報を指定するには、次の原則が適用されます:

- グループプリファレンスおよび認証文字列のマッピング部分を # 接頭辞文字で開始します。
- グループプリファレンスとマッピングの指定は、カンマで区切られた 1 つ以上のアイテムのリストです。各アイテムの形式は、`group_name=user_name` または `group_name` です。アイテムは、グループ名のプリファレンス順にリストする必要があります。LDAP サーバーによって返されるグループ名のセットと一致するものとしてプラグインによって選択されたグループ名の場合、次の 2 つの構文が有効に異なります:
 - `group_name=user_name` (ユーザー名付き) として指定されたアイテムの場合、グループ名は MySQL プロキシユーザー名として使用されるユーザー名にマップされます。
 - `group_name` (ユーザー名なし) として指定されたアイテムの場合、グループ名は MySQL プロキシユーザー名として使用されます。
- スペースなどの特殊文字を含むグループまたはユーザー名を引用符で囲むには、二重引用符 (") 文字で囲みます。たとえば、アイテムのグループ名とユーザー名が `my group name` および `my user name` の場合は、引用符を使用してグループマッピングに記述する必要があります:

```
"my group name"="my user name"
```

アイテムに `my_group_name` および `my_user_name` のグループ名およびユーザー名 (特殊文字を含まない) がある場合、引用符を使用して記述する必要はありません。次のいずれかが有効です:

```
my_group_name=my_user_name
my_group_name="my_user_name"
"my_group_name"=my_user_name
"my_group_name"="my_user_name"
```

- 文字をエスケープするには、その前にバックスラッシュ (\) を付けます。これは、リテラルの二重引用符またはバックスラッシュを含める場合に特に便利です。それ以外の場合は、リテラルの二重引用符またはバックスラッシュは含まれません。
- ユーザー DN は認証文字列に存在する必要はありませんが、存在する場合は、グループプリファレンスおよびマッピング部分の前に存在する必要があります。ユーザー DN は、完全なユーザー DN または + 接頭辞文字を含むユーザー DN 接尾辞として指定できます。(LDAP 認証ユーザー DN 接尾辞を参照してください。)

LDAP 認証ユーザー DN 接尾辞

LDAP 認証プラグインでは、ユーザー DN 情報を提供する認証文字列を + 接頭辞文字で始めることができます:

- + 文字がない場合、認証文字列値は変更されずにそのまま処理されます。
- 認証文字列が + で始まる場合、プラグインは、(+ を削除して) 認証文字列で指定された DN とともに、クライアントによって送信されたユーザー名から完全なユーザー DN 値を構成します。構築された DN では、クライアントユーザー名は LDAP ユーザー名を指定する属性の値になります。これはデフォルトでは `uid` です。属性を変更するには、適切なシステム変数 (`authentication_ldap_simple_user_search_attr` または `authentication_ldap_sasl_user_search_attr`) を変更します。認証文字列は、`mysql.user` システムテーブルに指定されているとおりに格納され、完全なユーザー DN は認証前に即時に構成されます。

このアカウント認証文字列の先頭には + がないため、完全なユーザー DN とみなされます:

```
CREATE USER 'baldwin'
IDENTIFIED WITH authentication_ldap_simple
```

```
AS 'uid=admin,ou=People,dc=example,dc=com';
```

クライアントは、アカウント (**baldwin**) で指定されたユーザー名で接続します。この場合、認証文字列に接頭辞がなく、ユーザー DN を完全に指定するため、その名前は使用されません。

このアカウント認証文字列の先頭には **+** があるため、ユーザー DN の一部として取得されます:

```
CREATE USER 'accounting'  
  IDENTIFIED WITH authentication_ldap_simple  
  AS '+ou=People,dc=example,dc=com';
```

クライアントは、アカウント (**accounting**) で指定されたユーザー名で接続します。この場合、ユーザー DN を構成するための認証文字列とともに **uid** 属性として使用されます: **uid=accounting,ou=People,dc=example,dc=com**

前述の例のアカウントには空でないユーザー名が含まれているため、クライアントは常にアカウント定義で指定されたのと同じ名前を使用して MySQL サーバーに接続します。 [プロキシを使用した LDAP 認証](#) で説明されているデフォルトの匿名 **"@%"** プロキシアカウントなど、アカウントのユーザー名が空の場合、クライアントは様々なユーザー名で MySQL サーバーに接続することがあります。ただし、原則は同じです: 認証文字列が **+** で始まる場合、プラグインはクライアントから送信されたユーザー名と認証文字列を使用してユーザー DN を構築します。

LDAP 認証方式

LDAP 認証プラグインは、構成可能な認証方式を使用します。適切なシステム変数および使用可能なメソッドの選択肢は、プラグイン固有です:

- **authentication_ldap_simple** プラグインの場合: メソッドを構成するには、**authentication_ldap_simple_auth_method_name** システム変数を設定します。許可される選択肢は、**SIMPLE** および **AD-FOREST** です。
- **authentication_ldap_sasl** プラグインの場合: メソッドを構成するには、**authentication_ldap_sasl_auth_method_name** システム変数を設定します。許可される選択肢は、**SCRAM-SHA-1**、**SCRAM-SHA-256** および **GSSAPI** です。(ホストシステムで実際に使用可能な SASL LDAP メソッドを判別するには、**Authentication_ldap_sasl_supported_methods** ステータス変数の値を確認します。)

許可される各メソッドの詳細は、システム変数の説明を参照してください。また、方法によっては、次の各セクションで説明するように、追加の構成が必要になる場合があります。

GSSAPI/Kerberos 認証方式

Generic Security Service Application Program Interface (GSSAPI) は、セキュリティ抽象化インタフェースです。Kerberos は、その抽象インタフェースを介して使用できる特定のセキュリティプロトコルのインスタンスです。GSSAPI を使用すると、アプリケーションは Kerberos に対して認証を行い、サービス資格を取得してから、それらの資格を使用してほかのサービスへのセキュアなアクセスを有効にします。

そのようなサービスの 1 つは LDAP で、クライアント側とサーバー側の SASL LDAP 認証プラグインによって使用されます。**authentication_ldap_sasl_auth_method_name** システム変数が **GSSAPI** に設定されている場合、これらのプラグインは GSSAPI/Kerberos 認証方式を使用します。この場合、プラグインは LDAP メッセージを直接使用せずに Kerberos を使用して安全に通信します。次に、サーバー側プラグインは LDAP サーバーと通信して、LDAP 認証メッセージを解釈し、LDAP グループを取得します。

GSSAPI/Kerberos は、Linux 上の MySQL クライアントおよびサーバーの認証方式としてのみサポートされます。これは、Kerberos がデフォルトで有効になっている Microsoft Active Directory を使用してアプリケーションが LDAP にアクセスする Linux 環境で役立ちます。

次の説明では、GSSAPI メソッドを使用するための構成要件について説明します。次の一般的な Kerberos 用語など、Kerberos の概念および操作に精通していることを前提としています:

- プリンシパル = ユーザーやサービスなどの名前付きエンティティ。
- KDC = AS および TGS で構成されるキー配布センター。
- AS = KDC の一部である認証サーバー。TGT の取得に必要な初期チケットを提供します。
- TGS = KDC の一部であるチケット認可サービス。
- TGT = チケット認可チケット。TGS に提示され、サービスアクセス用のサービスチケットを取得します。

Kerberos 認証には KDC サーバーと LDAP サーバーの両方が必要です。この要件は、様々な方法で満たすことができます:

- Active Directory には両方のサーバーが含まれており、Kerberos 認証は Active Directory LDAP サーバーでデフォルトで有効になっています。
- OpenLDAP には LDAP サーバーが用意されていますが、別の KDC サーバーが必要になる場合があり、追加の Kerberos 設定が必要です。

Kerberos はクライアントホストでも使用可能である必要があります。クライアントは、TGT を取得するためにパスワードを使用して AS に接続します。その後、クライアントは TGT を使用して TGS から LDAP などの他のサービスへのアクセスを取得します。

次のセクションでは、MySQL で SASL LDAP 認証に GSSAPI/Kerberos を使用するための構成ステップについて説明します:

- [Kerberos 設定の確認](#)
- [GSSAPI/Kerberos 用のサーバー側 SASL LDAP 認証プラグインの構成](#)
- [GSSAPI/Kerberos を使用する MySQL アカウントの作成](#)
- [MySQL アカウントを使用した MySQL Server への接続](#)
- [/etc/krb5.conf クライアント構成パラメータ](#)

Kerberos 設定の確認

次の例は、Active Directory で Kerberos の可用性をテストする方法を示しています。この例では、次のことを想定しています:

- Active Directory は、IP アドレス `198.51.100.10` の `ldap_auth.example.com` という名前のホストで実行されています。
- `MYSQL.LOCAL` ドメインは、MySQL-related Kerberos 認証および LDAP 検索に使用されます。
- `bredon@MYSQL.LOCAL` という名前の主体が KDC に登録されます。(後で説明しますが、このプリンシパル名は GSSAPI/Kerberos を使用して MySQL サーバーに対する認証を行う MySQL ユーザーにも使用されます。)

これらの前提条件を満たしている場合は、次の手順に従います:

1. Kerberos ライブラリがオペレーティングシステムに正しくインストールおよび構成されていることを確認します。たとえば、MySQL 認証中に使用する `MYSQL.LOCAL` ドメインを構成するには、Kerberos 構成ファイル `/etc/krb5.conf` に次のような内容を含める必要があります:

```
[realms]
MYSQL.LOCAL = {
  kdc = ldap_auth.example.com
  admin_server = ldap_auth.example.com
  default_domain = MYSQL.LOCAL
}
```

2. サーバーホストの `/etc/hosts` にエントリを追加する必要がある場合があります:

```
198.51.100.10 ldap_auth ldap_auth.example.com
```

3. Kerberos 認証が正しく機能するかどうかを確認します:

- a. `kinit` を使用して、Kerberos に対する認証を行います:

```
kinit bredon@MYSQL.LOCAL
```

このコマンドは、`bredon@MYSQL.LOCAL` という名前の Kerberos 主体の認証を行います。入力を求めるプロンプトが表示されたら、プリンシパルパスワードを入力します。KDC は、ほかの Kerberos-aware アプリケーションで使用するためにクライアント側にキャッシュされた TGT を返します。

- b. `klist` を使用して TGT が正しく取得されたかどうかを確認します。出力は次のようになります:

```
Ticket cache: FILE:/tmp/krb5cc_244306  
Default principal: bredon@MYSQL.LOCAL
```

```
Valid starting Expires Service principal  
03/23/2020 08:18:33 03/23/2020 18:18:33 krbtgt/MYSQL.LOCAL@MYSQL.LOCAL
```

4. **MYSQL.LOCAL** ドメイン内のユーザーを検索する次のコマンドを使用して、**ldapsearch** が Kerberos TGT と連携するかどうかを確認します:

```
ldapsearch -h 198.51.100.10 -Y GSSAPI -b "dc=MYSQL,dc=LOCAL"
```

GSSAPI/Kerberos 用のサーバー側 SASL LDAP 認証プラグインの構成

前述のように、LDAP サーバーが Kerberos を介してアクセス可能であると仮定して、GSSAPI/Kerberos 認証方式を使用するようにサーバー側 SASL LDAP 認証プラグインを構成します。(LDAP プラグインの一般的なインストール情報については、[LDAP プラガブル認証のインストール](#) を参照してください。)次に、サーバー **my.cnf** ファイルに含まれるプラグイン関連の設定の例を示します:

```
[mysqld]  
plugin-load-add=authentication_ldap_sasl.so  
authentication_ldap_sasl_auth_method_name="GSSAPI"  
authentication_ldap_sasl_server_host=198.51.100.10  
authentication_ldap_sasl_server_port=389  
authentication_ldap_sasl_bind_root_dn="cn=admin,cn=users,dc=MYSQL,dc=LOCAL"  
authentication_ldap_sasl_bind_root_pwd="password"  
authentication_ldap_sasl_bind_base_dn="cn=users,dc=MYSQL,dc=LOCAL"  
authentication_ldap_sasl_user_search_attr="sAMAccountName"
```

これらのオプションファイルの設定では、SASL LDAP プラグインを次のように構成します:

- `--plugin-load-add` オプションはプラグインをロードします (必要に応じて、プラットフォームの `.so` 接尾辞を調整します)。以前に **INSTALL PLUGIN** ステートメントを使用してプラグインをロードした場合、このオプションは不要です。
- GSSAPI/Kerberos を SASL LDAP 認証方式として使用するには、`authentication_ldap_sasl_auth_method_name` を **GSSAPI** に設定する必要があります。
- `authentication_ldap_sasl_server_host` および `authentication_ldap_sasl_server_port` は、認証用の Active Directory サーバーホストの IP アドレスとポート番号を示します。
- `authentication_ldap_sasl_bind_root_dn` および `authentication_ldap_sasl_bind_root_pwd` は、グループ検索機能のルート DN およびパスワードを構成します。この機能は必須ですが、ユーザーには検索権限がない可能性があります。このような場合は、ルート DN 情報を指定する必要があります。
 - DN オプション値では、`admin` はユーザー検索を実行する権限を持つ管理 LDAP アカウントの名前である必要があります。
 - パスワードオプションの値で、`password` は `admin` アカウントのパスワードである必要があります。
- `authentication_ldap_sasl_bind_base_dn` は、**MYSQL.LOCAL** ドメイン内のユーザーを検索するためのユーザー DN ベースパスを示します。
- `authentication_ldap_sasl_user_search_attr` では、標準の Active Directory 検索属性 `sAMAccountName` が指定されています。この属性は、ログオン名を照合するために検索で使用されます。属性値はユーザー DN 値と同じではありません。

GSSAPI/Kerberos を使用する MySQL アカウントの作成

GSSAPI/Kerberos メソッドで SASL LDAP 認証プラグインを使用した MySQL 認証は、Kerberos 主体であるユーザーに基づいています。次の説明では、このユーザーとして **bredon@MYSQL.LOCAL** という名前のプリンシパルを使用します。これは複数の場所に登録する必要があります:

- Kerberos 管理者は、ユーザー名を Kerberos プリンシパルとして登録する必要があります。この名前にはドメイン名を含める必要があります。プリンシパル名とパスワードは、クライアントが Kerberos で認証し、TGT を取得するために使用されます。

- LDAP 管理者は、LDAP エントリにユーザー名を登録する必要があります。例:

```
uid=bredon,dc=MYSQL,dc=LOCAL
```

注記

Active Directory (デフォルトの認証方法として Kerberos を使用) では、ユーザーを作成すると、Kerberos プリンシパルと LDAP エントリの両方が作成されます。

- MySQL DBA は、ユーザー名として Kerberos プリンシパル名を持ち、SASL LDAP プラグインを使用して認証するアカウントを作成する必要があります。

Kerberos 主体と LDAP エントリが適切なサービス管理者によって登録されており、前述の `my.cnf` 設定を使用して MySQL サーバーが起動されていると仮定して、ドメイン名を含む Kerberos 主体名に対応する MySQL アカウントを作成します。

注記

SASL LDAP プラグインは、Kerberos 認証に一定のユーザー DN を使用し、MySQL から構成されたユーザー DN を無視します。これには、次のような影響があります:

- GSSAPI/Kerberos 認証を使用する MySQL アカウントの場合、`CREATE USER` ステートメントまたは `ALTER USER` ステートメントの認証文字列にはユーザー DN が含まれていない必要があります。これは効果がないためです。
- 認証文字列にはユーザー DN が含まれていないため、必要なプロキシユーザーにマップされたプロキシユーザーとしてユーザーを処理できるように、グループマッピング情報を含める必要があります。LDAP 認証プラグインによるプロキシの詳細は、[プロキシを使用した LDAP 認証](#) を参照してください。

次のステートメントは、`proxied_krb_usr` という名前のプロキシユーザーの権限を引き受ける `bredon@MYSQL.LOCAL` という名前のプロキシユーザーを作成します。同じ特権を持つほかの GSSAPI/Kerberos ユーザーも、同じプロキシユーザーのプロキシユーザーとして同様に作成できます。

```
-- create proxy account
CREATE USER 'bredon@MYSQL.LOCAL'
  IDENTIFIED WITH authentication_ldap_sasl
  BY '#krb_grp=proxied_krb_user';

-- create proxied account and grant its privileges;
-- use mysql_no_login plugin to prevent direct login
CREATE USER 'proxied_krb_user'
  IDENTIFIED WITH mysql_no_login;
GRANT ALL
  ON krb_user_db.*
  TO 'proxied_krb_user';

-- grant to proxy account the
-- PROXY privilege for proxied account
GRANT PROXY
  ON 'proxied_krb_user'
  TO 'bredon@MYSQL.LOCAL';
```

最初の `CREATE USER` ステートメントと `GRANT PROXY` ステートメントでプロキシアカウント名の引用符をよく確認します:

- ほとんどの MySQL アカウントでは、ユーザーとホストはアカウント名の別々の部分であるため、`'user_name'@'host_name'` とは別に引用符で囲まれます。
- Kerberos 認証の場合、アカウント名のユーザー部分にはプリンシパルドメインが含まれるため、`'bredon@MYSQL.LOCAL'` は単一の値として引用符で囲まれます。ホスト部分が指定されていないため、完全な MySQL アカウント名ではデフォルトの `'%'` がホスト部分として使用されます: `'bredon@MYSQL.LOCAL'@'%'`

プロキシ設定されたアカウントは、`mysql_no_login` 認証プラグインを使用して、クライアントがアカウントを使用して MySQL サーバーに直接ログインできないようにします。かわりに、LDAP を使用して認証するユーザーは `bredon@MYSQL.LOCAL` プロキシアカウントを使用する必要があります。(これは、`mysql_no_login` プラグインが

インストールされていることを前提としています。手順については、[セクション6.4.1.8「ログインなしのプラグイン認証」](#)を参照してください。)プロキシ設定されたアカウントを直接使用しないように保護する別の方法については、[プロキシアカウントへの直接ログインの防止](#)を参照してください。

MySQL アカウントを使用した MySQL Server への接続

GSSAPI/Kerberos を使用する MySQL アカウントが設定されると、クライアントは Kerberos に対して認証を行い、そのアカウントを使用して MySQL サーバーに接続できます。Kerberos 認証は、MySQL クライアントプログラムの起動前または起動時に実行できます:

- クライアントユーザーは、MySQL クライアントプログラムを起動する前に、MySQL とは別に TGT を取得できません。たとえば、クライアントユーザーは、Kerberos プリンシパル名とプリンシパルパスワードを指定して、`kinit` を使用して Kerberos への認証を行うことができます。TGT はキャッシュされ、クライアント側 SASL LDAP 認証プラグインなどの他の Kerberos-aware アプリケーションで使用できるようになります。この場合、MySQL クライアントプログラムは TGT を使用して MySQL サーバーに対する認証を行うため、ユーザー名またはパスワードを指定せずにクライアントを呼び出します:

```
shell> kinit bredon@MYSQL.LOCAL
Password for bredon@MYSQL.LOCAL: (enter password here)
shell> mysql --default-auth=authentication_ldap_sasl_client
```

MySQL クライアントコマンドに資格証明が含まれている場合は、次のように処理されます:

- コマンドにユーザー名が含まれている場合、TGT 内のプリンシパル名と一致しないと認証は失敗します。
- コマンドにパスワードが含まれている場合、パスワードは無視されます。認証は TGT に基づいているため、ユーザーが指定したパスワードが正しくない場合でも成功させることができます。このため、パスワードが無視される原因となる有効な TGT が見つかった場合、プラグインは警告を生成します。
- TGT が存在しない場合、クライアント側 SASL LDAP 認証プラグイン自体が KDC から TGT を取得できます。この場合、クライアントを起動するには、MySQL アカウントに関連付けられた Kerberos プリンシパルの名前とパスワードを指定します (コマンドを単一行で入力し、プロンプトでプリンシパルパスワードを入力します):

```
shell> mysql --default-auth=authentication_ldap_sasl_client
--user=bredon@MYSQL.LOCAL
--password
Enter password: (enter password here)
```

- `client` コマンドで主体名がユーザー名として指定されておらず、TGT が存在しないためにクライアント側プラグインが Kerberos キャッシュを空にした場合、認証は失敗します。

TGT が存在するかどうか不明な場合は、`klist` を使用して確認できます。

認証は次のように行われます:

- クライアントは TGT を使用して Kerberos を使用して認証します。
- サーバーはプリンシパルの LDAP エントリを検索し、それを使用して `bredon@MYSQL.LOCAL` MySQL プロキシアカウントの接続を認証します。
- プロキシアカウント認証文字列 (`#krb_grp=proxied_krb_user'`) のグループマッピング情報は、認証されたプロキシユーザーが `proxied_krb_user` である必要があることを示します。
- `bredon@MYSQL.LOCAL` は `proxied_krb_user` のプロキシとして扱われ、次のクエリーは次のような出力を返します:

```
mysql> SELECT USER(), CURRENT_USER(), @@proxy_user;
+-----+-----+-----+
| USER()          | CURRENT_USER() | @@proxy_user      |
+-----+-----+-----+
| bredon@MYSQL.LOCAL@localhost | proxied_krb_user@% | 'bredon@MYSQL.LOCAL'@%' |
+-----+-----+-----+
```

`USER()` 値は、クライアントコマンド (`bredon@MYSQL.LOCAL`) およびクライアントの接続元ホスト (`localhost`) に使用されるユーザー名を示します。

`CURRENT_USER()` 値は、プロキシ設定されたユーザーアカウントのフルネームで、`proxied_krb_user` ユーザー部分と `%` ホスト部分で構成されます。

`@@proxy_user` 値は、MySQL サーバーへの接続に使用されるアカウントのフルネームを示します。これは、`bredon@MYSQL.LOCAL` ユーザー部分と `%` ホスト部分で構成されます。

これは、プロキシが `bredon@MYSQL.LOCAL` プロキシユーザーアカウントを介して発生し、`bredon@MYSQL.LOCAL` が `proxied_krb_user` プロキシユーザーアカウントに付与された権限を引き受けることを示しています。

取得された TGT はクライアント側にキャッシュされ、パスワードを再度指定せずに期限切れになるまで使用できます。ただし、TGT が取得されると、クライアント側プラグインはそれを使用してサービスチケットを取得し、サーバー側プラグインと通信します。

クライアント側プラグイン自体が TGT を取得する場合、クライアントユーザーは TGT を再利用しないようにすることができます。`/etc/krb5.conf` クライアント構成パラメータで説明されているように、ローカル `/etc/krb5.conf` ファイルを使用すると、クライアント側プラグインが TGT の処理を完了したときに TGT を破棄できます。

サーバー側プラグインは TGT 自体または TGT の取得に使用される Kerberos パスワードにアクセスできません。

LDAP 認証プラグインはキャッシュメカニズム (ローカルファイル内、メモリー内などの記憶域) を制御しませんが、`kswitch` などの Kerberos ユーティリティをこの目的で使用できます。

/etc/krb5.conf クライアント構成パラメータ

クライアント側 SASL LDAP プラグインは、ローカル `/etc/krb5.conf` ファイルを読み取ります。このファイルが欠落しているかアクセスできない場合は、エラーが発生します。ファイルがアクセス可能であると仮定すると、オプションの `[appdefaults]` グループを使用して、プラグインで使用される情報を提供できます。このような情報は、グループの `MySQL` セクションに配置します。例:

```
[appdefaults]
MySQL = {
  ldap_server_host = "ldap_host.example.com"
  ldap_destroy_tgt = true
}
```

クライアント側プラグインは、`MySQL` セクションの次のパラメータを認識します:

- `ldap_server_host` 値は LDAP サーバーホストを指定し、そのホストが `[realms]` グループで指定された KDC サーバーホストと異なる場合に役立ちます。デフォルトでは、プラグインは KDC サーバーホストを LDAP サーバーホストとして使用します。
- `ldap_destroy_tgt` 値は、クライアント側プラグインが TGT を取得して使用したあとに破棄するかどうかを示します。デフォルトでは、`ldap_destroy_tgt` は `false` ですが、TGT の再利用を避けるために `true` に設定できます。(この設定は、クライアント側プラグインによって作成された TGT にのみ適用され、MySQL の外部で作成された TGT には適用されません。)

LDAP 検索照会

LDAP サーバーは、LDAP 検索を別の LDAP サーバー (LDAP リフェラルと呼ばれる機能) に委任するように構成できます。サーバー `a.example.com` が `"dc=example,dc=com"` ルート DN を保持し、別のサーバー `b.example.com` に検索を委任するとします。これを有効にするために、`a.example.com` は次の属性を持つ名前付き参照オブジェクトで構成されます:

```
dn: dc=subtree,dc=example,dc=com
objectClass: referral
objectClass: extensibleObject
dc: subtree
ref: ldap://b.example.com/dc=subtree,dc=example,dc=com
```

LDAP 参照を有効にする問題は、検索ベース DN がルート DN で、参照オブジェクトが設定されていない場合に、検索が LDAP 操作エラーで失敗する可能性があることです。LDAP リフェラルが `ldap.conf` 構成ファイルでグローバルに設定されている場合でも、MySQL DBA は LDAP 認証プラグインでこのようなリフェラルエラーを回避できます。

各プラグインとの通信時に LDAP サーバーが LDAP リフェラルを使用するかどうかをプラグイン固有のベースで構成するには、`authentication_ldap_simple_referral` および `authentication_ldap_sasl_referral` システム変数を設定します。変数を `ON` または `OFF` に設定すると、対応する LDAP 認証プラグインによって、MySQL 認証時にリフェラルを使用するかどうか LDAP サーバーに通知されます。各変数にはプラグイン固有の効果があり、LDAP サーバーと通信する他のアプリケーションには影響しません。デフォルトでは、どちらの変数も `OFF` です。

6.4.1.8 ログインなしのプラグブル認証

`mysql_no_login` サーバー側認証プラグインは、それを使用するアカウントへのすべてのクライアント接続を防ぎます。このプラグインのユースケースは次のとおりです:

- 通常のユーザーに権限を公開せずに、昇格された権限を持つストアドプログラムおよびビューを実行できる必要があるアカウント。
- 直接ログインを許可しないが、プロキシアカウントを介してのみアクセスすることを目的としたプロキシアカウント。

次の表には、プラグインおよびライブラリファイルの名前を示します。ファイル名のサフィクスは、システムによって異なる場合があります。ファイルは、`plugin_dir` システム変数で指定されたディレクトリに配置する必要があります。

表 6.20 ログインなし認証のプラグインおよびライブラリ名

プラグインまたはファイル	プラグインまたはファイル名
サーバー側プラグイン	<code>mysql_no_login</code>
クライアント側プラグイン	なし
ライブラリファイル	<code>mysql_no_login.so</code>

次の各セクションでは、ログインなしのプラグブル認証に固有のインストールおよび使用方法について説明します:

- [ログインなしのプラグブル認証のインストール](#)
- [ログインなしのプラグブル認証のアンインストール](#)
- [ログインなしのプラグブル認証の使用](#)

MySQL のプラグブル認証に関する一般的な情報については、[セクション6.2.17「プラグブル認証」](#)を参照してください。プロキシユーザーについては、[セクション6.2.18「プロキシユーザー」](#)を参照してください。

ログインなしのプラグブル認証のインストール

このセクションでは、ログインなしの認証プラグインをインストールする方法について説明します。プラグインのインストールについての一般的な情報は、[セクション5.6.1「プラグインのインストールおよびアンインストール」](#)を参照してください。

サーバーで使用できるようにするには、プラグインライブラリファイルを MySQL プラグインディレクトリ (`plugin_dir` システム変数で指定されたディレクトリ) に配置する必要があります。必要に応じて、サーバーの起動時に `plugin_dir` の値を設定してプラグインディレクトリの場所を構成します。

プラグインライブラリファイルのベース名は `mysql_no_login` です。ファイル名の接尾辞は、プラットフォームごとに異なります (たとえば、`.so` for Unix and Unix-like systems, `.dll` for Windows)。

サーバーの起動時にプラグインをロードするには、`--plugin-load-add` オプションを使用して、プラグインを含むライブラリファイルに名前を付けます。このプラグインのロード方式では、サーバーを起動するたびにオプションを指定する必要があります。たとえば、サーバー `my.cnf` ファイルに次の行を入力し、必要に応じてプラットフォームの `.so` 接尾辞を調整します:

```
[mysqld]
plugin-load-add=mysql_no_login.so
```

`my.cnf` を変更したら、新しい設定を有効にするためにサーバーを再起動します。

または、実行時にプラグインをロードするには、次のステートメントを使用して、必要に応じてプラットフォームの `.so` 接尾辞を調整します:

```
INSTALL PLUGIN mysql_no_login SONAME 'mysql_no_login.so';
```

`INSTALL PLUGIN` はプラグインをただちにロードし、`mysql.plugins` システムテーブルにも登録して、`--plugin-load-add` を必要とせずに後続の通常の起動のたびにサーバーがプラグインをロードするようにします。

プラグインのインストールを確認するには、`INFORMATION_SCHEMA.PLUGINS` テーブルを調べるか、`SHOW PLUGINS` ステートメントを使用します (セクション 5.6.2 「サーバープラグイン情報の取得」を参照)。例:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME LIKE '%login%';
+-----+-----+
| PLUGIN_NAME | PLUGIN_STATUS |
+-----+-----+
| mysql_no_login | ACTIVE      |
+-----+-----+
```

プラグインの初期化に失敗した場合は、サーバーエラーログで診断メッセージを確認してください。

MySQL アカウントを非ログインプラグインに関連付けるには、[ログインなしのプラグイン認証の使用](#) を参照してください。

ログインなしのプラグイン認証のアンインストール

ログインなしの認証プラグインのアンインストールに使用される方法は、インストール方法によって異なります:

- `--plugin-load-add` オプションを使用してサーバーの起動時にプラグインをインストールした場合は、オプションなしでサーバーを再起動します。
- `INSTALL PLUGIN` ステートメントを使用して実行時にプラグインをインストールした場合、サーバーの再起動後もインストールされたままになります。アンインストールするには、`UNINSTALL PLUGIN` を使用します:

```
UNINSTALL PLUGIN mysql_no_login;
```

ログインなしのプラグイン認証の使用

このセクションでは、ログインなしの認証プラグインを使用して、MySQL クライアントプログラムからサーバーへの接続にアカウントが使用されないようにする方法について説明します。[ログインなしのプラグイン認証のインストール](#) で説明されているように、ログインなしプラグインを有効にしてサーバーが実行されていることを前提としています。

`CREATE USER` ステートメントの `IDENTIFIED WITH` 句でログインなしの認証プラグインを参照するには、`mysql_no_login` という名前を使用します。

`mysql_no_login` を使用して認証するアカウントは、ストアプログラムおよびビューオブジェクトの `DEFINER` として使用できます。このようなオブジェクト定義に `SQL SECURITY DEFINER` も含まれている場合は、そのアカウント権限で実行されます。DBA は、この動作を使用して、適切に制御されたインタフェースを介してのみ公開される機密データまたは機密データにアクセスできます。

次の例は、これらの原則を示しています。クライアント接続を許可しないアカウントを定義し、`mysql.user` システムテーブルの特定の列のみを公開するビューに関連付けます:

```
CREATE DATABASE nologindb;
CREATE USER 'nologin'@'localhost'
  IDENTIFIED WITH mysql_no_login;
GRANT ALL ON nologindb.*
  TO 'nologin'@'localhost';
GRANT SELECT ON mysql.user
  TO 'nologin'@'localhost';
CREATE DEFINER = 'nologin'@'localhost'
  SQL SECURITY DEFINER
  VIEW nologindb.myview
```

```
AS SELECT User, Host FROM mysql.user;
```

ビューへの保護されたアクセスを通常のユーザーに提供するには、次の手順を実行します:

```
GRANT SELECT ON nologindb.myview
TO 'ordinaryuser'@'localhost';
```

これで、通常のユーザーはビューを使用して、表示される制限された情報にアクセスできます:

```
SELECT * FROM nologindb.myview;
```

ユーザーがビューによって公開されているカラム以外のカラムにアクセスしようとする、アクセス権を付与されていないユーザーがビューから選択しようとするため、エラーが発生します。

注記

`nologin` アカウントは直接使用できないため、使用するオブジェクトの設定に必要な操作は、`root` またはオブジェクトの作成および `DEFINER` 値の設定に必要な権限を持つ同様のアカウントによって実行される必要があります。

`mysql_no_login` プラグインは、プロキシシナリオでも役立ちます。(プロキシに関連する概念については、[セクション6.2.18「プロキシユーザー」](#)を参照してください。) `mysql_no_login` を使用して認証するアカウントは、プロキシアカウントのプロキシユーザーとして使用できます:

```
-- create proxied account
CREATE USER 'proxied_user'@'localhost'
IDENTIFIED WITH mysql_no_login;
-- grant privileges to proxied account
GRANT ...
ON ...
TO 'proxied_user'@'localhost';
-- permit proxy_user to be a proxy account for proxied account
GRANT PROXY
ON 'proxied_user'@'localhost'
TO 'proxy_user'@'localhost';
```

これにより、クライアントはプロキシアカウント (`proxy_user`) を介して MySQL にアクセスできますが、プロキシユーザー (`proxied_user`) として直接接続してプロキシメカニズムをバイパスすることはできません。 `proxy_user` アカウントを使用して接続するクライアントには `proxied_user` アカウントの権限がありますが、`proxied_user` 自体を使用して接続することはできません。

プロキシ設定されたアカウントを直接使用しないように保護する別の方法については、[プロキシアカウントへの直接ログインの防止](#)を参照してください。

6.4.1.9 ソケットピア資格証明プラグイン認証

サーバー側 `auth_socket` 認証プラグインは、Unix ソケットファイルを介してローカルホストから接続するクライアントを認証します。このプラグインは `SO_PEERCRED` ソケットオプションを使用して、クライアントプログラムを実行しているユーザーに関する情報を取得します。したがって、プラグインは、Linux など、`SO_PEERCRED` オプションをサポートするシステムでのみ使用できます。

このプラグインのソースコードは、ロード可能な認証プラグインを記述する方法を示す比較的単純な例として調査できます。

次の表には、プラグインおよびライブラリファイルの名前を示します。ファイルは、`plugin_dir` システム変数で指定されたディレクトリに配置する必要があります。

表 6.21 ソケットピア資格証明認証のプラグインおよびライブラリ名

プラグインまたはファイル	プラグインまたはファイル名
サーバー側プラグイン	<code>auth_socket</code>
クライアント側プラグイン	なし (説明を参照してください)
ライブラリファイル	<code>auth_socket.so</code>

次の各セクションでは、ソケットプラグブル認証に固有のインストールおよび使用方法について説明します:

- [ソケットプラグブル認証のインストール](#)
- [ソケットプラグブル認証のアンインストール](#)
- [ソケットプラグブル認証の使用](#)

MySQL のプラグブル認証に関する一般的な情報については、[セクション6.2.17「プラグブル認証」](#)を参照してください。

ソケットプラグブル認証のインストール

このセクションでは、ソケット認証プラグインをインストールする方法について説明します。プラグインのインストールについての一般的な情報は、[セクション5.6.1「プラグインのインストールおよびアンインストール」](#)を参照してください。

サーバーで使用できるようにするには、プラグインライブラリファイルを MySQL プラグインディレクトリ (`plugin_dir` システム変数で指定されたディレクトリ) に配置する必要があります。必要に応じて、サーバーの起動時に `plugin_dir` の値を設定してプラグインディレクトリの場所を構成します。

サーバーの起動時にプラグインをロードするには、`--plugin-load-add` オプションを使用して、プラグインを含むライブラリファイルに名前を付けます。このプラグインのロード方式では、サーバーを起動するたびにオプションを指定する必要があります。たとえば、サーバー `my.cnf` ファイルに次の行を挿入します:

```
[mysqld]
plugin-load-add=auth_socket.so
```

`my.cnf` を変更したら、新しい設定を有効にするためにサーバーを再起動します。

または、実行時にプラグインをロードするには、次のステートメントを使用します:

```
INSTALL PLUGIN auth_socket SONAME 'auth_socket.so';
```

`INSTALL PLUGIN` はプラグインをただちにロードし、`mysql.plugins` システムテーブルにも登録して、`--plugin-load-add` を必要とせずに後続の通常の起動のたびにサーバーがプラグインをロードするようにします。

プラグインのインストールを確認するには、`INFORMATION_SCHEMA.PLUGINS` テーブルを調べるか、`SHOW PLUGINS` ステートメントを使用します ([セクション5.6.2「サーバープラグイン情報の取得」](#)を参照)。例:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
FROM INFORMATION_SCHEMA.PLUGINS
WHERE PLUGIN_NAME LIKE '%socket%';
+-----+-----+
| PLUGIN_NAME | PLUGIN_STATUS |
+-----+-----+
| auth_socket | ACTIVE       |
+-----+-----+
```

プラグインの初期化に失敗した場合は、サーバーエラーログで診断メッセージを確認してください。

MySQL アカウントをソケットプラグインに関連付けるには、[ソケットプラグブル認証の使用](#)を参照してください。

ソケットプラグブル認証のアンインストール

ソケット認証プラグインのアンインストールに使用する方法は、インストール方法によって異なります:

- `--plugin-load-add` オプションを使用してサーバーの起動時にプラグインをインストールした場合は、オプションなしでサーバーを再起動します。
- `INSTALL PLUGIN` ステートメントを使用して実行時にプラグインをインストールした場合、サーバーの再起動後もインストールされたままになります。アンインストールするには、`UNINSTALL PLUGIN` を使用します:

```
UNINSTALL PLUGIN auth_socket;
```

ソケットプラグブル認証の使用

ソケットプラグインは、ソケットユーザー名 (オペレーティングシステムユーザー名) がクライアントプログラムによって指定された MySQL ユーザー名とサーバーと一致するかどうかをチェックします。名前が一致しない場合、プラグインはソケットユーザー名が `mysql.user` システムテーブルの行の `authentication_string` カラムで指定された名前と一致するかどうかを確認します。一致が見つかった場合、プラグインは接続を許可します。 `authentication_string` 値は、`CREATE USER` または `ALTER USER` で `IDENTIFIED ...AS` 句を使用して指定できます。

ソケットファイルを介してローカルホストから接続するために `auth_socket` プラグインによって認証される `valerie` というオペレーティングシステムユーザーに対して、MySQL アカウントが作成されているとします:

```
CREATE USER 'valerie'@'localhost' IDENTIFIED WITH auth_socket;
```

`stefanie` というログイン名を持つローカルホスト上のユーザーが `--user=valerie` オプションを付けて `mysql` を呼び出して、ソケットファイル経由で接続する場合、サーバーは `auth_socket` を使用してクライアントを認証します。このプラグインは、`--user` オプションの値 (`valerie`) がクライアントユーザー名 (`stefanie`) とは異なると判断し、接続を拒否します。 `valerie` という名前のユーザーが同じことを試みた場合、プラグインはユーザー名と MySQL ユーザー名が両方とも `valerie` であると判断し、接続を許可します。ただし、TCP/IP などの別のプロトコルを使用して接続されると、`valerie` の場合でもプラグインは接続を拒否します。

`valerie` と `stefanie` の両方のオペレーティングシステムユーザーが、アカウントを使用するソケットファイル接続を介して MySQL にアクセスできるようにするには、次の 2 つの方法があります:

- アカウント作成時に両方のユーザーに名前を付けます。一方は `CREATE USER` に従い、もう一方は認証文字列になります:

```
CREATE USER 'valerie'@'localhost' IDENTIFIED WITH auth_socket AS 'stefanie';
```

- すでに `CREATE USER` を使用して単一ユーザーのアカウントを作成している場合は、`ALTER USER` を使用して 2 番目のユーザーを追加します:

```
CREATE USER 'valerie'@'localhost' IDENTIFIED WITH auth_socket;
ALTER USER 'valerie'@'localhost' IDENTIFIED WITH auth_socket AS 'stefanie';
```

アカウントにアクセスするには、`valerie` と `stefanie` の両方で、接続時に `--user=valerie` を指定します。

6.4.1.10 プラグブル認証のテスト

MySQL には、アカウント資格証明をチェックし、成功または失敗をサーバーエラーログに記録するテストプラグインが含まれています。これは (組込みではなく) ロード可能なプラグインであり、使用する前にインストールする必要があります。

テストプラグインのソースコードは組み込みのネイティブプラグインとは異なり、サーバーソースとは別々のものであるため、ロード可能な認証プラグインを記述する方法を示す比較的単純な例として調査できます。

注記

このプラグインはテストおよび開発を目的としており、本番環境やパブリックネットワークに公開されているサーバーでは使用できません。

次の表には、プラグインおよびライブラリファイルの名前を示します。ファイル名のサフィクスは、システムによって異なる場合があります。ファイルは、`plugin_dir` システム変数で指定されたディレクトリに配置する必要があります。

表 6.22 テスト認証用のプラグインおよびライブラリ名

プラグインまたはファイル	プラグインまたはファイル名
サーバー側プラグイン	<code>test_plugin_server</code>
クライアント側プラグイン	<code>auth_test_plugin</code>
ライブラリファイル	<code>auth_test_plugin.so</code>

次の各セクションでは、プラグブル認証のテストに固有のインストールおよび使用方法について説明します:

- [テストプラグブル認証のインストール](#)
- [テストプラグブル認証のアンインストール](#)
- [テストプラグブル認証の使用](#)

MySQL のプラグブル認証に関する一般的な情報については、[セクション6.2.17「プラグブル認証」](#)を参照してください。

テストプラグブル認証のインストール

このセクションでは、テスト認証プラグインをインストールする方法について説明します。プラグインのインストールについての一般的な情報は、[セクション5.6.1「プラグインのインストールおよびアンインストール」](#)を参照してください。

サーバーで使用できるようにするには、プラグインライブラリファイルを MySQL プラグインディレクトリ (`plugin_dir` システム変数で指定されたディレクトリ) に配置する必要があります。必要に応じて、サーバーの起動時に `plugin_dir` の値を設定してプラグインディレクトリの場所を構成します。

サーバーの起動時にプラグインをロードするには、`--plugin-load-add` オプションを使用して、プラグインを含むライブラリファイルに名前を付けます。このプラグインのロード方式では、サーバーを起動するたびにオプションを指定する必要があります。たとえば、サーバー `my.cnf` ファイルに次の行を入力し、必要に応じてプラットフォームの `.so` 接尾辞を調整します:

```
[mysqld]
plugin-load-add=auth_test_plugin.so
```

`my.cnf` を変更したら、新しい設定を有効にするためにサーバーを再起動します。

または、実行時にプラグインをロードするには、次のステートメントを使用して、必要に応じてプラットフォームの `.so` 接尾辞を調整します:

```
INSTALL PLUGIN test_plugin_server SONAME 'auth_test_plugin.so';
```

`INSTALL PLUGIN` はプラグインをただちにロードし、`mysql.plugins` システムテーブルにも登録して、`--plugin-load-add` を必要とせずに後続の通常の起動のたびにサーバーがプラグインをロードするようにします。

プラグインのインストールを確認するには、`INFORMATION_SCHEMA.PLUGINS` テーブルを調べるか、`SHOW PLUGINS` ステートメントを使用します ([セクション5.6.2「サーバープラグイン情報の取得」](#)を参照)。例:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
FROM INFORMATION_SCHEMA.PLUGINS
WHERE PLUGIN_NAME LIKE '%test_plugin%';
+-----+-----+
| PLUGIN_NAME | PLUGIN_STATUS |
+-----+-----+
| test_plugin_server | ACTIVE |
+-----+-----+
```

プラグインの初期化に失敗した場合は、サーバーエラーログで診断メッセージを確認してください。

MySQL アカウントをテストプラグインに関連付けるには、[テストプラグブル認証の使用](#)を参照してください。

テストプラグブル認証のアンインストール

テスト認証プラグインのアンインストールに使用する方法は、インストール方法によって異なります:

- `--plugin-load-add` オプションを使用してサーバーの起動時にプラグインをインストールした場合は、オプションなしでサーバーを再起動します。
- `INSTALL PLUGIN` ステートメントを使用して実行時にプラグインをインストールした場合、サーバーの再起動後もインストールされたままになります。アンインストールするには、`UNINSTALL PLUGIN` を使用します:

```
UNINSTALL PLUGIN test_plugin_server;
```

テストプラグブル認証の使用

テスト認証プラグインを使用するには、`IDENTIFIED WITH` 句でアカウントを作成し、そのプラグインに名前を付けます:

```
CREATE USER 'testuser'@'localhost'
IDENTIFIED WITH test_plugin_server
BY 'testpassword';
```

次に、サーバーへの接続時に、そのアカウントの `--user` および `--password` オプションを指定します。例:

```
shell> mysql --user=testuser --password
Enter password: testpassword
```

プラグインは、クライアントから受信したパスワードをフェッチし、`mysql.user` システムテーブルのアカウント行の `authentication_string` カラムに格納されている値と比較します。2つの値が一致する場合、プラグインは新しい実効ユーザー ID として `authentication_string` 値を返します。

サーバーエラーログで、認証が成功したかどうかを示すメッセージを確認できます (パスワードが「user」として報告されることに注意してください):

```
[Note] Plugin test_plugin_server reported:
'successfully authenticated user testpassword'
```

6.4.1.11 プラグブル認証システム変数

これらの変数は、適切なサーバー側プラグインがインストールされていないかぎり使用できません:

- `authentication_ldap_sasl_xxx` という形式の名前を持つシステム変数の `authentication_ldap_sasl`
- `authentication_ldap_simple_xxx` という形式の名前を持つシステム変数の `authentication_ldap_simple`

表 6.23 「認証プラグインシステム変数サマリー」

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
<code>authentication_ldap_sasl_auth_method</code>	<code>authentication_ldap_sasl_auth_method</code>	<code>authentication_ldap_sasl_auth_method</code>	はい		グローバル	はい
<code>authentication_ldap_sasl_bind_base</code>	<code>authentication_ldap_sasl_bind_base</code>	<code>authentication_ldap_sasl_bind_base</code>	はい		グローバル	はい
<code>authentication_ldap_sasl_bind_roots</code>	<code>authentication_ldap_sasl_bind_roots</code>	<code>authentication_ldap_sasl_bind_roots</code>	はい		グローバル	はい
<code>authentication_ldap_sasl_bind_roots_pwd</code>	<code>authentication_ldap_sasl_bind_roots_pwd</code>	<code>authentication_ldap_sasl_bind_roots_pwd</code>	はい		グローバル	はい
<code>authentication_ldap_sasl_ca_path</code>	<code>authentication_ldap_sasl_ca_path</code>	<code>authentication_ldap_sasl_ca_path</code>	はい		グローバル	はい
<code>authentication_ldap_sasl_group_search_attr</code>	<code>authentication_ldap_sasl_group_search_attr</code>	<code>authentication_ldap_sasl_group_search_attr</code>	はい		グローバル	はい
<code>authentication_ldap_sasl_group_search_filter</code>	<code>authentication_ldap_sasl_group_search_filter</code>	<code>authentication_ldap_sasl_group_search_filter</code>	はい		グローバル	はい
<code>authentication_ldap_sasl_init_pool_size</code>	<code>authentication_ldap_sasl_init_pool_size</code>	<code>authentication_ldap_sasl_init_pool_size</code>	はい		グローバル	はい
<code>authentication_ldap_sasl_log_status</code>	<code>authentication_ldap_sasl_log_status</code>	<code>authentication_ldap_sasl_log_status</code>	はい		グローバル	はい
<code>authentication_ldap_sasl_max_pool_size</code>	<code>authentication_ldap_sasl_max_pool_size</code>	<code>authentication_ldap_sasl_max_pool_size</code>	はい		グローバル	はい
<code>authentication_ldap_sasl_referral</code>	<code>authentication_ldap_sasl_referral</code>	<code>authentication_ldap_sasl_referral</code>	はい		グローバル	はい
<code>authentication_ldap_sasl_server_host</code>	<code>authentication_ldap_sasl_server_host</code>	<code>authentication_ldap_sasl_server_host</code>	はい		グローバル	はい
<code>authentication_ldap_sasl_server_port</code>	<code>authentication_ldap_sasl_server_port</code>	<code>authentication_ldap_sasl_server_port</code>	はい		グローバル	はい
<code>authentication_ldap_sasl_tls</code>	<code>authentication_ldap_sasl_tls</code>	<code>authentication_ldap_sasl_tls</code>	はい		グローバル	はい
<code>authentication_ldap_sasl_user_search_attr</code>	<code>authentication_ldap_sasl_user_search_attr</code>	<code>authentication_ldap_sasl_user_search_attr</code>	はい		グローバル	はい
<code>authentication_ldap_simple_auth_method</code>	<code>authentication_ldap_simple_auth_method</code>	<code>authentication_ldap_simple_auth_method</code>	はい		グローバル	はい
<code>authentication_ldap_simple_bind_base</code>	<code>authentication_ldap_simple_bind_base</code>	<code>authentication_ldap_simple_bind_base</code>	はい		グローバル	はい

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
authentication_ldap_simple_bind_dn	はい	はい	はい		グローバル	はい
authentication_ldap_simple_bind_pwd	はい	はい	はい		グローバル	はい
authentication_ldap_simple_ca_path	はい	はい	はい		グローバル	はい
authentication_ldap_simple_group_search_attr	はい	はい	はい		グローバル	はい
authentication_ldap_simple_group_search_filter	はい	はい	はい		グローバル	はい
authentication_ldap_simple_init_page_size	はい	はい	はい		グローバル	はい
authentication_ldap_simple_log_status	はい	はい	はい		グローバル	はい
authentication_ldap_simple_max_page_size	はい	はい	はい		グローバル	はい
authentication_ldap_simple_referral	はい	はい	はい		グローバル	はい
authentication_ldap_simple_server_host	はい	はい	はい		グローバル	はい
authentication_ldap_simple_server_port	はい	はい	はい		グローバル	はい
authentication_ldap_simple_tls	はい	はい	はい		グローバル	はい
authentication_ldap_simple_user_search_attr	はい	はい	はい		グローバル	はい
authentication_windows_log_level	はい	はい	はい		グローバル	いいえ
authentication_windows_use_principal_name	はい	はい	はい		グローバル	いいえ

- [authentication_ldap_sasl_auth_method_name](#)

コマンド行形式	<code>--authentication-ldap-sasl-auth-method-name=value</code>
システム変数	<code>authentication_ldap_sasl_auth_method_name</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	SCRAM-SHA-1
有効な値 (≥ 8.0.23)	SCRAM-SHA-1 SCRAM-SHA-256 GSSAPI
有効な値 (≥ 8.0.20, ≤ 8.0.22)	SCRAM-SHA-1 GSSAPI
有効な値 (≤ 8.0.19)	SCRAM-SHA-1

SASL LDAP 認証の場合、認証方式名。認証プラグインと LDAP サーバー間の通信は、パスワードのセキュリティを確保するために、この認証方式に従って行われます。

次の認証方式の値が許可されます:

- **SCRAM-SHA-1:** SASL チャレンジレスポンスメカニズムを使用します。

クライアント側 [authentication_ldap_sasl_client](#) プラグインは、チャレンジを作成して SASL リクエストバッファを取得するためのパスワードを使用して SASL サーバーと通信し、このバッファをサーバー側 [authentication_ldap_sasl](#) プラグインに渡します。クライアント側およびサーバー側 SASL LDAP プラグインは、SASL メッセージを使用して LDAP プロトコル内での資格証明のセキュアな転送を行い、MySQL クライアントとサーバー間でクリアテキストパスワードが送信されないようにします。

- **SCRAM-SHA-256:** SASL チャレンジレスポンスメカニズムを使用します。

この方法は [SCRAM-SHA-1](#) と似ていますが、よりセキュアです。MySQL 8.0.23 以上で使用できます。Cyrus SASL 2.1.27 以上を使用して構築された OpenLDAP サーバーが必要です。

- [GSSAPI](#): パスワードなしおよびチケットベースのプロトコルである Kerberos を使用します。

GSSAPI/Kerberos は、Linux 上の MySQL クライアントおよびサーバーの認証方式としてのみサポートされます。これは、Kerberos がデフォルトで有効になっている Microsoft Active Directory を使用してアプリケーションが LDAP にアクセスする Linux 環境で役立ちます。

クライアント側 [authentication_ldap_sasl_client](#) プラグインは、チケット認可チケット (TGT) を使用して Kerberos からサービスチケットを取得しますが、LDAP サービスは直接使用しません。サーバー側 [authentication_ldap_sasl](#) プラグインは、Kerberos メッセージをクライアント側プラグインと LDAP サーバーの間でルーティングします。このように取得された資格証明を使用して、サーバー側プラグインは LDAP サーバーと通信し、LDAP 認証メッセージを解釈して LDAP グループを取得します。

- [authentication_ldap_sasl_bind_base_dn](#)

コマンド行形式	<code>--authentication-ldap-sasl-bind-base-dn=value</code>
システム変数	<code>authentication_ldap_sasl_bind_base_dn</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	NULL

SASL LDAP 認証の場合、ベース識別名 (DN)。この変数を使用すると、検索ツリー内の特定の場所 (「base」) に検索範囲を固定することで、検索範囲を制限できます。

LDAP ユーザーエントリのセットのメンバーがそれぞれ次の形式であるとします:

```
uid=user_name,ou=People,dc=example,dc=com
```

また、LDAP ユーザーエントリの別のセットのメンバーは、それぞれ次の形式になります:

```
uid=user_name,ou=Admin,dc=example,dc=com
```

検索は、異なるベース DN 値に対して次のように機能します:

- ベース DN が `ou=People,dc=example,dc=com` の場合: 検索では、最初のセットのユーザーエントリのみが検索されます。
- ベース DN が `ou=Admin,dc=example,dc=com` の場合: 検索では、2 番目のセット内のユーザーエントリだけが検索されます。
- ベース DN が `ou=dc=example,dc=com` の場合: 検索では、1 番目または 2 番目のセット内のユーザーエントリが検索されます。

一般に、より具体的なベース DN 値を使用すると、検索範囲が制限されるため、検索速度が向上します。

- [authentication_ldap_sasl_bind_root_dn](#)

コマンド行形式	<code>--authentication-ldap-sasl-bind-root-dn=value</code>
システム変数	<code>authentication_ldap_sasl_bind_root_dn</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列

デフォルト値	NULL
--------	------

SASL LDAP 認証の場合、ルート識別名 (DN)。この変数は、検索を実行する目的で LDAP サーバーに対して認証するための資格証明として `authentication_ldap_sasl_bind_root_pwd` とともに使用されます。認証では、MySQL アカウントが LDAP ユーザー DN を指定しているかどうかに応じて、次のいずれかまたは 2 つの LDAP バインド操作を使用します:

- アカウントがユーザー DN を指定していない場合: `authentication_ldap_sasl` は、`authentication_ldap_sasl_bind_root_dn` および `authentication_ldap_sasl_bind_root_pwd` を使用して初期 LDAP バインディングを実行します。(これらは両方ともデフォルトで空であるため、設定しない場合、LDAP サーバーは匿名接続を許可する必要があります。) 生成されたバインド LDAP ハンドルは、クライアントユーザー名に基づいてユーザー DN を検索するために使用されます。`authentication_ldap_sasl` は、ユーザー DN とクライアント提供のパスワードを使用して、別のバインドを実行します。
 - アカウントがユーザー DN を指定する場合: この場合、最初のバインド操作は不要です。`authentication_ldap_sasl` は、ユーザー DN とクライアント提供のパスワードを使用して単一のバインドを実行します。これは、MySQL アカウントで LDAP ユーザー DN が指定されていない場合より高速です。
- [authentication_ldap_sasl_bind_root_pwd](#)

コマンド行形式	<code>--authentication-ldap-sasl-bind-root-pwd=value</code>
システム変数	<code>authentication_ldap_sasl_bind_root_pwd</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	NULL

SASL LDAP 認証の場合、ルート識別名のパスワード。この変数は、`authentication_ldap_sasl_bind_root_dn` とともに使用されます。その変数の説明を参照してください。

- [authentication_ldap_sasl_ca_path](#)

コマンド行形式	<code>--authentication-ldap-sasl-ca-path=value</code>
システム変数	<code>authentication_ldap_sasl_ca_path</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	NULL

SASL LDAP 認証の場合、認証局ファイルの絶対パス。認証プラグインで LDAP サーバー証明書の検証を実行する場合は、このファイルを指定します。

注記

`authentication_ldap_sasl_ca_path` 変数をファイル名に設定するだけでなく、適切な認証局の証明書をファイルに追加し、`authentication_ldap_sasl_tls` システム変数を有効にする必要があります。これらの変数は、デフォルトの OpenLDAP TLS 構成をオーバーライドするように設定できます。[LDAP プラグイン認証および ldap.conf](#) を参照してください

- [authentication_ldap_sasl_group_search_attr](#)

コマンド行形式	<code>--authentication-ldap-sasl-group-search-attr=value</code>
システム変数	<code>authentication_ldap_sasl_group_search_attr</code>

スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	cn

SASL LDAP 認証の場合、LDAP ディレクトリエントリ内のグループ名を指定する属性の名前。
[authentication_ldap_sasl_group_search_attr](#) のデフォルト値が `cn` の場合、検索では `cn` 値がグループ名として返されます。たとえば、`uid` 値が `user1` の LDAP エントリに `mygroup` の `cn` 属性がある場合、`user1` を検索すると、`mygroup` がグループ名として返されます。

グループまたはプロキシ認証が不要な場合は、この変数を空の文字列にする必要があります。

グループ検索属性が `isMemberOf` の場合、LDAP 認証はユーザー属性 `isMemberOf` 値を直接取得し、グループ情報として割り当てます。グループ検索属性が `isMemberOf` でない場合、LDAP 認証はユーザーがメンバーであるすべてのグループを検索します。(後者がデフォルトの動作です。) この動作は、LDAP グループ情報を格納する方法に基づいています: 1) グループエントリは、ユーザー名の値を持つ `memberUid` または `member` という名前の属性を持つことができます。2) ユーザーエントリは、グループ名の値を持つ `isMemberOf` という名前の属性を持つことができます。

- [authentication_ldap_sasl_group_search_filter](#)

コマンド行形式	<code>--authentication-ldap-sasl-group-search-filter=value</code>
システム変数	<code>authentication_ldap_sasl_group_search_filter</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	<code>(((&(objectClass=posixGroup)(memberUid=%s))(&(objectClass=group)(member=%s))))</code>

SASL LDAP 認証の場合、カスタムグループ検索フィルタ。

検索フィルタ値には、ユーザー名と完全なユーザー DN を表す `{UA}` および `{UD}` 表記を含めることができます。たとえば、`{UA}` は `"admin"` などのユーザー名に置き換えられますが、`{UD}` は `"uid=admin,ou=People,dc=example,dc=com"` などの完全 DN に置き換えられます。次の値がデフォルトで、OpenLDAP と Active Directory の両方をサポートしています:

```
(((&(objectClass=posixGroup)(memberUid={UA}))
(&(objectClass=group)(member={UD})))
```

場合によっては、`memberOf` はグループ情報を保持しない単純なユーザー属性です。柔軟性を高めるために、オプションの `{GA}` 接頭辞をグループ検索属性とともに使用できます。`{GA}` 接頭辞を持つグループ属性は、グループ名を持つユーザー属性として扱われます。たとえば、`{GA}MemberOf` の値では、グループ値が DN の場合、グループ DN の最初の属性値がグループ名として返されます。

- [authentication_ldap_sasl_init_pool_size](#)

コマンド行形式	<code>--authentication-ldap-sasl-init-pool-size=#</code>
システム変数	<code>authentication_ldap_sasl_init_pool_size</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer

デフォルト値	10
最小値	0
最大値	32767

SASL LDAP 認証の場合、LDAP サーバーへの接続プールの初期サイズ。LDAP サーバーに対する同時認証リクエストの平均数に基づいて、この変数の値を選択します。

プラグインは、[authentication_ldap_sasl_init_pool_size](#) と [authentication_ldap_sasl_max_pool_size](#) を一緒に使用して接続プールを管理します:

- 認証プラグインが初期化されると、[authentication_ldap_sasl_max_pool_size=0](#) がプーリングを無効にしないかぎり、[authentication_ldap_sasl_init_pool_size](#) 接続が作成されます。
- 現在の接続プールに空き接続がないときにプラグインが認可リクエストを受信した場合、プラグインは [authentication_ldap_sasl_max_pool_size](#) で指定された最大接続プールサイズまで新しい接続を作成できます。
- プールサイズがすでに最大値に達していて、空き接続がないときにプラグインがリクエストを受信した場合、認証は失敗します。
- プラグインがアンロードされると、プールされたすべての接続が閉じられます。

プラグインシステム変数の設定を変更しても、プール内にすでに存在する接続には影響しない可能性があります。たとえば、LDAP サーバーのホスト、ポートまたは TLS 設定を変更しても、既存の接続には影響しません。ただし、元の変数値が無効で接続プールを初期化できなかった場合、プラグインは次の LDAP リクエストのためにプールを再初期化しようとします。この場合、新しいシステム変数値が再初期化の試行に使用されます。

[authentication_ldap_sasl_max_pool_size=0](#) でプーリングを無効にする場合、プラグインによってオープンされる各 LDAP 接続では、その時点でシステム変数に設定されている値が使用されます。

- [authentication_ldap_sasl_log_status](#)

コマンド行形式	<code>--authentication-ldap-sasl-log-status=#</code>
システム変数	<code>authentication_ldap_sasl_log_status</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1
最小値	1
最大値 (≥ 8.0.18)	6
最大値 (≤ 8.0.17)	5

SASL LDAP 認証の場合、エラーログに書き込まれるメッセージのロギングレベル。次のテーブルに、許可されるレベル値とその意味を示します。

表 6.24 authentication_ldap_sasl_log_status のログレベル

オプション値	記録されるメッセージのタイプ
1	メッセージがありません
2	エラーメッセージ
3	エラーメッセージと警告メッセージ
4	エラー、警告および情報メッセージ

オプション値	記録されるメッセージのタイプ
5	MySQL からの以前のレベルおよびデバッグメッセージと同じ
6	LDAP ライブラリからの前のレベルおよびデバッグメッセージと同じ

ログレベル 6 は、MySQL 8.0.18 で使用できます。

クライアント側では、`AUTHENTICATION_LDAP_CLIENT_LOG` 環境変数を設定することで、メッセージを標準出力に記録できます。許可される値とデフォルト値は、`authentication_ldap_sasl_log_status` の場合と同じです。

`AUTHENTICATION_LDAP_CLIENT_LOG` 環境変数は SASL LDAP 認証にのみ適用されます。この場合のクライアントプラグインは `mysql_clear_password` であり、LDAP 操作については何も認識されないため、単純な LDAP 認証には影響しません。

- [authentication_ldap_sasl_max_pool_size](#)

コマンド行形式	<code>--authentication-ldap-sasl-max-pool-size=#</code>
システム変数	authentication_ldap_sasl_max_pool_size
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1000
最小値	0
最大値	32767

SASL LDAP 認証の場合、LDAP サーバーへの接続プールの最大サイズ。接続プーリングを無効にするには、この変数を 0 に設定します。

この変数は、[authentication_ldap_sasl_init_pool_size](#) とともに使用されます。その変数の説明を参照してください。

- [authentication_ldap_sasl_referral](#)

コマンド行形式	<code>--authentication-ldap-sasl-referral[={OFF ON}]</code>
導入	8.0.20
システム変数	authentication_ldap_sasl_referral
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

SASL LDAP 認証の場合、LDAP 検索リフェラルを有効にするかどうか。LDAP 検索照会を参照してください。

この変数は、デフォルトの OpenLDAP リフェラル構成をオーバーライドするように設定できます。LDAP プラグイン認証および `ldap.conf` を参照してください

- [authentication_ldap_sasl_server_host](#)

コマンド行形式	<code>--authentication-ldap-sasl-server-host=host_name</code>
システム変数	authentication_ldap_sasl_server_host

スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列

SASL LDAP 認証の場合、LDAP サーバーホスト。この変数に指定できる値は、認証方法によって異なります:

- [authentication_ldap_sasl_auth_method_name=SCRAM-SHA-1](#) の場合: LDAP サーバーホストには、ホスト名または IP アドレスを指定できます。
- [authentication_ldap_sasl_auth_method_name=SCRAM-SHA-256](#) の場合: LDAP サーバーホストには、ホスト名または IP アドレスを指定できます。
- [authentication_ldap_sasl_server_port](#)

コマンド行形式	<code>--authentication-ldap-sasl-server-port=port_num</code>
システム変数	authentication_ldap_sasl_server_port
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	389
最小値	1
最大値	32376

SASL LDAP 認証の場合は、LDAP サーバーの TCP/IP ポート番号。

MySQL 8.0.14 では、LDAP ポート番号が 636 または 3269 として構成されている場合、プラグインは LDAP ではなく LDAPS (LDAP over SSL) を使用します。(LDAPS は [startTLS](#) とは異なります。)

- [authentication_ldap_sasl_tls](#)

コマンド行形式	<code>--authentication-ldap-sasl-tls[={OFF ON}]</code>
システム変数	authentication_ldap_sasl_tls
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

SASL LDAP 認証の場合、プラグインによる LDAP サーバーへの接続がセキュリティー保護されているかどうか。この変数が有効な場合、プラグインは TLS を使用して LDAP サーバーにセキュアに接続します。この変数は、デフォルトの OpenLDAP TLS 構成をオーバーライドするように設定できます。[LDAP プラグイン認証および ldap.conf](#) を参照してください。この変数を有効にする場合は、[authentication_ldap_sasl_ca_path](#) 変数も設定できます。

MySQL LDAP プラグインは、プレーン LDAP 接続上で TLS を初期化する StartTLS メソッドをサポートしています。

MySQL 8.0.14 では、LDAPS は [authentication_ldap_sasl_server_port](#) システム変数を設定することで使用できません。

- [authentication_ldap_sasl_user_search_attr](#)

コマンド行形式	<code>--authentication-ldap-sasl-user-search-attr=value</code>
システム変数	<code>authentication_ldap_sasl_user_search_attr</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	<code>uid</code>

SASL LDAP 認証の場合、LDAP ディレクトリエントリ内のユーザー名を指定する属性の名前。ユーザー識別名が指定されていない場合、認証プラグインはこの属性を使用して名前を検索します。たとえば、`authentication_ldap_sasl_user_search_attr` 値が `uid` の場合、ユーザー名 `user1` を検索すると、`uid` 値が `user1` のエントリが検索されます。

- [authentication_ldap_simple_auth_method_name](#)

コマンド行形式	<code>--authentication-ldap-simple-auth-method-name=value</code>
システム変数	<code>authentication_ldap_simple_auth_method_name</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	<code>SIMPLE</code>
有効な値	<code>SIMPLE</code> <code>AD-FOREST</code>

簡易 LDAP 認証の場合は、認証方式名。認証プラグインと LDAP サーバー間の通信は、この認証方式に従って行われます。

注記

すべての単純な LDAP 認証方式では、セキュアな接続を介して LDAP サーバーとの通信が行われるように TLS パラメータを設定することもお勧めします。

次の認証方式の値が許可されます:

- **SIMPLE**: 簡易 LDAP 認証を使用します。この方法では、MySQL アカウントが LDAP ユーザー識別名を指定しているかどうかに応じて、いずれかまたは 2 つの LDAP バインド操作が使用されます。[authentication_ldap_simple_bind_root_dn](#) の説明を参照してください。
 - **AD-FOREST**: 認証によって Active Directory フォレスト内のすべてのドメインが検索され、ユーザーが特定のドメインで見つかるまで各 Active Directory ドメインへの LDAP バインドが実行されるような、**SIMPLE** のバリエーション。
- [authentication_ldap_simple_bind_base_dn](#)

コマンド行形式	<code>--authentication-ldap-simple-bind-base-dn=value</code>
システム変数	<code>authentication_ldap_simple_bind_base_dn</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ

型	文字列
デフォルト値	NULL

単純な LDAP 認証の場合は、ベース識別名 (DN)。この変数を使用すると、検索ツリー内の特定の場所 (「base」) に検索範囲を固定することで、検索範囲を制限できます。

LDAP ユーザーエントリのセットのメンバーがそれぞれ次の形式であるとして:

```
uid=user_name,ou=People,dc=example,dc=com
```

また、LDAP ユーザーエントリの別のセットのメンバーは、それぞれ次の形式になります:

```
uid=user_name,ou=Admin,dc=example,dc=com
```

検索は、異なるベース DN 値に対して次のように機能します:

- ベース DN が `ou=People,dc=example,dc=com` の場合: 検索では、最初のセットのユーザーエントリのみが検索されます。
- ベース DN が `ou=Admin,dc=example,dc=com` の場合: 検索では、2 番目のセット内のユーザーエントリだけが検索されます。
- ベース DN が `ou=dc=example,dc=com` の場合: 検索では、1 番目または 2 番目のセット内のユーザーエントリが検索されます。

一般に、より具体的なベース DN 値を使用すると、検索範囲が制限されるため、検索速度が向上します。

- [authentication_ldap_simple_bind_root_dn](#)

コマンド行形式	<code>--authentication-ldap-simple-bind-root-dn=value</code>
システム変数	<code>authentication_ldap_simple_bind_root_dn</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	NULL

単純な LDAP 認証の場合は、ルート識別名 (DN)。この変数は、検索を実行する目的で LDAP サーバーに対して認証するための資格証明として `authentication_ldap_simple_bind_root_pwd` とともに使用されます。認証では、MySQL アカウントが LDAP ユーザー DN を指定しているかどうかに応じて、次のいずれかまたは 2 つの LDAP バインド操作を使用します:

- アカウントがユーザー DN を指定していない場合: `authentication_ldap_simple` は、`authentication_ldap_simple_bind_root_dn` および `authentication_ldap_simple_bind_root_pwd` を使用して初期 LDAP バインディングを実行します。(これらは両方ともデフォルトで空であるため、設定しない場合、LDAP サーバーは匿名接続を許可する必要があります。) 生成されたバインド LDAP ハンドルは、クライアントユーザー名に基づいてユーザー DN を検索するために使用されます。`authentication_ldap_simple` は、ユーザー DN とクライアント提供のパスワードを使用して、別のバインドを実行します。
 - アカウントがユーザー DN を指定する場合: この場合、最初のバインド操作は不要です。`authentication_ldap_simple` は、ユーザー DN とクライアント提供のパスワードを使用して単一のバインドを実行します。これは、MySQL アカウントで LDAP ユーザー DN が指定されていない場合より高速です。
- [authentication_ldap_simple_bind_root_pwd](#)

コマンド行形式	<code>--authentication-ldap-simple-bind-root-pwd=value</code>
システム変数	<code>authentication_ldap_simple_bind_root_pwd</code>
スコープ	グローバル

動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	NULL

簡易 LDAP 認証の場合、ルート識別名のパスワード。この変数は、`authentication_ldap_simple_bind_root_dn` とともに使用されます。その変数の説明を参照してください。

- [authentication_ldap_simple_ca_path](#)

コマンド行形式	<code>--authentication-ldap-simple-ca-path=value</code>
システム変数	<code>authentication_ldap_simple_ca_path</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	NULL

簡易 LDAP 認証の場合、認証局ファイルの絶対パス。認証プラグインで LDAP サーバー証明書の検証を実行する場合は、このファイルを指定します。

注記

`authentication_ldap_simple_ca_path` 変数をファイル名に設定するだけでなく、適切な認証局の証明書をファイルに追加し、`authentication_ldap_simple_tls` システム変数を有効にする必要があります。これらの変数は、デフォルトの OpenLDAP TLS 構成をオーバーライドするように設定できます。[LDAP プラグイン認証および ldap.conf](#) を参照してください

- [authentication_ldap_simple_group_search_attr](#)

コマンド行形式	<code>--authentication-ldap-simple-group-search-attr=value</code>
システム変数	<code>authentication_ldap_simple_group_search_attr</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	<code>cn</code>

簡易 LDAP 認証の場合、LDAP ディレクトリエントリ内のグループ名を指定する属性の名前。

`authentication_ldap_simple_group_search_attr` のデフォルト値が `cn` の場合、検索では `cn` 値がグループ名として返されます。たとえば、`uid` 値が `user1` の LDAP エントリに `mygroup` の `cn` 属性がある場合、`user1` を検索すると、`mygroup` がグループ名として返されます。

グループ検索属性が `isMemberOf` の場合、LDAP 認証はユーザー属性 `isMemberOf` 値を直接取得し、グループ情報として割り当てます。グループ検索属性が `isMemberOf` でない場合、LDAP 認証はユーザーがメンバーであるすべてのグループを検索します。(後者がデフォルトの動作です。) この動作は、LDAP グループ情報を格納する方法に基づいています: 1) グループエントリは、ユーザー名の値を持つ `memberUid` または `member` という名前の属性を持つことができます。2) ユーザーエントリは、グループ名の値を持つ `isMemberOf` という名前の属性を持つことができます。

- [authentication_ldap_simple_group_search_filter](#)

コマンド行形式	<code>--authentication-ldap-simple-group-search-filter=value</code>
システム変数	<code>authentication_ldap_simple_group_search_filter</code>

スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	(((&(objectClass=posixGroup)(memberUid=%s)) (&(objectClass=group)(member=%s)))

単純 LDAP 認証の場合は、カスタムグループ検索フィルタ。

検索フィルタ値には、ユーザー名と完全なユーザー DN を表す {UA} および {UD} 表記を含めることができます。たとえば、{UA} は "admin" などのユーザー名に置き換えられますが、{UD} は "uid=admin,ou=People,dc=example,dc=com" などの完全 DN に置き換えられます。次の値がデフォルトで、OpenLDAP と Active Directory の両方をサポートしています:

```
(((&(objectClass=posixGroup)(memberUid={UA}))  
(&(objectClass=group)(member={UD})))
```

場合によっては、memberOf はグループ情報を保持しない単純なユーザー属性です。柔軟性を高めるために、オプションの {GA} 接頭辞をグループ検索属性とともに使用できます。{GA} 接頭辞を持つグループ属性は、グループ名を持つユーザー属性として扱われます。たとえば、{GA}MemberOf の値では、グループ値が DN の場合、グループ DN の最初の属性値がグループ名として返されます。

- [authentication_ldap_simple_init_pool_size](#)

コマンド行形式	--authentication-ldap-simple-init-pool-size=#
システム変数	authentication_ldap_simple_init_pool_size
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	10
最小値	0
最大値	32767

単純な LDAP 認証の場合、LDAP サーバーへの接続プールの初期サイズ。LDAP サーバーに対する同時認証リクエストの平均数に基づいて、この変数の値を選択します。

プラグインは、[authentication_ldap_simple_init_pool_size](#) と [authentication_ldap_simple_max_pool_size](#) を一緒に使用して接続プールを管理します:

- 認証プラグインが初期化されると、[authentication_ldap_simple_max_pool_size=0](#) がプーリングを無効にしない限り、[authentication_ldap_simple_init_pool_size](#) 接続が作成されます。
- 現在の接続プールに空き接続がないときにプラグインが認可リクエストを受信した場合、プラグインは [authentication_ldap_simple_max_pool_size](#) で指定された最大接続プールサイズまで新しい接続を作成できます。
- プールサイズがすでに最大値に達していて、空き接続がないときにプラグインがリクエストを受信した場合、認証は失敗します。
- プラグインがアンロードされると、プールされたすべての接続が閉じられます。

プラグインシステム変数の設定を変更しても、プール内にすでに存在する接続には影響しない可能性があります。たとえば、LDAP サーバーのホスト、ポートまたは TLS 設定を変更しても、既存の接続には影響しません。ただ

し、元の変数値が無効で接続プールを初期化できなかった場合、プラグインは次の LDAP リクエストのためにプールを再初期化しようとします。この場合、新しいシステム変数値が再初期化の試行に使用されます。

`authentication_ldap_simple_max_pool_size=0` でプーリングを無効にする場合、プラグインによってオープンされる各 LDAP 接続では、その時点でシステム変数に設定されている値が使用されます。

- `authentication_ldap_simple_log_status`

コマンド行形式	<code>--authentication-ldap-simple-log-status=#</code>
システム変数	<code>authentication_ldap_simple_log_status</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1
最小値	1
最大値 (≥ 8.0.18)	6
最大値 (≤ 8.0.17)	5

単純な LDAP 認証の場合、エラーログに書き込まれるメッセージのロギングレベル。次のテーブルに、許可されるレベル値とその意味を示します。

表 6.25 `authentication_ldap_simple_log_status` のログレベル

オプション値	記録されるメッセージのタイプ
1	メッセージがありません
2	エラーメッセージ
3	エラーメッセージと警告メッセージ
4	エラー、警告および情報メッセージ
5	MySQL からの以前のレベルおよびデバッグメッセージと同じ
6	LDAP ライブラリからの前のレベルおよびデバッグメッセージと同じ

ログレベル 6 は、MySQL 8.0.18 で使用できます。

- `authentication_ldap_simple_max_pool_size`

コマンド行形式	<code>--authentication-ldap-simple-max-pool-size=#</code>
システム変数	<code>authentication_ldap_simple_max_pool_size</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1000
最小値	0

最大値	32767
-----	-------

簡易 LDAP 認証の場合、LDAP サーバーへの接続プールの最大サイズ。接続プーリングを無効にするには、この変数を 0 に設定します。

この変数は、[authentication_ldap_simple_init_pool_size](#) とともに使用されます。その変数の説明を参照してください。

- [authentication_ldap_simple_referral](#)

コマンド行形式	<code>--authentication-ldap-simple-referral[={OFF ON}]</code>
導入	8.0.20
システム変数	authentication_ldap_simple_referral
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

単純な LDAP 認証の場合、LDAP 検索リフェラルを有効にするかどうか。 [LDAP 検索照会](#) を参照してください。

- [authentication_ldap_simple_server_host](#)

コマンド行形式	<code>--authentication-ldap-simple-server-host=host_name</code>
システム変数	authentication_ldap_simple_server_host
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列

単純な LDAP 認証の場合は、LDAP サーバーホスト。この変数に指定できる値は、認証方法によって異なります:

- [authentication_ldap_simple_auth_method_name=SIMPLE](#) の場合: LDAP サーバーホストには、ホスト名または IP アドレスを指定できます。
- [authentication_ldap_simple_auth_method_name=AD-FOREST](#) の場合。LDAP サーバーホストには、Active Directory ドメイン名を指定できます。たとえば、LDAP サーバー URL が `ldap://example.mem.local:389` の場合、サーバー名は `mem.local` になります。

Active Directory フォレスト設定には、DNS を使用して検出できる複数のドメイン (LDAP サーバー IP) を含めることができます。Unix および Unix に似たシステムでは、Active Directory ドメインの LDAP サーバーを指定する

SRV レコードを使用して DNS サーバーを構成するために、いくつかの追加設定が必要になる場合があります。DNS SRV の詳細は、[RFC 2782](#) を参照してください。

構成に次のプロパティがあるとします:

- Active Directory ドメインに関する情報を提供するネームサーバーには、IP アドレス `10.172.166.100` があります。
- LDAP サーバーの名前は `ldap1.mem.local` から `ldap3.mem.local` で、IP アドレスは `10.172.166.101` から `10.172.166.103` です。

SRV 検索を使用して LDAP サーバーを検出できるようにします。たとえば、コマンドラインでは、次のようなコマンドで LDAP サーバーをリストする必要があります:

```
host -t SRV _ldap._tcp.mem.local
```

次のように DNS 構成を実行します:

1. `/etc/resolv.conf` に行を追加して、Active Directory ドメインに関する情報を提供するネームサーバーを指定します:

```
nameserver 10.172.166.100
```

2. LDAP サーバーの SRV レコードを使用して、ネームサーバーの適切なゾーンファイルを構成します:

```
_ldap._tcp.mem.local. 86400 IN SRV 0 100 389 ldap1.mem.local.
_ldap._tcp.mem.local. 86400 IN SRV 0 100 389 ldap2.mem.local.
_ldap._tcp.mem.local. 86400 IN SRV 0 100 389 ldap3.mem.local.
```

3. サーバーホストを解決できない場合は、`/etc/hosts` で LDAP サーバーの IP アドレスを指定する必要がある場合もあります。たとえば、次のような行をファイルに追加します:

```
10.172.166.101 ldap1.mem.local
10.172.166.102 ldap2.mem.local
10.172.166.103 ldap3.mem.local
```

前述のように DNS を構成すると、サーバー側 LDAP プラグインは LDAP サーバーを検出し、認証が成功するかサーバーがなくなるまで、すべてのドメインで認証を試行できます。

ここで説明したような設定は必要ありません。 `authentication_ldap_simple_server_host` 値に LDAP サーバーホストが指定されている場合、Windows LDAP ライブラリはすべてのドメインを検索し、認証を試みます。

- `authentication_ldap_simple_server_port`

コマンド行形式	<code>--authentication-ldap-simple-server-port=port_num</code>
システム変数	<code>authentication_ldap_simple_server_port</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	<code>389</code>
最小値	<code>1</code>
最大値	<code>32376</code>

単純な LDAP 認証の場合は、LDAP サーバーの TCP/IP ポート番号。

MySQL 8.0.14 では、LDAP ポート番号が 636 または 3269 として構成されている場合、プラグインは LDAP ではなく LDAPS (LDAP over SSL) を使用します。(LDAPS は `startTLS` とは異なります。)

- `authentication_ldap_simple_tls`

コマンド行形式	<code>--authentication-ldap-simple-tls[={OFF ON}]</code>
システム変数	<code>authentication_ldap_simple_tls</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

単純な LDAP 認証の場合、プラグインによる LDAP サーバーへの接続がセキュリティー保護されているかどうか。この変数が有効な場合、プラグインは TLS を使用して LDAP サーバーにセキュアに接続します。この変数は、デフォルトの OpenLDAP TLS 構成をオーバーライドするように設定できます。LDAP プラガブル認証および `ldap.conf` を参照してください。この変数を有効にする場合は、`authentication_ldap_simple_ca_path` 変数も設定できます。

MySQL LDAP プラグインは、プレーン LDAP 接続上で TLS を初期化する StartTLS メソッドをサポートしています。

MySQL 8.0.14 では、LDAPS は `authentication_ldap_simple_server_port` システム変数を設定することで使用できません。

- `authentication_ldap_simple_user_search_attr`

コマンド行形式	<code>--authentication-ldap-simple-user-search-attr=value</code>
システム変数	<code>authentication_ldap_simple_user_search_attr</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	<code>uid</code>

簡易 LDAP 認証の場合、LDAP ディレクトリエントリ内のユーザー名を指定する属性の名前。ユーザー識別名が指定されていない場合、認証プラグインはこの属性を使用して名前を検索します。たとえば、`authentication_ldap_simple_user_search_attr` 値が `uid` の場合、ユーザー名 `user1` を検索すると、`uid` 値が `user1` のエントリが検索されます。

6.4.2 Connection-Control プラグイン

MySQL Server にはプラグインライブラリが含まれており、これを使用すると、管理者は、構成可能な連続して失敗した試行回数あとの、接続試行に対するサーバー応答の遅延を増やすことができます。この機能は、MySQL ユーザーアカウントに対する総当たり攻撃の速度を低下させる抑止機能を提供します。プラグインライブラリには、次の 2 つのプラグインが含まれます:

- `CONNECTION_CONTROL` は着信接続試行をチェックし、必要に応じてサーバーレスポンスに遅延を追加します。このプラグインは、操作を構成できるシステム変数と、基本的なモニタリング情報を提供するステータス変数も公開します。

`CONNECTION_CONTROL` プラグインは、監査プラグインインタフェースを使用します ([Writing Audit Plugins](#) を参照)。情報を収集するために、`MYSQL_AUDIT_CONNECTION_CLASSMASK` イベントクラスをサブスクライブし、`MYSQL_AUDIT_CONNECTION_CONNECT` および `MYSQL_AUDIT_CONNECTION_CHANGE_USER` サブイベントを処理して、接続試行に応答する前にサーバーが遅延を導入する必要があるかどうかを確認します。

- `CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` は、失敗した接続試行に関するより詳細な監視情報を公開する `INFORMATION_SCHEMA` テーブルを実装します。

次の各セクションでは、接続制御プラグインのインストールおよび構成について説明します。 [CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS](#) テーブルの詳細は、[セクション 26.53.1 「INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS テーブル」](#) を参照してください。

6.4.2.1 Connection-Control プラグインのインストール

このセクションでは、接続制御プラグイン、[CONNECTION_CONTROL](#) および [CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS](#) をインストールする方法について説明します。プラグインのインストールについての一般的な情報は、[セクション 5.6.1 「プラグインのインストールおよびアンインストール」](#) を参照してください。

サーバーで使用できるようにするには、プラグインライブラリファイルを MySQL プラグインディレクトリ (`plugin_dir` システム変数で指定されたディレクトリ) に配置する必要があります。必要に応じて、サーバーの起動時に `plugin_dir` の値を設定してプラグインディレクトリの場所を構成します。

プラグインライブラリファイルのベース名は `connection_control` です。ファイル名の接尾辞は、プラットフォームごとに異なります (たとえば、`.so` for Unix and Unix-like systems, `.dll` for Windows)。

サーバーの起動時にプラグインをロードするには、`--plugin-load-add` オプションを使用して、プラグインを含むライブラリファイルに名前を付けます。このプラグインのロード方式では、サーバーを起動するたびにオプションを指定する必要があります。たとえば、サーバー `my.cnf` ファイルに次の行を入力し、必要に応じてプラットフォームの `.so` 接尾辞を調整します:

```
[mysqld]
plugin-load-add=connection_control.so
```

`my.cnf` を変更したら、新しい設定を有効にするためにサーバーを再起動します。

または、実行時にプラグインをロードするには、次のステートメントを使用して、プラットフォームの `.so` 接尾辞を必要に応じて調整します:

```
INSTALL PLUGIN CONNECTION_CONTROL
  SONAME 'connection_control.so';
INSTALL PLUGIN CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS
  SONAME 'connection_control.so';
```

`INSTALL PLUGIN` はプラグインをただちにロードし、`mysql.plugins` システムテーブルにも登録して、`--plugin-load-add` を必要とせずに後続の通常の起動のたびにサーバーがプラグインをロードするようにします。

プラグインのインストールを確認するには、`INFORMATION_SCHEMA.PLUGINS` テーブルを調べるか、`SHOW PLUGINS` ステートメントを使用します ([セクション 5.6.2 「サーバープラグイン情報の取得」](#) を参照)。例:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
  FROM INFORMATION_SCHEMA.PLUGINS
  WHERE PLUGIN_NAME LIKE 'connection%';
+-----+-----+
| PLUGIN_NAME          | PLUGIN_STATUS |
+-----+-----+
| CONNECTION_CONTROL  | ACTIVE        |
| CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS | ACTIVE        |
+-----+-----+
```

プラグインの初期化に失敗した場合は、サーバーエラーログで診断メッセージを確認してください。

プラグインが以前に `INSTALL PLUGIN` に登録されているか、`--plugin-load-add` にロードされている場合は、サーバーの起動時に `--connection-control` および `--connection-control-failed-login-attempts` オプションを使用してプラグインのアクティブ化を制御できます。たとえば、起動時にプラグインをロードし、実行時に削除されないようにするには、次のオプションを使用します:

```
[mysqld]
plugin-load-add=connection_control.so
connection-control=FORCE_PLUS_PERMANENT
connection-control-failed-login-attempts=FORCE_PLUS_PERMANENT
```

指定された接続制御プラグインなしでサーバーが実行されないようにするには、`FORCE` または `FORCE_PLUS_PERMANENT` のオプション値を使用して、プラグインが正常に初期化されない場合にサーバーの起動が強制的に失敗するようにします。

注記

一方のプラグインをもう一方のプラグインなしでインストールすることは可能ですが、完全な接続制御機能を使用するには両方をインストールする必要があります。特に、`CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` プラグインのみのインストールはほとんど使用されません。これは、`CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` テーブルに移入するデータを提供する `CONNECTION_CONTROL` プラグインがない場合、テーブルは常に空になるためです。

- [接続遅延の構成](#)
- [接続失敗評価](#)
- [接続失敗のモニタリング](#)

接続遅延の構成

操作の構成を有効にするために、`CONNECTION_CONTROL` プラグインは次のシステム変数を公開します：

- `connection_control_failed_connections_threshold`: サーバーが後続の接続試行の遅延を追加する前に、アカウントに対して許可された連続して失敗した接続試行の数。失敗した接続カウントを無効にするには、`connection_control_failed_connections_threshold` をゼロに設定します。
- `connection_control_min_connection_delay`: しきい値を超える接続失敗の最小遅延 (ミリ秒)。
- `connection_control_max_connection_delay`: しきい値を超える接続失敗の最大遅延 (ミリ秒)。

`connection_control_failed_connections_threshold` がゼロ以外の場合、失敗した接続カウントは有効になり、次のプロパティがあります：

- `connection_control_failed_connections_threshold` による接続試行の連続失敗により、遅延はゼロになります。
- その後、サーバーは、接続が成功するまで、後続の連続した試行の遅延を増やします。未調整の初期遅延は 1000 ミリ秒 (1 秒) から始まり、試行ごとに 1000 ミリ秒増加します。つまり、アカウントに対して遅延がアクティブ化されると、後続の失敗した試行の未調整の遅延は 1000 ミリ秒、2000 ミリ秒、3000 ミリ秒などになります。
- クライアントで発生する実際の遅延は、`connection_control_min_connection_delay` および `connection_control_max_connection_delay` システム変数の値内に収まるように調整された未調整の遅延です。
- アカウントの遅延がアクティブ化されると、そのアカウントによって成功した最初の接続でも遅延が発生しますが、後続の接続の失敗カウントはリセットされます。

たとえば、デフォルトの `connection_control_failed_connections_threshold` 値が 3 の場合、アカウントによる最初の 3 回連続して失敗した接続試行は遅延されません。4 番目および後続の失敗した接続のために発生する実際の調整済遅延は、`connection_control_min_connection_delay` および `connection_control_max_connection_delay` の値によって異なります：

- `connection_control_min_connection_delay` および `connection_control_max_connection_delay` が 1000 および 20000 の場合、調整された遅延は未調整の遅延と同じで、最大 20000 ミリ秒です。4 番目以降に失敗した接続は 1000 ミリ秒、2000 ミリ秒、3000 ミリ秒など遅延します。
- `connection_control_min_connection_delay` および `connection_control_max_connection_delay` が 1500 および 20000 の場合、4 番目および後続の失敗した接続の調整済遅延は 1500 ミリ秒、2000 ミリ秒、3000 ミリ秒などで、最大 20000 ミリ秒です。
- `connection_control_min_connection_delay` および `connection_control_max_connection_delay` が 2000 および 3000 の場合、4 番目以降に失敗した接続の調整済遅延は 2000 ミリ秒、2000 ミリ秒および 3000 ミリ秒で、後続のすべての失敗した接続も 3000 ミリ秒遅延します。

`CONNECTION_CONTROL` システム変数は、サーバーの起動時または実行時に設定できます。サーバーがレスポンスの遅延を開始する前に、2000 ミリ秒の最小遅延で 4 回連続して失敗した接続試行を許可するとします。サーバーの起動時に関連する変数を設定するには、サーバーの `my.cnf` ファイルに次の行を挿入します：

```
[mysqld]
```

```
plugin-load-add=connection_control.so
connection_control_failed_connections_threshold=4
connection_control_min_connection_delay=2000
```

実行時に変数を設定して永続化するには、次のステートメントを使用します:

```
SET PERSIST connection_control_failed_connections_threshold = 4;
SET PERSIST connection_control_min_connection_delay = 2000;
```

SET PERSIST は、実行中の MySQL インスタンスの値を設定します。また、値が保存され、その後のサーバーの再起動に引き継がれます。後続の再起動に引き継ぐことなく、実行中の MySQL インスタンスの値を変更するには、**PERSIST** ではなく **GLOBAL** キーワードを使用します。セクション13.7.6.1「変数代入の SET 構文」を参照してください。

`connection_control_min_connection_delay` および `connection_control_max_connection_delay` システム変数の最小値と最大値はどちらも 1000 および 2147483647 です。また、各変数に許可される値の範囲は、他の変数の現在の値によっても異なります:

- `connection_control_min_connection_delay` は、`connection_control_max_connection_delay` の現在の値より大きく設定できません。
- `connection_control_max_connection_delay` は、`connection_control_min_connection_delay` の現在の値より小さく設定できません。

したがって、一部の構成に必要な変更を行うには、特定の順序で変数を設定する必要がある場合があります。現在の最小遅延および最大遅延が 1000 および 2000 で、3000 および 5000 に設定するとします。現在の `connection_control_max_connection_delay` 値 2000 より大きいため、最初に `connection_control_min_connection_delay` を 3000 に設定することはできません。かわりに、`connection_control_max_connection_delay` を 5000 に設定してから、`connection_control_min_connection_delay` を 3000 に設定します。

接続失敗評価

CONNECTION_CONTROL プラグインがインストールされると、接続試行がチェックされ、失敗したか成功したかが追跡されます。このため、失敗した接続試行は、クライアントユーザーとホストが既知の MySQL アカウントと一致するが、指定された資格証明が正しくないか、既知のアカウントと一致しない接続試行です。

失敗した接続数は、各接続試行のユーザー/ホストの組合せに基づきます。適用可能なユーザー名とホスト名を決定すると、プロキシが考慮され、次のようになります:

- クライアントユーザーが別のユーザーをプロキシする場合、失敗した接続カウントのアカウントはプロキシユーザーではなくプロキシユーザーです。たとえば、`external_user@example.com` が `proxy_user@example.com` をプロキシする場合、接続カウントでは、プロキシユーザーである `proxy_user@example.com` ではなく、プロキシユーザーである `external_user@example.com` が使用されます。`external_user@example.com` と `proxy_user@example.com` の両方で、`mysql.user` システムテーブルに有効なエントリがあり、それらの間のプロキシ関係が `mysql.proxies_priv` システムテーブルで定義されている必要があります(セクション6.2.18「プロキシユーザー」を参照)。
- クライアントユーザーが別のユーザーをプロキシしないが、`mysql.user` エントリと一致する場合、カウントではそのエントリに対応する `CURRENT_USER()` 値が使用されます。たとえば、ホスト `host1.example.com` から接続しているユーザー `user1` が `user1@host1.example.com` エントリと一致する場合、カウントには `user1@host1.example.com` が使用されます。かわりに、ユーザーが `user1@%.example.com`、`user1@%.com` または `user1@%` エントリと一致する場合、カウントではそれぞれ `user1@%.example.com`、`user1@%.com` または `user1@%` が使用されます。

前述のケースでは、接続試行は `mysql.user` エントリと一致し、リクエストが成功するか失敗するかは、クライアントが正しい認証資格証明を提供するかどうかによって決まります。たとえば、クライアントが間違ったパスワードを提示した場合、接続の試行は失敗します。

接続試行が `mysql.user` エントリと一致しない場合、試行は失敗します。この場合、使用可能な `CURRENT_USER()` 値はなく、接続失敗カウントでは、サーバーによって決定されたクライアントおよびクライアントホストによって指定されたユーザー名が使用されます。たとえば、クライアントがホスト `host2.example.com` からユーザー `user2` として接続しようとする、クライアントリクエストでユーザー名の部分が使用可能になり、サーバーによってホスト情報が決定されます。カウントに使用されるユーザー/ホストの組合せは `user2@host2.example.com` です。

注記

サーバーは、サーバーに接続できるクライアントホストに関する情報を保持します (基本的に、`mysql.user` エントリのホスト値の和集合)。クライアントが他のホストから接続しようとする、サーバーは接続設定の初期段階で試行を拒否します:

```
ERROR 1130 (HY000): Host 'host_name' is not
allowed to connect to this MySQL server
```

このタイプの拒否は非常に早く発生するため、`CONNECTION_CONTROL` ではそれは表示されず、カウントされません。

接続失敗のモニタリング

失敗した接続を監視するには、次の情報ソースを使用します:

- `Connection_control_delay_generated` ステータス変数は、失敗した接続試行に対するレスポンスにサーバーが遅延を追加した回数を示します。これにより、`connection_control_failed_connections_threshold` システム変数で定義されたしきい値に達するまでに発生する試行はカウントされません。
- `INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` テーブルには、アカウント (ユーザー/ホストの組合せ) ごとの接続試行の現在の連続失敗回数に関する情報が表示されます。これにより、遅延したかどうかに関係なく、失敗したすべての試行がカウントされます。

実行時に `connection_control_failed_connections_threshold` に値を割り当てると、次の効果があります:

- 累積されたすべての失敗した接続カウンタがゼロにリセットされます。
- `Connection_control_delay_generated` ステータス変数はゼロにリセットされます。
- `CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` テーブルが空になります。

6.4.2.2 Connection-Control のシステム変数とステータス変数

このセクションでは、操作を構成およびモニターできるようにするために `CONNECTION_CONTROL` プラグインが提供するシステム変数とステータス変数について説明します。

- [Connection-Control システム変数](#)
- [接続制御ステータス変数](#)

Connection-Control システム変数

`CONNECTION_CONTROL` プラグインがインストールされている場合は、次のシステム変数が公開されます:

- `connection_control_failed_connections_threshold`

コマンド行形式	<code>--connection-control-failed-connections-threshold=#</code>
システム変数	<code>connection_control_failed_connections_threshold</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	3
最小値	0
最大値	2147483647

サーバーが後続の接続試行の遅延を追加する前に、アカウントに対して許可された連続して失敗した接続試行の数:

- 変数にゼロ以外の値の N がある場合、サーバーは連続して失敗した試行 $N+1$ から始まる遅延を追加します。アカウントが接続レスポンスの遅延ポイントに到達すると、次の成功した接続に対しても遅延が発生します。

- この変数をゼロに設定すると、失敗した接続カウントが無効になります。この場合、サーバーは遅延を追加しません。

`connection_control_failed_connections_threshold` と他の接続制御システムおよびステータス変数との相互作用の詳細は、[セクション6.4.2.1「Connection-Control プラグインのインストール」](#) を参照してください。

- `connection_control_max_connection_delay`

コマンド行形式	<code>--connection-control-max-connection-delay=#</code>
システム変数	<code>connection_control_max_connection_delay</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	2147483647
最小値	1000
最大値	2147483647
単位	milliseconds

`connection_control_failed_connections_threshold` がゼロより大きい場合の、失敗した接続試行に対するサーバーレスポンスの最大遅延 (ミリ秒)。

`connection_control_max_connection_delay` と他の接続制御システムおよびステータス変数との相互作用の詳細は、[セクション6.4.2.1「Connection-Control プラグインのインストール」](#) を参照してください。

- `connection_control_min_connection_delay`

コマンド行形式	<code>--connection-control-min-connection-delay=#</code>
システム変数	<code>connection_control_min_connection_delay</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1000
最小値	1000
最大値	2147483647
単位	milliseconds

`connection_control_failed_connections_threshold` がゼロより大きい場合の、失敗した接続試行に対するサーバーレスポンスの最小遅延 (ミリ秒)。

`connection_control_min_connection_delay` と他の接続制御システムおよびステータス変数との相互作用の詳細は、[セクション6.4.2.1「Connection-Control プラグインのインストール」](#) を参照してください。

接続制御ステータス変数

CONNECTION_CONTROL プラグインがインストールされている場合、次のステータス変数が公開されます:

- `Connection_control_delay_generated`

失敗した接続試行に対するレスポンスにサーバーが遅延を追加した回数。これにより、`connection_control_failed_connections_threshold` システム変数で定義されたしきい値に達するまでに発生する試行はカウントされません。

この変数は単純なカウンタを提供します。接続制御モニタリング情報の詳細は、[INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS](#) テーブルを確認してください。[セクション 26.53.1 「INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS テーブル」](#) を参照してください。

実行時に `connection_control_failed_connections_threshold` に値を割り当てると、`Connection_control_delay_generated` はゼロにリセットされます。

6.4.3 パスワード検証コンポーネント

`validate_password` コンポーネントは、アカウントパスワードを要求し、潜在的なパスワードの強度テストを有効にすることで、セキュリティを向上させます。このコンポーネントは、コンポーネント監視用のパスワードポリシーおよびステータス変数を構成できるシステム変数を公開します。

注記

MySQL 8.0 では、`validate_password` プラグインが `validate_password` コンポーネントとして再実装されました。(コンポーネントの一般情報は、[セクション 5.5 「MySQL のコンポーネント」](#) を参照してください。) 次の手順では、プラグインではなくコンポーネントの使用方法について説明します。プラグインフォームの `validate_password` を使用する手順は、[MySQL 5.7 Reference Manual の The Password Validation Plugin](#) を参照してください。

`validate_password` のプラグインフォームは引き続き使用できますが、非推奨になっています。将来のバージョンの MySQL で削除される予定です。プラグインを使用する MySQL インストールでは、かわりにコンポーネントの使用に移行する必要があります。[セクション 6.4.3.3 「パスワード検証コンポーネントへの移行」](#) を参照してください。

`validate_password` コンポーネントは、次の機能を実装します:

- クリアテキスト値として指定されたパスワードを割り当てる SQL ステートメントの場合、`validate_password` は現在のパスワードポリシーに対してパスワードをチェックし、弱い場合はパスワードを拒否します (ステートメントは `ER_NOT_VALID_PASSWORD` エラーを返します)。これは、`ALTER USER`、`CREATE USER` および `SET PASSWORD` ステートメントに適用されます。
- `CREATE USER` ステートメントの場合、`validate_password` ではパスワードを指定する必要があり、パスワードポリシーを満たしている必要があります。これは、アカウントが最初にロックされている場合でも当てはまります。アカウントを後でロック解除すると、ポリシーを満たすパスワードなしでアクセス可能になるためです。
- `validate_password` には、潜在的なパスワードの強度を評価する `VALIDATE_PASSWORD_STRENGTH()` SQL 関数が実装されています。この関数は、パスワード引数を取り、0 (弱い) から 100 (強い) までの整数を返します。

注記

アカウントパスワード (`ALTER USER`、`CREATE USER` および `SET PASSWORD`) を割り当てまたは変更するステートメントの場合、ここで説明する `validate_password` 機能は、MySQL に内部的に資格証明を格納する認証プラグインを使用するアカウントにのみ適用されます。MySQL 外部の資格証明システムに対して認証を実行するプラグインを使用するアカウントの場合、パスワード管理もそのシステムに対して外部で処理する必要があります。内部資格証明記憶域の詳細は、[セクション 6.2.15 「パスワード管理」](#) を参照してください。

前述の制限は、アカウントに直接影響しないため、`VALIDATE_PASSWORD_STRENGTH()` 関数の使用には適用されません。

例:

- `validate_password` は、次のステートメントでクリアテキストパスワードを確認します。デフォルトのパスワードポリシーではパスワードに最低 8 文字の長さが要求されるため、パスワードが弱いことからステートメントはエラーを生成します。

```
mysql> ALTER USER USER() IDENTIFIED BY 'abc';
ERROR 1819 (HY000): Your password does not satisfy the current
```



```
policy requirements
```

- 元のパスワード値はチェックに使用できないため、ハッシュ値として指定されたパスワードはチェックされません:

```
mysql> ALTER USER 'jeffrey'@'localhost'
  IDENTIFIED WITH mysql_native_password
  AS '*0D3CED9BEC10A777AEC23CCC353A8C08A633045E';
Query OK, 0 rows affected (0.01 sec)
```

- 現在のパスワードポリシーを満たすパスワードがアカウントに含まれていないため、アカウントが最初にロックされていても、このアカウント作成ステートメントは失敗します:

```
mysql> CREATE USER 'juanita'@'localhost' ACCOUNT LOCK;
ERROR 1819 (HY000): Your password does not satisfy the current
policy requirements
```

- パスワードを確認するには、`VALIDATE_PASSWORD_STRENGTH()` 関数を使用します:

```
mysql> SELECT VALIDATE_PASSWORD_STRENGTH('weak');
+-----+
| VALIDATE_PASSWORD_STRENGTH('weak') |
+-----+
|                25 |
+-----+
mysql> SELECT VALIDATE_PASSWORD_STRENGTH('lessweak$_@123');
+-----+
| VALIDATE_PASSWORD_STRENGTH('lessweak$_@123') |
+-----+
|                50 |
+-----+
mysql> SELECT VALIDATE_PASSWORD_STRENGTH('N0Tweak$_@123!');
+-----+
| VALIDATE_PASSWORD_STRENGTH('N0Tweak$_@123!') |
+-----+
|                100 |
+-----+
```

パスワードチェックを構成するには、`validate_password.xxx` という形式の名前を持つシステム変数を変更します。これらは、パスワードポリシーを制御するパラメータです。 [セクション6.4.3.2「パスワード検証オプションおよび変数」](#)を参照してください。

`validate_password` がインストールされていない場合、`validate_password.xxx` システム変数は使用できず、ステートメントのパスワードはチェックされず、`VALIDATE_PASSWORD_STRENGTH()` 関数は常に 0 を返します。たとえば、プラグインがインストールされていない場合、アカウントには 8 文字未満のパスワードを割り当てることも、パスワードをまったく割り当てないこともできます。

`validate_password` がインストールされていると仮定すると、3 レベルのパスワードチェックが実装されます: **LOW**、**MEDIUM** および **STRONG**。デフォルトは **MEDIUM** です。これを変更するには、`validate_password.policy` の値を変更します。これらのポリシーにより、実装されるパスワードテストはますます厳密になります。次の説明では、適切なシステム変数を変更して変更できるデフォルトのパラメータ値について説明します。

- LOW** ポリシーは、パスワードの長さのみテストします。パスワードは少なくとも 8 文字の長さでなければなりません。この長さを変更するには、`validate_password.length` を変更します。
- MEDIUM** ポリシーでは、パスワードに少なくとも 1 つの数字、1 つの小文字、1 つの大文字および 1 つの特殊文字 (英数字以外) を含める必要があるという条件が追加されます。これらの値を変更するには、`validate_password.number_count`、`validate_password.mixed_case_count` および `validate_password.special_char_count` を変更します。
- STRONG** ポリシーは、パスワードの 4 文字以上の部分文字列が、(辞書ファイルが指定された場合に) 辞書ファイル内の単語と一致してはならないという条件を追加します。デクシヨナリファイルを指定するには、`validate_password.dictionary_file` を変更します。

また、`validate_password` では、現在のセッションの有効なユーザーアカウントのユーザー名部分と一致するパスワードを転送または逆方向に拒否する機能がサポートされています。この機能を制御するために、`validate_password` では、デフォルトで有効になっている `validate_password.check_user_name` システム変数が公開されます。

6.4.3.1 パスワード検証コンポーネントのインストールおよびアンインストール

このセクションでは、`validate_password` パスワード検証コンポーネントをインストールおよびアンインストールする方法について説明します。コンポーネントのインストールおよびアンインストールの一般情報は、[セクション 5.5「MySQL のコンポーネント」](#) を参照してください。

注記

「MySQL Yum リポジトリ」、「MySQL SLES リポジトリ」または [RPM packages provided by Oracle](#) を使用して MySQL 8.0 をインストールする場合、MySQL Server を初めて起動した後、`validate_password` コンポーネントはデフォルトで有効になります。

Yum または RPM パッケージを使用して 5.7 から MySQL 8.0 にアップグレードすると、`validate_password` プラグインはそのまま残ります。`validate_password` プラグインから `validate_password` コンポーネントへの移行を行うには、[セクション 6.4.3.3「パスワード検証コンポーネントへの移行」](#) を参照してください。

サーバーで使用できるようにするには、コンポーネントライブラリファイルが MySQL プラグインディレクトリ (`plugin_dir` システム変数で指定されたディレクトリ) にある必要があります。必要に応じて、サーバーの起動時に `plugin_dir` の値を設定してプラグインディレクトリの場所を構成します。

`validate_password` コンポーネントをインストールするには、次のステートメントを使用します:

```
INSTALL COMPONENT 'file://component_validate_password';
```

コンポーネントのインストールは、サーバーの起動ごとに実行する必要のない一度限りの操作です。`INSTALL COMPONENT` によってコンポーネントがロードされ、`mysql.component` システムテーブルにも登録されて、後続のサーバー起動時にロードされます。

`validate_password` コンポーネントをアンインストールするには、次のステートメントを使用します:

```
UNINSTALL COMPONENT 'file://component_validate_password';
```

`UNINSTALL COMPONENT` はコンポーネントをアンロードし、`mysql.component` システムテーブルから登録解除して、後続のサーバー起動時にロードされないようにします。

6.4.3.2 パスワード検証オプションおよび変数

このセクションでは、操作を構成および監視できるようにするために `validate_password` が提供するシステム変数およびステータス変数について説明します。

- [パスワード検証コンポーネントのシステム変数](#)
- [パスワード検証コンポーネントのステータス変数](#)
- [パスワード検証プラグインオプション](#)
- [パスワード検証プラグインシステム変数](#)
- [パスワード検証プラグインステータス変数](#)

パスワード検証コンポーネントのシステム変数

`validate_password` コンポーネントが有効になっている場合、パスワードチェックの構成を有効にするいくつかのシステム変数が公開されます:

```
mysql> SHOW VARIABLES LIKE 'validate_password.%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| validate_password.check_user_name | ON |
| validate_password.dictionary_file | |
| validate_password.length | 8 |
| validate_password.mixed_case_count | 1 |
| validate_password.number_count | 1 |
| validate_password.policy | MEDIUM |
```

```
validate_password.special_char_count | 1 |
+-----+-----+
```

パスワードのチェック方法を変更するには、サーバーの起動時または実行時にこれらのシステム変数を設定します。次のリストは、各変数の意味を説明したものです。

- [validate_password.check_user_name](#)

コマンド行形式	--validate-password.check-user-name[={OFF ON}]
システム変数	validate_password.check_user_name
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

[validate_password](#) で、現在のセッションの有効なユーザーアカウントのユーザー名部分とパスワードを比較し、一致する場合は拒否するかどうか。この変数は、[validate_password](#) がインストールされていないかぎり使用できません。

デフォルトでは、[validate_password.check_user_name](#) は有効です。この変数は、[validate_password.policy](#) の値に関係なく、ユーザー名の一致を制御します。

[validate_password.check_user_name](#) を有効にすると、次の効果があります:

- チェックは、[validate_password](#) が起動されるすべてのコンテキストで行われます。これには、[ALTER USER](#) や [SET PASSWORD](#) などのステートメントを使用した現在のユーザーパスワードの変更、[VALIDATE_PASSWORD_STRENGTH\(\)](#) などの関数の起動が含まれます。
 - 比較に使用されるユーザー名は、現行セッションの [USER\(\)](#) および [CURRENT_USER\(\)](#) 関数の値から取得されます。つまり、別のユーザーパスワードを設定するための十分な権限を持つユーザーは、そのユーザー名にパスワードを設定でき、そのユーザーパスワードをステートメントを実行するユーザーの名前に設定できません。たとえば、`'root'@'localhost'` では `'jeffrey'@'localhost'` のパスワードを `'jeffrey'` に設定できますが、パスワードを `'root'` に設定することはできません。
 - [USER\(\)](#) および [CURRENT_USER\(\)](#) 関数の値のユーザー名部分のみが使用され、ホスト名部分は使用されません。ユーザー名が空の場合、比較は行われません。
 - パスワードがユーザー名と同じであるか、その逆の場合、一致が発生し、パスワードは拒否されます。
 - ユーザー名の照合では、大文字と小文字が区別されます。パスワードとユーザー名の値は、バイト単位でバイナリ文字列として比較されます。
 - パスワードがユーザー名と一致する場合、他の [validate_password](#) システム変数の設定方法に関係なく、[VALIDATE_PASSWORD_STRENGTH\(\)](#) は 0 を返します。
- [validate_password.dictionary_file](#)

コマンド行形式	--validate-password.dictionary-file=file_name
システム変数	validate_password.dictionary_file
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	ファイル名

[validate_password](#) がパスワードのチェックに使用するディクショナリファイルのパス名。この変数は、[validate_password](#) がインストールされていないかぎり使用できません。

デフォルトでは、この変数は空の値を持ち、辞書検査は実行されません。ディクショナリチェックを実行するには、変数値を空にしないでください。ファイル名が相対パスとして指定された場合、サーバーのデータディレクトリを基準として解釈されます。ファイルの内容は小文字で、1行につき1語にする必要があります。内容は、`utf8`の文字セットを持つものとして処理されます。許可される最大のファイルサイズは1Mバイトです。

パスワードチェック中にディクショナリファイルを使用するには、パスワードポリシーを2 (**STRONG**) に設定する必要があります。`validate_password.policy` システム変数の説明を参照してください。これが `true` である場合、長さが4から100までのパスワードの各部分文字列が辞書ファイル内の単語と比較されます。いずれかが一致すると、パスワードが拒否されます。比較では大文字と小文字は区別されません。

`VALIDATE_PASSWORD_STRENGTH()` の場合、パスワードは **STRONG** を含むすべてのポリシーに対してチェックされるため、強度評価には `validate_password.policy` 値に関係なくディクショナリチェックが含まれます。

`validate_password.dictionary_file` は実行時に設定でき、値を割り当てると、サーバーを再起動せずに名前付きファイルが読み取られます。

- `validate_password.length`

コマンド行形式	<code>--validate-password.length=#</code>
システム変数	<code>validate_password.length</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	8
最小値	0

`validate_password` がパスワードを必要とする最小文字数。この変数は、`validate_password` がインストールされていないかぎり使用できません。

`validate_password.length` の最小値は、他のいくつかの関連するシステム変数の関数です。この式の値より小さい値は設定できません:

```
validate_password.number_count
+ validate_password.special_char_count
+ (2 * validate_password.mixed_case_count)
```

`validate_password` は、前述の制約のために `validate_password.length` の値を調整すると、エラーログにメッセージを書き込みます。

- `validate_password.mixed_case_count`

コマンド行形式	<code>--validate-password.mixed-case-count=#</code>
システム変数	<code>validate_password.mixed_case_count</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1

最小値	0
-----	---

パスワードポリシーが **MEDIUM** 以上の場合に、`validate_password` がパスワードを必要とする小文字と大文字の最小数。この変数は、`validate_password` がインストールされていないかぎり使用できません。

特定の `validate_password.mixed_case_count` 値について、パスワードには小文字が多数含まれ、大文字が多数含まれている必要があります。

- `validate_password.number_count`

コマンド行形式	<code>--validate-password.number-count=#</code>
システム変数	<code>validate_password.number_count</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1
最小値	0

パスワードポリシーが **MEDIUM** 以上の場合に、`validate_password` がパスワードを必要とする数字 (数字) の最小数。この変数は、`validate_password` がインストールされていないかぎり使用できません。

- `validate_password.policy`

コマンド行形式	<code>--validate-password.policy=value</code>
システム変数	<code>validate_password.policy</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	1
有効な値	0 1 2

`validate_password` によって強制されるパスワードポリシー。この変数は、`validate_password` がインストールされていないかぎり使用できません。

`validate_password.policy` は、`validate_password.check_user_name` によって独立して制御されるユーザー名に対するパスワードのチェックを除き、`validate_password` が他のポリシー設定システム変数を使用する方法に影響しません。

`validate_password.policy` 値は、数値 0、1、2、または対応するシンボリック値 **LOW**、**MEDIUM**、**STRONG** を使用して指定できます。次の表では、それぞれのポリシーに対して実施されるテストについて説明します。長さテストの場合、必要な長さは `validate_password.length` システム変数の値です。同様に、他のテストに必要な値は、他の `validate_password.xxx` 変数によって指定されます。

ポリシー	実施されるテスト
0 または LOW	長さ
1 または MEDIUM	長さ。数値、小文字、大文字、および特殊文字

ポリシー	実施されるテスト
2 または STRONG	長さ。数値、小文字、大文字、および特殊文字。辞書ファイル

- [validate_password.special_char_count](#)

コマンド行形式	<code>--validate-password.special-char-count=#</code>
システム変数	validate_password.special_char_count
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1
最小値	0

パスワードポリシーが **MEDIUM** 以上の場合に、[validate_password](#) がパスワードを必要とする英数字以外の最小文字数。この変数は、[validate_password](#) がインストールされていないかぎり使用できません。

パスワード検証コンポーネントのステータス変数

[validate_password](#) コンポーネントが有効になっている場合、操作情報を提供するステータス変数が公開されます:

```
mysql> SHOW STATUS LIKE 'validate_password.%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| validate_password.dictionary_file_last_parsed | 2019-10-03 08:33:49 |
| validate_password.dictionary_file_words_count | 1902 |
+-----+-----+
```

次のリストに、各ステータス変数の意味を示します。

- [validate_password.dictionary_file_last_parsed](#)

ディクショナリファイルが最後に解析された日時。この変数は、[validate_password](#) がインストールされていないかぎり使用できません。

- [validate_password.dictionary_file_words_count](#)

辞書ファイルから読み取られた単語の数。この変数は、[validate_password](#) がインストールされていないかぎり使用できません。

パスワード検証プラグインオプション

注記

MySQL 8.0 では、[validate_password](#) プラグインが [validate_password](#) コンポーネントとして再実装されました。[validate_password](#) プラグインは非推奨です。将来のバージョンの MySQL で削除される予定です。したがって、そのオプションも非推奨になり、それらも削除されることが予想されます。プラグインを使用する MySQL インストールでは、かわりにコンポーネントの使用に移行する必要があります。[セクション6.4.3.3「パスワード検証コンポーネントへの移行」](#)を参照してください。

[validate_password](#) プラグインのアクティブ化を制御するには、このオプションを使用します:

- `--validate-password[=value]`

コマンド行形式	<code>--validate-password[=value]</code>
---------	--

型	列挙
デフォルト値	ON
有効な値	ON OFF FORCE FORCE_PLUS_PERMANENT

このオプションは、サーバーが起動時に非推奨の `validate_password` プラグインをロードする方法を制御します。値は [セクション5.6.1「プラグインのインストールおよびアンインストール」](#) に記載されているような、プラグインロードオプション用に指定可能ないずれかの値とする必要があります。たとえば、`--validate-password=FORCE_PLUS_PERMANENT` は、起動時にプラグインをロードし、サーバーの実行中にプラグインが削除されないようにサーバーに指示します。

このオプションは、`validate_password` プラグインが以前に `INSTALL PLUGIN` に登録されているか、`--plugin-load-add` にロードされている場合にのみ使用できます。 [セクション6.4.3.1「パスワード検証コンポーネントのインストールおよびアンインストール」](#) を参照してください。

パスワード検証プラグインシステム変数

注記

MySQL 8.0 では、`validate_password` プラグインが `validate_password` コンポーネントとして再実装されました。`validate_password` プラグインは非推奨です。将来のバージョンの MySQL で削除される予定です。したがって、そのシステム変数も非推奨になるため、これらも削除する必要があります。かわりに、`validate_password` コンポーネントの対応するシステム変数を使用してください。[パスワード検証コンポーネントのシステム変数](#) を参照してください。プラグインを使用する MySQL インストールでは、かわりにコンポーネントの使用に移行する必要があります。 [セクション6.4.3.3「パスワード検証コンポーネントへの移行」](#) を参照してください。

- [validate_password_check_user_name](#)

コマンド行形式	<code>--validate-password-check-user-name[={OFF ON}]</code>
システム変数	<code>validate_password_check_user_name</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

この `validate_password` プラグインシステム変数は非推奨です。将来のバージョンの MySQL で削除される予定です。かわりに、`validate_password` コンポーネントの対応する `validate_password.check_user_name` システム変数を使用してください。

- [validate_password_dictionary_file](#)

コマンド行形式	<code>--validate-password-dictionary-file=file_name</code>
システム変数	<code>validate_password_dictionary_file</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	ファイル名

この `validate_password` プラグインシステム変数は非推奨です。将来のバージョンの MySQL で削除される予定です。かわりに、`validate_password` コンポーネントの対応する `validate_password.dictionary_file` システム変数を使用してください。

- `validate_password_length`

コマンド行形式	<code>--validate-password-length=#</code>
システム変数	<code>validate_password_length</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	8
最小値	0

この `validate_password` プラグインシステム変数は非推奨です。将来のバージョンの MySQL で削除される予定です。かわりに、`validate_password` コンポーネントの対応する `validate_password.length` システム変数を使用してください。

- `validate_password_mixed_case_count`

コマンド行形式	<code>--validate-password-mixed-case-count=#</code>
システム変数	<code>validate_password_mixed_case_count</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1
最小値	0

この `validate_password` プラグインシステム変数は非推奨です。将来のバージョンの MySQL で削除される予定です。かわりに、`validate_password` コンポーネントの対応する `validate_password.mixed_case_count` システム変数を使用してください。

- `validate_password_number_count`

コマンド行形式	<code>--validate-password-number-count=#</code>
システム変数	<code>validate_password_number_count</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1
最小値	0

この `validate_password` プラグインシステム変数は非推奨です。将来のバージョンの MySQL で削除される予定です。かわりに、`validate_password` コンポーネントの対応する `validate_password.number_count` システム変数を使用してください。

- [validate_password_policy](#)

コマンド行形式	<code>--validate-password-policy=value</code>
システム変数	validate_password_policy
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	1
有効な値	0 1 2

この [validate_password](#) プラグインシステム変数は非推奨です。将来のバージョンの MySQL で削除される予定です。かわりに、[validate_password](#) コンポーネントの対応する [validate_password.policy](#) システム変数を使用してください。

- [validate_password_special_char_count](#)

コマンド行形式	<code>--validate-password-special-char-count=#</code>
システム変数	validate_password_special_char_count
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1
最小値	0

この [validate_password](#) プラグインシステム変数は非推奨です。将来のバージョンの MySQL で削除される予定です。かわりに、[validate_password](#) コンポーネントの対応する [validate_password.special_char_count](#) システム変数を使用してください。

パスワード検証プラグインステータス変数

注記

MySQL 8.0 では、[validate_password](#) プラグインが [validate_password](#) コンポーネントとして再実装されました。[validate_password](#) プラグインは非推奨です。将来のバージョンの MySQL で削除される予定です。その結果、そのステータス変数も非推奨になり、削除されることが予想されます。[validate_password](#) コンポーネントの対応するステータス変数を使用します。[パスワード検証コンポーネントのステータス変数](#) を参照してください。プラグインを使用する MySQL インストールでは、かわりにコンポーネントの使用に移行する必要があります。[セクション6.4.3.3「パスワード検証コンポーネントへの移行」](#) を参照してください。

- [validate_password_dictionary_file_last_parsed](#)

この [validate_password](#) プラグインステータス変数は非推奨です。将来のバージョンの MySQL で削除される予定です。かわりに、[validate_password](#) コンポーネントの対応する [validate_password.dictionary_file_last_parsed](#) ステータス変数を使用してください。

- [validate_password_dictionary_file_words_count](#)

この `validate_password` プラグインステータス変数は非推奨です。将来のバージョンの MySQL で削除される予定です。かわりに、`validate_password` コンポーネントの対応する `validate_password.dictionary_file_words_count` ステータス変数を使用してください。

6.4.3.3 パスワード検証コンポーネントへの移行

注記

MySQL 8.0 では、`validate_password` プラグインが `validate_password` コンポーネントとして再実装されました。`validate_password` プラグインは非推奨です。将来のバージョンの MySQL で削除される予定です。

`validate_password` プラグインを現在使用している MySQL インストールでは、かわりに `validate_password` コンポーネントの使用に移行する必要があります。これを行うには、次の手順を使用します。この手順では、プラグインをアンインストールする前にコンポーネントをインストールして、パスワード検証が行われない時間ウィンドウが表示されないようにします。(コンポーネントとプラグインは同時にインストールできます。この場合、サーバーはコンポーネントを使用しようとし、コンポーネントが使用できない場合はプラグインにフォールバックします。)

1. `validate_password` コンポーネントをインストールします:

```
INSTALL COMPONENT 'file://component_validate_password';
```

2. `validate_password` コンポーネントをテストして、予想どおりに機能することを確認します。`validate_password.xxx` システム変数を設定する必要がある場合は、`SET GLOBAL` を使用して実行時に設定できます。(必要なオプションファイルの変更は、次のステップで実行します。)
3. プラグインシステムおよびステータス変数への参照を、対応するコンポーネントシステムおよびステータス変数を参照するように調整します。以前に、次のようなオプションファイルを使用して起動時にプラグインを構成したとします:

```
[mysqld]
validate_password=FORCE_PLUS_PERMANENT
validate_password_dictionary_file=/usr/share/dict/words
validate_password_length=10
validate_password_number_count=2
```

これらの設定はプラグインに適していますが、コンポーネントに適用するように変更する必要があります。オプションファイルを調整するには、`--validate-password` オプションを省略し(コンポーネントではなくプラグインにのみ適用)、プラグインに適したドットなしの名前からコンポーネントに適したドット付きの名前にシステム変数参照を変更します:

```
[mysqld]
validate_password.dictionary_file=/usr/share/dict/words
validate_password.length=10
validate_password.number_count=2
```

実行時に `validate_password` プラグインシステムおよびステータス変数を参照するアプリケーションにも同様の調整が必要です。ドットなしプラグイン変数名を、対応するドット付きコンポーネント変数名に変更します。

4. `validate_password` プラグインをアンインストールします:

```
UNINSTALL PLUGIN validate_password;
```

`--plugin-load` または `--plugin-load-add` オプションを使用してサーバーの起動時に `validate_password` プラグインがロードされる場合は、サーバーの起動手順でそのオプションを省略します。たとえば、オプションがサーバーオプションファイルにリストされている場合は、ファイルから削除します。

5. サーバーを再起動します。

6.4.4 MySQL キーリング

MySQL Server は、内部サーバーコンポーネントおよびプラグインが機密情報を安全に格納して後で取得できるようにするキーリングをサポートしています。実装は、次の要素で構成されます:

- バッキングストアを管理したり、ストレージバックエンドと通信したりするキーリングプラグイン。次のキーリングプラグインを使用できます:
 - `keyring_file` は、キーリングデータをサーバーホストのローカルファイルに格納します。このプラグインは、MySQL Community Edition および MySQL Enterprise Edition のディストリビューションで使用できます。 [セクション6.4.4.2「keyring_file ファイルベースプラグインの使用」](#)を参照してください。
 - `keyring_encrypted_file` は、サーバーホストに対してローカルな暗号化されたファイルにキーリングデータを格納します。このプラグインは、MySQL Enterprise Edition ディストリビューションで使用できます。 [セクション6.4.4.3「keyring_encrypted_file キーリングプラグインの使用」](#)を参照してください。
 - `keyring_okv` は、Oracle Key Vault や Gemalto SafeNet KeySecure Appliance などの KMIP 互換のバックエンドキーリングストレージ製品で使用する KMIP 1.1 プラグインです。このプラグインは、MySQL Enterprise Edition ディストリビューションで使用できます。 [セクション6.4.4.4「keyring_okv KMIP プラグインの使用」](#)を参照してください。
 - `keyring_aws` は、キーの生成のために Amazon Web Services Key Management Service と通信し、キーの格納にローカルファイルを使用します。このプラグインは、MySQL Enterprise Edition ディストリビューションで使用できます。 [セクション6.4.4.5「keyring_aws Amazon Web Services キーリングプラグインの使用」](#)を参照してください。
 - `keyring_hashicorp` は、バックエンドストレージのために HashiCorp Vault と通信します。このプラグインは、MySQL 8.0.18 の時点の MySQL Enterprise Edition ディストリビューションで使用できます。 [セクション6.4.4.6「HashiCorp Vault キーリングプラグインの使用」](#)を参照してください。
 - `keyring_oci` は、バックエンドストレージのために Oracle Cloud Infrastructure Vault と通信します。このプラグインは、MySQL 8.0.22 の時点の MySQL Enterprise Edition ディストリビューションで使用できます。 [セクション6.4.4.7「Oracle Cloud Infrastructure Vault キーリングプラグインの使用」](#)を参照してください。
- 主要な移行機能。MySQL サーバーの操作モードでは、基礎となるキーリングキーストア間でキーを移行できるため、DBA は MySQL インストールにあるキーリングプラグインから別のキーリングプラグインに切り替えることができます。 [セクション6.4.4.9「キーリングキーストア間のキーの移行」](#)を参照してください。
- 鍵リング鍵管理用の鍵リングサービスインターフェース。次の2つのレベルでアクセスできます:
 - SQL インターフェース: SQL ステートメントで、 [セクション6.4.4.10「汎用キーリングキー管理関数」](#) で説明されているユーザー定義関数 (UDF) をコールします。
 - C インターフェース: C 言語コードでは、 [セクション5.6.8.2「キーリングサービス」](#) で説明されているキーリングサービス関数をコールします。
- キーメタデータアクセス。MySQL 8.0.16 以降では、パフォーマンススキーマ `keyring_keys` テーブルは鍵リング内の鍵のメタデータを公開します。キーメタデータには、キー ID、キー所有者およびバックエンドキー ID が含まれます。 `keyring_keys` テーブルでは、キーの内容などの機密キーリングデータは公開されません。 [セクション27.12.19.6「keyring_keys テーブル」](#)を参照してください。

警告

暗号化キー管理用の `keyring_file` および `keyring_encrypted_file` プラグインは、規制コンプライアンスソリューションとしては意図されていません。PCI、FIPS などのセキュリティ標準では、キーホルトまたはハードウェアセキュリティモジュール (HSM) 内の暗号化キーを保護、管理および保護するためにキー管理システムを使用する必要があります。

MySQL 内では、キーリングの使用方法は次のとおりです:

- InnoDB ストレージエンジンは、鍵リングを使用して、テーブルスペース暗号化用の鍵を格納します。InnoDB は、サポートされている任意のキーリングプラグインを使用できます。 [セクション15.13「InnoDB 保存データ暗号化」](#)を参照してください。
- MySQL Enterprise Audit では、キーリングを使用して監査ログファイルの暗号化パスワードを格納します。監査ログプラグインは、サポートされている任意のキーリングプラグインを使用できます。 [監査ログファイルの暗号化](#)を参照してください。

- バイナリログとリレーログの管理では、ログファイルの鍵リングベースの暗号化がサポートされています。ログファイルの暗号化を有効にすると、バイナリログファイルとリレーログファイルのパスワードを暗号化するために使用される鍵が鍵リングに格納されます。この機能では、サポートされている任意のキーリングプラグインを使用できます。 [セクション17.3.2「バイナリログファイルとリレーログファイルの暗号化」](#)を参照してください。

鍵リングの一般的なインストール手順については、 [セクション6.4.4.1「キーリングプラグインのインストール」](#)を参照してください。特定のキーリングプラグインに固有のインストールおよび構成情報については、そのプラグインについて説明しているセクションを参照してください。

キーリング UDF の使用の詳細は、 [セクション6.4.4.10「汎用キーリングキー管理関数」](#)を参照してください。

キープラグインおよび UDF は、キーリングへのコンポーネントのインタフェースを提供するキーリングサービスにアクセスします。キーリングプラグインサービスへのアクセスおよびキーリングプラグインの記述については、 [セクション5.6.8.2「キーリングサービス」](#) および [Writing Keyring Plugins](#)を参照してください。

6.4.4.1 キーリングプラグインのインストール

鍵リングサービスコンシューマには、鍵リングプラグインがインストールされている必要があります。MySQL には、次のプラグインの選択肢があります:

- [keyring_file](#): キーリングデータをサーバーホストのローカルファイルに格納します。すべての MySQL ディストリビューションで使用できます。
- [keyring_encrypted_file](#): キーリングデータをサーバーホストに対してローカルな暗号化されたファイルに格納します。MySQL Enterprise Edition ディストリビューションで使用できます。
- [keyring_okv](#): Oracle Key Vault、Gemalto SafeNet KeySecure Appliance などの KMIP 互換のバックエンドキーリングストレージ製品を使用します。MySQL Enterprise Edition ディストリビューションで使用できます。
- [keyring_aws](#): キー生成のバックエンドとして Amazon Web Services Key Management Service と通信し、キーの格納にローカルファイルを使用します。MySQL Enterprise Edition ディストリビューションで使用できます。
- [keyring_hashicorp](#): バックエンドストレージのために HashiCorp Vault と通信します。MySQL Enterprise Edition ディストリビューションで使用できます。
- [keyring_oci](#) の場合: [セクション6.4.4.7「Oracle Cloud Infrastructure Vault キーリングプラグインの使用」](#)

このセクションでは、選択したキーリングプラグインをインストールする方法について説明します。プラグインのインストールについての一般的な情報は、 [セクション5.6.1「プラグインのインストールおよびアンインストール」](#)を参照してください。

キーリングプラグインとともにキーリングユーザー定義関数 (UDF) を使用する場合は、 [セクション6.4.4.10「汎用キーリングキー管理関数」](#)の手順に従って、プラグインのインストール後に UDF をインストールします。

サーバーで使用できるようにするには、プラグインライブラリファイルを MySQL プラグインディレクトリ (`plugin_dir` システム変数で指定されたディレクトリ) に配置する必要があります。必要に応じて、サーバーの起動時に `plugin_dir` の値を設定してプラグインディレクトリの場所を構成します。

各キーリングプラグインのインストールは似ています。次の手順では、[keyring_file](#) を使用します。別のキーリングプラグインのユーザーは、[keyring_file](#) をその名前に置き換えることができます。

[keyring_file](#) プラグインライブラリファイルのベース名は [keyring_file](#) です。ファイル名の接尾辞は、プラットフォームごとに異なります (たとえば、`.so` for Unix and Unix-like systems, `.dll` for Windows)。

注記

キーリングプラグインは一度に 1 つだけイネーブルにする必要があります。複数のキーリングプラグインの有効化はサポートされていないため、予想どおりに結果が得られないことがあります。

キーリングプラグインは、コンポーネントが独自の初期化中に必要に応じてアクセスできるように、サーバーの起動シーケンスの最早段階でロードする必要があります。たとえば、InnoDB ストレージエンジンはテーブルスペースの暗号化に鍵リングを使用するため、InnoDB を初期化する前に鍵リングプラグインをロードして使用可能にする必要があります。

プラグインをロードするには、`--early-plugin-load` オプションを使用して、プラグインを含むプラグインライブラリファイルに名前を付けます。たとえば、プラグインライブラリファイルの接尾辞が `.so` であるプラットフォームでは、サーバーの `my.cnf` ファイルで次の行を使用し、必要に応じてプラットフォームの `.so` 接尾辞を調整します:

```
[mysqld]
early-plugin-load=keyring_file.so
```

サーバーを起動する前に、選択したキーリングプラグインのノートをチェックして、追加の構成が許可されているか必要かを確認します:

- `keyring_file`: [セクション6.4.4.2「keyring_file ファイルベースプラグインの使用」](#)。
- `keyring_encrypted_file`: [セクション6.4.4.3「keyring_encrypted_file キーリングプラグインの使用」](#)。
- `keyring_okv`: [セクション6.4.4.4「keyring_okv KMIP プラグインの使用」](#)。
- `keyring_aws`: [セクション6.4.4.5「keyring_aws Amazon Web Services キーリングプラグインの使用」](#)。
- `keyring_hashicorp`: [セクション6.4.4.6「HashiCorp Vault キーリングプラグインの使用」](#)。

プラグイン固有の構成を実行した後、プラグインのインストールを検証します。MySQL サーバーを実行した状態で、`INFORMATION_SCHEMA.PLUGINS` テーブルを調べるか、`SHOW PLUGINS` ステートメントを使用します ([セクション5.6.2「サーバープラグイン情報の取得」](#) を参照)。例:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
        FROM INFORMATION_SCHEMA.PLUGINS
        WHERE PLUGIN_NAME LIKE 'keyring%';
+-----+-----+
| PLUGIN_NAME | PLUGIN_STATUS |
+-----+-----+
| keyring_file | ACTIVE        |
+-----+-----+
```

プラグインの初期化に失敗した場合は、サーバーエラーログで診断メッセージを確認してください。

プラグインは、`--plugin-load` や `--plugin-load-add` オプション、`INSTALL PLUGIN` ステートメントなど、`--early-plugin-load` 以外の方法でロードできます。ただし、これらのメソッドを使用してロードされたキーリングプラグインは、`InnoDB` などの特定のコンポーネントのサーバー起動シーケンスでは遅すぎる可能性があります:

- `--plugin-load` または `--plugin-load-add` を使用したプラグインのロードは、`InnoDB` の初期化後に行われます。
- `INSTALL PLUGIN` を使用してインストールされたプラグインは、`mysql.plugin` システムテーブルに登録され、その後のサーバーの再起動のために自動的にロードされます。ただし、`mysql.plugin` は `InnoDB` テーブルであるため、そこに指定されたプラグインは、`InnoDB` の初期化後にのみ起動時にロードできます。

コンポーネントが鍵リングサービスにアクセスしようとしたときに鍵リングプラグインを使用できない場合、そのコンポーネントはそのサービスを使用できません。その結果、コンポーネントの初期化に失敗したり、機能が制限されて初期化されたりする可能性があります。たとえば、`InnoDB` は、初期化時に暗号化されたテーブルスペースがあることを検出すると、キーリングへのアクセスを試みます。キーリングが使用できない場合、`InnoDB` は暗号化されていないテーブルスペースにのみアクセスできます。`InnoDB` が暗号化されたテーブルスペースにも確実にアクセスできるようにするには、`--early-plugin-load` を使用してキープラグインをロードします。

6.4.4.2 keyring_file ファイルベースプラグインの使用

`keyring_file` キーリングプラグインは、サーバーホストに対してローカルなファイルに鍵リングデータを格納します。

警告

暗号化キー管理用の `keyring_file` プラグインは、規制コンプライアンスソリューションとしては意図されていません。PCI、FIPS などのセキュリティ標準では、キーボルトまたはハードウェアセキュリティモジュール (HSM) 内の暗号化キーを保護、管理および保護するためにキー管理システムを使用する必要があります。

`keyring_file` プラグインをインストールするには、[セクション6.4.4.1「キーリングプラグインのインストール」](#) にある一般的なキーリングのインストール手順と、ここにある `keyring_file` に固有の構成情報を使用します。

サーバーの起動プロセス中に使用できるようにするには、`--early-plugin-load` オプションを使用して `keyring_file` をロードする必要があります。 `keyring_file_data` システム変数は、オプションで、`keyring_file` プラグインがデータストレージに使用するファイルの場所を構成します。 デフォルト値はプラットフォーム固有です。 ファイルの場所を明示的に構成するには、起動時に変数値を設定します。 たとえば、サーバー `my.cnf` ファイルで次の行を使用して、プラットフォームの `.so` 接尾辞とファイルの場所を必要に応じて調整します:

```
[mysqld]
early-plugin-load=keyring_file.so
keyring_file_data=/usr/local/mysql/mysql-keyring/keyring
```

キーリング操作はトランザクションです: `keyring_file` プラグインは、書き込み操作中にバックアップファイルを使用して、操作が失敗した場合に元のファイルにロールバックできるようにします。 バックアップファイルの名前は、接尾辞が `.backup` の `keyring_file_data` システム変数の値と同じです。

`keyring_file_data` の詳細は、[セクション6.4.4.13「キーリングシステム変数」](#) を参照してください。

正しい鍵リングストレージファイルが存在する場合にのみ鍵が確実にフラッシュされるように、`keyring_file` は鍵リングの SHA-256 チェックサムをファイルに格納します。 ファイルを更新する前に、プラグインは、予想されるチェックサムが含まれていることを確認します。

`keyring_file` プラグインは、標準の MySQL キーリングサービスインタフェースを構成する関数をサポートしています。 これらの関数によって実行されるキーリング操作には、次の2つのレベルでアクセスできます:

- SQL インタフェース: SQL ステートメントで、[セクション6.4.4.10「汎用キーリングキー管理関数」](#) で説明されているユーザー定義関数 (UDF) をコールします。
- C インタフェース: C 言語コードでは、[セクション5.6.8.2「キーリングサービス」](#) で説明されているキーリングサービス関数をコールします。

例 (UDF を使用):

```
SELECT keyring_key_generate('MyKey', 'AES', 32);
SELECT keyring_key_remove('MyKey');
```

`keyring_file` で許可されるキータイプの詳細は、[セクション6.4.4.8「サポートされているキーリングキーのタイプと長さ」](#) を参照してください。

6.4.4.3 `keyring_encrypted_file` キーリングプラグインの使用

注記

`keyring_encrypted_file` プラグインは、商用製品である MySQL Enterprise Edition に含まれている拡張機能です。 商用製品の詳細は、<https://www.mysql.com/products/> を参照してください。

`keyring_encrypted_file` キーリングプラグインは、サーバーホストに対してローカルな暗号化ファイルに鍵リングデータを格納します。

警告

暗号化キー管理用の `keyring_encrypted_file` プラグインは、規制コンプライアンスソリューションとしては意図されていません。 PCI、FIPS などのセキュリティ標準では、キーポルトまたはハードウェアセキュリティモジュール (HSM) 内の暗号化キーを保護、管理および保護するためにキー管理システムを使用する必要があります。

`keyring_encrypted_file` プラグインをインストールするには、[セクション6.4.4.1「キーリングプラグインのインストール」](#) にある一般的なキーリングのインストール手順と、ここにある `keyring_encrypted_file` に固有の構成情報を使用します。

サーバーの起動プロセス中に使用できるようにするには、`--early-plugin-load` オプションを使用して `keyring_encrypted_file` をロードする必要があります。 キーリングデータファイルを暗号化するためのパスワードを指定するには、`keyring_encrypted_file_password` システム変数を設定します。 (パスワードは必須です。サーバーの起動時に指定しない場合、`keyring_encrypted_file` の初期化は失敗します。) `keyring_encrypted_file_data` システム変数は、オプションで、`keyring_encrypted_file` プラグインがデータストレージに使用するファイルの場所を構成します。 デフォルト値はプラットフォーム固有です。 ファイルの場所を明示的に構成するには、起動時に変数値を設定します。

たとえば、サーバー `my.cnf` ファイルで次の行を使用し、必要に応じてプラットフォームの `.so` 接尾辞とファイルの場所を調整し、選択したパスワードを置き換えます:

```
[mysqld]
early-plugin-load=keyring_encrypted_file.so
keyring_encrypted_file_data=/usr/local/mysql/mysql-keyring/keyring-encrypted
keyring_encrypted_file_password=password
```

示されているように書き込まれると、`my.cnf` ファイルにはパスワードが格納されるため、パスワードは制限モードであり、MySQL サーバーの実行に使用されるアカウントからのみアクセス可能である必要があります。

キーリング操作はトランザクションです: `keyring_encrypted_file` プラグインは、書き込み操作中にバックアップファイルを使用して、操作が失敗した場合に元のファイルにロールバックできるようにします。バックアップファイルの名前は、接尾辞が `.backup` の `keyring_encrypted_file_data` システム変数の値と同じです。

`keyring_encrypted_file` プラグインの構成に使用されるシステム変数の詳細は、[セクション6.4.4.13「キーリングシステム変数」](#) を参照してください。

正しい鍵リングストレージファイルが存在する場合にのみ鍵が確実にフラッシュされるように、`keyring_encrypted_file` は鍵リングの SHA-256 チェックサムをファイルに格納します。ファイルを更新する前に、プラグインは、予想されるチェックサムが含まれていることを確認します。また、`keyring_encrypted_file` では、AES を使用してファイルの内容を暗号化してからファイルに書き込み、ファイルの内容を復号化します。

`keyring_encrypted_file` プラグインは、標準の MySQL キーリングサービスインタフェースを構成する関数をサポートしています。これらの関数によって実行されるキーリング操作には、次の 2 つのレベルでアクセスできます:

- SQL インタフェース: SQL ステートメントで、[セクション6.4.4.10「汎用キーリングキー管理関数」](#) で説明されているユーザー定義関数 (UDF) をコールします。
- C インタフェース: C 言語コードでは、[セクション5.6.8.2「キーリングサービス」](#) で説明されているキーリングサービス関数をコールします。

例 (UDF を使用):

```
SELECT keyring_key_generate('MyKey', 'AES', 32);
SELECT keyring_key_remove('MyKey');
```

`keyring_encrypted_file` で許可されるキータイプの詳細は、[セクション6.4.4.8「サポートされているキーリングキーのタイプと長さ」](#) を参照してください。

6.4.4.4 keyring_okv KMIP プラグインの使用

注記

`keyring_okv` プラグインは、商用製品である MySQL Enterprise Edition に含まれている拡張機能です。商用製品の詳細は、<https://www.mysql.com/products/> を参照してください。

Key Management Interoperability Protocol (KMIP) は、鍵管理サーバーとそのクライアント間の暗号化鍵の通信を可能にします。`keyring_okv` キーリングプラグインは KMIP 1.1 プロトコルを使用して、KMIP バックエンドのクライアントとしてセキュアに通信します。キーリング材料は、`keyring_okv` ではなくバックエンドによって排他的に生成されます。プラグインは、次の KMIP 互換製品で動作します:

- Oracle Key Vault
- Gemalto SafeNet KeySecure アプライアンス
- Townsend Alliance Key Manager

`keyring_okv` プラグインは、標準の MySQL キーリングサービスインタフェースを構成する関数をサポートしています。これらの関数によって実行されるキーリング操作には、次の 2 つのレベルでアクセスできます:

- SQL インタフェース: SQL ステートメントで、[セクション6.4.4.10「汎用キーリングキー管理関数」](#) で説明されているユーザー定義関数 (UDF) をコールします。
- C インタフェース: C 言語コードでは、[セクション5.6.8.2「キーリングサービス」](#) で説明されているキーリングサービス関数をコールします。

例 (UDF を使用):

```
SELECT keyring_key_generate('MyKey', 'AES', 32);
SELECT keyring_key_remove('MyKey');
```

`keyring_okv`、[セクション6.4.4.8「サポートされているキーリングキーのタイプと長さ」](#)で許可されるキータイプの詳細。

`keyring_okv` プラグインをインストールするには、[セクション6.4.4.1「キーリングプラグインのインストール」](#)にある一般的なキーリングのインストール手順と、ここにある `keyring_okv` に固有の構成情報を使用します。

- [一般的な `keyring_okv` 構成](#)
- [Oracle Key Vault 用の `keyring_okv` の構成](#)
- [Gemalto SafeNet KeySecure Appliance 用の `keyring_okv` の構成](#)
- [Townsend Alliance Key Manager 用の `keyring_okv` の構成](#)
- [keyring_okv キーファイルのパスワード保護](#)

一般的な `keyring_okv` 構成

`keyring_okv` プラグインが鍵リングストレージに使用する KMIP バックエンドに関係なく、`keyring_okv_conf_dir` システム変数は、`keyring_okv` がサポートファイルに使用するディレクトリの場所を構成します。デフォルト値は空であるため、プラグインが KMIP バックエンドと通信する前に、適切に構成されたディレクトリに名前を付けるように変数を設定する必要があります。そうしないかぎり、`keyring_okv` は、通信できないメッセージをサーバーの起動時にエラーログに書き込みます:

```
[Warning] Plugin keyring_okv reported: 'For keyring_okv to be
initialized, please point the keyring_okv_conf_dir variable to a directory
containing Oracle Key Vault configuration file and ssl materials'
```

`keyring_okv_conf_dir` 変数は、次の項目を含むディレクトリに名前を付ける必要があります:

- `okvclient.ora`: `keyring_okv` が通信する KMIP バックエンドの詳細を含むファイル。
- `ssl`: KMIP バックエンドとのセキュアな接続を確立するために必要な証明書および鍵ファイルを含むディレクトリ: `CA.pem`、`cert.pem` および `key.pem`。キーファイルがパスワードで保護されている場合、`ssl` ディレクトリには、キーファイルの復号化に必要なパスワードを含む `password.txt` という単一行のテキストファイルを含めることができます。

`keyring_okv` が正常に動作するには、証明書およびキーファイルを含む `okvclient.ora` ファイルと `ssl` ディレクトリの両方が必要です。これらのファイルを構成ディレクトリに移入する手順は、他の場所で説明されているように、`keyring_okv` で使用される KMIP バックエンドによって異なります。

`keyring_okv` でサポートファイルの場所として使用される構成ディレクトリには制限モードが必要で、MySQL サーバーの実行に使用されるアカウントからのみアクセスできます。たとえば、Unix および Unix に似たシステムで `usr/local/mysql/mysql-keyring-okv` ディレクトリを使用するには、次のコマンド (`root` として実行) を実行してディレクトリを作成し、そのモードと所有権を設定します:

```
cd /usr/local/mysql
mkdir mysql-keyring-okv
chmod 750 mysql-keyring-okv
chown mysql:mysql-keyring-okv
chgrp mysql:mysql-keyring-okv
```

サーバーの起動プロセス中に使用できるようにするには、`--early-plugin-load` オプションを使用して `keyring_okv` をロードする必要があります。また、`keyring_okv_conf_dir` システム変数を設定して、構成ディレクトリの場所を `keyring_okv` に通知します。たとえば、サーバー `my.cnf` ファイルで次の行を使用して、プラットフォームの `.so` 接尾辞とディレクトリの場所を必要に応じて調整します:

```
[mysqld]
early-plugin-load=keyring_okv.so
keyring_okv_conf_dir=/usr/local/mysql/mysql-keyring-okv
```

`keyring_okv_conf_dir` の詳細は、[セクション6.4.4.13「キーリングシステム変数」](#) を参照してください。

Oracle Key Vault 用の `keyring_okv` の構成

ここでの説明は、Oracle Key Vault に精通していることを前提としています。いくつかの関連情報ソース:

- [Oracle Key Vault サイト](#)
- [Oracle Key Vault のドキュメント](#)

Oracle Key Vault 用語では、Oracle Key Vault を使用してセキュリティオブジェクトを格納および取得するクライアントはエンドポイントと呼ばれます。Oracle Key Vault と通信するには、エンドポイントとして登録し、エンドポイントサポートファイルをダウンロードしてインストールする必要があります。

次の手順は、Oracle Key Vault で使用するための `keyring_okv` の設定プロセスを簡単に要約したものです:

1. 使用する `keyring_okv` プラグインの構成ディレクトリを作成します。
2. エンドポイントを Oracle Key Vault に登録して、エンロールトークンを取得します。
3. エンロールトークンを使用して、`okvclient.jar` クライアントソフトウェアのダウンロードを取得します。
4. クライアントソフトウェアをインストールして、Oracle Key Vault サポートファイルを含む `keyring_okv` 構成ディレクトリに移入します。

`keyring_okv` と Oracle Key Vault が連携するように構成するには、次の手順を使用します。この説明では、Oracle Key Vault との対話方法のみをまとめています。詳細は、[Oracle Key Vault サイト](#) にアクセスし、『Oracle Key Vault 管理者ガイド』を参照してください。

1. Oracle Key Vault サポートファイルを含む構成ディレクトリを作成し、`keyring_okv_conf_dir` システム変数とそのディレクトリに名前を付けるように設定されていることを確認します (詳細は、[一般的な `keyring_okv` 構成](#) を参照)。
2. システム管理者ロールを持つユーザーとして Oracle Key Vault 管理コンソールにログインします。
3. Endpoints タブを選択して、Endpoints ページにアクセスします。Endpoints ページで、Add をクリックします。
4. 必要なエンドポイント情報を指定し、Register をクリックします。エンドポイントタイプはその他である必要があります。登録に成功すると、エンロールトークンが生成されます。
5. Oracle Key Vault サーバーからログアウトします。
6. 今回はログインせずに、Oracle Key Vault サーバーに再度接続します。エンドポイントエンロールトークンを使用して、`okvclient.jar` ソフトウェアのダウンロードをエンロールおよびリクエストします。このファイルをシステムに保存します。
7. 次のコマンドを使用して、`okvclient.jar` ファイルをインストールします (JDK 1.4 以上が必要です):

```
java -jar okvclient.jar -d dir_name [-v]
```

`-d` オプションに続くディレクトリ名は、抽出したファイルをインストールする場所です。 `-v` オプションを指定すると、コマンドが失敗した場合に役立つログ情報が生成されます。

コマンドで Oracle Key Vault エンドポイントパスワードを要求された場合は、指定しないでください。代わりに、Enter キーを押します。(その結果、エンドポイントが Oracle Key Vault に接続するときパスワードは必要ありません。)

8. 前述のコマンドにより、`okvclient.ora` ファイルが生成されます。このファイルは、前述の `java -jar` コマンドの `-d` オプションで指定されたディレクトリの下のこの場所にある必要があります:

```
install_dir/conf/okvclient.ora
```

ファイルの内容には、次のような行が含まれます:

```
SERVER=host_ip:port_num  
STANDBY_SERVER=host_ip:port_num
```

`keyring_okv` プラグインは、`SERVER` 変数で指定されたホスト上で実行されているサーバーと通信しようとし、失敗した場合は `STANDBY_SERVER` にフォールバックします:

- `SERVER` 変数の場合、`okvclient.ora` ファイルの設定は必須です。
- `STANDBY_SERVER` 変数の場合、`okvclient.ora` ファイルの設定はオプションです。

9. Oracle Key Vault インストーラディレクトリに移動し、次のコマンドを実行して設定をテストします:

```
okvutil/bin/okvutil list
```

出力は次のようになります:

```
Unique ID          Type          Identifier
255AB8DE-C97F-482C-E053-0100007F28B9 Symmetric Key -
264BF6E0-A20E-7C42-E053-0100007FB29C Symmetric Key -
```

新しい Oracle Key Vault サーバー (キーのないサーバー) の場合、ポートにキーがないことを示すために、出力は次のようになります:

```
no objects found
```

10. 次のコマンドを使用して、SSL 材料を含む `ssl` ディレクトリを `okvclient.jar` ファイルから抽出します:

```
jar xf okvclient.jar ssl
```

11. Oracle Key Vault サポートファイル (`okvclient.ora` ファイルおよび `ssl` ディレクトリ) を構成ディレクトリにコピーします。
12. (オプション) キーファイルをパスワードで保護する場合は、`keyring_okv` キーファイルのパスワード保護の手順を使用します。

前述の手順を完了したら、MySQL サーバーを再起動します。 `keyring_okv` プラグインがロードされ、`keyring_okv` は構成ディレクトリ内のファイルを使用して Oracle Key Vault と通信します。

Gemalto SafeNet KeySecure Appliance 用の `keyring_okv` の構成

Gemalto SafeNet KeySecure Appliance は KMIP プロトコル (バージョン 1.1 または 1.2) を使用します。 `keyring_okv` キーリングプラグイン (KMIP 1.1 をサポート) は、鍵リングストレージの KMIP バックエンドとして KeySecure を使用できます。

`keyring_okv` と KeySecure が連携するように構成するには、次の手順を使用します。説明には、KeySecure との対話方法のみがまとめられています。詳細は、Add a KMIP Server in the 「[KeySecure ユーザーガイド](#)」のセクションを参照してください。

1. KeySecure サポートファイルを含む構成ディレクトリを作成し、`keyring_okv_conf_dir` システム変数とそのディレクトリに名前を付けるように設定されていることを確認します (詳細は、[一般的な `keyring_okv` 構成](#) を参照)。
2. 構成ディレクトリで、必要な SSL 証明書およびキーファイルの格納に使用する `ssl` という名前のサブディレクトリを作成します。
3. 構成ディレクトリで、`okvclient.ora` という名前のファイルを作成します。次の形式である必要があります:

```
SERVER=host_ip:port_num
STANDBY_SERVER=host_ip:port_num
```

たとえば、KeySecure がホスト 198.51.100.20 で実行されており、ポート 9002 でリスニングしている場合、`okvclient.ora` ファイルは次のようになります:

```
SERVER=198.51.100.20:9002
STANDBY_SERVER=198.51.100.20:9002
```

4. 認証局アクセスの資格証明を持つ管理者として KeySecure 管理コンソールに接続します。
5. セキュリティ→ローカル CA に移動し、ローカル認証局 (CA) を作成します。

- 信頼できる CA リストに移動します。デフォルトを選択し、プロパティをクリックします。次に、信頼できる認証局リストの編集を選択し、作成した CA を追加します。
- CA をダウンロードし、[CA.pem](#) という名前のファイルとして `ssl` ディレクトリに保存します。
- Security >> Certificate Requests に移動し、証明書を作成します。その後、証明書 PEM ファイルを含む圧縮 `tar` ファイルをダウンロードできます。
- ダウンロードしたファイルから PEM ファイルを抽出します。たとえば、ファイル名が `csr_w_pk_pkcs8.gz` の場合は、次のコマンドを使用して解凍および解凍します:

```
tar zxvf csr_w_pk_pkcs8.gz
```

2 つのファイルが抽出操作によって生成されます: [certificate_request.pem](#) および [private_key_pkcs8.pem](#)。

- この `openssl` コマンドを使用して、秘密キーを復号化し、[key.pem](#) という名前のファイルを作成します:

```
openssl pkcs8 -in private_key_pkcs8.pem -out key.pem
```

- [key.pem](#) ファイルを `ssl` ディレクトリにコピーします。
- [certificate_request.pem](#) の証明書リクエストをクリップボードにコピーします。
- Security >> Local CA に移動します。以前に作成した CA ([CA.pem](#) ファイルを作成するためにダウンロードした CA) を選択し、署名リクエストをクリックします。クリップボードから証明書リクエストを貼り付け、クライアントの証明書の目的 (キーリングは KeySecure のクライアント) を選択して、署名リクエストをクリックします。その結果、選択した CA で署名された証明書が新しいページに表示されます。
- 署名付き証明書をクリップボードにコピーし、クリップボードの内容を [cert.pem](#) という名前のファイルとして `ssl` ディレクトリに保存します。
- (オプション) キーファイルをパスワードで保護する場合は、[keyring_okv キーファイルのパスワード保護](#) の手順を使用します。

前述の手順を完了したら、MySQL サーバーを再起動します。[keyring_okv](#) プラグインがロードされ、[keyring_okv](#) は構成ディレクトリ内のファイルを使用して KeySecure と通信します。

Townsend Alliance Key Manager 用の `keyring_okv` の構成

Townsend Alliance Key Manager は KMIP プロトコルを使用します。[keyring_okv](#) キーリングプラグインは、キーリングストレージの KMIP バックエンドとして Alliance Key Manager を使用できます。詳細は、「[MySQL のアライアンスキーマネージャ](#)」を参照してください。

`keyring_okv` キーファイルのパスワード保護

オプションで、キーファイルをパスワードで保護し、パスワードを含むファイルを指定してキーファイルを復号化できます。これを行うには、場所を `ssl` ディレクトリに変更し、次のステップを実行します:

- [key.pem](#) キーファイルを暗号化します。たとえば、次のようなコマンドを使用して、プロンプトで暗号化パスワードを入力します:

```
shell> openssl rsa -des3 -in key.pem -out key.pem.new
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
```

- `ssl` ディレクトリの [password.txt](#) という単一行のテキストファイルに暗号化パスワードを保存します。
- 次のコマンドを使用して、暗号化されたキーファイルを復号化できることを確認します。復号化されたファイルがコンソールに表示されます:

```
shell> openssl rsa -in key.pem.new -passin file:password.txt
```

- 元の [key.pem](#) ファイルを削除し、[key.pem.new](#) の名前を [key.pem](#) に変更します。
- 必要に応じて、新しい [key.pem](#) ファイルおよび [password.txt](#) ファイルの所有権とアクセスモードを変更し、`ssl` ディレクトリ内の他のファイルと同じ制限があることを確認します。

6.4.4.5 keyring_aws Amazon Web Services キーリングプラグインの使用

注記

`keyring_aws` プラグインは、商用製品である MySQL Enterprise Edition に含まれている拡張機能です。商用製品の詳細は、<https://www.mysql.com/products/> を参照してください。

`keyring_aws` キーリングプラグインは、キー生成のバックエンドとして Amazon Web Services Key Management Service (AWS KMS) と通信し、キーの格納にローカルファイルを使用します。すべてのキーリングデータは、AWS サーバーによってのみ生成され、`keyring_aws` によっては生成されません。

`keyring_aws` は、次のプラットフォームで使用できます:

- EL7
- macOS 10.13 および 10.14
- SLES 12
- Ubuntu 14.04 および 16.04
- Windows

ここでの説明は、AWS 全般および KMS に精通していることを前提としています。いくつかの関連情報ソース:

- [AWS サイト](#)
- [KMS ドキュメント](#)

次の各セクションでは、`keyring_aws` キープラグインの構成および使用方法について説明します:

- [keyring_aws 構成](#)
- [keyring_aws 操作](#)
- [keyring_aws 資格証明の変更](#)

keyring_aws 構成

`keyring_aws` プラグインをインストールするには、ここにあるプラグイン固有の構成情報とともに、[セクション 6.4.4.1 「キーリングプラグインのインストール」](#) にある一般的なキーリングインストール手順を使用します。

プラグインライブラリファイルには、`keyring_aws` プラグインと、`keyring_aws_rotate_cmk()` および `keyring_aws_rotate_keys()` の 2 つのユーザー定義関数 (UDF) が含まれます。

`keyring_aws` を構成するには、AWS KMS と通信するための資格証明を提供する秘密アクセスキーを取得し、それを構成ファイルに書き込む必要があります:

1. AWS KMS アカウントを作成します。
2. AWS KMS を使用して、秘密アクセスキー ID および秘密アクセスキーを作成します。アクセスキーは、アイデンティティおよびアプリケーションのアイデンティティを検証するために機能します。
3. AWS KMS アカウントを使用して、顧客マスターキー (CMK) ID を作成します。MySQL の起動時に、`keyring_aws_cmk_id` システム変数を CMK ID 値に設定します。この変数は必須であり、デフォルトはありません。(この値は、必要に応じて `SET GLOBAL` を使用して実行時に変更できます。)
4. 必要に応じて、構成ファイルを配置するディレクトリを作成します。ディレクトリには制限モードがあり、MySQL サーバーの実行に使用されるアカウントからのみアクセスできる必要があります。たとえば、Unix および Unix に似たシステムでは、`/usr/local/mysql/mysql-keyring/keyring_aws_conf` をファイル名として使用するために、次のコマンド (`root` として実行) によって親ディレクトリが作成され、ディレクトリのモードと所有権が設定されます:

```
shell> cd /usr/local/mysql
shell> mkdir mysql-keyring
```

```
shell> chmod 750 mysql-keyring
shell> chown mysql:mysql-keyring
shell> chgrp mysql:mysql-keyring
```

MySQL の起動時に、`keyring_aws_conf_file` システム変数を `/usr/local/mysql/mysql-keyring/keyring_aws_conf` に設定して、構成ファイルの場所をサーバーに示します。

5. 次の 2 つの行を含む `keyring_aws` 構成ファイルを準備します:

- 明細 1: 秘密アクセスキー ID
- 明細 2: 秘密アクセスキー

たとえば、キー ID が `wwwwwwwwwwwwwwwwEXAMPLE` で、キーが `xxxxxxxxxxxxx/yyyyyy/zzzzzzzEXAMPLEKEY` の場合、構成ファイルは次のようになります:

```
wwwwwwwwwwwwwwwwEXAMPLE
xxxxxxxxxxxxx/yyyyyy/zzzzzzzEXAMPLEKEY
```

サーバーの起動プロセス中に使用できるようにするには、`--early-plugin-load` オプションを使用して `keyring_aws` をロードする必要があります。`keyring_aws_cmk_id` システム変数は必須であり、AWS KMS サーバーから取得した顧客マスターキー (CMK) ID を構成します。`keyring_aws_conf_file` および `keyring_aws_data_file` システム変数は、オプションで、`keyring_aws` プラグインが構成情報およびデータ記憶域に使用するファイルの場所を構成します。ファイルの場所変数のデフォルト値はプラットフォーム固有です。場所を明示的に構成するには、起動時に変数値を設定します。たとえば、サーバー `my.cnf` ファイルで次の行を使用して、プラットフォームの `.so` 接尾辞とファイルの場所を必要に応じて調整します:

```
[mysql]
early-plugin-load=keyring_aws.so
keyring_aws_cmk_id='arn:aws:kms:us-west-2:111122223333:key/abcd1234-ef56-ab12-cd34-ef56abcd1234'
keyring_aws_conf_file=/usr/local/mysql/mysql-keyring/keyring_aws_conf
keyring_aws_data_file=/usr/local/mysql/mysql-keyring/keyring_aws_data
```

`keyring_aws` プラグインを正常に起動するには、構成ファイルが存在し、前述のように初期化された有効な秘密アクセスキー情報が含まれている必要があります。ストレージファイルが存在する必要はありません。そうでない場合、`keyring_aws` はそれを (必要に応じて親ディレクトリとともに) 作成しようとします。

`keyring_aws` プラグインの構成に使用されるシステム変数の詳細は、[セクション6.4.4.13「キーリングシステム変数」](#)を参照してください。

MySQL サーバーを起動し、`keyring_aws` プラグインに関連付けられた UDF をインストールします。これはワンタイム操作で、次のステートメントを実行して、プラットフォームの `.so` 接尾辞を必要に応じて調整します:

```
CREATE FUNCTION keyring_aws_rotate_cmk RETURNS INTEGER
SONAME 'keyring_aws.so';
CREATE FUNCTION keyring_aws_rotate_keys RETURNS INTEGER
SONAME 'keyring_aws.so';
```

`keyring_aws` UDF の詳細は、[セクション6.4.4.11「プラグイン固有のキーリングキー管理関数」](#)を参照してください。

keyring_aws 操作

プラグインの起動時に、`keyring_aws` プラグインは AWS 秘密アクセスキー ID とキーをその構成ファイルから読み取ります。また、記憶域ファイルに含まれる暗号化されたキーをメモリー内キャッシュに読み取ります。

操作中、`keyring_aws` は暗号化されたキーをインメモリーキャッシュに保持し、記憶域ファイルをローカル永続記憶域として使用します。各キーリング操作はトランザクション型です:`keyring_aws` では、メモリー内キーキャッシュとキーリングストレージファイルの両方が正常に変更されるか、操作が失敗してキーリングの状態は変更されません。

正しい鍵リングストレージファイルが存在する場合にのみ鍵が確実にフラッシュされるように、`keyring_aws` は鍵リングの SHA-256 チェックサムをファイルに格納します。ファイルを更新する前に、プラグインは、予想されるチェックサムが含まれていることを確認します。

`keyring_aws` プラグインは、標準の MySQL キーリングサービスインタフェースを構成する関数をサポートしています。これらの関数によって実行されるキーリング操作には、次の 2 つのレベルでアクセスできます:

- SQL インタフェース: SQL ステートメントで、[セクション6.4.4.10「汎用キーリングキー管理関数」](#)で説明されているユーザー定義関数 (UDF) をコールします。
- C インタフェース: C 言語コードでは、[セクション5.6.8.2「キーリングサービス」](#)で説明されているキーリングサービス関数をコールします。

例 (UDF を使用):

```
SELECT keyring_key_generate('MyKey', 'AES', 32);  
SELECT keyring_key_remove('MyKey');
```

また、`keyring_aws_rotate_cmk()` および `keyring_aws_rotate_keys()` UDF は、標準のキーリングサービスインタフェースでカバーされていない AWS 関連の機能を提供するために、キーリングプラグインインタフェースを「extend」に提供します。これらの機能には、UDF を呼び出すことによるのみアクセスできます。対応する C 言語キーサービス関数はありません。

`keyring_aws` で許可されるキータイプの詳細は、[セクション6.4.4.8「サポートされているキーリングキーのタイプと長さ」](#)を参照してください。

keyring_aws 資格証明の変更

`keyring_aws` プラグインがサーバーの起動時に適切に初期化されていると仮定すると、AWS KMS との通信に使用される資格証明を変更できます:

1. AWS KMS を使用して、新しい秘密アクセスキー ID および秘密アクセスキーを作成します。
2. 新しい資格証明を構成ファイル (`keyring_aws_conf_file` システム変数で指定されたファイル) に格納します。ファイル形式は前述のとおりです。
3. `keyring_aws` プラグインを再初期化して、構成ファイルを再読み込みします。新しい資格証明が有効であると仮定すると、プラグインは正常に初期化されます。

プラグインを再初期化するには、2つの方法があります:

- サーバーを再起動します。これは単純で副作用はありませんが、できるだけ再起動が少ないサーバーの停止時間を最小限に抑えるインストールには適していません。
- 次のステートメントを実行して、必要に応じてプラットフォームの `.so` 接尾辞を調整し、サーバーを再起動せずにプラグインを再初期化します:

```
UNINSTALL PLUGIN keyring_aws;  
INSTALL PLUGIN keyring_aws SONAME 'keyring_aws.so';
```

注記

`INSTALL PLUGIN` では、実行時にプラグインをロードするだけでなく、プラグインを `mysql.plugin` システムテーブルに登録するという副作用もあります。このため、`keyring_aws` の使用を停止する場合は、サーバーの起動に使用される一連のオプションから `--early-plugin-load` オプションを削除するだけでは不十分です。これにより、プラグインの早期ロードは停止されますが、サーバーは、`mysql.plugin` に登録されているプラグインをロードする起動シーケンス内のポイントに到達したときに、プラグインのロードを試行します。

したがって、AWS KMS 資格証明を変更するために説明した `UNINSTALL PLUGIN` と `INSTALL PLUGIN` の順序を実行した場合、`keyring_aws` の使用を停止するには、`--early-plugin-load` オプションの削除に加えて、`UNINSTALL PLUGIN` を再度実行してプラグインを登録解除する必要があります。

6.4.4.6 HashiCorp Vault キーリングプラグインの使用

注記

`keyring_hashicorp` プラグインは、商用製品である MySQL Enterprise Edition に含まれている拡張機能です。商用製品の詳細は、<https://www.mysql.com/products/> を参照してください。

[keyring_hashicorp](#) キーリングプラグインは、バックエンドストレージのために HashiCorp Vault と通信します。プラグインは、HashiCorp Vault AppRole 認証をサポートしています。キー情報は、MySQL サーバーのローカル記憶域に永続的に格納されません。(オプションのインメモリキーキャッシュを中間記憶域として使用できます。) ランダムキー生成は MySQL サーバー側で実行され、その後、キーは Hashicorp Vault に格納されます。

[keyring_hashicorp](#) プラグインは、標準の MySQL キーリングサービスインタフェースを構成する関数をサポートしています。これらの関数によって実行されるキーリング操作には、次の 2 つのレベルでアクセスできます:

- SQL インタフェース: SQL ステートメントで、[セクション6.4.4.10「汎用キーリングキー管理関数」](#) で説明されているユーザー定義関数 (UDF) をコールします。
- C インタフェース: C 言語コードでは、[セクション5.6.8.2「キーリングサービス」](#) で説明されているキーリングサービス関数をコールします。

例 (UDF を使用):

```
SELECT keyring_key_generate('MyKey', 'AES', 32);
SELECT keyring_key_remove('MyKey');
```

[keyring_hashicorp](#) で許可されるキータイプの詳細は、[セクション6.4.4.8「サポートされているキーリングキーのタイプと長さ」](#) を参照してください。

[keyring_hashicorp](#) プラグインをインストールするには、[セクション6.4.4.1「キーリングプラグインのインストール」](#) にある一般的なキーリングのインストール手順と、ここにある [keyring_hashicorp](#) に固有の構成情報を使用します。プラグイン固有の構成には、HashiCorp Vault への接続および Vault 自体の構成に必要な証明書およびキーファイルの準備が含まれます。次の各セクションでは、必要な手順について説明します。

- [証明書とキーの準備](#)
- [HashiCorp Vault の設定](#)
- [keyring_hashicorp 構成](#)

証明書とキーの準備

[keyring_hashicorp](#) プラグインには、HTTPS プロトコルを使用した HashiCorp Vault サーバーへのセキュアな接続が必要です。一般的な設定には、証明書とキーファイルのセットが含まれます:

- **company.crt**: 組織に属するカスタム CA 証明書。このファイルは、HashiCorp Vault サーバーと [keyring_hashicorp](#) プラグインの両方で使用されます。
- **vault.key**: HashiCorp Vault サーバーインスタンスの秘密キー。このファイルは、HashiCorp Vault サーバーで使用されます。
- **vault.crt**: HashiCorp Vault サーバーインスタンスの証明書。このファイルは、組織の CA 証明書によって署名されている必要があります。

次の手順では、OpenSSL を使用して証明書およびキーファイルを作成する方法について説明します。(すでにファイルがある場合は、[HashiCorp Vault の設定](#) に進みます。) 示されている手順は Linux プラットフォームに適用され、他のプラットフォームに対する調整が必要になる場合があります。

重要

これらの手順で生成される証明書は自己署名されており、あまりセキュアではない可能性があります。このようなファイルの使用経験がある場合は、登録された認証局から証明書/キーデータを取得することを検討してください。

1. 会社キーと HashiCorp Vault サーバーキーを準備します。

次のコマンドを使用して、キーファイルを生成します:

```
openssl genrsa -aes256 -out company.key 4096
openssl genrsa -aes256 -out vault.key 2048
```

これらのコマンドは、会社の秘密キー ([company.key](#)) および Vault サーバーの秘密キー ([vault.key](#)) を保持するファイルを作成します。鍵はそれぞれ 4,096 ビットと 2,048 ビットのランダムに生成された RSA 鍵です。

各コマンドでは、パスワードの入力を求められます。(テスト目的では、パスワードは必要ありません。無効にするには、[-aes256](#) 引数を省略します。)

キーファイルは機密情報を保持し、セキュアな場所に格納する必要があります。パスワードは後で必要になるため、書き留めて安全な場所に格納します。

(オプション) キーファイルの内容と有効性を確認するには、次のコマンドを使用します:

```
openssl rsa -in company.key -check
openssl rsa -in vault.key -check
```

2. 会社の CA 証明書を作成します。

365 日間有効な [company.crt](#) という名前の会社の CA 証明書ファイルを作成するには、次のコマンドを使用します (コマンドは単一行で入力します):

```
openssl req -x509 -new -nodes -key company.key
-sha256 -days 365 -out company.crt
```

[-aes256](#) 引数を使用してキーの生成時にキーの暗号化を実行した場合、CA 証明書の作成時に会社のキーパスワードの入力を求められます。次に示すように、証明書ホルダー (つまり、自分または会社) に関する情報の入力も求められます:

```
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:
```

プロンプトに適切な値を入力します。

3. 証明書署名リクエストを作成します。

HashiCorp Vault サーバー証明書を作成するには、新しく作成したサーバーキーに対して証明書署名リクエスト (CSR) を準備する必要があります。次の行を含む [request.conf](#) という名前の構成ファイルを作成します。HashiCorp Vault サーバーがローカルホストで実行されていない場合は、適切な CN 値と IP 値を置き換え、必要に応じて他の変更を行います。

```
[req]
distinguished_name = vault
x509_extensions = v3_req
prompt = no

[vault]
C = US
ST = CA
L = RWC
O = Company
CN = 127.0.0.1

[v3_req]
subjectAltName = @alternatives
authorityKeyIdentifier = keyid,issuer
basicConstraints = CA:TRUE

[alternatives]
IP = 127.0.0.1
```

署名要求を作成するには、次のコマンドを使用します:

```
openssl req -new -key vault.key -config request.conf -out request.csr
```

出力ファイル ([request.csr](#)) は、サーバー証明書を作成するための入力として機能する中間ファイルです。

4. HashiCorp Vault サーバー証明書を作成します。

HashiCorp Vault サーバーキー (`vault.key`) と CSR (`request.csr`) の結合情報に会社の証明書 (`company.crt`) を使用して署名し、HashiCorp Vault サーバー証明書 (`vault.crt`) を作成します。これを行うには、次のコマンドを使用します (コマンドは単一行で入力します):

```
openssl x509 -req -in request.csr
-CA company.crt -CAkey company.key -CAcreateserial
-out vault.crt -days 365 -sha256
```

`vault.crt` サーバー証明書を役に立つようにするには、`company.crt` 会社の証明書の内容を追加します。これは、会社の証明書がリクエストでサーバー証明書とともに配信されるようにするために必要です。

```
cat company.crt >> vault.crt
```

テキストエディタで `vault.crt` ファイルを開くと、その内容は次のようになります:

```
-----BEGIN CERTIFICATE-----
... content of HashiCorp Vault server certificate ...
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
... content of company certificate ...
-----END CERTIFICATE-----
```

HashiCorp Vault の設定

次の手順では、`keyring_hashicorp` プラグインのテストを容易にする HashiCorp Vault 設定を作成する方法について説明します。

重要

テスト設定は本番設定と似ていますが、HashiCorp Vault を本番で使用するには、自己署名されていない証明書の使用やシステムトラストストアへの会社の証明書の格納など、追加のセキュリティ上の考慮事項が必要です。運用要件を満たすために必要な追加のセキュリティステップを実装する必要があります。

次の手順では、`証明書とキーの準備` で作成された証明書およびキーファイルが使用可能であることを前提としています。ファイルがない場合は、そのセクションを参照してください。

1. HashiCorp Vault バイナリをフェッチします。

ご使用のプラットフォームに適した HashiCorp Vault バイナリを <https://www.vaultproject.io/downloads.html> からダウンロードします。

アーカイブの内容を抽出して、HashiCorp Vault 操作の実行に使用される実行可能な `vault` コマンドを生成します。必要に応じて、コマンドをインストールするディレクトリをシステムパスに追加します。

(オプション) HashiCorp Vault では、使いやすくするオートコンプリートオプションがサポートされています。詳細は、<https://learn.hashicorp.com/vault/getting-started/install#command-completion> を参照してください。

2. HashiCorp Vault サーバー構成ファイルを作成します。

次の内容を含む `config.hcl` という名前の構成ファイルを準備します。 `tls_cert_file`、`tls_key_file` および `path` の値は、使用しているシステムに適したパス名に置き換えてください。

```
listener "tcp" {
  address="127.0.0.1:8200"
  tls_cert_file="/home/username/certificates/vault.crt"
  tls_key_file="/home/username/certificates/vault.key"
}

storage "file" {
  path = "/home/username/vaultstorage/storage"
}

ui = true
```

3. HashiCorp Vault サーバーを起動します。

Vault サーバーを起動するには、次のコマンドを使用します。ここで、`-config` オプションは作成したばかりの構成ファイルへのパスを指定します:

```
vault server -config=config.hcl
```

このステップでは、`vault.key` ファイルに格納されている Vault サーバーの秘密キーのパスワードの入力を求められる場合があります。

サーバーが起動し、コンソールにいくつかの情報 (IP、ポートなど) が表示されます。

残りのコマンドを入力できるように、`vault server` コマンドをバックグラウンドに置か、続行する前に別の端末を開きます。

4. HashiCorp Vault サーバーを初期化します。

注記

この手順で説明する操作は、開封キーとルートトークンを取得するために Vault を初めて起動する場合にのみ必要です。その後の Vault インスタンスの再起動では、シール解除キーを使用したアンシールのみが必要です。

次のコマンドを発行します (Bourne シェル構文を想定):

```
export VAULT_SKIP_VERIFY=1
vault operator init -n 1 -t 1
```

最初のコマンドを使用すると、`vault` コマンドは、システムトラストストアに会社の証明書が追加されていないことを一時的に無視できます。自己署名 CA がそのストアに追加されていないという事実を補う。(本番で使用するには、このような証明書を追加する必要があります。)

2 番目のコマンドは、シール解除のために 1 つのシール解除キーが存在する必要がある単一のシール解除キーを作成します。(本番で使用する場合、インスタンスには複数のアンシールキーがあり、アンシールするために入力する必要があるキーはその数までです。開封キーは、会社内の主要な顧客に配信する必要があります。単一のキーを使用すると、ポルトを単一のキーカスタムによってシール解除できるため、セキュリティ上の問題とみなされる場合があります。)

Vault は、シール解除キーとルートトークンに関する情報に加えて、追加のテキストを返信する必要があります (実際のシール解除キーとルートトークンの値はここに示す値とは異なります):

```
...
Unseal Key 1: I2xwcFQc892O0Nt2pBiRNlnkHHzTUrWS+JybL39BjcOE=
Initial Root Token: s.vTvXeo3tPEYehfcd9WH7oUKz
...
```

シール解除キーとルートトークンをセキュアな場所に格納します。

5. HashiCorp Vault サーバーをシール解除します。

Vault サーバをシール解除するには、次のコマンドを使用します:

```
vault operator unseal
```

開封キーの入力を求めるプロンプトが表示されたら、Vault の初期化時に以前に取得したキーを使用します。

Vault は、セットアップが完了し、ポルトがシールなしであることを示す出力を生成します。

6. HashiCorp Vault サーバーにログインし、そのステータスを確認します。

`root` としてログインするために必要な環境変数を準備します:

```
vault login s.vTvXeo3tPEYehfcd9WH7oUKz
```

このコマンドのトークン値は、Vault の初期化中に以前に取得した `root` トークンの内容に置き換えてください。

Vault サーバのステータスを確認します:

```
vault status
```

出力には次の行が含まれる必要があります (特に):

```
...
Initialized true
Sealed      false
...
```

7. HashiCorp Vault の認証および記憶域を設定します。

注記

この手順で説明する操作は、Vault インスタンスの初回実行時にのみ必要です。後で繰り返す必要はありません。

AppRole 認証方式を有効にして、認証方式リストに含まれていることを確認します:

```
vault auth enable approle
vault auth list
```

Vault KeyValue ストレージエンジンを有効にします:

```
vault secrets enable -version=1 kv
```

[keyring_hashicorp](#) プラグインで使用するロールを作成および設定します (コマンドを単一行で入力します):

```
vault write auth/approle/role/mysql token_num_uses=0
token_ttl=20m token_max_ttl=30m secret_id_num_uses=0
```

8. AppRole セキュリティポリシーを追加します。

注記

この手順で説明する操作は、Vault インスタンスの初回実行時にのみ必要です。後で繰り返す必要はありません。

以前に作成したロールに適切なシークレットへのアクセスを許可するポリシーを準備します。次の内容で [mysql.hcl](#) という名前の新しいファイルを作成します:

```
path "kv/mysql/*" {
  capabilities = ["create", "read", "update", "delete", "list"]
}
```

ポリシーファイルを Vault サーバーにインポートして [mysql-policy](#) という名前のポリシーを作成し、そのポリシーを新しいロールに割り当てます:

```
vault policy write mysql-policy mysql.hcl
vault write auth/approle/role/mysql policies=mysql-policy
```

新しく作成したロールの ID を取得し、セキュアな場所に格納します:

```
vault read auth/approle/role/mysql/role-id
```

ロールのシークレット ID を生成し、セキュアな場所に格納します:

```
vault write -f auth/approle/role/mysql/secret-id
```

これらの AppRole ロール ID およびシークレット ID の資格証明が生成された後、それらは無期限に有効なままである必要があります。これらを再度生成する必要はなく、継続的に使用するよう [keyring_hashicorp](#) プラグインを構成できます。AuthRole 認証の詳細は、<https://www.vaultproject.io/docs/auth/approle.html> を参照してください。

keyring_hashicorp 構成

プラグインライブラリファイルには、[keyring_hashicorp](#) プラグインとユーザー定義関数 (UDF)、[keyring_hashicorp_update_config\(\)](#) が含まれています。プラグインが初期化および終了すると、UDF が自動的にロードおよびアンロードされるため、UDF を手動でロードおよびアンロードする必要はありません。

[keyring_hashicorp](#) プラグインは、次のテーブルに示す構成パラメータをサポートしています。これらのパラメータを指定するには、対応するシステム変数に値を割り当てます。

構成パラメータ	システム変数	必須
HashiCorp サーバー URL	keyring_hashicorp_server_url	いいえ
AppRole ロール ID	keyring_hashicorp_role_id	はい
AppRole シークレット ID	keyring_hashicorp_secret_id	はい
ストアパス	keyring_hashicorp_store_path	はい
認可パス	keyring_hashicorp_auth_path	いいえ
CA 証明書ファイルのパス	keyring_hashicorp_ca_path	いいえ
キャッシュ制御	keyring_hashicorp_caching	いいえ

サーバーの起動プロセス中に使用できるようにするには、`--early-plugin-load` オプションを使用して [keyring_hashicorp](#) をロードする必要があります。前述のテーブルに示されているように、プラグイン関連のいくつかのシステム変数は必須であり、設定する必要もあります。たとえば、サーバー `my.cnf` ファイルで次の行を使用して、プラットフォームの `.so` 接尾辞とファイルの場所を必要に応じて調整します：

```
[mysqld]
early-plugin-load=keyring_hashicorp.so
keyring_hashicorp_role_id='ee3b495c-d0c9-11e9-8881-8444c71c32aa'
keyring_hashicorp_secret_id='0512af29-d0ca-11e9-95ee-0010e00dd718'
keyring_hashicorp_store_path='/v1/kv/mysql'
```

MySQL Server は、AppRole 認証を使用して HashiCorp Vault に対して認証します。認証に成功するには、Vault にロール ID とシークレット ID の 2 つのシークレットを提供する必要があります。これらは、ユーザー名とパスワードの概念に似ています。使用するロール ID およびシークレット ID の値は、以前に実行した HashiCorp Vault の設定手順で取得した値です。2 つの ID を指定するには、それぞれの値を [keyring_hashicorp_role_id](#) および [keyring_hashicorp_secret_id](#) システム変数に割り当てます。設定手順では、`/v1/kv/mysql` のストアパスも生成されます。これは、[keyring_hashicorp_commit_store_path](#) に割り当てられる値です。

プラグインの初期化時に、[keyring_hashicorp](#) は構成値を使用して HashiCorp Vault サーバーへの接続を試みます。接続が成功すると、プラグインは名前に `_commit` が含まれる対応するシステム変数に値を格納します。たとえば、接続が成功すると、プラグインは [keyring_hashicorp_role_id](#) および [keyring_hashicorp_store_path](#) の値を [keyring_hashicorp_commit_role_id](#) および [keyring_hashicorp_commit_store_path](#) に格納します。

実行時の再構成は、[keyring_hashicorp_update_config\(\)](#) UDF を使用して実行できます：

1. `SET` ステートメントを使用して、前述のテーブルに示す構成システム変数に必要な新しい値を割り当てます。これらの割り当て自体は、進行中のプラグイン操作には影響しません。
2. [keyring_hashicorp_update_config\(\)](#) を起動してプラグインを再構成し、新しい変数値を使用して HashiCorp Vault サーバーに再接続します。
3. 接続が成功すると、プラグインは、名前に `_commit` が含まれる対応するシステム変数に更新された構成値を格納します。

たとえば、デフォルト 8200 ではなくポート 8201 でリスニングするように HashiCorp Vault を再構成した場合は、次のように [keyring_hashicorp](#) を再構成します：

```
mysql> SET GLOBAL keyring_hashicorp_server_url = 'https://127.0.0.1:8201';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT keyring_hashicorp_update_config();
+-----+
| keyring_hashicorp_update_config() |
+-----+
| Configuration update was successful. |
+-----+
1 row in set (0.03 sec)
```

初期化または再構成中にプラグインが HashiCorp Vault に接続できず、既存の接続がなかった場合、`_commit_` システム変数は文字列値変数に対して 'Not committed' に設定され、ブール値変数に対して OFF に設定されます。プラグインが接続できないが、既存の接続があった場合、その接続はアクティブなままで、`_commit_` 変数にはその接続に使用された値が反映されます。

注記

サーバーの起動時に必須のシステム変数を設定しない場合、または他のプラグイン初期化エラーが発生した場合、初期化は失敗します。この場合、実行時再構成手順を使用すると、サーバーを再起動せずにプラグインを初期化できます。

`keyring_hashicorp` プラグイン固有のシステム変数および UDF の詳細は、[セクション6.4.4.13「キーリングシステム変数」](#) および [セクション6.4.4.11「プラグイン固有のキーリングキー管理関数」](#) を参照してください。

6.4.4.7 Oracle Cloud Infrastructure Vault キーリングプラグインの使用

注記

`keyring_oci` プラグインは、商用製品である MySQL Enterprise Edition に含まれている拡張機能です。商用製品の詳細は、<https://www.mysql.com/products/> を参照してください。

`keyring_oci` プラグインは、バックエンドストレージのために Oracle Cloud Infrastructure Vault と通信するキーリングプラグインです。キー情報は、MySQL サーバーのローカル記憶域に永続的に格納されません。すべてのキーは Oracle Cloud Infrastructure Vault に格納されるため、このプラグインは Oracle Cloud Infrastructure MySQL 顧客の MySQL Enterprise Edition キーの管理に適しています。

`keyring_oci` プラグインは、標準の MySQL キーリングサービスインタフェースを構成する関数をサポートしています。これらの関数によって実行されるキーリング操作には、次の2つのレベルでアクセスできます：

- SQL インタフェース: SQL ステートメントで、[セクション6.4.4.10「汎用キーリングキー管理関数」](#) で説明されているユーザー定義関数 (UDF) をコールします。
- C インタフェース: C 言語コードでは、[セクション5.6.8.2「キーリングサービス」](#) で説明されているキーリングサービス関数をコールします。

例 (UDF を使用):

```
SELECT keyring_key_generate('MyKey', 'AES', 32);
SELECT keyring_key_remove('MyKey');
```

`keyring_oci` で許可されるキータイプの詳細は、[セクション6.4.4.8「サポートされているキーリングキーのタイプと長さ」](#) を参照してください。

`keyring_oci` プラグインをインストールするには、[セクション6.4.4.1「キーリングプラグインのインストール」](#) にある一般的なキーリングのインストール手順と、ここにある `keyring_oci` に固有の構成情報を使用します。プラグイン固有の構成には、Oracle Cloud Infrastructure リソースの名前または値を示す多数のシステム変数の設定が含まれます。

Oracle Cloud Infrastructure の概念に精通していることを前提としていますが、`keyring_oci` プラグインで使用するリソースを設定する際に次のドキュメントが役立つ場合があります：

- [Vault の概要](#)
- [リソース識別子](#)
- [必要なキーおよび OCID](#)
- [キーの管理](#)
- [コンパートメントの管理](#)
- [ポールの管理](#)
- [シークレットの管理](#)

`keyring_oci` プラグインは、次のテーブルに示す構成パラメータをサポートしています。これらのパラメータを指定するには、対応するシステム変数に値を割り当てます。

構成パラメータ	システム変数	必須
ユーザー OCID	keyring_oci_user	はい
テナンシ OCID	keyring_oci_tenancy	はい
コンパートメント OCID	keyring_oci_compartment	はい
ポールの OCID	keyring_oci_virtual_vault	はい
マスターキー OCID	keyring_oci_master_key	はい
暗号化サーバーエンドポイント	keyring_oci_encryption_endpoint	はい
キー管理サーバーエンドポイント	keyring_oci_management_endpoint	はい
ポールのサーバーエンドポイント	keyring_oci_vaults_endpoint	はい
シークレットサーバーエンドポイント	keyring_oci_secrets_endpoint	はい
RSA 秘密キーファイル	keyring_oci_key_file	はい
RSA 秘密キーのフィンガープリント	keyring_oci_key_fingerprint	はい
CA 証明書バンドルファイル	keyring_oci_ca_certificate	いいえ

サーバーの起動プロセス中に使用できるようにするには、`--early-plugin-load` オプションを使用して [keyring_oci](#) をロードする必要があります。前述のテーブルに示されているように、プラグイン関連のいくつかのシステム変数は必須であり、設定する必要があります:

- Oracle Cloud Infrastructure では、Oracle Cloud ID (OCID) を広範囲に使用してリソースを指定し、いくつかの [keyring_oci](#) パラメータでは、使用するリソースの OCID 値を指定します。したがって、[keyring_oci](#) プラグインを使用する前に、次の前提条件を満たす必要があります:
 - Oracle Cloud Infrastructure に接続するためのユーザーが存在する必要があります。必要に応じてユーザーを作成し、ユーザー OCID を [keyring_oci_user](#) システム変数に割り当てます。
 - 使用する Oracle Cloud Infrastructure テナンシ、テナンシ内の MySQL コンパートメントおよびコンパートメント内のポールの存在する必要があります。必要に応じてこれらのリソースを作成し、ユーザーが使用できるようになっていることを確認します。テナンシ、コンパートメントおよびポールの OCID を [keyring_oci_tenancy](#)、[keyring_oci_compartment](#) および [keyring_oci_virtual_vault](#) システム変数に割り当てます。
 - 暗号化用のマスターキーが存在する必要があります。必要に応じて OCID を作成し、[keyring_oci_master_key](#) システム変数に割り当てます。
- 複数のサーバーエンドポイントを指定する必要があります。これらのエンドポイントはポールの固有であり、Oracle Cloud Infrastructure によってポールの作成時に割り当てられます。ポールの詳細ページから値を取得し、[keyring_oci_encryption_endpoint](#)、[keyring_oci_management_endpoint](#)、[keyring_oci_vaults_endpoint](#) および [keyring_oci_secrets_endpoint](#) システム変数に割り当てます。
- Oracle Cloud Infrastructure API では、RSA 秘密キーと公開キーのペアを認証に使用します。このキーペアを作成してキーフィンガープリントを取得するには、「[必要なキーおよび OCID](#)」の手順を使用します。秘密キーファイル名とキーフィンガープリントを [keyring_oci_key_file](#) および [keyring_oci_key_fingerprint](#) システム変数に割り当てます。

必須のシステム変数に加えて、ピア認証用の認証局 (CA) 証明書バンドルファイルを指定するように [keyring_oci_ca_certificate](#) をオプションで設定できます。

重要

Oracle Cloud Infrastructure コンソールからパラメータをコピーする場合、コピーされた値に初期 [https://](#)部分が含まれている可能性があります。対応する [keyring_oci](#) システム変数を設定する場合は、その部分を省略します。

たとえば、[keyring_oci8](#) をロードして構成するには、サーバーの `my.cnf` ファイルで次の行を使用します (必要に応じて、プラットフォームの `.so` 接尾辞とファイルの場所を調整します):

```
[mysqld]
early-plugin-load=keyring_oci.so
```

```
keyring_oci_user=ocid1.user.oc1..longAlphaNumericString
keyring_oci_tenancy=ocid1.tenancy.oc1..longAlphaNumericString
keyring_oci_compartment=ocid1.compartment.oc1..longAlphaNumericString
keyring_oci_virtual_vault=ocid1.vault.oc1.iad.shortAlphaNumericString.longAlphaNumericString
keyring_oci_master_key=ocid1.key.oc1.iad.shortAlphaNumericString.longAlphaNumericString
keyring_oci_encryption_endpoint=shortAlphaNumericString-crypto.kms.us-ashburn-1.oraclecloud.com
keyring_oci_management_endpoint=shortAlphaNumericString-management.kms.us-ashburn-1.oraclecloud.com
keyring_oci_vaults_endpoint=vaults.us-ashburn-1.oci.oraclecloud.com
keyring_oci_secrets_endpoint=secrets.vaults.us-ashburn-1.oci.oraclecloud.com
keyring_oci_key_file=file_name
keyring_oci_key_fingerprint=12:34:56:78:90:ab:cd:ef:12:34:56:78:90:ab:cd:ef
```

`keyring_oci` プラグイン固有のシステム変数の詳細は、[セクション6.4.4.13「キーリングシステム変数」](#)を参照してください。

`keyring_oci` プラグインは実行時再構成をサポートしておらず、そのシステム変数は実行時に変更できません。構成パラメータを変更するには、次のようにします:

- `my.cnf` ファイルのパラメータ設定を変更するか、`mysqld-auto.conf` に永続化されるパラメータに `SET PERSIST_ONLY` を使用します。
- サーバーを再起動します。

6.4.4.8 サポートされているキーリングキーのタイプと長さ

MySQL キーリングでは、様々なタイプ (暗号化アルゴリズム) および長さのキーがサポートされます:

- 使用可能なキータイプは、インストールされているキープラグインによって異なります。
- 許可されるキーの長さには、複数の要因があります:
 - 一般的なキーリング UDF インタフェース制限 ([セクション6.4.4.10「汎用キーリングキー管理関数」](#)で説明されているいずれかのキーリング UDF を使用して管理されるキーの場合)、またはバックエンド実装からの制限。これらの長さ制限は、キー操作タイプによって異なる場合があります。
 - 一般的な制限に加えて、個々のプラグインではキータイプごとにキーの長さに制限が課される場合があります。

[表6.26「一般的なキーリングキーの長さ制限」](#)には、一般的なキー長制限が表示されます。(`keyring_aws` の下限は、キーリング UDF ではなく AWS KMS インタフェースによって課されます。) [表6.27「プラグインのキータイプと長さのキー設定」](#)は、キーリングプラグインごとに、許可されているキータイプおよびプラグイン固有のキー長制限を表示します。

表 6.26 一般的なキーリングキーの長さ制限

キー操作	最大キー長
キーの生成	16,384 バイト (MySQL 8.0.18 より前の 2,048)、 <code>keyring_aws</code> の場合 1,024
ストアキー	16,384 バイト (MySQL 8.0.18 より前の 2,048)、 <code>keyring_aws</code> の場合 4,096
Fetch key	16,384 バイト (MySQL 8.0.18 より前の 2,048)、 <code>keyring_aws</code> の場合 4,096

表 6.27 プラグインのキータイプと長さのキー設定

プラグイン名	許可されたキータイプ	プラグイン固有の長さの制限
<code>keyring_aws</code>	AES	16 バイト、24 バイトまたは 32 バイト
	SECRET	なし
<code>keyring_encrypted_file</code>	AES	なし
	DSA	なし
	RSA	なし

プラグイン名	許可されたキータイプ	プラグイン固有の長さの制限
	SECRET	なし
keyring_file	AES DSA RSA SECRET	なし なし なし なし
keyring_hashicorp	AES DSA RSA SECRET	なし なし なし なし
keyring_oci	AES	16 バイト、24 バイトまたは 32 バイト
keyring_okv	AES SECRET	16 バイト、24 バイトまたは 32 バイト なし

MySQL 8.0.19 で使用可能な **SECRET** キータイプは、MySQL キーリングを使用した機密データの汎用的な格納を目的としており、ほとんどのキーリングプラグインでサポートされています。キーリングは、格納および取得時に **SECRET** データをバイトストリームとして暗号化および復号化します。

SECRET キータイプを含むキーリング操作の例:

```
SELECT keyring_key_generate('MySecret1', 'SECRET', 20);
SELECT keyring_key_remove('MySecret1');

SELECT keyring_key_store('MySecret2', 'SECRET', 'MySecretData');
SELECT keyring_key_fetch('MySecret2');
SELECT keyring_key_length_fetch('MySecret2');
SELECT keyring_key_type_fetch('MySecret2');
SELECT keyring_key_remove('MySecret2');
```

6.4.4.9 キーリングキーストア間のキーの移行

MySQL サーバーは、基礎となるキーリングキーストア間での鍵の移行を可能にする操作モードをサポートしているため、DBA は MySQL インストールにあるキーリングプラグインから別のキーリングプラグインに切り替えることができます。移行サーバー (キー移行モードで起動されたサーバー) は、クライアント接続を受け入れません。かわりに、キーの移行に十分な時間だけ実行され、終了します。移行サーバーは、コンソールにエラーを報告します (標準エラー出力)。

オフラインまたはオンラインのキー移行を実行できます:

- ローカルホストで実行中のサーバーがソースキーストアまたは宛先キーストアを使用していないことが確実な場合は、オフライン移行が可能です。この場合、移行中にサーバーを実行してキーストアコンテンツを変更することなく、移行サーバーはキーストアを変更できます。
- ローカルホストで実行中のサーバーがソースキーストアまたは宛先キーストアを使用している場合は、オンライン移行を実行する必要があります。この場合、移行サーバーは実行中のサーバーに接続し、キー移行の進行中にキーリング操作を一時停止するように指示します。

キーの移行操作の結果、移行前のキーとソースキーストアのキーが宛先キーストアに含まれます。キーは移動されずにコピーされるため、ソースキーストアは移行の前後と同じです。コピーするキーが宛先キーストアにすでに存在する場合は、エラーが発生し、宛先キーストアは事前統合状態にリストアされます。

キー移行モードでサーバーを起動するユーザーは、**root** オペレーティングシステムユーザーではなく、キーリングファイルの読取りおよび書込み権限を持っている必要があります。

鍵の移行操作を実行するには、必要な鍵の移行オプションを決定します。移行オプションは、関係するキーリングプラグインと、オフライン移行とオンライン移行のどちらを実行するかを示します:

- ソースおよび宛先のキーリングプラグインを指定するには、次のオプションを指定します:

- `--keyring-migration-source`: 移行するキーを管理するソースキープラグイン。
- `--keyring-migration-destination`: 移行された鍵のコピー先の宛先キーリングプラグイン。

これらのオプションは、キー移行モードで実行するようにサーバーに指示します。両方のオプションは、すべてのキー移行操作に必須です。ソースプラグインと移行先プラグインは異なる必要があり、移行サーバーは両方のプラグインをサポートする必要があります。

- オフライン移行の場合、追加のキー移行オプションは必要ありません。

警告

実行中のサーバーで使用されているキーストアを含むオフライン移行を実行しないでください。

- オンライン移行の場合、現在実行中の一部のサーバーがソースキーストアまたは宛先キーストアを使用しています。実行中のサーバーへの接続方法を示す主要な移行オプションを指定します。これは、移行サーバーが実行中のサーバーに接続し、移行操作中にキーリングの使用を一時停止するように指示するために必要です。

次のいずれかのオプションを使用することは、オンライン移行を意味します:

- `--keyring-migration-host`: 実行中のサーバーがあるホスト。これは常にローカルホストです。
- `--keyring-migration-user`, `--keyring-migration-password`: 実行中のサーバーへの接続に使用するアカウントのユーザー名とパスワード。
- `--keyring-migration-port`: TCP/IP 接続の場合、実行中のサーバー上の接続先のポート番号。
- `--keyring-migration-socket`: Unix ソケットファイルまたは Windows 名前付きパイプ接続の場合、実行中のサーバーで接続するソケットファイルまたは名前付きパイプ。

キー移行オプションの詳細は、[セクション6.4.4.12「キーリングコマンドのオプション」](#)を参照してください。

前述のとおり決定されたキー移行オプションを使用して(場合によっては他のオプションを使用して)、移行サーバーを起動します。次の考慮事項に注意してください:

- 2つのキーリングプラグインのほかの構成パラメータなど、ほかのサーバーオプションが必要になることがあります。たとえば、`keyring_file` がプラグインのいずれかである場合、キーリングデータファイルの場所がデフォルトの場所でないときは、`keyring_file_data` システム変数を設定する必要があります。その他の非キーリングオプションも必要な場合があります。これらのオプションを指定する方法の1つは、`--defaults-file` を使用して、必要なオプションを含むオプションファイルに名前を付けることです。
- MySQL の実行に通常使用されるシステムアカウントとは異なるシステムアカウントから移行サーバーを起動すると、通常の操作中にサーバーからアクセスできないキーリングディレクトリまたはファイルが作成される場合があります。通常、`mysqld` は `mysql` オペレーティングシステムユーザーとして実行されますが、`isabel` としてログインしている間に移行サーバーを起動するとします。移行サーバーによって作成された新しいディレクトリまたはファイルは、`isabel` によって所有されます。`mysql` オペレーティングシステムユーザーとして実行されているサーバーが、`isabel` が所有するファイルシステムオブジェクトにアクセスしようとすると、後続の起動が失敗します。

この問題を回避するには、`root` オペレーティングシステムユーザーとして移行サーバーを起動し、`--user=user_name` オプションを指定します。`user_name` は、MySQL の実行に通常使用されるシステムアカウントです。

- 移行サーバーでは、パス名オプションの値はフルパスである必要があります。相対パス名が予期したとおりに解決されない場合があります。

オフラインキー移行のコマンドラインの例:

```
mysqld --defaults-file=/usr/local/mysql/etc/my.cnf
```

```
--keyring-migration-source=keyring_file.so  
--keyring-migration-destination=keyring_encrypted_file.so  
--keyring_encrypted_file_password=password
```

オンラインキー移行のコマンドラインの例:

```
mysqld --defaults-file=/usr/local/mysql/etc/my.cnf  
--keyring-migration-source=keyring_file.so  
--keyring-migration-destination=keyring_encrypted_file.so  
--keyring_encrypted_file_password=password  
--keyring-migration-host=localhost  
--keyring-migration-user=root  
--keyring-migration-password=root_password
```

キー移行サーバーは、次のように移行操作を実行します:

1. (オンライン移行のみ) 接続オプションを使用して、実行中のサーバーに接続します。接続に使用するアカウントには、グローバル `keyring_operations` システム変数 (`SYSTEM_VARIABLES_ADMIN` または非推奨の `SUPER` 権限に加えて `ENCRYPTION_KEY_ADMIN`) を変更するために必要な権限が必要です。
2. (オンライン移行のみ) 実行中のサーバーで `keyring_operations` を無効にします。(実行中のサーバーが `keyring_operations` をサポートしている必要があります。)
3. ソースおよび宛先のキーリングプラグインをロードします。
4. ソースキーリングから宛先キーリングにキーをコピーします。
5. キーリングプラグインをアンロードします。
6. (オンライン移行のみ) 実行中のサーバーで `keyring_operations` を有効にします。
7. (オンライン移行のみ) 実行中のサーバーから切断します。
8. Exit.

キーの移行中にエラーが発生した場合、宛先プラグインにコピーされたキーは削除され、宛先キーストアは変更されません。

重要

オンライン移行操作の場合、移行サーバーは、実行中のサーバーで `keyring_operations` を有効または無効にします。移行サーバーが異常終了した場合(強制終了された場合など)、実行中のサーバーで `keyring_operations` が再度有効化されず、キーリング操作を実行できない可能性があります。この場合、実行中のサーバーに接続し、`keyring_operations` を手動で有効にする必要がある場合があります。

オンラインキー移行操作が成功した後、実行中のサーバーの再起動が必要になる場合があります:

- 実行中のサーバーがソースキーストアを使用していた場合は、移行後に再起動する必要はありません。
- 移行前に実行中のサーバーがソースキーストアを使用していたが、移行後に宛先キーストアを使用する必要がある場合は、宛先キープラグインを使用するように再構成して再起動する必要があります。
- 実行中のサーバーが宛先キーストアを使用していて、それを引き続き使用している場合は、移行後に再起動して、移行先キーストアに移行されたすべてのキーをロードする必要があります。

注記

MySQL サーバーのキー移行モードでは、単一の実行中のサーバーの一時停止がサポートされます。関連するキーストアが複数のキーサーバーで使用されている場合にキーの移行を実行するには、次の手順を使用します:

1. 実行中の各サーバーに手動で接続し、`keyring_operations=OFF` を設定します。
2. 移行サーバーを使用して、オフラインキー移行を実行します。

3. 実行中の各サーバーに手動で接続し、`keyring_operations=ON` を設定します。

実行中のすべてのサーバーは、`keyring_operations=ON` システム変数をサポートする必要があります。

6.4.4.10 汎用キーリングキー管理関数

MySQL Server は、内部コンポーネントおよびプラグインが機密情報を安全に格納して後で取得できるようにするキーリングサービスをサポートしています。

MySQL Server には、キーリングキー管理用の SQL インタフェースも含まれており、内部キーリングサービスによって提供される関数にアクセスする一連の汎用ユーザー定義関数 (UDF) として実装されています。キーリング UDF はプラグインライブラリファイルに含まれており、UDF を起動する前に有効にする必要がある `keyring_udf` プラグインも含まれています。これらの UDF を使用するには、`keyring_file` や `keyring_okv` などのキーリングプラグインを有効にする必要があります。

ここで説明する UDF は一般的な目的であり、任意のキーリングプラグインでの使用を目的としています。特定のキーリングプラグインには、そのプラグインでのみ使用することを目的とした独自の UDF がある場合があります。[セクション 6.4.4.11 「プラグイン固有のキーリングキー管理関数」](#) を参照してください。

次の各セクションでは、UDF のキー設定のインストール手順を示し、それらの使用方法を示します。UDF によって呼び出されるキーリングサービス関数の詳細は、[セクション 5.6.8.2 「キーリングサービス」](#) を参照してください。一般的なキーリング情報については、[セクション 6.4.4 「MySQL キーリング」](#) を参照してください。

- [汎用キーリング関数のインストールまたはアンインストール](#)
- [汎用キーリング関数の使用](#)
- [汎用キーリング関数リファレンス](#)

汎用キーリング関数のインストールまたはアンインストール

このセクションでは、`keyring_udf` プラグインも含むプラグインライブラリファイルに実装されているキーリングユーザー定義関数 (UDF) をインストールまたはアンインストールする方法について説明します。プラグインおよび UDF のインストールまたはアンインストールの一般情報は、[セクション 5.6.1 「プラグインのインストールおよびアンインストール」](#) および [セクション 5.7.1 「ユーザー定義関数のインストールおよびアンインストール」](#) を参照してください。

キーリング UDF を使用すると、キーリングキー管理操作が可能になりますが、UDF が正しく機能しないため、`keyring_udf` プラグインもインストールする必要があります。`keyring_udf` プラグインなしで UDF を使用しようとすると、エラーが発生します。

サーバーで使用できるようにするには、プラグインライブラリファイルを MySQL プラグインディレクトリ (`plugin_dir` システム変数で指定されたディレクトリ) に配置する必要があります。必要に応じて、サーバーの起動時に `plugin_dir` の値を設定してプラグインディレクトリの場所を構成します。

プラグインライブラリファイルのベース名は `keyring_udf` です。ファイル名の接尾辞は、プラットフォームごとに異なります (たとえば、`.so` for Unix and Unix-like systems, `.dll` for Windows)。

`keyring_udf` プラグインおよび UDF をインストールするには、必要に応じてプラットフォームの `.so` 接尾辞を調整して、`INSTALL PLUGIN` および `CREATE FUNCTION` ステートメントを使用します:

```
INSTALL PLUGIN keyring_udf SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_generate RETURNS INTEGER
  SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_fetch RETURNS STRING
  SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_length_fetch RETURNS INTEGER
  SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_type_fetch RETURNS STRING
  SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_store RETURNS INTEGER
  SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_remove RETURNS INTEGER
```

```
SONAME 'keyring_udf.so';
```

プラグインと UDF がソースレプリケーションサーバーで使用されている場合は、レプリケーションの問題を回避するために、それらをすべてのレプリカにインストールします。

前述のようにインストールすると、プラグインと UDF はアンインストールされるまでインストールされたままになります。これらを削除するには、[UNINSTALL PLUGIN](#) および [DROP FUNCTION](#) ステートメントを使用します:

```
UNINSTALL PLUGIN keyring_udf;  
DROP FUNCTION keyring_key_generate;  
DROP FUNCTION keyring_key_fetch;  
DROP FUNCTION keyring_key_length_fetch;  
DROP FUNCTION keyring_key_type_fetch;  
DROP FUNCTION keyring_key_store;  
DROP FUNCTION keyring_key_remove;
```

汎用キーリング関数の使用

キーリングユーザー定義関数 (UDF) を使用する前に、[汎用キーリング関数のインストールまたはアンインストール](#) の指示に従ってインストールします。

キーリング UDF には、次の制約があります:

- 任意のキーリング UDF を使用するには、[keyring_udf](#) プラグインを有効にする必要があります。それ以外の場合には、次のエラーが発生します:

```
ERROR 1123 (HY000): Can't initialize function 'keyring_key_generate':  
This function requires keyring_udf plugin which is not installed.  
Please install
```

[keyring_udf](#) プラグインをインストールするには、[汎用キーリング関数のインストールまたはアンインストール](#) を参照してください。

- キーリング UDF は、キーリングサービス関数を呼び出します ([セクション5.6.8.2「キーリングサービス」](#) を参照)。次に、サービス関数はインストールされているキーリングプラグイン ([keyring_file](#) や [keyring_okv](#) など) を使用します。したがって、任意のキーリング UDF を使用するには、基礎となるキーリングプラグインの一部を有効にする必要があります。それ以外の場合には、次のエラーが発生します:

```
ERROR 3188 (HY000): Function 'keyring_key_generate' failed because  
underlying keyring service returned an error. Please check if a  
keyring plugin is installed and that provided arguments are valid  
for the keyring you are using.
```

キーリングプラグインをインストールするには、[セクション6.4.4.1「キーリングプラグインのインストール」](#) を参照してください。

- 任意のキーリング UDF を使用するには、ユーザーはグローバル [EXECUTE](#) 権限を持っている必要があります。それ以外の場合には、次のエラーが発生します:

```
ERROR 1123 (HY000): Can't initialize function 'keyring_key_generate':  
The user is not privileged to execute this function. User needs to  
have EXECUTE
```

グローバル [EXECUTE](#) 権限をユーザーに付与するには、次のステートメントを使用します:

```
GRANT EXECUTE ON *.* TO user;
```

または、ユーザーに特定のキー管理操作へのアクセスを許可しながらグローバル [EXECUTE](#) 権限を付与しないようにする場合は、「ラッパー」ストアプログラムを定義できます (このセクションの後半で説明します)。

- 特定のユーザーがキーリングに格納したキーは、後で同じユーザーのみが操作できます。つまり、キー操作時の [CURRENT_USER\(\)](#) 関数の値は、キーリングにキーが格納されたときと同じ値である必要があります。(この制約は、テーブルスペースの暗号化をサポートするために [InnoDB](#) によって作成されたキーなど、インスタンス全体のキーを操作するためのキーリング UDF の使用を除外します。)

複数のユーザーが同じキーに対して操作を実行できるようにするには、「ラッパー」ストアプログラムを定義します (このセクションの後半で説明します)。

- キーリング UDF では、基礎となるキーリングプラグインでサポートされるキーのタイプと長さがサポートされます。特定のキーリングプラグインに固有の鍵については、[セクション6.4.4.8「サポートされているキーリングキーのタイプと長さ」](#)を参照してください。

新しいランダムキーを作成してキーリングに格納するには、キータイプ (暗号化メソッド) およびその長さ (バイト単位) とともに、`keyring_key_generate()` をコールしてキーの ID を渡します。次のコールは、`MyKey` という名前の 2,048-bit DSA 暗号化キーを作成します:

```
mysql> SELECT keyring_key_generate('MyKey', 'DSA', 256);
+-----+
| keyring_key_generate('MyKey', 'DSA', 256) |
+-----+
| 1 |
+-----+
```

戻り値 1 は成功を示します。キーを作成できない場合、戻り値は `NULL` であり、エラーが発生します。これは、基礎となるキーリングプラグインが、指定された鍵タイプと鍵長の組み合わせをサポートしていないことが原因である可能性があります。[セクション6.4.4.8「サポートされているキーリングキーのタイプと長さ」](#)を参照してください。

エラーが発生したかどうかに関係なく戻り型をチェックできるようにするには、`SELECT ... INTO @var_name` を使用して変数値をテストします:

```
mysql> SELECT keyring_key_generate("", "", -1) INTO @x;
ERROR 3188 (HY000): Function 'keyring_key_generate' failed because
underlying keyring service returned an error. Please check if a
keyring plugin is installed and that provided arguments are valid
for the keyring you are using.
mysql> SELECT @x;
+-----+
| @x |
+-----+
| NULL |
+-----+
mysql> SELECT keyring_key_generate('x', 'AES', 16) INTO @x;
mysql> SELECT @x;
+-----+
| @x |
+-----+
| 1 |
+-----+
```

この手法は、失敗した場合に値とエラーを返す他のキーリング UDF にも適用されます。

`keyring_key_generate()` に渡される ID は、後続の UDF コールでキーを参照する手段を提供します。たとえば、キー ID を使用して、そのタイプを文字列またはバイト単位の長さとして整数として取得します:

```
mysql> SELECT keyring_key_type_fetch('MyKey');
+-----+
| keyring_key_type_fetch('MyKey') |
+-----+
| DSA |
+-----+
mysql> SELECT keyring_key_length_fetch('MyKey');
+-----+
| keyring_key_length_fetch('MyKey') |
+-----+
| 256 |
+-----+
```

キー値を取得するには、キー ID を `keyring_key_fetch()` に渡します。次の例では、印刷できない文字が含まれている可能性があるため、`HEX()` を使用してキー値を表示します。この例では、簡潔にするために短いキーも使用していますが、長いキーを使用するとセキュリティが向上することに注意してください:

```
mysql> SELECT keyring_key_generate('MyShortKey', 'DSA', 8);
+-----+
| keyring_key_generate('MyShortKey', 'DSA', 8) |
+-----+
| 1 |
+-----+
mysql> SELECT HEX(keyring_key_fetch('MyShortKey'));
```



```

+-----+
| HEX(keyring_key_fetch('MyShortKey')) |
+-----+
| 1DB3B0FC3328A24C          |
+-----+

```

キーリング UDF では、キー ID、タイプおよび値がバイナリ文字列として扱われるため、比較では大/小文字が区別されます。たとえば、`MyKey` と `mykey` の ID は異なるキーを参照します。

キーを削除するには、キー ID を `keyring_key_remove()` に渡します:

```

mysql> SELECT keyring_key_remove('MyKey');
+-----+
| keyring_key_remove('MyKey') |
+-----+
| 1 |
+-----+

```

指定したキーを不明瞭化して格納するには、キー ID、タイプおよび値を `keyring_key_store()` に渡します:

```

mysql> SELECT keyring_key_store('AES_key', 'AES', 'Secret string');
+-----+
| keyring_key_store('AES_key', 'AES', 'Secret string') |
+-----+
| 1 |
+-----+

```

前述のように、ユーザーはキーリング UDF をコールするためのグローバル `EXECUTE` 権限を持っている必要があり、キーリングにキーを最初に格納するユーザーは、UDF コールごとに有効な `CURRENT_USER()` 値から決定される、後でキーに対して後続の操作を実行するユーザーと同じである必要があります。グローバル `EXECUTE` 権限を持たないユーザー、またはキー「owner,」ではないユーザーにキー操作を許可するには、次の方法を使用します:

1. 必要なキー操作をカプセル化し、`DEFINER` 値がキー所有者と等しい「ラッパー」ストアードプログラムを定義します。
2. 特定のストアードプログラムを起動できるようにする個々のユーザーに、それらのストアードプログラムに対する `EXECUTE` 権限を付与します。
3. ラッパーストアードプログラムによって実装される操作にキー作成が含まれていない場合は、ストアードプログラム定義で `DEFINER` として指定されたアカウントを使用して、必要なキーを事前に作成します。

この手法を使用すると、キーをユーザー間で共有でき、グローバル権限を付与せずに、誰がキーを使用して何を実行できるかを DBA がよりきめ細かく制御できます。

次の例は、DBA が所有する `SharedKey` という名前の共有キーと、現在のキー値へのアクセスを提供する `get_shared_key()` ストアドファンクションの設定方法を示しています。この値は、`key_schema` スキーマに作成された、その関数に対する `EXECUTE` 権限を持つすべてのユーザーが取得できます。

MySQL 管理アカウント (この例では `'root'@'localhost'`) から、キーにアクセスするための管理スキーマおよびストアードファンクションを作成します:

```

mysql> CREATE SCHEMA key_schema;

mysql> CREATE DEFINER = 'root'@'localhost'
  FUNCTION key_schema.get_shared_key()
  RETURNS BLOB READS SQL DATA
  RETURN keyring_key_fetch('SharedKey');

```

管理アカウントから、共有キーが存在することを確認します:

```

mysql> SELECT keyring_key_generate('SharedKey', 'DSA', 8);
+-----+
| keyring_key_generate('SharedKey', 'DSA', 8) |
+-----+
| 1 |
+-----+

```

管理アカウントから、キーアクセス権を付与する通常のユーザーアカウントを作成します:

```
mysql> CREATE USER 'key_user'@'localhost'
IDENTIFIED BY 'key_user_pwd';
```

`key_user` アカウントから、適切な `EXECUTE` 権限がない場合、新しいアカウントが共有キーにアクセスできないことを確認します:

```
mysql> SELECT HEX(key_schema.get_shared_key());
ERROR 1370 (42000): execute command denied to user 'key_user'@'localhost'
for routine 'key_schema.get_shared_key'
```

管理アカウントから、ストアドファンクションの `EXECUTE` を `key_user` に付与します:

```
mysql> GRANT EXECUTE ON FUNCTION key_schema.get_shared_key
TO 'key_user'@'localhost';
```

`key_user` アカウントから、キーにアクセスできるようになったことを確認します:

```
mysql> SELECT HEX(key_schema.get_shared_key());
+-----+
| HEX(key_schema.get_shared_key()) |
+-----+
| 9BAFB9E75CEEB013          |
+-----+
```

汎用キーリング関数リファレンス

このセクションでは、汎用キーリングユーザー定義関数 (UDF) ごとに、その目的、コール順序および戻り値について説明します。これらの UDF を起動できる条件の詳細は、[汎用キーリング関数の使用](#) を参照してください。

- `keyring_key_fetch(key_id)`

キー ID を指定すると、キー値を不明瞭化して戻します。

引数:

- `key_id`: キー ID を指定する文字列。

戻り値:

成功した場合は文字列、キーが存在しない場合は `NULL`、失敗した場合は `NULL` およびエラーとしてキー値を返します。

注記

`keyring_key_fetch()` を使用して取得されたキー値は、[セクション6.4.4.8「サポートされているキーリングキーのタイプと長さ」](#) で説明されている一般的なキーリング UDF 制限の対象となります。その長さより長いキー値は、キーリングサービス関数 ([セクション5.6.8.2「キーリングサービス」](#) を参照) を使用して格納できますが、`keyring_key_fetch()` を使用して取得されたキー値は、一般的なキーリング UDF 制限に切り捨てられます。

例:

```
mysql> SELECT keyring_key_generate('RSA_key', 'RSA', 16);
+-----+
| keyring_key_generate('RSA_key', 'RSA', 16) |
+-----+
|          1 |
+-----+
mysql> SELECT HEX(keyring_key_fetch('RSA_key'));
+-----+
| HEX(keyring_key_fetch('RSA_key')) |
+-----+
| 91C2253B696064D3556984B6630F891A |
+-----+
mysql> SELECT keyring_key_type_fetch('RSA_key');
+-----+
| keyring_key_type_fetch('RSA_key') |
```

```
+-----+
| RSA          |
+-----+
mysql> SELECT keyring_key_length_fetch('RSA_key');
+-----+
| keyring_key_length_fetch('RSA_key') |
+-----+
|          16 |
+-----+
```

この例では、印刷できない文字が含まれている可能性があるため、`HEX()` を使用してキー値を表示します。この例では、簡潔にするために短いキーも使用していますが、長いキーを使用するとセキュリティが向上することに注意してください。

- `keyring_key_generate(key_id, key_type, key_length)`

指定された ID、タイプおよび長さの新しいランダムキーを生成し、キーリングに格納します。型と長さの値は、基礎となるキーリングプラグインでサポートされている値と一致している必要があります。 [セクション6.4.4.8「サポートされているキーリングキーのタイプと長さ」](#) を参照してください。

引数:

- `key_id`: キー ID を指定する文字列。
- `key_type`: キータイプを指定する文字列。
- `key_length`: キーの長さをバイト単位で指定する整数。

戻り値:

成功した場合は 1、失敗した場合は `NULL` およびエラーを返します。

例:

```
mysql> SELECT keyring_key_generate('RSA_key', 'RSA', 384);
+-----+
| keyring_key_generate('RSA_key', 'RSA', 384) |
+-----+
|          1 |
+-----+
```

- `keyring_key_length_fetch(key_id)`

キー ID を指定すると、キーの長さを返します。

引数:

- `key_id`: キー ID を指定する文字列。

戻り値:

成功した場合はキーの長さをバイト単位で返し、キーが存在しない場合は `NULL`、失敗した場合は `NULL` およびエラーを返します。

例:

`keyring_key_fetch()` の説明を参照してください。

- `keyring_key_remove(key_id)`

指定された ID のキーをキーリングから削除します。

引数:

- `key_id`: キー ID を指定する文字列。

戻り値:

成功の場合は 1 を返し、失敗の場合は `NULL` を返します。

例:

```
mysql> SELECT keyring_key_remove('AES_key');
+-----+
| keyring_key_remove('AES_key') |
+-----+
| 1 |
+-----+
```

- `keyring_key_store(key_id, key_type, key)`

キーリングにキーを不明瞭化して格納します。

引数:

- `key_id`: キー ID を指定する文字列。
- `key_type`: キータイプを指定する文字列。
- `key`: キー値を指定する文字列。

戻り値:

成功した場合は 1、失敗した場合は `NULL` およびエラーを返します。

例:

```
mysql> SELECT keyring_key_store('new key', 'DSA', 'My key value');
+-----+
| keyring_key_store('new key', 'DSA', 'My key value') |
+-----+
| 1 |
+-----+
```

- `keyring_key_type_fetch(key_id)`

キー ID を指定すると、キータイプを返します。

引数:

- `key_id`: キー ID を指定する文字列。

戻り値:

成功した場合は文字列、キーが存在しない場合は `NULL`、失敗した場合は `NULL` およびエラーとしてキータイプを返します。

例:

`keyring_key_fetch()` の説明を参照してください。

6.4.4.11 プラグイン固有のキーリングキー管理関数

このセクションでは、キーリングプラグイン固有のユーザー定義関数 (UDF) ごとに、その目的、呼び出しシーケンス、および戻り値について説明します。汎用キーリング UDF の詳細は、[セクション6.4.4.10「汎用キーリングキー管理関数」](#) を参照してください。

- `keyring_aws_rotate_cmk()`

関連付けられたキーリングプラグイン: `keyring_aws`

`keyring_aws_rotate_cmk()` は、顧客マスターキー (CMK) をローテーションします。ローテーションでは、AWS KMS が後続のデータキー暗号化操作に使用するキーのみが変更されます。AWS KMS は以前の CMK バージョンを保持するため、以前の CMK を使用して生成されたキーはローテーション後も復号化可能なままになります。

ローテーションによって AWS KMS 内で使用される CMK 値が変更されますが、参照に使用される ID は変更されないため、`keyring_aws_rotate_cmk()` のコール後に `keyring_aws_cmk_id` システム変数を変更する必要はありません。

この UDF には `SUPER` 権限が必要です。

引数:

なし

戻り値:

成功した場合は 1、失敗した場合は `NULL` およびエラーを返します。

- `keyring_aws_rotate_keys()`

関連付けられたキーリングプラグイン: `keyring_aws`

`keyring_aws_rotate_keys()` は、`keyring_aws_data_file` システム変数で指定された `keyring_aws` ストレージファイルに格納されているキーをローテーションします。ローテーションでは、`keyring_aws_cmk_id` システム変数の値を CMK 値として使用して、ファイルに格納されている各キーを AWS KMS に再暗号化のために送信し、新しい暗号化キーをファイルに格納します。

`keyring_aws_rotate_keys()` は、次の場合にキーの再暗号化に役立ちます:

- CMK のローテーション後 (`keyring_aws_rotate_cmk()` UDF の起動後)
- `keyring_aws_cmk_id` システム変数を別のキー値に変更した後

この UDF には `SUPER` 権限が必要です。

引数:

なし

戻り値:

成功した場合は 1、失敗した場合は `NULL` およびエラーを返します。

- `keyring_hashicorp_update_config()`

関連付けられたキーリングプラグイン: `keyring_hashicorp`

`keyring_hashicorp` 構成 で説明されているように、`keyring_hashicorp_update_config()` UDF が呼び出されると、`keyring_hashicorp` はランタイム再構成を実行します。

この UDF ではグローバルシステム変数を変更されるため、`SYSTEM_VARIABLES_ADMIN` 権限が必要です。

引数:

なし

戻り値:

成功した場合は文字列 `'Configuration update was successful.'`、失敗した場合は `'Configuration update failed.'` を返します。

6.4.4.12 キーリングコマンドのオプション

MySQL は、次のキーリング関連のコマンドラインオプションをサポートしています:

- `--keyring-migration-destination=plugin`

コマンド行形式	<code>--keyring-migration-destination=plugin_name</code>
---------	--

型	文字列
---	-----

キー移行用の宛先キーリングプラグイン。 [セクション6.4.4.9「キーリングキーストア間のキーの移行」](#)を参照してください。 オプション値の形式と解釈は、`--keyring-migration-source` オプションで説明されているものと同じです。

注記

`--keyring-migration-source` および `--keyring-migration-destination` は、すべてのキーリング移行操作に必須です。 ソースプラグインと移行先プラグインは異なる必要があり、移行サーバーは両方のプラグインをサポートする必要があります。

- `--keyring-migration-host=host_name`

コマンド行形式	<code>--keyring-migration-host=host_name</code>
型	文字列
デフォルト値	localhost

いずれかのキー移行キーストアを現在使用している実行中のサーバーのホストの場所。 [セクション6.4.4.9「キーリングキーストア間のキーの移行」](#)を参照してください。 移行は常にローカルホストで行われるため、このオプションでは、`localhost`, `127.0.0.1`, `:::1` などのローカルサーバーに接続するための値、またはローカルホストの IP アドレスまたはホスト名を常に指定します。

- `--keyring-migration-password[=password]`

コマンド行形式	<code>--keyring-migration-password[=password]</code>
型	文字列

いずれかのキー移行キーストアを現在使用している実行中のサーバーに接続するためのパスワード。 [セクション6.4.4.9「キーリングキーストア間のキーの移行」](#)を参照してください。 コマンドラインでオプション名の後に `password` 値を省略すると、サーバーによって値の入力が求められます。

コマンド行でのパスワード指定は、セキュアでないと考えるべきです。 [セクション6.1.2.1「パスワードセキュリティのためのエンドユーザーガイドライン」](#)を参照してください。 オプションファイルを使用すれば、コマンド行でパスワードを指定することを回避できます。 この場合、ファイルには制限モードがあり、移行サーバーの実行に使用されるアカウントからのみアクセスできる必要があります。

- `--keyring-migration-port=port_num`

コマンド行形式	<code>--keyring-migration-port=port_num</code>
型	数値
デフォルト値	3306

TCP/IP 接続の場合、キー移行キーストアのいずれかを現在使用している実行中のサーバーに接続するためのポート番号。 [セクション6.4.4.9「キーリングキーストア間のキーの移行」](#)を参照してください。

- `--keyring-migration-socket=path`

コマンド行形式	<code>--keyring-migration-socket={file_name pipe_name}</code>
型	文字列

Unix ソケットファイルまたは Windows 名前付きパイプ接続の場合、キー移行キーストアのいずれかを現在使用している実行中のサーバーに接続するためのソケットファイルまたは名前付きパイプ。 [セクション6.4.4.9「キーリングキーストア間のキーの移行」](#)を参照してください。

- `--keyring-migration-source=plugin`

コマンド行形式	<code>--keyring-migration-source=plugin_name</code>
---------	---

型	文字列
---	-----

キー移行用のソースキープラグイン。 [セクション6.4.4.9「キーリングキーストア間のキーの移行」](#)を参照してください。

このオプションの値は `--plugin-load` の値と似ていますが、指定できるプラグインライブラリが1つだけである点が異なります。この値は、`name = plugin_library` または `plugin_library` として指定されます。`name` はロードするプラグインの名前で、`plugin_library` はプラグインコードを含むライブラリファイルの名前です。プラグインライブラリに先行するプラグイン名を付けずに名前を付けると、サーバーはライブラリ内のすべてのプラグインをロードします。サーバーは、`plugin_dir` システム変数で指定されたディレクトリ内でプラグインライブラリファイルを検索します。

注記

`--keyring-migration-source` および `--keyring-migration-destination` は、すべてのキーリング移行操作に必須です。ソースプラグインと移行先プラグインは異なる必要があり、移行サーバーは両方のプラグインをサポートする必要があります。

- `--keyring-migration-user=user_name`

コマンド行形式	<code>--keyring-migration-user=user_name</code>
型	文字列

いずれかのキー移行キーストアを現在使用している実行中のサーバーに接続するためのユーザー名。 [セクション6.4.4.9「キーリングキーストア間のキーの移行」](#)を参照してください。

6.4.4.13 キーリングシステム変数

MySQL キーリングプラグインは、次のシステム変数をサポートしています。これらを使用して、キーリングプラグイン操作を構成します。これらの変数は、適切なキーリングプラグインがインストールされていないかぎり使用できません ([セクション6.4.4.1「キーリングプラグインのインストール」](#)を参照)。

- `keyring_aws_cmk_id`

コマンド行形式	<code>--keyring-aws-cmk-id=value</code>
システム変数	<code>keyring_aws_cmk_id</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列

AWS KMS サーバーから取得され、`keyring_aws` プラグインで使用される顧客マスターキー (CMK) ID。この変数は、そのプラグインがインストールされないかぎり利用できません。

この変数は必須です。指定しない場合、`keyring_aws` の初期化は失敗します。

- `keyring_aws_conf_file`

コマンド行形式	<code>--keyring-aws-conf-file=file_name</code>
システム変数	<code>keyring_aws_conf_file</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	<code>platform specific</code>

`keyring_aws` プラグインの構成ファイルの場所。この変数は、そのプラグインがインストールされないかぎり利用できません。

プラグインの起動時に、`keyring_aws` は AWS 秘密アクセスキー ID およびキーを構成ファイルから読み取ります。`keyring_aws` プラグインを正常に起動するには、構成ファイルが存在し、[セクション6.4.4.5「keyring_aws Amazon Web Services キーリングプラグインの使用」](#)の説明に従って初期化された有効な秘密アクセスキー情報が含まれている必要があります。

デフォルトのファイル名は `keyring_aws_conf` で、デフォルトのキーリングファイルディレクトリにあります。このデフォルトディレクトリの場所は、`keyring_file_data` システム変数の場所と同じです。ディレクトリを手動で作成する場合に考慮する必要がある考慮事項の詳細は、その変数の説明を参照してください。

- [keyring_aws_data_file](#)

コマンド行形式	<code>--keyring-aws-data-file</code>
システム変数	<code>keyring_aws_data_file</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	<code>platform specific</code>

`keyring_aws` プラグインのストレージファイルの場所。この変数は、そのプラグインがインストールされないかぎり利用できません。

プラグインの起動時に、`keyring_aws_data_file` に割り当てられた値で存在しないファイルが指定された場合、`keyring_aws` プラグインはそれを (必要に応じて親ディレクトリとともに) 作成しようとします。ファイルが存在する場合、`keyring_aws` はファイルに含まれている暗号化キーをメモリー内キャッシュに読み取ります。`keyring_aws` では、暗号化されていないキーはメモリーにキャッシュされません。

デフォルトのファイル名は `keyring_aws_data` で、デフォルトのキーリングファイルディレクトリにあります。このデフォルトディレクトリの場所は、`keyring_file_data` システム変数の場所と同じです。ディレクトリを手動で作成する場合に考慮する必要がある考慮事項の詳細は、その変数の説明を参照してください。

- [keyring_aws_region](#)

コマンド行形式	<code>--keyring-aws-region=value</code>
システム変数	<code>keyring_aws_region</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	<code>us-east-1</code>
有効な値	<code>ap-northeast-1</code> <code>ap-northeast-2</code> <code>ap-south-1</code> <code>ap-southeast-1</code> <code>ap-southeast-2</code> <code>eu-central-1</code>

	eu-west-1
	sa-east-1
	us-east-1
	us-west-1
	us-west-2

`keyring_aws` プラグインの AWS リージョン。この変数は、そのプラグインがインストールされないかぎり利用できません。

- `keyring_encrypted_file_data`

コマンド行形式	<code>--keyring-encrypted-file-data=file_name</code>
システム変数	<code>keyring_encrypted_file_data</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	platform specific

`keyring_encrypted_file` プラグインによってセキュアなデータ記憶域に使用されるデータファイルのパス名。この変数は、そのプラグインがインストールされないかぎり利用できません。ファイルの場所は、キーリングプラグインによってのみ使用されると見なされるディレクトリ内にある必要があります。たとえば、データディレクトリの下にファイルを配置しないでください。

キーリング操作はトランザクションです: `keyring_encrypted_file` プラグインは、書き込み操作中にバックアップファイルを使用して、操作が失敗した場合に元のファイルにロールバックできるようにします。バックアップファイルの名前は、接尾辞が `.backup` の `keyring_encrypted_file_data` システム変数の値と同じです。

複数の MySQL インスタンスに同じ `keyring_encrypted_file` データファイルを使用しないでください。各インスタンスには、独自の一意のデータファイルが必要です。

次のテーブルに示すように、デフォルトのファイル名は `keyring_encrypted` で、プラットフォーム固有のディレクトリにあり、`INSTALL_LAYOUT CMake` オプションの値によって異なります。ソースからビルドする場合にファイルのデフォルトディレクトリを明示的に指定するには、`INSTALL_MYSQLKEYRINGDIR CMake` オプションを使用します。

INSTALL_LAYOUT 値	デフォルトの <code>keyring_encrypted_file_data</code> 値
DEB, RPM, SVR4	<code>/var/lib/mysql-keyring/keyring_encrypted</code>
それ以外の場合	<code>CMAKE_INSTALL_PREFIX</code> 値の下の <code>keyring/keyring_encrypted</code>

プラグインの起動時に、`keyring_encrypted_file_data` に割り当てられた値で存在しないファイルが指定された場合、`keyring_encrypted_file` プラグインはそれを (必要に応じて親ディレクトリとともに) 作成しようとします。

ディレクトリを手動で作成する場合は、制限モードであり、MySQL サーバーの実行に使用されるアカウントからのみアクセスする必要があります。たとえば、Unix および Unix に似たシステムで `/usr/local/mysql/mysql-keyring` ディレクトリを使用するには、次のコマンド (`root` として実行) を実行してディレクトリを作成し、そのモードと所有権を設定します:

```
cd /usr/local/mysql
mkdir mysql-keyring
chmod 750 mysql-keyring
chown mysql mysql-keyring
```

chgrp mysql mysql-keyring

`keyring_encrypted_file` プラグインは、データファイルを作成またはアクセスできない場合、エラーログにエラーメッセージを書き込みます。 `keyring_encrypted_file_data` への実行時割当てが試行されてもエラーが発生した場合、変数値は変更されません。

重要

`keyring_encrypted_file` プラグインがデータファイルを作成して使用を開始したら、ファイルを削除しないことが重要です。ファイルが失われると、キーを使用して暗号化されたデータにアクセスできなくなります。(一致するように `keyring_encrypted_file_data` の値を変更しているかぎり、ファイルの名前を変更したり、ファイルを移動したりできません。)

- `keyring_encrypted_file_password`

コマンド行形式	<code>--keyring-encrypted-file-password=password</code>
システム変数	<code>keyring_encrypted_file_password</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列

`keyring_encrypted_file` プラグインで使用されるパスワード。この変数は、そのプラグインがインストールされないかぎり利用できません。

この変数は必須です。指定しない場合、`keyring_encrypted_file` の初期化は失敗します。

この変数がオプションファイルで指定されている場合、ファイルは制限モードであり、MySQL サーバーの実行に使用されるアカウントからのみアクセスできる必要があります。

重要

`keyring_encrypted_file_password` 値が設定されると、変更してもキーリングパスワードはローテーションされず、サーバーにアクセスできなくなる可能性があります。不正なパスワードが指定された場合、`keyring_encrypted_file` プラグインは暗号化キーリングファイルからキーをロードできません。

表示値が不明瞭化されているため、`SHOW VARIABLES` またはパフォーマンススキーマ `global_variables` テーブルでは実行時にパスワード値を表示できません。

- `keyring_file_data`

コマンド行形式	<code>--keyring-file-data=file_name</code>
システム変数	<code>keyring_file_data</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	<code>platform specific</code>

`keyring_file` プラグインによってセキュアなデータ記憶域に使用されるデータファイルのパス名。この変数は、そのプラグインがインストールされないかぎり利用できません。ファイルの場所は、キーリングプラグインによっての

み使用されると見なされるディレクトリ内にある必要があります。たとえば、データディレクトリの下にファイルを配置しないでください。

キーリング操作はトランザクションです: `keyring_file` プラグインは、書き込み操作中にバックアップファイルを使用して、操作が失敗した場合に元のファイルにロールバックできるようにします。バックアップファイルの名前は、接尾辞が `.backup` の `keyring_file_data` システム変数の値と同じです。

複数の MySQL インスタンスに同じ `keyring_file` データファイルを使用しないでください。各インスタンスには、独自の一意のデータファイルが必要です。

次のテーブルに示すように、デフォルトのファイル名は `keyring` で、プラットフォーム固有のディレクトリにあり、`INSTALL_LAYOUT CMake` オプションの値によって異なります。ソースからビルドする場合にファイルのデフォルトディレクトリを明示的に指定するには、`INSTALL_MYSQLKEYRINGDIR CMake` オプションを使用します。

<code>INSTALL_LAYOUT</code> 値	デフォルトの <code>keyring_file_data</code> 値
<code>DEB, RPM, SVR4</code>	<code>/var/lib/mysql-keyring/keyring</code>
それ以外の場合	<code>CMAKE_INSTALL_PREFIX</code> 値の下の <code>keyring/keyring</code>

プラグインの起動時に、`keyring_file_data` に割り当てられた値で存在しないファイルが指定された場合、`keyring_file` プラグインはそれを (必要に応じて親ディレクトリとともに) 作成しようとします。

ディレクトリを手動で作成する場合は、制限モードであり、MySQL サーバーの実行に使用されるアカウントからのみアクセスできる必要があります。たとえば、Unix および Unix に似たシステムで `/usr/local/mysql/mysql-keyring` ディレクトリを使用するには、次のコマンド (`root` として実行) を実行してディレクトリを作成し、そのモードと所有権を設定します:

```
cd /usr/local/mysql
mkdir mysql-keyring
chmod 750 mysql-keyring
chown mysql mysql-keyring
chgrp mysql mysql-keyring
```

`keyring_file` プラグインは、データファイルを作成またはアクセスできない場合、エラーログにエラーメッセージを書き込みます。 `keyring_file_data` への実行時割当てが試行されてもエラーが発生した場合、変数値は変更されません。

重要

`keyring_file` プラグインがデータファイルを作成して使用を開始したら、ファイルを削除しないことが重要です。たとえば、`InnoDB` は、このファイルを使用して、`InnoDB` テーブルスペース暗号化を使用するテーブルのデータの復号化に使用されるマスターキーを格納します。 [セクション15.13「InnoDB 保存データ暗号化」](#) を参照してください。ファイルが失われると、このようなテーブルのデータにアクセスできなくなります。(一致するように `keyring_file_data` の値を変更しているかぎり、ファイルの名前を変更したり、ファイルを移動したりできます。) 最初の暗号化されたテーブルを作成した直後、およびマスターキーのローテーションの前後に、キーリングデータファイルの個別のバックアップを作成することをお勧めします。

- `keyring_hashicorp_auth_path`

コマンド行形式	<code>--keyring-hashicorp-auth-path=value</code>
導入	8.0.18
システム変数	<code>keyring_hashicorp_auth_path</code>
スコープ	グローバル
動的	はい
<code>SET_VAR</code> ヒントの適用	いいえ
型	文字列

デフォルト値	/v1/auth/approle/login
--------	------------------------

[keyring_hashicorp](#) プラグインで使用するために、HashiCorp Vault サーバー内で AppRole 認証が有効になっている認証パス。この変数は、そのプラグインがインストールされないかぎり利用できません。

- [keyring_hashicorp_ca_path](#)

コマンド行形式	--keyring-hashicorp-ca-path=file_name
導入	8.0.18
システム変数	keyring_hashicorp_ca_path
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	empty string

[keyring_hashicorp](#) プラグインで使用する適切にフォーマットされた TLS 認証局を含む、MySQL サーバーにアクセス可能なローカルファイルの絶対パス名。この変数は、そのプラグインがインストールされないかぎり利用できません。

この変数が設定されていない場合、[keyring_hashicorp](#) プラグインはサーバー証明書の検証を使用せずに HTTPS 接続を開き、HashiCorp Vault サーバーによって配信された証明書を信頼します。これを安全にするには、Vault サーバが悪意のないものと想定し、中間者攻撃が発生しないようにする必要があります。これらの仮定が無効な場合は、[keyring_hashicorp_ca_path](#) を信頼できる CA 証明書のパスに設定します。(たとえば、[証明書とキーの準備](#) の手順の場合、これは `company.crt` ファイルです。)

- [keyring_hashicorp_caching](#)

コマンド行形式	--keyring-hashicorp-caching[={OFF ON}]
導入	8.0.18
システム変数	keyring_hashicorp_caching
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

[keyring_hashicorp](#) プラグインが HashiCorp Vault サーバーからキーをキャッシュするために使用するオプションのインメモリキーキャッシュを有効にするかどうか。この変数は、そのプラグインがインストールされないかぎり利用できません。キャッシュが有効になっている場合、プラグインは初期化中にキャッシュを移入します。それ以外の場合、プラグインは初期化中にキーリストのみを移入します。

キャッシュの有効化は危険です: パフォーマンスは向上しますが、機密キー情報のコピーはメモリー内に保持されるため、セキュリティ上の目的では望ましくない場合があります。

- [keyring_hashicorp_commit_auth_path](#)

導入	8.0.18
システム変数	keyring_hashicorp_commit_auth_path
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ

型	文字列
---	-----

この変数は、[keyring_hashicorp](#) プラグインの初期化時に値を取得する [keyring_hashicorp_auth_path](#) に関連付けられています。この変数は、そのプラグインがインストールされないかぎり利用できません。これには、初期化が成功した場合にプラグイン操作に実際に使用される「committed」値が反映されます。追加情報については [keyring_hashicorp 構成](#) を参照してください。

- [keyring_hashicorp_commit_ca_path](#)

導入	8.0.18
システム変数	keyring_hashicorp_commit_ca_path
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

この変数は、[keyring_hashicorp](#) プラグインの初期化時に値を取得する [keyring_hashicorp_ca_path](#) に関連付けられています。この変数は、そのプラグインがインストールされないかぎり利用できません。これには、初期化が成功した場合にプラグイン操作に実際に使用される「committed」値が反映されます。追加情報については [keyring_hashicorp 構成](#) を参照してください。

- [keyring_hashicorp_commit_caching](#)

導入	8.0.18
システム変数	keyring_hashicorp_commit_caching
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

この変数は、[keyring_hashicorp](#) プラグインの初期化時に値を取得する [keyring_hashicorp_caching](#) に関連付けられています。この変数は、そのプラグインがインストールされないかぎり利用できません。これには、初期化が成功した場合にプラグイン操作に実際に使用される「committed」値が反映されます。追加情報については [keyring_hashicorp 構成](#) を参照してください。

- [keyring_hashicorp_commit_role_id](#)

導入	8.0.18
システム変数	keyring_hashicorp_commit_role_id
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

この変数は、[keyring_hashicorp](#) プラグインの初期化時に値を取得する [keyring_hashicorp_role_id](#) に関連付けられています。この変数は、そのプラグインがインストールされないかぎり利用できません。これには、初期化が成功した場合にプラグイン操作に実際に使用される「committed」値が反映されます。追加情報については [keyring_hashicorp 構成](#) を参照してください。

- [keyring_hashicorp_commit_server_url](#)

導入	8.0.18	
システム変数	keyring_hashicorp_commit_server_url	1271

スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

この変数は、`keyring_hashicorp` プラグインの初期化時に値を取得する `keyring_hashicorp_server_url` に関連付けられています。この変数は、そのプラグインがインストールされないかぎり利用できません。これには、初期化が成功した場合にプラグイン操作に実際に使用される「committed」値が反映されます。追加情報については [keyring_hashicorp 構成](#) を参照してください。

- `keyring_hashicorp_commit_store_path`

導入	8.0.18
システム変数	keyring_hashicorp_commit_store_path
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

この変数は、`keyring_hashicorp` プラグインの初期化時に値を取得する `keyring_hashicorp_store_path` に関連付けられています。この変数は、そのプラグインがインストールされないかぎり利用できません。これには、初期化が成功した場合にプラグイン操作に実際に使用される「committed」値が反映されます。追加情報については [keyring_hashicorp 構成](#) を参照してください。

- `keyring_hashicorp_role_id`

コマンド行形式	<code>--keyring-hashicorp-role-id=value</code>
導入	8.0.18
システム変数	keyring_hashicorp_role_id
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	<code>empty string</code>

`keyring_hashicorp` プラグインで使用する HashiCorp Vault AppRole 認証ロール ID。この変数は、そのプラグインがインストールされないかぎり利用できません。値は UUID 形式である必要があります。

この変数は必須です。指定しない場合、`keyring_hashicorp` の初期化は失敗します。

- `keyring_hashicorp_secret_id`

コマンド行形式	<code>--keyring-hashicorp-secret-id=value</code>
導入	8.0.18
システム変数	keyring_hashicorp_secret_id
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列

デフォルト値	empty string
--------	--------------

`keyring_hashicorp` プラグインで使用する HashiCorp Vault AppRole 認証シークレット ID。この変数は、そのプラグインがインストールされないかぎり利用できません。値は UUID 形式である必要があります。

この変数は必須です。指定しない場合、`keyring_hashicorp` の初期化は失敗します。

この変数の値は機密であるため、その値は表示時に * 文字によってマスクされます。

- `keyring_hashicorp_server_url`

コマンド行形式	<code>--keyring-hashicorp-server-url=value</code>
導入	8.0.18
システム変数	<code>keyring_hashicorp_server_url</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	<code>https://127.0.0.1:8200</code>

`keyring_hashicorp` プラグインで使用する HashiCorp Vault サーバーの URL。この変数は、そのプラグインがインストールされないかぎり利用できません。値は `https://` で始まる必要があります。

- `keyring_hashicorp_store_path`

コマンド行形式	<code>--keyring-hashicorp-store-path=value</code>
導入	8.0.18
システム変数	<code>keyring_hashicorp_store_path</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	empty string

`keyring_hashicorp` プラグインによって適切な AppRole AppRole 資格証明が提供される場合に書き込み可能な HashiCorp Vault サーバー内のストアパス。この変数は、そのプラグインがインストールされないかぎり利用できません。資格証明を指定するには、`keyring_hashicorp_role_id` および `keyring_hashicorp_secret_id` システム変数を設定します (たとえば、`keyring_hashicorp 構成` を参照)。

この変数は必須です。指定しない場合、`keyring_hashicorp` の初期化は失敗します。

- `keyring_oci_ca_certificate`

コマンド行形式	<code>--keyring-oci-ca-certificate=file_name</code>
導入	8.0.22
システム変数	<code>keyring_oci_ca_certificate</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

デフォルト値	empty string
--------	--------------

`keyring_oci` プラグインが Oracle Cloud Infrastructure 証明書の検証に使用する CA 証明書バンドルファイルのパス名。この変数は、そのプラグインがインストールされないかぎり利用できません。

ファイルにピア検証用の 1 つまたは複数の証明書が含まれています。ファイルが指定されていない場合、システムにインストールされているデフォルトの CA バンドルが使用されます。値が `disabled` の場合 (大/小文字を区別)、`keyring_oci` は証明書の検証を実行しません。

- `keyring_oci_compartment`

コマンド行形式	<code>--keyring-oci-compartment=ocid</code>
導入	8.0.22
システム変数	<code>keyring_oci_compartment</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

`keyring_oci` プラグインが MySQL キーの場所として使用するテナンシコンパートメントの OCID。この変数は、そのプラグインがインストールされないかぎり利用できません。

`keyring_oci` を使用する前に、MySQL コンパートメントまたはサブコンパートメントが存在しない場合は作成する必要があります。このコンパートメントにボルトキーまたはボルトシークレットを含めることはできません。MySQL Keyring 以外のシステムでは使用しないでください。

コンパートメントの管理および OCID の取得の詳細は、「[コンパートメントの管理](#)」を参照してください。

この変数は必須です。指定しない場合、`keyring_oci` の初期化は失敗します。

- `keyring_oci_encryption_endpoint`

コマンド行形式	<code>--keyring-oci-encryption-endpoint=value</code>
導入	8.0.22
システム変数	<code>keyring_oci_encryption_endpoint</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

`keyring_oci` プラグインが新しいキーの暗号文を生成するために使用する Oracle Cloud Infrastructure 暗号化サーバーのエンドポイント。この変数は、そのプラグインがインストールされないかぎり利用できません。

暗号化エンドポイントはボルト固有であり、Oracle Cloud Infrastructure によってボルト作成時に割り当てられます。エンドポイント OCID を取得するには、「[ボルトの管理](#)」の手順を使用して、`keyring_oci` ボルトの構成詳細を表示します。

この変数は必須です。指定しない場合、`keyring_oci` の初期化は失敗します。

- `keyring_oci_key_file`

コマンド行形式	<code>--keyring-oci-key-file=file_name</code>
導入	8.0.22
システム変数	<code>keyring_oci_key_file</code>
スコープ	グローバル

動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

`keyring_oci` プラグインが Oracle Cloud Infrastructure 認証に使用する RSA 秘密キーを含むファイルのパス名。この変数は、そのプラグインがインストールされないかぎり利用できません。

コンソールを使用して、対応する RSA 公開キーもアップロードする必要があります。コンソールには、`keyring_oci_key_fingerprint` システム変数の設定に使用できるキーフィンガープリント値が表示されます。

API キーの生成およびアップロードの詳細は、「[必要なキーおよび OCID](#)」を参照してください。

この変数は必須です。指定しない場合、`keyring_oci` の初期化は失敗します。

- `keyring_oci_key_fingerprint`

コマンド行形式	<code>--keyring-oci-key-fingerprint=value</code>
導入	8.0.22
システム変数	<code>keyring_oci_key_fingerprint</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

`keyring_oci` プラグインが Oracle Cloud Infrastructure 認証に使用する RSA 秘密キーのフィンガープリント。この変数は、そのプラグインがインストールされないかぎり利用できません。

API キーの作成中にキーフィンガープリントを取得するには、次のコマンドを実行します:

```
openssl rsa -pubout -outform DER -in ~/.oci/oci_api_key.pem | openssl md5 -c
```

または、RSA 公開キーをアップロードするとフィンガープリントが自動的に表示されるコンソールからフィンガープリントを取得します。

キーフィンガープリントの取得の詳細は、「[必要なキーおよび OCID](#)」を参照してください。

この変数は必須です。指定しない場合、`keyring_oci` の初期化は失敗します。

- `keyring_oci_management_endpoint`

コマンド行形式	<code>--keyring-oci-management-endpoint=value</code>
導入	8.0.22
システム変数	<code>keyring_oci_management_endpoint</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

`keyring_oci` プラグインが既存のキーをリストするために使用する Oracle Cloud Infrastructure キー管理サーバーのエンドポイント。この変数は、そのプラグインがインストールされないかぎり利用できません。

キー管理エンドポイントはポート固有で、Oracle Cloud Infrastructure によってポート作成時に割り当てられます。エンドポイント OCID を取得するには、「[ポートの管理](#)」の手順を使用して、`keyring_oci` ポートの構成詳細を表示します。

この変数は必須です。指定しない場合、`keyring_oci` の初期化は失敗します。

- [keyring_oci_master_key](#)

コマンド行形式	<code>--keyring-oci-master-key=ocid</code>
導入	8.0.22
システム変数	keyring_oci_master_key
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

[keyring_oci](#) プラグインがシークレットの暗号化に使用する Oracle Cloud Infrastructure マスター暗号化キーの OCID。この変数は、そのプラグインがインストールされないかぎり利用できません。

[keyring_oci](#) を使用する前に、Oracle Cloud Infrastructure コンパートメントの暗号化キーを作成する必要があります (存在しない場合)。生成されたキーの MySQL 固有名を指定し、他の目的には使用しないでください。

キーの作成の詳細は、「[キーの管理](#)」を参照してください。

この変数は必須です。指定しない場合、[keyring_oci](#) の初期化は失敗します。

- [keyring_oci_secrets_endpoint](#)

コマンド行形式	<code>--keyring-oci-secrets-endpoint=value</code>
導入	8.0.22
システム変数	keyring_oci_secrets_endpoint
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

[keyring_oci](#) プラグインがシークレットのリスト、作成およびリタイアに使用する Oracle Cloud Infrastructure シークレットサーバーのエンドポイント。この変数は、そのプラグインがインストールされないかぎり利用できません。

シークレットのエンドポイントはポート固有で、Oracle Cloud Infrastructure によってポート作成時に割り当てられます。エンドポイント OCID を取得するには、「[ポートの管理](#)」の手順を使用して、[keyring_oci](#) ポートの構成詳細を表示します。

この変数は必須です。指定しない場合、[keyring_oci](#) の初期化は失敗します。

- [keyring_oci_tenancy](#)

コマンド行形式	<code>--keyring-oci-tenancy=ocid</code>
導入	8.0.22
システム変数	keyring_oci_tenancy
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ

型	文字列
---	-----

`keyring_oci` プラグインが MySQL コンパートメントの場所として使用する Oracle Cloud Infrastructure テナンスの OCID。この変数は、そのプラグインがインストールされないかぎり利用できません。

`keyring_oci` を使用する前に、テナンスが存在しない場合は作成する必要があります。コンソールからテナンス OCID を取得するには、「[必要なキーおよび OCID](#)」の手順を使用します。

この変数は必須です。指定しない場合、`keyring_oci` の初期化は失敗します。

- `keyring_oci_user`

コマンド行形式	<code>--keyring-oci-user=ocid</code>
導入	8.0.22
システム変数	<code>keyring_oci_user</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

`keyring_oci` プラグインがクラウド接続に使用する Oracle Cloud Infrastructure ユーザーの OCID。この変数は、そのプラグインがインストールされないかぎり利用できません。

`keyring_oci` を使用する前に、このユーザーが存在し、構成済の Oracle Cloud Infrastructure テナンス、コンパートメントおよびポールのリソースを使用するためのアクセス権が付与されている必要があります。

コンソールからユーザー OCID を取得するには、「[必要なキーおよび OCID](#)」の手順を使用します。

この変数は必須です。指定しない場合、`keyring_oci` の初期化は失敗します。

- `keyring_oci_vaults_endpoint`

コマンド行形式	<code>--keyring-oci-vaults-endpoint=value</code>
導入	8.0.22
システム変数	<code>keyring_oci_vaults_endpoint</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

`keyring_oci` プラグインがシークレットの値を取得するために使用する Oracle Cloud Infrastructure ポールトサーバーのエンドポイント。この変数は、そのプラグインがインストールされないかぎり利用できません。

ポールのエンドポイントはポールの固有であり、Oracle Cloud Infrastructure によってポールの作成時に割り当てられます。エンドポイント OCID を取得するには、「[ポールの管理](#)」の手順を使用して、`keyring_oci` ポールの構成詳細を表示します。

この変数は必須です。指定しない場合、`keyring_oci` の初期化は失敗します。

- `keyring_oci_virtual_vault`

コマンド行形式	<code>--keyring-oci-virtual-vault=ocid</code>
導入	8.0.22
システム変数	<code>keyring_oci_virtual_vault</code>
スコープ	グローバル

動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

`keyring_oci` プラグインが暗号化操作に使用する Oracle Cloud Infrastructure Vault の OCID。この変数は、そのプラグインがインストールされないかぎり利用できません。

`keyring_oci` を使用する前に、MySQL コンパートメントに新しいポールの存在しない場合は作成する必要があります。(または、MySQL コンパートメントの親コンパートメントにある既存のポールの再利用できます。) コンパートメントユーザーは、それぞれのコンパートメントのキーのみを表示および使用できます。

ポールの作成およびポール OCID の取得の詳細は、「[ポールの管理](#)」を参照してください。

この変数は必須です。指定しない場合、`keyring_oci` の初期化は失敗します。

- `keyring_okv_conf_dir`

コマンド行形式	<code>--keyring-okv-conf-dir=dir_name</code>
システム変数	<code>keyring_okv_conf_dir</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	ディレクトリ名
デフォルト値	<code>empty string</code>

`keyring_okv` プラグインで使用される構成情報を格納するディレクトリのパス名。この変数は、そのプラグインがインストールされないかぎり利用できません。この場所は、`keyring_okv` プラグインでのみ使用されるディレクトリである必要があります。たとえば、データディレクトリの下にディレクトリを配置しないでください。

デフォルトの `keyring_okv_conf_dir` 値は空です。`keyring_okv` プラグインが Oracle Key Vault にアクセスできるようにするには、値を Oracle Key Vault 構成および SSL 材料を含むディレクトリに設定する必要があります。このディレクトリの設定手順は、[セクション6.4.4.4「keyring_okv KMIP プラグインの使用」](#)を参照してください。

ディレクトリには制限モードがあり、MySQL サーバーの実行に使用されるアカウントからのみアクセスできる必要があります。たとえば、Unix および Unix に似たシステムで `/usr/local/mysql/mysql-keyring-okv` ディレクトリを使用するには、次のコマンド (`root` として実行) を実行してディレクトリを作成し、そのモードと所有権を設定します:

```
cd /usr/local/mysql
mkdir mysql-keyring-okv
chmod 750 mysql-keyring-okv
chown mysql mysql-keyring-okv
chgrp mysql mysql-keyring-okv
```

`keyring_okv_conf_dir` に割り当てられた値で、存在しないディレクトリが指定された場合、または Oracle Key Vault への接続を確立できる構成情報が含まれていない場合、`keyring_okv` はエラーメッセージをエラーログに書き込みます。`keyring_okv_conf_dir` への実行時割当てが試行された結果、エラーが発生した場合、変数値およびキーリング操作は変更されません。

- `keyring_operations`

システム変数	<code>keyring_operations</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean

デフォルト値

ON

キーリング操作が有効かどうか。この変数は、キーの移行操作中に使用されます。 [セクション 6.4.4.9「キーリングキーストア間のキーの移行」](#) を参照してください。この変数の変更に必要な権限は、`SYSTEM_VARIABLES_ADMIN` または非推奨の `SUPER` 権限に加えて、`ENCRYPTION_KEY_ADMIN` です。

6.4.5 MySQL Enterprise Audit

注記

MySQL Enterprise Audit は、商用製品である MySQL Enterprise Edition に含まれる拡張機能です。商用製品の詳細は、<https://www.mysql.com/products/> を参照してください。

MySQL Enterprise Edition には、`audit_log` というサーバープラグインを使用して実装された MySQL Enterprise Audit が含まれています。MySQL Enterprise Audit では、オープン MySQL 監査 API を使用して、特定の MySQL サーバーで実行される接続およびクエリアクティビティの標準、ポリシーベースの監視、ロギングおよびブロックを有効にします。MySQL Enterprise Audit は、Oracle 監査仕様を満たすように設計されており、内部および外部の規制ガイドラインによって管理されるアプリケーションに対して、すぐに使用できる監査およびコンプライアンスソリューションを提供します。

インストール時に監査プラグインを使用すると、MySQL サーバーはサーバーアクティビティの監査レコードを含むログファイルを生成できます。ログの内容には、クライアントが接続および切断した時間、接続中に実行したアクション（アクセスしたデータベースおよびテーブルなど）が含まれます。

監査プラグインをインストールすると ([セクション 6.4.5.2「MySQL Enterprise Audit のインストールまたはアンインストール」](#) を参照)、監査ログファイルが書き込まれます。デフォルトでは、そのファイルはサーバーのデータディレクトリ内の `audit.log` という名前です。ファイルの名前を変更するには、サーバーの起動時に `audit_log_file` システム変数を設定します。

デフォルトでは、監査ログファイルの内容は、圧縮や暗号化を行わずに新しい形式の XML 形式で書き込まれます。ファイル形式を選択するには、サーバーの起動時に `audit_log_format` システム変数を設定します。ファイルの形式および内容についての詳細は、[セクション 6.4.5.4「監査ログファイル形式」](#) を参照してください。

監査ログファイルの名前や形式の選択など、ロギングの実行方法の制御の詳細は、[セクション 6.4.5.5「監査ロギング特性の構成」](#) を参照してください。監査イベントのフィルタリングを実行するには、[セクション 6.4.5.7「監査ログのフィルタリング」](#) を参照してください。監査ログプラグインを構成する際に使用されるパラメータについては、[監査ログのオプションおよび変数](#) を参照してください。

監査ログプラグインが有効になっている場合、パフォーマンススキーマ ([第 27 章「MySQL パフォーマンススキーマ」](#) を参照) にはそのインストールメンテーションがあります。関連するインストールメントを識別するには、次のクエリを使用します。

```
SELECT NAME FROM performance_schema.setup_instruments
WHERE NAME LIKE '%/alog/%';
```

6.4.5.1 MySQL Enterprise Audit の要素

MySQL Enterprise Audit は、監査ログプラグインおよび関連する要素に基づいています：

- `audit_log` という名前のサーバー側プラグインは、監査可能なイベントを調べ、それらを監査ログに書き込むかどうかを決定します。
- ユーザー定義関数を使用すると、ロギング動作、暗号化パスワードおよびログファイルの読取りを制御するフィルタリング定義を操作できます。
- `mysql` システムデータベースのテーブルは、フィルタおよびユーザーアカウントデータの永続的な記憶域を提供します。
- システム変数を使用すると、監査ログの構成が可能になり、ステータス変数を使用すると実行時の操作情報が提供されます。

- `AUDIT_ADMIN` 権限を使用すると、ユーザーは監査ログを管理できます。

6.4.5.2 MySQL Enterprise Audit のインストールまたはアンインストール

このセクションでは、[セクション6.4.5.1「MySQL Enterprise Audit の要素」](#)で説明されている監査ログプラグインおよび関連要素を使用して実装される MySQL Enterprise Audit をインストールまたはアンインストールする方法について説明します。プラグインのインストールについての一般的な情報は、[セクション5.6.1「プラグインのインストールおよびアンインストール」](#)を参照してください。

重要

指示に従う前に、このセクション全体をお読みください。手順の一部は、環境によって異なります。

注記

インストールされている場合、`audit_log` プラグインは、無効になっていても最小限のオーバーヘッドを伴います。このオーバーヘッドを回避するには、使用する予定がないかぎり、MySQL Enterprise Audit をインストールしないでください。

サーバーで使用できるようにするには、プラグインライブラリファイルを MySQL プラグインディレクトリ (`plugin_dir` システム変数で指定されたディレクトリ) に配置する必要があります。必要に応じて、サーバーの起動時に `plugin_dir` の値を設定してプラグインディレクトリの場所を構成します。

MySQL Enterprise Audit をインストールするには、MySQL インストールの `share` ディレクトリを検索し、ご使用のプラットフォームに適したスクリプトを選択します。使用可能なスクリプトは、プラグインライブラリファイルの参照に使用される接尾辞とは異なります:

- `audit_log_filter_win_install.sql`: ファイル名の接尾辞として `.dll` を使用する Windows システムの場合は、このスクリプトを選択します。
- `audit_log_filter_linux_install.sql`: Linux および `.so` をファイル名接尾辞として使用する類似システムの場合は、このスクリプトを選択します。

次のようにスクリプトを実行します。この例では、Linux インストールスクリプトを使用します。システムに適切な置換を行います。

```
shell> mysql -u root -p < audit_log_filter_linux_install.sql
Enter password: (enter root password here)
```

注記

一部の MySQL バージョンでは、MySQL Enterprise Audit テーブルの構造が変更されています。以前のバージョンの MySQL 8.0 からのアップグレードでテーブルが最新であることを確認するには、MySQL のアップグレード手順を実行し、更新を強制するオプションを使用してください ([セクション2.11「MySQL のアップグレード」](#)を参照)。MySQL Enterprise Audit テーブルに対してのみ UPDATE ステートメントを実行する場合は、次の説明を参照してください。

MySQL 8.0.12 以降、新規 MySQL インストールでは、MySQL Enterprise Audit により使用される `audit_log_user` テーブルの `USER` および `HOST` カラムは、`mysql.user` システムテーブルの `User` および `Host` カラムの定義によりよく対応する定義を持ちます。MySQL Enterprise Audit がすでにインストールされているインストールにアップグレードする場合は、次のようにテーブル定義を変更することをお勧めします:

```
ALTER TABLE mysql.audit_log_user
  DROP FOREIGN KEY audit_log_user_ibfk_1;
ALTER TABLE mysql.audit_log_filter
  CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_as_ci;
ALTER TABLE mysql.audit_log_user
  CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_as_ci;
ALTER TABLE mysql.audit_log_user
  MODIFY COLUMN USER VARCHAR(32);
```

```
ALTER TABLE mysql.audit_log_user
ADD FOREIGN KEY (FILTERNAME) REFERENCES mysql.audit_log_filter(NAME);
```

注記

ソース/レプリカレプリケーション、グループレプリケーションまたは InnoDB クラスターのコンテキストで MySQL Enterprise Audit を使用するには、ソースノードでインストールスクリプトを実行する前にレプリカノードを準備する必要があります。これは、スクリプト内の **INSTALL PLUGIN** ステートメントがレプリケートされないために必要です。

1. 各レプリカノードで、インストールスクリプトから **INSTALL PLUGIN** ステートメントを抽出し、手動で実行します。
2. ソースノードで、前述のようにインストールスクリプトを実行します。

プラグインのインストールを確認するには、**INFORMATION_SCHEMA.PLUGINS** テーブルを調べるか、**SHOW PLUGINS** ステートメントを使用します (セクション 5.6.2 「サーバープラグイン情報の取得」を参照)。例:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
FROM INFORMATION_SCHEMA.PLUGINS
WHERE PLUGIN_NAME LIKE 'audit%';
+-----+-----+
| PLUGIN_NAME | PLUGIN_STATUS |
+-----+-----+
| audit_log   | ACTIVE       |
+-----+-----+
```

プラグインの初期化に失敗した場合は、サーバーエラーログで診断メッセージを確認してください。

MySQL Enterprise Audit のインストール後、**--audit-log** オプションを使用して、後続のサーバー起動で **audit_log** プラグインのアクティブ化を制御できます。たとえば、プラグインが実行時に削除されないようにするには、このオプションを使用します:

```
[mysql]
audit-log=FORCE_PLUS_PERMANENT
```

監査プラグインを使用せずにサーバーが実行されることを回避する必要がある場合は、**FORCE** または **FORCE_PLUS_PERMANENT** の値とともに **--audit-log** を使用して、プラグインが正常に初期化されない場合にサーバーの起動を強制的に失敗させます。

重要

デフォルトでは、ルールベースの監査ログのフィルタリングでは、ユーザーの監査可能なイベントは記録されません。これは、すべてのユーザーのすべての監査可能イベントを記録するレガシー監査ログの動作とは異なります (セクション 6.4.5.9 「レガシーモード監査ログのフィルタリング」を参照)。ルールベースのフィルタリングを使用してログ詳細動作を生成する場合は、ロギングを有効にする単純なフィルタを作成し、それをデフォルトアカウントに割り当てます:

```
SELECT audit_log_filter_set_filter('log_all', '{ "filter": { "log": true } }');
SELECT audit_log_filter_set_user('%', 'log_all');
```

% に割り当てられたフィルタは、フィルタが明示的に割り当てられていないアカウントからの接続に使用されます (最初はすべてのアカウントに適用されます)。

前述のようにインストールすると、MySQL Enterprise Audit はアンインストールされるまでインストールされたままになります。削除するには、次のステートメントを実行します:

```
DROP TABLE IF EXISTS mysql.audit_log_user;
DROP TABLE IF EXISTS mysql.audit_log_filter;
UNINSTALL PLUGIN audit_log;
DROP FUNCTION audit_log_filter_set_filter;
DROP FUNCTION audit_log_filter_remove_filter;
DROP FUNCTION audit_log_filter_set_user;
DROP FUNCTION audit_log_filter_remove_user;
DROP FUNCTION audit_log_filter_flush;
```

```
DROP FUNCTION audit_log_encryption_password_get;  
DROP FUNCTION audit_log_encryption_password_set;  
DROP FUNCTION audit_log_read;  
DROP FUNCTION audit_log_read_bookmark;
```

6.4.5.3 MySQL Enterprise Audit のセキュリティに関する考慮事項

デフォルトでは、監査ログプラグインによって生成される監査ログファイルの内容は暗号化されず、SQL ステートメントのテキストなどの機密情報が含まれることがあります。セキュリティ上の理由から、監査ログファイルは、MySQL サーバーおよびログを表示する正当な理由を持つユーザーのみがアクセスできるディレクトリに書き込む必要があります。デフォルトのファイル名は、データディレクトリ内の `audit.log` です。これは、サーバーの起動時に `audit_log_file` システム変数を設定することで変更できます。ログローテーションが原因で、他の監査ログファイルが存在する可能性があります。

セキュリティを強化するには、監査ログファイルの暗号化を有効にします。[監査ログファイルの暗号化](#)を参照してください。

6.4.5.4 監査ログファイル形式

MySQL サーバーは、監査可能なイベントが発生するたびに、監査ログプラグインを呼び出して監査レコードをそのログファイルに書き込みます。通常、プラグインの起動後に書き込まれる最初の監査レコードには、サーバーの説明と起動オプションが含まれます。そのあとの要素は、クライアントの接続および切断イベント、SQL ステートメントの実行などのイベントを表します。最上位のステートメントのみのログが記録され、トリガーやストアードプロシージャなどのストアードプログラム内のステートメントのログは記録されません。`LOAD DATA` などのステートメントで参照されるファイルの内容は記録されません。

監査ログプラグインがログファイルの書き込みに使用するログ形式を選択するには、サーバーの起動時に `audit_log_format` システム変数を設定します。次の形式を使用できます:

- 新しいスタイルの XML 形式 (`audit_log_format=NEW`): 古い形式の XML 形式よりも Oracle Audit Vault との互換性が高い XML 形式。MySQL 8.0 では、デフォルトで新しいスタイルの XML 形式が使用されます。
- 「古いスタイルの XML」形式 (`audit_log_format=OLD`): 古い MySQL シリーズでデフォルトで使用される元の監査ログ形式。
- JSON 形式 (`audit_log_format=JSON`)

デフォルトでは、監査ログファイルの内容は、圧縮や暗号化を行わずに新しい形式の XML 形式で書き込まれます。

注記

ログ形式を変更する際に考慮する問題の詳細は、[監査ログファイル形式の選択](#)を参照してください。

次の各セクションでは、使用可能な監査ロギング形式について説明します:

- [新規スタイルの XML 監査ログファイル形式](#)
- [古い形式の XML 監査ログファイル形式](#)
- [JSON 監査ログファイル形式](#)

新規スタイルの XML 監査ログファイル形式

次に、読みやすくするために若干再フォーマットされた新しい形式の XML 形式 (`audit_log_format=NEW`) のサンプルログファイルを示します:

```
<?xml version="1.0" encoding="utf-8"?>  
<AUDIT>  
<AUDIT_RECORD>  
<TIMESTAMP>2019-10-03T14:06:33 UTC</TIMESTAMP>  
<RECORD_ID>1_2019-10-03T14:06:33</RECORD_ID>  
<NAME>Audit</NAME>
```



```
<SERVER_ID>1</SERVER_ID>
<VERSION>1</VERSION>
<STARTUP_OPTIONS>/usr/local/mysql/bin/mysqld
  --socket=/usr/local/mysql/mysql.sock
  --port=3306</STARTUP_OPTIONS>
<OS_VERSION>i686-Linux</OS_VERSION>
<MYSQL_VERSION>5.7.21-log</MYSQL_VERSION>
</AUDIT_RECORD>
<AUDIT_RECORD>
<TIMESTAMP>2019-10-03T14:09:38 UTC</TIMESTAMP>
<RECORD_ID>2_2019-10-03T14:06:33</RECORD_ID>
<NAME>Connect</NAME>
<CONNECTION_ID>5</CONNECTION_ID>
<STATUS>0</STATUS>
<STATUS_CODE>0</STATUS_CODE>
<USER>root</USER>
<OS_LOGIN/>
<HOST>localhost</HOST>
<IP>127.0.0.1</IP>
<COMMAND_CLASS>connect</COMMAND_CLASS>
<CONNECTION_TYPE>SSL/TLS</CONNECTION_TYPE>
<CONNECTION_ATTRIBUTES>
<ATTRIBUTE>
<NAME>_pid</NAME>
<VALUE>42794</VALUE>
</ATTRIBUTE>
...
<ATTRIBUTE>
<NAME>program_name</NAME>
<VALUE>mysqladmin</VALUE>
</ATTRIBUTE>
</CONNECTION_ATTRIBUTES>
<PRIV_USER>root</PRIV_USER>
<PROXY_USER/>
<DB>test</DB>
</AUDIT_RECORD>
...
<AUDIT_RECORD>
<TIMESTAMP>2019-10-03T14:09:38 UTC</TIMESTAMP>
<RECORD_ID>6_2019-10-03T14:06:33</RECORD_ID>
<NAME>Query</NAME>
<CONNECTION_ID>5</CONNECTION_ID>
<STATUS>0</STATUS>
<STATUS_CODE>0</STATUS_CODE>
<USER>root[root] @ localhost [127.0.0.1]</USER>
<OS_LOGIN/>
<HOST>localhost</HOST>
<IP>127.0.0.1</IP>
<COMMAND_CLASS>drop_table</COMMAND_CLASS>
<SQLTEXT>DROP TABLE IF EXISTS t</SQLTEXT>
</AUDIT_RECORD>
...
<AUDIT_RECORD>
<TIMESTAMP>2019-10-03T14:09:39 UTC</TIMESTAMP>
<RECORD_ID>8_2019-10-03T14:06:33</RECORD_ID>
<NAME>Quit</NAME>
<CONNECTION_ID>5</CONNECTION_ID>
<STATUS>0</STATUS>
<STATUS_CODE>0</STATUS_CODE>
<USER>root</USER>
<OS_LOGIN/>
<HOST>localhost</HOST>
<IP>127.0.0.1</IP>
<COMMAND_CLASS>connect</COMMAND_CLASS>
<CONNECTION_TYPE>SSL/TLS</CONNECTION_TYPE>
</AUDIT_RECORD>
...
<AUDIT_RECORD>
<TIMESTAMP>2019-10-03T14:09:43 UTC</TIMESTAMP>
```

```

<RECORD_ID>11_2019-10-03T14:06:33</RECORD_ID>
<NAME>Quit</NAME>
<CONNECTION_ID>6</CONNECTION_ID>
<STATUS>0</STATUS>
<STATUS_CODE>0</STATUS_CODE>
<USER>root</USER>
<OS_LOGIN/>
<HOST>localhost</HOST>
<IP>127.0.0.1</IP>
<COMMAND_CLASS>connect</COMMAND_CLASS>
<CONNECTION_TYPE>SSL/TLS</CONNECTION_TYPE>
</AUDIT_RECORD>
<AUDIT_RECORD>
<TIMESTAMP>2019-10-03T14:09:45 UTC</TIMESTAMP>
<RECORD_ID>12_2019-10-03T14:06:33</RECORD_ID>
<NAME>NoAudit</NAME>
<SERVER_ID>1</SERVER_ID>
</AUDIT_RECORD>
</AUDIT>

```

監査ログファイルは、UTF-8 (1文字あたり最大4バイト) を使用した XML として記述されています。ルート要素は、`<AUDIT>` です。ルート要素には、`<AUDIT_RECORD>` 要素が含まれています。この要素のそれぞれは、監査対象のイベントに関する情報を提供します。監査ログプラグインは、新しいログファイルの書き込みを開始すると、XML 宣言と開始 `<AUDIT>` ルート要素タグを書き込みます。プラグインは、ログファイルを閉じると、終了 `</AUDIT>` ルート要素タグを書き込みます。ファイルが開いている間は、終了タグは存在しません。

`<AUDIT_RECORD>` 要素内の要素には、次の特性があります:

- 一部の要素はすべての `<AUDIT_RECORD>` 要素に表示されます。その他はオプションで、監査レコードタイプに応じて表示される場合があります。
- `<AUDIT_RECORD>` 要素内の要素の順序は保証されません。
- 要素値が固定長ではありません。長い値は、後で説明する要素の説明に従って切り捨てられる場合があります。
- `<`、`>`、`"`、および `&` 文字は、それぞれ `<`、`>`、`"`、および `&` としてエンコードされます。NUL バイト (U+00) は、`?` 文字としてエンコードされます。
- XML 文字として有効でない文字は、数値の文字参照を使用してエンコードされます。有効な XML 文字は次のとおりです。

```
#x9 | #xA | #xD | [#x20-#xD7FF] | [#xE000-#xFFFD] | [#x10000-#x10FFFF]
```

次の要素は、すべての `<AUDIT_RECORD>` 要素で必須です。

- `<NAME>`

監査イベントを生成した命令 (サーバーがクライアントから受信したコマンドなど) のタイプを表す文字列。

例:

```
<NAME>Query</NAME>
```

一部の一般的な `<NAME>` 値:

```

Audit   When auditing starts, which may be server startup time
Connect When a client connects, also known as logging in
Query   An SQL statement (executed directly)
Prepare Preparation of an SQL statement; usually followed by Execute
Execute Execution of an SQL statement; usually follows Prepare
Shutdown Server shutdown
Quit    When a client disconnects
NoAudit Auditing has been turned off

```

使用可能な値は `Audit`、`Binlog Dump`、`Change user`、`Close stmt`、`Connect Out`、`Connect`、`Create DB`、`Daemon`、`Debug`、`Delayed insert`、`Drop DB`、`Execute`、`Fetch`、`Field List`、`Init DB`、`Kill`、`Long Data`、`NoAudit`、`Ping`、`Prepare`、`Processlist`、`Query`、`Quit`、`Refresh`、`Register Slave`、`Reset stmt`、`Set option`、`Shutdown`、`Sleep`、`Statistics`、`Table Dump`、`TableDelete`、`TableInsert`、`TableRead`、`TableUpdate`、`Time` です。

これらの値の多くは、`my_command.h` ヘッダーファイルにリストされている `COM_xxx` コマンド値に対応しています。たとえば、`Create DB` および `Change user` は、それぞれ `COM_CREATE_DB` および `COM_CHANGE_USER` に対応します。

`TableXXX` の `<NAME>` 値を持つイベントには、`Query` イベントが付随します。たとえば、次のステートメントは、1つの `Query` イベント、2つの `TableRead` イベントおよび `TableInsert` イベントを生成します:

```
INSERT INTO t3 SELECT t1.* FROM t1 JOIN t2;
```

各 `TableXXX` イベントには、イベントが参照するテーブルを識別する `<DB>` および `<TABLE>` 要素が含まれます。

- `<RECORD_ID>`

監査レコードを表す一意の識別子。この値はシーケンス番号とタイムスタンプで構成され、形式は `SEQ_TIMESTAMP` です。監査ログプラグインが監査ログファイルを開くと、順序番号が監査ログファイルのサイズに初期化され、記録されるレコードごとに順序が1ずつ増分されます。タイムスタンプは、監査ログプラグインがファイルを開いた日時を示す `YYYY-MM-DDThh:mm:ss` 形式の UTC 値です。

例:

```
<RECORD_ID>12_2019-10-03T14:06:33</RECORD_ID>
```

- `<TIMESTAMP>`

監査イベントが生成された日時を示す `YYYY-MM-DDThh:mm:ss UTC` 形式の UTC 値を表す文字列。たとえば、クライアントから受信した SQL ステートメントの実行に対応するイベントの `<TIMESTAMP>` 値は、受信時ではなく、ステートメントの終了後に発生します。

例:

```
<TIMESTAMP>2019-10-03T14:09:45 UTC</TIMESTAMP>
```

次の要素は、`<AUDIT_RECORD>` 要素ではオプションです。これらの多くは、特定の `<NAME>` 要素値でのみ発生します。

- `<COMMAND_CLASS>`

実行されたアクションのタイプを示す文字列。

例:

```
<COMMAND_CLASS>drop_table</COMMAND_CLASS>
```

値は `statement/sql/xxx` コマンドカウンタに対応しています。たとえば、`xxx` は、`DROP TABLE` および `SELECT` ステートメントのそれぞれ `drop_table` および `select` です。次のステートメントは、使用可能な名前を表示します:

```
SELECT REPLACE(EVENT_NAME, 'statement/sql/', '') AS name
FROM performance_schema.events_statements_summary_global_by_event_name
WHERE EVENT_NAME LIKE 'statement/sql/%'
ORDER BY name;
```

- `<CONNECTION_ATTRIBUTES>`

MySQL 8.0.19 では、`<COMMAND_CLASS>` 値が `connect` のイベントに `<CONNECTION_ATTRIBUTES>` 要素が含まれ、接続時にクライアントによって渡される接続属性が表示される場合があります。(「パフォーマンススキーマ」テーブルでも公開されるこれらの属性の詳細は、[セクション27.12.9「パフォーマンススキーマ接続属性テーブル」](#)を参照してください。)

`<CONNECTION_ATTRIBUTES>` 要素には属性ごとに1つの `<ATTRIBUTE>` 要素が含まれ、それぞれに属性名と値を示す `<NAME>` 要素と `<VALUE>` 要素が含まれます。

例:

```
<CONNECTION_ATTRIBUTES>
<ATTRIBUTE>
```

```
<NAME>_pid</NAME>
<VALUE>42794</VALUE>
</ATTRIBUTE>
<ATTRIBUTE>
<NAME>_os</NAME>
<VALUE>macos10.14</VALUE>
</ATTRIBUTE>
<ATTRIBUTE>
<NAME>_platform</NAME>
<VALUE>x86_64</VALUE>
</ATTRIBUTE>
<ATTRIBUTE>
<NAME>_client_version</NAME>
<VALUE>8.0.19</VALUE>
</ATTRIBUTE>
<ATTRIBUTE>
<NAME>_client_name</NAME>
<VALUE>libmysql</VALUE>
</ATTRIBUTE>
<ATTRIBUTE>
<NAME>program_name</NAME>
<VALUE>mysqladmin</VALUE>
</ATTRIBUTE>
</CONNECTION_ATTRIBUTES>
```

イベントに接続属性が存在しない場合、何も記録されず、`<CONNECTION_ATTRIBUTES>` 要素は表示されません。これは、接続試行が成功しなかった場合、クライアントが属性を渡さなかった場合、またはサーバーの起動時やプラグインによって開始されたときなどに接続が内部的に発生した場合に発生することがあります。

- `<CONNECTION_ID>`

クライアント接続識別子を表す符号なし整数。これは、セッション内の `CONNECTION_ID()` 関数によって戻される値と同じです。

例:

```
<CONNECTION_ID>127</CONNECTION_ID>
```

- `<CONNECTION_TYPE>`

サーバーへの接続のセキュリティ状態。許可される値は、`TCP/IP` (暗号化なしで確立された TCP/IP 接続)、`SSL/TLS` (暗号化で確立された TCP/IP 接続)、`Socket` (Unix ソケットファイル接続)、`Named Pipe` (Windows 名前付きパイプ接続) および `Shared Memory` (Windows 共有メモリ接続) です。

例:

```
<CONNECTION_TYPE>SSL/TLS</CONNECTION_TYPE>
```

- `<DB>`

デフォルトのデータベース名を表す文字列。

例:

```
<DB>test</DB>
```

- `<HOST>`

クライアントのホスト名を表す文字列。

例:

```
<HOST>localhost</HOST>
```

- `<IP>`

クライアントの IP アドレスを表す文字列。

例:

```
<IP>127.0.0.1</IP>
```

- **<MYSQL_VERSION>**

MySQL サーバーのバージョンを表す文字列。これは、セッション内の `VERSION()` 関数または `version` システム変数の値と同じです。

例:

```
<MYSQL_VERSION>5.7.21-log</MYSQL_VERSION>
```

- **<OS_LOGIN>**

認証プロセス中に使用される外部ユーザー名を表す文字列。クライアントの認証に使用されるプラグインによって設定されます。ネイティブ (組み込み) MySQL 認証の場合、またはプラグインで値が設定されていない場合、この要素は空です。この値は、`external_user` システム変数の値と同じです ([セクション6.2.18「プロキシユーザー」](#)を参照)。

例:

```
<OS_LOGIN>jeffrey</OS_LOGIN>
```

- **<OS_VERSION>**

サーバーが構築された、または実行されているオペレーティングシステムを表す文字列。

例:

```
<OS_VERSION>x86_64-Linux</OS_VERSION>
```

- **<PRIV_USER>**

サーバーがクライアントを認証する際に使用したユーザーを表す文字列。これは、サーバーが権限チェックを行う際に使用するユーザー名であり、**<USER>** の値とは異なる可能性があります。

例:

```
<PRIV_USER>jeffrey</PRIV_USER>
```

- **<PROXY_USER>**

プロキシユーザーを表す文字列 ([セクション6.2.18「プロキシユーザー」](#)を参照)。ユーザーのプロキシ処理が有効になっていない場合は、値が空です。

例:

```
<PROXY_USER>developer</PROXY_USER>
```

- **<SERVER_ID>**

サーバー ID を表す符号なし整数。これは、`server_id` システム変数の値と同じです。

例:

```
<SERVER_ID>1</SERVER_ID>
```

- **<SQLTEXT>**

SQL ステートメントのテキストを表す文字列。この値は、空にすることができます。長い値は、切り捨てられる可能性があります。監査ログファイル自体などの文字列は、UTF-8 (1文字あたり最大4バイト) を使用して記述されるため、この値が変換の結果となる場合があります。たとえば、元のステートメントは、SJIS 文字列としてクライアントから受信された可能性があります。

例:

```
<SQLTEXT>DELETE FROM t1</SQLTEXT>
```

- <STARTUP_OPTIONS>

MySQL サーバーの起動時に、コマンド行またはオプションファイルで指定されたオプションを表す文字列。最初のオプションは、サーバー実行可能ファイルへのパスです。

例:

```
<STARTUP_OPTIONS>/usr/local/mysql/bin/mysqld
--port=3306 --log_output=FILE</STARTUP_OPTIONS>
```

- <STATUS>

コマンドのステータスを表す符号なし整数 (成功した場合は 0、エラーが発生した場合はゼロ以外)。これは、`mysql_errno()` C API 関数の値と同じです。<STATUS> と異なる点については、<STATUS_CODE> の説明を参照してください。

監査ログには、SQLSTATE 値またはエラーメッセージが含まれていません。エラーコード、SQLSTATE 値、およびメッセージ間の関連性を確認する方法については、[Server Error Message Reference](#)を参照してください。

警告のログは記録されません。

例:

```
<STATUS>1051</STATUS>
```

- <STATUS_CODE>

コマンドのステータスを表す符号なし整数 (成功した場合は 0、エラーが発生した場合は 1)。

`STATUS_CODE` の値は、`STATUS` の値とは異なります。`STATUS_CODE` は、成功した場合は 0、エラーが発生した場合は 1 であり、Audit Vault の `EZ_collector` コンシューマとの互換性があります。`STATUS` は、`mysql_errno()` C API 関数の値です。これは、成功した場合は 0、エラーが発生した場合はゼロ以外です。そのため、エラーが発生した場合、必ずしも 1 であるとはかぎりません。

例:

```
<STATUS_CODE>0</STATUS_CODE>
```

- <TABLE>

テーブル名を表す文字列。

例:

```
<TABLE>t3</TABLE>
```

- <USER>

クライアントによって送信されたユーザー名を表す文字列。これは、<PRIV_USER> の値とは異なる可能性があります。

例:

```
<USER>root[root] @ localhost [127.0.0.1]</USER>
```

- <VERSION>

監査ログファイル形式のバージョンを表す符号なし整数。

例:

```
<VERSION>1</VERSION>
```


古い形式の XML 監査ログファイル形式

次に、読みやすくするために若干再フォーマットされた古い形式の XML 形式 (`audit_log_format=OLD`) のサンプルログファイルを示します:

```
<?xml version="1.0" encoding="utf-8"?>
<AUDIT>
  <AUDIT_RECORD
    TIMESTAMP="2019-10-03T14:25:00 UTC"
    RECORD_ID="1_2019-10-03T14:25:00"
    NAME="Audit"
    SERVER_ID="1"
    VERSION="1"
    STARTUP_OPTIONS="--port=3306"
    OS_VERSION="i686-Linux"
    MYSQL_VERSION="5.7.21-log"/>
  <AUDIT_RECORD
    TIMESTAMP="2019-10-03T14:25:24 UTC"
    RECORD_ID="2_2019-10-03T14:25:00"
    NAME="Connect"
    CONNECTION_ID="4"
    STATUS="0"
    STATUS_CODE="0"
    USER="root"
    OS_LOGIN=""
    HOST="localhost"
    IP="127.0.0.1"
    COMMAND_CLASS="connect"
    CONNECTION_TYPE="SSL/TLS"
    PRIV_USER="root"
    PROXY_USER=""
    DB="test"/>
  ...
  <AUDIT_RECORD
    TIMESTAMP="2019-10-03T14:25:24 UTC"
    RECORD_ID="6_2019-10-03T14:25:00"
    NAME="Query"
    CONNECTION_ID="4"
    STATUS="0"
    STATUS_CODE="0"
    USER="root[root] @ localhost [127.0.0.1]"
    OS_LOGIN=""
    HOST="localhost"
    IP="127.0.0.1"
    COMMAND_CLASS="drop_table"
    SQLTEXT="DROP TABLE IF EXISTS t"/>
  ...
  <AUDIT_RECORD
    TIMESTAMP="2019-10-03T14:25:24 UTC"
    RECORD_ID="8_2019-10-03T14:25:00"
    NAME="Quit"
    CONNECTION_ID="4"
    STATUS="0"
    STATUS_CODE="0"
    USER="root"
    OS_LOGIN=""
    HOST="localhost"
    IP="127.0.0.1"
    COMMAND_CLASS="connect"
    CONNECTION_TYPE="SSL/TLS"/>
  <AUDIT_RECORD
    TIMESTAMP="2019-10-03T14:25:32 UTC"
    RECORD_ID="12_2019-10-03T14:25:00"
    NAME="NoAudit"
    SERVER_ID="1"/>
</AUDIT>
```

監査ログファイルは、UTF-8 (1 文字当たり最大 4 バイト) を使用した XML として記述されています。ルート要素は、`<AUDIT>` です。ルート要素には、`<AUDIT_RECORD>` 要素が含まれています。この要素のそれぞれは、監査

対象のイベントに関する情報を提供します。監査ログプラグインは、新しいログファイルの書き込みを開始すると、XML 宣言と開始 `<AUDIT>` ルート要素タグを書き込みます。プラグインは、ログファイルを閉じると、終了 `</AUDIT>` ルート要素タグを書き込みます。ファイルが開いている間は、終了タグは存在しません。

`<AUDIT_RECORD>` 要素の属性には、次のような特性があります。

- 一部の属性はすべての `<AUDIT_RECORD>` 要素に表示されます。その他はオプションで、監査レコードタイプに応じて表示される場合があります。
- `<AUDIT_RECORD>` 要素内の属性の順序は保証されません。
- 属性値は固定長ではありません。あとで属性の説明で示すように、長い値は切り捨てられる可能性があります。
- `<`、`>`、`"`、および `&` 文字は、それぞれ `<`、`>`、`"`、および `&` としてエンコードされます。NUL バイト (U+00) は、`?` 文字としてエンコードされます。
- XML 文字として有効でない文字は、数値の文字参照を使用してエンコードされます。有効な XML 文字は次のとおりです。

```
#xA | #xD | [#x20-#xD7FF] | [#xE000-#xFFFF] | [#x10000-#x10FFFF]
```

次の属性は、すべての `<AUDIT_RECORD>` 要素で必須です。

- NAME**

監査イベントを生成した命令 (サーバーがクライアントから受信したコマンドなど) のタイプを表す文字列。

例: `NAME="Query"`

一部の一般的な **NAME** 値:

```
Audit   When auditing starts, which may be server startup time
Connect When a client connects, also known as logging in
Query   An SQL statement (executed directly)
Prepare Preparation of an SQL statement; usually followed by Execute
Execute Execution of an SQL statement; usually follows Prepare
Shutdown Server shutdown
Quit    When a client disconnects
NoAudit Auditing has been turned off
```

使用可能な値は `Audit`、`Binlog Dump`、`Change user`、`Close stmt`、`Connect Out`、`Connect`、`Create DB`、`Daemon`、`Debug`、`Delayed insert`、`Drop DB`、`Execute`、`Fetch`、`Field List`、`Init DB`、`Kill`、`Long Data`、`NoAudit`、`Ping`、`Prepare`、`Processlist`、`Query`、`Quit`、`Refresh`、`Register Slave`、`Reset stmt`、`Set option`、`Shutdown`、`Sleep`、`Statistics`、`Table Dump`、`TableDelete`、`TableInsert`、`TableRead`、`TableUpdate`、`Time` です。

これらの値の多くは、`my_command.h` ヘッダーファイルにリストされている `COM_xxx` コマンド値に対応しています。たとえば、`"Create DB"` および `"Change user"` は、それぞれ `COM_CREATE_DB` および `COM_CHANGE_USER` に対応します。

`TableXXX` の **NAME** 値を持つイベントには、`Query` イベントが付随します。たとえば、次のステートメントは、1 つの `Query` イベント、2 つの `TableRead` イベントおよび `TableInsert` イベントを生成します:

```
INSERT INTO t3 SELECT t1.* FROM t1 JOIN t2;
```

各 `TableXXX` イベントには、イベントが参照するテーブルを識別するための **DB** および **TABLE** 属性があります。

古い形式の XML 監査ログ形式の `Connect` イベントには、接続属性は含まれません。

- RECORD_ID**

監査レコードを表す一意の識別子。この値はシーケンス番号とタイムスタンプで構成され、形式は `SEQ_TIMESTAMP` です。監査ログプラグインが監査ログファイルを開くと、順序番号が監査ログファイルのサイズに初期化され、記録されるレコードごとに順序が 1 ずつ増分されます。タイムスタンプは、監査ログプラグインがファイルを開いた日時を示す `YYYY-MM-DDThh:mm:ss` 形式の UTC 値です。

例: `RECORD_ID="12_2019-10-03T14:25:00"`

- **TIMESTAMP**

監査イベントが生成された日時を示す `YYYY-MM-DDThh:mm:ss UTC` 形式の UTC 値を表す文字列。たとえば、クライアントから受信した SQL ステートメントの実行に対応するイベントの **TIMESTAMP** 値は、受信時ではなく、ステートメントの終了後に発生します。

例: `TIMESTAMP="2019-10-03T14:25:32 UTC"`

次の属性は、`<AUDIT_RECORD>` 要素ではオプションです。これらの多くは、**NAME** 属性の特定の値を含む要素でのみ発生します。

- **COMMAND_CLASS**

実行されたアクションのタイプを示す文字列。

例: `COMMAND_CLASS="drop_table"`

値は `statement/sql/xxx` コマンドカウンタに対応しています。たとえば、`xxx` は、**DROP TABLE** および **SELECT** ステートメントのそれぞれ `drop_table` および `select` です。次のステートメントは、使用可能な名前を表示します:

```
SELECT REPLACE(EVENT_NAME, 'statement/sql/', '') AS name
FROM performance_schema.events_statements_summary_global_by_event_name
WHERE EVENT_NAME LIKE 'statement/sql/%'
ORDER BY name;
```

- **CONNECTION_ID**

クライアント接続識別子を表す符号なし整数。これは、セッション内の `CONNECTION_ID()` 関数によって戻される値と同じです。

例: `CONNECTION_ID="127"`

- **CONNECTION_TYPE**

サーバーへの接続のセキュリティ状態。許可される値は、**TCP/IP** (暗号化なしで確立された TCP/IP 接続)、**SSL/TLS** (暗号化で確立された TCP/IP 接続)、**Socket** (Unix ソケットファイル接続)、**Named Pipe** (Windows 名前付きパイプ接続) および **Shared Memory** (Windows 共有メモリー接続) です。

例: `CONNECTION_TYPE="SSL/TLS"`

- **DB**

デフォルトのデータベース名を表す文字列。

例: `DB="test"`

- **HOST**

クライアントのホスト名を表す文字列。

例: `HOST="localhost"`

- **IP**

クライアントの IP アドレスを表す文字列。

例: `IP="127.0.0.1"`

- **MYSQL_VERSION**

MySQL サーバーのバージョンを表す文字列。これは、セッション内の `VERSION()` 関数または `version` システム変数の値と同じです。

例: `MYSQL_VERSION="5.7.21-log"`

- **OS_LOGIN**

認証プロセス中に使用される外部ユーザー名を表す文字列。クライアントの認証に使用されるプラグインによって設定されます。ネイティブ (組込み) MySQL 認証の場合、またはプラグインで値が設定されていない場合、この属性は空です。この値は、`external_user` システム変数の値と同じです ([セクション6.2.18「プロキシユーザー」](#)を参照)。

例: `OS_LOGIN="jeffrey"`

- `OS_VERSION`

サーバーが構築された、または実行されているオペレーティングシステムを表す文字列。

例: `OS_VERSION="x86_64-Linux"`

- `PRIV_USER`

サーバーがクライアントを認証する際に使用したユーザーを表す文字列。これは、サーバーが権限チェックに使用するユーザー名で、`USER` の値とは異なる場合があります。

例: `PRIV_USER="jeffrey"`

- `PROXY_USER`

プロキシユーザーを表す文字列 ([セクション6.2.18「プロキシユーザー」](#)を参照)。ユーザーのプロキシ処理が有効になっていない場合は、値が空です。

例: `PROXY_USER="developer"`

- `SERVER_ID`

サーバー ID を表す符号なし整数。これは、`server_id` システム変数の値と同じです。

例: `SERVER_ID="1"`

- `SQLTEXT`

SQL ステートメントのテキストを表す文字列。この値は、空にすることができます。長い値は、切り捨てられる可能性があります。監査ログファイル自体などの文字列は、UTF-8 (1 文字当たり最大 4 バイト) を使用して記述されるため、この値が変換の結果となる場合があります。たとえば、元のステートメントは、SJIS 文字列としてクライアントから受信された可能性があります。

例: `SQLTEXT="DELETE FROM t1"`

- `STARTUP_OPTIONS`

MySQL サーバーの起動時に、コマンド行またはオプションファイルで指定されたオプションを表す文字列。

例: `STARTUP_OPTIONS="--port=3306 --log_output=FILE"`

- `STATUS`

コマンドのステータスを表す符号なし整数 (成功した場合は 0、エラーが発生した場合はゼロ以外)。これは、`mysql_errno()` C API 関数の値と同じです。`STATUS` との違いの詳細は、`STATUS_CODE` の説明を参照してください。

監査ログには、SQLSTATE 値またはエラーメッセージが含まれていません。エラーコード、SQLSTATE 値、およびメッセージ間の関連性を確認する方法については、[Server Error Message Reference](#)を参照してください。

警告のログは記録されません。

例: `STATUS="1051"`

- `STATUS_CODE`

コマンドのステータスを表す符号なし整数 (成功した場合は 0、エラーが発生した場合は 1)。

`STATUS_CODE` の値は、`STATUS` の値とは異なります。`STATUS_CODE` は、成功した場合は 0、エラーが発生した場合は 1 であり、Audit Vault の EZ_collector コンシューマとの互換性があります。`STATUS` は、`mysql_errno()` C API 関数の値です。これは、成功した場合は 0、エラーが発生した場合はゼロ以外です。そのため、エラーが発生した場合、必ずしも 1 であるとはかぎりません。

例: `STATUS_CODE="0"`

- **TABLE**

テーブル名を表す文字列。

例: `TABLE="t3"`

- **USER**

クライアントによって送信されたユーザー名を表す文字列。これは `PRIV_USER` の値とは異なる場合があります。

- **VERSION**

監査ログファイル形式のバージョンを表す符号なし整数。

例: `VERSION="1"`

JSON 監査ログファイル形式

JSON 形式の監査ロギング (`audit_log_format=JSON`) の場合、ログファイルの内容は、監査対象イベントをキーと値のペアの **JSON** ハッシュとして表す各配列要素を持つ **JSON** 配列を形成します。完全なイベントレコードの例は、このセクションの後半で説明します。部分的なイベントの抜粋を次に示します:

```
[
  {
    "timestamp": "2019-10-03 13:50:01",
    "id": 0,
    "class": "audit",
    "event": "startup",
    ...
  },
  {
    "timestamp": "2019-10-03 15:02:32",
    "id": 0,
    "class": "connection",
    "event": "connect",
    ...
  },
  ...
  {
    "timestamp": "2019-10-03 17:37:26",
    "id": 0,
    "class": "table_access",
    "event": "insert",
    ...
  }
  ...
]
```

監査ログファイルは、UTF-8 を使用して書き込まれます (文字当たり最大 4 バイト)。監査ログプラグインは、新しいログファイルの書き込みを開始すると、開いている[アレイマーカ-を書き込みます。プラグインは、ログファイルを閉じるときに、閉じている]配列マーカ-を書き込みます。ファイルが開いている間は、閉じマーカ-は存在しません。

監査レコード内の項目には、次の特性があります:

- 一部の項目は、すべての監査レコードに表示されます。その他はオプションで、監査レコードタイプに応じて表示される場合があります。
- 監査レコード内の項目の順序は保証されません。

- アイテム値が固定長ではありません。長い値は、後で指定する品目摘要で示されるように切り捨てられる場合があります。
- "および"の文字は、それぞれ"および"としてエンコードされます。

次の例は、読みやすくするために若干再フォーマットされた様々なイベントタイプ (`class` および `event` 項目で示される) の JSON オブジェクト形式を示しています:

起動イベントの監査:

```
{ "timestamp": "2019-10-03 14:21:56",
  "id": 0,
  "class": "audit",
  "event": "startup",
  "connection_id": 0,
  "startup_data": { "server_id": 1,
    "os_version": "i686-Linux",
    "mysql_version": "5.7.21-log",
    "args": ["/usr/local/mysql/bin/mysqld",
      "--loose-audit-log-format=JSON",
      "--log-error=log.err",
      "--pid-file=mysqld.pid",
      "--port=3306" ]}}
```

(実行時に有効にされるのではなく) サーバーの起動の結果として監査ログプラグインが起動すると、`connection_id` は 0 に設定され、`account` および `login` は存在しません。

停止イベントの監査:

```
{ "timestamp": "2019-10-03 14:28:20",
  "id": 3,
  "class": "audit",
  "event": "shutdown",
  "connection_id": 0,
  "shutdown_data": { "server_id": 1 }}
```

(実行時に無効にされるのではなく) サーバーのシャットダウンの結果として監査ログプラグインがアンインストールされると、`connection_id` は 0 に設定され、`account` および `login` は存在しません。

Connect または change-user イベント:

```
{ "timestamp": "2019-10-03 14:23:18",
  "id": 1,
  "class": "connection",
  "event": "connect",
  "connection_id": 5,
  "account": { "user": "root", "host": "localhost" },
  "login": { "user": "root", "os": "", "ip": "::1", "proxy": "" },
  "connection_data": { "connection_type": "ssl",
    "status": 0,
    "db": "test",
    "connection_attributes": {
      "_pid": "43236",
      ...
      "program_name": "mysqladmin"
    }
  }
}
```

切断イベント:

```
{ "timestamp": "2019-10-03 14:24:45",
  "id": 3,
  "class": "connection",
  "event": "disconnect",
  "connection_id": 5,
  "account": { "user": "root", "host": "localhost" },
  "login": { "user": "root", "os": "", "ip": "::1", "proxy": "" },
  "connection_data": { "connection_type": "ssl" }}
```

クエリーイベント:

```
{ "timestamp": "2019-10-03 14:23:35",
  "id": 2,
```



```
"class": "general",
"event": "status",
"connection_id": 5,
"account": { "user": "root", "host": "localhost" },
"login": { "user": "root", "os": "", "ip": "::1", "proxy": "" },
"general_data": { "command": "Query",
                  "sql_command": "show_variables",
                  "query": "SHOW VARIABLES",
                  "status": 0 }
```

テーブルアクセスイベント (読取り、削除、挿入、更新):

```
{ "timestamp": "2019-10-03 14:23:41",
  "id": 0,
  "class": "table_access",
  "event": "insert",
  "connection_id": 5,
  "account": { "user": "root", "host": "localhost" },
  "login": { "user": "root", "os": "", "ip": "127.0.0.1", "proxy": "" },
  "table_access_data": { "db": "test",
                        "table": "t1",
                        "query": "INSERT INTO t1 (i) VALUES(1),(2),(3)",
                        "sql_command": "insert" }
```

次のリストの項目は、JSON 形式の監査レコードの最上位レベルに表示されます: 各アイテム値はスカラーまたは JSON ハッシュのいずれかです。ハッシュ値を持つアイテムの場合、説明にはそのハッシュ内のアイテム名のみがリストされます。セカンドレベルハッシュアイテムの詳細は、このセクションの後半のを参照してください。

- [account](#)

イベントに関連付けられた MySQL アカウント。この値は、セクション内の [CURRENT_USER\(\)](#) 関数の値と同等の項目を含むハッシュです: [user](#)、[host](#)。

例:

```
"account": { "user": "root", "host": "localhost" }
```

- [class](#)

イベントクラスを表す文字列。このクラスは、イベントサブクラスを指定する [event](#) アイテムとともに取得されるイベントのタイプを定義します。

例:

```
"class": "connection"
```

次のテーブルに、[class](#) 値と [event](#) 値の許可される組合せを示します。

表 6.28 監査ログクラスとイベントの組合せ

クラス値	許可されたイベント値
audit	startup , shutdown
接続	connect , change_user , disconnect
general	status
table_access_data	read , delete , insert , update

- [connection_data](#)

クライアント接続に関する情報。値は、これらのアイテムを含むハッシュです: [connection_type](#)、[status](#)、[db](#)、および場合によっては [connection_attributes](#)。この項目は、[class](#) 値が [connection](#) の監査レコードに対してのみ発生します。

例:

```
"connection_data": { "connection_type": "ssl",
                    "status": 0,
```

```
"db": "test" }
```

MySQL 8.0.19 の時点では、`class` 値が `connection` で `event` 値が `connect` のイベントには、接続時にクライアントによって渡された接続属性を表示するための `connection_attributes` アイテムが含まれる場合があります。(「パフォーマンススキーマ」テーブルでも公開されるこれらの属性の詳細は、[セクション27.12.9「パフォーマンススキーマ接続属性テーブル」](#)を参照してください。)

`connection_attributes` 値は、各属性をその名前と値で表すハッシュです。

例:

```
"connection_attributes": {
  "_pid": "43236",
  "_os": "macos10.14",
  "_platform": "x86_64",
  "_client_version": "8.0.19",
  "_client_name": "libmysql",
  "program_name": "mysqladmin"
}
```

イベントに接続属性が存在しない場合、何も記録されず、`connection_attributes` アイテムは表示されません。これは、接続試行が成功しなかった場合、クライアントが属性を渡さなかった場合、またはサーバーの起動時やプラグインによって開始されたときなどに接続が内部的に発生した場合に発生することがあります。

- `connection_id`

クライアント接続識別子を表す符号なし整数。これは、セッション内の `CONNECTION_ID()` 関数によって戻される値と同じです。

例:

```
"connection_id": 5
```

- `event`

イベントクラスのサブクラスを表す文字列。サブクラスは、イベントクラスを指定する `class` アイテムとともに取得されるイベントのタイプを定義します。詳細は、`class` アイテムの説明を参照してください。

例:

```
"event": "connect"
```

- `general_data`

実行されたステートメントまたはコマンドに関する情報。値は、これらのアイテムを含むハッシュです: `command`, `sql_command`, `query`, `status`。この項目は、`class` 値が `general` の監査レコードに対してのみ発生します。

例:

```
"general_data": { "command": "Query",
  "sql_command": "show_variables",
  "query": "SHOW VARIABLES",
  "status": 0 }
```

- `id`

イベント ID を表す符号なし整数。

例:

```
"id": 2
```

同じ `timestamp` 値を持つ監査レコードの場合、`id` 値によってそれらが区別され、順序が形成されます。監査ログ内では、`timestamp/id` のペアは一意です。これらのペアは、ログ内のイベントの場所を識別するブックマークです。

- [login](#)

クライアントがサーバーに接続した方法を示す情報。値は、これらのアイテムを含むハッシュです: [user](#), [os](#), [ip](#), [proxy](#)。

例:

```
"login": { "user": "root", "os": "", "ip": "::1", "proxy": "" }
```

- [shutdown_data](#)

監査ログプラグインの終了に関する情報。値は、これらのアイテムを含むハッシュです: [server_id](#) この項目は、[class](#) 値と [event](#) 値がそれぞれ [audit](#) および [shutdown](#) である監査レコードに対してのみ発生します。

例:

```
"shutdown_data": { "server_id": 1 }
```

- [startup_data](#)

監査ログプラグインの初期化に関する情報。値は、これらのアイテムを含むハッシュです: [server_id](#), [os_version](#), [mysql_version](#), [args](#)。この項目は、[class](#) 値と [event](#) 値がそれぞれ [audit](#) および [startup](#) である監査レコードに対してのみ発生します。

例:

```
"startup_data": { "server_id": 1,
  "os_version": "i686-Linux",
  "mysql_version": "5.7.21-log",
  "args": ["/usr/local/mysql/bin/mysqld",
    "--loose-audit-log-format=JSON",
    "--log-error=log.err",
    "--pid-file=mysqld.pid",
    "--port=3306" ] }
```

- [table_access_data](#)

テーブルへのアクセスに関する情報。値は、これらのアイテムを含むハッシュです: [db](#), [table](#), [query](#), [sql_command](#)、この項目は、[class](#) 値が [table_access](#) の監査レコードに対してのみ発生します。

例:

```
"table_access_data": { "db": "test",
  "table": "t1",
  "query": "INSERT INTO t1 (i) VALUES(1),(2),(3)",
  "sql_command": "insert" }
```

- [timestamp](#)

監査イベントが生成された日時を示す `YYYY-MM-DD hh:mm:ss` 形式の UTC 値を表す文字列。たとえば、クライアントから受信した SQL ステートメントの実行に対応するイベントの [timestamp](#) 値は、受信時ではなく、ステートメントの終了後に発生します。

例:

```
"timestamp": "2019-10-03 13:50:01"
```

同じ [timestamp](#) 値を持つ監査レコードの場合、[id](#) 値によってそれらが区別され、順序が形成されます。監査ログ内では、[timestamp/id](#) のペアは一意です。これらのペアは、ログ内のイベントの場所を識別するブックマークです。

JSON 形式の監査レコードのトップレベル項目に関連付けられたハッシュ値内には、次の項目が表示されます:

- [args](#)

MySQL サーバーの起動時にコマンド行またはオプションファイルで指定されたオプションの配列。最初のオプションは、サーバー実行可能ファイルへのパスです。

例:

```
"args": ["/usr/local/mysql/bin/mysqld",
"--loose-audit-log-format=JSON",
"--log-error=log.err",
"--pid-file=mysqld.pid",
"--port=3306"]
```

- **command**

監査イベントを生成した命令 (サーバーがクライアントから受信したコマンドなど) のタイプを表す文字列。

例:

```
"command": "Query"
```

- **connection_type**

サーバーへの接続のセキュリティ状態。許可される値は、[tcp/ip](#) (暗号化なしで確立された TCP/IP 接続)、[ssl](#) (暗号化で確立された TCP/IP 接続)、[socket](#) (Unix ソケットファイル接続)、[named_pipe](#) (Windows 名前付きパイプ接続) および [shared_memory](#) (Windows 共有メモリー接続) です。

例:

```
"connection_type": "tcp/tcp"
```

- **db**

データベース名を表す文字列。[connection_data](#) の場合、これはデフォルトのデータベースです。[table_access_data](#) の場合は、テーブルデータベースです。

例:

```
"db": "test"
```

- **host**

クライアントのホスト名を表す文字列。

例:

```
"host": "localhost"
```

- **ip**

クライアントの IP アドレスを表す文字列。

例:

```
"ip": "::1"
```

- **mysql_version**

MySQL サーバーのバージョンを表す文字列。これは、セッション内の [VERSION\(\)](#) 関数または [version](#) システム変数の値と同じです。

例:

```
"mysql_version": "5.7.21-log"
```

- **os**

認証プロセス中に使用される外部ユーザー名を表す文字列。クライアントの認証に使用されるプラグインによって設定されます。ネイティブ (組込み) MySQL 認証の場合、またはプラグインで値が設定されていない場合、この属性は空です。この値は、[external_user](#) システム変数の値と同じです。[セクション6.2.18「プロキシユーザー」](#)を参照してください。

例:

```
"os": "jeffrey"
```

- [os_version](#)

サーバーが構築された、または実行されているオペレーティングシステムを表す文字列。

例:

```
"os_version": "i686-Linux"
```

- [プロキシ](#)

プロキシユーザーを表す文字列 ([セクション6.2.18「プロキシユーザー」](#)を参照)。ユーザーのプロキシ処理が有効になっていない場合は、値が空です。

例:

```
"proxy": "developer"
```

- [クエリー](#)

SQL ステートメントのテキストを表す文字列。この値は、空にすることができます。長い値は、切り捨てられる可能性があります。監査ログファイル自体などの文字列は、UTF-8 (1文字あたり最大4バイト) を使用して記述されるため、この値が変換の結果となる場合があります。たとえば、元のステートメントは、SJIS 文字列としてクライアントから受信された可能性があります。

例:

```
"query": "DELETE FROM t1"
```

- [server_id](#)

サーバー ID を表す符号なし整数。これは、[server_id](#) システム変数の値と同じです。

例:

```
"server_id": 1
```

- [sql_command](#)

SQL ステートメントのタイプを示す文字列。

例:

```
"sql_command": "insert"
```

値は [statement/sql/xxx](#) コマンドカウンタに対応しています。たとえば、xxx は、[DROP TABLE](#) および [SELECT](#) ステートメントのそれぞれ [drop_table](#) および [select](#) です。次のステートメントは、使用可能な名前を表示します:

```
SELECT REPLACE(EVENT_NAME, 'statement/sql/', '') AS name
FROM performance_schema.events_statements_summary_global_by_event_name
WHERE EVENT_NAME LIKE 'statement/sql/%'
ORDER BY name;
```

- [status](#)

コマンドのステータスを表す符号なし整数 (成功した場合は 0、エラーが発生した場合はゼロ以外)。これは、[mysql_errno\(\)](#) C API 関数の値と同じです。

監査ログには、SQLSTATE 値またはエラーメッセージが含まれていません。エラーコード、SQLSTATE 値、およびメッセージ間の関連性を確認する方法については、[Server Error Message Reference](#)を参照してください。

警告のログは記録されません。

例:

```
"status": 1051
```

- [table](#)

テーブル名を表す文字列。

例:

```
"table": "t1"
```

- [user](#)

ユーザー名を表す文字列。意味は、[user](#) が発生するアイテムによって異なります:

- [account](#) アイテム内の [user](#) は、サーバーがクライアントを認証したユーザーを表す文字列です。これは、サーバーが権限チェックに使用するユーザー名です。
- [login](#) アイテム内の [user](#) は、クライアントによって送信されるユーザー名を表す文字列です。

例:

```
"user": "root"
```

6.4.5.5 監査ロギング特性の構成

このセクションでは、監査ログプラグインがイベントを書き込むファイル、書き込まれたイベントの形式、ログファイルの圧縮と暗号化を有効にするかどうかなど、監査ロギング特性を構成する方法について説明します。

- [監査ログファイルのネーミング規則](#)
- [監査ログファイル形式の選択](#)
- [監査ログファイルの圧縮](#)
- [監査ログファイルの暗号化](#)
- [監査ログファイルの手動での解凍および復号化](#)
- [MySQL 8.0.17 より前の監査ログファイルの暗号化](#)
- [監査ログファイルの領域管理](#)
- [監査ロギングの書き込み戦略](#)

注記

ここで説明する暗号化機能は、現在の暗号化機能を以前の制限付き機能と比較するセクションを除き、MySQL 8.0.17 の時点で適用されます。[MySQL 8.0.17 より前の監査ログファイルの暗号化](#)を参照してください。

監査ロギングに影響するユーザー定義関数およびシステム変数の詳細は、[監査ログ関数](#) および [監査ログのオプションおよび変数](#)を参照してください。

監査ログプラグインは、イベントの内容またはイベントの発生元のアカウントに基づいて、監査ログファイルに書き込む監査イベントを制御することもできます。セクション6.4.5.7「監査ログのフィルタリング」を参照してください。

監査ログファイルのネーミング規則

監査ログファイル名を構成するには、サーバーの起動時に `audit_log_file` システム変数を設定します。サーバーデータディレクトリのデフォルト名は `audit.log` です。セキュリティを最適化するには、MySQL サーバーおよびログを表示する正当な理由があるユーザーのみがアクセスできるディレクトリに監査ログを書き込みます。

プラグインは、`audit_log_file` 値を、オプションの先頭ディレクトリ名、ベース名、およびオプションの接尾辞で構成されるものとして解釈します。圧縮または暗号化が有効になっている場合、有効なファイル名 (ログファイルの作成に実際に使用される名前) は、追加の接尾辞があるため、構成されたファイル名とは異なります:

- 圧縮が有効な場合、プラグインは `.gz` の接尾辞を追加します。
- 暗号化が有効な場合、プラグインは `.pwd_id.enc` の接尾辞を追加します。ここで、`pwd_id` はログファイル操作に使用する暗号化パスワードを示します。監査ログプラグインは、暗号化パスワードを鍵リングに格納します。監査ログファイルの暗号化を参照してください。

有効な監査ログファイル名は、構成されたファイル名に適用可能な圧縮および暗号化接尾辞を追加した結果の名前です。たとえば、構成された `audit_log_file` 値が `audit.log` の場合、有効なファイル名は次のテーブルに示す値のいずれかになります。

有効な機能	有効なファイル名
圧縮または暗号化なし	<code>audit.log</code>
Compression	<code>audit.log.gz</code>
暗号化	<code>audit.log.pwd_id.enc</code>
圧縮、暗号化	<code>audit.log.gz.pwd_id.enc</code>

`pwd_id` は、ファイルの暗号化または復号化に使用されるパスワードの ID を示します。`pwd_id` 形式は `pwd_timestamp-seq` です。ここでは:

- `pwd_timestamp` は、パスワードがいつ作成されたかを示す `YYYYMMDDThhmmss` 形式の UTC 値です。
- `seq` は順序番号です。順序番号は 1 から始まり、同じ `pwd_timestamp` 値を持つパスワードに対して増加します。

`pwd_id` パスワード ID の値の例を次に示します:

```
20190403T142359-1
20190403T142400-1
20190403T142400-2
```

パスワードをキーリングに格納するための対応するキーリング ID を構築するために、監査ログプラグインは `audit_log-` の接頭辞を `pwd_id` 値に追加します。前述のパスワード ID の例では、対応するキーリング ID は次のとおりです:

```
audit_log-20190403T142359-1
audit_log-20190403T142400-1
audit_log-20190403T142400-2
```

監査ログプラグインによって暗号化に現在使用されているパスワードの ID は、`pwd_timestamp` 値がもっとも大きいパスワードです。複数のパスワードにその `pwd_timestamp` 値がある場合、現在のパスワード ID は順序番号が最も大きいものになります。たとえば、前述のパスワード ID のセットでは、パスワード ID のうち 2 つの ID のタイムスタンプが最大の `20190403T142400` であるため、現在のパスワード ID は順序番号 (2) が最も大きいものになります。

監査ログプラグインは、有効な監査ログファイル名に基づいて、初期化および終了時に特定のアクションを実行します:

- 初期化中に、プラグインは監査ログファイル名を持つファイルがすでに存在するかどうかをチェックし、存在する場合は名前を変更します。(この場合、プラグインは、監査ログプラグインの実行中に以前のサーバー起動が予期せず終了したことを前提としています。)次に、プラグインは新しい空の監査ログファイルに書き込みます。

- 終了時に、プラグインは監査ログファイルの名前を変更します。
- ファイルの名前変更 (プラグインの初期化時または終了時) は、サイズベースのログファイルの自動ローテーションの通常のルールに従って行われます。 [サイズベースの監査ログファイルのローテーション](#) を参照してください。

監査ログファイル形式の選択

監査ログファイル形式を構成するには、サーバーの起動時に `audit_log_format` システム変数を設定します。デフォルトでは、フォーマットは `NEW` (新しいスタイルの XML フォーマット) です。各形式についての詳細は、[セクション 6.4.5.4 「監査ログファイル形式」](#) を参照してください。

`audit_log_format` を変更する場合は、`audit_log_file` も変更することをお勧めします。それ以外の場合は、ベース名は同じで形式が異なる 2 つのログファイルのセットが存在します。

監査ログファイルの圧縮

監査ログファイルの圧縮は、任意のロギング形式に対して有効にできます。

監査ログファイルの圧縮を構成するには、サーバーの起動時に `audit_log_compression` システム変数を設定します。許可される値は、`NONE` (圧縮なし、デフォルト) および `GZIP` (GNU Zip 圧縮) です。

圧縮と暗号化の両方が有効な場合、圧縮は暗号化の前に行われます。元のファイルを手動でリカバリするには、最初に復号化してから解凍します。 [監査ログファイルの手動での解凍および復号化](#) を参照してください。

監査ログファイルの暗号化

監査ログファイルの暗号化は、任意のロギング形式に対して有効にできます。暗号化は、ユーザー定義のパスワードに基づきます (監査ログプラグインによって生成される初期パスワードを除く)。この機能を使用するには、監査ロギングでパスワード記憶域に使用されるため、MySQL キーリングを有効にする必要があります。どのキーリングプラグインも使用できます。手順については、[セクション 6.4.4 「MySQL キーリング」](#) を参照してください。

監査ログファイルの暗号化を構成するには、サーバーの起動時に `audit_log_encryption` システム変数を設定します。許可される値は、`NONE` (暗号化なし、デフォルト) および `AES` (AES-256-CBC 暗号化) です。

実行時に暗号化パスワードを設定または取得するには、次のユーザー定義関数 (UDF) を使用します:

- 現在の暗号化パスワードを設定するには、`audit_log_encryption_password_set()` を起動します。この関数は、キーリングに新しいパスワードを格納します。暗号化が有効な場合は、現在のログファイルの名前を変更するログファイルローテーション操作も実行され、パスワードで暗号化された新しいログファイルが開始されます。ファイル名の変更は、サイズベースの自動ログファイルローテーションの通常のルールに従って行われます。 [サイズベースの監査ログファイルのローテーション](#) を参照してください。

`audit_log_password_history_keep_days` システム変数がゼロ以外の場合、`audit_log_encryption_password_set()` を起動すると、古いアーカイブ監査ログの暗号化パスワードも期限切れになります。パスワードのアーカイブや有効期限など、監査ログのパスワード履歴の詳細は、その変数の説明を参照してください。

- 現在の暗号化パスワードを取得するには、引数を指定せずに `audit_log_encryption_password_get()` を起動します。ID でパスワードを取得するには、現在のパスワードまたはアーカイブされたパスワードのキーリング ID を指定する引数を渡します。

存在する監査ログ鍵リング ID を確認するには、パフォーマンススキーマ `keyring_keys` テーブルをクエリーします:

```
mysql> SELECT KEY_ID FROM performance_schema.keyring_keys
      WHERE KEY_ID LIKE 'audit_log%'
      ORDER BY KEY_ID;
+-----+
| KEY_ID |
+-----+
| audit_log-20190415T152248-1 |
| audit_log-20190415T153507-1 |
| audit_log-20190416T125122-1 |
| audit_log-20190416T141608-1 |
+-----+
```

監査ログの暗号化 UDF の詳細は、[監査ログ関数](#) を参照してください。

監査ログプラグインが初期化されるときに、ログファイルの暗号化が有効になっていることが判明すると、鍵リングに監査ログの暗号化パスワードが含まれているかどうかをチェックされます。そうでない場合、プラグインはランダムな初期暗号化パスワードを自動的に生成し、キーリングに格納します。このパスワードを検出するには、`audit_log_encryption_password_get()` を起動します。

圧縮と暗号化の両方が有効な場合、圧縮は暗号化の前に行われます。元のファイルを手動でリカバリするには、最初に復号化してから解凍します。[監査ログファイルの手動での解凍および復号化](#)を参照してください。

監査ログファイルの手動での解凍および復号化

監査ログファイルは、標準ツールを使用して圧縮解除および復号化できます。これは、閉じられた (アーカイブされた) 使用されなくなったログファイルに対してのみ実行してください。監査ログプラグインが現在書き込んでいるログファイルに対しては実行しないでください。アーカイブログファイルは、ベース名の直後のファイル名にタイムスタンプを含めるように監査ログプラグインによって名前が変更されているため、認識できます。

この説明では、`audit_log_file` が `audit.log` に設定されていると仮定します。その場合、アーカイブされた監査ログファイルには、次のテーブルに示すいずれかの名前が付けられます。

有効な機能	アーカイブファイル名
圧縮または暗号化なし	<code>audit.timestamp.log</code>
Compression	<code>audit.timestamp.log.gz</code>
暗号化	<code>audit.timestamp.log.pwd_id.enc</code>
圧縮、暗号化	<code>audit.timestamp.log.gz.pwd_id.enc</code>

[監査ログファイルのネーミング規則](#) で説明されているように、`pwd_id` 形式は `pwd_timestamp-seq` です。したがって、アーカイブされた暗号化ログファイルの名前には、実際には 2 つのタイムスタンプが含まれています。1 つ目はファイルのローテーション時間を示し、2 つ目は暗号化パスワードが作成された時間を示します。

次のアーカイブ暗号化ログファイル名のセットについて考えてみます:

```
audit.20190410T205827.log.20190403T185337-1.enc
audit.20190410T210243.log.20190403T185337-1.enc
audit.20190415T145309.log.20190414T223342-1.enc
audit.20190415T151322.log.20190414T223342-2.enc
```

各ファイル名には、一意のローテーション時間タイムスタンプがあります。対照的に、パスワードのタイムスタンプは一意ではありません:

- 最初の 2 つのファイルのパスワード ID と順序番号は同じです (`20190403T185337-1`)。これらの暗号化パスワードは同じです。
- 2 つ目のファイルのパスワード ID (`20190414T223342`) は同じですが、順序番号 (`1`、`2`) が異なります。これらのファイルの暗号化パスワードは異なります。

圧縮されたログファイルを手動で解凍するには、`gunzip`、`gzip -d` または同等のコマンドを使用します。例:

```
gunzip -c audit.timestamp.log.gz > audit.timestamp.log
```

暗号化されたログファイルを手動で復号化するには、`openssl` コマンドを使用します。例:

```
openssl enc -d -aes-256-cbc -pass pass:password -md sha256
-in audit.timestamp.log.pwd_id.enc
-out audit.timestamp.log
```

このコマンドを実行するには、暗号化パスワードである `password` を取得する必要があります。これを行うには、`audit_log_encryption_password_get()` を使用します。たとえば、監査ログファイル名が `audit.20190415T151322.log.20190414T223342-2.enc` の場合、パスワード ID は `20190414T223342-2` で、キーリング ID は `audit-log-20190414T223342-2` です。次のようなキーリングパスワードを取得します:

```
SELECT audit_log_encryption_password_get('audit-log-20190414T223342-2');
```

圧縮と暗号化の両方が監査ロギングに対して有効になっている場合、圧縮は暗号化の前に行われます。この場合、ファイル名には、これらの操作が発生する順序に対応する `.gz` および `.pwd_id.enc` 接尾辞が追加されます。元のファイルを手動でリカバリするには、操作を逆に実行します。つまり、最初にファイルを復号化してから解凍します:

```
openssl enc -d -aes-256-cbc -pass pass:password -md sha256
-in audit.timestamp.log.gz.pwd_id.enc
-out audit.timestamp.log.gz
gunzip -c audit.timestamp.log.gz > audit.timestamp.log
```

MySQL 8.0.17 より前の監査ログファイルの暗号化

このセクションでは、パスワード履歴 (パスワードのアーカイブと有効期限を含む) が実装されたときの、MySQL 8.0.17 の前後の監査ログファイルの暗号化機能の違いについて説明します。また、監査ログプラグインが 8.0.17 より前のバージョンから MySQL 8.0.17 以上へのアップグレードを処理する方法も示します。

機能	MySQL 8.0.17 より前	MySQL 8.0.17 の時点
パスワードの数	単一パスワードのみ	複数のパスワードを許可
暗号化されたログファイル名	.enc 接尾辞	.pwd_id.enc 接尾辞
パスワードキーリング ID	audit_log	audit_log-pwd_id
パスワード履歴	いいえ	はい

MySQL 8.0.17 より前ではパスワード履歴がないため、新しいパスワードを設定すると古いパスワードにアクセスできなくなり、MySQL Enterprise Audit は古いパスワードで暗号化されたログファイルを読み取ることができなくなりました。これらのファイルを手動で復号化する必要があると予想される場合は、以前のパスワードのレコードを保持する必要があります。

下位バージョンから MySQL 8.0.17 以上にアップグレードするときに監査ログファイルの暗号化が有効になっている場合、監査ログプラグインは次のアップグレードアクションを実行します:

- プラグインの初期化中に、プラグインはキーリング ID が `audit_log` の暗号化パスワードをチェックします。見つかった場合、プラグインは `audit_log-pwd_id` 形式のキーリング ID を使用してパスワードを複製し、現在の暗号化パスワードとして使用します。(`pwd_id` 構文の詳細は、[監査ログファイルのネーミング規則](#) を参照してください。)
- 既存の暗号化されたログファイルには、接尾辞 `.enc` が付きます。プラグインは、接尾辞が `.pwd_id.enc` になるようにこれらの名前を変更しませんが、ID が `audit_log` のキーがキーリングに残っているかぎり読み取ることができません。
- パスワードのクリーンアップが発生したときに、プラグインが `audit_log-pwd_id` 形式のキーリング ID を持つパスワードを期限切れにすると、`audit_log` のキーリング ID を持つパスワードも期限切れになります (存在する場合)。(この時点で、`.pwd_id.enc` ではなく `.enc` という接尾辞を持つ暗号化されたログファイルはプラグインで読み取れなくなるため、不要になったと想定されます。)

監査ログファイルの領域管理

監査ログファイルは、非常に大きくなり、大量のディスク領域を消費する可能性があります。ログファイルで使用する領域の管理を有効にするために、監査ログプラグインでは、ファイルサイズに基づいて手動または自動でログファイルをローテーションできます。MySQL 8.0.24 では、このプラグインは JSON 形式のログファイルのログファイルプルーニングもサポートしています。

監査ログファイルの領域管理機能では、`audit_log_rotate_on_size`、`audit_log_flush` および `audit_log_prune_seconds` システム変数が使用され、次のように組み合わせられています:

- `audit_log_rotate_on_size` が 0 (デフォルト) の場合:
 - 自動ログファイルローテーションが無効です。手動で実行しないかぎり、ローテーションは発生しません。
 - 手動で名前を変更した後、`audit_log_flush` を使用して現在のログファイルを閉じ、再度開きます。
 - ログファイルのプルーニングを有効にできず、`audit_log_prune_seconds` は無効です。
- `audit_log_rotate_on_size` が 0 より大きい場合:
 - 自動ローテーションは、現在のログファイルへの書き込みによってサイズがこの値を超える場合に発生します。監査ログプラグインは、ファイルを閉じて名前を変更し、新しいログファイルを開きます。

- ログファイルのプルーニングを有効にでき、`audit_log_prune_seconds` はプルーニングが行われるかどうかを判断します。
- `audit_log_flush` には影響はありません。
- 自動サイズベースローテーションは、後で説明する他のいくつかの条件下でも発生します。

注記

名前が変更されたログファイルは自動的に削除されません。たとえば、サイズベースのログファイルローテーションでは、名前が変更されたログファイルは名前シーケンスの最後からローテーションされません。かわりに、一意の名前を持ち、無期限に蓄積されます。過剰な領域使用を回避するには:

- MySQL 8.0.24 以降 (JSON 形式のログファイル用): [監査ログファイルのプルーニング](#) の説明に従って、ログファイルのプルーニングを有効にします。
- それ以外の場合 (JSON 以外のファイルの場合、またはすべてのログ形式について MySQL 8.0.24 より前の場合): 古いファイルを定期的に削除し、必要に応じて最初にバックアップします。バックアップされたログファイルが暗号化されている場合は、後でファイルを復号化する必要があるときに、対応する暗号化パスワードも安全な場所にバックアップします。

次の各セクションでは、ログファイルのローテーションとプルーニングについて詳しく説明します。

- [手動監査ログファイルローテーション](#)
- [サイズベースの監査ログファイルのローテーション](#)
- [監査ログファイルのプルーニング](#)

手動監査ログファイルローテーション

`audit_log_rotate_on_size` が 0 (デフォルト) の場合、手動で実行しないかぎり、ログローテーションは発生しません。この場合、`audit_log_flush` の値が無効から有効に変更されると、監査ログプラグインはログファイルを閉じてから再度開きます。ログファイル名の変更は、サーバーの外部で実行される必要があります。ログファイル名が `audit.log` で、`audit.log.3` を介して `audit.log.1` という名前を循環して、最新の 3 つのログファイルを保持するとします。Unix 上で、次のように手動でローテーションを実行します。

1. コマンド行から、現在のログファイル名を変更します。

```
mv audit.log.2 audit.log.3
mv audit.log.1 audit.log.2
mv audit.log.1
```

この方法では、現在の `audit.log.3` コンテンツが上書きされ、アーカイブログファイルの数とそれらが使用する領域にバインドが配置されます。

2. この時点で、プラグインは引き続き、`audit.log.1` に名前が変更された現在のログファイルに書き込みます。サーバーに接続し、ログファイルをフラッシュします。これにより、プラグインはログファイルを閉じて、新しい `audit.log` ファイルログを再度開きます。

```
SET GLOBAL audit_log_flush = ON;
```

`audit_log_flush` は、その値が `OFF` のままであるため、別のフラッシュを実行するために再度有効にする前に明示的に無効にする必要がないという点で特殊です。

注記

圧縮または暗号化が有効になっている場合、ログファイル名には、有効な機能を示す接尾辞と、暗号化が有効になっている場合はパスワード ID が含まれます。ファイル名にパスワード ID が含まれている場合は、復号化操作に使用するパスワードを決定できるように、手動で名前を変更するファイルの名前に ID を保持してください。

注記

JSON 形式のロギングの場合、監査ログファイルの名前を手動で変更すると、監査ログプラグインはログファイル順序の一部であると判断できなくなるため、ログ読取り機能で使用できなくなります (セクション6.4.5.6「監査ログファイルの読取り」を参照)。かわりに、サイズベースのローテーションを使用するように 0 より大きい `audit_log_rotate_on_size` を設定することを検討してください。

サイズベースの監査ログファイルのローテーション

`audit_log_rotate_on_size` が 0 よりも大きい場合は、`audit_log_flush` を設定しても効果がありません。代わりに、現在のログファイルへの書き込みによってそのサイズが `audit_log_rotate_on_size` 値を超えるたびに、監査ログプラグインは自動的にファイルを閉じて名前を変更し、新しいログファイルを開きます。

自動サイズベースローテーションは、次の条件下でも発生します:

- プラグインの初期化中に、監査ログファイル名を持つファイルがすでに存在する場合 (監査ログファイルのネーミング規則を参照)。
- プラグインの終了時。
- 暗号化が有効になっている場合に、`audit_log_encryption_password_set()` 関数をコールして暗号化パスワードを設定するとき。(暗号化が無効になっている場合、ローテーションは行われません。)

プラグインは、ベース名の直後にタイムスタンプを挿入して、元のファイルの名前を変更します。たとえば、ファイル名が `audit.log` の場合、プラグインはその名前を `audit.20190115T140633.log` などの値に変更します。タイムスタンプは、`YYYYMMDDThhmmss` 形式の UTC 値です。タイムスタンプは、XML ロギングのローテーション時間と、JSON ロギングのためにファイルに最後に書き込まれたイベントのタイムスタンプを示します。

監査ログファイルのプルーニング

MySQL 8.0.24 では、監査ログプラグインは JSON 形式の監査ログファイルのプルーニングをサポートしています。この機能を有効にするには:

- `audit_log_format` を JSON に設定します。
- `audit_log_rotate_on_size` を 0 より大きい値に設定して、ログファイルのローテーションが発生するサイズを指定します。
- 0 より大きい `audit_log_prune_seconds` を設定して、ログファイルがプルーニングの対象になるまでの秒数を指定します。

ログファイルのプルーニングは、次の条件下で発生します:

- プラグインの初期化中。
- 現在のログファイルがローテーションサイズを超えたためにサイズベースの自動ローテーションが発生した場合。
- `SET GLOBAL audit_log_prune_seconds` が実行時に実行される場合。

プルーニングポイントは、現在の時間から `audit_log_prune_seconds` の値を引いた値です。ローテーションされた JSON 形式のログファイルでは、各ファイル名のタイムスタンプ部分は、ファイルに最後に書き込まれたイベントのタイムスタンプを示します。プルーニングが発生すると、監査ログプラグインはファイル名のタイムスタンプを使用して、プルーニングポイントより古いイベントのみを含むファイルを特定し、それらを削除します。

監査ロギングの書き込み戦略

監査プラグインは、ログの書き込みに関する複数の戦略のいずれかを使用できます。戦略に関係なく、ロギングはベストエフォートベースで発生するため、一貫性は保証されません。

書き込み戦略を指定するには、サーバーの起動時に `audit_log_strategy` システム変数を設定します。デフォルトでは、戦略の値は `ASYNCHRONOUS` であり、プラグインは非同期的にログをバッファーに記録し、バッファーがいっぱいの場合は待機します。ファイルシステムのキャッシュ処理を使用するか (`SEMISYNCHRONOUS`)、各書き込みリクエ

ストのあとに `sync()` を呼び出して出力を強制すれば (`SYNCHRONOUS`)、待機しないように (`PERFORMANCE`)、または同期的にログを記録するようにプラグインに指示できます。

非同期書き込み戦略の場合、`audit_log_buffer_size` システム変数はバイト単位のバッファサイズです。バッファサイズを変更するには、サーバー起動時にこの変数を設定します。このプラグインでは、初期化時に割り当てられ、終了時に削除される単一のバッファが使用されます。プラグインは、このバッファを非同期でない書き込み戦略に割り当てません。

非同期ロギングの戦略には、次のような特性があります。

- サーバーのパフォーマンスと拡張性への影響が最小限です。
- できるかぎり最短の時間 (つまり、バッファを割り当てる時間とそのバッファにイベントをコピーする時間を足した時間) で、監査イベントを生成するスレッドをブロックします。
- 出力はバッファに書き込まれます。個別のスレッドがバッファからログファイルへの書き込みに対処します。

非同期ロギングでは、ファイルへの書き込み中に問題が発生した場合、またはプラグインが正常に停止しない場合 (サーバーホストが予期せず終了した場合など)、ログファイルの整合性が損なわれる可能性があります。このリスクを減らすには、同期ロギングが使用されるように `audit_log_strategy` を設定します。

`PERFORMANCE` 戦略のデメリットは、バッファがいっぱいの場合にイベントが破棄される点です。負荷の高いサーバーの場合、監査ログにイベントが欠落している可能性があります。

6.4.5.6 監査ログファイルの読取り

監査ログプラグインは、JSON 形式の監査ログファイルを読み取るための SQL インタフェースを提供するユーザー定義関数をサポートしています。(この機能は、他の形式で書き込まれたログファイルには適用されません。)

監査ログプラグインが初期化され、JSON ロギング用に構成されている場合、読取り可能な監査ログファイルを検索する場所として、現在の監査ログファイルを含むディレクトリが使用されます。プラグインは、`audit_log_file` システム変数の値からファイルの場所、ベース名および接尾辞を決定し、次のパターンに一致する名前のファイルを検索します ([...]はオプションのファイル名部分を示します)：

```
basename[.timestamp].suffix[.gz][[.pwd_id].enc]
```

ファイル名が `.enc` で終わる場合、ファイルは暗号化され、暗号化されていない内容を読み取るには、キーリングから取得した復号化パスワードが必要です。監査ログプラグインは、復号化パスワードの鍵リング ID を次のように決定します：

- `.enc` の前に `pwd_id` がある場合、キーリング ID は `audit_log-pwd_id` です。
- `.enc` の前に `pwd_id` が付いていない場合、監査ログの暗号化パスワード履歴が実装される前からの古い名前がファイルに含まれます。キーリング ID は `audit_log` です。

暗号化された監査ログファイルの詳細は、[監査ログファイルの暗号化](#) を参照してください。

プラグインは、手動で名前が変更され、パターンと一致しないファイルと、パスワードで暗号化されたファイルがキーリングで使用できなくなったファイルを見捨てます。プラグインは残りの各候補ファイルを開き、ファイルに JSON 監査イベントが実際に含まれていることを確認し、各ファイルの最初のイベントのタイムスタンプを使用してファイルをソートします。その結果、ログ読取りユーザー定義関数 (UDF) を使用してアクセスされる一連のファイルが生成されます：

- `audit_log_read()` は、監査ログからイベントを読み取るか、読取りプロセスをクローズします。
- `audit_log_read_bookmark()` は、最後に書き込まれた監査ログイベントのブックマークを返します。このブックマークは、読取りを開始する場所を示すために `audit_log_read()` に渡すのに適しています。

`audit_log_read()` はオプションの JSON 文字列引数を取り、いずれかの関数のコールが成功したときに返される結果は JSON 文字列です。

関数を使用して監査ログを読み取るには、次の原則に従います：

- `audit_log_read()` をコールして、特定の位置または現在の位置から開始するイベントを読み取るか、検針をクローズします：

- 監査ログの読取り順序を初期化するには、開始位置を示す引数を渡します。これを行うには、`audit_log_read_bookmark()` によって返されたブックマークを渡す方法があります:

```
SELECT audit_log_read(audit_log_read_bookmark());
```

- 順序内の現在の位置からの読取りを続行するには、位置を指定せずに `audit_log_read()` をコールします:

```
SELECT audit_log_read();
```

- 読取り順序を明示的にクローズするには、`JSON null` 引数を渡します:

```
SELECT audit_log_read('null');
```

読取りを明示的にクローズする必要はありません。読取りは、セッションが終了するか、開始位置を示す引数を指定して `audit_log_read()` をコールすることで、新しい読取り順序が初期化されると暗黙的にクローズされます。

- `audit_log_read()` を正常にコールしてイベントを読み取ると、監査イベントの配列を含む `JSON` 文字列が返されます:
 - 返された配列の最終値が `JSON null` 値でない場合は、読み取られたばかりのイベントのあとにさらにイベントが存在し、`audit_log_read()` を再度呼び出してさらに多くのイベントを読み取ることができます。
 - 返される配列の最終値が `JSON null` 値である場合、現在の読取り順序で読み取られるイベントは残っていません。

`null` 以外の各配列要素は、`JSON` ハッシュとして表されるイベントです。例:

```
[
  {
    "timestamp": "2020-05-18 13:39:33", "id": 0,
    "class": "connection", "event": "connect",
    ...
  },
  {
    "timestamp": "2020-05-18 13:39:33", "id": 1,
    "class": "general", "event": "status",
    ...
  },
  {
    "timestamp": "2020-05-18 13:39:33", "id": 2,
    "class": "connection", "event": "disconnect",
    ...
  },
  null
]
```

`JSON` 形式の監査イベントの内容の詳細は、[JSON 監査ログファイル形式](#) を参照してください。

- 位置を指定しないイベントを読み取る `audit_log_read()` コールは、次のいずれかの条件下でエラーを生成します:
 - 読取り順序は、`audit_log_read()` に位置を渡して初期化されていません。
 - 現在の読み取りシーケンスで読み取られるイベントは残っていません。つまり、`audit_log_read()` は以前に `JSON null` 値で終わる配列を返しました。
 - 最新の読取り順序は、`JSON null` 値を `audit_log_read()` に渡すことでクローズされました。

これらの条件下でイベントを読み取るには、まず位置を指定する引数を使用して `audit_log_read()` をコールし、読取り順序を初期化する必要があります。

`audit_log_read()` の位置を指定するには、読取りを開始する場所を示す引数を含めます。たとえば、特定のイベントを一意に識別する `timestamp` および `id` 要素を含む `JSON` ハッシュであるブックマークを渡します。次に、`audit_log_read_bookmark()` 関数をコールして取得したブックマークの例を示します:

```
mysql> SELECT audit_log_read_bookmark();
```

```

| audit_log_read_bookmark() |
+-----+
| {"timestamp": "2020-05-18 21:03:44", "id": 0} |
+-----+

```

現在のブックマークを `audit_log_read()` に渡すと、ブックマーク位置から始まるイベント読取りが初期化されます:

```

mysql> SELECT audit_log_read(audit_log_read_bookmark());
+-----+
| audit_log_read(audit_log_read_bookmark()) |
+-----+
| [{"timestamp": "2020-05-18 22:41:24", "id": 0, "class": "connection", ... |
+-----+

```

`audit_log_read()` の引数はオプションです。存在する場合は、読取り順序をクローズする JSON null 値または JSON ハッシュを指定できます。

`audit_log_read()` のハッシュ引数内では、項目はオプションであり、読取りを開始する位置や読み取るイベントの数など、読取り操作の側面を制御します。次の項目は重要です (他の項目は無視されます):

- **start**: 最初に読み取るイベントの監査ログ内の位置。位置はタイムスタンプとして指定され、タイムスタンプ値以降に発生する最初のイベントから読取りが開始されます。start アイテムの形式は次のとおりです。ここで、value はリテラルのタイムスタンプ値です:

```
"start": { "timestamp": "value" }
```

start 品目は、MySQL 8.0.22 で許可されています。

- **timestamp, id**: 最初に読み取るイベントの監査ログ内の位置。timestamp アイテムと id アイテムは、特定のイベントを一意に識別するブックマークで構成されます。audit_log_read() 引数にいずれかの項目が含まれている場合、位置を完全に指定するには両方を含める必要があります。そうしないと、エラーが発生します。
- **max_array_length**: ログから読み取るイベントの最大数。この項目を省略した場合、デフォルトでは、ログの最後まで、または読取りバッファがいっぱいになるまで (いずれか早い方まで) 読み取られます。

開始位置を `audit_log_read()` に指定するには、start アイテムまたは timestamp アイテムと id アイテムで構成されるブックマークを含むハッシュ引数を渡します。ハッシュ引数に start アイテムとブックマークの両方が含まれている場合は、エラーが発生します。

ハッシュ引数で開始位置が指定されていない場合、読取りは現在の位置から続行されます。

タイムスタンプ値に時間部分が含まれていない場合は、00:00:00 の時間部分が想定されます。

`audit_log_read()` で受け入れられる引数の例:

- 指定されたタイムスタンプ以降に発生する最初のイベントから開始するイベントを読み取ります:

```
audit_log_read({ "start": { "timestamp": "2020-05-24 12:30:00" } })
```

- 前の例と似ていますが、最大 3 つのイベントを読み取ります:

```
audit_log_read({ "start": { "timestamp": "2020-05-24 12:30:00" }, "max_array_length": 3 })
```

- 2020-05-24 00:00:00 以降に発生する最初のイベントから始まるイベントを読み取ります (タイムスタンプには時間部分が含まれないため、00:00:00 が想定されます):

```
audit_log_read({ "start": { "timestamp": "2020-05-24" } })
```

- 正確なタイムスタンプとイベント ID を持つイベントで始まるイベントを読み取ります:

```
audit_log_read({ "timestamp": "2020-05-24 12:30:00", "id": 0 })
```

- 前の例と似ていますが、最大 3 つのイベントを読み取ります:

```
audit_log_read({ "timestamp": "2020-05-24 12:30:00", "id": 0, "max_array_length": 3 })
```

- 検針シーケンスの現在の位置からイベントを読み取ります:

```
audit_log_read()
```

- 検針シーケンスの現在の位置から開始する最大 5 つのイベントを読み取ります:

```
audit_log_read({'max_array_length': 5})
```

- 現在の検針シーケンスをクローズします:

```
audit_log_read('null')
```

いずれかのログ読み取り関数から返された **JSON** 文字列は、必要に応じて操作できます。ブックマークを取得するコールによって次の値が生成されるとします:

```
mysql> SET @mark := audit_log_read_bookmark();
mysql> SELECT @mark;
+-----+
| @mark |
+-----+
| {"timestamp": "2020-05-18 16:10:28", "id": 2 } |
+-----+
```

その引数を使用して `audit_log_read()` をコールすると、複数のイベントを返すことができます。 `audit_log_read()` を最大 **N** イベント数に制限するには、その値を持つ `max_array_length` アイテムを文字列に追加します。たとえば、単一のイベントを読み取るには、次のように文字列を変更します:

```
mysql> SET @mark := JSON_SET(@mark, '$.max_array_length', 1);
mysql> SELECT @mark;
+-----+
| @mark |
+-----+
| {"id": 2, "timestamp": "2020-05-18 16:10:28", "max_array_length": 1} |
+-----+
```

変更された文字列が `audit_log_read()` に渡されると、使用可能なイベントの数に関係なく、最大 1 つのイベントを含む結果が生成されます。

MySQL 8.0.19 より前では、監査ログ UDF からの文字列の戻り値はバイナリ文字列です。バイナリ以外の文字列 (JSON 値を操作する関数など) を必要とする関数でバイナリ文字列を使用するには、バイナリ以外の文字列に変換します。たとえば、ブックマークを `JSON_SET()` に渡す前に、次のように `utf8mb4` に変換します:

```
SET @mark = CONVERT(@mark USING utf8mb4);
```

このステートメントは、MySQL 8.0.19 以上でも使用できます。これらのバージョンでは、基本的に no-op であり、無害です。

`audit_log_read()` が読み取るバイト数の制限を設定するには、`audit_log_read_buffer_size` システム変数を設定します。MySQL 8.0.12 では、この変数のデフォルトは 32KB で、実行時に設定できます。各クライアントは、`audit_log_read()` を使用するために `audit_log_read_buffer_size` のセッション値を適切に設定する必要があります。

`audit_log_read()` をコールするたびに、バッファサイズ内に収まる数の使用可能なイベントが返されます。バッファサイズ内に収まらないイベントはスキップされ、警告が生成されます。この動作では、アプリケーションの適切なバッファサイズを評価する際に次の要因を考慮してください:

- `audit_log_read()` へのコール数とコールごとに返されるイベントとの間にはトレードオフがあります:
 - バッファサイズを小さくすると、コールによって返されるイベントが少なくなるため、必要なコールが増えます。
 - バッファサイズが大きいくほど、コールはより多くのイベントを返すため、必要なコールは少なくなります。
- デフォルトサイズの 32KB など、バッファサイズが小さいほど、イベントがバッファサイズを超えてスキップされる可能性が高くなります。

MySQL 8.0.12 より前では、`audit_log_read_buffer_size` のデフォルトは 1MB で、すべてのクライアントに影響し、サーバーの起動時にのみ変更できます。

監査ログ読み取り関数の詳細は、[監査ログ関数](#) を参照してください。

6.4.5.7 監査ログのフィルタリング

注記

このセクションでは、監査ログプラグインおよび付随する監査テーブルと UDF がインストールされている場合の監査ログのフィルタリングの動作について説明します。プラグインがインストールされているが、付随する監査テーブルおよび UDF がインストールされていない場合、プラグインはレガシーフィルタリングモードで動作します (セクション 6.4.5.9 「レガシーモード監査ログのフィルタリング」を参照)。レガシーモードは、MySQL 5.7.13 より前、つまりルールベースのフィルタリングの導入前のフィルタリング動作です。

監査ログプラグインには、監査対象イベントのロギングをフィルタリングして制御する機能があります:

- 監査イベントは、次の特性を使用してフィルタできます:
 - ユーザーアカウント
 - 監査イベントクラス
 - 監査イベントサブクラス
 - 操作ステータスや実行された SQL ステートメントを示すイベントフィールドなどのイベントフィールドの値
- 監査フィルタリングはルールベースです:
 - フィルタ定義によって、一連の監査ルールが作成されます。定義は、前述の特性に基づいてロギングのイベントを含めるか除外するように構成できます。
 - フィルタルールには、イベントロギングの既存の機能に加えて、適格なイベントの実行をブロック (中断) する機能があります。
 - 複数のフィルタを定義でき、任意の数のユーザーアカウントに任意のフィルタを割り当てることができます。
 - フィルタが明示的に割り当てられていないユーザーアカウントで使用するデフォルトフィルタを定義できます。
- 監査フィルタは、ユーザー定義関数 (UDF) に基づく SQL インタフェースを使用して定義、表示および変更できます。
- 監査フィルタ定義は、mysql システムデータベースのテーブルに格納されます。
- 特定のセッション内で、読取り専用 `audit_log_filter_id` システム変数の値は、フィルタがセッションに割り当てられているかどうかを示します。

注記

デフォルトでは、ルールベースの監査ログのフィルタリングでは、ユーザーの監査可能なイベントは記録されません。すべてのユーザーのすべての監査可能イベントをログに記録するには、次のステートメントを使用します。これにより、ロギングを有効にしてデフォルトアカウントに割り当てた単純なフィルタが作成されます:

```
SELECT audit_log_filter_set_filter('log_all', '{ "filter": { "log": true } }');  
SELECT audit_log_filter_set_user('%', 'log_all');
```

% に割り当てられたフィルタは、フィルタが明示的に割り当てられていないアカウントからの接続に使用されます (最初はすべてのアカウントに適用されます)。

次のリストに、監査フィルタリング制御用の SQL インタフェースを実装する UDF の概要を示します:

- `audit_log_filter_set_filter()`: フィルタの定義
- `audit_log_filter_remove_filter()`: フィルタの削除
- `audit_log_filter_set_user()`: ユーザーアカウントのフィルタリングの開始
- `audit_log_filter_remove_user()`: ユーザーアカウントのフィルタリングの停止

- `audit_log_filter_flush()`: フィルタテーブルへの手動変更をフラッシュして、進行中のフィルタリングに影響を与えます

使用例およびフィルタリング関数の詳細は、[監査ログフィルタリング関数の使用](#) および [監査ログ関数](#) を参照してください。

監査ログのフィルタリング関数には、次の制約があります:

- フィルタリング関数を使用するには、`audit_log` プラグインを有効にする必要があります。有効にしないと、エラーが発生します。また、監査テーブルが存在する必要があります。存在しない場合はエラーが発生します。`audit_log` プラグインとそれに付随する UDF およびテーブルをインストールするには、[セクション6.4.5.2「MySQL Enterprise Audit のインストールまたはアンインストール」](#) を参照してください。
- フィルタリング機能を使用するには、ユーザーが `SUPER` 権限を持っている必要があり、持っていない場合はエラーが発生します。`SUPER` 権限をユーザーアカウントに付与するには、次のステートメントを使用します:

```
GRANT SUPER ON *.* TO user;
```

または、ユーザーに特定のフィルタリング関数へのアクセスを許可しながら `SUPER` 権限を付与しないようにする場合は、「ラッパー」ストアドプログラムを定義できます。この手法については、[汎用キーリング関数の使用](#) での UDF のキー設定のコンテキストで説明されています。UDF のフィルタリングでの使用に適応できます。

- `audit_log` プラグインは、インストールされているが、付随する監査テーブルおよび関数が作成されていない場合、レガシーモードで動作します。プラグインは、サーバーの起動時に次のメッセージをエラーログに書き込みます:

```
[Warning] Plugin audit_log reported: 'Failed to open the audit log filter tables.'  
[Warning] Plugin audit_log reported: 'Audit Log plugin supports a filtering,  
which has not been installed yet. Audit Log plugin will run in the legacy  
mode, which will be disabled in the next release.'
```

レガシーモードでは、フィルタリングはイベントアカウントまたはステータスにのみ実行できます。詳細は、[セクション6.4.5.9「レガシーモード監査ログのフィルタリング」](#) を参照してください。

- [監査ログフィルタリング関数の使用](#)

監査ログフィルタリング関数の使用

監査ログのユーザー定義関数 (UDF) を使用する前に、[セクション6.4.5.2「MySQL Enterprise Audit のインストールまたはアンインストール」](#) の指示に従ってインストールします。これらの関数を使用するには、`SUPER` 権限が必要です。

監査ログのフィルタリング機能を使用すると、フィルタ定義を作成、変更および削除し、ユーザーアカウントにフィルタを割り当てるためのインタフェースを提供することで、フィルタリングを制御できます。

フィルタ定義は JSON 値です。MySQL での JSON データの使用の詳細は、[セクション11.5「JSON データ型」](#) を参照してください。このセクションでは、単純なフィルタ定義をいくつか示します。フィルタ定義の詳細は、[セクション6.4.5.8「監査ログフィルタ定義の書込み」](#) を参照してください。

接続が到着すると、監査ログプラグインは、現在のフィルタ割り当てでユーザーアカウント名を検索して、新しいセッションに使用するフィルタを決定します:

- フィルタがユーザーに割り当てられている場合、監査ログはそのフィルタを使用します。
- それ以外の場合、ユーザー固有のフィルタ割り当てが存在せず、デフォルトアカウント (%) にフィルタが割り当てられていると、監査ログではデフォルトフィルタが使用されます。
- それ以外の場合、監査ログはセッションから処理する監査イベントを選択しません。

セッション中に変更ユーザー操作が発生した場合 (`mysql_change_user()` を参照)、セッションのフィルタ割り当ては同じルールを使用して更新されますが、新規ユーザーの場合です。

デフォルトでは、アカウントにフィルタが割り当てられていないため、どのアカウントに対しても監査可能なイベントの処理は行われません。

かわりに、デフォルトで接続関連のアクティビティのみをログに記録するとします (たとえば、接続、変更ユーザーおよび切断イベントは表示しますが、接続中にユーザーが実行する SQL ステートメントは表示しません)。これを実現

するには、`connection` クラスのイベントのみのロギングを有効にするフィルタ (ここでは `log_conn_events` という名前) を定義し、そのフィルタを % アカウント名で表されるデフォルトアカウントに割り当てます:

```
SET @f = '{ "filter": { "class": { "name": "connection" } } }';
SELECT audit_log_filter_set_filter('log_conn_events', @f);
SELECT audit_log_filter_set_user('%', 'log_conn_events');
```

監査ログでは、フィルタが明示的に定義されていないアカウントからの接続に、このデフォルトのアカウントフィルタが使用されるようになりました。

フィルタを特定のユーザーアカウントに明示的に割り当てるには、フィルタを定義してから、関連するアカウントに割り当てます:

```
SELECT audit_log_filter_set_filter('log_all', '{ "log": true }');
SELECT audit_log_filter_set_user('user1@localhost', 'log_all');
SELECT audit_log_filter_set_user('user2@localhost', 'log_all');
```

これで、`user1@localhost` および `user2@localhost` の完全ロギングが有効になりました。他のアカウントからの接続は、デフォルトのアカウントフィルタを使用して引き続きフィルタされます。

ユーザーアカウントと現在のフィルタの関連付けを解除するには、フィルタの割当てを解除するか、別のフィルタを割り当てます:

- ユーザーアカウントからフィルタの割当てを解除するには:

```
SELECT audit_log_filter_remove_user('user1@localhost');
```

アカウントの現在のセッションのフィルタリングは影響を受けません。アカウントからの後続の接続は、デフォルトのアカウントフィルタ (存在する場合) を使用してフィルタされ、それ以外の場合はログに記録されません。

- ユーザーアカウントに別のフィルタを割り当てるには:

```
SELECT audit_log_filter_set_filter('log_nothing', '{ "log": false }');
SELECT audit_log_filter_set_user('user1@localhost', 'log_nothing');
```

アカウントの現在のセッションのフィルタリングは影響を受けません。アカウントからの後続の接続は、新しいフィルタを使用してフィルタ処理されます。ここに示すフィルタの場合、`user1@localhost` からの新規接続のロギングがないことを意味します。

監査ログのフィルタリングでは、ユーザー名とホスト名の比較で大/小文字が区別されます。これは、ホスト名の比較で大/小文字が区別されない権限チェックの比較とは異なります。

フィルタを削除するには、次のようにします:

```
SELECT audit_log_filter_remove_filter('log_nothing');
```

フィルタを削除すると、そのフィルタが割り当てられているユーザー (それらのユーザーの現在のセッションを含む) からもフィルタの割当てが解除されます。

ここで説明したフィルタ UDF は、監査フィルタリングにすぐに影響し、フィルタおよびユーザーアカウントを格納する `mysql` システムデータベースの監査ログテーブルを更新します (監査ログテーブルを参照)。`INSERT`、`UPDATE`、`DELETE` などのステートメントを使用して監査ログテーブルを直接変更することもできますが、このような変更はフィルタリングにすぐには影響しません。変更をフラッシュして操作可能にするには、`audit_log_filter_flush()` をコールします:

```
SELECT audit_log_filter_flush();
```

警告

`audit_log_filter_flush()` は、すべてのフィルタを強制的にリロードするために、監査テーブルを直接変更した後にのみ使用してください。それ以外の場合は、この関数を使用しないでください。実際には、`UNINSTALL PLUGIN` と `INSTALL PLUGIN` を使用した `audit_log` プラグインのアンロードおよびリロードの簡略化されたバージョンです。

`audit_log_filter_flush()` は、現在のすべてのセッションに影響を与え、以前のフィルタからデータタッチします。現在のセッションは、切断して再接続するか、ユーザー変更操作を実行しないかぎり、ログに記録されなくなります。

フィルタが現在のセッションに割り当てられているかどうかを確認するには、読取り専用 `audit_log_filter_id` システム変数のセッション値を確認します。値が 0 の場合、フィルタは割り当てられません。ゼロ以外の値は、割り当てられたフィルタの内部的に保持されている ID を示します:

```
mysql> SELECT @@audit_log_filter_id;
+-----+
| @@audit_log_filter_id |
+-----+
|          2 |
+-----+
```

6.4.5.8 監査ログフィルタ定義の書込み

フィルタ定義は JSON 値です。MySQL での JSON データの使用の詳細は、[セクション11.5「JSON データ型」](#) を参照してください。

フィルタ定義の形式は次のとおりです。ここで、`actions` はフィルタリングの実行方法を示します:

```
{ "filter": actions }
```

次の説明では、フィルタ定義で許可される構成要素について説明します。

- [すべてのイベントのロギング](#)
- [ロギング固有のイベントクラス](#)
- [特定のイベントサブクラスのロギング](#)
- [包括的および排他的ロギング](#)
- [イベントフィールド値のテスト](#)
- [特定のイベントの実行のブロック](#)
- [論理演算子](#)
- [事前定義変数の参照](#)
- [事前定義関数の参照](#)
- [ユーザーフィルタの置換](#)

すべてのイベントのロギング

すべてのイベントのロギングを明示的に有効または無効にするには、フィルタで `log` 要素を使用します:

```
{
  "filter": { "log": true }
}
```

`log` の値は、`true` または `false` のいずれかです。

前述のフィルタは、すべてのイベントのロギングを有効にします。これは次と同等です:

```
{
  "filter": {}
}
```

ロギングの動作は、`log` の値、および `class` アイテムと `event` アイテムのどちらが指定されているかによって異なります:

- `log` が指定されている場合は、指定された値が使用されます。
- `log` が指定されていない場合、`class` または `event` アイテムが指定されていないとロギングは `true` になり、それ以外の場合は `false` になります (この場合、`class` または `event` に独自の `log` アイテムを含めることができます)。

ロギング固有のイベントクラス

特定のクラスのイベントをログに記録するには、ログに記録するクラスの名前を示す `name` フィールドとともに、フィルタで `class` 要素を使用します:

```
{
  "filter": {
    "class": { "name": "connection" }
  }
}
```

`name` の値は、それぞれ `connection`、`general` または `table_access` で、接続、一般またはテーブルアクセスイベントをログに記録できます。

前述のフィルタは、`connection` クラスでのイベントのロギングを有効にします。これは、明示的に作成された `log` アイテムの次のフィルタと同等です:

```
{
  "filter": {
    "log": false,
    "class": { "log": true,
              "name": "connection" }
  }
}
```

複数のクラスのロギングを有効にするには、クラスを指定する `JSON` 配列要素として `class` 値を定義します:

```
{
  "filter": {
    "class": [
      { "name": "connection" },
      { "name": "general" },
      { "name": "table_access" }
    ]
  }
}
```

注記

特定のアイテムの複数のインスタンスがフィルタ定義内の同じレベルに表示される場合、アイテム値を配列値内のそのアイテムの単一のインスタンスに結合できます。前述の定義は、次のように記述できます:

```
{
  "filter": {
    "class": [
      { "name": [ "connection", "general", "table_access" ] }
    ]
  }
}
```

特定のイベントサブクラスのロギング

特定のイベントサブクラスを選択するには、サブクラスを指定する `name` アイテムを含む `event` アイテムを使用します。`event` アイテムによって選択されたイベントのデフォルトのアクションは、それらをログに記録することです。たとえば、このフィルタは、指定されたイベントサブクラスのロギングを有効にします:

```
{
  "filter": {
    "class": [
      {
        "name": "connection",
        "event": [
          { "name": "connect" },
          { "name": "disconnect" }
        ]
      },
      { "name": "general" },
      {
        "name": "table_access",
        "event": [

```

```

    {"name": "insert"},
    {"name": "delete"},
    {"name": "update"}
  ]
}
]
}
}

```

event 品目には、適格なイベントをログに記録するかどうかを示す明示的な **log** 品目を含めることもできます。この **event** アイテムは、複数のイベントを選択し、それらのロギング動作を明示的に示します:

```

"event": [
  {"name": "read", "log": false },
  {"name": "insert", "log": true },
  {"name": "delete", "log": true },
  {"name": "update", "log": true }
]

```

event 項目に **abort** 項目が含まれている場合は、該当するイベントをブロックするかどうかも指定できます。詳細は、[特定のイベントの実行のブロック](#)を参照してください。

表6.29「イベントクラスとサブクラスの組合せ」は、各イベントクラスに許可されるサブクラス値を記述します。

表 6.29 イベントクラスとサブクラスの組合せ

イベントクラス	イベントサブクラス	説明
接続	connect	接続の開始 (成功または失敗)
接続	change_user	セッション中に異なるユーザー/パスワードでユーザーを再認証
接続	disconnect	接続の終了
general	status	一般的な操作情報
message	internal	内部で生成されたメッセージ
message	user	<code>audit_api_message_emit_udf()</code> によって生成されたメッセージ
table_access	read	<code>SELECT</code> や <code>INSERT INTO ... SELECT</code> などのテーブル読取りステートメント
table_access	delete	<code>DELETE</code> や <code>TRUNCATE TABLE</code> などのテーブル削除ステートメント
table_access	挿入	<code>INSERT</code> や <code>REPLACE</code> などのテーブル挿入ステートメント
table_access	update	<code>UPDATE</code> などのテーブル更新ステートメント

表6.30「イベントクラスとサブクラスの組合せごとのログおよび中断特性」では、イベントサブクラスごとに、ログに記録できるか中断できるかが記述されます。

表 6.30 イベントクラスとサブクラスの組合せごとのログおよび中断特性

イベントクラス	イベントサブクラス	記録可能	中断可能
接続	connect	はい	いいえ
接続	change_user	はい	いいえ
接続	disconnect	はい	いいえ
general	status	はい	いいえ
message	internal	はい	はい
message	user	はい	はい
table_access	read	はい	はい

イベントクラス	イベントサブクラス	記録可能	中断可能
table_access	delete	はい	はい
table_access	挿入	はい	はい
table_access	update	はい	はい

包括的および排他的ロギング

フィルタは包含モードまたは排他モードで定義できます:

- 包含モードでは、明示的に指定された項目のみが記録されます。
- 排他モードでは、明示的に指定された項目以外はすべてログに記録されます。

包括的ロギングを実行するには、ロギングをグローバルに無効にし、特定のクラスのロギングを有効にします。このフィルタは、[connect](#) および [disconnect](#) イベントを [connection](#) クラスに記録し、イベントを [general](#) クラスに記録します:

```
{
  "filter": {
    "log": false,
    "class": [
      {
        "name": "connection",
        "event": [
          { "name": "connect", "log": true },
          { "name": "disconnect", "log": true }
        ]
      }
    ],
    { "name": "general", "log": true }
  }
}
```

排他的ロギングを実行するには、ロギングをグローバルに有効にし、特定のクラスのロギングを無効にします。このフィルタは、[general](#) クラスのイベントを除くすべてのログを記録します:

```
{
  "filter": {
    "log": true,
    "class": [
      { "name": "general", "log": false }
    ]
  }
}
```

このフィルタは、他のすべてをログに記録しないことによって、[change_user](#) イベントを [connection](#) クラス、[message](#) イベントおよび [table_access](#) イベントに記録します:

```
{
  "filter": {
    "log": true,
    "class": [
      {
        "name": "connection",
        "event": [
          { "name": "connect", "log": false },
          { "name": "disconnect", "log": false }
        ]
      },
      { "name": "general", "log": false }
    ]
  }
}
```

イベントフィールド値のテスト

特定のイベントフィールド値に基づいてロギングを有効にするには、フィールド名と予想される値を示す [field](#) アイテムを [log](#) アイテム内で指定します:

```
{
  "filter": {
    "class": {
      "name": "general",
      "event": {
        "name": "status",
        "log": {
          "field": { "name": "general_command.str", "value": "Query" }
        }
      }
    }
  }
}
```

各イベントには、カスタムフィルタリングを実行するためにフィルタ内からアクセスできるイベントクラス固有のフィールドが含まれます。

接続イベントは、ユーザーがサーバーに接続したり、サーバーから切断したりするなど、セッション中に接続関連のアクティビティが発生したときを示します。表6.31「[接続イベントフィールド](#)」は、接続イベントに許可されているフィールドを示します。

表 6.31 接続イベントフィールド

フィールド名	フィールドタイプ	説明
status	整数	イベントステータス: 0: OK それ以外の場合: 失敗
connection_id	unsigned integer	接続 ID
user.str	文字列	認証時に指定されたユーザー名
user.length	unsigned integer	ユーザー名の長さ
priv_user.str	文字列	認証されたユーザー名 (アカウントユーザー名)
priv_user.length	unsigned integer	認証済ユーザー名の長さ
external_user.str	文字列	外部ユーザー名 (サードパーティの認証プラグインによって提供される)
external_user.length	unsigned integer	外部ユーザー名の長さ
proxy_user.str	文字列	プロキシユーザー名
proxy_user.length	unsigned integer	プロキシユーザー名の長さ
host.str	文字列	接続ユーザーホスト
host.length	unsigned integer	接続ユーザーホストの長さ
ip.str	文字列	接続ユーザーの IP アドレス
ip.length	unsigned integer	接続ユーザーの IP アドレスの長さ
database.str	文字列	接続時に指定されたデータベース名
database.length	unsigned integer	データベース名長
connection_type	整数	接続タイプ: または <code>::undefined</code> : 未定義 または <code>::tcp/ip</code> : TCP/IP または <code>::socket</code> : Socket または <code>::named_pipe</code> : 名前付きパイプ

フィールド名	フィールドタイプ	説明
		または "::ssl" : 暗号化を使用した TCP/IP または "::shared_memory" : 共有メモリー

"::xxx"値は、リテラルの数値のかわりに指定できるシンボリック擬似定数です。これらは文字列として引用符で囲む必要があり、大/小文字が区別されます。

一般イベントは、操作のステータスコードとその詳細を示します。表6.32「一般イベントフィールド」は、一般イベントに許可されているフィールドを示します。

表 6.32 一般イベントフィールド

フィールド名	フィールドタイプ	説明
general_error_code	整数	イベントステータス: 0: OK それ以外の場合: 失敗
general_thread_id	unsigned integer	接続/スレッド ID
general_user.str	文字列	認証時に指定されたユーザー名
general_user.length	unsigned integer	ユーザー名の長さ
general_command.str	文字列	コマンド名
general_command.length	unsigned integer	コマンド名長
general_query.str	文字列	SQL ステートメントテキスト
general_query.length	unsigned integer	SQL ステートメントのテキスト長
general_host.str	文字列	ホスト名
general_host.length	unsigned integer	ホスト名長
general_sql_command.str	文字列	SQL コマンドタイプ名
general_sql_command.length	unsigned integer	SQL コマンドタイプ名の長さ
general_external_user.str	文字列	外部ユーザー名 (サードパーティの認証プラグインによって提供される)
general_external_user.length	unsigned integer	外部ユーザー名の長さ
general_ip.str	文字列	接続ユーザーの IP アドレス
general_ip.length	unsigned integer	接続ユーザー IP アドレスの長さ

[general_command.str](#) はコマンド名を示します: [Query](#), [Execute](#), [Quit](#) または [Change user](#)。

[general_command.str](#) フィールドが [Query](#) または [Execute](#) に設定された一般イベントには、SQL コマンドのタイプを指定する値に設定された [general_sql_command.str](#) が含まれます: [alter_db](#), [alter_db_upgrade](#), [admin_commands](#) など。これらの値は、次のステートメントによって表示されるパフォーマンススキーマインストゥルメントの最後のコンポーネントとして表示できます:

```
mysql> SELECT NAME FROM performance_schema.setup_instruments
WHERE NAME LIKE 'statement/sql/%' ORDER BY NAME;
+-----+
| NAME                                     |
+-----+
| statement/sql/alter_db                   |
| statement/sql/alter_db_upgrade           |
| statement/sql/alter_event                 |
| statement/sql/alter_function             |
| statement/sql/alter_instance             |
| statement/sql/alter_procedure            |
| statement/sql/alter_server                |
```

...

テーブルアクセスイベントは、特定のテーブルアクセスに関する情報を提供します。表6.33「テーブルアクセスイベントフィールド」は、テーブルアクセスイベントに許可されているフィールドを示します。

表 6.33 テーブルアクセスイベントフィールド

フィールド名	フィールドタイプ	説明
connection_id	unsigned integer	イベント接続 ID
sql_command_id	整数	SQL コマンド ID
query.str	文字列	SQL ステートメントテキスト
query.length	unsigned integer	SQL ステートメントのテキスト長
table_database.str	文字列	イベントに関連付けられたデータベース名
table_database.length	unsigned integer	データベース名長
table_name.str	文字列	イベントに関連付けられたテーブル名
table_name.length	unsigned integer	テーブル名の長さ

次のリストに、どのステートメントがどのテーブルアクセスイベントを生成するかを示します：

- read イベント：
 - SELECT
 - INSERT ... SELECT (SELECT 句で参照されるテーブルの場合)
 - REPLACE ... SELECT (SELECT 句で参照されるテーブルの場合)
 - UPDATE ... WHERE (WHERE 句で参照されるテーブルの場合)
 - HANDLER ... READ
- delete イベント：
 - DELETE
 - TRUNCATE TABLE
- insert イベント：
 - INSERT
 - INSERT ... SELECT (INSERT 句で参照されるテーブルの場合)
 - REPLACE
 - REPLACE ... SELECT (REPLACE 句で参照されるテーブルの場合)
 - LOAD DATA
 - LOAD XML
- update イベント：
 - UPDATE
 - UPDATE ... WHERE (UPDATE 句で参照されるテーブルの場合)

特定のイベントの実行のブロック

event 品目には、適格イベントが実行されないようにするかどうかを示す abort 品目を含めることができます。たとえば、abort では、特定の SQL ステートメントの実行をブロックするルールを記述できます。

abort 品目は、**event** 品目内に表示される必要があります。例:

```
"event": {
  "name": qualifying event subclass names
  "abort": condition
}
```

name アイテムによって選択されたイベントサブクラスの場合、**abort** アクションは **condition** 評価に応じて true または false です。条件が true と評価された場合、イベントはブロックされます。それ以外の場合、イベントは引き続き実行されます。

condition 仕様は、**true** または **false** のように単純にすることも、イベント特性に依存するようにより複雑にすることもできます。

このフィルタは、**INSERT**、**UPDATE** および **DELETE** ステートメントをブロックします:

```
{
  "filter": {
    "class": {
      "name": "table_access",
      "event": {
        "name": ["insert", "update", "delete"],
        "abort": true
      }
    }
  }
}
```

この複雑なフィルタは、特定のテーブル (**finances.bank_account**) に対してのみ同じステートメントをブロックします:

```
{
  "filter": {
    "class": {
      "name": "table_access",
      "event": {
        "name": ["insert", "update", "delete"],
        "abort": {
          "and": [
            { "field": { "name": "table_database.str", "value": "finances" } },
            { "field": { "name": "table_name.str", "value": "bank_account" } }
          ]
        }
      }
    }
  }
}
```

フィルタによって一致およびブロックされたステートメントは、クライアントにエラーを返します:

```
ERROR 1045 (28000): Statement was aborted by an audit log filter
```

すべてのイベントをブロックできるわけではありません (表6.30「イベントクラスとサブクラスの組合せごとのログおよび中断特性」を参照)。できないイベントの場合、監査ログは警告をブロックするのではなく、エラーログに書き込みます。

abort アイテムが **event** アイテム以外の場所に表示されるフィルタを定義しようとすると、エラーが発生します。

論理演算子

論理演算子 (**and**, **or**, **not**) は、**log** アイテムで使用できます。これにより、より高度なフィルタリング構成を構築できます:

```
{
  "filter": {
    "class": {
      "name": "general",
      "event": {
        "name": "status",
        "log": {
          "or": [
            {
              "and": [
```

```

        { "field": { "name": "general_command.str", "value": "Query" } },
        { "field": { "name": "general_command.length", "value": 5 } }
      ]
    },
    {
      "and": [
        { "field": { "name": "general_command.str", "value": "Execute" } },
        { "field": { "name": "general_command.length", "value": 7 } }
      ]
    }
  ]
}
}
}
}
}
}

```

事前定義変数の参照

log 条件で事前定義済の変数を参照するには、特定の値と等しいかどうかをテストする `variable` アイテムを使用します:

```

{
  "filter": {
    "class": {
      "name": "general",
      "event": {
        "name": "status",
        "log": {
          "variable": {
            "name": "audit_log_connection_policy_value", "value": "::none"
          }
        }
      }
    }
  }
}
}

```

事前定義された各変数は、システム変数に対応します。事前定義された変数をテストするフィルタを記述することで、フィルタを再定義しなくても、対応するシステム変数を設定してフィルタ操作を変更できます。たとえば、`audit_log_connection_policy_value` の事前定義済変数の値をテストするフィルタを記述することで、`audit_log_connection_policy` システム変数の値を変更することでフィルタ操作を変更できます。

`audit_log_xxx_policy` システム変数は、レガシーモードの監査ログに使用されます ([セクション6.4.5.9「レガシーモード監査ログのフィルタリング」](#)を参照)。ルールベースの監査ログフィルタリングでは、これらの変数は表示されたままになりますが (たとえば、`SHOW VARIABLES` を使用して)、変数を参照する構成を含むフィルタを記述しないかぎり、変更は効果がありません。

次のリストでは、`variable` 品目に許可されている事前定義済変数について説明します:

- `audit_log_connection_policy_value`

この変数は、`audit_log_connection_policy` システム変数の値に対応します。値は符号なし整数です。表 6.34「`audit_log_connection_policy_value` の値」には、許可された値および対応する `audit_log_connection_policy` 値が表示されます。

表 6.34 `audit_log_connection_policy_value` の値

値	対応する <code>audit_log_connection_policy</code> 値
0 または "::none"	NONE
1 または "::errors"	ERRORS
2 または "::all"	ALL

"::xxx"値は、リテラルの数値のかわりに指定できるシンボリック擬似定数です。これらは文字列として引用符で囲む必要があり、大/小文字が区別されます。

- `audit_log_policy_value`

この変数は、`audit_log_policy` システム変数の値に対応します。値は符号なし整数です。表 6.35 「`audit_log_policy_value` の値」には、許可された値および対応する `audit_log_policy` 値が表示されます。

表 6.35 `audit_log_policy_value` の値

値	対応する <code>audit_log_policy</code> 値
0 または "::none"	NONE
1 または "::logins"	LOGINS
2 または "::all"	ALL
3 または "::queries"	QUERIES

"::xxx"値は、リテラルの数値のかわりに指定できるシンボリック擬似定数です。これらは文字列として引用符で囲む必要があり、大/小文字が区別されます。

- `audit_log_statement_policy_value`

この変数は、`audit_log_statement_policy` システム変数の値に対応します。値は符号なし整数です。表 6.36 「`audit_log_statement_policy_value` の値」には、許可された値および対応する `audit_log_statement_policy` 値が表示されます。

表 6.36 `audit_log_statement_policy_value` の値

値	対応する <code>audit_log_statement_policy</code> 値
0 または "::none"	NONE
1 または "::errors"	ERRORS
2 または "::all"	ALL

"::xxx"値は、リテラルの数値のかわりに指定できるシンボリック擬似定数です。これらは文字列として引用符で囲む必要があり、大/小文字が区別されます。

事前定義関数の参照

`log` 条件で事前定義関数を参照するには、`function` アイテムを使用します。このアイテムは、`name` および `args` の値を使用して、関数名とその引数をそれぞれ指定します:

```
{
  "filter": {
    "class": {
      "name": "general",
      "event": {
        "name": "status",
        "log": {
          "function": {
            "name": "find_in_include_list",
            "args": [ { "string": [ { "field": "user.str" },
                               { "string": "@" },
                               { "field": "host.str" } ] } ]
          }
        }
      }
    }
  }
}
```

`name` 項目に指定されている関数は、カッコや引数リストを含まない関数名のみである必要があります。`args` アイテムに引数がある場合は、関数の説明にリストされている順序で引数を指定する必要があります。引数は、事前定義済の変数、イベントフィールド、文字列定数または数値定数を参照できます。

前述のフィルタは、現在のユーザーが `audit_log_include_accounts` システム変数で見つかったかどうかに応じて、`general` クラスの `status` イベントをログに記録するかどうかを決定します。そのユーザーは、イベントのフィールドを使用して作成されます。

次のリストでは、`function` 品目に許可されている事前定義済関数について説明します:

- `audit_log_exclude_accounts_is_null()`

`audit_log_exclude_accounts` システム変数が `NULL` かどうかを確認します。この関数は、レガシー監査ログの実装に対応するフィルタを定義する場合に役立ちます。

引数:

なし

- `audit_log_include_accounts_is_null()`

`audit_log_include_accounts` システム変数が `NULL` かどうかを確認します。この関数は、レガシー監査ログの実装に対応するフィルタを定義する場合に役立ちます。

引数:

なし

- `debug_sleep(millisecond)`

指定されたミリ秒数スリープします。この関数は、パフォーマンス測定時に使用されます。

`debug_sleep()` はデバッグビルドにのみ使用できます。

引数:

- `millisecond`: スリープするミリ秒数を指定する符号なし整数。

- `find_in_exclude_list(account)`

アカウント文字列が監査ログ除外リスト (`audit_log_exclude_accounts` システム変数の値) に存在するかどうかを確認します。

引数:

- `account`: ユーザーアカウント名を指定する文字列。

- `find_in_include_list(account)`

アカウント文字列が監査ログインクルードリスト (`audit_log_include_accounts` システム変数の値) に存在するかどうかを確認します。

引数:

- `account`: ユーザーアカウント名を指定する文字列。

- `string_find(text, substr)`

`substr` 値が `text` 値に含まれているかどうかを確認します。この検索では大文字と小文字が区別されます。

引数:

- `text`: 検索するテキスト文字列。
- `substr`: `text` で検索する部分文字列。

ユーザーフィルタの置換

場合によっては、フィルタ定義を動的に変更できます。これを行うには、既存の `filter` 内で `filter` 構成を定義します。
例:

```
{  
  "filter": {
```



```
"id": "main",
"class": {
  "name": "table_access",
  "event": {
    "name": [ "update", "delete" ],
    "log": false,
    "filter": {
      "class": {
        "name": "general",
        "event": { "name": "status",
                  "filter": { "ref": "main" } }
      }
    },
    "activate": {
      "or": [
        { "field": { "name": "table_name.str", "value": "temp_1" } },
        { "field": { "name": "table_name.str", "value": "temp_2" } }
      ]
    }
  }
}
}
```

サブフィルタ内の `activate` 要素が `true` と評価されると、新しいフィルタがアクティブ化されます。最上位の `filter` で `activate` の使用は許可されていません。

サブフィルタ内の `ref` アイテムを使用して元のフィルタ `id` を参照することで、新しいフィルタを元のフィルタに置き換えることができます。

表示されるフィルタは次のように動作します:

- `main` フィルタは、`update` または `delete` のいずれかの `table_access` イベントを待機します。
- `temp_1` または `temp_2` テーブルで `update` または `delete` の `table_access` イベントが発生した場合、フィルタは内部イベントに置き換えられます (明示的に参照する必要がないため、`id` はありません)。
- コマンドの終了が通知されると (`general / status` イベント)、エントリが監査ログファイルに書き込まれ、フィルタが `main` フィルタに置き換えられます。

このフィルタは、次のような `temp_1` または `temp_2` テーブルに対して更新または削除を行うステートメントをログに記録する場合に役立ちます:

```
UPDATE temp_1, temp_3 SET temp_1.a=21, temp_3.a=23;
```

このステートメントでは複数の `table_access` イベントが生成されますが、監査ログファイルには `general / status` エントリのみが含まれます。

注記

定義で使用されている `id` 値は、その定義に対してのみ評価されます。これらは、`audit_log_filter_id` システム変数の値とは関係ありません。

6.4.5.9 レガシーモード監査ログのフィルタリング

注記

このセクションでは、レガシー監査ログのフィルタリングについて説明します。このフィルタリングは、`audit_log` プラグインがインストールされているが、ルールベースのフィルタリングに必要な監査テーブルおよび UDF には付随しない場合に適用されます。

監査ログプラグインは、監査対象イベントをフィルタリングできます。これにより、イベントの発生元またはイベントステータスに基づいて、監査イベントを監査ログファイルに書き込むかどうかを制御できます。ステータスのフィルタリングは、接続イベントおよびステートメントイベントごとに個別に発生します。

- [アカウント別のイベントフィルタリング](#)

- ステータス別のイベントフィルタリング

アカウント別のイベントフィルタリング

元のアカウントに基づいて監査イベントをフィルタするには、サーバーの起動時または実行時に次のいずれかのシステム変数を設定します:

- `audit_log_include_accounts`: 監査ロギングに含めるアカウント。この変数が設定されている場合は、これらのアカウントのみが監査されます。
- `audit_log_exclude_accounts`: 監査ロギングから除外するアカウント。この変数が設定されている場合は、これらのアカウント以外がすべて監査されます。

いずれかの変数の値には、`NULL` またはカンマで区切った 1 つ以上のアカウント名を含む文字列を指定できます。それぞれの形式は `user_name@host_name` です。デフォルトでは、両方の変数が `NULL` になっています。この場合、アカウントのフィルタリングは実行されず、すべてのアカウントで監査が発生します。

`audit_log_include_accounts` または `audit_log_exclude_accounts` の変更は、変更後に作成された接続にのみ影響し、既存の接続には影響しません。

例: `user1` および `user2` ローカルホストのアカウントでのみ監査ロギングを有効にするには、次のように `audit_log_include_accounts` システム変数を設定します。

```
SET GLOBAL audit_log_include_accounts = 'user1@localhost,user2@localhost';
```

同時に `NULL` 以外に設定できるのは、`audit_log_include_accounts` と `audit_log_exclude_accounts` のいずれかのみです。

- `audit_log_include_accounts` を設定すると、サーバーは `audit_log_exclude_accounts` を `NULL` に設定します。
- `audit_log_include_accounts` が `NULL` でない場合を除いて、`audit_log_exclude_accounts` を設定しようとするエラーが発生します。この場合は、まず `audit_log_include_accounts` を `NULL` に設定することでクリアする必要があります。

```
-- This sets audit_log_exclude_accounts to NULL
SET GLOBAL audit_log_include_accounts = value;

-- This fails because audit_log_include_accounts is not NULL
SET GLOBAL audit_log_exclude_accounts = value;

-- To set audit_log_exclude_accounts, first set
-- audit_log_include_accounts to NULL
SET GLOBAL audit_log_include_accounts = NULL;
SET GLOBAL audit_log_exclude_accounts = value;
```

いずれかの変数の値を調査する場合は、`SHOW VARIABLES` で `NULL` が空の文字列として表示されることに注意してください。これを回避するには、代わりに `SELECT` を使用してください。

```
mysql> SHOW VARIABLES LIKE 'audit_log_include_accounts';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| audit_log_include_accounts |      |
+-----+-----+
mysql> SELECT @@audit_log_include_accounts;
+-----+
| @@audit_log_include_accounts |
+-----+
| NULL |
+-----+
```

カンマ、スペース、またはその他の特殊文字が含まれているために、ユーザー名やホスト名を引用符で囲む必要がある場合は、一重引用符を使用して囲みます。変数の値自体が一重引用符で囲まれている場合は、内側の各一重引用符を二重に入力するか、バックスラッシュを使用してエスケープします。次のステートメントはそれぞれ、ローカルの `root` アカウントの監査ロギングを有効にします。引用符のスタイルが異なりますが、いずれも同等です。

```
SET GLOBAL audit_log_include_accounts = 'root@localhost';
```

```
SET GLOBAL audit_log_include_accounts = "root'@'localhost";
SET GLOBAL audit_log_include_accounts = '\root'@\localhost";
SET GLOBAL audit_log_include_accounts = "root'@'localhost";
```

ANSI_QUOTES SQL モードが有効な場合、最後のステートメントは機能しません。このモードでは、二重引用符は文字列引用符ではなく識別子引用符を示しているためです。

ステータス別のイベントフィルタリング

ステータスに基づいて監査イベントをフィルタするには、サーバーの起動時または実行時に次のシステム変数を設定します。これらの変数は、レガシー監査ログのフィルタリングにのみ適用されます。JSON 監査ログのフィルタリングについては、様々なステータス変数が適用されます。[監査ログのオプションおよび変数](#) を参照してください。

- **audit_log_connection_policy**: 接続イベントのロギングポリシーです
- **audit_log_statement_policy**: ステートメントイベントのロギングポリシーです

各変数には、**ALL** (関連付けられたすべてのイベントのログを記録します。これがデフォルトです)、**ERRORS** (失敗したイベントのログのみを記録します)、または **NONE** (イベントのログを記録しません) の値が指定されます。たとえば、ステートメントイベントのログはすべて記録するが、接続イベントのログは失敗したもののみを記録する場合は、次の設定を使用します。

```
SET GLOBAL audit_log_statement_policy = ALL;
SET GLOBAL audit_log_connection_policy = ERRORS;
```

別のポリシーシステム変数 **audit_log_policy** を使用できますが、**audit_log_connection_policy** および **audit_log_statement_policy** ほど制御できません。それはサーバーの起動時にのみ設定できます。実行時は、読み取り専用の変数です。これには、**ALL** (すべてのイベントのログを記録します。これがデフォルトです)、**LOGINS** (接続イベントのログを記録します)、**QUERIES** (ステートメントイベントのログを記録します)、または **NONE** (イベントのログを記録しません) の値が指定されます。これらの値のいずれかを指定しても、監査ログプラグインは成功と失敗を区別せずに、選択したイベントのログをすべて記録します。起動時の **audit_log_policy** の使用は、次のように動作します:

- **audit_log_policy** を設定しない場合や、デフォルト値の **ALL** に設定した場合でも、**audit_log_connection_policy** または **audit_log_statement_policy** を明示的に設定すれば、指定どおりに適用されます。指定しない場合は、デフォルトが **ALL** に設定されます。
- **audit_log_policy** を **ALL** 以外の値に設定した場合は、次の表に示すように、その値が優先され、**audit_log_connection_policy** および **audit_log_statement_policy** を設定する際に使用されます。また、これらの変数のいずれかをデフォルトの **ALL** 以外の値に設定する場合、サーバーはそれらの値がオーバーライドされることを示すメッセージをエラーログに書き込みます。

起動時の audit_log_policy 値	結果として返される audit_log_connection_policy 値	結果として返される audit_log_statement_policy 値
LOGINS	ALL	NONE
QUERIES	NONE	ALL
NONE	NONE	NONE

6.4.5.10 監査ログ参照

次の各セクションでは、MySQL Enterprise Audit 要素のリファレンスを示します:

- [監査ログテーブル](#)
- [監査ログ関数](#)
- [監査ログオプションおよび変数リファレンス](#)
- [監査ログのオプションおよび変数](#)
- [監査ログステータス変数](#)

監査ログのテーブルおよび関数をインストールするには、[セクション6.4.5.2「MySQL Enterprise Audit のインストールまたはアンインストール」](#)で提供されている手順を使用します。これらのオブジェクトがインストールされていないかぎり、`audit_log` プラグインはレガシーモードで動作します。[セクション6.4.5.9「レガシーモード監査ログのフィルタリング」](#)を参照してください。

監査ログテーブル

MySQL Enterprise Audit では、フィルタおよびユーザーアカウントデータの永続的な格納に `mysql` システムデータベースのテーブルが使用されます。テーブルにアクセスできるのは、そのデータベースに対する権限を持つユーザーのみです。これらのテーブルは、`InnoDB` ストレージエンジンを使用します。

これらのテーブルがない場合、`audit_log` プラグインはレガシーモードで動作します。[セクション6.4.5.9「レガシーモード監査ログのフィルタリング」](#)を参照してください。

`audit_log_filter` テーブルには、フィルタ定義が格納されます。テーブルには次のカラムがあります。

- **NAME**

フィルタ名。

- **FILTER**

フィルタ名に関連付けられたフィルタ定義。定義は `JSON` 値として格納されます。

`audit_log_user` テーブルには、ユーザーアカウント情報が格納されます。テーブルには次のカラムがあります。

- **USER**

アカウントのユーザー名部分。アカウント `user1@localhost` の場合、`USER` 部分は `user1` です。

- **HOST**

アカウントのホスト名部分。アカウント `user1@localhost` の場合、`HOST` 部分は `localhost` です。

- **FILTERNAME**

アカウントに割り当てられたフィルタの名前。フィルタ名は、アカウントを `audit_log_filter` テーブルで定義されているフィルタに関連付けます。

監査ログ関数

このセクションでは、監査ログのユーザー定義関数 (UDF) ごとに、その目的、コール順序および戻り値について説明します。これらの UDF を起動できる条件の詳細は、[セクション6.4.5.7「監査ログのフィルタリング」](#)を参照してください。

各監査ログ UDF は、操作が成功したかどうかを示す文字列を返します。`OK` は成功を示します。`ERROR: message` は失敗を示します。

MySQL 8.0.19 の時点では、監査ログ UDF は文字列引数を `utf8mb4` に変換し、文字列戻り値は `utf8mb4` 文字列です。MySQL 8.0.19 より前では、監査ログ UDF は文字列引数をバイナリ文字列として扱い (大文字と小文字を区別しない)、文字列の戻り値はバイナリ文字列です。

次の監査ログ UDF を使用できます:

- `audit_log_encryption_password_get([keyring_id])`

この関数は、監査ログの暗号化パスワードを MySQL キーリングからフェッチします。このキーリングは有効にする必要があり、有効にしないとエラーが発生します。どのキーリングプラグインも使用できます。手順については、[セクション6.4.4「MySQL キーリング」](#)を参照してください。

引数を指定しない場合、関数は現在の暗号化パスワードをバイナリ文字列として取得します。取得する監査ログ暗号化パスワードを指定する引数を指定できます。引数は、現在のパスワードまたはアーカイブされたパスワードのキーリング ID である必要があります。

監査ログの暗号化の詳細は、[監査ログファイルの暗号化](#) を参照してください。

引数:

keyring_id: MySQL 8.0.17 では、このオプションの引数は取得するパスワードのキーリング ID を示します。最大許容長は 766 バイトです。省略すると、現在のパスワードが取得されます。

MySQL 8.0.17 より前では、引数は使用できません。この関数は、常に現在のパスワードを取得します。

戻り値:

成功のパスワード文字列 (最大 766 バイト)、または **NULL** と失敗のエラー。

例:

現在のパスワードを取得します:

```
mysql> SELECT audit_log_encryption_password_get();
+-----+
| audit_log_encryption_password_get() |
+-----+
| secret                               |
+-----+
```

ID でパスワードを取得するには、パフォーマンススキーマ **keyring_keys** テーブルをクエリーすることによって、存在する監査ログ鍵リング ID を確認できます:

```
mysql> SELECT KEY_ID FROM performance_schema.keyring_keys
WHERE KEY_ID LIKE 'audit_log%'
ORDER BY KEY_ID;
+-----+
| KEY_ID                               |
+-----+
| audit_log-20190415T152248-1 |
| audit_log-20190415T153507-1 |
| audit_log-20190416T125122-1 |
| audit_log-20190416T141608-1 |
+-----+
mysql> SELECT audit_log_encryption_password_get('audit_log-20190416T125122-1');
+-----+
| audit_log_encryption_password_get('audit_log-20190416T125122-1') |
+-----+
| segreto                               |
+-----+
```

- [audit_log_encryption_password_set\(password\)](#)

現在の監査ログ暗号化パスワードを引数に設定し、パスワードを MySQL キーリングに格納します。MySQL 8.0.19 では、パスワードは **utf8mb4** 文字列として格納されます。MySQL 8.0.19 より前は、パスワードはバイナリ形式で格納されます。

暗号化が有効な場合、この関数は、現在のログファイルの名前を変更するログファイルローテーション操作を実行し、パスワードで暗号化された新しいログファイルを開始します。キーリングを有効にする必要があり、有効にしないとエラーが発生します。どのキーリングプラグインも使用できます。手順については、[セクション 6.4.4 「MySQL キーリング」](#) を参照してください。

監査ログの暗号化の詳細は、[監査ログファイルの暗号化](#) を参照してください。

引数:

password: パスワード文字列。最大許容長は 766 バイトです。

戻り値:

成功の場合は 1、失敗の場合は 0。

例:

```
mysql> SELECT audit_log_encryption_password_set(password);
+-----+
| audit_log_encryption_password_set(password) |
+-----+
| 1 |
+-----+
```

- `audit_log_filter_flush()`

他のフィルタリング UDF を呼び出すと、操作監査ログのフィルタリングにただちに影響し、監査ログテーブルが更新されます。かわりに、`INSERT`、`UPDATE` および `DELETE` などのステートメントを使用してこれらのテーブルの内容を直接変更しても、変更はフィルタリングにすぐには影響しません。変更をフラッシュして操作可能にするには、`audit_log_filter_flush()` をコールします。

警告

`audit_log_filter_flush()` は、すべてのフィルタを強制的にリロードするために、監査テーブルを直接変更した後にのみ使用してください。それ以外の場合は、この関数を使用しないでください。実際には、`UNINSTALL PLUGIN` と `INSTALL PLUGIN` を使用した `audit_log` プラグインのアンロードおよびリロードの簡略化されたバージョンです。

`audit_log_filter_flush()` は、現在のすべてのセッションに影響を与え、以前のフィルタからデタッチします。現在のセッションは、切断して再接続するか、ユーザー変更操作を実行しないかぎり、ログに記録されなくなります。

この関数が失敗すると、エラーメッセージが返され、次に `audit_log_filter_flush()` が正常にコールされるまで監査ログは無効になります。

引数:

なし

戻り値:

操作が成功したかどうかを示す文字列。 `OK` は成功を示します。 `ERROR: message` は失敗を示します。

例:

```
mysql> SELECT audit_log_filter_flush();
+-----+
| audit_log_filter_flush() |
+-----+
| OK |
+-----+
```

- `audit_log_filter_remove_filter(filter_name)`

フィルタ名を指定すると、現在のフィルタセットからフィルタが削除されます。フィルタが存在しない場合はエラーではありません。

削除されたフィルタがいずれかのユーザーアカウントに割り当てられている場合、それらのユーザーはフィルタ処理を停止します (`audit_log_user` テーブルから削除されます)。フィルタリングの終了には、それらのユーザーの現在のセッションが含まれます: フィルタからデタッチされ、ログに記録されなくなります。

引数:

- `filter_name`: フィルタ名を指定する文字列。

戻り値:

操作が成功したかどうかを示す文字列。 `OK` は成功を示します。 `ERROR: message` は失敗を示します。

例:

```
mysql> SELECT audit_log_filter_remove_filter('SomeFilter');
```



```

+-----+
| audit_log_filter_remove_filter('SomeFilter') |
+-----+
| OK |
+-----+

```

- `audit_log_filter_remove_user(user_name)`

ユーザーアカウント名を指定すると、ユーザーはフィルタに割り当てられなくなります。ユーザーにフィルタが割り当てられていない場合は、エラーになりません。ユーザーの現在のセッションのフィルタリングは影響を受けません。ユーザーの新しい接続は、デフォルトのアカウントフィルタ (存在する場合) を使用してフィルタされ、それ以外の場合はログに記録されません。

名前が % の場合、関数は、明示的にフィルタが割り当てられていないユーザーアカウントに使用されるデフォルトのアカウントフィルタを削除します。

引数:

- **user_name:** `user_name @host_name` 形式の文字列としてのユーザーアカウント名、またはデフォルトアカウントを表す %。

戻り値:

操作が成功したかどうかを示す文字列。OK は成功を示します。ERROR: message は失敗を示します。

例:

```

mysql> SELECT audit_log_filter_remove_user('user1@localhost');
+-----+
| audit_log_filter_remove_user('user1@localhost') |
+-----+
| OK |
+-----+

```

- `audit_log_filter_set_filter(filter_name, definition)`

フィルタ名と定義を指定すると、現在のフィルタセットにフィルタが追加されます。フィルタがすでに存在し、現在のセッションで使用されている場合、それらのセッションはフィルタからデタッチされ、ログに記録されなくなります。これは、新しいフィルタ定義に以前の ID とは異なる新しいフィルタ ID があるために発生します。

引数:

- **filter_name:** フィルタ名を指定する文字列。
- **definition:** フィルタ定義を指定する JSON 値。

戻り値:

操作が成功したかどうかを示す文字列。OK は成功を示します。ERROR: message は失敗を示します。

例:

```

mysql> SET @f = '{ "filter": { "log": false } }';
mysql> SELECT audit_log_filter_set_filter('SomeFilter', @f);
+-----+
| audit_log_filter_set_filter('SomeFilter', @f) |
+-----+
| OK |
+-----+

```

- `audit_log_filter_set_user(user_name, filter_name)`

ユーザーアカウント名とフィルタ名を指定すると、ユーザーにフィルタが割り当てられます。ユーザーには 1 つのフィルタのみを割り当てることができるため、ユーザーにフィルタがすでに割り当てられている場合は、割り当て

が置き換えられます。ユーザーの現在のセッションのフィルタリングは影響を受けません。新しい接続は、新しいフィルタを使用してフィルタ処理されます。

特殊なケースとして、`%` という名前はデフォルトのアカウントを表します。フィルタは、フィルタが明示的に割り当てられていないユーザーアカウントからの接続に使用されます。

引数:

- `user_name`: `user_name @host_name` 形式の文字列としてのユーザーアカウント名、またはデフォルトアカウントを表す `%`。
- `filter_name`: フィルタ名を指定する文字列。

戻り値:

操作が成功したかどうかを示す文字列。 `OK` は成功を示します。 `ERROR: message` は失敗を示します。

例:

```
mysql> SELECT audit_log_filter_set_user('user1@localhost', 'SomeFilter');
+-----+
| audit_log_filter_set_user('user1@localhost', 'SomeFilter') |
+-----+
| OK |
+-----+
```

- `audit_log_read([arg])`

監査ログを読み取り、`JSON` 文字列の結果を返します。監査ログ形式が `JSON` でない場合は、エラーが発生します。

引数または `JSON` ハッシュ引数を指定しない場合、`audit_log_read()` は監査ログからイベントを読み取り、監査イベントの配列を含む `JSON` 文字列を返します。hash 引数の項目は、あとで説明するように、読み取りの発生方法に影響します。返される配列内の各要素は、`JSON` ハッシュとして表されるイベントですが、最後の要素が `JSON null` 値であり、次のイベントを読み取ることができないことを示す例外があります。

`JSON null` 値で構成される引数を使用すると、`audit_log_read()` は現在の読み取り順序をクローズします。

監査ログの読み取りプロセスの詳細は、[セクション6.4.5.6「監査ログファイルの読み取り」](#) を参照してください。

引数:

最後に書き込まれたイベントのブックマークを取得するには、`audit_log_read_bookmark()` をコールします。

`arg`: 引数はオプションです。省略すると、関数は現在の位置からイベントを読み取ります。引数が存在する場合は、読み取り順序をクローズする `JSON null` 値または `JSON` ハッシュを指定できます。ハッシュ引数内では、項目は

オプションであり、読取りを開始する位置や読み取るイベントの数など、読取り操作の側面を制御します。次の項目は重要です (他の項目は無視されます):

- **start**: 最初に読み取るイベントの監査ログ内の位置。位置はタイムスタンプとして指定され、タイムスタンプ値以降に発生する最初のイベントから読取りが開始されます。 **start** アイテムの形式は次のとおりです。ここで、**value** はリテラルのタイムスタンプ値です:

```
"start": { "timestamp": "value" }
```

start 品目は、MySQL 8.0.22 で許可されています。

- **timestamp, id**: 最初に読み取るイベントの監査ログ内の位置。 **timestamp** アイテムと **id** アイテムは、特定のイベントを一意に識別するブックマークで構成されます。 **audit_log_read()** 引数にいずれかの項目が含まれている場合、位置を完全に指定するには両方を含める必要があります。そうしないと、エラーが発生します。
- **max_array_length**: ログから読み取るイベントの最大数。この項目を省略した場合、デフォルトでは、ログの最後まで、または読取りバッファがいっぱいになるまで (いずれか早い方まで) 読み取られます。

開始位置を **audit_log_read()** に指定するには、**start** アイテムまたは **timestamp** アイテムと **id** アイテムで構成されるブックマークを含むハッシュ引数を渡します。ハッシュ引数に **start** アイテムとブックマークの両方が含まれている場合は、エラーが発生します。

ハッシュ引数で開始位置が指定されていない場合、読取りは現在の位置から続行されます。

タイムスタンプ値に時間部分が含まれていない場合は、**00:00:00** の時間部分が想定されます。

戻り値:

コールが成功した場合、戻り値は監査イベントの配列を含む **JSON** 文字列、または読取りシーケンスをクローズする引数として渡された場合は **JSON null** 値です。コールが失敗すると、戻り値は **NULL** になり、エラーが発生します。

例:

```
mysql> SELECT audit_log_read(audit_log_read_bookmark());
+-----+
| audit_log_read(audit_log_read_bookmark()) |
+-----+
| [{"timestamp":"2020-05-18 22:41:24","id":0,"class":"connection", ... } |
+-----+
mysql> SELECT audit_log_read('null');
+-----+
| audit_log_read('null') |
+-----+
| null |
+-----+
```

メモ:

MySQL 8.0.19 より前では、文字列の戻り値はバイナリ **JSON** 文字列です。このような値を非バイナリ文字列に変換する方法については、[セクション6.4.5.6「監査ログファイルの読取り」](#) を参照してください。

- `audit_log_read_bookmark()`

最後に書き込まれた監査ログイベントのブックマークを表す JSON 文字列を返します。監査ログ形式が JSON でない場合は、エラーが発生します。

ブックマークは、監査ログ内のイベントの位置を一意に識別する `timestamp` および `id` アイテムを含む JSON ハッシュです。`audit_log_read()` に渡して、その関数に読取りを開始する位置を示すのに適しています。

監査ログの読取りプロセスの詳細は、[セクション6.4.5.6「監査ログファイルの読取り」](#)を参照してください。

引数:

なし

戻り値:

成功のためのブックマーク、または失敗のための NULL とエラーを含む JSON 文字列。

例:

```
mysql> SELECT audit_log_read_bookmark();
+-----+
| audit_log_read_bookmark() |
+-----+
| {"timestamp": "2019-10-03 21:03:44", "id": 0 } |
+-----+
```

メモ:

MySQL 8.0.19 より前では、文字列の戻り値はバイナリ JSON 文字列です。このような値を非バイナリ文字列に変換する方法については、[セクション6.4.5.6「監査ログファイルの読取り」](#)を参照してください。

監査ログオプションおよび変数リファレンス

表 6.37 「監査ログオプションおよび変数リファレンス」

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
<code>audit-log</code>	はい	はい				
<code>audit_log_buffer_size</code>	はい	はい	はい		グローバル	いいえ
<code>audit_log_compression</code>	はい	はい	はい		グローバル	いいえ
<code>audit_log_connection_policy</code>	はい	はい	はい		グローバル	はい
<code>audit_log_current_session</code>			はい		両方	いいえ
<code>Audit_log_current_size</code>				はい	グローバル	いいえ
<code>audit_log_encrypt</code>	はい	はい	はい		グローバル	いいえ
<code>Audit_log_event_max_drop_size</code>				はい	グローバル	いいえ
<code>Audit_log_events</code>				はい	グローバル	いいえ
<code>Audit_log_events_filtered</code>				はい	グローバル	いいえ
<code>Audit_log_events_lost</code>				はい	グローバル	いいえ
<code>Audit_log_events_written</code>				はい	グローバル	いいえ
<code>audit_log_exclude_accounts</code>	はい	はい	はい		グローバル	はい
<code>audit_log_file</code>	はい	はい	はい		グローバル	いいえ
<code>audit_log_filter_id</code>			はい		両方	いいえ
<code>audit_log_flush</code>			はい		グローバル	はい
<code>audit_log_format</code>	はい	はい	はい		グローバル	いいえ
<code>audit_log_include_accounts</code>	はい	はい	はい		グローバル	はい

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
audit_log_password_history_keep_days	はい	はい	はい		グローバル	はい
audit_log_policy	はい	はい	はい		グローバル	いいえ
audit_log_prune_interval	はい	はい	はい		グローバル	はい
audit_log_read_buffer_size	はい	はい	はい		異なる	異なる
audit_log_rotate_interval_size	はい	はい	はい		グローバル	はい
audit_log_statement_policy	はい	はい	はい		グローバル	はい
audit_log_strategy	はい	はい	はい		グローバル	いいえ
Audit_log_total_size				はい	グローバル	いいえ
Audit_log_write_waits				はい	グローバル	いいえ

監査ログのオプションおよび変数

このセクションでは、MySQL Enterprise Audit の操作を構成するコマンドオプションおよびシステム変数について説明します。起動時に指定された値が正しくない場合、`audit_log` プラグインが正しく初期化されず、サーバーがロードしない可能性があります。この場合、サーバーは他の監査ログ設定を認識しないため、エラーメッセージを生成することもあります。

監査ログプラグインのアクティブ化を構成するには、このオプションを使用します：

- `--audit-log[=value]`

コマンド行形式	<code>--audit-log[=value]</code>
型	列挙
デフォルト値	ON
有効な値	ON OFF FORCE FORCE_PLUS_PERMANENT

このオプションは、サーバーの起動時に `audit_log` プラグインをロードする方法を制御します。プラグインが以前に `INSTALL PLUGIN` に登録されているか、`--plugin-load` または `--plugin-load-add` にロードされている場合にのみ使用できます。セクション6.4.5.2「MySQL Enterprise Audit のインストールまたはアンインストール」を参照してください。

セクション5.6.1「プラグインのインストールおよびアンインストール」で説明したように、オプションの値は、プラグインのロードオプションに指定可能な値のいずれかである必要があります。たとえば、`--audit-log=FORCE_PLUS_PERMANENT` は、プラグインをロードし、それがサーバーの実行時に削除されることを回避するようにサーバーに指示します。

監査ログプラグインが有効になっている場合、ロギングの制御を許可するいくつかのシステム変数が公開されます：

```
mysql> SHOW VARIABLES LIKE 'audit_log%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| audit_log_buffer_size | 1048576 |
| audit_log_connection_policy | ALL |
| audit_log_current_session | OFF |
| audit_log_exclude_accounts | |
| audit_log_file | audit.log |
| audit_log_filter_id | 0 |
| audit_log_flush | OFF |
| audit_log_format | NEW |
| audit_log_include_accounts | |
```

```

audit_log_policy      | ALL      |
audit_log_rotate_on_size | 0      |
audit_log_statement_policy | ALL    |
audit_log_strategy    | ASYNCHRONOUS |
+-----+-----+

```

これらの変数のいずれも、サーバーの起動時 (一部は実行時) に設定できます。レガシーモードの監査ログフィルタリングでのみ使用可能なものは、このように記載されています。

- [audit_log_buffer_size](#)

コマンド行形式	<code>--audit-log-buffer-size=#</code>
システム変数	audit_log_buffer_size
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1048576
最小値	4096
最大値 (64 ビットプラットフォーム)	18446744073709547520
最大値 (32 ビットプラットフォーム)	4294967295

監査ログプラグインが非同期的にイベントをログに書き込むと、イベントの内容を書き込む前に、バッファーを使用してそれらを格納します。この変数は、そのバッファーのサイズ (バイト単位) を制御します。サーバーは、この値を 4096 の倍数に調整します。このプラグインでは、初期化時に割り当てられ、終了時に削除される単一のバッファーが使用されます。このプラグインは、ロギングが非同期の場合にのみ、このバッファーを割り当てます。

- [audit_log_compression](#)

コマンド行形式	<code>--audit-log-compression=value</code>
システム変数	audit_log_compression
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	NONE
有効な値	NONE GZIP

監査ログファイルの圧縮のタイプ。許可される値は、NONE (圧縮なし、デフォルト) および GZIP (GNU Zip 圧縮) です。詳細は、[監査ログファイルの圧縮](#)を参照してください。

- [audit_log_connection_policy](#)

コマンド行形式	<code>--audit-log-connection-policy=value</code>
システム変数	audit_log_connection_policy
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙

デフォルト値	ALL
有効な値	ALL ERRORS NONE

注記

この変数は、レガシーモードの監査ログフィルタリングにのみ適用されます (セクション 6.4.5.9 「レガシーモード監査ログのフィルタリング」 を参照)。

監査ログプラグインが接続イベントをそのログファイルに書き込む方法を制御するポリシーです。次の表は、許可される値を示しています。

値	説明
ALL	接続イベントのログをすべて記録します
ERRORS	失敗した接続イベントのログのみを記録します
NONE	接続イベントのログを記録しません

注記

セクション 6.4.5.5 「監査ロギング特性の構成」 で説明したように、`audit_log_policy` も指定されている場合は、サーバーの起動時に、`audit_log_connection_policy` に明示的に指定された値がオーバーライドされる可能性があります。

- [audit_log_current_session](#)

システム変数	audit_log_current_session
スコープ	グローバル、セッション
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	depends on filtering policy

現在のセッションで監査ロギングが有効になっているかどうかを示します。この変数のセッションの値は、読み取り専用です。セッションの開始時に、`audit_log_include_accounts` および `audit_log_exclude_accounts` システム変数の値に基づいて設定されます。監査ログプラグインはこのセッション値を使用して、そのセッションでイベントを監査するかどうかを決定します。(グローバル値もありますが、このプラグインでは使用されません。)

- [audit_log_encryption](#)

コマンド行形式	<code>--audit-log-encryption=value</code>
システム変数	audit_log_encryption
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	NONE
有効な値	NONE

AES

監査ログファイルの暗号化のタイプ。許可される値は、[NONE](#) (暗号化なし、デフォルト) および [AES](#) (AES-256-CBC 暗号化) です。詳細は、[監査ログファイルの暗号化](#)を参照してください。

- [audit_log_exclude_accounts](#)

コマンド行形式	<code>--audit-log-exclude-accounts=value</code>
システム変数	audit_log_exclude_accounts
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	NULL

注記

この変数は、レガシーモードの監査ログフィルタリングにのみ適用されます ([セクション 6.4.5.9 「レガシーモード監査ログのフィルタリング」](#) を参照)。

イベントのログが記録されないアカウント。この値には、[NULL](#) またはカンマで区切った 1 つ以上のアカウント名のリストを含む文字列を指定するようにしてください。詳細は、[セクション 6.4.5.7 「監査ログのフィルタリング」](#) を参照してください。

[audit_log_exclude_accounts](#) の変更は、変更後に作成された接続にのみ影響し、既存の接続には影響しません。

- [audit_log_file](#)

コマンド行形式	<code>--audit-log-file=file_name</code>
システム変数	audit_log_file
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	audit.log

監査ログプラグインがイベントを書き込むファイルのベース名と接尾辞。ロギング形式に関係なく、デフォルト値は [audit.log](#) です。名前接尾辞をフォーマットに対応させるには、別の接尾辞を選択して、名前を明示的に設定します (たとえば、XML フォーマットの場合は [audit.xml](#)、JSON フォーマットの場合は [audit.json](#))。

[audit_log_file](#) の値が相対パス名の場合、プラグインはデータディレクトリを基準にした相対パス名を解釈します。値がフルパス名の場合、プラグインは値をそのまま使用します。フルパス名は、監査ファイルを別のファイルシステムまたはディレクトリに配置することをお勧めします。セキュリティ上の理由から、MySQL サーバーおよびログを表示する正当な理由を持つユーザーのみがアクセスできるディレクトリに監査ログファイルを書き込みます。

監査ログプラグインが [audit_log_file](#) 値を解釈する方法と、プラグインの初期化および終了時に発生するファイル名変更の規則の詳細は、[監査ログファイルのネーミング規則](#) を参照してください。

監査ログプラグインは、読み取り可能な監査ログファイルを検索する場所として、([audit_log_file](#) の値から決定された) 監査ログファイルを含むディレクトリを使用します。これらのログファイルと現在のファイルから、プラグインは監査ログのブックマークおよび読み取り関数で使用する対象のものリストを構築します。[セクション 6.4.5.6 「監査ログファイルの読取り」](#) を参照してください。

- [audit_log_filter_id](#)

システム変数	audit_log_filter_id
スコープ	グローバル、セッション
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer

この変数のセッション値は、現在のセッションの監査フィルタの内部的に保持されている ID を示します。値 0 は、セッションにフィルタが割り当てられていないことを意味します。

- [audit_log_flush](#)

システム変数	audit_log_flush
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

この変数が有効になるよう (1 または ON) に設定されている場合、監査ログプラグインはそのログファイルを閉じてから再度開いて、フラッシュします。(この値は、別のフラッシュを実行するために再度有効にする前に、明示的に無効にする必要がないように、OFF のままになっています。) [audit_log_rotate_on_size](#) が 0 である場合を除いて、この変数を有効にしても効果はありません。詳細は、[セクション6.4.5.5「監査ロギング特性の構成」](#)を参照してください。

- [audit_log_format](#)

コマンド行形式	<code>--audit-log-format=value</code>
システム変数	audit_log_format
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	NEW
有効な値	OLD NEW JSON

監査ログファイルの形式。許可される値は、OLD (古いスタイルの XML)、NEW (新しいスタイルの XML。デフォルト) および JSON です。各形式についての詳細は、[セクション6.4.5.4「監査ログファイル形式」](#)を参照してください。

注記

ログ形式を変更する際に考慮する問題の詳細は、[監査ログファイル形式の選択](#)を参照してください。

- [audit_log_include_accounts](#)

コマンド行形式	<code>--audit-log-include-accounts=value</code>	1339
---------	---	------

システム変数	audit_log_include_accounts
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	NULL

注記

この変数は、レガシーモードの監査ログフィルタリングにのみ適用されます ([セクション 6.4.5.9 「レガシーモード監査ログのフィルタリング」](#) を参照)。

イベントのログが記録されるアカウント。この値には、NULL またはカンマで区切った 1 つ以上のアカウント名のリストを含む文字列を指定するようにしてください。詳細は、[セクション 6.4.5.7 「監査ログのフィルタリング」](#) を参照してください。

[audit_log_include_accounts](#) の変更は、変更後に作成された接続にのみ影響し、既存の接続には影響しません。

- [audit_log_password_history_keep_days](#)

コマンド行形式	<code>--audit-log-password-history-keep-days=#</code>
導入	8.0.17
システム変数	audit_log_password_history_keep_days
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	4294967295

監査ログプラグインは、MySQL 鍵リングに格納されている暗号化パスワードを使用してログファイルの暗号化を実装します ([監査ログファイルの暗号化](#) を参照)。このプラグインは、パスワードのアーカイブと有効期限 (削除) を含むパスワード履歴も実装します。

監査ログプラグインは、新しい暗号化パスワードを作成するときに、以前のパスワード (存在する場合) をあとで使用するためにアーカイブします。 [audit_log_password_history_keep_days](#) 変数は、期限切れのアーカイブ済パスワードの自動削除を制御します。この値は、アーカイブ監査ログの暗号化パスワードが削除されるまでの日数を示します。デフォルトの 0 は、パスワードの有効期限を無効にします: パスワードの保存期間は永久的です。

新しい監査ログ暗号化パスワードは、次の状況で作成されます:

- プラグインの初期化中に、プラグインがログファイルの暗号化が有効であることを検出すると、キーリングに監査ログの暗号化パスワードが含まれているかどうかをチェックします。そうでない場合、プラグインはランダムな初期暗号化パスワードを自動的に生成します。
- 特定のパスワードを設定するために [audit_log_encryption_password_set\(\)](#) 関数がコールされた場合。

いずれの場合も、プラグインは新しいパスワードをキーリングに格納し、それを使用して新しいログファイルを暗号化します。

期限切れの監査ログ暗号化パスワードの削除は、次の状況で発生します:

- プラグインの初期化中。

- `audit_log_encryption_password_set()` 関数がコールされたとき。
- `audit_log_password_history_keep_days` のランタイム値が現在の値から 0 より大きい値に変更された場合。ランタイム値の変更は、`GLOBAL` または `PERSIST` キーワードを使用するが `PERSIST_ONLY` キーワードを使用しない `SET` ステートメントで発生します。 `PERSIST_ONLY` は変数設定を `mysqld-auto.cnf` に書き込みますが、ランタイム値には影響しません。

パスワードの削除が発生すると、`audit_log_password_history_keep_days` の現在の値によって、削除するパスワードが決まります:

- 値が 0 の場合、プラグインはパスワードを削除しません。
- 値が $N > 0$ の場合、プラグインは N 日より古いパスワードを削除します。

注記

アーカイブされた暗号化ログファイルの読取りに必要な古いパスワードを期限切れにしないように注意してください。

通常、パスワードの有効期限を無効のままにすると (つまり、`audit_log_password_history_keep_days` の値が 0 の場合)、変数にゼロより大きい値を一時的に割り当てることで、オンデマンドクリーンアップ操作を実行できます。たとえば、365 日より古いパスワードを期限切れにするには、次のようにします:

```
SET GLOBAL audit_log_password_history_keep_days = 365;
SET GLOBAL audit_log_password_history_keep_days = 0;
```

`audit_log_password_history_keep_days` のランタイム値を設定するには、グローバルシステム変数のランタイム値を設定するために通常必要な `SYSTEM_VARIABLES_ADMIN` 権限 (または非推奨の `SUPER` 権限) に加えて、`AUDIT_ADMIN` 権限が必要です。

- `audit_log_policy`

コマンド行形式	<code>--audit-log-policy=value</code>
システム変数	<code>audit_log_policy</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	<code>ALL</code>
有効な値	<code>ALL</code> <code>LOGINS</code> <code>QUERIES</code> <code>NONE</code>

注記

この変数は、レガシーモードの監査ログフィルタリングにのみ適用されます ([セクション 6.4.5.9 「レガシーモード監査ログのフィルタリング」](#) を参照)。

監査ログプラグインがイベントをそのログファイルに書き込む方法を制御するポリシーです。次の表は、許可される値を示しています。

値	説明
<code>ALL</code>	イベントのログをすべて記録します

値	説明
LOGINS	ログインイベントのログのみを記録します
QUERIES	クエリーイベントのログのみを記録します
NONE	ログを何も記録しません (監査ストリームを無効にします)

`audit_log_policy` は、サーバーの起動時にのみ設定できます。実行時は、読み取り専用の変数です。他の 2 つのシステム変数 (`audit_log_connection_policy` および `audit_log_statement_policy`) は、ロギングポリシーをより細かく制御し、起動時または実行時に設定できます。起動時に他の 2 つの変数ではなく `audit_log_policy` を使用する場合は、サーバーはその値を使用してこれらの変数を設定します。ポリシーの変数とそれらの相互作用についての詳細は、[セクション6.4.5.5「監査ロギング特性の構成」](#)を参照してください。

- `audit_log_prune_seconds`

コマンド行形式	<code>--audit-log-prune-seconds=#</code>
導入	8.0.24
システム変数	<code>audit_log_prune_seconds</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	4294967295
単位	seconds

この変数は、JSON 形式のログファイルでのみサポートされている監査ログファイルのプルーニングに関連します。

`audit_log_rotate_on_size` が 0 より大きい場合を除き、`audit_log_prune_seconds` は効果がありません。true であると仮定した場合:

- `audit_log_prune_seconds` が 0 (デフォルト) の場合、プルーニングは無効になり、サイズベースのローテーションの結果として作成されたログファイルは無期限に蓄積されます。
- `audit_log_prune_seconds` が 0 より大きい場合、プルーニングは有効で、値は監査ログファイルがプルーニングの対象になるまでの秒数です。

プルーニングを有効にすると、[監査ログファイルの領域管理](#) で説明されている条件で実行されます。

- `audit_log_read_buffer_size`

コマンド行形式	<code>--audit-log-read-buffer-size=#</code>
システム変数	<code>audit_log_read_buffer_size</code>
スコープ (≥ 8.0.12)	グローバル、セッション
スコープ (8.0.11)	グローバル
動的 (≥ 8.0.12)	はい
動的 (8.0.11)	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値 (≥ 8.0.12)	32768

デフォルト値 (8.0.11)	1048576
最小値 (≥ 8.0.12)	32768
最小値 (8.0.11)	1024
最大値	4194304

監査ログファイルから読み取るバッファサイズ (バイト単位)。 `audit_log_read()` 関数は、このバイト数以下を読み取ります。ログファイルの読取りは、JSON ログ形式でのみサポートされます。詳細は、[セクション6.4.5.6「監査ログファイルの読取り」](#)を参照してください。

MySQL 8.0.12 では、この変数のデフォルトは 32KB で、実行時に設定できます。各クライアントは、`audit_log_read()` を使用するために `audit_log_read_buffer_size` のセッション値を適切に設定する必要があります。MySQL 8.0.12 より前では、`audit_log_read_buffer_size` のデフォルトは 1MB で、すべてのクライアントに影響し、サーバーの起動時にのみ変更できます。

- [audit_log_rotate_on_size](#)

コマンド行形式	<code>--audit-log-rotate-on-size=#</code>
システム変数	<code>audit_log_rotate_on_size</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
単位	bytes

`audit_log_rotate_on_size` が 0 の場合、監査ログプラグインはサイズベースのログファイルの自動ローテーションを実行しません。代わりに、`audit_log_flush` 使用して、要求に応じてログを閉じてから再度開きます。この場合は、フラッシュする前に、ファイルの名前をサーバーの外部に手動で変更します。

`audit_log_rotate_on_size` が 0 より大きい場合、サイズベースのログファイルの自動ローテーションが発生します。ログファイルへの書き込みによってそのサイズが `audit_log_rotate_on_size` 値を超えるたびに、監査ログプラグインは現在のログファイルを閉じて名前を変更し、新しいログファイルを開きます。

`audit_log_rotate_on_size` を 4096 の倍数でない値に設定すると、最も近い倍数に切り捨てられます。(このため、4096 未満の値に設定すると 0 に設定され、手動以外は回転は発生しません。)

MySQL 8.0.24 では、`audit_log_rotate_on_size` は監査ログファイルのプルーニングを有効にできるかどうかを制御します:

- `audit_log_rotate_on_size` が無効 (0) の場合、プルーニングは有効にできず、`audit_log_prune_seconds` は無効になります。
- `audit_log_rotate_on_size` が有効になっている (0 より大きい) 場合は、プルーニングを有効にでき、`audit_log_prune_seconds` によってプルーニングが行われるかどうか決定されます。

監査ログファイルのローテーションおよびプルーニングの詳細は、[監査ログファイルの領域管理](#) を参照してください。

- [audit_log_statement_policy](#)

コマンド行形式	<code>--audit-log-statement-policy=value</code>
システム変数	<code>audit_log_statement_policy</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ

型	列挙
デフォルト値	ALL
有効な値	ALL ERRORS NONE

注記

この変数は、レガシーモードの監査ログフィルタリングにのみ適用されます ([セクション 6.4.5.9「レガシーモード監査ログのフィルタリング」](#) を参照)。

監査ログプラグインがステートメントイベントをそのログファイルに書き込む方法を制御するポリシーです。次の表は、許可される値を示しています。

値	説明
ALL	ステートメントイベントのログをすべて記録します
ERRORS	失敗したステートメントイベントのログのみを記録します
NONE	ステートメントイベントのログを記録しません

注記

[セクション 6.4.5.5「監査ロギング特性の構成」](#) で説明したように、`audit_log_policy` も指定されている場合は、サーバーの起動時に、`audit_log_statement_policy` に明示的に指定された値がオーバーライドされる可能性があります。

• [audit_log_strategy](#)

コマンド行形式	<code>--audit-log-strategy=value</code>
システム変数	<code>audit_log_strategy</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	ASYNCHRONOUS
有効な値	ASYNCHRONOUS PERFORMANCE SEMISYNCHRONOUS SYNCHRONOUS

監査ログプラグインで使用されるロギング方法。次の戦略値を使用できます:

- **ASYNCHRONOUS**: 非同期でログを記録します。出力バッファ内の領域を待機します。
- **PERFORMANCE**: 非同期でログを記録します。出力バッファに十分な領域がないリクエストを削除します。
- **SEMISYNCHRONOUS**: 同期的にログを記録します。オペレーティングシステムによるキャッシュを許可します。
- **SYNCHRONOUS**: 同期的にログを記録します。各リクエストの後に `sync()` をコールします。

監査ログステータス変数

監査ログプラグインが有効になっている場合は、操作情報を提供するいくつかのステータス変数が公開されます。これらの変数は、レガシーモードの監査フィルタリングおよび JSON モードの監査フィルタリングに使用できます。

- [Audit_log_current_size](#)

現在の監査ログファイルのサイズ。この値は、イベントがログに書き込まれると増加し、ログがローテーションされると 0 にリセットされます。

- [Audit_log_event_max_drop_size](#)

パフォーマンスロギングモードで破棄された最大イベントのサイズ。ロギングモードについては、[セクション 6.4.5.5 「監査ロギング特性の構成」](#)を参照してください。

- [Audit_log_events](#)

フィルタリングのポリシーに基づいてログに書き込まれたかどうかに関係なく、監査ログプラグインによって処理されたイベントの数 ([セクション 6.4.5.5 「監査ロギング特性の構成」](#)を参照してください)。

- [Audit_log_events_filtered](#)

フィルタリングのポリシーに基づいて、監査ログプラグインによってフィルタリングされた (ログに書き込まれなかった) イベントの数 ([セクション 6.4.5.5 「監査ロギング特性の構成」](#)を参照してください)。

- [Audit_log_events_lost](#)

イベントが使用可能な監査ログバッファ領域よりも大きかったために、パフォーマンスロギングモードで失われたイベントの数。この値は、[audit_log_buffer_size](#) を設定して、パフォーマンスモード用にバッファのサイズを調整する方法を評価する際に役立つことがあります。ロギングモードについては、[セクション 6.4.5.5 「監査ロギング特性の構成」](#)を参照してください。

- [Audit_log_events_written](#)

監査ログに書き込まれたイベントの数。

- [Audit_log_total_size](#)

すべての監査ログファイルに書き込まれたイベントの合計サイズ。[Audit_log_current_size](#) とは異なり、[Audit_log_total_size](#) の値は、ログがローテーションされたときにも増加します。

- [Audit_log_write_waits](#)

非同期ロギングモードでイベントが監査ログバッファ内の領域を待機する必要があった回数。ロギングモードについては、[セクション 6.4.5.5 「監査ロギング特性の構成」](#)を参照してください。

6.4.5.11 監査ログの制限

MySQL Enterprise Audit には、次の一般的な制限事項があります:

- SQL ステートメントのみがログに記録されます。memcached、Node.JS、NDB API などの非 SQL API によって行われた変更はログに記録されません。
- 最上位のステートメントのみのログが記録され、トリガーやストアードプロシージャなどのストアードプログラム内のステートメントのログは記録されません。
- **LOAD DATA** などのステートメントで参照されるファイルの内容は記録されません。

NDB Cluster. 次の条件に従って、MySQL NDB Cluster で MySQL Enterprise Audit を使用できます:

- ログに記録されるすべての変更は、SQL インタフェースを使用して行う必要があります。NDB API、memcached、または ClusterJ によって提供されるインタフェースなど、SQL 以外のインタフェースを使用した変更はログに記録されません。
- このプラグインは、クラスターで SQL を実行するために使用される各 MySQL サーバーにインストールする必要があります。

- 監査プラグインデータは、クラスタで使用されるすべての MySQL サーバー間で集約する必要があります。この集計は、アプリケーションまたはユーザーの役割を果たします。

6.4.6 監査メッセージコンポーネント

MySQL 8.0.14 では、`audit_api_message_emit` コンポーネントを使用して、`audit_api_message_emit_udf()` ユーザー定義関数を使用して独自のメッセージイベントを監査ログに追加できます。

`audit_api_message_emit` コンポーネントは、監査タイプのすべてのプラグインと連携します。簡潔にするために、例では [セクション6.4.5「MySQL Enterprise Audit」](#) で説明されている `audit_log` プラグインを使用します。

- [監査メッセージコンポーネントのインストールまたはアンインストール](#)
- [監査メッセージ機能](#)

監査メッセージコンポーネントのインストールまたはアンインストール

サーバーで使用できるようにするには、コンポーネントライブラリファイルが MySQL プラグインディレクトリ (`plugin_dir` システム変数で指定されたディレクトリ) にある必要があります。必要に応じて、サーバーの起動時に `plugin_dir` の値を設定してプラグインディレクトリの場所を構成します。

`audit_api_message_emit` コンポーネントをインストールするには、次のステートメントを使用します:

```
INSTALL COMPONENT "file://component_audit_api_message_emit";
```

コンポーネントのインストールは、サーバーの起動ごとに実行する必要のない一度限りの操作です。 `INSTALL COMPONENT` によってコンポーネントがロードされ、`mysql.component` システムテーブルにも登録されて、後続のサーバー起動時にロードされます。

`audit_api_message_emit` コンポーネントをアンインストールするには、次のステートメントを使用します:

```
UNINSTALL COMPONENT "file://component_audit_api_message_emit";
```

`UNINSTALL COMPONENT` はコンポーネントをアンロードし、`mysql.component` システムテーブルから登録解除して、後続のサーバー起動時にロードされないようにします。

`audit_api_message_emit` コンポーネントをインストールおよびアンインストールすると、コンポーネントが実装する `audit_api_message_emit_udf()` 関数がインストールおよびアンインストールされるため、`CREATE FUNCTION` または `DROP FUNCTION` を使用して行う必要はありません。

監査メッセージ機能

このセクションでは、`audit_api_message_emit` コンポーネントによって実装される `audit_api_message_emit_udf()` ユーザー定義関数 (UDF) について説明します。

監査メッセージ機能を使用する前に、[監査メッセージコンポーネントのインストールまたはアンインストール](#) の指示に従って監査メッセージコンポーネントをインストールします。

- `audit_api_message_emit_udf(component, producer, message[, key, value] ...)`

監査ログにメッセージイベントを追加します。メッセージイベントには、コール元が選択するコンポーネント、プロデューサおよびメッセージ文字列と、オプションでキーと値のペアのセットが含まれます。

この UDF によってポストされたイベントは、監査タイプの有効なすべてのプラグインに送信され、それぞれが独自のルールに従ってイベントを処理します。監査タイプのプラグインが有効になっていない場合、イベントをポストしても効果はありません。

引数:

- `component`: コンポーネント名を指定する文字列。
- `producer`: プロデューサ名を指定する文字列。
- `message`: イベントメッセージを指定する文字列。

- **key, value**: イベントには、任意のアプリケーション提供のデータマップを指定する 0 個以上のキーと値のペアを含めることができます。各 **key** 引数は、**value** 引数の直後の名前を指定する文字列です。各 **value** 引数は、**key** 引数の直後の値を指定します。各 **value** には、文字列、数値または **NULL** を指定できます。

戻り値:

成功を示す文字列 **OK**。関数が失敗すると、エラーが発生します。

例:

```
mysql> SELECT audit_api_message_emit_udf('component_text',
    'producer_text',
    'message_text',
    'key1', 'value1',
    'key2', 123,
    'key3', NULL) AS 'Message';
+-----+
| Message |
+-----+
| OK      |
+-----+
```

追加情報:

`audit_api_message_emit_udf()` によってポストされたイベントを受信する各監査プラグインは、プラグイン固有の形式でイベントを記録します。たとえば、`audit_log` プラグイン ([セクション6.4.5「MySQL Enterprise Audit」](#)を参照)は、`audit_log_format` システム変数で構成されたログ形式に応じて、次のようにメッセージ値をログに記録します:

- JSON 形式 (`audit_log_format=JSON`):

```
{
  ...
  "class": "message",
  "event": "user",
  ...
  "message_data": {
    "component": "component_text",
    "producer": "producer_text",
    "message": "message_text",
    "map": {
      "key1": "value1",
      "key2": 123,
      "key3": null
    }
  }
}
```

- 「新規スタイルの XML」形式 (`audit_log_format=NEW`):

```
<AUDIT_RECORD>
...
<NAME>Message</NAME>
...
<COMMAND_CLASS>user</COMMAND_CLASS>
<COMPONENT>component_text</COMPONENT>
<PRODUCER>producer_text</PRODUCER>
<MESSAGE>message_text</MESSAGE>
<MAP>
  <ELEMENT>
    <KEY>key1</KEY>
    <VALUE>value1</VALUE>
  </ELEMENT>
  <ELEMENT>
    <KEY>key2</KEY>
    <VALUE>123</VALUE>
  </ELEMENT>
  <ELEMENT>
    <KEY>key3</KEY>
    <VALUE/>
  </ELEMENT>
</MAP>
```

```
</ELEMENT>
</MAP>
</AUDIT_RECORD>
```

- 「古いスタイルの XML」形式 (`audit_log_format=OLD`):

```
<AUDIT_RECORD
...
NAME="Message"
...
COMMAND_CLASS="user"
COMPONENT="component_text"
PRODUCER="producer_text"
MESSAGE="message_text"/>
```

注記

古い形式の XML 形式で記録されたメッセージイベントには、この形式による表現上の制約のため、キーと値のマッピングは含まれません。

`audit_api_message_emit_udf()` によってポストされるメッセージには、`MYSQL_AUDIT_MESSAGE_CLASS` のイベントクラスと `MYSQL_AUDIT_MESSAGE_USER` のサブクラスがあります。(相互に生成された監査メッセージは、同じクラスと `MYSQL_AUDIT_MESSAGE_INTERNAL` のサブクラスを持ちます。このサブクラスは現在使用されていません。) `audit_log` フィルタリングルールでこのようなイベントを参照するには、`name` 値が `message` の `class` 要素を使用します。例:

```
{
  "filter": {
    "class": {
      "name": "message"
    }
  }
}
```

ユーザー生成メッセージイベントと内部生成メッセージイベントを区別する必要がある場合は、`user` または `internal` に対して `subclass` 値をテストします。

キーと値のマッピングの内容に基づくフィルタリングはサポートされていません。

フィルタリングルールの作成の詳細は、[セクション6.4.5.7「監査ログのフィルタリング」](#) を参照してください。

6.4.7 MySQL Enterprise Firewall

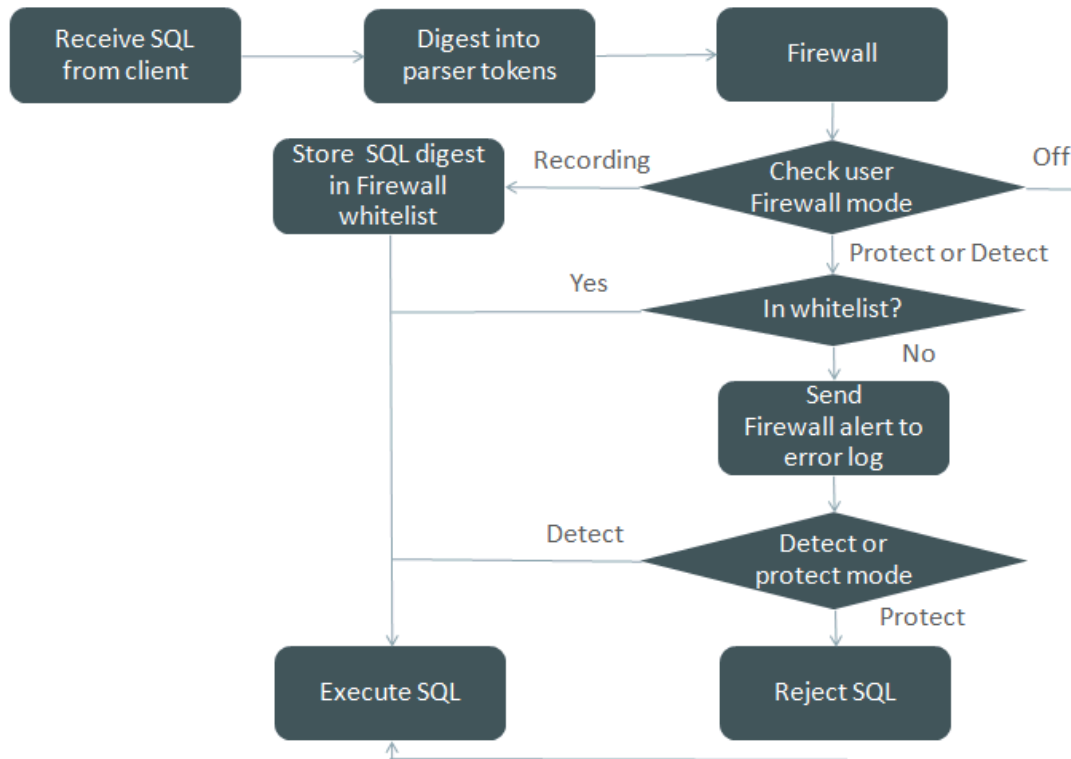
注記

MySQL Enterprise Firewall は、商用製品である MySQL Enterprise Edition に含まれる拡張機能です。商用製品の詳細は、<https://www.mysql.com/products/> を参照してください。

MySQL Enterprise Edition には、アプリケーションレベルのファイアウォールである MySQL Enterprise Firewall が含まれています。これにより、データベース管理者は、受け入れられたステートメントパターンのリストに対する照合に基づいて SQL ステートメントの実行を許可または拒否できます。これにより、SQL インジェクションなどの攻撃や、正当なクエリーワークロード特性の外部でアプリケーションを使用することで、アプリケーションを利用しようとする攻撃に対して MySQL Server を強化できます。

ファイアウォールに登録された各 MySQL アカウントには独自のステートメント allowlist があり、アカウントごとに保護を調整できます。特定のアカウントについて、ファイアウォールは記録、保護または検出モードで動作し、受け入れられるステートメントパターンでのトレーニング、受け入れられないステートメントに対するアクティブな保護、または受入れ不可能なステートメントの受動的な検出を行うことができます。この図は、ファイアウォールが各モードで受信ステートメントを処理する方法を示しています。

図 6.1 MySQL Enterprise Firewall 操作



次の各セクションでは、MySQL Enterprise Firewall の要素について説明し、それをインストールおよび使用方法を説明し、その要素のリファレンス情報を提供します。

6.4.7.1 MySQL Enterprise Firewall の要素

MySQL Enterprise Firewall は、次の要素を含むプラグインライブラリに基づいています：

- `MYSQL_FIREWALL` という名前のサーバー側プラグインは、実行前に SQL ステートメントを調べ、登録されたファイアウォールプロファイルに基づいて、各ステートメントを実行するか拒否するかを決定します。
- `MYSQL_FIREWALL` プラグインは、`MYSQL_FIREWALL_USERS` および `MYSQL_FIREWALL_WHITELIST` という名前のサーバー側プラグインとともに、登録済プロファイルへのビューを提供するパフォーマンススキーマおよび `INFORMATION_SCHEMA` テーブルを実装します。
- プロファイルは、パフォーマンスを向上させるためにメモリーにキャッシュされます。mysql システムデータベースのテーブルは、サーバーの再起動後もプロファイルの永続性を維持するために、ファイアウォールデータのバッキング記憶域を提供します。
- ストアドプロシージャは、ファイアウォールプロファイルの登録、操作モードの確立、キャッシュと永続記憶域間のファイアウォールデータの転送の管理などのタスクを実行します。
- ユーザー定義関数は、キャッシュと永続記憶域の同期化などの下位レベルのタスク用の API を提供します。
- システム変数はファイアウォール構成を可能にし、ステータス変数はランタイム操作情報を提供します。
- `FIREWALL_ADMIN` および `FIREWALL_USER` 権限を使用すると、ユーザーはそれぞれ任意のユーザーのファイアウォールルールおよび独自のファイアウォールルールを管理できます。

6.4.7.2 MySQL Enterprise Firewall のインストールまたはアンインストール

MySQL Enterprise Firewall のインストールは、セクション6.4.7.1「MySQL Enterprise Firewall の要素」で説明されている要素をインストールするワンタイム操作です。インストールはグラフィカルインターフェースを使用して実行することも、手動で実行することもできます：

- Windows では、MySQL Installer に MySQL Enterprise Firewall を有効にするオプションが含まれています。
- MySQL Workbench 6.3.4 以上では、MySQL Enterprise Firewall のインストール、インストールされたファイアウォールの有効化または無効化、ファイアウォールのアンインストールを行うことができます。
- MySQL Enterprise Firewall の手動インストールでは、MySQL インストールの [share](#) ディレクトリにあるスクリプトを実行します。

重要

指示に従う前に、このセクション全体をお読みください。手順の一部は、環境によって異なります。

注記

インストールされている場合、MySQL Enterprise Firewall は、無効化されていても最小限のオーバーヘッドを伴います。このオーバーヘッドを回避するには、ファイアウォールを使用する予定がないかぎり、ファイアウォールをインストールしないでください。

使用手順については、[セクション6.4.7.3「MySQL Enterprise Firewall の使用」](#)を参照してください。参照情報については、[セクション6.4.7.4「MySQL Enterprise Firewall リファレンス」](#)を参照してください。

- [MySQL Enterprise Firewall のインストール](#)
- [MySQL Enterprise Firewall のアンインストール](#)

MySQL Enterprise Firewall のインストール

MySQL Enterprise Firewall が古いバージョンの MySQL からすでにインストールされている場合は、このセクションで後述する手順を使用してアンインストールし、現在のバージョンをインストールする前にサーバーを再起動します。この場合は、構成を再度登録する必要もあります。

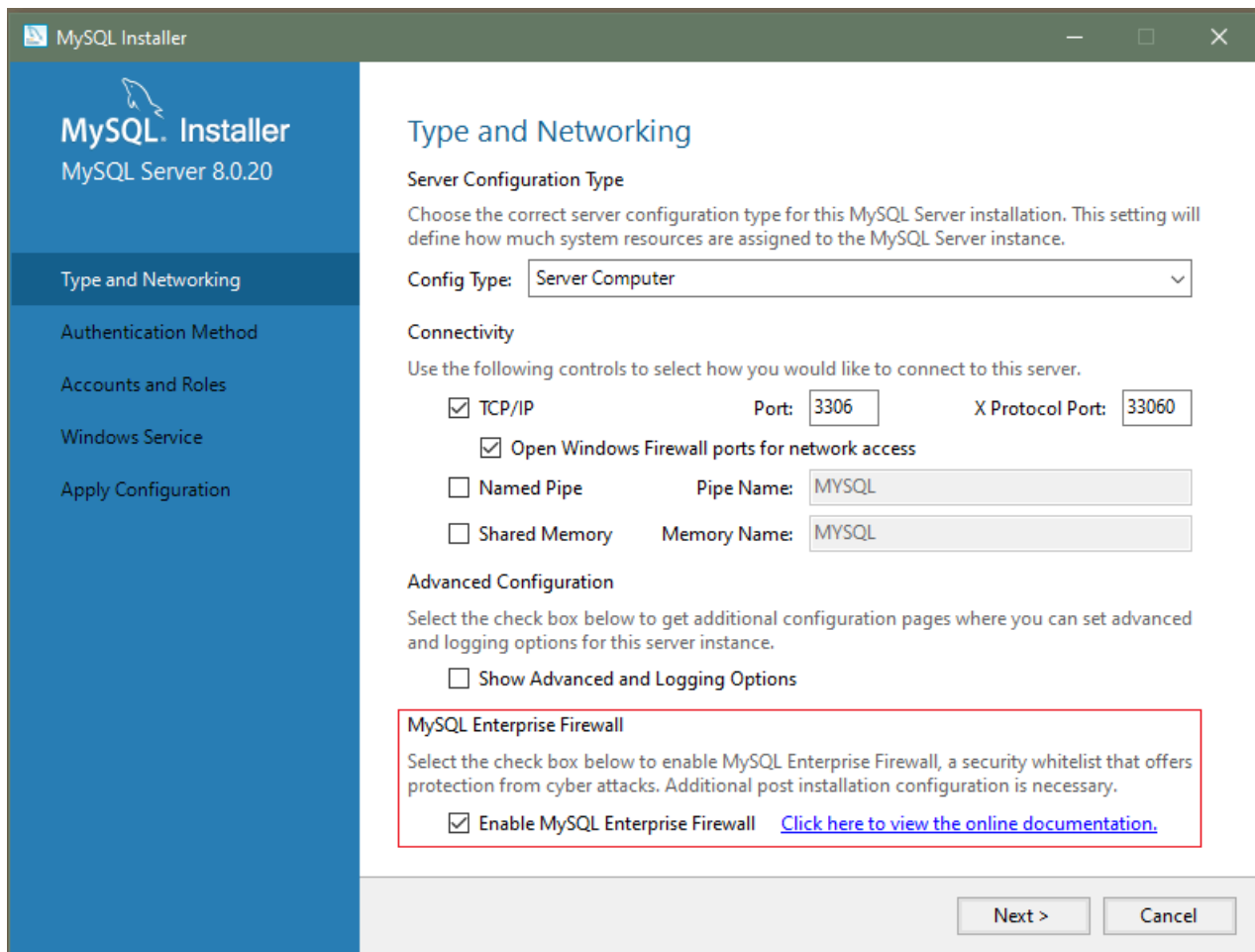
Windows では、[図6.2「Windows での MySQL Enterprise Firewall のインストール」](#)に示すように、MySQL Installer を使用して MySQL Enterprise Firewall をインストールできます。「MySQL Enterprise Firewall の有効化」チェックボックスを選択します。（「ネットワークアクセス用の Open Firewall ポート」には異なる目的があります。Windows ファイアウォールを参照し、MySQL サーバーがクライアント接続をリスニングする TCP/IP ポートを Windows でブロックするかどうかを制御します。）

重要

MySQL Installer を使用してインストールされた MySQL 8.0.19 には、サーバーの構成ステップ中に MySQL Enterprise Firewall が選択された場合にサーバーを起動できない問題があります。サーバーの起動操作が失敗した場合は、取消をクリックして構成プロセスを終了し、ダッシュボードに戻ります。サーバーをアンインストールする必要があります。

回避策は、MySQL Enterprise Firewall を選択せずに MySQL Installer を実行することです。（つまり、「MySQL Enterprise Firewall の有効化」チェックボックスを選択しないでください。）その後、このセクションの後半にある手動インストールの手順を使用して、MySQL Enterprise Firewall をインストールします。この問題は、MySQL 8.0.20 で修正されています。

図 6.2 Windows での MySQL Enterprise Firewall のインストール



MySQL Workbench 6.3.4 以上を使用して MySQL Enterprise Firewall をインストールするには、[MySQL Enterprise Firewall Interface](#) を参照してください。

MySQL Enterprise Firewall を手動でインストールするには、MySQL インストールの `share` ディレクトリを検索し、ご使用のプラットフォームに適したスクリプトを選択します。使用可能なスクリプトは、プラグインライブラリファイルの参照に使用される接尾辞とは異なります：

- `win_install_firewall.sql`: ファイル名の接尾辞として `.dll` を使用する Windows システムの場合は、このスクリプトを選択します。
- `linux_install_firewall.sql`: Linux および `.so` をファイル名接尾辞として使用する類似システムの場合は、このスクリプトを選択します。

インストールスクリプトはデフォルトデータベースにストアードプロシージャを作成するため、使用するデータベースを選択します。次に、コマンドラインで選択したデータベースに名前を付けて、次のようにスクリプトを実行します。この例では、`mysql` システムデータベースおよび Linux インストールスクリプトを使用します。システムに適切な置換を行います。

```
shell> mysql -u root -p mysql < linux_install_firewall.sql
Enter password: (enter root password here)
```

注記

ソース/レプリカレプリケーション、グループレプリケーションまたは InnoDB クラスターのコンテキストで MySQL Enterprise Firewall を使用するには、ソースノードでインストール

スクリプトを実行する前にレプリカノードを準備する必要があります。これが必要なのは、スクリプト内の `INSTALL PLUGIN` ステートメントがレプリケートされないためです。

1. 各レプリカノードで、インストールスクリプトから `INSTALL PLUGIN` ステートメントを抽出し、手動で実行します。
2. ソースノードで、前述のようにインストールスクリプトを実行します。

グラフィカルインターフェースを使用して MySQL Enterprise Firewall をインストールするか、手動でファイアウォールを有効にする必要があります。これを確認するには、サーバーに接続して次のステートメントを実行します:

```
mysql> SHOW GLOBAL VARIABLES LIKE 'mysql_firewall_mode';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| mysql_firewall_mode | ON |
+-----+-----+
```

プラグインの初期化に失敗した場合は、サーバーエラーログで診断メッセージを確認してください。

MySQL Enterprise Firewall のアンインストール

MySQL Enterprise Firewall は、MySQL Workbench を使用して、または手動でアンインストールできます。

MySQL Workbench 6.3.4 以上を使用して MySQL Enterprise Firewall をアンインストールするには、[第31章「MySQL Workbench」](#) の [MySQL Enterprise Firewall Interface](#) を参照してください。

MySQL Enterprise Firewall を手動でアンインストールするには、次のステートメントを実行します。ステートメントは、以前にインストールされたファイアウォールバージョンによっては、一部のオブジェクトが存在しないか、それらをインストールしたプラグインをアンインストールすることによって暗黙的に削除される可能性があるため、`IF EXISTS` を使用します。

```
DROP TABLE IF EXISTS mysql.firewall_group_allowlist;
DROP TABLE IF EXISTS mysql.firewall_groups;
DROP TABLE IF EXISTS mysql.firewall_membership;
DROP TABLE IF EXISTS mysql.firewall_users;
DROP TABLE IF EXISTS mysql.firewall_whitelist;

UNINSTALL PLUGIN MYSQL_FIREWALL;
UNINSTALL PLUGIN MYSQL_FIREWALL_USERS;
UNINSTALL PLUGIN MYSQL_FIREWALL_WHITELIST;

DROP FUNCTION IF EXISTS firewall_group_delist;
DROP FUNCTION IF EXISTS firewall_group_enlist;
DROP FUNCTION IF EXISTS mysql_firewall_flush_status;
DROP FUNCTION IF EXISTS normalize_statement;
DROP FUNCTION IF EXISTS read_firewall_group_allowlist;
DROP FUNCTION IF EXISTS read_firewall_groups;
DROP FUNCTION IF EXISTS read_firewall_users;
DROP FUNCTION IF EXISTS read_firewall_whitelist;
DROP FUNCTION IF EXISTS set_firewall_group_mode;
DROP FUNCTION IF EXISTS set_firewall_mode;

DROP PROCEDURE IF EXISTS mysql.sp_firewall_group_delist;
DROP PROCEDURE IF EXISTS mysql.sp_firewall_group_enlist;
DROP PROCEDURE IF EXISTS mysql.sp_reload_firewall_group_rules;
DROP PROCEDURE IF EXISTS mysql.sp_reload_firewall_rules;
DROP PROCEDURE IF EXISTS mysql.sp_set_firewall_group_mode;
DROP PROCEDURE IF EXISTS mysql.sp_set_firewall_group_mode_and_user;
DROP PROCEDURE IF EXISTS mysql.sp_set_firewall_mode;
```

6.4.7.3 MySQL Enterprise Firewall の使用

MySQL Enterprise Firewall を使用する前に、[セクション6.4.7.2「MySQL Enterprise Firewall のインストールまたはアンインストール」](#)に記載されている手順に従ってインストールします。

このセクションでは、SQL ステートメントを使用して MySQL Enterprise Firewall を構成する方法について説明します。または、MySQL Workbench 6.3.4 以上では、ファイアウォール制御用のグラフィカルインターフェースが提供されます。[MySQL Enterprise Firewall Interface](#)を参照してください。

- [ファイアウォールの有効化または無効化](#)
- [ファイアウォール権限の割当て](#)
- [ファイアウォール操作の概念](#)
- [ファイアウォールアカウントプロファイルの登録](#)
- [ファイアウォールグループプロファイルの登録](#)
- [複数の適用可能なプロファイルに対するファイアウォール操作](#)
- [ファイアウォールの監視](#)

ファイアウォールの有効化または無効化

ファイアウォールを有効または無効にするには、`mysql_firewall_mode` システム変数を設定します。デフォルトでは、この変数はファイアウォールのインストール時に有効になります。ファイアウォールの初期状態を明示的に制御するには、サーバーの起動時に変数を設定します。たとえば、オプションファイルでファイアウォールを有効にするには、次の行を使用します:

```
[mysqld]
mysql_firewall_mode=ON
```

`my.cnf` を変更したら、新しい設定を有効にするためにサーバーを再起動します。

または、実行時にファイアウォール設定を設定して永続化するには:

```
SET PERSIST mysql_firewall_mode = OFF;
SET PERSIST mysql_firewall_mode = ON;
```

`SET PERSIST` は、実行中の MySQL インスタンスの値を設定します。また、値が保存され、その後のサーバーの再起動に引き継がれます。後続の再起動に引き継ぐことなく、実行中の MySQL インスタンスの値を変更するには、`PERSIST` ではなく `GLOBAL` キーワードを使用します。[セクション13.7.6.1「変数代入の SET 構文」](#)を参照してください。

ファイアウォール権限の割当て

ファイアウォールがインストールされている状態で、それを管理する予定の MySQL アカウントに適切な権限を付与します。権限は、アカウントが実行を許可するファイアウォール操作によって異なります:

- 完全な管理ファイアウォールアクセスを必要とする任意のアカウントに `FIREWALL_ADMIN` 権限を付与します。(一部のユーザー定義プロシージャは、個々の UDF の説明に示されているように、`FIREWALL_ADMIN` または非推奨の `SUPER` 権限を持つアカウントによって起動できます。)
- 独自のファイアウォールルールに対してのみ管理アクセス権を持つアカウントに `FIREWALL_USER` 権限を付与します。
- `mysql` システムデータベースのファイアウォールストアードプロシージャに対する `EXECUTE` 権限を付与します。これらは UDF を呼び出す可能性があるため、ストアードプロシージャアクセスには、これらの UDF に必要な前述の権限も必要です。

注記

`FIREWALL_ADMIN` および `FIREWALL_USER` 権限は、ファイアウォールコンポーネントで定義されているため、ファイアウォールのインストール中のみ付与できます。

ファイアウォール操作の概念

MySQL サーバーを使用すると、クライアントはそれらの SQL ステートメントとの間で接続および受信を実行できます。サーバーは、構文エラーですぐには失敗しない各受信ステートメントをファイアウォールに渡します。ファイアウォールがステートメントを受け入れるかどうかに基づいて、サーバーはステートメントを実行するか、クライアントにエラーを返します。

ファイアウォール操作は、ステートメントの実行保護を適用できるプロファイルのレジストリに基づいています。プロファイルには次の属性があります:

- プロファイルで受け入れ可能なステートメントを定義するルール。このルールセットは、プロファイル allowlist を形成します。
- 現在の操作モード。このモードでは、プロファイルを様々な方法で使用できます。次に例を示します: プロファイルをトレーニングモードにすると、許可リストを確立できます。許可リストは、ステートメントの実行または侵入の検出を制限するために使用できます。プロファイルは完全に無効にできます。
- プロファイルが適用されるクライアント接続を示す適用範囲。

ファイアウォールは、各プロファイルが特定のクライアントアカウント (クライアントユーザー名とホスト名の組み合わせ) に一致するように、アカウントベースのプロファイルをサポートしています。たとえば、許可リストが `admin@localhost` からの接続に適用されるアカウントプロファイルと、許可リストが `myapp@apphost.example.com` からの接続に適用される別のアカウントプロファイルを登録できます。

MySQL 8.0.23 では、ファイアウォールは複数のアカウントをメンバーとして持つことができるグループプロファイルもサポートしています。グループプロファイルを使用すると、管理が容易になり、特定の一連の許可リストルールを複数のアカウントに適用する必要があるデプロイメントの柔軟性が向上します:

- アカウントプロファイルを使用して、アカウントごとに 1 つのプロファイルを登録し、各プロファイル間で許可リストを複製する必要があります。
- より簡単な方法として、すべてのアカウントをメンバーとして持つグループプロファイルを使用できます。グループプロファイル allowlist はすべてのメンバーアカウントに適用され、アカウントごとに複製する必要はありません。

クライアント接続ごとに、ファイアウォールはどのプロファイルを適用するかを決定し、プロファイルで許可されているステートメントのみを受け入れます。(クライアントがプロファイルに一致しない場合、ファイアウォールはそれを無視し、すべてのステートメントを受け入れます。)

デフォルトでは、ファイアウォールはすべてのステートメントを受け入れ、MySQL アカウントが実行できるステートメントには影響しません。ファイアウォール保護機能を適用するには、明示的なアクションを実行する必要があります:

- ファイアウォールに 1 つ以上のプロファイルを登録します。(これは、ファイアウォールがプロファイルに一致しないクライアントを無視するために必要です。)
- ファイアウォールをトレーニングして、各プロファイルの許可リスト (つまり、プロファイルがクライアントに実行を許可するステートメントのタイプ) を確立します。
- トレーニングされた各プロファイルを使用して MySQL を保護するようにファイアウォールに指示します。つまり、クライアントの接続時に受信ステートメントを適切な許可リストと照合します。

ファイアウォールによって実行されるステートメント照合では、クライアントから受信した SQL ステートメントは使用されません。かわりに、サーバーは受信ステートメントを正規化されたダイジェスト形式に変換し、ファイアウォール操作はこれらのダイジェストを使用します。ステートメントの正規化の利点は、単一のパターンを使用して同様のステートメントをグループ化および認識できることです。たとえば、次のステートメントは互いに異なります:

```
SELECT first_name, last_name FROM customer WHERE customer_id = 1;
select first_name, last_name from customer where customer_id = 99;
SELECT first_name, last_name FROM customer WHERE customer_id = 143;
```

ただし、これらはすべて同じ正規化ダイジェストフォームを持ちます:

```
SELECT `first_name`, `last_name` FROM `customer` WHERE `customer_id` = ?
```

正規化を使用することで、ファイアウォールは、クライアントから受信した様々なステートメントにそれぞれ一致する許可ダイジェストに格納できます。正規化およびダイジェストの詳細は、[セクション27.10「パフォーマンススキーマのステートメントダイジェストとサンプリング」](#)を参照してください。

ファイアウォールに登録された各プロファイルには、次の値から選択された独自の操作モードがあります:

- **OFF**: このモードでは、プロファイルが無効になります。ファイアウォールは非アクティブとみなし、無視します。

- **RECORDING**: これはファイアウォールトレーニングモードです。プロファイルと一致するクライアントから受信した受信ステートメントは、プロファイルで受け入れ可能とみなされ、その「指紋」の一部になります。ファイアウォールは、各ステートメントの正規化されたダイジェスト形式を記録して、プロファイルの受け入れ可能なステートメントパターンを学習します。各パターンはルールであり、ルールの和集合はプロファイル allowlist です。

アカウントプロファイルとグループプロファイルの違いは、グループプロファイルのステートメントの記録を単一のグループメンバーから受け取ったステートメントに制限できることです。

- **PROTECTING**: このモードでは、プロファイルによってステートメントの実行が許可または防止されます。ファイアウォールは、受信ステートメントをプロファイル allowlist と照合し、一致しないステートメントのみを受け入れて拒否します。RECORDING モードでプロファイルをトレーニングした後、PROTECTING モードに切り替えて、許可リストから逸脱するステートメントによるアクセスに対して MySQL を強化します。
- **DETECTING**: このモードでは、侵入 (プロファイル allowlist 内で何にも一致しないため疑わしいステートメント) は検出されますが、ブロックされません。DETECTING モードでは、ファイアウォールは疑わしいステートメントをエラーログに書き込みますが、アクセスを拒否せずに受け入れます。

プロファイルに前述のモード値のいずれかが割り当てられると、ファイアウォールはそのモードをプロファイルに格納します。ファイアウォールモード設定操作では RESET のモード値も許可されますが、この値は格納されません: プロファイルを RESET モードに設定すると、ファイアウォールはプロファイルのすべてのルールを削除し、そのモードを OFF に設定します。

次のセクションでは、ファイアウォールプロファイルの使用方法について説明します。簡単にするために、この説明では単一のアカウントプロファイルと単一のグループプロファイルの設定について説明し、複数のプロファイルが同時に適用された場合にファイアウォールがどのように動作するかについて、より複雑なケースに移動します。

ファイアウォールアカウントプロファイルの登録

MySQL Enterprise Firewall では、個々のアカウントに対応するプロファイルを登録できます。

MySQL は、特定のユーザー名とホスト名の組合せに対して各クライアントセッションを認証します。この組合せはセッションアカウントです。ファイアウォールは、セッションアカウントを登録済アカウントプロファイルと照合して、セッションからの受信ステートメントの処理に適用されるプロファイルを決めます:

- ファイアウォールは非アクティブなプロファイル (OFF モードのプロファイル) を無視します。
- セッションアカウントは、同じユーザーとホストを持つアクティブなアカウントプロファイルと一致します (存在する場合)。そのようなアカウントプロファイルは最大 1 つあります。

つまり、特定のセッションには最大 1 つのアクティブなアカウントプロファイルを適用でき、ファイアウォールは各受信ステートメントを次のように処理します:

- 適用可能なプロファイルがない場合、制限はありません。ファイアウォールはステートメントを受け入れます。
- 適用可能なプロファイルがある場合、そのモードによってステートメントの処理が決定されます:
 - **RECORDING** モードでは、ファイアウォールはプロファイル allowlist ルールにステートメントを追加し、それを受け入れます。
 - **PROTECTING** モードでは、ファイアウォールはステートメントをプロファイル allowlist のルールと比較します。ファイアウォールは、一致がある場合はステートメントを受け入れ、それ以外の場合は拒否します。mysql_firewall_trace システム変数が有効になっている場合、ファイアウォールは拒否されたステートメントもエラーログに書き込みます。
 - **DETECTING** モードでは、ファイアウォールはアクセスを拒否せずに侵入を検出します。ファイアウォールはステートメントを受け入れますが、PROTECTING モードと同様にプロファイル allowlist にも一致します。ステートメントが疑わしい (不一致) 場合、ファイアウォールはそれをエラーログに書き込みます。

注記

この説明は、セッションアカウントが 1 つ以上の適用可能なアカウントプロファイルと一致することを前提としているため、簡略化されています。複数プロファイルの場合については、[複数の適用可能なプロファイルに対するファイアウォール操作](#) を参照してください。

ファイアウォールアカウントプロファイルを使用して、特定のアカウントからの受信ステートメントから MySQL を保護するには、次のステップに従います:

1. アカウントプロファイルを登録し、**RECORDING** モードにします。
2. アカウントを使用して MySQL サーバーに接続し、学習するステートメントを実行します。これにより、対応するアカウントプロファイルがトレーニングされ、プロファイル allowlist を形成するルールが設定されます。
3. アカウントプロファイルを **PROTECTING** モードに切り替えます。クライアントがアカウントを使用してサーバーに接続すると、アカウントプロファイル allowlist によってステートメントの実行が制限されます。
4. 追加のトレーニングが必要な場合は、アカウントプロファイルを再度 **RECORDING** モードに切り替え、新しいステートメントパターンで許可リストを更新してから、**PROTECTING** モードに戻します。

プロファイルを維持することで、ファイアウォールは次のような保護戦略の実装を可能にします:

- アプリケーションに一意の保護要件がある場合は、他の目的に使用されていないアカウントを使用するように構成し、対応するアカウントプロファイルを設定します。
- 関連アプリケーションで保護要件を共有する場合は、すべて同じアカウント (および同じアカウントプロファイル) を使用するように構成します。

または、各アプリケーションを独自のアカウントに関連付けてから、これらのアカウントを同じグループプロファイルに追加します。 [ファイアウォールグループプロファイルの登録](#) を参照してください。

ファイアウォール関連のアカウント参照については、次のガイドラインに従います:

- アカウント参照が発生するコンテキストに注意してください。ファイアウォール操作のアカウントに名前を付けるには、一重引用符で囲まれた文字列 ('user_name@host_name') として指定します。これは、アカウント名のユーザー部分とホスト部分を別々に引用符で囲む **CREATE USER** や **GRANT** などのステートメントの通常の MySQL 規則とは異なります ('user_name'@'host_name')。

ファイアウォール操作のために一重引用符で囲まれた文字列としてアカウントに名前を付ける要件は、ユーザー名に @ 文字が埋め込まれているアカウントを使用できないことを意味します。

- ファイアウォールは、サーバーによって認証された実際のユーザー名とホスト名で表されるアカウントに対してステートメントを評価します。アカウントプロファイルを登録する場合は、ワイルドカード文字またはネットマスクを使用しないでください:
 - `me@%.example.org` という名前のアカウントが存在し、クライアントがそれを使用してホスト `abc.example.org` からサーバーに接続するとします。
 - アカウント名には `%` ワイルドカード文字が含まれていますが、サーバーは、ユーザー名が `me` でホスト名が `abc.example.com` であること、つまりファイアウォールに表示されることとしてクライアントを認証します。
 - したがって、ファイアウォール操作に使用するアカウント名は、`me@%.example.org` ではなく `me@abc.example.org` です。

次の例では、アカウントプロファイルをファイアウォールに登録し、そのプロファイルの受入れ可能なステートメントをファイアウォールに指示し、そのプロファイルを使用して、アカウントによる受入れ不可能なステートメントの実行から MySQL を保護する方法を示します。サンプルアカウント `fwuser@localhost` は、`sakila` データベース (<https://dev.mysql.com/doc/index-other.html> で入手可能) のテーブルにアクセスするアプリケーションで使用することを前提としています。

ファイアウォールに登録されたアカウントプロファイルに対応する `fwuser@localhost` アカウントで実行するように指定されたステップを除き、管理 MySQL アカウントを使用してこの手順のステップを実行します。このアカウントを使用して実行されるステートメントの場合、デフォルトのデータベースは `sakila` である必要があります。(指示を適宜調整することで、別のデータベースを使用できます。)

1. 必要に応じて、ステートメントの実行に使用するアカウントを作成し (適切なパスワードを選択)、`sakila` データベースに対する権限を付与します:

```
CREATE USER 'fwuser'@'localhost' IDENTIFIED BY 'password';
```

```
GRANT ALL ON sakila.* TO 'fwuser'@'localhost';
```

2. `sp_set_firewall_mode()` ストアドプロシージャを使用して、アカウントプロファイルをファイアウォールに登録し、プロファイルを **RECORDING** (トレーニング) モードにします:

```
CALL mysql.sp_set_firewall_mode('fwuser@localhost', 'RECORDING');
```

ストアドプロシージャは、実行時にファイアウォールのユーザー定義関数を起動し、独自の出力を生成できます。

3. 登録済アカウントプロファイルをトレーニングするには、ファイアウォールに `fwuser@localhost` のセッションアカウントが表示されるように、サーバーホストから `fwuser` としてサーバーに接続します。次に、アカウントを使用して、プロファイルに対して正当とみなされるいくつかのステートメントを実行します。例:

```
SELECT first_name, last_name FROM customer WHERE customer_id = 1;
UPDATE rental SET return_date = NOW() WHERE rental_id = 1;
SELECT get_customer_balance(1, NOW());
```

プロファイルは **RECORDING** モードであるため、ファイアウォールは正規化されたダイジェスト形式のステートメントをルールとしてプロファイル `allowlist` に記録します。

注記

`fwuser@localhost` アカウントプロファイルが **RECORDING** モードでステートメントを受信するまで、その `allowlist` は空であり、これは「すべて拒否」と同等です。空の `allowlist` に一致するステートメントはないため、次のような影響があります:

- アカウントプロファイルを **PROTECTING** モードに切り替えることはできません。すべてのステートメントが拒否され、アカウントによるステートメントの実行が事実上禁止されます。
- アカウントプロファイルは **DETECTING** モードに切り替えることができます。この場合、プロファイルはすべてのステートメントを受け入れますが、疑わしいステートメントとして記録します。

4. この時点で、アカウントプロファイル情報がキャッシュされます。この情報を表示するには、ファイアウォール **INFORMATION_SCHEMA** テーブルをクエリーします:

```
mysql> SELECT MODE FROM INFORMATION_SCHEMA.MYSQL_FIREWALL_USERS
      WHERE USERHOST = 'fwuser@localhost';
+-----+
| MODE |
+-----+
| RECORDING |
+-----+
mysql> SELECT RULE FROM INFORMATION_SCHEMA.MYSQL_FIREWALL_WHITELIST
      WHERE USERHOST = 'fwuser@localhost';
+-----+
| RULE |
+-----+
| SELECT `first_name` , `last_name` FROM `customer` WHERE `customer_id` = ? |
| SELECT `get_customer_balance` ( ? , NOW ( ) ) |
| UPDATE `rental` SET `return_date` = NOW ( ) WHERE `rental_id` = ? |
| SELECT @@`version_comment` LIMIT ? |
+-----+
```

注記

`@@version_comment` ルールは、アカウントプロファイルに対応するアカウントを使用してサーバーに接続したときに、`mysql` クライアントによって自動的に送信されるステートメントから取得されます。

重要

アプリケーションの使用に一致する条件でファイアウォールをトレーニングします。たとえば、サーバーの特性と機能を判断するために、特定の MySQL コネクタが各セッションの開始時にサーバーにステートメントを送信する場合があります。アプリケーションが通常そのコネクタを介して使用される場合は、コネクタを使用してファイア

ウォールもトレーニングします。これにより、これらの初期ステートメントを、アプリケーションに関連付けられたアカウントプロファイルの許可リストの一部にすることができます。

5. `sp_set_firewall_mode()` を再度起動します。今回は、アカウントプロファイルを **PROTECTING** モードに切り替えます:

```
CALL mysql.sp_set_firewall_mode('fwuser@localhost', 'PROTECTING');
```

重要

アカウントプロファイルを **RECORDING** モードから切り替えると、キャッシュされたデータが、基礎となる永続的な記憶域を提供する `mysql` システムデータベーステーブルと同期されます。記録されているプロファイルのモードを切り替えない場合、キャッシュされたデータは永続ストレージに書き込まれず、サーバーの再起動時に失われます。

6. アカウントを使用して、許容できるステートメントと許容できないステートメントを実行し、アカウントプロファイル进行测试します。ファイアウォールは、各ステートメントをプロファイル `allowlist` と照合し、それを受け入れるか拒否します:

- このステートメントはトレーニングステートメントと同じではありませんが、いずれかのステートメントと同じ正規化ステートメントを生成するため、ファイアウォールはそれを受け入れます:

```
mysql> SELECT first_name, last_name FROM customer WHERE customer_id = '48';
+-----+-----+
| first_name | last_name |
+-----+-----+
| ANN      | EVANS    |
+-----+-----+
```

- これらのステートメントは `allowlist` 内の何も一致しないため、ファイアウォールはそれぞれエラーで拒否します:

```
mysql> SELECT first_name, last_name FROM customer WHERE customer_id = 1 OR TRUE;
ERROR 1045 (28000): Statement was blocked by Firewall
mysql> SHOW TABLES LIKE 'customer%';
ERROR 1045 (28000): Statement was blocked by Firewall
mysql> TRUNCATE TABLE mysql.slow_log;
ERROR 1045 (28000): Statement was blocked by Firewall
```

- `mysql_firewall_trace` システム変数が有効になっている場合、ファイアウォールは拒否されたステートメントもエラーログに書き込みます。例:

```
[Note] Plugin MYSQL_FIREWALL reported:
'ACCESS DENIED for fwuser@localhost. Reason: No match in whitelist.
Statement: TRUNCATE TABLE `mysql`.`slow_log`'
```

これらのログメッセージは、必要に応じて攻撃の原因を特定する際に役立ちます。

ファイアウォールアカウントプロファイルは、`fwuser@localhost` アカウントについてトレーニングされるようになりました。クライアントがそのアカウントを使用して接続し、ステートメントを実行しようとする、プロファイルによって、プロファイル `allowlist` で一致しないステートメントから MySQL が保護されます。

一致しないステートメントをアクセスを拒否せずに疑わしいステートメントとして記録することで、侵入を検出することもできます。まず、アカウントプロファイルを **DETECTING** モードにします:

```
CALL mysql.sp_set_firewall_mode('fwuser@localhost', 'DETECTING');
```

次に、アカウントを使用して、アカウントプロファイル `allowlist` と一致しないステートメントを実行します。**DETECTING** モードでは、ファイアウォールは一致しないステートメントの実行を許可します:

```
mysql> SHOW TABLES LIKE 'customer%';
+-----+
| Tables_in_sakila (customer%) |
+-----+
| customer                      |
| customer_list                 |
+-----+
```

+-----+

さらに、ファイアウォールはエラーログにメッセージを書き込みます:

```
[Note] Plugin MYSQL_FIREWALL reported:
'SUSPICIOUS STATEMENT from 'fwuser@localhost'. Reason: No match in whitelist.
Statement: SHOW TABLES LIKE ?'
```

注記

DETECTING モードでは、メッセージはノート (情報メッセージ) として書き込まれます。このようなメッセージがエラーログに表示され、破棄されないようにするには、情報メッセージを記録するためにエラーロギングの冗長性が十分であることを確認します。たとえば、[セクション5.4.2.5「優先度ベースのエラーログのフィルタリング \(log_filter_internal\)」](#) で説明されている優先度ベースのログフィルタリングを使用している場合は、`log_error_verbosity` システム変数の値を 3 に設定します。

アカウントプロファイルを無効にするには、そのモードを **OFF** に変更します:

```
CALL mysql.sp_set_firewall_mode(user, 'OFF');
```

プロファイルのすべてのトレーニングを忘れて無効にするには、リセットします:

```
CALL mysql.sp_set_firewall_mode(user, 'RESET');
```

リセット操作により、ファイアウォールはプロファイルのすべてのルールを削除し、そのモードを **OFF** に設定します。

ファイアウォールグループプロファイルの登録

MySQL Enterprise Firewall では、個々のアカウントに対応するサポートプロファイルに加えて、MySQL 8.0.23 の時点でグループプロファイルがサポートされています。グループプロファイルは、そのメンバーとして複数のアカウントを持つことができます。

グループプロファイルは、次の点でアカウントプロファイルと異なります:

- グループプロファイルの場合、その許可リストは、セッションアカウントがグループのいずれかのメンバーと一致するときに適用されます。グループプロファイルを使用すると、アカウントごとに許可リスト内の各ルールを複製せずに、特定の許可リストを複数のアカウントに適用できます。
- アカウントプロファイルは、それが適用される単一のアカウントを使用してのみトレーニングできます。グループプロファイルは、グループメンバーアカウントのいずれかまたはすべてを使用してトレーニングすることも、それらのアカウントのいずれかに限定することもできます。
- アカウントプロファイル名は、MySQL サーバーに接続するクライアントに依存する特定のユーザー名とホスト名の組合せに基づきます。グループプロファイル名は、長さが 1 から 288 文字である必要があるという制約なしで選択されます。

その他の点では、アカウントプロファイルの使用に関する原則は、グループプロファイルの使用にも適用されます。ここでは、これらの原則をよく理解していることを前提としています。[ファイアウォールアカウントプロファイルの登録](#) を参照してください。

注記

この説明は、セッションアカウントが 1 つ以上の適用可能なグループプロファイルと一致することを前提としているため、簡略化されています。複数プロファイルの場合については、[複数の適用可能なプロファイルに対するファイアウォール操作](#) を参照してください。

次の例では、グループプロファイルをファイアウォールに登録し、その許可リストをトレーニングし、それを使用して MySQL を許容できないステートメントの実行から保護し、メンバーを追加および削除する方法を示します。この例では、`mygrp` のグループプロファイル名を使用します。

管理 MySQL アカウントを使用して、この手順の手順を実行します。ただし、ファイアウォールグループプロファイルのメンバーアカウントによって実行されるように指定された手順は除きます。

- 必要に応じて、`mygrp` グループプロファイルのメンバーとなるアカウントを作成し、適切なアクセス権限を付与します。1 つのメンバーに対するステートメントを次に示します (適切なパスワードを選択してください):


```
CREATE USER 'member1'@'localhost' IDENTIFIED BY 'password';
GRANT ALL ON sakila.* TO 'member1'@'localhost';
```

2. `sp_set_firewall_group_mode()` ストアドプロシージャを使用して、グループプロファイルをファイアウォールに登録し、そのプロファイルを **RECORDING** (トレーニング) モードにします:

```
CALL mysql.sp_set_firewall_group_mode('mygrp', 'RECORDING');
```

3. `sp_firewall_group_enlist()` ストアドプロシージャを使用して、グループプロファイル許可リストのトレーニングに使用する初期メンバーアカウントを追加します:

```
CALL mysql.sp_firewall_group_enlist('mygrp', 'member1@localhost');
```

4. グループプロファイルのトレーニングに初期メンバーアカウントを使用するには、サーバーホストから `member1` としてサーバーに接続し、ファイアウォールに `member1@localhost` のセッションアカウントが表示されるようにします。次に、プロファイルに対して正当とみなされるいくつかのステートメントを実行します。例:

```
SELECT title, description FROM film WHERE film_id = 1;
UPDATE actor SET last_update = NOW() WHERE actor_id = 1;
SELECT store_id, COUNT(*) FROM inventory GROUP BY store_id;
```

ファイアウォールは、`member1@localhost` アカウントからステートメントを受け取ります。このアカウントは **RECORDING** モードの `mygrp` プロファイルのメンバーであるため、ファイアウォールはステートメントを `mygrp` に適用可能として解釈し、正規化されたダイジェスト形式のステートメントを `mygrp` 許可リスト内のルールとして記録します。これらのルールは、`mygrp` のメンバーであるすべてのアカウントに適用されます。

5. この時点で、名前、メンバーシップ、許可リストなどのグループプロファイル情報がキャッシュされます。この情報を表示するには、ファイアウォールの「パフォーマンススキーマ」テーブルをクエリーします:

```
mysql> SELECT MODE FROM performance_schema.firewall_groups
      WHERE NAME = 'mygrp';
+-----+
| MODE |
+-----+
| RECORDING |
+-----+
mysql> SELECT * FROM performance_schema.firewall_membership
      ORDER BY USERHOST;
+-----+-----+
| GROUP_NAME | USERHOST |
+-----+-----+
| mygrp      | member1@localhost |
+-----+-----+
mysql> SELECT RULE FROM performance_schema.firewall_group_allowlist
      WHERE NAME = 'mygrp';
+-----+
| RULE |
+-----+
| SELECT @@`version_comment` LIMIT ? |
| SELECT `title`, DESCRIPTION FROM `film` WHERE `film_id` = ? |
| UPDATE `actor` SET `last_update` = NOW () WHERE `actor_id` = ? |
| SELECT `store_id`, COUNT ( * ) FROM `inventory` GROUP BY `store_id` |
+-----+
```

6. `sp_set_firewall_group_mode()` を再度起動して、グループプロファイルを **PROTECTING** モードに切り替えます:

```
CALL mysql.sp_set_firewall_group_mode('mygrp', 'PROTECTING');
```

7. メンバーである必要がある他のアカウントをグループプロファイルに追加します:

```
CALL mysql.sp_firewall_group_enlist('mygrp', 'member2@localhost');
CALL mysql.sp_firewall_group_enlist('mygrp', 'member3@localhost');
CALL mysql.sp_firewall_group_enlist('mygrp', 'member4@localhost');
```

`member1@localhost` アカウントを使用してトレーニングされたプロファイル許可リストは、追加アカウントにも適用されるようになりました。

8. 更新されたグループメンバーシップを確認するには、`firewall_membership` テーブルを再度クエリーします:

```
mysql> SELECT * FROM performance_schema.firewall_membership
```



```
ORDER BY USERHOST;
+-----+-----+
| GROUP_NAME | USERHOST |
+-----+-----+
| mygrp      | member1@localhost |
| mygrp      | member2@localhost |
| mygrp      | member3@localhost |
| mygrp      | member4@localhost |
+-----+-----+
```

- グループ内の任意のアカウントを使用してファイアウォールに対してグループプロファイルをテストし、受け入れ可能なステートメントと受け入れられないステートメントを実行します。ファイアウォールは、アカウントからの各ステートメントをプロファイル許可リストと照合し、それを受け入れるか拒否します。
- グループプロファイルからメンバーを削除する必要がある場合は、`sp_firewall_group_enlist()` ではなく `sp_firewall_group_delist()` ストアドプロシージャを使用します:

```
CALL mysql.sp_firewall_group_delist('mygrp', 'member3@localhost');
```

前述の手順では、許可リストをトレーニングする前に、1人のメンバーのみをグループプロファイルに追加しました。こうすることで、新規の許容ステートメントを許可リストに追加できるアカウントを制限することで、研修期間をより適切に制御できます。追加のトレーニングが必要な場合は、プロファイルを **RECORDING** モードに戻すことができます:

```
CALL mysql.sp_set_firewall_group_mode('mygrp', 'RECORDING');
```

ただし、グループの任意のメンバーがステートメントを実行して許可リストに追加できるようになります。追加のトレーニングを単一のグループメンバーに制限するには、`sp_set_firewall_group_mode_and_user()` をコールします。これは `sp_set_firewall_group_mode()` に似ていますが、**RECORDING** モードでプロファイルのトレーニングを許可するアカウントを指定する引数を使用します。たとえば、`member4@localhost` によるトレーニングのみを有効にするには、次のようにします:

```
CALL mysql.sp_set_firewall_group_mode_and_user('mygrp', 'RECORDING', 'member4@localhost');
```

これにより、他のグループメンバーを削除せずに、指定したアカウントによる追加のトレーニングが可能になります。ステートメントを実行できますが、そのステートメントは許可リストに追加されません。(ただし、**RECORDING** モードでは、他のメンバーが任意のステートメントを実行できることに注意してください。)

追加のトレーニングの後、グループプロファイルを **PROTECTING** モードに戻します:

```
CALL mysql.sp_set_firewall_group_mode('mygrp', 'PROTECTING');
```

`sp_set_firewall_group_mode_and_user()` によって確立されたトレーニングアカウントはグループプロファイルに保存されるため、後でさらにトレーニングが必要になった場合に備えて、ファイアウォールに保存されます。したがって、`sp_set_firewall_group_mode()` (トレーニングアカウント引数なし) をコールした場合、現在のプロファイルトレーニングアカウントである `member4@localhost` は変更されません。

すべてのグループメンバーが **RECORDING** モードでトレーニングを実行できるようにする必要がある場合にトレーニングアカウントをクリアするには、`sp_set_firewall_group_mode_and_user()` をコールし、アカウント引数に **NULL** 値を渡します:

```
CALL mysql.sp_set_firewall_group_mode_and_user('mygrp', 'RECORDING', NULL);
```

注記

特定のアカウントがグループプロファイルのトレーニングアカウントとして指定された場合の予期しない動作を回避するには、そのアカウントがグループのメンバーであることを常に確認してください。

アカウントプロファイルと同様に、侵入検出の場合は **DETECTING** のモード、プロファイルを無効にする場合は **OFF**、プロファイルルールを忘れてそのモードを **OFF** に設定する場合は **RESET** のモードをグループプロファイルに割り当てることができます。

複数の適用可能なプロファイルに対するファイアウォール操作

簡単にするために、前のセクションでは、ファイアウォールがクライアントからの受信ステートメントを単一のプロファイル (アカウントプロファイルまたはグループプロファイル) に対してのみ照合する観点を説明しました。ただし、ファイアウォール操作はより複雑になる可能性があります:

- グループプロファイルには、複数のアカウントをメンバーとして含めることができます。
- アカウントは、複数のグループプロファイルのメンバーになることができます。
- 複数のプロファイルを特定のクライアントに一致させることができます。

このセクションでは、複数のプロファイルが受信ステートメントに適用される場合のファイアウォールの動作について説明します。この説明では、単一の適用可能なアカウントまたはグループプロファイルで前述したケースを含む、一般的なケースについて説明します。

MySQL は、特定のユーザー名とホスト名の組合せに対して各クライアントセッションを認証します。この組合せはセッションアカウントです。ファイアウォールは、セッションアカウントを登録済プロファイルと照合して、セッションからの受信ステートメントの処理に適用するプロファイルを決定します:

- ファイアウォールは非アクティブなプロファイル (OFF モードのプロファイル) を無視します。
- セッションアカウントは、同じユーザーとホストを持つアクティブなアカウントプロファイルと一致します (存在する場合)。そのようなアカウントプロファイルは最大 1 つあります。
- セッションアカウントは、同じユーザーおよびホストを持つメンバーを含むすべてのアクティブなグループプロファイルと一致します。そのようなグループプロファイルは複数存在できます。

つまり、セッションアカウントは、0 個または 1 個のアクティブなアカウントプロファイルと、0 個以上のアクティブなグループプロファイルと一致します。つまり、0、1 または複数のファイアウォールプロファイルが特定のセッションに適用可能であり、ファイアウォールが各受信ステートメントを次のように処理することを意味します:

- 適用可能なプロファイルがない場合、制限はありません。ファイアウォールはステートメントを受け入れます。
- 適用可能なプロファイルがある場合は、そのモードによってステートメントの処理が決まります:
 - ファイアウォールは、**RECORDING** モードの適用可能な各プロファイルの許可リストにステートメントを記録します。
 - ファイアウォールは、**DETECTING** モードの適用可能な各プロファイルのエラーログにステートメントを書き込みます。
 - 少なくとも 1 つの適用可能なプロファイルが **RECORDING** モードまたは **DETECTING** モードの場合 (これらのモードはすべてのステートメントを受け入れます)、あるいは少なくとも 1 つの適用可能なプロファイルの許可リストと **PROTECTING** モードで一致する場合、ファイアウォールはそのステートメントを受け入れます。そうでない場合、ファイアウォールはステートメントを拒否します (`mysql_firewall_trace` システム変数が有効になっている場合はエラーログに書き込みます)。

ファイアウォールの監視

ファイアウォールアクティビティを評価するには、そのステータス変数を調べます。たとえば、前述の手順を実行して `fwuser@localhost` アカウントをトレーニングおよび保護すると、変数は次のようになります:

```
mysql> SHOW GLOBAL STATUS LIKE 'Firewall%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Firewall_access_denied | 3 |
| Firewall_access_granted | 4 |
| Firewall_access_suspicious | 1 |
| Firewall_cached_entries | 4 |
+-----+-----+
```

変数は、拒否されたステートメント、受け入れられたステートメント、疑わしいステートメントとして記録されたステートメントおよびキャッシュに追加されたステートメントの数をそれぞれ示します。登録済アカウントを使用して

接続した 3 回ごとに `mysql` クライアントによって送信される `@@version_comment` ステートメントと `DETECTING` モードでブロックされなかった `SHOW TABLES` ステートメントのため、`Firewall_access_granted` 数は 4 です。

6.4.7.4 MySQL Enterprise Firewall リファレンス

次の各セクションでは、MySQL Enterprise Firewall 要素のリファレンスを示します:

- [MySQL Enterprise Firewall テーブル](#)
- [MySQL Enterprise Firewall ストアドプロシージャ](#)
- [MySQL Enterprise Firewall のユーザー定義関数](#)
- [MySQL Enterprise Firewall システム変数](#)
- [MySQL Enterprise Firewall ステータス変数](#)

MySQL Enterprise Firewall テーブル

MySQL Enterprise Firewall では、アカウントごとおよびグループごとにプロファイル情報が保持されます。 `mysql` システムデータベース内のテーブルを永続記憶域に使用し、`INFORMATION_SCHEMA` テーブルまたは「パフォーマンススキーマ」テーブルを使用してメモリー内にキャッシュされたデータを表示します。有効にすると、ファイアウォールはキャッシュされたデータに基づいて操作上の決定を行います。

- [ファイアウォールアカウントプロファイルテーブル](#)
- [ファイアウォールグループプロファイルテーブル](#)

ファイアウォールアカウントプロファイルテーブル

MySQL Enterprise Firewall では、永続記憶域の `mysql` システムデータベースのテーブルおよび `INFORMATION_SCHEMA` テーブルを使用してアカウントプロファイル情報を保持し、メモリー内にキャッシュされたデータのビューを提供します。

各 `mysql` システムデータベーステーブルにアクセスできるのは、そのテーブルに対する `SELECT` 権限を持つアカウントのみです。 `INFORMATION_SCHEMA` テーブルには、誰でもアクセスできます。

`mysql.firewall_users` テーブルには、登録済ファイアウォールアカウントプロファイルの名前と操作モードがリストされます。テーブルには次のカラムがあります (対応する `INFORMATION_SCHEMA.MYSQL_FIREWALL_USERS` テーブルには類似したカラムがありますが、必ずしも同一ではありません):

- `USERHOST`

アカウントプロファイル名。各アカウント名の形式は `user_name@host_name` です。

- `MODE`

プロファイルの現在の操作モード。許可されるモード値は、`OFF`、`DETECTING`、`PROTECTING`、`RECORDING` および `RESET` です。意味の詳細は、[ファイアウォール操作の概念](#) を参照してください。

`mysql.firewall_whitelist` テーブルには、登録済ファイアウォールアカウントプロファイルの許可リストルールがリストされます。テーブルには次のカラムがあります (対応する `INFORMATION_SCHEMA.MYSQL_FIREWALL_WHITELIST` テーブルには類似したカラムがありますが、必ずしも同一ではありません):

- `USERHOST`

アカウントプロファイル名。各アカウント名の形式は `user_name@host_name` です。

- `RULE`

プロファイルの許容可能なステートメントパターンを示す正規化されたステートメント。プロファイル許可リストは、そのルールの和集合です。

- ID

テーブルの主キーである整数カラム。このカラムは、MySQL 8.0.12 で追加されました。

ファイアウォールグループプロファイルテーブル

MySQL 8.0.23 の時点では、MySQL Enterprise Firewall は、永続記憶域用の `mysql` システムデータベースのテーブルおよび「パフォーマンススキーマ」テーブルを使用してグループプロファイル情報を保持し、メモリー内にキャッシュされたデータへのビューを提供します。

各システムおよび「パフォーマンススキーマ」テーブルには、そのシステムおよび「パフォーマンススキーマ」テーブルに対する `SELECT` 権限を持つアカウントのみがアクセスできます。

`mysql.firewall_groups` テーブルには、登録されているファイアウォールグループプロファイルの名前と動作モードが一覧表示されます。テーブルには次のカラムがあります (対応するパフォーマンススキーマ `firewall_groups` テーブルには類似していますが、必ずしも同一ではありません):

- NAME

グループプロファイル名。

- MODE

プロファイルの現在の操作モード。許可されるモード値は、`OFF`、`DETECTING`、`PROTECTING` および `RECORDING` です。意味の詳細は、[ファイアウォール操作の概念](#) を参照してください。

- USERHOST

プロファイルが `RECORDING` モードの場合に使用される、グループプロファイルのトレーニングアカウント。値は、`NULL` または `user_name@host_name` 形式の `NULL` 以外のアカウントです:

- 値が `NULL` の場合、ファイアウォールはグループのメンバーであるアカウントから受信したステートメントの許可リストルールを記録します。
- 値が `NULL` 以外の場合、ファイアウォールは、指定されたアカウント (グループのメンバーである必要があります) から受信したステートメントの許可リストルールのみを記録します。

`mysql.firewall_group_allowlist` テーブルには、登録されているファイアウォールグループプロファイルの許可リストルールが一覧表示されます。テーブルには次のカラムがあります (対応するパフォーマンススキーマ `firewall_group_allowlist` テーブルには類似していますが、必ずしも同一ではありません):

- NAME

グループプロファイル名。

- RULE

プロファイルの許容可能なステートメントパターンを示す正規化されたステートメント。プロファイル許可リストは、そのルールの和集合です。

- ID

テーブルの主キーである整数カラム。

`mysql.firewall_membership` テーブルには、登録済のファイアウォールグループプロファイルのメンバー (アカウント) がリストされます。テーブルには次のカラムがあります (対応するパフォーマンススキーマ `firewall_membership` テーブルには類似していますが、必ずしも同一ではありません):

- GROUP_ID

グループプロファイル名。

- MEMBER_ID

プロファイルのメンバーであるアカウントの名前。

MySQL Enterprise Firewall ストアドプロシージャ

MySQL Enterprise Firewall ストアドプロシージャは、ファイアウォールへのプロファイルの登録、操作モードの確立、キャッシュと永続記憶域間のファイアウォールデータの転送の管理などのタスクを実行します。これらのプロシージャは、下位レベルのタスク用の API を提供するユーザー定義関数 (UDF) を起動します。

ファイアウォールストアドプロシージャは、`mysql` システムデータベースに作成されます。ファイアウォールストアドプロシージャを起動するには、`mysql` がデフォルトのデータベースである間に起動するか、プロシージャ名をデータベース名で修飾します。例:

```
CALL mysql.sp_set_firewall_mode(user, mode);
```

- [ファイアウォールアカウントプロファイルのストアドプロシージャ](#)
- [ファイアウォールグループプロファイルのストアドプロシージャ](#)

ファイアウォールアカウントプロファイルのストアドプロシージャ

次のストアドプロシージャは、ファイアウォールアカウントプロファイルに対して管理操作を実行します:

- `sp_reload_firewall_rules(user)`

このストアドプロシージャは、個々のアカウントプロファイルのファイアウォール操作を制御します。この手順では、ファイアウォール UDF を使用して、`mysql.firewall_whitelist` テーブルに格納されているルールからアカウントプロファイルのメモリー内ルールをリロードします。

引数:

- `user`: 影響を受けるアカウントプロファイルの名前 (`user_name@host_name` 形式の文字列)。

例:

```
CALL mysql.sp_reload_firewall_rules('fwuser@localhost');
```

警告

このプロシージャは、永続ストレージからリロードする前にアカウントプロファイルのインメモリー許可リストルールをクリアし、プロファイルモードを `OFF` に設定します。 `sp_reload_firewall_rules()` コールの前にプロファイルモードが `OFF` でなかった場合は、ルールのリロード後に `sp_set_firewall_mode()` を使用して以前のモードをリストアします。たとえば、プロファイルが `PROTECTING` モードであった場合、`sp_reload_firewall_rules()` をコールした後で `true` でなくなり、それを明示的に `PROTECTING` に再度設定する必要があります。

- `sp_set_firewall_mode(user, mode)`

このストアドプロシージャは、ファイアウォールにプロファイルを登録した後 (まだ登録されていない場合)、ファイアウォールアカウントプロファイルの操作モードを確立します。この手順では、必要に応じてファイアウォール UDF を呼び出して、キャッシュと永続ストレージの間でファイアウォールデータを転送することもできます。このプロシージャは、`mysql_firewall_mode` システム変数が `OFF` の場合でもコールできますが、プロファイルのモードを設定しても、ファイアウォールが有効になるまで操作上の影響はありません。

引数:

- `user`: 影響を受けるアカウントプロファイルの名前 (`user_name@host_name` 形式の文字列)。
- `mode`: 文字列としてのプロファイルの操作モード。許可されるモード値は、`OFF`、`DETECTING`、`PROTECTING`、`RECORDING` および `RESET` です。意味の詳細は、[ファイアウォール操作の概念](#) を参照してください。

アカウントプロファイルを任意のモードに切り替えますが、`RECORDING` はファイアウォールキャッシュデータを、基礎となる永続的な記憶域を提供する `mysql` システムデータベーステーブルと同期します。モードを `OFF` から `RECORDING` に切り替えると、`mysql.firewall_whitelist` テーブルからキャッシュに allowlist がリロードされます。

アカウントプロファイルに空の許可リストがある場合、そのモードは **PROTECTING** に設定できません。これは、プロファイルがすべてのステートメントを拒否し、効果的にアカウントによるステートメントの実行を禁止するためです。このようなモード設定の試行に応じて、ファイアウォールは SQL エラーとしてではなく結果セットとして返される診断メッセージを生成します:

```
mysql> CALL mysql.sp_set_firewall_mode('a@b','PROTECTING');
+-----+
| set_firewall_mode(arg_userhost, arg_mode) |
+-----+
| ERROR: PROTECTING mode requested for a@b but the whitelist is empty. |
+-----+
```

ファイアウォールグループプロファイルのストアドプロシージャ

これらのストアドプロシージャは、ファイアウォールグループプロファイルに対して管理操作を実行します:

- **sp_firewall_group_delist(group, user)**

このストアドプロシージャは、ファイアウォールグループプロファイルからアカウントを削除します。

コールが成功した場合、グループメンバーシップの変更は、インメモリーキャッシュと永続記憶域の両方に対して行われます。

引数:

- **group**: 影響を受けるグループプロファイルの名前。
- **user**: 削除するアカウント (**user_name@host_name** 形式の文字列)。

例:

```
CALL sp_firewall_group_delist('g', 'fwuser@localhost');
```

このプロシージャは、MySQL 8.0.23 で追加されました。

- **sp_firewall_group_enlist(group, user)**

このストアドプロシージャは、ファイアウォールグループプロファイルにアカウントを追加します。アカウントをグループに追加する前に、アカウント自体をファイアウォールに登録する必要はありません。

コールが成功した場合、グループメンバーシップの変更は、インメモリーキャッシュと永続記憶域の両方に対して行われます。

引数:

- **group**: 影響を受けるグループプロファイルの名前。
- **user**: **user_name@host_name** 形式の文字列として追加するアカウント。

例:

```
CALL sp_firewall_group_enlist('g', 'fwuser@localhost');
```

このプロシージャは、MySQL 8.0.23 で追加されました。

- **sp_reload_firewall_group_rules(group)**

このストアドプロシージャは、個々のグループプロファイルのファイアウォール操作を制御します。この手順では、ファイアウォール UDF を使用して、**mysql.firewall_group_allowlist** テーブルに格納されているルールからグループプロファイルのインメモリールールをリロードします。

引数:

- **group**: 影響を受けるグループプロファイルの名前。

例:

```
CALL sp_reload_firewall_group_rules('myapp');
```

警告

このプロシージャは、永続記憶域からリロードする前にグループプロファイルインメモリ許可リストルールをクリアし、プロファイルモードを **OFF** に設定します。`sp_reload_firewall_group_rules()` コールの前にプロファイルモードが **OFF** でなかった場合は、ルールのリロード後に `sp_set_firewall_group_mode()` を使用して以前のモードをリストアします。たとえば、プロファイルが **PROTECTING** モードであった場合、`sp_reload_firewall_group_rules()` をコールした後で **true** でなくなり、それを明示的に **PROTECTING** に再度設定する必要があります。

このプロシージャは、MySQL 8.0.23 で追加されました。

- `sp_set_firewall_group_mode(group, mode)`

このストアードプロシージャは、ファイアウォールにプロファイルを登録した後、ファイアウォールグループプロファイルの動作モードを確立します (まだ登録されていない場合)。この手順では、必要に応じてファイアウォール UDF を呼び出して、キャッシュと永続ストレージの間でファイアウォールデータを転送することもできます。このプロシージャは、`mysql_firewall_mode` システム変数が **OFF** の場合でもコールできますが、プロファイルのモードを設定しても、ファイアウォールが有効になるまで操作上の影響はありません。

プロファイルが以前に存在していた場合、その記録制限は変更されません。制限を設定またはクリアするには、代わりに `sp_set_firewall_group_mode_and_user()` をコールします。

引数:

- `group`: 影響を受けるグループプロファイルの名前。
- `mode`: 文字列としてのプロファイルの操作モード。許可されるモード値は、**OFF**, **DETECTING**, **PROTECTING** および **RECORDING** です。意味の詳細は、[ファイアウォール操作の概念](#) を参照してください。

例:

```
CALL sp_set_firewall_group_mode('myapp', 'PROTECTING');
```

このプロシージャは、MySQL 8.0.23 で追加されました。

- [sp_set_firewall_group_mode_and_user\(group, mode, user\)](#)

このストアードプロシージャは、[sp_set_firewall_group_mode\(\)](#)と同様に、グループをファイアウォールに登録し、その操作モードを確立しますが、グループが **RECORDING** モードの場合に使用するトレーニングアカウントも指定します。

引数:

- **group**: 影響を受けるグループプロファイルの名前。
- **mode**: 文字列としてのプロファイルの操作モード。許可されるモード値は、**OFF**、**DETECTING**、**PROTECTING** および **RECORDING** です。意味の詳細は、[ファイアウォール操作の概念](#) を参照してください。
- **user**: プロファイルが **RECORDING** モードの場合に使用される、グループプロファイルのトレーニングアカウント。値は、**NULL** または **user_name@host_name** 形式の **NULL** 以外のアカウントです:
 - 値が **NULL** の場合、ファイアウォールはグループのメンバーであるアカウントから受信したステートメントの許可リストルールを記録します。
 - 値が **NULL** 以外の場合、ファイアウォールは、指定されたアカウント (グループのメンバーである必要があります) から受信したステートメントの許可リストルールのみを記録します。

例:

```
CALL sp_set_firewall_group_mode_and_user('myapp', 'RECORDING', 'myapp_user1@localhost');
```

このプロシージャは、MySQL 8.0.23 で追加されました。

MySQL Enterprise Firewall のユーザー定義関数

MySQL Enterprise Firewall ユーザー定義関数 (UDF) は、ファイアウォールキャッシュを基礎となるシステムテーブルと同期化するなどの下位レベルのタスク用の API を提供します。

通常の操作では、これらの UDF はユーザーによって直接ではなく、ファイアウォールストアードプロシージャによって起動されます。そのため、これらの UDF の説明には、引数や戻り型に関する情報など、通常の詳細は含まれません。

- [ファイアウォールアカウントプロファイルのユーザー定義関数](#)
- [ファイアウォールグループプロファイルのユーザー定義関数](#)
- [ファイアウォールのその他のユーザー定義関数](#)

ファイアウォールアカウントプロファイルのユーザー定義関数

次の UDF は、ファイアウォールアカウントプロファイルに対する管理操作を実行します:

- [read_firewall_users\(user, mode\)](#)

この集計 UDF は、[mysql.firewall_users](#) テーブルの **SELECT** ステートメントを使用してファイアウォールアカウントプロファイルキャッシュを更新します。 **FIREWALL_ADMIN** 権限または非推奨の **SUPER** 権限が必要です。

例:

```
SELECT read_firewall_users('fwuser@localhost', 'RECORDING')
FROM mysql.firewall_users;
```

- [read_firewall_whitelist\(user, rule\)](#)

この集計 UDF は、[mysql.firewall_whitelist](#) テーブルの **SELECT** ステートメントを使用して、指定されたアカウントプロファイルの記録済ステートメントキャッシュを更新します。 **FIREWALL_ADMIN** 権限または非推奨の **SUPER** 権限が必要です。

例:

```
SELECT read_firewall_whitelist('fwuser@localhost', fw.rule)
```

```
FROM mysql.firewall_whitelist AS fw
WHERE USERHOST = 'fwuser@localhost';
```

- [set_firewall_mode\(user, mode\)](#)

この UDF は、アカウントプロファイルキャッシュを管理し、プロファイル操作モードを確立します。[FIREWALL_ADMIN](#) 権限または非推奨の [SUPER](#) 権限が必要です。

例:

```
SELECT set_firewall_mode('fwuser@localhost', 'RECORDING');
```

ファイアウォールグループプロファイルのユーザー定義関数

これらの UDF は、ファイアウォールグループプロファイルに対して管理操作を実行します:

- [firewall_group_delist\(group, user\)](#)

この UDF は、グループプロファイルからアカウントを削除します。[FIREWALL_ADMIN](#) 権限が必要です。

例:

```
SELECT firewall_group_delist('g', 'fwuser@localhost');
```

この関数は、MySQL 8.0.23 で追加されました。

- [firewall_group_enlist\(group, user\)](#)

この UDF は、アカウントをグループプロファイルに追加します。[FIREWALL_ADMIN](#) 権限が必要です。

アカウントをグループに追加する前に、アカウント自体をファイアウォールに登録する必要はありません。

例:

```
SELECT firewall_group_enlist('g', 'fwuser@localhost');
```

この関数は、MySQL 8.0.23 で追加されました。

- [read_firewall_group_allowlist\(group, rule\)](#)

この集計 UDF は、[mysql.firewall_group_allowlist](#) テーブルの [SELECT](#) ステートメントを介して、指定されたグループプロファイルの記録されたステートメントキャッシュを更新します。[FIREWALL_ADMIN](#) 権限が必要です。

例:

```
SELECT read_firewall_group_allowlist('my_fw_group', fgw.rule)
FROM mysql.firewall_group_allowlist AS fgw
WHERE NAME = 'my_fw_group';
```

この関数は、MySQL 8.0.23 で追加されました。

- [read_firewall_groups\(group, mode, user\)](#)

この集計 UDF は、[mysql.firewall_groups](#) テーブルの [SELECT](#) ステートメントを介してファイアウォールグループプロファイルキャッシュを更新します。[FIREWALL_ADMIN](#) 権限が必要です。

例:

```
SELECT read_firewall_groups('g', 'RECORDING', 'fwuser@localhost')
FROM mysql.firewall_groups;
```

この関数は、MySQL 8.0.23 で追加されました。

- [set_firewall_group_mode\(group, mode\[, user\]\)](#)

この UDF は、グループプロファイルキャッシュを管理し、プロファイル操作モードを確立し、オプションでプロファイルトレーニングアカウントを指定します。[FIREWALL_ADMIN](#) 権限が必要です。

オプションの `user` 引数が指定されていない場合、プロファイルの以前の `user` 設定は変更されません。設定を変更するには、3 番目の引数を指定して UDF を呼び出します。

オプションの `user` 引数が指定されている場合は、プロファイルが `RECORDING` モードのときに使用される、グループプロファイルのトレーニングアカウントを指定します。値は、`NULL` または `user_name@host_name` 形式の `NULL` 以外のアカウントです:

- 値が `NULL` の場合、ファイアウォールはグループのメンバーであるアカウントから受信したステートメントの許可リストルールを記録します。
- 値が `NULL` 以外の場合、ファイアウォールは、指定されたアカウント (グループのメンバーである必要があります) から受信したステートメントの許可リストルールのみを記録します。

例:

```
SELECT set_firewall_group_mode('g', 'DETECTING');
```

この関数は、MySQL 8.0.23 で追加されました。

ファイアウォールのその他のユーザー定義関数

次の UDF は、その他のファイアウォール操作を実行します:

- `mysql_firewall_flush_status()`

この UDF は、いくつかのファイアウォールステータス変数を 0 にリセットします:

- `Firewall_access_denied`
- `Firewall_access_granted`
- `Firewall_access_suspicious`

この UDF には、`FIREWALL_ADMIN` 権限または非推奨の `SUPER` 権限が必要です。

例:

```
SELECT mysql_firewall_flush_status();
```

- `normalize_statement(stmt)`

この UDF は、SQL ステートメントを許可リストルールに使用されるダイジェストフォームに正規化します。`FIREWALL_ADMIN` 権限または非推奨の `SUPER` 権限が必要です。

例:

```
SELECT normalize_statement('SELECT * FROM t1 WHERE c1 > 2');
```

注記

`STATEMENT_DIGEST_TEXT()` SQL 関数を使用して、ファイアウォールコンテキストの外で同じダイジェスト機能を使用できます。

MySQL Enterprise Firewall システム変数

MySQL Enterprise Firewall は、次のシステム変数をサポートしています。これらを使用してファイアウォール操作を構成します。これらの変数は、ファイアウォールがインストールされていないかぎり使用できません ([セクション 6.4.7.2 「MySQL Enterprise Firewall のインストールまたはアンインストール」](#) を参照)。

- `mysql_firewall_mode`

コマンド行形式	<code>--mysql-firewall-mode[={OFF ON}]</code>
システム変数	<code>mysql_firewall_mode</code>

スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

MySQL Enterprise Firewall が有効 (デフォルト) か無効か。

- [mysql_firewall_trace](#)

コマンド行形式	--mysql-firewall-trace[={OFF ON}]
システム変数	mysql_firewall_trace
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

MySQL Enterprise Firewall トレースが有効か無効か (デフォルト)。 [mysql_firewall_trace](#) が有効な場合、**PROTECTING** モードでは、ファイアウォールは拒否されたステートメントをエラーログに書き込みます。

MySQL Enterprise Firewall ステータス変数

MySQL Enterprise Firewall では、次のステータス変数がサポートされます。これらを使用して、ファイアウォールの操作ステータスに関する情報を取得します。これらの変数は、ファイアウォールがインストールされていないがざり使用できません (セクション6.4.7.2「MySQL Enterprise Firewall のインストールまたはアンインストール」を参照)。[MYSQL_FIREWALL](#) プラグインがインストールされるか、サーバーが起動されるたびに、ファイアウォールステータス変数は 0 に設定されます。これらの多くは、[mysql_firewall_flush_status\(\)](#) UDF によってゼロにリセットされます (MySQL Enterprise Firewall のユーザー定義関数を参照)。

- [Firewall_access_denied](#)

MySQL Enterprise Firewall によって拒否されたステートメントの数。

- [Firewall_access_granted](#)

MySQL Enterprise Firewall で受け入れられるステートメントの数。

- [Firewall_access_suspicious](#)

DETECTING モードのユーザーの疑わしいステートメントとして MySQL Enterprise Firewall によって記録されたステートメントの数。

- [Firewall_cached_entries](#)

MySQL Enterprise Firewall によって記録されたステートメントの数 (重複を含む)。

6.5 MySQL Enterprise Data Masking and De-Identification

注記

MySQL Enterprise Data Masking and De-Identification は、商用製品である MySQL Enterprise Edition に含まれる拡張機能です。商用製品の詳細は、<https://www.mysql.com/products/> を参照してください。

MySQL 8.0.13 では、MySQL Enterprise Edition はデータマスキングおよび識別解除機能を提供します:

- クレジットカード番号の最後の 4 桁を 'X' 文字に変更するなど、既存のデータをマスクして識別特性を削除するための変換。
- E メールアドレスや支払カード番号などのランダムデータの生成。

アプリケーションでこれらの機能を使用する方法は、データが使用される目的およびデータにアクセスするユーザーによって異なります:

- 機密データを使用するアプリケーションは、データマスキングを実行し、部分的にマスクされたデータをクライアント識別に使用できるようにすることで、機密データを保護できます。例: コールセンターは、最後の 4 桁の社会保障番号を提供するようクライアントに要求する場合があります。
- 適切にフォーマットされたデータを必要とするが、必ずしも元のデータを必要としないアプリケーションでは、サンプルデータを合成できます。例: データバリデータをテストしているが、元のデータにアクセスできないアプリケーション開発者は、ランダムデータを同じ形式で合成できます。

例 1:

医療研究施設は、個人データと医療データの混合を含む患者データを保持できます。これには、遺伝的順序 (長い文字列)、JSON 形式で格納されたテスト結果およびその他のデータ型が含まれます。ほとんどの場合、データは自動分析ソフトウェアで使用できますが、特定の患者のゲノムデータまたはテスト結果へのアクセスは可能です。このような場合は、データマスキングを使用して、個人を識別できないこの情報をレンダリングする必要があります。

例 2:

クレジットカードプロセッサ会社は、次のような機密データを使用して一連のサービスを提供します:

- 1 秒当たりの多数の財務トランザクションの処理。
- 大量のトランザクション関連データの格納。
- 個人データの厳格な要件によるトランザクション関連データの保護。
- 可逆または部分的にマスクされたデータを使用したトランザクションに関するクライアントの苦情を処理します。

一般的なトランザクションには、次のような多くのタイプの機密情報が含まれる場合があります:

- クレジットカード番号。
- トランザクションタイプおよび金額。
- 業者タイプ。
- トランザクション暗号化 (トランザクション正当性を確認するため)。
- GPS 対応端末の地理的位置 (不正検出用)。

これらのタイプの情報は、銀行またはその他のカード発行金融機関内で、次のようなクライアント個人データと結合できます:

- 完全なクライアント名 (個人または会社)。
- Address.
- 生年月日。
- 社会保障番号。
- 電子メールアドレス。
- 電話番号。

カード処理会社と金融機関の両方で様々な従業員ロールを使用するには、そのデータにアクセスする必要があります。これらのロールの中には、マスキングされたデータへのアクセスのみを必要とするものがあります。他のロールでは、ケースごとに元のデータへのアクセスが必要になる場合があります。このデータは監査ログに記録されます。

マスキングおよび識別解除は規制コンプライアンスの中核となるため、MySQL Enterprise Data Masking and De-Identification はアプリケーション開発者がプライバシー要件を満たすのに役立ちます:

- PCI - DSS: 支払カードデータ。
- HIPAA: 医療データのプライバシー、経済臨床医療法 (HITECH 法) の健康情報技術。
- EU 一般データ保護ディレクティブ (GDPR): 個人データの保護。
- データ保護法 (英国): 個人データの保護。
- Sarbanes Oxley, GLBA, The USA Patriot Act, Identity Theft and Assumption Deterrence Act of 1998。
- FERPA - Student Data, NASD, CA SB1386 および AB 1950, State Data Protection Laws, Basel II。

次の各セクションでは、MySQL Enterprise Data Masking and De-Identification の要素について説明し、それをインストールおよび使用する方法を説明し、その要素のリファレンス情報を提供します。

6.5.1 MySQL Enterprise Data Masking and De-Identification 要素

MySQL Enterprise Data Masking and De-Identification は、次の要素を実装するプラグインライブラリに基づいています:

- `data_masking` という名前のサーバー側プラグイン。
- 一連のユーザー定義関数 (UDF) は、マスキングおよび識別解除操作を実行するための SQL レベルの API を提供します。これらの関数の中には、`SUPER` 権限を必要とするものがあります。

6.5.2 MySQL Enterprise Data Masking and De-Identification のインストールまたはアンインストール

このセクションでは、プラグインおよびユーザー定義関数 (UDF) を含むプラグインライブラリファイルとして実装される MySQL Enterprise Data Masking and De-Identification をインストールまたはアンインストールする方法について説明します。プラグインおよび UDF のインストールまたはアンインストールの一般情報は、[セクション5.6.1「プラグインのインストールおよびアンインストール」](#) および [セクション5.7.1「ユーザー定義関数のインストールおよびアンインストール」](#) を参照してください。

サーバーで使用できるようにするには、プラグインライブラリファイルを MySQL プラグインディレクトリ (`plugin_dir` システム変数で指定されたディレクトリ) に配置する必要があります。必要に応じて、サーバーの起動時に `plugin_dir` の値を設定してプラグインディレクトリの場所を構成します。

プラグインライブラリファイルのベース名は `data_masking` です。ファイル名の接尾辞は、プラットフォームごとに異なります (たとえば、`.so` for Unix and Unix-like systems, `.dll` for Windows)。

MySQL Enterprise Data Masking and De-Identification プラグインおよび UDF をインストールするには、必要に応じてプラットフォームの `.so` 接尾辞を調整して、`INSTALL PLUGIN` および `CREATE FUNCTION` ステートメントを使用します:

```
INSTALL PLUGIN data_masking SONAME 'data_masking.so';
CREATE FUNCTION gen_blocklist RETURNS STRING
SONAME 'data_masking.so';
CREATE FUNCTION gen_dictionary RETURNS STRING
SONAME 'data_masking.so';
CREATE FUNCTION gen_dictionary_drop RETURNS STRING
SONAME 'data_masking.so';
CREATE FUNCTION gen_dictionary_load RETURNS STRING
SONAME 'data_masking.so';
CREATE FUNCTION gen_range RETURNS INTEGER
SONAME 'data_masking.so';
CREATE FUNCTION gen_rnd_email RETURNS STRING
SONAME 'data_masking.so';
CREATE FUNCTION gen_rnd_pan RETURNS STRING
SONAME 'data_masking.so';
CREATE FUNCTION gen_rnd_ssn RETURNS STRING
SONAME 'data_masking.so';
CREATE FUNCTION gen_rnd_us_phone RETURNS STRING
SONAME 'data_masking.so';
CREATE FUNCTION mask_inner RETURNS STRING
SONAME 'data_masking.so';
CREATE FUNCTION mask_outer RETURNS STRING
```

```
SONAME 'data_masking.so';
CREATE FUNCTION mask_pan RETURNS STRING
SONAME 'data_masking.so';
CREATE FUNCTION mask_pan_relaxed RETURNS STRING
SONAME 'data_masking.so';
CREATE FUNCTION mask_ssn RETURNS STRING
SONAME 'data_masking.so';
```

プラグインおよび UDF がソースレプリケーションサーバーで使用されている場合は、レプリケーションの問題を回避するために、それらをすべてのレプリカサーバーにインストールします。

前述のようにインストールすると、プラグインと UDF はアンインストールされるまでインストールされたままになります。これらを削除するには、[UNINSTALL PLUGIN](#) および [DROP FUNCTION](#) ステートメントを使用します：

```
UNINSTALL PLUGIN data_masking;
DROP FUNCTION gen_blocklist;
DROP FUNCTION gen_dictionary;
DROP FUNCTION gen_dictionary_drop;
DROP FUNCTION gen_dictionary_load;
DROP FUNCTION gen_range;
DROP FUNCTION gen_rnd_email;
DROP FUNCTION gen_rnd_pan;
DROP FUNCTION gen_rnd_ssn;
DROP FUNCTION gen_rnd_us_phone;
DROP FUNCTION mask_inner;
DROP FUNCTION mask_outer;
DROP FUNCTION mask_pan;
DROP FUNCTION mask_pan_relaxed;
DROP FUNCTION mask_ssn;
```

6.5.3 MySQL Enterprise Data Masking and De-Identification の使用

MySQL Enterprise Data Masking and De-Identification を使用する前に、[セクション6.5.2「MySQL Enterprise Data Masking and De-Identification のインストールまたはアンインストール」](#)に記載されている手順に従ってインストールします。

アプリケーションで MySQL Enterprise Data Masking and De-Identification を使用するには、実行する操作に適した関数を呼び出します。関数の詳細は、[セクション6.5.5「MySQL Enterprise Data Masking and De-Identification ユーザー定義関数の説明」](#)を参照してください。このセクションでは、関数を使用して代表的なタスクを実行する方法を示します。最初に、使用可能な関数の概要を示し、次にその関数を実際のコンテキストで使用する方法の例をいくつか示します：

- [識別特性を削除するためのデータのマスクング](#)
- [特定の特性を持つランダムデータの生成](#)
- [ディクショナリを使用したランダムデータの生成](#)
- [顧客識別のためのマスクされたデータの使用](#)
- [マスクされたデータを表示するビューの作成](#)

識別特性を削除するためのデータのマスクング

MySQL には、任意の文字列をマスクする汎用マスクング関数と、特定のタイプの値をマスクする特別な目的のマスクング関数が用意されています。

汎用マスクング関数

[mask_inner\(\)](#) および [mask_outer\(\)](#) は、文字列内の位置に基づいて任意の文字列の一部をマスクする汎用関数です：

- [mask_inner\(\)](#) は、その文字列引数の内部をマスクし、末尾はマスクされないままにします。その他の引数は、マスクされていない端のサイズを指定します。

```
mysql> SELECT mask_inner('This is a string', 5, 1);
+-----+
| mask_inner('This is a string', 5, 1) |
+-----+
| This XXXXXXXXXXg |
```

```

+-----+
mysql> SELECT mask_inner('This is a string', 1, 5);
+-----+
| mask_inner('This is a string', 1, 5) |
+-----+
| TXXXXXXXXXtring |
+-----+

```

- `mask_outer()` は逆の処理を行い、文字列引数の終わりをマスキングして、内部をマスクしないままにします。その他の引数は、マスクされた端のサイズを指定します。

```

mysql> SELECT mask_outer('This is a string', 5, 1);
+-----+
| mask_outer('This is a string', 5, 1) |
+-----+
| XXXXis a strinX |
+-----+
mysql> SELECT mask_outer('This is a string', 1, 5);
+-----+
| mask_outer('This is a string', 1, 5) |
+-----+
| Xhis is a sXXXXX |
+-----+

```

デフォルトでは、`mask_inner()` および `mask_outer()` はマスキング文字として 'X' を使用しますが、オプションのマスキング文字引数を使用できます:

```

mysql> SELECT mask_inner('This is a string', 5, 1, '*');
+-----+
| mask_inner('This is a string', 5, 1, '*') |
+-----+
| This *****g |
+-----+
mysql> SELECT mask_outer('This is a string', 5, 1, '#');
+-----+
| mask_outer('This is a string', 5, 1, '#') |
+-----+
| #####is a strin# |
+-----+

```

特殊用途マスキング関数

その他のマスキング関数では、特定のタイプの値を表す文字列引数を想定し、それをマスクして識別特性を削除します。

注記

この例では、適切なタイプの値を返すランダム値生成関数を使用して関数の引数を指定します。生成関数の詳細は、[特定の特性を持つランダムデータの生成](#) を参照してください。

支払カードプライマリアカウント番号マスキング。 マスキング機能は、主要アカウント番号の厳密で緩和されたマスキングを提供します。

- `mask_pan()` は、数値の最後の 4 桁を除くすべての桁をマスクします:

```

mysql> SELECT mask_pan(gen_rnd_pan());
+-----+
| mask_pan(gen_rnd_pan()) |
+-----+
| XXXXXXXXXXXX2461 |
+-----+

```

- `mask_pan_relaxed()` は似ていますが、支払カード会社のマスクが解除されたことを示す最初の 6 桁はマスクされません:

```

mysql> SELECT mask_pan_relaxed(gen_rnd_pan());
+-----+
| mask_pan_relaxed(gen_rnd_pan()) |
+-----+
| 770630XXXXXX0807 |
+-----+

```

US 社会保障番号マスキング. `mask_ssn()` は、数値の最後の 4 桁を除くすべての桁をマスクします:

```
mysql> SELECT mask_ssn(gen_rand_ssn());
+-----+
| mask_ssn(gen_rand_ssn()) |
+-----+
| XXX-XX-1723             |
+-----+
```

特定の特性を持つランダムデータの生成

いくつかの関数でランダム値が生成されます。これらの値は、テストやシミュレーションなどに使用できます。

`gen_range()` は、指定された範囲から選択されたランダムな整数を返します:

```
mysql> SELECT gen_range(1, 10);
+-----+
| gen_range(1, 10) |
+-----+
|          6      |
+-----+
```

`gen_rand_email()` は、`example.com` ドメインでランダムな電子メールアドレスを返します:

```
mysql> SELECT gen_rand_email();
+-----+
| gen_rand_email()      |
+-----+
| ayxnq.xmkpvvy@example.com |
+-----+
```

`gen_rand_pan()` は、ランダムな支払カード主要アカウント番号を返します:

```
mysql> SELECT gen_rand_pan();
```

(戻り値は公開ではなくテスト目的でのみ使用する必要があるため、`gen_rand_pan()` 関数の結果は表示されません。番号が正当な支払アカウントに割り当てられていないことを保証できません。)

`gen_rand_ssn()` はランダムな US 社会保障番号を返し、それぞれ正当な番号に使用されていない範囲から選択された最初と 2 番目の部分が含まれます:

```
mysql> SELECT gen_rand_ssn();
+-----+
| gen_rand_ssn() |
+-----+
| 912-45-1615    |
+-----+
```

`gen_rand_us_phone()` は、正当な番号に使用されていない 555 市外局番でランダムな米国電話番号を返します:

```
mysql> SELECT gen_rand_us_phone();
+-----+
| gen_rand_us_phone() |
+-----+
| 1-555-747-5627      |
+-----+
```

ディクショナリを使用したランダムデータの生成

MySQL Enterprise Data Masking and De-Identification では、ディクショナリをランダム値のソースとして使用できます。ディクショナリを使用するには、最初にファイルからロードし、名前を指定する必要があります。ロードされた各ディクショナリは、ディクショナリレジストリの一部になります。その後、登録済ディクショナリからアイテムを選択し、ランダム値または他の値の置換として使用できます。

有効なディクショナリファイルには、次の特性があります:

- ファイルの内容はプレーンテキストで、1 行につき 1 語です。
- 空の行は無視されます。
- ファイルには少なくとも 1 つの用語が含まれている必要があります。

`de_cities.txt` という名前のファイルにドイツの次の市区町村名が含まれているとします:

```
Berlin
Munich
Bremen
```

また、`us_cities.txt` という名前のファイルに米国の次の市区町村名が含まれているとします:

```
Chicago
Houston
Phoenix
El Paso
Detroit
```

`secure_file_priv` システム変数が `/usr/local/mysql/mysql-files` に設定されているとします。その場合は、MySQL サーバーがアクセスできるように、ディクショナリファイルをそのディレクトリにコピーします。次に、`gen_dictionary_load()` を使用してディクショナリをディクショナリレジストリにロードし、ディクショナリに名前を割り当てます:

```
mysql> SELECT gen_dictionary_load('/usr/local/mysql/mysql-files/de_cities.txt', 'DE_Cities');
+-----+
| gen_dictionary_load('/usr/local/mysql/mysql-files/de_cities.txt', 'DE_Cities') |
+-----+
| Dictionary load success |
+-----+
mysql> SELECT gen_dictionary_load('/usr/local/mysql/mysql-files/us_cities.txt', 'US_Cities');
+-----+
| gen_dictionary_load('/usr/local/mysql/mysql-files/us_cities.txt', 'US_Cities') |
+-----+
| Dictionary load success |
+-----+
```

ディクショナリからランダムな用語を選択するには、`gen_dictionary()` を使用します:

```
mysql> SELECT gen_dictionary('DE_Cities');
+-----+
| gen_dictionary('DE_Cities') |
+-----+
| Berlin |
+-----+
mysql> SELECT gen_dictionary('US_Cities');
+-----+
| gen_dictionary('US_Cities') |
+-----+
| Phoenix |
+-----+
```

複数のディクショナリからランダムな用語を選択するには、いずれかのディクショナリをランダムに選択し、そこから用語を選択します:

```
mysql> SELECT gen_dictionary(ELT(gen_range(1,2), 'DE_Cities', 'US_Cities'));
+-----+
| gen_dictionary(ELT(gen_range(1,2), 'DE_Cities', 'US_Cities')) |
+-----+
| Detroit |
+-----+
mysql> SELECT gen_dictionary(ELT(gen_range(1,2), 'DE_Cities', 'US_Cities'));
+-----+
| gen_dictionary(ELT(gen_range(1,2), 'DE_Cities', 'US_Cities')) |
+-----+
| Bremen |
+-----+
```

`gen_blocklist()` 関数を使用すると、あるディクショナリの用語を別のディクショナリの用語に置き換えることができ、置換によるマスキングに影響します。引数は、置換する用語、用語が表示されるディクショナリおよび置換を選択するディクショナリです。たとえば、米国の市をドイツの市に、またはその逆に置き換えるには、次のように `gen_blocklist()` を使用します:

```
mysql> SELECT gen_blocklist('Munich', 'DE_Cities', 'US_Cities');
+-----+
| gen_blocklist('Munich', 'DE_Cities', 'US_Cities') |
+-----+
```

```

| Houston          |
+-----+-----+
mysql> SELECT gen_blocklist('El Paso', 'US_Cities', 'DE_Cities');
+-----+-----+
| gen_blocklist('El Paso', 'US_Cities', 'DE_Cities') |
+-----+-----+
| Bremen          |
+-----+-----+

```

置換する用語が最初のディクショナリにない場合、`gen_blocklist()` はそれを変更せずに戻します:

```

mysql> SELECT gen_blocklist('Moscow', 'DE_Cities', 'US_Cities');
+-----+-----+
| gen_blocklist('Moscow', 'DE_Cities', 'US_Cities') |
+-----+-----+
| Moscow        |
+-----+-----+

```

顧客識別のためのマスクされたデータの使用

カスタマーサービスコールセンターでは、一般的なアイデンティティ検証手法として、顧客に最後の 4 桁の社会保障番号 (SSN) を提供するように依頼します。たとえば、顧客の名前が Joanna Bond で、最後の 4 桁の SSN が 0007 であるとします。

顧客レコードを含む `customer` テーブルに次のカラムがあるとします:

- `id`: 顧客 ID 番号。
- `first_name`: 顧客の名。
- `last_name`: 顧客の姓。
- `ssn`: 顧客社会保障番号。

たとえば、テーブルは次のように定義できます:

```

CREATE TABLE customer
(
  id      BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  first_name VARCHAR(40),
  last_name VARCHAR(40),
  ssn     VARCHAR(11)
);

```

顧客サービス担当者が顧客 SSN をチェックするために使用するアプリケーションは、次のようなクエリーを実行します:

```

mysql> SELECT id, ssn
mysql> FROM customer
mysql> WHERE first_name = 'Joanna' AND last_name = 'Bond';
+----+-----+
| id | ssn   |
+----+-----+
| 786 | 906-39-0007 |
+----+-----+

```

ただし、は SSN をカスタマーサービス担当者に公開します。カスタマーサービス担当者は、最後の 4 桁以外を表示する必要はありません。かわりに、アプリケーションは次のクエリーを使用して、マスクされた SSN のみを表示できます:

```

mysql> SELECT id, mask_ssn(CONVERT(ssn USING binary)) AS masked_ssn
mysql> FROM customer
mysql> WHERE first_name = 'Joanna' AND last_name = 'Bond';
+----+-----+
| id | masked_ssn |
+----+-----+
| 786 | XXX-XX-0007 |
+----+-----+

```

これで、担当者には必要なもののみが表示され、お客様のプライバシーは保持されます。

`mask_ssn()` への引数に `CONVERT()` 関数が使用されたのはなぜですか。 `mask_ssn()` には長さ 11 の引数が必要であるためです。したがって、`ssn` が `VARCHAR(11)` として定義されている場合でも、`ssn` カラムにマルチバイト文字セットが含まれていると、UDF に渡されたときに 11 バイトを超える可能性があり、エラーが発生します。値をバイナリ文字列に変換すると、UDF に長さ 11 の引数が表示されます。

文字列引数にシングルバイト文字セットがない場合は、他のデータマスキング関数でも同様の方法が必要になることがあります。

マスクされたデータを表示するビューの作成

テーブルのマスキングされたデータが複数のクエリーに使用される場合、マスキングされたデータを生成するビューを定義すると便利です。これにより、アプリケーションは個々のクエリーでマスキングを実行せずにビューから選択できます。

たとえば、前のセクションの `customer` テーブルのマスキングビューは、次のように定義できます：

```
CREATE VIEW masked_customer AS
SELECT id, first_name, last_name,
mask_ssn(CONVERT(ssn USING binary)) AS masked_ssn
FROM customer;
```

その後、顧客を検索するクエリーは単純になりますが、マスキングされたデータは返されます：

```
mysql> SELECT id, masked_ssn
mysql> FROM masked_customer
mysql> WHERE first_name = 'Joanna' AND last_name = 'Bond';
+----+-----+
| id | masked_ssn |
+----+-----+
| 786 | XXX-XX-0007 |
+----+-----+
```

6.5.4 MySQL Enterprise Data Masking and De-Identification ユーザー定義関数リファレンス

表 6.38 「MySQL Enterprise Data Masking and De-Identification のユーザー定義関数」

名前	説明	導入	非推奨
<code>gen_blacklist()</code>	辞書用語置換の実行		8.0.23
<code>gen_blocklist()</code>	辞書用語置換の実行	8.0.23	
<code>gen_dictionary()</code>	辞書からランダムな用語を返します		
<code>gen_dictionary_drop()</code>	レジストリからディクショナリを削除		
<code>gen_dictionary_load()</code>	ディクショナリをレジストリにロード		
<code>gen_range()</code>	範囲内の乱数を生成		
<code>gen_rnd_email()</code>	ランダムな電子メールアドレスの生成		
<code>gen_rnd_pan()</code>	ランダム支払カードプライマリアカウント番号の生成		
<code>gen_rnd_ssn()</code>	ランダム US 社会保障番号の生成		
<code>gen_rnd_us_phone()</code>	ランダムな米国電話番号の生成		
<code>mask_inner()</code>	文字列の内部部分のマスク		
<code>mask_outer()</code>	文字列の左右の部分のマスク		

名前	説明	導入	非推奨
<code>mask_pan()</code>	文字列の支払カードプライマリアカウント番号部分のマスク		
<code>mask_pan_relaxed()</code>	文字列の支払カードプライマリアカウント番号部分のマスク		
<code>mask_ssn()</code>	US 社会保障番号のマスク		

6.5.5 MySQL Enterprise Data Masking and De-Identification ユーザー定義関数の説明

MySQL Enterprise Data Masking and De-Identification プラグインライブラリには、次のカテゴリにグループ化できるいくつかのユーザー定義関数 (UDF) が含まれています:

- データマスキング関数
- ランダムデータ生成関数
- ランダムデータディクショナリビューの関数

MySQL 8.0.19 では、これらの UDF は文字列引数および戻り値のシングルバイト `latin1` 文字セットをサポートしています。MySQL 8.0.19 より前では、UDF は文字列引数をバイナリ文字列として扱い (大文字と小文字を区別しない)、文字列の戻り値はバイナリ文字列です。戻り値の文字セットの違いは、次のとおりです:

MySQL 8.0.19 以上:

```
mysql> SELECT CHARSET(gen_rnd_email());
+-----+
| CHARSET(gen_rnd_email()) |
+-----+
| latin1                    |
+-----+
```

MySQL 8.0.19 より前:

```
mysql> SELECT CHARSET(gen_rnd_email());
+-----+
| CHARSET(gen_rnd_email()) |
+-----+
| binary                    |
+-----+
```

いずれのバージョンでも、文字列の戻り値を別の文字セットにする必要がある場合は、変換します。次の例は、`gen_rnd_email()` の結果を `utf8mb4` 文字セットに変換する方法を示しています:

```
SET @email = CONVERT(gen_rnd_email() USING utf8mb4);
```

[顧客識別のためのマスクされたデータの使用](#) に示されているように、文字列引数の変換が必要になる場合もあります。

データマスキング関数

このセクションの各関数は、文字列引数に対してマスキング操作を実行し、マスキングされた結果を返します。

- `mask_inner(str, margin1, margin2 [, mask_char])`

文字列の内部部分をマスクし、末尾はそのままにして、結果を返します。オプションのマスキング文字を指定できます。

引数:

- `str`: マスクする文字列。

- **margin1**: マスクされないままにする文字列の左端の文字数を指定する負でない整数。値が 0 の場合、左の終了文字はマスクされません。
- **margin2**: マスクされないままにする文字列の右端の文字数を指定する負でない整数。値が 0 の場合、右側の終了文字はマスクされません。
- **mask_char**: (オプション) マスキングに使用する単一の文字。 **mask_char** が指定されていない場合、デフォルトは 'X' です。

マスキング文字はシングルバイト文字である必要があります。マルチバイト文字を使用しようとすると、エラーが発生します。

戻り値:

マスクされた文字列。いずれかのマージンが負の場合は **NULL**。

マージン値の合計が引数の長さより大きい場合、マスキングは行われず、引数は変更されずに返されます。

例:

```
mysql> SELECT mask_inner('abcdef', 1, 2), mask_inner('abcdef', 0, 5);
+-----+-----+
| mask_inner('abcdef', 1, 2) | mask_inner('abcdef', 0, 5) |
+-----+-----+
| aXXef           | Xbcdef           |
+-----+-----+
mysql> SELECT mask_inner('abcdef', 1, 2, '*'), mask_inner('abcdef', 0, 5, '#');
+-----+-----+
| mask_inner('abcdef', 1, 2, '*') | mask_inner('abcdef', 0, 5, '#') |
+-----+-----+
| a***ef           | #bcdef           |
+-----+-----+
```

- **mask_outer(str, margin1, margin2 [, mask_char])**

文字列の左端と右端をマスクし、内部をマスクしないままにして、結果を返します。オプションのマスキング文字を指定できます。

引数:

- **str**: マスクする文字列。
- **margin1**: マスクする文字列の左端の文字数を指定する負でない整数。値が 0 の場合、左端の文字はマスクされません。
- **margin2**: マスクする文字列の右端の文字数を指定する負でない整数。値が 0 の場合、右端の文字はマスクされません。
- **mask_char**: (オプション) マスキングに使用する単一の文字。 **mask_char** が指定されていない場合、デフォルトは 'X' です。

マスキング文字はシングルバイト文字である必要があります。マルチバイト文字を使用しようとすると、エラーが発生します。

戻り値:

マスクされた文字列。いずれかのマージンが負の場合は **NULL**。

マージン値の合計が引数の長さより大きい場合、引数全体がマスクされます。

例:

```
mysql> SELECT mask_outer('abcdef', 1, 2), mask_outer('abcdef', 0, 5);
+-----+-----+
| mask_outer('abcdef', 1, 2) | mask_outer('abcdef', 0, 5) |
+-----+-----+
```

```

| XbcdXX          | aXXXXXX          |
+-----+-----+
mysql> SELECT mask_outer('abcdef', 1, 2, '**'), mask_outer('abcdef',0, 5, '#');
+-----+-----+
| mask_outer('abcdef', 1, 2, '**') | mask_outer('abcdef',0, 5, '#') |
+-----+-----+
| *bcd**          | a#####          |
+-----+-----+

```

- `mask_pan(str)`

支払カードプライマリアカウント番号をマスクし、最後の 4 桁を除くすべての数字を 'X' 文字に置き換えて返します。

引数:

- **str**: マスクする文字列。文字列はプライマリアカウント番号に適した長さである必要がありますが、それ以外の場合は選択されません。

戻り値:

マスクされた支払番号を文字列として指定します。引数が必須より短い場合は、変更されずに返されます。

例:

```

mysql> SELECT mask_pan(gen_rnd_pan());
+-----+
| mask_pan(gen_rnd_pan()) |
+-----+
| XXXXXXXXXXXXX9102     |
+-----+
mysql> SELECT mask_pan(gen_rnd_pan(19));
+-----+
| mask_pan(gen_rnd_pan(19)) |
+-----+
| XXXXXXXXXXXXXXXXXXX8268 |
+-----+
mysql> SELECT mask_pan('a*Z');
+-----+
| mask_pan('a*Z') |
+-----+
| a*Z             |
+-----+

```

- `mask_pan_relaxed(str)`

支払カードプライマリアカウント番号をマスクし、最初の 6 桁と最後の 4 桁を除くすべての数字を 'X' 文字に置き換えて返します。最初の 6 桁は、支払カード会社を示します。

引数:

- **str**: マスクする文字列。文字列はプライマリアカウント番号に適した長さである必要がありますが、それ以外の場合は選択されません。

戻り値:

マスクされた支払番号を文字列として指定します。引数が必須より短い場合は、変更されずに返されます。

例:

```

mysql> SELECT mask_pan_relaxed(gen_rnd_pan());
+-----+
| mask_pan_relaxed(gen_rnd_pan()) |
+-----+
| 551279XXXXXXXX3108             |
+-----+
mysql> SELECT mask_pan_relaxed(gen_rnd_pan(19));
+-----+
| mask_pan_relaxed(gen_rnd_pan(19)) |
+-----+

```

```

| 462634XXXXXXXXXX6739 |
+-----+
mysql> SELECT mask_pan_relaxed('a*Z');
+-----+
| mask_pan_relaxed('a*Z') |
+-----+
| a*Z |
+-----+

```

- `mask_ssn(str)`

米国社会保障番号をマスクし、最後の 4 桁を除くすべての数字を 'X' 文字に置き換えて返します。

引数:

- `str`: マスクする文字列。文字列の長さは 11 文字である必要がありますが、それ以外の場合はチェックされません。

戻り値:

マスクされた社会保障番号 (文字列)。引数の長さが正しくない場合は `NULL`。

例:

```

mysql> SELECT mask_ssn('909-63-6922'), mask_ssn('abcdefghijk');
+-----+-----+
| mask_ssn('909-63-6922') | mask_ssn('abcdefghijk') |
+-----+-----+
| XXX-XX-6922 | XXX-XX-hijk |
+-----+-----+
mysql> SELECT mask_ssn('909');
+-----+
| mask_ssn('909') |
+-----+
| NULL |
+-----+

```

ランダムデータ生成関数

このセクションの関数は、様々なタイプのデータに対してランダムな値を生成します。可能な場合、生成された値にはデモンストレーションまたはテスト値用に予約された特性があり、正当なデータを間違えないようにします。たとえば、`gen_rnd_us_phone()` は 555 市外局番を使用する米国電話番号を返しますが、これは実際の使用では電話番号に割り当てられていません。個々の関数の説明では、この原則の例外について説明します。

- `gen_range(lower, upper)`

指定された範囲から選択された乱数を生成します。

引数:

- `lower`: 範囲の下限を指定する整数。
- `upper`: 範囲の上限を指定する整数。下限より小さくすることはできません。

戻り値:

`lower` から `upper` までの範囲のランダムな整数 (`upper` 引数が `lower` より小さい場合は `NULL`)。

例:

```

mysql> SELECT gen_range(100, 200), gen_range(-1000, -800);
+-----+-----+
| gen_range(100, 200) | gen_range(-1000, -800) |
+-----+-----+
| 177 | -917 |
+-----+-----+
mysql> SELECT gen_range(1, 0);
+-----+
| gen_range(1, 0) |
+-----+

```

```
+-----+
|      NULL      |
+-----+
```

- `gen_rnd_email()`

`example.com` ドメインにランダムな電子メールアドレスを生成します。

引数:

なし

戻り値:

文字列としてのランダムな電子メールアドレス。

例:

```
mysql> SELECT gen_rnd_email();
+-----+
| gen_rnd_email() |
+-----+
| ijocv.mwvhuf@example.com |
+-----+
```

- `gen_rnd_pan([size])`

ランダム支払カードプライマリアカウント番号を生成します。数値は Luhn チェック (検証桁に対してチェックサム検証を実行するアルゴリズム) に合格します。

警告

`gen_rnd_pan()` から返される値はテスト目的でのみ使用する必要があり、公開には適していません。特定の戻り値が正当な支払アカウントに割り当てられていないことを保証する方法はありません。`gen_rnd_pan()` の結果を公開する必要がある場合は、`mask_pan()` または `mask_pan_relaxed()` でマスキングすることを検討してください。

引数:

- **size:** (オプション) 結果のサイズを指定する整数。 `size` が指定されていない場合、デフォルトは 16 です。指定する場合、`size` は 12 から 19 の範囲の整数である必要があります。

戻り値:

文字列としてのランダムな支払番号。許可された範囲外の `size` 引数が指定されている場合は `NULL`。

例:

```
mysql> SELECT mask_pan(gen_rnd_pan());
+-----+
| mask_pan(gen_rnd_pan()) |
+-----+
| XXXXXXXXXXXXX5805      |
+-----+
mysql> SELECT mask_pan(gen_rnd_pan(19));
+-----+
| mask_pan(gen_rnd_pan(19)) |
+-----+
| XXXXXXXXXXXXXXX5067      |
+-----+
mysql> SELECT mask_pan_relaxed(gen_rnd_pan());
+-----+
| mask_pan_relaxed(gen_rnd_pan()) |
+-----+
| 398403XXXXXX9547        |
+-----+
mysql> SELECT mask_pan_relaxed(gen_rnd_pan(19));
+-----+
| mask_pan_relaxed(gen_rnd_pan(19)) |
+-----+
```



```

+-----+
| 578416XXXXXXXXXX6509 |
+-----+
mysql> SELECT gen_rnd_pan(11), gen_rnd_pan(20);
+-----+
| gen_rnd_pan(11) | gen_rnd_pan(20) |
+-----+
| NULL          | NULL          |
+-----+

```

- [gen_rnd_ssn\(\)](#)

ランダムな US 社会保障番号を **AAA-BB-CCCC** 形式で生成します。AAA 部分が 900 を超え、BB 部分が 70 未満です。これは、正当な社会保障番号に使用されない特性です。

引数:

なし

戻り値:

文字列としてのランダムな社会保障番号。

例:

```

mysql> SELECT gen_rnd_ssn();
+-----+
| gen_rnd_ssn() |
+-----+
| 951-26-0058  |
+-----+

```

- [gen_rnd_us_phone\(\)](#)

ランダムな米国電話番号を **1-555-AAA-BBBB** 形式で生成します。555 市外局番は正当な電話番号には使用されません。

引数:

なし

戻り値:

文字列としてのランダムな US 電話番号。

例:

```

mysql> SELECT gen_rnd_us_phone();
+-----+
| gen_rnd_us_phone() |
+-----+
| 1-555-682-5423  |
+-----+

```

ランダムデータディクショナリビューの関数

このセクションの関数は、用語のディクショナリを操作し、それらに基づいて生成およびマスキング操作を実行します。これらの関数の中には、**SUPER** 権限を必要とするものがあります。

ディクショナリがロードされると、ディクショナリレジストリの一部になり、他のディクショナリ関数で使用される名前が割り当てられます。ディクショナリは、行ごとに 1 語を含むプレーンテキストファイルからロードされます。空の行は無視されます。有効にするには、辞書ファイルに空でない行が少なくとも 1 つ含まれている必要があります。

- [gen_blacklist\(str, dictionary_name, replacement_dictionary_name\)](#)

1つの辞書に存在する用語を2つ目の辞書用語に置き換え、置換する用語を返します。これは、置換によって元の用語をマスクします。この関数はMySQL 8.0.23では非推奨です。かわりに `gen_blocklist()` を使用してください。

- `gen_blocklist(str, dictionary_name, replacement_dictionary_name)`

1つの辞書に存在する用語を2つ目の辞書用語に置き換え、置換する用語を返します。これは、置換によって元の用語をマスクします。この関数はMySQL 8.0.23で追加されました。 `gen_blacklist()` のかわりに使用してください。

引数:

- `str`: 置換する用語を示す文字列。
- `dictionary_name`: 置換する用語を含むディクショナリを指定する文字列。
- `replacement_dictionary_name`: 置換語を選択するディクショナリを指定する文字列。

戻り値:

`str` の代替として `replacement_dictionary_name` からランダムに選択された文字列。 `dictionary_name` に表示されない場合は `str`、いずれかのディクショナリ名がディクショナリレジストリにない場合は `NULL`。

置換する用語が両方のディクショナリにある場合は、戻り値が同じ用語である可能性があります。

例:

```
mysql> SELECT gen_blocklist('Berlin', 'DE_Cities', 'US_Cities');
+-----+
| gen_blocklist('Berlin', 'DE_Cities', 'US_Cities') |
+-----+
| Phoenix |
+-----+
```

- `gen_dictionary(dictionary_name)`

ディクショナリからランダムな用語を返します。

引数:

- `dictionary_name`: 用語を選択するディクショナリを指定する文字列。

戻り値:

ディクショナリからの文字列としてのランダムな用語。ディクショナリ名がディクショナリレジストリにない場合は `NULL`。

例:

```
mysql> SELECT gen_dictionary('mydict');
+-----+
| gen_dictionary('mydict') |
+-----+
| My term |
+-----+
mysql> SELECT gen_dictionary('no-such-dict');
+-----+
| gen_dictionary('no-such-dict') |
+-----+
| NULL |
+-----+
```

- `gen_dictionary_drop(dictionary_name)`

ディクショナリレジストリからディクショナリを削除します。

この関数には、`SUPER` 権限が必要です。

引数:

- `dictionary_name`: ディクショナリレジストリから削除するディクショナリを指定する文字列。

戻り値:

削除操作が成功したかどうかを示す文字列。 `Dictionary removed` は成功を示します。 `Dictionary removal error` は失敗を示します。

例:

```
mysql> SELECT gen_dictionary_drop('mydict');
+-----+
| gen_dictionary_drop('mydict') |
+-----+
| Dictionary removed           |
+-----+
mysql> SELECT gen_dictionary_drop('no-such-dict');
+-----+
| gen_dictionary_drop('no-such-dict') |
+-----+
| Dictionary removal error         |
+-----+
```

- `gen_dictionary_load(dictionary_path, dictionary_name)`

ファイルをディクショナリレジストリにロードし、ディクショナリ名引数を必要とする他の関数で使用される名前をディクショナリに割り当てます。

この関数には、`SUPER` 権限が必要です。

重要

ディクショナリは永続的ではありません。アプリケーションで使用されるディクショナリは、サーバーの起動ごとにロードする必要があります。

レジストリにロードされたディクショナリは、基礎となるディクショナリファイルが変更された場合でもそのまま使用されます。ディクショナリをリロードするには、まず `gen_dictionary_drop()` でディクショナリを削除してから、`gen_dictionary_load()` で再度ロードします。

引数:

- `dictionary_path`: ディクショナリファイルのパス名を指定する文字列。
- `dictionary_name`: ディクショナリの名前を示す文字列。

戻り値:

ロード操作が成功したかどうかを示す文字列。 `Dictionary load success` は成功を示します。 `Dictionary load error` は失敗を示します。ディクショナリロードの失敗は、次のようないくつかの理由で発生します:

- 指定された名前のディクショナリはすでにロードされています。
- ディクショナリファイルが見つかりません。
- 辞書ファイルに用語が含まれていません。
- `secure_file_priv` システム変数が設定されており、ディクショナリファイルが変数で指定されたディレクトリにありません。

例:

```
mysql> SELECT gen_dictionary_load('/usr/local/mysql/mysql-files/mydict','mydict');
+-----+
| gen_dictionary_load('/usr/local/mysql/mysql-files/mydict','mydict') |
+-----+
| Dictionary load success |
+-----+
mysql> SELECT gen_dictionary_load('/dev/null','null');
+-----+
| gen_dictionary_load('/dev/null','null') |
+-----+
| Dictionary load error |
+-----+
```

6.6 MySQL Enterprise Encryption

注記

MySQL Enterprise Encryption は、商用製品である MySQL Enterprise Edition に含まれる拡張機能です。商用製品の詳細は、<https://www.mysql.com/products/> を参照してください。

MySQL Enterprise Edition には、SQL レベルで OpenSSL 機能を公開する OpenSSL ライブラリに基づく一連の暗号化機能が含まれています。これらの関数を使用することによって、エンタープライズアプリケーションが次の操作を実行できるようになります。

- 公開鍵非対称暗号方式を使用した、追加のデータ保護の実装
- 公開鍵、秘密鍵、およびデジタル署名の作成

- 非対称暗号化および非対称復号化の実行
- デジタル署名およびデータの検証や妥当性検査に対する暗号化ハッシュの使用

MySQL Enterprise Encryption は、RSA、DSA および DH 暗号化アルゴリズムをサポートしています。

MySQL Enterprise Encryption はユーザー定義関数 (UDF) ライブラリとして提供され、個々の関数を個別にインストールできます。

6.6.1 MySQL Enterprise Encryption のインストール

MySQL Enterprise Encryption 関数は、プラグインディレクトリ (`plugin_dir` システム変数で指定されたディレクトリ) にインストールされているユーザー定義関数 (UDF) ライブラリファイルにあります。UDF ライブラリのベース名は `openssl_udf` であり、サフィクスはプラットフォームに依存します。たとえば、ファイル名は Linux では `openssl_udf.so`、Windows では `openssl_udf.dll` です。

ライブラリファイルから関数をインストールするには、`CREATE FUNCTION` ステートメントを使用します。ライブラリからすべての関数をロードするには、必要に応じてファイル名接尾辞を調整して、次の一連のステートメントを使用します:

```
CREATE FUNCTION asymmetric_decrypt RETURNS STRING
SONAME 'openssl_udf.so';
CREATE FUNCTION asymmetric_derive RETURNS STRING
SONAME 'openssl_udf.so';
CREATE FUNCTION asymmetric_encrypt RETURNS STRING
SONAME 'openssl_udf.so';
CREATE FUNCTION asymmetric_sign RETURNS STRING
SONAME 'openssl_udf.so';
CREATE FUNCTION asymmetric_verify RETURNS INTEGER
SONAME 'openssl_udf.so';
CREATE FUNCTION create_asymmetric_priv_key RETURNS STRING
SONAME 'openssl_udf.so';
CREATE FUNCTION create_asymmetric_pub_key RETURNS STRING
SONAME 'openssl_udf.so';
CREATE FUNCTION create_dh_parameters RETURNS STRING
SONAME 'openssl_udf.so';
CREATE FUNCTION create_digest RETURNS STRING
SONAME 'openssl_udf.so';
```

一度インストールすれば、何回サーバーを再起動しても UDF はインストールされたままです。UDF をアンロードするには、`DROP FUNCTION` ステートメントを使用します:

```
DROP FUNCTION asymmetric_decrypt;
DROP FUNCTION asymmetric_derive;
DROP FUNCTION asymmetric_encrypt;
DROP FUNCTION asymmetric_sign;
DROP FUNCTION asymmetric_verify;
DROP FUNCTION create_asymmetric_priv_key;
DROP FUNCTION create_asymmetric_pub_key;
DROP FUNCTION create_dh_parameters;
DROP FUNCTION create_digest;
```

`CREATE FUNCTION` および `DROP FUNCTION` ステートメントでは、関数名を小文字で指定する必要があります。これは、大文字と小文字のどちらでも使用できる関数の呼び出し時での使用とは異なります。

`mysql` データベースの場合、`CREATE FUNCTION` および `DROP FUNCTION` ステートメントでは、それぞれ `INSERT` および `DROP` 権限が必要です。

6.6.2 MySQL Enterprise Encryption の使用方法と例

アプリケーションで MySQL Enterprise Encryption を使用するには、実行する操作に適した関数を呼び出します。このセクションでは、代表的なタスクの実行方法を示します:

- [RSA 暗号化を使用した秘密キーと公開キーのペアの作成](#)
- [秘密キーを使用してデータを暗号化し、公開キーを使用して復号化](#)

- 文字列からダイジェストを生成
- キーペアでダイジェストを使用
- 対称キーの作成
- キー生成操作による CPU 使用率の制限

RSA 暗号化を使用した秘密キーと公開キーのペアの作成

```
-- Encryption algorithm; can be 'DSA' or 'DH' instead
SET @algo = 'RSA';
-- Key length in bits; make larger for stronger keys
SET @key_len = 1024;

-- Create private key
SET @priv = create_asymmetric_priv_key(@algo, @key_len);
-- Derive corresponding public key from private key, using same algorithm
SET @pub = create_asymmetric_pub_key(@algo, @priv);
```

鍵のペアを使用すると、データを暗号化および復号化したり、データを署名および検証したり、対称鍵を生成したりできるようになりました。

秘密キーを使用してデータを暗号化し、公開キーを使用して復号化

これには、鍵ペアのメンバーが RSA 鍵である必要があります。

```
SET @ciphertext = asymmetric_encrypt(@algo, 'My secret text', @priv);
SET @plaintext = asymmetric_decrypt(@algo, @ciphertext, @pub);
```

反対に、公開鍵を使用して暗号化し、秘密鍵を使用して復号化できます。

```
SET @ciphertext = asymmetric_encrypt(@algo, 'My secret text', @pub);
SET @plaintext = asymmetric_decrypt(@algo, @ciphertext, @priv);
```

いずれの場合でも、暗号化関数および復号化関数用に指定されたアルゴリズムは、鍵を生成したときに使用されたアルゴリズムと一致する必要があります。

文字列からダイジェストを生成

```
-- Digest type; can be 'SHA256', 'SHA384', or 'SHA512' instead
SET @dig_type = 'SHA224';

-- Generate digest string
SET @dig = create_digest(@dig_type, 'My text to digest');
```

キーペアでダイジェストを使用

キーペアを使用してデータに署名し、署名がダイジェストと一致することを確認できます。

```
-- Encryption algorithm; could be 'DSA' instead; keys must
-- have been created using same algorithm
SET @algo = 'RSA';

-- Generate signature for digest and verify signature against digest
SET @sig = asymmetric_sign(@algo, @dig, @priv, @dig_type);
-- Verify signature against digest
SET @verf = asymmetric_verify(@algo, @dig, @sig, @pub, @dig_type);
```

対称キーの作成

これには、共有対称シークレットを使用して作成される DH 秘密鍵/公開鍵が入力として必要です。キー長を `create_dh_parameters()` に渡してシークレットを作成し、そのシークレットを「キー長」として `create_asymmetric_priv_key()` に渡します。

```
-- Generate DH shared symmetric secret
```



```

SET @dhp = create_dh_parameters(1024);
-- Generate DH key pairs
SET @algo = 'DH';
SET @priv1 = create_asymmetric_priv_key(@algo, @dhp);
SET @pub1 = create_asymmetric_pub_key(@algo, @priv1);
SET @priv2 = create_asymmetric_priv_key(@algo, @dhp);
SET @pub2 = create_asymmetric_pub_key(@algo, @priv2);

-- Generate symmetric key using public key of first party,
-- private key of second party
SET @sym1 = asymmetric_derive(@pub1, @priv2);

-- Or use public key of second party, private key of first party
SET @sym2 = asymmetric_derive(@pub2, @priv1);

```

鍵文字列の値は、[SET](#)、[SELECT](#)、または [INSERT](#) を使用することで実行時に作成し、変数やテーブルに格納できます。

```

SET @priv1 = create_asymmetric_priv_key('RSA', 1024);
SELECT create_asymmetric_priv_key('RSA', 1024) INTO @priv2;
INSERT INTO t (key_col) VALUES(create_asymmetric_priv_key('RSA', 1024));

```

ファイルに格納されている鍵文字列の値は、[FILE](#) 権限を持つユーザーが [LOAD_FILE\(\)](#) 関数を使用することで読み取ることができます。

ダイジェストと署名の文字列は、同様に処理できます。

キー生成操作による CPU 使用率の制限

[create_asymmetric_priv_key\(\)](#) および [create_dh_parameters\(\)](#) 暗号化関数はキー長パラメータを取り、これらの関数に必要な CPU リソースの量はキーの長さが増加するにつれて増加します。一部のインストールでは、アプリケーションが過度に長いキーを頻繁に生成する場合、これによって CPU 使用率が許容できなくなることがあります。

OpenSSL では、すべてのキーに 1,024 ビットの最小キー長が課されます。OpenSSL では、[create_asymmetric_priv_key\(\)](#) では DSA キーおよび RSA キーに対してそれぞれ 10,000 ビットおよび 16,384 ビットの最大キー長、および [create_dh_parameters\(\)](#) では 10,000 ビットの最大キー長も課されます。これらの最大値が大きすぎる場合は、MySQL サーバー管理者がキー生成の最大長を低く設定できるようにするために、3 つの環境変数を使用できます。これにより、CPU 使用率を制限できます:

- [MYSQL_OPENSSL_UDF_DSA_BITS_THRESHOLD](#): [create_asymmetric_priv_key\(\)](#) の DSA キーの最大長 (ビット単位)。この変数の最小値と最大値は 1,024 と 10,000 です。
- [MYSQL_OPENSSL_UDF_RSA_BITS_THRESHOLD](#): [create_asymmetric_priv_key\(\)](#) の RSA キーの最大長 (ビット単位)。この変数の最小値と最大値は 1,024 と 16,384 です。
- [MYSQL_OPENSSL_UDF_DH_BITS_THRESHOLD](#): [create_dh_parameters\(\)](#) の最大キー長 (ビット単位)。この変数の最小値と最大値は 1,024 と 10,000 です。

これらの環境変数のいずれかを使用するには、サーバーを起動するプロセスの環境に設定します。設定されている場合、それらの値は OpenSSL によって課される最大キー長よりも優先されます。たとえば、DSA に 4,096 ビット、[create_asymmetric_priv_key\(\)](#) に RSA キーの最大キー長を設定するには、次の変数を設定します:

```

export MYSQL_OPENSSL_UDF_DSA_BITS_THRESHOLD=4096
export MYSQL_OPENSSL_UDF_RSA_BITS_THRESHOLD=4096

```

この例では Bourne シェル構文を使用します。ほかのシェルの構文は異なる場合があります。

6.6.3 MySQL Enterprise Encryption ユーザー定義関数リファレンス

表 6.39 「MySQL Enterprise Encryption のユーザー定義関数」

名前	説明
asymmetric_decrypt()	秘密鍵または公開鍵を使用して暗号文を復号化します

名前	説明
asymmetric_derive()	非対称鍵から対称鍵を導出します
asymmetric_encrypt()	秘密キーまたは公開キーを使用したクリアテキストの暗号化
asymmetric_sign()	ダイジェストから署名を生成します
asymmetric_verify()	署名がダイジェストと一致することを確認します
create_asymmetric_priv_key()	秘密鍵を作成します
create_asymmetric_pub_key()	公開鍵を作成します
create_dh_parameters()	共有 DH シークレットを生成します
create_digest()	文字列からダイジェストを生成します

6.6.4 MySQL Enterprise Encryption ユーザー定義関数の説明

MySQL Enterprise Encryption 関数には、次の一般的な特性があります:

- 引数の型が不正な場合や引数の数が間違っている場合は、各関数でエラーが返されます。
- 要求された演算を実行することを関数に許可するのに引数が適していない場合は、必要に応じて `NULL` または `0` が返されます。これは、指定されたアルゴリズムが関数でサポートされていない場合、鍵の長さが短すぎたり長すぎたりする場合、PEM 書式の鍵文字列として要求される文字列が有効な鍵でない場合などに発生します。(OpenSSL では独自のキー長制限が課され、サーバー管理者は環境変数を設定して最大キー長に追加の制限を課すことができます。 [セクション6.6.2「MySQL Enterprise Encryption の使用方法と例」](#)を参照してください。)
- ベースとなる SSL ライブラリでは、ランダム度の初期化が処理されます。

関数の一部には、暗号化アルゴリズムの引数が指定されます。次の表には、サポートされているアルゴリズムのサマリーを関数別に示します。

表 6.40 関数でサポートされているアルゴリズム

関数	サポートされているアルゴリズム
asymmetric_decrypt()	RSA
asymmetric_derive()	DH
asymmetric_encrypt()	RSA
asymmetric_sign()	RSA、DSA
asymmetric_verify()	RSA、DSA
create_asymmetric_priv_key()	RSA、DSA、DH
create_asymmetric_pub_key()	RSA、DSA、DH
create_dh_parameters()	DH

注記

RSA、DSA、または DH のいずれかの暗号化アルゴリズムを使用すれば鍵を作成できますが、鍵引数が指定されるその他の関数では、特定のタイプの鍵のみが許可される可能性があります。たとえば、[asymmetric_encrypt\(\)](#) および [asymmetric_decrypt\(\)](#) は RSA キーのみを受け入れます。

次の説明では、MySQL Enterprise Encryption 関数のコール順序について説明します。追加の例と説明については、[セクション6.6.2「MySQL Enterprise Encryption の使用方法と例」](#)を参照してください。

- [asymmetric_decrypt\(algorithm, crypt_str, key_str\)](#)

指定されたアルゴリズムおよび鍵文字列を使用して、暗号化された文字列を復号化し、結果として生成されるプレーンテキストをバイナリ文字列として返します。復号化に失敗した場合は、結果が `NULL` になります。

`key_str` は、PEM 書式の有効な鍵文字列である必要があります。復号化を成功させるには、`asymmetric_encrypt()` で暗号化文字列を生成するために使用される秘密キー文字列または公開キー文字列に対応する公開キー文字列または秘密キー文字列である必要があります。`algorithm` は、キーの作成に使用される暗号化アルゴリズムを示します。

サポートされている `algorithm` 値: 'RSA'

使用例については、`asymmetric_encrypt()` の説明を参照してください。

- `asymmetric_derive(pub_key_str, priv_key_str)`

あるパーティーの秘密鍵と別のパーティーの公開鍵を使用して対称鍵を導出し、結果として生成される鍵をバイナリ文字列として返します。鍵の抽出に失敗した場合は、結果が `NULL` になります。

`pub_key_str` および `priv_key_str` は、PEM 書式の有効な鍵文字列である必要があります。これらは、DH アルゴリズムを使用して作成する必要があります。

公開鍵と秘密鍵の 2 つのペアを持っていると仮定します。

```
SET @dhp = create_dh_parameters(1024);
SET @priv1 = create_asymmetric_priv_key('DH', @dhp);
SET @pub1 = create_asymmetric_pub_key('DH', @priv1);
SET @priv2 = create_asymmetric_priv_key('DH', @dhp);
SET @pub2 = create_asymmetric_pub_key('DH', @priv2);
```

さらに、1 つのペアから秘密鍵を使用し、もう 1 つのペアから公開鍵を使用して、対称鍵文字列を作成すると仮定します。その後、この対称鍵の ID 関係が次のように保持されます。

```
asymmetric_derive(@pub1, @priv2) = asymmetric_derive(@pub2, @priv1)
```

- `asymmetric_encrypt(algorithm, str, key_str)`

指定されたアルゴリズムおよび鍵文字列を使用して文字列を暗号化し、結果として生成される暗号化テキストをバイナリ文字列として返します。暗号化に失敗した場合は、結果が `NULL` になります。

`str` の長さは、バイト単位で `key_str` の長さ - 11 よりも大きくすることができません。

`key_str` は、PEM 書式の有効な鍵文字列にする必要があります。`algorithm` は、鍵を作成する際に使用された暗号化アルゴリズムを示します。

サポートされている `algorithm` 値: 'RSA'

文字列を暗号化するには、秘密キーまたは公開キーの文字列を `asymmetric_encrypt()` に渡します。元の暗号化されていない文字列をリカバリするには、暗号化に使用される秘密キー文字列または公開キー文字列に対応する公開キー文字列または秘密キー文字列とともに、暗号化された文字列を `asymmetric_decrypt()` に渡します。

```
-- Generate private/public key pair
SET @priv = create_asymmetric_priv_key('RSA', 1024);
SET @pub = create_asymmetric_pub_key('RSA', @priv);

-- Encrypt using private key, decrypt using public key
SET @ciphertext = asymmetric_encrypt('RSA', 'The quick brown fox', @priv);
SET @plaintext = asymmetric_decrypt('RSA', @ciphertext, @pub);

-- Encrypt using public key, decrypt using private key
SET @ciphertext = asymmetric_encrypt('RSA', 'The quick brown fox', @pub);
SET @plaintext = asymmetric_decrypt('RSA', @ciphertext, @priv);
```

次のように仮定します。

```
SET @s = a string to be encrypted
SET @priv = a valid private RSA key string in PEM format
SET @pub = the corresponding public RSA key string in PEM format
```

その後、これらの ID 関係が次のように保持されます。

```
asymmetric_decrypt('RSA', asymmetric_encrypt('RSA', @s, @priv), @pub) = @s
asymmetric_decrypt('RSA', asymmetric_encrypt('RSA', @s, @pub), @priv) = @s
```

- `asymmetric_sign(algorithm, digest_str, priv_key_str, digest_type)`

秘密鍵文字列を使用してダイジェスト文字列に署名し、その署名をバイナリ文字列として返します。署名に失敗した場合は、結果が `NULL` になります。

`digest_str` はダイジェスト文字列です。これは、`create_digest()` をコールして生成できます。`digest_type` は、ダイジェスト文字列の生成に使用されるダイジェストアルゴリズムを示します。

`priv_key_str` は、ダイジェスト文字列に署名する際に使用される秘密鍵文字列です。これは、PEM 書式の有効な鍵文字列にする必要があります。`algorithm` は、鍵を作成する際に使用された暗号化アルゴリズムを示します。

サポートされている `algorithm` 値: 'RSA'、'DSA'

サポートされている `digest_type` 値: 'SHA224'、'SHA256'、'SHA384'、'SHA512'

使用例については、`asymmetric_verify()` の説明を参照してください。

- `asymmetric_verify(algorithm, digest_str, sig_str, pub_key_str, digest_type)`

署名文字列がダイジェスト文字列と一致するかどうかを確認し、確認に成功したのか失敗したのかを示す 1 または 0 を返します。

`digest_str` はダイジェスト文字列です。これは、`create_digest()` をコールして生成できます。`digest_type` は、ダイジェスト文字列の生成に使用されるダイジェストアルゴリズムを示します。

`sig_str` は署名文字列です。これは、`asymmetric_sign()` をコールして生成できます。

`pub_key_str` は、署名者の公開鍵文字列です。署名文字列を生成するために `asymmetric_sign()` に渡される秘密キーに対応し、PEM 形式の有効なキー文字列である必要があります。`algorithm` は、キーの作成に使用される暗号化アルゴリズムを示します。

サポートされている `algorithm` 値: 'RSA'、'DSA'

サポートされている `digest_type` 値: 'SHA224'、'SHA256'、'SHA384'、'SHA512'

```
-- Set the encryption algorithm and digest type
SET @algo = 'RSA';
SET @dig_type = 'SHA224';

-- Create private/public key pair
SET @priv = create_asymmetric_priv_key(@algo, 1024);
SET @pub = create_asymmetric_pub_key(@algo, @priv);

-- Generate digest from string
SET @dig = create_digest(@dig_type, 'The quick brown fox');

-- Generate signature for digest and verify signature against digest
SET @sig = asymmetric_sign(@algo, @dig, @priv, @dig_type);
SET @verf = asymmetric_verify(@algo, @dig, @sig, @pub, @dig_type);
```

- `create_asymmetric_priv_key(algorithm, {key_len|dh_secret})`

指定されたアルゴリズムおよび鍵の長さまたは DH シークレットを使用して秘密鍵を作成し、その鍵を PEM 書式のバイナリ文字列として返します。鍵の生成に失敗した場合は、結果が `NULL` になります。

サポートされている `algorithm` 値: 'RSA'、'DSA'、'DH'

サポートされている `key_len` 値: 最小の鍵の長さは 1,024 ビットです。最大の鍵の長さはアルゴリズムによって異なり、RSA の場合は 16,384、DSA の場合は 10,000 です。これらのキー長制限は、OpenSSL によって課される制

約です。サーバー管理者は、環境変数を設定することで、最大キー長に追加の制限を課することができます。 [セクション6.6.2「MySQL Enterprise Encryption の使用方法と例」](#)を参照してください。

DH 鍵の場合は、キーの長さの代わりに、共有 DH シークレットを渡します。シークレットを作成するには、キーの長さを `create_dh_parameters()` に渡します。

この例では、2,048 ビットの DSA 秘密鍵を作成してから、その秘密鍵から公開鍵を導出します。

```
SET @priv = create_asymmetric_priv_key('DSA', 2048);
SET @pub = create_asymmetric_pub_key('DSA', @priv);
```

DH 鍵の生成を示す例については、`asymmetric_derive()` の説明を参照してください。

鍵の長さや暗号化アルゴリズムを選択する際の一般的ないくつかの考慮事項は、次のとおりです。

- 鍵のサイズとともに、公開鍵と秘密鍵の暗号化強度が増加しますが、鍵の生成時間も同様に増加します。
 - DH 鍵の生成時間は、RSA 鍵または RSA 鍵よりも大幅に長くなります。
 - 非対称暗号化関数は、対称関数よりも遅くなります。パフォーマンスが重要な要素であり、その関数が非常に頻繁に使用される場合は、対称暗号化を使用した方が適切です。たとえば、`AES_ENCRYPT()` および `AES_DECRYPT()` を使用することを検討してください。
- `create_asymmetric_pub_key(algorithm, priv_key_str)`

指定されたアルゴリズムを使用して、指定された秘密鍵から公開鍵を導出し、その鍵を PEM 書式のバイナリ文字列として返します。鍵の抽出に失敗した場合は、結果が `NULL` になります。

`priv_key_str` は、PEM 書式の有効な鍵文字列にする必要があります。`algorithm` は、鍵を作成する際に使用された暗号化アルゴリズムを示します。

サポートされている `algorithm` 値: 'RSA'、'DSA'、'DH'

使用例については、`create_asymmetric_priv_key()` の説明を参照してください。

- `create_dh_parameters(key_len)`

DH 秘密鍵と公開鍵のペアを生成するための共有シークレットを作成し、`create_asymmetric_priv_key()` に渡すことができるバイナリ文字列を返します。シークレットの生成に失敗した場合は、結果が `NULL` になります。

サポートされている `key_len` 値: 最小および最大の鍵の長さは、1,024 ビットおよび 10,000 ビットです。これらのキー長制限は、OpenSSL によって課される制約です。サーバー管理者は、環境変数を設定することで、最大キー長に追加の制限を課することができます。 [セクション6.6.2「MySQL Enterprise Encryption の使用方法と例」](#)を参照してください。

対称鍵の生成に戻り値を使用する方法を示す例は、`asymmetric_derive()` の説明を参照してください。

```
SET @dhp = create_dh_parameters(1024);
```

- `create_digest(digest_type, str)`

指定されたダイジェストタイプを使用して、指定された文字列からダイジェストを作成し、そのダイジェストをバイナリ文字列として返します。ダイジェストの生成に失敗した場合は、結果が `NULL` になります。

サポートされている `digest_type` 値: 'SHA224'、'SHA256'、'SHA384'、'SHA512'

```
SET @dig = create_digest('SHA512', 'The quick brown fox');
```

生成されるダイジェスト文字列は、`asymmetric_sign()` および `asymmetric_verify()` での使用に適しています。

6.7 SELinux

セキュリティ強化された Linux (SELinux) は、SELinux コンテキストと呼ばれるセキュリティラベルを各システムオブジェクトに適用することでアクセス権を実装する必須アクセス制御 (MAC) システムです。SELinux ポリシーモ

ジュールは、SELinux コンテキストを使用して、プロセス、ファイル、ポートおよびその他のシステムオブジェクトが相互作用する方法のルールを定義します。システムオブジェクト間の相互作用は、ポリシールールで許可されている場合にのみ許可されます。

SELinux コンテキスト (システムオブジェクトに適用されるラベル) には、次のフィールドがあります: `user`, `role`, `type` および `security level`。SELinux コンテキスト全体ではなく型情報は、プロセスが他のシステムオブジェクトと相互作用する方法のルールを定義するために最も一般的に使用されます。たとえば、MySQL SELinux ポリシーモジュールでは、`type` 情報を使用してポリシールールを定義します。

`ls` や `ps` などのオペレーティングシステムコマンドを `-Z` オプションとともに使用して、SELinux コンテキストを表示できます。SELinux が有効で、MySQL Server が実行されている場合、次のコマンドは `mysqld` プロセスおよび MySQL データディレクトリの SELinux コンテキストを表示します:

`mysqld` プロセス:

```
shell> ps -eZ | grep mysqld
system_u:system_r:mysqld_t:s0 5924 ? 00:00:03 mysqld
```

MySQL データディレクトリ:

```
shell> cd /var/lib
shell> ls -Z | grep mysql
system_u:object_r:mysqld_db_t:s0 mysql
```

ここでは:

- `system_u` は、システムプロセスおよびオブジェクトの SELinux ユーザーアイデンティティです。
- `system_r` は、システムプロセスに使用される SELinux ロールです。
- `objects_r` は、システムオブジェクトに使用される SELinux ロールです。
- `mysqld_t` は `mysqld` プロセスに関連付けられたタイプです。
- `mysqld_db_t` は、MySQL データディレクトリとそのファイルに関連付けられたタイプです。
- `s0` はセキュリティレベルです。

SELinux コンテキストの解釈の詳細は、ディストリビューションの SELinux ドキュメントを参照してください。

6.7.1 SELinux が有効かどうかの確認

Oracle Linux、RHEL、CentOS、Fedora などの一部の Linux ディストリビューションでは、SELinux がデフォルトで有効になっています。 `sestatus` コマンドを使用して、ディストリビューションで SELinux が有効になっているかどうかを確認します:

```
shell> sestatus
SELinux status:                enabled
SELinuxfs mount:              /sys/fs/selinux
SELinux root directory:      /etc/selinux
Loaded policy name:          targeted
Current mode:                 enforcing
Mode from config file:       enforcing
Policy MLS status:           enabled
Policy deny_unknown status:  allowed
Memory protection checking:   actual (secure)
Max kernel policy version:    31
```

SELinux が無効になっているか、`sestatus` コマンドが見つからない場合は、SELinux を有効にする前にディストリビューションの SELinux ドキュメントを参照してガイダンスを確認してください。

6.7.2 SELinux モードの変更

SELinux は、強制モード、許可モードおよび無効モードをサポートしています。Enforcing モードがデフォルトです。許可モードでは、強制モードで許可されていない操作が許可され、それらの操作が SELinux 監査ログに記録されます。通常、許可モードはポリシーの開発時またはトラブルシューティング時に使用されます。無効モードでは、ポ

リシーは強制されず、コンテキストはシステムオブジェクトに適用されないため、後で SELinux を有効にすることは困難です。

現在の SELinux モードを表示するには、前述の `sestatus` コマンドまたは `getenforce` ユーティリティを使用します。

```
shell> getenforce
Enforcing
```

SELinux モードを変更するには、`setenforce` ユーティリティを使用します:

```
shell> setenforce 0
shell> getenforce
Permissive
```

```
shell> setenforce 1
shell> getenforce
Enforcing
```

`setenforce` で行われた変更は、システムを再起動すると失われます。SELinux モードを永続的に変更するには、`/etc/selinux/config` ファイルを編集してシステムを再起動します。

6.7.3 MySQL Server SELinux ポリシー

通常、MySQL Server SELinux ポリシーモジュールはデフォルトでインストールされます。`semodule -l` コマンドを使用して、インストールされているモジュールを表示できます。MySQL Server SELinux ポリシーモジュールには、次のものがあります:

- `mysqld_selinux`
- `mysqld_safe_selinux`

MySQL Server SELinux ポリシーモジュールの詳細は、SELinux マニュアルページを参照してください。マニュアルページには、MySQL サービスに関連付けられたタイプおよびブールに関する情報が表示されます。マニュアルページには、`service-name_selinux` 形式で名前が付けられます。

```
man mysqld_selinux
```

SELinux のマニュアルページが使用できない場合は、`sepolicy manpage` ユーティリティーを使用してマニュアルページを生成する方法について、配布 SELinux のドキュメントを参照してください。

6.7.4 SELinux ファイルコンテキスト

MySQL Server は、多くのファイルに対して読取りおよび書込みを行います。これらのファイルに対して SELinux コンテキストが正しく設定されていない場合、ファイルへのアクセスが拒否される可能性があります。

次の手順では、`semanage` バイナリを使用してファイルコンテキストを管理します。RHEL では、これは `policycoreutils-python-utils` パッケージの一部です:

```
yum install -y policycoreutils-python-utils
```

`semanage` バイナリをインストールした後、`semanage` と `fcontext` オプションを使用して MySQL ファイルコンテキストをリストできます。

```
semanage fcontext -l | grep -i mysql
```

MySQL データディレクトリコンテキストの設定

デフォルトのデータディレクトリの場所は `/var/lib/mysql/` で、使用される SELinux コンテキストは `mysqld_db_t` です。

構成ファイルを編集して、データディレクトリまたはデータディレクトリ内の通常のファイル (バイナリログなど) に別の場所を使用する場合は、新しい場所のコンテキストを設定する必要がある場合があります。例:

```
semanage fcontext -a -t mysqld_db_t "/path/to/my/custom/datadir(/.*)?"
restorecon -Rv /path/to/my/custom/datadir

semanage fcontext -a -t mysqld_db_t "/path/to/my/custom/logdir(/.*)?"
```

```
restorecon -Rv /path/to/my/custom/logdir
```

MySQL エラーログファイルコンテキストの設定

RedHat RPM のデフォルトの場所は `/var/log/mysql.log` で、使用される SELinux コンテキストタイプは `mysqld_log_t` です。

構成ファイルを編集して別の場所を使用する場合は、新しい場所のコンテキストを設定する必要がある場合があります。例:

```
semanage fcontext -a -t mysqld_log_t "/path/to/my/custom/error.log"
restorecon -Rv /path/to/my/custom/error.log
```

PID ファイルコンテキストの設定

PID ファイルのデフォルトの場所は `/var/run/mysql/mysql.pid` で、使用される SELinux コンテキストタイプは `mysqld_var_run_t` です。

構成ファイルを編集して別の場所を使用する場合は、新しい場所のコンテキストを設定する必要がある場合があります。例:

```
semanage fcontext -a -t mysqld_var_run_t "/path/to/my/custom/pidfile/directory/*?"
restorecon -Rv /path/to/my/custom/pidfile/directory
```

Unix ドメインソケットコンテキストの設定

Unix ドメインソケットのデフォルトの場所は `/var/lib/mysql/mysql.sock` で、使用される SELinux コンテキストタイプは `mysqld_var_run_t` です。

構成ファイルを編集して別の場所を使用する場合は、新しい場所のコンテキストを設定する必要がある場合があります。例:

```
semanage fcontext -a -t mysqld_var_run_t "/path/to/my/custom/mysql\socket"
restorecon -Rv /path/to/my/custom/mysql.sock
```

secure_file_priv ディレクトリコンテキストの設定

5.6.34、5.7.16 および 8.0.11 以降の MySQL バージョンの場合。

MySQL Server RPM をインストールすると、`/var/lib/mysql-files/` ディレクトリが作成されますが、SELinux コンテキストは設定されません。`/var/lib/mysql-files/` ディレクトリは、`SELECT ... INTO OUTFILE` などの操作に使用されます。

`secure_file_priv` を設定してこのディレクトリの使用を有効にした場合、次のようにコンテキストを設定する必要があります:

```
semanage fcontext -a -t mysqld_db_t "/var/lib/mysql-files/(.*)?"
restorecon -Rv /var/lib/mysql-files
```

別の場所を使用した場合は、このパスを編集します。セキュリティ上の理由から、このディレクトリはデータディレクトリ内にはありません。

この変数の詳細は、`secure_file_priv` のドキュメントを参照してください。

6.7.5 SELinux TCP ポートコンテキスト

次の手順では、`semanage` バイナリを使用してポートコンテキストを管理します。RHEL では、これは `policycoreutils-python-utils` パッケージの一部です:

```
yum install -y policycoreutils-python-utils
```

`semanage` バイナリをインストールした後、`semanage` を `port` オプションとともに使用して、`mysqld_port_t` コンテキストで定義されたポートをリストできます。

```
shell> semanage port -l | grep mysqld
mysqld_port_t      tcp      1186, 3306, 63132-63164
```

6.7.5.1 mysqld の TCP ポートコンテキストの設定

`mysqld` のデフォルトの TCP ポートは `3306` で、使用される SELinux コンテキストタイプは `mysqld_port_t` です。

別の TCP `port` を使用するように `mysqld` を構成する場合は、新しいポートのコンテキストを設定する必要がある場合があります。たとえば、ポート `3307` など、デフォルト以外のポートの SELinux コンテキストを定義するには、次のようにします:

```
semanage port -a -t mysqld_port_t -p tcp 3307
```

ポートが追加されたことを確認するには:

```
shell> semanage port -l | grep mysqld
mysqld_port_t      tcp      3307, 1186, 3306, 63132-63164
```

6.7.5.2 MySQL 機能の TCP ポートコンテキストの設定

特定の MySQL 機能を有効にする場合は、それらの機能で使用される追加ポートの SELinux TCP ポートコンテキストの設定が必要になることがあります。MySQL 機能で使用されるポートに正しい SELinux コンテキストがない場合、機能が正しく機能しない可能性があります。

次の各セクションでは、MySQL 機能のポートコンテキストを設定する方法について説明します。通常、同じメソッドを使用して、MySQL 機能のポートコンテキストを設定できます。MySQL 機能で使用されるポートの詳細は、[MySQL Port Reference](#) を参照してください。

MySQL 8.0.14 から MySQL 8.0.17 へ、`mysql_connect_any` SELinux プールを `ON` に設定する必要があります。MySQL 8.0.18 では、`mysql_connect_any` の有効化は必須ではなく、推奨されません。

```
setsebool -P mysql_connect_any=ON
```

グループレプリケーションの TCP ポートコンテキストの設定

SELinux が有効な場合は、`group_replication_local_address` 変数で定義されている Group Replication 通信ポートのポートコンテキストを設定する必要があります。`mysqld` は、Group Replication 通信ポートにバインドし、そこでリスニングする必要があります。InnoDB クラスタはグループレプリケーションに依存するため、これはクラスタで使用されるインスタンスにも同様に適用されます。MySQL で現在使用されているポートを表示するには、次のコマンドを発行します:

```
semanage port -l | grep mysqld
```

Group Replication 通信ポートが `33061` の場合は、次を発行してポートコンテキストを設定します:

```
semanage port -a -t mysqld_port_t -p tcp 33061
```

ドキュメントストアの TCP ポートコンテキストの設定

SELinux が有効になっている場合は、`mysqlx_port` 変数で定義されている X プラグインで使用される通信ポートのポートコンテキストを設定する必要があります。`mysqld` は、X プラグイン 通信ポートにバインドし、そこでリスニングする必要があります。

X プラグイン 通信ポートが `33060` の場合は、次を発行してポートコンテキストを設定します:

```
semanage port -a -t mysqld_port_t -p tcp 33060
```

MySQL Router の TCP ポートコンテキストの設定

SELinux が有効な場合は、MySQL Router で使用される通信ポートのポートコンテキストを設定する必要があります。MySQL Router で使用される追加の通信ポートがデフォルトの `6446`、`6447`、`64460` および `64470` であると仮定すると、各インスタンスで次を発行してポートコンテキストを設定します:

```
semanage port -a -t mysqld_port_t -p tcp 6446
semanage port -a -t mysqld_port_t -p tcp 6447
semanage port -a -t mysqld_port_t -p tcp 64460
semanage port -a -t mysqld_port_t -p tcp 64470
```

6.7.6 SELinux のトラブルシューティング

SELinux のトラブルシューティングでは、通常、SELinux を許容モードにし、問題のある操作を再実行し、SELinux 監査ログでアクセス拒否メッセージをチェックし、問題が解決した後に SELinux を強制モードに戻します。

`setenforce` を使用してシステム全体を許可モードにしないようにするには、`semanage` コマンドを使用して SELinux ドメイン (`mysqld_t`) を許可モードにすることで、MySQL サービスのみを許可できます:

```
semanage permissive -a mysqld_t
```

トラブルシューティングが終了したら、次のコマンドを使用して `mysqld_t` ドメインを強制モードに戻します:

```
semanage permissive -d mysqld_t
```

SELinux は、拒否された操作のログを `/var/log/audit/audit.log` に書き込みます。拒否を確認するには、「拒否」メッセージを検索します。

```
grep "denied" /var/log/audit/audit.log
```

次の各セクションでは、SELinux 関連の問題が発生する可能性のあるいくつかの一般的な領域について説明します。

ファイルコンテキスト

MySQL ディレクトリまたはファイルの SELinux コンテキストが正しくない場合、アクセスが拒否される可能性があります。この問題は、MySQL がデフォルト以外のディレクトリまたはファイルに対して読み取りまたは書き込みを行うように構成されている場合に発生することがあります。たとえば、デフォルト以外のデータディレクトリを使用するように MySQL を構成する場合、ディレクトリには予期される SELinux コンテキストがない可能性があります。

無効な SELinux コンテキストを持つデフォルト以外のデータディレクトリで MySQL サービスを起動しようとすると、次の起動が失敗します。

```
shell> systemctl start mysql.service
Job for mysqld.service failed because the control process exited with error code.
See "systemctl status mysql.service" and "journalctl -xe" for details.
```

この場合、「否認」メッセージは `/var/log/audit/audit.log` に記録されます:

```
shell> grep "denied" /var/log/audit/audit.log
type=AVC msg=audit(1587133719.786:194): avc: denied { write } for pid=7133 comm="mysqld"
name="mysqld" dev="dm-0" ino=51347078 scontext=system_u:system_r:mysqld_t:s0
tcontext=unconfined_u:object_r:default_t:s0 tclass=dir permissive=0
```

MySQL ディレクトリおよびファイルに対する適切な SELinux コンテキストの設定の詳細は、[セクション 6.7.4 「SELinux ファイルコンテキスト」](#) を参照してください。

ポートアクセス

SELinux は、MySQL Server などのサービスが特定のポートを使用することを想定しています。SELinux ポリシーを更新せずにポートを変更すると、サービス障害が発生する場合があります。

`mysqld_port_t` ポートタイプは、MySQL がリスニングするポートを定義します。ポート 3307 などのデフォルト以外のポートを使用するように MySQL Server を構成し、変更を反映するようにポリシーを更新しない場合、MySQL サービスは起動に失敗します:

```
shell> systemctl start mysqld.service
Job for mysqld.service failed because the control process exited with error code.
See "systemctl status mysqld.service" and "journalctl -xe" for details.
```

この場合、拒否メッセージが `/var/log/audit/audit.log` に記録されます:

```
shell> grep "denied" /var/log/audit/audit.log
type=AVC msg=audit(1587134375.845:198): avc: denied { name_bind } for pid=7340
comm="mysqld" src=3307 scontext=system_u:system_r:mysqld_t:s0
tcontext=system_u:object_r:unreserved_port_t:s0 tclass=tcp_socket permissive=0
```

MySQL の適切な SELinux ポートコンテキストの設定の詳細は、[セクション 6.7.5 「SELinux TCP ポートコンテキスト」](#) を参照してください。必要なコンテキストで定義されていないポートを使用する MySQL 機能を有効にすると、

同様のポートアクセスの問題が発生する可能性があります。詳細は、[セクション6.7.5.2「MySQL 機能の TCP ポートコンテキストの設定」](#)を参照してください。

アプリケーションの変更

SELinux は、アプリケーションの変更を認識しない場合があります。たとえば、新しいリリース、アプリケーション拡張機能または新機能が SELinux で許可されていない方法でシステムリソースにアクセスすると、アクセスが拒否される場合があります。このような場合は、[audit2allow](#) ユーティリティを使用して、必要に応じてアクセスを許可するカスタムポリシーを作成できます。カスタムポリシーを作成する一般的な方法は、SELinux モードを `permissive` に変更し、SELinux 監査ログでアクセス拒否メッセージを識別し、[audit2allow](#) ユーティリティを使用してアクセスを許可するカスタムポリシーを作成することです。

[audit2allow](#) ユーティリティの使用の詳細は、[ディストリビューションの SELinux ドキュメント](#)を参照してください。

標準の MySQL SELinux ポリシーモジュールで処理する必要があると思われる MySQL のアクセスの問題が発生した場合は、[配布バグ追跡システム](#)でバグレポートを開きます。

6.8 FIPS のサポート

OpenSSL 1.0.2 を使用してコンパイルされた場合、MySQL は FIPS モードをサポートし、OpenSSL ライブラリおよび FIPS オブジェクトモジュールを実行時に使用できます。

サーバー側の FIPS モードは、サーバーによって実行される暗号化操作に適用されます。これには、サーバー内で実行されるレプリケーション (ソース/レプリカおよびグループレプリケーション) および X プラグインが含まれます。FIPS モードは、クライアントによるサーバーへの接続試行にも適用されます。

次の各セクションでは、FIPS モードと、MySQL 内で FIPS モードを利用する方法について説明します:

- [FIPS の概要](#)
- [MySQL での FIPS モードのシステム要件](#)
- [MySQL での FIPS モードの構成](#)

FIPS の概要

Federal Information Processing Standards 140-2 (FIPS 140-2) では、機密情報または貴重な情報を保護するために使用される暗号化モジュールのために連邦政府 (US Government) 機関で必要とされるセキュリティ標準について説明します。このような連邦政府で使用できるとみなされるには、FIPS 140-2 で暗号化モジュールが動作保証されている必要があります。機密データを保護するシステムに適切な FIPS 140-2 証明書がない場合、連邦政府機関はそれを購入できません。

OpenSSL などの製品は FIPS モードで使用できますが、OpenSSL ライブラリ自体は FIPS に対して検証されません。かわりに、OpenSSL ライブラリを OpenSSL FIPS オブジェクトモジュールとともに使用して、OpenSSL ベースのアプリケーションが FIPS モードで動作できるようにします。

FIPS と OpenSSL でのその実装に関する一般的な情報については、次の参考資料を参照してください:

- [国立標準技術研究所 FIPS PUB 140-2](#)
- [OpenSSL FIPS 140-2 のセキュリティポリシー](#)
- [OpenSSL FIPS Object Module v 2.0 のユーザーガイド](#)

重要

FIPS モードでは、許容される暗号化アルゴリズムの制限や長いキー長の要件などの暗号化操作に条件が適用されます。OpenSSL の場合、FIPS の正確な動作は OpenSSL のバージョンによって異なります。詳細は、[OpenSSL FIPS User Guide](#)を参照してください。

MySQL での FIPS モードのシステム要件

MySQL で FIPS モードをサポートするには、次のシステム要件を満たす必要があります:

- ビルド時に、MySQL は OpenSSL を使用してコンパイルする必要があります。コンパイルで OpenSSL とは異なる SSL ライブラリが使用されている場合、FIPS モードは MySQL では使用できません。

また、FIPS での使用が動作保証されている OpenSSL バージョンで MySQL をコンパイルする必要があります。OpenSSL 1.0.2 は認定されていますが、OpenSSL 1.1.1 は認定されていません。最新バージョンの MySQL のバイナリディストリビューションは、一部のプラットフォームでは OpenSSL 1.1.1 を使用してコンパイルされます。つまり、FIPS に対して動作保証されていません。これにより、システムおよび MySQL の構成に応じて、使用可能な MySQL 機能がトレードオフされます。

- OpenSSL 1.0.2 および必要な FIPS オブジェクトモジュールを備えたシステムを使用します。この場合、OpenSSL 1.0.2 を使用してコンパイルされたバイナリ配布を使用するか、OpenSSL 1.0.2 を使用してソースから MySQL をコンパイルすると、MySQL の FIPS モードを有効にできます。ただし、この場合、TLSv1.3 プロトコルまたは暗号スイート (OpenSSL 1.1.1 が必要) は使用できません。さらに、2019 年末に End of Life ステータスに達した OpenSSL バージョンを使用しています。
- OpenSSL 1.1.1 以上のシステムを使用します。この場合、バイナリパッケージを使用して MySQL をインストールでき、すでにサポートされている他の TLS プロトコルに加えて、TLSv1.3 プロトコルおよび暗号スイートを使用できます。ただし、MySQL で FIPS モードを有効にすることはできません。
- 実行時に、OpenSSL ライブラリおよび OpenSSL FIPS オブジェクトモジュールが共有 (動的にリンクされた) オブジェクトとして使用可能である必要があります。静的にリンクされた OpenSSL オブジェクトは構築できますが、MySQL では使用できません。

FIPS モードは、EL7 上の MySQL に対してテストされていますが、他のシステムでも動作する可能性があります。

プラットフォームまたはオペレーティングシステムで OpenSSL FIPS オブジェクトモジュールが提供されている場合は、それを使用できます。それ以外の場合は、ソースから OpenSSL ライブラリおよび FIPS オブジェクトモジュールをビルドできます。OpenSSL FIPS ユーザーガイド ([FIPS の概要](#) を参照) の手順を使用します。

MySQL での FIPS モードの構成

MySQL では、サーバー側とクライアント側で FIPS モードを制御できます:

- `ssl_fips_mode` システム変数は、サーバーが FIPS モードで動作するかどうかを制御します。
- `--ssl-fips-mode` クライアントオプションは、特定の MySQL クライアントが FIPS モードで動作するかどうかを制御します。

`ssl_fips_mode` システム変数および `--ssl-fips-mode` クライアントオプションでは、次の値が許可されます:

- OFF**: FIPS モードを無効にします。
- ON**: FIPS モードを有効にします。
- STRICT**: 「strict」 FIPS モードを有効にします。

サーバー側では、`ssl_fips_mode` の数値 0、1 および 2 は **OFF**、**ON** および **STRICT** と同等です。

重要

一般に、**STRICT** には **ON** よりも多くの制限がありますが、MySQL 自体には FIPS モード値を OpenSSL に指定する以外に FIPS 固有のコードはありません。**ON** または **STRICT** の FIPS モードの正確な動作は、OpenSSL のバージョンによって異なります。詳細は、OpenSSL FIPS ユーザーガイド ([FIPS の概要](#) を参照) を参照してください。

注記

OpenSSL FIPS オブジェクトモジュールが使用できない場合、`ssl_fips_mode` および `--ssl-fips-mode` で許可される値は **OFF** のみです。FIPS モードを別の値に設定しようとすると、エラーが発生します。

サーバー側の FIPS モードは、サーバーによって実行される暗号化操作に適用されます。これには、サーバー内で実行されるレプリケーション (ソース/レプリカおよびグループレプリケーション) および X プラグインが含まれます。

FIPS モードは、クライアントによるサーバーへの接続試行にも適用されます。有効にすると、クライアント側またはサーバー側のいずれかで、サポートされている暗号化方式のどれを選択できるかが制限されます。ただし、FIPS モードを有効にする場合、暗号化された接続を使用する必要はありません。また、ユーザー資格証明を暗号化する必要もあります。たとえば、FIPS モードが有効になっている場合は、より強力な暗号化アルゴリズムが必要です。特に、MD5 は制限されているため、[RC4-MD5](#) などの暗号化暗号を使用して暗号化された接続を確立しようとしても機能しません。ただし、FIPS モードには、暗号化されていない接続の確立を妨げるものではありません。(これを行うには、特定のユーザーアカウントに対して [CREATE USER](#) または [ALTER USER](#) の [REQUIRE](#) 句を使用するか、すべてのアカウントに影響するように [require_secure_transport](#) システム変数を設定します。)

第 7 章 バックアップとリカバリ

目次

7.1 バックアップとリカバリの種類	1406
7.2 データベースバックアップ方法	1409
7.3 バックアップおよびリカバリ戦略の例	1410
7.3.1 バックアップポリシーの確立	1411
7.3.2 リカバリへのバックアップの使用	1413
7.3.3 バックアップ戦略サマリー	1413
7.4 バックアップへの mysqldump の使用	1414
7.4.1 mysqldump による SQL フォーマットでのデータのダンプ	1414
7.4.2 SQL フォーマットバックアップのリロード	1415
7.4.3 mysqldump による区切りテキストフォーマットでのデータのダンプ	1416
7.4.4 区切りテキストフォーマットバックアップのリロード	1417
7.4.5 mysqldump のヒント	1417
7.5 Point-in-Time (増分) リカバリ	1419
7.5.1 バイナリログを使用したポイントインタイムリカバリ	1419
7.5.2 イベントの位置を使用したポイントインタイムリカバリ	1421
7.6 MyISAM テーブルの保守とクラッシュリカバリ	1422
7.6.1 クラッシュリカバリへの myisamchk の使用	1423
7.6.2 MyISAM テーブルのエラーのチェック方法	1424
7.6.3 MyISAM テーブルの修復方法	1424
7.6.4 MyISAM テーブルの最適化	1426
7.6.5 MyISAM テーブル保守スケジュールのセットアップ	1427

システムクラッシュ、ハードウェアの障害、またはユーザーが誤ってデータを削除するなどの問題が発生した場合に、データをリカバリし、再度起動して、実行できるように、データベースをバックアップすることが重要です。バックアップは、MySQL インストールをアップグレードする前の保護手段としても不可欠であり、MySQL インストールを別のシステムに転送したり、レプリカサーバーを設定するために使用できます。

MySQL では、多様なバックアップ戦略を提供しており、それらからインストールの要件にもっとも適合する方法を選択できます。この章では、熟知しておくべきであるいくつかのバックアップとリカバリのトピックについて説明します。

- バックアップの種類: 論理と物理、完全と増分など。
- バックアップの作成の方法。
- ポイントインタイムリカバリを含むリカバリ方法。
- バックアップのスケジュールリング、圧縮、および暗号化。
- 破損したテーブルのリカバリを可能にするためのテーブルの保守。

追加のリソース

バックアップまたはデータの可用性の維持に関連するリソースには次のものが含まれます。

- MySQL Enterprise Edition の顧客は、MySQL Enterprise Backup 製品をバックアップに使用できます。MySQL Enterprise Backup 製品の概要については、[セクション30.2「MySQL Enterprise Backup の概要」](#)を参照してください。
- バックアップの問題に特化したフォーラムは <https://forums.mysql.com/list.php?28> にあります。
- `mysqldump` の詳細は、[第4章「MySQL プログラム」](#)にあります。
- ここで説明している SQL ステートメントの構文は、[第13章「SQL ステートメント」](#)にあります。

- InnoDB バックアップ手順の追加情報については、[セクション15.18.1「InnoDB バックアップ」](#)を参照してください。
- レプリケーションにより、複数のサーバーで同一のデータを保持できます。これには、サーバー間でのクライアントクエリーロードの分散の有効化、特定のサーバーがオフライン化または障害が発生した場合でもデータの可用性、レプリカを使用してソースに影響を与えずにバックアップを作成する機能など、いくつかの利点があります。[第17章「レプリケーション」](#)を参照してください。
- MySQL InnoDB クラスターは、連携して高可用性ソリューションを提供する製品の集合です。MySQL サーバーのグループは、MySQL Shell を使用してクラスターを作成するように構成できます。サーバーのクラスターにはプライマリと呼ばれる単一のソースがあり、読取り/書き込みソースとして機能します。複数のセカンダリサーバーがソースの複製です。高可用性クラスターを作成するには、少なくとも3つのサーバーが必要です。クライアントアプリケーションは、MySQL Router を介してプライマリに接続されます。プライマリに障害が発生すると、セカンダリはプライマリのロールに自動的に昇格され、MySQL Router はリクエストを新しいプライマリにルーティングします。
- NDB Cluster は、分散コンピューティング環境に適合した高可用性高冗長性バージョンの MySQL を提供します。MySQL NDB Cluster 8.0 に関する情報を提供する [第23章「MySQL NDB Cluster 8.0」](#)を参照してください。

7.1 バックアップとリカバリの種類

このセクションでは、さまざまな種類のバックアップの特性について説明します。

物理 (raw) バックアップと論理バックアップ

物理バックアップは、データベースの内容を格納するディレクトリとファイルの raw コピーから構成されます。この種類のバックアップは、問題の発生時に早急にリカバリさせる必要がある大規模で重要なデータベースに適しています。

論理バックアップは、論理データベース構造として表される情報 ([CREATE DATABASE](#)、[CREATE TABLE](#) ステートメント) と内容 ([INSERT](#) ステートメントまたは区切りテキストファイル) を保存します。この種類のバックアップは、ユーザーがデータ値やテーブル構造を編集したり、別のマシンアーキテクチャーにデータを再作成したりできる少量のデータに適しています。

物理バックアップ方法にはこれらの特性があります。

- バックアップはデータベースディレクトリおよびファイルの正確なコピーから構成されます。一般的に、これは MySQL データディレクトリのすべてまたは一部のコピーです。
- 物理バックアップ方法は、変換しないファイルコピーのみが含まれるため、論理より高速です。
- 出力は論理バックアップの場合よりコンパクトです。
- ビジーで、重要なデータベースには、バックアップの速度やコンパクトさが重要であるため、MySQL Enterprise Backup 製品は物理バックアップを実行します。MySQL Enterprise Backup 製品の概要については、[セクション 30.2「MySQL Enterprise Backup の概要」](#)を参照してください。
- バックアップとリストアの粒度は、データディレクトリ全体のレベルから個々のファイルのレベルまでの範囲になります。これは、ストレージエンジンに応じて、テーブルレベルの粒度を提供する場合としない場合があります。たとえば、InnoDB テーブルは、それぞれ個別のファイルにしたり、ほかの InnoDB テーブルとファイルストレージを共有したりできます。各 MyISAM テーブルはファイルのセットに一意に対応します。
- データベースに加えて、バックアップにはログファイルや構成ファイルなどの関連ファイルを含めることができます。
- MEMORY テーブルの内容はディスクに格納されないため、それらのデータをこの方法でバックアップすることは困難です。(MySQL Enterprise Backup 製品には、バックアップ中に MEMORY テーブルからデータを取得できる機能があります。)
- バックアップは、同一か類似のハードウェア特性を持つほかのマシンにのみ移植可能です。
- バックアップは MySQL サーバーが実行していない間に実行できます。サーバーが実行中の場合は、バックアップ中にサーバーがデータベースの内容を変更しないように、適切なロックを実行する必要があります。MySQL Enterprise Backup は、このロックが必要なテーブルに対して、自動的にロックを実行します。

- 物理バックアップツールには、InnoDB の MySQL Enterprise Backup の `mysqlbackup` またはその他のテーブル、あるいは MyISAM テーブルのファイルシステムレベルのコマンド (`cp`, `scp`, `tar`, `rsync` など) が含まれます。
- リストアの場合:
 - MySQL Enterprise Backup はバックアップした InnoDB およびその他のテーブルをリストアします。
 - `ndb_restore` は NDB テーブルをリストアします。
 - ファイルシステムレベルでコピーされたファイルは、ファイルシステムコマンドを使用して元の場所にコピーできます。

論理バックアップ方法にはこれらの特性があります。

- バックアップは、MySQL サーバーをクエリーし、データベース構造と内容情報を取得して実行されます。
- サーバーがデータベース情報にアクセスし、それを論理フォーマットに変換する必要があるため、バックアップは物理方法より遅くなります。クライアント側で出力が書き込まれた場合、サーバーはそれをバックアッププログラムに送信する必要もあります。
- 出力は特にテキストフォーマットで保存された場合に物理バックアップより大きくなります。
- バックアップとリストアの粒度は、サーバーレベル (すべてのデータベース)、データベースレベル (特定のデータベースのすべてのテーブル)、またはテーブルレベルで利用できます。これはストレージエンジンに関係なく当てはまります。
- バックアップには、ログファイルや構成ファイル、またはデータベースの一部ではないその他のデータベース関連ファイルは含まれません。
- 論理フォーマットで格納されているバックアップはマシンに依存せず、高度に移植可能です。
- 論理バックアップは MySQL サーバーの実行中に実行されます。サーバーはオフラインにされません。
- 論理バックアップツールには、`mysqldump` プログラムと `SELECT ... INTO OUTFILE` ステートメントが含まれます。これらは `MEMORY` でも、すべてのストレージエンジンで機能します。
- 論理バックアップをリストアするには、`mysql` クライアントを使用して、SQL フォーマットダンプファイルを処理できます。デリミタ付きテキストファイルをロードするには、`LOAD DATA` ステートメントまたは `mysqlimport` クライアントを使用します。

オンラインバックアップとオフラインバックアップ

オンラインバックアップは、データベース情報をサーバーから取得できるように、MySQL サーバーが実行中に行われます。オフラインバックアップは、サーバーが停止中に行われます。この区別は、「ホット」バックアップと「コールド」バックアップとして表すこともできます。「ウォーム」バックアップは、サーバーが実行したままですが、外部からデータベースファイルにアクセスしている間のデータの変更に対してロックされます。

オンラインバックアップ方法にはこれらの特性があります。

- このバックアップはほかのクライアントの邪魔になりにくく、クライアントはバックアップ中に MySQL サーバーに接続でき、実行する必要がある操作に応じて、データにアクセスできます。
- バックアップの完全性を損なう可能性のあるデータの変更が行われないように、適切なロックを適用する場合は、注意を払う必要があります。MySQL Enterprise Backup 製品はそのようなロックを自動的に実行します。

オフラインバックアップ方法にはこれらの特性があります。

- バックアップ中にサーバーを使用できないため、クライアントは影響を受ける可能性があります。そのため、このようなバックアップは、可用性を損なわずにオフラインにできるレプリカから取得されることがよくあります。
- バックアップ手順は、クライアントのアクティビティーからの干渉の可能性がないため単純になります。

オンラインとオフラインの同様の違いは、リカバリ操作にも当てはまり、同様の特性が当てはまります。ただし、リカバリにはより強力なロックが必要であるため、オンラインバックアップよりもオンラインリカバリの影響を受ける

可能性が高くなります。バックアップ時、クライアントはバックアップ中にデータを読み取ることができます。リカバリはデータを変更し、読み取るだけではないため、データのリストア中は、クライアントのデータへのアクセスを妨げる必要があります。

ローカルバックアップとリモートバックアップ

ローカルバックアップは MySQL サーバーが実行している同じホストで実行され、リモートバックアップは別のホストから実行されます。特定の種類のバックアップでは、出力がサーバーホストにローカルで書きこまれる場合でも、バックアップをリモートホストから開始できます。

- `mysqldump` はローカルまたはリモートサーバーに接続できます。SQL 出力 (`CREATE` および `INSERT` ステートメント) の場合、ローカルまたはリモートダンプを実行でき、クライアント上に出力が生成されます。区切りテキスト出力 (`--tab` オプションを使用して) の場合、サーバーホスト上にデータファイルが作成されます。
- `SELECT ... INTO OUTFILE` はローカルまたはリモートクライアントホストから起動できますが、出力ファイルはサーバーホスト上に作成されます。
- 物理バックアップ方法は一般に、サーバーをオフラインにできるように、MySQL サーバーホスト上でローカルに開始されますが、コピーされるファイルの宛先はリモートにすることができます。

スナップショットバックアップ

一部のファイルシステム実装では、「スナップショット」を取得できます。これらは、ファイルシステム全体の物理コピーを必要とせずに、特定の時点のファイルシステムの論理コピーを提供します。(たとえば、実装では、スナップショット取得時間後に変更されたファイルシステムの部分のみがコピーされるように、コピーオンライト (copy-on-write) 技法を使用することがあります。) MySQL 自体はファイルシステムスナップショットを取得するための機能を提供していません。これは Veritas、LVM、または ZFS などのサードパーティーソリューションから使用できます。

完全バックアップと増分バックアップ

完全バックアップには、特定の時点の MySQL サーバーによって管理されるすべてのデータが含まれます。増分バックアップは、特定の期間 (ある時点から別の時点まで) 中にデータに行われた変更から構成されます。MySQL では、このセクションで先述したものなど、完全バックアップを実行するためのさまざまな方法があります。増分バックアップは、サーバーのバイナリログを有効にすることによって可能になります。サーバーはそれをデータの変更を記録するために使用します。

完全リカバリとポイントインタイム (増分) リカバリ

完全リカバリでは、完全バックアップからすべてのデータをリストアします。これは、サーバーインスタンスをバックアップが行われたときのその状態にリストアします。その状態が十分に最新でない場合、完全リカバリのあとに、完全バックアップ以降に行われた増分バックアップのリカバリを行なって、サーバーをより新しい状態にすることができます。

増分リカバリは、特定の期間中に行われた変更のリカバリです。これは、サーバーの状態を特定の時点の最新にするため、ポイントインタイムリカバリとも呼ばれます。ポイントインタイムリカバリは、バイナリログに基づき、一般にバックアップが行われたときの状態にサーバーをリストアするバックアップファイルからの完全リカバリのあとに行われます。バイナリログファイルに書き込まれたデータの変更が増分リカバリとして適用され、データの変更が元に戻され、サーバーが目的の時点の状態になります。

テーブルの保守

テーブルが破損した場合、データの完全性が損なわれる可能性があります。InnoDB テーブルの場合、これはよくある問題ではありません。MyISAM テーブルをチェックし、問題が見つかった場合にそれらを修復するプログラムについては、[セクション7.6「MyISAM テーブルの保守とクラッシュリカバリ」](#)を参照してください。

バックアップのスケジューリング、圧縮、および暗号化

バックアップスケジューリングはバックアップ手順の自動化に役立ちます。バックアップ出力の圧縮によって、領域要件が縮小し、出力の暗号化により、バックアップされたデータの権限のないアクセスに対するセキュリティが強化されます。MySQL 自体はこれらの機能を提供していません。MySQL Enterprise Backup 製品によって InnoDB

バックアップを圧縮し、バックアップ出力の圧縮や暗号化は、ファイルシステムユーティリティーを使用して実現できます。その他のサードパーティソリューションも利用できます。

7.2 データベースバックアップ方法

このセクションでは、バックアップを作成する場合の一般的な方法をまとめています。

MySQL Enterprise Backup によるホットバックアップの作成

MySQL Enterprise Edition の顧客は、[MySQL Enterprise Backup](#) 製品を使用して、インスタンス全体または選択したデータベース、テーブル、あるいはその両方の [physical](#) バックアップを実行できます。この製品には、[増分および圧縮](#) バックアップの機能が含まれます。物理データベースファイルのバックアップは、リストアが `mysqldump` コマンドなどの論理技法よりはるかに高速になります。InnoDB テーブルは [ホットバックアップ](#) メカニズムを使用してコピーされます。(理想的には InnoDB テーブルでデータの大部分を表しているべきです。)ほかのストレージエンジンのテーブルは、[ウォームバックアップ](#) メカニズムを使用してコピーされます。MySQL Enterprise Backup 製品の概要については、[セクション30.2「MySQL Enterprise Backup の概要」](#)を参照してください。

mysqldump によるバックアップの作成

`mysqldump` プログラムでバックアップを作成できます。すべての種類のテーブルをバックアップできます。([セクション7.4「バックアップへのmysqldumpの使用」](#)を参照してください。)

InnoDB テーブルの場合、`mysqldump` に `--single-transaction` オプションを使用して、テーブルをロックしないオンラインバックアップを実行できます。 [セクション7.3.1「バックアップポリシーの確立」](#)を参照してください。

テーブルファイルのコピーによるバックアップの作成

MyISAM テーブルは、テーブルファイル (`*.MYD`、`*.MYI` ファイルおよび関連する `*.sdi` ファイル) をコピーすることでバックアップできます。一貫したバックアップを取得するには、サーバーを停止するか、関連するテーブルをロックしてフラッシュします。

```
FLUSH TABLES tbl_list WITH READ LOCK;
```

読み取りロックのみが必要です。これにより、データベースディレクトリ内のファイルのコピー中に、ほかのクライアントが引き続きテーブルをクエリーすることができます。バックアップを開始する前に、すべてのアクティブインデックスページがディスクに書き込まれるようにするため、フラッシュが必要です。 [セクション13.3.6「LOCK TABLES および UNLOCK TABLES ステートメント」](#) および [セクション13.7.8.3「FLUSH ステートメント」](#)を参照してください。

サーバーが何も更新していないかぎり、テーブルファイルをコピーするだけでバイナリバックアップを作成することもできます。(ただし、データベースに InnoDB テーブルが含まれている場合、テーブルファイルのコピー方法は機能しません。さらに、サーバーがアクティブにデータを更新していない場合、InnoDB は変更されたデータをまだメモリー内にキャッシュしており、ディスクにフラッシュしていないことがあります。)

このバックアップ方法の例は、 [セクション13.2.5「IMPORT TABLE ステートメント」](#) のエクスポートおよびインポートの例を参照してください。

区切りテキストファイルバックアップの作成

テーブルのデータを含むテキストファイルを作成するには、`SELECT * INTO OUTFILE 'file_name' FROM tbl_name` を使用できます。このファイルはクライアントホストではなく、MySQL サーバーホスト上に作成されます。このステートメントの場合、ファイルの上書きを許可すると、セキュリティリスクになるため、出力ファイルがすでに存在してはなりません。 [セクション13.2.10「SELECT ステートメント」](#)を参照してください。この方法はあらゆる種類のデータファイルに機能しますが、テーブルデータのみ保存し、テーブル構造は保存しません。

テキストデータファイル (バックアップされたテーブルの `CREATE TABLE` ステートメントを含むファイルに加えて) を作成する別の方法は、`mysqldump` と `--tab` オプションを使用することです。 [セクション7.4.3「mysqldump による区切りテキストフォーマットでのデータのダンプ」](#)を参照してください。

デリミタ付きテキストデータファイルをリロードするには、`LOAD DATA` または `mysqlimport` を使用します。

バイナリログを有効にすることによる増分バックアップの作成

MySQL は、バイナリログを使用した増分バックアップをサポートしています。バイナリログファイルは、バックアップを実行した時点のあとに行われたデータベースへの変更のレプリケートする必要がある情報を提供します。したがって、サーバーを point-in-time にリストアできるようにするには、MySQL 8.0 のデフォルト設定であるバイナリロギングを有効にする必要があります。セクション5.4.4「バイナリログ」を参照してください。

増分バックアップ (最後の完全バックアップまたは増分バックアップ以降に発生したすべての変更を含む) を作成しようとするときは、`FLUSH LOGS` を使用して、バイナリログをローテーションしてください。これが完了したら、最後の完全または増分バックアップの瞬間から最後の 1 つ前の範囲のすべてのバイナリログをバックアップの場所にコピーする必要があります。これらのバイナリログは増分バックアップで、リストア時に、セクション7.5「Point-in-Time (増分) リカバリ」に説明するように、それらを適用します。次回全体バックアップを実行するときは、`FLUSH LOGS` または `mysqldump --flush-logs` を使用してバイナリログもローテーションするようにしてください。セクション4.5.4「mysqldump — データベースバックアッププログラム」を参照してください。

レプリカを使用したバックアップの作成

バックアップの作成中にサーバーでパフォーマンスの問題が発生した場合、レプリケーションを設定し、ソースではなくレプリカでバックアップを実行するという戦略が役立ちます。セクション17.4.1「バックアップ用にレプリケーションを使用する」を参照してください。

レプリカをバックアップする場合は、選択したバックアップ方法に関係なく、レプリカデータベースのバックアップ時に接続メタデータリポジトリと適用者メタデータリポジトリ (セクション17.2.4「リレーログおよびレプリケーションメタデータリポジトリ」を参照) をバックアップする必要があります。この情報は、レプリカデータのリストア後にレプリケーションを再開するために常に必要です。レプリカが `LOAD DATA` ステートメントをレプリケートしている場合は、レプリカがこの目的で使用するディレクトリに存在する `SQL_LOAD-*` ファイルもバックアップする必要があります。レプリカでは、中断された `LOAD DATA` 操作のレプリケーションを再開するために、これらのファイルが必要です。このディレクトリの場所は、`slave_load_tmpdir` システム変数の値です。その変数を設定してサーバーを起動しなかった場合、ディレクトリの場所は `tmpdir` システム変数の値になります。

破損したテーブルのリカバリ

破損した `MyISAM` テーブルをリストアする必要がある場合、まず `REPAIR TABLE` または `myisamchk -r` を使用して、それらのリカバリを試みます。それは、すべてのケースの 99.9% で機能するはずですが、`myisamchk` が失敗した場合は、セクション7.6「MyISAM テーブルの保守とクラッシュリカバリ」を参照してください。

ファイルシステムスナップショットを使用したバックアップの作成

Veritas ファイルシステムを使用している場合、次のようにバックアップを作成できます。

1. クライアントプログラムから、`FLUSH TABLES WITH READ LOCK` を実行します。
2. 別のシェルから、`mount vxfs snapshot` を実行します。
3. 最初のクライアントから、`UNLOCK TABLES` を実行します。
4. スナップショットからファイルをコピーします。
5. スナップショットをアンマウントします。

同様のスナップショット機能は、LVM や ZFS などのほかのファイルシステムでも利用できます。

7.3 バックアップおよびリカバリ戦略の例

このセクションでは、いくつかの種類のクラッシュ後にデータをリカバリできるようにするバックアップを実行するための手順について説明します。

- オペレーティングシステムのクラッシュ
- 停電

- ファイルシステムのクラッシュ
- ハードウェアの問題 (ハードドライブ、マザーボードなど)

コマンド例には、`mysqldump` および `mysql` クライアントプログラム用の `--user` や `--password` などのオプションは含まれていません。クライアントプログラムが MySQL サーバーに接続できるようにする必要に応じて、それらのオプションを含めてください。

データは、トランザクションと自動クラッシュリカバリをサポートする `InnoDB` ストレージエンジンに格納されています。さらに、MySQL サーバーはクラッシュ時に負荷がかかっているとします。そうでなければ、リカバリは必要ないことがあります。

オペレーティングシステムのクラッシュや停電の場合、再起動後、MySQL のディスクデータを使用できるものと考えられます。 `InnoDB` データファイルにはクラッシュのために一貫したデータが格納されていない可能性があります。 `InnoDB` はそのログを読み取り、データファイルにフラッシュされていないコミット保留中のトランザクションやコミットされていないトランザクションのリストを見つけます。 `InnoDB` はまだコミットされていないトランザクションを自動的にロールバックし、コミットされたものはデータファイルにフラッシュします。このリカバリプロセスに関する情報は、MySQL エラーログによってユーザーに伝えられます。次はログの抜粋の例です。

```
InnoDB: Database was not shut down normally.
InnoDB: Starting recovery from log files...
InnoDB: Starting log scan based on checkpoint at
InnoDB: log sequence number 0 13674004
InnoDB: Doing recovery: scanned up to log sequence number 0 13739520
InnoDB: Doing recovery: scanned up to log sequence number 0 13805056
InnoDB: Doing recovery: scanned up to log sequence number 0 13870592
InnoDB: Doing recovery: scanned up to log sequence number 0 13936128
...
InnoDB: Doing recovery: scanned up to log sequence number 0 20555264
InnoDB: Doing recovery: scanned up to log sequence number 0 20620800
InnoDB: Doing recovery: scanned up to log sequence number 0 20664692
InnoDB: 1 uncommitted transaction(s) which must be rolled back
InnoDB: Starting rollback of uncommitted transactions
InnoDB: Rolling back trx no 16745
InnoDB: Rolling back of trx no 16745 completed
InnoDB: Rollback of uncommitted transactions completed
InnoDB: Starting an apply batch of log records to the database...
InnoDB: Apply batch completed
InnoDB: Started
mysqld: ready for connections
```

ファイルシステムのクラッシュやハードウェアの問題の場合、再起動後、MySQL デスクデータを使用できないものと考えられます。これは、ディスクデータの一部のブロックが読み取り不可能になったため、MySQL が正常な起動に失敗することを意味します。この場合、ディスクを再フォーマットするか、新しいディスクをインストールするか、または根本的な問題を修正する必要があります。さらに、バックアップから MySQL データをリカバリする必要があります。これはバックアップがすでに行われていることを意味します。それが確実に当てはまるようにするには、バックアップポリシーを設計し、実装します。

7.3.1 バックアップポリシーの確立

役に立つように、バックアップは定期的にスケジュールする必要があります。完全バックアップ (特定の時点でのデータのスナップショット) は、MySQL でいくつかのツールを使用して実行できます。たとえば、`MySQL Enterprise Backup` は、`InnoDB` データファイルのバックアップ時にオーバーヘッドを最小にし、中断を防ぐ最適化を伴うインスタンス全体の物理バックアップを実行できます。`mysqldump` はオンライン論理バックアップを提供します。この説明では `mysqldump` を使用します。

負荷が少ない日曜日の午後 1 時に、次のコマンドを使用して、すべてのデータベースのすべての `InnoDB` テーブルの完全バックアップを作成するとします。

```
shell> mysqldump --all-databases --master-data --single-transaction > backup_sunday_1_PM.sql
```

`mysqldump` によって生成される結果の `.sql` ファイルには、あとでダンプしたテーブルのリロードに使用できる SQL `INSERT` ステートメントのセットが含まれます。

このバックアップ操作では、ダンプの最初ですべてのテーブルに対するグローバル読み取りロックを取得します (`FLUSH TABLES WITH READ LOCK` を使用して)。このロックが取得されるとすぐに、バイナリログの座標が読み

取られ、ロックが解除されます。FLUSH ステートメントが発行されたときに長い更新ステートメントが実行中の場合、バックアップ操作はそれらのステートメントが終了するまで停止する可能性があります。その後、ダンプはロックフリーとなり、テーブルの読み取りと書き込みを妨げません。

先に、バックアップするテーブルは InnoDB テーブルであるとしたため、`--single-transaction` は、一貫性読み取りを使用し、`mysqldump` によって表示されたデータが変更されないことを保証します。(ほかのクライアントによる InnoDB テーブルへの変更は、`mysqldump` プロセスによって表示されません)。バックアップ操作に非トランザクションテーブルが含まれる場合、一貫性には、バックアップ中にそれらに変更されない必要があります。たとえば、`mysql` データベース内の `MyISAM` テーブルの場合、バックアップ中に、MySQL アカウントへの管理上の変更があってはなりません。

完全バックアップが必要ですが、それらを作成するために常に都合がよいとは限りません。それらは大きなバックアップファイルを生成し、生成に時間がかかります。それらは、連続した各完全バックアップに、前回の完全バックアップから変更されていない部分でもすべてのデータが含まれるという点で、最適ではありません。初期完全バックアップを作成し、次に増分バックアップを作成するほうが効率的です。増分バックアップは小さく、生成にかかる時間が少なくなります。このトレードオフは、リカバリ時に、完全バックアップをリロードするだけではデータをリストアできないことです。増分バックアップを処理して、増分の変更もリカバリする必要があります。

増分バックアップを作成するには、増分の変更を保存する必要があります。MySQL では、これらの変更はバイナリログで表されるため、MySQL サーバーを常に `--log-bin` オプションで起動して、そのログを有効にしてください。バイナリロギングが有効にされていると、サーバーはデータの更新中に、各データの変更をファイルに書き込みます。数日間実行されている MySQL サーバーのデータディレクトリを調べると、次の MySQL バイナリログファイルが見つかります:

```
-rw-rw---- 1 guilhem guilhem 1277324 Nov 10 23:59 gbichot2-bin.000001
-rw-rw---- 1 guilhem guilhem      4 Nov 10 23:59 gbichot2-bin.000002
-rw-rw---- 1 guilhem guilhem    79 Nov 11 11:06 gbichot2-bin.000003
-rw-rw---- 1 guilhem guilhem   508 Nov 11 11:08 gbichot2-bin.000004
-rw-rw---- 1 guilhem guilhem 220047446 Nov 12 16:47 gbichot2-bin.000005
-rw-rw---- 1 guilhem guilhem  998412 Nov 14 10:08 gbichot2-bin.000006
-rw-rw---- 1 guilhem guilhem   361 Nov 14 10:07 gbichot2-bin.index
```

MySQL サーバーは再起動するたびに、シーケンスの次の番号を使用して、新しいバイナリログファイルを作成します。サーバーが実行している間、`FLUSH LOGS` SQL ステートメントを発行するか、`mysqladmin flush-logs` コマンドによって、手動で、それに現在のバイナリログファイルをクローズし、新しいファイルを開始するように伝えることもできます。`mysqldump` にはログをフラッシュするオプションもあります。データディレクトリ内の `.index` ファイルには、ディレクトリ内のすべての MySQL バイナリログのリストが含まれます。

MySQL バイナリログは、増分バックアップのセットを形成するため、リカバリに重要です。完全バックアップの作成時にログをフラッシュさせる場合、その後作成されるバイナリログファイルには、バックアップ以降に行われたすべてのデータの変更が含まれます。ここで、前述の `mysqldump` コマンドを少し修正して、完全バックアップの時点で MySQL バイナリログをフラッシュするようにし、ダンプファイルに新しい現在のバイナリログの名前が含まれるようにします。

```
shell> mysqldump --single-transaction --flush-logs --master-data=2 \
--all-databases > backup_sunday_1_PM.sql
```

このコマンドの実行後、`--flush-logs` オプションによって、サーバーにそのログをフラッシュさせるため、データディレクトリには新しいバイナリログファイル `gbichot2-bin.000007` が格納されます。`--master-data` オプションは `mysqldump` でその出力にバイナリログ情報を書き込ませるため、結果の `.sql` ダンプファイルにはこれらの行が含まれます。

```
-- Position to start replication or point-in-time recovery from
-- CHANGE MASTER TO MASTER_LOG_FILE='gbichot2-bin.000007',MASTER_LOG_POS=4;
```

`mysqldump` コマンドで完全バックアップを作成しているため、これらの行は 2 つのことを意味します。

- ダンプファイルには、`gbichot2-bin.000007` バイナリログファイル以上に書き込まれた変更の前に行われたすべての変更が含まれます。
- バックアップ後に記録されたすべてのデータ変更はダンプファイルには存在しませんが、`gbichot2-bin.000007` バイナリログファイル以上に存在します。

月曜日の午後 1 時に、ログをフラッシュし、新しいバイナリログファイルを開始することによって、増分バックアップを作成できます。たとえば、`mysqladmin flush-logs` コマンドを実行すると、`gbichot2-bin.000008` が作成されま

す。日曜日の午後 1 時から月曜日の午後 1 時までのすべての変更は、`gbichot2-bin.000007` に書き込まれます。この増分バックアップは重要であるため、それを安全な場所にコピーすることをお勧めします。(たとえば、それをテープや DVD にバックアップするか、別のマシンにコピーします。) 火曜日の午後 1 時に、さらに `mysqldadmin flush-logs` コマンドを実行します。月曜日の午後 1 時から火曜日の午後 1 時までのすべての変更は、`gbichot2-bin.000008` で書き込まれます(これも安全な場所にコピーする必要があります)。

MySQL バイナリログはディスク領域を占有します。領域を解放するため、ときどきそれらをパーージします。これを実行する 1 つの方法は、完全バックアップを作成したときなど、必要なくなったバイナリログを削除することです。

```
shell> mysqldump --single-transaction --flush-logs --master-data=2 \
--all-databases --delete-master-logs > backup_sunday_1_PM.sql
```

注記

サーバーがレプリケーションソースサーバーである場合、レプリカがまだバイナリログの内容を完全に処理していない可能性があるため、`mysqldump --delete-master-logs` で MySQL バイナリログを削除すると危険になる可能性があります。PURGE BINARY LOGS ステートメントの説明では、MySQL バイナリログを削除する前に確認すべきことを説明しています。セクション 13.4.1.1 「PURGE BINARY LOGS ステートメント」を参照してください。

7.3.2 リカバリへのバックアップの使用

今度は、水曜日の午前 8 時に、バックアップからのリカバリが必要な致命的な予期しない終了があるとして。リカバリするには、まず存在する最後の完全バックアップ(日曜日の午後 1 時のもの)をリストアします。完全バックアップファイルは一連の SQL ステートメントにすぎないため、そのリストアはきわめて簡単です。

```
shell> mysql < backup_sunday_1_PM.sql
```

この時点で、データは日曜日の午後 1 時現在の状態にリストアされます。それ以降に行われた変更をリストアするには、増分バックアップを使用する必要があります。つまり、`gbichot2-bin.000007` と `gbichot2-bin.000008` バイナリログファイルです。必要に応じて、バックアップされた場所からファイルをフェッチして、次のようにそれらの内容を処理します。

```
shell> mysqlbinlog gbichot2-bin.000007 gbichot2-bin.000008 | mysql
```

これで、データを火曜日の午後 1 時現在の状態にリカバリしましたが、まだその日からクラッシュの日までの変更が不足しています。それらを失わないために、MySQL サーバーにその MySQL バイナリログを、そのデータファイルを格納している場所と異なる安全な場所(RAID ディスク、SAN など)に保存させ、これらのログが破損したディスク上にないようにする必要があります。(つまり、データディレクトリが存在する場所と別の物理デバイス上の場所を指定する `--log-bin` オプションでサーバーを起動できます。このようにすると、ディレクトリを格納するデバイスが失われてもログは安全です。) これを実行していた場合、`gbichot2-bin.000009` ファイル(および任意の後続のファイル)が手元にあるため、`mysqlbinlog` と `mysql` を使用して、それらを適用し、クラッシュの瞬間まで損失なく、最新のデータの変更をリストアできます。

```
shell> mysqlbinlog gbichot2-bin.000009 ... | mysql
```

`mysqlbinlog` を使用して、バイナリログファイルを処理する詳細については、セクション 7.5 「Point-in-Time (増分) リカバリ」を参照してください。

7.3.3 バックアップ戦略サマリー

オペレーティングシステムのクラッシュまたは停電の場合、InnoDB 自体がデータのリカバリのすべてのジョブを実行します。ただし、安心のため、次のガイドラインを参照してください。

- バイナリロギングを有効にして、常に MySQL サーバーをチューニングします(MySQL 8.0 のデフォルト設定)。そのような安全なメディアがある場合、この技法は、ディスクの負荷分散にも役立ちます(その結果パフォーマンスも向上します)。
- セクション 7.3.1 「バックアップポリシーの確立」で先に示した、オンラインのブロックしないバックアップを作成する `mysqldump` コマンドを使用して、定期的な完全バックアップを作成します。
- FLUSH LOGS または `mysqldadmin flush-logs` を使用して、ログをフラッシュして、定期的な増分バックアップを作成します。

7.4 バックアップへの mysqldump の使用

ヒント

複数のスレッド、ファイル圧縮、進捗情報の表示、および Oracle Cloud Infrastructure Object Storage ストリーミングや MySQL データベースサービス 互換性チェックおよび変更などのクラウド機能で並列ダンプを提供する [MySQL Shell dump utilities](#) の使用を検討してください。ダンプは、[MySQL Shell load dump utilities](#) を使用して MySQL Server インスタンスまたは MySQL データベースサービス DB システムに簡単にインポートできます。MySQL Shell のインストール手順は、[here](#) にあります。

このセクションでは、[mysqldump](#) を使用して、ダンプファイルを生成する方法およびダンプファイルをリロードする方法について説明します。ダンプファイルはいくつかの方法で使用できます。

- データ損失の場合にデータリカバリを可能にするためのバックアップとして。
- レプリカを設定するためのデータのソースとして。
- 実験用のデータのソースとして。
 - 元のデータを変更せずに使用できるデータベースのコピーを作成する場合。
 - アップグレードの非互換性の可能性をテストする場合。

[mysqldump](#) は `--tab` オプションを指定するかどうかに応じて、2 種類の出力を生成します。

- `--tab` がないと、[mysqldump](#) は SQL ステートメントを標準出力に書き込みます。この出力は、ダンプされるオブジェクト (データベース、テーブル、ストアドルーチンなど) を作成する `CREATE` ステートメントとデータをテーブルにロードする `INSERT` ステートメントから構成されます。出力はファイルに保存して、あとで [mysql](#) を使用してリロードし、ダンプされたオブジェクトを再作成できます。SQL ステートメントのフォーマットを変更し、ダンプされるオブジェクトを制御するためにオプションを使用できます。
- `--tab` を付けると、[mysqldump](#) はダンプされるテーブルごとに 2 つの出力ファイルを生成します。サーバーは、テーブル行ごとに 1 行ずつ、タブ区切りテキストとして 1 つのファイルを書き込みます。このファイルは出力ディレクトリ内で `tbl_name.txt` という名前が付けられます。サーバーはテーブルの `CREATE TABLE` ステートメントも [mysqldump](#) に送信し、それは `tbl_name.sql` という名前のファイルとしてそれを出力ディレクトリに書き込みます。

7.4.1 mysqldump による SQL フォーマットでのデータのダンプ

このセクションでは、[mysqldump](#) を使用して、SQL フォーマットのダンプファイルを作成する方法について説明します。そのようなダンプファイルのリロードについては、[セクション7.4.2「SQL フォーマットバックアップのリロード」](#)を参照してください。

デフォルトで、[mysqldump](#) は情報を SQL ステートメントとして標準出力に書き込みます。出力をファイルに保存できます。

```
shell> mysqldump [arguments] > file_name
```

すべてのデータベースをダンプするには、`--all-databases` オプションを付けて [mysqldump](#) を呼び出します。

```
shell> mysqldump --all-databases > dump.sql
```

特定のデータベースのみをダンプするには、コマンド行でそれらを指定し、`--databases` オプションを使用します。

```
shell> mysqldump --databases db1 db2 db3 > dump.sql
```

`--databases` オプションによって、コマンド行上のすべての名前がデータベース名として扱われます。このオプションを使用しないと、[mysqldump](#) は最初の名前をデータベース名として、そのあとに続く名前をテーブル名として扱います。

`--all-databases` または `--databases` を使用すると、[mysqldump](#) は、各データベースのダンプ出力の前に、`CREATE DATABASE` および `USE` ステートメントを書き込みます。これにより、ダンプファイルがリロードされると、それが

各データベースが存在しなければ作成して、デフォルトのデータベースにするため、データベースの内容がそれらの作成元と同じデータベースにロードされます。ダンプファイルに、各データベースを再作成する前にその削除を強制する場合、`--add-drop-database` オプションも使用します。この場合、`mysqldump` は各 `CREATE DATABASE` ステートメントの前に、`DROP DATABASE` ステートメントを書き込みます。

単一のデータベースをダンプするには、コマンド行でそれを指定します。

```
shell> mysqldump --databases test > dump.sql
```

単一のデータベースの場合、`--databases` オプションを省略できます。

```
shell> mysqldump test > dump.sql
```

2つの先述のコマンドの違いは、`--databases` を付けないと、ダンプの出力に `CREATE DATABASE` または `USE` ステートメントが含まれません。これにはいくつかの問題があります。

- ダンプファイルをリロードする場合、サーバーがリロードするデータベースを認識するように、デフォルトのデータベース名を指定する必要があります。
- リロードする場合、元の名前と異なるデータベース名を指定でき、これにより、データを別のデータベースにリロードできます。
- リロードするデータベースが存在しない場合、まずそれを作成する必要があります。
- 出力には `CREATE DATABASE` ステートメントが含まれていないため、`--add-drop-database` オプションは効果がありません。それを使用しても `DROP DATABASE` ステートメントは生成されません。

データベースから特定のテーブルのみをダンプするには、コマンド行でデータベース名に続いてそれらを指定します。

```
shell> mysqldump test t1 t3 t7 > dump.sql
```

デフォルトでは、ダンプファイル (`gtid_mode=ON`) を作成するサーバーで GTID が使用されている場合、`mysqldump` は、ソースサーバーの `gtid_executed` セットからターゲットサーバーの `gtid_purged` セットに GTID を追加する `SET @@GLOBAL.gtid_purged` ステートメントを出力に含めます。特定のデータベースまたはテーブルのみをダンプする場合、`mysqldump` に含まれる値には、ソースサーバー上の `gtid_executed` セット内のすべてのトランザクションの GTID (データベースの抑制された部分を変更したトランザクションや、部分ダンプに含まれていなかったサーバー上のその他のデータベースも含む) が含まれることに注意してください。ターゲットサーバーで部分ダンプファイルを 1 つしかリプレイしない場合、余分な GTID はそのサーバーの今後の操作で問題を引き起こしません。ただし、同じ GTID を含むターゲットサーバー上の別のダンプファイル (たとえば、同じソースサーバーからの別の部分ダンプ) をリプレイすると、2 番目のダンプファイル内の `SET @@GLOBAL.gtid_purged` ステートメントは失敗します。この問題を回避するには、`mysqldump` オプション `--set-gtid-purged` を `OFF` または `COMMENTED` に設定して、アクティブな `SET @@GLOBAL.gtid_purged` ステートメントなしで 2 番目のダンプファイルを出力するか、ダンプファイルをリプレイする前にステートメントを手動で削除します。

7.4.2 SQL フォーマットバックアップのリロード

SQL ステートメントから構成される `mysqldump` によって書き込まれたダンプファイルをリロードするには、それを `mysql` クライアントへの入力として使用します。`--all-databases` または `--databases` オプションを使用して、`mysqldump` によってダンプファイルが作成された場合、それには `CREATE DATABASE` および `USE` ステートメントが含まれ、データをロードするデフォルトのデータベースを指定する必要がありません。

```
shell> mysql < dump.sql
```

または、`mysql` 内から、`source` コマンドを使用します。

```
mysql> source dump.sql
```

ファイルが `CREATE DATABASE` および `USE` ステートメントを含まない単一データベースダンプである場合、まずデータベースを作成します (必要に応じて)。

```
shell> mysqladmin create db1
```

次に、ダンプファイルをロードする場合、データベース名を指定します。

```
shell> mysql db1 < dump.sql
```

または `mysql` 内から、データベースを作成し、それをデフォルトのデータベースとして選択し、ダンプファイルをロードします。

```
mysql> CREATE DATABASE IF NOT EXISTS db1;
mysql> USE db1;
mysql> source dump.sql
```

注記

Windows PowerShell ユーザーの場合: 「<」文字は PowerShell で将来使用するために予約されているため、`cmd.exe /c "mysql < dump.sql"`で引用符を使用するなど、別の方法が必要です。

7.4.3 mysqldump による区切りテキストフォーマットでのデータのダンプ

このセクションでは、`mysqldump` を使用して、区切りテキストのダンプファイルを作成する方法について説明します。そのようなダンプファイルのリロードについては、[セクション7.4.4「区切りテキストフォーマットバックアップのリロード」](#)を参照してください。

`--tab=dir_name` オプションを付けて、`mysqldump` を呼び出した場合、それは `dir_name` を出力ディレクトリとして使用し、テーブルごとに2つのファイルを使用して、そのディレクトリに個別にテーブルをダンプします。テーブル名は、これらのファイルのベース名です。 `t1` という名前のテーブルの場合、ファイルには `t1.sql` および `t1.txt` という名前が付けられます。 `.sql` ファイルにはテーブルの `CREATE TABLE` ステートメントが含まれます。 `.txt` ファイルにはテーブル行ごとに1行のテーブルデータが含まれます。

次のコマンドは `db1` データベースの内容を `/tmp` データベース内のファイルにダンプします。

```
shell> mysqldump --tab=/tmp db1
```

テーブルデータを格納する `.txt` ファイルはサーバーによって書き込まれるため、それらはサーバーの実行に使用されるシステムアカウントによって所有されます。サーバーは `SELECT ... INTO OUTFILE` を使用して、ファイルを書き込むため、この操作を実行するために `FILE` 権限が必要であり、指定した `.txt` ファイルがすでに存在する場合はエラーが発生します。

サーバーはダンプされるテーブルの `CREATE` 定義を `mysqldump` に送信し、それはそれらを `.sql` ファイルに書き込みます。そのためこれらのファイルは、`mysqldump` を実行するユーザーによって所有されます。

`--tab` はローカルサーバーのダンプにのみ使用することをお勧めします。リモートサーバーで使用する場合、`--tab` ディレクトリはローカルホストとリモートホストの両方に存在する必要があるため、`.txt` ファイルはサーバーによってリモートディレクトリ(サーバーホスト上)に書き込まれますが、`.sql` ファイルは `mysqldump` によってローカルディレクトリ(クライアントホスト上)に書き込まれます。

`mysqldump --tab` の場合、サーバーはデフォルトでテーブルデータを、カラム値間にタブを、カラム値を引用符で囲まず、行ターミネータとして改行を使用して、1行あたり1行で `.txt` ファイルに書き込みます。(これは、`SELECT ... INTO OUTFILE` の場合と同じデフォルトです。)

別のフォーマットを使用して、データファイルを書き込めるようにするため、`mysqldump` はこれらのオプションをサポートしています。

- `--fields-terminated-by=str`
カラム値を区切るための文字列(デフォルト: タブ)。
- `--fields-enclosed-by=char`
カラム値を囲む文字(デフォルト: 文字なし)。
- `--fields-optionally-enclosed-by=char`
数値以外のカラム値を囲む文字(デフォルト: 文字なし)。
- `--fields-escaped-by=char`

特殊文字をエスケープするための文字 (デフォルト: エスケープなし)。

- `--lines-terminated-by=str`

行終了文字列 (デフォルト: 改行)。

これらのオプションに指定する値に応じて、コマンド行で、コマンドインタプリタに合わせて値を引用符で囲むかエスケープする必要がある場合があります。または、16 進表記を使用して、値を指定します。 `mysqldump` にカラム値を二重引用符で囲ませたいとします。そうするには、`--fields-enclosed-by` オプションの値として、二重引用符を指定します。ただし、この文字は多くの場合コマンドインタプリタに特有であるため、特別に扱う必要があります。たとえば、Unix ではこのように二重引用符を表すことができます。

```
--fields-enclosed-by=""
```

どのプラットフォームでも 16 進で値を指定できます。

```
--fields-enclosed-by=0x22
```

複数のデータフォーマットオプションを一緒に使用することもよくあります。たとえば、行を改行文字/復帰改行ペア (`\r\n`) で終了させたカンマ区切り値フォーマットでテーブルをダンプするには、このコマンドを使用します (1 行で入力します)。

```
shell> mysqldump --tab=/tmp --fields-terminated-by=,  
--fields-enclosed-by="" --lines-terminated-by=0x0d0a db1
```

データ書式設定オプションのいずれかを使用してテーブルデータをダンプする場合は、後でデータファイルをリロードするときに同じ書式を指定して、ファイルの内容が正しく解釈されるようにする必要があります。

7.4.4 区切りテキストフォーマットバックアップのリロード

`mysqldump --tab` によって生成されるバックアップの場合、各テーブルは出力ディレクトリに、テーブルの `CREATE TABLE` ステートメントを含む `.sql` ファイルと、テーブルデータを含む `.txt` ファイルで表されます。テーブルをリロードするには、まず場所を出力ディレクトリに変更します。次に、`mysql` で `.sql` ファイルを処理し、空のテーブルを作成し、`.txt` ファイルを処理して、データをテーブルにロードします。

```
shell> mysql db1 < t1.sql  
shell> mysqlimport db1 t1.txt
```

`mysqlimport` を使用してデータファイルをロードするかわりに、`mysql` クライアント内から `LOAD DATA` ステートメントを使用する方法もあります:

```
mysql> USE db1;  
mysql> LOAD DATA INFILE 't1.txt' INTO TABLE t1;
```

テーブルを最初にダンプしたときに `mysqldump` でデータ書式設定オプションを使用した場合は、`mysqlimport` または `LOAD DATA` で同じオプションを使用して、データファイルの内容を正しく解釈する必要があります:

```
shell> mysqlimport --fields-terminated-by=,  
--fields-enclosed-by="" --lines-terminated-by=0x0d0a db1 t1.txt
```

または:

```
mysql> USE db1;  
mysql> LOAD DATA INFILE 't1.txt' INTO TABLE t1  
FIELDS TERMINATED BY ',' FIELDS ENCLOSED BY ''  
LINES TERMINATED BY '\r\n';
```

7.4.5 mysqldump のヒント

このセクションでは、`mysqldump` を使用して特定の問題を解決できる技法を調査します。

- データベースのコピーの作成方法
- サーバー間でデータベースをコピーする方法

- ストアドプログラム (ストアドプロシージャーおよび関数、トリガー、およびイベント) をダンプする方法
- 定義とデータを個別にダンプする方法

7.4.5.1 データベースのコピーの作成

```
shell> mysqldump db1 > dump.sql
shell> mysqladmin create db2
shell> mysql db2 < dump.sql
```

`mysqldump` コマンド行に `--databases` を使用すると、ダンプファイルに `USE db1` が含まれ、それによって `mysql` コマンド行の `db2` の指定の効果がオーバーライドされるため、使用しないでください。

7.4.5.2 サーバー間でのデータベースのコピー

サーバー 1 で:

```
shell> mysqldump --databases db1 > dump.sql
```

サーバー 1 からサーバー 2 にダンプファイルをコピーします。

サーバー 2 で:

```
shell> mysql < dump.sql
```

`mysqldump` コマンド行で `--databases` を使用すると、それが存在する場合にデータベースを作成し、それをリロードされるデータのデフォルトのデータベースにする `CREATE DATABASE` および `USE` ステートメントがダンプファイルに含まれます。

または、`mysqldump` コマンドから `--databases` を省略できます。次に、(必要に応じて) サーバー 2 にデータベースを作成し、ダンプファイルをリロードするときにデフォルトデータベースとして指定する必要があります。

サーバー 1 で:

```
shell> mysqldump db1 > dump.sql
```

サーバー 2 で:

```
shell> mysqladmin create db1
shell> mysql db1 < dump.sql
```

この場合、別のデータベース名を指定できるため、`mysqldump` コマンドから `--databases` を省略すると、あるデータベースからデータをダンプし、別のデータベースにそれをロードすることができます。

7.4.5.3 ストアドプログラムのダンプ

いくつかのオプションは、`mysqldump` がストアドプログラム (ストアドプロシージャーおよび関数、トリガー、およびイベント) を処理する方法を制御します。

- `--events`: イベントスケジューラのイベントのダンプ
- `--routines`: ストアドプロシージャーおよびストアドファンクションのダンプ
- `--triggers`: テーブルのトリガーのダンプ

テーブルがダンプされるときに、それらにそれらが持ついずれかのトリガーが伴うように、`--triggers` オプションはデフォルトで有効にされています。ほかのオプションはデフォルトで無効にされ、対応するオブジェクトをダンプするために明示的に指定する必要があります。これらのオプションのいずれかを明示的に無効にするには、そのスキップフォーム `--skip-events`、`--skip-routines`、または `--skip-triggers` を使用します。

7.4.5.4 テーブル定義と内容の個別のダンプ

`--no-data` オプションは `mysqldump` にテーブルデータをダンプしないように伝えるため、ダンプファイルにはテーブルを作成するステートメントのみが含まれます。逆に、`--no-create-info` オプションは、ダンプファイルにテーブルデータのみが含まれるように、`mysqldump` に出力から `CREATE` ステートメントを抑制するように伝えます。

たとえば、`test` データベースのテーブル定義とデータを別々にダンプするには、これらのコマンドを使用します。

```
shell> mysqldump --no-data test > dump-defs.sql
shell> mysqldump --no-create-info test > dump-data.sql
```

定義のみのダンプの場合、ストアルーチンとイベントの定義も含めるには、`--routines` および `--events` オプションを追加します。

```
shell> mysqldump --no-data --routines --events test > dump-defs.sql
```

7.4.5.5 mysqldump を使用したアップグレードの非互換性のテスト

MySQL のアップグレードを検討する場合、新しいバージョンを現在の本番バージョンとべつにインストールすることが賢明です。これによって、本番サーバーからデータベースとデータベースオブジェクト定義をダンプし、新しいサーバーにロードして、それらが正しく処理されることを確認できます。(これはダウングレードのテストの場合にも役立ちます。)

本番サーバーで:

```
shell> mysqldump --all-databases --no-data --routines --events > dump-defs.sql
```

アップグレードされたサーバーで:

```
shell> mysql < dump-defs.sql
```

ダンプファイルにはテーブルデータが含まれないため、すばやく処理できます。これにより、長いデータロード操作を待つことなく、可能性のある非互換性を見分けることができます。ダンプファイルの処理中の警告やエラーを探します。

定義が正しく処理されていることを確認したら、データをダンプし、それをアップグレードしたサーバーにロードしてみます。

本番サーバーで:

```
shell> mysqldump --all-databases --no-create-info > dump-data.sql
```

アップグレードされたサーバーで:

```
shell> mysql < dump-data.sql
```

ここでテーブルの内容を確認し、いくつかのテストクエリーを実行します。

7.5 Point-in-Time (増分) リカバリ

point-in-time リカバリとは、特定の時点までのデータ変更のリカバリのことです。一般に、この種類のリカバリは、サーバーをバックアップが行われた時点の状態にする完全バックアップのリストア後に実行されます。(完全バックアップは、[セクション7.2「データベースバックアップ方法」](#)に示すものなど、いくつかの方法で行うことができます。)さらに、ポイントインタイムリカバリは、完全バックアップの時点からより最近の時点まで、増分的にサーバーを最新にします。

7.5.1 バイナリログを使用したポイントインタイムリカバリ

このセクションでは、バイナリログを使用してポイントインタイムリカバリを実行する一般的な概念について説明します。次のセクション [セクション7.5.2「イベントの位置を使用したポイントインタイムリカバリ」](#) では、操作の詳細を例とともに説明します。

注記

このセクションと次のセクションの例の多くは、`mysql` クライアントを使用して、`mysqlbinlog` によって生成されたバイナリログ出力を処理します。バイナリログに `\0` (null) 文字が含まれている場合は、`--binary-mode` オプションを指定して呼び出さなければ、その出力を `mysql` で解析できません。

point-in-time リカバリの情報のソースは、全体バックアップ操作の後に生成されるバイナリログファイルのセットです。したがって、サーバーを point-in-time にリストアできるようにするには、MySQL 8.0 のデフォルト設定であるバイナリロギングを有効にする必要があります (セクション5.4.4 「バイナリログ」 を参照してください)。

バイナリログからデータをリストアするには、現在のバイナリログファイルの名前と場所を知っている必要があります。デフォルトで、サーバーはデータディレクトリにバイナリログファイルを作成しますが、`--log-bin` オプションでパス名を指定して、別の場所にファイルを配置できます。すべてのバイナリログファイルのリストを表示するには、次のステートメントを使用します。

```
mysql> SHOW BINARY LOGS;
```

現在のバイナリログファイルの名前を判断するには、次のステートメントを発行します。

```
mysql> SHOW MASTER STATUS;
```

`mysqlbinlog` ユーティリティーは、バイナリログファイル内のイベントをバイナリ形式からテキストに変換して、それらを表示または適用できるようにします。`mysqlbinlog` には、イベント時間またはログ内のイベントの位置に基づいてバイナリログのセクションを選択するためのオプションがあります。セクション4.6.8 「`mysqlbinlog` — バイナリログファイルを処理するためのユーティリティー」を参照してください。

バイナリログからイベントを適用すると、それらが表すデータ変更が再実行されます。これにより、特定の期間のデータの変更のリカバリが可能です。バイナリログからイベントを適用するには、`mysql` クライアントを使用して `mysqlbinlog` 出力を処理します:

```
shell> mysqlbinlog binlog_files | mysql -u root -p
```

バイナリログファイルが暗号化されている場合 (MySQL 8.0.14 以降で実行可能)、`mysqlbinlog` は前述の例のようにそれらを直接読み取ることはできませんが、`--read-from-remote-server (-R)` オプションを使用してサーバーから読み取ることができます。例:

```
shell> mysqlbinlog --read-from-remote-server --host=host_name --port=3306 --user=root --password --ssl-mode=required binlog_files | mysql -u root -p
```

ここでは、バイナリログファイルのデータが暗号化されていない形式で `mysqlbinlog` に送信されるため、バイナリログファイルのデータが転送中に保護されるように `--ssl-mode=required` オプションが使用されています。

ログの内容を表示すると、イベントを実行する前に、イベントの時間や位置を特定して、ログの内容の一部を選択する必要があります。ログからイベントを表示するには、`mysqlbinlog` 出力をページングプログラムに送信します。

```
shell> mysqlbinlog binlog_files | more
```

または、出力をファイルに保存し、テキストエディタでファイルを表示します。

```
shell> mysqlbinlog binlog_files > tmpfile
shell> ... edit tmpfile ...
```

出力をファイルに保存すると、誤った `DROP TABLE` などの特定のイベントを削除してログの内容を実行するための予備として役立ちます。ファイルの内容を実行する前に、実行されないステートメントをファイルから削除できます。ファイルを編集した後、次のように内容を適用します:

```
shell> mysql -u root -p < tmpfile
```

MySQL サーバーに適用するバイナリログが複数ある場合、安全な方法は、サーバーへの単一の接続を使用してそれらをすべて処理することです。これは、安全でない可能性があることを示す例です。

```
shell> mysqlbinlog binlog.000001 | mysql -u root -p # DANGER!!
shell> mysqlbinlog binlog.000002 | mysql -u root -p # DANGER!!
```

最初のログファイルに `CREATE TEMPORARY TABLE` ステートメントが含まれており、2 番目のログに一時テーブルを使用するステートメントが含まれている場合、サーバーへの異なる接続を使用して、このようにバイナリログを処理すると問題が発生します。最初の `mysql` プロセスが終了すると、サーバーは一時テーブルを削除します。2 番目の `mysql` プロセスでテーブルの使用を試みると、サーバーは「不明なテーブル」と報告します。

このような問題を回避するには、single 接続を使用して、処理するすべてのバイナリログファイルの内容を適用します。これはそれを実行する 1 つの方法です。


```
shell> mysqlbinlog binlog.000001 binlog.000002 | mysql -u root -p
```

別の方法として、ログ全体を単一のファイルに書き込み、そのファイルを処理する方法があります:

```
shell> mysqlbinlog binlog.000001 > /tmp/statements.sql  
shell> mysqlbinlog binlog.000002 >> /tmp/statements.sql  
shell> mysql -u root -p -e "source /tmp/statements.sql"
```

GTID (セクション17.1.3「グローバルトランザクション識別子を使用したレプリケーション」を参照)を含むバイナリログから読み取りながら、ダンプファイルに書き込む場合、次のように、`mysqlbinlog` で `--skip-gtids` オプションを使用します。

```
shell> mysqlbinlog --skip-gtids binlog.000001 > /tmp/dump.sql  
shell> mysqlbinlog --skip-gtids binlog.000002 >> /tmp/dump.sql  
shell> mysql -u root -p -e "source /tmp/dump.sql"
```

7.5.2 イベントの位置を使用したポイントインタイムリカバリ

最後のセクション [セクション7.5.1「バイナリログを使用したポイントインタイムリカバリ」](#) では、バイナリログを使用してポイントインタイムリカバリを実行する一般的な考え方について説明します。このセクションでは、操作の詳細を例とともに説明します。

たとえば、2020年3月11日の約20:06:00に、テーブルを削除するSQLステートメントが実行されたとします。point-in-time リカバリを実行して、テーブルの削除直前の状態にサーバーをリストアできます。これを実現するためのサンプルステップを次に示します:

1. 目的の時点より前に作成された最後の全体バックアップをリストアします (この例では2020年3月11日の20:06:00である t_p と呼びます)。終了したら、あとで使用するためにサーバーを復元したバイナリログの位置を書き留め、サーバーを再起動します。

注記

最後にリカバリされたバイナリログの位置は、リストアおよびサーバーの再起動後に InnoDB によっても表示されますが、表示された位置に反映された時間の後に DDL イベントおよび InnoDB 以外の変更が発生した可能性があるため、リストアの終了ログの位置を取得する信頼性のない方法です。バックアップおよびリストアツールでは、リカバリ用の最後のバイナリログの位置を指定する必要があります: たとえば、タスクに `mysqlbinlog` を使用している場合は、バイナリログリプレイの停止位置を確認します。MySQL Enterprise Backup を使用している場合は、最後のバイナリログの位置がバックアップに保存されています。 [Point-in-Time Recovery](#) を参照してください。

2. データベースをリストアする時点に対応する正確なバイナリログイベント位置を検索します。この例では、テーブルの削除が行われたおよその時間 (t_p) がわかっている場合、`mysqlbinlog` ユーティリティを使用してその時間の前後のログコンテンツを確認することで、ログの位置を確認できます。 `--start-datetime` および `--stop-datetime` オプションを使用して、 t_p の前後の短い期間を指定し、出力でイベントを探します。例:

```
shell> mysqlbinlog --start-datetime="2020-03-11 20:05:00" \  
--stop-datetime="2020-03-11 20:08:00" --verbose \  
/var/lib/mysql/bin.123456 | grep -C 15 "DROP TABLE"  
  
/*!80014 SET @@session.original_server_version=80019/*!*/;  
/*!80014 SET @@session.immediate_server_version=80019/*!*/;  
SET @@SESSION.GTID_NEXT= 'ANONYMOUS'/*!*/;  
# at 232  
#200311 20:06:20 server id 1 end_log_pos 355 CRC32 0x2fc1e5ea Query thread_id=16 exec_time=0 error_code=0  
SET TIMESTAMP=1583971580/*!*/;  
SET @@session.pseudo_thread_id=16/*!*/;  
SET @@session.foreign_key_checks=1, @@session.sql_auto_is_null=0, @@session.unique_checks=1, @@session.autocommit=1/*!*/;  
SET @@session.sql_mode=1168113696/*!*/;  
SET @@session.auto_increment_increment=1, @@session.auto_increment_offset=1/*!*/;  
/*!C utf8mb4 *//*!*/;  
SET @@session.character_set_client=255,@@session.collation_connection=255,@@session.collation_server=255/*!*/;  
SET @@session.lc_time_names=0/*!*/;  
SET @@session.collation_database=DEFAULT/*!*/;  
/*!80011 SET @@session.default_collation_for_utf8mb4=255/*!*/;  
DROP TABLE `pets`.`cats` /* generated by server */  
/*!*/;
```

```
# at 355
#200311 20:07:48 server id 1 end_log_pos 434 CRC32 0x123d65df Anonymous_GTID last_committed=1 sequence_number=2 rbr_only=no original_committed=
# original_commit_timestamp=1583971668462467 (2020-03-11 20:07:48.462467 EDT)
# immediate_commit_timestamp=1583971668462467 (2020-03-11 20:07:48.462467 EDT)
/*!80001 SET @@session.original_commit_timestamp=1583971668462467/*!*/;
/*!80014 SET @@session.original_server_version=80019/*!*/;
/*!80014 SET @@session.immediate_server_version=80019/*!*/;
SET @@SESSION.GTID_NEXT= 'ANONYMOUS'/*!*/;
# at 434
#200311 20:07:48 server id 1 end_log_pos 828 CRC32 0x57fac9ac Query thread_id=16 exec_time=0 error_code=0 Xid = 217
use `pets`/*!*/;
SET TIMESTAMP=1583971668/*!*/;
/*!80013 SET @@session.sql_require_primary_key=0/*!*/;
CREATE TABLE dogs
```

mysqlbinlog の出力から、`DROP TABLE `pets`.`cats`` ステートメントは # at 232 行と # at 355 行の間のバイナリログのセグメントにあります。つまり、ステートメントはログ位置 232 のあとに実行され、ログは `DROP TABLE` ステートメントのあとの位置 355 にあります。

注記

対象となる実際のイベント位置の検索に役立つのは、`--start-datetime` および `--stop-datetime` オプションのみです。2 つのオプションを使用して適用するバイナリログセグメントの範囲を指定することはお勧めしません: オプションを使用すると、バイナリログイベントが欠落するリスクが高くなります。かわりに `--start-position` および `--stop-position` を使用してください。

- バイナリログファイル内のイベントをサーバーに適用します。これは、手順 1 で見つかったログ位置 (155 であると仮定) から始まり、手順 2 で見つけた、目的の時点 (232) より前の位置で終わります:

```
shell> mysqlbinlog --start-position=155 --stop-position=232 /var/lib/mysql/bin.123456 \
| mysql -u root -p
```

このコマンドは、すべてのトランザクションを開始位置から停止位置の直前までリカバリします。mysqlbinlog の出力には各 SQL ステートメントが記録される前に `SET TIMESTAMP` ステートメントが含まれるため、リカバリされたデータおよび関連する MySQL ログには、トランザクションが実行された元の時間が反映されます。

これで、テーブル `pets.cats` が削除される直前に、データベースが目的の時点である `tp` にリストアされました。

- 終了した point-in-time リカバリを超えて、目的の時点の後にすべてのステートメントも再実行する場合は、mysqlbinlog を再度使用して、`tp` の後のすべてのイベントをサーバーに適用します。ステップ 2 では、スキップするステートメントの後、ログは 355 の位置にあります。これを `--start-position` オプションに使用して、位置の後のステートメントを含めることができます:

```
shell> mysqlbinlog --start-position=355 /var/lib/mysql/bin.123456 \
| mysql -u root -p
```

データベースはバイナリログファイルに記録された最新のステートメントを復元しましたが、選択したイベントはスキップされました。

7.6 MyISAM テーブルの保守とクラッシュリカバリ

このセクションでは、`myisamchk` を使用して、MyISAM テーブル (データとインデックスを格納するための `.MYD` ファイルおよび `.MYI` ファイルのあるテーブル) をチェックまたは修復する方法について説明します。`myisamchk` の一般的な背景に関しては、[セクション 4.6.4 「myisamchk — MyISAM テーブルメンテナンスユーティリティ」](#) を参照してください。その他のテーブル修復情報については、[セクション 2.11.13 「テーブルまたはインデックスの再作成または修復」](#) にあります。

`myisamchk` を使用して、データベーステーブルをチェック、修復、または最適化できます。次のセクションでは、これらの操作を実行する方法と、テーブル保守スケジュールをセットアップする方法について説明します。`myisamchk` を使用して、テーブルに関する情報を取得することについては、[セクション 4.6.4.5 「myisamchk によるテーブル情報の取得」](#) を参照してください。

`myisamchk` によるテーブルの修復はきわめてセキュアですが、テーブルに対して多くの変更を行う可能性のある修復または保守操作を実行する前に、常にバックアップを作成することをお勧めします。

インデックスに影響する `myisamchk` 操作により、`MyISAM FULLTEXT` インデックスが、MySQL サーバーで使用されている値と互換性がない全文パラメータで再構築される可能性があります。この問題を回避するには、[セクション 4.6.4.1 「myisamchk の一般オプション」](#) のガイドラインに従ってください。

`MyISAM` テーブルの保守は、`myisamchk` が実行するものと似た操作を実行する SQL ステートメントを使用しても実行できます。

- `MyISAM` テーブルをチェックするには、`CHECK TABLE` を使用します。
- `MyISAM` テーブルを修復するには、`REPAIR TABLE` を使用します。
- `MyISAM` テーブルを最適化するには、`OPTIMIZE TABLE` を使用します。
- `MyISAM` テーブルを分析するには、`ANALYZE TABLE` を使用します。

これらのステートメントの詳細については、[セクション 13.7.3 「テーブル保守ステートメント」](#) を参照してください。

これらのステートメントは、直接または `mysqlcheck` クライアントプログラムを利用して使用できます。`myisamchk` に勝るこれらのステートメントの利点の 1 つは、サーバーがすべての作業を行うということです。`myisamchk` では、`myisamchk` とサーバー間で不要なやり取りがないように、サーバーが同時にテーブルを使用しないことを確認する必要があります。

7.6.1 クラッシュリカバリへの myisamchk の使用

このセクションでは、MySQL データベースのデータの破損をチェックし、処理する方法について説明します。テーブルが頻繁に破損する場合は、その理由を見つけるようにしてください。[セクション B.3.3.3 「MySQL が繰り返しくラッシュする場合の対処方法」](#) を参照してください。

`MyISAM` テーブルがどのように破損する可能性があるかについての説明は、[セクション 16.2.4 「MyISAM テーブルの問題点」](#) を参照してください。

外部ロックを無効にして `mysqld` を実行した (これはデフォルトです) 場合、`mysqld` が同じテーブルを使用中に、`myisamchk` を使用して、テーブルを確実にチェックすることはできません。`myisamchk` の実行中に `mysqld` を使用してテーブルにアクセスできないことが確実な場合は、テーブルのチェックを開始する前に `mysqladmin flush-tables` を実行する必要があります。これを保証できない場合は、テーブルのチェック中に、`mysqld` を停止する必要があります。`myisamchk` を実行して、`mysqld` が同時に更新しているテーブルをチェックすると、テーブルが破損していても、破損しているという警告を受け取ることがあります。

外部ロックを有効にしてサーバーを実行する場合は、`myisamchk` を使用していつでもテーブルをチェックできます。この場合、`myisamchk` が使用しているテーブルをサーバーが更新しようとする、サーバーは `myisamchk` が終了するまで待機してから続行します。

`myisamchk` を使用して、テーブルを修復または最適化する場合は、`mysqld` サーバーがそのテーブルを使用していないことを常に確認する必要があります (これは外部ロックが無効にされている場合にも適用されます)。`mysqld` を停止しない場合、`myisamchk` を実行する前に、少なくとも `mysqladmin flush-tables` を実行してください。サーバーと `myisamchk` が同時にテーブルにアクセスすると、テーブルが破損する可能性があります。

クラッシュリカバリを実行する場合、データベース内の各 `MyISAM` テーブル `tbl_name` が次の表に示すデータベースディレクトリ内の 3 つのファイルに対応することを理解しておくことが重要です。

ファイル	目的
<code>tbl_name.MYD</code>	データファイル
<code>tbl_name.MYI</code>	インデックスファイル

これらの 3 つのファイルの種類はそれぞれさまざまに破損することがありますが、ほとんどの場合に問題はデータファイルとインデックスファイルで発生します。

`myisamchk` は、`.MYD` データファイルのコピーを行ごとに作成することによって機能します。これは、古い `.MYD` ファイルを削除し、新しいファイルを元のファイル名に変更して、修復ステージを終了します。`--quick` を使用した場合、`myisamchk` は一時 `.MYD` ファイルを作成せず、代わりに `.MYD` ファイルが正しいとみなし、`.MYD` ファイルに手

を加えずに新しいインデックスファイルだけを生成します。 `myisamchk` は `.MYD` ファイルが破損しているかどうかを自動的に検出し、破損している場合は修復を中止するため、これは安全です。 `myisamchk` に `--quick` オプションを2回指定することもできます。この場合、`myisamchk` は一部のエラー (重複キーエラーなど) で中止せず、`.MYD` ファイルを修正して、それらを解決しようとします。通常、2つの `--quick` オプションの使用は、通常の修復を実行するためにディスクの空き容量が少なすぎる場合にのみ役立ちます。その場合、少なくとも `myisamchk` を実行する前に、テーブルのバックアップを作成してください。

7.6.2 MyISAM テーブルのエラーのチェック方法

MyISAM テーブルをチェックするには、次のコマンドを使用します。

- `myisamchk tbl_name`

これはすべてのエラーの 99.99% を発見します。これで発見できないエラーは、データファイルのみに関連する破損です (きわめてまれです)。テーブルをチェックする場合、通常、`myisamchk` をオプションなし、または `-s` (サイレント) オプションで実行してください。

- `myisamchk -m tbl_name`

これはすべてのエラーの 99.999% を発見します。それは最初にすべてのインデックスエントリでエラーをチェックし、次にすべての行を読み取ります。それは行内のすべてのキー値のチェックサムを計算し、チェックサムがインデックスツリー内のキーのチェックサムと一致することを確認します。

- `myisamchk -e tbl_name`

これはすべてのデータの完全で徹底的なチェックを実行します (`-e` は「拡張チェック」を意味します)。それは各行のすべてのキーのチェック読み取りを実行し、それらが実際に正しい行を指していることを確認します。これは、多数のインデックスを持つ大きなテーブルの場合に長い時間がかかることがあります。通常、`myisamchk` は見つかった最初のエラーのあとで停止します。詳細情報を取得する場合は、`-v` (verbose) オプションを追加できます。これにより、`myisamchk` は最大 20 のエラーまで続行します。

- `myisamchk -e -i tbl_name`

これは、前述のコマンドと同様ですが、`-i` オプションは `myisamchk` に追加の統計情報を出力するように伝えます。

ほとんどの場合、テーブルをチェックするためには、テーブル名以外の引数なしの単純な `myisamchk` コマンドで十分です。

7.6.3 MyISAM テーブルの修復方法

このセクションの説明では、MyISAM テーブル (拡張子 `.MYI` および `.MYD`) に対し `myisamchk` を使用方法について説明します。

さらに、`CHECK TABLE` および `REPAIR TABLE` ステートメントを使用して、MyISAM テーブルをチェックして修復することもできます。 [セクション13.7.3.2「CHECK TABLE ステートメント」](#) および [セクション13.7.3.5「REPAIR TABLE ステートメント」](#) を参照してください。

破損したテーブルの兆候として、予期せずに中止するクエリーや次のような観察可能なエラーが含まれます。

- ファイル `tbl_name.MYI` が見つかりません (エラーコード: `nnn`)。
- 予期しないファイルの終わり
- レコードファイルがクラッシュしました
- テーブルハンドラからエラー `nnn` を取得します

エラーの詳細を取得するには、`perror nnn` を実行します。ここで、`nnn` はエラー番号です。次の例は、`perror` を使用して、テーブルの問題を示すもっとも一般的なエラー番号の意味を見つける方法を示しています。

```
shell> perror 126 127 132 134 135 136 141 144 145
MySQL error code 126 = Index file is crashed
MySQL error code 127 = Record-file is crashed
MySQL error code 132 = Old database file
```

```
MySQL error code 134 = Record was already deleted (or record file crashed)
MySQL error code 135 = No more room in record file
MySQL error code 136 = No more room in index file
MySQL error code 141 = Duplicate unique key or constraint on write or update
MySQL error code 144 = Table is crashed and last repair failed
MySQL error code 145 = Table was marked as crashed and should be repaired
```

エラー 135 (レコードファイルに空きがない) およびエラー 136 (インデックスファイルに空きがない) は、単純な修復で修正できるエラーではありません。この場合、`ALTER TABLE` を使用して、`MAX_ROWS` および `AVG_ROW_LENGTH` テーブルオプションの値を増やす必要があります。

```
ALTER TABLE tbl_name MAX_ROWS=xxx AVG_ROW_LENGTH=yyy;
```

現在のテーブルオプション値が不明な場合は、`SHOW CREATE TABLE` を使用します。

その他のエラーの場合は、テーブルを修復する必要があります。`myisamchk` は通常発生するほとんどの問題を検出し、修正できます。

修復プロセスには、ここで説明する 3 つのステージが含まれます。始める前に、場所をデータベースディレクトリに変更し、テーブルファイルの権限をチェックしてください。Unix では、`mysqld` を実行するユーザーによって (およびチェックするファイルにアクセスする必要があるため、チェックするユーザーにも)、それらが読み取り可能であることを確認します。ファイルを変更する必要があることが分かったら、それらに書き込みできる必要もあります。

このセクションでは、テーブルチェックが失敗した (セクション 7.6.2 「MyISAM テーブルのエラーのチェック方法」で説明しているものなど) 場合、または `myisamchk` が提供する拡張機能を使用する場合について説明します。

テーブル保守に使用される `myisamchk` オプションについては、セクション 4.6.4 「`myisamchk` — MyISAM テーブルメンテナンスユーティリティ」で説明しています。`myisamchk` には、パフォーマンスを向上できるメモリー割り当てを制御するために設定できる変数もあります。セクション 4.6.4.6 「`myisamchk` メモリー使用量」を参照してください。

コマンド行からテーブルを修復する場合は、まず `mysqld` サーバーを停止する必要があります。リモートサーバーで `mysqldadmin shutdown` を実行すると、`mysqldadmin` が戻ったあとに、すべてのステートメント処理が停止し、すべてのインデックス変更がディスクにフラッシュされるまで、しばらくの間 `mysqld` サーバーがまだ使用できることに注意してください。

ステージ 1: テーブルのチェック

`myisamchk *.MYI` または時間があれば `myisamchk -e *.MYI` を実行します。 `-s` (サイレント) オプションを使用すると、不要な情報を抑制します。

`mysqld` サーバーが停止している場合は、`--update-state` オプションを使用して、`myisamchk` にテーブルを「チェック済み」とマークするように指示してください。

`myisamchk` がエラーを報告しているテーブルだけを修復する必要があります。そのようなテーブルの場合、ステージ 2 に進みます。

チェック時に、予期しないエラー (out of memory エラーなど) を受け取った場合、または `myisamchk` がクラッシュした場合、ステージ 3 へ進みます。

ステージ 2: 簡単で安全な修復

まず `myisamchk -r -q tbl_name` を試します (`-r -q` は「クイックリカバリモード」を意味します)。これは、データファイルにアクセスせずに、インデックスファイルを修復しようとしています。データファイルに、必要なすべてのものが含まれ、削除リンクがデータファイル内の正しい場所を指している場合、これは機能するはずであり、テーブルが修正されます。次のテーブルの修復を開始します。そうでない場合は、次の手順を使用します。

1. 続行する前に、データファイルのバックアップを作成します。
2. `myisamchk -r tbl_name` を使用します (`-r` は「リカバリモード」を意味します)。これによって、正しくない行と削除された行がデータファイルから削除され、インデックスファイルが再構築されます。
3. 先述のステップが失敗した場合、`myisamchk --safe-recover tbl_name` を使用します。安全なリカバリモードでは、通常のリカバリモードで扱われないわずかなケースを処理する古いリカバリ方法を使用します (ただし遅くなります)。

注記

修復操作を大幅に高速化する場合、`sort_buffer_size` および `key_buffer_size` 変数の値をそれぞれ、`myisamchk` の実行時に使用可能なメモリーの約 25% に設定してください。

修復時に、予期しないエラー (`out of memory` エラーなど) を受け取った場合、または `myisamchk` がクラッシュした場合、ステージ 3 へ進みます。

ステージ 3: 困難な修復

このステージに到達するのは、インデックスファイル内の最初の 16K バイトのブロックが破損しているか、誤った情報が含まれている場合、またはインデックスファイルが失われている場合に限られるはずですが。この場合、新しいインデックスファイルを作成する必要があります。次のように実行します。

1. データファイルを安全な場所に移動します。
2. テーブル記述ファイルを使用して、新しい (空の) データファイルとインデックスファイルを作成します。

```
shell> mysql db_name
```

```
mysql> SET autocommit=1;
mysql> TRUNCATE TABLE tbl_name;
mysql> quit
```

3. 古いデータファイルを新しく作成したデータファイルにコピーします。(古いファイルを新しいファイルに単に移動しないでください。何か異常があった場合に備えて、コピーを保持する必要があります。)

重要

レプリケーションを使用している場合、それにはファイルシステム操作が含まれ、これらは MySQL によって記録されないため、上記の手順を実行する前に、それを停止してください。

ステージ 2 に戻ります。`myisamchk -r -q` が機能するはずですが。(これは無限ループにならないはずですが。)

手順全体を自動的に実行する `REPAIR TABLE tbl_name USE_FRM SQL` ステートメントを使用することもできます。`REPAIR TABLE` を使用すると、サーバーがすべての作業を実行するため、ユーティリティーとサーバー間の不要なやり取りの可能性もなくなります。[セクション 13.7.3.5 「REPAIR TABLE ステートメント」](#) を参照してください。

7.6.4 MyISAM テーブルの最適化

断片化した行を結合し、行の削除または更新の結果発生した無駄な領域を削除するには、`myisamchk` をリカバリモードで実行します。

```
shell> myisamchk -r tbl_name
```

`OPTIMIZE TABLE SQL` ステートメントを使用して、同様にテーブルを最適化することができます。`OPTIMIZE TABLE` はテーブルの修復とキー分析を行い、さらに、キーのルックアップが速くなるように、インデックスツリーをソートします。`OPTIMIZE TABLE` を使用すると、サーバーがすべての作業を実行するため、ユーティリティーとサーバー間の不要なやり取りの可能性もなくなります。[セクション 13.7.3.4 「OPTIMIZE TABLE ステートメント」](#) を参照してください。

`myisamchk` には、テーブルのパフォーマンスを向上させるために使用できる多数のその他オプションがあります。

- `--analyze` または `-a`: キー分布分析を実行します。これは、結合オプティマイザが、テーブルを結合する順番と、それが使用するインデックスをより適切に選択できるようにすることで、結合パフォーマンスを向上させます。
- `--sort-index` または `-S`: インデックスブロックをソートします。これはシークを最適化し、インデックスを使用するテーブルスキャンを高速化します。
- `--sort-records=index_num` または `-R index_num`: 特定のインデックスに従って、データ行をソートします。これにより、データが大幅に局所に集中化されるため、このインデックスを使用する、範囲に基づいた `SELECT` または `ORDER BY` 操作が高速化する可能性があります。

利用可能なすべてのオプションの完全な説明については、[セクション4.6.4「myisamchk — MyISAM テーブルメンテナンスユーティリティ」](#)を参照してください。

7.6.5 MyISAM テーブル保守スケジュールのセットアップ

問題が発生するのを待つより、テーブルチェックを定期的に行うことをお勧めします。MyISAM テーブルをチェックまたは修復する1つの方法は、[CHECK TABLE](#) および [REPAIR TABLE](#) ステートメントを使用することです。[セクション13.7.3「テーブル保守ステートメント」](#)を参照してください。

テーブルをチェックする別の方法は、[myisamchk](#) を使用することです。保守の目的には、[myisamchk -s](#) を使用できます。[-s](#) オプション ([--silent](#) の短縮形) により、サイレントモードで [myisamchk](#) が実行され、エラーが発生した場合のみ、メッセージが出力されます。

自動 MyISAM テーブルチェックを有効にすることをお勧めします。たとえば、マシンが更新の途中で再起動を実行した場合、通常、影響を受けた可能性のある各テーブルが使用される前に、それをチェックする必要があります。(これらは「クラッシュしたテーブルが必要です。」) サーバーが MyISAM テーブルを自動的にチェックするには、[myisam_recover_options](#) システム変数を設定して起動します。[セクション5.1.8「サーバーシステム変数」](#)を参照してください。

通常のシステム操作時にも定期的にテーブルをチェックしてください。たとえば、[crontab](#) ファイル内の次のような行を使用して、週1回 [cron](#) ジョブを実行し、重要なテーブルをチェックします。

```
35 0 * * 0 /path/to/myisamchk --fast --silent /path/to/datadir/*/*.MYI
```

これはクラッシュしたテーブルに関する情報を出力するため、テーブルを調査し、必要に応じて修復できます。

はじめに、過去 24 時間中に更新されたすべてのテーブルに対して、毎晩 [myisamchk -s](#) を実行します。その問題がまれにしか発生しないことがわかったら、チェックの頻度を週1回などに減らすことができます。

通常、MySQL テーブルはほとんど保守が必要ありません。動的サイズの行のある MyISAM テーブル ([VARCHAR](#)、[BLOB](#)、または [TEXT](#) カラムのあるテーブル) に何回も更新を実行するか、または多くの削除された行のあるテーブルがある場合、ときどきテーブルの領域をデフラグ/再利用する必要がある場合があります。これは、問題のテーブルに [OPTIMIZE TABLE](#) を使用して実行できます。または、しばらくの間、[mysqld](#) サーバーを停止できる場合は、サーバーの停止中に、場所をデータディレクトリ内に変更し、次のコマンドを使用します。

```
shell> myisamchk -r -s --sort-index --myisam_sort_buffer_size=16M /*/*.MYI
```


第 8 章 最適化

目次

8.1 最適化の概要	1430
8.2 SQL ステートメントの最適化	1432
8.2.1 SELECT ステートメントの最適化	1432
8.2.2 サブクエリー、導出テーブル、ビュー参照および共通テーブル式の最適化	1479
8.2.3 INFORMATION_SCHEMA クエリーの最適化	1492
8.2.4 パフォーマンススキーマクエリーの最適化	1494
8.2.5 データ変更ステートメントの最適化	1496
8.2.6 データベース権限の最適化	1497
8.2.7 その他の最適化のヒント	1497
8.3 最適化とインデックス	1498
8.3.1 MySQL のインデックスの使用の仕組み	1498
8.3.2 主キーの最適化	1499
8.3.3 SPATIAL インデックス最適化	1499
8.3.4 外部キーの最適化	1500
8.3.5 カラムインデックス	1500
8.3.6 マルチカラムインデックス	1501
8.3.7 インデックスの使用の確認	1503
8.3.8 InnoDB および MyISAM インデックス統計コレクション	1503
8.3.9 B ツリーインデックスとハッシュインデックスの比較	1504
8.3.10 インデックス拡張の使用	1506
8.3.11 生成されたカラムインデックスのオプティマイザによる使用	1508
8.3.12 不可視のインデックス	1509
8.3.13 降順インデックス	1511
8.3.14 TIMESTAMP カラムからのインデックス付きルックアップ	1512
8.4 データベース構造の最適化	1514
8.4.1 データサイズの最適化	1514
8.4.2 MySQL データ型の最適化	1515
8.4.3 多数のテーブルの最適化	1516
8.4.4 MySQL での内部一時テーブルの使用	1518
8.4.5 データベースおよびテーブルの数に対する制限	1521
8.4.6 テーブルサイズの制限	1521
8.4.7 テーブルカラム数と行サイズの制限	1523
8.5 InnoDB テーブルの最適化	1525
8.5.1 InnoDB テーブルのストレージレイアウトの最適化	1525
8.5.2 InnoDB トランザクション管理の最適化	1526
8.5.3 InnoDB の読み取り専用トランザクションの最適化	1527
8.5.4 InnoDB redo ロギングの最適化	1527
8.5.5 InnoDB テーブルの一括データロード	1528
8.5.6 InnoDB クエリーの最適化	1530
8.5.7 InnoDB DDL 操作の最適化	1530
8.5.8 InnoDB ディスク I/O の最適化	1530
8.5.9 InnoDB 構成変数の最適化	1533
8.5.10 多くのテーブルのあるシステムに対する InnoDB の最適化	1535
8.6 MyISAM テーブルの最適化	1535
8.6.1 MyISAM クエリーの最適化	1535
8.6.2 MyISAM テーブルの一括データロード	1536
8.6.3 REPAIR TABLE ステートメントの最適化	1537
8.7 MEMORY テーブルの最適化	1538
8.8 クエリー実行プランの理解	1539
8.8.1 EXPLAIN によるクエリーの最適化	1539
8.8.2 EXPLAIN 出カフォーマット	1540
8.8.3 拡張 EXPLAIN 出力形式	1552
8.8.4 名前付き接続の実行計画情報の取得	1554

8.8.5 クエリーパフォーマンスの推定	1555
8.9 クエリー最適化の制御	1555
8.9.1 クエリー計画評価の制御	1555
8.9.2 切り替え可能な最適化	1556
8.9.3 オプティマイザヒント	1565
8.9.4 インデックスヒント	1579
8.9.5 オプティマイザコストモデル	1581
8.9.6 オプティマイザ統計	1584
8.10 バッファリングとキャッシュ	1587
8.10.1 InnoDB バッファプールの最適化	1587
8.10.2 MyISAM キーキャッシュ	1587
8.10.3 プリペアドステートメントおよびストアドプログラムのキャッシュ	1592
8.11 ロック操作の最適化	1593
8.11.1 内部ロック方法	1593
8.11.2 テーブルロックの問題	1595
8.11.3 同時挿入	1597
8.11.4 メタデータのロック	1597
8.11.5 外部ロック	1600
8.12 MySQL サーバーの最適化	1601
8.12.1 ディスク I/O の最適化	1601
8.12.2 シンボリックリンクの使用	1603
8.12.3 メモリーの使用の最適化	1605
8.13 パフォーマンスの測定 (ベンチマーク)	1611
8.13.1 式と関数の速度の測定	1611
8.13.2 独自のベンチマークの使用	1611
8.13.3 performance_schema によるパフォーマンスの測定	1612
8.14 サーバースレッド (プロセス) 情報の確認	1612
8.14.1 プロセスリストへのアクセス	1612
8.14.2 スレッドのコマンド値	1614
8.14.3 一般的なスレッドの状態	1616
8.14.4 レプリケーションソーススレッドの状態	1622
8.14.5 レプリケーション I/O スレッドの状態	1622
8.14.6 レプリケーション SQL スレッドの状態	1623
8.14.7 レプリケーション接続スレッドの状態	1624
8.14.8 NDB Cluster スレッドの状態	1624
8.14.9 イベントスケジューラスレッドの状態	1625

この章では、MySQL のパフォーマンスを最適化する方法について説明し、例を示します。最適化には、いくつかのレベルでの構成、チューニング、およびパフォーマンスの測定が含まれます。業務の役割 (開発者、データベース管理者、または両方の組み合わせ) に応じて、個々の SQL ステートメント、アプリケーション全体、単一のデータベースサーバー、または複数のネットワーク接続されたデータベースサーバーのレベルで最適化できます。プロアクティブにパフォーマンスを事前に計画する場合や、または問題の発生後に、構成やコードの問題のトラブルシューティングを行う場合があります。CPU やメモリーの使用を最適化することで、スケーラビリティを向上し、データベースを低下させず、より多くの負荷を処理させることもできます。

8.1 最適化の概要

データベースのパフォーマンスは、テーブル、クエリー、構成設定など、データベースレベルの複数の要因に依存します。これらのソフトウェア構造は、ハードウェアレベルでの CPU および I/O 操作につながり、それらを最小限にし、可能な限り効率的にする必要があります。データベースのパフォーマンスを行う際は、ソフトウェア側の高レベルのルールとガイドラインについて学び、時計を使ってパフォーマンスを測定することから始めます。熟練するにつれ、内部で起こっていることについて詳しく学習し、CPU サイクルや I/O 操作などの測定を開始します。

一般的なユーザーの目標は、既存のソフトウェアやハードウェア構成から、最高のデータベースパフォーマンスを得ることです。上級ユーザーは、MySQL ソフトウェア自体を改善する機会を見つけたり、独自のストレージエンジンやハードウェアアプライアンスを開発して、MySQL エコシステムを拡張したりします。

- [データベースレベルでの最適化](#)

- [ハードウェアレベルでの最適化](#)
- [移植性とパフォーマンスのバランス](#)

データベースレベルでの最適化

データベースアプリケーションを高速にすることにおいてもっとも重要な要素は、その基本設計です。

- テーブルは適切に構築されていますか。特に、カラムに適切なデータ型があり、各テーブルに、作業の種類に適切なカラムがありますか。たとえば、頻繁な更新を実行するアプリケーションでは、多くの場合に少数のカラムのある多数のテーブルを使用し、大量のデータを解析するアプリケーションでは、多くの場合に多数のカラムのある少数のテーブルを使用します。
- クエリーを効率的にするため、適切な[インデックス](#)が設定されていますか。
- テーブルごとに適切なストレージエンジンを使用しており、使用している各ストレージエンジンの長所と機能を生かしていますか。特に、[InnoDB](#)などのトランザクションストレージエンジンまたは[MyISAM](#)などの非トランザクションストレージエンジンの選択は、パフォーマンスとスケーラビリティにきわめて重要な場合があります。

注記

[InnoDB](#) は、新しいテーブルのデフォルトのストレージエンジンです。実際に、高度な [InnoDB](#) パフォーマンス機能は、[InnoDB](#) テーブルが、特にビジューなデータベースに対して、多くの場合に単純な [MyISAM](#) テーブルよりパフォーマンスが優れていることを意味します。

- 各テーブルは適切な行フォーマットを使用していますか。この選択は、テーブルに使用されるストレージエンジンによっても異なります。特に、圧縮テーブルは使用するディスク領域が減るため、データの読み取りと書き込みに必要なディスク I/O も少なくなります。圧縮は、[InnoDB](#) テーブルでのあらゆる種類のワークロードと、読み取り専用 [MyISAM](#) テーブルに使用できます。
- アプリケーションでは、適切な[ロック戦略](#)を使用していますか。たとえば、データベース操作を同時に実行できるように、可能なかぎり共有アクセスを許可したり、重要な操作が最優先されるように、適切な場合に排他的アクセスを要求したりするなどです。ここでも、ストレージエンジンの選択が重要です。[InnoDB](#) ストレージエンジンは、ユーザーが関与せずに、ほとんどのロックの問題を処理するため、データベースの同時実行性を向上し、コードの実験やチューニングの量を削減できます。
- [キャッシュ](#)に使用される[メモリー領域](#)がすべて正しくサイズ設定されていますか。つまり、頻繁にアクセスされるデータを保持するために十分な大きさがありながらも、物理メモリーをオーバーロードし、ページングを発生させるほど大きくしません。構成するメインメモリー領域は、[InnoDB](#) バッファプールおよび [MyISAM](#) キーキャッシュです。

ハードウェアレベルでの最適化

データベースがビジューになるほど、どんなデータベースアプリケーションも最終的にハードウェアの制限に達します。データベース管理者は、アプリケーションをチューニングするか、サーバーを再構成してこれらの[ボトルネック](#)を回避できるかどうか、または追加のハードウェアリソースが必要かどうかを評価する必要があります。システムボトルネックは一般に次の原因から発生します。

- ディスクシーク。ディスクがデータを検索するには時間がかかります。最新のディスクでは、通常この平均時間が 10 ms 未満であるため、理論的には 1 秒間に約 100 シーク実行できることとなります。この時間は、新しいディスクでは徐々に改善されますが、1つのテーブルに対して最適化することはきわめて困難です。シーク時間を最適化する方法は、複数のディスクにデータを分散することです。
- ディスクの読み取りと書き込み。ディスクが正しい位置にある場合に、データを読み取りまたは書き込みする必要があります。最新のディスクでは、1つのディスクで少なくとも 10 - 20M バイト/秒のスループットを実現します。これは、複数のディスクから並列で読み取ることができると、シークより最適化が簡単です。
- CPU サイクル。データがメインメモリー内にある場合、結果を得るために、それを処理する必要があります。メモリーの量と比較して大きなテーブルを使用することは、もっとも一般的な制限要因となります。しかし、小さいテーブルでは、通常速度は問題になりません。

- メモリー帯域幅。CPU で、CPU キャッシュに収められるより多くのデータを必要とする場合、メインメモリーの帯域幅がボトルネックになります。これは、ほとんどのシステムでまれなボトルネックですが、認識しておくべきです。

移植性とパフォーマンスのバランス

ポータブル MySQL プログラムで、パフォーマンス指向の SQL 拡張を使用するには、ステートメント内の MySQL 固有のキーワードを `/*! */` コメント区切り文字で囲むことができます。ほかの SQL サーバーはコメントにされたキーワードを無視します。コメントの作成については、[セクション9.7「コメント」](#)を参照してください。

8.2 SQL ステートメントの最適化

インタプリタから直接発行されるか、API によって内部で送信されるかに関係なく、データベースアプリケーションのコアロジックは SQL ステートメントによって実行されます。このセクションのチューニングのガイドラインは、あらゆる種類の MySQL アプリケーションの高速化に役立ちます。このガイドラインでは、データを読み取りおよび書き込みする SQL 操作、一般的な SQL 操作の内部オーバーヘッド、およびデータベースモニタリングなどの特定のシナリオで使われる操作について説明します。

8.2.1 SELECT ステートメントの最適化

SELECT ステートメントの形式のクエリーは、データベースのすべてのルックアップ操作を実行します。動的 Web ページの 1 秒未満の応答時間を達成するためでも、または巨大な夜間のレポートを生成するための時間から数時間を取り除くためでも、これらのステートメントのチューニングは最優先です。

SELECT ステートメントに加えて、クエリーのチューニング手法は **DELETE** ステートメントの **CREATE TABLE...AS SELECT**、**INSERT INTO...SELECT**、**WHERE** 句などの構成要素にも適用されます。これらのステートメントは、書き込み操作と読み取り指向クエリー操作を組み合わせるため、パフォーマンスに関する追加の考慮事項があります。

NDB Cluster は結合プッシュダウン最適化をサポートしており、そこで適格な結合が NDB Cluster データノードに完全に送信され、そこでそれらのノード間で分散して並列で実行できます。この最適化の詳細は、[NDB プッシュダウン結合の条件](#)を参照してください。

クエリーの最適化の主な考慮事項は次のとおりです。

- 遅い **SELECT ... WHERE** クエリーを高速化するため、最初に確認することは、[インデックス](#)を追加できるかどうかです。**WHERE** 句で使用するカラムにインデックスをセットアップし、評価、フィルタリング、および最終的な結果の取得を高速化します。無駄なディスク領域を避けるため、アプリケーションで使用される多くの関連クエリーを高速化する少数のインデックスのセットを構築します。

インデックスは、[結合](#)や[外部キー](#)などの機能を使用して、さまざまなテーブルを参照するクエリーに特に重要です。**EXPLAIN** ステートメントを使用して、**SELECT** に使用するインデックスを判断できます。[セクション 8.3.1「MySQL のインデックスの使用の仕組み」](#)および[セクション8.8.1「EXPLAIN によるクエリーの最適化」](#)を参照してください。

- 過度な時間がかかる関数呼び出しなどのクエリーの部分を特定し、チューニングします。クエリーの構築の仕方によっては、関数が結果セットのすべての行に対して 1 回ずつ、さらにはテーブル内のすべての行に対して 1 回ずつ呼び出されるなど、大幅に非効率性を拡大させていることがあります。
- 特に大きなテーブルの場合に、クエリーでの[完全テーブルスキャン](#)の回数を最小にします。
- **ANALYZE TABLE** ステートメントを定期的に使用して、テーブル統計を最新に維持し、オプティマイザが、効率的な実行プランを立てるために必要な情報が得られるようにします。
- チューニング技法、インデックス作成技法、および各テーブルのストレージエンジンに固有の構成パラメータについて学習します。[InnoDB](#)と[MyISAM](#)のどちらでも、クエリーの高いパフォーマンスを可能にし、維持するための一連のガイドラインがあります。詳細については、[セクション8.5.6「InnoDB クエリーの最適化」](#)および[セクション8.6.1「MyISAM クエリーの最適化」](#)を参照してください。
- [セクション8.5.3「InnoDB の読み取り専用トランザクションの最適化」](#)の手法を使用して、[InnoDB](#) テーブルの単一クエリートランザクションを最適化できます。

- 特にオプティマイザで同じ変換の一部を自動的に実行する場合、理解が困難になるようなクエリーの変換を避けます。
- いずれかの基本ガイドラインによって、パフォーマンスの問題が簡単に解決されない場合、**EXPLAIN** プランを読み、インデックス、**WHERE** 句、結合句などを調整して、特定のクエリーの内部の詳細を調査します。(ある程度の専門技術に達している場合は、**EXPLAIN** プランを読むことがすべてのクエリーの最初の手順になると考えられます。)
- MySQL がキャッシュに使用するメモリー領域のサイズとプロパティを調整します。**InnoDB バッファプール**、**MyISAM** キーキャッシュ、および MySQL クエリーキャッシュの効率的な使用によって、2 回目以降、メモリーから結果が取得されるため、繰り返しのクエリーの実行が高速化します。
- キャッシュメモリー領域を使用して高速に実行するクエリーでも、必要なキャッシュメモリーを減らして、アプリケーションがよりスケーラブルになるように、さらに最適化できます。スケーラビリティは、パフォーマンスを大幅に低下させずに、アプリケーションでより多くの同時ユーザー、大きなリクエストなどを処理できることを意味します。
- クエリーの速度が、テーブルに同時にアクセスしているほかのセッションによって影響を受ける可能性があるロックの問題を処理します。

8.2.1.1 WHERE 句の最適化

このセクションでは、**WHERE** 句の処理で実行可能な最適化について説明します。例では **SELECT** ステートメントを使用していますが、**DELETE** および **UPDATE** ステートメント内の **WHERE** 句にも同じ最適化を適用します。

注記

MySQL オプティマイザへの取り組みは継続中であるため、MySQL が実行する最適化のすべてをここで説明しているわけではありません。

読みやすさを犠牲にしても、算術演算を高速化するように、クエリーを書き換えたいと考えがちです。MySQL では同様の最適化を自動的に実行するため、多くの場合にこの作業を回避でき、クエリーを理解しやすく、保守しやすい形式のままにしておくことができます。MySQL によって実行される最適化の一部を次に示します。

- 不要なかつこの削除:

```
((a AND b) AND c OR ((a AND b) AND (c AND d)))
-> (a AND b AND c) OR (a AND b AND c AND d)
```

- 定数量み込み:

```
(a<b AND b=c) AND a=5
-> b>5 AND b=c AND a=5
```

- 一定条件の削除:

```
(b>=5 AND b=5) OR (b=6 AND 5=5) OR (b=7 AND 5=6)
-> b=5 OR b=6
```

MySQL 8.0.14 以降では、これは最適化フェーズではなく準備中に行われるため、結合の簡略化に役立ちます。詳細および例は、[セクション8.2.1.9「外部結合の最適化」](#)を参照してください。

- インデックスによって使用される定数式は 1 回だけ評価されます。
- MySQL 8.0.16 以降では、定数値を持つ数値型のカラムの比較がチェックされて折りたたまれるか、無効な値または範囲外の値がないかが削除されます:

```
# CREATE TABLE t (c TINYINT UNSIGNED NOT NULL);
SELECT * FROM t WHERE c << 256;
-> SELECT * FROM t WHERE 1;
```

詳しくは[セクション8.2.1.14「定数 - フォールディングの最適化」](#), をご覧ください。

- **WHERE** を使用しない単一テーブルの **COUNT(*)** は、**MyISAM** テーブルと **MEMORY** テーブルのテーブル情報から直接取得されます。これは、1 つだけのテーブルで使用された場合に、**NOT NULL** 式にも実行されます。

- 無効な定数式の早期の検出。MySQL は一部の **SELECT** ステートメントが実行不可能であることをすみやかに検出し、行を返しません。
- **GROUP BY** または集約関数 (**COUNT()**、**MIN()** など) を使用しない場合、**HAVING** は **WHERE** とマージされます。
- 結合内の各テーブルについて、テーブルの高速の **WHERE** 評価を取得し、可能なかぎり早く行をスキップするために、より単純な **WHERE** が構築されます。
- クエリー内のほかのすべてのテーブルの前に、まず、すべての定数テーブルが読み取られます。定数テーブルは次のいずれかです。
 - 空白のテーブルまたは 1 行のテーブル。
 - **PRIMARY KEY** または **UNIQUE** インデックスでの **WHERE** 句で使用されるテーブル。ここではすべてのインデックス部分が定数式と比較され、**NOT NULL** として定義されます。

次のテーブルはすべて定数テーブルとして使用されます。

```
SELECT * FROM t WHERE primary_key=1;
SELECT * FROM t1,t2
WHERE t1.primary_key=1 AND t2.primary_key=t1.id;
```

- テーブルを結合するための最適な結合の組み合わせは、すべての可能性を試してみることで見つかります。**ORDER BY** および **GROUP BY** 句内のすべてのカラムが同じテーブルにある場合、結合する際に最初にそのテーブルが選ばれます。
- **ORDER BY** 句と別の **GROUP BY** 句がある場合、または、**ORDER BY** または **GROUP BY** に結合キュー内の最初のテーブルと異なるテーブルのカラムが含まれている場合は、一時テーブルが作成されます。
- **SQL_SMALL_RESULT** 修飾子を使用する場合、MySQL はインメモリー一時テーブルを使用します。
- オプティマイザがテーブルスキャンを使用する方が効率的であると判断しないかぎり、各テーブルインデックスがクエリーされ、最適なインデックスが使用されます。かつて、スキャンは、最適なインデックスがテーブルの 30% 超にまたがっているかどうかに基づいて使用されていましたが、固定のパーセンテージによって、インデックスを使用するか、スキャンを使用するかを選択が決定されなくなりました。現在のオプティマイザは複雑になり、テーブルサイズ、行数、I/O ブロックサイズなどの追加の要因に基づいて推定します。
- 場合によって、MySQL はデータファイルを参照しなくてもインデックスから行を読み取ることができます。インデックスから使用されるすべてのカラムが数値の場合、クエリーの解決にインデックスツリーのみが使用されます。
- 各行が出力される前に、**HAVING** 句に一致しないものはスキップされます。

きわめて高速なクエリーのいくつかの例:

```
SELECT COUNT(*) FROM tbl_name;

SELECT MIN(key_part1),MAX(key_part1) FROM tbl_name;

SELECT MAX(key_part2) FROM tbl_name
WHERE key_part1=constant;

SELECT ... FROM tbl_name
ORDER BY key_part1,key_part2,... LIMIT 10;

SELECT ... FROM tbl_name
ORDER BY key_part1 DESC, key_part2 DESC, ... LIMIT 10;
```

MySQL は、インデックス設定されたカラムが数値であるとして、インデックスツリーのみを使用して、次のクエリーを解決します。

```
SELECT key_part1,key_part2 FROM tbl_name WHERE key_part1=val;

SELECT COUNT(*) FROM tbl_name
WHERE key_part1=val1 AND key_part2=val2;

SELECT key_part2 FROM tbl_name GROUP BY key_part1;
```

次のクエリーは、個別のソーティングパスを使用せずに、インデックスを使用して、ソート順で行を取得します。

```
SELECT ... FROM tbl_name  
ORDER BY key_part1,key_part2,... ;  
  
SELECT ... FROM tbl_name  
ORDER BY key_part1 DESC, key_part2 DESC, ... ;
```

8.2.1.2 range の最適化

range アクセスメソッドは単一のインデックスを使用して、1 つまたは複数のインデックス値間隔の中に含まれるテーブル行のサブセットを取得します。これは、シングルパートまたはマルチパートインデックスに使用できます。次の各セクションでは、オプティマイザが範囲アクセスを使用する条件について説明します。

- [単一部品インデックスの範囲アクセス方法](#)
- [マルチパートインデックスの範囲アクセス方法](#)
- [複数值比較の等価範囲の最適化](#)
- [スキャン範囲アクセス方法のスキップ](#)
- [行コンストラクタ式の範囲最適化](#)
- [範囲最適化のためのメモリー使用の制限](#)

単一部品インデックスの範囲アクセス方法

単一部品インデックスの場合、インデックス値の間隔は、「intervals」ではなく範囲条件として示される **WHERE** 句内の対応する条件によって便利に表現できます。

シングルパートインデックスの範囲条件の定義は次のとおりです。

- **BTREE** インデックスと **HASH** インデックスの両方で、キー部分と定数値の比較は、**=**, **<=>**, **IN()**, **IS NULL** または **IS NOT NULL** 演算子を使用する場合の範囲条件です。
- さらに、**BTREE** インデックスでは、**>**, **<**, **>=**, **<=**, **BETWEEN**, **!=**, または **<>** 演算子、または **LIKE** への引数が、ワイルドカード文字で始まっていない定数文字列である場合の **LIKE** 比較を使用した場合に、キーパートと定数値の比較は範囲条件です。
- すべてのインデックスタイプで、**OR** または **AND** と組み合わせた複数の範囲条件によって範囲条件が形成されます。

先述の「定数値」とは次のいずれかを意味します。

- クエリー文字列からの定数
- 同じ結合からの **const** または **system** テーブルのカラム
- 非相関サブクエリーの結果
- 以前の型の部分式からのみ構成された式

以下に **WHERE** 句内で範囲条件を使用したクエリーのいくつかの例を示します。

```
SELECT * FROM t1  
WHERE key_col > 1  
AND key_col < 10;  
  
SELECT * FROM t1  
WHERE key_col = 1  
OR key_col IN (15,18,20);  
  
SELECT * FROM t1  
WHERE key_col LIKE 'ab%'  
OR key_col BETWEEN 'bar' AND 'foo';
```

一部の非定数値は、オプティマイザ定数伝播フェーズで定数に変換できます。

MySQL は可能なインデックスごとに、**WHERE** 句から範囲条件を抽出しようとします。抽出プロセス時に、範囲条件の構築に使用できない条件はドロップされ、重複する範囲を生成する条件は組み合わせられて、空の範囲を生成する条件は削除されます。

key1 がインデックス設定されたカラムで **nonkey** がインデックス設定されていない、次のステートメントを考慮します。

```
SELECT * FROM t1 WHERE
(key1 < 'abc' AND (key1 LIKE 'abcde%' OR key1 LIKE '%b')) OR
(key1 < 'bar' AND nonkey = 4) OR
(key1 < 'uux' AND key1 > 'z');
```

キー **key1** の抽出プロセスは次のとおりです。

1. 元の **WHERE** 句から始めます。

```
(key1 < 'abc' AND (key1 LIKE 'abcde%' OR key1 LIKE '%b')) OR
(key1 < 'bar' AND nonkey = 4) OR
(key1 < 'uux' AND key1 > 'z')
```

2. **nonkey = 4** と **key1 LIKE '%b'** は、範囲スキャンに使用できないため、削除します。それらを削除する正しい方法は、範囲スキャンの実行時に一致する行を見落とさないように、それらを **TRUE** で置き換えることです。これらを **TRUE** の歩留まりに置き換えます:

```
(key1 < 'abc' AND (key1 LIKE 'abcde%' OR TRUE)) OR
(key1 < 'bar' AND TRUE) OR
(key1 < 'uux' AND key1 > 'z')
```

3. 常に **true** または **false** である条件を縮小します。

- **(key1 LIKE 'abcde%' OR TRUE)** は常に **true** です
- **(key1 < 'uux' AND key1 > 'z')** は常に **false** です

これらの条件を定数に置き換えると、次のようになります:

```
(key1 < 'abc' AND TRUE) OR (key1 < 'bar' AND TRUE) OR (FALSE)
```

不要な **TRUE** 定数および **FALSE** 定数を削除すると、次のようになります:

```
(key1 < 'abc') OR (key1 < 'bar')
```

4. 重複する間隔を 1 つに組み合わせて、範囲スキャンに使用される最終的な条件が生成されます。

```
(key1 < 'bar')
```

一般に (前の例で示したように)、範囲スキャンに使用される条件は、**WHERE** 句より制限がゆるくなります。MySQL は、範囲条件を満たすが、完全な **WHERE** 句でない行をフィルタ処理する追加のチェックを実行します。

範囲条件抽出アルゴリズムは、任意の深さのネストの **AND/OR** 構造を処理でき、その出力は **WHERE** 句内の条件が存在する順番に依存しません。

MySQL では、空間インデックスの **range** アクセス方法に対する複数の範囲のマージはサポートされていません。この制限を回避するには、同じ **SELECT** ステートメントで **UNION** を使用できますが、ただし、各空間述語は、別の **SELECT** に入れます。

マルチパートインデックスの範囲アクセス方法

マルチパートインデックスの範囲条件は、シングルパートインデックスの範囲条件の拡張です。マルチパートインデックスの範囲条件は、インデックス行を 1 つまたは複数のキータプル間隔内に入るように制限します。キータプル間隔は、インデックスからの順序付けを使用して、キータプルのセットに定義されます。

たとえば、**key1(key_part1, key_part2, key_part3)** として定義されたマルチパートインデックスと、キー順で示された次のキータプルのセットを考慮します。

```
key_part1 key_part2 key_part3
NULL      1         'abc'
NULL      1         'xyz'
```

NULL	2	'foo'
1	1	'abc'
1	1	'xyz'
1	2	'abc'
2	1	'aaa'

条件 `key_part1 = 1` は次の間隔を定義します。

```
(1,-inf,-inf) <= (key_part1,key_part2,key_part3) < (1,+inf,+inf)
```

間隔は前のデータセットの 4、5、6 番目のタプルをカバーし、range アクセスメソッドで使用できます。

対照的に、条件 `key_part3 = 'abc'` は単一の間隔を定義せず、range アクセスメソッドで使用できません。

次の説明では、マルチパートインデックスに対して、範囲条件がどのように作用するかを詳しく示します。

- **HASH** インデックスでは、同一の値を含む各間隔を使用できます。これは次の形式の条件に対してのみ、間隔を生成できることを意味します。

```
key_part1 cmp const1
AND key_part2 cmp const2
AND ...
AND key_partN cmp constN;
```

ここで、`const1`、`const2`、... は定数で、`cmp` は、`=`、`<=>`、または `IS NULL` 比較演算子のいずれかで、条件はすべてのインデックスパートをカバーします。(つまり、`N` パートインデックスの各パートに 1 つずつ `N` 条件があります。) たとえば、次は 3 パート **HASH** インデックスの範囲条件です。

```
key_part1 = 1 AND key_part2 IS NULL AND key_part3 = 'foo'
```

何を定数とみなすかの定義については、「[単一部品インデックスの範囲アクセス方法](#)」を参照してください。

- **BTREE** インデックスでは、各条件で `=`、`<=>`、`IS NULL`、`>`、`<`、`>=`、`<=`、`!=`、`<>`、`BETWEEN`、または `LIKE` `'pattern'` (ここで `'pattern'` はワイルドカードで始まらない) を使用して、キーパートと定数値を比較する、`AND` で組み合わせられた条件に、間隔を使用できます。条件に一致するすべての行を含む単一のキータプルを判断できる場合にかぎり、1 つの間隔を使用できます (または `<>` または `!=` を使用する場合は 2 つの間隔)。

最適化は、比較演算子が `=`、`<=>`、または `IS NULL` である場合にかぎり、追加のキーパートを使用して、間隔を判断しようとします。演算子が `>`、`<`、`>=`、`<=`、`!=`、`<>`、`BETWEEN`、または `LIKE` の場合、最適化はそれを使用しますが、追加のキーパートは考慮しません。次の式では、最適化は最初の比較からの `=` を使用します。さらに 2 番目の比較からの `>=` も使用しますが、それ以上のキーパートを考慮せず、間隔の構築に 3 番目の比較を使用しません。

```
key_part1 = 'foo' AND key_part2 >= 10 AND key_part3 > 10
```

単一の間隔は次のとおりです。

```
('foo',10,-inf) < (key_part1,key_part2,key_part3) < ('foo',+inf,+inf)
```

作成された間隔に初期条件よりも多い行が含まれる可能性があります。たとえば、前の間隔は値 `('foo', 11, 0)` を含みますが、これは元の条件を満たしません。

- 間隔内に含まれる行セットをカバーする条件が `OR` で組み合わせられている場合、それらは、それらの間隔の和集合内に含まれる行セットをカバーする条件を形成します。条件が `AND` で組み合わせられている場合、それらは間隔の共通集合内に含まれる行セットを対象とする条件を形成します。たとえば、2 パートインデックスでのこの条件の場合:

```
(key_part1 = 1 AND key_part2 < 2) OR (key_part1 > 5)
```

間隔は次のとおりです。

```
(1,-inf) < (key_part1,key_part2) < (1,2)
(5,-inf) < (key_part1,key_part2)
```

この例で、1 行目の間隔は、左境界に 1 つのキーパートを使用し、右境界に 2 つのキーパートを使用しています。2 行目の間隔は 1 つのキーパートのみを使用しています。EXPLAIN 出力の `key_len` カラムは、使用されたキープリフィクスの最大長を示しています。

場合によって、`key_len` はキーパートが使用されたことを示しますが、それが予期したものではないことがあります。`key_part1` と `key_part2` が `NULL` になることがあるとします。次に、`key_len` カラムに、次の条件の 2 つのキーパート長が表示されます。

```
key_part1 >= 1 AND key_part2 < 2
```

しかし、実際は条件が次に変換されます。

```
key_part1 >= 1 AND key_part2 IS NOT NULL
```

単一パートインデックスの範囲条件の間隔を結合または排除するために最適化を実行する方法の詳細は、[単一部品インデックスの範囲アクセス方法](#) を参照してください。マルチパートインデックスでの範囲条件にも類似の手順が実行されます。

複数值比較の等価範囲の最適化

`col_name` がインデックス設定されたカラムである次の式を考慮します。

```
col_name IN(val1, ..., valN)  
col_name = val1 OR ... OR col_name = valN
```

`col_name` が複数の値のいずれかと等しい場合に、各式は `true` になります。これらの比較は等価範囲比較です (ここで「範囲」は単一の値です)。最適化は、次のように等価範囲比較の対象とする行の読み取りのコストを推定します。

- `col_name` に一意のインデックスがある場合、指定した値を持つことができる行は多くても 1 つであるため、各範囲の行の見積もりは 1 です。
- それ以外の場合、`col_name` のインデックスは一意ではなく、最適化はインデックスまたはインデックス統計に分割して、各範囲の行数を見積もることができます。

インデックスダイブでは、最適化は範囲の両端でダイブを作成し、範囲内の行数を見積もりとして使用します。たとえば、式 `col_name IN (10, 20, 30)` には 3 つの等価範囲があり、最適化は範囲あたり 2 つのダイブを作成して、行の見積もりを生成します。ダイブのペアごとに、指定した値を持つ行数の見積もりを生成します。

インデックスダイブは、正確な行見積もりを提供しますが、式内の比較値の数が増えるほど、最適化の行見積もりの生成に時間がかかるようになります。インデックス統計の使用は、インデックスダイブより正確ではありませんが、大きな値リストの場合に、行見積もりが高速になります。

`eq_range_index_dive_limit` システム変数を使用して、最適化が行の見積もり戦略を別の戦略に切り替える値の数を構成できます。最大 `N` 個の等価範囲の比較にインデックスダイブの使用を許可するには、`eq_range_index_dive_limit` を `N + 1` に設定します。統計の使用を無効にし、`N` に関係なく常にインデックスダイブを使用するには、`eq_range_index_dive_limit` を 0 に設定します。

最適な推定を行うためにテーブルインデックス統計を更新するには、`ANALYZE TABLE` を使用します。

MySQL 8.0 より前では、`eq_range_index_dive_limit` システム変数を使用する場合を除き、インデックス dives の使用をスキップしてインデックスの有用性を見積もる方法はありません。MySQL 8.0 では、次のすべての条件を満たすクエリでインデックス分割スキップが可能です:

- クエリは、複数のテーブルに対する結合ではなく、単一のテーブルに対するものです。
- 単一インデックスの `FORCE INDEX` インデックスヒントが存在します。インデックスの使用が強制される場合、インデックスへの分割を実行する追加のオーバーヘッドからは何も取得されません。
- インデックスは一意ではなく、`FULLTEXT` インデックスではありません。
- サブクエリが存在しません。
- `DISTINCT`、`GROUP BY` または `ORDER BY` 句が存在しません。

`EXPLAIN FOR CONNECTION` の場合、`index dives` がスキップされると、出力は次のように変更されます:

- 従来の出力では、`rows` および `filtered` の値は `NULL` です。
- JSON 出力の場合、`rows_examined_per_scan` および `rows_produced_per_join` は表示されず、`skip_index_dive_due_to_force` は `true` で、コスト計算は正確ではありません。

`FOR CONNECTION` を使用しない場合、インデックス dives がスキップされても、`EXPLAIN` 出力は変更されません。

インデックス分割がスキップされるクエリーの実行後、`INFORMATION_SCHEMA.OPTIMIZER_TRACE` テーブルの対応する行に `skipped_due_to_force_index` の `index_dives_for_range_access` 値が含まれます。

スキャン範囲アクセス方法のスキップ

次のシナリオを考えてみます:

```
CREATE TABLE t1 (f1 INT NOT NULL, f2 INT NOT NULL, PRIMARY KEY(f1, f2));
INSERT INTO t1 VALUES
  (1,1), (1,2), (1,3), (1,4), (1,5),
  (2,1), (2,2), (2,3), (2,4), (2,5);
INSERT INTO t1 SELECT f1, f2 + 5 FROM t1;
INSERT INTO t1 SELECT f1, f2 + 10 FROM t1;
INSERT INTO t1 SELECT f1, f2 + 20 FROM t1;
INSERT INTO t1 SELECT f1, f2 + 40 FROM t1;
ANALYZE TABLE t1;

EXPLAIN SELECT f1, f2 FROM t1 WHERE f2 > 40;
```

このクエリーを実行するために、MySQL では、インデックススキャンを選択してすべての行をフェッチし (インデックスには選択するすべてのカラムが含まれます)、`WHERE` 句から `f2 > 40` 条件を適用して最終結果セットを生成できます。

レンジスキャンは全インデックススキャンよりも効率的ですが、最初のインデックスカラムである `f1` に条件がないため、この場合は使用できません。ただし、MySQL 8.0.13 の時点では、オプティマイザは、ループインデックススキャンと同様のスキップスキャンという方法を使用して、`f1` の値ごとに複数のレンジスキャンを実行できます ([セクション 8.2.1.17 「GROUP BY の最適化」](#) を参照):

1. 最初のインデックス部分、`f1` (インデックス接頭辞) の個別値間でスキップします。
2. 残りのインデックス部分で、`f2 > 40` 条件の個別の接頭辞値ごとにサブレンジスキャンを実行します。

前述のデータセットの場合、アルゴリズムは次のように動作します:

1. 最初のキー部分 (`f1 = 1`) の最初の個別値を取得します。
2. 最初と 2 番目のキー部分 (`f1 = 1 AND f2 > 40`) に基づいて範囲を構築します。
3. 範囲スキャンを実行します。
4. 最初のキー部分の次の個別値を取得します (`f1 = 2`)。
5. 最初と 2 番目のキー部分 (`f1 = 2 AND f2 > 40`) に基づいて範囲を構築します。
6. 範囲スキャンを実行します。

この方法を使用すると、構成された範囲ごとに適格でない行が MySQL によってスキップされるため、アクセスされる行の数が減ります。このスキップスキャンアクセス方法は、次の条件下で適用できます:

- 表 T には、フォーム (`[A_1, ..., A_k,] B_1, ..., B_m, C [, D_1, ..., D_n]`) のキー部分を持つ複合インデックスが少なくとも 1 つあります。キー部分 A および D は空でもかまいませんが、B および C は空でない必要があります。
- クエリーは 1 つのテーブルだけを参照します。
- クエリーでは、`GROUP BY` または `DISTINCT` は使用されません。
- クエリーは、インデックス内のカラムのみを参照します。
- `A_1, ..., A_k` の述語は等価述語であり、定数である必要があります。これには、`IN()` 演算子が含まれます。

- クエリーは結合クエリー (OR 条件の AND) である必要があります: (cond1(key_part1) OR cond2(key_part1)) AND (cond1(key_part2) OR ...) AND ...
- C には範囲条件が必要です。
- D カラムに対する条件は許可されています。D の条件は、C の範囲条件と組み合わせる必要があります。

スキップスキンの使用は、EXPLAIN 出力で次のように示されます:

- Extra カラムの Using index for skip scan は、ルーズインデックススキップスキンアクセス方法が使用されていることを示します。
- インデックスをスキップスキンに使用できる場合は、インデックスが possible_keys カラムに表示されます。

スキップスキンの使用は、オプティマイザトレース出力で次の形式の"skip scan"要素によって示されます:

```
"skip_scan_range": {
  "type": "skip_scan",
  "index": index_used_for_skip_scan,
  "key_parts_used_for_access": [key_parts_used_for_access],
  "range": [range]
}
```

"best_skip_scan_summary"要素も表示される場合があります。最適なレンジアクセスバリエーションとしてスキップのスキップを選択すると、"chosen_range_access_summary"が書き込まれます。全体的な最適なアクセス方法としてスキップのスキップが選択されている場合は、"best_access_path"要素が存在します。

スキップスキンの使用は、optimizer_switch システム変数の skip_scan フラグの値の影響を受けます。セクション8.9.2「切り替え可能な最適化」を参照してください。デフォルトでは、このフラグは on です。無効にするには、skip_scan を off に設定します。

optimizer_switch システム変数を使用してスキップスキンセッション全体のオプティマイザの使用を制御することに加えて、MySQL ではオプティマイザヒントをサポートしてステートメントごとにオプティマイザに影響を与えます。セクション8.9.3「オプティマイザヒント」を参照してください。

行コンストラクタ式の範囲最適化

オプティマイザは、レンジスキンアクセス方法を次の形式のクエリーに適用できます:

```
SELECT ... FROM t1 WHERE ( col_1, col_2 ) IN (('a', 'b'), ('c', 'd'));
```

以前は、レンジスキンを使用するには、次のようにクエリーを記述する必要がありました:

```
SELECT ... FROM t1 WHERE ( col_1 = 'a' AND col_2 = 'b' )
OR ( col_1 = 'c' AND col_2 = 'd' );
```

オプティマイザでレンジスキンを使用するには、クエリーが次の条件を満たす必要があります:

- IN() 述語のみが使用され、NOT IN() は使用されません。
- IN() 述語の左側では、行コンストラクタにはカラム参照のみが含まれます。
- IN() 述語の右側にある行コンストラクタには、実行中に定数にバインドされるリテラルまたはローカルカラム参照であるランタイム定数のみが含まれます。
- IN() 述語の右側には、複数の行コンストラクタがあります。

オプティマイザおよび行コンストラクタの詳細は、セクション8.2.1.22「行コンストラクタ式の最適化」を参照してください

範囲最適化のためのメモリー使用の制限

範囲オプティマイザで使用可能なメモリーを制御するには、range_optimizer_max_mem_size システム変数を使用します:

- 値 0 は「制限なし」を表します。

- 0 より大きい値を指定すると、オプティマイザは範囲アクセス方法を考慮する際に消費されるメモリーを追跡します。指定した制限を超えると、レンジアクセス方法が破棄され、全テーブルスキャンなどの他の方法がかわりに考慮されます。これは最適ではない可能性があります。これが発生すると、次の警告が発生します (N は現在の `range_optimizer_max_mem_size` 値です):

```
Warning 3170 Memory capacity of N bytes for
'range_optimizer_max_mem_size' exceeded. Range
optimization was not done for this query.
```

- UPDATE** および **DELETE** ステートメントでは、オプティマイザが全テーブルスキャンにフォールバックし、`sql_safe_updates` システム変数が有効になっている場合、変更する行の決定にキーが使用されないため、警告ではなくエラーが発生します。詳細は、[セーフ更新モードの使用 \(--safe-updates\)](#)を参照してください。

使用可能な範囲最適化メモリーを超え、オプティマイザがより最適でない計画にフォールバックする個々のクエリーの場合、`range_optimizer_max_mem_size` 値を増やすとパフォーマンスが向上する可能性があります。

範囲式の処理に必要なメモリー量を見積もるには、次のガイドラインを使用します:

- 範囲アクセス方法の候補キーが 1 つある次のような単純なクエリーの場合、**OR** と組み合わされた各述語では約 230 バイトが使用されます:

```
SELECT COUNT(*) FROM t
WHERE a=1 OR a=2 OR a=3 OR ... a=N;
```

- 同様に、次のようなクエリーでは、**AND** と組み合わされた各述語で約 125 バイトが使用されます:

```
SELECT COUNT(*) FROM t
WHERE a=1 AND b=1 AND c=1 ... N;
```

- IN()** 述語を含むクエリーの場合:

```
SELECT COUNT(*) FROM t
WHERE a IN (1,2, ..., M) AND b IN (1,2, ..., N);
```

IN() リストの各リテラル値は、**OR** と組み合わされた述語としてカウントされます。2 つの **IN()** リストがある場合、**OR** と組み合わせた述語の数は、各リストのリテラル値の数になります。したがって、前述の例で **OR** と組み合わされている述語の数は、 $M \times N$ です。

8.2.1.3 インデックスマージの最適化

インデックスマージアクセス方法では、複数の `range` スキャンを含む行が取得され、その結果が 1 つにマージされます。このアクセス方法では、単一のテーブルからのみインデックススキャンがマージされ、複数のテーブルにわたるスキャンはマージされません。このマージによって、その基盤となるスキャンの和集合、共通集合、または共通集合の和集合を生成できます。

インデックスマージを使用できるクエリーの例:

```
SELECT * FROM tbl_name WHERE key1 = 10 OR key2 = 20;

SELECT * FROM tbl_name
WHERE (key1 = 10 OR key2 = 20) AND non_key = 30;

SELECT * FROM t1, t2
WHERE (t1.key1 IN (1,2) OR t1.key2 LIKE 'value%')
AND t2.key1 = t1.some_col;

SELECT * FROM t1, t2
WHERE t1.key1 = 1
AND (t2.key1 = t1.some_col OR t2.key2 = t1.some_col2);
```

注記

インデックスマージ最適化アルゴリズムには、次の既知の制限事項があります:

- クエリーに深い **AND/OR** ネストを含む複雑な **WHERE** 句があり、MySQL が最適な計画を選択しない場合は、次のアイデンティティ変換を使用して用語を配布してみてください:

```
(x AND y) OR z => (x OR z) AND (y OR z)
(x OR y) AND z => (x AND z) OR (y AND z)
```

- インデックスマージは全文インデックスには適用できません。

EXPLAIN 出力では、インデックスマージメソッドは `type` カラムに `index_merge` と表示されます。この場合、`key` カラムには使用されたインデックスのリストが含まれ、`key_len` にはそれらのインデックスの最長のキーパートのリストが含まれます。

インデックスマージアクセス方法には、**EXPLAIN** 出力の `Extra` フィールドに表示されるいくつかのアルゴリズムがあります:

- [Using intersect\(...\)](#)
- [Using union\(...\)](#)
- [Using sort_union\(...\)](#)

次の各セクションでは、これらのアルゴリズムについて詳しく説明します。オプティマイザは、使用可能な様々なオプションのコスト見積りに基づいて、様々なインデックスマージアルゴリズムとその他のアクセス方法のいずれかを選択します。

- [インデックスマージ交差アクセスアルゴリズム](#)
- [インデックスマージ結合アクセスアルゴリズム](#)
- [インデックスマージソート - ユニオンアクセスアルゴリズム](#)
- [インデックスマージ最適化への影響](#)

インデックスマージ交差アクセスアルゴリズム

このアクセスアルゴリズムは、**WHERE** 句が **AND** と組み合わされた異なるキーの複数の範囲条件に変換され、各条件が次のいずれかである場合に適用されます:

- この形式の `N` 部分式。インデックスには正確に `N` 部分が含まれます (つまり、すべてのインデックス部分が対象となります):

```
key_part1 = const1 AND key_part2 = const2 ... AND key_partN = constN
```

- InnoDB テーブルの主キーに対する範囲条件。

例:

```
SELECT * FROM innodb_table
WHERE primary_key < 10 AND key_col1 = 20;

SELECT * FROM tbl_name
WHERE key1_part1 = 1 AND key1_part2 = 2 AND key2 = 2;
```

インデックスマージ共通集合アルゴリズムは、使用されたすべてのインデックスの同時スキャンを実行し、マージされたインデックススキャンから受け取る行シーケンスの共通集合を生成します。

クエリーに使用されているすべてのカラムが、使用されるインデックスによってカバーされている場合、完全なテーブル行は取得されません (この場合、**EXPLAIN** 出力の `Extra` フィールドに `Using index` が含まれます)。次はそのようなクエリーの例です。

```
SELECT COUNT(*) FROM t1 WHERE key1 = 1 AND key2 = 1;
```

使用されているインデックスがクエリーで使用されているすべてのカラムをカバーしているわけではない場合、使用されているすべてのキーの範囲条件が満たされている場合にのみ、行全体が取得されます。

マージされた条件のいずれかが InnoDB テーブルの主キーに対する条件である場合、行の取得には使用されませんが、他の条件を使用して取得された行を除外するために使用されます。

インデックスマージ結合アクセスアルゴリズム

このアルゴリズムの基準は、インデックスマージ交差アルゴリズムの基準と似ています。このアルゴリズムは、テーブルの `WHERE` 句が `OR` と組み合わされた異なるキーの複数の範囲条件に変換され、各条件が次のいずれかである場合に適用されます:

- この形式の `N` 部分式。インデックスには正確に `N` 部分が含まれます (つまり、すべてのインデックス部分が対象となります):

```
key_part1 = const1 AND key_part2 = const2 ... AND key_partN = constN
```

- InnoDB テーブルの主キーに対する範囲条件。
- インデックスマージ交差アルゴリズムが適用可能な条件。

例:

```
SELECT * FROM t1
WHERE key1 = 1 OR key2 = 2 OR key3 = 3;

SELECT * FROM innodb_table
WHERE (key1 = 1 AND key2 = 2)
OR (key3 = 'foo' AND key4 = 'bar') AND key5 = 5;
```

インデックスマージソート - ユニオンアクセスアルゴリズム

このアクセスアルゴリズムは、`WHERE` 句が `OR` で結合された複数の範囲条件に変換されるが、インデックスマージ結合アルゴリズムが適用できない場合に適用できます。

例:

```
SELECT * FROM tbl_name
WHERE key_col1 < 10 OR key_col2 < 20;

SELECT * FROM tbl_name
WHERE (key_col1 > 10 OR key_col2 = 20) AND nonkey_col = 30;
```

ソートと集合アルゴリズムと和集合アルゴリズムの違いは、ソートと集合アルゴリズムでは、行を返す前にまずすべての行の行 ID をフェッチし、それらをソートする必要があることです。

インデックスマージ最適化への影響

インデックスマージの使用は、`optimizer_switch` システム変数の `index_merge`, `index_merge_intersection`, `index_merge_union` および `index_merge_sort_union` フラグの値の影響を受けます。セクション8.9.2「切り替え可能な最適化」を参照してください。デフォルトでは、これらのフラグはすべて `on` です。特定のアルゴリズムのみを有効にするには、`index_merge` を `off` に設定し、許可される他のアルゴリズムのみを有効にします。

MySQL では、`optimizer_switch` システム変数を使用してインデックスマージアルゴリズムのセッション全体のオプティマイザ使用を制御することに加えて、オプティマイザヒントをサポートしてステートメントごとにオプティマイザに影響を与えます。セクション8.9.3「オプティマイザヒント」を参照してください。

8.2.1.4 ハッシュ結合の最適化

MySQL 8.0.18 以降、MySQL では、各結合に等価結合条件があり、次のような結合条件に適用できるインデックスがないクエリーに対してハッシュ結合が使用されます:

```
SELECT *
FROM t1
JOIN t2
ON t1.c1=t2.c1;
```

ハッシュ結合は、単一テーブルの述語に使用できるインデックスが 1 つ以上ある場合にも使用できます。

ハッシュ結合は通常、以前のバージョンの MySQL で採用されていたブロックネストループアルゴリズム ([Block Nested Loop 結合アルゴリズム](#) を参照) のかわりに、このような場合に使用することを目的としています。MySQL

8.0.20 以降では、ブロックネストループのサポートが削除され、以前にブロックネステッドループが使用されていた場所では、サーバーはハッシュ結合を採用します。

前述の例およびこのセクションの残りの例では、次のステートメントを使用して **t1**、**t2** および **t3** の 3 つのテーブルが作成されていることを前提としています：

```
CREATE TABLE t1 (c1 INT, c2 INT);
CREATE TABLE t2 (c1 INT, c2 INT);
CREATE TABLE t3 (c1 INT, c2 INT);
```

ハッシュ結合が採用されていることは、次のように **EXPLAIN** を使用して確認できます：

```
mysql> EXPLAIN
-> SELECT * FROM t1
-> JOIN t2 ON t1.c1=t2.c1\G
***** 1. row *****
  id: 1
select_type: SIMPLE
  table: t1
  partitions: NULL
  type: ALL
possible_keys: NULL
  key: NULL
  key_len: NULL
  ref: NULL
  rows: 1
  filtered: 100.00
  Extra: NULL
***** 2. row *****
  id: 1
select_type: SIMPLE
  table: t2
  partitions: NULL
  type: ALL
possible_keys: NULL
  key: NULL
  key_len: NULL
  ref: NULL
  rows: 1
  filtered: 100.00
  Extra: Using where; Using join buffer (hash join)
```

(MySQL 8.0.20 より前では、ハッシュ結合が特定の結合に使用されていたかどうかを確認するには、**FORMAT=TREE** オプションを含める必要がありました。)

EXPLAIN ANALYZE には、使用されているハッシュ結合に関する情報も表示されます。

ハッシュ結合は、次に示すように、テーブルのペアごとに少なくとも 1 つの結合条件が等価結合であるかぎり、複数の結合を含むクエリーにも使用されます：

```
SELECT * FROM t1
JOIN t2 ON (t1.c1 = t2.c1 AND t1.c2 < t2.c2)
JOIN t3 ON (t2.c1 = t3.c1);
```

内部結合を使用する前述のような場合、等価結合ではない追加の条件は、結合の実行後にフィルタとして適用されます。(左結合、準結合、アンチ結合などの外部結合の場合は、結合の一部として出力されます。)これは、**EXPLAIN** の出力に表示されます：

```
mysql> EXPLAIN FORMAT=TREE
-> SELECT *
-> FROM t1
-> JOIN t2
-> ON (t1.c1 = t2.c1 AND t1.c2 < t2.c2)
-> JOIN t3
-> ON (t2.c1 = t3.c1)\G
***** 1. row *****
EXPLAIN: -> Inner hash join (t3.c1 = t1.c1) (cost=1.05 rows=1)
-> Table scan on t3 (cost=0.35 rows=1)
-> Hash
-> Filter: (t1.c2 < t2.c2) (cost=0.70 rows=1)
-> Inner hash join (t2.c1 = t1.c1) (cost=0.70 rows=1)
```



```
-> Table scan on t2 (cost=0.35 rows=1)
-> Hash
-> Table scan on t1 (cost=0.35 rows=1)
```

前述の出力からもわかるように、複数の等価結合条件を持つ結合には複数のハッシュ結合を使用できます。

MySQL 8.0.20 より前は、結合テーブルのいずれかのペアに等価結合条件がなく、より遅いブロックのネステッドループアルゴリズムが使用されていた場合、ハッシュ結合は使用できませんでした。MySQL 8.0.20 以降では、次に示すように、ハッシュ結合が使用されます：

```
mysql> EXPLAIN FORMAT=TREE
-> SELECT * FROM t1
-> JOIN t2 ON (t1.c1 = t2.c1)
-> JOIN t3 ON (t2.c1 < t3.c1)\G
***** 1. row *****
EXPLAIN: -> Filter: (t1.c1 < t3.c1) (cost=1.05 rows=1)
-> Inner hash join (no condition) (cost=1.05 rows=1)
-> Table scan on t3 (cost=0.35 rows=1)
-> Hash
-> Inner hash join (t2.c1 = t1.c1) (cost=0.70 rows=1)
-> Table scan on t2 (cost=0.35 rows=1)
-> Hash
-> Table scan on t1 (cost=0.35 rows=1)
```

(追加の例は、このセクションの後半で説明します。)

ハッシュ結合はデカルト積にも適用されます。つまり、結合条件が指定されていない場合は、次のようになります：

```
mysql> EXPLAIN FORMAT=TREE
-> SELECT *
-> FROM t1
-> JOIN t2
-> WHERE t1.c2 > 50\G
***** 1. row *****
EXPLAIN: -> Inner hash join (cost=0.70 rows=1)
-> Table scan on t2 (cost=0.35 rows=1)
-> Hash
-> Filter: (t1.c2 > 50) (cost=0.35 rows=1)
-> Table scan on t1 (cost=0.35 rows=1)
```

MySQL 8.0.20 以降では、ハッシュ結合を使用するために、結合に少なくとも 1 つの等価結合条件を含める必要がなくなりました。つまり、ハッシュ結合を使用して最適化できるクエリーのタイプには、次のリスト (および例) のクエリーが含まれます：

- 内部非等価結合:

```
mysql> EXPLAIN FORMAT=TREE SELECT * FROM t1 JOIN t2 ON t1.c1 < t2.c1\G
***** 1. row *****
EXPLAIN: -> Filter: (t1.c1 < t2.c1) (cost=4.70 rows=12)
-> Inner hash join (no condition) (cost=4.70 rows=12)
-> Table scan on t2 (cost=0.08 rows=6)
-> Hash
-> Table scan on t1 (cost=0.85 rows=6)
```

- 準結合:

```
mysql> EXPLAIN FORMAT=TREE SELECT * FROM t1
-> WHERE t1.c1 IN (SELECT t2.c2 FROM t2)\G
***** 1. row *****
EXPLAIN: -> Nested loop inner join
-> Filter: (t1.c1 is not null) (cost=0.85 rows=6)
-> Table scan on t1 (cost=0.85 rows=6)
-> Single-row index lookup on <subquery2> using <auto_distinct_key> (c2=t1.c1)
-> Materialize with deduplication
-> Filter: (t2.c2 is not null) (cost=0.85 rows=6)
-> Table scan on t2 (cost=0.85 rows=6)
```

- アンチ結合:

```
mysql> EXPLAIN FORMAT=TREE SELECT * FROM t2
-> WHERE NOT EXISTS (SELECT * FROM t1 WHERE t1.col1 = t2.col1)\G
***** 1. row *****
```

```
EXPLAIN: -> Nested loop antijoin
-> Table scan on t2 (cost=0.85 rows=6)
-> Single-row index lookup on <subquery2> using <auto_distinct_key> (c1=t2.c1)
-> Materialize with deduplication
-> Filter: (t1.c1 is not null) (cost=0.85 rows=6)
-> Table scan on t1 (cost=0.85 rows=6)
```

- 左外部結合:

```
mysql> EXPLAIN FORMAT=TREE SELECT * FROM t1 LEFT JOIN t2 ON t1.c1 = t2.c1\G
***** 1. row *****
EXPLAIN: -> Left hash join (t2.c1 = t1.c1) (cost=3.99 rows=36)
-> Table scan on t1 (cost=0.85 rows=6)
-> Hash
-> Table scan on t2 (cost=0.14 rows=6)
```

- 右外部結合 (MySQL では、すべての右外部結合が左外部結合としてリライトされることを確認します):

```
mysql> EXPLAIN FORMAT=TREE SELECT * FROM t1 RIGHT JOIN t2 ON t1.c1 = t2.c1\G
***** 1. row *****
EXPLAIN: -> Left hash join (t1.c1 = t2.c1) (cost=3.99 rows=36)
-> Table scan on t2 (cost=0.85 rows=6)
-> Hash
-> Table scan on t1 (cost=0.14 rows=6)
```

デフォルトでは、MySQL 8.0.18 以降では可能な限りハッシュ結合が使用されます。ハッシュ結合を使用するかどうかは、[BNL](#) オプティマイザヒントと [NO_BNL](#) オプティマイザヒントのいずれかを使用して制御できます。

(MySQL 8.0.18 は、[optimizer_switch](#) サーバースステム変数の設定の一部として [hash_join=on](#) または [hash_join=off](#) をサポートし、オプティマイザヒント [HASH_JOIN](#) または [NO_HASH_JOIN](#) もサポートしていました。MySQL 8.0.19 以降では、これらは効果がなくなりました。)

ハッシュ結合によるメモリー使用量は、[join_buffer_size](#) システム変数を使用して制御できます。ハッシュ結合では、この量を超えるメモリーを使用できません。ハッシュ結合に必要なメモリーが使用可能な量を超えると、MySQL はディスク上のファイルを使用してこれを処理します。これが発生した場合、ハッシュ結合がメモリーに収まらず、[open_files_limit](#) に設定されているよりも多くのファイルが作成されると、結合が成功しない可能性があることに注意してください。このような問題を回避するには、次のいずれかの変更を行います:

- ハッシュ結合がディスクにオーバーフローしないように、[join_buffer_size](#) を増やします。
- [open_files_limit](#) を増やします。

MySQL 8.0.18 以降、ハッシュ結合の結合バッファは増分的に割り当てられるため、非常に大量の RAM を割り当てる小さいクエリーなしに [join_buffer_size](#) をより高く設定できますが、外部結合ではバッファ全体が割り当てられます。MySQL 8.0.20 以降では、外部結合 (アンチ結合および準結合を含む) にもハッシュ結合が使用されるため、これは問題ではなくなりました。

8.2.1.5 エンジンコンディションプッシュダウンの最適化

この最適化は、インデックスが設定されていないカラムと定数との直接比較の効率性を向上します。このような場合、条件が評価のためにストレージエンジンに「プッシュダウン」されます。この最適化は、[NDB](#) ストレージエンジンでのみ使用できます。

NDB Cluster の場合、この最適化により、クラスタデータノードとクエリーを発行した MySQL サーバーの間で一致しない行をネットワーク経由で送信する必要がなくなり、条件プッシュダウンが可能だが使用されない場合に 5 から 10 倍の係数で使用されるクエリーを高速化できます。

「NDB Cluster」テーブルが次のように定義されているとします:

```
CREATE TABLE t1 (
  a INT,
  b INT,
  KEY(a)
) ENGINE=NDB;
```

エンジン条件プッシュダウンは、インデックス付けされていないカラムと定数の比較を含む、次に示すようなクエリーで使用できます:

```
SELECT a, b FROM t1 WHERE b = 10;
```

エンジン条件プッシュダウンの使用は、[EXPLAIN](#) の出力で確認できます:

```
mysql> EXPLAIN SELECT a,b FROM t1 WHERE b = 10\G
***** 1. row *****
   id: 1
 select_type: SIMPLE
   table: t1
   type: ALL
possible_keys: NULL
   key: NULL
  key_len: NULL
   ref: NULL
  rows: 10
 Extra: Using where with pushed condition
```

ただし、エンジン条件プッシュダウンは、次のクエリーでは使用できません:

```
SELECT a,b FROM t1 WHERE a = 10;
```

カラム `a` にインデックスが存在するため、エンジン条件プッシュダウンはここでは適用できません。(インデックスアクセスメソッドの方が効率的であるため、コンディションプッシュダウンよりも優先して選択されます。)

エンジン条件プッシュダウンは、インデックス付けされたカラムが `>` または `<` 演算子を使用して定数と比較される場合にも使用できます:

```
mysql> EXPLAIN SELECT a, b FROM t1 WHERE a < 2\G
***** 1. row *****
   id: 1
 select_type: SIMPLE
   table: t1
   type: range
possible_keys: a
   key: a
  key_len: 5
   ref: NULL
  rows: 2
 Extra: Using where with pushed condition
```

エンジン条件プッシュダウンでサポートされているその他の比較には、次のものがあります:

- `column [NOT] LIKE pattern`

`pattern` は、照合するパターンを含む文字列リテラルである必要があります。構文については、[セクション 12.8.1「文字列比較関数および演算子」](#)を参照してください。

- `column IS [NOT] NULL`
- `column IN (value_list)`

`value_list` の各項目は定数のリテラル値である必要があります。

- `column BETWEEN constant1 AND constant2`

`constant1` と `constant2` はそれぞれ、定数のリテラル値である必要があります。

前のリストのすべての場合で、条件をカラムと定数との 1 つ以上の直接比較の形式に変換できます。

エンジンコンディションプッシュダウンはデフォルトで有効です。サーバーの起動時に無効にするには、`optimizer_switch` システム変数の `engine_condition_pushdown` フラグを `off` に設定します。たとえば、`my.cnf` ファイルで、次の行を使用します。

```
[mysqld]
optimizer_switch=engine_condition_pushdown=off
```

実行時に、次のように条件プッシュダウンを無効にします:

```
SET optimizer_switch='engine_condition_pushdown=off';
```

制限。 エンジンコンディションプッシュダウンには次の制限があります。

- エンジン条件プッシュダウンは、**NDB** ストレージエンジンでのみサポートされます。
- NDB 8.0.18 より前では、カラムは定数または定数値にのみ評価される式と比較できました。NDB 8.0.18 以降では、カラムがまったく同じタイプであるかぎり、カラムを相互に比較できます (それらが該当する場合は、同じ符号性、長さ、文字セット、精度、およびスケールを含む)。
- 比較に使用されるカラムは、**BLOB** 型または **TEXT** 型のいずれかであってははいけません。この除外は、**JSON**、**BIT** および **ENUM** カラムにも拡張されます。
- カラムと比較される文字列値は、カラムと同じ照合順序を使用する必要があります。
- 結合は直接サポートされていません。複数のテーブルを含む条件は、可能な場合に個別にプッシュされます。拡張 **EXPLAIN** 出力を使用して、実際にプッシュダウンされる条件を決定します。[セクション8.8.3「拡張 EXPLAIN 出力形式」](#)を参照してください。

以前は、エンジン条件プッシュダウンは、条件のプッシュ先と同じテーブルのカラム値を参照する用語に制限されてきました。NDB 8.0.16 以降では、クエリープラン内の以前のテーブルのカラム値をプッシュされた条件から参照することもできます。これにより、結合処理中に SQL ノードで処理する必要がある行数が削減されます。フィルタリングは、単一の **mysqld** プロセスではなく LDM スレッドで並列に実行することもできます。これにより、クエリーのパフォーマンスが大幅に向上する可能性があります。

NDB 8.0.20 以降では、スキャンを使用する外部結合は、同じ結合入れ子で使用されるテーブル、またはそれが依存する結合 **nmests** 内のテーブルにプッシュ不可能な条件がない場合にプッシュできます。これは、使用される最適化戦略が **firstMatch** である場合には、準結合でも当てはまります ([セクション8.2.2.1「準結合変換による IN および EXISTS サブクエリー述語の最適化」](#)を参照)。

次の 2 つの状況では、結合アルゴリズムを前のテーブルの参照カラムと組み合わせることはできません:

1. 参照された以前のテーブルのいずれかが結合バッファ内にある場合。この場合、スキャンフィルタ処理されたテーブルから取得された各行は、バッファ内のすべての行と照合されます。つまり、スキャンフィルタの生成時にカラム値をフェッチできる単一の特定の行はありません。
2. プッシュされた結合の子操作からカラムが作成された場合。これは、スキャンフィルタの生成時に、結合の祖先操作から参照される行がまだ取得されていないためです。

8.2.1.6 インデックスコンディションプッシュダウンの最適化

インデックスコンディションプッシュダウン (ICP) は、MySQL がインデックスを使用してテーブルから行を取得する場合の最適化です。ICP を使用しない場合、ストレージエンジンはインデックスをトラバースして、ベーステーブル内で行を検索し、MySQL Server に返し、MySQL Server が行に対して **WHERE** 条件を評価します。ICP が有効で、インデックスのカラムのみを使用して **WHERE** 条件の一部を評価できる場合、MySQL サーバーは **WHERE** 条件のこの部分をストレージエンジンにプッシュダウンします。ストレージエンジンは、インデックスエントリを使用して、プッシュされたインデックス条件を評価し、これが満たされている場合にのみ、テーブルから行を読み取ります。ICP は、ストレージエンジンがベーステーブルにアクセスする必要がある回数と、MySQL サーバーがストレージエンジンにアクセスする必要がある回数を削減できます。

インデックス条件プッシュダウン最適化の適用には、次の条件があります:

- ICP は、完全なテーブルの行にアクセスする必要がある場合に、**range**、**ref**、**eq_ref** および **ref_or_null** のアクセス方法に使用されます。
- ICP は、パーティション化された **InnoDB** テーブルおよび **MyISAM** テーブルを含む **InnoDB** テーブルおよび **MyISAM** テーブルに使用できます。
- **InnoDB** テーブルの場合、ICP はセカンダリインデックスにのみ使用されます。ICP の目的は、全行読取りの数を減らして I/O 操作を減らすことです。**InnoDB** のクラスタ化されたインデックスの場合、完全なレコードはすでに **InnoDB** バッファに読み込まれています。この場合、ICP を使用しても I/O は削減されません。
- ICP は、仮想生成カラムに作成されたセカンダリインデックスではサポートされていません。**InnoDB** では、仮想生成カラムのセカンダリインデックスがサポートされます。

- サブクエリーを参照する条件はプッシュダウンできません。
- ストアドファンクションを参照する条件はプッシュダウンできません。ストレージエンジンはストアドファンクションを呼び出せません。
- トリガー条件はプッシュダウンできません。(トリガーされる条件の詳細は、[セクション8.2.2.3「EXISTS 戦略を使用したサブクエリーの最適化」](#)を参照してください。)

この最適化の仕組みを理解するには、最初に、インデックス条件プッシュダウンが使用されない場合のインデックススキャンの進行方法を検討します:

1. まず、インデックスタプルを読み取り、次にそのインデックスタプルを使用して、完全なテーブル行を見つけて読み取ることで、次の行を取得します。
2. このテーブルに適用される **WHERE** 条件の部分をテストします。テスト結果に基づいて行を受け入れるか、拒否します。

インデックス条件プッシュダウンを使用すると、かわりに次のようにスキャンが続行されます:

1. 次の行のインデックスタプルを取得します (ただし完全なテーブル行ではありません)。
2. このテーブルに適用され、インデックスカラムのみを使用してチェックできる **WHERE** 条件の部分をテストします。条件が満たされている場合、次の行のインデックスタプルに進みます。
3. 条件が満たされている場合、インデックスタプルを使用して、完全なテーブル行を見つけて読み取ります。
4. このテーブルに適用される **WHERE** 条件の残りの部分をテストします。テスト結果に基づいて行を受け入れるか、拒否します。

インデックス条件プッシュダウンが使用されている場合、**EXPLAIN** 出力の **Extra** カラムに **Using index condition** が表示されます。完全なテーブルの行を読み取る必要がある場合は適用されないため、**Using index** は表示されません。

テーブルに人とそのアドレスに関する情報が含まれており、テーブルに **INDEX (zipcode, lastname, firstname)** として定義されたインデックスがあるとします。個人の **zipcode** 値がわかっているが、姓がわからない場合は、次のように検索できます:

```
SELECT * FROM people
WHERE zipcode='95054'
AND lastname LIKE '%etrunia%'
AND address LIKE '%Main Street%';
```

MySQL はインデックスを使用して、**zipcode='95054'** を持つ人をスキャンします。2 番目の部分 (**lastname LIKE '%etrunia%'**) を使用してスキャンする必要がある行数を制限することはできないため、インデックス条件プッシュダウンを使用しない場合、このクエリーでは、**zipcode='95054'** を持つすべてのユーザーの完全なテーブルの行を取得する必要があります。

インデックス条件プッシュダウンでは、MySQL はテーブルの行全体を読み取る前に **lastname LIKE '%etrunia%'** 部分をチェックします。これにより、**zipcode** 条件に一致するが **lastname** 条件に一致しないインデックスタプルに対応する完全な行の読み取りが回避されます。

インデックス条件のプッシュダウンはデフォルトで有効になっています。 **index_condition_pushdown** フラグを設定することで、**optimizer_switch** システム変数で制御できます:

```
SET optimizer_switch = 'index_condition_pushdown=off';
SET optimizer_switch = 'index_condition_pushdown=on';
```

[セクション8.9.2「切り替え可能な最適化」](#)を参照してください。

8.2.1.7 Nested Loop 結合アルゴリズム

MySQL は、Nested Loop アルゴリズムまたはそのバリエーションを使用してテーブル間の結合を実行します。

- [Nested Loop 結合アルゴリズム](#)
- [Block Nested Loop 結合アルゴリズム](#)

Nested Loop 結合アルゴリズム

単純な Nested Loop Join (NLJ) アルゴリズムは、ループ内の最初のテーブルから行を一度に 1 つずつ読み取り、各行を、結合の次のテーブルを処理するネストしたループに渡します。このプロセスは、結合するテーブルが残っている回数だけ繰り返されます。

3 つのテーブル `t1`、`t2`、および `t3` 間の結合が、次の結合型を使用して実行されるとします。

Table	Join Type
<code>t1</code>	<code>range</code>
<code>t2</code>	<code>ref</code>
<code>t3</code>	<code>ALL</code>

単純な NLJ アルゴリズムを使用した場合、結合は次のように処理されます。

```
for each row in t1 matching range {
  for each row in t2 matching reference key {
    for each row in t3 {
      if row satisfies join conditions, send to client
    }
  }
}
```

NLJ アルゴリズムでは、外側のループから内側のループに、一度に 1 つずつ行を渡すため、一般に内側のループで処理されるテーブルを何回も読み取ります。

Block Nested Loop 結合アルゴリズム

Block Nested-Loop (BNL) 結合アルゴリズムは、外側のループで読み取られた行のバッファリングを使用して、内側のループでテーブルを読み取る必要がある回数が削減されます。たとえば、バッファに 10 行が読み込まれ、このバッファが次の内側のループに渡される場合、内側のループで読み取られる各行をバッファ内のすべての 10 行と比較できます。これにより、内部テーブルを読み取る必要がある回数が大幅に減少します。

MySQL 8.0.18 より前は、このアルゴリズムはインデックスを使用できなかった場合に等価結合に適用されていました。MySQL 8.0.18 以降では、このような場合にハッシュ結合の最適化が採用されます。MySQL 8.0.20 以降では、ブロックのネステッドループは MySQL で使用されなくなり、ブロックのネステッドループが以前に使用されていたすべての場合にハッシュ結合が使用されます。[セクション 8.2.1.4 「ハッシュ結合の最適化」](#) を参照してください。

MySQL 結合バッファリングには、次の特性があります：

- 結合バッファリングは、結合の型が `ALL` または `index` である (つまり、使用できるキーがなく、データ行またはインデックス行の完全スキャンがそれぞれ実行される場合) が、または `range` である場合に使用できます。[セクション 8.2.1.12 「Block Nested Loop 結合と Batched Key Access 結合」](#) で説明されているように、バッファリングの使用は外部結合にも適用できます。
- 結合バッファは、`ALL` または `index` タイプであっても、最初の非定数テーブルには割り当てられません。
- 結合に関連するカラムのみが、行全体ではなく結合バッファに格納されます。
- `join_buffer_size` システム変数は、クエリーの処理に使用される各結合バッファのサイズを決定します。
- バッファリング可能な結合ごとに 1 つのバッファが割り当てられるため、特定のクエリーが、複数の結合バッファを使用して処理されることがあります。
- 結合バッファは、結合の実行前に割り当てられ、クエリーの完了後に解放されます。

NLJ アルゴリズム (バッファリングなし) で前述した結合の例では、結合は結合バッファリングを使用して次のように実行されます：

```
for each row in t1 matching range {
  for each row in t2 matching reference key {
    store used columns from t1, t2 in join buffer
    if buffer is full {
      for each row in t3 {
        for each t1, t2 combination in join buffer {
```



```

    if row satisfies join conditions, send to client
  }
}
empty join buffer
}
}
}
}

if buffer is not empty {
  for each row in t3 {
    for each t1, t2 combination in join buffer {
      if row satisfies join conditions, send to client
    }
  }
}
}
}

```

S が格納されている各 $t1$ 、結合バッファ内の $t2$ の組合せのサイズであり、 C がバッファ内の組合せの数である場合、 $t3$ テーブルがスキャンされる回数は次のとおりです:

$$(S * C) / \text{join_buffer_size} + 1$$

`join_buffer_size` が前のすべての行の組み合わせを保持できるだけの大きさになる時点まで、`join_buffer_size` の値が大きくなるほど、 $t3$ スキャンの回数は減少します。その時点では、それを大きくしても速度は向上しません。

8.2.1.8 ネストした結合の最適化

結合を表す構文では、ネストした結合を使用できます。次の説明は、[セクション13.2.10.2「JOIN 句」](#)に説明する結合構文について言及しています。

`table_factor` の構文は SQL 標準と比較して拡張されています。後者は `table_reference` のみを受け付け、カッコ内のそれらのリストは受け付けません。これは、`table_reference` 項目のリストの各カンマを内部結合と同等とみなす場合、保守的な拡張です。例:

```
SELECT * FROM t1 LEFT JOIN (t2, t3, t4)
  ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

次と同等です:

```
SELECT * FROM t1 LEFT JOIN (t2 CROSS JOIN t3 CROSS JOIN t4)
  ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

MySQL では、`CROSS JOIN` は `INNER JOIN` と構文的に同等であり、相互に置換できます。標準 SQL では、それらは同等ではありません。`INNER JOIN` は `ON` 句と一緒に使用します。`CROSS JOIN` はそうでない場合でも使用できます。

一般に、内部結合操作のみを含む結合式内のカッコは無視できます。次の結合式について考えてみます:

```
t1 LEFT JOIN (t2 LEFT JOIN t3 ON t2.b=t3.b OR t2.b IS NULL)
  ON t1.a=t2.a
```

左のカッコおよびグループ化操作を削除すると、その結合式は次の式に変換されます:

```
(t1 LEFT JOIN t2 ON t1.a=t2.a) LEFT JOIN t3
  ON t2.b=t3.b OR t2.b IS NULL
```

まだ、2つの式は同等ではありません。これを確認するには、テーブル $t1$ 、 $t2$ 、 $t3$ が次の状態であるとします。

- テーブル $t1$ には行 (1)、(2) が含まれます
- テーブル $t2$ には行 (1,101) が含まれます
- テーブル $t3$ には行 (101) が含まれます

この場合、最初の式は行 (1,1,101,101)、(2,NULL,NULL,NULL) を含む結果セットを返し、2番目の式は行 (1,1,101,101)、(2,NULL,NULL,101) を返します。

```
mysql> SELECT *
```

```
FROM t1
LEFT JOIN
(t2 LEFT JOIN t3 ON t2.b=t3.b OR t2.b IS NULL)
ON t1.a=t2.a;
```

```
+-----+-----+-----+
| a | a | b | b |
+-----+-----+-----+
| 1 | 1 | 101 | 101 |
| 2 | NULL | NULL | NULL |
+-----+-----+-----+
```

```
mysql> SELECT *
FROM (t1 LEFT JOIN t2 ON t1.a=t2.a)
LEFT JOIN t3
ON t2.b=t3.b OR t2.b IS NULL;
```

```
+-----+-----+-----+
| a | a | b | b |
+-----+-----+-----+
| 1 | 1 | 101 | 101 |
| 2 | NULL | NULL | 101 |
+-----+-----+-----+
```

次の例では、外部結合操作が内部結合操作と一緒に使用されています。

```
t1 LEFT JOIN (t2, t3) ON t1.a=t2.a
```

その式は次の式に変換できません。

```
t1 LEFT JOIN t2 ON t1.a=t2.a, t3
```

指定されたテーブル状態では、次の 2 つの式は異なる行セットを返します。

```
mysql> SELECT *
FROM t1 LEFT JOIN (t2, t3) ON t1.a=t2.a;
```

```
+-----+-----+-----+
| a | a | b | b |
+-----+-----+-----+
| 1 | 1 | 101 | 101 |
| 2 | NULL | NULL | NULL |
+-----+-----+-----+
```

```
mysql> SELECT *
FROM t1 LEFT JOIN t2 ON t1.a=t2.a, t3;
```

```
+-----+-----+-----+
| a | a | b | b |
+-----+-----+-----+
| 1 | 1 | 101 | 101 |
| 2 | NULL | NULL | 101 |
+-----+-----+-----+
```

したがって、外部結合演算子を含む結合式のかっこを省略すると、元の式の結果セットが変わることがあります。

正確に言えば、左外部結合操作の右オペランドと右結合操作の左オペランドのかっこを無視することはできません。言い換えれば、外部結合操作の内部テーブル式のかっこを無視することはできません。ほかのオペランド (外部テーブルのオペランド) のかっこは無視できます。

次の式:

```
(t1,t2) LEFT JOIN t3 ON P(t2.b,t3.b)
```

属性 **t2.b** および **t3.b** に対する任意のテーブル **t1,t2,t3** および任意の条件 **P** のこの式と同等です:

```
t1, t2 LEFT JOIN t3 ON P(t2.b,t3.b)
```

結合式 (**joined_table**) での結合操作の実行順序が左から右にならない場合は、ネストされた結合について説明します。次のクエリーを考慮します。

```
SELECT * FROM t1 LEFT JOIN (t2 LEFT JOIN t3 ON t2.b=t3.b) ON t1.a=t2.a
WHERE t1.a > 1
```

```
SELECT * FROM t1 LEFT JOIN (t2, t3) ON t1.a=t2.a
```

```
WHERE (t2.b=t3.b OR t2.b IS NULL) AND t1.a > 1
```

それらのクエリーは次のネストした結合が含まれるとみなされます。

```
t2 LEFT JOIN t3 ON t2.b=t3.b
t2, t3
```

最初のクエリーでは、ネストされた結合は左結合操作で形成されます。2番目のクエリーでは、内部結合操作を使用して形成されます。

最初のクエリーでは、カッコを省略できます: 結合式の文法構造によって、結合操作の実行順序が同じになります。2番目のクエリーでは、カッコを省略できますが、それらがなくてもこの結合式は一義的に解釈できます。拡張構文では、理論的にはクエリーは解析されますが、2番目のクエリーの `(t2, t3)` にカッコが必要です: `LEFT JOIN` および `ON` は式 `(t2,t3)` の左右のデリミタの役割を果たすため、クエリーの構文構造は明確なままです。

前の例でこれらの点を説明します。

- 内部結合のみを含む (外部結合を含まない) 結合式の場合は、カッコを削除して、左から右に結合を評価できます。実際には、テーブルは任意の順序で評価できます。
- 一般に、外部結合、または内部結合と混在した外部結合の場合には、同じことが当てはまりません。カッコの削除によって、結果が変わることがあります。

ネストした外部結合を含むクエリーは内部結合を含むクエリーと同じパイプライン方式で実行されます。正確には、Nested Loop 結合アルゴリズムのバリエーションが利用されます。ネステッドループ結合でクエリーを実行するアルゴリズムを思い出します ([セクション8.2.1.7「Nested Loop 結合アルゴリズム」](#)を参照)。3つのテーブルに対する結合クエリー `T1,T2,T3` が次の形式であるとします:

```
SELECT * FROM T1 INNER JOIN T2 ON P1(T1,T2)
      INNER JOIN T3 ON P2(T2,T3)
WHERE P(T1,T2,T3)
```

ここでは、`P1(T1,T2)` と `P2(T2,T3)` が何らかの結合条件 (式での) で、`P(T1,T2,T3)` はテーブル `T1,T2,T3` のカラムに対する条件です。

Nested Loop 結合アルゴリズムでは、このクエリーを次のように実行します。

```
FOR each row t1 in T1 {
  FOR each row t2 in T2 such that P1(t1,t2) {
    FOR each row t3 in T3 such that P2(t2,t3) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
    }
  }
}
```

`t1||t2||t3` という表記法は、`t1`、`t2` および `t3` の行のカラムを連結して構築された行を示します。次の例の一部では、テーブル名が表示される `NULL` は、そのテーブルの各カラムに `NULL` が使用される行を意味します。たとえば、`t1||t2||NULL` は、`t3` のカラムごとに行 `t1` および `t2` のカラムと `NULL` を連結して構築された行を示します。このような行は、`NULL` で補完されていると言われます。

ここで、ネストされた外部結合を含むクエリーについて考えてみます:

```
SELECT * FROM T1 LEFT JOIN
      (T2 LEFT JOIN T3 ON P2(T2,T3))
      ON P1(T1,T2)
WHERE P(T1,T2,T3)
```

このクエリーでは、ネステッドループパターンを変更して次のものを取得します:

```
FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t2 in T2 such that P1(t1,t2) {
    BOOL f2:=FALSE;
    FOR each row t3 in T3 such that P2(t2,t3) {
```

```

IF P(t1,t2,t3) {
  t:=t1||t2||t3; OUTPUT t;
}
f2=TRUE;
f1=TRUE;
}
IF (!f2) {
  IF P(t1,t2,NULL) {
    t:=t1||t2||NULL; OUTPUT t;
  }
  f1=TRUE;
}
}
IF (!f1) {
  IF P(t1,NULL,NULL) {
    t:=t1||NULL||NULL; OUTPUT t;
  }
}
}

```

一般に、外部結合操作の最初の内部テーブルのネストしたループでは、ループの前にオフにされ、ループのあとにチェックされるフラグが導入されます。フラグは、外部テーブルの現在行で、内側オペランドを表すテーブルからの一致が見つかったときにオンにされます。ループサイクルの最後でフラグがまだオフの場合は、外部テーブルの現在行で一致が見つかりませんでした。この例では、行が内部テーブルのカラムの **NULL** 値で補完されます。結果の行は、出力の最終チェックまたは次のネストしたループに渡されますが、行が、埋め込まれたすべての外部結合の結合条件を満たしている場合に限られます。

この例では、次の式でテーブルされる外部結合テーブルが埋め込まれています:

```
(T2 LEFT JOIN T3 ON P2(T2,T3))
```

内部結合を含むクエリの場合、オプティマイザは次のように、ネステッドループの異なる順序を選択できます:

```

FOR each row t3 in T3 {
  FOR each row t2 in T2 such that P2(t2,t3) {
    FOR each row t1 in T1 such that P1(t1,t2) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
    }
  }
}

```

外部結合を使用するクエリの場合、オプティマイザは、外部テーブルのループが内部テーブルのループより前にある順序のみを選択できます。つまり、外部結合を含むクエリでは、1つだけのネスト順序しか使用できません。次のクエリでは、オプティマイザは2つの異なるネストを評価します。両方のネストで、**T1** は外部結合で使用されているため、外側のループで処理される必要があります。**T2** と **T3** は内部結合で使用されているため、その結合は内側のループで処理される必要があります。ただし、結合は内部結合であるため、**T2** と **T3** はどちらの順序でも処理できます。

```

SELECT * T1 LEFT JOIN (T2,T3) ON P1(T1,T2) AND P2(T1,T3)
WHERE P(T1,T2,T3)

```

あるネストによって **T2** が評価され、次に **T3** が評価されます:

```

FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t2 in T2 such that P1(t1,t2) {
    FOR each row t3 in T3 such that P2(t1,t3) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
      f1:=TRUE
    }
  }
  IF (!f1) {
    IF P(t1,NULL,NULL) {
      t:=t1||NULL||NULL; OUTPUT t;
    }
  }
}

```

}

もう一方のネストでは、**T3** を評価してから、**T2** を評価します:

```
FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t3 in T3 such that P2(t1,t3) {
    FOR each row t2 in T2 such that P1(t1,t2) {
      IF P(t1,t2,t3) {
        t=t1||t2||t3; OUTPUT t;
      }
      f1:=TRUE
    }
  }
  IF (!f1) {
    IF P(t1,NULL,NULL) {
      t=t1||NULL||NULL; OUTPUT t;
    }
  }
}
```

内部結合の Nested Loop アルゴリズムについて説明した際に、クエリー実行のパフォーマンスに与える影響が大きい場合があるという詳細については省きました。いわゆる「プッシュダウン」条件については説明しませんでした。たとえば、**WHERE** 条件 **P(T1,T2,T3)** を論理積標準形によって表現できるとします。

```
P(T1,T2,T2) = C1(T1) AND C2(T2) AND C3(T3).
```

この場合、MySQL では、内部結合を使用したクエリーの実行に次のネステッドループアルゴリズムが実際に使用されます:

```
FOR each row t1 in T1 such that C1(t1) {
  FOR each row t2 in T2 such that P1(t1,t2) AND C2(t2) {
    FOR each row t3 in T3 such that P2(t2,t3) AND C3(t3) {
      IF P(t1,t2,t3) {
        t=t1||t2||t3; OUTPUT t;
      }
    }
  }
}
```

等位項 **C1(T1)**、**C2(T2)**、**C3(T3)** がそれぞれ、もっとも内側のループから、評価可能なもっとも外側のループまで押し出されることがわかります。**C1(T1)** がきわめて制限の強い条件である場合、このコンディションプッシュダウンによって、テーブル **T1** から内側ループに渡される行数が大幅に少なくなることがあります。結果として、クエリーの実行時間が大幅に短縮される可能性があります。

外部結合を含むクエリーでは、外部テーブルの現在行で内部テーブルに一致があることが見つかったあとのみ、**WHERE** 条件がチェックされます。そのため、内側のネストしたループからのプッシュ条件の最適化は、外部結合を含むクエリーには直接適用できません。ここでは、一致が発生したときにオンになるフラグで保護される条件付きプッシュダウン述語を導入する必要があります。

外部結合を使用した次の例を思い出してください:

```
P(T1,T2,T3)=C1(T1) AND C(T2) AND C3(T3)
```

この例では、保護されたプッシュダウン条件を使用するネステッドループアルゴリズムは次のようになります:

```
FOR each row t1 in T1 such that C1(t1) {
  BOOL f1:=FALSE;
  FOR each row t2 in T2
    such that P1(t1,t2) AND (f1?C2(t2):TRUE) {
    BOOL f2:=FALSE;
    FOR each row t3 in T3
      such that P2(t2,t3) AND (f1&&f2?C3(t3):TRUE) {
      IF (f1&&f2?TRUE:(C2(t2) AND C3(t3))) {
        t=t1||t2||t3; OUTPUT t;
      }
      f2=TRUE;
    }
    f1=TRUE;
  }
  IF (!f2) {
```

```
IF (f1?TRUE:C2(t2) && P(t1,t2,NULL)) {
  t:=t1||t2||NULL; OUTPUT t;
}
f1=TRUE;
}
}
IF (f1 && P(t1,NULL,NULL)) {
  t:=t1||NULL||NULL; OUTPUT t;
}
}
```

一般に、プッシュダウン述語は $P_1(T_1, T_2)$ や $P(T_2, T_3)$ などの結合条件から抽出できます。この場合、プッシュダウン述語は、対応する外部結合操作によって生成される `NULL` が補完された行の述語のチェックを妨げるフラグによっても保護されます。

同じネストされた結合内のある内部テーブルから別のテーブルへのキーによるアクセスは、`WHERE` 条件から述語によって誘導された場合は禁止されます。

8.2.1.9 外部結合の最適化

外部結合には、`LEFT JOIN` および `RIGHT JOIN` が含まれます。

MySQL は、次のように `A LEFT JOIN B join_specification` を実装します:

- テーブル `B` は、テーブル `A` と `A` が依存するすべてのテーブルに依存して設定されます。
- テーブル `A` は、`LEFT JOIN` 条件で使用されるすべてのテーブル (`B` を除く) に依存して設定されます。
- `LEFT JOIN` 条件は、テーブル `B` からの行の取得方法を決定するために使用されます。(言い換えると、`WHERE` 句内のすべての条件が使用されません)。
- テーブルは常にそれが依存するすべてのテーブルのあとに読み取られることを除き、すべての標準の結合最適化が実行されます。循環依存関係がある場合は、エラーが発生します。
- すべての標準 `WHERE` 最適化が実行されます。
- `A` に `WHERE` 句に一致する行があるが、`B` に `ON` 条件に一致する行がない場合、すべてのカラムが `NULL` に設定された追加の `B` 行が生成されます。
- `LEFT JOIN` を使用して、一部のテーブルに存在しない行を検索し、`WHERE` 部分の `col_name IS NULL` のテストを実行した場合 (ここで `col_name` は `NOT NULL` と宣言されているカラム)、MySQL は `LEFT JOIN` 条件に一致する 1 つの行が見つかったあとに、それ以上の行 (の特定のキーの組み合わせ) の検索を停止します。

`RIGHT JOIN` の実装は、テーブルロールを逆にした `LEFT JOIN` の実装に似ています。セクション8.2.1.10「外部結合の単純化」で説明されているように、右結合は同等の左結合に変換されます。

`LEFT JOIN` では、生成された `NULL` 行の `WHERE` 条件が常に `false` の場合、`LEFT JOIN` は内部結合に変更されます。たとえば、`t2.column1` が `NULL` であった場合、次のクエリーの `WHERE` 句は `false` になります。

```
SELECT * FROM t1 LEFT JOIN t2 ON (column1) WHERE t2.column2=5;
```

したがって、クエリーを内部結合に変換しても安全です:

```
SELECT * FROM t1, t2 WHERE t2.column2=5 AND t1.column1=t2.column1;
```

MySQL 8.0.14 以降では、定数リテラル式から発生する簡単な `WHERE` 条件は、最適化の後の段階ではなく、準備中に削除され、結合がすでに簡略化されています。以前に簡易条件を削除すると、オプティマイザは外部結合を内部結合に変換できます。これにより、次のような `WHERE` 句に簡易条件を含む外部結合を含むクエリーの計画が改善される可能性があります:

```
SELECT * FROM t1 LEFT JOIN t2 ON condition_1 WHERE condition_2 OR 0 = 1
```

オプティマイザは、準備中に `0 = 1` が常に `false` であることを確認し、`OR 0 = 1` を冗長にして削除し、次の状態のままにします:

```
SELECT * FROM t1 LEFT JOIN t2 ON condition_1 where condition_2
```


これで、オプティマイザは、次のようにクエリーを内部結合としてリライトできます:

```
SELECT * FROM t1 JOIN t2 WHERE condition_1 AND condition_2
```

これによりクエリー計画が改善される場合、オプティマイザはテーブル `t1` の前にテーブル `t2` を使用できるようになりました。テーブルの結合順序に関するヒントを提供するには、オプティマイザヒントを使用します。セクション 8.9.3 「オプティマイザヒント」を参照してください。または、`STRAIGHT_JOIN` を使用します。セクション 13.2.10 「SELECT ステートメント」を参照してください。ただし、`STRAIGHT_JOIN` では準結合変換が無効になるため、インデックスの使用が妨げられる場合があります。セクション 8.2.2.1 「準結合変換による IN および EXISTS サブクエリー述語の最適化」を参照してください。

8.2.1.10 外部結合の単純化

クエリーの `FROM` 句内のテーブル式は、多くの場合単純化されます。

パーサーステージでは、右外部結合操作を含むクエリーは、左結合操作のみを含む同等のクエリーに変換されます。一般的に、変換は次の右結合になるように実行されます:

```
(T1, ...) RIGHT JOIN (T2, ...) ON P(T1, ..., T2, ...)
```

次の同等の左結合になります:

```
(T2, ...) LEFT JOIN (T1, ...) ON P(T1, ..., T2, ...)
```

形式 `T1 INNER JOIN T2 ON P(T1,T2)` のすべての内部結合式は、`WHERE` 条件に (または埋め込まれる結合の結合条件が存在する場合は、それに) 等位項として結合されるリスト `T1,T2, P(T1,T2)` によって、置き換えられます。

オプティマイザが外部結合操作の計画を評価するときは、そのような操作ごとに外部テーブルが内部テーブルの前にアクセスされる計画のみが考慮されます。このような計画でのみネステッドループアルゴリズムを使用して外部結合を実行できるため、オプティマイザの選択は制限されます。

次の形式のクエリーについて考えてみます。`R(T2)` では、テーブル `T2` の一致する行数が大幅に絞り込まれます:

```
SELECT * T1 LEFT JOIN T2 ON P1(T1,T2)
WHERE P(T1,T2) AND R(T2)
```

クエリーが書き込み済として実行される場合、オプティマイザでは選択できませんが、より制限の少ないテーブル `T1` にアクセスしてから、より限定されたテーブル `T2` にアクセスすると、非常に非効率的な実行計画が生成される可能性があります。

かわりに、`WHERE` 条件が `NULL` 拒否の場合、MySQL はクエリーを外部結合操作なしのクエリーに変換します。(つまり、外部結合を内部結合に変換します。) 条件は、その操作に対して生成された `NULL` で補完された行に対して `FALSE` または `UNKNOWN` と評価された場合、外部結合操作に対して `NULL` 拒否と呼ばれます。

したがって、この外部結合の場合:

```
T1 LEFT JOIN T2 ON T1.A=T2.A
```

これらのような条件は、`NULL` で補完された行 (`T2` カラムが `NULL` に設定されている場合) では `true` にできないため、`NULL` 拒否されます:

```
T2.B IS NOT NULL
T2.B > 3
T2.C <= T1.C
T2.B < 2 OR T2.C > 1
```

次のような条件は、`NULL` で補完された行に対して `true` になる可能性があるため、`NULL` 拒否されません:

```
T2.B IS NULL
T1.B < 3 OR T2.B IS NOT NULL
T1.B < 3 OR T2.B > 3
```

外部結合操作で条件が `NULL` 拒否かどうかをチェックする一般的なルールは、次のとおりです:

- `A IS NOT NULL` の形式で、`A` は内部テーブルのいずれかの属性です

- いずれかの引数が **NULL** の場合に **UNKNOWN** に評価される内部テーブルへの参照を含む述語です
- 結合として **NULL** 拒否条件を含む結合です
- **NULL** 拒否条件の論理積です

条件は、クエリー内で、ある外部結合操作に対しては **NULL** を受け付けませんが、ほかの外部結合操作に対しては **NULL** を受け付ける場合があります。このクエリーでは、**WHERE** 条件は 2 番目の外部結合操作では **NULL** 拒否されますが、最初の外部結合操作では **NULL** 拒否されません:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
      LEFT JOIN T3 ON T3.B=T1.B
WHERE T3.C > 0
```

WHERE 条件がクエリーの外部結合操作に対して **NULL** を受け付けない場合、外部結合操作は内部結合操作に置き換えられます。

たとえば、前述のクエリーでは、2 番目の外部結合は **NULL** 拒否され、内部結合で置換できます:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
      INNER JOIN T3 ON T3.B=T1.B
WHERE T3.C > 0
```

元のクエリーの場合、最適化は単一のテーブルアクセス順序 **T1,T2,T3** と互換性のある計画のみを評価します。リライトされたクエリーでは、アクセス順序 **T3,T1,T2** も考慮されます。

ある外部結合操作の変換によって、別の操作の変換がトリガーされることがあります。そのため、次のクエリー:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
      LEFT JOIN T3 ON T3.B=T2.B
WHERE T3.C > 0
```

最初にクエリーに変換されます:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
      INNER JOIN T3 ON T3.B=T2.B
WHERE T3.C > 0
```

これはクエリーと同等です:

```
SELECT * FROM (T1 LEFT JOIN T2 ON T2.A=T1.A), T3
WHERE T3.C > 0 AND T3.B=T2.B
```

条件 **T3.B=T2.B** が **NULL** 拒否であるため、残りの外部結合操作を内部結合に置き換えることもできます。これにより、外部結合のないクエリーが発生します:

```
SELECT * FROM (T1 INNER JOIN T2 ON T2.A=T1.A), T3
WHERE T3.C > 0 AND T3.B=T2.B
```

最適化が埋込み外部結合操作の置換に成功しても、埋込み外部結合を変換できない場合があります。次のクエリー:

```
SELECT * FROM T1 LEFT JOIN
      (T2 LEFT JOIN T3 ON T3.B=T2.B)
      ON T2.A=T1.A
WHERE T3.C > 0
```

次に変換されます:

```
SELECT * FROM T1 LEFT JOIN
      (T2 INNER JOIN T3 ON T3.B=T2.B)
      ON T2.A=T1.A
WHERE T3.C > 0
```

それは埋め込む外部結合操作を含む形式にのみ書き換えることができます。

```
SELECT * FROM T1 LEFT JOIN
      (T2,T3)
```

```
ON (T2.A=T1.A AND T3.B=T2.B)
WHERE T3.C > 0
```

クエリーで埋込み外部結合操作を変換しようとする場合は、埋込み外部結合の結合条件を **WHERE** 条件とともに考慮する必要があります。このクエリーでは、埋込み外部結合の **WHERE** 条件は NULL 拒否されませんが、埋込み外部結合 **T2.A=T1.A AND T3.C=T1.C** の結合条件は NULL 拒否されます:

```
SELECT * FROM T1 LEFT JOIN
  (T2 LEFT JOIN T3 ON T3.B=T2.B)
  ON T2.A=T1.A AND T3.C=T1.C
WHERE T3.D > 0 OR T1.D > 0
```

したがって、クエリーは次のように変換できます:

```
SELECT * FROM T1 LEFT JOIN
  (T2, T3)
  ON T2.A=T1.A AND T3.C=T1.C AND T3.B=T2.B
WHERE T3.D > 0 OR T1.D > 0
```

8.2.1.11 Multi-Range Read の最適化

セカンダリインデックスでの範囲スキャンを使用して行を読み取ると、テーブルが大きく、ストレージエンジンのキャッシュに格納されていない場合、ベーステーブルへのランダムディスクアクセスが多発する結果になることがあります。Disk-Sweep Multi-Range Read (MRR) 最適化を使用すると、MySQL は、最初にインデックスだけをスキャンし、該当する行のキーを収集することによって、範囲スキャンのランダムディスクアクセスの回数を軽減しようとします。続いてキーがソートされ、最後に主キーの順序を使用してベーステーブルから行が取得されます。Disk-Sweep MRR の目的は、ランダムディスクアクセスの回数を減らし、その代わりに、ベーステーブルデータの順次スキャンを増やすことです。

Multi-Range Read の最適化には、次のメリットがあります。

- MRR により、データ行はインデックスタプルに基づいて、ランダムな順序ではなく、順次アクセスできます。サーバーはクエリー条件を満たすインデックスタプルセットを取得し、それらをデータ行 ID 順に従ってソートし、ソートされたタプルを使用して、データ行を順番に取得します。これにより、データアクセスの効率が向上し、負荷が軽減されます。
- MRR により、範囲インデックススキャンや結合属性にインデックスを使用する等価結合などの、インデックスタプル経由でのデータ行へのアクセスを必要とする操作のキーアクセスのリクエストのバッチ処理が可能になります。MRR はインデックス範囲のシーケンスを反復処理して、対象のインデックスタプルを取得します。これらの結果が累積されると、それらに対応するデータ行にアクセスするために使用されます。データ行の読み取りを開始する前に、すべてのインデックスタプルを取得する必要はありません。

MRR 最適化は、仮想生成カラムに作成されたセカンダリインデックスではサポートされていません。InnoDB では、仮想生成カラムのセカンダリインデックスがサポートされます。

次のシナリオでは、MRR の最適化に利益がある場合について説明しています。

シナリオ A: インデックス範囲スキャンと等価結合操作で、InnoDB テーブルと MyISAM テーブルに対して MRR を使用できます。

1. インデックスタブルの一部はバッファーに累積されます。
2. バッファー内のタプルはそれらのデータ行 ID によってソートされます。
3. データ行には、ソートされたインデックスタブルシーケンスに従ってアクセスされます。

シナリオ B: 複数範囲インデックススキャンで、または属性によって等価結合を実行する際に、NDB テーブルに対して、MRR を使用できます。

1. 単一キー範囲の可能性のある範囲の一部は、クエリーが送信される中央ノード上のバッファーに累積されます。
2. 範囲はデータ行にアクセスする実行ノードに送信されます。
3. アクセスされた行はパッケージに格納され、中央ノードに返送されます。

- 受け取ったデータ行を含むパッケージはバッファーに入れられます。
- データ行がバッファーから読み取られます。

MRR が使用された場合は、`EXPLAIN` 出力の `Extra` カラムに `Using MRR` と示されます。

InnoDB と MyISAM は、クエリー結果を生成するために完全なテーブル行にアクセスする必要がない場合、MRR を使用しません。これは、(カバーするインデックス経由で) インデックスタブル内の情報に完全に基づいて結果を生成できる場合であり、MRR にメリットはありません。

2 つの `optimizer_switch` システム変数フラグは、MRR 最適化の使用へのインターフェースを提供します。 `mrr` フラグは MRR を有効にするかどうかを制御します。 `mrr` が有効 (on) の場合、 `mrr_cost_based` フラグは、オプティマイザが MRR (on) を使用するかどうか、または可能な場合は MRR (off) を使用するかどうかを制御します。 デフォルトでは、 `mrr` は on で、 `mrr_cost_based` は on です。 [セクション 8.9.2 「切り替え可能な最適化」](#) を参照してください。

MRR では、ストレージエンジンが、そのバッファーに割り当てることができるメモリーの量のガイドラインとして、 `read_rnd_buffer_size` システム変数の値を使用します。 エンジンは最大 `read_rnd_buffer_size` バイトを使用して、単一のパスで処理する範囲の数を判断します。

8.2.1.12 Block Nested Loop 結合と Batched Key Access 結合

MySQL では、結合テーブルと結合バッファの両方へのインデックスアクセスを使用するバッチキーアクセス (BKA) 結合アルゴリズムを使用できます。 BKA アルゴリズムでは、内部結合、外部結合、およびネストされた外部結合を含む準結合操作がサポートされています。 BKA には、テーブルスキャンの効率性の向上による結合パフォーマンスの改善というメリットもあります。 また、以前は内部結合にのみ使用されていた Block Nested-Loop (BNL) 結合アルゴリズムが拡張され、ネストされた外部結合を含む外部結合および準結合操作に使用できます。

次のセクションでは、元の BNL アルゴリズムの拡張の基礎にある結合バッファ管理、拡張 BNL アルゴリズム、および BKA アルゴリズムについて説明します。 準結合戦略の詳細は、 [セクション 8.2.2.1 「準結合変換による IN および EXISTS サブクエリー述語の最適化」](#) を参照してください

- [Block Nested Loop および Batched Key Access アルゴリズムの結合バッファ管理](#)
- [外部結合および準結合のブロックネストループアルゴリズム](#)
- [Batched Key Access 結合](#)
- [ブロックネストループおよびバッチキーアクセスアルゴリズムのオプティマイザヒント](#)

Block Nested Loop および Batched Key Access アルゴリズムの結合バッファ管理

MySQL では、結合バッファを使用して、内部テーブルへのインデックスアクセスなしで内部結合のみでなく、サブクエリーのフラット化後に表示される外部結合および準結合も実行できます。 さらに、内部テーブルへのインデックスアクセスがある場合、結合バッファを効率的に使用できます。

結合バッファ管理コードは、目的の行カラムの値を格納する際に、結合バッファ領域を少し効率的に利用します。 行カラムの値が `NULL` の場合に行カラムにバッファ内の追加バイトを割り当てず、 `VARCHAR` 型の値には最小数のバイトが割り当てられます。

コードでは、標準と増分の 2 つの種類のバッファをサポートします。 結合テーブル `t1` と `t2` に結合バッファ `B1` が使用されており、この操作の結果が結合バッファ `B2` を使用して、テーブル `t3` と結合されるとします。

- 標準結合バッファには、各結合オペランドからのカラムが格納されます。 `B2` が通常の結合バッファの場合、 `B2` に配置される各行 `r` は、 `B1` の行 `r1` のカラムと、テーブル `t3` の一致する行 `r2` の興味深いカラムで構成されます。
- 増分結合バッファには、2 つめの結合オペランドによって生成されるテーブルの行からのカラムのみが格納されます。 つまり、それは 1 つめのオペランドバッファからの行の増分になります。 `B2` が増分結合バッファである場合、それには、 `B1` からの行 `r1` へのリンクとともに、行 `r2` の対象のカラムが格納されます。

増分結合バッファは常に、前の結合操作からの結合バッファに相対的な増分になるため、最初の結合操作からのバッファは常に標準バッファになります。 直前の例では、テーブル `t1` および `t2` を結合するために使用されるバッファ `B1` は標準バッファである必要があります。

結合操作に使用される増分バッファの各行には、結合されるテーブルからの行の対象カラムのみが格納されます。これらのカラムには、最初の結合オペランドによって生成されたテーブルからの一致する行の対象カラムへの参照が追加されます。増分バッファ内の複数の行から、カラムが前の結合バッファに格納されている同じ行 *r* を参照できます。ただし、これらのすべての行が行 *r* に一致する場合にかぎりです。

増分バッファにより、前の結合操作で使用されたバッファからのカラムのコピーの頻度を少なくできます。これにより、一般に、最初の結合オペランドによって生成された行が 2 つめの結合オペランドによって生成される複数の行に一致する可能性があるため、バッファ領域が節約されます。最初のオペランドからの行のコピーを何度も行う必要がありません。さらに、増分バッファにより、コピー時間の短縮のため、処理時間も節約されます。

MySQL 8.0 では、`optimizer_switch` システム変数の `block_nested_loop` フラグは次のように機能します:

- MySQL 8.0.20 より前は、オプティマイザが Block Nested Loop 結合アルゴリズムを使用する方法を制御していました。
- MySQL 8.0.18 以降では、ハッシュ結合の使用も制御します ([セクション 8.2.1.4 「ハッシュ結合の最適化」](#) を参照)。
- MySQL 8.0.20 以降、このフラグはハッシュ結合のみを制御し、ブロックネストループアルゴリズムはサポートされなくなりました。

`batched_key_access` フラグは、オプティマイザがバッチキーアクセス結合アルゴリズムを使用する方法を制御します。

デフォルトで、`block_nested_loop` は `on` で `batched_key_access` は `off` です。 [セクション 8.9.2 「切り替え可能な最適化」](#) を参照してください。オプティマイザヒントも適用できます。 [ブロックネストループおよびバッチキーアクセスアルゴリズムのオプティマイザヒント](#) を参照してください。

準結合戦略の詳細は、 [セクション 8.2.2.1 「準結合変換による IN および EXISTS サブクエリー述語の最適化」](#) を参照してください

外部結合および準結合のブロックネストループアルゴリズム

MySQL BNL アルゴリズムの元の実装は、外部結合および準結合操作をサポートするように拡張されました (後でハッシュ結合アルゴリズムに置き換えられました。 [セクション 8.2.1.4 「ハッシュ結合の最適化」](#) を参照)。

結合バッファを使用して、これらの操作が実行されると、バッファに入れられた各行に一致フラグが付加されます。

結合バッファを使用して、外部結合操作が実行された場合、2 つめのオペランドによって生成されたテーブルの各行で、結合バッファ内の各行に対する一致がチェックされます。一致が見つかったら、新しく拡張された行が形成され (元の行に 2 つめのオペランドからのカラムを追加)、残りの結合操作によるさらなる拡張のために送られます。さらに、バッファ内の一致した行の一致フラグが有効にされます。結合されるテーブル内のすべての行が調査されたあとに、結合バッファがスキャンされます。有効にされた一致フラグがないバッファからの各行は、`NULL` の補完 (2 つめのオペランドの各カラムの `NULL` 値) によって拡張され、残りの結合操作によるさらなる拡張のために送られます。

MySQL 8.0 では、`optimizer_switch` システム変数の `block_nested_loop` フラグは次のように機能します:

- MySQL 8.0.20 より前は、オプティマイザが Block Nested Loop 結合アルゴリズムを使用する方法を制御していました。
- MySQL 8.0.18 以降では、ハッシュ結合の使用も制御します ([セクション 8.2.1.4 「ハッシュ結合の最適化」](#) を参照)。
- MySQL 8.0.20 以降、このフラグはハッシュ結合のみを制御し、ブロックネストループアルゴリズムはサポートされなくなりました。

詳しくは [セクション 8.9.2 「切り替え可能な最適化」](#) をご覧ください。オプティマイザヒントも適用できます。 [ブロックネストループおよびバッチキーアクセスアルゴリズムのオプティマイザヒント](#) を参照してください。

`EXPLAIN` 出力で、`Extra` 値に `Using join buffer (Block Nested Loop)` が含まれ、`type` 値が `ALL`、`index`、または `range` の場合に、テーブルへの BNL の使用が示されます。

準結合戦略の詳細は、[セクション8.2.2.1「準結合変換による IN および EXISTS サブクエリー述語の最適化」](#) を参照してください

Batched Key Access 結合

MySQL では Batched Key Access (BKA) 結合アルゴリズムと呼ばれるテーブルの結合の方法を実装しています。BKA は、2 つめの結合オペランドによって生成されるテーブルへのインデックスアクセスがある場合に適用できます。BNL 結合アルゴリズムと同様、BKA 結合アルゴリズムでは、結合バッファを使用して、結合操作の最初のオペランドによって生成された行の対象カラムを累積します。次に、BKA アルゴリズムは、バッファ内のすべての行に対し、結合されるテーブルにアクセスするためのキーを構築し、これらのキーをインデックスルックアップのために、データベースエンジンに一括で送信します。キーは、Multi-Range Read (MRR) インタフェース経由で、エンジンに送信されます ([セクション8.2.1.11「Multi-Range Read の最適化」](#) を参照してください)。キーの送信後、MRR エンジン関数は最適な方法で、インデックス内のルックアップを実行し、これらのキーによって見つかった結合されたテーブルの行をフェッチし、BKA 結合アルゴリズムに一致する行の提供を開始します。一致する各行は結合バッファ内の行への参照が組み合わされます。

BKA が使用される場合、[join_buffer_size](#) の値によって、ストレージエンジンへの個々のリクエストでのキーのバッチの大きさが定義されます。バッファが大きいほど、結合操作の右側のテーブルへの順次アクセスが多くなり、パフォーマンスが大幅に向上する可能性があります。

BKA を使用するには、[optimizer_switch](#) システム変数の [batched_key_access](#) フラグが `on` に設定されている必要があります。BKA では MRR を使用するため、[mrr](#) フラグも `on` に設定されている必要があります。現在、MRR のコスト見積もりはきわめて悲観的です。したがって、BKA を使用するには、[mrr_cost_based](#) を `off` にする必要があります。次の設定によって、BKA が有効になります。

```
mysql> SET optimizer_switch='mrr=on,mrr_cost_based=off,batched_key_access=on';
```

MRR 関数が実行される 2 つのシナリオがあります。

- 最初のシナリオは、[InnoDB](#) や [MyISAM](#) などの従来のディスクベースのストレージエンジンで使用されます。これらのエンジンでは通常、結合バッファからのすべての行のキーが一度に MRR インタフェースに送信されます。エンジン固有の MRR 関数は、送信されたキーのインデックスルックアップを実行し、それらから行 ID (または主キー) を取得して、BKA アルゴリズムからのリクエストによって、これらの選択されたすべての行 ID の行を 1 つずつフェッチします。各行は、結合バッファ内の一一致した行へのアクセスを可能にするアソシエーション参照とともに返されます。行は MRR 関数によって最適な方法でフェッチされます。それらは、行 ID (主キー) 順でフェッチされます。これにより、読み取りがランダムな順序ではなく、ディスク順になるため、パフォーマンスが向上します。
- 2 つめのシナリオは、[NDB](#) などのリモートストレージエンジンで使用されます。結合バッファからの行の一部のキーのパッケージが、それらのアソシエーションとともに、MySQL Server (SQL ノード) によって、MySQL Cluster データノードに送信されます。返信で、SQL ノードは、対応するアソシエーションが組み合わされた一致する行のパッケージ (または複数のパッケージ) を受け取ります。BKA 結合アルゴリズムでは、これらの行を取得し、新しく結合された行を構築します。次に、新しいキーセットがデータノードに送信され、返されたパッケージからの行が新しい結合された行の構築に使用されます。このプロセスは、結合バッファからの最後のキーがデータノードに送信され、SQL ノードがこれらのキーに一致するすべての行を受け取り、結合するまで、続行されます。これにより、SQL ノードによってデータノードに送信されるキーを含むパッケージが少なくなることは、結合操作を実行するために、それとデータノード間のラウンドトリップが少なくなることを意味するため、パフォーマンスが向上します。

最初のシナリオでは、結合バッファの一部がインデックスルックアップによって選択され、MRR 関数へのパラメータとして渡される行 ID (主キー) を格納するために予約されます。

結合バッファからの行に対して構築されるキーを格納するための特別なバッファはありません。代わりに、バッファ内の次の行のキーを構築する関数が、MRR 関数へのパラメータとして渡されます。

`EXPLAIN` 出力で、`Extra` 値に `Using join buffer (Batched Key Access)` が含まれ、`type` 値が `ref` または `eq_ref` の場合に、テーブルへの BKA の使用が示されます。

ブロックネストループおよびバッチキーアクセスアルゴリズムのオプティマイザヒント

BNL および BKA アルゴリズムのセッション全体でのオプティマイザの使用を制御するために [optimizer_switch](#) システム変数を使用することに加えて、MySQL はオプティマイザヒントをサポートして、ステートメントごとにオプティマイザに影響を与えます。 [セクション8.9.3「オプティマイザヒント」](#) を参照してください。

BNL または BKA ヒントを使用して外部結合の内部テーブルの結合バッファリングを有効にするには、外部結合のすべての内部テーブルに対して結合バッファリングを有効にする必要があります。

8.2.1.13 条件フィルタ

結合処理では、接頭辞行は、結合のあるテーブルから次のテーブルに渡される行です。通常、オプティマイザは、行の組合せの数が急速に増加しないように、接頭辞数が少ないテーブルを結合順序の早い段階に配置しようとします。オプティマイザがあるテーブルから選択されて次のテーブルに渡される行の条件に関する情報を使用できる範囲では、行の見積りをより正確に計算し、最適な実行計画を選択できます。

条件フィルタリングを使用しない場合、テーブルの接頭辞行数は、オプティマイザが選択したアクセス方法に応じて、**WHERE** 句によって選択された推定行数に基づきます。条件フィルタリングを使用すると、オプティマイザは、アクセス方法で考慮されない他の関連条件を **WHERE** 句で使用できるため、接頭辞の行数の見積りが改善されます。たとえば、結合で現在のテーブルから行を選択するために使用できるインデックススペースのアクセス方法がある場合でも、次のテーブルに渡される行を修飾するための見積りをフィルタ (さらに制限) できる、**WHERE** 句内のテーブルに対する追加の条件が存在する可能性があります。

条件は、次の場合にのみフィルタリング推定に寄与します:

- 現在のテーブルを参照します。
- これは、定数値または結合順序内の以前のテーブルの値に依存します。
- アクセス方法ではまだ考慮されていません。

EXPLAIN 出力では、**rows** カラムに選択したアクセス方法の行の見積りが示され、**filtered** カラムに条件フィルタの効果が反映されます。**filtered** 値はパーセンテージで表されます。最大値は 100 で、これは行のフィルタリングが行われなかったことを意味します。100 から減少する値は、フィルタリングの量が増加していることを示します。

接頭辞の行数 (結合で現在のテーブルから次のテーブルに渡されると推定される行数) は、**rows** 値と **filtered** 値の積です。つまり、接頭辞の行数は推定された行数で、推定されたフィルタリング効果によって削減されます。たとえば、**rows** が 1000 で、**filtered** が 20% の場合、条件フィルタリングによって 1000 の推定行数が接頭辞の行数 $1000 \times 20\% = 1000 \times .2 = 200$ に削減されます。

次のクエリーを考慮してください。

```
SELECT *
FROM employee JOIN department ON employee.dept_no = department.dept_no
WHERE employee.first_name = 'John'
AND employee.hire_date BETWEEN '2018-01-01' AND '2018-06-01';
```

データセットに次の特性があるとします:

- **employee** テーブルには 1024 行あります。
- **department** テーブルには 12 行あります。
- どちらのテーブルにも、**dept_no** のインデックスがあります。
- **employee** テーブルには、**first_name** のインデックスがあります。
- **employee.first_name** では、8 行が次の条件を満たします:

```
employee.first_name = 'John'
```

- 150 行が **employee.hire_date** で次の条件を満たしています:

```
employee.hire_date BETWEEN '2018-01-01' AND '2018-06-01'
```

- 1 行が両方の条件を満たしています:

```
employee.first_name = 'John'
AND employee.hire_date BETWEEN '2018-01-01' AND '2018-06-01'
```

条件フィルタリングを使用しない場合、**EXPLAIN** は次のような出力を生成します:

```

+-----+-----+-----+-----+-----+-----+
| id | table | type | possible_keys | key | ref | rows | filtered |
+-----+-----+-----+-----+-----+-----+
| 1 | employee | ref | name,h_date,dept | name | const | 8 | 100.00 |
| 1 | department | eq_ref | PRIMARY | PRIMARY | dept_no | 1 | 100.00 |
+-----+-----+-----+-----+-----+

```

`employee` の場合、`name` インデックスのアクセス方法では、'John' の名前と一致する 8 行が取得されます。フィルタリングは行われず (`filtered` は 100%) ため、すべての行が次のテーブルの接頭辞行になります: 接頭辞の行数は、`rows × filtered = 8 × 100% = 8` です。

条件フィルタリングを使用すると、オプティマイザでは、アクセス方法で考慮されない `WHERE` 句の条件も考慮されます。この場合、オプティマイザはヒューリスティックを使用して、`employee.hire_date` での `BETWEEN` 条件に対する 16.31% のフィルタリング効果を見積もります。その結果、`EXPLAIN` では次のような出力が生成されます:

```

+-----+-----+-----+-----+-----+-----+
| id | table | type | possible_keys | key | ref | rows | filtered |
+-----+-----+-----+-----+-----+-----+
| 1 | employee | ref | name,h_date,dept | name | const | 8 | 16.31 |
| 1 | department | eq_ref | PRIMARY | PRIMARY | dept_no | 1 | 100.00 |
+-----+-----+-----+-----+-----+

```

接頭辞の行数は、実際のデータセットをより厳密に反映した `rows × filtered = 8 × 16.31% = 1.3` になりました。

通常、オプティマイザでは、行を渡す次のテーブルがないため、最後に結合されたテーブルの条件フィルタリング効果 (接頭辞行数の削減) は計算されません。 `EXPLAIN` で例外が発生しました: 詳細情報を提供するために、最後の結合テーブルを含むすべての結合テーブルのフィルタリング効果が計算されます。

オプティマイザが追加のフィルタリング条件を考慮するかどうかを制御するには、`optimizer_switch` システム変数の `condition_fanout_filter` フラグを使用します (セクション 8.9.2 「切り替え可能な最適化」を参照)。このフラグはデフォルトで有効になっていますが、条件フィルタリングを抑制するために無効にできます (たとえば、特定のクエリーでパフォーマンスが向上することが判明した場合)。

オプティマイザが条件フィルタリングの影響を過度に見積もる場合、条件フィルタリングが使用されていない場合よりもパフォーマンスが低下する可能性があります。このような場合は、次の方法が役立ちます:

- カラムがインデックス付けされていない場合は、オプティマイザがカラム値の分散に関する情報を取得し、その行の見積りを改善できるようにインデックス付けします。
- 同様に、使用可能なカラムヒストグラム情報がない場合は、ヒストグラムを生成します (セクション 8.9.6 「オプティマイザ統計」を参照)。
- 結合順序を変更します。これを実現する方法には、結合順序オプティマイザヒント (セクション 8.9.3 「オプティマイザヒント」を参照)、`SELECT` 直後の `STRAIGHT_JOIN` および `STRAIGHT_JOIN` 結合演算子が含まれます。
- セッションの条件フィルタリングを無効にします:

```
SET optimizer_switch = 'condition_fanout_filter=off';
```

または、特定のクエリーに対して、オプティマイザヒントを使用します:

```
SELECT /*+ SET_VAR(optimizer_switch = 'condition_fanout_filter=off') */ ...
```

8.2.1.14 定数 - フォールディングの最適化

定数値が範囲外であるか、カラムタイプに関して間違ったタイプである定数とカラム値の比較は、クエリーの最適化中に、実行中ではなく行単位で処理されるようになりました。この方法で処理できる比較は、`>`、`>=`、`<`、`<=`、`<>`、`!=`、`=` および `<=>` です。

次のステートメントで作成されたテーブルについて考えてみます:

```
CREATE TABLE t (c TINYINT UNSIGNED NOT NULL);
```

クエリー `SELECT * FROM t WHERE c < 256` の `WHERE` 条件に、`TINYINT UNSIGNED` カラムの範囲外の整数定数 256 が含まれています。これまでは、両方のオペランドを大きい型として扱うことで処理されていましたが、`c` に許

可されている値が定数より小さいため、かわりに `WHERE` 式を `WHERE 1` として折りたたんで、クエリーを `SELECT * FROM t WHERE 1` としてリライトできます。

これにより、オプティマイザは `WHERE` 式を完全に削除できます。カラム `c` が `NULL` 値可能だった (つまり、`TINYINT UNSIGNED` としてのみ定義されていた) 場合、クエリーは次のようにリライトされます:

```
SELECT * FROM t WHERE ti IS NOT NULL
```

折りたたみは、サポートされている MySQL カラムタイプと比較して、次のように定数に対して実行されます:

- **整数カラムタイプ.** 整数型は、次に説明するように、次の型の定数と比較されます:
 - **整数値.** 定数がカラムタイプの範囲外の場合、すでに示すように、比較は `1` または `IS NOT NULL` に折りたたまれます。

定数が範囲境界の場合、比較は `=` に折りたたまれます。次に例を示します (すでに定義されているのと同じテーブルを使用):

```
mysql> EXPLAIN SELECT * FROM t WHERE c >= 255;
***** 1. row *****
   id: 1
  select_type: SIMPLE
    table: t
  partitions: NULL
    type: ALL
possible_keys: NULL
    key: NULL
   key_len: NULL
    ref: NULL
    rows: 5
  filtered: 20.00
  Extra: Using where
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
***** 1. row *****
Level: Note
Code: 1003
Message: /* select#1 */ select `test`.`t`.`ti` AS `ti` from `test`.`t` where (`test`.`t`.`ti` = 255)
1 row in set (0.00 sec)
```

- **浮動小数点値または固定小数点値.** 定数が小数型 (`DECIMAL`, `REAL`, `DOUBLE`, `FLOAT` など) のいずれかで、小数部がゼロ以外の場合は等しくできません。それに応じて折りたたみます。その他の比較の場合は、符号に従って整数値に切り上げまたは切り下げしてから、整数比較ですでに説明されている範囲チェックおよびハンドルを実行します。

小さすぎて `DECIMAL` として表現できない `REAL` 値は、記号に応じて `.01` または `-.01` に丸められ、`DECIMAL` として処理されます。

- **文字列型.** 文字列値を整数型として解釈し、整数値間の比較を処理します。これが失敗した場合は、値を `REAL` として処理してみてください。
- **DECIMAL または REAL カラム.** ここで説明するように、小数型は次の型の定数と比較されます:
 - **整数値.** カラム値の整数部分に対して範囲チェックを実行します。折りたたみ結果がない場合は、定数をカラム値と同じ小数点以下の桁数で `DECIMAL` に変換し、`DECIMAL` として確認します (次を参照)。
 - **DECIMAL または REAL 値.** オーバーフローをチェックします (つまり、定数の整数部分に、カラムの小数型に許可されている桁数より多い数値があるかどうか)。その場合は折りたたみます。

定数の小数点以下の桁数がカラムの型より多い場合は、定数を切り捨てます。比較演算子が `=` または `<>` の場合は、折りたたみます。演算子が `>=` または `<=` の場合は、切捨てるために演算子を調整します。たとえば、カラムタイプが `DECIMAL(3,1)` の場合、`SELECT * FROM t WHERE f >= 10.13` は `SELECT * FROM t WHERE f > 10.1` になります。

定数の小数点以下の桁数がカラムの型より少ない場合は、同じ桁数の定数に変換します。 `REAL` 値のアンダーフロー (小数が少なすぎて表すことができない) の場合は、定数を小数 `0` に変換します。

- 文字列値. 値を整数型として解釈できる場合は、そのように処理します。それ以外の場合は、**REAL** として処理してください。
- **FLOAT** または **DOUBLE** カラム. 定数と比較した **FLOAT(m,n)** または **DOUBLE(m,n)** の値は、次のように処理されます:

値がカラムの範囲をオーバーフローする場合は、折りたたみます。

値の小数点以下が **n** を超える場合は、折りたたみ時に切り捨てて補正します。= と <> の比較の場合は、前述のように **TRUE**、**FALSE** または **IS [NOT] NULL** に折りたたみます。他の演算子の場合は、演算子を調整します。

値が **m** 整数より大きい場合は折りたたみます。

制限. この最適化は、次の場合には使用できません:

1. **BETWEEN** または **IN** を使用した比較。
2. 日付型または時間型を使用する **BIT** のカラムまたはカラム。
3. プリペアドステートメントの準備フェーズ中。ただし、プリペアドステートメントが実際に実行されるときに最適化フェーズ中に適用できます。これは、ステートメントの準備中に定数の値が不明であるためです。

8.2.1.15 IS NULL の最適化

MySQL は、**col_name = constant_value** に対して使用できる同じ最適化を **col_name IS NULL** に対しても実行できます。たとえば、MySQL は、インデックスと範囲を使用して、**IS NULL** を含む **NULL** を検索できます。

例:

```
SELECT * FROM tbl_name WHERE key_col IS NULL;

SELECT * FROM tbl_name WHERE key_col <=> NULL;

SELECT * FROM tbl_name
WHERE key_col=const1 OR key_col=const2 OR key_col IS NULL;
```

WHERE 句に、**NOT NULL** として宣言されているカラムの **col_name IS NULL** 条件が含まれている場合、その式は最適化により除去されます。この最適化は、カラムが **NULL** を生成する可能性がある場合 (たとえば、**LEFT JOIN** の右側のテーブルから生成される場合) には発生しません。

MySQL は、解決済みのサブクエリーで一般的な形式である **col_name = expr OR col_name IS NULL** の組み合わせを最適化することもできます。この最適化が使用された場合、**EXPLAIN** で **ref_or_null** と示されます。

この最適化は、任意のキーパートに対して 1 つの **IS NULL** を処理できます。

テーブル **t2** のカラム **a** および **b** にインデックスがあるとして、最適化されるクエリーのいくつかの例:

```
SELECT * FROM t1 WHERE t1.a=expr OR t1.a IS NULL;

SELECT * FROM t1, t2 WHERE t1.a=t2.a OR t2.a IS NULL;

SELECT * FROM t1, t2
WHERE (t1.a=t2.a OR t2.a IS NULL) AND t2.b=t1.b;

SELECT * FROM t1, t2
WHERE t1.a=t2.a AND (t2.b=t1.b OR t2.b IS NULL);

SELECT * FROM t1, t2
WHERE (t1.a=t2.a AND t2.a IS NULL AND ...)
OR (t1.a=t2.a AND t2.a IS NULL AND ...);
```

ref_or_null はまずリファレンスキーの読み取りを行い、次に **NULL** キー値のある行の個別の検索を実行します。

最適化では、単一の **IS NULL** レベルのみを処理できます。次のクエリーでは、MySQL は式 **(t1.a=t2.a AND t2.a IS NULL)** に対してのみキールックアップを使用し、**b** に対してはキーパートを使用できません。

```
SELECT * FROM t1, t2
WHERE (t1.a=t2.a AND t2.a IS NULL)
OR (t1.b=t2.b AND t2.b IS NULL);
```

8.2.1.16 ORDER BY の最適化

このセクションでは、MySQL が **ORDER BY** 句を満たすためにインデックスを使用できるタイミング、インデックスを使用できない場合に使用される **filesort** 操作、および **ORDER BY** に関するオプティマイザから使用可能な実行計画情報について説明します。

セクション8.2.1.19「**LIMIT クエリーの最適化**」で説明されているように、**LIMIT** を使用する場合と使用しない場合で **ORDER BY** が異なる順序で行を返すことがあります。

- [ORDER BY を満たすためのインデックスの使用](#)
- [filesort を使用して ORDER BY を満たす](#)
- [ORDER BY 最適化への影響](#)
- [ORDER BY 実行計画情報使用可能](#)

ORDER BY を満たすためのインデックスの使用

場合によっては、MySQL でインデックスを使用して **ORDER BY** 句を満たし、**filesort** 操作の実行に伴う余分なソートを回避できます。

インデックスのすべての未使用部分と追加の **ORDER BY** カラムが **WHERE** 句の定数であるかぎり、**ORDER BY** がインデックスと完全に一致しない場合でもインデックスを使用できます。クエリーによってアクセスされるすべてのカラムがインデックスに含まれていない場合、インデックスアクセスが他のアクセス方法よりも安い場合にのみインデックスが使用されます。

([key_part1](#), [key_part2](#)) にインデックスがあると仮定すると、次のクエリーではインデックスを使用して **ORDER BY** 部分を解決できます。オプティマイザが実際にこれを行うかどうかは、インデックスに含まれていないカラムも読み取る必要がある場合に、インデックスの読取りがテーブルスキャンよりも効率的かどうかによって異なります。

- このクエリーでは、([key_part1](#), [key_part2](#)) のインデックスにより、オプティマイザはソートを回避できます:

```
SELECT * FROM t1
ORDER BY key\_part1, key\_part2;
```

ただし、クエリーでは、[key_part1](#) および [key_part2](#) よりも多くのカラムを選択できる **SELECT *** が使用されます。その場合、インデックス全体をスキャンしてテーブルの行を検索し、インデックスにないカラムを検索すると、テーブルをスキャンして結果をソートするよりコストがかかる可能性があります。その場合、オプティマイザはおそらくインデックスを使用しません。**SELECT *** がインデックスカラムのみを選択した場合、インデックスが使用され、ソートは回避されます。

[t1](#) が InnoDB テーブルの場合、テーブルの主キーは暗黙的にインデックスの一部であり、インデックスを使用してこのクエリーの **ORDER BY** を解決できます:

```
SELECT pk, key\_part1, key\_part2 FROM t1
ORDER BY key\_part1, key\_part2;
```

- このクエリーでは、[key_part1](#) は定数であるため、インデックスを介してアクセスされるすべての行は [key_part2](#) の順序であり、**WHERE** 句が選択的でテーブルスキャンよりも安価なインデックスレンジスキャンを行うことができる場合、([key_part1](#), [key_part2](#)) のインデックスはソートを回避します:

```
SELECT * FROM t1
WHERE key\_part1 = constant
ORDER BY key\_part2;
```

- 次の 2 つのクエリーでは、インデックスを使用するかどうか、前述の **DESC** を使用しない同じクエリーと類似しています:

```
SELECT * FROM t1
ORDER BY key\_part1 DESC, key\_part2 DESC;
```

```
SELECT * FROM t1
WHERE key_part1 = constant
ORDER BY key_part2 DESC;
```

- **ORDER BY** の 2 つのカラムは、同じ方向 (**ASC** または両方の **DESC**) または反対方向 (**ASC**、一方の **DESC**) でソートできます。インデックスの使用条件は、インデックスの均一性は同じである必要があるが、実際の方向は同じである必要がないことです。

クエリーで **ASC** と **DESC** が混在している場合、インデックスで対応する昇順と降順の混合カラムも使用されていれば、オプティマイザはカラムにインデックスを使用できます:

```
SELECT * FROM t1
ORDER BY key_part1 DESC, key_part2 ASC;
```

key_part1 が降順で **key_part2** が昇順の場合、オプティマイザは (**key_part1**、**key_part2**) のインデックスを使用できます。**key_part1** が昇順で **key_part2** が降順の場合は、これらのカラムにインデックスを使用することもできます (バックワードスキャンを使用)。セクション8.3.13「降順インデックス」を参照してください。

- 次の 2 つのクエリーでは、**key_part1** が定数と比較されます。インデックスは、テーブルスキャンよりもインデックスレンジスキャンの方が安くなるように、**WHERE** 句が選択的である場合に使用されます:

```
SELECT * FROM t1
WHERE key_part1 > constant
ORDER BY key_part1 ASC;
```

```
SELECT * FROM t1
WHERE key_part1 < constant
ORDER BY key_part1 DESC;
```

- 次のクエリーでは、**ORDER BY** は **key_part1** を指定しませんが、選択されたすべての行には定数の **key_part1** 値があるため、インデックスは引き続き使用できます:

```
SELECT * FROM t1
WHERE key_part1 = constant1 AND key_part2 > constant2
ORDER BY key_part2;
```

場合によっては、MySQL はインデックスを使用して **ORDER BY** を解決できませんが、インデックスを使用して **WHERE** 句に一致する行を見つけることができます。例:

- このクエリーでは、異なるインデックスで **ORDER BY** を使用します:

```
SELECT * FROM t1 ORDER BY key1, key2;
```

- クエリーでは、インデックスの連続していない部分で **ORDER BY** を使用します:

```
SELECT * FROM t1 WHERE key2=constant ORDER BY key1_part1, key1_part3;
```

- 行のフェッチに使用されるインデックスは、**ORDER BY** で使用されるインデックスとは異なります:

```
SELECT * FROM t1 WHERE key2=constant ORDER BY key1;
```

- クエリーでは、インデックスカラム名以外の用語を含む式を使用して **ORDER BY** を使用します:

```
SELECT * FROM t1 ORDER BY ABS(key);
SELECT * FROM t1 ORDER BY -key;
```

- クエリーによって多数のテーブルが結合され、**ORDER BY** のカラムは、行の取得に使用される最初の非定数テーブルのすべてではありません。(これは **EXPLAIN** 出力で、**const** 結合型を持たない最初のテーブルです。)
- クエリーの **ORDER BY** 式と **GROUP BY** 式が異なります。
- **ORDER BY** 句で指定されたカラムの接頭辞にのみインデックスがあります。この場合、インデックスを使用してソート順序を完全には解決できません。たとえば、**CHAR(20)** カラムの最初の 10 バイトのみがインデックス付けされている場合、インデックスでは 10 バイトを超える値を区別できず、**filesort** が必要です。
- インデックスには、行は順番に格納されません。たとえば、これは、**MEMORY** テーブルの **HASH** インデックスに当てはまります。

インデックスをソートに使用できるかどうかは、カラムエイリアスの使用によって影響を受けることがあります。カラム `t1.a` にインデックスが設定されているとします。次のステートメントでは、選択リスト内のカラム名は `a` です。`ORDER BY` 内の `a` への参照と同様に、`t1.a` を参照するため、`t1.a` 上のインデックスを使用できます:

```
SELECT a FROM t1 ORDER BY a;
```

次のステートメントでも、選択リスト内のカラム名は `a` ですが、これはエイリアス名です。`ORDER BY` 内の `a` への参照と同様に、`ABS(a)` を参照するため、`t1.a` 上のインデックスは使用できません:

```
SELECT ABS(a) AS a FROM t1 ORDER BY a;
```

次のステートメントでは、`ORDER BY` は、選択リスト内のカラムの名前でない名前を参照しています。ただし、`t1` には `a` という名前のカラムがあるため、`ORDER BY` は `t1.a` を参照し、`t1.a` のインデックスを使用できます。(当然ながら、結果のソート順序は、`ABS(a)` の順序とはまったく異なる可能性があります。)

```
SELECT ABS(a) AS b FROM t1 ORDER BY a;
```

以前は (MySQL 5.7 以下)、`GROUP BY` は特定の条件下で暗黙的にソートされていました。MySQL 8.0 では発生しなくなったため、暗黙的ソートを抑制するために最後に `ORDER BY NULL` を指定する必要はなくなりました (前述のとおり)。ただし、クエリー結果は以前の MySQL バージョンとは異なる場合があります。特定のソート順序を生成するには、`ORDER BY` 句を指定します。

filesort を使用して ORDER BY を満たす

インデックスを使用して `ORDER BY` 句を満たすことができない場合、MySQL はテーブルの行を読み取ってソートする `filesort` 操作を実行します。`filesort` は、クエリーの実行時に追加のソートフェーズを構成します。

MySQL 8.0.12 の時点で、`filesort` 操作のメモリを取得するために、オプティマイザは、MySQL 8.0.12 より前に行われた一定量の `sort_buffer_size` バイトを割り当てるのではなく、`sort_buffer_size` システム変数で指定されたサイズまで必要に応じて増分的にメモリバッファを割り当てます。これにより、ユーザーは小さいソートに過剰なメモリ使用を考慮せずに、大きいソートを高速化するために `sort_buffer_size` を大きい値に設定できます。(この利点は、マルチスレッド `malloc` が弱い Windows での複数の同時ソートでは発生しない場合があります。)

結果セットが大きすぎてメモリに収まらない場合、`filesort` 操作は必要に応じて一時ディスクファイルを使用します。一部のタイプのクエリーは、完全にインメモリ `filesort` 操作に特に適しています。たとえば、オプティマイザは `filesort` を使用して、一時ファイルを使用せずに、次の形式のクエリー (およびサブクエリー) に対する `ORDER BY` 操作をメモリ内で効率的に処理できます:

```
SELECT ... FROM single_table ... ORDER BY non_index_column [DESC] LIMIT [M,]N;
```

このようなクエリーは、より大きな結果セットの少数の行のみを表示する web アプリケーションで一般的です。例:

```
SELECT col1, ... FROM t1 ... ORDER BY name LIMIT 10;  
SELECT col1, ... FROM t1 ... ORDER BY RAND() LIMIT 15;
```

ORDER BY 最適化への影響

`filesort` が使用されていない低速な `ORDER BY` クエリーの場合は、`max_length_for_sort_data` システム変数を `filesort` のトリガーに適した値に下げてください。(この変数の値を著しく高く設定すると、高いディスクアクティビティと低い CPU アクティビティの組み合わせが見られます。) この方法は、MySQL 8.0.20 の前にも適用されます。8.0.20 では、`max_length_for_sort_data` は非推奨になりました。これは、オプティマイザの変更によって廃止され、効果がないためです。

`ORDER BY` 速度を向上するには、MySQL で、追加のソートフェーズではなく、インデックスを使用させることができるかどうかをチェックします。これが不可能な場合は、次の方法を試してください:

- `sort_buffer_size` 変数値を増やします。理想的には、(ディスクへの書き込みおよびマージパスを回避するために) 結果セット全体がソートバッファに収まるように値を十分に大きくする必要があります。

ソートバッファに格納されているカラム値のサイズは、`max_sort_length` システム変数値の影響を受けることを考慮してください。たとえば、タプルに長い文字列カラムの値が格納されていて、`max_sort_length` の値を増やすと、ソートバッファータプルのサイズも増加し、`sort_buffer_size` を増やす必要がある場合があります。

(一時ファイルをマージするための) マージパスの数を監視するには、`Sort_merge_passes` ステータス変数を確認します。

- 一度に読み取られる行が増えるように、`read_rnd_buffer_size` 変数の値を増やします。
- `tmpdir` システム変数を変更して、大量の空き領域のある専用ファイルシステムを指すようにします。変数値には、ラウンドロビン方式で使用される複数のパスをリストできます。この機能を使用して、複数のディレクトリに負荷を分散できます。パスは、Unix ではコロン文字 (:) で区切り、Windows ではセミコロン文字 (;) で区切ります。パスには、同じディスク上の異なるパーティションではなく、異なる物理ディスクにあるファイルシステム内のディレクトリを指定してください。

ORDER BY 実行計画情報使用可能

`EXPLAIN` (セクション8.8.1「`EXPLAIN` によるクエリーの最適化」を参照) では、MySQL がインデックスを使用して `ORDER BY` 句を解決できるかどうかを確認できます:

- `EXPLAIN` 出力の `Extra` カラムに `Using filesort` が含まれていない場合、インデックスが使用され、`filesort` は実行されません。
- `EXPLAIN` 出力の `Extra` カラムに `Using filesort` が含まれている場合、インデックスは使用されず、`filesort` が実行されます。

また、`filesort` が実行されると、オプティマイザのトレース出力に `filesort_summary` ブロックが含まれます。例:

```
"filesort_summary": {
  "rows": 100,
  "examined_rows": 100,
  "number_of_tmp_files": 0,
  "peak_memory_used": 25192,
  "sort_mode": "<sort_key, packed_additional_fields>"
}
```

`peak_memory_used` は、ソート中に一度に使用される最大メモリを示します。これは、`sort_buffer_size` システム変数の値までの値ですが、必ずしも大きくなるとはかぎりません。MySQL 8.0.12 より前の出力では、かわりに `sort_buffer_size` の値を示す `sort_buffer_size` が表示されます。(MySQL 8.0.12 より前では、オプティマイザは常に `sort_buffer_size` バイトをソートバッファに割り当てます。8.0.12 の時点では、オプティマイザは、少量から始まり、必要に応じて `sort_buffer_size` バイトまで、ソートバッファメモリを増分的に割り当てます。)

`sort_mode` 値は、ソートバッファ内のタプルの内容に関する情報を提供します:

- `<sort_key, rowid>`: これは、ソートバッファータプルが、元のテーブル行のソートキー値と行 ID を含むペアであることを示します。タプルはソートキー値でソートされ、行 ID は、テーブルからの行の読み取りに使用されます。
- `<sort_key, additional_fields>`: これは、ソートバッファータプルにソートキー値とクエリーによって参照されるカラムが含まれていることを示します。タプルはソートキー値でソートされ、カラム値は、タプルから直接読み取られます。
- `<sort_key, packed_additional_fields>`: 前のバリエーションと同様ですが、追加のカラムは固定長エンコーディングを使用するかわりに密接にパックされます。

`EXPLAIN` は、オプティマイザがメモリ内で `filesort` を実行するかどうかを区別しません。インメモリ `filesort` の使用は、オプティマイザのトレース出力で確認できます。 `filesort_priority_queue_optimization` を探します。オプティマイザのトレースについては、「[MySQL Internals: Tracing the Optimizer](#)」を参照してください。

8.2.1.17 GROUP BY の最適化

`GROUP BY` 句を満たすもっとも一般的な方法は、テーブル全体をスキャンし、各グループのすべての行が連続する新しい一時テーブルを作成することであり、それにより、この一時テーブルを使用してグループを見つけて、集約関数(ある場合)を適用できます。場合によっては、MySQL はそれよりはるかに優れた処理を実行でき、インデックスアクセスを使用した一時テーブルの作成を回避できます。

`GROUP BY` のインデックスを使用するための最も重要な前提条件は、すべての `GROUP BY` カラムが同じインデックスの属性を参照し、インデックスにそのキーが順番に格納されることです(たとえば、`BTREE` インデックスの場合は

該当しますが、HASH インデックスの場合は該当しません)。一時テーブルの使用をインデックスアクセスに置き換えられるかどうかは、クエリー内でインデックスのどの部分が使用されているか、その部分に指定された条件、および選択された集約関数にもよります。

次のセクションで詳しく説明するように、インデックスアクセスによって GROUP BY クエリーを実行する方法は 2 つあります。最初の方法では、すべての範囲述語 (ある場合) とともにグループ化操作が適用されます。2 つめの方法では、まず範囲スキャンを実行し、次に結果タプルをグループ化します。

- [ルースインデックススキャン](#)
- [タイトインデックススキャン](#)

一部の条件下では、GROUP BY が存在しない場合にも、緩やかなインデックススキャンを使用できます。 [スキャン範囲アクセス方法のスキップ](#) を参照してください。

ルースインデックススキャン

GROUP BY を処理するもっとも効率的な方法は、インデックスを使用してグループ化するカラムを直接取得することです。このアクセスメソッドでは、MySQL はキーが順序付けられている、インデックス型のプロパティーを使用します。(たとえば、BTREE)。このプロパティーにより、インデックス内のすべての WHERE 条件を満たすキーを考慮する必要なく、インデックス内のルックアップグループを使用できます。このアクセス方法では、インデックス内のキーの一部のみが考慮されるため、ルースインデックススキャンと呼ばれます。WHERE 句がない場合、ループインデックススキャンはグループ数と同じ数のキーを読み取ります。これは、すべてのキーの数よりはるかに小さい場合があります。WHERE 句に範囲述語が含まれている場合 ([セクション 8.8.1 「EXPLAIN によるクエリーの最適化」](#) の range 結合タイプの説明を参照)、ルースインデックススキャンは範囲条件を満たす各グループの最初のキーを検索し、可能なかぎり少ない数のキーを再度読み取ります。これは次の条件の下で可能です。

- クエリーが単一テーブルに対するものです。
- GROUP BY はインデックスの左端のプリフィクスを形成するカラムのみを指定し、ほかのカラムは指定しません。(GROUP BY の代わりに、クエリーに DISTINCT 句がある場合、個々のすべての属性がインデックスの左端のプリフィクスを形成するカラムを参照します。)たとえば、テーブル t1 の (c1,c2,c3) にインデックスがある場合、クエリーに GROUP BY c1, c2 が含まれていれば、Loose Index Scan が適用されます。クエリーに GROUP BY c2, c3 (カラムは左端のプリフィクスでない) または GROUP BY c1, c2, c4 (c4 はインデックス内にない) がある場合は適用できません。
- 選択リスト (ある場合) で使用されている集約関数が、MIN() と MAX() だけであり、それらはすべて同じカラムを参照します。カラムはインデックス内にあり、GROUP BY のカラムの直後にある必要があります。
- クエリーで参照された GROUP BY からの部分以外のインデックスの部分は、定数である必要があります (つまり、定数と同等のもので参照されている必要があります) が、MIN() または MAX() 関数の引数を除きます。
- インデックス内のカラムの場合、プリフィクスだけでなく、完全なカラム値にインデックスが設定されている必要があります。たとえば、c1 VARCHAR(20), INDEX (c1(10)) では、インデックスは c1 値の接頭辞のみを使用し、ループインデックススキャンには使用できません。

ループインデックススキャンがクエリーに適用可能な場合、EXPLAIN 出力の Extra カラムに Using index for group-by が表示されます。

テーブル t1(c1,c2,c3,c4) にインデックス idx(c1,c2,c3) があると仮定します。Loose Index Scan アクセス方法は、次のクエリーに使用できます:

```
SELECT c1, c2 FROM t1 GROUP BY c1, c2;
SELECT DISTINCT c1, c2 FROM t1;
SELECT c1, MIN(c2) FROM t1 GROUP BY c1;
SELECT c1, c2 FROM t1 WHERE c1 < const GROUP BY c1, c2;
SELECT MAX(c3), MIN(c3), c1, c2 FROM t1 WHERE c2 > const GROUP BY c1, c2;
SELECT c2 FROM t1 WHERE c1 < const GROUP BY c1, c2;
SELECT c1, c2 FROM t1 WHERE c3 = const GROUP BY c1, c2;
```

次に示す理由により、以下のクエリーはこのクイック選択メソッドで実行できません。

- MIN() または MAX() 以外の集約関数があります。

```
SELECT c1, SUM(c2) FROM t1 GROUP BY c1;
```

- **GROUP BY** 句内のカラムがインデックスの左端のプリフィクスを形成していません。

```
SELECT c1, c2 FROM t1 GROUP BY c2, c3;
```

- クエリーは **GROUP BY** 部分のあとに続くキーの部分を参照し、そこに定数と同等のものはありません。

```
SELECT c1, c3 FROM t1 GROUP BY c1, c2;
```

クエリーに **WHERE c3 = const** が含まれていたため、ループインデックススキャンを使用できました。

Loose Index Scan アクセス方法は、すでにサポートされている **MIN()** および **MAX()** 参照に加えて、選択リストの他の形式の集計関数参照にも適用できます:

- **AVG(DISTINCT)**、**SUM(DISTINCT)**、および **COUNT(DISTINCT)** がサポートされています。 **AVG(DISTINCT)** と **SUM(DISTINCT)** は 1 つの引数をとります。 **COUNT(DISTINCT)** には複数のカラム引数を指定できます。
- クエリーに **GROUP BY** または **DISTINCT** 句があってははいけません。
- 前に説明した Loose Index Scan の制限が引き続き適用されます。

テーブル **t1(c1,c2,c3,c4)** にインデックス **idx(c1,c2,c3)** があると仮定します。 Loose Index Scan アクセス方法は、次のクエリーに使用できます:

```
SELECT COUNT(DISTINCT c1), SUM(DISTINCT c1) FROM t1;
```

```
SELECT COUNT(DISTINCT c1, c2), COUNT(DISTINCT c2, c1) FROM t1;
```

タイトインデックススキャン

密インデックススキャンは、クエリー条件に応じて、全インデックススキャンまたはレンジインデックススキャンのいずれかになります。

ループインデックススキャンの条件が満たされていない場合でも、**GROUP BY** クエリーの一時テーブルの作成を回避できます。 **WHERE** 句に範囲条件がある場合、このメソッドはこれらの条件を満たすキーだけを読み取ります。 そうでない場合は、インデックススキャンを実行します。 このメソッドは、**WHERE** 句で定義された各範囲内のすべてのキーを読み取るか、範囲条件がない場合はインデックス全体をスキャンするため、タイトインデックススキャンと呼ばれます。 厳密なインデックススキャンでは、範囲条件を満たすすべてのキーが検出された後にのみグループ化操作が実行されます。

この方法が機能するには、**GROUP BY** キーの一部の前後に来るキーの一部を参照するクエリーのすべてのカラムに一定の等価条件がある必要があります。 同等条件からの定数は、インデックスの完全なプリフィクスを形成できるように、検索キーの「ギャップ」を埋めます。 これらのインデックスのプリフィクスは、インデックスルックアップに使用できます。 **GROUP BY** の結果にソートが必要で、インデックスの接頭辞である検索キーを形成できる場合、順序付きインデックスの接頭辞で検索するとすべてのキーがすでに順番に取得されるため、MySQL では余分なソート操作も回避されます。

テーブル **t1(c1,c2,c3,c4)** にインデックス **idx(c1,c2,c3)** があると仮定します。 次のクエリーは、前述の Loose Index Scan アクセス方法では機能しませんが、Tight Index Scan アクセス方法では機能します。

- **GROUP BY** にはギャップがありますが、条件 **c2 = 'a'** によってカバーされます。

```
SELECT c1, c2, c3 FROM t1 WHERE c2 = 'a' GROUP BY c1, c3;
```

- **GROUP BY** は、キーの最初の部分から開始されませんが、その部分に対して定数を与える条件があります。

```
SELECT c1, c2, c3 FROM t1 WHERE c1 = 'a' GROUP BY c2, c3;
```

8.2.1.18 DISTINCT の最適化

ORDER BY と組み合わせられた **DISTINCT** では多くの場合に一時テーブルが必要です。

DISTINCT では **GROUP BY** を使用できるため、MySQL が **ORDER BY** または **HAVING** 句内の選択したカラムの部分でないカラムをどのように処理するかを学んでください。 [セクション12.20.3「MySQL での GROUP BY の処理」](#) を参照してください。

ほとんどの場合、`DISTINCT` 句は `GROUP BY` の特殊な例と考えることができます。たとえば、次の 2 つのクエリーは同等です。

```
SELECT DISTINCT c1, c2, c3 FROM t1
WHERE c1 > const;
```

```
SELECT c1, c2, c3 FROM t1
WHERE c1 > const GROUP BY c1, c2, c3;
```

この同等性のため、`GROUP BY` クエリーに適用できる最適化は `DISTINCT` 句のあるクエリーにも適用できます。そのため、`DISTINCT` クエリー最適化の可能性の詳細については、[セクション 8.2.1.17 「GROUP BY の最適化」](#) を参照してください。

`LIMIT row_count` を `DISTINCT` と組み合わせた場合、MySQL は `row_count` 固有の行が見つかるただちに停止します。

クエリーに指定されたすべてのテーブルのカラムを使用しない場合、MySQL は最初の一致が見つかるただちに未使用テーブルのスキャンを停止します。次の例では、`t1` が `t2` の前に使用され (これは、`EXPLAIN` で確認できます)、MySQL は `t2` (`t1` 内の特定の行の) で、最初の行を見つけると、`t2` からの読み取りを停止します。

```
SELECT DISTINCT t1.a FROM t1, t2 where t1.a=t2.a;
```

8.2.1.19 LIMIT クエリーの最適化

結果セットから指定した数の行のみが必要な場合、結果セット全体をフェッチして、余分なデータを破棄するのではなく、クエリーで `LIMIT` 句を使用します。

MySQL は `LIMIT row_count` 句があり `HAVING` 句のないクエリーを最適化することがあります。

- `LIMIT` で少数の行のみを選択すると、MySQL では、通常フルテーブルスキャンを実行するより望ましい特定の場合に、インデックスが使用されます。
- `LIMIT row_count` を `ORDER BY` と組み合わせると、MySQL はソート結果の最初の `row_count` 行を見つけた直後に、結果全体をソートするのではなくソートを停止します。インデックスを使用して順序付けが行われている場合、これはきわめて高速になります。`filesort` を実行する必要がある場合、最初の `row_count` を見つける前に、`LIMIT` 句を使用しないクエリーに一致するすべての行が選択され、それらのほとんどまたはすべてがソートされます。初期の行が見つかったら、MySQL は結果セットの残りをすべてソートしません。

この動作をはっきり示している現象の 1 つは、このセクションで後述するように、`LIMIT` を付けるか付けないかで `ORDER BY` クエリーは異なる順序で行を返す場合があることです。

- `LIMIT row_count` を `DISTINCT` と組み合わせると、MySQL は `row_count` の一意の行を検出するとすぐに停止します。
- 場合によっては、インデックスを順番に読み取る (またはインデックスでソートする) ことで `GROUP BY` を解決し、インデックス値が変更されるまでサマリーを計算できます。この場合、`LIMIT row_count` は不要な `GROUP BY` 値を計算しません。
- MySQL は必要な数の行をクライアントに送信するとただちに、`SQL_CALC_FOUND_ROWS` が使用されていないかぎり、クエリーを中止します。その場合、`SELECT FOUND_ROWS()` を使用して行数を取得できます。[セクション 12.16 「情報関数」](#) を参照してください。
- `LIMIT 0` は迅速に空のセットを返します。これは、クエリーの妥当性のチェックに役立つことがあります。また、結果セットメタデータを使用可能にする MySQL API を使用するアプリケーション内の結果カラムのタイプを取得するためにも使用できます。`mysql` クライアントプログラムでは、`--column-type-info` オプションを使用して結果カラムタイプを表示できます。
- サーバーは、一時テーブルを使用してクエリーを解決する場合、`LIMIT row_count` 句を使用して必要な容量を計算します。
- `ORDER BY` にインデックスが使用されていないが、`LIMIT` 句も存在する場合、オプティマイザはインメモリー `filesort` 操作を使用して、マージファイルの使用を回避し、メモリー内の行をソートできます。

複数の行の `ORDER BY` カラムに同一の値がある場合、サーバーは自由にそれらの行を任意の順序で返しますが、その実行は実行プラン全体によって異なることがあります。言い換えると、それらの行のソート順序は、順序付けされていないカラムに関して決定的ではありません。

実行プランに影響する 1 つの要素は `LIMIT` であるため、`LIMIT` を付けるか付けないかで `ORDER BY` クエリーは異なる順序で行を返すことがあります。`category` カラムによってソートされるが、`id` および `rating` カラムに関して非決定的である次のクエリーを考慮します。

```
mysql> SELECT * FROM ratings ORDER BY category;
```

```
+-----+-----+
| id | category | rating |
+-----+-----+
| 1 | 1 | 4.5 |
| 5 | 1 | 3.2 |
| 3 | 2 | 3.7 |
| 4 | 2 | 3.5 |
| 6 | 2 | 3.5 |
| 2 | 3 | 5.0 |
| 7 | 3 | 2.7 |
+-----+-----+
```

`LIMIT` を含めると、各 `category` 値内の行の順序に影響することがあります。たとえば、これは有効なクエリー結果です。

```
mysql> SELECT * FROM ratings ORDER BY category LIMIT 5;
```

```
+-----+-----+
| id | category | rating |
+-----+-----+
| 1 | 1 | 4.5 |
| 5 | 1 | 3.2 |
| 4 | 2 | 3.5 |
| 3 | 2 | 3.7 |
| 6 | 2 | 3.5 |
+-----+-----+
```

各ケースで、行は `ORDER BY` カラムによってソートされますが、SQL 標準で必要とされるのはこれだけです。

`LIMIT` を使用してもしなくても同じ行順序を確保することが重要な場合は、`ORDER BY` 句に順序を決定的にする追加カラムを含めます。たとえば、`id` 値が一意的な場合、次のようにソートすることで、特定の `category` 値の行を `id` の順序で表示できます：

```
mysql> SELECT * FROM ratings ORDER BY category, id;
```

```
+-----+-----+
| id | category | rating |
+-----+-----+
| 1 | 1 | 4.5 |
| 5 | 1 | 3.2 |
| 3 | 2 | 3.7 |
| 4 | 2 | 3.5 |
| 6 | 2 | 3.5 |
| 2 | 3 | 5.0 |
| 7 | 3 | 2.7 |
+-----+-----+
```

```
mysql> SELECT * FROM ratings ORDER BY category, id LIMIT 5;
```

```
+-----+-----+
| id | category | rating |
+-----+-----+
| 1 | 1 | 4.5 |
| 5 | 1 | 3.2 |
| 3 | 2 | 3.7 |
| 4 | 2 | 3.5 |
| 6 | 2 | 3.5 |
+-----+-----+
```

`ORDER BY` または `GROUP BY` と `LIMIT` 句を含むクエリーの場合、オプティマイザは、クエリーの実行速度を上げるために、順序付けられたインデックスをデフォルトで選択しようとしています。MySQL 8.0.21 より前は、他の最適化を使用する方が高速であっても、この動作をオーバーライドする方法はありませんでした。MySQL 8.0.21 以降では、`optimizer_switch` システム変数 `prefer_ordering_index` フラグを `off` に設定することで、この最適化をオフにできます。

例: まず、次に示すように、テーブル `t` を作成して移入します：

```
# Create and populate a table t:
```



```
mysql> CREATE TABLE t (  
-> id1 BIGINT NOT NULL,  
-> id2 BIGINT NOT NULL,  
-> c1 VARCHAR(50) NOT NULL,  
-> c2 VARCHAR(50) NOT NULL,  
-> PRIMARY KEY (id1),  
-> INDEX i (id2, c1)  
-> );
```

```
# [Insert some rows into table t - not shown]
```

`prefer_ordering_index` フラグが有効になっていることを確認します:

```
mysql> SELECT @@optimizer_switch LIKE '%prefer_ordering_index=on%';  
+-----+  
| @@optimizer_switch LIKE '%prefer_ordering_index=on%' |  
+-----+  
| 1 |  
+-----+
```

次のクエリーには `LIMIT` 句があるため、可能な場合は順序付けされたインデックスを使用する必要があります。この場合、`EXPLAIN` 出力からわかるように、テーブルの主キーが使用されます。

```
mysql> EXPLAIN SELECT c2 FROM t  
-> WHERE id2 > 3  
-> ORDER BY id1 ASC LIMIT 2\G  
***** 1. row *****  
id: 1  
select_type: SIMPLE  
table: t  
partitions: NULL  
type: index  
possible_keys: i  
key: PRIMARY  
key_len: 8  
ref: NULL  
rows: 2  
filtered: 70.00  
Extra: Using where
```

次に、`prefer_ordering_index` フラグを無効にし、同じクエリーを再実行します。今回は、インデックス `i` (`WHERE` 句で使用される `id2` カラムを含む) と `filesort` を使用します:

```
mysql> SET optimizer_switch = "prefer_ordering_index=off";  
  
mysql> EXPLAIN SELECT c2 FROM t  
-> WHERE id2 > 3  
-> ORDER BY id1 ASC LIMIT 2\G  
***** 1. row *****  
id: 1  
select_type: SIMPLE  
table: t  
partitions: NULL  
type: range  
possible_keys: i  
key: i  
key_len: 8  
ref: NULL  
rows: 14  
filtered: 100.00  
Extra: Using index condition; Using filesort
```

[セクション8.9.2「切り替え可能な最適化」](#)も参照してください。

8.2.1.20 関数コールの最適化

MySQL 関数は、内部的に決定論的または非決定論的としてタグ付けされます。関数に決定性がない状態とは、引数の値が固定されていても、呼び出しごとに異なる結果が返されることがある場合です。非決定論的関数の例: `RAND()`、`UUID()`。

関数が非決定論的にタグ付けされている場合、`WHERE` 句でのその関数への参照は、行ごと (あるテーブルから選択する場合) または行の組合せごと (複数テーブル結合から選択する場合) に評価されます。

MySQL では、引数がテーブルのカラムであるか定数値であるかにかかわらず、引数のタイプに基づいて関数を評価するタイミングも決定されます。テーブルのカラムを引数として取る決定的関数は、そのカラムの値が変更されるたびに評価される必要があります。

非決定的関数は、クエリーのパフォーマンスに影響を与える可能性があります。たとえば、一部の最適化を使用できない場合や、より多くのロックが必要になる場合があります。次の説明では、`RAND()` を使用しますが、他の非決定的関数にも適用されます。

テーブル `t` に次の定義があるとします:

```
CREATE TABLE t (id INT NOT NULL PRIMARY KEY, col_a VARCHAR(100));
```

次の 2 つのクエリーについて考えてみます:

```
SELECT * FROM t WHERE id = POW(1,2);  
SELECT * FROM t WHERE id = FLOOR(1 + RAND() * 49);
```

主キーに対する等価比較のため、両方のクエリーで主キー参照が使用されているように見えますが、これは最初のクエリーにのみ当てはまります:

- 定数引数を持つ `POW()` は定数値であり、インデックスルックアップに使用されるため、最初のクエリーでは常に最大 1 行が生成されます。
- 2 番目のクエリーには、非決定的関数 `RAND()` を使用する式が含まれています。これはクエリーでは定数ではありませんが、実際にはテーブル `t` の各行に新しい値があります。したがって、クエリーはテーブルのすべての行を読み取り、各行の述語を評価して、主キーがランダム値と一致するすべての行を出力します。これは、`id` のカラム値および `RAND()` 順序内の値に応じて、ゼロ、1 または複数の行になります。

非決定的影響は、`SELECT` ステートメントに限定されません。次の `UPDATE` ステートメントでは、非決定的関数を使用して、変更する行を選択します:

```
UPDATE t SET col_a = some_expr WHERE id = FLOOR(1 + RAND() * 49);
```

主キーが式と一致する単一の行のみを更新することを意図しています。ただし、`id` カラムの値および `RAND()` 順序の値によっては、ゼロ、1 または複数の行が更新される場合があります。

ここで説明した動作は、パフォーマンスとレプリケーションに影響します:

- 非決定的関数では定数値が生成されないため、オプティマイザは、それ以外の場合には適用可能な戦略 (インデックス参照など) を使用できません。結果はテーブルスキャンである可能性があります。
- `InnoDB` は、一致する行に対して単一行ロックを取得するのではなく、範囲キーロックにエスカレートする場合があります。
- 決定的に実行されない更新は、レプリケーションに対して安全ではありません。

問題は、`RAND()` 関数がテーブルのすべての行に対して 1 回評価されるという事実から発生します。複数の関数の評価を回避するには、次のいずれかの方法を使用します:

- 非決定的関数を含む式を別のステートメントに移動し、値を変数に保存します。元のステートメントで、式を変数への参照に置き換えます。この変数は、オプティマイザで定数値として処理できます:

```
SET @keyval = FLOOR(1 + RAND() * 49);  
UPDATE t SET col_a = some_expr WHERE id = @keyval;
```

- 導出テーブルの変数にランダム値を割り当てます。この方法では、`WHERE` 句での比較で変数を使用する前に、変数に値が一度割り当てられます:

```
UPDATE /*+ NO_MERGE(dt) */ t, (SELECT FLOOR(1 + RAND() * 49) AS r) AS dt  
SET col_a = some_expr WHERE id = dt.r;
```

前述のように、`WHERE` 句の非決定的な式によって最適化が妨げられ、テーブルスキャンが発生する可能性があります。ただし、他の式が決定的である場合は、`WHERE` 句を部分的に最適化できます。例:

```
SELECT * FROM t WHERE partial_key=5 AND some_column=RAND();
```

オプティマイザが `partial_key` を使用して選択された行セットを減らすことができる場合、`RAND()` の実行回数が少なくなり、最適化に対する非決定の影響が低下します。

8.2.1.21 ウィンドウ機能最適化

ウィンドウ関数は、オプティマイザが考慮する戦略に影響します:

- サブクエリーにウィンドウ関数がある場合、サブクエリーの導出テーブルマージは無効になります。サブクエリーは常に実体化されます。
- 準結合は、ウィンドウ関数の最適化には適用できません。これは、ウィンドウ関数を含むことができない `WHERE` および `JOIN ... ON` のサブクエリーに準結合が適用されるためです。
- オプティマイザは同じ順序付け要件を持つ複数のウィンドウを順番に処理するため、最初のウィンドウに続くウィンドウではソートをスキップできます。
- オプティマイザは、単一のステップで評価できるウィンドウをマージしようとしません (たとえば、複数の `OVER` 句に同一のウィンドウ定義が含まれている場合)。回避策は、`WINDOW` 句でウィンドウを定義し、`OVER` 句でウィンドウ名を参照することです。

ウィンドウ関数として使用されない集計関数は、可能なかぎり外側のクエリーで集計されます。たとえば、次のクエリーでは、MySQL は、`COUNT(t1.b)` が `WHERE` 句に配置されているために外部クエリーに存在できないものであることを確認します:

```
SELECT * FROM t1 WHERE t1.a = (SELECT COUNT(t1.b) FROM t2);
```

したがって、MySQL はサブクエリー内で集計し、`t1.b` を定数として扱い、`t2` の行数を返します。

`WHERE` を `HAVING` に置き換えると、エラーが発生します:

```
mysql> SELECT * FROM t1 HAVING t1.a = (SELECT COUNT(t1.b) FROM t2);
ERROR 1140 (42000): In aggregated query without GROUP BY, expression #1
of SELECT list contains nonaggregated column 'test.t1.a'; this is
incompatible with sql_mode=only_full_group_by
```

このエラーは、`COUNT(t1.b)` が `HAVING` に存在し、外部クエリーが集計されるために発生します。

ウィンドウ関数 (ウィンドウ関数として使用される集計関数を含む) には、前述の複雑さはありません。これらは常に、外部クエリーではなく、書き込まれるサブクエリーで集計されます。

ウィンドウ関数の評価は、精度を失わずにウィンドウ操作を計算するかどうかを決定する `windowing_use_high_precision` システム変数の値の影響を受ける可能性があります。デフォルトでは、`windowing_use_high_precision` は有効です。

一部の移動フレーム集計では、逆集計関数を適用して集計から値を削除できます。これにより、パフォーマンスは向上しますが、精度が失われる可能性があります。たとえば、非常に小さい浮動小数点値を非常に大きな値に追加すると、非常に小さい値が大きい値の「非表示」になります。大きい値を後で反転すると、小さい値の効果は失われます。

逆集約による精度の損失は、浮動小数点 (近似値) データ型に対する演算の場合にのみ係数となります。他のタイプの場合、逆集約は安全です。これには、小数部を許可するが正確な値タイプである `DECIMAL` が含まれます。

高速実行のために、安全な場合、MySQL は常に逆集約を使用します:

- 浮動小数点値の場合、逆集約は必ずしも安全ではなく、精度が失われる可能性があります。デフォルトでは、逆集約は回避されます。逆集約は低速ですが、精度は維持されます。速度の安全性を犠牲にすることが許可されている場合は、`windowing_use_high_precision` を無効にして逆集約を許可できます。
- 非浮動小数点データ型の場合、逆集約は常に安全であり、`windowing_use_high_precision` 値に関係なく使用されません。
- `windowing_use_high_precision` は、`MIN()` および `MAX()` には影響せず、いずれの場合も逆集約を使用しません。

`STDDEV_POP()`、`STDDEV_SAMP()`、`VAR_POP()`、`VAR_SAMP()` の分散関数およびそのシノニムの評価では、最適化モードまたはデフォルトモードで評価を行うことができます。最適化モードでは、最後の有効桁の結果が若干異なる

場合があります。このような違いが許容される場合は、`windowing_use_high_precision` を無効にして最適化モードを許可できます。

`EXPLAIN` の場合、ウィンドウ実行計画情報は大きすぎるため、従来の出力形式で表示できません。ウィンドウ情報を表示するには、`EXPLAIN FORMAT=JSON` を使用して `windowing` 要素を探します。

8.2.1.22 行コンストラクタ式の最適化

行コンストラクタを使用すると、複数の値を同時に比較できます。たとえば、次の 2 つのステートメントは意味的に同等です:

```
SELECT * FROM t1 WHERE (column1,column2) = (1,1);
SELECT * FROM t1 WHERE column1 = 1 AND column2 = 1;
```

また、オプティマイザは両方の式を同じ方法で処理します。

行コンストラクタカラムがインデックスの接頭辞をカバーしていない場合、オプティマイザは使用可能なインデックスを使用する可能性が低くなります。`(c1, c2, c3)` で主キーを持つ次のテーブルについて考えてみます:

```
CREATE TABLE t1 (
  c1 INT, c2 INT, c3 INT, c4 CHAR(100),
  PRIMARY KEY(c1,c2,c3)
);
```

このクエリーでは、`WHERE` 句はインデックス内のすべてのカラムを使用します。ただし、行コンストラクタ自体はインデックス接頭辞をカバーしません。その結果、オプティマイザは `c1` (`key_len=4`、つまり `c1` のサイズ) のみを使用します:

```
mysql> EXPLAIN SELECT * FROM t1
  WHERE c1=1 AND (c2,c3) > (1,1))G
***** 1. row *****
  id: 1
  select_type: SIMPLE
  table: t1
  partitions: NULL
  type: ref
possible_keys: PRIMARY
  key: PRIMARY
  key_len: 4
  ref: const
  rows: 3
  filtered: 100.00
  Extra: Using where
```

このような場合、等価の非コンストラクタ式を使用して行コンストラクタ式をリライトすると、より完全なインデックスが使用される可能性があります。指定されたクエリーについて、行コンストラクタおよび同等の非コンストラクタ式は次のとおりです:

```
(c2,c3) > (1,1)
c2 > 1 OR ((c2 = 1) AND (c3 > 1))
```

非コンストラクタ式を使用するようにクエリーをリライトすると、オプティマイザはインデックス内の 3 つのカラムすべて (`key_len=12`) を使用します:

```
mysql> EXPLAIN SELECT * FROM t1
  WHERE c1 = 1 AND (c2 > 1 OR ((c2 = 1) AND (c3 > 1)))G
***** 1. row *****
  id: 1
  select_type: SIMPLE
  table: t1
  partitions: NULL
  type: range
possible_keys: PRIMARY
  key: PRIMARY
  key_len: 12
  ref: NULL
  rows: 3
  filtered: 100.00
  Extra: Using where
```

したがって、より適切な結果を得るには、行コンストラクタと **AND/OR** 式を混在させないでください。一方を使用してください。

特定の条件下では、オプティマイザは行コンストラクタ引数を持つ **IN()** 式に範囲アクセス方法を適用できます。 [行コンストラクタ式の範囲最適化](#) を参照してください。

8.2.1.23 全テーブルスキャンの回避

MySQL が **フルテーブルスキャン** を使用してクエリーを解決する場合、**EXPLAIN** からの出力には **type** カラムに **ALL** と示されます。これは通常は次の条件で発生します。

- テーブルがきわめて小さいため、キールックアップで煩わされるよりもテーブルスキャンを実行する方が速くなります。これは、10 行未満の行や短い行長のテーブルによくあります。
- インデックスが設定されたカラムに対して、**ON** または **WHERE** 句に使用可能な制限がありません。
- インデックスが設定されたカラムと定数値を比較していて、MySQL が (インデックスツリーに基づいて) その定数がテーブルのきわめて大きい部分をカバーしており、テーブルスキャンが高速に行われると計算しました。 [セクション 8.2.1.1 「WHERE 句の最適化」](#) を参照してください。
- 別のカラム経由で、カーディナリティーが低い (多数の行がキー値に一致する) キーを使用しています。この場合、MySQL では、キーを使用することで多くのキー検索が必要になり、テーブルスキャンが高速になると想定しています。

小さいテーブルでは、テーブルスキャンは多くの場合に適切であり、実行の影響は無視できます。大きいテーブルでは、オプティマイザがテーブルスキャンを誤って選択しないように、次の技法を試してください。

- **ANALYZE TABLE tbl_name** を使用して、スキャンされるテーブルのキー分布を更新します。 [セクション 13.7.3.1 「ANALYZE TABLE ステートメント」](#) を参照してください。
- スキャンされるテーブルに **FORCE INDEX** を使用して、MySQL に、テーブルスキャンは指定したインデックスを使用するのと比較して著しく負荷が大きいことを伝えます。

```
SELECT * FROM t1, t2 FORCE INDEX (index_for_column)
WHERE t1.col_name=t2.col_name;
```

[セクション 8.9.4 「インデックスヒント」](#) を参照してください。

- **--max-seeks-for-key=1000** オプションを使用して **mysqld** を開始するか、または **SET max_seeks_for_key=1000** を使用して、オプティマイザに、キースキャンでは 1,000 より多くのキーシークは発生しないと想定するように伝えます。 [セクション 5.1.8 「サーバーシステム変数」](#) を参照してください。

8.2.2 サブクエリー、導出テーブル、ビュー参照および共通テーブル式の最適化

MySQL クエリーオプティマイザには、サブクエリーの評価に使用できる様々な戦略があります:

- **IN**、**= ANY** または **EXISTS** 述語で使用されるサブクエリーの場合、オプティマイザには次の選択肢があります:
 - 準結合
 - 実体化
 - **EXISTS** 戦略
- **NOT IN**、**<> ALL** または **NOT EXISTS** 述語で使用されるサブクエリーの場合、オプティマイザには次の選択肢があります:
 - 実体化
 - **EXISTS** 戦略

導出テーブルの場合、オプティマイザには次の選択肢があります (ビュー参照および共通テーブル式にも適用されます):

- 導出テーブルの外部クエリーブロックへのマージ
- 導出テーブルを内部一時テーブルに実体化

次の説明では、前述の最適化戦略について詳しく説明します。

注記

サブクエリーを使用して単一のテーブルを変更する `UPDATE` ステートメントおよび `DELETE` ステートメントの制限は、オプティマイザが準結合サブクエリーまたは実体化サブクエリーの最適化を使用しないことです。回避策として、サブクエリーではなく結合を使用する複数テーブルの `UPDATE` ステートメントおよび `DELETE` ステートメントとしてリライトしてみてください。

8.2.2.1 準結合変換による IN および EXISTS サブクエリー述語の最適化

準結合は、テーブルのプルアウト、重複の除去、最初の一致、緩やかなスキャン、実体化などの複数の実行戦略を可能にする準備時変換です。オプティマイザは、このセクションで説明するように、準結合戦略を使用してサブクエリーの実行を改善します。

2つのテーブル間の内部結合の場合、結合は、他方のテーブルに一致がある回数だけ、一方のテーブルから1行を返します。ただし、問題によっては、重要な情報は一致の数ではなく、一致があるかどうかだけの場合があります。コースカリキュラムのクラスとクラス名簿(各クラスに登録されている生徒)をそれぞれ一覧表示する `class` と `roster` というテーブルがあるとします。実際に生徒が登録されているクラスを一覧表示するには、次の結合を使用できます。

```
SELECT class.class_num, class.class_name
FROM class
INNER JOIN roster
WHERE class.class_num = roster.class_num;
```

ただし、結果には、登録された生徒ごとに、各クラスが1回ずつ一覧表示されます。ここでの問題では、これは不要な情報の重複です。

`class_num` が `class` テーブルの主キーであると仮定すると、`SELECT DISTINCT` を使用して重複抑制が可能ですが、後で重複を排除するためにのみ、最初に一致するすべての行を生成することは非効率的です。

同じ重複のない結果は、次のサブクエリーを使用して取得できます。

```
SELECT class_num, class_name
FROM class
WHERE class_num IN
(SELECT class_num FROM roster);
```

ここで、オプティマイザは `IN` 句に `roster` テーブルから各クラス番号のインスタンスを1つだけ返すサブクエリーが必要であることを認識できます。この場合、クエリーでは準結合を使用できます。つまり、`roster` の行と一致する `class` の各行のインスタンスを1つのみ返す操作です。

`EXISTS` サブクエリー述語を含む次のステートメントは、`IN` サブクエリー述語を含む前述のステートメントと同等です:

```
SELECT class_num, class_name
FROM class
WHERE EXISTS
(SELECT * FROM roster WHERE class.class_num = roster.class_num);
```

MySQL 8.0.16 以降では、`EXISTS` サブクエリー述語を含むステートメントは、同等の `IN` サブクエリー述語を含むステートメントと同じ準結合変換の対象となります。

MySQL 8.0.17 以降、次のサブクエリーはアンチ結合に変換されます:

- `NOT IN (SELECT ... FROM ...)`
- `NOT EXISTS (SELECT ... FROM ...)`。
- `IN (SELECT ... FROM ...) IS NOT TRUE`

- EXISTS (SELECT ... FROM ...) IS NOT TRUE。
- IN (SELECT ... FROM ...) IS FALSE
- EXISTS (SELECT ... FROM ...) IS FALSE。

つまり、IN (SELECT ... FROM ...) または EXISTS (SELECT ... FROM ...) 形式のサブクエリーの否定は、アンチ結合に変換されます。

アンチ結合は、一致がない行のみを返す操作です。次に示すクエリーについて考えてみます:

```
SELECT class_num, class_name
FROM class
WHERE class_num NOT IN
  (SELECT class_num FROM roster);
```

このクエリーは内部的にアンチ結合 SELECT class_num, class_name FROM class ANTIJOIN roster ON class_num としてリライトされ、roster のどの行とも一致しない class の各行のインスタンスが返されます。つまり、class の各行については、roster で一致が見つかるたびに class の行を破棄できます。

比較対象の式が NULL 値可能な場合、ほとんどの場合、アンチ結合変換は適用できません。このルールの例外は、(... NOT IN (SELECT ...)) IS NOT FALSE とそれに相当する (... IN (SELECT ...)) IS NOT TRUE をアンチ結合に変換できることです。

外部結合および内部結合の構文は外部クエリー仕様で許可され、テーブル参照は実テーブル、導出テーブル、ビュー参照または共通テーブル式です。

MySQL では、サブクエリーを準結合として処理するには、次の基準を満たす必要があります (MySQL 8.0.17 以降では、NOT がサブクエリーを変更する場合はアンチ結合):

- WHERE 句または ON 句の最上位レベルに表示される IN、= ANY または EXISTS 述語の一部である必要があります。AND 式の用語として使用することもできます。例:

```
SELECT ...
FROM ot1, ...
WHERE (oe1, ...) IN
  (SELECT ie1, ... FROM it1, ... WHERE ...);
```

ここで、ot_i と it_i は、クエリーの外側部分と内側部分のテーブルを表し、oe_i と ie_i は、外部テーブルと内部テーブル内のカラムを参照する式を表します。

MySQL 8.0.17 以降では、サブクエリーは NOT、IS [NOT] TRUE または IS [NOT] FALSE によって変更された式の引数にすることもできます。

- それは UNION コンストラクトのない単一の SELECT である必要があります。
- HAVING 句を含めることはできません。
- (明示的または暗黙的にグループ化されているかどうかにかかわらず) 集計関数を含めることはできません。
- LIMIT 句を指定しないでください。
- ステートメントでは、外部クエリーで STRAIGHT_JOIN 結合タイプを使用しないでください。
- STRAIGHT_JOIN 修飾子は指定できません。
- 外部テーブルとおよび内部テーブルの合計数が結合で許可されている最大テーブル数より少なくなければなりません。
- サブクエリーは関連する場合と関連しない場合があります。MySQL 8.0.16 以降では、デコレーションは EXISTS への引数として使用されるサブクエリーの WHERE 句内の簡易関連述語を調べ、IN (SELECT b FROM ...) 内で使用されたかのように最適化できます。簡易関連という用語は、述語が等価述語であり、WHERE 句の唯一の述語である (または AND と組み合されている) こと、および一方のオペランドがサブクエリーで参照されるテーブルのもので、もう一方のオペランドが外部クエリーブロックのものであることを意味します。
- DISTINCT キーワードは使用できますが、無視されます。準結合戦略では、重複削除が自動的に処理されます。

- サブクエリーに集計関数も含まれていないかぎり、**GROUP BY** 句は許可されますが無視されます。
- 順序付けは準結合戦略の評価には関係ないため、**ORDER BY** 句は許可されますが無視されます。

サブクエリーが前述の基準を満たす場合、MySQL は準結合 (MySQL 8.0.17 以降ではアンチ結合) に変換し、次の方法からコストベースの選択を行います:

- サブクエリーを結合に変換するか、テーブルプルアウトを使用して、クエリーをサブクエリーテーブルと外部テーブル間の内部結合として実行します。テーブルプルアウトは、テーブルをサブクエリーから外部クエリーに引き出します。
- 重複の除去: 準結合を結合として実行し、一時テーブルを使用して重複レコードを削除します。
- FirstMatch: 内部テーブルで行の組合せをスキャンし、特定の値グループに複数のインスタンスがある場合は、すべてを戻すのではなく、いずれかを選択します。これはスキャンを「ショートカット」し、不要な行の生成をなくします。
- LooseScan : 各サブクエリー値グループから単一の値を選択できるインデックスを使用して、サブクエリーテーブルをスキャンします。
- 結合の実行に使用されるインデックス付き一時テーブルにサブクエリーを実体化します。インデックスは重複を削除するために使用されます。さらに、インデックスはあとで一時テーブルと外部テーブルを結合する際のルックアップにも使用されることがあります。そうでない場合はテーブルがスキャンされます。実体化の詳細は、[セクション8.2.2.2「実体化を使用したサブクエリーの最適化」](#)を参照してください。

これらの各戦略は、次の `optimizer_switch` システム変数フラグを使用して有効または無効にできます:

- `semijoin` フラグは、準結合を使用するかどうかを制御します。MySQL 8.0.17 以降、これはアンチ結合にも適用されます。
- `semijoin` が有効になっている場合、`firstmatch`、`loosescan`、`duplicateweedout` および `materialization` フラグを使用すると、許可される準結合戦略をより詳細に制御できます。
- `duplicateweedout` 準結合戦略が無効になっている場合は、他のすべての適用可能な戦略も無効にしないかぎり、使用されません。
- `duplicateweedout` が無効になっている場合、最適マイザによって最適ではないクエリー計画が生成されることがあります。これは、最長一致検索中のヒューリスティックプルーニングが原因で発生します。これは、`optimizer_prune_level=0` を設定することで回避できます。

これらのフラグはデフォルトで有効になっています。[セクション8.9.2「切り替え可能な最適化」](#)を参照してください。

最適マイザは、ビューおよび導出テーブルの処理の違いを最小限に抑えます。これは、`STRAIGHT_JOIN` 修飾子を使用するクエリーおよび準結合に変換可能な `IN` サブクエリーを含むビューに影響します。次のクエリーは、処理の変更によって変換が変更されるため、実行計画が異なるため、これを示しています:

```
CREATE VIEW v AS
SELECT *
FROM t1
WHERE a IN (SELECT b
            FROM t2);

SELECT STRAIGHT_JOIN *
FROM t3 JOIN v ON t3.x = v.a;
```

最適マイザはまずビューを参照し、`IN` サブクエリーを準結合に変換してから、ビューを外部クエリーにマージできるかどうかをチェックします。外部クエリーの `STRAIGHT_JOIN` 修飾子によって準結合が妨げられるため、最適マイザはマージを拒否し、実体化テーブルを使用して導出テーブルを評価します。

`EXPLAIN` 出力には、次のような準結合戦略の使用が示されています:

- 拡張 `EXPLAIN` 出力の場合、次の `SHOW WARNINGS` によって表示されるテキストは、準結合構造を表示するリライトされたクエリーを示しています。([セクション8.8.3「拡張 EXPLAIN 出力形式」](#)を参照してください。) このページから、準結合から取得されたテーブルについて理解できます。サブクエリーが準結合に変換された場合は、

サブクエリー述語がなくなり、そのテーブルおよび `WHERE` 句が外部クエリー結合リストおよび `WHERE` 句にマージされたことがわかります。

- 重複の除去のための一時テーブルの使用は、`Extra` カラムの `Start temporary` と `End temporary` によって示されます。プルされず、`Start temporary` および `End temporary` でカバーされる `EXPLAIN` 出力行の範囲内にあるテーブルの一時テーブルには、`rowid` があります。
- `Extra` カラムの `FirstMatch(tbl_name)` は結合のショートカットを示します。
- `Extra` カラムの `LooseScan(m..n)` は `LooseScan` 戦略の使用を示します。`m` と `n` はキーパート番号です。
- 実体化での一時テーブルの使用は、`select_type` 値が `MATERIALIZED` の行と、`table` 値が `<subqueryN>` の行によって示されます。

MySQL 8.0.21 以降では、準結合変換は、単一テーブル `UPDATE`、または `[NOT] IN` または `[NOT] EXISTS` サブクエリー述語を使用する `DELETE` ステートメントへ適用もできます。ただし、ステートメントが `ORDER BY` または `LIMIT` を使用せず、準結合変換がオブティマイザヒントにより、または `optimizer_switch` 設定により許可されている場合です。

8.2.2.2 実体化を使用したサブクエリーの最適化

オブティマイザは実体化を使用して、より効率的なサブクエリー処理を可能にします。実体化は、通常メモリー内に一時テーブルとしてサブクエリー結果を生成することによって、クエリー実行を高速化します。MySQL ははじめてサブクエリー結果を必要としたときに、その結果を一時テーブルに実体化します。あとで結果が必要になったときに、MySQL は再度一時テーブルを参照します。オブティマイザはハッシュインデックスを使用してテーブルをインデックス付けし、高速かつ低コストにルックアップできる場合があります。インデックスには、重複を排除してテーブルを小さくするための一意の値が含まれています。

サブクエリーの実体化では、可能な場合はインメモリー一時テーブルが使用され、テーブルが大きすぎるとディスク上の記憶域にフォールバックします。セクション 8.4.4 「MySQL での内部一時テーブルの使用」を参照してください。

実体化を使用しない場合、オブティマイザは、非相関サブクエリーを相関サブクエリーとして書き換えることがあります。たとえば、次の `IN` サブクエリーは非相関です (`where_condition` には `t2` からのカラムのみが含まれ、`t1` からは含まれません)。

```
SELECT * FROM t1
WHERE t1.a IN (SELECT t2.b FROM t2 WHERE where_condition);
```

オブティマイザはこれを `EXISTS` 相関サブクエリーとして書き換えることがあります。

```
SELECT * FROM t1
WHERE EXISTS (SELECT t2.b FROM t2 WHERE where_condition AND t1.a=t2.b);
```

一時テーブルを使用したサブクエリー実体化により、そのような書き換えを回避し、外部クエリーの行ごとに 1 回ではなく、1 回だけサブクエリーを実行させることができます。

MySQL でサブクエリー実体化を使用するには、`optimizer_switch` システム変数 `materialization` フラグを有効にする必要があります。(セクション 8.9.2 「切り替え可能な最適化」を参照してください。) `materialization` フラグを有効にすると、実体化は、次のユースケースのいずれかに該当する述語に対して、(選択リスト、`WHERE`、`ON`、`GROUP BY`、`HAVING` または `ORDER BY`) 任意の場所に出現するサブクエリー述語に適用されます:

- 外側の式 `oe_i` または内側の式 `ie_i` が `NULL` 可能でない場合に、述語はこの形式になります。`N` は 1 以上です。

```
(oe_1, oe_2, ..., oe_N) [NOT] IN (SELECT ie_1, i_2, ..., ie_N ...)
```

- 単一の外側の式 `oe` と内側の式 `ie` がある場合に、述語はこの形式になります。式は `NULL` 可能にできます。

```
oe [NOT] IN (SELECT ie ...)
```

- 述語は `IN` または `NOT IN` で `UNKNOWN` (`NULL`) の結果は `FALSE` の結果と同じ意味になります。

次の例に、`UNKNOWN` および `FALSE` 述語評価の同等性の要件が、サブクエリー実体化を使用できるかどうかにかどのように影響するかを示します。サブクエリーが非相関になるように、`where_condition` に `t2` からのカラムのみが含まれ、`t1` からは含まれないとします。

このクエリーは実体化の対象になります。

```
SELECT * FROM t1
WHERE t1.a IN (SELECT t2.b FROM t2 WHERE where_condition);
```

ここでは、**IN** 述語が **UNKNOWN** を返すか、**FALSE** を返すかは問題ではありません。どちらも **t1** からの行はクエリー結果に含まれません。

サブクエリーの実体化が使用されていない例は、次のクエリーで、**t2.b** は **NULL** 値可能なカラムです:

```
SELECT * FROM t1
WHERE (t1.a,t1.b) NOT IN (SELECT t2.a,t2.b FROM t2
WHERE where_condition);
```

サブクエリー実体化の使用には、次の制限が適用されます:

- 内部式と外部式の型は一致する必要があります。たとえば、両方の式が整数であるか、両方が小数の場合、オプティマイザは実体化を使用できませんが、一方の式が整数でもう一方が小数の場合は使用できません。
- 内部式は **BLOB** にできません。

クエリーで **EXPLAIN** を使用すると、オプティマイザがサブクエリーの実体化を使用するかどうかわかります:

- 実体化を使用しないクエリー実行と比較して、**select_type** が **DEPENDENT SUBQUERY** から **SUBQUERY** に変更されることがあります。これは、外部行ごとに 1 回実行されるサブクエリーの場合、実体化によってサブクエリーが 1 回だけ実行されるようにできることを示します。
- 拡張 **EXPLAIN** 出力の場合、次の **SHOW WARNINGS** によって表示されるテキストには、**materialize** および **materialized-subquery** が含まれます。

MySQL 8.0.21 以降では、MySQL は、単一テーブル **UPDATE** へ、または **[NOT] IN** または **[NOT] EXISTS** サブクエリー述語を使用する **DELETE** ステートメントへサブクエリー実体化が、ステートメントが **ORDER BY** または **LIMIT** を使用せず、およびサブクエリーの実体化がオプティマイザヒントにより、または **optimizer_switch** 設定により許可されていれば、提供できます。

8.2.2.3 EXISTS 戦略を使用したサブクエリーの最適化

特定の最適化は、**IN** (または **=ANY**) 演算子を使用してサブクエリーの結果をテストする比較に適用できます。このセクションでは、これらの最適化について、特に **NULL** 値が存在する課題に関して説明します。この説明の最後の部分では、オプティマイザの支援方法を示します。

次のようなサブクエリーの比較を考慮します。

```
outer_expr IN (SELECT inner_expr FROM ... WHERE subquery_where)
```

MySQL は「外側から内側に」クエリーを評価します。つまり、まず外側の式 **outer_expr** の値を取得してから、サブクエリーを実行し、それによって生成される行を取得します。

内側の式 **inner_expr** が **outer_expr** と等しい行だけが目的の行であることをサブクエリーに「通知する」ことは、かなり役に立つ最適化です。これを行うには、サブクエリーの **WHERE** 句に適切な等価をプッシュダウンして、より限定的にします。変換された比較は次のようになります:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where AND outer_expr=inner_expr)
```

変換後、MySQL はプッシュダウンされた等価を使用して、サブクエリーを評価するために調査する必要がある行数を制限できます。

より一般的には、**N** 個の値と **N** 値の行を返すサブクエリーとの比較は、同じ変換の対象になります。**oe_i** と **ie_i** が対応する外側と内側の式の値を表す場合、次のサブクエリー比較は:

```
(oe_1, ..., oe_N) IN
(SELECT ie_1, ..., ie_N FROM ... WHERE subquery_where)
```

次のようになります。

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where
```

```
AND oe_1 = ie_1
AND ...
AND oe_N = ie_N)
```

簡単にするために、次の説明では、外部式と内部式の値の単一のペアを想定しています。

前述の「「プッシュダウン」」戦略は、次のいずれかの条件に該当する場合に機能します：

- `outer_expr` と `inner_expr` は `NULL` にできません。
- `NULL` と `FALSE` サブクエリーの結果を区別する必要はありません。サブクエリーが `WHERE` 句の `OR` 式または `AND` 式の一部である場合、MySQL では考慮されないものとみなされます。オプティマイザが `NULL` と `FALSE` サブクエリーの結果を区別する必要がないことに気付いた別のインスタンスは、次の構成です：

```
... WHERE outer_expr IN (subquery)
```

この場合、`IN (subquery)` が `NULL` または `FALSE` を返すかどうかにかかわらず、`WHERE` 句は行を拒否します。

`outer_expr` は `NULL` 以外の値であることがわかっているが、サブクエリーは `outer_expr = inner_expr` となるような行を生成しないものとします。その場合、`outer_expr IN (SELECT ...)` は次のように評価されます。

- `inner_expr` が `NULL` である行を `SELECT` が生成する場合は `NULL`
- `SELECT` が `NULL` 以外の値のみを生成するかまたは何も生成しない場合は `FALSE`

この状況では、`outer_expr = inner_expr` である行を探すアプローチは有効でなくなります。そのような行を探すことは必要ですが、何も見つからない場合には、`inner_expr` が `NULL` となる行も探します。概して言えば、サブクエリーは次のように変換できます：

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where AND
(outer_expr=inner_expr OR inner_expr IS NULL))
```

追加の `IS NULL` 条件を評価する必要性は、MySQL に `ref_or_null` アクセスメソッドがある理由です。

```
mysql> EXPLAIN
SELECT outer_expr IN (SELECT t2.maybe_null_key
FROM t2, t3 WHERE ...)
FROM t1;
***** 1. row *****
id: 1
select_type: PRIMARY
table: t1
...
***** 2. row *****
id: 2
select_type: DEPENDENT SUBQUERY
table: t2
type: ref_or_null
possible_keys: maybe_null_key
key: maybe_null_key
key_len: 5
ref: func
rows: 2
Extra: Using where; Using index
...
```

`unique_subquery` および `index_subquery` サブクエリー固有のアクセスメソッドには「`or NULL`」バリエーションもあります。

追加の `OR ... IS NULL` 条件によってクエリーの実行は多少複雑になり、サブクエリー内の最適化の一部も適用できなくなります。通常これは許容できます。

`outer_expr` が `NULL` になる可能性がある場合、状況ははるかに悪くなります。「不明な値」としての `NULL` の SQL の解釈によると、`NULL IN (SELECT inner_expr ...)` は次のように評価されるはずですが。

- `SELECT` が何らかの行を生成する場合は `NULL`
- `SELECT` が行を生成しない場合は `FALSE`

正しい評価には、`SELECT` がとにかく何らかの行を生成したかどうかを確認できるようにする必要がありますため、`outer_expr = inner_expr` をサブクエリーにプッシュダウンすることはできません。等価をプッシュダウンできないが、多くの実世界のサブクエリーが非常に遅くなるため、これは問題です。

基本的に、`outer_expr` の値に応じて、サブクエリーを実行するさまざまな方法が存在する必要があります。

オプティマイザは速度よりも SQL 準拠を選択するため、`outer_expr` が `NULL` である可能性があります：

- `outer_expr` が `NULL` の場合、次の式を評価するには、`SELECT` を実行して行を生成するかどうかを判断する必要があります：

```
NULL IN (SELECT inner_expr FROM ... WHERE subquery_where)
```

前述の種類と同等にプッシュダウンせずに、ここで元の `SELECT` を実行する必要があります。

- 一方、`outer_expr` が `NULL` でない場合、次の比較が絶対に必要です：

```
outer_expr IN (SELECT inner_expr FROM ... WHERE subquery_where)
```

プッシュダウン条件を使用する次の式に変換する必要があります：

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where AND outer_expr=inner_expr)
```

この変換を行わないと、サブクエリーは遅くなります。

条件をサブクエリーにプッシュダウンするかどうかの問題を解決するために、条件は「trigger」関数内にラップされます。したがって、次の形式の式は：

```
outer_expr IN (SELECT inner_expr FROM ... WHERE subquery_where)
```

次に変換されます：

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where
        AND trigcond(outer_expr=inner_expr))
```

より一般的には、サブクエリーの比較が外側の式と内側の式の複数のペアに基づく場合、変換は次の比較をします。

```
(oe_1, ..., oe_N) IN (SELECT ie_1, ..., ie_N FROM ... WHERE subquery_where)
```

これを次の式に変換します：

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where
        AND trigcond(oe_1=ie_1)
        AND ...
        AND trigcond(oe_N=ie_N)
    )
```

各 `trigcond(X)` は、次の値に評価される特殊な関数です。

- 「リンクされた」外側の式 `oe_i` が `NULL` でない場合は `X`
- 「リンクされた」外側の式 `oe_i` が `NULL` の場合は `TRUE`

注記

トリガー関数は、`CREATE TRIGGER` で作成する種類のトリガーではありません。

`trigcond()` 関数内でラップされる等価は、クエリーオプティマイザのファーストクラス述語ではありません。ほとんどの最適化では、クエリーの実行時にオンまたはオフになる可能性のある述語を処理できないため、`trigcond(X)` をすべて不明な関数であるとみなし、無視します。トリガーされた等価は、次の最適化で使用できます：

- 参照の最適化: `trigcond(X=Y [OR Y IS NULL])` を使用して、`ref`、`eq_ref`、または `ref_or_null` テーブルアクセスを構築できます。
- インデックスルックアップベースのサブクエリー実行エンジン: `trigcond(X=Y)` を使用して、`unique_subquery` または `index_subquery` アクセスを構築できます。

- テーブル条件ジェネレータ: サブクエリーが複数のテーブルの結合である場合、トリガーされた条件はできるだけ早くチェックされます。

オプティマイザがトリガー条件を使用して、何らかの種類のインデックスルックアップベースのアクセスを作成する場合 (上記リストの最初の 2 項目に関して)、条件がオフである場合のフォールバック戦略が必要です。このフォールバック戦略は常に同じで、フルテーブルスキャンを実行します。EXPLAIN の出力で、フォールバックは **Extra** カラムに **Full scan on NULL key** と表示されます。

```
mysql> EXPLAIN SELECT t1.col1,
  t1.col1 IN (SELECT t2.key1 FROM t2 WHERE t2.col2=t1.col2) FROM t1\G
***** 1. row *****
  id: 1
 select_type: PRIMARY
  table: t1
  ...
***** 2. row *****
  id: 2
 select_type: DEPENDENT SUBQUERY
  table: t2
  type: index_subquery
 possible_keys: key1
  key: key1
  key_len: 5
  ref: func
  rows: 2
  Extra: Using where; Full scan on NULL key
```

EXPLAIN の後に SHOW WARNINGS を実行すると、トリガーされた条件が表示されます:

```
***** 1. row *****
Level: Note
Code: 1003
Message: select `test`.`t1`.`col1` AS `col1`,
  <in_optimizer>(`test`.`t1`.`col1`,
  <exists>(<index_lookup>(<cache>(`test`.`t1`.`col1`) in t2
  on key1 checking NULL
  where (`test`.`t2`.`col2` = `test`.`t1`.`col2`) having
  trigcond(<is_not_null_test>(`test`.`t2`.`key1`)))) AS
  `t1.col1 IN (select t2.key1 from t2 where t2.col2=t1.col2)`
  from `test`.`t1`
```

トリガー条件を使用すると、パフォーマンスに多少の影響があります。現在 **NULL IN (SELECT ...)** 式では、以前に実行されなかった (遅い) フルテーブルスキャンが行われる可能性があります。これは、正しい結果を得るために支払われる価格です (トリガー条件戦略の目的は、速度ではなくコンプライアンスを向上させることです)。

複数テーブルサブクエリーの場合、結合オプティマイザは外部式が **NULL** の場合に最適化しないため、**NULL IN (SELECT ...)** の実行は特に遅くなります。それは、左辺が **NULL** の場合のサブクエリーの評価はめったにないものと想定しています (そうでないことを示す統計があっても)。一方、外側の式が **NULL** になる可能性があっても実際にそうなることがない場合、パフォーマンスの低下はありません。

クエリーオプティマイザがクエリーをより効率的に実行できるようにするには、次の提案を使用します:

- カラムが実際に **NOT NULL** である場合は、そのように宣言します。これは、カラムの条件テストを簡略化することでオプティマイザの他の側面にも役立ちます。
- **NULL** と **FALSE** サブクエリーの結果を区別する必要がない場合は、実行速度の低下を簡単に回避できます。次のような比較を置き換えます。

```
outer_expr [NOT] IN (SELECT inner_expr FROM ...)
```

次の式で:

```
(outer_expr IS NOT NULL) AND (outer_expr [NOT] IN (SELECT inner_expr FROM ...))
```

式の結果が明らかになるとすぐに MySQL が **AND** 部分の評価を停止するため、**NULL IN (SELECT ...)** は評価されません。

別のリライトも可能です:

```
[NOT] EXISTS (SELECT inner_expr FROM ...
WHERE inner_expr=outer_expr)
```

`optimizer_switch` システム変数の `subquery_materialization_cost_based` フラグを使用すると、サブクエリー実体化と `IN` から `EXISTS` へのサブクエリー変換の選択を制御できます。 [セクション8.9.2「切り替え可能な最適化」](#) を参照してください。

8.2.2.4 マージまたは実体化を使用した導出テーブル、ビュー参照および共通テーブル式の最適化

オプティマイザは、次の 2 つの戦略を使用して導出テーブル参照を処理できます (ビュー参照および共通テーブル式にも適用されます):

- 導出テーブルの外部クエリーブロックへのマージ
- 導出テーブルを内部一時テーブルに実体化

例 1:

```
SELECT * FROM (SELECT * FROM t1) AS derived_t1;
```

導出テーブル `derived_t1` のマージでは、そのクエリーは次のように実行されます:

```
SELECT * FROM t1;
```

例 2:

```
SELECT *
FROM t1 JOIN (SELECT t2.f1 FROM t2) AS derived_t2 ON t1.f2=derived_t2.f1
WHERE t1.f1 > 0;
```

導出テーブル `derived_t2` のマージでは、そのクエリーは次のように実行されます:

```
SELECT t1.*, t2.f1
FROM t1 JOIN t2 ON t1.f2=t2.f1
WHERE t1.f1 > 0;
```

実体化では、`derived_t1` と `derived_t2` はそれぞれ、それぞれのクエリー内で個別のテーブルとして扱われます。

オプティマイザは、導出テーブル、ビュー参照および共通テーブル式を同様に処理: これにより、可能なかぎり不要な実体化が回避され、外部クエリーから導出テーブルへの条件のプッシュダウンが可能になり、より効率的な実行計画が生成されます。(例については、[セクション8.2.2.2「実体化を使用したサブクエリーの最適化」](#) を参照してください。)

マージによって 61 を超える実テーブルを参照する外部クエリーブロックが生成される場合、オプティマイザはかわりに実体化を選択します。

次の条件がすべて満たされている場合、オプティマイザは導出テーブルまたはビュー参照の `ORDER BY` 句を外部クエリーブロックに伝播します:

- 外部クエリーはグループ化または集計されません。
- 外部クエリーでは、`DISTINCT`、`HAVING` または `ORDER BY` は指定されません。
- 外部クエリーでは、この導出テーブルまたはビュー参照が `FROM` 句の唯一のソースとして使用されます。

それ以外の場合、オプティマイザは `ORDER BY` 句を無視します。

オプティマイザが導出テーブル、ビュー参照および共通テーブル式を外部クエリーブロックにマージしようとするかどうかに影響を与えるために、次の方法を使用できます:

- `MERGE` および `NO_MERGE` オプティマイザヒントを使用できます。マージを妨げる他のルールがないことを前提としています。 [セクション8.9.3「オプティマイザヒント」](#) を参照してください。
- 同様に、`optimizer_switch` システム変数の `derived_merge` フラグを使用できます。 [セクション8.9.2「切り替え可能な最適化」](#) を参照してください。デフォルトでは、このフラグはマージを許可するように有効になっています。フラグを無効にすると、マージが回避され、`ER_UPDATE_TABLE_USED` エラーが回避されます。

`derived_merge` フラグは、`ALGORITHM` 句を含まないビューにも適用されます。したがって、サブクエリーと同等の式を使用するビュー参照に対して `ER_UPDATE_TABLE_USED` エラーが発生した場合、ビュー定義に `ALGORITHM=TEMPTABLE` を追加するとマージが回避され、`derived_merge` 値よりも優先されます。

- マージを妨げる構造体はサブクエリーで使用することでマージを無効にできますが、実体化への影響では明示的ではありません。マージが行われないようにする構成は、導出テーブル、共通テーブル式およびビュー参照と同じです:
 - 集計関数またはウィンドウ関数 (`SUM()`, `MIN()`, `MAX()`, `COUNT()` など)
 - `DISTINCT`
 - `GROUP BY`
 - `HAVING`
 - `LIMIT`
 - `UNION` または `UNION ALL`
 - 選択リストのサブクエリー
 - ユーザー変数への割当て
 - リテラル値のみを参照します (この場合、基礎となるテーブルはありません)

最適化が導出テーブルのマージではなく実体化戦略を選択した場合、クエリーは次のように処理されます:

- 最適化は、クエリーの実行中にその内容が必要になるまで、導出テーブルの実体化を延期します。これにより、実体化の遅延によってパフォーマンスが向上する可能性があるため、パフォーマンスが向上します。導出テーブルの結果を別のテーブルに結合するクエリーについて考えます: 最適化が最初に他のテーブルを処理し、行を戻さないことが判明した場合、結合をさらに実行する必要はなく、最適化は導出テーブルの実体化を完全にスキップできます。
- クエリー実行中に、最適化は派生テーブルにインデックスを追加して、そこからの行の取得を高速化できます。

導出テーブルを含む `SELECT` クエリーについて、次の `EXPLAIN` ステートメントを考えてみます:

```
EXPLAIN SELECT * FROM (SELECT * FROM t1) AS derived_t1;
```

最適化は、`SELECT` の実行中に結果が必要になるまで導出テーブルを遅延させることで、導出テーブルの実体化を回避します。この場合、(`EXPLAIN` ステートメントで発生するため) クエリーは実行されないため、結果は必要ありません。

実行されるクエリーの場合でも、導出テーブルの実体化の遅延により、最適化は実体化を完全に回避できます。これが発生すると、クエリー実行は実体化の実行に必要な時間が短縮されます。導出テーブルの結果を別のテーブルに結合する次のクエリーについて考えてみます:

```
SELECT *  
FROM t1 JOIN (SELECT t2.f1 FROM t2) AS derived_t2  
ON t1.f2=derived_t2.f1  
WHERE t1.f1 > 0;
```

最適化が最初に `t1` を処理し、`WHERE` 句で空の結果が生成される場合、結合は空である必要があり、導出テーブルを実体化する必要はありません。

導出テーブルに実体化が必要な場合は、最適化によって実体化ビューにインデックスが追加され、アクセスが高速化される可能性があります。このようなインデックスによってテーブルへの `ref` アクセスが可能になると、クエリーの実行中に読み取られるデータ量を大幅に削減できます。次のクエリーを考慮してください。

```
SELECT *  
FROM t1 JOIN (SELECT DISTINCT f1 FROM t2) AS derived_t2  
ON t1.f1=derived_t2.f1;
```

オプティマイザは、コストが最も低い実行計画に対して `ref` アクセスを使用できるようにする場合、`derived_t2` からコラム `f1` に対するインデックスを構成します。インデックスを追加した後、オプティマイザは実体化導出テーブルをインデックス付きの通常のテーブルと同様に処理でき、生成されたインデックスと同様の利点があります。インデックス作成のオーバーヘッドは、インデックスを使用しないクエリー実行のコストと比較して無視できます。`ref` アクセスによって他のアクセス方法よりコストが高くなる場合、オプティマイザはインデックスを作成せず、何も失われません。

オプティマイザトレース出力の場合、マージされた導出テーブルまたはビュー参照はノードとして表示されません。最上位のクエリー計画には、基礎となるテーブルのみが表示されます。

導出テーブルの実体化に関して正しい記述は、共通テーブル式 (CTE) についても当てはまります。さらに、CTE には特に次の考慮事項があります。

CTE がクエリーによって実体化されている場合、クエリーが複数回参照していても、その CTE はクエリーに対して 1 回実体化されます。

再帰 CTE は常に実体化されます。

CTE が実体化されている場合、オプティマイザは、トップレベルのステートメントによる CTE へのアクセスを高速化できると予想すると、関連するインデックスを自動的に追加します。これは導出テーブルの自動インデックス付けと似ていますが、CTE が複数回参照される場合、オプティマイザは複数のインデックスを作成して、各参照によるアクセスを最適な方法で高速化できます。

CTE には、`MERGE` および `NO_MERGE` オプティマイザヒントを適用できます。トップレベルのステートメントの各 CTE 参照には独自のヒントを指定でき、CTE 参照を選択的にマージまたは実体化できます。次のステートメントは、ヒントを使用して、`cte1` をマージし、`cte2` を実体化する必要があることを示します：

```
WITH
  cte1 AS (SELECT a, b FROM table1),
  cte2 AS (SELECT c, d FROM table2)
SELECT /*+ MERGE(cte1) NO_MERGE(cte2) */ cte1.b, cte2.d
FROM cte1 JOIN cte2
WHERE cte1.a = cte2.c;
```

`CREATE VIEW` の `ALGORITHM` 句は、ビュー定義の `SELECT` ステートメントの前にある `WITH` 句の実体化には影響しません。次のステートメントがあるとします。

```
CREATE ALGORITHM={TEMPTABLE|MERGE} VIEW v1 AS WITH ... SELECT ...
```

`ALGORITHM` 値は、`WITH` 句ではなく、`SELECT` の実体化にのみ影響します。

MySQL 8.0.16 より前では、`internal_tmp_disk_storage_engine=MYISAM` でディスク上の一時テーブルを使用して CTE を実体化しようとすると、CTE の場合、ディスク上の内部一時テーブルに使用されるストレージエンジンを `MyISAM` にできなかったため、エラーが発生しました。MySQL 8.0.16 以降、`TempTable` ではディスク上の内部一時テーブルに `InnoDB` が常に使用されるようになったため、これは問題ではなくなりました。

前述のように、CTE(実体化されている場合) は、複数回参照されていても実体化されます。一時的な実体化を示すために、オプティマイザトレース出力には、`creating_tmp_table` のオカレンスと `reusing_tmp_table` のオカレンスが含まれます。

CTE は、`materialized_from_subquery` ノードが参照に従う導出テーブルに似ています。これは、複数回参照される CTE に当てはまるため、`materialized_from_subquery` ノードの複製はありません(これにより、サブクエリーが複数回実行され、不必要に冗長な出力が生成されるインプレッションが提供されます)。CTE への参照には、サブクエリー計画の説明を含む完全な `materialized_from_subquery` ノードのみが含まれます。その他の参照では、`materialized_from_subquery` ノードが削減されます。`TRADITIONAL` 形式の `EXPLAIN` 出力にも同じ考えが適用されます: 他の参照のサブクエリーは表示されません。

8.2.2.5 導出条件プッシュダウン最適化

MySQL 8.0.22 以降では、適格なサブクエリーの導出条件プッシュダウンがサポートされています。`SELECT * FROM (SELECT i, j FROM t1) AS dt WHERE i > constant` などのクエリーでは、多くの場合、外部 `WHERE` 条件を導出テーブルにプッシュダウンできます(この場合、`SELECT * FROM (SELECT i, j FROM t1 WHERE i > constant) AS`

dt になります)。導出テーブルを外部クエリーにマージできない場合 (導出テーブルで集計が使用されている場合など)、外部 `WHERE` 条件を導出テーブルにプッシュダウンすると、処理する必要がある行数が減り、クエリーの実行が高速化されます。

注記

MySQL 8.0.22 より前では、導出テーブルが実体化されているがマージされていない場合、MySQL はテーブル全体を実体化し、結果のすべての行を `WHERE` 条件で修飾していました。これは、導出条件プッシュダウンが有効になっていない場合や、なんらかの理由で採用できない場合にも当てはまります。

次の状況では、外部 `WHERE` 条件を導出実体化テーブルにプッシュダウンできます：

- 導出テーブルで集計関数またはウィンドウ関数を使用されていない場合は、外部 `WHERE` 条件を直接プッシュダウンできます。これには、`AND`、`OR` またはその両方と結合された複数の述語を持つ `WHERE` 条件が含まれます。

たとえば、クエリー `SELECT * FROM (SELECT f1, f2 FROM t1) AS dt WHERE f1 < 3 AND f2 > 11` は `SELECT f1, f2 FROM (SELECT f1, f2 FROM t1 WHERE f1 < 3 AND f2 > 11) AS dt` としてリライトされます。

- 導出テーブルに `GROUP BY` があり、ウィンドウ関数を使用しない場合、`GROUP BY` の一部ではない 1 つ以上の列を参照する外部 `WHERE` 条件を `HAVING` 条件として導出テーブルにプッシュダウンできます。

たとえば、`SELECT * FROM (SELECT i, j, SUM(k) AS sum FROM t1 GROUP BY i, j) AS dt WHERE sum > 100` は、導出条件プッシュダウンに従って `SELECT * FROM (SELECT i, j, SUM(k) AS sum FROM t1 GROUP BY i, j HAVING sum > 100) AS dt` としてリライトされます。

- 導出テーブルで `GROUP BY` が使用され、外部 `WHERE` 条件の列が `GROUP BY` 列の場合、これらの列を参照する `WHERE` 条件を導出テーブルに直接プッシュダウンできます。

たとえば、クエリー `SELECT * FROM (SELECT i, j, SUM(k) AS sum FROM t1 GROUP BY i, j) AS dt WHERE i > 10` は `SELECT * FROM (SELECT i, j, SUM(k) AS sum FROM t1 WHERE i > 10 GROUP BY i, j) AS dt` としてリライトされます。

外部 `WHERE` 条件に、`GROUP BY` の一部である列を参照する述語とそうでない列を参照する述語がある場合、前のソートの述語は `WHERE` 条件としてプッシュダウンされ、後者のタイプの述語は `HAVING` 条件としてプッシュダウンされます。たとえば、クエリー `SELECT * FROM (SELECT i, j, SUM(k) AS sum FROM t1 GROUP BY i, j) AS dt WHERE i > 10 AND sum > 100` では、外部 `WHERE` 句の述語 `i > 10` は `GROUP BY` 列を参照しますが、述語 `sum > 100` は `GROUP BY` 列を参照しません。このため、導出テーブルプッシュダウン最適化によって、クエリーは次に示すような方法でリライトされます：

```
SELECT * FROM (
  SELECT i, j, SUM(k) AS sum FROM t1
  WHERE i > 10
  GROUP BY i, j
  HAVING sum > 100
) AS dt;
```

導出条件プッシュダウンを有効にするには、`optimizer_switch` システム変数の `derived_condition_pushdown` フラグ (このリリースで追加) を `on` (デフォルト設定) に設定する必要があります。この最適化が `optimizer_switch` によって無効化されている場合は、`DERIVED_CONDITION_PUSHDOWN` オプティマイザヒントを使用して特定のクエリーに対して有効化できます。特定のクエリーの最適化を無効にするには、`NO_DERIVED_CONDITION_PUSHDOWN` オプティマイザヒントを使用します。

導出テーブル条件プッシュダウン最適化には、次の制限事項および制限事項が適用されます：

- 導出テーブルに `UNION` が含まれている場合、最適化は使用できません。
- 導出テーブルでは `LIMIT` 句を使用できません。
- サブクエリーを含む条件はプッシュダウンできません。
- 導出テーブルが外部結合の内部テーブルである場合、最適化は使用できません。
- 実体化導出テーブルが共通テーブル式である場合、条件が複数回参照されても条件はプッシュされません。

- 条件が `derived_column > ?` 形式の場合は、パラメータを使用して条件をプッシュダウンできます。外部 `WHERE` 条件の導出カラムが、基礎となる導出テーブルに `?` を持つ式である場合、この条件はプッシュダウンできません。

8.2.3 INFORMATION_SCHEMA クエリーの最適化

データベースを監視するアプリケーションでは、`INFORMATION_SCHEMA` テーブルを頻繁に使用できます。これらのテーブルに対するクエリーを最も効率的に記述するには、次の一般的なガイドラインを使用します:

- データディクショナリテーブルのビューである `INFORMATION_SCHEMA` テーブルのみのクエリーを試行します。
- 静的メタデータのみのクエリーを試行してください。動的メタデータの列を選択するか、取得条件を静的メタデータとともに使用すると、動的メタデータを処理するためのオーバーヘッドが増加します。

注記

`INFORMATION_SCHEMA` クエリーでのデータベース名とテーブル名の比較動作は、予想とは異なる場合があります。詳細は、[セクション10.8.7「INFORMATION_SCHEMA 検索での照合の使用」](#)を参照してください。

これらの `INFORMATION_SCHEMA` テーブルはデータディクショナリテーブルのビューとして実装されるため、これらのテーブルに対するクエリーではデータディクショナリから情報が取得されます:

```
CHARACTER_SETS
CHECK_CONSTRAINTS
COLLATIONS
COLLATION_CHARACTER_SET_APPLICABILITY
COLUMNS
EVENTS
FILES
INNODB_COLUMNS
INNODB_DATAFILES
INNODB_FIELDS
INNODB_FOREIGN
INNODB_FOREIGN_COLS
INNODB_INDEXES
INNODB_TABLES
INNODB_TABLESPACES
INNODB_TABLESPACES_BRIEF
INNODB_TABLESTATS
KEY_COLUMN_USAGE
PARAMETERS
PARTITIONS
REFERENTIAL_CONSTRAINTS
RESOURCE_GROUPS
ROUTINES
SCHEMATA
STATISTICS
TABLES
TABLE_CONSTRAINTS
TRIGGERS
VIEWS
VIEW_ROUTINE_USAGE
VIEW_TABLE_USAGE
```

一部のタイプの値は、非ビューの `INFORMATION_SCHEMA` テーブルの場合でも、データディクショナリからの参照によって取得されます。これには、データベース名、テーブル名、テーブルタイプ、ストレージエンジンなどの値が含まれます。

一部の `INFORMATION_SCHEMA` テーブルには、テーブル統計を提供する列が含まれています:

```
STATISTICS.CARDINALITY
TABLES.AUTO_INCREMENT
TABLES.AVG_ROW_LENGTH
TABLES.CHECKSUM
TABLES.CHECK_TIME
TABLES.CREATE_TIME
TABLES.DATA_FREE
TABLES.DATA_LENGTH
TABLES.INDEX_LENGTH
```



```
TABLES.MAX_DATA_LENGTH  
TABLES.TABLE_ROWS  
TABLES.UPDATE_TIME
```

これらのカラムは、動的テーブルメタデータ、つまりテーブルの内容の変更に応じて変更される情報を表します。

デフォルトでは、MySQL は、カラムのクエリー時に `mysql.index_stats` および `mysql.table_stats` デイクシヨナリテーブルからこれらのカラムのキャッシュされた値を取得します。これは、ストレージエンジンから統計を直接取得するよりも効率的です。キャッシュされた統計が使用できないか、期限切れになっている場合、MySQL はストレージエンジンから最新の統計を取得し、`mysql.index_stats` および `mysql.table_stats` デイクシヨナリテーブルにキャッシュします。後続のクエリーでは、キャッシュされた統計が期限切れになるまで、キャッシュされた統計が取得されます。

`information_schema_stats_expiry` セッション変数は、キャッシュされた統計が期限切れになるまでの期間を定義します。デフォルトは 86400 秒 (24 時間) ですが、期間は 1 年まで延長できます。

特定のテーブルのキャッシュされた値をいつでも更新するには、`ANALYZE TABLE` を使用します。

次の場合、統計カラムのクエリーでは、`mysql.index_stats` および `mysql.table_stats` デイクシヨナリテーブルの統計は格納または更新されません:

- キャッシュされた統計が失効していない場合。
- `information_schema_stats_expiry` が 0 に設定されている場合。
- サーバーが `read_only`, `super_read_only`, `transaction_read_only` または `innodb_read_only` モードで起動されたとき。
- クエリーでパフォーマンススキーマデータもフェッチされる場合。

`information_schema_stats_expiry` はセッション変数であり、各クライアントセッションで独自の有効期限値を定義できます。ストレージエンジンから取得され、あるセッションによってキャッシュされた統計は、ほかのセッションで使用できます。

注記

`innodb_read_only` システム変数が有効になっている場合、InnoDB を使用するデータデイクシヨナリの統計テーブルを更新できないため、`ANALYZE TABLE` が失敗することがあります。キー分散を更新する `ANALYZE TABLE` 操作では、操作によってテーブル自体が更新された場合でも (MyISAM テーブルの場合など)、障害が発生する可能性があります。更新された分散統計を取得するには、`information_schema_stats_expiry=0` を設定します。

データデイクシヨナリテーブルのビューとして実装された `INFORMATION_SCHEMA` テーブルの場合、基礎となるデータデイクシヨナリテーブルのインデックスを使用すると、オプティマイザで効率的なクエリー実行計画を作成できます。オプティマイザによる選択を確認するには、`EXPLAIN` を使用します。サーバーが `INFORMATION_SCHEMA` クエリーを実行するために使用するクエリーも表示するには、`EXPLAIN` の直後に `SHOW WARNINGS` を使用します。

`utf8mb4` 文字セットの照合順序を識別する次のステートメントについて考えてみます:

```
mysql> SELECT COLLATION_NAME  
FROM INFORMATION_SCHEMA.COLLATION_CHARACTER_SET_APPLICABILITY  
WHERE CHARACTER_SET_NAME = 'utf8mb4';  
+-----+  
| COLLATION_NAME |  
+-----+  
| utf8mb4_general_ci |  
| utf8mb4_bin |  
| utf8mb4_unicode_ci |  
| utf8mb4_icelandic_ci |  
| utf8mb4_latvian_ci |  
| utf8mb4_romanian_ci |  
| utf8mb4_slovenian_ci |  
| ...
```

サーバーはこのステートメントをどのように処理しますか。確認するには、`EXPLAIN` を使用します:

```
mysql> EXPLAIN SELECT COLLATION_NAME
```

```
FROM INFORMATION_SCHEMA.COLLATION_CHARACTER_SET_APPLICABILITY
WHERE CHARACTER_SET_NAME = 'utf8mb4'G
***** 1. row *****
  id: 1
select_type: SIMPLE
  table: cs
  partitions: NULL
  type: const
possible_keys: PRIMARY,name
  key: name
  key_len: 194
  ref: const
  rows: 1
  filtered: 100.00
  Extra: Using index
***** 2. row *****
  id: 1
select_type: SIMPLE
  table: col
  partitions: NULL
  type: ref
possible_keys: character_set_id
  key: character_set_id
  key_len: 8
  ref: const
  rows: 68
  filtered: 100.00
  Extra: NULL
2 rows in set, 1 warning (0.01 sec)
```

そのステートメントの静的化に使用されたクエリーを表示するには、[SHOW WARNINGS](#) を使用します:

```
mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Note
Code: 1003
Message: /* select#1 */ select `mysql`.`col`.`name` AS `COLLATION_NAME`
        from `mysql`.`character_sets` `cs`
        join `mysql`.`collations` `col`
        where ((`mysql`.`col`.`character_set_id` = '45')
        and ('utf8mb4' = 'utf8mb4'))
```

[SHOW WARNINGS](#) で示されているように、サーバーは、[mysql](#) システムデータベースの [character_sets](#) および [collations](#) データディクショナリテーブルに対するクエリーとして [COLLATION_CHARACTER_SET_APPLICABILITY](#) に対するクエリーを処理します。

8.2.4 パフォーマンススキーマクエリーの最適化

データベースを監視するアプリケーションでは、「パフォーマンススキーマ」テーブルを頻繁に使用できます。これらのテーブルに対するクエリーを最も効率的に記述するには、インデックスを利用します。たとえば、インデックス付けされたカラムの特定の値との比較に基づいて、取得される行を制限する [WHERE](#) 句を含めます。

「最もパフォーマンスの高いスキーマ」テーブルにはインデックスがあります。通常は少数の行を含むテーブルではないが、頻繁にクエリーが行われる可能性が低いテーブル。パフォーマンススキーマインデックスを使用すると、オプティマイザは全テーブルスキャン以外の実行計画にアクセスできます。これらのインデックスは、それらのテーブルを使用する [sys](#) スキーマビューなど、関連するオブジェクトのパフォーマンスも向上します。

特定の「パフォーマンススキーマ」テーブルにインデックスがあるかどうかとその内容を確認するには、[SHOW INDEX](#) または [SHOW CREATE TABLE](#) を使用します:

```
mysql> SHOW INDEX FROM performance_schema.accounts\G
***** 1. row *****
  Table: accounts
  Non_unique: 0
  Key_name: ACCOUNT
  Seq_in_index: 1
  Column_name: USER
  Collation: NULL
  Cardinality: NULL
  Sub_part: NULL
```

```
Packed: NULL
Null: YES
Index_type: HASH
Comment:
Index_comment:
Visible: YES
***** 2. row *****
Table: accounts
Non_unique: 0
Key_name: ACCOUNT
Seq_in_index: 2
Column_name: HOST
Collation: NULL
Cardinality: NULL
Sub_part: NULL
Packed: NULL
Null: YES
Index_type: HASH
Comment:
Index_comment:
Visible: YES
```

```
mysql> SHOW CREATE TABLE performance_schema.rwlock_instances\G
***** 1. row *****
Table: rwlock_instances
Create Table: CREATE TABLE `rwlock_instances` (
  `NAME` varchar(128) NOT NULL,
  `OBJECT_INSTANCE_BEGIN` bigint(20) unsigned NOT NULL,
  `WRITE_LOCKED_BY_THREAD_ID` bigint(20) unsigned DEFAULT NULL,
  `READ_LOCKED_BY_COUNT` int(10) unsigned NOT NULL,
  PRIMARY KEY (`OBJECT_INSTANCE_BEGIN`),
  KEY `NAME` (`NAME`),
  KEY `WRITE_LOCKED_BY_THREAD_ID` (`WRITE_LOCKED_BY_THREAD_ID`)
) ENGINE=PERFORMANCE_SCHEMA DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

パフォーマンススキーマクエリーの実行計画およびインデックスを使用するかどうかを確認するには、**EXPLAIN** を使用します:

```
mysql> EXPLAIN SELECT * FROM performance_schema.accounts
WHERE (USER,HOST) = ('root','localhost')\G
***** 1. row *****
id: 1
select_type: SIMPLE
table: accounts
partitions: NULL
type: const
possible_keys: ACCOUNT
key: ACCOUNT
key_len: 278
ref: const,const
rows: 1
filtered: 100.00
Extra: NULL
```

EXPLAIN 出力は、**USER** カラムと **HOST** カラムで構成される **accounts** テーブルの **ACCOUNT** インデックスをオプティマイザが使用することを示しています。

パフォーマンススキーマのインデックスは仮想です: これらはパフォーマンススキーマストレージエンジンの構成であり、メモリーやディスクストレージを使用しません。パフォーマンススキーマは、効率的な実行計画を構築できるように、オプティマイザにインデックス情報を報告します。パフォーマンススキーマは、実際のインデックス構造を構築せずに効率的な検索を実行できるように、検索対象に関するオプティマイザ情報 (特定のキー値など) を使用します。この実装には、次の2つの重要な利点があります:

- 頻繁に更新されるテーブルで通常発生するメンテナンスコストを完全に回避します。
- これにより、クエリー実行の初期段階で取得されるデータ量が削減されます。インデックス付きカラムの条件の場合、パフォーマンススキーマはクエリー条件を満たすテーブル行のみを効率的に返します。インデックスがない場合、パフォーマンススキーマはテーブル内のすべての行を返すため、オプティマイザはあとで各行に対して条件を評価して最終結果を生成する必要があります。

パフォーマンススキーマのインデックスは事前定義されており、削除、追加、または変更できません。

パフォーマンススキーマのインデックスはハッシュインデックスに似ています。例:

- これらは、`=` または `<=>` 演算子を使用する等価比較にのみ使用されます。
- これらは順序付けられていません。クエリー結果に特定の行順序付け特性が必要な場合は、`ORDER BY` 句を含めます。

ハッシュインデックスの詳細は、[セクション8.3.9「B ツリーインデックスとハッシュインデックスの比較」](#) を参照してください。

8.2.5 データ変更ステートメントの最適化

このセクションでは、データ変更ステートメントを高速化する方法について説明: `INSERT`、`UPDATE` および `DELETE`。従来の OLTP アプリケーションおよび最新の web アプリケーションでは、通常、同時実行性が重要な多数の小さなデータ変更操作が実行されます。データ分析およびレポートアプリケーションでは、通常、一度に多数の行に影響するデータ変更操作が実行されます。主な考慮事項は、大量のデータを書き込み、インデックスを最新の状態に保つための I/O です。大量のデータの挿入と更新 (業界では ETL (「extract-transform-load」) と呼ばれる) では、`INSERT`、`UPDATE`、および `DELETE` ステートメントの効果を模倣する、その他の SQL ステートメントや外部コマンドを使用することがあります。

8.2.5.1 INSERT ステートメントの最適化

挿入の速度を最適化するには、多くの小さな操作を 1 つの大きな操作に組み合わせます。理想的には、単一の接続を作成し、多くの新しい行のデータを一度に送信し、すべてのインデックスの更新と一貫性チェックを最後まで延期します。

行の挿入に必要な時間は、次の要因によって決まります。ここでの数はおよその割合を示しています。

- 接続: (3)
- サーバーへのクエリーの送信: (2)
- クエリーの解析: (2)
- 行の挿入: (1 × 行サイズ)
- インデックスの挿入: (1 × インデックス数)
- クローズ: (1)

これには、テーブルを開く初期オーバーヘッドを考慮に入れていません。これは同時実行クエリーごとに 1 回実行されます。

テーブルのサイズによって、 $\log N$ だけインデックスの挿入が遅くなります (B ツリーインデックスであるとして)。

次の方法を使用して、挿入を高速化できます。

- 同じクライアントから同時に多数の行を挿入する場合は、複数の `VALUES` リストで `INSERT` ステートメントを使用して、同時に複数の行を挿入します。これは、個別の単一行の `INSERT` ステートメントを使用するより、大幅に (場合によっては数倍) 速くなります。空ではないテーブルにデータを追加する場合は、データの挿入をさらに速くするために、`bulk_insert_buffer_size` 変数を調整できます。[セクション5.1.8「サーバーシステム変数」](#) を参照してください。
- テキストファイルからテーブルをロードする場合は、`LOAD DATA` を使用します。通常、これは `INSERT` ステートメントを使用する場合より、20 倍速くなります。[セクション13.2.7「LOAD DATA ステートメント」](#) を参照してください。
- カラムにデフォルト値があることを利用します。挿入する値がデフォルト値と異なる場合にのみ、明示的に値を挿入します。これにより、MySQL が実行する必要がある解析が減り、挿入速度が向上します。
- InnoDB テーブルに固有のヒントについては、[セクション8.5.5「InnoDB テーブルの一括データロード」](#) を参照してください。

- [MyISAM](#) テーブルに固有のヒントについては、[セクション8.6.2「MyISAM テーブルの一括データロード」](#)を参照してください。

8.2.5.2 UPDATE ステートメントの最適化

更新ステートメントは、[SELECT](#) クエリーと同様に最適化されますが、書き込みの追加のオーバーヘッドがあります。書き込みの速度は更新されるデータの量と更新されるインデックス数によって異なります。変更がないインデックスは更新されません。

更新を速くするもう 1 つの方法は、更新を遅延して、あとで 1 行で多くの更新を実行することです。複数の更新をまとめて実行することで、テーブルをロックした場合に、一度に 1 つずつ実行するよりはるかに高速になります。

動的な行フォーマットを使用する [MyISAM](#) テーブルの場合、行を長い合計長に更新すると、行が分割されることがあります。頻繁にこれを実行する場合は、ときどき [OPTIMIZE TABLE](#) を使用することがきわめて重要になります。[セクション13.7.3.4「OPTIMIZE TABLE ステートメント」](#)を参照してください。

8.2.5.3 DELETE ステートメントの最適化

[MyISAM](#) テーブル内の個々の行を削除するために必要な時間は、インデックスの数に正確に比例します。行をもっと速く削除するには、[key_buffer_size](#) システム変数を増やして、キーキャッシュのサイズを大きくできます。[セクション5.1.1「サーバーの構成」](#)を参照してください。

[MyISAM](#) テーブルからすべての行を削除するには、[TRUNCATE TABLE tbl_name](#) の方が [DELETE FROM tbl_name](#) より速くなります。切り捨て操作はトランザクションセーフではありません。アクティブなトランザクションやアクティブなテーブルロックの途中で試みるとエラーが発生します。[セクション13.1.37「TRUNCATE TABLE ステートメント」](#)を参照してください。

8.2.6 データベース権限の最適化

権限のセットアップが複雑であるほど、すべての SQL ステートメントに適用されるオーバーヘッドが大きくなります。[GRANT](#) ステートメントによって確立された権限を簡単にすることで、クライアントがステートメントを実行するときの MySQL の権限チェックのオーバーヘッドを軽減できます。たとえば、テーブルレベルやカラムレベルの権限を付与しない場合、サーバーは [tables_priv](#) と [columns_priv](#) テーブルの内容をチェックする必要はなくなります。同じように、どのアカウントにもリソース制限を設けない場合、サーバーはリソースのカウンタを実行する必要がありません。ステートメント処理の負荷が著しく高い場合は、簡略化した付与構造を使用して、権限チェックのオーバーヘッドを軽減することを考慮してください。

8.2.7 その他の最適化のヒント

このセクションでは、クエリー処理速度を向上するためのさまざまな多くのヒントを示します。

- アプリケーションが関連する更新を実行するために複数のデータベースリクエストを作成する場合、ステートメントをストアドルーチンに結合するとパフォーマンスが向上します。同様に、アプリケーションで複数のカラム値または大量のデータに基づいて単一の結果を計算する場合、計算を UDF (ユーザー定義関数) に結合するとパフォーマンスが向上します。結果の高速データベース操作は、他のクエリー、アプリケーションおよび異なるプログラミング言語で記述されたコードで再利用できます。詳細は、[セクション25.2「ストアドルーチンの使用」](#) および [Adding Functions to MySQL](#) を参照してください。
- [ARCHIVE](#) テーブルで発生する圧縮の問題を修正するには、[OPTIMIZE TABLE](#) を使用します。[セクション16.5「ARCHIVE ストレージエンジン」](#)を参照してください。
- 可能な場合は、レポートを「[ライブ](#)」または「[統計](#)」として分類します。統計レポートに必要なデータは、ライブデータから定期的に生成されるサマリーテーブルからのみ作成されます。
- 行とカラムのテーブル構造に適合しないデータがある場合は、データを圧縮して [BLOB](#) カラムに格納できます。この場合、情報を圧縮および解凍するためのコードをアプリケーションに指定する必要がありますが、これにより、関連する値のセットの読み取りおよび書き込みのための I/O 操作が保存される可能性があります。
- Web サーバーでは、イメージおよびその他のバイナリアセットをファイルとして格納し、パス名をファイル自体ではなくデータベースに格納します。ほとんどの Web サーバーは、データベースコンテンツよりファイルのキャッ

シユに優れているため、ファイルの使用は一般に高速です。(ただし、この場合はバックアップとストレージの問題を自分で処理する必要があります。)

- 実際に高速化が必要な場合は、低レベルの MySQL インタフェースを参照してください。たとえば、MySQL InnoDB または MyISAM ストレージエンジンに直接アクセスすると、SQL インタフェースを使用する場合と比べて大幅に高速化できます。

同様に、NDBCLUSTER ストレージエンジンを使用するデータベースの場合は、NDB API の使用可能な使用状況を調査することをお勧めします (MySQL NDB Cluster API Developer Guide を参照)。

- レプリケーションは、特定の操作でパフォーマンスの向上を実現できます。クライアントの取得をレプリカ間で分散して、負荷を分割できます。バックアップの作成中にソースの速度が低下しないように、レプリカを使用してバックアップを作成できます。第17章「レプリケーション」を参照してください。

8.3 最適化とインデックス

SELECT 操作のパフォーマンスを向上する最善の方法は、クエリーでテストされる 1 つ以上のカラムにインデックスを作成することです。インデックスエントリは、テーブル行へのポインタのように動作し、クエリーが WHERE 句の条件に一致する行を迅速に特定し、それらの行のほかのカラム値を取得できます。すべての MySQL データ型にインデックスを設定できます。

クエリーで使用されている可能なすべてのカラムにインデックスを作成しようとしがちですが、不要なインデックスは領域を無駄にし、MySQL が使用するインデックスを判断するための時間を無駄にします。各インデックスを更新する必要があるため、インデックスは挿入、更新、削除のコストも追加します。最適なインデックスのセットを使用して、高速のクエリーを実現するために、適切なバランスを見つける必要があります。

8.3.1 MySQL のインデックスの使用の仕組み

インデックスは特定のカラム値のある行をすばやく見つけるために使用されます。インデックスがないと、MySQL は関連する行を見つけるために、先頭行から始めてテーブル全体を読み取る必要があります。テーブルが大きいほど、このコストが大きくなります。テーブルに問題のカラムのインデックスが含まれている場合、MySQL はすべてのデータを調べる必要なく、データファイルの途中のシークする位置をすばやく特定できます。これはすべての行を順次読み取るよりはるかに高速です。

ほとんどの MySQL インデックス (PRIMARY KEY、UNIQUE、INDEX、および FULLTEXT) は B ツリーに格納されます。例外: 空間データ型のインデックスは R ツリーを使用します。MEMORY テーブルはハッシュインデックスもサポートします。InnoDB は FULLTEXT インデックスの逆のリストを使用します。

一般に、インデックスは次の説明に示すように使われます。ハッシュインデックス (MEMORY テーブルで使用されているような) に固有の特性については、セクション 8.3.9 「B ツリーインデックスとハッシュインデックスの比較」で説明しています。

MySQL はこれらの操作にインデックスを使用します。

- WHERE 句に一致する行をすばやく見つけるため。
- 行を考慮に入れないようにするため。複数のインデックスから選択する場合、MySQL は通常最小数の行を見つめるインデックス (もっとも選択的なインデックス) を使用します。
- テーブルにマルチカラムインデックスがある場合、オプティマイザは、インデックスの左端のプリフィクスを使用して行をルックアップできます。たとえば、(col1, col2, col3) に 3 カラムのインデックスがある場合、(col1)、(col1, col2)、および (col1, col2, col3) に対して、インデックス検索機能を使用できます。詳細については、セクション 8.3.6 「マルチカラムインデックス」を参照してください。
- 結合の実行時に、ほかのテーブルから行を取得するため。カラムが同じ型とサイズで宣言されていると、MySQL はカラムのインデックスをより効率的に使用できます。このコンテキストでは、VARCHAR と CHAR は同じサイズで宣言されていれば同じとみなされます。たとえば、VARCHAR(10) と CHAR(10) は同じサイズですが、VARCHAR(10) と CHAR(15) は異なります。

非バイナリ文字列カラム間での比較の場合、両方のカラムで同じ文字セットを使用しているべきです。たとえば、utf8 カラムと latin1 カラムの比較はインデックスの使用の可能性を否定します。

異種のカラムの比較 (文字列カラムを時間または数値カラムと比較するなど) では、値を変換せずに直接比較できない場合、インデックスの使用が妨げられることがあります。数値カラム内の 1 などの特定の値の場合、'1'、'1'、'00001'、または '01.e1' などの文字列カラム内の任意の数の値と等しくなる可能性があります。これは、文字列カラムのインデックスの使用を除外します。

- 特定のインデックス設定されたカラム `key_col` に対して、`MIN()` あるいは `MAX()` 値を見つけるため。これはインデックス内の `key_col` より前に発生するすべてのキーパートで、`WHERE key_part_N = constant` が使用されているかどうかをチェックするプリプロセスによって最適化されます。この場合、MySQL は各 `MIN()` または `MAX()` 式に対して単一キールックアップを行い、それを定数で置き換えます。すべての式が定数で置き換えられた場合、クエリーは同時に返されます。例:

```
SELECT MIN(key_part2),MAX(key_part2)
FROM tbl_name WHERE key_part1=10;
```

- 使用可能なインデックスの左端のプリフィクスに対してソートまたはグループ化が行われている場合 (たとえば、`ORDER BY key_part1, key_part2`) に、テーブルをソートまたはグループ化するため。すべてのキーパートのあとに `DESC` が付けられている場合、キーは逆の順序で読み取られます。(または、インデックスが降順の場合、キーは順方向に読み取られます。) [セクション8.2.1.16「ORDER BY の最適化」](#)、[セクション8.2.1.17「GROUP BY の最適化」](#)、および [セクション8.3.13「降順インデックス」](#) を参照してください。
- 場合によっては、データ行を参照しないで値を取得するように、クエリーを最適化できます。(クエリーの必要なすべての結果を提供するインデックスは、[カバーするインデックス](#)と呼ばれます。) クエリーがテーブルから特定のインデックスに含まれるカラムのみを使用している場合、きわめて高速に、選択した値をインデックスツリーから取得できます。

```
SELECT key_part3 FROM tbl_name
WHERE key_part1=1
```

小さなテーブルまたは、レポートクエリーが行の大半またはすべてを処理する大きなテーブルに対するクエリーでは、インデックスはあまり重要ではありません。クエリーで行の大半にアクセスする必要がある場合は、順次読み取る方が、インデックスを処理するより高速です。クエリーですべての行が必要でない場合でも、順次読み取りは、ディスクシークを最小にします。詳細は、[セクション8.2.1.23「全テーブルスキャンの回避」](#) を参照してください。

8.3.2 主キーの最適化

テーブルの主キーは、もっとも重要なクエリーで使用するカラムやカラムのセットを表します。それには、高速のクエリーパフォーマンスのため、インデックスが関連付けられます。それには `NULL` 値を含めることができないため、クエリーパフォーマンスは `NOT NULL` 最適化からメリットが得られます。InnoDB ストレージエンジンによって、テーブルデータが、主キーカラムに基づいて、超高速ルックアップおよびソートを実行するように物理的に編成されます。

テーブルが大きく、重要でも、主キーとして使用する明確なカラムやカラムのセットがない場合は、自動インクリメント値で個別のカラムを作成して、主キーとして使用できます。これらの一意の ID は、外部キーを使用してテーブルを結合する場合に、ほかのテーブル内の対応する行へのポインタとして使用できます。

8.3.3 SPATIAL インデックス最適化

MySQL では、`NOT NULL` ジオメトリ値カラムに `SPATIAL` インデックスを作成できます ([セクション11.4.10「空間インデックスの作成」](#) を参照)。オプティマイザは、インデックス付けされたカラムの `SRID` 属性をチェックして、比較に使用する空間参照システム (SRS) を決定し、SRS に適した計算を使用します。(MySQL 8.0 より前では、オプティマイザはデカルト計算を使用して `SPATIAL` インデックス値の比較を実行します。このような操作の結果は、非デカルト `SRID` を持つ値がカラムに含まれている場合は未定義です。)

比較が適切に機能するには、`SPATIAL` インデックスの各カラムが `SRID` 制限付きである必要があります。つまり、カラム定義には明示的な `SRID` 属性が含まれている必要があり、すべてのカラム値は同じ `SRID` を持つ必要があります。

オプティマイザは、`SRID` で制限されたカラムに対してのみ `SPATIAL` インデックスを考慮します:

- デカルト `SRID` に制限されたカラムのインデックスを使用すると、デカルト境界ボックスの計算が可能になります。

- 地理 SRID に制限されたカラムのインデックスを使用すると、地理的境界ボックスの計算が可能になります。

オプティマイザは、SRID 属性を持たない (したがって SRID 制限のない) カラムの SPATIAL インデックスを無視します。MySQL では、このようなインデックスは次のように維持されます:

- テーブルの変更 (INSERT, UPDATE, DELETE など) のために更新されます。カラムにデカルト値と地理的値が混在している場合でも、インデックスがデカルトであるかのように更新が行われます。
- これらは下位互換性のためにのみ存在します (たとえば、MySQL 5.7 でダンプを実行し、MySQL 8.0 でリストアする機能)。SRID 制限のないカラムの SPATIAL インデックスはオプティマイザで使用されないため、このような各カラムを変更する必要があります:
- カラム内のすべての値が同じ SRID を持つことを確認します。ジオメトリカラム `col_name` に含まれる SRID を確認するには、次のクエリーを使用します:

```
SELECT DISTINCT ST_SRID(col_name) FROM tbl_name;
```

クエリーで複数の行が返された場合、カラムには SRID の混在が含まれます。その場合は、その内容を変更して、すべての値が同じ SRID を持つようにします。

- 明示的な SRID 属性を持つようにカラムを再定義します。
- SPATIAL インデックスを再作成します。

8.3.4 外部キーの最適化

テーブルに多くのカラムがあり、多くのさまざまなカラムの組み合わせをクエリーする場合、あまり頻繁に使用されないデータをそれぞれ少数のカラムを持つ個別のテーブルに分割し、それらを、メインテーブルの数値 ID カラムを複製してメインテーブルに関連付けると、効率的なことがあります。そのようにして、小さな各テーブルに、そのデータの高速ルックアップのための主キーを設定でき、結合操作を使用して必要とするカラムのセットだけをクエリーできます。データの分散状況に応じて、関連カラムがディスク上にまとめてパックされるため、クエリーで実行する I/O が少なくなり、使用するキャッシュメモリーが減る可能性があります。(パフォーマンスを最大にするため、クエリーはディスクから可能なかぎり少ないデータブロックを読み取ろうとします。数個のカラムしかないテーブルでは各データブロックにより多くのデータを収めることができます。)

8.3.5 カラムインデックス

もっとも一般的なインデックスの種類には、単一カラムがあり、データ構造にそのカラムの値のコピーを格納し、対応するカラム値のある行を高速にルックアップできます。B ツリーデータ構造により、インデックスは、WHERE 句内の =、>、≤、BETWEEN、IN などの演算子に対応する特定の値、値のセット、または値の範囲をすばやく見つけることができます。

テーブルあたりの最大インデックス数とインデックスの最大長は、ストレージエンジンごとに定義されます。第15章「InnoDB ストレージエンジン」および第16章「代替ストレージエンジン」を参照してください。すべてのストレージエンジンは、1 テーブルあたり 16 個以上のインデックスと 256 バイト以上の合計インデックス長をサポートします。ほとんどのストレージエンジンでは、制限が高く設定されています。

カラムインデックスの詳細は、セクション13.1.15「CREATE INDEX ステートメント」を参照してください。

- [インデックス接頭辞](#)
- [FULLTEXT インデックス](#)
- [空間インデックス](#)
- [MEMORY ストレージエンジンでのインデックス](#)

インデックス接頭辞

文字列カラムのインデックス指定に `col_name(N)` 構文を使用すると、カラムの最初の N 文字のみを使用するインデックスを作成できます。このようにカラム値のプリフィクスだけのインデックスを作成すると、インデックスファイル

をかなり小さくできます。BLOB または TEXT カラムにインデックス設定する場合、インデックスのプリフィクス長を指定する必要があります。例:

```
CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
```

REDUNDANT または COMPACT の行形式を使用する InnoDB テーブルでは、接頭辞の長さは最大 767 バイトです。DYNAMIC または COMPRESSED の行形式を使用する InnoDB テーブルでは、接頭辞の長さの制限は 3072 バイトです。「MyISAM の場合」テーブルでは、接頭辞の長さの制限は 1000 バイトです。

注記

接頭辞の制限はバイト単位で測定されますが、CREATE TABLE、ALTER TABLE および CREATE INDEX ステートメントの接頭辞の長さは、非バイナリ文字列型 (CHAR, VARCHAR, TEXT) の場合は文字数、バイナリ文字列型 (BINARY, VARBINARY, BLOB) の場合はバイト数として解釈されます。マルチバイト文字セットを使用する非バイナリ文字列カラムに接頭辞の長さを指定する場合は、これを考慮してください。

検索語がインデックス接頭辞の長さを超える場合、インデックスを使用して一致しない行が除外され、残りの行で一致の可能性が調査されます。

インデックス接頭辞の詳細は、[セクション 13.1.15 「CREATE INDEX ステートメント」](#) を参照してください。

FULLTEXT インデックス

FULLTEXT インデックスは、全文検索に使用されます。InnoDB および MyISAM ストレージエンジンのみが、CHAR、VARCHAR、および TEXT カラムに対してのみ、FULLTEXT インデックスをサポートしています。インデックス設定は常にカラム全体に対して行われ、カラムプリフィクスインデックス設定はサポートされていません。詳細は、[セクション 12.10 「全文検索関数」](#) を参照してください。

最適化は、単一の InnoDB テーブルに対する特定の種類の FULLTEXT クエリーに適用されます。これらの特性を持つクエリーは特に効率的です。

- ドキュメント ID またはドキュメント ID と検索ランクのみを返す FULLTEXT クエリー。
- 一致する行をスコアの降順でソートし、LIMIT 句を適用して、上位 N 個の一致する行を取得する FULLTEXT クエリー。この最適化を適用するには、WHERE 句がなく、降順の単一の ORDER BY 句のみがある必要があります。
- 検索語に一致する行の COUNT(*) 値のみを取得し、追加の WHERE 句がない FULLTEXT クエリー。WHERE 句を > 0 比較演算子を使用せずに、WHERE MATCH(text) AGAINST ('other_text') とコーディングします。

全文式を含むクエリーの場合、MySQL では、クエリー実行の最適化フェーズ中にこれらの式が評価されます。オプティマイザは、全文式を参照して見積りを行うだけでなく、実行計画の開発プロセスでそれらを実際に評価します。

この動作の影響は、全文クエリーの EXPLAIN は通常、最適化フェーズ中に式の評価が行われない非全文クエリーより遅くなることです。

全文クエリーの EXPLAIN では、最適化中に一致が発生するため、Extra カラムに Select tables optimized away が表示される場合があります。この場合、後で実行する際にテーブルへのアクセスは必要ありません。

空間インデックス

空間データ型にインデックスを作成できます。MyISAM および InnoDB は、空間型の R ツリーインデックスをサポートしています。ほかのストレージエンジンは、空間型のインデックス設定に B ツリーを使用します (ARCHIVE を除きます。これは空間型のインデックス設定をサポートしていません)。

MEMORY ストレージエンジンでのインデックス

MEMORY ストレージエンジンはデフォルトで HASH インデックスを使用しますが、BTREE インデックスもサポートしています。

8.3.6 マルチカラムインデックス

MySQL は複合インデックス (つまり、複数のカラムに対するインデックス) を作成できます。インデックスは最大 16 カラムで構成できます。特定のデータ型では、カラムのプリフィクスにインデックスを設定できます ([セクション 8.3.5 「カラムインデックス」](#) を参照してください)。

MySQL では、インデックスで、すべてのカラムをテストするクエリーまたは最初のカラム、最初の 2 つのカラム、最初の 3 つのカラムというようにテストするクエリーにマルチカラムインデックスを使用できます。インデックス定義の正しい順序でカラムを指定する場合、単一の複合インデックスにより、同じテーブルへの複数の種類のクエリーを高速化できます。

マルチカラムインデックスは、インデックス設定されたカラムの値を連結して作成された値を格納する行である、ソート済みの配列とみなすことができます。

注記

複合インデックスの代わりに、ほかのカラムの情報に基づいて「ハッシュされた」カラムを導入できます。このカラムが短く、十分に一意で、インデックスが設定されている場合は、多数のカラムへの「広範な」インデックスより速くなる可能性があります。MySQL では、この追加カラムをきわめて簡単に使用できます。

```
SELECT * FROM tbl_name
WHERE hash_col=MD5(CONCAT(val1,val2))
AND col1=val1 AND col2=val2;
```

テーブルが次のような仕様であるとして。

```
CREATE TABLE test (
  id INT NOT NULL,
  last_name CHAR(30) NOT NULL,
  first_name CHAR(30) NOT NULL,
  PRIMARY KEY (id),
  INDEX name (last_name,first_name)
);
```

`name` インデックスは、`last_name` カラムと `first_name` カラムに対するインデックスです。このインデックスは、`last_name` 値と `first_name` 値の組み合わせに既知の範囲の値を指定するクエリーで、ルックアップに使用できません。そのカラムはインデックスの左端のプリフィクスであるため、`last_name` 値だけを指定するクエリーにも使用できます (このセクションで後述するように)。そのため、`name` インデックスは、次のクエリーでのルックアップに使用されます。

```
SELECT * FROM test WHERE last_name='Jones';

SELECT * FROM test
WHERE last_name='Jones' AND first_name='John';

SELECT * FROM test
WHERE last_name='Jones'
AND (first_name='John' OR first_name='Jon');

SELECT * FROM test
WHERE last_name='Jones'
AND first_name >='M' AND first_name < 'N';
```

ただし、`name` インデックスは次のクエリーでのルックアップには使用されません。

```
SELECT * FROM test WHERE first_name='John';

SELECT * FROM test
WHERE last_name='Jones' OR first_name='John';
```

次の `SELECT` ステートメントを発行するとします。

```
SELECT * FROM tbl_name
WHERE col1=val1 AND col2=val2;
```

`col1` と `col2` に対するマルチカラムインデックスが存在する場合、該当する行を直接フェッチできます。`col1` および `col2` に対して個別の単一カラムのインデックスが存在する場合、オプティマイザは、インデックスマージ最適化 ([セクション 8.2.1.3 「インデックスマージの最適化」](#) を参照してください) の使用を試みるか、またはより多くの行を除

外するインデックスを判断して、そのインデックスを使用して行をフェッチすることで、もっとも制限の厳しいインデックスを見つけようとしています。

テーブルにマルチカラムインデックスがある場合、オプティマイザは、インデックスの左端のプリフィクスを使用して行をルックアップできます。たとえば、(col1, col2, col3) に 3 カラムのインデックスがある場合、(col1)、(col1, col2)、および (col1, col2, col3) に対して、インデックス検索機能を使用できます。

カラムがインデックスの左端のプリフィクスを形成していない場合、MySQL はこのインデックスを使用してルックアップを実行できません。ここに示す `SELECT` ステートメントがあるとしています。

```
SELECT * FROM tbl_name WHERE col1=val1;
SELECT * FROM tbl_name WHERE col1=val1 AND col2=val2;

SELECT * FROM tbl_name WHERE col2=val2;
SELECT * FROM tbl_name WHERE col2=val2 AND col3=val3;
```

(col1, col2, col3) にインデックスが存在する場合、最初の 2 つのクエリーだけがインデックスを使用します。3 番目と 4 番目のクエリーにはインデックス付けされたカラムが含まれますが、(col2) および (col2, col3) は (col1, col2, col3) の左端の接頭辞ではないため、インデックスを使用してルックアップを実行しません。

8.3.7 インデックスの使用の確認

すべてのクエリーがテーブル内に作成したインデックスを実際に使用していることを常に確認します。セクション 8.8.1 「`EXPLAIN` によるクエリーの最適化」に説明するように、`EXPLAIN` ステートメントを使用します。

8.3.8 InnoDB および MyISAM インデックス統計コレクション

ストレージエンジンはオプティマイザによって使用されるテーブルに関する統計を収集します。テーブル統計は値グループに基づきますが、ここで値グループは同じキープリフィクス値を持つ行のセットです。オプティマイザの目的で、重要な統計は平均値グループサイズです。

MySQL は平均値グループサイズを次のように使用します。

- `ref` アクセスごとに読み取る必要がある行数を見積もるには
- 部分結合によって生成される行数、つまりフォームの操作によって生成される行数を見積もるため

```
(...) JOIN tbl_name ON tbl_name.key = expr
```

インデックスの平均値グループサイズが増えるほど、ルックアップあたりの平均行数が増えるため、それらの 2 つの目的でインデックスが役立たなくなります。インデックスが最適化の目的に役立つようにするには、各インデックス値でターゲットとするテーブル内の行を少なくすることがもっとも適切です。指定したインデックス値が多数の行を生成する場合、そのインデックスはあまり役に立たず、MySQL がそれを使用する可能性は少なくなります。

平均値グループサイズは、値グループの数であるテーブルカーディナリティーと関連しています。`SHOW INDEX` ステートメントは、`N/S` に基づいて、カーディナリティー値を表示します。ここで `N` はテーブル内の行数で、`S` は平均値グループサイズです。その比率から、テーブル内の値グループの概数がわかります。

`<=>` 比較演算子に基づいた結合では、`NULL` の扱いはほかの値と異なりません。ほかのどの `N` に対しても `N <=> N` とまったく同じように、`NULL <=> NULL` です。

ただし、`=` 演算子に基づく結合では、`NULL` は `NULL` 以外の値と異なります。`expr1` または `expr2` (または両方) が `NULL` である場合、`expr1 = expr2` は `true` になりません。これは、`tbl_name.key = expr` 形式の比較のための `ref` アクセスに影響: 比較は `true` にできないため、`expr` の現在の値が `NULL` の場合、MySQL はテーブルにアクセスしません。

`=` 比較では、テーブルにある `NULL` 値の数は問題になりません。最適化の目的で、関連のある値は `NULL` 以外の値グループの平均サイズです。ただし、MySQL では現在その平均サイズを収集したり、使用したりできません。

InnoDB および MyISAM テーブルでは、`innodb_stats_method` および `myisam_stats_method` システム変数をそれぞれ使用して、テーブル統計のコレクションに対していくらかの制御ができます。これらの変数には、3 つの可能性のある値を使用でき、次のように異なります。

- 変数が `nulls_equal` に設定されている場合、すべての `NULL` 値が同一として扱われます (つまり、それらすべてが単一の値グループを形成します)。

`NULL` 値グループサイズが、`NULL` 以外の値グループサイズよりはるかに大きい場合、このメソッドは平均値グループサイズを上方に歪めます。これにより、オプティマイザには、`NULL` 以外の値を検索する結合に対して、インデックスが実際以上に役に立たないかのように見えます。結果として、`nulls_equal` メソッドにより、オプティマイザに `ref` アクセスに対してインデックスを使用すべきときでも使用させないようにすることがあります。

- 変数が `nulls_unequal` に設定されている場合、`NULL` 値は同じとみなされません。代わりに、各 `NULL` 値はサイズ 1 の個別の値グループを形成します。

多くの `NULL` 値がある場合、このメソッドは平均値グループサイズを下方に歪めます。`NULL` 以外の平均値グループサイズが大きい場合、`NULL` 値をサイズ 1 のグループとしてカウントすると、オプティマイザは `NULL` 以外の値を検索する結合に対して、インデックスの値を多く見積もりすぎます。結果として、`nulls_unequal` メソッドによって、ほかのメソッドの方が適している可能性がある場合に、オプティマイザに `ref` ルックアップに対してこのインデックスを使用させることがあります。

- 変数が `nulls_ignored` に設定されている場合、`NULL` 値は無視されます。

= より <=> を使用する多くの結合を使用する傾向がある場合、比較で `NULL` 値は特別ではなく、`NULL` は互いに等しくなります。この場合、`nulls_equal` は適切な統計メソッドです。

`innodb_stats_method` システム変数にはグローバル値があります。`myisam_stats_method` システム変数にはグローバル値とセッション値の両方があります。グローバル値を設定すると、対応するストレージエンジンからのテーブルの統計収集に影響します。セッション値を設定すると、現在のクライアント接続のみに対する統計収集に影響します。つまり、`myisam_stats_method` のセッション値を設定することで、他のクライアントに影響を与えずに、特定のメソッドでテーブル統計を強制的に再生成できます。

MyISAM テーブルの統計を再生成するには、次のいずれかの方法を使用できます:

- `myisamchk --stats_method=method_name --analyze` を実行します
- テーブルを変更して、統計を古くさせ (たとえば、行を挿入してから削除します)、次に `myisam_stats_method` を設定して、`ANALYZE TABLE` ステートメントを発行します。

`innodb_stats_method` と `myisam_stats_method` の使用に関するいくつかの警告は次のとおりです。

- 先述したように、テーブル統計を明示的に収集させることができます。ただし、MySQL は統計を自動的に収集することもあります。たとえば、テーブルへのステートメントの実行の途中で、そうしたステートメントの中にはテーブルを変更するものもあり、MySQL は統計を収集する場合があります。(たとえば、これは一括挿入や削除、または一部の `ALTER TABLE` ステートメントで行われることがあります。) これが行われた場合、その時点での `innodb_stats_method` または `myisam_stats_method` の値を使用して、統計が収集されます。したがって、一方の方法を使用して統計を収集し、後でテーブル統計が自動的に収集されるときにシステム変数がもう一方の方法に設定されている場合は、もう一方の方法が使用されます。
- 特定のテーブルの統計の生成に使用されたメソッドを伝える方法はありません。
- これらの変数は InnoDB および MyISAM テーブルにのみ適用されます。ほかのストレージエンジンはテーブル統計を収集するメソッドが 1 つしかありません。通常、それは `nulls_equal` メソッドに近いものになります。

8.3.9 B ツリーインデックスとハッシュインデックスの比較

B ツリーおよびハッシュデータ構造を理解することは、インデックスにこれらのデータ構造を使用するさまざまなストレージエンジンで (特に B ツリーインデックスを使用するか、ハッシュインデックスを使用するかを選択できる `MEMORY` ストレージエンジンの場合に)、さまざまなクエリーがどのように実行されるかを予測するのに役立つ可能性があります。

- B ツリーインデックスの特性
- ハッシュインデックスの特性

B ツリーインデックスの特性

B ツリーインデックスは `=`、`>`、`>=`、`<`、`<=`、または `BETWEEN` 演算子を使用する式で、カラム比較に使用できます。このインデックスは、`LIKE` への引数がワイルドカード文字で始まらない定数文字列の場合の `LIKE` 比較にも使用できます。たとえば、次の `SELECT` ステートメントはインデックスを使用します。

```
SELECT * FROM tbl_name WHERE key_col LIKE 'Patrick%';
SELECT * FROM tbl_name WHERE key_col LIKE 'Pat%_ck%';
```

最初のステートメントでは、`'Patrick' <= key_col < 'Patricl'` の行のみが考慮されます。2 つめのステートメントでは、`'Pat' <= key_col < 'Pau'` の行のみが考慮されます。

次の `SELECT` ステートメントはインデックスを使用しません。

```
SELECT * FROM tbl_name WHERE key_col LIKE '%Patrick%';
SELECT * FROM tbl_name WHERE key_col LIKE other_col;
```

最初のステートメントでは、`LIKE` 値はワイルドカード文字で始まります。2 つめのステートメントでは、`LIKE` 値は定数ではありません。

... `LIKE '%string%'` を使用し、`string` が 3 文字より長い場合、MySQL は Turbo Boyer-Moore アルゴリズムを使用して、文字列のパターンを初期化してから、このパターンを使用して検索をより迅速に実行します。

`col_name IS NULL` を使用した検索では、`col_name` にインデックスが設定されている場合にインデックスが使用されます。

`WHERE` 句内のすべての `AND` レベルにまたがっていないインデックスは、クエリーの最適化に使用されません。言い換えると、インデックスの使用を可能にするには、インデックスのプリフィクスがすべての `AND` グループで使用されている必要があります。

次の `WHERE` 句ではインデックスが使用されます。

```
... WHERE index_part1=1 AND index_part2=2 AND other_column=3

/* index = 1 OR index = 2 */
... WHERE index=1 OR A=10 AND index=2

/* optimized like "index_part1='hello'" */
... WHERE index_part1='hello' AND index_part3=5

/* Can use index on index1 but not on index2 or index3 */
... WHERE index1=1 AND index2=2 OR index1=3 AND index3=3;
```

これらの `WHERE` 句ではインデックスが使用されません。

```
/* index_part1 is not used */
... WHERE index_part2=1 AND index_part3=2

/* Index is not used in both parts of the WHERE clause */
... WHERE index=1 OR A=10

/* No index spans all rows */
... WHERE index_part1=1 OR index_part2=10
```

MySQL ではインデックスが使用できる場合でも使用しないことがあります。これが発生する 1 つの状況は、オプティマイザが、インデックスを使用することによって MySQL がテーブルの大部分の行にアクセスする必要があると推定した場合です。(この場合、必要なシークが少ないため、テーブルスキャンの方がはるかに高速になる可能性があります。)ただし、そのようなクエリーで、行の一部のみを取得する `LIMIT` を使用している場合、結果で返す少数の行をはるかにすばやく見つけることができるため、MySQL はとにかくインデックスを使用します。

ハッシュインデックスの特性

ハッシュインデックスは先述したものといくらか異なる特性を持ちます。

- それらは、`=` または `<=>` 演算子を使用する等価比較にのみ使用されます (ただしきわめて高速です)。それらは、値の範囲を見つける `<` などの比較演算子には使用されません。この種類の単一値ルックアップに依存するシステムは、「キー値ストア」として知られています。そのようなアプリケーションで MySQL を使用するには、可能な限りハッシュインデックスを使用します。

- オプティマイザはハッシュインデックスを使用して、**ORDER BY** 操作を高速化することはできません。(この種類のインデックスは順番に次のエントリを検索するために使用できません。)
- MySQL は 2 つの値の間におよそどのくらいの行数があるかを判断できません (これは範囲オプティマイザによって使用するインデックスを特定するために使用されます)。これは、**MyISAM** または **InnoDB** テーブルをハッシュインデックス設定された **MEMORY** テーブルに変更した場合に、一部のクエリーに影響することがあります。
- 行の検索にはキー全体のみを使用できます。(B ツリーインデックスでは、キーの任意の左端のプリフィクスを使用して行を検索できます。)

8.3.10 インデックス拡張の使用

InnoDB は、自動的に各セカンダリインデックスに主キーカラムを追加して、それを拡張します。このテーブル定義について考えます。

```
CREATE TABLE t1 (
  i1 INT NOT NULL DEFAULT 0,
  i2 INT NOT NULL DEFAULT 0,
  d DATE DEFAULT NULL,
  PRIMARY KEY (i1, i2),
  INDEX k_d (d)
) ENGINE = InnoDB;
```

このテーブルでは、カラム (**i1**, **i2**) に主キーを定義しています。さらに、カラム (**d**) にセカンダリインデックス **k_d** を定義していますが、内部で **InnoDB** はこのインデックスを拡張し、それをカラム (**d**, **i1**, **i2**) として処理します。

オプティマイザは、拡張セカンダリインデックスの主キーカラムを考慮して、そのインデックスを使用する方法と使用するかどうかを決定します。これにより、クエリー実行計画の効率が向上し、パフォーマンスが向上する可能性があります。

オプティマイザは、**ref**、**range** および **index_merge** のインデックスアクセス、ルーズインデックススキャンアクセス、結合およびソートの最適化、**MIN()/MAX()** の最適化に拡張セカンダリインデックスを使用できます。

次の例に、オプティマイザが拡張セカンダリインデックスを使用するかどうかによって、実行プランにどのような影響を与えるかを示します。これらの行に **t1** が移入されているとします。

```
INSERT INTO t1 VALUES
(1, 1, '1998-01-01'), (1, 2, '1999-01-01'),
(1, 3, '2000-01-01'), (1, 4, '2001-01-01'),
(1, 5, '2002-01-01'), (2, 1, '1998-01-01'),
(2, 2, '1999-01-01'), (2, 3, '2000-01-01'),
(2, 4, '2001-01-01'), (2, 5, '2002-01-01'),
(3, 1, '1998-01-01'), (3, 2, '1999-01-01'),
(3, 3, '2000-01-01'), (3, 4, '2001-01-01'),
(3, 5, '2002-01-01'), (4, 1, '1998-01-01'),
(4, 2, '1999-01-01'), (4, 3, '2000-01-01'),
(4, 4, '2001-01-01'), (4, 5, '2002-01-01'),
(5, 1, '1998-01-01'), (5, 2, '1999-01-01'),
(5, 3, '2000-01-01'), (5, 4, '2001-01-01'),
(5, 5, '2002-01-01');
```

ここで次のクエリーを考慮します。

```
EXPLAIN SELECT COUNT(*) FROM t1 WHERE i1 = 3 AND d = '2000-01-01'
```

実行計画は、拡張インデックスが使用されているかどうかによって異なります。

オプティマイザがインデックス拡張を考慮しない場合、それはインデックス **k_d** を (**d**) のみとして扱います。クエリーの **EXPLAIN** では次の結果が生成されます。

```
mysql> EXPLAIN SELECT COUNT(*) FROM t1 WHERE i1 = 3 AND d = '2000-01-01'\G
***** 1. row *****
   id: 1
  select_type: SIMPLE
    table: t1
     type: ref
possible_keys: PRIMARY,k_d
      key: k_d
```

```
key_len: 4
ref: const
rows: 5
Extra: Using where; Using index
```

オプティマイザがインデックス拡張を考慮する場合、それはインデックス `k_d` を `(d, i1, i2)` として扱います。この場合、それは左端のインデックスプリフィクス `(d, i1)` を使用して、より適切な実行プランを生成できます。

```
mysql> EXPLAIN SELECT COUNT(*) FROM t1 WHERE i1 = 3 AND d = '2000-01-01'\G
***** 1. row *****
id: 1
select_type: SIMPLE
table: t1
type: ref
possible_keys: PRIMARY,k_d
key: k_d
key_len: 8
ref: const,const
rows: 1
Extra: Using index
```

どちらの場合も、`key` はオプティマイザがセカンダリインデックス `k_d` を使用することを示しますが、`EXPLAIN` 出力には拡張インデックスの使用による次の改善点が示されています:

- `key_len` は 4 バイトから 8 バイトになり、キールックアップでカラム `d` だけでなく、`d` と `i1` も使用されていることを示しています。
- キールックアップで 1 つではなく 2 つのキーパートが使用されるため、`ref` 値が `const` から `const,const` に変更されています。
- `rows` 数は 5 から 1 に減少し、`InnoDB` が結果を生成するために調査する必要がある行数が少なくなることを示しています。
- `Extra` 値が `Using where; Using index` から `Using index` に変更されています。このことは、データ行のカラムを参照せずに、インデックスのみを使用して、行を読み取れることを意味します。

拡張インデックスの使用のオプティマイザの動作の違いは、`SHOW STATUS` でも確認できます。

```
FLUSH TABLE t1;
FLUSH STATUS;
SELECT COUNT(*) FROM t1 WHERE i1 = 3 AND d = '2000-01-01';
SHOW STATUS LIKE 'handler_read%'
```

前述のステートメントには、テーブルキャッシュをフラッシュしてステータスカウンタをクリアする `FLUSH TABLES` および `FLUSH STATUS` が含まれています。

インデックス拡張を使用しないと、`SHOW STATUS` は次の結果を生成します。

```
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Handler_read_first | 0 |
| Handler_read_key | 1 |
| Handler_read_last | 0 |
| Handler_read_next | 5 |
| Handler_read_prev | 0 |
| Handler_read_rnd | 0 |
| Handler_read_rnd_next | 0 |
+-----+-----+
```

インデックス拡張を使用すると、`SHOW STATUS` は次の結果を生成します。 `Handler_read_next` 値が 5 から 1 に減少し、インデックスをより効率的に使用していることを示しています。

```
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Handler_read_first | 0 |
| Handler_read_key | 1 |
| Handler_read_last | 0 |
| Handler_read_next | 1 |
+-----+-----+
```

```
| Handler_read_prev | 0 |  
| Handler_read_rnd | 0 |  
| Handler_read_rnd_next | 0 |  
+-----+-----+
```

`optimizer_switch` システム変数の `use_index_extensions` フラグを使用すると、InnoDB テーブルのセカンダリ索引の使用方法を決定するときにオプティマイザが主キーカラムを考慮するかどうかを制御できます。デフォルトで、`use_index_extensions` は有効です。インデックス拡張機能の使用を無効にするとパフォーマンスが向上するかどうかを確認するには、次のステートメントを使用します:

```
SET optimizer_switch = 'use_index_extensions=off';
```

オプティマイザによるインデックス拡張の使用は、インデックス (16) のキーパートの数と最大キー長 (3072 バイト) への通常の制限によります。

8.3.11 生成されたカラムインデックスのオプティマイザによる使用

MySQL では、生成されたカラムのインデックスがサポートされます。例:

```
CREATE TABLE t1 (f1 INT, gc INT AS (f1 + 1) STORED, INDEX (gc));
```

生成されたカラム `gc` は、式 `f1 + 1` として定義されます。カラムもインデックス付けされ、オプティマイザは実行計画の構成時にそのインデックスを考慮できます。次のクエリーでは、`WHERE` 句が `gc` を参照し、オプティマイザはそのカラムのインデックスがより効率的な計画をもたらすかどうかを考慮します:

```
SELECT * FROM t1 WHERE gc > 9;
```

オプティマイザは、生成されたカラムのインデックスを使用して実行計画を生成できます。これらのカラムへの名前によるクエリーに直接参照がない場合でも同様です。これは、`WHERE`、`ORDER BY` または `GROUP BY` 句が、インデックス付けされた生成カラムの定義と一致する式を参照している場合に発生します。次のクエリーは、`gc` を直接参照しませんが、`gc` の定義と一致する式を使用します:

```
SELECT * FROM t1 WHERE f1 + 1 > 9;
```

オプティマイザは、式 `f1 + 1` が `gc` の定義と一致し、`gc` がインデックス付けされていることを認識するため、実行計画の構成時にそのインデックスが考慮されます。これは、`EXPLAIN` を使用して表示できます:

```
mysql> EXPLAIN SELECT * FROM t1 WHERE f1 + 1 > 9\G  
***** 1. row *****  
id: 1  
select_type: SIMPLE  
table: t1  
partitions: NULL  
type: range  
possible_keys: gc  
key: gc  
key_len: 5  
ref: NULL  
rows: 1  
filtered: 100.00  
Extra: Using index condition
```

実際には、オプティマイザは式 `f1 + 1` を、式に一致する生成されたカラムの名前に置き換えています。これは、`SHOW WARNINGS` によって表示される拡張 `EXPLAIN` 情報で使用可能なリライトされたクエリーでもわかります:

```
mysql> SHOW WARNINGS\G  
***** 1. row *****  
Level: Note  
Code: 1003  
Message: /* select#1 */ select `test`.`t1`.`f1` AS `f1`,`test`.`t1`.`gc`  
AS `gc` from `test`.`t1` where (`test`.`t1`.`gc` > 9)
```

生成されたカラムインデックスのオプティマイザでの使用には、次の制限および条件が適用されます:

- クエリー式が生成されたカラム定義と一致するには、式が同一であり、同じ結果タイプである必要があります。たとえば、生成されたカラム式が `f1 + 1` の場合、クエリーで `1 + f1` が使用されているか、`f1 + 1` (整数式) が文字列と比較されても、オプティマイザは一致を認識しません。

- 最適化はこれらの演算子に適用されます: `=`, `<`, `<=`, `>`, `>=`, `BETWEEN` および `IN()`。

`BETWEEN` および `IN()` 以外の演算子の場合、どちらのオペランドも一致する生成されたカラムで置換できます。`BETWEEN` および `IN()` の場合、最初の引数のみが一致する生成されたカラムで置換でき、他の引数の結果タイプは同じである必要があります。`BETWEEN` および `IN()` は、JSON 値を含む比較ではまだサポートされていません。

- 生成されたカラムは、少なくとも関数コールまたは前述のいずれかの演算子を含む式として定義する必要があります。式は、別のカラムへの単純な参照で構成できません。たとえば、`gc INT AS (f1) STORED` はカラム参照のみで構成されるため、`gc` のインデックスは考慮されません。
- 引用符で囲まれた文字列を返す JSON 関数の値を計算するインデックス付き生成カラムと文字列を比較するには、関数値から余分な引用符を削除するために、カラム定義に `JSON_UNQUOTE()` が必要です。(文字列と関数の結果を直接比較する場合、JSON コンパレータは引用符の削除を処理しますが、これはインデックスルックアップでは発生しません。)たとえば、次のようなカラム定義を記述するかわりに:

```
doc_name TEXT AS (JSON_EXTRACT(jdoc, '$.name')) STORED
```

次のように記述します:

```
doc_name TEXT AS (JSON_UNQUOTE(JSON_EXTRACT(jdoc, '$.name'))) STORED
```

後者の定義では、オプティマイザは次の両方の比較で一致を検出できます:

```
... WHERE JSON_EXTRACT(jdoc, '$.name') = 'some_string' ...
... WHERE JSON_UNQUOTE(JSON_EXTRACT(jdoc, '$.name')) = 'some_string' ...
```

カラム定義に `JSON_UNQUOTE()` が指定されていない場合、オプティマイザはこれらの最初の比較に対してのみ一致を検出します。

- オプティマイザが間違ったインデックスを選択した場合は、インデックスヒントを使用して無効にし、オプティマイザに別の選択を強制できます。

8.3.12 不可視のインデックス

MySQL では、不可視のインデックス (オプティマイザで使用されないインデックス) がサポートされています。この機能は、主キー以外のインデックス (明示的または暗黙的) に適用されます。

インデックスはデフォルトで可視化されます。新しいインデックスの可視性を明示的に制御するには、`CREATE TABLE`、`CREATE INDEX` または `ALTER TABLE` のインデックス定義の一部として `VISIBLE` または `INVISIBLE` キーワードを使用します:

```
CREATE TABLE t1 (
  i INT,
  j INT,
  k INT,
  INDEX i_idx (i) INVISIBLE
) ENGINE = InnoDB;
CREATE INDEX j_idx ON t1 (j) INVISIBLE;
ALTER TABLE t1 ADD INDEX k_idx (k) INVISIBLE;
```

既存のインデックスの可視性を変更するには、`ALTER TABLE ... ALTER INDEX` 操作で `VISIBLE` または `INVISIBLE` キーワードを使用します:

```
ALTER TABLE t1 ALTER INDEX i_idx INVISIBLE;
ALTER TABLE t1 ALTER INDEX i_idx VISIBLE;
```

インデックスが可視か不可視かに関する情報は、`INFORMATION_SCHEMA.STATISTICS` テーブルまたは `SHOW INDEX` 出力から入手できます。例:

```
mysql> SELECT INDEX_NAME, IS_VISIBLE
FROM INFORMATION_SCHEMA.STATISTICS
WHERE TABLE_SCHEMA = 'db1' AND TABLE_NAME = 't1';
+-----+-----+
| INDEX_NAME | IS_VISIBLE |
+-----+-----+
| i_idx     | YES       |
| j_idx     | NO        |
```

```
| k_idx | NO |
+-----+-----+
```

不可視のインデックスを使用すると、インデックスが必要になった場合に元に戻す必要がある破壊的な変更を行わずに、クエリーのパフォーマンスに対するインデックスの削除の影響をテストできます。大規模なテーブルの場合、インデックスを削除して再追加するとコストがかかる可能性があります。インデックスを非表示にして可視にすると、高速なインプレース操作になります。

インデックスが実際に不可視になった場合、またはオプティマイザによって使用された場合は、そのインデックスが存在しないことがテーブルのクエリーに与える影響に注意する方法がいくつかあります：

- 不可視インデックスを参照するインデックスヒントを含むクエリーでは、エラーが発生します。
- パフォーマンススキーマデータは、影響を受けるクエリーのワークロードの増加を示します。
- クエリーには、異なる **EXPLAIN** 実行計画があります。
- クエリーは、以前に表示されていなかったスロークエリーログに表示されます。

optimizer_switch システム変数の **use_invisible_indexes** フラグは、オプティマイザがクエリー実行計画の構成に不可視のインデックスを使用するかどうかを制御します。フラグが **off** (デフォルト) の場合、オプティマイザは不可視のインデックス (このフラグの導入前と同じ動作) を無視します。フラグが **on** の場合、不可視のインデックスは不可視のままですが、オプティマイザは実行計画の構成を考慮に入れます。

SET VAR オプティマイザヒントを使用して一時的に **optimizer_switch** の値を更新すると、次のように、単一のクエリーの間のみ不可視インデックスを有効にできます：

```
mysql> EXPLAIN SELECT /*+ SET_VAR(optimizer_switch = 'use_invisible_indexes=on') */
> i, j FROM t1 WHERE j >= 50\G
***** 1. row *****
id: 1
select_type: SIMPLE
table: t1
partitions: NULL
type: range
possible_keys: j_idx
key: j_idx
key_len: 5
ref: NULL
rows: 2
filtered: 100.00
Extra: Using index condition
```

```
mysql> EXPLAIN SELECT i, j FROM t1 WHERE j >= 50\G
***** 1. row *****
id: 1
select_type: SIMPLE
table: t1
partitions: NULL
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 5
filtered: 33.33
Extra: Using where
```

インデックスの可視性は、インデックスメンテナンスには影響しません。たとえば、インデックスはテーブルの行に対する変更ごとに引き続き更新され、一意のインデックスによって、インデックスが可視か不可視かに関係なく、コラムへの重複の挿入が防止されます。

明示的な主キーを持たないテーブルは、**NOT NULL** コラムに **UNIQUE** インデックスがある場合でも、有効な暗黙的主キーを持つことができます。この場合、最初のこのようなインデックスは明示的な主キーと同じ制約をテーブルの行に配置し、そのインデックスを不可視にすることはできません。次のテーブル定義について考えてみます：

```
CREATE TABLE t2 (
  i INT NOT NULL,
```



```
j INT NOT NULL,
UNIQUE j_idx (j)
) ENGINE = InnoDB;
```

この定義には明示的な主キーは含まれていませんが、**NOT NULL** カラム **j** のインデックスは主キーと同じ制約を行に設定し、非表示にすることはできません:

```
mysql> ALTER TABLE t2 ALTER INDEX j_idx INVISIBLE;
ERROR 3522 (HY000): A primary key index cannot be invisible.
```

ここで、明示的な主キーがテーブルに追加されるとします:

```
ALTER TABLE t2 ADD PRIMARY KEY (i);
```

明示的な主キーは非表示にできません。また、**j** の一意インデックスは暗黙の主キーとして機能しなくなり、その結果非表示になる可能性があります:

```
mysql> ALTER TABLE t2 ALTER INDEX j_idx INVISIBLE;
Query OK, 0 rows affected (0.03 sec)
```

8.3.13 降順インデックス

MySQL は降順インデックスをサポートしています: インデックス定義内の **DESC** は無視されなくなりましたが、キー値が降順で格納されます。以前は、インデックスを逆の順序でスキャンできましたが、パフォーマンスが低下していました。降順インデックスは順にスキャンできるため、より効率的です。降順インデックスを使用すると、オプティマイザが複数カラムインデックスを使用できるようになります (最も効率的なスキャン順序で一部のカラムに昇順が混在し、他のカラムに降順が混在している場合)。

カラムの昇順インデックスと降順インデックスの様々な組合せに対する 2 つのカラムと 4 つの 2 カラムインデックス定義を含む、次のテーブル定義について考えてみます:

```
CREATE TABLE t (
  c1 INT, c2 INT,
  INDEX idx1 (c1 ASC, c2 ASC),
  INDEX idx2 (c1 ASC, c2 DESC),
  INDEX idx3 (c1 DESC, c2 ASC),
  INDEX idx4 (c1 DESC, c2 DESC)
);
```

テーブル定義の結果、4 つの個別インデックスが作成されます。オプティマイザは、各 **ORDER BY** 句に対して正引きインデックススキャンを実行できるため、**filesort** 操作を使用する必要はありません:

```
ORDER BY c1 ASC, c2 ASC -- optimizer can use idx1
ORDER BY c1 DESC, c2 DESC -- optimizer can use idx4
ORDER BY c1 ASC, c2 DESC -- optimizer can use idx2
ORDER BY c1 DESC, c2 ASC -- optimizer can use idx3
```

降順インデックスの使用には、次の条件があります:

- 降順インデックスは **InnoDB** ストレージエンジンでのみサポートされますが、次の制限があります:
 - インデックスに降順のインデックスキーカラムが含まれている場合、または主キーに降順のインデックスカラムが含まれている場合、変更バッファリングはセカンダリインデックスではサポートされません。
 - InnoDB** SQL パーサーは降順インデックスを使用しません。 **InnoDB** 全文検索の場合、これは、インデックス付きテーブルの **FTS_DOC_ID** カラムに必要なインデックスを降順インデックスとして定義できないことを意味します。詳細は、[セクション15.6.2.4 「InnoDB FULLTEXT インデックス」](#) を参照してください。
- 降順インデックスは、昇順インデックスが使用可能なすべてのデータ型でサポートされています。
- 降順インデックスは、通常の (生成されない) カラムおよび生成されるカラム (**VIRTUAL** と **STORED** の両方) でサポートされます。
- DISTINCT** では、降順のキー部分など、一致するカラムを含む任意のインデックスを使用できます。
- 降順のキー部分を持つインデックスは、集計関数を呼び出す **GROUP BY** 句を持たないクエリーの **MIN()/MAX()** 最適化には使用されません。

- 降順インデックスは **BTREE** ではサポートされていますが、**HASH** インデックスではサポートされていません。降順インデックスは、**FULLTEXT** または **SPATIAL** インデックスではサポートされていません。

HASH、**FULLTEXT** および **SPATIAL** インデックスに対して明示的に指定された **ASC** および **DESC** の指定では、エラーが発生します。

8.3.14 TIMESTAMP カラムからのインデックス付きルックアップ

一時値は UTC 値として **TIMESTAMP** カラムに格納され、**TIMESTAMP** カラムに対して挿入および取得された値はセッションタイムゾーンと UTC の間で変換されます。(これは、**CONVERT_TZ()** 関数によって実行される変換と同じタイプです。セッションのタイムゾーンが UTC の場合、事実上タイムゾーンの変換は行われません。)

夏時間 (DST) などのローカルタイムゾーンの変更の規則により、UTC タイムゾーンと UTC 以外のタイムゾーン間の変換は両方向で 1 対 1 ではありません。個別の UTC 値は、別のタイムゾーンでは区別できません。次の例は、UTC 以外のタイムゾーンで同一になる個別の UTC 値を示しています:

```
mysql> CREATE TABLE tstable (ts TIMESTAMP);
mysql> SET time_zone = 'UTC'; -- insert UTC values
mysql> INSERT INTO tstable VALUES
    ('2018-10-28 00:30:00'),
    ('2018-10-28 01:30:00');
mysql> SELECT ts FROM tstable;
+-----+
| ts          |
+-----+
| 2018-10-28 00:30:00 |
| 2018-10-28 01:30:00 |
+-----+
mysql> SET time_zone = 'MET'; -- retrieve non-UTC values
mysql> SELECT ts FROM tstable;
+-----+
| ts          |
+-----+
| 2018-10-28 02:30:00 |
| 2018-10-28 02:30:00 |
+-----+
```

注記

'MET'や'Europe/Amsterdam'などの名前付きタイムゾーンを使用するには、タイムゾーンテーブルが適切に設定されている必要があります。その手順は、[セクション 5.1.15 「MySQL Server でのタイムゾーンのサポート」](#)を参照してください。

'MET'タイムゾーンに変換すると、2 つの異なる UTC 値が同じであることがわかります。この現象は、オプティマイザがインデックスを使用してクエリーを実行するかどうかに応じて、特定の **TIMESTAMP** カラムクエリーの結果が異なる可能性があります。

クエリーで、**WHERE** 句を使用して **ts** カラムでユーザー指定のタイムスタンプリテラルなどの単一の特定の値を検索するために、前述のテーブルから値を選択するとします:

```
SELECT ts FROM tstable
WHERE ts = 'literal';
```

さらに、クエリーが次の条件下で実行されるとします:

- セッションのタイムゾーンは UTC ではなく、DST シフトがあります。例:

```
SET time_zone = 'MET';
```

- DST シフトのため、**TIMESTAMP** カラムに格納されている一意の UTC 値がセッションタイムゾーンで一意ではありません。(前述の例は、これがどのように発生するかを示しています。)
- クエリーでは、セッションタイムゾーンの DST への入力時間内の検索値を指定します。

これらの条件では、インデックス付けされていない参照とインデックス付けされた参照で **WHERE** 句の比較が様々な方法で行われ、結果が異なります:

- インデックスがない場合、またはオプティマイザがインデックスを使用できない場合は、セッションのタイムゾーンで比較が行われます。オプティマイザは、各 `ts` カラム値を取得し、UTC からセッションタイムゾーンに変換して、検索値と比較するテーブルスキャンを実行します (セッションタイムゾーンでも解釈されます):

```
mysql> SELECT ts FROM tstable
        WHERE ts = '2018-10-28 02:30:00';
+-----+
| ts          |
+-----+
| 2018-10-28 02:30:00 |
| 2018-10-28 02:30:00 |
+-----+
```

格納された `ts` 値はセッションタイムゾーンに変換されるため、UTC 値とは異なるが、セッションタイムゾーンでは等しい 2 つのタイムスタンプ値をクエリーで返すことができます: クロックが変更されたときに DST シフトの前に発生する値と、DST シフトの後に発生した値。

- 使用可能なインデックスがある場合、UTC で比較が行われます。オプティマイザはインデックススキャンを実行し、最初に検索値をセッションタイムゾーンから UTC に変換してから、結果を UTC インデックスエントリと比較します:

```
mysql> ALTER TABLE tstable ADD INDEX (ts);
mysql> SELECT ts FROM tstable
        WHERE ts = '2018-10-28 02:30:00';
+-----+
| ts          |
+-----+
| 2018-10-28 02:30:00 |
+-----+
```

この場合、(変換された) 検索値はインデックスエントリにのみ一致し、格納されている個別の UTC 値のインデックスエントリも個別であるため、検索値はそれらのいずれかにのみ一致できます。

インデックス付けされていないルックアップとインデックス付けされたルックアップのオプティマイザ操作が異なるため、クエリーではそれぞれ異なる結果が生成されます。インデックス付けされていない参照の結果は、セッションタイムゾーンで一致するすべての値を返します。インデックス付きルックアップではこれを実行できません:

- UTC 値についてのみ認識されるストレージエンジン内で実行されます。
- 同じ UTC 値にマップされる 2 つの異なるセッションタイムゾーン値の場合、インデックス付き参照は対応する UTC インデックスエントリにのみ一致し、単一行のみを返します。

前述の説明では、`tstable` に格納されているデータセットは個別の UTC 値で構成されています。このような場合、表示された形式のすべてのインデックス使用クエリーは、最大 1 つのインデックスエントリに一致します。

インデックスが `UNIQUE` でない場合は、テーブル (およびインデックス) に特定の UTC 値の複数のインスタンスを格納できます。たとえば、`ts` カラムに UTC 値 '2018-10-28 00:30:00' の複数のインスタンスが含まれる場合があります。この場合、インデックス使用クエリーはそれぞれを戻します (結果セットで MET 値 '2018-10-28 02:30:00' に変換されます)。インデックス使用クエリーは、セッションタイムゾーンの検索値に変換される複数の UTC 値を照合するのではなく、変換された検索値を UTC インデックスエントリの単一の値に一致させることができます。

セッションタイムゾーンに一致するすべての `ts` 値を返すことが重要な場合、回避策は `IGNORE INDEX` ヒントでインデックスを使用しないようにすることです:

```
mysql> SELECT ts FROM tstable
        IGNORE INDEX (ts)
        WHERE ts = '2018-10-28 02:30:00';
+-----+
| ts          |
+-----+
| 2018-10-28 02:30:00 |
| 2018-10-28 02:30:00 |
+-----+
```

`FROM_UNIXTIME()` 関数や `UNIX_TIMESTAMP()` 関数で実行される変換など、両方の方向でのタイムゾーン変換に同じ一対一マッピングがないことが他のコンテキストでも発生します。セクション 12.7 「日付および時間関数」を参照してください。

8.4 データベース構造の最適化

データベース設計者としての役割では、スキーマ、テーブル、およびカラムを編成するもっとも効率的な方法を探します。アプリケーションコードをチューニングする場合、I/O を最小にし、関連項目をまとめて、データボリュームが増加してもパフォーマンスを高く維持するように、事前に計画します。効率的なデータベース設計から始めることで、チームメンバーは高性能のアプリケーションコードを簡単に書けるようになり、アプリケーションが発展して、書き換えられても、データベースを持ちこたえさせる可能性が高くなります。

8.4.1 データサイズの最適化

ディスク上の領域を最小にするようにテーブルを設計します。これにより、ディスクに対して読み取りおよび書き込みされるデータの量が減ることで、大幅な改善が見られます。内容がクエリー実行中にアクティブに処理される間、テーブルが小さいほど、通常必要なメインメモリーの量は少なくなります。テーブルデータの領域の削減により、インデックスも小さくなり、高速に処理できます。

MySQL は多数のさまざまなストレージエンジン (テーブル型) と行フォーマットをサポートしています。テーブルごとに、使用するストレージとインデックス設定方法を決定できます。アプリケーションに適切なテーブル形式を選択することで、大幅なパフォーマンスの向上が得られることがあります。第15章「InnoDB ストレージエンジン」および第16章「代替ストレージエンジン」を参照してください。

ここで挙げられた技法を使用して、テーブルのパフォーマンス改善とストレージ領域の最小化を図ることができます。

- [テーブルカラム](#)
- [行フォーマット](#)
- [インデックス](#)
- [結合](#)
- [正規化](#)

テーブルカラム

- 可能なかぎりもっとも効率的 (最小) のデータ型を使用します。MySQL にはディスク領域とメモリーを節約する多くの専用の型があります。たとえば、可能な場合は、小さなテーブルを取得するために、小さな整数型を使用します。MEDIUMINT カラムが使用する領域は 25% 少ないため、MEDIUMINT は多くの場合に INT より適切な選択肢です。
- 可能な場合は、カラムを NOT NULL として宣言します。これにより、インデックスを適切に使用し、各値が NULL であるかどうかをテストするためのオーバーヘッドがなくなることで、SQL の操作が速くなります。カラムあたり 1 ビットでいくらかのストレージ領域も節約します。テーブルで実際に NULL 値が必要な場合、それらを使用します。単にすべてのカラムで NULL 値を許可するデフォルトの設定を避けます。

行フォーマット

- InnoDB テーブルは、デフォルトでは DYNAMIC 行形式を使用して作成されます。DYNAMIC 以外の行フォーマットを使用するには、innodb_default_row_format を構成するか、CREATE TABLE または ALTER TABLE ステートメントで ROW_FORMAT オプションを明示的に指定します。

COMPACT、DYNAMIC および COMPRESSED を含む行形式のコンパクトファミリでは、一部の操作で CPU 使用率を増やすコストで行の記憶領域が削減されます。ワークロードが、キャッシュヒット率とディスク速度によって制限される通常のワークロードであれば、速くなる可能性があります。CPU 速度によって制限されるまれな例では、遅くなる可能性があります。

行形式のコンパクトファミリでは、utf8mb3 や utf8mb4 などの可変長文字セットを使用する場合にも、CHAR カラムの格納が最適化されます。ROW_FORMAT=REDUNDANT では、CHAR(N) は N×文字セットの最大バイト長を占有します。多くの言語は主にシングルバイトの utf8 文字を使用して記述できるため、固定記憶域の長さによって領域が無駄になることがよくあります。行フォーマットのコンパクトファミリでは、InnoDB は、末尾の空白を削除することで、N の範囲内の可変量の記憶域を N×これらのカラムの文字セットの最大バイト長に割り当てま

す。一般的な場合にインプレース更新を容易にするために、記憶域の最小長は N バイトです。詳細は、[セクション15.10「InnoDB の行フォーマット」](#)を参照してください。

- テーブルデータを圧縮形式で保存することで、さらに領域を最小にするには、InnoDB テーブルを作成する際に `ROW_FORMAT=COMPRESSED` を指定するか、既存の MyISAM テーブルに対して、`myisampack` コマンドを実行します。(InnoDB 圧縮テーブルは読み取りおよび書き込み可能ですが、MyISAM 圧縮テーブルは読み取り専用です。)
- MyISAM テーブルで、可変長カラム (`VARCHAR`、`TEXT`、あるいは `BLOB` など) がない場合は、固定サイズ行フォーマットが使用されます。これは高速ですが、いくらか領域を無駄にすることがあります。[セクション16.2.3「MyISAM テーブルのストレージフォーマット」](#)を参照してください。`CREATE TABLE` オプション `ROW_FORMAT=FIXED` によって、`VARCHAR` カラムがある場合でも、固定長の行を必要としていることを伝えることができます。

インデックス

- テーブルのプライマリインデックスは可能な限り短くしてください。これにより、各行の識別が容易になり効率的になります。InnoDB テーブルの場合、主キーカラムは、各セカンダリインデックスエントリに複製されるため、多数のセカンダリインデックスがある場合に、短い主キーによって、かなりの領域が節約されます。
- クエリーパフォーマンスを向上するために必要なインデックスのみを作成します。インデックスは取得には有効ですが、挿入および更新操作を遅くします。ほとんどカラムの組み合わせに対して検索することによって、テーブルにアクセスする場合、カラムごとに個別のインデックスを作成するのではなく、それらに対して単一の複合インデックスを作成します。インデックスの最初の部分は、もっとも使用されるカラムにするべきです。テーブルから選択する場合に、常に多くのカラムを使用する場合、適切なインデックスの圧縮を取得するため、インデックスの最初のカラムは、もっとも重複の多いカラムにするべきです。
- 長い文字列のカラムの最初の文字数に一意の接頭辞がある可能性が高い場合は、カラムの左端に索引を作成するための MySQL サポートを使用して、この接頭辞のみを索引付けすることをお勧めします ([セクション13.1.15「CREATE INDEX ステートメント」](#)を参照)。短いインデックスほど速くなるのは、必要なディスク領域が少ないだけでなく、インデックスキャッシュでのヒットが多くなり、そのためにディスクシークが少なくなるためでもあります。[セクション5.1.1「サーバーの構成」](#)を参照してください。

結合

- 状況によって、頻繁にスキャンされるテーブルを2つに分割することで、メリットがある場合があります。これは特に、それが動的形式テーブルで、テーブルのスキャン時に、関連行を見つけるために使用できる小さな静的形式テーブルを使用できる場合に当てはまります。
- 対応するカラムに基づいた結合を高速化するには、異なるテーブル内の同一の情報を持つカラムを同一のデータ型で宣言します。
- 異なるテーブルで同じ名前を使用し、結合クエリーを簡略化できるように、カラム名を簡単にします。たとえば、`customer` というテーブルでは `customer_name` ではなく `name` のカラム名を使用します。名前をほかの SQL サーバーに移植できるようにするため、18文字より短くすることを考慮します。

正規化

- 通常、すべてのデータを非冗長に維持しようとしてください(データベース理論で第3正規形と呼ばれるものを順守します)。名前や住所などの長い値を繰り返す代わりに、それらに一意の ID を割り当て、複数の小さなテーブルで必要なだけこれらの ID を繰り返し、結合句で ID を参照して、クエリーでテーブルを結合します。
- たとえば、大きなテーブルからすべてのデータを解析するビジネスインテリジェンスシナリオなどで、ディスク領域やデータの複数のコピーを維持する保守コストより、速度の方が重要である場合、正規化ルールを緩和して、情報を複製したり、サマリーテーブルを作成したりして、速度を向上させることができます。

8.4.2 MySQL データ型の最適化

8.4.2.1 数値データの最適化

- 文字列または数値として表すことができる一意の ID やその他の値の場合、文字列カラムより数値カラムをお勧めします。大きな数値は、対応する文字列より少ないバイト数で格納できるため、それらの転送と比較が高速になり、使用するメモリーが少なくなります。

- 数値データを使用している場合、多くの場合にテキストファイルにアクセスするより、ライブ接続を使用して、データベースから情報にアクセスする方が高速です。データベース内の情報はテキストファイルよりコンパクトなフォーマットで格納される可能性が高いため、それへのアクセスにかかわるディスクアクセスが少なくなります。また、テキストファイルを解析して、行とカラムの境界を見つけることを回避できるため、アプリケーションのコードも節約できます。

8.4.2.2 文字および文字列型の最適化

文字および文字列カラムの場合、次のガイドラインに従います。

- 言語固有の照合機能が必要でない場合は、比較およびソート操作を速くするため、バイナリ照合順序を使用します。特定のクエリー内でバイナリ照合順序を使用するには、[BINARY](#) 演算子を使用できます。
- さまざまなカラムの値を比較する場合、可能なかぎり、それらのカラムを同じ文字セットと照合順序で宣言し、クエリー実行中の文字列変換を避けます。
- サイズが 8K バイト未満のカラム値では、[BLOB](#) の代わりにバイナリ [VARCHAR](#) を使用します。[GROUP BY](#) および [ORDER BY](#) 句は一時テーブルを生成する可能性があり、これらの一時テーブルでは、元のテーブルに [BLOB](#) カラムが含まれない場合に、[MEMORY](#) ストレージエンジンを使用することがあります。
- テーブルに名前や住所などの文字列カラムが含まれるが、多くのクエリーでそれらのカラムを取得しない場合、文字列カラムを個別のテーブルに分割し、必要に応じて、外部キーで結合クエリーを使用することを考慮します。MySQL で行から何らかの値を取得する場合、その行 (およびおそらくその他の隣接する行) のすべてのカラムを含むデータブロックを読み取ります。もっとも頻繁に使用するカラムのみで、各行を小さくすることで、より多くの行を各データブロックに収めることができます。そのようなコンパクトなテーブルは、一般的なクエリーのディスク I/O やメモリーの使用量を削減します。
- [InnoDB](#) テーブルで主キーとして、ランダムに生成された値を使用する場合、可能であれば、現在の日時などの降順の値でプリフィクスを付けます。連続したプライマリ値が、相互に物理的に近くに保存されていれば、[InnoDB](#) はそれらを高速に挿入し、取得できます。
- 数値カラムの方が通常同等の文字列カラムより推奨される理由については、[セクション 8.4.2.1 「数値データの最適化」](#) を参照してください。

8.4.2.3 BLOB 型の最適化

- テキストデータを格納する大きな BLOB を保存する場合、まずそれを圧縮することを考慮します。テーブル全体が [InnoDB](#) または [MyISAM](#) によって圧縮されている場合は、この技法を使用しないでください。
- 複数のカラムのあるテーブルで、BLOB カラムを使用しないクエリーのメモリー要件を削減するには、BLOB カラムを個別のテーブルに分割し、必要に応じて、結合クエリーでそれを参照することを考慮します。
- BLOB 値を取得し、表示するためのパフォーマンス要件は、ほかのデータ型と大きく異なることがあるため、BLOB 固有テーブルを別のストレージデバイスまたは個別のデータベースインスタンスに置くことができます。たとえば、BLOB を取得するには、大量の順次ディスク読み取りが必要で、[SSD デバイス](#) より、従来のハードドライブの方が適しています。
- バイナリ [VARCHAR](#) カラムの方が同等の BLOB カラムより推奨されることがある理由については、[セクション 8.4.2.2 「文字および文字列型の最適化」](#) を参照してください。
- きわめて長いテキスト文字列に対して、同等性をテストする代わりに、個別のカラムにカラムのハッシュを格納し、そのカラムにインデックスを設定して、クエリー内のハッシュ値をテストします。([MD5\(\)](#) または [CRC32\(\)](#) 関数を使用して、ハッシュ値を生成します。) ハッシュ関数は、異なる入力で重複した結果を生成することがあるため、引き続きクエリーに句 [AND blob_column = long_string_value](#) を含めて、誤った一致に対して保護します。パフォーマンスは、ハッシュ値の小さく、簡単にスキャンされるインデックスからメリットが得られます。

8.4.3 多数のテーブルの最適化

各クエリーを高速にするいくつかの技法には、多数のテーブルへのデータの分割が含まれます。テーブルの数が数千または数百万にもなる場合、これらすべてのテーブルの処理のオーバーヘッドは新たなパフォーマンスの考慮事項になります。

8.4.3.1 MySQL でのテーブルのオープンとクローズの方法

`mysqladmin status` コマンドを実行すると、次のように表示されるはずです。

```
Uptime: 426 Running threads: 1 Questions: 11082
Reloads: 1 Open tables: 12
```

12 未満のテーブルがある場合、`Open tables` 値の 12 はある程度パズルになる可能性があります。

MySQL はマルチスレッド化されているため、特定のテーブルに対するクエリーを同時に発行するクライアントが多数存在する場合があります。同じテーブルに対して、複数のクライアントセッションが異なる状態を持つ問題を最小にするため、テーブルは各同時セッションに独立して開かれます。これは追加メモリーを使用しますが、一般にパフォーマンスは向上します。`MyISAM` テーブルでは、テーブルを開いているクライアントごとに、データファイルに 1 つの追加のファイルディスクリプタが必要になります。(対照的に、インデックスファイルディスクリプタはすべてのセッションで共有されます。)

`table_open_cache` および `max_connections` システム変数は、サーバーが開いたままにするファイルの最大数に影響します。これらの値のいずれかまたは両方を増やすと、オープンファイルディスクリプタのプロセスあたりの数に関して、オペレーティングシステムによって適用されている制限に達する可能性があります。多くのオペレーティングシステムでは、オープンファイル制限を増やすことができますが、方法はシステムによって大きく異なります。制限値を増やすことができるかどうか、およびその実行方法については、使用するオペレーティングシステムのドキュメントを参照してください。

`table_open_cache` は `max_connections` に関連します。たとえば、200 の同時実行接続の場合、少なくとも $200 * N$ のテーブルキャッシュサイズを指定します。ここで N は実行するクエリーの結合あたりのテーブルの最大数です。また、一時テーブルとファイル用のいくつかの追加のファイルディスクリプタを予約する必要もあります。

オペレーティングシステムで、`table_open_cache` の設定に示されたオープンファイルディスクリプタの数を処理できることを確認してください。`table_open_cache` の設定が高すぎると、MySQL でファイル記述子が不足し、接続の拒否やクエリーの実行の失敗などの症状が発生する可能性があります。

また、`MyISAM` ストレージエンジンでは、一意のオープンテーブルごとに 2 つのファイル記述子が必要であることも考慮してください。MySQL で使用可能なファイル記述子の数を増やすには、`open_files_limit` システム変数を設定します。セクション B.3.2.16 「ファイルが見つからず同様のエラーが発生しました」を参照してください。

オープンテーブルのキャッシュは、`table_open_cache` エントリのレベルで保持されます。サーバーはスタートアップ時にキャッシュサイズを自動サイズ設定します。サイズを明示的に設定するには、スタートアップ時に `table_open_cache` システム変数を設定します。MySQL では、このセクションの後半で説明するように、クエリーを実行するために一時的にこれよりも多くのテーブルを開く場合があります。

次の状況では、MySQL は未使用のテーブルを閉じ、それをテーブルキャッシュから削除します。

- キャッシュがいっぱいで、スレッドがキャッシュにないテーブルを開こうとした場合。
- キャッシュに `table_open_cache` を超えるエントリがあり、キャッシュ内のテーブルがどのスレッドによっても使用されなくなった場合。
- テーブルフラッシュ操作が発生した場合。これは、だれかが `FLUSH TABLES` ステートメントを発行するか、または `mysqladmin flush-tables` または `mysqladmin refresh` コマンドを実行した場合に行われます。

テーブルキャッシュがいっぱいになると、サーバーは次の手順に従って使用するキャッシュエントリを見つけます。

- 現在使用されていないテーブルは、最も最近使用されていないテーブルからリリースされます。
- 新しいテーブルを開く必要があるが、キャッシュがいっぱいでテーブルを解放できない場合、キャッシュは必要に応じて一時的に拡張されます。キャッシュが一時的に拡張された状況で、テーブルが使用中から未使用状態になったときは、そのテーブルが閉じられ、キャッシュから解放されます。

`MyISAM` テーブルは同時アクセスごとに開かれます。つまり、2 つのスレッドで同じテーブルにアクセスする場合、または 1 つのスレッドが同一クエリーでテーブルに 2 回アクセスする場合 (テーブルをそれ自体に結合することによってなど) は、テーブルを 2 回開く必要があることを意味します。同時オープン、それぞれテーブルキャッシュにエントリが必要になります。いずれかの `MyISAM` テーブルを最初に開くと、データファイルに 1 つとインデックスファイルに 1 つの 2 つのファイルディスクリプタが必要になります。テーブルの追加の使用では、それぞれデータファイルに 1 つだけのファイルディスクリプタが必要です。インデックスファイルディスクリプタはすべてのスレッドで共有されます。

`HANDLER tbl_name OPEN` ステートメントを使用してテーブルを開く場合、専用のテーブルオブジェクトがスレッドに割り当てられます。このテーブルオブジェクトはほかのスレッドと共有されず、スレッドが `HANDLER tbl_name CLOSE` を呼び出すか、スレッドが終了するまでクローズされません。これが発生すると、テーブルがテーブルキャッシュに戻されます (キャッシュがいっぱいでない場合)。セクション13.2.4「`HANDLER ステートメント`」を参照してください。

テーブルキャッシュが小さすぎるかどうかを判断するには、サーバーの起動後のテーブルオープン操作の数を示す `Opened_tables` ステータス変数を確認します:

```
mysql> SHOW GLOBAL STATUS LIKE 'Opened_tables';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Opened_tables | 2741  |
+-----+-----+
```

多数の `FLUSH TABLES` ステートメントを発行していない場合でも、値が非常に大きいか急速に増加する場合は、サーバーの起動時に `table_open_cache` 値を増やします。

8.4.3.2 同じデータベースに大量のテーブルを作成することの短所

同じデータベースディレクトリに多数の `MyISAM` テーブルがある場合、オープン、クローズ、および作成操作が遅くなります。多数のテーブルに対して `SELECT` ステートメントを実行した場合、開く必要があるテーブルごとに、別のテーブルを開じる必要があるため、テーブルキャッシュがいっぱいの場合にオーバーヘッドが少し発生します。テーブルキャッシュで許可されるエントリ数を増やすことによって、このオーバーヘッドを減らすことができます。

8.4.4 MySQL での内部一時テーブルの使用

場合によっては、サーバーはステートメントの処理中に内部一時テーブルを作成します。これが発生した場合、ユーザーは直接的に制御できません。

サーバーは、次のような条件下で一時テーブルを作成します:

- `UNION` ステートメントの評価 (ただし、後で説明するいくつかの例外があります)。
- `TEMPTABLE` アルゴリズム、`UNION` または集計を使用するビューなど、一部のビューの評価。
- 導出テーブルの評価 (セクション13.2.11.8「`導出テーブル`」を参照)。
- 共通テーブル式の評価 (セクション13.2.15「`WITH (共通テーブル式)`」を参照)。
- サブクエリーまたは準結合の実体化用に作成されたテーブル (セクション8.2.2「`サブクエリー、導出テーブル、ビュー参照および共通テーブル式の最適化`」を参照)。
- `ORDER BY` 句と異なる `GROUP BY` 句を含むステートメント、または結合キューの最初のテーブル以外のテーブルのカラムが `ORDER BY` または `GROUP BY` に含まれるステートメントの評価。
- `ORDER BY` と組み合わせた `DISTINCT` の評価には、一時テーブルが必要になる場合があります。
- `SQL_SMALL_RESULT` 修飾子を使用するクエリーの場合、ディスク上の記憶域を必要とする要素 (後述) もクエリーに含まれていないかぎり、MySQL はインメモリー一時テーブルを使用します。
- 同じテーブルから選択して同じテーブルに挿入する `INSERT ... SELECT` ステートメントを評価するために、MySQL は `SELECT` の行を保持する内部一時テーブルを作成し、それらの行をターゲットテーブルに挿入します。セクション13.2.6.1「`INSERT ... SELECT ステートメント`」を参照してください。
- 複数テーブルの `UPDATE` ステートメントの評価。
- `GROUP_CONCAT()` または `COUNT(DISTINCT)` 式の評価。
- ウィンドウ関数の評価 (セクション12.21「`ウィンドウ関数`」を参照) では、必要に応じて一時テーブルが使用されます。

ステートメントに一時テーブルが必要かどうかを判断するには、`EXPLAIN` を使用し、`Extra` カラムをチェックして、`Using temporary` と表示されているかどうかを確認します (セクション8.8.1「`EXPLAIN によるクエリーの最適`

化」を参照)。EXPLAIN では、導出一時テーブルまたは実体化一時テーブルの場合、必ずしも Using temporary とは言えません。ウィンドウ関数を使用するステートメントの場合、EXPLAIN と FORMAT=JSON では常にウィンドウステップに関する情報が提供されます。ウィンドウ関数で一時テーブルが使用されている場合は、ステップごとに示されます。

一部のクエリー条件では、インメモリー一時テーブルを使用できません。この場合、サーバーは代わりにディスク上のテーブルを使用します:

- テーブル内の BLOB または TEXT カラムの存在 ただし、MySQL 8.0 のインメモリー内部一時テーブルのデフォルトのストレージエンジンである TempTable ストレージエンジンは、MySQL 8.0.13 の時点でバイナリラージオブジェクト型をサポートしています。内部一時テーブルストレージエンジンを参照してください。
- UNION または UNION ALL が使用された場合に、SELECT リスト内の 512 (バイナリ文字列の場合はバイト数、非バイナリ文字列の場合は文字数) より大きい最大長を持つ文字列カラムの存在。
- SHOW COLUMNS および DESCRIBE ステートメントでは、一部のカラムのタイプとして BLOB が使用されるため、結果に使用される一時テーブルはディスク上のテーブルです。

サーバーは、特定の条件を満たす UNION ステートメントに一時テーブルを使用しません。かわりに、結果カラムの型キャストの実行に必要なデータ構造のみが一時テーブルの作成から保持されます。テーブルは完全にはインスタンス化されず、テーブルに対する書き込みまたは読取りは行われません。行はクライアントに直接送信されます。最後のクエリーブロックが実行されるまでサーバーが待機する必要がないため、結果としてメモリーおよびディスクの要件が削減され、最初の行がクライアントに送信されるまでの遅延が短くなります。EXPLAIN およびオプティマイザのトレース出力には、この実行計画が反映されます: UNION RESULT クエリーブロックは、一時テーブルから読み取る部分に対応しているため、存在しません。

次の条件は、一時テーブルを使用しない UNION の評価に適しています:

- 共用体は、UNION や UNION DISTINCT ではなく、UNION ALL です。
- グローバル ORDER BY 句がありません。
- 共用体は、{INSERT | REPLACE} ... SELECT ... ステートメントのトップレベルのクエリーブロックではありません。

内部一時テーブルストレージエンジン

内部一時テーブルはメモリー内に保持され、TempTable または MEMORY ストレージエンジンによって処理されるが、InnoDB ストレージエンジンによってディスクに格納されます。

インメモリー内部一時テーブルのストレージエンジン

internal_tmp_mem_storage_engine セッション変数は、インメモリー内部一時テーブルのストレージエンジンを定義します。許可される値は、TempTable (デフォルト) および MEMORY です。

TempTable ストレージエンジンは、VARCHAR および VARBINARY カラム、および MySQL 8.0.13 の時点でのその他のバイナリラージオブジェクト型の効率的なストレージを提供します。

temptable_max_ram 変数は、TempTable ストレージエンジンがメモリーマップ一時ファイルまたは InnoDB ディスク上の内部一時テーブルの形式でディスクから領域の割り当てを開始する前に占有できる RAM の最大量を定義します。デフォルトの temptable_max_ram 設定は 1GiB です。temptable_use_mmap 変数 (MySQL 8.0.16 で導入) は、temptable_max_ram の制限を超えた場合に、TempTable ストレージエンジンがメモリーマップされたファイルまたは InnoDB ディスク上の内部一時テーブルを使用するかどうかを制御します。デフォルト設定は temptable_use_mmap=ON です。MySQL 8.0.23 で導入された temptable_max_mmap 変数は、TempTable ストレージエンジンが内部一時テーブルデータの InnoDB ディスク上の内部一時テーブルへの格納を開始する前に、メモリーマップされたファイルから割り当てることができるメモリーの最大量を定義します。temptable_max_mmap=0 設定では、メモリーマップファイルからの割り当てが無効化され、temptable_use_mmap 設定に関係なく、使用が効率的に無効化されます。

注記

temptable_max_ram 設定では、TempTable ストレージエンジンを使用する各スレッドに割り当てられたスレッドローカルメモリーブロックは考慮されません。スレッドローカルメモ

リーブロックのサイズは、スレッドの最初のメモリー割当てリクエストのサイズによって異なります。リクエストが 1MB 未満の場合 (ほとんどの場合)、スレッドローカルメモリーブロックサイズは 1MB です。リクエストが 1MB を超える場合、スレッドローカルメモリーブロックは初期メモリーリクエストとほぼ同じサイズになります。スレッドローカルメモリーブロックは、スレッドが終了するまでスレッドローカル記憶域に保持されます。

TempTable ストレージエンジンによる内部一時テーブルのオーバーフローメカニズムとしてのメモリーマップ一時ファイルの使用は、次の規則によって制御されます:

- 一時ファイルは、`tmpdir` 変数で定義されたディレクトリに作成されます。
- 一時ファイルは、作成して開いた直後に削除されるため、`tmpdir` ディレクトリには表示されません。一時ファイルが占有する領域は、一時ファイルが開いている間はオペレーティングシステムによって保持されます。領域は、一時ファイルが **TempTable** ストレージエンジンによって閉じられたとき、または `mysqld` プロセスがシャットダウンされたときに再利用されます。
- RAM と一時ファイル間、RAM 内、または一時ファイル間でデータが移動されることはありません。
- `temptable_max_ram` で定義された制限内に領域が使用可能になると、新しいデータが RAM に格納されます。それ以外の場合、新しいデータは一時ファイルに格納されます。
- テーブルの一部のデータが一時ファイルに書き込まれた後に RAM で領域が使用可能になった場合、残りのテーブルデータを RAM に格納できます。

TempTable ストレージエンジンが **InnoDB** ディスク上の内部一時テーブルをオーバーフローメカニズムとして使用するよう構成されている場合 (`temptable_use_mmap=OFF` または `temptable_max_mmap=0`)、`temptable_max_ram` 制限を超えるインメモリーテーブルは **InnoDB** ディスク上の内部一時テーブルに変換され、そのテーブルに属する行はメモリーから **InnoDB** ディスク上の内部一時テーブルに移動されます。MySQL 8.0.16 で削除された `internal_tmp_disk_storage_engine` 設定は、**TempTable** ストレージエンジンオーバーフローメカニズムには影響しません。

MySQL 8.0.23 より前は、**TempTable** ストレージエンジンが `temptable_max_ram` の制限を超え、メモリーマップされたファイル用に一時ディレクトリ内の過剰な領域を使用することが多い場合、**TempTable** オーバーフローメカニズムとして **InnoDB** のディスク上の内部一時テーブルが推奨されていました。MySQL 8.0.23 の時点で、`temptable_max_mmap` 変数は **TempTable** ストレージエンジンがメモリーマップされたファイルから割り当てるメモリー量の制限を定義するため、これらのファイルが使用する領域が多すぎるリスクに対処します。通常、`temptable_max_ram` の制限を超えるのは、大規模な内部一時テーブルの使用または内部一時テーブルの広範囲な使用が原因です。**InnoDB** のディスク上の内部一時テーブルは、デフォルトでデータディレクトリに存在するセッション一時テーブルスペースに作成されます。詳細は、[セクション15.6.3.5「一時テーブルスペース」](#)を参照してください。

インメモリー一時テーブルに **MEMORY** ストレージエンジンを使用する場合、MySQL はインメモリー一時テーブルが大きすぎると、インメモリー一時テーブルをディスク上のテーブルに自動的に変換します。インメモリー一時テーブルの最大サイズは、`tmp_table_size` または `max_heap_table_size` のいずれか小さい方の値によって定義されます。これは、**CREATE TABLE** で明示的に作成される **MEMORY** テーブルとは異なります。このようなテーブルの場合、`max_heap_table_size` 変数のみがテーブルの大きさを決定し、ディスク上の形式への変換はありません。

オンディスク内部一時テーブルのストレージエンジン

MySQL 8.0.16 以降、サーバーは常に **InnoDB** ストレージエンジンを使用してディスク上の内部一時テーブルを管理します。

MySQL 8.0.15 以前では、`internal_tmp_disk_storage_engine` 変数を使用して、ディスク上の内部一時テーブルに使用されるストレージエンジンを定義していました。この変数は MySQL 8.0.16 で削除され、この目的に使用されるストレージエンジンはユーザーが構成できなくなりました。

MySQL 8.0.15 以前: 共通テーブル式 (CTE) の場合、ディスク上の内部一時テーブルに使用されるストレージエンジンを **MyISAM** にすることはできません。`internal_tmp_disk_storage_engine=MYISAM` の場合、ディスク上の一時テーブルを使用して CTE を実体化しようとする、エラーが発生します。

MySQL 8.0.15 以前: `internal_tmp_disk_storage_engine=INNODB` を使用している場合、**InnoDB row or column limits** を超えるディスク上の内部一時テーブルを生成するクエリーは、「行サイズが大きすぎます」または「カラムが多すぎます」エラーを返します。回避策は、`internal_tmp_disk_storage_engine` を **MYISAM** に設定することです。

内部一時テーブル記憶域形式

インメモリ内部一時テーブルが `TempTable` ストレージエンジンによって管理される場合、`VARCHAR` カラム、`VARBINARY` カラムおよびその他のバイナリラージオブジェクト型のカラム (MySQL 8.0.13 時点でサポートされています) を含む行は、セルの配列によってメモリ内にテーブルされ、各セルには `NULL` フラグ、データ長およびデータポイントが含まれます。カラム値は、配列の後の単一のメモリ領域に、パディングなしで連続した順序で配置されます。配列内の各セルは 16 バイトの記憶域を使用します。`TempTable` ストレージエンジンが `temptable_max_ram` 制限を超え、メモリマップされたファイルまたは `InnoDB` ディスク上の内部一時テーブルとしてディスクから領域の割り当てを開始した場合も、同じストレージ形式が適用されます。

インメモリ内部一時テーブルが `MEMORY` ストレージエンジンによって管理される場合、固定長の行形式が使用されます。`VARCHAR` および `VARBINARY` のカラム値は最大カラム長に埋め込まれ、実質的には `CHAR` および `BINARY` のカラムとして格納されます。

MySQL 8.0.16 より前は、ディスク上の内部一時テーブルは `InnoDB` または `MyISAM` ストレージエンジンによって管理されていました (`internal_tmp_disk_storage_engine` の設定によって異なります)。どちらのエンジンも、動的幅の行形式を使用して内部一時テーブルを格納します。カラムには必要な記憶域のみが必要です。これにより、固定長の行を使用するディスク上のテーブルと比較して、ディスク I/O、領域要件および処理時間が短縮されます。MySQL 8.0.16 以降、`internal_tmp_disk_storage_engine` はサポートされず、ディスク上の内部一時テーブルは常に `InnoDB` によって処理されます。

`MEMORY` ストレージエンジンを使用する場合、ステートメントは最初にインメモリ内部一時テーブルを作成し、テーブルが大きすぎる場合はそれをディスク上のテーブルに変換できます。このような場合は、変換をスキップし、ディスク上に内部一時テーブルを作成して開始することで、パフォーマンスが向上する可能性があります。`big_tables` 変数を使用して、内部一時テーブルのディスク記憶域を強制できます。

内部一時テーブルの作成の監視

内部一時テーブルがメモリまたはディスクに作成されると、サーバーは `Created_tmp_tables` 値を増分します。内部一時テーブルがディスク上に作成されると、サーバーは `Created_tmp_disk_tables` 値を増分します。ディスク上に作成される内部一時テーブルが多すぎる場合は、`tmp_table_size` および `max_heap_table_size` の設定を増やすことを検討してください。

注記

既知の制限のため、`Created_tmp_disk_tables` ではメモリマップファイルに作成されたディスク上の一時テーブルはカウントされません。デフォルトでは、`TempTable` ストレージエンジンオーバーフローメカニズムは、メモリマップされたファイルに内部一時テーブルを作成します。この動作は、`temptable_use_mmap` および `temptable_max_mmap` 変数によって制御されます。

`memory/temptable/physical_ram` および `memory/temptable/physical_disk` パフォーマンススキーマインストゥルメントを使用すると、メモリおよびディスクからの `TempTable` 領域割り当てをモニターできます。`memory/temptable/physical_ram` では、割り当てられた RAM の量がレポートされます。メモリマップされたファイルが `TempTable` オーバーフローメカニズムとして使用されている場合、`memory/temptable/physical_disk` はディスクから割り当てられた領域の量を報告します。`physical_disk` インストゥルメントが 0 以外の値を報告し、メモリマップされたファイルが `TempTable` オーバーフローメカニズムとして使用される場合、`temptable_max_ram` しきい値にはある時点で到達しました。データは、`memory_summary_global_by_event_name` などのパフォーマンススキーマメモリーサマリーテーブルでクエリーできます。[セクション27.12.18.10「メモリーサマリーテーブル」](#)を参照してください。

8.4.5 データベースおよびテーブルの数に対する制限

MySQL にはデータベース数の制限はありません。ベースとなるファイルシステムによっては、ディレクトリ数に制限がある場合があります。

MySQL にはテーブル数の制限はありません。ベースとなるファイルシステムによっては、テーブルを表すファイル数に制限がある場合があります。個々のストレージエンジンには、エンジン固有の制約が課される場合があります。`InnoDB` では、最大 40 億個のテーブルを使用できます。

8.4.6 テーブルサイズの制限

MySQL データベースの事実上の最大テーブルサイズは、通常、MySQL の内部制限ではなくオペレーティングシステムのファイルサイズに関する制約によって判断します。オペレーティングシステムのファイルサイズに関する最新情報は、オペレーティングシステム固有のドキュメントを参照してください。

Windows ユーザーの場合、FAT および VFAT (FAT32) は、MySQL で本番使用に適しているとは見なされません。代わりに NTFS を使用してください。

全テーブルエラーが発生した場合は、いくつかの理由が考えられます:

- ディスクがいっぱいになっている可能性がある。
- InnoDB テーブルを使用しており、InnoDB テーブルスペースファイルの領域が不足しています。最大テーブルスペースサイズは、テーブルの最大サイズでもあります。テーブルスペースのサイズ制限については、[セクション 15.22 「InnoDB の制限」](#) を参照してください。
一般に、サイズが 1TB を超えるテーブルでは、テーブルを複数のテーブルスペースファイルにパーティション化することをお勧めします。
- オペレーティングシステムのファイルサイズ制限に達しました。たとえば、最大 2GB のサイズのファイルのみをサポートするオペレーティングシステムで MyISAM テーブルを使用しており、データファイルまたはインデックスファイルに対してこの制限に達しているとします。
- MyISAM テーブルを使用していて、テーブルに必要な領域が内部ポインタサイズによって許可されているサイズを超えている。MyISAM では、データファイルとインデックスファイルのサイズはデフォルトで最大 256T バイトですが、この制限は、65,536T バイト ($256^7 - 1$ バイト) の最大許容サイズまで変更できます。

デフォルトの制限より大きな MyISAM テーブルが必要であり、オペレーティングシステムが大きなファイルをサポートしている場合は、`CREATE TABLE` ステートメントは `AVG_ROW_LENGTH` オプションと `MAX_ROWS` オプションをサポートします。[セクション 13.1.20 「CREATE TABLE ステートメント」](#) を参照してください。サーバーはこれらのオプションを使用して、許可するテーブルのサイズを決定します。

ポインタサイズが既存のテーブルに対して小さすぎる場合は、`ALTER TABLE` でオプションを変更して、テーブルの最大許容サイズを増やすことができます。[セクション 13.1.9 「ALTER TABLE ステートメント」](#) を参照してください。

```
ALTER TABLE tbl_name MAX_ROWS=1000000000 AVG_ROW_LENGTH=nnn;
```

`AVG_ROW_LENGTH` は、`BLOB` カラムまたは `TEXT` カラムを含むテーブルに対してのみ指定する必要があります。この場合、MySQL では、行数にのみ基づいて必要な領域を最適化することはできません。

MyISAM テーブルのデフォルトのサイズ制限を変更するには、内部行ポインタに使用されるバイト数を設定する `myisam_data_pointer_size` を設定します。`MAX_ROWS` オプションを指定しない場合、新しいテーブルのポインタサイズを設定するためにこの値が使用されます。`myisam_data_pointer_size` の値は 2 から 7 で指定できます。4 の値は 4G バイトまでのテーブルを許可し、6 の値は 256T バイトまでのテーブルを許可します。

次のステートメントを使用すると、データファイルおよびインデックスファイルの最大サイズを確認できます。

```
SHOW TABLE STATUS FROM db_name LIKE 'tbl_name';
```

`myisamchk -dv /path/to/table-index-file` も使用できます。[セクション 13.7.7 「SHOW ステートメント」](#) または [セクション 4.6.4 「myisamchk — MyISAM テーブルメンテナンスユーティリティ」](#) を参照してください。

MyISAM テーブルのファイルサイズ制限に対処するその他の方法は次のとおりです。

- 大きなテーブルが読み取り専用である場合、`myisampack` を使用してこのテーブルを圧縮できます。`myisampack` は通常、少なくとも 50% 圧縮するので、実質上、さらに大きなテーブルを保有できます。`myisampack` で複数のテーブルを単一のテーブルにマージすることもできます。[セクション 4.6.6 「myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成」](#) を参照してください。
- MySQL には、単一の `MERGE` テーブルと同一の構造を持つ MyISAM テーブルの集まりを処理できるようにする `MERGE` ライブラリが含まれます。[セクション 16.7 「MERGE ストレージエンジン」](#) を参照してください。
- `MEMORY (HEAP)` ストレージエンジンを使用している場合は、`max_heap_table_size` システム変数の値を増やす必要があります。[セクション 5.1.8 「サーバーシステム変数」](#) を参照してください。

8.4.7 テーブルカラム数と行サイズの制限

このセクションでは、テーブルのカラム数および個々の行のサイズの制限について説明します。

- [カラム数制限](#)
- [行サイズ制限](#)

カラム数制限

MySQL にはテーブル当たり 4096 カラムの強い制限がありますが、特定のテーブルの有効な最大値が少なくなる可能性があります。カラムの正確な制限は、いくつかの要因によって異なります：

- すべてのカラムの合計長がこのサイズを超えることはできないため、テーブルの最大行サイズによってカラムの数（および場合によってはサイズ）が制限されます。 [行サイズ制限](#) を参照してください。
- 個々のカラムの記憶域要件によって、指定された最大行サイズ内に収まるカラム数が制限されます。一部のデータ型の記憶域要件は、記憶域エンジン、記憶域形式、文字セットなどの要因によって異なります。 [セクション 11.7 「データ型のストレージ要件」](#) を参照してください。
- ストレージエンジンは、テーブルカラム数を制限する追加の制限を課す場合があります。たとえば、[InnoDB](#) には、テーブル当たり 1017 カラムの制限があります。 [セクション 15.22 「InnoDB の制限」](#) を参照してください。その他のストレージエンジンについては、 [第 16 章 「代替ストレージエンジン」](#) を参照してください。
- 関数キー部分 ([セクション 13.1.15 「CREATE INDEX ステートメント」](#) を参照) は、非表示の仮想生成ストアカラムとして実装されるため、テーブルインデックス内の各関数キー部分は、テーブルの合計カラム制限に対してカウントされます。

行サイズ制限

特定のテーブルの最大行サイズは、いくつかの要因によって決定されます：

- ストレージエンジンがより大きな行をサポートできる場合でも、MySQL テーブルの内部表現の最大行サイズは 65,535 バイトです。 [BLOB](#) および [TEXT](#) のカラムは、行サイズ制限に 9 から 12 バイトのみ寄与します。これは、その内容が行の他の部分とは別に格納されるためです。
- データベースページ内にローカルに格納されたデータに適用される [InnoDB](#) テーブルの最大行サイズは、4KB、8KB、16KB および 32KB の `innodb_page_size` 設定のページの半分未満です。たとえば、デフォルトの 16KB の [InnoDB](#) ページサイズでは、最大行サイズは 8KB 未満です。64KB ページの場合、最大行サイズは 16KB 未満です。 [セクション 15.22 「InnoDB の制限」](#) を参照してください。

[variable-length columns](#) を含む行が [InnoDB](#) の最大行サイズを超える場合、[InnoDB](#) は、行が [InnoDB](#) の行サイズ制限内に収まるまで、外部オフページストレージの可変長カラムを選択します。ページ外に格納される可変長カラムに対してローカルに格納されるデータ量は、行形式によって異なります。詳細は、 [セクション 15.10 「InnoDB の行フォーマット」](#) を参照してください。

- 記憶域形式が異なると、ページヘッダーおよびトレーラデータが異なるため、行に使用可能な記憶域の量に影響します。
 - [InnoDB](#) の行フォーマットの詳細は、 [セクション 15.10 「InnoDB の行フォーマット」](#) を参照してください。
 - [MyISAM](#) 記憶域形式の詳細は、 [セクション 16.2.3 「MyISAM テーブルのストレージフォーマット」](#) を参照してください。

行サイズ制限の例

- 次の [InnoDB](#) および [MyISAM](#) の例では、65,535 バイトの MySQL 最大行サイズ制限を示します。この制限は、ストレージエンジンがより大きな行をサポートできる場合でも、ストレージエンジンに関係なく適用されます。

```
mysql> CREATE TABLE t (a VARCHAR(10000), b VARCHAR(10000),  
c VARCHAR(10000), d VARCHAR(10000), e VARCHAR(10000),  
f VARCHAR(10000), g VARCHAR(6000)) ENGINE=InnoDB CHARACTER SET latin1;
```

```
ERROR 1118 (42000): Row size too large. The maximum row size for the used
table type, not counting BLOBs, is 65535. This includes storage overhead,
check the manual. You have to change some columns to TEXT or BLOBs
```

```
mysql> CREATE TABLE t (a VARCHAR(10000), b VARCHAR(10000),
  c VARCHAR(10000), d VARCHAR(10000), e VARCHAR(10000),
  f VARCHAR(10000), g VARCHAR(6000)) ENGINE=MyISAM CHARACTER SET latin1;
```

```
ERROR 1118 (42000): Row size too large. The maximum row size for the used
table type, not counting BLOBs, is 65535. This includes storage overhead,
check the manual. You have to change some columns to TEXT or BLOBs
```

次の **MyISAM** の例では、カラムを **TEXT** に変更すると、65,535-byte の行サイズ制限が回避され、**BLOB** および **TEXT** のカラムは行サイズに 9 から 12 バイトしか寄与しないため、操作を成功させることができます。

```
mysql> CREATE TABLE t (a VARCHAR(10000), b VARCHAR(10000),
  c VARCHAR(10000), d VARCHAR(10000), e VARCHAR(10000),
  f VARCHAR(10000), g TEXT(6000)) ENGINE=MyISAM CHARACTER SET latin1;
```

```
Query OK, 0 rows affected (0.02 sec)
```

カラムを **TEXT** に変更すると MySQL 65,535-byte の行サイズ制限が回避され、可変長カラムの **InnoDB** オフページストレージによって **InnoDB** の行サイズ制限が回避されるため、**InnoDB** テーブルに対する操作は成功します。

```
mysql> CREATE TABLE t (a VARCHAR(10000), b VARCHAR(10000),
  c VARCHAR(10000), d VARCHAR(10000), e VARCHAR(10000),
  f VARCHAR(10000), g TEXT(6000)) ENGINE=InnoDB CHARACTER SET latin1;
```

```
Query OK, 0 rows affected (0.02 sec)
```

- 可変長カラムの記憶域には、行サイズにカウントされる長さバイトが含まれます。たとえば、**VARCHAR(255) CHARACTER SET utf8mb3** カラムには値の長さを格納するために 2 バイトかかるため、各値には最大 767 バイトを使用できます。

カラムに 32,765 + 2 バイトおよび 32,766 + 2 バイトが必要で、これは最大行サイズ 65,535 バイト内にあるため、テーブル **t1** を作成するステートメントは成功します:

```
mysql> CREATE TABLE t1
  (c1 VARCHAR(32765) NOT NULL, c2 VARCHAR(32766) NOT NULL)
  ENGINE = InnoDB CHARACTER SET latin1;
```

```
Query OK, 0 rows affected (0.02 sec)
```

カラムの長さが 65,535 バイトの最大長内にあるにもかかわらず、行サイズが 65,535 バイトを超えるため、テーブル **t2** を作成するステートメントは失敗します:

```
mysql> CREATE TABLE t2
  (c1 VARCHAR(65535) NOT NULL)
  ENGINE = InnoDB CHARACTER SET latin1;
```

```
ERROR 1118 (42000): Row size too large. The maximum row size for the used
table type, not counting BLOBs, is 65535. This includes storage overhead,
check the manual. You have to change some columns to TEXT or BLOBs
```

カラム長を 65,533 以下に減らすと、ステートメントは成功します。

```
mysql> CREATE TABLE t2
  (c1 VARCHAR(65533) NOT NULL)
  ENGINE = InnoDB CHARACTER SET latin1;
```

```
Query OK, 0 rows affected (0.01 sec)
```

- MyISAM** テーブルの場合、**NULL** カラムは、値が **NULL** であるかどうかを記録するための追加領域を行内に必要とします。各 **NULL** カラムは 1 ビット余分に占め、もっとも近いバイトまで丸められます。

MyISAM には可変長のカラム長バイトに必要な領域に加えて **NULL** カラムの領域が必要であり、行サイズが 65,535 バイトを超えるため、テーブル **t3** を作成するステートメントは失敗します:

```
mysql> CREATE TABLE t3
  (c1 VARCHAR(32765) NULL, c2 VARCHAR(32766) NULL)
  ENGINE = MyISAM CHARACTER SET latin1;
```

```
ERROR 1118 (42000): Row size too large. The maximum row size for the used
table type, not counting BLOBs, is 65535. This includes storage overhead,
check the manual. You have to change some columns to TEXT or BLOBs
```

InnoDB NULL カラムの記憶域の詳細は、[セクション15.10「InnoDB の行フォーマット」](#) を参照してください。

- InnoDB では、行サイズ (データベースページ内にローカルに格納されるデータの場合) は、4KB、8KB、16KB および 32KB の `innodb_page_size` 設定ではデータベースページの半分未満に制限され、64KB ページでは 16KB 未満に制限されます。

定義されたカラムが 16KB の InnoDB ページの行サイズ制限を超えているため、テーブル `t4` を作成するステートメントは失敗します。

```
mysql> CREATE TABLE t4 (  
  c1 CHAR(255),c2 CHAR(255),c3 CHAR(255),  
  c4 CHAR(255),c5 CHAR(255),c6 CHAR(255),  
  c7 CHAR(255),c8 CHAR(255),c9 CHAR(255),  
  c10 CHAR(255),c11 CHAR(255),c12 CHAR(255),  
  c13 CHAR(255),c14 CHAR(255),c15 CHAR(255),  
  c16 CHAR(255),c17 CHAR(255),c18 CHAR(255),  
  c19 CHAR(255),c20 CHAR(255),c21 CHAR(255),  
  c22 CHAR(255),c23 CHAR(255),c24 CHAR(255),  
  c25 CHAR(255),c26 CHAR(255),c27 CHAR(255),  
  c28 CHAR(255),c29 CHAR(255),c30 CHAR(255),  
  c31 CHAR(255),c32 CHAR(255),c33 CHAR(255)  
 ) ENGINE=InnoDB ROW_FORMAT=DYNAMIC DEFAULT CHARSET latin1;  
ERROR 1118 (42000): Row size too large (> 8126). Changing some columns to TEXT or BLOB may help.  
In current row format, BLOB prefix of 0 bytes is stored inline.
```

8.5 InnoDB テーブルの最適化

InnoDB は、MySQL のお客様が一般に、信頼性と並列性が重要である本番環境のデータベースで使用するストレージエンジンです。InnoDB は、MySQL のデフォルトのストレージエンジンです。このセクションでは、InnoDB テーブルに対するデータベース操作を最適化する方法について説明します。

8.5.1 InnoDB テーブルのストレージレイアウトの最適化

- データが安定したサイズに達するか、拡大しているテーブルが数十または数百メガバイト単位で増大した場合、`OPTIMIZE TABLE` ステートメントを使用して、テーブルを再編成し、無駄なスペースを圧縮することを考慮します。再編成されたテーブルでは、フルテーブルスキャンを実行するために必要なディスク I/O が減ります。これは、インデックスの使用の改善やアプリケーションコードのチューニングなどのほかの技法が現実的でない場合に、パフォーマンスを向上できる直接的な技法です。

`OPTIMIZE TABLE` はテーブルのデータ部分をコピーし、インデックスを再構築します。インデックス内へのデータのバツクの改善とテーブルスペース内およびディスク上の断片化の削減からメリットが得られます。このメリットは各テーブル内のデータによって異なります。利点が大きいものとそうでないものがあること、またはテーブルの次の最適化まで、時間の経過とともに利点が減っていくことに気付く場合があります。テーブルが大きい場合、または再構築されるインデックスがバッファプールに収まらない場合、この操作は遅くなる可能性があります。大量のデータをテーブルに追加したあとの最初の実行では、多くの場合にその後の実行よりかなり遅くなります。

- InnoDB では、長い `PRIMARY KEY` (長い値を持つ単一カラムまたは長い複合値を形成する複数のカラムのいずれか) があると、大量のディスク領域を無駄にします。行の主キー値は、同じ行を指すすべてのセカンダリインデックスレコードに複製されます。(セクション15.6.2.1「クラスティンデックスとセカンダリインデックス」を参照してください。) 主キーが長い場合、`AUTO_INCREMENT` カラムを主キーとして作成するか、カラム全体ではなく、長い `VARCHAR` カラムのプリフィクスをインデックス設定します。
- 可変長の文字列を格納するために、または多くの `NULL` 値を持つカラムに対して、`CHAR` の代わりに `VARCHAR` データ型を使用します。`CHAR(N)` カラムは、文字列が短い、その値が `NULL` だとしても、データを格納するために常に `N` 文字を必要とします。テーブルが小さいほどバッファプールに収まりやすく、ディスク I/O が減ります。

`COMPACT` 行フォーマット (デフォルトの InnoDB フォーマット) および可変長文字セット (`utf8` や `sjis` など) を使用する場合、`CHAR(N)` カラムは可変容量の領域を占有しますが、`N` バイト以上のままです。

- 大きい、繰り返しの多い大量のテキストや数値データを格納するテーブルでは、`COMPRESSED` 行フォーマットを使用することを考慮します。データをバッファプールに入れたい、フルテーブルスキャンを実行したりするために必要なディスク I/O が減ります。永続的な決断を下す前に、`COMPRESSED` と `COMPACT` の行フォーマットを使用して達成できる圧縮の量を測定してください。

8.5.2 InnoDB トランザクション管理の最適化

InnoDB トランザクション処理を最適化するには、トランザクション機能のパフォーマンスオーバーヘッドとサーバーのワークロードの理想的なバランスを見つけます。たとえば、アプリケーションで、秒あたり数千回コミットする場合にパフォーマンスの問題が発生し、2、3時間に1回だけコミットする場合に別のパフォーマンスの問題が発生することがあります。

- デフォルトの MySQL 設定 `AUTOCOMMIT=1` は、ビジネスデータベースサーバーにパフォーマンスの制限を課すことがあります。実用的な場合は、`SET AUTOCOMMIT=0` または `START TRANSACTION` ステートメントを発行し、すべての変更を行った後に `COMMIT` ステートメントを発行して、関連する複数のデータ変更操作を単一のトランザクションにラップします。

InnoDB は、トランザクションによってデータベースが変更された場合、そのトランザクションのコミットのたびにディスクにログをフラッシュする必要があります。変更のたびにあとでコミットされる場合(デフォルトの自動コミット設定のように)、ストレージデバイスの I/O スループットによって、秒あたりに可能な操作数が制限されます。

- または、単一の `SELECT` ステートメントのみから構成されるトランザクションの場合、`AUTOCOMMIT` をオンにすると、InnoDB が読み取り専用トランザクションを認識し、それらを最適化するのに役立ちます。要件については、[セクション8.5.3「InnoDB の読み取り専用トランザクションの最適化」](#)を参照してください。
- 大量の行の挿入、更新、または削除後のロールバックの実行は避けます。大きなトランザクションがサーバーのパフォーマンスを低下させている場合、ロールバックすると問題が悪化し、元のデータ変更操作の実行に数回かかる可能性があります。ロールバックはサーバーの起動時に再度開始されるため、データベースプロセスを強制終了しても役立ちません。

この問題が発生する可能性を最小限に抑えるには:

- すべてのデータ変更がディスクに即座に書き込まれるのではなくキャッシュされるように、`buffer pool` のサイズを増やします。
- 挿入に加えて更新および削除操作がバッファされるように `innodb_change_buffering=all` を設定します。
- ビッグデータ変更操作中に `COMMIT` ステートメントを定期的に発行することを検討してください。これにより、単一の削除または更新が少数の行で動作する複数のステートメントに分割される可能性があります。

ロールバックの暴走が発生した場合にそれを解消するには、ロールバックが CPU に依存して高速に実行するように、バッファプールを増加するか、[セクション15.18.2「InnoDB のリカバリ」](#)に説明するように、サーバーを強制終了し、`innodb_force_recovery=3` で再起動します。

この問題は、デフォルト設定の `innodb_change_buffering=all` ではあまり発生しないことが予想されます。これにより、更新および削除操作をメモリーにキャッシュできるため、最初の場所での実行が高速になり、必要に応じてロールバックも高速になります。多くの挿入、更新、または削除を伴う長時間実行トランザクションを処理するサーバーでこのパラメータ設定を使うようにしてください。

- 予期しない終了が発生した場合に最新のコミット済トランザクションの一部が失われる可能性がある場合は、`innodb_flush_log_at_trx_commit` パラメータを 0 に設定できます。フラッシュが保証されていなくても、InnoDB はとにかく 1 秒に 1 回ログをフラッシュしようとしています。
- 行が変更されるか削除される場合、行と関連付けられた **Undo ログ** はただちに、またはトランザクションのコミットの直後でも、物理的に削除されません。以前または同時に開始したトランザクションが終了するまで古いデータは保持されるため、それらのトランザクションは変更または削除された行の以前の状態にアクセスできます。そのため、長時間実行トランザクションは、InnoDB が別のトランザクションによって変更されたデータをパージすることを妨げることがあります。
- 長時間実行トランザクション内で行が変更されるか、削除された場合、`READ COMMITTED` および `REPEATABLE READ` 分離レベルを使用するほかのトランザクションは、古いデータを再構築するために、それらの同じ行を読み取る場合、多くの作業を実行する必要があります。
- 長時間実行トランザクションでテーブルが変更された場合、ほかのトランザクションからのそのテーブルに対するクエリーは、**カバリングインデックス**技法を利用しません。通常、セカンダリインデックスからすべての結果カラムを取得できるクエリーは、代わりにテーブルデータから該当する値をルックアップします。

セカンダリインデックスページに、新しすぎる `PAGE_MAX_TRX_ID` があることが検出された場合、またはセカンダリインデックス内のレコードに削除がマークされている場合、InnoDB はクラスタ化されたインデックスを使用してレコードをルックアップする必要がある可能性があります。

8.5.3 InnoDB の読み取り専用トランザクションの最適化

InnoDB では、読み取り専用として認識されているトランザクションの `transaction ID` (`TRX_ID` フィールド) の設定に関連するオーバーヘッドを回避できます。トランザクション ID は、書き込み操作または **ロック読み取り** (`SELECT ... FOR UPDATE` など) を実行する可能性のある **トランザクション** にのみ必要です。不要なトランザクション ID を排除すると、クエリーまたはデータ変更ステートメントが `read view` を構成するたびに参照される内部データ構造のサイズが削減されます。

InnoDB は、次の場合に読み取り専用トランザクションを検出します:

- トランザクションが `START TRANSACTION READ ONLY` ステートメントで開始された場合。この場合は、データベース (InnoDB、MyISAM、またはその他のタイプのテーブル) に対して変更を行おうとするとエラーが発生し、そのトランザクションは読み取り専用状態のままになります。

```
ERROR 1792 (25006): Cannot execute statement in a READ ONLY transaction.
```

ただし、読み取り専用トランザクションでのセッション固有の一時テーブルの変更や、それらのテーブルに対するロックエラーの発行は、その変更やロックがほかのどのトランザクションにも表示されないため引き続き可能です。

- `autocommit` 設定がオンになっているため、トランザクションが 1 つのステートメントであることが保証され、そのトランザクションを構成している 1 つのステートメントが「非ロック」の `SELECT` ステートメントである場合。つまり、`FOR UPDATE` または `LOCK IN SHARED MODE` 句を使用しない `SELECT` です。
- トランザクションは `READ ONLY` オプションなしで開始されますが、行を明示的にロックする更新またはステートメントはまだ実行されていません。更新または明示的ロックが必要になるまで、トランザクションは読み取り専用モードのままです。

したがって、レポートジェネレータなどの読み取り集中型アプリケーションの場合は、`START TRANSACTION READ ONLY` および `COMMIT` 内でグループ化するか、`SELECT` ステートメントを実行する前に `autocommit` 設定をオンにするか、単にクエリーと混在するデータ変更ステートメントを回避することで、一連の InnoDB クエリーをチューニングできます。

`START TRANSACTION` および `autocommit` については、[セクション 13.3.1 「START TRANSACTION、COMMIT および ROLLBACK ステートメント」](#) を参照してください。

注記

自動コミット、非ロック、および読み取り専用 (AC-NL-RO) として承認されたトランザクションは、InnoDB の特定の内部データ構造から除外されるため、`SHOW ENGINE INNODB STATUS` の出力には表示されません。

8.5.4 InnoDB redo ロギングの最適化

redo ロギングを最適化するために、次のガイドラインを考慮してください:

- redo ログファイルを `buffer pool` と同じ大きさにします。InnoDB が redo ログファイルをいっぱいにした場合、バッファプールの変更された内容を `checkpoint` のディスクに書き込む必要があります。redo ログファイルが小さいと、不要なディスク書き込みが多数発生します。以前は大きな redo ログファイルが原因で長いリカバリ時間が発生していましたが、リカバリが大幅に高速になり、大きな redo ログファイルを確実に使用できるようになりました。

redo ログファイルのサイズと数は、`innodb_log_file_size` および `innodb_log_files_in_group` の構成オプションを使用して構成します。既存の redo ログファイル構成の変更の詳細は、[redo ログファイルの数またはサイズの変更](#) を参照してください。

- `log buffer` のサイズを増やすことを検討してください。ログバッファを大きくすると、トランザクションが **コミット** する前にディスクにログを書き込まなくても、大規模な **トランザクション** を実行できます。したがって、多

数の行を更新、挿入、または削除するトランザクションの場合、ログバッファを大きくすると、ディスク I/O を節約できます。ログバッファサイズは、MySQL 8.0 で動的に構成できる `innodb_log_buffer_size` 構成オプションを使用して構成します。

- 「read-on-write」を回避するには、`innodb_log_write_ahead_size` 構成オプションを構成します。このオプションは、redo ログの先行書込みブロックサイズを定義します。オペレーティングシステムまたはファイルシステムのキャッシュブロックサイズと一致するように `innodb_log_write_ahead_size` を設定します。読取り/書込みは、redo ログの先行書込みブロックサイズとオペレーティングシステムまたはファイルシステムのキャッシュブロックサイズが一致しないために、redo ログブロックがオペレーティングシステムまたはファイルシステムに完全にキャッシュされない場合に発生します。

`innodb_log_write_ahead_size` の有効な値は、InnoDB ログファイルのブロックサイズ (2^n) の倍数です。最小値は、InnoDB ログファイルのブロックサイズ (512) です。最小値が指定されている場合、ライトアヘッドは発生しません。最大値は `innodb_page_size` 値と同じです。`innodb_log_write_ahead_size` に `innodb_page_size` 値より大きい値を指定すると、`innodb_log_write_ahead_size` 設定は `innodb_page_size` 値に切り捨てられます。

オペレーティングシステムまたはファイルシステムのキャッシュブロックサイズに関連して `innodb_log_write_ahead_size` 値の設定が小さすぎると、読取り/書込みになります。値を高く設定しすぎると、一度に複数のブロックが書き込まれるため、ログファイル書込みの `fsync` パフォーマンスにわずかな影響を与える可能性があります。

- MySQL 8.0.11 では、ログバッファからシステムバッファに redo ログレコードを書き込み、システムバッファを redo ログファイルにフラッシュするための専用のログライタースレッドが導入されました。以前は、個々のユーザースレッドがこれらのタスクを担当していました。MySQL 8.0.22 では、`innodb_log_writer_threads` 変数を使用してログライタースレッドを有効または無効にできます。専用ログライタースレッドを使用すると、同時実行性の高いシステムのパフォーマンスを向上させることができますが、同時実行性の低いシステムでは、専用ログライタースレッドを無効にすると、パフォーマンスが向上します。
- フラッシュされた redo を待機するユーザースレッドによるスピン遅延の使用を最適化します。スピン遅延は、待機時間の短縮に役立ちます。同時実行性の低い期間では、待機時間を短縮すると優先度が低くなり、これらの期間中のスピン遅延の使用を回避するとエネルギー消費が削減される可能性があります。同時実行性が高い期間は、スピン遅延時の処理能力の消費を回避して、他の作業に使用できるようにすることが必要な場合があります。次のシステム変数では、スピン遅延を使用するための境界を定義する最高水位標値と最低水位標値を設定できます。
- `innodb_log_wait_for_flush_spin_hwm`: フラッシュされた redo の待機中にユーザースレッドがスピンしなくなる最大平均ログフラッシュ時間を定義します。デフォルト値は 400 マイクロ秒です。
- `innodb_log_spin_cpu_abs_lwm`: フラッシュされた redo の待機中にユーザースレッドがスピンしなくなる CPU 使用率の最小量を定義します。この値は、CPU コア使用率の合計として表されます。たとえば、80 のデフォルト値は、単一の CPU コアの 80% です。マルチコアプロセッサを搭載したシステムでは、150 の値は、1 つの CPU コアの 100% 使用率と 2 つ目の CPU コアの 50% 使用率を表します。
- `innodb_log_spin_cpu_pct_hwm`: フラッシュされた redo の待機中にユーザースレッドがスピンしなくなる CPU 使用率の最大量を定義します。この値は、すべての CPU コアの合計処理能力の割合として表されます。デフォルト値は 50% です。たとえば、2 つの CPU コアの 100% 使用率は、4 つの CPU コアを持つサーバーでの CPU 処理能力の合計の 50% です。

`innodb_log_spin_cpu_pct_hwm` 構成オプションは、プロセッサアフィニティを考慮します。たとえば、サーバーに 48 個のコアがあり、`mysqld` プロセスが 4 個の CPU コアにのみ固定されている場合、他の 44 個の CPU コアは無視されます。

8.5.5 InnoDB テーブルの一括データロード

これらのパフォーマンスのヒントは、[セクション8.2.5.1「INSERT ステートメントの最適化」](#)の高速挿入の一般的なガイドラインを補足するものです。

- InnoDB にデータをインポートする場合、自動コミットモードでは挿入のたびに、ディスクへのログのフラッシュを実行するため、それをオフにします。インポート操作時に自動コミットを無効にするには、それを、`SET autocommit` ステートメントと `COMMIT` ステートメントで囲みます。

```
SET autocommit=0;  
... SQL import statements ...  
COMMIT;
```


`mysqldump` オプション `--opt` は、それらを `SET autocommit` ステートメントと `COMMIT` ステートメントで囲まなくても、InnoDB テーブルに高速にインポートするダンプファイルを作成します。

- 副キーに **UNIQUE** 制約がある場合、インポートセッション中に一意性チェックを一時的にオフにすることで、テーブルインポートを高速化できます。

```
SET unique_checks=0;
... SQL import statements ...
SET unique_checks=1;
```

大きなテーブルの場合、InnoDB はその変更バッファを使用してセカンダリインデックスレコードをバッチで書き込むことができるため、これによりディスク I/O が大量に節約されます。データに重複キーが含まれていないことを確認してください。

- テーブルに **FOREIGN KEY** 制約がある場合、インポートセッションの間の外部キーチェックをオフにすることで、テーブルインポートを高速化できます。

```
SET foreign_key_checks=0;
... SQL import statements ...
SET foreign_key_checks=1;
```

大きいテーブルの場合、これにより、大量のディスク I/O を節約できます。

- 多くの行を挿入する必要がある場合、複数行 `INSERT` 構文を使用して、クライアントとサーバー間の通信オーバーヘッドを軽減します。

```
INSERT INTO yourtable VALUES (1,2), (5,5), ...;
```

このヒントは、InnoDB テーブルだけではなく、任意のテーブルへの挿入に有効です。

- 自動増分カラムを含むテーブルに一括挿入を実行する場合は、`innodb_autoinc_lock_mode` を 1 (連続) ではなく 2 (インターリーブ) に設定します。詳細は、[セクション15.6.1.6 「InnoDB での AUTO_INCREMENT 処理」](#) を参照してください。
- 一括挿入を実行する場合は、**PRIMARY KEY** の順序で行を挿入する方が高速です。InnoDB テーブルでは **clustered index** が使用されるため、**PRIMARY KEY** の順序で比較的高速にデータを使用できます。**PRIMARY KEY** 順序での一括挿入の実行は、バッファプール内に完全に収まらないテーブルで特に重要です。
- InnoDB **FULLTEXT** インデックスにデータをロードする場合の最高のパフォーマンスのため、次の一連のステップに従います。
 1. テーブル作成時に、**FTS_DOC_ID_INDEX** という一意のインデックスで、型 **BIGINT UNSIGNED NOT NULL** のカラム **FTS_DOC_ID** を定義します。例:

```
CREATE TABLE t1 (
  FTS_DOC_ID BIGINT unsigned NOT NULL AUTO_INCREMENT,
  title varchar(255) NOT NULL DEFAULT "",
  text mediumtext NOT NULL,
  PRIMARY KEY (FTS_DOC_ID)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
CREATE UNIQUE INDEX FTS_DOC_ID_INDEX on t1(FTS_DOC_ID);
```

2. テーブルにデータをロードします。
3. データがロードされたら、**FULLTEXT** インデックスを作成します。

注記

テーブル作成時に **FTS_DOC_ID** カラムを追加する場合、**FTS_DOC_ID** は各 `INSERT` または `UPDATE` によって単調に増分される必要があるため、**FULLTEXT** インデックス設定されたカラムが更新されたときに、**FTS_DOC_ID** カラムが更新されることを確認します。テーブルの作成時に **FTS_DOC_ID** を追加せずに、InnoDB で DOC ID を管理する場合、InnoDB は、次の `CREATE FULLTEXT INDEX` コールで **FTS_DOC_ID** を非表示カラムとして追加します。ただし、このアプローチでは、パフォーマンスに影響を与える可能性のあるテーブルの再構築が必要です。

8.5.6 InnoDB クエリーの最適化

InnoDB テーブルのクエリーをチューニングするには、各テーブルに適切なインデックスのセットを作成します。詳細は、[セクション8.3.1「MySQL のインデックスの使用の仕組み」](#)を参照してください。InnoDB インデックスに関する次のガイドラインに従います。

- 各 InnoDB テーブルには**主キー**がある(それをリクエストするかしないかに関係なく)ため、もっとも重要で緊急を要するクエリーで使用されるカラムとして、テーブルごとに主キーカラムのセットを指定します。
- 主キーのカラム値は各セカンダリインデックスに複製されるため、あまり多く、長すぎるカラムを指定しないでください。インデックスに不要なデータが含まれていると、このデータを読み取る I/O とそれをキャッシュするメモリーによって、サーバーのパフォーマンスとスケーラビリティが低下します。
- 各クエリーで使用できるインデックスは 1 つだけであるため、カラムごとに個別の**セカンダリインデックス**を作成しないでください。ほんの少数の異なる値を持ち、めったにテストされないカラムへのインデックスは、どのクエリーにも役立たない可能性があります。同じテーブルに対して多くのクエリーがあり、カラムのさまざまな組み合わせをテストする場合、多数の単一カラムインデックスよりも、少数の**連結されたインデックス**を作成してみてください。インデックスに結果セットに必要なすべてのカラムが含まれている(**カバリングインデックス**と呼ばれる)場合、クエリーはテーブルデータをまったく読み取らなくても済む可能性があります。
- インデックス設定されたカラムに **NULL** 値が含まれることがない場合は、テーブルの作成時に、それを **NOT NULL** として宣言します。オプティマイザは、各カラムに **NULL** 値が含まれているかどうかがわかっている場合、クエリーに最も効果的なインデックスを判断できます。
- [セクション8.5.3「InnoDB の読み取り専用トランザクションの最適化」](#)の手法を使用して、InnoDB テーブルの単一クエリートランザクションを最適化できます。

8.5.7 InnoDB DDL 操作の最適化

- テーブルおよびインデックス(**CREATE**、**ALTER** および **DROP** ステートメント)に対する多くの DDL 操作は、オンラインで実行できます。詳細は、[セクション15.12「InnoDB とオンライン DDL」](#)を参照してください。
- セカンダリインデックスを追加するためのオンライン DDL サポートとは、通常、セカンダリインデックスのないテーブルを作成してからデータのロード後にセカンダリインデックスを追加することで、テーブルおよび関連するインデックスの作成およびロードのプロセスを高速化できることを意味します。
- テーブルを空にするには **DELETE FROM tbl_name** ではなく、**TRUNCATE TABLE** を使用します。外部キー制約により、**TRUNCATE** ステートメントを通常の **DELETE** ステートメントのように動作させることができます。その場合、**DROP TABLE** や **CREATE TABLE** のようなコマンドのシーケンスがもっとも速くなる可能性があります。
- 主キーは、各 InnoDB テーブルのストレージレイアウトに不可欠であり、主キーの定義の変更には、テーブル全体の再編成が必要であるため、常に主キーを **CREATE TABLE** ステートメントの一部としてセットアップし、あとで主キーを **ALTER** または **DROP** する必要があるないように、事前に計画してください。

8.5.8 InnoDB ディスク I/O の最適化

SQL 操作のためのデータベース設計およびチューニング方法のベストプラクティスに従っているが、ディスク I/O アクティビティが多いためにデータベースの速度がまだ遅い場合は、これらのディスク I/O 最適化を検討してください。Unix **top** ツールまたは Windows タスクマネージャーに、ワークロードの CPU 使用率が 70% 未満であることが示されている場合、ワークロードはディスクに依存している可能性があります。

- バッファプールサイズの増加

テーブルデータが InnoDB バッファプールにキャッシュされると、ディスク I/O を必要とせずにクエリーによって繰り返しアクセスできます。バッファプールサイズは、**innodb_buffer_pool_size** オプションで指定します。このメモリー領域は、通常、**innodb_buffer_pool_size** を 50 から 75% のシステムメモリーに構成することをお勧めします。詳細は、[セクション8.12.3.1「MySQL のメモリーの使用方法」](#)を参照してください。

- フラッシュ方法の調整

GNU/Linux および Unix の一部のバージョンでは、Unix **fsync()** 呼び出し(これは InnoDB がデフォルトで使用します)および類似のメソッドによるファイルのディスクへのフラッシュが驚くほど低速です。データベースの書き込

みパフォーマンスが問題である場合、`innodb_flush_method` パラメータを `O_DSYNC` に設定してベンチマークを実行します。

- fsync しきい値の構成

デフォルトでは、InnoDB が新しいログファイルやテーブルスペースファイルなどの新しいデータファイルを作成すると、ファイルはディスクにフラッシュされる前にオペレーティングシステムキャッシュに完全に書き込まれるため、大量のディスク書き込みアクティビティが一度に発生する可能性があります。オペレーティングシステムキャッシュから定期的にデータを強制的に小さいフラッシュするには、`innodb_fsync_threshold` 変数を使用してしきい値をバイト単位で定義します。バイトしきい値に達すると、オペレーティングシステムキャッシュの内容がディスクにフラッシュされます。デフォルト値の 0 では、デフォルトの動作が強制されます。つまり、ファイルがキャッシュに完全に書き込まれた後のみ、データがディスクにフラッシュされます。

複数の MySQL インスタンスが同じストレージデバイスを使用している場合は、より小さい定期的なフラッシュを強制的に実行するためのしきい値を指定すると有益です。たとえば、新しい MySQL インスタンスとそれに関連付けられたデータファイルを作成すると、ディスク書き込みアクティビティが大きくなり、同じストレージデバイスを使用する他の MySQL インスタンスのパフォーマンスが低下する可能性があります。しきい値を構成すると、書き込みアクティビティでのこのようなサージの回避に役立ちます。

- Linux でネイティブ AIO を使用した noop または deadline I/O スケジューラの使用

InnoDB は、Linux で非同期 I/O サブシステム (ネイティブ AIO) を使用して、データファイルページの先読みおよび書き込みリクエストを実行します。この動作は、デフォルトで有効になっている `innodb_use_native_aio` 構成オプションによって制御されます。ネイティブ AIO では、I/O スケジューラのタイプが I/O のパフォーマンスに大きく影響します。通常、noop および deadline I/O スケジューラをお勧めします。ベンチマークを実行して、ワークロードおよび環境に最適な結果を提供する I/O スケジューラを決定します。詳細は、[セクション 15.8.6 「Linux での非同期 I/O の使用」](#) を参照してください。

- x86_64 アーキテクチャーに Solaris 10 上の直接 I/O を使用

Solaris 10 for x86_64 Architecture (AMD Opteron) で InnoDB ストレージエンジンを使用する場合は、InnoDB 関連ファイルに直接 I/O を使用して、InnoDB パフォーマンスの低下を回避します。InnoDB 関連ファイルを格納するために使用される UFS ファイルシステム全体にダイレクト I/O を使用するには、それを `forcedirectio` オプションでマウントします。`mount_ufs(1M)` を参照してください。(Solaris 10/x86_64 のデフォルトではこのオプションを使用しません)。ダイレクト I/O をファイルシステム全体ではなく、InnoDB ファイル操作にのみ適用するには、`innodb_flush_method = O_DIRECT` を設定します。この設定では、InnoDB はデータファイルへの I/O (ログファイルへの I/O ではなく) に `fcntl()` ではなく、`directio()` を呼び出します。

- Solaris 2.6 以上でのデータおよびログファイルの RAW 記憶域の使用

任意のリリースの Solaris 2.6 および任意のプラットフォーム (sparc/x86/x64/amd64) で、`innodb_buffer_pool_size` 値が大きい InnoDB ストレージエンジンを使用する場合は、前に説明した `forcedirectio` マウントオプションを使用して、raw デバイスまたは別の直接 I/O UFS ファイルシステム上の InnoDB データファイルおよびログファイルでベンチマークを実行します。(ログファイルのダイレクト I/O が必要な場合、`innodb_flush_method` を設定する代わりに、マウントオプションを使用する必要があります。) Veritas ファイルシステム VxFS のユーザーは、`convosync=direct` マウントオプションを使用してください。

ダイレクト I/O ファイルシステムに、MyISAM テーブルのファイルなど、ほかの MySQL データファイルを配置しないでください。実行ファイルやライブラリは、ダイレクト I/O ファイルシステムに配置しないでください。

- 追加のストレージデバイスの使用

RAID 構成の設定には、追加のストレージデバイスを使用できます。関連情報については、[セクション 8.12.1 「ディスク I/O の最適化」](#) を参照してください。

または、InnoDB テーブルスペースデータファイルおよびログファイルを別の物理ディスクに配置できます。詳細は、次のセクションを参照してください:

- [セクション 15.8.1 「InnoDB の起動構成」](#)
- [セクション 15.6.1.2 「外部でのテーブルの作成」](#)
- [一般的なテーブルスペースの作成](#)

- [セクション15.6.1.4 「InnoDB テーブルの移動またはコピー」](#)

- 非ローテーション記憶域の検討

通常、非ローテーション記憶域を使用すると、ランダムな I/O 操作の場合はパフォーマンスが向上し、順次 I/O 操作の場合はローテーション記憶域が向上します。ローテーションストレージデバイスと非ローテーションストレージデバイスにデータファイルとログファイルを分散する場合は、主に各ファイルで実行される I/O 操作のタイプを考慮してください。

ランダム I/O-oriented ファイルには、通常、[file-per-table](#) および [general tablespace](#) データファイル、[undo tablespace](#) ファイルおよび [temporary tablespace](#) ファイルが含まれます。順次 I/O-oriented ファイルには、[InnoDB system tablespace](#) ファイル (MySQL 8.0.20 および [change buffering](#) より前の [doublewrite buffering](#) のため)、MySQL 8.0.20 で導入された二重書き込みファイル、およびリンク [binary log](#) ファイルや [redo log](#) ファイルなどのログファイルが含まれます。

非ローテーション記憶域を使用する場合は、次の構成オプションの設定を確認します:

- [innodb_checksum_algorithm](#)

[crc32](#) オプションでは、高速チェックサムアルゴリズムが使用されるため、高速ストレージシステムにお勧めします。

- [innodb_flush_neighbors](#)

ローテーションストレージデバイス用に I/O を最適化します。非ローテーション記憶域の場合、またはローテーション記憶域と非ローテーション記憶域の混在の場合は無効にします。デフォルトでは無効になっています。

- [innodb_idle_flush_pct](#)

アイドル期間中のページフラッシュに制限を設定できます。これにより、非ローテーション型ストレージデバイスの存続期間を延長できます。MySQL 8.0.18 で導入されました。

- [innodb_io_capacity](#)

一般に、デフォルト設定の 200 は、ローエンドの非ローテーション型ストレージデバイスには十分です。高エンドのバス接続デバイスの場合は、1000 などの高い設定を検討してください。

- [innodb_io_capacity_max](#)

デフォルト値 2000 は、非ローテーション記憶域を使用するワークロードを対象としています。ハイエンドのバス接続された非ローテーション型ストレージデバイスの場合は、2500 などの高い設定を検討してください。

- [innodb_log_compressed_pages](#)

redo ログが非ローテーション記憶域にある場合は、このオプションを無効にしてロギングを減らすことを検討してください。[Disable logging of compressed pages](#) を参照してください。

- [innodb_log_file_size](#)

redo ログが非繰返し記憶域にある場合は、このオプションを構成してキャッシュと書き込みの組合せを最大化します。

- [innodb_page_size](#)

ディスクの内部セクターサイズと一致するページサイズの使用を検討してください。早期世代 SSD デバイスは、多くの場合、セクターサイズが 4KB です。一部の新しいデバイスには、16K バイトのセクターサイズがあります。デフォルトの [InnoDB](#) ページサイズは 16KB です。ページサイズをストレージデバイスのブロックサイズに近づけると、ディスクに書き換えられる変更されていないデータの量が最小限に抑えられます。

- [binlog_row_image](#)

バイナリログが非ローテーションストレージ上にあり、すべてのテーブルに主キーがある場合は、ロギングを減らすためにこのオプションを [minimal](#) に設定することを検討してください。

オペレーティングシステムで TRIM サポートが有効になっていることを確認します。通常は、デフォルトで有効になっています。

- バックログを回避するための I/O 容量の増加

InnoDB チェックポイント操作のため、スループットが周期的に低下する場合、`innodb_io_capacity` 構成オプションの値を増加することを検討します。値を大きくすると、フラッシュが頻繁になり、スループットを低下させる可能性のある作業のバックログが避けられます。

- フラッシュが遅れない場合の I/O 容量の削減

InnoDB フラッシュ操作によって、システムが遅くならない場合は、`innodb_io_capacity` 構成オプションの値を小さくすることを検討します。一般に、このオプション値はできるかぎり小さくしますが、前の箇条書きで示したように、スループットに周期的な低下が発生するほど小さくしないでください。オプション値を小さくすることができる一般的なシナリオでは、`SHOW ENGINE INNODB STATUS` からの出力に、次のような組み合わせが示されることがあります。

- 履歴リストの長さが短く、数千未満です。
- 挿入バッファーマージ数が挿入された行数に近いです。
- バッファープール内の変更されたページが、一貫してバッファープールの `innodb_max_dirty_pages_pct` をはるかに下回っています。(サーバーが一括挿入を実行していないときに測定します。変更されたページの一括挿入時に、パーセンテージが大幅に高くなるのは正常です。)
- `Log sequence number - Last checkpoint` が、InnoDB ログファイルの合計サイズの 7/8 未満か、理想的には 6/8 未満です。
- Fusion-io デバイスへのシステムテーブルスペースファイルの格納

二重書き込みバッファ関連の I/O 最適化を利用するには、二重書き込み記憶域を含むファイルを、アトミック書き込みをサポートする Fusion-io デバイスに格納します。(MySQL 8.0.20 より前では、二重書き込みバッファ記憶域はシステムテーブルスペースデータファイルに存在していました。MySQL 8.0.20 では、記憶域は二重書き込みファイルに存在します。セクション15.6.4「二重書き込みバッファ」を参照してください。)二重書き込みストレージ領域ファイルがアトミック書き込みをサポートする Fusion-io デバイスに配置されると、二重書き込みバッファは自動的に無効になり、Fusion-io アトミック書き込みがすべてのデータファイルに使用されます。この機能は Fusion-io ハードウェアでのみサポートされ、Linux の Fusion-io NVMFS でのみ有効になります。この機能を最大限に活用するには、`O_DIRECT` の `innodb_flush_method` 設定をお勧めします。

注記

二重書き込みバッファ設定はグローバルであるため、Fusion-io ハードウェアに存在しないデータファイルの二重書き込みバッファも無効になります。

- 圧縮されたページのロギングの無効化

InnoDB テーブルの `compression` 機能を使用する場合、圧縮されたデータが変更されると、再圧縮された `pages` のイメージが `redo log` に書き込まれます。この動作は `innodb_log_compressed_pages` によって制御され、リカバリ中に異なるバージョンの `zlib` 圧縮アルゴリズムが使用された場合に発生する可能性のある破損を防ぐためにデフォルトで有効になっています。`zlib` のバージョンが変更されないことが確実な場合は、`innodb_log_compressed_pages` を無効にして、圧縮データを変更するワークロードの redo ログ生成を減らします。

8.5.9 InnoDB 構成変数の最適化

軽量の予測可能な負荷のあるサーバーと、常時ほぼいっぱいの容量で実行していたり、高アクティビティの急増が発生したりするサーバーとでは、もっとも適切に機能する設定が異なります。

InnoDB ストレージエンジンは、多くの最適化を自動的に実行するため、多くのパフォーマンスチューニングタスクには、データベースが適切に実行していることを確認するためのモニタリングと、パフォーマンスの低下時の構成オプションの変更が含まれます。詳細な InnoDB のパフォーマンスモニタリングについては、セクション15.16「InnoDB の MySQL パフォーマンススキーマとの統合」を参照してください。

実行できる主な構成ステップは次のようになります。

- InnoDB が変更されたデータをバッファするデータ変更操作のタイプを制御して、小さいディスク書込みの頻度を回避します。 [変更バッファリングの構成](#) を参照してください。デフォルトでは、すべてのタイプのデータ変更操作をバッファするため、バッファリングの量を減らす必要がある場合にのみ、この設定を変更します。
- `innodb_adaptive_hash_index` オプションを使用して、アダプティブハッシュインデックス機能をオンまたはオフにします。詳しくは [セクション 15.5.3 「適応型ハッシュインデックス」](#) をご覧ください。異常なアクティビティーの間にこの設定を変更し、その後、その元の設定にリストアできます。
- コンテキストスイッチングがボトルネックである場合に、InnoDB が処理する同時スレッドの数に制限を設定します。 [セクション 15.8.4 「InnoDB のスレッド並列性の構成」](#) を参照してください。
- InnoDB がその先読み操作で実行するプリフェッチの量を制御します。システムに未使用の I/O 容量がある場合、先読みによってクエリーのパフォーマンスが向上することがあります。先読みが多すぎると、負荷の大きいシステムで、パフォーマンスが周期的に低下する可能性があります。 [セクション 15.8.3.4 「InnoDB バッファープールのプリフェッチ \(先読み\) の構成」](#) を参照してください。
- デフォルト値で十分に活用されていないハイエンド I/O サブシステムがある場合、読み取りまたは書き込み操作のバックグラウンドスレッドの数を増やします。 [セクション 15.8.5 「InnoDB バックグラウンド I/O スレッドの数の構成」](#) を参照してください。
- バックグラウンドで InnoDB が実行する I/O の量を制御します。 [セクション 15.8.7 「InnoDB I/O Capacity の構成」](#) を参照してください。パフォーマンスが定期的に低下する場合は、この設定をスケールバックできます。
- InnoDB が特定の種類のバックグラウンドの書き込みを実行するタイミングを判断するアルゴリズムを制御します。 [セクション 15.8.3.5 「バッファープールのフラッシュの構成」](#) を参照してください。このアルゴリズムは一部のタイプのワークロードでは機能しますが、他のワークロードでは機能しないため、パフォーマンスが定期的に低下することがある場合は、この機能を無効にできます。
- コンテキストスイッチングの遅延を最小にするため、マルチコアプロセッサとそれらのキャッシュメモリ構成を利用します。 [セクション 15.8.8 「スピンロックのポーリングの構成」](#) を参照してください。
- テーブルスキャンなどの一度だけの操作が、InnoDB バッファークッシュに格納された頻りにアクセスされるデータを妨げることを防ぎます。 [セクション 15.8.3.3 「バッファープールをスキャンに耐えられるようにする」](#) を参照してください。
- 信頼性とクラッシュリカバリに適切なサイズにログファイルを調整します。InnoDB ログファイルは、多くの場合にクラッシュ後の長い起動時間を避けるため、小さく維持されてきました。MySQL 5.5 で導入された最適化によって、クラッシュリカバリプロセスの特定のステップが高速化します。特に、Redo ログのスキャンと Redo ログの適用は、メモリ管理のアルゴリズムの改善のため、高速化します。長い起動時間を避けるため、ログファイルを人為的に小さく維持していた場合、ログファイルサイズを拡大し、Redo ログレコードのリサイクルのために発生する I/O を削減することを考慮できるようになりました。
- InnoDB バッファープールのインスタンスのサイズと数を構成します。特に数ギガバイトのバッファープールのあるシステムに重要です。 [セクション 15.8.3.2 「複数のバッファープールインスタンスの構成」](#) を参照してください。
- 同時トランザクションの最大数を増やします。これはきわめてビジーなデータベースのスケラビリティを劇的に向上します。 [セクション 15.6.6 「undo ログ」](#) を参照してください。
- パージ操作 (ガベージコレクションの一種) をバックグラウンドスレッドに移動します。 [セクション 15.8.9 「パージ構成」](#) を参照してください。この設定の結果を効率的に測定するには、ほかの I/O 関連およびスレッド関連の構成設定を先にチューニングします。
- ビジーなサーバーで SQL 操作が列を成し、「渋滞」が発生しないように、InnoDB が同時スレッド間で実行するスイッチングの量を削減します。 `innodb_thread_concurrency` オプションの値を設定します (強力な最新のシステムで最大約 32)。 `innodb_concurrency_tickets` オプションの値を増やします (通常は 5000 など)。このオプションの組合せにより、InnoDB が一度に処理するスレッド数の上限が設定され、スワップアウトする前に各スレッドがかなりの作業を実行できるため、待機中のスレッド数は少なくなり、過剰なコンテキスト切替えなしで操作を完了できます。

8.5.10 多くのテーブルのあるシステムに対する InnoDB の最適化

- `non-persistent optimizer statistics` (デフォルト以外の構成) を構成している場合、InnoDB では、起動後に初めてそのテーブルにアクセスしたときに、そのような値をテーブルに格納するのではなく、テーブルのインデックス `cardinality` 値が計算されます。データを多くのテーブルに分割しているシステムでは、このステップに大量の時間がかかることがあります。このオーバーヘッドは最初のテーブルオープン操作にのみ適用されるため、テーブルをあとで使用するために「ウォームアップ」するには、`SELECT 1 FROM tbl_name LIMIT 1` などのステートメントを発行して、起動後すぐにそれにアクセスします。

オプティマイザ統計はデフォルトでディスクに永続化され、`innodb_stats_persistent` 構成オプションによって有効化されます。永続的オプティマイザ統計については、[セクション15.8.10.1「永続的オプティマイザ統計のパラメータの構成」](#)を参照してください。

8.6 MyISAM テーブルの最適化

MyISAM ストレージエンジンは、テーブルロックによって同時更新を実行する機能を制限するため、読み取りが大半のデータや並列性の低い操作で最適に実行します。MySQL では、InnoDB は MyISAM ではなくデフォルトのストレージエンジンです。

8.6.1 MyISAM クエリーの最適化

MyISAM テーブルのクエリーを高速化するためのいくつかの一般的なヒント:

- MySQL がクエリーをより適切に最適化できるようにするには、テーブルにデータがロードされたあとに、それに対して `ANALYZE TABLE` を使用するか、または `myisamchk --analyze` を実行します。これにより、同じ値がある平均行数を示す各インデックスパートの値を更新します。(一意のインデックスの場合、これは常に 1 です。) MySQL はこれを使用して、非定数式に基づいて、2 つのテーブルを結合する際に選択するインデックスを決定します。`SHOW INDEX FROM tbl_name` を使用し、`Cardinality` 値を調べることで、テーブル分析の結果を確認できます。`myisamchk --description --verbose` はインデックスの分布情報を示します。
- インデックスに従ってインデックスとデータをソートするには、`myisamchk --sort-index --sort-records=1` を使用します(インデックス 1 でソートすると仮定して)。インデックスに従って順番にすべての行を読み取りたいと考える一意のインデックスがある場合、これはクエリーを高速にする適切な方法です。この方法で大きなテーブルをはじめてソートするときは、長い時間がかかることがあります。
- 頻繁に更新される MyISAM テーブルに対する複雑な `SELECT` クエリーを避け、リーダーとライターの競合のために発生するテーブルロックの問題を回避するようにしてください。
- MyISAM は同時挿入をサポートしています。テーブルのデータファイルの途中に空きブロックがなければ、ほかのスレッドがテーブルから読み取るのと同時に新しい行をそれに `INSERT` できます。これを実行できることが重要な場合、行の削除を避けるようにテーブルを使用することを考慮してください。別の可能性は、テーブルの大量の行を削除したあとに `OPTIMIZE TABLE` を実行して、テーブルをデフラグすることです。この動作は `concurrent_insert` 変数の設定によって変更されます。行を削除したテーブルにも新しい行を強制的に追加(したがって同時挿入を許可)できます。[セクション8.11.3「同時挿入」](#)を参照してください。
- 頻繁に変更される MyISAM テーブルでは、すべての可変長カラム (`VARCHAR`、`TEXT`、および `BLOB`) を避けるようにします。テーブルに 1 つしか可変長カラムが含まれていない場合でも、テーブルは動的行フォーマットを使用します。[第16章「代替ストレージエンジン」](#)を参照してください。
- 一般に、行が大きくなるためだけに、1 つのテーブルを異なるテーブルに分割することは有益ではありません。行へのアクセスで、もっとも大きくパフォーマンスに打撃を与えるものは、行の先頭バイトを見つけるために必要なディスクシークです。データが見つかったあとは、ほとんどの最新のディスクで、大多数のアプリケーションに十分な速度で行全体を読み取ることができます。テーブルを分割することがかなりの違いをもたらす状況は、固定の行サイズに変更できる動的行フォーマットを使用している MyISAM テーブルの場合か、またはテーブルを著しく頻繁にスキャンする必要があるが、ほとんどのカラムには必要でない場合だけです。[第16章「代替ストレージエンジン」](#)を参照してください。
- 通常 `expr1`、`expr2`、... の順で行を取得する場合は、`ALTER TABLE ... ORDER BY expr1, expr2, ...` を使用します。テーブルを大幅に変更したあとにこのオプションを使用することで、パフォーマンスを向上させることができます。

- 多数の行の情報に基づいたカウントなど、結果を頻繁に計算する必要がある場合、新しいテーブルを導入し、リアルタイムでカウンタを更新する方が望ましいことがあります。次のような形式の更新はきわめて高速です。

```
UPDATE tbl_name SET count_col=count_col+1 WHERE key_col=constant;
```

これは、テーブルレベルのロック(単一ライターと複数リーダー)しかない **MyISAM** のような MySQL ストレージエンジンを使用する場合に、きわめて重要です。また、この場合に行ロックマネージャーが実行する必要があることは少ないため、ほとんどのデータベースシステムでパフォーマンスが向上します。

- 定期的に **OPTIMIZE TABLE** を使用して、動的フォーマット **MyISAM** テーブルの断片化を防ぎます。 [セクション 16.2.3 「MyISAM テーブルのストレージフォーマット」](#) を参照してください。
- **DELAY_KEY_WRITE=1** テーブルオプションを使用して **MyISAM** テーブルを宣言すると、テーブルが閉じられるまで、ディスクにフラッシュされないため、インデックスの更新が速くなります。ダウンサイドは、そのようなテーブルが開いているときにサーバーを強制終了する場合、**mysam_recover_options** システム変数を設定してサーバーを実行するか、サーバーを再起動する前に **mysamchk** を実行して、テーブルが正常であることを確認する必要があります。(ただし、この場合でも、キー情報は常にデータ行から生成できるため、**DELAY_KEY_WRITE** を使用しても何も失われはしません。)
- **MyISAM** インデックスでは、文字列の前後のスペースが自動的に圧縮されます。 [セクション 13.1.15 「CREATE INDEX ステートメント」](#) を参照してください。
- アプリケーションでクエリーや応答をキャッシュしてから、多くの挿入や更新をまとめて実行することによって、パフォーマンスを向上できます。この操作中にテーブルをロックすることで、すべての更新後にインデックスキャッシュが1回だけフラッシュされます。

8.6.2 MyISAM テーブルの一括データロード

これらのパフォーマンスのヒントは、 [セクション 8.2.5.1 「INSERT ステートメントの最適化」](#) の高速挿入の一般的なガイドラインを補足するものです。

- **MyISAM** テーブルでは、データファイルの途中で削除された行がない場合、**SELECT** ステートメントの実行中に同時に、同時挿入を使用して行を追加できます。 [セクション 8.11.3 「同時挿入」](#) を参照してください。
- 一部の追加作業では、テーブルに多数のインデックスがある場合に、**MyISAM** テーブルに対して **LOAD DATA** をさらに高速に実行できます。次の手順を使用します。
 1. **FLUSH TABLES** ステートメントまたは **mysqladmin flush-tables** コマンドを実行します。
 2. テーブルのインデックスのすべての使用を削除するには、**mysamchk --keys-used=0 -rq /path/to/db/tbl_name** を使用します。
 3. **LOAD DATA** を使用してテーブルにデータを挿入します。これはインデックスを更新しないため、非常に高速です。
 4. 今後、テーブルから読み取りだけをする場合は、**mysampack** を使用してそれを圧縮します。 [セクション 16.2.3.3 「圧縮テーブルの特徴」](#) を参照してください。
 5. **mysamchk -rq /path/to/db/tbl_name** を使用してインデックスを再作成します。これにより、ディスクに書き込む前にメモリーにインデックスツリーが作成されます。これは、多くのディスクシークが回避されるため、**LOAD DATA** 中にインデックスを更新するよりはるかに高速です。結果のインデックスツリーは完全にバランスも取れています。
 6. **FLUSH TABLES** ステートメントまたは **mysqladmin flush-tables** コマンドを実行します。

データを挿入する **MyISAM** テーブルが空の場合、**LOAD DATA** は前述の最適化を自動的に実行します。自動最適化とプロシージャの明示的な使用の主な違いは、**LOAD DATA** ステートメントの実行時にサーバーがインデックスの再作成に割り当てることができるよりも多くの一時メモリーを **mysamchk** でインデックス作成に割り当てることができることです。

mysamchk の代わりに次のステートメントを使用して、**MyISAM** テーブルの一意でないインデックスを無効または有効にすることもできます。これらのステートメントを使用する場合は、**FLUSH TABLES** 操作をスキップできません:

```
ALTER TABLE tbl_name DISABLE KEYS;  
ALTER TABLE tbl_name ENABLE KEYS;
```

- 非トランザクションテーブルに対して、複数ステートメントで実行される **INSERT** 操作を高速化するには、テーブルをロックします。

```
LOCK TABLES a WRITE;  
INSERT INTO a VALUES (1,23),(2,34),(4,33);  
INSERT INTO a VALUES (8,26),(6,29);  
...  
UNLOCK TABLES;
```

これは、すべての **INSERT** ステートメントの完了後に、インデックスバッファが 1 回だけディスクにフラッシュされるため、パフォーマンスにメリットがあります。通常は、**INSERT** ステートメントの数と同じだけ、インデックスバッファのフラッシュが行われます。すべての行を 1 つの **INSERT** で挿入できる場合は、明示的なロックステートメントは必要ありません。

ロックは複数接続テストの合計時間も短縮しますが、個々の接続がロックを待機するため、それらの最大待機時間は長くなることがあります。次のように 5 台のクライアントが同時に挿入の実行を試みるとします。

- 接続 1 は 1000 回の挿入を実行します
- 接続 2、3、および 4 は 1 回の挿入を実行します
- 接続 5 は 1000 回の挿入を実行します

ロックを使用しない場合、接続 2、3、および 4 は 1 と 5 の前に終了します。ロックを使用した場合、接続 2、3、および 4 は 1 または 5 の前に終了しない可能性があります、合計時間は約 40% 高速化するはずですが。

MySQL では、**INSERT**、**UPDATE**、および **DELETE** 操作はきわめて高速ですが、約 5 回超の連続した挿入や更新を実行するすべての操作の周囲にロックを追加することによって、全体のパフォーマンスを向上できます。著しく多くの連続した挿入を実行する場合、**LOCK TABLES** のあとにときどき (1,000 行程度ごとに) **UNLOCK TABLES** を実行して、ほかのスレッドのテーブルへのアクセスを許可できます。これによってもパフォーマンスの向上が得られます。

INSERT では、前述の戦略を使用している場合でも、データのロードは **LOAD DATA** よりもはるかに遅くなります。

- **MyISAM** テーブルのパフォーマンスを向上させるには、**LOAD DATA** と **INSERT** の両方について、`key_buffer_size` システム変数を増やしてキーキャッシュを拡大します。 [セクション 5.1.1 「サーバーの構成」](#) を参照してください。

8.6.3 REPAIR TABLE ステートメントの最適化

MyISAM テーブルの **REPAIR TABLE** は、修復操作に `myisamchk` を使用することと似ており、同じパフォーマンス最適化の一部が適用されます。

- `myisamchk` にはメモリー割り当てを制御する変数があります。 [セクション 4.6.4.6 「myisamchk メモリー使用量」](#) に説明するように、これらの変数を設定してパフォーマンスを向上できることがあります。
- **REPAIR TABLE** では、同じ原則が適用されますが、修復はサーバーによって実行されるため、`myisamchk` 変数の代わりに、サーバーシステム変数を設定します。また、メモリー割り当て変数の設定に加えて、`myisam_max_sort_file_size` システム変数を増やすと、修復でより高速な `filesort` 方式が使用される可能性が高くなり、キーキャッシュ方式によるより遅い修復が回避されます。テーブルファイルのコピーを保持できるだけの十分な空き領域があることを確認したら、システムの最大ファイルサイズに変数を設定します。元のテーブルファイルを格納しているファイルシステムで、空き領域が使用できる必要があります。

次のオプションを使用して、そのメモリー割り当て変数を設定して、`myisamchk` テーブル修復操作が実行されたとします。

```
--key_buffer_size=128M --myisam_sort_buffer_size=256M  
--read_buffer_size=64M --write_buffer_size=64M
```

それらの `myisamchk` 変数の一部はサーバーシステム変数に対応します。

myisamchk 変数	システム変数
key_buffer_size	key_buffer_size
myisam_sort_buffer_size	myisam_sort_buffer_size
read_buffer_size	read_buffer_size
write_buffer_size	none

各サーバーシステム変数は実行時に設定でき、それらの一部 ([myisam_sort_buffer_size](#)、[read_buffer_size](#)) にはグローバル値に加えてセッション値もあります。セッション値を設定することで、現在のセッションへの変更の影響を制限し、ほかのユーザーに影響しません。グローバルのみの変数 ([key_buffer_size](#)、[myisam_max_sort_file_size](#)) を変更すると、ほかのユーザーにも影響します。[key_buffer_size](#) の場合、バッファがそれらのユーザーと共有されることを考慮しておく必要があります。たとえば、[myisamchk key_buffer_size](#) 変数を 128M バイトに設定した場合、対応する [key_buffer_size](#) システム変数をそれより大きく設定し (それがすでに大きく設定されていない場合)、ほかのセッションのアクティビティによるキーバッファの使用を許可できます。ただし、グローバルキーバッファサイズを変更すると、バッファが無効になり、ディスク I/O が増加して、ほかのセッションが遅くなります。この問題を回避する代替策は、個別のキーキャッシュを使用し、それを修復対象のテーブルのインデックスに割り当て、修復が完了したら、その割り当てを解除することです。[セクション 8.10.2.2 「複合キーキャッシュ」](#) を参照してください。

先述の説明に基づいて、[REPAIR TABLE](#) 操作は、次のように実行して、[myisamchk](#) コマンドに似た設定を使用できます。ここでは、個別の 128M バイトのキーバッファが割り当てられ、ファイルシステムは 100G バイト以上のファイルサイズを許可するものとします。

```
SET SESSION myisam_sort_buffer_size = 256*1024*1024;
SET SESSION read_buffer_size = 64*1024*1024;
SET GLOBAL myisam_max_sort_file_size = 100*1024*1024*1024;
SET GLOBAL repair_cache.key_buffer_size = 128*1024*1024;
CACHE INDEX tbl_name IN repair_cache;
LOAD INDEX INTO CACHE tbl_name;
REPAIR TABLE tbl_name;
SET GLOBAL repair_cache.key_buffer_size = 0;
```

グローバル変数を変更するが、ほかのユーザーへの影響を最小にするため、[REPAIR TABLE](#) 操作の間のみ実行するようにしたい場合、その値をユーザー変数に保存して、あとでそれをリストアします。例:

```
SET @old_myisam_sort_buffer_size = @@GLOBAL.myisam_max_sort_file_size;
SET GLOBAL myisam_max_sort_file_size = 100*1024*1024*1024;
REPAIR TABLE tbl_name;
SET GLOBAL myisam_max_sort_file_size = @old_myisam_max_sort_file_size;
```

[REPAIR TABLE](#) に影響するシステム変数は、変数をデフォルトで有効にしたい場合、サーバーの起動時にグローバルに設定できます。たとえば、次の行をサーバーの [my.cnf](#) ファイルに追加します。

```
[mysqld]
myisam_sort_buffer_size=256M
key_buffer_size=1G
myisam_max_sort_file_size=100G
```

これらの設定には [read_buffer_size](#) は含まれません。[read_buffer_size](#) をグローバルに大きな値に設定すると、すべてのセッションに対してそれが実行され、多くの同時セッションのあるサーバーに過剰なメモリーが割り当てられるため、パフォーマンスが低下する可能性があります。

8.7 MEMORY テーブルの最適化

頻繁にアクセスされ、読み取り専用かめったに更新されない非クリティカルデータに [MEMORY](#) テーブルを使用することを考慮します。現実的なワークロードで、同等の [InnoDB](#) または [MyISAM](#) テーブルに対してアプリケーションのベンチマークを実行し、追加のパフォーマンスが、データの損失のリスクやアプリケーションの起動時にディスクベースのテーブルからデータをコピーすることのオーバーヘッドに値するかを確認します。

[MEMORY](#) テーブルで最高のパフォーマンスを得るには、各テーブルに対するクエリーの種類を調査し、関連付けられた各インデックスに使用する B ツリーインデックスまたはハッシュインデックスのいずれかの種類を指定します。[CREATE INDEX](#) ステートメントで、句 [USING BTREE](#) または [USING HASH](#) を使用します。B ツリーインデックスは、[>](#) や [BETWEEN](#) などの操作によって、greater-than または less-than の比較を実行するクエリーで高速です。ハッシュインデックスは、[=](#) 演算子によって単一の値、または [IN](#) 演算子によって制限された値のセットをルックアップするクエリーでのみ高速です。[USING BTREE](#) が多くの場合にデフォルトの [USING HASH](#) より適切な選択

である理由については、[セクション8.2.1.23「全テーブルスキャンの回避」](#)を参照してください。さまざまな種類の `MEMORY` インデックスの実装の詳細については、[セクション8.3.9「B ツリーインデックスとハッシュインデックスの比較」](#)を参照してください。

8.8 クエリー実行プランの理解

`WHERE` 句内のテーブル、カラム、インデックス、および条件の詳細に応じて、MySQL オプティマイザは SQL クエリーに含まれるルックアップを効率的に実行するための多くの技法を考慮します。巨大なテーブルに対するクエリーは、すべての行を読み取らなくても実行でき、複数のテーブルを含む結合は、行のすべての組み合わせを比較しなくても実行できます。オプティマイザがもっとも効率的なクエリーを実行するために選択する操作のセットは、「クエリー実行プラン」と呼ばれ、`EXPLAIN` プランとも呼ばれます。目的は、クエリーが適切に最適化されていることを示す `EXPLAIN` プランの側面を認識し、非効率的な操作が見られた場合に、プランを改善するための SQL 構文とインデックス設定技法を学ぶことです。

8.8.1 EXPLAIN によるクエリーの最適化

`EXPLAIN` ステートメントは、MySQL がステートメントをどのように実行するかに関する情報を提供します。

- `EXPLAIN` は、`SELECT`、`DELETE`、`INSERT`、`REPLACE` および `UPDATE` ステートメントで動作します。
- 説明可能なステートメントで `EXPLAIN` を使用すると、MySQL は、オプティマイザからのステートメント実行プランに関する情報を表示します。つまり、MySQL はテーブルがどのように、どんな順番で結合されているかに関する情報を含む、ステートメントを処理する方法を説明します。`EXPLAIN` を使用して、実行プラン情報を取得することについては、[セクション8.8.2「EXPLAIN 出力フォーマット」](#)を参照してください。
- `EXPLAIN` を説明可能なステートメントではなく `FOR CONNECTION connection_id` とともに使用すると、名前付き接続で実行されているステートメントの実行計画が表示されます。[セクション8.8.4「名前付き接続の実行計画情報の取得」](#)を参照してください。
- `SELECT` ステートメントの場合、`EXPLAIN` は、`SHOW WARNINGS` を使用して表示できる追加の実行計画情報を生成します。[セクション8.8.3「拡張 EXPLAIN 出力形式」](#)を参照してください。
- `EXPLAIN` は、パーティションテーブルを含むクエリーの調査に役立ちます。[セクション24.3.5「パーティションに関する情報を取得する」](#)を参照してください。
- `FORMAT` オプションを使用して、出力形式を選択できます。`TRADITIONAL` は表形式で出力を表示します。`FORMAT` オプションが存在しない場合、これはデフォルトです。`JSON` フォーマットは JSON フォーマットで情報を表示します。

`EXPLAIN` によって、インデックスを使用して行を見つけることで、ステートメントが高速に実行されるように、テーブルにインデックスを追加すべき場所がわかります。また、`EXPLAIN` を使用して、オプティマイザがテーブルを最適な順序で結合しているかどうかを確認することもできます。`SELECT` ステートメントでテーブルが指定されている順序に対応する結合順序を使用するように、オプティマイザにヒントを提供するには、ステートメントを `SELECT` だけでなく、`SELECT STRAIGHT_JOIN` で始めます。(セクション13.2.10「`SELECT` ステートメント」を参照してください。)ただし、準結合変換が無効になっているため、`STRAIGHT_JOIN` ではインデックスが使用されない場合があります。[セクション8.2.2.1「準結合変換による IN および EXISTS サブクエリー述語の最適化」](#)を参照してください。

オプティマイザトレースは、`EXPLAIN` のトレースを補完する情報を提供する場合があります。ただし、オプティマイザのトレース形式と内容はバージョン間で変更される可能性があります。詳細については、「[MySQL Internals: Tracing the Optimizer](#)」を参照してください。

インデックスが使われるはずであると思うタイミングでそれらが使われていない問題がある場合、`ANALYZE TABLE` を実行して、オプティマイザが行う選択に影響する可能性があるキーのカーディナリティーなどのテーブル統計を更新します。[セクション13.7.3.1「ANALYZE TABLE ステートメント」](#)を参照してください。

注記

`EXPLAIN` はテーブル内のカラムに関する情報を取得するためにも使用できます。`EXPLAIN tbl_name` は `DESCRIBE tbl_name` および `SHOW COLUMNS FROM tbl_name` と同義です。詳細については、[セクション13.8.1「DESCRIBE ステートメント」](#) および [セクション13.7.7.5「SHOW COLUMNS ステートメント」](#)を参照してください。

8.8.2 EXPLAIN 出力フォーマット

EXPLAIN ステートメントは、MySQL がステートメントを実行する方法に関する情報を提供します。 **EXPLAIN** は、**SELECT**、**DELETE**、**INSERT**、**REPLACE** および **UPDATE** ステートメントで動作します。

EXPLAIN は **SELECT** ステートメントで使用される各テーブルに関する情報の行を返します。これは、MySQL がステートメントの処理中にテーブルを読み取る順番で、出力にテーブルを一覧表示します。これは、MySQL が最初のテーブルから行を読み取り、次に 2 番目のテーブル、3 番目のテーブルなどで一致する行を検索することを意味します。すべてのテーブルが処理されると、MySQL は選択したカラムを出力し、さらに一致する行があるテーブルが見つかるまで、テーブルリストを逆戻りします。次の行がテーブルから読み取られ、プロセスは次のテーブルに進みます。

注記

MySQL Workbench には、**EXPLAIN** 出力を視覚的に表現する Visual Explain 機能があります。 [Tutorial: Using Explain to Improve Query Performance](#) を参照してください。

- [EXPLAIN 出力カラム](#)
- [EXPLAIN 結合型](#)
- [EXPLAIN 追加情報](#)
- [EXPLAIN 出力の解釈](#)

EXPLAIN 出力カラム

このセクションでは、**EXPLAIN** によって生成される出力カラムについて説明します。あとのセクションで、**type** と **Extra** カラムに関する追加情報を提供します。

EXPLAIN からの各出力行は 1 つのテーブルに関する情報を提供します。各行には、表 8.1 「**EXPLAIN 出力カラム**」で要約し、次の表に詳しく説明している値が格納されます。テーブルの最初のカラムにはカラム名が表示されます。2 番目のカラムには、**FORMAT=JSON** を使用した場合の出力に表示される同等のプロパティ名が表示されます。

表 8.1 EXPLAIN 出力カラム

カラム	JSON 名	意味
id	select_id	SELECT 識別子。
select_type	なし	SELECT 型
table	table_name	出力行のテーブル
partitions	partitions	一致するパーティション
type	access_type	結合型
possible_keys	possible_keys	選択可能なインデックス
key	key	実際に選択されたインデックス
key_len	key_length	選択されたキーの長さ
ref	ref	インデックスと比較されるカラム
rows	rows	調査される行の見積もり
filtered	filtered	テーブル条件によってフィルタ処理される行の割合
Extra	なし	追加情報

注記

NULL である JSON プロパティは、JSON 形式の **EXPLAIN** 出力には表示されません。

- id (JSON 名): `select_id`

SELECT 識別子。これはクエリー内の **SELECT** の連番です。行がほかの行の和集合結果を参照する場合に、値は **NULL** になることがあります。この場合、**table** カラムには、**<unionM,N>** などの値が表示され、行が **M** および **N** の **id** 値のある行の和集合を参照していることが示されます。

- **select_type** (JSON 名): none)

SELECT の種類で、次の表に示すもののいずれかになります。JSON 形式の **EXPLAIN** は、**SIMPLE** または **PRIMARY** でないかぎり、**SELECT** タイプを **query_block** のプロパティとして公開します。JSON 名 (該当する場合) もテーブルに示されます。

select_type 値	JSON 名	意味
SIMPLE	なし	単純な SELECT (UNION やサブクエリーを使用しません)
PRIMARY	なし	もっとも外側の SELECT
UNION	なし	UNION 内の 2 つめ以降の SELECT ステートメント
DEPENDENT UNION	dependent (true)	UNION 内の 2 つめ以降の SELECT ステートメントで、外側のクエリーに依存します
UNION RESULT	union_result	UNION の結果。
SUBQUERY	なし	サブクエリー内の最初の SELECT
DEPENDENT SUBQUERY	dependent (true)	サブクエリー内の最初の SELECT で、外側のクエリーに依存します
DERIVED	なし	導出テーブル
DEPENDENT DERIVED	dependent (true)	別のテーブルに依存する導出テーブル
MATERIALIZED	materialized_from_subquery	実体化されたサブクエリー
UNCACHEABLE SUBQUERY	cacheable (false)	結果をキャッシュできず、外側のクエリーの行ごとに再評価される必要があるサブクエリー
UNCACHEABLE UNION	cacheable (false)	キャッシュ不可能なサブクエリー (UNCACHEABLE SUBQUERY を参照してください) に属する UNION 内の 2 つめ以降の SELECT

DEPENDENT は一般に、相関サブクエリーの使用を示します。 [セクション13.2.11.7「相関サブクエリー」](#) を参照してください。

DEPENDENT SUBQUERY の評価は **UNCACHEABLE SUBQUERY** の評価とは異なります。 **DEPENDENT SUBQUERY** の場合、その外部コンテキストの変数の異なる値の各セットにつき、一回だけサブクエリーが再評価されます。 **UNCACHEABLE SUBQUERY** の場合、外部コンテキストの行ごとにサブクエリーが再評価されます。

EXPLAIN で **FORMAT=JSON** を指定した場合、出力には **select_type** と直接同等の単一のプロパティはありません。 **query_block** プロパティは特定の **SELECT** に対応します。表示されているほとんどの **SELECT** サブクエリータイプに相当するプロパティが使用可能で (たとえば、**materialized_from_subquery** for **MATERIALIZED**)、必要に応じて表示されます。 **SIMPLE** または **PRIMARY** に相当する JSON はありません。

SELECT 以外のステートメントの **select_type** 値には、影響を受けるテーブルのステートメントタイプが表示されます。たとえば、**select_type** は **DELETE** ステートメント用の **DELETE** です。

- `table` (JSON 名): `table_name`)

出力の行で参照しているテーブルの名前。これも次のいずれかの値になることがあります。

- `<unionM,N>`: 行は `M` および `N` の `id` 値のある行の和集合を参照しています。
- `<derivedN>`: 行は `N` の `id` 値のある行の派生テーブル結果を参照しています。派生テーブルは、たとえば `FROM` 句内のサブクエリーの結果などになります。
- `<subqueryN>`: 行は `N` の `id` 値のある行の実体化されたサブクエリーの結果を参照しています。 [セクション 8.2.2.2 「実体化を使用したサブクエリーの最適化」](#) を参照してください。

- `partitions` (JSON 名): `partitions`)

クエリーでレコードが照合されるパーティション。パーティション化されていないテーブルの場合、この値は `NULL` です。 [セクション 24.3.5 「パーティションに関する情報を取得する」](#) を参照してください。

- `type` (JSON 名): `access_type`)

結合型。さまざまな型の説明については、「[EXPLAIN 結合型](#)」を参照してください。

- `possible_keys` (JSON 名): `possible_keys`)

`possible_keys` カラムは、MySQL がこのテーブルの行を検索するために選択できるインデックスを示します。このカラムは `EXPLAIN` の出力に表示されたテーブルの順序にまったく依存しません。つまり、`possible_keys` のキーの一部は、生成されたテーブルの順序で実際に使用できないことがあります。

このカラムが `NULL` の場合 (または JSON 形式の出力で未定義の場合)、関連するインデックスはありません。この場合、`WHERE` 句を調査して、それがインデックス設定に適したカラムを参照しているかどうかをチェックすることで、クエリーのパフォーマンスを向上させることができます。その場合は、適切なインデックスを作成し、再度 `EXPLAIN` でクエリーをチェックします。 [セクション 13.1.9 「ALTER TABLE ステートメント」](#) を参照してください。

テーブルにあるインデックスを確認するには、`SHOW INDEX FROM tbl_name` を使用します。

- `key` (JSON 名): `key`)

`key` カラムは、MySQL が実際に使用することを決定したキー (インデックス) を示します。MySQL が行をルックアップするために、いずれかの `possible_keys` インデックスを使用することを決定した場合、キー値としてそのインデックスが一覧表示されます。

`key` は、`possible_keys` 値に存在しないインデックスに名前を付けることができます。これは `possible_keys` インデックスのどれも行のルックアップに適していない場合に発生する可能性があります。クエリーによって選択されるすべてのカラムはほかのインデックスのカラムになります。つまり、指定されたインデックスは選択されたカラムをカバーするため、取得する行を決定するために使用されませんが、インデックススキャンはデータ行スキャンよりも効率的です。

InnoDB は各セカンダリインデックスとともに主キー値を保存するため、InnoDB では、クエリーで主キーも選択している場合でも、セカンダリインデックスで選択されたカラムをカバーしている可能性があります。`key` が `NULL` の場合、MySQL はクエリーをより効率的に実行するために使用するインデックスを見つけれませんでした。

MySQL で `possible_keys` カラムに示されたインデックスを強制的に使用させるか、無視させるには、クエリーで `FORCE INDEX`、`USE INDEX`、または `IGNORE INDEX` を使用します。 [セクション 8.9.4 「インデックスヒント」](#) を参照してください。

MyISAM テーブルの場合、`ANALYZE TABLE` を実行すると、オプティマイザがより適切なインデックスを選択するのに役立ちます。MyISAM テーブルの場合、`myisamchk --analyze` も同様に動作します。 [セクション 13.7.3.1 「ANALYZE TABLE ステートメント」](#) および [セクション 7.6 「MyISAM テーブルの保守とクラッシュリカバリ」](#) を参照してください。

- `key_len` (JSON 名): `key_length`)

`key_len` カラムは、MySQL が使用することを決定したキーの長さを示します。 `key_len` の値を使用すると、MySQL が実際に使用するマルチパーティキーの部分の数を決定できます。 `key` カラムに `NULL` と表示されている場合、`key_len` カラムにも `NULL` と表示されます。

キーの格納形式のため、キーの長さは、`NULL` にできるカラムの長さが `NOT NULL` カラムの長さより大きくなります。

- `ref` (JSON 名): `ref`)

`ref` カラムは、テーブルから行を選択するために、`key` カラムに指定されたインデックスに対して比較されるカラムまたは定数を示します。

値が `func` の場合、使用される値は、特定の関数の結果です。 どの関数を表示するには、`EXPLAIN` の後の `SHOW WARNINGS` を使用して、拡張 `EXPLAIN` 出力を表示します。 関数は、実際には算術演算子などの演算子である場合があります。

- `rows` (JSON 名): `rows`)

`rows` カラムは、MySQL がクエリーを実行するために調査する必要があると考える行数を示します。

InnoDB テーブルの場合、これは推定値であり、常に正確ではないことがあります。

- `filtered` (JSON 名): `filtered`)

`filtered` カラムは、テーブル条件でフィルタされるテーブルの行の推定割合を示します。 最大値は 100 で、これは行のフィルタリングが行われなかったことを意味します。 100 から減少する値は、フィルタリングの量が増加していることを示します。 `rows` には調査された推定行数が表示され、`rows×filtered` には次のテーブルと結合された行数が表示されます。 たとえば、`rows` が 1000 で `filtered` が 50.00 (50%) の場合、次のテーブルと結合される行数は $1000 \times 50\% = 500$ になります。

- `Extra` (JSON 名): `none`)

このカラムには、MySQL がクエリーを解決する方法に関する追加情報が含まれます。 さまざまな値の説明については、「[EXPLAIN の追加情報](#)」を参照してください。

`Extra` カラムに対応する単一の JSON プロパティはありませんが、このカラムで発生する可能性のある値は JSON プロパティまたは `message` プロパティのテキストとして公開されます。

EXPLAIN 結合型

`EXPLAIN` 出力の `type` カラムには、テーブルの結合方法が示されます。 JSON 形式の出力では、これらは `access_type` プロパティの値として検出されます。 次のリストに、もっとも適切な型からもっとも不適切な型の順番で並べた結合型を示します。

- `system`

テーブルには行が 1 つしかありません (= system テーブル)。 これは、`const` 結合型の特殊なケースです。

- `const`

テーブルには、一致するレコードが最大で 1 つあり、クエリーの開始時に読み取られます。 行が 1 つしかないため、この行のカラムの値は、最適化エンジンの残りによって定数とみなされることがあります。 `const` テーブルは、1 回しか読み取られないため、非常に高速です。

`const` は `PRIMARY KEY` または `UNIQUE` インデックスのすべての部分を定数値と比較する場合に使用されます。 次のクエリーでは、`tbl_name` は `const` テーブルとして使用できます。

```
SELECT * FROM tbl_name WHERE primary_key=1;
```

```
SELECT * FROM tbl_name
WHERE primary_key_part1=1 AND primary_key_part2=2;
```

- `eq_ref`

前のテーブルの行の組み合わせごとに、このテーブルから 1 行ずつ読み取られます。 `system` と `const` 型以外で、これは最適な結合型です。これは、結合でインデックスのすべてのパートが使用されており、インデックスが `PRIMARY KEY` または `UNIQUE NOT NULL` インデックスである場合に使用されます。

`eq_ref` は、`=` 演算子を使用して比較されるインデックス設定されたカラムに使用できます。比較値は、定数またはこのテーブルより前に読み取られたテーブルのカラムを使用する式を指定できます。次の例では、MySQL は `eq_ref` 結合を使用して、`ref_table` を処理できます。

```
SELECT * FROM ref_table,other_table
WHERE ref_table.key_column=other_table.column;

SELECT * FROM ref_table,other_table
WHERE ref_table.key_column_part1=other_table.column
AND ref_table.key_column_part2=1;
```

- `ref`

前のテーブルの行の組み合わせごとに、一致するインデックス値を持つすべての行がこのテーブルから読み取られます。`ref` は、結合でキーの左端のプリフィクスのみが使用される場合、またはキーが `PRIMARY KEY` や `UNIQUE` インデックスではない場合 (つまり、結合で、キー値に基づいて単一の行を選択できない場合) に使用されます。使用されているキーがほんの数行にしか一致しない場合、これは適切な結合型です。

`ref` は、`=` または `<=>` 演算子を使用して比較されるインデックス設定されたカラムに使用できます。次の例では、MySQL は `ref` 結合を使用して、`ref_table` を処理できます。

```
SELECT * FROM ref_table WHERE key_column=expr;

SELECT * FROM ref_table,other_table
WHERE ref_table.key_column=other_table.column;

SELECT * FROM ref_table,other_table
WHERE ref_table.key_column_part1=other_table.column
AND ref_table.key_column_part2=1;
```

- `fulltext`

結合は `FULLTEXT` インデックスを使用して実行されます。

- `ref_or_null`

この結合型は、`ref` と似ていますが、MySQL が `NULL` 値を含む行の追加検索を実行することが追加されます。この結合型の最適化は、ほとんどの場合に、サブクエリーの解決で使用されます。次の例では、MySQL は `ref_or_null` 結合を使用して、`ref_table` を処理できます。

```
SELECT * FROM ref_table
WHERE key_column=expr OR key_column IS NULL;
```

[セクション8.2.1.15 「IS NULL の最適化」](#) を参照してください。

- `index_merge`

この結合型はインデクスマージ最適化が使用されたことを示します。この場合、出力行の `key` カラムには使用されたインデックスのリストが含まれ、`key_len` には使用されたインデックスの最長キーパートのリストが含まれます。詳細については、[セクション8.2.1.3 「インデクスマージの最適化」](#) を参照してください。

- `unique_subquery`

このタイプは、次の形式の一部の `IN` サブクエリーで `eq_ref` に置き換わります:

```
value IN (SELECT primary_key FROM single_table WHERE some_expr)
```

`unique_subquery` は、効率化のため、サブクエリーを完全に置き換える単なるインデックスルックアップ関数です。

- `index_subquery`

この結合型は `unique_subquery` に似ています。IN サブクエリーを置き換えますが、次の形式のサブクエリー内の一意でないインデックスに対して機能します。

```
value IN (SELECT key_column FROM single_table WHERE some_expr)
```

- `range`

行を選択するためのインデックスを使用して、特定の範囲にある行のみが取得されます。出力行の `key` カラムは、使用されるインデックスを示します。 `key_len` には使用された最長のインデックスパートが格納されます。この型の `ref` カラムは `NULL` です。

`range` は、`=`, `<>`, `>`, `>=`, `<`, `<=`, `IS NULL`, `<=>`, `BETWEEN`, `LIKE` または `IN()` 演算子のいずれかを使用してキーカラムを定数と比較する場合に使用できます:

```
SELECT * FROM tbl_name
WHERE key_column = 10;

SELECT * FROM tbl_name
WHERE key_column BETWEEN 10 and 20;

SELECT * FROM tbl_name
WHERE key_column IN (10,20,30);

SELECT * FROM tbl_name
WHERE key_part1 = 10 AND key_part2 IN (10,20,30);
```

- `インデックス`

`index` 結合型は、インデックスツリーがスキャンされることを除いて、`ALL` と同じです。これは 2 つの方法で行われます。

- インデックスがクエリーのカバリングインデックスで、使用すると、テーブルから必要なすべてのデータを満たすことができる場合、インデックスツリーのみがスキャンされます。この場合、`Extra` カラムには `Using index` と示されます。インデックスのサイズは通常テーブルデータより小さいため、インデックスのみのスキャンは通常、`ALL` より高速です。
- フルテーブルスキャンは、インデックスからの読み取りを使用して、インデックス順でデータ行をルックアップして実行されます。`Extra` カラムに `Uses index` が表示されません。

MySQL は、クエリーで単一のインデックスの一部であるカラムのみが使用されている場合に、この結合型を使用できます。

- `ALL`

フルテーブルスキャンは、前のテーブルの行の組み合わせごとに実行されます。これは、通常テーブルが `const` とマークされていない最初のテーブルである場合には適しておらず、通常ほかのすべてのケースで著しく不適切です。通常、定数値または以前のテーブルからのカラム値に基づいて、テーブルからの行の取得を可能にするインデックスを追加することで、`ALL` を回避できます。

EXPLAIN 追加情報

`EXPLAIN` 出力の `Extra` カラムには、MySQL がクエリーを解決する方法に関する追加情報が含まれます。次のリストに、このカラムに表示される可能性のある値について説明します。各セクション目は、JSON 形式の出力に対して、`Extra` 値を表示するプロパティも示します。これらの一部には、特定のプロパティがあります。その他は、`message` プロパティのテキストとして表示されます。

クエリーをできるだけ高速にする場合は、`Using filesort` および `Using temporary` の `Extra` カラムの値を検索するか、JSON 形式の `EXPLAIN` 出力で `using_filesort` および `using_temporary_table` のプロパティが `true` に等しいかどうかを調べます。

- `Child of 'table' pushed join@1` (JSON: `message` テキスト)

このテーブルは、NDB カーネルにプッシュダウンできる結合内の `table` の子として参照されます。プッシュダウン結合が有効になっている場合、NDB Cluster でのみ適用されます。詳細と例については、`ndb_join_pushdown` サーバースystem変数の説明を参照してください。

- `const row not found` (JSON プロパティ): `const_row_not_found`)
`SELECT ... FROM tbl_name` などのクエリーの場合、テーブルは空でした。
- `Deleting all rows` (JSON プロパティ): `message`)
`DELETE` に対し、一部のストレージエンジン (`MyISAM` など) は簡単で高速にすべての行テーブルを削除するハンドラメソッドをサポートしています。この `Extra` 値は、エンジンでこの最適化が使用された場合に表示されます。
- `Distinct` (JSON プロパティ): `distinct`)
MySQL は個別の値を検索するため、最初に一致する行が見つかったら、現在の行の組み合わせについてのそれ以上の行の検索を停止します。
- `FirstMatch(tbl_name)` (JSON プロパティ): `first_match`)
準結合 `FirstMatch` 結合ショートカット戦略は、`tbl_name` に使用されます。
- `Full scan on NULL key` (JSON プロパティ): `message`)
これは、オプティマイザがインデックスルックアップアクセスメソッドを使用できない場合の代替の戦略として、サブクエリーの最適化で行われます。
- `Impossible HAVING` (JSON プロパティ): `message`)
`HAVING` 句は常に `false` で、どの行も選択できません。
- `Impossible WHERE` (JSON プロパティ): `message`)
`WHERE` 句は常に `false` で、どの行も選択できません。
- `Impossible WHERE noticed after reading const tables` (JSON プロパティ): `message`)
MySQL はすべての `const` (および `system`) テーブルを読み取り、`WHERE` 句が常に `false` であることを通知します。
- `LooseScan(m..n)` (JSON プロパティ): `message`)
準結合 `LooseScan` 戦略が使用されます。 `m` および `n` は主要な部品番号です。
- `No matching min/max row` (JSON プロパティ): `message`)
`SELECT MIN(...)` `FROM ... WHERE condition` などのクエリーの条件を満たす行がありません。
- `no matching row in const table` (JSON プロパティ): `message`)
結合のあるクエリーで、空のテーブルまたは一意のインデックス条件を満足する行がないテーブルがありました。
- `No matching rows after partition pruning` (JSON プロパティ): `message`)
`DELETE` または `UPDATE` に対し、オプティマイザはパーティションのプルーニング後に削除または更新するものが何も見つかりませんでした。それは、`SELECT` ステートメントの `Impossible WHERE` に意味が似ています。
- `No tables used` (JSON プロパティ): `message`)
クエリーに `FROM` 句がないか、`FROM DUAL` 句があります。
`INSERT` または `REPLACE` ステートメントで、`SELECT` パートがない場合に、`EXPLAIN` にこの値が表示されます。たとえば、`EXPLAIN INSERT INTO t VALUES(10)` に対して、それは `EXPLAIN INSERT INTO t SELECT 10 FROM DUAL` と同等であるために表示されます。
- `Not exists` (JSON プロパティ): `message`)

MySQL はクエリーに対する **LEFT JOIN** 最適化を実行でき、**LEFT JOIN** 条件に一致する 1 つの行が見つかったら、前の行の組み合わせについて、このテーブルでそれ以上の行を調査しません。これは、このように最適化できるクエリーの種類の例です。

```
SELECT * FROM t1 LEFT JOIN t2 ON t1.id=t2.id
WHERE t2.id IS NULL;
```

t2.id が **NOT NULL** で定義されているとします。この場合、MySQL は **t1** をスキャンし、**t1.id** の値を使用して **t2** 内の行をルックアップします。MySQL が **t2** 内に一致する行を見つけた場合、**t2.id** は **NULL** にならないことがわかっているため、同じ **id** 値を持つ **t2** 内の残りの行をスキャンしません。つまり、**t1** の各行について、MySQL は、**t2** 内の実際に一致する行数にかかわらず、**t2** 内の単一のルックアップのみを実行する必要があります。

MySQL 8.0.17 以降では、**NOT IN (subquery)** または **NOT EXISTS (subquery)** 形式の **WHERE** 条件が内部的にアンチ結合に変換されたことを示すこともできます。これにより、サブクエリーが削除され、そのテーブルが最上位のクエリーの計画に追加され、コスト計画が改善されます。準結合とアンチ結合をマージすることで、オプティマイザは実行計画内のテーブルの順序をより自由に変更できるため、計画が高速になる場合があります。

特定のクエリーに対してアンチ結合変換が実行されるタイミングを確認するには、**EXPLAIN** の実行後に **SHOW WARNINGS** から **Message** カラムを確認するか、**EXPLAIN FORMAT=TREE** の出力で確認します。

注記

アンチ結合は、準結合 **table_a JOIN table_b ON condition** を補完したものです。アンチ結合では、**condition** に一致する行が **table_b** にない **table_a** のすべての行が返されます。

- **Plan isn't ready yet (JSON プロパティ): none**

この値は、オプティマイザが名前付き接続で実行中のステートメントの実行計画の作成を終了していない場合に、**EXPLAIN FOR CONNECTION** で発生します。実行計画の出力が複数の行で構成されている場合、オプティマイザが完全な実行計画を決定する進行状況に応じて、そのいずれかまたはすべてがこの **Extra** 値を持つ可能性があります。

- **Range checked for each record (index map: N) (JSON プロパティ): message**

MySQL は使用に適したインデックスを見つけられませんでした。前のテーブルからのカラム値がわかったあとに、いくつかのインデックスが使用できることがわかりました。以前のテーブルの行の組み合わせごとに、MySQL は **range** または **index_merge** アクセスメソッドを使用して、行を取得できるかどうかをチェックします。これは、非常に高速ではありませんが、インデックスがまったくない結合の実行より高速です。前のテーブルのすべてのカラム値がわかっており、定数とみなされることを除き、適用基準は、**セクション 8.2.1.2 「range の最適化」** と **セクション 8.2.1.3 「インデックスマージの最適化」** で説明されているとおりです。

インデックスは、テーブルの **SHOW INDEX** に示される同じ順序で 1 から番号付けされます。インデックスマップ値 **N** は、候補となるインデックスを示すビットマスク値です。たとえば、**0x19** (バイナリ 11001) の値は、インデックス 1、4、および 5 が考慮されることを意味します。

- **Recursive (JSON プロパティ): recursive**

これは、行が再帰的共通テーブル式の再帰的 **SELECT** 部分に適用されることを示します。**セクション 13.2.15 「WITH (共通テーブル式)」** を参照してください。

- **Rematerialize (JSON プロパティ): rematerialize**

Rematerialize (X,...) は、**T** テーブルの **EXPLAIN** 行に表示されます。**X** は、**T** の新しい行が読み取られたときに再実体化がトリガーされるラテラル導出テーブルです。例:

```
SELECT
...
FROM
t,
LATERAL (derived table that refers to t) AS dt
...
```

導出テーブルの内容は、上位クエリーによって **t** の新しい行が処理されるたびに最新になるように再実体化されません。

- `Scanned N databases` (JSON プロパティ): `message`)

これは、[セクション8.2.3「INFORMATION_SCHEMA クエリーの最適化」](#)に説明するように、サーバーが `INFORMATION_SCHEMA` テーブルのクエリーを処理する際に実行するディレクトリスキャンの数を示します。 `N` の値は 0、1、または `all` です。

- `Select tables optimized away` (JSON プロパティ): `message`)

オプティマイザは、1) 最大 1 つの行を戻す必要があると判断しました。2) この行を生成するには、確定的な行セットを読み取る必要があります。読み取り対象の行を最適化フェーズ中 (インデックス行の読み取りなど) に読み取ることができる場合、クエリーの実行中にテーブルを読み取る必要はありません。

最初の条件は、クエリーが暗黙的にグループ化されるときに満たされます (集計関数は含まれますが、`GROUP BY` 句は含まれません)。2 番目の条件は、使用されるインデックスごとに 1 つの行検索が実行されるときに満たされます。読み取られるインデックスの数によって、読み取る行数が決まります。

暗黙的にグループ化された次のクエリーについて考えてみます:

```
SELECT MIN(c1), MIN(c2) FROM t1;
```

あるインデックス行を読み取ることで `MIN(c1)` を取得でき、別のインデックスからある行を読み取ることで `MIN(c2)` を取得できるとします。つまり、カラム `c1` および `c2` ごとに、カラムがインデックスの最初のカラムであるインデックスが存在します。この場合、2 つの決定的な行を読み取ることによって生成された 1 つの行が返されます。

読み取る行が決定的でない場合、この `Extra` 値は発生しません。次のクエリーについて考えてみます:

```
SELECT MIN(c2) FROM t1 WHERE c1 <= 10;
```

`(c1, c2)` がカバーインデックスであるとして、このインデックスを使用して、`c1 <= 10` のすべての行をスキャンし、`c2` の最小値を検索する必要があります。対照的に、次のクエリーについて考えてみます:

```
SELECT MIN(c2) FROM t1 WHERE c1 = 10;
```

この場合、`c1 = 10` の最初のインデックス行には、`c2` の最小値が含まれます。返される行を生成するには、1 つの行のみを読み取る必要があります。

テーブルごとに正確な行数を保持するストレージエンジン (`MyISAM` など、`InnoDB` は保持しない) の場合、`WHERE` 句が欠落しているか常に `true` で、`GROUP BY` 句がない `COUNT(*)` クエリーに対してこの `Extra` 値が発生することがあります。(これは暗黙的にグループ化されたクエリーのインスタンスであり、ストレージエンジンは確定的な行数を読み取ることができるかどうかに影響します。)

- `Skip_open_table`, `Open_frm_only`, `Open_full_table` (JSON プロパティ): `message`)

これらの値は、`INFORMATION_SCHEMA` テーブルのクエリーに適用されるファイルオープンの最適化を示します。

- `Skip_open_table`: テーブルファイルを開く必要はありません。この情報はデータディクショナリからすでに使用可能です。
- `Open_frm_only`: テーブル情報を読み取る必要があるのはデータディクショナリのみです。
- `Open_full_table`: 最適化されていない情報参照。テーブル情報は、データディクショナリから、およびテーブルファイルを読み取ることによって読み取る必要があります。

- `Start temporary`, `End temporary` (JSON プロパティ): `message`)

これは、準結合重複除去ストラテジの一時テーブルの使用を示します。

- `unique row not found` (JSON プロパティ): `message`)

`SELECT ... FROM tbl_name` などのクエリーの場合に、テーブルに `UNIQUE` インデックスまたは `PRIMARY KEY` の条件を満たす行がありません。

- `Using filesort` (JSON プロパティ): `using_filesort`)

MySQL はソート順で行を取得する方法を見つけるために、追加のパスを実行する必要があります。ソートは、結合型に従ってすべての行を進み、ソートキーと `WHERE` 句に一致するすべての行について行へのポインタを格納して実行されます。次にキーがソートされ、ソート順で行が取得されます。 [セクション8.2.1.16「ORDER BY の最適化」](#) を参照してください。

- [Using index \(JSON プロパティ\): `using_index`](#))

実際の行を読み取るための追加のシークを実行する必要がなく、インデックスツリーの情報のみを使用して、テーブルからカラム情報が取得されます。この戦略は、クエリーで単一のインデックスの一部であるカラムのみを使用している場合に使用できます。

ユーザー定義のクラスタ化されたインデックスを持つ InnoDB テーブルの場合、そのインデックスは `Extra` カラムに `Using index` がない場合でも使用できます。これは、`type` が `index` で `key` が `PRIMARY` の場合です。

- [Using index condition \(JSON プロパティ\): `using_index_condition`](#))

インデックスタプルにアクセスし、まずそれらをテストして、すべてのテーブル行を読み取るかどうかを判断することによって、テーブルが読み取られます。このように、必要でないがぎり、すべてのテーブル行の読み取りを遅延(「プッシュダウン」)するためにインデックス情報が使用されます。 [セクション8.2.1.6「インデックスコンディションプッシュダウンの最適化」](#) を参照してください。

- [Using index for group-by \(JSON プロパティ\): `using_index_for_group_by`](#))

`Using index` テーブルアクセスメソッドと同様に、`Using index for group-by` は MySQL が、実際のテーブルへの追加のディスクアクセスをせずに、`GROUP BY` または `DISTINCT` クエリーのすべてのカラムを取得するために使用できるインデックスを見つけたことを示します。さらに、各グループに対して、少数のインデックスエントリだけが読み取られるように、インデックスがもっとも効率的に使われます。詳細は、 [セクション8.2.1.17「GROUP BY の最適化」](#) を参照してください。

- [Using index for skip scan \(JSON プロパティ\): `using_index_for_skip_scan`](#))

スキップスキャンアクセスメソッドが使用されていることを示します。 [スキャン範囲アクセス方法のスキップ](#) を参照してください。

- [Using join buffer \(Block Nested Loop\), Using join buffer \(Batched Key Access\), Using join buffer \(hash join\) \(JSON プロパティ\): `using_join_buffer`](#))

初期の結合からのテーブルは、部分ごとに結合バッファに読み込まれ、それらの行がバッファから使用されて、現在のテーブルとの結合が実行されます。(`Block Nested Loop`) ではブロックネスト - ループアルゴリズムが使用され、(`Batched Key Access`) ではバッチキーアクセスアルゴリズムが使用され、(`hash join`) ではハッシュ結合が使用されます。つまり、`EXPLAIN` 出力の前の行にあるテーブルのキーがバッファされ、`Using join buffer` が表示される行で表されるテーブルから一致する行がバッチでフェッチされます。

JSON 形式の出力では、`using_join_buffer` の値は常に `Block Nested Loop`、`Batched Key Access` または `hash join` のいずれかです。

ハッシュ結合は、MySQL 8.0.18 以降で使用できます。Block Nested-Loop アルゴリズムは、MySQL 8.0.20 以降の MySQL リリースでは使用されません。これらの最適化の詳細は、 [セクション8.2.1.4「ハッシュ結合の最適化」](#) および [Block Nested Loop 結合アルゴリズム](#) を参照してください。

バッチキーアクセスアルゴリズムの詳細は、 [Batched Key Access 結合](#) を参照してください。

- [Using MRR \(JSON プロパティ\): `message`](#))

テーブルは Multi-Range Read 最適化戦略を使用して読み取られます。 [セクション8.2.1.11「Multi-Range Read の最適化」](#) を参照してください。

- [Using sort_union\(...\), Using union\(...\), Using intersect\(...\)](#) (JSON プロパティ): `message`)

これらは、`index_merge` 結合タイプのインデックススキャンのマージ方法を示す特定のアルゴリズムを示します。 [セクション8.2.1.3「インデックスマージの最適化」](#) を参照してください。

- [Using temporary \(JSON プロパティ\): `using_temporary_table`](#))

クエリーを解決するために、MySQL は結果を保持する一時テーブルを作成する必要があります。これは一般に、クエリーに、カラムを異なって一覧表示する **GROUP BY** 句と **ORDER BY** 句が含まれる場合に発生します。

- **Using where** (JSON プロパティ): `attached_condition`)

WHERE 句は、次のテーブルに対して照合されるか、またはクライアントに送信される行を制限するために使用されます。具体的にテーブルからすべての行をフェッチするか、調査する意図がないかぎり、**Extra** 値が **Using where** でなく、テーブル結合型が **ALL** または **index** である場合、クエリーに何らかの誤りがある可能性があります。

JSON 形式の出力では、**Using where** に直接対応するものではありません。`attached_condition` プロパティには、使用される **WHERE** 条件が含まれます。

- **Using where with pushed condition** (JSON プロパティ): `message`)

この項目は **NDB** テーブルのみに適用されます。つまり、**NDB Cluster** は条件プッシュダウン最適化を使用して、インデックスなしカラムと定数の間の直接比較の効率を向上させています。そのような場合、条件がクラスタのデータノードに「プッシュダウン」され、すべてのデータノードで同時に評価されます。これにより、一致しない行をネットワーク経由で送る必要がなくなり、コンディションプッシュダウンを使用できるが使用しない場合より、そのようなクエリーを 5 - 10 倍高速化できます。詳細は、[セクション 8.2.1.5 「エンジンコンディションプッシュダウンの最適化」](#) を参照してください。

- **Zero limit** (JSON プロパティ): `message`)

クエリーに **LIMIT 0** 句があり、行を選択できません。

EXPLAIN 出力の解釈

EXPLAIN 出力の **rows** カラムの値の積を取得することで、結合がどの程度適しているかを示す適切な目安を得ることができます。これは、クエリーを実行するために MySQL が調査する必要がある行数を大ざっぱに示すだけです。`max_join_size` システム変数によってクエリーを制限する場合、この行の積は、どの複数テーブル **SELECT** ステートメントを実行し、どれを中止するかを判断するためにも使用されます。[セクション 5.1.1 「サーバーの構成」](#) を参照してください。

次の例は、**EXPLAIN** によって得られた情報に基づいて、複数テーブル結合を段階的に最適化する方法を示しています。

ここに示す **SELECT** ステートメントがあり、**EXPLAIN** を使用して調査するつもりであるとします。

```
EXPLAIN SELECT tt.TicketNumber, tt.TimeIn,
             tt.ProjectReference, tt.EstimatedShipDate,
             tt.ActualShipDate, tt.ClientID,
             tt.ServiceCodes, tt.RepetitiveID,
             tt.CurrentProcess, tt.CurrentDPPerson,
             tt.RecordVolume, tt.DPPrinted, et.COUNTRY,
             et_1.COUNTRY, do.CUSTNAME
FROM tt, et, et AS et_1, do
WHERE tt.SubmitTime IS NULL
      AND tt.ActualPC = et.EMPLOYID
      AND tt.AssignedPC = et_1.EMPLOYID
      AND tt.ClientID = do.CUSTNMBR;
```

この例では次のように想定しています。

- 比較対象のカラムは次のように宣言されています。

Table	カラム	データ型
tt	ActualPC	CHAR(10)
tt	AssignedPC	CHAR(10)
tt	ClientID	CHAR(10)
et	EMPLOYID	CHAR(15)
do	CUSTNMBR	CHAR(15)

- テーブルには次のインデックスがあります。

Table	インデックス
tt	ActualPC
tt	AssignedPC
tt	ClientID
et	EMPLOYID (主キー)
do	CUSTNMBR (主キー)

- tt.ActualPC 値は均一に分布されていません。

最初、最適化が実行される前は、EXPLAIN ステートメントで次の情報が生成されました。

```
table type possible_keys key key_len ref rows Extra
et ALL PRIMARY NULL NULL NULL 74
do ALL PRIMARY NULL NULL NULL 2135
et_1 ALL PRIMARY NULL NULL NULL 74
tt ALL AssignedPC, NULL NULL NULL 3872
    ClientID,
    ActualPC
Range checked for each record (index map: 0x23)
```

各テーブルの type は ALL であるため、この出力は MySQL がすべてのテーブル、つまりすべての行の組み合わせのデカルト積を生成することを示しています。これは、各テーブルの行数の積を調査する必要があるため、著しく時間がかかります。このケースの場合は、この積が $74 \times 2135 \times 74 \times 3872 = 45,268,558,720$ 行になります。テーブルがもっと大きければ、どのくらい時間がかかっていたか簡単に想像がつかます。

ここでの問題の 1 つは、カラムが同じ型とサイズで宣言されている場合に、MySQL はカラムに対してインデックスをより効率的に使用できることです。このコンテキストでは、VARCHAR と CHAR は同じサイズとして宣言されている場合、それらは同じとみなされます。tt.ActualPC は CHAR(10) として宣言されており、et.EMPLOYID は CHAR(15) であるため、長さの不一致があります。

このカラム長の不一致を修正するには、ALTER TABLE を使用して ActualPC を 10 文字から 15 文字に長くします。

```
mysql> ALTER TABLE tt MODIFY ActualPC VARCHAR(15);
```

これで tt.ActualPC と et.EMPLOYID はいずれも VARCHAR(15) になります。EXPLAIN ステートメントを再度実行すると、次の結果が生成されます。

```
table type possible_keys key key_len ref rows Extra
tt ALL AssignedPC, NULL NULL NULL 3872 Using
    ClientID, where
    ActualPC
do ALL PRIMARY NULL NULL NULL 2135
    Range checked for each record (index map: 0x1)
et_1 ALL PRIMARY NULL NULL NULL 74
    Range checked for each record (index map: 0x1)
et eq_ref PRIMARY PRIMARY 15 tt.ActualPC 1
```

これは完全ではありませんが、はるかに改善されています。rows 値の積は 74 の係数分だけ少なくなります。このバージョンは、数秒で実行します。

2 つめの変更を実行して、tt.AssignedPC = et_1.EMPLOYID と tt.ClientID = do.CUSTNMBR の比較でのカラム長の不一致を解消できます。

```
mysql> ALTER TABLE tt MODIFY AssignedPC VARCHAR(15),
    MODIFY ClientID VARCHAR(15);
```

その変更後、EXPLAIN は次に示す出力を生成します。

```
table type possible_keys key key_len ref rows Extra
et ALL PRIMARY NULL NULL NULL 74
tt ref AssignedPC, ActualPC 15 et.EMPLOYID 52 Using
    ClientID, where
    ActualPC
```



```
et_1 eq_ref PRIMARY PRIMARY 15 tt.AssignedPC 1
do eq_ref PRIMARY PRIMARY 15 tt.ClientID 1
```

この時点で、クエリーはほぼ可能なかぎり十分に最適化されています。残りの問題は、MySQL はデフォルトで `tt.ActualPC` カラムの値が均一に分布しているものと想定しますが、`tt` テーブルにはそれが当てはまらないことです。さいわい、MySQL にキー分布を分析するように伝えることは簡単です。

```
mysql> ANALYZE TABLE tt;
```

追加のインデックス情報によって、結合が完全になり、`EXPLAIN` が次の結果を生成します。

```
table type possible_keys key key_len ref rows Extra
tt ALL AssignedPC NULL NULL NULL 3872 Using
ClientID, where
ActualPC
et eq_ref PRIMARY PRIMARY 15 tt.ActualPC 1
et_1 eq_ref PRIMARY PRIMARY 15 tt.AssignedPC 1
do eq_ref PRIMARY PRIMARY 15 tt.ClientID 1
```

`EXPLAIN` からの出力の `rows` カラムは、MySQL 結合オプティマイザからの教育を受けた推測です。`rows` の積とクエリーが返す実際の行数を比較して、数値が実際と近いかどうかをチェックしてください。数値がかなり異なる場合は、`SELECT` ステートメントで `STRAIGHT_JOIN` を使用し、`FROM` 句で異なる順序でテーブルを一覧表示してみるとパフォーマンスを改善できる可能性があります。(ただし、`STRAIGHT_JOIN` では準結合変換が無効になるため、インデックスの使用が妨げられる場合があります。セクション8.2.2.1「準結合変換による `IN` および `EXISTS` サブクエリー述語の最適化」を参照してください。)

場合によっては、サブクエリーで `EXPLAIN SELECT` を使用するとき、データを変更するステートメントを実行できることもあります。詳細については、セクション13.2.11.8「導出テーブル」を参照してください。

8.8.3 拡張 EXPLAIN 出力形式

`EXPLAIN` ステートメントは、`EXPLAIN` 出力の一部ではないが、`EXPLAIN` の後に `SHOW WARNINGS` ステートメントを発行することで表示できる追加(「extended」)情報を生成します。MySQL 8.0.12 では、拡張情報は `SELECT`、`DELETE`、`INSERT`、`REPLACE` および `UPDATE` ステートメントで使用できます。8.0.12 より前のリリースでは、拡張情報は `SELECT` ステートメントでのみ使用できます。

`SHOW WARNINGS` 出力の `Message` 値には、オプティマイザが `SELECT` ステートメント内のテーブルおよびカラム名をどのように修飾するか、書き換えおよび最適化ルール適用後に `SELECT` がどのように見えるか、および場合によって最適化プロセスに関するその他のメモが表示されます。

`EXPLAIN` に続く `SHOW WARNINGS` ステートメントで表示可能な拡張情報は、`SELECT` ステートメントに対してのみ生成されます。`SHOW WARNINGS` では、説明可能な他のステートメント(`DELETE`、`INSERT`、`REPLACE` および `UPDATE`)に対して空の結果が表示されます。

次に、拡張 `EXPLAIN` 出力の例を示します:

```
mysql> EXPLAIN
SELECT t1.a, t1.a IN (SELECT t2.a FROM t2) FROM t1\G
***** 1. row *****
id: 1
select_type: PRIMARY
table: t1
type: index
possible_keys: NULL
key: PRIMARY
key_len: 4
ref: NULL
rows: 4
filtered: 100.00
Extra: Using index
***** 2. row *****
id: 2
select_type: SUBQUERY
table: t2
type: index
possible_keys: a
key: a
key_len: 5
```



```

      ref: NULL
      rows: 3
      filtered: 100.00
      Extra: Using index
2 rows in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Note
Code: 1003
Message: /* select#1 */ select `test`.`t1`.`a` AS `a`,
      <in_optimizer>(`test`.`t1`.`a`,`test`.`t1`.`a` in
      ( <materialize> (/* select#2 */ select `test`.`t2`.`a`
      from `test`.`t2` where 1 having 1 ),
      <primary_index_lookup>(`test`.`t1`.`a` in
      <temporary table> on <auto_key>
      where ((`test`.`t1`.`a` = `materialized-subquery`.`a`)))) AS `t1.a`
      IN (SELECT t2.a FROM t2) from `test`.`t1`
1 row in set (0.00 sec)

```

`SHOW WARNINGS` によって表示されるステートメントには、クエリーの書き換えやオプティマイザのアクションに関する情報を提供する特別なマーカーが含まれることがあるため、ステートメントは必ずしも有効な SQL ではなく、実行されることを目的としていません。出力には、オプティマイザによってとられたアクションに関する追加の SQL でない説明のメモを提供する `Message` 値のある行が含まれることもあります。

次のリストでは、`SHOW WARNINGS` によって表示される拡張出力に表示される特殊なマーカーについて説明します:

- `<auto_key>`

一時テーブルの自動的に生成されるキー。

- `<cache>(expr)`

式 (スカラーサブクエリーなど) が 1 回実行され、あとで使用するために、結果の値がメモリーに保存されます。複数の値で構成される結果の場合、一時テーブルが作成され、かわりに `<temporary table>` が表示されます。

- `<exists>(query fragment)`

サブクエリー述語は `EXISTS` 述語に変換され、サブクエリーは `EXISTS` 述語と一緒に使用できるように変換されます。

- `<in_optimizer>(query fragment)`

これは、ユーザーにとっては意味がない内部オプティマイザオブジェクトです。

- `<index_lookup>(query fragment)`

対象の行を見つけるためにインデックスルックアップを使用して、クエリーフラグメントが処理されます。

- `<if>(condition, expr1, expr2)`

条件が true の場合は `expr1`、そうでない場合は `expr2` に評価されます。

- `<is_not_null_test>(expr)`

式が `NULL` に評価されないことを確認するためのテスト。

- `<materialize>(query fragment)`

サブクエリーの実体化が使用されます。

- ``materialized-subquery`.col_name`

サブクエリーの評価の結果を保持するために実体化された内部一時テーブル内のカラム `col_name` への参照。

- `<primary_index_lookup>(query fragment)`

対象の行を見つけるために主キールックアップを使用して、クエリーフラグメントが処理されます。

- `<ref_null_helper>(expr)`

これは、ユーザーにとっては意味がない内部最適化オブジェクトです。

- `/* select#N */ select_stmt`

`SELECT` は、`id` 値が `N` の拡張されていない `EXPLAIN` 出力の行に関連付けられます。

- `outer_tables semi join (inner_tables)`

準結合操作。`inner_tables` には、プルされなかったテーブルが表示されます。セクション8.2.2.1「準結合変換による `IN` および `EXISTS` サブクエリー述語の最適化」を参照してください。

- `<temporary table>`

これは、中間結果をキャッシュするために作成される内部一時テーブルを表します。

一部のテーブルが `const` または `system` 型である場合、これらのテーブルからのカラムを含む式は、最適化によって早期に評価され、表示されるステートメントに含まれません。ただし、`FORMAT=JSON` では、一部の `const` テーブルアクセスが定数値を使用する `ref` アクセスとして表示されます。

8.8.4 名前付き接続の実行計画情報の取得

名前付き接続で実行されている説明可能なステートメントの実行計画を取得するには、次のステートメントを使用します:

```
EXPLAIN [options] FOR CONNECTION connection_id;
```

`EXPLAIN FOR CONNECTION` は、特定の接続でクエリーを実行するために現在使用されている `EXPLAIN` 情報を返します。データ (およびサポート統計) が変更されたため、同等のクエリーテキストで `EXPLAIN` を実行すると、異なる結果が生成される可能性があります。この動作の違いは、より一時的なパフォーマンスの問題を診断する場合に役立ちます。たとえば、あるセッションで完了までに長時間かかるステートメントを実行している場合、別のセッションで `EXPLAIN FOR CONNECTION` を使用すると、遅延の原因に関する有用な情報が得られることがあります。

`connection_id` は、`INFORMATION_SCHEMA PROCESSLIST` テーブルまたは `SHOW PROCESSLIST` ステートメントから取得される接続識別子です。`PROCESS` 権限がある場合は、任意の接続の識別子を指定できます。それ以外の場合は、独自の接続に対してのみ識別子を指定できます。いずれの場合も、指定した接続に対するクエリーを説明するのに十分な権限が必要です。

名前付き接続がステートメントを実行していない場合、結果は空になります。それ以外の場合、`EXPLAIN FOR CONNECTION` は、名前付き接続で実行されているステートメントが説明可能な場合にのみ適用されます。これには、`SELECT`、`DELETE`、`INSERT`、`REPLACE` および `UPDATE` が含まれます。(ただし、`EXPLAIN FOR CONNECTION` は、これらのタイプのプリペアドステートメントであっても、プリペアドステートメントに対しては機能しません。)

名前付き接続が説明可能なステートメントを実行している場合、出力はステートメント自体で `EXPLAIN` を使用して取得します。

指定された接続が説明不可能なステートメントを実行している場合は、エラーが発生します。たとえば、`EXPLAIN` は説明できないため、現在のセッションの接続識別子に名前を付けることはできません:

```
mysql> SELECT CONNECTION_ID();
+-----+
| CONNECTION_ID() |
+-----+
|          373 |
+-----+
1 row in set (0.00 sec)

mysql> EXPLAIN FOR CONNECTION 373;
ERROR 1889 (HY000): EXPLAIN FOR CONNECTION command is supported
only for SELECT/UPDATE/INSERT/DELETE/REPLACE
```

`Com_explain_other` ステータス変数は、実行された `EXPLAIN FOR CONNECTION` ステートメントの数を示します。

8.8.5 クエリーパフォーマンスの推定

ほとんどの場合、ディスクシークをカウントしてクエリーパフォーマンスを推定できます。小さいテーブルの場合は一般に 1 回のディスクシークでレコードが見つかります (インデックスがキャッシュされている可能性が高いため)。大きなテーブルの場合、B ツリーインデックスを使用して、それを推定できますが、行を見つけるためにこのように多くのシークが必要です。 $\log(\text{row_count}) / \log(\text{index_block_length} / 3 * 2 / (\text{index_length} + \text{data_pointer_length})) + 1$ 。

MySQL では、インデックスブロックが通常 1,024 バイトで、データポインタは通常 4 バイトです。3 バイトのキー値長 (MEDIUMINT のサイズ) の 500,000 行のテーブルの場合、この公式は $\log(500,000) / \log(1024 / 3 * 2 / (3 + 4)) + 1 = 4$ シークを示します。

このインデックスには、約 $500,000 * 7 * 3/2 = 5.2\text{M}$ バイト (2/3 の一般的なインデックスバッファで充てん率と想定して) のストレージが必要であるため、インデックスの多くをメモリーに置く可能性が高く、データを読み取り、行を見つけるために 1 つか 2 つの呼び出しだけで済みます。

ただし、書き込みについては、新しいインデックス値の配置場所を見つけるために 4 つのシークリクエスト、およびインデックスの更新と行の書き込みに通常 2 回のシークが必要になります。

前述の説明は、アプリケーションのパフォーマンスがログ N によってゆっくりと低下することを意味しません。OS または MySQL サーバーによってすべてがキャッシュされているかぎり、テーブルが大きくなってもほんの少し遅くなるだけです。データがキャッシュできないほど大きくなると、アプリケーションがディスクシーク (これは $\log NN$ ずつ増加する) によってのみ制限されるまで著しく遅くなり始めます。これを回避するには、データの増加に合わせてキーキャッシュを増やします。MyISAM テーブルでは、キーキャッシュサイズは `key_buffer_size` システム変数によって制御されます。セクション 5.1.1 「サーバーの構成」を参照してください。

8.9 クエリーオプティマイザの制御

MySQL では、クエリー計画の評価方法、切替え可能な最適化、オプティマイザとインデックスのヒントおよびオプティマイザコストモデルに影響するシステム変数を介してオプティマイザを制御できます。

サーバーは、カラム値に関するヒストグラム統計を `column_statistics` データディクショナリテーブルに保持します (セクション 8.9.6 「オプティマイザ統計」を参照)。他のデータディクショナリテーブルと同様に、このテーブルにはユーザーは直接アクセスできません。かわりに、データディクショナリテーブルのビューとして実装されている `INFORMATION_SCHEMA.COLUMN_STATISTICS` をクエリーすることで、ヒストグラム情報を取得できます。`ANALYZE TABLE` ステートメントを使用してヒストグラム管理を実行することもできます。

8.9.1 クエリー計画評価の制御

クエリーオプティマイザのタスクは SQL クエリーを実行するために最適なプランを見つけることです。「良い」プランと「悪い」プランのパフォーマンスの差は、桁違い (つまり、数秒に対して数時間や数日にまで) になる可能性があるため、MySQL のオプティマイザを含むほとんどのクエリーオプティマイザは、多かれ少なかれ、すべての可能なクエリー評価プランの中から最適なプランを徹底的に探します。結合クエリーに対して、MySQL オプティマイザによって調査される可能なプランの数は、クエリーで参照されるテーブル数とともに指数関数的に増大します。少数のテーブル (一般に 7 から 10 未満) の場合、これは問題になりません。ただし、大きいクエリーが発行されると、クエリーの最適化に費やされる時間がサーバーのパフォーマンスの大きなボトルネックになる可能性があります。

クエリー最適化のより柔軟な方法により、ユーザーはオプティマイザが最適なクエリー評価プランをどの程度徹底的に探すかを制御できます。一般的な考えは、オプティマイザによって調査されるプランが少ないほど、クエリーのコンパイルに費やす時間も少なくなるということです。一方、オプティマイザは一部のプランをスキップするため、最適なプランを見逃す可能性もあります。

評価するプランの数に関して、オプティマイザの動作を 2 つのシステム変数を使用して制御できます。

- `optimizer_prune_level` 変数は、オプティマイザに、テーブルごとにアクセスされる行数の見積もりに基づいて、特定のプランをスキップするように伝えます。経験上、この種類の「学習による推測」は最適なプランをめったに見逃すことはなく、クエリーのコンパイル時間を劇的に短縮できます。デフォルトでこのオプションがオン `optimizer_prune_level=1` であるのはこのためです。ただし、オプティマイザがより適したクエリー計画を見逃したと思う場合は、クエリーのコンパイルにかなりの時間がかかるリスクを伴いますが、このオプションをオフにする (`optimizer_prune_level=0`) ことができます。この経験則を使用しても、オプティマイザはまだ指数関数的な数のプランを探索します。

- `optimizer_search_depth` 変数は、オプティマイザがそれ以上拡張すべきかどうかを評価するために、不完全な各プランの「将来」をどの程度見通すかを伝えます。 `optimizer_search_depth` の値を小さくするほど、クエリーのコンパイル時間が桁違いに少なくなる可能性があります。たとえば、12、13、またはそれ以上のテーブルのクエリーは、 `optimizer_search_depth` がクエリー内のテーブル数に近い場合、コンパイルに数時間または数日間も容易に必要なことがあります。同時に、3 か 4 に等しい `optimizer_search_depth` でコンパイルされた場合、オプティマイザは同じクエリーで 1 分以内にコンパイルできることがあります。 `optimizer_search_depth` の適切な値が不明な場合、この変数を 0 に設定することで、オプティマイザに自動的に値を決定させることができます。

8.9.2 切り替え可能な最適化

`optimizer_switch` システム変数を使用するとオプティマイザの動作を制御できます。その値はフラグのセットで、それぞれ対応するオプティマイザの動作を有効にするかまたは無効にするかを示す `on` または `off` の値を持ちます。この変数はグローバル値およびセッション値を持ち、実行時に変更できます。グローバル値のデフォルトはサーバーの起動時に設定できます。

オプティマイザの現在のフラグセットを表示するには、変数値を選択します。

```
mysql> SELECT @@optimizer_switch\G
***** 1. row *****
@@optimizer_switch: index_merge=on,index_merge_union=on,
                    index_merge_sort_union=on,index_merge_intersection=on,
                    engine_condition_pushdown=on,index_condition_pushdown=on,
                    mrr=on,mrr_cost_based=on,block_nested_loop=on,
                    batched_key_access=off,materialization=on,semijoin=on,
                    loosescan=on,firstmatch=on,duplicateweedout=on,
                    subquery_materialization_cost_based=on,
                    use_index_extensions=on,condition_fanout_filter=on,
                    derived_merge=on,use_invisible_indexes=off,skip_scan=on,
                    hash_join=on,subquery_to_derived=off,
                    prefer_ordering_index=on,hypergraph_optimizer=off,
                    derived_condition_pushdown=on
1 row in set (0.00 sec)
```

`optimizer_switch` の値を変更するには、1 つ以上のコマンドのカンマ区切りのリストから構成される値を割り当てます。

```
SET [GLOBAL|SESSION] optimizer_switch='command[,command]...';
```

各 `command` 値は、次の表に示すいずれかの形式になるようにしてください。

コマンドの構文	意味
<code>default</code>	すべての最適化をそのデフォルト値にリセットします
<code>opt_name=default</code>	指定した最適化をそのデフォルト値に設定します
<code>opt_name=off</code>	指定した最適化を無効にします
<code>opt_name=on</code>	指定した最適化を有効にします

`default` コマンドが存在する場合最初に実行されますが、値の中のコマンドの順序は問題ではありません。 `opt_name` フラグを `default` に設定すると、そのデフォルト値が `on` または `off` のどちらであってもそれに設定されます。値に特定の `opt_name` を複数回指定することは許可されず、エラーが発生します。値のエラーによって、割り当てがエラーを伴って失敗し、 `optimizer_switch` の値が変更されないままになります。

次のリストは、最適化戦略別にグループ化された、許可される `opt_name` フラグ名を示しています：

- バッチキーアクセスフラグ
 - `batched_key_access` (default `off`)

BKA 結合アルゴリズムの使用を制御します

`batched_key_access` が `on` に設定されている場合に何らかの効果を持つためには、 `mrr` フラグも `on` である必要があります。現在、MRR のコスト見積もりはきわめて悲観的です。したがって、BKA を使用するには、 `mrr_cost_based` を `off` にする必要もあります。

詳細は、[セクション8.2.1.12「Block Nested Loop 結合と Batched Key Access 結合」](#)を参照してください。

- ブロックネストループフラグ

- [block_nested_loop](#) (default on)

BNL 結合アルゴリズムの使用を制御します MySQL 8.0.18 以降では、[BNL](#) および [NO_BNL](#) オプティマイザヒントと同様に、ハッシュ結合の使用も制御します。MySQL 8.0.20 以降では、ブロックネストループのサポートは MySQL サーバーから削除され、このフラグはハッシュ結合の使用のみを制御します。

詳細は、[セクション8.2.1.12「Block Nested Loop 結合と Batched Key Access 結合」](#)を参照してください。

- 条件フィルタリングフラグ

- [condition_fanout_filter](#) (default on)

条件フィルタリングの使用を制御します。

詳細は、[セクション8.2.1.13「条件フィルタ」](#)を参照してください。

- 導出条件プッシュダウンフラグ

- [derived_condition_pushdown](#) (default on)

導出条件プッシュダウンを制御します。

詳細は、[セクション8.2.2.5「導出条件プッシュダウン最適化」](#)を参照してください

- 導出テーブルマージフラグ

- [derived_merge](#) (default on)

導出テーブルおよびビューの外部クエリーブロックへのマージを制御します。

[derived_merge](#) フラグは、オプティマイザが導出テーブル、ビュー参照および共通テーブル式を外部クエリーブロックにマージしようとするかどうかを制御します。ただし、他のルールがマージを妨げることはないと想定しています。たとえば、ビューの [ALGORITHM](#) デイレクティブが [derived_merge](#) 設定より優先されます。デフォルトでは、マージを有効にするフラグは [on](#) です。

詳細は、[セクション8.2.2.4「マージまたは実体化を使用した導出テーブル、ビュー参照および共通テーブル式の最適化」](#)を参照してください。

- エンジン条件プッシュダウンフラグ

- [engine_condition_pushdown](#) (default on)

エンジンコンディションプッシュダウンを制御します

詳細は、[セクション8.2.1.5「エンジンコンディションプッシュダウンの最適化」](#)を参照してください。

- ハッシュ結合フラグ

- [hash_join](#) (default on)

ハッシュ結合を制御します (MySQL 8.0.18 のみ。MySQL 8.0.19 以降では影響しません)。

詳細は、[セクション8.2.1.4「ハッシュ結合の最適化」](#)を参照してください。

- インデックス条件プッシュダウンフラグ

- [index_condition_pushdown](#) (default on)

インデックスコンディションプッシュダウンを制御します

詳細は、[セクション8.2.1.6「インデックスコンディションプッシュダウンの最適化」](#)を参照してください。

- インデックス拡張フラグ
 - `use_index_extensions` (default on)
インデックス拡張の使用を制御します
詳細は、[セクション8.3.10「インデックス拡張の使用」](#)を参照してください。
- インデックスマージフラグ
 - `index_merge` (default on)
すべてのインデックスマージ最適化を制御します
 - `index_merge_intersection` (default on)
インデックスマージ共通集合アクセス最適化を制御します
 - `index_merge_sort_union` (default on)
インデックスマージソート集合アクセス最適化を制御します
 - `index_merge_union` (default on)
インデックスマージ和集合アクセス最適化を制御します
詳細については、[セクション8.2.1.3「インデックスマージの最適化」](#)を参照してください。
- インデックス可視性フラグ
 - `use_invisible_indexes` (default off)
不可視インデックスの使用を制御します。
詳細は、[セクション8.3.12「不可視のインデックス」](#)を参照してください。
- 制限最適化フラグ
 - `prefer_ordering_index` (default on)
`LIMIT` 句を含む `ORDER BY` または `GROUP BY` を持つクエリーの場合に、オプティマイザが順序付けされていないインデックス、`filesort` またはその他の最適化のかわりに順序付けられたインデックスを使用しようとするかどうかを制御します。この最適化は、オプティマイザがこの最適化を使用するとクエリーの実行速度が速くなると判断するたびに、デフォルトで実行されます。

この決定を行うアルゴリズムではすべての同義のケースを処理できないため(データの分散が常に均一であるか、それほど均一でないことが前提となります)、この最適化が望ましくない場合があります。MySQL 8.0.21 より前は、この最適化を無効にすることはできませんが、MySQL 8.0.21 以降ではデフォルトの動作のままですが、`prefer_ordering_index` フラグを `off` に設定することで無効にできます。

詳細および例については、[セクション8.2.1.19「LIMIT クエリーの最適化」](#)を参照してください。
- 複数範囲検針フラグ
 - `mrr` (default on)
Multi-Range Read 戦略を制御します
 - `mrr_cost_based` (default on)
`mrr=on` の場合にコストベースの MRR の使用を制御します
詳細は、[セクション8.2.1.11「Multi-Range Read の最適化」](#)を参照してください。
- 準結合フラグ

- [duplicateweedout](#) (default on)
準結合重複除去ストラテジを制御します。
- [firstmatch](#) (default on)
準結合 FirstMatch 戦略を制御します。
- [loosescan](#) (default on)
準結合 LooseScan 戦略を制御します (Loose Index Scan for [GROUP BY](#) と混同しないでください)。
- [semijoin](#) (default on)
すべての準結合方針を制御します。
MySQL 8.0.17 以降では、これはアンチ結合の最適化にも適用されます。

[semijoin](#), [firstmatch](#), [loosescan](#) および [duplicateweedout](#) フラグを使用すると、準結合戦略を制御できます。[semijoin](#) フラグは、準結合を使用するかどうかを制御します。on に設定されている場合、[firstmatch](#) および [loosescan](#) フラグを使用すると、許可された準結合戦略をより細かく制御できます。

[duplicateweedout](#) 準結合戦略が無効になっている場合は、他のすべての適用可能な戦略も無効にしないかぎり、使用されません。

[semijoin](#) と [materialization](#) の両方が on の場合、準結合でも実体化が使用されます (該当する場合)。これらのフラグはデフォルトで on です。

詳細は、[セクション8.2.2.1「準結合変換による IN および EXISTS サブクエリー述語の最適化」](#)を参照してください。

- スキャンフラグのスキップ
 - [skip_scan](#) (default on)
スキップスキャンアクセス方法の使用を制御します。
詳細は、[スキャン範囲アクセス方法のスキップ](#)を参照してください。
- サブクエリー実体化フラグ
 - [materialization](#) (default on)
実体化 (準結合実体化を含む) を制御します。
 - [subquery_materialization_cost_based](#) (default on)
原価ベースの実体化の選択を使用します。

[materialization](#) フラグはサブクエリー実体化を使用するかどうかを制御します。[semijoin](#) と [materialization](#) の両方が on の場合、準結合でも実体化が使用されます (該当する場合)。これらのフラグはデフォルトで on です。

[subquery_materialization_cost_based](#) フラグを使用すると、サブクエリー実体化と IN から EXISTS へのサブクエリー変換の選択を制御できます。フラグが on (デフォルト) の場合、オプティマイザはサブクエリー実体化と IN から -EXISTS サブクエリー変換 (いずれかの方法を使用できる場合) の間でコストベースの選択を実行します。フラグが off の場合、オプティマイザは IN から EXISTS へのサブクエリー変換よりもサブクエリーの実体化を選択します。

詳細は、[セクション8.2.2「サブクエリー、導出テーブル、ビュー参照および共通テーブル式の最適化」](#)を参照してください。

- サブクエリー変換フラグ
 - [subquery_to_derived](#) (default off)

MySQL 8.0.21 以降、オプティマイザは多くの場合、`SELECT`、`WHERE`、`JOIN` または `HAVING` 句のスカラーサブクエリーを導出テーブルの左外部結合に変換できます。(導出テーブルの `NULL` 値可能性によっては、内部結合にさらに簡略化される場合があります。)これは、次の条件を満たすサブクエリーに対して実行できます:

- サブクエリーでは、`RAND()` などの非決定的関数は使用されません。
- サブクエリーは、`MIN()` または `MAX()` を使用するようリライトできる `ANY` または `ALL` サブクエリーではありません。
- 親クエリーはユーザー変数を設定しません。リライトすると実行順序に影響する可能性があるため、同じクエリーで変数に複数回アクセスすると、予期しない結果が発生する可能性があります。
- サブクエリーは相関付けしないでください。つまり、外部クエリーのテーブルからカラムを参照したり、外部クエリーで評価される集計を含むことはできません。

MySQL 8.0.22 より前は、サブクエリーに `GROUP BY` 句を含めることはできませんでした。

この最適化は、`GROUP BY` を含まない `IN`、`NOT IN`、`EXISTS` または `NOT EXISTS` の引数であるテーブルサブクエリーにも適用できます。

このフラグのデフォルト値は `off` ですが、ほとんどの場合、この最適化を有効にしてもパフォーマンスが著しく向上することはありません(多くの場合、クエリーの実行速度が遅くなることもあります)。ただし、`subquery_to_derived` フラグを `on` に設定することで最適化を有効にできます。主にテストで使用することを目的としています。

スカラーサブクエリーを使用する例:

```
d
mysql> CREATE TABLE t1(a INT);

mysql> CREATE TABLE t2(a INT);

mysql> INSERT INTO t1 VALUES ROW(1), ROW(2), ROW(3), ROW(4);

mysql> INSERT INTO t2 VALUES ROW(1), ROW(2);

mysql> SELECT * FROM t1
-> WHERE t1.a > (SELECT COUNT(a) FROM t2);
+----+
| a  |
+----+
| 3  |
| 4  |
+----+

mysql> SELECT @@optimizer_switch LIKE '%subquery_to_derived=off%';
+-----+
| @@optimizer_switch LIKE '%subquery_to_derived=off%' |
+-----+
| 1 |
+-----+

mysql> EXPLAIN SELECT * FROM t1 WHERE t1.a > (SELECT COUNT(a) FROM t2)G
***** 1. row *****
      id: 1
  select_type: PRIMARY
    table: t1
  partitions: NULL
     type: ALL
possible_keys: NULL
      key: NULL
    key_len: NULL
       ref: NULL
      rows: 4
  filtered: 33.33
  Extra: Using where
***** 2. row *****
      id: 2
  select_type: SUBQUERY
```

```

table: t2
partitions: NULL
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 2
filtered: 100.00
Extra: NULL

mysql> SET @@optimizer_switch='subquery_to_derived=on';

mysql> SELECT @@optimizer_switch LIKE '%subquery_to_derived=off%';
+-----+
| @@optimizer_switch LIKE '%subquery_to_derived=off%' |
+-----+
|                                0 |
+-----+

mysql> SELECT @@optimizer_switch LIKE '%subquery_to_derived=on%';
+-----+
| @@optimizer_switch LIKE '%subquery_to_derived=on%' |
+-----+
|                                1 |
+-----+

mysql> EXPLAIN SELECT * FROM t1 WHERE t1.a > (SELECT COUNT(a) FROM t2)\G
***** 1. row *****
id: 1
select_type: PRIMARY
table: <derived2>
partitions: NULL
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 1
filtered: 100.00
Extra: NULL
***** 2. row *****
id: 1
select_type: PRIMARY
table: t1
partitions: NULL
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 4
filtered: 33.33
Extra: Using where; Using join buffer (hash join)
***** 3. row *****
id: 2
select_type: DERIVED
table: t2
partitions: NULL
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 2
filtered: 100.00
Extra: NULL

```

2 番目の EXPLAIN ステートメントの直後に SHOW WARNINGS を実行するとわかるように、最適化を有効にすると、クエリー `SELECT * FROM t1 WHERE t1.a > (SELECT COUNT(a) FROM t2)` は次に示すような形式でリライトされます:

```
SELECT t1.a FROM t1
```

```
JOIN ( SELECT COUNT(t2.a) AS c FROM t2 ) AS d
WHERE t1.a > d.c;
```

IN (subquery) でクエリーを使用する例:

```
mysql> DROP TABLE IF EXISTS t1, t2;

mysql> CREATE TABLE t1 (a INT, b INT);
mysql> CREATE TABLE t2 (a INT, b INT);

mysql> INSERT INTO t1 VALUES ROW(1,10), ROW(2,20), ROW(3,30);
mysql> INSERT INTO t2
-> VALUES ROW(1,10), ROW(2,20), ROW(3,30), ROW(1,110), ROW(2,120), ROW(3,130);

mysql> SELECT * FROM t1
-> WHERE t1.b < 0
-> OR
-> t1.a IN (SELECT t2.a + 1 FROM t2);
+----+-----+
| a  | b   |
+----+-----+
| 2  | 20  |
| 3  | 30  |
+----+-----+

mysql> SET @@optimizer_switch="subquery_to_derived=off";

mysql> EXPLAIN SELECT * FROM t1
-> WHERE t1.b < 0
-> OR
-> t1.a IN (SELECT t2.a + 1 FROM t2)\G
***** 1. row *****
      id: 1
  select_type: PRIMARY
      table: t1
  partitions: NULL
      type: ALL
possible_keys: NULL
      key: NULL
      key_len: NULL
      ref: NULL
      rows: 3
  filtered: 100.00
  Extra: Using where
***** 2. row *****
      id: 2
  select_type: DEPENDENT SUBQUERY
      table: t2
  partitions: NULL
      type: ALL
possible_keys: NULL
      key: NULL
      key_len: NULL
      ref: NULL
      rows: 6
  filtered: 100.00
  Extra: Using where

mysql> SET @@optimizer_switch="subquery_to_derived=on";

mysql> EXPLAIN SELECT * FROM t1
-> WHERE t1.b < 0
-> OR
-> t1.a IN (SELECT t2.a + 1 FROM t2)\G
***** 1. row *****
      id: 1
  select_type: PRIMARY
      table: t1
  partitions: NULL
      type: ALL
possible_keys: NULL
      key: NULL
      key_len: NULL
      ref: NULL
```

```

rows: 3
filtered: 100.00
Extra: NULL
***** 2. row *****
id: 1
select_type: PRIMARY
table: <derived2>
partitions: NULL
type: ref
possible_keys: <auto_key0>
key: <auto_key0>
key_len: 9
ref: std2.t1.a
rows: 2
filtered: 100.00
Extra: Using where; Using index
***** 3. row *****
id: 2
select_type: DERIVED
table: t2
partitions: NULL
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 6
filtered: 100.00
Extra: Using temporary

```

このクエリーで `EXPLAIN` を実行した後の `SHOW WARNINGS` の結果のチェックおよび簡略化は、`subquery_to_derived` フラグが有効になっている場合、`SELECT * FROM t1 WHERE t1.b < 0 OR t1.a IN (SELECT t2.a + 1 FROM t2)` が次に示すような形式でリライトされることを示しています:

```

SELECT a, b FROM t1
LEFT JOIN (SELECT DISTINCT a + 1 AS e FROM t2) d
ON t1.a = d.e
WHERE t1.b < 0
OR
d.e IS NOT NULL;

```

例:`EXISTS (subquery)` と前述の例と同じテーブルおよびデータを使用したクエリーを使用します:

```

mysql> SELECT * FROM t1
-> WHERE t1.b < 0
-> OR
-> EXISTS(SELECT * FROM t2 WHERE t2.a = t1.a + 1);
+----+-----+
| a | b |
+----+-----+
| 1 | 10 |
| 2 | 20 |
+----+-----+

mysql> SET @@optimizer_switch="subquery_to_derived=off";

mysql> EXPLAIN SELECT * FROM t1
-> WHERE t1.b < 0
-> OR
-> EXISTS(SELECT * FROM t2 WHERE t2.a = t1.a + 1)\G
***** 1. row *****
id: 1
select_type: PRIMARY
table: t1
partitions: NULL
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 3
filtered: 100.00
Extra: Using where

```

```

***** 2. row *****
  id: 2
  select_type: DEPENDENT SUBQUERY
  table: t2
  partitions: NULL
  type: ALL
  possible_keys: NULL
  key: NULL
  key_len: NULL
  ref: NULL
  rows: 6
  filtered: 16.67
  Extra: Using where

mysql> SET @@optimizer_switch="subquery_to_derived=on";

mysql> EXPLAIN SELECT * FROM t1
->      WHERE t1.b < 0
->      OR
->      EXISTS(SELECT * FROM t2 WHERE t2.a = t1.a + 1)G
***** 1. row *****
  id: 1
  select_type: PRIMARY
  table: t1
  partitions: NULL
  type: ALL
  possible_keys: NULL
  key: NULL
  key_len: NULL
  ref: NULL
  rows: 3
  filtered: 100.00
  Extra: NULL
***** 2. row *****
  id: 1
  select_type: PRIMARY
  table: <derived2>
  partitions: NULL
  type: ALL
  possible_keys: NULL
  key: NULL
  key_len: NULL
  ref: NULL
  rows: 6
  filtered: 100.00
  Extra: Using where; Using join buffer (hash join)
***** 3. row *****
  id: 2
  select_type: DERIVED
  table: t2
  partitions: NULL
  type: ALL
  possible_keys: NULL
  key: NULL
  key_len: NULL
  ref: NULL
  rows: 6
  filtered: 100.00
  Extra: Using temporary

```

`subquery_to_derived` が有効になっているときに、クエリー `SELECT * FROM t1 WHERE t1.b < 0 OR EXISTS(SELECT * FROM t2 WHERE t2.a = t1.a + 1)` で `EXPLAIN` を実行した後に `SHOW WARNINGS` を実行し、結果の 2 行目を簡略化すると、次のような形式でリライトされていることがわかります:

```

SELECT a, b FROM t1
LEFT JOIN (SELECT DISTINCT 1 AS e1, t2.a AS e2 FROM t2) d
ON t1.a + 1 = d.e2
WHERE t1.b < 0
OR

```



```
d.e1 IS NOT NULL;
```

詳細は、[セクション8.2.2.4「マージまたは実体化を使用した導出テーブル、ビュー参照および共通テーブル式の最適化」](#)、[セクション8.2.1.19「LIMIT クエリーの最適化」](#) および [セクション8.2.2.1「準結合変換による IN および EXISTS サブクエリー述語の最適化」](#) を参照してください。

`optimizer_switch` に値を割り当てると、指定されていないフラグはそれらの現在の値を維持します。これにより、ほかの動作に影響を与えることなく、単一のステートメントで特定のオプティマイザの動作を有効または無効にできます。ステートメントは、ほかの存在するオプティマイザフラグやそれらの値に依存しません。すべてのインデックスマージ最適化が有効になっているとします。

```
mysql> SELECT @@optimizer_switch\G
***** 1. row *****
@@optimizer_switch: index_merge=on,index_merge_union=on,
                    index_merge_sort_union=on,index_merge_intersection=on,
                    engine_condition_pushdown=on,index_condition_pushdown=on,
                    mrr=on,mrr_cost_based=on,block_nested_loop=on,
                    batched_key_access=off,materialization=on,semijoin=on,
                    loosescan=on, firstmatch=on,
                    subquery_materialization_cost_based=on,
                    use_index_extensions=on,condition_fanout_filter=on,
                    derived_merge=on,use_invisible_indexes=off,skip_scan=on,
                    hash_join=on,subquery_to_derived=off,
                    prefer_ordering_index=on
```

サーバーが特定のクエリーに対して Index Merge Union または Index Merge Sort-Union アクセス方法を使用しており、オプティマイザがそれらなしでパフォーマンスを向上できるかどうかを確認する場合は、次のように変数値を設定します:

```
mysql> SET optimizer_switch='index_merge_union=off,index_merge_sort_union=off';
```

```
mysql> SELECT @@optimizer_switch\G
***** 1. row *****
@@optimizer_switch: index_merge=on,index_merge_union=off,
                    index_merge_sort_union=off,index_merge_intersection=on,
                    engine_condition_pushdown=on,index_condition_pushdown=on,
                    mrr=on,mrr_cost_based=on,block_nested_loop=on,
                    batched_key_access=off,materialization=on,semijoin=on,
                    loosescan=on, firstmatch=on,
                    subquery_materialization_cost_based=on,
                    use_index_extensions=on,condition_fanout_filter=on,
                    derived_merge=on,use_invisible_indexes=off,skip_scan=on,
                    hash_join=on,subquery_to_derived=off,
                    prefer_ordering_index=on
```

8.9.3 オプティマイザヒント

オプティマイザ戦略を制御する方法の1つは、`optimizer_switch` システム変数を設定することです ([セクション8.9.2「切り替え可能な最適化」](#) を参照)。この変数を変更すると、後続のすべてのクエリーの実行に影響します。あるクエリーと別のクエリーに影響を与えるには、各クエリーの前に `optimizer_switch` を変更する必要があります。

オプティマイザを制御する別の方法は、オプティマイザヒントを使用することです。オプティマイザヒントは、個々のステートメント内で指定できます。オプティマイザヒントはステートメントごとに適用されるため、`optimizer_switch` を使用して達成できるよりも詳細にステートメントの実行計画を制御できます。たとえば、ステートメントのあるテーブルに対して最適化を有効にし、別のテーブルに対して最適化を無効にできます。ステートメント内のヒントは、`optimizer_switch` フラグよりも優先されます。

例:

```
SELECT /*+ NO_RANGE_OPTIMIZATION(t3 PRIMARY, f2_idx) */ f1
  FROM t3 WHERE f1 > 30 AND f1 < 33;
SELECT /*+ BKA(t1) NO_BKA(t2) */ * FROM t1 INNER JOIN t2 WHERE ...;
SELECT /*+ NO_ICP(t1, t2) */ * FROM t1 INNER JOIN t2 WHERE ...;
SELECT /*+ SEMIJOIN(FIRSTMATCH, LOOESCAN) */ * FROM t1 ...;
EXPLAIN SELECT /*+ NO_ICP(t1) */ * FROM t1 WHERE ...;
SELECT /*+ MERGE(dt) */ * FROM (SELECT * FROM t1) AS dt;
INSERT /*+ SET_VAR(foreign_key_checks=OFF) */ INTO t2 VALUES(2);
```

ここで説明するオプティマイザヒントは、[セクション8.9.4「インデックスヒント」](#)で説明されているインデックスヒントとは異なります。オプティマイザヒントとインデックスヒントは、別々に、または一緒に使用できます。

- [オプティマイザヒントの概要](#)
- [オプティマイザヒントの構文](#)
- [結合順序オプティマイザヒント](#)
- [テーブルレベルのオプティマイザヒント](#)
- [インデックスレベルのオプティマイザヒント](#)
- [サブクエリーオプティマイザヒント](#)
- [ステートメント実行時オプティマイザヒント](#)
- [可変設定のヒント構文](#)
- [リソースグループのヒント構文](#)
- [クエリーブロックのネーミングのためのオプティマイザヒント](#)

オプティマイザヒントの概要

オプティマイザヒントは、様々な有効範囲レベルで適用されます:

- グローバル: ヒントはステートメント全体に影響
- クエリーブロック: ヒントは、ステートメント内の特定のクエリーブロックに影響
- Table-level: ヒントは、クエリーブロック内の特定のテーブルに影響
- Index-level: ヒントは、テーブル内の特定のインデックスに影響

次のテーブルに、使用可能なオプティマイザヒント、それらが影響するオプティマイザ戦略、およびそれらが適用される有効範囲をまとめます。詳細は後で説明します。

表 8.2 使用可能なオプティマイザヒント

ヒント名	説明	適用可能なスコープ
BKA, NO_BKA	バッチキーアクセス結合処理に影響	クエリーブロック、テーブル
BNL, NO_BNL	MySQL 8.0.20 より前: Block Nested-Loop 結合処理 (MySQL 8.0.18 以降) に影響: ハッシュ結合の最適化にも影響します (MySQL 8.0.20 以降): ハッシュ結合の最適化にのみ影響	クエリーブロック、テーブル
DERIVED_CONDITION_PUSHDOWN, NO_DERIVED_CONDITION_PUSHDOWN	実体化導出テーブルに対する導出条件ハッシュダウ最適化の使用または無視 (MySQL 8.0.22 で追加)	クエリーブロック、テーブル
GROUP_INDEX, NO_GROUP_INDEX	GROUP BY 操作でのインデックススキャンのための指定したインデックスの使用または無視 (MySQL 8.0.20 で追加)	インデックス
HASH_JOIN, NO_HASH_JOIN	ハッシュ結合の最適化に影響 (MySQL 8.0.18 のみ)	クエリーブロック、テーブル
INDEX, NO_INDEX	JOIN_INDEX 、 GROUP_INDEX および ORDER_INDEX の組合せとして、または NO_JOIN_INDEX 、 NO_GROUP_INDEX	インデックス

ヒント名	説明	適用可能なスコープ
	および <code>NO_ORDER_INDEX</code> (MySQL 8.0.20 に追加) の組合せとして機能	
<code>INDEX_MERGE, NO_INDEX_MERGE</code>	インデックスマージの最適化に影響	Table、インデックス
<code>JOIN_FIXED_ORDER</code>	結合順序に <code>FROM</code> 句で指定されたテーブルの順序を使用	クエリーブロック
<code>JOIN_INDEX, NO_JOIN_INDEX</code>	任意のアクセス方法に指定されたインデックスを使用または無視します (MySQL 8.0.20 で追加)	インデックス
<code>JOIN_ORDER</code>	結合順序のヒントに指定されたテーブルの順序を使用	クエリーブロック
<code>JOIN_PREFIX</code>	結合順序の最初のテーブルにヒントで指定されたテーブル順序を使用	クエリーブロック
<code>JOIN_SUFFIX</code>	結合順序の最後のテーブルにヒントで指定されたテーブル順序を使用	クエリーブロック
<code>MAX_EXECUTION_TIME</code>	ステートメントの実行時間の制限	グローバル
<code>MERGE, NO_MERGE</code>	外部クエリーブロックへの導出テーブル/ビューのマージに影響	Table
<code>MRR, NO_MRR</code>	マルチレンジ読取り最適化に影響	Table、インデックス
<code>NO_ICP</code>	インデックス条件プッシュダウンの最適化に影響	Table、インデックス
<code>NO_RANGE_OPTIMIZATION</code>	範囲の最適化に影響	Table、インデックス
<code>ORDER_INDEX, NO_ORDER_INDEX</code>	指定したインデックスを使用または無視して行をソートします (MySQL 8.0.20 で追加)	インデックス
<code>QB_NAME</code>	クエリーブロックに名前を割り当てます	クエリーブロック
<code>RESOURCE_GROUP</code>	ステートメントの実行中にリソースグループを設定	グローバル
<code>SEMIJOIN, NO_SEMIJOIN</code>	準結合戦略に影響します。MySQL 8.0.17 以降、これはアンチ結合にも適用されます	クエリーブロック
<code>SKIP_SCAN, NO_SKIP_SCAN</code>	スキップスキャンの最適化に影響	Table、インデックス
<code>SET_VAR</code>	ステートメントの実行中に変数を設定	グローバル
<code>SUBQUERY</code>	実体化、 <code>IN-</code> から <code>-EXISTS</code> サブクエリー戦略に影響	クエリーブロック

最適化を無効にすると、オプティマイザで使用できなくなります。最適化を有効にすることは、オプティマイザがステートメントの実行に適用する場合に、オプティマイザが戦略を自由に使用できることを意味します。オプティマイザが使用する必要はありません。

オプティマイザヒントの構文

MySQL では、[セクション9.7「コメント」](#)で説明されているように、SQL ステートメントのコメントがサポートされます。オプティマイザヒントは、`/*+ ... */`コメント内で指定する必要があります。つまり、オプティマイザヒントでは、`/*`コメントのオープン順序の後に `+ 文字`が付いた `/* ... */` C 形式のコメント構文のバリエーションが使用されます。例:

```
/*+ BKA(t1) */
/*+ BNL(t1, t2) */
/*+ NO_RANGE_OPTIMIZATION(t4 PRIMARY) */
/*+ QB_NAME(qb2) */
```

+ 文字の後には空白を使用できます。

パーサーは、[SELECT](#)、[UPDATE](#)、[INSERT](#)、[REPLACE](#) ステートメントおよび [DELETE](#) ステートメントの最初のキーワードの後にオプティマイザヒントコメントを認識します。ヒントは、次のコンテキストで使用できます:

- クエリーステートメントおよびデータ変更ステートメントの開始時:

```
SELECT /*+ ... */ ...
INSERT /*+ ... */ ...
REPLACE /*+ ... */ ...
UPDATE /*+ ... */ ...
DELETE /*+ ... */ ...
```

- クエリーブロックの先頭:

```
(SELECT /*+ ... */ ...)
(SELECT ...) UNION (SELECT /*+ ... */ ...)
(SELECT /*+ ... */ ...) UNION (SELECT /*+ ... */ ...)
UPDATE ... WHERE x IN (SELECT /*+ ... */ ...)
INSERT ... SELECT /*+ ... */ ...
```

- [EXPLAIN](#) で始まるヒント可能ステートメント。例:

```
EXPLAIN SELECT /*+ ... */ ...
EXPLAIN UPDATE ... WHERE x IN (SELECT /*+ ... */ ...)
```

これは、[EXPLAIN](#) を使用してオプティマイザヒントが実行計画に与える影響を確認できることを意味します。[EXPLAIN](#) の直後に [SHOW WARNINGS](#) を使用して、ヒントの使用方法を確認します。次の [SHOW WARNINGS](#) によって表示される拡張 [EXPLAIN](#) 出力は、使用されたヒントを示します。無視されたヒントは表示されません。

ヒントコメントには複数のヒントを含めることができますが、クエリーブロックに複数のヒントコメントを含めることはできません。これは有効です:

```
SELECT /*+ BNL(t1) BKA(t2) */ ...
```

ただし、これは無効です:

```
SELECT /*+ BNL(t1) */ /* BKA(t2) */ ...
```

ヒントコメントに複数のヒントが含まれている場合、重複および競合の可能性があります。次の一般的なガイドラインが適用されます。特定のヒントタイプについては、ヒントの説明に示されているように、追加のルールが適用される場合があります。

- ヒントの複製: `/*+ MRR(idx1) MRR(idx1) */`などのヒントの場合、MySQL では最初のヒントが使用され、重複ヒントに関する警告が発行されます。
- 競合するヒント: `/*+ MRR(idx1) NO_MRR(idx1) */`などのヒントの場合、MySQL は最初のヒントを使用し、競合する2つ目のヒントに関する警告を発行します。

クエリーブロック名は識別子であり、有効な名前とその引用符の方法に関する通常のルールに従います ([セクション 9.2 「スキーマオブジェクト名」](#) を参照)。

ヒント名、クエリーブロック名および方針名では、大文字と小文字は区別されません。テーブル名およびインデックス名への参照は、通常の識別子の大小文字区別ルールに従います ([セクション 9.2.3 「識別子の大小文字の区別」](#) を参照)。

結合順序オプティマイザヒント

結合順序ヒントは、オプティマイザがテーブルを結合する順序に影響します。

[JOIN_FIXED_ORDER](#) ヒントの構文:

```
hint_name(@query_block_name)
```

その他の結合順序ヒントの構文は、次のとおりです:

```
hint_name(@query_block_name tbl_name [, tbl_name] ...)
hint_name(tbl_name@query_block_name [, tbl_name@query_block_name] ...)
```

構文は、次の用語を指します:

- **hint_name**: 次のヒント名を使用できます:
 - **JOIN_FIXED_ORDER**: オプティマイザが、**FROM** 句に出現する順序を使用してテーブルを結合するように強制します。これは、**SELECT STRAIGHT_JOIN** の指定と同じです。
 - **JOIN_ORDER**: 指定されたテーブルの順序を使用してテーブルを結合するようオプティマイザに指示します。ヒントは、指定したテーブルに適用されます。オプティマイザは、指定されたテーブルの間を含め、結合順序のどこにも名前が付いていないテーブルを配置できます。
 - **JOIN_PREFIX**: 結合実行計画の最初のテーブルに指定されたテーブル順序を使用してテーブルを結合するようオプティマイザに指示します。ヒントは、指定したテーブルに適用されます。オプティマイザは、他のすべてのテーブルを指定されたテーブルの後に配置します。
 - **JOIN_SUFFIX**: 結合実行計画の最後のテーブルに対して指定されたテーブルの順序を使用してテーブルを結合するようオプティマイザに指示します。ヒントは、指定したテーブルに適用されます。オプティマイザは、他のすべてのテーブルを名前付きテーブルの前に配置します。
- **tbl_name**: ステートメントで使用されるテーブルの名前。テーブルに名前を付けるヒントは、名前を付けるすべてのテーブルに適用されます。**JOIN_FIXED_ORDER** ヒントでは、テーブルに名前が付けられず、クエリーブロックの **FROM** 句のすべてのテーブルに適用されます。

テーブルにエイリアスがある場合、ヒントはテーブル名ではなくエイリアスを参照する必要があります。

ヒントのテーブル名はスキーマ名で修飾できません。

- **query_block_name**: ヒントが適用されるクエリーブロック。ヒントに先行する **@query_block_name** が含まれていない場合、ヒントは発生したクエリーブロックに適用されます。**tbl_name@query_block_name** 構文の場合、ヒントは名前付きクエリーブロック内の名前付きテーブルに適用されます。クエリーブロックに名前を割り当てるには、**クエリーブロックのネーミングのためのオプティマイザヒント** を参照してください。

例:

```
SELECT
/*+ JOIN_PREFIX(t2, t5@subq2, t4@subq1)
   JOIN_ORDER(t4@subq1, t3)
   JOIN_SUFFIX(t1) */
COUNT(*) FROM t1 JOIN t2 JOIN t3
WHERE t1.f1 IN (SELECT /*+ QB_NAME(subq1) */ f1 FROM t4)
AND t2.f1 IN (SELECT /*+ QB_NAME(subq2) */ f1 FROM t5);
```

ヒントは、外部クエリーブロックにマージされる準結合テーブルの動作を制御します。サブクエリー **subq1** および **subq2** が準結合に変換されると、テーブル **t4@subq1** および **t5@subq2** が外部クエリーブロックにマージされます。この場合、外部クエリーブロックで指定されたヒントによって、**t4@subq1**、**t5@subq2** テーブルの動作が制御されます。

オプティマイザは、次の原則に従って結合順序のヒントを解決します:

- 複数のヒントインスタンス

各タイプの **JOIN_PREFIX** および **JOIN_SUFFIX** ヒントのみが適用されます。同じタイプの後のヒントは無視され、警告が表示されます。**JOIN_ORDER** は複数回指定できます。

例:

```
/*+ JOIN_PREFIX(t1) JOIN_PREFIX(t2) */
```

2 番目の **JOIN_PREFIX** ヒントは無視され、警告が表示されます。

```
/*+ JOIN_PREFIX(t1) JOIN_SUFFIX(t2) */
```

両方のヒントが適用されます。警告は発生しません。

```
/*+ JOIN_ORDER(t1, t2) JOIN_ORDER(t2, t3) */
```

両方のヒントが適用されます。警告は発生しません。

- 競合するヒント

`JOIN_ORDER` と `JOIN_PREFIX` に同時に適用できないテーブルの順序がある場合など、ヒントが競合することがあります:

```
SELECT /*+ JOIN_ORDER(t1, t2) JOIN_PREFIX(t2, t1) */ ... FROM t1, t2;
```

この場合、最初に指定したヒントが適用され、後続の競合するヒントは警告なしで無視されます。適用できない有効なヒントは、警告なしで暗黙的に無視されます。

- 無視されたヒント

ヒントで指定されたテーブルに循環依存性がある場合、ヒントは無視されます。

例:

```
/*+ JOIN_ORDER(t1, t2) JOIN_PREFIX(t2, t1) */
```

`JOIN_ORDER` ヒントは、`t1` に依存するテーブル `t2` を設定します。テーブル `t1` は `t2` に依存できないため、`JOIN_PREFIX` ヒントは無視されます。無視されたヒントは、拡張 `EXPLAIN` 出力には表示されません。

- `const` テーブルとの相互作用

MySQL オプティマイザは、`const` テーブルを結合順序の最初に配置し、`const` テーブルの位置はヒントの影響を受けません。結合順序ヒント内の `const` テーブルへの参照は無視されますが、ヒントは引き続き適用可能です。たとえば、これらは同等です:

```
JOIN_ORDER(t1, const_tbl, t2)  
JOIN_ORDER(t1, t2)
```

拡張 `EXPLAIN` 出力に表示される許容ヒントには、指定されたとおりに `const` テーブルが含まれます。

- 結合操作のタイプとの相互作用

MySQL では、複数のタイプの結合がサポートされています: `LEFT`, `RIGHT`, `INNER`, `CROSS`, `STRAIGHT_JOIN`。指定したタイプの結合と競合するヒントは、警告なしで無視されます。

例:

```
SELECT /*+ JOIN_PREFIX(t1, t2) */FROM t2 LEFT JOIN t1;
```

ここでは、ヒント内のリクエストされた結合順序と `LEFT JOIN` に必要な順序の間で競合が発生します。ヒントは警告なしで無視されます。

テーブルレベルのオプティマイザヒント

テーブルレベルのヒントは次のものに影響します:

- Block Nested-Loop (BNL) および Batched Key Access (BKA) 結合処理アルゴリズムの使用 ([セクション 8.2.1.12 「Block Nested Loop 結合と Batched Key Access 結合」](#) を参照)。
- 導出テーブル、ビュー参照または共通テーブル式を外部クエリーブロックにマージするか、内部一時テーブルを使用して実体化するか。
- 導出テーブル条件プッシュダウン最適化の使用 (MySQL 8.0.22 で追加)。 [セクション 8.2.2.5 「導出条件プッシュダウン最適化」](#) を参照してください。

これらのヒントタイプは、特定のテーブルまたはクエリーブロック内のすべてのテーブルに適用されます。

テーブルレベルのヒントの構文:

```
hint_name([@query_block_name] [tbl_name [, tbl_name] ...])
```



```
hint_name([tbl_name@query_block_name [, tbl_name@query_block_name] ...])
```

構文は、次の用語を指します:

- **hint_name**: 次のヒント名を使用できます:
 - **BKA, NO_BKA**: 指定したテーブルに対するバッチングキーアクセスを有効または無効にします。
 - **BNL, NO_BNL**: 指定したテーブルのブロックネストループを有効または無効にします。MySQL 8.0.18 以降では、これらのヒントによってハッシュ結合の最適化も有効化および無効化されます。

注記

ブロックネストループの最適化は MySQL 8.0.20 以降のリリースでは削除されていますが、ハッシュ結合の有効化および無効化のためにこれらのヒントは引き続きサポートされています。

- **DERIVED_CONDITION_PUSHDOWN, NO_DERIVED_CONDITION_PUSHDOWN**: 指定したテーブルに対する導出表条件プッシュダウンの使用を有効または無効にします (MySQL 8.0.22 で追加)。詳細は、[セクション 8.2.2.5 「導出条件プッシュダウン最適化」](#) を参照してください。
- **HASH_JOIN, NO_HASH_JOIN**: 指定したテーブルに対するハッシュ結合の使用を有効または無効にします (MySQL 8.0.18 のみ。MySQL 8.0.19 以降では無効です)。
- **MERGE, NO_MERGE**: 指定したテーブル、ビュー参照または共通テーブル式のマージを有効にするか、マージを無効にしてかわりに実体化を使用してください。

注記

ブロックネストループまたはバッチキーアクセスヒントを使用して外部結合の内部テーブルの結合バッファリングを有効にするには、外部結合のすべての内部テーブルに対して結合バッファリングを有効にする必要があります。

- **tbl_name**: ステートメントで使用されるテーブルの名前。ヒントは、名前を付けるすべてのテーブルに適用されます。ヒントにテーブルが指定されていない場合は、そのヒントが発生したクエリーブロックのすべてのテーブルに適用されます。

テーブルにエイリアスがある場合、ヒントはテーブル名ではなくエイリアスを参照する必要があります。

ヒントのテーブル名はスキーマ名で修飾できません。

- **query_block_name**: ヒントが適用されるクエリーブロック。ヒントに先行する **@query_block_name** が含まれていない場合、ヒントは発生したクエリーブロックに適用されます。 **tbl_name@query_block_name** 構文の場合、ヒントは名前付きクエリーブロック内の名前付きテーブルに適用されます。クエリーブロックに名前を割り当てるには、[クエリーブロックのネーミングのためのオプティマイザヒント](#) を参照してください。

例:

```
SELECT /*+ NO_BKA(t1, t2) */ t1.* FROM t1 INNER JOIN t2 INNER JOIN t3;
SELECT /*+ NO_BNL() BKA(t1) */ t1.* FROM t1 INNER JOIN t2 INNER JOIN t3;
SELECT /*+ NO_MERGE(dt) */ * FROM (SELECT * FROM t1) AS dt;
```

テーブルレベルのヒントは、送信者テーブルではなく、前のテーブルからレコードを受信するテーブルに適用されます。次のステートメントがあるとします。

```
SELECT /*+ BNL(t2) */ FROM t1, t2;
```

オプティマイザが最初に **t1** を処理することを選択した場合、**t2** からの読取りを開始する前に **t1** から行をバッファリングすることで、ブロックネストループ結合が **t2** に適用されます。オプティマイザが最初に **t2** を処理することを選択した場合、**t2** は送信者テーブルであるため、ヒントは効果がありません。

MERGE ヒントおよび **NO_MERGE** ヒントには、次の優先順位ルールが適用されます:

- ヒントは、技術的な制約ではないオプティマイザのヒューリスティックより優先されます。(ヒントを提案として提供しても効果が無い場合、オプティマイザはヒントを無視する理由があります。)

- ヒントは、`optimizer_switch` システム変数の `derived_merge` フラグより優先されます。
- ビュー参照の場合、ビュー定義の `ALGORITHM={MERGE|TEMPTABLE}` 句は、ビューを参照するクエリーで指定されたヒントよりも優先されます。

インデックスレベルのオプティマイザヒント

インデックスレベルのヒントは、オプティマイザが特定のテーブルまたはインデックスに使用するインデックス処理戦略に影響します。これらのヒントタイプは、インデックス条件プッシュダウン (ICP)、マルチレンジ読み取り (MRR)、インデックスマージおよび範囲最適化の使用に影響します ([セクション8.2.1「SELECT ステートメントの最適化」](#)を参照)。

インデックスレベルのヒントの構文:

```
hint_name([@query_block_name] tbl_name [index_name [, index_name] ...])
hint_name(tbl_name@query_block_name [index_name [, index_name] ...])
```

構文は、次の用語を指します:

- `hint_name`: 次のヒント名を使用できます:
 - `GROUP_INDEX, NO_GROUP_INDEX`: `GROUP BY` 操作のインデックススキャンに対して、指定したインデックスを有効または無効にします。インデックスヒント `FORCE INDEX FOR GROUP BY`、`IGNORE INDEX FOR GROUP BY` と同等です。MySQL 8.0.20 以降で使用できます。
 - `INDEX, NO_INDEX`: `JOIN_INDEX`、`GROUP_INDEX` および `ORDER_INDEX` の組合せとして機能し、指定されたインデックスを任意のスコープおよびすべてのスコープに強制的に使用するか、`NO_JOIN_INDEX`、`NO_GROUP_INDEX` および `NO_ORDER_INDEX` の組合せとして使用します。これにより、サーバーは、任意のスコープおよびすべてのスコープに指定されたインデックスを無視します。`FORCE INDEX`、`IGNORE INDEX` と同等です。MySQL 8.0.20 以降で使用可能です。
 - `INDEX_MERGE, NO_INDEX_MERGE`: 指定したテーブルまたはインデックスのインデックスマージアクセス方法を有効または無効にします。このアクセス方法の詳細は、[セクション8.2.1.3「インデックスマージの最適化」](#)を参照してください。これらのヒントは、3つのすべてのインデックスマージアルゴリズムに適用されます。

`INDEX_MERGE` ヒントでは、オプティマイザは、指定されたインデックスセットを使用して、指定されたテーブルに対してインデックスマージを強制的に使用します。インデックスが指定されていない場合、オプティマイザは考えられるすべてのインデックスの組合せを考慮し、最もコストの低いものを選択します。インデックスの組合せが特定のステートメントに適用できない場合、ヒントは無視されることがあります。

`NO_INDEX_MERGE` ヒントは、指定されたインデックスのいずれかを含むインデックスマージの組合せを無効にします。ヒントにインデックスが指定されていない場合、テーブルに対するインデックスのマージは許可されません。
 - `JOIN_INDEX, NO_JOIN_INDEX`: MySQL で、`ref`、`range`、`index_merge` などのアクセス方法に対して指定されたインデックスを強制的に使用または無視します。`FORCE INDEX FOR JOIN`、`IGNORE INDEX FOR JOIN` と同等です。MySQL 8.0.20 以降で使用できます。
 - `MRR, NO_MRR`: 指定したテーブルまたはインデックスの MRR を有効または無効にします。MRR ヒントは、InnoDB および MyISAM テーブルにのみ適用されます。このアクセス方法の詳細は、[セクション8.2.1.11「Multi-Range Read の最適化」](#)を参照してください。
 - `NO_ICP`: 指定したテーブルまたはインデックスの ICP を無効にします。デフォルトでは、ICP は最適化戦略の候補であるため、有効にするヒントはありません。このアクセス方法の詳細は、[セクション8.2.1.6「インデックスコンディションプッシュダウンの最適化」](#)を参照してください。
 - `NO_RANGE_OPTIMIZATION`: 指定したテーブルまたはインデックスのインデックス範囲アクセスを無効にします。このヒントは、テーブルまたはインデックスのインデックスマージおよびインデックスのループスキャンも無効にします。デフォルトでは、範囲アクセスは最適化戦略の候補であるため、有効にするヒントはありません。

このヒントは、範囲の数が多く、範囲の最適化に多くのリソースが必要な場合に役立ちます。

- **ORDER_INDEX, NO_ORDER_INDEX:** MySQL で、指定されたインデックスを使用または無視して行をソートします。 **FORCE INDEX FOR ORDER BY**、 **IGNORE INDEX FOR ORDER BY** と同等です。 MySQL 8.0.20 以降で使用可能です。
- **SKIP_SCAN, NO_SKIP_SCAN:** 指定したテーブルまたはインデックスのスキップアクセス方法を有効または無効にします。 このアクセス方法の詳細は、 [スキャン範囲アクセス方法のスキップ](#) を参照してください。 これらのヒントは、 MySQL 8.0.13 の時点で使用できます。

SKIP_SCAN ヒントでは、オプティマイザは、指定されたインデックスセットを使用して、指定されたテーブルに対してスキップスキャンを強制的に使用します。 インデックスが指定されていない場合、オプティマイザは考えられるすべてのインデックスを考慮し、最もコストの低いインデックスを選択します。 インデックスが特定のステートメントに適用できない場合、ヒントは無視されることがあります。

NO_SKIP_SCAN ヒントは、指定されたインデックスのスキップスキャンを無効にします。 ヒントにインデックスが指定されていない場合、テーブルに対してスキップスキャンは許可されません。

- **tbl_name:** ヒントが適用されるテーブル。
- **index_name:** 指定したテーブル内のインデックスの名前。 ヒントは、名前を付けるすべてのインデックスに適用されます。 ヒントにインデックスが指定されていない場合は、テーブル内のすべてのインデックスに適用されます。

主キーを参照するには、 **PRIMARY** という名前を使用します。 テーブルのインデックス名を表示するには、 **SHOW INDEX** を使用します。

- **query_block_name:** ヒントが適用されるクエリーブロック。 ヒントに先行する **@query_block_name** が含まれていない場合、ヒントは発生したクエリーブロックに適用されます。 **tbl_name@query_block_name** 構文の場合、ヒントは名前付きクエリーブロック内の名前付きテーブルに適用されます。 クエリーブロックに名前を割り当てるには、 [クエリーブロックのネーミングのためのオプティマイザヒント](#) を参照してください。

例:

```
SELECT /*+ INDEX_MERGE(t1 f3, PRIMARY) */ f2 FROM t1
  WHERE f1 = 'o' AND f2 = f3 AND f3 <= 4;
SELECT /*+ MRR(t1) */ * FROM t1 WHERE f2 <= 3 AND 3 <= f3;
SELECT /*+ NO_RANGE_OPTIMIZATION(t3 PRIMARY, f2_idx) */ f1
  FROM t3 WHERE f1 > 30 AND f1 < 33;
INSERT INTO t3(f1, f2, f3)
  (SELECT /*+ NO_ICP(t2) */ t2.f1, t2.f2, t2.f3 FROM t1, t2
   WHERE t1.f1=t2.f1 AND t2.f2 BETWEEN t1.f1
     AND t1.f2 AND t2.f2 + 1 >= t1.f1 + 1);
SELECT /*+ SKIP_SCAN(t1 PRIMARY) */ f1, f2
  FROM t1 WHERE f2 > 40;
```

次の例ではインデックスマージヒントを使用しますが、他のインデックスレベルのヒントは、 **optimizer_switch** システム変数またはインデックスヒントに関して、オプティマイザヒントの無視と優先順位に関して同じ原則に従います。

t1 テーブルに **a**, **b**, **c** および **d** カラムがあり、 **i_a**, **i_b** および **i_c** という名前のインデックスが **a**, **b**, および **c** にそれぞれ存在するとします:

```
SELECT /*+ INDEX_MERGE(t1 i_a, i_b, i_c) */ * FROM t1
  WHERE a = 1 AND b = 2 AND c = 3 AND d = 4;
```

この場合、 **(i_a, i_b, i_c)** にはインデックスマージが使用されます。

```
SELECT /*+ INDEX_MERGE(t1 i_a, i_b, i_c) */ * FROM t1
  WHERE b = 1 AND c = 2 AND d = 3;
```

この場合、 **(i_b, i_c)** にはインデックスマージが使用されます。

```
/*+ INDEX_MERGE(t1 i_a, i_b) NO_INDEX_MERGE(t1 i_b) */
```

同じテーブルに先行するヒントがあるため、 **NO_INDEX_MERGE** は無視されます。

```
/*+ NO_INDEX_MERGE(t1 i_a, i_b) INDEX_MERGE(t1 i_b) */
```

同じテーブルに先行するヒントがあるため、`INDEX_MERGE` は無視されます。

`INDEX_MERGE` および `NO_INDEX_MERGE` オプティマイザヒントには、次の優先順位ルールが適用されます:

- オプティマイザヒントが指定され、適用可能な場合は、`optimizer_switch` システム変数のインデックスマージ関連フラグよりも優先されます。

```
SET optimizer_switch='index_merge_intersection=off';
SELECT /*+ INDEX_MERGE(t1 i_b, i_c) */ * FROM t1
WHERE b = 1 AND c = 2 AND d = 3;
```

ヒントは `optimizer_switch` よりも優先されます。この場合、`(i_b, i_c)` にはインデックスマージが使用されます。

```
SET optimizer_switch='index_merge_intersection=on';
SELECT /*+ INDEX_MERGE(t1 i_b) */ * FROM t1
WHERE b = 1 AND c = 2 AND d = 3;
```

ヒントには 1 つのインデックスのみが指定されているため、適用できず、`optimizer_switch` フラグ (`on`) が適用されます。インデックスマージは、オプティマイザがコスト効率に優れていると評価した場合に使用されます。

```
SET optimizer_switch='index_merge_intersection=off';
SELECT /*+ INDEX_MERGE(t1 i_b) */ * FROM t1
WHERE b = 1 AND c = 2 AND d = 3;
```

ヒントには 1 つのインデックスのみが指定されているため、適用できず、`optimizer_switch` フラグ (`off`) が適用されます。インデックスマージは使用されません。

- インデックスレベルのオプティマイザヒント `GROUP_INDEX`, `INDEX`, `JOIN_INDEX` および `ORDER_INDEX` はすべて、同等の `FORCE INDEX` ヒントより優先されます。つまり、`FORCE INDEX` ヒントは無視されます。同様に、`NO_GROUP_INDEX`, `NO_INDEX`, `NO_JOIN_INDEX` および `NO_ORDER_INDEX` のヒントはすべて `IGNORE INDEX` の同等のヒントよりも優先され、無視されます。

インデックスレベルのオプティマイザヒント `GROUP_INDEX`, `NO_GROUP_INDEX`, `INDEX`, `NO_INDEX`, `JOIN_INDEX`, `NO_JOIN_INDEX`, `ORDER_INDEX` および `NO_ORDER_INDEX` ヒントは、他のすべてのオプティマイザヒント (他のインデックスレベルのオプティマイザヒントを含む) よりも優先されます。その他のオプティマイザヒントは、これらによって許可されるインデックスにのみ適用されます。

`GROUP_INDEX`, `INDEX`, `JOIN_INDEX` および `ORDER_INDEX` のヒントはすべて `FORCE INDEX` と同等であり、`USE INDEX` とは同等ではありません。これは、これらのヒントの 1 つ以上を使用することは、いずれかの名前付きインデックスを使用してテーブル内の行を検索する方法がない場合にのみテーブルスキャンが使用されることを意味するためです。MySQL が `USE INDEX` の特定のインスタンスと同じインデックスまたはインデックスのセットを使用するようにするには、`NO_INDEX`, `NO_JOIN_INDEX`, `NO_GROUP_INDEX`, `NO_ORDER_INDEX` またはこれらの組合せを使用できます。

`USE INDEX` がクエリー `SELECT a,c FROM t1 USE INDEX FOR ORDER BY (i_a) ORDER BY a` に与える影響をシミュレートするには、`NO_ORDER_INDEX` オプティマイザヒントを使用して、次のようなものを除くテーブルのすべてのインデックスをカバーできます:

```
SELECT /*+ NO_ORDER_INDEX(t1 i_b,i_c) */ a,c
FROM t1
ORDER BY a;
```

次に示すように、`NO_ORDER_BY` によって `USE INDEX` が無視されるため、テーブル全体の `NO_ORDER_INDEX` を `USE INDEX FOR ORDER BY` と組み合わせようとしても機能しません:

```
mysql> EXPLAIN SELECT /*+ NO_ORDER_INDEX(t1) */ a,c FROM t1
-> USE INDEX FOR ORDER BY (i_a) ORDER BY aIG
***** 1. row *****
id: 1
select_type: SIMPLE
table: t1
partitions: NULL
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 256
```

```
filtered: 100.00
Extra: Using filesort
```

- **USE INDEX**、**FORCE INDEX** および **IGNORE INDEX** のインデックスヒントは、**INDEX_MERGE** および **NO_INDEX_MERGE** のオプティマイザヒントよりも優先度が高くなります。

```
/*+ INDEX_MERGE(t1 i_a, i_b, i_c) */ ... IGNORE INDEX i_a
```

IGNORE INDEX は **INDEX_MERGE** よりも優先されるため、インデックス *i_a* はインデックスマージで使用可能な範囲から除外されます。

```
/*+ NO_INDEX_MERGE(t1 i_a, i_b) */ ... FORCE INDEX i_a, i_b
```

FORCE INDEX のため、*i_a*、*i_b* ではインデックスマージは許可されていませんが、オプティマイザは **range** または **ref** のアクセスに *i_a* または *i_b* のいずれかを使用するように強制されます。競合はありません。両方のヒントを使用できます。

- **IGNORE INDEX** ヒントで複数のインデックスが指定されている場合、それらのインデックスはインデックスマージに使用できません。
- **FORCE INDEX** ヒントおよび **USE INDEX** ヒントでは、名前付きインデックスのみをインデックスマージに使用できます。

```
SELECT /*+ INDEX_MERGE(t1 i_a, i_b, i_c) */ a FROM t1
FORCE INDEX (i_a, i_b) WHERE c = 'h' AND a = 2 AND b = 'b';
```

インデックスマージ交差アクセスアルゴリズムは、(*i_a*, *i_b*) に使用されます。 **FORCE INDEX** が **USE INDEX** に変更された場合も同様です。

サブクエリーオプティマイザヒント

サブクエリーヒントは、準結合変換を使用するかどうか、許可する準結合戦略、および準結合を使用しない場合はサブクエリー実体化と **IN** から **EXISTS** への変換のどちらを使用するかに影響します。これらの最適化の詳細は、[セクション8.2.2「サブクエリー、導出テーブル、ビュー参照および共通テーブル式の最適化」](#)を参照してください。

準結合戦略に影響するヒントの構文は、次のとおりです：

```
hint_name([[@query_block_name] [strategy [ , strategy] ...])
```

構文は、次の用語を指します：

- **hint_name** : 次のヒント名を使用できます：
 - **SEMIJOIN**, **NO_SEMIJOIN**: 名前付き準結合戦略を有効または無効にします。
- **strategy**: 有効または無効にする準結合戦略。これらの戦略名は許可されます: **DUPSWEEP**, **FIRSTMATCH**, **LOOSESCAN**, **MATERIALIZATION**。

SEMIJOIN ヒントでは、戦略に名前が付けられていない場合、可能であれば、**optimizer_switch** システム変数に従って有効化された戦略に基づいて準結合が使用されます。戦略に名前が付けられているが、ステートメントには適用できない場合は、**DUPSWEEP** が使用されます。

NO_SEMIJOIN ヒントでは、戦略に名前が付いていない場合、準結合は使用されません。ステートメントに適用可能なすべての戦略を除外する戦略に名前が付けられている場合は、**DUPSWEEP** が使用されます。

あるサブクエリーが別のサブクエリー内にネストされ、その両方が外部クエリーの準結合にマージされる場合、最も内側のクエリーに対する準結合戦略の指定は無視されます。 **SEMIJOIN** および **NO_SEMIJOIN** ヒントを使用して、このようなネストしたサブクエリーの準結合変換を有効化または無効化できます。

DUPSWEEP が無効になっている場合、オプティマイザによって最適ではないクエリー計画が生成されることがあります。これは、最長一致検索中のヒューリスティックプルーニングが原因で発生します。これは、**optimizer_prune_level=0** を設定することで回避できます。

例:


```
SELECT /*+ NO_SEMIJOIN(@subq1 FIRSTMATCH, LOOSECAN) */ * FROM t2
WHERE t2.a IN (SELECT /*+ QB_NAME(subq1) */ a FROM t3);
SELECT /*+ SEMIJOIN(@subq1 MATERIALIZATION, DUPSWEEDEOUT) */ * FROM t2
WHERE t2.a IN (SELECT /*+ QB_NAME(subq1) */ a FROM t3);
```

サブクエリー実体化または **IN** から **EXISTS** への変換のどちらを使用するかに影響するヒントの構文は、次のとおりです:

```
SUBQUERY([@query_block_name] strategy)
```

ヒント名は常に **SUBQUERY** です。

SUBQUERY ヒントの場合、これらの **strategy** 値は許可されます: **INTOEXISTS**、**MATERIALIZATION**。

例:

```
SELECT id, a IN (SELECT /*+ SUBQUERY(MATERIALIZATION) */ a FROM t1) FROM t2;
SELECT * FROM t2 WHERE t2.a IN (SELECT /*+ SUBQUERY(INTOEXISTS) */ a FROM t1);
```

準結合および **SUBQUERY** ヒントの場合、先頭の **@query_block_name** でヒントが適用されるクエリーブロックを指定します。ヒントに先行する **@query_block_name** が含まれていない場合、ヒントは発生したクエリーブロックに適用されます。クエリーブロックに名前を割り当てるには、[クエリーブロックのネーミングのためのオプティマイザヒント](#) を参照してください。

ヒントコメントに複数のサブクエリーヒントが含まれている場合は、最初のヒントが使用されます。そのタイプの他の後続のヒントがある場合は、警告が生成されます。他のタイプの次のヒントは、暗黙的に無視されます。

ステートメント実行時オプティマイザヒント

MAX_EXECUTION_TIME ヒントは、**SELECT** ステートメントでのみ使用できます。サーバーがステートメントを終了するまでに、ステートメントの実行が許可される期間に制限 **N** (ミリ秒単位のタイムアウト値) を設定します:

```
MAX_EXECUTION_TIME(N)
```

タイムアウトが 1 秒 (1000 ミリ秒) の例:

```
SELECT /*+ MAX_EXECUTION_TIME(1000) */ * FROM t1 INNER JOIN t2 WHERE ...
```

MAX_EXECUTION_TIME(N) ヒントは、**N** ミリ秒のステートメント実行タイムアウトを設定します。このオプションが指定されていないか、**N** が 0 の場合、**max_execution_time** システム変数によって設定されたステートメントタイムアウトが適用されます。

MAX_EXECUTION_TIME ヒントは次のように適用できます:

- **UNION** やサブクエリーを含むステートメントなど、複数の **SELECT** キーワードを持つステートメントの場合、**MAX_EXECUTION_TIME** はステートメント全体に適用され、最初の **SELECT** の後に出現する必要があります。
- 読み取り専用の **SELECT** ステートメントに適用されます。読み取り専用でないステートメントは、副作用としてデータを変更するストアドファンクションを呼び出すステートメントです。
- ストアドプログラムの **SELECT** ステートメントには適用されず、無視されます。

可変設定のヒント構文

SET_VAR ヒントは、システム変数のセッション値を一時的に設定します (単一のステートメントの実行中)。例:

```
SELECT /*+ SET_VAR(sort_buffer_size = 16M) */ name FROM people ORDER BY name;
INSERT /*+ SET_VAR(foreign_key_checks=OFF) */ INTO t2 VALUES(2);
SELECT /*+ SET_VAR(optimizer_switch = 'mrr_cost_based=off') */ 1;
```

SET_VAR ヒントの構文:

```
SET_VAR(var_name = value)
```


`var_name` は、セッション値を持つシステム変数に名前を付けます (ただし、後で説明するように、そのようなすべての変数に名前を付けることはできません)。`value` は変数に割り当てる値で、値はスカラーである必要があります。

次のステートメントで示すように、`SET_VAR` では一時変数が変更されます:

```
mysql> SELECT @@unique_checks;
+-----+
| @@unique_checks |
+-----+
| 1 |
+-----+
mysql> SELECT /*+ SET_VAR(unique_checks=OFF) */ @@unique_checks;
+-----+
| @@unique_checks |
+-----+
| 0 |
+-----+
mysql> SELECT @@unique_checks;
+-----+
| @@unique_checks |
+-----+
| 1 |
+-----+
```

`SET_VAR` では、変数値を保存およびリストアする必要はありません。これにより、複数のステートメントを単一のステートメントで置き換えることができます。次の一連のステートメントについて考えてみます:

```
SET @saved_val = @@SESSION.var_name;
SET @@SESSION.var_name = value;
SELECT ...
SET @@SESSION.var_name = @saved_val;
```

順序は、次の単一のステートメントで置換できます:

```
SELECT /*+ SET_VAR(var_name = value) ...
```

スタンドアロンの `SET` ステートメントでは、セッション変数のネーミングに次の構文を使用できます:

```
SET SESSION var_name = value;
SET @@SESSION.var_name = value;
SET @@var_name = value;
```

`SET_VAR` ヒントはセッション変数にのみ適用されるため、セッションスコープは暗黙的であり、`SESSION`、`@@SESSION`、および`@@`は必要なく、許可されません。明示的なセッションインジケータ構文を含めると、`SET_VAR` ヒントは無視され、警告が表示されます。

すべてのセッション変数を `SET_VAR` で使用できるわけではありません。個々のシステム変数の説明は、各変数がヒント可能かどうかを示します。[セクション5.1.8「サーバーシステム変数」](#)を参照してください。また、システム変数を `SET_VAR` で使用して、実行時にチェックすることもできます。変数がヒント可能でない場合は、警告が発生します:

```
mysql> SELECT /*+ SET_VAR(collation_server = 'utf8') */ 1;
+---+
| 1 |
+---+
| 1 |
+---+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Warning
Code: 4537
Message: Variable 'collation_server' cannot be set using SET_VAR hint.
```

`SET_VAR` 構文では単一の変数のみを設定できますが、複数のヒントを指定して複数の変数を設定できます:

```
SELECT /*+ SET_VAR(optimizer_switch = 'mrr_cost_based=off')
          SET_VAR(max_heap_table_size = 1G) */ 1;
```

同じ変数名を持つ複数のヒントが同じステートメントに出現すると、最初のヒントが適用され、他のヒントは警告付きで無視されます:

```
SELECT /*+ SET_VAR(max_heap_table_size = 1G)
      SET_VAR(max_heap_table_size = 3G) */ 1;
```

この場合、2 番目のヒントは無視され、競合しているという警告が表示されます。

指定された名前を持つシステム変数がない場合、または変数値が正しくない場合、`SET_VAR` ヒントは警告付きで無視されます:

```
SELECT /*+ SET_VAR(max_size = 1G) */ 1;
SELECT /*+ SET_VAR(optimizer_switch = 'mrr_cost_based=yes') */ 1;
```

最初のステートメントには、`max_size` 変数はありません。2 番目のステートメントでは、`mrr_cost_based` は `on` または `off` の値を取るため、`yes` に設定しようとする試みは正しくありません。いずれの場合も、ヒントは警告付きで無視されます。

`SET_VAR` ヒントはステートメントレベルでのみ使用できます。サブクエリーで使用する場合、ヒントは無視され、警告が表示されます。

レプリカは、レプリケートされたステートメントの `SET_VAR` ヒントを無視して、セキュリティの問題が発生する可能性を回避します。

リソースグループのヒント構文

`RESOURCE_GROUP` オプティマイザヒントは、リソースグループの管理に使用されます ([セクション5.1.16「リソースグループ」](#) を参照)。このヒントは、ステートメントを実行するスレッドを名前付きリソースグループに一時的に割り当てます (ステートメントの期間中)。`RESOURCE_GROUP_ADMIN` または `RESOURCE_GROUP_USER` 権限が必要です。

例:

```
SELECT /*+ RESOURCE_GROUP(USR_default) */ name FROM people ORDER BY name;
INSERT /*+ RESOURCE_GROUP(Batch) */ INTO t2 VALUES(2);
```

`RESOURCE_GROUP` ヒントの構文:

```
RESOURCE_GROUP(group_name)
```

`group_name` は、ステートメントの実行中にスレッドを割り当てるリソースグループを示します。グループが存在しない場合、警告が発生し、ヒントは無視されます。

`RESOURCE_GROUP` ヒントは、最初のステートメントキーワード (`SELECT`, `INSERT`, `REPLACE`, `UPDATE` または `DELETE`) の後に指定する必要があります。

`RESOURCE_GROUP` の代替手段として、一時的にスレッドをリソースグループに割り当てる `SET RESOURCE GROUP` ステートメントがあります。[セクション13.7.2.4「SET RESOURCE GROUP ステートメント」](#) を参照してください。

クエリーブロックのネーミングのためのオプティマイザヒント

テーブルレベル、インデックスレベルおよびサブクエリーオプティマイザヒントでは、特定のクエリーブロックに引数構文の一部として名前を付けることができます。これらの名前を作成するには、`QB_NAME` ヒントを使用します。これにより、名前が発生したクエリーブロックに名前が割り当てられます:

```
QB_NAME(name)
```

`QB_NAME` ヒントを使用すると、他のヒントが適用されるクエリーブロックを明確にすることができます。また、複雑なステートメントを理解しやすくするために、すべての非クエリーブロック名ヒントを単一のヒントコメント内に指定することもできます。次のステートメントについて考えてみます:

```
SELECT ...
```

```
FROM (SELECT ...  
FROM (SELECT ... FROM ...)) ...
```

QB_NAME ヒントは、ステートメントのクエリーブロックに名前を割り当てます:

```
SELECT /*+ QB_NAME(qb1) */ ...  
FROM (SELECT /*+ QB_NAME(qb2) */ ...  
FROM (SELECT /*+ QB_NAME(qb3) */ ... FROM ...)) ...
```

他のヒントでは、これらの名前を使用して適切なクエリーブロックを参照できます:

```
SELECT /*+ QB_NAME(qb1) MRR(@qb1 t1) BKA(@qb2) NO_MRR(@qb3t1 idx1, id2) */ ...  
FROM (SELECT /*+ QB_NAME(qb2) */ ...  
FROM (SELECT /*+ QB_NAME(qb3) */ ... FROM ...)) ...
```

その結果、次のようになります:

- **MRR(@qb1 t1)** は、クエリーブロック **qb1** のテーブル **t1** に適用されます。
- **BKA(@qb2)** は、クエリーブロック **qb2** に適用されます。
- **NO_MRR(@qb3 t1 idx1, id2)** は、クエリーブロック **qb3** の **t1** テーブルのインデックス **idx1** および **idx2** に適用されます。

クエリーブロック名は識別子であり、有効な名前とその引用符の方法に関する通常のルールに従います ([セクション 9.2 「スキーマオブジェクト名」](#) を参照)。たとえば、空白を含むクエリーブロック名は引用符で囲む必要があります。引用符はバックティックを使用して使用できます:

```
SELECT /*+ BKA(@`my hint name`) */ ...  
FROM (SELECT /*+ QB_NAME(`my hint name`) */ ...) ...
```

ANSI_QUOTES SQL モードが有効な場合は、クエリーブロック名を二重引用符で囲むこともできます:

```
SELECT /*+ BKA(@"my hint name") */ ...  
FROM (SELECT /*+ QB_NAME("my hint name") */ ...) ...
```

8.9.4 インデックスヒント

インデックスヒントは、クエリー処理中にインデックスを選択する方法に関するオプティマイザ情報を提供します。ここで説明するインデックスヒントは、[セクション 8.9.3 「オプティマイザヒント」](#) で説明されているオプティマイザヒントとは異なります。インデックスヒントとオプティマイザヒントは、別々に、または一緒に使用できます。

インデックスヒントは、**SELECT** および **UPDATE** ステートメントにのみ適用されます。

インデックスヒントは、テーブル名の後に指定します。(SELECT ステートメントでテーブルを指定するための一般的な構文は、[セクション 13.2.10.2 「JOIN 句」](#) を参照してください。) インデックスヒントを含む個々のテーブルを参照する構文は、次のようになります:

```
tbl_name [[AS] alias] [index_hint_list]  
  
index_hint_list:  
  index_hint [index_hint] ...  
  
index_hint:  
  USE {INDEX|KEY}  
  [FOR {JOIN|ORDER BY|GROUP BY}] ([index_list])  
  | {IGNORE|FORCE} {INDEX|KEY}  
  [FOR {JOIN|ORDER BY|GROUP BY}] (index_list)  
  
index_list:  
  index_name [, index_name] ...
```

USE INDEX (index_list) ヒントは、名前付きインデックスのいずれかのみを使用してテーブル内の行を検索するように MySQL に指示します。代替構文 **IGNORE INDEX (index_list)** は、特定のインデックスを使用しないように MySQL に指示します。これらのヒントは、**EXPLAIN** によって、MySQL が可能性のあるインデックスのリストから間違ったインデックスを使用していることが示された場合に役立ちます。

FORCE INDEX ヒントは **USE INDEX (index_list)** のように機能しますが、テーブルスキャンは very コストが高いとみなされます。つまり、テーブルスキャンは、名前付きインデックスのいずれかを使用してテーブル内の行を検索する方法がない場合にのみ使用されます。

注記

MySQL 8.0.20 の時点で、サーバーは、インデックスオプティマイザヒント **JOIN_INDEX**、**GROUP_INDEX**、**ORDER_INDEX**、および **INDEX(NO_JOIN_INDEX, NO_GROUP_INDEX, NO_ORDER_INDEX)** および **FORCE INDEX** オプティマイザヒントに相当および置き替える)、および **NO_INDEX** オプティマイザヒント (**IGNORE INDEX** インデックスヒントに相当し、それらを置き換える) をサポートします。したがって、**USE INDEX**、**FORCE INDEX** および **IGNORE INDEX** は、MySQL の将来のリリースで非推奨になり、後で完全に削除される予定です。詳細は、[インデックスレベルのオプティマイザヒント](#) を参照してください。

各ヒントには、カラム名ではなくインデックス名が必要です。主キーを参照するには、**PRIMARY** という名前を使用します。テーブルのインデックス名を表示するには、**SHOW INDEX** ステートメントまたは **INFORMATION_SCHEMA.STATISTICS** テーブルを使用します。

index_name 値は、完全なインデックス名である必要はありません。インデックス名のあいまいでないプリフィクスにすることができます。プリフィクスがあいまいな場合は、エラーが発生します。

例:

```
SELECT * FROM table1 USE INDEX (col1_index,col2_index)
WHERE col1=1 AND col2=2 AND col3=3;

SELECT * FROM table1 IGNORE INDEX (col3_index)
WHERE col1=1 AND col2=2 AND col3=3;
```

インデックスヒントの構文には、次の特性があります。

- 構文的には、**index_list** for **USE INDEX** (「「インデックスを使用しない」」を意味する) を省略することが有効です。**index_list** for **FORCE INDEX** または **IGNORE INDEX** の省略は構文エラーです。
- ヒントに **FOR** 句を追加することで、インデックスヒントの有効範囲を指定できます。これにより、クエリー処理の様々なフェーズの実行計画のオプティマイザ選択をよりきめ細かく制御できます。MySQL がテーブル内の行の検索方法および結合の処理方法を決定するときに使用されるインデックスにのみ影響を与えるには、**FOR JOIN** を使用します。行をソートまたはグループ化するためのインデックス使用に影響を与えるには、**FOR ORDER BY** または **FOR GROUP BY** を使用します。
- 複数のインデックスヒントを指定できます。

```
SELECT * FROM t1 USE INDEX (i1) IGNORE INDEX FOR ORDER BY (i2) ORDER BY a;
```

(同じヒント内であっても) 複数のヒントで同じインデックスに名前を付けるとエラーになりません:

```
SELECT * FROM t1 USE INDEX (i1) USE INDEX (i1,i1);
```

ただし、同じテーブルに対して **USE INDEX** と **FORCE INDEX** を混在させると、エラーが発生します。

```
SELECT * FROM t1 USE INDEX FOR JOIN (i1) FORCE INDEX FOR JOIN (i2);
```

インデックスヒントに **FOR** 句が含まれていない場合、ヒントの有効範囲はステートメントのすべての部分に適用されます。たとえば、次のヒント

```
IGNORE INDEX (i1)
```

は次のヒントの組み合わせと同等です。

```
IGNORE INDEX FOR JOIN (i1)
IGNORE INDEX FOR ORDER BY (i1)
IGNORE INDEX FOR GROUP BY (i1)
```

MySQL 5.0 では、**FOR** 句のないヒントスコープは、行の取得にのみ適用されました。**FOR** 句が存在しないときにサーバーがこの古い動作を使用するには、サーバーの起動時に **old** システム変数を有効にします。レプリケーションセットアップでこの変数を有効にする場合は注意してください。ステートメントベースのバイナリロギングでは、ソースとレプリカのモードが異なると、レプリケーションエラーが発生する可能性があります。

インデックスヒントが処理されるとき、これらのインデックスヒントは、型 ([USE](#)、[FORCE](#)、[IGNORE](#)) およびスコープ ([FOR JOIN](#)、[FOR ORDER BY](#)、[FOR GROUP BY](#)) ごとに 1 つのリストに収集されます。例:

```
SELECT * FROM t1
  USE INDEX () IGNORE INDEX (i2) USE INDEX (i1) USE INDEX (i2);
```

次と同等です。

```
SELECT * FROM t1
  USE INDEX (i1,i2) IGNORE INDEX (i2);
```

そのあと、インデックスヒントは、スコープごとに次の順序で適用されます。

1. [{USE|FORCE} INDEX](#) が存在する場合は、これが適用されます。(存在しない場合は、オプティマイザによって決定されたインデックスのセットが使用されます。)
2. 前の手順の結果に対して、[IGNORE INDEX](#) が適用されます。たとえば、次の 2 つのクエリーは同等です。

```
SELECT * FROM t1 USE INDEX (i1) IGNORE INDEX (i2) USE INDEX (i2);
SELECT * FROM t1 USE INDEX (i1);
```

[FULLTEXT](#) の検索の場合、インデックスヒントは次のように機能します。

- 自然言語モードの検索の場合、インデックスヒントは暗黙のうちに無視されます。たとえば、[IGNORE INDEX\(i1\)](#) は警告なしで無視され、インデックスは引き続き使用されます。
- ブールモードの検索の場合、[FOR ORDER BY](#) または [FOR GROUP BY](#) を含むインデックスヒントは暗黙のうちに無視されます。[FOR JOIN](#) を含むインデックスヒント、または [FOR](#) 修飾子を含まないインデックスヒントは受け付けられます。ヒントが [FULLTEXT](#) 以外の検索に適用される場合とは異なり、このヒントは、クエリー実行のすべてのフェーズ (行の検索と取得、グループ化、および順序付け) に使用されます。これは、ヒントが [FULLTEXT](#) 以外のインデックスに対して指定されている場合でも当てはまります。

たとえば、次の 2 つのクエリーは同等です。

```
SELECT * FROM t
  USE INDEX (index1)
  IGNORE INDEX (index1) FOR ORDER BY
  IGNORE INDEX (index1) FOR GROUP BY
  WHERE ... IN BOOLEAN MODE ...;

SELECT * FROM t
  USE INDEX (index1)
  WHERE ... IN BOOLEAN MODE ...;
```

8.9.5 オプティマイザコストモデル

実行計画を生成するために、オプティマイザは、クエリーの実行中に発生する様々な操作のコストの見積りに基づくコストモデルを使用します。オプティマイザには、実行計画に関する決定を下すために、コンパイル済みのデフォルトの「コスト定数」のセットが用意されています。

オプティマイザには、実行計画の作成時に使用するコスト見積りのデータベースもあります。これらの見積りは、[mysql](#) システムデータベースの [server_cost](#) テーブルおよび [engine_cost](#) テーブルに格納され、いつでも構成できます。これらのテーブルの目的は、オプティマイザがクエリー実行計画に到達しようとしたときに使用するコスト見積りを簡単に調整できるようにすることです。

- [原価モデル一般工程](#)
- [コストモデルデータベース](#)
- [コストモデルデータベースの変更](#)

原価モデル一般工程

構成可能なオプティマイザコストモデルは、次のように動作します:

- サーバーは、起動時にコストモデルテーブルをメモリーに読み取り、実行時にインメモリー値を使用します。テーブルに指定されている `NULL` 以外のコスト見積りは、対応するコンパイル済のデフォルトのコスト定数より優先されます。 `NULL` の見積りは、コンパイル済のデフォルトを使用するようオプティマイザに指示します。
- 実行時に、サーバーはコストテーブルを再度読み取ることができます。これは、ストレージエンジンが動的にロードされたとき、または `FLUSH OPTIMIZER_COSTS` ステートメントが実行されたときに発生します。
- 原価テーブルを使用すると、サーバー管理者はテーブルのエントリを変更することで、原価見積りを簡単に調整できます。また、エントリコストを `NULL` に設定することで、デフォルトに戻すことも簡単です。オプティマイザはインメモリーのコスト値を使用するため、テーブルに対する変更の後に `FLUSH OPTIMIZER_COSTS` を実行して有効にする必要があります。
- クライアントセッションの開始時に現在のメモリー内コストの見積りは、そのセッションが終了するまでそのセッション全体に適用されます。特に、サーバーがコストテーブルを再度読み取る場合、変更された見積りはその後に開始されるセッションにのみ適用されます。既存のセッションは影響を受けません。
- コストテーブルは、特定のサーバーインスタンスに固有です。サーバーは、コストテーブルの変更をレプリカにレプリケートしません。

コストモデルデータベース

オプティマイザコストモデルデータベースは、クエリーの実行中に発生する操作のコスト見積り情報を含む、`mysql` システムデータベース内の次の 2 つのテーブルで構成されます:

- `server_cost`: 一般的なサーバー操作のオプティマイザコストの見積り
- `engine_cost`: 特定のストレージエンジンに固有の操作のオプティマイザコストの見積り

`server_cost` テーブルには、次のカラムが含まれます:

- `cost_name`

コストモデルで使用されるコスト見積りの名前。名前では大文字と小文字は区別されません。このテーブルの読み取り時にサーバーがコスト名を認識しない場合、エラーログに警告が書き込まれます。

- `cost_value`

コスト見積り値。値が `NULL` 以外の場合、サーバーはそれをコストとして使用します。それ以外の場合は、デフォルトの見積り (コンパイルされた値) が使用されます。DBA は、このカラムを更新することでコスト見積りを変更できます。サーバーは、このテーブルの読み取り時にコスト値が無効 (正でない) であることを検出すると、エラーログに警告を書き込みます。

(`NULL` を指定するエントリの) デフォルトのコスト見積りを上書きするには、コストを `NULL` 以外の値に設定します。デフォルトに戻すには、値を `NULL` に設定します。次に、`FLUSH OPTIMIZER_COSTS` を実行して、コストテーブルを再度読み取るようにサーバーに指示します。

- `last_update`

最後の行更新の時刻。

- `comment`

原価見積りに関連付けられた摘要コメント。DBA は、このカラムを使用して、コスト見積り行に特定の値が格納される理由に関する情報を提供できます。

- `default_value`

原価見積りのデフォルト (コンパイル済) 値。このカラムは、関連するコスト見積りが変更された場合でもその値を保持する読み取り専用の生成カラムです。実行時にテーブルに追加される行の場合、このカラムの値は `NULL` です。

`server_cost` テーブルの主キーは `cost_name` カラムであるため、コスト見積りに対して複数のエントリを作成することはできません。

サーバーは、`server_cost` テーブルの次の `cost_name` 値を認識します:

- `disk temptable create cost`, `disk temptable row cost`

ディスクベースのストレージエンジン (InnoDB または MyISAM) に格納されている内部的に作成された一時テーブルのコスト見積り。これらの値を大きくすると、内部一時テーブルを使用するコストの見積りが増加し、オプティマイザがクエリー計画を使用しやすくなります。このようなテーブルの詳細は、[セクション8.4.4「MySQLでの内部一時テーブルの使用」](#)を参照してください。

これらのディスクパラメータのデフォルト値が、対応するメモリーパラメータ (`memory temptable create cost`, `memory temptable row cost`) のデフォルト値より大きいほど、ディスクベースのテーブルの処理コストが高くなります。

- `key compare cost`

レコードキーを比較するコスト。この値を増やすと、多数のキーを比較するクエリー計画のコストが高くなります。たとえば、`filesort` を実行するクエリー計画は、インデックスを使用したソートを回避するクエリー計画よりも比較的成本が高くなります。

- `memory temptable create cost`, `memory temptable row cost`

MEMORY ストレージエンジンに格納されている内部的に作成された一時テーブルのコスト見積り。これらの値を大きくすると、内部一時テーブルを使用するコストの見積りが増加し、オプティマイザがクエリー計画を使用しやすくなります。このようなテーブルの詳細は、[セクション8.4.4「MySQLでの内部一時テーブルの使用」](#)を参照してください。

これらのメモリーパラメータのデフォルト値が、対応するディスクパラメータ (`disk temptable create cost`, `disk temptable row cost`) のデフォルト値より小さいほど、メモリーベースのテーブルの処理コストは低くなります。

- `row evaluate cost`

レコード条件を評価するコスト。この値を増やすと、多数の行を調査するクエリー計画が、調査する行数が少ないクエリー計画よりもコストが高くなります。たとえば、読取り行数が少ないレンジスキャンよりも、テーブルスキャンの方が比較的成本が高くなります。

`engine cost` テーブルには、次のカラムが含まれます:

- `engine name`

このコスト見積りが適用されるストレージエンジンの名前。名前では大文字と小文字は区別されません。値が `default` の場合は、独自の名前付きエントリを持たないすべてのストレージエンジンに適用されます。このテーブルの読取り時にサーバーがエンジン名を認識しない場合、エラーログに警告が書き込まれます。

- `device type`

この原価見積りが適用される設備タイプ。このカラムは、ハードディスクドライブとソリッドステートドライブなど、ストレージデバイスタイプごとに異なるコスト見積りを指定するためのものです。現在、この情報は使用されず、許可される値は0のみです。

- `cost name`

`server cost` テーブルと同じです。

- `cost value`

`server cost` テーブルと同じです。

- `last update`

`server cost` テーブルと同じです。

- `comment`

`server cost` テーブルと同じです。

- `default value`

原価見積のデフォルト (コンパイル済) 値。このカラムは、関連するコスト見積りが変更された場合でもその値を保持する読取り専用の生成カラムです。実行時にテーブルに追加される行の場合、このカラムの値は `NULL` ですが、行の `cost_name` 値が元の行のいずれかと同じ場合、`default_value` カラムの値はその行と同じになります。

`engine_cost` テーブルの主キーは (`cost_name`, `engine_name`, `device_type`) カラムで構成されるタプルであるため、これらのカラムの値の組合せに対して複数のエントリを作成することはできません。

サーバーは、`engine_cost` テーブルの次の `cost_name` 値を認識します:

- `io_block_read_cost`

ディスクからインデックスまたはデータブロックを読み取るコスト。この値を増やすと、多くのディスクブロックを読み取るクエリー計画が、読み取るディスクブロックが少ないクエリー計画よりもコストが高くなります。たとえば、読取りブロック数が少ないレンジスキャンよりも、テーブルスキャンの方が比較的成本が高くなります。

- `memory_block_read_cost`

`io_block_read_cost` と似ていますが、インメモリーデータベースバッファからインデックスまたはデータブロックを読み取るコストを表します。

`io_block_read_cost` と `memory_block_read_cost` の値が異なる場合、同じクエリーの 2 つの実行間で実行計画が変わる可能性があります。メモリーアクセスのコストがディスクアクセスのコストよりも低いとします。その場合、データがメモリー内にあるため、データがバッファプールに読み取られる前のサーバー起動時に、クエリーの実行後とは異なる計画が得られることがあります。

コストモデルデータベースの変更

コストモデルパラメータをデフォルトから変更する DBA の場合は、値を二重または停止して効果を測定してください。

`io_block_read_cost` および `memory_block_read_cost` パラメータを変更すると、結果が得られる価値が高くなる可能性があります。これらのパラメータ値を使用すると、データアクセス方法のコストモデルで、様々なソースからの情報の読取りコスト (ディスクからの情報の読取りコストとメモリーバッファにすでに存在する情報の読取りコスト) を考慮できます。たとえば、`io_block_read_cost` を `memory_block_read_cost` より大きい値に設定すると、メモリーにすでに保持されている情報を読み取るクエリー計画が、ディスクから読み取る必要がある計画よりもオプティマイザによって優先されます。

次の例では、`io_block_read_cost` のデフォルト値を変更する方法を示します:

```
UPDATE mysql.engine_cost
SET cost_value = 2.0
WHERE cost_name = 'io_block_read_cost';
FLUSH OPTIMIZER_COSTS;
```

次の例では、`InnoDB` ストレージエンジンについてのみ `io_block_read_cost` の値を変更する方法を示します:

```
INSERT INTO mysql.engine_cost
VALUES ('InnoDB', 0, 'io_block_read_cost', 3.0,
CURRENT_TIMESTAMP, 'Using a slower disk for InnoDB');
FLUSH OPTIMIZER_COSTS;
```

8.9.6 オプティマイザ統計

`column_statistics` データディクショナリテーブルには、オプティマイザがクエリー実行計画を作成するために使用する、カラム値に関するヒストグラム統計が格納されます。ヒストグラム管理を実行するには、`ANALYZE TABLE` ステートメントを使用します。

`column_statistics` テーブルには、次の特性があります:

- このテーブルには、ジオメトリタイプ (空間データ) および `JSON` を除くすべてのデータ型のカラムの統計が含まれます。

- テーブルは永続的であるため、サーバーが起動するたびにカラム統計を作成する必要はありません。
- サーバーはテーブルの更新を実行しますが、ユーザーは実行しません。

`column_statistics` テーブルはデータディクショナリの一部であるため、ユーザーは直接アクセスできません。ヒストグラム情報は、データディクショナリテーブルのビューとして実装されている `INFORMATION_SCHEMA.COLUMN_STATISTICS` を使用して入手できます。 `COLUMN_STATISTICS` には、次のカラムがあります:

- `SCHEMA_NAME`, `TABLE_NAME`, `COLUMN_NAME`: 統計が適用されるスキーマ、テーブルおよびカラムの名前。
- `HISTOGRAM`: ヒストグラムとして格納される、カラム統計を記述する `JSON` 値。

カラムヒストグラムには、カラムに格納されている値の範囲の一部のバケットが含まれます。ヒストグラムは、カラム統計の柔軟な表現を可能にする `JSON` オブジェクトです。ヒストグラムオブジェクトのサンプルを次に示します:

```
{
  "buckets": [
    [
      1,
      0.3333333333333333
    ],
    [
      2,
      0.6666666666666666
    ],
    [
      3,
      1
    ]
  ],
  "null-values": 0,
  "last-updated": "2017-03-24 13:32:40.000000",
  "sampling-rate": 1,
  "histogram-type": "singleton",
  "number-of-buckets-specified": 128,
  "data-type": "int",
  "collation-id": 8
}
```

ヒストグラムオブジェクトには、次のキーがあります:

- `buckets`: ヒストグラムバケット。バケット構造はヒストグラムタイプによって異なります。
 - `singleton` ヒストグラムの場合、バケットには次の 2 つの値が含まれます:
 - 値 1: バケットの値。型はカラムのデータ型によって異なります。
 - 値 2: 値の累積周波数を表す `double`。たとえば、.25 および .75 は、カラムの値の 25% および 75% がバケット値以下であることを示します。
 - `equi-height` ヒストグラムの場合、バケットには次の 4 つの値が含まれます:
 - 値 1, 2: バケットの下限値と上限値。型はカラムのデータ型によって異なります。
 - 値 3: 値の累積周波数を表す `double`。たとえば、.25 および .75 は、カラムの値の 25% および 75% がバケットの上限値以下であることを示します。
 - 値 4: バケットの下限値から上限値までの範囲内の個別値の数。
- `null-values`: 0.0 と 1.0 の間の数値で、SQL `NULL` 値であるカラム値の割合を示します。0 の場合、カラムに `NULL` 値は含まれません。
- `last-updated`: ヒストグラムが生成された日時 (`YYYY-MM-DD hh:mm:ss.uuuuuu` 形式の UTC 値として)。
- `sampling-rate`: ヒストグラムを作成するためにサンプリングされたデータの割合を示す、0.0 と 1.0 の間の数値。値 1 は、すべてのデータが読み取られたことを意味します (サンプリングなし)。
- `histogram-type`: ヒストグラムタイプ:

- **singleton**: 1つのバケットは、カラム内の1つの値を表します。このヒストグラムタイプは、カラムの個別値の数が、ヒストグラムを生成した **ANALYZE TABLE** ステートメントで指定されたバケットの数以下の場合に作成されます。
- **equi-height**: 1つのバケットは値の範囲を表します。このヒストグラムタイプは、カラム内の個別値の数が、ヒストグラムを生成した **ANALYZE TABLE** ステートメントで指定されたバケットの数より多い場合に作成されます。
- **number-of-buckets-specified**: ヒストグラムを生成した **ANALYZE TABLE** ステートメントで指定されたバケットの数。
- **data-type**: このヒストグラムに含まれるデータのタイプ。これは、永続記憶域からメモリーにヒストグラムを読み取って解析する場合に必要です。値は、**int**、**uint** (符号なし整数)、**double**、**decimal**、**datetime** または **string** (文字列およびバイナリ文字列を含む) のいずれかです。
- **collation-id**: ヒストグラムデータの照合 ID。これは、**data-type** 値が **string** の場合に最も意味があります。値は、**INFORMATION_SCHEMA.COLLATIONS** テーブルの **ID** カラムの値に対応します。

ヒストグラムオブジェクトから特定の値を抽出するには、**JSON** 操作を使用できます。例:

```
mysql> SELECT
  TABLE_NAME, COLUMN_NAME,
  HISTOGRAM->'$.data-type' AS 'data-type',
  JSON_LENGTH(HISTOGRAM->'$.buckets') AS 'bucket-count'
FROM INFORMATION_SCHEMA.COLUMN_STATISTICS;
```

TABLE_NAME	COLUMN_NAME	data-type	bucket-count
country	Population	int	226
city	Population	int	1024
countrylanguage	Language	string	457

オプティマイザは、統計が収集されるデータ型のカラムにヒストグラム統計を使用します (該当する場合)。オプティマイザはヒストグラム統計を適用し、定数値に対するカラム値の比較の選択性 (フィルタリング効果) に基づいて行の見積りを決定します。これらのフォームの述語は、ヒストグラムの使用に適しています:

```
col_name = constant
col_name <> constant
col_name != constant
col_name > constant
col_name < constant
col_name >= constant
col_name <= constant
col_name IS NULL
col_name IS NOT NULL
col_name BETWEEN constant AND constant
col_name NOT BETWEEN constant AND constant
col_name IN (constant[, constant] ...)
col_name NOT IN (constant[, constant] ...)
```

たとえば、次のステートメントにはヒストグラムの使用に適した述語が含まれています:

```
SELECT * FROM orders WHERE amount BETWEEN 100.0 AND 300.0;
SELECT * FROM tbl WHERE col1 = 15 AND col2 > 100;
```

定数値と比較するための要件には、**ABS()** や **FLOOR()** などの定数である関数が含まれます:

```
SELECT * FROM tbl WHERE col1 < ABS(-34);
```

ヒストグラム統計は、主にインデックス付けされていないカラムに役立ちます。ヒストグラム統計が適用可能なカラムにインデックスを追加すると、オプティマイザが行の見積りを行うのにも役立ちます。トレードオフは次のとおりです:

- テーブルデータが変更された場合は、インデックスを更新する必要があります。
- ヒストグラムはオンデマンドでのみ作成または更新されるため、テーブルデータの変更時にオーバーヘッドは発生しません。一方、統計は、次回更新されるまで、テーブルの変更が発生すると徐々に期限切れになります。

オブティマイザは、ヒストグラム統計から取得したものよりも範囲オブティマイザ行の見積りを優先します。オブティマイザが範囲オブティマイザが適用されると判断した場合、ヒストグラム統計は使用されません。

インデックス付けされたカラムの場合、インデックス除算を使用して等価比較のために行の見積りを取得できます ([セクション8.2.1.2「rangeの最適化」](#)を参照)。この場合、ヒストグラム統計は必ずしも役に立つとはかぎりません。これは、インデックスの分割によって見積もりが向上するためです。

ヒストグラム統計を使用しても、クエリーの実行が改善されない場合があります (統計が最新でない場合など)。この場合に該当するかどうかを確認するには、[ANALYZE TABLE](#) を使用してヒストグラム統計を再生成し、クエリーを再実行します。

または、ヒストグラム統計を無効にするには、[ANALYZE TABLE](#) を使用して削除します。ヒストグラム統計を無効にする別の方法は、[optimizer_switch](#) システム変数の [condition_fanout_filter](#) フラグをオフにすることです (ただし、他の最適化も無効になる可能性があります)：

```
SET optimizer_switch='condition_fanout_filter=off';
```

ヒストグラム統計が使用されている場合、結果の効果は [EXPLAIN](#) を使用して確認できます。カラム `col1` に使用可能なインデックスがない次のクエリーについて考えてみます：

```
SELECT * FROM t1 WHERE col1 < 24;
```

ヒストグラム統計で、`t1` の行の 57% が `col1 < 24` 述語を満たしていることが示されている場合、インデックスがなくてもフィルタリングが発生し、[EXPLAIN](#) の `filtered` カラムに 57.00 が表示されます。

8.10 バッファリングとキャッシュ

MySQL は、パフォーマンスを向上するため、メモリーバッファに情報をキャッシュするいくつかの戦略を使用します。

8.10.1 InnoDB バッファプールの最適化

InnoDB は、データとインデックスをメモリーにキャッシュするための [バッファプール](#) と呼ばれるストレージ領域を維持しています。InnoDB バッファプールの仕組みを知り、頻繁にアクセスされるデータをメモリーに維持するためにそれを利用することは、MySQL チューニングの重要な側面です。

InnoDB バッファプールの内部動作、LRU 置換アルゴリズムの概要、および一般的な構成情報については、[セクション15.5.1「バッファプール」](#) を参照してください。

その他の InnoDB バッファプールの構成およびチューニング情報については、これらのセクションを参照してください：

- [セクション15.8.3.4「InnoDB バッファプールのプリフェッチ \(先読み\) の構成」](#)
- [セクション15.8.3.5「バッファプールのフラッシュの構成」](#)
- [セクション15.8.3.3「バッファプールをスキャンに耐えられるようにする」](#)
- [セクション15.8.3.2「複数のバッファプールインスタンスの構成」](#)
- [セクション15.8.3.6「バッファプールの状態の保存と復元」](#)
- [セクション15.8.3.1「InnoDB バッファプールサイズの構成」](#)

8.10.2 MyISAM キーキャッシュ

ディスク I/O を最小にするために、[MyISAM](#) ストレージエンジンは多くのデータベース管理システムで使用されている戦略を利用します。それは、もっとも頻繁にアクセスされるテーブルブロックをメモリー内で保持するキャッシュメカニズムを採用しています。

- インデックスブロックの場合、キーキャッシュ (またはキーバッファ) と呼ばれる特別な構造が維持されます。その構造には、もっとも多く使用されるインデックスブロックが置かれる多数のブロックバッファが含まれます。

- データブロックに対しては、MySQL は特別なキャッシュを使用しません。代わりに、ネイティブオペレーティングシステムのファイルシステムキャッシュに依存します。

このセクションではまず **MyISAM** キーキャッシュの基本動作について説明します。次に、キーキャッシュパフォーマンスを向上させる機能と、キャッシュ操作をより適切に制御できるようにする機能について説明します。

- 複数のセッションが同時にキャッシュにアクセスできます。
- 複数のキーキャッシュをセットアップし、特定のキャッシュにテーブルインデックスを割り当てることができず。

キーキャッシュのサイズを制御するには、`key_buffer_size` システム変数を使用します。この変数がゼロに設定されている場合、キーキャッシュは使われません。キーキャッシュは、`key_buffer_size` 値が小さすぎて、最小数のブロックバッファ (8) を割り当てられない場合も使用されません。

キーキャッシュが動作していない場合、インデックスファイルはオペレーティングシステムによって提供されるネイティブファイルシステムバッファリングのみを使用してアクセスされます。(つまり、テーブルインデックスブロックは、テーブルデータブロックに採用されている同じ戦略を使用してアクセスされます。)

インデックスブロックは **MyISAM** インデックスファイルへの連続したアクセスの単位です。通常、インデックスブロックのサイズは、インデックス B ツリーのノードのサイズと等しくなります。(インデックスはディスク上で B ツリーデータ構造を使用して表されます。ツリーの下部にあるノードはリーフノードです。リーフノードの上にあるノードは非リーフノードです。)

キーキャッシュ構造内のすべてのブロックバッファは同じサイズです。このサイズは、テーブルインデックスブロックのサイズと等しいか、大きいか、小さくできます。通常これら 2 つの値のうち的一方は、他方の倍数になります。

いずれかのテーブルインデックスブロックのデータにアクセスする必要がある場合、サーバーはまず、キーキャッシュの何らかのブロックバッファでそれを使用できるかどうかを確認します。そうである場合、サーバーはディスク上ではなく、キーキャッシュ内のデータにアクセスします。つまり、ディスクから読み取ったり、それに書き込んだりするのはなく、キャッシュから読み取ったり、それに書き込んだりします。そうでない場合、サーバーは別のテーブルインデックスブロックを含むキャッシュブロックバッファを選択し、そのデータを必要なテーブルインデックスブロックのコピーで置き換えます。新しいインデックスブロックがキャッシュに入れられるとただちに、インデックスデータにアクセスできます。

置き換えのために選択されているブロックが変更されていた場合、ブロックは「ダーティー」とみなされます。この場合、置き換えられる前に、その内容が取得元のテーブルインデックスにフラッシュされます。

通常サーバーは LRU (Least Recently Used) 戦略に従います。置き換えるブロックを選択する場合、直近で使用されていないインデックスブロックを選択します。この選択を簡単にするため、キーキャッシュモジュールは、使用されたすべてのブロックを特別なリスト (LRU チェーン) に使用時間で順序付けして保持しています。ブロックがアクセスされると、それは直近で使用されたものになり、リストの末尾に置かれます。ブロックを置き換える必要がある場合、リストの先頭にあるブロックが、直近で使用されていないことになり、エビクションの最初の候補になります。

InnoDB ストレージエンジンは、そのバッファプールを管理するためにも LRU アルゴリズムを使用します。 [セクション 15.5.1 「バッファプール」](#) を参照してください。

8.10.2.1 共有キーキャッシュアクセス

スレッドはキーキャッシュバッファに同時にアクセスでき、次の条件に従います。

- 更新中でないバッファは複数のセッションによってアクセスできます。
- 更新中のバッファは、更新が完了するまで、それを使用する必要があるセッションを待機させます。
- 複数のセッションは、互いに干渉しないかぎり (つまり、それらは異なるインデックスブロックを必要とし、そのため、異なるキャッシュブロックが置き換えられるかぎり)、キャッシュブロックの置換を引き起こすリクエストを開始できます。

キーキャッシュへの共有アクセスによって、サーバーのスループットを大幅に向上できます。

8.10.2.2 複合キーキャッシュ

注記

MySQL 8.0 では、複数の MyISAM キーキャッシュを参照するためにここで説明する複合部分構造変数構文は非推奨になりました。

キーキャッシュへの共有アクセスはパフォーマンスを向上させますが、セッション間の競合を完全には排除しません。それらはまだキーキャッシュバッファへのアクセスを管理する制御構造を得るために争います。キーキャッシュアクセスの競合をもっと軽減するために、MySQL は複合キーキャッシュも提供しています。この機能により、異なるキーキャッシュにさまざまなテーブルインデックスを割り当てることができます。

複合キーキャッシュがある場合、サーバーは特定の MyISAM テーブルに対してクエリーを処理する際に、使用するべきキャッシュを知っている必要があります。デフォルトでは、すべての MyISAM テーブルインデックスはデフォルトのキーキャッシュにキャッシュされます。テーブルインデックスを特定のキーキャッシュに割り当てるには、`CACHE INDEX` ステートメントを使用します (セクション 13.7.8.2 「`CACHE INDEX` ステートメント」を参照してください)。たとえば、次のステートメントは `t1`、`t2`、および `t3` テーブルから、`hot_cache` という名前のキーキャッシュにインデックスを割り当てます。

```
mysql> CACHE INDEX t1, t2, t3 IN hot_cache;
+-----+-----+-----+
| Table | Op          | Msg_type | Msg_text |
+-----+-----+-----+
| test.t1 | assign_to_keycache | status | OK |
| test.t2 | assign_to_keycache | status | OK |
| test.t3 | assign_to_keycache | status | OK |
+-----+-----+-----+
```

`CACHE INDEX` ステートメントで参照されているキーキャッシュは、`SET GLOBAL` パラメータ設定ステートメントでそのサイズを設定するか、またはサーバー起動オプションを使用して作成できます。例:

```
mysql> SET GLOBAL keycache1.key_buffer_size=128*1024;
```

キーキャッシュを破棄するには、そのサイズをゼロに設定します。

```
mysql> SET GLOBAL keycache1.key_buffer_size=0;
```

デフォルトの鍵キャッシュは破棄できません。これを実行しようとしても無視されます:

```
mysql> SET GLOBAL key_buffer_size = 0;
mysql> SHOW VARIABLES LIKE 'key_buffer_size';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| key_buffer_size | 8384512 |
+-----+-----+
```

キーキャッシュ変数は名前とコンポーネントのある構造化システム変数です。 `keycache1.key_buffer_size` の場合、`keycache1` はキャッシュ変数名であり、`key_buffer_size` はキャッシュコンポーネントです。構造化キーキャッシュシステム変数を参照するために使用する構文の詳細については、セクション 5.1.9.5 「`構造化システム変数`」を参照してください。

デフォルトで、テーブルインデックスは、サーバー起動時に作成されるメイン (デフォルト) キーキャッシュに割り当てられます。キーキャッシュが破棄されると、それに割り当てられたすべてのインデックスはデフォルトのキーキャッシュに再割り当てされます。

ビジーなサーバーの場合、3 つのキーキャッシュを含む戦略を使用できます。

- すべてのキーキャッシュに割り当てられたスペースの 20% を占める「ホット」キーキャッシュ。これは、検索に頻繁に使用されるが、更新されないテーブルに使用します。
- すべてのキーキャッシュに割り当てられたスペースの 20% を占める「コールド」キーキャッシュ。このキャッシュは、一時テーブルなどの中規模の集中的に変更されるテーブルに使用します。
- キーキャッシュスペースの 60% を占める「ウォーム」キーキャッシュ。これは、デフォルトでほかのすべてのテーブルに使用されるように、デフォルトのキーキャッシュとして使用します。

3つのキーキャッシュを使用することに利点がある理由の1つは、1つのキーキャッシュ構造へのアクセスが、ほかへのアクセスをブロックしないことです。あるキャッシュに割り当てられたテーブルにアクセスするステートメントは、ほかのキャッシュに割り当てられたテーブルにアクセスするステートメントと競合しません。パフォーマンスの向上はほかの理由でも発生します。

- ホットキャッシュはクエリーの取得にのみ使用されるため、その内容が変更されることはありません。その結果、インデックスブロックをディスクから取り出す必要がある場合常に、置き換えのために選択されたキャッシュブロックの内容を最初にフラッシュする必要はありません。
- ホットキャッシュに割り当てられたインデックスの場合、インデックススキャンを必要とするクエリーがなければ、インデックスBツリーの非リーフノードに対応するインデックスブロックがキャッシュに残っている可能性が高くなります。
- 一時テーブルに対して最も頻繁に実行される更新操作は、更新されたノードがキャッシュ内にあり、最初にディスクから読み取る必要がない場合にはるかに高速に実行されます。一時テーブルのインデックスのサイズがコールドキーキャッシュのサイズと同程度である場合、更新されるノードがキャッシュ内にある可能性が高くなります。

CACHE INDEX ステートメントは、テーブルとキーキャッシュ間のアソシエーションをセットアップしますが、そのアソシエーションはサーバーが再起動されるたびに失われます。サーバーが起動するたびにアソシエーションを有効にするには、オプションファイルを使用: キーキャッシュを構成する変数設定と、実行する **CACHE INDEX** ステートメントを含むファイルを指定する `init_file` システム変数を含めます。例:

```
key_buffer_size = 4G
hot_cache.key_buffer_size = 2G
cold_cache.key_buffer_size = 2G
init_file=/path/to/data-directory/mysqld_init.sql
```

サーバーが起動するたびに `mysqld_init.sql` 内のステートメントが実行されます。ファイルには1行に1つずつ SQL ステートメントを含めてください。次の例は `hot_cache` と `cold_cache` に複数のテーブルをそれぞれ割り当てます。

```
CACHE INDEX db1.t1, db1.t2, db2.t3 IN hot_cache
CACHE INDEX db1.t4, db2.t5, db2.t6 IN cold_cache
```

8.10.2.3 ミッドポイント挿入戦略

デフォルトで、キーキャッシュ管理システムは、削除されるキーキャッシュブロックの選択に、単純な LRU 戦略を使用しますが、ミッドポイント挿入戦略というさらに高度な方法もサポートしています。

ミッドポイント挿入戦略を使用すると、LRU チェーンがホットサブリストとウォームサブリストの2つのパートに分割されます。2つのパート間の分割点は固定ではありませんが、キーキャッシュ管理システムでは、ウォームパートが「短くなりすぎ」ず、常にキーキャッシュブロックの少なくとも `key_cache_division_limit` パーセントを含むように配慮されます。`key_cache_division_limit` は構造化キーキャッシュ変数のコンポーネントであるため、その値はキャッシュごとに設定可能なパラメータです。

インデックスブロックがテーブルからキーキャッシュに読み込まれると、それはウォームサブリストの末尾に置かれます。特定の数のヒット (ブロックのアクセス) 後、それはホットサブリストに昇格されます。現在のところ、ブロックを昇格させるために必要なヒット数 (3) はすべてのインデックスブロックで同じです。

ホットサブリストに昇格されるブロックはリストの末尾に置かれます。ブロックはこのサブリスト内で循環されます。ブロックが十分な時間サブリストの先頭にとどまっている場合、それはウォームサブリストに降格されます。この時間はキーキャッシュの `key_cache_age_threshold` コンポーネントの値によって決定されます。

しきい値は、 N ブロックを含むキーキャッシュの場合、最後の $N * \text{key_cache_age_threshold} / 100$ ヒット内にアクセスされないホットサブリストの先頭のブロックが、ウォームサブリストの先頭に移動されることを規定します。置き換えられるブロックは常にウォームサブリストの先頭から取得されるため、その後、それは削除の最初の候補になります。

ミッドポイント挿入戦略により、価値の高いブロックを常にキャッシュ内に保持できます。単純な LRU 戦略を使用したい場合は、`key_cache_division_limit` 値をそのデフォルトの 100 に設定したままにします。

ミッドポイント挿入戦略は、インデックススキャンを必要とするクエリーの実行で、価値の高い高レベル B ツリーノードに対応するすべてのインデックスブロックを、キャッシュから効率的に押し出す際のパフォーマンスの向上に役立ちます。これを回避するには、`key_cache_division_limit` を 100 よりかなり小さい値に設定して、ミッドポイント

挿入戦略を使用する必要があります。これにより、インデックススキャン操作中でも、価値の高い頻繁にヒットされるノードがホットサブリストに保持されます。

8.10.2.4 インデックスプリロード

キーキャッシュ内に、インデックス全体のブロックを保持するために十分なブロックがあるか、または少なくともその非リーフノードに対応するブロックがある場合、使用を開始する前に、キーキャッシュにインデックスブロックをプリロードすることは役立ちます。プリロードにより、インデックスブロックをディスクから順番に読み取ることでもっとも効率的にテーブルインデックスをキーキャッシュバッファに挿入できます。

プリロードしない場合、ブロックは、引き続きクエリーによって必要とされるときに、キーキャッシュに置かれます。ブロックはキャッシュ内に残りますが、すべてのバッファが十分にあるため、ディスクからランダムな順序でフェッチされ、順次フェッチされません。

インデックスをキャッシュにプリロードするには `LOAD INDEX INTO CACHE` ステートメントを使用します。たとえば、次のステートメントはテーブル `t1` および `t2` のインデックスのノード (インデックスブロック) をプリロードします。

```
mysql> LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES;
+-----+-----+-----+-----+
| Table | Op      | Msg_type | Msg_text |
+-----+-----+-----+-----+
| test.t1 | preload_keys | status | OK      |
| test.t2 | preload_keys | status | OK      |
+-----+-----+-----+-----+
```

`IGNORE LEAVES` 修飾子によって、インデックスの非リーフノードのブロックのみがプリロードされます。したがって、上記のステートメントは `t1` からすべてのインデックスブロックをプリロードしますが、`t2` からは非リーフノードのブロックのみをプリロードします。

インデックスが `CACHE INDEX` ステートメントを使用してキーキャッシュに割り当てられている場合、プリロードによって、インデックスブロックがそのキャッシュに置かれます。そうでない場合は、インデックスはデフォルトのキーキャッシュにロードされます。

8.10.2.5 キーキャッシュブロックサイズ

`key_cache_block_size` 変数を使用して、個々のキーキャッシュのブロックバッファのサイズを指定できます。これによって、インデックスファイルの I/O 操作のパフォーマンスをチューニングできます。

I/O 操作の最適なパフォーマンスは、読み取りバッファのサイズがネイティブオペレーティングシステム I/O バッファのサイズに等しい場合に達成されます。ただし、キーノードのサイズを I/O バッファのサイズと等しく設定しても、常に全体の最適なパフォーマンスが確保されるわけではありません。大きなリーフノードを読み取る場合、サーバーは大量の不要なデータを取り出し、事実上ほかのリーフノードの読み取りを妨げます。

MyISAM テーブルの `.MYI` インデックスファイルのブロックのサイズを制御するには、サーバーの起動時に `--myisam-block-size` オプションを使用します。

8.10.2.6 キーキャッシュの再構築

キーキャッシュはそのパラメータ値を更新することで、いつでも再構築できます。例:

```
mysql> SET GLOBAL cold_cache.key_buffer_size=4*1024*1024;
```

`key_buffer_size` または `key_cache_block_size` キーキャッシュコンポーネントにコンポーネントの現在の値とは異なる値を割り当てると、サーバーはキャッシュを破棄し、新しい値に基づいて新しい構造を作成します。キャッシュにダーティーブロックが含まれる場合、サーバーはキャッシュを破棄し、再作成する前にディスクにそれらを保存します。ほかのキーキャッシュパラメータを変更した場合は、再構築が行われません。

キーキャッシュを再構築する場合、サーバーはまずダーティーバッファの内容をディスクにフラッシュします。その後、キャッシュの内容は使用できなくなります。しかし、再構築は、キャッシュに割り当てられたインデックスを使用する必要があるクエリーをブロックしません。代わりに、サーバーはネイティブファイルシステムキャッシュを使用して、テーブルインデックスに直接アクセスします。ファイルシステムキャッシュはキーキャッシュを使用する

ときほど効率的ではないため、クエリーが実行されても速度の低下が予想されます。キャッシュが再構築されると、それに割り当てられたインデックスをキャッシュするためにふたたび使用できるようになり、インデックスのファイルシステムキャッシュの使用は停止されます。

8.10.3 プリペアドステートメントおよびストアドプログラムのキャッシュ

セッション中にクライアントが複数回実行する可能性がある特定のステートメントに対し、サーバーはステートメントを内部構造に変換し、実行時にその構造が使用されるようにキャッシュします。キャッシュによって、セッション中にそれが再度必要になった場合に、ステートメントを再変換するオーバーヘッドが避けられるため、サーバーはより効率的に実行できます。変換とキャッシュは、次のステートメントに対して行われます。

- SQL レベルで処理されるもの (PREPARE ステートメントを使用して) とバイナリクライアント/サーバープロトコルを使用して処理されるもの (mysql_stmt_prepare() C API 関数を使用して) の両方のプリペアドステートメント。max_prepared_stmt_count システム変数は、サーバーがキャッシュするステートメントの合計数を制御します。(すべてのセッションでのプリペアドステートメントの合計数。)
- ストアドプログラム (ストアドプロシージャーおよび関数、トリガー、およびイベント)。この場合、サーバーはプログラム本体全体を変換し、キャッシュします。stored_program_cache システム変数は、サーバーがセッションあたりにキャッシュするストアドプログラムのおおよその数を示します。

サーバーは、セッション単位でプリペアドステートメントおよびストアドプログラム用のキャッシュを保守します。1つのセッションでキャッシュされたステートメントは、ほかのセッションからアクセスできません。セッションが終了すると、サーバーはそのためにキャッシュされたすべてのステートメントを破棄します。

サーバーがキャッシュされた内部ステートメント構造を使用する場合、構造が古くなっていないことに注意する必要があります。ステートメントによって使用されているオブジェクトにメタデータの変更があり、現在のオブジェクト定義と内部ステートメント構造で表されている定義に不一致が発生することがあります。メタデータの変更は、テーブルの作成、削除、変更、名前変更、切り捨てを行う DDL ステートメントや、テーブルの解析、最適化、修復を行う DDL ステートメントなどに対して発生します。テーブルの内容の変更 (INSERT や UPDATE などによる) ではメタデータが変更されず、SELECT ステートメントも変更されません。

次にこの問題を説明します。クライアントがこのステートメントを準備するとします。

```
PREPARE s1 FROM 'SELECT * FROM t1';
```

SELECT * は内部構造からテーブル内のカラムのリストに展開します。テーブル内のカラムのセットが ALTER TABLE によって変更されている場合、プリペアドステートメントが古くなります。クライアントが次回 s1 を実行したときにサーバーがこの変更を検出しない場合、プリペアドステートメントは誤った結果を返します。

プリペアドステートメントによって参照されているテーブルやビューのメタデータの変更による問題を避けるため、サーバーはこれらの変更を検出し、次の実行時にステートメントを自動的に再準備します。つまり、サーバーはステートメントを再解析し、内部構造を再構築します。再解析は、キャッシュ内に新しいエントリのための空きを作るために暗黙的に、または FLUSH TABLES によって明示的に、参照されているテーブルやビューがテーブル定義キャッシュからフラッシュされたあとにも行われます。

同様に、ストアドプログラムによって使用されているオブジェクトに変更が発生した場合、サーバーはプログラム内の影響のあるステートメントを再解析します。

サーバーは式内のオブジェクトのメタデータの変更も検出します。これらは、DECLARE CURSOR などのストアドプログラムに固有のステートメントや IF、CASE、および RETURN などのフロー制御ステートメントで使用できません。

ストアドプログラム全体の再解析を避けるため、サーバーは必要に応じて、プログラム内の影響のあるステートメントや式のみを再解析します。例:

- テーブルまたはビューのメタデータが変更されているとします。再解析は、テーブルやビューにアクセスするプログラム内の SELECT * に対して行われますが、テーブルやビューにアクセスしない SELECT * に対しては行われません。
- ステートメントが影響を受ける場合、サーバーは可能なかぎり部分的にのみそれを再解析します。この CASE ステートメントを考慮します。

```
CASE case_expr
WHEN when_expr1 ...
WHEN when_expr2 ...
WHEN when_expr3 ...
...
END CASE
```

メタデータの変更が `WHEN when_expr3` にのみ影響する場合、その式が再解析されます。`case_expr` およびその他の `WHEN` 式は再解析されません。

再解析では、元の内部形式への変換に有効であったデフォルトのデータベースと SQL モードが使われます。

サーバーは最大 3 回再解析を試みます。すべての試みが失敗した場合、エラーが発生します。

再解析は自動ですが、それが行われた場合、プリパードステートメントとストアードプログラムのパフォーマンスが低下します。

プリパードステートメントの場合、`Com_stmt_reprepare` ステータス変数が再準備の数を追跡します。

8.11 ロック操作の最適化

MySQL は [ロック](#) を使用して、テーブルの内容の競合を管理します。

- 内部ロックは、複数スレッドによるテーブルの内容の競合を管理するために、MySQL サーバー自体の内部で実行されます。この種類のロックは、完全にサーバーによって実行され、ほかのプログラムは関与しないため、内部です。[セクション 8.11.1 「内部ロック方法」](#) を参照してください。
- 外部ロックは、サーバーとほかのプログラム間で、どのプログラムがいつテーブルにアクセスできるかを調整するために、`MyISAM` テーブルファイルをロックする場合に発生します。[セクション 8.11.5 「外部ロック」](#) を参照してください。

8.11.1 内部ロック方法

このセクションでは、内部ロック、つまり複数のセッションによるテーブル内容の競合を管理するために、MySQL サーバー自体の内部で実行されるロックについて説明します。この種類のロックは、完全にサーバーによって実行され、ほかのプログラムは関与しないため、内部です。ほかのプログラムによって MySQL ファイルに対して実行されるロックについては、[セクション 8.11.5 「外部ロック」](#) を参照してください。

- [行レベルロック](#)
- [テーブルレベルロック](#)
- [ロックのタイプの選択](#)

行レベルロック

MySQL は `InnoDB` テーブルに [行レベルロック](#) を使用して、複数のセッションによる同時書き込みアクセスをサポートし、それらを複数ユーザー、高度な並列性、および OLTP アプリケーションに適したものにします。

単一の `InnoDB` テーブルに対して複数の同時書き込み操作を実行する場合に [deadlocks](#) を回避するには、データ変更ステートメントがトランザクションの後半にある場合でも、変更が予想される行のグループごとに `SELECT ... FOR UPDATE` ステートメントを発行して、トランザクションの開始時に必要なロックを取得します。トランザクションで複数のテーブルを変更またはロックする場合、各トランザクション内で、該当するステートメントを同じ順序で発行します。デッドロックは、重大なエラーを表すのではなくパフォーマンスに影響します。これは、`InnoDB` ではデフォルトで `detects` デッドロック状態が自動的に発生し、影響を受けるトランザクションのいずれかがロールバックされるためです。

同時実行性の高いシステムでは、多数のスレッドが同じロックを待機している場合、デッドロック検出によって速度が低下する可能性があります。デッドロック検出を無効にし、デッドロック発生時のトランザクションロールバックの `innodb_lock_wait_timeout` 設定に依存する方が効率的な場合があります。デッドロック検出は、`innodb_deadlock_detect` 構成オプションを使用して無効にできます。

行レベルロックの利点:

- 異なるセッションが異なる行にアクセスする場合、ロックの競合は少なくなります。
- ロールバックする変更が少なくなります。
- 1つの行を長時間ロックできます。

テーブルレベルロック

MySQL では、[MyISAM](#)、[MEMORY](#) および [MERGE](#) テーブルに [table-level locking](#) を使用し、一度に更新できるセッションは 1 つのみです。このロックレベルにより、これらのストレージエンジンは読み取り専用、読み取りの大部分、またはシングルユーザーアプリケーションに適しています。

これらのストレージエンジンは、常にクエリーの最初に 1 回だけ必要なすべてのロックをリクエストし、常に同じ順序でテーブルをロックすることによって、[デッドロック](#)を回避します。トレードオフとは、この戦略によって同時実行性が低下することです。テーブルを変更する他のセッションは、現在のデータ変更ステートメントが終了するまで待機する必要があります。

テーブルレベルロックの利点:

- 必要なメモリーは比較的少なくなります (行ロックでは、ロックされた行または行のグループごとにメモリーが必要です)
- 単一のロックだけが必要であるため、テーブルの大部分に対して使用する場合に高速です。
- データの大部分に対して [GROUP BY](#) 操作を頻繁に実行する場合、またはテーブル全体を頻繁にスキャンする必要がある場合は高速です。

MySQL はテーブル書き込みロックを次のように許可します。

1. テーブルにロックがない場合、それを書き込みロックします。
2. そうでない場合、書き込みロックキューにロックリクエストを入れます。

MySQL はテーブル読み取りロックを次のように許可します。

1. テーブルに書き込みロックがない場合、それを読み取りロックします。
2. そうでない場合、読み取りロックキューにロックリクエストを入れます。

テーブルの更新は、テーブルの取得よりも高い優先度が与えられます。そのため、ロックが解放されると、ロックは書き込みロックキュー内のリクエストに使用できるようになり、次に読み取りロックキュー内のリクエストに使用できるようになります。これにより、テーブルに大量の [SELECT](#) アクティビティがある場合でも、テーブルに対する更新が「starved」ではなくなります。ただし、テーブルの更新が多数ある場合、[SELECT](#) ステートメントは更新がなくなるまで待機します。

読み取りと書き込みの優先度を変更する方法については、[セクション8.11.2「テーブルロックの問題」](#)を参照してください。

[Table_locks_immediate](#) および [Table_locks_waited](#) ステータス変数をチェックすることでシステム上のテーブルロック競合を分析できます。これらは、テーブルロックのリクエストがすぐに許可された回数と待機する必要があった回数を示します。

```
mysql> SHOW STATUS LIKE 'Table%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Table_locks_immediate | 1151552 |
| Table_locks_waited | 15324 |
+-----+-----+
```

パフォーマンススキーマロックテーブルは、ロック情報も提供します。[セクション27.12.13「パフォーマンススキーマロックテーブル」](#)を参照してください。

MyISAM ストレージエンジンでは、特定のテーブルのリーダーとライター間の競合を軽減するために、同時挿入をサポートしています。MyISAM テーブルでデータファイルの途中に空きブロックがない場合、行は常にデータファイルの末尾に挿入されます。この場合、ロックなしで MyISAM テーブルに対して同時 INSERT および SELECT ステートメントを自由に組み合わせることができます。つまり、ほかのクライアントが MyISAM テーブルから読み取ると同時に、それに行を挿入できます。テーブルの途中で行が削除されるか更新されると、隙間が発生します。穴がある場合、同時挿入は無効になりますが、すべての穴が新しいデータでいっぱいになると、再度自動的に有効になります。この動作を制御するには、`concurrent_insert` システム変数を使用します。セクション 8.11.3 「同時挿入」を参照してください。

LOCK TABLES で明示的にテーブルロックを獲得する場合、READ ロックではなく READ LOCAL ロックをリクエストして、テーブルをロックしている間に、ほかのセッションが同時挿入を実行できるようにできます。

同時挿入が不可能な場合に、テーブル `t1` で多くの INSERT および SELECT 操作を実行するには、一時テーブル `temp_t1` に行を挿入し、実際のテーブルを一時テーブルの行で更新します：

```
mysql> LOCK TABLES t1 WRITE, temp_t1 WRITE;
mysql> INSERT INTO t1 SELECT * FROM temp_t1;
mysql> DELETE FROM temp_t1;
mysql> UNLOCK TABLES;
```

ロックのタイプの選択

通常、次の場合、テーブルロックは行レベルロックよりも優れています：

- テーブルに対するほとんどのステートメントが読み取りです。
- テーブルに対するステートメントが読み取りと書き込みの組み合わせであり、そのうち書き込みは 1 つのキーの読み取りでフェッチできる単一の行に対する更新または削除です。

```
UPDATE tbl_name SET column=value WHERE unique_key_col=key_value;
DELETE FROM tbl_name WHERE unique_key_col=key_value;
```

- 同時 INSERT ステートメントとごく少数の UPDATE または DELETE ステートメントと組み合わせられた SELECT。
- ライターを使用しない、テーブル全体への多くのスキャンまたは GROUP BY 操作。

高レベルロックでは、行レベルロックよりロックのオーバーヘッドが少ないため、様々なタイプのロックをサポートすることでアプリケーションをより簡単にチューニングできます。

行レベルロック以外のオプション：

- 複数のリーダーを同時に持つことができるバージョンング (MySQL で同時挿入に使用されるものなど)。つまり、データベースまたはテーブルでは、アクセスの開始時期に応じて異なるデータのビューがサポートされます。この他の一般的な用語は、「時間移動」、「copy on write」または「オンデマンドでコピー」です。
- コピーオンデマンドは、多くの場合、行レベルロックよりも優れています。ただし、最悪の場合は、通常のロックを使用するよりもはるかに多くのメモリーを使用できます。
- 行レベルロックを使用するかわりに、MySQL の GET_LOCK() および RELEASE_LOCK() で提供されるロックなど、アプリケーションレベルのロックを使用できます。これらはアドバイザロックであるため、相互に連携するアプリケーションでのみ機能します。セクション 12.15 「ロック関数」を参照してください。

8.11.2 テーブルロックの問題

InnoDB テーブルでは、複数のセッションとアプリケーションが互いに待機したり、不整合の結果を生成したりすることなく、同じテーブルに対して同時に読み取りや書き込みを実行できるように、行レベルロックを使用します。このストレージエンジンでは、LOCK TABLES ステートメントは特別な保護を提供せず、代わりに並列性が低くなるため、この使用を避けてください。自動の行レベルロックにより、これらのテーブルがもっとも重要なデータを格納するもっともビジーなデータベースに適合し、同時にテーブルのロックやロック解除が必要ないためアプリケーションロジックが簡単になります。その結果、InnoDB ストレージエンジンが MySQL のデフォルトになります。

MySQL は InnoDB を除く、すべてのストレージエンジンに対して、テーブルロック (ページ、行、またはカラムロックの代わりに) を使用します。ロック操作自体には、あまりオーバーヘッドがありません。ただし、一度に 1 つの

セッションしかテーブルに書き込むことができないため、これらのほかのストレージエンジンでの最高のパフォーマンスのため、頻繁にクエリーされ、めったに挿入または更新されないテーブルに対して主にそれらを使用します。

- [InnoDB を優先するパフォーマンスの考慮事項](#)
- [ロックパフォーマンスの問題の回避](#)

InnoDB を優先するパフォーマンスの考慮事項

テーブルを作成するために、[InnoDB](#) を使用するか、別のストレージエンジンを使用するかを選択する場合、テーブルロックの次の短所を考慮してください。

- テーブルロックにより、多くのセッションを同時にテーブルから読み取ることができず、セッションでテーブルに書き込む必要がある場合、まず排他的アクセスを取得する必要がありますが、これはまずほかのセッションがテーブルを処理し終わるのを待つ必要がある可能性があることを意味します。更新中、この特定のテーブルにアクセスしようとするほかのすべてのセッションは、更新が完了するまで待機する必要があります。
- ディスクがいっぱいで、セッションを続行するには空き領域が使用できるようになる必要があるため、セッションが待機している場合にテーブルロックによって問題が発生します。この場合、問題のテーブルにアクセスしようとするすべてのセッションが、より多くのディスク領域が使用できるようになるまで待機状態になります。
- 実行に長時間かかる [SELECT](#) ステートメントにより、その間ほかのセッションのテーブルの更新が妨げられ、ほかのセッションが遅くなり、応答していないように見えます。セッションが更新のためにテーブルへの排他的アクセスを取得するのを待機している間、[SELECT](#) ステートメントを発行する他のセッションはその背後でキューに入れられるため、読取り専用セッションでも同時実行性が低下します。

ロックパフォーマンスの問題の回避

次の項目では、テーブルロックによって発生する競合を回避または軽減するいくつかの方法について説明します。

- セットアップ時に [CREATE TABLE ... ENGINE=INNODB](#) を使用するか、既存のテーブルに対して [ALTER TABLE ... ENGINE=INNODB](#) を使用して、テーブルを [InnoDB](#) ストレージエンジンに切り替えることを考慮します。このストレージエンジンの詳細については、[第15章「InnoDB ストレージエンジン」](#)を参照してください。
- テーブルをロックする時間が短くなるように、[SELECT](#) ステートメントを最適化して、実行を高速化します。これを実行するには、いくつかのサマリーテーブルを作成する必要がある場合があります。
- [--low-priority-updates](#) で [mysqld](#) を起動します。テーブルレベルロックのみを使用するストレージエンジン ([MyISAM](#)、[MEMORY](#)、および [MERGE](#) など) の場合、これにより、テーブルを更新 (変更) するすべてのステートメントに [SELECT](#) ステートメントより低い優先度を与えます。この場合、先述のシナリオの 2 つめの [SELECT](#) ステートメントは [UPDATE](#) ステートメントの前に実行され、最初の [SELECT](#) の終了を待機しません。
- 特定の接続で発行されたすべての更新を低い優先度で実行させるように指定するには、[low_priority_updates](#) サーバースystem変数を 1 に等しく設定します。
- 特定の [INSERT](#)、[UPDATE](#)、または [DELETE](#) ステートメントに低い優先度を与えるには、[LOW_PRIORITY](#) 属性を使用します。
- 特定の [SELECT](#) ステートメントに高い優先度を与えるには、[HIGH_PRIORITY](#) 属性を使用します。[セクション 13.2.10「SELECT ステートメント」](#)を参照してください。
- [max_write_lock_count](#) システム変数に低い値を指定して [mysqld](#) を起動し、テーブルに対する特定の数の書き込みロックが発生した後 (挿入操作など)、テーブルを待機しているすべての [SELECT](#) ステートメントの優先度を MySQL によって一時的に昇格させます。これにより、特定の数の書き込みロックの後に読取りロックが許可されません。
- 組み合わせられた [SELECT](#) と [DELETE](#) ステートメントに問題がある場合、[DELETE](#) への [LIMIT](#) オプションが役立つことがあります。[セクション 13.2.2「DELETE ステートメント」](#)を参照してください。
- [SELECT](#) ステートメントで [SQL_BUFFER_RESULT](#) を使用すると、テーブルロックの時間の短縮に役立つことがあります。[セクション 13.2.10「SELECT ステートメント」](#)を参照してください。
- テーブルの内容を個別のテーブルに分割すると (クエリーを 1 つのテーブルのカラムに対して実行し、更新を別のテーブルのカラムに制限することによって)、役立つことがあります。

- 単一のキューを使用するように、`mysys/thr_lock.c` のロックコードを変更できます。この場合、書き込みロックと読み取りロックは同じ優先度を持ち、一部のアプリケーションに役立つことがあります。

8.11.3 同時挿入

MyISAM ストレージエンジンでは、特定のテーブルに対する読み取りと書き込みの競合を軽減するために、同時挿入をサポートしています。MyISAM テーブルのデータファイルに隙間 (途中の削除された行) がない場合、`SELECT` ステートメントがテーブルの行を読み取るのと同時に、`INSERT` ステートメントを実行してテーブルの末尾に行を追加できます。複数の `INSERT` ステートメントがある場合、それらはキューに入れられ、`SELECT` ステートメントと同時に順番に実行されます。同時 `INSERT` の結果はすぐに見られないことがあります。

`concurrent_insert` システム変数を設定して、同時挿入の処理を変更できます。デフォルトで、変数は `AUTO` (または 1) に設定され、同時挿入が先述のように処理されます。`concurrent_insert` が `NEVER` (または 0) に設定されている場合、同時挿入は無効にされます。変数が `ALWAYS` (または 2) に設定されている場合、行が削除されたテーブルに対してもテーブルの末尾での同時挿入が許可されます。`concurrent_insert` システム変数の説明も参照してください。

バイナリログを使用している場合、同時挿入は `CREATE ... SELECT` または `INSERT ... SELECT` ステートメントの通常の挿入に変換されます。これは、バックアップ操作中にログを適用することでテーブルの正確なコピーを確実に再作成できるようにするために行われます。[セクション5.4.4「バイナリログ」](#)を参照してください。また、これらのステートメントに対しては、選択元のテーブルへの挿入がブロックされるように、そのテーブルに読み取りロックが設定されます。その結果、そのテーブルに対する同時挿入も待機する必要があります。

`LOAD DATA` では、同時挿入の条件を満たす MyISAM テーブルを使用して `CONCURRENT` を指定した場合 (つまり、中央に空きブロックが含まれていない場合)、他のセッションは `LOAD DATA` の実行中にテーブルからデータを取得できません。`CONCURRENT` オプションの使用は、同時にテーブルを使用しているほかのセッションがない場合でも、`LOAD DATA` のパフォーマンスに多少の影響があります。

`HIGH_PRIORITY` を指定すると、サーバーが `--low-priority-updates` オプションで起動されている場合に、その効果がオーバーライドされます。また、同時挿入も使用されなくなります。

`LOCK TABLE` の場合、`READ LOCAL` と `READ` の違いは `READ LOCAL` が、ロックが保持されている間に、競合していない `INSERT` ステートメント (同時挿入) の実行を許可することです。ただし、ロックを保持している間にサーバーの外部のプロセスを使用してデータベースを操作する場合、これを使用することはできません。

8.11.4 メタデータのロック

MySQL では、メタデータロックを使用して、データベースオブジェクトへの同時アクセスを管理し、データの一貫性を確保します。メタデータのロックは、テーブルのみでなく、スキーマ、ストアードプログラム (プロシージャ、ファンクション、トリガー、スケジューライベント)、テーブルスペース、`GET_LOCK()` 関数で取得されたユーザーロック ([セクション12.15「ロック関数」](#)を参照)、および [セクション5.6.8.1「ロックサービス」](#) で説明されているロックサービスで取得されたロックにも適用されます。

パフォーマンススキーマ `metadata_locks` テーブルは、メタデータロック情報を公開します。この情報は、ロックを保持しているセッションや、ロックを待機してブロックされているセッションなどを確認する場合に役立ちます。詳細は、[セクション27.12.13.3「metadata_locks テーブル」](#)を参照してください。

メタデータロックには多少のオーバーヘッドが伴い、クエリボリュームが増加するにつれて増加します。複数のクエリが同じオブジェクトにアクセスを試みるが多くなるほど、メタデータの競合が増加します。

メタデータのロックは、テーブル定義キャッシュの代替ではなく、その相互排他ロックとロックは、`LOCK_open` 相互排他ロックと異なります。次の説明では、メタデータのロックの仕組みに関する情報を提供します。

- [メタデータロックの取得](#)
- [メタデータロックの解放](#)

メタデータロックの取得

特定のロックに複数の待機者がいる場合は、`max_write_lock_count` システム変数に関連する例外を除いて、優先度の高いロックリクエストが最初に満たされます。書き込みロック要求の優先順位は、読み取りロック要求よりも高くなります。ただし、`max_write_lock_count` がある程度低い値 (たとえば、10) に設定されている場合、読み取りロック要

求がすでに 10 個の書き込みロック要求を優先して渡されていれば、保留中の書き込みロック要求よりも読み取りロック要求が優先されることがあります。通常、`max_write_lock_count` のデフォルト値は非常に大きいため、この動作は発生しません。

ステートメントは、同時にではなくメタデータロックを 1 つずつ取得し、プロセスでデッドロック検出を実行します。

DML ステートメントは、通常、ステートメントで記述されているテーブルの順序でロックを取得します。

DDL ステートメント、`LOCK TABLES` およびその他の類似するステートメントは、明示的に指定されたテーブルに対するロックを名前順に取得することで、同時 DDL ステートメント間のデッドロックの可能性のある数を減らします。暗黙的に使用されるテーブル (ロックする必要がある外部キー関係のテーブルなど) では、ロックが異なる順序で取得される場合があります。

たとえば、`RENAME TABLE` は、名前順にロックを取得する DDL ステートメントです:

- 次の `RENAME TABLE` ステートメントは、`tbla` の名前を他の名前に変更し、`tblc` の名前を `tbla` に変更します:

```
RENAME TABLE tbla TO tbld, tblc TO tbla;
```

このステートメントは、`tbla`、`tblc` および `tbld` でメタデータロックを順番に取得します (`tbld` は名前順に `tblc` に従うため):

- この若干異なるステートメントによって、`tbla` の名前が他の名前に変更され、`tblc` の名前が `tbla` に変更されます:

```
RENAME TABLE tbla TO tblb, tblc TO tbla;
```

この場合、ステートメントは `tbla`、`tblb` および `tblc` でメタデータロックを順番に取得します (`tblb` は `tblc` の前に名前順に付くため):

どちらのステートメントも、`tbla` および `tblc` のロックをこの順序で取得しますが、残りのテーブル名のロックが `tblc` の前と後のどちらで取得されるかが異なります。

メタデータロックの取得順序は、次の例に示すように、複数のトランザクションが同時に実行される場合に操作結果に違いが生じる可能性があります。

同じ構造を持つ 2 つのテーブル `x` および `x_new` から開始します。次の 3 つのクライアントが、これらのテーブルを含むステートメントを発行します:

クライアント 1:

```
LOCK TABLE x WRITE, x_new WRITE;
```

このステートメントは、`x` および `x_new` で名前順に書き込みロックを要求および取得します。

クライアント 2:

```
INSERT INTO x VALUES(1);
```

ステートメントは、`x` で書き込みロックを待機していることをリクエストおよびブロックします。

クライアント 3:

```
RENAME TABLE x TO x_old, x_new TO x;
```

このステートメントは、`x`、`x_new` および `x_old` で排他ロックを名前順に要求しますが、`x` でのロックの待機をブロックします。

クライアント 1:

```
UNLOCK TABLES;
```

このステートメントは、`x` および `x_new` の書き込みロックを解放します。クライアント 3 による `x` の排他ロックリクエストは、クライアント 2 による書き込みロックリクエストよりも優先度が高いため、クライアント 3 は `x` でロックを取

得し、`x_new` および `x_old` でも名前変更を実行してロックを解放します。次に、クライアント 2 は `x` でロックを取得し、挿入を実行してロックを解放します。

ロック取得順序により、`INSERT` の前に `RENAME TABLE` が実行されます。挿入が行われる `x` は、クライアント 2 が挿入を発行し、クライアント 3 によって `x` に名前が変更されたときに `x_new` という名前のテーブルです：

```
mysql> SELECT * FROM x;
+----+
|i |
+----+
| 1|
+----+

mysql> SELECT * FROM x_old;
Empty set (0.01 sec)
```

かわりに、同じ構造を持つ `x` および `new_x` という名前のテーブルから始めます。ここでも、3 つのクライアントが、次のテーブルを含むステートメントを発行します：

クライアント 1:

```
LOCK TABLE x WRITE, new_x WRITE;
```

このステートメントは、`new_x` および `x` で名前順に書き込みロックを要求および取得します。

クライアント 2:

```
INSERT INTO x VALUES(1);
```

ステートメントは、`x` で書き込みロックを待機していることをリクエストおよびブロックします。

クライアント 3:

```
RENAME TABLE x TO old_x, new_x TO x;
```

このステートメントは、`new_x`、`old_x` および `x` で排他ロックを名前順に要求しますが、`new_x` でのロックの待機をブロックします。

クライアント 1:

```
UNLOCK TABLES;
```

このステートメントは、`x` および `new_x` の書き込みロックを解放します。`x` の場合、保留中のリクエストはクライアント 2 のみであるため、クライアント 2 はロックを取得し、挿入を実行してロックを解放します。`new_x` の場合、保留中のリクエストはクライアント 3 のみで、クライアント 3 はそのロックを取得できます (`old_x` でのロックも取得できます)。名前変更操作は、クライアント 2 の挿入が終了してロックを解放するまで、`x` でのロックに対してブロックされます。次に、クライアント 3 は `x` でロックを取得し、名前変更を実行してロックを解放します。

この場合、ロック取得順序により、`RENAME TABLE` の前に `INSERT` が実行されます。挿入先の `x` は元の `x` で、名前変更操作によって `old_x` に名前が変更されました：

```
mysql> SELECT * FROM x;
Empty set (0.01 sec)

mysql> SELECT * FROM old_x;
+----+
|i |
+----+
| 1|
+----+
```

前述の例のように、同時ステートメントでのロック取得の順序によって操作結果のアプリケーションが異なる場合は、ロック取得の順序に影響を与えるようにテーブル名を調整できます。

メタデータロックは、必要に応じて外部キー制約によって関連付けられたテーブルに拡張され、DML 操作と DDL 操作の競合が関連するテーブルで同時に実行されないようにします。親テーブルを更新すると、外部キーメタデータの更新中に子テーブルのメタデータロックが取得されます。外部キーメタデータは子テーブルによって所有されます。

メタデータロックの解放

トランザクションのシリアライズビリティを確保するため、サーバーは、別のセッションで、未完了の明示的または暗黙的に開始されたトランザクションで使用されているテーブルに対して、セッションがデータ定義言語 (DDL) ステートメントを実行することを許可してはいけません。サーバーは、トランザクション内で使用されているテーブルに対してメタデータロックを獲得し、トランザクションが終了するまでそれらのロックの解放を延期させることによって、これを実現します。テーブルへのメタデータロックは、テーブルの構造への変更を妨げます。このロックアプローチには、あるセッション内のトランザクションによって使用されているテーブルは、トランザクションが終了するまで、ほかのセッションによって DDL ステートメントで使用できないという問題があります。

この原則は、トランザクションテーブルだけでなく、非トランザクションテーブルにも適用されます。あるセッションがトランザクションテーブル `t` と非トランザクションテーブル `nt` を次のように使用するトランザクションを開始するとします。

```
START TRANSACTION;
SELECT * FROM t;
SELECT * FROM nt;
```

サーバーはトランザクションが終了するまで、`t` と `nt` の両方に対するメタデータロックを保持します。別のセッションがいずれかのテーブルに対して、DDL または書き込みロック操作を試みると、それはトランザクションの終了時にメタデータロックが解放されるまでブロックされます。たとえば、2 つめのセッションはこれらのいずれかの操作を試みるとブロックされます。

```
DROP TABLE t;
ALTER TABLE t ...;
DROP TABLE nt;
ALTER TABLE nt ...;
LOCK TABLE t ... WRITE;
```

`LOCK TABLES ... READ` にも同じ動作が適用されます。つまり、テーブル (トランザクションまたは非トランザクション) ブロックを更新し、そのテーブルに対して `LOCK TABLES ... READ` によってブロックされる明示的または暗黙的に開始されたトランザクションです。

サーバーが構文上有効であるが、実行中に失敗するステートメントのメタデータロックを獲得した場合、そのロックを早期に解放しません。失敗したステートメントがバイナリログに書き込まれ、ロックによってログの一貫性が保護されるため、ロックの解放はまだトランザクションの終了まで延期されます。

自動コミットモードでは、各ステートメントが事実上完全なトランザクションであるため、そのステートメントに対して獲得されたメタデータロックは、ステートメントの終了までしか保持されません。

`PREPARE` ステートメント中に獲得されたメタデータロックは、準備が複数ステートメントトランザクション内で行われる場合でも、ステートメントが準備されると解放されます。

MySQL 8.0.13 では、`PREPARED` 状態の XA トランザクションの場合、`XA COMMIT` または `XA ROLLBACK` が実行されるまで、クライアントの接続が切断され、サーバーが再起動されてもメタデータロックは維持されます。

8.11.5 外部ロック

外部ロックは、複数のプロセスによる `MyISAM` データベーステーブルの競合を管理するためのファイルシステムロックの使用です。外部ロックは、MySQL サーバーなどの単一のプロセスが、テーブルへのアクセスを必要とする唯一のプロセスであると想定できない状況で使用されます。次にいくつかの例を示します。

- 同じデータベースディレクトリを使用する複数のサーバーを実行する場合 (推奨されません)、各サーバーで外部ロックが有効になっている必要があります。
- `myisamchk` を使用して `MyISAM` テーブルに対して保守操作を実行する場合、サーバーが実行中でないことを確認するか、サーバーが必要に応じてテーブルファイルをロックし、テーブルへのアクセスを `myisamchk` によって調整するように、サーバーで外部ロックが有効になっていることを確認する必要があります。同じことが、`MyISAM` テーブルをバックアップするために `myisampack` を使用する場合にも当てはまります。

外部ロックを有効にしてサーバーを実行する場合、テーブルのチェックなどの読み取り操作のために、いつでも `myisamchk` を使用できます。この場合、`myisamchk` が使用しているテーブルをサーバーが更新しようとするとき、サーバーは `myisamchk` が終了するまで待機してから続行します。

テーブルの修復や最適化などの書き込み操作のために `myisamchk` を使用する場合、または `myisampack` を使用してテーブルをバックアップする場合は、`mysqld` サーバーがそのテーブルを使用していないことを常に確認する必要があります。`mysqld` を停止しない場合は、少なくとも `mysqldadmin flush-tables` を実行してから `myisamchk` を実行します。サーバーと `myisamchk` が同時にテーブルにアクセスすると、テーブルが破損する可能性があります。

外部ロックが有効になっていると、テーブルへのアクセスを必要とする各プロセスは、テーブルへのアクセスに進む前にテーブルファイルに対するファイルシステムロックを獲得します。必要なすべてのロックを獲得できない場合、(現在ロックを保持しているプロセスがそれらを解放したあとに) ロックを取得できるまで、プロセスはテーブルへのアクセスをブロックされます。

サーバーは場合によってはテーブルにアクセスできるまでほかのプロセスを待機する必要があるため、外部ロックはサーバーのパフォーマンスに影響します。

単一のサーバーを実行して特定のデータディレクトリにアクセスする場合 (これは通常のケースです) およびサーバーの実行中に `myisamchk` などのほかのプログラムでテーブルを変更する必要がない場合、外部ロックは不要です。ほかのプログラムでテーブルを読み取るだけである場合、外部ロックは不要ですが、`myisamchk` がテーブルを読み取っている間にサーバーがテーブルを変更すると、`myisamchk` が警告をレポートすることがあります。

外部ロックが無効になっていて、`myisamchk` を使用するには、`myisamchk` の実行中にサーバーを停止するか、`myisamchk` を実行する前にテーブルをロックし、フラッシュする必要があります。この要件を回避するには、`CHECK TABLE` および `REPAIR TABLE` ステートメントを使用して、`MyISAM` テーブルをチェックし、修復します。

`mysqld` の場合、外部ロックは `skip_external_locking` システム変数の値で制御されます。この変数が有効にされている場合、外部ロックは無効になり、その逆も同じです。外部ロックはデフォルトで無効になっています。

外部ロックの使用は、サーバーの起動時に `--external-locking` または `--skip-external-locking` オプションを使用して制御できます。

外部ロックオプションを使用して多数の MySQL プロセスから `MyISAM` テーブルへの更新を有効にする場合は、`delay_key_write` システム変数を `ALL` に設定してサーバーを起動したり、共有テーブルに対して `DELAY_KEY_WRITE=1` テーブルオプションを使用しないでください。そうでないと、インデックスが破損する可能性があります。

この条件を満たす最も簡単な方法は、常に `--external-locking` を `--delay-key-write=OFF` とともに使用することです。(これは、多くのセットアップで、前述のオプションを組み合わせることが有用であるため、デフォルトで実行されません。)

8.12 MySQL サーバーの最適化

このセクションでは、主に SQL ステートメントのチューニングではなくシステム構成を扱うデータベースサーバーの最適化技法について説明します。このセクションの情報は、管理しているサーバー全体のパフォーマンスとスケーラビリティを確保したい DBA、データベースのセットアップを含むインストールスクリプトを構築する開発者、および生産性を最大にしたいと考え、開発、テストなどのために自分自身で MySQL を実行しているユーザーに適しています。

8.12.1 ディスク I/O の最適化

このセクションでは、より高速なストレージハードウェアをデータベースサーバー専用に行ける場合に、ストレージデバイスを構成する方法について説明します。I/O のパフォーマンスを向上させるための `InnoDB` 構成の最適化の詳細は、[セクション 8.5.8 「InnoDB ディスク I/O の最適化」](#) を参照してください。

- ディスクシークはパフォーマンスの大きなボトルネックです。この問題は、データの量が、効果的なキャッシュが実行不能になるほど大きくなり始めると、明確になります。多かれ少なかれランダムにデータにアクセスする大きなデータベースの場合、読み取りには最低 1 回、書き込みには 2 回のディスクシークが確実に必要になります。この問題を最小にするには、少ないシーク回数でディスクを使用します。
- さまざまなディスクにファイルをシンボリックリンクするか、ディスクストライピングを行なって、使用可能なディスクスピンドル数を増やします (およびそれによってシークのオーバーヘッドを軽減します)。

- シンボリックリンクの使用

これは、MyISAM テーブルの場合、データディレクトリ内の通常の場合から別のディスクへのインデックスファイルやデータファイルのシンボリックリンクを作成する(ストライピングされることもある)ことを意味します。ディスクがほかの目的にも使用されていないものとして、これによって、シークと読み取り時間がともに改善されます。セクション8.12.2「シンボリックリンクの使用」を参照してください。

シンボリックリンクは、InnoDB テーブルでの使用はサポートされていません。ただし、InnoDB データおよびログファイルを異なる物理ディスクに配置することは可能です。詳細は、セクション8.5.8「InnoDB ディスク I/O の最適化」を参照してください。

- ストライピング

ストライピングは、多数のディスクがあり、最初のブロックを最初のディスクに、2番目のブロックを2番目のディスクに、 N 番目のブロックを $(N \text{ MOD } \text{number_of_disks})$ 番目のディスクにというように配置することを意味します。つまり、通常のデータサイズがストライプサイズより小さい(または完全に一致している)場合に、パフォーマンスが大幅に向上します。ストライピングはオペレーティングシステムとストライプサイズに大きく依存するため、さまざまなストライプサイズでアプリケーションのベンチマークを行なってください。セクション8.13.2「独自のベンチマークの使用」を参照してください。

ストライピングの速度の違いは、パラメータに大きく依存します。ストライピングパラメータの設定方法とディスク数によって、桁違いの差が測定されることがあります。ランダムアクセスに対する最適化が順次アクセスに対する最適化かを選択する必要があります。

- 信頼性のため、RAID 0+1(ストライピングとミラーリング)を使用したいと考える場合がありますが、この場合、 N 個のドライブのデータを保持するために $2 \times N$ 個のドライブが必要です。これは、そのための資金がある場合に最適なオプションである可能性があります。ただし、それを効率的に処理するために、何らかのボリューム管理ソフトウェアに投資する必要がある場合もあります。
- 適切なオプションは、ある種類のデータがどのくらい重要であるかに応じて RAID レベルを変えることです。たとえば、再生成が可能なやや重要なデータは RAID 0 ディスクに格納しますが、ホスト情報やログなどの本当に重要なデータは、RAID 0+1 または RAID N ディスクに格納します。RAID N は、パリティビットの更新に必要な時間のため、多くの書き込みがある場合に問題になる可能性があります。
- データベースが使用するファイルシステムのパラメータを設定することもできます。

ファイルに最後にアクセスされたタイミングを知る必要がない(実際にデータベースサーバーで役立つ)場合、`-o noatime` オプションを使用してファイルシステムをマウントできます。それは、ファイルシステム上のiノードの最終アクセス時間への更新をスキップするため、一部のディスクシークが避けられます。

多くのオペレーティングシステムで、`-o async` オプションを使用してファイルシステムをマウントすることによって、ファイルシステムが非同期に更新されるように設定できます。コンピュータが適度に安定している場合、これにより、それほど信頼性を犠牲にすることなく、パフォーマンスが向上するはずですが。(Linux ではこのフラグがデフォルトでオンにされています。)

MySQL での NFS の使用

MySQL で NFS を使用するかどうかを検討する場合は注意が必要です。オペレーティングシステムおよび NFS バージョンによって異なる潜在的な問題には、次のものがあります:

- NFS ボリュームに配置された MySQL データおよびログファイルはロックされ、使用できなくなります。ロックの問題は、MySQL の複数のインスタンスが同じデータディレクトリにアクセスする場合や、停電などが原因で MySQL が適切に停止されない場合に発生することがあります。NFS version 4 は、アドバイザおよびリースベースのロックの導入に関する基本的なロックの問題に対処します。ただし、MySQL インスタンス間でデータディレクトリを共有することはお勧めしません。
- 誤った順序で受信したメッセージまたはネットワークトラフィックが失われたため、データの不整合が発生しました。この問題を回避するには、TCP with `hard` および `intr` マウントオプションを使用します。
- 最大ファイルサイズの制限。NFS Version 2 クライアントは、最低 2GB のファイル (signed 32 bit offset) にのみアクセスできます。NFS Version 3 クライアントは、より大きなファイル (最大 64 ビットオフセット) をサポートします。サポートされる最大ファイルサイズは、NFS サーバーのローカルファイルシステムによっても異なります。

プロフェッショナル SAN 環境またはその他のストレージシステム内で NFS を使用すると、このような環境外で NFS を使用するよりも高い信頼性が得られる傾向があります。ただし、SAN 環境内の NFS は、直接接続されたストレージまたはバス接続された非ローテーションストレージよりも低速になる可能性があります。

NFS を使用する場合は、本番環境に配備する前に NFS 設定を徹底的にテストするため、NFS Version 4 以降をお勧めします。

8.12.2 シンボリックリンクの使用

データベースやテーブルをデータベースディレクトリからほかの場所に移動して、それらを新しい場所へのシンボリックリンクに置き換えることができます。これを実行したいと考える可能性があるのは、たとえば、データベースを空き領域の多いファイルシステムに移動する場合や、テーブルを別のディスクに分散させてシステムの速度を高める場合です。

InnoDB テーブルの場合は、[セクション15.6.1.2「外部でのテーブルの作成」](#)で説明されているように、シンボリックリンクのかわりに `CREATE TABLE` ステートメントの `DATA DIRECTORY` 句を使用します。この新機能は、サポートされるクロスプラットフォーム技法です。

これを実行する推奨される方法は、データベースディレクトリ全体の別のディスクへのシンボリックリンクを作成することです。MyISAM テーブルのシンボリックリンク作成は最後の手段として行います。

データディレクトリの場所を特定するには、次のステートメントを使用します。

```
SHOW VARIABLES LIKE 'datadir';
```

8.12.2.1 Unix 上のデータベースへのシンボリックリンクの使用

Unix では、次の手順を使用してデータベースを symlink します:

1. `CREATE DATABASE` を使用してデータベースを作成します:

```
mysql> CREATE DATABASE mydb1;
```

`CREATE DATABASE` を使用すると、MySQL データディレクトリにデータベースが作成され、サーバーはデータベースディレクトリに関する情報でデータディレクトリを更新できます。

2. サーバーを停止して、移動中に新しいデータベースでアクティビティが発生しないようにします。
3. 空き領域があるディスクにデータベースディレクトリを移動します。たとえば、`tar` または `mv` を使用します。データベースディレクトリを移動するのではなく、コピーする方法を使用する場合は、コピー後に元のデータベースディレクトリを削除します。
4. 移動したデータベースディレクトリへのソフトリンクをデータディレクトリに作成します:

```
shell> ln -s /path/to/mydb1 /path/to/datadir
```

このコマンドは、`mydb1` というシンボリックリンクをデータディレクトリに作成します。

5. サーバーを再起動します。

8.12.2.2 Unix 上の MyISAM へのシンボリックリンクの使用

注記

ここで説明するシンボリックリンクのサポートと、それを制御する `--symbolic-links` オプションは非推奨になりました。これらは将来のバージョンの MySQL で削除される予定です。また、このオプションはデフォルトで無効になっています。

シンボリックリンクは、MyISAM テーブルに対してのみ完全にサポートされています。ほかのストレージエンジンのテーブルで使用されているファイルの場合、シンボリックリンクを使用しようとすると、未知の問題が発生することがあります。InnoDB テーブルの場合は、代わりに[セクション15.6.1.2「外部でのテーブルの作成」](#)に説明する代替の技法を使用します。

完全に動作する `realpath()` 呼び出しがないシステムでは、テーブルのシンボリックリンクを作成しないでください。(Linux と Solaris では `realpath()` をサポートしています)。システムでシンボリックリンクをサポートしているかどうかを判断するには、次のステートメントを使用して、`have_symlink` システム変数の値をチェックします。

```
SHOW VARIABLES LIKE 'have_symlink';
```

MyISAM テーブルのシンボリックリンクの処理は次のように機能します。

- データディレクトリには、常にデータ (`.MYD`) ファイルとインデックス (`.MYI`) ファイルがあります。データファイルとインデックスファイルは、ほかの場所に移動し、データディレクトリ内でシンボリックリンクによって置き換えることができます。
- データファイルとインデックスファイルは、独立して別々のディレクトリへのシンボリックリンクを作成できます。
- 実行中の MySQL サーバーにシンボリックリンクの作成を実行するように指示するには、`CREATE TABLE` に `DATA DIRECTORY` および `INDEX DIRECTORY` オプションを使用します。セクション 13.1.20 「`CREATE TABLE` ステートメント」を参照してください。または、`mysqld` が実行中でない場合は、コマンド行から `ln -s` を使用して、シンボリックリンクの作成を手動で実行できます。

注記

`DATA DIRECTORY` および `INDEX DIRECTORY` オプションのいずれか、または両方で使用されるパスには、MySQL `data` ディレクトリを含めることができません。(Bug #32167)

- `myisamchk` が、シンボリックリンクをデータファイルやインデックスファイルに置き換えません。それは、シンボリックリンクが指しているファイルに対して直接作用します。一時ファイルはすべてデータファイルやインデックスファイルが配置されているディレクトリに作成されます。同じことが `ALTER TABLE`、`OPTIMIZE TABLE`、および `REPAIR TABLE` ステートメントにも当てはまります。

注記

シンボリックリンクを使用しているテーブルを削除すると、シンボリックリンクとシンボリックリンクが指しているファイルの両方が削除されます。これは、`mysqld` を `root` オペレーティングシステムユーザーとして実行しないか、オペレーティングシステムユーザーに MySQL データベースディレクトリへの書き込みアクセス権を付与しないことが非常によい理由です。

- `ALTER TABLE ... RENAME` または `RENAME TABLE` を使用してテーブルの名前を変更し、テーブルを別のデータベースに移動しない場合、データベースディレクトリのシンボリックリンクの名前が新しい名前に変更され、データファイルとインデックスファイルもそれによって名前が変更されます。
- `ALTER TABLE ... RENAME` または `RENAME TABLE` を使用してテーブルを別のデータベースに移動すると、テーブルが別のデータベースディレクトリに移動されます。テーブル名が変更された場合、新しいデータベースディレクトリ内のシンボリックリンクの名前が新しい名前に変更され、データファイルとインデックスファイルもそれによって名前が変更されます。
- シンボリックリンクを使用していない場合、`--skip-symbolic-links` オプションを付けて `mysqld` を起動し、だれも `mysqld` を使用して、データディレクトリ外のファイルを削除したり名前を変更したりできないようにします。

これらのテーブルシンボリックリンクの操作はサポートされていません。

- `ALTER TABLE` は `DATA DIRECTORY` および `INDEX DIRECTORY` テーブルオプションを無視します。

8.12.2.3 Windows 上のデータベースへのシンボリックリンクの使用

Windows では、データベースディレクトリにシンボリックリンクを使用できます。これにより、データベースディレクトリへのシンボリックリンクを設定して、それを別の場所 (別のディスク上など) に置くことができます。Windows でのデータベースシンボリックリンクの使用は、Unix でのそれらの使用に似ていますが、リンクのセットアップの手順は異なります。

`mydb` というデータベースのデータベースディレクトリを `D:\data\mydb` に配置したいとします。これを実行するには、MySQL データディレクトリ内に `D:\data\mydb` を指すシンボリックリンクを作成します。ただし、シンボリックリンクを作成する前に、必要に応じて `D:\data\mydb` ディレクトリを作成して、それが存在することを確認します。データディレクトリ内に `mydb` というデータベースディレクトリがすでにある場合は、それを `D:\data` に移動しま

す。それ以外の場合、シンボリックリンクは無効です。問題を避けるために、データベースディレクトリの移動時にサーバーが実行していないことを確認してください。

Windows では、`mklink` コマンドを使用してシンボリックリンクを作成できます。このコマンドには管理者権限が必要です。

1. 場所をデータディレクトリ内に変更します。

```
C:\> cd \path\to\datadir
```

2. データディレクトリで、データベースディレクトリの場所を指す `mydb` というシンボリックリンクを作成します。

```
C:\> mklink /d mydb D:\data\mydb
```

このあと、データベース `mydb` に作成されるすべてのテーブルが `D:\data\mydb` に作成されます。

8.12.3 メモリーの使用の最適化

8.12.3.1 MySQL のメモリーの使用方法

MySQL はバッファおよびキャッシュを割り当て、データベース操作のパフォーマンスを向上させます。デフォルトの構成は、RAM が約 512MB の仮想マシンで MySQL サーバーを起動できるように設計されています。特定のキャッシュおよびバッファ関連のシステム変数の値を増やすことで、MySQL のパフォーマンスを向上できます。メモリーが制限されたシステムで MySQL を実行するように、デフォルトの構成を変更することもできます。

次のリストでは、MySQL がメモリーを使用する方法をいくつか説明します。該当する場合は、関連するシステム変数が参照されます。ストレージエンジンまたは機能固有の項目もあります。

- **InnoDB** バッファプールは、テーブル、インデックスおよびその他の補助バッファのキャッシュされた **InnoDB** データを保持するメモリー領域です。大容量読み取り操作の効率を高めるため、バッファプールは複数行を保持できる **ページ** に分割されます。キャッシュ管理の効率のために、バッファプールはページのリンクリストとして実装されます。まれにしか使用されないデータは、**LRU** アルゴリズムのバリエーションを使用してキャッシュからエージアウトされます。詳細は、[セクション15.5.1「バッファプール」](#)を参照してください。

バッファプールのサイズは、システムパフォーマンスのために重要です:

- **InnoDB** は、`malloc()` 操作を使用して、サーバー起動時にバッファプール全体にメモリーを割り当てます。`innodb_buffer_pool_size` システム変数は、バッファプールサイズを定義します。通常、推奨される `innodb_buffer_pool_size` 値は、システムメモリーの 50 から 75% です。`innodb_buffer_pool_size` は、サーバーの実行中に動的に構成できます。詳細は、[セクション15.8.3.1「InnoDB バッファプールサイズの構成」](#)を参照してください。
- メモリーが大量にあるシステムでは、バッファプールを複数の **buffer pool instances** に分割することで同時実行性を向上させることができます。`innodb_buffer_pool_instances` システム変数は、バッファプールインスタンスの数を定義します。
- バッファプールが小さすぎると、ページがバッファプールからフラッシュされるときに、後で再度必要になるだけであるため、過剰なチャージングが発生する可能性があります。
- バッファプールが大きすぎると、メモリーの競合が原因でスワッピングが発生する場合があります。
- ストレージエンジンインタフェースを使用すると、**オプティマイザ**は、オプティマイザが複数の行を読み取る可能性が高いスキャンに使用されるレコードバッファのサイズに関する情報を提供できます。バッファサイズは、見積りのサイズによって異なる場合があります。**InnoDB** では、この可変サイズのバッファリング機能を使用して、行プリフェッチを利用し、ラッチおよび B ツリーナビゲーションのオーバーヘッドを削減します。
- すべてのスレッドが **MyISAM** キーバッファを共有します。`key_buffer_size` システム変数によってサイズが決まります。

サーバーが開く **MyISAM** テーブルごとに、インデックスファイルが一度開かれます。テーブルにアクセスする同時に行われているスレッドごとに、データファイルが一度開かれます。同時スレッドごとに、テーブル構造、各カラムのカラム構造、およびサイズ $3 * N$ のバッファが割り当てられます (ここで N は最大行長で、**BLOB** カラムをカウントしていません)。**BLOB** カラムには、5 から 8 バイト + **BLOB** データの長さが必要です。**MyISAM** ストレージエンジンは、内部使用のため 1 つ余分な行バッファを保持します。

- `mysam_use_mmap` システム変数を 1 に設定して、すべての MyISAM テーブルのメモリーマッピングを有効にできます。
- 内部インメモリー一時テーブルが大きすぎる場合 (`tmp_table_size` および `max_heap_table_size` システム変数を使用して決定)、MySQL はテーブルをインメモリーからディスク上の形式に自動的に変換します。MySQL 8.0.16 の時点では、ディスク上の一時テーブルは常に InnoDB ストレージエンジンを使用します。(以前は、この目的に使用されていたストレージエンジンは、サポートされなくなった `internal_tmp_disk_storage_engine` システム変数によって決定されていました。) [セクション8.4.4 「MySQL での内部一時テーブルの使用」](#) に説明するように、許可される一時テーブルのサイズを増やすことができます。

`CREATE TABLE` を使用して明示的に作成された MEMORY テーブルの場合、`max_heap_table_size` システム変数のみがテーブルの大きさを決定し、ディスク上の形式への変換はありません。

- [MySQL Performance Schema](#) は、MySQL サーバーの実行を低レベルで監視するための機能です。パフォーマンススキーマは、サーバーの起動時に必要なメモリーを割り当てるのではなく、メモリー使用量を実際のサーバー負荷に合わせて増分的に割り当てます。割り当てられたメモリーは、サーバーが再起動されるまで解放されません。詳細は、[セクション27.17 「パフォーマンススキーマのメモリー割り当てモデル」](#) を参照してください。
- サーバーがクライアント接続の管理に使用する各スレッドには、スレッド固有の領域が必要です。次のリストは、これらの変数とそのサイズを制御するシステム変数を示しています:
 - スタック (`thread_stack`)
 - 接続バッファ (`net_buffer_length`)
 - 結果バッファ (`net_buffer_length`)

接続バッファと結果バッファはそれぞれ `net_buffer_length` バイトに等しいサイズから開始されますが、必要に応じて `max_allowed_packet` バイトまで動的に拡大されます。結果バッファは各 SQL ステートメントのあとに `net_buffer_length` バイトに縮小されます。ステートメントの実行中は現在のステートメント文字列のコピーも割り当てられます。

各接続スレッドは、ステートメントダイジェストの計算にメモリーを使用します。サーバーは、セッションごとに `max_digest_length` バイトを割り当てます。[セクション27.10 「パフォーマンススキーマのステートメントダイジェストとサンプリング」](#) を参照してください。

- すべてのスレッドで同じベースメモリーを共有します。
- スレッドが必要ない場合、それに割り当てられたメモリーが解放され、スレッドがスレッドキャッシュに戻らないかぎり、システムに返されます。その場合、メモリーは割り当てられた状態のままになります。
- テーブルの順次スキャンを実行する各リクエストによって、`read buffer` が割り当てられます。`read_buffer_size` システム変数によってバッファサイズが決まります。
- 任意の順序 (ソート後など) で行を読み取る場合、ディスクシークを回避するためにランダム読み取りバッファを割り当てることができます。`read_rnd_buffer_size` システム変数によってバッファサイズが決まります。
- すべての結合は単一のパスで実行され、ほとんどの結合は一時テーブルも使用せずに実行できます。ほとんどの一時テーブルはメモリーベースのハッシュテーブルです。大きな行長 (すべてのカラム長の合計として算出される) を持つ `BLOB` カラムを含む一時テーブルはディスク上に格納されます。
- ソートを実行するほとんどのリクエストは、ソートバッファおよび結果セットサイズに応じた 0 から 2 つの一時ファイルを割り当てます。[セクションB.3.3.5 「MySQL が一時ファイルを格納する場所」](#) を参照してください。
- ほとんどすべての解析と計算は、スレッドローカルの再利用可能なメモリープールで実行されます。小さい項目にはメモリーオーバーヘッドは必要ないため、通常の低速メモリー割当ておよび解放が回避されます。メモリーは、予測外に大きな文字列にのみ割り当てられます。
- `BLOB` カラムがあるテーブルごとに、大きな `BLOB` 値を読み取るためにバッファが動的に拡大されます。テーブルをスキャンすると、バッファは最大の `BLOB` 値と同じ大きさになります。
- MySQL には、テーブルキャッシュ用のメモリーおよびディスクリプタが必要です。使用中のすべてのテーブルのハンドラ構造はテーブルキャッシュに保存され、「先入れ先出し」(FIFO) として管理されます。

`table_open_cache` システム変数は、初期テーブルキャッシュサイズを定義します。セクション8.4.3.1「MySQLでのテーブルのオープンとクローズの方法」を参照してください。

MySQLには、テーブル定義キャッシュ用のメモリーも必要です。`table_definition_cache` システム変数は、テーブル定義キャッシュに格納できるテーブル定義の数を定義します。多数のテーブルを使用する場合は、大規模なテーブル定義キャッシュを作成してテーブルのオープンを高速化できます。テーブル定義キャッシュは、テーブルキャッシュとは異なり、使用する領域が少なく、ファイル記述子を使用しません。

- `FLUSH TABLES` ステートメントまたは `mysqladmin flush-tables` コマンドは、使用中でないすべてのテーブルを一度に閉じ、現在実行中のスレッドの終了時に閉じられるように使用中のすべてのテーブルをマークします。これにより、事実上ほとんどの使用中のメモリーが解放されます。`FLUSH TABLES` はすべてのテーブルが閉じられるまで戻りません。
- `GRANT`、`CREATE USER`、`CREATE SERVER`、および `INSTALL PLUGIN` ステートメントの結果として、サーバーは情報をメモリーにキャッシュします。このメモリーは、対応する `REVOKE`、`DROP USER`、`DROP SERVER` ステートメントおよび `UNINSTALL PLUGIN` ステートメントによって解放されないため、キャッシュを引き起こすステートメントの多くのインスタンスを実行するサーバーでは、`FLUSH PRIVILEGES` で解放されないかぎり、キャッシュされたメモリーの使用量が増加します。
- レプリケーショントポロジでは、次の設定はメモリー使用量に影響し、必要に応じて調整できます：
 - レプリケーションソースの `max_allowed_packet` システム変数は、ソースが処理のためにレプリカに送信する最大メッセージサイズを制限します。この設定のデフォルトは 64M です。
 - マルチスレッドレプリカ上の `slave_pending_jobs_size_max` システム変数は、処理待ちのメッセージを保持するために使用できるメモリーの最大量を設定します。この設定のデフォルトは 128M です。メモリーは必要な場合にのみ割り当てられますが、レプリケーショントポロジが大規模なトランザクションを処理する場合に使用されることがあります。これは弱い制限であり、より大きなトランザクションを処理できます。
 - レプリケーションソースまたはレプリカ上の `rpl_read_size` システム変数は、バイナリログファイルおよびリレーログファイルから読み取られるデータの最小量をバイト単位で制御します。デフォルトは 8192 バイトです。バイナリログおよびリレーログファイルから読み取るスレッドごとに、この値のバッファが割り当てられます。これには、ソース上のダンプスレッドやレプリカ上のコーディネータスレッドも含まれます。
 - `binlog_transaction_dependency_history_size` システム変数は、インメモリー履歴として保持される行ハッシュの数を制限します。
 - `max_binlog_cache_size` システム変数は、個々のトランザクションによるメモリー使用量の上限を指定します。
 - `max_binlog_stmt_cache_size` システム変数は、ステートメントキャッシュによるメモリー使用量の上限を指定します。

`ps` およびその他のステータスプログラムが、`mysqld` が大量のメモリーを使用していることをレポートすることがあります。これは、さまざまなメモリーアドレス上のスレッドスタックによって発生する可能性があります。たとえば、Solaris バージョンの `ps` はスタック間の未使用のメモリーが使用されているメモリーとしてカウントされます。これを確認するには、`swap -s` で使用可能なスワップをチェックします。いくつかのメモリーリーク検出ツール(市販とオープンソースの両方の)で `mysqld` をテストしているため、メモリーリークはないはずですが。

MySQL メモリー使用量の監視

次の例は、`Performance Schema` および `sys schema` を使用して MySQL メモリー使用量を監視する方法を示しています。

ほとんどのパフォーマンススキーマメモリーインストールメンテーションはデフォルトで無効になっています。インストールメントを有効にするには、パフォーマンススキーマ `setup_instruments` テーブルの `ENABLED` カラムを更新します。メモリーインストールメントには `memory/code_area/instrument_name` という形式の名前が付けられます。ここで、`code_area` は `sql` や `innodb` などの値で、`instrument_name` はインストールメントの詳細です。

1. 使用可能な MySQL メモリーインストールメントを表示するには、パフォーマンススキーマの `setup_instruments` テーブルをクエリーします。次のクエリーは、すべてのコード領域に対して何百ものメモリーインストールメントを返します。

```
mysql> SELECT * FROM performance_schema.setup_instruments
WHERE NAME LIKE '%memory%';
```

コード領域を指定して結果を絞り込むことができます。たとえば、コード領域として `innodb` を指定することで、結果を `InnoDB` メモリーインストゥルメントに制限できます。

```
mysql> SELECT * FROM performance_schema.setup_instruments
      WHERE NAME LIKE '%memory/innodb%';
```

NAME	ENABLED	TIMED
memory/innodb/adaptive hash index	NO	NO
memory/innodb/buf_buf_pool	NO	NO
memory/innodb/dict_stats_bg_recalc_pool_t	NO	NO
memory/innodb/dict_stats_index_map_t	NO	NO
memory/innodb/dict_stats_n_diff_on_level	NO	NO
memory/innodb/other	NO	NO
memory/innodb/row_log_buf	NO	NO
memory/innodb/row_merge_sort	NO	NO
memory/innodb/std	NO	NO
memory/innodb/trx_sys_t:rw_trx_ids	NO	NO

MySQL のインストールによっては、コード領域に `performance_schema`, `sql`, `client`, `innodb`, `myisam`, `csv`, `memory`, `blackhole`, `archive`, `partition` などが含まれる場合があります。

- メモリーインストゥルメントを有効にするには、MySQL 構成ファイルに `performance-schema-instrument` ルールを追加します。たとえば、すべてのメモリーインストゥルメントを有効にするには、このルールを構成ファイルに追加し、サーバーを再起動します:

```
performance-schema-instrument='memory/%=COUNTED'
```

注記

起動時にメモリーインストゥルメントを有効にすると、起動時に発生するメモリー割り当てが確実にカウントされます。

サーバーを再起動したあと、パフォーマンススキーマ `setup_instruments` テーブルの `ENABLED` カラムに、有効にしたメモリーインストゥルメントの `YES` が報告されます。メモリー操作が時間指定されていないため、`setup_instruments` テーブルの `TIMED` カラムはメモリーインストゥルメントで無視されます。

```
mysql> SELECT * FROM performance_schema.setup_instruments
      WHERE NAME LIKE '%memory/innodb%';
```

NAME	ENABLED	TIMED
memory/innodb/adaptive hash index	NO	NO
memory/innodb/buf_buf_pool	NO	NO
memory/innodb/dict_stats_bg_recalc_pool_t	NO	NO
memory/innodb/dict_stats_index_map_t	NO	NO
memory/innodb/dict_stats_n_diff_on_level	NO	NO
memory/innodb/other	NO	NO
memory/innodb/row_log_buf	NO	NO
memory/innodb/row_merge_sort	NO	NO
memory/innodb/std	NO	NO
memory/innodb/trx_sys_t:rw_trx_ids	NO	NO

- メモリーインストゥルメントデータをクエリーします。この例では、メモリーインストゥルメントデータがパフォーマンススキーマ `memory_summary_global_by_event_name` テーブルでクエリーされ、`EVENT_NAME` によってデータが要約されます。`EVENT_NAME` はインストゥルメントの名前です。

次のクエリーは、`InnoDB` バッファプールのメモリーデータを返します。カラムの説明は、[セクション 27.12.18.10「メモリーサマリーテーブル」](#) を参照してください。

```
mysql> SELECT * FROM performance_schema.memory_summary_global_by_event_name
      WHERE EVENT_NAME LIKE 'memory/innodb/buf_buf_pool\%G
      EVENT_NAME: memory/innodb/buf_buf_pool
      COUNT_ALLOC: 1
      COUNT_FREE: 0
      SUM_NUMBER_OF_BYTES_ALLOC: 137428992
      SUM_NUMBER_OF_BYTES_FREE: 0
```

```

LOW_COUNT_USED: 0
CURRENT_COUNT_USED: 1
HIGH_COUNT_USED: 1
LOW_NUMBER_OF_BYTES_USED: 0
CURRENT_NUMBER_OF_BYTES_USED: 137428992
HIGH_NUMBER_OF_BYTES_USED: 137428992

```

同じ基礎となるデータを `sys` スキーマ `memory_global_by_current_bytes` テーブルを使用してクエリーすることができます。このテーブルには、グローバルにサーバー内の現在のメモリー使用量が割当てタイプ別に分類されて表示されます。

```

mysql> SELECT * FROM sys.memory_global_by_current_bytes
      WHERE event_name LIKE 'memory/innodb/buf_buf_pool\%G
***** 1. row *****
event_name: memory/innodb/buf_buf_pool
current_count: 1
current_alloc: 131.06 MiB
current_avg_alloc: 131.06 MiB
high_count: 1
high_alloc: 131.06 MiB
high_avg_alloc: 131.06 MiB

```

この `sys` スキーマクエリーは、現在割り当てられているメモリー (`current_alloc`) をコード領域別に集計します:

```

mysql> SELECT SUBSTRING_INDEX(event_name,'/',2) AS
code_area, FORMAT_BYTES(SUM(current_alloc))
AS current_alloc
FROM sys.x$memory_global_by_current_bytes
GROUP BY SUBSTRING_INDEX(event_name,'/',2)
ORDER BY SUM(current_alloc) DESC;

```

code_area	current_alloc
memory/innodb	843.24 MiB
memory/performance_schema	81.29 MiB
memory/mysys	8.20 MiB
memory/sql	2.47 MiB
memory/memory	174.01 KiB
memory/myisam	46.53 KiB
memory/blackhole	512 bytes
memory/federated	512 bytes
memory/csv	512 bytes
memory/vio	496 bytes

注記

MySQL 8.0.16 より前は、`sys.format_bytes()` は `FORMAT_BYTES()` に使用されていました。

`sys` スキーマの詳細は、[第28章「MySQL sys スキーマ」](#) を参照してください。

8.12.3.2 ラージページのサポートの有効化

ハードウェアまたはオペレーティングシステムのアーキテクチャーによっては、デフォルト (通常は 4K バイト) よりも大きいメモリーページをサポートしています。このサポートの実際の実装は、ベースとなるハードウェアやオペレーティングシステムに依存します。大量のメモリーアクセスがあるアプリケーションの場合、大きいページを使用して、トランスレーションルックアサイドバッファ (TLB; Translation Lookaside Buffer) のミスが減ることによってパフォーマンスが改善される可能性があります。

MySQL では、InnoDB でラージページを使用して、バッファプールと追加のメモリープールにメモリーを割り当てることができます。

MySQL での標準的な大規模ページの使用では、サポートされる最大サイズである 4M バイトまでの使用が試行されます。Solaris では「超大規模ページ」機能により 256M バイトまでのページの使用が可能です。この機能は最新の SPARC プラットフォームで使用できます。これは `--super-large-pages` または `--skip-super-large-pages` オプションを使用して有効または無効にできます。

MySQL は、ラージページのサポートの Linux 実装 (Linux では HugeTLB と呼ばれる) もサポートします。

Linux でラージページを使用する前に、カーネルで、それらをサポートできるようにする必要があり、HugeTLB メモリプールを構成する必要があります。参考のため、HugeTBL API は、Linux ソースの [Documentation/vm/hugetlbpage.txt](#) ファイルで説明されています。

Red Hat Enterprise Linux などの一部の最近のシステムのカーネルでは、ラージページ機能がデフォルトで有効にされているようです。使用しているカーネルにこれが当てはまるかどうかを確認するには、次のコマンドを使用し、「huge」を含む出力行を探します。

```
shell> cat /proc/meminfo | grep -i huge
HugePages_Total: 0
HugePages_Free: 0
HugePages_Rsvd: 0
HugePages_Surp: 0
Hugepagesize: 4096 kB
```

空でないコマンド出力は、ラージページのサポートが存在することを示しますが、ゼロの値は、使用するよう構成されたページがないことを示します。

ラージページをサポートするようにカーネルを再構成する必要がある場合、手順については [hugetlbpage.txt](#) ファイルを参照してください。

Linux カーネルでラージページのサポートが有効にされていると仮定し、それを次のコマンドを使用して、MySQL で使用するよう構成します。通常、システムが起動するたびにコマンドが実行されるように、システムのブートシーケンス中に実行される `rc` ファイルまたは同等の起動ファイルにこれらを入れます。コマンドは、ブートシーケンスの早期の、MySQL サーバーが起動する前に実行されるべきです。システムに適切のように、割り当ての数値とグループ番号を変更してください。

```
# Set the number of pages to be used.
# Each page is normally 2MB, so a value of 20 = 40MB.
# This command actually allocates memory, so this much
# memory must be available.
echo 20 > /proc/sys/vm/nr_hugepages

# Set the group number that is permitted to access this
# memory (102 in this case). The mysql user must be a
# member of this group.
echo 102 > /proc/sys/vm/hugetlb_shm_group

# Increase the amount of shmем permitted per segment
# (12G in this case).
echo 1560281088 > /proc/sys/kernel/shmmax

# Increase total amount of shared memory. The value
# is the number of pages. At 4KB/page, 4194304 = 16GB.
echo 4194304 > /proc/sys/kernel/shmall
```

MySQL で使用する場合、通常 `shmmax` の値を `shmall` の値に近くなるようにしたいと考えます。

ラージページの構成を確認するには、前述のとおり再度 `/proc/meminfo` をチェックします。これで、0 以外の値が表示されるはずですが。

```
shell> cat /proc/meminfo | grep -i huge
HugePages_Total: 20
HugePages_Free: 20
HugePages_Rsvd: 0
HugePages_Surp: 0
Hugepagesize: 4096 kB
```

`hugetlb_shm_group` を使用するのための最後の手順は、`mysql` ユーザーに、`memlock` 制限として「unlimited」値を指定することです。これを行うには、`/etc/security/limits.conf` を編集するか、次のコマンドを `mysqld_safe` スクリプトに追加します:

```
ulimit -l unlimited
```

`ulimit` コマンドを `mysqld_safe` に追加すると、`mysql` ユーザーに切り替える前に `root` ユーザーの `memlock` 制限が `unlimited` に設定されます。(これは、`mysqld_safe` が `root` によって起動されたものと仮定します。)

MySQL のラージページのサポートはデフォルトで無効にされています。それを有効にするには、サーバーを `--large-pages` オプションで起動します。たとえば、サーバー `my.cnf` ファイルで次の行を使用できます:

```
[mysqld]  
large-pages
```

このオプションを使用すると、InnoDB はそのバッファプールと追加のメモリープールに自動的にラージページを使用します。InnoDB がこれを実行できない場合、従来のメモリーの使用に戻り、エラーログに警告を書き込みます：
[Warning: Using conventional memory pool](#)

ラージページが使用されていることを確認するには、再度 `/proc/meminfo` をチェックします。

```
shell> cat /proc/meminfo | grep -i huge  
HugePages_Total: 20  
HugePages_Free: 20  
HugePages_Rsvd: 2  
HugePages_Surp: 0  
Hugepagesize: 4096 kB
```

8.13 パフォーマンスの測定 (ベンチマーク)

パフォーマンスを測定するには、次の要因を考慮します。

- ビジーでないシステムで単一の操作の速度を測定するかどうか、一連の操作 (「ワークロード」) が一定の期間でどの程度機能するか。単純なテストでは、通常 1 つの側面 (構成設定、テーブルのインデックスのセット、クエリー内の SQL 句) の変化がパフォーマンスにどのように影響するかをテストします。ベンチマークは一般に長時間実行の複雑なパフォーマンステストであり、結果によって、ハードウェアやストレージ構成などの高レベルの選択や新しい MySQL バージョンにあとどのくらいでアップグレードするかが決まります。
- ベンチマークでは、正確な実態を得るために、重いデータベースワークロードをシミュレートする必要がある場合があります。
- パフォーマンスはきわめて多くのさまざまな要因によって異なる可能性があり、数パーセントの違いが決定的勝利にならないことがあります。結果は、別の環境でテストした場合に、逆の方向に転換することもあります。
- 特定の MySQL 機能は、ワークロードに応じて、パフォーマンスに役立つ場合と役立つ場合があります。完全性のため、常にそれらの機能をオンにした状態とオフにした状態でパフォーマンスをテストします。各ワークロードで試行する最も重要な機能は、InnoDB テーブルの [adaptive hash index](#) です。

このセクションでは、1 人の開発者が実行できる単純で直接的な測定技法から、実行と結果の解釈に追加の専門技術が必要とするもっと複雑な技法に進めていきます。

8.13.1 式と関数の速度の測定

特定の MySQL 式または関数の速度を測定するには、`mysql` クライアントプログラムを使用して、`BENCHMARK()` 関数を呼び出します。構文は `BENCHMARK(loop_count,expr)` です。戻り値は常に 0 ですが、`mysql` はステートメントの実行にどのくらいの時間を要したかを表示する行を出力します。例:

```
mysql> SELECT BENCHMARK(1000000,1+1);  
+-----+  
| BENCHMARK(1000000,1+1) |  
+-----+  
|          0 |  
+-----+  
1 row in set (0.32 sec)
```

この結果は Pentium II 400MHz システムで取得されました。これは、MySQL がそのシステムで 1,000,000 件の単純な加算式を 0.32 秒間で実行できることを示しています。

組み込みの MySQL 関数は一般に高度に最適化されますが、例外がある場合もあります。`BENCHMARK()` はクエリーで特定の関数が問題になっているかどうかを調べる場合に優れたツールです。

8.13.2 独自のベンチマークの使用

アプリケーションとデータベースのベンチマークを行い、ボトルネックのある場所を見つけます。1 つのボトルネックを修正 (または、それを「ダミー」モジュールで置換) することによって、次のボトルネックの識別に進むことができます。現在のアプリケーションの全体的なパフォーマンスが許容できるものであっても、いつか実際にパフォーマンスの強化が必要になった場合に、少なくとも各ボトルネックの計画を立て、解決方法を決定しておくべきです。

無料ベンチマークスイートは、<http://osdb.sourceforge.net/> で入手可能なオープンソースデータベースベンチマークです。

システムの負荷が非常に高い場合にのみ問題が発生することはよくあることです。(テスト済みの) システムを本稼働させて、負荷の問題が発生したときに、問い合わせしてくる顧客が多数いました。ほとんどの場合、パフォーマンスの問題は、高負荷時のテーブルスキャンの不良などデータベースの基本的な設計の問題か、オペレーティングシステムやライブラリの問題によると判明しています。ほとんどの場合、システムがまだ本稼働に入っていない場合の方がこれらの問題の修正がはるかに容易です。

このような問題を回避するには、可能性のある最悪の負荷でアプリケーション全体のベンチマークを行います。

- 複数のクライアントが同時にクエリーを発行して生成される高い負荷をシミュレートするには、[mysqlslap](#) プログラムが役立つ可能性があります。[セクション4.5.8「mysqlslap — ロードエミュレーションクライアント」](#)を参照してください。
- SysBench および DBT2 などのベンチマークパッケージを試してみることもできます。これは、<https://launchpad.net/sysbench> および <http://osdl.dbt.sourceforge.net/#dbt2> で入手できます。

これらのプログラムやパッケージはシステムを破損させる可能性があるため、それらは開発システムでのみ使用するようしてください。

8.13.3 performance_schema によるパフォーマンスの測定

[performance_schema](#) データベースのテーブルをクエリーし、それを実行しているサーバーとアプリケーションのパフォーマンス特性に関するリアルタイムの情報を確認できます。詳細は、[第27章「MySQL パフォーマンススキーマ」](#)を参照してください。

8.14 サーバースレッド (プロセス) 情報の確認

MySQL サーバーの動作を確認するには、サーバー内で実行されているスレッドのセットによって現在実行されている操作を示すプロセスリストを調べると役立ちます。例:

```
mysql> SHOW PROCESSLIST
***** 1. row *****
  Id: 5
  User: event_scheduler
  Host: localhost
  db: NULL
  Command: Daemon
  Time: 2756681
  State: Waiting on empty queue
  Info: NULL
***** 2. row *****
  Id: 20
  User: me
  Host: localhost:52943
  db: test
  Command: Query
  Time: 0
  State: starting
  Info: SHOW PROCESSLIST
```

スレッドは、[KILL](#) ステートメントを使用して強制終了できます。[セクション13.7.8.4「KILL ステートメント」](#)を参照してください。

8.14.1 プロセスリストへのアクセス

次の説明では、プロセス情報のソース、プロセス情報の表示に必要な特権を列挙し、プロセスリストエントリの内容について説明します。

- [プロセス情報のソース](#)
- [プロセスリストへのアクセスに必要な特権](#)
- [プロセスリストエントリの内容](#)

プロセス情報のソース

プロセス情報は、次のソースから入手できます:

- `SHOW PROCESSLIST` ステートメント: [セクション13.7.7.29「SHOW PROCESSLIST ステートメント」](#)
- `mysqladmin processlist` コマンド: [セクション4.5.2「mysqladmin — A MySQL Server 管理プログラム」](#)
- `INFORMATION_SCHEMA PROCESSLIST` テーブル: [セクション26.23「INFORMATION_SCHEMA PROCESSLIST テーブル」](#)
- パフォーマンススキーマの `processlist` テーブル: [セクション27.12.19.9「processlist テーブル」](#)
- 接頭辞が `PROCESSLIST_` の名前を持つパフォーマンススキーマ `threads` テーブルのカラム: [セクション27.12.19.10「スレッドテーブル」](#)
- `sys` スキーマの `processlist` ビューおよび `session` ビュー: [セクション28.4.3.22「processlist ビューと x\\$processlist ビュー」](#) および [セクション28.4.3.33「セッションおよび x\\$session ビュー」](#)

`threads` テーブルは、`SHOW PROCESSLIST`、`INFORMATION_SCHEMA PROCESSLIST` および `mysqladmin processlist` と次のように比較されます:

- `threads` テーブルへのアクセスに `mutex` は必要なく、サーバーのパフォーマンスへの影響は最小限です。他のソースには `mutex` が必要なため、パフォーマンスに悪影響があります。

注記

MySQL 8.0.22 の時点では、`threads` テーブルと同様に `mutex` を必要とせず、より優れたパフォーマンス特性を持つパフォーマンススキーマ `processlist` テーブルに基づいて、`SHOW PROCESSLIST` の代替実装を使用できます。詳細は、[セクション27.12.19.9「processlist テーブル」](#)を参照してください。

- `threads` テーブルにバックグラウンドスレッドが表示されますが、他のソースには表示されません。また、スレッドがフォアグラウンドまたはバックグラウンドスレッドであるかどうか、スレッドに関連付けられたサーバー内の場所など、他のソースが実行しない各スレッドの追加情報も提供します。つまり、`threads` テーブルを使用して、他のソースが監視できないスレッドアクティビティを監視できます。
- [セクション27.12.19.10「スレッドテーブル」](#) で説明されているように、パフォーマンススキーマスレッドのモニタリングを有効または無効にできます。

このような理由から、他のスレッド情報ソースのいずれかを使用してサーバー監視を実行する DBA は、かわりに `threads` テーブルを使用して監視できます。

`sys` スキーマの `processlist` ビューには、パフォーマンススキーマの `threads` テーブルの情報がよりアクセスしやすい形式で表示されます。 `sys` スキーマの `session` ビューには、`sys` スキーマの `processlist` ビューなどのユーザーセッションに関する情報が表示されますが、バックグラウンドプロセスはフィルタで除外されます。

プロセスリストへのアクセスに必要な特権

ほとんどのプロセス情報のソースでは、`PROCESS` 権限を持っている場合、他のユーザーに属するスレッドも含めて、すべてのスレッドを表示できます。それ以外の場合 (`PROCESS` 権限なし)、非匿名ユーザーは自分のスレッドに関する情報にはアクセスできますが、他のユーザーのスレッドにはアクセスできず、匿名ユーザーはスレッド情報にアクセスできません。

パフォーマンススキーマ `threads` テーブルはスレッド情報も提供しますが、テーブルアクセスは別の特権モデルを使用します。 [セクション27.12.19.10「スレッドテーブル」](#)を参照してください。

プロセスリストエントリの内容

各プロセスリストエントリには、いくつかの情報が含まれています。次のリストでは、`SHOW PROCESSLIST` 出力のラベルを使用してこれらについて説明します。その他のプロセス情報ソースでは、同様のラベルが使用されます。

- `Id` は、スレッドに関連付けられているクライアントの接続識別子です。
- `User` と `Host` は、スレッドに関連付けられているアカウントを示します。

- `db` はスレッドのデフォルトデータベースで、何も選択されていない場合は `NULL` です。
- `Command` と `State` は、スレッドが何を実行しているかを示します。

ほとんどの状態がきわめてすばやい操作に対応します。スレッドの状態が何秒間も特定の状態にとどまっている場合は、調査が必要な問題が発生している可能性があります。

以下のセクションでは、`Command` の可能な値と、カテゴリ別にグループ化した `State` の値を説明します。これらの一部の値の意味は自明です。その他については追加の説明を提供しています。

注記

プロセスリスト情報を検査するアプリケーションは、コマンドと状態が変更される可能性があることに注意する必要があります。

- `Time` は、スレッドの現在の状態がどれだけ続いているかを示します。特定の場合に、スレッドの現在の時間の概念が変わることがあります。スレッドは、`SET TIMESTAMP = value` によって時間を変更することがあります。レプリカ SQL スレッドの場合、この値は、最後にレプリケートされたイベントのタイムスタンプとレプリカホストのリアルタイムの間の秒数です。セクション17.2.3「レプリケーションスレッド」を参照してください。
- `Info` は、スレッドが実行しているステートメントを示します。ステートメントを実行していない場合は `NULL` を示します。`SHOW PROCESSLIST` の場合、この値にはステートメントの最初の 100 文字のみが含まれます。完全なステートメントを表示するには、`SHOW FULL PROCESSLIST` を使用します (または、異なるプロセス情報ソースをクエリーします)。

8.14.2 スレッドのコマンド値

スレッドの `Command` 値は次のいずれかになります。

- `Binlog Dump`
これは、バイナリログの内容をレプリカに送信するためのレプリケーションソース上のスレッドです。
- `Change user`
スレッドはユーザー変更操作を実行しています。
- `Close stmt`
スレッドはプリペアドステートメントをクローズしています。
- `Connect`
ソースに接続されているレプリケーションレシーバスレッドおよびレプリケーションワーカースレッドによって使用されます。
- `Connect Out`
レプリカはソースに接続しています。
- `Create DB`
スレッドはデータベース作成操作を実行しています。
- `Daemon`
このスレッドはサーバーの内部で使用され、クライアント接続をホストするスレッドではありません。
- `Debug`
スレッドはデバッグ情報を生成しています。
- `Delayed insert`
スレッドは遅延挿入ハンドラです。

- [Drop DB](#)
スレッドはデータベースの削除操作を実行しています。
- [Error](#)
- [Execute](#)
スレッドはプリペアドステートメントを実行しています。
- [Fetch](#)
スレッドはプリペアドステートメントの実行から結果をフェッチしています。
- [Field List](#)
スレッドはテーブルカラムの情報を取得しています。
- [Init DB](#)
スレッドはデフォルトのデータベースを選択しています。
- [Kill](#)
スレッドは別のスレッドを強制終了しています。
- [Long Data](#)
スレッドはプリペアドステートメントの実行の結果から長いデータを取得しています。
- [Ping](#)
スレッドはサーバー ping リクエストを処理しています。
- [Prepare](#)
スレッドはプリペアドステートメントを準備しています。
- [Processlist](#)
スレッドはサーバースレッドに関する情報を生成しています。
- [Query](#)
シングルスレッドレプリケーションアプライヤスレッドおよびレプリケーションコーディネータスレッドによるクエリーの実行中に、ユーザークライアントに使用されます。
- [Quit](#)
スレッドは終了しています。
- [Refresh](#)
スレッドは、テーブル、ログ、またはキャッシュをフラッシュしているか、ステータス変数またはレプリケーションサーバーの情報をリセットしています。
- [Register Slave](#)
スレッドはレプリカサーバーを登録しています。
- [Reset stmt](#)
スレッドはプリペアドステートメントをリセットしています。
- [Set option](#)
スレッドはクライアントステートメントの実行オプションを設定またはリセットしています。

- [Shutdown](#)

スレッドはサーバーをシャットダウンしています。

- [Sleep](#)

スレッドはクライアントが新しいステートメントをそれに送信するのを待機しています。

- [Statistics](#)

スレッドはサーバーステータス情報を生成しています。

- [Time](#)

使用されません。

8.14.3 一般的なスレッドの状態

次のリストは、レプリケーションなどの特殊なアクティビティーではなく、一般的なクエリーの処理に関連付けられた、スレッドの [State](#) 値を説明しています。これらの多くは、サーバーのバグを見つけるためにのみ役立ちます。

- [After create](#)

これは、スレッドがテーブル (内部一時テーブルも含む) を作成する際の、テーブルを作成する関数の最後に発生します。何らかのエラーのためテーブルを作成できなかった場合でも、この状態が使われます。

- [altering table](#)

サーバーはインプレース [ALTER TABLE](#) の実行中です。

- [Analyzing](#)

スレッドは [MyISAM](#) テーブルのキー分布を計算しています ([ANALYZE TABLE](#) など)。

- [checking permissions](#)

スレッドは、サーバーがステートメントを実行するために必要な権限を持っているかどうかを確認しています。

- [Checking table](#)

スレッドはテーブルチェック操作を実行しています。

- [cleaning up](#)

スレッドは 1 つのコマンドを処理し、メモリーの解放と特定の状態変数のリセットを準備しています。

- [closing tables](#)

スレッドは、変更されたテーブルデータをディスクにフラッシュし、使用されたテーブルをクローズしています。これは高速の操作であるはずですが、そうでない場合は、ディスクがいっぱいでないか、ディスクが著しく頻繁に使用されていないかを確認してください。

- [converting HEAP to ondisk](#)

スレッドは内部一時テーブルを [MEMORY](#) テーブルからディスク上のテーブルに変換しています。

- [copy to tmp table](#)

スレッドは [ALTER TABLE](#) ステートメントを処理しています。この状態は、新しい構造でテーブルが作成されたあと、ただし、それに行がコピーされる前に発生します。

この状態のスレッドの場合、パフォーマンススキーマを使用してコピー操作の進行状況を取得できます。 [セクション 27.12.5「パフォーマンススキーマステージイベントテーブル」](#) を参照してください。

- [Copying to group table](#)

ステートメントの `ORDER BY` と `GROUP BY` の基準が異なる場合、行はグループによってソートされ、一時テーブルにコピーされます。

- [Copying to tmp table](#)

サーバーはメモリー内の一時テーブルにコピーしています。

- [Copying to tmp table on disk](#)

サーバーはディスク上の一時テーブルにコピーしています。一時結果セットが大きくなりすぎました ([セクション 8.4.4 「MySQL での内部一時テーブルの使用」](#)を参照してください)。その結果、スレッドは一時テーブルをインメモリーからディスクベースのフォーマットに変更して、メモリーを節約します。

- [Creating index](#)

スレッドは `MyISAM` テーブルに対する `ALTER TABLE ... ENABLE KEYS` を処理しています。

- [Creating sort index](#)

スレッドは内部一時テーブルを使用して解決される `SELECT` を処理しています。

- [creating table](#)

スレッドはテーブルを作成しています。これには一時テーブルの作成が含まれます。

- [Creating tmp table](#)

スレッドはメモリー内またはディスク上に一時テーブルを作成しています。テーブルがメモリー内に作成され、後でディスク上のテーブルに変換される場合、その操作中の状態は [Copying to tmp table on disk](#) です。

- [committing alter table to storage engine](#)

サーバーはインプレース `ALTER TABLE` を終了し、結果をコミットしています。

- [deleting from main table](#)

サーバーは複数テーブル削除の最初の部分を実行しています。最初のテーブルからのみ削除し、別の (参照) テーブルからの削除に使用されるカラムとオフセットを保存しています。

- [deleting from reference tables](#)

サーバーは複数テーブル削除の 2 番目の部分を実行しており、別のテーブルから一致した行を削除しています。

- [discard_or_import_tablespace](#)

スレッドは `ALTER TABLE ... DISCARD TABLESPACE` または `ALTER TABLE ... IMPORT TABLESPACE` ステートメントを処理しています。

- [end](#)

これは、`ALTER TABLE`、`CREATE VIEW`、`DELETE`、`INSERT`、`SELECT`、または `UPDATE` ステートメントの最後、ただしクリーンアップの前に発生します。

`end` 状態では、次の操作が行われることがあります。

- バイナリログへのイベントの書き込み
- BLOB 用を含むメモリーバッファの解放

- [executing](#)

スレッドはステートメントの実行を開始しました。

- [Execution of init_command](#)

スレッドは `init_command` システム変数の値のステートメントを実行しています。

- `freeing items`

スレッドはコマンドを実行しました。通常、この状態のあとは `cleaning up` になります。

- `FULLTEXT initialization`

サーバーは自然言語全文検索を実行する準備をしています。

- `init`

これは、`ALTER TABLE`、`DELETE`、`INSERT`、`SELECT`、または `UPDATE` ステートメントの初期化の前に発生します。この状態のサーバーによって実行されるアクションには、バイナリログと `InnoDB` ログのフラッシュが含まれます。

- `Killed`

だれかがスレッドに `KILL` ステートメントを送っており、スレッドは次に強制終了フラグをチェックしたときに中止するはずですが、フラグは MySQL の各主要ループ内でチェックされますが、場合によってはスレッドが停止するまでに少し時間がかかる場合があります。スレッドがほかのスレッドにロックされている場合、強制終了はほかのスレッドがそのロックを解除するとすぐに有効になります。

- `Locking system tables`

スレッドがシステムテーブル (タイムゾーンやログテーブルなど) をロックしようとしています。

- `logging slow query`

スレッドはステートメントを低速クエリーログに書き込んでいます。

- `login`

クライアントが正常に認証されるまでの接続スレッドの初期状態です。

- `manage keys`

サーバーはテーブルインデックスを有効または無効にしています。

- `Opening system tables`

スレッドがシステムテーブル (タイムゾーンやログテーブルなど) を開こうとしています。

- `Opening tables`

スレッドはテーブルをオープンしようとして試みています。これは、何かにオープンを妨げられないかぎり、きわめて高速な手順であるはずですが、たとえば、`ALTER TABLE` または `LOCK TABLE` ステートメントは、そのステートメントが終了するまでテーブルのオープンを妨げることがあります。`table_open_cache` 値が十分に大きいことをチェックすることも価値があります。

システムテーブルの場合は、かわりに `Opening system tables` の状態が使用されます。

- `optimizing`

サーバーはクエリーの初期最適化を実行しています。

- `preparing`

この状態はクエリーの最適化中に発生します。

- `Purging old relay logs`

スレッドは不要なリレーログファイルを削除しています。

- `query end`

この状態は、クエリーを処理したあと、ただし `freeing items` 状態の前に発生します。

- `Receiving from client`

サーバーはクライアントからパケットを読み取っています。

- `Removing duplicates`

クエリーは、MySQL が早い段階で個別の操作を最適化できなくなるような方法で `SELECT DISTINCT` を使用していました。このため、MySQL は結果をクライアントに送る前にすべての重複した行を削除するための追加の段階を必要とします。

- `removing tmp table`

スレッドは `SELECT` ステートメントを処理したあとに内部一時テーブルを削除しています。一時テーブルが作成されなかった場合、この状態は使用されません。

- `rename`

スレッドはテーブルの名前を変更しています。

- `rename result table`

スレッドは `ALTER TABLE` ステートメントを処理しており、新しいテーブルを作成し、元のテーブルを置き換えるためにその名前を変更しています。

- `Reopen tables`

スレッドはテーブルのロックを取得しましたが、ロックの取得後、基盤となるテーブル構造が変更されたことを認識しました。それはロックを解除し、テーブルをクローズして、再度オープンしようとしています。

- `Repair by sorting`

修復コードはインデックスを作成するためにソートを使用しています。

- `preparing for alter table`

サーバーはインプレース `ALTER TABLE` の実行を準備しています。

- `Repair done`

スレッドは `MyISAM` テーブルのマルチスレッド修復を完了しました。

- `Repair with keycache`

修復コードはキーキャッシュ経由で、1つずつキーの作成を使用しています。これは `Repair by sorting` よりはるかに遅くなります。

- `Rolling back`

スレッドはトランザクションをロールバックしています。

- `Saving state`

`MyISAM` テーブルの修復や分析などの操作で、スレッドは新しいテーブルの状態を `.MYI` ファイルヘッダーに保存しています。状態には、行の数、`AUTO_INCREMENT` カウンタ、キー分布などの情報が含まれています。

- `Searching rows for update`

スレッドは、すべての一致する行を更新する前に、それらを見つけるための第1フェーズを実行しています。これは、`UPDATE` が、関連する行を見つけるために使用されるインデックスを変更している場合に、実行される必要があります。

- `Sending data`

スレッドは `SELECT` ステートメントの行を読み取り、処理して、データをクライアントに送信しています。この状態で行われる操作は、大量のディスクアクセス (読み取り) を実行する傾向があるため、特定のクエリーの存続期間にわたる最長時間実行状態になることがあります。

- `Sending to client`

サーバーがクライアントにパケットを書き込んでいます。

- `setup`

スレッドは `ALTER TABLE` 操作を開始しています。

- `Sorting for group`

スレッドは `GROUP BY` を満たすためにソートを実行しています。

- `Sorting for order`

スレッドは `ORDER BY` を満たすためにソートを実行しています。

- `Sorting index`

スレッドは `MyISAM` テーブルの最適化操作中に、より効率的なアクセスのためにインデックスページをソートしています。

- `Sorting result`

`SELECT` ステートメントの場合、これは `Creating sort index` と似ていますが、非一時テーブルに対するものです。

- `starting`

ステートメントの実行開始時の最初のステージ。

- `statistics`

サーバーはクエリー実行プランを開発するための統計を計算しています。スレッドが長期間この状態にある場合、サーバーはディスクに依存してほかの作業を実行している可能性があります。

- `System lock`

スレッドが `mysql_lock_tables()` を呼び出しましたが、以降スレッドの状態は更新されていません。これは、多くの理由で発生する可能性がある非常に一般的な状態です。

たとえば、スレッドがリクエストしようとしているか、テーブルの内部または外部システムロックを待機しています。これは、`InnoDB` が `LOCK TABLES` の実行中にテーブルレベルのロックを待機している場合に発生することがあります。この状態が外部ロックへのリクエストによって発生しており、同じ `MyISAM` テーブルにアクセスしている複数の `mysqld` サーバーを使用していない場合、`--skip-external-locking` オプションによって外部システムロックを無効にできます。ただし、外部ロックはデフォルトで無効になっているため、このオプションは無効になっている可能性があります。`SHOW PROFILE` の場合、この状態はスレッドがロックをリクエストしている (待機しているのではなく) ことを意味します。

システムテーブルの場合は、かわりに `Locking system tables` の状態が使用されます。

- `update`

スレッドはテーブルの更新を開始する準備ができています。

- `Updating`

スレッドは更新する行を探していて、それらを更新しています。

- `updating main table`

サーバーは複数テーブル更新の最初の部分を実行しています。最初のテーブルのみを更新しており、別の (参照) テーブルの更新に利用されるカラムとオフセットを保存しています。

- [updating reference tables](#)

サーバーは複数テーブル更新の 2 番目の部分を実行しており、ほかのテーブルから一致した行を更新しています。

- [User lock](#)

スレッドは [GET_LOCK\(\)](#) 呼び出しによってリクエストされたアドバイザリロックを、リクエストしようとしているか待機しています。 [SHOW PROFILE](#) の場合、この状態はスレッドがロックをリクエストしている (待機しているのではなく) ことを意味します。

- [User sleep](#)

スレッドは [SLEEP\(\)](#) 呼び出しを呼び出しました。

- [Waiting for commit lock](#)

[FLUSH TABLES WITH READ LOCK](#) はコミットロックを待機しています。

- [waiting for handler commit](#)

スレッドは、クエリー処理の他の部分に対するトランザクションのコミットを待機しています。

- [Waiting for tables](#)

スレッドは、テーブルの基盤となる構造が変更され、その新しい構造を得るためにテーブルを再度オープンする必要があるという通知を受け取りました。ただし、テーブルを再度オープンするには、ほかのすべてのスレッドが問題のテーブルをクローズするまで待機する必要があります。

この通知は、別のスレッドが [FLUSH TABLES](#) か、問題のテーブルに次のステートメントのいずれかを使用した場合に、この通知が行われます: [FLUSH TABLES tbl_name](#)、[ALTER TABLE](#)、[RENAME TABLE](#)、[REPAIR TABLE](#)、[ANALYZE TABLE](#)、または [OPTIMIZE TABLE](#)。

- [Waiting for table flush](#)

スレッドは [FLUSH TABLES](#) を実行しており、すべてのスレッドがテーブルを閉じるのを待機しているか、テーブルの基盤となる構造が変更されたため、新しい構造を取得するにはテーブルを再度開く必要があるという通知をスレッドが受け取りました。ただし、テーブルを再度オープンするには、ほかのすべてのスレッドが問題のテーブルをクローズするまで待機する必要があります。

この通知は、別のスレッドが [FLUSH TABLES](#) か、問題のテーブルに次のステートメントのいずれかを使用した場合に、この通知が行われます: [FLUSH TABLES tbl_name](#)、[ALTER TABLE](#)、[RENAME TABLE](#)、[REPAIR TABLE](#)、[ANALYZE TABLE](#)、または [OPTIMIZE TABLE](#)。

- [Waiting for lock_type lock](#)

サーバーは、メタデータロックサブシステムからの [THR_LOCK](#) ロックまたはロックの取得を待機しています ([lock_type](#) はロックのタイプを示します)。

この状態は、[THR_LOCK](#) の待機を示します:

- [Waiting for table level lock](#)

次の状態は、メタデータロックの待機を示します:

- [Waiting for event metadata lock](#)

- [Waiting for global read lock](#)

- [Waiting for schema metadata lock](#)

- [Waiting for stored function metadata lock](#)

- [Waiting for stored procedure metadata lock](#)

- [Waiting for table metadata lock](#)

- [Waiting for trigger metadata lock](#)

テーブルロックインジケータの詳細は、[セクション8.11.1「内部ロック方法」](#)を参照してください。メタデータのロックの詳細は、[セクション8.11.4「メタデータのロック」](#)を参照してください。ロック要求をブロックしているロックを確認するには、[セクション27.12.13「パフォーマンススキーマロックテーブル」](#)で説明されているパフォーマンススキーマロックテーブルを使用します。

- [Waiting on cond](#)

スレッドが条件が true になるのを待機している一般的な状態です。特定の状態情報は使用できません。

- [Writing to net](#)

サーバーはネットワークにパケットを書き込んでいます。

8.14.4 レプリケーションソーススレッドの状態

次のリストは、レプリケーションソースの [Binlog Dump](#) スレッドの [State](#) カラムに表示される可能性のあるもっとも一般的な状態を示しています。ソースに [Binlog Dump](#) スレッドが表示されない場合は、レプリケーションが実行されていない (つまり、現在接続されているレプリカがない) ことを意味します。

- [Finished reading one binlog; switching to next binlog](#)

スレッドはバイナリログファイルの読み取りを終了し、次にレプリカに送信するファイルを開いています。

- [Master has sent all binlog to slave; waiting for more updates](#)

スレッドはバイナリログから残りのすべての更新を読み取り、それらをレプリカに送信しました。スレッドはアイドル状態になり、ソースで新しい更新が発生した結果として新しいイベントがバイナリログに表示されるのを待機しています。

- [Sending binlog event to slave](#)

バイナリログはイベントで構成され、そこではイベントが通常更新と何らかのその他の情報が追加されたものになります。スレッドはバイナリログからイベントを読み取り、レプリカに送信しています。

- [Waiting to finalize termination](#)

スレッド停止中に発生するきわめて短い状態。

8.14.5 レプリケーション I/O スレッドの状態

次のリストに、レプリカサーバー上のレプリケーション I/O スレッドの [State](#) カラムに表示される最も一般的な状態を示します。この状態は、[SHOW REPLICAS | SLAVE STATUS](#) によって表示される [Replica_IO_State](#) カラムにも表示されるため、そのステートメントを使用して何が起きているかを適切に把握できます。

- [Checking master version](#)

ソースへの接続が確立された後、非常に短い状態になります。

- [Connecting to master](#)

スレッドはソースに接続しようとしています。

- [Queueing master event to the relay log](#)

スレッドはイベントを読み取っており、SQL スレッドがそれを処理できるように、それをリレーログにコピーしています。

- [Reconnecting after a failed binlog dump request](#)

スレッドはソースに再接続しようとしています。

- [Reconnecting after a failed master event read](#)

スレッドはソースに再接続しようとしています。ふたたび接続が確立されると、状態は [Waiting for master to send event](#) になります。

- [Registering slave on master](#)

ソースへの接続が確立された後に非常に短時間発生する状態。

- [Requesting binlog dump](#)

ソースへの接続が確立された後、非常に短い状態になります。スレッドは、要求されたバイナリログファイルの名前と位置から開始して、バイナリログの内容に対する要求をソースに送信します。

- [Waiting for its turn to commit](#)

[slave_preserve_commit_order](#) が有効な場合に、レプリカスレッドが古いワーカースレッドのコミットを待機しているときに発生する状態。

- [Waiting for master to send event](#)

スレッドはソースに接続し、バイナリログイベントの到着を待機しています。これは、ソースがアイドル状態の場合に長時間持続することがあります。待機が [slave_net_timeout](#) 秒継続した場合、タイムアウトになります。その時点で、スレッドは接続が切断されているとみなし、再接続を試みます。

- [Waiting for master update](#)

[Connecting to master](#) の前の初期状態。

- [Waiting for slave mutex on exit](#)

スレッドの停止中に一時的に発生する状態。

- [Waiting for the slave SQL thread to free enough relay log space](#)

0 以外の [relay_log_space_limit](#) 値を使用しており、リレーログの組み合わせたサイズがこの値を超えるまで拡大しています。I/O スレッドは、一部のリレーログファイルを削除できるように、リレーログ内容を処理することによって SQL スレッドが十分な領域を解放するまで、待機しています。

- [Waiting to reconnect after a failed binlog dump request](#)

切断のため、バイナリログダンプリクエストに失敗した場合、スレッドはスリープ中にこの状態になり、定期的に再接続を試みます。再試行間隔は、[CHANGE REPLICATION SOURCE TO](#) ステートメント (MySQL 8.0.23 の場合) または [CHANGE MASTER TO](#) ステートメント (MySQL 8.0.23 の場合) を使用して指定できます。

- [Waiting to reconnect after a failed master event read](#)

切断のため、読み取り中にエラーが発生しました。スレッドは、再接続を試行する前に、[CHANGE REPLICATION SOURCE TO](#) ステートメント (MySQL 8.0.23 の場合) または [CHANGE MASTER TO](#) ステートメント (MySQL 8.0.23 の場合) で設定された秒数 (デフォルトは 60) スリープしています。

8.14.6 レプリケーション SQL スレッドの状態

次のリストは、レプリカサーバー上のレプリケーション SQL スレッドの [State](#) カラムに表示される可能性のある最も一般的な状態を示しています:

- [Making temporary file \(append\) before replaying LOAD DATA INFILE](#)

スレッドは [LOAD DATA](#) ステートメントを実行しており、レプリカが行を読み取るデータを含む一時ファイルにデータを追加しています。

- [Making temporary file \(create\) before replaying LOAD DATA INFILE](#)

スレッドは [LOAD DATA](#) ステートメントを実行し、レプリカが行を読み取るデータを含む一時ファイルを作成しています。この状態は、元の [LOAD DATA](#) ステートメントが、MySQL 5.0.3 より前のバージョンの MySQL を実行しているソースによってログに記録された場合にのみ発生します。

- [Reading event from the relay log](#)
スレッドはイベントを処理できるように、イベントをリレーログから読み取りました。
- [Slave has read all relay log; waiting for more updates](#)
スレッドはリレーログファイル内のすべてのイベントを処理しており、現在 I/O スレッドが新しいイベントをリレーログに書き込むのを待機しています。
- [Waiting for an event from Coordinator](#)
マルチスレッドレプリカ (`slave_parallel_workers` が 1 より大きい) を使用して、レプリカワーカースレッドのいずれかがコーディネータスレッドからのイベントを待機しています。
- [Waiting for slave mutex on exit](#)
スレッド停止中に発生するきわめて短い状態。
- [Waiting for Slave Workers to free pending events](#)
この待機処理は、ワーカーが処理しているイベントの合計サイズが `slave_pending_jobs_size_max` システム変数のサイズを超えた場合に発生します。サイズがこの制限を下回ると、コーディネータはスケジューリングを再開します。この状態は、`slave_parallel_workers` が 0 より大きい値に設定されている場合にのみ発生します。
- [Waiting for the next event in relay log](#)
[Reading event from the relay log](#) の前の初期状態です。
- [Waiting until MASTER_DELAY seconds after master executed event](#)
SQL スレッドはイベントを読み取りましたが、レプリカ遅延の経過を待機しています。この遅延は、[CHANGE REPLICATION SOURCE TO](#) ステートメント (MySQL 8.0.23 の場合) または [CHANGE MASTER TO](#) ステートメント (MySQL 8.0.23 の場合) の `SOURCE_DELAY` | `MASTER_DELAY` オプションを使用して設定します。
SQL スレッドの `Info` カラムには、ステートメントのテキストも表示される場合があります。これは、スレッドがリレーログからイベントを読み取り、それからステートメントを抽出して、それを実行している可能性があることを示しています。

8.14.7 レプリケーション接続スレッドの状態

これらのスレッドの状態はレプリカサーバーで発生しますが、I/O または SQL スレッドではなく接続スレッドに関連付けられています。

- [Changing master](#)
スレッドは、(MySQL 8.0.23 の) [CHANGE REPLICATION SOURCE TO](#) ステートメントまたは (MySQL 8.0.23 の前の) [CHANGE MASTER TO](#) ステートメントを処理しています。
- [Killing slave](#)
スレッドは [STOP REPLICA](#) | [SLAVE](#) ステートメントを処理しています。
- [Opening master dump table](#)
この状態は [Creating table from master dump](#) のあとに発生します。
- [Reading master dump table data](#)
この状態は [Opening master dump table](#) のあとに発生します。
- [Rebuilding the index on master dump table](#)
この状態は [Reading master dump table data](#) のあとに発生します。

8.14.8 NDB Cluster スレッドの状態

- [Committing events to binlog](#)
- [Opening mysql.ndb_apply_status](#)
- [Processing events](#)

スレッドはバイナリロギングのイベントを処理しています。
- [Processing events from schema table](#)

スレッドはスキーマレプリケーションの作業を実行しています。
- [Shutting down](#)
- [Syncing ndb table schema operation and binlog](#)

これは、NDB のスキーマ操作の正しいバイナリログを維持するために使用されます。
- [Waiting for allowed to take ndbcluster global schema lock](#)

スレッドはグローバルスキーマロックを取得する権限を待機しています。
- [Waiting for event from ndbcluster](#)

サーバーは NDB Cluster で SQL ノードとして機能し、クラスタ管理ノードに接続されています。
- [Waiting for first event from ndbcluster](#)
- [Waiting for ndbcluster binlog update to reach current position](#)
- [Waiting for ndbcluster global schema lock](#)

スレッドは、別のスレッドによって保持されているグローバルスキーマロックが解放されるのを待機しています。
- [Waiting for ndbcluster to start](#)
- [Waiting for schema epoch](#)

スレッドはスキーマエポック (つまり、グローバルチェックポイント) を待機しています。

8.14.9 イベントスケジューラスレッドの状態

これらの状態は、イベントスケジューラスレッド、スケジュールされたイベントを実行するために作成されるスレッド、またはスケジューラを終了するスレッドで発生します。

- [Clearing](#)

スケジューラスレッドまたはイベントを実行していたスレッドは終了中で、まもなく終了します。
- [Initialized](#)

スケジューラスレッドまたはイベントを実行するスレッドが初期化されました。
- [Waiting for next activation](#)

スケジューラには空でないイベントキューがありますが、次のアクティブ化はあとで行われます。
- [Waiting for scheduler to stop](#)

スレッドは `SET GLOBAL event_scheduler=OFF` を発行し、スケジューラが停止するのを待機しています。
- [Waiting on empty queue](#)

スケジューラのイベントキューは空で、スリープ中です。

第 9 章 言語構造

目次

9.1 リテラル値	1627
9.1.1 文字列リテラル	1627
9.1.2 数値リテラル	1630
9.1.3 日付リテラルと時間リテラル	1630
9.1.4 16 進数リテラル	1632
9.1.5 ビット値リテラル	1634
9.1.6 boolean リテラル	1635
9.1.7 NULL 値	1635
9.2 スキーマオブジェクト名	1635
9.2.1 識別子の長さ制限	1637
9.2.2 識別子の修飾子	1638
9.2.3 識別子の小文字と大文字の区別	1639
9.2.4 識別子とファイル名のマッピング	1641
9.2.5 関数名の構文解析と解決	1642
9.3 キーワードと予約語	1645
9.4 ユーザー定義変数	1673
9.5 式	1676
9.6 クエリー属性	1680
9.7 コメント	1683

この章では、MySQL を使用するとき、SQL ステートメントの次の要素を書き込むためのルールについて説明します。

- リテラル値: 文字列や数値など
- 識別子: データベース名、テーブル名、カラム名など
- キーワードと予約語
- ユーザー定義変数とシステム変数
- 式
- クエリー属性
- コメント

9.1 リテラル値

このセクションでは、MySQL でリテラル値を記述する方法について説明します。これには、文字列、数値、16 進数とビット値、ブール値および `NULL` が含まれます。このセクションでは、MySQL でこれらの基本的なタイプを処理する際に発生する可能性のある様々なナツスについても説明します。

9.1.1 文字列リテラル

文字列は、一重引用符 (') または二重引用符 (") 文字で囲まれた一連のバイトまたは文字です。例:

```
'a string'
"another string"
```

隣同士にある引用符付きの文字列は、1 つの文字列に連結されます。次の行は同等です。

```
'a string'
```

```
'a''string'
```

ANSI_QUOTES SQLモードを有効にしている場合は、二重引用符で囲んだ文字列は識別子として解釈されるため、文字列リテラルを囲む引用符には単一引用符だけを使用できます。

バイナリ文字列はバイトの文字列です。すべてのバイナリ文字列には、**binary** という名前の文字セットと照合順序があります。非バイナリ文字列は文字列です。**binary** 以外の文字セットと、文字セットと互換性のある照合順序があります。

これらの両方の文字列タイプは、文字列単位の数値に基づいて比較されます。バイナリ文字列の場合、単位はバイトです。比較では数値バイト値が使用されます。非バイナリ文字列の場合、単位は文字であり、一部の文字セットではマルチバイト文字がサポートされます。比較では数値文字コード値が使用されます。文字コードの順序付けは、文字列照合の関数です。(詳細は[セクション10.8.5「バイナリ照合順序と_bin 照合順序」](#)を参照してください。)

文字列リテラルには、特定の文字セットおよび照合順序を使用する文字列として指定するために、オプションの文字セットイントロデューサおよび **COLLATE** 句を含めることができます:

```
[_charset_name]'string' [COLLATE collation_name]
```

例:

```
SELECT _latin1'string';
SELECT _binary'string';
SELECT _utf8'string' COLLATE utf8_danish_ci;
```

N'literal' (または **n'literal'**) を使用すると、各国文字セットの文字列を作成できます。次のステートメントは同等です。

```
SELECT N'some text';
SELECT n'some text';
SELECT _utf8'some text';
```

これらの形式の文字列構文の詳細は、[セクション10.3.7「各国語文字セット」](#) および [セクション10.3.8「文字セットイントロデューサ」](#) を参照してください。

NO_BACKSLASH_ESCAPES SQL モードが有効になっている場合を除いて、一部のシーケンスが文字列内で特別な意味を持ちます。これらのシーケンスはいずれも、エスケープ文字として知られるバックスラッシュ (\) で始まります。MySQL は、[表9.1「特殊文字エスケープシーケンス」](#) に示すエスケープシーケンスを認識します。ほかのすべてのエスケープシーケンスでは、バックスラッシュは無視されます。つまり、エスケープされた文字がエスケープされていないと解釈されます。たとえば、**\x** は単なる **x** です。これらの順序では、大/小文字が区別されます。たとえば、**\b** はバックスペースとして解釈されますが、**\B** は **B** として解釈されます。エスケープ処理は **character_set_connection** システム変数で指定された文字セットに応じて実行されます。[セクション10.3.6「文字列リテラルの文字セットおよび照合順序」](#) で説明するとおり、ほかの文字セットを示すイントロデューサが前に置かれている文字列についても同じことがいえます。

表 9.1 特殊文字エスケープシーケンス

エスケープシーケンス	シーケンスが表す文字
\0	ASCII NUL (X'00') 文字
\'	単一引用符 (') 文字
\"	二重引用符 (") 文字
\b	バックスペース文字
\n	改行 (ラインフィード) 文字
\r	復帰改行文字
\t	タブ文字
\Z	ASCII 26 (Control+Z): テーブルの後のノートを参照してください
\\	バックスラッシュ (\) 文字
\%	% 文字。テーブルの後のノートを参照してください

エスケープシーケンス	シーケンスが表す文字
<code>_</code>	<code>_</code> 文字。テーブルの後のノートを参照してください

ASCII 26 文字を`\Z`としてエンコードすると、Windows で ASCII 26 が END-OF-FILE を表すという問題を回避できます。 `mysql db_name < file_name` を使用しようとする、ファイル内の ASCII 26 が問題を引き起こします。

`\%` および `_` の順序は、パターン一致コンテキストで `%` および `_` のリテラルインスタンスを検索するために使用され、それ以外の場合はワイルドカード文字として解釈されます。 [セクション12.8.1「文字列比較関数および演算子」](#)内の `LIKE` 演算子に関する記述を参照してください。パターン一致コンテキストの外部で `\%` または `_` を使用する場合、`%` および `_` ではなく、文字列 `\%` および `_` に評価されます。

文字列に引用符を含める方法は、いくつかあります。

- ' で引用符で囲まれた文字列内の ' は、" として記述できます。
- " で引用符で囲まれた文字列内の " は、" として記述できます。
- 引用符文字の直前にエスケープ文字 (`\`) を指定します。
- " で引用符で囲まれた文字列内の ' は特別な処理を必要とせず、二重にしたりエスケープしたりする必要はありません。同様に、' で引用符で囲まれた文字列内の " では、特別な処理は必要ありません。

次の `SELECT` ステートメントは、引用符を使用した場合とエスケープを使用した場合にどのような効果があるかを示しています。

```
mysql> SELECT 'hello', "hello", ""hello"", 'hel"lo', \'hello';
+-----+-----+-----+-----+
| hello | "hello" | ""hello"" | hel"lo | \'hello |
+-----+-----+-----+-----+

mysql> SELECT "hello", ""hello"", ""hello"", "hel""lo", "\"hello";
+-----+-----+-----+-----+
| hello | 'hello' | "hello" | hel"lo | "hello |
+-----+-----+-----+-----+

mysql> SELECT 'This\nIs\nFour\nLines';
+-----+
| This
Is
Four
Lines |
+-----+

mysql> SELECT 'disappearing\ backslash';
+-----+
| disappearing backslash |
+-----+
```

バイナリデータを文字列カラム (`BLOB` カラムなど) に挿入するには、特定の文字をエスケープシーケンスで表す必要があります。バックslash (`\`) と、文字列を囲む引用符は、エスケープする必要があります。ある種のクライアント環境では、`NUL` や `Ctrl+Z` もエスケープする必要があります。 `mysql` クライアントは、`NUL` 文字がエスケープされていない場合、これを含む引用符付きの文字列を切り捨てます。 `Ctrl+Z` は、エスケープされていない場合、Windows で END-OF-FILE を表すと見なされる可能性があります。これらのそれぞれの文字を表すエスケープシーケンスについては、 [表9.1「特殊文字エスケープシーケンス」](#) を参照してください。

アプリケーションプログラムを書く場合、MySQL Server に送信される SQL ステートメント内で文字列がデータ値として使用される前に、これらの特殊文字を含む可能性のある文字列は適切にエスケープする必要があります。これには次の2つの方法があります。

- 特殊文字をエスケープする関数を使用して文字列を処理します。C プログラムでは、 `mysql_real_escape_string_quote()` C API 関数を使用して文字をエスケープできます。 `mysql_real_escape_string_quote()` を参照してください。ほかの SQL ステートメントを構成する SQL ステートメント内では、 `QUOTE()` 関数を使用できます。Perl DBI インタフェースでは、 `quote` メソッドを使用して特殊文字を適切なエスケープシーケンスに変換できます。 [セクション29.9「MySQL Perl API」](#) を参照してください。ほかの言語インタフェースでも同様の機能を利用できることがあります。

- 特殊文字を明示的にエスケープする方法以外に、多くの MySQL API には、ステートメント文字列に特殊なマーカーを挿入し、ステートメントの発行時にデータ値をそれらのマーカーにバインドできるプレースホルダー機能が備わっています。この場合、値内の特殊文字のエスケープ処理は API によって自動で行われます。

9.1.2 数値リテラル

数値リテラルには、正確値 (整数と **DECIMAL**) リテラルと近似値 (浮動小数点) リテラルが含まれます。

整数は数字の列として表現されます。数値には小数点としてが含まれる場合があります。数値の前に **-** または **+** を付けて、それぞれ負または正の値を示すことができます。仮数と指数による指数表現で表される数値は、近似値の数値です。

正確値の数値リテラルには、整数部または小数部、あるいはその両方が含まれています。これらには符号を付けることができます。例: **1**、**.2**、**3.4**、**-5**、**-6.78**、**+9.10**。

近似値の数値リテラルは仮数と指数による指数表現で表されます。一方または両方の部分に符号を付けることができます。例: **1.2E3**、**1.2E-3**、**-1.2E3**、**-1.2E-3**。

2つの数値が同じように見えても、別々に扱われることがあります。たとえば、**2.34** は (固定小数点の) 正確値ですが、**2.34E0** は (浮動小数点の) 近似値です。

DECIMAL データ型は固定小数点型で、計算は正確です。MySQL では、**DECIMAL** 型には、**NUMERIC**、**DEC**、**FIXED** という複数のシノニムがあります。整数型も正確値型です。正確値計算の詳細は、[セクション12.25「高精度計算」](#)を参照してください。

FLOAT データ型および **DOUBLE** データ型は浮動小数点型で、計算によって近似値が得られます。MySQL では、**FLOAT** または **DOUBLE** のシノニムである型は **DOUBLE PRECISION** および **REAL** です。

整数は浮動小数点コンテキストで使用でき、同等の浮動小数点数として解釈されます。

9.1.3 日付リテラルと時間リテラル

日付値と時間値は、値の正確型とほかの要因に応じて、引用符付きの文字列や数値など、複数の形式で表現できます。たとえば、MySQL が日付を予想するコンテキストでは、**'2015-07-21'**、**20150721**、**20150721** のいずれも日付と解釈します。

このセクションでは日付リテラルと時間リテラルの許容される形式について説明します。許可される値の範囲など、時間データ型の詳細は、[セクション11.2「日時データ型」](#)を参照してください。

標準 SQL と ODBC の日付および時間リテラル。標準 SQL では、型キーワードと文字列を使用して時間リテラルを指定する必要があります。キーワードと文字列の間の空白はオプションです。

```
DATE 'str'  
TIME 'str'  
TIMESTAMP 'str'
```

MySQL は認識しますが、標準 SQL とは異なり、type キーワードは必要ありません。標準に準拠するアプリケーションには、時間リテラルの type キーワードを含める必要があります。

MySQL では、標準 SQL 構文に対応する ODBC 構文も認識されます:

```
{ d 'str' }  
{ t 'str' }  
{ ts 'str' }
```

MySQL では、型キーワードと ODBC コンストラクトを使用して、**DATE**、**TIME** および **DATETIME** の値がそれぞれ生成されます。後続の小数秒部分 (指定されている場合) も含まれます。**DATETIME** には、標準 SQL の **TIMESTAMP** 型により密接に対応する範囲があり、ここには **0001** から **9999** の年範囲が含まれるので、**TIMESTAMP** 構文は、MySQL で **DATETIME** 値を生成します。(MySQL の **TIMESTAMP** 年範囲は **1970** から **2038** です。)

日時コンテキストでの文字列リテラルと数値リテラル。MySQL は次の形式で **DATE** 値を認識します。

- 'YYYY-MM-DD'または'YY-MM-DD'形式の文字列として。「緩やかな」構文が許可されます。どの句読点文字でも、日付部分間の区切り文字として使用できます。たとえば、**'2012-12-31'**、**'2012/12/31'**、**'2012^12^31'**、および **'2012@12@31'** は同等です。

- 文字列が日付として意味を持つ場合は、'YYYYMMDD'または'YYMMDD'形式のデリミタのない文字列として。たとえば、'20070523'と'070523'は'2007-05-23'として解釈されますが、'071332'は正しくないため(月と日の部分が不適切)、'0000-00-00'になります。
- YYYYMMDD または YYMMDD 形式の数値(日付として適切なもの)として。たとえば、19830905 と 830905 は '1983-09-05' として解釈されます。

MySQL は次の形式で DATETIME および TIMESTAMP 値を認識します。

- 'YYYY-MM-DD hh:mm:ss'または'YY-MM-DD hh:mm:ss'形式の文字列として。「緩やかな」構文はここでも許可されます。どの句読点文字でも、日付部分または時間部分の間の区切り文字として使用できます。たとえば、'2012-12-31 11:30:45'、'2012^12^31 11+30+45'、'2012/12/31 11*30*45'、および '2012@12@31 11^30^45' は同等です。

日付および時間の部分と小数秒部分との間の区切り文字として認識される唯一の文字が小数点です。

日付部分と時間部分は、空白ではなく T で区切ることもできます。たとえば、'2012-12-31 11:30:45' と '2012-12-31T11:30:45' は同等です。

- 文字列が日付として意味を持つ場合は、'YYYYMMDDhhmmss'または'YYMMDDhhmmss'形式のデリミタのない文字列として。たとえば、'20070523091528' と '070523091528' は '2007-05-23 09:15:28' として解釈されますが、'071122129015' は正しくないため(分の部分が不適切)、'0000-00-00 00:00:00' になります。
- 数値が日付として意味を持つ場合は、YYYYMMDDhhmmss または YYMMDDhhmmss 形式の数値として。たとえば、19830905132800 と 830905132800 は '1983-09-05 13:28:00' として解釈されます。

DATETIME または TIMESTAMP 値には、マイクロ秒(6桁)までの精度の後続の小数秒部分を含めることができます。小数部は、常に時間の残りの部分から小数点で区別する必要があります。これ以外の小数秒区切り文字は認識されません。MySQL の小数秒のサポートの詳細は、[セクション11.2.6「時間値での小数秒」](#)を参照してください。

2桁の年を含む日付の値は、世紀が不明なためあいまいです。MySQL は次のルールを使用して2桁の年の値を解釈します。

- 70-99 の範囲内の年の値は 1970-1999 になります。
- 00-69 の範囲内の年の値は 2000-2069 になります。

[セクション11.2.8「日付の2桁の年」](#)も参照してください。

日付部分の区切り文字を含む文字列として指定される値の場合、10未満の月または日の値に2桁を指定する必要はありません。'2015-6-9'は'2015-06-09'と同じです。同様に、時間部分の区切り文字を含む文字列として指定される値の場合、10未満の時、分、または秒の値に2桁を指定する必要はありません。'2015-10-30 1:2:3'は'2015-10-30 01:02:03'と同じです。

数値として指定する値は、6、8、12、14のいずれかの桁数にするようにしてください。数値の長さが8桁または14桁の場合、YYYYMMDD または YYYYMMDDhhmmss 形式であるとみなされ、年は最初の4桁で指定されます。数値の長さが6桁または12桁の場合、YYMMDD または YYMMDDhhmmss 形式であるとみなされ、年は最初の2桁で指定されます。これらの長さではない数字は、いちばん近い長さまで先行ゼロで埋められているかのように解釈されます。

区切り文字がない文字列として指定された値は、その長さに従って解釈されます。8または14文字の長さの文字列の場合、年は最初の4文字で表されていると見なされます。そうでなければ、年は最初の2文字で表されていると見なされます。文字列は、その文字列に存在するだけの部分について、左から右に順番に、年、月、日、時、分、秒として解釈されます。これは、6文字より少ない文字列は利用してはいけないということを意味します。たとえば、1999年3月を表すと考えて'9903'を指定しても、MySQLは「ゼロ」日付値に変換します。これは、年および月の値は99と03であるけれども、日の部分が完全に欠落しているために起こります。ただし、明示的に値ゼロを指定することによって、欠落している月や日の部分を表すことができます。たとえば、'1999-03-00'の値を挿入するには、'990300'を使用します。

MySQL は次の形式で TIME 値を認識します。

- 'D hh:mm:ss'形式の文字列として。次の「リラックス」構文のいずれかを使用することもできます:'hh:mm:ss'、'hh:mm'、'D hh:mm'、'D hh'または'ss'。この場合、D は日を表し、0 から 34 の値を指定できます。

- 'hhmmss'形式のデリミタのない文字列として(時間として意味がある場合)。たとえば、'101112'は'10:11:12'として認識されますが、'109712'は正しくないため(分の部分が不適切)、'00:00:00'になります。
- 時間として意味がある場合は、hhmmss形式の数値として。たとえば、101112は'10:11:12'として認識されます。次の代替形式も理解されます: ss、mmss または hhmmss。

後続の小数秒部分は、'D hh:mm:ss.fraction', 'hh:mm:ss.fraction', 'hhmmss.fraction'および hhmmss.fraction の時間書式で認識されます。ここで、fraction はマイクロ秒 (6 桁) までの精度の小数部です。小数部は、常に時間の残りの部分から小数点で区分する必要があります。これ以外の小数秒区切り文字は認識されません。MySQL の小数秒のサポートの詳細は、[セクション11.2.6「時間値での小数秒」](#)を参照してください。

時間部分の区切り文字を含む文字列として指定される TIME 値の場合、10 未満の時、分、秒の値に 2 桁を指定する必要はありません。'8:3:2'は'08:03:02'と同じです。

9.1.4 16 進数リテラル

16 進数リテラル値は、X'val'または 0xval 表記を使用して書き込まれます。val には 16 進数の桁 (0..9、A..F) が含まれます。数字および先行する X の大文字と小文字は関係ありません。先頭の 0x では大/小文字が区別され、0X として書き込むことはできません。

有効な 16 進数リテラル:

```
X'01AF'
X'01af'
x'01AF'
x'01af'
0x01AF
0x01af
```

不正な 16 進数リテラル:

```
X'0G' (G is not a hexadecimal digit)
0X01AF (0X must be written as 0x)
```

X'val'表記法を使用して記述された値には偶数の桁数が含まれている必要があり、含まれていない場合は構文エラーが発生します。問題を修正するには、値の先頭にゼロを埋め込みます:

```
mysql> SET @s = X'FFF';
ERROR 1064 (42000): You have an error in your SQL syntax;
check the manual that corresponds to your MySQL server
version for the right syntax to use near 'X'FFF'

mysql> SET @s = X'0FFF';
Query OK, 0 rows affected (0.00 sec)
```

奇数桁を含む 0xval 表記を使用して記述された値は、余分な先行 0 を持つものとして扱われます。たとえば、0xaaa は 0x0aaa として解釈されます。

デフォルトでは、16 進数リテラルはバイナリ文字列で、16 進数の各ペアは文字を表します:

```
mysql> SELECT X'4D7953514C', CHARSET(X'4D7953514C');
+-----+-----+
| X'4D7953514C' | CHARSET(X'4D7953514C') |
+-----+-----+
| MySQL        | binary                  |
+-----+-----+
mysql> SELECT 0x5461626c65, CHARSET(0x5461626c65);
+-----+-----+
| 0x5461626c65 | CHARSET(0x5461626c65) |
+-----+-----+
| Table        | binary                  |
+-----+-----+
```

16 進数リテラルには、特定の文字セットおよび照合順序を使用する文字列として指定するために、オプションの文字セットイントロデューサおよび COLLATE 句を含めることができます:

```
[charset_name] X'val' [COLLATE collation_name]
```

例:

```
SELECT _latin1 X'4D7953514C';
SELECT _utf8 0x4D7953514C COLLATE utf8_danish_ci;
```

この例では `X'val'` 表記法を使用しますが、`0xval` 表記法ではイントロデューサも許可されています。イントロデューサについては、[セクション10.3.8「文字セットイントロデューサ」](#)を参照してください。

数値コンテキストでは、MySQL は 16 進数リテラルを `BIGINT` (64-bit 整数) のように扱います。16 進リテラルの数値処理を保証するには、数値コンテキストで使用します。これには、0 の追加または `CAST(... AS UNSIGNED)` の使用が含まれます。たとえば、ユーザー定義変数に割り当てられた 16 進リテラルは、デフォルトではバイナリ文字列です。値を数値として割り当てるには、数値コンテキストで使用します:

```
mysql> SET @v1 = X'41';
mysql> SET @v2 = X'41'+0;
mysql> SET @v3 = CAST(X'41' AS UNSIGNED);
mysql> SELECT @v1, @v2, @v3;
+-----+-----+
| @v1 | @v2 | @v3 |
+-----+-----+
| A   | 65  | 65  |
+-----+-----+
```

空の 16 進数値 (`X''`) は、長さゼロのバイナリ文字列に評価されます。数値に変換され、0 が生成されます:

```
mysql> SELECT CHARSET(X''), LENGTH(X'');
+-----+-----+
| CHARSET(X'') | LENGTH(X'') |
+-----+-----+
| binary      | 0           |
+-----+-----+
mysql> SELECT X''+0;
+-----+
| X''+0      |
+-----+
| 0          |
+-----+
```

`X'val'` 表記は標準 SQL に基づいています。0x 表記は ODBC に基づいており、`BLOB` カラムの値を指定するために 16 進数文字列がよく使用されます。

文字列または数値を 16 進数形式の文字列に変換するには、`HEX()` 関数を使用します:

```
mysql> SELECT HEX('cat');
+-----+
| HEX('cat') |
+-----+
| 636174     |
+-----+
mysql> SELECT X'636174';
+-----+
| X'636174'  |
+-----+
| cat        |
+-----+
```

16 進数リテラルの場合、ビット演算は数値コンテキストとみなされますが、ビット演算では MySQL 8.0 以上で数値またはバイナリ文字列引数が許可されます。16 進数リテラルのバイナリ文字列コンテキストを明示的に指定するには、少なくとも次のいずれかの引数に `_binary` イントロデューサを使用します:

```
mysql> SET @v1 = X'000D' | X'0BC0';
mysql> SET @v2 = _binary X'000D' | X'0BC0';
mysql> SELECT HEX(@v1), HEX(@v2);
+-----+-----+
| HEX(@v1) | HEX(@v2) |
+-----+-----+
| BCD     | 0BCD    |
+-----+-----+
```

両方のビット操作で表示される結果は似ていますが、`_binary` のない結果は `BIGINT` 値ですが、`_binary` の結果はバイナリ文字列です。結果タイプが異なるため、表示される値は異なります: 数値の結果には、上位 0 桁は表示されません。

9.1.5 ビット値リテラル

ビット値リテラルは、`b'val'`または `Obval` 表記を使用して書き込まれます。`val` は、ゼロとゼロを使用して書き込まれるバイナリ値です。先行する `b` の大文字と小文字は関係ありません。先頭の `0b` では大/小文字が区別され、`OB` として書き込むことはできません。

有効なビット値リテラル:

```
b'01'
B'01'
0b01
```

不正なビット値リテラル:

```
b'2' (2 is not a binary digit)
0B01 (0B must be written as 0b)
```

デフォルトでは、ビット値リテラルはバイナリ文字列です:

```
mysql> SELECT b'1000001', CHARSET(b'1000001');
+-----+-----+
| b'1000001' | CHARSET(b'1000001') |
+-----+-----+
| A          | binary              |
+-----+-----+
mysql> SELECT 0b1100001, CHARSET(0b1100001);
+-----+-----+
| 0b1100001 | CHARSET(0b1100001) |
+-----+-----+
| a          | binary              |
+-----+-----+
```

ビット値リテラルには、オプションの文字セットイントロデューサおよび `COLLATE` 句を指定して、特定の文字セットおよび照合順序を使用する文字列として指定できます:

```
[_charset_name] b'val' [COLLATE collation_name]
```

例:

```
SELECT _latin1 b'1000001';
SELECT _utf8 0b1000001 COLLATE utf8_danish_ci;
```

この例では `b'val'` 表記法を使用しますが、`Obval` 表記法ではイントロデューサも許可されています。イントロデューサについては、[セクション10.3.8「文字セットイントロデューサ」](#)を参照してください。

数値コンテキストでは、MySQL はビットリテラルを整数として扱います。ビットリテラルの数値処理を保証するには、数値コンテキストで使用します。これには、0 の追加または `CAST(... AS UNSIGNED)` の使用が含まれます。たとえば、ユーザー定義変数に割り当てられるビットリテラルは、デフォルトでバイナリ文字列です。値を数値として割り当てるには、数値コンテキストで使用します:

```
mysql> SET @v1 = b'1100001';
mysql> SET @v2 = b'1100001'+0;
mysql> SET @v3 = CAST(b'1100001' AS UNSIGNED);
mysql> SELECT @v1, @v2, @v3;
+-----+-----+
| @v1 | @v2 | @v3 |
+-----+-----+
| a   | 97  | 97  |
+-----+-----+
```

空のビット値 (`b''`) は、長さゼロのバイナリ文字列に評価されます。数値に変換され、0 が生成されます:

```
mysql> SELECT CHARSET(b''), LENGTH(b'');
+-----+-----+
| CHARSET(b'') | LENGTH(b'') |
+-----+-----+
| binary      | 0           |
+-----+-----+
mysql> SELECT b''+0;
+-----+
```

```
| b"+0 |
+-----+
|  0 |
+-----+
```

ビット値表記は、BIT カラムに割り当てる値を指定する場合に便利です:

```
mysql> CREATE TABLE t (b BIT(8));
mysql> INSERT INTO t SET b = b'11111111';
mysql> INSERT INTO t SET b = b'1010';
mysql> INSERT INTO t SET b = b'0101';
```

結果セットのビット値はバイナリ値として返されますが、適切に表示されない場合があります。ビット値を印刷可能な形式に変換するには、数値コンテキストで使用するか、BIN() や HEX() などの変換関数を使用します。変換された値に上位 0 桁は表示されません。

```
mysql> SELECT b+0, BIN(b), OCT(b), HEX(b) FROM t;
+-----+-----+-----+-----+
| b+0 | BIN(b) | OCT(b) | HEX(b) |
+-----+-----+-----+-----+
| 255 | 11111111 | 377 | FF |
| 10 | 1010 | 12 | A |
| 5 | 101 | 5 | 5 |
+-----+-----+-----+-----+
```

ビットリテラルの場合、ビット操作は数値コンテキストとみなされますが、ビット操作では MySQL 8.0 以上で数値またはバイナリ文字列引数が許可されます。ビットリテラルのバイナリ文字列コンテキストを明示的に指定するには、少なくとも次のいずれかの引数に `_binary` イントロデューサを使用します:

```
mysql> SET @v1 = b'000010101' | b'000101010';
mysql> SET @v2 = _binary b'000010101' | _binary b'000101010';
mysql> SELECT HEX(@v1), HEX(@v2);
+-----+-----+
| HEX(@v1) | HEX(@v2) |
+-----+-----+
| 3F | 003F |
+-----+-----+
```

両方のビット操作で表示される結果は似ていますが、`_binary` のない結果は BIGINT 値ですが、`_binary` の結果はバイナリ文字列です。結果タイプが異なるため、表示される値は異なります: 数値の結果には、上位 0 桁は表示されません。

9.1.6 boolean リテラル

TRUE および FALSE 定数はそれぞれ 1 と 0 として評価されます。定数名は大文字でも小文字でも記述できます。

```
mysql> SELECT TRUE, true, FALSE, false;
-> 1, 1, 0, 0
```

9.1.7 NULL 値

NULL 値は「データなし」を意味します。NULL は大文字と小文字のどちらでも記述できます。

NULL 値は、数値型での 0 や文字列型での空文字列などの値とは異なります。詳細は、[セクション B.3.4.3 「NULL 値に関する問題」](#) を参照してください。

LOAD DATA または SELECT ... INTO OUTFILE で実行されるテキストファイルのインポートまたはエクスポート操作の場合、NULL は \N 順序で表されます。[セクション 13.2.7 「LOAD DATA ステートメント」](#) を参照してください。

ORDER BY でソートする場合、NULL 値は、昇順ソートの場合は他の値の前にソートされ、降順ソートの場合は他の値の後にソートされます。

9.2 スキーマオブジェクト名

データベース、テーブル、インデックス、カラム、エイリアス、ビュー、ストアドプロシージャ、パーティション、テーブルスペース、リソースグループ、その他のオブジェクト名など、MySQL 内の特定のオブジェクトは識別子と

呼ばれます。このセクションでは、MySQL で使用可能な識別子の構文について説明します。[セクション9.2.1「識別子の長さ制限」](#)は、各タイプの識別子の最大長を示します。[セクション9.2.3「識別子の大文字と小文字の区別」](#)では、大/小文字が区別される識別子のタイプとその条件について説明します。

識別子は引用符で囲むことも囲まないこともあります。識別子に特殊文字が含まれている場合、または識別子が予約語である場合、その識別子を参照するときは必ず引用符で囲む必要があります。(例外: 修飾名内でピリオドのあとに続く予約語は識別子である必要があるため、引用符で囲む必要はありません。) 予約語は[セクション9.3「キーワードと予約語」](#)に記載されています。

内部的に、識別子は Unicode (UTF-8) に変換され、格納されます。識別子に使用できる Unicode 文字は、Basic Multilingual Plane (BMP) の Unicode 文字です。補助文字は許可されません。したがって、識別子には次の文字を含めることができます:

- 引用符で囲まれていない識別子で許可される文字。
 - ASCII: [0-9,a-z,A-Z\$_] (基本的なラテン文字、0-9 の数字、ドル、下線)
 - 拡張: U+0080 .. U+FFFF
- 引用符で囲まれている識別子で許可される文字には、U+0000 を除き、完全な Unicode Basic Multilingual Plane (BMP) が含まれます。
 - ASCII: U+0001 .. U+007F
 - 拡張: U+0080 .. U+FFFF
- ASCII NUL (U+0000) と補助文字 (U+10000 以上) は、引用符で囲まれた識別子または引用符で囲まれていない識別子では許可されません。
- 識別子は数字で始めることができますが、引用符で囲まれていないかぎり、数字のみで構成することはできません。
- データベース名、テーブル名、およびカラム名は、空白文字で終わることはできません。

識別子引用符文字は逆引用符 (`) です。

```
mysql> SELECT * FROM `select` WHERE `select`.id > 100;
```

[ANSI_QUOTES](#) SQL モードが有効になっている場合、二重引用符内で識別子を引用符で囲むことも許可されています。

```
mysql> CREATE TABLE "test" (col INT);
ERROR 1064: You have an error in your SQL syntax...
mysql> SET sql_mode=ANSI_QUOTES;
mysql> CREATE TABLE "test" (col INT);
Query OK, 0 rows affected (0.00 sec)
```

[ANSI_QUOTES](#) モードでは、サーバーは二重引用符で囲まれた文字列を識別子として解釈します。この結果、このモードが有効になっているときには、文字列リテラルは単一引用符で囲む必要があります。二重引用符で囲むことはできません。サーバー SQL モードは、[セクション5.1.11「サーバー SQL モード」](#)で説明しているように制御されません。

識別子が引用符で囲まれていれば、識別子引用符文字を識別子内に含めることができます。識別子内に含める文字が識別子自体を囲むのに使用している引用符と同じ場合、文字を二重にする必要があります。次のステートメントは、`c`d` という名前のカラムを含んだ `a`b` という名前のテーブルを作成します。

```
mysql> CREATE TABLE `a`b` (`c`d` INT);
```

クエリーの選択リストで、引用したカラムエイリアスを指定するには、識別子または文字列引用文字を使用します。

```
mysql> SELECT 1 AS `one`, 2 AS `two`;
+----+----+
| one | two |
+----+----+
| 1 | 2 |
+----+----+
```


ステートメント内のどこに指定する場合でも、エイリアスへの引用した参照には、識別子引用符を使用する必要があります。そうしないと、参照は文字列リテラルとして扱われます。

Me や MeN (この場合の M と N は整数) で始まる名前を使用しないことが推奨されています。たとえば、1e を識別子として使用しないでください。これは、1e+3 などの式があいまいになるためです。コンテキストに応じて、式 1e + 3 として、または数値 1e+3 として解釈される場合があります。

テーブル名前を作成するのに MD5() を使用する場合は注意が必要です。なぜなら、これは、前述のような不正な形式やあいまいな形式で名前を生成する可能性があるからです。

ユーザー変数は、識別子または識別子の一部として SQL ステートメントの中で直接使用することはできません。回避策の詳細と例については、[セクション9.4「ユーザー定義変数」](#)を参照してください。

[セクション9.2.4「識別子とファイル名のマッピング」](#)で説明しているように、データベース名とテーブル名内の特殊文字は、対応するファイルシステム名でエンコードされます。

9.2.1 識別子の長さ制限

次の表には、識別子のタイプごとの最大の長さが示されています。

識別子タイプ	最大の長さ (文字)
Database	64 (NDB Cluster 8.0.18 以降を含む)
Table	64 (NDB Cluster 8.0.18 以降を含む)
カラム	64
インデックス	64
制約	64
ストアドプログラム	64
ビュー	64
テーブルスペース	64
サーバー	64
ログファイルグループ	64
エイリアス	256 (表のあとの例外を参照してください)
複合ステートメントラベル	16
ユーザー定義変数	64
リソースグループ	64

CREATE VIEW ステートメント内のカラム名に対するエイリアスは、(256 文字の最大のエイリアス長ではなく) 64 文字の最大のカラム長に対してチェックされます。

制約名を含まない制約定義の場合、サーバーは関連付けられたテーブル名から導出された名前を内部的に生成します。たとえば、内部的に生成された外部キーおよび **CHECK** 制約名は、テーブル名に `_ibfk_` または `_chk_` と数値を加えたもので構成されます。テーブル名が制約名の長さ制限に近い場合、制約名に追加の文字が必要になると、その名前が制限を超える可能性があるため、エラーが発生します。

識別子は Unicode (UTF-8) を使用して格納されます。これは、テーブル定義の識別子および **mysql** データベースの付与テーブルに格納されている識別子に適用されます。付与テーブル内の識別子文字列カラムのサイズは文字数で測定されます。マルチバイト文字は、これらのカラムに格納されている値に許可される文字数を減らすことなく使用できます。

NDB 8.0.18 より前では、NDB Cluster はデータベースおよびテーブルの名前に 63 文字の最大長を課していました。NDB 8.0.18 の時点では、この制限は削除されています。[セクション23.1.7.11「前 NDB Cluster 8.0 で解決される NDB Cluster の問題」](#)を参照してください。

MySQL アカウント名のユーザー名やホスト名などの値は、識別子ではなく文字列です。付与テーブルに格納されるこのような値の最大長については、[付与テーブルのスコープカラムのプロパティ](#)を参照してください。

9.2.2 識別子の修飾子

オブジェクト名は、修飾されていない場合と修飾されている場合があります。名前の解釈があいまいでないコンテキストでは、修飾されていない名前を使用できます。修飾名には、デフォルトコンテキストをオーバーライドするか、欠落しているコンテキストを提供することによって解釈コンテキストを明確にするための修飾子が少なくとも1つ含まれています。

たとえば、次のステートメントは、修飾されていない名前 `t1` を使用してテーブルを作成します：

```
CREATE TABLE t1 (i INT);
```

`t1` にはデータベースを指定する修飾子が含まれていないため、このステートメントはデフォルトデータベースにテーブルを作成します。デフォルトのデータベースがない場合は、エラーが発生します。

次のステートメントは、修飾名 `db1.t1` を使用してテーブルを作成します：

```
CREATE TABLE db1.t1 (i INT);
```

`db1.t1` にはデータベース修飾子 `db1` が含まれているため、このステートメントは、デフォルトのデータベースに関係なく、`db1` という名前のデータベースに `t1` を作成します。デフォルトのデータベースがない場合は、修飾子を指定する必要があります。修飾子は、デフォルトデータベースがある場合、デフォルトとは異なるデータベースを指定する場合、またはデフォルトが指定したものと同一場合にデータベースを明示的にする場合に指定できます。

クオリファイアには次の特性があります：

- 修飾されていない名前は単一の識別子で構成されます。修飾名は複数の識別子で構成されます。
- 複数部分名のコンポーネントは、ピリオド (.) 文字で区切る必要があります。マルチパート名の最初の部分は、最終識別子を解釈するコンテキストに影響する修飾子として機能します。
- 修飾子文字は別個のトークンであり、関連付けられた識別子と隣接する必要はありません。たとえば、`tbl_name.col_name` と `tbl_name . col_name` は同等です。
- 複数部分名のコンポーネントを引用符で囲む必要がある場合、名前全体を引用符で囲むのではなく、各コンポーネントを個別に引用符で囲んでください。たとえば、``my-table.my-column`` ではなく、``my-table`.`my-column`` と記述します。
- 修飾名内でピリオドのあとに続く予約語は識別子である必要があるため、そのコンテキストでは引用符で囲む必要はありません。

オブジェクト名に使用できる修飾子は、オブジェクトタイプによって異なります：

- データベース名は完全修飾名であり、修飾子を取りません：

```
CREATE DATABASE db1;
```

- テーブル、ビューまたはストアプログラム名には、データベース名修飾子を指定できます。CREATE ステートメントの修飾されていない名前と修飾名の例：

```
CREATE TABLE mytable ...;
CREATE VIEW myview ...;
CREATE PROCEDURE myproc ...;
CREATE FUNCTION myfunc ...;
CREATE EVENT myevent ...;
```

```
CREATE TABLE mydb.mytable ...;
CREATE VIEW mydb.myview ...;
CREATE PROCEDURE mydb.myproc ...;
CREATE FUNCTION mydb.myfunc ...;
CREATE EVENT mydb.myevent ...;
```

- トリガーはテーブルに関連付けられるため、任意の修飾子がテーブル名に適用されます：

```
CREATE TRIGGER mytrigger ... ON mytable ...;
```

```
CREATE TRIGGER mytrigger ... ON mydb.mytable ...;
```

- 次のテーブルに示すように、カラム名には、そのカラムを参照するステートメントのコンテキストを示す複数の修飾子を指定できます。

カラム参照	意味
<code>col_name</code>	ステートメントで使用されるいずれかのテーブルのカラム <code>col_name</code> にその名前のカラムが含まれている場合
<code>tbl_name.col_name</code>	デフォルトデータベースのテーブル <code>tbl_name</code> のカラム <code>col_name</code>
<code>db_name.tbl_name.col_name</code>	データベース <code>db_name</code> のテーブル <code>tbl_name</code> のカラム <code>col_name</code>

つまり、カラム名にはテーブル名修飾子を指定でき、テーブル名修飾子自体にはデータベース名修飾子を指定できません。SELECT ステートメントでの修飾されていないカラム参照および修飾されたカラム参照の例:

```
SELECT c1 FROM mytable
WHERE c2 > 100;

SELECT mytable.c1 FROM mytable
WHERE mytable.c2 > 100;

SELECT mydb.mytable.c1 FROM mydb.mytable
WHERE mydb.mytable.c2 > 100;
```

修飾されていない参照があいまいでないかぎり、ステートメントでオブジェクト参照の修飾子を指定する必要はありません。カラム `c1` がテーブル `t1`、`t2` のみの `c2` および `t1` と `t2` の両方の `c` に出現するとします。 `c` への修飾されていない参照は、両方のテーブルを参照するステートメントであいまいであり、どのテーブルを意味するかを示すために `t1.c` または `t2.c` として修飾する必要があります:

```
SELECT c1, c2, t1.c FROM t1 INNER JOIN t2
WHERE t2.c > 100;
```

同様に、データベース `db1` のテーブル `t` および同じステートメントのデータベース `db2` のテーブル `t` から取得するには、テーブル参照を修飾する必要があります: これらのテーブルのカラムを参照する場合、修飾子は両方のテーブルに表示されるカラム名にのみ必要です。カラム `c1` がテーブル `db1.t`、`db2.t` のみの `c2` および `db1.t` と `db2.t` の両方の `c` に出現するとします。この場合、`c` はあいまいであり、修飾する必要がありますが、`c1` および `c2` は次のようにする必要はありません:

```
SELECT c1, c2, db1.t.c FROM db1.t INNER JOIN db2.t
WHERE db2.t.c > 100;
```

テーブルのエイリアスを使用すると、修飾カラム参照をより簡単に書き込むことができます:

```
SELECT c1, c2, t1.c FROM db1.t AS t1 INNER JOIN db2.t AS t2
WHERE t2.c > 100;
```

9.2.3 識別子の小文字と大文字の区別

MySQL において、データベースはデータディレクトリ内のディレクトリに対応しています。データベース内の各テーブルも、データベースディレクトリ内の少なくとも 1 つ (ストレージエンジンによってはそれ以上) のファイルに対応しています。トリガーもファイルに対応しています。この結果、基になるオペレーティングシステムで大文字と小文字が区別されるかどうか、データベース名、テーブル名、およびトリガー名で大文字と小文字が区別されるかどうかに影響します。つまり、このような名前前は Windows では大文字と小文字が区別されませんが、ほとんどの種類の Unix では大文字と小文字が区別されます。特に重要な例外は、Unix ベースであるが、大/小文字を区別しないデフォルトのファイルシステムタイプ (HFS+) を使用する macOS です。ただし、macOS は UFS ボリュームもサポートします。UFS ボリュームでは、Unix と同様に大文字と小文字が区別されます。 [セクション 1.7.1「標準 SQL に対する MySQL 拡張機能」](#) を参照してください。このセクションで後述するように、`lower_case_table_names` システム変数も、サーバーが識別子の小文字と大文字をどのように扱うかに影響を与えます。

注記

一部のプラットフォームでは、データベース名、テーブル名およびトリガー名の大/小文字は区別されませんが、同じステートメント内で異なる大/小文字を使用してこれらのい

れかを参照しないでください。次のステートメントは、同じテーブルを `my_table` および `MY_TABLE` として参照するため、機能しません。

```
mysql> SELECT * FROM my_table WHERE MY_TABLE.col=1;
```

パーティション名、サブパーティション名、カラム名、インデックス名、ストアドルーチン名、イベント名およびリソースグループ名は、プラットフォームでは大/小文字が区別されず、カラムのエイリアスでもありません。

ただし、ログファイルグループの名前では大/小文字が区別されます。これは標準 SQL とは異なります。

デフォルトでは、テーブルのエイリアスは Unix では大/小文字が区別されますが、Windows または macOS では区別されません。Unix では、次のステートメントは、エイリアスを `a` と `A` の両方で参照しているため機能しません。

```
mysql> SELECT col_name FROM tbl_name AS a
WHERE a.col_name = 1 OR A.col_name = 2;
```

ただし、Windows ではこの同じステートメントは許可されます。このような違いで生じる問題を回避するために、データベースとテーブルの作成および参照では常に小文字の名前を使用するなどの一貫した規則を設けることをお勧めします。この規則は移植性と使いやすさを最大限にするために推奨されています。

テーブル名およびデータベース名をディスクに格納して MySQL で使用する方法は、`lower_case_table_names` システム変数の影響を受けます。`lower_case_table_names` は、次のテーブルに示す値を取ることができます。この変数は、トリガー識別子の小文字と大文字の区別には影響を及ぼしません。Unix では、`lower_case_table_names` のデフォルト値は 0 です。Windows, では、デフォルト値は 1 です。macOS では、デフォルト値は 2 です。

`lower_case_table_names` は、サーバーの初期化時にのみ構成できます。サーバーの初期化後の `lower_case_table_names` 設定の変更は禁止されています。

値	意味
0	テーブル名とデータベース名は、 <code>CREATE TABLE</code> または <code>CREATE DATABASE</code> ステートメントで指定された大文字または小文字を使用してディスク上に格納されます。名前の比較では大文字と小文字が区別されます。大/小文字を区別しないファイル名 (Windows や macOS など) を持つシステムで MySQL を実行している場合は、この変数を 0 に設定しないでください。大文字と小文字を区別しないファイルシステムで <code>--lower-case-table-names=0</code> を使用して強制的にこの変数を 0 に設定し、大文字と小文字を変えて <code>MyISAM</code> テーブル名にアクセスした場合、インデックスが破損することがあります。
1	テーブル名はディスクに小文字で格納され、名前の比較では大/小文字は区別されません。MySQL では、保存およびバックアップ時にすべてのテーブル名が小文字に変換されます。この動作はデータベース名やテーブルエイリアスにも適用されます。
2	テーブル名とデータベース名は、 <code>CREATE TABLE</code> または <code>CREATE DATABASE</code> ステートメントで指定された大文字または小文字を使用してディスク上に格納されますが、MySQL ではバックアップ時に小文字に変換されます。名前の比較では、大文字と小文字は区別されません。これは、大/小文字が区別されないファイルシステムでのみを動作させます。InnoDB のテーブル名およびビュー名は、 <code>lower_case_table_names=1</code> の場合と同様に小文字で格納されます。

MySQL を単一のプラットフォームでのみ使用している場合は、通常、デフォルト以外の `lower_case_table_names` 設定を使用する必要はありません。ただし、ファイルシステム上の大文字と小文字の区別が異なるプラットフォーム間でテーブルを転送する場合は、問題が生じる可能性があります。たとえば、Unix 上では `my_table` と `MY_TABLE` という名前の異なる 2 つのテーブルを使用できますが、Windows 上ではこれらは同一のものとして扱われます。データベース名やテーブル名の大文字と小文字の区別が原因で発生するデータ転送の問題を回避するには、次の 2 つのオプションがあります。

- `lower_case_table_names=1` を全システムで使用してください。これの主な欠点は、`SHOW TABLES` または `SHOW DATABASES` を使用したときに、元の大文字または小文字で名前が表示されないことです。
- Unix 上では `lower_case_table_names=0` を、Windows 上では `lower_case_table_names=2` を使用してください。これでデータベース名とテーブル名の大文字と小文字の区別が保持されます。この欠点は、ユーザーのステートメントが、Windows 上で正しい大文字または小文字でデータベース名およびテーブル名を常に参照していることを確認する必要があります。大文字と小文字が区別される Unix にステートメントを転送する場合、大文字と小文字が正しくなければこのステートメントは機能しません。

例外: InnoDB テーブルを使用していて、これらのデータ転送の問題を回避しようとしている場合は、すべてのプラットフォームで `lower_case_table_names=1` を使用して名前を強制的に小文字に変換する必要があります。

バイナリ照合順序に応じて大文字形式が同等である場合、オブジェクト名は複製と見なされる場合があります。これは、カーソル、条件、プロシージャ、関数、セーブポイント、ストアドルーチンパラメータ、ストアプログラムローカル変数、およびプラグインの名前にも当てはまります。カラム、制約、データベース、パーティション、`PREPARE` を使用して準備されたステートメント、テーブル、トリガー、ユーザー、およびユーザー定義変数の名前には当てはまりません。

ファイルシステムの大文字と小文字の区別は、`INFORMATION_SCHEMA` テーブルの文字列カラムでの検索に影響する場合があります。詳細は、[セクション10.8.7「INFORMATION_SCHEMA 検索での照合の使用」](#)を参照してください。

9.2.4 識別子とファイル名のマッピング

データベース識別子やテーブル識別子とファイルシステム内の名前との間には対応があります。基本構造では、MySQL は各データベースをデータディレクトリ内のディレクトリとして表し、ストレージエンジンに応じて、各テーブルは適切なデータベースディレクトリ内の 1 つ以上のファイルによって表されます。

データファイルとインデックスファイルの場合、ディスク上の正確な表現はストレージエンジンによって異なります。これらのファイルは、データベースディレクトリに格納することも、別のファイルに格納することもできます。InnoDB データは InnoDB データファイルに格納されます。InnoDB でテーブルスペースを使用する場合は、新たに作成した特定のテーブルスペースファイルが代わりに使用されます。

ASCII NUL (`X'00'`) を除くすべての文字は、データベース識別子またはテーブル識別子で有効です。MySQL では、データベースディレクトリやテーブルファイルを作成するとき、対応するファイルシステムオブジェクト内で問題のある文字をすべてエンコードします。

- 基本的なラテン文字 (`a..zA..Z`)、数字 (`0..9`)、および下線 (`_`) はそのままエンコードされます。このため、それらが小文字と大文字を区別するかどうかは、ファイルシステムの特性に直接依存します。
- 大文字と小文字のマッピングを持つアルファベット起源のほかの国の文字はすべて、次の表に示すようにエンコードされます。コード範囲カラムの値は UCS-2 値です。

コード範囲	パターン	数値	使用	未使用	ブロック
00C0..017F	<code>[@][0..4][g..z]</code>	$5 \times 20 = 100$	97	3	補足ラテン語-1 + 拡張ラテン語-A
0370..03FF	<code>[@][5..9][g..z]</code>	$5 \times 20 = 100$	88	12	ギリシア語およびコプト語
0400..052F	<code>[@][g..z][0..6]</code>	$20 \times 7 = 140$	137	3	キリル文字 + 補足キリル文字
0530..058F	<code>[@][g..z][7..8]</code>	$20 \times 2 = 40$	38	2	米語
2160..217F	<code>[@][g..z][9]</code>	$20 \times 1 = 20$	16	4	数の形式
0180..02AF	<code>[@][g..z][a..k]</code>	$20 \times 11 = 220$	203	17	拡張ラテン語-B + 拡張 IPA
1E00..1EFF	<code>[@][g..z][l..r]</code>	$20 \times 7 = 140$	136	4	拡張ラテン語追加
1F00..1FFF	<code>[@][g..z][s..z]</code>	$20 \times 8 = 160$	144	16	拡張ギリシア語
.....	<code>[@][a..f][g..z]</code>	$6 \times 20 = 120$	0	120	RESERVED

コード範囲	パターン	数値	使用	未使用	ブロック
24B6..24E9	[@][@][a..z]	26	26	0	囲み文字
FF21..FF5A	[@][a..z][@]	26	26	0	全角と半角

シーケンス内の 1 バイトが大文字と小文字の区別をエンコードします。例: `LATIN CAPITAL LETTER A WITH GRAVE` は `@0G` としてエンコードされ、`LATIN SMALL LETTER A WITH GRAVE` は `@0g` としてエンコードされます。ここでは、3 番目のバイト (`G` または `g`) が大文字と小文字の区別を示します。(大/小文字を区別しないファイルシステムでは、両方の文字が同じとして扱われます。)

言語ブロックの中にはキリル文字のように、2 番目のバイトが大文字と小文字の区別を決定することもあります。補足ラテン語 1 などのほかの言語ブロックでは、3 番目のバイトが大文字と小文字の区別を決定します。シーケンス内の 2 バイトが文字の場合は (拡張ギリシャ語など)、いちばん左の文字が大文字と小文字の区別を表します。ほかの文字バイトはすべて、小文字である必要があります。

- アンダースコア (`_`) 以外のすべての非文字および大文字/小文字のマッピングを持たないアルファベット (ヘブライ語など) の文字は、16 進数の `a..f` に小文字を使用して 16 進数表現を使用してエンコードされます:

```
0x003F -> @003f
0xFFFF -> @ffff
```

16 進値は、`ucs2` ダブルバイト文字セット内のキャラクタ値に対応します。

Windows では、`nul`、`prn`、`aux` などの一部の名前は、サーバーが対応するファイルまたはディレクトリを作成するときに、`@@@` を名前に付加することによってエンコードされます。これは、対応するデータベースオブジェクトのプラットフォーム間での移植性のためにすべてのプラットフォームで行われます。

9.2.5 関数名の構文解析と解決

MySQL は、組み込み (ネイティブ) 関数、ユーザー定義関数 (UDF) およびストアドファンクションをサポートしています。このセクションでは、組み込み関数名が関数呼び出しとして使用されているか、識別子として使用されているかをサーバーで認識する方法と、指定された名前と異なる型の関数が存在する場合に使用する関数をサーバーが特定する方法について説明します。

- [組み込み関数名の構文解析](#)
- [関数名の解決](#)

組み込み関数名の構文解析

パーサーは組み込み関数名を構文解析するためのデフォルトのルールを使用します。これらのルールは `IGNORE_SPACE` SQL モードを有効にすることで変更できます。

パーサーは、組み込み関数の名前である単語を検出すると、その名前が関数呼び出しを示しているのか、それともテーブル名やカラム名などの識別子の式以外の参照であるのかを判別する必要があります。たとえば、次のステートメントでは `count` に対する最初の参照は関数呼び出しですが、2 番目の参照はテーブル名です。

```
SELECT COUNT(*) FROM mytable;
CREATE TABLE count (i INT);
```

パーサーは、式であると予想される対象を解析しているときにのみ、組み込み関数の名前を、関数呼び出しを示している名前として認識する必要があります。つまり、式以外のコンテキストでは、関数名は識別子として許可されません。

ただし、組み込み関数の中には構文解析や実装に関する特別な考慮事項を含んでいるものがあるため、パーサーはデフォルトで次のルールに従って、その名前が関数呼び出しとして使用されているのか、それとも式以外のコンテキストで識別子として使用されているのかを判別します。

- 式の中で関数呼び出しとして名前を使用するには、名前とそれに続く括弧文字 (の間に空白が存在しないことが必要です)。
- 反対に、関数名を識別子として使用するには、括弧文字を直後に続けないでください。

名前と括弧の間に空白を入れずに関数呼び出しを記述するという要件は、特別な考慮事項を持つ組み込み関数にのみ適用されます。COUNT がこのような名前の 1 つです。sql/lex.h ソースファイルには、次の空白によって解釈が決定されるこれらの特殊関数の名前がリストされます: symbols[] 配列の SYM_FN() マクロによって定義された名前。

次のリストに、IGNORE_SPACE 設定の影響を受け、sql/lex.h ソースファイルに特殊としてリストされる MySQL 8.0 の関数の名前を示します。非ヒットスペース要件をすべての関数コールに適用するのが最も簡単な場合があります。

- ADDDATE
- BIT_AND
- BIT_OR
- BIT_XOR
- CAST
- COUNT
- CURDATE
- CURTIME
- DATE_ADD
- DATE_SUB
- EXTRACT
- GROUP_CONCAT
- MAX
- MID
- MIN
- NOW
- POSITION
- SESSION_USER
- STD
- STDDEV
- STDDEV_POP
- STDDEV_SAMP
- SUBDATE
- SUBSTR
- SUBSTRING
- SUM
- SYSDATE
- SYSTEM_USER
- TRIM
- VARIANCE
- VAR_POP

- `VAR_SAMP`

`sql/lex.h` で特殊としてリストされていない関数の場合、空白は関係ありません。それらは式のコンテキストで使用されるときにのみ関数呼び出しとして解釈され、それ以外では識別子として自由に使用できます。ASCII がこのような名前の 1 つです。ただし、このような影響を受けない関数名に対する解釈は、式のコンテキストによって変わることがあります。つまり、`func_name ()` は、指定の名前が単独で使用される場合は組み込み関数として解釈されますが、単独ではない場合、`func_name ()` はユーザー定義関数またはストアドファンクション (その名前の関数が存在する場合) として解釈されます。

`IGNORE_SPACE` SQL モードは、空白の有無で区別される関数名をパーサーで扱う方法を変更するために使用できません。

- `IGNORE_SPACE` が無効になっていると、名前と後続の括弧の間に空白がない場合、パーサーはその名前を関数呼び出しと解釈します。これは、関数名が式以外のコンテキストで使用されているときにも行われます。

```
mysql> CREATE TABLE count(i INT);
ERROR 1064 (42000): You have an error in your SQL syntax ...
near 'count(i INT)'
```

エラーを取り除き、名前が識別子として扱われるようにするには、名前のあとに続く空白を使うか、引用符で囲んだ識別子として記述してください (あるいはこの両方)。

```
CREATE TABLE count (i INT);
CREATE TABLE `count` (i INT);
CREATE TABLE `count` (i INT);
```

- `IGNORE_SPACE` が有効になっていると、パーサーは関数名と後続の括弧の間に空白は存在しないという要件を緩和します。このことで、関数呼び出しの記述がより自由に行えるようになります。たとえば、次のどちらの関数呼び出しも有効です。

```
SELECT COUNT(*) FROM mytable;
SELECT COUNT (*) FROM mytable;
```

ただし、`IGNORE_SPACE` を有効にした場合、影響を受ける関数名をパーサーが予約語として扱うという副作用も生じます (セクション 9.3 「キーワードと予約語」を参照してください)。つまり、名前のあとに続く空白は、それが識別子として使用されることを示すものではなくなくなります。後続の空白の有無を問わず、名前は関数呼び出しとして使用できますが、引用符で囲まれていないかぎり、式以外のコンテキストでは構文エラーが発生します。たとえば、`IGNORE_SPACE` が有効になっていると、パーサーが `count` を予約語として解釈するため、次のステートメントはどちらも構文エラーが表示されて失敗します。

```
CREATE TABLE count(i INT);
CREATE TABLE count (i INT);
```

式以外のコンテキストで関数名を使用するには、これを引用符で囲まれた識別子として記述します。

```
CREATE TABLE `count` (i INT);
CREATE TABLE `count` (i INT);
```

`IGNORE_SPACE` SQL モードを有効にするには、次のステートメントを使用します。

```
SET sql_mode = 'IGNORE_SPACE';
```

`IGNORE_SPACE` は、ANSI などのほかの特定のコンポジットモードの値に含められている場合にも有効になります。

```
SET sql_mode = 'ANSI';
```

どのコンポジットモードが `IGNORE_SPACE` を有効にするかを調べるには、セクション 5.1.11 「サーバー SQL モード」を参照してください。

`IGNORE_SPACE` 設定における SQL コードの依存関係を最小にするには、以下のガイドラインを使用してください。

- 組み込み関数と同じ名前の UDF またはストアドファンクションは作成しないでください。
- 式以外のコンテキストでは関数名を使用しないでください。たとえば、これらのステートメントは、`count` (`IGNORE_SPACE` の影響を受ける関数名の 1 つ) を使用するため、`IGNORE_SPACE` が有効であれば、名前に続く空白の有無によらずこれらのステートメントは失敗します。

```
CREATE TABLE count(i INT);
CREATE TABLE count (i INT);
```

式以外のコンテキストで関数名を使用する必要がある場合は、これを引用符で囲まれた識別子として記述します。

```
CREATE TABLE `count` (i INT);
CREATE TABLE `count` (i INT);
```

関数名の解決

次のルールは、関数の作成と呼び出しのために、サーバーで関数名の参照を解決する方法について記述します。

- 組み込み関数とユーザー定義関数

組み込み関数と同じ名前でも UDF を作成しようとした場合、エラーが発生します。

- 組み込み関数とストアードファンクション

組み込み関数と同名のストアードファンクションを作成することは可能ですが、このストアードファンクションを呼び出すにはスキーマ名で修飾する必要があります。たとえば、`test` スキーマに `PI` という名前のストアードファンクションを作成した場合、サーバーは組み込み関数への参照として修飾子なしで `PI()` を解決するため、それを `test.PI()` として起動します。ストアードファンクション名が組み込み関数名と競合する場合、サーバーは警告を生成します。この警告は `SHOW WARNINGS` で表示できます。

- ユーザー定義関数とストアードファンクション

ユーザー定義関数とストアードファンクションは同じ名前空間を共有します。したがって、同名の UDF とストアードファンクションを作成することはできません。

前述の関数名の解決ルールは、新しい組み込み関数を実装する MySQL のバージョンへのアップグレードに影響を及ぼします。

- 指定の名前のユーザー定義関数がすでに作成されており、同じ名前でも新しい組み込み関数を実装するバージョンに MySQL をアップグレードすると、この UDF にはアクセスできなくなります。これを修正するには、`DROP FUNCTION` を使用して UDF および `CREATE FUNCTION` を削除し、競合しない別の名前でも UDF を再作成します。次に、影響を受けるコードを変更して新しい名前を使用します。
- 新しいバージョンの MySQL で、既存のストアードファンクションと同じ名前でも組み込み関数を実装する場合、ストアードファンクションの名前を競合しない名前に変更するか、スキーマ修飾子を使用するように関数の呼び出しを変更する（つまり、`schema_name.func_name()` 構文を使用する）という 2 つの選択肢があります。いずれの場合も、影響を受けるコードを適宜変更します。

9.3 キーワードと予約語

キーワードは、SQL で意味を持つ単語です。`SELECT`、`DELETE`、`BIGINT` などの特定のキーワードは予約されており、テーブル名やカラム名などの識別子として使用するための特別な処理が必要です。これは、組み込み関数の名前にも当てはまる場合があります。

予約されていないキーワードは、引用符なしで識別子として使用できます。[セクション9.2「スキーマオブジェクト名」](#)で説明しているように、予約語は、引用符で囲まれている場合、識別子として許可されます。

```
mysql> CREATE TABLE interval (begin INT, end INT);
ERROR 1064 (42000): You have an error in your SQL syntax ...
near 'interval (begin INT, end INT)'
```

`BEGIN` および `END` はキーワードですが予約されていないため、識別子として使用する際に引用符を付ける必要はありません。`INTERVAL` は予約済キーワードであり、識別子として使用するには引用符で囲む必要があります：

```
mysql> CREATE TABLE `interval` (begin INT, end INT);
Query OK, 0 rows affected (0.01 sec)
```

例外: 修飾名でピリオドに続く語は識別子のため、予約されていても引用符で囲む必要はありません。

```
mysql> CREATE TABLE mydb.interval (begin INT, end INT);
```

Query OK, 0 rows affected (0.01 sec)

組み込み関数名は識別子として許可されますが、識別子として使用する場合は注意してください。たとえば、`COUNT` はカラム名として許容されます。ただし、デフォルトでは、関数呼び出しにおいて、関数名と後続の(文字の間に空白を入れないことが許可されています。この要件によって、パーサーはその名前が関数の呼び出しで使用されるのか、それとも関数でないコンテキストで使用されるのかを判別できます。関数名の認識の詳細は、[セクション 9.2.5 「関数名の構文解析と解決」](#) を参照してください。

`INFORMATION_SCHEMA.KEYWORDS` テーブルには、MySQL で考慮されるキーワードがリストされ、予約されているかが示されます。[セクション 26.18 「INFORMATION_SCHEMA KEYWORDS テーブル」](#) を参照してください。

- [MySQL 8.0 のキーワードおよび予約語](#)
- [MySQL 8.0 の新しいキーワードおよび予約語](#)
- [MySQL 8.0 で削除されたキーワードおよび予約語](#)

MySQL 8.0 のキーワードおよび予約語

次のリストは、MySQL 8.0 のキーワードと予約語、および個々の単語のバージョンごとの変更を示しています。予約済キーワードは (R) でマークされます。さらに、`_FILENAME` も予約されています。

今後バージョンアップするときのことを考慮に入れて、使用予定のある予約語を確認しておくことをお勧めします。新しいバージョンの MySQL を扱ったマニュアルでもこれらを確認できます。リスト内の予約語のほとんどは、標準 SQL によってカラム名またはテーブル名として禁止されています (例: `GROUP`)。いくつかは、MySQL が必要とし、`yacc` パーサーを使用するので予約されています。

[A](#) | [B](#) | [C](#) | [D](#) | [E](#) | [F](#) | [G](#) | [H](#) | [I](#) | [J](#) | [K](#) | [L](#) | [M](#) | [N](#) | [O](#) | [P](#) | [Q](#) | [R](#) | [S](#) | [T](#) | [U](#) | [V](#) | [W](#) | [X](#) | [Y](#) | [Z](#)

A

- [ACCESSIBLE](#) (R)
- [ACCOUNT](#)
- [ACTION](#)
- [ACTIVE](#); 追加されたバージョン: 8.0.14 (予約なし)
- [ADD](#) (R)
- [ADMIN](#); 予約されていない 8.0.12
- [AFTER](#)
- [AGAINST](#)
- [AGGREGATE](#)
- [ALGORITHM](#)
- [ALL](#) (R)
- [ALTER](#) (R)
- [ALWAYS](#)
- [ANALYSE](#); 削除されたバージョン: 8.0.1
- [ANALYZE](#) (R)
- [AND](#) (R)
- [ANY](#)

- [ARRAY](#); 追加されたバージョン: 8.0.17 (reserved); 予約されていない 8.0.19
- [AS](#) (R)
- [ASC](#) (R)
- [ASCII](#)
- [ASENSITIVE](#) (R)
- [AT](#)
- [ATTRIBUTE](#); 追加されたバージョン: 8.0.21 (予約なし)
- [AUTOEXTEND_SIZE](#)
- [AUTO_INCREMENT](#)
- [AVG](#)
- [AVG_ROW_LENGTH](#)

B

- [BACKUP](#)
- [BEFORE](#) (R)
- [BEGIN](#)
- [BETWEEN](#) (R)
- [BIGINT](#) (R)
- [BINARY](#) (R)
- [BINLOG](#)
- [BIT](#)
- [BLOB](#) (R)
- [BLOCK](#)
- [BOOL](#)
- [BOOLEAN](#)
- [BOTH](#) (R)
- [BTREE](#)
- [BUCKETS](#); 追加されたバージョン: 8.0.2 (予約なし)
- [BY](#) (R)
- [BYTE](#)

C

- [CACHE](#)
- [CALL](#) (R)
- [CASCADE](#) (R)
- [CASCADED](#)

- CASE (R)
- CATALOG_NAME
- CHAIN
- CHANGE (R)
- CHANGED
- CHANNEL
- CHAR (R)
- CHARACTER (R)
- CHARSET
- CHECK (R)
- CHECKSUM
- CIPHER
- CLASS_ORIGIN
- CLIENT
- CLONE; 追加されたバージョン: 8.0.3 (予約なし)
- CLOSE
- COALESCE
- CODE
- COLLATE (R)
- COLLATION
- COLUMN (R)
- COLUMNS
- COLUMN_FORMAT
- COLUMN_NAME
- COMMENT
- COMMIT
- COMMITTED
- COMPACT
- COMPLETION
- COMPONENT
- COMPRESSED
- COMPRESSION
- CONCURRENT
- CONDITION (R)

- CONNECTION
- CONSISTENT
- CONSTRAINT (R)
- CONSTRAINT_CATALOG
- CONSTRAINT_NAME
- CONSTRAINT_SCHEMA
- CONTAINS
- CONTEXT
- CONTINUE (R)
- CONVERT (R)
- CPU
- CREATE (R)
- CROSS (R)
- CUBE (R); 予約になったバージョン: 8.0.1
- CUME_DIST (R); 追加されたバージョン: 8.0.2 (reserved)
- CURRENT
- CURRENT_DATE (R)
- CURRENT_TIME (R)
- CURRENT_TIMESTAMP (R)
- CURRENT_USER (R)
- CURSOR (R)
- CURSOR_NAME

D

- DATA
- DATABASE (R)
- DATABASES (R)
- DATAFILE
- DATE
- DATETIME
- DAY
- DAY_HOUR (R)
- DAY_MICROSECOND (R)
- DAY_MINUTE (R)
- DAY_SECOND (R)

- DEALLOCATE
 - DEC (R)
 - DECIMAL (R)
 - DECLARE (R)
 - DEFAULT (R)
 - DEFAULT_AUTH
 - DEFINER
 - DEFINITION; 追加されたバージョン: 8.0.4 (予約なし)
 - DELAYED (R)
 - DELAY_KEY_WRITE
 - DELETE (R)
 - DENSE_RANK (R); 追加されたバージョン: 8.0.2 (reserved)
 - DESC (R)
 - DESCRIBE (R)
 - DESCRIPTION; 追加されたバージョン: 8.0.4 (予約なし)
 - DES_KEY_FILE; 削除されたバージョン: 8.0.3
 - DETERMINISTIC (R)
 - DIAGNOSTICS
 - DIRECTORY
 - DISABLE
 - DISCARD
 - DISK
 - DISTINCT (R)
 - DISTINCTROW (R)
 - DIV (R)
 - DO
 - DOUBLE (R)
 - DROP (R)
 - DUAL (R)
 - DUMPFILE
 - DUPLICATE
 - DYNAMIC
- E
- EACH (R)

- [ELSE \(R\)](#)
 - [ELSEIF \(R\)](#)
 - [EMPTY \(R\)](#); 追加されたバージョン: 8.0.4 (reserved)
 - [ENABLE](#)
 - [ENCLOSED \(R\)](#)
 - [ENCRYPTION](#)
 - [END](#)
 - [ENDS](#)
 - [ENFORCED](#); 追加されたバージョン: 8.0.16 (予約なし)
 - [ENGINE](#)
 - [ENGINES](#)
 - [ENGINE_ATTRIBUTE](#); 追加されたバージョン: 8.0.21 (予約なし)
 - [ENUM](#)
 - [ERROR](#)
 - [ERRORS](#)
 - [ESCAPE](#)
 - [ESCAPED \(R\)](#)
 - [EVENT](#)
 - [EVENTS](#)
 - [EVERY](#)
 - [EXCEPT \(R\)](#)
 - [EXCHANGE](#)
 - [EXCLUDE](#); 追加されたバージョン: 8.0.2 (予約なし)
 - [EXECUTE](#)
 - [EXISTS \(R\)](#)
 - [EXIT \(R\)](#)
 - [EXPANSION](#)
 - [EXPIRE](#)
 - [EXPLAIN \(R\)](#)
 - [EXPORT](#)
 - [EXTENDED](#)
 - [EXTENT_SIZE](#)
- F
- [FAILED_LOGIN_ATTEMPTS](#); 追加されたバージョン: 8.0.19 (予約なし)

- FALSE (R)
- FAST
- FAULTS
- FETCH (R)
- FIELDS
- FILE
- FILE_BLOCK_SIZE
- FILTER
- FIRST
- FIRST_VALUE (R); 追加されたバージョン: 8.0.2 (reserved)
- FIXED
- FLOAT (R)
- FLOAT4 (R)
- FLOAT8 (R)
- FLUSH
- FOLLOWING; 追加されたバージョン: 8.0.2 (予約なし)
- FOLLOWS
- FOR (R)
- FORCE (R)
- FOREIGN (R)
- FORMAT
- FOUND
- FROM (R)
- FULL
- FULLTEXT (R)
- FUNCTION (R); 予約になったバージョン: 8.0.1

G

- GENERAL
- GENERATED (R)
- GEOMCOLLECTION; 追加されたバージョン: 8.0.11 (予約なし)
- GEOMETRY
- GEOMETRYCOLLECTION
- GET (R)
- GET_FORMAT

- `GET_MASTER_PUBLIC_KEY`; 追加されたバージョン: 8.0.4 (reserved); 予約されていない 8.0.11
- `GET_SOURCE_PUBLIC_KEY`; 追加されたバージョン: 8.0.23 (予約なし)
- `GLOBAL`
- `GRANT` (R)
- `GRANTS`
- `GROUP` (R)
- `GROUPING` (R); 追加されたバージョン: 8.0.1 (reserved)
- `GROUPS` (R); 追加されたバージョン: 8.0.2 (reserved)
- `GROUP_REPLICATION`

H

- `HANDLER`
- `HASH`
- `HAVING` (R)
- `HELP`
- `HIGH_PRIORITY` (R)
- `HISTOGRAM`; 追加されたバージョン: 8.0.2 (予約なし)
- `HISTORY`; 追加されたバージョン: 8.0.3 (予約なし)
- `HOST`
- `HOSTS`
- `HOURL`
- `HOURL_MICROSECOND` (R)
- `HOURL_MINUTE` (R)
- `HOURL_SECOND` (R)

I

- `IDENTIFIED`
- `IF` (R)
- `IGNORE` (R)
- `IGNORE_SERVER_IDS`
- `IMPORT`
- `IN` (R)
- `INACTIVE`; 追加されたバージョン: 8.0.14 (予約なし)
- `INDEX` (R)
- `INDEXES`
- `INFILE` (R)

- INITIAL_SIZE
- INNER (R)
- INOUT (R)
- INSENSITIVE (R)
- INSERT (R)
- INSERT_METHOD
- INSTALL
- INSTANCE
- INT (R)
- INT1 (R)
- INT2 (R)
- INT3 (R)
- INT4 (R)
- INT8 (R)
- INTEGER (R)
- INTERVAL (R)
- INTO (R)
- INVISIBLE
- INVOKER
- IO
- IO_AFTER_GTIDS (R)
- IO_BEFORE_GTIDS (R)
- IO_THREAD
- IPC
- IS (R)
- ISOLATION
- ISSUER
- ITERATE (R)

J

- JOIN (R)
- JSON
- JSON_TABLE (R); 追加されたバージョン: 8.0.4 (reserved)
- JSON_VALUE; 追加されたバージョン: 8.0.21 (予約なし)

K

- KEY (R)
 - KEYRING; 追加されたバージョン: 8.0.24 (予約なし)
 - KEYS (R)
 - KEY_BLOCK_SIZE
 - KILL (R)
- L
- LAG (R); 追加されたバージョン: 8.0.2 (reserved)
 - LANGUAGE
 - LAST
 - LAST_VALUE (R); 追加されたバージョン: 8.0.2 (reserved)
 - LATERAL (R); 追加されたバージョン: 8.0.14 (reserved)
 - LEAD (R); 追加されたバージョン: 8.0.2 (reserved)
 - LEADING (R)
 - LEAVE (R)
 - LEAVES
 - LEFT (R)
 - LESS
 - LEVEL
 - LIKE (R)
 - LIMIT (R)
 - LINEAR (R)
 - LINES (R)
 - LINestring
 - LIST
 - LOAD (R)
 - LOCAL
 - LOCALTIME (R)
 - LOCALTIMESTAMP (R)
 - LOCK (R)
 - LOCKED; 追加されたバージョン: 8.0.1 (予約なし)
 - LOCKS
 - LOGFILE
 - LOGS
 - LONG (R)

- `LOBLOB (R)`
- `LONGTEXT (R)`
- `LOOP (R)`
- `LOW_PRIORITY (R)`

M

- `MASTER`
- `MASTER_AUTO_POSITION`
- `MASTER_BIND (R)`
- `MASTER_COMPRESSION_ALGORITHMS`; 追加されたバージョン: 8.0.18 (予約なし)
- `MASTER_CONNECT_RETRY`
- `MASTER_DELAY`
- `MASTER_HEARTBEAT_PERIOD`
- `MASTER_HOST`
- `MASTER_LOG_FILE`
- `MASTER_LOG_POS`
- `MASTER_PASSWORD`
- `MASTER_PORT`
- `MASTER_PUBLIC_KEY_PATH`; 追加されたバージョン: 8.0.4 (予約なし)
- `MASTER_RETRY_COUNT`
- `MASTER_SERVER_ID`; 削除されたバージョン: 8.0.23
- `MASTER_SSL`
- `MASTER_SSL_CA`
- `MASTER_SSL_CAPATH`
- `MASTER_SSL_CERT`
- `MASTER_SSL_CIPHER`
- `MASTER_SSL_CRL`
- `MASTER_SSL_CRLPATH`
- `MASTER_SSL_KEY`
- `MASTER_SSL_VERIFY_SERVER_CERT (R)`
- `MASTER_TLS_CIPHERSUITES`; 追加されたバージョン: 8.0.19 (予約なし)
- `MASTER_TLS_VERSION`
- `MASTER_USER`
- `MASTER_ZSTD_COMPRESSION_LEVEL`; 追加されたバージョン: 8.0.18 (予約なし)
- `MATCH (R)`

- MAXVALUE (R)
- MAX_CONNECTIONS_PER_HOUR
- MAX_QUERIES_PER_HOUR
- MAX_ROWS
- MAX_SIZE
- MAX_UPDATES_PER_HOUR
- MAX_USER_CONNECTIONS
- MEDIUM
- MEDIUMBLOB (R)
- MEDIUMINT (R)
- MEDIUMTEXT (R)
- MEMBER; 追加されたバージョン: 8.0.17 (reserved); 予約されていない 8.0.19
- MEMORY
- MERGE
- MESSAGE_TEXT
- MICROSECOND
- MIDDLEINT (R)
- MIGRATE
- MINUTE
- MINUTE_MICROSECOND (R)
- MINUTE_SECOND (R)
- MIN_ROWS
- MOD (R)
- MODE
- MODIFIES (R)
- MODIFY
- MONTH
- MULTILINESTRING
- MULTIPOINT
- MULTIPOLYGON
- MUTEX
- MYSQL_ERRNO

N

- NAME

- NAMES
 - NATIONAL
 - NATURAL (R)
 - NCHAR
 - NDB
 - NDBCLUSTER
 - NESTED; 追加されたバージョン: 8.0.4 (予約なし)
 - NETWORK_NAMESPACE; 追加されたバージョン: 8.0.16 (予約なし)
 - NEVER
 - NEW
 - NEXT
 - NO
 - NODEGROUP
 - NONE
 - NOT (R)
 - NOWAIT; 追加されたバージョン: 8.0.1 (予約なし)
 - NO_WAIT
 - NO_WRITE_TO_BINLOG (R)
 - NTH_VALUE (R); 追加されたバージョン: 8.0.2 (reserved)
 - NTILE (R); 追加されたバージョン: 8.0.2 (reserved)
 - NULL (R)
 - NULLS; 追加されたバージョン: 8.0.2 (予約なし)
 - NUMBER
 - NUMERIC (R)
 - NVARCHAR
- O
- OF (R); 追加されたバージョン: 8.0.1 (reserved)
 - OFF; 追加されたバージョン: 8.0.20 (予約なし)
 - OFFSET
 - OJ; 追加されたバージョン: 8.0.16 (予約なし)
 - OLD; 追加されたバージョン: 8.0.14 (予約なし)
 - ON (R)
 - ONE
 - ONLY

- OPEN
 - OPTIMIZE (R)
 - OPTIMIZER_COSTS (R)
 - OPTION (R)
 - OPTIONAL; 追加されたバージョン: 8.0.13 (予約なし)
 - OPTIONALLY (R)
 - OPTIONS
 - OR (R)
 - ORDER (R)
 - ORDINALITY; 追加されたバージョン: 8.0.4 (予約なし)
 - ORGANIZATION; 追加されたバージョン: 8.0.4 (予約なし)
 - OTHERS; 追加されたバージョン: 8.0.2 (予約なし)
 - OUT (R)
 - OUTER (R)
 - OUTFILE (R)
 - OVER (R); 追加されたバージョン: 8.0.2 (reserved)
 - OWNER
- P
- PACK_KEYS
 - PAGE
 - PARSER
 - PARTIAL
 - PARTITION (R)
 - PARTITIONING
 - PARTITIONS
 - PASSWORD
 - PASSWORD_LOCK_TIME; 追加されたバージョン: 8.0.19 (予約なし)
 - PATH; 追加されたバージョン: 8.0.4 (予約なし)
 - PERCENT_RANK (R); 追加されたバージョン: 8.0.2 (reserved)
 - PERSIST; 予約されていない 8.0.16
 - PERSIST_ONLY; 追加されたバージョン: 8.0.2 (reserved); 予約されていない 8.0.16
 - PHASE
 - PLUGIN
 - PLUGINS

- PLUGIN_DIR
 - POINT
 - POLYGON
 - PORT
 - PRECEDES
 - PRECEDING; 追加されたバージョン: 8.0.2 (予約なし)
 - PRECISION (R)
 - PREPARE
 - PRESERVE
 - PREV
 - PRIMARY (R)
 - PRIVILEGES
 - PRIVILEGE_CHECKS_USER; 追加されたバージョン: 8.0.18 (予約なし)
 - PROCEDURE (R)
 - PROCESS; 追加されたバージョン: 8.0.11 (予約なし)
 - PROCESSLIST
 - PROFILE
 - PROFILES
 - PROXY
 - PURGE (R)
- Q
- QUARTER
 - QUERY
 - QUICK
- R
- RANDOM; 追加されたバージョン: 8.0.18 (予約なし)
 - RANGE (R)
 - RANK (R); 追加されたバージョン: 8.0.2 (reserved)
 - READ (R)
 - READS (R)
 - READ_ONLY
 - READ_WRITE (R)
 - REAL (R)
 - REBUILD

- RECOVER
- RECURSIVE (R); 追加されたバージョン: 8.0.1 (reserved)
- REDOFILE; 削除されたバージョン: 8.0.3
- REDO_BUFFER_SIZE
- REDUNDANT
- REFERENCE; 追加されたバージョン: 8.0.4 (予約なし)
- REFERENCES (R)
- REGEXP (R)
- RELAY
- RELAYLOG
- RELAY_LOG_FILE
- RELAY_LOG_POS
- RELAY_THREAD
- RELEASE (R)
- RELOAD
- REMOTE; 追加されたバージョン: 8.0.3 (予約なし); 削除されたバージョン: 8.0.14
- REMOVE
- RENAME (R)
- REORGANIZE
- REPAIR
- REPEAT (R)
- REPEATABLE
- REPLACE (R)
- REPLICA; 追加されたバージョン: 8.0.22 (予約なし)
- REPLICAS; 追加されたバージョン: 8.0.22 (予約なし)
- REPLICATE_DO_DB
- REPLICATE_DO_TABLE
- REPLICATE_IGNORE_DB
- REPLICATE_IGNORE_TABLE
- REPLICATE_REWRITE_DB
- REPLICATE_WILD_DO_TABLE
- REPLICATE_WILD_IGNORE_TABLE
- REPLICATION
- REQUIRE (R)

- `REQUIRE_ROW_FORMAT`; 追加されたバージョン: 8.0.19 (予約なし)
 - `RESET`
 - `RESIGNAL` (R)
 - `RESOURCE`; 追加されたバージョン: 8.0.3 (予約なし)
 - `RESPECT`; 追加されたバージョン: 8.0.2 (予約なし)
 - `RESTART`; 追加されたバージョン: 8.0.4 (予約なし)
 - `RESTORE`
 - `RESTRICT` (R)
 - `RESUME`
 - `RETAIN`; 追加されたバージョン: 8.0.14 (予約なし)
 - `RETURN` (R)
 - `RETURNED_SQLSTATE`
 - `RETURNING`; 追加されたバージョン: 8.0.21 (予約なし)
 - `RETURNS`
 - `REUSE`; 追加されたバージョン: 8.0.3 (予約なし)
 - `REVERSE`
 - `REVOKE` (R)
 - `RIGHT` (R)
 - `RLIKE` (R)
 - `ROLE`; 予約されていない 8.0.1
 - `ROLLBACK`
 - `ROLLUP`
 - `ROTATE`
 - `ROUTINE`
 - `ROW` (R); 予約になったバージョン: 8.0.2
 - `ROWS` (R); 予約になったバージョン: 8.0.2
 - `ROW_COUNT`
 - `ROW_FORMAT`
 - `ROW_NUMBER` (R); 追加されたバージョン: 8.0.2 (reserved)
 - `RTREE`
- S
- `SAVEPOINT`
 - `SCHEDULE`
 - `SCHEMA` (R)

- SCHEMAS (R)
- SCHEMA_NAME
- SECOND
- SECONDARY; 追加されたバージョン: 8.0.16 (予約なし)
- SECONDARY_ENGINE; 追加されたバージョン: 8.0.13 (予約なし)
- SECONDARY_ENGINE_ATTRIBUTE; 追加されたバージョン: 8.0.21 (予約なし)
- SECONDARY_LOAD; 追加されたバージョン: 8.0.13 (予約なし)
- SECONDARY_UNLOAD; 追加されたバージョン: 8.0.13 (予約なし)
- SECOND_MICROSECOND (R)
- SECURITY
- SELECT (R)
- SENSITIVE (R)
- SEPARATOR (R)
- SERIAL
- SERIALIZABLE
- SERVER
- SESSION
- SET (R)
- SHARE
- SHOW (R)
- SHUTDOWN
- SIGNAL (R)
- SIGNED
- SIMPLE
- SKIP; 追加されたバージョン: 8.0.1 (予約なし)
- SLAVE
- SLOW
- SMALLINT (R)
- SNAPSHOT
- SOCKET
- SOME
- SONAME
- SOUNDS
- SOURCE

- [SOURCE_AUTO_POSITION](#); 追加されたバージョン: 8.0.23 (予約なし)
- [SOURCE_BIND](#); 追加されたバージョン: 8.0.23 (予約なし)
- [SOURCE_COMPRESSION_ALGORITHMS](#); 追加されたバージョン: 8.0.23 (予約なし)
- [SOURCE_CONNECT_RETRY](#); 追加されたバージョン: 8.0.23 (予約なし)
- [SOURCE_DELAY](#); 追加されたバージョン: 8.0.23 (予約なし)
- [SOURCE_HEARTBEAT_PERIOD](#); 追加されたバージョン: 8.0.23 (予約なし)
- [SOURCE_HOST](#); 追加されたバージョン: 8.0.23 (予約なし)
- [SOURCE_LOG_FILE](#); 追加されたバージョン: 8.0.23 (予約なし)
- [SOURCE_LOG_POS](#); 追加されたバージョン: 8.0.23 (予約なし)
- [SOURCE_PASSWORD](#); 追加されたバージョン: 8.0.23 (予約なし)
- [SOURCE_PORT](#); 追加されたバージョン: 8.0.23 (予約なし)
- [SOURCE_PUBLIC_KEY_PATH](#); 追加されたバージョン: 8.0.23 (予約なし)
- [SOURCE_RETRY_COUNT](#); 追加されたバージョン: 8.0.23 (予約なし)
- [SOURCE_SSL](#); 追加されたバージョン: 8.0.23 (予約なし)
- [SOURCE_SSL_CA](#); 追加されたバージョン: 8.0.23 (予約なし)
- [SOURCE_SSL_CAPATH](#); 追加されたバージョン: 8.0.23 (予約なし)
- [SOURCE_SSL_CERT](#); 追加されたバージョン: 8.0.23 (予約なし)
- [SOURCE_SSL_CIPHER](#); 追加されたバージョン: 8.0.23 (予約なし)
- [SOURCE_SSL_CRL](#); 追加されたバージョン: 8.0.23 (予約なし)
- [SOURCE_SSL_CRLPATH](#); 追加されたバージョン: 8.0.23 (予約なし)
- [SOURCE_SSL_KEY](#); 追加されたバージョン: 8.0.23 (予約なし)
- [SOURCE_SSL_VERIFY_SERVER_CERT](#); 追加されたバージョン: 8.0.23 (予約なし)
- [SOURCE_TLS_CIPHERSUITES](#); 追加されたバージョン: 8.0.23 (予約なし)
- [SOURCE_TLS_VERSION](#); 追加されたバージョン: 8.0.23 (予約なし)
- [SOURCE_USER](#); 追加されたバージョン: 8.0.23 (予約なし)
- [SOURCE_ZSTD_COMPRESSION_LEVEL](#); 追加されたバージョン: 8.0.23 (予約なし)
- [SPATIAL](#) (R)
- [SPECIFIC](#) (R)
- [SQL](#) (R)
- [SQLEXCEPTION](#) (R)
- [SQLSTATE](#) (R)
- [SQLWARNING](#) (R)
- [SQL_AFTER_GTIDS](#)
- [SQL_AFTER_MTS_GAPS](#)

- SQL_BEFORE_GTIDS
- SQL_BIG_RESULT (R)
- SQL_BUFFER_RESULT
- SQL_CACHE; 削除されたバージョン: 8.0.3
- SQL_CALC_FOUND_ROWS (R)
- SQL_NO_CACHE
- SQL_SMALL_RESULT (R)
- SQL_THREAD
- SQL_TSI_DAY
- SQL_TSI_HOUR
- SQL_TSI_MINUTE
- SQL_TSI_MONTH
- SQL_TSI_QUARTER
- SQL_TSI_SECOND
- SQL_TSI_WEEK
- SQL_TSI_YEAR
- SRID; 追加されたバージョン: 8.0.3 (予約なし)
- SSL (R)
- STACKED
- START
- STARTING (R)
- STARTS
- STATS_AUTO_RECALC
- STATS_PERSISTENT
- STATS_SAMPLE_PAGES
- STATUS
- STOP
- STORAGE
- STORED (R)
- STRAIGHT_JOIN (R)
- STREAM; 追加されたバージョン: 8.0.20 (予約なし)
- STRING
- SUBCLASS_ORIGIN
- SUBJECT

- SUBPARTITION
- SUBPARTITIONS
- SUPER
- SUSPEND
- SWAPS
- SWITCHES
- SYSTEM (R); 追加されたバージョン: 8.0.3 (reserved)

T

- TABLE (R)
- TABLES
- TABLESPACE
- TABLE_CHECKSUM
- TABLE_NAME
- TEMPORARY
- TEMPTABLE
- TERMINATED (R)
- TEXT
- THAN
- THEN (R)
- THREAD_PRIORITY; 追加されたバージョン: 8.0.3 (予約なし)
- TIES; 追加されたバージョン: 8.0.2 (予約なし)
- TIME
- TIMESTAMP
- TIMESTAMPADD
- TIMESTAMPDIF
- TINYBLOB (R)
- TINYINT (R)
- TINYTEXT (R)
- TLS; 追加されたバージョン: 8.0.21 (予約なし)
- TO (R)
- TRAILING (R)
- TRANSACTION
- TRIGGER (R)
- TRIGGERS

- TRUE (R)
- TRUNCATE
- TYPE
- TYPES

U

- UNBOUNDED; 追加されたバージョン: 8.0.2 (予約なし)
- UNCOMMITTED
- UNDEFINED
- UNDO (R)
- UNDOFILE
- UNDO_BUFFER_SIZE
- UNICODE
- UNINSTALL
- UNION (R)
- UNIQUE (R)
- UNKNOWN
- UNLOCK (R)
- UNSIGNED (R)
- UNTIL
- UPDATE (R)
- UPGRADE
- USAGE (R)
- USE (R)
- USER
- USER_RESOURCES
- USE_FRM
- USING (R)
- UTC_DATE (R)
- UTC_TIME (R)
- UTC_TIMESTAMP (R)

V

- VALIDATION
- VALUE
- VALUES (R)

- [VARBINARY](#) (R)
- [VARCHAR](#) (R)
- [VARCHARACTER](#) (R)
- [VARIABLES](#)
- [VARYING](#) (R)
- [VCPU](#); 追加されたバージョン: 8.0.3 (予約なし)
- [VIEW](#)
- [VIRTUAL](#) (R)
- [VISIBLE](#)

W

- [WAIT](#)
- [WARNINGS](#)
- [WEEK](#)
- [WEIGHT_STRING](#)
- [WHEN](#) (R)
- [WHERE](#) (R)
- [WHILE](#) (R)
- [WINDOW](#) (R); 追加されたバージョン: 8.0.2 (reserved)
- [WITH](#) (R)
- [WITHOUT](#)
- [WORK](#)
- [WRAPPER](#)
- [WRITE](#) (R)

X

- [X509](#)
- [XA](#)
- [XID](#)
- [XML](#)
- [XOR](#) (R)

Y

- [YEAR](#)
- [YEAR_MONTH](#) (R)

Z

- [ZEROFILL](#) (R)

- [ZONE](#); 追加されたバージョン: 8.0.22 (予約なし)

MySQL 8.0 の新しいキーワードおよび予約語

次のリストに、MySQL 5.7 と比較して MySQL 8.0 に追加されるキーワードおよび予約語を示します。予約済キーワードは (R) でマークされます。

[A](#) | [B](#) | [C](#) | [D](#) | [E](#) | [F](#) | [G](#) | [H](#) | [I](#) | [J](#) | [K](#) | [L](#) | [M](#) | [N](#) | [O](#) | [P](#) | [R](#) | [S](#) | [T](#) | [U](#) | [V](#) | [W](#) | [Z](#)

A

- [ACTIVE](#)
- [ADMIN](#)
- [ARRAY](#)
- [ATTRIBUTE](#)

B

- [BUCKETS](#)

C

- [CLONE](#)
- [COMPONENT](#)
- [CUME_DIST](#) (R)

D

- [DEFINITION](#)
- [DENSE_RANK](#) (R)
- [DESCRIPTION](#)

E

- [EMPTY](#) (R)
- [ENFORCED](#)
- [ENGINE_ATTRIBUTE](#)
- [EXCEPT](#) (R)
- [EXCLUDE](#)

F

- [FAILED_LOGIN_ATTEMPTS](#)
- [FIRST_VALUE](#) (R)
- [FOLLOWING](#)

G

- [GEOMCOLLECTION](#)
- [GET_MASTER_PUBLIC_KEY](#)
- [GET_SOURCE_PUBLIC_KEY](#)

- GROUPING (R)

- GROUPS (R)

H

- HISTOGRAM

- HISTORY

I

- INACTIVE

- INVISIBLE

J

- JSON_TABLE (R)

- JSON_VALUE

K

- KEYRING

L

- LAG (R)

- LAST_VALUE (R)

- LATERAL (R)

- LEAD (R)

- LOCKED

M

- MASTER_COMPRESSION_ALGORITHMS

- MASTER_PUBLIC_KEY_PATH

- MASTER_TLS_CIPHERSUITES

- MASTER_ZSTD_COMPRESSION_LEVEL

- MEMBER

N

- NESTED

- NETWORK_NAMESPACE

- NOWAIT

- NTH_VALUE (R)

- NTILE (R)

- NULLS

O

- OF (R)

- OFF
- OJ
- OLD
- OPTIONAL
- ORDINALITY
- ORGANIZATION
- OTHERS
- OVER (R)

P

- PASSWORD_LOCK_TIME
- PATH
- PERCENT_RANK (R)
- PERSIST
- PERSIST_ONLY
- PRECEDING
- PRIVILEGE_CHECKS_USER
- PROCESS

R

- RANDOM
- RANK (R)
- RECURSIVE (R)
- REFERENCE
- REPLICA
- REPLICAS
- REQUIRE_ROW_FORMAT
- RESOURCE
- RESPECT
- RESTART
- RETAIN
- RETURNING
- REUSE
- ROLE
- ROW_NUMBER (R)

S

- SECONDARY
- SECONDARY_ENGINE
- SECONDARY_ENGINE_ATTRIBUTE
- SECONDARY_LOAD
- SECONDARY_UNLOAD
- SKIP
- SOURCE_AUTO_POSITION
- SOURCE_BIND
- SOURCE_COMPRESSION_ALGORITHMS
- SOURCE_CONNECT_RETRY
- SOURCE_DELAY
- SOURCE_HEARTBEAT_PERIOD
- SOURCE_HOST
- SOURCE_LOG_FILE
- SOURCE_LOG_POS
- SOURCE_PASSWORD
- SOURCE_PORT
- SOURCE_PUBLIC_KEY_PATH
- SOURCE_RETRY_COUNT
- SOURCE_SSL
- SOURCE_SSL_CA
- SOURCE_SSL_CAPATH
- SOURCE_SSL_CERT
- SOURCE_SSL_CIPHER
- SOURCE_SSL_CRL
- SOURCE_SSL_CRLPATH
- SOURCE_SSL_KEY
- SOURCE_SSL_VERIFY_SERVER_CERT
- SOURCE_TLS_CIPHERSUITES
- SOURCE_TLS_VERSION
- SOURCE_USER
- SOURCE_ZSTD_COMPRESSION_LEVEL
- SRID
- STREAM

- [SYSTEM \(R\)](#)

T

- [THREAD_PRIORITY](#)

- [TIES](#)

- [TLS](#)

U

- [UNBOUNDED](#)

V

- [VCPU](#)

- [VISIBLE](#)

W

- [WINDOW \(R\)](#)

Z

- [ZONE](#)

MySQL 8.0 で削除されたキーワードおよび予約語

次のリストに、MySQL 5.7 と比較して MySQL 8.0 で削除されるキーワードおよび予約語を示します。予約済キーワードは (R) でマークされます。

- [ANALYSE](#)

- [DES_KEY_FILE](#)

- [MASTER_SERVER_ID](#)

- [PARSE_GCOL_EXPR](#)

- [REDOFILE](#)

- [SQL_CACHE](#)

9.4 ユーザー定義変数

あるステートメントのユーザー定義変数に値を格納し、後で別のステートメントで参照できます。これにより、あるステートメントから別のステートメントに値を渡すことができます。

ユーザー変数は `@var_name` として記述され、変数名 `var_name` は英数字、`.`、`_` および `$` で構成されます。ユーザー変数名を文字列や識別子として引用符で囲めば、ほかの文字も含めることができます (`@'my-var'`、`@"my-var"`、`@`my-var`` など)。

ユーザー定義変数はセッション固有です。あるクライアントで定義されたユーザー変数は、他のクライアントでは表示または使用できません。(例外: パフォーマンススキーマ `user_variables_by_thread` テーブルへのアクセス権を持つユーザーは、すべてのセッションのすべてのユーザー変数を表示できます。) 所定のクライアントセッションのすべての変数は、クライアントが終了すると自動的に解放されます。

ユーザー変数名では大/小文字は区別されません。名前の最大長は 64 文字です。

ユーザー定義変数を設定する方法の 1 つに、`SET` ステートメントを発行する方法が挙げられます。

```
SET @var_name = expr [, @var_name = expr] ...
```

SET では、**=** または **:=** のどちらかを割り当て演算子として使用できます。

ユーザー変数には、限定された一連のデータ型の値 (整数、小数、浮動小数点、バイナリ文字列、非バイナリ文字列、または **NULL** 値) を割り当てることができます。10 進値と実数値の割り当てでは、値の精度やスケールは維持されません。許可されている型以外の型の値は、許可されている型に変換されます。たとえば、時間を表すデータ型や空間データ型の値は、バイナリ文字列に変換されます。JSON データ型の値は、**utf8mb4** の文字セットと **utf8mb4_bin** の照合順序で文字列に変換されます。

ユーザー変数に非バイナリ (文字) 文字列値を割り当てた場合、その変数には文字列と同じ文字セットと照合順序が含まれます。ユーザー変数の強制性は暗黙的です。(これはテーブルカラム値と同等の強制性です。)

ユーザー変数に割り当てられた 16 進数またはビット値は、バイナリ文字列として扱われます。16 進数またはビット値を数値としてユーザー変数に割り当てするには、数値コンテキストで使用します。たとえば、0 を追加するか、**CAST(... AS UNSIGNED)** を使用します:

```
mysql> SET @v1 = X'41';
mysql> SET @v2 = X'41'+0;
mysql> SET @v3 = CAST(X'41' AS UNSIGNED);
mysql> SELECT @v1, @v2, @v3;
+-----+-----+-----+
| @v1 | @v2 | @v3 |
+-----+-----+-----+
| A | 65 | 65 |
+-----+-----+-----+
mysql> SET @v1 = b'1000001';
mysql> SET @v2 = b'1000001'+0;
mysql> SET @v3 = CAST(b'1000001' AS UNSIGNED);
mysql> SELECT @v1, @v2, @v3;
+-----+-----+-----+
| @v1 | @v2 | @v3 |
+-----+-----+-----+
| A | 65 | 65 |
+-----+-----+-----+
```

結果セットでユーザー変数の値が選択された場合、それは文字列としてクライアントに返されます。

初期化されていない変数を参照する場合、その値は **NULL** で、型は文字列です。

MySQL 8.0.22 以降、プリペアドステートメント内のユーザー変数への参照のタイプは、ステートメントが最初に準備されたときに決定され、それ以降にステートメントが実行されるたびにこのタイプが保持されます。同様に、ストアードプロシージャ内のステートメントで使用されるユーザー変数の型は、ストアードプロシージャが最初に起動されたときに決定され、後続の起動のたびにこの型が保持されます。

ユーザー変数は、式が許可されているほとんどのコンテキストで使用できます。これには現在、**SELECT** ステートメントの **LIMIT** 句の中や、**LOAD DATA** ステートメントの **IGNORE N LINES** 句の中など、リテラル値を明示的に要求するコンテキストは含まれません。

以前のリリースの MySQL では、**SET** 以外のステートメントでユーザー変数に値を割り当てることができました。この機能は、下位互換性のために MySQL 8.0 でサポートされていますが、MySQL の将来のリリースで削除される予定です。

この方法で割当てを行う場合は、**:=** を割当て演算子として使用する必要があります。**=** は、**SET** 以外のステートメントで比較演算子として扱われます。

ユーザー変数を含む式の評価順序が定義されていません。たとえば、**SELECT @a, @a:=@a+1** が最初に **@a** を評価してから割当てを実行する保証はありません。

また、変数のデフォルトの結果タイプは、ステートメントの先頭のタイプに基づきます。変数がステートメントの先頭にあるあるあるある型の値を保持していて、別の型の新しい値も割り当てられている場合、これは意図しない影響を与える可能性があります。

この動作による問題を回避するには、単一のステートメント内で同じ変数に値を割り当ててその値を読み取ることを行わないか、使用する前に変数を **0**、**0.0**、または **"** に設定して、その型を定義してください。

HAVING、**GROUP BY** および **ORDER BY** では、選択式リストの値が割り当てられている変数を参照する場合、式はクライアントで評価されるため、予期したとおりに機能しないため、前の行の失効したカラム値を使用できます。

ユーザー変数は、データ値を提供するためのものです。これらは、テーブル名やデータベース名が想定されるコンテキストなどでの識別子または識別子の一部として、または `SELECT` などの予約語として、SQL ステートメントの中で直接使用することはできません。これは、次の例に示すように、変数が引用符で囲まれている場合でも同じです。

```
mysql> SELECT c1 FROM t;
+----+
| c1 |
+----+
| 0 |
+----+
| 1 |
+----+
2 rows in set (0.00 sec)

mysql> SET @col = "c1";
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @col FROM t;
+-----+
| @col |
+-----+
| c1 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT `@col` FROM t;
ERROR 1054 (42S22): Unknown column '@col' in 'field list'

mysql> SET @col = "`c1`";
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @col FROM t;
+-----+
| @col |
+-----+
| `c1` |
+-----+
1 row in set (0.00 sec)
```

識別子を提供するためにユーザー変数を使用できないというこの原則の例外が、あとから実行する準備されたステートメントとして使用するために文字列を構築している場合です。この場合、ユーザー変数はステートメントの一部を提供するために使用できます。次の例は、その方法を示しています。

```
mysql> SET @c = "c1";
Query OK, 0 rows affected (0.00 sec)

mysql> SET @s = CONCAT("SELECT ", @c, " FROM t");
Query OK, 0 rows affected (0.00 sec)

mysql> PREPARE stmt FROM @s;
Query OK, 0 rows affected (0.04 sec)
Statement prepared

mysql> EXECUTE stmt;
+----+
| c1 |
+----+
| 0 |
+----+
| 1 |
+----+
2 rows in set (0.00 sec)

mysql> DEALLOCATE PREPARE stmt;
Query OK, 0 rows affected (0.00 sec)
```

詳しくは[セクション13.5「プリパードステートメント」](#)をご覧ください。

PHP 5 を使用して次に示すように、同じような手法をアプリケーションプログラムの中で使用することによって、プログラム変数を使用する SQL ステートメントを作成できます。

```
<?php
$mysqli = new mysqli("localhost", "user", "pass", "test");
```

```
if( mysqli_connect_errno() )
    die("Connection failed: %s\n", mysqli_connect_error());

$col = "c1";

$query = "SELECT $col FROM t";

$result = $mysqli->query($query);

while($row = $result->fetch_assoc())
{
    echo "<p>" . $row["$col"] . "</p>\n";
}

$result->close();

$mysqli->close();
?>
```

この方法で SQL ステートメントを作成することを「動的 SQL」と呼ぶことがあります。

9.5 式

このセクションでは、式が MySQL で従う必要がある文法ルールをリストし、式に使用できる用語のタイプに関する追加情報を提供します。

- [式の構文](#)
- [式の用語のノート](#)
- [時間間隔](#)

式の構文

次の文法ルールは、MySQL で式の構文を定義します。ここで示す文法は、MySQL ソース配布の `sql/sql_yacc.yy` ファイルで与えられた文法に基づいています。一部の式の用語の詳細は、[式の用語のノート](#) を参照してください。

```
expr:
  expr OR expr
| expr || expr
| expr XOR expr
| expr AND expr
| expr && expr
| NOT expr
| ! expr
| boolean_primary IS [NOT] {TRUE | FALSE | UNKNOWN}
| boolean_primary

boolean_primary:
  boolean_primary IS [NOT] NULL
| boolean_primary <=> predicate
| boolean_primary comparison_operator predicate
| boolean_primary comparison_operator {ALL | ANY} (subquery)
| predicate

comparison_operator: = | >= | > | <= | < | <> | !=

predicate:
  bit_expr [NOT] IN (subquery)
| bit_expr [NOT] IN (expr [, expr] ...)
| bit_expr [NOT] BETWEEN bit_expr AND predicate
| bit_expr SOUNDS LIKE bit_expr
| bit_expr [NOT] LIKE simple_expr [ESCAPE simple_expr]
| bit_expr [NOT] REGEXP bit_expr
| bit_expr

bit_expr:
  bit_expr | bit_expr
| bit_expr & bit_expr
```

```

| bit_expr << bit_expr
| bit_expr >> bit_expr
| bit_expr + bit_expr
| bit_expr - bit_expr
| bit_expr * bit_expr
| bit_expr / bit_expr
| bit_expr DIV bit_expr
| bit_expr MOD bit_expr
| bit_expr % bit_expr
| bit_expr ^ bit_expr
| bit_expr + interval_expr
| bit_expr - interval_expr
| simple_expr

simple_expr:
  literal
| identifier
| function_call
| simple_expr COLLATE collation_name
| param_marker
| variable
| simple_expr || simple_expr
| + simple_expr
| - simple_expr
| ~ simple_expr
| ! simple_expr
| BINARY simple_expr
| (expr [, expr] ...)
| ROW (expr, expr [, expr] ...)
| (subquery)
| EXISTS (subquery)
| {identifier expr}
| match_expr
| case_expr
| interval_expr

```

演算子の優先順位については、[セクション12.4.1「演算子の優先順位」](#)を参照してください。一部の演算子の優先順位と意味は、SQL モードによって異なります:

- デフォルトでは、`||` は論理 `OR` 演算子です。 `PIPES_AS_CONCAT` が有効になっている場合は、`||` は `^` と単項演算子間の優先順位を持つ文字列連結です。
- デフォルトでは、`!` は `NOT` よりも高い優先順位です。 `HIGH_NOT_PRECEDENCE` が有効になっている場合は、`!` と `NOT` の優先順位は同じです。

[セクション5.1.11「サーバー SQL モード」](#)を参照してください。

式の用語のノート

リテラル値の構文については、[セクション9.1「リテラル値」](#)を参照してください。

識別子の構文については、[セクション9.2「スキーマオブジェクト名」](#)を参照してください。

変数には、ユーザー変数、システム変数、ストアドプログラムのローカル変数またはパラメータがあります。

- ユーザー変数: [セクション9.4「ユーザー定義変数」](#)
- システム変数: [セクション5.1.9「システム変数の使用」](#)
- ストアドプログラムローカル変数: [セクション13.6.4.1「ローカル変数 DECLARE ステートメント」](#)
- ストアドプログラムパラメータ: [セクション13.1.17「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」](#)

`param_marker` は、準備されたステートメントでプレースホルダーに使用されているように `?` です。 [セクション13.5.1「PREPARE ステートメント」](#)を参照してください。

`(subquery)` は、単一の値を返すサブクエリー、つまりスカラーサブクエリーを示します。 [セクション13.2.11.1「スカラーオペランドとしてのサブクエリー」](#)を参照してください。

{*identifier expr*} は、ODBC エスケープ構文であり、ODBC との互換性のために認められています。値は *expr* です。構文内の {および} 中カッコは文字どおりに記述する必要があります。構文の説明の他の場所で使用されているメタ構文ではありません。

match_expr は MATCH 式を示します。 [セクション12.10「全文検索関数」](#) を参照してください。

case_expr は CASE 式を示します。 [セクション12.5「フロー制御関数」](#) を参照してください。

interval_expr は時間間隔を表します。 [時間間隔](#) を参照してください。

時間間隔

式の *interval_expr* は時間間隔を表します。間隔の構文は次のとおりです:

```
INTERVAL expr unit
```

expr は数量を表します。*unit* は、数量を解釈するための単位を表します。HOUR、DAY、WEEK などの指定子です。INTERVAL キーワードおよび *unit* 指定子では、大文字と小文字は区別されません。

次の表には、*unit* 値ごとに要求される形式の *expr* 引数を表示します。

表 9.2 時間隔式および単位引数

unit 値	要求される <i>expr</i> 書式
MICROSECOND	MICROSECONDS
SECOND	SECONDS
MINUTE	MINUTES
HOUR	HOURS
DAY	DAYS
WEEK	WEEKS
MONTH	MONTHS
QUARTER	QUARTERS
YEAR	YEARS
SECOND_MICROSECOND	'SECONDS.MICROSECONDS'
MINUTE_MICROSECOND	'MINUTES:SECONDS.MICROSECONDS'
MINUTE_SECOND	'MINUTES:SECONDS'
HOUR_MICROSECOND	'HOURS:MINUTES:SECONDS.MICROSECONDS'
HOUR_SECOND	'HOURS:MINUTES:SECONDS'
HOUR_MINUTE	'HOURS:MINUTES'
DAY_MICROSECOND	'DAYS HOURS:MINUTES:SECONDS.MICROSECONDS'
DAY_SECOND	'DAYS HOURS:MINUTES:SECONDS'
DAY_MINUTE	'DAYS HOURS:MINUTES'
DAY_HOUR	'DAYS HOURS'
YEAR_MONTH	'YEARS-MONTHS'

MySQL では、*expr* 書式の句読点区切り文字が許可されます。表には、提案される区切り文字を表示します。

一時間隔は、DATE_ADD() や DATE_SUB() などの特定の関数に使用されます:

```
mysql> SELECT DATE_ADD('2018-05-01',INTERVAL 1 DAY);
-> '2018-05-02'
mysql> SELECT DATE_SUB('2018-05-01',INTERVAL 1 YEAR);
-> '2017-05-01'
mysql> SELECT DATE_ADD('2020-12-31 23:59:59',
-> INTERVAL 1 SECOND);
```



```

-> '2021-01-01 00:00:00'
mysql> SELECT DATE_ADD('2018-12-31 23:59:59',
-> INTERVAL 1 DAY);
-> '2019-01-01 23:59:59'
mysql> SELECT DATE_ADD('2100-12-31 23:59:59',
-> INTERVAL '1:1' MINUTE_SECOND);
-> '2101-01-01 00:01:00'
mysql> SELECT DATE_SUB('2025-01-01 00:00:00',
-> INTERVAL '1 1:1:1' DAY_SECOND);
-> '2024-12-30 22:58:59'
mysql> SELECT DATE_ADD('1900-01-01 00:00:00',
-> INTERVAL '-1 10' DAY_HOUR);
-> '1899-12-30 14:00:00'
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
-> '1997-12-02'
mysql> SELECT DATE_ADD('1992-12-31 23:59:59.000002',
-> INTERVAL '1.999999' SECOND_MICROSECOND);
-> '1993-01-01 00:00:01.000001'

```

一時演算は、**INTERVAL** を + または - 演算子とともに使用して式で実行することもできます:

```

date + INTERVAL expr unit
date - INTERVAL expr unit

```

INTERVAL expr unit は、他方の側の式が日付または日付間値である場合に、+ 演算子の一方の側で許可されます。- 演算子では、間隔から日付または日付間値を抽出しても意味がないため、**INTERVAL expr unit** は右側でのみ許可されません。

```

mysql> SELECT '2018-12-31 23:59:59' + INTERVAL 1 SECOND;
-> '2019-01-01 00:00:00'
mysql> SELECT INTERVAL 1 DAY + '2018-12-31';
-> '2019-01-01'
mysql> SELECT '2025-01-01' - INTERVAL 1 SECOND;
-> '2024-12-31 23:59:59'

```

EXTRACT() 関数は、**DATE_ADD()** または **DATE_SUB()** と同じ種類の **unit** 指定子を使用しますが、日付演算を実行するのではなく、日付から部分を抽出します:

```

mysql> SELECT EXTRACT(YEAR FROM '2019-07-02');
-> 2019
mysql> SELECT EXTRACT(YEAR_MONTH FROM '2019-07-02 01:02:03');
-> 201907

```

一時間隔は、**CREATE EVENT** ステートメントで使用できます:

```

CREATE EVENT myevent
ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 HOUR
DO
UPDATE myschema.mytable SET mycol = mycol + 1;

```

指定した間隔値 (**unit** キーワードから要求されるすべての間隔部分は含まれません) が短すぎる場合は、MySQL では間隔値の左端部分が省略されているとみなされます。たとえば、**DAY_SECOND** の **unit** を指定した場合は、**expr** の値には日、時間、分、秒の部分が含まれるとみなされます。'1:10' のような値を指定すると、MySQL では日と時間の部分が欠落していて、値は分と秒を表しているとみなされます。つまり、'1:10' **DAY_SECOND** は、'1:10' **MINUTE_SECOND** と同等の方法で解釈されます。これは、MySQL で **TIME** 値が時間ではなく経過時間を表していると解釈される方法に類似しています。

expr は文字列として扱われるため、**INTERVAL** で文字列以外の値を指定する場合は注意してください。たとえば、間隔指定子が **HOUR_MINUTE** の場合、6/4 は 6 時間 4 分として扱われますが、6/4 は 1.5000 と評価され、1 時間 5000 分として扱われます:

```

mysql> SELECT '6/4', 6/4;
-> 1.5000
mysql> SELECT DATE_ADD('2019-01-01', INTERVAL '6/4' HOUR_MINUTE);
-> '2019-01-01 06:04:00'
mysql> SELECT DATE_ADD('2019-01-01', INTERVAL 6/4 HOUR_MINUTE);
-> '2019-01-04 12:20:00'

```

間隔値が予想どおりに解釈されるようにするには、**CAST()** 演算を使用します。6/4 を 1 時間 5 分として処理するには、小数点以下の桁数が 1 桁の **DECIMAL** 値にキャストします。

```
mysql> SELECT CAST(6/4 AS DECIMAL(3,1));
-> 1.5
mysql> SELECT DATE_ADD('1970-01-01 12:00:00',
-> INTERVAL CAST(6/4 AS DECIMAL(3,1)) HOUR_MINUTE);
-> '1970-01-01 13:05:00'
```

時間部分が含まれるものを日付値に加算したり、日付値から減算したりすると、自動的に結果が日付時間値に変換されます。

```
mysql> SELECT DATE_ADD('2023-01-01', INTERVAL 1 DAY);
-> '2023-01-02'
mysql> SELECT DATE_ADD('2023-01-01', INTERVAL 1 HOUR);
-> '2023-01-01 01:00:00'
```

MONTH、YEAR_MONTH、または YEAR を加算した結果の日付に、新しい月の最大日数よりも大きな日が含まれる場合は、その日が新しい月の最大日数に調整されます。

```
mysql> SELECT DATE_ADD('2019-01-30', INTERVAL 1 MONTH);
-> '2019-02-28'
```

日付算術演算では、完全な日付が必須であるため、'2016-07-00' のような不完全な日付や、誤った形式の日付では正常に機能しません。

```
mysql> SELECT DATE_ADD('2016-07-00', INTERVAL 1 DAY);
-> NULL
mysql> SELECT '2005-03-32' + INTERVAL 1 MONTH;
-> NULL
```

9.6 クエリー属性

SQL ステートメントの最も表示される部分は、ステートメントのテキストです。MySQL 8.0.23 以降、クライアントは、実行のためにサーバーに送信される次のステートメントに適用されるクエリー属性を定義することもできます：

- 属性は、ステートメントを送信する前に定義されます。
- 属性はステートメントの実行が終了するまで存在し、その時点で属性セットがクリアされます。
- 属性は存在しますが、サーバー側でアクセスできます。

クエリー属性の使用法の例を次に示します：

- web アプリケーションは、データベースクエリーを生成するページを生成し、クエリーごとに、そのクエリーを生成したページの URL を追跡する必要があります。
- アプリケーションは、監査プラグインやクエリーリライトプラグインなどのプラグインで使用するために、各クエリーに追加の処理情報を渡します。

MySQL では、クエリー文字列に含まれる特別にフォーマットされたコメントなどの回避策を使用せずにこれらの機能をサポートしています。このセクションの残りの部分では、満たす必要がある前提条件など、クエリー属性のサポートの使用法について説明します。

- [クエリー属性の定義およびアクセス](#)
- [クエリー属性の使用の前提条件](#)
- [クエリー属性のユーザー定義関数](#)

クエリー属性の定義およびアクセス

MySQL C API を使用するアプリケーションでは、`mysql_bind_param()` 関数をコールしてクエリー属性を定義します。`mysql_bind_param()` を参照してください。その他の MySQL コネクタでは、クエリー属性のサポートも提供される場合があります。個々のコネクタのドキュメントを参照してください。

mysql クライアントには、最大 32 の属性名と値のペアを定義できる `query_attributes` コマンドがあります。[セクション 4.5.1.2 「mysql クライアントコマンド」](#) を参照してください。

クエリー属性名は、`character_set_client` システム変数で示される文字セットを使用して転送されます。

属性が定義されている SQL ステートメント内のクエリー属性にアクセスするには、[クエリー属性の使用の前提条件](#)の説明に従って `query_attributes` コンポーネントをインストールします。コンポーネントは、属性名引数を取り、属性値を文字列として返す `mysql_query_attribute_string()` ユーザー定義関数 (UDF) を実装します。属性が存在しない場合は `NULL` を返します。[クエリー属性のユーザー定義関数](#)を参照してください。

次の例では、`mysql` クライアントの `query_attributes` コマンドを使用して属性名と値のペアを定義し、`mysql_query_attribute_string()` UDF を使用して名前属性値にアクセスします。

この例では、`n1` および `n2` という名前の 2 つの属性を定義します。最初の `SELECT` では、これらの属性を取得する方法を示し、存在しない属性 (`n3`) を取得すると `NULL` が返されることも示します。2 番目の `SELECT` は、属性が複数のステートメントにわたって永続化されないことを示しています。

```
mysql> query_attributes n1 v1 n2 v2;
mysql> SELECT
  mysql_query_attribute_string('n1') AS 'attr 1',
  mysql_query_attribute_string('n2') AS 'attr 2',
  mysql_query_attribute_string('n3') AS 'attr 3';
+-----+-----+-----+
| attr 1 | attr 2 | attr 3 |
+-----+-----+-----+
| v1     | v2     | NULL   |
+-----+-----+-----+

mysql> SELECT
  mysql_query_attribute_string('n1') AS 'attr 1',
  mysql_query_attribute_string('n2') AS 'attr 2';
+-----+-----+
| attr 1 | attr 2 |
+-----+-----+
| NULL  | NULL  |
+-----+-----+
```

2 番目の `SELECT` ステートメントで示されているように、特定のステートメントの前に定義された属性は、そのステートメントでのみ使用可能であり、ステートメントの実行後にクリアされます。複数のステートメントで属性値を使用するには、属性値を変数に割り当てます。次の例では、これを行う方法を示し、属性値が変数を使用して後続のステートメントで使用可能であり、`mysql_query_attribute_string()` をコールしては使用できないことを示します：

```
mysql> query_attributes n1 v1 n2 v2;
mysql> SET
  @attr1 = mysql_query_attribute_string('n1'),
  @attr2 = mysql_query_attribute_string('n2');

mysql> SELECT
  @attr1, mysql_query_attribute_string('n1') AS 'attr 1',
  @attr2, mysql_query_attribute_string('n2') AS 'attr 2';
+-----+-----+-----+-----+
| @attr1 | attr 1 | @attr2 | attr 2 |
+-----+-----+-----+-----+
| v1     | NULL  | v2     | NULL  |
+-----+-----+-----+-----+
```

属性は、後で使用するためにテーブルに格納することで保存することもできます：

```
mysql> CREATE TABLE t1 (c1 CHAR(20), c2 CHAR(20));

mysql> query_attributes n1 v1 n2 v2;
mysql> INSERT INTO t1 (c1, c2) VALUES(
  mysql_query_attribute_string('n1'),
  mysql_query_attribute_string('n2')
);

mysql> SELECT * FROM t1;
+-----+-----+
| c1   | c2   |
+-----+-----+
| v1   | v2   |
+-----+-----+
```

クエリー属性には、次の制限事項があります：

- サーバーにステートメントを送信して実行する前に複数の属性定義操作が発生した場合、最新の定義操作が適用され、以前の操作で定義された属性が置換されます。
- 複数の属性が同じ名前前で定義されている場合、属性値を取得しようとすると未定義の結果になります。
- 空の名前で定義された属性は名前前で取得できません。
- 属性は、`PREPARE` で準備されたステートメントでは使用できません。
- `mysql_query_attribute_string()` UDF は DDL ステートメントでは使用できません。
- 属性はレプリケートされません。 `mysql_query_attribute_string()` UDF を呼び出すステートメントは、すべてのサーバーで同じ値を取得するわけではありません。

クエリー属性の使用の前提条件

属性が定義されている SQL ステートメント内のクエリー属性にアクセスするには、`query_attributes` コンポーネントをインストールする必要があります。これを行うには、次のステートメントを使用します:

```
INSTALL COMPONENT "file://component_query_attributes";
```

コンポーネントのインストールは、サーバーの起動ごとに実行する必要のない一度限りの操作です。 `INSTALL COMPONENT` によってコンポーネントがロードされ、`mysql.component` システムテーブルにも登録されて、後続のサーバー起動時にロードされます。

`query_attributes` コンポーネントは、クエリー属性にアクセスして `mysql_query_attribute_string()` UDF を実装します。 [セクション5.5.4「クエリー属性コンポーネント」](#)を参照してください。

`query_attributes` コンポーネントをアンインストールするには、次のステートメントを使用します:

```
UNINSTALL COMPONENT "file://component_query_attributes";
```

`UNINSTALL COMPONENT` はコンポーネントをアンロードし、`mysql.component` システムテーブルから登録解除して、後続のサーバー起動時にロードされないようにします。

`query_attributes` コンポーネントをインストールおよびアンインストールすると、コンポーネントが実装する `mysql_query_attribute_string()` 関数がインストールおよびアンインストールされるため、`CREATE FUNCTION` または `DROP FUNCTION` を使用して行う必要はありません。

クエリー属性のユーザー定義関数

- `mysql_query_attribute_string(name)`

アプリケーションでは、サーバーに送信される次のクエリーに適用される属性を定義できます。MySQL 8.0.23 で使用可能な `mysql_query_attribute_string()` 関数は、属性名を指定して属性値を文字列として戻します。この関数を使用すると、クエリーは、クエリーに適用される属性の値にアクセスして組み込むことができます。

`mysql_query_attribute_string()` は、`query_attributes` コンポーネントをインストールすることによってインストールされます。クエリー属性の目的および使用方法についても説明している [セクション9.6「クエリー属性」](#)を参照してください。

引数:

- `name`: 属性名。

戻り値:

成功した場合は文字列として属性値を返し、属性が存在しない場合は `NULL` を返します。

例:

次の例では、`mysql` クライアントの `query_attributes` コマンドを使用して、`mysql_query_attribute_string()` で取得できるクエリー属性を定義します。 `SELECT` は、存在しない属性 (`n3`) を取得すると `NULL` が返されることを示しています。

```
mysql> query_attributes n1 v1 n2 v2;
mysql> SELECT
-> mysql_query_attribute_string('n1') AS 'attr 1',
-> mysql_query_attribute_string('n2') AS 'attr 2',
-> mysql_query_attribute_string('n3') AS 'attr 3';
+-----+-----+-----+
| attr 1 | attr 2 | attr 3 |
+-----+-----+-----+
| v1     | v2     | NULL   |
+-----+-----+-----+
```

9.7 コメント

MySQL Server では、次の 3 つのコメントスタイルをサポートしています。

- #文字から行末まで。
- -- シーケンスから行末まで。MySQL では、-- (二重ダッシュ) のコメントスタイルは、2 番目のダッシュに少なくとも 1 つの空白または制御文字 (空白、タブ、改行など) を続ける必要があります。 [セクション1.7.2.4「コメントの先頭としての「--」」](#) で述べているように、この構文は標準 SQL のコメントの構文とは少し異なります。
- C プログラミング言語のように、/* シーケンスから次の */ シーケンスまで。この構文では、開始と終了のシーケンスは同じ行にある必要はないので、複数の行にわたってコメントを記すことができます。

次の例には、3 つのコメントスタイルがすべて示されています。

```
mysql> SELECT 1+1; # This comment continues to the end of line
mysql> SELECT 1+1; -- This comment continues to the end of line
mysql> SELECT 1 /* this is an in-line comment */ + 1;
mysql> SELECT 1+
/*
this is a
multiple-line comment
*/
1;
```

ネストされたコメントはサポートされておらず、非推奨になりました。将来の MySQL リリースで削除される予定です。(一部の条件下では、ネストされたコメントが可能な場合がありますが、通常は可能ではなく、ユーザーはネストされたコメントを使用しないでください。)

MySQL Server では、C 形式のコメントの特定のバリエーションがサポートされています。これらでは、次の形式のコメントを使用することにより、MySQL 拡張を含んでいるが、移植性を維持しているコードを記述できます。

```
/*! MySQL-specific code */
```

この場合、MySQL Server は他の SQL ステートメントと同様にコメント内のコードを解析して実行しますが、他の SQL サーバーは拡張を無視する必要があります。たとえば、MySQL Server は次のステートメントの **STRAIGHT_JOIN** キーワードを認識しますが、他のサーバーは認識できません:

```
SELECT /*! STRAIGHT_JOIN */ col1 FROM table1,table2 WHERE ...
```

!文字のあとにバージョン番号を追加すると、コメント内の構文は、MySQL のバージョンが指定されたバージョン番号以上の場合にだけ実行されます。次のコメントの **KEY_BLOCK_SIZE** キーワードは、MySQL 5.1.10 以上のサーバーによってのみ実行されます:

```
CREATE TABLE t1(a INT, KEY (a)) /*!50110 KEY_BLOCK_SIZE=1024 */;
```

前述のコメントの構文は、**mysqld** サーバーによる SQL ステートメントの構文解析に適用されます。**mysql** クライアントプログラムは、ステートメントをサーバーに送信する前に、その一部の構文解析も実行します。(これは複数ステートメント入力行で、ステートメント境界を決定するために行われます。) サーバーと **mysql** クライアントサーバーの違いの詳細は、[セクション4.5.1.6「mysql クライアントのヒント」](#) を参照してください。

/*!12345 ... */形式のコメントはサーバーに格納されません。この形式を使用してストアードプログラムにコメントを付ける場合、コメントはプログラム本体に保持されません。

オプティマイザヒントの指定には、C 形式のコメント構文の別のバリエーションが使用されます。ヒントコメントには、/ * コメントの開始シーケンスに続く + 文字が含まれます。例:

```
SELECT /*+ BKA(t1) */ FROM ... ;
```

詳細は、[セクション8.9.3「オプティマイザヒント」](#)を参照してください。

複数行の`/* ... */`コメント内での`\C`などの短い形式の `mysql` コマンドの使用はサポートされていません。ショートフォームコマンドは、オブジェクト定義に格納されている`/*+ ... */`オプティマイザヒントコメントと同様に、単一行の`/*! ... */`バージョンコメント内で機能します。オプティマイザヒントのコメントがオブジェクト定義に格納され、`mysql` でリロードされたときにダンプファイルによってそのようなコマンドが実行されることが懸念される場合は、`--binary-mode` オプションを指定して `mysql` を起動するか、`mysql` 以外のリロードクライアントを使用します。

第 10 章 文字セット、照合順序、Unicode

目次

10.1 一般の文字セットおよび照合順序	1686
10.2 MySQL での文字セットと照合順序	1687
10.2.1 文字セットレパートリー	1689
10.2.2 メタデータ用の UTF-8	1691
10.3 文字セットと照合順序の指定	1692
10.3.1 照合の命名規則	1692
10.3.2 サーバー文字セットおよび照合順序	1693
10.3.3 データベース文字セットおよび照合順序	1693
10.3.4 テーブル文字セットおよび照合順序	1695
10.3.5 カラム文字セットおよび照合順序	1695
10.3.6 文字列リテラルの文字セットおよび照合順序	1696
10.3.7 各国語文字セット	1698
10.3.8 文字セットイントロデューサ	1698
10.3.9 文字セットと照合順序の割り当ての例	1700
10.3.10 ほかの DBMS との互換性	1701
10.4 接続文字セットおよび照合順序	1701
10.5 アプリケーションの文字セットおよび照合順序の構成	1707
10.6 エラーメッセージ文字セット	1708
10.7 カラム文字セットの変換	1709
10.8 照合順序の問題	1710
10.8.1 SQL ステートメントでの COLLATE の使用	1710
10.8.2 COLLATE 句の優先順位	1711
10.8.3 文字セットと照合順序の互換性	1711
10.8.4 式での照合の強制性	1711
10.8.5 バイナリ照合順序と_bin 照合順序	1712
10.8.6 照合順序の効果の例	1715
10.8.7 INFORMATION_SCHEMA 検索での照合の使用	1716
10.9 Unicode のサポート	1718
10.9.1 utf8mb4 文字セット (4 バイトの UTF-8 Unicode エンコーディング)	1719
10.9.2 utf8mb3 文字セット (3 バイトの UTF-8 Unicode エンコーディング)	1720
10.9.3 utf8 文字セット (utf8mb3 のエイリアス)	1720
10.9.4 ucs2 文字セット (UCS-2 Unicode エンコーディング)	1721
10.9.5 utf16 文字セット (UTF-16 Unicode エンコーディング)	1721
10.9.6 utf16le 文字セット (UTF-16LE Unicode エンコーディング)	1721
10.9.7 utf32 文字セット (UTF-32 Unicode エンコーディング)	1721
10.9.8 3 バイト Unicode 文字セットと 4 バイト Unicode 文字セット間の変換	1722
10.10 サポートされる文字セットおよび照合順序	1724
10.10.1 Unicode 文字セット	1725
10.10.2 西ヨーロッパの文字セット	1731
10.10.3 中央ヨーロッパの文字セット	1733
10.10.4 南ヨーロッパおよび中東の文字セット	1733
10.10.5 バルト語の文字セット	1734
10.10.6 キリル文字の文字セット	1734
10.10.7 アジアの文字セット	1735
10.10.8 バイナリ文字セット	1739
10.11 文字セットの制約	1740
10.12 エラーメッセージ言語の設定	1740
10.13 文字セットの追加	1741
10.13.1 文字定義配列	1742
10.13.2 複雑な文字セットの文字列照合のサポート	1743
10.13.3 複雑な文字セットのマルチバイト文字のサポート	1743
10.14 文字セットへの照合順序の追加	1744
10.14.1 照合順序の実装タイプ	1745

10.14.2 照合順序 ID の選択	1747
10.14.3 8 ビットの文字セットへの単純な照合順序の追加	1748
10.14.4 Unicode 文字セットへの UCA 照合順序の追加	1749
10.15 文字セットの構成	1755
10.16 MySQL Server のロケールサポート	1756

MySQL では、さまざまな文字セットを使用してデータを格納し、さまざまな照合順序に従って比較を実行できます。デフォルトの MySQL サーバー文字セットおよび照合順序は `utf8mb4` および `utf8mb4_0900_ai_ci` ですが、サーバー、データベース、テーブル、カラムおよび文字列リテラルレベルで文字セットを指定できます。

この章では次のトピックについて説明します。

- 文字セットと照合順序とは。
- 文字セットの割り当てに対する複数レベルのデフォルトシステム。
- 文字セットと照合順序を指定するための構文。
- 影響を受ける関数と演算。
- Unicode のサポート。
- 使用可能な文字セットと照合順序 (ノート付き)。
- エラーメッセージの言語の選択。
- 曜日と月の名前に使用するロケールの選択。

文字セットの問題は、データストレージだけではなく、クライアントプログラムと MySQL Server との通信にも影響を与えます。デフォルトと異なる文字セットを使用してクライアントプログラムとサーバー間の通信を行う場合、どの文字セットを使用するのかが示す必要があります。たとえば、`utf8` Unicode 文字セットを使用するには、サーバー接続後に次のステートメントを発行してください。

```
SET NAMES 'utf8';
```

アプリケーションで使用する文字セットの構成と、クライアント/サーバー間の通信に関する文字セット関連の問題の詳細は、[セクション10.5「アプリケーションの文字セットおよび照合順序の構成」](#) および [セクション10.4「接続文字セットおよび照合順序」](#) を参照してください。

10.1 一般の文字セットおよび照合順序

文字セットとは、記号とエンコーディングのセットです。照合順序とは、文字セット内の文字を比較するためのルールを集めたものです。架空の文字セットを例にして、文字セットと照合順序の違いを見てみましょう。

4 文字のアルファベットがあるとします: `A, B, a, b`。各文字に数字を付けます: `A = 0, B = 1, a = 2, b = 3`。文字 `A` は記号、数字 `0` は `A` 用の encoding で、4 文字すべてとそのエンコーディングの組合せは文字セットです。

`A` と `B` の 2 つの文字列値を比較するとします。これを行う最も簡単な方法は、エンコーディングを確認することです: `0` (`A` の場合)、`1` (`B` の場合)。`0` は `1` より小さいため、`A` は `B` より小さいと言います。今ここで行なったのは、文字セットに対する照合順序の適用です。照合順序はルールの集まりであり、この場合、ルールは「エンコーディングの比較」の 1 つだけになります。これは可能な照合順序のうちでもっとも単純なものであり、バイナリ照合順序と呼ばれています。

しかし、小文字と大文字が同等であることを表すにはどうなるのでしょうか。この場合、少なくとも 2 つのルールがあります: (1) 小文字の `a` および `b` を `A` および `B` と同等の文字として扱い、(2) エンコーディングを比較します。これは大文字と小文字を区別しない照合順序と呼ばれます。バイナリ照合順序よりも少し複雑になります。

実際には、ほとんどの文字セットに多数の文字があります: `A` および `B` だけでなく、アルファベット全体でもあり、数千文字を含む複数のアルファベットまたは東部手書きシステムに加えて、特殊記号や句読点も多数含まれる場合があります。また、ほとんどの照合には、大文字と小文字を区別するかどうかだけでなく、アクセントを区別するかも

うか(「アクセント」はドイツ語Öのように文字に添付されたマーク)、複数文字マッピング(ドイツ語照合のいずれかでÖ=OEとなるルールなど)に関する多くのルールがあります。

MySQL では以下が可能です。

- さまざまな文字セットを使用して文字列を格納します。
- さまざまな照合順序を使用して文字列を比較します。
- 文字セットまたは照合順序が異なる文字列を、同じサーバー、同じデータベース、または同じテーブル内にも混在させます。
- どのレベルでも文字セットと照合順序を指定できるようにします。

これらの機能を効果的に使用するには、使用可能な文字セットと照合順序、デフォルトの変更方法、および文字列演算子と関数の動作への影響を把握する必要があります。

10.2 MySQL での文字セットと照合順序

MySQL Server では、複数の Unicode 文字セットを含む複数の文字セットがサポートされます。使用可能な文字セットを表示するには、`INFORMATION_SCHEMA.CHARACTER_SETS` テーブルまたは `SHOW CHARACTER SET` ステートメントを使用します。リストの一部は次のとおりです。完全な情報については、[セクション10.10「サポートされる文字セットおよび照合順序」](#)を参照してください。

```
mysql> SHOW CHARACTER SET;
+-----+-----+-----+-----+
| Charset | Description          | Default collation | Maxlen |
+-----+-----+-----+-----+
| big5    | Big5 Traditional Chinese | big5_chinese_ci   | 2      |
| binary  | Binary pseudo charset  | binary            | 1      |
| ...     | ...                   | ...               | ...    |
| latin1  | cp1252 West European  | latin1_swedish_ci | 1      |
| ...     | ...                   | ...               | ...    |
| ucs2    | UCS-2 Unicode          | ucs2_general_ci  | 2      |
| ...     | ...                   | ...               | ...    |
| utf8    | UTF-8 Unicode          | utf8_general_ci  | 3      |
| utf8mb4 | UTF-8 Unicode          | utf8mb4_0900_ai_ci | 4      |
| ...     | ...                   | ...               | ...    |
```

デフォルトでは、`SHOW CHARACTER SET` ステートメントは使用可能なすべての文字セットを表示します。一致させる文字セット名を指定するオプションの `LIKE` 句または `WHERE` 句を取ります。次の例は、Unicode 文字セット(Unicode 変換形式に基づく文字セット)の一部を示しています：

```
mysql> SHOW CHARACTER SET LIKE 'utf%';
+-----+-----+-----+-----+
| Charset | Description          | Default collation | Maxlen |
+-----+-----+-----+-----+
| utf16   | UTF-16 Unicode      | utf16_general_ci  | 4      |
| utf16le | UTF-16LE Unicode    | utf16le_general_ci | 4      |
| utf32   | UTF-32 Unicode      | utf32_general_ci  | 4      |
| utf8    | UTF-8 Unicode        | utf8_general_ci   | 3      |
| utf8mb4 | UTF-8 Unicode        | utf8mb4_0900_ai_ci | 4      |
+-----+-----+-----+-----+
```

特定の文字セットには常に少なくとも1つの照合順序があり、ほとんどの文字セットには複数の照合順序があります。文字セットの表示照合順序をリストするには、`INFORMATION_SCHEMA.COLLATIONS` テーブルまたは `SHOW COLLATION` ステートメントを使用します。

デフォルトでは、`SHOW COLLATION` ステートメントは使用可能なすべての照合を表示します。表示する照合名を指定するオプションの `LIKE` 句または `WHERE` 句を取ります。たとえば、デフォルトの文字セット `utf8mb4` の照合順序を表示するには、次のステートメントを使用します：

```
mysql> SHOW COLLATION WHERE Charset = 'utf8mb4';
+-----+-----+-----+-----+-----+
| Collation | Charset | Id | Default | Compiled | Sortlen | Pad_attribute |
+-----+-----+-----+-----+-----+
| utf8mb4_0900_ai_ci | utf8mb4 | 330 | utf8mb4_0900_ai_ci | utf8mb4_0900_ai_ci | 4 |  |
| utf8mb4_0900_as_ci | utf8mb4 | 331 | utf8mb4_0900_as_ci | utf8mb4_0900_as_ci | 4 |  |
| utf8mb4_0900_bin | utf8mb4 | 332 | utf8mb4_0900_bin | utf8mb4_0900_bin | 4 |  |
| utf8mb4_0900_ci_as | utf8mb4 | 333 | utf8mb4_0900_ci_as | utf8mb4_0900_ci_as | 4 |  |
| utf8mb4_0900_ci_bin | utf8mb4 | 334 | utf8mb4_0900_ci_bin | utf8mb4_0900_ci_bin | 4 |  |
| utf8mb4_0900_general_ci | utf8mb4 | 335 | utf8mb4_0900_general_ci | utf8mb4_0900_general_ci | 4 |  |
| utf8mb4_0900_no_case | utf8mb4 | 336 | utf8mb4_0900_no_case | utf8mb4_0900_no_case | 4 |  |
| utf8mb4_32bit_ci_as | utf8mb4 | 337 | utf8mb4_32bit_ci_as | utf8mb4_32bit_ci_as | 4 |  |
| utf8mb4_32bit_ci_bin | utf8mb4 | 338 | utf8mb4_32bit_ci_bin | utf8mb4_32bit_ci_bin | 4 |  |
| utf8mb4_bin | utf8mb4 | 339 | utf8mb4_bin | utf8mb4_bin | 4 |  |
+-----+-----+-----+-----+-----+
```

utf8mb4_0900_ai_ci	utf8mb4 255 Yes Yes 0 NO PAD
utf8mb4_0900_as_ci	utf8mb4 305 Yes 0 NO PAD
utf8mb4_0900_as_cs	utf8mb4 278 Yes 0 NO PAD
utf8mb4_0900_bin	utf8mb4 309 Yes 1 NO PAD
utf8mb4_bin	utf8mb4 46 Yes 1 PAD SPACE
utf8mb4_croatian_ci	utf8mb4 245 Yes 8 PAD SPACE
utf8mb4_cs_0900_ai_ci	utf8mb4 266 Yes 0 NO PAD
utf8mb4_cs_0900_as_cs	utf8mb4 289 Yes 0 NO PAD
utf8mb4_czech_ci	utf8mb4 234 Yes 8 PAD SPACE
utf8mb4_danish_ci	utf8mb4 235 Yes 8 PAD SPACE
utf8mb4_da_0900_ai_ci	utf8mb4 267 Yes 0 NO PAD
utf8mb4_da_0900_as_cs	utf8mb4 290 Yes 0 NO PAD
utf8mb4_de_pb_0900_ai_ci	utf8mb4 256 Yes 0 NO PAD
utf8mb4_de_pb_0900_as_cs	utf8mb4 279 Yes 0 NO PAD
utf8mb4_eo_0900_ai_ci	utf8mb4 273 Yes 0 NO PAD
utf8mb4_eo_0900_as_cs	utf8mb4 296 Yes 0 NO PAD
utf8mb4_esperanto_ci	utf8mb4 241 Yes 8 PAD SPACE
utf8mb4_estonian_ci	utf8mb4 230 Yes 8 PAD SPACE
utf8mb4_es_0900_ai_ci	utf8mb4 263 Yes 0 NO PAD
utf8mb4_es_0900_as_cs	utf8mb4 286 Yes 0 NO PAD
utf8mb4_es_trad_0900_ai_ci	utf8mb4 270 Yes 0 NO PAD
utf8mb4_es_trad_0900_as_cs	utf8mb4 293 Yes 0 NO PAD
utf8mb4_et_0900_ai_ci	utf8mb4 262 Yes 0 NO PAD
utf8mb4_et_0900_as_cs	utf8mb4 285 Yes 0 NO PAD
utf8mb4_general_ci	utf8mb4 45 Yes 1 PAD SPACE
utf8mb4_german2_ci	utf8mb4 244 Yes 8 PAD SPACE
utf8mb4_hr_0900_ai_ci	utf8mb4 275 Yes 0 NO PAD
utf8mb4_hr_0900_as_cs	utf8mb4 298 Yes 0 NO PAD
utf8mb4_hungarian_ci	utf8mb4 242 Yes 8 PAD SPACE
utf8mb4_hu_0900_ai_ci	utf8mb4 274 Yes 0 NO PAD
utf8mb4_hu_0900_as_cs	utf8mb4 297 Yes 0 NO PAD
utf8mb4_icecelandic_ci	utf8mb4 225 Yes 8 PAD SPACE
utf8mb4_is_0900_ai_ci	utf8mb4 257 Yes 0 NO PAD
utf8mb4_is_0900_as_cs	utf8mb4 280 Yes 0 NO PAD
utf8mb4_ja_0900_as_cs	utf8mb4 303 Yes 0 NO PAD
utf8mb4_ja_0900_as_cs_ks	utf8mb4 304 Yes 24 NO PAD
utf8mb4_latvian_ci	utf8mb4 226 Yes 8 PAD SPACE
utf8mb4_la_0900_ai_ci	utf8mb4 271 Yes 0 NO PAD
utf8mb4_la_0900_as_cs	utf8mb4 294 Yes 0 NO PAD
utf8mb4_lithuanian_ci	utf8mb4 236 Yes 8 PAD SPACE
utf8mb4_lt_0900_ai_ci	utf8mb4 268 Yes 0 NO PAD
utf8mb4_lt_0900_as_cs	utf8mb4 291 Yes 0 NO PAD
utf8mb4_lv_0900_ai_ci	utf8mb4 258 Yes 0 NO PAD
utf8mb4_lv_0900_as_cs	utf8mb4 281 Yes 0 NO PAD
utf8mb4_persian_ci	utf8mb4 240 Yes 8 PAD SPACE
utf8mb4_pl_0900_ai_ci	utf8mb4 261 Yes 0 NO PAD
utf8mb4_pl_0900_as_cs	utf8mb4 284 Yes 0 NO PAD
utf8mb4_polish_ci	utf8mb4 229 Yes 8 PAD SPACE
utf8mb4_romanian_ci	utf8mb4 227 Yes 8 PAD SPACE
utf8mb4_roman_ci	utf8mb4 239 Yes 8 PAD SPACE
utf8mb4_ro_0900_ai_ci	utf8mb4 259 Yes 0 NO PAD
utf8mb4_ro_0900_as_cs	utf8mb4 282 Yes 0 NO PAD
utf8mb4_ru_0900_ai_ci	utf8mb4 306 Yes 0 NO PAD
utf8mb4_ru_0900_as_cs	utf8mb4 307 Yes 0 NO PAD
utf8mb4_sinhala_ci	utf8mb4 243 Yes 8 PAD SPACE
utf8mb4_sk_0900_ai_ci	utf8mb4 269 Yes 0 NO PAD
utf8mb4_sk_0900_as_cs	utf8mb4 292 Yes 0 NO PAD
utf8mb4_slovak_ci	utf8mb4 237 Yes 8 PAD SPACE
utf8mb4_slovenian_ci	utf8mb4 228 Yes 8 PAD SPACE
utf8mb4_sl_0900_ai_ci	utf8mb4 260 Yes 0 NO PAD
utf8mb4_sl_0900_as_cs	utf8mb4 283 Yes 0 NO PAD
utf8mb4_spanish2_ci	utf8mb4 238 Yes 8 PAD SPACE
utf8mb4_spanish_ci	utf8mb4 231 Yes 8 PAD SPACE
utf8mb4_sv_0900_ai_ci	utf8mb4 264 Yes 0 NO PAD
utf8mb4_sv_0900_as_cs	utf8mb4 287 Yes 0 NO PAD
utf8mb4_swedish_ci	utf8mb4 232 Yes 8 PAD SPACE
utf8mb4_tr_0900_ai_ci	utf8mb4 265 Yes 0 NO PAD
utf8mb4_tr_0900_as_cs	utf8mb4 288 Yes 0 NO PAD
utf8mb4_turkish_ci	utf8mb4 233 Yes 8 PAD SPACE
utf8mb4_unicode_520_ci	utf8mb4 246 Yes 8 PAD SPACE
utf8mb4_unicode_ci	utf8mb4 224 Yes 8 PAD SPACE
utf8mb4_vietnamese_ci	utf8mb4 247 Yes 8 PAD SPACE
utf8mb4_vi_0900_ai_ci	utf8mb4 277 Yes 0 NO PAD
utf8mb4_vi_0900_as_cs	utf8mb4 300 Yes 0 NO PAD
utf8mb4_zh_0900_as_cs	utf8mb4 308 Yes 0 NO PAD

これらの照合の詳細は、[セクション10.10.1「Unicode 文字セット」](#)を参照してください。

照合順序には次のような一般的な特徴があります。

- 2つの異なる文字セットで、同じ照合順序を共有できません。
- 各文字セットにはデフォルト照合があります。たとえば、`utf8mb4` および `latin1` のデフォルトの照合は、それぞれ `utf8mb4_0900_ai_ci` および `latin1_swedish_ci` です。`INFORMATION_SCHEMA CHARACTER_SETS` テーブルおよび `SHOW CHARACTER SET` ステートメントは、各文字セットのデフォルトの照合順序を示します。`INFORMATION_SCHEMA COLLATIONS` テーブルおよび `SHOW COLLATION` ステートメントには、各照合順序が文字セットのデフォルトであるかどうかを示すカラムがあります (デフォルトである場合は `Yes`、そうでない場合は空)。
- 照合順序名は、関連付けられている文字セットの名前で始まり、通常は、他の照合順序特性を示す1つ以上の接尾辞が続きます。ネーミング規則の詳細は、[セクション10.3.1「照合の命名規則」](#)を参照してください。

文字セットに複数の照合順序がある場合、特定のアプリケーションに最適な照合順序が明確でない可能性があります。不適切な照合を選択しないようにするには、代表的なデータ値を使用していくつかの比較を実行し、特定の照合によって期待どおりに値がソートされるようにします。

10.2.1 文字セットレパートリー

文字セットのレパートリーとは、そのセット内の文字の集合です。

文字列式にはレパートリー属性があり、その値は次の2つです。

- **ASCII:** 式には ASCII 文字、つまり `U+0000` から `U+007F` までの Unicode 範囲の文字のみを含めることができます。
- **UNICODE:** `U+0000` から `U+10FFFF` の Unicode 範囲内の文字を式に含めることができます。これには、Basic Multilingual Plane (BMP) 範囲 (`U+0000` から `U+FFFF`) の文字と BMP 範囲外の補助文字 (`U+10000` から `U+10FFFF`) が含まれます。

ASCII 範囲は UNICODE 範囲のサブセットであるため、ASCII レパートリーを含む文字列は、情報を失うことなく、UNICODE レパートリーを含む文字列の文字セットに安全に変換できます。また、`ascii` 文字セットのスーパーセットである任意の文字セットに安全に変換できます。(すべての MySQL 文字セットは、`swe7` を除いて `ascii` のスーパーセットであり、スウェーデン語アクセント付き文字に一部の句読点文字が再利用されます。)

レパートリーを使用すると、照合強制性のルールがあいまいさを解決するには不十分な場合に MySQL が「照合順序の組合せが不正です」エラーを返す多くの場合に、式で文字セット変換が可能になります。(強制性の詳細は、[セクション10.8.4「式での照合の強制性」](#)を参照してください。)

次の説明では、式とそのレパートリーの例を挙げ、レパートリーの使用によって、文字列式の評価がどのように変わるかを示します。

- 文字列定数のレパートリーは文字列の内容に依存し、文字列文字セットのレパートリーとは異なる場合があります。これらのステートメントを考慮してください。

```
SET NAMES utf8mb4; SELECT 'abc';
SELECT _utf8mb4'def';
```

前述の各場合、文字セットは `utf8mb4` ですが、文字列には実際に ASCII 範囲外の文字が含まれていないため、そのレパートリーは **UNICODE** ではなく **ASCII** です。

- `ascii` 文字セットを持つカラムには、その文字セットのために **ASCII** レパートリーがあります。次のテーブルでは、`c1` には **ASCII** レパートリーがあります。

```
CREATE TABLE t1 (c1 CHAR(1) CHARACTER SET ascii);
```

次の例では、レパートリーがないときにエラーが発生する場合に、レパートリーによってどのように結果が求められるようになるかを示しています。

```
CREATE TABLE t1 (
  c1 CHAR(1) CHARACTER SET latin1,
  c2 CHAR(1) CHARACTER SET ascii
);
INSERT INTO t1 VALUES ('a','b');
SELECT CONCAT(c1,c2) FROM t1;
```

レパートリーがない場合、次のエラーが発生します。

```
ERROR 1267 (HY000): Illegal mix of collations (latin1_swedish_ci,IMPLICIT)
and (ascii_general_ci,IMPLICIT) for operation 'concat'
```

レパートリーを使用すると、サブセットからスーパーセットへ (`ascii` から `latin1` へ) の変換を行うことができ、結果が返されます。

```
+-----+
| CONCAT(c1,c2) |
+-----+
| ab          |
+-----+
```

- 1 つの文字列引数を持つ関数は、その引数のレパートリーを継承します。引数に `ASCII` レパートリーがあるため、`UPPER(_utf8mb4'abc')` の結果には `ASCII` レパートリーがあります。 (`_utf8mb4` イントロデューサにもかかわらず、文字列 `'abc'` に `ASCII` 範囲外の文字が含まれていません。)
- 文字列を返すが、文字列引数を持たず、結果文字セットとして `character_set_connection` を使用する関数の場合、結果のレパートリーは、`character_set_connection` が `ascii` である場合は `ASCII` に、それ以外の場合は `UNICODE` になります。

```
FORMAT(numeric_column, 4);
```

レパートリーを使用するかどうかによって、MySQL が次の例をどのように評価するかが変わります。

```
SET NAMES ascii;
CREATE TABLE t1 (a INT, b VARCHAR(10) CHARACTER SET latin1);
INSERT INTO t1 VALUES (1,'b');
SELECT CONCAT(FORMAT(a, 4), b) FROM t1;
```

レパートリーがない場合、次のエラーが発生します。

```
ERROR 1267 (HY000): Illegal mix of collations (ascii_general_ci,COERCIBLE)
and (latin1_swedish_ci,IMPLICIT) for operation 'concat'
```

レパートリーを使用すると、次のように結果が返されます。

```
+-----+
| CONCAT(FORMAT(a, 4), b) |
+-----+
| 1.0000b                |
+-----+
```

- 複数の文字列引数を持つ関数は、`UNICODE` が `ASCII` よりも広い場合に、結果レパートリーに「ワイドスト」引数 repertoire を使用します。次の `CONCAT()` 呼び出しを考えてみます。

```
CONCAT(_ucs2 X'0041', _ucs2 X'0042')
CONCAT(_ucs2 X'0041', _ucs2 X'00C2')
```

最初のコールでは、両方の引数が `ASCII` 範囲内にあるため、レパートリーは `ASCII` です。2 番目のコールでは、2 番目の引数が `ASCII` 範囲外であるため、レパートリーは `UNICODE` です。

- 関数の戻り値のレパートリーは、結果の文字セットおよび照合順序に影響する引数のみのレパートリーに基づいて決定されます。

```
IF(column1 < column2, 'smaller', 'greater')
```

結果のレパートリーは、2 つの文字列引数 (2 番目の引数と 3 番目の引数) がどちらも `ASCII` レパートリーなので、`ASCII` です。最初の引数は、式で文字列値を使用する場合でも、結果のレパートリーにとっては重要ではありません。

10.2.2 メタデータ用の UTF-8

メタデータは「データに関するデータ」です。データベースについて記述しているすべてのものがメタデータであり、データベースの内容ではありません。したがって、カラム名、データベース名、ユーザー名、バージョン名、および `SHOW` の文字列結果のほとんどがメタデータです。 `INFORMATION_SCHEMA` 内のテーブルは定義上、データベースオブジェクトに関する情報を含んでいるので、これは、このテーブルの内容にも当てはまります。

メタデータの表現は次の要件を満たしている必要があります。

- すべてのメタデータで文字セットが一致している必要があります。それ以外の場合、 `INFORMATION_SCHEMA` 内のテーブルに対する `SHOW` ステートメントも `SELECT` ステートメントも正しく機能しません。これらの演算結果の同一カラム内の各行で文字セットが異なるからです。
- メタデータはすべての言語のすべての文字が含まれている必要があります。そうでない場合、ユーザーはそれぞれの言語を使用してカラムとテーブルに名前を付けることはできません。

両方の要件を満たすために、MySQL では、Unicode 文字セット、つまり UTF-8 でメタデータを格納します。アクセント符号付きの文字またはラテン語以外の文字を使用しなければ、混乱が生じることはありません。ただし、使用した場合は、メタデータの文字セットが UTF-8 であることを認識する必要があります。

このメタデータ要件は、 `USER()`、 `CURRENT_USER()`、 `SESSION_USER()`、 `SYSTEM_USER()`、 `DATABASE()`、および `VERSION()` の関数の戻り値で、UTF-8 文字セットがデフォルトで使用されることを意味します。

サーバーは、 `character_set_system` システム変数をメタデータ文字セットの名前に設定します。

```
mysql> SHOW VARIABLES LIKE 'character_set_system';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| character_set_system | utf8 |
+-----+-----+
```

Unicode を使用してメタデータを格納しても、サーバーが、カラムのヘッダーや `DESCRIBE` 関数の結果を、デフォルトで `character_set_system` 文字セットで返すことにはなりません。 `SELECT column1 FROM t` を使用すると、 `column1` 自体の名前が、 `utf8mb4` のデフォルト値を持つ `character_set_results` システム変数の値によって決定された文字セットでサーバーからクライアントに返されます。別の文字セットでメタデータの結果をサーバーに返させる場合は、 `SET NAMES` ステートメントを使用してサーバーに文字セット変換を強制的に実行させてください。 `SET NAMES` は `character_set_results` および関連するほかのシステム変数を設定します。(セクション10.4「[接続文字セットおよび照合順序](#)」を参照してください。) また、サーバーから結果を受け取ったあとで、クライアントプログラムが変換を実行できます。クライアントが変換を実行するとより効率的ですが、このオプションは、すべてのクライアントが常に使用できるとはかぎりません。

`character_set_results` が `NULL` に設定されている場合、変換は実行されず、サーバーはオリジナルの文字セット (`character_set_system` によって指定されたセット) を使用してメタデータを返します。

サーバーからクライアントに返されるエラーメッセージは、メタデータと同様に自動的にクライアントの文字セットに変換されます。

たとえば、 `USER()` 関数を比較または割り当てのために単一のステートメント内で使用している場合、問題はありません。MySQL が自動的に変換を実行します。

```
SELECT * FROM t1 WHERE USER() = latin1_column;
```

これが機能するのは、 `latin1_column` の内容が UTF-8 に自動的に変換されてから比較が行われるからです。

```
INSERT INTO t1 (latin1_column) SELECT USER();
```

これが機能するのは、 `USER()` の内容が `latin1` に自動的に変換されてから割り当てが行われるからです。

自動変換は SQL 標準には含まれていませんが、標準では、すべての文字セットが Unicode の「サブセット」であることが示されています。「スーパーセットに適用されるものはサブセットにも適用される」というよく知られた原則があるので、Unicode の照合順序は Unicode 以外の文字列との比較にも適用できると考えられます。文字列の強制力の詳細は、セクション10.8.4「[式での照合の強制性](#)」を参照してください。

10.3 文字セットと照合順序の指定

サーバー、データベース、テーブル、カラムの4つのレベルで、文字セットと照合順序のデフォルト設定が用意されています。以降のセクションの説明は複雑に思われますが、実際、複数レベルのデフォルト設定によって自然で明白な結果が得られることがわかっています。

CHARACTER SET は文字セットを指定する句で使用します。**CHARSET** は、**CHARACTER SET** のシノニムとして使用できます。

文字セットの問題は、データストレージだけではなく、クライアントプログラムと MySQL Server との通信にも影響を与えます。デフォルトと異なる文字セットを使用してクライアントプログラムとサーバー間の通信を行う場合、どの文字セットを使用するのかが示す必要があります。たとえば、**utf8mb4** Unicode 文字セットを使用するには、サーバーへの接続後に次のステートメントを発行します：

```
SET NAMES 'utf8mb4';
```

クライアント/サーバー間の通信に関する文字セット関連の問題の詳細は、[セクション10.4「接続文字セットおよび照合順序」](#)を参照してください。

10.3.1 照合の命名規則

MySQL 照合順序名は、次の規則に従います：

- 照合順序名は、関連付けられている文字セットの名前で始まり、通常は、他の照合順序特性を示す1つ以上の接尾辞が続きます。たとえば、**utf8mb4_general_ci** と **latin1_swedish_ci** は、それぞれ **utf8mb4** と **latin1** の文字セットの照合順序です。**binary** 文字セットには、接尾辞のない単一の照合順序 (**binary** と呼ばれる) があります。
- 言語固有の照合には、ロケールコードまたは言語名が含まれます。たとえば、トルコ語とハンガリー語のルールをそれぞれ使用して、**utf8mb4** 文字セットの **utf8mb4_tr_0900_ai_ci** および **utf8mb4_hu_0900_ai_ci** ソート文字をソートします。**utf8mb4_turkish_ci** と **utf8mb4_hungarian_ci** は似ていますが、Unicode 照合アルゴリズムの新しいバージョンに基づいています。
- 照合接尾辞は、照合で大/小文字が区別されるか、アクセントが区別されるか、またはカナが区別されるか (またはその一部の組合せ)、あるいはバイナリかを示します。次のテーブルに、これらの特性を示すために使用される接尾辞を示します。

表 10.1 照合サフィックスの意味

サフィックス	意味
_ai	アクセントを区別しない
_as	アクセントを区別する
_ci	大文字小文字を区別しない
_cs	Case-sensitive
_ks	カナを区別する
_bin	バイナリ

アクセントの区別を指定しない非バイナリ照合順序名の場合、大文字と小文字の区別によって決定されます。照合名に **_ai** または **_as** が含まれていない場合、名前に含まれる **_ci** は **_ai** を意味し、名前に含まれる **_cs** は **_as** を意味します。たとえば、**latin1_general_ci** では、大/小文字が明示的に区別されず、暗黙的にアクセントが区別されません。**latin1_general_cs** では、明示的に大/小文字が区別され、暗黙的にアクセントが区別されます。**utf8mb4_0900_ai_ci** では、大/小文字が明示的に区別されず、アクセントも区別されません。

日本語照合の場合、**_ks** 接尾辞は照合順序がカタカナに依存することを示します。つまり、カタカナ文字とひらがな文字を区別します。**_ks** 接尾辞のない日本語照合は、カナに依存せず、カタカナ文字とひらがな文字をソート用に同等に扱います。

binary 文字セットの **binary** 照合順序の場合、比較は数値バイト値に基づきます。非バイナリ文字セットの **_bin** 照合の場合、比較は、マルチバイト文字のバイト値とは異なる数値文字コード値に基づきます。**binary** 文字セットの

binary 照合順序と非バイナリ文字セットの_bin 照合順序の違いについては、[セクション10.8.5「バイナリ照合順序と_bin 照合順序」](#)を参照してください。

- Unicode 文字セットの照合順序名には、照合順序の基になる Unicode 照合アルゴリズム (UCA) のバージョンを示すバージョン番号を含めることができます。名前にバージョン番号が含まれていない UCA ベースの照合では、version-4.0.0 UCA の重みキーが使用されます。例:
 - `utf8mb4_0900_ai_ci` は UCA 9.0.0 の重みキー (<http://www.unicode.org/Public/UCA/9.0.0/allkeys.txt>) に基づいています。
 - `utf8mb4_unicode_520_ci` は UCA 5.2.0 の重みキー (<http://www.unicode.org/Public/UCA/5.2.0/allkeys.txt>) に基づいています。
 - `utf8mb4_unicode_ci` (バージョン名なし) は UCA 4.0.0 重みキー (<http://www.unicode.org/Public/UCA/4.0.0/allkeys-4.0.0.txt>) に基づいています。
- Unicode 文字セットの場合、`xxx_general_mysql500_ci` 照合順序では、元の `xxx_general_ci` 照合順序の 5.1.24 以前の順序が保持され、MySQL 5.1.24 より前に作成されたテーブルのアップグレードが許可されます (Bug #27877)。

10.3.2 サーバー文字セットおよび照合順序

MySQL Server にはサーバー文字セットとサーバー照合順序があります。デフォルトでは、これらは `utf8mb4` および `utf8mb4_0900_ai_ci` ですが、サーバーの起動時にコマンドラインまたはオプションファイルで明示的に設定し、実行時に変更できます。

サーバー文字セットおよび照合順序は最初、`mysqld` の起動時に使用するオプションに依存します。文字セットに `--character-set-server` を使用できます。これに加え、照合順序に `--collation-server` を追加できます。文字セットを指定しない場合は、`--character-set-server=utf8mb4` と同じです。文字セット (`utf8mb4` など) のみを指定し、照合順序を指定しない場合は、`utf8mb4_0900_ai_ci` が `utf8mb4` のデフォルト照合順序であるため、`--character-set-server=utf8mb4 --collation-server=utf8mb4_0900_ai_ci` と同じです。したがって、次の 3 つのコマンドを実行した結果はいずれも同じになります。

```
mysqld
mysqld --character-set-server=utf8mb4
mysqld --character-set-server=utf8mb4 \
--collation-server=utf8mb4_0900_ai_ci
```

設定を変更する手段の 1 つは再コンパイルです。ソースから構築するときに、デフォルトのサーバー文字セットおよび照合順序を変更するには、`CMake` の `DEFAULT_CHARSET` および `DEFAULT_COLLATION` オプションを使用します。例:

```
cmake . -DDEFAULT_CHARSET=latin1
```

または:

```
cmake . -DDEFAULT_CHARSET=latin1 \
-DDEFAULT_COLLATION=latin1_german1_ci
```

`mysqld` と `CMake` の両方は、文字セットと照合順序の組み合わせが有効であることを検証します。組み合わせが有効でない場合、各プログラムによってエラーメッセージが表示され、強制終了されます。

サーバー文字セットおよび照合順序は、データベース文字セットおよび照合順序が `CREATE DATABASE` ステートメントで指定されていない場合にデフォルト値として使用されます。これらにほかの用途はありません。

現在のサーバー文字セットおよび照合順序は、`character_set_server` および `collation_server` システム変数の値で判別できます。これらの変数は実行時に変更できます。

10.3.3 データベース文字セットおよび照合順序

各データベースにはデータベース文字セットとデータベース照合順序があります。`CREATE DATABASE` および `ALTER DATABASE` ステートメントには、データベース文字セットおよび照合順序を指定するためのオプション句があります。

```
CREATE DATABASE db_name
  [[DEFAULT] CHARACTER SET charset_name]
  [[DEFAULT] COLLATE collation_name]
```

```
ALTER DATABASE db_name
  [[DEFAULT] CHARACTER SET charset_name]
  [[DEFAULT] COLLATE collation_name]
```

SCHEMA というキーワードは **DATABASE** の代わりに使用できます。

CHARACTER SET および **COLLATE** 句を使用すると、文字セットと照合順序が異なる複数のデータベースを同一の MySQL Server に作成できます。

データベースオプションはデータディクショナリに格納され、**INFORMATION_SCHEMA.SCHEMATA** テーブルをチェックして調べることができます。

例:

```
CREATE DATABASE db_name CHARACTER SET latin1 COLLATE latin1_swedish_ci;
```

MySQL では、データベース文字セットとデータベース照合順序が次のように選択されます。

- **CHARACTER SET charset_name** と **COLLATE collation_name** の両方が指定されている場合、文字セット **charset_name** と照合順序 **collation_name** が使用されます。
- **CHARACTER SET charset_name** が **COLLATE** なしで指定されている場合、文字セット **charset_name** とそのデフォルトの照合順序が使用されます。各文字セットのデフォルトの照合順序を確認するには、**SHOW CHARACTER SET** ステートメントを使用するか、**INFORMATION_SCHEMA.CHARACTER_SETS** テーブルをクエリーします。
- **COLLATE collation_name** が **CHARACTER SET** なしで指定されている場合は、**collation_name** および照合 **collation_name** に関連付けられた文字セットが使用されます。
- それ以外の場合 (**CHARACTER SET** も **COLLATE** も指定されていない場合)、サーバー文字セットとサーバー照合順序が使用されます。

デフォルトのデータベースに対する文字セットと照合順序は、**character_set_database** および **collation_database** システム変数の値から判別できます。デフォルトのデータベースが変わるたびに、サーバーはこれらの変数を設定します。デフォルトのデータベースがない場合、変数は、**character_set_server** および **collation_server** という対応するサーバーレベルのシステム変数と同値になります。

特定のデータベースのデフォルト文字セットおよび照合順序を確認するには、次のステートメントを使用します。

```
USE db_name;
SELECT @@character_set_database, @@collation_database;
```

また、デフォルトのデータベースを変更しないで値を表示するには、次のステートメントを使用します。

```
SELECT DEFAULT_CHARACTER_SET_NAME, DEFAULT_COLLATION_NAME
FROM INFORMATION_SCHEMA.SCHEMATA WHERE SCHEMA_NAME = 'db_name';
```

データベース文字セットおよび照合順序は、サーバー操作の次の側面に影響します。

- **CREATE TABLE** ステートメントでは、テーブル文字セットおよび照合順序が指定されていない場合、データベース文字セットおよび照合順序がテーブル定義のデフォルト値として使用されます。これをオーバーライドするには、**CHARACTER SET** および **COLLATE** テーブルオプションを明示的に指定します。
- **CHARACTER SET** 句を含まない **LOAD DATA** ステートメントでは、サーバーは、**character_set_database** システム変数によって示された文字セットを使用して、ファイル内の情報を解釈します。これをオーバーライドするには、**CHARACTER SET** 句を明示的に指定します。
- ストアドルーチン (プロシージャおよびファンクション) の場合、宣言に **CHARACTER SET** または **COLLATE** 属性が含まれていない文字データパラメータの文字セットおよび照合順序として、ルーチン作成時に有効なデータベース文字セットおよび照合順序が使用されます。これをオーバーライドするには、**CHARACTER SET** および **COLLATE** を明示的に指定します。

10.3.4 テーブル文字セットおよび照合順序

各テーブルにはテーブル文字セットとテーブル照合順序があります。CREATE TABLE および ALTER TABLE ステートメントには、テーブル文字セットおよび照合順序を指定するためのオプション句があります。

```
CREATE TABLE tbl_name (column_list)
  [[DEFAULT] CHARACTER SET charset_name]
  [COLLATE collation_name]

ALTER TABLE tbl_name
  [[DEFAULT] CHARACTER SET charset_name]
  [COLLATE collation_name]
```

例:

```
CREATE TABLE t1 ( ... )
CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

MySQL では、テーブル文字セットとテーブル照合順序が次のように選択されます。

- CHARACTER SET charset_name と COLLATE collation_name の両方が指定されている場合、文字セット charset_name と照合順序 collation_name が使用されます。
- CHARACTER SET charset_name が COLLATE なしで指定されている場合、文字セット charset_name とそのデフォルトの照合順序が使用されます。各文字セットのデフォルトの照合順序を確認するには、SHOW CHARACTER SET ステートメントを使用するか、INFORMATION_SCHEMA CHARACTER_SETS テーブルをクエリーします。
- COLLATE collation_name が CHARACTER SET なしで指定されている場合は、collation_name および照合 collation_name に関連付けられた文字セットが使用されます。
- それ以外の場合 (CHARACTER SET も COLLATE も指定されていない場合)、データベース文字セットと照合順序が使用されます。

個々のカラム定義でカラム文字セットおよび照合順序が指定されていない場合は、テーブル文字セットおよび照合順序がカラム定義のデフォルト値として使用されます。テーブル文字セットおよび照合順序は MySQL の拡張機能です。このような機能は標準 SQL には存在しません。

10.3.5 カラム文字セットおよび照合順序

すべての「文字型」カラム (CHAR、VARCHAR、TEXT 型またはシノニム型のカラム) には、カラム文字セットおよびカラム照合順序があります。CREATE TABLE および ALTER TABLE のカラム定義構文には、カラム文字セットおよび照合順序を指定するためのオプション句があります。

```
col_name {CHAR | VARCHAR | TEXT} (col_length)
  [CHARACTER SET charset_name]
  [COLLATE collation_name]
```

これらの句は、ENUM および SET カラムにも使用できます。

```
col_name {ENUM | SET} (val_list)
  [CHARACTER SET charset_name]
  [COLLATE collation_name]
```

例:

```
CREATE TABLE t1
(
  col1 VARCHAR(5)
  CHARACTER SET latin1
  COLLATE latin1_german1_ci
);

ALTER TABLE t1 MODIFY
col1 VARCHAR(5)
CHARACTER SET latin1
```



```
COLLATE latin1_swedish_ci;
```

MySQL では、カラム文字セットとカラム照合順序が次のように選択されます。

- `CHARACTER SET charset_name` と `COLLATE collation_name` の両方が指定されている場合、文字セット `charset_name` と照合順序 `collation_name` が使用されます。

```
CREATE TABLE t1  
(  
  col1 CHAR(10) CHARACTER SET utf8 COLLATE utf8_unicode_ci  
) CHARACTER SET latin1 COLLATE latin1_bin;
```

カラムに対して文字セットと照合順序が指定されているので、それらが使用されます。このカラムには、文字セット `utf8` と照合順序 `utf8_unicode_ci` があります。

- `CHARACTER SET charset_name` が `COLLATE` なしで指定されている場合、文字セット `charset_name` とそのデフォルトの照合順序が使用されます。

```
CREATE TABLE t1  
(  
  col1 CHAR(10) CHARACTER SET utf8  
) CHARACTER SET latin1 COLLATE latin1_bin;
```

カラムに対して文字セットは指定されていますが、照合順序は指定されていません。このカラムには、文字セット `utf8` と、`utf8` のデフォルトの照合順序である `utf8_general_ci` があります。各文字セットのデフォルトの照合順序を確認するには、`SHOW CHARACTER SET` ステートメントを使用するか、`INFORMATION_SCHEMA.CHARACTER_SETS` テーブルをクエリーします。

- `COLLATE collation_name` が `CHARACTER SET` なしで指定されている場合は、`collation_name` および照合 `collation_name` に関連付けられた文字セットが使用されます。

```
CREATE TABLE t1  
(  
  col1 CHAR(10) COLLATE utf8_polish_ci  
) CHARACTER SET latin1 COLLATE latin1_bin;
```

カラムに対して照合順序は指定されていますが、文字セットは指定されていません。このカラムには照合順序 `utf8_polish_ci` があり、この照合順序に関連付けられた文字セットである `utf8` があります。

- それ以外の場合 (`CHARACTER SET` も `COLLATE` も指定されていない場合)、テーブルの文字セットと照合順序が使用されます。

```
CREATE TABLE t1  
(  
  col1 CHAR(10)  
) CHARACTER SET latin1 COLLATE latin1_bin;
```

カラムに文字セットも照合順序も指定されていないため、テーブルのデフォルトが使用されます。このカラムには、文字セット `latin1` と照合順序 `latin1_bin` があります。

`CHARACTER SET` および `COLLATE` 句は標準 SQL です。

`ALTER TABLE` を使用して、ある文字セットから別の文字セットにカラムを変換する場合、MySQL はデータ値をマップしようとしますが、文字セットに互換性がない場合、データの損失が生じる可能性があります。

10.3.6 文字列リテラルの文字セットおよび照合順序

各文字列リテラルには文字セットと照合順序があります。

単純なステートメント `SELECT 'string'` の場合、文字列には、`character_set_connection` および `collation_connection` システム変数によって定義された接続のデフォルトの文字セットと照合順序が含まれます。

文字列リテラルには、特定の文字セットおよび照合順序を使用する文字列として指定するために、オプションの文字セットイントロデューサおよび `COLLATE` 句を含めることができます：

```
[charset_name]'string' [COLLATE collation_name]
```


`_charset_name` 式は正式にはイントロデューサと呼ばれます。パーサーに「後続の文字列が文字セット `charset_name` を使用する」ことを通知します。イントロデューサは、`CONVERT()` のように文字列をイントロデューサ文字セットに変更しません。パディングは発生する可能性があります、文字列値は変更されません。イントロデューサは単なる信号です。セクション10.3.8「文字セットイントロデューサ」を参照してください。

例:

```
SELECT 'abc';
SELECT _latin1'abc';
SELECT _binary'abc';
SELECT _utf8mb4'abc' COLLATE utf8mb4_danish_ci;
```

文字セットイントロデューサと `COLLATE` 句は、標準 SQL 仕様に基づいて実装されています。

MySQL は、次の方法で文字列リテラルの文字セットおよび照合順序を決定します:

- `_charset_name` と `COLLATE collation_name` の両方が指定されている場合、文字セット `charset_name` と照合順序 `collation_name` が使用されます。 `collation_name` は、`charset_name` に対して許可された照合である必要があります。
- `_charset_name` が指定されているが、`COLLATE` が指定されていない場合は、文字セット `charset_name` とそのデフォルトの照合順序が使用されます。各文字セットのデフォルトの照合順序を確認するには、`SHOW CHARACTER SET` ステートメントを使用するか、`INFORMATION_SCHEMA CHARACTER_SETS` テーブルをクエリーします。
- `_charset_name` は指定されていないが、`COLLATE collation_name` が指定されている場合は、`character_set_connection` システム変数および照合 `collation_name` によって指定された接続のデフォルト文字セットが使用されます。 `collation_name` は、接続のデフォルト文字セットに対して許可された照合順序である必要があります。
- それ以外の場合 (`_charset_name` も `COLLATE collation_name` も指定されていない場合)、`character_set_connection` および `collation_connection` システム変数によって指定された接続のデフォルトの文字セットと照合順序が使用されます。

例:

- `latin1` 文字セットと `latin1_german1_ci` 照合順序を持つ非バイナリ文字列:

```
SELECT _latin1'Müller' COLLATE latin1_german1_ci;
```

- `utf8mb4` 文字セットとそのデフォルトの照合順序 (つまり、`utf8mb4_general_ci`) を持つ非バイナリ文字列:

```
SELECT _utf8mb4'Müller';
```

- `binary` 文字セットとそのデフォルト照合順序 (`binary`) を持つバイナリ文字列:

```
SELECT _binary'Müller';
```

- 接続のデフォルト文字セットおよび `utf8mb4_general_ci` 照合順序を持つ非バイナリ文字列 (接続文字セットが `utf8mb4` でない場合は失敗します):

```
SELECT 'Müller' COLLATE utf8mb4_general_ci;
```

- 文字列に接続デフォルト文字セットおよび照合順序が指定されている場合

```
SELECT 'Müller';
```

イントロデューサは、次の文字列の文字セットを示しますが、パーサーが文字列内でエスケープ処理を実行する方法は変更しません。エスケープは常に、`character_set_connection` で指定された文字セットに従ってパーサーが解釈します。

次の例は、イントロデューサが存在する場合でも、`character_set_connection` を使用して、エスケープ処理が行われることを示します。例では、`SET NAMES` (セクション10.4「接続文字セットおよび照合順序」で説明しているように、`character_set_connection` を変更します) を使用し、正確な文字列の内容を確認できるように `HEX()` 関数を使用して結果の文字列を表示します。

例 1:

```
mysql> SET NAMES latin1;
mysql> SELECT HEX('\n'), HEX(_sjis'\n');
+-----+-----+
| HEX('\n') | HEX(_sjis'\n') |
+-----+-----+
| E00A     | E00A           |
+-----+-----+
```

ここでは、`à` (16 進数値 `E0`) の後に、改行のエスケープシーケンスである `\n` が続きます。エスケープシーケンスは、`latin1` の `character_set_connection` 値を使用して解釈され、リテラル改行 (16 進数値 `0A`) が生成されます。これは 2 番目の文字列にも行われます。つまり、`_sjis` イントロデューサはパーサーエスケープ処理には影響しません。

例 2:

```
mysql> SET NAMES sjis;
mysql> SELECT HEX('\n'), HEX(_latin1'\n');
+-----+-----+
| HEX('\n') | HEX(_latin1'\n') |
+-----+-----+
| E05C6E   | E05C6E           |
+-----+-----+
```

ここで、`character_set_connection` は `sjis` で、`à` の順序の後に `\` (16 進数値 `05` および `5C`) が続く文字セットは有効なマルチバイト文字です。したがって、文字列の最初の 2 バイトは、単一の `sjis` 文字として解釈され、`\` はエスケープ文字として解釈されません。次の `n` (16 進数値 `6E`) は、エスケープシーケンスであるため、解釈されません。これは、2 つ目の文字列にも当てはまります。`_latin1` イントロデューサはエスケープ処理に影響しません。

10.3.7 各国語文字セット

標準 SQL では、`NCHAR` または `NATIONAL CHAR` は、`CHAR` カラムで事前定義された文字セットを使用するように指定する方法として定義されています。MySQL では、この事前定義済文字セットとして `utf8` を使用します。たとえば、次のデータ型宣言は同等です。

```
CHAR(10) CHARACTER SET utf8
NATIONAL CHARACTER(10)
NCHAR(10)
```

次も同様です。

```
VARCHAR(10) CHARACTER SET utf8
NATIONAL VARCHAR(10)
NVARCHAR(10)
NCHAR VARCHAR(10)
NATIONAL CHARACTER VARYING(10)
NATIONAL CHAR VARYING(10)
```

`N'literal'` (または `n'literal'`) を使用すると、各国文字セットの文字列を作成できます。次のステートメントは同等です。

```
SELECT N'some text';
SELECT n'some text';
SELECT _utf8'some text';
```

10.3.8 文字セットイントロデューサ

文字列リテラル、16 進リテラルまたはビット値リテラルには、特定の文字セットおよび照合順序を使用する文字列として指定するために、オプションの文字セットイントロデューサおよび `COLLATE` 句を指定できます:

```
[_charset_name] literal [COLLATE collation_name]
```

`_charset_name` 式は正式にはイントロデューサと呼ばれます。パーサーに「「後続の文字列が文字セット `charset_name` を使用する」」ことを通知します。イントロデューサは、`CONVERT()` のように文字列をイントロデューサ文字セットに変更しません。パディングは発生する可能性がありますが、文字列値は変更されません。イントロデューサは単なる信号です。

文字列リテラルの場合、イントロデューサと文字列の間の空白は許可されますが、オプションです。

文字セットリテラルの場合、イントロデューサは次の文字列の文字セットを示しますが、パーサーが文字列内でエスケープ処理を実行する方法は変更しません。エスケープは常に、`character_set_connection` で指定された文字セットに従ってパーサーが解釈します。その他の説明と例については、[セクション10.3.6「文字列リテラルの文字セットおよび照合順序」](#) を参照してください。

例:

```
SELECT 'abc';
SELECT _latin1'abc';
SELECT _binary'abc';
SELECT _utf8mb4'abc' COLLATE utf8mb4_danish_ci;

SELECT _latin1 X'4D7953514C';
SELECT _utf8mb4 0x4D7953514C COLLATE utf8mb4_danish_ci;

SELECT _latin1 b'1000001';
SELECT _utf8mb4 0b1000001 COLLATE utf8mb4_danish_ci;
```

文字セットイントロデューサと `COLLATE` 句は、標準 SQL 仕様に基づいて実装されています。

文字列リテラルは、`_binary` イントロデューサを使用してバイナリ文字列として指定できます。16 進数リテラルおよびビット値リテラルはデフォルトでバイナリ文字列であるため、`_binary` は許可されますが、通常は不要です。`_binary` は、16 進数リテラルまたはビットリテラルをバイナリ文字列として保持し、それ以外の場合はリテラルを数値として処理する場合に役立ちます。たとえば、ビット操作では、MySQL 8.0 以上で数値またはバイナリ文字列引数が許可されますが、デフォルトでは 16 進数リテラルおよびビットリテラルは数値として扱われます。このようなリテラルのバイナリ文字列コンテキストを明示的に指定するには、少なくとも次のいずれかの引数に `_binary` イントロデューサを使用します:

```
mysql> SET @v1 = X'000D' | X'0BC0';
mysql> SET @v2 = _binary X'000D' | X'0BC0';
mysql> SELECT HEX(@v1), HEX(@v2);
+-----+-----+
| HEX(@v1) | HEX(@v2) |
+-----+-----+
| BCD    | 0BCD    |
+-----+-----+
```

両方のビット操作で表示される結果は似ていますが、`_binary` のない結果は `BIGINT` 値ですが、`_binary` の結果はバイナリ文字列です。結果タイプが異なるため、表示される値は異なります: 数値の結果には、上位 0 桁は表示されません。

MySQL は、文字列リテラル、16 進数リテラルまたはビット値リテラルの文字セットおよび照合順序を次の方法で決定します:

- `_charset_name` と `COLLATE collation_name` の両方が指定されている場合、文字セット `charset_name` と照合順序 `collation_name` が使用されます。`collation_name` は、`charset_name` に対して許可された照合である必要があります。
- `_charset_name` が指定されているが、`COLLATE` が指定されていない場合は、文字セット `charset_name` とそのデフォルトの照合順序が使用されます。各文字セットのデフォルトの照合順序を確認するには、`SHOW CHARACTER SET` ステートメントを使用するか、`INFORMATION_SCHEMA CHARACTER_SETS` テーブルをクエリーします。
- `_charset_name` が指定されていないが、`COLLATE collation_name` が指定されている場合:
 - 文字列リテラルの場合、`character_set_connection` システム変数および照合 `collation_name` によって指定された接続のデフォルト文字セットが使用されます。`collation_name` は、接続のデフォルト文字セットに対して許可された照合順序である必要があります。
 - 16 進数リテラルまたはビット値リテラルの場合、これらのタイプのリテラルはデフォルトでバイナリ文字列であるため、許可される照合は `binary` のみです。
- それ以外の場合 (`_charset_name` も `COLLATE collation_name` も指定されていません):
 - 文字列リテラルの場合は、接続のデフォルトの文字セットと、`character_set_connection` および `collation_connection` システム変数によって指定された照合順序が使用されます。

- 16 進リテラルまたはビット値リテラルの場合、文字セットおよび照合順序は `binary` です。

例:

- `latin1` 文字セットと `latin1_german1_ci` 照合順序を持つ非バイナリ文字列:

```
SELECT _latin1'Müller' COLLATE latin1_german1_ci;
SELECT _latin1 X'0A0D' COLLATE latin1_german1_ci;
SELECT _latin1 b'0110' COLLATE latin1_german1_ci;
```

- `utf8mb4` 文字セットとそのデフォルト照合順序 (つまり、`utf8mb4_0900_ai_ci`) を持つ非バイナリ文字列:

```
SELECT _utf8mb4'Müller';
SELECT _utf8mb4 X'0A0D';
SELECT _utf8mb4 b'0110';
```

- `binary` 文字セットとそのデフォルト照合順序 (`binary`) を持つバイナリ文字列:

```
SELECT _binary'Müller';
SELECT X'0A0D';
SELECT b'0110';
```

16 進リテラルおよびビット値リテラルはデフォルトでバイナリ文字列であるため、導入者は必要ありません。

- 接続のデフォルト文字セットおよび `utf8mb4_general_ci` 照合順序を持つ非バイナリ文字列 (接続文字セットが `utf8mb4` でない場合は失敗します):

```
SELECT 'Müller' COLLATE utf8mb4_general_ci;
```

この構成 (`COLLATE` のみ) は 16 進数リテラルまたはビットリテラルでは機能しません。接続文字セットに関係なく、`binary` は `utf8mb4_general_ci` 照合と互換性がないためです。イントロデューサがない場合に許可される `COLLATE` 句は、`COLLATE binary` のみです。

- 文字列に接続デフォルト文字セットおよび照合順序が指定されている場合

```
SELECT 'Müller';
```

10.3.9 文字セットと照合順序の割り当ての例

MySQL でどのようにしてデフォルトの文字セットおよび照合順序の値が決定されるかを、次の例で示します。

例 1: テーブルおよびカラムの定義

```
CREATE TABLE t1
(
  c1 CHAR(10) CHARACTER SET latin1 COLLATE latin1_german1_ci
) DEFAULT CHARACTER SET latin2 COLLATE latin2_bin;
```

ここでは `latin1` 文字セットと `latin1_german1_ci` 照合順序がカラムに指定されています。この定義は明確であり、簡単明瞭です。なお、`latin1` カラムを `latin2` テーブルに格納することに問題ははありません。

例 2: テーブルおよびカラムの定義

```
CREATE TABLE t1
(
  c1 CHAR(10) CHARACTER SET latin1
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

ここでは、`latin1` 文字セットとデフォルト照合順序がカラムに指定されています。当然のようですが、デフォルトの照合順序はテーブルレベルからは取得されません。`latin1` のデフォルト照合順序は常に `latin1_swedish_ci` なので、カラム `c1` には `latin1_danish_ci` ではなく `latin1_swedish_ci` の照合順序が設定されます。

例 3: テーブルおよびカラムの定義

```
CREATE TABLE t1
(
  c1 CHAR(10)
```

```
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

ここでは、デフォルト文字セットとデフォルト照合順序がカラムに指定されています。この状況では、MySQL はテーブルレベルをチェックして、カラムの文字セットおよび照合順序を特定します。したがって、カラム `c1` の文字セットは `latin1`、照合順序は `latin1_danish_ci` となります。

例 4: データベース、テーブル、およびカラムの定義

```
CREATE DATABASE d1
  DEFAULT CHARACTER SET latin2 COLLATE latin2_czech_ci;
USE d1;
CREATE TABLE t1
(
  c1 CHAR(10)
);
```

文字セットと照合順序を指定せずにカラムを作成します。テーブルレベルの文字セットと照合順序も指定しません。この状況では、MySQL は、データベースレベルをチェックして、テーブル設定を特定します。その後、この設定がカラム設定になります。したがって、カラム `c1` の文字セットは `latin2`、照合順序は `latin2_czech_ci` となります。

10.3.10 ほかの DBMS との互換性

MaxDB との互換性について、次の 2 つのステートメントは同じです。

```
CREATE TABLE t1 (f1 CHAR(N) UNICODE);
CREATE TABLE t1 (f1 CHAR(N) CHARACTER SET ucs2);
```

10.4 接続文字セットおよび照合順序

「connection」は、クライアントプログラムがサーバーに接続したときに、サーバーと対話するセッションを開始するために行われるものです。クライアントは、クエリーなどの SQL ステートメントをセッション接続を介して送信します。サーバーは接続を介して、結果セットやエラーメッセージなどの応答をクライアントに送信します。

- [接続文字セットおよび照合順序システム変数](#)
- [許可されていないクライアント文字セット](#)
- [クライアントプログラム接続文字セット構成](#)
- [接続文字セット構成用の SQL ステートメント](#)
- [接続文字セットのエラー処理](#)

接続文字セットおよび照合順序システム変数

複数の文字セットおよび照合順序のシステム変数は、サーバーとのクライアントの通信に関係しています。これらのいくつかは、これまでのセクションですでに説明されています。

- `character_set_server` および `collation_server` システム変数は、サーバーの文字セットと照合順序を示します。 [セクション 10.3.2 「サーバー文字セットおよび照合順序」](#) を参照してください。
- `character_set_database` および `collation_database` システム変数は、デフォルトデータベースの文字セットおよび照合順序を示します。 [セクション 10.3.3 「データベース文字セットおよび照合順序」](#) を参照してください。

その他の文字セットおよび照合順序システム変数は、クライアントとサーバー間の接続のトラフィックの処理に関わっています。すべてのクライアントには、セッション固有の接続関連の文字セットおよび照合順序システム変数があります。これらのセッションシステム変数値は接続時に初期化されますが、セッション内で変更できます。

クライアント接続の文字セットおよび照合順序処理に関するいくつかの質問には、システム変数の観点から回答できます:

- クライアントから離れるときのステートメントの文字セットは何ですか。

サーバーは、`character_set_client` システム変数値を、クライアントが送信するステートメントの文字セットにします。

注記

一部の文字セットは、クライアントの文字セットとして使用できません。許可されていないクライアント文字セットを参照してください。

- サーバーがステートメントを受信したあと、どの文字セットに変換すべきですか。

これを確認するために、サーバーは `character_set_connection` および `collation_connection` システム変数を使用します:

- サーバーは、クライアントによって送信されたステートメントを `character_set_client` から `character_set_connection` に変換します。例外: `_utf8mb4` や `_latin2` などのイントロデューサを持つ文字列リテラルの場合、イントロデューサは文字セットを決定します。セクション10.3.8「文字セットイントロデューサ」を参照してください。
- `collation_connection` は、リテラル文字列の比較に重要です。カラム値と文字列を比較する場合、`collation_connection` は関係ありません。これは、カラムには照合優先度の高い独自の照合があるためです(セクション10.8.4「式での照合の強制性」を参照)。
- クエリー結果をクライアントに返送する前に、サーバーはどの文字セットに変換する必要がありますか。

`character_set_results` システム変数値は、サーバーがクライアントにクエリー結果を返信するときに使用する文字セットを示します。これには、カラム値、結果メタデータ(カラム名など)、エラーメッセージなどの結果データが含まれます。

結果セットまたはエラーメッセージの変換を実行しないようにサーバーに指示するには、`character_set_results` を `NULL` または `binary` に設定します:

```
SET character_set_results = NULL;  
SET character_set_results = binary;
```

文字セットおよびエラーメッセージの詳細は、セクション10.6「エラーメッセージ文字セット」を参照してください。

現在のセッションに適用される文字セットおよび照合順序システム変数の値を確認するには、次のステートメントを使用します:

```
SELECT * FROM performance_schema.session_variables  
WHERE VARIABLE_NAME IN (  
'character_set_client', 'character_set_connection',  
'character_set_results', 'collation_connection'  
) ORDER BY VARIABLE_NAME;
```

次の単純なステートメントでも接続変数が表示されますが、関連する他の変数も含まれます。これらは、all 文字セットおよび照合順序システム変数の表示に役立ちます:

```
SHOW SESSION VARIABLES LIKE 'character\_set\_%';  
SHOW SESSION VARIABLES LIKE 'collation\_%';
```

クライアントは、これらの変数の設定を微調整することも、デフォルトに従うこともできます(この場合は、このセッションの残りをスキップできます)。デフォルトを使用しない場合、サーバーへの接続ごとに文字設定を変更する必要があります。

許可されていないクライアント文字セット

`character_set_client` システム変数は、特定の文字セットに設定できません:

```
ucs2  
utf16  
utf16le  
utf32
```

これらの文字セットのいずれかをクライアント文字セットとして使用しようとする、エラーが発生します:

```
mysql> SET character_set_client = 'ucs2';
```



```
ERROR 1231 (42000): Variable 'character_set_client'
can't be set to the value of 'ucs2'
```

これらの文字セットのいずれかが次のコンテキストで使用されている場合、同じエラーが発生し、その結果、`character_set_client` に名前付き文字セットを設定しようとしています:

- `mysql` や `mysqladmin` などの MySQL クライアントプログラムで使用される `--default-character-set=charset_name` コマンドオプション。
- `SET NAMES 'charset_name'` ステートメント。
- `SET CHARACTER SET 'charset_name'` ステートメント。

クライアントプログラム接続文字セット構成

クライアントがサーバーに接続すると、サーバーとの通信に使用する文字セットが示されます。(実際には、クライアントはその文字セットのデフォルトの照合順序を示し、サーバーはこの照合順序から文字セットを決定できます。)サーバーは、この情報を使用して `character_set_client`, `character_set_results`, `character_set_connection` システム変数を文字セットに設定し、`collation_connection` を文字セットのデフォルトの照合順序に設定します。実際には、サーバーは `SET NAMES` 操作と同等の処理を実行します。

サーバーが要求された文字セットまたは照合順序をサポートしていない場合は、サーバー文字セットおよび照合順序を使用して接続を構成します。このフォールバック動作の詳細は、[接続文字セットのエラー処理](#) を参照してください。

`mysql`, `mysqladmin`, `mysqlcheck`, `mysqlimport` および `mysqlshow` クライアントプログラムは、使用するデフォルトの文字セットを次のように決定します:

- その他の情報がない場合、各クライアントはコンパイルされたデフォルト文字セット (通常は `utf8mb4`) を使用します。
- 各クライアントは、Unix システムの `LANG` または `LC_ALL` ロケール環境変数の値や Windows システムのコードページ設定など、オペレーティングシステムの設定に基づいて、使用する文字セットを自動検出できます。ロケールが OS から利用できるシステムの場合、クライアントはコンパイル時のデフォルトを使用するのではなく、このロケールを使用してデフォルトの文字セットを設定します。たとえば、`LANG` を `ru_RU.KOI8-R` に設定すると、`koi8r` 文字セットが使用されます。したがってユーザーは、MySQL クライアントが使用できるように、自身の環境内でロケールを構成できます。

OS 文字セットは、正確に一致するものがない場合は、もっとも近い MySQL 文字セットにマップされます。一致した文字セットをサポートしていない場合、クライアントはコンパイル時のデフォルトを使用します。たとえば、`utf8` および `utf-8` は `utf8mb4` にマップされ、`ucs2` は接続文字セットとしてサポートされていないため、コンパイル済みのデフォルトにマップされます。

C アプリケーションは、サーバーに接続する前に次のように `mysql_options()` を呼び出すことによって、OS 設定に基づいて文字セットの自動検出を使用できます。

```
mysql_options(mysql,
    MYSQL_SET_CHARSET_NAME,
    MYSQL_AUTODETECT_CHARSET_NAME);
```

- 各クライアントは `--default-character-set` オプションをサポートしているため、ユーザーは文字セットを明示的に指定して、それ以外の場合にクライアントが決定するデフォルトをオーバーライドできます。

注記

一部の文字セットは、クライアントの文字セットとして使用できません。`--default-character-set` でこれらを使用しようとすると、エラーが発生します。[許可されていないクライアント文字セット](#) を参照してください。

`mysql` クライアントでは、デフォルトとは異なる文字セットを使用するために、サーバーに接続するたびに `SET NAMES` ステートメントを明示的に実行できます ([クライアントプログラム接続文字セット構成](#) を参照)。同じ結果をより簡単に得るには、オプションファイルに文字セットを指定します。たとえば、次のオプションファイル設定では、`mysql` を起動するたびに、接続関連の 3 つの文字セットシステム変数が `koi8r` に設定されます:

```
[mysql]  
default-character-set=koi8r
```

自動再接続を有効にして `mysql` クライアントを使用している場合は (推奨しません)、`SET NAMES` ではなく `charset` コマンドを使用することをお勧めします。例:

```
mysql> charset koi8r  
Charset changed
```

`charset` コマンドは、`SET NAMES` ステートメントを発行し、接続の切断後に再接続するときに `mysql` が使用するデフォルトの文字セットも変更します。

クライアントプログラムを構成する場合は、クライアントプログラムが実行される環境も考慮する必要があります。[セクション10.5「アプリケーションの文字セットおよび照合順序の構成」](#)を参照してください。

接続文字セット構成用の SQL ステートメント

接続が確立されると、クライアントは現在のセッションの文字セットおよび照合順序システム変数を変更できます。これらの変数は `SET` ステートメントを使用して個別に変更できますが、接続関連の文字セットシステム変数にグループとして影響する便利なステートメントは 2 つあります:

- `SET NAMES 'charset_name' [COLLATE 'collation_name']`

`SET NAMES` は、クライアントが SQL ステートメントをサーバーに送信するために使用する文字セットを示します。したがって、`SET NAMES 'cp1251'` は、「このクライアントから今後受信するメッセージが文字セット `cp1251` で送信される」ことを、サーバーに知らせます。また、クライアントに結果を返信するときにサーバーが使用する文字セットも指定します。(たとえば、結果セットを生成する `SELECT` ステートメントを使用する場合、どの文字セットをカラム値に使用するかを示します。)

`SET NAMES 'charset_name'` ステートメントは次の 3 つのステートメントと同等です。

```
SET character_set_client = charset_name;  
SET character_set_results = charset_name;  
SET character_set_connection = charset_name;
```

`character_set_connection` を `charset_name` に設定すると、`collation_connection` も暗黙的に `charset_name` のデフォルト照合順序に設定されます。この照合順序を明示的に設定する必要はありません。`collation_connection` に使用する特定の照合を指定するには、`COLLATE` 句を追加します:

```
SET NAMES 'charset_name' COLLATE 'collation_name'
```

- `SET CHARACTER SET 'charset_name'`

`SET CHARACTER SET` は `SET NAMES` と似ていますが、`character_set_connection` および `collation_connection` を `character_set_database` および `collation_database` に設定します (前述のように、デフォルトデータベースの文字セットおよび照合順序を示します)。

`SET CHARACTER SET charset_name` ステートメントは次の 3 つのステートメントと同等です。

```
SET character_set_client = charset_name;  
SET character_set_results = charset_name;  
SET collation_connection = @@collation_database;
```

`collation_connection` を設定すると、`character_set_connection` も、関連付けられた文字セットに暗黙的に設定されます (`SET character_set_connection = @@character_set_database` の実行と同等です)。`character_set_connection` を明示的に設定する必要はありません。

注記

一部の文字セットは、クライアントの文字セットとして使用できません。`SET NAMES` または `SET CHARACTER SET` でこれらを使用しようとすると、エラーが発生します。[許可されていないクライアント文字セット](#)を参照してください。

例: `column1` が `CHAR(5) CHARACTER SET latin2` として定義されているとします。`SET NAMES` または `SET CHARACTER SET` を指定しない場合、`SELECT column1 FROM t` に対して、サーバーは、接続時にクライアントが

指定した文字セットを使用して、`column1` のすべての値を送り返します。一方、`SELECT` ステートメントを発行する前に `SET NAMES 'latin1'` または `SET CHARACTER SET 'latin1'` と言うと、サーバーは結果を返送する直前に `latin2` 値を `latin1` に変換します。両方の文字セットにない文字の変換は失われる可能性があります。

接続文字セットのエラー処理

不適切な接続文字セットまたは照合順序を使用しようとすると、エラーが発生するか、サーバーが特定の接続のデフォルトの文字セットおよび照合順序にフォールバックする可能性があります。このセクションでは、接続文字セットの構成時に発生する可能性のある問題について説明します。これらの問題は、接続を確立するとき、または確立された接続内の文字セットを変更するときが発生することがあります。

- [接続時のエラー処理](#)
- [ランタイムエラー処理](#)

接続時のエラー処理

一部の文字セットは、クライアント文字セットとして使用できません。[許可されていないクライアント文字セット](#) を参照してください。有効だがクライアント文字セットとして許可されていない文字セットを指定すると、サーバーはエラーを返します:

```
shell> mysql --default-character-set=ucs2
ERROR 1231 (42000): Variable 'character_set_client' can't be set to
the value of 'ucs2'
```

クライアントが認識しない文字セットを指定すると、エラーが発生します:

```
shell> mysql --default-character-set=bogus
mysql: Character set 'bogus' is not a compiled character set and is
not specified in the '/usr/local/mysql/share/charsets/Index.xml' file
ERROR 2019 (HY000): Can't initialize character set bogus
(path: /usr/local/mysql/share/charsets/)
```

クライアントが認識するがサーバーが認識しない文字セットを指定した場合、サーバーはデフォルトの文字セットおよび照合順序にフォールバックします。サーバーが `latin1` および `latin1_swedish_ci` をデフォルトとして使用するよう構成されており、`gb18030` が有効な文字セットとして認識されないとします。`--default-character-set=gb18030` を指定するクライアントはサーバーに接続できますが、結果の文字セットはクライアントが希望する文字セットではありません:

```
mysql> SHOW SESSION VARIABLES LIKE 'character\_set\_%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| character_set_client | latin1 |
| character_set_connection | latin1 |
...
| character_set_results | latin1 |
...
mysql> SHOW SESSION VARIABLES LIKE 'collation\_connection';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| collation_connection | latin1_swedish_ci |
+-----+-----+
```

接続システム変数が、`latin1` および `latin1_swedish_ci` の文字セットおよび照合順序を反映するように設定されていることがわかります。これは、サーバーがクライアント文字セットリクエストを満たすことができず、デフォルトに戻すために発生します。

この場合、サーバーがサポートしていないため、クライアントは必要な文字セットを使用できません。クライアントは、別の文字セットを使用するか、目的の文字セットをサポートする別のサーバーに接続する必要があります。

同じ問題がより微妙なコンテキストで発生: クライアントがサーバーが認識する文字セットを使用するようにサーバーに指示したが、クライアント側のその文字セットのデフォルトの照合順序がサーバー側で認識されない場合。これは、たとえば、MySQL 8.0 クライアントが `utf8mb4` をクライアント文字セットとして使用して MySQL 5.7 サーバーに接続する場合に発生します。`--default-character-set=utf8mb4` を指定するクライアントは、サーバーに接続できま

す。ただし、前の例と同様に、サーバーはクライアントが要求した文字セットと照合順序ではなく、デフォルトの文字セットと照合順序にフォールバックします:

```
mysql> SHOW SESSION VARIABLES LIKE 'character_set_%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| character_set_client | latin1 |
| character_set_connection | latin1 |
| ... | ... |
| character_set_results | latin1 |
| ... | ... |
+-----+-----+
mysql> SHOW SESSION VARIABLES LIKE 'collation_connection';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| collation_connection | latin1_swedish_ci |
+-----+-----+
```

これが発生するのはなぜですか。その後、`utf8mb4` は 8.0 クライアントと 5.7 サーバーに認識されるため、両方で認識されます。この動作を理解するには、クライアントが使用する文字セットをサーバーに通知するときに、その文字セットのデフォルト照合順序をサーバーに実際に通知することを理解する必要があります。したがって、前述の動作はフアクタの組合せによって発生します:

- `utf8mb4` のデフォルトの照合は、MySQL 5.7 と 8.0 (5.7 の場合は `utf8mb4_general_ci`、8.0 の場合は `utf8mb4_0900_ai_ci`) で異なります。
- 8.0 クライアントが `utf8mb4` の文字セットをリクエストすると、サーバーに送信されるものがデフォルトの 8.0 `utf8mb4` 照合 (`utf8mb4_0900_ai_ci`) になります。
- `utf8mb4_0900_ai_ci` は MySQL 8.0 の時点でのみ実装されるため、5.7 サーバーでは認識されません。
- 5.7 サーバーは `utf8mb4_0900_ai_ci` を認識しないため、クライアント文字セットリクエストを満たすことはできず、デフォルトの文字セットおよび照合 (`latin1` および `latin1_swedish_ci`) にフォールバックします。

この場合でも、クライアントは接続後に `SET NAMES 'utf8mb4'` ステートメントを発行することで `utf8mb4` を使用できます。結果の照合は、5.7 のデフォルトの `utf8mb4` 照合 (`utf8mb4_general_ci`) です。クライアントがさらに `utf8mb4_0900_ai_ci` の照合を必要とする場合、サーバーはその照合を認識しないため、これを実現できません。クライアントは、別の `utf8mb4` 照合を使用するか、MySQL 8.0 以上からサーバーに接続する必要があります。

ランタイムエラー処理

確立された接続内で、クライアントは `SET NAMES` または `SET CHARACTER SET` との接続文字セットおよび照合順序の変更をリクエストできます。

一部の文字セットは、クライアント文字セットとして使用できません。[許可されていないクライアント文字セット](#) を参照してください。有効だがクライアント文字セットとして許可されていない文字セットを指定すると、サーバーはエラーを返します:

```
mysql> SET NAMES 'ucs2';
ERROR 1231 (42000): Variable 'character_set_client' can't be set to the value of 'ucs2'
```

サーバーが文字セット (または照合順序) を認識しない場合、エラーが発生します:

```
mysql> SET NAMES 'bogus';
ERROR 1115 (42000): Unknown character set: 'bogus'

mysql> SET NAMES 'utf8mb4' COLLATE 'bogus';
ERROR 1273 (HY000): Unknown collation: 'bogus'
```

ヒント

要求された文字セットがサーバーによって適用されたかどうかを確認するクライアントは、接続して結果が予想される文字セットであることを確認したあとに、次のステートメントを実行できます:

```
SELECT @@character_set_client;
```

10.5 アプリケーションの文字セットおよび照合順序の構成

デフォルトの MySQL 文字セットおよび照合順序 (`utf8mb4`、`utf8mb4_0900_ai_ci`) を使用してデータを格納するアプリケーションの場合、特別な構成は必要ありません。別の文字セットまたは照合順序を使用したデータストレージが必要なアプリケーションの場合は、次の複数の方法で、文字セット情報を構成できます。

- データベースごとに文字設定を指定します。たとえば、あるデータベースを使用するアプリケーションではデフォルトの `utf8mb4` を使用し、別のデータベースを使用するアプリケーションでは `sjis` を使用できます。
- サーバーの起動時に文字設定を指定します。これにより、サーバーは、ほかの調整を行わないすべてのアプリケーションに、所定の設定を使用します。
- ソースから MySQL を構築する場合は、構成時に文字設定を指定します。これにより、サーバーの起動時に指定しなくても、サーバーは指定された設定をすべてのアプリケーションのデフォルトとして使用します。

異なるアプリケーションで別々の文字設定が必要な場合は、データベースごとの手法で十分対応できます。ほとんどまたはすべてのアプリケーションで同じ文字セットを使用する場合は、サーバーの起動時または構成時に文字設定を指定する方法がもっとも便利です。

データベースごとの手法またはサーバー起動の手法では、設定によって、データストレージの文字セットが制御されます。アプリケーションはまた、次の手順で説明するように、クライアントとサーバー間の通信に使用する文字セットをサーバーに知らせる必要もあります。

ここに示す例では、`utf8mb4` および `utf8mb4_0900_ai_ci` のデフォルトの代替として、特定のコンテキストで `latin1` 文字セットおよび `latin1_swedish_ci` 照合順序を使用することを前提としています。

- データベースごとに文字設定を指定します。テーブルが特定のデフォルトの文字セットおよび照合順序をデータ記憶域に使用するようにデータベースを作成するには、次のような `CREATE DATABASE` ステートメントを使用します:

```
CREATE DATABASE mydb
CHARACTER SET latin1
COLLATE latin1_swedish_ci;
```

データベースで作成されたテーブルでは、文字カラムに `latin1` および `latin1_swedish_ci` がデフォルトで使用されません。

データベースを使用するアプリケーションは、接続するたびに、サーバーへの接続を構成する必要もあります。これを行うには、接続後に `SET NAMES 'latin1'` ステートメントを実行します。このステートメントは、接続方法 (`mysql` クライアント、PHP スクリプトなど) に関係なく使用できます。

場合によっては、必要な文字セットを別の方法で使用するように接続を構成できます。たとえば、`mysql` を使用して接続するには、`--default-character-set=latin1` コマンドラインオプションを指定して、`SET NAMES 'latin1'` と同じ効果を得ることができます。

クライアント接続の構成の詳細は、[セクション10.4「接続文字セットおよび照合順序」](#)を参照してください。

注記

`ALTER DATABASE` を使用してデータベースのデフォルトの文字セットまたは照合順序を変更する場合は、それらのデフォルトを使用するデータベース内の既存のストアドルーチンを削除して再作成し、新しいデフォルト値を使用する必要があります。(ストアドルーチンでは、文字セットまたは照合順序が明示的に指定されていない場合、文字データ型を伴う変数は、データベースのデフォルトを使用します。[セクション13.1.17「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」](#)を参照してください。)

- サーバーの起動時に文字設定を指定します。サーバーの起動時に文字セットおよび照合順序を選択するには、`--character-set-server` および `--collation-server` オプションを使用します。たとえば、オプションファイルでオプションを指定するには、次の行を含めます。


```
[mysqld]
character-set-server=latin1
collation-server=latin1_swedish_ci
```

これらの設定は、サーバー全体に適用され、任意のアプリケーションによって作成されたデータベースのデフォルトおよびこれらのデータベース内に作成されたテーブルのデフォルトとして適用されます。

前述のように、アプリケーションは引き続き、接続したあとで `SET NAMES` または同等のステートメントを使用して、その接続を構成する必要があります。接続するクライアントごとに `SET NAMES` が自動的に実行されるように、`--init_connect="SET NAMES 'latin1'"` オプションを使用してサーバーを起動することが一時的な場合があります。ただし、`CONNECTION_ADMIN` 権限 (または非推奨の `SUPER` 権限) を持つユーザーに対して `init_connect` 値が実行されないため、これにより一貫性のない結果が生じる可能性があります。

- MySQL の構成時に文字設定を指定します。ソースから MySQL を構成および構築する場合に文字セットおよび照合順序を選択するには、`DEFAULT_CHARSET` および `DEFAULT_COLLATION CMake` オプションを使用します:

```
cmake . -DDEFAULT_CHARSET=latin1 \
-DDEFAULT_COLLATION=latin1_swedish_ci
```

結果のサーバーでは、データベースおよびテーブルのデフォルトおよびクライアント接続として `latin1` および `latin1_swedish_ci` が使用されます。`--character-set-server` と `--collation-server` を使用して、サーバー起動時にこれらのデフォルトを指定する必要はありません。また、アプリケーションがサーバーに接続したあとで `SET NAMES` または同等のステートメントを使用して、その接続を構成する必要もありません。

アプリケーション用に MySQL 文字セットを構成する方法とは無関係に、これらのアプリケーションが実行する環境も考慮する必要があります。たとえば、エディタで作成したファイルから取得した UTF-8 テキストを使用してステートメントを送信する場合は、ファイルエンコーディングが正しく、オペレーティングシステムで正しく処理されるように、環境のロケールを UTF-8 に設定してファイルを編集する必要があります。端末ウィンドウ内から `mysql` クライアントを使用する場合、UTF-8 を使用するようにこのウィンドウを構成する必要があります。そうしないと、文字が正しく表示されない可能性があります。Web 環境で実行するスクリプトの場合、このスクリプトは、MySQL Server とやり取りできるように文字のエンコーディングを正しく処理する必要があり、ページ内容の表示方法をブラウザが認識できるようにエンコーディングを正しく指定したページを生成する必要があります。たとえば、次の `<meta>` タグを `<head>` 要素内に含められます。

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

10.6 エラーメッセージ文字セット

このセクションでは、MySQL サーバーが文字セットを使用してエラーメッセージを作成する方法について説明します。エラーメッセージの (文字セットではなく) 言語の詳細は、[セクション10.12「エラーメッセージ言語の設定」](#)を参照してください。エラーロギングの構成に関する一般情報は、[セクション5.4.2「エラーログ」](#)を参照してください。

- [エラーメッセージ構成の文字セット](#)
- [エラーメッセージ処理の文字セット](#)

エラーメッセージ構成の文字セット

サーバーは次のようにエラーメッセージを構築します。

- メッセージテンプレートでは、UTF-8 (`utf8mb3`) が使用されます。
- メッセージテンプレートのパラメータが、特定のエラーの発生に適用される値に置き換えられます。
- テーブル名やカラム名などの識別子は、UTF-8 を内部で使用するので、そのままコピーされます。
- 文字 (非バイナリ) 文字列値は、その文字セットから UTF-8 に変換されます。
- バイナリ文字列値は、`0x20` から `0x7E` までの範囲のバイトの場合と同様にコピーされ、その範囲外のバイトの場合は `\x` 16 進数エンコーディングが使用されます。たとえば、`0x41CF9F` を `VARBINARY` の一意のカラムに挿入しようとしたときに重複キーエラーが発生した場合、生成されるエラーメッセージでは、16 進数でエンコードされた UTF-8 が使用されます:


```
Duplicate entry 'A\xC3\x9F' for key 1
```

エラーメッセージ処理の文字セット

構成後のエラーメッセージは、サーバーによってエラーログに書き込まれるか、クライアントに送信されます:

- サーバーは、エラーメッセージをエラーログに書き込むと、別の文字セットに変換せずに、構築されたとおりに UTF-8 に書き込みます。
- サーバーがエラーメッセージをクライアントプログラムに送信すると、サーバーはそれを UTF-8 から `character_set_results` システム変数で指定された文字セットに変換します。 `character_set_results` に `NULL` または `binary` の値がある場合、変換は行われません。変数値が `utf8mb3` または `utf8mb4` の場合、これらの文字セットには、メッセージ構成で使用されるすべての UTF-8 文字を含むレパートリーがあるため、変換は行われません。

`character_set_results` で文字を表現できない場合、変換中に一部のエンコーディングが発生する可能性があります。エンコーディングは、Unicode コードポイント値を使用します。

- Basic Multilingual Plane (BMP) 範囲 (`0x0000` から `0xFFFF`) 内の文字は、`\nnnn` 表記を使用して書き込まれます。
- BMP 範囲外の文字 (`0x10000` から `0x10FFFF`) は、`\+nnnnnn` 表記を使用して書き込まれます。

クライアントは、`character_set_results` を設定して、エラーメッセージを受信するときの文字セットを制御できます。この変数は直接設定することも、`SET NAMES` などの手段で間接的に設定することもできます。`character_set_results` の詳細は、[セクション10.4「接続文字セットおよび照合順序」](#)を参照してください。

10.7 カラム文字セットの変換

特定の文字セットを使用するようにバイナリ文字列または非バイナリ文字列カラムを変換するには、`ALTER TABLE` を使用します。正しく変換が行われるには、次の条件のいずれかを適用する必要があります。

- カラムにバイナリデータ型 (`BINARY`、`VARBINARY`、`BLOB`) がある場合、含まれるすべての値は、単一の文字セット (カラムの変換後の文字セット) を使用してエンコードされる必要があります。バイナリカラムを使用して複数の文字セットで情報を格納する場合、MySQL はどの値がどの文字セットを使用するかを認識できず、データを正確に変換できません。
- カラムに非バイナリデータ型 (`CHAR`、`VARCHAR`、`TEXT`) がある場合、その内容は、カラムの文字セットでエンコードする必要があり、ほかの文字セットは使用できません。内容が別の文字セットでエンコードされている場合、最初にバイナリデータ型を使用するようにカラムを変換してから、使用する文字セットで非バイナリカラムに変換できます。

`VARBINARY(50)` として定義された `col1` という名前のテーブル `t` にバイナリカラムがあるとして、カラム内の情報が、単一の文字セットを使用してエンコードされているとすると、このカラムを、その文字セットを含む非バイナリカラムに変換できます。たとえば、`col1` に `greek` 文字セットの文字を表すバイナリデータが含まれる場合、次のように変換できます。

```
ALTER TABLE t MODIFY col1 VARCHAR(50) CHARACTER SET greek;
```

元のカラムに `BINARY(50)` のタイプがある場合は、それを `CHAR(50)` に変換できますが、結果の値の末尾に `0x00` バイトが埋め込まれるため、望ましくない場合があります。これらのバイトを削除するには、`TRIM()` 関数を使用します。

```
UPDATE t SET col1 = TRIM(TRAILING 0x00 FROM col1);
```

テーブル `t` に `CHAR(50) CHARACTER SET latin1` として定義された `col1` という名前の非バイナリ列があるが、`utf8` を使用するようにこれを変換し、多くの言語の値を格納できるようにするとします。次のステートメントでこれを行います。

```
ALTER TABLE t MODIFY col1 CHAR(50) CHARACTER SET utf8;
```

両方の文字セットにない文字がカラムに含まれている場合、変換の損失が大きくなることがあります。

MySQL 4.1 より前の古いテーブルがある場合、実際にはサーバーのデフォルトの文字セットと異なる文字セットでエンコードされた値が、非バイナリカラムに含まれるという特殊な状況が起こります。たとえば、アプリケーションで

は、MySQL のデフォルトの文字セットが異なる場合でも、カラムに `sjis` 値が格納されていることがあります。適切な文字セットを使用するために、カラムを変換することは可能ですが、追加ステップが必要になります。たとえば、サーバーのデフォルト文字セットが `latin1` で、`col1` は `CHAR(50)` と定義されているにもかかわらず、内容は `sjis` 値であるとします。最初のステップでは、バイナリデータ型にカラムを変換することで、文字変換を実行しないで既存の文字セット情報を取り除きます。

```
ALTER TABLE t MODIFY col1 BLOB;
```

次のステップでは、適切な文字セットを使用して、非バイナリデータ型にカラムを変換します。

```
ALTER TABLE t MODIFY col1 CHAR(50) CHARACTER SET sjis;
```

この手順では、MySQL 4.1 以上へのアップグレード後に、`INSERT` や `UPDATE` などのステートメントを使用してテーブルがまだ変更されていないことが必要です。その場合、MySQL は `latin1` を使用してカラムに新しい値を格納し、カラムには `sjis` と `latin1` の値が混在しているため、適切に変換できません。

最初にカラムを作成するときに属性を指定した場合、`ALTER TABLE` を使用してテーブルを変更しているときにも属性を指定する必要があります。たとえば、`NOT NULL` と明示的な `DEFAULT` 値を指定した場合、`ALTER TABLE` ステートメントでも指定する必要があります。それ以外の場合、結果のカラム定義にはこれらの属性は含まれません。

テーブル内のすべての文字カラムを変換するには、`ALTER TABLE ... CONVERT TO CHARACTER SET charset` ステートメントが役立つことがあります。 [セクション13.1.9「ALTER TABLE ステートメント」](#) を参照してください。

10.8 照合順序の問題

以降のセクションでは、文字セットの照合順序のさまざまな側面について説明します。

10.8.1 SQL ステートメントでの COLLATE の使用

`COLLATE` 句では、比較に対するデフォルト照合順序が何であれ、オーバーライドできます。SQL ステートメントのさまざまな個所で `COLLATE` を使用できます。次にいくつかの例を示します。

- `ORDER BY` を指定した場合

```
SELECT k
FROM t1
ORDER BY k COLLATE latin1_german2_ci;
```

- `AS` を指定した場合

```
SELECT k COLLATE latin1_german2_ci AS k1
FROM t1
ORDER BY k1;
```

- `GROUP BY` を指定した場合

```
SELECT k
FROM t1
GROUP BY k COLLATE latin1_german2_ci;
```

- 集計関数を指定した場合

```
SELECT MAX(k COLLATE latin1_german2_ci)
FROM t1;
```

- `DISTINCT` を指定した場合

```
SELECT DISTINCT k COLLATE latin1_german2_ci
FROM t1;
```

- `WHERE` を指定した場合

```
SELECT *
FROM t1
WHERE _latin1 'Müller' COLLATE latin1_german2_ci = k;
```

```
SELECT *
```

```
FROM t1
WHERE k LIKE '_latin1 'Müller' COLLATE latin1_german2_ci;
```

- **HAVING** を指定した場合

```
SELECT k
FROM t1
GROUP BY k
HAVING k = '_latin1 'Müller' COLLATE latin1_german2_ci;
```

10.8.2 COLLATE 句の優先順位

COLLATE 句は優先順位が高いため (`||` より上位)、次の 2 つの式は同等です。

```
x || y COLLATE z
x || (y COLLATE z)
```

10.8.3 文字セットと照合順序の互換性

各文字セットには 1 つ以上の照合順序があり、各照合順序は 1 つの文字セットにのみ関連付けられています。したがって、次のステートメントではエラーメッセージが表示されます。`latin2_bin` 照合順序は `latin1` 文字セットに対して有効ではないからです。

```
mysql> SELECT _latin1 'x' COLLATE latin2_bin;
ERROR 1253 (42000): COLLATION 'latin2_bin' is not valid
for CHARACTER SET 'latin1'
```

10.8.4 式での照合の強制性

大多数のステートメントでは、MySQL がどの照合順序を使用して比較演算を行うかは明確になっています。たとえば、次の場合、照合がカラム `x` の照合であることを明確にする必要があります：

```
SELECT x FROM T ORDER BY x;
SELECT x FROM T WHERE x = x;
SELECT DISTINCT x FROM T;
```

ただし、複数のオペランドがあると、あいまいさが生じることがあります。たとえば、次のステートメントは、カラム `x` と文字列リテラル `'Y'` の比較を実行します：

```
SELECT x FROM T WHERE x = 'Y';
```

`x` と `'Y'` の照合順序が同じ場合、比較に使用する照合順序のあいまいさはありません。ただし、照合順序が異なる場合、比較で `x` の照合順序または `'Y'` の照合順序を使用する必要がありますか。 `x` と `'Y'` の両方に照合順序がありますが、どちらの照合順序が優先されるのでしょうか。

照合の混在は、比較以外のコンテキストでも発生する可能性があります。たとえば、`CONCAT(x,'Y')` などの複数引数の連結操作では、引数を組み合わせて単一の文字列が生成されます。結果にはどのような照合が必要ですか。

これらのような質問を解決するために、MySQL では、一方のアイテムの照合を他方のアイテムの照合に強制できるかどうかをチェックします。MySQL は次のように強制性値を割り当てます。

- 明示的な **COLLATE** 句の強制性は 0 です (強制可能ではありません)。
- 照合順序の異なる 2 つの文字列を連結すると強制性は 1 になります。
- カラムまたはストアードルーチンのパラメータまたはローカル変数の照合順序の強制性は 2 です。
- 「システム定数」 (`USER()` または `VERSION()` などの関数で返される文字列) の強制性は 3 です。
- リテラルの照合順序の強制性は 4 です。
- 数値または時間値の照合の強制性は 5 です。
- **NULL** または **NULL** から派生した式の強制性は 6 です。

MySQL は、次のルールとともに強制性値を使用して、あいまいさを解決します。

- 強制性値がもっとも低い照合順序を使用します。
- 両方の側で強制性が同じ場合は、次のようになります。
 - 両方の側が Unicode であるか、両方の側が Unicode ではない場合は、エラーになります。
- どちらかの側に Unicode 文字セットがあり、もう一方の側に Unicode 以外の文字セットがある場合、Unicode 文字セットの側が優先され、Unicode 以外の側に自動文字セット変換が適用されます。たとえば、次のステートメントはエラーを返しません。

```
SELECT CONCAT(utf8mb4_column, latin1_column) FROM t1;
```

utf8mb4 の文字セットと utf8mb4_column と同じ照合順序を持つ結果を返します。latin1_column の値は、連結する前に自動的に utf8mb4 に変換されます。

- 同じ文字セットのオペランドだが、_bin 照合順序と_ci または _cs 照合順序が混在したオペランドを使用した演算の場合、_bin 照合順序が使用されます。これは、非バイナリ文字列とバイナリ文字列を混在させる操作によってオペランドがバイナリ文字列として評価され、データ型ではなく照合順序に適用される方法と似ています。

自動変換は SQL 標準には含まれていませんが、標準では、すべての文字セットが Unicode の「サブセット」であることが示されています。「スーパーセットに適用されるものはサブセットにも適用される」というよく知られた原則があるので、Unicode の照合順序は Unicode 以外の文字列との比較にも適用できると考えられます。より一般的に、MySQL では文字セットレパートリー概念が使用されます。この概念は、文字セット間のサブセット関係を判断し、それ以外の場合はエラーが発生する操作でオペランドの変換を可能にするために使用されることがあります。[セクション10.2.1「文字セットレパートリー」](#)を参照してください。

次のテーブルに、前述のルールの適用例を示します。

比較	使用される照合順序
column1 = 'A'	column1 の照合順序を使用します
column1 = 'A' COLLATE x	'A' COLLATE x の照合順序を使用します
column1 COLLATE x = 'A' COLLATE y	エラー

文字列式の強制性を判断するには、`COERCIBILITY()` 関数を使用します ([セクション12.16「情報関数」](#)を参照)：

```
mysql> SELECT COERCIBILITY(_utf8'A' COLLATE utf8_bin);
-> 0
mysql> SELECT COERCIBILITY(VERSION());
-> 3
mysql> SELECT COERCIBILITY('A');
-> 4
mysql> SELECT COERCIBILITY(1000);
-> 5
mysql> SELECT COERCIBILITY(NULL);
-> 6
```

式 `CONCAT(1, 'abc')` での引数 1 に対して行われる変換など、数値または時間値の文字列への暗黙の変換の場合、その結果は、`character_set_connection` および `collation_connection` システム変数で指定された文字セットおよび照合順序を持つ文字 (非バイナリ) 列になります。[セクション12.3「式評価での型変換」](#)を参照してください。

10.8.5 バイナリ照合順序と_bin 照合順序

このセクションでは、バイナリ文字列の `binary` 照合順序を非バイナリ文字列の `_bin` 照合順序と比較する方法について説明します。

バイナリ文字列 (`BINARY`、`VARBINARY` および `BLOB` データ型を使用して格納される) には、`binary` という名前の文字セットと照合順序があります。バイナリ文字列はバイトのシーケンスであり、それらのバイトの数値によって比較およびソート順序が決まります。[セクション10.10.8「バイナリ文字セット」](#)を参照してください。

(`CHAR`、`VARCHAR` および `TEXT` データ型を使用して格納された) 非バイナリ文字列には、`binary` 以外の文字セットおよび照合順序があります。特定の非バイナリ文字セットには複数の照合順序を含めることができ、それぞれがセット内の文字の特定の比較順序およびソート順序を定義します。ほとんどの文字セットでは、これらのいずれかがバイナリ照合順序で、照合順序名に `_bin` 接尾辞が付いています。たとえば、`utf8` および `latin1` のバイナリ照合

は、`utf8_bin` および `latin1_bin` という名前です。`utf8mb4` は、`utf8mb4_bin` と `utf8mb4_0900_bin` の 2 つのバイナリ照合順序を持つ例外です。[セクション10.10.1「Unicode 文字セット」](#) を参照してください。

`binary` 照合は、次の各セクションで説明するように、いくつかの点で `_bin` 照合と異なります:

- [比較およびソートの単位](#)
- [文字セット変換](#)
- [大文字小文字変換](#)
- [比較での後続領域の処理](#)
- [挿入および取得のための後続領域処理](#)

比較およびソートの単位

バイナリ文字列は、バイトのシーケンスです。`binary` 照合の場合、比較およびソートは数値バイト値に基づきます。非バイナリ文字列は文字のシーケンスであり、これはマルチバイトであることもあります。非バイナリ文字列の照合順序は、比較およびソートのための文字値の順序を定義します。`_bin` 照合順序では、この順序付けは数値文字コード値に基づきます。これは、文字コード値がマルチバイトであることを除き、バイナリ文字列の順序付けに似ています。

文字セット変換

非バイナリ文字列には文字セットがあり、多くの場合、文字列に `_bin` 照合順序がある場合でも自動的に別の文字セットに変換されます:

- 異なる文字セットを持つ別のコラムにコラム値を割り当てる場合:

```
UPDATE t1 SET utf8_bin_column=latin1_column;
INSERT INTO t1 (latin1_column) SELECT utf8_bin_column FROM t2;
```

- 文字列リテラルを使用して、`INSERT` または `UPDATE` のコラム値を割り当てる場合:

```
SET NAMES latin1;
INSERT INTO t1 (utf8_bin_column) VALUES ('string-in-latin1');
```

- サーバーからクライアントに結果を送信する場合:

```
SET NAMES latin1;
SELECT utf8_bin_column FROM t2;
```

バイナリ文字列コラムの場合、変換は行われません。前述のような場合は、文字列値がバイト単位でコピーされます。

大文字小文字変換

非バイナリ文字セットの照合順序は、文字の大文字と小文字に関する情報を提供するため、順序付けで大文字と小文字を無視する `_bin` 照合順序の場合でも、非バイナリ文字列の文字のある大文字から別の文字に変換できます:

```
mysql> SET NAMES utf8mb4 COLLATE utf8mb4_bin;
mysql> SELECT LOWER('aA'), UPPER('zZ');
+-----+-----+
| LOWER('aA') | UPPER('zZ') |
+-----+-----+
| aa          | ZZ          |
+-----+-----+
```

大文字と小文字という概念は、バイナリ文字列内のバイトには適用されません。大文字と小文字の変換を実行するには、まず文字列に格納されているデータに適した文字セットを使用して、文字列を非バイナリ文字列に変換する必要があります:

```
mysql> SET NAMES binary;
mysql> SELECT LOWER('aA'), LOWER(CONVERT('aA' USING utf8mb4));
+-----+-----+
| LOWER('aA') | LOWER(CONVERT('aA' USING utf8mb4)) |
```

```
+-----+
| aA   | aa   |
+-----+
```

比較での後続領域の処理

MySQL 照合には、**PAD SPACE** または **NO PAD** の値を持つ `pad` 属性があります:

- ほとんどの MySQL 照合には、**PAD SPACE** のパッド属性があります。
- UCA 9.0.0 以上に基づく Unicode 照合には、**NO PAD** のパッド属性があります。[セクション10.10.1「Unicode 文字セット」](#)を参照してください。

非バイナリ文字列 (**CHAR**、**VARCHAR** および **TEXT** 値) の場合、文字列照合パッド属性によって、文字列の末尾にある末尾の空白の比較での処理が決まります:

- PAD SPACE** 照合の場合、末尾のスペースは比較では重要ではありません。文字列は末尾のスペースに関係なく比較されます。
- NO PAD** 照合順序では、他の文字と同様に、比較で末尾のスペースが重要として扱われます。

異なる動作を示すには、いずれか一方が **PAD SPACE** で、もう一方が **NO PAD** の `utf8mb4` バイナリ照合を使用します。この例では、**INFORMATION_SCHEMA COLLATIONS** テーブルを使用して照合のパッド属性を決定する方法も示します。

```
mysql> SELECT COLLATION_NAME, PAD_ATTRIBUTE
FROM INFORMATION_SCHEMA.COLLATIONS
WHERE COLLATION_NAME LIKE 'utf8mb4%bin';
+-----+
| COLLATION_NAME | PAD_ATTRIBUTE |
+-----+
| utf8mb4_bin   | PAD SPACE    |
| utf8mb4_0900_bin | NO PAD      |
+-----+
mysql> SET NAMES utf8mb4 COLLATE utf8mb4_bin;
mysql> SELECT 'a' = 'a';
+-----+
| 'a' = 'a' |
+-----+
|          1 |
+-----+
mysql> SET NAMES utf8mb4 COLLATE utf8mb4_0900_bin;
mysql> SELECT 'a' = 'a';
+-----+
| 'a' = 'a' |
+-----+
|          0 |
+-----+
```

注記

このコンテキストの「比較」には、**LIKE** パターン一致演算子は含まれていません。この演算子の末尾の空白は、照合に関係なく重要です。

バイナリ文字列 (**BINARY**、**VARBINARY** および **BLOB** の値) の場合、後続の空白を含むすべてのバイトが比較で意味を持ちます:

```
mysql> SET NAMES binary;
mysql> SELECT 'a' = 'a';
+-----+
| 'a' = 'a' |
+-----+
|          0 |
+-----+
```

挿入および取得のための後続領域処理

CHAR(N) カラムには、バイナリ以外の文字列の **N** 文字の長さが格納されます。挿入の場合、**N** 文字より短い値はスペースで拡張されます。取得の場合、末尾の空白は削除されます。

BINARY(N) カラムには、バイナリ文字列の **N** バイト長が格納されます。挿入の場合、**N** バイトより短い値は **0x00** バイトで拡張されます。取得の場合、何も削除されません。宣言された長さの値が常に返されます。

```
mysql> CREATE TABLE t1 (
  a CHAR(10) CHARACTER SET utf8 COLLATE utf8_bin,
  b BINARY(10)
);
mysql> INSERT INTO t1 VALUES ('x','x');
mysql> INSERT INTO t1 VALUES ('x ','x ');
mysql> SELECT a, b, HEX(a), HEX(b) FROM t1;
+-----+-----+-----+-----+
| a | b | HEX(a) | HEX(b) |
+-----+-----+-----+-----+
| x | x | 78     | 78     |
| x | x | 78     | 78     |
+-----+-----+-----+-----+
```

10.8.6 照合順序の効果の例

例 1: ドイツ語のウムラウトのソート

テーブル **T** のカラム **X** に次の **latin1** カラムの値が設定されているとします。

```
Muffler
Müller
MX Systems
MySQL
```

さらに、次のステートメントを使用してカラムの値を取得するとします。

```
SELECT X FROM T ORDER BY X COLLATE collation_name;
```

次の表に、別の照合順序で **ORDER BY** を使用した場合に得られる値の順序を示します。

latin1_swedish_ci	latin1_german1_ci	latin1_german2_ci
Muffler	Muffler	Müller
MX Systems	Müller	Muffler
Müller	MX Systems	MX Systems
MySQL	MySQL	MySQL

この例でソート順の違いを生じさせている文字は、頭に 2 つの点が付いた U (ü) であり、これはドイツ語で「U ウムラウト」と呼ばれているものです。

- 最初のカラムには、スウェーデン語/フィンランド語の照合ルールを使用した **SELECT** の結果が示されています。この照合ルールによると、U ウムラウトは Y とソート順が一致します。
- 2 番目のカラムには、German DIN-1 ルールを使用した **SELECT** の結果が示されています。この照合ルールによると、U ウムラウトは U とソート順が一致します。
- 3 番目のカラムには、German DIN-2 ルールを使用した **SELECT** の結果が示されています。この照合ルールによると、U ウムラウトは UE とソート順が一致します。

例 2: ドイツ語のウムラウトの検索

使用される文字セットと照合順序だけが異なる次の 3 つのテーブルがあるとします。

```
mysql> SET NAMES utf8;
mysql> CREATE TABLE german1 (
  c CHAR(10)
) CHARACTER SET latin1 COLLATE latin1_german1_ci;
mysql> CREATE TABLE german2 (
  c CHAR(10)
) CHARACTER SET latin1 COLLATE latin1_german2_ci;
mysql> CREATE TABLE germanutf8 (
  c CHAR(10)
) CHARACTER SET utf8 COLLATE utf8_unicode_ci;
```

各テーブルには次の 2 つのレコードが含まれます。

```
mysql> INSERT INTO german1 VALUES ('Bar'), ('Bär');
mysql> INSERT INTO german2 VALUES ('Bar'), ('Bär');
mysql> INSERT INTO germanutf8 VALUES ('Bar'), ('Bär');
```

上記の照合順序の 2 つには $A = \text{Ä}$ の等式が含まれ、1 つにはこのような等式はありません (`latin1_german2_ci`)。そのため、比較では次の結果が得られます。

```
mysql> SELECT * FROM german1 WHERE c = 'Bär';
+----+
| c |
+----+
| Bar |
| Bär |
+----+
mysql> SELECT * FROM german2 WHERE c = 'Bär';
+----+
| c |
+----+
| Bär |
+----+
mysql> SELECT * FROM germanutf8 WHERE c = 'Bär';
+----+
| c |
+----+
| Bar |
| Bär |
+----+
```

これはバグではなく、`latin1_german1_ci` と `utf8_unicode_ci` のソートプロパティの結果です (示されたソートは、German DIN 5007 標準に従って行われています)。

10.8.7 INFORMATION_SCHEMA 検索での照合の使用

`INFORMATION_SCHEMA` テーブルの文字列カラムには `utf8_general_ci` の照合順序があり、大/小文字は区別されません。ただし、データベースやテーブルなどのファイルシステムでテーブルされるオブジェクトに対応する値の場合、基礎となるファイルシステムの特徴および `lower_case_table_names` システム変数の設定に応じて、`INFORMATION_SCHEMA` 文字列カラムでの検索では大/小文字が区別される場合と区別されない場合があります。たとえば、ファイルシステムで大文字と小文字が区別される場合、検索では大文字と小文字が区別されることがあります。このセクションでは、この動作と、必要に応じてその動作を変更する方法について説明します。

クエリーで、`test` データベースに対し `SCHEMATA.SCHEMA_NAME` カラムを検索するとします。Linux では、ファイルシステムは大/小文字が区別されるため、`SCHEMATA.SCHEMA_NAME` と `'test'` の比較は一致しますが、`'TEST'` との比較は一致しません:

```
mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
  WHERE SCHEMA_NAME = 'test';
+-----+
| SCHEMA_NAME |
+-----+
| test |
+-----+

mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
  WHERE SCHEMA_NAME = 'TEST';
Empty set (0.00 sec)
```

これらの結果は、`lower_case_table_names` システム変数が 0 に設定されている場合に発生します。`lower_case_table_names` 設定が 1 または 2 の場合、2 番目のクエリーは最初のクエリーと同じ (空でない) 結果を返します。

注記

サーバーの初期化時に使用された設定とは異なる `lower_case_table_names` 設定でサーバーを起動することは禁止されています。

Windows または macOS では、ファイルシステムは大/小文字を区別しないため、比較は `'test'` と `'TEST'` の両方に一致します:

```
mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
        WHERE SCHEMA_NAME = 'test';
+-----+
| SCHEMA_NAME |
+-----+
| test       |
+-----+

mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
        WHERE SCHEMA_NAME = 'TEST';
+-----+
| SCHEMA_NAME |
+-----+
| TEST        |
+-----+
```

このコンテキストでは、`lower_case_table_names` の値に違いはありません。

前述の動作は、ファイルシステムで表されるオブジェクトに対応する値を検索する際に、`INFORMATION_SCHEMA` クエリーに `utf8_general_ci` 照合が使用されないために発生します。

`INFORMATION_SCHEMA` カラムに対する文字列操作の結果が予想と異なる場合、回避策は、明示的な `COLLATE` 句を使用して適切な照合を強制することです (セクション10.8.1「SQL ステートメントでの `COLLATE` の使用」を参照)。たとえば、大文字と小文字を区別しない検索を実行するには、`INFORMATION_SCHEMA` カラム名とともに `COLLATE` を使用します。

```
mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
        WHERE SCHEMA_NAME COLLATE utf8_general_ci = 'test';
+-----+
| SCHEMA_NAME |
+-----+
| test        |
+-----+

mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
        WHERE SCHEMA_NAME COLLATE utf8_general_ci = 'TEST';
+-----+
| SCHEMA_NAME |
+-----+
| test        |
+-----+
```

`UPPER()` または `LOWER()` 関数を使用することもできます。

```
WHERE UPPER(SCHEMA_NAME) = 'TEST'
WHERE LOWER(SCHEMA_NAME) = 'test'
```

大文字と小文字を区別するファイルシステムのプラットフォームでも、大文字と小文字を区別しない比較を実行できますが、前述のように、必ずしも常に正しい処理になるとはかぎりません。このようなプラットフォームでは、大文字と小文字だけが異なる名前の複数のオブジェクトが存在する可能性があります。たとえば、`city`、`CITY`、および `City` という名前のテーブルがすべて同時に存在することが可能です。検索がこのようすべての名前と一致するか、1つのみ一致するかを検討し、それに応じてクエリーを書き込みます。次の比較 (`utf8_bin` を使用) のうち、最初の比較では大/小文字が区別されますが、それ以外の比較では区別されません:

```
WHERE TABLE_NAME COLLATE utf8_bin = 'City'
WHERE TABLE_NAME COLLATE utf8_general_ci = 'city'
WHERE UPPER(TABLE_NAME) = 'CITY'
WHERE LOWER(TABLE_NAME) = 'city'
```

`INFORMATION_SCHEMA` はファイルシステムで表されていない「virtual」データベースであるため、`INFORMATION_SCHEMA` 文字列カラム内で `INFORMATION_SCHEMA` 自体を参照する値を検索すると、`utf8_general_ci` 照合が使用されます。たとえば、`SCHEMATA.SCHEMA_NAME` との比較は、プラットフォームに関係なく、`'information_schema'` または `'INFORMATION_SCHEMA'` に一致します。

```
mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
        WHERE SCHEMA_NAME = 'information_schema';
+-----+
| SCHEMA_NAME |
+-----+
| information_schema |
+-----+
```

```
+-----+
mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
      WHERE SCHEMA_NAME = 'INFORMATION_SCHEMA';
+-----+
| SCHEMA_NAME |
+-----+
| information_schema |
+-----+
```

10.9 Unicode のサポート

Unicode 標準には、Basic Multilingual Plane (BMP) の文字と BMP の外部にある補助文字が含まれています。このセクションでは、MySQL での Unicode のサポートについて説明します。Unicode 規格自体の詳細は、「[Unicode Consortium の web サイト](#)」を参照してください。

BMP 文字には次の 3 つの特性があります。

- コードポイント値は 0 から 65535 (または `U+0000` と `U+FFFF`) の間です。
- これらは、8、16 または 24 ビット (1 から 3 バイト) を使用して可変長エンコーディングでエンコードできます。
- これらは、16 ビット (2 バイト) を使用して固定長エンコーディングでエンコードできます。
- 主要言語のほとんどすべての文字には、これらで十分です。

補助文字は BMP の外部にあります:

- コードポイント値は、`U+10000` と `U+10FFFF` の間です。
- Unicode による補助文字のサポートには、BMP 文字の範囲外の文字セットが必要であるため、BMP 文字 (文字当たり最大 4 バイト) よりも多くの領域が必要です。

Unicode データをエンコードするための UTF-8 (8 ビット単位の Unicode 変換形式) メソッドは、RFC 3629 に従って実装されます。RFC 3629 には、1 バイトから 4 バイトまでのエンコーディング順序が記述されています。UTF-8 の概念は、長さの異なるバイトシーケンスを使用してさまざまな Unicode 文字をエンコードするというものです。

- 基本的なラテン文字、数字、句読点は 1 バイトを使用します。
- 拡張ラテン文字 (チルダ、長音符号、アキュート、グラヴエ、およびほかのアクセント符号)、キリル文字、ギリシャ語、アルメニア語、ヘブライ語、アラビア語、シリア語などのほとんどのヨーロッパおよび中東のスク립ト文字は、2 バイトシーケンスに収まります。
- 韓国語、中国語、および日本語の表意文字は、3 バイトまたは 4 バイトのシーケンスを使用します。

MySQL では、次の Unicode 文字セットがサポートされます:

- `utf8mb4`: Unicode 文字セットの UTF-8 エンコーディング。文字ごとに 1 バイトから 4 バイトを使用します。
- `utf8mb3`: Unicode 文字セットの UTF-8 エンコーディング。文字ごとに 1 バイトから 3 バイトを使用します。
- `utf8`: `utf8mb3` のエイリアス。
- `ucs2`: 文字ごとに 2 バイトを使用した Unicode 文字セットの UCS-2 エンコーディング。
- `utf16`: 文字ごとに 2 バイトまたは 4 バイトを使用する Unicode 文字セットの UTF-16 エンコーディング。`ucs2` と似ていますが、補助文字の拡張子が付いています。
- `utf16le`: Unicode 文字セットの UTF-16LE エンコーディング。`utf16` と似ていますが、ビッグエンディアンではなくリトルエンディアンです。
- `utf32`: 文字ごとに 4 バイトを使用する Unicode 文字セットの UTF-32 エンコーディング。

注記

`utf8mb3` 文字セットは非推奨であり、将来の MySQL リリースで削除される予定です。かわりに `utf8mb4` を使用してください。`utf8` は現在 `utf8mb3` のエイリアスですが、ある時点で

は `utf8` が `utf8mb4` への参照になることが予想されます。 `utf8` の意味があいまいにならないように、`utf8` ではなく文字セット参照に `utf8mb4` を明示的に指定することを検討してください。

表 10.2 「Unicode 文字セットの一般的な特性」には、MySQL でサポートされている Unicode 文字セットの一般的な特性がまとめられています。

表 10.2 Unicode 文字セットの一般的な特性

文字セット	サポートされる文字	文字ごとに必要な記憶域
<code>utf8mb3</code> , <code>utf8</code>	BMP のみ	1 バイト、2 バイトまたは 3 バイト
<code>ucs2</code>	BMP のみ	2 バイト
<code>utf8mb4</code>	BMP および補足	1、2、3 または 4 バイト
<code>utf16</code>	BMP および補足	2 バイトまたは 4 バイト
<code>utf16le</code>	BMP および補足	2 バイトまたは 4 バイト
<code>utf32</code>	BMP および補足	4 バイト

BMP の外部の文字は REPLACEMENT CHARACTER と比較され、BMP 文字 (`utf8mb3` または `ucs2`) のみをサポートする Unicode 文字セットに変換される場合は '?' に変換されます。

補助文字をサポートし、BMP 専用の `utf8mb3` および `ucs2` 文字セットより「広い」である文字セットを使用する場合、アプリケーションに互換性のない問題が発生する可能性があります。[セクション 10.9.8 「3 バイト Unicode 文字セットと 4 バイト Unicode 文字セット間の変換」](#) を参照してください。このセクションでは、テーブルを (3 バイト) `utf8mb3` から (4 バイト) `utf8mb4` に変換する方法と、その実行時に適用される制約についても説明します。

類似した一連の照合順序を、ほとんどの Unicode 文字セットで使用できます。たとえば、それぞれにデンマーク語照合順序があり、その名前は `utf8mb4_danish_ci`, `utf8mb3_danish_ci`, `utf8_danish_ci`, `ucs2_danish_ci`, `utf16_danish_ci` および `utf32_danish_ci` です。例外は `utf16le` であり、2 つの照合順序しかありません。Unicode 照合およびそれらの区別プロパティ (補助文字の照合プロパティを含む) の詳細は、[セクション 10.10.1 「Unicode 文字セット」](#) を参照してください。

UCS-2、UTF-16、および UTF-32 の MySQL での実装は、ビッグエンディアンのバイト順で文字を格納し、値の先頭にバイト順マーク (BOM) を使用しません。ほかのデータベースシステムでは、リトルエンディアンのバイト順または BOM を使用していることもあります。このような場合は、これらのシステムと MySQL 間でデータを転送するときに値の変換を実行する必要があります。UTF-16LE の実装はリトルエンディアンです。

MySQL は UTF-8 値に BOM を使用しません。

Unicode を使用してサーバーと通信するクライアントアプリケーションでは、クライアント文字セットを適宜設定する必要があります (たとえば、`SET NAMES 'utf8mb4'` ステートメントを発行します)。一部の文字セットは、クライアントの文字セットとして使用できません。`SET NAMES` または `SET CHARACTER SET` でこれらを使用しようとすると、エラーが発生します。[許可されていないクライアント文字セット](#) を参照してください。

以降のセクションでは、MySQL における Unicode 文字セットについてさらに詳しく説明します。

10.9.1 utf8mb4 文字セット (4 バイトの UTF-8 Unicode エンコーディング)

`utfmb4` 文字セットには、次の特性があります:

- BMP および補助文字をサポートします。
- マルチバイト文字ごとに最大 4 バイトが必要です。

`utf8mb4` は、BMP 文字のみをサポートし、文字当たり最大 3 バイトを使用する `utf8mb3` 文字セットと対比しています:

- BMP 文字の場合、`utf8mb4` と `utf8mb3` のストレージ特性は同じです: 同じコード値、同じエンコーディング、同じ長さ。

- 補助文字の場合、`utf8mb4` ではそれを格納するために 4 バイトが必要ですが、`utf8mb3` では文字を格納できません。`utf8mb3` カラムを `utf8mb4` に変換する場合、補助文字がないため、変換の心配は必要ありません。

`utf8mb4` は `utf8mb3` のスーパーセットであるため、次の連結などの操作の場合、結果には文字セット `utf8mb4` と `utf8mb4_col` の照合順序が含まれます:

```
SELECT CONCAT(utf8mb3_col, utf8mb4_col);
```

同様に、次の `WHERE` 句内の比較は、`utf8mb4_col` の照合順序に従って行われます。

```
SELECT * FROM utf8mb3_tbl, utf8mb4_tbl  
WHERE utf8mb3_tbl.utf8mb3_col = utf8mb4_tbl.utf8mb4_col;
```

マルチバイト文字セットに関連するデータ型の記憶域の詳細は、[文字列型の格納要件](#) を参照してください。

10.9.2 utf8mb3 文字セット (3 バイトの UTF-8 Unicode エンコーディング)

`utf8mb3` 文字セットには、次の特性があります:

- BMP 文字のみをサポート (補助文字はサポートされません)
- マルチバイト文字ごとに最大 3 バイトが必要です。

UTF-8 データを使用するが補助文字のサポートが必要なアプリケーションでは、`utf8mb3` ではなく `utf8mb4` を使用する必要があります ([セクション10.9.1「utf8mb4 文字セット \(4 バイトの UTF-8 Unicode エンコーディング\)」](#) を参照)。

`utf8mb3` および `ucs2` では、まったく同じ文字セットを使用できます。つまり、これらは同じ `repertoire` を持ちます。

`utf8` は `utf8mb3` のエイリアスです。文字制限は、名前に明示的ではなく暗黙的です。

注記

`utf8mb3` 文字セットは非推奨であり、将来の MySQL リリースで削除される予定です。かわりに `utf8mb4` を使用してください。 `utf8` は現在 `utf8mb3` のエイリアスですが、ある時点では `utf8` が `utf8mb4` への参照になることが予想されます。 `utf8` の意味があいまいにならないように、`utf8` ではなく文字セット参照に `utf8mb4` を明示的に指定することを検討してください。

`utf8mb3` は `CHARACTER SET` 句、`utf8mb3_collation_substring` は `COLLATE` 句 (`collation_substring` が `bin`, `czech_ci`, `danish_ci`, `esperanto_ci`, `estonian_ci` など) で使用できます。例:

```
CREATE TABLE t (s1 CHAR(1) CHARACTER SET utf8mb3;  
SELECT * FROM t WHERE s1 COLLATE utf8mb3_general_ci = 'x';  
DECLARE x VARCHAR(5) CHARACTER SET utf8mb3 COLLATE utf8mb3_danish_ci;  
SELECT CAST('a' AS CHAR CHARACTER SET utf8) COLLATE utf8_czech_ci;
```

MySQL では、ステートメントの `utf8mb3` のインスタンスが `utf8` に即時に変換されるため、`SHOW CREATE TABLE`、`SELECT CHARACTER_SET_NAME FROM INFORMATION_SCHEMA.COLUMNS`、`SELECT COLLATION_NAME FROM INFORMATION_SCHEMA.COLUMNS` などのステートメントでは、ユーザーには `utf8` または `utf8_collation_substring` という名前が表示されます。

`utf8mb3` は、`CHARACTER SET` 句以外のコンテキストでも有効です。例:

```
mysqld --character-set-server=utf8mb3
```

```
SET NAMES 'utf8mb3'; /* and other SET statements that have similar effect */  
SELECT _utf8mb3 'a';
```

マルチバイト文字セットに関連するデータ型の記憶域の詳細は、[文字列型の格納要件](#) を参照してください。

10.9.3 utf8 文字セット (utf8mb3 のエイリアス)

`utf8` は、`utf8mb3` 文字セットのエイリアスです。詳細は、[セクション10.9.2「utf8mb3 文字セット \(3 バイトの UTF-8 Unicode エンコーディング\)」](#) を参照してください。

注記

`utf8mb3` 文字セットは非推奨であり、将来の MySQL リリースで削除される予定です。かわりに `utf8mb4` を使用してください。 `utf8` は現在 `utf8mb3` のエイリアスですが、ある時点では `utf8` が `utf8mb4` への参照になることが予想されます。 `utf8` の意味があいまいにならないように、`utf8` ではなく文字セット参照に `utf8mb4` を明示的に指定することを検討してください。

10.9.4 ucs2 文字セット (UCS-2 Unicode エンコーディング)

UCS-2 では、すべての文字が 2 バイトの Unicode コードで表され、もっとも重要なバイトは 1 番目のバイトです。例: `LATIN CAPITAL LETTER A` にはコード `0x0041` があり、これは 2 バイトのシーケンス (`0x00 0x41`) として格納されます。 `CYRILLIC SMALL LETTER YERU` (Unicode `0x044B`) は、2 バイトシーケンス (`0x04 0x4B`) として格納されます。 Unicode 文字とそのコードについては、「[Unicode Consortium の web サイト](#)」を参照してください。

`ucs2` 文字セットには、次の特性があります:

- BMP 文字のみをサポート (補助文字はサポートされません)
- 固定長の 16 ビットエンコーディングを使用し、文字ごとに 2 バイトが必要です。

10.9.5 utf16 文字セット (UTF-16 Unicode エンコーディング)

`utf16` 文字セットは、補助文字のエンコーディングを可能にする拡張機能を備えた `ucs2` 文字セットです。

- BMP 文字の場合、`utf16` と `ucs2` には、同じコード値、同じエンコーディング、同じ長さという同一のストレージ特性があります。
- 補助文字の場合、`utf16` には、32 ビットを使用する文字を表すための特殊シーケンスがあります。これは、「サロゲート」メカニズムと呼ばれます。 `0xffff` より大きな数値の場合、10 ビットを確保して、これらを `0xd800` に追加し、最初の 16 ビット語に配置します。さらに 10 ビットを確保して、これらを `0xdc00` に追加し、次の 16 ビット語に配置します。この結果、すべての補助文字に 32 ビットが必要になります。このうち最初の 16 ビットは `0xd800` と `0xdbff` の間の数値であり、残りの 16 ビットは `0xdc00` と `0xdfff` の間の数値になります。 Unicode 4.0 ドキュメントの「[15.5 Surrogates Area](#)」に例が記載されています。

`utf16` はサロゲートをサポートし、`ucs2` をサポートしていないので、`utf16` でのみ適用される妥当性チェックがあります。下位サロゲートがなければ上位サロゲートを挿入できず、逆も同様です。例:

```
INSERT INTO t (ucs2_column) VALUES (0xd800); /* legal */
INSERT INTO t (utf16_column)VALUES (0xd800); /* illegal */
```

技術的に有効であるが真の Unicode ではない文字 (つまり、Unicode が「未割り当てコードポイント」または「個人使用」文字、さらには `0xffff` のように「不正」と見なす文字) に対する妥当性チェックはありません。たとえば、`U+F8FF` は Apple のロゴなので、これは有効です。

```
INSERT INTO t (utf16_column)VALUES (0xf8ff); /* legal */
```

このような文字は、すべてのユーザーに対し同じ意味を持たせることは期待できません。

MySQL は、最悪の場合 (文字が 4 バイトを必要とする場合) に対応する必要があるため、`utf16` カラムまたはインデックスの最大長は、`ucs2` カラムまたはインデックスの最大長の半分しかありません。たとえば、`MEMORY` テーブルのインデックスキーの最大長は 3072 バイトであるため、次のステートメントは、`ucs2` および `utf16` カラムに対して許可されている最長のインデックスを持つテーブルを作成します:

```
CREATE TABLE tf (s1 VARCHAR(1536) CHARACTER SET ucs2) ENGINE=MEMORY;
CREATE INDEX i ON tf (s1);
CREATE TABLE tg (s1 VARCHAR(768) CHARACTER SET utf16) ENGINE=MEMORY;
CREATE INDEX i ON tg (s1);
```

10.9.6 utf16le 文字セット (UTF-16LE Unicode エンコーディング)

これは、`utf16` と同じですが、ビッグエンディアンではなくリトルエンディアンです。

10.9.7 utf32 文字セット (UTF-32 Unicode エンコーディング)

utf32 文字セットは固定長です (**ucs2** と同様で、**utf16** とは異なります)。**utf32** はすべての文字に 32 ビットを使用し、**ucs2** (すべての文字に 16 ビットを使用します) と同様に、**utf16** (一部の文字に 16 ビットを、ほかの文字に 32 ビットを使用します) と異なります。

utf32 は、**ucs2** の 2 倍のスペース、**utf16** よりも多くのスペースを必要としますが、**utf32** には、ストレージについて予測可能であるという **ucs2** と同じ利点があります。**utf32** に必要なバイト数は文字数の 4 倍になります。また、**utf16** とは異なり、**utf32** でのエンコーディングにはトリックはないので、格納された値はコード値と等しくなります。

後者の利点がどのように役立つかを説明するために、**utf32** コード値のときに **utf8mb4** 値を求める方法を示した例を挙げます。

```
/* Assume code value = 100cc LINEAR B WHEELED CHARIOT */
CREATE TABLE tmp (utf32_col CHAR(1) CHARACTER SET utf32,
                  utf8mb4_col CHAR(1) CHARACTER SET utf8mb4);
INSERT INTO tmp VALUES (0x000100cc,NULL);
UPDATE tmp SET utf8mb4_col = utf32_col;
SELECT HEX(utf32_col),HEX(utf8mb4_col) FROM tmp;
```

MySQL では、未割り当ての Unicode 文字または個人使用領域の文字の追加について広く許容しています。実際、**utf32** の妥当性チェックは 1 つしかありません。**0x10ffff** よりも大きなコード値はありません。たとえば次の場合は不正です。

```
INSERT INTO t (utf32_column) VALUES (0x110000); /* illegal */
```

10.9.8 3 バイト Unicode 文字セットと 4 バイト Unicode 文字セット間の変換

このセクションでは、**utf8mb3** と **utf8mb4** の文字セット間で文字データを変換する際に発生する可能性のある問題について説明します。

注記

この説明では、主に **utf8mb3** と **utf8mb4** の間の変換に焦点を当てていますが、**ucs2** 文字セットと **utf16** や **utf32** などの文字セットの間の変換にも同様の原則が適用されます。

utf8mb3 と **utf8mb4** の文字セットは、次の点で異なります:

- **utf8mb3** では、Basic Multilingual Plane (BMP) の文字のみがサポートされます。**utf8mb4** では、BMP の外部にある補助文字もサポートされます。
- **utf8mb3** では、文字当たり最大 3 バイトが使用されます。**utf8mb4** では、文字当たり最大 4 バイトが使用されます。

注記

この説明では、3 バイトおよび 4 バイトの UTF-8 文字セットデータの参照に関する明示的な **utf8mb3** および **utf8mb4** 文字セット名について説明します。ただし、テーブル定義では、**utf8** が使用されるのは、このような定義で指定された **utf8mb3** のインスタンスが、**utf8mb3** のエイリアスである **utf8** に MySQL によって変換されるためです。

utf8mb3 から **utf8mb4** に変換する利点の 1 つは、アプリケーションで補助文字を使用できることです。1 つのトレードオフは、これによってデータストレージ領域要件が増加する可能性があることです。

テーブルの内容に関しては、**utf8mb3** から **utf8mb4** への変換で問題は発生しません:

- BMP 文字の場合、**utf8mb4** と **utf8mb3** のストレージ特性は同じです: 同じコード値、同じエンコーディング、同じ長さ。
- 補助文字の場合、**utf8mb4** ではそれを格納するために 4 バイトが必要ですが、**utf8mb3** では文字を格納できません。**utf8mb3** カラムを **utf8mb4** に変換する場合、補助文字がないため、変換の心配は必要ありません。

テーブル構造に関して、主な潜在的な非互換性は次のとおりです:

- 可変長文字データ型 (**VARCHAR** および **TEXT** 型) の場合、**utf8mb4** カラムに許可される最大長は **utf8mb3** カラムより少なくなります。

- すべての文字データ型 ([CHAR](#)、[VARCHAR](#) および [TEXT](#) 型) について、[utf8mb4](#) カラムでインデックス付けできる最大文字数は [utf8mb3](#) カラムより少なくなります。

したがって、テーブルを [utf8mb3](#) から [utf8mb4](#) に変換するには、一部のカラムまたはインデックスの定義の変更が必要になる場合があります。

テーブルは、[ALTER TABLE](#) を使用して [utf8mb3](#) から [utf8mb4](#) に変換できます。テーブルに次の定義があるとし
ます:

```
CREATE TABLE t1 (  
  col1 CHAR(10) CHARACTER SET utf8 COLLATE utf8_unicode_ci NOT NULL,  
  col2 CHAR(10) CHARACTER SET utf8 COLLATE utf8_bin NOT NULL  
) CHARACTER SET utf8;
```

次のステートメントは、[utf8mb4](#) を使用するように [t1](#) を変換します。

```
ALTER TABLE t1  
  DEFAULT CHARACTER SET utf8mb4,  
  MODIFY col1 CHAR(10)  
    CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,  
  MODIFY col2 CHAR(10)  
    CHARACTER SET utf8mb4 COLLATE utf8mb4_bin NOT NULL;
```

[utf8mb3](#) から [utf8mb4](#) への変換時のキャッチは、bytes に関してカラムまたはインデックスキーの最大長が変更されないことです。そのため、文字の最大長が 3 ではなく 4 なので、文字の点ではこれはより小さくなります。[CHAR](#)、[VARCHAR](#) および [TEXT](#) データ型の場合は、MySQL テーブルを変換する際に次の問題を確認してください:

- [utf8mb3](#) カラムのすべての定義をチェックし、ストレージエンジンの最大長を超えていないことを確認します。
- [utf8mb3](#) カラムのすべてのインデックスをチェックし、ストレージエンジンの最大長を超えていないことを確認します。最大値は、ストレージエンジンの機能強化によって変更されることがあります。

前述の条件が適用される場合は、定義されているカラムまたはインデックスの長さを減らすか、[utf8mb4](#) ではなく [utf8mb3](#) を引き続き使用する必要があります。

次に、構造的な変更が必要な例をいくつか示します。

- [TINYTEXT](#) カラムは、最大 255 バイトを保持できるので、3 バイトの文字は 85 個まで、4 バイトの文字は 63 個まで保持できます。[utf8mb3](#) を使用する [TINYTEXT](#) カラムがあるが、63 文字を超えることができる必要があるとします。データ型も [TEXT](#) などのより長い型に変更しないかぎり、これを [utf8mb4](#) に変換できません。

同様に、[utf8mb3](#) から [utf8mb4](#) に変換する場合は、非常に長い [VARCHAR](#) カラムを長い [TEXT](#) 型のいずれかに変更する必要があります。

- [InnoDB](#) では、[COMPACT](#) または [REDUNDANT](#) の行形式を使用するテーブルのインデックスの最大長は 767 バイトであるため、[utf8mb3](#) または [utf8mb4](#) のカラムでは、それぞれ 255 文字または 191 文字までインデックス付けできます。191 文字を超えるインデックスを持つ [utf8mb3](#) カラムが現在ある場合は、インデックス付けする文字数を減らす必要があります。

[COMPACT](#) または [REDUNDANT](#) の行形式を使用する [InnoDB](#) テーブルでは、次のカラムおよびインデックスの定義が有効です:

```
col1 VARCHAR(500) CHARACTER SET utf8, INDEX (col1(255))
```

代わりに [utf8mb4](#) を使用するには、インデックスをより小さくする必要があります。

```
col1 VARCHAR(500) CHARACTER SET utf8mb4, INDEX (col1(191))
```

注記

[COMPRESSED](#) または [DYNAMIC](#) の行形式を使用する [InnoDB](#) テーブルの場合、767 バイト (最大 3072 バイト) を超える [index key prefixes](#) が許可されます。これらの行フォーマットを使用して作成されたテーブルでは、[utf8mb3](#) カラムまたは [utf8mb4](#) カラムに対してそれぞれ最大 1024 文字または 768 文字をインデックス付けできます。関連情報については、[セクション15.22 「InnoDB の制限」](#)、および [DYNAMIC 行フォーマット](#) を参照してください。

前述の変更のタイプは、非常に長いカラムまたはインデックスを持つ場合にのみ必要になる可能性が高くなります。それ以外の場合は、前述の `ALTER TABLE` を使用して、問題なくテーブルを `utf8mb3` から `utf8mb4` に変換できます。

次の項目は、その他の潜在的な非互換性をまとめたものです:

- `SET NAMES 'utf8mb4'` では、接続文字セットに対して 4 バイトの文字セットを使用する必要があります。4 バイト文字がサーバーから送信されていないかぎり、問題は起こりません。それ以外の場合は、文字ごとに最大 3 バイトの受信を要求するアプリケーションで問題が発生する可能性があります。反対に、4 バイトの文字の送信を要求するアプリケーションは、その文字がサーバーで認識されることを確認する必要があります。
- レプリケーションでは、補助文字をサポートする文字セットをソースで使用する場合、すべてのレプリカもそれらを理解する必要があります。

また、ソースとレプリカでテーブルの定義が異なる場合、予期しない結果が発生する可能性があることにも注意してください。たとえば、インデックスキーの最大長が異なると、ソースで `utf8mb3` を使用し、レプリカで `utf8mb4` を使用するリスクがあります。

`utf8mb4`、`utf16`、`utf16le` または `utf32` に変換し、`utf8mb3` または `ucs2` に変換しなおすことにした場合 (古いバージョンの MySQL にダウングレードする場合など)、次の考慮事項が適用されます:

- `utf8mb3` および `ucs2` データに問題はありません。
- サーバーは、変換元の文字セットを参照する定義を認識するのに十分な最新の状態である必要があります。
- `utf8mb4` 文字セットを参照するオブジェクト定義の場合、ダウングレード前に `mysqldump` でダンプし、ダンプファイルを編集して `utf8mb4` のインスタンスを `utf8` に変更し、データに 4 バイト文字がないかぎり、古いサーバーでファイルをリロードできます。古いサーバーでは、ダンプファイルのオブジェクト定義に `utf8` が表示され、(3 バイト) `utf8` 文字セットを使用する新しいオブジェクトが作成されます。

10.10 サポートされる文字セットおよび照合順序

このセクションでは MySQL がどの文字セットをサポートするかを示します。関連する文字セットのグループごとに、1 つのサブセクションがあります。文字セットごとに、許容される照合順序が一覧表示されます。

使用可能な文字セットとそのデフォルトの照合順序をリストするには、`SHOW CHARACTER SET` ステートメントを使用するか、`INFORMATION_SCHEMA.CHARACTER_SETS` テーブルをクエリーします。例:

```
mysql> SHOW CHARACTER SET;
+-----+-----+-----+-----+
| Charset | Description                | Default collation | Maxlen |
+-----+-----+-----+-----+
| armSCII8 | ARMSCII-8 Armenian        | armSCII8_general_ci | 1 |
| ascii    | US ASCII                   | ascii_general_ci   | 1 |
| big5     | Big5 Traditional Chinese  | big5_chinese_ci   | 2 |
| binary   | Binary pseudo charset     | binary             | 1 |
| cp1250   | Windows Central European | cp1250_general_ci | 1 |
| cp1251   | Windows Cyrillic          | cp1251_general_ci | 1 |
| cp1256   | Windows Arabic            | cp1256_general_ci | 1 |
| cp1257   | Windows Baltic            | cp1257_general_ci | 1 |
| cp850    | DOS West European         | cp850_general_ci  | 1 |
| cp852    | DOS Central European      | cp852_general_ci  | 1 |
| cp866    | DOS Russian               | cp866_general_ci  | 1 |
| cp932    | SJIS for Windows Japanese | cp932_japanese_ci | 2 |
| dec8     | DEC West European         | dec8_swedish_ci   | 1 |
| eucjpms  | UJIS for Windows Japanese | eucjpms_japanese_ci | 3 |
| euckr    | EUC-KR Korean             | euckr_korean_ci   | 2 |
| gb18030  | China National Standard GB18030 | gb18030_chinese_ci | 4 |
| gb2312   | GB2312 Simplified Chinese | gb2312_chinese_ci | 2 |
| gbk      | GBK Simplified Chinese    | gbk_chinese_ci    | 2 |
| geostd8  | GEOSTD8 Georgian          | geostd8_general_ci | 1 |
| greek    | ISO 8859-7 Greek          | greek_general_ci  | 1 |
| hebrew   | ISO 8859-8 Hebrew         | hebrew_general_ci | 1 |
| hp8      | HP West European          | hp8_english_ci    | 1 |
| keybcs2  | DOS Kamenicky Czech-Slovak | keybcs2_general_ci | 1 |
| koi8r    | KOI8-R Relcom Russian     | koi8r_general_ci  | 1 |
| koi8u    | KOI8-U Ukrainian         | koi8u_general_ci  | 1 |
| latin1   | cp1252 West European      | latin1_swedish_ci | 1 |
| latin2   | ISO 8859-2 Central European | latin2_general_ci  | 1 |
```

latin5	ISO 8859-9 Turkish	latin5_turkish_ci	1
latin7	ISO 8859-13 Baltic	latin7_general_ci	1
macce	Mac Central European	macce_general_ci	1
macroman	Mac West European	macroman_general_ci	1
sjis	Shift-JIS Japanese	sjis_japanese_ci	2
swe7	7bit Swedish	swe7_swedish_ci	1
tis620	TIS620 Thai	tis620_thai_ci	1
ucs2	UCS-2 Unicode	ucs2_general_ci	2
ujis	EUC-JP Japanese	ujis_japanese_ci	3
utf16	UTF-16 Unicode	utf16_general_ci	4
utf16le	UTF-16LE Unicode	utf16le_general_ci	4
utf32	UTF-32 Unicode	utf32_general_ci	4
utf8	UTF-8 Unicode	utf8_general_ci	3
utf8mb4	UTF-8 Unicode	utf8mb4_0900_ai_ci	4

文字セットに複数の照合順序が存在する場合、どの照合順序が所定のアプリケーションにもっとも適しているかが明確でないことがあります。正しくない照合順序を選択しないようにするには、代表的なデータ値で比較を行い、特定の照合順序で期待どおりに値がソートされることを確認すると役立ちます。

10.10.1 Unicode 文字セット

このセクションでは、Unicode 文字セットで使用可能な照合順序とその区別プロパティについて説明します。Unicode の一般情報については、[セクション10.9「Unicode のサポート」](#)を参照してください。

MySQL では、複数の Unicode 文字セットがサポートされています:

- **utf8mb4**: Unicode 文字セットの UTF-8 エンコーディング。文字ごとに 1 バイトから 4 バイトを使用します。
- **utf8mb3**: Unicode 文字セットの UTF-8 エンコーディング。文字ごとに 1 バイトから 3 バイトを使用します。
- **utf8**: **utf8mb3** のエイリアス。
- **ucs2**: 文字ごとに 2 バイトを使用した Unicode 文字セットの UCS-2 エンコーディング。
- **utf16**: 文字ごとに 2 バイトまたは 4 バイトを使用する Unicode 文字セットの UTF-16 エンコーディング。**ucs2** と似ていますが、補助文字の拡張子が付いています。
- **utf16le**: Unicode 文字セットの UTF-16LE エンコーディング。**utf16** と似ていますが、ビッグエンディアンではなくリトルエンディアンです。
- **utf32**: 文字ごとに 4 バイトを使用する Unicode 文字セットの UTF-32 エンコーディング。

注記

utf8mb3 文字セットは非推奨であり、将来の MySQL リリースで削除される予定です。かわりに **utf8mb4** を使用してください。**utf8** は現在 **utf8mb3** のエイリアスですが、ある時点では **utf8** が **utf8mb4** への参照になることが予想されます。**utf8** の意味があいまいにならないように、**utf8** ではなく文字セット参照に **utf8mb4** を明示的に指定することを検討してください。

utf8mb4, **utf16**, **utf16le** および **utf32** は、BMP の外部にある Basic Multilingual Plane (BMP) 文字および補助文字をサポートしています。**utf8** および **ucs2** では BMP 文字のみがサポートされます。

ほとんどの Unicode 文字セットには、一般的な照合順序 (名前または言語指定子がない場合は **_general** で示されます)、バイナリ照合順序 (名前に **_bin** で示されます) および複数の言語固有の照合順序 (言語指定子で示されます) があります。たとえば、**utf8mb4** の場合、**utf8mb4_general_ci** および **utf8mb4_bin** はその一般照合およびバイナリ照合であり、**utf8mb4_danish_ci** は言語固有の照合のいずれかです。

ほとんどの文字セットには単一のバイナリ照合順序があります。**utf8mb4** は、次の例外です: **utf8mb4_bin** および (MySQL 8.0.17 以降の) **utf8mb4_0900_bin**。これら 2 つのバイナリ照合順序は同じですが、パッド属性と照合順序特性によって区別されます。[照合パッド属性](#)および[文字照合ウエイト](#)を参照してください。

utf16le の照合サポートは制限されています。使用可能な唯一の照合順序は、**utf16le_general_ci** と **utf16le_bin** です。これらは、**utf16_general_ci** と **utf16_bin** に似ています。

- [Unicode 照合アルゴリズム \(UCA\) のバージョン](#)
- [照合パッド属性](#)
- [言語固有の照合](#)
- [_general_ci と unicode_ci の照合順序](#)
- [文字照合ウェイト](#)
- [その他の情報](#)

Unicode 照合アルゴリズム (UCA) のバージョン

MySQL は、<http://www.unicode.org/reports/tr10/> で説明している Unicode 照合順序アルゴリズム (UCA) に従って `xxx_unicode_ci` 照合順序を実装します。照合順序は、バージョン 4.0.0 UCA 重みキー (<http://www.unicode.org/Public/UCA/4.0.0/allkeys-4.0.0.txt>) を使用します。`xxx_unicode_ci` 照合では、Unicode 照合アルゴリズムの一部のみがサポートされます。一部の文字はサポートされておらず、結合マークは完全にサポートされていません。これは主に、ベトナム語、ヨルバ語、ナバホ語などの一部のより小さな言語に影響します。結合された文字は、文字列比較で単一の Unicode 文字を使用して書き込まれた同じ文字とは異なるとみなされ、その 2 つの文字の長さは異なるとみなされず (たとえば、`CHAR_LENGTH()` 関数または結果セットメタデータによって返されます)。

4.0.0 より上位の UCA バージョンに基づく Unicode 照合では、照合名にバージョンが含まれます。例:

- `utf8mb4_unicode_520_ci` は UCA 5.2.0 重みキー (<http://www.unicode.org/Public/UCA/5.2.0/allkeys.txt>) に基づいています。
- `utf8mb4_0900_ai_ci` は UCA 9.0.0 の重みキー (<http://www.unicode.org/Public/UCA/9.0.0/allkeys.txt>) に基づいています。

`LOWER()` および `UPPER()` 関数は、引数の照合順序に従って大/小文字の折りたたみを実行します。4.0.0 より大きい Unicode バージョンでのみ大文字と小文字のバージョンを持つ文字は、引数照合で十分な UCA バージョンが使用されている場合にのみ、これらの関数によって変換されます。

照合パッド属性

UCA 9.0.0 以上に基づく照合は、9.0.0 より前の UCA バージョンに基づく照合より高速です。また、9.0.0 より前の UCA バージョンに基づく照合で使用される `PAD SPACE` とは対照的に、`NO PAD` のパッド属性もあります。非バイナリ文字列を比較するために、`NO PAD` 照合順序では、文字列の末尾のスペースは他の文字と同様に扱われます ([比較での後続領域の処理](#) を参照)。

照合のパッド属性を決定するには、`PAD_ATTRIBUTE` カラムを含む `INFORMATION_SCHEMA COLLATIONS` テーブルを使用します。例:

```
mysql> SELECT COLLATION_NAME, PAD_ATTRIBUTE
FROM INFORMATION_SCHEMA.COLLATIONS
WHERE CHARACTER_SET_NAME = 'utf8mb4';
```

COLLATION_NAME	PAD_ATTRIBUTE
utf8mb4_general_ci	PAD SPACE
utf8mb4_bin	PAD SPACE
utf8mb4_unicode_ci	PAD SPACE
utf8mb4_icelandic_ci	PAD SPACE
...	
utf8mb4_0900_ai_ci	NO PAD
utf8mb4_de_pb_0900_ai_ci	NO PAD
utf8mb4_is_0900_ai_ci	NO PAD
...	
utf8mb4_ja_0900_as_cs	NO PAD
utf8mb4_ja_0900_as_cs_ks	NO PAD
utf8mb4_0900_as_ci	NO PAD
utf8mb4_ru_0900_ai_ci	NO PAD
utf8mb4_ru_0900_as_cs	NO PAD
utf8mb4_zh_0900_as_cs	NO PAD
utf8mb4_0900_bin	NO PAD

NO PAD 照合順序を持つ非バイナリ文字列値 (CHAR、VARCHAR および TEXT) の比較は、末尾の空白に関して他の照合順序とは異なります。たとえば、'a' と 'a ' は、同じ文字列ではなく異なる文字列として比較されます。これは、utf8mb4 のバイナリ照合を使用して確認できます。utf8mb4_bin の pad 属性は PAD SPACE ですが、utf8mb4_0900_bin の場合は NO PAD です。したがって、utf8mb4_0900_bin に関連する操作では末尾にスペースが追加されず、末尾にスペースがある文字列を含む比較は、次の 2 つの照合順序で異なる場合があります:

```
mysql> CREATE TABLE t1 (c CHAR(10) COLLATE utf8mb4_bin);
Query OK, 0 rows affected (0.03 sec)

mysql> INSERT INTO t1 VALUES('a');
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM t1 WHERE c = 'a';
+----+
| c  |
+----+
| a  |
+----+
1 row in set (0.00 sec)

mysql> ALTER TABLE t1 MODIFY c CHAR(10) COLLATE utf8mb4_0900_bin;
Query OK, 0 rows affected (0.02 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM t1 WHERE c = 'a';
Empty set (0.00 sec)
```

言語固有の照合

Unicode 照合アルゴリズム (UCA) のみに基づく順序付けが言語に対して適切に機能しない場合、MySQL は言語固有の Unicode 照合を実装します。言語固有の照合は UCA ベースであり、追加の言語調整ルールがあります。このようなルールの例は、このセクションの後半で説明します。特定の言語の順序付けに関する質問については、[unicode.org](http://www.unicode.org/cldr/charts/30/collation/index.html) の <http://www.unicode.org/cldr/charts/30/collation/index.html> に Common Locale Data Repository (CLDR) 照合チャートが用意されています。

たとえば、非言語固有の utf8mb4_0900_ai_ci および言語固有の utf8mb4_LOCALE_0900_ai_ci Unicode 照合には、それぞれ次の特性があります:

- 照合は UCA 9.0.0 および CLDR v30 に基づいており、アクセントは区別されず、大/小文字も区別されません。これらの特性は、照合名に `_0900`、`_ai` および `_ci` で示されます。例外: CLDR で古典ラテンが定義されていないため、`utf8mb4_la_0900_ai_ci` は CLDR に基づいていません。
- 照合は、[U+0, U+10FFFF]の範囲のすべての文字に対して機能します。
- 照合順序が言語固有でない場合は、補助文字を含むすべての文字がデフォルトの順序 (後述) でソートされます。照合順序が言語固有の場合、言語固有のルールに従って言語の文字が正しくソートされ、言語内にない文字がデフォルトの順序でソートされます。
- デフォルトでは、照合では、DUCET テーブル (デフォルト Unicode 照合要素テーブル) にリストされているコードポイントを持つ文字が、テーブルに割り当てられている重み値に従ってソートされます。照合では、UCA に従って構成される暗黙的な重み値を使用して、DUCET テーブルにコードポイントがリストされていない文字がソートされます。
- 言語固有でない照合の場合、縮約シーケンス内の文字は個別の文字として扱われます。言語固有の照合順序の場合、縮約によって文字のソート順が変更されることがあります。

次のテーブルに示すロケールコードまたは言語名を含む照合名は、言語固有の照合です。Unicode 文字セットには、これらの言語の照合順序を含めることができます。

表 10.3 Unicode 照合言語指定子

Language	言語指定子
中国語	zh

Language	言語指定子
古典ラテン語	la または roman
クロアチア語	hr または croatian
チェコ語	cs または czech
デンマーク語	da または danish
エスペラント	eo または esperanto
エストニア語	et または estonian
ドイツ電話帳の順序	de_pb または german2
ハンガリー語	hu または hungarian
アイスランド語	is または icelandic
日本語	ja
ラトビア語	lv または latvian
リトアニア語	lt または lithuanian
ペルシア語	persian
ポーランド語	pl または polish
ルーマニア語	ro または romanian
ロシア語	ru
シンハラ	sinhala
スロバキア語	sk または slovak
スロベニア語	sl または slovenian
現代スペイン語	es または spanish
スペイン語 (繁体字)	es_trad または spanish2
スウェーデン語	sv または swedish
トルコ語	tr または turkish
ベトナム語	vi または vietnamese

クロアチア語の照合順序は、これらのクロアチア語の文字に合わせて調整されます: Č, Ć, Dž, Đ, Lj, Nj, Š, Ž。

デンマーク語の照合順序は、ノルウェー語にも使用できます。

日本語の場合、utf8mb4 文字セットには utf8mb4_ja_0900_as_cs 照合順序および utf8mb4_ja_0900_as_cs_ks 照合順序が含まれます。両方の照合でアクセントが区別され、大/小文字が区別されます。また、utf8mb4_ja_0900_as_cs_ks はカタカナ文字をひらがな文字と区別し、utf8mb4_ja_0900_as_cs はカタカナ文字とひらがな文字をソート用に同等として扱います。日本語照合を必要とするが、kana 感度を必要としないアプリケーションでは、ソートパフォーマンスを向上させるために utf8mb4_ja_0900_as_cs を使用できます。utf8mb4_ja_0900_as_cs ではソートに 3 つの重みレベルが使用され、utf8mb4_ja_0900_as_cs_ks では 4 つが使用されます。

アクセントを区別しない古典ラテン照合の場合、I と J は等しいと比較され、U と V は等しいと比較されます。I と J、および U と V は、基本文字レベルと同じように比較されます。つまり、J はアクセント付き I とみなされ、U はアクセント付き V とみなされます。

スペイン語照合は、最新のスペイン語および従来スペイン語で使用できます。どちらの場合も、ñ (n-tilde) は n と o の間の個別の文字です。また、スペイン語 (繁体字) の場合、ch は c と d の間の個別の文字であり、ll は l と m の間の個別の文字です。

スペイン語の従来照合は、アストゥリア語およびガリシア語にも使用できます。

スウェーデン語照合にはスウェーデン語ルールが含まれます。たとえば、スウェーデン語には、ドイツ語やフランス語を使用するユーザーでは予期しないような次の関係があります。

```
Ü = Y < Ö
```

_general_ci と unicode_ci の照合順序

Unicode 文字セットの場合、`xxx_general_ci` 照合順序を使用して実行する演算は、`xxx_unicode_ci` 照合順序のものよりも高速です。たとえば、`utf8_general_ci` 照合順序の比較は、`utf8_unicode_ci` の比較よりも高速ですが、精度は少し低くなります。これは、`utf8_unicode_ci` で拡張などのマッピングがサポートされているためです。つまり、一方の文字が他の文字の組合せと等しいと比較される場合です。たとえば、ß はドイツ語および他の一部の言語の `ss` と同等です。`utf8_unicode_ci` では、縮約および無視可能な文字もサポートされています。`utf8_general_ci` は、拡張、縮小または無視可能な文字をサポートしないレガシー照合です。文字間で 1 対 1 の比較しかできません。

さらに詳しく説明するために、`utf8_general_ci` と `utf8_unicode_ci` の両方で次の等価が保持されています (比較または検索での影響については、[セクション10.8.6「照合順序の効果の例」](#) を参照してください):

```
Ä = A
Ö = O
Ü = U
```

照合順序間の違いは、次の式が `utf8_general_ci` に当てはまるという点です。

```
ß = s
```

一方、次の式は、ドイツ語 DIN-1 順序 (辞書順序とも呼ばれます) をサポートする `utf8_unicode_ci` に当てはまりません。

```
ß = ss
```

`utf8_unicode_ci` での順序付けが言語に対して適切に機能しない場合、MySQL は `utf8` 言語固有の照合を実装します。たとえば、`utf8_unicode_ci` は、ドイツ語辞書順序とフランス語には適切に機能するので、特殊な `utf8` 照合順序を作成する必要はありません。

`utf8_general_ci` は、ß が `ss` ではなく `s` と同等であることを除き、ドイツ語とフランス語の両方にも満足しています。これがアプリケーションで許容可能な場合は、`utf8_general_ci` のほうが高速なので、こちらを使用する必要があります。これが許容できない場合 (たとえば、ドイツ語辞書順序が必要な場合) は、`utf8_unicode_ci` のほうがより正確なので、こちらを使用してください。

ドイツ語 DIN-2 (電話帳) 順序が必要な場合は、`utf8_german2_ci` 照合順序を使用してください。これは次の文字セットを等しいものと見なします。

```
Ä = Æ = AE
Ö = Œ = OE
Ü = UE
ß = ss
```

`utf8_german2_ci` は `latin1_german2_ci` に似ていますが、後者は `AE` と同等の `Æ` または `OE` と同等の `Œ` を比較しません。`utf8_general_ci` で十分であるので、ドイツ語辞書順序のための `latin1_german_ci` に対応する `utf8_german_ci` はありません。

文字照合ウェイト

文字照合の重みは、次のように決定されます:

- `_bin` (バイナリ) 照合順序を除くすべての Unicode 照合順序について、MySQL はテーブル検索を実行して文字照合順序を検索します。
- `utf8mb4_0900_bin` 以外の `_bin` 照合順序の場合、重みはコードポイントに基づき、先行するゼロバイトが追加される場合があります。
- `utf8mb4_0900_bin` の場合、重みは `utf8mb4` エンコーディングバイトです。ソート順序は `utf8mb4_bin` の場合と同じですが、はるかに高速です。

照合加重は、`WEIGHT_STRING()` 関数を使用して表示できます。([セクション12.8「文字列関数および演算子」](#) を参照してください。) 照合順序で重み参照テーブルが使用されているが、文字がテーブルにない場合 (たとえば、「new」文字であるため)、照合順序決定はより複雑になります:

- 一般照合 (`xxx_general_ci`) の BMP 文字の場合、重みはコードポイントです。
- UCA 照合順序 (たとえば、`xxx_unicode_ci` と言語固有の照合順序) での BMP 文字の場合、次のアルゴリズムが適用されます。

```
if (code >= 0x3400 && code <= 0x4DB5)
  base= 0xFB80; /* CJK Ideograph Extension */
else if (code >= 0x4E00 && code <= 0x9FA5)
  base= 0xFB40; /* CJK Ideograph */
else
  base= 0xFBC0; /* All other characters */
aaaa= base + (code >> 15);
bbbb= (code & 0x7FFF) | 0x8000;
```

結果は、`aaaa` に `bbbb` が続いた 2 つの照合要素のシーケンスになります。例:

```
mysql> SELECT HEX(WEIGHT_STRING(_ucs2 0x04CF COLLATE ucs2_unicode_ci));
+-----+
| HEX(WEIGHT_STRING(_ucs2 0x04CF COLLATE ucs2_unicode_ci)) |
+-----+
| FBC084CF |
+-----+
```

したがって、`U+04cf CYRILLIC SMALL LETTER PALOCHKA` は、すべての UCA 4.0.0 照合順序で、`U+04c0 CYRILLIC LETTER PALOCHKA` より大きくなります。UCA 5.2.0 照合順序では、すべての `palochka` は一緒にソートされます。

- 一般的な照合順序の補助文字の場合、重みは `0xffff REPLACEMENT CHARACTER` の重みです。UCA 4.0.0 照合順序の補助文字の場合、照合重みは `0xffff` です。つまり、MySQL では、すべての補助文字は互いに等しく、ほとんどすべての BMP 文字より大きくなります。

デザレット文字と `COUNT(DISTINCT)` を使用した例:

```
CREATE TABLE t (s1 VARCHAR(5) CHARACTER SET utf32 COLLATE utf32_unicode_ci);
INSERT INTO t VALUES (0xffff); /* REPLACEMENT CHARACTER */
INSERT INTO t VALUES (0x010412); /* DESERET CAPITAL LETTER BEE */
INSERT INTO t VALUES (0x010413); /* DESERET CAPITAL LETTER TEE */
SELECT COUNT(DISTINCT s1) FROM t;
```

結果は 2 です。これは、MySQL `xxx_unicode_ci` 照合順序で、置換文字は `0x0dc6` の重みを持ちますが、デザレット B とデザレット T はどちらも、`0xffff` の重みを持つからです。(`utf32_general_ci` 照合順序がかわりに使用された場合、3 文字すべての照合順序の重みが `0xffff` であるため、結果は 1 になります。)

くさび形文字と `WEIGHT_STRING()` を使用した例:

```
/*
The four characters in the INSERT string are
00000041 # LATIN CAPITAL LETTER A
0001218F # CUNEIFORM SIGN KAB
000121A7 # CUNEIFORM SIGN KISH
00000042 # LATIN CAPITAL LETTER B
*/
CREATE TABLE t (s1 CHAR(4) CHARACTER SET utf32 COLLATE utf32_unicode_ci);
INSERT INTO t VALUES (0x000000410001218f000121a700000042);
SELECT HEX(WEIGHT_STRING(s1)) FROM t;
```

結果は次のようになります。

```
0E33 FFFD FFFD 0E4A
```

`0E33` と `0E4A` は、UCA 4.0.0 の主要な重みです。 `FFFD` は KAB の重みであり、KISH の重みでもあります。

すべての補助文字が互いに同じであるというルールは、最適ではありませんが、問題を引き起こすとは考えられません。これらの文字は非常にまれであるため、複数文字列が完全に補助文字で構成されることは非常にまれです。日本では、補助文字はいまいな漢字表意文字であるので、一般的なユーザーは、いずれにしてもそれらの順序を気にしません。実際には、行を MySQL ルールでソートし、次にコードポイント値でソートする場合は、次のように簡単です:

```
ORDER BY s1 COLLATE utf32_unicode_ci, s1 COLLATE utf32_bin
```

- 4.0.0 より上位の UCA バージョン (たとえば、`xxx_unicode_520_ci`) に基づく補助文字の場合、補助文字の照合加重がすべて同じであるとはかぎりません。一部には、UCA `allkeys.txt` ファイルからの明示的な重みがあります。それ以外には、このアルゴリズムから計算された重みがあります。

```
aaaa= base + (code >> 15);
bbbb= (code & 0x7FFF) | 0x8000;
```

「文字のコード値による順序付け」と「文字のバイナリ表現による順序付け」と間には違いがあります。これは、サロゲートのために、`utf16_bin` でのみ表示される違いです。

`utf16_bin` (`utf16` のバイナリ照合順序) が、「文字ごと」ではなく「バイトごと」のバイナリ比較であったとします。その場合は、`utf16_bin` で文字の順序は `utf8_bin` で文字の順序とは異なります。たとえば、次の表には 2 つの珍しい文字が示されています。最初の文字は `E000-FFFF` の範囲にあるので、サロゲートより大きくなりますが、補助文字より小さくなります。2 番目の文字は補助文字です。

Code point	Character	utf8	utf16
0FF9D	HALFWIDTH KATAKANA LETTER N	EF BE 9D	FF 9D
10384	UGARITIC LETTER DELTA	F0 90 8E 84	D8 00 DF 84

表の 2 つの文字は、`0xff9d < 0x10384` であるので、コードポイント値による順序になります。また、`0xef < 0xf0` なので、`utf8` 値による順序になります。ただし、`0xff > 0xd8` なので、バイトごとの比較を使用する場合は、`utf16` 値による順序にはなりません。

したがって、MySQL の `utf16_bin` 照合順序は「バイトごと」にはなりません。「コードポイントによる」順序になります。MySQL は、`utf16` で補助文字エンコーディングを認識すると、文字のコードポイント値に変換してから比較します。したがって、`utf8_bin` と `utf16_bin` の順序付けは同じになります。これは、UCS_BASIC 照合順序の SQL:2008 標準の要件に一致します。「UCS_BASIC は、ソートされている文字列の文字の Unicode スカラー値によって、順序付け全体が決定される照合順序です。これは UCS 文字レパートリーに適用できます。すべての文字レパートリーは、UCS レパートリーのサブセットなので、UCS_BASIC 照合順序は、すべての文字セットに適用できる可能性があります。ノート 11: 文字の Unicode スカラー値は、符号なしの整数として扱われるそのコードポイントです。」

文字セットが `ucs2` である場合、比較はバイトごとになりますが、いずれにしても、`ucs2` 文字列にはサロゲートを含めないのでください。

その他の情報

`xxx_general_mysql500_ci` 照合では、元の `xxx_general_ci` 照合の 5.1.24 より前の順序が保持され、MySQL 5.1.24 より前に作成されたテーブルのアップグレードが許可されます (Bug #27877)。

10.10.2 西ヨーロッパの文字セット

西ヨーロッパの文字セットには、大部分の西ヨーロッパ言語 (フランス語、スペイン語、カタロニア語、バスク語、ポルトガル語、イタリア語、アルバニア語、オランダ語、ドイツ語、デンマーク語、スウェーデン語、ノルウェー語、フィンランド語、フェロー語、アイスランド語、アイルランド語、スコットランド語、英語など) が含まれます。

- `ascii` (US ASCII) 照合順序:
 - `ascii_bin`
 - `ascii_general_ci` (デフォルト)
- `cp850` (DOS 西ヨーロッパ言語) 照合順序:
 - `cp850_bin`
 - `cp850_general_ci` (デフォルト)
- `dec8` (DEC 西ヨーロッパ言語) 照合順序:
 - `dec8_bin`

- `dec8_swedish_ci` (デフォルト)
- `hp8` (HP 西ヨーロッパ言語) 照合順序:
 - `hp8_bin`
 - `hp8_english_ci` (デフォルト)
- `latin1` (cp1252 西ヨーロッパ言語) 照合順序:
 - `latin1_bin`
 - `latin1_danish_ci`
 - `latin1_general_ci`
 - `latin1_general_cs`
 - `latin1_german1_ci`
 - `latin1_german2_ci`
 - `latin1_spanish_ci`
 - `latin1_swedish_ci` (デフォルト)

MySQL の `latin1` は Windows `cp1252` 文字セットと同じです。つまり、これは公式の `ISO 8859-1` または IANA (Internet Assigned Numbers Authority) `latin1` と同じですが、IANA `latin1` が、`0x80` と `0x9f` の間のコードポイントを「未定義」として扱うのに対し、`cp1252`、および MySQL の `latin1` はこれらの位置の文字を割り当てる点が異なることを示します。たとえば `0x80` はユーロの記号です。`cp1252` の「定義されていない」エントリでは、MySQL は `0x81` を Unicode `0x0081` に、`0x8d` を `0x008d` に、`0x8f` を `0x008f` に、`0x90` を `0x0090` に、`0x9d` を `0x009d` に変換します。

`latin1_swedish_ci` 照合順序は、大部分の MySQL カスタマが使用しているデフォルトです。スウェーデン語/フィンランド語の照合順序ルールに基づいているとされていますが、スウェーデン人やフィンランド人の中にはこの意見に賛同しないユーザーもいます。

`latin1_german1_ci` と `latin1_german2_ci` の照合順序は、DIN-1 と DIN-2 の標準に基づいています。DIN は「ドイツ工科大学フュールノルムン校」(ANSI と同等のドイツ語) を表します。DIN-1 は「辞書の照合順序」と呼ばれ、DIN-2 は「電話帳の照合順序」と呼ばれています。比較、または検索を行うときのこの効果の例については、[セクション10.8.6「照合順序の効果の例」](#)を参照してください。

- `latin1_german1_ci` (辞書) ルール:

```
Ä = A
Ö = O
Ü = U
ß = s
```

- `latin1_german2_ci` (電話帳) ルール:

```
Ä = AE
Ö = OE
Ü = UE
ß = ss
```

`latin1_spanish_ci` 照合では、`ñ` (n-tilde) は `n` と `o` の間の個別の文字です。

- `macroman` (Mac 西ヨーロッパ言語) 照合順序:
 - `macroman_bin`
 - `macroman_general_ci` (デフォルト)
- `swe7` (7 ビットスウェーデン語) 照合順序:

- [swe7_bin](#)
- [swe7_swedish_ci](#) (デフォルト)

10.10.3 中央ヨーロッパの文字セット

MySQL ではチェコ、スロバキア、ハンガリー、ルーマニア、スロベニア、クロアチア、ポーランド、セルビア (ラテン) で使用される文字セットも一部サポートされています。

- [cp1250](#) (Windows 中央ヨーロッパ言語) 照合順序:
 - [cp1250_bin](#)
 - [cp1250_croatian_ci](#)
 - [cp1250_czech_cs](#)
 - [cp1250_general_ci](#) (デフォルト)
 - [cp1250_polish_ci](#)
- [cp852](#) (DOS 中央ヨーロッパ言語) 照合順序:
 - [cp852_bin](#)
 - [cp852_general_ci](#) (デフォルト)
- [keybcs2](#) (DOS Kamenicky Czech-Slovak) 照合順序:
 - [keybcs2_bin](#)
 - [keybcs2_general_ci](#) (デフォルト)
- [latin2](#) (ISO 8859-2 中央ヨーロッパ言語) 照合順序:
 - [latin2_bin](#)
 - [latin2_croatian_ci](#)
 - [latin2_czech_cs](#)
 - [latin2_general_ci](#) (デフォルト)
 - [latin2_hungarian_ci](#)
- [macce](#) (Mac 中央ヨーロッパ言語) 照合順序:
 - [macce_bin](#)
 - [macce_general_ci](#) (デフォルト)

10.10.4 南ヨーロッパおよび中東の文字セット

MySQL では南ヨーロッパや中東、アルメニア語、アラビア語、ジョージア語、ギリシャ語、ヘブライ語、トルコ語の文字セットがサポートされています。

- [armscii8](#) (ARMScii8 アルメニア語) 照合順序:
 - [armscii8_bin](#)
 - [armscii8_general_ci](#) (デフォルト)
- [cp1256](#) (Windows アラビア語) 照合順序:

- [cp1256_bin](#)
- [cp1256_general_ci](#) (デフォルト)
- [geostd8](#) (GEOSTD8 ジョージア語) 照合順序:
 - [geostd8_bin](#)
 - [geostd8_general_ci](#) (デフォルト)
- [greek](#) (ISO 8859-7 ギリシャ語) 照合順序:
 - [greek_bin](#)
 - [greek_general_ci](#) (デフォルト)
- [hebrew](#) (ISO 8859-8 ヘブライ語) 照合順序:
 - [hebrew_bin](#)
 - [hebrew_general_ci](#) (デフォルト)
- [latin5](#) (ISO 8859-9 トルコ語) 照合順序:
 - [latin5_bin](#)
 - [latin5_turkish_ci](#) (デフォルト)

10.10.5 バルト語の文字セット

バルト語に含まれる文字セットにはエストニア語、ラトビア語、リトアニア言語が含まれます。

- [cp1257](#) (Windows バルト語) 照合順序:
 - [cp1257_bin](#)
 - [cp1257_general_ci](#) (デフォルト)
 - [cp1257_lithuanian_ci](#)
- [latin7](#) (ISO 8859-13 バルト語) 照合順序:
 - [latin7_bin](#)
 - [latin7_estonian_cs](#)
 - [latin7_general_ci](#) (デフォルト)
 - [latin7_general_cs](#)

10.10.6 キリル文字の文字セット

ベラルーシ語、ブルガリア語、ロシア語、ウクライナ語、セルビア語 (キリル) とともに使用するキリル文字の文字セットおよび照合順序を以下に示します。

- [cp1251](#) (Windows キリル文字) 照合順序:
 - [cp1251_bin](#)
 - [cp1251_bulgarian_ci](#)
 - [cp1251_general_ci](#) (デフォルト)
 - [cp1251_general_cs](#)

- [cp1251_ukrainian_ci](#)
- [cp866](#) (DOS ロシア語) 照合順序:
 - [cp866_bin](#)
 - [cp866_general_ci](#) (デフォルト)
- [koi8r](#) (KOI8-R Relcom Russian) 照合順序:
 - [koi8r_bin](#)
 - [koi8r_general_ci](#) (デフォルト)
- [koi8u](#) (KOI8-U ウクライナ語) 照合順序:
 - [koi8u_bin](#)
 - [koi8u_general_ci](#) (デフォルト)

10.10.7 アジアの文字セット

サポートされているアジアの文字セットには、中国語、日本語、韓国語、タイ語が含まれています。これらは複雑な場合があります。たとえば、中国語の文字セットは数千種類の文字に対応している必要があります。[cp932](#) および [sjis](#) 文字セットの追加情報については、[セクション10.10.7.1「cp932 文字セット」](#)を参照してください。中国標準 GB 18030 の文字セットサポートの詳細は、[セクション10.10.7.2「gb18030 文字セット」](#)を参照してください。

MySQL でのアジアの文字セットのサポートに関連したよくある質問および問題に対する回答については、[セクション A.11「MySQL 8.0 FAQ: MySQL の中国語、日本語、および韓国語の文字セット」](#)を参照してください。

- [big5](#) (Big5 繁体字中国語) 照合順序:
 - [big5_bin](#)
 - [big5_chinese_ci](#) (デフォルト)
- [cp932](#) (SJIS for Windows 日本語) 照合順序:
 - [cp932_bin](#)
 - [cp932_japanese_ci](#) (デフォルト)
- [eucjpms](#) (UJIS for Windows 日本語) 照合順序:
 - [eucjpms_bin](#)
 - [eucjpms_japanese_ci](#) (デフォルト)
- [euckr](#) (EUC-KR 韓国語) 照合順序:
 - [euckr_bin](#)
 - [euckr_korean_ci](#) (デフォルト)
- [gb2312](#) (GB2312 簡体字中国語) 照合順序:
 - [gb2312_bin](#)
 - [gb2312_chinese_ci](#) (デフォルト)
- [gbk](#) (GBK 簡体字中国語) 照合順序:
 - [gbk_bin](#)
 - [gbk_chinese_ci](#) (デフォルト)

- `gb18030` (China National Standard GB18030) 照合順序:

- `gb18030_bin`
- `gb18030_chinese_ci` (default)
- `gb18030_unicode_520_ci`

- `sjis` (Shift-JIS 日本語) 照合順序:

- `sjis_bin`
- `sjis_japanese_ci` (デフォルト)

- `tis620` (TIS620 タイ語) 照合順序:

- `tis620_bin`
- `tis620_thai_ci` (デフォルト)

- `ujis` (EUC-JP 日本語) 照合順序:

- `ujis_bin`
- `ujis_japanese_ci` (デフォルト)

`big5_chinese_ci` 照合順序は画数でソートします。

10.10.7.1 cp932 文字セット

`cp932` が必要な理由

MySQL では `sjis` 文字セットは IANA で定義される `Shift_JIS` 文字セットに対応しており、これらは JIS X0201 および JIS X0208 文字セットをサポートしています。(<http://www.iana.org/assignments/character-sets> を参照してください。)

ただし、記述用語としての「SHIFT JIS」の意味は非常にあいまいになっており、さまざまなベンダーが定義した `Shift_JIS` に対する拡張も含める場合があります。

たとえば、日本語 Windows 環境で使用される「SHIFT JIS」は、Microsoft による `Shift_JIS` の拡張であり、正式な名称は `Microsoft Windows Codepage : 932` または `cp932` です。 `Shift_JIS` でサポートされる文字に加え、`cp932` では、NEC 特殊文字、NEC 選定 IBM 拡張文字、IBM 選定文字などの拡張文字をサポートします。

多くの日本語ユーザーは、これらの拡張文字を使用するときに問題に直面してきました。これらの問題は次の要因によって生じていました。

- MySQL が自動的に文字セットの変換を行なっていること。
- 文字セットが Unicode (`ucs2`) を使用して変換されていること。
- `sjis` 文字セットが、これらの拡張文字の変換をサポートしていないこと。
- いわゆる「SHIFT JIS」から Unicode への変換には複数の変換ルールが存在し、文字によっては、変換ルールに従って別々に Unicode に変換される場合があること。MySQL ではこれらの変換ルールのうち、1 つしかサポートしていません (詳細は後述します)。

MySQL の `cp932` 文字セットは、これらの問題を解決するように設計されています。

MySQL が文字セットの変換をサポートするので、異なる変換ルールを持つ IANA の `Shift_JIS` と `cp932` を 2 つの異なる文字セットに区分することが重要になります。

`cp932` と `sjis` との相違点

`cp932` 文字セットは次の点で `sjis` と異なります。

- **cp932** は、NEC 特殊文字、NEC 選定 IBM 拡張文字、IBM 選定文字をサポートします。
- 一部の **cp932** 文字には、2 つの異なるコードポイントがあり、両方とも同一の Unicode コードポイントに変換されます。Unicode から **cp932** に戻すときに、どちらかのコードポイントを選択する必要があります。この「ラウンドトリップ変換」については、Microsoft が推奨するルールが使用されます。(<http://support.microsoft.com/kb/170559/EN-US/> を参照してください。)

この変換ルールは次のように機能します。

- 文字が JIS X 0208 文字と NEC 特殊文字の両方に存在する場合には、JIS X 0208 のコードポイントを使用します。
- 文字が NEC 特殊文字と IBM 選定文字の両方に存在する場合には、NEC 特殊文字のコードポイントを使用します。
- 文字が IBM 選定文字と NEC 選定 IBM 拡張文字の両方に存在する場合は、IBM 拡張文字のコードポイントを使用します。

<https://msdn.microsoft.com/en-us/goglobal/cc305152.aspx> に示す表には、**cp932** 文字の Unicode 値に関する情報が記載されています。**cp932** の表で下に 4 桁の数字が表示されている項目については、その数字は対応する Unicode (**ucs2**) エンコーディングを表します。下線付きの 2 桁の値のある表項目では、これらの 2 桁の値で始まる一定範囲の **cp932** 文字があります。このような表項目をクリックすると、これらの桁の値で始まる **cp932** 文字に対する各 Unicode 値を示したページが表示されます。

詳細は次のリンクを参照してください。それぞれ、次の文字セットのエンコーディングに対応します。

- NEC 特殊文字 (先頭バイト 0x87):

<https://msdn.microsoft.com/en-us/goglobal/gg674964>

- NEC 選択 -IBM 拡張文字 (リードバイト 0xED および 0xEE):

<https://msdn.microsoft.com/en-us/goglobal/gg671837>
<https://msdn.microsoft.com/en-us/goglobal/gg671838>

- IBM が選択した文字 (リードバイト 0xFA, 0xFB, 0xFC):

<https://msdn.microsoft.com/en-us/goglobal/gg671839>
<https://msdn.microsoft.com/en-us/goglobal/gg671840>
<https://msdn.microsoft.com/en-us/goglobal/gg671841>

- **cp932** は、**eucjpm**s と組み合わせて使用することで、ユーザー定義の文字の変換をサポートし、**sjis/ujis** の変換での問題に対応します。詳細は、<http://www.sjfaq.org/afaq/encodings.html> を参照してください。

一部の文字については、**ucs2** との間の変換は、**sjis** と **cp932** との場合と異なります。次の表に、これらの違いを示します。

ucs2 への変換:

sjis/cp932 値	sjis -> ucs2 の変換	cp932 -> ucs2 の変換
5C	005C	005C
7E	007E	007E
815C	2015	2015
815F	005C	FF3C
8160	301C	FF5E
8161	2016	2225
817C	2212	FF0D
8191	00A2	FFE0
8192	00A3	FFE1

sjis/cp932 値	sjis -> ucs2 の変換	cp932 -> ucs2 の変換
81CA	00AC	FFE2

ucs2 からの変換:

ucs2 値	ucs2 -> sjis の変換	ucs2 -> cp932 の変換
005C	815F	5C
007E	7E	7E
00A2	8191	3F
00A3	8192	3F
00AC	81CA	3F
2015	815C	815C
2016	8161	3F
2212	817C	3F
2225	3F	8161
301C	8160	3F
FF0D	3F	817C
FF3C	3F	815F
FF5E	3F	8160
FFE0	3F	8191
FFE1	3F	8192
FFE2	3F	81CA

日本語文字セットのユーザーは、`--character-set-client-handshake` (または `--skip-character-set-client-handshake`) を使用すると大きな効果が得られることに注意してください。 [セクション5.1.7「サーバーコマンドオプション」](#) を参照してください。

10.10.7.2 gb18030 文字セット

MySQL では、[gb18030](#) 文字セットは、中華人民共和国 (PRC) の正式な文字セットである「中国規格 GB 18030-2005: 情報テクノロジー - 中国語のコード化された文字セット」に対応します。

MySQL gb18030 文字セットの特性

- GB 18030-2005 標準で定義されているすべてのコードポイントをサポートします。範囲内の未割当てのコードポイント (GB+8431A439、GB+90308130) および (GB+E3329A36、GB+EF39EF39) は、'?' (0x3F) として扱われます。未割当てのコードポイントの変換では、'?'が返されます。
- すべての GB18030 コードポイントに対して UPPER および LOWER 変換をサポートします。Unicode で定義された大/小文字の折りたたみもサポートされます ([CaseFolding-6.3.0.txt](#) に基づく)。
- 他の文字セットとの間でのデータ変換をサポートします。
- `SET NAMES` などの SQL ステートメントをサポートします。
- [gb18030](#) 文字列間、および [gb18030](#) 文字列と他の文字セットの文字列間の比較をサポートします。文字列の文字セットが異なる場合は、変換が行われます。末尾の空白を含むが無視する比較もサポートされています。
- Unicode のプライベート使用領域 (U+E000、U+F8FF) は、[gb18030](#) にマップされます。
- (U+D800、U+DFFF) と GB18030 の間にマッピングはありません。この範囲のコードポイントを変換しようとする、'?'が返されます。
- 着信シーケンスが不正な場合は、エラーまたは警告が返されます。 `CONVERT()` で不正な順序が使用されると、エラーが返されます。それ以外の場合は、警告が返されます。

- `utf8` および `utf8mb4` との一貫性のために、合字で `UPPER` はサポートされていません。
- `gb18030_unicode_520_ci` 照合順序を使用する場合、合字も大文字の合字と一致します。
- 文字に複数の大文字が含まれている場合、選択した大文字が小文字自体になります。
- マルチバイトの最小長は 1、最大長は 4 です。文字セットは、最初の 1 バイトまたは 2 バイトを使用して順序の長さを決定します。

サポートされる照合

- `gb18030_bin`: バイナリ照合。
- `gb18030_chinese_ci`: Pinyin をサポートするデフォルトの照合。中国語以外の文字のソートは、元のソートキーの順序に基づきます。`UPPER(ch)` が存在する場合、元のソートキーは `GB(UPPER(ch))` です。それ以外の場合、元のソートキーは `GB(ch)` です。中国語の文字は、Unicode 共通ロケールデータリポジトリ (CLDR 24) で定義されている Pinyin 照合順序に従ってソートされます。中国語以外の文字は、コードポイントの最大値である `GB+FE39FE39` を除き、中国語の文字の前にソートされます。
- `gb18030_unicode_520_ci`: Unicode 照合。合字が正しくソートされていることを確認する必要がある場合は、この照合を使用します。

10.10.8 バイナリ文字セット

`binary` 文字セットは、バイトのシーケンスであるバイナリ文字列の文字セットです。`binary` 文字セットには、`binary` と呼ばれる照合順序があります。比較およびソートは、数値文字コード値 (マルチバイト文字の場合は数値バイト値とは異なる) ではなく、数値バイト値に基づきます。`binary` 文字セットの `binary` 照合順序と非バイナリ文字セットの `_bin` 照合順序の違いについては、[セクション10.8.5「バイナリ照合順序と_bin 照合順序」](#) を参照してください。

`binary` 文字セットの場合、大文字と小文字の区別およびアクセントの等価の概念は適用されません:

- バイナリ文字列として格納されたシングルバイト文字の場合、文字境界とバイト境界は同じであるため、大文字と小文字の違いとアクセントの違いは比較で重要です。つまり、`binary` 照合では大/小文字が区別され、アクセントが区別されます。

```
mysql> SET NAMES 'binary';
mysql> SELECT CHARSET('abc'), COLLATION('abc');
+-----+-----+
| CHARSET('abc') | COLLATION('abc') |
+-----+-----+
| binary        | binary          |
+-----+-----+
mysql> SELECT 'abc' = 'ABC', 'a' = 'ä';
+-----+-----+
| 'abc' = 'ABC' | 'a' = 'ä' |
+-----+-----+
|          0 |          0 |
+-----+-----+
```

- バイナリ文字列として格納されるマルチバイト文字の場合、文字とバイトの境界は異なります。文字境界は失われるため、文字境界に依存する比較は意味がありません。

バイナリ文字列の大文字と小文字の変換を実行するには、まず文字列に格納されているデータに適した文字セットを使用して、非バイナリ文字列に変換します:

```
mysql> SET @str = BINARY 'New York';
mysql> SELECT LOWER(@str), LOWER(CONVERT(@str USING utf8mb4));
+-----+-----+
| LOWER(@str) | LOWER(CONVERT(@str USING utf8mb4)) |
+-----+-----+
| New York   | new york                            |
+-----+-----+
```

文字列式をバイナリ文字列に変換する場合、次の構造体は同等です:

```
BINARY expr
CAST(expr AS BINARY)
```

```
CONVERT(expr USING BINARY)
```

値が文字列リテラルの場合、`_binary` イントロデューサを使用してバイナリ文字列として指定できます。例:

```
_binary 'a'
```

`_binary` イントロデューサは 16 進数リテラルおよびビット値リテラルにも使用できますが、不要です。このようなりテラルはデフォルトでバイナリ文字列です。

イントロデューサの詳細は、[セクション10.3.8「文字セットイントロデューサ」](#) を参照してください。

10.11 文字セットの制約

- 識別子は、`utf8` を使用して `mysql` データベーステーブル (`user`、`db` など) に格納されますが、識別子には Basic Multilingual Plane (BMP) の文字だけを含めることができます。識別子では補助文字は許可されません。
- `ucs2`、`utf16`、`utf16le`、および `utf32` 文字セットには次の制約があります。
 - クライアント文字セットとして使用できるものではありません。[許可されていないクライアント文字セット](#)を参照してください。
 - 現在、`LOAD DATA` を使用して、これらの文字セットを使用するデータファイルをロードすることはできません。
 - `FULLTEXT` インデックスは、これらのいずれかの文字セットを使用するカラムでは作成できません。ただし、インデックスのないカラムでは `IN BOOLEAN MODE` 検索を実行できます。
- `REGEXP` および `RLIKE` 演算子はバイト単位で機能するため、マルチバイトセーフではなく、マルチバイト文字セットを使用すると想定外の結果が生成される可能性があります。さらに、これらの演算子ではそのバイト値に基づいて文字が比較されるため、アクセント記号付き文字は、指定された照合順序では等しいとみなされた場合でも、等しいとして比較されない可能性があります。

10.12 エラーメッセージ言語の設定

デフォルトでは、`mysqld` は英語でエラーメッセージを生成しますが、他のいくつかの言語では表示できます: チェコ語、デンマーク語、オランダ語、エストニア語、フランス語、ドイツ語、ギリシャ語、ハンガリー語、イタリア語、日本語、韓国語、ノルウェー語、ノルウェー語、ニュージー語、ポーランド語、ポルトガル語、ルーマニア語、ロシア語、スロバキア語、スペイン語またはスウェーデン語。これは、サーバーがエラーログに書き込み、クライアントに送信するメッセージに適用されます。

サーバーがエラーメッセージを書き込む言語を選択するには、このセクションの手順に従います。エラーメッセージの (言語ではなく) 文字セットの変更の詳細は、[セクション10.6「エラーメッセージ文字セット」](#) を参照してください。エラーロギングの構成に関する一般情報は、[セクション5.4.2「エラーログ」](#) を参照してください。

サーバーは、次のルールを使用してエラーメッセージファイルを検索します:

- ファイルは、`lc_messages_dir` および `lc_messages` の 2 つのシステム変数値から作成されたディレクトリ内で検索され、後者は言語名に変換されます。次のコマンドを使用してサーバーを起動するとします。

```
mysqld --lc_messages_dir=/usr/share/mysql --lc_messages=fr_FR
```

この場合、`mysqld` は、ロケール `fr_FR` を言語 `french` にマップし、`/usr/share/mysql/french` ディレクトリでエラーファイルを検索します。

デフォルトでは、言語ファイルは、MySQL ベースディレクトリ下の `share/mysql/LANGUAGE` ディレクトリにあります。

- 前述のように構築されたディレクトリでメッセージファイルが見つからない場合、サーバーは、`lc_messages` 値を無視し、検索する場所として `lc_messages_dir` 値だけを使用します。
- 構成されたメッセージファイルが見つからない場合は、問題を示すメッセージがエラーログに書き込まれ、デフォルトで組み込み英語メッセージに設定されます。

`lc_messages_dir` システム変数は、サーバーの起動時にのみ設定でき、実行時にはグローバルな読み取り専用値のみを持ちます。`lc_messages` はサーバーの起動時に設定でき、実行時に変更できるグローバル値およびセッション値を持ちます。したがって、エラーメッセージの言語はサーバーの実行中に変更でき、各クライアントはセッション `lc_messages` 値を目的のロケール名に設定することで、独自のエラーメッセージの言語を持つことができます。たとえば、サーバーがエラーメッセージに `fr_FR` ロケールを使用している場合、クライアントは、次のステートメントを実行すると、英語でエラーメッセージを受信できます。

```
SET lc_messages = 'en_US';
```

10.13 文字セットの追加

このセクションでは、MySQL に文字セットを追加する手順について説明します。適切な手順は、文字セットが単純か複雑かによって異なります。

- ソートに特別な文字列照合ルーチンを必要とせず、マルチバイト文字のサポートを必要としない文字セットが、単純な文字セットです。
- これらのどちらかの機能が必要な文字セットが、複雑な文字セットです。

たとえば、`greek` と `swe7` は単純な文字セットですが、`big5` と `czech` は複雑な文字セットです。

次の手順を使用するには、MySQL ソース配布が必要です。この手順では、`MYSET` は追加する文字セットの名前を表します。

1. `MYSET` の `<charset>` 要素を `sql/share/charsets/Index.xml` ファイルに追加します。ファイル内既存の内容を、新しい内容を追加するためのガイドとして使用します。`latin1 <charset>` 要素のリストの一部を以下に示します。

```
<charset name="latin1">
  <family>Western</family>
  <description>cp1252 West European</description>
  ...
  <collation name="latin1_swedish_ci" id="8" order="Finnish, Swedish">
    <flag>primary</flag>
    <flag>compiled</flag>
  </collation>
  <collation name="latin1_danish_ci" id="15" order="Danish"/>
  ...
  <collation name="latin1_bin" id="47" order="Binary">
    <flag>binary</flag>
    <flag>compiled</flag>
  </collation>
  ...
</charset>
```

`<charset>` 要素は、文字セットのすべての照合順序を一覧表示します。これらには少なくとも、バイナリ照合順序とデフォルト (プライマリ) 照合順序が含まれます。多くの場合、デフォルトの照合には接尾辞 `general_ci` を使用して名前が付けられます (一般に、大/小文字は区別されません)。バイナリ照合順序をデフォルト照合順序にすることは可能ですが、通常、これらは異なります。デフォルト照合順序には `primary` フラグを付ける必要があります。バイナリ照合順序には `binary` フラグを付ける必要があります。

それぞれの照合順序に一意の ID 番号を割り当てる必要があります。1024 から 2047 の ID 範囲は、ユーザー定義の照合順序に予約されています。現在使用されている照合順序 ID の最大値を検索するには、次のクエリーを使用します。

```
SELECT MAX(ID) FROM INFORMATION_SCHEMA.COLLATIONS;
```

2. このステップは、追加しているのが単純な文字セットか、複雑な文字セットかにより異なります。単純な文字セットには、構成ファイルだけが必要がありますが、複雑な文字セットには、照合順序関数またはマルチバイト関数あるいはその両方を定義する C ソースファイルが必要です。

単純な文字セットの場合、文字セットプロパティーについて記した構成ファイル (`MYSET.xml`) を作成します。`sql/share/charsets` ディレクトリにこのファイルを作成します。このファイルの土台として `latin1.xml` のコピーを使用できます。ファイルの構文は非常に単純です。

- コメントは、通常の XML コメント (`<!-- text -->`) として記述されます。

- `<map>` 配列要素内の単語は、任意の数の空白によって区切られます。
- `<map>` 配列要素内の各単語は、16 進形式の数値で表す必要があります。
- `<ctype>` 要素の `<map>` 配列要素には 257 語が含まれます。そのあとのほかの `<map>` 配列要素には 256 語が含まれます。 [セクション10.13.1「文字定義配列」](#) を参照してください。
- `Index.xml` 内の文字セットに対して `<charset>` 要素に一覧表示された照合順序ごとに、文字の順序を定義する `<collation>` 要素を `MYSET.xml` に含める必要があります。

複雑な文字セットの場合、文字セットプロパティについて記述し、文字セットに対する演算を適切に実行するために必要なサポートルーチンを定義した C ソースファイルを作成します。

- `strings` ディレクトリに `ctype-MYSET.c` ファイルを作成します。既存の `ctype-*.c` ファイルのいずれか (`ctype-big5.c` など) を調べて、定義する必要のあるものを確認します。ファイル内の配列には、`ctype_MYSET`、`to_lower_MYSET` などの名前を付ける必要があります。これらは、単純な文字セットの配列に対応します。 [セクション10.13.1「文字定義配列」](#) を参照してください。
 - `Index.xml` 内の文字セットに対して `<charset>` 要素に一覧表示された `<collation>` 要素ごとに、`ctype-MYSET.c` ファイルが照合順序の実装を提供する必要があります。
 - 文字セットで文字列照合関数が必要な場合は、 [セクション10.13.2「複雑な文字セットの文字列照合のサポート」](#) を参照してください。
 - 文字セットでマルチバイト文字のサポートが必要な場合は、 [セクション10.13.3「複雑な文字セットのマルチバイト文字のサポート」](#) を参照してください。
3. 構成情報を変更します。 `MYSYS` の情報を追加するためのガイドとして、既存の構成情報を使用します。ここでの例では、文字セットにデフォルト照合順序とバイナリ照合順序があることを想定していますが、`MYSET` に追加の照合順序がある場合には、さらに多くの行が必要になります。
 - a. `mysys/charset-def.c` を編集し、新しい文字セットの照合順序を「登録」します。

「宣言」セクションに次の行を追加します。

```
#ifndef HAVE_CHARSET_MYSET
extern CHARSET_INFO my_charset_MYSET_general_ci;
extern CHARSET_INFO my_charset_MYSET_bin;
#endif
```

「登録」セクションに次の行を追加します。

```
#ifndef HAVE_CHARSET_MYSET
add_compiled_collation(&my_charset_MYSET_general_ci);
add_compiled_collation(&my_charset_MYSET_bin);
#endif
```

- b. 文字セットが `ctype-MYSET.c` を使用する場合、`strings/CMakeLists.txt` を編集して、`ctype-MYSET.c` を `STRINGS_SOURCES` 変数の定義に追加します。
- c. `cmake/character_sets.cmake` を編集します。
 - i. アルファベット順で `CHARSETS_AVAILABLE` の値に `MYSET` を追加します。
 - ii. アルファベット順で `CHARSETS_COMPLEX` の値に `MYSET` を追加します。これは、`CMake` が `-DDEFAULT_CHARSET=MYSET` を認識できるように、単純な文字セットにも必要です。

4. 再構成し、再コンパイルし、テストします。

10.13.1 文字定義配列

それぞれの単純な文字セットには、`sql/share/charsets` ディレクトリに置かれた構成ファイルがあります。 `MYSYS` という名前の文字セットの場合、ファイルには `MYSET.xml` の名前が付けられます。これは、`<map>` 配列要素を使用して、文字セットプロパティを一覧表示します。 `<map>` 要素は、これらの要素内に表示されます。

- `<ctype>` は文字ごとに属性を定義します。
- `<lower>` と `<upper>` は、小文字と大文字を一覧表示します。
- `<unicode>` は、8ビット文字値を Unicode 値にマップします。
- `<collation>` 要素は、照合順序ごとに1つの要素を比較およびソートするための文字順序を示します。文字コード自体が順序を提供するので、バイナリ照合順序には `<map>` 要素は不要です。

`strings` ディレクトリ内の `ctype-MYSET.c` ファイルに実装された複雑な文字セットには、`ctype_MYSET[]`、`to_lower_MYSET[]` などの対応する配列があります。すべての複雑な文字セットがすべての配列を持つわけではありません。例については、既存の `ctype-*.c` ファイルも参照してください。追加情報については、`strings` ディレクトリ内の `CHARSET_INFO.txt` ファイルを参照してください。

ほとんどの配列には、文字値でインデックスが付けられ、256個の要素があります。`<ctype>` 配列には、文字値 + 1 でインデックスが付けられ、257個の要素があります。これは、EOF を処理するための従来の規則です。

`<ctype>` 配列要素は、ビット値です。各要素は、文字セット内の単一の文字の属性について記述します。各属性は、`include/m_ctype.h` での定義に従って、ビットマスクに関連付けられます。

```
#define _MY_U 01 /* Upper case */
#define _MY_L 02 /* Lower case */
#define _MY_NMR 04 /* Numeral (digit) */
#define _MY_SPC 010 /* Spacing character */
#define _MY_PNT 020 /* Punctuation */
#define _MY_CTR 040 /* Control character */
#define _MY_B 0100 /* Blank */
#define _MY_X 0200 /* hexadecimal digit */
```

所定の文字の `<ctype>` 値は、その文字について記述した適用可能なビットマスク値の結合である必要があります。たとえば、`'A'` は 16進数 (`_MY_X`) と同様に大文字 (`_MY_U`) であるので、その `ctype` 値は次のように定義する必要があります。

```
ctype['A'+1] = _MY_U | _MY_X = 01 | 0200 = 0201
```

`m_ctype.h` のビットマスク値は 8進数値ですが、`MYSET.xml` 内の `<ctype>` 配列の要素は 16進値として書き込む必要があります。

`<lower>` と `<upper>` 配列は、文字セットの各メンバーに対応した小文字と大文字を保持します。例:

```
lower['A'] should contain 'a'
upper['a'] should contain 'A'
```

各 `<collation>` 配列は、比較およびソートでどのように文字を順序付けする必要があるかを示します。MySQL は、この情報の値に基づいて文字をソートします。場合によっては、これは `<upper>` 配列と同じであり、ソートで大文字と小文字が区別されないこととなります。さらに複雑なソートルールについては (複雑な文字セットの場合)、[セクション 10.13.2 「複雑な文字セットの文字列照合のサポート」](#) での文字列照合の説明を参照してください。

10.13.2 複雑な文字セットの文字列照合のサポート

`MYSET` という名前の単純な文字セットの場合、ソートルールは、`<collation>` 要素内の `<map>` 配列要素を使用して、`MYSET.xml` 構成ファイルで指定されます。言語に関するソートルールが非常に複雑で、単純な配列では扱えない場合、`strings` ディレクトリ内の `ctype-MYSET.c` ソースファイルで文字列照合関数を定義する必要があります。

既存の文字セットからは、これらの関数がどのように実装されているかを示す最適なドキュメントおよび例が得られます。`big5`、`czech`、`gbk`、`sjis`、`tis160` 文字セットのファイルなど、`strings` ディレクトリ内の `ctype-*.c` ファイルを調べます。`MY_COLLATION_HANDLER` 構造を見て、どのように使用されているかを確認します。追加情報については、`strings` ディレクトリ内の `CHARSET_INFO.txt` ファイルも参照してください。

10.13.3 複雑な文字セットのマルチバイト文字のサポート

マルチバイト文字を含む `MYSET` という名前の新しい文字セットのサポートを追加する場合、`strings` ディレクトリ内の `ctype-MYSET.c` ソースファイルのマルチバイト文字関数を使用する必要があります。

既存の文字セットからは、これらの関数がどのように実装されているかを示す最適なドキュメントおよび例が得られます。 `euc_kr`、`gb2312`、`gbk`、`sjis`、`ujis` 文字セットのファイルなど、`strings` ディレクトリ内の `ctype-*.c` ファイルを調べます。 `MY_CHARSET_HANDLER` 構造を見て、どのように使用されているかを確認します。追加情報については、`strings` ディレクトリ内の `CHARSET_INFO.txt` ファイルも参照してください。

10.14 文字セットへの照合順序の追加

照合順序は、文字列を比較およびソートする方法を定義した一連のルールです。MySQL でのそれぞれの照合順序は、単一の文字セットに属しています。すべての文字セットには少なくとも 1 つの照合順序が属し、ほとんどの文字セットのは 2 つ以上の照合順序が属しています。

照合順序は重みに基づいて文字を順序付けします。文字セット内のそれぞれの文字が重みにマップされています。重みが等しい文字は同等と見なされ、重みが等しくない文字は、その重みの相対的な大きさに従って比較されます。

`WEIGHT_STRING()` 関数を使用すると、文字列内の文字の重みを確認できます。重みを示した返される値はバイナリ文字列であるので、`HEX(WEIGHT_STRING(str))` を使用して重みを出力可能な形式で表示すると便利です。次の例は、大文字と小文字を区別しない非バイナリ文字列の場合に、`'AaBb'` の文字の大文字と小文字で重みが異なることなく、バイナリ文字列の場合に重みが異なることを示しています：

```
mysql> SELECT HEX(WEIGHT_STRING('AaBb' COLLATE latin1_swedish_ci));
+-----+
| HEX(WEIGHT_STRING('AaBb' COLLATE latin1_swedish_ci)) |
+-----+
| 41414242                |
+-----+
mysql> SELECT HEX(WEIGHT_STRING(BINARY 'AaBb'));
+-----+
| HEX(WEIGHT_STRING(BINARY 'AaBb')) |
+-----+
| 41614262                |
+-----+
```

セクション10.14.1「照合順序の実装タイプ」で説明するように、MySQL では複数の照合順序の実装をサポートしています。これらの中には、再コンパイルせずに、MySQL に追加できるものもあります。

- 8 ビットの文字セットの単純な照合順序。
- Unicode 文字セットの UCA ベースの照合順序。
- バイナリ (`xxx_bin`) 照合順序。

次のセクションでは、最初の 2 つのタイプのユーザー定義照合を既存の文字セットに追加する方法について説明します。バイナリ照合順序については、既存の文字セットにもすでに用意されているので、ここでは追加方法は説明しません。

警告

組込み照合の再定義はサポートされていないため、予期しないサーバー動作が発生する可能性があります。

新しいユーザー定義照合を追加する手順のサマリー：

1. 照合順序 ID を選択します。
2. 照合順序に名前を付ける構成情報を追加し、文字順序付けルールについて記述します。
3. サーバーを再起動します。
4. サーバーが照合順序を認識していることを確認します。

この手順では、MySQL を再コンパイルせずに追加できるユーザー定義の照合のみについて説明します。再コンパイルを必要とする照合順序 (C ソースファイル内の関数を利用して実装されたものなど) を追加するには、[セクション 10.13 「文字セットの追加」](#) の手順を使用してください。ただし、完全な文字セットに必要なすべての情報を追加す

るのではなく、既存の文字セットに合わせて適切なファイルを変更します。つまり、文字セットの現在の照合順序ですでに存在するものに基づいて、新しい照合順序のデータ構造、関数、構成情報を追加します。

注記

既存のユーザー定義照合を変更すると、その照合を使用するカラムのインデックスの行の順序に影響する可能性があります。この場合、間違ったクエリー結果などの問題が起こらないように、これらのインデックスを再構築してください。 [セクション2.11.13「テーブルまたはインデックスの再作成または修復」](#)を参照してください。

追加のリソース

- 全文検索の照合順序を追加する方法を示す例: [セクション12.10.7「全文インデックス付けのためのユーザー定義照合の追加」](#)
- Unicode 照合順序アルゴリズム (UCA) の仕様: <http://www.unicode.org/reports/tr10/>
- Locale Data Markup Language (LDML) の仕様: <http://www.unicode.org/reports/tr35/>

10.14.1 照合順序の実装タイプ

MySQL は複数のタイプの照合順序を実装します。

8 ビットの文字セットに対する単純な照合順序

この種の照合順序は、文字コードと重みの 1 対 1 のマッピングを定義した 256 個の重みの配列を使用して実装されます。 [latin1_swedish_ci](#) がその一例です。これは、大文字と小文字を区別しない照合順序なので、大文字と小文字は同じ重みで、等しいものと見なされます。

```
mysql> SET NAMES 'latin1' COLLATE 'latin1_swedish_ci';
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT HEX(WEIGHT_STRING('a')), HEX(WEIGHT_STRING('A'));
+-----+-----+
| HEX(WEIGHT_STRING('a')) | HEX(WEIGHT_STRING('A')) |
+-----+-----+
| 41          | 41          |
+-----+-----+
1 row in set (0.01 sec)

mysql> SELECT 'a' = 'A';
+-----+
| 'a' = 'A' |
+-----+
|          1 |
+-----+
1 row in set (0.12 sec)
```

実装の手順については、 [セクション10.14.3「8 ビットの文字セットへの単純な照合順序の追加」](#) を参照してください。

8 ビット文字セットに対する複雑な照合順序

この種の照合順序は、 [セクション10.13「文字セットの追加」](#) で説明しているように、文字を順序付けする方法を定義した C ソースファイル内の関数を使用して実装されます。

Unicode 以外のマルチバイト文字セットの照合順序

この種の照合順序では、8 ビット (シングルバイト) 文字とマルチバイト文字が異なる方法で処理されます。8 ビットの文字の場合、文字コードは大文字と小文字を区別しない形式で重みにマップされます。(たとえば、シングルバイト文字 'a' と 'A' はどちらも重みが `0x41` です。) マルチバイト文字の場合、文字コードと重みの間には、2 種類の関係があります。

- 重みが文字コードと等しい場合。 [sjis_japanese_ci](#) がこの種の照合順序の一例です。マルチバイト文字 'ぢ' の文字コードは `0x82C0` であり、重みも `0x82C0` です。

```
mysql> CREATE TABLE t1
(c1 VARCHAR(2) CHARACTER SET sjis COLLATE sjis_japanese_ci);
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> INSERT INTO t1 VALUES ('a'),('A'),('ぢ');
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0
```

```
mysql> SELECT c1, HEX(c1), HEX(WEIGHT_STRING(c1)) FROM t1;
+-----+-----+-----+
| c1 | HEX(c1) | HEX(WEIGHT_STRING(c1)) |
+-----+-----+-----+
| a | 61 | 41 |
| A | 41 | 41 |
| ぢ | 82C0 | 82C0 |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

- 文字コードが 1 対 1 で重みにマップされていても、必ずしもコードと重みが等しくない場合。gbk_chinese_ci がこの種の照合順序の一例です。マルチバイト文字 '臙' の文字コードは 0x81B0 ですが、重みは 0xC286 です。

```
mysql> CREATE TABLE t1
(c1 VARCHAR(2) CHARACTER SET gbk COLLATE gbk_chinese_ci);
Query OK, 0 rows affected (0.33 sec)
```

```
mysql> INSERT INTO t1 VALUES ('a'),('A'),('臙');
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0
```

```
mysql> SELECT c1, HEX(c1), HEX(WEIGHT_STRING(c1)) FROM t1;
+-----+-----+-----+
| c1 | HEX(c1) | HEX(WEIGHT_STRING(c1)) |
+-----+-----+-----+
| a | 61 | 41 |
| A | 41 | 41 |
| 臙 | 81B0 | C286 |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

実装の手順については、[セクション10.13「文字セットの追加」](#)を参照してください。

Unicode マルチバイト文字セットの照合順序

これらの照合順序の一部は Unicode 照合順序アルゴリズム (UCA) に基づきますが、それ以外は基づいていません。

UCA に基づいていない照合順序では、文字コードと重みは 1 対 1 でマップしています。MySQL では、このような照合順序では大/小文字が区別されず、アクセントは区別されません。utf8_general_ci は例です: 'a', 'A', 'À' と 'á' はそれぞれ異なる文字コードを持ちますが、すべての重みは 0x0041 であり、等しいものとして比較されます。

```
mysql> SET NAMES 'utf8' COLLATE 'utf8_general_ci';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> CREATE TABLE t1
(c1 CHAR(1) CHARACTER SET UTF8 COLLATE utf8_general_ci);
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> INSERT INTO t1 VALUES ('a'),('A'),('À'),('á');
Query OK, 4 rows affected (0.00 sec)
Records: 4 Duplicates: 0 Warnings: 0
```

```
mysql> SELECT c1, HEX(c1), HEX(WEIGHT_STRING(c1)) FROM t1;
+-----+-----+-----+
| c1 | HEX(c1) | HEX(WEIGHT_STRING(c1)) |
+-----+-----+-----+
| a | 61 | 0041 |
| A | 41 | 0041 |
| À | C380 | 0041 |
| á | C3A1 | 0041 |
+-----+-----+-----+
4 rows in set (0.00 sec)
```

MySQL の UCA ベースの照合順序には、次の 3 つのプロパティがあります。

- 文字に重みがある場合、それぞれの重みは 2 バイト (16 ビット) を使用します。
- 文字の重みはゼロ (または空の重み) の場合があります。この場合、文字は無視できます。例: 「U+0000 NULL」は重みがなく、無視できます。
- 1 つの文字が 1 つの重みを持つ場合があります。例: 'a' には 0x0E33 の重みがあります。

```
mysql> SET NAMES 'utf8' COLLATE 'utf8_unicode_ci';
Query OK, 0 rows affected (0.05 sec)

mysql> SELECT HEX('a'), HEX(WEIGHT_STRING('a'));
+-----+-----+
| HEX('a') | HEX(WEIGHT_STRING('a')) |
+-----+-----+
| 61      | 0E33                    |
+-----+-----+
1 row in set (0.02 sec)
```

- 1 つの文字が複数の重みを持つ場合があります。これは拡張形式です。例: ドイツ語の文字 'ß' (SZ リガチャーまたは SHARP S) には、0x0FEA0FEA の重みがあります。

```
mysql> SET NAMES 'utf8' COLLATE 'utf8_unicode_ci';
Query OK, 0 rows affected (0.11 sec)

mysql> SELECT HEX('ß'), HEX(WEIGHT_STRING('ß'));
+-----+-----+
| HEX('ß') | HEX(WEIGHT_STRING('ß')) |
+-----+-----+
| C39F    | 0FEA0FEA                |
+-----+-----+
1 row in set (0.00 sec)
```

- 複数の文字が 1 つの重みを持つ場合があります。これは短縮形式です。例: 'ch' は、チェコ語の単一の文字であり、0x0EE2 の重みを持ちます。

```
mysql> SET NAMES 'utf8' COLLATE 'utf8_czech_ci';
Query OK, 0 rows affected (0.09 sec)

mysql> SELECT HEX('ch'), HEX(WEIGHT_STRING('ch'));
+-----+-----+
| HEX('ch') | HEX(WEIGHT_STRING('ch')) |
+-----+-----+
| 6368     | 0EE2                    |
+-----+-----+
1 row in set (0.00 sec)
```

複数の文字と複数の重みのマッピングも可能です (これは拡張形式を使用した短縮形式です) が、MySQL ではサポートされていません。

UCA に基づいていない照合順序に関する実装の手順については、[セクション10.13「文字セットの追加」](#)を参照してください。UCA 照合順序については、[セクション10.14.4「Unicode 文字セットへの UCA 照合順序の追加」](#)を参照してください。

その他の照合順序

上記のカテゴリのどれにも該当しない照合順序も少数存在します。

10.14.2 照合順序 ID の選択

各照合順序には一意の ID が必要です。照合順序を追加するには、現在使用されていない ID 値を選択する必要があります。MySQL は、2 バイトの照合 ID をサポートしています。1024 から 2047 の ID 範囲は、ユーザー定義の照合順序に予約されています。

選択した照合 ID は、次のコンテキストで表示されます:

- `INFORMATION_SCHEMA.COLLATIONS` テーブルの ID カラム。
- `SHOW COLLATION` 出力の Id カラム。

- `MYSQL_FIELD` C API データ構造の `charsetnr` メンバー。
- `mysql_get_character_set_info()` C API 関数で返される `MY_CHARSET_INFO` データ構造の `number` メンバー。

現在使用されている最大の ID を判別するには、次のステートメントを発行します。

```
mysql> SELECT MAX(ID) FROM INFORMATION_SCHEMA.COLLATIONS;
+-----+
| MAX(ID) |
+-----+
| 247 |
+-----+
```

現在使用されているすべての ID のリストを表示するには、次のステートメントを発行します。

```
mysql> SELECT ID FROM INFORMATION_SCHEMA.COLLATIONS ORDER BY ID;
+----+
| ID |
+----+
| 1 |
| 2 |
| ... |
| 52 |
| 53 |
| 57 |
| 58 |
| ... |
| 98 |
| 99 |
| 128 |
| 129 |
| ... |
| 247 |
+----+
```

警告

アップグレードする前に、変更した構成ファイルを保存する必要があります。適切にアップグレードすると、変更されたファイルがプロセスによって置き換えられます。

10.14.3 8ビットの文字セットへの単純な照合順序の追加

このセクションでは、MySQL `Index.xml` ファイル内の `<charset>` 文字セットの記述に関連付けられた `<collation>` 要素を書き込むことによって、8ビット文字セットの単純な照合順序を追加する方法について説明します。ここで説明した手順では、MySQL の再コンパイルは不要です。この例では、`latin1_test_ci` という名前の照合順序を `latin1` 文字セットに追加します。

1. [セクション10.14.2「照合順序 ID の選択」](#) で示したように、照合順序 ID を選択します。次のステップでは、1024 の ID を使用します。
2. `Index.xml` および `latin1.xml` 構成ファイルを変更します。これらのファイルは、`character_sets_dir` システム変数で指定されたディレクトリにあります。使用しているシステムではパス名が異なることがありますが、次のようにして変数値を確認できます。

```
mysql> SHOW VARIABLES LIKE 'character_sets_dir';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| character_sets_dir | /user/local/mysql/share/mysql/charsets/ |
+-----+-----+
```

3. 照合順序の名前を選択して、`Index.xml` ファイルに表示します。照合順序を追加する文字セットの `<charset>` 要素を探し、照合順序名および ID を指定する `<collation>` 要素を追加して、名前を ID に関連付けます。例:

```
<charset name="latin1">
...
<collation name="latin1_test_ci" id="1024"/>
...
```

```
</charset>
```

4. `latin1.xml` 構成ファイルで、照合順序に名前を付ける `<collation>` 要素と、0 から 255 の文字コードの文字コードと重みのマッピングテーブルを定義する `<map>` 要素を追加します。 `<map>` 要素内のそれぞれの値は、16 進形式の数値にする必要があります。

```
<collation name="latin1_test_ci">
<map>
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F
60 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
50 51 52 53 54 55 56 57 58 59 5A 7B 7C 7D 7E 7F
80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F
90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF
41 41 41 41 5B 5D 5B 43 45 45 45 45 49 49 49 49
44 4E 4F 4F 4F 4F 5C D7 5C 55 55 55 59 59 DE DF
41 41 41 41 5B 5D 5B 43 45 45 45 45 49 49 49 49
44 4E 4F 4F 4F 4F 5C F7 5C 55 55 55 59 59 DE FF
</map>
</collation>
```

5. サーバーを再起動し、このステートメントを使用して、照合順序の有無を検証します。

```
mysql> SHOW COLLATION WHERE Collation = 'latin1_test_ci';
+-----+-----+-----+-----+-----+
| Collation | Charset | Id | Default | Compiled | Sortlen |
+-----+-----+-----+-----+-----+
| latin1_test_ci | latin1 | 1024 | | | 1 |
+-----+-----+-----+-----+-----+
```

10.14.4 Unicode 文字セットへの UCA 照合順序の追加

このセクションでは、MySQL `Index.xml` ファイルで `<charset>` 文字セットの記述内に `<collation>` 要素を書き込むことによって、Unicode 文字セットの UCA 照合順序を追加する方法について説明します。ここで説明した手順では、MySQL の再コンパイルは不要です。これは、<http://www.unicode.org/reports/tr35/> で入手できる Locale Data Markup Language (LDML) 仕様のサブセットを使用します。この方法を使用すれば、照合順序全体を定義する必要はありません。代わりに、既存の「基本」照合順序から始め、基本照合順序とどのように異なるかという点で新しい照合順序について記述します。次の表は、UCA 照合順序を定義できる Unicode 文字セットの基本照合順序を一覧表示しています。 `utf16le` のユーザー定義 UCA 照合順序は作成できません。このような照合順序のベースとして役立つ `utf16le_unicode_ci` 照合順序はありません。

表 10.4 ユーザー定義の UCA 照合順序に使用可能な MySQL 文字セット

文字セット	基本照合順序
<code>utf8</code>	<code>utf8_unicode_ci</code>
<code>ucs2</code>	<code>ucs2_unicode_ci</code>
<code>utf16</code>	<code>utf16_unicode_ci</code>
<code>utf32</code>	<code>utf32_unicode_ci</code>

以降のセクションでは、LDML 構文を使用して定義された照合順序を追加する方法について説明し、MySQL でサポートされている LDML ルールのサマリーを示します。

10.14.4.1 LDML 構文を使用した UCA 照合順序の定義

MySQL を再コンパイルせずに Unicode 文字セットの UCA 照合順序を追加するには、次の手順を使用します。照合順序のソート特性の記述に使用する LDML ルールを把握していない場合は、[セクション10.14.4.2「MySQL でサポートされる LDML 構文」](#)を参照してください。

この例では、`utf8_phone_ci` という名前の照合順序を `utf8` 文字セットを追加します。この照合順序は、ユーザーが名前と電話番号を投稿する Web アプリケーションに関連したシナリオ用に設計されています。電話番号は、次のようなさまざまな形式で指定できます。

```
+7-12345-67
+7-12-345-67
+7 12 345 67
+7 (12) 345 67
+71234567
```

このような値を扱うときに生じる問題は、さまざまな形式が許容されることで、特定の電話番号の検索が非常に困難になるということです。この解決方法として、句読点文字を並べ替えて無視できるようにする新しい照合順序を定義します。

1. [セクション10.14.2「照合順序 ID の選択」](#) で示したように、照合順序 ID を選択します。次のステップでは、1029 の ID を使用します。
2. `Index.xml` 構成ファイルを変更します。このファイルは、`character_sets_dir` システム変数で指定されたディレクトリにあります。使用しているシステムではパス名が異なる場合がありますが、次のようにして変数値を確認できます。

```
mysql> SHOW VARIABLES LIKE 'character_sets_dir';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| character_sets_dir | /user/local/mysql/share/mysql/charsets/ |
+-----+-----+
```

3. 照合順序の名前を選択して、`Index.xml` ファイルに表示します。さらに、照合順序の順序付けルールを提供する必要があります。照合順序を追加する文字セットの `<charset>` 要素を探し、照合順序名および ID を指定する `<collation>` 要素を追加して、名前を ID に関連付けます。 `<collation>` 要素内で、順序付けルールを含む `<rules>` 要素を提供します。

```
<charset name="utf8">
...
<collation name="utf8_phone_ci" id="1029">
  <rules>
    <reset>\u0000</reset>
    <i>\u0020</i> <!-- space -->
    <i>\u0028</i> <!-- left parenthesis -->
    <i>\u0029</i> <!-- right parenthesis -->
    <i>\u002B</i> <!-- plus -->
    <i>\u002D</i> <!-- hyphen -->
  </rules>
</collation>
...
</charset>
```

4. その他の Unicode 文字セットに同様の照合順序が必要な場合は、ほかの `<collation>` 要素を追加します。たとえば、`ucs2_phone_ci` を定義するには、`<collation>` 要素を `<charset name="ucs2">` 要素に追加します。それぞれの照合順序には一意の独自 ID が必要になります。
5. サーバーを再起動し、このステートメントを使用して、照合順序の有無を検証します。

```
mysql> SHOW COLLATION WHERE Collation = 'utf8_phone_ci';
+-----+-----+-----+-----+-----+
| Collation | Charset | Id | Default | Compiled | Sortlen |
+-----+-----+-----+-----+-----+
| utf8_phone_ci | utf8 | 1029 | | | 8 |
+-----+-----+-----+-----+-----+
```

次に、照合順序をテストして、目的のプロパティがあることを確認します。

新しい照合順序を使用して、いくつかのサンプルの電話番号を含むテーブルを作成します。

```
mysql> CREATE TABLE phonebook (
  name VARCHAR(64),
  phone VARCHAR(64) CHARACTER SET utf8 COLLATE utf8_phone_ci
);
```



```
Query OK, 0 rows affected (0.09 sec)

mysql> INSERT INTO phonebook VALUES ('Svoj','+7 912 800 80 02');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO phonebook VALUES ('Hf','+7 (912) 800 80 04');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO phonebook VALUES ('Bar','+7-912-800-80-01');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO phonebook VALUES ('Ramil','(7912) 800 80 03');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO phonebook VALUES ('Sanja','+380 (912) 8008005');
Query OK, 1 row affected (0.00 sec)
```

一部のクエリーを実行して、無視された句読点文字が実際に比較およびソートで無視されているかどうかを確認します:

```
mysql> SELECT * FROM phonebook ORDER BY phone;
+----+-----+
| name | phone          |
+----+-----+
| Sanja | +380 (912) 8008005 |
| Bar   | +7-912-800-80-01 |
| Svoj  | +7 912 800 80 02 |
| Ramil | (7912) 800 80 03 |
| Hf    | +7 (912) 800 80 04 |
+----+-----+
5 rows in set (0.00 sec)

mysql> SELECT * FROM phonebook WHERE phone='+7(912)800-80-01';
+----+-----+
| name | phone          |
+----+-----+
| Bar   | +7-912-800-80-01 |
+----+-----+
1 row in set (0.00 sec)

mysql> SELECT * FROM phonebook WHERE phone='79128008001';
+----+-----+
| name | phone          |
+----+-----+
| Bar   | +7-912-800-80-01 |
+----+-----+
1 row in set (0.00 sec)

mysql> SELECT * FROM phonebook WHERE phone='7 9 1 2 8 0 0 8 0 0 1';
+----+-----+
| name | phone          |
+----+-----+
| Bar   | +7-912-800-80-01 |
+----+-----+
1 row in set (0.00 sec)
```

10.14.4.2 MySQL でサポートされる LDML 構文

このセクションでは、MySQL が認識する LDML 構文について説明します。これは、<http://www.unicode.org/reports/tr35/> で入手できる LDML 仕様で説明されている構文のサブセットであり、詳細についてはこの仕様を参照してください。MySQL は、サイズの大きい構文のサブセットを認識するので、多くの場合、Unicode 共通ロケールデータリポジトリから照合順序定義をダウンロードして、該当する部分 (<rules> タグと </rules> タグの間の部分) を MySQL `Index.xml` ファイルに貼り付けることが可能です。ここで説明するルールは、文字のソートがプライマリレベルのみで行われることを除いて、すべてサポートされます。セカンダリ以上のソートレベルでの差を指定するルールは認識されますが (たとえば、照合順序の定義に含めることができます)、プライマリレベルでは等式として扱われます。

MySQL Server は、`Index.xml` ファイルの構文解析中に問題を検出すると、診断を生成します。 [セクション 10.14.4.3 「Index.xml の構文解析中の診断」](#) を参照してください。

文字表現

LDML ルールで名前が付けられた文字は、文字どおりに、または `\unnnn` 形式で書き込めます。ただし、`nnnn` は 16 進 Unicode コードポイント値です。たとえば、`A` と `á` は、文字どおりに、または `\u0041` および `\u00E1` として書き込めます。16 進数の値内では、`A` から `F` までの桁は大/小文字が区別されません。`\u00E1` と `\u00e1` は同等です。UCA 4.0.0 照合順序の場合、16 進表記は、Basic Multilingual Plane の文字にのみ使用でき、`0000` から `FFFF` の BMP 範囲外の文字には使用できません。UCA 5.2.0 照合順序の場合、16 進表記をすべての文字に使用できます。

`Index.xml` ファイル自体は、UTF-8 エンコーディングを使用して書き込む必要があります。

構文ルール

LDML には、文字の順序付けを指定するリセットルールとシフトルールがあります。順序付けは、アンカーポイントを確認するリセットルールから開始し、そのアンカーポイントを基準として文字をソートする方法を示すシフトルールが続く一連のルールとして指定されます。

- `<reset>` ルールはそれ自体ではどの順序付けも指定しません。その代わりにこれは、所定の文字に関連して後続のシフトルールを実行できるように、順序付けを「リセット」します。次のどちらのルールも、文字 '`A`' に関連して実行されるように後続のシフトルールをリセットします。

```
<reset>A</reset>
<reset>\u0041</reset>
```

- `<p>`、`<s>`、および `<t>` のシフトルールは、文字と文字とのプライマリ、セカンダリ、およびターシャリの差を定義します。

- プライマリの差を使用して、個々の文字を区別します。
- セカンダリの差を使用して、アクセントバリエーションを区別します。
- ターシャリの差を使用して、大文字と小文字のバリエーションを区別します。

次のどちらのルールも、'`G`' 文字のプライマリシフトルールを指定します。

```
<p>G</p>
<p>\u0047</p>
```

- `<i>` シフトルールは、ある文字が別の文字とまったく同じようにソートするよう指定します。次のルールは、'`b`' が '`a`' と同じようにソートします。

```
<reset>a</reset>
<i>b</i>
```

- 略記されたシフト構文は、タグの単一のペアを使用して、複数のシフトルールを指定します。次の表は、略記された構文ルールと同等の略記されていないルールとの対応を示します。

表 10.5 略記されたシフト構文

略記された構文	略記されていない構文
<code><pc>xyz</pc></code>	<code><p>x</p><p>y</p><p>z</p></code>
<code><sc>xyz</sc></code>	<code><s>x</s><s>y</s><s>z</s></code>
<code><tc>xyz</tc></code>	<code><t>x</t><t>y</t><t>z</t></code>
<code><ic>xyz</ic></code>	<code><i>x</i><i>y</i><i>z</i></code>

- 拡張形式は、複数文字のシーケンスのアンカーポイントを確認するリセットルールです。MySQL は 2 から 6 文字長の拡張形式をサポートしています。次のルールは、プライマリレベルで '`z`' を 3 文字のシーケンス '`abc`' よりも大きくします。

```
<reset>abc</reset>
<p>z</p>
```

- 短縮形式は、複数文字のシーケンスをソートするシフトルールです。MySQL は、2 から 6 文字長の短縮形式をサポートしています。次のルールは、プライマリレベルで 3 文字のシーケンス '`xyz`' を '`a`' よりも大きくします。

```
<reset>a</reset>
<p>xyz</p>
```

- 長い拡張形式と長い短縮形式を一緒に使用できます。次のルールは、プライマリレベルで 3 文字のシーケンス 'xyz' を 3 文字のシーケンス 'abc' よりも大きくします。

```
<reset>abc</reset>
<p>xyz</p>
```

- 通常の拡張形式の構文では、`<x>` と `<extend>` 要素を使用して、拡張形式を指定します。次のルールは、セカンダリレベルで文字 'k' をシーケンス 'ch' よりも大きくします。つまり、'k' は、'c' に 'h' が続いたあとの文字に拡張したかのように動作します。

```
<reset>c</reset>
<x><s>k</s><extend>h</extend></x>
```

この構文は長いシーケンスを許可します。次のルールは、ターシャリレベルでシーケンス 'ccs' をシーケンス 'cscs' よりも大きくします。

```
<reset>cs</reset>
<x><t>ccs</t><extend>cs</extend></x>
```

LDML 仕様では、通常の拡張形式構文を「慎重を要するもの」と記述しています。詳細についてはその仕様を参照してください。

- 前コンテキスト構文は、`<x>` および `<context>` 要素を使用して、文字の前のコンテキストによってソートが変更されるよう指定します。次のルールは、セカンダリレベルで 'l' を 'a' よりも大きくしますが、これは 'l' の前に 'b' があつたときだけです。

```
<reset>a</reset>
<x><context>b</context><s>-</s></x>
```

- 前コンテキスト構文には、`<extend>` 要素を含めることができます。次のルールは、プライマリレベルで 'def' を 'aghi' よりも大きくしますが、これは 'def' の前に 'abc' があつたときだけです。

```
<reset>a</reset>
<x><context>abc</context><p>def</p><extend>ghi</extend></x>
```

- リセットルールでは `before` 属性が許可されています。通常、リセットルールのあとのシフトルールは、リセット文字のあとにソートする文字を指定します。 `before` 属性を伴うリセットルール後のシフトルールは、リセット文字の前にソートする文字を指定します。次のルールは、プライマリレベルで文字 'b' を 'a' の直前に配置します。

```
<reset before="primary">a</reset>
<p>b</p>
```

許容されている `before` 属性値は、名前または同等な数値でソートレベルを指定します。

```
<reset before="primary">
<reset before="1">
```

```
<reset before="secondary">
<reset before="2">
```

```
<reset before="tertiary">
<reset before="3">
```

- リセットルールでは、リテラル文字ではなく、論理リセット位置に名前を付けることができます。

```
<first_tertiary_ignorable/>
<last_tertiary_ignorable/>
<first_secondary_ignorable/>
<last_secondary_ignorable/>
<first_primary_ignorable/>
<last_primary_ignorable/>
<first_variable/>
<last_variable/>
<first_non_ignorable/>
<last_non_ignorable/>
<first_trailing/>
```

```
<last_trailing/>
```

次のルールは、プライマリレベルで 'z' を、Default Unicode Collation Element Table (DUCET) エントリを伴い、CJK ではない無視できない文字よりも大きくします。

```
<reset><last_non_ignorable/></reset>  
<p>z</p>
```

論理位置には、次の表に示すコードポイントが設定されています。

表 10.6 論理リセット位置のコードポイント

論理位置	Unicode 4.0.0 コードポイント	Unicode 5.2.0 コードポイント
<first_non_ignorable/>	U+02D0	U+02D0
<last_non_ignorable/>	U+A48C	U+1342E
<first_primary_ignorable/>	U+0332	U+0332
<last_primary_ignorable/>	U+20EA	U+101FD
<first_secondary_ignorable/>	U+0000	U+0000
<last_secondary_ignorable/>	U+FE73	U+FE73
<first_tertiary_ignorable/>	U+0000	U+0000
<last_tertiary_ignorable/>	U+FE73	U+FE73
<first_trailing/>	U+0000	U+0000
<last_trailing/>	U+0000	U+0000
<first_variable/>	U+0009	U+0009
<last_variable/>	U+2183	U+1D371

- **<collation>** 要素は、シフトルールの文字重み計算に影響する **shift-after-method** 属性を許可します。この属性には、次の値が許可されています。
 - **simple: before** 属性を持たないリセットルールで、文字の重みを計算します。これは、属性が指定されない場合のデフォルトです。
 - **expand**: リセットルールのあとのシフトに拡張形式を使用します。

'0' と '1' が 0E29 と 0E2A の重みを持ち、すべての基本ラテン文字を '0' から '1' の間に設定するとします。

```
<reset>0</reset>  
<pc>abcdefghijklmnopqrstuvwxyz</pc>
```

単純なシフトモードの場合、重みは次のように計算されます。

```
'a' has weight 0E29+1  
'b' has weight 0E29+2  
'c' has weight 0E29+3  
...
```

ただし、'0' から '1' の間には 26 個の文字を設定するのに十分な未使用の位置がありません。数字と文字が混在する結果になります。

これを解決するには **shift-after-method="expand"** を使用します。この場合、重みは次のように計算されます。

```
'a' has weight [0E29][233D+1]  
'b' has weight [0E29][233D+2]  
'c' has weight [0E29][233D+3]  
...
```

233D は、文字 0xA48C の UCA 4.0.0 重みです。これは、最後の無視できない文字 (CJK を除き、照合順序で一種のもっとも大きな文字) です。UCA 5.2.0 も同様ですが、文字 0x1342E に 3ACA を使用します。

MySQL 固有の LDML 拡張機能

LDML ルールの拡張により、`<collation>` 要素では、照合のベースとなるUCAバージョンを示すオプションの `version` 属性を `<collation>` タグに含めることができます。 `version` 属性を省略すると、そのデフォルト値は `4.0.0` になります。たとえば、次の指定は、UCA 5.2.0 に基づいている照合順序を示します。

```
<collation id="nnn" name="utf8_xxx_ci" version="5.2.0">
...
</collation>
```

10.14.4.3 Index.xml の構文解析中の診断

MySQL Server は、`Index.xml` ファイルの構文解析中に問題が見つかったときに、診断を生成します。

- 不明なタグはエラーログに書き込まれます。たとえば、照合順序定義に `<aaa>` タグが含まれる場合、次のメッセージが表示されます。

```
[Warning] Buffered warning: Unknown LDML tag:
'charsets/charset/collation/rules/aaa'
```

- 照合順序の初期化が可能でない場合、サーバーは「不明な照合順序」エラーをレポートし、前の例のように問題点を説明した警告も生成します。それ以外の場合、照合順序の説明が全体的に正しいが、いくつかの不明なタグが含まれているときに、照合順序が初期化され、使用できません。不明な部分は無視されますが、警告がエラーログに生成されます。
- 照合順序の問題によって、クライアントが `SHOW WARNINGS` で表示できる警告が生成されます。6文字のサポートされる最大長より長い拡張形式が、リセットルールに含まれているとします。

```
<reset>abcdefghi</reset>
<i>x</i>
```

この照合順序を使用しようとすると、次の警告が生成されます。

```
mysql> SELECT _utf8'test' COLLATE utf8_test_ci;
ERROR 1273 (HY000): Unknown collation: 'utf8_test_ci'
mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message                                     |
+-----+-----+-----+
| Error | 1273 | Unknown collation: 'utf8_test_ci'         |
| Warning | 1273 | Expansion is too long at 'abcdefghi=x'    |
+-----+-----+-----+
```

10.15 文字セットの構成

MySQL サーバーには、コンパイルされたデフォルトの文字セットと照合順序があります。これらのデフォルトを変更するには、サーバーの起動時に `--character-set-server` および `--collation-server` オプションを使用します。 [セクション5.1.7「サーバーコマンドオプション」](#) を参照してください。照合順序は、デフォルト文字セットの正当な照合順序である必要があります。各文字セットで使用可能な照合順序を確認するには、`SHOW COLLATION` ステートメントを使用するか、`INFORMATION_SCHEMA COLLATIONS` テーブルをクエリーします。

バイナリにコンパイルされない文字セットを使用しようとすると、次の問題が生じることがあります。

- プログラムで誤ったパスを使用して文字セットが格納されている場所 (通常は、MySQL インストールディレクトリの下 `share/mysql/charsets` または `share/charsets` ディレクトリ) を判別する場合は、プログラムの実行時に `--character-sets-dir` オプションを使用して修正できます。たとえば、MySQL クライアントプログラムで使用されるディレクトリを指定するには、オプションファイルの `[client]` グループに記述します。ここに挙げる例は、それぞれ Unix または Windows の場合に設定がどのようになるかを示します。

```
[client]
character-sets-dir=/usr/local/mysql/share/mysql/charsets
```

```
[client]
character-sets-dir="C:/Program Files/MySQL/MySQL Server 8.0/share/charsets"
```

- 文字セットが動的にロードできない複雑な文字セットの場合は、文字セットをサポートしてプログラムを再コンパイルする必要があります。

Unicode 文字セットの場合、LDML 表記を使用することによって、再コンパイルせずに照合順序を定義できます。[セクション10.14.4「Unicode 文字セットへの UCA 照合順序の追加」](#)を参照してください。

- 文字セットが動的な文字セットであるが、その構成ファイルがない場合は、新しい MySQL ディストリビューションから文字セットの構成ファイルをインストールする必要があります。
- 文字セットインデックスファイル (`index.xml`) に文字セットの名前が含まれていない場合は、次のエラーメッセージが表示されます:

```
Character set 'charset_name' is not a compiled character set and is not
specified in the '/usr/share/mysql/charsets/Index.xml' file
```

この問題を解決するには、新しいインデックスファイルを取得するか、欠落している文字セットの名前を手動で現在のファイルに追加する必要があります。

次のようにしてクライアントプログラムに強制的に特定の文字セットを使用させることができます。

```
[client]
default-character-set=charset_name
```

これは通常は不要です。ただし、`character_set_system` が `character_set_server` または `character_set_client` と異なり、(データベースオブジェクト識別子またはカラム値、あるいはその両方として) 手動で文字を入力した場合、これらの文字はクライアントからの出力に間違っ表示されたり、出力自体が間違っ書式設定されたりすることがあります。このような場合、`--default-character-set=system_character_set` を使用して MySQL クライアントを起動し、システム文字セットに一致するようにクライアント文字セットを設定すると、問題が修正されます。

10.16 MySQL Server の口ケールサポート

`lc_time_names` システム変数で示された口ケールは、曜日および月の名前と短縮形を表示するために使用する言語を制御します。この変数は `DATE_FORMAT()`、`DAYNAME()`、および `MONTHNAME()` 関数の出力に影響を与えます。

`lc_time_names` は、`STR_TO_DATE()` または `GET_FORMAT()` 関数には影響しません。

`lc_time_names` 値は、`FORMAT()` の結果に影響しませんが、この関数は、結果の数値の小数点、桁区切り、および区切り文字のグルーピングに使用する口ケールを指定できるようにするオプションの 3 番目のパラメータを取ります。許可される口ケール値は、`lc_time_names` システム変数の正当な値と同じです。

口ケール名には、`'ja_JP'` や `'pt_BR'` など、IANA (<http://www.iana.org/assignments/language-subtag-registry>) に記載された言語および地域のサブタグが含まれます。システムの口ケール設定に関係なく、デフォルト値は `'en_US'` ですが、グローバルシステム変数の設定に十分な権限がある場合は、サーバーの起動時に値を設定するか、実行時に `GLOBAL` 値を設定できます。[セクション5.1.9.1「システム変数権限」](#)を参照してください。どのクライアントでも、`lc_time_names` の値を調べたり、その `SESSION` 値を設定してそれ自体の接続用の口ケールに影響を与えたりできます。

```
mysql> SET NAMES 'utf8';
Query OK, 0 rows affected (0.09 sec)

mysql> SELECT @@lc_time_names;
+-----+
| @@lc_time_names |
+-----+
| en_US           |
+-----+
1 row in set (0.00 sec)

mysql> SELECT DAYNAME('2010-01-01'), MONTHNAME('2010-01-01');
+-----+-----+
| DAYNAME('2010-01-01') | MONTHNAME('2010-01-01') |
+-----+-----+
| Friday                | January                  |
+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT DATE_FORMAT('2010-01-01','%W %a %M %b');
+-----+
| DATE_FORMAT('2010-01-01','%W %a %M %b') |
+-----+
```



```
| Friday Fri January Jan |
+-----+
1 row in set (0.00 sec)

mysql> SET lc_time_names = 'es_MX';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @@lc_time_names;
+-----+
| @@lc_time_names |
+-----+
| es_MX          |
+-----+
1 row in set (0.00 sec)

mysql> SELECT DAYNAME('2010-01-01'), MONTHNAME('2010-01-01');
+-----+
| DAYNAME('2010-01-01') | MONTHNAME('2010-01-01') |
+-----+
| viernes              | enero                    |
+-----+
1 row in set (0.00 sec)

mysql> SELECT DATE_FORMAT('2010-01-01','%W %a %M %b');
+-----+
| DATE_FORMAT('2010-01-01','%W %a %M %b') |
+-----+
| viernes vie enero ene                    |
+-----+
1 row in set (0.00 sec)
```

影響を受けた関数それぞれの曜日または月の名前は、`utf8` から `character_set_connection` システム変数で指定される文字セットに変換されます。

`lc_time_names` は次のどのロケール値にも設定できます。MySQL でサポートされるロケールのセットは、オペレーティングシステムでサポートされるものと異なる場合があります。

ロケール値	意味
ar_AE	アラビア語 - アラブ首長国連邦
ar_BH	アラビア語 - バーレーン
ar_DZ	アラビア語 - アルジェリア
ar_EG	アラビア語 - エジプト
ar_IN	アラビア語 - インド
ar_IQ	アラビア語 - イラク
ar_JO	アラビア語 - ヨルダン
ar_KW	アラビア語 - クウェート
ar_LB	アラビア語 - レバノン
ar_LY	アラビア語 - リビア
ar_MA	アラビア語 - モロッコ
ar_OM	アラビア語 - オマーン
ar_QA	アラビア語 - カタール
ar_SA	アラビア語 - サウジアラビア
ar_SD	アラビア語 - スーダン
ar_SY	アラビア語 - シリア
ar_TN	アラビア語 - チュニジア
ar_YE	アラビア語 - イエメン
be_BY	ベラルーシ語 - ベラルーシ
bg_BG	ブルガリア語 - ブルガリア

ロケール値	意味
ca_ES	カタラン - スペイン
cs_CZ	チェコ語 - チェコ共和国
da_DK	デンマーク語 - デンマーク
de_AT	ドイツ語 - オーストリア
de_BE	ドイツ語 - ベルギー
de_CH	ドイツ語 - スイス
de_DE	ドイツ語 - ドイツ
de_LU	ドイツ語 - ルクセンブルク
el_GR	ギリシャ語 - ギリシャ
en_AU	英語 - オーストラリア
en_CA	英語 - カナダ
en_GB	英語 - 英国
en_IN	英語 - インド
en_NZ	英語 - ニュージーランド
en_PH	英語 - フィリピン
en_US	英語 - アメリカ合衆国
en_ZA	英語 - 南アフリカ
en_ZW	英語 - ジンバブエ
es_AR	スペイン語 - アルゼンチン
es_BO	スペイン語 - ボリビア
es_CL	スペイン語 - チリ
es_CO	スペイン語 - コロンビア
es_CR	スペイン語 - コスタリカ
es_DO	スペイン語 - ドミニカ共和国
es_EC	スペイン語 - エクアドル
es_ES	スペイン語 - スペイン
es_GT	スペイン語 - グアテマラ
es_HN	スペイン語 - ホンジュラス
es_MX	スペイン語 - メキシコ
es_NI	スペイン語 - ニカラグア
es_PA	スペイン語 - パナマ
es_PE	スペイン語 - ペルー
es_PR	スペイン語 - プエルトリコ
es_PY	スペイン語 - パラグアイ
es_SV	スペイン語 - エルサルバドル
es_US	スペイン語 - アメリカ合衆国
es_UY	スペイン語 - ウルグアイ
es_VE	スペイン語 - ベネズエラ
et_EE	エストニア語 - エストニア
eu_ES	バスク - スペイン
fi_FI	フィンランド語 - フィンランド

ロケール値	意味
fo_FO	フォロー語 - フェロー諸島
fr_BE	フランス語 - ベルギー
fr_CA	フランス語 - カナダ
fr_CH	フランス語 - スイス
fr_FR	フランス語 - フランス
fr_LU	フランス語 - ルクセンブルク
gl_ES	ガリシア - スペイン
gu_IN	グジャラート - インド
he_IL	ヘブライ語 - イスラエル
hi_IN	ヒンディー語 - インド
hr_HR	クロアチア語 - クロアチア
hu_HU	ハンガリー語 - ハンガリー
id_ID	インドネシア語 - インドネシア
is_IS	アイスランド語 - アイスランド
it_CH	イタリア語 - スイス
it_IT	イタリア語 - イタリア
ja_JP	日本語 - 日本
ko_KR	韓国語 - 大韓民国
lt_LT	リトアニア語 - リトアニア
lv_LV	ラトビア語 - ラトビア
mk_MK	マケドニア語 - 北マケドニア
mn_MN	モンゴル語 - モンゴル
ms_MY	マレー - マレーシア
nb_NO	ノルウェー (ブークモール) - ノルウェー
nl_BE	オランダ語 - ベルギー
nl_NL	オランダ語 - オランダ
no_NO	ノルウェー語 - ノルウェー
pl_PL	ポーランド語 - ポーランド
pt_BR	ポルトガル語 - ブラジル
pt_PT	ポルトガル語 - ポルトガル
rm_CH	ロマンシュ語 - スイス
ro_RO	ルーマニア語 - ルーマニア
ru_RU	ロシア語 - ロシア
ru_UA	ロシア語 - ウクライナ
sk_SK	スロバキア語 - スロバキア
sl_SI	スロベニア語 - スロベニア
sq_AL	アルバニア語 - アルバニア
sr_RS	セルビア語 - セルビア
sv_FI	スウェーデン語 - フィンランド
sv_SE	スウェーデン語 - スウェーデン
ta_IN	タミル語 - インド

ロケール値	意味
te_IN	テルグ - インド
th_TH	タイ語 - タイ
tr_TR	トルコ語 - トルコ
uk_UA	ウクライナ語 - ウクライナ
ur_PK	ウルドゥ語 - パキスタン
vi_VN	ベトナム語 - ベトナム
zh_CN	中国語 - 中国
zh_HK	中国語 - 香港
zh_TW	中国語 - 台湾

第 11 章 データ型

目次

11.1 数値データ型	1762
11.1.1 数値データ型の構文	1762
11.1.2 整数型 (真数値) - INTEGER、INT、SMALLINT、TINYINT、MEDIUMINT、BIGINT	1765
11.1.3 固定小数点型 (真数値) - DECIMAL、NUMERIC	1766
11.1.4 浮動小数点型 (概数値) - FLOAT、DOUBLE	1766
11.1.5 ビット値型 - BIT	1767
11.1.6 数値型の属性	1767
11.1.7 範囲外およびオーバーフローの処理	1768
11.2 日時データ型	1769
11.2.1 日時データ型の構文	1770
11.2.2 DATE、DATETIME、および TIMESTAMP 型	1772
11.2.3 TIME 型	1775
11.2.4 YEAR 型	1775
11.2.5 TIMESTAMP および DATETIME の自動初期化および更新機能	1776
11.2.6 時間値での小数秒	1779
11.2.7 日付と時間型間での変換	1780
11.2.8 日付の 2 桁の年	1780
11.3 文字列データ型	1781
11.3.1 文字列データ型の構文	1781
11.3.2 CHAR および VARCHAR 型	1784
11.3.3 BINARY および VARBINARY 型	1786
11.3.4 BLOB 型と TEXT 型	1787
11.3.5 ENUM 型	1788
11.3.6 SET 型	1791
11.4 空間データ型	1793
11.4.1 空間データ型	1794
11.4.2 OpenGIS ジオメトリモデル	1795
11.4.3 サポートされる空間データ形式	1800
11.4.4 ジオメトリの整形形式と妥当性	1803
11.4.5 空間参照システムのサポート	1804
11.4.6 空間カラムの作成	1805
11.4.7 空間カラムへのデータ移入	1805
11.4.8 空間データのフェッチ	1806
11.4.9 空間分析の最適化	1806
11.4.10 空間インデックスの作成	1807
11.4.11 空間インデックスの使用	1808
11.5 JSON データ型	1809
11.6 データ型デフォルト値	1824
11.7 データ型のストレージ要件	1826
11.8 カラムに適した型の選択	1830
11.9 その他のデータベースエンジンのデータ型の使用	1830

MySQL では、複数のカテゴリの SQL データ型がサポートされています: 数値型、日時型、文字列 (文字およびバイト) 型、空間型および JSON データ型。この章では、各カテゴリのタイプのプロパティの概要と詳細、およびデータ型の記憶域要件の概要について説明します。最初の概要は意図的に簡単です。値を指定できる許容形式など、特定のデータ型に関する追加情報は、より詳細な説明を参照してください。

データ型の説明では、次の規則を使用しています。

- 整数型の場合、**M** は最大表示幅を示します。浮動小数点型と固定小数点型の場合、**M** は格納可能な桁数の合計 (精度) です。文字列型の場合は、**M** は最大長です。**M** の許可される最大値は、データ型によって異なります。
- **D** は、浮動小数点型と固定小数点型に適用され、小数点以下の桁数 (スケール) を表します。指定可能な最大値は 30 ですが、**M**-2 以下にしてください。

- `fsp` は、`TIME`、`DATETIME` および `TIMESTAMP` タイプに適用され、小数秒精度、つまり秒の小数部の小数点以下の桁数を表します。`fsp` 値を指定する場合、0 から 6 の範囲にする必要があります。0 の値は、小数部がないことを表します。省略した場合、デフォルトの精度は 0 です。(これは、以前の MySQL バージョンと互換性を保つため、標準 SQL のデフォルトである 6 とは異なっています。)
- 角カッコ (`[`および`]`) は、型定義のオプション部分を示します。

11.1 数値データ型

MySQL はすべての標準 SQL 数値データ型をサポートします。これらの型は、概数値データ型 (`FLOAT`、`REAL`、`DOUBLE PRECISION`) だけでなく、真数値データ型 (`INTEGER`、`SMALLINT`、`DECIMAL`、`NUMERIC`) を含みます。キーワード `INT` は `INTEGER` のシノニムで、キーワード `DEC` および `FIXED` は `DECIMAL` のシノニムです。MySQL では、`DOUBLE` は `DOUBLE PRECISION` (非標準の拡張) のシノニムと見なされます。また、`REAL_AS_FLOAT` SQL モードが有効でないかぎり、`REAL` は `DOUBLE PRECISION` (非標準のバリエーション) のシノニムと見なされます。

`BIT` データ型はビット値を格納し、`MyISAM`、`MEMORY`、`InnoDB` および `NDB` テーブルでサポートされます。

範囲外の値のカラムへの割り当てと、式の評価中のオーバーフローに対する MySQL での処理の詳細は、[セクション 11.1.7 「範囲外およびオーバーフローの処理」](#) を参照してください。

数値データ型の記憶域要件の詳細は、[セクション 11.7 「データ型のストレージ要件」](#) を参照してください。

数値を操作する関数の説明は、[セクション 12.6 「数値関数と演算子」](#) を参照してください。数値オペランドで計算の結果に使用されるデータ型は、オペランドの型と実行される演算によって異なります。詳細は、[セクション 12.6.1 「算術演算子」](#) を参照してください。

11.1.1 数値データ型の構文

整数データ型の場合、`M` は最大表示幅を示します。最大表示幅は 255 です。表示幅は、[セクション 11.1.6 「数値型の属性」](#) で説明されているように、型が格納できる値の範囲とは無関係です。

浮動小数点データ型および固定小数点データ型の場合、`M` は格納できる合計桁数です。

MySQL 8.0.17 では、整数データ型の表示幅属性は非推奨になりました。将来のバージョンの MySQL ではサポートされなくなる予定です。

数値カラムに対して `ZEROFILL` を指定すると、MySQL は自動的にそのカラムに `UNSIGNED` 属性を追加します。

MySQL 8.0.17 では、`ZEROFILL` 属性は数値データ型では非推奨です。将来のバージョンの MySQL ではサポートされなくなる予定です。この属性の効果を生成する別の方法の使用を検討してください。たとえば、アプリケーションでは、`LPAD()` 関数を使用して、必要な幅まで数値をゼロ埋めたり、書式設定された数値を `CHAR` カラムに格納したりできます。

`UNSIGNED` 属性を許可している数値データ型は、`SIGNED` も許可します。ただし、このデータ型はデフォルトで符号付きになっているため、`SIGNED` 属性を指定しても効果はありません。

MySQL 8.0.17 では、`FLOAT`、`DOUBLE` および `DECIMAL` (およびすべてのシノニム) タイプのカラムに対して `UNSIGNED` 属性は非推奨になりました。将来のバージョンの MySQL ではサポートされなくなる予定です。このようなカラムには、かわりに単純な `CHECK` 制約の使用を検討してください。

`SERIAL` は `BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE` のエイリアスです。

整数カラム定義の中の `SERIAL DEFAULT VALUE` は `NOT NULL AUTO_INCREMENT UNIQUE` のエイリアスです。

警告

一方が `UNSIGNED` 型のときに 2 つの整数値の間で減算を行うと、`NO_UNSIGNED_SUBTRACTION` SQL モードが有効でないかぎり、結果の値は符号なしになります。[セクション 12.11 「キャスト関数と演算子」](#) を参照してください。

- `BIT[(M)]`

ビット値型。M は、値あたりのビット数 (1 から 64) を表します。M を省略した場合のデフォルトは 1 です。

- **TINYINT[(M)] [UNSIGNED] [ZEROFILL]**

非常に小さい整数。符号付きの範囲は -128 から 127 です。符号なしの範囲は 0 から 255 です。

- **BOOL、BOOLEAN**

これらの型は **TINYINT(1)** のシノニムです。ゼロの値は false と見なされます。ゼロ以外の値は true と見なされま

```
mysql> SELECT IF(0, 'true', 'false');
+-----+
| IF(0, 'true', 'false') |
+-----+
| false                   |
+-----+

mysql> SELECT IF(1, 'true', 'false');
+-----+
| IF(1, 'true', 'false') |
+-----+
| true                    |
+-----+

mysql> SELECT IF(2, 'true', 'false');
+-----+
| IF(2, 'true', 'false') |
+-----+
| true                    |
+-----+
```

ただし、ここに示されているように、**TRUE** 値と **FALSE** 値はそれぞれ、1 と 0 の単なるエイリアスです。

```
mysql> SELECT IF(0 = FALSE, 'true', 'false');
+-----+
| IF(0 = FALSE, 'true', 'false') |
+-----+
| true                             |
+-----+

mysql> SELECT IF(1 = TRUE, 'true', 'false');
+-----+
| IF(1 = TRUE, 'true', 'false') |
+-----+
| true                             |
+-----+

mysql> SELECT IF(2 = TRUE, 'true', 'false');
+-----+
| IF(2 = TRUE, 'true', 'false') |
+-----+
| false                            |
+-----+

mysql> SELECT IF(2 = FALSE, 'true', 'false');
+-----+
| IF(2 = FALSE, 'true', 'false') |
+-----+
| false                            |
+-----+
```

最後の 2 つのステートメントは、2 が 1 とも 0 とも等しくないために示される結果を表示します。

- **SMALLINT[(M)] [UNSIGNED] [ZEROFILL]**

小さい整数。符号付きの範囲は -32768 から 32767 です。符号なしの範囲は 0 から 65535 です。

- **MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]**

中間サイズの整数。符号付きの範囲は -8388608 から 8388607 です。符号なしの範囲は 0 から 16777215 です。

- `INT[(M)] [UNSIGNED] [ZEROFILL]`

普通サイズの整数。符号付きの範囲は `-2147483648` から `2147483647` です。符号なしの範囲は `0` から `4294967295` です。

- `INTEGER[(M)] [UNSIGNED] [ZEROFILL]`

この型は `INT` のシノニムです。

- `BIGINT[(M)] [UNSIGNED] [ZEROFILL]`

大きい整数。符号付きの範囲は `-9223372036854775808` から `9223372036854775807` です。符号なしの範囲は `0` から `18446744073709551615` です。

`SERIAL` は `BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE` のエイリアスです。

`BIGINT` カラムについて注意の必要な点は、次のとおりです。

- すべての演算は符号付きの `BIGINT` 値または `DOUBLE` 値を使用して行われるため、ビット関数を使用しないかぎり、`9223372036854775807` (63 ビット) よりも大きい符号なしの整数を使用しないでください。そのようにした場合、`BIGINT` 値から `DOUBLE` 値への変換時に、丸め誤差のために結果の最後の数桁に誤差が生じる可能性があります。

MySQL は、次の場合に、`BIGINT` を扱うことができます。

- 符号なしの大きな値を `BIGINT` カラムに格納するために整数を使用するとき。
- `MIN(col_name)` または `MAX(col_name)` 内。ここで `col_name` は `BIGINT` カラムを指します。
- 演算子 (+, -, * など) を使用する場合。ここで両方のオペランドは整数です。
- 文字列を使用して格納すると、いつでも正確な整数値を `BIGINT` カラムに格納できます。この場合、MySQL は、中間倍精度表現を含まない文字列から数値に変換します。
- 両方のオペランドが整数値の場合、`-`、`+`、および `*` の演算子は、`BIGINT` 演算を使用します。これは、2 つの大きい整数 (または整数を返す関数からの結果) を掛け合わせた場合、その結果が `9223372036854775807` より大きいときには、予期しない結果になるということを意味します。

- `DECIMAL[(M[,D])] [UNSIGNED] [ZEROFILL]`

パックされた「正確な」固定小数点数。M は桁数の合計 (精度) で、D は小数点以下の桁数 (スケール) です。小数点と、負の数に対する `-` の記号は M にはカウントされません。D が 0 のときは、小数点や小数部はありません。`DECIMAL` の最大桁数 (M) は 65 です。サポートされる小数部の最大桁数 (D) は 30 です。D が省略された場合のデフォルトは 0 です。M が省略された場合のデフォルトは 10 です。(`DECIMAL` リテラルのテキストの長さには制限もあります。 [セクション 12.25.3 「式の処理」](#) を参照してください。)

`UNSIGNED` が指定されている場合、負の値は許可されません。MySQL 8.0.17 では、`DECIMAL` 型のカラム (およびシノニム) の `UNSIGNED` 属性は非推奨になりました。将来のバージョンの MySQL ではサポートされなくなる予定です。このようなカラムには、かわりに単純な `CHECK` 制約の使用を検討してください。

`DECIMAL` カラムを使用したすべての基本的な計算 (+, -, *, /) は、65 桁の精度で行われます。

- `DEC[(M[,D])] [UNSIGNED] [ZEROFILL]`, `NUMERIC[(M[,D])] [UNSIGNED] [ZEROFILL]`, `FIXED[(M[,D])] [UNSIGNED] [ZEROFILL]`

これらの型は `DECIMAL` のシノニムです。 `FIXED` シノニムは、ほかのデータベースシステムとの互換性のために使用できます。

- `FLOAT[(M,D)] [UNSIGNED] [ZEROFILL]`

小さい (単精度) 浮動小数点数。許可される値は、`-3.402823466E+38` から `-1.175494351E-38`、`0`、および `1.175494351E-38` から `3.402823466E+38` です。これらは、IEEE スタンドに基いた理論的な限度です。使用しているハードウェアまたはオペレーティングシステムによっては、実際の範囲は少し小さくなる場合があります。

M は桁数の合計で、**D** は小数点以下の桁数です。 **M** と **D** を省略した場合、値はハードウェアで許可された限度まで格納されます。単精度小数点数はおおよそ小数第 7 位まで正確です。

FLOAT(M,D) は、非標準の MySQL 拡張機能です。MySQL 8.0.17 では、この構文は非推奨であり、将来のバージョンの MySQL ではサポートされなくなる予定です。

UNSIGNED が指定されている場合、負の値は許可されません。MySQL 8.0.17 では、**FLOAT** 型のカラム (およびシノニム) の **UNSIGNED** 属性は非推奨になっており、将来のバージョンの MySQL ではサポートされなくなる予定です。このようなカラムには、かわりに単純な **CHECK** 制約の使用を検討してください。

MySQL ではすべての計算が倍精度で行われているので、**FLOAT** を使用すると、予想外の問題が起きることがあります。 [セクション B.3.4.7 「一致する行がない場合の問題の解決」](#) を参照してください。

- **FLOAT(p) [UNSIGNED] [ZEROFILL]**

浮動小数点数です。 **p** は精度をビットで表現しますが、MySQL は、結果として得られるデータ型に対して **FLOAT** または **DOUBLE** のどちらを使用するかを決めるためだけにこの値を使用します。 **p** が 0 から 24 のとき、そのデータ型は **M** 値も **D** 値もない **FLOAT** になります。 **p** が 25 から 53 のとき、そのデータ型は **M** 値も **D** 値もない **DOUBLE** になります。結果となるカラムの範囲は、このセクションで前述した単精度 **FLOAT** または倍精度 **DOUBLE** データ型の場合と同じです。

UNSIGNED が指定されている場合、負の値は許可されません。MySQL 8.0.17 では、**FLOAT** 型のカラム (およびシノニム) の **UNSIGNED** 属性は非推奨になっており、将来のバージョンの MySQL ではサポートされなくなる予定です。このようなカラムには、かわりに単純な **CHECK** 制約の使用を検討してください。

FLOAT(p) 構文は ODBC との互換性を確保するために用意されています。

- **DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL]**

普通サイズ (倍精度) の浮動小数点数。許可されている値は、**-1.7976931348623157E+308** から **-2.2250738585072014E-308**、**0**、および **2.2250738585072014E-308** から **1.7976931348623157E+308** です。これらは、IEEE スタンドに基いた理論的な限度です。使用しているハードウェアまたはオペレーティングシステムによっては、実際の範囲は少し小さくなる場合があります。

M は桁数の合計で、**D** は小数点以下の桁数です。 **M** と **D** を省略した場合、値はハードウェアで許可された限度まで格納されます。倍精度小数点数はおおよそ小数第 15 位まで正確です。

DOUBLE(M,D) は、非標準の MySQL 拡張機能です。MySQL 8.0.17 では、この構文は非推奨であり、将来のバージョンの MySQL ではサポートされなくなる予定です。

UNSIGNED が指定されている場合、負の値は許可されません。MySQL 8.0.17 では、**DOUBLE** 型のカラム (およびシノニム) の **UNSIGNED** 属性は非推奨になっており、将来のバージョンの MySQL ではサポートされなくなる予定です。このようなカラムには、かわりに単純な **CHECK** 制約の使用を検討してください。

- **DOUBLE PRECISION[(M,D)] [UNSIGNED] [ZEROFILL], REAL[(M,D)] [UNSIGNED] [ZEROFILL]**

これらの型は **DOUBLE** のシノニムです。例外: **REAL_AS_FLOAT** SQL モードが有効な場合は、**DOUBLE** ではなく **REAL** が **FLOAT** のシノニムになります。

11.1.2 整数型 (真数値) - INTEGER、INT、SMALLINT、TINYINT、MEDIUMINT、BIGINT

MySQL では、**INTEGER** (または **INT**) および **SMALLINT** の SQL 標準整数型をサポートします。標準に対する拡張として、MySQL では、**TINYINT**、**MEDIUMINT**、および **BIGINT** の整数型もサポートします。次の表に、整数型ごとの必要なストレージと範囲を示します。

表 11.1 MySQL でサポートされる整数型に必要な記憶域および範囲

型	記憶域 (バイト)	署名済最小値	最小未署名値	署名された最大値	最大未署名値
TINYINT	1	-128	0	127	255

型	記憶域 (バイト)	署名済最小値	最小未署名値	署名された最大値	最大未署名値
SMALLINT	2	-32768	0	32767	65535
MEDIUMINT	3	-8388608	0	8388607	16777215
INT	4	-2147483648	0	2147483647	4294967295
BIGINT	8	-2 ⁶³	0	2 ⁶³ -1	2 ⁶⁴ -1

11.1.3 固定小数点型 (真数値) - DECIMAL、NUMERIC

DECIMAL および NUMERIC 型は真数値データ値を格納します。これらの型は、金銭データを扱う場合など、正確な精度を保持することが重要な場合に使用されます。MySQL では、NUMERIC は DECIMAL として実装されるので、DECIMAL に関する次の注意事項が NUMERIC にも同様に適用されます。

MySQL は、DECIMAL 値をバイナリ形式で格納します。セクション12.25「高精度計算」を参照してください。

DECIMAL のカラム宣言では、精度とスケールを指定できます (通常は指定します)。例:

```
salary DECIMAL(5,2)
```

この例では、5 が精度で、2 がスケールです。精度は、その値に格納された有効な桁数を表し、スケールは小数点以下に格納できる桁数を表しています。

標準 SQL では、DECIMAL(5,2) には小数部が 2 桁の合計 5 桁の値を格納できる必要があるため、salary カラムに格納できる値は、-999.99 から 999.99 の範囲になります。

標準 SQL では、構文 DECIMAL(M) は、DECIMAL(M,0) と同等です。同様に、構文 DECIMAL は DECIMAL(M,0) と同等です。M の値を決定するために、実装は許可されています。MySQL は、DECIMAL 構文のこれらのバリエーション形式をどちらもサポートします。M のデフォルト値は 10 です。

スケールが 0 の場合、DECIMAL 値には小数点も小数部も含まれません。

DECIMAL の最大桁数は 65 ですが、指定した DECIMAL カラムの実際の範囲は、その指定したカラムの精度またはスケールによって制約される場合があります。指定のスケールで許可されている数より多くの桁が小数点以下にある値が、このようなカラムに割り当てられた場合、値はそのスケールに変換されます。(正確な動作はオペレーティングシステム固有ですが、一般的には効果は許可されている桁数に切り捨てられます。)

11.1.4 浮動小数点型 (概数値) - FLOAT、DOUBLE

FLOAT および DOUBLE 型は概数値データ値を表します。MySQL は、単精度値には 4 バイトを、倍精度値には 8 バイトを使用します。

FLOAT の場合、SQL 標準では、カッコで囲まれたキーワード FLOAT の後のビット単位の精度 (指数の範囲ではない) をオプションで指定できます。つまり、FLOAT(p) です。MySQL では、このオプションの精度指定もサポートされていますが、FLOAT(p) の精度値は記憶域サイズの決定にのみ使用されます。0 から 23 の精度は、4 バイト単精度の FLOAT カラムになります。24 から 53 の精度は、8 バイト倍精度の DOUBLE カラムになります。

MySQL は、FLOAT(M,D) または REAL(M,D) または DOUBLE PRECISION(M,D) の非標準の構文を許可します。ここで、(M、D) は、値は合計で M 桁まで格納でき、そのうちの D 桁は小数点以下です。たとえば、FLOAT(7,4) として定義されたカラムは、-999.9999 として表示されます。MySQL は、値を格納するときに丸めを行うので、FLOAT(7,4) カラムに 999.00009 を挿入すると、近似の結果は 999.0001 になります。

MySQL 8.0.17 では、非標準の FLOAT(M,D) および DOUBLE(M,D) 構文は非推奨であり、将来のバージョンの MySQL ではサポートされなくなる予定です。

浮動小数点値は概数値であり、真数値としては格納されないため、比較で値を真数値として扱おうとすると、問題が発生することがあります。これらはまた、プラットフォームまたは実装の依存関係にも従います。詳細は、セクション B.3.4.8「浮動小数点値に関する問題」を参照してください。

移植性を最大にするために、概数値データ値のストレージを必要とするコードでは、精度または桁数が指定されていない FLOAT または DOUBLE PRECISION を使用する必要があります。

11.1.5 ビット値型 - BIT

BIT データ型は、ビット値の格納に使用されます。BIT(M) の型は、M ビット値のストレージを有効にします。M の範囲は 1 から 64 までが可能です。

ビット値を指定するには、b'value' 表記を使用できます。value は、0 と 1 で書かれたバイナリ値です。たとえば、b'111' と b'1000000' はそれぞれ 7 と 128 を表しています。セクション 9.1.5 「ビット値リテラル」を参照してください。

M ビット長よりも短い BIT(M) カラムに値を割り当てた場合、その値の左側はゼロで埋められます。たとえば、b'101' という値を BIT(6) カラムに割り当てると、実際には b'000101' を割り当てた場合と同じこととなります。

NDB Cluster. 特定の NDB テーブルで使用されるすべての BIT カラムの最大合計サイズは 4096 ビットを超えることはできません。

11.1.6 数値型の属性

MySQL では、整数データ型の基本キーワードに続く括弧内で、その型の表示幅をオプションで指定する拡張をサポートしています。たとえば、INT(4) は、4 桁の表示幅の INT を指定しています。このオプションの表示幅は、左側をスペースでパディングすることによって、カラムに対して指定された幅よりも狭く整数値を表示するために、アプリケーションで使用される場合があります。(つまり、この幅は結果セットで返されるメタデータの中にあります。使用されるかどうかはアプリケーションによって決まります。)

表示幅は、カラムに格納できない値の範囲を制約しません。カラムの表示幅より広い値が正しく表示されなくなることもあります。たとえば、SMALLINT(3) として指定されたカラムには、-32768 から 32767 の通常の SMALLINT 範囲があり、3 桁が許可されたこの範囲外の値は、4 桁以上を使用してすべて表示されます。

オプションの (非標準の) ZEROFILL 属性とともに使用すると、空白のデフォルトの埋込みがゼロに置き換えられます。たとえば、INT(4) ZEROFILL として宣言されたカラムの場合、5 の値は 0005 として取得されます。

注記

式または UNION クエリーに含まれるカラムでは、ZEROFILL 属性は無視されます。

ZEROFILL 属性を持つ整数カラムに表示幅より大きな値を格納した場合、MySQL が一部の複雑な結合に対して一時テーブルを生成するときの問題が発生することがあります。これらの場合、MySQL は、カラムの表示幅内でデータ値が適合すると想定します。

MySQL 8.0.17 では、整数データ型の表示幅属性と同様に、数値データ型の ZEROFILL 属性は非推奨になりました。ZEROFILL のサポートおよび整数データ型の表示幅は、将来のバージョンの MySQL で削除される予定です。これらの属性の効果を生成する別の方法の使用を検討してください。たとえば、アプリケーションでは、LPAD() 関数を使用して、必要な幅まで数値をゼロ埋めしたり、書式設定された数値を CHAR カラムに格納したりできます。

すべての整数型は、オプション (非標準) の UNSIGNED 属性を持つことができます。符号なし型を使用すると、カラムに負でない数値のみを許可したり、カラムの上限の数値範囲を大きくする必要がある場合に使用できます。たとえば、INT カラムが UNSIGNED の場合、カラム範囲のサイズは同じですが、そのエンドポイントは -2147483648 および 2147483647 から 0 および 4294967295 にシフトします。

浮動小数点と固定小数点も UNSIGNED になり得ます。整数型と同じように、この属性は負の値がカラムに格納されるのを防ぎます。整数型とは異なり、カラム値の上限範囲は変わりません。MySQL 8.0.17 では、FLOAT、DOUBLE および DECIMAL (およびすべてのシノニム) タイプのカラムに対して UNSIGNED 属性は非推奨であり、将来のバージョンの MySQL でサポートされなくなる予定です。このようなカラムには、かわりに単純な CHECK 制約の使用を検討してください。

数値カラムに ZEROFILL を指定すると、MySQL によって UNSIGNED 属性が自動的に追加されます。

整数または浮動小数点データ型は、AUTO_INCREMENT 属性を持つことができます。インデックス付けされた AUTO_INCREMENT カラムに NULL の値を挿入すると、そのカラムは次の順序値に設定されます。通常、これは value+1 です。ここで value は現在テーブルにあるカラムの最大値です。(AUTO_INCREMENT の順序は 1 で始まります。)

AUTO_INCREMENT カラムへの 0 の格納は、NO_AUTO_VALUE_ON_ZERO SQL モードが有効になっていないかぎり、NULL の格納と同じ効果があります。

`AUTO_INCREMENT` 値を生成するために `NULL` を挿入する場合、カラムを `NOT NULL` と宣言する必要があります。カラムが `NULL` として宣言されている場合、`NULL` を挿入すると `NULL` が格納されます。他の値を `AUTO_INCREMENT` カラムに挿入すると、カラムはその値に設定され、次に自動的に生成される値が挿入された値から順番に続くように順序がリセットされます。

`AUTO_INCREMENT` カラムの負の値はサポートされていません。

`CHECK` 制約は、`AUTO_INCREMENT` 属性を持つカラムを参照することも、`CHECK` 制約で使用される既存のカラムに `AUTO_INCREMENT` 属性を追加することもできません。

MySQL 8.0.17 では、`FLOAT` および `DOUBLE` カラムに対する `AUTO_INCREMENT` のサポートは非推奨になりました。将来のバージョンの MySQL で削除される予定です。このようなカラムから `AUTO_INCREMENT` 属性を削除するか、整数型に変換することを検討してください。

11.1.7 範囲外およびオーバーフローの処理

MySQL が、カラムデータ型の許可できる範囲外にある数値カラムに値を格納すると、結果は、その時点で有効な SQL モードによって異なります。

- 厳密な SQL モードが有効な場合、SQL 標準に従って、MySQL は範囲外の値を拒否してエラーを表示し、挿入は失敗します。
- 制限モードが有効になっていない場合、MySQL はカラムのデータ型範囲の適切なエンドポイントに値をクリップし、かわりに結果の値を格納します。

範囲外の値が整数カラムに割り当てられると、MySQL は、カラムデータ型の範囲の対応する終点を表す値を格納します。

浮動小数点または固定小数点カラムに、指定された (またはデフォルトの) 精度とスケールによって暗示された範囲を超えた値が割り当てられると、MySQL はその範囲の対応する終点を表す値を格納します。

テーブル `t1` に次の定義があるとします。

```
CREATE TABLE t1 (i1 TINYINT, i2 TINYINT UNSIGNED);
```

厳密な SQL モードを有効にすると、範囲外エラーが発生します:

```
mysql> SET sql_mode = 'TRADITIONAL';
mysql> INSERT INTO t1 (i1, i2) VALUES(256, 256);
ERROR 1264 (22003): Out of range value for column 'i1' at row 1
mysql> SELECT * FROM t1;
Empty set (0.00 sec)
```

厳密な SQL モードが有効になっていない場合、警告付きでクリッピングが発生します:

```
mysql> SET sql_mode = "";
mysql> INSERT INTO t1 (i1, i2) VALUES(256, 256);
mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message                                     |
+-----+-----+-----+
| Warning | 1264 | Out of range value for column 'i1' at row 1 |
| Warning | 1264 | Out of range value for column 'i2' at row 1 |
+-----+-----+-----+
mysql> SELECT * FROM t1;
+-----+-----+
| i1 | i2 |
+-----+-----+
| 127 | 255 |
+-----+-----+
```

厳密な SQL モードが有効になっていない場合、クリッピングによって発生するカラム割当て変換は、`ALTER TABLE`、`LOAD DATA`、`UPDATE` および複数行の `INSERT` ステートメントの警告としてレポートされます。厳密モードでは、これらのステートメントは失敗し、テーブルがトランザクションテーブルかどうかやその他の要因に応じて、一部またはすべての値が挿入または変更されません。詳細は、[セクション 5.1.11 「サーバー SQL モード」](#) を参照してください。

数値式の評価中にオーバーフローすると、エラーが発生します。たとえば、符号付きの `BIGINT` の最大値は `9223372036854775807` なので、次の式ではエラーが発生します。

```
mysql> SELECT 9223372036854775807 + 1;
ERROR 1690 (22003): BIGINT value is out of range in '(9223372036854775807 + 1)'
```

この場合に演算を成功させるには、値を符号なしに変換します。

```
mysql> SELECT CAST(9223372036854775807 AS UNSIGNED) + 1;
+-----+
| CAST(9223372036854775807 AS UNSIGNED) + 1 |
+-----+
|          9223372036854775808 |
+-----+
```

オーバーフローが起きるかどうかはオペランドの範囲に応じて異なります。したがって、前述の式を処理するもう 1 つの方法として、`DECIMAL` 値に整数より大きな範囲があるので正確な値の演算を使用します。

```
mysql> SELECT 9223372036854775807.0 + 1;
+-----+
| 9223372036854775807.0 + 1 |
+-----+
|          9223372036854775808.0 |
+-----+
```

一方が `UNSIGNED` 型のときに 2 つの整数値の間で減算を行うと、デフォルトでは符号なしの結果が生成されます。それ以外の場合は、エラーが発生します:

```
mysql> SET sql_mode = "";
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT CAST(0 AS UNSIGNED) - 1;
ERROR 1690 (22003): BIGINT UNSIGNED value is out of range in '(cast(0 as unsigned) - 1)'
```

`NO_UNSIGNED_SUBTRACTION` SQL モードが有効な場合は、結果は負になります。

```
mysql> SET sql_mode = 'NO_UNSIGNED_SUBTRACTION';
mysql> SELECT CAST(0 AS UNSIGNED) - 1;
+-----+
| CAST(0 AS UNSIGNED) - 1 |
+-----+
|          -1 |
+-----+
```

このような演算の結果を使用して `UNSIGNED` 整数カラムが更新されると、結果はそのカラム型の最大値に切り落とされます。`NO_UNSIGNED_SUBTRACTION` が有効になっている場合は、0 に切り落とされます。厳密な SQL モードが有効になっている場合は、エラーが発生し、カラムは変わりません。

11.2 日時データ型

時間値を表すための日時データ型は、`DATE`、`TIME`、`DATETIME`、`TIMESTAMP` および `YEAR` です。それぞれの時間型には、一定範囲の有効な値のほかに、MySQL では表すことのできない無効な値の指定時に使用できる「ゼロ」値があります。`TIMESTAMP` および `DATETIME` タイプには、[セクション11.2.5「TIMESTAMP および DATETIME の自動初期化および更新機能」](#) で説明されている特別な自動更新動作があります。

時間データ型の記憶域要件の詳細は、[セクション11.7「データ型のストレージ要件」](#) を参照してください。

時間値を演算する関数の説明については、[セクション12.7「日付および時間関数」](#) を参照してください。

日付と時間型を処理するときに、次の考慮事項に留意してください。

- MySQL は、標準出力形式で所定の日付または時間型の値を取得しますが、(たとえば、日付または時間型に割り当てたり、比較したりする値を指定するときに) 入力した入力値に対してさまざまな形式を解釈しようとします。日付と時間型に許可されている形式の説明については、[セクション9.1.3「日付リテラルと時間リテラル」](#) を参照してください。有効な値を入力する必要があります。ほかの形式で値を使用すると、予期しない結果が生じることがあります。

- MySQL は、複数の形式で値を解釈しようとしますが、日付の部分は、ほかでは一般的に使用される月-日-年や日-月-年の順 ('09-04-98' や '04-09-98' など) ではなく、年-月-日の順 ('98-09-04' など) で常に指定する必要があります。他の順序の文字列を年 - 月 - 日の順序に変換するには、`STR_TO_DATE()` 関数が役立つ場合があります。
- 世紀が不明であるため、2桁の年の値を含む日付はあいまいです。MySQL は、次のルールを使用して2桁の年の値を解釈します:
 - 70-99 の範囲内の年の値は 1970-1999 になります。
 - 00-69 の範囲内の年の値は 2000-2069 になります。
 セクション11.2.8「日付の2桁の年」も参照してください。
- ある時間型から別の時間型への値の変換は、セクション11.2.7「日付と時間型間での変換」でのルールに従って行われます。
- 値が数値コンテキストで使用されている場合、MySQL は日付または時間の値を数値に自動的に変換します。逆の場合も同様です。
- デフォルトで MySQL は、日付または時間型の値で、範囲外であるか、それ以外で型にとって無効である値を見つけた場合、値をその型の「ゼロ」値に変換します。その例外では、範囲外の `TIME` 値は `TIME` 範囲の適切な終点に切り落とされます。
- SQL モードを適切な値に設定することで、MySQL がサポートする日付の種類をより正確に指定できます。(セクション5.1.11「サーバー SQL モード」を参照してください。) `ALLOW_INVALID_DATES` SQL モードを有効にすることによって、'2009-11-31' などの特定の日付を MySQL に受け入れさせることができます。これは、ユーザーが今後の処理のために、(たとえば Web フォームで) 指定した「間違っている可能性のある」値をデータベースに格納するときに役立ちます。このモードでは、MySQL は、月が 1 から 12 までの範囲にあることと、日付が 1 から 31 までの範囲にあることのみ検証します。
- MySQL では、`DATE` または `DATETIME` カラムに、日がゼロ、または月および日がゼロである日付の格納を許可しています。これは、正確な日付がわかっていない可能性のある生年月日を格納する必要があるアプリケーションで役立ちます。この場合は、単に日付を '2009-00-00' または '2009-01-00' として格納します。ただし、これらのような日付を使用すると、完全な日付を必要とする `DATE_SUB()` や `DATE_ADD()` などの関数の正しい結果が得られないようにする必要があります。日付にゼロの月または日の部分を許可しないようにするには、`NO_ZERO_IN_DATE` モードを有効にします。
- MySQL では、「ダミーの日付」として '0000-00-00' の「ゼロ」の値を格納できます。場合によっては、これは `NULL` 値を使用するよりも便利であり、使用するデータおよびインデックス領域が少なくなることがあります。'0000-00-00'を禁止するには、`NO_ZERO_DATE` モードを有効にします。
- Connector/ODBC で使用される「ゼロ」の日付または時間の値は、ODBC がこのような値を処理できないため、`NULL` に自動的に変換されます。

次の表に、それぞれの型の「ゼロ」値の形式を示します。「ゼロ」値は特別ですが、表に示されている値を使用して、格納したり、明示的に参照したりできます。また、より簡単に記述できる '0' や 0 の値を使用してこれを行うこともできます。日付部分 (`DATE`、`DATETIME` および `TIMESTAMP`) を含む時間型の場合、これらの値を使用すると警告またはエラーが発生する可能性があります。正確な動作は、厳密および `NO_ZERO_DATE` SQL モードが有効になっているかどうかによって異なります。セクション5.1.11「サーバー SQL モード」を参照してください。

データ型	「ゼロ」値
<code>DATE</code>	'0000-00-00'
<code>TIME</code>	'00:00:00'
<code>DATETIME</code>	'0000-00-00 00:00:00'
<code>TIMESTAMP</code>	'0000-00-00 00:00:00'
<code>YEAR</code>	0000

11.2.1 日時データ型の構文

時間値を表すための日時データ型は、`DATE`、`TIME`、`DATETIME`、`TIMESTAMP` および `YEAR` です。

DATE および DATETIME 範囲の説明では、「サポートされている」とは、以前の値は機能するが、保証はないことを意味します。

MySQL では、マイクロ秒 (6 桁) までの精度で、TIME、DATETIME および TIMESTAMP の値に小数秒を使用できます。小数秒部を含むカラムを定義するには、`type_name(fsp)` の構文を使用します。ここで、`type_name` は TIME、DATETIME、または TIMESTAMP であり、`fsp` は小数秒の精度です。例:

```
CREATE TABLE t1 (t TIME(3), dt DATETIME(6), ts TIMESTAMP(0));
```

`fsp` 値を指定する場合、0 から 6 の範囲にする必要があります。0 の値は、小数部がないことを表します。省略した場合、デフォルトの精度は 0 です。(これは、以前の MySQL バージョンと互換性を保つため、標準 SQL のデフォルトである 6 とは異なっています。)

テーブル内の TIMESTAMP または DATETIME カラムには、自動初期化および自動更新プロパティを設定できます。セクション 11.2.5 「TIMESTAMP および DATETIME の自動初期化および更新機能」を参照してください。

- DATE

日付です。サポートしている範囲は '1000-01-01' から '9999-12-31' です。MySQL では、DATE 値は 'YYYY-MM-DD' 形式で表示されますが、文字列または数値を使用した DATE カラムへの値の割当ては許可されます。

- DATETIME[(fsp)]

日付と時間の組み合わせです。サポートしている範囲は '1000-01-01 00:00:00.000000' から '9999-12-31 23:59:59.999999' です。MySQL では、DATETIME 値は 'YYYY-MM-DD hh:mm:ss[.fraction]' 形式で表示されますが、文字列または数値を使用した DATETIME カラムへの値の割当ては許可されます。

小数秒精度を指定するには、0 から 6 の範囲のオプションの `fsp` 値を指定できます。0 の値は、小数部がないことを表します。省略した場合、デフォルトの精度は 0 です。

セクション 11.2.5 「TIMESTAMP および DATETIME の自動初期化および更新機能」で説明されているように、DEFAULT および ON UPDATE のカラム定義句を使用して、DATETIME カラムの現在の日時への自動初期化および更新を指定できます。

- TIMESTAMP[(fsp)]

タイムスタンプです。範囲は '1970-01-01 00:00:01.000000' UTC から '2038-01-19 03:14:07.999999' UTC です。TIMESTAMP 値は、エポック ('1970-01-01 00:00:00' UTC) からの秒数として格納されます。TIMESTAMP は、'1970-01-01 00:00:00' という値を表すことはできません。これは、エポックからの秒数が 0 であることと同等で、0 という値は '0000-00-00 00:00:00'、つまり「ゼロ」の TIMESTAMP 値を表すために予約されているからです。

小数秒精度を指定するには、0 から 6 の範囲のオプションの `fsp` 値を指定できます。0 の値は、小数部がないことを表します。省略した場合、デフォルトの精度は 0 です。

サーバーで TIMESTAMP 定義をどのように扱うかは、`explicit_defaults_for_timestamp` システム変数の値によって異なります (セクション 5.1.8 「サーバーシステム変数」を参照してください)。

`explicit_defaults_for_timestamp` が有効な場合、すべての TIMESTAMP カラムへの DEFAULT CURRENT_TIMESTAMP または ON UPDATE CURRENT_TIMESTAMP 属性の自動的な割り当ては行われません。これらはカラム定義に明示的に含める必要があります。また、NOT NULL として明示的に宣言されていないすべての TIMESTAMP は、NULL 値を許可します。

`explicit_defaults_for_timestamp` が無効になっている場合、サーバーは次のように TIMESTAMP を処理します:

特に指定されていないかぎり、テーブル内の最初の TIMESTAMP カラムは、明示的に値が割り当てられていなくてももっとも新しい変更の日時に自動的に設定されるように定義されています。これにより、TIMESTAMP は、INSERT または UPDATE 操作のタイムスタンプの記録に役立ちます。NULL 値を許可するように NULL 属性で定義されていないかぎり、NULL 値を割り当てることによって、すべての TIMESTAMP カラムを現在の日付と時間に設定することもできます。

自動初期化および現在の日付と時間への自動更新は、DEFAULT CURRENT_TIMESTAMP および ON UPDATE CURRENT_TIMESTAMP カラム定義句を使用して指定できます。デフォルトでは、前述のように最初の

TIMESTAMP カラムにこれらのプロパティが含まれます。ただし、テーブル内の **TIMESTAMP** カラムは、これらのプロパティを持つように定義できます。

- **TIME[(fsp)]**

時間です。範囲は、'-838:59:59.000000' から '838:59:59.000000' です。MySQL では、**TIME** 値は 'hh:mm:ss[.fraction]' 形式で表示されますが、文字列または数値を使用した **TIME** カラムへの値の割当ては許可されます。

小数秒精度を指定するには、0 から 6 の範囲のオプションの **fsp** 値を指定できます。0 の値は、小数部がないことを表します。省略した場合、デフォルトの精度は 0 です。

- **YEAR[(4)]**

4 桁形式の年。MySQL では、**YEAR** 値は **YYYY** 形式で表示されますが、文字列または数値を使用した **YEAR** カラムへの値の割当ては許可されます。値は、1901 から 2155 または 0000 として表示されます。

入力値の **YEAR** の表示形式および解釈に関する追加情報については、[セクション11.2.4 「YEAR 型」](#) を参照してください。

注記

MySQL 8.0.19 では、明示的な表示幅を持つ **YEAR(4)** データ型は非推奨になりました。将来のバージョンの MySQL ではサポートされなくなる予定です。かわりに、同じ意味を持つ表示幅を指定せずに **YEAR** を使用してください。

MySQL 8.0 では、古いバージョンの MySQL で許可されている 2 桁の **YEAR(2)** データ型はサポートされていません。4 桁の **YEAR** に変換する手順は、[MySQL 5.7 Reference Manual](#) の [2-Digit YEAR\(2\) Limitations and Migrating to 4-Digit YEAR](#) を参照してください。

SUM() および **AVG()** 集計関数は時間値を扱いません。(これらは値を数字に変換するので、最初の数字以外の文字のあとのすべての情報が失われます。) この問題を回避するには、数値単位に変換し、集計操作を実行してから、時間値に戻します。例:

```
SELECT SEC_TO_TIME(SUM(TIME_TO_SEC(time_col))) FROM tbl_name;  
SELECT FROM_DAYS(SUM(TO_DAYS(date_col))) FROM tbl_name;
```

11.2.2 DATE、DATETIME、および TIMESTAMP 型

DATE、**DATETIME**、および **TIMESTAMP** 型は関連しています。このセクションでは、これらの特徴、似ている点、および異なる点について説明します。MySQL は、[セクション9.1.3 「日付リテラルと時間リテラル」](#) で説明している複数の形式で、**DATE**、**DATETIME**、および **TIMESTAMP** 値を認識します。**DATE** および **DATETIME** 範囲の説明では、「サポートされている」とは、以前の値は機能するが、保証はないということを意味します。

DATE 型は、日付部分を含むが時間部分は含まない値に使用されます。MySQL は、**DATE** 値を 'YYYY-MM-DD' 形式で取得して表示します。サポートしている範囲は '1000-01-01' から '9999-12-31' です。

DATETIME 型は、日付と時間の両方の部分を含む値に使用されます。MySQL は、**DATETIME** 値を 'YYYY-MM-DD hh:mm:ss' 形式で取得して表示します。サポートしている範囲は '1000-01-01 00:00:00' から '9999-12-31 23:59:59' です。

TIMESTAMP データ型は、日付と時間の両方の部分を含む値に使用されます。**TIMESTAMP** には、'1970-01-01 00:00:01' UTC から '2038-01-19 03:14:07' UTC の範囲があります。

DATETIME または **TIMESTAMP** 値には、マイクロ秒 (6 桁) までの精度で後続の小数秒部分を含めることができます。特に、**DATETIME** または **TIMESTAMP** カラムに挿入された値の小数部は、破棄されるのではなく格納されます。小数部が含まれている場合、これらの値の書式は 'YYYY-MM-DD hh:mm:ss[.fraction]'、**DATETIME** 値の範囲は '1000-01-01 00:00:00.000000' から '9999-12-31 23:59:59.999999'、**TIMESTAMP** 値の範囲は '1970-01-01 00:00:01.000000' から '2038-01-19 03:14:07.999999' です。小数部は、常に時間の残りの部分から小数点で区分する必要があります。これ以外の小数秒区切り文字は認識されません。MySQL の小数秒のサポートの詳細は、[セクション11.2.6 「時間値での小数秒」](#) を参照してください。

TIMESTAMP および DATETIME データ型は、現在の日時への自動初期化および更新を提供します。詳細は、[セクション11.2.5「TIMESTAMP および DATETIME の自動初期化および更新機能」](#)を参照してください。

MySQL は、TIMESTAMP 値を、ストレージでは現在のタイムゾーンを UTC に変換し、取得では UTC から現在のタイムゾーンに戻します。(DATETIME などのほかの型ではこれは行われません。) デフォルトでは、接続ごとの現在のタイムゾーンはサーバーの時間です。タイムゾーンは接続ごとに設定できます。タイムゾーン設定が一定であるかぎり、格納した値と同じ値に戻すことができます。TIMESTAMP 値を格納したあとで、タイムゾーンを変更して値を取り出すと、取り出された値は格納した値とは異なります。これは、同じタイムゾーンが両方向への変換に使用されなかったために起こります。現在のタイムゾーンは、time_zone システム変数の値として使用できます。詳細は、[セクション5.1.15「MySQL Server でのタイムゾーンのサポート」](#)を参照してください。

MySQL 8.0.19 では、TIMESTAMP および DATETIME の値をテーブルに挿入するときにタイムゾーンオフセットを指定できます。オフセットは、内部スペースを使用せずに日時リテラルの時間部分に追加され、time_zone システム変数の設定に使用されるのと同じ書式を使用しますが、次の例外があります：

- 10 未満の時間値の場合、先行するゼロが必要です。
- 値'-00:00'は拒否されます。
- 'EET'や'Asia/Shanghai'などのタイムゾーン名は使用できません。このコンテキストでは'SYSTEM'も使用できません。

挿入される値は、月、日、またはその両方の部分に対してゼロを持つことはできません。これは、サーバーの SQL モードの設定に関係なく、MySQL 8.0.22 以降で強制されます。

この例では、異なる time_zone 設定を使用してタイムゾーンオフセットを含む日時値を TIMESTAMP および DATETIME カラムに挿入し、それらを取得する方法を示します：

```
mysql> CREATE TABLE ts (  
-> id INTEGER NOT NULL AUTO_INCREMENT PRIMARY KEY,  
-> col TIMESTAMP NOT NULL  
-> ) AUTO_INCREMENT = 1;  
  
mysql> CREATE TABLE dt (  
-> id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,  
-> col DATETIME NOT NULL  
-> ) AUTO_INCREMENT = 1;  
  
mysql> SET @@time_zone = 'SYSTEM';  
  
mysql> INSERT INTO ts (col) VALUES ('2020-01-01 10:10:10'),  
-> ('2020-01-01 10:10:10+05:30'), ('2020-01-01 10:10:10-08:00');  
  
mysql> SET @@time_zone = '+00:00';  
  
mysql> INSERT INTO ts (col) VALUES ('2020-01-01 10:10:10'),  
-> ('2020-01-01 10:10:10+05:30'), ('2020-01-01 10:10:10-08:00');  
  
mysql> SET @@time_zone = 'SYSTEM';  
  
mysql> INSERT INTO dt (col) VALUES ('2020-01-01 10:10:10'),  
-> ('2020-01-01 10:10:10+05:30'), ('2020-01-01 10:10:10-08:00');  
  
mysql> SET @@time_zone = '+00:00';  
  
mysql> INSERT INTO dt (col) VALUES ('2020-01-01 10:10:10'),  
-> ('2020-01-01 10:10:10+05:30'), ('2020-01-01 10:10:10-08:00');  
  
mysql> SET @@time_zone = 'SYSTEM';  
  
mysql> SELECT @@system_time_zone;  
+-----+  
| @@system_time_zone |  
+-----+  
| EST                |  
+-----+  
  
mysql> SELECT col, UNIX_TIMESTAMP(col) FROM dt ORDER BY id;  
+-----+-----+  
| col                | UNIX_TIMESTAMP(col) |  
+-----+-----+
```



```
| 2020-01-01 10:10:10 | 1577891410 |
| 2019-12-31 23:40:10 | 1577853610 |
| 2020-01-01 13:10:10 | 1577902210 |
| 2020-01-01 10:10:10 | 1577891410 |
| 2020-01-01 04:40:10 | 1577871610 |
| 2020-01-01 18:10:10 | 1577920210 |
+-----+-----+
```

```
mysql> SELECT col, UNIX_TIMESTAMP(col) FROM ts ORDER BY id;
```

```
+-----+-----+
| col          | UNIX_TIMESTAMP(col) |
+-----+-----+
| 2020-01-01 10:10:10 | 1577891410 |
| 2019-12-31 23:40:10 | 1577853610 |
| 2020-01-01 13:10:10 | 1577902210 |
| 2020-01-01 05:10:10 | 1577873410 |
| 2019-12-31 23:40:10 | 1577853610 |
| 2020-01-01 13:10:10 | 1577902210 |
+-----+-----+
```

日時値の選択時にオフセットが使用された場合でも、オフセットは表示されません。

サポートされるオフセット値の範囲は、**-14:00** から **+14:00** までです。

タイムゾーンオフセットを含む日時リテラルは、プリペアドステートメントによってパラメータ値として受け入れられます。

SQL モードでこの変換が許可されている場合、無効な **DATE**、**DATETIME** または **TIMESTAMP** 値は適切なタイプ ('0000-00-00'または'0000-00-00 00:00:00') の「ゼロ」値に変換されます。正確な動作は、厳密な SQL モードと **NO_ZERO_DATE** SQL モードのいずれかが有効かどうかによって異なります。[セクション5.1.11「サーバー SQL モード」](#) を参照してください。

MySQL 8.0.22 以降では、次に示すように **CAST()** を **AT TIME ZONE** 演算子とともに使用して取得するときに、**TIMESTAMP** 値を UTC **DATETIME** 値に変換できます：

```
mysql> SELECT col,
> CAST(col AT TIME ZONE INTERVAL '+00:00' AS DATETIME) AS ut
> FROM ts ORDER BY id;
```

```
+-----+-----+
| col          | ut          |
+-----+-----+
| 2020-01-01 10:10:10 | 2020-01-01 15:10:10 |
| 2019-12-31 23:40:10 | 2020-01-01 04:40:10 |
| 2020-01-01 13:10:10 | 2020-01-01 18:10:10 |
| 2020-01-01 10:10:10 | 2020-01-01 15:10:10 |
| 2020-01-01 04:40:10 | 2020-01-01 09:40:10 |
| 2020-01-01 18:10:10 | 2020-01-01 23:10:10 |
+-----+-----+
```

構文およびその他の例の詳細は、[CAST\(\)](#) 関数の説明を参照してください。

MySQL では日付値解釈の特定のプロパティに注意してください。

- MySQL は、文字列として指定された値に、「緩やかな」形式を使用でき、この形式では、どの句読点文字でも日付部分と時間部分の区切り文字として使用できます。場合によっては、この構文は偽りになることがあります。たとえば、'10:11:12'などの値は、:が原因で時間値のように見えますが、日付コンテキストで使用されている場合は'2010-11-12'年として解釈されます。値 '10:45:15' は、'45' が有効な月ではないので、'0000-00-00' に変換されません。

日付および時間の部分と小数秒部分との間の区切り文字として認識される唯一の文字が小数点です。

- サーバーは、月と日の値が、それぞれが 1 から 12 と 1 から 31 の範囲内にあるだけでなく、有効である必要があります。厳密モードが無効になっていると、'2004-04-31' のような無効な日付は '0000-00-00' に変換され、警告メッセージが表示されます。厳密モードが有効なときは、無効な日付によってエラーが発生します。このような日付を許可するには、[ALLOW_INVALID_DATES](#) を有効にします。詳細は、[セクション5.1.11「サーバー SQL モード」](#) を参照してください。
- MySQL は、日または月カラムにゼロを含んだ **TIMESTAMP** 値や、無効な日付の値を受け入れません。SQL モードでこの値が許可されている場合、このルールの唯一の例外は、特別な「ゼロ」値 '0000-00-00 00:00:00' です。

正確な動作は、厳密な SQL モードと `NO_ZERO_DATE` SQL モードのいずれかが有効かどうかによって異なります。[セクション5.1.11「サーバー SQL モード」](#) を参照してください。

- 世紀が不明であるため、2 桁の年の値を含む日付はあいまいです。MySQL は、次のルールを使用して 2 桁の年の値を解釈します:
 - `00-69` の範囲内の年の値は `2000-2069` になります。
 - `70-99` の範囲内の年の値は `1970-1999` になります。

[セクション11.2.8「日付の 2 桁の年」](#) も参照してください。

11.2.3 TIME 型

MySQL では、`TIME` 値が `hh:mm:ss` 形式 (大きい時間値の場合は `hhh:mm:ss` 形式) で取得および表示されます。`TIME` 値の範囲は、`'-838:59:59'` から `'838:59:59'` です。`TIME` 型は、時間 (24 時間以下にする必要があります) を表すだけでなく、経過時間や、2 つのイベント間の時間 (24 時間よりも非常に長くなる場合も、負になる場合もあります) を表すこともできるので、時間の部分は非常に大きくなる可能性があります。

MySQL が `TIME` 値を認識する形式は複数あり、そのいくつかにはマイクロ秒 (6 秒) までの精度で後続の小数秒部分を含めることができます。[セクション9.1.3「日付リテラルと時間リテラル」](#) を参照してください。MySQL の小数秒のサポートの詳細は、[セクション11.2.6「時間値での小数秒」](#) を参照してください。特に、`TIME` カラムに挿入された値の小数部は、破棄されるのではなく格納されます。小数部が含まれている場合、`TIME` 値の範囲は `'-838:59:59.000000'` から `'838:59:59.000000'` です。

`TIME` カラムに省略された値を割り当てる場合は注意してください。MySQL は、コロン付きの省略された `TIME` 値を時間と解釈します。つまり、`'11:12'` は `'00:11:12'` ではなく `'11:12:00'` を意味します。MySQL は、右端の 2 桁が秒を表すという仮定を使用して (つまり、時間としてではなく経過時間として)、コロンのない省略された値を解釈します。たとえば、`'1112'` と `1112` は `'11:12:00'` (11 時 12 分) を表すように見えますが、MySQL では `'00:11:12'` (11 分 12 秒) と解釈されます。同様に、`'12'` や `12` は `'00:00:12'` と解釈されます。

時間部分と小数秒部分との間の区切り文字として認識される唯一の文字が小数点です。

デフォルトでは、`TIME` 範囲外にあるが、それ以外は有効な値は、範囲のもっとも近い終点に切り落とされます。たとえば、`'-850:00:00'` と `'850:00:00'` は、それぞれ `'-838:59:59'` と `'838:59:59'` に変換されます。無効な `TIME` 値は、`'00:00:00'` に変換されます。`'00:00:00'` はそれ自身が有効な `TIME` 値なので、元の値が `'00:00:00'` と指定されたかどうか、無効であったかどうか、テーブルに格納された `'00:00:00'` の値から判断できません。

無効な `TIME` 値の制限を厳しくするには、エラーが発生するように厳密な SQL モードを有効にしてください。[セクション5.1.11「サーバー SQL モード」](#) を参照してください。

11.2.4 YEAR 型

`YEAR` 型は年の値を表すために使用される 1 バイトの型です。これは、4 文字の暗黙的な表示幅で `YEAR` として宣言することも、明示的な表示幅で `YEAR(4)` と同等に宣言することもできます。

注記

MySQL 8.0.19 では、明示的な表示幅を持つ `YEAR(4)` データ型は非推奨であり、将来のバージョンの MySQL ではサポートされなくなる予定です。かわりに、同じ意味を持つ表示幅を指定せずに `YEAR` を使用してください。

MySQL 8.0 では、古いバージョンの MySQL で許可されている 2 桁の `YEAR(2)` データ型はサポートされていません。4 桁の `YEAR` に変換する手順は、[MySQL 5.7 Reference Manual の 2-Digit YEAR\(2\) Limitations and Migrating to 4-Digit YEAR](#) を参照してください。

MySQL では、`YEAR` 値が `1901` から `2155` および `0000` の範囲で `YYYY` 形式で表示されます。

`YEAR` は、次のような様々な形式で入力値を受け入れます:

- `'1901'` から `'2155'` の範囲の 4 桁の文字列として。
- `1901` から `2155` までの範囲の 4 桁の数値として。

- '0'から'99'までの範囲の 1 桁または 2 桁の文字列として。MySQL は、'0' から '69' と '70' から '99' の範囲の値を、2000 から 2069 と 1970 から 1999 の範囲の YEAR 値に変換します。
- 0 から 99 までの範囲の 1 桁または 2 桁の数値として。MySQL は、1 から 69 と 70 から 99 の範囲の値を、2001 から 2069 と 1970 から 1999 の範囲の YEAR 値に変換します。

数値 0 を挿入した結果の表示値は 0000 で、内部値は 0000 です。ゼロを挿入して 2000 として解釈するには、文字列'0'または'00'として指定します。

- NOW() などの YEAR コンテキストで許容される値を返す関数の結果として。

厳密な SQL モードが有効になっていない場合、MySQL は無効な YEAR 値を 0000 に変換します。厳密な SQL モードでは、無効な YEAR 値を挿入しようとするとエラーが発生します。

[セクション11.2.8「日付の 2 桁の年」](#) も参照してください。

11.2.5 TIMESTAMP および DATETIME の自動初期化および更新機能

TIMESTAMP および DATETIME のカラムは、自動的に初期化して現在の日時 (つまり、現在のタイムスタンプ) に更新できます。

テーブル内のあらゆる TIMESTAMP または DATETIME カラムに対して、デフォルト値または自動更新値、あるいはその両方として、現在のタイムスタンプを割り当てることができます。

- 自動初期化されたカラムは、カラムに値を指定しない挿入行に対して現在のタイムスタンプに設定されます。
- 自動更新されたカラムは、行内のほかのカラムの値がその現在の値から変更されると、現在のタイムスタンプに自動的に更新されます。自動更新されたカラムは、ほかのすべてのカラムがその現在の値に設定されていれば、変更されないまま保持されます。ほかのカラムが変更したときに、自動更新したカラムが更新しないようにするには、明示的にこれを現在の値に設定します。ほかのカラムが変更しない場合でも、自動更新カラムを更新するには、明示的にこれを必要な値に設定します (たとえば CURRENT_TIMESTAMP に設定します)。

また、[explicit_defaults_for_timestamp](#) システム変数が無効になっている場合は、NULL 値を許可するように NULL 属性で定義されていないかぎり、NULL 値を割り当てることで、任意の TIMESTAMP (DATETIME 以外) カラムを現在の日時に初期化または更新できます。

自動プロパティを指定するには、カラム定義で DEFAULT CURRENT_TIMESTAMP および ON UPDATE CURRENT_TIMESTAMP 句を使用します。句の順序は関係ありません。両方がカラム定義にある場合、どちらも最初に実行できます。CURRENT_TIMESTAMP のシノニムのいずれも、CURRENT_TIMESTAMP と同じ意味があります。これらは、CURRENT_TIMESTAMP()、NOW()、LOCALTIME、LOCALTIME()、LOCALTIMESTAMP、および LOCALTIMESTAMP() です。

DEFAULT CURRENT_TIMESTAMP および ON UPDATE CURRENT_TIMESTAMP の使用は、TIMESTAMP および DATETIME に固有です。DEFAULT 句を使用して、定数 (非自動) のデフォルト値 (DEFAULT 0 や DEFAULT '2000-01-01 00:00:00' など) を指定することもできます。

注記

次の例では、厳密な SQL モードと NO_ZERO_DATE SQL モードのどちらが有効になっているかに応じて警告またはエラーを生成できるデフォルトの DEFAULT 0 を使用します。TRADITIONAL SQL モードには、厳密モードおよび NO_ZERO_DATE が含まれることに注意してください。 [セクション5.1.11「サーバー SQL モード」](#) を参照してください。

TIMESTAMP または DATETIME カラム定義では、現在のタイムスタンプをデフォルト値と自動更新値の両方に対して指定することも、どちらか一方について指定することも、両方について指定しないこともできます。異なるカラムは、自動プロパティの別々の組み合わせを持つことができます。次のルールは可能性のある場合について記述しています。

- DEFAULT CURRENT_TIMESTAMP と ON UPDATE CURRENT_TIMESTAMP の両方を使用した場合、カラムは、デフォルト値が現在のタイムスタンプになり、現在のタイムスタンプに自動的に更新されます。

```
CREATE TABLE t1 (
  ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  dt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

- **DEFAULT** 句を使用するが **ON UPDATE CURRENT_TIMESTAMP** 句を使用しない場合、カラムには所定のデフォルト値が設定され、現在のタイムスタンプに自動的に更新されません。

デフォルトは、**DEFAULT** 句で **CURRENT_TIMESTAMP** を指定するか定数値を指定するかに応じて異なります。**CURRENT_TIMESTAMP** を使用した場合、デフォルトは現在のタイムスタンプになります。

```
CREATE TABLE t1 (
  ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  dt DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

定数を使用した場合、デフォルトは所定の値になります。この場合、カラムには自動的なプロパティはありません。

```
CREATE TABLE t1 (
  ts TIMESTAMP DEFAULT 0,
  dt DATETIME DEFAULT 0
);
```

- **ON UPDATE CURRENT_TIMESTAMP** 句と定数の **DEFAULT** 句を使用した場合、カラムは、現在のタイムスタンプに自動的に更新され、所定の定数のデフォルト値があります。

```
CREATE TABLE t1 (
  ts TIMESTAMP DEFAULT 0 ON UPDATE CURRENT_TIMESTAMP,
  dt DATETIME DEFAULT 0 ON UPDATE CURRENT_TIMESTAMP
);
```

- **ON UPDATE CURRENT_TIMESTAMP** 句を使用するが **DEFAULT** 句を使用しない場合、カラムは、自動的に現在のタイムスタンプに更新され、そのデフォルト値に現在のタイムスタンプは使用されません。

この場合のデフォルトは型により異なります。**TIMESTAMP** は、**NULL** 属性を使用して定義されていないかぎり (この場合はデフォルトは **NULL** です)、デフォルトは 0 です。

```
CREATE TABLE t1 (
  ts1 TIMESTAMP ON UPDATE CURRENT_TIMESTAMP, -- default 0
  ts2 TIMESTAMP NULL ON UPDATE CURRENT_TIMESTAMP -- default NULL
);
```

DATETIME は、**NOT NULL** 属性で定義されていないかぎり (この場合、デフォルトは 0 です)、デフォルトは **NULL** です。

```
CREATE TABLE t1 (
  dt1 DATETIME ON UPDATE CURRENT_TIMESTAMP, -- default NULL
  dt2 DATETIME NOT NULL ON UPDATE CURRENT_TIMESTAMP -- default 0
);
```

TIMESTAMP および **DATETIME** のカラムには、明示的に指定しないかぎり、自動プロパティはありませんが、この例外があります: [explicit_defaults_for_timestamp](#) システム変数が無効になっている場合、first **TIMESTAMP** カラムに **DEFAULT CURRENT_TIMESTAMP** と **ON UPDATE CURRENT_TIMESTAMP** の両方が含まれます (どちらも明示的に指定されていない場合)。最初の **TIMESTAMP** カラムについて自動プロパティを抑制するには、次のいずれかの戦略を使用します。

- [explicit_defaults_for_timestamp](#) システム変数を有効にします。この場合、自動初期化および自動更新を指定する **DEFAULT CURRENT_TIMESTAMP** 句および **ON UPDATE CURRENT_TIMESTAMP** 句は使用できますが、カラム定義に明示的に含まれていないかぎり、**TIMESTAMP** カラムには割り当てられません。
- または、[explicit_defaults_for_timestamp](#) が無効になっている場合は、次のいずれかを実行します:
 - 定数のデフォルト値を指定する **DEFAULT** 句を含むカラムを定義します。
 - **NULL** 属性を指定します。またこれにより、カラムで **NULL** 値が許可されます。つまり、カラムを **NULL** に設定することによって現在のタイムスタンプを割り当てることができなくなります。**NULL** を割り当てると、カラムは現在のタイムスタンプではなく **NULL** に設定されます。現在のタイムスタンプを割り当てるには、カラムを **CURRENT_TIMESTAMP** または **NOW()** などのシノニムに設定します。

次のテーブル定義を考慮してください。

```
CREATE TABLE t1 (
```

```
ts1 TIMESTAMP DEFAULT 0,  
ts2 TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
ON UPDATE CURRENT_TIMESTAMP);  
CREATE TABLE t2 (  
ts1 TIMESTAMP NULL,  
ts2 TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
ON UPDATE CURRENT_TIMESTAMP);  
CREATE TABLE t3 (  
ts1 TIMESTAMP NULL DEFAULT 0,  
ts2 TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
ON UPDATE CURRENT_TIMESTAMP);
```

テーブルには次のプロパティがあります。

- 各テーブル定義において、最初の **TIMESTAMP** カラムには、自動初期化または更新機能はありません。
- 各テーブルでは、**ts1** カラムで **NULL** 値を処理する方法が異なります。 **t1** の場合、**ts1** は **NOT NULL** であり、これに **NULL** の値を割り当てると、現在のタイムスタンプに設定されます。 **t2** と **t3** の場合、**ts1** では **NULL** を使用でき、これに **NULL** の値を割り当てると、**NULL** に設定されます。
- **t2** と **t3** では、**ts1** のデフォルト値が異なります。 **t2** の場合、**ts1** は、**NULL** を許可するように定義されているので、明示的な **DEFAULT** 句がない場合はデフォルトも **NULL** です。 **t3** の場合、**ts1** は **NULL** を使用できますが、明示的なデフォルトは **0** です。

TIMESTAMP または **DATETIME** カラム定義のいずれかの場所に明示的な小数秒精度値が含まれる場合、カラム定義全体で同じ値を使用する必要があります。次の場合は許可されます。

```
CREATE TABLE t1 (  
ts TIMESTAMP(6) DEFAULT CURRENT_TIMESTAMP(6) ON UPDATE CURRENT_TIMESTAMP(6)  
);
```

次の場合は許可されません。

```
CREATE TABLE t1 (  
ts TIMESTAMP(6) DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP(3)  
);
```

TIMESTAMP の初期化と NULL 属性

[explicit_defaults_for_timestamp](#) システム変数が無効になっている場合、**TIMESTAMP** カラムはデフォルトで **NOT NULL** であり、**NULL** 値を含めることはできず、**NULL** を割り当てると現在のタイムスタンプが割り当てられます。**NULL** を含めるように **TIMESTAMP** カラムを許可するには、**NULL** 属性で明示的に宣言します。この場合、別のデフォルト値を指定する **DEFAULT** 句でオーバーライドされないかぎり、デフォルト値も **NULL** になります。**DEFAULT NULL** を使用すると、デフォルト値として **NULL** を明示的に指定できます。(**NULL** 属性が宣言されていない **TIMESTAMP** カラムの場合、**DEFAULT NULL** は無効です。) **TIMESTAMP** カラムで **NULL** 値を許可する場合、**NULL** を割り当てると、このカラムは現在のタイムスタンプではなく **NULL** に設定されます。

次のテーブルには、**NULL** 値を許可している複数の **TIMESTAMP** カラムが含まれています。

```
CREATE TABLE t  
(  
ts1 TIMESTAMP NULL DEFAULT NULL,  
ts2 TIMESTAMP NULL DEFAULT 0,  
ts3 TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP  
);
```

NULL 値を許可する **TIMESTAMP** カラムは、次のいずれかの状況に当てはまる場合を除き、挿入時に現在のタイムスタンプを取りません。

- デフォルト値が **CURRENT_TIMESTAMP** と定義され、カラムに対して値が指定されていない
- **CURRENT_TIMESTAMP**、または **NOW()** などのそのいずれかのシノニムが明示的にカラムに挿入されている

つまり、**NULL** 値を許可するように定義されている **TIMESTAMP** カラムは、その定義に **DEFAULT CURRENT_TIMESTAMP** が含まれている場合にのみ自動初期化します。

```
CREATE TABLE t (ts TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP);
```

TIMESTAMP カラムで **NULL** 値を許可しているが、定義に **DEFAULT CURRENT_TIMESTAMP** が含まれていない場合、現在の日付と時間に対応する値を明示的に挿入する必要があります。 **t1** および **t2** テーブルに次の定義があるとします。

```
CREATE TABLE t1 (ts TIMESTAMP NULL DEFAULT '0000-00-00 00:00:00');
CREATE TABLE t2 (ts TIMESTAMP NULL DEFAULT NULL);
```

挿入時にどちらかのテーブルの **TIMESTAMP** カラムを現在のタイムスタンプに設定するには、明示的にそのカラムにこの値を割り当てます。 例:

```
INSERT INTO t2 VALUES (CURRENT_TIMESTAMP);
INSERT INTO t1 VALUES (NOW());
```

explicit_defaults_for_timestamp システム変数が有効になっている場合、**TIMESTAMP** カラムで **NULL** 値が許可されるのは、**NULL** 属性で宣言されている場合のみです。また、**TIMESTAMP** カラムでは、**NULL** または **NOT NULL** 属性で宣言されているかどうかにかかわらず、**NULL** を割り当てて現在のタイムスタンプを割り当てることはできません。現在のタイムスタンプを割り当てるには、カラムを **CURRENT_TIMESTAMP** または **NOW()** などのシノニムに設定します。

11.2.6 時間値での小数秒

MySQL では、マイクロ秒 (6 桁) までの精度で、**TIME**、**DATETIME** および **TIMESTAMP** 値の小数秒がサポートされています:

- 小数秒部を含むカラムを定義するには、**type_name(fsp)** の構文を使用します。ここで、**type_name** は **TIME**、**DATETIME**、または **TIMESTAMP** であり、**fsp** は小数秒の精度です。 例:

```
CREATE TABLE t1 (t TIME(3), dt DATETIME(6));
```

fsp 値を指定する場合、0 から 6 の範囲にする必要があります。0 の値は、小数部がないことを表します。省略した場合、デフォルトの精度は 0 です。(これは、以前の MySQL バージョンと互換性を保つため、標準 SQL のデフォルトである 6 とは異なっています。)

- 小数秒の部分を含む **TIME**、**DATE** または **TIMESTAMP** 値を同じタイプのカラムに挿入すると、小数点以下の桁数が少なくなります。次のように作成および移入されたテーブルについて考えてみます:

```
CREATE TABLE fractest( c1 TIME(2), c2 DATETIME(2), c3 TIMESTAMP(2) );
INSERT INTO fractest VALUES
('17:51:04.777', '2018-09-08 17:51:04.777', '2018-09-08 17:51:04.777');
```

時間値は丸めでテーブルに挿入されます:

```
mysql> SELECT * FROM fractest;
+-----+-----+-----+
| c1    | c2          | c3          |
+-----+-----+-----+
| 17:51:04.78 | 2018-09-08 17:51:04.78 | 2018-09-08 17:51:04.78 |
+-----+-----+-----+
```

このような丸め行われたときに、警告やエラーは表示されません。この動作は SQL 標準に従います。

かわりに切捨てを使用して値を挿入するには、**TIME_TRUNCATE_FRACTIONAL** SQL モードを有効にします:

```
SET @@sql_mode = sys.list_add(@@sql_mode, 'TIME_TRUNCATE_FRACTIONAL');
```

この SQL モードを有効にすると、時間値は切捨てとともに挿入されます:

```
mysql> SELECT * FROM fractest;
+-----+-----+-----+
| c1    | c2          | c3          |
+-----+-----+-----+
| 17:51:04.77 | 2018-09-08 17:51:04.77 | 2018-09-08 17:51:04.77 |
+-----+-----+-----+
```

- 時間引数を取る関数は、小数秒を含む値を受け入れます。時間関数からの戻り値には、必要に応じて小数秒が含まれます。たとえば、引数を付けない **NOW()** は、小数部のない現在の日付と時間を返しますが、0 から 6 のオプション引数を取って、その桁数の小数秒部が戻り値に含まれていることを指定します。

- 時間リテラルの構文は、`DATE 'str'`、`TIME 'str'`、および `TIMESTAMP 'str'` の時間値と ODBC 構文同等の値を生み出します。指定されている場合、結果の値には後続の小数秒部分が含まれます。以前は、時間型キーワードは無視され、これらの構造は文字列値を生成していました。標準 SQL と ODBC の日付および時間リテラルを参照してください

11.2.7 日付と時間型間での変換

ある程度まで、ある時間型から別の時間型に値を変換できます。ただし、値の変更や情報の損失が生じることがあります。どの場合でも、時間型間の変換は、変換される型で有効な値の範囲に依存します。たとえば、`DATE`、`DATETIME`、および `TIMESTAMP` 値はすべて、同じセットの形式を使用して指定できますが、すべての型で値の範囲が同じではありません。`TIMESTAMP` 値は、1970 UTC より古い値にしたり、'2038-01-19 03:14:07' UTC より新しい値にしたりできません。つまり、'1968-01-01' などの日付は、`DATE` または `DATETIME` 値としては有効ですが、`TIMESTAMP` 値としては有効ではなく、0 に変換されます。

`DATE` 値の変換:

- `DATE` 値には時間情報が含まれないので、`DATETIME` または `TIMESTAMP` 値に変換すると、'00:00:00' の時間部分が追加されます。
- `TIME` 値への変換は有用ではありません。結果は '00:00:00' になります。

`DATETIME` および `TIMESTAMP` 値の変換:

- `DATE` 値への変換では、小数秒が考慮され、時間部分が丸められます。たとえば、'1999-12-31 23:59:59.499' は '1999-12-31' になり、'1999-12-31 23:59:59.500' は '2000-01-01' になります。
- `TIME` 型には日付情報が含まれないので、`TIME` 値に変換すると日付部分が破棄されます。

`TIME` 値を他の時間型に変換する場合、`CURRENT_DATE()` の値が日付部分に使用されます。`TIME` は (時間ではなく) 経過時間として解釈され、日付に追加されます。これは、時間値が '00:00:00' から '23:59:59' の範囲から外れている場合に、結果の日付部分が現在の日付と異なることを意味します。

現在の日付が '2012-01-01' であるとします。'12:00:00'、'24:00:00'、'-12:00:00' の `TIME` 値は、`DATETIME` または `TIMESTAMP` 値に変換されると、それぞれ '2012-01-01 12:00:00'、'2012-01-02 00:00:00'、'2011-12-31 12:00:00' になります。

`TIME` から `DATE` への変換も同様ですが、結果から時間部分が破棄され、それぞれ '2012-01-01'、'2012-01-02'、'2011-12-31' になります。

明示的な変換を使用して暗黙的な変換をオーバーライドできます。たとえば、`DATE` および `DATETIME` 値の比較で、`DATE` 値は、'00:00:00' の時間部分を追加することにより、強制的に `DATETIME` 型に変更されます。代わりに `DATETIME` 値の時間部分を無視して比較を実行するには、次の方法で `CAST()` 関数を使用します。

```
date_col = CAST(datetime_col AS DATE)
```

`TIME` および `DATETIME` 値の数値形式への変換 (+0 の追加など) は、値に小数秒部分が含まれているかどうかによって異なります。`TIME(N)` または `DATETIME(N)` は、`N` が 0 (または省略) の場合は整数に変換され、`N` が 0 より大きい場合は `N` 小数点を含む `DECIMAL` 値に変換されます:

```
mysql> SELECT CURTIME(), CURTIME()+0, CURTIME(3)+0;
+-----+-----+-----+
| CURTIME() | CURTIME()+0 | CURTIME(3)+0 |
+-----+-----+-----+
| 09:28:00 | 92800 | 92800.887 |
+-----+-----+-----+
mysql> SELECT NOW(), NOW()+0, NOW(3)+0;
+-----+-----+-----+
| NOW() | NOW()+0 | NOW(3)+0 |
+-----+-----+-----+
| 2012-08-15 09:28:00 | 20120815092800 | 20120815092800.889 |
+-----+-----+-----+
```

11.2.8 日付の 2 桁の年

世紀が不明であるため、年が 2 桁の日付値はあいまいです。MySQL では 4 桁を使用して内部的に年が格納されるため、このような値は 4 桁の形式に解釈する必要があります。

`DATETIME`、`DATE`、および `TIMESTAMP` 型では、MySQL は、次のルールを使用して、あいまいな年の値で指定された日付を変換します。

- `00-69` の範囲内の年の値は `2000-2069` になります。
- `70-99` の範囲内の年の値は `1970-1999` になります。

`YEAR` ではルールは同じですが、`YEAR` に挿入された数値 `00` は `2000` ではなく `0000` になります。`YEAR` にゼロを指定し、これを `2000` として解釈させるには、文字列 `'0'` または `'00'` としてこれを指定します。

これらのルールは、データ値が何を表すかを妥当に推測する単なる経験則であることを覚えておいてください。MySQL で使用されるルールで必要な値が生成されない場合は、4 桁の年の値を含む明確な入力を指定する必要があります。

`ORDER BY` では、年が 2 桁の `YEAR` 値が適切にソートされます。

`MIN()` や `MAX()` などの一部の関数は、`YEAR` を数値に変換します。つまり、年が 2 桁の値は、これらの関数では正しく機能しません。この場合の修正は、`YEAR` を 4 桁の年形式に変換することです。

11.3 文字列データ型

文字列データ型は、`CHAR`、`VARCHAR`、`BINARY`、`VARBINARY`、`BLOB`、`TEXT`、`ENUM` および `SET` です。

文字列データ型の記憶域要件の詳細は、[セクション11.7「データ型のストレージ要件」](#) を参照してください。

文字列値を操作する関数については、[セクション12.8「文字列関数および演算子」](#) を参照してください。

11.3.1 文字列データ型の構文

文字列データ型は、`CHAR`、`VARCHAR`、`BINARY`、`VARBINARY`、`BLOB`、`TEXT`、`ENUM` および `SET` です。

MySQL は、文字列カラムを `CREATE TABLE` または `ALTER TABLE` ステートメントで与えられている型とは異なる型に変更することがあります。[セクション13.1.20.7「暗黙のカラム指定の変更」](#) を参照してください。

文字列カラム (`CHAR`、`VARCHAR` および `TEXT` 型) の定義では、MySQL は長さの指定を文字単位で解釈します。バイナリ文字列カラム (`BINARY`、`VARBINARY` および `BLOB` 型) の定義では、MySQL は長さの指定をバイト単位で解釈します。

文字列データ型 `CHAR`、`VARCHAR`、`TEXT` 型、`ENUM`、`SET` および任意のシノニムのカラム定義では、カラムの文字セットおよび照合順序を指定できます:

- `CHARACTER SET` では、文字セットを指定します。必要に応じて、文字セットの照合順序を `COLLATE` 属性とともに他の属性とともに指定できます。例:

```
CREATE TABLE t
(
  c1 VARCHAR(20) CHARACTER SET utf8,
  c2 TEXT CHARACTER SET latin1 COLLATE latin1_general_cs
);
```

このテーブル定義では、`c1` という名前のカラムを作成します。このカラムには、その文字セットのデフォルト照合順序を持つ `utf8` の文字セットと、`latin1` という文字セットおよび大/小文字を区別する (`_cs`) 照合順序を持つ `c2` という名前のカラムがあります。

`CHARACTER SET` と `COLLATE` 属性のいずれかまたは両方が欠落している場合に文字セットと照合順序を割り当てるためのルールは、[セクション10.3.5「カラム文字セットおよび照合順序」](#) で説明されています。

`CHARSET` は `CHARACTER SET` のシノニムです。

- 文字列データ型に `CHARACTER SET binary` 属性を指定すると、対応するバイナリ文字列データ型としてカラムが作成されます: `CHAR` は `BINARY`、`VARCHAR` は `VARBINARY`、`TEXT` は `BLOB` になります。`ENUM` および `SET` データ型では、これは行われず、宣言されたとおりに作成されます。この定義を使用して、テーブルを指定したとします。

```
CREATE TABLE t
(
```

```
c1 VARCHAR(10) CHARACTER SET binary,
c2 TEXT CHARACTER SET binary,
c3 ENUM('a','b','c') CHARACTER SET binary
);
```

結果のテーブルには、この定義が含まれています。

```
CREATE TABLE t
(
  c1 VARBINARY(10),
  c2 BLOB,
  c3 ENUM('a','b','c') CHARACTER SET binary
);
```

- **BINARY** 属性は、カラム文字セット (またはカラム文字セットが指定されていない場合はテーブルのデフォルト文字セット) のバイナリ (`_bin`) 照合順序を指定するための短縮形である非標準の MySQL 拡張機能です。この場合、比較およびソートは数値文字コード値に基づきます。この定義を使用して、テーブルを指定したとします。

```
CREATE TABLE t
(
  c1 VARCHAR(10) CHARACTER SET latin1 BINARY,
  c2 TEXT BINARY
) CHARACTER SET utf8mb4;
```

結果のテーブルには、この定義が含まれています。

```
CREATE TABLE t (
  c1 VARCHAR(10) CHARACTER SET latin1 COLLATE latin1_bin,
  c2 TEXT CHARACTER SET utf8mb4 COLLATE utf8mb4_bin
) CHARACTER SET utf8mb4;
```

MySQL 8.0 では、`utf8mb4` 文字セットに複数の `_bin` 照合順序があるため、**BINARY** 属性のこの非標準の使用はあいまいです。MySQL 8.0.17 では、**BINARY** 属性は非推奨であり、将来のバージョンの MySQL でサポートが削除される予定です。かわりに、明示的な `_bin` 照合を使用するようにアプリケーションを調整する必要があります。

BINARY を使用してデータ型または文字セットを指定する方法は変わりません。

- **ASCII** 属性は **CHARACTER SET latin1** の短縮形です。
- **UNICODE** 属性は **CHARACTER SET ucs2** の短縮形です。

文字カラムの比較およびソートは、カラムに割り当てられた照合に基づきます。**CHAR**、**VARCHAR**、**TEXT**、**ENUM** および **SET** データ型の場合は、バイナリ (`_bin`) 照合順序または **BINARY** 属性を使用してカラムを宣言し、比較およびソートで字句順序ではなく基礎となる文字コード値を使用できます。

MySQL での文字セットの使用の詳細は、[第10章「文字セット、照合順序、Unicode」](#) を参照してください。

- `[NATIONAL] CHAR[(M)] [CHARACTER SET charset_name] [COLLATE collation_name]`

格納時に必ず、指定された長さになるように右側がスペースで埋められる固定長文字列です。**M** はカラムの長さを文字数で表します。**M** の範囲は 0 から 255 です。**M** を省略すると、長さは 1 になります。

注記

PAD_CHAR_TO_FULL_LENGTH SQL モードが有効になっていないがぎり、**CHAR** 値が取り出されるときに末尾のスペースは削除されます。

CHAR は **CHARACTER** の短縮形です。**NATIONAL CHAR** (またはそれと同等の短縮形である **NCHAR**) は、**CHAR** カラムが事前に定義された文字セットを使用する必要があることを定義する標準 SQL の方法です。MySQL では、この事前定義済文字セットとして `utf8` を使用します。[セクション10.3.7「各国語文字セット」](#)。

CHAR BYTE データ型は **BINARY** データ型のエイリアスです。これは互換性機能です。

MySQL では、**CHAR(0)** の型のカラムを作成できます。これは主に、カラムの存在に依存するが、実際にはその値を使用しない古いアプリケーションに準拠する必要がある場合に役立ちます。**CHAR(0)** は、2 つの値しか取れないカラムが必要な場合にも非常に便利です。**CHAR(0) NULL** として定義されたカラムは 1 ビットだけを占め、**NULL** と " (空の文字列) 値だけを取ることができます。

- `[NATIONAL] VARCHAR(M) [CHARACTER SET charset_name] [COLLATE collation_name]`

可変長文字列です。M はカラムの最大長を文字数で表します。M の範囲は 0 から 65,535 です。VARCHAR の有効な最大長は、最大行サイズ (65,535 バイト、すべてのカラムで共有されます) と使用される文字セットによって決まります。たとえば、utf8 の文字は 1 文字につき最大 3 バイトを必要とする場合があるため、utf8 の文字セットを使用する VARCHAR カラムは、最大 21,844 文字になるように宣言できます。セクション 8.4.7 「テーブルカラム数と行サイズの制限」を参照してください。

MySQL は、VARCHAR 値を 1 バイトまたは 2 バイト長のプリフィクスが付いたデータとして格納します。長さプリフィクスは、値に含まれるバイト数を示します。VARCHAR カラムは、格納できる値が 255 バイト以下の場合には 1 バイト長のプリフィクスを使用し、255 バイトより大きい場合は 2 バイト長のプリフィクスを使用します。

注記

MySQL は標準の SQL 仕様に準拠しており、VARCHAR 値の末尾の空白は削除されません。

VARCHAR は CHARACTER VARYING の短縮形です。NATIONAL VARCHAR は、VARCHAR カラムが事前定義された文字セットを使用する必要があることを定義するための標準 SQL の方法です。MySQL では、この事前定義済文字セットとして utf8 を使用します。セクション 10.3.7 「各国語文字セット」。NVARCHAR は NATIONAL VARCHAR の短縮形です。

- `BINARY(M)`

BINARY 型は CHAR 型と似ていますが、非バイナリ文字列ではなく、バイナリバイト文字列を格納します。オプションの長さの M は、カラムの長さをバイト単位で表します。省略すると、M はデフォルトで 1 になります。

- `VARBINARY(M)`

VARBINARY 型は VARCHAR 型と似ていますが、非バイナリ文字列ではなく、バイナリバイト文字列を格納します。M はカラムの最大の長さをバイト数で表します。

- `TINYBLOB`

最大長が $255(2^8 - 1)$ バイトの BLOB カラム。各 TINYBLOB 値は、値のバイト数を示す 1 バイト長のプリフィクスを使用して格納されます。

- `TINYTEXT [CHARACTER SET charset_name] [COLLATE collation_name]`

最大長が $255(2^8 - 1)$ 文字の TEXT カラム。値にマルチバイト文字が含まれる場合、有効な最大長は少なくなります。各 TINYTEXT 値は、値のバイト数を示す 1 バイト長のプリフィクスを使用して格納されます。

- `BLOB(M)`

最大長が $65,535(2^{16} - 1)$ バイトの BLOB カラム。各 BLOB 値は、値のバイト数を示す 2 バイト長のプリフィクスを使用して格納されます。

この型には、オプションの長さ M を指定できます。これが行われた場合、MySQL は M バイトの長さの値を保持するのに十分な最小の BLOB 型としてカラムを作成します。

- `TEXT(M) [CHARACTER SET charset_name] [COLLATE collation_name]`

最大長が $65,535(2^{16} - 1)$ 文字の TEXT カラム。値にマルチバイト文字が含まれる場合、有効な最大長は少なくなります。各 TEXT 値は、値のバイト数を示す 2 バイト長のプリフィクスを使用して格納されます。

この型には、オプションの長さ M を指定できます。これが行われた場合、MySQL は M 文字の長さの値を保持するのに十分な最小 TEXT 型としてカラムを作成します。

- `MEDIUMBLOB`

最大長が $16,777,215(2^{24} - 1)$ バイトの BLOB カラム。各 MEDIUMBLOB 値は、値のバイト数を示す 3 バイト長のプリフィクスを使用して格納されます。

- `MEDIUMTEXT [CHARACTER SET charset_name] [COLLATE collation_name]`

最大長が 16,777,215 ($2^{24} - 1$) 文字の TEXT カラム。値にマルチバイト文字が含まれる場合、有効な最大長は少なくなります。各 MEDIUMTEXT 値は、値のバイト数を示す 3 バイト長のプリフィクスを使用して格納されます。

- **LONGBLOB**

最大長が 4,294,967,295 または 4G バイト ($2^{32} - 1$) バイトの BLOB カラム。LONGBLOB カラムの有効な最大長は、クライアント/サーバープロトコルと使用可能なメモリー内の構成済み最大パケットサイズにより決まります。各 LONGBLOB 値は、値のバイト数を示す 4 バイト長のプリフィクスを使用して格納されます。

- **LONGTEXT [CHARACTER SET charset_name] [COLLATE collation_name]**

最大長が 4,294,967,295 または 4G バイト ($2^{32} - 1$) 文字の TEXT カラム。値にマルチバイト文字が含まれる場合、有効な最大長は少なくなります。LONGTEXT カラムの有効な最大長もまた、クライアント/サーバープロトコルと使用可能メモリー内の構成済みの最大パケットサイズにより決まります。各 LONGTEXT 値は、値のバイト数を示す 4 バイト長のプリフィクスを使用して格納されます。

- **ENUM('value1','value2',...) [CHARACTER SET charset_name] [COLLATE collation_name]**

列挙です。'value1'、'value2'、... の値、NULL、または特殊な " エラー値のリストから選択された値を 1 つだけを持つことができる文字列オブジェクトです。ENUM 値は、内部では整数として表されます。

ENUM カラムには、最大 65,535 個の個別の要素を含めることができます。

個々の ENUM 要素でサポートされる最大長は、 $M \leq 255$ および $(M \times w) \leq 1020$ です。M は要素リテラルの長さ、w は文字セットの最大長文字に必要なバイト数です。

- **SET('value1','value2',...) [CHARACTER SET charset_name] [COLLATE collation_name]**

セットです。ゼロ個以上の値を持つことができる文字列オブジェクトであり、そのそれぞれの値は、'value1'、'value2'、... 値のリストから選択する必要があります。SET 値は整数として内部で表されます。

SET カラムには最大 64 個の個別のメンバーを含めることができます。

個々の SET 要素でサポートされる最大長は、 $M \leq 255$ および $(M \times w) \leq 1020$ です。M は要素リテラルの長さ、w は文字セットの最大長文字に必要なバイト数です。

11.3.2 CHAR および VARCHAR 型

CHAR 型と VARCHAR 型は似ていますが、格納および取得方法が異なります。また、最大長と、末尾のスペースが保持されるかどうかという点でも異なります。

CHAR 型と VARCHAR 型には、格納する最大文字数を表す長さが宣言されています。たとえば、CHAR(30) には最大 30 文字を格納できます。

CHAR カラムの長さは、テーブルを作成したときに宣言した長さに修正されます。この長さには、0 から 255 までの任意の値を指定できます。CHAR 値は格納されると、指定された長さになるように右側がスペースで埋められます。PAD_CHAR_TO_FULL_LENGTH SQL モードが有効になっていないかぎり、CHAR 値が取り出されるときに、末尾のスペースが削除されます。

VARCHAR カラム内の値は可変長の文字列です。長さは 0 から 65,535 までの値で指定できます。VARCHAR カラムの有効な最大長は、最大行サイズ (65,535 バイト、すべてのカラムで共有される) と使用される文字セットによって決まります。セクション 8.4.7 「テーブルカラム数と行サイズの制限」を参照してください。

CHAR とは対照的に、VARCHAR 値は、1 バイトまたは 2 バイト長のプリフィクスの付いたデータとして格納されます。長さプリフィクスは、値に含まれるバイト数を示します。255 バイト以下の値を格納するカラムでは 1 バイト長のプリフィクスを使用し、255 バイトよりも大きい値を格納するカラムでは 2 バイト長のプリフィクスを使用します。

厳密な SQL モードが有効でない場合に、CHAR または VARCHAR カラムにその最大長を超える値を割り当てると、その値はカラムの最大長に合わせて切り捨てられ、警告メッセージが表示されます。スペース以外の文字の切り捨てに関しては、厳密な SQL モードを使用することで、警告ではなくエラーを発生させて、その値の挿入を抑制できます。セクション 5.1.11 「サーバー SQL モード」を参照してください。

VARCHAR カラムの場合、使用している SQL モードに関係なく、カラム長を超える末尾のスペースは挿入前に切り捨てられ、警告メッセージが表示されます。**CHAR** カラムの場合、SQL モードに関係なく、超過した末尾のスペースは通知なしに挿入される値から切り捨てられます。

VARCHAR 値は格納されるときに埋められません。標準 SQL に従い、値を格納し取り出すときに末尾のスペースは保持されます。

次の表は、**CHAR(4)** カラムと **VARCHAR(4)** カラムにさまざまな文字列値を格納した結果を表示して、**CHAR** と **VARCHAR** の違いを示しています (カラムには **latin1** などのシングルバイト文字セットを使用するものとします)。

値	CHAR(4)	必要なストレージ	VARCHAR(4)	必要なストレージ
"	' '	4 バイト	"	1 バイト
'ab'	'ab '	4 バイト	'ab'	3 バイト
'abcd'	'abcd'	4 バイト	'abcd'	5 バイト
'abcdefgh'	'abcd'	4 バイト	'abcd'	5 バイト

テーブルの最後の行に格納されている値が厳密な SQL モードを使用していない場合のみを適用します。厳密モードが有効な場合、カラムの長さを超える値は保管されていませんであり、エラーが発生します。

InnoDB では、768 バイト以上の固定長フィールドが可変長フィールドとしてエンコードされ、オフページに格納できません。たとえば、**utf8mb4** と同様に、文字セットの最大バイト長が 3 より大きい場合、**CHAR(255)** カラムは 768 バイトを超えることがあります。

所定の値が **CHAR(4)** および **VARCHAR(4)** カラムに格納されると、取り出しのときに末尾のスペースが **CHAR** カラムから削除されるので、カラムから取り出された値は必ずしも同じではありません。次の例はこの違いを示しています。

```
mysql> CREATE TABLE vc (v VARCHAR(4), c CHAR(4));
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO vc VALUES ('ab ', 'ab ');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT CONCAT('( ', v, ')'), CONCAT('( ', c, ')') FROM vc;
+-----+-----+
| CONCAT('( ', v, ')') | CONCAT('( ', c, ')') |
+-----+-----+
| (ab )           | (ab)           |
+-----+-----+
1 row in set (0.06 sec)
```

CHAR、**VARCHAR** および **TEXT** カラムの値は、カラムに割り当てられた文字セット照合に従ってソートおよび比較されます。

MySQL 照合には、UCA 9.0.0 以上に基づく Unicode 照合以外の **PAD SPACE** のパッド属性があり、これには **NO PAD** のパッド属性があります。(セクション 10.10.1「Unicode 文字セット」を参照)。

照合のパッド属性を決定するには、**PAD_ATTRIBUTE** カラムを含む **INFORMATION_SCHEMA COLLATIONS** テーブルを使用します。

非バイナリ文字列 (**CHAR**、**VARCHAR** および **TEXT** 値) の場合、文字列照合パッド属性によって、文字列の末尾にある末尾の空白の比較での処理が決定されます。**NO PAD** 照合順序では、他の文字と同様に、比較で末尾のスペースが重要として扱われます。**PAD SPACE** 照合順序では、比較で末尾のスペースは重要ではありません。文字列は末尾のスペースに関係なく比較されます。比較での後続領域の処理を参照してください。サーバーの SQL モードは、末尾の空白に関する比較動作には影響しません。

注記

MySQL の文字セットおよび照合順序の詳細は、第 10 章「文字セット、照合順序、Unicode」を参照してください。ストレージ要件の追加情報については、セクション 11.7「データ型のストレージ要件」を参照してください。

後続パッド文字が削除または比較される場合、一意の値を必要とするインデックスがカラムにあると、後続パッド文字の数のみが異なるカラム値に挿入すると重複キーエラーが発生します。たとえば、テーブルに 'a' が含まれている場合、'a' を格納しようとする、重複キーエラーが発生します。

11.3.3 BINARY および VARBINARY 型

BINARY および **VARBINARY** 型は、非バイナリ文字列ではなくバイナリ文字列を格納する点を除き、**CHAR** および **VARCHAR** と似ています。つまり、文字列ではなくバイト文字列が格納されます。これは、**binary** 文字セットと照合順序があり、比較とソートは値のバイトの数値に基づいていることを意味します。

BINARY および **VARBINARY** の最大許容長は、**CHAR** および **VARCHAR** の場合と同じですが、**BINARY** および **VARBINARY** の長さは文字ではなくバイト単位で測定されます。

BINARY および **VARBINARY** データ型は **CHAR BINARY** および **VARCHAR BINARY** データ型とは異なります。後者の型は、**BINARY** 属性によってカラムがバイナリ文字列カラムとして扱われることはありません。代わりに、カラム文字セット (またはカラム文字セットが指定されていない場合はテーブルのデフォルト文字セット) のバイナリ (**_bin**) 照合順序が使用され、カラム自体にバイナリバイト文字列ではなく非バイナリ文字列が格納されます。たとえば、デフォルトの文字セットが **utf8mb4** の場合、**CHAR(5) BINARY** は **CHAR(5) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin** として扱われます。これは、**binary** 文字セットおよび照合順序を持つ 5 バイトのバイナリ文字列を格納する **BINARY(5)** とは異なります。**binary** 文字セットの **binary** 照合順序と非バイナリ文字セットの **_bin** 照合順序の違いについては、[セクション10.8.5「バイナリ照合順序と_bin 照合順序」](#) を参照してください。

厳密な SQL モードが有効でない場合に、**BINARY** または **VARBINARY** カラムにその最大長を超える値を割り当てると、その値はカラムの最大長に合わせて切り捨てられ、警告メッセージが表示されます。切捨ての場合、(警告ではなく) エラーが発生し、値の挿入を抑制するには、厳密な SQL モードを使用します。[セクション5.1.11「サーバー SQL モード」](#) を参照してください。

BINARY 値は格納されると、特定の長さまで右側がパッド値で埋められます。パッド値は **0x00** (ゼロバイト) です。値は挿入のために **0x00** で右パディングされ、取得のために後続のバイトは削除されません。**ORDER BY** および **DISTINCT** 操作を含むすべてのバイトが比較で重要です。**0x00** と領域の比較は異なり、**0x00** は領域の前にソートされます。

例: **BINARY(3)** カラムの場合、'a' は挿入時に 'a\0' になります。'a\0' は挿入時に 'a\0\0' になります。挿入された両方の値は変更されずに取得されます。

VARBINARY の場合、挿入用のパディングはなく、取得用のバイトは削除されません。**ORDER BY** および **DISTINCT** 操作を含むすべてのバイトが比較で重要です。**0x00** と領域の比較は異なり、**0x00** は領域の前にソートされます。

後続パッドバイトが削除または比較される場合、それらは無視され、カラムに一意の値を必要とするインデックスがあると、後続パッドバイト数のみが異なるカラムに値を挿入すると重複キーエラーが発生します。たとえば、テーブルに 'a' が含まれている場合、'a\0' を格納しようとする、重複キーエラーが発生します。

バイナリデータの格納に **BINARY** データ型を使用する予定であり、取り出した値を格納した値とまったく同じにする必要がある場合は、先行のパディングと削除文字を考慮する必要があります。次の例は、**BINARY** 値の **0x00** パディングによって、カラム値の比較がどのような影響を受けるかについて示しています。

```
mysql> CREATE TABLE t (c BINARY(3));
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t SET c = 'a';
Query OK, 1 row affected (0.01 sec)

mysql> SELECT HEX(c), c = 'a', c = 'a\0' from t;
+-----+-----+-----+
| HEX(c) | c = 'a' | c = 'a\0' |
+-----+-----+-----+
| 610000 | 0 | 1 |
+-----+-----+-----+
1 row in set (0.09 sec)
```

取り出される値を、パディングなしのストレージに指定した値と同じにする必要がある場合は、代わりに **VARBINARY** か、いずれかの **BLOB** データ型を使用することをお勧めします。

11.3.4 BLOB 型と TEXT 型

BLOB はさまざまな容量のデータを保持できる大きなバイナリオブジェクトです。BLOB 型は、TINYBLOB、BLOB、MEDIUMBLOB、および LONGBLOB の 4 つがあります。これらの違いは、保持できる値の最大長だけです。TEXT 型は、TINYTEXT、TEXT、MEDIUMTEXT、および LONGTEXT の 4 つがあります。これらは 4 つの BLOB 型に対応し、最大長とストレージ要件は同じです。セクション 11.7 「データ型のストレージ要件」を参照してください。

BLOB 値はバイナリ文字列 (バイトの文字列) として扱われます。これらには binary 文字セットと照合順序があり、比較とソートはカラム値のバイトの数値に基づきます。TEXT 値は非バイナリ文字列 (文字の文字列) として扱われます。これらには binary 以外の文字セットがあり、値は文字セットの照合に基づいてソートおよび比較されます。

厳密な SQL モードが有効でない場合に、BLOB または TEXT カラムにその最大長を超える値を割り当てると、その値はカラムの最大長に合わせて切り捨てられ、警告メッセージが表示されます。スペース以外の文字の切り捨てに関しては、厳密な SQL モードを使用することで、警告ではなくエラーを発生させて、その値の挿入を抑制できます。セクション 5.1.11 「サーバー SQL モード」を参照してください。

TEXT カラムに挿入される値から、超過した末尾のスペースを切り捨てると、SQL モードには関係なく、常に警告が生成されます。

TEXT および BLOB カラムでは、挿入時にパディングは行われず、選択時にバイトは削除されません。

TEXT カラムにインデックスが設定されている場合、インデックスエントリの比較では末尾がスペースで埋められます。つまり、インデックスに一意の値が必要な場合は、末尾のスペースの数のみが異なる値に対して重複キーエラーが発生します。たとえば、テーブルに 'a' が含まれている場合、'a' を格納しようとする、重複キーエラーが発生します。これは BLOB カラムには当てはまりません。

ほとんどの点で、BLOB カラムを、任意の長さに設定できる VARBINARY カラムと見なすことができます。同様に、TEXT カラムを VARCHAR カラムと見なすことができます。BLOB と TEXT は、次の点で VARBINARY と VARCHAR とは異なっています。

- BLOB と TEXT カラムのインデックスには、インデックスプリフィクス長を指定する必要があります。CHAR と VARCHAR では、プリフィクス長はオプションです。セクション 8.3.5 「カラムインデックス」を参照してください。
- BLOB および TEXT カラムに DEFAULT 値を含めることはできません。

BINARY 属性を TEXT データ型とともに使用する場合、カラムにはカラム文字セットのバイナリ (_bin) 照合順序が割り当てられます。

LONG と LONG VARCHAR は MEDIUMTEXT データ型にマップします。これは互換性機能です。

MySQL Connector/ODBC は BLOB 値を LONGVARBINARY として、TEXT 値を LONGVARCHAR として定義します。

BLOB 値と TEXT 値は非常に長くなる可能性があるため、使用するときには次の制約が生じることがあります。

- ソート時には、カラムの `max_sort_length` バイトだけが使用されます。`max_sort_length` のデフォルト値は 1024 です。サーバーの起動時または実行時に、`max_sort_length` の値を増やすことによって、ソートまたはグループ化に影響するバイトを増やすことができます。すべてのクライアントで `max_sort_length` セッション変数の値を変更できます。

```
mysql> SET max_sort_length = 2000;
mysql> SELECT id, comment FROM t
-> ORDER BY comment;
```

- 一時テーブルを使用して処理されるクエリーの結果に BLOB カラムまたは TEXT カラムのインスタンスがあると、MEMORY ストレージエンジンがこれらのデータ型をサポートしていないので、サーバーはメモリー内ではなくディスク上でテーブルを使用します (セクション 8.4.4 「MySQL での内部一時テーブルの使用」を参照してください)。ディスクの使用はパフォーマンスの低下を伴うので、クエリーの結果に BLOB カラムまたは TEXT カラムを含めるのは必要な場合に限定してください。たとえば、`SELECT *` はすべてのカラムを選択するので使用しないでください。

- BLOB または TEXT オブジェクトの最大サイズはその型で決まりますが、クライアントとサーバー間で実際に送信できる最大値は、使用可能なメモリの容量と通信バッファのサイズで決まります。 `max_allowed_packet` 変数の値を変更することでメッセージバッファサイズを変更できますが、サーバーとクライアントプログラムの両方で変更する必要があります。たとえば、`mysql` と `mysqldump` のどちらを使用しても、クライアント側の `max_allowed_packet` 値を変更できます。 [セクション5.1.1「サーバーの構成」](#)、[セクション4.5.1「mysql — MySQL コマンドラインクライアント」](#)、および [セクション4.5.4「mysqldump — データベースバックアッププログラム」](#) を参照してください。 バケットサイズおよびソートしているデータオブジェクトのサイズを、ストレージ要件と比較することもできます。 [セクション11.7「データ型のストレージ要件」](#) を参照してください。

BLOB 値または TEXT 値はそれぞれ、別々に割り当てられたオブジェクトによって内部的に表現されます。これは、テーブルが開かれるときにカラムごとに一度ストレージが割り当てられる、ほかのすべてのデータ型と対照的です。

メディアファイルなどのバイナリデータを BLOB または TEXT カラムに格納するほうがよい場合もあります。このようなデータの処理には、MySQL の文字列操作関数が役立つことがあります。 [セクション12.8「文字列関数および演算子」](#) を参照してください。 セキュリティーなどの理由のために、通常は、アプリケーションユーザーに FILE 権限を与えるのではなく、アプリケーションコードを使用して実行することをお勧めします。 MySQL フォーラム (<http://forums.mysql.com/>) では、さまざまな言語やプラットフォームの詳細について話し合うことができます。

11.3.5 ENUM 型

ENUM は、テーブル作成時にカラム仕様に明示的に列挙された、許可されている値のリストから選択された値を持つ文字列オブジェクトです。

ENUM 型の構文および長さの制限については、 [セクション11.3.1「文字列データ型の構文」](#) を参照してください。

ENUM タイプには、次の利点があります：

- 指定可能な値のセットがカラムで制限されている状況でのコンパクトなデータストレージ。 入力値として指定した文字列は自動的に数値としてエンコードされます。 ENUM タイプの記憶域要件については、 [セクション11.7「データ型のストレージ要件」](#) を参照してください。
- 読みやすいクエリーと出力。 数値は、クエリー結果で対応する文字列に戻されます。

また、次のような考慮が必要な問題が生じる可能性があります。

- [列挙の制限](#) で説明しているように、数値のように見える列挙値を作成した場合、リテラル値とその内部インデックス番号を混同しやすくなります。
- [列挙のソート](#) で説明しているように、ORDER BY 句で ENUM カラムを使用するには特に注意が必要です。
- [ENUM カラムの作成と使用](#)
- [列挙リテラルのインデックス値](#)
- [列挙リテラルの処理](#)
- [空または NULL の列挙値](#)
- [列挙のソート](#)
- [列挙の制限](#)

ENUM カラムの作成と使用

列挙値は引用符で囲んだ文字列リテラルにする必要があります。たとえば、次のように ENUM カラムを持つテーブルを作成できます。

```
CREATE TABLE shirts (  
  name VARCHAR(40),  
  size ENUM('x-small', 'small', 'medium', 'large', 'x-large')  
);  
INSERT INTO shirts (name, size) VALUES ('dress shirt', 'large'), ('t-shirt', 'medium'),  
('polo shirt', 'small');  
SELECT name, size FROM shirts WHERE size = 'medium';
```

```
+-----+-----+
| name | size |
+-----+-----+
| t-shirt | medium |
+-----+-----+
UPDATE shirts SET size = 'small' WHERE size = 'large';
COMMIT;
```

'medium' の値を持つ 100 万個の行をこのテーブルに挿入するには、100 万バイトのストレージが必要ですが、実際の文字列 'medium' を `VARCHAR` カラムに格納した場合は、600 万バイト必要になります。

列挙リテラルのインデックス値

それぞれの列挙値にはインデックスが設定されています。

- カラム仕様にリストされている要素には、1 から始まるインデックス番号が割り当てられています。
- 空の文字列エラー値のインデックス値は 0 です。つまり、次の `SELECT` ステートメントを使用して、無効な `ENUM` 値が割り当てられた行を検索できます。

```
mysql> SELECT * FROM tbl_name WHERE enum_col=0;
```

- `NULL` 値のインデックスは `NULL` です。
- ここでの「インデックス」という語は、列挙値のリスト内での位置を示します。これは、テーブルインデックスとはまったく関係ありません。

たとえば、`ENUM('Mercury', 'Venus', 'Earth')` と指定されたカラムには、次に示すどの値でも含めることができます。それぞれの値のインデックスも示しています。

値	インデックス
<code>NULL</code>	<code>NULL</code>
<code>"</code>	0
<code>'Mercury'</code>	1
<code>'Venus'</code>	2
<code>'Earth'</code>	3

`ENUM` カラムには、最大 65,535 個の個別の要素を含めることができます。

`ENUM` 値を数値コンテキストで取得した場合、カラム値のインデックスが返されます。たとえば、次のように `ENUM` カラムから数値を取得できます。

```
mysql> SELECT enum_col+0 FROM tbl_name;
```

数値引数を取る `SUM()` や `AVG()` などの関数は、必要に応じて引数を数値にキャストします。`ENUM` 値の計算にはインデックス番号が使用されます。

列挙リテラルの処理

テーブルが作成されるときに、テーブル定義内の `ENUM` メンバー値から末尾のスペースが自動的に削除されます。

`ENUM` カラムに格納された値は、取得されたときに、カラム定義で使用された大文字/小文字で表示されます。`ENUM` カラムには文字セットと照合順序を割り当てられています。バイナリ照合順序、または大文字と小文字を区別する照合順序の場合、カラムに値を割り当てるときに、大文字/小文字が考慮されます。

`ENUM` カラムに数字を格納すると、その数字は指定可能な値のインデックスとして扱われ、格納された値がそのインデックスを持つ列挙メンバーとなります。(ただし、これはすべての入力を文字列として扱う `LOAD DATA` では機能しません。) 数値が引用符で囲まれている場合、列挙値のリストに一致する文字列がなければ、そのままインデックスとして解釈されます。これらの理由により、`ENUM` カラムを数字のように見える列挙値で定義することは、混乱を招きやすくなるのでお勧めできません。たとえば、次のカラムには '0'、'1'、および '2' の文字列値を持つ列挙メンバーが指定されていますが、数値インデックス値は 1、2、および 3 です。

```
numbers ENUM('0','1','2')
```

2 を格納すると、それはインデックス値と解釈され、'1' (インデックス 2 の値) になります。'2' を格納すると、それは列挙値と一致するので、'2' として格納されます。'3' を格納すると、どの列挙値とも一致しないのでインデックスとして扱われ、'2' (インデックス 3 の値) になります。

```
mysql> INSERT INTO t (numbers) VALUES(2),(2),(3);
mysql> SELECT * FROM t;
+-----+
| numbers |
+-----+
| 1       |
| 2       |
| 2       |
+-----+
```

ENUM カラムのすべての指定可能な値を特定するには、`SHOW COLUMNS FROM tbl_name LIKE 'enum_col'` を使用して、出力の `Type` カラム内の ENUM 定義を構文解析します。

C API では、ENUM 値は文字列として返されます。結果セットのメタデータを使用してこれらをほかの文字列から区別する方法については、[C API Basic Data Structures](#) を参照してください。

空または NULL の列挙値

以下のような特定の状況下では、列挙値は、空の文字列 (") や NULL になることもあります。

- ENUM に無効な値 (つまり、許可された値のリストに存在しない文字列) を挿入すると、特殊なエラー値として空の文字列が代わりに挿入されます。この文字列は、この文字列に 0 の数値が含まれていることで、「通常」の空の文字列と区別できます。列挙値の数値インデックスの詳細は、[列挙リテラルのインデックス値](#) を参照してください。

厳密な SQL モードが有効な場合は、無効な ENUM 値を挿入しようとするエラーが発生します。

- ENUM カラムが NULL を許可するように宣言されている場合、NULL 値は、そのカラムに対して有効な値であり、デフォルト値は NULL になります。ENUM カラムが NOT NULL として宣言されている場合、デフォルト値は許可されている値のリストの最初の要素になります。

列挙のソート

ENUM 値は、インデックス番号に基づいてソートされますが、この数値は、カラム仕様で列挙メンバーがリストされていた順序に従います。たとえば、ENUM('b', 'a') の場合、'b' は 'a' の前にソートされます。空の文字列は空ではない文字列の前にソートされ、NULL 値はその他のすべての列挙値の前にソートされます。

ENUM カラムで ORDER BY 句の使用時に予想外の結果になることを回避するには、次のいずれかの手法を使用します。

- アルファベット順で ENUM リストを指定します。
- ORDER BY CAST(col AS CHAR) または ORDER BY CONCAT(col) をコード化することにより、カラムがインデックス番号ではなく、辞書順でソートされることを確認します。

列挙の制限

列挙値は、文字列値に評価されるものであっても、式にはできません。

たとえば、次の CREATE TABLE ステートメントは、CONCAT 関数を列挙値の構築に使用できないので、機能しません。

```
CREATE TABLE sizes (
  size ENUM('small', CONCAT('med','ium'), 'large')
);
```

ユーザー変数を列挙値として使用することもできません。次のステートメントのペアは機能しません。

```
SET @mysize = 'medium';
CREATE TABLE sizes (
```

```
size ENUM('small', @mysize, 'large')
);
```

数字を列挙値として使用しないことを強くお勧めします。これは、適切な **TINYINT** または **SMALLINT** 型よりもストレージを節約するわけでもなく、**ENUM** 値を間違えて引用符で囲んだ場合には、文字列とベースになる数値とを混同しやすくなる (同じでない場合もあります) からです。数字を列挙値として使用する場合は、必ず引用符で囲んでください。引用符を省略した場合は、その数字はインデックスと見なされます。 [列挙リテラルの処理](#) を参照して、引用符で囲まれた数字でも間違えて数字のインデックス値として使用されるか場合について確認してください。

定義の中に重複した値が含まれていると、警告 (厳密な SQL モードが有効になっている場合はエラー) が発生します。

11.3.6 SET 型

SET は、ゼロ以上の値を取ることができる文字列オブジェクトであり、それぞれの値は、テーブルの作成時に指定された許可される値のリストから選択する必要があります。 **SET** カラム値が複数のセットメンバーで構成される場合は、各メンバーはカンマ (,) で区切って指定されます。 このため、**SET** メンバーの値自体にはカンマを含めないでください。

たとえば、**SET('one', 'two') NOT NULL** として指定したカラムは、次に示す値のいずれかを取ります。

```
"
'one'
'two'
'one,two'
```

SET カラムには最大 64 個の個別のメンバーを含めることができます。

定義の中に重複した値が含まれていると、警告 (厳密な SQL モードが有効になっている場合はエラー) が発生します。

テーブルが作成されるときに、テーブル定義内の **SET** メンバー値から末尾のスペースが自動的に削除されます。

SET タイプの記憶域要件については、 [文字列型の格納要件](#) を参照してください。

SET 型の構文および長さの制限については、 [セクション11.3.1「文字列データ型の構文」](#) を参照してください。

SET カラムに格納された値は、取得されるときに、カラム定義で使用されていた大文字/小文字で表示されます。 **SET** カラムには、文字セットと照合順序を割り当てることができます。バイナリ照合順序、または大文字と小文字を区別する照合順序の場合、カラムに値を割り当てるときに、大文字/小文字が考慮されます。

MySQL は、最初のセットメンバーに対応する格納値の低位ビットを使用して **SET** 値を数値で格納します。 **SET** 値を数値コンテキストで取得した場合、その取得された値には、カラム値を構成するセットメンバーに対応するビットセットが含まれます。 たとえば、次のように **SET** カラムから数値を取得できます。

```
mysql> SELECT set_col+0 FROM tbl_name;
```

メンバーが **SET** カラムに格納されると、その数字のバイナリ表現に設定されているビットからカラム値のセットメンバーが特定されます。 カラムが **SET('a','b','c','d')** として指定されている場合、セットメンバーは次の 10 進値と 2 進値を持ちます。

SET メンバー	10 進値	2 進値
'a'	1	0001
'b'	2	0010
'c'	4	0100
'd'	8	1000

このカラムに 9 の値を割り当てた場合、2 進数では 1001 となるため、**SET** 値の最初と 4 番目のメンバーである 'a' と 'd' が選択され、結果として得られる値は 'a,d' になります。

1 つ以上の **SET** 要素を含む値には、値を挿入するときに要素がどの順序でリストされているかは関係ありません。 また、所定の要素が値の中で何回リストされているかも関係ありません。 あとから値を取得するときに、値内のそれぞ

れの要素は、テーブル作成時に指定された順序に従って、一度表示されます。カラムが `SET('a','b','c','d')` として指定されているとします:

```
mysql> CREATE TABLE myset (col SET('a', 'b', 'c', 'd'));
```

'a,d'、'd,a'、'a,d,d'、'a,d,a'、および 'd,a,d' の値を挿入した場合、

```
mysql> INSERT INTO myset (col) VALUES
-> ('a,d'), ('d,a'), ('a,d,a'), ('a,d,d'), ('d,a,d');
Query OK, 5 rows affected (0.01 sec)
Records: 5 Duplicates: 0 Warnings: 0
```

これらの値はすべて、取得されるときに 'a,d' と表示されます。

```
mysql> SELECT col FROM myset;
+-----+
| col |
+-----+
| a,d |
| a,d |
| a,d |
| a,d |
| a,d |
+-----+
5 rows in set (0.04 sec)
```

サポートされていない値に `SET` カラムを設定すると、その値は無視され警告が表示されます。

```
mysql> INSERT INTO myset (col) VALUES ('a,d,d,s');
Query OK, 1 row affected, 1 warning (0.03 sec)

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Warning | 1265 | Data truncated for column 'col' at row 1 |
+-----+-----+-----+
1 row in set (0.04 sec)

mysql> SELECT col FROM myset;
+-----+
| col |
+-----+
| a,d |
| a,d |
| a,d |
| a,d |
| a,d |
| a,d |
+-----+
6 rows in set (0.01 sec)
```

厳密な SQL モードが有効な場合、無効な `SET` 値を挿入しようとするエラーが発生します。

`SET` 値は数値でソートされます。 `NULL` 値は非 `NULL SET` 値の前にソートされます。

数値引数を取る `SUM()` や `AVG()` などの関数は、必要に応じて引数を数値にキャストします。 `SET` 値の場合は、キャスト操作によって数値が使用されます。

通常は、`FIND_IN_SET()` 関数が `LIKE` 演算子を使用して `SET` 値を検索します。

```
mysql> SELECT * FROM tbl_name WHERE FIND_IN_SET('value',set_col)>0;
mysql> SELECT * FROM tbl_name WHERE set_col LIKE '%value%';
```

最初のステートメントは `set_col` が `value` セットメンバーを含む行を見つけます。2 番目も似ていますが、同じではありません。ほかのセットメンバーの部分文字列としてであっても、`set_col` が `value` を含む行を見つけます。

次のステートメントも使用できます。

```
mysql> SELECT * FROM tbl_name WHERE set_col & 1;
```



```
mysql> SELECT * FROM tbl_name WHERE set_col = 'val1,val2';
```

これらのうち最初のステートメントが最初のセットメンバーを含む値を探します。2番目のステートメントは正確に一致する値を探します。2番目の型は慎重に比較してください。セット値を 'val1,val2' と比較すると、値を 'val2,val1' と比較した場合は異なる結果が返されます。カラム定義にリストされている順序どおりに値を指定する必要があります。

SET カラムの指定可能な値をすべて特定するには、`SHOW COLUMNS FROM tbl_name LIKE set_col` を使用して、出力の `Type` カラム内の SET 定義を構文解析します。

C API では、SET 値は文字列として返されます。結果セットのメタデータを使用してこれらをほかの文字列から区別する方法については、[C API Basic Data Structures](#) を参照してください。

11.4 空間データ型

[Open Geospatial Consortium](#) (OGC) は、公開されている概念ソリューションの開発に参加している 250 を超える企業、機関および大学の国際コンソーシアムであり、空間データを管理するあらゆる種類のアプリケーションで役立ちます。

空間データをサポートするように SQL RDBMS を拡張するための複数の概念的な方法を提案したドキュメントとして、Open Geospatial Consortium から「[OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 2: SQL option](#)」が発行されています。この仕様は、OGC の web サイト (<http://www.opengeospatial.org/standards/sfs>) から入手できます。

MySQL は、OGC の仕様書に従って、ジオメトリ型を含む SQL 環境のサブセットとして空間拡張を実装しています。この用語は、一連のジオメトリ型で拡張された SQL 環境を意味しています。ジオメトリ値を含む SQL カラムは、ジオメトリ型のカラムとして実装されています。仕様書では、一連の SQL ジオメトリ型のほか、ジオメトリ値を作成し分析するためにこれらの型に対して行われる関数について説明しています。

MySQL 空間拡張により、地理的特性の生成、ストレージ、および分析が可能になります。

- 空間値を表すデータ型
- 空間値を操作する関数
- 空間カラムへのアクセス時間を改善するための空間インデックス設定

空間データ型および空間関数は、[MyISAM](#)、[InnoDB](#)、[NDB](#) テーブルおよび [ARCHIVE](#) テーブルで使用できます。空間カラムのインデックス付けの場合、[MyISAM](#) および [InnoDB](#) では [SPATIAL](#) インデックスと [SPATIAL](#) 以外のインデックスの両方がサポートされます。その他のストレージエンジンは、[セクション 13.1.15 「CREATE INDEX ステートメント」](#) で説明しているように、非 [SPATIAL](#) インデックスをサポートします。

地理的特性とは、位置を特定できる世界中のあらゆるもののことです。特性は次のいずれかになります。

- 実体。山、池、都市など。
- 領域。町の区域や熱帯地域など。
- 定義可能な位置。2つの道路が交差する特定の場所となる交差点など。

ドキュメントによっては、地理空間特性という用語を地理的特性の意味で使用している場合もあります。

ジオメトリも地理的特性を表す用語です。ジオメトリという用語はもともと、地球の測量を意味していました。地図作成者が世界のマッピングに使用するジオメトリ特性を指す別の意味は、地図作成の分野からのものです。

ここでの説明では、地理的特性、地理空間特性、特性、ジオメトリの用語をシノニムと見なします。もっともよく使用される用語はジオメトリであり、位置を特定できる世界中のあらゆるものを表す点または点の集合として定義されています。

次の資料では次のトピックを取り上げます。

- MySQL モデルに実装された空間データ型

- OpenGIS ジオメトリモデルでの空間拡張の基本
- 空間データを表現するためのデータ形式
- MySQL で空間データを使用する方法
- 空間データのインデックスの使用法
- OpenGIS 仕様と MySQL 実装との差異

空間データを演算する関数の詳細は、[セクション12.17「空間分析関数」](#)を参照してください。

追加のリソース

空間操作の MySQL 実装では、次の標準が重要です:

- SQL/MM パート 3: Spatial.
- [Open Geospatial Consortium](#) は、空間データをサポートするために SQL RDBMS を拡張するいくつかの概念的な方法を提案するドキュメントである OpenGIS® Implementation Standard for Geographic information を公開します。特定の簡易機能アクセスを参照してください - パート 1: 共通アーキテクチャおよび単純な機能アクセス - パート 2: SQL オプション。Open Geospatial Consortium (OGC) は、<http://www.opengeospatial.org/> の web サイトを管理します。仕様書は <http://www.opengeospatial.org/standards/sfs> で入手できます。ここでの資料に関連した追加情報が用意されています。
- [spatial reference system \(SRS\)](#) 定義の文法は、「OpenGIS 実装仕様: 座標変換サービス」, Revision 1.00, OGC 01-009, January 12, 2001, Section 7.2 で定義されている文法に基づいています。この仕様は、<http://www.opengeospatial.org/standards/ct> で入手できます。MySQL に実装されている SRS 定義の仕様との相違点については、[セクション13.1.19「CREATE SPATIAL REFERENCE SYSTEM ステートメント」](#)を参照してください。

MySQL に対する空間拡張の使用について質問や関心がある場合は、GIS フォーラム (<https://forums.mysql.com/list.php?23>) で議論できます。

11.4.1 空間データ型

MySQL には、OpenGIS クラスに対応する空間データ型があります。これらのタイプの基礎は、[セクション11.4.2「OpenGIS ジオメトリモデル」](#)で説明されています。

空間データ型の中には、単一のジオメトリ値を保持するものがあります:

- GEOMETRY
- POINT
- LINESTRING
- POLYGON

GEOMETRY にはどの型のジオメトリ値でも格納できます。その他の単一値型 (POINT、LINESTRING、および POLYGON) では、特定のジオメトリ型に値が制限されます。

他の空間データ型には、次の値のコレクションが保持されます:

- MULTIPOINT
- MULTILINESTRING
- MULTIPOLYGON
- GEOMETRYCOLLECTION

GEOMETRYCOLLECTION には、任意の型のオブジェクトのコレクションを格納できます。他のコレクション型 (MULTIPOINT、MULTILINESTRING および MULTIPOLYGON) では、コレクションメンバーは特定のジオメトリタイプを持つメンバーに制限されます。

例: 任意のジオメトリタイプの値を格納できる `g` という名前のカラムを持つ `geom` という名前のテーブルを作成するには、次のステートメントを使用します:

```
CREATE TABLE geom (g GEOMETRY);
```

空間データ型のカラムには、カラムに格納されている値の空間参照システム (SRS) を明示的に示す `SRID` 属性を指定できます。例:

```
CREATE TABLE geom (  
  p POINT SRID 0,  
  g GEOMETRY NOT NULL SRID 4326  
);
```

`SPATIAL` インデックスは、`NOT NULL` で特定の `SRID` を持つ空間カラムに作成できるため、そのカラムをインデックス付けする場合は、`NOT NULL` および `SRID` 属性を使用して宣言します:

```
CREATE TABLE geom (g GEOMETRY NOT NULL SRID 4326);
```

InnoDB テーブルでは、デカルトおよび地理的 SRS の `SRID` 値が許可されます。MyISAM テーブルでは、デカルト SRS の `SRID` 値が許可されません。

`SRID` 属性を使用すると、空間カラム `SRID` が制限され、次のような影響があります:

- カラムには、指定した `SRID` の値のみを含めることができます。 `SRID` が異なる値を挿入しようとすると、エラーが発生します。
- オプティマイザは、カラムにつけられた `SPATIAL` インデックスを使用できます。 [セクション8.3.3「SPATIAL インデックス最適化」](#) を参照してください。

`SRID` 属性のない空間カラムは、`SRID` に制限されず、`SRID` の値を受け入れます。ただし、オプティマイザは、カラム定義が `SRID` 属性を含むように変更されるまで、`SPATIAL` インデックスを使用できません。これには、すべての値が同じ `SRID` を持つように、最初にカラムの内容を変更する必要がある場合があります。

MySQL で空間データ型を使用する方法を示すその他の例は、[セクション11.4.6「空間カラムの作成」](#) を参照してください。空間参照システムの詳細は、[セクション11.4.5「空間参照システムのサポート」](#) を参照してください。

11.4.2 OpenGIS ジオメトリモデル

OGC のジオメトリ型を含む SQL 環境で提案されている一連のジオメトリ型は、OpenGIS ジオメトリモデルに基づいています。このモデルの各ジオメトリオブジェクトには、次のような一般的なプロパティがあります。

- オブジェクトが定義されている座標空間を記述する空間参照システムに関連付けられます。
- 特定のジオメトリクラスに属しています。

11.4.2.1 ジオメトリクラスの階層

ジオメトリクラスの階層は次のように定義されています。

- `Geometry` (インスタンス化不可能)
 - `Point` (インスタンス化可能)
 - `Curve` (インスタンス化不可能)
 - `LineString` (インスタンス化可能)
 - `Line`
 - `LinearRing`
 - `Surface` (インスタンス化不可能)
 - `Polygon` (インスタンス化可能)

- [GeometryCollection](#) (インスタンス化可能)
 - [MultiPoint](#) (インスタンス化可能)
 - [MultiCurve](#) (インスタンス化不可能)
 - [MultiLineString](#) (インスタンス化可能)
 - [MultiSurface](#) (インスタンス化不可能)
 - [MultiPolygon](#) (インスタンス化可能)

インスタンス化不可能なクラスのオブジェクトは作成できません。インスタンス化可能なクラスのオブジェクトは作成できます。どのクラスもプロパティを持っていますが、インスタンス化可能なクラスはさらに表明 (有効なクラスインスタンスを定義するルール) も持つことができます。

[Geometry](#) は基本クラスです。これは抽象クラスです。 [Geometry](#) のインスタンス化可能なサブクラスは、2次元座標空間内に存在する 0次元、1次元、および2次元のジオメトリオブジェクトに限定されます。インスタンス化可能なジオメトリクラスはすべて、ジオメトリクラスの有効なインスタンスが位相的に閉じている (つまり、定義されたすべてのジオメトリに境界が含まれる) ように定義されています。

[Geometry](#) 基本クラスには、[Point](#)、[Curve](#)、[Surface](#)、および [GeometryCollection](#) のサブクラスがあります。

- [Point](#) は 0次元のオブジェクトを表します。
- [Curve](#) は 1次元のオブジェクトを表し、そのサブクラス [LineString](#) は、[Line](#) と [LinearRing](#) をサブクラスに持ちます。
- [Surface](#) は 2次元のオブジェクト用に設計されたもので、[Polygon](#) をサブクラスに持ちます。
- [GeometryCollection](#) には [MultiPoint](#)、[MultiLineString](#)、[MultiPolygon](#) という 0、1、2次元の特殊化コレクションクラスが用意されており、それぞれ [Points](#)、[LineStrings](#)、[Polygons](#) のコレクションに対応するジオメトリをモデル化しています。 [MultiCurve](#) と [MultiSurface](#) は、このコレクションインタフェースを汎化して [Curves](#) および [Surfaces](#) を処理できるよう抽象スーパークラスとして導入されたものです。

[Geometry](#)、[Curve](#)、[Surface](#)、[MultiCurve](#)、および [MultiSurface](#) は、インスタンス化不可能なクラスとして定義されています。これらはサブクラスに共通する一連のメソッドを定義しており、今後の拡張に含められます。

[Point](#)、[LineString](#)、[Polygon](#)、[GeometryCollection](#)、[MultiPoint](#)、[MultiLineString](#)、および [MultiPolygon](#) はインスタンス化可能なクラスです。

11.4.2.2 Geometry クラス

[Geometry](#) は階層のルートクラスです。これはインスタンス化不可能なクラスですが、次のリストに説明しているように、[Geometry](#) サブクラスのいずれかから作成したすべてのジオメトリ値に共通である多数のプロパティがあります。個々のサブクラスも独自のプロパティを備えています。これについては後述します。

[Geometry](#) のプロパティ

ジオメトリ値に含まれるプロパティは次のとおりです。

- その型。各ジオメトリは、階層内のインスタンス化可能クラスのいずれかに属します。
- SRID または空間参照識別子。この値は、ジオメトリオブジェクトが定義されている座標空間を記述する空間参照システムに関連付けられているジオメトリを識別します。

MySQL の SRID 値は、ジオメトリ値に関連付けられた整数です。使用可能な SRID の最大値は $2^{32}-1$ です。より大きな値が指定されると、低位の 32ビットだけが使用されます。

SRID 0 は、軸に単位が割り当てられていない無限平坦なデカルト平面を表します。SRID 0 の動作を保証するには、SRID 0 を使用してジオメトリ値を作成します。SRID 0 は、SRID が指定されていない場合の新しいジオメトリ値のデフォルトです。

複数のジオメトリ値の計算では、すべての値が同じ SRID を持つ必要があり、そうでない場合はエラーが発生します。

- 空間参照システムでの座標で、倍精度 (8 バイト) 数値で表されます。空でないジオメトリには必ず、(X,Y) 座標ペアが少なくとも 1 つ含まれます。空のジオメトリには座標は含まれません。

座標は SRID に対する相対的なものです。たとえば、座標系が異なると、平面座標系上の距離と測地システム上の距離 (地球表面上の座標) が異なるため、オブジェクトが同じ座標を持つ場合でも、2 つのオブジェクト間の距離が異なる場合があります。

- 内部、境界、外部。

ジオメトリは必ず、ある位置の領域を占有します。ジオメトリの外部とは、そのジオメトリによって占有されていないすべての領域のことです。内部とは、そのジオメトリによって占有されている領域のことです。境界とは、ジオメトリの内部と外部が接する部分のことです。

- その MBR (最小境界矩形)、またはエンベロープ。これは範囲を規定するジオメトリであり、次のように最小および最大の (X,Y) 座標から形成されます。

```
((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
```

- 値が単純である、単純でないのいずれであるか。 [LineString](#)、[MultiPoint](#)、[MultiLineString](#) の型のジオメトリ値は「単純である」、「単純でない」のいずれかになります。「単純である」、「単純でない」のいずれであるかの表明は、型ごとに決定されます。
- 値が閉じている、閉じていないのいずれであるか。 [LineString](#)、[MultiString](#) の型のジオメトリ値は「閉じている」、「閉じていない」のいずれかになります。「閉じている」、「閉じていない」のいずれであるかの表明は、型ごとに決定されます。
- 値が空である、空でないのいずれであるか。点を 1 つも含まないジオメトリは空です。空のジオメトリの外部、内部、および境界は定義されていません (つまり、それらは `NULL` 値で表されます)。空のジオメトリは、常に単純で面積が 0 になるように定義されています。
- その次元。ジオメトリには -1、0、1、または 2 の次元があります。
 - -1 は、空のジオメトリを表します。
 - 0 は長さも面積も持たないジオメトリを表します。
 - 1 は、長さがゼロ以外で面積がゼロのジオメトリを表します。
 - 2 は、面積がゼロ以外のジオメトリを表します。

[Point](#) オブジェクトの次元は 0 です。 [LineString](#) オブジェクトの次元は 1 です。 [Polygon](#) オブジェクトの次元は 2 です。 [MultiPoint](#)、[MultiLineString](#)、および [MultiPolygon](#) オブジェクトの次元は、構成要素の次元と同じになります。

11.4.2.3 Point クラス

[Point](#) は、座標空間内の単一の位置を表すジオメトリです。

[Point](#) の例

- 多数の都市を含む大規模な世界地図を想像してください。 [Point](#) オブジェクトは各都市を表すことができます。
- 市内地図で、 [Point](#) オブジェクトはバス停を表すことができます。

[Point](#) のプロパティ

- X 座標値。
- Y 座標値。
- [Point](#) は 0 次元のジオメトリとして定義されています。

- `Point` の境界は空セットになります。

11.4.2.4 Curve クラス

`Curve` は 1 次元のジオメトリであり、通常は一連の点で表されます。点の間の補間方法は、`Curve` の特定のサブクラスで定義されています。`Curve` はインスタンス化不可能なクラスです。

`Curve` のプロパティー

- `Curve` はその点の座標を持ちます。
- `Curve` は 1 次元のジオメトリとして定義されています。
- `Curve` は、同じポイントを 2 回通過しない場合は単純ですが、開始ポイントと終了ポイントが同じ場合でも曲線は単純になります。
- 始点と終点が等しい場合、`Curve` は閉じています。
- 閉じた `Curve` の境界は、空になります。
- 閉じていない `Curve` の境界は、その 2 つの端点から構成されます。
- 単純で閉じた `Curve` としては、`LinearRing` が挙げられます。

11.4.2.5 LineString クラス

`LineString` は、点の間を直線で補間した `Curve` です。

`LineString` の例

- 世界地図で、`LineString` オブジェクトは河川を表すことができます。
- 市内地図で、`LineString` オブジェクトは通りを表すことができます。

`LineString` のプロパティー

- `LineString` は、隣り合う 1 対の点で定義される各線分の座標を持ちます。
- ちょうど 2 つの点から構成されている場合、`LineString` は `Line` になります。
- 閉じていて、かつ単純である場合は、`LineString` は `LinearRing` になります。

11.4.2.6 Surface クラス

`Surface` は 2 次元のジオメトリです。これはインスタンス化不可能なクラスです。その唯一のインスタンス化可能なサブクラスは、`Polygon` です。

`Surface` のプロパティー

- `Surface` は 2 次元のジオメトリとして定義されています。
- OpenGIS 仕様では、単純な `Surface` が、1 個の外側の境界と 0 個以上の内側の境界に関連付けられた単一の「パッチ」からなるジオメトリとして定義されています。
- 単純な `Surface` の境界は、その外側と内側の境界に対応する一連の閉じた曲線になります。

11.4.2.7 Polygon クラス

`Polygon` は、多辺のジオメトリを表す平面 `Surface` です。これは 1 個の外側の境界と 0 個以上の内側の境界で定義されますが、それらの内側の各境界によって `Polygon` 内の 1 個の穴が定義されます。

`Polygon` の例

- 地域マップで、`Polygon` オブジェクトは森林や区域などを表すことができます。

`Polygon` の表明

- **Polygon** の境界は、外側と内側の境界を構成する一連の **LinearRing** オブジェクト (つまり、単純かつ閉じた **LineString** オブジェクト) から構成されます。
- **Polygon** のリングは交差しません。 **Polygon** の境界に含まれるリングは、 **Point** で交わりますが、接することしかできません。
- **Polygon** には線分、突起、亀裂は含まれません。
- **Polygon** は、連続した点集合からなる内部を持ちます。
- **Polygon** は穴を持つことができます。穴のある **Polygon** の外部は、連続していません。それぞれの穴が、連続した1つの外部コンポーネントを定義します。

以上の表明により、 **Polygon** は単純なジオメトリになります。

11.4.2.8 GeometryCollection クラス

GeomCollection は、任意のクラスのゼロ個以上のジオメトリの集合であるジオメトリです。

GeomCollection と **GeometryCollection** は同義であり、 **GeomCollection** を優先する型名とします。

ジオメトリコレクション内のすべての要素は、同じ空間参照システム (つまり、同じ座標系) 内にある必要があります。ジオメトリコレクションの要素には他の制約はありませんが、次のセクションで説明する **GeomCollection** のサブクラスによってメンバーシップが制限される場合があります。これらの制限は次の情報に基づくことがあります。

- 要素の型 (たとえば、 **MultiPoint** に格納できるのは **Point** 要素だけです)
- 次元
- 要素間の空間的な重なり具合に関する制約

11.4.2.9 MultiPoint クラス

MultiPoint は、 **Point** 要素から構成されるジオメトリコレクションです。点の接続や順序付けは一切行われません。

MultiPoint の例

- 世界地図で、 **MultiPoint** は一連の小さな島々を表すことができます。
- 市内地図で、 **MultiPoint** はチケットオフィスの系列店を表すことができます。

MultiPoint のプロパティー

- **MultiPoint** は 0 次元のジオメトリです。
- この2つの **Point** の値 (座標値) が等しくない場合は、 **MultiPoint** は単純になります。
- **MultiPoint** の境界は空セットになります。

11.4.2.10 MultiCurve クラス

MultiCurve は、 **Curve** 要素から構成されるジオメトリコレクションです。 **MultiCurve** はインスタンス化不可能なクラスです。

MultiCurve のプロパティー

- **MultiCurve** は 1 次元のジオメトリです。
- **MultiCurve** が単純になるのは、そのすべての要素が単純である場合だけです。2つの要素の唯一の交点は、両方の要素の境界上にある点になります。
- **MultiCurve** 境界は、「「mod 2 共用体ルール」」 (「「奇数偶数ルール」」とも呼ばれる) を適用することによって取得されます: ポイントが奇数の **Curve** 要素の境界内にある場合、そのポイントは **MultiCurve** の境界内にありません。

- すべての要素が閉じている場合、MultiCurve は閉じています。
- 閉じた MultiCurve の境界は、常に空になります。

11.4.2.11 MultiLineString クラス

MultiLineString は、LineString 要素から構成される MultiCurve ジオメトリコレクションです。

MultiLineString の例

- 地域マップで、MultiLineString は河川系や高速道路システムを表すことができます。

11.4.2.12 MultiSurface クラス

MultiSurface は、面要素から構成されるジオメトリコレクションです。MultiSurface はインスタンス化不可能なクラスです。その唯一のインスタンス化可能なサブクラスは、MultiPolygon です。

MultiSurface の表明

- MultiSurface 内のサーフェスには、交差する内部がありません。
- MultiSurface 内のサーフェスには、最大で有限のポイント数と交差する境界があります。

11.4.2.13 MultiPolygon クラス

MultiPolygon は、Polygon 要素から構成される MultiSurface オブジェクトです。

MultiPolygon の例

- 地域マップで、MultiPolygon は湖の系列を表すことができます。

MultiPolygon の表明

- MultiPolygon のどの 2 つの Polygon 要素も、交差する内部を持つことはありません。
- MultiPolygon のどの 2 つの Polygon 要素も、交差したり (交差は 1 つ前の表明でも禁止されています)、無限個の点で接したりしません。
- MultiPolygon にカットライン、突起、亀裂を含めることはできません。MultiPolygon は通常の閉じた点集合です。
- 複数の Polygon を含む MultiPolygon は、連続していない内部を持ちます。MultiPolygon の連続する内部コンポーネントの個数は、MultiPolygon 内の Polygon 値の数と等しくなります。

MultiPolygon のプロパティ

- MultiPolygon は 2 次元のジオメトリです。
- MultiPolygon の境界は、その Polygon 要素の境界に対応する一連の閉じた曲線 (LineString 値) になります。
- MultiPolygon の境界に含まれる各 Curve は、どれか 1 つの Polygon 要素の境界にのみ含まれます。
- Polygon 要素の境界に含まれる Curve は必ず、MultiPolygon の境界にも含まれます。

11.4.3 サポートされる空間データ形式

クエリーでジオメトリオブジェクトを表現するために、次の 2 つの標準空間データ形式が使用されます。

- WKT (Well-Known Text) 形式
- WKB (Well-Known Binary) 形式

MySQL の内部では、WKT、WKB のどちらの形式とも異なる形式でジオメトリ値が格納されます。(内部形式は WKB と似ていますが、SRID を示す最初の 4 バイトがあります。)

異なるデータ形式間の変換に使用できる関数があります。セクション12.17.6「ジオメトリ形式変換関数」を参照してください。

次の各セクションでは、MySQL で使用される空間データフォーマットについて説明します：

- [WKT \(Well-Known Text\) 形式](#)
- [WKB \(Well-Known Binary\) 形式](#)
- [内部ジオメトリ記憶形式](#)

WKT (Well-Known Text) 形式

ジオメトリ値の WKT (Well-Known Text) 表現は、ASCII 形式のジオメトリデータを交換するために設計されています。OpenGIS 仕様書には、WKT 値を書き込むための公式の運用ルールを指定するバックス-ナウア記法が用意されています (セクション11.4「空間データ型」を参照してください)。

ジオメトリオブジェクトの WKT 表現の例：

- [Point](#):

```
POINT(15 20)
```

点の座標は、区切り用のカンマなしに指定されます。これは、座標間にカンマを必要とする SQL [Point\(\)](#) 関数の構文とは異なります。特定の空間演算のコンテキストに適した構文を慎重に使用してください。たとえば、次のステートメントはどちらも [ST_X\(\)](#) を使用して [Point](#) オブジェクトから X 座標を抽出します。最初の場合は、[Point\(\)](#) 関数を直接使用してオブジェクトを生成します。2 番目は、[ST_GeomFromText\(\)](#) を使用して [Point](#) に変換された WKT 表現を使用します。

```
mysql> SELECT ST_X(Point(15, 20));
+-----+
| ST_X(Point(15, 20)) |
+-----+
|          15 |
+-----+

mysql> SELECT ST_X(ST_GeomFromText('POINT(15 20)'));
+-----+
| ST_X(ST_GeomFromText('POINT(15 20)')) |
+-----+
|          15 |
+-----+
```

- 4 つの点を含む [LineString](#):

```
LINestring(0 0, 10 10, 20 25, 50 60)
```

点の座標のペアはカンマで区切られます。

- 外側のリングと内側のリングを 1 つずつ含む [Polygon](#):

```
POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))
```

- 3 つの [Point](#) 値を含む [MultiPoint](#):

```
MULTIPOINT(0 0, 20 20, 60 60)
```

[MultiPoint](#) 値の WKT 形式表現を受け入れる [ST_MPointFromText\(\)](#) や [ST_GeomFromText\(\)](#) などの空間関数では、値内の個々のポイントをカッコで囲むことができます。たとえば、次の関数コールは両方とも有効です：

```
ST_MPointFromText('MULTIPOINT (1 1, 2 2, 3 3)')
ST_MPointFromText('MULTIPOINT ((1 1), (2 2), (3 3))')
```

- 2 つの [LineString](#) 値を含む [MultiLineString](#):

```
MULTILINESTRING((10 10, 20 20), (15 15, 30 15))
```

- 2 つの [Polygon](#) 値を含む [MultiPolygon](#):

```
MULTIPOLYGON(((0 0,10 0,10 10,0 10,0 0)),((5 5,7 5,7 7,5 7, 5 5)))
```

- 2つの [Point](#) 値と 1つの [LineString](#) から構成された [GeometryCollection](#):

```
GEOMETRYCOLLECTION(POINT(10 10), POINT(30 30), LINESTRING(15 15, 20 20))
```

WKB (Well-Known Binary) 形式

ジオメトリ値の WKB (Well-Known Binary) 表現は、ジオメトリ WKB 情報を含む [BLOB](#) 値によって表現されたバイナリストリームとしてジオメトリデータを交換するために使用されます。この形式は、OpenGIS 仕様によって定義されています ([セクション11.4「空間データ型」](#)を参照してください)。これはまた、ISO の SQL/MM Part 3: Spatial 標準でも定義されています。

WKB は、1 バイトの符号なしの整数、4 バイトの符号なしの整数、および 8 バイトの倍精度数 (IEEE 754 形式) を使用します。1 バイトは 8 ビットです。

たとえば、[POINT\(1 -1\)](#) に対応する WKB 値は、21 バイトのシーケンスで構成され、それぞれが 2 桁の 16 進数で表されます:

```
010100000000000000000000F03F000000000000F0BF
```

順序は、次のテーブルに示すコンポーネントで構成されます。

表 11.2 WKB コンポーネントの例

コンポーネント	Size	値
バイト順	1 バイト	01
WKB タイプ	4 バイト	01000000
X 座標	8 バイト	000000000000F03F
Y 座標	8 バイト	000000000000F0BF

各コンポーネントが表す内容は次のとおりです。

- バイト順序インジケータは 1 または 0 で、リトルエンディアンまたはビッグエンディアンの記憶域を示します。リトルエンディアンバイト順序、ビッグエンディアンバイト順序はそれぞれ NDR (Network Data Representation)、XDR (External Data Representation) とも呼ばれます。
- WKB 型はジオメトリ型を示すコードです。MySQL は、1 から 7 までの値を使用して、[Point](#)、[LineString](#)、[Polygon](#)、[MultiPoint](#)、[MultiLineString](#)、[MultiPolygon](#) および [GeometryCollection](#) を示します。
- [Point](#) 値には X 座標と Y 座標が含まれますが、それぞれ倍精度値として表現されます。

さらに複雑なジオメトリ値の WKB 値は、OpenGIS 仕様書に詳しく記されているように、より複雑なデータ構造になります。

内部ジオメトリ記憶形式

MySQL では、SRID の後に WKB 表現の値が続くことを示す 4 バイトを使用してジオメトリ値が格納されます。WKB 形式については、[WKB \(Well-Known Binary\) 形式](#)を参照してください。

WKB 部分では、次の MySQL 固有の考慮事項が適用されます:

- MySQL ではジオメトリがリトルエンディアン値として格納されるため、バイト順序インジケータバイトは 1 です。
- MySQL は、[Point](#)、[LineString](#)、[Polygon](#)、[MultiPoint](#)、[MultiLineString](#)、[MultiPolygon](#) および [GeometryCollection](#) のジオメトリタイプをサポートしています。その他のジオメトリタイプはサポートされていません。
- 空にできるのは [GeometryCollection](#) のみです。このような値は 0 個の要素で格納されます。
- [Polygon](#) リングは時計回りと反時計回りの両方で指定できます。MySQL は、データの読取り時にリングを自動的に反転します。

デカルト座標は空間参照システムの長さ単位で格納され、X 座標に X 値、Y 座標に Y 値が格納されます。軸方向は、空間参照システムによって指定された方向です。

地理座標は空間参照システムの角度単位で格納され、経度は X 座標に、緯度は Y 座標に格納されます。軸方向と子午線は、空間参照システムによって指定された方向です。

`LENGTH()` 関数は、値の格納に必要な領域をバイト単位で返します。例:

```
mysql> SET @g = ST_GeomFromText("POINT(1 -1)");
mysql> SELECT LENGTH(@g);
+-----+
| LENGTH(@g) |
+-----+
|      25 |
+-----+
mysql> SELECT HEX(@g);
+-----+
| HEX(@g) |
+-----+
| 000000000101000000000000000000F03F000000000000F0BF |
+-----+
```

値の長さは 25 バイトで、次のコンポーネントで構成されます (16 進数値から確認できます):

- SRID を示す整数 4 バイト (0)
- バイトオーダーを示す整数 1 バイト (1 = リトルエンディアン)
- 型情報を示す整数 4 バイト (1 = `Point`)
- X 座標を示す倍精度 8 バイト (1)
- Y 座標を示す倍精度 8 バイト (-1)

11.4.4 ジオメトリの整形形式と妥当性

ジオメトリ値の場合、MySQL では、構文的に整形形式の概念とジオメトリ学的に有効な概念が区別されます。

ジオメトリは、次の (完全でない) リストのような条件を満たす場合、構文的に整形形式になります:

- `Linestring` には少なくとも 2 つの点があります
- `Polygon` に少なくとも 1 つのリングがあります
- `Polygon` リングが閉じています (最初と最後のポイントが同じです)
- `Polygon` リングには少なくとも 4 つの点があります (最小 `polygon` は最初と最後の点と同じ三角形です)
- コレクションが空でない (`GeometryCollection` を除く)

ジオメトリは、構文的に整形形式で、次の (完全でない) リストのような条件を満たしている場合、ジオメトリ学的に有効です:

- `Polygon` が自己交差していません
- `Polygon` 内部リングは外部リングの内側にあります
- `Multipolygons` に重なり合う `Polygon` がありません

ジオメトリが構文的に整形形式でない場合、空間関数は失敗します。WKT 値または WKB 値を解析する空間インポート関数では、構文的に整形形式でないジオメトリを作成しようとするとエラーが発生します。また、ジオメトリをテーブルに格納しようとする構文の整形形式もチェックされます。

ジオメトリ学的に無効なジオメトリの挿入、選択および更新は許可されますが、構文的に整形形式である必要があります。計算費用のため、MySQL はジオメトリ学的な有効性を明示的にチェックしません。空間計算では、無効なジオメトリが検出されてエラーが発生する場合がありますが、無効性を検出せずに未定義の結果が返されることもあ

ります。地理的に有効なジオメトリを必要とするアプリケーションでは、`ST_IsValid()` 関数を使用してジオメトリをチェックする必要があります。

11.4.5 空間参照システムのサポート

空間データの空間参照システム (SRS) は、地理的位置の調整ベースのシステムです。

空間参照システムには様々なタイプがあります:

- 投影 SRS は、地球を平面に投影したものです; すなわち、平面の地図です。たとえば、地球の周囲の紙の円柱を照らす地球の内側の電球は、地図を紙に投影します。結果は地理参照されます: 各ポイントは地球上の場所にマッピングされます。その平面上の座標系は、経度と緯度ではなく長さの単位 (メートル、フィートなど) を使用したデカルト座標です。

この場合の地球は楕円体、すなわち押しつぶされた球体です。地球は、東西の軸よりも北西の軸が少し短いため、正確にはわずかに押しつぶされた球体ですが、完全な球体を使用すると計算速度は速くなります。

- 地理 SRS は、楕円体上の経度 - 緯度 (または緯度 - 経度) 座標を任意の角度単位で表す非投影 SRS です。
- SRID 0 によって MySQL で示される SRS は、軸に単位が割り当てられていない無限の平らなデカルト平面を表します。投影 SRS とは異なり、地理参照されず、必ずしも地球を表すわけではありません。これは、任意の用途に使用できる抽象平面です。SRID 0 は、MySQL の空間データのデフォルト SRID です。

MySQL では、空間データに使用可能な空間参照システムに関する情報がデータディクショナリ `mysql.st_spatial_reference_systems` テーブルに保持されます。このテーブルには、投影 SRID および地理 SRS のエントリを格納できます。このデータディクショナリテーブルは非表示ですが、SRS エントリの内容は、`mysql.st_spatial_reference_systems` 上のビューとして実装された `INFORMATION_SCHEMA.ST_SPATIAL_REFERENCE_SYSTEMS` テーブルを介して使用できます (セクション 26.36 「`INFORMATION_SCHEMA.ST_SPATIAL_REFERENCE_SYSTEMS` テーブル」を参照)。

SRS エントリの例を次に示します:

```
mysql> SELECT *
FROM INFORMATION_SCHEMA.ST_SPATIAL_REFERENCE_SYSTEMS
WHERE SRS_ID = 4326\G
***** 1. row *****
SRS_NAME: WGS 84
SRS_ID: 4326
ORGANIZATION: EPSG
ORGANIZATION_COORDSYS_ID: 4326
DEFINITION: GEOGCS["WGS 84",DATUM["World Geodetic System 1984",
SPHEROID["WGS 84",6378137,298.257223563,
AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],
PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],
UNIT["degree",0.017453292519943278,
AUTHORITY["EPSG","9122"]],
AXIS["Lat",NORTH],AXIS["Long",EAST],
AUTHORITY["EPSG","4326"]]
DESCRIPTION:
```

このエントリでは、GPS システムに使用される SRS について説明します。名前 (`SRS_NAME`) は WGS 84 で、ID (`SRS_ID`) は 4326 で、これは「欧州石油調査グループ」(EPSG) で使用される ID です。

`DEFINITION` カラムの SRS 定義は WKT 値であり、Open Geospatial Consortium 文書 `OGC 12-063r5` で指定されています。

`SRS_ID` 値は、ジオメトリ値の SRID と同じ種類の値を表すか、SRID 引数として空間関数に渡されます。SRID 0 (単位なしデカルト平面) は特殊です。常に有効な空間参照システム ID であり、SRID 値に依存する空間データの計算に使用できます。

複数のジオメトリ値の計算では、すべての値が同じ SRID を持つ必要があります、そうでない場合はエラーが発生します。

SRS 定義解析は、GIS 関数で定義が必要な場合にオンデマンドで実行されます。解析された定義は、再利用を可能にし、SRS 情報を必要とするすべてのステートメントの解析オーバーヘッドが発生しないように、データディクショナリキャッシュに格納されます。

データディクショナリに格納されている SRS エントリの操作を可能にするために、MySQL には次の SQL ステートメントが用意されています:

- [CREATE SPATIAL REFERENCE SYSTEM: セクション13.1.19「CREATE SPATIAL REFERENCE SYSTEM ステートメント」](#) を参照してください。このステートメントの説明には、SRS 構成部品に関する追加情報が含まれます。
- [DROP SPATIAL REFERENCE SYSTEM: セクション13.1.31「DROP SPATIAL REFERENCE SYSTEM ステートメント」](#) を参照してください。

11.4.6 空間カラムの作成

MySQL には、[CREATE TABLE](#) や [ALTER TABLE](#) を使用する方法など、ジオメトリ型の空間カラムを作成するための標準的な方法が用意されています。空間カラムは、[MyISAM](#)、[InnoDB](#)、[NDB](#)、および [ARCHIVE](#) テーブルでサポートされています。[セクション11.4.10「空間インデックスの作成」](#)の空間インデックスに関するノートも参照してください。

空間データ型のカラムに SRID 属性を指定して、カラムに格納されている値の空間参照システム (SRS) を明示的に指定できます。SRID 制限カラムの意味については、[セクション11.4.1「空間データ型」](#)を参照してください。

- 空間カラムを含むテーブルを作成するには、[CREATE TABLE](#) ステートメントを使用します。

```
CREATE TABLE geom (g GEOMETRY);
```

- 既存のテーブルに対して空間カラムの追加や削除を行うには、[ALTER TABLE](#) ステートメントを使用します。

```
ALTER TABLE geom ADD pt POINT;  
ALTER TABLE geom DROP pt;
```

11.4.7 空間カラムへのデータ移入

空間カラムを作成し終わったら、空間データを移入できます。

値は内部ジオメトリ形式で格納する必要がありますが、WKT (Well-Known Text)、WKB (Well-Known Binary) のいずれの形式からでも、その形式に値を変換できます。次の例は、WKT 値を内部ジオメトリ形式に変換することによって、ジオメトリ値をテーブルに挿入する方法を示しています。

- 次のように [INSERT](#) ステートメント内で直接変換を実行します。

```
INSERT INTO geom VALUES (ST_GeomFromText('POINT(1 1)');  
  
SET @g = 'POINT(1 1);  
INSERT INTO geom VALUES (ST_GeomFromText(@g));
```

- 次のように [INSERT](#) の前に変換を実行します。

```
SET @g = ST_GeomFromText('POINT(1 1)');  
INSERT INTO geom VALUES (@g);
```

次の例では、より複雑なジオメトリをテーブルに挿入しています。

```
SET @g = 'LINESTRING(0 0,1 1,2 2);  
INSERT INTO geom VALUES (ST_GeomFromText(@g));  
  
SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7,5 5));  
INSERT INTO geom VALUES (ST_GeomFromText(@g));  
  
SET @g =  
'GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 4));  
INSERT INTO geom VALUES (ST_GeomFromText(@g));
```

前述の例では、[ST_GeomFromText\(\)](#) を使用してジオメトリ値を作成しています。次のように型に固有の関数を使用することもできます。

```
SET @g = 'POINT(1 1);  
INSERT INTO geom VALUES (ST_PointFromText(@g));
```

```
SET @g = 'LINESTRING(0 0,1 1,2 2);
INSERT INTO geom VALUES (ST_LineStringFromText(@g));

SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5));
INSERT INTO geom VALUES (ST_PolygonFromText(@g));

SET @g =
'GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4));
INSERT INTO geom VALUES (ST_GeomCollFromText(@g));
```

ジオメトリ値の WKB 表現を使用するクライアントアプリケーションプログラムが、クエリーで正しく作成された WKB のサーバーへの送信を担います。この要件を満たす方法は複数あります。例:

- 次のように、16 進リテラル構文を使用して、`POINT(1 1)` 値を挿入します。

```
INSERT INTO geom VALUES
(ST_GeomFromWKB(X'010100000000000000000000F03F000000000000F03F'));
```

- ODBC アプリケーションは、`BLOB` 型の引数を使用して WKB 表現をプレースホルダにバインドし、WKB 表現を送信できます。

```
INSERT INTO geom VALUES (ST_GeomFromWKB(?))
```

ほかのプログラミングインタフェースも似たようなプレースホルダメカニズムをサポートしている可能性があります。

- C プログラムでは、`mysql_real_escape_string_quote()` を使用してバイナリ値をエスケープし、サーバーに送信されるクエリー文字列に結果を含めることができます。`mysql_real_escape_string_quote()` を参照してください。

11.4.8 空間データのフェッチ

テーブルに格納されたジオメトリ値は内部形式でフェッチできます。WKT 形式から WKB 形式に変換することもできます。

- 内部形式での空間データのフェッチ:

内部形式でジオメトリ値をフェッチする方法は、テーブル間でデータの転送を行う場合に便利です。

```
CREATE TABLE geom2 (g GEOMETRY) SELECT g FROM geom;
```

- WKT 形式での空間データのフェッチ:

`ST_AsText()` 関数は、ジオメトリを内部形式から WKT 文字列に変換します。

```
SELECT ST_AsText(g) FROM geom;
```

- WKB 形式での空間データのフェッチ:

`ST_AsBinary()` 関数は、ジオメトリを内部形式から WKB 値を含む `BLOB` に変換します。

```
SELECT ST_AsBinary(g) FROM geom;
```

11.4.9 空間分析の最適化

`MyISAM` および `InnoDB` テーブルの場合、空間データを含むカラムでの検索操作は、`SPATIAL` インデックスを使用して最適化できます。もっとも典型的な操作は次のとおりです。

- 指定された点を含むすべてのオブジェクトを検索する点クエリー
- 所定の領域と重なるすべてのオブジェクトを検索する領域クエリー

MySQL では、2 次分割 R ツリーを使用して空間カラムの `SPATIAL` インデックスが実装されています。`SPATIAL` インデックスは、ジオメトリの最小境界矩形 (MBR) を使用して構築されます。大部分のジオメトリでは、MBR はそのジオメトリを囲む最小矩形となります。水平または垂直方向のライン文字列では、MBR は矩形からライン文字列に縮退します。点の場合、MBR は矩形から点に縮退します。

空間カラムに通常のインデックスを作成することも可能です。非 **SPATIAL** インデックスでは、**POINT** カラムを除くすべての空間カラムでプリフィクスを宣言する必要があります。

MyISAM および **InnoDB** では、**SPATIAL** インデックスと **SPATIAL** 以外のインデックスの両方がサポートされます。その他のストレージエンジンは[セクション13.1.15「CREATE INDEX ステートメント」](#)で説明しているように、非 **SPATIAL** インデックスをサポートします。

11.4.10 空間インデックスの作成

InnoDB および **MyISAM** テーブルの場合、MySQL では、通常のインデックスを作成する場合と同様の構文を使用して空間インデックスを作成できますが、**SPATIAL** キーワードを使用します。空間インデックスのカラムは、**NOT NULL** と宣言する必要があります。次の各例では空間インデックスの作成方法を示します。

- **CREATE TABLE** を使用する場合:

```
CREATE TABLE geom (g GEOMETRY NOT NULL SRID 4326, SPATIAL INDEX(g));
```

- **ALTER TABLE** を使用する場合:

```
CREATE TABLE geom (g GEOMETRY NOT NULL SRID 4326);
ALTER TABLE geom ADD SPATIAL INDEX(g);
```

- **CREATE INDEX** を使用する場合:

```
CREATE TABLE geom (g GEOMETRY NOT NULL SRID 4326);
CREATE SPATIAL INDEX g ON geom (g);
```

SPATIAL INDEX は R ツリーインデックスを作成します。空間カラムの非空間インデックスをサポートするストレージエンジンでは、B ツリーインデックスが作成されます。空間値に対する B ツリーインデックスは、正確な値の検索に役立ちますが、範囲スキャンには役立ちません。

オプティマイザは、SRID 制限のあるカラムに定義された空間インデックスを使用できます。詳細は、[セクション11.4.1「空間データ型」](#)および[セクション8.3.3「SPATIAL インデックス最適化」](#)を参照してください。

空間カラムのインデックス作成の詳細については、[セクション13.1.15「CREATE INDEX ステートメント」](#)を参照してください。

空間インデックスを削除するには、次のように **ALTER TABLE** または **DROP INDEX** を使用します。

- **ALTER TABLE** を使用する場合:

```
ALTER TABLE geom DROP INDEX g;
```

- **DROP INDEX** を使用する場合:

```
DROP INDEX g ON geom;
```

例: テーブル **geom** に 32,000 件を超えるジオメトリが含まれていて、それらの図形が型 **GEOMETRY** のカラム **g** に格納されているものとします。またこのテーブルには、オブジェクト ID の値を格納するための **AUTO_INCREMENT** カラム **fid** も含まれています。

```
mysql> DESCRIBE geom;
+-----+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| fid  | int(11) | | PRI | NULL    | auto_increment |
| g    | geometry | | | | |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> SELECT COUNT(*) FROM geom;
+-----+
| count(*) |
+-----+
| 32376 |
+-----+
1 row in set (0.00 sec)
```

コラム `g` に空間インデックスを追加するには、次のステートメントを使用します。

```
mysql> ALTER TABLE geom ADD SPATIAL INDEX(g);
Query OK, 32376 rows affected (4.05 sec)
Records: 32376 Duplicates: 0 Warnings: 0
```

11.4.11 空間インデックスの使用

オプティマイザは、`WHERE` 句で `MBRContains()` や `MBRWithin()` などの関数が使用されているクエリーの検索に、使用可能な空間インデックスを含めることができるかどうかを調べます。次のクエリーは、所定の矩形に含まれるすべてのオブジェクトを検索します。

```
mysql> SET @poly =
-> 'Polygon((30000 15000,
          31000 15000,
          31000 16000,
          30000 16000,
          30000 15000));'
mysql> SELECT fid,ST_AsText(g) FROM geom WHERE
-> MBRContains(ST_GeomFromText(@poly),g);
+-----+
| fid | ST_AsText(g) |
+-----+
| 21 | LINESTRING(30350.4 15828.8,30350.6 15845,30333.8 15845,30 ... |
| 22 | LINESTRING(30350.6 15871.4,30350.6 15887.8,30334 15887.8, ... |
| 23 | LINESTRING(30350.6 15914.2,30350.6 15930.4,30334 15930.4, ... |
| 24 | LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4, ... |
| 25 | LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882. ... |
| 26 | LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4, ... |
| 249 | LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946. ... |
| 1 | LINESTRING(30250.4 15129.2,30248.8 15138.4,30238.2 15136. ... |
| 2 | LINESTRING(30220.2 15122.8,30217.2 15137.8,30207.6 15136, ... |
| 3 | LINESTRING(30179 15114.4,30176.6 15129.4,30167 15128,3016 ... |
| 4 | LINESTRING(30155.2 15121.4,30140.4 15118.6,30142 15109,30 ... |
| 5 | LINESTRING(30192.4 15085,30177.6 15082.2,30179.2 15072.4, ... |
| 6 | LINESTRING(30244 15087,30229 15086.2,30229.4 15076.4,3024 ... |
| 7 | LINESTRING(30200.6 15059.4,30185.6 15058.6,30186 15048.8, ... |
| 10 | LINESTRING(30179.6 15017.8,30181 15002.8,30190.8 15003.6, ... |
| 11 | LINESTRING(30154.2 15000.4,30168.6 15004.8,30166 15014.2, ... |
| 13 | LINESTRING(30105 15065.8,30108.4 15050.8,30118 15053,3011 ... |
| 154 | LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30 ... |
| 155 | LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30 ... |
| 157 | LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4, ... |
+-----+
20 rows in set (0.00 sec)
```

このクエリーがどのように実行されているのかを、`EXPLAIN` を使用して確認します。

```
mysql> SET @poly =
-> 'Polygon((30000 15000,
          31000 15000,
          31000 16000,
          30000 16000,
          30000 15000));'
mysql> EXPLAIN SELECT fid,ST_AsText(g) FROM geom WHERE
-> MBRContains(ST_GeomFromText(@poly),g)\G
***** 1. row *****
id: 1
select_type: SIMPLE
table: geom
type: range
possible_keys: g
key: g
key_len: 32
ref: NULL
rows: 50
Extra: Using where
1 row in set (0.00 sec)
```

空間インデックスがないとどうなるのかを確認します。

```
mysql> SET @poly =
-> 'Polygon((30000 15000,
```

```

31000 15000,
31000 16000,
30000 16000,
30000 15000));
mysql> EXPLAIN SELECT fid,ST_AsText(g) FROM g IGNORE INDEX (g) WHERE
-> MBRContains(ST_GeomFromText(@poly),g)\G
***** 1. row *****
id: 1
select_type: SIMPLE
table: geom
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 32376
Extra: Using where
1 row in set (0.00 sec)

```

空間インデックスを使用せずに `SELECT` ステートメントを実行しても結果は同じになりますが、実行時間は 0.00 秒から 0.46 秒に増大します。

```

mysql> SET @poly =
-> 'Polygon((30000 15000,
31000 15000,
31000 16000,
30000 16000,
30000 15000));
mysql> SELECT fid,ST_AsText(g) FROM geom IGNORE INDEX (g) WHERE
-> MBRContains(ST_GeomFromText(@poly),g);
+-----+-----+
| fid | ST_AsText(g) |
+-----+-----+
| 1 | LINESTRING(30250.4 15129.2,30248.8 15138.4,30238.2 15136. ... |
| 2 | LINESTRING(30220.2 15122.8,30217.2 15137.8,30207.6 15136, ... |
| 3 | LINESTRING(30179 15114.4,30176.6 15129.4,30167 15128,3016 ... |
| 4 | LINESTRING(30155.2 15121.4,30140.4 15118.6,30142 15109,30 ... |
| 5 | LINESTRING(30192.4 15085,30177.6 15082.2,30179.2 15072.4, ... |
| 6 | LINESTRING(30244 15087,30229 15086.2,30229.4 15076.4,3024 ... |
| 7 | LINESTRING(30200.6 15059.4,30185.6 15058.6,30186 15048.8, ... |
| 10 | LINESTRING(30179.6 15017.8,30181 15002.8,30190.8 15003.6, ... |
| 11 | LINESTRING(30154.2 15000.4,30168.6 15004.8,30166 15014.2, ... |
| 13 | LINESTRING(30105 15065.8,30108.4 15050.8,30118 15053,3011 ... |
| 21 | LINESTRING(30350.4 15828.8,30350.6 15845,30333.8 15845,30 ... |
| 22 | LINESTRING(30350.6 15871.4,30350.6 15887.8,30334 15887.8, ... |
| 23 | LINESTRING(30350.6 15914.2,30350.6 15930.4,30334 15930.4, ... |
| 24 | LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4, ... |
| 25 | LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882. ... |
| 26 | LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4, ... |
| 154 | LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30 ... |
| 155 | LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30 ... |
| 157 | LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4, ... |
| 249 | LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946. ... |
+-----+-----+
20 rows in set (0.46 sec)

```

11.5 JSON データ型

- [JSON 値の作成](#)
- [JSON 値の正規化、マージおよび自動ラップ](#)
- [JSON 値の検索および変更](#)
- [JSON パス構文](#)
- [JSON 値の比較および順序付け](#)
- [JSON 値と非 JSON 値の間の変換](#)
- [JSON 値の集計](#)

MySQL は、JSON (JavaScript Object Notation) ドキュメント内のデータへの効率的なアクセスを可能にする、[RFC 7159](#) によって定義されたネイティブ **JSON** データ型をサポートしています。JSON データ型には、JSON 形式の文字列を文字列カラムに格納するよりも、次の利点があります：

- **JSON** カラムに格納されている JSON ドキュメントの自動検証。無効なドキュメントではエラーが発生します。
- 最適化された記憶域形式。JSON カラムに格納された JSON ドキュメントは、ドキュメント要素へのクイック読み取りアクセスを許可する内部形式に変換されます。サーバーが後でこのバイナリ形式で格納された JSON 値を読み取る必要がある場合、テキスト表現から値を解析する必要はありません。バイナリ形式は、サーバーがドキュメント内のサブオブジェクトまたはネストされた値の前後のすべての値を読み取ることなく、キーまたは配列インデックスによって直接サブオブジェクトまたはネストされた値を検索できるように構造化されています。

MySQL 8.0 では、[JSON_MERGE_PATCH\(\)](#) 関数を使用して [RFC 7396](#) で定義された JSON マージパッチ形式もサポートされます。例および詳細は、この関数の説明および [JSON 値の正規化、マージおよび自動ラップ](#) を参照してください。

注記

この説明では、monotype で **JSON** を使用して特に JSON データ型を示し、JSON データ全般を示す通常のフォントで「JSON」を使用します。

JSON ドキュメントの格納に必要な領域は、[LONGBLOB](#) または [LONGTEXT](#) の場合とほぼ同じです。詳細は、[セクション 11.7 「データ型のストレージ要件」](#) を参照してください。JSON カラムに格納される JSON ドキュメントのサイズは、[max_allowed_packet](#) システム変数の値に制限されることに注意してください。(サーバーが JSON 値をメモリ内で内部的に操作している場合、これより大きくなる可能性があります。制限は、サーバーが JSON 値を格納するときに適用されます。) JSON ドキュメントの格納に必要な領域の量は、[JSON_STORAGE_SIZE\(\)](#) 関数を使用して取得できます。JSON カラムの場合、記憶域サイズ、つまり、部分更新が実行される前にこの関数によって使用された値 (このセクションで後述する JSON 部分更新最適化の説明を参照)。

MySQL 8.0.13 より前は、JSON カラムに **NULL** 以外のデフォルト値を含めることはできません。

JSON データ型とともに、作成、操作、検索などの JSON 値に対する操作を可能にする一連の SQL 関数を使用できます。次に、これらの操作の例を示します。個々の関数の詳細は、[セクション 12.18 「JSON 関数」](#) を参照してください。

GeoJSON 値を操作するための一連の空間関数も使用できます。[セクション 12.17.11 「空間 GeoJSON 関数」](#) を参照してください。

他のバイナリ型のカラムと同様に、JSON カラムは直接インデックス付けされません。かわりに、JSON カラムからスカラー値を抽出するインデックスを生成されたカラムに作成できます。詳細な例は、[JSON カラムインデックスを提供するための生成されたカラムのインデックス付け](#) を参照してください。

MySQL オプティマイザは、JSON 式に一致する仮想カラムの互換性のあるインデックスも検索します。

MySQL 8.0.17 以降では、[InnoDB](#) ストレージエンジンは JSON 配列で複数値インデックスをサポートします。[複数値インデックス](#) を参照してください。

MySQL NDB Cluster 8.0 は、JSON カラムおよび MySQL JSON 関数をサポートしています。これには、JSON カラムにインデックスを作成できないための回避方法として、JSON カラムから生成されたカラムに対するインデックスの作成が含まれます。[NDB](#) テーブルごとに最大 3 つの JSON カラムがサポートされます。

JSON 値の部分更新

MySQL 8.0 では、オプティマイザは、古いドキュメントを削除して新しいドキュメント全体をカラムに書き込むかわりに、JSON カラムの部分的なインプレース更新を実行できます。この最適化は、次の条件を満たす更新に対して実行できます：

- 更新するカラムが **JSON** として宣言されました。
- **UPDATE** ステートメントでは、[JSON_SET\(\)](#)、[JSON_REPLACE\(\)](#) または [JSON_REMOVE\(\)](#) のいずれかの関数を使用してカラムを更新します。カラム値の直接割当て (**UPDATE mytable SET jcol = '{"a": 10, "b": 25}'** など) は、部分更新として実行できません。

単一の `UPDATE` ステートメントでの複数の `JSON` カラムの更新は、この方法で最適化できます。MySQL では、前述の 3 つの関数を使用して値が更新されるカラムのみの部分更新を実行できます。

- 入力カラムとターゲットカラムは同じカラムである必要があります。 `UPDATE mytable SET jcol1 = JSON_SET(jcol2, '$.a', 100)` などのステートメントは部分更新として実行できません。

更新では、入力カラムとターゲットカラムが同じであるかぎり、前の項目にリストされている関数へのネストされたコールを任意の組合せで使用できます。

- すべての変更により、既存の配列またはオブジェクト値が新しい配列またはオブジェクト値に置き換えられ、新しい要素は親オブジェクトまたは配列に追加されません。
- 置換する値は、少なくとも置換値と同じ大きさである必要があります。つまり、新しい値を古い値より大きくすることはできません。

この要件で発生する可能性がある例外は、以前の部分更新で大きい値のための十分な領域が残っている場合に発生します。 `JSON_STORAGE_FREE()` 関数を使用すると、`JSON` カラムの部分更新によって解放された領域の量を確認できます。

このような部分更新は、領域を節約するコンパクトな形式を使用してバイナリログに書き込むことができます。これは、`binlog_row_value_options` システム変数を `PARTIAL_JSON` に設定することで有効にできます。詳細は、この変数の説明を参照してください。

次のいくつかのセクションでは、JSON 値の作成および操作に関する基本情報を示します。

JSON 値の作成

JSON 配列には、カンマで区切られ、`[`および`]`文字で囲まれた値のリストが含まれます:

```
["abc", 10, null, true, false]
```

JSON オブジェクトには、カンマで区切られ、`{`および`}`文字で囲まれたキーと値のペアのセットが含まれます:

```
{"k1": "value", "k2": 10}
```

例に示すように、JSON 配列およびオブジェクトには、文字列または数値、JSON `null` リテラルまたは JSON ブール `true` または `false` リテラルであるスカラー値を含めることができます。JSON オブジェクトのキーは文字列である必要があります。時間的 (日付、時間または日時) スカラー値も使用できます:

```
["12:18:29.000000", "2015-07-29", "2015-07-29 12:18:29.000000"]
```

ネストは、JSON 配列要素および JSON オブジェクトキー値内で許可されます:

```
[99, {"id": "HK500", "cost": 75.99}, [{"hot", "cold"}],  
{"k1": "value", "k2": [10, 20]}
```

この目的 ([セクション12.18.2「JSON 値を作成する関数」](#) を参照) のために MySQL によって提供される多数の関数から JSON 値を取得したり、`CAST(value AS JSON)` ([JSON 値と非 JSON 値の間の変換](#) を参照) を使用して他のタイプの値を `JSON` タイプにキャストすることもできます。次のいくつかの段落では、MySQL が入力として提供される JSON 値を処理する方法について説明します。

MySQL では、JSON 値は文字列として書き込まれます。MySQL は、JSON 値を必要とするコンテキストで使用される文字列を解析し、JSON として有効でない場合はエラーを生成します。次の例に示すように、これらのコンテキストには、`JSON` データ型を持つカラムへの値の挿入、および JSON 値を想定する関数への引数の受渡し (通常は MySQL JSON 関数のドキュメントで `json_doc` または `json_val` として示されています) が含まれます:

- 値を `JSON` カラムに挿入しようとすると、その値が有効な JSON 値である場合は成功しますが、そうでない場合は失敗します:

```
mysql> CREATE TABLE t1 (jdoc JSON);  
Query OK, 0 rows affected (0.20 sec)  
  
mysql> INSERT INTO t1 VALUES({'key1': 'value1', 'key2': 'value2'});  
Query OK, 1 row affected (0.01 sec)
```

```
mysql> INSERT INTO t1 VALUES('[1, 2,'];
ERROR 3140 (22032) at line 2: Invalid JSON text:
"Invalid value." at position 6 in value (or column) '[1, 2,']
```

このようなエラーメッセージ内の「位置 N」の位置は 0 ベースですが、値の問題が実際に発生する場所を大まかに示すものとみなす必要があります。

- `JSON_TYPE()` 関数は、JSON 引数を想定し、JSON 値に解析しようとします。値 JSON 型が有効な場合はそれを返し、それ以外の場合はエラーを生成します:

```
mysql> SELECT JSON_TYPE('[a', 'b', 1]);
+-----+
| JSON_TYPE('[a', 'b', 1]) |
+-----+
| ARRAY                |
+-----+

mysql> SELECT JSON_TYPE('hello');
+-----+
| JSON_TYPE('hello') |
+-----+
| STRING              |
+-----+

mysql> SELECT JSON_TYPE('hello');
ERROR 3146 (22032): Invalid data type for JSON data in argument 1
to function json_type; a JSON string or JSON type is required.
```

MySQL は、`utf8mb4` 文字セットおよび `utf8mb4_bin` 照合順序を使用して、JSON コンテキストで使用される文字列を処理します。他の文字セットの文字列は、必要に応じて `utf8mb4` に変換されます。(ASCII および `utf8` は `utf8mb4` のサブセットであるため、`ascii` または `utf8` 文字セットの文字列の場合、変換は必要ありません。)

リテラル文字列を使用して JSON 値を記述するかわりに、コンポーネント要素から JSON 値を構成するための関数が存在します。`JSON_ARRAY()` は、(空の可能性のある) 値リストを取得し、これらの値を含む JSON 配列を返します:

```
mysql> SELECT JSON_ARRAY('a', 1, NOW());
+-----+
| JSON_ARRAY('a', 1, NOW()) |
+-----+
| ["a", 1, "2015-07-27 09:43:47.000000"] |
+-----+
```

`JSON_OBJECT()` は、キーと値のペアの (空の可能性のある) リストを取得し、それらのペアを含む JSON オブジェクトを返します:

```
mysql> SELECT JSON_OBJECT('key1', 1, 'key2', 'abc');
+-----+
| JSON_OBJECT('key1', 1, 'key2', 'abc') |
+-----+
| {"key1": 1, "key2": "abc"} |
+-----+
```

`JSON_MERGE_PRESERVE()` は、複数の JSON ドキュメントを取得し、結合された結果を返します:

```
mysql> SELECT JSON_MERGE_PRESERVE('[a', 1], '{"key": "value"}');
+-----+
| JSON_MERGE_PRESERVE('[a', 1], '{"key": "value"}') |
+-----+
| ["a", 1, {"key": "value"}] |
+-----+
1 row in set (0.00 sec)
```

マージルールの詳細は、[JSON 値の正規化、マージおよび自動ラップ](#) を参照してください。

(MySQL 8.0.3 以降では、動作が多少異なる `JSON_MERGE_PATCH()` もサポートされます。これらの関数の違いの詳細は、[JSON_MERGE_PATCH\(\) と JSON_MERGE_PRESERVE\(\) の比較](#) を参照してください。)

JSON 値はユーザー定義変数に割り当てることができます:

```
mysql> SET @j = JSON_OBJECT('key', 'value');
```

```
mysql> SELECT @j;
+-----+
| @j      |
+-----+
| {"key": "value"} |
+-----+
```

ただし、ユーザ一定義変数は JSON データ型にできないため、前述の例の @j は JSON 値のように見え、JSON 値と同じ文字セットおよび照合順序を持ちますが、JSON データ型を持ちません。かわりに、JSON_OBJECT() からの結果は、変数に割り当てられるときに文字列に変換されます。

JSON 値の変換によって生成される文字列には、utf8mb4 の文字セットと utf8mb4_bin の照合順序があります:

```
mysql> SELECT CHARSET(@j), COLLATION(@j);
+-----+-----+
| CHARSET(@j) | COLLATION(@j) |
+-----+-----+
| utf8mb4     | utf8mb4_bin   |
+-----+-----+
```

utf8mb4_bin はバイナリ照合であるため、JSON 値の比較では大/小文字が区別されます。

```
mysql> SELECT JSON_ARRAY('x') = JSON_ARRAY('X');
+-----+
| JSON_ARRAY('x') = JSON_ARRAY('X') |
+-----+
| 0 |
+-----+
```

大/小文字の区別は、JSON null、true および false リテラルにも適用され、常に小文字で記述する必要があります:

```
mysql> SELECT JSON_VALID('null'), JSON_VALID('Null'), JSON_VALID('NULL');
+-----+-----+-----+
| JSON_VALID('null') | JSON_VALID('Null') | JSON_VALID('NULL') |
+-----+-----+-----+
| 1 | 0 | 0 |
+-----+-----+-----+

mysql> SELECT CAST('null' AS JSON);
+-----+
| CAST('null' AS JSON) |
+-----+
| null |
+-----+
1 row in set (0.00 sec)

mysql> SELECT CAST('NULL' AS JSON);
ERROR 3141 (22032): Invalid JSON text in argument 1 to function cast_as_json:
"Invalid value." at position 0 in 'NULL'.
```

JSON リテラルの大/小文字の区別は、任意の文字で記述できる SQL NULL、TRUE および FALSE リテラルの区別とは異なります:

```
mysql> SELECT ISNULL(null), ISNULL(Null), ISNULL(NULL);
+-----+-----+-----+
| ISNULL(null) | ISNULL(Null) | ISNULL(NULL) |
+-----+-----+-----+
| 1 | 1 | 1 |
+-----+-----+-----+
```

JSON ドキュメントに引用符文字 ("または') を挿入する必要がある場合や望ましい場合があります。この例では、次に示す SQL ステートメントを使用して作成されたテーブルに、それぞれ適切なキーワードとペアになっている MySQL に関するファクトを示すステートメントを表す文字列を含む JSON オブジェクトを挿入するとします:

```
mysql> CREATE TABLE facts (sentence JSON);
```

キーワードと文のペアは次のとおりです:

```
mascot: The MySQL mascot is a dolphin named "Sakila".
```

これを JSON オブジェクトとして `facts` テーブルに挿入する方法の 1 つは、MySQL `JSON_OBJECT()` 関数を使用することです。この場合、次に示すように、バックスラッシュを使用して各引用文字をエスケープする必要があります:

```
mysql> INSERT INTO facts VALUES
> (JSON_OBJECT("mascot", "Our mascot is a dolphin named \"Sakila\".));
```

JSON オブジェクトリテラルとして値を挿入する場合、これは同じように機能しません。この場合、次のように二重のバックスラッシュエスケープシーケンスを使用する必要があります:

```
mysql> INSERT INTO facts VALUES
> ({"mascot": "Our mascot is a dolphin named \\\"Sakila\\\"."});
```

二重バックスラッシュを使用すると、MySQL はエスケープシーケンス処理を実行せず、代わりに文字列リテラルを処理のためにストレージエンジンに渡します。前述のいずれかの方法で JSON オブジェクトを挿入した後、次のように単純な `SELECT` を実行することで、JSON カラム値にバックスラッシュが存在することを確認できます:

```
mysql> SELECT sentence FROM facts;
+-----+
| sentence |
+-----+
| {"mascot": "Our mascot is a dolphin named \"Sakila\"."} |
+-----+
```

`mascot` をキーとして使用するこの特定の文を検索するには、次に示すように、カラムパス演算子 `->` を使用できます:

```
mysql> SELECT col->"$.mascot" FROM qtest;
+-----+
| col->"$.mascot" |
+-----+
| "Our mascot is a dolphin named \"Sakila\"." |
+-----+
1 row in set (0.00 sec)
```

これにより、バックスラッシュは引用符とともにそのまま残ります。`mascot` をキーとして使用し、引用符やエスケープを含めずに目的の値を表示するには、次のようにインラインパス演算子 `->>` を使用します:

```
mysql> SELECT sentence->>"$.mascot" FROM facts;
+-----+
| sentence->>"$.mascot" |
+-----+
| Our mascot is a dolphin named "Sakila". |
+-----+
```

注記

前述の例は、`NO_BACKSLASH_ESCAPES` サーバーの SQL モードが有効になっている場合は動作しません。このモードが設定されている場合、ダブルバックスラッシュのかわりに単一のバックスラッシュを使用して JSON オブジェクトリテラルを挿入でき、バックスラッシュは保持されます。挿入の実行時に `JSON_OBJECT()` 関数を使用し、このモードが設定されている場合は、次のように一重引用符および二重引用符を使用する必要があります:

```
mysql> INSERT INTO facts VALUES
> (JSON_OBJECT('mascot', 'Our mascot is a dolphin named "Sakila".));
```

JSON 値のエスケープ文字に対するこのモードの影響の詳細は、`JSON_UNQUOTE()` 関数の説明を参照してください。

JSON 値の正規化、マージおよび自動ラップ

文字列が解析され、有効な JSON ドキュメントであることが判明すると、文字列も正規化されます。つまり、ドキュメントの後半で見つかったキーを複製するキーを持つメンバーは、左から右に読み取られて破棄されます。次の `JSON_OBJECT()` コールによって生成されるオブジェクト値には、次に示すように、そのキー名が値の前に出現するため、2 番目の `key1` 要素のみが含まれます:

```
mysql> SELECT JSON_OBJECT('key1', 1, 'key2', 'abc', 'key1', 'def');
+-----+
| JSON_OBJECT("key1", 1, "key2", "abc", "key1", "def") |
```

```
+-----+
|{"key1": "def", "key2": "abc"} |
+-----+
```

正規化は、次に示すように、値が JSON カラムに挿入されるときにも実行されます:

```
mysql> CREATE TABLE t1 (c1 JSON);

mysql> INSERT INTO t1 VALUES
  > ({"x": 17, "x": "red"}),
  > ({"x": 17, "x": "red", "x": [3, 5, 7]});

mysql> SELECT c1 FROM t1;
+-----+
|c1      |
+-----+
|{"x": "red"} |
|{"x": [3, 5, 7]} |
+-----+
```

この「最後の重複キー優先」の動作は、[RFC 7159](#) によって推奨され、ほとんどの JavaScript パーサーによって実装されます。(Bug #86866、Bug #26369555)

8.0.3 より前のバージョンの MySQL では、ドキュメント内で以前に見つかったキーを複製したキーを持つメンバーは破棄されました。次の `JSON_OBJECT()` コールによって生成されたオブジェクト値には、2 番目の `key1` 要素は含まれません。これは、そのキー名が値の前にあるためです:

```
mysql> SELECT JSON_OBJECT("key1", 1, "key2", 'abc', "key1", 'def');
+-----+
|JSON_OBJECT("key1", 1, "key2", 'abc', "key1", 'def') |
+-----+
|{"key1": 1, "key2": "abc"} |
+-----+
```

MySQL 8.0.3 より前は、この「最初の重複キー優先」正規化は JSON カラムに値を挿入するときにも実行されていました。

```
mysql> CREATE TABLE t1 (c1 JSON);

mysql> INSERT INTO t1 VALUES
  > ({"x": 17, "x": "red"}),
  > ({"x": 17, "x": "red", "x": [3, 5, 7]});

mysql> SELECT c1 FROM t1;
+-----+
|c1      |
+-----+
|{"x": 17}|
|{"x": 17}|
+-----+
```

MySQL では、元の JSON ドキュメント内のキー、値または要素間の余分な空白も破棄され、各カンマ (,) またはコロン (:) の後に単一の空白が表示されたままになります (または必要に応じて挿入されます)。これは、読みやすさを高めるために行われます。

JSON 値を生成する MySQL 関数 ([セクション 12.18.2 「JSON 値を作成する関数」](#) を参照) は、常に正規化された値を返します。

ルックアップをより効率的にするために、MySQL では JSON オブジェクトのキーもソートされます。この順序付けの結果は変更される可能性があり、リリース間での一貫性が保証されないことに注意してください。

JSON 値のマージ

関数 `JSON_MERGE_PRESERVE()` および `JSON_MERGE_PATCH()` によって実装される MySQL 8.0.3 (以降) では、2 つのマージアルゴリズムがサポートされています。重複キーの処理方法が異なります:
`JSON_MERGE_PRESERVE()` では重複キーの値が保持されますが、`JSON_MERGE_PATCH()` では最後の値以外のすべての値が破棄されます。次のいくつかの段落では、これら 2 つの関数のそれぞれが JSON ドキュメント (つまり、オブジェクトと配列) の様々な組合せのマージを処理する方法について説明します。

注記

`JSON_MERGE_PRESERVE()` は、以前のバージョンの MySQL (MySQL 8.0.3 で名前が変更された) で検出された `JSON_MERGE()` 関数と同じです。 `JSON_MERGE()` は、MySQL 8.0 で `JSON_MERGE_PRESERVE()` のエイリアスとして引き続きサポートされていますが、非推奨であり、将来のリリースで削除される可能性があります。

配列のマージ。 複数の配列を組み合わせるコンテキストでは、配列は単一の配列にマージされます。 `JSON_MERGE_PRESERVE()` では、後で名前を付けた配列を最初の配列の最後に連結することで、これを行います。 `JSON_MERGE_PATCH()` は、各引数を単一の要素で構成される配列とみなし (したがって、インデックスとして 0 を持つ)、「最後の重複キー優先」ロジックを適用して最後の引数のみを選択します。 次のクエリーで表示される結果を比較できます:

```
mysql> SELECT
-> JSON_MERGE_PRESERVE('[1, 2]', ['a', 'b', 'c'], [true, false]) AS Preserve,
-> JSON_MERGE_PATCH('[1, 2]', ['a', 'b', 'c'], [true, false]) AS Patch\G
***** 1. row *****
Preserve: [1, 2, "a", "b", "c", true, false]
Patch: [true, false]
```

マージ時に複数のオブジェクトを使用すると、単一のオブジェクトが生成されます。 `JSON_MERGE_PRESERVE()` は、配列内のそのキーの一意の値をすべて組み合わせることで、同じキーを持つ複数のオブジェクトを処理します。この配列は、結果でそのキーの値として使用されます。 `JSON_MERGE_PATCH()` では、左から右に向かって重複キーが見つかった値が破棄されるため、結果にはそのキーの最後の値のみが含まれます。 次のクエリーは、重複キー `a` の結果の違いを示しています:

```
mysql> SELECT
-> JSON_MERGE_PRESERVE('{"a": 1, "b": 2}', '{"c": 3, "a": 4}', '{"c": 5, "d": 3}') AS Preserve,
-> JSON_MERGE_PATCH('{"a": 3, "b": 2}', '{"c": 3, "a": 4}', '{"c": 5, "d": 3}') AS Patch\G
***** 1. row *****
Preserve: {"a": [1, 4], "b": 2, "c": [3, 5], "d": 3}
Patch: {"a": 4, "b": 2, "c": 5, "d": 3}
```

配列値を必要とするコンテキストで使用される非配列値は、自動ラップされます: この値は、配列に変換するために `[および]` 文字で囲まれています。 次のステートメントでは、各引数が配列 (`[1]`、`[2]`) として自動ラップされます。 これらはマージされて単一の結果配列が生成されます。 前述の 2 つの場合と同様に、`JSON_MERGE_PRESERVE()` は同じキーを持つ値を結合し、`JSON_MERGE_PATCH()` は最後のキーを除くすべての重複キーの値を破棄します:

```
mysql> SELECT
-> JSON_MERGE_PRESERVE('1', '2') AS Preserve,
-> JSON_MERGE_PATCH('1', '2') AS Patch\G
***** 1. row *****
Preserve: [1, 2]
Patch: 2
```

配列およびオブジェクト値をマージするには、次の例に示すように、オブジェクトを配列として自動ラップし、マージ機能の選択に従って値を組み合わせるか、「最後の重複キー優先」によって配列をマージします (それぞれ `JSON_MERGE_PRESERVE()` または `JSON_MERGE_PATCH()`):

```
mysql> SELECT
-> JSON_MERGE_PRESERVE('[10, 20]', '{"a": "x", "b": "y"}') AS Preserve,
-> JSON_MERGE_PATCH('[10, 20]', '{"a": "x", "b": "y"}') AS Patch\G
***** 1. row *****
Preserve: [10, 20, {"a": "x", "b": "y"}]
Patch: {"a": "x", "b": "y"}
```

JSON 値の検索および変更

JSON パス式は、JSON ドキュメント内の値を選択します。

パス式は、JSON ドキュメントの一部を抽出または変更して、そのドキュメント内のどこで操作するかを指定する関数で役立ちます。 たとえば、次のクエリーは、`name` キーを持つメンバーの値を JSON ドキュメントから抽出します:

```
mysql> SELECT JSON_EXTRACT('{"id": 14, "name": "Aztalan"}', '$.name');
+-----+
| JSON_EXTRACT('{"id": 14, "name": "Aztalan"}', '$.name') |
+-----+
| "Aztalan" |
```


パス構文では、検討中の JSON ドキュメントを表すために先頭の \$ 文字が使用され、オプションで、ドキュメントの連続して具体的な部分を示すセレクタが続きます:

- ピリオドの後にキー名を指定すると、オブジェクト内のメンバーに特定のキーが付けられます。引用符のない名前がパス式内で有効でない場合 (たとえば、空白が含まれている場合)、キー名は二重引用符で囲む必要があります。
- 配列を選択する `path` に追加された `[N]` は、配列内の `N` の位置にある値に名前を付けます。配列の位置はゼロで始まる整数です。 `path` が配列値を選択しない場合、 `path [0]` は `path` と同じ値に評価されます:

```
mysql> SELECT JSON_SET('x', '$[0]', 'a');
+-----+
| JSON_SET('x', '$[0]', 'a') |
+-----+
| "a" |
+-----+
1 row in set (0.00 sec)
```

- `[M to N]` では、位置 `M` の値で始まり、位置 `N` の値で終わる配列値のサブセットまたは範囲を指定します。

`last` は、右端の配列要素のインデックスのシノニムとしてサポートされています。配列要素の相対アドレス指定もサポートされています。 `path` が配列値を選択しない場合、このセクションの後半に示すように、 `path[last]` は `path` と同じ値に評価されます (右端の配列要素を参照)。

- パスには、 `*` または `**` ワイルドカードを含めることができます:
 - `[*]` は、JSON オブジェクトのすべてのメンバーの値に評価されます。
 - `[**]` は、JSON 配列内のすべての要素の値に評価されます。
 - `prefix**suffix` は、名前付き接頭辞で始まり、名前付き接尾辞で終わるすべてのパスに評価されます。
- ドキュメントに存在しないパス (存在しないデータに評価される) は、 `NULL` に評価されます。

\$ では、次の 3 つの要素を使用してこの JSON 配列を参照します:

```
[3, {"a": [5, 6], "b": 10}, [99, 100]]
```

このとき、次のようになります。

- `$(0)` は、 `3` に評価されます。
- `$(1)` は、 `{"a": [5, 6], "b": 10}` に評価されます。
- `$(2)` は、 `[99, 100]` に評価されます。
- `$(3)` は `NULL` に評価されます (存在しない 4 番目の配列要素を参照します)。

`$(1)` および `$(2)` は非スカラー値と評価されるため、ネストされた値を選択するより具体的なパス式の基礎として使用できます。例:

- `$(1).a` は、 `[5, 6]` に評価されます。
- `$(1).a[1]` は、 `6` に評価されます。
- `$(1).b` は、 `10` に評価されます。
- `$(2)[0]` は、 `99` に評価されます。

前述のように、引用符で囲まれていないキー名がパス式で有効でない場合は、キーに名前を付けるパスコンポーネントを引用符で囲む必要があります。 \$ がこの値を参照するようにします:

```
{"a fish": "shark", "a bird": "sparrow"}
```

キーにはスペースが含まれているため、引用符で囲む必要があります:

- `$. "a fish"` は、 `shark` に評価されます。

- `$.a bird`は、`sparrow` に評価されます。

ワイルドカードを使用するパスは、複数の値を含むことができる配列に評価されます:

```
mysql> SELECT JSON_EXTRACT('{\"a\": 1, \"b\": 2, \"c\": [3, 4, 5]}', '$.*');
+-----+
| JSON_EXTRACT('{\"a\": 1, \"b\": 2, \"c\": [3, 4, 5]}', '$.*') |
+-----+
| [1, 2, [3, 4, 5]] |
+-----+
mysql> SELECT JSON_EXTRACT('{\"a\": 1, \"b\": 2, \"c\": [3, 4, 5]}', '$.c[*]');
+-----+
| JSON_EXTRACT('{\"a\": 1, \"b\": 2, \"c\": [3, 4, 5]}', '$.c[*]') |
+-----+
| [3, 4, 5] |
+-----+
```

次の例では、`**b` が複数のパス (`$.a.b` および `$.c.b`) に評価し、一致するパス値の配列を生成します:

```
mysql> SELECT JSON_EXTRACT('{\"a\": {\"b\": 1}, \"c\": {\"b\": 2}}', '$**b');
+-----+
| JSON_EXTRACT('{\"a\": {\"b\": 1}, \"c\": {\"b\": 2}}', '$**b') |
+-----+
| [1, 2] |
+-----+
```

JSON 配列からの範囲。 範囲を `to` キーワードとともに使用して、JSON 配列のサブセットを指定できます。たとえば、次に示すように、`$(1 to 3)`には配列の 2 番目、3 番目および 4 番目の要素が含まれます:

```
mysql> SELECT JSON_EXTRACT('[1, 2, 3, 4, 5]', '$[1 to 3]');
+-----+
| JSON_EXTRACT('[1, 2, 3, 4, 5]', '$[1 to 3]') |
+-----+
| [2, 3, 4] |
+-----+
1 row in set (0.00 sec)
```

構文は `M to N` です。ここで、`M` と `N` はそれぞれ、JSON 配列の要素の範囲の最初と最後のインデックスです。`N` は `M` より大きい必要があります。`M` は 0 以上である必要があります。配列要素は 0 から始まるインデックス付けされます。

ワイルドカードがサポートされているコンテキストで範囲を使用できます。

右端の配列要素。 `last` キーワードは、配列の最後の要素のインデックスのシノニムとしてサポートされています。`last - N` 形式の式は、次のように、相対アドレス指定および範囲定義内で使用できます:

```
mysql> SELECT JSON_EXTRACT('[1, 2, 3, 4, 5]', '$[last-3 to last-1]');
+-----+
| JSON_EXTRACT('[1, 2, 3, 4, 5]', '$[last-3 to last-1]') |
+-----+
| [2, 3, 4] |
+-----+
1 row in set (0.01 sec)
```

パスが配列ではない値に対して評価される場合、評価の結果は値が単一要素配列にラップされた場合と同じです:

```
mysql> SELECT JSON_REPLACE('\"Sakila\"', '$[last]', 10);
+-----+
| JSON_REPLACE('\"Sakila\"', '$[last]', 10) |
+-----+
| 10 |
+-----+
1 row in set (0.00 sec)
```

`column->path` は、JSON カラム識別子および JSON パス式とともに `JSON_EXTRACT(column, path)` のシノニムとして使用できます。詳しくは[セクション12.18.3「JSON 値を検索する関数」](#)をご覧ください。JSON カラムインデックスを提供するための生成されたカラムのインデックス付けも参照してください。

一部の関数では、既存の JSON ドキュメントを取得し、なんらかの方法で変更して、結果として変更されたドキュメントを戻します。パス式は、変更を加えるドキュメント内の場所を示します。たとえ

ば、`JSON_SET()`、`JSON_INSERT()` および `JSON_REPLACE()` の各関数は、JSON ドキュメントに加えて、ドキュメントを変更する場所と使用する値を記述する 1 つ以上のパスと値のペアを取ります。関数は、ドキュメント内の既存の値と存在しない値の処理方法が異なります。

このドキュメントについて考えてみます:

```
mysql> SET @j = '{"a", {"b": [true, false]}, [10, 20]};
```

`JSON_SET()` は、存在するパスの値を置き換え、存在しないパスの値を追加します。

```
mysql> SELECT JSON_SET(@j, '$[1].b[0]', 1, '$[2][2]', 2);
+-----+
| JSON_SET(@j, '$[1].b[0]', 1, '$[2][2]', 2) |
+-----+
| [{"a": {"b": [1, false]}, [10, 20, 2]}] |
+-----+
```

この場合、パス `[$[1].b[0]]` は既存の値 (`true`) を選択します。これは、パス引数 (1) の後の値に置き換えられます。パス `[$[2][2]]` が存在しないため、`[$[2]]` によって選択された値に対応する値 (2) が追加されます。

`JSON_INSERT()` によって新しい値が追加されますが、既存の値は置換されません:

```
mysql> SELECT JSON_INSERT(@j, '$[1].b[0]', 1, '$[2][2]', 2);
+-----+
| JSON_INSERT(@j, '$[1].b[0]', 1, '$[2][2]', 2) |
+-----+
| [{"a": {"b": [true, false]}, [10, 20, 2]}] |
+-----+
```

`JSON_REPLACE()` は既存の値を置換し、新しい値を無視します:

```
mysql> SELECT JSON_REPLACE(@j, '$[1].b[0]', 1, '$[2][2]', 2);
+-----+
| JSON_REPLACE(@j, '$[1].b[0]', 1, '$[2][2]', 2) |
+-----+
| [{"a": {"b": [1, false]}, [10, 20]}] |
+-----+
```

パスと値のペアは左から右に評価されます。あるペアを評価して生成されたドキュメントは、次のペアが評価される新しい値になります。

`JSON_REMOVE()` は、JSON ドキュメントと、ドキュメントから削除する値を指定する 1 つ以上のパスを取ります。戻り値は、元のドキュメントから、ドキュメント内に存在するパスによって選択された値を引いたものです:

```
mysql> SELECT JSON_REMOVE(@j, '$[2]', '$[1].b[1]', '$[1].b[1]');
+-----+
| JSON_REMOVE(@j, '$[2]', '$[1].b[1]', '$[1].b[1]') |
+-----+
| [{"a": {"b": [true]}}] |
+-----+
```

パスには次の効果があります:

- `[$[2]]` は、`[10, 20]` を照合して削除します。
- `[$[1].b[1]]` の最初のインスタンスは、`b` 要素内の `false` と一致し、削除されます。
- `[$[1].b[1]]` の 2 番目のインスタンスが一致しません: その要素はすでに削除されており、パスは存在せず、効果もありません。

JSON パス構文

MySQL でサポートされ、このマニュアルの他の場所で説明されている JSON 関数の多く ([セクション 12.18 「JSON 関数」](#) を参照) では、JSON ドキュメント内の特定の要素を識別するためにパス式が必要です。パスは、パススコープとそれに続く 1 つ以上のパスレグで構成されます。MySQL JSON 関数で 사용되는パスの場合、有効範囲は常に検索または操作されるドキュメントで、先頭の `$` 文字で表されます。パスレグはピリオド文字 (`.`) で区切られます。配列内のセルは `[N]` で表され、`N` は負でない整数です。キーの名前は、二重引用符で囲まれた文字列または有効

な ECMAScript 識別子である必要があります (「ECMAScript 言語仕様」の「[識別子名および識別子](#)」を参照)。JSON テキストなどのパス式は、`ascii`、`utf8` または `utf8mb4` 文字セットを使用してエンコードする必要があります。その他の文字エンコーディングは、暗黙的に `utf8mb4` に強制変換されます。完全な構文は次のとおりです:

```
pathExpression:
  scope[(pathLeg)*]

pathLeg:
  member | arrayLocation | doubleAsterisk

member:
  period ( keyName | asterisk )

arrayLocation:
  leftBracket ( nonNegativeInteger | asterisk ) rightBracket

keyName:
  ESIdentifier | doubleQuotedString

doubleAsterisk:
  '**'

period:
  '.'

asterisk:
  '*'

leftBracket:
  '['

rightBracket:
  ']'
```

前述のように、MySQL では、パスのスコープは常に操作対象のドキュメントで、`$` として表されます。`'$'` を JSON パス式のドキュメントの構文として使用できます。

注記

一部の実装では、JSON パスのスコープのカラム参照がサポートされています。現在、MySQL ではこれらはサポートされていません。

ワイルドカード `*` および `**` トークンは、次のように使用されます:

- `*` は、オブジェクト内のすべてのメンバーの値を表します。
- `[*]` は、配列内のすべてのセルの値を表します。
- `[prefix]**suffix` は、`prefix` で始まり `suffix` で終わるすべてのパスを表します。`prefix` はオプションですが、`suffix` は必須です。つまり、パスが `**` で終わることはできません。

また、パスに順序 `***` が含まれていない場合もあります。

パス構文の例は、[JSON_CONTAINS_PATH\(\)](#)、[JSON_SET\(\)](#)、[JSON_REPLACE\(\)](#) などの引数としてパスを取る様々な JSON 関数の説明を参照してください。`*` および `**` ワイルドカードの使用を含む例は、[JSON_SEARCH\(\)](#) 関数の説明を参照してください。

MySQL 8.0.2 以降では、`to` キーワード (`[$2 to 10]` など) を使用した JSON 配列のサブセットの範囲表記、および配列の右端の要素のシノニムとして `last` キーワードもサポートされています。詳細および例については、[JSON 値の検索および変更](#) を参照してください。

JSON 値の比較および順序付け

JSON 値は、`=`、`<`、`<=`、`>`、`>=`、`<>`、`!=` および `<=>` 演算子を使用して比較できます。

次の比較演算子および関数は、JSON 値ではまだサポートされていません:

- `BETWEEN`

- [IN\(\)](#)
- [GREATEST\(\)](#)
- [LEAST\(\)](#)

リストされている比較演算子および関数の回避策は、JSON 値をネイティブの MySQL 数値または文字列データ型にキャストして、JSON 以外のスカラー型が一貫しているようにすることです。

JSON 値の比較は 2 つのレベルで行われます。最初のレベルの比較は、比較された値の JSON 型に基づきます。タイプが異なる場合、比較結果は優先順位の高いタイプによってのみ決定されます。2 つの値が同じ JSON 型である場合、2 つ目のレベルの比較は型固有のルールを使用して行われます。

次のリストは、JSON 型の優先順位の高いものから低いものへの優先順位を示しています。(型名は、[JSON_TYPE\(\)](#) 関数によって戻される型名です。) 行にまとめて表示されるタイプの優先順位は同じです。リストの前半にリストされている JSON 型を持つ値は、リストの後半にリストされている JSON 型を持つ値よりも大きく比較されます。

```
BLOB
BIT
OPAQUE
DATETIME
TIME
DATE
BOOLEAN
ARRAY
OBJECT
STRING
INTEGER, DOUBLE
NULL
```

同じ優先順位の JSON 値の場合、比較ルールはタイプ固有です:

- [BLOB](#)

2 つの値の最初の **N** バイトが比較されます。ここで、**N** は短い値のバイト数です。2 つの値の最初の **N** バイトが同一の場合、短い方の値が長い方の値の前に順序付けされます。

- [BIT](#)

[BLOB](#) と同じルール。

- [OPAQUE](#)

[BLOB](#) と同じルール。 [OPAQUE](#) 値は、他のタイプのいずれにも分類されない値です。

- [DATETIME](#)

以前のポイントインタイムを表す値は、後のポイントインタイムを表す値の前に順序付けられます。最初に MySQL [DATETIME](#) 型と [TIMESTAMP](#) 型の値がそれぞれ同じ時点を表す場合、それらは等しくなります。

- [TIME](#)

2 つの時間値のうち小さい方が大きい方の値の前に順序付けられます。

- [DATE](#)

以前の日付は、より新しい日付より前にオーダーされます。

- [ARRAY](#)

長さが同じで、配列内の対応する位置の値が等しい場合、2 つの JSON 配列は等しくなります。

配列が等しくない場合、配列の順序は、違いがある最初の位置の要素によって決まります。その位置の値が小さい配列が最初に順序付けられます。短い配列のすべての値が長い配列の対応する値と等しい場合は、短い配列が最初に順序付けられます。

例:

```
[] < ["a"] < ["ab"] < ["ab", "cd", "ef"] < [{"ab", "ef"}]
```

• BOOLEAN

JSON false リテラルが JSON true リテラルより小さい場合。

• OBJECT

2 つの JSON オブジェクトは、同じキーセットを持ち、各キーが両方のオブジェクトで同じ値を持つ場合に等しくなります。

例:

```
{"a": 1, "b": 2} = {"b": 2, "a": 1}
```

等しくない 2 つのオブジェクトの順序は指定されませんが、決定的です。

• STRING

文字列は、比較される 2 つの文字列の `utf8mb4` 表現の最初の `N` バイトで字句的に順序付けされます (`N` は短い文字列の長さです)。2 つの文字列の最初の `N` バイトが同一の場合、短い文字列は長い文字列より小さいとみなされます。

例:

```
"a" < "ab" < "b" < "bc"
```

この順序付けは、照合 `utf8mb4_bin` を使用した SQL 文字列の順序付けと同等です。`utf8mb4_bin` はバイナリ照合であるため、JSON 値の比較では大/小文字が区別されます:

```
"A" < "a"
```

• INTEGER, DOUBLE

JSON 値には、正確な値の数値および近似値の数値を含めることができます。これらのタイプの数値の概要は、[セクション9.1.2「数値リテラル」](#)を参照してください。

ネイティブ MySQL 数値型を比較するためのルールは [セクション12.3「式評価での型変換」](#)で説明されていますが、JSON 値内の数値を比較するためのルールは多少異なります:

- ネイティブの MySQL `INT` 数値型と `DOUBLE` 数値型を使用する 2 つのカラムの比較では、すべての比較に整数と double が含まれることがわかっているため、すべての行で整数が double に変換されます。つまり、正確な値の数値は近似値の数値に変換されます。
- 一方、クエリーで数値を含む 2 つの JSON カラムを比較する場合、数値が整数であるか倍精度であるかは事前にわかりません。すべての行で最も一貫性のある動作を提供するために、MySQL は近似値の数値を正確な値の数値に変換します。結果の順序付けは一貫性があり、正確な値の数値の精度は失われません。たとえば、スカラー 9223372036854775805、9223372036854775806、9223372036854775807 および 9.223372036854776 e18 の場合、順序は次のようになります:

```
9223372036854775805 < 9223372036854775806 < 9223372036854775807  
< 9.223372036854776e18 = 9223372036854776000 < 9223372036854776001
```

JSON 以外の数値比較ルールを使用するための JSON 比較では、順序に一貫性がない可能性があります。数値の通常の MySQL 比較ルールでは、次の順序付けが行われます:

- 整数比較:

```
9223372036854775805 < 9223372036854775806 < 9223372036854775807
```

(9.223372036854776 e18 には定義されていません)

- 二重比較:

```
9223372036854775805 = 9223372036854775806 = 9223372036854775807 = 9.223372036854776e18
```


JSON 値を SQL `NULL` と比較する場合、結果は `UNKNOWN` になります。

JSON 値と非 JSON 値を比較するために、非 JSON 値は次のテーブルのルールに従って JSON に変換されてから、前述のように比較されます。

JSON 値と非 JSON 値の間の変換

次のテーブルに、JSON 値と他のタイプの値の間のカスト時に MySQL が従うルールのサマリーを示します：

表 11.3 JSON 変換ルール

その他のタイプ	CAST(other type AS JSON)	CAST(JSON AS other type)
JSON	変更なし	変更なし
utf8 文字 (utf8mb4, utf8, ascii)	文字列は JSON 値に解析されます。	JSON 値は utf8mb4 文字列にシリアライズされます。
その他の文字タイプ	その他の文字エンコーディングは、暗黙的に utf8mb4 に変換され、utf8 文字タイプで説明されているように扱われます。	JSON 値は utf8mb4 文字列にシリアライズされ、他の文字エンコーディングにキャストされます。結果が意味を持たない場合があります。
NULL	JSON 型の NULL 値になります。	該当なし。
ジオメトリタイプ	ジオメトリ値は、 <code>ST_AsGeoJSON()</code> をコールして JSON ドキュメントに変換されます。	不正な操作です。回避策： <code>CAST(json_val AS CHAR)</code> の結果を <code>ST_GeomFromGeoJSON()</code> に渡します。
その他すべてのタイプ	単一のスカラー値で構成される JSON ドキュメントになります。	JSON ドキュメントがターゲット型の単一のスカラー値で構成され、そのスカラー値をターゲット型にキャストできる場合は成功します。それ以外の場合は、 <code>NULL</code> を返し、警告を生成します。

JSON 値の `ORDER BY` および `GROUP BY` は、次の原則に従って機能します：

- スカラー JSON 値の順序付けでは、前述のルールと同じルールが使用されます。
- 昇順ソートの場合、SQL `NULL` は JSON `null` リテラルを含むすべての JSON 値の前に順序付けられます。降順ソートの場合、SQL `NULL` は JSON `null` リテラルを含むすべての JSON 値の後に順序付けされます。
- JSON 値のソートキーは `max_sort_length` システム変数の値によってバインドされるため、最初の `max_sort_length` バイトの後にのみ異なるキーは等しいと比較されます。
- 非スカラー値のソートは現在サポートされておらず、警告が発生します。

ソートの場合、JSON スカラーを他のネイティブ MySQL 型にキャストすると便利です。たとえば、`jdoc` という名前のカラムに、`id` キーと負でない値で構成されるメンバーを持つ JSON オブジェクトが含まれる場合、次の式を使用して `id` 値でソートします：

```
ORDER BY CAST(JSON_EXTRACT(jdoc, '$.id') AS UNSIGNED)
```

生成されたカラムが `ORDER BY` と同じ式を使用するように定義されている場合、MySQL オプティマイザはそれを認識し、クエリー実行計画のインデックスの使用を検討します。セクション 8.3.11 「生成されたカラムインデックスのオプティマイザによる使用」を参照してください。

JSON 値の集計

JSON 値の集計では、SQL `NULL` 値は他のデータ型と同様に無視されます。NULL 以外の値は数値型に変換され、`MIN()`、`MAX()` および `GROUP_CONCAT()` を除いて集計されます。数値への変換では、数値スカラーである JSON 値に対して意味のある結果が生成される必要がありますが、(値によっては) 精度の切捨ておよび損失が発生する可能性があります。他の JSON 値の数に変換しても、意味のある結果が得られない場合があります。

11.6 データ型デフォルト値

データ型指定には、明示的または暗黙的なデフォルト値を指定できます。

データ型指定の `DEFAULT value` 句は、カラムのデフォルト値を明示的に示します。例:

```
CREATE TABLE t1 (  
  i INT DEFAULT -1,  
  c VARCHAR(10) DEFAULT "",  
  price DOUBLE(16,2) DEFAULT 0.00  
);
```

`SERIAL DEFAULT VALUE` は特殊なケースです。整数カラムの定義では、これは `NOT NULL AUTO_INCREMENT UNIQUE` のエイリアスです。

明示的な `DEFAULT` 句の処理の一部の側面は、次に説明するように、バージョンに依存します。

- [MySQL 8.0.13 での明示的なデフォルト処理](#)
- [MySQL 8.0.13 より前の明示的なデフォルト処理](#)
- [暗黙的なデフォルト処理](#)

MySQL 8.0.13 での明示的なデフォルト処理

`DEFAULT` 句で指定されるデフォルト値は、リテラル定数または式です。例外として、式のデフォルト値をカッコで囲み、リテラル定数のデフォルト値と区別します。例:

```
CREATE TABLE t1 (  
  -- literal defaults  
  i INT DEFAULT 0,  
  c VARCHAR(10) DEFAULT "",  
  -- expression defaults  
  f FLOAT DEFAULT (RAND() * RAND()),  
  b BINARY(16) DEFAULT (UUID_TO_BIN(UUID())),  
  d DATE DEFAULT (CURRENT_DATE + INTERVAL 1 YEAR),  
  p POINT DEFAULT (Point(0,0)),  
  j JSON DEFAULT (JSON_ARRAY())  
);
```

ただし、`TIMESTAMP` カラムおよび `DATETIME` カラムの場合は、カッコを囲まずに `CURRENT_TIMESTAMP` 関数をデフォルトとして指定できます。 [セクション11.2.5「TIMESTAMP および DATETIME の自動初期化および更新機能」](#)を参照してください。

`BLOB`, `TEXT`, `GEOMETRY` および `JSON` データ型にデフォルト値を割り当てることができるのは、式の値がリテラルの場合でも、値が式として書き込まれる場合のみです:

- これは許可されています (式として指定されたリテラルのデフォルト):

```
CREATE TABLE t2 (b BLOB DEFAULT ('abc'));
```

- これにより、エラーが生成されます (リテラルのデフォルトが式として指定されていません):

```
CREATE TABLE t2 (b BLOB DEFAULT 'abc');
```

式のデフォルト値は、次のルールに従う必要があります。許可されていない構造が式に含まれている場合は、エラーが発生します。

- リテラル、組み込み関数 (決定的および非決定的の両方)、および演算子が許可されます。
- サブクエリー、パラメータ、変数、ストアドファンクションおよびユーザー定義関数は使用できません。
- 式のデフォルト値は、`AUTO_INCREMENT` 属性を持つカラムに依存できません。
- あるカラムの式のデフォルト値は他のテーブルのカラムを参照できますが、生成されたカラムまたは式のデフォルト値を持つカラムへの参照は、テーブル定義の前半で発生したカラムにする必要があります。つまり、式のデフォ

ルト値には、生成されたカラムまたは式のデフォルト値を持つカラムへのフォワード参照を含めることはできません。

順序付け制約は、[ALTER TABLE](#) を使用してテーブルのカラムを並べ替える場合にも適用されます。結果のテーブルに、式のデフォルト値を持つ生成されたカラムまたはカラムへのフォワード参照を含む式のデフォルト値が含まれる場合、ステートメントは失敗します。

注記

式のデフォルト値のいずれかのコンポーネントが SQL モードに依存している場合、すべての使用中に SQL モードが同じでないかぎり、テーブルの使用方法によって異なる結果が発生する可能性があります。

[CREATE TABLE ... LIKE](#) および [CREATE TABLE ... SELECT](#) の場合、宛先テーブルには元のテーブルの式のデフォルト値が保持されます。

式のデフォルト値が非決定的関数を参照している場合、式を評価するステートメントはステートメントベースレプリケーションでは安全ではありません。これには、[INSERT](#) や [UPDATE](#) などのステートメントが含まれます。この場合、バイナリロギングが無効になっていると、ステートメントは通常どおりに実行されます。バイナリロギングが有効で、[binlog_format](#) が [STATEMENT](#) に設定されている場合、ステートメントはログに記録されて実行されますが、レプリケーションスレーブが相違する可能性があるため、警告メッセージがエラーログに書き込まれます。[binlog_format](#) が [MIXED](#) または [ROW](#) に設定されている場合、ステートメントは通常どおりに実行されます。

新しい行を挿入する場合、式 default を持つカラムのデフォルト値を挿入するには、カラム名を省略するか、カラムを [DEFAULT](#) として指定します (リテラル default を持つカラムの場合と同様):

```
mysql> CREATE TABLE t4 (uid BINARY(16) DEFAULT (UUID_TO_BIN(UUID())));
mysql> INSERT INTO t4 () VALUES();
mysql> INSERT INTO t4 () VALUES(DEFAULT);
mysql> SELECT BIN_TO_UUID(uid) AS uid FROM t4;
```

```
+-----+
| uid          |
+-----+
| f1109174-94c9-11e8-971d-3bf1095aa633 |
| f110cf9a-94c9-11e8-971d-3bf1095aa633 |
+-----+
```

ただし、名前付きカラムのデフォルト値を指定するための [DEFAULT\(col_name\)](#) の使用は、リテラルのデフォルト値を持つカラムにのみ許可され、式のデフォルト値を持つカラムには許可されません。

すべてのストレージエンジンが式のデフォルト値を許可するわけではありません。そうでない場合は、[ER_UNSUPPORTED_ACTION_ON_DEFAULT_VAL_GENERATED](#) エラーが発生します。

デフォルト値が宣言されたカラム型とは異なるデータ型に評価された場合、宣言された型への暗黙的な強制は、通常の MySQL 型変換ルールに従って行われます。[セクション12.3「式評価での型変換」](#)を参照してください。

MySQL 8.0.13 より前の明示的なデフォルト処理

例外として、[DEFAULT](#) 句で指定されるデフォルト値はリテラル定数である必要があります。関数または式は使用できません。これは、たとえば日付カラムのデフォルト値に [NOW\(\)](#) や [CURRENT_DATE](#) などの関数の値を設定できないことを意味します。ただし、[TIMESTAMP](#) カラムおよび [DATETIME](#) カラムの場合は、[CURRENT_TIMESTAMP](#) をデフォルトとして指定できます。[セクション11.2.5「TIMESTAMP および DATETIME の自動初期化および更新機能」](#)を参照してください。

[BLOB](#)、[TEXT](#)、[GEOMETRY](#) および [JSON](#) データ型にデフォルト値を割り当てることはできません。

デフォルト値が宣言されたカラム型とは異なるデータ型に評価された場合、宣言された型への暗黙的な強制は、通常の MySQL 型変換ルールに従って行われます。[セクション12.3「式評価での型変換」](#)を参照してください。

暗黙的なデフォルト処理

データ型指定に明示的な [DEFAULT](#) 値が含まれていない場合、MySQL は次のようにデフォルト値を決定します:

[NULL](#) を値として取ることができる場合は、そのカラムは明示的な [DEFAULT NULL](#) 句で定義ができます。

カラムが値として `NULL` を取ることができない場合、MySQL は明示的な `DEFAULT` 句を使用せずにカラムを定義します。

明示的な `DEFAULT` 句のない `NOT NULL` カラムに対するデータエントリでは、`INSERT` または `REPLACE` ステートメントにカラムの値が含まれていない場合、または `UPDATE` ステートメントがカラムを `NULL` に設定する場合、MySQL はその時点で有効な SQL モードに従ってカラムを処理します。

- 厳密な SQL モードを有効にした場合、トランザクションテーブルに対してエラーが発生し、ステートメントがロールバックされます。非トランザクションテーブルではエラーが発生しますが、これが複数行ステートメントの 2 行目または後続の行で発生した場合は、前の行が挿入されます。
- 厳密モードが有効でない場合、MySQL はカラムデータ型の暗黙的なデフォルト値にカラムを設定します。

テーブル `t` が次のように定義されるとします。

```
CREATE TABLE t (i INT NOT NULL);
```

この場合、`i` は明示的なデフォルトがないので、厳密モードでは次のそれぞれはステートメントはエラーになり、行は挿入されません。厳密モードを使用しない場合、3 番目のステートメントだけでエラーが発生します。最初の 2 つのステートメントでは暗黙のデフォルトが挿入されますが、`DEFAULT(i)` が値を生成できないので 3 番目のステートメントは失敗します。

```
INSERT INTO t VALUES();  
INSERT INTO t VALUES(DEFAULT);  
INSERT INTO t VALUES(DEFAULT(i));
```

[セクション 5.1.11 「サーバー SQL モード」](#) を参照してください。

特定のテーブルについて、`SHOW CREATE TABLE` ステートメントは明示的な `DEFAULT` 句を持つカラムを表示します。

暗黙的なデフォルトは次のように定義されます。

- 数値型のデフォルトは `0` です。ただし、例外として `AUTO_INCREMENT` 属性で宣言された整数型または浮動小数点型のデフォルトは、そのシーケンスの次の値になります。
- `TIMESTAMP` 以外の日付と時間型のデフォルトには、「ゼロ」値が適切です。[explicit_defaults_for_timestamp](#) システム変数が有効な場合、これは `TIMESTAMP` にも当てはまります ([セクション 5.1.8 「サーバーシステム変数」](#) を参照してください)。それ以外の場合、テーブルの最初の `TIMESTAMP` カラムのデフォルト値は現在の日付と時間になります。[セクション 11.2 「日時データ型」](#) を参照してください。
- `ENUM` ではない文字列型のデフォルト値は空の文字列です。`ENUM` のデフォルトは、最初の列挙値です。

11.7 データ型のストレージ要件

- [InnoDB テーブルの記憶域要件](#)
- [NDB テーブルのストレージ要件](#)
- [数値型の記憶域要件](#)
- [日時型の格納要件](#)
- [文字列型の格納要件](#)
- [空間型の記憶域要件](#)
- [JSON 記憶域の要件](#)

ディスク上のテーブルデータのストレージ要件は、複数の要因によって異なります。別々のストレージエンジンは異なる方法でデータ型を表し、ローデータを格納します。カラムが行全体のどちらかでテーブルデータを圧縮できますが、テーブルまたはカラムのストレージ要件の計算が複雑になります。

ディスク上のストレージレイアウトが違っていても、テーブル行に関する情報を通信および交換する内部 MySQL API は、すべてのストレージエンジンにわたって適用される一貫したデータ構造を使用します。

このセクションでは、データ型の固定サイズ表現を使用するストレージエンジンの内部形式およびサイズを含め、MySQL がサポートするデータ型ごとのストレージ要件に関するガイドラインおよび情報について説明します。情報はカテゴリまたはストレージエンジンごとに示します。

テーブルの内部表現の最大行サイズは 65,535 バイトであり、ストレージエンジンがこれ以上のサイズの行をサポートできる場合でもこのサイズになります。BLOB または TEXT カラムはこのサイズに 9 から 12 バイトしか関与しないので、これらのカラムはこのサイズに含まれません。BLOB および TEXT データについての情報は、行バッファとは異なるメモリ領域に内部的に格納されます。それぞれのストレージエンジンは、対応する型の処理に使用する方法に従って異なる方法で、このデータの割り当ておよびストレージを扱います。詳細は、[第16章「代替ストレージエンジン」](#) および [セクション8.4.7「テーブルカラム数と行サイズの制限」](#) を参照してください。

InnoDB テーブルの記憶域要件

InnoDB テーブルのストレージ要件の詳細は、[セクション15.10「InnoDB の行フォーマット」](#) を参照してください。

NDB テーブルのストレージ要件

重要

NDB テーブルは、4 バイトアライメントを使用します。すべての NDB データストレージは、4 バイトの倍数で行われます。したがって、通常であれば 15 バイトを使用するカラム値は、NDB テーブルでは 16 バイトを必要とします。たとえば、NDB テーブルでは、TINYINT、SMALLINT、MEDIUMINT、および INTEGER (INT) カラム型はそれぞれ、アライメント係数により、レコードあたり 4 バイトのストレージが必要になります。

各 BIT(M) カラムは M ビットのストレージ領域を使用します。各 BIT カラムは 4 バイトアライメントが行われていませんが、NDB は、BIT カラムに必要な最初の 1 から 32 ビットに行あたり 4 バイト (32 ビット) を、33 から 64 ビットに別の 4 ビットを、というように予約します。

NULL 自体には記憶域領域は必要ありませんが、テーブル定義に NULL を許可するカラムが含まれている場合、NDB は行ごとに最大 32 の NULL カラムを予約します。(「NDB Cluster」テーブルに 32 を超える NULL カラムが定義されている場合、行あたり 8 バイトが予約されます。)

NDB ストレージエンジンを使用するすべてのテーブルで主キーが必要になります。主キーを定義していない場合、「非表示」の主キーが NDB によって作成されます。この非表示の主キーはテーブルレコードあたり 31 から 35 バイトを消費します。

`ndb_size.pl` Perl スクリプトを使用して、NDB ストレージ要件を評価します。現在の MySQL (NDB Cluster ではない) データベースに接続し、データベースが NDB ストレージエンジンを使用した場合に必要な領域の量に関するレポートを作成します。詳細は、[セクション23.4.28「ndb_size.pl — NDBCLUSTER サイズ要件エスティメータ」](#) を参照してください。

数値型の記憶域要件

データ型	必要なストレージ
TINYINT	1 バイト
SMALLINT	2 バイト
MEDIUMINT	3 バイト
INT、INTEGER	4 バイト
BIGINT	8 バイト
FLOAT(p)	0 ≤ p ≤ 24 の場合は 4 バイト、25 ≤ p ≤ 53 の場合は 8 バイト
FLOAT	4 バイト
DOUBLE [PRECISION]、REAL	8 バイト

データ型	必要なストレージ
DECIMAL(M,D)、NUMERIC(M,D)	変動; 次の説明を参照
BIT(M)	約 $(M+7)/8$ バイト

DECIMAL (および NUMERIC) カラムの値は、9 桁の 10 進数 (10 進法) を 4 バイトにパックするバイナリ形式を使用して表現されます。各値の整数部と小数部のストレージは、個別に決定されます。9 桁の倍ごとに 4 バイトが必要であり、「余りの」桁には 4 バイトのうちの一部が必要です。余りの桁に必要なストレージ要件を次の表に示します。

余りの桁	バイト数
0	0
1	1
2	1
3	2
4	2
5	3
6	3
7	4
8	4

日時型の格納要件

TIME、DATETIME、および TIMESTAMP カラムの場合、MySQL 5.6.4 よりも前に作成されたテーブルに必要なストレージは、5.6.4 以降で作成されたテーブルとは異なります。これは、5.6.4 で、0 から 3 バイトを必要とする小数部をこれらの型が持つことを許可するように変更されたためです。

データ型	MySQL 5.6.4 より前で必要なストレージ	MySQL 5.6.4 以降で必要なストレージ
YEAR	1 バイト	1 バイト
DATE	3 バイト	3 バイト
TIME	3 バイト	3 バイト + 小数秒ストレージ
DATETIME	8 バイト	5 バイト + 小数秒ストレージ
TIMESTAMP	4 バイト	4 バイト + 小数秒ストレージ

MySQL 5.6.4 以降、YEAR および DATE のストレージは変更ありません。ただし、TIME、DATETIME、および TIMESTAMP は異なって表現されます。DATETIME はより効率的にパックされ、非小数部に必要なバイト数は 8 バイトではなく 5 バイトであり、3 つの部分すべてに、格納値の小数秒精度に応じて 0 から 3 バイトが必要な小数部があります。

小数秒精度	必要なストレージ
0	0 バイト
1、2	1 バイト
3、4	2 バイト
5、6	3 バイト

たとえば、TIME(0)、TIME(2)、TIME(4)、および TIME(6) はそれぞれ 3、4、5、6 バイトを使用します。TIME と TIME(0) は同等で、必要なストレージは同じです。

時間値の内部表現の詳細は、「[MySQL Internals: Important Algorithms and Structures](#)」を参照してください。

文字列型の格納要件

次の表では、**M** は宣言されたカラムの長さを、非バイナリ文字列型の場合は文字数で、バイナリ文字列型の場合はバイト数で表します。**L** は指定された文字列値の実際の長さをバイト数で表します。

データ型	必要なストレージ
CHAR(M)	InnoDB 行形式のコンパクトファミリは、可変長文字セットの記憶域を最適化します。 COMPACT 行形式の格納特性 を参照してください。それ以外の場合は、 $M \times w$ バイト、 $\leq M \leq 255$ 。ここで、 w は、文字セット内の最大長文字に必要なバイト数です。
BINARY(M)	M バイト、 $0 \leq M \leq 255$
VARCHAR(M)、VARBINARY(M)	カラム値が 0 から 255 バイトを必要とする場合は、 $L + 1$ バイト、値が 255 バイト以上を必要とする可能性のある場合は、 $L + 2$ バイト
TINYBLOB、TINYTEXT	$L + 1$ バイト、ここで $L < 2^8$
BLOB、TEXT	$L + 2$ バイト、ここで $L < 2^{16}$
MEDIUMBLOB、MEDIUMTEXT	$L + 3$ バイト、ここで $L < 2^{24}$
LOBLOB、LONGTEXT	$L + 4$ バイト、ここで $L < 2^{32}$
ENUM('value1','value2',...)	列挙値の数 (最大 65,535 個の値) により 1 または 2 バイト
SET('value1','value2',...)	セットメンバーの数 (最大 64 メンバー) により、1、2、3、4、または 8 バイト

可変長の文字列型は、長さプリフィクスが付いたデータを使用して格納されます。長さプリフィクスにはデータ型に応じて 1 から 4 バイトが必要で、プリフィクスの値は L (文字列のバイト長) です。たとえば、[MEDIUMTEXT](#) 値のストレージには、値を格納するための L バイトに加えて、値の長さを格納するための 3 バイトが必要です。

特定の [CHAR](#)、[VARCHAR](#)、または [TEXT](#) カラム値の格納に使用されるバイト数を計算するには、そのカラムに使用される文字セットと、値にマルチバイト文字が含まれるかどうかを考慮する必要があります。特に、[utf8](#) Unicode 文字セットを使用する場合は、すべての文字が同じバイト数を使用するわけではないことに注意する必要があります。[utf8mb3](#) および [utf8mb4](#) の文字セットには、それぞれ最大 3 バイトと 4 バイトが必要です。様々なカテゴリの [utf8mb3](#) または [utf8mb4](#) 文字に使用される記憶域の内訳は、[セクション 10.9 「Unicode のサポート」](#) を参照してください。

[VARCHAR](#)、[VARBINARY](#)、および [BLOB](#) と [TEXT](#) 型は可変長型です。それぞれのストレージ要件は次の要因によって決まります。

- カラム値の実際の長さ
- カラムの可能な最大の長さ
- カラムに使用される文字セット。一部の文字セットにはマルチバイト文字が含まれるため。

たとえば、[VARCHAR\(255\)](#) カラムには最大 255 文字の長さの文字列を格納できます。そのカラムが [latin1](#) 文字セット (1 文字あたり 1 バイト) を使用すると仮定すると、実際に必要なストレージは文字列の長さ (L) に、文字列の長さを記録するための 1 バイトを加えた大きさとなります。文字列 'abcd' の場合、 L は 4 で、ストレージ要件は 5 バイトになります。同じカラムが代わりにダブルバイト文字セット [ucs2](#) を使用するよう宣言されている場合、ストレージ要件は 10 バイトになります。'abcd' の長さは 8 バイトで、カラムの最大長が 255 よりも大きい (最大 510 バイト) ため、長さを格納するために 2 バイト必要になります。

[VARCHAR](#) または [VARBINARY](#) カラムに格納できる有効な最大バイト数は最大行サイズ (65,535 バイト、すべてのカラムで共有される) によって決まります。複数バイト文字を格納する [VARCHAR](#) カラムの場合、文字の有効な最大数は少なくなります。たとえば、[utf8mb4](#) 文字には文字ごとに最大 4 バイトが必要な場合があるため、[utf8mb4](#) 文字セットを使用する [VARCHAR](#) カラムは 16,383 文字まで宣言できます。[セクション 8.4.7 「テーブルカラム数と行サイズの制限」](#) を参照してください。

InnoDB では、768 バイト以上の固定長フィールドが可変長フィールドとしてエンコードされ、オフページに格納できます。たとえば、[utf8mb4](#) と同様に、文字セットの最大バイト長が 3 より大きい場合、[CHAR\(255\)](#) カラムは 768 バイトを超えることがあります。

NDB ストレージエンジンは可変幅カラムをサポートします。つまり、「NDB Cluster」テーブルの `VARCHAR` カラムには、ほかのストレージエンジンと同じ量のストレージが必要ですが、このような値が 4 バイトに整列されている点が異なります。したがって、`latin1` 文字セットを使用して `VARCHAR(50)` カラムに格納された文字列 'abcd' は、(MyISAM テーブル内の同じカラム値に対する 5 バイトではなく) 8 バイトを必要とします。

NDB では、`TEXT` カラムと `BLOB` カラムは異なる方法で実装されます。`TEXT` カラムの各行は、2 つの部分で構成されます。そのうちの 1 つは固定サイズ (256 バイト) で、実際に元のテーブルに格納されます。もう 1 つは 256 バイトを超えるデータで構成され、非表示のテーブルに格納されます。2 番目のテーブルの行の長さは常に 2000 バイトです。これは、`size <= 256` (`size` は行のサイズを表す) の場合、`TEXT` カラムのサイズが 256 であることを意味します。それ以外の場合、サイズは `256 + size + (2000 × (size - 256) % 2000)` です。

`ENUM` オブジェクトのサイズは異なる列挙値の数によって決まります。最大 255 の値を持つ列挙に 1 バイトが使用されます。256 から 65,535 の値を持つ列挙に 2 バイトが使用されます。[セクション 11.3.5 「ENUM 型」](#) を参照してください。

`SET` オブジェクトのサイズは異なるセットメンバーの数によって決まります。セットサイズが `N` である場合、オブジェクトは 1、2、3、4、または 8 バイトに丸められた $(N+7)/8$ バイトを占めます。`SET` は最大 64 メンバーを持つことができます。[セクション 11.3.6 「SET 型」](#) を参照してください。

空間型の記憶域要件

MySQL では、SRID の後に WKB 表現の値が続くことを示す 4 バイトを使用してジオメトリ値が格納されます。`LENGTH()` 関数は、値の格納に必要な領域をバイト単位で返します。

WKB および空間値の内部記憶域形式の詳細は、[セクション 11.4.3 「サポートされる空間データ形式」](#) を参照してください。

JSON 記憶域の要件

一般に、`JSON` カラムの記憶域要件は、`LONGBLOB` または `LONGTEXT` カラムの場合とほぼ同じです。つまり、`JSON` ドキュメントによって消費される領域は、これらのいずれかの型のカラムに格納されるドキュメント文字列表現の場合とほぼ同じです。ただし、`JSON` ドキュメントに格納されている個々の値のメタデータやディクショナリなど、バイナリエンコーディングによるオーバーヘッドがあります。たとえば、`JSON` ドキュメントに格納されている文字列には、文字列の長さおよび格納されているオブジェクトまたは配列のサイズに応じて、4 から 10 バイトの追加記憶域が必要です。

また、MySQL では、`max_allowed_packet` の値より大きくできないように、`JSON` カラムに格納される `JSON` ドキュメントのサイズに制限が課されます。

11.8 カラムに適した型の選択

最適なストレージのために、毎回もつとも正確な型を使用するよう試みる必要があります。たとえば、整数カラムを 1 から 99999 の範囲の値に使用する場合、`MEDIUMINT UNSIGNED` が最適な型になります。必要なすべての値を表す型の中で、これが、使用するストレージの容量がもっとも少ない型になります。

`DECIMAL` カラムを使用した基本的なすべての計算 (+、-、*、および /) は、65 桁 (10 進法) の精度で行われます。[セクション 11.1.1 「数値データ型の構文」](#) を参照してください。

精度がそれほど重要でない場合や、スピードが最優先事項である場合は、`DOUBLE` 型で十分と考えられます。精度を高めるために、`BIGINT` に格納されている固定小数点型にいつでも変換できます。これにより、64 ビット整数のすべての計算を行い、続いて必要に応じて結果を浮動小数点値に戻すことができます。

11.9 その他のデータベースエンジンのデータ型の使用

ほかのベンダーからの SQL 実装用に作成されたコードを使用しやすくするために、次の表に示すように、MySQL はデータ型をマップします。これらのマッピングにより、ほかのデータベースシステムから MySQL へのテーブル定義の取り込みが簡単に行えるようになります。

その他のベンダーの型	MySQL の型
<code>BOOL</code>	<code>TINYINT</code>

その他のベンダーの型	MySQL の型
BOOLEAN	TINYINT
CHARACTER VARYING(M)	VARCHAR(M)
FIXED	DECIMAL
FLOAT4	FLOAT
FLOAT8	DOUBLE
INT1	TINYINT
INT2	SMALLINT
INT3	MEDIUMINT
INT4	INT
INT8	BIGINT
LONG VARBINARY	MEDIUMBLOB
LONG VARCHAR	MEDIUMTEXT
LONG	MEDIUMTEXT
MIDDLEINT	MEDIUMINT
NUMERIC	DECIMAL

データ型のマッピングはテーブル作成時に行われ、作成後に元の型の仕様は破棄されます。ほかのベンダーで使用されている型でテーブルを作成したあとで、`DESCRIBE tbl_name` ステートメントを発行した場合、MySQL は、その型と同等の MySQL の型を使用したテーブル構造をレポートします。例:

```
mysql> CREATE TABLE t (a BOOL, b FLOAT8, c LONG VARCHAR, d NUMERIC);  
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> DESCRIBE t;  
+-----+-----+-----+-----+-----+  
| Field | Type      | Null | Key | Default | Extra |  
+-----+-----+-----+-----+-----+  
| a     | tinyint(1) | YES  |     | NULL    |      |  
| b     | double     | YES  |     | NULL    |      |  
| c     | mediantext | YES  |     | NULL    |      |  
| d     | decimal(10,0) | YES  |     | NULL    |      |  
+-----+-----+-----+-----+-----+  
4 rows in set (0.01 sec)
```


第 12 章 関数と演算子

目次

12.1 SQL 関数および演算子リファレンス	1835
12.2 ユーザー定義関数参照	1852
12.3 式評価での型変換	1856
12.4 演算子	1858
12.4.1 演算子の優先順位	1859
12.4.2 比較関数と演算子	1860
12.4.3 論理演算子	1866
12.4.4 割り当て演算子	1868
12.5 フロー制御関数	1869
12.6 数値関数と演算子	1872
12.6.1 算術演算子	1873
12.6.2 数学関数	1874
12.7 日付および時間関数	1883
12.8 文字列関数および演算子	1901
12.8.1 文字列比較関数および演算子	1914
12.8.2 正規表現	1917
12.8.3 関数結果の文字セットと照合順序	1926
12.9 MySQL で使用されるカレンダー	1927
12.10 全文検索関数	1927
12.10.1 自然言語全文検索	1929
12.10.2 ブール全文検索	1932
12.10.3 クエリー拡張を使用した全文検索	1937
12.10.4 全文ストップワード	1937
12.10.5 全文制限	1941
12.10.6 MySQL の全文検索の微調整	1942
12.10.7 全文インデックス付けのためのユーザー定義照合の追加	1945
12.10.8 ngram 全文パーサー	1946
12.10.9 MeCab フルテキストパーサープラグイン	1948
12.11 キャスト関数と演算子	1952
12.12 XML 関数	1965
12.13 ビット関数と演算子	1975
12.14 暗号化関数と圧縮関数	1985
12.15 ロック関数	1990
12.16 情報関数	1993
12.17 空間分析関数	2003
12.17.1 空間関数のリファレンス	2003
12.17.2 空間関数による引数処理	2007
12.17.3 WKT 値からジオメトリ値を作成する関数	2007
12.17.4 WKB 値からジオメトリ値を作成する関数	2009
12.17.5 ジオメトリ値を作成する MySQL 固有の関数	2011
12.17.6 ジオメトリ形式変換関数	2012
12.17.7 ジオメトリプロパティ関数	2013
12.17.8 空間演算子関数	2024
12.17.9 ジオメトリオブジェクト間の空間関係をテストする関数	2030
12.17.10 空間 Geohash 関数	2036
12.17.11 空間 GeoJSON 関数	2038
12.17.12 空間集計関数	2040
12.17.13 空間の便利な関数	2042
12.18 JSON 関数	2045
12.18.1 JSON 関数リファレンス	2046
12.18.2 JSON 値を作成する関数	2048
12.18.3 JSON 値を検索する関数	2049
12.18.4 JSON 値を変更する関数	2063

12.18.5 JSON 値属性を返す関数	2071
12.18.6 JSON テーブル関数	2074
12.18.7 JSON スキーマ検証関数	2078
12.18.8 JSON ユーティリティ関数	2083
12.19 グローバルトランザクション識別子 (GTID) で使用される機能	2088
12.20 集計関数	2090
12.20.1 集計関数の説明	2090
12.20.2 GROUP BY 修飾子	2098
12.20.3 MySQL での GROUP BY の処理	2104
12.20.4 機能依存性の検出	2107
12.21 ウィンドウ関数	2110
12.21.1 Window 関数の説明	2110
12.21.2 Window 関数の概念と構文	2116
12.21.3 ウィンドウ機能フレーム仕様	2120
12.21.4 名前付きウィンドウ	2123
12.21.5 ウィンドウ機能の制限事項	2124
12.22 パフォーマンススキーマ関数	2124
12.23 内部関数	2127
12.24 その他の関数	2128
12.25 高精度計算	2142
12.25.1 数値の型	2142
12.25.2 DECIMAL データ型の特徴	2142
12.25.3 式の処理	2143
12.25.4 丸め動作	2145
12.25.5 高精度計算の例	2146

SQL ステートメントのいくつかのポイント ([SELECT](#) ステートメントの [ORDER BY](#) または [HAVING](#) 句、[SELECT](#)、[DELETE](#)、または [UPDATE](#) ステートメントの [WHERE](#) 句、[SET](#) ステートメントなど) では、式を使用できます。式は、リテラル値、カラム値、[NULL](#)、組み込み関数、ストアドファンクション、ユーザー定義関数、および演算子を使用して作成できます。この章では、MySQL で式を記述できる SQL 関数および演算子について説明します。ストアドファンクションおよびユーザー定義関数を作成する手順については、[セクション25.2「ストアドルーチンの使用」](#) および [Adding Functions to MySQL](#) で説明されています。各種の関数への参照をサーバーが解釈する方法を記述したルールについては、[セクション9.2.5「関数名の構文解析と解決」](#) を参照してください。

[NULL](#) を含む式では、その関数または演算子に関するドキュメントで明記されていないかぎり、常に [NULL](#) 値が生成されます。

注記

デフォルトでは、関数名とそれに続く丸括弧の間には空白を入れることはできません。これは、MySQL パーサーが、関数呼び出しと、偶然に関数と同じ名前を持つテーブルまたはカラムへの参照を区別するのに役立ちます。ただし、関数の引数の前後にスペースを入れることは許可されています。

最初に `--sql-mode=IGNORE_SPACE` オプションを付けると、関数名のあとの空白文字を受け入れるように MySQL サーバーに指示できます。([セクション5.1.11「サーバー SQL モード」](#) を参照してください。) 各クライアントプログラムで `mysql_real_connect()` に `CLIENT_IGNORE_SPACE` オプションを使用すると、この動作をリクエストできます。どちらの場合でも、すべての関数名は予約語になります。

簡略化のため、この章で示すほとんどの例では、`mysql` プログラムからの出力が省略形で表示されています。例は次の書式で表示されません。

```
mysql> SELECT MOD(29,9);
+-----+
| mod(29,9) |
+-----+
|      2 |
+-----+
1 rows in set (0.00 sec)
```

代わりに、次の書式が使用されます。

```
mysql> SELECT MOD(29,9);
-> 2
```


12.1 SQL 関数および演算子リファレンス

次のテーブルに、各 SQL 関数と演算子を示し、それぞれについて簡単に説明します。ユーザー定義関数を示すテーブルについては、[セクション12.2「ユーザー定義関数参照」](#)を参照してください。

表 12.1 「SQL 関数および演算子」

名前	説明	導入	非推奨
&	ビット単位の AND		
>	右不等 (より多い) 演算子		
>>	右シフト		
>=	以上 (より多いか等しい) 演算子		
<	左不等 (より少ない) 演算子		
<>, !=	不等価 (等しくない) 演算子		
<<	左シフト		
<=	以下 (より少ないか等しい) 演算子		
<=>	NULL 安全等価演算子		
%, MOD	モジュロ演算子		
*	乗算演算子		
+	加算演算子		
-	減算演算子		
-	引数の符号を変更します		
->	パスを評価した後の JSON カラムからの戻り値 (JSON_EXTRACT() と同等)。		
->>	パスを評価して結果を引用符で囲まずに JSON カラムから値を返します (JSON_UNQUOTE(JSON_EXTRACT()) と同等)。		
/	除算演算子		
:=	値を割り当てます		
=	(SET ステートメントの一部として、または UPDATE ステートメントの SET 句の一部として) 値を割り当てます		
=	等価 (等しい) 演算子		
^	ビット単位の XOR		
ABS()	絶対値を返します		
ACOS()	アークコサインを返します		
ADDDATE()	日付値に時間値 (間隔) を加算します		
ADDTIME()	時間を加算します		
AES_DECRYPT()	AES を使用して復号化します		

名前	説明	導入	非推奨
AES_ENCRYPT()	AES を使用して暗号化します		
AND, &&	論理 AND		
ANY_VALUE()	ONLY_FULL_GROUP_BY 値の拒否を抑制します		
ASCII()	左端の文字の数値を返します		
ASIN()	アークサインを返します		
ATAN()	アークタンジェントを返します		
ATAN2(), ATAN()	2 つの引数のアークタンジェントを返します		
AVG()	引数の平均値を返します		
BENCHMARK()	式を繰り返し実行します		
BETWEEN ... AND ...	値が値の範囲内にあるかどうか		
BIN()	数値のバイナリ表現を含む文字列を返します		
BIN_TO_UUID()	バイナリ UUID を文字列に変換		
BINARY	文字列をバイナリ文字列にキャストします		
BIT_AND()	ビット単位 AND を返します		
BIT_COUNT()	設定されているビット数を返します		
BIT_LENGTH()	ビット単位で引数の長さを返します		
BIT_OR()	ビット単位の OR を返します		
BIT_XOR()	ビット単位 XOR を返します		
CAN_ACCESS_COLUMN()	内部使用のみ		
CAN_ACCESS_DATABASE()	内部使用のみ		
CAN_ACCESS_TABLE()	内部使用のみ		
CAN_ACCESS_USER()	内部使用のみ	8.0.22	
CAN_ACCESS_VIEW()	内部使用のみ		
CASE	CASE 演算子		
CAST()	値を特定の型としてキャストします		
CEIL()	引数以上のもっとも小さな整数値を返します		
CEILING()	引数以上のもっとも小さな整数値を返します		
CHAR()	渡された各整数の文字を返します		
CHAR_LENGTH()	引数の文字数を返します		

名前	説明	導入	非推奨
CHARACTER_LENGTH()	CHAR_LENGTH() のシノニムです		
CHARSET()	引数の文字セットを返します		
COALESCE()	NULL 以外の最初の引数を返します		
COERCIBILITY()	文字列引数の照合順序強制性値を返します		
COLLATION()	文字列引数の照合順序を返します		
COMPRESS()	バイナリ文字列として結果を返します		
CONCAT()	連結された文字列を返します		
CONCAT_WS()	連結されたものをセパレータ付きで返します		
CONNECTION_ID()	接続のための接続 ID (スレッド ID) を返します		
CONV()	数値を異なる基数間で変換します		
CONVERT()	値を特定の型としてキャストします		
CONVERT_TZ()	タイムゾーン間での変換		
COS()	コサインを返します		
COT()	コタンジェントを返します		
COUNT()	返された行数のカウントを返します		
COUNT(DISTINCT)	異なる値のカウントを返します		
CRC32()	巡回冗長検査値を計算します		
CUME_DIST()	累積分布値		
CURDATE()	現在の日付を返します		
CURRENT_DATE(), CURRENT_DATE	CURDATE() のシノニムです		
CURRENT_ROLE()	現在アクティブなロールを返します		
CURRENT_TIME(), CURRENT_TIME	CURTIME() のシノニムです		
CURRENT_TIMESTAMP(), CURRENT_TIMESTAMP	NOW() のシノニムです		
CURRENT_USER(), CURRENT_USER	認証済みユーザー名とホスト名		
CURTIME()	現在の時間を返します		
DATABASE()	デフォルト (現在) のデータベース名を返します		

名前	説明	導入	非推奨
DATE()	日付または日付時間式の日付部分を抽出します		
DATE_ADD()	日付値に時間値 (間隔) を加算します		
DATE_FORMAT()	日付を指定された書式に設定します		
DATE_SUB()	日付から時間値 (間隔) を引きます		
DATEDIFF()	2 つの日付の差を求めます		
DAY()	DAYOFMONTH() のシノニムです		
DAYNAME()	曜日の名前を返します		
DAYOFMONTH()	月の日を返します (0 - 31)		
DAYOFWEEK()	引数の曜日インデックスを返します		
DAYOFYEAR()	年の日を返します (1 - 366)		
DEFAULT()	テーブルカラムのデフォルト値を返します		
DEGREES()	ラジアンを角度に変換します		
DENSE_RANK()	パーティション内の現在の行のランク (ギャップなし)		
DIV	整数除算		
ELT()	インデックス番号位置の文字列を返します		
EXP()	累乗します		
EXPORT_SET()	値 bits 内の各ビットが設定されている場合は on 文字列を取得し、各ビットが設定されていない場合には off 文字列を取得するように、文字列を返します		
EXTRACT()	日付の一部を抽出します		
ExtractValue()	XPath 表記法を使用した XML 文字列からの値の抽出		
FIELD()	後続の引数の最初の引数のインデックス (位置)		
FIND_IN_SET()	2 番目の引数内の最初の引数のインデックス (位置)		
FIRST_VALUE()	ウィンドウフレームの最初の行からの引数の値		
FLOOR()	引数以下のもっとも大きな整数値を返します		
FORMAT()	指定された小数点以下桁数に書式設定された数値を返します		
FORMAT_BYTES()	バイト数を単位付きの値に変換	8.0.16	

名前	説明	導入	非推奨
FORMAT_PICO_TIME()	時間をピコ秒単位で値に変換	8.0.16	
FOUND_ROWS()	LIMIT 句付き SELECT で、LIMIT 句がない場合に戻される可能性がある行の数です		
FROM_BASE64()	base64 でエンコードされた文字列をデコードして結果を返す		
FROM_DAYS()	日数を日付に変換します		
FROM_UNIXTIME()	Unix タイムスタンプを日付として書式設定		
GeomCollection()	ジオメトリからジオメトリコレクションを構築します		
GeometryCollection()	ジオメトリからジオメトリコレクションを構築します		
GET_DD_COLUMN_PRIVILEGE()	内部使用のみ		
GET_DD_CREATE_OPTIONS()	内部使用のみ		
GET_DD_INDEX_SUB_PARTS()	内部使用のみ		
GET_FORMAT()	日付書式文字列を返します		
GET_LOCK()	名前付きロックを取得します		
GREATEST()	最大の引数を返します		
GROUP_CONCAT()	連結された文字列を返します		
GROUPING()	ROLLUP 行と通常の行の区別		
GTID_SUBSET()	サブセット内のすべての GTID がセット内にもある場合は、true を返します。そうでない場合は、false を返します。		
GTID_SUBTRACT()	セット内の GTID のうち、サブセット内にないものをすべてを返します。		
HEX()	小数または文字列値の 16 進数表現		
HOUR()	時を抽出します		
ICU_VERSION()	ICU ライブラリバージョン		
IF()	If/else 構文		
IFNULL()	Null if/else 構文		
IN()	値が値セット内にあるかどうか		
INET_ATON()	IP アドレスの数値を返します		
INET_NTOA()	数値から IP アドレスを返します		

名前	説明	導入	非推奨
INET6_ATON()	IPv6 アドレスの数値を返します		
INET6_NTOA()	数値から IPv6 アドレスを返します		
INSERT()	指定した位置に指定した文字数まで部分文字列を挿入		
INSTR()	部分文字列が最初に出現する位置のインデックスを返します		
INTERNAL_AUTO_INCREMENT()	内部使用のみ		
INTERNAL_AVG_ROW_LENGTH()	内部使用のみ		
INTERNAL_CHECK_TIME()	内部使用のみ		
INTERNAL_CHECKSUM()	内部使用のみ		
INTERNAL_DATA_FREE()	内部使用のみ		
INTERNAL_DATA_LENGTH()	内部使用のみ		
INTERNAL_DD_CHAR_LENGTH()	内部使用のみ		
INTERNAL_GET_COMMENT()	内部使用のみ		
INTERNAL_GET_ENABLED()	内部使用のみ	8.0.19	
INTERNAL_GET_HOSTNAME()	内部使用のみ	8.0.19	
INTERNAL_GET_USERNAME()	内部使用のみ	8.0.19	
INTERNAL_GET_VIEW_WARNINGS()	内部使用のみ		
INTERNAL_INDEX_COLUMN_NAME()	内部使用のみ		
INTERNAL_INDEX_LENGTH()	内部使用のみ		
INTERNAL_IS_ENABLED_ROW()	内部使用のみ	8.0.19	
INTERNAL_IS_MANDATORY_ROW()	内部使用のみ	8.0.19	
INTERNAL_KEYS_DISABLED()	内部使用のみ		
INTERNAL_MAX_DATA_LENGTH()	内部使用のみ		
INTERNAL_TABLE_ROWS()	内部使用のみ		
INTERNAL_UPDATE_TIME()	内部使用のみ		
INTERVAL()	第 1 引数より小さい引数のインデックスを返します		
IS	ブーリアンに対して値をテストします		
IS_FREE_LOCK()	名前付きロックが解放されているかどうか		
IS_IPV4()	引数が IPv4 アドレスかどうか		
IS_IPV4_COMPAT()	引数が IPv4 互換アドレスかどうか		
IS_IPV4_MAPPED()	引数が IPv4 マップアドレスかどうか		
IS_IPV6()	引数が IPv6 アドレスかどうか		
IS NOT	ブーリアンに対して値をテストします		

名前	説明	導入	非推奨
IS NOT NULL	NOT NULL 値テスト		
IS NULL	NULL 値テスト		
IS_USED_LOCK()	指定されたロックが使用中かどうか。true の場合は接続識別子を返します		
IS_UUID()	引数が有効な UUID かどうか		
ISNULL()	引数が NULL かどうかをテストします		
JSON_ARRAY()	JSON 配列の作成		
JSON_ARRAY_APPEND()	JSON ドキュメントへのデータの追加		
JSON_ARRAY_INSERT()	JSON 配列に挿入		
JSON_ARRAYAGG()	結果セットを単一の JSON 配列として返します		
JSON_CONTAINS()	JSON ドキュメントのパスに特定のオブジェクトが含まれているかどうか		
JSON_CONTAINS_PATH()	JSON ドキュメントにパスのデータが含まれているかどうか		
JSON_DEPTH()	JSON ドキュメントの最大深度		
JSON_EXTRACT()	JSON ドキュメントからデータを返します		
JSON_INSERT()	JSON ドキュメントへのデータの挿入		
JSON_KEYS()	JSON ドキュメントからのキーの配列		
JSON_LENGTH()	JSON ドキュメント内の要素数		
JSON_MERGE()	JSON ドキュメントをマージし、重複するキーを保持します。 JSON_MERGE_PRESERVE() の非推奨シノニム		はい
JSON_MERGE_PATCH()	重複キーの値を置換して JSON ドキュメントをマージ		
JSON_MERGE_PRESERVE()	重複キーを保持した JSON ドキュメントのマージ		
JSON_OBJECT()	JSON オブジェクトの作成		
JSON_OBJECTAGG()	結果セットを単一の JSON オブジェクトとして返します		
JSON_OVERLAPS()	2 つの JSON ドキュメントを比較し、共通のキーと値のペアまたは配列要素がある場合は TRUE (1) を	8.0.17	

名前	説明	導入	非推奨
	戻し、それ以外の場合は FALSE (0) を戻します		
JSON_PRETTY()	JSON ドキュメントを人間が読める形式で出力		
JSON_QUOTE()	見積 JSON 文書		
JSON_REMOVE()	JSON ドキュメントからのデータの削除		
JSON_REPLACE()	JSON ドキュメントの値の置換		
JSON_SCHEMA_VALID()	JSON スキーマに対して JSON ドキュメントを検証します。ドキュメントがスキーマに対して検証される場合は TRUE/1 を返し、検証されない場合は FALSE/0 を返します	8.0.17	
JSON_SCHEMA_VALIDATION_ERROR()	JSON スキーマに対して JSON ドキュメントを検証します。成功や失敗、失敗の理由など、検証の結果に関するレポートを JSON 形式で返します	8.0.17	
JSON_SEARCH()	JSON ドキュメント内の値へのパス		
JSON_SET()	JSON ドキュメントへのデータの挿入		
JSON_STORAGE_FREE()	部分更新後の JSON カラム値のバイナリ表現内の空き領域		
JSON_STORAGE_SIZE()	JSON ドキュメントのバイナリ表現の格納に使用される領域		
JSON_TABLE()	JSON 式からリレーショナルテーブルとしてデータを返します		
JSON_TYPE()	JSON 値のタイプ		
JSON_UNQUOTE()	JSON 値の引用符なし		
JSON_VALID()	JSON 値が有効かどうか		
JSON_VALUE()	指定されたパスでポイントされた場所にある JSON ドキュメントから値を抽出します。この値を VARCHAR(512) または指定されたタイプとして返します	8.0.21	
LAG()	パーティション内の現在行より遅れている行の引数の値		
LAST_DAY	引数の月の最終日を返します		

名前	説明	導入	非推奨
LAST_INSERT_ID()	前回の INSERT での AUTOINCREMENT カラムの値です		
LAST_VALUE()	ウィンドウフレームの最後の行からの引数の値		
LCASE()	LOWER() のシノニムです		
LEAD()	パーティション内の現在の行の先頭行からの引数の値		
LEAST()	最小の引数を返します		
LEFT()	左端から指定された数の文字を返します		
LENGTH()	文字列の長さをバイト単位で返します		
LIKE	単純なパターン一致		
LineString()	Point 値から LineString を構築します		
LN()	引数の自然対数を返します		
LOAD_FILE()	指定されたファイルをロードします		
LOCALTIME(), LOCALTIME	NOW() のシノニムです		
LOCALTIMESTAMP, LOCALTIMESTAMP()	NOW() のシノニムです		
LOCATE()	部分文字列が最初に出現する位置を返します		
LOG()	最初の引数の自然対数を返します		
LOG10()	引数の底 10 の対数を返します		
LOG2()	引数の底 2 の対数を返します		
LOWER()	引数を小文字で返します		
LPAD()	指定された文字列で左からパディングした文字列引数を返します		
LTRIM()	先頭の空白を削除します		
MAKE_SET()	bits セット内の対応するビットを持つ、カンマ区切り文字列のセットを返します		
MAKEDATE()	年と年間通算日から日付を作成します		
MAKETIME()	時、分、秒から時間を作成します		
MASTER_POS_WAIT()	レプリカが指定された位置までのすべての更新を読み取り、適用するまでブロック		
MATCH	全文検索を実行します		

名前	説明	導入	非推奨
MAX()	最大値を返します		
MBRContains()	あるジオメトリの MBR に、別のジオメトリの MBR が含まれているかどうか		
MBRCoveredBy()	ある MBR が別の MBR でカバーされているかどうか		
MBRCovers()	ある MBR が別の MBR をカバーしているかどうか		
MBRDisjoint()	2 つのジオメトリの MBR が切り離されているかどうか		
MBREquals()	2 つのジオメトリの MBR が等しいかどうか		
MBRIntersects()	2 つのジオメトリの MBR が交差しているかどうか		
MBROverlaps()	2 つのジオメトリの MBR がオーバーラップしているかどうか		
MBRTouches()	2 つのジオメトリの MBR が接しているかどうか		
MBRWithin()	あるジオメトリの MBR が、別のジオメトリの MBR の内部にあるかどうか		
MD5()	MD5 チェックサムを計算します		
MEMBER OF()	最初のオペランドが 2 番目のオペランドとして渡された JSON 配列のいずれかの要素と一致する場合は true (1) を返し、それ以外の場合は false (0) を返します	8.0.17	
MICROSECOND()	引数からマイクロ秒を返します		
MID()	指定された位置から始まる部分文字列を返します		
MIN()	最小値を返します		
MINUTE()	引数から分を返します		
MOD()	余りを返します		
MONTH()	渡された日付から月を返します		
MONTHNAME()	月の名前を返します		
MultiLineString()	LineString 値から MultiLineString を構築します		
MultiPoint()	Point 値から MultiPoint を構築します		
MultiPolygon()	Polygon 値から MultiPolygon を構築します		
NAME_CONST()	カラムの名前を指定		

名前	説明	導入	非推奨
NOT, !	値を否定します		
NOT BETWEEN ... AND ...	値が値の範囲内にあるかどうか		
NOT IN()	値が値セット内にあるかどうか		
NOT LIKE	単純なパターン一致の否定		
NOT REGEXP	REGEXP の否定		
NOW()	現在の日付と時間を返します		
NTH_VALUE()	ウィンドウフレームの N 番目の行からの引数の値		
NTILE()	パーティション内の現在の行のバケット番号。		
NULLIF()	expr1 = expr2 の場合に NULL を返します		
OCT()	数値の 8 進数表現を含む文字列を返します		
OCTET_LENGTH()	LENGTH() のシノニムです		
OR,	論理 OR		
ORD()	引数の左端の文字の文字コードを返します		
PERCENT_RANK()	パーセントランク値		
PERIOD_ADD()	年月に期間を加算します		
PERIOD_DIFF()	期間内の月数を返します		
PI()	pi の値を返します		
Point()	座標から Point を構築します		
Polygon()	LineString 引数から Polygon を構築します		
POSITION()	LOCATE() のシノニムです		
POW()	指定した指数で累乗された引数を返します		
POWER()	指定した指数で累乗された引数を返します		
PS_CURRENT_THREAD_ID()	現在のスレッドのパフォーマンススキーマスレッド ID	8.0.16	
PS_THREAD_ID()	指定されたスレッドのパフォーマンススキーマスレッド ID	8.0.16	
QUARTER()	日付引数から四半期を返します		
QUOTE()	SQL ステートメント内で使用するために引数をエスケープします		
RADIANS()	ラジアンに変換された引数を返します		

名前	説明	導入	非推奨
RAND()	ランダムな浮動小数点値を返します		
RANDOM_BYTES()	ランダムなバイトベクトルを返します		
RANK()	パーティション内の現在の行のランク (ギャップあり)		
REGEXP	文字列が正規表現と一致するかどうか		
REGEXP_INSTR()	正規表現に一致する部分文字列の開始インデックス		
REGEXP_LIKE()	文字列が正規表現と一致するかどうか		
REGEXP_REPLACE()	正規表現に一致する部分文字列の置換		
REGEXP_SUBSTR()	正規表現に一致する部分文字列を返します		
RELEASE_ALL_LOCKS()	現在の名前付きロックをすべて解放		
RELEASE_LOCK()	指定したロックを解放		
REPEAT()	文字列を指定された回数だけ繰り返します		
REPLACE()	指定された文字列の出現箇所を置き換えます		
REVERSE()	文字列内の文字を逆順に並べ替えます		
RIGHT()	右端から指定された数の文字を返します		
RLIKE	文字列が正規表現と一致するかどうか		
ROLES_GRAPHML()	メモリーロールのサブグラフを表す GraphML ドキュメントを返します		
ROUND()	引数を丸めます		
ROW_COUNT()	更新された行数		
ROW_NUMBER()	パーティション内の現在の行数		
RPAD()	指定された回数だけ文字列を追加します		
RTRIM()	末尾の空白を削除します		
SCHEMA()	DATABASE() のシノニムです		
SEC_TO_TIME()	秒を hh:mm:ss 形式に変換		
SECOND()	秒 (0-59) を返します		
SESSION_USER()	USER() のシノニムです		
SHA1(), SHA()	SHA-1 160 ビットチェックサムを計算します		

名前	説明	導入	非推奨
SHA2()	SHA-2 チェックサムを計算します		
SIGN()	引数の符号を返します		
SIN()	引数のサインを返します		
SLEEP()	ある秒数間スリープ状態にします		
SOUNDEX()	soundex 文字列を返します		
SOUNDS LIKE	音声を比較します		
SPACE()	指定された数の空白で構成される文字列を返します		
SQRT()	引数の平方根を返します		
ST_Area()	Polygon または MultiPolygon 領域を返します		
ST_AsBinary(), ST_AsWKB()	内部ジオメトリ形式から WKB に変換します		
ST_AsGeoJSON()	ジオメトリから GeoJSON オブジェクトを生成します		
ST_AsText(), ST_AsWKT()	内部ジオメトリ形式から WKT に変換します		
ST_Buffer()	ジオメトリから指定された距離内にある点のジオメトリを返します		
ST_Buffer_Strategy()	ST_Buffer() の作成戦略オプション		
ST_Centroid()	重心を Point として返します		
ST_Collect()	空間値をコレクションに集約	8.0.24	
ST_Contains()	あるジオメトリに別のジオメトリが含まれているかどうか		
ST_ConvexHull()	ジオメトリの凸包を返します		
ST_Crosses()	あるジオメトリが別のジオメトリと交差しているかどうか		
ST_Difference()	2 つのジオメトリの点集合の差集合を返します		
ST_Dimension()	ジオメトリの次元		
ST_Disjoint()	あるジオメトリが別のジオメトリから切り離されているかどうか		
ST_Distance()	あるジオメトリの別のジオメトリからの距離		
ST_Distance_Sphere()	2 つのジオメトリ間の地球上の最小距離		
ST_EndPoint()	LineString の終点		

名前	説明	導入	非推奨
ST_Envelope()	ジオメトリの MBR を返します		
ST_Equals()	あるジオメトリが別のジオメトリに等しいかどうか		
ST_ExteriorRing()	Polygon の外側のリングを返します		
ST_FrechetDistance()	ジオメトリ間の離散フレシェ距離	8.0.23	
ST_GeoHash()	geohash 値を生成します		
ST_GeomCollFromText(), ST_GeometryCollectionFromText(), ST_GeomCollFromTxt()	WKT からジオメトリコレクションを返します		
ST_GeomCollFromWKB(), ST_GeometryCollectionFromWKB()	WKB からジオメトリコレクションを返します		
ST_GeometryN()	ジオメトリコレクションから N 番目のジオメトリを返します		
ST_GeometryType()	ジオメトリ型の名前を返します		
ST_GeomFromGeoJSON()	GeoJSON オブジェクトからジオメトリを生成します		
ST_GeomFromText(), ST_GeometryFromText()	WKT からジオメトリを返します		
ST_GeomFromWKB(), ST_GeometryFromWKB()	WKB からジオメトリを返します		
ST_HausdorffDistance()	ジオメトリ間の離散ハウスドルフ距離	8.0.23	
ST_InteriorRingN()	Polygon の N 番目の内側のリングを返します		
ST_Intersection()	2 つのジオメトリの点集合の積集合を返します		
ST_Intersects()	あるジオメトリが別のジオメトリと交差しているかどうか		
ST_IsClosed()	ジオメトリが閉じていて単純かどうか		
ST_IsEmpty()	ジオメトリが空かどうか		
ST_IsSimple()	ジオメトリが単純かどうか		
ST_IsValid()	ジオメトリが有効かどうか		
ST_LatFromGeoHash()	geohash 値から緯度を返します		
ST_Latitude()	点の緯度を返します	8.0.12	
ST_Length()	LineString の長さを返します		
ST_LineFromText(), ST_LineStringFromText()	WKT から LineString を構築します		
ST_LineFromWKB(), ST_LineStringFromWKB()	WKB から LineString を構築します		

名前	説明	導入	非推奨
ST_LineInterpolatePoint()	LineString に沿った特定のパーセンテージのポイント	8.0.24	
ST_LineInterpolatePoints()	LineString に沿って指定された割合をポイント	8.0.24	
ST_LongFromGeoHash()	geohash 値から経度を返します		
ST_Longitude()	点の経度を返します	8.0.12	
ST_MakeEnvelope()	2 点周りの長方形		
ST_MLineFromText(), ST_MultiLineStringFromText()	WKT から MultiLineString を構築します		
ST_MLineFromWKB(), ST_MultiLineStringFromWKB()	WKB から MultiLineString を構築します		
ST_MPointFromText(), ST_MultiPointFromText()	WKT から MultiPoint を構築します		
ST_MPointFromWKB(), ST_MultiPointFromWKB()	WKB から MultiPoint を構築します		
ST_MPolyFromText(), ST_MultiPolygonFromText()	WKT から MultiPolygon を構築します		
ST_MPolyFromWKB(), ST_MultiPolygonFromWKB()	WKB から MultiPolygon を構築します		
ST_NumGeometries()	ジオメトリコレクション内のジオメトリ数を返します		
ST_NumInteriorRing(), ST_NumInteriorRings()	Polygon 内の内側のリング数を返します		
ST_NumPoints()	LineString 内の Point の数を返します		
ST_Overlaps()	あるジオメトリが別のジオメトリとオーバーラップしているかどうか		
ST_PointAtDistance()	LineString に沿った特定の距離の点	8.0.24	
ST_PointFromGeoHash()	geohash 値を POINT 値に変換します		
ST_PointFromText()	WKT から Point を構築します		
ST_PointFromWKB()	WKB から Point を構築します		
ST_PointN()	LineString から N 番目の Point を返します		
ST_PolyFromText(), ST_PolygonFromText()	WKT から Polygon を構築します		
ST_PolyFromWKB(), ST_PolygonFromWKB()	WKB から Polygon を構築します		
ST_Simplify()	簡略化されたジオメトリを返す		
ST_SRID()	ジオメトリの空間参照システム ID を返します		
ST_StartPoint()	LineString の始点		

名前	説明	導入	非推奨
ST_SwapXY()	X/Y 座標が入れ替えられた引数を返します		
ST_SymDifference()	2 つのジオメトリの点集合の対称差を返します		
ST_Touches()	あるジオメトリが別のジオメトリに接しているかどうか		
ST_Transform()	ジオメトリの座標を変換	8.0.13	
ST_Union()	2 つのジオメトリの点集合の和集合を返します		
ST_Validate()	検証済ジオメトリを戻します		
ST_Within()	あるジオメトリが別のジオメトリの内部にあるかどうか		
ST_X()	Point の X 座標を返します		
ST_Y()	Point の Y 座標を返します		
STATEMENT_DIGEST()	ステートメントダイジェストハッシュ値の計算		
STATEMENT_DIGEST_TEXT()	正規化されたステートメントダイジェストの計算		
STD()	母標準偏差を返します		
STDDEV()	母標準偏差を返します		
STDDEV_POP()	母標準偏差を返します		
STDDEV_SAMP()	標本標準偏差を返します		
STR_TO_DATE()	文字列を日付に変換します		
STRCMP()	2 つの文字列を比較します		
SUBDATE()	3 つの引数で呼び出される場合は DATE_SUB() のシノニムです		
SUBSTR()	指定された部分文字列を返します		
SUBSTRING()	指定された部分文字列を返します		
SUBSTRING_INDEX()	文字列から、区切り文字が指定された回数出現する前の部分文字列を返します		
SUBTIME()	時間の差を求めます		
SUM()	集計を返します		
SYSDATE()	この関数が実行される時間を返します		
SYSTEM_USER()	USER() のシノニムです		
TAN()	引数のタンジェントを返します		
TIME()	渡された式の時部分を抽出します		

名前	説明	導入	非推奨
TIME_FORMAT()	時間として書式設定します		
TIME_TO_SEC()	秒に変換された引数を返します		
TIMEDIFF()	時間の差を求めます		
TIMESTAMP()	引数が 1 つの場合、この関数は日付または日付時間式を返します。引数が 2 つの場合、引数の合計を返します		
TIMESTAMPADD()	日付時間式に間隔を加算します		
TIMESTAMPDIFF()	日付時間式から間隔を減算します		
TO_BASE64()	base 64 文字列に変換された引数を返します		
TO_DAYS()	日に変換された日付引数を返します		
TO_SECONDS()	0 年以降の秒数に変換された日付または日付時間引数を返します		
TRIM()	先頭と末尾にある空白を削除します		
TRUNCATE()	指定された小数点以下の桁数に切り捨てます		
UCASE()	UPPER() のシノニムです		
UNCOMPRESS()	圧縮された文字列を圧縮解除します		
UNCOMPRESSED_LENGTH()	圧縮前の文字列長を返します		
UNHEX()	数値の 16 進数表現を含む文字列を返します		
UNIX_TIMESTAMP()	Unix タイムスタンプを返します		
UpdateXML()	置換後 XML フラグメントを返します		
UPPER()	大文字に変換します		
USER()	ユーザー名と、クライアントによって提供されるホスト名です		
UTC_DATE()	現在の UTC 日付を返します		
UTC_TIME()	現在の UTC 時間を返します		
UTC_TIMESTAMP()	現在の UTC 日付と時間を返します		
UUID()	ユニバーサル固有識別子 (UUID) を返します		
UUID_SHORT()	整数値のユニバーサル識別子を返します		

名前	説明	導入	非推奨
UUID_TO_BIN()	文字列 UUID をバイナリに変換		
VALIDATE_PASSWORD_STRENGTH()	PASSWORD() の強度を判断します		
VALUES()	INSERT 中に使用する値の定義		
VAR_POP()	母標準分散を返します		
VAR_SAMP()	標本分散を返します		
VARIANCE()	母標準分散を返します		
VERSION()	MySQL サーバーのバージョンを示す文字列を返します		
WAIT_FOR_EXECUTED_GTID_BEFORE_TIMER()	指定された GTID がレプリカで実行されるまで待機します。		
WAIT_UNTIL_SQL_THREAD_WAIT_BEFORE_BEGINNING()	WAIT_UNTIL_SQL_THREAD_WAIT_BEFORE_BEGINNING() の使用。		8.0.18
WEEK()	週番号を返します		
WEEKDAY()	曜日インデックスを返します		
WEEKOFYEAR()	日付の暦週を返します (1 - 53)		
WEIGHT_STRING()	文字列の重み文字列を返します		
XOR	論理 XOR		
YEAR()	年を返します		
YEARWEEK()	年と週を返します		
 	ビット単位の OR		
~	ビット単位の反転		

12.2 ユーザー定義関数参照

次のテーブルに、各ユーザー定義関数を示し、それぞれについて簡単に説明します。SQL 関数および演算子を示すテーブルは、[セクション12.1「SQL 関数および演算子リファレンス」](#)を参照してください

ユーザー定義関数の一般情報は、[セクション5.7「MySQL Server のユーザー定義関数」](#)を参照してください。

表 12.2 「ユーザー定義関数」

名前	説明	導入	非推奨
asymmetric_decrypt()	秘密鍵または公開鍵を使用して暗号文を復号化します		
asymmetric_derive()	非対称鍵から対称鍵を導出します		
asymmetric_encrypt()	秘密キーまたは公開キーを使用したクリアテキストの暗号化		
asymmetric_sign()	ダイジェストから署名を生成します		

名前	説明	導入	非推奨
asymmetric_verify()	署名がダイジェストと一致することを確認します		
asynchronous_connection_failover_from_magoo()	管理対象グループのレプリケーションソースサーバーをソースリストに追加	8.0.23	
asynchronous_connection_failover_from_source()	レプリケーションソースサーバーをソースリストに追加	8.0.22	
asynchronous_connection_failover_to_magoo()	ソースリストからのレプリケーションサーバーの管理対象グループの削除	8.0.23	
asynchronous_connection_failover_to_source()	ソースリストからのレプリケーションソースサーバーの削除	8.0.22	
audit_api_message_emit_udf()	監査ログへのメッセージイベントの追加		
audit_log_encryption_password_get()	監査ログ暗号化パスワードのフェッチ		
audit_log_encryption_password_set()	監査ログの暗号化パスワードの設定		
audit_log_filter_flush()	監査ログフィルタテーブルのフラッシュ		
audit_log_filter_remove_filter()	監査ログフィルタの削除		
audit_log_filter_remove_user()	ユーザーからの監査ログフィルタの割当て解除		
audit_log_filter_set_filter()	監査ログフィルタの定義		
audit_log_filter_set_user()	ユーザーへの監査ログフィルタの割当て		
audit_log_read()	監査ログレコードを返します		
audit_log_read_bookmark()	最新の監査ログイベントのブックマーク		
create_asymmetric_priv_key()	秘密鍵を作成します		
create_asymmetric_pub_key()	公開鍵を作成します		
create_dh_parameters()	共有 DH シークレットを生成します		
create_digest()	文字列からダイジェストを生成します		
firewall_group_delist()	ファイアウォールグループプロファイルからアカウントを削除	8.0.23	
firewall_group_enlist()	ファイアウォールグループプロファイルにアカウントを追加	8.0.23	
gen_blacklist()	辞書用語置換の実行		8.0.23
gen_blocklist()	辞書用語置換の実行	8.0.23	
gen_dictionary()	辞書からランダムな用語を返します		

名前	説明	導入	非推奨
<code>gen_dictionary_drop()</code>	レジストリからディクショナリを削除		
<code>gen_dictionary_load()</code>	ディクショナリをレジストリにロード		
<code>gen_range()</code>	範囲内の乱数を生成		
<code>gen_rnd_email()</code>	ランダムな電子メールアドレスの生成		
<code>gen_rnd_pan()</code>	ランダム支払カードプライマリアカウント番号の生成		
<code>gen_rnd_ssn()</code>	ランダム US 社会保障番号の生成		
<code>gen_rnd_us_phone()</code>	ランダムな米国電話番号の生成		
<code>group_replication_get_communication_protocol_version()</code>	Group Replication) プロトコルバージョンを返します		
<code>group_replication_get_write_concurrency_max_parallel_workers_executing()</code>	グループで実行可能なコンセンサスインスタンスの最大数を返します		
<code>group_replication_set_as_primary_member()</code>	グループメンバーを新規プライマリとして割り当てます		
<code>group_replication_set_communication_protocol_version()</code>	グループレプリケーションプロトコルバージョンの設定		
<code>group_replication_set_write_concurrency_max_parallel_workers_executing()</code>	コンセンサスインスタンスの最大実行可能数をパラレルに設定		
<code>group_replication_switch_to_multi_primary_mode()</code>	単一プライマリモードからマルチプライマリモードへのグループの変更		
<code>group_replication_switch_to_single_primary_mode()</code>	マルチプライマリモードからシングルプライマリモードへのグループの変更		
<code>keyring_aws_rotate_cmek()</code>	AWS 顧客マスターキーのローテーション		
<code>keyring_aws_rotate_keys()</code>	keyring_aws ストレージファイル内のキーのローテーション		
<code>keyring_hashicorp_update_configuration()</code>	ランタイム keyring_hashicorp 再構成の原因		
<code>keyring_key_fetch()</code>	キーリングキー値のフェッチ		
<code>keyring_key_generate()</code>	ランダムキーリングキーの生成		
<code>keyring_key_length_fetch()</code>	キーリング鍵の長さを返します		
<code>keyring_key_remove()</code>	キーリングキーの削除		
<code>keyring_key_store()</code>	キーリングにキーを格納		

名前	説明	導入	非推奨
keyring_key_type_fetch()	キーリングキータイプを返します		
load_rewrite_rules()	リライタプラグインヘルパールーチン		
mask_inner()	文字列の内部部分のマスク		
mask_outer()	文字列の左右の部分のマスク		
mask_pan()	文字列の支払カードプライマリアカウント番号部分のマスク		
mask_pan_relaxed()	文字列の支払カードプライマリアカウント番号部分のマスク		
mask_ssn()	US 社会保障番号のマスク		
mysql_firewall_flush_status()	ファイアウォールステータス変数のリセット		
mysql_query_attribute_string()	クエリー属性値のフェッチ	8.0.23	
normalize_statement()	ダイジェスト形式への SQL ステートメントの正規化		
read_firewall_group_allowlist()	ファイアウォールグループプロファイルの記録されたステートメントキャッシュの更新	8.0.23	
read_firewall_groups()	ファイアウォールグループプロファイルキャッシュの更新	8.0.23	
read_firewall_users()	ファイアウォールアカウントプロファイルキャッシュの更新		
read_firewall_whitelist()	ファイアウォールアカウントプロファイルの記録ステートメントキャッシュの更新		
service_get_read_locks()	ロックサービス共有ロックの取得		
service_get_write_locks()	ロックサービス排他ロックの取得		
service_release_locks()	ロックサービスロックの解放		
set_firewall_group_mode()	ファイアウォールグループプロファイルの動作モードを確立	8.0.23	
set_firewall_mode()	ファイアウォールアカウントプロファイル操作モードの確立		
version_tokens_delete()	バージョントークンリストからのトークンの削除		
version_tokens_edit()	バージョントークンリストの変更		

名前	説明	導入	非推奨
<code>version_tokens_lock_exclusive()</code>	バージョントークンの排他ロックの取得		
<code>version_tokens_lock_shared()</code>	バージョントークンの共有ロックの取得		
<code>version_tokens_set()</code>	バージョントークンリストの設定		
<code>version_tokens_show()</code>	バージョントークンリストを返します		
<code>version_tokens_unlock()</code>	バージョントークンロックの解放		

12.3 式評価での型変換

演算子が別の型のオペランドとともに使用されると、オペランドの互換性を保つために型変換が発生します。一部の型変換は暗黙的に発生します。たとえば、MySQL は必要に応じて文字列を数値に自動的に変換し、その逆も行いません。

```
mysql> SELECT 1+1';
-> 2
mysql> SELECT CONCAT(2,' test');
-> '2 test'
```

また、`CAST()` 関数を明示的に使用して、数字を文字列に変換することもできます。`CONCAT()` 関数では文字列の引数が要求されるため、使用すると暗黙的に変換が発生します。

```
mysql> SELECT 38.8, CAST(38.8 AS CHAR);
-> 38.8, '38.8'
mysql> SELECT 38.8, CONCAT(38.8);
-> 38.8, '38.8'
```

文字セットの数字から文字列への暗黙的な変換について、および `CREATE TABLE ... SELECT` ステートメントに適用される変更済みのルールについては、このセクションの後半を参照してください。

次のルールでは、比較演算の際にどのように変換が発生するのかについて説明します。

- `NULL-safe <=>` 等価比較演算子の場合を除いて、一方または両方の引数が `NULL` の場合は、比較の結果も `NULL` になります。`NULL <=> NULL` の場合は、結果が `true` になります。変換は必要ありません。
- 比較演算の両方の引数が文字列の場合は、文字列として比較されます。
- 両方の引数が整数の場合は、整数として比較されます。
- 16 進値が数字と比較されない場合は、バイナリ文字列として処理されます。
- いずれかの引数が `TIMESTAMP` カラムまたは `DATETIME` カラムで、もう一方の引数が定数の場合、定数は比較が実行される前にタイムスタンプに変換されます。これは、ODBC により適合させるために実行されます。これは、`IN()` への引数には実行されません。念のため、比較を行う際は、常に完全な日付時間、日付、または時間文字列を使用してください。たとえば、日付または時間の値とともに `BETWEEN` を使用したときの結果を最適にするには、`CAST()` を使用して、明示的に値を目的のデータ型に変換します。

テーブル (複数可) からの単一行のサブクエリーは、定数とみなされません。たとえば、サブクエリーで `DATETIME` 値と比較される整数が返される場合は、比較が 2 つの整数として実行されます。整数は時間値には変換されません。オペランドを `DATETIME` 値として比較するには、`CAST()` を使用して、明示的にサブクエリーの値を `DATETIME` に変換します。

- 引数のいずれかが 10 進値の場合、比較はその他の引数に依存します。その他の引数が 10 進値または整数値の場合、引数は 10 進値として比較され、その他の引数が浮動小数点値の場合、引数は浮動小数点値として比較されません。
- ほかのすべてのケースでは、引数は浮動小数点 (実) 数として比較されます。たとえば、文字列オペランドと数値オペランドの比較は、浮動小数点数の比較として行われます。

別の時間型への値の変換については、[セクション11.2.7「日付と時間型間での変換」](#)を参照してください。

JSON 値の比較は 2 つのレベルで行われます。最初のレベルの比較は、比較された値の JSON 型に基づきます。タイプが異なる場合、比較結果は優先順位の高いタイプによってのみ決定されます。2 つの値が同じ JSON 型である場合、2 つ目のレベルの比較は型固有のルールを使用して行われます。JSON 値と非 JSON 値を比較するために、非 JSON 値は JSON に変換され、値は JSON 値として比較されます。詳細は、[JSON 値の比較および順序付け](#)を参照してください。

次の例は、比較演算での文字列から数字への変換を示しています。

```
mysql> SELECT 1 > '6x';
-> 0
mysql> SELECT 7 > '6x';
-> 1
mysql> SELECT 0 > 'x6';
-> 0
mysql> SELECT 0 = 'x6';
-> 1
```

文字列カラムと数字との比較では、MySQL はカラム上のインデックスを使用して、値をすばやく検索できません。`str_col` がインデックスの付いた文字列カラムである場合は、次のステートメントで検索を実行するときに、そのインデックスを使用できません。

```
SELECT * FROM tbl_name WHERE str_col=1;
```

その理由は、`'1'`、`' 1'`、`'1a'` のように、値 `1` に変換できるさまざまな文字列があるためです。

整数は比較前に倍精度浮動小数点に変換されるため、浮動小数点数と `INTEGER` 型の大きい値との比較は概算されます。これは、すべての 64 ビット整数を正確に表現できないためです。たとえば、整数値 $2^{53} + 1$ は float として表現できず、プラットフォームに応じて float 比較の前に 2^{53} または $2^{53} + 2$ に丸められます。

説明するために、次の比較のうち最初の比較では等しい値のみが比較されますが、両方の比較で true (1) が返されます:

```
mysql> SELECT '9223372036854775807' = 9223372036854775807;
-> 1
mysql> SELECT '9223372036854775807' = 9223372036854775806;
-> 1
```

文字列から浮動小数点への変換および整数から浮動小数点への変換が行われる場合、必ずしも同じように行われるとはかぎりません。整数は、CPU によって浮動小数点に変換される可能性があります。一方、文字列は、浮動小数点の乗算を伴う演算で 1 桁ずつ変換されます。また、結果は、コンピュータアーキテクチャ、コンパイラのバージョン、最適化レベルなどの要因の影響を受ける可能性があります。このような問題を回避する方法の 1 つは、値が暗黙的に浮動小数点値に変換されないように、`CAST()` を使用することです。

```
mysql> SELECT CAST('9223372036854775807' AS UNSIGNED) = 9223372036854775806;
-> 0
```

浮動小数点の比較についての詳細は、[セクションB.3.4.8「浮動小数点値に関する問題」](#)を参照してください。

サーバーには、文字列または `DECIMAL` の値と概算値 (`FLOAT/DOUBLE`) の数値との間の変換を改善するための基準を提供する変換ライブラリである `dtoa` が含まれています:

- Unix と Windows 間の変換の相違などが除去された、プラットフォーム間で整合性のある変換結果。
- 以前の結果に十分な精度がなかった場合 (IEEE の制限に近い値など) の正確な値の表示。
- 最大限の精度を持つ文字列書式への数字の変換。 `dtoa` の精度は、常に標準の C ライブラリ関数の精度と同じか、それ以上です。

このライブラリで生成された変換は、`dtoa` 以外の結果とは異なる場合があるため、以前の結果に依存するアプリケーションとの互換性が保たれない可能性があります。たとえば、以前の変換からの特定の正確な結果に依存するアプリケーションでは、追加の精度に対応するように調整が必要となる場合があります。

`dtoa` ライブラリでは、次のプロパティーを使用した変換が提供されます。 `D` は `DECIMAL` または文字列表現を含む値を表し、 `F` はネイティブバイナリ (IEEE) 書式の浮動小数点数を表します。

演算子

- $F \rightarrow D$ の変換は、最大限の精度で実行され、読み取り時に F が生成されるもっとも短い文字列として D が返され、IEEE で指定されているネイティブバイナリ形式でもっとも近い値に丸められます。
- $D \rightarrow F$ の変換は、 F が入力された 10 進文字列 D にもっとも近いネイティブバイナリの数字になるように実行されます。

F が $-\text{inf}$ 、 $+\text{inf}$ 、または NaN の場合を除いて、これらのプロパティーは、 $F \rightarrow D \rightarrow F$ の変換が可逆であることを暗黙的に示しています。後者の値は、SQL 標準では FLOAT または DOUBLE の無効な値として定義されているため、サポートされていません。

$D \rightarrow F \rightarrow D$ の変換では、 D が 15 桁以下の精度を使用し、非正規値 ($-\text{inf}$ 、 $+\text{inf}$ 、または NaN) でないことが可逆のための十分な条件となります。 D の精度が 15 桁よりも大きい場合でも、変換が可逆であるケースもありますが、常に該当するとはかぎりません。

数値または時間値を文字列に暗黙的に変換すると、`character_set_connection` および `collation_connection` システム変数によって決定された文字セットと照合順序を持つ値が生成されます。(一般に、これらの変数は `SET NAMES` を使用して設定されます。接続文字セットについては、[セクション 10.4 「接続文字セットおよび照合順序」](#) を参照してください。)

つまり、接続文字セットが `binary` に設定されている場合を除いて、このような変換では、(非バイナリ) 文字列 (`CHAR`、`VARCHAR`、または `LONGTEXT` 値) が生成されます。この場合、変換の結果はバイナリ文字列 (`BINARY`、`VARBINARY`、または `LONGBLOB` 値) になります。

整数式では、式の評価に関する上記の備考は、式の割り当てには多少異なる方法で適用されます。次のようなステートメントでの例を示します。

```
CREATE TABLE t SELECT integer_expr;
```

この場合、式の結果として生成されるカラム内のテーブルの型は、整数式の長さに応じて、`INT` または `BIGINT` になります。式の最大長が `INT` に収まらない場合は、代わりに `BIGINT` が使用されます。長さは、`SELECT` 結果セットメタデータの `max_length` 値から取得されます ([C API Basic Data Structures](#) を参照してください)。つまり、十分に長い式を使用することで、`INT` ではなく、`BIGINT` を強制的に適用できます。

```
CREATE TABLE t SELECT 00000000000000000000000000;
```

12.4 演算子

表 12.3 「演算子」

名前	説明	導入
&	ビット単位の AND	
>	右不等 (より多い) 演算子	
>>	右シフト	
>=	以上 (より多いか等しい) 演算子	
<	左不等 (より少ない) 演算子	
<>, !=	不等価 (等しくない) 演算子	
<<	左シフト	
<=	以下 (より少ないか等しい) 演算子	
<=	NULL 安全等価演算子	
%, MOD	モジュロ演算子	
*	乗算演算子	
+	加算演算子	
-	減算演算子	
-	引数の符号を変更します	
->	パスを評価した後の JSON カラムからの戻り値 (<code>JSON_EXTRACT()</code> と同等)。	

名前	説明	導入
->>	パスを評価して結果を引用符で囲ま ずに JSON カラムから値を返します (JSON_UNQUOTE(JSON_EXTRACT() と同等)。	
/	除算演算子	
:=	値を割り当てます	
=	(SET ステートメントの一部として、 または UPDATE ステートメントの SET 句の一部として) 値を割り当てま す	
=	等価 (等しい) 演算子	
^	ビット単位の XOR	
AND, &&	論理 AND	
BETWEEN ... AND ...	値が値の範囲内にあるかどうか	
BINARY	文字列をバイナリ文字列にキャストし ます	
CASE	CASE 演算子	
DIV	整数除算	
IN()	値が値セット内にあるかどうか	
IS	ブーリアンに対して値をテストします	
IS NOT	ブーリアンに対して値をテストします	
IS NOT NULL	NOT NULL 値テスト	
IS NULL	NULL 値テスト	
LIKE	単純なパターン一致	
MEMBER OF()	最初のオペランドが 2 番目のオペラ ンドとして渡された JSON 配列のい ずれかの要素と一致する場合は true (1) を返し、それ以外の場合は false (0) を返します	8.0.17
NOT, !	値を否定します	
NOT BETWEEN ... AND ...	値が値の範囲内でないかどうか	
NOT IN()	値が値セット内でないかどうか	
NOT LIKE	単純なパターン一致の否定	
NOT REGEXP	REGEXP の否定	
OR,	論理 OR	
REGEXP	文字列が正規表現と一致するかどうか	
RLIKE	文字列が正規表現と一致するかどうか	
SOUNDS LIKE	音声を比較します	
XOR	論理 XOR	
	ビット単位の OR	
~	ビット単位の反転	

12.4.1 演算子の優先順位

次のリストには、演算子の優先順位をもっとも高いものから順番に示しています。同じ行に並んで記載されている演算子は、優先順位が同じものです。

```

INTERVAL
BINARY, COLLATE
!
- (unary minus), ~ (unary bit inversion)
^
*, /, DIV, %, MOD
-, +
<<, >>
&
|
= (comparison), <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN, MEMBER OF
BETWEEN, CASE, WHEN, THEN, ELSE
NOT
AND, &&
XOR
OR, ||
= (assignment), :=

```

= の優先順位は、比較演算子 (=) として使用されるのか、割り当て演算子 (:=) として使用されるのかによって異なります。比較演算子として使用する場合、<=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP および IN() と同じ優先順位を持ちます。割り当て演算子として使用される場合は、優先順位が := と同じです。セクション13.7.6.1「変数代入の SET 構文」およびセクション9.4「ユーザー定義変数」では、適用される = の解釈が MySQL でどのように決定されるのかについて説明されています。

式内で同じ優先順位レベルで発生する演算子の場合、評価は左から右に進みますが、割当てが右から左に評価される点が異なります。

一部の演算子の優先順位と意味は、SQL モードによって異なります:

- デフォルトでは、|| は論理 OR 演算子です。PIPES_AS_CONCAT が有効になっている場合は、|| は ^ と単項演算子間の優先順位を持つ文字列連結です。
- デフォルトでは、! は NOT よりも高い優先順位です。HIGH_NOT_PRECEDENCE が有効になっている場合は、! と NOT の優先順位は同じです。

セクション5.1.11「サーバー SQL モード」を参照してください。

演算子の優先順位によって、式の項の評価順序が決まります。この順序をオーバーライドし、明示的に項をグループ化するには、丸括弧を使用します。例:

```

mysql> SELECT 1+2*3;
-> 7
mysql> SELECT (1+2)*3;
-> 9

```

12.4.2 比較関数と演算子

表 12.4 「比較演算子」

名前	説明
>	右不等 (より多い) 演算子
>=	以上 (より多いか等しい) 演算子
<	左不等 (より少ない) 演算子
<>, !=	不等価 (等しくない) 演算子
<=	以下 (より少ないか等しい) 演算子
<=>	NULL 安全等価演算子
=	等価 (等しい) 演算子
BETWEEN ... AND ...	値が値の範囲内にあるかどうか
COALESCE()	NULL 以外の最初の引数を返します
GREATEST()	最大の引数を返します
IN()	値が値セット内にあるかどうか

名前	説明
INTERVAL()	第 1 引数より小さい引数のインデックスを返します
IS	ブーリアンに対して値をテストします
IS NOT	ブーリアンに対して値をテストします
IS NOT NULL	NOT NULL 値テスト
IS NULL	NULL 値テスト
ISNULL()	引数が NULL かどうかをテストします
LEAST()	最小の引数を返します
LIKE	単純なパターン一致
NOT BETWEEN ... AND ...	値が値の範囲内にはないかどうか
NOT IN()	値が値セット内にはないかどうか
NOT LIKE	単純なパターン一致の否定
STRCMP()	2 つの文字列を比較します

比較演算の結果は、1 (TRUE)、0 (FALSE)、または NULL の値になります。これらの演算は、数字と文字列の両方で機能します。必要に応じて、文字列は数字に、数字は文字列に自動的に変換されます。

次の関係比較演算子を使用すれば、スカラーオペランドだけでなく行オペランドも比較できます。

```
= > < >= <= <> !=
```

これらの演算子の詳細は、このセクションの後半で説明します。行サブクエリーのコンテキストにおける行比較のその他の例は、[セクション13.2.11.5「行サブクエリー」](#)を参照してください。

このセクションで示す関数の一部では、1 (TRUE)、0 (FALSE)、または NULL 以外の値が返されます。[LEAST\(\)](#) および [GREATEST\(\)](#) はこのような関数の例です。[セクション12.3「式評価での型変換」](#)では、これらの関数と同様の関数によって実行される、戻り値を決定するための比較操作のルールについて説明しています。

注記

以前のバージョンの MySQL では、[LEAST\(\)](#) または [GREATEST\(\)](#) を含む式を評価するときに、サーバーは関数が使用されたコンテキストを推測し、関数の引数を式全体のデータ型に強制変換しようとしていました。たとえば、[LEAST\("11", "45", "2"\)](#) の引数は文字列として評価およびソートされるため、この式は"11"を返します。MySQL 8.0.3 以前では、式 [LEAST\("11", "45", "2"\) + 0](#) を評価する際に、サーバーは引数をソートする前に整数に変換し(結果に整数 0 を追加することが予想されます)、2 を返します。

MySQL 8.0.4 以降、サーバーはこの方法でコンテキストの推測を試行しなくなりました。かわりに、関数は指定された引数を使用して実行され、すべて同じ型でない場合にのみ、1 つ以上の引数へのデータ型変換が実行されます。戻り値を使用する式によって強制される型強制は、関数の実行後に実行されるようになりました。つまり、MySQL 8.0.4 以降では、[LEAST\("11", "45", "2"\) + 0](#) は"11" + 0 と評価されるため、整数 11 になります。(Bug #83895、Bug #25123839)

[CAST\(\)](#) 関数を使用すると、比較目的で値を特定の型に変換できます。[CONVERT\(\)](#) を使用すると、文字列値を別の文字セットに変換できます。[セクション12.11「キャスト関数と演算子」](#)を参照してください。

デフォルトでは、文字列比較では大/小文字は区別されず、現在の文字セットが使用されます。デフォルトは [utf8mb4](#) です。

• =

等しい:

```
mysql> SELECT 1 = 0;
-> 0
mysql> SELECT '0' = 0;
-> 1
```

```
mysql> SELECT '0.0' = 0;
-> 1
mysql> SELECT '0.01' = 0;
-> 0
mysql> SELECT '.01' = 0.01;
-> 1
```

行の比較では、 $(a, b) = (x, y)$ は次と同等です:

```
(a = x) AND (b = y)
```

- **<=>**

NULL - 安全等価。この演算子では、**=** 演算子のように等価比較が実行されますが、両方のオペランドが **NULL** であれば、**NULL** でなく **1** が返され、一方のオペランドが **NULL** の場合は、**NULL** でなく **0** が返されます。

<=> 演算子は、標準の SQL **IS NOT DISTINCT FROM** 演算子と同等です。

```
mysql> SELECT 1 <=> 1, NULL <=> NULL, 1 <=> NULL;
-> 1, 1, 0
mysql> SELECT 1 = 1, NULL = NULL, 1 = NULL;
-> 1, NULL, NULL
```

行の比較では、 $(a, b) <=> (x, y)$ は次と同等です:

```
(a <=> x) AND (b <=> y)
```

- **<>, !=**

等しくない:

```
mysql> SELECT '.01' <> '0.01';
-> 1
mysql> SELECT .01 <> '0.01';
-> 0
mysql> SELECT 'zapp' <> 'zapp';
-> 1
```

行の比較では、 $(a, b) <> (x, y)$ と $(a, b) != (x, y)$ は次と同等です:

```
(a <> x) OR (b <> y)
```

- **<=**

より少ないか等しい:

```
mysql> SELECT 0.1 <= 2;
-> 1
```

行の比較では、 $(a, b) <= (x, y)$ は次と同等です:

```
(a < x) OR ((a = x) AND (b <= y))
```

- **<**

より少ない:

```
mysql> SELECT 2 < 2;
-> 0
```

行の比較では、 $(a, b) < (x, y)$ は次と同等です:

```
(a < x) OR ((a = x) AND (b < y))
```

- **>=**

より多いか等しい:

```
mysql> SELECT 2 >= 2;
-> 1
```

行の比較では、 $(a, b) \geq (x, y)$ は次と同等です:

```
(a > x) OR ((a = x) AND (b >= y))
```

- >

より多い:

```
mysql> SELECT 2 > 2;  
-> 0
```

行の比較では、 $(a, b) > (x, y)$ は次と同等です:

```
(a > x) OR ((a = x) AND (b > y))
```

- `expr BETWEEN min AND max`

`expr` が `min` より多いか等しく、`expr` が `max` より少ないか等しい場合、`BETWEEN` は 1 を返し、それ以外では 0 を返します。すべての引数の型が同じであれば、これは式 $(\min \leq \text{expr} \text{ AND } \text{expr} \leq \max)$ と同等です。それ以外の場合は、[セクション12.3「式評価での型変換」](#)に記載したルールに従って型変換が実行されますが、3つのすべての引数に適用されます。

```
mysql> SELECT 2 BETWEEN 1 AND 3, 2 BETWEEN 3 and 1;  
-> 1, 0  
mysql> SELECT 1 BETWEEN 2 AND 3;  
-> 0  
mysql> SELECT 'b' BETWEEN 'a' AND 'c';  
-> 1  
mysql> SELECT 2 BETWEEN 2 AND '3';  
-> 1  
mysql> SELECT 2 BETWEEN 2 AND 'x-3';  
-> 0
```

日付または時間の値とともに `BETWEEN` を使用したときの結果を最適にするには、`CAST()` を使用して明示的に値を目的のデータ型に変換します。例: `DATETIME` を2つの `DATE` 値と比較する場合は、`DATE` 値を `DATETIME` 値に変換します。`DATE` との比較で '2001-1-1' などの文字列定数を使用する場合は、文字列を `DATE` にキャストします。

- `expr NOT BETWEEN min AND max`

これは、`NOT (expr BETWEEN min AND max)` と同じです。

- `COALESCE(value,...)`

リストの最初の非 `NULL` 値を返します。非 `NULL` 値がない場合は、`NULL` を返します。

`COALESCE()` の戻り型は、引数型の集計型です。

```
mysql> SELECT COALESCE(NULL,1);  
-> 1  
mysql> SELECT COALESCE(NULL,NULL,NULL);  
-> NULL
```

- `GREATEST(value1,value2,...)`

2つ以上の引数がある場合は、最大の(最大値の)引数を返します。引数は、`LEAST()` のルールと同じルールを使用して比較されます。

```
mysql> SELECT GREATEST(2,0);  
-> 2  
mysql> SELECT GREATEST(34.0,3.0,5.0,767.0);  
-> 767.0  
mysql> SELECT GREATEST('B','A','C');  
-> 'C'
```

引数のいずれかが `NULL` である場合、`GREATEST()` は `NULL` を返します。

- `expr IN (value,...)`

`expr` が `IN()` リストのいずれかの値と等しい場合は `1` (true) を返し、それ以外の場合は `0` (false) を返します。

型変換は、[セクション12.3「式評価での型変換」](#) で説明されているルールに従って行われ、すべての引数に適用されます。`IN()` リストの値に型変換が必要ない場合、それらはすべて同じ型の `JSON` 以外の定数であり、`expr` を同じ型の値として比較できます (型変換後の場合もあります)。リストがソートされ、バイナリ検索を使用して `expr` の検索が実行されるため、`IN()` 操作が非常に迅速になります。

```
mysql> SELECT 2 IN (0,3,5,7);
-> 0
mysql> SELECT 'wefwf' IN ('wee','wefwf','weg');
-> 1
```

`IN()` を使用して、行コンストラクタを比較できます:

```
mysql> SELECT (3,4) IN ((1,2), (3,4));
-> 1
mysql> SELECT (3,4) IN ((1,2), (3,5));
-> 0
```

引用符で囲まれた値 (文字列など) と囲まれていない値 (数字など) の比較ルールは異なるため、`IN()` リストの引用符で囲まれた値と囲まれていない値を決して混同しないでください。したがって、型を混同すると、整合性のない結果になる可能性があります。たとえば、`IN()` 式を次のように記述しないでください。

```
SELECT val1 FROM tbl1 WHERE val1 IN (1,2,'a');
```

代わりに、次のように記述してください。

```
SELECT val1 FROM tbl1 WHERE val1 IN ('1','2','a');
```

暗黙的な型変換では、直感的でない結果が生成される場合があります:

```
mysql> SELECT 'a' IN (0), 0 IN ('b');
-> 1, 1
```

どちらの場合も、比較値は浮動小数点値に変換され、それぞれの場合に `0.0` が生成され、比較結果は `1` (true) になります。

`IN()` リストの値の数は、`max_allowed_packet` 値によってのみ制限されます。

SQL の標準に準拠するために、左側の式が `NULL` である場合だけでなく、リストに一致が見つからない場合やリストの式のいずれかが `NULL` である場合にも、`IN()` は `NULL` を返します。

`IN()` 構文は、特定のタイプのサブクエリーを作成する際にも使用できます。[セクション13.2.11.3「ANY、IN、または SOME を使用したサブクエリー」](#) を参照してください。

- `expr NOT IN (value,...)`

これは、`NOT (expr IN (value,...))` と同じです。

- `INTERVAL(N,N1,N2,N3,...)`

`N < N1` の場合は `0` を返し、`N < N2` などの場合は `1` を返し、`N` が `NULL` の場合は `-1` を返します。すべての引数は整数として処理されます。この関数が正しく機能するには、`N1 < N2 < N3 < ... < Nn` とする必要があります。これは、バイナリ検索が使用されていることが理由です (非常に高速)。

```
mysql> SELECT INTERVAL(23, 1, 15, 17, 30, 44, 200);
-> 3
mysql> SELECT INTERVAL(10, 1, 10, 100, 1000);
-> 2
mysql> SELECT INTERVAL(22, 23, 30, 44, 200);
-> 0
```

- `IS boolean_value`

`boolean_value` を `TRUE`、`FALSE`、または `UNKNOWN` にすることができるブール値に対して値をテストします。

```
mysql> SELECT 1 IS TRUE, 0 IS FALSE, NULL IS UNKNOWN;
```



```
-> 1, 1, 1
```

- **IS NOT boolean_value**

boolean_value を **TRUE**、**FALSE**、または **UNKNOWN** にすることができるブール値に対して値をテストします。

```
mysql> SELECT 1 IS NOT UNKNOWN, 0 IS NOT UNKNOWN, NULL IS NOT UNKNOWN;
-> 1, 1, 0
```

- **IS NULL**

値が **NULL** かどうかをテストします。

```
mysql> SELECT 1 IS NULL, 0 IS NULL, NULL IS NULL;
-> 0, 0, 1
```

ODBC プログラムを適切に使用するために、MySQL では、**IS NULL** の使用時に次の追加機能がサポートされています:

- **sql_auto_is_null** 変数が 1 に設定されている場合は、自動的に生成された **AUTO_INCREMENT** 値を正常に挿入するステートメントのあとに、次の形式のステートメントを発行すれば、その値を検索できます。

```
SELECT * FROM tbl_name WHERE auto_col IS NULL
```

ステートメントが行を返す場合、返される値は **LAST_INSERT_ID()** 関数を呼び出した場合と同じです。複数行の挿入後の戻り値などについての詳細は、[セクション12.16「情報関数」](#)を参照してください。**AUTO_INCREMENT** 値を正常に挿入できなかった場合、**SELECT** ステートメントは行を返しませんが、

IS NULL の比較を使用して **AUTO_INCREMENT** 値を取得する動作は、**sql_auto_is_null = 0** を設定すると無効にできます。[セクション5.1.8「サーブシステム変数」](#)を参照してください。

sql_auto_is_null のデフォルト値は 0 です。

- **NOT NULL** として宣言された **DATE** および **DATETIME** カラムでは、次のようなステートメントを使用することで、特殊な日付 '0000-00-00' を検索できます。

```
SELECT * FROM tbl_name WHERE date_column IS NULL
```

ODBC では '0000-00-00' 日付値がサポートされていないため、一部の ODBC アプリケーションを取得する際に、これが必要になります。

[Obtaining Auto-Increment Values](#)、および [Connector/ODBC Connection Parameters](#) の **FLAG_AUTO_IS_NULL** オプションについての説明を参照してください。

- **IS NOT NULL**

値が **NULL** でないかどうかをテストします。

```
mysql> SELECT 1 IS NOT NULL, 0 IS NOT NULL, NULL IS NOT NULL;
-> 1, 1, 0
```

- **ISNULL(expr)**

expr が **NULL** の場合、**ISNULL()** は 1 を返し、それ以外の場合は 0 を返します。

```
mysql> SELECT ISNULL(1+1);
-> 0
mysql> SELECT ISNULL(1/0);
-> 1
```

= の代わりに **ISNULL()** を使用すると、値が **NULL** であるかどうかをテストできます。(=**を使用して値を NULL と比較すると、常に NULL が生成されます。**)

ISNULL() 関数は **IS NULL** 比較演算子と、いくつかの特殊な動作を共有します。**IS NULL** の説明を参照してください。

- **LEAST(value1,value2,...)**

2 つ以上の引数がある場合は、最小の (最小値の) 引数を返します。引数は、次のルールを使用して比較されます。

- 引数が **NULL** である場合、結果は **NULL** になります。比較は必要ありません。
- すべての引数が整数値の場合、整数として比較されます。
- 少なくとも 1 つの引数が倍精度の場合は、倍精度の値として比較されます。それ以外の場合、少なくとも 1 つの引数が **DECIMAL** 値であれば、**DECIMAL** 値として比較されます。
- 引数が数値と文字列の組合せで構成されている場合、それらは文字列として比較されます。
- 引数が非バイナリ (文字) 文字列の場合は、非バイナリ文字列として比較されます。
- ほかのすべてのケースでは、引数はバイナリ文字列として比較されます。

LEAST() の戻り型は、比較引数型の集計型です。

```
mysql> SELECT LEAST(2,0);
-> 0
mysql> SELECT LEAST(34.0,3.0,5.0,767.0);
-> 3.0
mysql> SELECT LEAST('B','A','C');
-> 'A'
```

12.4.3 論理演算子

表 12.5 「論理演算子」

名前	説明
AND, &&	論理 AND
NOT, !	値を否定します
OR, 	論理 OR
XOR	論理 XOR

SQL では、すべての論理演算子は **TRUE**、**FALSE**、または **NULL (UNKNOWN)** に評価されます。MySQL では、これらは **1 (TRUE)**、**0 (FALSE)**、および **NULL** として実装されます。この大部分は、さまざまな SQL データベースサーバーに共通のもので、ただし、一部のサーバーは **TRUE** にゼロ以外の任意の値を返す場合があります。

MySQL では、ゼロ以外の任意の非 **NULL** 値が **TRUE** に評価されます。たとえば、次のステートメントはすべて **TRUE** に評価されます。

```
mysql> SELECT 10 IS TRUE;
-> 1
mysql> SELECT -10 IS TRUE;
-> 1
mysql> SELECT 'string' IS NOT NULL;
-> 1
```

- **NOT, !**

NOT 演算。オペランドが **0** の場合は **1** に、オペランドがゼロ以外の場合は **0** にそれぞれ評価され、**NOT NULL** の場合は **NULL** が返されます。

```
mysql> SELECT NOT 10;
-> 0
mysql> SELECT NOT 0;
-> 1
mysql> SELECT NOT NULL;
-> NULL
mysql> SELECT !(1+1);
-> 0
mysql> SELECT ! 1+1;
-> 1
```

最後の例では、式が **!(1)+1** と同様に評価されるため、**1** が生成されています。

!、演算子は非標準の MySQL 拡張機能です。MySQL 8.0.17 では、この演算子は非推奨です。将来のバージョンの MySQL で削除される予定です。アプリケーションは、標準の SQL `NOT` 演算子を使用するように調整する必要があります。

- **AND, &&**

論理 AND すべてのオペランドがゼロ以外で非 `NULL` の場合は `1` に、1 つ以上のオペランドが `0` の場合は `0` に評価され、それ以外の場合は `NULL` が返されます。

```
mysql> SELECT 1 AND 1;
-> 1
mysql> SELECT 1 AND 0;
-> 0
mysql> SELECT 1 AND NULL;
-> NULL
mysql> SELECT 0 AND NULL;
-> 0
mysql> SELECT NULL AND 0;
-> 0
```

`&&`、演算子は非標準の MySQL 拡張機能です。MySQL 8.0.17 では、この演算子は非推奨です。将来のバージョンの MySQL ではサポートされなくなる予定です。アプリケーションは、標準の SQL `AND` 演算子を使用するように調整する必要があります。

- **OR, ||**

論理 OR。両方のオペランドが非 `NULL` であれば、オペランドのいずれかがゼロ以外である場合の結果は `1`、それ以外の場合は `0` になります。 `NULL` オペランドが 1 つあれば、ほかのオペランドがゼロ以外である場合の結果は `1`、それ以外の場合は `NULL` になります。両方のオペランドが `NULL` であれば、結果は `NULL` になります。

```
mysql> SELECT 1 OR 1;
-> 1
mysql> SELECT 1 OR 0;
-> 1
mysql> SELECT 0 OR 0;
-> 0
mysql> SELECT 0 OR NULL;
-> NULL
mysql> SELECT 1 OR NULL;
-> 1
```

注記

`PIPES_AS_CONCAT` SQL モードが有効な場合、`||` は SQL 標準の文字列連結演算子 (`CONCAT()` など) を示します。

`||`、演算子は非標準の MySQL 拡張機能です。MySQL 8.0.17 では、この演算子は非推奨です。将来のバージョンの MySQL ではサポートされなくなる予定です。アプリケーションは、標準の SQL `OR` 演算子を使用するように調整する必要があります。例外: `PIPES_AS_CONCAT` が有効な場合、`||` は文字列の連結を示すため、非推奨は適用されません。

- **XOR**

論理 XOR。オペランドのいずれかが `NULL` である場合は、`NULL` を返します。非 `NULL` のオペランドでは、奇数のオペランドがゼロ以外の場合は `1` に評価され、それ以外の場合は `0` が返されます。

```
mysql> SELECT 1 XOR 1;
-> 0
mysql> SELECT 1 XOR 0;
-> 1
mysql> SELECT 1 XOR NULL;
-> NULL
mysql> SELECT 1 XOR 1 XOR 1;
-> 1
```

`a XOR b` は、数学的に `(a AND (NOT b)) OR ((NOT a) and b)` に等しくなります。

12.4.4 割り当て演算子

表 12.6 「代入演算子」

名前	説明
<code>:=</code>	値を割り当てます
<code>=</code>	(<code>SET</code> ステートメントの一部として、または <code>UPDATE</code> ステートメントの <code>SET</code> 句の一部として) 値を割り当てます

- `:=`

割り当て演算子。演算子の左側にあるユーザー変数が右側にある値に代入されます。右側の値は、リテラル値、値を格納する別の変数、またはクエリーの結果を含むスカラー値を生成する任意の有効な式 (この値がスカラー値の場合) である可能性があります。同じ `SET` ステートメントで、複数の割り当てを実行できます。同じステートメントで複数の割り当てを実行できます。

`=`とは異なり、`:=` 演算子は比較演算子として解釈されません。つまり、(`SET` ステートメントだけでなく) 有効な任意の SQL ステートメントで `:=` を使用すれば、値を変数に割り当てることができます。

```
mysql> SELECT @var1, @var2;
-> NULL, NULL
mysql> SELECT @var1 := 1, @var2;
-> 1, NULL
mysql> SELECT @var1, @var2;
-> 1, NULL
mysql> SELECT @var1, @var2 := @var1;
-> 1, 1
mysql> SELECT @var1, @var2;
-> 1, 1

mysql> SELECT @var1:=COUNT(*) FROM t1;
-> 4
mysql> SELECT @var1;
-> 4
```

次に示すように、`SELECT` 以外のステートメント (`UPDATE` など) でも、`:=` を使用して値の割り当てを実行できます。

```
mysql> SELECT @var1;
-> 4
mysql> SELECT * FROM t1;
-> 1, 3, 5, 7

mysql> UPDATE t1 SET c1 = 2 WHERE c1 = @var1:= 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT @var1;
-> 1
mysql> SELECT * FROM t1;
-> 2, 3, 5, 7
```

`:=` 演算子を使用すれば、単一の SQL ステートメントで同じ変数の値の設定と読み取りの両方を行うこともできますが、これは推奨されていません。[セクション9.4「ユーザー定義変数」](#)では、これを回避すべき理由について説明されています。

- `=`

この演算子は、次の 2 つのパラグラフで説明する 2 つのケースで値の割り当てを実行する際に使用されます。

`SET` ステートメントでは、`=` は、演算子の左側にあるユーザー変数を右側にある値に代入する割り当て演算子として処理されます。(言い換えると、`SET` ステートメントで使用されると、`=` は `:=` と同じように処理されます。) 右側の値は、リテラル値、値を格納する別の変数、またはクエリーの結果を含むスカラー値を生成する任意の有効な式 (この値がスカラー値の場合) である可能性があります。同じ `SET` ステートメントで、複数の割り当てを実行できます。

UPDATE ステートメントの SET 句では、= は割り当て演算子としても機能します。ただし、この場合、UPDATE の一部である WHERE 条件に一致していれば、演算子の左側で指定されたカラムが右側に指定された値であるとみなされます。UPDATE ステートメントの同じ SET 句で、複数の割り当てを実行できます。

その他のコンテキストでは、= は比較演算子として処理されます。

```
mysql> SELECT @var1, @var2;
-> NULL, NULL
mysql> SELECT @var1 := 1, @var2;
-> 1, NULL
mysql> SELECT @var1, @var2;
-> 1, NULL
mysql> SELECT @var1, @var2 := @var1;
-> 1, 1
mysql> SELECT @var1, @var2;
-> 1, 1
```

詳細は、[セクション13.7.6.1「変数代入の SET 構文」](#)、[セクション13.2.13「UPDATE ステートメント」](#)、および [セクション13.2.11「サブクエリー」](#) を参照してください。

12.5 フロー制御関数

表 12.7 「フロー制御演算子」

名前	説明
CASE	CASE 演算子
IF()	If/else 構文
IFNULL()	Null if/else 構文
NULLIF()	expr1 = expr2 の場合に NULL を返します

- CASE value WHEN compare_value THEN result [WHEN compare_value THEN result ...] [ELSE result] END

CASE WHEN condition THEN result [WHEN condition THEN result ...] [ELSE result] END

最初の CASE 構文では、true である最初の value=compare_value 比較の result が返されます。2 番目の構文は、true である最初の条件の結果を返します。比較または条件が true の場合、ELSE が返された後の結果、または ELSE 部分がない場合は NULL が返されます。

注記

ここで説明する CASE operator の構文は、[セクション13.6.5.1「CASE ステートメント」](#) で説明する SQL CASE statement の構文とは若干異なり、ストアプログラム内で使用されます。CASE ステートメントは ELSE NULL 句を持つことができず、END でなく、END CASE で終了します。

CASE 式の結果の戻り型は、すべての結果値の集計型です:

- すべてのタイプが数値の場合、集計タイプも数値になります。
 - 少なくとも 1 つの引数が倍精度の場合、結果は倍精度になります。
 - それ以外の場合、少なくとも 1 つの引数が DECIMAL であれば、結果は DECIMAL になります。
 - それ以外の場合、結果は整数型になります (ただし、次の例外があります):
 - すべての整数型がすべて符号付きまたは符号なしの場合、結果は同じ符号になり、精度は指定されたすべての整数型 (TINYINT, SMALLINT, MEDIUMINT, INT または BIGINT) の中で最も高くなります。
 - 符号付き整数型と符号なし整数型の組合せがある場合、結果は符号付きになり、精度が高くなる可能性があります。たとえば、型が署名付き INT および署名なし INT の場合、結果は署名付き BIGINT になります。

- 例外は、符号なし **BIGINT** と符号付き整数型を組み合わせたものです。その結果、精度とスケール 0 が十分な **DECIMAL** になります。
- すべてのタイプが **BIT** の場合、結果は **BIT** になります。それ以外の場合、**BIT** 引数は **BIGINT** と同様に扱われます。
- すべてのタイプが **YEAR** の場合、結果は **YEAR** になります。それ以外の場合、**YEAR** 引数は **INT** と同様に扱われます。
- すべての型が文字列 (**CHAR** または **VARCHAR**) の場合、結果は、オペランドの最長文字長によって決定される最大長の **VARCHAR** になります。
- すべての型が文字列またはバイナリ文字列の場合、結果は **VARBINARY** になります。
- **SET** および **ENUM** は **VARCHAR** と同様に処理され、結果は **VARCHAR** になります。
- すべてのタイプが **JSON** の場合、結果は **JSON** になります。
- すべての型が時間的な場合、結果は時間的になります:
 - すべての時間型が **DATE**、**TIME** または **TIMESTAMP** の場合、結果はそれぞれ **DATE**、**TIME** または **TIMESTAMP** になります。
 - それ以外の場合、時間型が混在すると、結果は **DATETIME** になります。
- すべてのタイプが **GEOMETRY** の場合、結果は **GEOMETRY** になります。
- いずれかのタイプが **BLOB** の場合、結果は **BLOB** になります。
- 他のすべてのタイプの組合せの場合、結果は **VARCHAR** です。
- リテラル **NULL** オペランドは、集計型では無視されます。

```
mysql> SELECT CASE 1 WHEN 1 THEN 'one'
->   WHEN 2 THEN 'two' ELSE 'more' END;
-> 'one'
mysql> SELECT CASE WHEN 1>0 THEN 'true' ELSE 'false' END;
-> 'true'
mysql> SELECT CASE BINARY 'B'
->   WHEN 'a' THEN 1 WHEN 'b' THEN 2 END;
-> NULL
```


- IF(expr1,expr2,expr3)

expr1 が TRUE (expr1 <> 0 および expr1 <> NULL) の場合、IF() は expr2 を返します。それ以外の場合は、expr3 を返します。

注記

IF ステートメントもありますが、ここで説明されている IF() 関数とは異なります。 [セクション13.6.5.2「IF ステートメント」](#) を参照してください。

expr2 と expr3 の一方のみが明示的に NULL である場合は、IF() 関数の結果型は非 NULL 式の型になります。

IF() のデフォルトの戻り型 (一時テーブルに格納されている場合でもかまいません) は、次のように計算されます:

- expr2 または expr3 が文字列を生成する場合、結果は文字列になります。

expr2 と expr3 の両方が文字列の場合、いずれかの文字列で大文字と小文字が区別されると、結果では大文字と小文字が区別されます。

- expr2 または expr3 が浮動小数点値を生成する場合、結果は浮動小数点値になります。
- expr2 または expr3 が整数を生成する場合、結果は整数になります。

```
mysql> SELECT IF(1>2,3);
-> 3
mysql> SELECT IF(1<2,'yes','no');
-> 'yes'
mysql> SELECT IF(STRCMP('test','test1'),'no','yes');
-> 'no'
```

- IFNULL(expr1,expr2)

expr1 が NULL でない場合、IFNULL() は expr1 を返し、それ以外の場合は expr2 を返します。

```
mysql> SELECT IFNULL(1,0);
-> 1
mysql> SELECT IFNULL(NULL,10);
-> 10
mysql> SELECT IFNULL(1/0,10);
-> 10
mysql> SELECT IFNULL(1/0,'yes');
-> 'yes'
```

IFNULL(expr1,expr2) のデフォルトの戻り型は、STRING、REAL または INTEGER の順で、2つの式のうちより多くの「一般」です。式や MySQL が一時テーブルの IFNULL() で返された値を内部に格納する必要がある場所に基いて、テーブルの大文字と小文字を考慮してください。

```
mysql> CREATE TABLE tmp SELECT IFNULL(1,'test') AS test;
mysql> DESCRIBE tmp;
+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| test  | varbinary(4) | NO   |    |         |       |
+-----+-----+-----+-----+-----+
```

この例では、test カラムの型は VARBINARY(4) (文字列型) です。

- NULLIF(expr1,expr2)

expr1 = expr2 が true の場合は NULL を返し、それ以外の場合は expr1 を返します。これは、CASE WHEN expr1 = expr2 THEN NULL ELSE expr1 END と同じです。

戻り値の型は最初の引数と同じです。

```
mysql> SELECT NULLIF(1,1);
-> NULL
mysql> SELECT NULLIF(1,2);
```

注記

引数が等しくない場合は、MySQL で `expr1` が 2 回評価されます。

12.6 数値関数と演算子

表 12.8 「数値関数および演算子」

名前	説明
<code>%, MOD</code>	モジュロ演算子
<code>*</code>	乗算演算子
<code>+</code>	加算演算子
<code>-</code>	減算演算子
<code>-</code>	引数の符号を変更します
<code>/</code>	除算演算子
<code>ABS()</code>	絶対値を返します
<code>ACOS()</code>	アークコサインを返します
<code>ASIN()</code>	アークサインを返します
<code>ATAN()</code>	アークタンジェントを返します
<code>ATAN2(), ATAN()</code>	2 つの引数のアークタンジェントを返します
<code>CEIL()</code>	引数以上のもっとも小さな整数値を返します
<code>CEILING()</code>	引数以上のもっとも小さな整数値を返します
<code>CONV()</code>	数値を異なる基数間で変換します
<code>COS()</code>	コサインを返します
<code>COT()</code>	コタンジェントを返します
<code>CRC32()</code>	巡回冗長検査値を計算します
<code>DEGREES()</code>	ラジアンを角度に変換します
<code>DIV</code>	整数除算
<code>EXP()</code>	累乗します
<code>FLOOR()</code>	引数以下のもっとも大きな整数値を返します
<code>LN()</code>	引数の自然対数を返します
<code>LOG()</code>	最初の引数の自然対数を返します
<code>LOG10()</code>	引数の底 10 の対数を返します
<code>LOG2()</code>	引数の底 2 の対数を返します
<code>MOD()</code>	余りを返します
<code>PI()</code>	pi の値を返します
<code>POW()</code>	指定した指数で累乗された引数を返します
<code>POWER()</code>	指定した指数で累乗された引数を返します
<code>RADIANS()</code>	ラジアンに変換された引数を返します
<code>RAND()</code>	ランダムな浮動小数点値を返します
<code>ROUND()</code>	引数を丸めます
<code>SIGN()</code>	引数の符号を返します
<code>SIN()</code>	引数のサインを返します

名前	説明
SQRT()	引数の平方根を返します
TAN()	引数のタンジェントを返します
TRUNCATE()	指定された小数点以下の桁数に切り捨てます

12.6.1 算術演算子

表 12.9 「算術演算子」

名前	説明
%, MOD	モジュロ演算子
*	乗算演算子
+	加算演算子
-	減算演算子
-	引数の符号を変更します
/	除算演算子
DIV	整数除算

通常の算術演算子を使用できます。結果は次のルールに従って決定されます。

- [-](#)、[+](#)、および [*](#) の場合は、両方のオペランドが整数であれば、結果が [BIGINT](#) (64 ビット) 精度で計算されます。
- 両方のオペランドが整数で、いずれかが符合なしの場合は、結果が符合なし整数になります。減算では、[NO_UNSIGNED_SUBTRACTION](#) SQL モードが有効になっている場合は、オペランドのいずれかが符号なしでも、結果が符号付きになります。
- [+](#)、[-](#)、[/](#)、[*](#)、[%](#) のオペランドのいずれかが実数値または文字列値である場合は、結果の精度が、最大の精度を持つオペランドの精度になります。
- [/](#) を使用して実行される除算では、2 つの正確な値のオペランドを使用したときの結果のスケールが、1 番目のオペランドに [div_precision_increment](#) システム変数 (デフォルトでは 4) の値を加えた値になります。たとえば、式 [5.05 / 0.014](#) の結果のスケールは、小数点以下 6 桁になります ([360.714286](#))。

ネストされた計算が各コンポーネントの精度を暗黙的に示すように、これらのルールは演算ごとに適用されます。したがって、[\(14620 / 9432456\) / \(24250 / 9432456\)](#) では、まず [\(0.0014\) / \(0.0026\)](#) に解かれて、最終的な結果は小数点以下 8 桁 ([0.60288653](#)) になります。

このようなルールおよびそれらが適用される方法があるため、計算のコンポーネントおよびサブコンポーネントで適切な精度レベルが使用されていることを慎重に確認してください。[セクション12.11「キャスト関数と演算子」](#)を参照してください。

数値式評価でのオーバーフロー処理については、[セクション11.1.7「範囲外およびオーバーフローの処理」](#)を参照してください。

算術演算子は数字に適用されます。その他の型の値では、代替の演算が使用できる場合もあります。たとえば、日付値を追加するには、[DATE_ADD\(\)](#) を使用します。[セクション12.7「日付および時間関数」](#)を参照してください。

- [+](#)

加算:

```
mysql> SELECT 3+5;
-> 8
```

- [-](#)

減算:

```
mysql> SELECT 3-5;
-> -2
```

-

単項マイナス。この演算子は、オペランドの符号を変更します。

```
mysql> SELECT -2;
-> -2
```

注記

この演算子が **BIGINT** で使用される場合は、戻り値も **BIGINT** になります。つまり、値 -2^{63} を持つ可能性のある整数では、**-** の使用を避けるべきです。

- *

乗算:

```
mysql> SELECT 3*5;
-> 15
mysql> SELECT 18014398509481984*18014398509481984.0;
-> 324518553658426726783156020576256.0
mysql> SELECT 18014398509481984*18014398509481984;
-> out-of-range error
```

最後の式では、整数乗算の結果が 64 ビット範囲の **BIGINT** 計算を超過するため、エラーが生成されます。(セクション11.1「数値データ型」を参照してください。)

- /

除算:

```
mysql> SELECT 3/5;
-> 0.60
```

ゼロによる除算では、**NULL** の結果が生成されます。

```
mysql> SELECT 102/(1-1);
-> NULL
```

結果が整数に変換されるコンテキストで実行される場合にのみ、除算は **BIGINT** 算術を使用して計算されます。

- DIV

整数除算。除算結果から小数点以下の小数部を破棄します。

いずれかのオペランドが整数以外の型の場合、オペランドは **DECIMAL** に変換され、**DECIMAL** 算術を使用して除算されてから、結果が **BIGINT** に変換されます。結果が **BIGINT** の範囲を超過する場合は、エラーが発生します。

```
mysql> SELECT 5 DIV 2, -5 DIV 2, 5 DIV -2, -5 DIV -2;
-> 2, -2, -2, 2
```

- N % M, N MOD M

モジュロ演算。M で除算された N の余りを返します。詳細は、セクション12.6.2「数学関数」の MOD() 関数に関する説明を参照してください。

12.6.2 数学関数

表 12.10 「数学関数」

名前	説明
ABS()	絶対値を返します
ACOS()	アークコサインを返します

名前	説明
ASIN()	アークサインを返します
ATAN()	アークタンジェントを返します
ATAN2(), ATAN()	2つの引数のアークタンジェントを返します
CEIL()	引数以上のもっとも小さな整数値を返します
CEILING()	引数以上のもっとも小さな整数値を返します
CONV()	数値を異なる基数間で変換します
COS()	コサインを返します
COT()	コタンジェントを返します
CRC32()	巡回冗長検査値を計算します
DEGREES()	ラジアンを角度に変換します
EXP()	累乗します
FLOOR()	引数以下のもっとも大きな整数値を返します
LN()	引数の自然対数を返します
LOG()	最初の引数の自然対数を返します
LOG10()	引数の底 10 の対数を返します
LOG2()	引数の底 2 の対数を返します
MOD()	余りを返します
PI()	pi の値を返します
POW()	指定した指数で累乗された引数を返します
POWER()	指定した指数で累乗された引数を返します
RADIANS()	ラジアンに変換された引数を返します
RAND()	ランダムな浮動小数点値を返します
ROUND()	引数を丸めます
SIGN()	引数の符号を返します
SIN()	引数のサインを返します
SQRT()	引数の平方根を返します
TAN()	引数のタンジェントを返します
TRUNCATE()	指定された小数点以下の桁数に切り捨てます

すべての数学関数は、エラーの発生時に `NULL` を返します。

- `ABS(X)`

`X` の絶対値、または `X` が `NULL` の場合は `NULL` を返します。

結果の型は引数の型から導出されます。これは、結果を署名付き `BIGINT` 値に格納できないため、`ABS(-9223372036854775808)` でエラーが発生することを意味します。

```
mysql> SELECT ABS(2);
-> 2
mysql> SELECT ABS(-32);
-> 32
```

この関数は、`BIGINT` 値でも安全に使用できます。

- `ACOS(X)`

`X` のアークコサイン (つまり、コサインが `X` である値) を返します。`X` が `-1` から `1` までの範囲内でない場合は、`NULL` を返します。

```
mysql> SELECT ACOS(1);
-> 0
mysql> SELECT ACOS(1.0001);
-> NULL
mysql> SELECT ACOS(0);
-> 1.5707963267949
```

- **ASIN(X)**

X のアークサイン (つまり、サインが X である値) を返します。 X が -1 から 1 までの範囲内がない場合は、**NULL** を返します。

```
mysql> SELECT ASIN(0.2);
-> 0.20135792079033
mysql> SELECT ASIN('foo');
+-----+
| ASIN('foo') |
+-----+
| 0 |
+-----+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Warning | 1292 | Truncated incorrect DOUBLE value: 'foo' |
+-----+-----+-----+
```

- **ATAN(X)**

X のアークタンジェント (つまり、タンジェントが X である値) を返します。

```
mysql> SELECT ATAN(2);
-> 1.1071487177941
mysql> SELECT ATAN(-2);
-> -1.1071487177941
```

- **ATAN(Y,X), ATAN2(Y,X)**

2 つの変数 X および Y のアークタンジェントを返します。これは、両方の引数の符号が結果の象限の判定に使用される点を除いて、 Y/X のアークタンジェントの計算と同様です。

```
mysql> SELECT ATAN(-2,2);
-> -0.78539816339745
mysql> SELECT ATAN2(PI(),0);
-> 1.5707963267949
```

- **CEIL(X)**

CEIL() は **CEILING()** のシノニムです。

- **CEILING(X)**

X 以上で最小の整数値を返します。

```
mysql> SELECT CEILING(1.23);
-> 2
mysql> SELECT CEILING(-1.23);
-> -1
```

引数が厳密値数値の場合は、戻り値の型も厳密値数値になります。引数が文字列または浮動小数点の場合は、戻り値の型が浮動小数点になります。

- **CONV(N,from_base,to_base)**

数値を異なる基数間で変換します。基数 **from_base** から基数 **to_base** に変換された数値 N の文字列表現を返します。引数のいずれかが **NULL** である場合は、**NULL** を返します。引数 N は整数として解釈されますが、整数また

は文字列として指定される場合もあります。最小の基数は 2 で、最大の基数は 36 です。from_base が負数の場合、N は符号付き数値とみなされます。それ以外の場合は、N は符号なしとみなされます。CONV() は 64 ビット精度で動作します。

```
mysql> SELECT CONV('a',16,2);
-> '1010'
mysql> SELECT CONV('6E',18,8);
-> '172'
mysql> SELECT CONV(-17,10,-18);
-> '-H'
mysql> SELECT CONV(10+'10'+10+'X'0a',10,10);
-> '40'
```

- COS(X)

X のコサインを返します。X はラジアンで指定されます。

```
mysql> SELECT COS(PI());
-> -1
```

- COT(X)

X のコタンジェントを返します。

```
mysql> SELECT COT(12);
-> -1.5726734063977
mysql> SELECT COT(0);
-> out-of-range error
```

- CRC32(expr)

巡回冗長検査値を計算し、32 ビット値の符号なし値を返します。引数が NULL である場合は、結果も NULL になります。引数は文字列であると想定され、(可能な場合は) 文字列でない場合でも文字列として処理されます。

```
mysql> SELECT CRC32('MySQL');
-> 3259397556
mysql> SELECT CRC32('mysql');
-> 2501908538
```

- DEGREES(X)

ラジアンからディグリーに変換された引数 X を返します。

```
mysql> SELECT DEGREES(PI());
-> 180
mysql> SELECT DEGREES(PI() / 2);
-> 90
```

- EXP(X)

e (自然対数の底) の X 乗の値を返します。この関数の逆は、(単一の引数のみを使用する) LOG() または LN() です。

```
mysql> SELECT EXP(2);
-> 7.3890560989307
mysql> SELECT EXP(-2);
-> 0.13533528323661
mysql> SELECT EXP(0);
-> 1
```

- FLOOR(X)

X 以下で最大の整数値を返します。

```
mysql> SELECT FLOOR(1.23), FLOOR(-1.23);
-> 1, -2
```

引数が厳密値数値の場合は、戻り値の型も厳密値数値になります。引数が文字列または浮動小数点の場合は、戻り値の型が浮動小数点になります。

- [FORMAT\(X,D\)](#)

数値 X を '#,###,###.##' のような書式に変換し、小数点第 D 位に丸めて、その結果を文字列として返します。詳細は、[セクション12.8「文字列関数および演算子」](#) を参照してください。

- [HEX\(N_or_S\)](#)

この関数を使用すると、10 進数または文字列の 16 進表現を取得できます。その方法は、引数の型によって異なります。詳細は、[セクション12.8「文字列関数および演算子」](#) で、この関数の説明を参照してください。

- [LN\(X\)](#)

X の自然対数 (つまり、 X の底 e の対数) を返します。 X が 0.0 E0 以下の場合、この関数は `NULL` を返し、「対数の引数が無効です」という警告が報告されます。

```
mysql> SELECT LN(2);
-> 0.69314718055995
mysql> SELECT LN(-2);
-> NULL
```

この関数は [LOG\(X\)](#) のシノニムです。この関数の逆は、[EXP\(\)](#) 関数です。

- [LOG\(X\), LOG\(B,X\)](#)

1 つのパラメータで呼び出される場合、この関数は X の自然対数を返します。 X が 0.0 E0 以下の場合、この関数は `NULL` を返し、「対数の引数が無効です」という警告が報告されます。

この関数 (単一の引数で呼び出された場合) の逆は、[EXP\(\)](#) 関数です。

```
mysql> SELECT LOG(2);
-> 0.69314718055995
mysql> SELECT LOG(-2);
-> NULL
```

この関数が 2 つのパラメータで呼び出される場合は、 B を底とする X の対数が返されます。 X が 0 以下である場合、または B が 1 以下である場合は、`NULL` が返されます。

```
mysql> SELECT LOG(2,65536);
-> 16
mysql> SELECT LOG(10,100);
-> 2
mysql> SELECT LOG(1,100);
-> NULL
```

[LOG\(B,X\)](#) は [LOG\(X\) / LOG\(B\)](#) と同等です。

- [LOG2\(X\)](#)

X の底 2 の対数を返します。 X が 0.0 E0 以下の場合、この関数は `NULL` を返し、「対数の引数が無効です」という警告が報告されます。

```
mysql> SELECT LOG2(65536);
-> 16
mysql> SELECT LOG2(-100);
-> NULL
```

[LOG2\(\)](#) は、格納に必要なビット数を調べる際に役立ちます。この関数は式 [LOG\(X\) / LOG\(2\)](#) と同等です。

- [LOG10\(X\)](#)

X の底 10 の対数を返します。 X が 0.0 E0 以下の場合、この関数は `NULL` を返し、「対数の引数が無効です」という警告が報告されます。

```
mysql> SELECT LOG10(2);
-> 0.30102999566398
mysql> SELECT LOG10(100);
-> 2
mysql> SELECT LOG10(-100);
-> NULL
```

$\text{LOG}_{10}(X)$ は $\text{LOG}(10,X)$ と同等です。

- $\text{MOD}(N,M)$, $N \% M$, $N \text{ MOD } M$

モジュロ演算。 M で除算された N の余りを返します。

```
mysql> SELECT MOD(234, 10);
-> 4
mysql> SELECT 253 % 7;
-> 1
mysql> SELECT MOD(29,9);
-> 2
mysql> SELECT 29 MOD 9;
-> 2
```

この関数は、**BIGINT** 値でも安全に使用できます。

$\text{MOD}()$ は、小数部を持つ値でも機能し、除算後の正確な余りを返します。

```
mysql> SELECT MOD(34.5,3);
-> 1.5
```

$\text{MOD}(N,0)$ は **NULL** を返します。

- $\text{PI}()$

π (pi) の値を返します。表示されるデフォルトの小数点以下の桁数は 7 ですが、MySQL では内部的に全倍精度値が使用されます。

```
mysql> SELECT PI();
-> 3.141593
mysql> SELECT PI()+0.00000000000000000000;
-> 3.141592653589793116
```

- $\text{POW}(X,Y)$

X の Y 乗の値を返します。

```
mysql> SELECT POW(2,2);
-> 4
mysql> SELECT POW(2,-2);
-> 0.25
```

- $\text{POWER}(X,Y)$

これは $\text{POW}()$ のシノニムです。

- $\text{RADIANS}(X)$

ディグリーからラジアンに変換された引数 X を返します。(ラジアンは 180 度になります。)

```
mysql> SELECT RADIANS(90);
-> 1.5707963267949
```

- $\text{RAND}([N])$

$0 \leq v < 1.0$ の範囲内で、ランダムな浮動小数点値 v を返します。 $i \leq R < j$ の範囲内でランダムな整数 R を取得するには、式 $\text{FLOOR}(i + \text{RAND}() * (j - i))$ を使用します。たとえば、 $7 \leq R < 12$ の範囲のランダム整数を取得するには、次のステートメントを使用します:

```
SELECT FLOOR(7 + (RAND() * 5));
```

整数引数 N が指定されている場合は、シード値として使用されます:

- 定数イニシャライザ引数を使用すると、ステートメントの準備時に、実行前にシードが一度初期化されます。
- 定数以外のイニシャライザ引数 (カラム名など) を使用すると、シードは $\text{RAND}()$ の起動ごとに値で初期化されません。

この動作の影響の1つは、等しい引数値の場合、`RAND(N)` は毎回同じ値を返すため、カラム値の繰り返し可能なシーケンスを生成することです。次の例では、`RAND(3)` によって生成される値の順序は両方とも同じです。

```
mysql> CREATE TABLE t (i INT);
Query OK, 0 rows affected (0.42 sec)

mysql> INSERT INTO t VALUES(1),(2),(3);
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> SELECT i, RAND() FROM t;
+----+-----+
| i | RAND() |
+----+-----+
| 1 | 0.61914388706828 |
| 2 | 0.93845168309142 |
| 3 | 0.83482678498591 |
+----+-----+
3 rows in set (0.00 sec)

mysql> SELECT i, RAND(3) FROM t;
+----+-----+
| i | RAND(3) |
+----+-----+
| 1 | 0.90576975597606 |
| 2 | 0.37307905813035 |
| 3 | 0.14808605345719 |
+----+-----+
3 rows in set (0.00 sec)

mysql> SELECT i, RAND() FROM t;
+----+-----+
| i | RAND() |
+----+-----+
| 1 | 0.35877890638893 |
| 2 | 0.28941420772058 |
| 3 | 0.37073435016976 |
+----+-----+
3 rows in set (0.00 sec)

mysql> SELECT i, RAND(3) FROM t;
+----+-----+
| i | RAND(3) |
+----+-----+
| 1 | 0.90576975597606 |
| 2 | 0.37307905813035 |
| 3 | 0.14808605345719 |
+----+-----+
3 rows in set (0.01 sec)
```

`WHERE` 句内の `RAND()` は、行ごと (テーブルから選択する場合) または行の組合せごと (複数テーブル結合から選択する場合) に評価されます。したがって、オプティマイザのために、`RAND()` は定数値ではなく、インデックスの最適化に使用できません。詳細は、[セクション8.2.1.20「関数コールの最適化」](#)を参照してください。

`ORDER BY` 句または `GROUP BY` 句で `RAND()` 値を含むカラムを使用すると、いずれかの句で `RAND()` 式が同じ行に対して複数回評価され、毎回異なる結果が返されるため、予期しない結果になる可能性があります。行をランダムな順序で取得することを目的としている場合は、次のようなステートメントを使用できます:

```
SELECT * FROM tbl_name ORDER BY RAND();
```

一連の行からランダムサンプルを選択するには、`ORDER BY RAND()` と `LIMIT` を組み合せます:

```
SELECT * FROM table1, table2 WHERE a=b AND c<d ORDER BY RAND() LIMIT 1000;
```

`RAND()` は、完全なランダムジェネレータとしては設計されていません。要求に応じてランダムな数字をすばやく生成する方法であり、同じ MySQL バージョンのプラットフォーム間で移植可能です。

この関数は、ステートメントベースのレプリケーションでは安全に使用できません。`binlog_format` が `STATEMENT` に設定されているときに、この関数を使用すると、警告のログが記録されます。

- `ROUND(X)`, `ROUND(X,D)`

引数 `X` を `D` 小数点に丸めます。丸めアルゴリズムは、`X` のデータ型に依存します。`D` が指定されていない場合は、デフォルトで 0 に設定されます。`D` を負の数に指定すると、値 `X` の小数点左側の `D` 桁をゼロにすることができます。`D` の最大絶対値は 30 です。30 (または -30) を超える桁は切り捨てられます。

```
mysql> SELECT ROUND(-1.23);
-> -1
mysql> SELECT ROUND(-1.58);
-> -2
mysql> SELECT ROUND(1.58);
-> 2
mysql> SELECT ROUND(1.298, 1);
-> 1.3
mysql> SELECT ROUND(1.298, 0);
-> 1
mysql> SELECT ROUND(23.298, -1);
-> 20
mysql> SELECT ROUND(12345678901234567890123456789012345, 35);
-> 0.123456789012345678901234567890
```

戻り値の型は、最初の引数と同じです (integer、double または decimal であると想定)。つまり、引数が整数の場合は、結果が整数 (小数点なし) になります。

```
mysql> SELECT ROUND(150.000,2), ROUND(150,2);
+-----+-----+
| ROUND(150.000,2) | ROUND(150,2) |
+-----+-----+
| 150.00 | 150 |
+-----+-----+
```

`ROUND()` では、第 1 引数の型に応じて次のルールが使用されます。

- 真値の数字の場合、`ROUND()` では「四捨五入」または「切り捨て (切り上げ)」ルールが使用されます。0.5 以上の小数部を持つ値は、正の場合は次の整数に切り上げられ、負の場合は次の整数に切り下げられます。(つまり、ゼロから遠い方に丸められます。) 0.5 未満の小数部を持つ値は、正の場合は次の整数に切り下げられ、負の場合は次の整数に切り上げられます。
- 近似値の数字の場合、結果は C ライブラリによって異なります。多くのシステムでは、これは `ROUND()` が「最も近い偶数に丸める」ルールを使用することを意味: 2 つの整数の間に小数部がある値は、最も近い偶数の整数に丸められます。

次の例では、正確な値の丸めと近似値の丸めの相違点を示します。

```
mysql> SELECT ROUND(2.5), ROUND(25E-1);
+-----+-----+
| ROUND(2.5) | ROUND(25E-1) |
+-----+-----+
| 3 | 2 |
+-----+-----+
```

詳細は、[セクション 12.25 「高精度計算」](#) を参照してください。

MySQL 8.0.21 以降では、`ROUND()` (および `TRUNCATE()`) によって返されるデータ型は、次に示すルールに従って決定されます:

- 最初の引数が任意の整数型の場合、戻り型は常に `BIGINT` です。
- 最初の引数が浮動小数点型または非数値型の場合、戻り型は常に `DOUBLE` です。
- 最初の引数が `DECIMAL` 値の場合、戻り型も `DECIMAL` です。

- 戻り値の型属性も最初の引数からコピーされますが、DECIMAL の場合は、2 番目の引数が定数値の場合を除きません。

小数点以下の桁数が引数の位取りより小さい場合は、結果の位取りと精度が適宜調整されます。

また、ROUND() (TRUNCATE() 関数ではありません) の場合、精度は、有効桁数を増やす端数処理に対応するために 1 箇所まで拡張されます。2 番目の引数が負の場合、戻り値の型は、対応する精度でスケールが 0 になるように調整されます。たとえば、ROUND(99.999, 2) は 100.00 を返します。最初の引数は DECIMAL(5, 3) で、戻り値の型は DECIMAL(5, 2) です。

2 番目の引数が負の場合、戻り値の型はスケール 0 および対応する精度を持ち、ROUND(99.999, -1) は 100 (DECIMAL(3, 0)) を返します。

- SIGN(X)

X が負、ゼロ、または正のいずれであるのかに応じて、引数の符号を -1、0、または 1 として返します。

```
mysql> SELECT SIGN(-32);
-> -1
mysql> SELECT SIGN(0);
-> 0
mysql> SELECT SIGN(234);
-> 1
```

- SIN(X)

X のサインを返します。X はラジアンで指定されます。

```
mysql> SELECT SIN(PI());
-> 1.2246063538224e-16
mysql> SELECT ROUND(SIN(PI()));
-> 0
```

- SQRT(X)

負ではない数字 X の平方根を返します。

```
mysql> SELECT SQRT(4);
-> 2
mysql> SELECT SQRT(20);
-> 4.4721359549996
mysql> SELECT SQRT(-16);
-> NULL
```

- TAN(X)

X のタンジェントを返します。X はラジアンで指定されます。

```
mysql> SELECT TAN(PI());
-> -1.2246063538224e-16
mysql> SELECT TAN(PI()+1);
-> 1.5574077246549
```

- TRUNCATE(X,D)

D 小数点に切り捨てて、数字 X を返します。D が 0 の場合は、結果に小数点または小数部が含まれません。D を負の数に指定すると、値 X の小数点左側の D 桁をゼロにすることができます。

```
mysql> SELECT TRUNCATE(1.223,1);
-> 1.2
mysql> SELECT TRUNCATE(1.999,1);
-> 1.9
mysql> SELECT TRUNCATE(1.999,0);
-> 1
mysql> SELECT TRUNCATE(-1.999,1);
-> -1.9
mysql> SELECT TRUNCATE(122,-2);
-> 100
mysql> SELECT TRUNCATE(10.28*100,0);
```


-> 1028

すべての数字は、ゼロ方向に丸められます。

MySQL 8.0.21 以降では、[TRUNCATE\(\)](#) によって返されるデータ型は、[ROUND\(\)](#) 関数の戻り型を決定するものと同じルールに従います。詳細は、[ROUND\(\)](#) の説明を参照してください。

12.7 日付および時間関数

このセクションでは、時間値の処理に使用できる関数について説明します。各日付日時型が持つ値の範囲、および値を指定する際の有効な書式については、[セクション11.2「日時データ型」](#)を参照してください。

表 12.11 「日付および時刻関数」

名前	説明
ADDDATE()	日付値に時間値 (間隔) を加算します
ADDTIME()	時間を加算します
CONVERT_TZ()	タイムゾーン間での変換
CURDATE()	現在の日付を返します
CURRENT_DATE() , CURRENT_DATE	CURDATE() のシノニムです
CURRENT_TIME() , CURRENT_TIME	CURTIME() のシノニムです
CURRENT_TIMESTAMP() , CURRENT_TIMESTAMP	NOW() のシノニムです
CURTIME()	現在の時間を返します
DATE()	日付または日付時間式の日付部分を抽出します
DATE_ADD()	日付値に時間値 (間隔) を加算します
DATE_FORMAT()	日付を指定された書式に設定します
DATE_SUB()	日付から時間値 (間隔) を引きます
DATEDIFF()	2 つの日付の差を求めます
DAY()	DAYOFMONTH() のシノニムです
DAYNAME()	曜日の名前を返します
DAYOFMONTH()	月の日を返します (0 - 31)
DAYOFWEEK()	引数の曜日インデックスを返します
DAYOFYEAR()	年の日を返します (1 - 366)
EXTRACT()	日付の一部を抽出します
FROM_DAYS()	日数を日付に変換します
FROM_UNIXTIME()	Unix タイムスタンプを日付として書式設定
GET_FORMAT()	日付書式文字列を返します
HOUR()	時を抽出します
LAST_DAY	引数の月の最終日を返します
LOCALTIME() , LOCALTIME	NOW() のシノニムです
LOCALTIMESTAMP , LOCALTIMESTAMP()	NOW() のシノニムです
MAKEDATE()	年と年間通算日から日付を作成します
MAKETIME()	時、分、秒から時間を作成します
MICROSECOND()	引数からマイクロ秒を返します
MINUTE()	引数から分を返します
MONTH()	渡された日付から月を返します

名前	説明
MONTHNAME()	月の名前を返します
NOW()	現在の日付と時間を返します
PERIOD_ADD()	年月に期間を加算します
PERIOD_DIFF()	期間内の月数を返します
QUARTER()	日付引数から四半期を返します
SEC_TO_TIME()	秒を hh:mm:ss 形式に変換
SECOND()	秒 (0-59) を返します
STR_TO_DATE()	文字列を日付に変換します
SUBDATE()	3 つの引数で呼び出されるときは DATE_SUB() のシノニムです
SUBTIME()	時間の差を求めます
SYSDATE()	この関数が実行される時間を返します
TIME()	渡された式の時部分を抽出します
TIME_FORMAT()	時間として書式設定します
TIME_TO_SEC()	秒に変換された引数を返します
TIMEDIFF()	時間の差を求めます
TIMESTAMP()	引数が 1 つの場合、この関数は日付または日付時間式を返します。引数が 2 つの場合、引数の合計を返します
TIMESTAMPADD()	日付時間式に間隔を加算します
TIMESTAMPDIFF()	日付時間式から間隔を減算します
TO_DAYS()	日に変換された日付引数を返します
TO_SECONDS()	0 年以降の秒数に変換された日付または日付時間引数を返します
UNIX_TIMESTAMP()	Unix タイムスタンプを返します
UTC_DATE()	現在の UTC 日付を返します
UTC_TIME()	現在の UTC 時間を返します
UTC_TIMESTAMP()	現在の UTC 日付と時間を返します
WEEK()	週番号を返します
WEEKDAY()	曜日インデックスを返します
WEEKOFYEAR()	日付の暦週を返します (1 - 53)
YEAR()	年を返します
YEARWEEK()	年と週を返します

次に、日付関数の使用例を示します。次のクエリーは、過去 30 日以内の `date_col` 値を含むすべての行を選択します。

```
mysql> SELECT something FROM tbl_name
-> WHERE DATE_SUB(CURDATE(),INTERVAL 30 DAY) <= date_col;
```

このクエリーは、将来の日付を持つ行も選択します。

通常、日付値が要求される関数では、日付時間値が受け入れられ、時間の部分は無視されます。通常、時間値が要求される関数では、日付時間値が受け入れられ、日付の部分は無視されます。

現在の日付または時間をそれぞれ返す関数は、クエリー実行の開始時にクエリーごとに 1 回だけ評価されます。つまり、`NOW()` などの関数が単一クエリー内で複数回参照されても、常に同じ結果が生成されます。(設計上、単一クエリーにはストアプログラム (ストアルーチン、トリガー、またはイベント) の呼び出し、およびそのプログラムによって呼び出されるすべてのサブプログラムも含まれています。) この原則

は、`CURDATE()`、`CURTIME()`、`UTC_DATE()`、`UTC_TIME()`、`UTC_TIMESTAMP()`、およびそれらのシノニムにも適用されます。

`CURRENT_TIMESTAMP()`、`CURRENT_TIME()`、`CURRENT_DATE()` および `FROM_UNIXTIME()` 関数は、`time_zone` システム変数のセッション値として使用可能な現在のセッションタイムゾーンの値を返します。また、`UNIX_TIMESTAMP()` では、引数がセッションタイムゾーンの日時値であると想定しています。セクション 5.1.15 「MySQL Server でのタイムゾーンのサポート」を参照してください。

「zero」日付または '2001-11-00' のような不完全な日付とともに使用できる日付関数もありますが、使用できない日付関数もあります。通常、日付の一部を抽出する関数は不完全な日付でも正しく機能するため、ゼロ以外の値が要求される場合に 0 を返すことができます。例:

```
mysql> SELECT DAYOFMONTH('2001-11-00'), MONTH('2005-00-00');
-> 0, 0
```

その他の関数では完全な日付が要求され、日付が不完全な場合は `NULL` が返されます。これらには、日付演算を実行する関数や日付の一部を名前にマップする関数が含まれます。例:

```
mysql> SELECT DATE_ADD('2006-05-00',INTERVAL 1 DAY);
-> NULL
mysql> SELECT DAYNAME('2006-05-00');
-> NULL
```

`DATE()` 関数の値を引数として渡し、日部分がゼロの不完全な日付を拒否すると、いくつかの関数が厳密になります: `CONVERT_TZ()`、`DATE_ADD()`、`DATE_SUB()`、`DAYOFYEAR()`、`TIMESTAMPDIFF()`、`TO_DAYS()`、`TO_SECONDS()`、`WEEK()`、`WEEKDAY()`、`WEEKOFYEAR()`、`YEARWEEK()`。

`TIME`、`DATETIME` および `TIMESTAMP` 値の小数秒は、マイクロ秒までの精度でサポートされます。時間引数を取る関数は、小数秒を含む値を受け入れます。時間関数からの戻り値には、必要に応じて小数秒が含まれます。

- `ADDDATE(date,INTERVAL expr unit)`、`ADDDATE(expr,days)`

`INTERVAL` 形式の 2 番目の引数を付けて呼び出されると、`ADDDATE()` は `DATE_ADD()` のシノニムになります。関連する関数 `SUBDATE()` は `DATE_SUB()` のシノニムです。 `INTERVAL unit` 引数の詳細は、時間間隔を参照してください。

```
mysql> SELECT DATE_ADD('2008-01-02', INTERVAL 31 DAY);
-> '2008-02-02'
mysql> SELECT ADDDATE('2008-01-02', INTERVAL 31 DAY);
-> '2008-02-02'
```

`days` 形式の 2 番目の引数を付けて呼び出されると、MySQL では `expr` に加算される整数の日数として処理されます。

```
mysql> SELECT ADDDATE('2008-01-02', 31);
-> '2008-02-02'
```

- `ADDTIME(expr1,expr2)`

`ADDTIME()` は `expr2` と `expr1` を加算し、その結果を返します。 `expr1` は時間または日付時間式であり、 `expr2` は時間式です。

```
mysql> SELECT ADDTIME('2007-12-31 23:59:59.999999', '1 1:1:1.000002');
-> '2008-01-02 01:01:01.000001'
mysql> SELECT ADDTIME('01:00:00.999999', '02:00:00.999998');
-> '03:00:01.999997'
```

- `CONVERT_TZ(dt,from_tz,to_tz)`

`CONVERT_TZ()` は、日付時間値 `dt` を `from_tz` で指定されたタイムゾーンから、 `to_tz` で指定されたタイムゾーンに変換し、結果の値を返します。タイムゾーンは、セクション 5.1.15 「MySQL Server でのタイムゾーンのサポート」で説明されているように指定されます。引数が無効な場合、この関数は `NULL` を返します。

`from_tz` から UTC に変換される際に、値が `TIMESTAMP` でサポートされている範囲から外れている場合は、変換が実行されません。 `TIMESTAMP` の範囲については、セクション 11.2.1 「日時データ型の構文」で説明されています。

```
mysql> SELECT CONVERT_TZ('2004-01-01 12:00:00','GMT','MET');
```

```
mysql> SELECT CONVERT_TZ('2004-01-01 13:00:00',
mysql> '2004-01-01 12:00:00','+00:00','+10:00');
mysql> '2004-01-01 22:00:00'
```

注記

'MET'や'Europe/Amsterdam'などの名前付きタイムゾーンを使用するには、タイムゾーンテーブルが適切に設定されている必要があります。その手順は、[セクション 5.1.15「MySQL Server でのタイムゾーンのサポート」](#)を参照してください。

• CURDATE()

関数が文字列または数値のどちらのコンテキストで使用されているかに応じて、現在の日付を'YYYY-MM-DD'または YYYYMMDD 形式の値として返します。

```
mysql> SELECT CURDATE();
mysql> '2008-06-13'
mysql> SELECT CURDATE() + 0;
mysql> '20080613'
```

• CURRENT_DATE, CURRENT_DATE()

CURRENT_DATE および CURRENT_DATE() は CURDATE() のシノニムです。

• CURRENT_TIME, CURRENT_TIME([fsp])

CURRENT_TIME および CURRENT_TIME() は CURTIME() のシノニムです。

• CURRENT_TIMESTAMP, CURRENT_TIMESTAMP([fsp])

CURRENT_TIMESTAMP および CURRENT_TIMESTAMP() は NOW() のシノニムです。

• CURTIME([fsp])

関数が文字列または数値のどちらのコンテキストで使用されているかに応じて、現在の時間を'hh:mm:ss'または hhmmss 形式の値として返します。値はセッションのタイムゾーンで表されます。

0 から 6 までの小数秒精度を指定するために fsp 引数が指定されている場合、戻り値にはその桁数の小数秒部分が含まれます。

```
mysql> SELECT CURTIME();
mysql> '23:50:26'
mysql> SELECT CURTIME() + 0;
mysql> '235026.000000'
```

• DATE(expr)

日付または日付時間式 expr の日付部分を抽出します。

```
mysql> SELECT DATE('2003-12-31 01:02:03');
mysql> '2003-12-31'
```

• DATEDIFF(expr1,expr2)

DATEDIFF() は、ある日付から別の日付までの日数の値として表現された expr1 - expr2 を返します。expr1 および expr2 は、日付または日付時間式です。値の日付部分のみが計算に使用されます。

```
mysql> SELECT DATEDIFF('2007-12-31 23:59:59','2007-12-30');
mysql> '1'
mysql> SELECT DATEDIFF('2010-11-30 23:59:59','2010-12-31');
mysql> '-31'
```

• DATE_ADD(date,INTERVAL expr unit), DATE_SUB(date,INTERVAL expr unit)

これらの関数は日付演算を実行します。date 引数は、開始日または日時の値を指定します。expr は、開始日に対して加算または減算される間隔値を指定する式です。expr は文字列として評価されます。負の間隔では - で始まる場合があります。unit は、式を解釈する単位を示すキーワードです。

`unit` 指定子の完全なリスト、各 `unit` 値に必要な `expr` 引数の形式、および時間演算でのオペランド解釈の規則など、時間間隔構文の詳細は、[時間間隔](#) を参照してください。

戻り値は引数によって異なります。

- `date` 引数が `DATE` 値で、計算に `YEAR`、`MONTH` および `DAY` 部分のみが含まれる (つまり、時間部分が含まれない) 場合は、`DATE`。
- 第 1 引数が `DATETIME` (または `TIMESTAMP`) 値である場合と、第 1 引数が `DATE` で、`unit` 値に `HOURS`、`MINUTES`、または `SECONDS` が使用されている場合は、`DATETIME` です。
- それ以外の場合は文字列です。

必ず結果が `DATETIME` になるようにするには、`CAST()` を使用すれば、第 1 引数を `DATETIME` に変換できます。

```
mysql> SELECT DATE_ADD('2018-05-01',INTERVAL 1 DAY);
-> '2018-05-02'
mysql> SELECT DATE_SUB('2018-05-01',INTERVAL 1 YEAR);
-> '2017-05-01'
mysql> SELECT DATE_ADD('2020-12-31 23:59:59',
-> INTERVAL 1 SECOND);
-> '2021-01-01 00:00:00'
mysql> SELECT DATE_ADD('2018-12-31 23:59:59',
-> INTERVAL 1 DAY);
-> '2019-01-01 23:59:59'
mysql> SELECT DATE_ADD('2100-12-31 23:59:59',
-> INTERVAL '1:1' MINUTE_SECOND);
-> '2101-01-01 00:01:00'
mysql> SELECT DATE_SUB('2025-01-01 00:00:00',
-> INTERVAL '1 1:1:1' DAY_SECOND);
-> '2024-12-30 22:58:59'
mysql> SELECT DATE_ADD('1900-01-01 00:00:00',
-> INTERVAL '-1 10' DAY_HOUR);
-> '1899-12-30 14:00:00'
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
-> '1997-12-02'
mysql> SELECT DATE_ADD('1992-12-31 23:59:59.000002',
-> INTERVAL '1.999999' SECOND_MICROSECOND);
-> '1993-01-01 00:00:01.000001'
```

- `DATE_FORMAT(date,format)`

`format` 文字列に従って、`date` 値を書式設定します。

次のテーブルに示す指定子は、`format` 文字列で使用できます。書式指定子文字の前には、`%` 文字を付ける必要があります。指定子は他の関数にも適用されます: `STR_TO_DATE()`、`TIME_FORMAT()`、`UNIX_TIMESTAMP()`。

指定子	説明
<code>%a</code>	簡略曜日名 (Sun..Sat)
<code>%b</code>	簡略月名 (Jan..Dec)
<code>%c</code>	月、数字 (0..12)
<code>%D</code>	英語のサフィクスを持つ日付 (0th, 1st, 2nd, 3rd, ...)
<code>%d</code>	日、数字 (00..31)
<code>%e</code>	日、数字 (0..31)
<code>%f</code>	マイクロ秒 (000000..999999)
<code>%H</code>	時間 (00..23)
<code>%h</code>	時間 (01..12)
<code>%l</code>	時間 (01..12)
<code>%i</code>	分、数字 (00..59)
<code>%j</code>	年間通算日 (001..366)

指定子	説明
%k	時 (0..23)
%l	時 (1..12)
%M	月名 (January..December)
%m	月、数字 (00..12)
%p	AM または PM
%r	時間、12 時間単位 (hh:mm:ss に AM または PM が続く)
%S	秒数 (0059)
%s	秒数 (0059)
%T	時間、24 時間単位 (hh:mm:ss)
%U	週 (00..53)、日曜日が週の初日、WEEK() モード 0
%u	週 (00..53)、月曜日が週の初日、WEEK() モード 1
%V	週 (01..53)、日曜日が週の初日、WEEK() モード 2、%X とともに使用
%v	週 (01..53)、月曜日が週の初日、WEEK() モード 3、%x とともに使用
%W	曜日名 (Sunday..Saturday)
%w	曜日 (0=Sunday..6=Saturday)
%X	年間の週、日曜日が週の初日、数字、4 桁、%V とともに使用
%x	年間の週、月曜日が週の初日、数字、4 桁、%v とともに使用
%Y	年、数字、4 桁
%y	年、数字 (2 桁)
%%	リテラル % 文字
%x	x (上記にないすべての「x」)

MySQL では '2014-00-00' などの不完全な日付の格納が許可されるため、月および日の指定子の範囲はゼロから始まります。

日および月の名前と略語に使用される言語は、`lc_time_names` システム変数 (セクション10.16 「MySQL Server のロケールサポート」) の値で制御されます。

%U、%u、%V、および %v 指定子のモード値については、WEEK() 関数の説明を参照してください。モードによって、週番号が付与される方法が影響を受けます。

DATE_FORMAT() は、ASCII 以外の文字を含む月および週の名前を返すことができるように、`character_set_connection` および `collation_connection` で指定された文字セットおよび照合順序を含む文字列を返します。

```
mysql> SELECT DATE_FORMAT('2009-10-04 22:23:00', '%W %M %Y');
-> 'Sunday October 2009'
mysql> SELECT DATE_FORMAT('2007-10-04 22:23:00', '%H:%i:%s');
-> '22:23:00'
mysql> SELECT DATE_FORMAT('1900-10-04 22:23:00',
-> '%D %y %a %d %m %b %j');
-> '4th 00 Thu 04 10 Oct 277'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00',
-> '%H %k %l %r %T %S %w');
-> '22 22 10 10:23:00 PM 22:23:00 00 6'
mysql> SELECT DATE_FORMAT('1999-01-01', '%X %V');
-> '1998 52'
mysql> SELECT DATE_FORMAT('2006-06-00', '%d');
-> '00'
```


- [DATE_SUB\(date,INTERVAL expr unit\)](#)

[DATE_ADD\(\)](#) の説明を参照してください。

- [DAY\(date\)](#)

[DAY\(\)](#) は [DAYOFMONTH\(\)](#) のシノニムです。

- [DAYNAME\(date\)](#)

[date](#) に対応する曜日の名前を返します。名前に使用される言語は、[lc_time_names](#) システム変数 ([セクション 10.16 「MySQL Server のロケールサポート」](#)) の値で制御されます。

```
mysql> SELECT DAYNAME('2007-02-03');
-> 'Saturday'
```

- [DAYOFMONTH\(date\)](#)

1 から 31 までの範囲内で [date](#) に対応する日を返します。'0000-00-00' や '2008-00-00' のように日の部分がゼロの場合は、0 を返します。

```
mysql> SELECT DAYOFMONTH('2007-02-03');
-> 3
```

- [DAYOFWEEK\(date\)](#)

[date](#) の曜日インデックス (1 = Sunday、2 = Monday、...、7 = Saturday) を返します。これらのインデックス値は、ODBC 標準に対応しています。

```
mysql> SELECT DAYOFWEEK('2007-02-03');
-> 7
```

- [DAYOFYEAR\(date\)](#)

1 から 366 までの範囲内で [date](#) に対応する通日を返します。

```
mysql> SELECT DAYOFYEAR('2007-02-03');
-> 34
```

- [EXTRACT\(unit FROM date\)](#)

[EXTRACT\(\)](#) 関数は、[DATE_ADD\(\)](#) または [DATE_SUB\(\)](#) と同じ種類の [unit](#) 指定子を使用しますが、日付演算を実行するのではなく、日付から部分を抽出します。[unit](#) 引数の詳細は、[時間間隔](#) を参照してください。

```
mysql> SELECT EXTRACT(YEAR FROM '2019-07-02');
-> 2019
mysql> SELECT EXTRACT(YEAR_MONTH FROM '2019-07-02 01:02:03');
-> 201907
mysql> SELECT EXTRACT(DAY_MINUTE FROM '2019-07-02 01:02:03');
-> 20102
mysql> SELECT EXTRACT(MICROSECOND
-> FROM '2003-01-02 10:30:00.000123');
-> 123
```

- [FROM_DAYS\(N\)](#)

日数 [N](#) が指定され、[DATE](#) 値を返します。

```
mysql> SELECT FROM_DAYS(730669);
-> '2000-07-03'
```

古い日付では、[FROM_DAYS\(\)](#) を慎重に使用してください。グレゴリオ暦 (1582) の出現よりも前の値とともに使用することを目的に設計されていません。[セクション 12.9 「MySQL で使用されるカレンダー」](#) を参照してください。

- [FROM_UNIXTIME\(unix_timestamp\[,format\]\)](#)

関数が文字列または数値のどちらのコンテキストで使用されているかに応じて、[unix_timestamp](#) 引数の表現を 'YYYY-MM-DD hh:mm:ss' または 'YYYYMMDDhhmmss' 形式の値として返します。[unix_timestamp](#)

は、'1970-01-01 00:00:00' UTC 以降の秒数を表す内部タイムスタンプ値で、UNIX_TIMESTAMP() 関数によって生成されます。

戻り値は、セッションのタイムゾーンで表されます。(クライアントは、[セクション5.1.15「MySQL Serverでのタイムゾーンのサポート」](#)の説明に従ってセッションタイムゾーンを設定できます。) format 文字列 (指定されている場合は、DATE_FORMAT() 関数のエントリで説明されているのと同じ方法で結果を書式設定するために使用されます)。

```
mysql> SELECT FROM_UNIXTIME(1447430881);
-> '2015-11-13 10:08:01'
mysql> SELECT FROM_UNIXTIME(1447430881) + 0;
-> 20151113100801
mysql> SELECT FROM_UNIXTIME(1447430881,
-> '%Y %D %M %h:%i:%s %x');
-> '2015 13th November 10:08:01 2015'
```

注記

UNIX_TIMESTAMP() および FROM_UNIXTIME() を使用して UTC 以外のタイムゾーンの値と Unix タイムスタンプ値の間で変換する場合、マッピングは双方向では一対一ではないため、変換は失われます。詳細は UNIX_TIMESTAMP() 関数の説明を参照してください。

- GET_FORMAT({DATE|TIME|DATETIME}, {EUR|'USA'|JIS|'ISO'|INTERNAL'})

書式文字列を返します。この関数は、DATE_FORMAT() およびSTR_TO_DATE() 関数と組み合わせて使用すると便利です。

1 番目と 2 番目の引数に値を指定できるため、複数の書式文字列を生成できます (使用される指定子については、DATE_FORMAT() 関数の説明で示す表を参照してください)。ISO 書式は ISO 8601 ではなく、ISO 9075 を参照しています。

関数呼び出し	結果
GET_FORMAT(DATE,'USA')	'%m.%d.%Y'
GET_FORMAT(DATE,'JIS')	'%Y-%m-%d'
GET_FORMAT(DATE,'ISO')	'%Y-%m-%d'
GET_FORMAT(DATE,'EUR')	'%d.%m.%Y'
GET_FORMAT(DATE,'INTERNAL')	'%Y%m%d'
GET_FORMAT(DATETIME,'USA')	'%Y-%m-%d %H.%i.%s'
GET_FORMAT(DATETIME,'JIS')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(DATETIME,'ISO')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(DATETIME,'EUR')	'%Y-%m-%d %H.%i.%s'
GET_FORMAT(DATETIME,'INTERNAL')	'%Y%m%d%H%i%s'
GET_FORMAT(TIME,'USA')	'%h:%i:%s %p'
GET_FORMAT(TIME,'JIS')	'%H:%i:%s'
GET_FORMAT(TIME,'ISO')	'%H:%i:%s'
GET_FORMAT(TIME,'EUR')	'%H.%i.%s'
GET_FORMAT(TIME,'INTERNAL')	'%H%i%s'

TIMESTAMP は、GET_FORMAT() への 1 番目の引数としても使用できます。その場合、関数は DATETIME の場合と同じ値を返します。

```
mysql> SELECT DATE_FORMAT('2003-10-03',GET_FORMAT(DATE,'EUR'));
-> '03.10.2003'
mysql> SELECT STR_TO_DATE('10.31.2003',GET_FORMAT(DATE,'USA'));
-> '2003-10-31'
```

- **HOUR(time)**

time に対応する時を返します。戻り値の範囲は、日付時間値の **0** から **23** までです。ただし、**TIME** 値の範囲は実際にはもっと大きいため、**HOUR** は **23** よりも大きい値を返すことができます。

```
mysql> SELECT HOUR('10:05:03');
-> 10
mysql> SELECT HOUR('272:59:59');
-> 272
```

- **LAST_DAY(date)**

日付または日付時間の値が指定され、月の最終日に対応する値を返します。引数が無効である場合は、**NULL** を返します。

```
mysql> SELECT LAST_DAY('2003-02-05');
-> '2003-02-28'
mysql> SELECT LAST_DAY('2004-02-05');
-> '2004-02-29'
mysql> SELECT LAST_DAY('2004-01-01 01:01:01');
-> '2004-01-31'
mysql> SELECT LAST_DAY('2003-03-32');
-> NULL
```

- **LOCALTIME, LOCALTIME([fsp])**

LOCALTIME および **LOCALTIME()** は **NOW()** のシノニムです。

- **LOCALTIMESTAMP, LOCALTIMESTAMP([fsp])**

LOCALTIMESTAMP および **LOCALTIMESTAMP()** は **NOW()** のシノニムです。

- **MAKEDATE(year,dayofyear)**

指定された年と年間通算値から、日付を返します。**dayofyear** は 0 よりも大きくする必要があり、それ以外の場合結果が **NULL** になります。

```
mysql> SELECT MAKEDATE(2011,31), MAKEDATE(2011,32);
-> '2011-01-31', '2011-02-01'
mysql> SELECT MAKEDATE(2011,365), MAKEDATE(2014,365);
-> '2011-12-31', '2014-12-31'
mysql> SELECT MAKEDATE(2011,0);
-> NULL
```

- **MAKETIME(hour,minute,second)**

hour、**minute**、および **second** 引数から計算された時間値を返します。

second 引数には小数部を含めることができます。

```
mysql> SELECT MAKETIME(12,15,30);
-> '12:15:30'
```

- **MICROSECOND(expr)**

0 から **999999** までの範囲内の数値として、時間または日付時間式 **expr** からのマイクロ秒を返します。

```
mysql> SELECT MICROSECOND('12:00:00.123456');
-> 123456
mysql> SELECT MICROSECOND('2019-12-31 23:59:59.000010');
-> 10
```

- **MINUTE(time)**

0 から **59** までの範囲内で、**time** に対応する分を返します。

```
mysql> SELECT MINUTE('2008-02-03 10:05:03');
-> 5
```

- **MONTH(date)**

1 (1 月) から 12 (12 月) の範囲内で、`date` に対応する月を返します。'0000-00-00' や '2008-00-00' のように月の部分がゼロの場合は、0 を返します。

```
mysql> SELECT MONTH('2008-02-03');
-> 2
```

- `MONTHNAME(date)`

`date` に対応する月の完全名を返します。名前に使用される言語は、`lc_time_names` システム変数 (セクション 10.16 「MySQL Server のロケールサポート」) の値で制御されます。

```
mysql> SELECT MONTHNAME('2008-02-03');
-> 'February'
```

- `NOW([fsp])`

関数が文字列または数値のどちらのコンテキストで使用されているかに応じて、現在の日時を 'YYYY-MM-DD hh:mm:ss' または 'YYYYMMDDhhmmss' 形式の値として返します。値はセッションのタイムゾーンで表されます。

0 から 6 までの小数秒精度を指定するために `fsp` 引数が指定されている場合、戻り値にはその桁数の小数秒部分が含まれます。

```
mysql> SELECT NOW();
-> '2007-12-15 23:50:26'
mysql> SELECT NOW() + 0;
-> 20071215235026.000000
```

`NOW()` は、ステートメントが実行を開始する時刻を示す定数時間を返します。(ストアドファンクションまたはトリガーでは、`NOW()` は関数またはトリガーステートメントが実行を開始する時間を返します。) これは、正確な実行時間を返す `SYSDATE()` の動作とは異なります。

```
mysql> SELECT NOW(), SLEEP(2), NOW();
+-----+-----+-----+
| NOW()          | SLEEP(2) | NOW()          |
+-----+-----+-----+
| 2006-04-12 13:47:36 | 0 | 2006-04-12 13:47:36 |
+-----+-----+-----+

mysql> SELECT SYSDATE(), SLEEP(2), SYSDATE();
+-----+-----+-----+
| SYSDATE()      | SLEEP(2) | SYSDATE()      |
+-----+-----+-----+
| 2006-04-12 13:47:44 | 0 | 2006-04-12 13:47:46 |
+-----+-----+-----+
```

さらに、`SET TIMESTAMP` ステートメントによって、`NOW()` で返された値は影響を受けませんが、`SYSDATE()` で返された値は影響を受けません。つまり、バイナリログのタイムスタンプ設定は、`SYSDATE()` の呼び出しに影響しないことを意味します。タイムスタンプをゼロ以外の値に設定すると、後続の `NOW()` が起動されるたびに、その値が返されます。タイムスタンプをゼロに設定すると、この効果が取り消され、再度 `NOW()` が現在の日付と時間を返すようになります。

2 つの関数の相違点についての詳細は、`SYSDATE()` の説明を参照してください。

- `PERIOD_ADD(P,N)`

`N` 月を期間 `P` に (YYMM または YYYYMM の書式で) 加算します。YYYYMM の書式で値を返します。

注記

期間引数 `P` は、日付値ではありません。

```
mysql> SELECT PERIOD_ADD(200801,2);
-> 200803
```

- `PERIOD_DIFF(P1,P2)`

期間 `P1` と `P2` 間の月数を返します。 `P1` および `P2` は、`YYMM` または `YYYYMM` の書式にする必要があります。 期間引数 `P1` および `P2` は日付値ではないことに注意してください。

```
mysql> SELECT PERIOD_DIFF(200802,200703);
-> 11
```

- `QUARTER(date)`

1 から 4 までの範囲内で `date` に対応する四半期を返します。

```
mysql> SELECT QUARTER('2008-04-01');
-> 2
```

- `SECOND(time)`

0 から 59 までの範囲内で、`time` に対応する秒数を返します。

```
mysql> SELECT SECOND('10:05:03');
-> 3
```

- `SEC_TO_TIME(seconds)`

`TIME` 値として、時、分、秒に変換された `seconds` 引数を返します。 結果の範囲は、`TIME` データ型の範囲に制約されます。 引数とその範囲外の値に対応している場合は、警告が発行されます。

```
mysql> SELECT SEC_TO_TIME(2378);
-> '00:39:38'
mysql> SELECT SEC_TO_TIME(2378) + 0;
-> 3938
```

- `STR_TO_DATE(str,format)`

これは `DATE_FORMAT()` 関数の逆です。 文字列 `str` と書式文字列 `format` が指定されます。 `STR_TO_DATE()` は、書式文字列に日付と時間の両方の部分が含まれる場合は `DATETIME` 値を返し、文字列に日付と時間の部分の一方のみが含まれる場合は `DATE` または `TIME` 値を返します。 `str` から抽出された日付値、時間値、または日付時間値が不正な場合は、`STR_TO_DATE()` によって `NULL` が返され、警告が発行されます。

サーバーは `str` をスキャンすることで、`format` の一致を試みます。 書式文字列には、リテラル文字と `%` で始まる書式指定子を含めることができます。 `format` 内のリテラル文字は、`str` 内と完全に一致する必要があります。 `format` 内の書式指定子は、`str` 内の日付または時間の部分に一致する必要があります。 `format` で使用できる指定子については、`DATE_FORMAT()` 関数の説明を参照してください。

```
mysql> SELECT STR_TO_DATE('01,5,2013','%d,%m,%Y');
-> '2013-05-01'
mysql> SELECT STR_TO_DATE('May 1, 2013','%M %d,%Y');
-> '2013-05-01'
```

`str` の先頭からスキャンが開始され、一致しない `format` が見つかった場合は失敗します。 `str` の末尾にある余分な文字は、無視されます。

```
mysql> SELECT STR_TO_DATE('a09:30:17','a%h:%i:%s');
-> '09:30:17'
mysql> SELECT STR_TO_DATE('a09:30:17','%h:%i:%s');
-> NULL
mysql> SELECT STR_TO_DATE('09:30:17a','%h:%i:%s');
-> '09:30:17'
```

指定されていない日付または時間の部分の値は 0 になるため、`str` に指定された値が不完全な場合は、結果の一部または全部が 0 に設定されます。

```
mysql> SELECT STR_TO_DATE('abc','abc');
-> '0000-00-00'
mysql> SELECT STR_TO_DATE('9','%m');
-> '0000-09-00'
mysql> SELECT STR_TO_DATE('9','%s');
```

```
-> '00:00:09'
```

日付値の部分をチェックする範囲は、[セクション11.2.2「DATE、DATETIME、およびTIMESTAMP型」](#)の説明どおりです。たとえば、「ゼロ」の日付または部分値が0の日付は、このような値が許可されないようにSQLモードが設定されていなければ、許可されます。

```
mysql> SELECT STR_TO_DATE('00/00/0000', '%m/%d/%Y');
-> '0000-00-00'
mysql> SELECT STR_TO_DATE('04/31/2004', '%m/%d/%Y');
-> '2004-04-31'
```

NO_ZERO_DATE SQLモードが有効な場合、ゼロの日付は許可されません。その場合、**STR_TO_DATE()** は **NULL** を返し、警告を生成します:

```
mysql> SET sql_mode = '';
mysql> SELECT STR_TO_DATE('00/00/0000', '%m/%d/%Y');
+-----+
| STR_TO_DATE('00/00/0000', '%m/%d/%Y') |
+-----+
| 0000-00-00 |
+-----+
mysql> SET sql_mode = 'NO_ZERO_DATE';
mysql> SELECT STR_TO_DATE('00/00/0000', '%m/%d/%Y');
+-----+
| STR_TO_DATE('00/00/0000', '%m/%d/%Y') |
+-----+
| NULL |
+-----+
mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Warning
Code: 1411
Message: Incorrect datetime value: '00/00/0000' for function str_to_date
```

注記

週が月の境界を越えた場合は、年と週の組み合わせでは年と月を一意に識別できないため、"**%X%V**"の書式を使用しても、年と週の文字列を日付に変換できません。年と週を日付に変換するには、曜日も指定する必要があります。

```
mysql> SELECT STR_TO_DATE('200442 Monday', '%X%V %W');
-> '2004-10-18'
```

- **SUBDATE(date,INTERVAL expr unit)**, **SUBDATE(expr,days)**

INTERVAL 形式で2番目の引数を付けて呼び出されると、**SUBDATE()** は **DATE_SUB()** のシノニムになります。**INTERVAL unit** 引数については、**DATE_ADD()** の説明を参照してください。

```
mysql> SELECT DATE_SUB('2008-01-02', INTERVAL 31 DAY);
-> '2007-12-02'
mysql> SELECT SUBDATE('2008-01-02', INTERVAL 31 DAY);
-> '2007-12-02'
```

2番目の形式では、**days**の整数値を使用できます。このような場合は、日付または日付時間式 **expr** から日数が減算されると解釈されます。

```
mysql> SELECT SUBDATE('2008-01-02 12:00:00', 31);
-> '2007-12-02 12:00:00'
```

- **SUBTIME(expr1,expr2)**

SUBTIME() は、**expr1**と同じ書式で表現される **expr1** と **expr2** を返します。**expr1** は時間または日付時間式であり、**expr2** は時間式です。

```
mysql> SELECT SUBTIME('2007-12-31 23:59:59.999999','1 1:1:1.000002');
-> '2007-12-30 22:58:58.999997'
mysql> SELECT SUBTIME('01:00:00.999999','02:00:00.999998');
-> '-00:59:59.999999'
```

- **SYSDATE([fsp])**

関数が文字列または数値のどちらのコンテキストで使用されているかに応じて、現在の日時を 'YYYY-MM-DD hh:mm:ss' または 'YYYYMMDDhhmmss' 形式の値として返します。

0 から 6 までの小数秒精度を指定するために `fsp` 引数が指定されている場合、戻り値にはその桁数の小数秒部分が含まれます。

`SYSDATE()` は、実行された時間を返します。これは、ステートメントが実行を開始する時間を示す定数時間を返す `NOW()` の動作とは異なります。(ストアドファンクションまたはトリガーでは、`NOW()` は関数またはトリガーステートメントが実行を開始する時間を返します。)

```
mysql> SELECT NOW(), SLEEP(2), NOW();
+-----+-----+-----+
| NOW()          | SLEEP(2) | NOW()          |
+-----+-----+-----+
| 2006-04-12 13:47:36 | 0 | 2006-04-12 13:47:36 |
+-----+-----+-----+

mysql> SELECT SYSDATE(), SLEEP(2), SYSDATE();
+-----+-----+-----+
| SYSDATE()      | SLEEP(2) | SYSDATE()      |
+-----+-----+-----+
| 2006-04-12 13:47:44 | 0 | 2006-04-12 13:47:46 |
+-----+-----+-----+
```

さらに、`SET TIMESTAMP` ステートメントによって、`NOW()` で返された値は影響を受けますが、`SYSDATE()` で返された値は影響を受けません。つまり、バイナリログのタイムスタンプ設定は、`SYSDATE()` の呼び出しに影響しないことを意味します。

`SYSDATE()` は、同じステートメント内でもさまざまな値を返すことができ、`SET TIMESTAMP` による影響も受けないため、非決定的です。そのため、ステートメントベースのバイナリロギングが使用されている場合は、レプリケーションで安全に使用できません。これが問題となる場合は、行ベースのロギングを使用できます。

また、`--sysdate-is-now` オプションを使用すると、`SYSDATE()` を `NOW()` のエイリアスにすることができます。これは、レプリケーションソースサーバーとレプリカの両方でこのオプションが使用されている場合に機能します。

`SYSDATE()` に非決定的な特性があるということは、それを参照する式を評価する際にインデックスを使用できないことも意味します。

- `TIME(expr)`

時間または日付時間式 `expr` の時部分を抽出し、文字列として返します。

この関数は、ステートメントベースのレプリケーションでは安全に使用できません。`binlog_format` が `STATEMENT` に設定されているときに、この関数を使用すると、警告のログが記録されます。

```
mysql> SELECT TIME('2003-12-31 01:02:03');
-> '01:02:03'
mysql> SELECT TIME('2003-12-31 01:02:03.000123');
-> '01:02:03.000123'
```

- `TIMEDIFF(expr1,expr2)`

`TIMEDIFF()` は、時間値として表現された `expr1 - expr2` を返します。`expr1` および `expr2` は時間または日付時間式ですが、両方とも同じ型にする必要があります。

`TIMEDIFF()` で返される結果は、`TIME` 値で許可される範囲に制限されています。また、`TIMESTAMPDIFF()` および `UNIX_TIMESTAMP()` 関数のいずれかを使用することもできます。両方とも整数を返します。

```
mysql> SELECT TIMEDIFF('2000:01:01 00:00:00',
-> '2000:01:01 00:00:00.000001');
-> '-00:00:00.000001'
mysql> SELECT TIMEDIFF('2008-12-31 23:59:59.000001',
-> '2008-12-30 01:01:01.000002');
-> '46:58:57.999999'
```

- **TIMESTAMP(expr), TIMESTAMP(expr1,expr2)**

引数を 1 つ付けると、この関数は日付または日付時間式 `expr` を日付時間値として返します。引数を 2 つ付けると、時間式 `expr2` を日付または日付時間式 `expr1` に加算し、その結果を日付時間値として返します。

```
mysql> SELECT TIMESTAMP('2003-12-31');
-> '2003-12-31 00:00:00'
mysql> SELECT TIMESTAMP('2003-12-31 12:00:00','12:00:00');
-> '2004-01-01 00:00:00'
```

- **TIMESTAMPADD(unit,interval,datetime_expr)**

整数式 `interval` を日付または日付時間式 `datetime_expr` に加算します。 `interval` の単位は、 `unit` 引数で指定されます。この引数は、 `MICROSECOND` (マイクロ秒)、 `SECOND`、 `MINUTE`、 `HOURL`、 `DAY`、 `WEEK`、 `MONTH`、 `QUARTER`、 `YEAR` 値のいずれかにする必要があります。

`unit` 値を指定するには、ここで示したキーワードのいずれかを使用するか、 `SQL_TSI_` をプリフィクスとして付けます。たとえば、 `DAY` と `SQL_TSI_DAY` は両方とも有効です。

```
mysql> SELECT TIMESTAMPADD(MINUTE,1,'2003-01-02');
-> '2003-01-02 00:01:00'
mysql> SELECT TIMESTAMPADD(WEEK,1,'2003-01-02');
-> '2003-01-09'
```

- **TIMESTAMPDIFF(unit,datetime_expr1,datetime_expr2)**

`datetime_expr2 - datetime_expr1` を返します。 `datetime_expr1` と `datetime_expr2` は、日付または日付時間式です。式的一方が日付で、他方が日付時間にすることもできます。日付値は、必要に応じて時間部分が `'00:00:00'` の日付時間として処理されます。結果 (整数) の単位は、 `unit` 引数で指定されます。 `unit` の有効な値は、 `TIMESTAMPADD()` 関数の説明で一覧表示された値と同じです。

```
mysql> SELECT TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01');
-> 3
mysql> SELECT TIMESTAMPDIFF(YEAR,'2002-05-01','2001-01-01');
-> -1
mysql> SELECT TIMESTAMPDIFF(MINUTE,'2003-02-01','2003-05-01 12:05:55');
-> 128885
```

注記

この関数の日付または日付時間引数の順序は、 `TIMESTAMP()` 関数を 2 つの引数を指定して呼び出す場合の順序と逆になります。

- **TIME_FORMAT(time,format)**

これは `DATE_FORMAT()` 関数と同様に使用されますが、 `format` 文字列には時間、分、秒、マイクロ秒の書式指定子のみを含めることができます。その他の指定子では、 `NULL` 値または `0` が生成されます。

`time` 値に 23 よりも大きな時間部分が含まれる場合は、 `%H` および `%k` 時間書式指定子によって、 0..23 の通常の範囲よりも大きな値が生成されます。その他の時間書式指定子では、時間値モジュロ 12 が生成されます。

```
mysql> SELECT TIME_FORMAT('100:00:00', '%H %k %h %l %I');
-> '100 100 04 04 4'
```

- **TIME_TO_SEC(time)**

秒に変換された `time` 引数を返します。

```
mysql> SELECT TIME_TO_SEC('22:23:00');
-> 80580
mysql> SELECT TIME_TO_SEC('00:39:38');
-> 2378
```

- `TO_DAYS(date)`

日付 `date` が指定され、日数 (0 年以降の日数) を返します。

```
mysql> SELECT TO_DAYS(950501);
-> 728779
mysql> SELECT TO_DAYS('2007-10-07');
-> 733321
```

`TO_DAYS()` は、カレンダーが変更された際に失われた日が考慮されないため、グレゴリオ暦 (1582) の出現よりも前の値とともに使用するために設計されていません。日付が 1582 よりも前の場合は (ほかのロケールでは、さらにあとの年になる可能性があります)、この関数の結果は信頼できません。詳細は、[セクション12.9「MySQL で使用されるカレンダー」](#) を参照してください。

MySQL では [セクション11.2「日時データ型」](#) のルールを使用して、日付の 2 桁の年の値が 4 桁の形式に変換されることを忘れないでください。たとえば、`'2008-10-07'` と `'08-10-07'` は同じ日付と認識されます。

```
mysql> SELECT TO_DAYS('2008-10-07'), TO_DAYS('08-10-07');
-> 733687, 733687
```

MySQL では、ゼロの日付は `'0000-00-00'` として定義されます。ただし、このデータ自体は無効とみなされます。つまり、`'0000-00-00'` および `'0000-01-01'` の場合、`TO_DAYS()` は次に示す値を返します。

```
mysql> SELECT TO_DAYS('0000-00-00');
+-----+
| to_days('0000-00-00') |
+-----+
|          NULL         |
+-----+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message                                     |
+-----+-----+-----+
| Warning | 1292 | Incorrect datetime value: '0000-00-00' |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT TO_DAYS('0000-01-01');
+-----+
| to_days('0000-01-01') |
+-----+
|          1            |
+-----+
1 row in set (0.00 sec)
```

このことは、`ALLOW_INVALID_DATES` SQL サーバーモードが有効であるかどうかに関係なく当てはまります。

- `TO_SECONDS(expr)`

日付または日時 `expr` を指定すると、0 年以降の秒数を返します。 `expr` が有効な日付または日付時間の値ではない場合は、`NULL` を返します。

```
mysql> SELECT TO_SECONDS(950501);
-> 62966505600
mysql> SELECT TO_SECONDS('2009-11-29');
-> 63426672000
mysql> SELECT TO_SECONDS('2009-11-29 13:43:32');
-> 63426721412
mysql> SELECT TO_SECONDS(NOW());
-> 63426721458
```

`TO_DAYS()` と同様に、`TO_SECONDS()` は、カレンダーが変更された際に失われた日が考慮されないため、グレゴリオ暦 (1582) の出現よりも前の値とともに使用する目的で設計されていません。日付が 1582 よりも前の場合は

(ほかのロケールでは、さらにあとの年になる可能性があります)、この関数の結果は信頼できません。詳細は、[セクション12.9「MySQLで使用されるカレンダー」](#)を参照してください。

`TO_DAYS()`と同様に、`TO_SECONDS()`は[セクション11.2「日時データ型」](#)のルールを使用して日付の2桁の年の値を4桁の形式に変換します。

MySQLでは、ゼロの日付は'`0000-00-00`'として定義されます。ただし、このデータ自体は無効とみなされます。つまり、'`0000-00-00`'および'`0000-01-01`'の場合、`TO_SECONDS()`は次に示す値を返します。

```
mysql> SELECT TO_SECONDS('0000-00-00');
+-----+
| TO_SECONDS('0000-00-00') |
+-----+
|          NULL          |
+-----+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Warning | 1292 | Incorrect datetime value: '0000-00-00' |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT TO_SECONDS('0000-01-01');
+-----+
| TO_SECONDS('0000-01-01') |
+-----+
|          86400          |
+-----+
1 row in set (0.00 sec)
```

このことは、`ALLOW_INVALID_DATES` SQL サーバーモードが有効であるかどうかに関係なく当てはまります。

- `UNIX_TIMESTAMP([date])`

`date` 引数を指定せずに `UNIX_TIMESTAMP()` をコールすると、'`1970-01-01 00:00:00`' UTC 以降の秒数を表す Unix タイムスタンプが返されます。

`date` 引数を指定して `UNIX_TIMESTAMP()` をコールすると、'`1970-01-01 00:00:00`' UTC 以降の秒数として引数の値が返されます。サーバーは、`date` をセッションタイムゾーンの値として解釈し、UTC の内部 Unix タイムスタンプ値に変換します。(クライアントは、[セクション5.1.15「MySQL Serverでのタイムゾーンのサポート」](#)の説明に従ってセッションタイムゾーンを設定できます。) `date` 引数には、`DATE`、`DATETIME` または `TIMESTAMP` 文字列、`YYMMDD`、`YYMMDDhhmmss`、`YYYYMMDD` の数値または `YYYYMMDDhhmmss` 形式を指定できます。引数に時間部分が含まれている場合、オプションで小数秒部分を含めることができます。

戻り値は、引数が指定されていない場合、または引数に小数秒部分が含まれていない場合は整数、小数秒部分を含む引数が指定されている場合は `DECIMAL` です。

`date` 引数が `TIMESTAMP` カラムの場合、`UNIX_TIMESTAMP()` は内部タイムスタンプ値を直接返し、「string-to-Unix-timestamp」の暗黙的な変換は行いません。

引数値の有効範囲は、`TIMESTAMP` データ型の場合と同じです: '`1970-01-01 00:00:01.000000`' UTC から '`2038-01-19 03:14:07.999999`' UTC。 `UNIX_TIMESTAMP()` に範囲外の日付を渡すと、`0` が返されます。

```
mysql> SELECT UNIX_TIMESTAMP();
-> 1447431666
mysql> SELECT UNIX_TIMESTAMP('2015-11-13 10:20:19');
-> 1447431619
mysql> SELECT UNIX_TIMESTAMP('2015-11-13 10:20:19.012');
-> 1447431619.012
```

`UNIX_TIMESTAMP()` および `FROM_UNIXTIME()` を使用して UTC 以外のタイムゾーンの値と Unix タイムスタンプ値の間で変換する場合、マッピングは双方向では一対一ではないため、変換は失われます。たとえば、夏時間 (DST) などのローカルタイムゾーンの変更の規則により、`UNIX_TIMESTAMP()` では UTC 以外のタイムゾーンで区

別される 2 つの値を同じ Unix タイムスタンプ値にマップできます。 `FROM_UNIXTIME()` では、その値は元の値のいずれかにのみマップされます。次に、`MET` タイムゾーンで区別される値を使用する例を示します:

```
mysql> SET time_zone = 'MET';
mysql> SELECT UNIX_TIMESTAMP('2005-03-27 03:00:00');
+-----+
| UNIX_TIMESTAMP('2005-03-27 03:00:00') |
+-----+
| 1111885200 |
+-----+
mysql> SELECT UNIX_TIMESTAMP('2005-03-27 02:00:00');
+-----+
| UNIX_TIMESTAMP('2005-03-27 02:00:00') |
+-----+
| 1111885200 |
+-----+
mysql> SELECT FROM_UNIXTIME(1111885200);
+-----+
| FROM_UNIXTIME(1111885200) |
+-----+
| 2005-03-27 03:00:00 |
+-----+
```

注記

'MET'や'Europe/Amsterdam'などの名前付きタイムゾーンを使用するには、タイムゾーンテーブルが適切に設定されている必要があります。その手順は、[セクション 5.1.15 「MySQL Server でのタイムゾーンのサポート」](#)を参照してください。

`UNIX_TIMESTAMP()` カラムを減算する場合は、符号付き整数にキャストできます。[セクション 12.11 「キャスト関数と演算子」](#)を参照してください。

• `UTC_DATE, UTC_DATE()`

関数が文字列または数値のどちらのコンテキストで使用されているかに応じて、現在の UTC 日付を 'YYYY-MM-DD' または 'YYYYMMDD' 形式の値として返します。

```
mysql> SELECT UTC_DATE(), UTC_DATE() + 0;
-> '2003-08-14', 20030814
```

• `UTC_TIME, UTC_TIME([fsp])`

関数が文字列または数値のどちらのコンテキストで使用されているかに応じて、現在の UTC 時間を 'hh:mm:ss' または 'hhmmss' 形式の値として返します。

0 から 6 までの小数秒精度を指定するために `fsp` 引数が指定されている場合、戻り値にはその桁数の小数秒部分が含まれます。

```
mysql> SELECT UTC_TIME(), UTC_TIME() + 0;
-> '18:07:53', 180753.000000
```

• `UTC_TIMESTAMP, UTC_TIMESTAMP([fsp])`

関数が文字列または数値のどちらのコンテキストで使用されているかに応じて、現在の UTC 日時を 'YYYY-MM-DD hh:mm:ss' または 'YYYYMMDDhhmmss' 形式の値として返します。

0 から 6 までの小数秒精度を指定するために `fsp` 引数が指定されている場合、戻り値にはその桁数の小数秒部分が含まれます。

```
mysql> SELECT UTC_TIMESTAMP(), UTC_TIMESTAMP() + 0;
-> '2003-08-14 18:08:04', 20030814180804.000000
```

• `WEEK(date[,mode])`

この関数は、`date` に対応する週番号を返します。2 つの引数を取る形式の `WEEK()` を使用すると、週が日曜日と月曜日のどちらから始まるのか、および戻り値が 0 から 53 までと 1 から 53 までのどちらの範囲内であるのかを指定できます。`mode` 引数が省略された場合は、`default_week_format` システム変数の値が使用されます。[セクション 5.1.8 「サーバーシステム変数」](#)を参照してください。

次の表では、`mode` 引数がどのように機能するのかについて説明します。

モード	週の 1 日目	範囲	第 1 週は次の条件を満たす最初の週
0	日曜日	0-53	本年の日曜日を含む
1	月曜日	0-53	本年の 4 日以上を含む
2	日曜日	1-53	本年の日曜日を含む
3	月曜日	1-53	本年の 4 日以上を含む
4	日曜日	0-53	本年の 4 日以上を含む
5	月曜日	0-53	本年の月曜日を含む
6	日曜日	1-53	本年の 4 日以上を含む
7	月曜日	1-53	本年の月曜日を含む

「本年の 4 日以上を含む」という意味を持つ `mode` 値では、ISO 8601:1988 に従って週番が付けられます。

- 1 月 1 日を含む週に新年の 4 日以上が含まれる場合は、その週が第 1 週です。
- それ以外の場合は、前年の最終週となり、次の週が第 1 週です。

```
mysql> SELECT WEEK('2008-02-20');
-> 7
mysql> SELECT WEEK('2008-02-20',0);
-> 7
mysql> SELECT WEEK('2008-02-20',1);
-> 8
mysql> SELECT WEEK('2008-12-31',1);
-> 53
```

日付が前年の最終週に入っている場合は、オプションの `mode` 引数として 2、3、6、または 7 を使用しなければ、MySQL によって 0 が返されます。

```
mysql> SELECT YEAR('2000-01-01'), WEEK('2000-01-01',0);
-> 2000, 0
```

指定された日付は実際には 1999 年の第 52 週に発生するため、`WEEK()` は 52 を返す必要があると議論されることもあります。代わりに `WEEK()` は、戻り値が「指定された年の週番号」となるように 0 を返します。これにより、日付から日付部分を抽出するその他の関数と組み合わせると、`WEEK()` 関数を信頼して使用できるようになります。

指定された日付に対応する週の 1 日目を含む年について評価された結果を優先する場合は、オプションの `mode` 引数として 0、2、5、または 7 を使用します。

```
mysql> SELECT WEEK('2000-01-01',2);
-> 52
```

または、`YEARWEEK()` 関数を使用します。

```
mysql> SELECT YEARWEEK('2000-01-01');
-> 199952
mysql> SELECT MID(YEARWEEK('2000-01-01'),5,2);
-> '52'
```

• `WEEKDAY(date)`

`date` に対応する曜日インデックス (0 = Monday、1 = Tuesday、...6 = Sunday) を返します。

```
mysql> SELECT WEEKDAY('2008-02-03 22:23:00');
-> 6
mysql> SELECT WEEKDAY('2007-11-06');
-> 1
```

• `WEEKOFYEAR(date)`

1 から 53 までの範囲内で、日付の暦週を返します。WEEKOFYEAR() は WEEK(date,3) に同等の互換性のある関数です。

```
mysql> SELECT WEEKOFYEAR('2008-02-20');
-> 8
```

- YEAR(date)

1000 から 9999 までの範囲内で、date に対応する年を返します。日付が「ゼロ」の場合は、0 を返します。

```
mysql> SELECT YEAR('1987-01-01');
-> 1987
```

- YEARWEEK(date), YEARWEEK(date,mode)

日付に対応する年と週を返します。結果の年と日付引数の年では、その年の最初と最後の週が異なる可能性があります。

mode 引数は、WEEK() への mode 引数とまったく同様に機能します。単一引数構文では、mode 値 0 が使用されます。WEEK() とは異なり、default_week_format の値は YEARWEEK() に影響しません。

```
mysql> SELECT YEARWEEK('1987-01-01');
-> 198652
```

WEEK() はその後、指定された年のコンテキストで週を返すため、週番号はオプションの引数 0 または 1 を付けた場合に WEEK() 関数で返される数字 (0) とは異なります。

12.8 文字列関数および演算子

表 12.12 「文字列関数および演算子」

名前	説明
ASCII()	左端の文字の数値を返します
BIN()	数値のバイナリ表現を含む文字列を返します
BIT_LENGTH()	ビット単位で引数の長さを返します
CHAR()	渡された各整数の文字を返します
CHAR_LENGTH()	引数の文字数を返します
CHARACTER_LENGTH()	CHAR_LENGTH() のシノニムです
CONCAT()	連結された文字列を返します
CONCAT_WS()	連結されたものをセパレータ付きで返します
ELT()	インデックス番号位置の文字列を返します
EXPORT_SET()	値 bits 内の各ビットが設定されている場合は on 文字列を取得し、各ビットが設定されていない場合には off 文字列を取得するように、文字列を返します
FIELD()	後続の引数の最初の引数のインデックス (位置)
FIND_IN_SET()	2 番目の引数内の最初の引数のインデックス (位置)
FORMAT()	指定された小数点以下桁数に書式設定された数値を返します
FROM_BASE64()	base64 でエンコードされた文字列をデコードして結果を返す
HEX()	小数または文字列値の 16 進数表現
INSERT()	指定した位置に指定した文字数まで部分文字列を挿入
INSTR()	部分文字列が最初に出現する位置のインデックスを返します

名前	説明
LCASE()	LOWER() のシノニムです
LEFT()	左端から指定された数の文字を返します
LENGTH()	文字列の長さをバイト単位で返します
LIKE	単純なパターン一致
LOAD_FILE()	指定されたファイルをロードします
LOCATE()	部分文字列が最初に出現する位置を返します
LOWER()	引数を小文字で返します
LPAD()	指定された文字列で左からパディングした文字列引数を返します
LTRIM()	先頭の空白を削除します
MAKE_SET()	bits セット内の対応するビットを持つ、カンマ区切り文字列のセットを返します
MATCH	全文検索を実行します
MID()	指定された位置から始まる部分文字列を返します
NOT LIKE	単純なパターン一致の否定
NOT REGEXP	REGEXP の否定
OCT()	数値の 8 進数表現を含む文字列を返します
OCTET_LENGTH()	LENGTH() のシノニムです
ORD()	引数の左端の文字の文字コードを返します
POSITION()	LOCATE() のシノニムです
QUOTE()	SQL ステートメント内で使用するために引数をエスケープします
REGEXP	文字列が正規表現と一致するかどうか
REGEXP_INSTR()	正規表現に一致する部分文字列の開始インデックス
REGEXP_LIKE()	文字列が正規表現と一致するかどうか
REGEXP_REPLACE()	正規表現に一致する部分文字列の置換
REGEXP_SUBSTR()	正規表現に一致する部分文字列を返します
REPEAT()	文字列を指定された回数だけ繰り返し返します
REPLACE()	指定された文字列の出現箇所を置き換えます
REVERSE()	文字列内の文字を逆順に並べ替えます
RIGHT()	右端から指定された数の文字を返します
RLIKE	文字列が正規表現と一致するかどうか
RPAD()	指定された回数だけ文字列を追加します
RTRIM()	末尾の空白を削除します
SOUNDEX()	soundex 文字列を返します
SOUNDS LIKE	音声を比較します
SPACE()	指定された数の空白で構成される文字列を返します
STRCMP()	2 つの文字列を比較します
SUBSTR()	指定された部分文字列を返します
SUBSTRING()	指定された部分文字列を返します
SUBSTRING_INDEX()	文字列から、区切り文字が指定された回数出現する前の部分文字列を返します

名前	説明
TO_BASE64()	base 64 文字列に変換された引数を返します
TRIM()	先頭と末尾にある空白を削除します
UCASE()	UPPER() のシノニムです
UNHEX()	数値の 16 進数表現を含む文字列を返します
UPPER()	大文字に変換します
WEIGHT_STRING()	文字列の重み文字列を返します

文字列値の関数は、結果の長さが `max_allowed_packet` システム環境変数の値よりも長くなると、`NULL` を返します。[セクション5.1.1「サーバーの構成」](#)を参照してください。

文字列の位置を操作する関数では、最初の位置には数値 1 が付けられます。

長さの引数を取る関数では、整数以外の引数はもっとも近い整数に丸められます。

- [ASCII\(str\)](#)

文字列 `str` の左端の文字の数値を返します。 `str` が空の文字列である場合は、0 を返します。 `str` が `NULL` である場合は `NULL` を返します。 [ASCII\(\)](#) は、8 ビット文字の場合に動作します。

```
mysql> SELECT ASCII('2');
-> 50
mysql> SELECT ASCII(2);
-> 50
mysql> SELECT ASCII('dx');
-> 100
```

[ORD\(\)](#) 関数も参照してください。

- [BIN\(N\)](#)

`N` のバイナリ値の文字列表現を返します。 `N` は `longlong (BIGINT)` 数字です。これは、[CONV\(N,10,2\)](#) と同等です。 `N` が `NULL` である場合は `NULL` を返します。

```
mysql> SELECT BIN(12);
-> '1100'
```

- [BIT_LENGTH\(str\)](#)

文字列 `str` の長さをビット単位で返します。

```
mysql> SELECT BIT_LENGTH('text');
-> 32
```

- [CHAR\(N,... \[USING charset_name\]\)](#)

[CHAR\(\)](#) は各 `N` 引数を整数として解釈し、それらの整数のコード値で指定された文字を構成している文字列を返します。 `NULL` 値はスキップされます。

```
mysql> SELECT CHAR(77,121,83,81,'76');
-> 'MySQL'
mysql> SELECT CHAR(77,77.3,'77.3');
-> 'MMM'
```

255 よりも大きい [CHAR\(\)](#) 引数は、複数の結果バイトに変換されます。たとえば、[CHAR\(256\)](#) は [CHAR\(1,0\)](#) に同等で、[CHAR\(256*256\)](#) は [CHAR\(1,0,0\)](#) に同等です。

```
mysql> SELECT HEX(CHAR(1,0)), HEX(CHAR(256));
+-----+-----+
| HEX(CHAR(1,0)) | HEX(CHAR(256)) |
+-----+-----+
| 0100          | 0100          |
+-----+-----+
mysql> SELECT HEX(CHAR(1,0,0)), HEX(CHAR(256*256));
+-----+-----+
| HEX(CHAR(1,0,0)) | HEX(CHAR(256*256)) |
+-----+-----+
```

```
| HEX(CHAR(1,0,0)) | HEX(CHAR(256*256)) |
+-----+-----+
| 010000          | 010000          |
+-----+-----+
```

デフォルトでは、`CHAR()` はバイナリ文字列を返します。指定された文字セットで文字列を生成するには、オプションの `USING` 句を使用します。

```
mysql> SELECT CHARSET(CHAR(X'65')), CHARSET(CHAR(X'65' USING utf8));
+-----+-----+
| CHARSET(CHAR(X'65')) | CHARSET(CHAR(X'65' USING utf8)) |
+-----+-----+
| binary              | utf8                              |
+-----+-----+
```

`USING` が指定され、結果文字列が指定された文字セットで不正である場合は、警告が発行されます。また、厳密な SQL モードが有効になっている場合は、`CHAR()` からの結果は `NULL` になります。

- `CHAR_LENGTH(str)`

文字で測定された文字列 `str` の長さを返します。マルチバイト文字は、単一の文字としてカウントされます。つまり、5 つの 2 バイト文字を含む文字列では、`LENGTH()` は 10 を返し、`CHAR_LENGTH()` は 5 を返します。

- `CHARACTER_LENGTH(str)`

`CHARACTER_LENGTH()` は `CHAR_LENGTH()` のシノニムです。

- `CONCAT(str1,str2,...)`

引数を連結することで生成される文字列を返します。1 つ以上の引数を持つ場合があります。すべての引数が非バイナリ文字列の場合は、結果も非バイナリ文字列になります。引数にバイナリ文字列が含まれる場合は、結果はバイナリ文字列になります。数値の引数は、同等の非バイナリ文字列形式に変換されます。

引数のいずれかが `NULL` である場合、`CONCAT()` は `NULL` を返します。

```
mysql> SELECT CONCAT('My', 'S', 'QL');
-> 'MySQL'
mysql> SELECT CONCAT('My', NULL, 'QL');
-> NULL
mysql> SELECT CONCAT(14.3);
-> '14.3'
```

引用符で囲まれた文字列では、文字列を並べて配置することで連結が実行されます。

```
mysql> SELECT 'My' 'S' 'QL';
-> 'MySQL'
```

- `CONCAT_WS(separator,str1,str2,...)`

`CONCAT_WS()` は Concatenate With Separator (区切り文字を使用した連結) を表し、`CONCAT()` の特殊な形式です。最初の引数は、残りの引数の区切り文字です。区切り文字は、連結される文字列の間に追加されます。区切り文字は、残りの引数と同様に文字列にすることができます。区切り文字が `NULL` の場合は、結果も `NULL` になります。

```
mysql> SELECT CONCAT_WS(',', 'First name', 'Second name', 'Last Name');
-> 'First name,Second name,Last Name'
mysql> SELECT CONCAT_WS(',', 'First name', NULL, 'Last Name');
-> 'First name,Last Name'
```

`CONCAT_WS()` では、空の文字列がスキップされません。ただし、区切り文字引数のあとの `NULL` 値はすべてスキップされます。

- `ELT(N,str1,str2,str3,...)`

`ELT()` は、文字列リストの `N` 番目の要素を返します。`N = 1` の場合は `str1`、`N = 2` の場合は `str2` のように返します。`N` が 1 よりも小さいか、引数の数よりも大きい場合は、`NULL` を返します。`ELT()` は `FIELD()` の補数です。

```
mysql> SELECT ELT(1, 'Aa', 'Bb', 'Cc', 'Dd');
```

```

-> 'Aa'
mysql> SELECT ELT(4, 'Aa', 'Bb', 'Cc', 'Dd');
-> 'Dd'

```

- **EXPORT_SET(bits,on,off,separator[,number_of_bits])**

値 **bits** 内で各ビットが設定されている場合には **on** 文字列を取得し、値内で各ビットが設定されていない場合には **off** 文字列を取得するように、文字列を返します。 **bits** のビットは、右から左 (下位ビットから上位ビット) へと検証されます。文字列は、**separator** 文字列 (デフォルトはカンマ文字 ,) で区切られた結果に左から右へと追加されます。検証されるビット数は、**number_of_bits** で指定されます。指定されない場合のデフォルトは 64 です。 **number_of_bits** が 64 よりも大きい場合は、警告なしで 64 に短縮されます。符号なし整数として処理されるため、値 **-1** は実際には 64 と同じです。

```

mysql> SELECT EXPORT_SET(5,'Y','N',';',4);
-> 'Y,N,Y,N'
mysql> SELECT EXPORT_SET(6,'1','0',';',10);
-> '0,1,1,0,0,0,0,0,0,0'

```

- **FIELD(str,str1,str2,str3,...)**

str1、**str2**、**str3**、... リスト内で **str** のインデックス (位置) を返します。 **str** が見つからない場合は、**0** を返します。

FIELD() へのすべての引数が文字列の場合は、すべての引数が文字列として比較されます。すべての引数が数値の場合は、数字として比較されます。それ以外の場合は、引数が倍精度として比較されます。

NULL ではどの値との等価比較にも失敗するため、**str** が **NULL** である場合は、戻り値が **0** になります。 **FIELD()** は **ELT()** の補数です。

```

mysql> SELECT FIELD('Bb', 'Aa', 'Bb', 'Cc', 'Dd', 'Ff');
-> 2
mysql> SELECT FIELD('Gg', 'Aa', 'Bb', 'Cc', 'Dd', 'Ff');
-> 0

```

- **FIND_IN_SET(str, strlist)**

文字列 **str** が **N** 部分文字列で構成される文字列リスト **strlist** 内にある場合は、1 から **N** までの範囲内の値を返します。文字列リストは、, 文字で区切られた部分文字列で構成された文字列です。最初の引数が定数文字列で、2 番目が **SET** 型のカラムの場合、**FIND_IN_SET()** 関数はビット演算を使用するために最適化されます。 **str** が **strlist** 内がない場合、または **strlist** が空の文字列の場合は、**0** を返します。引数のいずれかが **NULL** である場合は、**NULL** を返します。最初の引数にカンマ (,) 文字が含まれる場合は、この関数が正しく動作しません。

```

mysql> SELECT FIND_IN_SET('b','a,b,c,d');
-> 2

```

- **FORMAT(X,D[,locale])**

数値 **X** を '#,###,###.##' のような書式に変換し、小数点第 **D** 位に丸めて、その結果を文字列として返します。 **D** が **0** の場合は、結果に小数点または小数部が含まれません。

オプションの 3 番目のパラメータを使用すると、結果数の小数点、3 桁の区切り文字、および区切り文字間のグループ化に使用されるロケールを指定できます。許可されるロケール値は、**lc_time_names** システム変数の有効な値と同じです (セクション 10.16 「MySQL Server のロケールサポート」を参照してください)。ロケールが指定されていない場合のデフォルトは、**'en_US'** です。

```

mysql> SELECT FORMAT(12332.123456, 4);
-> '12,332.1235'
mysql> SELECT FORMAT(12332.1,4);
-> '12,332.1000'
mysql> SELECT FORMAT(12332.2,0);
-> '12,332'
mysql> SELECT FORMAT(12332.2,2,'de_DE');
-> '12.332,20'

```

- **FROM_BASE64(str)**

TO_BASE64() で使用される base-64 でエンコードされたルールでエンコードされた文字列が指定され、デコードされた結果をバイナリ文字列として返します。引数が **NULL** の場合または有効な base-64 文字列でない場合は、結

果が `NULL` になります。ルールのエンコードおよびデコードについての詳細は、[TO_BASE64\(\)](#) の説明を参照してください。

```
mysql> SELECT TO_BASE64('abc'), FROM_BASE64(TO_BASE64('abc'));
-> 'JWJj', 'abc'
```

- [HEX\(str\)](#), [HEX\(N\)](#)

文字列の引数 `str` では、[HEX\(\)](#) は `str` の 16 進数文字列表現を返します。`str` 内の各文字の各バイトは、2 つの 16 進数字に変換されます。(したがって、マルチバイト文字は 2 桁よりも大きくなります。) この演算の逆は、[UNHEX\(\)](#) 関数で実行されます。

数値の引数 `N` では、[HEX\(\)](#) は、longlong ([BIGINT](#)) 数字として処理される `N` の 16 進数文字列表現を返します。これは、[CONV\(N,10,16\)](#) と同等です。この演算の逆は、[CONV\(HEX\(N\),16,10\)](#) で実行されます。

```
mysql> SELECT X'616263', HEX('abc'), UNHEX(HEX('abc'));
-> 'abc', 616263, 'abc'
mysql> SELECT HEX(255), CONV(HEX(255),16,10);
-> 'FF', 255
```

- [INSERT\(str,pos,len,newstr\)](#)

位置 `pos` で始まる部分文字列と、文字列 `newstr` で置換された `len` 文字長とともに、文字列 `str` を返します。`pos` が文字列の長さに収まらない場合は、元の文字列を返します。`len` が残りの文字列の長さに収まらない場合は、位置 `pos` からの残りの文字列を置換します。引数のいずれかが `NULL` である場合は、`NULL` を返します。

```
mysql> SELECT INSERT('Quadratic', 3, 4, 'What');
-> 'QuWhattic'
mysql> SELECT INSERT('Quadratic', -1, 4, 'What');
-> 'Quadratic'
mysql> SELECT INSERT('Quadratic', 3, 100, 'What');
-> 'QuWhat'
```

この関数はマルチバイトセーフです。

- [INSTR\(str,substr\)](#)

文字列 `str` 内で部分文字列 `substr` が最初に出現する位置を返します。これは、引数の順序が逆になる点を除いて、2 つの引数形式の [LOCATE\(\)](#) と同じです。

```
mysql> SELECT INSTR('foobarbar', 'bar');
-> 4
mysql> SELECT INSTR('xbar', 'foobar');
-> 0
```

この関数はマルチバイトセーフであり、1 つ以上の引数がバイナリ文字列である場合にのみ大文字と小文字が区別されます。

- [LCASE\(str\)](#)

[LCASE\(\)](#) は [LOWER\(\)](#) のシノニムです。

ビューで使用される [LCASE\(\)](#) は、ビュー定義の格納時に [LOWER\(\)](#) としてリライトされます。(Bug #12844279)

- [LEFT\(str,len\)](#)

文字列 `str` から左端の `len` 文字を返し、引数が `NULL` である場合は `NULL` を返します。

```
mysql> SELECT LEFT('foobarbar', 5);
-> 'fooba'
```

この関数はマルチバイトセーフです。

- **LENGTH(str)**

バイトで測定された文字列 `str` の長さを返します。マルチバイト文字は、複数のバイトとしてカウントされます。つまり、5 つの 2 バイト文字を含む文字列では、`LENGTH()` は `10` を返し、`CHAR_LENGTH()` は `5` を返します。

```
mysql> SELECT LENGTH('text');
-> 4
```

注記

MySQL では、`Length()` OpenGIS 空間関数の名前は `ST_Length()` です。

- **LOAD_FILE(file_name)**

ファイルを読み取り、ファイルの内容を文字列として返します。この関数を使用するには、ファイルがサーバーホストに配置されている必要があり、ファイルへのフルパス名を指定し、`FILE` 権限を持つ必要があります。ファイルは、サーバーで読取り可能で、そのサイズが `max_allowed_packet` バイト未満である必要があります。`secure_file_priv` システム変数が空でないディレクトリ名に設定されている場合は、そのディレクトリ内にロード対象のファイルが配置されている必要があります。(MySQL 8.0.17 より前では、ファイルはサーバーから読み取り可能であるだけでなく、すべてから読み取り可能である必要があります。)

ファイルが存在しない場合、または上記の条件が満たされていないために、ファイルを読み取ることができない場合、この関数は `NULL` を返します。

`character_set_filesystem` システム変数では、リテラル文字列として指定されているファイル名の解釈が制御されず。

```
mysql> UPDATE t
      SET blob_col=LOAD_FILE('/tmp/picture')
      WHERE id=1;
```

- **LOCATE(substr,str), LOCATE(substr,str,pos)**

1 番目の構文は、文字列 `str` 内で、部分文字列 `substr` が最初に出現する位置を返します。2 番目の構文は、文字列 `str` 内の位置 `pos` 以降で、部分文字列 `substr` が最初に出現する位置を返します。`str` 内に `substr` がない場合は、`0` を返します。引数のいずれかが `NULL` である場合は、`NULL` を返します。

```
mysql> SELECT LOCATE('bar', 'foobarbar');
-> 4
mysql> SELECT LOCATE('xbar', 'foobar');
-> 0
mysql> SELECT LOCATE('bar', 'foobarbar', 5);
-> 7
```

この関数はマルチバイトセーフであり、1 つ以上の引数がバイナリ文字列である場合にのみ大文字と小文字が区別されます。

- **LOWER(str)**

現在の文字セットのマッピングに従って、すべての文字が小文字に変更された文字列 `str` を返します。デフォルトは `utf8mb4` です。

```
mysql> SELECT LOWER('QUADRATICALLY');
-> 'quadratically'
```

`LOWER()` (および `UPPER()`) をバイナリ文字列 (`BINARY`、`VARBINARY`、`BLOB`) に適用しても、何の効果もありません。バイナリ文字列の大文字と小文字の変換を実行するには、まず文字列に格納されているデータに適した文字セットを使用して、非バイナリ文字列に変換します:

```
mysql> SET @str = BINARY 'New York';
mysql> SELECT LOWER(@str), LOWER(CONVERT(@str USING utf8mb4));
+-----+-----+
| LOWER(@str) | LOWER(CONVERT(@str USING utf8mb4)) |
+-----+-----+
| New York   | new york                           |
```

Unicode 文字セットの照合の場合、`LOWER()` および `UPPER()` は、照合名の Unicode 照合アルゴリズム (UCA) バージョン (存在する場合) および UCA 4.0.0 (バージョンが指定されていない場合) に従って動作します。たとえば、`utf8mb4_0900_ai_ci` と `utf8_unicode_520_ci` はそれぞれ UCA 9.0.0 と 5.2.0 に従って動作し、`utf8_unicode_ci` は UCA 4.0.0 に従って動作します。セクション10.10.1「Unicode 文字セット」を参照してください。

この関数はマルチバイトセーフです。

ビュー内で使用される `LCASE()` は、`LOWER()` としてリライトされます。

- `LPAD(str,len,padstr)`

`len` 文字の長さになるように文字列 `padstr` で左にパディングされた文字列 `str` を返します。 `str` が `len` よりも長い場合は、戻り値は `len` 文字に短縮されます。

```
mysql> SELECT LPAD('hi',4,'?');
-> '??hi'
mysql> SELECT LPAD('hi',1,'?');
-> 'h'
```

- `LTRIM(str)`

先頭の空白文字が削除された文字列 `str` を返します。

```
mysql> SELECT LTRIM(' barbar');
-> 'barbar'
```

この関数はマルチバイトセーフです。

- `MAKE_SET(bits,str1,str2,...)`

`bits` セット内の対応するビットを持つ文字列で構成されるセット値 (、文字で区切られた部分文字列を含む文字列) を返します。 `str1` はビット 0 に対応し、`str2` はビット 1 に対応する、などとなります。 `str1`、`str2`、... 内の `NULL` 値は結果に追加されません。

```
mysql> SELECT MAKE_SET(1,'a','b','c');
-> 'a'
mysql> SELECT MAKE_SET(1 | 4,'hello','nice','world');
-> 'hello,world'
mysql> SELECT MAKE_SET(1 | 4,'hello','nice',NULL,'world');
-> 'hello'
mysql> SELECT MAKE_SET(0,'a','b','c');
-> ''
```

- `MID(str,pos,len)`

`MID(str,pos,len)` は、`SUBSTRING(str,pos,len)` のシノニムです。

- `OCT(N)`

`N` の 8 進数の文字列表現を返します。 `N` は `longlong (BIGINT)` 数字です。これは、`CONV(N,10,8)` と同等です。 `N` が `NULL` である場合は `NULL` を返します。

```
mysql> SELECT OCT(12);
-> '14'
```

- `OCTET_LENGTH(str)`

`OCTET_LENGTH()` は `LENGTH()` のシノニムです。

- `ORD(str)`

文字列 `str` の左端の文字がマルチバイト文字である場合は、その文字のコードを返します。コードは、次の計算式を使用して、その構成要素の数値から計算されます。

```
(1st byte code)
+ (2nd byte code * 256)
```

```
+ (3rd byte code * 256^2) ...
```

左端の文字がマルチバイト文字でない場合は、`ORD()` は `ASCII()` 関数と同じ値を返します。

```
mysql> SELECT ORD('2');
-> 50
```

- `POSITION(substr IN str)`

`POSITION(substr IN str)` は `LOCATE(substr,str)` のシノニムです。

- `QUOTE(str)`

SQL ステートメントで、適切にエスケープされたデータ値として使用できる結果を生成する文字列を引用符で囲みます。文字列は一重引用符で囲まれ、バックスラッシュ (`\`)、一重引用符 (`'`)、ASCII NUL および Control+Z の各インスタンスの前にバックスラッシュが付いて返されます。引数が `NULL` の場合の戻り値は、一重引用符で囲まれていない単語「`NULL`」です。

```
mysql> SELECT QUOTE('Don\'t!');
-> 'Don\'t!'
mysql> SELECT QUOTE(NULL);
-> NULL
```

比較するために、[セクション9.1.1「文字列リテラル」](#) および `mysql_real_escape_string_quote()` で、リテラル文字列に対する引用ルールと C API 内の引用ルールを参照してください。

- `REPEAT(str,count)`

`count` 回繰り返された文字列 `str` で構成される文字列を返します。`count` が 1 よりも小さい場合は、空の文字列を返します。`str` または `count` が `NULL` である場合は `NULL` を返します。

```
mysql> SELECT REPEAT('MySQL', 3);
-> 'MySQLMySQLMySQL'
```

- `REPLACE(str,from_str,to_str)`

文字列 `from_str` のすべての出現箇所が文字列 `to_str` で置換された、文字列 `str` を返します。`REPLACE()` は、`from_str` を検索する際に、大文字と小文字を区別した一致を実行します。

```
mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
-> 'WwWwWw.mysql.com'
```

この関数はマルチバイトセーフです。

- `REVERSE(str)`

文字の順序が逆になった文字列 `str` を返します。

```
mysql> SELECT REVERSE('abc');
-> 'cba'
```

この関数はマルチバイトセーフです。

- `RIGHT(str,len)`

文字列 `str` から右端の `len` 文字を返し、引数が `NULL` である場合は `NULL` を返します。

```
mysql> SELECT RIGHT('foobarbar', 4);
-> 'rbar'
```

この関数はマルチバイトセーフです。

- **RPAD(str,len,padstr)**

len 文字の長さになるように文字列 **padstr** で右にパディングされた文字列 **str** を返します。 **str** が **len** よりも長い場合は、戻り値は **len** 文字に短縮されます。

```
mysql> SELECT RPAD('hi',5,'?');
-> 'hi???'
mysql> SELECT RPAD('hi',1,'?');
-> 'h'
```

この関数はマルチバイトセーフです。

- **RTRIM(str)**

末尾の空白文字が削除された文字列 **str** を返します。

```
mysql> SELECT RTRIM('barbar ');
-> 'barbar'
```

この関数はマルチバイトセーフです。

- **SOUNDEX(str)**

str から soundex 文字列を返します。ほぼ同じ発音の2つの文字列は、同じ soundex 文字列を持つはずですが、標準の soundex 文字列の長さは4文字ですが、**SOUNDEX()** 関数は任意の長さの文字列を返します。結果で **SUBSTRING()** を使用すると、標準の soundex 文字列を取得できます。 **str** 内のアルファベット以外の文字はすべて無視されます。A から Z までの範囲外の国際アルファベット文字はすべて、母音として処理されます。

重要

SOUNDEX() の使用時には、次の制限に注意してください。

- 現在実装されているこの関数は、文字列の言語が英語である場合にのみ機能するように設計されています。その他の言語の文字列では、信頼できる結果が生成されない可能性があります。
- この関数では、文字列でマルチバイト文字セット (**utf-8** など) が使用されている場合に、整合性のある結果が生成されることは保証されません。詳細は、Bug #22638 を参照してください。

```
mysql> SELECT SOUNDEX('Hello');
-> 'H400'
mysql> SELECT SOUNDEX('Quadratically');
-> 'Q36324'
```

注記

この関数には、オリジナルの Soundex アルゴリズムが実装されています。より人気のある拡張バージョンではありません (この作成者も D. Knuth です)。相違点としては、元のバージョンではまず母音が破棄されてから複製が破棄されますが、拡張バージョンではまず複製が破棄されてから母音が破棄されます。

- **expr1 SOUNDS LIKE expr2**

これは、**SOUNDEX(expr1) = SOUNDEX(expr2)** と同じです。

- **SPACE(N)**

N 空白文字で構成される文字列を返します。

```
mysql> SELECT SPACE(6);
-> '     '
```

- **SUBSTR(str,pos), SUBSTR(str FROM pos), SUBSTR(str,pos,len), SUBSTR(str FROM pos FOR len)**

SUBSTR() は **SUBSTRING()** のシノニムです。

- **SUBSTRING(str,pos), SUBSTRING(str FROM pos), SUBSTRING(str,pos,len), SUBSTRING(str FROM pos FOR len)**

`len` 引数を付けない形式では、位置 `pos` で始まる文字列 `str` からの部分文字列が返されます。 `len` 引数を付けた形式では、位置 `pos` で始まる文字列 `str` からの部分文字列 `len` 文字長が返されます。 `FROM` を使用する形式は、標準の SQL 構文です。また、`pos` に負の値を使用することもできます。その場合、部分文字列の先頭は文字列の先頭でなく、文字列の末尾からの `pos` 文字になります。この関数のどの形式でも、`pos` で負の値を使用できます。 `pos` の値が 0 の場合、空の文字列が返されます。

すべての形式の `SUBSTRING()` で、部分文字列の抽出が開始される文字列内の最初の文字の位置が 1 とみなされます。

```
mysql> SELECT SUBSTRING('Quadratically',5);
-> 'ratically'
mysql> SELECT SUBSTRING('foobarbar' FROM 4);
-> 'barbar'
mysql> SELECT SUBSTRING('Quadratically',5,6);
-> 'ratica'
mysql> SELECT SUBSTRING('Sakila', -3);
-> 'ila'
mysql> SELECT SUBSTRING('Sakila', -5, 3);
-> 'aki'
mysql> SELECT SUBSTRING('Sakila' FROM -4 FOR 2);
-> 'ki'
```

この関数はマルチバイトセーフです。

`len` が 1 よりも小さい場合は、結果が空の文字列になります。

- `SUBSTRING_INDEX(str,delim,count)`

文字列 `str` から、区切り文字 `delim` が `count` 回出現する前の部分文字列を返します。 `count` が正の値の場合は、(左から数えて)最後の区切り文字の左側にあるすべてが返されます。 `count` が負の値の場合は、(右から数えて)最後の区切り文字の右側にあるすべてが返されます。 `SUBSTRING_INDEX()` は、`delim` を検索する際に、大文字と小文字を区別した一致を実行します。

```
mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', 2);
-> 'www.mysql'
mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', -2);
-> 'mysql.com'
```

この関数はマルチバイトセーフです。

- `TO_BASE64(str)`

文字列の引数を base-64 でエンコードされた形式に変換し、その結果を接続文字セットと照合順序が含まれる文字列として返します。引数が文字列でない場合は、変換が実行される前に文字列に変換されます。引数が `NULL` である場合は、結果も `NULL` になります。 `FROM_BASE64()` 関数を使用すると、base-64 でエンコードされた文字列をデコードできます。

```
mysql> SELECT TO_BASE64('abc'), FROM_BASE64(TO_BASE64('abc'));
-> 'JWJj', 'abc'
```

さまざまな base-64 エンコードスキームが存在します。これらは、`TO_BASE64()` および `FROM_BASE64()` で使用されるエンコードおよびデコードのルールです。

- アルファベット値 62 のエンコードは '+' です。
- アルファベット値 63 のエンコードは '/' です。
- エンコードされた出力は、出力可能な 4 文字のグループで構成されます。入力データの各 3 バイトは、4 文字を使用してエンコードされます。最後のグループが不完全な場合は、長さが 4 になるまで '=' 文字でパディングされます。
- 長い出力を複数の行に分割するために、エンコードされた出力の各 76 文字の後ろに改行が追加されます。
- デコードでは改行、復帰改行、タブ、および空白が認識および無視されます。

- `TRIM([BOTH | LEADING | TRAILING] [remstr] FROM] str)`, `TRIM([remstr FROM] str)`

すべての `remstr` プリフィクスまたはサフィクスが削除された文字列 `str` を返します。
BOTH、**LEADING**、**TRAILING** のいずれの指定子も指定されない場合は、**BOTH** が指定されたとみなされます。
`remstr` はオプションであり、指定されない場合は空白文字が削除されます。

```
mysql> SELECT TRIM(' bar ');
-> 'bar'
mysql> SELECT TRIM(LEADING 'x' FROM 'xxxbarxxx');
-> 'barxxx'
mysql> SELECT TRIM(BOTH 'x' FROM 'xxxbarxxx');
-> 'bar'
mysql> SELECT TRIM(TRAILING 'xyz' FROM 'barxyz');
-> 'barx'
```

この関数はマルチバイトセーフです。

- **UCASE(str)**

`UCASE()` は `UPPER()` のシノニムです。

ビュー内で使用される `UCASE()` は、`UPPER()` としてリライトされます。

- **UNHEX(str)**

文字列の引数 `str` では、`UNHEX(str)` は引数の各文字ペアを 16 進数として解釈し、その数字で表されたバイトに変換します。戻り値はバイナリ文字列です。

```
mysql> SELECT UNHEX('4D7953514C');
-> 'MySQL'
mysql> SELECT X'4D7953514C';
-> 'MySQL'
mysql> SELECT UNHEX(HEX('string'));
-> 'string'
mysql> SELECT HEX(UNHEX('1267'));
-> '1267'
```

引数文字列内の文字は、正当な 16 進数である必要があります: '0' .. '9'、'A' .. 'F'、'a' .. 'f'。引数に 16 進以外の数字が含まれている場合は、結果が **NULL** になります。

```
mysql> SELECT UNHEX('GG');
+-----+
| UNHEX('GG') |
+-----+
| NULL       |
+-----+
```

NULL の結果は、`UNHEX()` への引数が **BINARY** カラムの場合に発生することがあります。これは、格納時に値に `0x00` バイトが埋め込まれても、それらのバイトは取得時に削除されないためです。たとえば、'41'は'41'として **CHAR(3)** カラムに格納され、'41'として取得されるため (末尾の空白が削除されます)、カラム値の `UNHEX()` は `X'41'` を返します。対照的に、'41'は'41\0'として **BINARY(3)** カラムに格納され、'41\0'として取得されます (後続パッド `0x00` バイトは削除されません)。`\0`は有効な 16 進数ではないため、カラム値の `UNHEX()` は **NULL** を返します。

数値の引数 `N` の場合、`HEX(N)` の逆は `UNHEX()` では実行されません。代わりに、`CONV(HEX(N),16,10)` を使用してください。`HEX()` の説明を参照してください。

- **UPPER(str)**

現在の文字セットのマッピングに従って、すべての文字が大文字に変更された文字列 `str` を返します。デフォルトは `utf8mb4` です。

```
mysql> SELECT UPPER('Hej');
```



```
-> 'HEJ'
```

UPPER() にも適用される情報については、LOWER() の説明を参照してください。これには、現在は機能が無効になっているバイナリ文字列 (BINARY、VARBINARY、BLOB) の大文字と小文字の変換を実行する方法に関する情報、および Unicode 文字セットの大文字と小文字の変換に関する情報が含まれていました。

この関数はマルチバイトセーフです。

ビュー内で使用される UCASE() は、UPPER() としてリライトされます。

- **WEIGHT_STRING(str [AS {CHAR|BINARY}(N)] [flags])**

この関数は、入力文字列の重み文字列を返します。戻り値は、文字列の比較およびソート値を表すバイナリ文字列です。これらのプロパティがあります。

- **WEIGHT_STRING(str1) = WEIGHT_STRING(str2)** の場合は、**str1 = str2** です (str1 と str2 は等しいとみなされます)。
- **WEIGHT_STRING(str1) < WEIGHT_STRING(str2)** の場合は、**str1 < str2** です (str1 は str2 の前にソートされます)。

WEIGHT_STRING() は、内部使用を目的としたデバッグ機能です。その動作は、MySQL のバージョン間で予告なしに変更できます。これは、特に新しい照合を追加する場合に、照合のテストおよびデバッグに使用できます。 [セクション10.14「文字セットへの照合順序の追加」](#) を参照してください。

このリストには、引数が簡単にまとめられています。詳細は、リストの後の説明を参照してください。

- **str**: 入力文字列式。
- **AS** 句: オプション。入力文字列を指定された型と長さにキャストします。
- **flags**: Optional; unused.

入力文字列 **str** は文字列式です。入力が非バイナリ (文字) 文字列 (CHAR、VARCHAR、TEXT 値など) である場合は、戻り値に文字列の照合順序重みが含まれます。入力がバイナリ (バイト) 文字列 (BINARY、VARBINARY、BLOB 値など) である場合は、戻り値は入力と同じです (バイナリ文字列のバイトごとの重みはバイト値です)。入力が NULL である場合は、WEIGHT_STRING() は NULL を返します。

例:

```
mysql> SET @s = _utf8mb4 'AB' COLLATE utf8mb4_0900_ai_ci;
mysql> SELECT @s, HEX(@s), HEX(WEIGHT_STRING(@s));
+-----+-----+-----+
| @s   | HEX(@s) | HEX(WEIGHT_STRING(@s)) |
+-----+-----+-----+
| AB   | 4142    | 1C471C60                |
+-----+-----+-----+
```

```
mysql> SET @s = _utf8mb4 'ab' COLLATE utf8mb4_0900_ai_ci;
mysql> SELECT @s, HEX(@s), HEX(WEIGHT_STRING(@s));
+-----+-----+-----+
| @s   | HEX(@s) | HEX(WEIGHT_STRING(@s)) |
+-----+-----+-----+
| ab   | 6162    | 1C471C60                |
+-----+-----+-----+
```

```
mysql> SET @s = CAST('AB' AS BINARY);
mysql> SELECT @s, HEX(@s), HEX(WEIGHT_STRING(@s));
+-----+-----+-----+
| @s   | HEX(@s) | HEX(WEIGHT_STRING(@s)) |
+-----+-----+-----+
| AB   | 4142    | 4142                    |
+-----+-----+-----+
```

```
mysql> SET @s = CAST('ab' AS BINARY);
mysql> SELECT @s, HEX(@s), HEX(WEIGHT_STRING(@s));
+-----+-----+-----+
| @s   | HEX(@s) | HEX(WEIGHT_STRING(@s)) |
+-----+-----+-----+
```

```

+-----+-----+-----+
| ab | 6162 | 6162 |
+-----+-----+-----+

```

上記の例では、`HEX()` を使用して `WEIGHT_STRING()` の結果を表示しています。結果はバイナリ値であるため、結果に出力されない値が含まれるときに出力可能な形式で表示する際に、`HEX()` が特に役立ちます。

```

mysql> SET @s = CONVERT('C39F' USING utf8) COLLATE utf8_czech_ci;
mysql> SELECT HEX(WEIGHT_STRING(@s));
+-----+
| HEX(WEIGHT_STRING(@s)) |
+-----+
| 0FEA0FEA                |
+-----+

```

非 `NULL` の戻り値では、長さが `VARBINARY` の最大長内である場合は、値のデータ型が `VARBINARY` であり、その他の場合はデータ型は `BLOB` です。

入力文字列を非バイナリまたはバイナリの文字列にキャストし、強制的に指定した長さになるように、`AS` 句が指定されている場合があります。

- `AS CHAR(N)` は、文字列を非バイナリ文字列にキャストし、`N` 文字の長さになるように空白文字で右側をパディングします。`N` は少なくとも 1 にする必要があります。`N` が入力文字列の長さよりも小さい場合は、文字列が `N` 文字まで切り捨てられます。切り捨てられても警告は発生しません。
- `AS BINARY(N)` は、文字列がバイナリ文字列にキャストされ、`N` が (文字単位ではなく) バイト単位で測定され、パディングで (空白文字でなく) `0x00` バイトが使用される点を除いて同様です。

```

mysql> SET NAMES 'latin1';
mysql> SELECT HEX(WEIGHT_STRING('ab' AS CHAR(4)));
+-----+
| HEX(WEIGHT_STRING('ab' AS CHAR(4))) |
+-----+
| 41422020                            |
+-----+
mysql> SET NAMES 'utf8';
mysql> SELECT HEX(WEIGHT_STRING('ab' AS CHAR(4)));
+-----+
| HEX(WEIGHT_STRING('ab' AS CHAR(4))) |
+-----+
| 0041004200200020                    |
+-----+

```

```

mysql> SELECT HEX(WEIGHT_STRING('ab' AS BINARY(4)));
+-----+
| HEX(WEIGHT_STRING('ab' AS BINARY(4))) |
+-----+
| 61620000                            |
+-----+

```

現在、`flags` 句は使用されていません。

12.8.1 文字列比較関数および演算子

表 12.13 「文字列比較関数および演算子」

名前	説明
<code>LIKE</code>	単純なパターン一致
<code>NOT LIKE</code>	単純なパターン一致の否定
<code>STRCMP()</code>	2つの文字列を比較します

文字列関数に引数としてバイナリ文字列が指定されている場合は、結果の文字列もバイナリ文字列になります。文字列に変換された数字は、バイナリ文字列として処理されます。比較のみがこの影響を受けます。

通常、文字列比較のいずれかの式で大/小文字が区別される場合、比較は大/小文字を区別して実行されます。

- `expr LIKE pat [ESCAPE 'escape_char']`

SQL パターンを使用したパターン一致。1 (TRUE) または 0 (FALSE) を返します。expr または pat のいずれかが NULL である場合は、結果も NULL になります。

パターンはリテラル文字列である必要はありません。たとえば、文字列式やテーブルカラムとして指定できます。後者の場合、カラムは MySQL 文字列型のいずれかとして定義する必要があります (セクション11.3「文字列データ型」を参照)。

SQL 標準では、LIKE は文字ごとに一致を実行するため、= 比較演算子とは異なる結果が生成される可能性があります。

```
mysql> SELECT 'ä' LIKE 'ae' COLLATE latin1_german2_ci;
+-----+
| 'ä' LIKE 'ae' COLLATE latin1_german2_ci |
+-----+
|                                0 |
+-----+
mysql> SELECT 'ä' = 'ae' COLLATE latin1_german2_ci;
+-----+
| 'ä' = 'ae' COLLATE latin1_german2_ci |
+-----+
|                                1 |
+-----+
```

特に、末尾のスペースは常に意味を持ちます。これは、非バイナリ文字列 (CHAR、VARCHAR および TEXT 値) の末尾のスペースの意味が比較に使用される照合のパッド属性に依存する = 演算子で実行される比較とは異なります。詳細は、[比較での後続領域の処理](#)を参照してください。

LIKE では、パターンに次の 2 つのワイルドカード文字を使用できます:

- % は、ゼロ文字を含め、任意の数の文字に一致します。
- _ は、単一の文字と完全に一致します。

```
mysql> SELECT 'David!' LIKE 'David_';
-> 1
mysql> SELECT 'David!' LIKE '%D%v%';
-> 1
```

ワイルドカード文字のリテラルインスタンスをテストするには、その前にエスケープ文字を指定します。ESCAPE 文字を指定しない場合は、\ と仮定されます。

- \% は、いずれかの % 文字と一致します。
- _ は、いずれかの _ 文字と一致します。

```
mysql> SELECT 'David!' LIKE 'David\';
-> 0
mysql> SELECT 'David_' LIKE 'David\';
-> 1
```

別のエスケープ文字を指定するには、ESCAPE 句を使用します。

```
mysql> SELECT 'David_' LIKE 'David_' ESCAPE '!';
-> 1
```

エスケープシーケンスは空にするか、1 文字の長さにするようにしてください。実行時に、式は定数として評価される必要があります。NO_BACKSLASH_ESCAPES SQL モードが有効になっている場合は、シーケンスを空にできません。

次の 2 つのステートメントは、オペランドのいずれかが大文字と小文字を区別しないかぎり、文字列の比較で大文字と小文字が区別されないことを示しています (大文字と小文字が区別される照合順序を使用するか、バイナリ文字列です):

```
mysql> SELECT 'abc' LIKE 'ABC';
-> 1
mysql> SELECT 'abc' LIKE _utf8mb4 'ABC' COLLATE utf8mb4_0900_as_cs;
```

```
-> 0
mysql> SELECT 'abc' LIKE _utf8mb4 'ABC' COLLATE utf8mb4_bin;
-> 0
mysql> SELECT 'abc' LIKE BINARY 'ABC';
-> 0
```

標準 SQL の拡張機能として、MySQL では数値式に対する `LIKE` が許可されます。

```
mysql> SELECT 10 LIKE '1%';
-> 1
```

注記

MySQL では文字列で C エスケープ構文 (改行文字を表す `\n` など) が使用されるため、`LIKE` 文字列で使用する `\` をダブルクリックする必要があります。たとえば、`\n` を検索するには、`\\n` と指定します。`\` を検索するには、`\\\\` として指定します。これは、バックスラッシュがパーサーによって一度だけ取り除かれ、パターン一致が行われたときに再度取り除かれ、単一のバックスラッシュが照合対象のままになるためです。

例外: パターン文字列の末尾では、バックスラッシュを `\\` と指定できます。文字列の末尾では、エスケープの後ろに何もいないため、バックスラッシュはそれ自体を表します。テーブルに次の値が含まれると仮定します。

```
mysql> SELECT filename FROM t1;
+-----+
| filename |
+-----+
| C:       |
| C:\     |
| C:\Programs |
| C:\Programs\ |
+-----+
```

バックスラッシュで終わる値をテストするには、次のパターンのいずれかを使用すると値をマッチングできます。

```
mysql> SELECT filename, filename LIKE '%\\' FROM t1;
+-----+-----+
| filename | filename LIKE '%\\' |
+-----+-----+
| C:       | 0 |
| C:\     | 1 |
| C:\Programs | 0 |
| C:\Programs\ | 1 |
+-----+-----+
```

```
mysql> SELECT filename, filename LIKE '%\\\\' FROM t1;
+-----+-----+
| filename | filename LIKE '%\\\\' |
+-----+-----+
| C:       | 0 |
| C:\     | 1 |
| C:\Programs | 0 |
| C:\Programs\ | 1 |
+-----+-----+
```

- `expr NOT LIKE pat [ESCAPE 'escape_char']`

これは、`NOT (expr LIKE pat [ESCAPE 'escape_char'])` と同じです。

注記

`NULL` を含むカラムとの `NOT LIKE` 比較を伴う集計クエリーでは、予想外の結果が生成される可能性があります。たとえば、次のテーブルとデータを検討してください。

```
CREATE TABLE foo (bar VARCHAR(10));
```

```
INSERT INTO foo VALUES (NULL), (NULL);
```

クエリー `SELECT COUNT(*) FROM foo WHERE bar LIKE '%baz%'` は 0 を返します。
`SELECT COUNT(*) FROM foo WHERE bar NOT LIKE '%baz%'` は 2 を返すと想定しています。ただし、事実は異なります。2 番目のクエリーは 0 を返します。これは、`expr` の値に関係なく、`NULL NOT LIKE expr` は常に `NULL` を返すためです。`NULL` を伴う集計クエリー、および `NOT RLIKE` または `NOT REGEXP` を使用する比較でも、同じことが当てはまります。このような場合は、次に示すように、(`AND` ではなく) `OR` を使用して `NOT NULL` を明示的にテストする必要があります。

```
SELECT COUNT(*) FROM foo WHERE bar NOT LIKE '%baz%' OR bar IS NULL;
```

- `STRCMP(expr1,expr2)`

`STRCMP()` は、文字列が同じ場合は 0 を返し、現在のソート順に従って 1 番目の引数が 2 番目よりも小さい場合は -1、それ以外の場合は 1 を返します。

```
mysql> SELECT STRCMP('text', 'text2');
-> -1
mysql> SELECT STRCMP('text2', 'text');
-> 1
mysql> SELECT STRCMP('text', 'text');
-> 0
```

`STRCMP()` は、引数の照合順序を使用して比較を実行します。

```
mysql> SET @s1 = _utf8mb4 'x' COLLATE utf8mb4_0900_ai_ci;
mysql> SET @s2 = _utf8mb4 'X' COLLATE utf8mb4_0900_ai_ci;
mysql> SET @s3 = _utf8mb4 'x' COLLATE utf8mb4_0900_as_cs;
mysql> SET @s4 = _utf8mb4 'X' COLLATE utf8mb4_0900_as_cs;
mysql> SELECT STRCMP(@s1, @s2), STRCMP(@s3, @s4);
+-----+-----+
| STRCMP(@s1, @s2) | STRCMP(@s3, @s4) |
+-----+-----+
|          0 |          -1 |
+-----+-----+
```

照合順序の互換性がない場合は、その他との互換性を保つために、引数のいずれかを変換する必要があります。[セクション10.8.4「式での照合の強制性」](#)を参照してください。

```
mysql> SET @s1 = _utf8mb4 'x' COLLATE utf8mb4_0900_ai_ci;
mysql> SET @s2 = _utf8mb4 'X' COLLATE utf8mb4_0900_ai_ci;
mysql> SET @s3 = _utf8mb4 'x' COLLATE utf8mb4_0900_as_cs;
mysql> SET @s4 = _utf8mb4 'X' COLLATE utf8mb4_0900_as_cs;
-->
mysql> SELECT STRCMP(@s1, @s3);
ERROR 1267 (HY000): Illegal mix of collations (utf8mb4_0900_ai_ci,IMPLICIT)
and (utf8mb4_0900_as_cs,IMPLICIT) for operation 'strcmp'
mysql> SELECT STRCMP(@s1, @s3 COLLATE utf8mb4_0900_ai_ci);
+-----+
| STRCMP(@s1, @s3 COLLATE utf8mb4_0900_ai_ci) |
+-----+
|                      0 |
+-----+
```

12.8.2 正規表現

表 12.14 「正規表現関数および演算子」

名前	説明
<code>NOT REGEXP</code>	<code>REGEXP</code> の否定
<code>REGEXP</code>	文字列が正規表現と一致するかどうか
<code>REGEXP_INSTR()</code>	正規表現に一致する部分文字列の開始インデックス
<code>REGEXP_LIKE()</code>	文字列が正規表現と一致するかどうか
<code>REGEXP_REPLACE()</code>	正規表現に一致する部分文字列の置換

名前	説明
REGEXP_SUBSTR()	正規表現に一致する部分文字列を返します
RLIKE	文字列が正規表現と一致するかどうか

正規表現は、複雑な検索でパターンを指定する強力な方法です。このセクションでは、正規表現の照合に使用できる関数と演算子について説明し、正規表現の操作に使用できる特殊文字と構造の一部を例とともに示します。[セクション3.3.4.7「パターンマッチング」](#)も参照してください。

MySQL では、Unicode の国際コンポーネント (ICU) を使用した正規表現サポートが実装されています。ICU は完全な Unicode サポートを提供し、マルチバイトセーフです。(MySQL 8.0.4 より前では、MySQL は Henry Spencer による正規表現の実装を使用していました。これはバイト単位で動作し、マルチバイトセーフではありません。正規表現を使用するアプリケーションが実装の変更の影響を受ける方法の詳細は、[正規表現の互換性に関する考慮事項](#)を参照してください。)

- [正規表現関数および演算子](#)
- [正規表現構文](#)
- [正規表現のリソース制御](#)
- [正規表現の互換性に関する考慮事項](#)

正規表現関数および演算子

- `expr NOT REGEXP pat, expr NOT RLIKE pat`

これは、`NOT (expr REGEXP pat)` と同じです。

- `expr REGEXP pat, expr RLIKE pat`

文字列 `expr` がパターン `pat` で指定された正規表現と一致する場合は 1 を返し、一致しない場合は 0 を返します。`expr` または `pat` が `NULL` の場合、戻り値は `NULL` です。

`REGEXP` および `RLIKE` は、`REGEXP_LIKE()` のシノニムです。

照合の実行方法の詳細は、`REGEXP_LIKE()` の説明を参照してください。

```
mysql> SELECT 'Michael!' REGEXP '!*';
+-----+
| 'Michael!' REGEXP '!*' |
+-----+
| 1 |
+-----+
mysql> SELECT 'new*\n*line' REGEXP 'new\\.*\\*line';
+-----+
| 'new*\n*line' REGEXP 'new\\.*\\*line' |
+-----+
| 0 |
+-----+
mysql> SELECT 'a' REGEXP '^[a-d]';
+-----+
| 'a' REGEXP '^[a-d]' |
+-----+
| 1 |
+-----+
mysql> SELECT 'a' REGEXP 'A', 'a' REGEXP BINARY 'A';
+-----+-----+
| 'a' REGEXP 'A' | 'a' REGEXP BINARY 'A' |
+-----+-----+
| 1 | 0 |
+-----+-----+
```

- `REGEXP_INSTR(expr, pat[, pos[, occurrence[, return_option[, match_type]]])`

パターン `pat` で指定された正規表現に一致する文字列 `expr` の部分文字列の開始インデックスを返します。一致がない場合は 0 を返します。`expr` または `pat` が `NULL` の場合、戻り値は `NULL` です。文字インデックスは 1 から始まります。

`REGEXP_INSTR()` は、次のオプションの引数を取ります:

- `pos`: 検索を開始する `expr` 内の位置。省略した場合、デフォルトは 1 です。
- `occurrence`: 検索する一致のオカレンス。省略した場合、デフォルトは 1 です。
- `return_option`: 返す位置のタイプ。この値が 0 の場合、`REGEXP_INSTR()` は一致した部分文字列の最初の文字の位置を返します。この値が 1 の場合、`REGEXP_INSTR()` は一致した部分文字列の後の位置を返します。省略した場合、デフォルトは 0 です。
- `match_type`: 照合の実行方法を指定する文字列。意味は、`REGEXP_LIKE()` で説明されているとおりです。

照合の実行方法の詳細は、`REGEXP_LIKE()` の説明を参照してください。

```
mysql> SELECT REGEXP_INSTR('dog cat dog', 'dog');
+-----+
| REGEXP_INSTR('dog cat dog', 'dog') |
+-----+
| 1 |
+-----+
mysql> SELECT REGEXP_INSTR('dog cat dog', 'dog', 2);
+-----+
| REGEXP_INSTR('dog cat dog', 'dog', 2) |
+-----+
| 9 |
+-----+
mysql> SELECT REGEXP_INSTR('aa aaa aaaa', 'a{2}');
+-----+
| REGEXP_INSTR('aa aaa aaaa', 'a{2}') |
+-----+
| 1 |
+-----+
mysql> SELECT REGEXP_INSTR('aa aaa aaaa', 'a{4}');
+-----+
| REGEXP_INSTR('aa aaa aaaa', 'a{4}') |
+-----+
| 8 |
+-----+
```

- `REGEXP_LIKE(expr, pat[, match_type])`

文字列 `expr` がパターン `pat` で指定された正規表現と一致する場合は 1 を返し、一致しない場合は 0 を返します。`expr` または `pat` が `NULL` の場合、戻り値は `NULL` です。

パターンには、[正規表現構文](#) で説明されている構文である拡張正規表現を使用できます。パターンはリテラル文字列である必要はありません。たとえば、文字列式やテーブルカラムとして指定できます。

オプションの `match_type` 引数は、照合の実行方法を指定する次の文字の一部またはすべてを含むことができる文字列です:

- `c`: 大文字小文字を区別する照合。
- `i`: 大文字小文字を区別しない照合。
- `m`: Multiple-line mode. 文字列内の行終了記号を認識します。デフォルトの動作では、文字列式の先頭と末尾でのみ行終了記号が照合されます。
- `n`: . 文字は行終了記号と一致します。デフォルトでは、. 照合は行の最後で停止します。
- `u`: Unix 専用の行の終わり。改行文字のみが、.、^ および \$ の一致演算子で終わる行として認識されます。

矛盾するオプションを指定する文字が `match_type` 内で指定されている場合は、右端の文字が優先されます。

デフォルトでは、正規表現操作は、文字のタイプを決定して比較を実行するときに、`expr` および `pat` 引数の文字セットと照合順序を使用します。引数にさまざまな文字セットまたは照合順序が含まれる場合は、[セクション](#)

10.8.4 「式での照合の強制性」で説明するとおりに、型変換属性ルールが適用されます。引数は、比較動作を変更するために、明示的な照合インジケータとともに指定できます。

```
mysql> SELECT REGEXP_LIKE('CamelCase', 'CAMELCASE');
+-----+
| REGEXP_LIKE('CamelCase', 'CAMELCASE') |
+-----+
| 1 |
+-----+
mysql> SELECT REGEXP_LIKE('CamelCase', 'CAMELCASE' COLLATE utf8mb4_0900_as_cs);
+-----+
| REGEXP_LIKE('CamelCase', 'CAMELCASE' COLLATE utf8mb4_0900_as_cs) |
+-----+
| 0 |
+-----+
```

`match_type` は、デフォルトの大/小文字の区別をオーバーライドするために、`c` または `i` 文字とともに指定できます。例外: いずれかの引数がバイナリ文字列の場合、`match_type` に `i` 文字が含まれていても、引数は大/小文字を区別してバイナリ文字列として処理されます。

注記

MySQL では文字列で C エスケープ構文 (改行文字を表す `\n` など) が使用されるため、`expr` および `pat` 引数で使用する `\` をダブルクリックする必要があります。

```
mysql> SELECT REGEXP_LIKE('Michael!', '!.*');
+-----+
| REGEXP_LIKE('Michael!', '!.*') |
+-----+
| 1 |
+-----+
mysql> SELECT REGEXP_LIKE('new*\n*line', 'new\\.*\n*line');
+-----+
| REGEXP_LIKE('new*\n*line', 'new\\.*\n*line') |
+-----+
| 0 |
+-----+
mysql> SELECT REGEXP_LIKE('a', '[a-d]');
+-----+
| REGEXP_LIKE('a', '[a-d]') |
+-----+
| 1 |
+-----+
mysql> SELECT REGEXP_LIKE('a', 'A'), REGEXP_LIKE('a', BINARY 'A');
+-----+
| REGEXP_LIKE('a', 'A') | REGEXP_LIKE('a', BINARY 'A') |
+-----+
| 1 | 0 |
+-----+
```

```
mysql> SELECT REGEXP_LIKE('abc', 'ABC');
+-----+
| REGEXP_LIKE('abc', 'ABC') |
+-----+
| 1 |
+-----+
mysql> SELECT REGEXP_LIKE('abc', 'ABC', 'c');
+-----+
| REGEXP_LIKE('abc', 'ABC', 'c') |
+-----+
| 0 |
+-----+
```

- `REGEXP_REPLACE(expr, pat, repl[, pos[, occurrence[, match_type]])]`

パターン `pat` で指定された正規表現に一致する文字列 `expr` 内のオカレンスを置換文字列 `repl` で置換し、結果の文字列を返します。 `expr`、`pat` または `repl` が `NULL` の場合、戻り値は `NULL` です。

`REGEXP_REPLACE()` は、次のオプションの引数を取ります:

- `pos`: 検索を開始する `expr` 内の位置。省略した場合、デフォルトは 1 です。

- **occurrence**: 置換する一致のオカレンス。省略した場合、デフォルトは 0 (「すべて置換」を意味します) です。
- **match_type**: 照合の実行方法を指定する文字列。意味は、`REGEXP_LIKE()` で説明されているとおりです。

MySQL 8.0.17 より前は、この関数によって戻された結果で **UTF-16** 文字セットが使用されていました。MySQL 8.0.17 以降では、一致を検索した式の文字セットおよび照合順序が使用されます。(Bug #94203、Bug #29308212)

照合の実行方法の詳細は、`REGEXP_LIKE()` の説明を参照してください。

```
mysql> SELECT REGEXP_REPLACE('a b c', 'b', 'X');
+-----+
| REGEXP_REPLACE('a b c', 'b', 'X') |
+-----+
| a X c                               |
+-----+
mysql> SELECT REGEXP_REPLACE('abc def ghi', '[a-z]+', 'X', 1, 3);
+-----+
| REGEXP_REPLACE('abc def ghi', '[a-z]+', 'X', 1, 3) |
+-----+
| abc def X                                       |
+-----+
```

- `REGEXP_SUBSTR(expr, pat[, pos[, occurrence[, match_type]])`

一致がない場合は、`pat`、`NULL` パターンで指定された正規表現に一致する文字列 `expr` の部分文字列を返します。`expr` または `pat` が `NULL` の場合、戻り値は `NULL` です。

`REGEXP_SUBSTR()` は、次のオプションの引数を取ります:

- **pos**: 検索を開始する `expr` 内の位置。省略した場合、デフォルトは 1 です。
- **occurrence**: 検索する一致のオカレンス。省略した場合、デフォルトは 1 です。
- **match_type**: 照合の実行方法を指定する文字列。意味は、`REGEXP_LIKE()` で説明されているとおりです。

MySQL 8.0.17 より前は、この関数によって戻された結果で **UTF-16** 文字セットが使用されていました。MySQL 8.0.17 以降では、一致を検索した式の文字セットおよび照合順序が使用されます。(Bug #94203、Bug #29308212)

照合の実行方法の詳細は、`REGEXP_LIKE()` の説明を参照してください。

```
mysql> SELECT REGEXP_SUBSTR('abc def ghi', '[a-z]+');
+-----+
| REGEXP_SUBSTR('abc def ghi', '[a-z]+') |
+-----+
| abc                                       |
+-----+
mysql> SELECT REGEXP_SUBSTR('abc def ghi', '[a-z]+', 1, 3);
+-----+
| REGEXP_SUBSTR('abc def ghi', '[a-z]+', 1, 3) |
+-----+
| ghi                                       |
+-----+
```

正規表現構文

正規表現では、文字列のセットが記述されます。もっとも単純な正規表現は、特殊文字を使用していないものです。たとえば、正規表現 `hello` は `hello` にのみ一致します。

重要な正規表現では、複数の文字列に一致できるように特定の特殊構造が使用されます。たとえば、正規表現 `hello|world` には `|` 代替演算子が含まれ、`hello` または `world` のいずれかと一致します。

さらに複雑な例として、正規表現 `B[an]*s` は、文字列 `Bananas`、`Baaaaas`、`Bs` のいずれか、および `B` で始まり、`s` で終わり、その間に任意の数字の `a` または `n` 文字が含まれるその他の文字列に一致します。

次のリストは、正規表現で利用できる基本的な特殊文字と構造体の一部を示しています。正規表現サポートの実装に使用される ICU ライブラリでサポートされる完全な正規表現構文の詳細は、「[Unicode web サイトの国際コンポーネント](#)」を参照してください。

- `^`

文字列の先頭に一致します。

```
mysql> SELECT REGEXP_LIKE('fo\lno', '^fo$');      -> 0
mysql> SELECT REGEXP_LIKE('fofo', '^fo');        -> 1
```

- `$`

文字列の末尾に一致します。

```
mysql> SELECT REGEXP_LIKE('fo\lno', 'fo\lno$');  -> 1
mysql> SELECT REGEXP_LIKE('fo\lno', '^fo$');     -> 0
```

- `.`

任意の文字に一致します (キャリッジリターンおよび改行を含むが、文字列の途中でこれらを一致させるには、`m` (複数行) の一致制御文字または `(?m)` のパターン内修飾子を指定する必要があります)。

```
mysql> SELECT REGEXP_LIKE('fofo', '^f.*$');      -> 1
mysql> SELECT REGEXP_LIKE('fo\r\nfo', '^f.*$'); -> 0
mysql> SELECT REGEXP_LIKE('fo\r\nfo', '^f.*$', 'm'); -> 1
mysql> SELECT REGEXP_LIKE('fo\r\nfo', '(?m)^f.*$'); -> 1
```

- `a*`

ゼロ個以上の `a` 文字のシーケンスに一致します。

```
mysql> SELECT REGEXP_LIKE('Ban', '^Ba*n');      -> 1
mysql> SELECT REGEXP_LIKE('Baaan', '^Ba*n');   -> 1
mysql> SELECT REGEXP_LIKE('Bn', '^Ba*n');      -> 1
```

- `a+`

1 個以上の `a` 文字のシーケンスに一致します。

```
mysql> SELECT REGEXP_LIKE('Ban', '^Ba+n');     -> 1
mysql> SELECT REGEXP_LIKE('Bn', '^Ba+n');     -> 0
```

- `a?`

ゼロまたは 1 個の `a` 文字に一致します。

```
mysql> SELECT REGEXP_LIKE('Bn', '^Ba?n');     -> 1
mysql> SELECT REGEXP_LIKE('Ban', '^Ba?n');    -> 1
mysql> SELECT REGEXP_LIKE('Baan', '^Ba?n');  -> 0
```

- `de|abc`

代替。順序 `de` または `abc` のいずれかに一致します。

```
mysql> SELECT REGEXP_LIKE('pi', 'pi|apa');    -> 1
mysql> SELECT REGEXP_LIKE('axe', 'pi|apa');   -> 0
mysql> SELECT REGEXP_LIKE('apa', 'pi|apa');   -> 1
mysql> SELECT REGEXP_LIKE('apa', '^pi|apa$'); -> 1
mysql> SELECT REGEXP_LIKE('pi', '^pi|apa$');  -> 1
mysql> SELECT REGEXP_LIKE('pix', '^pi|apa$'); -> 0
```

- `(abc)*`

シーケンス `abc` のゼロ個以上のインスタンスに一致します。

```
mysql> SELECT REGEXP_LIKE('pi', '^pi*$');     -> 1
mysql> SELECT REGEXP_LIKE('pip', '^pi*$');   -> 0
mysql> SELECT REGEXP_LIKE('pipi', '^pi*$');  -> 1
```

- {1}, {2,3}

繰返し: {n} および {m,n} 表記法では、パターンの以前の原子 (または「ピース」) の多くの出現に一致する正規表現を記述するより一般的な方法が提供されます。m および n は整数です。

- a*

a{0,} と記述できます。

- a+

a{1,} と記述できます。

- a?

a{0,1} と記述できます。

より正確にするために、a{n} は a の n インスタンスと完全に一致します。a{n,} は、a の n 以上のインスタンスと一致します。a{m, n} は、a の n インスタンス (これを含む) を介して m を照合します。m および n の両方が指定されている場合は、m を n 以下にする必要があります。

```
mysql> SELECT REGEXP_LIKE('abcde', 'a[bcd]{2}e');      -> 0
mysql> SELECT REGEXP_LIKE('abcde', 'a[bcd]{3}e');      -> 1
mysql> SELECT REGEXP_LIKE('abcde', 'a[bcd]{1,10}e');    -> 1
```

- [a-dX], [^a-dX]

a, b, c, d または X のいずれかである (または ^ が使用されている場合はそうでない) 任意の文字に一致します。2 つの文字の間の - 文字によって、1 番目の文字から 2 番目の文字までのすべての文字に一致する範囲が形成されます。たとえば、[0-9] は任意の 10 進数に一致します。リテラル文字] を含めるには、左括弧 [の直後に記述する必要があります。リテラル文字 - を含めるには、先頭または末尾に記述する必要があります。[] ペアの内に定義された特殊な意味を持たない文字は、それ自体としか一致しません。

```
mysql> SELECT REGEXP_LIKE('aXbc', '[a-dXYZ]');        -> 1
mysql> SELECT REGEXP_LIKE('aXbc', '[^a-dXYZ]$');       -> 0
mysql> SELECT REGEXP_LIKE('aXbc', '[a-dXYZ]+$');      -> 1
mysql> SELECT REGEXP_LIKE('aXbc', '[^a-dXYZ]+$');     -> 0
mysql> SELECT REGEXP_LIKE('gheis', '[^a-dXYZ]+$');    -> 1
mysql> SELECT REGEXP_LIKE('gheisa', '[^a-dXYZ]+$');   -> 0
```

- [=character_class=]

([と] 使用して記述された) 括弧式内の [=character_class=] は、等価クラスを表します。これは、同じ照合順序値を持つすべての文字 (それ自体を含む) に一致します。たとえば、o および (+) が等価クラスのメンバーである場合、[[=o=]], [[=(+)=]], および [o(+)] はすべてシノニムです。等価クラスは、範囲の終点として使用できない場合があります。

- [:character_class:]

([と] を使用して記述された) 括弧式内の [:character_class:] は、そのクラスに属するすべての文字と一致する文字クラスを表します。次の表には、標準のクラス名を一覧表示します。これらの名前は、ctype(3) のマニュアルページで定義されている文字クラスを表しています。特定のロケールでは、ほかのクラス名が提供される場合もあります。文字クラスは、範囲の終点として使用できない場合があります。

文字クラス名	意味
alnum	英数文字
alpha	アルファベット文字
blank	空白文字
cntrl	制御文字
digit	数字文字
graph	図形文字

文字クラス名	意味
<code>lower</code>	小文字アルファベット文字
<code>print</code>	図形または空白文字
<code>punct</code>	句読点文字
<code>space</code>	空白、タブ、改行、および復帰改行
<code>upper</code>	大文字アルファベット文字
<code>xdigit</code>	16 進数文字

```
mysql> SELECT REGEXP_LIKE('justalnums', '[:alnum:]+');      -> 1
mysql> SELECT REGEXP_LIKE('!', '[:alnum:]+');              -> 0
```

正規表現で特殊文字のリテラルインスタンスを使用するには、前に 2 つのバックスラッシュ (\) 文字を付けます。MySQL パーサーが 2 つのバックスラッシュの一方を解釈し、正規表現ライブラリがもう一方を解釈します。たとえば、特殊文字 + を含む文字列 `1+2` に一致する正規表現は、次のうちで最後のものだけです。

```
mysql> SELECT REGEXP_LIKE('1+2', '1+2');                  -> 0
mysql> SELECT REGEXP_LIKE('1+2', '1\+2');                 -> 0
mysql> SELECT REGEXP_LIKE('1+2', '1\\+2');                -> 1
```

正規表現のリソース制御

`REGEXP_LIKE()` および同様の関数は、システム変数を設定することで制御できるリソースを使用します:

- 照合エンジンは、内部スタックにメモリーを使用します。スタックで使用可能な最大メモリーをバイト単位で制御するには、`regexp_stack_limit` システム変数を設定します。
- 照合エンジンはステップで動作します。エンジンによって実行されるステップの最大数 (つまり、間接的に実行時間) を制御するには、`regexp_time_limit` システム変数を設定します。この制限はステップ数で表されるため、実行時間には間接的にのみ影響します。通常はミリ秒の順序で表示されます。

正規表現の互換性に関する考慮事項

MySQL 8.0.4 より前は、MySQL は、International Components for Unicode (ICU) ではなく Henry Spencer 正規表現ライブラリを使用して正規表現操作をサポートしていました。次の説明では、アプリケーションに影響を与える可能性のある Spencer ライブラリと ICU ライブラリの違いについて説明します:

- Spencer ライブラリでは、`REGEXP` および `RLIKE` 演算子はバイト単位で動作するため、マルチバイトセーフではなく、マルチバイト文字セットで予期しない結果が発生する可能性があります。さらに、これらの演算子ではそのバイト値に基づいて文字が比較されるため、アクセント記号付き文字は、指定された照合順序では等しいとみなされた場合でも、等しいとして比較されない可能性があります。

ICU では Unicode が完全にサポートされており、マルチバイトセーフです。その正規表現関数は、すべての文字列を `UTF-16` として扱います。位置インデックスは、コードポイントではなく 16 ビットチャンクに基づいていることに注意してください。つまり、このような関数に渡された場合、複数のチャンクを使用する文字は、次に示すような予期しない結果を生成する可能性があります:

```
mysql> SELECT REGEXP_INSTR('##b', 'b');
+-----+
| REGEXP_INSTR('??b', 'b') |
+-----+
|          5 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT REGEXP_INSTR('##bxxx', 'b', 4);
+-----+
| REGEXP_INSTR('??bxxx', 'b', 4) |
+-----+
|          5 |
+-----+
1 row in set (0.00 sec)
```

ほとんどの最新言語で使用される文字を含む Unicode Basic Multilingual Plane 内の文字は、次の点で安全です:


```
mysql> SELECT REGEXP_INSTR('6xb', 'b');
```

```
+-----+
| REGEXP_INSTR('6xb', 'b') |
+-----+
|           3 |
+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT REGEXP_INSTR('yab', 'b');
```

```
+-----+
| REGEXP_INSTR('yab', 'b') |
+-----+
|           3 |
+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT REGEXP_INSTR('μά周çö', '周');
```

```
+-----+
| REGEXP_INSTR('μά周çö', '周') |
+-----+
|           3 |
+-----+
1 row in set (0.00 sec)
```

最初の 2 つの例で使用されている「「寿司」」文字 (U+1F363) などの絵文字は、基本多言語プレーンではなく Unicode 補助多言語プレーンに含まれています。REGEXP_SUBSTR() または同様の関数が文字の途中で検索を開始すると、emoji およびその他の 4 バイト文字で別の問題が発生する可能性があります。次の例の 2 つのステートメントはそれぞれ、最初の引数の 2 バイト目の位置から始まります。最初のステートメントは、2 バイト (BMP) 文字のみで構成される文字列に対して機能します。2 番目のステートメントには 4 バイトの文字が含まれており、最初の 2 バイトが取り除かれ、残りの文字データが正しく配置されていないため、結果で正しく解釈されません。

```
mysql> SELECT REGEXP_SUBSTR('周周周周', '.*', 2);
```

```
+-----+
| REGEXP_SUBSTR('周周周周', '.*', 2) |
+-----+
| 周周周 |
+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT REGEXP_SUBSTR('#####', '.*', 2);
```

```
+-----+
| REGEXP_SUBSTR('#####', '.*', 2) |
+-----+
| ?#擡#擡#擡 |
+-----+
1 row in set (0.00 sec)
```

- 演算子の場合、Spencer ライブラリは、文字列式の任意の場所 (中央を含む) の行終了文字 (キャリッジリターン、改行) と一致します。文字列の途中にある行末文字を ICU と照合するには、`m` の一致制御文字を指定します。
- Spencer ライブラリは、ワードスタート境界マーカとワードエンド境界マーカ ([[:<:]] および [[:>:]] 表記法) をサポートしています。ICU はそうではありません。ICU の場合、`\b` を使用して単語境界を照合できます。MySQL では文字列内のエスケープ文字として解釈されるため、バックスラッシュをダブルクリックします。
- Spencer ライブラリは、照合要素ブラケット式 ([[:characters:]] 表記法) をサポートしています。ICU はそうではありません。
- 繰返し回数 (`{n}` および `{m,n}` 表記) の場合、Spencer ライブラリの最大数は 255 です。ICU にはこのような制限はありませんが、照合エンジンステップの最大数は `regex_time_limit` システム変数を設定することで制限できます。
- ICU はカッコをメタ文字として解釈します。リテラルの左カッコまたは右カッコ (を正規表現で指定するには、エスケープする必要があります) :

```
mysql> SELECT REGEXP_LIKE('(', '(');
```

```
ERROR 3692 (HY000): Mismatched parenthesis in regular expression.
```

```
mysql> SELECT REGEXP_LIKE('(', '\(');
```

```
+-----+
| REGEXP_LIKE('(', '\(') |
+-----+
|           1 |
+-----+
```

```

+-----+
mysql> SELECT REGEXP_LIKE(')', ');
ERROR 3692 (HY000): Mismatched parenthesis in regular expression.
mysql> SELECT REGEXP_LIKE(')', '\)');
+-----+
| REGEXP_LIKE(')', '\)') |
+-----+
|          1          |
+-----+

```

- ICU は大カッコもメタ文字として解釈しますが、リテラル文字として使用するには、左大カッコのみをエスケープする必要があります:

```

mysql> SELECT REGEXP_LIKE('[', '[');
ERROR 3696 (HY000): The regular expression contains an
unclosed bracket expression.
mysql> SELECT REGEXP_LIKE('[', '\[');
+-----+
| REGEXP_LIKE('[', '\[') |
+-----+
|          1          |
+-----+
mysql> SELECT REGEXP_LIKE('[', '[');
+-----+
| REGEXP_LIKE('[', '[') |
+-----+
|          1          |
+-----+

```

12.8.3 関数結果の文字セットと照合順序

MySQL には、文字列を返す多数の演算子と関数があります。このセクションでは、そのような文字列の文字セットと照合順序について説明します。

文字列の入力を取得して文字列の結果を出力として返す単純な関数では、出力の文字セットおよび照合順序は、主要な入力値の文字セットおよび照合順序と同じです。たとえば、`UPPER(X)` は、`X` と同じ文字列および照合順序を持つ文字列を返します。同じことは、`INSTR()`、`LCASE()`、`LOWER()`、`LTRIM()`、`MID()`、`REPEAT()`、`REPLACE()`、`REVERSE()`、`RIGHT()`、`RPAD()`、`RTRIM()`、`SC` および `UPPER()` についても当てはまります。

注記

`REPLACE()` 関数は、他のすべての関数とは異なり、常に文字列入力の照合を無視し、大/小文字を区別する比較を実行します。

文字列入力または関数結果がバイナリ文字列の場合、文字列は `binary` 文字セットおよび照合順序を持ちます。これは、`CHARSET()` および `COLLATION()` 関数を使用してチェックできます。どちらの関数もバイナリ文字列引数に `binary` を返します:

```

mysql> SELECT CHARSET(BINARY 'a'), COLLATION(BINARY 'a');
+-----+-----+
| CHARSET(BINARY 'a') | COLLATION(BINARY 'a') |
+-----+-----+
| binary              | binary                |
+-----+-----+

```

複数の文字列入力を組み合わせて単一の文字列出力を返す演算では、結果の照合順序の特定に次の標準 SQL の「アグリゲーションルール」が適用されます。

- 明示的な `COLLATE Y` が発生した場合は、`Y` を使用します。
- 明示的な `COLLATE Y` および `COLLATE Z` が発生した場合は、エラーが発生します。
- それ以外の場合、すべての照合順序が `Y` であれば、`Y` を使用します。
- その他の場合、結果に照合順序はありません。

たとえば、`CASE ... WHEN a THEN b WHEN b THEN c COLLATE X END` と指定されている場合、結果の照合順序は `X` になります。同じことは、`UNION`、`||`、`CONCAT()`、`ELT()`、`GREATEST()`、`IF()`、および `LEAST()` にも当てはまります。

文字データに変換する操作の場合、操作によって生成される文字列の文字セットと照合順序は、デフォルトの接続文字セットと照合順序を決定する `character_set_connection` および `collation_connection` システム変数によって定義されます (セクション 10.4 「接続文字セットおよび照合順序」を参照)。これは、`BIN_TO_UUID()`、`CAST()`、`CONV()`、`FORMAT()`、`HEX()` および `SPACE()` にのみ適用されます。

仮想生成カラムの式では、前述の原則の例外が発生します。このような式では、接続文字セットに関係なく、テーブルの文字セットが `BIN_TO_UUID()`、`CONV()` または `HEX()` の結果に使用されます。

文字列関数によって返される結果の文字セットまたは照合順序に関する質問がある場合は、`CHARSET()` または `COLLATION()` 関数を使用して次を確認してください:

```
mysql> SELECT USER(), CHARSET(USER()), COLLATION(USER());
+-----+-----+-----+
| USER() | CHARSET(USER()) | COLLATION(USER()) |
+-----+-----+-----+
| test@localhost | utf8 | utf8_general_ci |
+-----+-----+-----+
mysql> SELECT CHARSET(COMPRESS('abc')), COLLATION(COMPRESS('abc'));
+-----+-----+
| CHARSET(COMPRESS('abc')) | COLLATION(COMPRESS('abc')) |
+-----+-----+
| binary | binary |
+-----+-----+
```

12.9 MySQL で使用されるカレンダー

MySQL では、先発グレゴリオ暦と呼ばれるものが使用されています。

ユリウス暦からグレゴリオ暦に切り替えた国はすべて、切り替え時に少なくとも 10 日間を破棄する必要がありました。この動作を確認するために、最初にユリウス暦からグレゴリオ暦への切り替えが発生した 1582 年 10 月を考えてください。

月曜日	火曜日	水曜日	木曜日	金曜日	土曜日	日曜日
1	2	3	4	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

10 月 4 日から 10 月 15 日の間には日付がありません。この不連続性はカットオーバーと呼ばれます。カットオーバー前の日付はユリウス暦で、カットオーバー後の日付はグレゴリオ暦です。カットオーバー中の日付は存在しません。

実際に使用されていなかった日付に適用されるカレンダーは、先発と呼ばれます。したがって、カットオーバーが発生せず、常にグレゴリオ暦のルールで制御されていると考えられる場合は、先発グレゴリオ暦が使用されています。これが MySQL で使用されるものであり、標準 SQL でも必須です。そのため、MySQL `DATE` または `DATETIME` 値として格納されたカットオーバー前の日付は、その違いが補正されるように調整する必要があります。すべての国で同時にカットオーバーが発生しなかったこと、および発生が遅くなるほど失われる日数も多かったことに気付くことが重要です。たとえば、イギリスでは 1752 年に発生し、9 月 2 日水曜日の翌日が 9 月 14 日木曜日になりました。ロシアでは 1918 年までユリウス暦のままでしたが、その過程で 13 日間が失われました。「10 月革命」として知られる有名な事件は、グレゴリオ暦に従うと 11 月に発生しました。

12.10 全文検索関数

`MATCH (col1,col2,...) AGAINST (expr [search_modifier])`

`search_modifier:`

```
{
  IN NATURAL LANGUAGE MODE
| IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION
| IN BOOLEAN MODE
| WITH QUERY EXPANSION
}
```

MySQL では、次のような全文インデックス設定および検索がサポートされています。

- MySQL の全文インデックスは、型 `FULLTEXT` のインデックスです。
- 全文インデックスは、`InnoDB` または `MyISAM` テーブルでのみ使用でき、`CHAR`、`VARCHAR`、または `TEXT` カラムにのみ作成できます。
- MySQL には、中国語、日本語および韓国語 (CJK) をサポートする組み込みの全文 ngram パーサーと、日本語用のインストール可能な MeCab 全文パーサープラグインが用意されています。解析の相違点については、[セクション 12.10.8 「ngram 全文パーサー」](#) と [セクション 12.10.9 「MeCab フルテキストパーサープラグイン」](#) で概説します。
- `FULLTEXT` インデックスの定義は、テーブルの作成時に `CREATE TABLE` ステートメントで指定することも、あとで `ALTER TABLE` または `CREATE INDEX` を使用して追加することもできます。
- データセットが大きい場合は、`FULLTEXT` インデックスが付いていないテーブルにロードしてから、そのあとでインデックスを作成した方が、既存の `FULLTEXT` インデックスが付いているテーブルにロードするよりも断然速いです。

全文検索は、`MATCH() ... AGAINST` 構文を使用して実行されます。`MATCH()` には、検索対象のカラム名をカンマで区切ったリストを指定します。`AGAINST` には、検索する文字列と、実行する検索のタイプを示すオプションの修飾子を指定します。検索文字列は、クエリー評価時に定数である文字列値にする必要があります。たとえば、テーブルカラムは、行ごとに異なる可能性があるため除外されます。

全文検索には、次の 3 つの種類があります。

- 自然言語の検索では、検索文字列が人間の自然な言語でのフレーズ (フリーテキストのフレーズ) として解釈されます。二重引用符 (") 文字を除き、特殊演算子はありません。ストップワードリストが適用されます。ストップワードリストの詳細は、[セクション 12.10.4 「全文ストップワード」](#) を参照してください。

`IN NATURAL LANGUAGE MODE` 修飾子が指定されている場合または修飾子がまったく指定されていない場合は、全文検索が自然言語検索になります。詳細は、[セクション 12.10.1 「自然言語全文検索」](#) を参照してください。

- ブール検索では、特別なクエリー言語のルールを使用して検索文字列が解釈されます。文字列には、検索対象の単語が含まれます。また、一致する行に単語が存在しなければならない、または存在してはならないように、あるいは通常よりも単語の重みが高くまたは低くなるように、要件を指定する演算子を含めることもできます。特定の共通単語 (ストップワード) は、検索インデックスから省略され、検索文字列に存在しない場合は一致が行われません。`IN BOOLEAN MODE` 修飾子は、ブール検索を指定します。詳細は、[セクション 12.10.2 「ブール全文検索」](#) を参照してください。
- クエリー拡張検索は、自然言語検索を改善したものです。自然言語検索を実行する際は、検索文字列が使用されます。その後、検索で返されたもっとも関連性の高い行からの単語が検索文字列に追加され、再度検索が実行されます。クエリーでは、2 回目の検索からの行が返されます。`IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION` または `WITH QUERY EXPANSION` 修飾子は、クエリー拡張検索を指定します。詳細は、[セクション 12.10.3 「クエリー拡張を使用した全文検索」](#) を参照してください。

`FULLTEXT` クエリーのパフォーマンスについては、[セクション 8.3.5 「カラムインデックス」](#) を参照してください。

`InnoDB FULLTEXT` インデックスの詳細は、[セクション 15.6.2.4 「InnoDB FULLTEXT インデックス」](#) を参照してください。

全文検索上の制約については、[セクション 12.10.5 「全文制限」](#) に一覧表示されています。

`myisam_ftdump` ユーティリティーは、`MyISAM` 全文インデックスの内容をダンプします。これは、全文クエリーのデバッグ時に役立つことがあります。[セクション 4.6.3 「myisam_ftdump — 全文インデックス情報の表示」](#) を参照してください。

12.10.1 自然言語全文検索

デフォルトの場合や `IN NATURAL LANGUAGE MODE` 修飾子が指定された場合は、`MATCH()` 関数は、テキストコレクションに対して文字列の自然言語検索を実行します。コレクションは、`FULLTEXT` インデックスに含まれる 1 つ以上のカラムのセットです。検索文字列は、`AGAINST()` への引数として指定されます。`MATCH()` は、テーブルの行ごとに関連性の値を返します。つまり、検索文字列と、`MATCH()` リストで名前が指定されたカラムの該当行のテキスト間で類似性が評価されます。

```
mysql> CREATE TABLE articles (
  id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
  title VARCHAR(200),
  body TEXT,
  FULLTEXT (title,body)
) ENGINE=InnoDB;
Query OK, 0 rows affected (0.08 sec)

mysql> INSERT INTO articles (title,body) VALUES
('MySQL Tutorial','DBMS stands for DataBase ...'),
('How To Use MySQL Well','After you went through a ...'),
('Optimizing MySQL','In this tutorial, we show ...'),
('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
('MySQL vs. YourSQL','In the following database comparison ...'),
('MySQL Security','When configured properly, MySQL ...');
Query OK, 6 rows affected (0.01 sec)
Records: 6 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM articles
  WHERE MATCH (title,body)
  AGAINST ('database' IN NATURAL LANGUAGE MODE);
+-----+-----+-----+
| id | title           | body                                     |
+-----+-----+-----+
| 1 | MySQL Tutorial  | DBMS stands for DataBase ...           |
| 5 | MySQL vs. YourSQL | In the following database comparison ... |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

デフォルトでは、大文字と小文字が区別される方法で検索が実行されます。大/小文字を区別する全文検索を実行するには、インデックス付けされたカラムに大/小文字を区別する照合またはバイナリ照合を使用します。たとえば、`utf8mb4` 文字セットを使用するカラムには、`utf8mb4_0900_as_cs` または `utf8mb4_bin` の照合順序を割り当てて、全文検索で大文字と小文字を区別できます。

以前に例で示したように、`MATCH()` が `WHERE` 句で使用されると、もっとも関連性の高い行が 1 番目に返されるように、自動的にソートされます。関連性の値は、負ではない浮動小数点数です。ゼロの関連性は、類似性がないという意味です。関連性は、行 (ドキュメント) 内の単語の数、行内の一意の単語の数、コレクション内の単語の合計数および特定の単語を含む行の数に基づいて計算されます。

注記

「document」という用語は、「row」という用語と同じ意味で使用でき、どちらの用語も行のインデックス付けされた部分を指します。「collection」という用語は、インデックス付けされたカラムを指し、すべての行を含みます。

単に一致をカウントするには、次のようなクエリーを使用してください。

```
mysql> SELECT COUNT(*) FROM articles
  WHERE MATCH (title,body)
  AGAINST ('database' IN NATURAL LANGUAGE MODE);
+-----+
| COUNT(*) |
+-----+
|      2 |
+-----+
1 row in set (0.00 sec)
```

次のように、クエリーを再作成した方が早い場合もあります。

```
mysql> SELECT
  COUNT(IF(MATCH (title,body) AGAINST ('database' IN NATURAL LANGUAGE MODE), 1, NULL))
```

```
AS count
FROM articles;
+-----+
| count |
+-----+
| 2 |
+-----+
1 row in set (0.03 sec)
```

1 つ目のクエリーでは、いくつかの追加作業（関連性別の結果のソート）が実行されますが、**WHERE** 句に基づくインデックス検索を使用することもできます。インデックス検索では、検索でほとんど行が一致しない場合に、最初のクエリーが速くなる可能性があります。2 つ目のクエリーではテーブルの完全スキャンが実行され、ほとんどの行に検索語句が存在した場合に、インデックス検索よりも高速になる可能性があります。

自然言語による全文検索では、**MATCH()** 関数で名前が指定されたカラムは、テーブル内の一部の **FULLTEXT** インデックスに含まれるカラムと同じである必要があります。前述のクエリーでは、**MATCH()** 関数 (**title** および **body**) で指定されたカラムは、**article** テーブル **FULLTEXT** インデックスの定義で指定されたカラムと同じであることに注意してください。 **title** または **body** を個別に検索するには、カラムごとに個別の **FULLTEXT** インデックスを作成します。

ブール検索やクエリー拡張を使用した検索を実行することもできます。これらの検索タイプについては、[セクション 12.10.2 「ブール全文検索」](#) および [セクション 12.10.3 「クエリー拡張を使用した全文検索」](#) で説明されています。

インデックスを使用した全文検索では、インデックスが複数のテーブルに及ぶ可能性はないため、**MATCH()** 句の単一テーブルにあるカラムにしか名前を付けることができません。 **MyISAM** テーブルでは、インデックスが存在しない場合でも、ブール検索を実行できます（ただし、低速になります）。この場合、複数のテーブルのカラムに名前を指定できます。

上記の例では、関連性の降順で行が返される **MATCH()** 関数の使用方法について簡単に説明しました。次の例では、関連性の値を明示的に取得する方法を示します。 **SELECT** ステートメントには **WHERE** 句も **ORDER BY** 句も含まれていないため、返される行は順序付けられません。

```
mysql> SELECT id, MATCH (title,body)
  AGAINST ('Tutorial' IN NATURAL LANGUAGE MODE) AS score
  FROM articles;
+-----+
| id | score          |
+-----+
| 1 | 0.22764469683170319 |
| 2 | 0 |
| 3 | 0.22764469683170319 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
+-----+
6 rows in set (0.00 sec)
```

次の例はさらに複雑です。このクエリーでは関連性の値が返され、関連性の降順での行のソートも行われます。この結果を実現するには、**MATCH()** を 2 回（1 回は **SELECT** リストに、もう 1 回は **WHERE** 句に）指定します。MySQL オプティマイザによって、2 回の **MATCH()** 呼び出しが同じであり、全文検索コードが 1 回のみ起動されることが検出されるため、追加のオーバーヘッドは発生しません。

```
mysql> SELECT id, body, MATCH (title,body) AGAINST
  ('Security implications of running MySQL as root'
  IN NATURAL LANGUAGE MODE) AS score
  FROM articles WHERE MATCH (title,body) AGAINST
  ('Security implications of running MySQL as root'
  IN NATURAL LANGUAGE MODE);
+-----+
| id | body | score          |
+-----+
| 4 | 1. Never run mysqld as root. 2. ... | 1.5219271183014 |
| 6 | When configured properly, MySQL ... | 1.3114095926285 |
+-----+
2 rows in set (0.00 sec)
```

二重引用符 (") 文字内で囲まれたフレーズは、入力されたそのままのフレーズを含む行にのみ一致します。全文エンジンでは、フレーズが複数の単語に分割され、それらの単語の **FULLTEXT** インデックス内で検索が実行されます。単語以外の文字は、正確に一致する必要がありません。フレーズ検索では、そのフレーズとまったく同じ単語が同じ順序で一致に含まれることのみが必要です。たとえば、**"test phrase"** は **"test, phrase"** と一致します。フレーズにイ

インデックス内にある単語が含まれない場合は、結果が空になります。たとえば、すべての単語がストップワードであるか、インデックス付けされた単語の最小長より短い場合、結果は空になります。

MySQL **FULLTEXT** の実装では、ツールワード文字 (文字、数字、およびアンダースコア) のシーケンスが単語とみなされます。そのシーケンスには、アポストロフィー (') も含めることはできますが、1 行に 1 つまでです。つまり、`aaa'bbb` は 1 語とみなされますが、`aaa"bbb` は 2 語とみなされます。単語の先頭または末尾のアポストロフィーは、**FULLTEXT** パーサーによって削除されます。`'aaa'bbb` は、`aaa'bbb` として解析されます。

組み込み **FULLTEXT** パーサーは、(空白)、(カンマ)、(ピリオド) などの特定のデリミタ文字を検索して、単語の開始位置と終了位置を決定します。単語がデリミタで区切られていない場合 (中国語など)、組み込み **FULLTEXT** パーサーは単語の開始位置または終了位置を判別できません。組み込み **FULLTEXT** パーサーを使用する **FULLTEXT** インデックスに、このような言語の単語またはインデックス付けされた他の用語を追加できるようにするには、それらが任意のデリミタで区切られるように事前処理する必要があります。または、ngram パーサープラグイン (中国語、日本語または韓国語) または MeCab パーサープラグイン (日本語) を使用して、**FULLTEXT** インデックスを作成できます。

組み込みの全文パーサーを置き換えるプラグインを記述できます。詳細は、[The MySQL Plugin API](#)を参照してください。パーサープラグインのサンプルソースコードについては、MySQL のソース配布の [plugin/fulltext](#) ディレクトリを参照してください。

全文検索では、一部の単語が無視されます。

- 短すぎる単語は無視されます。全文検索で見つかった単語のデフォルトの最小長は、**InnoDB** 検索インデックスの場合は 3 文字、**MyISAM** の場合は 4 文字です。インデックスを作成する前に、構成オプション (**InnoDB** 検索インデックスの場合は `innodb_ft_min_token_size` 構成オプション、**MyISAM** の場合は `ft_min_word_len`) を設定すると、カットオフを制御できます。

注記

この動作は、ngram パーサーを使用する **FULLTEXT** インデックスには適用されません。ngram パーサーの場合、トークンの長さは `ngram_token_size` オプションによって定義されます。

- ストップワードリスト内の単語は無視されます。ストップワードは、セマンティクス値がゼロであると考えられるほど一般的な単語 (「the」や「some」など) です。組み込みのストップワードリストもありますが、ユーザー定義のリストでオーバーライドできます。ストップワードリストおよび関連する構成オプションは、**InnoDB** 検索インデックスと **MyISAM** 検索インデックスとで異なります。ストップワードの処理は、**InnoDB** 検索インデックスの場合は構成オプション `innodb_ft_enable_stopword`、`innodb_ft_server_stopword_table`、および `innodb_ft_user_stopword_table`、**MyISAM** 検索インデックスの場合は `ft_stopword_file` によって制御されます。

デフォルトのストップワードリストおよびそれを変更する方法を表示する方法については、[セクション12.10.4「全文ストップワード」](#)を参照してください。単語のデフォルト最小長は、[セクション12.10.6「MySQLの全文検索の微調整」](#)で説明したように変更できます。

コレクションおよびクエリー内のすべての正確な単語は、コレクションまたはクエリーでの重要性に従って重み付けられます。したがって、多くのドキュメント内に存在する単語では、この特定のコレクション内のセマンティクス値が低くなるため、重みも低くなります。反対に、まれな単語には、高い重みが付けられます。単語の重みを組み合わせることで、行の関連性が計算されます。この技術は、大きなコレクションで最適に機能します。

MyISAM の制限

非常に小さなテーブルでは、単語の配布が適切にセマンティクス値に反映されないため、このモデルでは、**MyISAM** テーブル上の検索インデックスに対して異常な結果が生成される可能性があります。たとえば、以前に示した `articles` テーブルのすべての行には、「MySQL」という単語が存在しますが、**MyISAM** 検索インデックス内の単語を検索しても結果が生成されません。

```
mysql> SELECT * FROM articles
  WHERE MATCH (title,body)
  AGAINST ('MySQL' IN NATURAL LANGUAGE MODE);
Empty set (0.00 sec)
```

「MySQL」という単語は 50% 以上の行に存在するため、検索の結果が空になります。そのため、事実上ストップワードとして処理されます。このフィルタ処理技術は、よくある語句に対して不適切な結果が生成される可能性のある小さなデータセットよりも、1G バイトの

テーブルから 1 行おきに結果セットが返される可能性のない大きなデータセットに適しています。

全文検索がどのように動作するかを確認するために最初に試すと、しきい値が 50% であることに驚くかもしれませんが、これによって **InnoDB** テーブルが全文検索での実験によりふさわしいことがわかります。 **MyISAM** テーブルを作成し、それに 1、2 行のテキストのみを挿入する場合は、テキスト内のすべての単語が 50% 以上の行に出現します。その結果、テーブルにより多くの行が含まれるまで、検索の結果が返されません。50% の制限を回避する必要があるユーザーは、**InnoDB** テーブル上に検索インデックスを構築したり、[セクション 12.10.2 「ブール全文検索」](#) で説明したブール検索モードを使用したりできます。

12.10.2 ブール全文検索

MySQL では、**IN BOOLEAN MODE** 修飾子を使用することでブール全文検索を実行できます。この修飾子を使用すると、検索文字列の先頭または末尾にある特定の文字が特別な意味を持ちます。次のクエリーでは、**+** および **-** 演算子は、一致が発生するために単語が存在しなければならないことと、単語が存在してはならないことをそれぞれ示します。したがって、このクエリーでは、「MySQL」という単語は含まれるが、「YourSQL」という単語は含まれないすべての行が取得されます。

```
mysql> SELECT * FROM articles WHERE MATCH (title,body)
  AGAINST ('+MySQL -YourSQL' IN BOOLEAN MODE);
+-----+-----+-----+
| id | title                | body                |
+-----+-----+-----+
| 1 | MySQL Tutorial       | DBMS stands for DataBase ... |
| 2 | How To Use MySQL Well | After you went through a ... |
| 3 | Optimizing MySQL    | In this tutorial, we show ... |
| 4 | 1001 MySQL Tricks   | 1. Never run mysqld as root. 2. ... |
| 6 | MySQL Security      | When configured properly, MySQL ... |
+-----+-----+-----+
```

注記

この機能を実装すると、MySQL では暗黙的ブール論理とも呼ばれる次のようなものが使用されます。

- **+** は **AND** を表します
- **-** は **NOT** を表します
- [演算子なし] は暗黙的に **OR** を表します

ブール全文検索には、次のような特徴があります。

- 行は自動的に関連性の降順にソートされません。
- **InnoDB** テーブルでブールクエリーを実行するには、**MATCH()** 式のすべてのカラム上に **FULLTEXT** インデックスが必要です。 **MyISAM** 検索インデックスに対するブールクエリーは、**FULLTEXT** インデックスなしでも機能します。ただし、この方法で実行される検索の速度は、非常に遅くなります。
- 単語の最小長および最大長の全文パラメータは、組み込み **FULLTEXT** パーサーおよび **MeCab** パーサープラグインを使用して作成された **FULLTEXT** インデックスに適用されます。 **innodb_ft_min_token_size** および **innodb_ft_max_token_size** は **InnoDB** 検索インデックスに使用されます。 **ft_min_word_len** および **ft_max_word_len** は **MyISAM** 検索インデックスに使用されます。
単語の最小長および最大長の全文パラメータは、**ngram** パーサーを使用して作成された **FULLTEXT** インデックスには適用されません。 **ngram** トークンサイズは、**ngram_token_size** オプションで定義されます。
- ストップワードリストが適用されます。これらは、**InnoDB** 検索インデックスの場合は **innodb_ft_enable_stopword**、**innodb_ft_server_stopword_table**、および **innodb_ft_user_stopword_table**、**MyISAM** 検索インデックスの場合は **ft_stopword_file** によって制御されます。
- **InnoDB** の全文検索では、**'++apple'** の例と同様に、単一の検索単語で複数の演算子を使用するようサポートされていません。単一の検索語で複数の演算子を使用すると、標準出力に構文エラーが返されます。 **MyISAM** 全文検索では、同じ検索が正常に処理され、検索語に隣接する演算子を除くすべての演算子が無視されます。

- InnoDB の全文検索では、先頭のプラス記号またはマイナス記号のみがサポートされています。たとえば、InnoDB では '+apple' がサポートされますが、'apple+' はサポートされていません。末尾にプラス記号またはマイナス記号を指定すると、InnoDB で構文エラーがレポートされます。
- InnoDB の全文検索では、ワイルドカード ('+') を使用した先頭のプラス記号、プラス記号とマイナス記号の組み合わせ ('+-'), または先頭のプラス記号とマイナス記号の組み合わせ ('+apple') はサポートされていません。このような無効なクエリーでは、構文エラーが返されます。
- InnoDB 全文検索では、ブール全文検索での @記号の使用はサポートされていません。@記号は、@distance 近接検索演算子で使用するために予約されています。
- MyISAM 検索インデックスに適用される 50% のしきい値は使用されません。

ブール全文検索機能では、次の演算子がサポートされています。

- +

先頭または末尾のプラス記号は、この単語が返される各行に存在しなければならないことを示します。InnoDB では、先頭のプラス記号のみがサポートされています。

- -

先頭または末尾のマイナス記号は、この単語が返される行のいずれにも存在してはならないことを示します。InnoDB では、先頭のマイナス記号のみがサポートされています。

注: - 演算子は、本来ならほかの検索語句で一致が行われる行を除外することのみに使用します。したがって、- の前にある検索語句のみを含むブールモードの検索では、空の結果が返されます。「除外された検索語句のいずれかを含む行を除いたすべての行」が返されるわけではありません。

- (演算子なし)

デフォルトでは (+ と - のどちらも指定されてない場合)、この単語はオプションですが、それを含む行の評価は高くなります。これは、IN BOOLEAN MODE 修飾子なしの MATCH() ... AGAINST() の動作と似ています。

- @distance

この演算子は、InnoDB テーブルでのみ機能します。2 つ以上の単語がすべて、相互に指定された距離内で始まっているかどうか単語単位でテストされます。@distance 演算子の直前に、二重引用符で囲まれた文字列内の検索単語を指定します (たとえば、MATCH(col1) AGAINST("word1 word2 word3" @8' IN BOOLEAN MODE))。

- > <

これらの 2 つの演算子は、行に割り当てられた関連性の値への単語の貢献度を変更する際に使用されます。> 演算子は貢献度を上げ、< 演算子は貢献度を下げます。次のリストのあとに示す例を参照してください。

- ()

丸括弧は、単語を部分式にグループ化します。丸括弧で囲まれたグループはネストできます。

- ~

先頭のチルダは否定演算子として機能するため、行の関連性への単語の貢献度がマイナスになります。これは、「ノイズ」単語にマークを付ける際に便利です。このような単語を含む行は、その他よりも低く評価されますが、- 演算子を使用した場合のように、完全に除外されることはありません。

- *

アスタリスクは、切り捨て (またはワイルドカード) 演算子として機能します。その他の演算子とは異なり、影響を受ける単語に追加されます。* 演算子の前の単語で始まれば、単語が一致します。

切り捨て演算子を付けて単語が指定されている場合は、その単語が短すぎたり、ストップワードであったりしても、ブールクエリーから削除されません。単語が短すぎるかどうかは、InnoDB テーブルの場合は innodb_ft_min_token_size 設定、MyISAM テーブルの場合は ft_min_word_len によって判断されます。これらのオプションは、ngram パーサーを使用する FULLTEXT インデックスには適用されません。

ワイルドカード単語は、1つ以上の単語の先頭に存在しなければならないプリフィクスとみなされます。単語の最小長が4である場合は、'+word +the*' の検索では、2番目のクエリーで短すぎる検索語句 `the` が無視されるため、'+word +the' の検索よりも少ない行が返される可能性があります。

• "

二重引用符 (") 文字内で囲まれたフレーズは、入力されたそのままのフレーズを含む行にのみ一致します。全文エンジンでは、フレーズが複数の単語に分割され、それらの単語の `FULLTEXT` インデックス内で検索が実行されます。単語以外の文字は、正確に一致する必要がありません。フレーズ検索では、そのフレーズとまったく同じ単語が同じ順序で一致に含まれることのみが必要です。たとえば、"`test phrase`" は "`test, phrase`" と一致します。

フレーズにインデックス内にある単語が含まれない場合は、結果が空になります。単語がテキスト内に存在しない場合、ストップワードである場合、またはインデックス付きの単語の最小長よりも短い場合の組み合わせが原因で、単語がインデックス内に存在しない可能性があります。

次の例では、ブール全文演算子を使用する一部の検索文字列を実演します。

• '`apple banana`'

2つの単語の1つ以上を含む行を検索します。

• '+`apple +juice`'

両方の単語を含む行を検索します。

• '+`apple macintosh`'

単語「apple」を含む行を検索しますが、行に「macintosh」も含まれる場合は行を高く評価されます。

• '+`apple -macintosh`'

単語「apple」を含むが、「macintosh」は含まない行を検索します。

• '+`apple ~macintosh`'

単語「apple」を含む行を検索しますが、行に単語「macintosh」も含まれる場合は、行に含まれない場合よりも低く評価されます。これは、「macintosh」が存在すると、行がまったく返されない '+`apple -macintosh`' の検索よりも「ソフト」です。

• '+`apple (>turnover <strudel)`'

単語「apple」と「turnover」、または「apple」と「strudel」（順序は不問）を含む行を検索しますが、「apple turnover」を「apple strudel」よりも高く評価します。

• '`apple*`'

「apple」、「apples」、「applesauce」、「applet」などの単語を含む行を検索します。

• '"`some words`"

「some words」とまったく同じフレーズを含む行を検索します (たとえば、「some words of wisdom」を含む行は検索しますが、「some noise words」は検索しません)。句を囲む"文字は、句を区切る演算子文字であることに注意してください。検索文字列自体を囲む引用符ではありません。

InnoDB ブールモード検索の関連性ランキング

InnoDB の全文検索は、`Sphinx` の全文検索エンジンをモデルにし、使用されるアルゴリズムは、`BM25` および `TF-IDF` のランキングアルゴリズムに基づいています。このような理由のため、InnoDB のブール全文検索の関連性ランキングは、`MyISAM` の関連性ランキングと異なる場合があります。

InnoDB では、「term frequency-inverse document frequency」(`TF-IDF`) 重み付けシステムの偏差を使用して、指定された全文検索クエリーのドキュメントの関連性にランクが付けられます。`TF-IDF` の重み付けは、ドキュメントで単語が出現する頻度に基づき、コレクション内のすべてのドキュメントで単語が出現する頻度によってオフセットされ

ます。言い換えると、ある単語がドキュメントで出現する頻度が高くなるほど、その単語がドキュメントコレクションで出現する頻度が低くなり、ドキュメントのランクが高くなります。

関連性ランキングの計算方法

単語の出現頻度 (TF) 値は、単語がドキュメントで出現する回数です。単語の逆文書頻度 (IDF) 値は、次の公式を使用して計算されます。ここで、`total_records` はコレクション内のレコード数、`matching_records` は検索語句が表示されるレコード数です。

```
$(IDF) = log10( ${total_records} / ${matching_records} )
```

ドキュメントに単語が複数回含まれる場合は、IDF 値が TF 値で乗算されます。

```
$(TF) * $(IDF)
```

TF および IDF 値を使用する場合は、ドキュメントの関連性ランキングが次の公式を使用して計算されます。

```
$(rank) = $(TF) * $(IDF) * $(IDF)
```

公式については、次の例で実演されています。

単一単語検索の関連性ランキング

この例では、単一単語検索の関連性ランキングの計算を実演します。

```
mysql> CREATE TABLE articles (
id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
title VARCHAR(200),
body TEXT,
FULLTEXT (title,body)
) ENGINE=InnoDB;
Query OK, 0 rows affected (1.04 sec)

mysql> INSERT INTO articles (title,body) VALUES
('MySQL Tutorial','This database tutorial ...'),
('How To Use MySQL','After you went through a ...'),
('Optimizing Your Database','In this database tutorial ...'),
('MySQL vs. YourSQL','When comparing databases ...'),
('MySQL Security','When configured properly, MySQL ...'),
('Database, Database, Database','database database database'),
('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
('MySQL Full-Text Indexes','MySQL fulltext indexes use a ...');
Query OK, 8 rows affected (0.06 sec)
Records: 8 Duplicates: 0 Warnings: 0

mysql> SELECT id, title, body, MATCH (title,body) AGAINST ('database' IN BOOLEAN MODE)
AS score FROM articles ORDER BY score DESC;
+----+-----+-----+-----+
| id | title                | body                | score          |
+----+-----+-----+-----+
| 6  | Database, Database, Database | database database database | 1.0886961221694946 |
| 3  | Optimizing Your Database | In this database tutorial ... | 0.36289870738983154 |
| 1  | MySQL Tutorial          | This database tutorial ... | 0.18144935369491577 |
| 2  | How To Use MySQL        | After you went through a ... | 0 |
| 4  | MySQL vs. YourSQL       | When comparing databases ... | 0 |
| 5  | MySQL Security          | When configured properly, MySQL ... | 0 |
| 7  | 1001 MySQL Tricks       | 1. Never run mysqld as root. 2. ... | 0 |
| 8  | MySQL Full-Text Indexes | MySQL fulltext indexes use a .. | 0 |
+----+-----+-----+-----+
8 rows in set (0.00 sec)
```

合計で 8 つのレコードがあり、そのうち 3 つが「database」という検索語句に一致します。1 つ目のレコード (id 6) には、検索語句が 6 回含まれ、関連性ランキングは **1.0886961221694946** です。このランキング値は、6 の TF 値 (レコード id 6 には「database」という検索語句が 6 回出現します)、および次のように計算される 0.42596873216370745 の IDF 値 (ここで、8 はレコードの合計数、3 は検索語句が出現するレコードの数) を使用して計算されます。

```
$(IDF) = log10( 8 / 3 ) = 0.42596873216370745
```


その後、TF および IDF 値はランキング公式に入力されます。

$$\text{\$rank} = \text{\$TF} * \text{\$IDF} * \text{\$IDF}$$

MySQL コマンド行クライアントで計算を実行すると、1.088696164686938 のランキング値が返されます。

```
mysql> SELECT 6*log10(8/3)*log10(8/3);
+-----+
| 6*log10(8/3)*log10(8/3) |
+-----+
| 1.088696164686938 |
+-----+
1 row in set (0.00 sec)
```

注記

SELECT ... MATCH ... AGAINST ステートメントと MySQL コマンド行クライアントで返されるランキング値 (1.0886961221694946 と 1.088696164686938) に、わずかな相違がある場合があります。この相違は、整数と浮動小数点/倍精度間のキャストが (関連する精度および丸めの決定とともに) InnoDB によって内部で実行される方法、およびその他の場所 (MySQL コマンド行クライアントやその他のタイプの計算機など) で実行される方法が原因で発生します。

複数単語検索の関連性ランキング

この例では、以前の例で使用された `articles` テーブルおよびデータに基づいて、複数単語の全文検索の関連性ランキングの計算を実演します。

複数の単語で検索する場合は、次の公式に示すように、関連性ランキングの値が各単語の関連性ランキングの合計になります。

$$\text{\$rank} = \text{\$TF} * \text{\$IDF} * \text{\$IDF} + \text{\$TF} * \text{\$IDF} * \text{\$IDF}$$

2 つの語句 ('mysql tutorial') で検索を実行すると、次の結果が返されます。

```
mysql> SELECT id, title, body, MATCH (title,body) AGAINST ('mysql tutorial' IN BOOLEAN MODE)
AS score FROM articles ORDER BY score DESC;
+-----+-----+-----+
| id | title | body | score |
+-----+-----+-----+
| 1 | MySQL Tutorial | This database tutorial ... | 0.7405621409416199 |
| 3 | Optimizing Your Database | In this database tutorial ... | 0.3624762296676636 |
| 5 | MySQL Security | When configured properly, MySQL ... | 0.031219376251101494 |
| 8 | MySQL Full-Text Indexes | MySQL fulltext indexes use a ... | 0.031219376251101494 |
| 2 | How To Use MySQL | After you went through a ... | 0.015609688125550747 |
| 4 | MySQL vs. YourSQL | When comparing databases ... | 0.015609688125550747 |
| 7 | 1001 MySQL Tricks | 1. Never run mysqld as root. 2. ... | 0.015609688125550747 |
| 6 | Database, Database, Database | database database database | 0 |
+-----+-----+-----+
8 rows in set (0.00 sec)
```

1 つ目のレコード (id 8) では、'mysql' が 1 回出現し、'tutorial' が 2 回出現します。'mysql' に一致するレコードは 6 つ、'tutorial' に一致するレコードは 2 つあります。MySQL コマンド行クライアントでは、これらの値を複数単語検索のランキング公式に挿入するときに、予期されるランキング値が返されます。

```
mysql> SELECT (1*log10(8/6)*log10(8/6)) + (2*log10(8/2)*log10(8/2));
+-----+
| (1*log10(8/6)*log10(8/6)) + (2*log10(8/2)*log10(8/2)) |
+-----+
| 0.7405621541938003 |
+-----+
1 row in set (0.00 sec)
```

注記

上記の例では、**SELECT ... MATCH ... AGAINST** ステートメントと MySQL コマンド行クライアントで返されるランキング値に、わずかな相違があることについて説明しました。

12.10.3 クエリー拡張を使用した全文検索

全文検索では、クエリー拡張 (特に、そのバリエーションの「ブラインドクエリー拡張」) がサポートされています。一般に、これは検索フレーズが短すぎる時に役立ちます。つまり、ユーザーは多くの場合暗黙的な知識に依存しますが、全文検索エンジンにはこれが不足していることを意味します。たとえば、ユーザーが「database」を検索することは、実際は「MySQL」、「Oracle」、「DB2」、および「RDBMS」がすべて、「databases」に一致して返されるはずのフレーズであることを意味する場合があります。これが暗黙的な知識です。

ブラインドクエリー拡張 (自動関連性フィードバックとも呼ばれる) は、検索フレーズのあとに **WITH QUERY EXPANSION** または **IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION** を追加することで有効になります。これは、検索を 2 回実行することで機能します。2 回目の検索での検索フレーズは、1 回目の検索でのもっとも関連性の高い数個のドキュメントと連結されたオリジナルの検索フレーズです。したがって、これらのドキュメントのいずれかに単語「databases」および単語「MySQL」が含まれている場合は、単語「database」が含まれていなくても、2 回目の検索で単語「MySQL」を含むドキュメントが検索されます。次の例では、この相違点を示します。

```
mysql> SELECT * FROM articles
  WHERE MATCH (title,body)
  AGAINST ('database' IN NATURAL LANGUAGE MODE);
+-----+-----+
| id | title | body |
+-----+-----+
| 1 | MySQL Tutorial | DBMS stands for DataBase ... |
| 5 | MySQL vs. YourSQL | In the following database comparison ... |
+-----+-----+
2 rows in set (0.00 sec)

mysql> SELECT * FROM articles
  WHERE MATCH (title,body)
  AGAINST ('database' WITH QUERY EXPANSION);
+-----+-----+
| id | title | body |
+-----+-----+
| 5 | MySQL vs. YourSQL | In the following database comparison ... |
| 1 | MySQL Tutorial | DBMS stands for DataBase ... |
| 3 | Optimizing MySQL | In this tutorial we show ... |
| 6 | MySQL Security | When configured properly, MySQL ... |
| 2 | How To Use MySQL Well | After you went through a ... |
| 4 | 1001 MySQL Tricks | 1. Never run mysqld as root. 2. ... |
+-----+-----+
6 rows in set (0.00 sec)
```

もう 1 つの例では、ユーザーが「Maigret」のスペルに自信がないときに、Georges Simenon 著の Maigret に関する書籍を検索しています。クエリー拡張を使用しなければ、「Megre and the reluctant witnesses」を検索しても「Maigret and the Reluctant Witnesses」のみが検索されます。クエリー拡張を使用すれば、2 回目の検索で単語「Maigret」を含むすべての書籍が検索されます。

注記

ブラインドクエリー拡張には、関連性のないドキュメントが返されるとノイズが大幅に増加する傾向があるため、検索フレーズが短すぎる場合にのみ使用してください。

12.10.4 全文ストップワード

サーバー文字セットおよび照合順序 (`character_set_server` および `collation_server` システム変数の値) を使用すると、全文クエリー用のストップワードリストがロードおよび検索されます。全文インデックス作成または検索で使用されるストップワードファイルまたはカラムに、`character_set_server` または `collation_server` とは異なる文字セットまたは照合順序が含まれている場合は、ストップワード検索で誤ったヒットまたはミスが発生する可能性があります。

ストップワード検索で大文字と小文字が区別されるかどうかは、サーバー照合順序によって異なります。たとえば、照合順序が `utf8mb4_0900_ai_ci` の場合、参照では大/小文字は区別されませんが、照合順序が `utf8mb4_0900_as_cs` または `utf8mb4_bin` の場合、参照では大/小文字が区別されます。

- [InnoDB 検索インデックスのストップワード](#)
- [MyISAM 検索インデックスのストップワード](#)

InnoDB 検索インデックスのストップワード

技術的、文学的、およびその他のソースからのドキュメントでは、キーワードとしてまたは重要なフレーズで短い単語が使用されることが多いため、InnoDB ではデフォルトのストップワードリストが比較的短くなります。たとえば、「to be or not to be」を検索し、これらの単語がすべて無視されるのではなく、適切な結果が取得されることを期待するとします。

デフォルトの InnoDB ストップワードリストを確認するには、`INFORMATION_SCHEMA.INNODB_FT_DEFAULT_STOPWORD` テーブルを問い合わせます。

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_DEFAULT_STOPWORD;
+-----+
| value |
+-----+
| a     |
| about |
| an    |
| are   |
| as    |
| at    |
| be    |
| by    |
| com   |
| de    |
| en    |
| for   |
| from  |
| how   |
| i     |
| in    |
| is    |
| it    |
| la    |
| of    |
| on    |
| or    |
| that  |
| the   |
| this  |
| to    |
| was   |
| what  |
| when  |
| where |
| who   |
| will  |
| with  |
| und   |
| the   |
| www   |
+-----+
36 rows in set (0.00 sec)
```

すべての InnoDB テーブルで独自のストップワードリストを定義するには、`INNODB_FT_DEFAULT_STOPWORD` テーブルと同じ構造を持つテーブルを定義し、それにストップワードを移入し、`innodb_ft_server_stopword_table` オプションの値を `db_name/table_name` 形式の値に設定してから、全文インデックスを作成します。ストップワードテーブルには、`value` という名前の単一の `VARCHAR` カラムが含まれている必要があります。次の例では、InnoDB 用に新しいグローバルストップワードテーブルを作成および構成するよう実演します。

```
-- Create a new stopword table

mysql> CREATE TABLE my_stopwords(value VARCHAR(30)) ENGINE = INNODB;
Query OK, 0 rows affected (0.01 sec)

-- Insert stopwords (for simplicity, a single stopword is used in this example)

mysql> INSERT INTO my_stopwords(value) VALUES ('Ishmael');
Query OK, 1 row affected (0.00 sec)

-- Create the table
```

```
mysql> CREATE TABLE opening_lines (
id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
opening_line TEXT(500),
author VARCHAR(200),
title VARCHAR(200)
) ENGINE=InnoDB;
Query OK, 0 rows affected (0.01 sec)

-- Insert data into the table

mysql> INSERT INTO opening_lines(opening_line,author,title) VALUES
('Call me Ishmael.','Herman Melville','Moby-Dick'),
('A screaming comes across the sky.','Thomas Pynchon','Gravity's Rainbow'),
('I am an invisible man.','Ralph Ellison','Invisible Man'),
('Where now? Who now? When now?','Samuel Beckett','The Unnamable'),
('It was love at first sight.','Joseph Heller','Catch-22'),
('All this happened, more or less.','Kurt Vonnegut','Slaughterhouse-Five'),
('Mrs. Dalloway said she would buy the flowers herself.','Virginia Woolf','Mrs. Dalloway'),
('It was a pleasure to burn.','Ray Bradbury','Fahrenheit 451');
Query OK, 8 rows affected (0.00 sec)
Records: 8 Duplicates: 0 Warnings: 0

-- Set the innodb_ft_server_stopword_table option to the new stopwords table

mysql> SET GLOBAL innodb_ft_server_stopword_table = 'test/my_stopwords';
Query OK, 0 rows affected (0.00 sec)

-- Create the full-text index (which rebuilds the table if no FTS_DOC_ID column is defined)

mysql> CREATE FULLTEXT INDEX idx ON opening_lines(opening_line);
Query OK, 0 rows affected, 1 warning (1.17 sec)
Records: 0 Duplicates: 0 Warnings: 1
```

[INFORMATION_SCHEMA.INNOODB_FT_INDEX_TABLE](#) で単語を問い合わせて、指定したストップワード ('Ishmael') が表示されないことを確認します。

注記

デフォルトでは、長さが 3 文字よりも少ない単語または 84 文字よりも多い単語は、`InnoDB` の全文検索インデックスに表示されません。単語の最大長および最小長の値は、`innodb_ft_max_token_size` および `innodb_ft_min_token_size` 変数を使用して構成できます。このデフォルトの動作は `ngram` パーサープラグインには適用されません。`ngram` トークンサイズは `ngram_token_size` オプションで定義されます。

```
mysql> SET GLOBAL innodb_ft_aux_table='test/opening_lines';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT word FROM INFORMATION_SCHEMA.INNOODB_FT_INDEX_TABLE LIMIT 15;
+-----+
| word  |
+-----+
| across |
| all   |
| burn  |
| buy   |
| call  |
| comes |
| dalloway |
| first |
| flowers |
| happened |
| herself |
| invisible |
| less  |
| love  |
| man   |
+-----+
15 rows in set (0.00 sec)
```

ストップワードリストをテーブルごとに作成するには、その他のストップワードテーブルを作成し、`innodb_ft_user_stopword_table` オプションを使用して使用されるストップワードテーブルを指定してから、全文インデックスを作成します。

MyISAM 検索インデックスのストップワード

ストップワードファイルは、`character_set_server` が `ucs2`, `utf16`, `utf16le` または `utf32` の場合、`latin1` を使用してロードおよび検索されます。

MyISAM テーブル用のデフォルトのストップワードリストをオーバーライドするには、`ft_stopword_file` システム変数を設定します。(セクション5.1.8「サーバシステム変数」を参照してください。) 変数の値は、ストップワードリストを含むファイルのパス名、またはストップワードのフィルタ処理が無効になる空の文字列になるようにしてください。サーバーは、別のディレクトリを指定する絶対パス名が指定されないかぎり、データディレクトリ内のファイルを検索します。この変数の値またはストップワードファイルの内容を変更したら、サーバーを再起動し、`FULLTEXT` インデックスを再構築してください。

ストップワードリストは自由形式で、改行、空白、カンマなどの英数字以外の文字でストップワードが区切られます。例外は、アンダースコア文字 (`_`) および単語の一部として扱われる単一のアポストロフィ (`'`) です。ストップワードリストの文字セットは、サーバーのデフォルト文字セットです。セクション10.3.2「サーバー文字セットおよび照合順序」を参照してください。

次のリストに、MyISAM 検索インデックスのデフォルトのストップワードを示します。このリストは、MySQL ソース配布の `storage/myisam/ft_static.c` ファイルで検索できます。

```
a's    able    about   above   according
accordingly across  actually after   afterwards
again  against ain't   all     allow
allows almost alone   along   already
also   although always  am      among
amongst an      and     another any
anybody anyhow  anyone  anything anyway
anyways anywhere  apart   appear  appreciate
appropriate are     aren't  around  as
aside  ask     asking  associated at
available away    awfully be       became
because become  becomes becoming been
before beforehand behind  being   believe
below  beside besides best    better
between beyond  both    brief   but
by     c'mon  c's     came    can
can't  cannot cant    cause   causes
certain certainly changes clearly  co
com     come   comes   concerning consequently
consider considering contain containing contains
corresponding could  couldn't course  currently
definitely described  despite did      didn't
different do     does   doesn't doing
don't  done  down   downwards during
each   edu   eg     eight   either
else   elsewhere enough  entirely especially
et     etc    even   ever    every
everybody everyone everything everywhere ex
exactly example  except  far     few
fifth  first  five   followed following
follows for     former  formerly forth
four   from   further furthermore get
gets   getting given   gives   go
goes   going  gone    got     gotten
greetings had    hadn't  happens hardly
has    hasn't have    haven't having
he     he's   hello   help    hence
her    here   here's  hereafter hereby
herein hereupon hers    herself hi
him    himself his     hither hopefully
how    howbeit however i'd     i'll
i'm    i've   ie      if      ignored
immediate in     inasmuch inc     indeed
indicate indicated indicates inner   insofar
instead into   inward  is     isn't
it     it'd   it'll  it's   its
itself just   keep   keeps  kept
know   known  knows  last   lately
later  latter  latterly least  less
lest   let    let's  like   liked
likely little  look   looking looks
```

ltd	mainly	many	may	maybe
me	mean	meanwhile	merely	might
more	moreover	most	mostly	much
must	my	myself	name	namely
nd	near	nearly	necessary	need
needs	neither	never	nevertheless	new
next	nine	no	nobody	non
none	noone	nor	normally	not
nothing	novel	now	nowhere	obviously
of	off	often	oh	ok
okay	old	on	once	one
ones	only	onto	or	other
others	otherwise	ought	our	ours
ourselves	out	outside	over	overall
own	particular	particularly	per	perhaps
placed	please	plus	possible	presumably
probably	provides	que	quite	qv
rather	rd	re	really	reasonably
regarding	regardless	regards	relatively	respectively
right	said	same	saw	say
saying	says	second	secondly	see
seeing	seem	seemed	seeming	seems
seen	self	selves	sensible	sent
serious	seriously	seven	several	shall
she	should	shouldn't	since	six
so	some	somebody	somehow	someone
something	sometime	sometimes	somewhat	somewhere
soon	sorry	specified	specify	specifying
still	sub	such	sup	sure
t's	take	taken	tell	tends
th	than	thank	thanks	thax
that	that's	thats	the	their
theirs	them	themselves	then	thence
there	there's	thereafter	thereby	therefore
therein	theres	thereupon	these	they
they'd	they'll	they're	they've	think
third	this	thorough	thoroughly	those
though	three	through	throughout	thru
thus	to	together	too	took
toward	towards	tried	tries	truly
try	trying	twice	two	un
under	unfortunately	unless	unlikely	until
unto	up	upon	us	use
used	useful	uses	using	usually
value	various	very	via	viz
vs	want	wants	was	wasn't
way	we	we'd	we'll	we're
we've	welcome	well	went	were
weren't	what	what's	whatever	when
whence	whenever	where	where's	whereafter
whereas	whereby	wherein	whereupon	wherever
whether	which	while	whither	who
who's	whoever	whole	whom	whose
why	will	willing	wish	with
within	without	won't	wonder	would
wouldn't	yes	yet	you	you'd
you'll	you're	you've	your	yours
yourself	yourselves	zero		

12.10.5 全文制限

- 全文検索は、[InnoDB](#) および [MyISAM](#) テーブルでのみサポートされています。
- パーティション化されたテーブルでは、全文検索がサポートされていません。 [セクション24.6「パーティショニングの制約と制限」](#)を参照してください。
- 全文検索は、ほとんどのマルチバイト文字セットで使用できます。例外として、Unicode では、[utf8](#) 文字セットは使用できますが、[ucs2](#) 文字セットは使用できません。 [ucs2](#) カラム上では [FULLTEXT](#) インデックスを使用できませんが、このようなインデックスが含まれない [ucs2](#) 上で [IN BOOLEAN MODE](#) 検索を実行することはできます。

[utf8](#) に関する備考は [utf8mb4](#) にも適用され、[ucs2](#) に関する備考は [utf16](#)、[utf16le](#)、および [utf32](#) にも適用されません。

- 中国語や日本語のような表意文字を使用する言語には、単語の区切り文字がありません。したがって、組込みの全文パーサーこれらの言語および他の言語で単語の開始位置と終了位置を判別できません。

中国語、日本語および韓国語 (CJK) をサポートする文字ベースの ngram 全文パーサーと、日本語をサポートするワードベースの MeCab パーサープラグインが、InnoDB および MyISAM テーブルで使用するために提供されています。
- 単一テーブル内で複数の文字セットを使用することはサポートされていますが、FULLTEXT インデックス内のすべてのコラムで同じ文字セットおよび照合順序が使用される必要があります。
- MATCH() コラムリストは、この MATCH() が MyISAM テーブル上の IN BOOLEAN MODE である場合を除いて、テーブルの一部の FULLTEXT インデックス定義に含まれるコラムリストに完全に一致する必要があります。MyISAM テーブルでは、インデックスが付いていないコラムでもブールモード検索を実行できます。ただし、検索が遅くなる可能性があります。
- AGAINST() への引数は、クエリー評価時に定数である文字列値にする必要があります。たとえば、テーブルコラムは、行ごとに異なる可能性があるため除外されます。
- FULLTEXT 以外の検索の場合よりも FULLTEXT 検索の場合の方が、インデックスヒントに対する制限が多くなります。セクション8.9.4「インデックスヒント」を参照してください。
- InnoDB では、全文インデックスを含むコラムを必要とするすべての DML 演算子 (INSERT、UPDATE、DELETE) は、トランザクションのコミット時に処理されます。たとえば、INSERT 演算では、挿入された文字列がトークン化され、個々の単語に分解されます。その後、個々の単語は、トランザクションのコミット時に全文インデックステーブルに追加されます。その結果、全文検索ではコミットされたデータのみが返されます。
- '%'文字は全文検索でサポートされているワイルドカード文字ではありません。

12.10.6 MySQL の全文検索の微調整

MySQL の全文検索機能には、ユーザーが調整できるパラメータがほとんどありません。一部の変更でソースコードを変更する必要があるために、MySQL ソース配布を持っている場合は、全文検索の動作をさらに制御できます。セクション2.9「ソースから MySQL をインストールする」を参照してください。

全文検索の有効性は、慎重に調整されます。ほとんど場合、デフォルトの動作を変更すると、実際には有効性が低くなる可能性があります。使用方法を理解していない場合は、MySQL ソースは変更しないでください。

このセクションで説明するほとんどの全文変数は、サーバーの起動時に設定する必要があります。変更するには、サーバーの再起動が必要です。サーバーが動作しているときは、変更できません。

一部の変数を変更するには、テーブル内の FULLTEXT インデックスを再構築する必要があります。これを行う手順については、このセクションの後半で説明します。

- [単語の最小長と最大長の構成](#)
- [自然言語検索のしきい値の構成](#)
- [ブール全文検索演算子の変更](#)
- [文字セットの変更](#)
- [InnoDB 全文インデックスの再構築](#)
- [InnoDB 全文インデックスの最適化](#)
- [MyISAM 全文インデックスの再構築](#)

単語の最小長と最大長の構成

インデックスが付けられる単語の最小長および最大長は、InnoDB 検索インデックスの場合は `innodb_ft_min_token_size` および `innodb_ft_max_token_size`、MyISAM 検索インデックスの場合は `ft_min_word_len` および `ft_max_word_len` で定義されます。

注記

単語の最小長および最大長の全文パラメータは、ngram パーサーを使用して作成された **FULLTEXT** インデックスには適用されません。ngram トークンサイズは、`ngram_token_size` オプションで定義されます。

これらのオプションのいずれかを変更したら、変更を有効にするために **FULLTEXT** インデックスを再構築してください。たとえば、2 文字の単語を検索可能にするには、オプションファイルに次の行を配置します。

```
[mysqld]
innodb_ft_min_token_size=2
ft_min_word_len=2
```

次に、サーバーを再起動し、**FULLTEXT** インデックスを再構築します。MyISAM テーブルについては、MyISAM の全文インデックスを再構築する際に従う手順で、`mysamchk` に関する備考に注意してください。

自然言語検索のしきい値の構成

MyISAM 検索インデックスでは、選択された特定の重み付けスキームによって、自然言語検索で 50% のしきい値が決定されます。これを無効にするには、`storage/myisam/ftdefs.h` で次の行を検索してください。

```
#define GWS_IN_USE GWS_PROB
```

この行を次のように変更します。

```
#define GWS_IN_USE GWS_FREQ
```

次に、MySQL を再コンパイルします。この場合、インデックスを再構築する必要はありません。

注記

このように変更すると、`MATCH()` 関数に適切な関連性値を提供する MySQL の能力が大幅に低下します。このような一般的な単語を検索する必要がある場合は、代わりに、50% のしきい値に従わない **IN BOOLEAN MODE** を使用して検索する方が適切です。

ブール全文検索演算子の変更

MyISAM テーブル上でブール全文検索に使用される演算子を変更するには、`ft_boolean_syntax` システム変数を設定します。(InnoDB には同等の設定がありません。) この変数は、サーバーの実行中に変更できますが、グローバルシステム変数を設定するのに十分な権限が必要です(セクション 5.1.9.1 「システム変数権限」を参照)。この場合は、インデックスを再構築する必要はありません。

文字セットの変更

組込み全文パーサーでは、次のリストで説明するように、単語文字とみなされる文字のセットをいくつかの方法で変更できます。変更が完了したら、任意の **FULLTEXT** インデックスを含むテーブルごとにインデックスを再構築します。ハイフン文字 ('-') を単語文字として処理すると仮定します。次の方法のいずれかを使用します。

- MySQL ソースを変更します。`storage/innobase/handler/ha_innodb.cc` (InnoDB の場合) または `storage/myisam/ftdefs.h` (MyISAM の場合) で、`true_word_char()` および `misc_word_char()` マクロを参照してください。それらのマクロのいずれかに `'` を追加し、MySQL を再コンパイルします。
- 文字セットファイルを変更します。再コンパイルする必要はありません。`true_word_char()` マクロでは、英数字とその他の文字を区別するために「character type」テーブルが使用されます。文字セット XML ファイルのいずれかで `<ctype><map>` 配列の内容を編集すると、`'` が「英字」になるように指定できます。次に、**FULLTEXT** インデックスに指定された文字セットを使用します。`<ctype><map>` 配列の書式については、セクション 10.13.1 「文字定義配列」を参照してください。
- インデックス付きのカラムで使用される文字セットに新しい照合順序を追加し、その照合順序が使用されるようにカラムを変更します。照合順序の追加に関する一般的な情報については、セクション 10.14 「文字セットへの照合順序の追加」を参照してください。全文インデックス作成に固有の例については、セクション 12.10.7 「全文インデックス付けのためのユーザー定義照合の追加」を参照してください。

InnoDB 全文インデックスの再構築

変更を有効にするには、次の全文インデックス変数のいずれかを変更した後に `FULLTEXT` インデックスを再構築する必要があります: `innodb_ft_min_token_size`; `innodb_ft_max_token_size`; `innodb_ft_server_stopword_table`; `innodb_ft_user_stopword_table`; `innodb_ft_enable_stopword`; `ngram_token_size`。 `innodb_ft_min_token_size`、`innodb_ft_max_token_size` または `ngram_token_size` を変更するには、サーバーを再起動する必要があります。

InnoDB テーブルの `FULLTEXT` インデックスを再構築するには、`DROP INDEX` および `ADD INDEX` オプションを指定して `ALTER TABLE` を使用し、各インデックスを削除して再作成します。

InnoDB 全文インデックスの最適化

全文インデックス付きのテーブル上で `OPTIMIZE TABLE` を実行すると、全文インデックスが再構築され、削除済みのドキュメント ID が削除され、同じ単語に対応する複数のエントリが連結されます (可能な場合)。

全文インデックスを最適化するには、`innodb_optimize_fulltext_only` を有効にして、`OPTIMIZE TABLE` を実行します。

```
mysql> set GLOBAL innodb_optimize_fulltext_only=ON;
Query OK, 0 rows affected (0.01 sec)

mysql> OPTIMIZE TABLE opening_lines;
+-----+-----+-----+
| Table      | Op      | Msg_type | Msg_text |
+-----+-----+-----+
| test.opening_lines | optimize | status  | OK       |
+-----+-----+-----+
1 row in set (0.01 sec)
```

大きなテーブルで全文インデックスの再構築時間が長くなることを回避するには、`innodb_ft_num_word_optimize` オプションを使用すれば、最適化を段階的に実行できます。 `innodb_ft_num_word_optimize` オプションでは、`OPTIMIZE TABLE` が実行されるたびに最適化される単語の数が定義されます。デフォルト設定は 2000 です。これは、`OPTIMIZE TABLE` が実行されるたびに 2000 個の単語が最適化されることを表します。後続の `OPTIMIZE TABLE` 演算は、先行する `OPTIMIZE TABLE` 演算が終了した場所から続行されます。

MyISAM 全文インデックスの再構築

インデックス作成に影響を与える全文変数 (`ft_min_word_len`、`ft_max_word_len`、または `ft_stopword_file`) を変更する場合や、ストップワードファイル自体を変更する場合は、変更して、サーバーを再起動したあとに、`FULLTEXT` インデックスを再構築する必要があります。

MyISAM テーブルの `FULLTEXT` インデックスを再構築するには、`QUICK` 修復演算を実行すれば十分です。

```
mysql> REPAIR TABLE tbl_name QUICK;
```

または、先ほど説明した `ALTER TABLE` を使用します。これは、修復演算よりも高速になる可能性もあります。

任意の `FULLTEXT` インデックスを含む各テーブルは、上記のように修復する必要があります。そうしないと、テーブルのクエリーによって誤った結果が生成される可能性があり、テーブルを変更すると、サーバーはテーブルが破損して修復が必要であることを確認します。

`myisamchk` を使用して、MyISAM テーブルインデックスを変更する演算 (修復や分析など) を実行する場合は、ほかに指定がなければ、単語の最小長、単語の最大長、およびストップワードファイルのデフォルトの全文パラメータ値を使用して、`FULLTEXT` インデックスが再構築されます。これにより、クエリーに失敗する可能性があります。

この問題は、これらのパラメータがサーバーでのみ認識されていることが原因で発生します。MyISAM インデックスファイルには格納されていません。サーバーで使用される単語の最小長や最大長、またはストップワードファイルの値を変更した場合の問題を回避するには、`mysqld` で使用される `myisamchk` と同じ `ft_min_word_len`、`ft_max_word_len`、および `ft_stopword_file` 値を指定します。たとえば、単語の最小長を 3 に設定した場合は、次のように `myisamchk` を使用してテーブルを修復できます。

```
myisamchk --recover --ft_min_word_len=3 tbl_name.MYI
```

`myisamchk` およびサーバーで全文パラメータに必ず同じ値が使用されるようにするには、オプションファイルの `[mysqld]` と `[myisamchk]` の両方のセクションにそれぞれを配置してください。

```
[mysql]
ft_min_word_len=3

[myisamchk]
ft_min_word_len=3
```

MyISAM テーブルインデックスの変更に `myisamchk` を使用する方法の代替として、`REPAIR TABLE`、`ANALYZE TABLE`、`OPTIMIZE TABLE`、または `ALTER TABLE` ステートメントを使用します。これらのステートメントは、適切に使用される全文パラメータ値が認識されているサーバーで実行されます。

12.10.7 全文インデックス付けのためのユーザー定義照合の追加

このセクションでは、組込み全文パーサーを使用して全文検索用のユーザー定義照合を追加する方法について説明します。サンプルの照合順序は `latin1_swedish_ci` と似ていますが、`'` 文字は、単語文字としてインデックスを付けることができるように、句読文字としてではなく英字として処理されます。照合順序の追加に関する一般的な情報については、[セクション10.14「文字セットへの照合順序の追加」](#)で説明されています。この情報を参照し、関与するファイルをよく理解することが前提となっています。

全文インデックス付けの照合を追加するには、次の手順を使用します。ここで説明する手順では、[セクション10.14「文字セットへの照合順序の追加」](#)で説明されているように、文字セットプロパティを記述する構成ファイルを使用して、単純な文字セットの照合順序を作成できます。Unicode などの複雑な文字セットの場合は、文字セットプロパティを記述する C ソースファイルを使用して照合を作成します。

1. 照合順序を `Index.xml` ファイルに追加します。ユーザー定義照合に許可される ID の範囲は、[セクション10.14.2「照合順序 ID の選択」](#)で指定されます。ID は未使用である必要があるため、その ID ガシシステムですでに使用されている場合は 1025 以外の値を選択してください。

```
<charset name="latin1">
...
<collation name="latin1_fulltext_ci" id="1025"/>
</charset>
```

2. `latin1.xml` ファイルで照合順序のソート順序を宣言します。この場合、`latin1_swedish_ci` から順序をコピーできます。

```
<collation name="latin1_fulltext_ci">
<map>
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F
60 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
50 51 52 53 54 55 56 57 58 59 5A 7B 7C 7D 7E 7F
80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F
90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF
41 41 41 41 5C 5B 5C 43 45 45 45 49 49 49 49
44 4E 4F 4F 4F 4F 5D D7 D8 55 55 55 59 59 DE DF
41 41 41 41 5C 5B 5C 43 45 45 45 49 49 49 49
44 4E 4F 4F 4F 4F 5D F7 D8 55 55 55 59 59 DE FF
</map>
</collation>
```

3. `latin1.xml` で `ctype` 配列を変更します。0x2D (`'` 文字のコード) に対応する値を 10 (句読点) から 01 (大文字) に変更します。次の配列では、これは 4 行目の要素で、最後から 3 番目の値です。

```
<ctype>
<map>
00
20 20 20 20 20 20 20 20 28 28 28 28 28 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
48 10 10 10 10 10 10 10 10 10 10 10 01 10 10
84 84 84 84 84 84 84 84 84 10 10 10 10 10
10 81 81 81 81 81 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01 10 10 10 10
10 82 82 82 82 82 02 02 02 02 02 02 02
```

```
02 02 02 02 02 02 02 02 02 02 10 10 10 10 20
10 00 10 02 10 10 10 10 10 10 01 10 01 00 01 00
00 10 10 10 10 10 10 10 10 02 10 02 00 02 01
48 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 10 01 01 01 01 01 01 01 02
02 02 02 02 02 02 02 02 02 02 02 02 02 02
02 02 02 02 02 02 10 02 02 02 02 02 02 02
</map>
</ctype>
```

4. サーバーを再起動します。
5. 新しい照合順序を使用するには、使用されるコラムの定義に追加します。

```
mysql> DROP TABLE IF EXISTS t1;
Query OK, 0 rows affected (0.13 sec)

mysql> CREATE TABLE t1 (
  a TEXT CHARACTER SET latin1 COLLATE latin1_fulltext_ci,
  FULLTEXT INDEX(a)
) ENGINE=InnoDB;
Query OK, 0 rows affected (0.47 sec)
```

6. 照合順序をテストして、ハイフンが単語文字としてみなされることを確認します。

```
mysql> INSERT INTO t1 VALUES ('----'),('...'),('abcd');
Query OK, 3 rows affected (0.22 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM t1 WHERE MATCH a AGAINST ('----' IN BOOLEAN MODE);
+-----+
| a |
+-----+
|----|
+-----+
1 row in set (0.00 sec)
```

12.10.8 ngram 全文パーサー

組み込みの MySQL 全文パーサーは、単語間の空白をデリミタとして使用して、単語の開始位置と終了位置を決定します。これは、単語デリミタを使用しない表意文字言語を使用する場合の制限です。この制限に対処するために、MySQL には、中国語、日本語および韓国語 (CJK) をサポートする ngram 全文パーサーが用意されています。ngram 全文パーサーは、InnoDB および MyISAM での使用がサポートされています。

注記

MySQL には、ドキュメントを意味のある単語にトークン化する日本語用の MeCab 全文パーサープラグインも用意されています。詳細は、[セクション12.10.9「MeCab フルテキストパーサープラグイン」](#)を参照してください。

ngram は、指定された一連のテキストから連続した一連の n 文字です。ngram パーサーは、一連のテキストを連続した一連の n 文字にトークン化します。たとえば、ngram 全文パーサーを使用して、 n の様々な値の「abcd」をトークン化できます。

```
n=1: 'a', 'b', 'c', 'd'
n=2: 'ab', 'bc', 'cd'
n=3: 'abc', 'bcd'
n=4: 'abcd'
```

ngram 全文パーサーは、組み込みのサーバープラグインです。ほかの組み込みサーバープラグインと同様に、サーバーの起動時に自動的にロードされます。

[セクション12.10「全文検索関数」](#)で説明されている全文検索構文は、ngram パーサープラグインに適用されます。このセクションでは、解析動作の違いについて説明します。単語の最小長および最大長オプション (innodb_ft_min_token_size, innodb_ft_max_token_size, ft_min_word_len, ft_max_word_len) を除く、全文関連の構成オプションも適用できます。

ngram トークンサイズの構成

ngram パーサーのデフォルトの ngram トークンサイズは 2 (bigram) です。たとえば、トークンサイズが 2 の場合、ngram パーサーは文字列「abc def」を 4 つのトークンに解析: 「ab」、「bc」、「de」および「ef」。

ngram トークンサイズは、最小値が 1 で最大値が 10 の `ngram_token_size` 構成オプションを使用して構成できます。

通常、`ngram_token_size` は、検索する最大トークンのサイズに設定されます。単一文字のみを検索する場合は、`ngram_token_size` を 1 に設定します。トークンサイズを小さくすると、全文検索インデックスが小さくなり、検索が高速になります。複数の文字で構成される単語を検索する必要がある場合は、それに応じて `ngram_token_size` を設定します。たとえば、「Happy Birthday」は簡体字中国語の「生日快乐」で、「生日」は「誕生日」で、「快乐」は「幸せ」として翻訳されます。これらのような 2 文字の単語を検索するには、`ngram_token_size` を 2 以上の値に設定します。

読取り専用変数として、`ngram_token_size` は起動文字列の一部または構成ファイルでのみ設定できます:

- 起動文字列:

```
mysqld --ngram_token_size=2
```

- 構成ファイル:

```
[mysqld]  
ngram_token_size=2
```

注記

ngram パーサーを使用する `FULLTEXT` インデックスでは、次の最小および最大ワード長構成オプションは無視されます: `innodb_ft_min_token_size`, `innodb_ft_max_token_size`, `ft_min_word_len` および `ft_max_word_len`。

ngram パーサーを使用する FULLTEXT インデックスの作成

ngram パーサーを使用する `FULLTEXT` インデックスを作成するには、`CREATE TABLE`、`ALTER TABLE` または `CREATE INDEX` とともに `WITH PARSE ngram` を指定します。

次の例では、`ngram FULLTEXT` インデックスを使用したテーブルの作成、サンプルデータの挿入 (簡体字中国語テキスト)、および `INFORMATION_SCHEMA.INNOODB_FT_INDEX_CACHE` テーブルでのトークン化されたデータの表示を示します。

```
mysql> USE test;  
  
mysql> CREATE TABLE articles (  
  id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,  
  title VARCHAR(200),  
  body TEXT,  
  FULLTEXT (title,body) WITH PARSE ngram  
 ) ENGINE=InnoDB CHARACTER SET utf8mb4;  
  
mysql> SET NAMES utf8mb4;  
  
INSERT INTO articles (title,body) VALUES  
 ('数据库管理','在本教程中我将向你展示如何管理数据库'),  
 ('数据库应用开发','学习开发数据库应用程序');  
  
mysql> SET GLOBAL innodb_ft_aux_table="test/articles";  
  
mysql> SELECT * FROM INFORMATION_SCHEMA.INNOODB_FT_INDEX_CACHE ORDER BY doc_id, position;
```

既存のテーブルに `FULLTEXT` インデックスを追加するには、`ALTER TABLE` または `CREATE INDEX` を使用できません。例:

```
CREATE TABLE articles (  
  id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,  
  title VARCHAR(200),  
  body TEXT  
 ) ENGINE=InnoDB CHARACTER SET utf8;  
  
ALTER TABLE articles ADD FULLTEXT INDEX ft_index (title,body) WITH PARSE ngram;
```



```
# Or:  
CREATE FULLTEXT INDEX ft_index ON articles (title,body) WITH PARSER ngram;
```

ngram パーサーの領域処理

ngram パーサーは、解析時にスペースを削除します。例:

- 「ab cd」が「ab」、「cd」に解析されます
- 「a bc」が「bc」に解析されます

ngram パーサーのストップワード処理

組込みの MySQL 全文パーサーは、ワードをストップワードリストのエントリと比較します。ワードがストップワードリストのエントリと等しい場合、そのワードはインデックスから除外されます。ngram パーサーの場合、ストップワード処理は異なる方法で実行されます。ngram パーサーは、ストップワードリストのエントリと等しいトークンを除外するかわりに、次を含むがストップワードするトークンを除外します。たとえば、`ngram_token_size=2` の場合、「a,b」を含むドキュメントは「a,」および「,b」に解析されます。カンマ(「,」)がストップワードとして定義されている場合、「a,」と「,b」の両方にカンマが含まれているため、インデックスから除外されます。

デフォルトでは、ngram パーサーは、英語のストップワードのリストを含むデフォルトのストップワードリストを使用します。中国語、日本語または韓国語に適用可能なストップワードリストの場合は、独自のものを作成する必要があります。ストップワードリストの作成の詳細は、[セクション12.10.4「全文ストップワード」](#)を参照してください。

`ngram_token_size` より長いストップワードは無視されます。

ngram パーサー用語検索

自然言語モード検索の場合、検索語は ngram 語の和集合に変換されます。たとえば、文字列「abc」(`ngram_token_size=2` を想定)は「ab bc」に変換されます。「ab」を含むドキュメントと「abc」を含むドキュメントがある場合、検索語「ab bc」は両方のドキュメントに一致します。

ブールモード検索の場合、検索語は ngram 句検索に変換されます。たとえば、文字列'abc' (`ngram_token_size=2` を想定)は'「ab bc」'に変換されます。'ab'を含むドキュメントと'abc'を含むドキュメントがある場合、検索フレーズ'「ab bc」'は'abc'を含むドキュメントにのみ一致します。

ngram パーサーワイルドカード検索

ngram `FULLTEXT` インデックスには ngrams のみが含まれ、用語の先頭に関する情報は含まれないため、ワイルドカード検索で予期しない結果が返されることがあります。ngram `FULLTEXT` 検索インデックスを使用したワイルドカード検索には、次の動作が適用されます:

- ワイルドカード検索の接頭辞語句が ngram トークンサイズより短い場合、クエリーは接頭辞 term で始まる ngram トークンを含むすべてのインデックス付き行を返します。たとえば、`ngram_token_size=2` の場合、「a*」で検索すると、「a」で始まるすべての行が返されます。
- ワイルドカード検索の接頭辞用語が ngram トークンサイズより長い場合、接頭辞用語は ngram 句に変換され、ワイルドカード演算子は無視されます。たとえば、`ngram_token_size=2` の場合、「abc*」ワイルドカード検索は「ab bc」に変換されます。

ngram パーサーフレーズ検索

フレーズ検索は ngram 句検索に変換されます。たとえば、「abc」という検索フレーズは、「abc」および「ab bc」を含むドキュメントを返す「ab bc」に変換されます。

「abc def」という検索フレーズは、「abc def」および「ab bc de ef」を含むドキュメントを返す「ab bc de ef」に変換されます。「abcdef」を含むドキュメントは返されません。

12.10.9 MeCab フルテキストパーサープラグイン

組込みの MySQL 全文パーサーは、単語間の空白をデリミタとして使用して、単語の開始位置と終了位置を決定します。これは、単語デリミタを使用しない表意文字言語を使用する場合の制限です。日本語のこの制限に対処するため

に、MySQL には MeCab 全文パーサープラグインが用意されています。MeCab 全文パーサープラグインは、InnoDB および MyISAM での使用がサポートされています。

注記

MySQL には、日本語をサポートする ngram 全文パーサープラグインも用意されています。詳細は、[セクション12.10.8「ngram 全文パーサー」](#)を参照してください。

MeCab 全文パーサープラグインは、一連のテキストを意味のある単語にトークン化する日本語の全文パーサープラグインです。たとえば、MeCab は、「データベース管理」(「データベース管理」)を「データベース」(「データベース」)および「管理」(「管理」)にトークン化します。比較すると、ngram 全文パーサーは、テキストを連続した一連の n 文字にトークン化します。ここで、n は 1 から 10 までの数値を表します。

テキストを意味のある単語にトークン化するだけでなく、MeCab インデックスは通常 ngram インデックスよりも小さく、MeCab 全文検索は一般的に高速です。欠点は、ngram 全文パーサーに比べて、MeCab 全文パーサーがドキュメントをトークン化するのに時間がかかることです。

[セクション12.10「全文検索関数」](#)で説明されている全文検索構文は、MeCab パーサープラグインに適用されます。このセクションでは、解析動作の違いについて説明します。全文関連の構成オプションも適用できます。

MeCab パーサーの詳細は、Github の「[MeCab: 他の品詞/形態アナライザ](#)」プロジェクトを参照してください。

MeCab パーサープラグインのインストール

MeCab パーサープラグインには、`mecab` および `mecab-ipadic` が必要です。

サポートされている Fedora、Debian および Ubuntu プラットフォーム (システム `mecab` バージョンが古すぎる Ubuntu 12.04 を除く) では、MySQL はシステム `mecab` インストールがデフォルトの場所にインストールされている場合に、そのインストールに動的にリンクします。サポートされている他の Unix に似たプラットフォームでは、`libmecab.so` は MySQL プラグインディレクトリにある `libpluginmecab.so` で静的にリンクされます。`mecab-ipadic` は MySQL バイナリに含まれており、`MYSQL_HOME/lib/mecab` にあります。

ネイティブパッケージ管理ユーティリティ (Fedora、Debian および Ubuntu 上) を使用して `mecab` および `mecab-ipadic` をインストールするか、ソースから `mecab` および `mecab-ipadic` をビルドできます。ネイティブパッケージ管理ユーティリティを使用した `mecab` および `mecab-ipadic` のインストールの詳細は、[Installing MeCab From a Binary Distribution \(Optional\)](#) を参照してください。ソースから `mecab` および `mecab-ipadic` をビルドする場合は、[Building MeCab From Source \(Optional\)](#) を参照してください。

Windows では、`libmecab.dll` は MySQL bin ディレクトリにあります。`mecab-ipadic` は `MYSQL_HOME/lib/mecab` にあります。

MeCab パーサープラグインをインストールおよび構成するには、次のステップを実行します:

1. MySQL 構成ファイルで、`mecab_rc_file` 構成オプションを、MeCab の構成ファイルである `mecabrc` 構成ファイルの場所に設定します。MySQL とともに配布される MeCab パッケージを使用している場合、`mecabrc` ファイルは `MYSQL_HOME/lib/mecab/etc/` にあります。

```
[mysqld]
loose-mecab-rc-file=MYSQL_HOME/lib/mecab/etc/mecabrc
```

`loose` 接頭辞は [option modifier](#) です。`mecab_rc_file` オプションは、MeCab パーサープラグインがインストールされるまで MySQL によって認識されませんが、MeCab パーサープラグインをインストールする前に設定する必要があります。`loose` 接頭辞を使用すると、認識されない変数が原因でエラーが発生することなく MySQL を再起動できます。

独自の MeCab インストールを使用する場合、またはソースから MeCab をビルドする場合、`mecabrc` 構成ファイルの場所が異なる可能性があります。

MySQL 構成ファイルとその場所の詳細は、[セクション4.2.2.2「オプションファイルの使用」](#)を参照してください。

2. また、MySQL 構成ファイルで、最小トークンサイズを 1 または 2 に設定します。これは、MeCab パーサーで使用することをお勧めします。InnoDB テーブルの場合、最小トークンサイズは `innodb_ft_min_token_size` 構

成オプションによって定義され、デフォルト値は 3 です。MyISAM テーブルの場合、最小トークンサイズは `ft_min_word_len` によって定義され、デフォルト値は 4 です。

```
[mysql]
innodb_ft_min_token_size=1
```

3. `mecabrc` 構成ファイルを変更して、使用するディクショナリを指定します。MySQL バイナリとともに配布される `mecab-ipadic` パッケージには、3 つのディクショナリ (`ipadic_euc-jp`、`ipadic_sjis` および `ipadic_utf-8`) が含まれます。MySQL とともにパッケージ化された `mecabrc` 構成ファイルには、次のようなエントリが含まれています:

```
dicdir = /path/to/mysql/lib/mecab/lib/mecab/dic/ipadic_euc-jp
```

たとえば、`ipadic_utf-8` ディクショナリを使用するには、次のようにエントリを変更します:

```
dicdir=MYSQL_HOME/lib/mecab/dic/ipadic_utf-8
```

独自の MeCab インストールを使用している場合、またはソースから MeCab をビルドしている場合、`mecabrc` ファイルのデフォルトの `dicdir` エントリはディクショナリとその場所とは異なる可能性があります。

注記

MeCab パーサープラグインのインストール後、`mecab_charset` ステータス変数を使用して、MeCab で使用される文字セットを表示できます。MySQL バイナリで提供される 3 つの MeCab ディクショナリでは、次の文字セットがサポートされます。

- `ipadic_euc-jp` ディクショナリは、`ujis` および `eucjpms` 文字セットをサポートしています。
- `ipadic_sjis` ディクショナリは、`sjis` および `cp932` 文字セットをサポートしています。
- `ipadic_utf-8` ディクショナリは、`utf8` および `utf8mb4` 文字セットをサポートしています。

`mecab_charset` では、最初にサポートされている文字セットのみがレポートされます。たとえば、`ipadic_utf-8` ディクショナリは、`utf8` と `utf8mb4` の両方をサポートしています。このディクショナリが使用されている場合、`mecab_charset` は常に `utf8` をレポートします。

4. MySQL を再開します。
5. MeCab パーサーのプラグインをインストールします:

MeCab パーサープラグインは、`INSTALL PLUGIN` 構文を使用してインストールされます。プラグイン名は `mecab` で、共有ライブラリ名は `libpluginmecab.so` です。プラグインのインストールの詳細は、[セクション 5.6.1「プラグインのインストールおよびアンインストール」](#) を参照してください。

```
INSTALL PLUGIN mecab SONAME 'libpluginmecab.so';
```

インストールされると、MeCab パーサープラグインは通常の MySQL の再起動のたびにロードされます。

6. `SHOW PLUGINS` ステートメントを使用して、MeCab パーサープラグインがロードされていることを確認します。

```
mysql> SHOW PLUGINS;
```

`mecab` プラグインがプラグインのリストに表示されます。

MeCab パーサーを使用する FULLTEXT インデックスの作成

`mecab` パーサーを使用する `FULLTEXT` インデックスを作成するには、`CREATE TABLE`、`ALTER TABLE` または `CREATE INDEX` とともに `WITH PARSER ngram` を指定します。

この例では、`mecab FULLTEXT` インデックスを使用したテーブルの作成、サンプルデータの挿入および `INFORMATION_SCHEMA.INNODB_FT_INDEX_CACHE` テーブルでのトークン化されたデータの表示を示します:

```
mysql> USE test;
```

```
mysql> CREATE TABLE articles (  
  id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,  
  title VARCHAR(200),  
  body TEXT,  
  FULLTEXT (title,body) WITH PARSER mecab  
 ) ENGINE=InnoDB CHARACTER SET utf8;  
  
mysql> SET NAMES utf8;  
  
mysql> INSERT INTO articles (title,body) VALUES  
 ('データベース管理','このチュートリアルでは、私はどのようにデータベースを管理する方法を紹介します'),  
 ('データベースアプリケーション開発','データベースアプリケーションを開発することを学ぶ');  
  
mysql> SET GLOBAL innodb_ft_aux_table="test/articles";  
  
mysql> SELECT * FROM INFORMATION_SCHEMA.INNOODB_FT_INDEX_CACHE ORDER BY doc_id, position;
```

既存のテーブルに **FULLTEXT** インデックスを追加するには、**ALTER TABLE** または **CREATE INDEX** を使用できません。例:

```
CREATE TABLE articles (  
  id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,  
  title VARCHAR(200),  
  body TEXT  
 ) ENGINE=InnoDB CHARACTER SET utf8;  
  
ALTER TABLE articles ADD FULLTEXT INDEX ft_index (title,body) WITH PARSER mecab;  
  
# Or:  
  
CREATE FULLTEXT INDEX ft_index ON articles (title,body) WITH PARSER mecab;
```

MeCab パーサーの領域処理

MeCab パーサーは、クエリー文字列のセパレータとして空白を使用します。たとえば、MeCab パーサーは、「データベース管理」を「データベース」および「管理」としてトークン化します。

MeCab パーサーのストップワード処理

デフォルトでは、MeCab パーサーは、英語のストップワードの短いリストを含むデフォルトのストップワードリストを使用します。日本語に適用可能なストップワードリストの場合は、独自のストップワードリストを作成する必要があります。ストップワードリストの作成の詳細は、[セクション12.10.4「全文ストップワード」](#) を参照してください。

MeCab パーサー用語検索

自然言語モードの検索では、検索語はトークンの和集合に変換されます。たとえば、「データベース管理」は「データベース管理」に変換されます。

```
SELECT COUNT(*) FROM articles WHERE MATCH(title,body) AGAINST('データベース管理' IN NATURAL LANGUAGE MODE);
```

ブールモード検索の場合、検索語は検索フレーズに変換されます。たとえば、「データベース管理」は「データベース管理」に変換されます。

```
SELECT COUNT(*) FROM articles WHERE MATCH(title,body) AGAINST('データベース管理' IN BOOLEAN MODE);
```

MeCab パーサーのワイルドカード検索

ワイルドカード検索語はトークン化されません。「データベース管理*」での検索は、接頭辞「データベース管理」に対して実行されます。

```
SELECT COUNT(*) FROM articles WHERE MATCH(title,body) AGAINST('データベース*' IN BOOLEAN MODE);
```

MeCab パーサーフレーズ検索

フレーズはトークン化されます。たとえば、「データベース管理」は「データベース管理」としてトークン化されます。

```
SELECT COUNT(*) FROM articles WHERE MATCH(title,body) AGAINST("データベース管理" IN BOOLEAN MODE);
```

バイナリ配布からの MeCab のインストール (オプション)

このセクションでは、ネイティブパッケージ管理ユーティリティを使用してバイナリ配布から `mecab` および `mecab-ipadic` をインストールする方法について説明します。たとえば、Fedora では、Yum を使用してインストールを実行できます:

```
yum mecab-devel
```

Debian または Ubuntu では、APT インストールを実行できます:

```
apt-get install mecab
apt-get install mecab-ipadic
```

ソースからの MeCab のインストール (オプション)

ソースから `mecab` および `mecab-ipadic` をビルドする場合、基本的なインストールステップは次のとおりです。詳細は、MeCab のドキュメントを参照してください。

1. `mecab` および `mecab-ipadic` 用の tar.gz パッケージを <http://taku910.github.io/mecab/#download> からダウンロードします。2016 年 2 月現在、使用可能な最新のパッケージは `mecab-0.996.tar.gz` および `mecab-ipadic-2.7.0-20070801.tar.gz` です。

2. `mecab` をインストールします:

```
tar zxvf mecab-0.996.tar
cd mecab-0.996
./configure
make
make check
su
make install
```

3. `mecab-ipadic` をインストールします:

```
tar zxvf mecab-ipadic-2.7.0-20070801.tar
cd mecab-ipadic-2.7.0-20070801
./configure
make
su
make install
```

4. `WITH_MECAB` CMake オプションを使用して MySQL をコンパイルします。 `mecab` および `mecab-ipadic` をデフォルトの場所にインストールした場合は、`WITH_MECAB` オプションを `system` に設定します。

```
-DWITH_MECAB=system
```

カスタムインストールディレクトリを定義した場合は、`WITH_MECAB` をカスタムディレクトリに設定します。例:

```
-DWITH_MECAB=/path/to/mecab
```

12.11 キャスト関数と演算子

表 12.15 「キャスト関数および演算子」

名前	説明
<code>BINARY</code>	文字列をバイナリ文字列にキャストします
<code>CAST()</code>	値を特定の型としてキャストします
<code>CONVERT()</code>	値を特定の型としてキャストします

キャスト関数および演算子を使用すると、あるデータ型から別のデータ型に値を変換できます。

`USING` 句を含む `CONVERT()` は、文字セット間でデータを変換します:

```
CONVERT(expr USING transcoding_name)
```

MySQL では、トランスコーディング名は対応する文字セット名と同じです。

例:

```
SELECT CONVERT('test' USING utf8mb4);
SELECT CONVERT(_latin1'Müller' USING utf8mb4);
INSERT INTO utf8mb4_table (utf8mb4_column)
  SELECT CONVERT(latin1_column USING utf8mb4) FROM latin1_table;
```

文字セット間で文字列を変換するには、`CONVERT(expr, type)` 構文 (`USING` なし) または `CAST(expr AS type)` も使用できます。これは同等です:

```
CONVERT(string, CHAR[(N)] CHARACTER SET charset_name)
CAST(string AS CHAR[(N)] CHARACTER SET charset_name)
```

例:

```
SELECT CONVERT('test', CHAR CHARACTER SET utf8mb4);
SELECT CAST('test' AS CHAR CHARACTER SET utf8mb4);
```

前述のように `CHARACTER SET charset_name` を指定した場合、結果の文字セットと照合順序は `charset_name` および `charset_name` のデフォルト照合順序になります。 `CHARACTER SET charset_name` を省略すると、結果の文字セットおよび照合順序は、デフォルトの接続文字セットおよび照合順序を決定する `character_set_connection` および `collation_connection` システム変数によって定義されます ([セクション10.4「接続文字セットおよび照合順序」](#) を参照)。

`COLLATE` 句は、`CONVERT()` または `CAST()` コール内では許可されませんが、関数の結果に適用できます。たとえば、次の場合は有効です。

```
SELECT CONVERT('test' USING utf8mb4) COLLATE utf8mb4_bin;
SELECT CONVERT('test', CHAR CHARACTER SET utf8mb4) COLLATE utf8mb4_bin;
SELECT CAST('test' AS CHAR CHARACTER SET utf8mb4) COLLATE utf8mb4_bin;
```

ただし、これらは不正です:

```
SELECT CONVERT('test' USING utf8mb4 COLLATE utf8mb4_bin);
SELECT CONVERT('test', CHAR CHARACTER SET utf8mb4 COLLATE utf8mb4_bin);
SELECT CAST('test' AS CHAR CHARACTER SET utf8mb4 COLLATE utf8mb4_bin);
```

通常、`BLOB` 値またはその他のバイナリ文字列は、大文字と小文字を区別しない方法で比較できません。バイナリ文字列では、大文字と小文字の概念と照合されない `binary` 文字セットが使用されるためです。大/小文字を区別しない比較を実行するには、まず `CONVERT()` または `CAST()` 関数を使用して、値を非バイナリ文字列に変換します。結果の文字列の比較では、その照合が使用されます。たとえば、変換結果の照合で大/小文字が区別されない場合、`LIKE` 操作では大/小文字は区別されません。これは、デフォルトの `utf8mb4` 照合 (`utf8mb4_0900_ai_ci`) では大/小文字が区別されないため、次の操作に当てはまります:

```
SELECT 'A' LIKE CONVERT(blob_col USING utf8mb4)
FROM tbl_name;
```

変換された文字列に特定の照合順序を指定するには、`CONVERT()` コールの後に `COLLATE` 句を使用します:

```
SELECT 'A' LIKE CONVERT(blob_col USING utf8mb4) COLLATE utf8mb4_unicode_ci
FROM tbl_name;
```

別の文字セットを使用するには、前述のステートメントで `utf8mb4` の名前を置き換えます (別の照合順序を使用する場合と同様です)。

`CONVERT()` および `CAST()` は、異なる文字セットで表される文字列を比較するために、より一般的に使用できます。たとえば、これらの文字列を比較すると、文字セットが異なるためエラーになります:

```
mysql> SET @s1 = _latin1 'abc', @s2 = _latin2 'abc';
mysql> SELECT @s1 = @s2;
ERROR 1267 (HY000): Illegal mix of collations (latin1_swedish_ci,IMPLICIT)
and (latin2_general_ci,IMPLICIT) for operation '='
```

いずれかの文字列を他の文字セットと互換性のある文字セットに変換すると、エラーなしで比較を実行できます:

```
mysql> SELECT @s1 = CONVERT(@s2 USING latin1);
+-----+
```

```
| @s1 = CONVERT(@s2 USING latin1) |
+-----+
|          1 |
+-----+
```

文字列リテラルの場合、文字セットを指定する別の方法は、文字セットイントロデューサを使用することです。前述の例の `_latin1` および `_latin2` は、イントロデューサのインスタンスです。文字列をある文字セットから別の文字セットに変換する `CAST()` や `CONVERT()` などの変換関数とは異なり、イントロデューサは文字列リテラルを特定の文字セットを持つものとして指定し、変換は必要ありません。詳細は、[セクション10.3.8「文字セットイントロデューサ」](#)を参照してください。

文字セット変換は、バイナリ文字列の大文字と小文字の変換の前にも役立ちます。 `LOWER()` および `UPPER()` は、大文字と小文字の概念が適用されないため、バイナリ文字列に直接適用すると無効になります。バイナリ文字列の大文字と小文字の変換を実行するには、まず文字列に格納されているデータに適した文字セットを使用して、非バイナリ文字列に変換します：

```
mysql> SET @str = BINARY 'New York';
mysql> SELECT LOWER(@str), LOWER(CONVERT(@str USING utf8mb4));
+-----+-----+
| LOWER(@str) | LOWER(CONVERT(@str USING utf8mb4)) |
+-----+-----+
| New York   | new york                           |
+-----+-----+
```

`BINARY`、`CAST()` または `CONVERT()` を使用してインデックス付けされたカラムを変換すると、MySQL でインデックスを効率的に使用できない場合があることに注意してください。

キャスト関数は、`CREATE TABLE ... SELECT` ステートメントで特定の型のカラムを作成する場合に役立ちます：

```
mysql> CREATE TABLE new_table SELECT CAST('2000-01-01' AS DATE) AS c1;
mysql> SHOW CREATE TABLE new_table\G
***** 1. row *****
      Table: new_table
      Create Table: CREATE TABLE `new_table` (
        `c1` date DEFAULT NULL
      ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```

キャスト関数は、`ENUM` カラムを字句順にソートする場合に役立ちます。通常、`ENUM` カラムのソートは、内部数値を使用して実行されます。値を `CHAR` にキャストすると、語彙順にソートされます。

```
SELECT enum_col FROM tbl_name ORDER BY CAST(enum_col AS CHAR);
```

`CONCAT('Date: ',CAST(NOW() AS DATE))` のように、より複雑な式の一部として使用する場合でも、`CAST()` の結果が変わります。

時間値の場合、`CAST()` を使用して様々な形式でデータを抽出する必要はほとんどありません。かわりに、`EXTRACT()`、`DATE_FORMAT()`、`TIME_FORMAT()` などの関数を使用してください。[セクション12.7「日付および時間関数」](#)を参照してください。

文字列を数値にキャストするには、通常、数値コンテキストで文字列値を使用するだけで十分です：

```
mysql> SELECT 1+'1';
-> 2
```

これは、16進数リテラルおよびビットリテラル(デフォルトではバイナリ文字列)にも当てはまります：

```
mysql> SELECT X'41', X'41'+0;
-> 'A', 65
mysql> SELECT b'1100001', b'1100001'+0;
-> 'a', 97
```

算術演算で使用される文字列は、式の評価時に浮動小数点数に変換されます。

文字列コンテキストで使用される数値は文字列に変換されます：

```
mysql> SELECT CONCAT('hello you ',2);
-> 'hello you 2'
```

数字から文字列への暗黙的な変換については、[セクション12.3「式評価での型変換」](#)を参照してください。

MySQL では、符号付きと符号なしの両方の 64 ビット値を使用した演算がサポートされています。オペランドのいずれかが符号なし整数である数値演算子 (+ や - など) の場合、結果はデフォルトで符号なしになります (セクション 12.6.1 「算術演算子」を参照)。これをオーバーライドするには、`SIGNED` または `UNSIGNED` キャスト演算子を使用して、それぞれ符号付きまたは符号なし 64 ビット整数に値をキャストします。

```
mysql> SELECT 1 - 2;
-> -1
mysql> SELECT CAST(1 - 2 AS UNSIGNED);
-> 18446744073709551615
mysql> SELECT CAST(CAST(1 - 2 AS UNSIGNED) AS SIGNED);
-> -1
```

オペランドのいずれかが浮動小数点値である場合は、結果が浮動小数点値になり、上記のルールによる影響を受けません。(このコンテキストでは、`DECIMAL` カラム値は浮動小数点値とみなされます。)

```
mysql> SELECT CAST(1 AS UNSIGNED) - 2.0;
-> -1.0
```

SQL モードは、変換操作の結果に影響します (セクション 5.1.11 「サーバー SQL モード」を参照)。例:

- 「「ゼロ」」の日付文字列を日付に変換する場合、`CONVERT()` および `CAST()` は `NULL` を返し、`NO_ZERO_DATE` SQL モードが有効なときに警告を生成します。
- 整数の減算では、`NO_UNSIGNED_SUBTRACTION` SQL モードが有効になっている場合は、オペランドのいずれかが符号なしでも、結果が符号付きになります。

次のリストでは、使用可能なキャスト関数および演算子について説明します:

- `BINARY expr`

`BINARY` 演算子は、式をバイナリ文字列 (`binary` 文字セットおよび `binary` 照合順序を持つ文字列) に変換します。`BINARY` の一般的な用途は、文字単位ではなく数値バイト値を使用して、文字列をバイト単位で強制的に比較することです。`BINARY` 演算子を使用すると、比較の末尾の空白も意味を持ちます。`binary` 文字セットの `binary` 照合順序と非バイナリ文字セットの `_bin` 照合順序の違いについては、セクション 10.8.5 「バイナリ照合順序と `_bin` 照合順序」を参照してください。

```
mysql> SELECT 'a' = 'A';
-> 1
mysql> SELECT BINARY 'a' = 'A';
-> 0
mysql> SELECT 'a' = 'a';
-> 1
mysql> SELECT BINARY 'a' = 'a';
-> 0
```

比較時に、`BINARY` によって演算全体が影響を受けます。これは、同じ結果を持ついずれかのオペランドの前に指定できます。

文字列式をバイナリ文字列に変換する場合、次の構造体は同等です:

```
CAST(expr AS BINARY)
CONVERT(expr USING BINARY)
BINARY expr
```

値が文字列リテラルの場合は、`_binary` 文字セットイントロデューサを使用して、変換せずにバイナリ文字列として指定できます:

```
mysql> SELECT 'a' = 'A';
-> 1
mysql> SELECT _binary 'a' = 'A';
-> 0
```

イントロデューサについては、セクション 10.3.8 「文字セットイントロデューサ」を参照してください。

式の `BINARY` 演算子は、文字カラム定義の `BINARY` 属性とは異なります。`BINARY` 属性で定義された文字列の場合、MySQL は、テーブルのデフォルトの文字セットと、その文字セットのバイナリ (`_bin`) 照合順序を割り当てます。すべての非バイナリ文字セットには、`_bin` 照合順序があります。たとえば、テーブルのデフォルトの文字セットが `utf8mb4` の場合、次の 2 つのカラム定義は同等です:

```
CHAR(10) BINARY
CHAR(10) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin
```

CHAR、**VARCHAR** または **TEXT** カラムの定義で **CHARACTER SET binary** を使用すると、カラムは対応するバイナリ文字列データ型として扱われます。たとえば、次のペアになった定義は同等です。

```
CHAR(10) CHARACTER SET binary
BINARY(10)

VARCHAR(10) CHARACTER SET binary
VARBINARY(10)

TEXT CHARACTER SET binary
BLOB
```

- **CAST(expr AS type [ARRAY])**

CAST(timestamp_value AT TIME ZONE timezone_specifier AS DATETIME[(precision)])

timezone_specifier: `[INTERVAL] '+00:00' | 'UTC'`

CAST() 関数は、任意の型の式を受け取り、**CONVERT()** と同様に、指定された型の結果値を生成します。詳細は、**CONVERT()** の説明を参照してください。

MySQL 8.0.17 以上では、**InnoDB** を使用して、**CREATE INDEX**、**CREATE TABLE** および **ALTER TABLE** ステートメントの一部として追加 **ARRAY** キーワードを使用して、**JSON** 配列に複数値インデックスを作成できます。これらのステートメントのいずれかで複数値インデックスを作成するために使用する場合を除き、**ARRAY** はサポートされていません。この場合は、複数値インデックスが必要です。インデックス付けするカラムは、**JSON** 型のカラムである必要があります。**ARRAY** では、**AS** キーワードの後に続く **type** は、**BINARY**、**JSON** および **YEAR** を除き、**CAST()** でサポートされている任意のタイプを指定できます。構文情報と例、およびその他の関連情報については、**複数値インデックス** を参照してください。

注記

CAST() とは異なり、**CONVERT()** では複数値インデックスの作成または **ARRAY** キーワードはサポートされていません。

MySQL 8.0.22 以降、**CAST()** では **AT TIMEZONE** 演算子を使用した UTC としての **TIMESTAMP** 値の取得がサポートされています。サポートされているタイムゾーンは UTC のみです。これは、`'+00:00'` または `'UTC'` のいずれかとして指定できます。この構文でサポートされている唯一の戻り型は **DATETIME** で、0 から 6 の範囲のオプションの精度指定子があります。

タイムゾーンオフセットを使用する **TIMESTAMP** 値もサポートされています。

```
mysql> SELECT @@system_time_zone;
+-----+
| @@system_time_zone |
+-----+
| EDT                |
+-----+
1 row in set (0.00 sec)

mysql> CREATE TABLE tz (c TIMESTAMP);
Query OK, 0 rows affected (0.41 sec)

mysql> INSERT INTO tz VALUES
  > ROW(CURRENT_TIMESTAMP),
  > ROW("2020-07-28 14:50:15+1:00");
Query OK, 1 row affected (0.08 sec)

mysql> TABLE tz;
+-----+
| c          |
+-----+
| 2020-07-28 09:22:41 |
| 2020-07-28 09:50:15 |
+-----+
2 rows in set (0.00 sec)
```

```
mysql> SELECT CAST(c AT TIME ZONE '+00:00' AS DATETIME) AS u FROM tz;
+-----+
| u          |
+-----+
| 2020-07-28 13:22:41 |
| 2020-07-28 13:50:15 |
+-----+
2 rows in set (0.00 sec)

mysql> SELECT CAST(c AT TIME ZONE 'UTC' AS DATETIME(2)) AS u FROM tz;
+-----+
| u          |
+-----+
| 2020-07-28 13:22:41.00 |
| 2020-07-28 13:50:15.00 |
+-----+
2 rows in set (0.00 sec)
```

この形式の `CAST()` で 'UTC' をタイムゾーン指定子として使用し、サーバーで「不明または不正なタイムゾーン: 'UTC'」などのエラーが発生した場合は、MySQL タイムゾーンテーブルのインストールが必要になることがあります ([タイムゾーンテーブルへの移入](#) を参照)。

`AT TIME ZONE` では `ARRAY` キーワードはサポートされておらず、`CONVERT()` 関数ではサポートされていません。

- `CONVERT(expr USING transcoding_name), CONVERT(expr,type)`

`CONVERT()` 関数は、任意の型の式を受け取り、指定された型の結果値を生成します。

`CONVERT(... USING ...)` は標準の SQL 構文です。 `USING` 形式以外の `CONVERT()` は ODBC の構文です。

`CONVERT(expr USING transcoding_name)` は、異なる文字セット間でデータを変換します。MySQL では、トランスコーディング名は対応する文字セット名と同じです。たとえば、次のステートメントは、デフォルト文字セットの文字列 'abc' を `utf8mb4` 文字セットの対応する文字列に変換します:

```
SELECT CONVERT('abc' USING utf8mb4);
```

`CONVERT(expr, type)` 構文 (`USING` を使用しない) では、結果タイプを指定する式および `type` 値を使用します。この操作は、同等の `CAST(expr AS type)` として表現することもできます。次の `type` 値を使用できます:

- `BINARY[(N)]`

`BINARY` データ型の文字列を生成します。これが比較に与える影響の詳細は、[セクション11.3.3「BINARY および VARBINARY 型」](#) を参照してください。オプションの長さ `N` が指定されている場合に、`BINARY(N)` を使用すると、キャストで `N` バイトの引数しか使用されなくなります。値が `N` バイトよりも短い場合は、`N` の長さになるまで `0x00` バイトでパディングされます。

- `CHAR[(N)] [charset_info]`

`CHAR` データ型の文字列を生成します。オプションの長さの `N` が指定されている場合、`CHAR(N)` はキャストで引数の `N` 文字以下を使用します。 `N` 文字より短い値にはパディングは行われません。

`charset_info` 句を指定しない場合、`CHAR` はデフォルトの文字セットを使用して文字列を生成します。文字セットを明示的に指定するには、次の `charset_info` 値を使用できます:

- `CHARACTER SET charset_name`: 指定された文字セットを持つ文字列を生成します。
- `ASCII`: `CHARACTER SET latin1` の短縮形。
- `UNICODE`: `CHARACTER SET ucs2` の短縮形。

いずれの場合も、文字列にはデフォルトの照合順序が設定されます。

- `DATE`

`DATE` 値を生成します。

- **DATETIME**

DATETIME 値を生成します。

- **DECIMAL[(M[,D])]**

DECIMAL 値を生成します。オプションの **M** および **D** 値が指定されている場合は、小数点以下の最大桁数 (精度) と桁数 (スケール) を指定します。

- **DOUBLE**

DOUBLE 結果を生成します。MySQL 8.0.17 で追加されました。

- **FLOAT[(p)]**

精度 **p** が指定されていない場合、**FLOAT** 型の結果が生成されます。 **p** が指定され、 $0 \leq p \leq 24$ の場合、結果のタイプは **FLOAT** になります。 $25 \leq p \leq 53$ の場合、結果のタイプは **DOUBLE** です。 $p < 0$ または $p > 53$ の場合、エラーが返されます。 MySQL 8.0.17 で追加されました。

- **JSON**

JSON 値を生成します。 **JSON** と他のタイプ間の値変換の規則の詳細は、[JSON 値の比較および順序付け](#) を参照してください。

- **NCHAR[(N)]**

CHAR と似ていますが、各国語文字セットの文字列を生成します。 [セクション10.3.7「各国語文字セット」](#) を参照してください。

CHAR とは異なり、**NCHAR** では後続の文字セット情報を指定できません。

- **REAL**

REAL タイプの結果を生成します。 **REAL_AS_FLOAT** SQL モードが有効な場合、これは実際には **FLOAT** です。それ以外の場合、結果のタイプは **DOUBLE** になります。

- **SIGNED [INTEGER]**

符号付き整数値を生成します。

- **spatial_type**

MySQL 8.0.24 では、**CAST()** および **CONVERT()** は、空間型の特定の組合せに対して、ある空間型から別の空間型へのジオメトリ値のキャストをサポートしています。次のリストに、許可されているタイプの組合せを示しま

す。ここで、「MySQL 拡張機能」は、[SQL/MM standard](#) で定義されているもの以外に、MySQL で実装されているキャストを指定します:

- [Point](#) から次の操作を行います:
 - [MultiPoint](#)
 - [GeometryCollection](#)
- [LineString](#) から次の操作を行います:
 - [Polygon](#) (MySQL 拡張機能)
 - [MultiPoint](#) (MySQL 拡張機能)
 - [MultiLineString](#)
 - [GeometryCollection](#)
- [Polygon](#) から次の操作を行います:
 - [LineString](#) (MySQL 拡張機能)
 - [MultiLineString](#) (MySQL 拡張機能)
 - [MultiPolygon](#)
 - [GeometryCollection](#)
- [MultiPoint](#) から次の操作を行います:
 - [Point](#)
 - [LineString](#) (MySQL 拡張機能)
 - [GeometryCollection](#)
- [MultiLineString](#) から次の操作を行います:
 - [LineString](#)
 - [Polygon](#) (MySQL 拡張機能)
 - [MultiPolygon](#) (MySQL 拡張機能)
 - [GeometryCollection](#)
- [MultiPolygon](#) から次の操作を行います:
 - [Polygon](#)
 - [MultiLineString](#) (MySQL 拡張機能)
 - [GeometryCollection](#)
- [GeometryCollection](#) から次の操作を行います:
 - [Point](#)
 - [LineString](#)
 - [Polygon](#)
 - [MultiPoint](#)

- [MultiLineString](#)
- [MultiPolygon](#)

空間キャストでは、[GeometryCollection](#) と [GeomCollection](#) は同じ結果タイプのシノニムです。

一部の条件はすべての空間型キャストに適用され、一部の条件はキャスト結果が特定の空間型を持つ場合にのみ適用されます。「整形形式のジオメトリ、」などの用語の詳細は、[セクション11.4.4「ジオメトリの整形形式と妥当性」](#)を参照してください。

- [空間キャストの一般的な条件](#)
- [ポイントへのキャストの条件](#)
- [LineString へのキャストの条件](#)
- [ポリゴンへのキャストの条件](#)
- [MultiPoint へのキャストの条件](#)
- [MultiLineString へのキャストの条件](#)
- [MultiPolygon へのキャストの条件](#)
- [GeometryCollection へのキャストの条件](#)

空間キャストの一般的な条件

次の条件は、結果の型に関係なく、すべての空間キャストに適用されます:

- キャストの結果は、キャストする式の SRS と同じ SRS になります。
- 空間型間のキャストでは、座標値または順序は変更されません。
- キャストする式が `NULL` の場合、関数の結果は `NULL` です。
- 空間型を指定する `RETURNING` 句とともに `JSON_VALUE()` 関数を使用して空間型にキャストすることはできません。
- 空間型の `ARRAY` へのキャストは許可されません。
- 空間型の組合せが許可されているが、キャストする式が構文的に整形形式のジオメトリでない場合は、`ER_GIS_INVALID_DATA` エラーが発生します。
- 空間型の組合せが許可されているが、キャストする式が未定義の空間参照システム (SRS) 内の構文的に整形されたジオメトリである場合、`ER_SRS_NOT_FOUND` エラーが発生します。
- キャストする式に地理 SRS があり、経度または緯度が範囲外の場合は、エラーが発生します:
 - 経度の値が $(-180, 180]$ の範囲にない場合は、`ER_GEOMETRY_PARAM_LONGITUDE_OUT_OF_RANGE` エラーが発生します。
 - 緯度の値が $[-90, 90]$ の範囲にない場合は、`ER_GEOMETRY_PARAM_LATITUDE_OUT_OF_RANGE` エラーが発生します。

表示される範囲は度数です。SRS で別の単位が使用されている場合、範囲ではその単位に対応する値が使用されます。浮動小数点演算のため、正確な範囲制限はわずかに偏差します。

ポイントへのキャストの条件

キャスト結果タイプが `Point` の場合は、次の条件が適用されます:

- キャストする式が `Point` 型の整形式ジオメトリである場合、関数の結果はその `Point` になります。
- キャストする式が単一の `Point` を含む `MultiPoint` 型の整形式ジオメトリである場合、関数の結果はその `Point` になります。式に複数の `Point` が含まれている場合、`ER_INVALID_CAST_TO_GEOMETRY` エラーが発生します。
- キャストする式が、単一の `Point` のみを含む `GeometryCollection` 型の整形式ジオメトリである場合、関数の結果はその `Point` になります。式が空の場合、複数の `Point` が含まれている場合、または他のジオメトリタイプが含まれている場合は、`ER_INVALID_CAST_TO_GEOMETRY` エラーが発生します。
- キャストする式が `Point`, `MultiPoint`, `GeometryCollection` 以外の型の整形式ジオメトリである場合、`ER_INVALID_CAST_TO_GEOMETRY` エラーが発生します。

LineString へのキャストの条件

キャスト結果タイプが `LineString` の場合は、次の条件が適用されます:

- キャストする式が `LineString` 型の整形式ジオメトリである場合、関数の結果はその `LineString` になります。
- キャストする式が、内部リングを持たない `Polygon` 型の整形式ジオメトリである場合、関数の結果は、外部リングの点を同じ順序で含む `LineString` になります。式に内部リングがある場合は、`ER_INVALID_CAST_TO_GEOMETRY` エラーが発生します。
- キャストする式が、少なくとも2つの点を含む `MultiPoint` 型の整形式ジオメトリである場合、関数の結果は、式に出現する順序で `MultiPoint` の点を含む `LineString` になります。式に `Point` が1つしか含まれていない場合、`ER_INVALID_CAST_TO_GEOMETRY` エラーが発生します。
- キャストする式が単一の `LineString` を含む `MultiLineString` 型の整形式ジオメトリである場合、関数の結果はその `LineString` になります。式に複数の `LineString` が含まれている場合、`ER_INVALID_CAST_TO_GEOMETRY` エラーが発生します。
- キャストする式が、単一の `LineString` のみを含む `GeometryCollection` 型の整形式ジオメトリである場合、関数の結果はその `LineString` になります。式が空の場合、複数の `LineString` が含まれている場合、または他のジオメトリタイプが含まれている場合は、`ER_INVALID_CAST_TO_GEOMETRY` エラーが発生します。
- キャストする式が `LineString`, `Polygon`, `MultiPoint`, `MultiLineString` または `GeometryCollection` 以外の型の整形式ジオメトリである場合、`ER_INVALID_CAST_TO_GEOMETRY` エラーが発生します。

ポリゴンへのキャストの条件

キャスト結果タイプが `Polygon` の場合は、次の条件が適用されます:

- キャストする式がリングである(つまり、開始点と終了点と同じである) `LineString` 型の整形式ジオメトリである場合、関数の結果は、同じ順序の `LineString` の点で構成される外部リングを持つ `Polygon` になります。式がリングでない場合は、`ER_INVALID_CAST_TO_GEOMETRY` エラーが発生します。リングの順序が正しくない場合(外側のリングは反時計回りである必要があります)、`ER_INVALID_CAST_POLYGON_RING_DIRECTION` エラーが発生します。
- キャストする式が `Polygon` 型の整形式ジオメトリである場合、関数の結果はその `Polygon` になります。
- キャストする式が、すべての要素がリングである `MultiLineString` 型の整形式ジオメトリである場合、関数の結果は、最初の `LineString` が外側のリングとして、追加の `LineString` 値が内側のリングとして含まれる `Polygon` になります。式のいずれかの要素がリングでない場合は、`ER_INVALID_CAST_TO_GEOMETRY` エラーが発生

します。正しい順序でないリングがある場合 (外側のリングは反時計回り、内側のリングは時計回りである必要があります)、`ER_INVALID_CAST_POLYGON_RING_DIRECTION` エラーが発生します。

- キャストする式が単一の `Polygon` を含む `MultiPolygon` 型の整形式ジオメトリである場合、関数の結果はその `Polygon` になります。式に複数の `Polygon` が含まれている場合、`ER_INVALID_CAST_TO_GEOMETRY` エラーが発生します。
- キャストする式が、単一の `Polygon` のみを含む `GeometryCollection` 型の整形式ジオメトリである場合、関数の結果はその `Polygon` になります。式が空の場合、複数の `Polygon` が含まれている場合、または他のジオメトリタイプが含まれている場合は、`ER_INVALID_CAST_TO_GEOMETRY` エラーが発生します。
- キャストする式が `LineString`, `Polygon`, `MultiLineString`, `MultiPolygon` または `GeometryCollection` 以外の型の整形式ジオメトリである場合、`ER_INVALID_CAST_TO_GEOMETRY` エラーが発生します。

MultiPoint へのキャストの条件

キャスト結果タイプが `MultiPoint` の場合は、次の条件が適用されます:

- キャストする式が `Point` 型の整形式のジオメトリである場合、関数の結果はその `Point` を唯一の要素として含む `MultiPoint` になります。
- キャストする式が `LineString` 型の整形式ジオメトリである場合、関数の結果は `LineString` の点を同じ順序で含む `MultiPoint` になります。
- キャストする式が `MultiPoint` 型の整形式ジオメトリである場合、関数の結果はその `MultiPoint` になります。
- キャストする式が、点のみを含む `GeometryCollection` 型の整形式ジオメトリである場合、関数の結果はこれらの点を含む `MultiPoint` になります。 `GeometryCollection` が空であるか、他のジオメトリタイプが含まれている場合、`ER_INVALID_CAST_TO_GEOMETRY` エラーが発生します。
- キャストする式が `Point`, `LineString`, `MultiPoint` または `GeometryCollection` 以外の型の整形式ジオメトリである場合、`ER_INVALID_CAST_TO_GEOMETRY` エラーが発生します。

MultiLineString へのキャストの条件

キャスト結果タイプが `MultiLineString` の場合は、次の条件が適用されます:

- キャストする式が `LineString` 型の整形式のジオメトリである場合、関数の結果はその `LineString` を唯一の要素として含む `MultiLineString` になります。
- キャストする式が `Polygon` 型の整形式ジオメトリである場合、関数の結果は、`Polygon` の外側のリングが最初の要素として含まれ、内側のリングが式に出現する順序で追加の要素として含まれる `MultiLineString` になります。
- キャストする式が `MultiLineString` 型の整形式ジオメトリである場合、関数の結果はその `MultiLineString` になります。
- キャストする式が、内部リングのないポリゴンのみを含む `MultiPolygon` 型の整形式ジオメトリである場合、関数の結果は、式に出現する順序でポリゴンのリングを含む `MultiLineString` になります。式に内部リングを持つポリゴンが含まれている場合は、`ER_WRONG_PARAMETERS_TO_STORED_FCT` エラーが発生します。
- キャストする式が `linestring` のみを含む `GeometryCollection` 型の整形式ジオメトリである場合、関数の結果はこれらの `linestring` を含む `MultiLineString` になります。式が空であるか、他のジオメトリタイプが含まれている場合、`ER_INVALID_CAST_TO_GEOMETRY` エラーが発生します。
- キャストする式が `LineString`, `Polygon`, `MultiLineString`, `MultiPolygon` または `GeometryCollection` 以外の型の整形式ジオメトリである場合、`ER_INVALID_CAST_TO_GEOMETRY` エラーが発生します。

MultiPolygon へのキャストの条件

キャスト結果タイプが `MultiPolygon` の場合は、次の条件が適用されます:

- キャストする式が Polygon 型の整形式のジオメトリである場合、関数の結果は、その唯一の要素として Polygon を含む MultiPolygon になります。
- キャストする式が、すべての要素がリングである MultiLineString 型の整形式ジオメトリである場合、関数の結果は、式の各要素の外側のリングのみを含む Polygon を含む MultiPolygon になります。いずれかの要素がリングでない場合は、ER_INVALID_CAST_TO_GEOMETRY エラーが発

生じます。正しい順序でないリングがある場合 (外側のリングは反時計回りである必要があります)、`ER_INVALID_CAST_POLYGON_RING_DIRECTION` エラーが発生します。

- キャストする式が `MultiPolygon` 型の整形式ジオメトリである場合、関数の結果はその `MultiPolygon` になりません。
- キャストする式がポリゴンのみを含む `GeometryCollection` 型の整形式ジオメトリである場合、関数の結果はそれらのポリゴンを含む `MultiPolygon` になります。式が空であるか、他のジオメトリタイプが含まれている場合、`ER_INVALID_CAST_TO_GEOMETRY` エラーが発生します。
- キャストする式が `Polygon`, `MultiLineString`, `MultiPolygon` または `GeometryCollection` 以外の型の整形式ジオメトリである場合、`ER_INVALID_CAST_TO_GEOMETRY` エラーが発生します。

GeometryCollection へのキャストの条件

キャスト結果タイプが `GeometryCollection` の場合は、次の条件が適用されます:

- `GeometryCollection` と `GeomCollection` は、同じ結果タイプのシノニムです。
- キャストする式が `Point` 型の整形式のジオメトリである場合、関数の結果はその `Point` を唯一の要素として含む `GeometryCollection` になります。
- キャストする式が `LineString` 型の整形式のジオメトリである場合、関数の結果はその `LineString` を唯一の要素として含む `GeometryCollection` になります。
- キャストする式が `Polygon` 型の整形式のジオメトリである場合、関数の結果はその `Polygon` を唯一の要素として含む `GeometryCollection` になります。
- キャストする式が `MultiPoint` 型の整形式ジオメトリである場合、関数の結果は、式に出現する順序で点を含む `GeometryCollection` になります。
- キャストする式が `MultiLineString` 型の整形式ジオメトリである場合、関数の結果は、式に出現する順序で `linestring` を含む `GeometryCollection` になります。
- キャストする式が `MultiPolygon` 型の整形式ジオメトリである場合、関数の結果は、式に出現する順序で `MultiPolygon` の要素を含む `GeometryCollection` になります。
- キャストする式が `GeometryCollection` 型の整形式ジオメトリである場合、関数の結果はその `GeometryCollection` になります。

- `TIME`

`TIME` 値を生成します。

- `UNSIGNED [INTEGER]`

符号なし整数値を生成します。

- `YEAR`

`YEAR` 値を生成します。MySQL 8.0.22 で追加されました。`YEAR` への変換は、次のルールによって制御されます:

- 1901-2155 の範囲の 4 桁の数値の場合、またはこの範囲の 4 桁の数値として解釈できる文字列の場合は、対応する `YEAR` 値を返します。
- 1 桁または 2 桁で構成される数値の場合、またはこのような数値としてインターベットできる文字列の場合、次のように `YEAR` 値を返します:
 - 数値が 1-69 の範囲内にある場合は、2000 を加算して合計を返します。
 - 数値が 70-99 の範囲内にある場合は、1900 を加算して合計を返します。

- 0 と評価される文字列の場合、2000 を返します。
- 数値 0 の場合は 0 を返します。
- `DATE`、`DATETIME` または `TIMESTAMP` 値の場合、値の `YEAR` 部分を返します。 `TIME` 値の場合、現在の年を返します。

`TIME` 引数の型を指定しない場合、次に示すように、予期したものとは異なる結果が得られる可能性があります:

```
mysql> SELECT CONVERT("11:35:00", YEAR), CONVERT(TIME "11:35:00", YEAR);
+-----+-----+
| CONVERT("11:35:00", YEAR) | CONVERT(TIME "11:35:00", YEAR) |
+-----+-----+
|          2011 |          2020 |
+-----+-----+
```

- 引数の型が `DECIMAL`、`DOUBLE`、`DECIMAL` または `REAL` の場合、次に示すように、値を最も近い整数に丸めながら、整数値のルールを使用して値を `YEAR` にキャストしようとします:

```
mysql> SELECT CONVERT(1944.35, YEAR), CONVERT(1944.50, YEAR);
+-----+-----+
| CONVERT(1944.35, YEAR) | CONVERT(1944.50, YEAR) |
+-----+-----+
|          1944 |          1945 |
+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT CONVERT(66.35, YEAR), CONVERT(66.50, YEAR);
+-----+-----+
| CONVERT(66.35, YEAR) | CONVERT(66.50, YEAR) |
+-----+-----+
|          2066 |          2067 |
+-----+-----+
1 row in set (0.00 sec)
```

- `GEOMETRY` 型の引数は、`YEAR` に変換できません。
- `YEAR` に正常に変換できない値の場合は、`NULL` を返します。

変換前に切り捨てる必要がある数値以外の文字を含む文字列値は、次に示すように警告を生成します:

```
mysql> SELECT CONVERT("1979aaa", YEAR);
+-----+
| CONVERT("1979aaa", YEAR) |
+-----+
|          1979 |
+-----+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Warning | 1292 | Truncated incorrect YEAR value: '1979aaa' |
+-----+-----+-----+
```

12.12 XML 関数

表 12.16 「XML 関数」

名前	説明
<code>ExtractValue()</code>	XPath 表記法を使用した XML 文字列からの値の抽出
<code>UpdateXML()</code>	置換後 XML フラグメントを返します

このセクションでは、MySQL での XML および関連する機能について説明します。

注記

--xml オプションを付けて呼び出すと、mysql および mysqldump クライアントで XML 書式の出力を MySQL から取得できます。セクション4.5.1「mysql — MySQL コマンドラインクライアント」およびセクション4.5.4「mysqldump — データベースバックアッププログラム」を参照してください。

基本的な XPath 1.0 (XML Path Language、バージョン 1.0) の機能を提供する 2 つの関数が使用可能です。XPath の構文および使用方法に関する基本情報の一部は、このセクションの後半で説明しますが、これらのトピックの詳細はこのマニュアルの範囲外であるため、「XML Path Language (XPath) 1.0 標準」で明確な情報を参照する必要があります。XPath がはじめてのユーザーや基本の復習を希望するユーザーに役立つリソースは、複数の言語で入手できる「Zvon.org XPath Tutorial」です。

注記

これらの関数はまだ開発中です。XML および XPath 機能のこれらの側面やその他の側面については、MySQL 8.0 以降で引き続き改善します。これらについて議論したり、質問したり、MySQL XML ユーザーフォーラムでほかのユーザーからの支援を得たりすることもできます。

これらの関数で使用される XPath の式では、ユーザー変数およびローカルストアプログラム変数がサポートされています。ユーザー変数は簡単にチェックされます。ストアプログラムへのローカル変数は厳密にチェックされます (Bug#26518 も参照してください)。

- ユーザー変数 (簡単なチェック). 構文 `$_variable_name` を使用する変数 (つまり、ユーザー変数) はチェックされません。変数の型が間違っている場合や、変数に値が事前に割り当てられていない場合でも、サーバーから警告やエラーが発行されません。これは、`$_myvariable` が意図された場所で (たとえば `$_myvariable` が使用された場合、警告が表示されないため、ユーザーが入力ミスを完全に担当することも意味します)。

例:

```
mysql> SET @xml = '<a><b>X</b><b>Y</b></a>';
Query OK, 0 rows affected (0.00 sec)

mysql> SET @i = 1, @j = 2;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @i, ExtractValue(@xml, '//b[$@i]');
+----+-----+
| @i | ExtractValue(@xml, '//b[$@i]') |
+----+-----+
| 1 | X |
+----+-----+
1 row in set (0.00 sec)

mysql> SELECT @j, ExtractValue(@xml, '//b[$@j]');
+----+-----+
| @j | ExtractValue(@xml, '//b[$@j]') |
+----+-----+
| 2 | Y |
+----+-----+
1 row in set (0.00 sec)

mysql> SELECT @k, ExtractValue(@xml, '//b[$@k]');
+----+-----+
| @k | ExtractValue(@xml, '//b[$@k]') |
+----+-----+
| NULL | |
+----+-----+
1 row in set (0.00 sec)
```

- ストアドプログラム内の変数 (厳密なチェック). これらの関数をストアプログラム内部で呼び出すときに、構文 `$_variable_name` を使用する変数を宣言し、これらの関数で使用できます。このような変数は、定義されているストアプログラムへのローカル変数であり、型および値について厳密にチェックされます。

例:

```
mysql> DELIMITER |
```



```
mysql> CREATE PROCEDURE myproc ()
-> BEGIN
-> DECLARE i INT DEFAULT 1;
-> DECLARE xml VARCHAR(25) DEFAULT '<a>X</a><a>Y</a><a>Z</a>';
->
-> WHILE i < 4 DO
-> SELECT xml, i, ExtractValue(xml, '//a[$i]');
-> SET i = i+1;
-> END WHILE;
-> END |
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;

mysql> CALL myproc();
+-----+-----+-----+
| xml          | i | ExtractValue(xml, '//a[$i]') |
+-----+-----+-----+
| <a>X</a><a>Y</a><a>Z</a> | 1 | X                               |
+-----+-----+-----+
1 row in set (0.00 sec)

+-----+-----+-----+
| xml          | i | ExtractValue(xml, '//a[$i]') |
+-----+-----+-----+
| <a>X</a><a>Y</a><a>Z</a> | 2 | Y                               |
+-----+-----+-----+
1 row in set (0.01 sec)

+-----+-----+-----+
| xml          | i | ExtractValue(xml, '//a[$i]') |
+-----+-----+-----+
| <a>X</a><a>Y</a><a>Z</a> | 3 | Z                               |
+-----+-----+-----+
1 row in set (0.01 sec)
```

パラメータ。 ストアドルーチン内部の XPath 式で使用され、パラメータとして渡される変数も、厳密なチェックの対象です。

ユーザー変数やストアドプログラムへのローカル変数を含む式は、その他の点 (表記法は除く) では、XPath 1.0 仕様で規定されている変数を含む XPath 式のルールに準拠する必要があります。

注記

XPath 式の格納に使用されるユーザー変数は、空の文字列として扱われます。このため、ユーザー変数として XPath 式を格納することはできません。(Bug #32911)

• ExtractValue(xml_frag, xpath_expr)

`ExtractValue()` は、XML マークアップ `xml_frag` のフラグメントと XPath 式 `xpath_expr` (ロケータとも呼ばれる) の 2 つの文字列引数を取ります。これは、XPath 式に一致する要素の子である最初のテキストノードのテキスト (CDATA) を返します。

この関数を使用することは、`/text()` を追加したあとに `xpath_expr` を使用して一致を実行することと同等です。言い換えると、`ExtractValue('<a>Sakila', '/a/b')` と `ExtractValue('<a>Sakila', '/a/b/text()')` では同じ結果が生成されます。

複数の一致が見つかった場合は、一致する各要素の 1 番目の子テキストノードの内容が空白文字で区切られた単一文字列として (一致した順序で) 返されます。

式に一致するテキストノード (暗黙的な `/text()` を含む) が見つからない場合は、どのような理由でも、`xpath_expr` が有効で、`xml_frag` が適切にネストされ、閉じられた要素で構成されていれば、空の文字列が返されます。空の要素で一致することと、まったく一致しないこととは区別されません。これは意図的なものです。

`xml_frag` で一致する要素が見つからなかったのか、またはこのような要素は見つかったが、子テキストノードが含まれていなかったのかを判断する必要がある場合は、XPath `count()` 関数を使用する式の結果をテストしてください。たとえば、次に示すように、これらのステートメントの両方で空の文字列が返されます。

```
mysql> SELECT ExtractValue('<a><b/></a>', '/a/b');
+-----+
| ExtractValue('<a><b/></a>', '/a/b') |
+-----+
|                                     |
+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT ExtractValue('<a><c/></a>', '/a/b');
+-----+
| ExtractValue('<a><c/></a>', '/a/b') |
+-----+
|                                     |
+-----+
1 row in set (0.00 sec)
```

ただし、次のコマンドを使用すれば、実際に一致する要素があったかどうかを判断できます。

```
mysql> SELECT ExtractValue('<a><b/></a>', 'count(/a/b)');
+-----+
| ExtractValue('<a><b/></a>', 'count(/a/b)') |
+-----+
| 1                                     |
+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT ExtractValue('<a><c/></a>', 'count(/a/b)');
+-----+
| ExtractValue('<a><c/></a>', 'count(/a/b)') |
+-----+
| 0                                     |
+-----+
1 row in set (0.01 sec)
```

重要

`ExtractValue()` では `CDATA` のみが返され、一致するタグ内に含まれるタグや、それらの内容は返されません (次の例で、`val1` として返された結果を参照してください)。

```
mysql> SELECT
-> ExtractValue('<a>ccc<b>ddd</b></a>', '/a') AS val1,
-> ExtractValue('<a>ccc<b>ddd</b></a>', '/a/b') AS val2,
-> ExtractValue('<a>ccc<b>ddd</b></a>', '/b') AS val3,
-> ExtractValue('<a>ccc<b>ddd</b></a>', '/b') AS val4,
-> ExtractValue('<a>ccc<b>ddd</b><b>eee</b></a>', '/b') AS val5;
+-----+-----+-----+-----+
| val1 | val2 | val3 | val4 | val5 |
+-----+-----+-----+-----+
| ccc  | ddd  | ddd  |      | ddd eee |
+-----+-----+-----+-----+
```

この関数では、`contains()` との比較を実行し、その他の文字列関数 (`CONCAT()` など) と同じ照合順序アグリゲーションを実行し、それらの引数の照合順序強制性を考慮に入れる際に、現在の SQL 照合順序が使用されます。この動作を制御するルールの説明については、[セクション 10.8.4 「式での照合の強制性」](#) を参照してください。

(以前は、大文字と小文字が区別されるバイナリが常に使用されていました。)

次の例に示すように、`xml_frag` に、適切にネストされていない要素や閉じられていない要素が含まれ、警告が生成された場合は、`NULL` が返されます。

```
mysql> SELECT ExtractValue('<a>c</a><b', '//a');
+-----+
| ExtractValue('<a>c</a><b', '//a') |
+-----+
| NULL                             |
+-----+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
```

```

***** 1. row *****
Level: Warning
Code: 1525
Message: Incorrect XML value: 'parse error at line 1 pos 11:
        END-OF-INPUT unexpected ('>' wanted)'
1 row in set (0.00 sec)

mysql> SELECT ExtractValue('<a>c</a><b/>', '//a');
+-----+
| ExtractValue('<a>c</a><b/>', '//a') |
+-----+
| c          |
+-----+
1 row in set (0.00 sec)

```

- `UpdateXML(xml_target, xpath_expr, new_xml)`

この関数は、XML マークアップ `xml_target` の特定のフラグメントの一部を新しい XML フラグメント `new_xml` に置き換えてから、変更された XML を返します。置換された `xml_target` の一部は、ユーザーが指定した XPath 式 `xpath_expr` に一致します。

`xpath_expr` に一致する式が見つからない場合、または複数の一致が見つかった場合、この関数は元の `xml_target` XML フラグメントを返します。3 つの引数はすべて文字列にする必要があります。

```

mysql> SELECT
-> UpdateXML('<a><b>ccc</b><d></d></a>', '/a', '<e>fff</e>') AS val1,
-> UpdateXML('<a><b>ccc</b><d></d></a>', '/b', '<e>fff</e>') AS val2,
-> UpdateXML('<a><b>ccc</b><d></d></a>', '/b', '<e>fff</e>') AS val3,
-> UpdateXML('<a><b>ccc</b><d></d></a>', '/a/d', '<e>fff</e>') AS val4,
-> UpdateXML('<a><d></d><b>ccc</b><d></d></a>', '/a/d', '<e>fff</e>') AS val5
-> \G
***** 1. row *****
val1: <e>fff</e>
val2: <a><b>ccc</b><d></d></a>
val3: <a><e>fff</e><d></d></a>
val4: <a><b>ccc</b><e>fff</e></a>
val5: <a><d></d><b>ccc</b><d></d></a>

```

注記

XPath の構文および使用方法の詳細は、このマニュアルの範囲外です。最終的な情報については、「[XML Path Language \(XPath\) 1.0 仕様](#)」を参照してください。XPath がはじめてのユーザーや基本の復習を希望するユーザーに役立つリソースは、複数の言語で入手できる「[Zvon.org XPath Tutorial](#)」です。

一部の基本的な XPath 式の説明および例は、次のとおりです。

- `/tag`

`<tag/>` がルート要素の場合にかぎり、`<tag/>` に一致します。

例: `/a` はいちばん外側の (ルート) タグに一致するため、`<a>` には一致がありません。この例では、別の要素の子であるため、`<a/>` の内側の `a` 要素には一致しません。

- `/tag1/tag2`

`<tag1/>` の子であり、`<tag1/>` がルート要素である場合にかぎり、`<tag2/>` に一致します。

例: `/a/b` はルート要素 `a` の子であるため、XML フラグメント `<a>` の `b` 要素に一致します。この場合、`b` はルート要素 (その他の要素の子) であるため、`<a/>` には一致がありません。XPath 式でも `<a><c></c>` に一致がありません。ここで、`b` は `a` の子孫ですが、実際には `a` の子ではありません。

この構成は、3 つ以上の要素に拡張できます。たとえば、XPath 式 `/a/b/c` は、フラグメント `<a><c/>` 内の `c` 要素に一致します。

- `//tag`

`<tag>` の任意のインスタンスに一致します。

例: //a は、<a><c>、<c><a>、<c><a/></c> のいずれかの a 要素に一致します。

// は / と組み合わせることができます。たとえば、//a/b は、フラグメント <a> または <c><a></c> のいずれかの b 要素と一致します。

注記

//tag は /descendant-or-self:*tag と同等です。よく見られる誤りは、これと /descendant-or-self:tag とを混同することです。次に示すように、実際には後者の式ではまったく異なる結果が生成される可能性があります。

```
mysql> SET @xml = '<a><b><c>w</c><b>x</b><d>y</d>z</b></a>';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT @xml;
+-----+
| @xml |
+-----+
| <a><b><c>w</c><b>x</b><d>y</d>z</b></a> |
+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT ExtractValue(@xml, '//b[1]');
+-----+
| ExtractValue(@xml, '//b[1]') |
+-----+
| x z |
+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT ExtractValue(@xml, '//b[2]');
+-----+
| ExtractValue(@xml, '//b[2]') |
+-----+
| |
+-----+
1 row in set (0.01 sec)
```

```
mysql> SELECT ExtractValue(@xml, '/descendant-or-self:*b[1]');
+-----+
| ExtractValue(@xml, '/descendant-or-self:*b[1]') |
+-----+
| x z |
+-----+
1 row in set (0.06 sec)
```

```
mysql> SELECT ExtractValue(@xml, '/descendant-or-self:*b[2]');
+-----+
| ExtractValue(@xml, '/descendant-or-self:*b[2]') |
+-----+
| |
+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT ExtractValue(@xml, '/descendant-or-self:b[1]');
+-----+
| ExtractValue(@xml, '/descendant-or-self:b[1]') |
+-----+
| z |
+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT ExtractValue(@xml, '/descendant-or-self:b[2]');
+-----+
| ExtractValue(@xml, '/descendant-or-self:b[2]') |
+-----+
| x |
+-----+
1 row in set (0.00 sec)
```

- * 演算子は、任意の要素に一致する「ワイルドカード」として機能します。たとえば、式 `*/b` は、XML フラグメント `<a>` または `<c></c>` 内の `b` 要素に一致します。ただし、`b` はその他の要素の子である必要があるため、この式ではフラグメント `<a/>` 内の一致は生成されません。ワイルドカードは任意の位置で使用できます。式 `*/b/*` は、それ自体がルート要素ではない `b` 要素の子と一致します。
- | (UNION) 演算子を使用すれば、複数のロケータのいずれかに一致できます。たとえば、式 `//b//c` は、XML ターゲット内のすべての `b` および `c` 要素に一致します。
- その属性の 1 つ以上の値に基づいて、要素に一致することもできます。これは、構文 `tag[@attribute="value"]` を使用して実行されます。たとえば、式 `//b[@id="idB"]` は、フラグメント `<a><b id="idA"/><b id="idB"/>` 内の 2 番目の `b` 要素に一致します。 `attribute="value"` を含む任意の要素に対して一致を行うには、XPath 式 `//*[@attribute="value"]` を使用します。

複数の属性値をフィルタ処理するには、単に複数の属性比較句を連続して使用するだけです。たとえば、式 `//b[@c="x"][@d="y"]` は、指定した XML フラグメントの任意の場所で発生した要素 `<b c="x" d="y"/>` に一致します。

同じ属性が複数の値のいずれかに一致する要素を見つけるには、| 演算子で結合された複数のロケータを使用します。たとえば、`c` 属性の値が 23 または 17 であるすべての `b` 要素に一致するには、式 `//b[@c="23"]//b[@c="17"]` を使用します。この目的のために、`//b[@c="23" or @c="17"]` のように論理 `or` 演算子を使用することもできます。

注記

`or` と | の相違点として、`or` は条件を結合するのに対し、| は結果セットを結合します。

XPath の制限。 現在、これらの関数でサポートされている XPath 構文は、次の制限の対象となっています。

- ノードセット間の比較 (`/a/b[@c=@d]` など) はサポートされていません。
- 標準の XPath 比較演算子はすべてサポートされています。(Bug #22823)
- 相対ロケータ式は、ルートノードのコンテキストで解決されます。たとえば、次のようなクエリーと結果を考えてみます。

```
mysql> SELECT ExtractValue(
-> ' <a><b c="1">X</b><b c="2">Y</b></a>',
-> 'a/b'
->) AS result;
+-----+
| result |
+-----+
| X Y   |
+-----+
1 row in set (0.03 sec)
```

この場合は、ロケータ `a/b` が `/a/b` に解決されています。

相対ロケータは、述語内でもサポートされています。次の例では、`d[./@c="1"]` が `/a/b[@c="1"]/d` として解決されています。

```
mysql> SELECT ExtractValue(
-> ' <a>
-> <b c="1"><d>X</d></b>
-> <b c="2"><d>X</d></b>
-> </a>',
-> 'a/b/d[./@c="1"]'
->) AS result;
+-----+
| result |
+-----+
| X     |
+-----+
1 row in set (0.00 sec)
```

- スカラー値 (変数参照、リテラル、数字、およびスカラー関数の呼び出しを含む) として評価する式が前に付けられたロケータは許可されず、使用するとエラーが発生します。
- :: 演算子を次のようなノード型と組み合わせることは、サポートされていません。

- `axis::comment()`
- `axis::text()`
- `axis::processing-instructions()`
- `axis::node()`

ただし、次の例に示すように、名前のテスト (`axis::name` や `axis::*` など) はサポートされています。

```
mysql> SELECT ExtractValue('<a><b>x</b><c>y</c></a>', '/a/child::b');
+-----+
| ExtractValue('<a><b>x</b><c>y</c></a>', '/a/child::b') |
+-----+
| x                                     |
+-----+
1 row in set (0.02 sec)

mysql> SELECT ExtractValue('<a><b>x</b><c>y</c></a>', '/a/child::*');
+-----+
| ExtractValue('<a><b>x</b><c>y</c></a>', '/a/child::*') |
+-----+
| x y                                     |
+-----+
1 row in set (0.01 sec)
```

- パスガルート要素を「上方向」に導いている場合は、「上下」の移動がサポートされていません。つまり、現在の要素の1つ以上の祖先ガルート要素の祖先でもある場合は、指定された要素の祖先の子孫で一致する式を使用できません (Bug #16321 を参照してください)。
- 次の XPath 関数はサポートされていないか、または説明したような既知の問題があります。
 - `id()`
 - `lang()`
 - `local-name()`
 - `name()`
 - `namespace-uri()`
 - `normalize-space()`
 - `starts-with()`
 - `string()`
 - `substring-after()`
 - `substring-before()`
 - `translate()`
- 次の軸はサポートされていません。
 - `following-sibling`
 - `following`
 - `preceding-sibling`
 - `preceding`

`ExtractValue()` および `UpdateXML()` への引数として渡される XPath 式の要素セレクタ内には、コロン文字 (:) が含まれている可能性があります。これにより、XML 名前空間の表記法を使用しているマークアップとの使用が有効になります。例:

```
mysql> SET @xml = '<a>111<b:c>222<d>333</d><e:f>444</e:f></b:c></a>';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT ExtractValue(@xml, '//e:f');
```

```
+-----+
| ExtractValue(@xml, '//e:f') |
+-----+
| 444 |
+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT UpdateXML(@xml, '//b:c', '<g:h>555</g:h>');
```

```
+-----+
| UpdateXML(@xml, '//b:c', '<g:h>555</g:h>') |
+-----+
| <a>111<g:h>555</g:h></a> |
+-----+
1 row in set (0.00 sec)
```

これは、いくつかの点で [Apache Xalan](#) およびその他の一部のパーサーで許可されているものと似ていますが、名前空間の制限や `namespace-uri()` および `local-name()` 関数の使用を必要とするよりも大幅に単純です。

エラー処理. `ExtractValue()` と `UpdateXML()` のどちらの場合でも、使用される XPath ロケータが有効であり、検索対象の XML が適切にネストされ、閉じられた要素で構成されている必要があります。ロケータが無効な場合は、次のようなエラーが生成されます。

```
mysql> SELECT ExtractValue('<a>c</a><b/>', '&a');
ERROR 1105 (HY000): XPATH syntax error: '&a'
```

`xml_frag` が適切にネストされ、閉じられている要素で構成されていない場合は、次の例に示すように、`NULL` が返され、警告が生成されます。

```
mysql> SELECT ExtractValue('<a>c</a><b', '//a');
```

```
+-----+
| ExtractValue('<a>c</a><b', '//a') |
+-----+
| NULL |
+-----+
1 row in set, 1 warning (0.00 sec)
```

```
mysql> SHOW WARNINGS\SIG
```

```
***** 1. row *****
Level: Warning
Code: 1525
Message: Incorrect XML value: 'parse error at line 1 pos 11:
END-OF-INPUT unexpected ('>' wanted)'
1 row in set (0.00 sec)
```

```
mysql> SELECT ExtractValue('<a>c</a><b/>', '//a');
```

```
+-----+
| ExtractValue('<a>c</a><b/>', '//a') |
+-----+
| c |
+-----+
1 row in set (0.00 sec)
```

重要

`UpdateXML()` への第 3 引数として使用される置換用の XML は、適切にネストされ、閉じられている要素のみで構成されているかどうかを判断するためにチェックされません。

XPath インジェクション. コードインジェクションは、権限やデータへの不正アクセス権を取得するために、悪意のあるコードがシステムに導入された場合に発生します。これは、ユーザーが入力したデータの型や内容について発行者が行なった想定外の悪用に基づいています。これに関しては、XPath も例外ではありません。

この問題が発生する可能性のある一般的なシナリオは、次のような XPath 式を使用して、ログイン名とパスワードの組み合わせを XML ファイル内で見つかったものと一致させることで承認を処理するアプリケーションのケースです。

```
//user[login/text()='neapolitan' and password/text()='1c3cr34m']/attribute::id
```

この XPath 式は、次のような SQL ステートメントと同等です。

```
SELECT id FROM users WHERE login='neapolitan' AND password='1c3cr34m';
```

XPath を使用している PHP アプリケーションでは、次のようにログインプロセスが処理される可能性があります。

```
<?php
$file = "users.xml";

$login = $_POST["login"];
$password = $_POST["password"];

$xmlpath = "//user[login/text()=$login and password/text()=$password]/attribute::id";

if( file_exists($file) )
{
    $xml = simplexml_load_file($file);

    if($result = $xml->xpath($xmlpath))
        echo "You are now logged in as user $result[0].";
    else
        echo "Invalid login name or password.";
}
else
    exit("Failed to open $file.");
?>
```

入力時にはチェックが実行されません。これは、悪意のあるユーザーがログイン名とパスワードの両方に ' or 1=1 と入力することで、テストを「回避」できることを意味します。その結果、`$xpath` が次のように評価されます。

```
//user[login/text()=' or 1=1 and password/text()=' or 1=1']/attribute::id
```

角括弧内の式は常に `true` と評価されるため、事実上、XML ドキュメント内のすべての `user` 要素の `id` 属性に一致する次の式と同じです。

```
//user/attribute::id
```

この攻撃を回避する方法の 1 つは、`$xpath` の定義内に挿入される変数名を単に引用符で囲むだけです。これにより、Web フォームから渡された値が強制的に文字列に変換されます。

```
$xpath = "//user[login/text()='$_login' and password/text()='$_password']/attribute::id";
```

これは、SQL インジェクション攻撃を回避する際に推奨されることの多い方法と同じです。一般に、XPath インジェクション攻撃を回避するために従うべき方法は、SQL インジェクションを回避するための方法と同じです。

- アプリケーションでは、テストされていないユーザーデータは許可されません。
- ユーザーが送信したすべてのデータの型をチェックします。不正な型のデータは拒否または変換します。
- 数値データに範囲外の値が含まれていないかをテストします。範囲外の値は切り捨てるか、丸めるか、拒否します。文字列に不正な文字が含まれていないかをテストし、不正な文字が含まれる入力は削除するか拒否します。
- 明示的なエラーメッセージは、システムを危険にさらすために使用できる手がかりを未承認ユーザーに与える可能性があるため、出力しないでください。その代わりに、ファイルやデータベーステーブルにログを記録してください。

SQL インジェクション攻撃を使用すればデータベーススキーマに関する情報を取得できるように、XPath インジェクションを使用すれば、Amit Klein 氏の論文『[Blind XPath Injection](#)』(PDF ファイル、46K バイト) で説明されているように、XML ファイルをスキャンして構造を明らかにできます。

クライアントに返送される出力をチェックすることも重要です。MySQL の `ExtractValue()` 関数を使用すると何が発生する可能性があるのかを検討します。

```
mysql> SELECT ExtractValue(
->   LOAD_FILE('users.xml'),
->   '//user[login/text()=' or 1=1 and password/text()=' or 1=1']/attribute::id'
```

```
-> ) AS id;
+-----+
| id          |
+-----+
| 00327 13579 02403 42354 28570 |
+-----+
1 row in set (0.01 sec)
```

`ExtractValue()` は複数の一致を空白で区切られた単一の文字列として返すため、このインジェクション攻撃によって、`users.xml` 内に含まれるすべての有効な ID が単一の出力行としてユーザーに提供されます。追加の保護手段として、ユーザーに返される前に出力のテストも行うべきです。次に、単純な例を示します。

```
mysql> SELECT @id = ExtractValue(
->  LOAD_FILE('users.xml'),
->  '/user[login/text()=' or 1=1 and password/text()=' or 1=1]/attribute::id'
-> );
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT IF(
->  INSTR(@id, ' ') = 0,
->  @id,
->  'Unable to retrieve user ID')
-> AS singleID;
+-----+
| singleID |
+-----+
| Unable to retrieve user ID |
+-----+
1 row in set (0.00 sec)
```

一般に、ユーザーにデータをセキュアに返すためのガイドラインは、ユーザー入力を受け入れるためのガイドラインと同じです。それらは、次のように要約できます。

- 常に、出力データの型および許可される値をテストします。
- 未承認ユーザーがエラーメッセージを表示することを許可しないでください。アプリケーションに関する情報が提供される可能性があり、それを悪用されるおそれがあります。

12.13 ビット関数と演算子

表 12.17 「ビット関数と演算子」

名前	説明
<code>&</code>	ビット単位の AND
<code>>></code>	右シフト
<code><<</code>	左シフト
<code>^</code>	ビット単位の XOR
<code>BIT_COUNT()</code>	設定されているビット数を返します
<code> </code>	ビット単位の OR
<code>~</code>	ビット単位の反転

ビット関数および演算子は、`BIT_COUNT()`、`BIT_AND()`、`BIT_OR()`、`BIT_XOR()`、`&`、`|`、`^`、`~`、`<<` および `>>` で構成されます。(`BIT_AND()`、`BIT_OR()` および `BIT_XOR()` の集計関数については、[セクション12.20.1「集計関数の説明」](#) を参照してください。) MySQL 8.0 より前では、ビット関数および演算子には `BIGINT` (64-bit 整数) 引数が必要であり、`BIGINT` 値が返されるため、最大範囲は 64 ビットでした。操作の実行前に `BIGINT` 以外の引数が `BIGINT` に変換され、切捨てが発生する可能性があります。

MySQL 8.0 では、ビット関数および演算子はバイナリ文字列型の引数 (`BINARY`、`VARBINARY` および `BLOB` 型) を許可し、同様の型の値を返します。これにより、引数を受け取り、64 ビットを超える戻り値を生成できます。バイナリ以外の文字列引数は `BIGINT` に変換され、以前と同様に処理されます。

この動作の変更の影響は、バイナリ文字列引数に対するビット操作によって、MySQL 8.0 と 5.7 では異なる結果が生成される可能性があることです。MySQL 5.7 と 8.0 間の潜在的な非互換性のために MySQL 5.7 で準備する方法の詳細は、[MySQL 5.7 Reference Manual の Bit Functions and Operators](#) を参照してください。

- MySQL 8.0 より前のビット操作
- MySQL 8.0 でのビット操作
- バイナリ文字列のビット操作の例
- ビット単位の AND、OR および XOR 操作
- ビット単位の補完およびシフト操作
- BIT_COUNT() の操作
- BIT_AND()、BIT_OR() および BIT_XOR() の操作
- 16 進数リテラル、ビットリテラルおよび NULL リテラルの特殊な処理
- MySQL 5.7 とのビット操作の非互換性

MySQL 8.0 より前のビット操作

MySQL 8.0 より前のビット操作では、符号なし 64 ビット整数引数と結果値 (つまり、符号なし **BIGINT** 値) のみが処理されます。他の型の引数の **BIGINT** への変換は、必要に応じて行われます。例:

- 次のステートメントは、符号なし 64 ビット整数として扱われる数値リテラルを操作します:

```
mysql> SELECT 127 | 128, 128 << 2, BIT_COUNT(15);
+-----+-----+-----+
| 127 | 128 | 128 << 2 | BIT_COUNT(15) |
+-----+-----+-----+
| 255 | 512 | 4 |
+-----+-----+-----+
```

- 次のステートメントは、最初のステートメントと同じ操作を実行して同じ結果を生成する前に、文字列引数 ('127' から 127 など) に対して数値への変換を実行します:

```
mysql> SELECT '127' | '128', '128' << 2, BIT_COUNT('15');
+-----+-----+-----+
| '127' | '128' | '128' << 2 | BIT_COUNT('15') |
+-----+-----+-----+
| 255 | 512 | 4 |
+-----+-----+-----+
```

- このステートメントは、ビット演算引数に 16 進数リテラルを使用します。MySQL では、デフォルトで 16 進リテラルはバイナリ文字列として扱われますが、数値コンテキストでは数値として評価されます ([セクション9.1.4「16 進数リテラル」](#)を参照)。MySQL 8.0 より前は、数値コンテキストにビット操作が含まれていました。例:

```
mysql> SELECT X'7F' | X'80', X'80' << 2, BIT_COUNT(X'0F');
+-----+-----+-----+
| X'7F' | X'80' | X'80' << 2 | BIT_COUNT(X'0F') |
+-----+-----+-----+
| 255 | 512 | 4 |
+-----+-----+-----+
```

ビット演算でのビット値リテラルの処理は、16 進数リテラル (つまり、数値) と似ています。

MySQL 8.0 でのビット操作

MySQL 8.0 はビット操作を拡張してバイナリ文字列引数を直接処理し (変換なし)、バイナリ文字列の結果を生成します。(整数またはバイナリ文字列ではない引数は、以前と同様に整数に変換されます。) この拡張機能は、次のようにビット操作を拡張します:

- 64 ビットを超える値でビット操作が可能になります。
- バイナリ文字列として表される値に対しては、整数として表される値よりもビット演算を実行する方が簡単です。

たとえば、次のような判読可能なテキスト形式を持つ UUID 値と IPv6 アドレスについて考えてみます:

```
UUID: 6ccd780c-baba-1026-9564-5b8c656024db
```

```
IPv6: fe80::219:d1ff:fe91:1a72
```

これらの形式のテキスト文字列を操作するのは面倒です。別の方法として、デリミタなしの固定長バイナリ文字列に変換する方法があります。 `UUID_TO_BIN()` および `INET6_ATON()` はそれぞれ、バイナリ文字列 16 バイト (128 ビット) の長さのデータ型 `BINARY(16)` の値を生成します。次のステートメントは、これを示しています (`HEX()` は表示可能な値を生成するために使用されます):

```
mysql> SELECT HEX(UUID_TO_BIN('6ccd780c-baba-1026-9564-5b8c656024db'));
+-----+
| HEX(UUID_TO_BIN('6ccd780c-baba-1026-9564-5b8c656024db')) |
+-----+
| 6CCD780CBABA102695645B8C656024DB |
+-----+
mysql> SELECT HEX(INET6_ATON('fe80::219:d1ff:fe91:1a72'));
+-----+
| HEX(INET6_ATON('fe80::219:d1ff:fe91:1a72')) |
+-----+
| FE800000000000000219D1FFFE911A72 |
+-----+
```

これらのバイナリ値は、UUID 値からのタイムスタンプの抽出や、IPv6 アドレスのネットワークおよびホスト部分の抽出などのアクションを実行するビット操作で簡単に指定できます。(例については、この説明の後半のを参照してください。)

バイナリ文字列としてカウントされる引数には、カラム値、ルーチンパラメータ、ローカル変数、およびバイナリ文字列型を持つユーザー定義変数が含まれます: `BINARY`、`VARBINARY`、またはいずれかの `BLOB` タイプ。

16 進数リテラルとビットリテラルについて これらは MySQL ではデフォルトでバイナリ文字列ですが、数値コンテキストの数値であることを思い出してください。MySQL 8.0 では、ビット操作はどのように処理されますか。MySQL は、MySQL 8.0 の前と同様に、引き続き数値コンテキストで評価しますか。または、ビット操作によってバイナリ文字列として評価され、バイナリ文字列を変換せずに「ネイティブ」を処理できるようになりましたか。

回答: 16 進数リテラルまたはビットリテラルを使用してビット操作の引数を指定することは一般的であり、これらは数値を表すことを意図しているため、MySQL では、下位互換性のために、すべてのビット引数が 16 進数リテラルまたはビットリテラルである場合に、数値コンテキストでビット操作が評価され続けます。かわりにバイナリ文字列としての評価が必要な場合は、簡単に実行できます: 少なくとも 1 つのリテラルに `_binary` イントロデューサを使用します。

- 次のビット演算は、16 進数リテラルおよびビットリテラルを整数として評価します:

```
mysql> SELECT X'40' | X'01', b'11110001' & b'01001111';
+-----+
| X'40' | X'01' | b'11110001' & b'01001111' |
+-----+
|      65 |      65 |
+-----+
```

- これらのビット操作では、`_binary` イントロデューサのために 16 進数リテラルおよびビットリテラルがバイナリ文字列として評価されます:

```
mysql> SELECT _binary X'40' | X'01', b'11110001' & _binary b'01001111';
+-----+
| _binary X'40' | X'01' | b'11110001' & _binary b'01001111' |
+-----+
| A          | A          |
+-----+
```

両方のステートメントのビット操作によって数値 65 の結果が生成されますが、2 番目のステートメントはバイナリ文字列コンテキスト (65 は ASCII A) で動作します。

数値評価コンテキストでは、16 進数リテラルおよびビットリテラル引数の許容値は、結果と同様に最大 64 ビットです。対照的に、バイナリ文字列の評価コンテキストでは、許可される引数 (および結果) は 64 ビットを超えることができます:

```
mysql> SELECT _binary X'4040404040404040' | X'0102030405060708';
+-----+
| _binary X'4040404040404040' | X'0102030405060708' |
+-----+
| ABCDEFGH |
+-----+
```

ビット演算で 16 進数リテラルまたはビットリテラルを参照してバイナリ文字列評価を行うには、いくつかの方法があります:

```
_binary literal  
BINARY literal  
CAST(literal AS BINARY)
```

16 進数リテラルまたはビットリテラルの二項文字列評価を生成する別の方法は、それらをユーザ一定義変数に割り当てることです。これにより、バイナリ文字列型の変数が生成されます:

```
mysql> SET @v1 = X'40', @v2 = X'01', @v3 = b'111110001', @v4 = b'010011111';  
mysql> SELECT @v1 | @v2, @v3 & @v4;  
+-----+  
| @v1 | @v2 | @v3 & @v4 |  
+-----+  
| A   | A   |           |  
+-----+
```

バイナリ文字列コンテキストでは、ビット単位の操作引数の長さが同じである必要があります。そうでない場合は、`ER_INVALID_BITWISE_OPERANDS_SIZE` エラーが発生します:

```
mysql> SELECT _binary X'40' | X'0001';  
ERROR 3513 (HY000): Binary operands of bitwise  
operators must be of equal length
```

等価長の要件を満たすには、短い値を先頭のゼロ桁に埋めるか、長い値が先頭のゼロ桁で始まり、短い結果値が許容される場合は削除します:

```
mysql> SELECT _binary X'0040' | X'0001';  
+-----+  
| _binary X'0040' | X'0001' |  
+-----+  
| A               |         |  
+-----+  
mysql> SELECT _binary X'40' | X'01';  
+-----+  
| _binary X'40' | X'01' |  
+-----+  
| A             |       |  
+-----+
```

パディングまたはストリップングは、`LPAD()`、`RPAD()`、`SUBSTR()` や `CAST()` などの機能を使用して実行することもできます。このような場合、式の引数はすべてのリテラルではなく、`_binary` は不要になります。例:

```
mysql> SELECT LPAD(X'40', 2, X'00') | X'0001';  
+-----+  
| LPAD(X'40', 2, X'00') | X'0001' |  
+-----+  
| A                     |         |  
+-----+  
mysql> SELECT X'40' | SUBSTR(X'0001', 2, 1);  
+-----+  
| X'40' | SUBSTR(X'0001', 2, 1) |  
+-----+  
| A     |                       |  
+-----+
```

バイナリ文字列のビット操作の例

次の例は、ビット操作を使用して UUID 値の一部 (この場合はタイムスタンプと IEEE 802 ノード番号) を抽出する方法を示しています。このテクニックでは、抽出されたパーツごとにビットマスクが必要です。

テキスト UUID を対応する 16 バイトのバイナリ値に変換して、バイナリ文字列コンテキストでビット操作を使用して操作できるようにします:

```
mysql> SET @uuid = UUID_TO_BIN('6ccd780c-baba-1026-9564-5b8c656024db');  
mysql> SELECT HEX(@uuid);  
+-----+  
| HEX(@uuid) |  
+-----+
```



```

+-----+
| 6CCD780CBABA102695645B8C656024DB |
+-----+

```

値のタイムスタンプおよびノード番号部分のビットマスクを構築します。タイムスタンプは最初の 3 つの部分 (64 ビット、ビット 0 から 63) で構成され、ノード番号は最後の部分 (48 ビット、ビット 80 から 127) です:

```

mysql> SET @ts_mask = CAST(X'FFFFFFFFFFFFFFFF' AS BINARY(16));
mysql> SET @node_mask = CAST(X'FFFFFFFFFFFFFFFF' AS BINARY(16)) >> 80;
mysql> SELECT HEX(@ts_mask);
+-----+
| HEX(@ts_mask) |
+-----+
| FFFFFFFFFFFFFFFF0000000000000000 |
+-----+
mysql> SELECT HEX(@node_mask);
+-----+
| HEX(@node_mask) |
+-----+
| 000000000000000000000000FFFFFFFF |
+-----+

```

ここでは、マスクが適用される UUID 値と同じ長さである必要があるため、`CAST(... AS BINARY(16))` 関数が使用されます。他の関数を使用して必要な長さまでマスクを埋め込むと、同じ結果が生成されます:

```

SET @ts_mask= RPAD(X'FFFFFFFFFFFFFFFF', 16, X'00');
SET @node_mask = LPAD(X'FFFFFFFFFFFFFFFF', 16, X'00');

```

マスクを使用して、タイムスタンプおよびノード番号の部分を抽出します:

```

mysql> SELECT HEX(@uuid & @ts_mask) AS 'timestamp part';
+-----+
| timestamp part |
+-----+
| 6CCD780CBABA10260000000000000000 |
+-----+
mysql> SELECT HEX(@uuid & @node_mask) AS 'node part';
+-----+
| node part |
+-----+
| 0000000000000000000000005B8C656024DB |
+-----+

```

前述の例では、これらのビット操作を使用しています: 右シフト (`>>`) およびビット単位 AND (`&`)。

注記

`UUID_TO_BIN()` は、生成されるバイナリ UUID 値にビット再配置の原因となるフラグを取ります。そのフラグを使用する場合は、それに応じて抽出マスクを変更します。

次の例では、ビット操作を使用して、IPv6 アドレスのネットワーク部分およびホスト部分を抽出します。ネットワークパーツの長さが 80 ビットであるとしています。この場合、ホスト部分の長さは $128 - 80 = 48$ ビットです。アドレスのネットワーク部分およびホスト部分を抽出するには、バイナリ文字列に変換してから、バイナリ文字列コンテキストでビット操作を使用します。

テキスト IPv6 アドレスを対応するバイナリ文字列に変換します:

```

mysql> SET @ip = INET6_ATON('fe80::219:d1ff:fe91:1a72');

```

ネットワークの長さをビット単位で定義します:

```

mysql> SET @net_len = 80;

```

all-one アドレスを左または右にシフトして、ネットワークマスクとホストマスクを構築します。これを行うには、次のようなバイナリ文字列に変換することでわかるように、すべてのゼロの短縮形であるアドレス::から開始します:

```

mysql> SELECT HEX(INET6_ATON('::')) AS 'all zeros';
+-----+
| all zeros |
+-----+
| 00000000000000000000000000000000 |

```

補完値 (すべて) を生成するには、`~` 演算子を使用してビットを反転します:

```
mysql> SELECT HEX(~INET6_ATON('::')) AS 'all ones';
+-----+
| all ones |
+-----+
| FFFFFFFF |
+-----+
```

all-one 値を左または右にシフトして、ネットワークマスクとホストマスクを生成します:

```
mysql> SET @net_mask = ~INET6_ATON('::') << (128 - @net_len);
mysql> SET @host_mask = ~INET6_ATON('::') >> @net_len;
```

マスクを表示して、アドレスの正しい部分をカバーしていることを確認します:

```
mysql> SELECT INET6_NTOA(@net_mask) AS 'network mask';
+-----+
| network mask |
+-----+
| ffff:ffff:ffff:ffff: |
+-----+
mysql> SELECT INET6_NTOA(@host_mask) AS 'host mask';
+-----+
| host mask |
+-----+
| ::ffff:255.255.255.255 |
+-----+
```

アドレスのネットワーク部分とホスト部分を抽出して表示します:

```
mysql> SET @net_part = @ip & @net_mask;
mysql> SET @host_part = @ip & @host_mask;
mysql> SELECT INET6_NTOA(@net_part) AS 'network part';
+-----+
| network part |
+-----+
| fe80::219:0:0:0 |
+-----+
mysql> SELECT INET6_NTOA(@host_part) AS 'host part';
+-----+
| host part |
+-----+
| ::d1ff:fe91:1a72 |
+-----+
```

前述の例では、これらのビット操作を使用しています: 補完 (`~`)、左シフト (`<<`) およびビット単位 AND (`&`)。

残りの説明では、ビット操作の各グループの引数処理、ビット操作でのリテラル値の処理、および MySQL 8.0 と以前の MySQL バージョンとの間の潜在的な非互換性について詳しく説明します。

ビット単位の AND、OR および XOR 操作

`&`、`|` および `^` ビット操作の場合、結果の型は、引数がバイナリ文字列として評価されるか、数値として評価されるかによって異なります:

- バイナリ文字列の評価は、引数がバイナリ文字列型で、少なくともそのいずれかが 16 進数リテラル、ビットリテラルまたは `NULL` リテラルでない場合に発生します。それ以外の場合は数値の評価が行われ、引数は必要に応じて符号なし 64 ビット整数に変換されます。
- バイナリ文字列の評価では、引数と同じ長さのバイナリ文字列が生成されます。引数の長さが等しくない場合は、`ER_INVALID_BITWISE_OPERANDS_SIZE` エラーが発生します。数値評価では符号なし 64 ビット整数が生成されます。

数値評価の例:

```
mysql> SELECT 64 | 1, X'40' | X'01';
+-----+
| 65 | 41 | 01 |
+-----+
```

```
| 64 | 1 | X'40' | X'01' |
+-----+-----+
| 65 |      65 |
+-----+-----+
```

バイナリ文字列の評価の例:

```
mysql> SELECT _binary X'40' | X'01';
+-----+
|_binary X'40' | X'01' |
+-----+
| A          |
+-----+
mysql> SET @var1 = X'40', @var2 = X'01';
mysql> SELECT @var1 | @var2;
+-----+
|@var1 | @var2 |
+-----+
| A          |
+-----+
```

ビット単位の補完およびシフト操作

~、<< および >> ビット操作の場合、結果の型はビット引数がバイナリ文字列として評価されるか、数値として評価されるかによって異なります:

- バイナリ文字列の評価は、bit 引数がバイナリ文字列型で、16 進数リテラル、ビットリテラルまたは NULL リテラルでない場合に発生します。それ以外の場合は、必要に応じて引数を符号なし 64 ビット整数に変換することで、数値の評価が行われます。
- binary-string 評価では、bit 引数と同じ長さのバイナリ文字列が生成されます。数値評価では符号なし 64 ビット整数が生成されます。

シフト操作では、引数の型に関係なく、値の末尾からシフトされたビットは警告なしで失われます。特に、シフトカウントが bit 引数のビット数以上の場合、結果のすべてのビットは 0 になります。

数値評価の例:

```
mysql> SELECT ~0, 64 << 2, X'40' << 2;
+-----+-----+-----+
| ~0          | 64 << 2 | X'40' << 2 |
+-----+-----+-----+
| 18446744073709551615 | 256 | 256 |
+-----+-----+-----+
```

バイナリ文字列の評価の例:

```
mysql> SELECT HEX(_binary X'1111000022220000' >> 16);
+-----+
| HEX(_binary X'1111000022220000' >> 16) |
+-----+
| 0000111100002222          |
+-----+
mysql> SELECT HEX(_binary X'1111000022220000' << 16);
+-----+
| HEX(_binary X'1111000022220000' << 16) |
+-----+
| 0000222200000000          |
+-----+
mysql> SET @var1 = X'F0F0F0F0';
mysql> SELECT HEX(~@var1);
+-----+
| HEX(~@var1) |
+-----+
| 0F0F0F0F    |
+-----+
```

BIT_COUNT() の操作

BIT_COUNT() 関数は、常に符号なし 64 ビット整数、または引数が NULL の場合は NULL を返します。

```
mysql> SELECT BIT_COUNT(127);
+-----+
| BIT_COUNT(127) |
+-----+
|          7 |
+-----+
mysql> SELECT BIT_COUNT(b'010101'), BIT_COUNT(_binary b'010101');
+-----+-----+
| BIT_COUNT(b'010101') | BIT_COUNT(_binary b'010101') |
+-----+-----+
|          3 |          3 |
+-----+-----+
```

BIT_AND()、BIT_OR() および BIT_XOR() の操作

BIT_AND()、**BIT_OR()** および **BIT_XOR()** ビット関数の場合、結果の型は、関数の引数値がバイナリ文字列として評価されるか、数値として評価されるかによって異なります:

- バイナリ文字列の評価は、引数値がバイナリ文字列型で、引数が 16 進数リテラル、ビットリテラルまたは **NULL** リテラルでない場合に発生します。それ以外の場合は数値の評価が行われ、必要に応じて引数値が符号なし 64 ビット整数に変換されます。
- バイナリ文字列の評価では、引数値と同じ長さのバイナリ文字列が生成されます。引数値の長さが等しくない場合は、**ER_INVALID_BITWISE_OPERANDS_SIZE** エラーが発生します。引数のサイズが 511 バイトを超えると、**ER_INVALID_BITWISE_AGGREGATE_OPERANDS_SIZE** エラーが発生します。数値評価では符号なし 64 ビット整数が生成されます。

すべての値が **NULL** でないかぎり、**NULL** 値は結果に影響しません。その場合、結果は引数値の長さと同じ長さの中立値になります (**BIT_AND()** の場合はすべてのビット 1、**BIT_OR()** の場合はすべてのビット 0 および **BIT_XOR()**)。

例:

```
mysql> CREATE TABLE t (group_id INT, a VARBINARY(6));
mysql> INSERT INTO t VALUES (1, NULL);
mysql> INSERT INTO t VALUES (1, NULL);
mysql> INSERT INTO t VALUES (2, NULL);
mysql> INSERT INTO t VALUES (2, X'1234');
mysql> INSERT INTO t VALUES (2, X'FF34');
mysql> SELECT HEX(BIT_AND(a)), HEX(BIT_OR(a)), HEX(BIT_XOR(a))
FROM t GROUP BY group_id;
+-----+-----+-----+
| HEX(BIT_AND(a)) | HEX(BIT_OR(a)) | HEX(BIT_XOR(a)) |
+-----+-----+-----+
| FFFFFFFFFFFFFF | 00000000000000 | 00000000000000 |
| 1234          | FF34          | ED00          |
+-----+-----+-----+
```

16 進数リテラル、ビットリテラルおよび NULL リテラルの特殊な処理

下位互換性のため、すべてのビット引数が 16 進数リテラル、ビットリテラルまたは **NULL** リテラルの場合、MySQL 8.0 は数値コンテキストでビット操作を評価します。つまり、すべてのビット引数が 16 進数リテラル、ビットリテラルまたは **NULL** リテラルである場合、バイナリ文字列ビット引数に対するビット操作ではバイナリ文字列評価は使用されません。(**_binary** イントロデューサ、**BINARY** 演算子、またはバイナリ文字列として明示的に指定するその他の方法を使用して記述されている場合、このようなリテラルには適用されません。)

ここで説明したリテラル処理は、MySQL 8.0 の前と同じです。例:

- 次のビット操作では、数値コンテキストのリテラルが評価され、**BIGINT** 結果が生成されます:

```
b'0001' | b'0010'
X'0008' << 8
```

- これらのビット操作は、数値コンテキストで **NULL** を評価し、**NULL** 値を持つ **BIGINT** 結果を生成します:

```
NULL & NULL
NULL >> 4
```

MySQL 8.0 では、少なくとも 1 つの引数がバイナリ文字列であることを明示的に示すことで、これらの操作でバイナリ文字列コンテキストの引数を評価できます:

```
_binary b'0001' | b'0010'  
_binary X'0008' << 8  
BINARY NULL & NULL  
BINARY NULL >> 4
```

最後の 2 つの式の結果は、**BINARY** 演算子を使用しない場合と同様に **NULL** ですが、結果のデータ型は整数型ではなくバイナリ文字列型です。

MySQL 5.7 とのビット操作の非互換性

ビット操作ではバイナリ文字列引数を MySQL 8.0 でネイティブに処理できるため、一部の式では 5.7 とは異なる結果が MySQL 8.0 で生成されます。監視する問題のある 5 つの式タイプは次のとおりです:

```
nonliteral_binary {&|^} binary  
binary {&|^} nonliteral_binary  
nonliteral_binary {<<>>} anything  
~ nonliteral_binary  
AGGR_BIT_FUNC(nonliteral_binary)
```

これらの式は、8.0 のバイナリ文字列である MySQL 5.7 の **BIGINT** を返します。

表記法:

- **{ op1 op2 ... }**: 指定された式タイプに適用される演算子のリスト。
- **binary**: 16 進数リテラル、ビットリテラルまたは **NULL** リテラルを含む任意の種類のバイナリ文字列引数。
- **nonliteral_binary**: 16 進数リテラル、ビットリテラルまたは **NULL** リテラル以外のバイナリ文字列値である引数。
- **AGGR_BIT_FUNC**: ビット値引数を取る集計関数: **BIT_AND()**, **BIT_OR()**, **BIT_XOR()**。

MySQL 5.7 と 8.0 間の潜在的な非互換性のために MySQL 5.7 で準備する方法の詳細は、[MySQL 5.7 Reference Manual](#) の [Bit Functions and Operators](#) を参照してください。

次のリストでは、使用可能なビット関数および演算子について説明します:

- |

ビット単位の OR

結果の型は、引数がバイナリ文字列として評価されるか数値として評価されるかによって異なります:

- バイナリ文字列の評価は、引数がバイナリ文字列型で、少なくともそのいずれかが 16 進数リテラル、ビットリテラルまたは **NULL** リテラルでない場合に発生します。それ以外の場合は数値の評価が行われ、引数は必要に応じて符号なし 64 ビット整数に変換されます。
- バイナリ文字列の評価では、引数と同じ長さのバイナリ文字列が生成されます。引数の長さが等しくない場合は、**ER_INVALID_BITWISE_OPERANDS_SIZE** エラーが発生します。数値評価では符号なし 64 ビット整数が生成されます。

詳細は、このセクションの概要の説明を参照してください。

```
mysql> SELECT 29 | 15;  
-> 31  
mysql> SELECT _binary X'40404040' | X'01020304';  
-> 'ABCD'
```

- &

ビット単位の AND

結果の型は、引数がバイナリ文字列として評価されるか数値として評価されるかによって異なります:

- バイナリ文字列の評価は、引数がバイナリ文字列型で、少なくともそのいずれかが 16 進数リテラル、ビットリテラルまたは `NULL` リテラルでない場合に発生します。それ以外の場合は数値の評価が行われ、引数は必要に応じて符号なし 64 ビット整数に変換されます。
- バイナリ文字列の評価では、引数と同じ長さのバイナリ文字列が生成されます。引数の長さが等しくない場合は、`ER_INVALID_BITWISE_OPERANDS_SIZE` エラーが発生します。数値評価では符号なし 64 ビット整数が生成されます。

詳細は、このセクションの概要の説明を参照してください。

```
mysql> SELECT 29 & 15;
-> 13
mysql> SELECT HEX(_binary X'FF' & b'11110000');
-> 'F0'
```

• ^

ビット単位の XOR

結果の型は、引数がバイナリ文字列として評価されるか数値として評価されるかによって異なります:

- バイナリ文字列の評価は、引数がバイナリ文字列型で、少なくともそのいずれかが 16 進数リテラル、ビットリテラルまたは `NULL` リテラルでない場合に発生します。それ以外の場合は数値の評価が行われ、引数は必要に応じて符号なし 64 ビット整数に変換されます。
- バイナリ文字列の評価では、引数と同じ長さのバイナリ文字列が生成されます。引数の長さが等しくない場合は、`ER_INVALID_BITWISE_OPERANDS_SIZE` エラーが発生します。数値評価では符号なし 64 ビット整数が生成されます。

詳細は、このセクションの概要の説明を参照してください。

```
mysql> SELECT 1 ^ 1;
-> 0
mysql> SELECT 1 ^ 0;
-> 1
mysql> SELECT 11 ^ 3;
-> 8
mysql> SELECT HEX(_binary X'FEDC' ^ X'1111');
-> 'EFGD'
```

• <<

`longlong (BIGINT)` 数値またはバイナリ文字列を左にシフトします。

結果の型は、`bit` 引数がバイナリ文字列として評価されるか、数値として評価されるかによって異なります:

- バイナリ文字列の評価は、`bit` 引数がバイナリ文字列型で、16 進数リテラル、ビットリテラルまたは `NULL` リテラルでない場合に発生します。それ以外の場合は、必要に応じて引数を符号なし 64 ビット整数に変換することで、数値の評価が行われます。
- `binary-string` 評価では、`bit` 引数と同じ長さのバイナリ文字列が生成されます。数値評価では符号なし 64 ビット整数が生成されます。

値の末尾からシフトされたビットは、引数の型に関係なく、警告なしで失われます。特に、シフトカウントが `bit` 引数のビット数以上の場合、結果のすべてのビットは 0 になります。

詳細は、このセクションの概要の説明を参照してください。

```
mysql> SELECT 1 << 2;
-> 4
mysql> SELECT HEX(_binary X'00FF00FF00FF' << 8);
-> 'FF00FF00FF00'
```

• >>

`longlong (BIGINT)` 数値またはバイナリ文字列を右にシフトします。

結果の型は、bit 引数がバイナリ文字列として評価されるか、数値として評価されるかによって異なります:

- バイナリ文字列の評価は、bit 引数がバイナリ文字列型で、16 進数リテラル、ビットリテラルまたは `NULL` リテラルでない場合に発生します。それ以外の場合は、必要に応じて引数を符号なし 64 ビット整数に変換することで、数値の評価が行われます。
- `binary-string` 評価では、bit 引数と同じ長さのバイナリ文字列が生成されます。数値評価では符号なし 64 ビット整数が生成されます。

値の末尾からシフトされたビットは、引数の型に関係なく、警告なしで失われます。特に、シフトカウントが bit 引数のビット数以上の場合、結果のすべてのビットは 0 になります。

詳細は、このセクションの概要の説明を参照してください。

```
mysql> SELECT 4 >> 2;
-> 1
mysql> SELECT HEX(_binary X'00FF00FF00FF' >> 8);
-> '0000FF00FF00'
```

• ~

すべてのビットを反転します。

結果の型は、bit 引数がバイナリ文字列として評価されるか、数値として評価されるかによって異なります:

- バイナリ文字列の評価は、bit 引数がバイナリ文字列型で、16 進数リテラル、ビットリテラルまたは `NULL` リテラルでない場合に発生します。それ以外の場合は、必要に応じて引数を符号なし 64 ビット整数に変換することで、数値の評価が行われます。
- `binary-string` 評価では、bit 引数と同じ長さのバイナリ文字列が生成されます。数値評価では符号なし 64 ビット整数が生成されます。

詳細は、このセクションの概要の説明を参照してください。

```
mysql> SELECT 5 & ~1;
-> 4
mysql> SELECT HEX(~X'0000FFFF1111EEEE');
-> 'FFFF0000EEEE1111'
```

• BIT_COUNT(N)

引数 `N` に符号なし 64 ビット整数として設定されているビット数を返します。引数が `NULL` の場合は `NULL` を返します。

```
mysql> SELECT BIT_COUNT(64), BIT_COUNT(BINARY 64);
-> 1, 7
mysql> SELECT BIT_COUNT('64'), BIT_COUNT(_binary '64');
-> 1, 7
mysql> SELECT BIT_COUNT(X'40'), BIT_COUNT(_binary X'40');
-> 1, 1
```

12.14 暗号化関数と圧縮関数

表 12.18 「暗号化関数」

名前	説明
<code>AES_DECRYPT()</code>	AES を使用して復号化します
<code>AES_ENCRYPT()</code>	AES を使用して暗号化します
<code>COMPRESS()</code>	バイナリ文字列として結果を返します
<code>MD5()</code>	MD5 チェックサムを計算します
<code>RANDOM_BYTES()</code>	ランダムなバイトベクトルを返します
<code>SHA1()</code> , <code>SHA()</code>	SHA-1 160 ビットチェックサムを計算します

名前	説明
SHA2()	SHA-2 チェックサムを計算します
STATEMENT_DIGEST()	ステートメントダイジェストハッシュ値の計算
STATEMENT_DIGEST_TEXT()	正規化されたステートメントダイジェストの計算
UNCOMPRESS()	圧縮された文字列を圧縮解除します
UNCOMPRESSED_LENGTH()	圧縮前の文字列長を返します
VALIDATE_PASSWORD_STRENGTH()	パスワードの強度を判断します

多くの暗号化関数および圧縮関数では、結果に任意のバイト値が含まれている可能性のある文字列が返されます。これらの結果を格納する場合は、[VARBINARY](#) または [BLOB](#) バイナリ文字列データ型の列を使用します。これにより、バイナリ以外の文字列データ型 ([CHAR](#), [VARCHAR](#), [TEXT](#)) を使用している場合など、データ値を変更する可能性がある後続の領域削除または文字セット変換の潜在的な問題を回避できます。

一部の暗号化関数は ASCII 文字の文字列を返します: [MD5\(\)](#), [SHA\(\)](#), [SHA1\(\)](#), [SHA2\(\)](#), [STATEMENT_DIGEST\(\)](#), [STATEMENT_DIGEST_TEXT\(\)](#)。戻り値は、[character_set_connection](#) および [collation_connection](#) システム変数によって決定される文字セットと照合順序を持つ文字列です。これは、文字セットが [binary](#) でないかぎり、非バイナリ文字列です。

アプリケーションで 16 進数の文字列を返す関数 ([MD5\(\)](#) や [SHA1\(\)](#) など) からの値を格納する場合は、[UNHEX\(\)](#) を使用して 16 進表現をバイナリに変換し、その結果を [BINARY\(N\)](#) 列に格納すれば、より効率的な格納および比較を実現できます。16 進数の各ペアにはバイナリ形式で 1 バイトが必要であるため、[N](#) の値は 16 進数文字列の長さに依存します。[N](#) は、[MD5\(\)](#) 値の場合は 16、[SHA1\(\)](#) の場合は 20 です。[SHA2\(\)](#) の場合、[N](#) の範囲は、結果の目的のビット長を指定する引数に応じて 28 から 32 までです。

[utf8](#) 文字セット (文字ごとに 4 バイト使用されます) が使用される列に値が格納される場合に、16 進文字列を [CHAR](#) 列に格納する際のサイズのペナルティーは最小で 2 回、最大で 8 回です。また、文字列を格納すると、値が大きくなり、文字セットの照合順序ルールを考慮に入れる必要があるため、比較が遅くなります。

アプリケーションで [MD5\(\)](#) 文字列値が [CHAR\(32\)](#) 列に格納されると仮定します。

```
CREATE TABLE md5_tbl (md5_val CHAR(32), ...);
INSERT INTO md5_tbl (md5_val, ...) VALUES(MD5('abcdef'), ...);
```

16 進文字列をよりコンパクトな形式に変換するには、次のように、代わりに [UNHEX\(\)](#) および [BINARY\(16\)](#) が使用されるようにアプリケーションを変更します。

```
CREATE TABLE md5_tbl (md5_val BINARY(16), ...);
INSERT INTO md5_tbl (md5_val, ...) VALUES(UNHEX(MD5('abcdef')), ...);
```

ハッシュ関数が 2 つの異なる入力値に同じ値を生成するという非常にまれなケースに対応できるように、アプリケーションが準備されるはずですが。競合を検出可能にする方法の 1 つは、ハッシュ列を主キーにすることです。

注記

MD5 および SHA-1 アルゴリズムの悪用が知られています。かわりに、このセクションで説明されている別の一方暗号化機能 ([SHA2\(\)](#) など) の使用を検討してください。

注意

暗号化機能の引数として指定されたパスワードまたはその他の機密値は、SSL 接続が使用されないかぎり、クリアテキストとして MySQL サーバーに送信されます。また、このような値は、書き込まれるすべての MySQL ログに表示されます。このようなタイプの露出を回避するために、アプリケーションはクライアント側で機密の値を暗号化してから、サーバーに送信できます。同じ考慮事項が暗号化鍵にも適用されます。これらの露出を回避するために、アプリケーションはストアプロシージャを使用して、サーバー側で値を暗号化および復号化できます。

- [AES_DECRYPT\(crypt_str,key_str\[,init_vector\]\)](#)

この関数は、公式の AES (Advanced Encryption Standard) アルゴリズムを使用してデータを復号化します。詳細は、[AES_ENCRYPT\(\)](#) の説明を参照してください。

`AES_DECRYPT()` を使用するステートメントは、ステートメントベースのレプリケーションでは安全ではありません。

- `AES_ENCRYPT(str,key_str[,init_vector])`

`AES_ENCRYPT()` および `AES_DECRYPT()` では、AES (Advanced Encryption Standard) アルゴリズム (以前は「Rijndael」と呼ばれていました) を使用したデータの暗号化および復号化が実装されます。AES の標準では、さまざまな鍵の長さが許可されます。デフォルトでは、これらの関数で鍵の長さが 128 ビットの AES が実装されます。あとで説明するように、196 または 256 ビットの鍵の長さを使用できます。鍵の長さは、パフォーマンスとセキュリティの間でのトレードオフです。

`AES_ENCRYPT()` は、鍵文字列 `key_str` を使用して文字列 `str` を暗号化し、暗号化された出力を含むバイナリ文字列を返します。`AES_DECRYPT()` は、鍵文字列 `key_str` を使用して暗号化された文字列 `crypt_str` を復号化し、元のプレーンテキスト文字列を返します。関数引数のいずれかが `NULL` の場合は、関数で `NULL` が返されます。

`str` および `crypt_str` 引数は任意の長さにするのができ、AES などのブロックベースのアルゴリズムによって、必要に応じてブロックの倍数になるように、自動的にパディングが `str` に追加されます。このパディングは、`AES_DECRYPT()` 関数によって自動的に削除されます。`crypt_str` の長さは、次の公式を使用して計算できます。

```
16 * (trunc(string_length / 16) + 1)
```

鍵の長さが 128 ビットの場合、`key_str` 引数に鍵を渡すもっともセキュアな方法は、完全にランダムな 128 ビット値を作成し、それをバイナリ値として渡すことです。例:

```
INSERT INTO t
VALUES (1,AES_ENCRYPT('text',UNHEX('F3229A0B371ED2D9441B830D21A390C3')));
```

パスフレーズを使用すると、パスフレーズをハッシュ化することで AES 鍵を生成できます。例:

```
INSERT INTO t
VALUES (1,AES_ENCRYPT('text', UNHEX(SHA2('My secret passphrase',512))));
```

パスワードまたはパスフレーズは直接 `crypt_str` に渡さず、最初にハッシュ化してください。このドキュメントの以前のバージョンでは、従来のアプローチが提案されていましたが、ここで示す例の方がセキュアであるため、推奨されなくなりました。

`AES_DECRYPT()` で無効な日付または不正なパディングが検出された場合は、`NULL` が返されます。ただし、入力データまたは鍵が無効になっている場合は、`AES_DECRYPT()` で `NULL` 以外の値 (ごみの可能性もあります) が返される可能性があります。

`AES_ENCRYPT()` および `AES_DECRYPT()` は、ブロック暗号化モードの制御を許可し、オプションの `init_vector` 初期化ベクトル引数を取ります:

- `block_encryption_mode` システム変数は、ブロックベースの暗号化アルゴリズムのモードを制御します。そのデフォルト値は、128 ビットの鍵の長さで ECB モードを使用した暗号化を表す `aes-128-ecb` です。この変数で許可されている値については、[セクション5.1.8「サーバースystem変数」](#)を参照してください。
- オプションの `init_vector` 引数では、必要とするブロック暗号化モードに対応する初期化ベクトルが提供されます。

オプションの `init_vector` 引数が必要なモードでは、16 バイト以上の長さにする必要があります (16 を超えるバイトは無視されます)。`init_vector` が欠落している場合は、エラーが発生します。

`init_vector` が必要ないモードでは、これが無視され、指定されている場合は警告が生成されます。

`RANDOM_BYTES(16)` を呼び出すと、初期化ベクトルに使用されるバイトのランダム文字列を生成できます。初期化ベクトルが必要な暗号化モードでは、暗号化および復号化でも同じベクトルを使用する必要があります。

```
mysql> SET block_encryption_mode = 'aes-256-cbc';
mysql> SET @key_str = SHA2('My secret passphrase',512);
mysql> SET @init_vector = RANDOM_BYTES(16);
mysql> SET @crypt_str = AES_ENCRYPT('text',@key_str,@init_vector);
mysql> SELECT AES_DECRYPT(@crypt_str,@key_str,@init_vector);
```

```

+-----+
| AES_DECRYPT(@crypt_str,@key_str,@init_vector) |
+-----+
| text |
+-----+

```

次のテーブルに、許可される各ブロック暗号化モードと、初期化ベクトル引数が必要かどうかを示します。

ブロック暗号化モード	初期化ベクトルが必要
ECB	いいえ
CBC	はい
CFB1	はい
CFB8	はい
CFB128	はい
OFB	はい

[AES_ENCRYPT\(\)](#) または [AES_DECRYPT\(\)](#) を使用するステートメントは、ステートメントベースのレプリケーションでは安全ではありません。

- [COMPRESS\(string_to_compress\)](#)

文字列を圧縮し、その結果をバイナリ文字列として返します。この関数を使用するには、MySQL が [zlib](#) などの圧縮ライブラリを使用してコンパイルされている必要があります。そうでない場合、戻り値は常に [NULL](#) になります。圧縮された文字列は、[UNCOMPRESS\(\)](#) を使用して圧縮解除できます。

```

mysql> SELECT LENGTH(COMPRESS(REPEAT('a',1000)));
-> 21
mysql> SELECT LENGTH(COMPRESS(""));
-> 0
mysql> SELECT LENGTH(COMPRESS('a'));
-> 13
mysql> SELECT LENGTH(COMPRESS(REPEAT('a',16)));
-> 15

```

圧縮された文字列の内容は、次の方法で格納されます。

- 空の文字列は、空の文字列として格納されます。
- 空以外の文字列は、4 バイトの長さの圧縮されていない文字列として格納され (低いバイトが 1 番目)、そのあとに圧縮された文字列が続きます。文字列が空白文字で終わる場合は、結果が [CHAR](#) または [VARCHAR](#) カラムに格納されていても、末尾の空白文字が削除されるという問題が回避されるように、文字が追加されます。(ただし、[CHAR](#) や [VARCHAR](#) などの非バイナリ文字列のデータ型を使用して圧縮された文字列を格納すると、文字セットの変換が発生する可能性があるため、いずれにしても推奨されません。代わりに、[VARBINARY](#) または [BLOB](#) バイナリ文字列のカラムを使用してください。)
- [MD5\(str\)](#)

文字列の MD5 128 ビットチェックサムを計算します。値は 32 桁の 16 進数の文字列、または引数が [NULL](#) の場合は [NULL](#) として返されます。たとえば、戻り値をハッシュ鍵として使用できます。効率的なハッシュ値の格納については、このセクションの冒頭で示した注記を参照してください。

戻り値は、接続文字セットの文字列です。

FIPS モードが有効な場合、[MD5\(\)](#) は [NULL](#) を返します。 [セクション6.8「FIPS のサポート」](#) を参照してください。

```

mysql> SELECT MD5('testing');
-> 'ae2b1fca515949e5d54fb22b8ed95575'

```

これは、「RSA Data Security, Inc. MD5 Message-Digest Algorithm」です。

このセクションの冒頭で示した MD5 アルゴリズムに関する注記を参照してください。

- `RANDOM_BYTES(len)`

この関数は、SSL ライブラリの乱数ジェネレータを使用して生成された `len` ランダムバイトのバイナリ文字列を戻します。 `len` で許可されている値は、1 から 1024 までの範囲内です。その範囲外の値の場合は、エラーが発生します。

`RANDOM_BYTES()` を使用すると、`AES_DECRYPT()` および `AES_ENCRYPT()` 関数に初期化ベクトルを提供できます。このコンテキストで使用するには、`len` を 16 以上にする必要があります。さらに大きい値も許可されますが、16 を超えるバイトは無視されます。

`RANDOM_BYTES()` は、その結果を非決定的にするランダムな値を生成します。したがって、この関数を使用するステートメントは、ステートメントベースレプリケーションでは安全ではありません。

- `SHA1(str)`, `SHA(str)`

RFC 3174 (Secure Hash Algorithm) で説明されているように、文字列の SHA-1 160 ビットチェックサムを計算します。値は 40 桁の 16 進数の文字列、または引数が `NULL` の場合は `NULL` として返されます。この関数を使用する一例として、ハッシュ鍵が考えられます。効率的なハッシュ値の格納については、このセクションの冒頭で示した注記を参照してください。 `SHA()` は `SHA1()` のシノニムです。

戻り値は、接続文字セットの文字列です。

```
mysql> SELECT SHA1('abc');
-> 'a9993e364706816aba3e25717850c26c9cd0d89d'
```

`SHA1()` は `MD5()` と同等ですが、暗号化に関してはよりセキュアであると考えられます。ただし、このセクションの冒頭で示した `MD5` と `SHA-1` アルゴリズムに関する注記を参照してください。

- `SHA2(str, hash_length)`

SHA-2 ファミリのハッシュ関数 (`SHA-224`、`SHA-256`、`SHA-384`、および `SHA-512`) を計算します。最初の引数は、ハッシュされるプレーンテキスト文字列です。2 番目の引数には、結果の目的のビット長が指定されます。値は、224、256、384、512、または 0 (256 と同等です) にする必要があります。引数のいずれかが `NULL` の場合や、ハッシュの長さが許可される値のいずれでもない場合は、戻り値が `NULL` になります。それ以外の場合は、関数の結果が目的のビット数が含まれるハッシュ値になります。効率的なハッシュ値の格納については、このセクションの冒頭で示した注記を参照してください。

戻り値は、接続文字セットの文字列です。

```
mysql> SELECT SHA2('abc', 224);
-> '23097d223405d8228642a477bda255b32aadbce4bda0b3f7e36c9da7'
```

この関数は、MySQL が SSL サポートで構成されている場合のみ機能します。 [セクション6.3「暗号化された接続の使用」](#) を参照してください。

`SHA2()` は `MD5()` や `SHA1()` よりも、暗号化に関してはよりセキュアであると考えられます。

- `STATEMENT_DIGEST(statement)`

SQL ステートメントを文字列として指定すると、ステートメントダイジェストハッシュ値を接続文字セットの文字列として、または引数が `NULL` の場合は `NULL` として戻します。関連する `STATEMENT_DIGEST_TEXT()` 関数は、正規化されたステートメントダイジェストを返します。ステートメントダイジェストの詳細は、[セクション27.10「パフォーマンススキーマのステートメントダイジェストとサンプリング」](#) を参照してください。

どちらの関数も、MySQL パーサーを使用してステートメントを解析します。解析に失敗すると、エラーが発生します。エラーメッセージに解析エラーが含まれるのは、ステートメントがリテラル文字列として指定されている場合のみです。

`max_digest_length` システム変数は、正規化されたステートメントダイジェストを計算するためにこれらの関数で利用できる最大バイト数を決定します。

```
mysql> SET @stmt = 'SELECT * FROM mytable WHERE cola = 10 AND colb = 20';
mysql> SELECT STATEMENT_DIGEST(@stmt);
+-----+
| STATEMENT_DIGEST(@stmt) |
```



```

+-----+
| 3bb95eeade896657c4526e74ff2a2862039d0a0fe8a9e7155b5fe492cbd78387 |
+-----+
mysql> SELECT STATEMENT_DIGEST_TEXT(@stmt);
+-----+
| STATEMENT_DIGEST_TEXT(@stmt) |
+-----+
| SELECT * FROM `mytable` WHERE `cola` = ? AND `colb` = ? |
+-----+

```

- [STATEMENT_DIGEST_TEXT\(statement\)](#)

SQL ステートメントを文字列として指定すると、正規化されたステートメントダイジェストを接続文字セットの文字列として、または引数が `NULL` の場合は `NULL` として返します。その他の説明と例については、関連する [STATEMENT_DIGEST\(\)](#) 関数の説明を参照してください。

- [UNCOMPRESS\(string_to_uncompress\)](#)

[COMPRESS\(\)](#) 関数で圧縮された文字列を圧縮解除します。引数が圧縮された値でない場合は、結果が `NULL` になります。この関数を使用するには、MySQL が `zlib` などの圧縮ライブラリを使用してコンパイルされている必要があります。そうでない場合、戻り値は常に `NULL` になります。

```

mysql> SELECT UNCOMPRESS(COMPRESS('any string'));
-> 'any string'
mysql> SELECT UNCOMPRESS('any string');
-> NULL

```

- [UNCOMPRESSED_LENGTH\(compressed_string\)](#)

圧縮された文字列が圧縮される前の長さを返します。

```

mysql> SELECT UNCOMPRESSED_LENGTH(COMPRESS(REPEAT('a',30)));
-> 30

```

- [VALIDATE_PASSWORD_STRENGTH\(str\)](#)

プレーンテキストパスワードを表す引数を指定すると、この関数はパスワードの強さを示す整数を返します。戻り値は、0 (弱) から 100 (強) までの範囲内です。

[VALIDATE_PASSWORD_STRENGTH\(\)](#) によるパスワード評価は、[validate_password](#) コンポーネントによって実行されます。そのコンポーネントがインストールされていない場合、この関数は常に 0 を返します。[validate_password](#) のインストールの詳細は、[セクション6.4.3「パスワード検証コンポーネント」](#)を参照してください。パスワードテストに影響するパラメータを調査または構成するには、[validate_password](#) によって実装されるシステム変数を確認または設定します。[セクション6.4.3.2「パスワード検証オプションおよび変数」](#)を参照してください。

パスワードは、一段と厳密になったテストの対象であり、戻り値は、次の表に示すように、どのテストに合格したのかを示します。また、[validate_password.check_user_name](#) システム変数が有効で、パスワードがユーザー名と一致する場合、他の [validate_password](#) システム変数の設定方法に関係なく、[VALIDATE_PASSWORD_STRENGTH\(\)](#) は 0 を返します。

パスワードテスト	戻り値
長さ <= 4	0
長さ ≥4 および < validate_password.length	25
ポリシー 1 を満たす (LOW)	50
ポリシー 2 を満たす (MEDIUM)	75
ポリシー 3 を満たす (STRONG)	100

12.15 ロック関数

このセクションでは、ユーザーレベルロックの操作に使用される関数について説明します。

表 12.19 「ロック関数」

名前	説明
<code>GET_LOCK()</code>	名前付きロックを取得します
<code>IS_FREE_LOCK()</code>	名前付きロックが解放されているかどうか
<code>IS_USED_LOCK()</code>	指定されたロックが使用中かどうか。true の場合は接続識別子を返します
<code>RELEASE_ALL_LOCKS()</code>	現在の名前付きロックをすべて解放
<code>RELEASE_LOCK()</code>	指定したロックを解放

- `GET_LOCK(str,timeout)`

`timeout` 秒のタイムアウトを使用して、文字列 `str` で指定された名前でのロックの取得を試みます。負の `timeout` 値は、無限のタイムアウトを表します。ロックは排他的です。あるセッションで保持されている間は、他のセッションは同じ名前のロックを取得できません。

ロックの取得に成功した場合は `1` を返し、試行がタイムアウトになった場合 (たとえば、ほかのクライアントがすでにその名前をロックしている場合) は `0` を返し、エラー (メモリー不足や `mysqladmin kill` によるスレッドの停止など) が発生した場合は `NULL` を返します。

`GET_LOCK()` で取得されたロックは、`RELEASE_LOCK()` を実行して明示的に解放されるか、セッションの終了時に自動的に解放されます (通常または異常)。`GET_LOCK()` で取得されたロックは、トランザクションのコミットまたはロールバック時に解放されません。

`GET_LOCK()` は、メタデータロック (MDL) サブシステムを使用して実装されます。複数の同時ロックを取得でき、`GET_LOCK()` は既存のロックを解放しません。たとえば、次のステートメントを実行するとします:

```
SELECT GET_LOCK('lock1',10);
SELECT GET_LOCK('lock2',10);
SELECT RELEASE_LOCK('lock2');
SELECT RELEASE_LOCK('lock1');
```

2 つ目の `GET_LOCK()` は 2 つ目のロックを取得し、両方の `RELEASE_LOCK()` コールは `1` (成功) を返します。

特定のセッションで同じ名前の複数のロックを取得することもできます。他のセッションは、取得しているセッションがその名前のロックをすべて解放するまで、その名前のロックを取得できません。

`GET_LOCK()` で取得された一意の名前付きロックは、パフォーマンススキーマの `metadata_locks` テーブルに表示されます。`OBJECT_TYPE` カラムには `USER LEVEL LOCK` と示され、`OBJECT_NAME` カラムにはロック名が表示されます。same 名に対して複数のロックが取得された場合、その名前の最初のロックのみが `metadata_locks` テーブルの行を登録します。名前の後続のロックでは、ロック内のカウンタが増分されますが、追加のメタデータロックは取得されません。名前の最後のロックインスタンスが解放されると、ロックの `metadata_locks` 行が削除されます。

複数のロックを取得する機能は、クライアント間でデッドロックが発生する可能性があることを意味します。これが発生すると、サーバーは呼出し側を選択し、そのロック取得リクエストを `ER_USER_LOCK_DEADLOCK` エラーで終了します。このエラーによってトランザクションがロールバックされることはありません。

MySQL では、64 文字のロック名に最大長が適用されます。

`GET_LOCK()` を使用すると、アプリケーションロックを実装したり、レコードロックのシミュレーションを行ったりできます。名前はサーバー全体にわたってロックされます。1 つのセッション内で名前がロックされた場合は、`GET_LOCK()` によって、別のセッションによる同じ名前を持つロックのリクエストがブロックされます。これにより、指定されたロック名について合意したクライアントは、その名前を使用すると共同のアドバイザリロックを実行できます。ただし、共同するクライアントのセットに属さないクライアントも、不注意と故意のどちらでも、名前をロックすることに注意してください。したがって、共同するクライアントがその名前をロックできないようにしてください。この可能性を減らす方法の 1 つは、データベース固有またはアプリケーション固有のロック名を使用することです。たとえば、`db_name.str` または `app_name.str` 形式のロック名を使用します。

複数のクライアントがロックを待機している場合、それらがロックを取得する順序は定義されていません。アプリケーションでは、クライアントがロックリクエストを発行した順序と同じ順序でロックを取得することを想定しないでください。

`GET_LOCK()` は、ステートメントベースのレプリケーションでは安全ではありません。 `binlog_format` が `STATEMENT` に設定されているときに、この関数を使用すると、警告のログが記録されます。

注意

複数の名前付きロックを取得する機能を使用すると、単一のステートメントで多数のロックを取得できます。例:

```
INSERT INTO ... SELECT GET_LOCK(t1.col_name) FROM t1;
```

これらのタイプのステートメントには、特定の悪影響がある場合があります。たとえば、ステートメントが途中で失敗してロールバックされた場合、障害ポイントまで取得されたロックは引き続き存在します。目的が、挿入された行と取得されたロックの間に対応するものである場合、その目的は満たされません。また、ロックが特定の順序で付与されることが重要な場合は、オプティマイザが選択する実行計画によって結果セットの順序が異なる可能性があることに注意してください。このような理由から、アプリケーションをステートメントごとに単一のロック取得コールに制限することをお勧めします。

プラグインサービスまたは一連のユーザー定義関数として、別のロックインタフェースを使用できます。このインタフェースは、`GET_LOCK()` および関連する関数によって提供されるインタフェースとは異なり、ロック名前空間と個別の読み取り/書き込みロックを提供します。詳細は、[セクション5.6.8.1「ロックサービス」](#)を参照してください。

• `IS_FREE_LOCK(str)`

`str` という名前が付けられたロックが使用可能であるか (つまり、ロックされていないか) どうかをチェックします。ロックが使用可能である (だれもロックを使用していない) 場合は `1` を返し、使用中である場合は `0` を返し、エラー (不正な引数など) が発生した場合は `NULL` を返します。

この関数は、ステートメントベースのレプリケーションでは安全に使用できません。 `binlog_format` が `STATEMENT` に設定されているときに、この関数を使用すると、警告のログが記録されます。

• `IS_USED_LOCK(str)`

`str` という名前が付けられたロックが使用中であるか (つまり、ロックされているか) どうかをチェックします。その場合は、ロックを保持するクライアントセッションの接続識別子を返します。そうでない場合は、`NULL` を返します。

この関数は、ステートメントベースのレプリケーションでは安全に使用できません。 `binlog_format` が `STATEMENT` に設定されているときに、この関数を使用すると、警告のログが記録されます。

• `RELEASE_ALL_LOCKS()`

現在のセッションで保持されているすべての名前付きロックを解放し、解放されたロックの数を返します (存在しなかった場合は `0`)

この関数は、ステートメントベースのレプリケーションでは安全に使用できません。 `binlog_format` が `STATEMENT` に設定されているときに、この関数を使用すると、警告のログが記録されます。

• `RELEASE_LOCK(str)`

`GET_LOCK()` を使用して取得された文字列 `str` によって名前が付けられたロックを解除します。ロックが解除された場合は `1` を返し、このスレッドによってロックが確立されなかった場合 (その場合、ロックは解除されません) は

0 を返し、名前付きのロックが存在しない場合は `NULL` を返します。 `GET_LOCK()` を呼び出しても取得されなかった場合や、事前に解除された場合は、ロックが存在しません。

`DO` ステートメントは、`RELEASE_LOCK()` とともに使用すると便利です。 [セクション13.2.3「DO ステートメント」](#) を参照してください。

この関数は、ステートメントベースのレプリケーションでは安全に使用できません。 `binlog_format` が `STATEMENT` に設定されているときに、この関数を使用すると、警告のログが記録されます。

12.16 情報関数

表 12.20 「情報関数」

名前	説明
<code>BENCHMARK()</code>	式を繰り返し実行します
<code>CHARSET()</code>	引数の文字セットを返します
<code>COERCIBILITY()</code>	文字列引数の照合順序強制性値を返します
<code>COLLATION()</code>	文字列引数の照合順序を返します
<code>CONNECTION_ID()</code>	接続のための接続 ID (スレッド ID) を返します
<code>CURRENT_ROLE()</code>	現在アクティブなロールを返します
<code>CURRENT_USER()</code> , <code>CURRENT_USER</code>	認証済みユーザー名とホスト名
<code>DATABASE()</code>	デフォルト (現在) のデータベース名を返します
<code>FOUND_ROWS()</code>	<code>LIMIT</code> 句付き <code>SELECT</code> で、 <code>LIMIT</code> 句がない場合に戻される可能性がある行の数です
<code>ICU_VERSION()</code>	ICU ライブラリバージョン
<code>LAST_INSERT_ID()</code>	前回の <code>INSERT</code> での <code>AUTOINCREMENT</code> カラムの値です
<code>ROLES_GRAPHML()</code>	メモリーロールのサブグラフを表す GraphML ドキュメントを返します
<code>ROW_COUNT()</code>	更新された行数
<code>SCHEMA()</code>	<code>DATABASE()</code> のシノニムです
<code>SESSION_USER()</code>	<code>USER()</code> のシノニムです
<code>SYSTEM_USER()</code>	<code>USER()</code> のシノニムです
<code>USER()</code>	ユーザー名と、クライアントによって提供されるホスト名です
<code>VERSION()</code>	MySQL サーバーのバージョンを示す文字列を返します

- `BENCHMARK(count,expr)`

`BENCHMARK()` 関数は、式 `expr` を `count` の回数だけ繰り返し実行します。MySQL による式の処理速度を計測する際に使用される場合もあります。 `NULL` や負の繰り返し回数などの不適切な引数の場合、結果値は 0 または `NULL` です。

この使用目的は、`mysql` クライアント内から、クエリーの実行時間をレポートすることです。

```
mysql> SELECT BENCHMARK(1000000,AES_ENCRYPT('hello','goodbye'));
+-----+
| BENCHMARK(1000000,AES_ENCRYPT('hello','goodbye')) |
+-----+
|                                0 |
+-----+
1 row in set (4.74 sec)
```

レポートされる時間は、クライアント側での経過時間であり、サーバー側での CPU 時間ではありません。 `BENCHMARK()` を複数回実行し、サーバーマシン上の負荷量について結果を解釈することをお勧めします。

BENCHMARK() の目的は、スカラー式の実行時パフォーマンスを測定することです。これにより、その使用方法や結果の解釈方法について、重要ないくつかの推測が提供されます。

- スカラー式しか使用できません。式をサブクエリーにすることはできますが、単一の列および最大でも単一の行が返される必要があります。たとえば、テーブル `t` に複数の列または複数の行がある場合、`BENCHMARK(10, (SELECT * FROM t))` は失敗します。
- `SELECT expr` ステートメントを `N` 回実行する場合と、`SELECT BENCHMARK(N, expr)` を実行する場合とは、発生するオーバーヘッドの量が異なります。この 2 つは非常に異なる実行プロファイルを持つため、両者の所要時間は同一になりません。前者では、パーサー、オプティマイザ、テーブルロック、および実行時評価がそれぞれ `N` 回ずつ発生します。後者では、実行時評価のみが `N` 回発生し、その他のすべてのコンポーネントは 1 回だけ発生します。割り当て済みのメモリー構造体は再使用され、集約関数で評価済みの結果をローカルキャッシュに入れるなどの実行時最適化によって、結果が変わる可能性もあります。したがって、`BENCHMARK()` を使用して、実行時コンポーネントに高い重みを付加し、ネットワーク、パーサー、オプティマイザなどで導入された「ノイズ」を削除することで、そのコンポーネントのパフォーマンスが測定されます。

CHARSET(str)

文字列引数の文字セットを返します。

```
mysql> SELECT CHARSET('abc');
-> 'utf8'
mysql> SELECT CHARSET(CONVERT('abc' USING latin1));
-> 'latin1'
mysql> SELECT CHARSET(USER());
-> 'utf8'
```

COERCIBILITY(str)

文字列引数の照合順序強制性値を返します。

```
mysql> SELECT COERCIBILITY('abc' COLLATE utf8_swedish_ci);
-> 0
mysql> SELECT COERCIBILITY(USER());
-> 3
mysql> SELECT COERCIBILITY('abc');
-> 4
mysql> SELECT COERCIBILITY(1000);
-> 5
```

戻り値の意味は、次の表に示すとおりです。値が低いほど、優先順位は高くなります。

型変換属性	意味	例
0	明示的な照合順序	<code>COLLATE</code> 句の値
1	照合順序なし	さまざまな照合順序との文字列の連結
2	暗黙的な照合順序	列値、ストアルーチンパラメータ、またはローカル変数
3	系統定数	<code>USER()</code> の戻り値
4	型変換可能	リテラル文字列
5	数値	数値または時間値
5	無視可能	<code>NULL</code> または <code>NULL</code> から派生した式

詳細は、[セクション 10.8.4 「式での照合の強制性」](#) を参照してください。

COLLATION(str)

文字列引数の照合順序を返します。

```
mysql> SELECT COLLATION('abc');
-> 'utf8_general_ci'
mysql> SELECT COLLATION(_utf8mb4'abc');
```

```
-> 'utf8mb4_0900_ai_ci'
mysql> SELECT COLLATION(_latin1'abc');
-> 'latin1_swedish_ci'
```

• CONNECTION_ID()

接続用の接続 ID (スレッド ID) を返します。すべての接続は、現在接続されているクライアントのセット間で一意の ID を持っています。

`CONNECTION_ID()` で返される値の型は、`INFORMATION_SCHEMA.PROCESSLIST` テーブルの `ID` カラム、`SHOW PROCESSLIST` 出力の `Id` カラム、およびパフォーマンススキーマ `threads` テーブルの `PROCESSLIST_ID` カラムに表示される値と同じです。

```
mysql> SELECT CONNECTION_ID();
-> 23786
```

警告

`pseudo_thread_id` システム変数のセッション値を変更すると、`CONNECTION_ID()` 関数によって返される値が変更されます。

• CURRENT_ROLE()

現在のセッションの現在アクティブなロールをカンマで区切って含む `utf8` 文字列を返します。存在しない場合は `NONE` を返します。この値は、`sql_quote_show_create` システム変数の設定を反映します。

アカウントに次のようなロールが付与されているとします:

```
GRANT 'r1', 'r2' TO 'u1'@'localhost';
SET DEFAULT ROLE ALL TO 'u1'@'localhost';
```

`u1` のセッションでは、`CURRENT_ROLE()` の初期値によってデフォルトのアカウントロールが指定されます。次のような `SET ROLE` の変更を使用:

```
mysql> SELECT CURRENT_ROLE();
+-----+
| CURRENT_ROLE() |
+-----+
| 'r1'@'%' , 'r2'@'%' |
+-----+
mysql> SET ROLE 'r1'; SELECT CURRENT_ROLE();
+-----+
| CURRENT_ROLE() |
+-----+
| 'r1'@'%' |
+-----+
```

• CURRENT_USER, CURRENT_USER()

現在のクライアントを認証する際にサーバーで使用された MySQL アカウントを表すユーザー名とホスト名の組み合わせを返します。このアカウントで、アクセス権限が決まります。戻り値は、`utf8` 文字セット内の文字列です。

`CURRENT_USER()` の値は、`USER()` の値とは異なる可能性があります。

```
mysql> SELECT USER();
-> 'davida@localhost'
mysql> SELECT * FROM mysql.user;
ERROR 1044: Access denied for user ''@'localhost' to
database 'mysql'
mysql> SELECT CURRENT_USER();
-> '@localhost'
```

この例は、クライアントが `davida` のユーザー名を指定 (`USER()` 関数の値で指定されます) したが、サーバーは匿名のユーザーアカウント (`CURRENT_USER()` 値の空のユーザー名部分に表示されます) を使用してクライアントを認

証したことを示しています。これが発生する原因として、`dauida` の付与テーブルにアカウントが一覧表示されていないことが考えられます。

ストアプログラムまたはビューでは、`SQL SECURITY INVOKER` 特性で定義されていなければ、`CURRENT_USER()` はオブジェクトを定義したユーザー (その `DEFINER` 値で指定されます) のアカウントを返します。後者の場合、`CURRENT_USER()` はオブジェクトを呼び出したユーザーを返します。

トリガーおよびイベントには、`SQL SECURITY` 特性を定義するためのオプションがありません。したがって、このようなオブジェクトの場合、`CURRENT_USER()` はオブジェクトを定義したユーザーのアカウントを返します。呼び出したユーザーを返すには、`USER()` または `SESSION_USER()` を使用します。

次のステートメントでは、影響を受けるユーザーや定義したユーザーの名前 (ホストの可能性もあります) の代わりに、`CURRENT_USER()` 関数を使用することがサポートされています。このような場合、必要に応じて `CURRENT_USER()` が拡張されます。

- `DROP USER`
- `RENAME USER`
- `GRANT`
- `REVOKE`
- `CREATE FUNCTION`
- `CREATE PROCEDURE`
- `CREATE TRIGGER`
- `CREATE EVENT`
- `CREATE VIEW`
- `ALTER EVENT`
- `ALTER VIEW`
- `SET PASSWORD`

`CURRENT_USER()` のこの拡張によるレプリケーションへの影響については、[セクション 17.5.1.8 「CURRENT_USER\(\) のレプリケーション」](#) を参照してください。

- `DATABASE()`

デフォルト (現在) のデータベース名を `utf8` 文字セット内の文字列として返します。デフォルトのデータベースがない場合は、`DATABASE()` は `NULL` を返します。ストアルーチン内では、デフォルトのデータベースはルーチンが関連付けられたデータベースですが、これは呼び出し元のコンテキストでのデフォルトのデータベースと同じであるとはかぎりません。

```
mysql> SELECT DATABASE();
-> 'test'
```

デフォルトのデータベースがない場合は、`DATABASE()` は `NULL` を返します。

- `FOUND_ROWS()`

注記

`SQL_CALC_FOUND_ROWS` クエリ修飾子および付随する `FOUND_ROWS()` 関数は、MySQL 8.0.17 で非推奨になりました。これらは、MySQL の将来のバージョンで削除される予定です。かわりに、`LIMIT` を使用してクエリを実行してから、追加の行がある

かどうかを判断するために `COUNT(*)` および `LIMIT` を使用しない別のクエリーを検討してください。たとえば、次のクエリーのかわりに:

```
SELECT SQL_CALC_FOUND_ROWS * FROM tbl_name WHERE id > 100 LIMIT 10;  
SELECT FOUND_ROWS();
```

かわりに次のクエリーを使用します:

```
SELECT * FROM tbl_name WHERE id > 100 LIMIT 10;  
SELECT COUNT(*) FROM tbl_name WHERE id > 100;
```

`COUNT(*)` は、特定の最適化の対象となります。 `SQL_CALC_FOUND_ROWS` では、一部の最適化が無効になります。

サーバーからクライアントに返される行数を制限するために、`SELECT` ステートメントに `LIMIT` 句が含まれている場合があります。場合によっては、ステートメントを再度実行せずに、`LIMIT` を付けなかった場合にステートメントで返されるはずの行数を知っておくことが望ましいことがあります。この行数を取得するには、`SELECT` ステートメントに `SQL_CALC_FOUND_ROWS` オプションを含めてから、`FOUND_ROWS()` を起動します:

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM tbl_name  
-> WHERE id > 100 LIMIT 10;  
mysql> SELECT FOUND_ROWS();
```

2 番目の `SELECT` は、1 番目の `SELECT` を `LIMIT` 句なしで記述した場合に返される行数を示す数字を返します。

最近成功した `SELECT` ステートメントに `SQL_CALC_FOUND_ROWS` オプションを付けなければ、`FOUND_ROWS()` は、そのステートメントで返された結果セットの行数を返します。ステートメントに `LIMIT` 句が含まれている場合、`FOUND_ROWS()` はその制限値以下の行数を返します。たとえば、ステートメントに `LIMIT 10` または `LIMIT 50, 10` が含まれている場合、`FOUND_ROWS()` はそれぞれ 10 と 60 を返します。

`FOUND_ROWS()` から取得できる行数は一時的なもので、`SELECT SQL_CALC_FOUND_ROWS` ステートメントのあとに、このステートメントを発行しても取得できるようには設計されていません。あとで値を参照する必要がある場合は、保存してください。

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM ... ;  
mysql> SET @rows = FOUND_ROWS();
```

`SELECT SQL_CALC_FOUND_ROWS` を使用している場合は、MySQL では完全な結果セット内の行数を計算する必要があります。ただし、結果セットはクライアントに送信される必要がないため、`LIMIT` なしでクエリーを再度実行するよりも速くなります。

`SQL_CALC_FOUND_ROWS` および `FOUND_ROWS()` は、クエリーで返される行数を制限するが、クエリーを再度実行しないで完全な結果セット内の行数を確認する必要がある状況でも役立ちます。例として、検索結果のほかのセクションを表示するページへのリンクを含むページが表示される Web スクリプトがあります。`FOUND_ROWS()` を使用すると、残りの結果を表示するために必要なその他のページ数を確認できます。

`SQL_CALC_FOUND_ROWS` および `FOUND_ROWS()` を使用すると、`UNION` の複数箇所で `LIMIT` が発生する可能性があるため、単純な `SELECT` ステートメントよりも、`UNION` ステートメントで使った方が複雑になります。

これは、`UNION` 内の個々の `SELECT` ステートメントに適用される場合と、`UNION` の結果全体にグローバルに適用される場合があります。

`UNION` で `SQL_CALC_FOUND_ROWS` を使用する目的は、グローバルな `LIMIT` なしで返される行数を返すことです。`UNION` で `SQL_CALC_FOUND_ROWS` を使用する条件は、次のとおりです。

- `UNION` の 1 番目の `SELECT` に、`SQL_CALC_FOUND_ROWS` キーワードが表示される必要があります。
- `FOUND_ROWS()` の値は、`UNION ALL` が使用されている場合にのみ正確です。`ALL` なしで `UNION` が使用される場合は、重複の削除が発生し、`FOUND_ROWS()` の値が単なる近似値になります。
- `UNION` で `LIMIT` が表示されない場合は、`SQL_CALC_FOUND_ROWS` が無視され、`UNION` を処理するために作成された一時テーブル内の行数が返されます。

ここで説明した以外のケースでは、`FOUND_ROWS()` の動作 (エラーが発生して `SELECT` ステートメントに失敗したあとの値など) が定義されません。

重要

ステートメントベースのレプリケーションでは、確実に `FOUND_ROWS()` をレプリケートすることはできません。行ベースのレプリケーションを使用すると、この関数は自動的にレプリケートされます。

- `ICU_VERSION()`

正規表現操作のサポートに使用される International Components for Unicode (ICU) ライブラリのバージョン ([セクション 12.8.2 「正規表現」](#) を参照)。この関数は、主にテストケースでの使用を目的としています。

- `LAST_INSERT_ID()`, `LAST_INSERT_ID(expr)`

引数を指定しない場合、`LAST_INSERT_ID()` は、最後に実行された `INSERT` ステートメントの結果として `AUTO_INCREMENT` カラムに正常に挿入された最初の自動生成値を表す `BIGINT UNSIGNED` (64-bit) 値を返します。正常に挿入された行がない場合は、`LAST_INSERT_ID()` の値は未変更のままです。

引数を指定すると、`LAST_INSERT_ID()` は符号なし整数を返します。

たとえば、`AUTO_INCREMENT` 値を生成する行を挿入したあとは、次のようにして値を取得できます。

```
mysql> SELECT LAST_INSERT_ID();
-> 195
```

現在実行中のステートメントによって、`LAST_INSERT_ID()` の値は影響を受けません。1 つのステートメントで `AUTO_INCREMENT` 値を生成してから、独自の `AUTO_INCREMENT` カラムを含むテーブルに行を挿入する複数行の `INSERT` ステートメントで `LAST_INSERT_ID()` を参照すると仮定します。2 番目のステートメントでは、`LAST_INSERT_ID()` の値は安定したままです。2 番目以降の行の値は、前の行の挿入の影響を受けません。(`LAST_INSERT_ID()` と `LAST_INSERT_ID(expr)` への参照を混在させる場合、影響は定義されていないことに注意してください。)

以前のステートメントでエラーが返された場合は、`LAST_INSERT_ID()` の値が定義されません。トランザクションテーブルでは、エラーによってステートメントがロールバックされる場合、`LAST_INSERT_ID()` の値は未定義のままです。手動の `ROLLBACK` では、`LAST_INSERT_ID()` の値はトランザクション前の値にリストアされず、`ROLLBACK` 時点と同じままです。

ストアルーチン (プロシージャや関数) またはトリガーの本文内では、`LAST_INSERT_ID()` の値は、このような種類のオブジェクトの本文外で実行されたステートメントと同様に変更されます。あとに続くステートメントで参

照される `LAST_INSERT_ID()` の値でのストアルーチンまたはトリガーの効果は、ルーチンの種類によって異なります。

- ストアドプロシージャで `LAST_INSERT_ID()` の値を変更するステートメントが実行される場合は、プロシージャ呼び出しが続くステートメントで変更された値が参照されます。
- 値を変更するストアドファンクションおよびトリガーの場合、関数またはトリガーの終了時に値がリストアされるため、後に続くステートメントには変更された値が表示されません。

生成された ID は、接続ごとにサーバー内に保持されます。つまり、関数によって指定されたクライアントに返された値は、そのクライアントによって `AUTO_INCREMENT` カラムに影響を与える最近のステートメント用に最初に生成された `AUTO_INCREMENT` 値です。この値は、ほかのクライアントが独自の `AUTO_INCREMENT` 値を生成した場合でも影響を受ける可能性はありません。この動作によって、各クライアントはほかのクライアントのアクティビティを気にすることなく、ロックやトランザクションを実行しないで独自の ID を取得できます。

行の `AUTO_INCREMENT` カラムを非「マジック」値（つまり、`NULL` でも `0` でもない値）に設定する場合は、`LAST_INSERT_ID()` の値が変更されません。

重要

単一の `INSERT` ステートメントを使用して複数の行を挿入する場合、`LAST_INSERT_ID()` は、最初に挿入された行のみに対して生成された値を返します。この理由は、ほかの一部のサーバーに対して同じ `INSERT` ステートメントを簡単に再現できるようにするためです。

例:

```
mysql> USE test;

mysql> CREATE TABLE t (
  id INT AUTO_INCREMENT NOT NULL PRIMARY KEY,
  name VARCHAR(10) NOT NULL
);

mysql> INSERT INTO t VALUES (NULL, 'Bob');

mysql> SELECT * FROM t;
+----+-----+
| id | name |
+----+-----+
| 1 | Bob |
+----+-----+

mysql> SELECT LAST_INSERT_ID();
+-----+
| LAST_INSERT_ID() |
+-----+
|          1 |
+-----+

mysql> INSERT INTO t VALUES
(NULL, 'Mary'), (NULL, 'Jane'), (NULL, 'Lisa');

mysql> SELECT * FROM t;
+----+-----+
| id | name |
+----+-----+
| 1 | Bob |
| 2 | Mary |
| 3 | Jane |
| 4 | Lisa |
+----+-----+

mysql> SELECT LAST_INSERT_ID();
+-----+
| LAST_INSERT_ID() |
+-----+
|          2 |
+-----+
```

2 番目の `INSERT` ステートメントで 3 つの新しい行が `t` に挿入されましたが、これらの行の 1 番目に生成された ID は 2 であり、あとに続く `SELECT` ステートメントでも、この値が `LAST_INSERT_ID()` によって返されます。

`INSERT IGNORE` を使用し、その行が無視された場合、`LAST_INSERT_ID()` は現在の値から未変更のままです (接続で正常な `INSERT` が実行されていない場合は、0 が返されます)。トランザクショナル以外のテーブルでは、`AUTO_INCREMENT` カウンタが増分されません。InnoDB テーブルでは、`innodb_autoinc_lock_mode` が 1 または 2 に設定されている場合は、次の例で示すように `AUTO_INCREMENT` が増分されます。

```
mysql> USE test;

mysql> SELECT @@innodb_autoinc_lock_mode;
+-----+
| @@innodb_autoinc_lock_mode |
+-----+
| 1 |
+-----+

mysql> CREATE TABLE `t` (
  `id` INT(11) NOT NULL AUTO_INCREMENT,
  `val` INT(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `i1` (`val`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

# Insert two rows

mysql> INSERT INTO t (val) VALUES (1),(2);

# With auto_increment_offset=1, the inserted rows
# result in an AUTO_INCREMENT value of 3

mysql> SHOW CREATE TABLE t\G
***** 1. row *****
Table: t
Create Table: CREATE TABLE `t` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `val` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `i1` (`val`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=latin1

# LAST_INSERT_ID() returns the first automatically generated
# value that is successfully inserted for the AUTO_INCREMENT column

mysql> SELECT LAST_INSERT_ID();
+-----+
| LAST_INSERT_ID() |
+-----+
| 1 |
+-----+

# The attempted insertion of duplicate rows fail but errors are ignored

mysql> INSERT IGNORE INTO t (val) VALUES (1),(2);
Query OK, 0 rows affected (0.00 sec)
Records: 2 Duplicates: 2 Warnings: 0

# With innodb_autoinc_lock_mode=1, the AUTO_INCREMENT counter
# is incremented for the ignored rows

mysql> SHOW CREATE TABLE t\G
***** 1. row *****
Table: t
Create Table: CREATE TABLE `t` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `val` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `i1` (`val`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=latin1

# The LAST_INSERT_ID is unchanged because the previous insert was unsuccessful
```

```
mysql> SELECT LAST_INSERT_ID();
+-----+
| LAST_INSERT_ID() |
+-----+
|          1 |
+-----+
```

詳細は、[セクション15.6.1.6「InnoDBでのAUTO_INCREMENT処理」](#)を参照してください。

`expr` が `LAST_INSERT_ID()` の引数として指定されている場合、引数の値は関数によって返され、`LAST_INSERT_ID()` によって返される次の値として記憶されます。これを使用すると、シーケンスのシミュレーションを行うことができます。

1. シーケンスカウンタを保持するテーブルを作成し、それを初期化します。

```
mysql> CREATE TABLE sequence (id INT NOT NULL);
mysql> INSERT INTO sequence VALUES (0);
```

2. そのテーブルを使用して、次のようにシーケンス番号を生成します。

```
mysql> UPDATE sequence SET id=LAST_INSERT_ID(id+1);
mysql> SELECT LAST_INSERT_ID();
```

`UPDATE` ステートメントは、シーケンスカウンタを増分し、`LAST_INSERT_ID()` への次の呼び出しで更新された値が返されるようにします。`SELECT` ステートメントは、その値を取得します。`mysql_insert_id()` C API 関数を使用して、値を取得することもできます。[mysql_insert_id\(\)](#)を参照してください。

`LAST_INSERT_ID()` を呼び出さずに、シーケンスを生成できます。このように関数を使用する有用性は、ID 値が最後に自動的に生成された値として保持されることです。独自のシーケンス値を生成するほかのクライアントと互いに影響しあうことなく、複数のクライアントが `UPDATE` ステートメントを発行し、`SELECT` ステートメント (または `mysql_insert_id()`) で独自のシーケンス値を取得できるため、マルチユーザーでも安全です。

`mysql_insert_id()` は `INSERT` および `UPDATE` ステートメントの後にのみ更新されるため、`SELECT` や `SET` などの他の SQL ステートメントを実行した後は、C API 関数を使用して `LAST_INSERT_ID(expr)` の値を取得できません。

- [ROLES_GRAPHML\(\)](#)

メモリーロールサブグラフを表す GraphML 文書を含む `utf8` 文字列を返します。`<graphml>` 要素のコンテンツを表示するには、`ROLE_ADMIN` 権限 (または非推奨の `SUPER` 権限) が必要です。それ以外の場合、結果には空の要素のみが表示されます:

```
mysql> SELECT ROLES_GRAPHML();
+-----+
| ROLES_GRAPHML() |
+-----+
| <?xml version="1.0" encoding="UTF-8"?><graphml /> |
+-----+
```

- `ROW_COUNT()`

`ROW_COUNT()` は、次のように値を返します:

- DDL ステートメント: 0。これは、`CREATE TABLE` や `DROP TABLE` などのステートメントに適用されます。
- `SELECT` 以外の DML ステートメント: 影響を受ける行数です。これは、`UPDATE`、`INSERT` または `DELETE` (以前と同様) などのステートメントに適用されますが、`ALTER TABLE` や `LOAD DATA` などのステートメントにも適用されるようになりました。
- `SELECT`: ステートメントで結果セットが返される場合は -1、そうでない場合は「影響を受ける」行数。たとえば、`SELECT * FROM t1` の場合、`ROW_COUNT()` は -1 を返します。`SELECT * FROM t1 INTO OUTFILE 'file_name'` の場合、`ROW_COUNT()` はファイルに書き込まれた行の数を返します。
- `SIGNAL` ステートメント: 0。

`UPDATE` ステートメントの場合、デフォルトで影響を受けた行の値は実際に変更された行数です。`mysqld` への接続時に `CLIENT_FOUND_ROWS` フラグを `mysql_real_connect()` に指定した場合、影響を受けた行の値は「見つかった」、つまり `WHERE` 句に一致した行数です。

`REPLACE` ステートメントの場合、影響を受けた行の値は、新しい行が古い行に置き換わった場合 2 です。この場合、重複が削除されたあとに行が挿入されたためです。

`INSERT ... ON DUPLICATE KEY UPDATE` ステートメントの場合、行ごとの影響を受けた行の値は、その行が新しい行として挿入された場合は 1、既存の行が更新された場合は 2、既存の行がその現在の値に設定された場合は 0 です。`CLIENT_FOUND_ROWS` フラグを指定した場合、影響を受けた行の値は、既存の行がその現在の値に設定された場合は (0 ではなく) 1 になります。

`ROW_COUNT()` は、`mysql_affected_rows()` C API 関数から取得される値と同様で、ステートメントの実行後に `mysql` クライアントに表示される行数です。

```
mysql> INSERT INTO t VALUES(1),(2),(3);
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0
```

```
mysql> SELECT ROW_COUNT();
+-----+
| ROW_COUNT() |
+-----+
|          3 |
+-----+
1 row in set (0.00 sec)
```

```
mysql> DELETE FROM t WHERE i IN(1,2);
Query OK, 2 rows affected (0.00 sec)
```

```
mysql> SELECT ROW_COUNT();
+-----+
| ROW_COUNT() |
+-----+
|          2 |
+-----+
1 row in set (0.00 sec)
```

重要

ステートメントベースのレプリケーションでは、確実に `ROW_COUNT()` をレプリケートすることはできません。行ベースのレプリケーションを使用すると、この関数は自動的にレプリケートされます。

- `SCHEMA()`

この関数は `DATABASE()` のシノニムです。

- `SESSION_USER()`

`SESSION_USER()` は `USER()` のシノニムです。

- `SYSTEM_USER()`

`SYSTEM_USER()` は `USER()` のシノニムです。

注記

`SYSTEM_USER()` 関数は、`SYSTEM_USER` 権限とは異なります。前者は、現在の MySQL アカウント名を返します。後者は、システムユーザーと通常のユーザーアカウントカテゴリを区別します ([セクション6.2.11「アカウントカテゴリ」](#) を参照)。

- `USER()`

現在の MySQL ユーザー名とホスト名を `utf8` 文字セット内の文字列として返します。

```
mysql> SELECT USER();
-> 'davida@localhost'
```

この値は、サーバーへの接続時に指定したユーザー名および接続元のクライアントホストを示します。`CURRENT_USER()` の値とは異なる可能性があります。

- `VERSION()`

MySQL サーバーのバージョンを示す文字列を返します。この文字列では、`utf8` 文字セットが使用されます。値にはバージョン番号に加えて、サフィクスが付いている場合もあります。[セクション5.1.8「サーバーシステム変数」](#) 内の `version` システム変数の説明を参照してください。

この関数は、ステートメントベースのレプリケーションでは安全に使用できません。`binlog_format` が `STATEMENT` に設定されているときに、この関数を使用すると、警告のログが記録されます。

```
mysql> SELECT VERSION();
-> '8.0.29-standard'
```

12.17 空間分析関数

MySQL には、空間データに対してさまざまな操作を実行する関数が用意されています。これらの関数は、実行する操作の種類に従って、次のいくつかの主なカテゴリにグループ化できます。

- さまざまな形式 (WKT、WKB、内部) のジオメトリを作成する関数
- ジオメトリの形式を変換する関数
- ジオメトリの定性的または定量的なプロパティにアクセスする関数
- 2つのジオメトリ間の関係を記述する関数
- 既存のジオメトリから新しいジオメトリを作成する関数

空間データを使用するための MySQL のサポートに関する一般的なバックグラウンドについては、[セクション11.4「空間データ型」](#) を参照してください。

12.17.1 空間関数のリファレンス

次のテーブルに、各空間関数を示し、それぞれについて簡単に説明します。

表 12.21 「空間関数」

名前	説明	導入
<code>GeomCollection()</code>	ジオメトリからジオメトリコレクションを構築します	
<code>GeometryCollection()</code>	ジオメトリからジオメトリコレクションを構築します	
<code>LineString()</code>	Point 値から LineString を構築します	

名前	説明	導入
MBRContains()	あるジオメトリの MBR に、別のジオメトリの MBR が含まれているかどうか	
MBRCoveredBy()	ある MBR が別の MBR でカバーされているかどうか	
MBRCovers()	ある MBR が別の MBR をカバーしているかどうか	
MBRDisjoint()	2 つのジオメトリの MBR が切り離されているかどうか	
MBREquals()	2 つのジオメトリの MBR が等しいかどうか	
MBRIntersects()	2 つのジオメトリの MBR が交差しているかどうか	
MBROverlaps()	2 つのジオメトリの MBR がオーバーラップしているかどうか	
MBRTouches()	2 つのジオメトリの MBR が接しているかどうか	
MBRWithin()	あるジオメトリの MBR が、別のジオメトリの MBR の内部にあるかどうか	
MultiLineString()	LineString 値から MultiLineString を構築します	
MultiPoint()	Point 値から MultiPoint を構築します	
MultiPolygon()	Polygon 値から MultiPolygon を構築します	
Point()	座標から Point を構築します	
Polygon()	LineString 引数から Polygon を構築します	
ST_Area()	Polygon または MultiPolygon 領域を返します	
ST_AsBinary(), ST_AsWKB()	内部ジオメトリ形式から WKB に変換します	
ST_AsGeoJSON()	ジオメトリから GeoJSON オブジェクトを生成します	
ST_AsText(), ST_AsWKT()	内部ジオメトリ形式から WKT に変換します	
ST_Buffer()	ジオメトリから指定された距離内にある点のジオメトリを返します	
ST_Buffer_Strategy()	ST_Buffer() の作成戦略オプション	
ST_Centroid()	重心を Point として返します	
ST_Collect()	空間値をコレクションに集約	8.0.24
ST_Contains()	あるジオメトリに別のジオメトリが含まれているかどうか	
ST_ConvexHull()	ジオメトリの凸包を返します	
ST_Crosses()	あるジオメトリが別のジオメトリと交差しているかどうか	
ST_Difference()	2 つのジオメトリの点集合の差集合を返します	

名前	説明	導入
ST_Dimension()	ジオメトリの次元	
ST_Disjoint()	あるジオメトリが別のジオメトリから切り離されているかどうか	
ST_Distance()	あるジオメトリの別のジオメトリからの距離	
ST_Distance_Sphere()	2つのジオメトリ間の地球上の最小距離	
ST_EndPoint()	LineString の終点	
ST_Envelope()	ジオメトリの MBR を返します	
ST_Equals()	あるジオメトリが別のジオメトリに等しいかどうか	
ST_ExteriorRing()	Polygon の外側のリングを返します	
ST_FrechetDistance()	ジオメトリ間の離散フレシェ距離	8.0.23
ST_GeoHash()	geohash 値を生成します	
ST_GeomCollFromText(), ST_GeometryCollectionFromText(), ST_GeomCollFromTxt()	WKT からジオメトリコレクションを返します	
ST_GeomCollFromWKB(), ST_GeometryCollectionFromWKB()	WKB からジオメトリコレクションを返します	
ST_GeometryN()	ジオメトリコレクションから N 番目のジオメトリを返します	
ST_GeometryType()	ジオメトリ型の名前を返します	
ST_GeomFromGeoJSON()	GeoJSON オブジェクトからジオメトリを生成します	
ST_GeomFromText(), ST_GeometryFromText()	WKT からジオメトリを返します	
ST_GeomFromWKB(), ST_GeometryFromWKB()	WKB からジオメトリを返します	
ST_HausdorffDistance()	ジオメトリ間の離散ハスドルフ距離	8.0.23
ST_InteriorRingN()	Polygon の N 番目の内側のリングを返します	
ST_Intersection()	2つのジオメトリの点集合の積集合を返します	
ST_Intersects()	あるジオメトリが別のジオメトリと交差しているかどうか	
ST_IsClosed()	ジオメトリが閉じていて単純かどうか	
ST_IsEmpty()	ジオメトリが空かどうか	
ST_IsSimple()	ジオメトリが単純かどうか	
ST_IsValid()	ジオメトリが有効かどうか	
ST_LatFromGeoHash()	geohash 値から緯度を返します	
ST_Latitude()	点の緯度を返します	8.0.12
ST_Length()	LineString の長さを返します	
ST_LineFromText(), ST_LineStringFromText()	WKT から LineString を構築します	
ST_LineFromWKB(), ST_LineStringFromWKB()	WKB から LineString を構築します	

名前	説明	導入
ST_LineInterpolatePoint()	LineString に沿った特定のパーセンテージの Point	8.0.24
ST_LineInterpolatePoints()	LineString に沿って指定された割合を Point	8.0.24
ST_LongFromGeoHash()	geohash 値から経度を返します	
ST_Longitude()	点の経度を返します	8.0.12
ST_MakeEnvelope()	2 点周りの長方形	
ST_MLineFromText(), ST_MultiLineStringFromText()	WKT から MultiLineString を構築します	
ST_MLineFromWKB(), ST_MultiLineStringFromWKB()	WKB から MultiLineString を構築します	
ST_MPointFromText(), ST_MultiPointFromText()	WKT から MultiPoint を構築します	
ST_MPointFromWKB(), ST_MultiPointFromWKB()	WKB から MultiPoint を構築します	
ST_MPolyFromText(), ST_MultiPolygonFromText()	WKT から MultiPolygon を構築します	
ST_MPolyFromWKB(), ST_MultiPolygonFromWKB()	WKB から MultiPolygon を構築します	
ST_NumGeometries()	ジオメトリコレクション内のジオメトリ数を返します	
ST_NumInteriorRing(), ST_NumInteriorRings()	Polygon 内の内側のリング数を返します	
ST_NumPoints()	LineString 内の Point の数を返します	
ST_Overlaps()	あるジオメトリが別のジオメトリとオーバーラップしているかどうか	
ST_PointAtDistance()	LineString に沿った特定の距離の点	8.0.24
ST_PointFromGeoHash()	geohash 値を POINT 値に変換します	
ST_PointFromText()	WKT から Point を構築します	
ST_PointFromWKB()	WKB から Point を構築します	
ST_PointN()	LineString から N 番目の Point を返します	
ST_PolyFromText(), ST_PolygonFromText()	WKT から Polygon を構築します	
ST_PolyFromWKB(), ST_PolygonFromWKB()	WKB から Polygon を構築します	
ST_Simplify()	簡略化されたジオメトリを返す	
ST_SRID()	ジオメトリの空間参照システム ID を返します	
ST_StartPoint()	LineString の始点	
ST_SwapXY()	X/Y 座標が入れ替えられた引数を返します	
ST_SymDifference()	2 つのジオメトリの点集合の対称差を返します	
ST_Touches()	あるジオメトリが別のジオメトリに接しているかどうか	

名前	説明	導入
ST_Transform()	ジオメトリの座標を変換	8.0.13
ST_Union()	2つのジオメトリの点集合の和集合を返します	
ST_Validate()	検証済ジオメトリを返します	
ST_Within()	あるジオメトリが別のジオメトリの内部にあるかどうか	
ST_X()	PointのX座標を返します	
ST_Y()	PointのY座標を返します	

12.17.2 空間関数による引数処理

空間値 (ジオメトリ) には、[セクション11.4.2.2「Geometry クラス」](#) で説明されているプロパティがあります。次の説明では、空間関数の一般的な引数処理特性を一覧表示します。特定の関数または関数のグループは、これらの関数の説明が発生するセクションで説明されているように、追加または異なる引数処理特性を持つことができます。true の場合、これらの説明はここでの一般的な説明よりも優先されます。

空間関数は、有効なジオメトリ値に対してのみ定義されています。[セクション11.4.4「ジオメトリの整形形式と妥当性」](#) を参照してください。

各ジオメトリ値は、地理的位置の座標ベースのシステムである空間参照システム (SRS) に関連付けられています。[セクション11.4.5「空間参照システムのサポート」](#) を参照してください。

ジオメトリの空間参照識別子 (SRID) は、ジオメトリが定義されている SRS を識別します。MySQL の SRID 値は、ジオメトリ値に関連付けられた整数です。使用可能な SRID の最大値は $2^{32}-1$ です。より大きな値が指定されると、低位の 32 ビットだけが使用されます。

SRID 0 は、軸に単位が割り当てられていない無限平坦なデカルト平面を表します。SRID 0 の動作を保証するには、SRID 0 を使用してジオメトリ値を作成します。SRID 0 は、SRID が指定されていない場合の新しいジオメトリ値のデフォルトです。

複数のジオメトリ値の計算では、すべての値が同じ SRS 内にある必要があり、そうでない場合はエラーが発生します。したがって、複数のジオメトリ引数を取る空間関数では、それらの引数が同じ SRS 内にある必要があります。空間関数が `ER_GIS_DIFFERENT_SRIDS` を戻す場合、ジオメトリ引数がすべて同じ SRS 内にあったわけではありません。SRS が同じになるように修正する必要があります。

空間関数によって生成されたジオメトリ値はジオメトリ引数の SRID を継承するため、空間関数によって戻されたジオメトリはジオメトリ引数の SRS と同一のものになります。

[Open Geospatial Consortium](#) ガイドラインでは、入力ポリゴンがすでに閉じられている必要があるため、閉じられていないポリゴンは閉じられるのではなく、無効として拒否されます。

MySQL では、有効な空のジオメトリのみが空のジオメトリコレクションの形式で表されます。空のジオメトリコレクションの処理は次のとおりです: 空の WKT 入力ジオメトリコレクションを `'GEOMETRYCOLLECTION()'` として指定できます。これは、空のジオメトリコレクションを生成する空間操作による出力 WKT でもあります。

ネストされたジオメトリコレクションの解析中に、コレクションがフラット化され、その基本コンポーネントが様々な GIS 操作で使用されて結果が計算されます。これにより、ジオメトリデータの一意性を考慮する必要がなくなるため、ユーザーに柔軟性が向上します。ネストされたジオメトリコレクションは、最初に明示的にフラット化しなくても、ネストされた GIS 関数コールから生成できます。

12.17.3 WKT 値からジオメトリ値を作成する関数

これらの関数は、引数として Well-Known Text (WKT) 表現と、オプションで空間参照システム識別子 (SRID) を受け取ります。これらは、対応するジオメトリを返します。WKT 形式については、[WKT \(Well-Known Text\) 形式](#) を参照してください。

このセクションの関数は、デカルトまたは地理空間参照システム (SRS) の引数を検出し、SRS に適した結果を返します。

`ST_GeomFromText()` は、ジオメトリタイプの WKT 値を最初の引数として受け入れます。その他の関数は、各ジオメトリ型のジオメトリ値の構築のための型固有の構築関数を提供します。

`MultiPoint` 値の WKT 形式表現を受け入れる `ST_MPointFromText()` や `ST_GeomFromText()` などの関数では、値内の個々のポイントをカッコで囲むことができます。たとえば、次の関数コールは両方とも有効です:

```
ST_MPointFromText('MULTIPOINT (1 1, 2 2, 3 3)')
ST_MPointFromText('MULTIPOINT ((1 1), (2 2), (3 3))')
```

WKT ジオメトリコレクション引数を受け入れる `ST_GeomFromText()` などの関数は、OpenGIS 'GEOMETRYCOLLECTION EMPTY' 標準構文と MySQL 'GEOMETRYCOLLECTION()' 非標準構文の両方を理解します。WKT 値を生成する `ST_AsWKT()` などの関数は、'GEOMETRYCOLLECTION EMPTY' 標準構文を生成します:

```
mysql> SET @s1 = ST_GeomFromText('GEOMETRYCOLLECTION()');
mysql> SET @s2 = ST_GeomFromText('GEOMETRYCOLLECTION EMPTY');
mysql> SELECT ST_AsWKT(@s1), ST_AsWKT(@s2);
+-----+-----+
| ST_AsWKT(@s1) | ST_AsWKT(@s2) |
+-----+-----+
| GEOMETRYCOLLECTION EMPTY | GEOMETRYCOLLECTION EMPTY |
+-----+-----+
```

特に指定がないかぎり、このセクションの関数はジオメトリ引数を次のように処理します:

- ジオメトリ引数が `NULL` の場合、構文的に整形形式のジオメトリでない場合、または SRID 引数が `NULL` の場合、戻り値は `NULL` です。
- デフォルトでは、地理座標 (緯度、経度) は、ジオメトリ引数の空間参照システムで指定された順序で解釈されます。オプションの `options` 引数を指定して、デフォルトの軸の順序をオーバーライドできます。`options` は、カンマ区切りの `key=value` のリストで構成されます。許可されている `key` 値は `axis-order` のみで、`lat-long`、`long-lat` および `srid-defined` (デフォルト) の値が許可されています。

`options` 引数が `NULL` の場合、戻り値は `NULL` です。`options` 引数が無効な場合は、理由を示すエラーが発生します。

- SRID 引数が未定義の空間参照システム (SRS) を参照すると、`ER_SRS_NOT_FOUND` エラーが発生します。
- 地理 SRS ジオメトリ引数で、範囲外の経度または緯度を持つ引数がある場合、エラーが発生します:
 - 経度の値が `(-180, 180]` の範囲にない場合は、`ER_LONGITUDE_OUT_OF_RANGE` エラーが発生します。
 - 緯度の値が `[-90, 90]` の範囲にない場合は、`ER_LATITUDE_OUT_OF_RANGE` エラーが発生します。

表示される範囲は度数です。SRS で別の単位が使用されている場合、範囲ではその単位に対応する値が使用されます。浮動小数点演算のため、正確な範囲制限はわずかに偏差します。

WKT 値からジオメトリを作成するには、次の関数を使用できます:

- `ST_GeomCollFromText(wkt [, srid [, options]])`, `ST_GeometryCollectionFromText(wkt [, srid [, options]])`, `ST_GeomCollFromText(wkt [, srid [, options]])`

WKT 表現と SRID を使用して `GeometryCollection` 値を構築します。

これらの関数は、このセクションの概要で説明されているように、引数を処理します。

```
mysql> SET @g = "MULTILINESTRING((10 10, 11 11), (9 9, 10 10))";
mysql> SELECT ST_AsText(ST_GeomCollFromText(@g));
+-----+
| ST_AsText(ST_GeomCollFromText(@g)) |
+-----+
| MULTILINESTRING((10 10,11 11),(9 9,10 10)) |
+-----+
```

- `ST_GeomFromText(wkt [, srid [, options]])`, `ST_GeometryFromText(wkt [, srid [, options]])`

WKT 表現と SRID を使用して任意の型のジオメトリ値を構築します。

これらの関数は、このセクションの概要で説明されているように、引数を処理します。

- `ST_LineFromText(wkt [, srid [, options]])`, `ST_LineStringFromText(wkt [, srid [, options]])`

WKT 表現と SRID を使用して `LineString` 値を構築します。

これらの関数は、このセクションの概要で説明されているように、引数を処理します。

- `ST_MLineFromText(wkt [, srid [, options]])`, `ST_MultiLineStringFromText(wkt [, srid [, options]])`

WKT 表現と SRID を使用して `MultiLineString` 値を構築します。

これらの関数は、このセクションの概要で説明されているように、引数を処理します。

- `ST_MPointFromText(wkt [, srid [, options]])`, `ST_MultiPointFromText(wkt [, srid [, options]])`

WKT 表現と SRID を使用して `MultiPoint` 値を構築します。

これらの関数は、このセクションの概要で説明されているように、引数を処理します。

- `ST_MPolyFromText(wkt [, srid [, options]])`, `ST_MultiPolygonFromText(wkt [, srid [, options]])`

WKT 表現と SRID を使用して `MultiPolygon` 値を構築します。

これらの関数は、このセクションの概要で説明されているように、引数を処理します。

- `ST_PointFromText(wkt [, srid [, options]])`

WKT 表現と SRID を使用して `Point` 値を構築します。

`ST_PointFromText()` は、このセクションの概要で説明されているように引数を処理します。

- `ST_PolyFromText(wkt [, srid [, options]])`, `ST_PolygonFromText(wkt [, srid [, options]])`

WKT 表現と SRID を使用して `Polygon` 値を構築します。

これらの関数は、このセクションの概要で説明されているように、引数を処理します。

12.17.4 WKB 値からジオメトリ値を作成する関数

これらの関数は、引数として Well-Known Binary (WKB) 表現を含む `BLOB` と、オプションで空間参照システム識別子 (SRID) を受け取ります。これらは、対応するジオメトリを返します。WKB 形式については、[WKB \(Well-Known Binary\) 形式](#) を参照してください。

このセクションの関数は、デカルトまたは地理空間参照システム (SRS) の引数を検出し、SRS に適した結果を返します。

`ST_GeomFromWKB()` は、ジオメトリタイプの WKB 値を最初の引数として受け入れます。その他の関数は、各ジオメトリ型のジオメトリ値の構築のための型固有の構築関数を提供します。

MySQL 8.0 より前は、これらの関数は [セクション12.17.5「ジオメトリ値を作成する MySQL 固有の関数」](#) の関数によって戻されたジオメトリオブジェクトも受け入れていました。ジオメトリ引数は許可されなくなり、エラーが発生します。ジオメトリ引数の使用から WKB 引数の使用にコールを移行するには、次のガイドラインに従います:

- `ST_GeomFromWKB(Point(0, 0))` などの構成を `Point(0, 0)` として書き換えます。
- `ST_SRID(Point(0, 0), 4326)` や `ST_GeomFromWKB(ST_AsWKB(Point(0, 0)), 4326)` などの `ST_GeomFromWKB(Point(0, 0), 4326)` の構成を書き換えます。

特に指定がない限り、このセクションの関数はジオメトリ引数を次のように処理します:

- WKB または SRID 引数が `NULL` の場合、戻り値は `NULL` です。
- デフォルトでは、地理座標 (緯度、経度) は、ジオメトリ引数の空間参照システムで指定された順序で解釈されます。オプションの `options` 引数を指定して、デフォルトの軸の順序をオーバーライドできます。`options` は、カンマ

区切りの `key=value` のリストで構成されます。許可されている `key` 値は `axis-order` のみで、`lat-long`、`long-lat` および `srid-defined` (デフォルト) の値が許可されています。

`options` 引数が `NULL` の場合、戻り値は `NULL` です。`options` 引数が無効な場合は、理由を示すエラーが発生します。

- SRID 引数が未定義の空間参照システム (SRS) を参照すると、`ER_SRS_NOT_FOUND` エラーが発生します。
- 地理 SRS ジオメトリ引数で、範囲外の経度または緯度を持つ引数がある場合、エラーが発生します:
 - 経度の値が `(-180, 180]` の範囲にない場合は、`ER_LONGITUDE_OUT_OF_RANGE` エラーが発生します。
 - 緯度の値が `[-90, 90]` の範囲にない場合は、`ER_LATITUDE_OUT_OF_RANGE` エラーが発生します。

表示される範囲は度数です。SRS で別の単位が使用されている場合、範囲ではその単位に対応する値が使用されます。浮動小数点演算のため、正確な範囲制限はわずかに偏差します。

WKB 値からジオメトリを作成するには、次の関数を使用できます:

- `ST_GeomCollFromWKB(wkb [, srid [, options]])`, `ST_GeometryCollectionFromWKB(wkb [, srid [, options]])`

WKB 表現と SRID を使用して `GeometryCollection` 値を構築します。

これらの関数は、このセクションの概要で説明されているように、引数を処理します。

- `ST_GeomFromWKB(wkb [, srid [, options]])`, `ST_GeometryFromWKB(wkb [, srid [, options]])`

WKB 表現と SRID を使用して任意の型のジオメトリ値を構築します。

これらの関数は、このセクションの概要で説明されているように、引数を処理します。

- `ST_LineFromWKB(wkb [, srid [, options]])`, `ST_LineStringFromWKB(wkb [, srid [, options]])`

WKB 表現と SRID を使用して `LineString` 値を構築します。

これらの関数は、このセクションの概要で説明されているように、引数を処理します。

- `ST_MLineFromWKB(wkb [, srid [, options]])`, `ST_MultiLineStringFromWKB(wkb [, srid [, options]])`

WKB 表現と SRID を使用して `MultiLineString` 値を構築します。

これらの関数は、このセクションの概要で説明されているように、引数を処理します。

- `ST_MPointFromWKB(wkb [, srid [, options]])`, `ST_MultiPointFromWKB(wkb [, srid [, options]])`

WKB 表現と SRID を使用して `MultiPoint` 値を構築します。

これらの関数は、このセクションの概要で説明されているように、引数を処理します。

- `ST_MPolyFromWKB(wkb [, srid [, options]])`, `ST_MultiPolygonFromWKB(wkb [, srid [, options]])`

WKB 表現と SRID を使用して `MultiPolygon` 値を構築します。

これらの関数は、このセクションの概要で説明されているように、引数を処理します。

- `ST_PointFromWKB(wkb [, srid [, options]])`

WKB 表現と SRID を使用して `Point` 値を構築します。

`ST_PointFromWKB()` は、このセクションの概要で説明されているように引数を処理します。

- `ST_PolyFromWKB(wkb [, srid [, options]])`, `ST_PolygonFromWKB(wkb [, srid [, options]])`

WKB 表現と SRID を使用して `Polygon` 値を構築します。

これらの関数は、このセクションの概要で説明されているように、引数を処理します。

12.17.5 ジオメトリ値を作成する MySQL 固有の関数

MySQL には、ジオメトリ値を作成するために役立つ一連の非標準関数が用意されています。このセクションで説明されている関数は、OpenGIS 仕様への MySQL 拡張です。

これらの関数は、引数としての WKB 値またはジオメトリオブジェクトからジオメトリオブジェクトを生成します。いずれかの引数が適切な WKB でも、適切なオブジェクト型のジオメトリ表現でもない場合、戻り値は `NULL` になります。

たとえば、`Point()` からのジオメトリの戻り値を `POINT` カラムに直接挿入できます。

```
INSERT INTO t1 (pt_col) VALUES(Point(1,2));
```

- `GeomCollection(g [, g] ...)`

ジオメトリ引数から `GeomCollection` 値を構築します。

`GeomCollection()` は、サポートされていないジオメトリが存在する場合でも、引数に含まれるすべての適切なジオメトリを戻します。

引数のない `GeomCollection()` は、空のジオメトリを作成する方法として許可されています。また、WKT ジオメトリコレクション引数を受け入れる `ST_GeomFromText()` などの関数は、OpenGIS 'GEOMETRYCOLLECTION EMPTY' 標準構文と MySQL 'GEOMETRYCOLLECTION()' 非標準構文の両方を理解します。

`GeomCollection()` と `GeometryCollection()` は同義ですが、`GeomCollection()` を使用することをお勧めします。

- `GeometryCollection(g [, g] ...)`

ジオメトリ引数から `GeomCollection` 値を構築します。

`GeometryCollection()` は、サポートされていないジオメトリが存在する場合でも、引数に含まれるすべての適切なジオメトリを戻します。

引数のない `GeometryCollection()` は、空のジオメトリを作成する方法として許可されています。また、WKT ジオメトリコレクション引数を受け入れる `ST_GeomFromText()` などの関数は、OpenGIS 'GEOMETRYCOLLECTION EMPTY' 標準構文と MySQL 'GEOMETRYCOLLECTION()' 非標準構文の両方を理解します。

`GeomCollection()` と `GeometryCollection()` は同義ですが、`GeomCollection()` を使用することをお勧めします。

- `LineString(pt [, pt] ...)`

複数の `Point` または WKB `Point` 引数から `LineString` 値を構築します。引数の数が 2 未満の場合、戻り値は `NULL` になります。

- `MultiLineString(ls [, ls] ...)`

`LineString` または WKB `LineString` 引数を使用して `MultiLineString` 値を構築します。

- `MultiPoint(pt [, pt2] ...)`

`Point` または WKB `Point` 引数を使用して `MultiPoint` 値を構築します。

- `MultiPolygon(poly [, poly] ...)`

一連の `Polygon` または WKB `Polygon` 引数から `MultiPolygon` 値を構築します。

- `Point(x, y)`

座標を使用して `Point` を構築します。

- `Polygon(ls [, ls] ...)`

複数の `LineString` または WKB `LineString` 引数から `Polygon` 値を構築します。いずれかの引数が `LinearRing` を表していない (つまり、閉じた単純な `LineString` でない) 場合、戻り値は `NULL` になります。

12.17.6 ジオメトリ形式変換関数

MySQL では、ジオメトリ値を内部ジオメトリフォーマットから WKT フォーマットまたは WKB フォーマットに変換したり、X 座標と Y 座標の順序を入れ替えるために、このセクションにリストされている関数がサポートされています。

文字列を WKT または WKB 形式から内部ジオメトリ形式に変換する関数もあります。 [セクション12.17.3 「WKT 値からジオメトリ値を作成する関数」](#) および [セクション12.17.4 「WKB 値からジオメトリ値を作成する関数」](#) を参照してください。

WKT ジオメトリコレクション引数を受け入れる `ST_GeomFromText()` などの関数は、OpenGIS 'GEOMETRYCOLLECTION EMPTY' 標準構文と MySQL 'GEOMETRYCOLLECTION()' 非標準構文の両方を理解します。空のジオメトリコレクションを生成する別の方法は、引数を指定せずに `GeometryCollection()` をコールすることです。WKT 値を生成する `ST_AsWKT()` などの関数は、'GEOMETRYCOLLECTION EMPTY' 標準構文を生成します:

```
mysql> SET @s1 = ST_GeomFromText('GEOMETRYCOLLECTION()');
mysql> SET @s2 = ST_GeomFromText('GEOMETRYCOLLECTION EMPTY');
mysql> SELECT ST_AsWKT(@s1), ST_AsWKT(@s2);
+-----+
| ST_AsWKT(@s1) | ST_AsWKT(@s2) |
+-----+
| GEOMETRYCOLLECTION EMPTY | GEOMETRYCOLLECTION EMPTY |
+-----+
mysql> SELECT ST_AsWKT(GeomCollection());
+-----+
| ST_AsWKT(GeomCollection()) |
+-----+
| GEOMETRYCOLLECTION EMPTY |
+-----+
```

特に指定がないかぎり、このセクションの関数はジオメトリ引数を次のように処理します:

- いずれかの引数が `NULL` の場合、戻り値は `NULL` です。
- ジオメトリ引数が構文的に整形形式のジオメトリでない場合は、`ER_GIS_INVALID_DATA` エラーが発生します。
- ジオメトリ引数が未定義の空間参照システムにある場合、軸はジオメトリに出現する順序で出力され、`ER_WARN_SRS_NOT_FOUND_AXIS_ORDER` 警告が発生します。
- デフォルトでは、地理座標 (緯度、経度) は、ジオメトリ引数の空間参照システムで指定された順序で解釈されます。オプションの `options` 引数を指定して、デフォルトの軸の順序をオーバーライドできます。`options` は、カンマ区切りの `key=value` のリストで構成されます。許可されている `key` 値は `axis-order` のみで、`lat-long`、`long-lat` および `srid-defined` (デフォルト) の値が許可されています。

`options` 引数が `NULL` の場合、戻り値は `NULL` です。`options` 引数が無効な場合は、理由を示すエラーが発生します。

- それ以外の場合、戻り値は `NULL` 以外です。

次の関数は、フォーマット変換または座標の入替えに使用できます:

- `ST_AsBinary(g [, options])`, `ST_AsWKB(g [, options])`

内部ジオメトリ形式の値を WKB 表現に変換し、そのバイナリの結果を返します。

関数の戻り値には、ジオメトリ引数に適用される空間参照システムで指定された順序で地理座標 (緯度、経度) が含まれます。オプションの `options` 引数を指定して、デフォルトの軸の順序をオーバーライドできます。

`ST_AsBinary()` および `ST_AsWKB()` は、このセクションの概要で説明されているように引数を処理します。

```
mysql> SET @g = ST_LineFromText('LINESTRING(0 5,5 10,10 15)', 4326);
mysql> SELECT ST_AsText(ST_GeomFromWKB(ST_AsWKB(@g)));
+-----+
| ST_AsText(ST_GeomFromWKB(ST_AsWKB(@g))) |
+-----+
| LINESTRING(5 0,10 5,15 10) |
+-----+
mysql> SELECT ST_AsText(ST_GeomFromWKB(ST_AsWKB(@g, 'axis-order=long-lat')));
```

```

+-----+
| ST_AsText(ST_GeomFromWKB(ST_AsWKB(@g, 'axis-order=long-lat'))) |
+-----+
| LINESTRING(0 5,5 10,10 15) |
+-----+
mysql> SELECT ST_AsText(ST_GeomFromWKB(ST_AsWKB(@g, 'axis-order=lat-long')));
+-----+
| ST_AsText(ST_GeomFromWKB(ST_AsWKB(@g, 'axis-order=lat-long'))) |
+-----+
| LINESTRING(5 0,10 5,15 10) |
+-----+

```

- `ST_AsText(g [, options])`, `ST_AsWKT(g [, options])`

内部ジオメトリ形式の値を WKT 表現に変換し、その文字列の結果を返します。

関数の戻り値には、ジオメトリ引数に適用される空間参照システムで指定された順序で地理座標 (緯度、経度) が含まれます。オプションの `options` 引数を指定して、デフォルトの軸の順序をオーバーライドできます。

`ST_AsText()` および `ST_AsWKT()` は、このセクションの概要で説明されているように引数を処理します。

```

mysql> SET @g = 'LineString(1 1,2 2,3 3)';
mysql> SELECT ST_AsText(ST_GeomFromText(@g));
+-----+
| ST_AsText(ST_GeomFromText(@g)) |
+-----+
| LINESTRING(1 1,2 2,3 3) |
+-----+

```

`MultiPoint` 値の出力では、各点の周囲にカッコが含まれます。例:

```

mysql> SELECT ST_AsText(ST_GeomFromText(@mp));
+-----+
| ST_AsText(ST_GeomFromText(@mp)) |
+-----+
| MULTIPOINT((1 1),(2 2),(3 3)) |
+-----+

```

- `ST_SwapXY(g)`

内部ジオメトリ形式の引数を受け入れ、ジオメトリ内の各座標ペアの X 値と Y 値を入れ替えて、結果を返します。

`ST_SwapXY()` は、このセクションの概要で説明されているように引数を処理します。

```

mysql> SET @g = ST_LineFromText('LINESTRING(0 5,5 10,10 15)');
mysql> SELECT ST_AsText(@g);
+-----+
| ST_AsText(@g) |
+-----+
| LINESTRING(0 5,5 10,10 15) |
+-----+
mysql> SELECT ST_AsText(ST_SwapXY(@g));
+-----+
| ST_AsText(ST_SwapXY(@g)) |
+-----+
| LINESTRING(5 0,10 5,15 10) |
+-----+

```

12.17.7 ジオメトリプロパティ関数

このグループに属する各関数は、引数としてジオメトリ値を受け取り、そのジオメトリの何らかの定量的または定性的なプロパティを返します。一部の関数では、その引数の型が制限されます。このような関数は、その引数のジオメトリ型が正しくない場合は `NULL` を返します。たとえば、オブジェクトタイプが `Polygon` でも `MultiPolygon` でもない場合、`ST_Area()` ポリゴン関数は `NULL` を返します。

12.17.7.1 一般的なジオメトリプロパティ関数

このセクションに示されている関数では引数が制限されず、任意の型のジオメトリ値が受け入れられます。

特に指定がないかぎり、このセクションの関数はジオメトリ引数を次のように処理します:

- いずれかの引数が `NULL` の場合、戻り値は `NULL` です。
- ジオメトリ引数が構文的に整形形式のジオメトリでない場合は、`ER_GIS_INVALID_DATA` エラーが発生します。
- 未定義の空間参照システム (SRS) でジオメトリ引数が構文的に整形形式のジオメトリである場合、`ER_SRS_NOT_FOUND` エラーが発生します。
- SRID 引数が 32 ビット符号なし整数の範囲内でない場合は、`ER_DATA_OUT_OF_RANGE` エラーが発生します。
- SRID 引数が未定義の SRS を参照すると、`ER_SRS_NOT_FOUND` エラーが発生します。
- それ以外の場合、戻り値は `NULL` 以外です。

ジオメトリプロパティを取得するには、次の関数を使用できます:

- `ST_Dimension(g)`

ジオメトリ値 `g` の固有の次元を返します。dimension は -1、0、1、または 2 の値を取ります。これらの値の意味は、[セクション11.4.2.2「Geometry クラス」](#) で指定されています。

`ST_Dimension()` は、このセクションの概要で説明されているように引数を処理します。

```
mysql> SELECT ST_Dimension(ST_GeomFromText('LineString(1 1,2 2)'));
+-----+
| ST_Dimension(ST_GeomFromText('LineString(1 1,2 2)')) |
+-----+
| 1 |
+-----+
```

- `ST_Envelope(g)`

ジオメトリ値 `g` の最小境界矩形 (MBR) を返します。結果は、その境界矩形の角の点によって定義される `Polygon` 値として返されます。

```
POLYGON((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
```

```
mysql> SELECT ST_AsText(ST_Envelope(ST_GeomFromText('LineString(1 1,2 2)')));
+-----+
| ST_AsText(ST_Envelope(ST_GeomFromText('LineString(1 1,2 2)')) |
+-----+
| POLYGON((1 1,2 2,1 2,1 1)) |
+-----+
```

引数が点、垂直または水平の線セグメントの場合、`ST_Envelope()` は無効なポリゴンを戻すのではなく、点または線セグメントを MBR として戻します:

```
mysql> SELECT ST_AsText(ST_Envelope(ST_GeomFromText('LineString(1 1,1 2)')));
+-----+
| ST_AsText(ST_Envelope(ST_GeomFromText('LineString(1 1,1 2)')) |
+-----+
| LINESTRING(1 1,1 2) |
+-----+
```

`ST_Envelope()` は、このセクションの概要で説明されているように引数を処理しますが、次の例外があります:

- ジオメトリに地理空間参照システム (SRS) の SRID 値がある場合、`ER_NOT_IMPLEMENTED_FOR_GEOGRAPHIC_SRS` エラーが発生します。
- `ST_GeometryType(g)`

ジオメトリインスタンス `g` がメンバーになっているジオメトリ型の名前を示すバイナリ文字列を返します。この名前は、インスタンス化可能な `Geometry` サブクラスのいずれかに対応します。

`ST_GeometryType()` は、このセクションの概要で説明されているように引数を処理します。

```
mysql> SELECT ST_GeometryType(ST_GeomFromText('POINT(1 1)'));
```



```

+-----+
| ST_GeometryType(ST_GeomFromText('POINT(1 1)')) |
+-----+
| POINT |
+-----+

```

- [ST_IsEmpty\(g\)](#)

この関数は、空のジオメトリコレクション値の場合は 1 を返し、それ以外の場合は 0 を返すプレースホルダです。

有効な空のジオメトリのみが、空のジオメトリコレクション値の形式で表されます。MySQL は、[POINT EMPTY](#) などの GIS の [EMPTY](#) 値をサポートしていません。

[ST_IsEmpty\(\)](#) は、このセクションの概要で説明されているように引数を処理します。

- [ST_IsSimple\(g\)](#)

ISO 「SQL/MM パート 3: 空間」標準に従ってジオメトリ値 [g](#) が単純な場合、1 を返します。引数が単純でない場合、[ST_IsSimple\(\)](#) は 0 を返します。

[セクション11.4.2「OpenGIS ジオメトリモデル」](#) で指定されるインスタンス化可能ジオメトリクラスの説明には、クラスインスタンスが単純ではないと分類される特定の条件が含まれます。

[ST_IsSimple\(\)](#) は、このセクションの概要で説明されているように引数を処理しますが、次の例外があります:

- ジオメトリに経度または緯度が範囲外の地理 SRS がある場合、エラーが発生します:
 - 経度の値が $[-180, 180]$ の範囲内でない場合は、[ER_GEOMETRY_PARAM_LONGITUDE_OUT_OF_RANGE](#) エラーが発生します (MySQL 8.0.12 より前の [ER_LONGITUDE_OUT_OF_RANGE](#))。
 - 緯度の値が $[-90, 90]$ の範囲内でない場合は、[ER_GEOMETRY_PARAM_LATITUDE_OUT_OF_RANGE](#) エラーが発生します (MySQL 8.0.12 より前の [ER_LATITUDE_OUT_OF_RANGE](#))。

表示される範囲は度数です。浮動小数点演算のため、正確な範囲制限はわずかに偏差します。

- [ST_SRID\(g \[, srid\]\)](#)

引数として有効なジオメトリオブジェクト [g](#) を 1 つだけ与えた場合、[ST_SRID\(\)](#) は、[g](#) に関連付けられた空間参照システム (SRS) の ID を示す整数を返します。

オプションである 2 番目の引数として有効な SRID 値を与えた場合、[ST_SRID\(\)](#) は、SRID 値が 2 番目の引数と等しい最初の引数と同じ型のオブジェクトを返します。これにより、オブジェクトの SRID 値のみが設定され、座標値の変換は実行されません。

[ST_SRID\(\)](#) は、このセクションの概要で説明されているように引数を処理しますが、次の例外があります:

- 単一引数構文の場合、[ST_SRID\(\)](#) は未定義の SRS を参照していてもジオメトリ SRID を返します。[ER_SRS_NOT_FOUND](#) エラーは発生しません。

[ST_SRID\(g, target_srid\)](#) と [ST_Transform\(g, target_srid\)](#) は、次の点で異なります:

- [ST_SRID\(\)](#) は、座標を変換せずにジオメトリ SRID 値を変更します。
- [ST_Transform\(\)](#) は、SRID 値の変更に加えてジオメトリ座標を変換します。

```

mysql> SET @g = ST_GeomFromText('LineString(1 1,2 2)', 0);
mysql> SELECT ST_SRID(@g);
+-----+
| ST_SRID(@g) |
+-----+
| 0 |
+-----+
mysql> SET @g = ST_SRID(@g, 4326);
mysql> SELECT ST_SRID(@g);
+-----+
| ST_SRID(@g) |
+-----+

```

```
| 4326 |
+-----+
```

空間値を作成するための MySQL 固有関数の結果を SRID 値とともに `ST_SRID()` に渡すことで、特定の SRID にジオメトリを作成できます。例:

```
SET @g1 = ST_SRID(Point(1, 1), 4326);
```

ただし、このメソッドは SRID 0 にジオメトリを作成し、SRID 4326 (WGS 84) にキャストします。最初に正しい空間参照システムを使用してジオメトリを作成することをお勧めします。例:

```
SET @g1 = ST_PointFromText('POINT(1 1)', 4326);
SET @g1 = ST_GeomFromText('POINT(1 1)', 4326);
```

`ST_SRID()` の 2 引数形式は、SRID が正しくないジオメトリの SRS の修正や変更などのタスクに役立ちます。

12.17.7.2 Pointプロパティ関数

`Point` は X 座標と Y 座標で構成され、それぞれ `ST_X()` 関数と `ST_Y()` 関数を使用して取得できます。これらの関数では、X または Y 座標値を指定するオプションの第 2 引数も許可されます。この場合、関数の結果は最初の引数の `Point` オブジェクトになり、X 座標または Y 座標が 2 番目の引数と等しくなるように変更されます。

地理空間参照システム (SRS) を持つ `Point` オブジェクトの場合、経度および緯度は、それぞれ `ST_Longitude()` 関数および `ST_Latitude()` 関数を使用して取得できます。これらの関数では、経度または緯度の値を指定するオプションの第 2 引数も許可されます。この場合、関数の結果は、経度または緯度に変更された最初の引数の `Point` オブジェクトとなり、2 番目の引数と等しくなります。

特に指定がない限り、このセクションの関数はジオメトリ引数を次のように処理します:

- いずれかの引数が `NULL` の場合、戻り値は `NULL` です。
- ジオメトリ引数が有効なジオメトリであるが、`Point` オブジェクトではない場合、`ER_UNEXPECTED_GEOMETRY_TYPE` エラーが発生します。
- ジオメトリ引数が構文的に整形形式のジオメトリでない場合は、`ER_GIS_INVALID_DATA` エラーが発生します。
- 未定義の空間参照システム (SRS) でジオメトリ引数が構文的に整形形式のジオメトリである場合、`ER_SRS_NOT_FOUND` エラーが発生します。
- X または Y 座標引数が指定され、値が `-inf`、`+inf` または `NaN` の場合、`ER_DATA_OUT_OF_RANGE` エラーが発生します。
- 経度または緯度の値が範囲外の場合は、エラーが発生します:
 - 経度の値が `(-180, 180]` の範囲にない場合は、`ER_LONGITUDE_OUT_OF_RANGE` エラーが発生します。
 - 緯度の値が `[-90, 90]` の範囲にない場合は、`ER_LATITUDE_OUT_OF_RANGE` エラーが発生します。

表示される範囲は度数です。浮動小数点演算のため、正確な範囲制限はわずかに偏差します。

- それ以外の場合、戻り値は `NULL` 以外です。

ポイントプロパティを取得するには、次の関数を使用できます:

- `ST_Latitude(p [, new_latitude_val])`

地理空間参照システム (SRS) を持つ有効な `Point` オブジェクト `p` を表す単一の引数を使用すると、`ST_Latitude()` は `p` の緯度値を倍精度数値として返します。

有効な緯度値を表すオプションの第 2 引数を使用すると、`ST_Latitude()` は、最初の引数のように、緯度が 2 番目の引数と等しい `Point` オブジェクトを返します。

`ST_Latitude()` では、このセクションの概要で説明されているように引数が処理されますが、`Point` オブジェクトは有効ですが地理 SRS がない場合は、`ER_SRS_NOT_GEOGRAPHIC` エラーが発生します。

```
mysql> SET @pt = ST_GeomFromText('POINT(45 90)', 4326);
mysql> SELECT ST_Latitude(@pt);
+-----+
| ST_Latitude(@pt) |
+-----+
|          45 |
+-----+
mysql> SELECT ST_AsText(ST_Latitude(@pt, 10));
+-----+
| ST_AsText(ST_Latitude(@pt, 10)) |
+-----+
| POINT(10 90) |
+-----+
```

この関数は、MySQL 8.0.12 で追加されました。

- `ST_Longitude(p [, new_longitude_val])`

地理空間参照システム (SRS) を持つ有効な `Point` オブジェクト `p` を表す単一の引数を使用すると、`ST_Longitude()` は `p` の経度値を倍精度数値として返します。

オプションの第 2 引数がある有効な経度値を表す場合、`ST_Longitude()` は、経度が 2 番目の引数と等しい最初の引数のような `Point` オブジェクトを返します。

`ST_Longitude()` では、このセクションの概要で説明されているように引数が処理されますが、`Point` オブジェクトは有効ですが地理 SRS がいない場合は、`ER_SRS_NOT_GEOGRAPHIC` エラーが発生します。

```
mysql> SET @pt = ST_GeomFromText('POINT(45 90)', 4326);
mysql> SELECT ST_Longitude(@pt);
+-----+
| ST_Longitude(@pt) |
+-----+
|          90 |
+-----+
mysql> SELECT ST_AsText(ST_Longitude(@pt, 10));
+-----+
| ST_AsText(ST_Longitude(@pt, 10)) |
+-----+
| POINT(45 10) |
+-----+
```

この関数は、MySQL 8.0.12 で追加されました。

- `ST_X(p [, new_x_val])`

有効な `Point` オブジェクト `p` を表す単一の引数を使用すると、`ST_X()` は `p` の X 座標値を倍精度数値として返します。MySQL 8.0.12 では、X 座標は `Point` 空間参照システム (SRS) 定義で最初に表示される軸を参照しているとみなされます。

オプションの second 引数を指定すると、`ST_X()` は、X 座標が 2 番目の引数と等しい最初の引数のような `Point` オブジェクトを返します。MySQL 8.0.12 では、`Point` オブジェクトに地理 SRS がある場合、2 番目の引数は経度または緯度の値の適切な範囲内にある必要があります。

`ST_X()` は、このセクションの概要で説明されているように引数を処理します。

```
mysql> SELECT ST_X(Point(56.7, 53.34));
+-----+
| ST_X(Point(56.7, 53.34)) |
+-----+
|          56.7 |
+-----+
mysql> SELECT ST_AsText(ST_X(Point(56.7, 53.34), 10.5));
+-----+
| ST_AsText(ST_X(Point(56.7, 53.34), 10.5)) |
+-----+
| POINT(10.5 53.34) |
+-----+
```

- `ST_Y(p [, new_y_val])`

有効な **Point** オブジェクト **p** を表す単一の引数を使用すると、**ST_Y()** は **p** の Y 座標値を倍精度の数値として返します。MySQL 8.0.12 では、Y 座標は **Point** 空間参照システム (SRS) 定義で次に表示される軸を参照しているとみなされます。

オプションの second 引数を指定すると、**ST_Y()** は、Y 座標が 2 番目の引数と等しい最初の引数のような **Point** オブジェクトを返します。MySQL 8.0.12 では、**Point** オブジェクトに地理 SRS がある場合、2 番目の引数は経度または緯度の値の適切な範囲内にある必要があります。

ST_Y() は、このセクションの概要で説明されているように引数を処理します。

```
mysql> SELECT ST_Y(Point(56.7, 53.34));
+-----+
| ST_Y(Point(56.7, 53.34)) |
+-----+
|          53.34 |
+-----+
mysql> SELECT ST_AsText(ST_Y(Point(56.7, 53.34), 10.5));
+-----+
| ST_AsText(ST_Y(Point(56.7, 53.34), 10.5)) |
+-----+
| POINT(56.7 10.5) |
+-----+
```

12.17.7.3 LineString および MultiLineString のプロパティ関数

LineString は、**Point** 値で構成されます。**LineString** の特定の点を抽出したり、そこに含まれている点の数をカウントしたり、その長さを取得したりできます。

このセクションの一部の関数は、**MultiLineString** 値に対しても機能します。

特に指定がないかぎり、このセクションの関数はジオメトリ引数を次のように処理します:

- 引数が **NULL** の場合、またはジオメトリ引数が空のジオメトリの場合、戻り値は **NULL** です。
- ジオメトリ引数が構文的に整形形式のジオメトリでない場合は、**ER_GIS_INVALID_DATA** エラーが発生します。
- 未定義の空間参照システム (SRS) でジオメトリ引数が構文的に整形形式のジオメトリである場合、**ER_SRS_NOT_FOUND** エラーが発生します。
- それ以外の場合、戻り値は **NULL** 以外です。

linestring プロパティを取得するには、次の関数を使用できます:

- **ST_EndPoint(Is)**

LineString 値 **Is** の終点である **Point** を返します。

ST_EndPoint() は、このセクションの概要で説明されているように引数を処理します。

```
mysql> SET @Is = 'LineString(1 1,2 2,3 3)';
mysql> SELECT ST_AsText(ST_EndPoint(ST_GeomFromText(@Is)));
+-----+
| ST_AsText(ST_EndPoint(ST_GeomFromText(@Is))) |
+-----+
| POINT(3 3) |
+-----+
```

- **ST_IsClosed(Is)**

LineString 値が **Is** の場合、**Is** がクローズされると、**ST_IsClosed()** は 1 を返します (つまり、**ST_StartPoint()** と **ST_EndPoint()** の値が同じです)。

MultiLineString 値が **Is** の場合、**Is** がクローズされると、**ST_IsClosed()** は 1 を返します (つまり、**ST_StartPoint()** と **ST_EndPoint()** の値は **Is** の **LineString** ごとに同じです)。

ST_IsClosed() は、**Is** がクローズされていない場合は 0 を返し、**Is** が **NULL** の場合は **NULL** を返します。

`ST_IsClosed()` は、このセクションの概要で説明されているように引数を処理しますが、次の例外があります:

- ジオメトリに地理空間参照システム (SRS) の SRID 値がある場合、`ER_NOT_IMPLEMENTED_FOR_GEOGRAPHIC_SRS` エラーが発生します。

```
mysql> SET @ls1 = 'LineString(1 1,2 2,3 3,2 2)';
mysql> SET @ls2 = 'LineString(1 1,2 2,3 3,1 1)';

mysql> SELECT ST_IsClosed(ST_GeomFromText(@ls1));
+-----+
| ST_IsClosed(ST_GeomFromText(@ls1)) |
+-----+
| 0 |
+-----+

mysql> SELECT ST_IsClosed(ST_GeomFromText(@ls2));
+-----+
| ST_IsClosed(ST_GeomFromText(@ls2)) |
+-----+
| 1 |
+-----+

mysql> SET @ls3 = 'MultiLineString((1 1,2 2,3 3),(4 4,5 5))';

mysql> SELECT ST_IsClosed(ST_GeomFromText(@ls3));
+-----+
| ST_IsClosed(ST_GeomFromText(@ls3)) |
+-----+
| 0 |
+-----+
```

- `ST_Length(Is [, unit])`

関連付けられた空間参照システムにおける `LineString` または `MultiLineString` 値 `Is` の長さを示す倍精度数値を戻します。 `MultiLineString` 値の長さは、その要素の長さの合計と等しくなります。

`ST_Length()` では、次のように結果が計算されます:

- ジオメトリがデカルト SRS の有効な `LineString` である場合、戻り値はジオメトリのデカルト長です。
- ジオメトリがデカルト SRS 内の有効な `MultiLineString` である場合、戻り値はその要素のデカルト長の合計です。
- ジオメトリが地理 SRS 内の有効な `LineString` である場合、戻り値はその SRS 内のジオメトリの測地長 (メートル) です。
- ジオメトリが地理 SRS 内の有効な `MultiLineString` である場合、戻り値は SRS 内の要素の測地長の合計 (メートル) です。

`ST_Length()` は、このセクションの概要で説明されているように、引数を処理しますが、次の例外があります:

- ジオメトリが `LineString` または `MultiLineString` でない場合、戻り値は `NULL` です。
- ジオメトリがジオメトリ学的に無効な場合は、結果の長さが未定義 (任意の数値) であるか、エラーが発生します。
- 長さの計算結果が `+inf` の場合、`ER_DATA_OUT_OF_RANGE` エラーが発生します。
- ジオメトリに経度または緯度が範囲外の地理 SRS がある場合、エラーが発生します:
 - 経度の値が `(-180, 180]` の範囲内でない場合は、`ER_GEOMETRY_PARAM_LONGITUDE_OUT_OF_RANGE` エラーが発生します (MySQL 8.0.12 より前の `ER_LONGITUDE_OUT_OF_RANGE`)。
 - 緯度の値が `[-90, 90]` の範囲内でない場合は、`ER_GEOMETRY_PARAM_LATITUDE_OUT_OF_RANGE` エラーが発生します (MySQL 8.0.12 より前の `ER_LATITUDE_OUT_OF_RANGE`)。

表示される範囲は度数です。浮動小数点演算のため、正確な範囲制限はわずかに偏差します。

MySQL 8.0.16 以降、`ST_Length()` では、戻される長さの値の線形単位を指定するオプションの `unit` 引数が許可されます。次のルールが適用されます:

- ユニットが指定されているが、MySQL でサポートされていない場合は、`ER_UNIT_NOT_FOUND` エラーが発生します。
- サポートされている線形単位が指定され、SRID が 0 の場合、`ER_GEOMETRY_IN_UNKNOWN_LENGTH_UNIT` エラーが発生します。
- サポートされている線形単位が指定され、SRID が 0 でない場合、結果はその単位になります。
- 単位が指定されていない場合、結果はデカルトか地理的にかかわらず、ジオメトリの SRS の単位になります。現在、すべての MySQL SRS はメートルで表されます。

ユニットは、`INFORMATION_SCHEMA ST_UNITS_OF_MEASURE` テーブルで見つかった場合にサポートされます。[セクション 26.37 「INFORMATION_SCHEMA ST_UNITS_OF_MEASURE テーブル」](#) を参照してください。

```
mysql> SET @ls = ST_GeomFromText('LineString(1 1,2 2,3 3)');
mysql> SELECT ST_Length(@ls);
+-----+
| ST_Length(@ls) |
+-----+
| 2.8284271247461903 |
+-----+

mysql> SET @mls = ST_GeomFromText('MultiLineString((1 1,2 2,3 3),(4 4,5 5))');
mysql> SELECT ST_Length(@mls);
+-----+
| ST_Length(@mls) |
+-----+
| 4.242640687119286 |
+-----+

mysql> SET @ls = ST_GeomFromText('LineString(1 1,2 2,3 3)', 4326);
mysql> SELECT ST_Length(@ls);
+-----+
| ST_Length(@ls) |
+-----+
| 313701.9623204328 |
+-----+
mysql> SELECT ST_Length(@ls, 'metre');
+-----+
| ST_Length(@ls, 'metre') |
+-----+
| 313701.9623204328 |
+-----+
mysql> SELECT ST_Length(@ls, 'foot');
+-----+
| ST_Length(@ls, 'foot') |
+-----+
| 1029205.9131247795 |
+-----+
```

- `ST_NumPoints(ls)`

`LineString` 値 `ls` 内の `Point` オブジェクトの数を返します。

`ST_NumPoints()` は、このセクションの概要で説明されているように引数を処理します。

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT ST_NumPoints(ST_GeomFromText(@ls));
+-----+
| ST_NumPoints(ST_GeomFromText(@ls)) |
+-----+
| 3 |
+-----+
```


- `ST_PointN(Is, N)`

`LineString` 値 `Is` 内の `N` 番目の `Point` を返します。点の番号は 1 から始まります。

`ST_PointN()` は、このセクションの概要で説明されているように引数を処理します。

```
mysql> SET @Is = 'LineString(1 1,2 2,3 3)';
mysql> SELECT ST_AsText(ST_PointN(ST_GeomFromText(@Is),2));
+-----+
| ST_AsText(ST_PointN(ST_GeomFromText(@Is),2)) |
+-----+
| POINT(2 2) |
+-----+
```

- `ST_StartPoint(Is)`

`LineString` 値 `Is` の始点である `Point` を返します。

`ST_StartPoint()` は、このセクションの概要で説明されているように引数を処理します。

```
mysql> SET @Is = 'LineString(1 1,2 2,3 3)';
mysql> SELECT ST_AsText(ST_StartPoint(ST_GeomFromText(@Is)));
+-----+
| ST_AsText(ST_StartPoint(ST_GeomFromText(@Is))) |
+-----+
| POINT(1 1) |
+-----+
```

12.17.7.4 Polygon および MultiPolygon プロパティ関数

このセクションの関数は、`Polygon` または `MultiPolygon` の値のプロパティを返します。

特に指定がないかぎり、このセクションの関数はジオメトリ引数を次のように処理します:

- 引数が `NULL` の場合、またはジオメトリ引数が空のジオメトリの場合、戻り値は `NULL` です。
- ジオメトリ引数が構文的に整形式のジオメトリでない場合は、`ER_GIS_INVALID_DATA` エラーが発生します。
- 未定義の空間参照システム (SRS) でジオメトリ引数が構文的に整形式のジオメトリである場合、`ER_SRS_NOT_FOUND` エラーが発生します。
- 複数のジオメトリ引数を取る関数では、それらの引数が同じ SRS 内にある場合、`ER_GIS_DIFFERENT_SRIDS` エラーが発生します。
- それ以外の場合、戻り値は `NULL` 以外です。

ポリゴンのプロパティを取得するには、次の関数を使用できます:

- `ST_Area({poly|mpoly})`

空間参照システムで測定された、`Polygon` または `MultiPolygon` 引数の領域を示す倍精度数値を返します。

MySQL 8.0.13 では、`ST_Area()` はこのセクションの概要で説明されているように引数を処理しますが、次の例外があります:

- ジオメトリがジオメトリ学的に無効な場合は、結果が未定義の領域 (任意の数値) であるか、エラーが発生します。
- ジオメトリは有効だが、`Polygon` または `MultiPolygon` オブジェクトではない場合、`ER_UNEXPECTED_GEOMETRY_TYPE` エラーが発生します。
- ジオメトリがデカルト SRS 内の有効な `Polygon` である場合、結果はポリゴンのデカルト領域になります。
- ジオメトリがデカルト SRS 内の有効な `MultiPolygon` である場合、結果はポリゴンのデカルト領域の合計になります。
- ジオメトリが地理 SRS 内の有効な `Polygon` である場合、結果はその SRS 内のポリゴンの測地領域 (平方メートル) になります。

- ジオメトリが地理 SRS 内の有効な **MultiPolygon** である場合、結果はその SRS 内のポリゴンの測地領域の合計 (平方メートル) になります。
- 面積計算の結果が `+inf` になった場合は、**ER_DATA_OUT_OF_RANGE** エラーが発生します。
- ジオメトリに経度または緯度が範囲外の地理 SRS がある場合、エラーが発生します:
 - 経度の値が `(-180, 180]` の範囲内がない場合は、**ER_GEOMETRY_PARAM_LONGITUDE_OUT_OF_RANGE** エラーが発生します (MySQL 8.0.12 より前の **ER_LONGITUDE_OUT_OF_RANGE**)。
 - 緯度の値が `[-90, 90]` の範囲内がない場合は、**ER_GEOMETRY_PARAM_LATITUDE_OUT_OF_RANGE** エラーが発生します (MySQL 8.0.12 より前の **ER_LATITUDE_OUT_OF_RANGE**)。

表示される範囲は度数です。浮動小数点演算のため、正確な範囲制限はわずかに偏差します。

MySQL 8.0.13 より前では、**ST_Area()** はこのセクションの概要で説明されているように引数を処理しますが、次の例外があります:

- 次元 0 または 1 の引数の場合、結果は 0 です。
- ジオメトリが空の場合、戻り値は **NULL** ではなく 0 です。
- ジオメトリコレクションの場合、結果はすべてのコンポーネントの面積値の合計になります。ジオメトリコレクションが空の場合、その領域は 0(ゼロ)として返されます。
- ジオメトリに地理空間参照システム (SRS) の SRID 値がある場合、**ER_NOT_IMPLEMENTED_FOR_GEOGRAPHIC_SRS** エラーが発生します。

```
mysql> SET @poly =
'Polygon((0 0,0 3,3 0,0 0),(1 1,1 2,2 1,1 1));
mysql> SELECT ST_Area(ST_GeomFromText(@poly));
+-----+
| ST_Area(ST_GeomFromText(@poly)) |
+-----+
|                4 |
+-----+

mysql> SET @mpoly =
'MultiPolygon(((0 0,0 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1)));
mysql> SELECT ST_Area(ST_GeomFromText(@mpoly));
+-----+
| ST_Area(ST_GeomFromText(@mpoly)) |
+-----+
|                8 |
+-----+
```

• **ST_Centroid({poly|mpoly})**

Polygon または **MultiPolygon** 引数の数学的重心を **Point** として返します。結果が **MultiPolygon** 上にある保証はありません。

この関数は、コレクション内の最上位次元のコンポーネントの中心点を計算してジオメトリコレクションを処理します。このようなコンポーネントは、集中計算のために単一の **MultiPolygon**、**MultiLineString** または **MultiPoint** に抽出および作成されます。

ST_Centroid() は、このセクションの概要で説明されているように引数を処理しますが、次の例外があります:

- 引数が空のジオメトリコレクションであるという追加条件では、戻り値は **NULL** です。
- ジオメトリに地理空間参照システム (SRS) の SRID 値がある場合、**ER_NOT_IMPLEMENTED_FOR_GEOGRAPHIC_SRS** エラーが発生します。

```
mysql> SET @poly =
ST_GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 5,5 5));
mysql> SELECT ST_GeometryType(@poly),ST_AsText(ST_Centroid(@poly));
+-----+
```

```
| ST_GeometryType(@poly) | ST_AsText(ST_Centroid(@poly)) |
+-----+-----+
| POLYGON                | POINT(4.958333333333333 4.958333333333333) |
+-----+-----+
```

- [ST_ExteriorRing\(poly\)](#)

`Polygon` 値 `poly` の外側のリングを `LineString` として返します。

`ST_ExteriorRing()` は、このセクションの概要で説明されているように引数を処理します。

```
mysql> SET @poly =
'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1));
mysql> SELECT ST_AsText(ST_ExteriorRing(ST_GeomFromText(@poly)));
+-----+
| ST_AsText(ST_ExteriorRing(ST_GeomFromText(@poly))) |
+-----+
| LINESTRING(0 0,0 3,3 3,3 0,0 0) |
+-----+
```

- [ST_InteriorRingN\(poly, N\)](#)

`Polygon` 値 `poly` の `N` 番目の内側のリングを `LineString` として返します。リングの番号は 1 から始まります。

`ST_InteriorRingN()` は、このセクションの概要で説明されているように引数を処理します。

```
mysql> SET @poly =
'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1));
mysql> SELECT ST_AsText(ST_InteriorRingN(ST_GeomFromText(@poly),1));
+-----+
| ST_AsText(ST_InteriorRingN(ST_GeomFromText(@poly),1)) |
+-----+
| LINESTRING(1 1,1 2,2 2,2 1,1 1) |
+-----+
```

- [ST_NumInteriorRing\(poly\)](#), [ST_NumInteriorRings\(poly\)](#)

`Polygon` 値 `poly` 内の内側のリングの数を返します。

`ST_NumInteriorRing()` および `ST_NumInteriorRings()` は、このセクションの概要で説明されているように引数を処理します。

```
mysql> SET @poly =
'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1));
mysql> SELECT ST_NumInteriorRings(ST_GeomFromText(@poly));
+-----+
| ST_NumInteriorRings(ST_GeomFromText(@poly)) |
+-----+
| 1 |
+-----+
```

12.17.7.5 GeometryCollection プロパティ関数

これらの関数は、`GeometryCollection` 値のプロパティを返します。

特に指定がない限り、このセクションの関数はジオメトリ引数を次のように処理します:

- 引数が `NULL` の場合、またはジオメトリ引数が空のジオメトリの場合、戻り値は `NULL` です。
- ジオメトリ引数が構文的に整形式のジオメトリでない場合は、`ER_GIS_INVALID_DATA` エラーが発生します。
- 未定義の空間参照システム (SRS) でジオメトリ引数が構文的に整形式のジオメトリである場合、`ER_SRS_NOT_FOUND` エラーが発生します。
- それ以外の場合、戻り値は `NULL` 以外です。

ジオメトリコレクションプロパティを取得するには、次の関数を使用できます:

- [ST_GeometryN\(gc, N\)](#)

`GeometryCollection` 値 `gc` 内の `N` 番目のジオメトリを返します。ジオメトリの番号は 1 から始まります。

`ST_GeometryN()` は、このセクションの概要で説明されているように引数を処理します。

```
mysql> SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2, 3 3));
mysql> SELECT ST_AsText(ST_GeometryN(ST_GeomFromText(@gc),1));
+-----+
| ST_AsText(ST_GeometryN(ST_GeomFromText(@gc),1)) |
+-----+
| POINT(1 1) |
+-----+
```

- `ST_NumGeometries(gc)`

`GeometryCollection` 値 `gc` 内のジオメトリの数を返します。

`ST_NumGeometries()` は、このセクションの概要で説明されているように引数を処理します。

```
mysql> SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2, 3 3));
mysql> SELECT ST_NumGeometries(ST_GeomFromText(@gc));
+-----+
| ST_NumGeometries(ST_GeomFromText(@gc)) |
+-----+
| 2 |
+-----+
```

12.17.8 空間演算子関数

OpenGIS では、ジオメトリを生成できる関数がいくつか提案されています。これらは、空間演算子を実装するように設計されています。これらの関数は、[Open Geospatial Consortium](#) 仕様に従って適用できないものを除き、すべての引数タイプの組合せをサポートします。

MySQL は、関数の説明に記載されているように、OpenGIS の拡張機能である特定の関数も実装します。

特に指定がないかぎり、このセクションの関数はジオメトリ引数を次のように処理します:

- いずれかの引数が `NULL` の場合、戻り値は `NULL` です。
- ジオメトリ引数が構文的に整形形式のジオメトリでない場合は、`ER_GIS_INVALID_DATA` エラーが発生します。
- 未定義の空間参照システム (SRS) でジオメトリ引数が構文的に整形形式のジオメトリである場合、`ER_SRS_NOT_FOUND` エラーが発生します。
- 複数のジオメトリ引数を取る関数では、それらの引数が同じ SRS 内でない場合、`ER_GIS_DIFFERENT_SRIDS` エラーが発生します。
- ジオメトリ引数に地理 SRS の SRID 値があり、関数が地理ジオメトリを処理しない場合は、`ER_NOT_IMPLEMENTED_FOR_GEOGRAPHIC_SRS` エラーが発生します。
- 地理 SRS ジオメトリ引数で、範囲外の経度または緯度を持つ引数がある場合、エラーが発生します:
 - 経度の値が $(-180, 180]$ の範囲内でない場合は、`ER_GEOMETRY_PARAM_LONGITUDE_OUT_OF_RANGE` エラーが発生します (MySQL 8.0.12 より前の `ER_LONGITUDE_OUT_OF_RANGE`)。
 - 緯度の値が $[-90, 90]$ の範囲内でない場合は、`ER_GEOMETRY_PARAM_LATITUDE_OUT_OF_RANGE` エラーが発生します (MySQL 8.0.12 より前の `ER_LATITUDE_OUT_OF_RANGE`)。

表示される範囲は度数です。SRS で別の単位が使用されている場合、範囲ではその単位に対応する値が使用されます。浮動小数点演算のため、正確な範囲制限はわずかに偏差します。

- それ以外の場合、戻り値は `NULL` 以外です。

次の空間演算子関数を使用できます:

- `ST_Buffer(g, d [, strategy1 [, strategy2 [, strategy3]])`

ジオメトリ値 `g` からの距離が `d` の距離以下であるすべての点を表すジオメトリを返します。

ジオメトリ引数が空の場合、`ST_Buffer()` は空のジオメトリを戻します。

距離が 0 の場合、`ST_Buffer()` はジオメトリ引数を変更せずに戻します:

```
mysql> SET @pt = ST_GeomFromText('POINT(0 0)');
mysql> SELECT ST_AsText(ST_Buffer(@pt, 0));
+-----+
| ST_AsText(ST_Buffer(@pt, 0)) |
+-----+
| POINT(0 0) |
+-----+
```

`ST_Buffer()` では、`Polygon` 値と `MultiPolygon` 値、および `Polygon` 値または `MultiPolygon` 値を含むジオメトリコレクションに対して負の距離がサポートされています。結果は空のジオメトリである可能性があります。

`ST_Buffer()` では、`distance` 引数の後に最大 3 つのオプションの戦略引数を指定できます。戦略はバッファ計算に影響します。これらの引数は、`ST_Buffer_Strategy()` 関数によって生成されるバイト文字列値で、点、結合および終了方針に使用されます:

- 点方針は、`Point` および `MultiPoint` ジオメトリに適用されます。点ストラテジが指定されていない場合、デフォルトは `ST_Buffer_Strategy('point_circle', 32)` です。
- 結合方法は、`LineString`、`MultiLineString`、`Polygon` および `MultiPolygon` ジオメトリに適用されます。結合方針が指定されていない場合、デフォルトは `ST_Buffer_Strategy('join_round', 32)` です。
- 終了方針は、`LineString` および `MultiLineString` ジオメトリに適用されます。終了方針が指定されていない場合、デフォルトは `ST_Buffer_Strategy('end_round', 32)` です。

各タイプの最大 1 つの戦略を指定でき、任意の順序で指定できます。

`ST_Buffer()` は、このセクションの概要で説明されているように、引数を処理しますが、次の例外があります:

- `Point`、`MultiPoint`、`LineString` の負の距離と `MultiLineString` 値、および `Polygon` または `MultiPolygon` 値を含まないジオメトリコレクションでは、`ER_WRONG_ARGUMENTS` エラーが発生します。
- 特定のタイプの戦略が複数指定されている場合は、`ER_WRONG_ARGUMENTS` エラーが発生します。

```
mysql> SET @pt = ST_GeomFromText('POINT(0 0)');
mysql> SET @pt_strategy = ST_Buffer_Strategy('point_square');
mysql> SELECT ST_AsText(ST_Buffer(@pt, 2, @pt_strategy));
+-----+
| ST_AsText(ST_Buffer(@pt, 2, @pt_strategy)) |
+-----+
| POLYGON((-2 -2,2 -2,2 2,-2 2,-2 -2)) |
+-----+
```

```
mysql> SET @ls = ST_GeomFromText('LINESTRING(0 0,0 5,5 5)');
mysql> SET @end_strategy = ST_Buffer_Strategy('end_flat');
mysql> SET @join_strategy = ST_Buffer_Strategy('join_round', 10);
mysql> SELECT ST_AsText(ST_Buffer(@ls, 5, @end_strategy, @join_strategy));
+-----+
| ST_AsText(ST_Buffer(@ls, 5, @end_strategy, @join_strategy)) |
+-----+
| POLYGON((5 5,5 10,0 10,-3.5355339059327373 8.535533905932738, |
| -5 5,-5 0,0 0,5 0,5 5)) |
+-----+
```

- `ST_Buffer_Strategy(strategy [, points_per_circle])`

この関数は、バッファの計算に影響を与えるために `ST_Buffer()` で使用する戦略バイト文字列を戻します。

戦略に関する情報は、Boost.org で入手できます。

最初の引数は、戦略オプションを示す文字列である必要があります:

- ポイント戦略の場合、許可される値は 'point_circle' および 'point_square' です。
- 結合方針の場合、許可される値は 'join_round' および 'join_miter' です。
- 終了戦略の場合、許可される値は 'end_round' および 'end_flat' です。

最初の引数が 'point_circle', 'join_round', 'join_miter' または 'end_round' の場合、`points_per_circle` 引数は正の数値で指定する必要があります。 `points_per_circle` の最大値は、`max_points_in_geometry` システム変数の値です。

例については、`ST_Buffer()` の説明を参照してください。

`ST_Buffer_Strategy()` は、このセクションの概要で説明されているように、引数を処理しますが、次の例外があります:

- 無効な引数があると、`ER_WRONG_ARGUMENTS` エラーが発生します。
- 最初の引数が 'point_square' または 'end_flat' の場合、`points_per_circle` 引数を指定しないでください。指定すると、`ER_WRONG_ARGUMENTS` エラーが発生します。

- `ST_ConvexHull(g)`

ジオメトリ値 `g` の凸型ハルを表すジオメトリを戻します。

この関数は、まず頂点ポイントが同一直線かどうかをチェックして、ジオメトリ凸型を計算します。その場合は線形ハルを返し、それ以外の場合はポリゴンハルを返します。この関数は、コレクションのすべてのコンポーネントのすべての頂点ポイントを抽出し、そこから `MultiPoint` 値を作成して凸型を計算することで、ジオメトリコレクションを処理します。

`ST_ConvexHull()` は、このセクションの概要で説明されているように引数を処理しますが、次の例外があります:

- 引数が空のジオメトリコレクションであるという追加条件では、戻り値は `NULL` です。

```
mysql> SET @g = 'MULTIPOINT(5 0,25 0,15 10,15 25)';
mysql> SELECT ST_AsText(ST_ConvexHull(ST_GeomFromText(@g)));
+-----+
| ST_AsText(ST_ConvexHull(ST_GeomFromText(@g))) |
+-----+
| POLYGON((5 0,25 0,15 25,5 0)) |
+-----+
```

- `ST_Difference(g1, g2)`

ジオメトリ値 `g1` と `g2` の点集合の差集合を表すジオメトリを返します。

`ST_Difference()` は、このセクションの概要で説明されているように引数を処理します。

```
mysql> SET @g1 = Point(1,1), @g2 = Point(2,2);
mysql> SELECT ST_AsText(ST_Difference(@g1, @g2));
+-----+
| ST_AsText(ST_Difference(@g1, @g2)) |
+-----+
| POINT(1 1) |
+-----+
```

- `ST_Intersection(g1, g2)`

ジオメトリ値 `g1` と `g2` の点集合の共通集合を表すジオメトリを返します。

`ST_Intersection()` は、このセクションの概要で説明されているように引数を処理します。


```
mysql> SET @g1 = ST_GeomFromText('LineString(1 1, 3 3)');
mysql> SET @g2 = ST_GeomFromText('LineString(1 3, 3 1)');
mysql> SELECT ST_AsText(ST_Intersection(@g1, @g2));
+-----+
| ST_AsText(ST_Intersection(@g1, @g2)) |
+-----+
| POINT(2 2) |
+-----+
```

- [ST_LineInterpolatePoint\(ls, fractional_distance\)](#)

この関数は、範囲[0.0, 1.0]内の [LineString](#) ジオメトリと小数距離を取得し、[LineString](#) に沿って、始点から終点までの指定された距離分の [Point](#) を戻します。これを使用すると、ジオメトリ引数で記述された道路の途中に [Point](#) があるかどうかなどの質問に回答できます。

この関数は、すべての空間参照システム (デカルトと地理の両方) の [LineString](#) ジオメトリに対して実装されます。

[fractional_distance](#) 引数が 1.0 の場合、近似値の計算で数値が不正確になるため、結果は [LineString](#) 引数の正確な最後のポイントではなく、その近くのポイントになる可能性があります。

関連する関数 [ST_LineInterpolatePoints\(\)](#) は同様の引数を取りますが、開始点からそのエンドポイントまでの距離の各部分の [LineString](#) に沿った [Point](#) 値で構成される [MultiPoint](#) を返します。両方の関数の例については、[ST_LineInterpolatePoints\(\)](#) の説明を参照してください。

[ST_LineInterpolatePoint\(\)](#) は、このセクションの概要で説明されているように、引数を処理しますが、次の例外があります:

- ジオメトリ引数が [LineString](#) でない場合は、[ER_UNEXPECTED_GEOMETRY_TYPE](#) エラーが発生します。
- 分数距離引数が [0.0, 1.0] の範囲外の場合、[ER_DATA_OUT_OF_RANGE](#) エラーが発生します。

[ST_LineInterpolatePoint\(\)](#) は、OpenGIS の MySQL 拡張機能です。この関数は、MySQL 8.0.24 で追加されました。

- [ST_LineInterpolatePoints\(ls, fractional_distance\)](#)

この関数は、範囲 (0.0, 1.0) 内の [LineString](#) ジオメトリおよび小数距離を取得し、[LineString](#) の開始点と、その開始点からエンドポイントまでの距離の各部分の [LineString](#) に沿った [Point](#) 値で構成される [MultiPoint](#) を戻します。これを使用すると、ジオメトリ引数で記述された道路に沿って 10% ずつの [Point](#) 値があるかなどの質問に答えることができます。

この関数は、すべての空間参照システム (デカルトと地理の両方) の [LineString](#) ジオメトリに対して実装されます。

[fractional_distance](#) 引数がゼロの余りで 1.0 を除算する場合、結果には [LineString](#) 引数の最後のポイントが含まれず、近似値の計算における数値の不正確のためにその近くのポイントが含まれる可能性があります。

関連する関数である [ST_LineInterpolatePoint\(\)](#) は、同様の引数を取りますが、開始点からエンドポイントまでの指定された距離で、[LineString](#) に沿って [Point](#) を返します。

[ST_LineInterpolatePoints\(\)](#) は、このセクションの概要で説明されているように、引数を処理しますが、次の例外があります:

- ジオメトリ引数が [LineString](#) でない場合は、[ER_UNEXPECTED_GEOMETRY_TYPE](#) エラーが発生します。
- 分数距離引数が [0.0, 1.0] の範囲外の場合、[ER_DATA_OUT_OF_RANGE](#) エラーが発生します。

```
mysql> SET @ls1 = ST_GeomFromText('LINESTRING(0 0,0 5,5 5)');
mysql> SELECT ST_AsText(ST_LineInterpolatePoint(@ls1, .5));
+-----+
| ST_AsText(ST_LineInterpolatePoint(@ls1, .5)) |
+-----+
| POINT(0 5) |
+-----+
mysql> SELECT ST_AsText(ST_LineInterpolatePoint(@ls1, .75));
+-----+
| ST_AsText(ST_LineInterpolatePoint(@ls1, .75)) |
+-----+
```

```

+-----+
| POINT(2.5 5) |
+-----+
mysql> SELECT ST_AsText(ST_LineInterpolatePoint(@ls1, 1));
+-----+
| ST_AsText(ST_LineInterpolatePoint(@ls1, 1)) |
+-----+
| POINT(5 5) |
+-----+
mysql> SELECT ST_AsText(ST_LineInterpolatePoints(@ls1, .25));
+-----+
| ST_AsText(ST_LineInterpolatePoints(@ls1, .25)) |
+-----+
| MULTIPOINT((0 2.5),(0 5),(2.5 5),(5 5)) |
+-----+

```

`ST_LineInterpolatePoints()` は、OpenGIS の MySQL 拡張機能です。この関数は、MySQL 8.0.24 で追加されました。

- `ST_PointAtDistance(ls, distance)`

この関数は、`LineString` ジオメトリと、`LineString` の空間参照システム (SRS) の単位で測定された `[0.0, ST_Length(ls)]` の範囲内の距離を取得し、開始点からその距離にある `LineString` に沿って `Point` を戻します。geometry 引数で記述された道路の始点から 400 メートルの `Point` 値などの質問に答えるために使用できます。

この関数は、すべての空間参照システム (デカルトと地理の両方) の `LineString` ジオメトリに対して実装されます。

`ST_PointAtDistance()` は、このセクションの概要で説明されているように、引数を処理しますが、次の例外があります:

- ジオメトリ引数が `LineString` でない場合は、`ER_UNEXPECTED_GEOMETRY_TYPE` エラーが発生します。
- 分数距離引数が `[0.0, ST_Length(ls)]` の範囲外の場合、`ER_DATA_OUT_OF_RANGE` エラーが発生します。

`ST_PointAtDistance()` は、OpenGIS の MySQL 拡張機能です。この関数は、MySQL 8.0.24 で追加されました。

- `ST_SymDifference(g1, g2)`

ジオメトリ値 `g1` と `g2` の点集合の対称差を表すジオメトリを返します。これは、次のように定義されます。

```
g1 symdifference g2 := (g1 union g2) difference (g1 intersection g2)
```

または、関数呼び出しの表記では次のようになります。

```
ST_SymDifference(g1, g2) = ST_Difference(ST_Union(g1, g2), ST_Intersection(g1, g2))
```

`ST_SymDifference()` は、このセクションの概要で説明されているように引数を処理します。

```

mysql> SET @g1 = Point(1,1), @g2 = Point(2,2);
mysql> SELECT ST_AsText(ST_SymDifference(@g1, @g2));
+-----+
| ST_AsText(ST_SymDifference(@g1, @g2)) |
+-----+
| MULTIPOINT((1 1),(2 2)) |
+-----+

```

- `ST_Transform(g, target_srid)`

ジオメトリをある空間参照システム (SRS) から別の空間参照システムに変換します。戻り値は、すべての座標がターゲット SRID (`target_srid`) に変換された入力ジオメトリと同じタイプのジオメトリです。ジオメトリ引数の

SRID がターゲット SRID 値と同じでないかぎり、変換のサポートは地理的 SRS に制限されます。この場合、戻り値は有効な SRS の入力ジオメトリです。

`ST_Transform()` は、このセクションの概要で説明されているように、引数を処理しますが、次の例外があります:

- 地理 SRS の SRID 値を持つジオメトリ引数では、エラーは発生しません。
- ジオメトリまたはターゲット SRID 引数に、未定義の空間参照システム (SRS) を参照する SRID 値がある場合、`ER_SRS_NOT_FOUND` エラーが発生します。
- ジオメトリが `ST_Transform()` で変換できない SRS 内にある場合は、`ER_TRANSFORM_SOURCE_SRS_NOT_SUPPORTED` エラーが発生します。
- ターゲット SRID が `ST_Transform()` が変換できない SRS 内にある場合は、`ER_TRANSFORM_TARGET_SRS_NOT_SUPPORTED` エラーが発生します。
- ジオメトリが WGS 84 以外の SRS にあり、TOWGS84 句がない場合は、`ER_TRANSFORM_SOURCE_SRS_MISSING_TOWGS84` エラーが発生します。
- ターゲット SRID が WGS 84 以外の SRS にあり、TOWGS84 句がない場合は、`ER_TRANSFORM_TARGET_SRS_MISSING_TOWGS84` エラーが発生します。

`ST_SRID(g, target_srid)` と `ST_Transform(g, target_srid)` は、次の点で異なります:

- `ST_SRID()` は、座標を変換せずにジオメトリ SRID 値を変更します。
- `ST_Transform()` は、SRID 値の変更に加えてジオメトリ座標を変換します。

```
mysql> SET @p = ST_GeomFromText('POINT(52.381389 13.064444)', 4326);
mysql> SELECT ST_AsText(@p);
+-----+
| ST_AsText(@p) |
+-----+
| POINT(52.381389 13.064444) |
+-----+
mysql> SET @p = ST_Transform(@p, 4230);
mysql> SELECT ST_AsText(@p);
+-----+
| ST_AsText(@p) |
+-----+
| POINT(52.38208611407426 13.065520672345304) |
+-----+
```

• `ST_Union(g1, g2)`

ジオメトリ値 `g1` と `g2` の点集合の和集合を表すジオメトリを返します。

`ST_Union()` は、このセクションの概要で説明されているように引数を処理します。

```
mysql> SET @g1 = ST_GeomFromText('LineString(1 1, 3 3)');
mysql> SET @g2 = ST_GeomFromText('LineString(1 3, 3 1)');
mysql> SELECT ST_AsText(ST_Union(@g1, @g2));
+-----+
| ST_AsText(ST_Union(@g1, @g2)) |
+-----+
| MULTILINESTRING((1 1,3 3),(1 3,3 1)) |
+-----+
```

さらに、[セクション12.17.7「ジオメトリプロパティ関数」](#)では、既存のジオメトリから新しいジオメトリを構築するいくつかの関数について説明しています。これらの関数の説明については、そのセクションを参照してください。

- `ST_Envelope(g)`
- `ST_StartPoint(ls)`
- `ST_EndPoint(ls)`
- `ST_PointN(ls, N)`

- [ST_ExteriorRing\(poly\)](#)
- [ST_InteriorRingN\(poly, N\)](#)
- [ST_GeometryN\(gc, N\)](#)

12.17.9 ジオメトリオブジェクト間の空間関係をテストする関数

このセクションで説明されている関数は、引数として2つのジオメトリを受け取り、それらの間の定性的または定量的な関係を返します。

MySQL は、OpenGIS 仕様で定義されている関数名を使用して、2つの関数セットを実装します。一方のセットは正確なオブジェクトシェイプを使用して2つのジオメトリ値間の関係をテストし、もう一方のセットはオブジェクトの最小境界矩形 (MBR) を使用します。

12.17.9.1 オブジェクト形状を使用する空間関係関数

OpenGIS 仕様では、正確なオブジェクト形状を使用して、`g1` と `g2` の2つのジオメトリ値間の関係をテストする次の関数が定義されています。戻り値 1 および 0 はそれぞれ true および false を示しますが、距離関数は距離値を返しません。

このセクションの関数は、デカルトまたは地理空間参照システム (SRS) の引数を検出し、SRS に適した結果を返します。

特に指定がない限り、このセクションの関数はジオメトリ引数を次のように処理します:

- 引数が `NULL` の場合、またはジオメトリ引数が空のジオメトリの場合、戻り値は `NULL` です。
- ジオメトリ引数が構文的に整形形式のジオメトリでない場合は、`ER_GIS_INVALID_DATA` エラーが発生します。
- 未定義の空間参照システム (SRS) でジオメトリ引数が構文的に整形形式のジオメトリである場合、`ER_SRS_NOT_FOUND` エラーが発生します。
- 複数のジオメトリ引数を取る関数では、それらの引数が同じ SRS 内でない場合、`ER_GIS_DIFFERENT_SRIDS` エラーが発生します。
- ジオメトリ学的に無効なジオメトリ引数がある場合は、結果が true または false (どちらでも定義されていない) が、エラーが発生します。
- 地理 SRS ジオメトリ引数で、範囲外の経度または緯度を持つ引数がある場合、エラーが発生します:
 - 経度の値が $(-180, 180]$ の範囲内でない場合は、`ER_GEOMETRY_PARAM_LONGITUDE_OUT_OF_RANGE` エラーが発生します (MySQL 8.0.12 より前の `ER_LONGITUDE_OUT_OF_RANGE`)。
 - 緯度の値が $[-90, 90]$ の範囲内でない場合は、`ER_GEOMETRY_PARAM_LATITUDE_OUT_OF_RANGE` エラーが発生します (MySQL 8.0.12 より前の `ER_LATITUDE_OUT_OF_RANGE`)。

表示される範囲は度数です。SRS で別の単位が使用されている場合、範囲ではその単位に対応する値が使用されます。浮動小数点演算のため、正確な範囲制限はわずかに偏差します。

- それ以外の場合、戻り値は `NULL` 以外です。

このセクションの一部の関数では、戻り値の長さの単位を指定する `unit` 引数が許可されます。特に指定がない限り、関数は単位引数を次のように処理します:

- ユニットの単位は、`INFORMATION_SCHEMA ST_UNITS_OF_MEASURE` テーブルで見つかった場合にサポートされます。[セクション26.37 「INFORMATION_SCHEMA ST_UNITS_OF_MEASURE テーブル」](#) を参照してください。
- ユニットの単位が指定されているが、MySQL でサポートされていない場合は、`ER_UNIT_NOT_FOUND` エラーが発生します。
- サポートされている線形単位が指定され、SRID が 0 の場合、`ER_GEOMETRY_IN_UNKNOWN_LENGTH_UNIT` エラーが発生します。
- サポートされている線形単位が指定され、SRID が 0 でない場合、結果はその単位になります。

- 単位が指定されていない場合、結果はデカルトか地理的にかかわらず、ジオメトリの SRS の単位になります。現在、すべての MySQL SRS はメートルで表されます。

ジオメトリ関係のテストには、次のオブジェクト形状関数を使用できます:

- `ST_Contains(g1, g2)`

`g1` が `g2` を完全に含んでいるかどうかを示す 1 または 0 を返します。これは、`ST_Within()` とは逆の関係をテストします。

`ST_Contains()` は、このセクションの概要で説明されているように引数を処理します。

- `ST_Crosses(g1, g2)`

空間リレーションに次のプロパティがある場合、2 つのジオメトリが空間的に交差になります:

- `g1` と `g2` の両方が次元 1 でない場合: `g2` の内部に `g1` の内部と共通の点があるが、`g2` が `g1` の内部全体をカバーしていない場合、`g1` は `g2` を横断します。
- `g1` と `g2` の両方が次元 1 の場合: 有限の数の点で線が相互に交差する場合 (つまり、共通の線セグメントがない場合、共通の単一の点のみ)。

この関数は、`g1` が `g2` を空間的に横断するかどうかを示す 1 または 0 を返します。

`ST_Crosses()` は、次の追加条件で戻り値が `NULL` であることを除き、このセクションの概要で説明されているように引数を処理します:

- `g1` の次元は 2 (`Polygon` または `MultiPolygon`) です。
- `g2` の次元は 1 (`Point` または `MultiPoint`) です。

- `ST_Disjoint(g1, g2)`

`g1` が `g2` と空間的に切り離されている (交差していない) かどうかを示す 1 または 0 を返します。

`ST_Disjoint()` は、このセクションの概要で説明されているように引数を処理します。

- `ST_Distance(g1, g2 [, unit])`

`g1` と `g2` の間の距離を返します。ジオメトリ引数の空間参照システム (SRS) の長さ単位で測定されるか、オプションの `unit` 引数 (指定されている場合) の単位で測定されます。

この関数は、2 つのジオメトリ引数のコンポーネントのすべての組合せ間の最短距離を返すことで、ジオメトリコレクションを処理します。

`ST_Distance()` は、このセクションの概要で説明されているようにジオメトリ引数を処理しますが、次の例外があります:

- `ST_Distance()` は、地理的 (楕円体) 空間参照システム内の引数を検出し、楕円体上の測地距離を返します。MySQL 8.0.18 では、`ST_Distance()` はすべてのジオメトリタイプの地理 SRS 引数の距離計算をサポートしています。MySQL 8.0.18 より前では、許可されている地理的引数の型は、`Point` と `Point`、または `Point` と `MultiPoint` (任意の引数の順序) のみです。地理 SRS で他のジオメトリタイプの引数の組合せを使用してコールすると、`ER_NOT_IMPLEMENTED_FOR_GEOGRAPHIC_SRS` エラーが発生します。
- いずれかの引数がジオメトリ学的に無効な場合は、結果が未定義の距離 (任意の数値) であるか、エラーが発生します。
- 中間結果または最終結果で `NaN` または負の数値が生成されると、`ER_GIS_INVALID_DATA` エラーが発生します。

MySQL 8.0.14 以降、`ST_Distance()` では、戻される距離値の線形単位を指定するオプションの `unit` 引数が許可されます。`ST_Distance()` は、このセクションの概要で説明されているように、その `unit` 引数を処理します。

```
mysql> SET @g1 = ST_GeomFromText('POINT(1 1)');  
mysql> SET @g2 = ST_GeomFromText('POINT(2 2)');
```

```
mysql> SELECT ST_Distance(@g1, @g2);
+-----+
| ST_Distance(@g1, @g2) |
+-----+
| 1.4142135623730951 |
+-----+

mysql> SET @g1 = ST_GeomFromText('POINT(1 1)', 4326);
mysql> SET @g2 = ST_GeomFromText('POINT(2 2)', 4326);
mysql> SELECT ST_Distance(@g1, @g2);
+-----+
| ST_Distance(@g1, @g2) |
+-----+
| 156874.3859490455 |
+-----+

mysql> SELECT ST_Distance(@g1, @g2, 'metre');
+-----+
| ST_Distance(@g1, @g2, 'metre') |
+-----+
| 156874.3859490455 |
+-----+

mysql> SELECT ST_Distance(@g1, @g2, 'foot');
+-----+
| ST_Distance(@g1, @g2, 'foot') |
+-----+
| 514679.7439273146 |
+-----+
```

球の距離計算の特殊なケースについては、[ST_Distance_Sphere\(\)](#) 関数を参照してください。

• [ST_Equals\(g1, g2\)](#)

`g1` が `g2` と空間的に等しいかどうかを示す 1 または 0 を返します。

[ST_Equals\(\)](#) は、空のジオメトリ引数に対して `NULL` を返さない点を除き、このセクションの概要で説明されているように引数を処理します。

```
mysql> SET @g1 = Point(1,1), @g2 = Point(2,2);
mysql> SELECT ST_Equals(@g1, @g1), ST_Equals(@g1, @g2);
+-----+-----+
| ST_Equals(@g1, @g1) | ST_Equals(@g1, @g2) |
+-----+-----+
| 1 | 0 |
+-----+-----+
```

• [ST_FrechetDistance\(g1, g2 \[, unit\]\)](#)

ジオメトリがどの程度類似しているかを反映して、2つのジオメトリ間の離散フレシェ距離を返します。結果は、ジオメトリ引数の空間参照システム (SRS) の長さ単位、または引数が指定されている場合は `unit` 引数の長さ単位で測定された倍精度の数値です。

この関数は、離散フレシェ距離を実装します。つまり、ジオメトリの点間の距離に制限されます。たとえば、2つの `LineString` 引数がある場合、ジオメトリに明示的に記述された点のみが考慮されます。これらの点の間の線分セグメント上の点は考慮されません。

[ST_FrechetDistance\(\)](#) は、このセクションの概要で説明されているようにジオメトリ引数を処理しますが、次の例外があります：

- ジオメトリはデカルトまたは地理 SRS を持つことができますが、`LineString` 値のみがサポートされています。引数が同一のデカルト SRS または地理 SRS 内にあり、どちらかが `LineString` でない場合は、SRS タイプに応じて `ER_NOT_IMPLEMENTED_FOR_CARTESIAN_SRS` または `ER_NOT_IMPLEMENTED_FOR_GEOGRAPHIC_SRS` エラーが発生します。

[ST_FrechetDistance\(\)](#) は、このセクションの概要で説明されているように、オプションの `unit` 引数を処理します。

```
mysql> SET @ls1 = ST_GeomFromText('LINESTRING(0 0,0 5,5 5)');
mysql> SET @ls2 = ST_GeomFromText('LINESTRING(0 1,0 6,3 3,5 6)');
mysql> SELECT ST_FrechetDistance(@ls1, @ls2);
+-----+
| ST_FrechetDistance(@ls1, @ls2) |
+-----+
```



```
+-----+
| 2.8284271247461903 |
+-----+

mysql> SET @ls1 = ST_GeomFromText('LINESTRING(0 0,0 5,5 5)', 4326);
mysql> SET @ls2 = ST_GeomFromText('LINESTRING(0 1,0 6,3 3,5 6)', 4326);
mysql> SELECT ST_FrechetDistance(@ls1, @ls2);
+-----+
| ST_FrechetDistance(@ls1, @ls2) |
+-----+
| 313421.1999416798 |
+-----+
mysql> SELECT ST_FrechetDistance(@ls1, @ls2, 'foot');
+-----+
| ST_FrechetDistance(@ls1, @ls2, 'foot') |
+-----+
| 1028284.7767115477 |
+-----+
```

この関数は、MySQL 8.0.23 で追加されました。

- [ST_HausdorffDistance\(g1, g2 \[, unit\]\)](#)

ジオメトリがどの程度類似しているかを反映して、2つのジオメトリ間の離散ハスドルフ距離を返します。結果は、ジオメトリ引数の空間参照システム (SRS) の長さ単位、または引数が指定されている場合は `unit` 引数の長さ単位で測定された倍精度の数値です。

この関数は、離散ハスドルフ距離を実装します。つまり、ジオメトリの点間の距離に制限されます。たとえば、2つの `LineString` 引数がある場合、ジオメトリに明示的に記述された点のみが考慮されます。これらの点の間の線分セグメント上の点は考慮されません。

[ST_HausdorffDistance\(\)](#) は、このセクションの概要で説明されているようにジオメトリ引数を処理しますが、次の例外があります:

- ジオメトリ引数が同じデカルト SRS または地理 SRS 内にあるが、サポートされている組合せにない場合は、SRS タイプに応じて [ER_NOT_IMPLEMENTED_FOR_CARTESIAN_SRS](#) または [ER_NOT_IMPLEMENTED_FOR_GEOGRAPHIC_SRS](#) エラーが発生します。次の組合せがサポートされています:
 - `LineString` および `LineString`
 - `Point` および `MultiPoint`
 - `LineString` および `MultiLineString`
 - `MultiPoint` および `MultiPoint`
 - `MultiLineString` および `MultiLineString`

[ST_HausdorffDistance\(\)](#) は、このセクションの概要で説明されているように、オプションの `unit` 引数を処理しません。

```
mysql> SET @ls1 = ST_GeomFromText('LINESTRING(0 0,0 5,5 5)');
mysql> SET @ls2 = ST_GeomFromText('LINESTRING(0 1,0 6,3 3,5 6)');
mysql> SELECT ST_HausdorffDistance(@ls1, @ls2);
+-----+
| ST_HausdorffDistance(@ls1, @ls2) |
+-----+
| 1 |
+-----+

mysql> SET @ls1 = ST_GeomFromText('LINESTRING(0 0,0 5,5 5)', 4326);
mysql> SET @ls2 = ST_GeomFromText('LINESTRING(0 1,0 6,3 3,5 6)', 4326);
mysql> SELECT ST_HausdorffDistance(@ls1, @ls2);
+-----+
| ST_HausdorffDistance(@ls1, @ls2) |
+-----+
| 111319.49079326246 |
+-----+
```

```
mysql> SELECT ST_HausdorffDistance(@ls1, @ls2, 'foot');
+-----+
| ST_HausdorffDistance(@ls1, @ls2, 'foot') |
+-----+
|                365221.4264870815 |
+-----+
```

この関数は、MySQL 8.0.23 で追加されました。

- [ST_Intersects\(g1, g2\)](#)

`g1` が `g2` と空間的に交差しているかどうかを示す 1 または 0 を返します。

`ST_Intersects()` は、このセクションの概要で説明されているように引数を処理します。

- [ST_Overlaps\(g1, g2\)](#)

2 つのジオメトリが交差し、その交差によって同じ次元のジオメトリが生成され、指定されたジオメトリのいずれとも等しくない場合、これらのジオメトリは空間的にオーバーラップになります。

この関数は、`g1` が `g2` と空間的にオーバーラップするかどうかを示す 1 または 0 を返します。

`ST_Overlaps()` は、このセクションの概要で説明されているように引数を処理します。ただし、戻り値は、2 つのジオメトリの次元が等しくないという追加条件では `NULL` です。

- [ST_Touches\(g1, g2\)](#)

内部が交差しませんが、いずれかのジオメトリの境界が他のジオメトリの境界または内部と交差する場合、2 つのジオメトリは空間的なタッチです。

この関数は、`g1` が `g2` に空間的に接触するかどうかを示す 1 または 0 を返します。

`ST_Touches()` では、両方のジオメトリの次元が 0 (`Point` または `MultiPoint`) である追加条件の戻り値が `NULL` であることを除き、このセクションの概要で説明するように引数が処理されます。

- [ST_Within\(g1, g2\)](#)

`g1` が空間的に `g2` の内部にあるかどうかを示す 1 または 0 を返します。これは、`ST_Contains()` とは逆の関係をテストします。

`ST_Within()` は、このセクションの概要で説明されているように引数を処理します。

12.17.9.2 最小境界矩形を使用する空間リレーション関数

MySQL には、`g1` と `g2` の 2 つのジオメトリの最小境界矩形 (MBR) 間の関係をテストする複数の MySQL 固有関数が用意されています。戻り値 1 と 0 は、それぞれ `true` と `false` を示します。

ポイントのバウンディングボックスは、境界と内部の両方のポイントとして解釈されます。

直線の水平線または垂直線の枠ボックスは、線の内側も境界である線として解釈されます。端点は境界点です。

いずれかのパラメータがジオメトリコレクションの場合、それらのパラメータの内部、境界および外部は、コレクション内のすべての要素の和集合のパラメータです。

このセクションの関数は、デカルトまたは地理空間参照システム (SRS) の引数を検出し、SRS に適した結果を返します。

特に指定がない限り、このセクションの関数はジオメトリ引数を次のように処理します:

- 引数が `NULL` または空のジオメトリの場合、戻り値は `NULL` です。
- ジオメトリ引数が構文的に整形形式のジオメトリでない場合は、`ER_GIS_INVALID_DATA` エラーが発生します。
- 未定義の空間参照システム (SRS) でジオメトリ引数が構文的に整形形式のジオメトリである場合、`ER_SRS_NOT_FOUND` エラーが発生します。

- 複数のジオメトリ引数を取る関数では、それらの引数が同じ SRS 内がない場合、[ER_GIS_DIFFERENT_SRIDS](#) エラーが発生します。
- いずれかの引数がジオメトリ学的に無効な場合、結果は true または false (どちらでも定義されていない) になるか、エラーが発生します。
- 地理 SRS ジオメトリ引数で、範囲外の経度または緯度を持つ引数がある場合、エラーが発生します:
 - 経度の値が (-180, 180] の範囲内がない場合は、[ER_GEOMETRY_PARAM_LONGITUDE_OUT_OF_RANGE](#) エラーが発生します (MySQL 8.0.12 より前の [ER_LONGITUDE_OUT_OF_RANGE](#))。
 - 緯度の値が [-90, 90] の範囲内がない場合は、[ER_GEOMETRY_PARAM_LATITUDE_OUT_OF_RANGE](#) エラーが発生します (MySQL 8.0.12 より前の [ER_LATITUDE_OUT_OF_RANGE](#))。

表示される範囲は度数です。SRS で別の単位が使用されている場合、範囲ではその単位に対応する値が使用されません。浮動小数点演算のため、正確な範囲制限はわずかに偏差します。

- それ以外の場合、戻り値は `NULL` 以外です。

ジオメトリ関係のテストには、次の MBR 関数を使用できます:

- [MBRContains\(g1, g2\)](#)

`g1` の最小境界矩形が `g2` の最小境界矩形を含んでいるかどうかを示す 1 または 0 を返します。これは、[MBRWithin\(\)](#) とは逆の関係をテストします。

[MBRContains\(\)](#) は、このセクションの概要で説明されているように引数を処理します。

```
mysql> SET @g1 = ST_GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = ST_GeomFromText('Point(1 1)');
mysql> SELECT MBRContains(@g1,@g2), MBRWithin(@g2,@g1);
+-----+-----+
| MBRContains(@g1,@g2) | MBRWithin(@g2,@g1) |
+-----+-----+
| 1 | 1 |
+-----+-----+
```

- [MBRCoveredBy\(g1, g2\)](#)

`g1` の最小境界矩形が `g2` の最小境界矩形で覆われているかどうかを示す 1 または 0 を返します。これにより、[MBRCovers\(\)](#) とは反対の関係がテストされます。

[MBRCoveredBy\(\)](#) は、このセクションの概要で説明されているように引数を処理します。

```
mysql> SET @g1 = ST_GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = ST_GeomFromText('Point(1 1)');
mysql> SELECT MBRCovers(@g1,@g2), MBRCoveredBy(@g1,@g2);
+-----+-----+
| MBRCovers(@g1,@g2) | MBRCoveredBy(@g1,@g2) |
+-----+-----+
| 1 | 0 |
+-----+-----+
mysql> SELECT MBRCovers(@g2,@g1), MBRCoveredBy(@g2,@g1);
+-----+-----+
| MBRCovers(@g2,@g1) | MBRCoveredBy(@g2,@g1) |
+-----+-----+
| 0 | 1 |
+-----+-----+
```

- [MBRCovers\(g1, g2\)](#)

`g1` の最小境界矩形が `g2` の最小境界矩形をカバーしているかどうかを示す 1 または 0 を返します。これにより、[MBRCoveredBy\(\)](#) とは反対の関係がテストされます。例については、[MBRCoveredBy\(\)](#) の説明を参照してください。

[MBRCovers\(\)](#) は、このセクションの概要で説明されているように引数を処理します。

- [MBRDisjoint\(g1, g2\)](#)

2つのジオメトリ `g1` と `g2` の最小境界矩形が切り離されている (交差していない) かどうかを示す 1 または 0 を返します。

`MBRDisjoint()` は、このセクションの概要で説明されているように引数を処理します。

- `MBREquals(g1, g2)`

2つのジオメトリ `g1` と `g2` の最小境界矩形が同じであるかどうかを示す 1 または 0 を返します。

`MBREquals()` は、空のジオメトリ引数に対して `NULL` を返さない点を除き、このセクションの概要で説明されているように引数を処理します。

- `MBRIntersects(g1, g2)`

2つのジオメトリ `g1` と `g2` の最小境界矩形が交差しているかどうかを示す 1 または 0 を返します。

`MBRIntersects()` は、このセクションの概要で説明されているように引数を処理します。

- `MBROverlaps(g1, g2)`

2つのジオメトリが交差し、その交差によって同じ次元のジオメトリが生成され、指定されたジオメトリのいずれとも等しくない場合、これらのジオメトリは空間的にオーバーラップになります。

この関数は、2つのジオメトリ `g1` と `g2` の最小境界矩形が重複しているかどうかを示す 1 または 0 を返します。

`MBROverlaps()` は、このセクションの概要で説明されているように引数を処理します。

- `MBRTouches(g1, g2)`

内部が交差しませんが、いずれかのジオメトリの境界が他のジオメトリの境界または内部と交差する場合、2つのジオメトリは空間的なタッチです。

この関数は、2つのジオメトリ `g1` および `g2` の最小境界矩形が接触するかどうかを示す 1 または 0 を返します。

`MBRTouches()` は、このセクションの概要で説明されているように引数を処理します。

- `MBRWithin(g1, g2)`

`g1` の最小境界矩形が `g2` の最小境界矩形の内部にあるかどうかを示す 1 または 0 を返します。これは、`MBRContains()` とは逆の関係をテストします。

`MBRWithin()` は、このセクションの概要で説明されているように引数を処理します。

```
mysql> SET @g1 = ST_GeomFromText('Polygon((0 0,0 3,3 3,0 0))');
mysql> SET @g2 = ST_GeomFromText('Polygon((0 0,0 5,5 5,0 0))');
mysql> SELECT MBRWithin(@g1,@g2), MBRWithin(@g2,@g1);
+-----+-----+
| MBRWithin(@g1,@g2) | MBRWithin(@g2,@g1) |
+-----+-----+
| 1 | 0 |
+-----+-----+
```

12.17.10 空間 Geohash 関数

geohash は、任意の精度の緯度と経度の座標をテキスト文字列にエンコードするためのシステムです。geohash 値は、"0123456789bcdefghjkmnpqrstuvwxy" から選択された文字のみを含む文字列です。

このセクションの関数を使用すると、geohash データのインポートとエクスポート、および geohash 値のインデックス付けと検索の機能をアプリケーションに提供する geohash 値を操作できます。

特に指定がない限り、このセクションの関数はジオメトリ引数を次のように処理します:

- いずれかの引数が `NULL` の場合、戻り値は `NULL` です。
- 無効な引数がある場合は、エラーが発生します。

- 範囲外の経度または緯度を持つ引数がある場合、エラーが発生します:
 - 経度の値が $(-180, 180]$ の範囲内でない場合は、`ER_GEOMETRY_PARAM_LONGITUDE_OUT_OF_RANGE` エラーが発生します (MySQL 8.0.12 より前の `ER_LONGITUDE_OUT_OF_RANGE`)。
 - 緯度の値が $[-90, 90]$ の範囲内でない場合は、`ER_GEOMETRY_PARAM_LATITUDE_OUT_OF_RANGE` エラーが発生します (MySQL 8.0.12 より前の `ER_LATITUDE_OUT_OF_RANGE`)。

表示される範囲は度数です。浮動小数点演算のため、正確な範囲制限はわずかに偏差します。
- いずれかの点引数に SRID 0 または 4326 がない場合は、`ER_SRS_NOT_FOUND` エラーが発生します。point 引数 SRID の有効性はチェックされません。
- SRID 引数が未定義の空間参照システム (SRS) を参照している場合、`ER_SRS_NOT_FOUND` エラーが発生します。
- SRID 引数が 32 ビット符号なし整数の範囲内でない場合は、`ER_DATA_OUT_OF_RANGE` エラーが発生します。
- それ以外の場合、戻り値は `NULL` 以外です。

次の geohash 関数を使用できます:

- `ST_GeoHash(longitude, latitude, max_length)`, `ST_GeoHash(point, max_length)`

接続文字セットおよび照合順序で geohash 文字列を戻します。

最初の構文では、`longitude` は $[-180, 180]$ の範囲の数値である必要があり、`latitude` は $[-90, 90]$ の範囲の数値である必要があります。2 番目の構文では、`POINT` 値が必要です。X 座標と Y 座標は、それぞれ経度と緯度の有効範囲内にあります。

結果の文字列は `max_length` 文字以下で、上限は 100 です。geohash 値を作成するアルゴリズムでは、場所または `max_length` 文字の正確な表現のいずれか (いずれか早い方) の文字列が作成されるまで続行されるため、文字列が `max_length` 文字より短い場合があります。

`ST_GeoHash()` は、このセクションの概要で説明されているように引数を処理します。

```
mysql> SELECT ST_GeoHash(180,0,10), ST_GeoHash(-180,-90,15);
+-----+-----+
| ST_GeoHash(180,0,10) | ST_GeoHash(-180,-90,15) |
+-----+-----+
| xbpbbppbpb | 0000000000000000 |
+-----+-----+
```

- `ST_LatFromGeoHash(geohash_str)`

geohash 文字列値から緯度を範囲 $[-90, 90]$ の倍精度数値として返します。

`ST_LatFromGeoHash()` デコード関数は、`geohash_str` 引数から 433 文字以内を読み取ります。座標値の内部表現における情報の上限を表します。433rd を超える文字は、それ以外の場合でも無視され、エラーが発生します。

`ST_LatFromGeoHash()` は、このセクションの概要で説明されているように引数を処理します。

```
mysql> SELECT ST_LatFromGeoHash(ST_GeoHash(45,-20,10));
+-----+
| ST_LatFromGeoHash(ST_GeoHash(45,-20,10)) |
+-----+
| -20 |
+-----+
```

- `ST_LongFromGeoHash(geohash_str)`

geohash 文字列値から経度を範囲 $[-180, 180]$ の倍精度数値として返します。

`geohash_str` 引数から処理される最大文字数に関する `ST_LatFromGeoHash()` の説明の注釈は、`ST_LongFromGeoHash()` にも適用されます。

`ST_LongFromGeoHash()` は、このセクションの概要で説明されているように引数を処理します。

```
mysql> SELECT ST_LongFromGeoHash(ST_GeoHash(45,-20,10));
+-----+
| ST_LongFromGeoHash(ST_GeoHash(45,-20,10)) |
+-----+
|                45 |
+-----+
```

- `ST_PointFromGeoHash(geohash_str, srid)`

geohash 文字列値を指定して、デコードされた geohash 値を含む POINT 値を返します。

点の X 座標と Y 座標は、それぞれ[-180, 180]の範囲の経度と[-90, 90]の範囲の緯度です。

srid 引数は 32 ビットの符号なし整数です。

geohash_str 引数から処理される最大文字数に関する `ST_LatFromGeoHash()` の説明の注釈は、`ST_PointFromGeoHash()` にも適用されます。

`ST_PointFromGeoHash()` は、このセクションの概要で説明されているように引数を処理します。

```
mysql> SET @gh = ST_GeoHash(45,-20,10);
mysql> SELECT ST_AsText(ST_PointFromGeoHash(@gh,0));
+-----+
| ST_AsText(ST_PointFromGeoHash(@gh,0)) |
+-----+
| POINT(45 -20) |
+-----+
```

12.17.11 空間 GeoJSON 関数

このセクションでは、GeoJSON ドキュメントと空間値の間の変換関数について説明します。GeoJSON は、ジオメトリ/地理的特徴をエンコードするためのオープンスタンダードです。詳細は、<http://geojson.org> を参照してください。ここで説明する関数は、GeoJSON 仕様リビジョン 1.0 に従います。

GeoJSON では、MySQL でサポートされているものと同じジオメトリデータ型または地理データ型がサポートされます。ジオメトリオブジェクトが抽出される点を除き、フィーチャおよび FeatureCollection オブジェクトはサポートされていません。CRS のサポートは SRID を識別する値に制限されています。

MySQL では、JSON 値に対する操作を可能にするために、ネイティブの JSON データ型および一連の SQL 関数もサポートされています。詳細は、[セクション11.5「JSON データ型」](#) および [セクション12.18「JSON 関数」](#) を参照してください。

- `ST_AsGeoJSON(g [, max_dec_digits [, options]])`

ジオメトリ `g` から GeoJSON オブジェクトを生成します。オブジェクト文字列には、接続文字セットと照合順序があります。

いずれかの引数が NULL の場合、戻り値は NULL です。NULL 以外の引数が無効な場合は、エラーが発生します。

max_dec_digits を指定すると、座標の小数点以下の桁数が制限され、出力が丸められます。指定しない場合、この引数のデフォルトは $2^{32}-1$ の最大値です。最小値は 0 です。

options はビットマスクです (指定されている場合)。次のテーブルに、許可されるフラグ値を示します。ジオメトリ引数の SRID が 0 の場合は、要求するフラグ値に対しても CRS オブジェクトは生成されません。

フラグ値	意味
0	オプションなし。これは、options が指定されていない場合のデフォルトです。
1	バウンディングボックスを出力に追加します。
2	短い形式の CRS URN を出力に追加します。デフォルトの形式は短い形式 (EPSG:srid) です。
4	長い形式の CRS URN (urn:ogc:def:crs:EPSG::srid) を追加します。このフラグはフラグ 2 をオーバーライドし

フラグ値	意味
	ます。たとえば、オプション値 5 と 7 は同じことを意味します (バウンディングボックスと長い形式の CRS URN を追加します)。

```
mysql> SELECT ST_AsGeoJSON(ST_GeomFromText('POINT(11.11111 12.22222)'),2);
+-----+
| ST_AsGeoJSON(ST_GeomFromText('POINT(11.11111 12.22222)'),2) |
+-----+
| {"type": "Point", "coordinates": [11.11, 12.22]} |
+-----+
```

• ST_GeomFromGeoJSON(str [, options [, srid]])

GeoJSON オブジェクトを表す文字列 `str` を解析し、ジオメトリを戻します。

いずれかの引数が `NULL` の場合、戻り値は `NULL` です。 `NULL` 以外の引数が無効な場合は、エラーが発生します。

`options` では、座標ディメンションが 2 より大きいジオメトリを含む GeoJSON 文書の処理方法を説明します (指定されている場合)。次のテーブルに、許可される `options` 値を示します。

オプション値	意味
1	ドキュメントを却下し、エラーを生成します。これは、 <code>options</code> が指定されていない場合のデフォルトです。
2, 3, 4	ドキュメントを受け入れ、より高い座標寸法の座標を除去します。

`options` の値が 2、3 および 4 の場合、現在も同じ結果になります。座標ディメンションが 2 より大きいジオメトリが将来サポートされる場合は、これらの値によって異なる効果が生じる可能性があります。

`srid` 引数を指定する場合は、32 ビットの符号なし整数である必要があります。指定しない場合、ジオメトリの戻り値の SRID は 4326 になります。

`srid` が未定義の空間参照システム (SRS) を参照すると、`ER_SRS_NOT_FOUND` エラーが発生します。

地理 SRS ジオメトリ引数で、範囲外の経度または緯度を持つ引数がある場合、エラーが発生します:

- 経度の値が `(-180, 180]` の範囲にない場合は、`ER_LONGITUDE_OUT_OF_RANGE` エラーが発生します。
- 緯度の値が `[-90, 90]` の範囲にない場合は、`ER_LATITUDE_OUT_OF_RANGE` エラーが発生します。

表示される範囲は度数です。SRS で別の単位が使用されている場合、範囲ではその単位に対応する値が使用されます。浮動小数点演算のため、正確な範囲制限はわずかに偏差します。

GeoJSON ジオメトリ、地物および地物コレクションオブジェクトには、`crs` プロパティがある場合があります。解析関数は、`urn:ogc:def:crs:EPSG::srid` および `EPSG:srid` ネームスペース内の名前付き CRS URN を解析しますが、リンクオブジェクトとして指定された CRS は解析しません。また、`urn:ogc:def:crs:OGC:1.3:CRS84` は SRID 4326 として認識されます。オブジェクトに認識されない CRS がある場合、エラーが発生しますが、オプションの `srid` 引数が指定された場合、CRS は無効であっても無視されます。

トップレベルオブジェクト SRID とは異なる SRID を指定する `crs` メンバーが GeoJSON 文書の下位レベルで見つかった場合、`ER_INVALID_GEOJSON_CRS_NOT_TOP_LEVEL` エラーが発生します。

GeoJSON 仕様で指定されているように、GeoJSON 入力 (`Point`、`LineString` など) の `type` メンバーの解析では大/小文字が区別されます。この指定は、MySQL では大/小文字が区別されない、他の解析の大/小文字の区別に関するサイレントです。

この例は、単純な GeoJSON オブジェクトの解析結果を示しています。座標の順序が使用される SRID に依存することを確認します。

```
mysql> SET @json = '{"type": "Point", "coordinates": [102.0, 0.0]}';
```

```
mysql> SELECT ST_AsText(ST_GeomFromGeoJSON(@json));
+-----+
| ST_AsText(ST_GeomFromGeoJSON(@json)) |
+-----+
| POINT(0 102) |
+-----+
mysql> SELECT ST_SRID(ST_GeomFromGeoJSON(@json));
+-----+
| ST_SRID(ST_GeomFromGeoJSON(@json)) |
+-----+
| 4326 |
+-----+
mysql> SELECT ST_AsText(ST_SRID(ST_GeomFromGeoJSON(@json),0));
+-----+
| ST_AsText(ST_SRID(ST_GeomFromGeoJSON(@json),0)) |
+-----+
| POINT(102 0) |
+-----+
```

12.17.12 空間集計関数

MySQL は、一連の値に対して計算を実行する集計関数をサポートしています。これらの関数の一般的な情報は、[セクション12.20.1「集計関数の説明」](#)を参照してください。このセクションでは、[ST_Collect\(\)](#) の空間集計関数について説明します。

[ST_Collect\(\)](#) は、オプションの [OVER](#) 句を表す、[\[over_clause\]](#)による構文の説明で示されているように、ウィンドウ関数として使用できます。[over_clause](#) については、[セクション12.21.2「Window 関数の概念と構文」](#)で説明されています。[セクション12.21.2「Window 関数の概念と構文」](#)には、ウィンドウ関数の使用方法に関するその他の情報も含まれています。

- [ST_Collect\(\[DISTINCT\] g\) \[over_clause\]](#)

ジオメトリ値を集計し、単一のジオメトリコレクション値を返します。 [DISTINCT](#) オプションを指定すると、個別のジオメトリ引数の集計が戻されます。

他の集計関数と同様に、[GROUP BY](#) を使用して引数をサブセットにグループ化できます。 [ST_Collect\(\)](#) は、各サブセットの集計値を返します。

[over_clause](#) が存在する場合、この関数はウィンドウ関数として実行されます。[over_clause](#) については、[セクション12.21.2「Window 関数の概念と構文」](#)を参照してください。ウィンドウ集計をサポートするほとんどの集計関数とは対照的に、[ST_Collect\(\)](#) では [over_clause](#) を [DISTINCT](#) とともに使用できます。

[ST_Collect\(\)](#) は、その引数を次のように処理します:

- [NULL](#) 引数は無視されます。
- すべての引数が [NULL](#) の場合、または集計結果が空の場合、戻り値は [NULL](#) です。
- ジオメトリ引数が構文的に整形形式のジオメトリでない場合は、[ER_GIS_INVALID_DATA](#) エラーが発生します。
- 未定義の空間参照システム (SRS) でジオメトリ引数が構文的に整形形式のジオメトリである場合、[ER_SRS_NOT_FOUND](#) エラーが発生します。
- 複数のジオメトリ引数があり、それらの引数が同じ SRS 内にある場合、戻り値はその SRS 内にあります。これらの引数が同じ SRS 内にはない場合は、[ER_GIS_DIFFERENT_SRIDS_AGGREGATION](#) エラーが発生します。
- 結果は、次のように [NULL](#) 以外のジオメトリ引数から決定された結果タイプを使用して、可能な限り狭い [MultiXxx](#) または [GeometryCollection](#) 値になります:
 - すべての引数が [Point](#) 値の場合、結果は [MultiPoint](#) 値になります。
 - すべての引数が [LineString](#) 値の場合、結果は [MultiLineString](#) 値になります。
 - すべての引数が [Polygon](#) 値の場合、結果は [MultiPolygon](#) 値になります。
 - それ以外の場合、引数はジオメトリタイプが混在し、結果は [GeometryCollection](#) 値になります。

このデータセットの例は、製造年および製造場所別の仮想製品を示しています:

```
CREATE TABLE product (
  year INTEGER,
  product VARCHAR(256),
  location Geometry
);

INSERT INTO product
(year, product, location) VALUES
(2000, "Calculator", ST_GeomFromText('point(60 -24)',4326)),
(2000, "Computer" , ST_GeomFromText('point(28 -77)',4326)),
(2000, "Abacus"   , ST_GeomFromText('point(28 -77)',4326)),
(2000, "TV"       , ST_GeomFromText('point(38 60)',4326)),
(2001, "Calculator", ST_GeomFromText('point(60 -24)',4326)),
(2001, "Computer" , ST_GeomFromText('point(28 -77)',4326));
```

データセットに対して `ST_Collect()` を使用するクエリーの例を次に示します:

```
mysql> SELECT ST_AsText(ST_Collect(location)) AS result
FROM product;
+-----+
| result |
+-----+
| MULTIPOINT((60 -24),(28 -77),(28 -77),(38 60),(60 -24),(28 -77)) |
+-----+

mysql> SELECT ST_AsText(ST_Collect(DISTINCT location)) AS result
FROM product;
+-----+
| result |
+-----+
| MULTIPOINT((60 -24),(28 -77),(38 60)) |
+-----+

mysql> SELECT year, ST_AsText(ST_Collect(location)) AS result
FROM product GROUP BY year;
+----+-----+
| year | result |
+----+-----+
| 2000 | MULTIPOINT((60 -24),(28 -77),(28 -77),(38 60)) |
| 2001 | MULTIPOINT((60 -24),(28 -77)) |
+----+-----+

mysql> SELECT year, ST_AsText(ST_Collect(DISTINCT location)) AS result
FROM product GROUP BY year;
+----+-----+
| year | result |
+----+-----+
| 2000 | MULTIPOINT((60 -24),(28 -77),(38 60)) |
| 2001 | MULTIPOINT((60 -24),(28 -77)) |
+----+-----+

# selects nothing
mysql> SELECT ST_Collect(location) AS result
FROM product WHERE year = 1999;
+-----+
| result |
+-----+
| NULL |
+-----+

mysql> SELECT ST_AsText(ST_Collect(location)
OVER (ORDER BY year, product ROWS BETWEEN 1 PRECEDING AND CURRENT ROW))
AS result
FROM product;
+-----+
| result |
+-----+
| MULTIPOINT((28 -77)) |
| MULTIPOINT((28 -77),(60 -24)) |
| MULTIPOINT((60 -24),(28 -77)) |
| MULTIPOINT((28 -77),(38 60)) |
```

```
| MULTIPOINT((38 60),(60 -24)) |
| MULTIPOINT((60 -24),(28 -77)) |
+-----+
+-----+
```

この関数は、MySQL 8.0.24 で追加されました。

12.17.13 空間の便利な関数

このセクションの関数は、ジオメトリ値に対する便利な操作を提供します。

特に指定がないかぎり、このセクションの関数はジオメトリ引数を次のように処理します:

- いずれかの引数が `NULL` の場合、戻り値は `NULL` です。
- ジオメトリ引数が構文的に整形形式のジオメトリでない場合は、`ER_GIS_INVALID_DATA` エラーが発生します。
- 未定義の空間参照システム (SRS) でジオメトリ引数が構文的に整形形式のジオメトリである場合、`ER_SRS_NOT_FOUND` エラーが発生します。
- 複数のジオメトリ引数を取る関数では、それらの引数が同じ SRS 内でない場合、`ER_GIS_DIFFERENT_SRIDS` エラーが発生します。
- それ以外の場合、戻り値は `NULL` 以外です。

次の便利な関数を使用できます:

- `ST_Distance_Sphere(g1, g2 [, radius])`

スフィア上の `Point` または `MultiPoint` 引数間の最小球面距離をメートル単位で返します。(汎用の距離計算については、`ST_Distance()` 関数を参照してください。) オプションの `radius` 引数はメートル単位で指定する必要があります。

両方のジオメトリパラメータが SRID 0 の有効なデカルト `Point` 値または `MultiPoint` 値である場合、戻り値は指定された半径を持つ球上の 2 つのジオメトリ間の最短距離です。省略すると、デフォルトの半径は 6,370,986 メートル、点 X および Y 座標はそれぞれ経度および緯度 (度) として解釈されます。

地理空間参照システム (SRS) で両方のジオメトリパラメータが有効な `Point` 値または `MultiPoint` 値である場合、戻り値は球上の 2 つのジオメトリ間の最短距離となり、半径が指定されます。省略した場合、デフォルトの半径は $(2a+b)/3$ 、として定義された平均半径と等しくなります。a は半主軸、b は SRS の半最小軸です。

`ST_Distance_Sphere()` は、このセクションの概要で説明されているように、引数を処理しますが、次の例外があります:

- サポートされているジオメトリ引数の組合せは、`Point` と `Point`、または `Point` と `MultiPoint` (任意の引数の順序) です。少なくともいずれかのジオメトリが `Point` でも `MultiPoint` でもなく、SRID が 0 の場合、`ER_NOT_IMPLEMENTED_FOR_CARTESIAN_SRS` エラーが発生します。少なくともいずれかのジオメトリが `Point` でも `MultiPoint` でもなく、SRID が地理 SRS を参照している場合は、`ER_NOT_IMPLEMENTED_FOR_GEOGRAPHIC_SRS` エラーが発生します。ジオメトリが投影 SRS を参照している場合、`ER_NOT_IMPLEMENTED_FOR_PROJECTED_SRS` エラーが発生します。
- 範囲外の経度または緯度を持つ引数がある場合、エラーが発生します:
 - 経度の値が $(-180, 180]$ の範囲内でない場合は、`ER_GEOMETRY_PARAM_LONGITUDE_OUT_OF_RANGE` エラーが発生します (MySQL 8.0.12 より前の `ER_LONGITUDE_OUT_OF_RANGE`)。
 - 緯度の値が $[-90, 90]$ の範囲内でない場合は、`ER_GEOMETRY_PARAM_LATITUDE_OUT_OF_RANGE` エラーが発生します (MySQL 8.0.12 より前の `ER_LATITUDE_OUT_OF_RANGE`)。

表示される範囲は度数です。SRS で別の単位が使用されている場合、範囲ではその単位に対応する値が使用されず、浮動小数点演算のため、正確な範囲制限はわずかに偏差します。

- `radius` 引数が存在するが正でない場合は、`ER_NONPOSITIVE_RADIUS` エラーが発生します。
- 距離が倍精度数値の範囲を超えると、`ER_STD_OVERFLOW_ERROR` エラーが発生します。

```
mysql> SET @pt1 = ST_GeomFromText('POINT(0 0)');
mysql> SET @pt2 = ST_GeomFromText('POINT(180 0)');
mysql> SELECT ST_Distance_Sphere(@pt1, @pt2);
+-----+
| ST_Distance_Sphere(@pt1, @pt2) |
+-----+
|          20015042.813723423 |
+-----+
```

• ST_IsValid(g)

引数がジオメトリ学的に有効な場合は 1 を返し、ジオメトリ学的に有効でない場合は 0 を返します。ジオメトリの有効性は OGC 仕様で定義されます。

有効な空のジオメトリのみが、空のジオメトリコレクション値の形式で表されます。この場合、`ST_IsValid()` は 1 を返します。MySQL は、`POINT EMPTY` などの GIS の `EMPTY` 値をサポートしていません。

`ST_IsValid()` は、このセクションの概要で説明されているように引数を処理しますが、次の例外があります:

- ジオメトリに経度または緯度が範囲外の地理 SRS がある場合、エラーが発生します:
 - 経度の値が $(-180, 180]$ の範囲内でない場合は、`ER_GEOMETRY_PARAM_LONGITUDE_OUT_OF_RANGE` エラーが発生します (MySQL 8.0.12 より前の `ER_LONGITUDE_OUT_OF_RANGE`)。
 - 緯度の値が $(-90, 90]$ の範囲内でない場合は、`ER_GEOMETRY_PARAM_LATITUDE_OUT_OF_RANGE` エラーが発生します (MySQL 8.0.12 より前の `ER_LATITUDE_OUT_OF_RANGE`)。

表示される範囲は度数です。SRS で別の単位が使用されている場合、範囲ではその単位に対応する値が使用されます。浮動小数点演算のため、正確な範囲制限はわずかに偏差します。

```
mysql> SET @ls1 = ST_GeomFromText('LINESTRING(0 0,-0.00 0,0.0 0)');
mysql> SET @ls2 = ST_GeomFromText('LINESTRING(0 0, 1 1)');
mysql> SELECT ST_IsValid(@ls1);
+-----+
| ST_IsValid(@ls1) |
+-----+
|          0 |
+-----+
mysql> SELECT ST_IsValid(@ls2);
+-----+
| ST_IsValid(@ls2) |
+-----+
|          1 |
+-----+
```

• ST_MakeEnvelope(pt1, pt2)

`Point`、`LineString` または `Polygon` として、2 つの点を中心にエンベロープを形成する矩形を返します。

計算は、球、回転楕円体、または地球ではなくデカルト座標系を使用して行われます。

`pt1` と `pt2` の 2 つの点を指定すると、`ST_MakeEnvelope()` は次のような抽象平面上に結果ジオメトリを作成します:

- `pt1` と `pt2` が等しい場合、結果はポイント `pt1` になります。
- それ以外の場合、`(pt1, pt2)` が垂直または水平線セグメントの場合、結果は線セグメント `(pt1, pt2)` になります。
- それ以外の場合、結果は `pt1` および `pt2` を対角ポイントとして使用するポリゴンになります。

結果のジオメトリの SRID は 0 です。

`ST_MakeEnvelope()` は、このセクションの概要で説明されているように、引数を処理しますが、次の例外があります:

- 引数が `Point` 値でない場合は、`ER_WRONG_ARGUMENTS` エラーが発生します。
- 2 つの点の座標値が無限または `NaN` であるという追加条件で、`ER_GIS_INVALID_DATA` エラーが発生します。

- 地理空間参照システム (SRS) の SRID 値を持つジオメトリがある場合、`ER_NOT_IMPLEMENTED_FOR_GEOGRAPHIC_SRS` エラーが発生します。

```
mysql> SET @pt1 = ST_GeomFromText('POINT(0 0)');
mysql> SET @pt2 = ST_GeomFromText('POINT(1 1)');
mysql> SELECT ST_AsText(ST_MakeEnvelope(@pt1, @pt2));
+-----+
| ST_AsText(ST_MakeEnvelope(@pt1, @pt2)) |
+-----+
| POLYGON((0 0,1 0,1 1,0 1,0 0))      |
+-----+
```

- `ST_Simplify(g, max_distance)`

Douglas-Peucker アルゴリズムを使用してジオメトリを簡略化し、同じタイプの簡略化された値を返します。

ジオメトリは任意のジオメトリタイプにできますが、Douglas-Peucker アルゴリズムは実際にはすべてのタイプを処理するわけではありません。ジオメトリコレクションは、そのコンポーネントに 1 つずつ簡略化アルゴリズムを指定することで処理され、戻されたジオメトリは結果としてジオメトリコレクションに格納されます。

`max_distance` 引数は、削除する頂点から他のセグメントまでの距離 (入力座標の単位) です。簡略化された線ストリングのこの距離内の頂点が削除されます。

Boost.Geometry によると、ジオメトリは簡略化プロセスの結果として無効になり、プロセスによって自己交差が作成される可能性があります。結果の妥当性をチェックするには、`ST_IsValid()` に渡します。

`ST_Simplify()` は、このセクションの概要で説明されているように引数を処理しますが、次の例外があります:

- `max_distance` 引数が正でないか、NaN である場合、`ER_WRONG_ARGUMENTS` エラーが発生します。

```
mysql> SET @g = ST_GeomFromText('LINESTRING(0 0,0 1,1 1,1 2,2 2,2 3,3 3)');
mysql> SELECT ST_AsText(ST_Simplify(@g, 0.5));
+-----+
| ST_AsText(ST_Simplify(@g, 0.5)) |
+-----+
| LINESTRING(0 0,0 1,1 1,2 2,3 3) |
+-----+
mysql> SELECT ST_AsText(ST_Simplify(@g, 1.0));
+-----+
| ST_AsText(ST_Simplify(@g, 1.0)) |
+-----+
| LINESTRING(0 0,3 3)              |
+-----+
```


- [ST_Validate\(g\)](#)

OGC 仕様に従ってジオメトリを検証します。ジオメトリは構文的に整形形式 (WKB 値と SRID) にすることができませんが、ジオメトリ学的には無効です。たとえば、このポリゴンはジオメトリ学的に無効です: `POLYGON((0 0, 0 0, 0 0, 0 0, 0 0))`

`ST_Validate()` は、ジオメトリが構文的に整形形式でジオメトリが有効な場合はジオメトリを戻し、引数が構文的に整形形式でないか、形状的に有効でないか、または `NULL` である場合は `NULL` を戻します。

`ST_Validate()` を使用すると、コストがかかりますが、無効なジオメトリデータをフィルタ処理で除外できます。無効なデータによって保持されないより正確な結果を必要とするアプリケーションの場合、このペナルティは価値がある可能性があります。

ジオメトリ引数が有効な場合、入力 `Polygon` または `MultiPolygon` に時計回りのリングがある場合を除き、それらのリングは有効性をチェックする前に逆になります。ジオメトリが有効な場合は、反転リングの値が返されます。

有効な空のジオメトリのみが、空のジオメトリコレクション値の形式で表されます。この場合、`ST_Validate()` はそれを直接返し、それ以上のチェックは行いません。

MySQL 8.0.13 では、`ST_Validate()` はこのセクションの概要で説明されているように引数を処理しますが、次の例外があります:

- ジオメトリに経度または緯度が範囲外の地理 SRS がある場合、エラーが発生します:
 - 経度の値が `[-180, 180]` の範囲内でない場合は、`ER_GEOOMETRY_PARAM_LONGITUDE_OUT_OF_RANGE` エラーが発生します (MySQL 8.0.12 より前の `ER_LONGITUDE_OUT_OF_RANGE`)。
 - 緯度の値が `[-90, 90]` の範囲内でない場合は、`ER_GEOOMETRY_PARAM_LATITUDE_OUT_OF_RANGE` エラーが発生します (MySQL 8.0.12 より前の `ER_LATITUDE_OUT_OF_RANGE`)。

表示される範囲は度数です。浮動小数点演算のため、正確な範囲制限はわずかに偏差します。

MySQL 8.0.13 より前では、`ST_Validate()` はこのセクションの概要で説明されているように引数を処理しますが、次の例外があります:

- ジオメトリが構文的に整形形式でない場合、戻り値は `NULL` です。 `ER_GIS_INVALID_DATA` エラーは発生しません。
- ジオメトリに地理空間参照システム (SRS) の SRID 値がある場合、`ER_NOT_IMPLEMENTED_FOR_GEOGRAPHIC_SRS` エラーが発生します。

```
mysql> SET @ls1 = ST_GeomFromText('LINESTRING(0 0)');
mysql> SET @ls2 = ST_GeomFromText('LINESTRING(0 0, 1 1)');
mysql> SELECT ST_AsText(ST_Validate(@ls1));
+-----+
| ST_AsText(ST_Validate(@ls1)) |
+-----+
| NULL                          |
+-----+
mysql> SELECT ST_AsText(ST_Validate(@ls2));
+-----+
| ST_AsText(ST_Validate(@ls2)) |
+-----+
| LINESTRING(0 0,1 1)          |
+-----+
```

12.18 JSON 関数

このセクションで説明する関数は、JSON 値に対する操作を実行します。JSON データ型の説明およびこれらの関数の使用方法を示すその他の例は、[セクション11.5「JSON データ型」](#) を参照してください。

JSON 引数を取る関数の場合、引数が有効な JSON 値でないとエラーが発生します。JSON として解析された引数は `json_doc` によって示され、`val` によって示された引数は解析されません。

GeoJSON 値を操作するための一連の空間関数も使用できます。 [セクション12.17.11「空間 GeoJSON 関数」](#) を参照してください。

12.18.1 JSON 関数リファレンス

表 12.22 「JSON 関数」

名前	説明	導入	非推奨
->	パスを評価した後の JSON カラムからの戻り値 (JSON_EXTRACT() と同等)。		
-->	パスを評価して結果を引用符で囲まずに JSON カラムから値を返します (JSON_UNQUOTE(JSON_EXTRACT()) と同等)。		
JSON_ARRAY()	JSON 配列の作成		
JSON_ARRAY_APPEND()	JSON ドキュメントへのデータの追加		
JSON_ARRAY_INSERT()	JSON 配列に挿入		
JSON_CONTAINS()	JSON ドキュメントのパスに特定のオブジェクトが含まれているかどうか		
JSON_CONTAINS_PATH()	JSON ドキュメントにパスのデータが含まれているかどうか		
JSON_DEPTH()	JSON ドキュメントの最大深度		
JSON_EXTRACT()	JSON ドキュメントからデータを返します		
JSON_INSERT()	JSON ドキュメントへのデータの挿入		
JSON_KEYS()	JSON ドキュメントからのキーの配列		
JSON_LENGTH()	JSON ドキュメント内の要素数		
JSON_MERGE()	JSON ドキュメントをマージし、重複するキーを保持します。JSON_MERGE_PRESERVE() の非推奨シノニム		はい
JSON_MERGE_PATCH()	重複キーの値を置換して JSON ドキュメントをマージ		
JSON_MERGE_PRESERVE()	重複キーを保持した JSON ドキュメントのマージ		
JSON_OBJECT()	JSON オブジェクトの作成		
JSON_OVERLAPS()	2 つの JSON ドキュメントを比較し、共通のキーと値のペアまたは配列要素がある場合は TRUE (1) を	8.0.17	

名前	説明	導入	非推奨
	戻し、それ以外の場合は FALSE (0) を戻します		
JSON_PRETTY()	JSON ドキュメントを人間が読める形式で出力		
JSON_QUOTE()	見積 JSON 文書		
JSON_REMOVE()	JSON ドキュメントからのデータの削除		
JSON_REPLACE()	JSON ドキュメントの値の置換		
JSON_SCHEMA_VALID()	JSON スキーマに対して JSON ドキュメントを検証します。ドキュメントがスキーマに対して検証される場合は TRUE/1 を返し、検証されない場合は FALSE/0 を返します	8.0.17	
JSON_SCHEMA_VALIDATION_ERROR()	JSON スキーマに対して JSON ドキュメントを検証します。成功や失敗、失敗の理由など、検証の結果に関するレポートを JSON 形式で返します	8.0.17	
JSON_SEARCH()	JSON ドキュメント内の値へのパス		
JSON_SET()	JSON ドキュメントへのデータの挿入		
JSON_STORAGE_FREE()	部分更新後の JSON カラム値のバイナリ表現内の空き領域		
JSON_STORAGE_SIZE()	JSON ドキュメントのバイナリ表現の格納に使用される領域		
JSON_TABLE()	JSON 式からリレーショナルテーブルとしてデータを返します		
JSON_TYPE()	JSON 値のタイプ		
JSON_UNQUOTE()	JSON 値の引用符なし		
JSON_VALID()	JSON 値が有効かどうか		
JSON_VALUE()	指定されたパスでポイントされた場所にある JSON ドキュメントから値を抽出します。この値を VARCHAR(512) または指定されたタイプとして返します	8.0.21	
MEMBER OF()	最初のオペランドが 2 番目のオペランドとして渡された JSON 配列のいずれかの要素と一致する場合は true (1) を返し、それ以外の場合は false (0) を返します	8.0.17	

MySQL は、[JSON_ARRAYAGG\(\)](#) および [JSON_OBJECTAGG\(\)](#) の 2 つの集計 JSON 関数をサポートしています。これらの詳細は、[セクション12.20「集計関数」](#) を参照してください。

MySQL では、[JSON_PRETTY\(\)](#) 関数を使用して、読みやすい形式の JSON 値の「pretty-printing」もサポートしています。[JSON_STORAGE_SIZE\(\)](#) および [JSON_STORAGE_FREE\(\)](#) をそれぞれ使用して、特定の JSON 値が占有するストレージ領域の量、および追加のストレージ用に残っている領域の量を確認できます。これらの関数の詳細は、[セクション12.18.8「JSON ユーティリティ関数」](#) を参照してください。

12.18.2 JSON 値を作成する関数

このセクションにリストされている関数は、コンポーネント要素から JSON 値を構成します。

- [JSON_ARRAY\(\[val\[, val\] ...\]\)](#)

値リスト (空の場合もある) を評価し、それらの値を含む JSON 配列を返します。

```
mysql> SELECT JSON_ARRAY(1, "abc", NULL, TRUE, CURTIME());
+-----+
| JSON_ARRAY(1, "abc", NULL, TRUE, CURTIME()) |
+-----+
| [1, "abc", null, true, "11:30:24.000000"] |
+-----+
```

- [JSON_OBJECT\(\[key, val\[, key, val\] ...\]\)](#)

キーと値のペアの (空の可能性もある) リストを評価し、それらのペアを含む JSON オブジェクトを返します。いずれかのキー名が `NULL` であるか、引数の数が奇数の場合、エラーが発生します。

```
mysql> SELECT JSON_OBJECT('id', 87, 'name', 'carrot');
+-----+
| JSON_OBJECT('id', 87, 'name', 'carrot') |
+-----+
| {"id": 87, "name": "carrot"} |
+-----+
```

- [JSON_QUOTE\(string\)](#)

文字列を二重引用符で囲み、内部引用符やその他の文字をエスケープして JSON 値として引用符で囲み、結果を `utf8mb4` 文字列として返します。引数が `NULL` の場合、`NULL` を返します。

この関数は通常、JSON ドキュメントに含める有効な JSON 文字列リテラルを生成するために使用されます。

特定の特殊文字は、[表12.23「JSON_UNQUOTE\(\) 特殊文字エスケープシーケンス」](#) に示されているエスケープシーケンスごとにバックスラッシュでエスケープされます。

```
mysql> SELECT JSON_QUOTE('null'), JSON_QUOTE("null");
+-----+
| JSON_QUOTE('null') | JSON_QUOTE("null") |
+-----+
| "null" | "\"null\"" |
+-----+
mysql> SELECT JSON_QUOTE('[1, 2, 3]');
+-----+
| JSON_QUOTE('[1, 2, 3]') |
+-----+
| "[1, 2, 3]" |
+-----+
```

[CAST\(value AS JSON\)](#) を使用して他の型の値を `JSON` 型にキャストすることで、JSON 値を取得することもできます。詳細は、[JSON 値と非 JSON 値の間の変換](#) を参照してください。

JSON 値を生成する 2 つの集計関数を使用できます。[JSON_ARRAYAGG\(\)](#) は結果セットを単一の JSON 配列として返し、[JSON_OBJECTAGG\(\)](#) は結果セットを単一の JSON オブジェクトとして返します。詳細は、[セクション12.20「集計関数」](#) を参照してください。

12.18.3 JSON 値を検索する関数

このセクションの関数では、JSON 値に対して検索または比較操作を実行して、JSON 値からデータを抽出したり、データが JSON 値内の場所に存在するかどうかをレポートしたり、JSON 値内のデータへのパスをレポートします。ここでは、[MEMBER OF\(\)](#) 演算子についても説明します。

- [JSON_CONTAINS\(target, candidate\[, path\]\)](#)

特定の [candidate](#) JSON ドキュメントが [target](#) JSON ドキュメント内に含まれているかどうか、または [path](#) 引数が指定されているかどうか (候補がターゲット内の特定のパスで見つかったかどうか) を 1 または 0 を返して示します。いずれかの引数が [NULL](#) の場合、またはパス引数がターゲットドキュメントのセクションを識別しない場合、[NULL](#) を返します。 [target](#) または [candidate](#) が有効な JSON ドキュメントでない場合、または [path](#) 引数が有効なパス式でないか、* または ** ワイルドカードが含まれている場合は、エラーが発生します。

パスにデータが存在するかどうかのみを確認するには、かわりに [JSON_CONTAINS_PATH\(\)](#) を使用します。

次のルールは包含を定義します:

- 候補スカラーは、比較可能で等しい場合にのみターゲットスカラーに含まれます。同じ [JSON_TYPE\(\)](#) 型を持つ場合、2 つのスカラー値が比較可能ですが、[INTEGER](#) 型と [DECIMAL](#) 型の値も相互に比較可能である点が異なります。
- 候補配列は、候補のすべての要素がターゲットの一部の要素に含まれている場合にのみ、ターゲット配列に含まれます。
- 候補がターゲットの一部の要素に含まれている場合にのみ、候補の非配列がターゲット配列に含まれます。
- 候補オブジェクトがターゲットオブジェクトに含まれるのは、候補の各キーに対して同じ名前のキーがターゲットにあり、候補キーに関連付けられた値がターゲットキーに関連付けられた値に含まれている場合のみです。

それ以外の場合、候補値はターゲットドキュメントに含まれません。

MySQL 8.0.17 以降、[InnoDB](#) テーブルで [JSON_CONTAINS\(\)](#) を使用するクエリーは、複数値インデックスを使用して最適化できます。詳細は、[複数値インデックス](#) を参照してください。

```
mysql> SET @j = '{"a": 1, "b": 2, "c": {"d": 4}}';
mysql> SET @j2 = '1';
mysql> SELECT JSON_CONTAINS(@j, @j2, '$.a');
+-----+
| JSON_CONTAINS(@j, @j2, '$.a') |
+-----+
| 1 |
+-----+
mysql> SELECT JSON_CONTAINS(@j, @j2, '$.b');
+-----+
| JSON_CONTAINS(@j, @j2, '$.b') |
+-----+
| 0 |
+-----+
mysql> SET @j2 = '{"d": 4}';
mysql> SELECT JSON_CONTAINS(@j, @j2, '$.a');
+-----+
| JSON_CONTAINS(@j, @j2, '$.a') |
+-----+
| 0 |
+-----+
mysql> SELECT JSON_CONTAINS(@j, @j2, '$.c');
+-----+
| JSON_CONTAINS(@j, @j2, '$.c') |
+-----+
| 1 |
+-----+
```

- [JSON_CONTAINS_PATH\(json_doc, one_or_all, path\[, path\] ...\)](#)

JSON ドキュメントに指定されたパスのデータが含まれているかどうかを示す 0 または 1 を返します。引数のいずれかが `NULL` である場合は、`NULL` を返します。 `json_doc` 引数が有効な JSON ドキュメントでない場合、 `path` 引数が有効なパス式でない場合、または `one_or_all` が `'one'` または `'all'` でない場合は、エラーが発生します。

パスで特定の値を確認するには、かわりに `JSON_CONTAINS()` を使用します。

指定されたパスがドキュメント内に存在しない場合、戻り値は 0 です。それ以外の場合、戻り値は `one_or_all` 引数によって異なります:

- `'one'`: ドキュメント内に少なくとも 1 つのパスが存在する場合は 1、それ以外の場合は 0。
- `'all'`: ドキュメント内にすべてのパスが存在する場合は 1、それ以外の場合は 0。

```
mysql> SET @j = '{"a": 1, "b": 2, "c": {"d": 4}}';
mysql> SELECT JSON_CONTAINS_PATH(@j, 'one', '$.a', '$.e');
+-----+
| JSON_CONTAINS_PATH(@j, 'one', '$.a', '$.e') |
+-----+
| 1 |
+-----+
mysql> SELECT JSON_CONTAINS_PATH(@j, 'all', '$.a', '$.e');
+-----+
| JSON_CONTAINS_PATH(@j, 'all', '$.a', '$.e') |
+-----+
| 0 |
+-----+
mysql> SELECT JSON_CONTAINS_PATH(@j, 'one', '$.c.d');
+-----+
| JSON_CONTAINS_PATH(@j, 'one', '$.c.d') |
+-----+
| 1 |
+-----+
mysql> SELECT JSON_CONTAINS_PATH(@j, 'one', '$.a.d');
+-----+
| JSON_CONTAINS_PATH(@j, 'one', '$.a.d') |
+-----+
| 0 |
+-----+
```

- `JSON_EXTRACT(json_doc, path[, path] ...)`

`path` 引数に一致するドキュメントの一部から選択された JSON ドキュメントからデータを返します。いずれかの引数が `NULL` の場合、またはドキュメント内の値を特定するパスがない場合は、`NULL` を返します。 `json_doc` 引数が有効な JSON ドキュメントでない場合、または `path` 引数が有効なパス式でない場合は、エラーが発生します。

戻り値は、`path` 引数に一致するすべての値で構成されます。これらの引数が複数の値を返す可能性がある場合、一致した値は、それらを生成したパスに対応する順序で配列として自動ラップされます。それ以外の場合、戻り値は単一の一致値です。

```
mysql> SELECT JSON_EXTRACT('[10, 20, [30, 40]]', '$[1]');
+-----+
| JSON_EXTRACT('[10, 20, [30, 40]]', '$[1]') |
+-----+
| 20 |
+-----+
mysql> SELECT JSON_EXTRACT('[10, 20, [30, 40]]', '$[1]', '$[0]');
+-----+
| JSON_EXTRACT('[10, 20, [30, 40]]', '$[1]', '$[0]') |
+-----+
| [20, 10] |
+-----+
mysql> SELECT JSON_EXTRACT('[10, 20, [30, 40]]', '$[2][*]');
+-----+
| JSON_EXTRACT('[10, 20, [30, 40]]', '$[2][*]') |
+-----+
| [30, 40] |
+-----+
```


MySQL では、この関数の短縮形として `->` 演算子がサポートされており、左側が `JSON` カラム識別子 (式ではなく) であり、右側がカラム内で照合される `JSON` パスである 2 つの引数で使用されます。

- `column->path`

`->` 演算子は、左側のカラム識別子と `JSON` ドキュメント (カラム値) に対して評価される右側の `JSON` パスの 2 つの引数で使用される `JSON_EXTRACT()` 関数のエイリアスとして機能します。このような式は、`SQL` ステートメントのどこにあるかにかかわらず、カラム識別子のかわりに使用できます。

次に示す 2 つの `SELECT` ステートメントでは、同じ出力が生成されます:

```
mysql> SELECT c, JSON_EXTRACT(c, "$.id"), g
> FROM jemp
> WHERE JSON_EXTRACT(c, "$.id") > 1
> ORDER BY JSON_EXTRACT(c, "$.name");
```

c	JSON_EXTRACT(c, "\$.id")	g
{ "id": "3", "name": "Barney" }	"3"	3
{ "id": "4", "name": "Betty" }	"4"	4
{ "id": "2", "name": "Wilma" }	"2"	2

3 rows in set (0.00 sec)

```
mysql> SELECT c, c->"$.id", g
> FROM jemp
> WHERE c->"$.id" > 1
> ORDER BY c->"$.name";
```

c	c->"\$.id"	g
{ "id": "3", "name": "Barney" }	"3"	3
{ "id": "4", "name": "Betty" }	"4"	4
{ "id": "2", "name": "Wilma" }	"2"	2

3 rows in set (0.00 sec)

次に示すように、この機能は `SELECT` に限定されません:

```
mysql> ALTER TABLE jemp ADD COLUMN n INT;
Query OK, 0 rows affected (0.68 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

```
mysql> UPDATE jemp SET n=1 WHERE c->"$.id" = "4";
Query OK, 1 row affected (0.04 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

```
mysql> SELECT c, c->"$.id", g, n
> FROM jemp
> WHERE JSON_EXTRACT(c, "$.id") > 1
> ORDER BY c->"$.name";
```

c	JSON_EXTRACT(c, "\$.id")	g	n
{ "id": "3", "name": "Barney" }	"3"	3	NULL
{ "id": "4", "name": "Betty" }	"4"	4	1
{ "id": "2", "name": "Wilma" }	"2"	2	NULL

3 rows in set (0.00 sec)

```
mysql> DELETE FROM jemp WHERE c->"$.id" = "4";
Query OK, 1 row affected (0.04 sec)
```

```
mysql> SELECT c, c->"$.id", g, n
> FROM jemp
> WHERE JSON_EXTRACT(c, "$.id") > 1
> ORDER BY c->"$.name";
```

c	JSON_EXTRACT(c, "\$.id")	g	n
{ "id": "3", "name": "Barney" }	"3"	3	NULL
{ "id": "2", "name": "Wilma" }	"2"	2	NULL

```

|{"id": "3", "name": "Barney"} | "3" | 3 | NULL |
|{"id": "2", "name": "Wilma"} | "2" | 2 | NULL |
+-----+-----+-----+
2 rows in set (0.00 sec)

```

(前述のテーブルの作成および移入に使用されるステートメントについては、[JSON カラムインデックスを提供するための生成されたカラムのインデックス付け](#)を参照してください。)

これは、次に示すように JSON 配列値でも機能します:

```

mysql> CREATE TABLE tj10 (a JSON, b INT);
Query OK, 0 rows affected (0.26 sec)

mysql> INSERT INTO tj10
> VALUES ("[3,10,5,17,44]", 33), ("[3,10,5,17,[22,44,66]]", 0);
Query OK, 1 row affected (0.04 sec)

mysql> SELECT a->"$[4]" FROM tj10;
+-----+
| a->"$[4]" |
+-----+
| 44        |
| [22, 44, 66] |
+-----+
2 rows in set (0.00 sec)

mysql> SELECT * FROM tj10 WHERE a->"$[0]" = 3;
+-----+-----+
| a                | b |
+-----+-----+
| [3, 10, 5, 17, 44] | 33 |
| [3, 10, 5, 17, [22, 44, 66]] | 0 |
+-----+-----+
2 rows in set (0.00 sec)

```

ネストされた配列がサポートされています。次に示すように、ターゲット JSON ドキュメントに一致するキーが見つからない場合、-> を使用する式は NULL として評価されます:

```

mysql> SELECT * FROM tj10 WHERE a->"$[4][1]" IS NOT NULL;
+-----+-----+
| a                | b |
+-----+-----+
| [3, 10, 5, 17, [22, 44, 66]] | 0 |
+-----+-----+

mysql> SELECT a->"$[4][1]" FROM tj10;
+-----+
| a->"$[4][1]" |
+-----+
| NULL        |
| 44          |
+-----+
2 rows in set (0.00 sec)

```

これは、JSON_EXTRACT() を使用している場合と同じ動作です:

```

mysql> SELECT JSON_EXTRACT(a, "$[4][1]") FROM tj10;
+-----+
| JSON_EXTRACT(a, "$[4][1]") |
+-----+
| NULL                        |
| 44                          |
+-----+
2 rows in set (0.00 sec)

```

- column->>path

これは、改善された引用符で囲まれていない抽出演算子です。-> 演算子は単に値を抽出するだけですが、->> 演算子は抽出された結果を引用符で囲みません。つまり、JSON カラム値が column で、パス式が path の場合、次の3つの式は同じ値を返します:

- JSON_UNQUOTE(JSON_EXTRACT(column, path))
- JSON_UNQUOTE(column -> path)
- column->>path

->> 演算子は、JSON_UNQUOTE(JSON_EXTRACT()) が許可される場所であればどこでも使用できます。これには、SELECT リスト、WHERE 句と HAVING 句、ORDER BY 句と GROUP BY 句が含まれます (これらに限定されません)。

次のいくつかのステートメントは、mysql クライアントの他の式と同等の ->> 演算子を示しています:

```
mysql> SELECT * FROM jemp WHERE g > 2;
+-----+-----+
| c          | g |
+-----+-----+
| {"id": "3", "name": "Barney"} | 3 |
| {"id": "4", "name": "Betty"} | 4 |
+-----+-----+
2 rows in set (0.01 sec)

mysql> SELECT c->'$.name' AS name
-> FROM jemp WHERE g > 2;
+-----+
| name |
+-----+
| "Barney" |
| "Betty" |
+-----+
2 rows in set (0.00 sec)

mysql> SELECT JSON_UNQUOTE(c->'$.name') AS name
-> FROM jemp WHERE g > 2;
+-----+
| name |
+-----+
| Barney |
| Betty |
+-----+
2 rows in set (0.00 sec)

mysql> SELECT c->>'$.name' AS name
-> FROM jemp WHERE g > 2;
+-----+
| name |
+-----+
| Barney |
| Betty |
+-----+
2 rows in set (0.00 sec)
```

前述の一連の例で jemp テーブルの作成および移入に使用される SQL ステートメントについては、JSON カラムインデックスを提供するための生成されたカラムのインデックス付けを参照してください。

この演算子は、次に示すように JSON 配列でも使用できます:

```
mysql> CREATE TABLE tj10 (a JSON, b INT);
Query OK, 0 rows affected (0.26 sec)

mysql> INSERT INTO tj10 VALUES
-> ([3,10,5,"x",44], 33),
-> ([3,10,5,17,[22,"y",66]], 0);
Query OK, 2 rows affected (0.04 sec)
Records: 2 Duplicates: 0 Warnings: 0
```

```
mysql> SELECT a->"$[3]", a->"$[4][1]" FROM tj10;
+-----+-----+
| a->"$[3]" | a->"$[4][1]" |
+-----+-----+
| "x"      | NULL        |
| 17      | "y"         |
+-----+-----+
2 rows in set (0.00 sec)

mysql> SELECT a->>"$[3]", a->>"$[4][1]" FROM tj10;
+-----+-----+
| a->>"$[3]" | a->>"$[4][1]" |
+-----+-----+
| x         | NULL         |
| 17        | y            |
+-----+-----+
2 rows in set (0.00 sec)
```

-> と同様に、次の例に示すように、->> 演算子は常に EXPLAIN の出力で展開されます:

```
mysql> EXPLAIN SELECT c->>'$.name' AS name
-> FROM jemp WHERE g > 2\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: jemp
  partitions: NULL
         type: range
possible_keys: i
         key: i
        key_len: 5
         ref: NULL
         rows: 2
    filtered: 100.00
      Extra: Using where
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Note
Code: 1003
Message: /* select#1 */ select
json_unquote(json_extract('jtest`.`jemp`.`c`, '$.name')) AS `name` from
`jtest`.`jemp` where (`jtest`.`jemp`.`g` > 2)
1 row in set (0.00 sec)
```

これは、同じ状況で MySQL が -> 演算子を拡張する方法と似ています。

- `JSON_KEYS(json_doc[, path])`

JSON オブジェクトの最上位値からキーを JSON 配列として返します。path 引数が指定されている場合は、選択されたパスの最上位キーを返します。いずれかの引数が NULL の場合、json_doc 引数がオブジェクトでない場合、または path(指定されている場合) がオブジェクトを検出しない場合に、NULL を返します。json_doc 引数が有効な JSON ドキュメントでないか、path 引数が有効なパス式でないか、* または ** ワイルドカードが含まれている場合、エラーが発生します。

選択したオブジェクトが空の場合、結果配列は空です。最上位の値にネストされたサブオブジェクトがある場合、戻り値にはそれらのサブオブジェクトのキーは含まれません。

```
mysql> SELECT JSON_KEYS('{"a": 1, "b": {"c": 30}}');
+-----+
| JSON_KEYS('{"a": 1, "b": {"c": 30}}') |
+-----+
| ["a", "b"]                            |
+-----+
mysql> SELECT JSON_KEYS('{"a": 1, "b": {"c": 30}}', '$.b');
+-----+
| JSON_KEYS('{"a": 1, "b": {"c": 30}}', '$.b') |
+-----+
| ["c"]                                     |
+-----+
```

- `JSON_OVERLAPS(json_doc1, json_doc2)`

2 つの JSON ドキュメントを比較します。2 つのドキュメントに共通のキーと値のペアまたは配列要素がある場合、`true` (1) を返します。両方の引数がスカラーの場合、この関数は単純な等価性テストを実行します。

この関数は `JSON_CONTAINS()` と同等の役割を果たします。これには、検索対象の配列のすべての要素が検索対象の配列に存在する必要があります。したがって、`JSON_CONTAINS()` は検索キーに対して `AND` 操作を実行し、`JSON_OVERLAPS()` は `OR` 操作を実行します。

`WHERE` 句で `JSON_OVERLAPS()` を使用する InnoDB テーブルの JSON カラムに対するクエリーは、複数値インデックスを使用して最適化できます。[複数値インデックス](#) では、詳細な情報と例を示します。

2 つの配列を比較する場合、`JSON_OVERLAPS()` は共通の配列要素を共有すると `true` を返し、共有しない場合は `false` を返します:

```
mysql> SELECT JSON_OVERLAPS("[1,3,5,7]", "[2,5,7]");
+-----+
| JSON_OVERLAPS("[1,3,5,7]", "[2,5,7]") |
+-----+
| 1 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT JSON_OVERLAPS("[1,3,5,7]", "[2,6,7]");
+-----+
| JSON_OVERLAPS("[1,3,5,7]", "[2,6,7]") |
+-----+
| 1 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT JSON_OVERLAPS("[1,3,5,7]", "[2,6,8]");
+-----+
| JSON_OVERLAPS("[1,3,5,7]", "[2,6,8]") |
+-----+
| 0 |
+-----+
1 row in set (0.00 sec)
```

部分一致は、次に示すように一致なしとして処理されます:

```
mysql> SELECT JSON_OVERLAPS("[[1,2],[3,4],5]", "[1,[2,3],[4,5]]");
+-----+
| JSON_OVERLAPS("[[1,2],[3,4],5]", "[1,[2,3],[4,5]]") |
+-----+
| 0 |
+-----+
1 row in set (0.00 sec)
```

オブジェクトを比較する場合、共通のキーと値のペアが 1 つ以上あると、結果は `true` になります。

```
mysql> SELECT JSON_OVERLAPS("{\"a\":1,\"b\":10,\"d\":10}", "{\"c\":1,\"e\":10,\"f\":1,\"d\":10}");
+-----+
| JSON_OVERLAPS("{\"a\":1,\"b\":10,\"d\":10}", "{\"c\":1,\"e\":10,\"f\":1,\"d\":10}") |
+-----+
| 1 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT JSON_OVERLAPS("{\"a\":1,\"b\":10,\"d\":10}", "{\"a\":5,\"e\":10,\"f\":1,\"d\":20}");
+-----+
| JSON_OVERLAPS("{\"a\":1,\"b\":10,\"d\":10}", "{\"a\":5,\"e\":10,\"f\":1,\"d\":20}") |
+-----+
| 0 |
+-----+
1 row in set (0.00 sec)
```

関数の引数として 2 つのスカラーが使用されている場合、`JSON_OVERLAPS()` は等価性の単純なテストを実行します:

```
mysql> SELECT JSON_OVERLAPS('5', '5');
```

```
+-----+
| JSON_OVERLAPS('5', '5') |
+-----+
|           1 |
+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT JSON_OVERLAPS('5', '6');
```

```
+-----+
| JSON_OVERLAPS('5', '6') |
+-----+
|           0 |
+-----+
1 row in set (0.00 sec)
```

スカラーを配列と比較する場合、`JSON_OVERLAPS()` はスカラーを配列要素として処理しようとします。この例では、次に示すように、2 番目の引数 `6` が `[6]` として解釈されます:

```
mysql> SELECT JSON_OVERLAPS('[4,5,6,7]', '6');
```

```
+-----+
| JSON_OVERLAPS('[4,5,6,7]', '6') |
+-----+
|           1 |
+-----+
1 row in set (0.00 sec)
```

この関数は型変換を実行しません:

```
mysql> SELECT JSON_OVERLAPS('[4,5,"6",7]', '6');
```

```
+-----+
| JSON_OVERLAPS('[4,5,"6",7]', '6') |
+-----+
|           0 |
+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT JSON_OVERLAPS('[4,5,6,7]', "6");
```

```
+-----+
| JSON_OVERLAPS('[4,5,6,7]', "6") |
+-----+
|           0 |
+-----+
1 row in set (0.00 sec)
```

`JSON_OVERLAPS()` が MySQL 8.0.17 に追加されました。

- `JSON_SEARCH(json_doc, one_or_all, search_str[, escape_char[, path] ...])`

JSON ドキュメント内の指定された文字列へのパスを返します。 `json_doc`、`search_str` または `path` 引数のいずれかが `NULL` の場合、ドキュメント内に `path` が存在しない場合、または `search_str` が見つからない場合は、`NULL` を返

します。 `json_doc` 引数が有効な JSON ドキュメントでない場合、 `path` 引数が有効なパス式でない場合、 `one_or_all` が `'one'` または `'all'` でない場合、または `escape_char` が定数式でない場合は、エラーが発生します。

`one_or_all` 引数は、次のように検索に影響します:

- `'one'`: 最初の一致の後に検索が終了し、1 つのパス文字列が返されます。一致が最初に考慮されるのは未定義です。
- `'all'`: 検索では、重複するパスが含まれないように、一致するすべてのパス文字列が返されます。複数の文字列がある場合は、配列として自動ラップされます。配列要素の順序が未定義です。

`search_str` 検索文字列引数内では、`%` および `_` 文字は LIKE 演算子と同様に機能: `%` は任意の数の文字 (ゼロ文字を含む) に一致し、`_` は完全に 1 文字に一致します。

検索文字列にリテラル `%` または `_` 文字を指定するには、その前にエスケープ文字を付けます。 `escape_char` 引数がない場合、または `NULL` の場合、デフォルトは `\` です。それ以外の場合、 `escape_char` は空または 1 文字の定数である必要があります。

一致およびエスケープ文字の動作の詳細は、[セクション 12.8.1 「文字列比較関数および演算子」](#) の LIKE の説明を参照してください。エスケープ文字処理の場合、LIKE の動作との違いは、`JSON_SEARCH()` のエスケープ文字は、実行時だけでなく、コンパイル時に定数に評価される必要があることです。たとえば、`JSON_SEARCH()` がブリアドステートメントで使用され、`?` パラメータを使用して `escape_char` 引数が指定されている場合、パラメータ値は実行時には一定ですが、コンパイル時には一定ではありません。

```
mysql> SET @j = '{"abc", [{"k": "10"}, "def"], {"x": "abc"}, {"y": "bcd"}}';
```

```
mysql> SELECT JSON_SEARCH(@j, 'one', 'abc');
```

```
+-----+
| JSON_SEARCH(@j, 'one', 'abc') |
+-----+
| "$[0]" |
+-----+
```

```
mysql> SELECT JSON_SEARCH(@j, 'all', 'abc');
```

```
+-----+
| JSON_SEARCH(@j, 'all', 'abc') |
+-----+
| ["$[0]", "$[2].x"] |
+-----+
```

```
mysql> SELECT JSON_SEARCH(@j, 'all', 'ghi');
```

```
+-----+
| JSON_SEARCH(@j, 'all', 'ghi') |
+-----+
| NULL |
+-----+
```

```
mysql> SELECT JSON_SEARCH(@j, 'all', '10');
```

```
+-----+
| JSON_SEARCH(@j, 'all', '10') |
+-----+
| "$[1][0].k" |
+-----+
```

```
mysql> SELECT JSON_SEARCH(@j, 'all', '10', NULL, '$');
```

```
+-----+
| JSON_SEARCH(@j, 'all', '10', NULL, '$') |
+-----+
| "$[1][0].k" |
+-----+
```

```
mysql> SELECT JSON_SEARCH(@j, 'all', '10', NULL, '$[*]');
```

```
+-----+
| JSON_SEARCH(@j, 'all', '10', NULL, '$[*']) |
+-----+
| "$[1][0].k" |
+-----+
```

```
mysql> SELECT JSON_SEARCH(@j, 'all', '10', NULL, '$*.k');
```

```
+-----+
```

```
| JSON_SEARCH(@j, 'all', '10', NULL, '$*.k') |
+-----+
| "$[1][0].k" |
+-----+

mysql> SELECT JSON_SEARCH(@j, 'all', '10', NULL, '$[*][0].k');
+-----+
| JSON_SEARCH(@j, 'all', '10', NULL, '$[*][0].k') |
+-----+
| "$[1][0].k" |
+-----+

mysql> SELECT JSON_SEARCH(@j, 'all', '10', NULL, '$[1]');
+-----+
| JSON_SEARCH(@j, 'all', '10', NULL, '$[1]') |
+-----+
| "$[1][0].k" |
+-----+

mysql> SELECT JSON_SEARCH(@j, 'all', '10', NULL, '$[1][0]');
+-----+
| JSON_SEARCH(@j, 'all', '10', NULL, '$[1][0]') |
+-----+
| "$[1][0].k" |
+-----+

mysql> SELECT JSON_SEARCH(@j, 'all', 'abc', NULL, '$[2]');
+-----+
| JSON_SEARCH(@j, 'all', 'abc', NULL, '$[2]') |
+-----+
| "$[2].x" |
+-----+

mysql> SELECT JSON_SEARCH(@j, 'all', '%a%');
+-----+
| JSON_SEARCH(@j, 'all', '%a%') |
+-----+
| ["$[0]", "$[2].x"] |
+-----+

mysql> SELECT JSON_SEARCH(@j, 'all', '%b%');
+-----+
| JSON_SEARCH(@j, 'all', '%b%') |
+-----+
| ["$[0]", "$[2].x", "$[3].y"] |
+-----+

mysql> SELECT JSON_SEARCH(@j, 'all', '%b%', NULL, '$[0]');
+-----+
| JSON_SEARCH(@j, 'all', '%b%', NULL, '$[0]') |
+-----+
| "$[0]" |
+-----+

mysql> SELECT JSON_SEARCH(@j, 'all', '%b%', NULL, '$[2]');
+-----+
| JSON_SEARCH(@j, 'all', '%b%', NULL, '$[2]') |
+-----+
| "$[2].x" |
+-----+

mysql> SELECT JSON_SEARCH(@j, 'all', '%b%', NULL, '$[1]');
+-----+
| JSON_SEARCH(@j, 'all', '%b%', NULL, '$[1]') |
+-----+
| NULL |
+-----+

mysql> SELECT JSON_SEARCH(@j, 'all', '%b%', "", '$[1]');
+-----+
| JSON_SEARCH(@j, 'all', '%b%', "", '$[1]') |
+-----+
| NULL |
+-----+
```

```
mysql> SELECT JSON_SEARCH(@j, 'all', '%b%', ", '$[3]');
+-----+
| JSON_SEARCH(@j, 'all', '%b%', ", '$[3]') |
+-----+
| "$[3].y" |
+-----+
```

ワイルドカード演算子 * および ** を制御するルールなど、MySQL でサポートされている JSON パス構文の詳細は、[JSON パス構文](#) を参照してください。

- [JSON_VALUE\(json_doc, path\)](#)

指定されたドキュメントで指定されたパスにある JSON ドキュメントから値を抽出し、抽出された値を返します。オプションで、必要なタイプに変換します。完全な構文は次のとおりです:

```
JSON_VALUE(json_doc, path [RETURNING type] [on_empty] [on_error])
```

```
on_empty:
{NULL | ERROR | DEFAULT value} ON EMPTY
```

```
on_error:
{NULL | ERROR | DEFAULT value} ON ERROR
```

`json_doc` は有効な JSON ドキュメントです。

`path` は、ドキュメント内の場所を指す JSON パスです。

`type` は、次のいずれかのデータ型です:

- [FLOAT](#)
- [DOUBLE](#)
- [DECIMAL](#)
- [SIGNED](#)
- [UNSIGNED](#)
- [DATE](#)
- [TIME](#)
- [DATETIME](#)
- [YEAR](#) (MySQL 8.0.22 以降)

1 桁または 2 桁の [YEAR](#) 値はサポートされていません。

- [CHAR](#)
- [JSON](#)

リストされている型は、[CAST\(\)](#) 関数でサポートされている (配列以外の) 型と同じです。

`RETURNING` 句で指定されていない場合、[JSON_VALUE\(\)](#) 関数の戻り型は [VARCHAR\(512\)](#) です。戻り型に文字セットが指定されていない場合、[JSON_VALUE\(\)](#) はバイナリ照合順序で [utf8mb4](#) を使用します。これは大/小文字

が区別されます。utf8mb4 が結果の文字セットとして指定されている場合、サーバーはこの文字セットのデフォルト照合順序を使用しますが、大文字と小文字は区別されません。

指定したパスのデータが JSON null リテラルで構成されているか、JSON NULL リテラルに解決されると、関数は SQL NULL を返します。

on_empty は、指定されたパスにデータが見つからない場合の JSON_VALUE() の動作を決定します。この句には、次のいずれかの値を指定します:

- **NULL ON EMPTY:** この関数は NULL を返します。これはデフォルトの ON EMPTY 動作です。
- **DEFAULT value ON EMPTY:** 指定された value が返されます。値の型は戻り値の型と一致する必要があります。
- **ERROR ON EMPTY:** この関数はエラーをスローします。

使用する場合、on_error は、次に示すように、エラーが発生したときに対応する結果とともに次のいずれかの値を取ります:

- **NULL ON ERROR: JSON_VALUE()** は NULL を返します。これは、ON ERROR 句が使用されていない場合のデフォルトの動作です。
- **DEFAULT value ON ERROR:** これは返される値です。その値は戻り型の値と一致する必要があります。
- **ERROR ON ERROR:** エラーがスローされます。

ON EMPTY を使用する場合は、ON ERROR 句の前に置く必要があります。間違った順序で指定すると、構文エラーが発生します。

エラー処理. 通常、エラーは JSON_VALUE() によって次のように処理されます:

- すべての JSON 入力 (ドキュメントおよびパス) の有効性がチェックされます。いずれかが有効でない場合、ON ERROR 句をトリガーせずに SQL エラーがスローされます。
- ON ERROR は、次のいずれかのイベントが発生するたびにトリガーされます:
 - JSON ドキュメント内の複数の場所に解決されるパスから生成されたオブジェクトまたは配列を抽出しようとしています
 - 'asdf' を UNSIGNED 値に変換しようとするなどの変換エラー
 - 値の切捨て
- NULL ON ERROR または DEFAULT ... ON ERROR が指定されている場合でも、変換エラーによって常に警告がトリガーされます。
- ON EMPTY 句は、ソース JSON ドキュメント (expr) の指定した場所 (path) にデータが含まれていない場合にトリガーされます。

JSON_VALUE() は、MySQL 8.0.21 で導入されました。

例. ここでは、2 つの簡単な例を示します:

```
mysql> SELECT JSON_VALUE({'fname': 'Joe', 'lname': 'Palmer'}, '$.fname');
+-----+
| JSON_VALUE({'fname': 'Joe', 'lname': 'Palmer'}, '$.fname') |
+-----+
| Joe |
+-----+

mysql> SELECT JSON_VALUE({'item': 'shoes', 'price': '49.95'}, '$.price'
-> RETURNING DECIMAL(4,2)) AS price;
+-----+
| price |
+-----+
| 49.95 |
```

```
+-----+
```

`SELECT JSON_VALUE(json_doc, path RETURNING type)` というステートメントは、次のステートメントと同等です:

```
SELECT CAST(
  JSON_UNQUOTE( JSON_EXTRACT(json_doc, path) )
  AS type
);
```

`JSON_VALUE()` では、多くの場合、生成されたカラムを作成してから生成されたカラムのインデックスを作成する必要がなくなるため、JSON カラムのインデックスの作成が簡略化されます。これを行うには、次に示すように、JSON カラムを含むテーブル `t1` を作成するときに、そのカラムで動作する `JSON_VALUE()` を使用する式にインデックスを作成します (そのカラムの値と一致するパスを使用):

```
CREATE TABLE t1(
  j JSON,
  INDEX i1 ( (JSON_VALUE(j, '$.id' RETURNING UNSIGNED)))
);
```

次の `EXPLAIN` 出力は、`WHERE` 句でインデックス式を使用する `t1` に対するクエリーで、作成されたインデックスが使用されることを示しています:

```
mysql> EXPLAIN SELECT * FROM t1
-> WHERE JSON_VALUE(j, '$.id' RETURNING UNSIGNED) = 123\G
***** 1. row *****
   id: 1
select_type: SIMPLE
  table: t1
 partitions: NULL
   type: ref
possible_keys: i1
   key: i1
  key_len: 9
   ref: const
   rows: 1
 filtered: 100.00
  Extra: NULL
```

これは、次のように、生成されたカラム ([JSON カラムインデックスを提供するための生成されたカラムのインデックス付け](#)を参照) にインデックスを使用してテーブル `t2` を作成するのとはほぼ同じ結果になります:

```
CREATE TABLE t2 (
  j JSON,
  g INT GENERATED ALWAYS AS (j->"$.id"),
  INDEX i1 (j)
);
```

生成されたカラムを参照する、このテーブルに対するクエリーの `EXPLAIN` 出力は、テーブル `t1` に対する前述のクエリーと同じ方法でインデックスが使用されることを示しています:

```
mysql> EXPLAIN SELECT * FROM t2 WHERE g = 123\G
***** 1. row *****
   id: 1
select_type: SIMPLE
  table: t2
 partitions: NULL
   type: ref
possible_keys: i1
   key: i1
  key_len: 5
   ref: const
   rows: 1
 filtered: 100.00
  Extra: NULL
```

生成されたカラムに対するインデックスを使用した JSON カラムの間接インデックス付けの詳細は、[JSON カラムインデックスを提供するための生成されたカラムのインデックス付け](#)を参照してください。

- `value MEMBER OF(json_array)`

`value` が `json_array` の要素である場合は `true` (1) を返し、それ以外の場合は `false` (0) を返します。`value` はスカラーまたは JSON 文書である必要があります。スカラーの場合、演算子は JSON 配列の要素として処理しようとしません。

`WHERE` 句の InnoDB テーブルの JSON カラムで `MEMBER OF()` を使用するクエリーは、複数値インデックスを使用して最適化できます。詳細および例は、[複数値インデックス](#) を参照してください。

単純なスカラーは、次に示すように配列値として扱われます:

```
mysql> SELECT 17 MEMBER OF(['23', 'abc', 17, 'ab', 10]);
+-----+
| 17 MEMBER OF(['23', 'abc', 17, 'ab', 10]) |
+-----+
|                      1 |
+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT 'ab' MEMBER OF(['23', 'abc', 17, 'ab', 10]);
+-----+
| 'ab' MEMBER OF(['23', 'abc', 17, 'ab', 10]) |
+-----+
|                      1 |
+-----+
1 row in set (0.00 sec)
```

配列要素値の部分一致が一致しません:

```
mysql> SELECT 7 MEMBER OF(['23', 'abc', 17, 'ab', 10]);
+-----+
| 7 MEMBER OF(['23', 'abc', 17, 'ab', 10]) |
+-----+
|                      0 |
+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT 'a' MEMBER OF(['23', 'abc', 17, 'ab', 10]);
+-----+
| 'a' MEMBER OF(['23', 'abc', 17, 'ab', 10]) |
+-----+
|                      0 |
+-----+
1 row in set (0.00 sec)
```

文字列型との間の変換は実行されません:

```
mysql> SELECT
-> 17 MEMBER OF(['23', 'abc', '17', 'ab', 10]),
-> '17' MEMBER OF(['23', 'abc', 17, 'ab', 10])\G
***** 1. row *****
17 MEMBER OF(['23', 'abc', '17', 'ab', 10]): 0
'17' MEMBER OF(['23', 'abc', 17, 'ab', 10]): 0
1 row in set (0.00 sec)
```

この演算子をそれ自体が配列である値とともに使用するには、JSON 配列として明示的にキャストする必要があります。これは、`CAST(... AS JSON)` で実行できます:

```
mysql> SELECT CAST('[4,5]' AS JSON) MEMBER OF('[[3,4],[4,5]]');
+-----+
| CAST('[4,5]' AS JSON) MEMBER OF('[[3,4],[4,5]]') |
+-----+
|                      1 |
+-----+
1 row in set (0.00 sec)
```

次のように、`JSON_ARRAY()` 関数を使用して必要なキャストを実行することもできます:

```
mysql> SELECT JSON_ARRAY(4,5) MEMBER OF('[[3,4],[4,5]]');
+-----+
| JSON_ARRAY(4,5) MEMBER OF('[[3,4],[4,5]]') |
```



```

+-----+
|          1 |
+-----+
1 row in set (0.00 sec)

```

テストする値として使用される JSON オブジェクト、またはターゲット配列に表示される JSON オブジェクトは、`CAST(... AS JSON)` または `JSON_OBJECT()` を使用して正しい型に強制変換する必要があります。また、JSON オブジェクトを含むターゲット配列自体は、`JSON_ARRAY` を使用してキャストする必要があります。これは、次の一連のステートメントで示されます:

```

mysql> SET @a = CAST('{"a":1}' AS JSON);
Query OK, 0 rows affected (0.00 sec)

mysql> SET @b = JSON_OBJECT("b", 2);
Query OK, 0 rows affected (0.00 sec)

mysql> SET @c = JSON_ARRAY(17, @b, "abc", @a, 23);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @a MEMBER OF(@c), @b MEMBER OF(@c);
+-----+-----+
| @a MEMBER OF(@c) | @b MEMBER OF(@c) |
+-----+-----+
|          1 |          1 |
+-----+-----+
1 row in set (0.00 sec)

```

`MEMBER OF()` 演算子が MySQL 8.0.17 に追加されました。

12.18.4 JSON 値を変更する関数

このセクションの関数は、JSON 値を変更して結果を返します。

- `JSON_ARRAY_APPEND(json_doc, path, val[, path, val] ...)`

JSON ドキュメント内の指定された配列の末尾に値を追加し、結果を返します。引数のいずれかが `NULL` である場合は、`NULL` を返します。`json_doc` 引数が有効な JSON ドキュメントでないか、`path` 引数が有効なパス式でないか、`*` または `**` ワイルドカードが含まれている場合、エラーが発生します。

パスと値のペアは左から右に評価されます。あるペアを評価して生成されたドキュメントは、次のペアが評価される新しい値になります。

パスがスカラー値またはオブジェクト値を選択すると、その値は配列内で自動ラップされ、新しい値がその配列に追加されます。パスが JSON ドキュメント内の値を識別しないペアは無視されます。

```

mysql> SET @j = '{"a", ["b", "c"], "d"}';
mysql> SELECT JSON_ARRAY_APPEND(@j, '$[1]', 1);
+-----+
| JSON_ARRAY_APPEND(@j, '$[1]', 1) |
+-----+
| ["a", ["b", "c", 1], "d"] |
+-----+
mysql> SELECT JSON_ARRAY_APPEND(@j, '$[0]', 2);
+-----+
| JSON_ARRAY_APPEND(@j, '$[0]', 2) |
+-----+
| [{"a": 2, ["b", "c"], "d"} |
+-----+
mysql> SELECT JSON_ARRAY_APPEND(@j, '$[1][0]', 3);
+-----+
| JSON_ARRAY_APPEND(@j, '$[1][0]', 3) |
+-----+
| ["a", [{"b": 3, "c"}, "d"] |
+-----+

mysql> SET @j = '{"a": 1, "b": [2, 3], "c": 4}';
mysql> SELECT JSON_ARRAY_APPEND(@j, '$.b', 'x');
+-----+
| JSON_ARRAY_APPEND(@j, '$.b', 'x') |
+-----+

```

```

|{"a": 1, "b": [2, 3, "x"], "c": 4}|
+-----+
mysql> SELECT JSON_ARRAY_APPEND(@j, '$.c', 'y');
+-----+
|JSON_ARRAY_APPEND(@j, '$.c', 'y')|
+-----+
|{"a": 1, "b": [2, 3], "c": [4, "y"]} |
+-----+

mysql> SET @j = '{"a": 1}';
mysql> SELECT JSON_ARRAY_APPEND(@j, '$', 'z');
+-----+
|JSON_ARRAY_APPEND(@j, '$', 'z')|
+-----+
|["a": 1, "z"]|
+-----+

```

MySQL 5.7 では、この関数は `JSON_APPEND()` という名前でした。この名前は、MySQL 8.0 ではサポートされなくなりました。

- `JSON_ARRAY_INSERT(json_doc, path, val[, path, val] ...)`

JSON ドキュメントを更新し、ドキュメント内の配列に挿入して、変更されたドキュメントを返します。引数のいずれかが `NULL` である場合は、`NULL` を返します。 `json_doc` 引数が有効な JSON ドキュメントでないか、 `path` 引数が有効なパス式でないか、 `*` または `**` ワイルドカードを含んでいるか、配列要素識別子で終わらない場合、エラーが発生します。

パスと値のペアは左から右に評価されます。あるペアを評価して生成されたドキュメントは、次のペアが評価される新しい値になります。

パスが JSON ドキュメント内の配列を識別しないペアは無視されます。パスが配列要素を識別する場合、対応する値がその要素の位置に挿入され、後続の値は右にシフトされます。パスが配列の末尾より後の配列位置を識別した場合、配列の末尾に値が挿入されます。

```

mysql> SET @j = '{"a", {"b": [1, 2]}, [3, 4]}';
mysql> SELECT JSON_ARRAY_INSERT(@j, '$[1]', 'x');
+-----+
|JSON_ARRAY_INSERT(@j, '$[1]', 'x')|
+-----+
|{"a", "x", {"b": [1, 2]}, [3, 4]} |
+-----+

mysql> SELECT JSON_ARRAY_INSERT(@j, '$[100]', 'x');
+-----+
|JSON_ARRAY_INSERT(@j, '$[100]', 'x')|
+-----+
|{"a", {"b": [1, 2]}, [3, 4], "x"} |
+-----+

mysql> SELECT JSON_ARRAY_INSERT(@j, '$[1].b[0]', 'x');
+-----+
|JSON_ARRAY_INSERT(@j, '$[1].b[0]', 'x')|
+-----+
|{"a", {"b": [{"x", 1, 2}], [3, 4]} |
+-----+

mysql> SELECT JSON_ARRAY_INSERT(@j, '$[2][1]', 'y');
+-----+
|JSON_ARRAY_INSERT(@j, '$[2][1]', 'y')|
+-----+
|{"a", {"b": [1, 2]}, [3, "y", 4]} |
+-----+

mysql> SELECT JSON_ARRAY_INSERT(@j, '$[0]', 'x', '$[2][1]', 'y');
+-----+
|JSON_ARRAY_INSERT(@j, '$[0]', 'x', '$[2][1]', 'y')|
+-----+
|{"x", "a", {"b": [1, 2]}, [3, 4]} |
+-----+

```

以前の変更は配列内の次の要素の位置に影響するため、同じ `JSON_ARRAY_INSERT()` コール内の後続のパスでこれを考慮する必要があります。最後の例では、最初の挿入後にパスが何にも一致しなくなったため、2 番目のパスは何も挿入しません。

- [JSON_INSERT\(json_doc, path, val\[, path, val\] ...\)](#)

JSON ドキュメントにデータを挿入し、結果を返します。引数のいずれかが `NULL` である場合は、`NULL` を返します。 `json_doc` 引数が有効な JSON ドキュメントでないか、 `path` 引数が有効なパス式でないか、 `*` または `**` ワイルドカードが含まれている場合、エラーが発生します。

パスと値のペアは左から右に評価されます。あるペアを評価して生成されたドキュメントは、次のペアが評価される新しい値になります。

ドキュメント内の既存のパスのパスと値のペアは無視され、既存のドキュメント値は上書きされません。パスが次のいずれかのタイプの値を識別する場合、ドキュメント内の存在しないパスのパスと値のペアによって、ドキュメントに値が追加されます:

- 既存のオブジェクトにメンバーが存在しません。メンバーがオブジェクトに追加され、新しい値に関連付けられます。
- 既存の配列の末尾を越えた位置。配列は新しい値で拡張されます。既存の値が配列でない場合は、配列として自動ラップされ、新しい値で拡張されます。

それ以外の場合、ドキュメント内に存在しないパスのパスと値のペアは無視され、効果はありません。

[JSON_INSERT\(\)](#)、[JSON_REPLACE\(\)](#) および [JSON_SET\(\)](#) の比較は、[JSON_SET\(\)](#) の説明を参照してください。

```
mysql> SET @j = '{"a": 1, "b": [2, 3]};
mysql> SELECT JSON_INSERT(@j, '$.a', 10, '$.c', [true, false]);
+-----+
| JSON_INSERT(@j, '$.a', 10, '$.c', [true, false]) |
+-----+
| {"a": 1, "b": [2, 3], "c": "[true, false]"}      |
+-----+
```

結果にリストされている 3 番目と最後の値は引用符で囲まれた文字列であり、2 番目の値 (出力で引用符で囲まれていない) のような配列ではありません。JSON 型への値のキャストは実行されません。配列を配列として挿入するには、次に示すように、このようなキャストを明示的に実行する必要があります:

```
mysql> SELECT JSON_INSERT(@j, '$.a', 10, '$.c', CAST([true, false] AS JSON));
+-----+
| JSON_INSERT(@j, '$.a', 10, '$.c', CAST([true, false] AS JSON)) |
+-----+
| {"a": 1, "b": [2, 3], "c": [true, false]}                    |
+-----+
1 row in set (0.00 sec)
```

- [JSON_MERGE\(json_doc, json_doc\[, json_doc\] ...\)](#)

複数の JSON ドキュメントをマージします。 [JSON_MERGE_PRESERVE\(\)](#) のシノニムです。MySQL 8.0.3 では非推奨であり、将来のリリースで削除される可能性があります。

```
mysql> SELECT JSON_MERGE('[1, 2]', [true, false]);
+-----+
| JSON_MERGE('[1, 2]', [true, false]) |
+-----+
| [1, 2, true, false]                 |
+-----+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Warning
Code: 1287
Message: 'JSON_MERGE' is deprecated and will be removed in a future release. \
Please use JSON_MERGE_PRESERVE/JSON_MERGE_PATCH instead
1 row in set (0.00 sec)
```

その他の例については、[JSON_MERGE_PRESERVE\(\)](#) のエントリを参照してください。

- `JSON_MERGE_PATCH(json_doc, json_doc[, json_doc] ...)`

複数の JSON ドキュメントの RFC 7396 準拠マージを実行し、重複するキーを持つメンバーを保持せずにマージ結果を返します。この関数に引数として渡されたドキュメントのいずれかが無効な場合は、エラーが発生します。

注記

この関数と `JSON_MERGE_PRESERVE()` の違いの説明および例は、[JSON_MERGE_PATCH\(\) と JSON_MERGE_PRESERVE\(\) の比較](#) を参照してください。

`JSON_MERGE_PATCH()` は、次のようにマージを実行します:

1. 最初の引数がオブジェクトでない場合、マージの結果は、空のオブジェクトが 2 番目の引数とマージされた場合と同じになります。
2. 2 番目の引数がオブジェクトでない場合、マージの結果は 2 番目の引数になります。
3. 両方の引数がオブジェクトの場合、マージの結果は次のメンバーを持つオブジェクトになります:
 - 2 番目のオブジェクトに同じキーを持つ対応するメンバを持たない、1 番目のオブジェクトのすべてのメンバ。
 - 最初のオブジェクトに対応するキーがなく、値が JSON `null` リテラルではない、2 番目のオブジェクトのすべてのメンバ。
 - 最初と 2 番目の両方のオブジェクトに存在し、2 番目のオブジェクトの値が JSON `null` リテラルではないキーを持つすべてのメンバ。これらのメンバの値は、最初のオブジェクトの値を 2 番目のオブジェクトの値と再帰的にマージした結果です。

追加情報については [JSON 値の正規化、マージおよび自動ラップ](#) を参照してください。

```
mysql> SELECT JSON_MERGE_PATCH('[1, 2]', '[true, false]');
+-----+
| JSON_MERGE_PATCH('[1, 2]', '[true, false]') |
+-----+
| [true, false] |
+-----+

mysql> SELECT JSON_MERGE_PATCH('{\"name\": \"x\"}, {\"id\": 47}');
+-----+
| JSON_MERGE_PATCH('{\"name\": \"x\"}, {\"id\": 47}') |
+-----+
| {\"id\": 47, \"name\": \"x\"} |
+-----+

mysql> SELECT JSON_MERGE_PATCH('1', 'true');
+-----+
| JSON_MERGE_PATCH('1', 'true') |
+-----+
| true |
+-----+

mysql> SELECT JSON_MERGE_PATCH('[1, 2]', '{\"id\": 47}');
+-----+
| JSON_MERGE_PATCH('[1, 2]', '{\"id\": 47}') |
+-----+
| {\"id\": 47} |
+-----+

mysql> SELECT JSON_MERGE_PATCH('{ \"a\": 1, \"b\":2 }',
> { \"a\": 3, \"c\":4 });
+-----+
| JSON_MERGE_PATCH('{ \"a\": 1, \"b\":2 }',{ \"a\": 3, \"c\":4 }) |
+-----+
| {\"a\": 3, \"b\": 2, \"c\": 4} |
+-----+

mysql> SELECT JSON_MERGE_PATCH('{ \"a\": 1, \"b\":2 }',{ \"a\": 3, \"c\":4 },
```

```

> '{ "a": 5, "d": 6 }';
+-----+
| JSON_MERGE_PATCH('{ "a": 1, "b": 2 }, { "a": 3, "c": 4 }, { "a": 5, "d": 6 }) |
+-----+
| { "a": 5, "b": 2, "c": 4, "d": 6 } |
+-----+

```

この関数を使用してメンバーを削除するには、次に示すように、second 引数で同じメンバーの値として `null` を指定します:

```

mysql> SELECT JSON_MERGE_PATCH('{ "a": 1, "b": 2 }, { "b": null });
+-----+
| JSON_MERGE_PATCH('{ "a": 1, "b": 2 }, { "b": null }) |
+-----+
| { "a": 1 } |
+-----+

```

この例は、関数が再帰的に動作することを示しています。つまり、メンバーの値はスカラーに制限されず、JSON ドキュメントにすることもできます:

```

mysql> SELECT JSON_MERGE_PATCH('{ "a": { "x": 1 } }, { "a": { "y": 2 } });
+-----+
| JSON_MERGE_PATCH('{ "a": { "x": 1 } }, { "a": { "y": 2 } }) |
+-----+
| { "a": { "x": 1, "y": 2 } } |
+-----+

```

`JSON_MERGE_PATCH()` は、MySQL 8.0.3 以降でサポートされます。

`JSON_MERGE_PATCH()` と `JSON_MERGE_PRESERVE()` の比較. `JSON_MERGE_PATCH()` の動作は `JSON_MERGE_PRESERVE()` の動作と同じですが、次の 2 つの例外があります:

- 2 番目のオブジェクト内のキーに関連付けられた値が JSON `null` でない場合、`JSON_MERGE_PATCH()` は、2 番目のオブジェクト内の一致するキーを持つ最初のオブジェクト内のメンバーを削除します。
- 2 つ目のオブジェクトに、最初のオブジェクトのメンバーと一致するキーを持つメンバーがある場合、`JSON_MERGE_PATCH()` replaces は最初のオブジェクトの値を 2 つ目のオブジェクトの値と照合し、`JSON_MERGE_PRESERVE()` appends は 2 つ目の値を最初の値と照合します。

この例では、同じ 3 つの JSON オブジェクトをマージした結果を比較します。それぞれに一致するキー "a" があり、次の 2 つの関数があります:

```

mysql> SET @x = '{ "a": 1, "b": 2 }',
> @y = '{ "a": 3, "c": 4 }',
> @z = '{ "a": 5, "d": 6 }';

mysql> SELECT JSON_MERGE_PATCH(@x, @y, @z) AS Patch,
-> JSON_MERGE_PRESERVE(@x, @y, @z) AS PreserveG
***** 1. row *****
Patch: { "a": 5, "b": 2, "c": 4, "d": 6 }
Preserve: { "a": [1, 3, 5], "b": 2, "c": 4, "d": 6 }

```

- `JSON_MERGE_PRESERVE(json_doc, json_doc[, json_doc] ...)`

複数の JSON ドキュメントをマージし、マージ結果を返します。引数のいずれかが `NULL` である場合は、`NULL` を返します。引数が有効な JSON ドキュメントでない場合は、エラーが発生します。

マージは、次のルールに従って行われます。追加情報については [JSON 値の正規化](#)、[マージおよび自動ラップ](#) を参照してください。

- 隣接する配列は単一の配列にマージされます。
- 隣接するオブジェクトは単一のオブジェクトにマージされます。
- スカラー値は配列として自動ラップされ、配列としてマージされます。
- 隣接する配列とオブジェクトをマージするには、オブジェクトを配列として自動ラップし、2 つの配列をマージします。

```
mysql> SELECT JSON_MERGE_PRESERVE('[1, 2]', '[true, false]');
```

```
+-----+
| JSON_MERGE_PRESERVE('[1, 2]', '[true, false]') |
+-----+
| [1, 2, true, false] |
+-----+
```

```
mysql> SELECT JSON_MERGE_PRESERVE('{\"name\": \"x\"}', '{\"id\": 47}');
```

```
+-----+
| JSON_MERGE_PRESERVE('{\"name\": \"x\"}', '{\"id\": 47}') |
+-----+
| {\"id\": 47, \"name\": \"x\"} |
+-----+
```

```
mysql> SELECT JSON_MERGE_PRESERVE('1', 'true');
```

```
+-----+
| JSON_MERGE_PRESERVE('1', 'true') |
+-----+
| [1, true] |
+-----+
```

```
mysql> SELECT JSON_MERGE_PRESERVE('[1, 2]', '{\"id\": 47}');
```

```
+-----+
| JSON_MERGE_PRESERVE('[1, 2]', '{\"id\": 47}') |
+-----+
| [1, 2, {\"id\": 47}] |
+-----+
```

```
mysql> SELECT JSON_MERGE_PRESERVE('{ \"a\": 1, \"b\": 2 },
> { \"a\": 3, \"c\": 4 }');
```

```
+-----+
| JSON_MERGE_PRESERVE('{ \"a\": 1, \"b\": 2 }, { \"a\": 3, \"c\": 4 }') |
+-----+
| {\"a\": [1, 3], \"b\": 2, \"c\": 4} |
+-----+
```

```
mysql> SELECT JSON_MERGE_PRESERVE('{ \"a\": 1, \"b\": 2 }, { \"a\": 3, \"c\": 4 },
> { \"a\": 5, \"d\": 6 }');
```

```
+-----+
| JSON_MERGE_PRESERVE('{ \"a\": 1, \"b\": 2 }, { \"a\": 3, \"c\": 4 }, { \"a\": 5, \"d\": 6 }') |
+-----+
| {\"a\": [1, 3, 5], \"b\": 2, \"c\": 4, \"d\": 6} |
+-----+
```

この関数は、`JSON_MERGE()` のシノニムとして MySQL 8.0.3 に追加されました。`JSON_MERGE()` 関数は非推奨になり、MySQL の将来のリリースで削除される予定です。

この関数は、`JSON_MERGE_PATCH()` と似ていますが、重要な点で異なります。詳細は、[JSON_MERGE_PATCH\(\) と JSON_MERGE_PRESERVE\(\) の比較](#) を参照してください。

- [JSON_REMOVE\(json_doc, path\[, path\] ...\)](#)

JSON ドキュメントからデータを削除し、結果を返します。引数のいずれかが `NULL` である場合は、`NULL` を返します。`json_doc` 引数が有効な JSON ドキュメントでないか、`path` 引数が有効なパス式でないか、`$` であるか、`*` または `**` ワイルドカードが含まれている場合、エラーが発生します。

`path` 引数は左から右に評価されます。あるパスを評価して生成されたドキュメントは、次のパスが評価される新しい値になります。

削除する要素がドキュメントに存在しない場合、エラーにはなりません。その場合、パスはドキュメントに影響しません。

```
mysql> SET @j = '{"a": ["b", "c"], "d": 1}';
mysql> SELECT JSON_REMOVE(@j, '$[1]');
+-----+
| JSON_REMOVE(@j, '$[1]') |
+-----+
| ["a", "d"]              |
+-----+
```

- [JSON_REPLACE\(json_doc, path, val\[, path, val\] ...\)](#)

JSON ドキュメント内の既存の値を置換し、結果を返します。引数のいずれかが `NULL` である場合は、`NULL` を返します。`json_doc` 引数が有効な JSON ドキュメントでないか、`path` 引数が有効なパス式でないか、`*` または `**` ワイルドカードが含まれている場合、エラーが発生します。

パスと値のペアは左から右に評価されます。あるペアを評価して生成されたドキュメントは、次のペアが評価される新しい値になります。

ドキュメント内の既存のパスのパスと値のペアによって、既存のドキュメント値が新しい値で上書きされます。ドキュメント内に存在しないパスのパスと値のペアは無視され、効果はありません。

MySQL 8.0.4 では、オプティマイザは、古いドキュメントを削除して新しいドキュメント全体をカラムに書き込むかわりに、JSON カラムの部分的なインプレース更新を実行できます。この最適化は、[JSON_REPLACE\(\)](#) 関数を使用し、[JSON 値の部分更新](#) で説明されている条件を満たす UPDATE ステートメントに対して実行できます。

[JSON_INSERT\(\)](#)、[JSON_REPLACE\(\)](#) および [JSON_SET\(\)](#) の比較は、[JSON_SET\(\)](#) の説明を参照してください。

```
mysql> SET @j = '{"a": 1, "b": [2, 3]}';
mysql> SELECT JSON_REPLACE(@j, '$.a', 10, '$.c', '[true, false]');
+-----+
| JSON_REPLACE(@j, '$.a', 10, '$.c', '[true, false]') |
+-----+
| {"a": 10, "b": [2, 3], "c": [true, false]}          |
+-----+
```

- [JSON_SET\(json_doc, path, val\[, path, val\] ...\)](#)

JSON ドキュメントのデータを挿入または更新し、結果を返します。いずれかの引数が `NULL` または `path` の場合、`NULL` を返します (指定されている場合)。`json_doc` 引数が有効な JSON ドキュメントでないか、`path` 引数が有効なパス式でないか、`*` または `**` ワイルドカードが含まれている場合、エラーが発生します。

パスと値のペアは左から右に評価されます。あるペアを評価して生成されたドキュメントは、次のペアが評価される新しい値になります。

ドキュメント内の既存のパスのパスと値のペアによって、既存のドキュメント値が新しい値で上書きされます。パスが次のいずれかのタイプの値を識別する場合、ドキュメント内の存在しないパスのパスと値のペアによって、ドキュメントに値が追加されます:

- 既存のオブジェクトにメンバーが存在しません。メンバーがオブジェクトに追加され、新しい値に関連付けられます。
- 既存の配列の末尾を越えた位置。配列は新しい値で拡張されます。既存の値が配列でない場合は、配列として自動ラップされ、新しい値で拡張されます。

それ以外の場合、ドキュメント内に存在しないパスのパスと値のペアは無視され、効果はありません。

MySQL 8.0.4 では、オプティマイザは、古いドキュメントを削除して新しいドキュメント全体をカラムに書き込むかわりに、JSON カラムの部分的なインプレース更新を実行できます。この最適化は、`JSON_SET()` 関数を使用し、[JSON 値の部分更新](#) で説明されている条件を満たす UPDATE ステートメントに対して実行できます。

`JSON_SET()`、`JSON_INSERT()` および `JSON_REPLACE()` 関数が関連しています：

- `JSON_SET()` では、既存の値が置換され、存在しない値が追加されます。
- `JSON_INSERT()` は、既存の値を置換せずに値を挿入します。
- `JSON_REPLACE()` は、のみの既存の値を置き換えます。

次の例では、ドキュメント (`$.a`) に存在するパスと存在しないパス (`$.c`) を使用して、これらの違いを示します：

```
mysql> SET @j = '{ "a": 1, "b": [2, 3] }';
mysql> SELECT JSON_SET(@j, '$.a', 10, '$.c', [true, false]);
+-----+
| JSON_SET(@j, '$.a', 10, '$.c', [true, false]) |
+-----+
| {"a": 10, "b": [2, 3], "c": "[true, false]"} |
+-----+
mysql> SELECT JSON_INSERT(@j, '$.a', 10, '$.c', [true, false]);
+-----+
| JSON_INSERT(@j, '$.a', 10, '$.c', [true, false]) |
+-----+
| {"a": 1, "b": [2, 3], "c": "[true, false]"} |
+-----+
mysql> SELECT JSON_REPLACE(@j, '$.a', 10, '$.c', [true, false]);
+-----+
| JSON_REPLACE(@j, '$.a', 10, '$.c', [true, false]) |
+-----+
| {"a": 10, "b": [2, 3]} |
+-----+
```

• `JSON_UNQUOTE(json_val)`

JSON 値を引用符で囲まずに、結果を `utf8mb4` 文字列として返します。引数が `NULL` の場合、`NULL` を返します。値の先頭と末尾が二重引用符であるが、有効な JSON 文字列リテラルではない場合、エラーが発生します。

`NO_BACKSLASH_ESCAPES` SQL モードが有効になっている場合を除いて、一部のシーケンスが文字列内で特別な意味を持ちます。これらのシーケンスはいずれも、エスケープ文字として知られるバックスラッシュ (`\`) で始まります。MySQL は、[表 12.23 「JSON_UNQUOTE\(\) 特殊文字エスケープシーケンス」](#) に示すエスケープシーケンスを認識します。ほかのすべてのエスケープシーケンスでは、バックスラッシュは無視されます。つまり、エスケープされた文字がエスケープされていないと解釈されます。たとえば、`\x` は単なる `x` です。これらの順序では、大/小文字が区別されます。たとえば、`\b` はバックスペースとして解釈されますが、`\B` は `B` として解釈されます。

表 12.23 `JSON_UNQUOTE()` 特殊文字エスケープシーケンス

エスケープシーケンス	シーケンスが表す文字
<code>"</code>	二重引用符 (<code>"</code>) 文字
<code>\b</code>	バックスペース文字
<code>\f</code>	フォームフィード文字
<code>\n</code>	改行 (ラインフィード) 文字
<code>\r</code>	復帰改行文字
<code>\t</code>	タブ文字
<code>\\</code>	バックスラッシュ (<code>\</code>) 文字
<code>\uXXXX</code>	UTF-8Unicode 値 <code>XXXX</code> のバイト数

この関数の 2 つの簡単な使用例を次に示します：

```
mysql> SET @j = '"abc"';
mysql> SELECT @j, JSON_UNQUOTE(@j);
```

```

+-----+
| @j | JSON_UNQUOTE(@j) |
+-----+
| "abc" | abc |
+-----+
mysql> SET @j = '[1, 2, 3]';
mysql> SELECT @j, JSON_UNQUOTE(@j);
+-----+
| @j | JSON_UNQUOTE(@j) |
+-----+
| [1, 2, 3] | [1, 2, 3] |
+-----+

```

次の一連の例は、`NO_BACKSLASH_ESCAPES` を無効にして有効にした場合に `JSON_UNQUOTE` がエスケープを処理する方法を示しています:

```

mysql> SELECT @@sql_mode;
+-----+
| @@sql_mode |
+-----+
|          |
+-----+

mysql> SELECT JSON_UNQUOTE("\t\u0032");
+-----+
| JSON_UNQUOTE("\t\u0032") |
+-----+
| 2 |
+-----+

mysql> SET @@sql_mode = 'NO_BACKSLASH_ESCAPES';
mysql> SELECT JSON_UNQUOTE("\t\u0032");
+-----+
| JSON_UNQUOTE("\t\u0032") |
+-----+
| \t\u0032 |
+-----+

mysql> SELECT JSON_UNQUOTE("\t\u0032");
+-----+
| JSON_UNQUOTE("\t\u0032") |
+-----+
| 2 |
+-----+

```

12.18.5 JSON 値属性を返す関数

このセクションの関数は、JSON 値の属性を戻します。

- `JSON_DEPTH(json_doc)`

JSON ドキュメントの最大深度を返します。引数が `NULL` の場合、`NULL` を返します。引数が有効な JSON ドキュメントでない場合は、エラーが発生します。

空の配列、空のオブジェクトまたはスカラー値の深さは 1 です。深さ 1 の要素のみを含む空でない配列、または深さ 1 のメンバー値のみを含む空でないオブジェクトは深さ 2 です。それ以外の場合、JSON ドキュメントの深さは 2 より大きくなります。

```

mysql> SELECT JSON_DEPTH('{}'), JSON_DEPTH('[]'), JSON_DEPTH('true');
+-----+
| JSON_DEPTH('{}') | JSON_DEPTH('[]') | JSON_DEPTH('true') |
+-----+
| 1 | 1 | 1 |
+-----+
mysql> SELECT JSON_DEPTH('[10, 20]'), JSON_DEPTH('[[, {}]');
+-----+
| JSON_DEPTH('[10, 20]') | JSON_DEPTH('[[, {}]') |
+-----+
| 2 | 2 |
+-----+
mysql> SELECT JSON_DEPTH('[10, {"a": 20}]');
+-----+

```

```
| JSON_DEPTH(['10', {'a': 20}]) |
+-----+
|           3 |
+-----+
```

- `JSON_LENGTH(json_doc[, path])`

JSON ドキュメントの長さを返します。path 引数が指定されている場合は、パスで識別されるドキュメント内の値の長さを返します。いずれかの引数が `NULL` の場合、または path 引数がドキュメント内の値を識別しない場合、`NULL` を返します。json_doc 引数が有効な JSON ドキュメントでないか、path 引数が有効なパス式でないか、* または ** ワイルドカードが含まれている場合、エラーが発生します。

ドキュメントの長さは次のように決定されます:

- スカラーの長さは 1 です。
- 配列の長さは配列要素の数です。
- オブジェクトの長さは、オブジェクトメンバーの数です。
- 長さは、ネストされた配列またはオブジェクトの長さをカウントしません。

```
mysql> SELECT JSON_LENGTH(['1, 2, {'a': 3}]);
+-----+
| JSON_LENGTH(['1, 2, {'a': 3}]) |
+-----+
|           3 |
+-----+
mysql> SELECT JSON_LENGTH({'a': 1, "b": {'c': 30}});
+-----+
| JSON_LENGTH({'a': 1, "b": {'c': 30}}) |
+-----+
|           2 |
+-----+
mysql> SELECT JSON_LENGTH({'a': 1, "b": {'c': 30}}, '$.b');
+-----+
| JSON_LENGTH({'a': 1, "b": {'c': 30}}, '$.b') |
+-----+
|           1 |
+-----+
```

- `JSON_TYPE(json_val)`

JSON 値のタイプを示す utf8mb4 文字列を返します。次に示すように、オブジェクト型、配列型またはスカラー型を指定できます:

```
mysql> SET @j = '{"a": [10, true]}';
mysql> SELECT JSON_TYPE(@j);
+-----+
| JSON_TYPE(@j) |
+-----+
| OBJECT      |
+-----+
mysql> SELECT JSON_TYPE(JSON_EXTRACT(@j, '$.a'));
+-----+
| JSON_TYPE(JSON_EXTRACT(@j, '$.a')) |
+-----+
| ARRAY      |
+-----+
mysql> SELECT JSON_TYPE(JSON_EXTRACT(@j, '$.a[0]'));
+-----+
| JSON_TYPE(JSON_EXTRACT(@j, '$.a[0]')) |
+-----+
| INTEGER    |
+-----+
mysql> SELECT JSON_TYPE(JSON_EXTRACT(@j, '$.a[1]'));
+-----+
| JSON_TYPE(JSON_EXTRACT(@j, '$.a[1]')) |
+-----+
| BOOLEAN    |
+-----+
```

```
+-----+
```

引数が `NULL` の場合、`JSON_TYPE()` は `NULL` を返します:

```
mysql> SELECT JSON_TYPE(NULL);
+-----+
| JSON_TYPE(NULL) |
+-----+
| NULL            |
+-----+
```

引数が有効な JSON 値でない場合は、エラーが発生します:

```
mysql> SELECT JSON_TYPE(1);
ERROR 3146 (22032): Invalid data type for JSON data in argument 1
to function json_type; a JSON string or JSON type is required.
```

`NULL` 以外のエラー以外の結果の場合、次のリストに、考えられる `JSON_TYPE()` の戻り値を示します:

- 純粋な JSON 型:
 - `OBJECT`: JSON オブジェクト
 - `ARRAY`: JSON 配列
 - `BOOLEAN`: JSON `true` および `false` リテラル
 - `NULL`: JSON `null` リテラル
- 数値型:
 - `INTEGER`: MySQL `TINYINT`, `SMALLINT`, `MEDIUMINT` および `INT` と `BIGINT` のスカラー
 - `DOUBLE`: MySQL `DOUBLE FLOAT` スカラー
 - `DECIMAL`: MySQL `DECIMAL` および `NUMERIC` スカラー
- 時間型:
 - `DATETIME`: MySQL `DATETIME` および `TIMESTAMP` スカラー
 - `DATE`: MySQL `DATE` スカラー
 - `TIME`: MySQL `TIME` スカラー
- 文字列型:
 - `STRING`: MySQL `utf8` 文字型スカラー: `CHAR`, `VARCHAR`, `TEXT`, `ENUM` および `SET`
- バイナリ型:
 - `BLOB`: `BINARY`, `VARBINARY`, `BLOB` および `BIT` を含む MySQL バイナリ型スカラー
- 他のすべてのタイプ:
 - `OPAQUE` (raw ビット)
- `JSON_VALID(val)`

値が有効な JSON かどうかを示す 0 または 1 を返します。引数が `NULL` の場合、`NULL` を返します。

```
mysql> SELECT JSON_VALID('{\"a\": 1}');
+-----+
| JSON_VALID('{\"a\": 1}') |
+-----+
| 1 |
+-----+
mysql> SELECT JSON_VALID('hello'), JSON_VALID('\"hello\"');
```

```

+-----+-----+
| JSON_VALID('hello') | JSON_VALID('hello') |
+-----+-----+
|          0          |          1          |
+-----+-----+

```

12.18.6 JSON テーブル関数

このセクションでは、JSON データを表形式データに変換する JSON 関数について説明します。MySQL 8.0.4 以降では、このような機能の `JSON_TABLE()` がサポートされています。

- `JSON_TABLE(expr, path COLUMNS (column_list) [AS] alias)`

JSON ドキュメントからデータを抽出し、指定されたカラムを持つリレーショナルテーブルとして戻します。この関数の完全な構文を次に示します:

```

JSON_TABLE(
  expr,
  path COLUMNS (column_list)
) [AS] alias

column_list:
  column[, column][, ...]

column:
  name FOR ORDINALITY
  | name type PATH string path [on_empty] [on_error]
  | name type EXISTS PATH string path
  | NESTED [PATH] path COLUMNS (column_list)

on_empty:
  {NULL | DEFAULT json_string | ERROR} ON EMPTY

on_error:
  {NULL | DEFAULT json_string | ERROR} ON ERROR

```

expr: これは JSON データを返す式です。定数 (`'{"a":1}'`)、カラム (`t1.json_data`、`FROM` 句で `JSON_TABLE()` の前に指定されたテーブル `t1`)、または関数コール (`JSON_EXTRACT(t1.json_data,'$.post.comments')`) を指定できません。

path: データソースに適用される JSON パス式。パスに一致する JSON 値を行ソースと呼びます。これはリレーショナルデータの行を生成するために使用されます。 `COLUMNS` 句は、行ソースを評価し、行ソース内の特定の JSON 値を検索して、リレーショナルデータの行の個々のカラムにそれらの JSON 値を SQL 値として返します。

alias は必須です。テーブルのエイリアスの通常のルールが適用されます ([セクション9.2「スキーマオブジェクト名」](#) を参照)。

`JSON_TABLE()` では、次のリストに示す 4 つのタイプのカラムがサポートされています:

1. **name FOR ORDINALITY:** このタイプは、`COLUMNS` 句の行を列挙します。 `name` という名前のカラムは、タイプが `UNSIGNED INT` で初期値が 1 のカウンタです。これは、`CREATE TABLE` ステートメントでカラムを `AUTO_INCREMENT` として指定することと同等で、`NESTED [PATH]` 句によって生成された複数の行に対して同じ値を持つ親行を区別するために使用できます。
2. **name type PATH string_path [on_empty] [on_error]:** このタイプのカラムは、`string_path` で指定された値を抽出するために使用されます。 `type` は MySQL スカラーデータ型です (つまり、オブジェクトまたは配列にすることはできません)。 `JSON_TABLE()` は、JSON としてデータを抽出し、MySQL の JSON データに適用される通常の自動型変換を使用して、そのデータをカラムタイプに強制的に変換します。欠損値によって `on_empty` 句がトリガーされます。オブジェクトまたは配列を保存すると、オプションの `on_error` 句がトリガーされます。これは、文字列 `'asd'` を整数カラムに保存しようとするなど、JSON として保存された値からテーブルのカラムへの強制変換中にエラーが発生した場合にも発生します。
3. **name type EXISTS PATH path:** このカラムは、`path` で指定された場所にデータが存在する場合は 1 を返し、それ以外の場合は 0 を返します。 `type` は任意の有効な MySQL データ型にできますが、通常は様々な `INT` として指定する必要があります。

4. **NESTED [PATH] path COLUMNS (column_list)**: これにより、JSON データ内のネストされたオブジェクトまたは配列が、親オブジェクトまたは配列からの JSON 値とともに単一行にフラット化されます。複数の **PATH** オプションを使用すると、複数レベルのネストから単一行に JSON 値を投影できます。

path は、**JSON_TABLE()** の親パス行パス、またはネストされたパスの場合の親 **NESTED [PATH]** 句のパスからの相対パスです。

on empty (指定されている場合) は、データが欠落している場合に **JSON_TABLE()** が何を行うかを決定します (タイプによって異なります)。この句は、**NESTED PATH** 句のカラムに一致がなく、**NULL** 補完行が生成された場合にもトリガーされます。**on empty** は、次のいずれかの値を取ります:

- **NULL ON EMPTY**: カラムは **NULL** に設定されています。これはデフォルトの動作です。
- **DEFAULT json_string ON EMPTY**: 指定された **json_string** は、有効であるかぎり JSON として解析され、欠落している値のかわりに格納されます。カラムタイプのルールは、デフォルト値にも適用されます。
- **ERROR ON EMPTY**: エラーがスローされます。

使用する場合、**on_error** は、次に示すように、対応する結果とともに次のいずれかの値を取ります:

- **NULL ON ERROR**: カラムは **NULL** に設定されています。これはデフォルトの動作です。
- **DEFAULT json string ON ERROR**: **json_string** は JSON として解析され (有効な場合)、オブジェクトまたは配列のかわりに格納されます。
- **ERROR ON ERROR**: エラーがスローされます。

MySQL 8.0.20 より前は、**NULL ON ERROR** または **DEFAULT ... ON ERROR** で型変換エラーが発生した場合、または暗黙的に指定された場合に警告がスローされました。MySQL 8.0.20 以降では、これは当てはまりません。(Bug #30628330)

以前は、**ON EMPTY** 句と **ON ERROR** 句をいずれかの順序で指定できました。これにより、SQL 標準へのカウンタが実行されます。このカウンタは、指定されている場合、**ON ERROR** 句の前にその **ON EMPTY** を指定する必要があります。このため、MySQL 8.0.20 以降では、**ON EMPTY** が非推奨になる前に **ON ERROR** を指定すると、サーバーで警告が発行されます。非標準構文のサポートは、MySQL の将来のバージョンで削除される予定です。

3.14159 を **DECIMAL(10,1)** カラムに保存するなど、カラムに保存された値が切り捨てられると、**ON ERROR** オプションとは関係なく警告が発行されます。単一のステートメントで複数の値が切り捨てられた場合、警告は一度のみ発行されます。

MySQL 8.0.21 より前は、この関数に渡された式およびパスが JSON null に解決されると、**JSON_TABLE()** でエラーが発生しました。MySQL 8.0.21 以降では、次に示すように、SQL 標準に従って SQL **NULL** が返されます (Bug #31345503、Bug #99557):

```
mysql> SELECT *
-> FROM
-> JSON_TABLE(
->   [{"c1": null}],
->   "$[*]" COLUMNS( c1 INT PATH '$.c1' ERROR ON ERROR )
-> ) as jt;
+-----+
| c1 |
+-----+
| NULL |
+-----+
1 row in set (0.00 sec)
```

次のクエリは、**ON EMPTY** および **ON ERROR** の使用方法を示しています。パス "\$.a" の {"b":1} に対応する行が空で、[1,2] をスカラーとして保存しようとするエラーが発生します。これらの行は、次に示す出力で強調表示されます。

```
mysql> SELECT *
-> FROM
-> JSON_TABLE(
->   [{"a": "3"}, {"a": "2"}, {"b": "1"}, {"a": "0"}, {"a": "[1,2]}],
```

```

-> "$[*]"
-> COLUMNS(
->   rowid FOR ORDINALITY,
->   ac VARCHAR(100) PATH "$.a" DEFAULT '111' ON EMPTY DEFAULT '999' ON ERROR,
->   aj JSON PATH "$.a" DEFAULT '{"x": 333}' ON EMPTY,
->   bx INT EXISTS PATH "$.b"
-> )
-> ) AS tt;

```

rowid	ac	aj	bx
1	3	"3"	0
2	2	2	0
3	111	'{"x": 333}'	1
4	0	0	0
5	999	'[1, 2]'	0

5 rows in set (0.00 sec)

カラム名には、テーブルのカラム名を制御する通常のルールおよび制限が適用されます。 [セクション9.2「スキーマオブジェクト名」](#) を参照してください。

すべての JSON および JSON パス式の妥当性がチェックされます。どちらのタイプの式も無効であると、エラーが発生します。

`COLUMNS` キーワードの前にある `path` の各一致は、結果テーブルの個々の行にマップされます。たとえば、次のクエリを実行すると、次に示す結果が得られます:

```

mysql> SELECT *
-> FROM
-> JSON_TABLE(
->   ['{"x":2,"y":"8"},{"x":"3","y":"7"},{"x":"4","y":"6}'],
->   "$[*]" COLUMNS(
->     xval VARCHAR(100) PATH "$.x",
->     yval VARCHAR(100) PATH "$.y"
->   )
-> ) AS jt1;

```

xval	yval
2	8
3	7
4	6

式 `"$[*]"` は配列の各要素と一致します。パスを変更することで、結果の行をフィルタできます。たとえば、`"$[1]"` を使用すると、次に示すように、抽出がソースとして使用される JSON 配列の 2 番目の要素に制限されます:

```

mysql> SELECT *
-> FROM
-> JSON_TABLE(
->   ['{"x":2,"y":"8"},{"x":"3","y":"7"},{"x":"4","y":"6}'],
->   "$[1]" COLUMNS(
->     xval VARCHAR(100) PATH "$.x",
->     yval VARCHAR(100) PATH "$.y"
->   )
-> ) AS jt1;

```

xval	yval
3	7

カラム定義内では、`"$"` は一致全体をカラムに渡します。`"$.x"` および `"$.y"` は、キー `x` および `y` に対応する値のみをその一致内で渡します。詳細は、 [JSON パス構文](#) を参照してください。

`NESTED PATH` (または単に `NESTED`。 `PATH` はオプション) では、所属する `COLUMNS` 句の一致ごとに一連のレコードが生成されます。一致しない場合、ネストされたパスのすべてのカラムが `NULL` に設定されます。これによ

り、最上位の句と `NESTED [PATH]` の間の外部結合が実装されます。内部結合をエミュレートするには、次に示すように、`WHERE` 句に適切な条件を適用します:

```
mysql> SELECT *
-> FROM
-> JSON_TABLE(
->   [{"a": 1, "b": [11,111]}, {"a": 2, "b": [22,222]}, {"a":3}],
->   "$[*]" COLUMNS(
->     a INT PATH '$.a',
->     NESTED PATH '$.b[*]' COLUMNS (b INT PATH '$')
->   )
-> ) AS jt
-> WHERE b IS NOT NULL;
```

a	b
1	11
1	111
2	22
2	222

兄弟のネストされたパス (同じ `COLUMNS` 句内の `NESTED [PATH]` の複数のインスタンス) は、一度に複数回処理されます。あるネストされたパスがレコードを生成している間、兄弟のネストされたパス式の列は `NULL` に設定されます。つまり、次に示すように、単一の包含 `COLUMNS` 句内の単一致のレコードの合計数は合計であり、`NESTED [PATH]` 修飾子によって生成されるすべてのレコードの積ではありません:

```
mysql> SELECT *
-> FROM
-> JSON_TABLE(
->   [{"a": 1, "b": [11,111]}, {"a": 2, "b": [22,222]}],
->   "$[*]" COLUMNS(
->     a INT PATH '$.a',
->     NESTED PATH '$.b[*]' COLUMNS (b1 INT PATH '$'),
->     NESTED PATH '$.b[*]' COLUMNS (b2 INT PATH '$')
->   )
-> ) AS jt;
```

a	b1	b2
1	11	NULL
1	111	NULL
1	NULL	11
1	NULL	111
2	22	NULL
2	222	NULL
2	NULL	22
2	NULL	222

`FOR ORDINALITY` 列は、`COLUMNS` 句によって生成されたレコードを列挙し、ネストされたパスの親レコードを区別するために使用できます。特に、親レコードの値が同じ場合は、次のようになります:

```
mysql> SELECT *
-> FROM
-> JSON_TABLE(
->   [{"a": "a_val",
->     "b": [{"c": "c_val", "l": [1,2]}],
->     "a": "a_val",
->     "b": [{"c": "c_val", "l": [11]}, {"c": "c_val", "l": [22]}]},
->   "$[*]" COLUMNS(
->     top_ord FOR ORDINALITY,
->     apath VARCHAR(10) PATH '$.a',
->     NESTED PATH '$.b[*]' COLUMNS (
->       bpath VARCHAR(10) PATH '$.c',
->       ord FOR ORDINALITY,
->       NESTED PATH '$.l[*]' COLUMNS (lpath varchar(10) PATH '$')
->     )
->   )
-> ) AS jt;
```

top_ord	apath	bpath	ord	lpath
1	a_val	c_val	1	1
1	a_val	c_val	1	2
2	a_val	c_val	1	11
2	a_val	c_val	2	22

ソースドキュメントには 2 つの要素の配列が含まれており、これらの各要素は 2 つの行を生成します。 `apath` と `bpath` の値は、結果セット全体で同じです。つまり、`lpath` 値が同じ親から取得されたのか、別の親から取得されたのかを判断するために使用できません。 `ord` カラムの値は、`top_ord` が 1 に等しいレコードのセットと同じままであるため、これらの 2 つの値は単一オブジェクトからの値です。残りの 2 つの値は、`ord` カラムの値が異なるため、異なるオブジェクトの値です。

12.18.7 JSON スキーマ検証関数

MySQL 8.0.17 以降、MySQL では、「JSON スキーマ仕様のドラフト 4」に準拠する JSON スキーマに対する JSON ドキュメントの検証がサポートされます。これは、このセクションで説明する関数のいずれかを使用して実行できます。どちらの関数も、JSON スキーマと、スキーマに対して検証される JSON ドキュメントの 2 つの引数を取ります。 `JSON_SCHEMA_VALID()` は、ドキュメントがスキーマに対して検証される場合は `true` を返し、検証されない場合は `false` を返します。 `JSON_SCHEMA_VALIDATION_REPORT()` は、検証に関する JSON 形式のレポートを提供します。

どちらの関数も、次のように NULL または無効な入力を処理します:

- 少なくともいずれかの引数が `NULL` の場合、この関数は `NULL` を返します。
- 少なくともいずれかの引数が有効な JSON でない場合、関数でエラーが発生します (`ER_INVALID_TYPE_FOR_JSON`)
- また、スキーマが有効な JSON オブジェクトでない場合、関数は `ER_INVALID_JSON_TYPE` を返します。

MySQL では、JSON スキーマの `required` 属性をサポートして、必要なプロパティを強制的に含めることができます (関数の説明の例を参照)。

MySQL では、JSON スキーマの `id`、`$schema`、`description` および `type` 属性がサポートされていますが、これらは必要ありません。

MySQL では、JSON スキーマの外部リソースはサポートされていません。 `$ref` キーワードを使用すると、`JSON_SCHEMA_VALID()` が `ER_NOT_SUPPORTED_YET` で失敗します。

注記

MySQL では、JSON スキーマの正規表現パターンがサポートされています。JSON スキーマでは、無効なパターンはサポートされませんが、暗黙的に無視されます (例は、`JSON_SCHEMA_VALID()` の説明を参照)。

これらの関数の詳細は、次のリストを参照してください:

- `JSON_SCHEMA_VALID(schema,document)`

JSON `schema` に対して JSON `document` を検証します。 `schema` と `document` の両方が必要です。スキーマは有効な JSON オブジェクトである必要があります。ドキュメントは有効な JSON ドキュメントである必要があります。これらの条件が満たされている場合: ドキュメントがスキーマに対して検証される場合、関数は `true` (1) を返し、それ以外の場合は `false` (0) を返します。

この例では、ユーザー変数 `@schema` を地理座標の JSON スキーマの値に設定し、別の `@document` をその座標を含む JSON ドキュメントの値に設定します。次に、`JSON_SCHEMA_VALID()` の引数として使用して、`@document` が `@schema` に従って検証されることを確認します:

```
mysql> SET @schema = '{
> "id": "http://json-schema.org/geo",
> "$schema": "http://json-schema.org/draft-04/schema#",
> "description": "A geographical coordinate",
```

```
> "type": "object",
> "properties": {
>   "latitude": {
>     "type": "number",
>     "minimum": -90,
>     "maximum": 90
>   },
>   "longitude": {
>     "type": "number",
>     "minimum": -180,
>     "maximum": 180
>   }
> },
> "required": ["latitude", "longitude"]
>};
```

Query OK, 0 rows affected (0.01 sec)

```
mysql> SET @document = '{
> "latitude": 63.444697,
> "longitude": 10.445118
>}';
```

Query OK, 0 rows affected (0.00 sec)

```
mysql> SELECT JSON_SCHEMA_VALID(@schema, @document);
```

```
+-----+
| JSON_SCHEMA_VALID(@schema, @document) |
+-----+
|                1 |
+-----+
1 row in set (0.00 sec)
```

`@schema` には `required` 属性が含まれているため、`@document` を、それ以外は有効だが必須プロパティを含まない値に設定し、次のように `@schema` に対してテストできます:

```
mysql> SET @document = '{}';
```

Query OK, 0 rows affected (0.00 sec)

```
mysql> SELECT JSON_SCHEMA_VALID(@schema, @document);
```

```
+-----+
| JSON_SCHEMA_VALID(@schema, @document) |
+-----+
|                0 |
+-----+
1 row in set (0.00 sec)
```

次に示すように、`@schema` の値を同じ JSON スキーマに設定し、`required` 属性を指定しない場合、`@document` は、プロパティが含まれていなくても有効な JSON オブジェクトであるため検証します:

```
mysql> SET @schema = '{
> "id": "http://json-schema.org/geo",
> "$schema": "http://json-schema.org/draft-04/schema#",
> "description": "A geographical coordinate",
> "type": "object",
> "properties": {
>   "latitude": {
>     "type": "number",
>     "minimum": -90,
>     "maximum": 90
>   },
>   "longitude": {
>     "type": "number",
>     "minimum": -180,
>     "maximum": 180
>   }
> }
>};
```

Query OK, 0 rows affected (0.00 sec)

```
mysql> SELECT JSON_SCHEMA_VALID(@schema, @document);
```

```
+-----+
| JSON_SCHEMA_VALID(@schema, @document) |
+-----+
```

```
|-----+
|                1 |
+-----+
1 row in set (0.00 sec)
```

JSON_SCHEMA_VALID() および CHECK 制約。 JSON_SCHEMA_VALID() を使用して、CHECK 制約を施行することもできます。

次のように作成されたテーブル geo について考えてみます。JSON カラム coordinate は、このテーブルの CHECK 制約の式として渡される JSON_SCHEMA_VALID() コールで引数として使用される JSON スキーマによって制御される、マップ上の緯度と経度の点を表します:

```
mysql> CREATE TABLE geo (
->   coordinate JSON,
->   CHECK(
->     JSON_SCHEMA_VALID(
->       '{
->         "type":"object",
->         "properties":{
->           "latitude":{"type":"number", "minimum":-90, "maximum":90},
->           "longitude":{"type":"number", "minimum":-180, "maximum":180}
->         }
->         "required": ["latitude", "longitude"]
->       },
->     coordinate
->   )
-> );
Query OK, 0 rows affected (0.45 sec)
```

注記

MySQL CHECK 制約には変数への参照を含めることができないため、JSON スキーマを使用してテーブルにこのような制約を指定する場合は、JSON スキーマを JSON_SCHEMA_VALID() インラインに渡す必要があります。

座標を表す JSON 値を次の 3 つの変数に割り当てます:

```
mysql> SET @point1 = '{"latitude":59, "longitude":18}';
Query OK, 0 rows affected (0.00 sec)

mysql> SET @point2 = '{"latitude":91, "longitude":0}';
Query OK, 0 rows affected (0.00 sec)

mysql> SET @point3 = '{"longitude":120}';
Query OK, 0 rows affected (0.00 sec)
```

次の INSERT ステートメントに示すように、これらの最初の値は有効です:

```
mysql> INSERT INTO geo VALUES(@point1);
Query OK, 1 row affected (0.05 sec)
```

2 番目の JSON 値は無効であるため、次に示すように制約は失敗します:

```
mysql> INSERT INTO geo VALUES(@point2);
ERROR 3819 (HY000): Check constraint 'geo_chk_1' is violated.
```

MySQL 8.0.19 以降では、障害の性質に関する正確な情報を取得できます。この場合、latitude 値が SHOW WARNINGS ステートメントの発行によってスキーマで定義された最大値を超えています:

```
mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Error
Code: 3934
Message: The JSON document location '#/latitude' failed requirement 'maximum' at
JSON Schema location '#/properties/latitude'.
***** 2. row *****
Level: Error
Code: 3819
Message: Check constraint 'geo_chk_1' is violated.
```



```
2 rows in set (0.00 sec)
```

必要な `latitude` プロパティがないため、前述の 3 番目の座標値も無効です。以前と同様に、`geo` テーブルに値を挿入してから、`SHOW WARNINGS` を発行することで、これを確認できます:

```
mysql> INSERT INTO geo VALUES(@point3);
ERROR 3819 (HY000): Check constraint 'geo_chk_1' is violated.
mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Error
Code: 3934
Message: The JSON document location '#' failed requirement 'required' at JSON
Schema location '#'.
***** 2. row *****
Level: Error
Code: 3819
Message: Check constraint 'geo_chk_1' is violated.
2 rows in set (0.00 sec)
```

詳しくは [セクション 13.1.20.6 「CHECK 制約」](#) をご覧ください。

JSON スキーマでは、文字列の正規表現パターンの指定がサポートされていますが、MySQL で使用される実装では、無効なパターンは暗黙的に無視されます。つまり、次に示すように、正規表現パターンが無効な場合でも、`JSON_SCHEMA_VALID()` は `true` を返すことができます:

```
mysql> SELECT JSON_SCHEMA_VALID('{\"type\":\"string\",\"pattern\":\"()\", \"abc\"}');
+-----+
| JSON_SCHEMA_VALID('{\"type\":\"string\",\"pattern\":\"()\", \"abc\"}') |
+-----+
| 1 |
+-----+
1 row in set (0.04 sec)
```

- `JSON_SCHEMA_VALIDATION_REPORT(schema,document)`

JSON `schema` に対して JSON `document` を検証します。 `schema` と `document` の両方が必要です。`JSON_VALID_SCHEMA()` と同様に、スキーマは有効な JSON オブジェクトである必要があります、ドキュメントは有効な JSON ドキュメントである必要があります。これらの条件が満たされている場合、関数は検証の結果に関するレポートを JSON ドキュメントとして返します。JSON スキーマに従って JSON ドキュメントが有効とみなされる場合、この関数は、値が `true` のプロパティ `valid` を持つ JSON オブジェクトを戻します。JSON ドキュメントが検証に失敗した場合、この関数は、次に示すプロパティを含む JSON オブジェクトを返します:

- `valid`: スキーマ検証に失敗した場合は常に `false`
- `reason`: 失敗の理由を含む判読可能な文字列
- `schema-location`: JSON スキーマ内で検証が失敗した場所を示す JSON ポインタ URI フラグメント識別子 (このリストの後のノートを参照)
- `document-location`: JSON ドキュメント内で検証が失敗した場所を示す JSON ポインタ URI フラグメント識別子 (このリストの後のノートを参照)
- `schema-failed-keyword`: 違反があった JSON スキーマ内のキーワードまたはプロパティの名前を含む文字列

注記

JSON ポインタ URI フラグメント識別子は、「[RFC 6901 - JavaScript Object Notation \(JSON\) ポインタ](#)」で定義されます。(これらは、`JSON_EXTRACT()` および他の MySQL JSON 関数で使用される JSON パス表記とは異なります。) この表記法では、`#` はドキュメント全体を表し、`#/myprop` は `myprop` という最上位プロパティに含まれるドキュメン

トの一部を表します。詳細は、前述の仕様と、このセクションで後述する例を参照してください。

この例では、ユーザー変数 `@schema` を地理座標の JSON スキーマの値に設定し、別の `@document` をその座標を含む JSON ドキュメントの値に設定します。次に、`JSON_SCHEMA_VALIDATION_REPORT()` の引数として使用して、`@document` が `@schema` に従って検証されることを確認します:

```
mysql> SET @schema = '{
  > "id": "http://json-schema.org/geo",
  > "$schema": "http://json-schema.org/draft-04/schema#",
  > "description": "A geographical coordinate",
  > "type": "object",
  > "properties": {
  >   "latitude": {
  >     "type": "number",
  >     "minimum": -90,
  >     "maximum": 90
  >   },
  >   "longitude": {
  >     "type": "number",
  >     "minimum": -180,
  >     "maximum": 180
  >   }
  > },
  > "required": ["latitude", "longitude"]
  > }';
Query OK, 0 rows affected (0.01 sec)

mysql> SET @document = '{
  > "latitude": 63.444697,
  > "longitude": 10.445118
  > }';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT JSON_SCHEMA_VALIDATION_REPORT(@schema, @document);
+-----+
| JSON_SCHEMA_VALIDATION_REPORT(@schema, @document) |
+-----+
| {"valid": true} |
+-----+
1 row in set (0.00 sec)
```

次に示すように、いずれかのプロパティに不正な値が指定されるように `@document` を設定します:

```
mysql> SET @document = '{
  > "latitude": 63.444697,
  > "longitude": 310.445118
  > }';
```

`JSON_SCHEMA_VALIDATION_REPORT()` でテストすると、`@document` の検証が失敗するようになりました。関数コールからの出力には、次に示すように、失敗に関する詳細情報が含まれます (より適切な書式設定を提供するために `JSON_PRETTY()` でラップされた関数を使用):

```
mysql> SELECT JSON_PRETTY(JSON_SCHEMA_VALIDATION_REPORT(@schema, @document))G
***** 1. row *****
JSON_PRETTY(JSON_SCHEMA_VALIDATION_REPORT(@schema, @document)): {
  "valid": false,
  "reason": "The JSON document location '#/longitude' failed requirement 'maximum' at JSON Schema location '#/properties/longitude'",
  "schema-location": "#/properties/longitude",
  "document-location": "#/longitude",
  "schema-failed-keyword": "maximum"
}
1 row in set (0.00 sec)
```

`@schema` には `required` 属性が含まれているため、`@document` を、それ以外は無効だが必須プロパティを含まない値に設定してから、`@schema` に対してテストできます。`JSON_SCHEMA_VALIDATION_REPORT()` の出力は、次のように、必要な要素がないために検証が失敗することを示しています:

```
mysql> SET @document = '{}';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT JSON_PRETTY(JSON_SCHEMA_VALIDATION_REPORT(@schema, @document))G
***** 1. row *****
JSON_PRETTY(JSON_SCHEMA_VALIDATION_REPORT(@schema, @document)): {
  "valid": false,
  "reason": "The JSON document location '#' failed requirement 'required' at JSON Schema location '#",
  "schema-location": "#",
  "document-location": "#",
  "schema-failed-keyword": "required"
}
1 row in set (0.00 sec)
```

次に示すように、`@schema` の値を同じ JSON スキーマに設定し、`required` 属性を指定しない場合、`@document` は、プロパティが含まれていなくても有効な JSON オブジェクトであるため検証します:

```
mysql> SET @schema = '{
  > "id": "http://json-schema.org/geo",
  > "$schema": "http://json-schema.org/draft-04/schema#",
  > "description": "A geographical coordinate",
  > "type": "object",
  > "properties": {
  >   "latitude": {
  >     "type": "number",
  >     "minimum": -90,
  >     "maximum": 90
  >   },
  >   "longitude": {
  >     "type": "number",
  >     "minimum": -180,
  >     "maximum": 180
  >   }
  > }
  > }';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT JSON_SCHEMA_VALIDATION_REPORT(@schema, @document);
+-----+
| JSON_SCHEMA_VALIDATION_REPORT(@schema, @document) |
+-----+
| {"valid": true} |
+-----+
1 row in set (0.00 sec)
```

12.18.8 JSON ユーティリティ関数

このセクションでは、JSON 値に作用するユーティリティ関数、または JSON 値として解析できる文字列について説明します。`JSON_PRETTY()` は、読みやすい形式で JSON 値を出力します。`JSON_STORAGE_SIZE()` および `JSON_STORAGE_FREE()` では、特定の JSON 値によって使用されるストレージ領域の量と、部分的な更新後に JSON カラムに残っている領域の量がそれぞれ表示されます。

• `JSON_PRETTY(json_val)`

PHP および他の言語やデータベースシステムで実装されているものと同様の JSON 値のプリティプリントを提供します。指定する値は、JSON 値または JSON 値の有効な文字列表現である必要があります。この値に存在する余分な空白および改行は、出力には影響しません。`NULL` 値の場合、関数は `NULL` を戻します。値が JSON ドキュメントでない場合、または JSON ドキュメントとして解析できない場合、関数はエラーで失敗します。

この関数からの出力の書式設定は、次のルールに従います:

- 各配列要素またはオブジェクトメンバーは、親と比較して 1 つの追加レベルでインデントされた個別の行に表示されます。
- インデントの各レベルでは、先頭に 2 つのスペースが追加されます。
- 2 つの要素またはメンバーを区切る改行の前に、個々の配列要素またはオブジェクトメンバーを区切るカンマが出力されます。
- オブジェクトメンバーのキーと値はコロンで区切られ、その後に空白 (': ') が続きます。

- 空のオブジェクトまたは配列は単一行に出力されます。開きカッコと閉じカッコの間にスペースは印刷されません。
- 文字列スカラーおよびキー名の特殊文字は、`JSON_QUOTE()` 関数で 사용되는ものと同じルールを使用してエスケープされます。

```
mysql> SELECT JSON_PRETTY('123'); # scalar
+-----+
| JSON_PRETTY('123') |
+-----+
| 123                |
+-----+

mysql> SELECT JSON_PRETTY('[1,3,5]'); # array
+-----+
| JSON_PRETTY('[1,3,5]') |
+-----+
| [
| 1,
| 3,
| 5
| ] |
+-----+

mysql> SELECT JSON_PRETTY({'a':"10","b":"15","x":"25"}); # object
+-----+
| JSON_PRETTY({'a':"10","b":"15","x":"25"}) |
+-----+
| {
| "a": "10",
| "b": "15",
| "x": "25"
| } |
+-----+

mysql> SELECT JSON_PRETTY(['a',1,{"key1":
> "value1"},"5", "77",
> {"key2":["value3","valueX",
> "valueY"]},"j", "2" ])\G # nested arrays and objects
***** 1. row *****
JSON_PRETTY(['a',1,{"key1":
"value1"},"5", "77",
{"key2":["value3","valueX",
"valueY"]},"j", "2" ]): [
"a",
1,
{
"key1": "value1"
},
"5",
"77",
{
"key2": [
"value3",
"valueX",
"valueY"
]
},
"j",
"2"
]
```

- `JSON_STORAGE_FREE(json_val)`

`JSON` カラム値の場合、この関数は、`JSON_SET()`、`JSON_REPLACE()` または `JSON_REMOVE()` を使用して更新された後にバイナリ表現で解放された記憶領域の量を示します。引数には、有効な `JSON` ドキュメントまたは文字列を指定することもできます。この文字列は、リテラル値として解析することも、ユーザー変数の値として解析することもできます。この場合、関数は 0 を返します。引数が、前述のように更新された `JSON` カラム値である場合、バイナリ表現が更新前よりも少ない領域を占めるように、ゼロ以外の正の値を返します。バイナリ表現が以前

と同じかそれより大きくなるように更新された JSON カラムの場合、または更新で部分更新を利用できなかった場合は 0 を返し、引数が NULL の場合は NULL を返します。

`json_val` が NULL ではなく、有効な JSON ドキュメントでも正常に解析できない場合は、エラーが発生します。

この例では、JSON カラムを含むテーブルを作成し、JSON オブジェクトを含む行を挿入します：

```
mysql> CREATE TABLE jtable (jcol JSON);
Query OK, 0 rows affected (0.38 sec)

mysql> INSERT INTO jtable VALUES
-> ({"a": 10, "b": "wxyz", "c": "[true, false]"});
Query OK, 1 row affected (0.04 sec)

mysql> SELECT * FROM jtable;
+-----+
| jcol          |
+-----+
| {"a": 10, "b": "wxyz", "c": "[true, false]"} |
+-----+
1 row in set (0.00 sec)
```

ここでは、部分更新を実行できるように、`JSON_SET()` を使用してカラム値を更新します。この場合、`c` キー (配列カラム `[true, false]`) が指す値を、使用する領域 (整数 1) が少なくなる値に置き換えます：

```
mysql> UPDATE jtable
-> SET jcol = JSON_SET(jcol, "$.a", 10, "$.b", "wxyz", "$.c", 1);
Query OK, 1 row affected (0.03 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT * FROM jtable;
+-----+
| jcol          |
+-----+
| {"a": 10, "b": "wxyz", "c": 1} |
+-----+
1 row in set (0.00 sec)

mysql> SELECT JSON_STORAGE_FREE(jcol) FROM jtable;
+-----+
| JSON_STORAGE_FREE(jcol) |
+-----+
|          14 |
+-----+
1 row in set (0.00 sec)
```

この空き領域に対する連続した部分更新の影響は累積されます (この例では、`JSON_SET()` を使用して、キー `b` を持つ値によって取得される領域を削減し、その他の変更は行いません)：

```
mysql> UPDATE jtable
-> SET jcol = JSON_SET(jcol, "$.a", 10, "$.b", "wx", "$.c", 1);
Query OK, 1 row affected (0.03 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT JSON_STORAGE_FREE(jcol) FROM jtable;
+-----+
| JSON_STORAGE_FREE(jcol) |
+-----+
|          16 |
+-----+
1 row in set (0.00 sec)
```

`JSON_SET()`、`JSON_REPLACE()` または `JSON_REMOVE()` を使用せずにカラムを更新することは、オプティマイザがインプレースで更新を実行できないことを意味します。この場合、`JSON_STORAGE_FREE()` は次のように 0 を返します：

```
mysql> UPDATE jtable SET jcol = '{"a": 10, "b": 1}';
Query OK, 1 row affected (0.05 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT JSON_STORAGE_FREE(jcol) FROM jtable;
```

```

+-----+
| JSON_STORAGE_FREE(jcol) |
+-----+
|          0 |
+-----+
1 row in set (0.00 sec)

```

JSON ドキュメントの部分更新は、カラム値に対してのみ実行できます。JSON 値を格納するユーザー変数の場合、`JSON_SET()` を使用して更新を実行しても、値は常に完全に置換されます:

```

mysql> SET @j = '{"a": 10, "b": "wxyz", "c": "[true, false]"}';
Query OK, 0 rows affected (0.00 sec)

mysql> SET @j = JSON_SET(@j, '$.a', 10, '$.b', 'wxyz', '$.c', '1');
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @j, JSON_STORAGE_FREE(@j) AS Free;
+-----+-----+
| @j          | Free |
+-----+-----+
| '{"a": 10, "b": "wxyz", "c": "1"}' | 0 |
+-----+-----+
1 row in set (0.00 sec)

```

JSON リテラルの場合、この関数は常に 0 を返します:

```

mysql> SELECT JSON_STORAGE_FREE('{"a": 10, "b": "wxyz", "c": "1"}') AS Free;
+-----+
| Free |
+-----+
| 0 |
+-----+
1 row in set (0.00 sec)

```

• `JSON_STORAGE_SIZE(json_val)`

この関数は、JSON ドキュメントのバイナリ表現の格納に使用されるバイト数を返します。引数が `JSON` カラムの場合、これは、後で実行された部分更新の前に、JSON ドキュメントをカラムに挿入したときに格納するために使用される領域です。`json_val` は、有効な JSON ドキュメントまたは解析可能な文字列である必要があります。文字列の場合、関数は、文字列を JSON として解析してバイナリに変換することで作成される JSON バイナリ表現の記憶領域の量を返します。引数が `NULL` の場合、`NULL` を返します。

`json_val` が `NULL` ではなく、JSON ドキュメントとして正常に解析できない場合、エラーが発生します。

`JSON` カラムを引数として使用する場合のこの関数の動作を説明するために、次に示すように、`JSON` カラム `jcol` を含む `jtable` という名前のテーブルを作成し、JSON 値をテーブルに挿入してから、`JSON_STORAGE_SIZE()` でこのカラムによって使用される記憶領域を取得します:

```

mysql> CREATE TABLE jtable (jcol JSON);
Query OK, 0 rows affected (0.42 sec)

mysql> INSERT INTO jtable VALUES
-> ('{"a": 1000, "b": "wxyz", "c": "[1, 3, 5, 7]"}');
Query OK, 1 row affected (0.04 sec)

mysql> SELECT
-> jcol,
-> JSON_STORAGE_SIZE(jcol) AS Size,
-> JSON_STORAGE_FREE(jcol) AS Free
-> FROM jtable;
+-----+-----+-----+
| jcol          | Size | Free |
+-----+-----+-----+
| '{"a": 1000, "b": "wxyz", "c": "[1, 3, 5, 7]"}' | 47 | 0 |
+-----+-----+-----+

```



```
1 row in set (0.00 sec)
```

`JSON_STORAGE_SIZE()` の出力によると、カラムに挿入される JSON ドキュメントは 47 バイトを占めます。また、`JSON_STORAGE_FREE()` を使用して、カラムの以前の部分更新によって解放された領域の量も確認しました。更新がまだ実行されていないため、これは予想どおり 0 です。

次に、`jcol` に格納されているドキュメントを部分的に更新するテーブルで `UPDATE` を実行し、次に示すように結果をテストします:

```
mysql> UPDATE jtable SET jcol =
-> JSON_SET(jcol, "$.b", "a");
Query OK, 1 row affected (0.04 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT
-> jcol,
-> JSON_STORAGE_SIZE(jcol) AS Size,
-> JSON_STORAGE_FREE(jcol) AS Free
-> FROM jtable;
+-----+-----+
| jcol                | Size | Free |
+-----+-----+
| {"a": 1000, "b": "a", "c": "[1, 3, 5, 7]"} | 47 | 3 |
+-----+-----+
1 row in set (0.00 sec)
```

前のクエリーで `JSON_STORAGE_FREE()` によって返された値は、JSON ドキュメントの部分更新が実行され、格納に使用された 3 バイトの領域が解放されたことを示しています。 `JSON_STORAGE_SIZE()` によって返される結果は、部分更新によって変更されません。

部分更新は、`JSON_SET()`、`JSON_REPLACE()` または `JSON_REMOVE()` を使用した更新でサポートされています。 `JSON` カラムへの値の直接割当ては部分的には更新できません。このような更新の後、`JSON_STORAGE_SIZE()` には常に、新しく設定された値に使用される記憶域が表示されます:

```
mysql> UPDATE jtable
mysql> SET jcol = '{"a": 4.55, "b": "wxyz", "c": "[true, false]"}';
Query OK, 1 row affected (0.04 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT
-> jcol,
-> JSON_STORAGE_SIZE(jcol) AS Size,
-> JSON_STORAGE_FREE(jcol) AS Free
-> FROM jtable;
+-----+-----+
| jcol                | Size | Free |
+-----+-----+
| {"a": 4.55, "b": "wxyz", "c": "[true, false]"} | 56 | 0 |
+-----+-----+
1 row in set (0.00 sec)
```

JSON ユーザー変数は部分的に更新できません。つまり、この関数は、JSON ドキュメントをユーザー変数に格納するために現在使用されている領域を常に表示します:

```
mysql> SET @j = '[100, "sakila", [1, 3, 5], 425.05]';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @j, JSON_STORAGE_SIZE(@j) AS Size;
+-----+-----+
| @j                | Size |
+-----+-----+
| [100, "sakila", [1, 3, 5], 425.05] | 45 |
+-----+-----+
1 row in set (0.00 sec)

mysql> SET @j = JSON_SET(@j, '$[1]', "json");
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @j, JSON_STORAGE_SIZE(@j) AS Size;
+-----+-----+
| @j                | Size |
+-----+-----+
```

```
+-----+-----+
| [100, "json", [1, 3, 5], 425.05] | 43 |
+-----+-----+
1 row in set (0.00 sec)

mysql> SET @j = JSON_SET(@j, '$[2][0]', JSON_ARRAY(10, 20, 30));
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @j, JSON_STORAGE_SIZE(@j) AS Size;
+-----+-----+
| @j | Size |
+-----+-----+
| [100, "json", [[10, 20, 30], 3, 5], 425.05] | 56 |
+-----+-----+
1 row in set (0.00 sec)
```

JSON リテラルの場合、この関数は常に現在使用されている記憶領域を戻します:

```
mysql> SELECT
-> JSON_STORAGE_SIZE(['100, "sakila", [1, 3, 5], 425.05']) AS A,
-> JSON_STORAGE_SIZE({'a': 1000, "b": "a", "c": "[1, 3, 5, 7]'}) AS B,
-> JSON_STORAGE_SIZE({'a': 1000, "b": "wxyz", "c": "[1, 3, 5, 7]'}) AS C,
-> JSON_STORAGE_SIZE(['100, "json", [[10, 20, 30], 3, 5], 425.05']) AS D;
+-----+-----+
| A | B | C | D |
+-----+-----+
| 45 | 44 | 47 | 56 |
+-----+-----+
1 row in set (0.00 sec)
```

12.19 グローバルトランザクション識別子 (GTID) で使用される機能

このセクションで説明する機能は GTID ベースのレプリケーションで使用されます。これらの関数はすべて GTID セットの文字列表現を引数として取ることに注意してください。そのため、GTID セットと一緒に使用する場合は、常に引用符で囲む必要があります。詳しくは [GTID セット](#) をご覧ください。

2 つの GTID セットの結合は、単にカンマを挿入して結合された文字列として表現されたものです。言い換えると、ここで作成した関数と同様に、非常に単純な関数を定義すれば、GTID セットの結合を取得できます。

```
CREATE FUNCTION GTID_UNION(g1 TEXT, g2 TEXT)
RETURNS TEXT DETERMINISTIC
RETURN CONCAT(g1,',',g2);
```

GTID の詳細およびこれらの GTID 関数を実際を使用する方法については、[セクション 17.1.3 「グローバルトランザクション識別子を使用したレプリケーション」](#) を参照してください。

表 12.24 「GTID 関数」

名前	説明	非推奨
GTID_SUBSET()	サブセット内のすべての GTID がセット内にもある場合は、true を返します。そうでない場合は、false を返します。	
GTID_SUBTRACT()	セット内の GTID のうち、サブセット内にはないものをすべてを返します。	
WAIT_FOR_EXECUTED_GTID_SET()	指定された GTID がレプリカで実行されるまで待機します。	
WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS()	WAIT_FOR_EXECUTED_GTID_SET() の使用。	8.0.18

- [GTID_SUBSET\(set1,set2\)](#)

グローバルトランザクション識別子 *set1* および *set2* の 2 つのセットが指定されている場合、*set1* 内のすべての GTID も *set2* 内にあると true を返します。それ以外の場合は、false を返します。

この関数で使用される GTID セットは、次の例で示すように文字列で表現されます。

```
mysql> SELECT GTID_SUBSET('3E11FA47-71CA-11E1-9E33-C80AA9429562:23',
-> '3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57')\G
***** 1. row *****
GTID_SUBSET('3E11FA47-71CA-11E1-9E33-C80AA9429562:23',
'3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57'): 1
1 row in set (0.00 sec)

mysql> SELECT GTID_SUBSET('3E11FA47-71CA-11E1-9E33-C80AA9429562:23-25',
-> '3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57')\G
***** 1. row *****
GTID_SUBSET('3E11FA47-71CA-11E1-9E33-C80AA9429562:23-25',
'3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57'): 1
1 row in set (0.00 sec)

mysql> SELECT GTID_SUBSET('3E11FA47-71CA-11E1-9E33-C80AA9429562:20-25',
-> '3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57')\G
***** 1. row *****
GTID_SUBSET('3E11FA47-71CA-11E1-9E33-C80AA9429562:20-25',
'3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57'): 0
1 row in set (0.00 sec)
```

- `GTID_SUBTRACT(set1,set2)`

グローバルトランザクション識別子 `set1` および `set2` の 2 つのセットがある場合、`set2` にない GTID のみを `set1` から返します。

この関数で使用される GTID セットはすべて文字列で表現されるため、次の例で示すように引用符で囲む必要があります。

```
mysql> SELECT GTID_SUBTRACT('3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57',
-> '3E11FA47-71CA-11E1-9E33-C80AA9429562:21')\G
***** 1. row *****
GTID_SUBTRACT('3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57',
'3E11FA47-71CA-11E1-9E33-C80AA9429562:21'): 3e11fa47-71ca-11e1-9e33-c80aa9429562:22-57
1 row in set (0.00 sec)

mysql> SELECT GTID_SUBTRACT('3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57',
-> '3E11FA47-71CA-11E1-9E33-C80AA9429562:20-25')\G
***** 1. row *****
GTID_SUBTRACT('3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57',
'3E11FA47-71CA-11E1-9E33-C80AA9429562:20-25'): 3e11fa47-71ca-11e1-9e33-c80aa9429562:26-57
1 row in set (0.00 sec)

mysql> SELECT GTID_SUBTRACT('3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57',
-> '3E11FA47-71CA-11E1-9E33-C80AA9429562:23-24')\G
***** 1. row *****
GTID_SUBTRACT('3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57',
'3E11FA47-71CA-11E1-9E33-C80AA9429562:23-24'): 3e11fa47-71ca-11e1-9e33-c80aa9429562:21-22:25-57
1 row in set (0.01 sec)
```

- `WAIT_FOR_EXECUTED_GTID_SET(gtid_set[, timeout])`

サーバーが、グローバルトランザクション識別子が `gtid_set` に含まれているすべてのトランザクションを適用する (つまり、`GTID_SUBSET(gtid_subset, @@GLOBAL.gtid_executed)` が保持する) まで待機します。GTID セットの定義については、[セクション 17.1.3.1 「GTID 形式および格納」](#) を参照してください。

タイムアウトが指定され、GTID セット内のすべてのトランザクションが適用されるまで `timeout` 秒が経過すると、関数は待機を停止します。`timeout` はオプションで、デフォルトのタイムアウトは 0 秒です。この場合、GTID セット内のすべてのトランザクションが適用されるまで、関数は常に待機します。

`WAIT_FOR_EXECUTED_GTID_SET()` は、すべてのレプリケーションチャンネルおよびユーザークライアントから到着したトランザクションを含め、サーバーに適用されている GTID をすべて監視します。レプリケーションチャンネルが起動しているか停止しているかは考慮されません。

詳細は、[セクション 17.1.3 「グローバルトランザクション識別子を使用したレプリケーション」](#) を参照してください。

この関数で使用される GTID セットは、次の例で示すように文字列で表現されるため、引用符で囲む必要があります。

```
mysql> SELECT WAIT_FOR_EXECUTED_GTID_SET('3E11FA47-71CA-11E1-9E33-C80AA9429562:1-5');
-> 0
```

GTID セットの構文の説明については、[セクション17.1.3.1「GTID 形式および格納」](#)を参照してください。

`WAIT_FOR_EXECUTED_GTID_SET()` の場合、戻り値はクエリーの状態です。0 は成功を表し、1 はタイムアウトを表します。その他の障害では、エラーが生成されます。

GTID が適用されるのを待機するためにクライアントがこの機能を使用している間は、`gtid_mode` を OFF に変更できません。

- `WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS(gtid_set[, timeout][,channel])`

`WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS()` は非推奨です。かわりに `WAIT_FOR_EXECUTED_GTID_SET()` を使用してください。これは、指定したトランザクションがサーバーに到着するレプリケーションチャンネルまたはユーザークライアントに関係なく機能します。

12.20 集計関数

集計関数は、値のセットを操作します。多くの場合、値をサブセットにグループ化するために `GROUP BY` 句とともに使用されます。このセクションでは、ほとんどの集計関数について説明します。ジオメトリ値を操作する集計関数の詳細は、[セクション12.17.12「空間集計関数」](#)を参照してください。

12.20.1 集計関数の説明

このセクションでは、値のセットを操作する集計関数について説明します。多くの場合、値をサブセットにグループ化するために `GROUP BY` 句とともに使用されます。

表 12.25 「集計関数」

名前	説明
<code>AVG()</code>	引数の平均値を返します
<code>BIT_AND()</code>	ビット単位 AND を返します
<code>BIT_OR()</code>	ビット単位の OR を返します
<code>BIT_XOR()</code>	ビット単位 XOR を返します
<code>COUNT()</code>	返された行数のカウンートを返します
<code>COUNT(DISTINCT)</code>	異なる値のカウンートを返します
<code>GROUP_CONCAT()</code>	連結された文字列を返します
<code>JSON_ARRAYAGG()</code>	結果セットを単一の JSON 配列として返します
<code>JSON_OBJECTAGG()</code>	結果セットを単一の JSON オブジェクトとして返します
<code>MAX()</code>	最大値を返します
<code>MIN()</code>	最小値を返します
<code>STD()</code>	母標準偏差を返します
<code>STDDEV()</code>	母標準偏差を返します
<code>STDDEV_POP()</code>	母標準偏差を返します
<code>STDDEV_SAMP()</code>	標本標準偏差を返します
<code>SUM()</code>	集計を返します
<code>VAR_POP()</code>	母標準分散を返します
<code>VAR_SAMP()</code>	標本分散を返します
<code>VARIANCE()</code>	母標準分散を返します

特に明記されていないかぎり、集計関数は `NULL` 値を無視します。

GROUP BY 句を含まないステートメントで集計関数を使用する場合、すべての行でのグループ化と同等です。詳細は、[セクション12.20.3「MySQLでのGROUP BYの処理」](#)を参照してください。

ほとんどの集計関数は、ウィンドウ関数として使用できます。この方法で使用できるものは、オプションの **OVER** 句を表す `[over_clause]` の構文の説明で示されます。`over_clause` は [セクション12.21.2「Window関数の概念と構文」](#) で説明されており、ウィンドウ関数の使用方法に関するその他の情報も含まれています。

数値の引数の場合、分散および標準偏差関数が **DOUBLE** 値を返します。**SUM()** および **AVG()** 関数は、正確な値の引数 (整数または **DECIMAL**) の場合は **DECIMAL** 値を返し、近似値の引数 (**FLOAT** または **DOUBLE**) の場合は **DOUBLE** 値を返します。

SUM() および **AVG()** 集計関数は時間値を扱いません。(これらは値を数字に変換するので、最初の数字以外の文字のあとのすべての情報が失われます。) この問題を回避するには、数値単位に変換し、集計操作を実行してから、時間値に戻します。例:

```
SELECT SEC_TO_TIME(SUM(TIME_TO_SEC(time_col))) FROM tbl_name;  
SELECT FROM_DAYS(SUM(TO_DAYS(date_col))) FROM tbl_name;
```

数値引数を取る **SUM()** や **AVG()** などの関数は、必要に応じて引数を数値にキャストします。**SET** や **ENUM** 値の場合、キャスト演算でベースとなる数値が使用されます。

BIT_AND()、**BIT_OR()** および **BIT_XOR()** の集計関数は、ビット操作を実行します。MySQL 8.0 より前では、ビット関数および演算子には **BIGINT** (64-bit 整数) 引数が必要であり、**BIGINT** 値が返されるため、最大範囲は 64 ビットでした。操作の実行前に **BIGINT** 以外の引数が **BIGINT** に変換され、切捨てが発生する可能性があります。

MySQL 8.0 では、ビット関数および演算子はバイナリ文字列型の引数 (**BINARY**、**VARBINARY** および **BLOB** 型) を許可し、同様の型の値を返します。これにより、引数を受け取り、64 ビットを超える戻り値を生成できます。ビット操作の引数評価および結果タイプの詳細は、[セクション12.13「ビット関数と演算子」](#) の概要を参照してください。

• **AVG([DISTINCT] expr) [over_clause]**

`expr` の平均値を返します。**DISTINCT** オプションを使用すると、個別の `expr` 値の平均を返すことができます。

一致する行がない場合、**AVG()** は **NULL** を返します。

`over_clause` が存在する場合、この関数はウィンドウ関数として実行されます。`over_clause` は、[セクション12.21.2「Window関数の概念と構文」](#) で説明されているように、**DISTINCT** では使用できません。

```
mysql> SELECT student_name, AVG(test_score)  
FROM student  
GROUP BY student_name;
```

• **BIT_AND(expr) [over_clause]**

`expr` 内のすべてのビットのビット単位の **AND** を返します。

結果の型は、関数の引数値がバイナリ文字列として評価されるか、数値として評価されるかによって異なります:

- バイナリ文字列の評価は、引数値がバイナリ文字列型で、引数が 16 進数リテラル、ビットリテラルまたは **NULL** リテラルでない場合に発生します。それ以外の場合は数値の評価が行われ、必要に応じて引数値が符号なし 64 ビット整数に変換されます。
- バイナリ文字列の評価では、引数値と同じ長さのバイナリ文字列が生成されます。引数値の長さが等しくない場合は、**ER_INVALID_BITWISE_OPERANDS_SIZE** エラーが発生します。引数のサイズが 511 バイトを超えると、**ER_INVALID_BITWISE_AGGREGATE_OPERANDS_SIZE** エラーが発生します。数値評価では符号なし 64 ビット整数が生成されます。

一致する行がない場合、**BIT_AND()** は引数値と同じ長さのニュートラル値 (すべて 1 に設定されたビット) を返します。

すべての値が **NULL** でない限り、**NULL** 値は結果に影響しません。その場合、結果は引数値と同じ長さの中立値になります。

引数の評価および結果タイプの詳細は、[セクション12.13「ビット関数と演算子」](#) の概要に関する説明を参照してください。

MySQL 8.0.12 では、`over_clause` が存在する場合、この関数はウィンドウ関数として実行されます。`over_clause` は [セクション12.21.2「Window 関数の概念と構文」](#) で説明されているとおりです。

- `BIT_OR(expr) [over_clause]`

`expr` 内のすべてのビットのビット単位の `OR` を返します。

結果の型は、関数の引数値がバイナリ文字列として評価されるか、数値として評価されるかによって異なります：

- バイナリ文字列の評価は、引数値がバイナリ文字列型で、引数が 16 進数リテラル、ビットリテラルまたは `NULL` リテラルでない場合に発生します。それ以外の場合は数値の評価が行われ、必要に応じて引数値が符号なし 64 ビット整数に変換されます。
- バイナリ文字列の評価では、引数値と同じ長さのバイナリ文字列が生成されます。引数値の長さが等しくない場合は、`ER_INVALID_BITWISE_OPERANDS_SIZE` エラーが発生します。引数のサイズが 511 バイトを超えると、`ER_INVALID_BITWISE_AGGREGATE_OPERANDS_SIZE` エラーが発生します。数値評価では符号なし 64 ビット整数が生成されます。

一致する行がない場合、`BIT_OR()` は引数値と同じ長さのニュートラル値 (すべて 0 に設定されたビット) を返しません。

すべての値が `NULL` でないかぎり、`NULL` 値は結果に影響しません。その場合、結果は引数値と同じ長さの中立値になります。

引数の評価および結果タイプの詳細は、[セクション12.13「ビット関数と演算子」](#) の概要に関する説明を参照してください。

MySQL 8.0.12 では、`over_clause` が存在する場合、この関数はウィンドウ関数として実行されます。`over_clause` は [セクション12.21.2「Window 関数の概念と構文」](#) で説明されているとおりです。

- `BIT_XOR(expr) [over_clause]`

`expr` 内のすべてのビットのビット単位の `XOR` を返します。

結果の型は、関数の引数値がバイナリ文字列として評価されるか、数値として評価されるかによって異なります：

- バイナリ文字列の評価は、引数値がバイナリ文字列型で、引数が 16 進数リテラル、ビットリテラルまたは `NULL` リテラルでない場合に発生します。それ以外の場合は数値の評価が行われ、必要に応じて引数値が符号なし 64 ビット整数に変換されます。
- バイナリ文字列の評価では、引数値と同じ長さのバイナリ文字列が生成されます。引数値の長さが等しくない場合は、`ER_INVALID_BITWISE_OPERANDS_SIZE` エラーが発生します。引数のサイズが 511 バイトを超えると、`ER_INVALID_BITWISE_AGGREGATE_OPERANDS_SIZE` エラーが発生します。数値評価では符号なし 64 ビット整数が生成されます。

一致する行がない場合、`BIT_XOR()` は引数値と同じ長さのニュートラル値 (すべて 0 に設定されたビット) を返しません。

すべての値が `NULL` でないかぎり、`NULL` 値は結果に影響しません。その場合、結果は引数値と同じ長さの中立値になります。

引数の評価および結果タイプの詳細は、[セクション12.13「ビット関数と演算子」](#) の概要に関する説明を参照してください。

MySQL 8.0.12 では、`over_clause` が存在する場合、この関数はウィンドウ関数として実行されます。`over_clause` は [セクション12.21.2「Window 関数の概念と構文」](#) で説明されているとおりです。

- `COUNT(expr) [over_clause]`

`SELECT` ステートメントで取得された行に含まれる `expr` の非 `NULL` 値の数を返します。結果は `BIGINT` 値になります。

一致する行がない場合、`COUNT()` は 0 を返します。

`over_clause` が存在する場合、この関数はウィンドウ関数として実行されます。`over_clause` については、[セクション12.21.2「Window 関数の概念と構文」](#)を参照してください。

```
mysql> SELECT student.student_name,COUNT(*)
FROM student,course
WHERE student.student_id=course.student_id
GROUP BY student_name;
```

`COUNT(*)` は、`NULL` 値が含まれるかどうかに関係なく、取得された行数を返すという点で少し異なります。

InnoDB などのトランザクションストレージエンジンでは、正確な行数の格納に問題があります。複数のトランザクションが同時に発生する可能性があり、それぞれがカウントに影響する可能性があります。

並列トランザクションでは同時にさまざまな数の行が「参照」される可能性があるため、InnoDB のテーブルには、行の内部的なカウントが保持されません。したがって、`SELECT COUNT(*)` ステートメントでは、現在のトランザクションで参照可能な行のみがカウントされます。

MySQL 8.0.13 では、`WHERE`、`GROUP BY` などの追加句がない場合、InnoDB テーブルの `SELECT COUNT(*) FROM tbl_name` クエリーパフォーマンスはシングルスレッドワークロード用に最適化されます。

InnoDB は、インデックスまたはオプティマイザヒントがオプティマイザに別のインデックスを使用するように指示しないかぎり、使用可能な最小のセカンダリインデックスを横断することで `SELECT COUNT(*)` ステートメントを処理します。セカンダリインデックスが存在しない場合、InnoDB はクラスタインデックスをスキャンして `SELECT COUNT(*)` ステートメントを処理します。

インデックスレコードがバッファプールに完全に含まれていない場合、`SELECT COUNT(*)` ステートメントの処理には時間がかかります。カウントを高速化するには、カウンタテーブルを作成し、挿入および削除に従ってカウンタテーブルを更新できるようにします。ただし、何千もの同時トランザクションが同じカウンタテーブルへの更新を開始している状況では、この方法は適切にスケールされない場合があります。概算の行数で十分な場合は、`SHOW TABLE STATUS` を使用します。

InnoDB は、`SELECT COUNT(*)` と `SELECT COUNT(1)` の操作を同じ方法で処理します。パフォーマンスに違いはありません。

MyISAM テーブルの場合、`COUNT(*)` は、`SELECT` があるテーブルから取得し、他のカラムが取得されず、`WHERE` 句がない場合に非常に迅速に返すように最適化されます。例:

```
mysql> SELECT COUNT(*) FROM student;
```

この最適化は MyISAM テーブルにのみ適用されます。これは、正確な行数がこのストレージエンジン用に格納され、非常に迅速にアクセスできるためです。`COUNT(1)` は、最初のカラムが `NOT NULL` として定義されている場合のみ、同じ最適化の対象となります。

- `COUNT(DISTINCT expr,[expr...])`

さまざまな非 `NULL` `expr` 値を含む行数を返します。

一致する行がない場合、`COUNT(DISTINCT)` は `0` を返します。

```
mysql> SELECT COUNT(DISTINCT results) FROM student;
```

MySQL では、式のリストを指定することで、`NULL` が含まれない個別の式の組み合わせ数を取得できます。標準 SQL では、`COUNT(DISTINCT ...)` 内部で、すべての式の連結を行う必要があります。

- `GROUP_CONCAT(expr)`

この関数は、グループから連結された非 `NULL` 値を含む文字列の結果を返します。非 `NULL` 値がない場合は、`NULL` を返します。完全な構文は次のとおりです。

```
GROUP_CONCAT([DISTINCT] expr [,expr ...]
[ORDER BY {unsigned_integer | col_name | expr}
[ASC | DESC] [,col_name ...]]
[SEPARATOR str_val])
```

```
mysql> SELECT student_name,
```

```
GROUP_CONCAT(test_score)
FROM student
GROUP BY student_name;
```

または:

```
mysql> SELECT student_name,
GROUP_CONCAT(DISTINCT test_score
ORDER BY test_score DESC SEPARATOR '')
FROM student
GROUP BY student_name;
```

MySQL では、式の組み合わせを連結した値を取得できます。重複する値を除去するには、`DISTINCT` 句を使用します。結果の値をソートするには、`ORDER BY` 句を使用します。逆順でソートするには、`ORDER BY` 句のソートするカラムの名前に `DESC` (降順) キーワードを追加します。デフォルトは昇順です。これは、`ASC` キーワードを使用することで明示的に指定できます。グループ内の値間のデフォルトの区切り文字は、カンマ (,) です。区切り文字を明示的に指定するには、`SEPARATOR` に続けて、グループ値の間に挿入される文字列リテラル値を指定します。区切り文字を完全に除去するには、`SEPARATOR ''` を指定します。

結果は、`group_concat_max_len` システム変数で指定された最大長まで切り捨てられます。その変数のデフォルト値は 1024 です。さらに高い値にも設定できますが、戻り値の有効な最大長は、`max_allowed_packet` の値によって制約されます。実行時に `group_concat_max_len` の値を変更するための構文は、次のとおりです。ここで、`val` は符号なし整数です。

```
SET [GLOBAL | SESSION] group_concat_max_len = val;
```

戻り値は、引数が非バイナリとバイナリのどちらの文字列であるのかに応じて、非バイナリ文字列またはバイナリ文字列になります。結果の型は、`group_concat_max_len` が 512 以下の場合 (この場合、結果の型は `VARCHAR` または `VARBINARY` です) を除いて、`TEXT` または `BLOB` です。

`CONCAT()` および `CONCAT_WS()`: [セクション12.8「文字列関数および演算子」](#) も参照してください。

- `JSON_ARRAYAGG(col_or_expr) [over_clause]`

結果セットを、要素が行で構成される単一の `JSON` 配列として集計します。この配列の要素の順序が定義されていません。この関数は、単一の値に評価されるカラムまたは式に対して機能します。結果に行が含まれていない場合、またはエラーが発生した場合は、`NULL` を返します。

MySQL 8.0.14 では、`over_clause` が存在する場合、この関数はウィンドウ関数として実行されます。`over_clause` は [セクション12.21.2「Window 関数の概念と構文」](#) で説明されているとおりです。

```
mysql> SELECT o_id, attribute, value FROM t3;
+----+-----+-----+
| o_id | attribute | value |
+----+-----+-----+
| 2 | color | red |
| 2 | fabric | silk |
| 3 | color | green |
| 3 | shape | square |
+----+-----+-----+
4 rows in set (0.00 sec)

mysql> SELECT o_id, JSON_ARRAYAGG(attribute) AS attributes
> FROM t3 GROUP BY o_id;
+----+-----+
| o_id | attributes |
+----+-----+
| 2 | ["color", "fabric"] |
| 3 | ["color", "shape"] |
+----+-----+
2 rows in set (0.00 sec)
```

- `JSON_OBJECTAGG(key, value) [over_clause]`

2つのカラム名または式を引数として取ります。最初のカラム名または式はキーとして使用され、2番目のカラム名または式は値として使用され、キーと値のペアを含む `JSON` オブジェクトを返します。結果に行が含まれていない場合、またはエラーが発生した場合は、`NULL` を返します。いずれかのキー名が `NULL` であるか、引数の数が2でない場合、エラーが発生します。

MySQL 8.0.14 では、`over_clause` が存在する場合、この関数はウィンドウ関数として実行されます。`over_clause` は [セクション12.21.2「Window 関数の概念と構文」](#) で説明されているとおりです。

```
mysql> SELECT o_id, attribute, value FROM t3;
+-----+-----+-----+
| o_id | attribute | value |
+-----+-----+-----+
| 2 | color | red |
| 2 | fabric | silk |
| 3 | color | green |
| 3 | shape | square |
+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> SELECT o_id, JSON_OBJECTAGG(attribute, value)
> FROM t3 GROUP BY o_id;
+-----+-----+
| o_id | JSON_OBJECTAGG(attribute, value) |
+-----+-----+
| 2 | {"color": "red", "fabric": "silk"} |
| 3 | {"color": "green", "shape": "square"} |
+-----+-----+
2 rows in set (0.00 sec)
```

重複キーの処理. この関数の結果が正規化されると、重複するキーを持つ値は破棄されます。重複キーを許可しない MySQL JSON データ型指定に従うと、返されるオブジェクト (「最後の重複キー優先」) でそのキーで最後に検出された値のみが使用されます。つまり、`SELECT` のカラムでこの関数を使用した結果は、戻される行の順序に依存する可能性があります、これは保証されていません。

ウィンドウ関数として使用する場合、フレーム内に重複するキーがあると、そのキーの最後の値のみが結果に表示されます。`ORDER BY` 仕様が値が特定の順序であることが保証されている場合、フレームの最後の行のキーの値は決定的です。そうでない場合、キーの結果値は非決定的です。

次について考えます。

```
mysql> CREATE TABLE t(c VARCHAR(10), i INT);
Query OK, 0 rows affected (0.33 sec)

mysql> INSERT INTO t VALUES ('key', 3), ('key', 4), ('key', 5);
Query OK, 3 rows affected (0.10 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> SELECT c, i FROM t;
+-----+-----+
| c | i |
+-----+-----+
| key | 3 |
| key | 4 |
| key | 5 |
+-----+-----+
3 rows in set (0.00 sec)

mysql> SELECT JSON_OBJECTAGG(c, i) FROM t;
+-----+
| JSON_OBJECTAGG(c, i) |
+-----+
| {"key": 5} |
+-----+
1 row in set (0.00 sec)

mysql> DELETE FROM t;
Query OK, 3 rows affected (0.08 sec)

mysql> INSERT INTO t VALUES ('key', 3), ('key', 5), ('key', 4);
Query OK, 3 rows affected (0.06 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> SELECT c, i FROM t;
+-----+-----+
| c | i |
+-----+-----+
```

```
| key | 3 |
| key | 5 |
| key | 4 |
+-----+
3 rows in set (0.00 sec)
```

```
mysql> SELECT JSON_OBJECTAGG(c, i) FROM t;
+-----+
| JSON_OBJECTAGG(c, i) |
+-----+
| {"key": 4}           |
+-----+
1 row in set (0.00 sec)
```

最後のクエリーから選択されたキーは非決定的です。特定のキーの順序付けが必要な場合は、フレーム行に特定の順序を適用する **ORDER BY** 仕様に **OVER** 句を含めることで、**JSON_OBJECTAGG()** をウィンドウ関数として起動できます。次の例は、いくつかの異なるフレーム仕様について、**ORDER BY** の有無に応じて何が起きるかを示しています。

ORDER BY がない場合、フレームはパーティション全体です:

```
mysql> SELECT JSON_OBJECTAGG(c, i)
         OVER () AS json_object FROM t;
+-----+
| json_object |
+-----+
| {"key": 4} |
| {"key": 4} |
| {"key": 4} |
+-----+
```

ORDER BY では、フレームはデフォルトの **RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW** (昇順と降順の両方) です:

```
mysql> SELECT JSON_OBJECTAGG(c, i)
         OVER (ORDER BY i) AS json_object FROM t;
+-----+
| json_object |
+-----+
| {"key": 3} |
| {"key": 4} |
| {"key": 5} |
+-----+
mysql> SELECT JSON_OBJECTAGG(c, i)
         OVER (ORDER BY i DESC) AS json_object FROM t;
+-----+
| json_object |
+-----+
| {"key": 5} |
| {"key": 4} |
| {"key": 3} |
+-----+
```

ORDER BY およびパーティション全体の明示的なフレームを使用する場合:

```
mysql> SELECT JSON_OBJECTAGG(c, i)
         OVER (ORDER BY i
              ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
         AS json_object
         FROM t;
+-----+
| json_object |
+-----+
| {"key": 5} |
| {"key": 5} |
| {"key": 5} |
+-----+
```

特定のキー値 (最小値や最大値など) を返すには、適切なクエリーに **LIMIT** 句を含めます。例:

```
mysql> SELECT JSON_OBJECTAGG(c, i)
         OVER (ORDER BY i) AS json_object FROM t LIMIT 1;
```

```

+-----+
| json_object |
+-----+
| {"key": 3} |
+-----+
mysql> SELECT JSON_OBJECTAGG(c, i)
      OVER (ORDER BY i DESC) AS json_object FROM t LIMIT 1;
+-----+
| json_object |
+-----+
| {"key": 5} |
+-----+

```

追加情報および例については、[JSON 値の正規化](#)、[マージおよび自動ラップ](#)を参照してください。

- `MAX([DISTINCT] expr) [over_clause]`

`expr` の最大値を返します。 `MAX()` には、文字列の引数を指定できます。このような場合は、最大の文字列値が返されます。 [セクション8.3.1「MySQL のインデックスの使用の仕組み」](#)を参照してください。 `DISTINCT` キーワードを使用すると、個別の `expr` 値の最大を検索できます。ただし、`DISTINCT` を省略した場合と同じ結果が生成されません。

一致する行がない場合、`MAX()` は `NULL` を返します。

`over_clause` が存在する場合、この関数はウィンドウ関数として実行されます。 `over_clause` は、[セクション12.21.2「Window 関数の概念と構文」](#)で説明されているように、`DISTINCT` では使用できません。

```

mysql> SELECT student_name, MIN(test_score), MAX(test_score)
      FROM student
      GROUP BY student_name;

```

現在、MySQL の `MAX()` では、`ENUM` と `SET` カラムが、セット内の文字列の相対位置ではなく、文字列値について比較されます。これは、`ORDER BY` による比較方法とは異なります。

- `MIN([DISTINCT] expr) [over_clause]`

`expr` の最小値を返します。 `MIN()` には、文字列の引数を指定できます。このような場合は、最小の文字列値が返されます。 [セクション8.3.1「MySQL のインデックスの使用の仕組み」](#)を参照してください。 `DISTINCT` キーワードを使用すると、個別の `expr` 値の最小を検索できます。ただし、`DISTINCT` を省略した場合と同じ結果が生成されません。

一致する行がない場合、`MIN()` は `NULL` を返します。

`over_clause` が存在する場合、この関数はウィンドウ関数として実行されます。 `over_clause` は、[セクション12.21.2「Window 関数の概念と構文」](#)で説明されているように、`DISTINCT` では使用できません。

```

mysql> SELECT student_name, MIN(test_score), MAX(test_score)
      FROM student
      GROUP BY student_name;

```

現在、MySQL の `MIN()` では、`ENUM` と `SET` カラムが、セット内の文字列の相対位置ではなく、文字列値について比較されます。これは、`ORDER BY` による比較方法とは異なります。

- `STD(expr) [over_clause]`

`expr` の母標準偏差を返します。 `STD()` は、MySQL 拡張機能として提供される標準 SQL 関数 `STDDEV_POP()` のシノニムです。

一致する行がない場合、`STD()` は `NULL` を返します。

`over_clause` が存在する場合、この関数はウィンドウ関数として実行されます。 `over_clause` については、[セクション12.21.2「Window 関数の概念と構文」](#)を参照してください。

- `STDDEV(expr) [over_clause]`

`expr` の母標準偏差を返します。 `STDDEV()` は、Oracle との互換性のために提供されている標準 SQL 関数 `STDDEV_POP()` のシノニムです。

一致する行がない場合、`STDDEV()` は `NULL` を返します。

`over_clause` が存在する場合、この関数はウィンドウ関数として実行されます。`over_clause` については、[セクション12.21.2「Window 関数の概念と構文」](#) を参照してください。

- `STDDEV_POP(expr) [over_clause]`

`expr` の母標準偏差 (`VAR_POP()` の平方根) を返します。`STD()` または `STDDEV()` を使用することもできます。これらは同等ですが、標準 SQL ではありません。

一致する行がない場合、`STDDEV_POP()` は `NULL` を返します。

`over_clause` が存在する場合、この関数はウィンドウ関数として実行されます。`over_clause` については、[セクション12.21.2「Window 関数の概念と構文」](#) を参照してください。

- `STDDEV_SAMP(expr) [over_clause]`

`expr` の標本標準偏差 (`VAR_SAMP()` の平方根) を返します。

一致する行がない場合、`STDDEV_SAMP()` は `NULL` を返します。

`over_clause` が存在する場合、この関数はウィンドウ関数として実行されます。`over_clause` については、[セクション12.21.2「Window 関数の概念と構文」](#) を参照してください。

- `SUM([DISTINCT] expr) [over_clause]`

`expr` の集計を返します。戻り値のセットに行が含まれていない場合、`SUM()` は `NULL` を返します。`DISTINCT` キーワードを使用すると、個別の `expr` 値のみを集計できます。

一致する行がない場合、`SUM()` は `NULL` を返します。

`over_clause` が存在する場合、この関数はウィンドウ関数として実行されます。`over_clause` は、[セクション12.21.2「Window 関数の概念と構文」](#) で説明されているように、`DISTINCT` では使用できません。

- `VAR_POP(expr) [over_clause]`

`expr` の母標準分散を返します。行は標本ではなく、母集団全体とみなされるため、行の数が分母とみなされます。また、`VARIANCE()` を使用することもできます。これは同等ですが、標準 SQL ではありません。

一致する行がない場合、`VAR_POP()` は `NULL` を返します。

`over_clause` が存在する場合、この関数はウィンドウ関数として実行されます。`over_clause` については、[セクション12.21.2「Window 関数の概念と構文」](#) を参照してください。

- `VAR_SAMP(expr) [over_clause]`

`expr` の標本分散を返します。つまり、分母は行の数から 1 を引いたものです。

一致する行がない場合、`VAR_SAMP()` は `NULL` を返します。

`over_clause` が存在する場合、この関数はウィンドウ関数として実行されます。`over_clause` については、[セクション12.21.2「Window 関数の概念と構文」](#) を参照してください。

- `VARIANCE(expr) [over_clause]`

`expr` の母標準分散を返します。`VARIANCE()` は、MySQL 拡張機能として提供される標準 SQL 関数 `VAR_POP()` のシノニムです。

一致する行がない場合、`VARIANCE()` は `NULL` を返します。

`over_clause` が存在する場合、この関数はウィンドウ関数として実行されます。`over_clause` については、[セクション12.21.2「Window 関数の概念と構文」](#) を参照してください。

12.20.2 GROUP BY 修飾子

GROUP BY 句を使用すると、サマリー出力に上位レベル (つまり、上位集計) のサマリー操作を表す追加の行を含めることができる **WITH ROLLUP** 修飾子が許可されます。したがって、**ROLLUP** で単一のクエリーを使用すれば、複数レベルの分析で質問に回答できます。たとえば、**ROLLUP** を使用して OLAP (オンライン分析処理) 操作をサポートできます。

sales テーブルに、売上収益性を記録するための **year**, **country**, **product** カラムと **profit** カラムがあるとします:

```
CREATE TABLE sales
(
  year INT,
  country VARCHAR(20),
  product VARCHAR(32),
  profit INT
);
```

年ごとにテーブルの内容を要約するには、次のような単純な **GROUP BY** を使用します:

```
mysql> SELECT year, SUM(profit) AS profit
        FROM sales
        GROUP BY year;
+-----+-----+
| year | profit |
+-----+-----+
| 2000 | 4525 |
| 2001 | 3010 |
+-----+-----+
```

出力には、各年の利益の合計 (集計) が表示されます。すべての年の合計利益も決定するには、個々の値を自分で加算するか、追加のクエリーを実行する必要があります。または、単一のクエリーで両方のレベルの分析を提供する **ROLLUP** も使用できます。 **GROUP BY** 句に **WITH ROLLUP** 修飾子を追加すると、クエリーによって、すべての年の値の総計を示す別の (スーパー集計) 行が生成されます:

```
mysql> SELECT year, SUM(profit) AS profit
        FROM sales
        GROUP BY year WITH ROLLUP;
+-----+-----+
| year | profit |
+-----+-----+
| 2000 | 4525 |
| 2001 | 3010 |
| NULL | 7535 |
+-----+-----+
```

year カラムの **NULL** 値は、超集計行の総計を識別します。

複数の **GROUP BY** カラムがある場合は、**ROLLUP** の効果がより複雑になります。この場合、最後のグループ化カラム以外の値が変更されるたびに、クエリーは追加の超集計サマリー行を生成します。

たとえば、**ROLLUP** がない場合、**year**、**country** および **product** に基づく **sales** テーブルのサマリーは次のようになります。出力には **year/country/product** レベルの分析でのみサマリー値が表示されます:

```
mysql> SELECT year, country, product, SUM(profit) AS profit
        FROM sales
        GROUP BY year, country, product;
+-----+-----+-----+-----+
| year | country | product | profit |
+-----+-----+-----+-----+
| 2000 | Finland | Computer | 1500 |
| 2000 | Finland | Phone   | 100 |
| 2000 | India   | Calculator | 150 |
| 2000 | India   | Computer | 1200 |
| 2000 | USA     | Calculator | 75 |
| 2000 | USA     | Computer | 1500 |
| 2001 | Finland | Phone   | 10 |
| 2001 | USA     | Calculator | 50 |
| 2001 | USA     | Computer | 2700 |
| 2001 | USA     | TV      | 250 |
+-----+-----+-----+-----+
```

ROLLUP が追加されると、クエリーによっていくつかの追加の行が生成されます:

```
mysql> SELECT year, country, product, SUM(profit) AS profit
FROM sales
GROUP BY year, country, product WITH ROLLUP;
+-----+-----+-----+
| year | country | product | profit |
+-----+-----+-----+
| 2000 | Finland | Computer | 1500 |
| 2000 | Finland | Phone | 100 |
| 2000 | Finland | NULL | 1600 |
| 2000 | India | Calculator | 150 |
| 2000 | India | Computer | 1200 |
| 2000 | India | NULL | 1350 |
| 2000 | USA | Calculator | 75 |
| 2000 | USA | Computer | 1500 |
| 2000 | USA | NULL | 1575 |
| 2000 | NULL | NULL | 4525 |
| 2001 | Finland | Phone | 10 |
| 2001 | Finland | NULL | 10 |
| 2001 | USA | Calculator | 50 |
| 2001 | USA | Computer | 2700 |
| 2001 | USA | TV | 250 |
| 2001 | USA | NULL | 3000 |
| 2001 | NULL | NULL | 3010 |
| NULL | NULL | NULL | 7535 |
+-----+-----+-----+
```

出力には、1つだけでなく、4つの分析レベルのサマリー情報が含まれるようになりました:

- 特定の年および国の各製品行の後に、すべての製品の合計を示す追加の上位集計サマリー行が表示されます。これらの行には、NULL に設定された `product` カラムが含まれています。
- 特定の年の各行の後に、追加の上位集計サマリー行が表示され、すべての国および製品の合計が示されます。これらの行には、NULL に設定された `country` および `products` カラムが含まれています。
- 最後に、他のすべての行の後に、追加の上位集計サマリー行が表示され、すべての年、国および製品の総計が示されます。この行には、NULL に設定された `year`、`country`、および `products` カラムが含まれています。

各超集約行の NULL 指示子は、行がクライアントに送信されるときに生成されます。サーバーでは、変更された値を持つ左端のカラムに続いて、GROUP BY 句で名前が指定されたカラムが調査されます。これらの名前のいずれかと一致する名前を持つ結果セット内のカラムの場合、その値は NULL に設定されます。(カラムの位置でカラムをグループ化するように指定した場合、サーバーは位置で NULL に設定するカラムを識別します。)

上位集計行の NULL 値は、クエリー処理のこのような遅延ステージで結果セットに配置されるため、SELECT 構文のリストまたは HAVING 句でのみ NULL 値としてテストできます。結合条件または WHERE 句で NULL 値としてテストして、選択する行を決定することはできません。たとえば、WHERE product IS NULL をクエリーに追加して、スーパー集計行以外のすべての行を出力から除外することはできません。

NULL 値は、クライアント側では NULL として表示され、任意の MySQL クライアントプログラミングインタフェースを使用してテストできます。ただし、この時点では、NULL が通常のグループ化された値を表すか、スーパー集計値を表すかは区別できません。区別をテストするには、後述する GROUPING() 関数を使用します。

以前は、MySQL では、WITH ROLLUP オプションを持つクエリーで DISTINCT または ORDER BY を使用できませんでした。この制限は、MySQL 8.0.12 以降ではなくなりました。(Bug #87450、Bug #86311、Bug #26640100、Bug #26073513)

GROUP BY ... WITH ROLLUP クエリーでは、結果の NULL 値が超集計値を表すかどうかをテストするために、GROUPING() 関数を選択リスト、HAVING 句および (MySQL 8.0.12 の時点) ORDER BY 句で使用できます。たとえば、GROUPING(year) は、year カラムの NULL がスーパー集計行にある場合は 1 を返し、それ以外の場合は 0 を返します。同様に、country カラムと product カラムのスーパー集計 NULL 値に対して、GROUPING(country) と GROUPING(product) はそれぞれ 1 を返します:

```
mysql> SELECT
year, country, product, SUM(profit) AS profit,
GROUPING(year) AS grp_year,
GROUPING(country) AS grp_country,
GROUPING(product) AS grp_product
FROM sales
GROUP BY year, country, product WITH ROLLUP;
+-----+-----+-----+-----+-----+-----+
| year | country | product | profit | grp_year | grp_country | grp_product |
+-----+-----+-----+-----+-----+-----+-----+
| 2000 | Finland | Computer | 1500 | 1 | 0 | 0 |
| 2000 | Finland | Phone | 100 | 1 | 0 | 0 |
| 2000 | Finland | NULL | 1600 | 1 | 0 | 1 |
| 2000 | India | Calculator | 150 | 1 | 1 | 0 |
| 2000 | India | Computer | 1200 | 1 | 1 | 0 |
| 2000 | India | NULL | 1350 | 1 | 1 | 1 |
| 2000 | USA | Calculator | 75 | 1 | 1 | 0 |
| 2000 | USA | Computer | 1500 | 1 | 1 | 0 |
| 2000 | USA | NULL | 1575 | 1 | 1 | 1 |
| 2000 | NULL | NULL | 4525 | 0 | 0 | 0 |
| 2001 | Finland | Phone | 10 | 1 | 0 | 0 |
| 2001 | Finland | NULL | 10 | 1 | 0 | 1 |
| 2001 | USA | Calculator | 50 | 1 | 1 | 0 |
| 2001 | USA | Computer | 2700 | 1 | 1 | 0 |
| 2001 | USA | TV | 250 | 1 | 1 | 0 |
| 2001 | USA | NULL | 3000 | 1 | 1 | 1 |
| 2001 | NULL | NULL | 3010 | 0 | 0 | 0 |
| NULL | NULL | NULL | 7535 | 0 | 0 | 0 |
+-----+-----+-----+-----+-----+-----+-----+
```

year	country	product	profit	grp_year	grp_country	grp_product
2000	Finland	Computer	1500	0	0	0
2000	Finland	Phone	100	0	0	0
2000	Finland	NULL	1600	0	0	1
2000	India	Calculator	150	0	0	0
2000	India	Computer	1200	0	0	0
2000	India	NULL	1350	0	0	1
2000	USA	Calculator	75	0	0	0
2000	USA	Computer	1500	0	0	0
2000	USA	NULL	1575	0	0	1
2000	NULL	NULL	4525	0	1	1
2001	Finland	Phone	10	0	0	0
2001	Finland	NULL	10	0	0	1
2001	USA	Calculator	50	0	0	0
2001	USA	Computer	2700	0	0	0
2001	USA	TV	250	0	0	0
2001	USA	NULL	3000	0	0	1
2001	NULL	NULL	3010	0	1	1
NULL	NULL	NULL	7535	1	1	1

`GROUPING()` の結果を直接表示するかわりに、`GROUPING()` を使用して、スーパー集計 `NULL` 値のラベルを置換できます:

```
mysql> SELECT
  IF(GROUPING(year), 'All years', year) AS year,
  IF(GROUPING(country), 'All countries', country) AS country,
  IF(GROUPING(product), 'All products', product) AS product,
  SUM(profit) AS profit
FROM sales
GROUP BY year, country, product WITH ROLLUP;
```

year	country	product	profit
2000	Finland	Computer	1500
2000	Finland	Phone	100
2000	Finland	All products	1600
2000	India	Calculator	150
2000	India	Computer	1200
2000	India	All products	1350
2000	USA	Calculator	75
2000	USA	Computer	1500
2000	USA	All products	1575
2000	All countries	All products	4525
2001	Finland	Phone	10
2001	Finland	All products	10
2001	USA	Calculator	50
2001	USA	Computer	2700
2001	USA	TV	250
2001	USA	All products	3000
2001	All countries	All products	3010
All years	All countries	All products	7535

複数の式引数を使用すると、`GROUPING()` は、各式の結果を結合するビットマスクを表す結果を、右端の式の結果に対応する最下位ビットと返します。たとえば、`GROUPING(year, country, product)` は次のように評価されます:

```
result for GROUPING(product)
+ result for GROUPING(country) << 1
+ result for GROUPING(year) << 2
```

このような `GROUPING()` の結果は、いずれかの式がスーパー集計 `NULL` を表す場合はゼロ以外であるため、スーパー集計行のみを返し、次のように通常のグループ化された行をフィルタで除外できます:

```
mysql> SELECT year, country, product, SUM(profit) AS profit
FROM sales
GROUP BY year, country, product WITH ROLLUP
HAVING GROUPING(year, country, product) <> 0;
```

year	country	product	profit
2000	Finland	NULL	1600

```

| 2000 | India | NULL | 1350 |
| 2000 | USA   | NULL | 1575 |
| 2000 | NULL  | NULL | 4525 |
| 2001 | Finland | NULL | 10 |
| 2001 | USA   | NULL | 3000 |
| 2001 | NULL  | NULL | 3010 |
| NULL | NULL  | NULL | 7535 |
+-----+-----+-----+

```

`sales` テーブルには `NULL` 値が含まれていないため、`ROLLUP` 結果のすべての `NULL` 値は超集計値をテーブルします。データセットに `NULL` 値が含まれている場合、`ROLLUP` サマリーには、スーパー集計だけでなく、通常のグループ化された行にも `NULL` 値が含まれることがあります。`GROUPING()` では、これらを区別できます。テーブル `t1` に、数量値のセットに対する 2 つのグループ化係数を持つ単純なデータセットが含まれているとします。ここで、`NULL` は「other」や「不明」などを示します:

```

mysql> SELECT * FROM t1;
+-----+-----+-----+
| name | size | quantity |
+-----+-----+-----+
| ball | small | 10 |
| ball | large | 20 |
| ball | NULL  | 5 |
| hoop | small | 15 |
| hoop | large | 5 |
| hoop | NULL  | 3 |
+-----+-----+-----+

```

単純な `ROLLUP` 操作では次の結果が生成されるため、通常のグループ化された行の `NULL` 値とスーパー集計行の `NULL` 値を区別するのはそれほど簡単ではありません:

```

mysql> SELECT name, size, SUM(quantity) AS quantity
FROM t1
GROUP BY name, size WITH ROLLUP;
+-----+-----+-----+
| name | size | quantity |
+-----+-----+-----+
| ball | NULL | 5 |
| ball | large | 20 |
| ball | small | 10 |
| ball | NULL | 35 |
| hoop | NULL | 3 |
| hoop | large | 5 |
| hoop | small | 15 |
| hoop | NULL | 23 |
| NULL | NULL | 58 |
+-----+-----+-----+

```

`GROUPING()` を使用してスーパー集計 `NULL` 値のラベルを置換すると、結果の解釈が容易になります:

```

mysql> SELECT
IF(GROUPING(name) = 1, 'All items', name) AS name,
IF(GROUPING(size) = 1, 'All sizes', size) AS size,
SUM(quantity) AS quantity
FROM t1
GROUP BY name, size WITH ROLLUP;
+-----+-----+-----+
| name | size | quantity |
+-----+-----+-----+
| ball | NULL | 5 |
| ball | large | 20 |
| ball | small | 10 |
| ball | All sizes | 35 |
| hoop | NULL | 3 |
| hoop | large | 5 |
| hoop | small | 15 |
| hoop | All sizes | 23 |
| All items | All sizes | 58 |
+-----+-----+-----+

```

ROLLUP 使用時のその他の考慮事項

次の説明では、`ROLLUP` の MySQL 実装に固有の動作の一部を示します。

MySQL 8.0.12 より前では、**ROLLUP** を使用する場合、**ORDER BY** 句を使用して結果をソートすることもできません。つまり、**ROLLUP** と **ORDER BY** は MySQL で相互に排他的でした。ただし、ソート順序を一部制御することはできます。**ORDER BY** で **ROLLUP** を使用せず、グループ化された結果の特定のソート順序を達成するという制限を回避するには、グループ化された結果セットを導出テーブルとして生成し、それに **ORDER BY** を適用します。例:

```
mysql> SELECT * FROM
  (SELECT year, SUM(profit) AS profit
   FROM sales GROUP BY year WITH ROLLUP) AS dt
 ORDER BY year DESC;
+-----+-----+
| year | profit |
+-----+-----+
| 2001 | 3010 |
| 2000 | 4525 |
| NULL | 7535 |
+-----+-----+
```

MySQL 8.0.12 では、**ORDER BY** と **ROLLUP** を一緒に使用できるため、**ORDER BY** と **GROUPING()** を使用して、グループ化された結果の特定のソート順序を実現できます。例:

```
mysql> SELECT year, SUM(profit) AS profit
 FROM sales
 GROUP BY year WITH ROLLUP
 ORDER BY GROUPING(year) DESC;
+-----+-----+
| year | profit |
+-----+-----+
| NULL | 7535 |
| 2000 | 4525 |
| 2001 | 3010 |
+-----+-----+
```

どちらの場合も、集計上のサマリー行は計算元の行でソートされ、その配置はソート順 (昇順ソートの場合は末尾、降順ソートの場合は先頭) に依存します。

LIMIT を使用すると、クライアントに返される行の数を制限できます。**LIMIT** は **ROLLUP** のあとに適用されるため、**ROLLUP** で追加された追加の行に対して制限が適用されます。例:

```
mysql> SELECT year, country, product, SUM(profit) AS profit
 FROM sales
 GROUP BY year, country, product WITH ROLLUP
 LIMIT 5;
+-----+-----+-----+-----+
| year | country | product | profit |
+-----+-----+-----+-----+
| 2000 | Finland | Computer | 1500 |
| 2000 | Finland | Phone | 100 |
| 2000 | Finland | NULL | 1600 |
| 2000 | India | Calculator | 150 |
| 2000 | India | Computer | 1200 |
+-----+-----+-----+-----+
```

LIMIT を **ROLLUP** とともに使用すると、スーパー集計行を理解するためのコンテキストが少なくなるため、解釈が困難な結果が生成される場合があります。

MySQL 拡張機能では、**GROUP BY** リストに表示されないカラムを選択リストに指定できます。(非集計カラムおよび **GROUP BY** の詳細は、[セクション12.20.3「MySQL での GROUP BY の処理」](#) を参照してください。) この場合、サーバーはサマリー行内のこのような非集約カラムから任意の値を自由に選択できます。これには、**WITH ROLLUP** で追加された追加の行も含まれます。たとえば、次のクエリーでは、**country** は **GROUP BY** リストに表示されない非集計カラムであり、このカラムに選択された値は非決定的です:

```
mysql> SELECT year, country, SUM(profit) AS profit
 FROM sales
 GROUP BY year WITH ROLLUP;
+-----+-----+-----+
| year | country | profit |
+-----+-----+-----+
| 2000 | India | 4525 |
| 2001 | USA | 3010 |
| NULL | USA | 7535 |
+-----+-----+-----+
```

この動作は、`ONLY_FULL_GROUP_BY` SQL モードが有効になっていない場合に許可されます。このモードが有効になっている場合は、`country` が `GROUP BY` 句に一覧表示されないため、サーバーはそのクエリーを不正として拒否します。`ONLY_FULL_GROUP_BY` が有効になっている場合でも、非決定的値カラムに対して `ANY_VALUE()` 関数を使用してクエリーを実行できます:

```
mysql> SELECT year, ANY_VALUE(country) AS country, SUM(profit) AS profit
FROM sales
GROUP BY year WITH ROLLUP;
```

year	country	profit
2000	India	4525
2001	USA	3010
NULL	USA	7535

12.20.3 MySQL での GROUP BY の処理

SQL-92 以前では、選択リスト、`HAVING` 条件または `ORDER BY` リストが `GROUP BY` 句で指定されていない非集計カラムを参照するクエリーは許可されません。たとえば、このクエリーは、選択リストの非集計 `name` カラムが `GROUP BY` に表示されないため、標準 SQL-92 では無効です:

```
SELECT o.custid, c.name, MAX(o.payment)
FROM orders AS o, customers AS c
WHERE o.custid = c.custid
GROUP BY o.custid;
```

クエリーを SQL-92 で有効にするには、選択リストから `name` カラムを省略するか、`GROUP BY` 句で名前を指定する必要があります。

SQL:1999 以降では、`GROUP BY` カラムに機能的に依存している場合、オプション機能 T301 ごとにこのような非集計が許可されます: このような関係が `name` と `custid` の間に存在する場合、クエリーは有効です。たとえば、これは `custid` が `customers` の主キーであった場合です。

MySQL は、関数従属性の検出を実装しています。`ONLY_FULL_GROUP_BY` SQL モードが有効な場合 (デフォルト)、MySQL は、選択リスト、`HAVING` 条件または `ORDER BY` リストが `GROUP BY` 句で名前が付けられておらず、機能的に依存していない非集計カラムを参照するクエリーを拒否します。

次の例に示すように、SQL `ONLY_FULL_GROUP_BY` モードが有効になっている場合、MySQL では、`GROUP BY` 句で指定されていない非集計カラムも許可されます:

```
mysql> CREATE TABLE mytable (
-> id INT UNSIGNED NOT NULL PRIMARY KEY,
-> a VARCHAR(10),
-> b INT
-> );

mysql> INSERT INTO mytable
-> VALUES (1, 'abc', 1000),
-> (2, 'abc', 2000),
-> (3, 'def', 4000);

mysql> SET SESSION sql_mode = sys.list_add(@@session.sql_mode, 'ONLY_FULL_GROUP_BY');
```

```
mysql> SELECT a, SUM(b) FROM mytable WHERE a = 'abc';
```

a	SUM(b)
abc	3000

`ONLY_FULL_GROUP_BY` を使用する場合は、`SELECT` リストに複数の非集計カラムを含めることもできます。この場合、次に示すように、このようなカラムはすべて `WHERE` 句の単一の値に制限する必要があり、このような制限条件はすべて論理 `AND` によって結合する必要があります:

```
mysql> DROP TABLE IF EXISTS mytable;
```



```
mysql> CREATE TABLE mytable (
-> id INT UNSIGNED NOT NULL PRIMARY KEY,
-> a VARCHAR(10),
-> b VARCHAR(10),
-> c INT
-> );

mysql> INSERT INTO mytable
-> VALUES (1, 'abc', 'qrs', 1000),
-> (2, 'abc', 'tuv', 2000),
-> (3, 'def', 'qrs', 4000),
-> (4, 'def', 'tuv', 8000),
-> (5, 'abc', 'qrs', 16000),
-> (6, 'def', 'tuv', 32000);

mysql> SELECT @@session.sql_mode;
+-----+
|@@session.sql_mode |
+-----+
|ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION|
+-----+

mysql> SELECT a, b, SUM(c) FROM mytable
-> WHERE a = 'abc' AND b = 'qrs';
+----+-----+
|a |b |SUM(c)|
+----+-----+
|abc|qrs|17000|
+----+-----+
```

`ONLY_FULL_GROUP_BY` が無効になっている場合、`GROUP BY` の標準 SQL 使用に対する MySQL 拡張機能を使用すると、カラムが `GROUP BY` カラムに機能的に依存していなくても、選択リスト、`HAVING` 条件または `ORDER BY` リストで集計されていないカラムを参照できます。これにより、MySQL は前述のクエリーを受け入れます。この場合、サーバーは各グループから任意の値を自由に選択できるため、それらが同じでないかぎり、選択される値は決定的ではなく、必要な値ではない可能性があります。さらに、`ORDER BY` 句を追加しても、各グループからの値の選択が影響を受ける可能性はありません。結果セットのソートは値が選択された後に行われ、`ORDER BY` はサーバーが選択する各グループ内の値には影響しません。`ONLY_FULL_GROUP_BY` の無効化は、主に、データの一部のプロパティのために、`GROUP BY` で指定されていない集計されていない各カラムのすべての値がグループごとに同じであることがわかっている場合に役立ちます。

`ANY_VALUE()` を使用して非集計カラムを参照することで、`ONLY_FULL_GROUP_BY` を無効にせずに同じ効果を得ることができます。

次の説明は、関数従属性、関数従属性が存在しない場合に MySQL が生成するエラーメッセージ、および関数従属性が存在しない場合に MySQL がクエリーを受け入れる方法を示しています。

選択リストの非集計 `address` カラムが `GROUP BY` 句で指定されていないため、`ONLY_FULL_GROUP_BY` が有効な場合、このクエリーは無効である可能性があります:

```
SELECT name, address, MAX(age) FROM t GROUP BY name;
```

このクエリーは、`name` が `t` の主キーであるか、一意の `NOT NULL` カラムである場合に有効です。このような場合、MySQL は、選択されたカラムが機能的にグループ化カラムに依存していることを認識します。たとえば、`name` が主キーの場合、各グループには主キーの値が 1 つのみであるため、その値によって `address` の値が決まります。そのため、グループ内の `address` 値の選択にランダム性はなく、クエリーを拒否する必要はありません。

`name` が `t` の主キーまたは一意の `NOT NULL` カラムでない場合、クエリーは無効です。この場合、関数従属性を推測できず、エラーが発生します:

```
mysql> SELECT name, address, MAX(age) FROM t GROUP BY name;
ERROR 1055 (42000): Expression #2 of SELECT list is not in GROUP
BY clause and contains nonaggregated column 'mydb.t.address' which
is not functionally dependent on columns in GROUP BY clause; this
is incompatible with sql_mode=only_full_group_by
```

それがわかっている場合、実際には特定のデータセットの各 `name` 値によって `address` 値が一意に決定され、`address` は事実上 `name` に依存します。クエリーを受け入れるように MySQL に指示するには、`ANY_VALUE()` 関数を使用します:

```
SELECT name, ANY_VALUE(address), MAX(age) FROM t GROUP BY name;
```

または、`ONLY_FULL_GROUP_BY` を無効にします。

ただし、前述の例は非常に単純です。特に、すべてのグループに 1 つの行のみが含まれるため、単一の主キーカラムでグループ化することはほとんどありません。より複雑なクエリーでの関数従属性を示す追加の例は、[セクション 12.20.4「機能依存性の検出」](#) を参照してください。

クエリーに集計関数があり、`GROUP BY` 句がない場合、`ONLY_FULL_GROUP_BY` が有効になっている選択リスト、`HAVING` 条件または `ORDER BY` リストに非集計カラムを含めることはできません:

```
mysql> SELECT name, MAX(age) FROM t;
ERROR 1140 (42000): In aggregated query without GROUP BY, expression
#1 of SELECT list contains nonaggregated column 'mydb.t.name'; this
is incompatible with sql_mode=only_full_group_by
```

`GROUP BY` がない場合、単一のグループが存在し、どの `name` 値をグループに選択するかは非決定的です。ここでも、MySQL が選択する `name` 値が重要でない場合は、`ANY_VALUE()` を使用できます:

```
SELECT ANY_VALUE(name), MAX(age) FROM t;
```

`ONLY_FULL_GROUP_BY` は、`DISTINCT` および `ORDER BY` を使用するクエリーの処理にも影響します。次の行を含む `c1`、`c2` および `c3` の 3 つのカラムを含むテーブル `t` の場合を考えてみます:

```
c1 c2 c3
1 2 A
3 4 B
1 2 C
```

次のクエリーを実行し、結果を `c3` で順序付けするとします:

```
SELECT DISTINCT c1, c2 FROM t ORDER BY c3;
```

結果を並べ替えるには、最初に重複を排除する必要があります。ただし、そのためには、最初の行または 3 番目の行を保持する必要がありますか。この任意の選択は `c3` の保持された値に影響し、これは順序付けに影響を与え、任意にすることもできます。この問題を回避するために、いずれかの `ORDER BY` 式が次のいずれかの条件を満たさない場合、`DISTINCT` および `ORDER BY` を含むクエリーは無効として拒否されます:

- 式が選択リストの式と等しい
- 式によって参照され、クエリーで選択されたテーブルに属するすべてのカラムは、選択リストの要素です

標準 SQL に対する別の MySQL 拡張機能では、選択リスト内のエイリアス式への `HAVING` 句内の参照が許可されます。たとえば、次のクエリーは、`orders` テーブルで 1 回だけ発生する `name` 値を返します。

```
SELECT name, COUNT(name) FROM orders
GROUP BY name
HAVING COUNT(name) = 1;
```

MySQL 拡張機能では、集計カラムの `HAVING` 句でエイリアスを使用できます:

```
SELECT name, COUNT(name) AS c FROM orders
GROUP BY name
HAVING c = 1;
```

標準 SQL では `GROUP BY` 句のカラム式のみが許可されるため、`FLOOR(value/100)` は非カラム式であるため、このようなステートメントは無効です:

```
SELECT id, FLOOR(value/100)
FROM tbl_name
GROUP BY id, FLOOR(value/100);
```

MySQL は、標準 SQL を拡張して `GROUP BY` 句でカラム以外の式を許可し、前述のステートメントが有効であるとみなします。

標準 SQL では、`GROUP BY` 句でエイリアスを使用することもできません。MySQL は標準 SQL を拡張してエイリアスを許可するため、クエリーを記述する別の方法は次のとおりです:

```
SELECT id, FLOOR(value/100) AS val
FROM tbl_name
GROUP BY id, val;
```

エイリアス `val` は、`GROUP BY` 句のカラム式とみなされます。

`GROUP BY` 句にカラム以外の式が存在する場合、MySQL はその式と選択リスト内の式の等価性を認識します。つまり、`ONLY_FULL_GROUP_BY` SQL モードが有効になっている場合、選択リストに同じ `FLOOR()` 式が出現するため、`GROUP BY id, FLOOR(value/100)` を含むクエリーは有効です。ただし、MySQL は `GROUP BY` の非カラム式への関数従属性を認識しないため、3 番目に選択された式が `id` カラムと `GROUP BY` 句の `FLOOR()` 式の単純な式であっても、`ONLY_FULL_GROUP_BY` が有効な場合は次のクエリーは無効です:

```
SELECT id, FLOOR(value/100), id+FLOOR(value/100)
FROM tbl_name
GROUP BY id, FLOOR(value/100);
```

回避策は、導出テーブルを使用することです:

```
SELECT id, F, id+F
FROM
  (SELECT id, FLOOR(value/100) AS F
   FROM tbl_name
   GROUP BY id, FLOOR(value/100)) AS dt;
```

12.20.4 機能依存性の検出

次の説明では、MySQL が関数従属性を検出する方法の例をいくつか示します。この例では、次の表記法を使用します:

```
{X} -> {Y}
```

これを「`X` は、`Y` を一意に決定」として理解します。つまり、`Y` は機能的に `X` に依存しています。

この例では、<https://dev.mysql.com/doc/index-other.html> からダウンロードできる `world` データベースを使用します。データベースのインストール方法の詳細は、同じページを参照してください。

- キーから導出された関数従属性
- 複数カラムキーおよび等価から導出された関数従属性
- 関数従属性特殊ケース
- 関数従属性とビュー
- 関数従属性の組合せ

キーから導出された関数従属性

次のクエリーは、国ごとに話された言語の数を選択します:

```
SELECT co.Name, COUNT(*)
FROM countrylanguage cl, country co
WHERE cl.CountryCode = co.Code
GROUP BY co.Code;
```

`co.Code` は `co` の主キーであるため、次の表記法を使用して表されるように、`co` のすべてのカラムは機能的に依存します:

```
{co.Code} -> {co.*}
```

したがって、`co.name` は機能的に `GROUP BY` カラムに依存し、クエリーは有効です。

主キーのかわりに `NOT NULL` カラムに対する `UNIQUE` インデックスを使用でき、同じ機能依存性が適用されます。(複数の `NULL` 値が許可され、その場合は一意性が失われるため、`NULL` 値を許可する `UNIQUE` インデックスには当てはまりません。)

複数カラムキーおよび等価から導出された関数従属性

このクエリーでは、国ごとに、すべての話し言葉のリストとそれらを話すユーザーの数を選択します:

```
SELECT co.Name, cl.Language,  
cl.Percentage * co.Population / 100.0 AS SpokenBy  
FROM countrylanguage cl, country co  
WHERE cl.CountryCode = co.Code  
GROUP BY cl.CountryCode, cl.Language;
```

ペア (`cl.CountryCode`, `cl.Language`) は `cl` の 2 カラム複合主キーであるため、カラムのペアによって `cl` のすべてのカラムが一意に決定されます:

```
{cl.CountryCode, cl.Language} -> {cl.*}
```

さらに、`WHERE` 句の等価性のため、次のようになります:

```
{cl.CountryCode} -> {co.Code}
```

また、`co.Code` は `co` の主キーであるため、次のようになります:

```
{co.Code} -> {co.*}
```

「一意に決定」関係は推移的であるため、次のようになります:

```
{cl.CountryCode, cl.Language} -> {cl.*,co.*}
```

その結果、クエリーは有効になります。

前の例と同様に、`NOT NULL` カラムに対する `UNIQUE` キーを主キーのかわりに使用できます。

`WHERE` のかわりに `INNER JOIN` 条件を使用できます。同じ機能依存性が適用されます:

```
SELECT co.Name, cl.Language,  
cl.Percentage * co.Population/100.0 AS SpokenBy  
FROM countrylanguage cl INNER JOIN country co  
ON cl.CountryCode = co.Code  
GROUP BY cl.CountryCode, cl.Language;
```

関数従属性特殊ケース

`WHERE` 条件または `INNER JOIN` 条件での等価テストは対称ですが、テーブルは異なる役割を果たすため、外部結合条件での等価テストは対称ではありません。

参照整合性が誤って破損し、対応する行がない `countrylanguage` の行が `country` に存在するとします。前述の例と同じクエリーを考えてみますが、`LEFT JOIN` の場合は次のようになります:

```
SELECT co.Name, cl.Language,  
cl.Percentage * co.Population/100.0 AS SpokenBy  
FROM countrylanguage cl LEFT JOIN country co  
ON cl.CountryCode = co.Code  
GROUP BY cl.CountryCode, cl.Language;
```

`cl.CountryCode` の特定の値について、結合結果の `co.Code` の値は、一致する行 (`cl.CountryCode` によって決定) で検出されるか、一致がない場合は `NULL` で補完されます (`cl.CountryCode` によっても決定されます)。いずれの場合も、この関係が適用されます:

```
{cl.CountryCode} -> {co.Code}
```

`cl.CountryCode` 自体は、主キーである `{cl.CountryCode, cl.Language}` に機能的に依存します。

結合結果で、`co.Code` が `NULL` で補完されている場合、`co.Name` も同様です。`co.Code` が `NULL` で補完されていない場合、`co.Code` は主キーであるため、`co.Name` が決定されます。したがって、すべての場合で次のようになります:

```
{co.Code} -> {co.Name}
```

次のものが生成されます:

```
{cl.CountryCode, cl.Language} -> {cl.*,co.*}
```

その結果、クエリーは有効になります。

ただし、次のクエリーのようにテーブルがスワップされるとします:

```
SELECT co.Name, cl.Language,
cl.Percentage * co.Population/100.0 AS SpokenBy
FROM country co LEFT JOIN countrylanguage cl
ON cl.CountryCode = co.Code
GROUP BY cl.CountryCode, cl.Language;
```

現在、この関係は適用されません:

```
{cl.CountryCode, cl.Language} -> {cl.*,co.*}
```

実際には、`cl` に対して作成された `NULL` で補完されたすべての行は単一のグループに配置され (両方の `GROUP BY` カラムが `NULL` と等しい)、このグループ内で `co.Name` の値が異なる可能性があります。クエリーが無効であり、MySQL によって拒否されます。

したがって、外部結合の関数従属性は、決定要因カラムが `LEFT JOIN` の左側に属するか右側に属するかにリンクされます。ネストされた外部結合がある場合、または結合条件が完全に等価比較で構成されていない場合、関数従属性の決定はより複雑になります。

関数従属性とビュー

国のビューでは、コード、名前 (大文字)、および国が持つ異なる公用語の数が生成されるとします:

```
CREATE VIEW country2 AS
SELECT co.Code, UPPER(co.Name) AS UpperName,
COUNT(cl.Language) AS OfficialLanguages
FROM country AS co JOIN countrylanguage AS cl
ON cl.CountryCode = co.Code
WHERE cl.isOfficial = 'T'
GROUP BY co.Code;
```

この定義は、次の理由で有効です:

```
{co.Code} -> {co.*}
```

ビュー結果では、最初に選択されたカラムは `co.Code` であり、これはグループカラムでもあるため、選択された他のすべての式が決定されます:

```
{country2.Code} -> {country2.*}
```

次に説明するように、MySQL はこれを理解し、この情報を使用します。

このクエリーでは、`city` テーブルとビューを結合することで、国、国の公用語の数および市区町村の数が表示されます:

```
SELECT co2.Code, co2.UpperName, co2.OfficialLanguages,
COUNT(*) AS Cities
FROM country2 AS co2 JOIN city ci
ON ci.CountryCode = co2.Code
GROUP BY co2.Code;
```

前述のとおり、このクエリーは次の理由で有効です:

```
{co2.Code} -> {co2.*}
```

MySQL では、ビューの結果で関数従属性を検出し、それを使用してビューを使用するクエリーを検証できます。次のように、`country2` が導出テーブル (または共通テーブル式) であった場合も同様です:

```
SELECT co2.Code, co2.UpperName, co2.OfficialLanguages,
COUNT(*) AS Cities
```

```

FROM
(
SELECT co.Code, UPPER(co.Name) AS UpperName,
COUNT(cl.Language) AS OfficialLanguages
FROM country AS co JOIN countrylanguage AS cl
ON cl.CountryCode=co.Code
WHERE cl.isOfficial='T'
GROUP BY co.Code
) AS co2
JOIN city ci ON ci.CountryCode = co2.Code
GROUP BY co2.Code;

```

関数従属性の組合せ

MySQL では、前述のすべてのタイプの関数従属性 (キーベース、等価ベース、ビューベース) を組み合わせて、より複雑なクエリーを検証できます。

12.21 ウィンドウ関数

MySQL では、クエリーの各行について、その行に関連する行を使用して計算を実行するウィンドウ関数がサポートされています。次の各セクションでは、[OVER](#) 句および [WINDOW](#) 句の説明を含む、ウィンドウ関数の使用方法について説明します。最初のセクションでは、非集計ウィンドウ関数について説明します。集計ウィンドウ関数の詳細は、[セクション12.20.1「集計関数の説明」](#) を参照してください。

最適化およびウィンドウ関数の詳細は、[セクション8.2.1.21「ウィンドウ機能最適化」](#) を参照してください。

12.21.1 Window 関数の説明

このセクションでは、クエリーの各行について、その行に関連する行を使用して計算を実行する非集計ウィンドウ関数について説明します。ほとんどの集計関数は、ウィンドウ関数としても使用できます。[セクション12.20.1「集計関数の説明」](#) を参照してください。

ウィンドウ関数の使用方法と例、および [OVER](#) 句、ウィンドウ、パーティション、フレーム、ピアなどの用語の定義については、[セクション12.21.2「Window 関数の概念と構文」](#) を参照してください。

表 12.26 「ウィンドウ関数」

名前	説明
CUME_DIST()	累積分布値
DENSE_RANK()	パーティション内の現在の行のランク (ギャップなし)
FIRST_VALUE()	ウィンドウフレームの最初の行からの引数の値
LAG()	パーティション内の現在行より遅れている行の引数の値
LAST_VALUE()	ウィンドウフレームの最後の行からの引数の値
LEAD()	パーティション内の現在の行の先頭行からの引数の値
NTH_VALUE()	ウィンドウフレームの N 番目の行からの引数の値
NTILE()	パーティション内の現在の行のバケット番号。
PERCENT_RANK()	パーセントランク値
RANK()	パーティション内の現在の行のランク (ギャップあり)
ROW_NUMBER()	パーティション内の現在の行数

次の関数の説明で、[over_clause](#) は、[セクション12.21.2「Window 関数の概念と構文」](#) で説明されている [OVER](#) 句を表します。一部のウィンドウ関数では、結果の計算時に [NULL](#) 値の処理方法を指定する [null_treatment](#) 句を使用できます。この句はオプションです。これは SQL 標準の一部ですが、MySQL 実装では [RESPECT NULLS](#) (デフォルト) のみが許可されます。これは、結果の計算時に [NULL](#) 値が考慮されることを意味します。[IGNORE NULLS](#) は解析されませんが、エラーが発生します。

- [CUME_DIST\(\)](#) [over_clause](#)

値のグループ内の値の累積分布、つまり現在の行の値以下のパーティション値の割合を戻します。これは、ウィンドウパーティションのウィンドウ順序で現在の行の前またはピアの行数をウィンドウパーティションの合計行数で割った数を表示します。戻り値の範囲は 0 から 1 です。

パーティション行を目的の順序にソートするには、**ORDER BY** でこの関数を使用する必要があります。**ORDER BY** を使用しない場合、すべての行はピアであり、 $N/N = 1$ の値を持ちます。ここで、**N** はパーティションサイズです。

over_clause については、[セクション12.21.2 「Window 関数の概念と構文」](#) を参照してください。

次のクエリーでは、**val** カラムの値セットについて、各行の **CUME_DIST()** 値、および同様の **PERCENT_RANK()** 関数によって返されたパーセントランク値が表示されます。参照のために、クエリーでは **ROW_NUMBER()** を使用して行番号も表示されます：

```
mysql> SELECT
  val,
  ROW_NUMBER() OVER w AS 'row_number',
  CUME_DIST() OVER w AS 'cume_dist',
  PERCENT_RANK() OVER w AS 'percent_rank'
FROM numbers
WINDOW w AS (ORDER BY val);
```

val	row_number	cume_dist	percent_rank
1	1	0.2222222222222222	0
1	2	0.2222222222222222	0
2	3	0.3333333333333333	0.25
3	4	0.6666666666666666	0.375
3	5	0.6666666666666666	0.375
3	6	0.6666666666666666	0.375
4	7	0.8888888888888888	0.75
4	8	0.8888888888888888	0.75
5	9	1	1

- **DENSE_RANK()** **over_clause**

パーティション内の現在の行のランクをギャップなしで返します。同僚は同僚とみなされ、同じランクを受け取ります。この関数は、ピアグループに連続するランクを割り当てます。その結果、サイズが 1 より大きいグループは、連続しないランク番号を生成しません。例については、**RANK()** 関数の説明を参照してください。

パーティション行を目的の順序にソートするには、**ORDER BY** でこの関数を使用する必要があります。**ORDER BY** が無い場合、すべての行がピアになります。

over_clause については、[セクション12.21.2 「Window 関数の概念と構文」](#) を参照してください。

- **FIRST_VALUE(expr)** [**null_treatment**] **over_clause**

ウィンドウフレームの最初の行から **expr** の値を返します。

over_clause については、[セクション12.21.2 「Window 関数の概念と構文」](#) を参照してください。**null_treatment** については、概要のセクションで説明しています。

次のクエリーは、**FIRST_VALUE()**、**LAST_VALUE()** および **NTH_VALUE()** の 2 つのインスタンスを示しています：

```
mysql> SELECT
  time, subject, val,
  FIRST_VALUE(val) OVER w AS 'first',
  LAST_VALUE(val) OVER w AS 'last',
  NTH_VALUE(val, 2) OVER w AS 'second',
  NTH_VALUE(val, 4) OVER w AS 'fourth'
FROM observations
WINDOW w AS (PARTITION BY subject ORDER BY time
  ROWS UNBOUNDED PRECEDING);
```

time	subject	val	first	last	second	fourth
07:00:00	st113	10	10	10	NULL	NULL

```
| 07:15:00 | st113 | 9 | 10 | 9 | 9 | NULL |
| 07:30:00 | st113 | 25 | 10 | 25 | 9 | NULL |
| 07:45:00 | st113 | 20 | 10 | 20 | 9 | 20 |
| 07:00:00 | xh458 | 0 | 0 | 0 | NULL | NULL |
| 07:15:00 | xh458 | 10 | 0 | 10 | 10 | NULL |
| 07:30:00 | xh458 | 5 | 0 | 5 | 10 | NULL |
| 07:45:00 | xh458 | 30 | 0 | 30 | 10 | 30 |
| 08:00:00 | xh458 | 25 | 0 | 25 | 10 | 30 |
+-----+-----+-----+-----+-----+-----+
```

各関数は、現在のフレーム内の行を使用します。この行は、表示されているウィンドウ定義に従って、最初のパーティション行から現在の行に拡張されます。NTH_VALUE() コールの場合、現在のフレームにはリクエストされた行が常に含まれるわけではありません。このような場合、戻り値は NULL です。

- LAG(expr [, N[, default]]) [null_treatment] over_clause

パーティション内の N 行で現在の行を遅延 (前) する行から expr の値を返します。そのような行がない場合、戻り値は default です。たとえば、N が 3 の場合、最初の 2 行の戻り値は default です。N または default がいない場合、デフォルトはそれぞれ 1 および NULL です。

N は、負でないリテラル整数である必要があります。N が 0 の場合、expr は現在の行に対して評価されます。

MySQL 8.0.22 以降、N を NULL にすることはできません。さらに、次のいずれかの形式で、1 から 2⁶³ までの範囲の整数である必要があります:

- 符号なし整数定数リテラル
- 位置パラメータマーカー (?)
- ユーザー定義変数
- ストアドルーチン内のローカル変数

over_clause については、[セクション12.21.2「Window 関数の概念と構文」](#)を参照してください。null_treatment については、概要のセクションで説明しています。

LAG() (および同様の LEAD() 関数) は、多くの場合、行間の差異を計算するために使用されます。次のクエリーは、時間順の監視のセットと、隣接する行からの LAG() および LEAD() の値、および現在の行と隣接する行の違いを示しています:

```
mysql> SELECT
  t, val,
  LAG(val) OVER w AS 'lag',
  LEAD(val) OVER w AS 'lead',
  val - LAG(val) OVER w AS 'lag diff',
  val - LEAD(val) OVER w AS 'lead diff'
FROM series
WINDOW w AS (ORDER BY t);
+-----+-----+-----+-----+-----+-----+
| t | val | lag | lead | lag diff | lead diff |
+-----+-----+-----+-----+-----+
| 12:00:00 | 100 | NULL | 125 | NULL | -25 |
| 13:00:00 | 125 | 100 | 132 | 25 | -7 |
| 14:00:00 | 132 | 125 | 145 | 7 | -13 |
| 15:00:00 | 145 | 132 | 140 | 13 | 5 |
| 16:00:00 | 140 | 145 | 150 | -5 | -10 |
| 17:00:00 | 150 | 140 | 200 | 10 | -50 |
| 18:00:00 | 200 | 150 | NULL | 50 | NULL |
+-----+-----+-----+-----+-----+-----+
```

この例では、LAG() コールと LEAD() コールは、それぞれデフォルトの N 値と default 値の 1 と NULL を使用します。

最初の行は、LAG() に前の行がない場合の処理を示しています: この関数は、default 値 (この場合は NULL) を返します。最後の行には、LEAD() の次の行がない場合と同じものが表示されます。

LAG() と LEAD() は、差異ではなく合計の計算にも役立ちます。Fibonacci シリーズの最初の数個を含む次のデータセットについて考えてみます:

```
mysql> SELECT n FROM fib ORDER BY n;
+----+
| n |
+----+
| 1 |
| 1 |
| 2 |
| 3 |
| 5 |
| 8 |
+----+
```

次のクエリは、現在の行に隣接する行の `LAG()` および `LEAD()` の値を表示します。また、これらの関数を使用して、前後の行の値を現在の行の値に追加します。効果は、Fibonacci シリーズの次の番号と、それ以降の次の番号を生成することです:

```
mysql> SELECT
  n,
  LAG(n, 1, 0) OVER w AS 'lag',
  LEAD(n, 1, 0) OVER w AS 'lead',
  n + LAG(n, 1, 0) OVER w AS 'next_n',
  n + LEAD(n, 1, 0) OVER w AS 'next_next_n'
FROM fib
WINDOW w AS (ORDER BY n);
+----+-----+-----+-----+-----+
| n | lag | lead | next_n | next_next_n |
+----+-----+-----+-----+-----+
| 1 | 0 | 1 | 1 | 2 |
| 1 | 1 | 2 | 2 | 3 |
| 2 | 1 | 3 | 3 | 5 |
| 3 | 2 | 5 | 5 | 8 |
| 5 | 3 | 8 | 8 | 13 |
| 8 | 5 | 0 | 13 | 8 |
+----+-----+-----+-----+-----+
```

Fibonacci 番号の初期セットを生成する方法の 1 つは、再帰的な共通テーブル式を使用することです。例については、[フィボナッチシリーズ世代](#)を参照してください。

MySQL 8.0.22 以降、この関数の `rows` 引数に負の値を使用することはできません。

- `LAST_VALUE(expr) [null_treatment] over_clause`

ウィンドウフレームの最後の行から `expr` の値を返します。

`over_clause` については、[セクション12.21.2「Window 関数の概念と構文」](#)を参照してください。`null_treatment` については、概要のセクションで説明しています。

例については、`FIRST_VALUE()` 関数の説明を参照してください。

- `LEAD(expr [, N[, default]]) [null_treatment] over_clause`

パーティション内の `N` 行ごとに現在の行を導く (後に続く) 行から、`expr` の値を返します。そのような行がない場合、戻り値は `default` です。たとえば、`N` が 3 の場合、戻り値は最後の 2 行の `default` です。`N` または `default` がいない場合、デフォルトはそれぞれ 1 および `NULL` です。

`N` は、負でないリテラル整数である必要があります。`N` が 0 の場合、`expr` は現在の行に対して評価されます。

MySQL 8.0.22 以降、`N` を `NULL` にすることはできません。さらに、次のいずれかの形式で、1 から 2^{63} までの範囲の整数である必要があります:

- 符号なし整数定数リテラル
- 位置パラメータマーカー (?)
- ユーザー定義変数
- ストアドルーチン内のローカル変数

`over_clause` については、[セクション12.21.2「Window 関数の概念と構文」](#) を参照してください。`null_treatment` については、概要のセクションで説明しています。

例については、`LAG()` 関数の説明を参照してください。

MySQL 8.0.22 以降では、この関数の `rows` 引数に負の値を使用することはできません。

- `NTH_VALUE(expr, N) [from_first_last] [null_treatment] over_clause`

ウィンドウフレームの `N` 番目の行から `expr` の値を返します。そのような行がない場合、戻り値は `NULL` です。

`N` はリテラルの正の整数である必要があります。

`from_first_last` は SQL 標準の一部ですが、MySQL 実装では `FROM FIRST` (デフォルト) のみが許可されます。つまり、ウィンドウの最初の行から計算が開始されます。`FROM LAST` は解析されますが、エラーが発生します。`FROM LAST` と同じ効果を得るには (ウィンドウの最後の行から計算を開始)、`ORDER BY` を使用して逆の順序でソートします。

`over_clause` については、[セクション12.21.2「Window 関数の概念と構文」](#) を参照してください。`null_treatment` については、概要のセクションで説明しています。

例については、`FIRST_VALUE()` 関数の説明を参照してください。

MySQL 8.0.22 以降では、この関数の行引数に `NULL` を使用できません。

- [NTILE\(N\) over_clause](#)

パーティションを N グループ (バケット) に分割し、パーティション内の各行にバケット番号を割り当て、パーティション内の現在の行のバケット番号を返します。たとえば、 N が 4 の場合、`NTILE()` は行を 4 つのバケットに分割します。 N が 100 の場合、`NTILE()` は行を 100 バケットに分割します。

N はリテラルの正の整数である必要があります。バケット番号の戻り値の範囲は 1 から N です。

MySQL 8.0.22 以降、 N を `NULL` にすることはできません。また、次のいずれかの形式で、1 から 2^{63} までの範囲の整数である必要があります:

- 符号なし整数定数リテラル
- 位置パラメータマーカ (?)
- ユーザー定義変数
- ストアドルーチン内のローカル変数

パーティション行を目的の順序にソートするには、`ORDER BY` でこの関数を使用する必要があります。

`over_clause` については、[セクション12.21.2「Window 関数の概念と構文」](#)を参照してください。

次のクエリーは、`val` カラムの値セットについて、行を複数または 4 つのグループに分割した結果のパーセンタイル値を示しています。参照のために、クエリーでは `ROW_NUMBER()` を使用して行番号も表示されます:

```
mysql> SELECT
  val,
  ROW_NUMBER() OVER w AS 'row_number',
  NTILE(2) OVER w AS 'ntile2',
  NTILE(4) OVER w AS 'ntile4'
FROM numbers
WINDOW w AS (ORDER BY val);
+----+-----+-----+-----+
| val | row_number | ntile2 | ntile4 |
+----+-----+-----+-----+
| 1 | 1 | 1 | 1 |
| 1 | 2 | 1 | 1 |
| 2 | 3 | 1 | 1 |
| 3 | 4 | 1 | 2 |
| 3 | 5 | 1 | 2 |
| 3 | 6 | 2 | 3 |
| 4 | 7 | 2 | 3 |
| 4 | 8 | 2 | 4 |
| 5 | 9 | 2 | 4 |
+----+-----+-----+-----+
```

MySQL 8.0.22 以降、構成 `NTILE(NULL)` は許可されなくなりました。

- [PERCENT_RANK\(\) over_clause](#)

現在の行の値より小さいパーティション値の割合を返します (最大値を除く)。戻り値の範囲は 0 から 1 で、次の式の結果として計算される行相対ランクを表します。ここで、`rank` は行ランク、`rows` はパーティション行数です:

$$\frac{(\text{rank} - 1)}{(\text{rows} - 1)}$$

パーティション行を目的の順序にソートするには、`ORDER BY` でこの関数を使用する必要があります。`ORDER BY` が無い場合、すべての行がピアになります。

`over_clause` については、[セクション12.21.2「Window 関数の概念と構文」](#)を参照してください。

例については、`CUME_DIST()` 関数の説明を参照してください。

- [RANK\(\) over_clause](#)

パーティション内の現在の行のランク (ギャップあり) を返します。同僚は同僚とみなされ、同じランクを受け取ります。複数のサイズのグループが存在する場合、この関数はピアグループに連続するランクを割り当てません。結果は連続しないランク番号になります。

パーティション行を目的の順序にソートするには、[ORDER BY](#) でこの関数を使用する必要があります。[ORDER BY](#) がない場合、すべての行がピアになります。

[over_clause](#) については、[セクション12.21.2「Window 関数の概念と構文」](#) を参照してください。

次のクエリーは、ギャップのあるランクを生成する [RANK\(\)](#) と、ギャップのないランクを生成する [DENSE_RANK\(\)](#) の違いを示しています。クエリーでは、`val` カラムの一連の値の各メンバーのランク値が表示されますが、これには重複が含まれています。[RANK\(\)](#) はピア (重複) に同じランク値を割り当て、次に大きい値はピア数からランクを引いてランクを高くします。[DENSE_RANK\(\)](#) ではピアにも同じランク値が割り当てられますが、次に高い値にはランクが 1 つ大きくなります。参照のために、クエリーでは [ROW_NUMBER\(\)](#) を使用して行番号も表示されます:

```
mysql> SELECT
  val,
  ROW_NUMBER() OVER w AS 'row_number',
  RANK() OVER w AS 'rank',
  DENSE_RANK() OVER w AS 'dense_rank'
FROM numbers
WINDOW w AS (ORDER BY val);
+-----+-----+-----+
| val | row_number | rank | dense_rank |
+-----+-----+-----+
| 1 | 1 | 1 | 1 |
| 1 | 2 | 1 | 1 |
| 2 | 3 | 3 | 2 |
| 3 | 4 | 4 | 3 |
| 3 | 5 | 4 | 3 |
| 3 | 6 | 4 | 3 |
| 4 | 7 | 7 | 4 |
| 4 | 8 | 7 | 4 |
| 5 | 9 | 9 | 5 |
+-----+-----+-----+
```

- [ROW_NUMBER\(\) over_clause](#)

パーティション内の現在の行の番号を返します。行番号の範囲は 1 からパーティション行の数です。

[ORDER BY](#) は、行の番号付けの順序に影響します。[ORDER BY](#) を使用しない場合、行番号付けは非決定的です。

[ROW_NUMBER\(\)](#) は、ピアに異なる行番号を割り当てます。ピアに同じ値を割り当てるには、[RANK\(\)](#) または [DENSE_RANK\(\)](#) を使用します。例については、[RANK\(\)](#) 関数の説明を参照してください。

[over_clause](#) については、[セクション12.21.2「Window 関数の概念と構文」](#) を参照してください。

12.21.2 Window 関数の概念と構文

このセクションでは、ウィンドウ関数の使用方法について説明します。例では、[セクション12.20.2「GROUP BY 修飾子」](#) の [GROUPING\(\)](#) 関数の説明にあるものと同じ販売情報データセットを使用します:

```
mysql> SELECT * FROM sales ORDER BY country, year, product;
+-----+-----+-----+
| year | country | product | profit |
+-----+-----+-----+
| 2000 | Finland | Computer | 1500 |
| 2000 | Finland | Phone | 100 |
| 2001 | Finland | Phone | 10 |
| 2000 | India | Calculator | 75 |
| 2000 | India | Calculator | 75 |
| 2000 | India | Computer | 1200 |
| 2000 | USA | Calculator | 75 |
| 2000 | USA | Computer | 1500 |
| 2001 | USA | Calculator | 50 |
| 2001 | USA | Computer | 1500 |
| 2001 | USA | Computer | 1200 |
+-----+-----+-----+
```



```

2001 | USA | TV | 150 |
2001 | USA | TV | 100 |
+-----+-----+-----+

```

ウィンドウ関数は、一連のクエリー行に対して集計のような操作を実行します。ただし、集計操作ではクエリー行が単一の結果行にグループ化されますが、ウィンドウ関数ではクエリー行ごとに結果が生成されます:

- 関数の評価が行われる行は、現在の行と呼ばれます。
- 関数評価が行われる現在の行に関連するクエリー行は、現在の行のウィンドウで構成されます。

たとえば、売上情報テーブルを使用すると、次の2つのクエリーで集計操作が実行され、グループとして取得されたすべての行に対して単一のグローバル合計が生成され、国ごとにグループ化されます:

```

mysql> SELECT SUM(profit) AS total_profit
        FROM sales;
+-----+
| total_profit |
+-----+
|      7535 |
+-----+
mysql> SELECT country, SUM(profit) AS country_profit
        FROM sales
        GROUP BY country
        ORDER BY country;
+-----+-----+
| country | country_profit |
+-----+-----+
| Finland |          1610 |
| India   |          1350 |
| USA     |          4575 |
+-----+-----+

```

対照的に、ウィンドウ操作では、クエリー行のグループは単一の出力行に縮小されません。かわりに、行ごとに結果が生成されます。前述のクエリーと同様に、次のクエリーでは `SUM()` を使用しますが、今回はウィンドウ関数として使用します:

```

mysql> SELECT
        year, country, product, profit,
        SUM(profit) OVER() AS total_profit,
        SUM(profit) OVER(PARTITION BY country) AS country_profit
        FROM sales
        ORDER BY country, year, product, profit;
+-----+-----+-----+-----+-----+-----+
| year | country | product | profit | total_profit | country_profit |
+-----+-----+-----+-----+-----+-----+
| 2000 | Finland | Computer | 1500 | 7535 | 1610 |
| 2000 | Finland | Phone   | 100  | 7535 | 1610 |
| 2001 | Finland | Phone   | 10   | 7535 | 1610 |
| 2000 | India   | Calculator | 75  | 7535 | 1350 |
| 2000 | India   | Calculator | 75  | 7535 | 1350 |
| 2000 | India   | Computer | 1200 | 7535 | 1350 |
| 2000 | USA     | Calculator | 75  | 7535 | 4575 |
| 2000 | USA     | Computer | 1500 | 7535 | 4575 |
| 2001 | USA     | Calculator | 50  | 7535 | 4575 |
| 2001 | USA     | Computer | 1200 | 7535 | 4575 |
| 2001 | USA     | Computer | 1500 | 7535 | 4575 |
| 2001 | USA     | TV       | 100  | 7535 | 4575 |
| 2001 | USA     | TV       | 150  | 7535 | 4575 |
+-----+-----+-----+-----+-----+-----+

```

クエリーの各ウィンドウ操作は、ウィンドウ関数で処理するためにクエリー行をグループにパーティション化する方法を指定する `OVER` 句を含めることで指定されます:

- 最初の `OVER` 句は空で、クエリー行のセット全体が単一のパーティションとして扱われます。このため、ウィンドウ関数ではグローバル合計が生成されますが、各行に対して生成されません。
- 2番目の `OVER` 句では、国ごとに行がパーティション化され、パーティションごとの合計が生成されます (国ごと)。この関数は、パーティション行ごとにこの合計を生成します。

ウィンドウ機能は、選択リストおよび **ORDER BY** 句でのみ使用できます。クエリー結果行は、**WHERE**、**GROUP BY** および **HAVING** の処理後に **FROM** 句から決定され、**ORDER BY**、**LIMIT** および **SELECT DISTINCT** の前にウィンドウ実行が行われます。

OVER 句は多くの集計関数で許可されているため、**OVER** 句が存在するかどうかに応じて、ウィンドウ関数または非ウィンドウ関数として使用できます:

```
AVG()
BIT_AND()
BIT_OR()
BIT_XOR()
COUNT()
JSON_ARRAYAGG()
JSON_OBJECTAGG()
MAX()
MIN()
STDDEV_POP(), STDDEV(), STD()
STDDEV_SAMP()
SUM()
VAR_POP(), VARIANCE()
VAR_SAMP()
```

各集計関数の詳細は、[セクション12.20.1「集計関数の説明」](#) を参照してください。

MySQL では、ウィンドウ関数としてのみ使用される非集計関数もサポートされています。これらの場合、**OVER** 句は必須です:

```
CUME_DIST()
DENSE_RANK()
FIRST_VALUE()
LAG()
LAST_VALUE()
LEAD()
NTH_VALUE()
NTILE()
PERCENT_RANK()
RANK()
ROW_NUMBER()
```

各非集計関数の詳細は、[セクション12.21.1「Window 関数の説明」](#) を参照してください。

これらの非集計ウィンドウ関数のいずれかの例として、このクエリーは、パーティション内の各行の行番号を生成する **ROW_NUMBER()** を使用します。この場合、行には国ごとに番号が付けられます。デフォルトでは、パーティション行は順序付けられず、行番号付けは非決定的です。パーティション行をソートするには、ウィンドウ定義に **ORDER BY** 句を含めます。このクエリーでは、順序付けされていないパーティション (**row_num1** カラムと **row_num2** カラム) を使用して、**ORDER BY** を省略した場合と含めた場合の違いを示します:

```
mysql> SELECT
  year, country, product, profit,
  ROW_NUMBER() OVER(PARTITION BY country) AS row_num1,
  ROW_NUMBER() OVER(PARTITION BY country ORDER BY year, product) AS row_num2
  FROM sales;
```

year	country	product	profit	row_num1	row_num2
2000	Finland	Computer	1500	2	1
2000	Finland	Phone	100	1	2
2001	Finland	Phone	10	3	3
2000	India	Calculator	75	2	1
2000	India	Calculator	75	3	2
2000	India	Computer	1200	1	3
2000	USA	Calculator	75	5	1
2000	USA	Computer	1500	4	2
2001	USA	Calculator	50	2	3
2001	USA	Computer	1500	3	4
2001	USA	Computer	1200	7	5
2001	USA	TV	150	1	6
2001	USA	TV	100	6	7

前述のように、ウィンドウ関数を使用する (または集計関数をウィンドウ関数として処理する) には、関数コールの後に **OVER** 句を含めます。 **OVER** 句には、次の 2 つの形式があります:

```
over_clause:
{OVER (window_spec) | OVER window_name}
```

どちらのフォームでも、ウィンドウ関数によるクエリー行の処理方法を定義します。これらは、ウィンドウが **OVER** 句で直接定義されているか、クエリーの他の場所で定義された名前付きウィンドウへの参照によって提供されているかによって異なります:

- 最初のケースでは、ウィンドウ指定は **OVER** 句のカッコの間に直接表示されます。
- 2 番目の場合、**window_name** は、クエリーの他の場所で **WINDOW** 句によって定義されたウィンドウ指定の名前です。詳細は、[セクション12.21.4「名前付きウィンドウ」](#)を参照してください。

OVER (window_spec) 構文の場合、ウィンドウ指定にはいくつかの部分があり、すべてオプションです:

```
window_spec:
[window_name] [partition_clause] [order_clause] [frame_clause]
```

OVER() が空の場合、ウィンドウはすべてのクエリー行で構成され、ウィンドウ関数はすべての行を使用して結果を計算します。それ以外の場合は、カッコ内にある句によって、関数結果の計算に使用されるクエリー行と、それらのパーティション化および順序付け方法が決まります:

- **window_name**: クエリーの他の場所で **WINDOW** 句によって定義されたウィンドウの名前。**window_name** 自身が **OVER** 句内に出現する場合は、ウィンドウを完全に定義します。パーティション化、順序付けまたはフレーム化も指定されている場合は、名前付きウィンドウの解釈が変更されます。詳細は、[セクション12.21.4「名前付きウィンドウ」](#)を参照してください。
- **partition_clause**: **PARTITION BY** 句は、クエリー行をグループに分割する方法を指定します。特定の行のウィンドウ関数の結果は、その行を含むパーティションの行に基づきます。**PARTITION BY** を省略すると、すべてのクエリー行で構成される単一のパーティションが存在します。

注記

ウィンドウ関数のパーティション化は、テーブルのパーティション化とは異なります。テーブルのパーティション化の詳細は、[第24章「パーティション化」](#)を参照してください。

partition_clause の構文は次のとおりです:

```
partition_clause:
PARTITION BY expr [, expr] ...
```

標準 SQL では、**PARTITION BY** の後にカラム名のみが続く必要があります。MySQL 拡張機能では、カラム名のみでなく式を使用できます。たとえば、テーブルに **ts** という名前の **TIMESTAMP** カラムが含まれている場合、標準 SQL では **PARTITION BY ts** は許可されますが **PARTITION BY HOUR(ts)** は許可されませんが、MySQL では両方が許可されます。

- **order_clause**: **ORDER BY** 句は、各パーティションの行をソートする方法を指定します。**ORDER BY** 句に従って等しいパーティション行はピアとみなされます。**ORDER BY** を省略すると、パーティション行は順序付けられず、処理順序は暗黙的に指定されず、すべてのパーティション行がピアになります。

order_clause の構文は次のとおりです:

```
order_clause:
ORDER BY expr [ASC|DESC] [, expr [ASC|DESC]] ...
```

オプションで、各 **ORDER BY** 式の後に **ASC** または **DESC** を指定してソート方向を示すことができます。方向が指定されていない場合、デフォルトは **ASC** です。**NULL** 値は、昇順ソートの場合は最初にソートされ、降順ソートの場合は最後にソートされます。

ウィンドウ定義の **ORDER BY** は、個々のパーティション内に適用されます。結果セット全体をソートするには、クエリーの最上位レベルに **ORDER BY** を含めます。

- **frame_clause**: フレームは現在のパーティションのサブセットであり、**frame** 句はサブセットの定義方法を指定します。**frame** 句には、独自の副次句が多数あります。詳細は、[セクション12.21.3「ウィンドウ機能フレーム仕様」](#)を参照してください。

12.21.3 ウィンドウ機能フレーム仕様

ウィンドウ関数で 사용되는ウィンドウの定義には、`frame` 句を含めることができます。フレームは現在のパーティションのサブセットであり、`frame` 句はサブセットの定義方法を指定します。

フレームは現在の行に対して決定されます。これにより、現在の行のパーティション内での位置に応じて、フレームをパーティション内で移動できます。例:

- パーティションの開始行から現在の行までのすべての行になるようにフレームを定義することで、各行の累積合計を計算できます。
- 現在の行の両側で N 行を拡張するようにフレームを定義することで、ローリング平均を計算できます。

次のクエリは、移動フレームを使用して、時間順序付けされた `level` 値の各グループ内の累積合計、および現在の行とその直前と直後の行から計算されたローリング平均を計算する方法を示しています:

```
mysql> SELECT
  time, subject, val,
  SUM(val) OVER (PARTITION BY subject ORDER BY time
    ROWS UNBOUNDED PRECEDING)
    AS running_total,
  AVG(val) OVER (PARTITION BY subject ORDER BY time
    ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING)
    AS running_average
  FROM observations;
```

time	subject	val	running_total	running_average
07:00:00	st113	10	10	9.5000
07:15:00	st113	9	19	14.6667
07:30:00	st113	25	44	18.0000
07:45:00	st113	20	64	22.5000
07:00:00	xh458	0	0	5.0000
07:15:00	xh458	10	10	5.0000
07:30:00	xh458	5	15	15.0000
07:45:00	xh458	30	45	20.0000
08:00:00	xh458	25	70	27.5000

`running_average` カラムの場合、最初のカラムの前または最後のカラムの後にフレーム行はありません。このような場合、`AVG()` は使用可能な行の平均を計算します。

ウィンドウ関数として使用される集計関数は、次の非集計ウィンドウ関数と同様に、現在の行フレームの行を操作します:

```
FIRST_VALUE()
LAST_VALUE()
NTH_VALUE()
```

標準 SQL は、パーティション全体で動作するウィンドウ関数に `frame` 句を含めないことを指定します。MySQL では、このような関数の `frame` 句は許可されますが、無視されます。これらの関数は、フレームが指定されている場合でもパーティション全体を使用します:

```
CUME_DIST()
DENSE_RANK()
LAG()
LEAD()
NTILE()
PERCENT_RANK()
RANK()
ROW_NUMBER()
```

`frame` 句が指定されている場合、構文は次のとおりです:

```
frame_clause:
  frame_units frame_extent

frame_units:
  {ROWS | RANGE}
```

frame 句がない場合、このセクションの後半で説明するように、デフォルトのフレームは **ORDER BY** 句が存在するかどうかによって異なります。

frame_units 値は、現在の行とフレーム行の関係のタイプを示します:

- **ROWS**: フレームは、開始行と終了行の位置によって定義されます。オフセットは、現在の行番号と行番号の違いです。
- **RANGE**: フレームは、値の範囲内の行によって定義されます。オフセットは、現在の行の値と行の値の違いです。

frame_extent 値は、フレームの開始点と終了点を示します。フレームの開始のみを指定するか (この場合、現在の行が暗黙的に終了します)、**BETWEEN** を使用して両方のフレームエンドポイントを指定できます:

```
frame_extent:
  {frame_start | frame_between}

frame_between:
  BETWEEN frame_start AND frame_end

frame_start, frame_end: {
  CURRENT ROW
  | UNBOUNDED PRECEDING
  | UNBOUNDED FOLLOWING
  | expr PRECEDING
  | expr FOLLOWING
}
```

BETWEEN 構文では、**frame_start** を **frame_end** より後にはできません。

許可される **frame_start** および **frame_end** の値には、次の意味があります:

- **CURRENT ROW**: **ROWS** の場合、バインドは現在の行です。 **RANGE** の場合、バインドは現在の行のペアです。
- **UNBOUNDED PRECEDING**: バインドは最初のパーティション行です。
- **UNBOUNDED FOLLOWING**: バインドは最後のパーティション行です。
- **expr PRECEDING**: **ROWS** の場合、バインドは現在の行の前の **expr** 行です。 **RANGE** の場合、バインドされるのは、現在の行の値から **expr** を引いた値を持つ行です。現在の行の値が **NULL** の場合、バインドされるのは行のペアです。

expr PRECEDING (および **expr FOLLOWING**) の場合、**expr** は ? パラメータマーカー (プリペアドステートメントで使用)、負でない数値リテラルまたは **INTERVAL val unit** 形式の時間間隔になります。 **INTERVAL** 式の場合、**val** は負でない間隔値を指定し、**unit** は値を解釈する単位を示すキーワードです。(許可されている **units** 指定子の詳細は、[セクション12.7「日付および時間関数」](#) の **DATE_ADD()** 関数の説明を参照してください。)

数値または時間的 **expr** 上の **RANGE** には、それぞれ数値式または時間的式上の **ORDER BY** が必要です。

有効な **expr PRECEDING** および **expr FOLLOWING** インジケータの例:

```
10 PRECEDING
INTERVAL 5 DAY PRECEDING
5 FOLLOWING
INTERVAL '2:30' MINUTE_SECOND FOLLOWING
```

- **expr FOLLOWING**: **ROWS** の場合、バインドは現在の行の後の **expr** 行です。 **RANGE** の場合、バインドされるのは、現在の行の値に **expr** を加えた値を持つ行です。現在の行の値が **NULL** の場合、バインドされるのは行のペアです。

expr の許容値については、**expr PRECEDING** の説明を参照してください。

次のクエリは、**FIRST_VALUE()**、**LAST_VALUE()** および **NTH_VALUE()** の2つのインスタンスを示しています:

```
mysql> SELECT
  time, subject, val,
  FIRST_VALUE(val) OVER w AS 'first',
  LAST_VALUE(val) OVER w AS 'last',
  NTH_VALUE(val, 2) OVER w AS 'second',
```

```
NTH_VALUE(val, 4) OVER w AS 'fourth'
FROM observations
WINDOW w AS (PARTITION BY subject ORDER BY time
ROWS UNBOUNDED PRECEDING);
```

time	subject	val	first	last	second	fourth
07:00:00	st113	10	10	10	NULL	NULL
07:15:00	st113	9	10	9	9	NULL
07:30:00	st113	25	10	25	9	NULL
07:45:00	st113	20	10	20	9	20
07:00:00	xh458	0	0	0	NULL	NULL
07:15:00	xh458	10	0	10	10	NULL
07:30:00	xh458	5	0	5	10	NULL
07:45:00	xh458	30	0	30	10	30
08:00:00	xh458	25	0	25	10	30

各関数は、現在のフレーム内の行を使用します。この行は、表示されているウィンドウ定義に従って、最初のパーティション行から現在の行に拡張されます。NTH_VALUE() コールの場合、現在のフレームにはリクエストされた行が常に含まれるわけではありません。このような場合、戻り値は NULL です。

frame 句がない場合、デフォルトのフレームは ORDER BY 句が存在するかどうかによって異なります:

- **ORDER BY** を使用: デフォルトのフレームには、現在の行のすべてのピア (ORDER BY 句に従って現在の行と等しい行) を含む、パーティションの開始行から現在の行までの行が含まれます。デフォルトは、次のフレーム仕様と同等です:

```
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
```

- **ORDER BY** なし: デフォルトのフレームには、すべてのパーティション行が含まれます (ORDER BY がない場合、すべてのパーティション行はピアであるため)。デフォルトは、次のフレーム仕様と同等です:

```
RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
```

デフォルトのフレームは ORDER BY の有無によって異なるため、ORDER BY をクエリーに追加して決定的な結果を取得すると、結果が変わる可能性があります。(たとえば、SUM() によって生成される値は変更される可能性があります。) 同じ結果を取得するが、ORDER BY ごとに順序付けするには、ORDER BY が存在するかどうかに関係なく、使用する明示的なフレーム仕様を指定します。

現在の行の値が NULL の場合、フレーム指定の意味は明白ではない可能性があります。その場合、次の例は様々なフレーム仕様ごどのように適用されるかを示しています:

- **ORDER BY X ASC RANGE BETWEEN 10 FOLLOWING AND 15 FOLLOWING**

フレームは NULL で始まり、NULL で停止するため、値が NULL の行のみが含まれます。

- **ORDER BY X ASC RANGE BETWEEN 10 FOLLOWING AND UNBOUNDED FOLLOWING**

フレームは NULL から始まり、パーティションの最後で停止します。ASC ソートでは NULL 値が最初に配置されるため、フレームはパーティション全体になります。

- **ORDER BY X DESC RANGE BETWEEN 10 FOLLOWING AND UNBOUNDED FOLLOWING**

フレームは NULL から始まり、パーティションの最後で停止します。DESC ソートでは NULL 値が最後に配置されるため、フレームは NULL 値のみです。

- **ORDER BY X ASC RANGE BETWEEN 10 PRECEDING AND UNBOUNDED FOLLOWING**

フレームは NULL から始まり、パーティションの最後で停止します。ASC ソートでは NULL 値が最初に配置されるため、フレームはパーティション全体になります。

- **ORDER BY X ASC RANGE BETWEEN 10 PRECEDING AND 10 FOLLOWING**

フレームは NULL で始まり、NULL で停止するため、値が NULL の行のみが含まれます。

- **ORDER BY X ASC RANGE BETWEEN 10 PRECEDING AND 1 PRECEDING**

フレームは `NULL` で始まり、`NULL` で停止するため、値が `NULL` の行のみが含まれます。

- `ORDER BY X ASC RANGE BETWEEN UNBOUNDED PRECEDING AND 10 FOLLOWING`

フレームはパーティションの先頭から始まり、値が `NULL` の行で停止します。`ASC` ソートでは `NULL` 値が最初に配置されるため、フレームは `NULL` 値のみです。

12.21.4 名前付きウィンドウ

Windows を定義し、`OVER` 句でそれらを参照するための名前を付けることができます。これを行うには、`WINDOW` 句を使用します。クエリーに存在する場合、`WINDOW` 句は `HAVING` 句と `ORDER BY` 句の位置の間にあり、構文は次のとおりです：

```
WINDOW window_name AS (window_spec)
[, window_name AS (window_spec)] ...
```

各ウィンドウ定義について、セクション12.21.2「Window 関数の概念と構文」で説明されているように、`window_name` はウィンドウ名で、`window_spec` は `OVER` 句のカッコの間に指定されているものと同じタイプのウィンドウ指定です：

```
window_spec:
[window_name] [partition_clause] [order_clause] [frame_clause]
```

`WINDOW` 句は、複数の `OVER` 句が同じウィンドウを定義するクエリーに役立ちます。かわりに、ウィンドウを一度定義して名前を付け、`OVER` 句でその名前を参照できます。同じウィンドウを複数回定義する次のクエリーについて考えてみます：

```
SELECT
  val,
  ROW_NUMBER() OVER (ORDER BY val) AS 'row_number',
  RANK() OVER (ORDER BY val) AS 'rank',
  DENSE_RANK() OVER (ORDER BY val) AS 'dense_rank'
FROM numbers;
```

`WINDOW` を使用してウィンドウを一度定義し、`OVER` 句でウィンドウを名前で参照するだけで、クエリーをより簡単に記述できます：

```
SELECT
  val,
  ROW_NUMBER() OVER w AS 'row_number',
  RANK() OVER w AS 'rank',
  DENSE_RANK() OVER w AS 'dense_rank'
FROM numbers
WINDOW w AS (ORDER BY val);
```

名前付きウィンドウを使用すると、クエリー結果への影響を確認するために、ウィンドウ定義を簡単に試すことができます。複数の `OVER` 句定義ではなく、`WINDOW` 句のウィンドウ定義のみを変更する必要があります。

`OVER` 句で `OVER window_name` ではなく `OVER (window_name ...)` を使用する場合、他の句を追加して名前付きウィンドウを変更できます。たとえば、次のクエリーはパーティション化を含むウィンドウを定義し、`OVER` 句で `ORDER BY` を使用してウィンドウを様々な方法で変更します：

```
SELECT
  DISTINCT year, country,
  FIRST_VALUE(year) OVER (w ORDER BY year ASC) AS first,
  FIRST_VALUE(year) OVER (w ORDER BY year DESC) AS last
FROM sales
WINDOW w AS (PARTITION BY country);
```

`OVER` 句は、名前付きウィンドウにのみプロパティを追加でき、変更はできません。名前付きウィンドウ定義にパーティション化、順序付けまたはフレーミングプロパティが含まれている場合、ウィンドウ名を参照する `OVER` 句にも同じ種類のプロパティを含めることはできず、そうしないとエラーが発生します：

- ウィンドウ定義と参照元の `OVER` 句に同じ種類のプロパティが含まれていないため、この構成は許可されず：

```
OVER (w ORDER BY country)
... WINDOW w AS (PARTITION BY country)
```

- **OVER** 句では、すでに **PARTITION BY** を持つ名前付きウィンドウに対して **PARTITION BY** が指定されているため、この構成は許可されません:

```
OVER (w PARTITION BY year)
... WINDOW w AS (PARTITION BY country)
```

名前付きウィンドウの定義自体を **window_name** で始めることができます。このような場合、前方参照と後方参照は許可されますが、循環は許可されません:

- これは許可されます。前方参照と後方参照は含まれますが、循環は含まれません:

```
WINDOW w1 AS (w2), w2 AS (), w3 AS (w1)
```

- サイクルが含まれているため、これは許可されません:

```
WINDOW w1 AS (w2), w2 AS (w3), w3 AS (w1)
```

12.21.5 ウィンドウ機能の制限事項

SQL 標準では、**UPDATE** ステートメントまたは **DELETE** ステートメントで行を更新するために使用できないという制約がウィンドウ関数に課されます。このような関数は、これらのステートメントのサブクエリーで (行を選択するために) 使用できます。

MySQL では、次のウィンドウ機能はサポートされていません:

- 集計ウィンドウ関数の **DISTINCT** 構文。
- ネストされたウィンドウ関数。
- 現在の行の値に依存する動的フレームエンドポイント。

パーサーは、サポートされていない次のウィンドウ構成を認識します:

- **GROUPS** フレーム単位指定子は解析されますが、エラーが発生します。 **ROWS** および **RANGE** のみがサポートされます。
- フレーム指定の **EXCLUDE** 句は解析されますが、エラーが発生します。
- **IGNORE NULLS** は解析されますが、エラーが発生します。 **RESPECT NULLS** のみがサポートされています。
- **FROM LAST** は解析されますが、エラーが発生します。 **FROM FIRST** のみがサポートされています。

12.22 パフォーマンススキーマ関数

MySQL 8.0.16 の時点では、MySQL には、パフォーマンススキーマデータを書式設定または取得する組込み SQL 関数が含まれており、対応する **sys** スキーマストアドファンクションの同等の機能として使用できます。組込み関数は、**sys.** スキーマ修飾子を必要とする **sys** 関数や、**sys** が現在のスキーマである **sys** 関数とは異なり、任意のスキーマで起動でき、修飾子は必要ありません。

表 12.27 「パフォーマンススキーマ関数」

名前	説明	導入
FORMAT_BYTES()	バイト数を単位付きの値に変換	8.0.16
FORMAT_PICO_TIME()	時間をピコ秒単位で値に変換	8.0.16
PS_CURRENT_THREAD_ID()	現在のスレッドのパフォーマンススキーマスレッド ID	8.0.16
PS_THREAD_ID()	指定されたスレッドのパフォーマンススキーマスレッド ID	8.0.16

組込み関数は、非推奨の対応する **sys** 関数よりも優先されます。将来のバージョンの MySQL で削除される予定です。 **sys** 関数を使用するアプリケーションは、かわりに組込み関数を使用するように調整する必要があります。 **sys** 関数と組込み関数の若干の違いに注意してください。これらの違いの詳細は、このセクションの関数の説明を参照してください。

- [FORMAT_BYTES\(count\)](#)

数値バイト数を指定すると、人間が読める形式に変換され、値と単位インジケータで構成される文字列が返されます。文字列には、小数点以下 2 桁に丸められたバイト数と、3 桁以上の有効桁数が含まれます。1024 バイト未満の数値は整数として表され、丸められません。

単位インジケータは、次のテーブルに示すようにバイトカウント引数のサイズによって異なります。

引数値	結果単位	結果ユニットインジケータ
1023 まで	バイト	バイト
$1024^2 - 1$ まで	キビバイト	KiB
$1024^3 - 1$ まで	mebibytes	MiB
$1024^4 - 1$ まで	gibibytes	GiB
$1024^5 - 1$ まで	テビバイト	TiB
$1024^6 - 1$ まで	ペビバイト	PiB
1024 以上の ⁶	exbibytes	EiB

```
mysql> SELECT FORMAT_BYTES(512), FORMAT_BYTES(18446644073709551615);
+-----+-----+
| FORMAT_BYTES(512) | FORMAT_BYTES(18446644073709551615) |
+-----+-----+
| 512 bytes      | 16.00 EiB                          |
+-----+-----+
```

[FORMAT_BYTES\(\)](#) が MySQL 8.0.16 に追加されました。sys スキーマ [format_bytes\(\)](#) 関数のかわりに使用でき、次の違いに注意してください:

- [FORMAT_BYTES\(\)](#) では、EiB ユニットインジケータが使用されます。[sys.format_bytes\(\)](#) ではサポートされません。
- [FORMAT_PICO_TIME\(time_val\)](#)

数値のパフォーマンススキーマレイテンシまたは待機時間をピコ秒単位で指定すると、それを人間が読める形式に変換し、値と単位インジケータで構成される文字列を返します。文字列には、小数点以下 2 桁に丸められた小数点以下の桁数と、3 桁以上の有効桁数が含まれます。1 ナノ秒未満の時間は整数で表され、丸められません。

単位インジケータは、次のテーブルに示すように、time-value 引数のサイズによって異なります。

引数値	結果単位	結果ユニットインジケータ
最大 $10^3 - 1$	ピコ秒	ps
最大 $10^6 - 1$	ナノ秒	ns
最大 $10^9 - 1$	マイクロ秒	us
最大 $10^{12} - 1$	ミリ秒	ms
最大 $60 \times 10^{12} - 1$	秒	s
$3.6 \times 10^{15} - 1$ まで	minutes	min
$8.64 \times 10^{16} - 1$ まで	hours	h
8.64×10^{16} 以上	days	d

```
mysql> SELECT FORMAT_PICO_TIME(3501), FORMAT_PICO_TIME(188732396662000);
+-----+-----+
| FORMAT_PICO_TIME(3501) | FORMAT_PICO_TIME(188732396662000) |
+-----+-----+
| 3.50 ns                | 3.15 min                          |
+-----+-----+
```

[FORMAT_PICO_TIME\(\)](#) が MySQL 8.0.16 に追加されました。sys スキーマ [format_time\(\)](#) 関数のかわりに使用でき、次の相違点に注意してください:

- 分を示すために、`sys.format_time()` では `m` ユニットインジケータを使用し、`FORMAT_PICO_TIME()` では `min` を使用します。
- `sys.format_time()` では、`w` (週) 単位インジケータが使用されます。`FORMAT_PICO_TIME()` はそうではありません。
- `PS_CURRENT_THREAD_ID()`

現在の接続に割り当てられているパフォーマンススキーマスレッド ID を表す `BIGINT UNSIGNED` 値を返します。

スレッド ID の戻り値は、「パフォーマンススキーマ」テーブルの `THREAD_ID` カラムで指定された型の値です。

パフォーマンススキーマの構成は、`PS_THREAD_ID()` の場合と同様に `PS_CURRENT_THREAD_ID()` に影響しません。詳細は、その関数の説明を参照してください。

```
mysql> SELECT PS_CURRENT_THREAD_ID();
+-----+
| PS_CURRENT_THREAD_ID() |
+-----+
|          52 |
+-----+
mysql> SELECT PS_THREAD_ID(CONNECTION_ID());
+-----+
| PS_THREAD_ID(CONNECTION_ID()) |
+-----+
|          52 |
+-----+
```

`PS_CURRENT_THREAD_ID()` が MySQL 8.0.16 に追加されました。これは、`NULL` または `CONNECTION_ID()` の引数を使用して `sys` スキーマ `ps_thread_id()` 関数を起動するショートカットとして使用できます。

- `PS_THREAD_ID(connection_id)`

接続 ID を指定すると、接続 ID に割り当てられたパフォーマンススキーマスレッド ID を表す `BIGINT UNSIGNED` 値を返します。接続 ID のスレッド ID が存在しない場合は、`NULL` を返します。後者は、インストールされていないスレッドで発生する可能性があります。

接続 ID 引数は、パフォーマンススキーマ `threads` テーブルの `PROCESSLIST_ID` カラムまたは `SHOW PROCESSLIST` 出力の `Id` カラムで指定されたタイプの値です。

スレッド ID の戻り値は、「パフォーマンススキーマ」テーブルの `THREAD_ID` カラムで指定された型の値です。

パフォーマンススキーマの構成は、次のように `PS_THREAD_ID()` の操作に影響します。(これらの備考は、`PS_CURRENT_THREAD_ID()` にも適用されます。)

- `thread_instrumentation` コンシューマを無効にすると、スレッドレベルでの統計の収集および集計は無効になりますが、`PS_THREAD_ID()` には影響しません。
- `performance_schema_max_thread_instances` が 0 でない場合、パフォーマンススキーマはスレッド統計用にメモリを割り当て、インスタンスメモリが使用可能な各スレッドに内部 ID を割り当てます。インスタンスメモリが使用できないスレッドがある場合、`PS_THREAD_ID()` は `NULL` を返します。この場合、`Performance_schema_thread_instances_lost` はゼロ以外です。
- `performance_schema_max_thread_instances` が 0 の場合、パフォーマンススキーマはスレッドメモリを割り当てず、`PS_THREAD_ID()` は `NULL` を返します。
- パフォーマンススキーマ自体が無効になっている場合、`PS_THREAD_ID()` はエラーを生成します。

```
mysql> SELECT PS_THREAD_ID(6);
+-----+
| PS_THREAD_ID(6) |
+-----+
|          45 |
+-----+
```

+-----+

`PS_THREAD_ID()` が MySQL 8.0.16 に追加されました。 `sys` スキーマ `ps_thread_id()` 関数のかわりに使用でき、次の違いに注意してください:

- `NULL` の引数を指定すると、`sys.ps_thread_id()` は現在の接続のスレッド ID を返し、`PS_THREAD_ID()` は `NULL` を返します。現在の接続スレッド ID を取得するには、かわりに `PS_CURRENT_THREAD_ID()` を使用します。

12.23 内部関数

表 12.28 「内部関数」

名前	説明	導入
<code>CAN_ACCESS_COLUMN()</code>	内部使用のみ	
<code>CAN_ACCESS_DATABASE()</code>	内部使用のみ	
<code>CAN_ACCESS_TABLE()</code>	内部使用のみ	
<code>CAN_ACCESS_USER()</code>	内部使用のみ	8.0.22
<code>CAN_ACCESS_VIEW()</code>	内部使用のみ	
<code>GET_DD_COLUMN_PRIVILEGES()</code>	内部使用のみ	
<code>GET_DD_CREATE_OPTIONS()</code>	内部使用のみ	
<code>GET_DD_INDEX_SUB_PART_LENGTH()</code>	内部使用のみ	
<code>INTERNAL_AUTO_INCREMENT()</code>	内部使用のみ	
<code>INTERNAL_AVG_ROW_LENGTH()</code>	内部使用のみ	
<code>INTERNAL_CHECK_TIME()</code>	内部使用のみ	
<code>INTERNAL_CHECKSUM()</code>	内部使用のみ	
<code>INTERNAL_DATA_FREE()</code>	内部使用のみ	
<code>INTERNAL_DATA_LENGTH()</code>	内部使用のみ	
<code>INTERNAL_DD_CHAR_LENGTH()</code>	内部使用のみ	
<code>INTERNAL_GET_COMMENT_OR_ERROR()</code>	内部使用のみ	
<code>INTERNAL_GET_ENABLED_ROLE_JS()</code>	内部使用のみ	8.0.19
<code>INTERNAL_GET_HOSTNAME()</code>	内部使用のみ	8.0.19
<code>INTERNAL_GET_USERNAME()</code>	内部使用のみ	8.0.19
<code>INTERNAL_GET_VIEW_WARNING_OR_ERROR()</code>	内部使用のみ	
<code>INTERNAL_INDEX_COLUMN_CARDINALITY()</code>	内部使用のみ	
<code>INTERNAL_INDEX_LENGTH()</code>	内部使用のみ	
<code>INTERNAL_IS_ENABLED_ROLE()</code>	内部使用のみ	8.0.19
<code>INTERNAL_IS_MANDATORY_ROLE()</code>	内部使用のみ	8.0.19
<code>INTERNAL_KEYS_DISABLED()</code>	内部使用のみ	
<code>INTERNAL_MAX_DATA_LENGTH()</code>	内部使用のみ	
<code>INTERNAL_TABLE_ROWS()</code>	内部使用のみ	
<code>INTERNAL_UPDATE_TIME()</code>	内部使用のみ	

このセクションに記載されている関数は、サーバーによる内部使用のみを目的としています。ユーザーがそれらを出そうとすると、エラーになります。

- `CAN_ACCESS_COLUMN(ARGS)`
- `CAN_ACCESS_DATABASE(ARGS)`
- `CAN_ACCESS_TABLE(ARGS)`

- CAN_ACCESS_USER(ARGS)
- CAN_ACCESS_VIEW(ARGS)
- GET_DD_COLUMN_PRIVILEGES(ARGS)
- GET_DD_CREATE_OPTIONS(ARGS)
- GET_DD_INDEX_SUB_PART_LENGTH(ARGS)
- INTERNAL_AUTO_INCREMENT(ARGS)
- INTERNAL_AVG_ROW_LENGTH(ARGS)
- INTERNAL_CHECK_TIME(ARGS)
- INTERNAL_CHECKSUM(ARGS)
- INTERNAL_DATA_FREE(ARGS)
- INTERNAL_DATA_LENGTH(ARGS)
- INTERNAL_DD_CHAR_LENGTH(ARGS)
- INTERNAL_GET_COMMENT_OR_ERROR(ARGS)
- INTERNAL_GET_ENABLED_ROLE_JSON(ARGS)
- INTERNAL_GET_HOSTNAME(ARGS)
- INTERNAL_GET_USERNAME(ARGS)
- INTERNAL_GET_VIEW_WARNING_OR_ERROR(ARGS)
- INTERNAL_INDEX_COLUMN_CARDINALITY(ARGS)
- INTERNAL_INDEX_LENGTH(ARGS)
- INTERNAL_IS_ENABLED_ROLE(ARGS)
- INTERNAL_IS_MANDATORY_ROLE(ARGS)
- INTERNAL_KEYS_DISABLED(ARGS)
- INTERNAL_MAX_DATA_LENGTH(ARGS)
- INTERNAL_TABLE_ROWS(ARGS)
- INTERNAL_UPDATE_TIME(ARGS)
- IS_VISIBLE_DD_OBJECT(ARGS)

12.24 その他の関数

表 12.29 「その他の関数」

名前	説明
ANY_VALUE()	ONLY_FULL_GROUP_BY 値の拒否を抑止します
BIN_TO_UUID()	バイナリ UUID を文字列に変換
DEFAULT()	テーブルカラムのデフォルト値を返します
GROUPING()	ROLLUP 行と通常の行の区別
INET_ATON()	IP アドレスの数値を返します
INET_NTOA()	数値から IP アドレスを返します
INET6_ATON()	IPv6 アドレスの数値を返します
INET6_NTOA()	数値から IPv6 アドレスを返します

名前	説明
IS_IPV4()	引数が IPv4 アドレスかどうか
IS_IPV4_COMPAT()	引数が IPv4 互換アドレスかどうか
IS_IPV4_MAPPED()	引数が IPv4 マップアドレスかどうか
IS_IPV6()	引数が IPv6 アドレスかどうか
IS_UUID()	引数が有効な UUID かどうか
MASTER_POS_WAIT()	レプリカが指定された位置までのすべての更新を読み取り、適用するまでブロック
NAME_CONST()	カラムの名前を指定
SLEEP()	ある秒数間スリープ状態にします
UUID()	ユニバーサル固有識別子 (UUID) を返します
UUID_SHORT()	整数値のユニバーサル識別子を返します
UUID_TO_BIN()	文字列 UUID をバイナリに変換
VALUES()	INSERT 中に使用する値の定義

- **ANY_VALUE(arg)**

この関数は、MySQL が特定できない理由で有効であるとわかっているクエリーを MySQL が拒否する場合に、**ONLY_FULL_GROUP_BY** SQL モードが有効な **GROUP BY** クエリーに役立ちます。関数の戻り値および型は、その引数の戻り値および型と同じですが、関数の結果は **ONLY_FULL_GROUP_BY** SQL モードではチェックされません。

たとえば、**name** がインデックス付けされていないカラムの場合、次のクエリーは **ONLY_FULL_GROUP_BY** が有効になっていると失敗します:

```
mysql> SELECT name, address, MAX(age) FROM t GROUP BY name;
ERROR 1055 (42000): Expression #2 of SELECT list is not in GROUP
BY clause and contains nonaggregated column 'mydb.t.address' which
is not functionally dependent on columns in GROUP BY clause; this
is incompatible with sql_mode=only_full_group_by
```

address は、**GROUP BY** カラム間で名前が付けられておらず、機能的に依存していない非集計カラムであるため、障害が発生します。その結果、各 **name** グループ内の行の **address** 値は非決定的になります。MySQL でクエリーを受け入れるには、複数の方法があります:

- テーブルを変更して、**name** を主キーまたは一意の **NOT NULL** カラムにします。これにより、MySQL は、**address** が **name** に機能的に依存していること、つまり **address** が **name** によって一意に決定されていることを判別できます。(NULL を有効な **name** 値として許可する必要がある場合、この方法は適用できません。)

- `ANY_VALUE()` を使用して、`address` を参照します:

```
SELECT name, ANY_VALUE(address), MAX(age) FROM t GROUP BY name;
```

この場合、MySQL は各 `name` グループ内の `address` 値の非決定性を無視し、クエリーを受け入れます。これは、グループごとに選択される非集計カラムの値を気にしない場合に便利です。`ANY_VALUE()` は、`SUM()` や `COUNT()` などの関数とは異なり、集計関数ではありません。非決定的なテストを抑制するだけです。

- `ONLY_FULL_GROUP_BY` を無効にします。これは、前の項目で説明したように、`ONLY_FULL_GROUP_BY` を有効にして `ANY_VALUE()` を使用することと同等です。

`ANY_VALUE()` は、カラム間に関数従属性が存在するが、MySQL がそれを判別できない場合にも役立ちます。`age` は機能的にグループ化カラム `age-1` に依存しているため、次のクエリーは有効ですが、MySQL は `ONLY_FULL_GROUP_BY` が有効になっているクエリーを通知および拒否できません:

```
SELECT age FROM t GROUP BY age-1;
```

MySQL がクエリーを受け入れるようにするには、`ANY_VALUE()` を使用します:

```
SELECT ANY_VALUE(age) FROM t GROUP BY age-1;
```

`ANY_VALUE()` は、`GROUP BY` 句がない場合に集計関数を参照するクエリーに使用できます:

```
mysql> SELECT name, MAX(age) FROM t;
ERROR 1140 (42000): In aggregated query without GROUP BY, expression
#1 of SELECT list contains nonaggregated column 'mydb.t.name'; this
is incompatible with sql_mode=only_full_group_by
```

`GROUP BY` がない場合、単一のグループが存在し、どの `name` 値をグループに選択するかは非決定的です。`ANY_VALUE()` は、クエリーを受け入れるように MySQL に指示します:

```
SELECT ANY_VALUE(name), MAX(age) FROM t;
```

特定のデータセットの一部のプロパティのために、選択した非集計カラムが `GROUP BY` カラムに実質的に依存していることがわかっている場合があります。たとえば、アプリケーションでは、あるカラムを別のカラムに対して一意にすることができます。この場合、効果的に機能的に依存するカラムに `ANY_VALUE()` を使用すると意味がある可能性があります。

追加の説明については、[セクション12.20.3「MySQLでのGROUP BYの処理」](#)を参照してください。

- `BIN_TO_UUID(binary_uuid)`, `BIN_TO_UUID(binary_uuid, swap_flag)`

`BIN_TO_UUID()` は、`UUID_TO_BIN()` の逆です。バイナリ UUID を文字列 UUID に変換し、結果を返します。バイナリ値は、`VARBINARY(16)` 値として UUID である必要があります。戻り値は、ダッシュで区切られた5つの16進数の utf8 文字列です。(この形式の詳細は、`UUID()` 関数の説明を参照してください。) UUID 引数が `NULL` の場合、戻り値は `NULL` です。無効な引数がある場合は、エラーが発生します。

`BIN_TO_UUID()` は、次のいずれかまたは2つの引数を取ります:

- 1つの引数形式はバイナリ UUID 値を取ります。UUID 値は、その時間の低い部分と時間の高い部分がスワップされていないと想定されます。文字列の結果は、バイナリ引数と同じ順序になります。
- 2つの引数形式は、バイナリ UUID 値と `swap-flag` 値を取ります:
 - `swap_flag` が0の場合、2つの引数の形式は1つの引数の形式と同等です。文字列の結果は、バイナリ引数と同じ順序になります。
 - `swap_flag` が1の場合、UUID 値には時間の低い部分と時間の高い部分がスワップされているとみなされます。これらのパーツは、結果値の元の位置にスワップバックされます。

使用例および時間部分スワッピングの詳細は、`UUID_TO_BIN()` 関数の説明を参照してください。

- `DEFAULT(col_name)`

テーブルカラムのデフォルト値を返します。カラムにデフォルト値がない場合は、エラーが発生します。

名前付きカラムのデフォルト値を指定するための `DEFAULT(col_name)` の使用は、リテラルのデフォルト値を持つカラムにのみ許可され、式のデフォルト値を持つカラムには許可されません。

```
mysql> UPDATE t SET i = DEFAULT(i)+1 WHERE id < 100;
```

- `FORMAT(X,D)`

数値 `X` を '#,###,###.##' のような書式に変換し、小数点第 `D` 位に丸めて、その結果を文字列として返します。詳細は、[セクション12.8「文字列関数および演算子」](#)を参照してください。

- `GROUPING(expr [, expr] ...)`

`WITH ROLLUP` 修飾子を含む `GROUP BY` クエリーの場合、`ROLLUP` 操作により、`NULL` がすべての値のセットを表す超集計出力行が生成されます。`GROUPING()` 関数を使用すると、スーパー集計行の `NULL` 値を、通常のグループ化された行の `NULL` 値と区別できます。

`GROUPING()` は、`SELECT` リストまたは `HAVING` 句でのみ使用できます。

`GROUPING()` の各引数は、`GROUP BY` 句の式と完全に一致する式である必要があります。式は位置指定子でできません。現在の行の式の値がスーパー集計値を表す `NULL` である場合、`GROUPING()` は式ごとに 1 を生成します。それ以外の場合、`GROUPING()` は、式の値が正規結果行の `NULL` であるか、`NULL` ではないことを示す 0 を生成します。

テーブル `t1` に次の行が含まれ、`NULL` が「other」や「不明」などを示しているとします:

```
mysql> SELECT * FROM t1;
+----+-----+-----+
| name | size | quantity |
+----+-----+-----+
| ball | small | 10 |
| ball | large | 20 |
| ball | NULL | 5 |
| hoop | small | 15 |
| hoop | large | 5 |
| hoop | NULL | 3 |
+----+-----+-----+
```

`WITH ROLLUP` を使用しないテーブルのサマリーは、次のようになります:

```
mysql> SELECT name, size, SUM(quantity) AS quantity
FROM t1
GROUP BY name, size;
+----+-----+-----+
| name | size | quantity |
+----+-----+-----+
| ball | small | 10 |
| ball | large | 20 |
| ball | NULL | 5 |
| hoop | small | 15 |
| hoop | large | 5 |
| hoop | NULL | 3 |
+----+-----+-----+
```

結果には `NULL` 値が含まれていますが、クエリーには `WITH ROLLUP` が含まれていないため、それらは上位集計行を表していません。

`WITH ROLLUP` を追加すると、追加の `NULL` 値を含む上位集計サマリー行が生成されます。ただし、この結果を前の結果と比較しないと、どの `NULL` 値がスーパー集計行に出現し、どの値が通常のグループ化された行に出現するかを簡単に確認できません:

```
mysql> SELECT name, size, SUM(quantity) AS quantity
FROM t1
GROUP BY name, size WITH ROLLUP;
+----+-----+-----+
| name | size | quantity |
```

```

+----+-----+
| ball | NULL | 5 |
| ball | large | 20 |
| ball | small | 10 |
| ball | NULL | 35 |
| hoop | NULL | 3 |
| hoop | large | 5 |
| hoop | small | 15 |
| hoop | NULL | 23 |
| NULL | NULL | 58 |
+----+-----+

```

スーパー集計行の `NULL` 値を通常のグループ化行の `NULL` 値と区別するには、`GROUPING()` を使用します。これは、スーパー集計 `NULL` 値に対してのみ `1` を返します:

```

mysql> SELECT
  name, size, SUM(quantity) AS quantity,
  GROUPING(name) AS grp_name,
  GROUPING(size) AS grp_size
FROM t1
GROUP BY name, size WITH ROLLUP;
+----+-----+-----+-----+-----+
| name | size | quantity | grp_name | grp_size |
+----+-----+-----+-----+-----+
| ball | NULL | 5 | 0 | 0 |
| ball | large | 20 | 0 | 0 |
| ball | small | 10 | 0 | 0 |
| ball | NULL | 35 | 0 | 1 |
| hoop | NULL | 3 | 0 | 0 |
| hoop | large | 5 | 0 | 0 |
| hoop | small | 15 | 0 | 0 |
| hoop | NULL | 23 | 0 | 1 |
| NULL | NULL | 58 | 1 | 1 |
+----+-----+-----+-----+-----+

```

`GROUPING()` の一般的な使用方法:

- スーパー集計の `NULL` 値をラベルに置き換えます:

```

mysql> SELECT
  IF(GROUPING(name) = 1, 'All items', name) AS name,
  IF(GROUPING(size) = 1, 'All sizes', size) AS size,
  SUM(quantity) AS quantity
FROM t1
GROUP BY name, size WITH ROLLUP;
+----+-----+-----+
| name | size | quantity |
+----+-----+-----+
| ball | NULL | 5 |
| ball | large | 20 |
| ball | small | 10 |
| ball | All sizes | 35 |
| hoop | NULL | 3 |
| hoop | large | 5 |
| hoop | small | 15 |
| hoop | All sizes | 23 |
| All items | All sizes | 58 |
+----+-----+-----+

```

- 通常のグループ化された明細をフィルタで除外して、超集約明細のみを返します:

```

mysql> SELECT name, size, SUM(quantity) AS quantity
FROM t1
GROUP BY name, size WITH ROLLUP
HAVING GROUPING(name) = 1 OR GROUPING(size) = 1;
+----+-----+-----+
| name | size | quantity |
+----+-----+-----+
| ball | NULL | 35 |
| hoop | NULL | 23 |
| NULL | NULL | 58 |
+----+-----+-----+

```

`GROUPING()` では、複数の式引数が許可されます。この場合、`GROUPING()` の戻り値は、各式の結果から結合されたビットマスクを表します。ここで、最下位ビットは右端の式の結果に対応します。たとえば、式の引数が 3 つある場合、`GROUPING(expr1, expr2, expr3)` は次のように評価されます：

```
result for GROUPING(expr3)
+ result for GROUPING(expr2) << 1
+ result for GROUPING(expr1) << 2
```

次のクエリーは、複数引数コールで単一の引数の `GROUPING()` 結果を組み合わせてビットマスク値を生成する方法を示しています：

```
mysql> SELECT
  name, size, SUM(quantity) AS quantity,
  GROUPING(name) AS grp_name,
  GROUPING(size) AS grp_size,
  GROUPING(name, size) AS grp_all
FROM t1
GROUP BY name, size WITH ROLLUP;
```

name	size	quantity	grp_name	grp_size	grp_all
ball	NULL	5	0	0	0
ball	large	20	0	0	0
ball	small	10	0	0	0
ball	NULL	35	0	1	1
hoop	NULL	3	0	0	0
hoop	large	5	0	0	0
hoop	small	15	0	0	0
hoop	NULL	23	0	1	1
NULL	NULL	58	1	1	3

複数の式引数では、いずれかの式がスーパー集計値を表す場合、`GROUPING()` の戻り値はゼロ以外になります。したがって、複数引数の `GROUPING()` 構文では、複数の単一引数コールではなく単一の複数引数 `GROUPING()` コールを使用して、上位集計行のみを戻す以前のクエリーを簡単に記述できます：

```
mysql> SELECT name, size, SUM(quantity) AS quantity
FROM t1
GROUP BY name, size WITH ROLLUP
HAVING GROUPING(name, size) <> 0;
```

name	size	quantity
ball	NULL	35
hoop	NULL	23
NULL	NULL	58

`GROUPING()` の使用には、次の制限事項があります：

- 照合が失敗する可能性があるため、サブクエリーの `GROUP BY` 式を `GROUPING()` 引数として使用しないでください。たとえば、次のクエリーの照合は失敗します：

```
mysql> SELECT GROUPING((SELECT MAX(name) FROM t1))
FROM t1
GROUP BY (SELECT MAX(name) FROM t1) WITH ROLLUP;
ERROR 3580 (HY000): Argument #1 of GROUPING function is not in GROUP BY
```

- `GROUP BY` リテラル式は、`HAVING` 句内で `GROUPING()` 引数として使用しないでください。オプティマイザが `GROUP BY` と `HAVING` を評価するタイミングの違いにより、照合は成功する可能性があります。が、`GROUPING()` の評価では予期した結果が生成されません。次のクエリーについて考えてみます：

```
SELECT a AS f1, 'w' AS f2
FROM t
GROUP BY f1, f2 WITH ROLLUP
```

```
HAVING GROUPING(f2) = 1;
```

`GROUPING()` は、リテラル定数式に対して、`HAVING` 句全体より早く評価され、0 を返します。このようなクエリーが影響を受けるかどうかを確認するには、`EXPLAIN` を使用して `Extra` カラムで `Impossible having` を探します。

`WITH ROLLUP` および `GROUPING()` の詳細は、[セクション12.20.2「GROUP BY 修飾子」](#) を参照してください。

• `INET_ATON(expr)`

IPv4 ネットワークアドレスのドット区切り表現が文字列として指定された場合、アドレスの数値を表す整数をネットワークバイト順序 (ビッグエンディアン) で返します。引数が認識されない場合、`INET_ATON()` は `NULL` を返します。

```
mysql> SELECT INET_ATON('10.0.5.9');
-> 167773449
```

この例では、戻り値は $10 \times 256^3 + 0 \times 256^2 + 5 \times 256 + 9$ として計算されます。

`INET_ATON()` は、短い形式の IP アドレス ('127.0.0.1' の表現として '127.1' など) の場合に、非 `NULL` を返す場合と返さない場合があります。このため、このようなアドレスには `INET_ATON()` を使用しないでください。

注記

`INET_ATON()` で生成された値を格納するには、署名される `INT` ではなく、`INT UNSIGNED` カラムを使用します。署名付きのカラムを使用する場合は、第 1 オクテットが 127 よりも大きい IP アドレスに対応する値を正常に格納できません。[セクション 11.1.7「範囲外およびオーバーフローの処理」](#) を参照してください。

• `INET_NTOA(expr)`

ネットワークバイト順に数値の IPv4 ネットワークアドレスを指定すると、接続文字セットの文字列としてアドレスのドット区切りの文字列表現を返します。引数が認識されない場合、`INET_NTOA()` は `NULL` を返します。

```
mysql> SELECT INET_NTOA(167773449);
-> '10.0.5.9'
```

• `INET6_ATON(expr)`

IPv6 または IPv4 ネットワークアドレスが文字列として指定された場合、アドレスの数値を表すバイナリ文字列をネットワークバイト順序 (ビッグエンディアン) で返します。数値書式の IPv6 アドレスでは最大の整数型よりも大きいバイトが必要であるため、この関数で返される表現のデータ型は、`VARBINARY` (IPv6 アドレスの場合は `VARBINARY(16)`、IPv4 アドレスの場合は `VARBINARY(4)`) です。引数が有効なアドレスでない場合、`INET6_ATON()` は `NULL` を返します。

次の例では、`HEX()` を使用して `INET6_ATON()` の結果を出力可能な形式で表示します。

```
mysql> SELECT HEX(INET6_ATON('fdfe::5a55:caff:fefa:9089'));
-> 'FDFE0000000000005A55CAFFFEFA9089'
mysql> SELECT HEX(INET6_ATON('10.0.5.9'));
-> '0A000509'
```

`INET6_ATON()` は、有効な引数上の複数の制約を監視します。これらについては、次のリストに例とともに示します。

- `fe80::3%1` や `fe80::3%eth0` のような末尾のゾーン ID は許可されません。
- `2001:45f:3:ba::/64` や `198.51.100.0/24` のような末尾のネットワークマスクは許可されません。
- IPv4 アドレスを表す値では、クラスレスアドレスのみがサポートされます。`198.51.1` などのクラスフルアドレスは拒否されます。`198.51.100.2:8080` のような末尾のポート番号は許可されません。`198.0xa0.1.2` のようなアドレスコンポーネント内の 16 進数は許可されません。8 進数はサポートされていません。`198.51.010.1` は

198.51.8.1 ではなく、198.51.10.1 として処理されます。このような IPv4 の制約は、IPv4 アドレス部分を持つ IPv6 アドレス (IPv4 互換アドレスや IPv4 マップアドレスなど) にも適用されます。

INT 値として数値形式で表現された IPv4 アドレス `expr` を、VARBINARY 値として数値形式で表現された IPv6 アドレスに変換するには、次の式を使用します。

```
INET6_ATON(INET_NTOA(expr))
```

例:

```
mysql> SELECT HEX(INET6_ATON(INET_NTOA(167773449)));
-> '0A000509'
```

• INET6_NTOA(expr)

数値形式でバイナリ文字列として表される IPv6 または IPv4 ネットワークアドレスを指定すると、接続文字セットの文字列としてアドレスの文字列表現を返します。引数が有効なアドレスでない場合、INET6_NTOA() は NULL を返します。

INET6_NTOA() には、次のようなプロパティがあります。

- 変換はオペレーティングシステムの機能を使用して実行されないため、出力文字列はプラットフォームに依存しません。
- 戻り文字列の最大長は 39 (4 x 8 + 7) です。次のステートメントが指定された場合、

```
CREATE TABLE t AS SELECT INET6_NTOA(expr) AS c1;
```

結果として生成されるテーブルに次の定義が含まれます。

```
CREATE TABLE t (c1 VARCHAR(39) CHARACTER SET utf8 DEFAULT NULL);
```

- 戻り文字列では、IPv6 アドレスを表す小文字が使用されます。

```
mysql> SELECT INET6_NTOA(INET6_ATON('fdfe::5a55:caff:fefa:9089'));
-> 'fdfe::5a55:caff:fefa:9089'
mysql> SELECT INET6_NTOA(INET6_ATON('10.0.5.9'));
-> '10.0.5.9'

mysql> SELECT INET6_NTOA(UNHEX('DFFE000000000005A55CAFFFEFA9089'));
-> 'fdfe::5a55:caff:fefa:9089'
mysql> SELECT INET6_NTOA(UNHEX('0A000509'));
-> '10.0.5.9'
```

• IS_IPV4(expr)

引数が文字列として指定された有効な IPv4 アドレスの場合は 1 を返し、それ以外の場合は 0 を返します。

```
mysql> SELECT IS_IPV4('10.0.5.9'), IS_IPV4('10.0.5.256');
-> 1, 0
```

特定の引数について、IS_IPV4() が 1 を返した場合、INET_ATON() (および INET6_ATON()) は NULL 以外を返しません。逆のステートメントは該当しません。一部のケースでは、IS_IPV4() が 0 を返すと、INET_ATON() は NULL 以外を返します。

上記の備考で示したように、IS_IPV4() は、有効な IPv4 アドレスを構成するものに関して、INET_ATON() よりも厳密であるため、無効な値に対して強固なチェックを実行する必要があるアプリケーションで役立つことがあります。または、INET6_ATON() を使用して、IPv4 アドレスを内部形式に変換し、(無効なアドレスを示す) NULL の結果をチェックします。INET6_ATON() は、IPv4 アドレスのチェックという点では、IS_IPV4() と同等の強固さです。

- `IS_IPV4_COMPAT(expr)`

この関数には、`INET6_ATON()` で返されるように、バイナリ文字列として数値形式で表現された IPv6 アドレスが指定されます。引数が有効な IPv4 互換 IPv6 アドレスの場合は 1 を返し、それ以外の場合は 0 を返します。IPv4 互換アドレスの形式は、`::ipv4_address` です。

```
mysql> SELECT IS_IPV4_COMPAT(INET6_ATON('::10.0.5.9'));
-> 1
mysql> SELECT IS_IPV4_COMPAT(INET6_ATON('::ffff:10.0.5.9'));
-> 0
```

IPv4 互換アドレスの IPv4 部分は、16 進表記法を使用して表現することもできます。たとえば、`198.51.100.1` には、次のような生の 16 進値が含まれます。

```
mysql> SELECT HEX(INET6_ATON('198.51.100.1'));
-> 'C6336401'
```

IPv4 互換形式で表現された `::198.51.100.1` は、`::c0a8:0001` または (先頭のゼロなしの) `::c0a8:1` と同等です。

```
mysql> SELECT
-> IS_IPV4_COMPAT(INET6_ATON('::198.51.100.1')),
-> IS_IPV4_COMPAT(INET6_ATON('::c0a8:0001')),
-> IS_IPV4_COMPAT(INET6_ATON('::c0a8:1'));
-> 1, 1, 1
```

- `IS_IPV4_MAPPED(expr)`

この関数には、`INET6_ATON()` で返されるように、バイナリ文字列として数値形式で表現された IPv6 アドレスが指定されます。引数が有効な IPv4 マップ IPv6 アドレスの場合は 1 を返し、それ以外の場合は 0 を返します。IPv4 マップアドレスの形式は、`::ffff:ipv4_address` です。

```
mysql> SELECT IS_IPV4_MAPPED(INET6_ATON('::10.0.5.9'));
-> 0
mysql> SELECT IS_IPV4_MAPPED(INET6_ATON('::ffff:10.0.5.9'));
-> 1
```

`IS_IPV4_COMPAT()` と同様に、IPv4 マップアドレスの IPv4 部分は、16 進表記法を使用して表現することもできます。

```
mysql> SELECT
-> IS_IPV4_MAPPED(INET6_ATON('::ffff:198.51.100.1')),
-> IS_IPV4_MAPPED(INET6_ATON('::ffff:c0a8:0001')),
-> IS_IPV4_MAPPED(INET6_ATON('::ffff:c0a8:1'));
-> 1, 1, 1
```

- `IS_IPV6(expr)`

引数が文字列として指定された有効な IPv6 アドレスの場合は 1 を返し、それ以外の場合は 0 を返します。この関数では、IPv4 アドレスが有効な IPv6 アドレスとみなされません。

```
mysql> SELECT IS_IPV6('10.0.5.9'), IS_IPV6('::1');
-> 0, 1
```

特定の引数について、`IS_IPV6()` が 1 を返した場合、`INET6_ATON()` は `NULL` 以外を返します。

- `IS_UUID(string_uuid)`

引数が有効な文字列形式 UUID の場合は 1、引数が有効な UUID でない場合は 0、引数が `NULL` の場合は `NULL` を返します。

「有効」は、値が解析可能な形式であることを意味します。つまり、正しい長さで、許可されている文字のみが含まれます (任意の文字の 16 進数の数字と、オプションでダッシュおよび中カッコ)。この形式は最も一般的です:

```
aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee
```

次の形式も使用できます:

```
aaaaaaaaabbbbccccdddeeeeeeeeeeee
{aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee}
```

境内のフィールドの意味については、`UUID()` 関数の説明を参照してください。

```
mysql> SELECT IS_UUID('6ccd780c-baba-1026-9564-5b8c656024db');
+-----+
| IS_UUID('6ccd780c-baba-1026-9564-5b8c656024db') |
+-----+
| 1 |
+-----+
mysql> SELECT IS_UUID('6CCD780C-BABA-1026-9564-5B8C656024DB');
+-----+
| IS_UUID('6CCD780C-BABA-1026-9564-5B8C656024DB') |
+-----+
| 1 |
+-----+
mysql> SELECT IS_UUID('6ccd780cbaba102695645b8c656024db');
+-----+
| IS_UUID('6ccd780cbaba102695645b8c656024db') |
+-----+
| 1 |
+-----+
mysql> SELECT IS_UUID('{6ccd780c-baba-1026-9564-5b8c656024db}');
+-----+
| IS_UUID('{6ccd780c-baba-1026-9564-5b8c656024db}') |
+-----+
| 1 |
+-----+
mysql> SELECT IS_UUID('6ccd780c-baba-1026-9564-5b8c6560');
+-----+
| IS_UUID('6ccd780c-baba-1026-9564-5b8c6560') |
+-----+
| 0 |
+-----+
mysql> SELECT IS_UUID(RAND());
+-----+
| IS_UUID(RAND()) |
+-----+
| 0 |
+-----+
```

- `MASTER_POS_WAIT(log_name,log_pos[,timeout][,channel])`

この関数は、ソース/レプリカの同期化の制御に役立ちます。レプリカがソースバイナリログ内の指定された位置までのすべての更新を読み取って適用するまでブロックされます。戻り値は、レプリカが指定された位置に進むまで待機する必要があったログイベントの数です。レプリケーション SQL スレッドが開始されていない場合、レプリカソース情報が初期化されていない場合、引数が正しくない場合、またはエラーが発生した場合、関数は `NULL` を返します。タイムアウトを超えた場合は、`-1` を返します。`MASTER_POS_WAIT()` の待機中にレプリケーション SQL スレッドが停止した場合、この関数は `NULL` を返します。レプリカが指定された位置を過ぎた場合、関数はただちに返ります。

マルチスレッドのレプリカでは、この関数は、レプリカのステータスを更新するためにチェックポイント操作がコールされたときに、`slave_checkpoint_group` または `slave_checkpoint_period` システム変数によって設定された制

限の期限が切れるまで待機します。したがって、システム変数の設定によっては、この関数は指定された位置に達した後に時間を返す場合があります。

バイナリログのトランザクション圧縮が使用中で、指定された位置のトランザクションペイロードが (`Transaction_payload_event` として) 圧縮されている場合、この関数はトランザクション全体が読み取られて適用され、位置が更新されるまで待機します。

`timeout` 値が指定されている場合、`MASTER_POS_WAIT()` は、`timeout` 秒が経過した時点で待機を停止します。`timeout` は 0 よりも大きい値にする必要があります。`timeout` がゼロまたは負の値である場合は、タイムアウトがないことを意味します。

オプションの `channel` 値を使用すると、関数が適用されるレプリケーションチャンネルの名前を指定できます。詳しくは [セクション 17.2.2 「レプリケーションチャンネル」](#) をご覧ください。

この関数は、ステートメントベースのレプリケーションでは安全に使用できません。`binlog_format` が `STATEMENT` に設定されているときに、この関数を使用すると、警告のログが記録されます。

- `NAME_CONST(name,value)`

指定された値を返します。結果セットのカラムを生成する際に `NAME_CONST()` を使用すると、指定された名前がカラムに付けられます。引数には定数を指定してください。

```
mysql> SELECT NAME_CONST('myname', 14);
+-----+
| myname |
+-----+
| 14 |
+-----+
```

この関数は、内部でのみ使用されます。サーバーは、[セクション 25.7 「ストアプログラムバイナリロギング」](#) で説明されているように、ローカルプログラム変数への参照を含むストアプログラムからステートメントを記述するときを使用します。この関数は、`mysqlbinlog` からの出力に表示される場合があります。

アプリケーションで次のような単純なエイリアスを使用しても、上記で示した例とまったく同じ結果を取得できません。

```
mysql> SELECT 14 AS myname;
+-----+
| myname |
+-----+
| 14 |
+-----+
1 row in set (0.00 sec)
```

カラムのエイリアスについての詳細は、[セクション 13.2.10 「SELECT ステートメント」](#) を参照してください。

- `SLEEP(duration)`

`duration` 引数で指定された秒数間スリープ状態に (一時停止) してから、0 を返します。この期間には、小数部分が含まれている場合もあります。引数が `NULL` または負の場合、`SLEEP()` は厳密な SQL モードで警告またはエラーを生成します。

スリープが正常に (中断なしで) 復帰すると、0 が返されます:

```
mysql> SELECT SLEEP(1000);
+-----+
| SLEEP(1000) |
+-----+
| 0 |
+-----+
```

`SLEEP()` が、中断されたクエリーによって起動される唯一のものである場合、1 を返し、クエリー自体はエラーを返しません。これは、クエリーが強制終了されたかタイムアウトしたかにかかわらず当てはまります:

- このステートメントは、別のセッションから `KILL QUERY` を使用して中断されます:

```
mysql> SELECT SLEEP(1000);
```

```
+-----+
| SLEEP(1000) |
+-----+
|      1 |
+-----+
```

- このステートメントは、タイムアウトによって中断されます:

```
mysql> SELECT /*+ MAX_EXECUTION_TIME(1) */ SLEEP(1000);
+-----+
| SLEEP(1000) |
+-----+
|      1 |
+-----+
```

`SLEEP()` が中断されたクエリーの一部である場合、クエリーはエラーを返します:

- このステートメントは、別のセッションから `KILL QUERY` を使用して中断されます:

```
mysql> SELECT 1 FROM t1 WHERE SLEEP(1000);
ERROR 1317 (70100): Query execution was interrupted
```

- このステートメントは、タイムアウトによって中断されます:

```
mysql> SELECT /*+ MAX_EXECUTION_TIME(1000) */ 1 FROM t1 WHERE SLEEP(1000);
ERROR 3024 (HY000): Query execution was interrupted, maximum statement
execution time exceeded
```

この関数は、ステートメントベースのレプリケーションでは安全に使用できません。 `binlog_format` が `STATEMENT` に設定されているときに、この関数を使用すると、警告のログが記録されます。

• UUID()

RFC 4122, 「汎用一意の Identifier (UUID) URN ネームスペース」 (<http://www.ietf.org/rfc/rfc4122.txt>) に従って生成された Universal Unique Identifier (UUID) を返します。

UUID は、空間と時間においてグローバルに一意の数字として設計されています。これらのコールが相互に接続されていない 2 つの別々のデバイスで実行された場合でも、`UUID()` への 2 つのコールでは 2 つの異なる値が生成されることが予想されます。

警告

`UUID()` 値の目的は一意性を保つことですが、必ずしも推測不可能または予測不可能であるとはかぎりません。予測不可能性が必要である場合は、UUID 値を何か別の方法で生成してください。

`UUID()` は、RFC 4122 で説明されている UUID バージョン 1 に準拠する値を返します。値は 128 ビットの数値で、`aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee` 形式の 5 つの 16 進数の utf8 文字列として表されます:

- 最初の 3 つの数値は、タイムスタンプの下位、中間および上位の部分から生成されます。上位には UUID バージョン番号も含まれます。
- 4 番目の数字には、(たとえば、サマータイムが原因で) タイムスタンプ値の単調性が失われる場合に備えて、時間の一意性が保持されます。
- 5 番目の数字は、空間の一意性を提供する IEEE 802 ノード番号です。ランダムな番号が代入されるのは、後者が使用できない場合です (たとえば、ホストデバイスにイーサネットカードがない場合や、ホストオペレーティングシステムでインタフェースのハードウェアアドレスを検出する方法が不明な場合など)。この場合、空間の一意性は保証できません。しかしそれでも、競合が発生する可能性は非常に低くなります。

インタフェースの MAC アドレスは、FreeBSD、Linux および Windows でのみ考慮されます。その他のオペレーティングシステムでは、ランダムに生成された 48 ビットの数字が MySQL で使用されます。

```
mysql> SELECT UUID();
```

```
-> '6ccd780c-baba-1026-9564-5b8c656024db'
```

文字列とバイナリ UUID の値を変換するには、`UUID_TO_BIN()` 関数と `BIN_TO_UUID()` 関数を使用します。文字列が有効な UUID 値であるかどうかを確認するには、`IS_UUID()` 関数を使用します。

この関数は、ステートメントベースのレプリケーションでは安全に使用できません。`binlog_format` が `STATEMENT` に設定されているときに、この関数を使用すると、警告のログが記録されます。

• `UUID_SHORT()`

「short」汎用識別子を 64 ビット符号なし整数として返します。`UUID_SHORT()` によって返される値は、`UUID()` 関数によって返される文字列形式 128 ビット識別子とは異なり、一意性プロパティが異なります。次の条件を満たす場合は、`UUID_SHORT()` の値が一意であることが保証されます。

- 現在のサーバーの `server_id` 値は 0 から 255 の間で、ソースサーバーとレプリカサーバーのセット間で一意です
- `mysqld` の再起動間にサーバーホストのシステム時間を設定しません
- `mysqld` の再起動間に、平均 16 百万回/秒未満で `UUID_SHORT()` を起動

`UUID_SHORT()` の戻り値は、次のように構成されます。

```
(server_id & 255) << 56
+ (server_startup_time_in_seconds << 24)
+ incremented_variable++;
```

```
mysql> SELECT UUID_SHORT();
-> 92395783831158784
```

注記

`UUID_SHORT()` は、ステートメントベースのレプリケーションでは正しく動作しません。

• `UUID_TO_BIN(string_uuid)`, `UUID_TO_BIN(string_uuid, swap_flag)`

文字列 UUID をバイナリ UUID に変換し、結果を返します。(`IS_UUID()` 関数の説明には、許可されている文字列 UUID 形式がリストされます。) 戻りバイナリ UUID は `VARBINARY(16)` 値です。UUID 引数が `NULL` の場合、戻り値は `NULL` です。無効な引数がある場合は、エラーが発生します。

`UUID_TO_BIN()` は、次のいずれかまたは 2 つの引数を取ります:

- 1 つの引数形式は、文字列 UUID 値を取ります。バイナリ結果は、文字列引数と同じ順序になります。
- 2 つの引数形式は、文字列 UUID 値とフラグ値を取ります:
 - `swap_flag` が 0 の場合、2 つの引数の形式は 1 つの引数の形式と同等です。バイナリ結果は、文字列引数と同じ順序になります。
 - `swap_flag` が 1 の場合、戻り値の形式は異なります: time-low 部分と time-high 部分 (それぞれ 16 進数の最初と 3 番目のグループ) がスワップされます。これにより、より迅速に変化する部分が右側に移動し、結果がインデックス付けされたカラムに格納されている場合はインデックス付けの効率を向上させることができます。

タイムパーティスワッピングでは、UUID バージョン 1 の値 (`UUID()` 関数によって生成される値など) を使用することを前提としています。他の方法で生成された UUID 値がバージョン 1 形式に従っていない場合、時間部分スワッピングには利点がありません。バージョン 1 形式の詳細は、`UUID()` 関数の説明を参照してください。

次の文字列 UUID 値があるとします:

```
mysql> SET @uuid = '6ccd780c-baba-1026-9564-5b8c656024db';
```

時間部分スワップの有無にかかわらず文字列 UUID をバイナリに変換するには、`UUID_TO_BIN()` を使用します:

```
mysql> SELECT HEX(UUID_TO_BIN(@uuid));
+-----+
```



```

| HEX(UUID_TO_BIN(@uuid)) |
+-----+
| 6CCD780CBABA102695645B8C656024DB |
+-----+
mysql> SELECT HEX(UUID_TO_BIN(@uuid, 0));
+-----+
| HEX(UUID_TO_BIN(@uuid, 0)) |
+-----+
| 6CCD780CBABA102695645B8C656024DB |
+-----+
mysql> SELECT HEX(UUID_TO_BIN(@uuid, 1));
+-----+
| HEX(UUID_TO_BIN(@uuid, 1)) |
+-----+
| 1026BABA6CCD780C95645B8C656024DB |
+-----+

```

`UUID_TO_BIN()` によって返されたバイナリ UUID を文字列 UUID に変換するには、`BIN_TO_UUID()` を使用します。時間部分を入れ替えるために、第 2 引数 1 を指定して `UUID_TO_BIN()` を呼び出してバイナリ UUID を生成する場合は、バイナリ UUID を文字列 UUID に戻すときに時間部分のスイッチを解除するために、第 2 引数 1 を `BIN_TO_UUID()` に渡すことも必要です:

```

mysql> SELECT BIN_TO_UUID(UUID_TO_BIN(@uuid));
+-----+
| BIN_TO_UUID(UUID_TO_BIN(@uuid)) |
+-----+
| 6ccd780c-baba-1026-9564-5b8c656024db |
+-----+
mysql> SELECT BIN_TO_UUID(UUID_TO_BIN(@uuid,0),0);
+-----+
| BIN_TO_UUID(UUID_TO_BIN(@uuid,0),0) |
+-----+
| 6ccd780c-baba-1026-9564-5b8c656024db |
+-----+
mysql> SELECT BIN_TO_UUID(UUID_TO_BIN(@uuid,1),1);
+-----+
| BIN_TO_UUID(UUID_TO_BIN(@uuid,1),1) |
+-----+
| 6ccd780c-baba-1026-9564-5b8c656024db |
+-----+

```

時間部分スイッチングの使用が両方の方向の変換で同じでない場合、元の UUID は適切にリカバリされません:

```

mysql> SELECT BIN_TO_UUID(UUID_TO_BIN(@uuid,0),1);
+-----+
| BIN_TO_UUID(UUID_TO_BIN(@uuid,0),1) |
+-----+
| baba1026-780c-6ccd-9564-5b8c656024db |
+-----+
mysql> SELECT BIN_TO_UUID(UUID_TO_BIN(@uuid,1),0);
+-----+
| BIN_TO_UUID(UUID_TO_BIN(@uuid,1),0) |
+-----+
| 1026baba-6ccd-780c-9564-5b8c656024db |
+-----+

```

- `VALUES(col_name)`

`INSERT ... ON DUPLICATE KEY UPDATE` ステートメントでは、`UPDATE` 句の `VALUES(col_name)` 関数を使用すると、ステートメントの `INSERT` 部分からカラム値を参照できます。つまり、`UPDATE` 句の `VALUES(col_name)` は、挿入される `col_name` 値を参照するため、重複キーの競合が発生しなくなります。この関数は、複数の行を挿入する際に特に役立ちます。`VALUES()` 関数は、`INSERT` ステートメントの `ON DUPLICATE KEY UPDATE` 句でのみ有効であり、それ以外の場合は `NULL` を返します。[セクション13.2.6.2「INSERT ... ON DUPLICATE KEY UPDATE ステートメント」](#)を参照してください。

```
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3),(4,5,6)
```

```
-> ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

重要

この使用法は MySQL 8.0.20 では非推奨であり、MySQL の将来のリリースで削除される予定です。かわりに、行のエイリアスまたは行とカラムのエイリアスを使用してください。詳細および例については、[セクション13.2.6.2「INSERT ... ON DUPLICATE KEY UPDATE ステートメント」](#)を参照してください。

12.25 高精度計算

MySQL では、精度計算がサポートされます。数値の処理によって非常に正確な結果が得られ、無効な値をより詳細に制御できます。高精度計算は、次の 2 つの機能に基づいています。

- サーバーが無効なデータの受け入れまたは拒否に関してどの程度厳密かを制御する SQL モード。
- 固定小数点数演算のための MySQL ライブラリ。

これらの機能は数値操作にいくつかの影響を与えると同時に、標準 SQL への高度の準拠を実現します。

- 正確な計算: 厳密値数値の場合、計算に浮動小数点エラーは導入されません。代わりに、正確な精度が使用されます。たとえば、MySQL は .0001 などの数値を近似値ではなく厳密値として処理し、それを 10,000 回合計すると 1 に「近い」だけの値ではなく、正確に 1 の結果が生成されます。
- 適切に定義された丸め動作: 厳密値数値の場合、[ROUND\(\)](#) の結果は、ベースとなる C ライブラリの動作方法などの環境要因ではなく、その引数に依存します。
- プラットフォームからの独立: 厳密値数値に対する操作は、Windows や Unix などの各プラットフォームにわたって同じです。
- 無効な値の処理に対する制御: オーバーフローおよび 0 による除算が検出可能であり、これらはエラーとして処理できます。たとえば、あるカラムに対して大きすぎる値は、そのカラムのデータ型の範囲内に収まるように値が切り捨てられるようにするのではなく、エラーとして処理できます。同様に、0 による除算は、[NULL](#) の結果を生成する操作としてではなく、エラーとして処理できます。どちらのアプローチをとるかの選択は、サーバー SQL モードの設定によって決定されます。

次の説明では、古いアプリケーションとの非互換性の可能性を含め、高精度計算の動作方法のいくつかの側面について触れています。最後に、MySQL が数値操作を正確に処理する方法を示すいくつかの例を示します。SQL モードの制御については、[セクション5.1.11「サーバー SQL モード」](#)を参照してください。

12.25.1 数値の型

厳密値の操作に関する高精度計算の範囲には、厳密値データ型 (整数および [DECIMAL](#) 型) と厳密値数値リテラルが含まれます。近似値データ型および数値リテラルは、浮動小数点数として処理されます。

正確値の数値リテラルには、整数部または小数部、あるいはその両方が含まれています。これらには符号を付けることができます。例: 1、.2、3.4、-5、-6.78、+9.10。

近似値の数値リテラルは仮数と指数による指数表現で表されます。一方または両方の部分に符号を付けることができます。例: 1.2E3、1.2E-3、-1.2E3、-1.2E-3。

2 つの数値が同じように見えても、別々に扱われることがあります。たとえば、2.34 は (固定小数点の) 正確値ですが、2.34E0 は (浮動小数点の) 近似値です。

[DECIMAL](#) データ型は固定小数点型で、計算は正確です。MySQL では、[DECIMAL](#) 型には、[NUMERIC](#)、[DEC](#)、[FIXED](#) という複数のシノニムがあります。整数型も正確値型です。

[FLOAT](#) データ型および [DOUBLE](#) データ型は浮動小数点型で、計算によって近似値が得られます。MySQL では、[FLOAT](#) または [DOUBLE](#) のシノニムである型は [DOUBLE PRECISION](#) および [REAL](#) です。

12.25.2 DECIMAL データ型の特性

このセクションでは、[DECIMAL](#) データ型 (およびそのシノニム) の特性について説明します。内容は次のとおりです:

- 最大桁数
- ストレージフォーマット
- ストレージ要件
- **DECIMAL** カラムの上限への非標準の MySQL 拡張。

DECIMAL カラムの宣言構文は、**DECIMAL(M,D)** です。引数の値の範囲は次のとおりです:

- **M** は、最大桁数 (精度) です。その範囲は 1 から 65 までです。
- **D** は、小数点の右側の桁数 (スケール) です。その範囲は 0 から 30 までであり、**M** より大きくすることはできません。

D が省略された場合のデフォルトは 0 です。**M** が省略された場合のデフォルトは 10 です。

M が 65 の最大値である場合は、**DECIMAL** 値に対する計算が 65 桁まで正確であることを示します。この 65 桁の精度の制限は厳密値数値リテラルにも適用されるため、このようなりテラルの最大範囲は以前とは異なります。(**DECIMAL** リテラルのテキストの長さには制限もあります。 [セクション 12.25.3 「式の処理」](#) を参照してください。)

DECIMAL カラムの値は、9 桁の小数点を 4 バイトにパックするバイナリ形式を使用して格納されます。各値の整数部と小数部に対するストレージ要件は個別に決定されます。9 桁の繰り返しごとに 4 バイトが必要であり、残りの桁がある場合は、4 バイトのうちの一部が必要になります。残りの桁に必要なストレージは、次の表で指定されます。

余りの桁	バイト数
0	0
1-2	1
3-4	2
5-6	3
7-9	4

たとえば、**DECIMAL(18,9)** カラムは小数点の両側に 9 桁あるため、整数部と小数部のそれぞれに 4 バイトが必要です。**DECIMAL(20,6)** カラムは整数部に 14 桁、小数部に 6 桁あります。整数部の桁は、そのうちの 9 桁に 4 バイト、残りの 5 桁に 3 バイトが必要です。小数部の 6 桁には 3 バイトが必要です。

DECIMAL カラムには、先頭の + 文字、- 文字または先頭の 0 桁は格納されません。**DECIMAL(5,1)** カラムに +0003.1 を挿入すると、それは 3.1 として格納されます。負の数の場合、リテラルの - 文字は格納されません。

DECIMAL カラムでは、カラム定義で暗黙的に指定された範囲を超える値は許可されません。たとえば、**DECIMAL(3,0)** カラムは -999 から 999 までの範囲をサポートします。**DECIMAL(M,D)** カラムでは、**M** まで使用できます - 小数点の左側の **D** 桁。

SQL 標準では、**NUMERIC(M,D)** の精度は正確に **M** 桁である必要があります。**DECIMAL(M,D)** の場合、この標準では少なくとも **M** 桁の精度が必要ですが、それを超える精度が許可されます。MySQL では、**DECIMAL(M,D)** と **NUMERIC(M,D)** は同じであり、どちらも正確に **M** 桁の精度を持っています。

DECIMAL 値の内部形式の完全な説明については、MySQL ソース配布内のファイル [strings/decimal.c](#) を参照してください。この形式は、[decimal2bin\(\)](#) 関数で (例を使用して) 説明されています。

12.25.3 式の処理

高精度計算では、可能な場合は常に、厳密値数値は与えられたとおりに使用されます。たとえば、比較での数値は、値を変更することなく正確に指定されたとおりに使用されます。厳密な SQL モードでは、厳密値データ型 (**DECIMAL** または整数) を持つカラムへの **INSERT** の場合、数値の厳密値がそのカラムの範囲内であれば、その数値が厳密値で挿入されます。取得されると、その値は、挿入された値と同じになるはずですが、(厳密な SQL モードが有効になっていない場合は、**INSERT** での切り捨てが許可されます。)

数値式の処理は、その式にどのような種類の値が含まれているかによって異なります。

- 近似値が存在する場合、その式は近似値であるため、浮動小数点演算を使用して評価されます。

- 近似値が存在しない場合、その式には厳密値のみが含まれています。いずれかの厳密値に小数部(小数点が続く値)が含まれている場合、その式は **DECIMAL** の正確な演算を使用して評価され、精度は 65 桁になります。「正確な」という用語は、バイナリで表すことができる精度の制限に従います。たとえば、 $1.0/3.0$ は 10 進数表記で $.333...$ と近似できますが、厳密値数値として記述されないため、 $(1.0/3.0)*3.0$ は正確には 1.0 に評価されません。
- それ以外の場合、式には整数値のみが含まれています。その式は厳密値であり、整数演算を使用して評価され、精度は **BIGINT** (64 ビット) と同じになります。

数値式に文字列が含まれている場合、その文字列は倍精度浮動小数点値に変換され、その式は近似値になります。

数値カラムへの挿入は、**sql_mode** システム変数によって制御される SQL モードによって影響を受けます。(セクション 5.1.11 「サーバー SQL モード」を参照してください。) 次の説明では、厳密モード (**STRICT_ALL_TABLES** または **STRICT_TRANS_TABLES** モード値によって選択されます) および **ERROR_FOR_DIVISION_BY_ZERO** について触れています。すべての制限を有効にするには、単純に、厳密モード値と **ERROR_FOR_DIVISION_BY_ZERO** の両方を含む **TRADITIONAL** モードを使用できます。

```
SET sql_mode='TRADITIONAL';
```

数値が正確な型のカラム (**DECIMAL** または整数) に挿入された場合、その数値がカラムの範囲および精度内にあれば、正確な値とともに挿入されます。

小数部の桁数が多すぎる場合、端数処理が行われ、ノートが生成されます。丸めは、セクション 12.25.4 「丸め動作」で説明されているとおりに実行されます。小数部の丸めによる切捨ては、厳密モードでもエラーではありません。

整数部の桁数が多すぎる場合、値は大きすぎる (範囲外) ため、次のように処理されます：

- 厳密モードが有効になっていない場合は、その値がもっとも近い正当な値に切り捨てられ、警告が生成されます。
- 厳密モードが有効になっている場合は、オーバーフローエラーが発生します。

DECIMAL リテラルの場合、65 桁の精度制限に加えて、リテラルのテキストの長さに制限があります。値が約 80 文字を超えると、予期しない結果が発生する可能性があります。例：

```
mysql> SELECT
  CAST('0000000000000000000000000000000000000000000000000000000000000000000020.01 AS DECIMAL(15,2)) as val;
+-----+
| val      |
+-----+
| 99999999999999.99 |
+-----+
1 row in set, 2 warnings (0.00 sec)

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message                                         |
+-----+-----+-----+
| Warning | 1292 | Truncated incorrect DECIMAL value: '20'  |
| Warning | 1264 | Out of range value for column 'val' at row 1 |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

アンダーフローは検出されないため、アンダーフローの処理は定義されていません。

数値カラムへの文字列の挿入では、その文字列に非数値の内容が含まれている場合、文字列から数値への変換は次のように処理されます。

- 数値で始まらない文字列を数値として使用することはできないため、この文字列により厳密モードではエラーが生成され、それ以外の場合は警告が生成されます。これには、空の文字列が含まれます。
- 数値で始まる文字列は変換できますが、後続の非数値の部分は切り捨てられます。その切り捨てられた部分にスペース以外の文字が含まれている場合、厳密モードではエラーが生成され、それ以外の場合は警告が生成されません。

デフォルトでは、0 による除算によって **NULL** の結果が生成され、警告は生成されません。SQL モードを適切に設定することにより、0 による除算を制限できます。

`ERROR_FOR_DIVISION_BY_ZERO` SQL モードが有効になっている場合、MySQL は、0 による除算を次の異なった方法で処理します。

- 厳密モードが有効になっていない場合は、警告が発生します。
- 厳密モードが有効になっている場合は、0 による除算を含む挿入と更新が禁止され、エラーが発生します。

つまり、0 による除算を実行する式を含む挿入や更新をエラーとして処理できますが、それには、厳密モードに加えて `ERROR_FOR_DIVISION_BY_ZERO` が必要です。

次のステートメントがあるとします。

```
INSERT INTO t SET i = 1/0;
```

次に、厳密モードと `ERROR_FOR_DIVISION_BY_ZERO` モードの各組み合わせに対する動作を示します。

sql_mode 値	結果
" (デフォルト)	警告なし、エラーなし。i は NULL に設定されます。
厳密	警告なし、エラーなし。i は NULL に設定されます。
<code>ERROR_FOR_DIVISION_BY_ZERO</code>	警告あり、エラーなし。i は NULL に設定されます。
厳密、 <code>ERROR_FOR_DIVISION_BY_ZERO</code>	エラー状態。行は挿入されません。

12.25.4 丸め動作

このセクションでは、`ROUND()` 関数、および厳密値型 (`DECIMAL` と整数) を持つカラムへの挿入での高精度計算の丸めについて説明します。

`ROUND()` 関数は、その引数が厳密値または近似値のどちらであるかに応じて、異なった方法で丸めます。

- 真値の数値の場合、`ROUND()` は「四捨五入」ルールを使用します。0.5 以上の小数部を持つ値は、正の場合は次の整数に切り上げられ、負の場合は次の整数に切り下げられます。(つまり、ゼロから遠い方に丸められます。) 0.5 未満の小数部を持つ値は、正の場合は次の整数に切り下げられ、負の場合は次の整数に切り上げられます。つまり、ゼロに丸められます。)
- 近似値の数字の場合、結果は C ライブラリによって異なります。多くのシステムでは、これは `ROUND()` が「最も近い偶数に丸める」ルールを使用することを意味: 2 つの整数の間に小数部がある値は、最も近い偶数の整数に丸められます。

次の例では、正確な値の丸めと近似値の丸めの相違点を示します。

```
mysql> SELECT ROUND(2.5), ROUND(25E-1);
+-----+-----+
| ROUND(2.5) | ROUND(25E-1) |
+-----+-----+
| 3         | 2         |
+-----+-----+
```

`DECIMAL` または整数カラムへの挿入では、ターゲットが厳密値データ型であるため、挿入される値が厳密値または近似値のどちらであるかには関係なく、丸めでは「四捨五入」が使用されます。

```
mysql> CREATE TABLE t (d DECIMAL(10,0));
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t VALUES(2.5),(2.5E0);
Query OK, 2 rows affected, 2 warnings (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 2

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Note | 1265 | Data truncated for column 'd' at row 1 |
| Note | 1265 | Data truncated for column 'd' at row 2 |
+-----+-----+-----+
```

```
2 rows in set (0.00 sec)

mysql> SELECT d FROM t;
+----+
| d  |
+----+
| 3  |
| 3  |
+----+
2 rows in set (0.00 sec)
```

SHOW WARNINGS ステートメントは、小数部の丸めによって切り捨てられた場合に生成されるノートを表示します。このような切捨ては、厳密な SQL モードでもエラーになりません ([セクション12.25.3「式の処理」](#) を参照)。

12.25.5 高精度計算の例

このセクションでは、MySQL での精度の高い数学的クエリ結果を示す例をいくつか示します。これらの例は、[セクション12.25.3「式の処理」](#) および [セクション12.25.4「丸め動作」](#) で説明されている原則を示しています。

例 1 数値は、可能であれば、指定されたとおりにその厳密値で使用されます。

```
mysql> SELECT (.1 + .2) = .3;
+-----+
| (.1 + .2) = .3 |
+-----+
|          1     |
+-----+
```

浮動小数点値の場合、結果は不正確です。

```
mysql> SELECT (.1E0 + .2E0) = .3E0;
+-----+
| (.1E0 + .2E0) = .3E0 |
+-----+
|          0           |
+-----+
```

厳密値と近似値の処理の違いを確認するための別の方法として、合計値に小さい数値を何回も加える方法があります。ある変数に **.0001** を 1,000 回加える次のストアプロシージャを考えてみます。

```
CREATE PROCEDURE p ()
BEGIN
  DECLARE i INT DEFAULT 0;
  DECLARE d DECIMAL(10,4) DEFAULT 0;
  DECLARE f FLOAT DEFAULT 0;
  WHILE i < 10000 DO
    SET d = d + .0001;
    SET f = f + .0001E0;
    SET i = i + 1;
  END WHILE;
  SELECT d, f;
END;
```

論理的には、**d** と **f** の両方の合計値が 1 になるはずですが、それは 10 進数の計算にしか当てはまりません。浮動小数点の計算では、小さなエラーが発生します。

```
+-----+-----+
| d  | f          |
+-----+-----+
| 1.0000 | 0.999999999999991 |
+-----+-----+
```

例 2 乗算は、標準 SQL に必要なスケールで実行されます。つまり、スケール **S1** と **S2** を持つ 2 つの数値 **X1** と **X2** の場合、結果のスケールは **S1 + S2** になります。

```
mysql> SELECT .01 * .01;
+-----+
|.01 * .01 |
+-----+
| 0.0001  |
+-----+
```



```
+-----+
```

例 3 厳密値数値に対する丸め動作は、適切に定義されています。

丸め動作(たとえば、`ROUND()` 関数を使用した動作)は、ベースとなる C ライブラリの実装には依存しません。つまり、その結果は各プラットフォームにわたって一貫性があります。

- 厳密値カラム (`DECIMAL` と整数) および厳密値数値に対する丸めでは、「四捨五入」ルールが使用されます。小数部が .5 以上の値は、次に示すようにゼロから最も近い整数に丸められます:

```
mysql> SELECT ROUND(2.5), ROUND(-2.5);
+-----+
| ROUND(2.5) | ROUND(-2.5) |
+-----+
| 3         | -3         |
+-----+
```

- 浮動小数点値に対する丸めでは、C ライブラリが使用されます。これは、多くのシステムでは「偶数丸め」ルールを使用します。2 つの整数の間に小数部が完全に半分の値がある場合、最も近い偶数に丸められます:

```
mysql> SELECT ROUND(2.5E0), ROUND(-2.5E0);
+-----+
| ROUND(2.5E0) | ROUND(-2.5E0) |
+-----+
| 2           | -2           |
+-----+
```

例 4 厳密モードでは、カラムの範囲を外れている値を挿入すると、正当な値への切り捨てではなく、エラーが発生します。

MySQL が厳密モードで実行されていない場合は、正当な値への切り捨てが発生します。

```
mysql> SET sql_mode="";
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t SET i = 128;
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> SELECT i FROM t;
+-----+
| i |
+-----+
| 127 |
+-----+
1 row in set (0.00 sec)
```

ただし、厳密モードが有効になっている場合は、エラーが発生します。

```
mysql> SET sql_mode='STRICT_ALL_TABLES';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t SET i = 128;
ERROR 1264 (22003): Out of range value adjusted for column 'i' at row 1

mysql> SELECT i FROM t;
Empty set (0.00 sec)
```

例 5: 厳密モードで `ERROR_FOR_DIVISION_BY_ZERO` が設定されている場合は、0 による除算によって `NULL` の結果ではなく、エラーが発生します。

非厳密モードでは、0 による除算によって `NULL` の結果が生成されます。

```
mysql> SET sql_mode="";
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t SET i = 1 / 0;
Query OK, 1 row affected (0.00 sec)

mysql> SELECT i FROM t;
+-----+
| i |
+-----+
| NULL |
+-----+
1 row in set (0.03 sec)
```

ただし、適切な SQL モードが有効になっている場合、0 による除算はエラーになります。

```
mysql> SET sql_mode='STRICT_ALL_TABLES,ERROR_FOR_DIVISION_BY_ZERO';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t SET i = 1 / 0;
ERROR 1365 (22012): Division by 0

mysql> SELECT i FROM t;
Empty set (0.01 sec)
```

例 6 厳密値リテラルは、厳密値として評価されます。

近似値リテラルは浮動小数点を使用して評価されますが、正確な値リテラルは **DECIMAL** として処理されます：

```
mysql> CREATE TABLE t SELECT 2.5 AS a, 25E-1 AS b;
Query OK, 1 row affected (0.01 sec)
Records: 1 Duplicates: 0 Warnings: 0

mysql> DESCRIBE t;
+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| a     | decimal(2,1) unsigned | NO   |    | 0.0     |      |
| b     | double        | NO   |    | 0       |      |
+-----+-----+-----+-----+-----+
2 rows in set (0.01 sec)
```

例 7 集約関数への引数が厳密値数値型である場合は、その結果も、少なくとも引数と同じスケールを持つ厳密値数値型になります。

これらのステートメントを考慮してください。

```
mysql> CREATE TABLE t (i INT, d DECIMAL, f FLOAT);
mysql> INSERT INTO t VALUES(1,1,1);
mysql> CREATE TABLE y SELECT AVG(i), AVG(d), AVG(f) FROM t;
```

結果が倍精度値になるのは浮動小数点引数の場合だけです。厳密値型引数の場合は、結果も厳密値型になります。

```
mysql> DESCRIBE y;
+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| AVG(i) | decimal(14,4) | YES  |    | NULL    |      |
| AVG(d) | decimal(14,4) | YES  |    | NULL    |      |
| AVG(f) | double        | YES  |    | NULL    |      |
+-----+-----+-----+-----+-----+
```

結果が倍精度値になるのは浮動小数点引数の場合だけです。厳密値型引数の場合は、結果も厳密値型になります。

第 13 章 SQL ステートメント

目次

13.1 データ定義ステートメント	2150
13.1.1 アトミックデータ定義ステートメントのサポート	2150
13.1.2 ALTER DATABASE ステートメント	2156
13.1.3 ALTER EVENT ステートメント	2160
13.1.4 ALTER FUNCTION ステートメント	2162
13.1.5 ALTER INSTANCE ステートメント	2162
13.1.6 ALTER LOGFILE GROUP ステートメント	2163
13.1.7 ALTER PROCEDURE ステートメント	2165
13.1.8 ALTER SERVER ステートメント	2165
13.1.9 ALTER TABLE ステートメント	2165
13.1.10 ALTER TABLESPACE ステートメント	2187
13.1.11 ALTER VIEW ステートメント	2188
13.1.12 CREATE DATABASE ステートメント	2189
13.1.13 CREATE EVENT ステートメント	2189
13.1.14 CREATE FUNCTION ステートメント	2194
13.1.15 CREATE INDEX ステートメント	2194
13.1.16 CREATE LOGFILE GROUP ステートメント	2207
13.1.17 CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント	2209
13.1.18 CREATE SERVER ステートメント	2213
13.1.19 CREATE SPATIAL REFERENCE SYSTEM ステートメント	2214
13.1.20 CREATE TABLE ステートメント	2218
13.1.21 CREATE TABLESPACE ステートメント	2268
13.1.22 CREATE TRIGGER ステートメント	2275
13.1.23 CREATE VIEW ステートメント	2277
13.1.24 DROP DATABASE ステートメント	2281
13.1.25 DROP EVENT ステートメント	2282
13.1.26 DROP FUNCTION ステートメント	2282
13.1.27 DROP INDEX ステートメント	2282
13.1.28 DROP LOGFILE GROUP ステートメント	2282
13.1.29 DROP PROCEDURE および DROP FUNCTION ステートメント	2283
13.1.30 DROP SERVER ステートメント	2283
13.1.31 DROP SPATIAL REFERENCE SYSTEM ステートメント	2283
13.1.32 DROP TABLE ステートメント	2284
13.1.33 DROP TABLESPACE ステートメント	2285
13.1.34 DROP TRIGGER ステートメント	2286
13.1.35 DROP VIEW ステートメント	2286
13.1.36 RENAME TABLE ステートメント	2287
13.1.37 TRUNCATE TABLE ステートメント	2288
13.2 データ操作ステートメント	2290
13.2.1 CALL ステートメント	2290
13.2.2 DELETE ステートメント	2291
13.2.3 DO ステートメント	2295
13.2.4 HANDLER ステートメント	2295
13.2.5 IMPORT TABLE ステートメント	2297
13.2.6 INSERT ステートメント	2299
13.2.7 LOAD DATA ステートメント	2307
13.2.8 LOAD XML ステートメント	2316
13.2.9 REPLACE ステートメント	2323
13.2.10 SELECT ステートメント	2325
13.2.11 サブクエリー	2344
13.2.12 TABLE ステートメント	2358
13.2.13 UPDATE ステートメント	2360
13.2.14 VALUES ステートメント	2364

13.2.15 WITH (共通テーブル式)	2366
13.3 トランザクションステートメントおよびロックステートメント	2376
13.3.1 START TRANSACTION、COMMIT および ROLLBACK ステートメント	2376
13.3.2 ロールバックできないステートメント	2379
13.3.3 暗黙的なコミットを発生させるステートメント	2379
13.3.4 SAVEPOINT、ROLLBACK TO SAVEPOINT および RELEASE SAVEPOINT ステートメント	2380
13.3.5 LOCK INSTANCE FOR BACKUP および UNLOCK INSTANCE ステートメント	2381
13.3.6 LOCK TABLES および UNLOCK TABLES ステートメント	2382
13.3.7 SET TRANSACTION ステートメント	2387
13.3.8 XA トランザクション	2390
13.4 レプリケーションステートメント	2395
13.4.1 ソースサーバーを制御する SQL ステートメント	2395
13.4.2 レプリケーションサーバーを制御するための SQL ステートメント	2398
13.4.3 グループレプリケーションを制御するための SQL ステートメント	2433
13.5 プリペアドステートメント	2437
13.5.1 PREPARE ステートメント	2441
13.5.2 EXECUTE ステートメント	2443
13.5.3 DEALLOCATE PREPARE ステートメント	2443
13.6 複合ステートメントの構文	2443
13.6.1 BEGIN ... END 複合ステートメント	2443
13.6.2 ステートメントラベル	2444
13.6.3 DECLARE ステートメント	2444
13.6.4 ストアドプログラム内の変数	2445
13.6.5 フロー制御ステートメント	2446
13.6.6 カーソル	2450
13.6.7 条件の処理	2452
13.6.8 条件処理の制約	2476
13.7 データベース管理ステートメント	2477
13.7.1 アカウント管理ステートメント	2477
13.7.2 リソースグループ管理ステートメント	2520
13.7.3 テーブル保守ステートメント	2523
13.7.4 コンポーネント、プラグインおよびユーザー定義関数のステートメント	2536
13.7.5 CLONE ステートメント	2540
13.7.6 SET ステートメント	2541
13.7.7 SHOW ステートメント	2547
13.7.8 その他の管理ステートメント	2598
13.8 ユーティリティステートメント	2610
13.8.1 DESCRIBE ステートメント	2610
13.8.2 EXPLAIN ステートメント	2611
13.8.3 HELP ステートメント	2614
13.8.4 USE ステートメント	2615

この章では、MySQL によってサポートされる [SQL](#) ステートメントの構文について説明します。

13.1 データ定義ステートメント

13.1.1 アトミックデータ定義ステートメントのサポート

MySQL 8.0 では、アトミックデータ定義言語 (DDL) ステートメントがサポートされます。この機能は、アトミック DDL と呼ばれます。アトミック DDL ステートメントは、DDL 操作に関連するデータディクショナリ更新、ストレージエンジン操作およびバイナリログ書き込みを単一のアトミックトランザクションとして結び付けます。操作は、データディクショナリ、ストレージエンジンおよびバイナリログに適用可能な変更を保持してコミットされるか、操作中にサーバーが停止した場合でもロールバックされます。

注記

アトミック DDL はトランザクション DDL ではありません。DDL ステートメント、アトミックまたはそれ以外の場合は、ステートメントを実行する前に [COMMIT](#) を実行したかのように、現在のセッションでアクティブなトランザクションを暗黙的に終了します。つまり、DDL ステートメントは、別のトランザクション内、[START TRANSACTION ...](#)

COMMIT などのトランザクション制御ステートメント内、または同じトランザクション内の他のステートメントと組み合わせることはできません。

アトミック DDL は、MySQL 8.0 で MySQL データディクショナリを導入することで可能になります。以前の MySQL バージョンでは、メタデータはメタデータファイル、非トランザクションテーブル、および中間コミットを必要とするストレージエンジン固有のディクショナリに格納されていました。MySQL データディクショナリによって提供される集中化されたトランザクションメタデータ記憶域により、このバリアが削除され、DDL ステートメントの操作をアトミックに再構築できるようになりました。

アトミック DDL 機能については、このセクションの次のトピックで説明します:

- サポートされている DDL ステートメント
- アトミック DDL の特性
- DDL ステートメントの動作の変更
- ストレージエンジンのサポート
- DDL ログの表示

サポートされている DDL ステートメント

アトミック DDL 機能では、テーブル DDL ステートメントと非テーブル DDL ステートメントの両方がサポートされています。テーブル関連の DDL 操作ではストレージエンジンのサポートが必要ですが、テーブル以外の DDL 操作では必要ありません。現在、**InnoDB** ストレージエンジンのみがアトミック DDL をサポートしています。

- サポートされているテーブル DDL ステートメントには、データベース、テーブルスペース、テーブルおよびインデックスに対する **CREATE**、**ALTER** および **DROP** ステートメントと、**TRUNCATE TABLE** ステートメントが含まれます。
- サポートされているテーブル以外の DDL ステートメントは次のとおりです:
 - **CREATE** ステートメントと **DROP** ステートメント、および該当する場合はストアドプログラム、トリガー、ビューおよびユーザー定義関数 (UDF) の **ALTER** ステートメント。
 - アカウント管理ステートメント: **CREATE**、**ALTER**、**DROP**、および該当する場合は、ユーザーおよびロール用の **RENAME** ステートメントと、**GRANT** および **REVOKE** ステートメント。

次のステートメントは、アトミック DDL 機能ではサポートされていません:

- **InnoDB** 以外のストレージエンジンを含むテーブル関連の DDL ステートメント。
- **INSTALL PLUGIN** および **UNINSTALL PLUGIN** ステートメント。
- **INSTALL COMPONENT** および **UNINSTALL COMPONENT** ステートメント。
- **CREATE SERVER**、**ALTER SERVER** および **DROP SERVER** ステートメント。

アトミック DDL の特性

アトミック DDL ステートメントの特性は次のとおりです:

- メタデータの更新、バイナリログの書き込み、およびストレージエンジンの操作 (該当する場合は)、単一の原子性操作に結合されます。
- DDL 操作中、SQL レイヤーに中間コミットはありません。
- 該当する場合:
 - データディクショナリ、ルーチン、イベントおよび UDF キャッシュの状態は、DDL 操作のステータスと一貫性があります。つまり、DDL 操作が正常に完了したかロールバックされたかを反映するようにキャッシュが更新されます。
 - DDL 操作に関連するストレージエンジンのメソッドは中間コミットを実行せず、ストレージエンジンはそれ自体を DDL 操作の一部として登録します。

- ストレージエンジンは、DDL 操作の Post-DDL フェーズで実行される DDL 操作の redo およびロールバックをサポートしています。
- DDL 操作の表示可能な動作はアトミックで、一部の DDL ステートメントの動作が変更されます。DDL ステートメントの動作の変更を参照してください。

DDL ステートメントの動作の変更

このセクションでは、アトミック DDL サポートの導入による DDL ステートメントの動作の変更について説明します。

- すべての名前付きテーブルがアトミック DDL でサポートされているストレージエンジンを使用している場合、[DROP TABLE](#) 操作は完全にアトミックです。このステートメントは、すべてのテーブルを正常に削除するか、ロールバックします。

指定されたテーブルが存在せず、ストレージエンジンに関係なく変更が行われない場合、[DROP TABLE](#) はエラーで失敗します。次の例では、指定したテーブルが存在しないために [DROP TABLE](#) ステートメントが失敗する動作の変更を示します：

```
mysql> CREATE TABLE t1 (c1 INT);
mysql> DROP TABLE t1, t2;
ERROR 1051 (42S02): Unknown table 'test.t2'
mysql> SHOW TABLES;
+-----+
| Tables_in_test |
+-----+
| t1              |
+-----+
```

アトミック DDL が導入される前に、[DROP TABLE](#) は、存在しないが、存在する名前付きテーブルに対して成功した名前付きテーブルのエラーを報告します：

```
mysql> CREATE TABLE t1 (c1 INT);
mysql> DROP TABLE t1, t2;
ERROR 1051 (42S02): Unknown table 'test.t2'
mysql> SHOW TABLES;
Empty set (0.00 sec)
```

注記

この動作の変更のため、MySQL 8.0 レプリカにレプリケートすると、MySQL 5.7 レプリケーションソースサーバーで部分的に完了した [DROP TABLE](#) ステートメントが失敗します。この失敗のシナリオを回避するには、[DROP TABLE](#) ステートメントで [IF EXISTS](#) 構文を使用して、存在しないテーブルに対してエラーが発生しないようにします。

- すべてのテーブルがアトミック DDL でサポートされているストレージエンジンを使用する場合、[DROP DATABASE](#) はアトミックです。このステートメントは、すべてのオブジェクトを正常に削除するか、ロールバックします。ただし、ファイルシステムからのデータベースディレクトリの削除は最後に行われ、アトミック操作の一部ではありません。ファイルシステムエラーまたはサーバーの停止が原因でデータベースディレクトリの削除に失敗した場合、[DROP DATABASE](#) トランザクションはロールバックされません。
- アトミック DDL でサポートされているストレージエンジンを使用しないテーブルの場合、テーブルの削除はアトミック [DROP TABLE](#) または [DROP DATABASE](#) トランザクションの外部で行われます。このようなテーブルの削除はバイナリログに個別に書き込まれるため、[DROP TABLE](#) または [DROP DATABASE](#) 操作が中断された場合は、ストレージエンジン、データディクショナリ、およびバイナリログ間の相違が最大で 1 つのテーブルに制限されます。複数のテーブルを削除する操作の場合、アトミック DDL でサポートされているストレージエンジンを使用しないテーブルは、それを実行するテーブルの前に削除されます。
- アトミック DDL でサポートされているストレージエンジンを使用するテーブルに対する [CREATE TABLE](#)、[ALTER TABLE](#)、[RENAME TABLE](#)、[TRUNCATE TABLE](#)、[CREATE TABLESPACE](#)、および [DROP TABLESPACE](#) 操作は、その操作中にサーバーが停止すると、完全にコミットまたはロールバックされます。以前の MySQL リリースでは、これらの操作が中断されると、ストレージエンジン、データディクショナリ、およびバイナリログ間の不一致が発生したり、孤立したファイルの背後に残されたりする可能性がありました。[RENAME TABLE](#) 操作は、すべての名

前付きテーブルがアトミック DDL でサポートされているストレージエンジンを使用している場合にのみアトミックです。

- MySQL 8.0.21 の時点では、アトミック DDL をサポートするストレージエンジンでは、行ベースレプリケーションが使用されているときに、`CREATE TABLE ... SELECT` ステートメントがバイナリログに 1 つのトランザクションとして記録されます。以前は、2 つのトランザクションとしてログに記録されていました。1 つはテーブルの作成用、もう 1 つはデータの挿入用です。2 つのトランザクション間またはデータの挿入中にサーバー障害が発生すると、空のテーブルがレプリケーションされる可能性があります。アトミック DDL サポートの導入により、`CREATE TABLE ... SELECT` ステートメントは行ベースレプリケーションに対して安全であり、GTID ベースレプリケーションでの使用が許可されるようになりました。

アトミック DDL 制約と外部キー制約の両方をサポートするストレージエンジンでは、行ベースレプリケーションが使用されている場合、`CREATE TABLE ... SELECT` ステートメントで外部キーの作成は許可されません。外部キー制約は、後で `ALTER TABLE` を使用して追加できます。

`CREATE TABLE ... SELECT` がアトミック操作として適用されると、データの挿入中にメタデータロックがテーブルに保持され、操作中にテーブルへの同時アクセスが防止されます。

- 名前付きビューが存在せず、変更が行われない場合、`DROP VIEW` は失敗します。この例では、名前付きビューが存在しないために `DROP VIEW` ステートメントが失敗する動作の変更を示します:

```
mysql> CREATE VIEW test.viewA AS SELECT * FROM t;
mysql> DROP VIEW test.viewA, test.viewB;
ERROR 1051 (42S02): Unknown table 'test.viewB'
mysql> SHOW FULL TABLES IN test WHERE TABLE_TYPE LIKE 'VIEW';
+-----+-----+
| Tables_in_test | Table_type |
+-----+-----+
| viewA          | VIEW      |
+-----+-----+
```

アトミック DDL が導入される前に、`DROP VIEW` は、存在しないが、存在する名前付きビューに対して成功した名前付きビューに対してエラーを返します:

```
mysql> CREATE VIEW test.viewA AS SELECT * FROM t;
mysql> DROP VIEW test.viewA, test.viewB;
ERROR 1051 (42S02): Unknown table 'test.viewB'
mysql> SHOW FULL TABLES IN test WHERE TABLE_TYPE LIKE 'VIEW';
Empty set (0.00 sec)
```

注記

この動作の変更のため、MySQL 8.0 レプリカでレプリケートすると、MySQL 5.7 レプリケーションソースサーバーで部分的に完了した `DROP VIEW` 操作が失敗します。この失敗のシナリオを回避するには、`DROP VIEW` ステートメントで `IF EXISTS` 構文を使用して、存在しないビューに対してエラーが発生しないようにします。

- アカウント管理ステートメントの部分実行は許可されなくなりました。アカウント管理ステートメントは、指定されたすべてのユーザーに対して成功するか、ロールバックされ、エラーが発生しても効果はありません。以前の MySQL バージョンでは、複数のユーザーを指定するアカウント管理ステートメントは、一部のユーザーでは成功し、他のユーザーでは失敗する可能性がありました。

この例では、動作の変更が示されています。この例では、2 番目の `CREATE USER` ステートメントはエラーを返しますが、すべての名前付きユーザーが成功するわけではないため、失敗します。

```
mysql> CREATE USER userA;
mysql> CREATE USER userA, userB;
ERROR 1396 (HY000): Operation CREATE USER failed for 'userA'@%'
mysql> SELECT User FROM mysql.user WHERE User LIKE 'user%';
+-----+
| User |
+-----+
| userA |
+-----+
```

アトミック DDL が導入される前に、2 番目の `CREATE USER` ステートメントは、存在しないが、存在する名前付きユーザーに対して成功した名前付きユーザーに対してエラーを返します:

```
mysql> CREATE USER userA;
mysql> CREATE USER userA, userB;
ERROR 1396 (HY000): Operation CREATE USER failed for 'userA'@'%
mysql> SELECT User FROM mysql.user WHERE User LIKE 'user%';
+-----+
| User |
+-----+
| userA |
| userB |
+-----+
```

注記

この動作の変更のため、MySQL 8.0 レプリカでレプリケートすると、MySQL 5.7 レプリケーションソースサーバーで部分的に完了したアカウント管理ステートメントが失敗します。この失敗のシナリオを回避するには、アカウント管理ステートメントで必要に応じて `IF EXISTS` または `IF NOT EXISTS` 構文を使用して、名前付きユーザーに関連するエラーを防止します。

ストレージエンジンのサポート

現在、[InnoDB](#) ストレージエンジンのみがアトミック DDL をサポートしています。アトミック DDL をサポートしないストレージエンジンは、DDL アトミック性から除外されます。除外されたストレージエンジンに関連する DDL 操作は、操作が中断されたとき、または部分的にしか完了しなかったときに発生する可能性のある不整合を引き起こすことができます。

DDL 操作の redo およびロールバックをサポートするために、[InnoDB](#) は、`mysql.ibd` データディクショナリテーブルスペースに存在する非表示のデータディクショナリテーブルである `mysql.innodb_ddl_log` テーブルに DDL ログを書き込みます。

DDL 操作中に `mysql.innodb_ddl_log` テーブルに書き込まれる DDL ログを表示するには、`innodb_print_ddl_logs` 構成オプションを有効にします。詳細は、[DDL ログの表示](#) を参照してください。

注記

`mysql.innodb_ddl_log` テーブルに対する変更の redo ログは、`innodb_flush_log_at_trx_commit` の設定に関係なく、すぐにディスクにフラッシュされます。redo ログをすぐにフラッシュすると、DDL 操作によってデータファイルが変更される状況を回避できますが、これらの操作によって生成された `mysql.innodb_ddl_log` テーブルに対する変更の redo ログはディスクに永続化されません。このような状況では、ロールバックまたはリカバリ中にエラーが発生する可能性があります。

[InnoDB](#) ストレージエンジンは、DDL 操作をフェーズで実行します。`ALTER TABLE` などの DDL 操作では、Commit フェーズの前に準備および Perform フェーズを複数回実行できます。

1. 準備: 必要なオブジェクトを作成し、DDL ログを `mysql.innodb_ddl_log` テーブルに書き込みます。DDL ログは、DDL 操作をロールフォワードおよびロールバックする方法を定義します。
2. Perform: DDL 操作を実行します。たとえば、`CREATE TABLE` 操作の作成ルーチンを実行します。
3. Commit: データディクショナリを更新し、データディクショナリのトランザクションをコミットします。
4. Post-DDL: DDL ログをリプレイし、`mysql.innodb_ddl_log` テーブルから削除します。不整合を引き起こさずにロールバックを安全に実行できるように、データファイルの名前変更や削除などのファイル操作はこの最終フェーズで実行されます。このフェーズでは、`DROP TABLE`、`TRUNCATE TABLE` およびテーブルを再構築するその他の DDL 操作のために、`mysql.innodb_dynamic_metadata` データディクショナリテーブルから動的メタデータも削除します。

DDL 操作がコミットされているかロールバックされているかに関係なく、Post-DDL フェーズ中に DDL ログがリプレイされ、`mysql.innodb_ddl_log` テーブルから削除されます。DDL ログは、DDL 操作中にサーバーが停止した場合にのみ `mysql.innodb_ddl_log` テーブルに残す必要があります。この場合、DDL ログはリカバリ後にリプレイおよび削除されます。

リカバリ状況では、サーバーの再起動時に DDL 操作をコミットまたはロールバックできます。DDL 操作の Commit フェーズで実行されたデータディクショナリトランザクションが redo ログおよびバイナリログに存在する場合、操作は成功したとみなされ、ロールフォワードされます。それ以外の場合、InnoDB がデータディクショナリ redo ログをリプレイし、DDL 操作がロールバックされると、不完全なデータディクショナリトランザクションがロールバックされます。

DDL ログの表示

InnoDB ストレージエンジンに関連するアトミック DDL 操作中に `mysql.innodb_ddl_log` データディクショナリテーブルに書き込まれる DDL ログを表示するには、`innodb_print_ddl_logs` で MySQL に `stderr` への DDL ログの書き込みを許可します。ホストのオペレーティングシステムおよび MySQL の構成によっては、`stderr` がエラーログ、端末またはコンソールウィンドウである場合があります。[セクション5.4.2.2「デフォルトのエラーログ保存先の構成」](#)を参照してください。

InnoDB は、DDL 操作の redo およびロールバックをサポートするために、DDL ログを `mysql.innodb_ddl_log` テーブルに書き込みます。`mysql.innodb_ddl_log` テーブルは、`mysql.ibd` データディクショナリテーブルスペースに存在する非表示のデータディクショナリテーブルです。他の非表示のデータディクショナリテーブルと同様に、MySQL の非デバッグバージョンでは `mysql.innodb_ddl_log` テーブルに直接アクセスできません。(セクション14.1「データディクショナリスキーマ」を参照してください。) `mysql.innodb_ddl_log` テーブルの構造は、次の定義に対応します：

```
CREATE TABLE mysql.innodb_ddl_log (  
  id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,  
  thread_id BIGINT UNSIGNED NOT NULL,  
  type INT UNSIGNED NOT NULL,  
  space_id INT UNSIGNED,  
  page_no INT UNSIGNED,  
  index_id BIGINT UNSIGNED,  
  table_id BIGINT UNSIGNED,  
  old_file_path VARCHAR(512) COLLATE UTF8_BIN,  
  new_file_path VARCHAR(512) COLLATE UTF8_BIN,  
  KEY(thread_id)  
);
```

- `id`: DDL ログレコードの一意的識別子。
- `thread_id`: 各 DDL ログレコードには、特定の DDL 操作に属する DDL ログのリプレイおよび削除に使用される `thread_id` が割り当てられます。複数のデータファイル操作を含む DDL 操作では、複数の DDL ログレコードが生成されます。
- `type`: DDL 操作タイプ。タイプには、[FREE](#) (インデックスツリーの削除)、[DELETE](#) (ファイルの削除)、[RENAME](#) (ファイルの名前変更) または [DROP](#) (`mysql.innodb_dynamic_metadata` データディクショナリテーブルからのメタデータの削除) があります。
- `space_id`: テーブルスペース ID。
- `page_no`: 割当て情報を含むページ。たとえば、インデックスツリーのルートページです。
- `index_id`: インデックス ID。
- `table_id`: テーブル ID。
- `old_file_path`: 古いテーブルスペースのファイルパス。テーブルスペースファイルを作成または削除する DDL 操作で使用されます。テーブルスペースの名前を変更する DDL 操作でも使用されます。
- `new_file_path`: 新しいテーブルスペースファイルのパス。テーブルスペースファイルの名前を変更する DDL 操作で使用されます。

この例では、`CREATE TABLE` 操作のために `stderr` に書き込まれた DDL ログを `innodb_print_ddl_logs` で表示できるようにする方法を示します。

```
mysql> SET GLOBAL innodb_print_ddl_logs=1;  
mysql> CREATE TABLE t1 (c1 INT) ENGINE = InnoDB;
```

```
[Note] [000000] InnoDB: DDL log insert : [DDL record: DELETE SPACE, id=18, thread_id=7,  
space_id=5, old_file_path= ./test/t1.ibd]  
[Note] [000000] InnoDB: DDL log delete : by id 18  
[Note] [000000] InnoDB: DDL log insert : [DDL record: REMOVE CACHE, id=19, thread_id=7,  
table_id=1058, new_file_path=test/t1]
```

```
[Note] [000000] InnoDB: DDL log delete : by id 19
[Note] [000000] InnoDB: DDL log insert : [DDL record: FREE, id=20, thread_id=7,
space_id=5, index_id=132, page_no=4]
[Note] [000000] InnoDB: DDL log delete : by id 20
[Note] [000000] InnoDB: DDL log post ddl : begin for thread id : 7
[Note] [000000] InnoDB: DDL log post ddl : end for thread id : 7
```

13.1.2 ALTER DATABASE ステートメント

```
ALTER {DATABASE | SCHEMA} [db_name]
  alter_option ...

alter_option: {
  [DEFAULT] CHARACTER SET [=] charset_name
  | [DEFAULT] COLLATE [=] collation_name
  | [DEFAULT] ENCRYPTION [=] {'Y' | 'N'}
  | READ ONLY [=] {DEFAULT | 0 | 1}
}
```

ALTER DATABASE を使用すると、データベースの全体的な特性を変更できます。これらの特性はデータディクショナリに格納されます。このステートメントには、データベースに対する **ALTER** 権限が必要です。**ALTER SCHEMA** は **ALTER DATABASE** のシノニムです。

データベース名を省略すると、ステートメントはデフォルトのデータベースに適用されます。その場合、デフォルトのデータベースがないとエラーが発生します。

ステートメントから省略された `alter_option` の場合、データベースでは現在のオプション値が保持されますが、文字セットを変更すると照合順序が変更される場合と逆があります。

- [文字セットと照合順序のオプション](#)
- [暗号化オプション](#)
- [読取り専用オプション](#)

文字セットと照合順序のオプション

CHARACTER SET オプションは、デフォルトのデータベース文字セットを変更します。**COLLATE** オプションは、デフォルトのデータベース照合順序を変更します。文字セットおよび照合順序名の詳細は、[第10章「文字セット、照合順序、Unicode」](#) を参照してください。

使用可能な文字セットおよび照合順序を確認するには、それぞれ **SHOW CHARACTER SET** ステートメントおよび **SHOW COLLATION** ステートメントを使用します。[セクション13.7.7.3「SHOW CHARACTER SET ステートメント」](#) および [セクション13.7.7.4「SHOW COLLATION ステートメント」](#) を参照してください。

ルーチンの作成時にデータベースのデフォルトを使用するストアードルーチンには、それらのデフォルトがその定義の一部として含まれます。(ストアードルーチンでは、文字セットまたは照合順序が明示的に指定されていない場合、文字データ型を伴う変数は、データベースのデフォルトを使用します。[セクション13.1.17「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」](#) を参照してください。) データベースのデフォルトの文字セットまたは照合順序を変更する場合は、新しいデフォルトを使用するストアードルーチンを削除して再作成する必要があります。

暗号化オプション

MySQL 8.0.16 で導入された **ENCRYPTION** オプションは、データベースで作成されたテーブルによって継承されるデフォルトのデータベース暗号化を定義します。許可される値は、**'Y'** (暗号化有効) および **'N'** (暗号化無効) です。新しく作成されたテーブルのみがデフォルトのデータベース暗号化を継承します。データベースに関連付けられている既存のテーブルの場合、暗号化は変更されません。`table_encryption_privilege_check` システム変数がある有効になっている場合、`default_table_encryption` システム変数の値とは異なるデフォルトの暗号化設定を指定するには、**TABLE_ENCRYPTION_ADMIN** 権限が必要です。詳細は、[スキーマおよび一般テーブルスペースの暗号化デフォルトの定義](#) を参照してください。

読取り専用オプション

MySQL 8.0.22 で導入された **READ ONLY** オプションは、データベースおよびデータベース内のオブジェクトの変更を許可するかどうかを制御します。許可される値は、**DEFAULT** または **0** (読取り専用ではない) および **1** (読取り専

用) です。 **READ ONLY** が有効になっているデータベースは、操作中にデータベースが変更される可能性があることに関係なく、別の MySQL インスタンスに移行できるため、このオプションはデータベースの移行に役立ちます。

NDB Cluster では、ある `mysqld` サーバー上のデータベースを読み取り専用にすると、同じクラスタ内のほかの `mysqld` サーバーと同期されるため、データベースはすべての `mysqld` サーバー上で読み取り専用になります。

READ ONLY オプションが有効になっている場合は、`INFORMATION_SCHEMA SCHEMATA_EXTENSIONS` テーブルに表示されます。 [セクション26.32「INFORMATION_SCHEMA SCHEMATA_EXTENSIONS テーブル」](#) を参照してください。

これらのシステムスキーマでは、**READ ONLY** オプションを有効にできません: `mysql`, `information_schema`, `performance_schema`。

ALTER DATABASE ステートメントでは、**READ ONLY** オプションは、次のようにそれ自体の他のインスタンスおよび他のオプションと相互作用します:

- **READ ONLY** の複数のインスタンス (`READ ONLY = 1 READ ONLY = 0` など) が競合する場合は、エラーが発生します。
- 読み取り専用データベースの場合でも、**READ ONLY** オプションのみ (競合しない) を含む **ALTER DATABASE** ステートメントは許可されます。
- ステートメントの前後のデータベースの読み取り専用状態で変更が許可されている場合は、**READ ONLY** オプションと他のオプションを混在させることができます。読み取り専用状態が変更前と変更後の両方で禁止されている場合、エラーが発生します。

このステートメントは、データベースが読み取り専用かどうかにかかわらず成功します:

```
ALTER DATABASE mydb READ ONLY = 0 DEFAULT COLLATE utf8mb4_bin;
```

このステートメントは、データベースが読み取り専用でない場合は成功しますが、すでに読み取り専用の場合は失敗します:

```
ALTER DATABASE mydb READ ONLY = 1 DEFAULT COLLATE utf8mb4_bin;
```

READ ONLY を有効にすると、データベースのすべてのユーザーに影響しますが、読み取り専用チェックの対象ではない次の例外があります:

- サーバーの初期化、再起動、アップグレード、またはレプリケーションの一環としてサーバーによって実行されるステートメント。
- サーバー起動時に `init_file` システム変数によって指定されたファイル内のステートメント。
- **TEMPORARY** テーブル。読み取り専用データベースで **TEMPORARY** テーブルを作成、変更、削除および書き込みできます。
- NDB Cluster の SQL 以外の挿入および更新。

前述の例外操作を除き、**READ ONLY** を有効にすると、データベースおよびそのオブジェクト (定義、データ、メタデータなど) への書き込み操作が禁止されます。次のリストに、影響を受ける SQL ステートメントおよび操作の詳細を示します:

- データベース自体:
 - **CREATE DATABASE**
 - **ALTER DATABASE** (**READ ONLY** オプションの変更を除く)
 - **DROP DATABASE**
- ビュー:
 - **CREATE VIEW**
 - **ALTER VIEW**

- [DROP VIEW](#)
- 副作用のある関数を起動するビューから選択します。
- 更新可能なビューの更新。
- 書き込み可能データベース内のオブジェクトを作成または削除するステートメントは、読取り専用データベース内のビューのメタデータに影響を与える場合 (たとえば、ビューを有効または無効にする場合)、拒否されます。
- ストアドルーチン:
 - [CREATE PROCEDURE](#)
 - [DROP PROCEDURE](#)
 - [CALL](#) (副作用のあるプロシージャ)
 - [CREATE FUNCTION](#)
 - [DROP FUNCTION](#)
 - [SELECT](#) (副作用を持つ関数)
 - プロシージャおよび関数の場合、読取り専用チェックは事前ロックの動作に従います。 [CALL](#) ステートメントの場合、読取り専用チェックはステートメントごとに実行されるため、読取り専用データベースへの書き込みを条件付きで実行したステートメントが実際には実行されない場合でも、コールは成功します。一方、[SELECT](#) 内でコールされる関数の場合、関数本体の実行は事前ロックモードで行われます。関数内の一部のステートメントが読取り専用データベースに書き込むかぎり、そのステートメントが実際に実行されるかどうかに関係なく、関数の実行はエラーで失敗します。
- トリガー:
 - [CREATE TRIGGER](#)
 - [DROP TRIGGER](#)
 - トリガーの起動。
- イベント:
 - [CREATE EVENT](#)
 - [ALTER EVENT](#)
 - [DROP EVENT](#)
 - イベント実行:
 - データディクショナリに格納されているイベントメタデータである最終実行タイムスタンプが変更されるため、データベースでのイベントの実行は失敗します。イベントの実行に失敗すると、イベントスケジューラが停止する影響もあります。
 - イベントが読取り専用データベースのオブジェクトに書き込まれた場合、イベントの実行はエラーで失敗しますが、イベントスケジューラは停止しません。
- テーブル:
 - [CREATE TABLE](#)
 - [ALTER TABLE](#)
 - [CREATE INDEX](#)
 - [DROP INDEX](#)

- RENAME TABLE
- TRUNCATE TABLE
- DROP TABLE
- DELETE
- INSERT
- IMPORT TABLE
- LOAD DATA
- LOAD XML
- REPLACE
- UPDATE
- 子テーブルが読取り専用データベースにあるカスケード外部キーの場合、子テーブルが直接影響を受けない場合でも、親での更新および削除は拒否されます。
- CREATE TABLE s1.t(i int) ENGINE MERGE UNION (s2.t, s3.t), INSERT_METHOD=... などの MERGE テーブルの場合、次の動作が適用されます:
 - 挿入方法に関係なく、少なくともいずれかの s1, s2, s3 が読取り専用の場合、MERGE テーブル (INSERT into s1.t) への挿入は失敗します。挿入は、実際に書き込み可能なテーブルで終了する場合でも拒否されます。
 - s1 が読取り専用でないかぎり、MERGE テーブル (DROP TABLE s1.t) の削除は成功します。読取り専用データベースを参照する MERGE テーブルを削除できます。

ALTER DATABASE ステートメントは、変更中のデータベース内のオブジェクトにすでにアクセスしているすべての同時トランザクションがコミットされるまでブロックされます。逆に、同時 ALTER DATABASE で変更されるデータベース内のオブジェクトにアクセスする書き込みトランザクションは、ALTER DATABASE がコミットされるまでブロックされます。

クローンプラグインを使用してローカルまたはリモートのデータディレクトリをクローニングする場合、クローン内のデータベースは、ソースデータディレクトリにあった読取り専用状態を保持します。読取り専用状態は、クローニングプロセス自体には影響しません。クローンで同じデータベース読取り専用状態にすることが望ましくない場合は、クローンで ALTER DATABASE 操作を使用して、クローニングプロセスの終了後にクローンのオプションを明示的に変更する必要があります。

ドナーから受信者にクローニングする場合、受信者に読取り専用のユーザーデータベースがあると、クローニングは失敗し、エラーメッセージが表示されます。クローニングは、データベースを書き込み可能にした後に再試行できます。

READ ONLY は、ALTER DATABASE では許可されますが、CREATE DATABASE では許可されません。ただし、読取り専用データベースの場合、SHOW CREATE DATABASE によって生成されるステートメントには、読取り専用ステータスを示す READ ONLY=1 がコメント内に含まれます:

```
mysql> ALTER DATABASE mydb READ ONLY = 1;
mysql> SHOW CREATE DATABASE mydb\G
***** 1. row *****
Database: mydb
Create Database: CREATE DATABASE `mydb`
/*!40100 DEFAULT CHARACTER SET utf8mb4
COLLATE utf8mb4_0900_ai_ci */
/*!80016 DEFAULT ENCRYPTION='N' */
/* READ ONLY = 1 */
```

サーバーがこのようなコメントを含む CREATE DATABASE ステートメントを実行すると、サーバーはそのコメントを無視し、READ ONLY オプションは処理されません。これは、SHOW CREATE DATABASE を使用してダンプ出力に CREATE DATABASE ステートメントを生成する mysqldump および mysqlpump に影響します:

- ダンプファイルでは、読取り専用データベースの `CREATE DATABASE` ステートメントにコメント付きの `READ ONLY` オプションが含まれています。
- ダンプファイルは通常どおりリストアできますが、サーバーはコメント化された `READ ONLY` オプションを無視するため、リストアされたデータベースは読取り専用ではありません。リストア後にデータベースを読取り専用にする場合は、`ALTER DATABASE` を手動で実行して読取り専用にする必要があります。

`mydb` が読取り専用で、次のようにダンプするとします:

```
shell> mysqldump --databases mydb > mydb.sql
```

`mydb` を読取り専用にする必要がある場合は、後でリストア操作の後に `ALTER DATABASE` を実行する必要があります:

```
shell> mysql
mysql> SOURCE mydb.sql;
mysql> ALTER DATABASE mydb READ ONLY = 1;
```

MySQL Enterprise Backup にはこの問題はありません。読取り専用データベースは、他のデータベースと同様にバックアップおよびリストアされますが、バックアップ時に `READ ONLY` オプションが有効になっていた場合は、リストア時に有効になります。

`ALTER DATABASE` はバイナリログに書き込まれるため、レプリケーションソースサーバーで `READ ONLY` オプションを変更すると、複製にも影響します。これが発生しないようにするには、`ALTER DATABASE` ステートメントを実行する前にバイナリロギングを無効にする必要があります。たとえば、レプリカに影響を与えずにデータベースの移行を準備するには、次の操作を実行します:

1. 単一のセッション内で、バイナリロギングを無効にし、データベースに対して `READ ONLY` を有効にします:

```
mysql> SET sql_log_bin = OFF;
mysql> ALTER DATABASE mydb READ ONLY = 1;
```

2. たとえば、`mysqldump` または `mysqlpump` を使用してデータベースをダンプします:

```
shell> mysqldump --databases mydb > mydb.sql
```

3. 単一セッション内で、バイナリロギングを無効にし、データベースの `READ ONLY` を無効にします:

```
mysql> SET sql_log_bin = OFF;
mysql> ALTER DATABASE mydb READ ONLY = 0;
```

13.1.3 ALTER EVENT ステートメント

```
ALTER
[DEFINER = user]
EVENT event_name
[ON SCHEDULE schedule]
[ON COMPLETION [NOT] PRESERVE]
[RENAME TO new_event_name]
[ENABLE | DISABLE | DISABLE ON SLAVE]
[COMMENT 'string']
[DO event_body]
```

`ALTER EVENT` ステートメントは、既存のイベントの1つ以上の特性を、そのイベントを削除して再作成することなく変更します。`DEFINER`、`ON SCHEDULE`、`ON COMPLETION`、`COMMENT`、`ENABLE/DISABLE`、`DO` の各句の構文は、`CREATE EVENT` で使用される場合とまったく同じです。(セクション13.1.13「`CREATE EVENT ステートメント`」を参照してください。)

どのユーザーも、そのユーザーが `EVENT` 権限を持っているデータベースで定義されたイベントを変更できます。ユーザーが正常な `ALTER EVENT` ステートメントを実行すると、そのユーザーは、影響を受けるイベントの定義者になります。

`ALTER EVENT` は、既存のイベントでのみ機能します。

```
mysql> ALTER EVENT no_such_event
> ON SCHEDULE
> EVERY '2:3' DAY_HOUR;
ERROR 1517 (HY000): Unknown event 'no_such_event'
```

次の各例では、**myevent** という名前のイベントが次に示すように定義されていることを前提にしています。

```
CREATE EVENT myevent
ON SCHEDULE
EVERY 6 HOUR
COMMENT 'A sample comment.'
DO
UPDATE myschema.mytable SET mycol = mycol + 1;
```

次のステートメントは、**myevent** のスケジュールを、ただちに開始して 6 時間ごとに 1 回から、ステートメントが実行された 4 時間後から開始して 12 時間ごとに 1 回に変更します。

```
ALTER EVENT myevent
ON SCHEDULE
EVERY 12 HOUR
STARTS CURRENT_TIMESTAMP + INTERVAL 4 HOUR;
```

イベントの複数の特性を 1 つのステートメントで変更できます。この例では、**myevent** によって実行される SQL ステートメントを、**mytable** のすべてのレコードを削除する SQL ステートメントに変更します。また、イベントのスケジュールも、この **ALTER EVENT** ステートメントが実行された 1 日あとに 1 回実行されるように変更します。

```
ALTER EVENT myevent
ON SCHEDULE
AT CURRENT_TIMESTAMP + INTERVAL 1 DAY
DO
TRUNCATE TABLE myschema.mytable;
```

ALTER EVENT ステートメントでは、変更したい特性のオプションのみを指定します。省略されたオプションでは、その既存の値が保持されます。これには、**ENABLE** などの、**CREATE EVENT** のデフォルト値もすべて含まれます。

myevent を無効にするには、この **ALTER EVENT** ステートメントを使用します。

```
ALTER EVENT myevent
DISABLE;
```

ON SCHEDULE 句では、組み込みの MySQL 関数やユーザー変数を含む式を使用して、そこに含まれているすべての **timestamp** または **interval** 値を取得できます。このような式でストアドルーチンやユーザー定義関数を使用したり、テーブル参照を使用したりすることはできません。ただし、**SELECT FROM DUAL** は使用できます。これは、**ALTER EVENT** ステートメントと **CREATE EVENT** ステートメントの両方に当てはまります。このような場合のストアドルーチン、ユーザー定義関数、およびテーブルへの参照は明確に禁止されており、エラーで失敗します (Bug #22830 を参照してください)。

DO 句に別の **ALTER EVENT** ステートメントを含む **ALTER EVENT** ステートメントは成功したように見えますが、結果として得られるスケジュールされたイベントをサーバーが実行しようとする、その実行はエラーで失敗します。

イベントの名前を変更するには、**ALTER EVENT** ステートメントの **RENAME TO** 句を使用します。このステートメントは、イベント **myevent** の名前を **yourevent** に変更します。

```
ALTER EVENT myevent
RENAME TO yourevent;
```

次に示すように、**ALTER EVENT ... RENAME TO ...** と **db_name.event_name** 表記を使用して、イベントを別のデータベースに移動することもできます。

```
ALTER EVENT olddb.myevent
RENAME TO newdb.yourevent;
```

前のステートメントを実行するには、それを実行するユーザーが、**olddb** および **newdb** データベースの両方に対する **EVENT** 権限を持っている必要があります。

注記

RENAME EVENT ステートメントはありません。

値 **DISABLE ON SLAVE** は、**ENABLE** または **DISABLE** のかわりにレプリカで使用され、レプリケーションソースサーバーで作成されレプリカにレプリケートされたがレプリカでは実行されなかったイベントを示します。通常、**DISABLE ON SLAVE** は必要に応じて自動的に設定されます。ただし、手動で変更することが必要になる場合もあります。詳細は、[セクション 17.5.1.16 「呼び出される機能のレプリケーション」](#) を参照してください。

13.1.4 ALTER FUNCTION ステートメント

```
ALTER FUNCTION func_name [characteristic ...]

characteristic: {
  COMMENT 'string'
| LANGUAGE SQL
| {CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA}
| SQL SECURITY {DEFINER | INVOKER}
}
```

このステートメントを使用すると、ストアドファンクションの特性を変更できます。ALTER FUNCTION ステートメントでは、複数の変更を指定できます。ただし、このステートメントを使用して、ストアドファンクションのパラメータまたは本体を変更することはできません。このような変更を行うには、DROP FUNCTION と CREATE FUNCTION を使用して、この関数を削除および再作成する必要があります。

この関数に対する ALTER ROUTINE 権限が必要です。(その権限は、関数作成者に自動的に付与されます。)バイナリロギングが有効になっている場合は、セクション25.7「ストアドプログラムバイナリロギング」で説明されているように、ALTER FUNCTION ステートメントに SUPER 権限も必要になる可能性があります。

13.1.5 ALTER INSTANCE ステートメント

```
ALTER INSTANCE instance_action

instance_action: {
| {ENABLE|DISABLE} INNODB REDO_LOG
| ROTATE INNODB MASTER KEY
| ROTATE BINLOG MASTER KEY
| RELOAD TLS
  [FOR CHANNEL {mysql_main | mysql_admin}]
| [NO ROLLBACK ON ERROR]
}
```

ALTER INSTANCE は、MySQL サーバーインスタンスに適用可能なアクションを定義します。このステートメントでは、次のアクションがサポートされています:

- ALTER INSTANCE {ENABLE | DISABLE} INNODB REDO_LOG

このアクションは、InnoDB redo ロギングを有効または無効にします。redo ロギングはデフォルトで有効になっています。この機能は、新しい MySQL インスタンスへのデータのロードのみを目的としています。ステートメントはバイナリログに書き込まれません。MySQL 8.0.21 で導入されました。

警告

本番システムで redo ロギングを無効にしないでください。redo ロギングが無効になっているときにサーバーを停止して再起動することは許可されていますが、redo ロギングが無効になっているときに予期しないサーバー停止ページが発生すると、データが失われ、インスタンスが破損する可能性があります。

ALTER INSTANCE [ENABLE|DISABLE] INNODB REDO_LOG 操作には排他的バックアップロックが必要で、これにより他の ALTER INSTANCE 操作が同時に実行されなくなります。その他の ALTER INSTANCE 操作は、ロックが解放されるまで待機してから実行する必要があります。

詳細は、redo ロギングの無効化を参照してください。

- ALTER INSTANCE ROTATE INNODB MASTER KEY

このアクションにより、InnoDB テーブルスペースの暗号化に使用されるマスター暗号化キーがローテーションされます。キーローテーションには、ENCRYPTION_KEY_ADMIN または SUPER 権限が必要です。このアクションを実行するには、キーリングプラグインをインストールして構成する必要があります。その手順は、セクション6.4.4「MySQL キーリング」を参照してください。

ALTER INSTANCE ROTATE INNODB MASTER KEY では、同時 DML がサポートされます。ただし、CREATE TABLE ... ENCRYPTION または ALTER TABLE ... ENCRYPTION 操作と同時に実行することはできず、これらのス

ステートメントの同時実行によって発生する可能性のある競合を防ぐためにロックが取得されます。競合するステートメントのいずれかが実行されている場合、別のステートメントを続行する前に完了する必要があります。

`ALTER INSTANCE ROTATE INNODB MASTER KEY` ステートメントは、レプリケートされたサーバーで実行できるようにバイナリログに書き込まれます。

`ALTER INSTANCE ROTATE INNODB MASTER KEY` の使用方法の詳細は、[セクション15.13「InnoDB 保存データ暗号化」](#) を参照してください。

- `ALTER INSTANCE ROTATE BINLOG MASTER KEY`

このアクションは、バイナリログの暗号化に使用されるバイナリログマスターキーをローテーションします。バイナリログマスターキーの鍵ローテーションには、`BINLOG_ENCRYPTION_ADMIN` または `SUPER` 権限が必要です。`binlog_encryption` システム変数が `OFF` に設定されている場合、このステートメントは使用できません。このアクションを実行するには、キーリングプラグインをインストールして構成する必要があります。その手順は、[セクション6.4.4「MySQL キーリング」](#) を参照してください。

`ALTER INSTANCE ROTATE BINLOG MASTER KEY` アクションはバイナリログに書き込まれず、レプリカでは実行されません。したがって、バイナリログマスターキーローテーションは、MySQL バージョンの混在を含むレプリケーション環境で実行できます。適用可能なすべてのソースサーバーおよびレプリカサーバーでバイナリログマスターキーの定期ローテーションをスケジュールするには、各サーバーで MySQL イベントスケジューラを有効にし、`CREATE EVENT` ステートメントを使用して `ALTER INSTANCE ROTATE BINLOG MASTER KEY` ステートメントを発行します。現在または以前のバイナリログマスターキーのいずれかが危険にさらされている可能性があるためにバイナリログマスターキーをローテーションした場合は、該当するすべてのソースおよびレプリカサーバーでステートメントを発行して、即時のコンプライアンスを検証できます。

プロセスが正しく完了しない場合や、予期しないサーバー停止によって中断された場合の対処方法など、`ALTER INSTANCE ROTATE BINLOG MASTER KEY` の追加の使用方法については、[セクション17.3.2「バイナリログファイルとリレーログファイルの暗号化」](#) を参照してください。

- `ALTER INSTANCE RELOAD TLS`

このアクションは、コンテキストを定義するシステム変数の現在の値から TLS コンテキストを再構成します。また、アクティブなコンテキスト値を反映するステータス変数も更新されます。このアクションには、`CONNECTION_ADMIN` 権限が必要です。TLS コンテキストの再構成の詳細 (コンテキスト関連のシステム変数やステータス変数など) は、[サーバー側のランタイム構成および暗号化された接続の監視](#) を参照してください。

デフォルトでは、ステートメントはメイン接続インタフェースの TLS コンテキストをリロードします。(MySQL 8.0.21 で使用可能な) `FOR CHANNEL` 句が指定されている場合、ステートメントは指定されたチャンネルの TLS コンテキストをリロード: メイン接続インタフェースの場合は `mysql_main`、管理接続インタフェースの場合は `mysql_admin`。様々なインタフェースの詳細は、[セクション5.1.12.1「接続インタフェース」](#) を参照してください。更新された TLS コンテキストプロパティは、パフォーマンススキーマ `tls_channel_status` テーブルで公開されます。[セクション27.12.19.11「tls_channel_status テーブル」](#) を参照してください。

メインインタフェースの TLS コンテキストを更新すると、そのインタフェースにデフォルト以外の TLS 値が構成されていないかぎり、メインインタフェースと同じ TLS コンテキストが使用されるため、管理インタフェースにも影響する可能性があります。

デフォルトでは、`RELOAD TLS` アクションはエラーでロールバックされ、構成値で新しい TLS コンテキストの作成が許可されていない場合は影響しません。以前のコンテキスト値は、引き続き新しい接続に使用されます。オプションの `NO ROLLBACK ON ERROR` 句が指定され、新しいコンテキストを作成できない場合、ロールバックは発生しません。代わりに、警告が生成され、ステートメントが適用されるインタフェース上の新しい接続の暗号化が無効になります。

`ALTER INSTANCE RELOAD TLS` ステートメントはバイナリログに書き込まれません (したがってレプリケートされません)。TLS 構成はローカルであり、関係するすべてのサーバーに必ずしも存在するわけではありません。

13.1.6 ALTER LOGFILE GROUP ステートメント

```
ALTER LOGFILE GROUP logfile_group
ADD UNDOFILE 'file_name'
[INITIAL_SIZE [=] size]
[WAIT]
```



```
ENGINE [=] engine_name
```

このステートメントは、'file_name' という名前の UNDO ファイルを既存のログファイルグループ logfile_group に追加します。ALTER LOGFILE GROUP ステートメントには、ADD UNDOFILE 句が 1 つだけ存在します。DROP UNDOFILE 句は、現在サポートされていません。

注記

NDB Cluster ディスクデータオブジェクトはすべて同じ名前空間を共有します。つまり、各ディスクデータオブジェクトは (単に、特定の型の各ディスクデータオブジェクトというだけでなく)、一意の名前が付けられている必要があります。たとえば、テーブルスペースと Undo ログファイルと同じ名前にしたり、Undo ログファイルとデータファイルと同じ名前にしたりすることはできません。

オプションの INITIAL_SIZE パラメータは、UNDO ファイルの初期サイズをバイト単位で設定します。指定されていない場合、初期サイズはデフォルトで 134217728 (128M バイト) になります。オプションで、my.cnf で使用されているものと同様に、size の後に一文字の略称を付けることもできます。一般に、これは M (M バイト) または G (G バイト) のどちらかの文字です。(Bug #13116514、Bug #16104705、Bug #62858)

32 ビットシステム上では、INITIAL_SIZE のサポートされる最大値は 4294967296 (4G バイト) です。(Bug #29186)

INITIAL_SIZE の許可される最小値は 1048576 (1M バイト) です。(Bug #29574)

注記

WAIT は解析されますが、それ以外は無視されます。このキーワードは現在何の効果もなく、将来の拡張のために用意されています。

ENGINE パラメータ (必須) は、このログファイルグループによって使用されるストレージエンジンを決定します。ここで、engine_name はそのストレージエンジンの名前です。現在、engine_name として受け入れられる値は「NDBCLUSTER」と「NDB」だけです。この 2 つの値は同等です。

次の例では、ログファイルグループ lg_3 がすでに CREATE LOGFILE GROUP を使用して作成されていることを前提にしています (セクション 13.1.16 「CREATE LOGFILE GROUP ステートメント」を参照してください)。

```
ALTER LOGFILE GROUP lg_3
  ADD UNDOFILE 'undo_10.dat'
  INITIAL_SIZE=32M
  ENGINE=NDBCLUSTER;
```

ALTER LOGFILE GROUP を ENGINE = NDBCLUSTER (または ENGINE = NDB) とともに使用すると、NDB Cluster データノードごとに UNDO ログファイルが作成されます。INFORMATION_SCHEMA.FILES テーブルをクエリーすることによって、UNDO ファイルが作成されたことを確認したり、それらに関する情報を取得したりできます。例:

```
mysql> SELECT FILE_NAME, LOGFILE_GROUP_NUMBER, EXTRA
-> FROM INFORMATION_SCHEMA.FILES
-> WHERE LOGFILE_GROUP_NAME = 'lg_3';
+-----+-----+-----+
| FILE_NAME | LOGFILE_GROUP_NUMBER | EXTRA |
+-----+-----+-----+
| newdata.dat | 0 | CLUSTER_NODE=3 |
| newdata.dat | 0 | CLUSTER_NODE=4 |
| undo_10.dat | 11 | CLUSTER_NODE=3 |
| undo_10.dat | 11 | CLUSTER_NODE=4 |
+-----+-----+-----+
4 rows in set (0.01 sec)
```

(セクション 26.15 「INFORMATION_SCHEMA FILES テーブル」を参照してください。)

UNDO_BUFFER_SIZE に使用されるメモリーは、サイズが SharedGlobalMemory データノード構成パラメータの値によって決定されるグローバルプールから取得されます。これには、InitialLogFileGroup データノード構成パラメータの設定により、このオプションに暗黙的に指定されるデフォルト値もすべて含まれます。

ALTER LOGFILE GROUP は NDB Cluster のディスクデータストレージでのみ役立ちます。詳細は、セクション 23.5.10 「NDB Cluster ディスクデータテーブル」を参照してください。

13.1.7 ALTER PROCEDURE ステートメント

```
ALTER PROCEDURE proc_name [characteristic ...]

characteristic: {
  COMMENT 'string'
| LANGUAGE SQL
| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
| SQL SECURITY { DEFINER | INVOKER }
}
```

このステートメントを使用すると、ストアードプロシージャの特性を変更できます。ALTER PROCEDURE ステートメントでは、複数の変更を指定できます。ただし、このステートメントを使用して、ストアードプロシージャのパラメータまたは本体を変更することはできません。このような変更を行うには、DROP PROCEDURE と CREATE PROCEDURE を使用して、このプロシージャを削除および再作成する必要があります。

このプロシージャに対する ALTER ROUTINE 権限が必要です。デフォルトでは、その権限は、プロシージャ作成者に自動的に付与されます。この動作は、automatic_sp_privileges システム変数を無効にすることによって変更できます。セクション25.2.2「ストアードルーチンと MySQL 権限」を参照してください。

13.1.8 ALTER SERVER ステートメント

```
ALTER SERVER server_name
  OPTIONS (option [, option] ...)
```

server_name のサーバー情報を変更して、CREATE SERVER ステートメント内で許可されているオプションのいずれかを調整します。それに応じて、mysql.servers テーブル内の対応するフィールドが更新されます。このステートメントには、SUPER 権限が必要です。

たとえば、USER オプションを更新するには、次のようにします。

```
ALTER SERVER s OPTIONS (USER 'sally');
```

ALTER SERVER によって暗黙的なコミットが発生します。セクション13.3.3「暗黙的なコミットを発生させるステートメント」を参照してください。

使用されているロギング形式に関係なく、ALTER SERVER はバイナリログに書き込まれません。

13.1.9 ALTER TABLE ステートメント

```
ALTER TABLE tbl_name
  [alter_option [, alter_option] ...]
  [partition_options]

alter_option: {
  table_options
| ADD [COLUMN] col_name column_definition
  [FIRST | AFTER col_name]
| ADD [COLUMN] (col_name column_definition,...)
| ADD {INDEX | KEY} [index_name]
  [index_type] (key_part,...) [index_option] ...
| ADD {FULLTEXT | SPATIAL} [INDEX | KEY] [index_name]
  (key_part,...) [index_option] ...
| ADD [CONSTRAINT [symbol]] PRIMARY KEY
  [index_type] (key_part,...)
  [index_option] ...
| ADD [CONSTRAINT [symbol]] UNIQUE [INDEX | KEY]
  [index_name] [index_type] (key_part,...)
  [index_option] ...
| ADD [CONSTRAINT [symbol]] FOREIGN KEY
  [index_name] (col_name,...)
  reference_definition
| ADD [CONSTRAINT [symbol]] CHECK (expr) [[NOT] ENFORCED]
| DROP {CHECK | CONSTRAINT} symbol
| ALTER {CHECK | CONSTRAINT} symbol [NOT] ENFORCED
| ALGORITHM [=] {DEFAULT | INSTANT | INPLACE | COPY}
| ALTER [COLUMN] col_name {
  SET DEFAULT {literal | expr}
| SET {VISIBLE | INVISIBLE}
```

```

| DROP DEFAULT
}
| ALTER INDEX index_name {VISIBLE | INVISIBLE}
| CHANGE [COLUMN] old_col_name new_col_name column_definition
  [FIRST | AFTER col_name]
| [DEFAULT] CHARACTER SET [=] charset_name [COLLATE [=] collation_name]
| CONVERT TO CHARACTER SET charset_name [COLLATE collation_name]
| {DISABLE | ENABLE} KEYS
| {DISCARD | IMPORT} TABLESPACE
| DROP [COLUMN] col_name
| DROP {INDEX | KEY} index_name
| DROP PRIMARY KEY
| DROP FOREIGN KEY fk_symbol
| FORCE
| LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE}
| MODIFY [COLUMN] col_name column_definition
  [FIRST | AFTER col_name]
| ORDER BY col_name [, col_name] ...
| RENAME COLUMN old_col_name TO new_col_name
| RENAME {INDEX | KEY} old_index_name TO new_index_name
| RENAME [TO | AS] new_tbl_name
| {WITHOUT | WITH} VALIDATION
}

partition_options:
  partition_option [partition_option] ...

partition_option: {
  ADD PARTITION (partition_definition)
| DROP PARTITION partition_names
| DISCARD PARTITION {partition_names | ALL} TABLESPACE
| IMPORT PARTITION {partition_names | ALL} TABLESPACE
| TRUNCATE PARTITION {partition_names | ALL}
| COALESCE PARTITION number
| REORGANIZE PARTITION partition_names INTO (partition_definitions)
| EXCHANGE PARTITION partition_name WITH TABLE tbl_name [{WITH | WITHOUT} VALIDATION]
| ANALYZE PARTITION {partition_names | ALL}
| CHECK PARTITION {partition_names | ALL}
| OPTIMIZE PARTITION {partition_names | ALL}
| REBUILD PARTITION {partition_names | ALL}
| REPAIR PARTITION {partition_names | ALL}
| REMOVE PARTITIONING
}

key_part: {col_name [(length)] | (expr)} [ASC | DESC]

index_type:
  USING {BTREE | HASH}

index_option: {
  KEY_BLOCK_SIZE [=] value
| index_type
| WITH PARSER parser_name
| COMMENT 'string'
| {VISIBLE | INVISIBLE}
}

table_options:
  table_option [, table_option] ...

table_option: {
  AUTOEXTEND_SIZE [=] value
| AUTO_INCREMENT [=] value
| AVG_ROW_LENGTH [=] value
| [DEFAULT] CHARACTER SET [=] charset_name
| CHECKSUM [=] {0 | 1}
| [DEFAULT] COLLATE [=] collation_name
| COMMENT [=] 'string'
| COMPRESSION [=] {'ZLIB' | 'LZ4' | 'NONE'}
| CONNECTION [=] 'connect_string'
| {DATA | INDEX} DIRECTORY [=] 'absolute path to directory'
| DELAY_KEY_WRITE [=] {0 | 1}
| ENCRYPTION [=] {'Y' | 'N'}
| ENGINE [=] engine_name
| ENGINE_ATTRIBUTE [=] 'string'

```

```

| INSERT_METHOD [=] { NO | FIRST | LAST }
| KEY_BLOCK_SIZE [=] value
| MAX_ROWS [=] value
| MIN_ROWS [=] value
| PACK_KEYS [=] { 0 | 1 | DEFAULT }
| PASSWORD [=] 'string'
| ROW_FORMAT [=] { DEFAULT | DYNAMIC | FIXED | COMPRESSED | REDUNDANT | COMPACT }
| SECONDARY_ENGINE_ATTRIBUTE [=] 'string'
| STATS_AUTO_RECALC [=] { DEFAULT | 0 | 1 }
| STATS_PERSISTENT [=] { DEFAULT | 0 | 1 }
| STATS_SAMPLE_PAGES [=] value
| TABLESPACE tablespace_name [STORAGE {DISK | MEMORY}]
| UNION [=] (tbl_name[,tbl_name]...)
}

partition_options:
(see CREATE TABLE options)

```

ALTER TABLE は、テーブルの構造を変更します。たとえば、カラムを追加または削除したり、インデックスを作成または破棄したり、既存のカラムの型を変更したり、カラムまたはテーブル自体の名前を変更したりできます。また、テーブルに使用されているストレージエンジンやテーブルのコメントなどの特性を変更することもできます。

- **ALTER TABLE** を使用するには、このテーブルに対する **ALTER**、**CREATE**、および **INSERT** 権限が必要です。テーブルを名前変更するには、古いテーブル側で **ALTER** および **DROP** と、新しいテーブル側で **ALTER**、**CREATE**、および **INSERT** 権限が必要です。
- テーブル名のあとに、行う変更を指定します。何も指定されていない場合、**ALTER TABLE** は何もしません。
- 許容される変更の多くの構文は、**CREATE TABLE** ステートメントの句と似ています。column_definition 句では、**ADD** および **CHANGE** に **CREATE TABLE** と同じ構文を使用します。詳細は、[セクション13.1.20「CREATE TABLE ステートメント」](#)を参照してください。
- **COLUMN** という語はオプションであり、**RENAME COLUMN** を除いて省略できます (**RENAME** テーブルのネーミング操作とカラムのネーミング操作を区別するため)。
- 複数の **ADD**、**ALTER**、**DROP** 句および **CHANGE** 句をカンマで区切って単一の **ALTER TABLE** ステートメントで使用できます。これは、**ALTER TABLE** ステートメントごとに各句が 1 つしか許可されない標準 SQL への MySQL 拡張です。たとえば、1 つのステートメントで複数のカラムを削除するには、次のようにします。

```
ALTER TABLE t2 DROP COLUMN c, DROP COLUMN d;
```

- ストレージエンジンが試行された **ALTER TABLE** 操作をサポートしていない場合は、警告が表示されることがあります。このような警告は、**SHOW WARNINGS** で表示できます。[セクション13.7.7.42「SHOW WARNINGS ステートメント」](#)を参照してください。**ALTER TABLE** のトラブルシューティングについては、[セクションB.3.6.1「ALTER TABLE での問題」](#)を参照してください。
- 生成されるカラムの詳細は、[セクション13.1.9.2「ALTER TABLE および生成されるカラム」](#)を参照してください。
- 使用例については、[セクション13.1.9.3「ALTER TABLE の例」](#)を参照してください。
- MySQL 8.0.17 以降の InnoDB では、key_part 仕様を使用した JSON カラムへの複数値インデックスの追加を (**CAST json_path AS type ARRAY**) の形式でサポートしています。複数値インデックスの作成と使用方法、および複数値インデックスの制限および制限の詳細は、[複数値インデックス](#)を参照してください。
- `mysql_info()` C API 関数を使用すると、**ALTER TABLE** によってコピーされた行数を確認できます。[mysql_info\(\)](#)を参照してください。

このセクションの次のトピックで説明するように、**ALTER TABLE** ステートメントにはさらにいくつかの側面があります:

- [テーブルオプション](#)
- [パフォーマンスおよび領域要件](#)
- [同時実行性制御](#)
- [カラムの追加および削除](#)

- [カラムの名前変更、再定義および並替え](#)
- [主キーとインデックス](#)
- [外部キーおよびその他の制約](#)
- [文字セットの変更](#)
- [InnoDB テーブルのインポート](#)
- [MyISAM テーブルの行順序](#)
- [パーティショニングオプション](#)

テーブルオプション

`table_options` は、`ENGINE`、`AUTO_INCREMENT`、`AVG_ROW_LENGTH`、`MAX_ROWS`、`ROW_FORMAT` や `TABLESPACE` などの `CREATE TABLE` ステートメントで使用できる種類のテーブルオプションを示します。

すべてのテーブルオプションの詳細は、[セクション13.1.20「CREATE TABLE ステートメント」](#) を参照してください。ただし、テーブルオプションとして指定されている場合、`ALTER TABLE` は `DATA DIRECTORY` および `INDEX DIRECTORY` を無視します。`ALTER TABLE` では、パーティション化オプションとしてのみ使用でき、`FILE` 権限が必要です。

`ALTER TABLE` でテーブルオプションを使用すると、単一のテーブル特性を簡単に変更できます。例:

- `t1` が現在 InnoDB テーブルでない場合、次のステートメントはストレージエンジンを InnoDB に変更します:

```
ALTER TABLE t1 ENGINE = InnoDB;
```

- テーブルを InnoDB ストレージエンジンに切り替えるときの考慮事項については、[セクション15.6.1.5「MyISAM から InnoDB へのテーブルの変換」](#) を参照してください。
- `ENGINE` 句を指定すると、`ALTER TABLE` はテーブルを再構築します。これは、そのテーブルに指定されたストレージエンジンがすでに存在する場合にも当てはまります。
- 既存の InnoDB テーブルに対して `ALTER TABLE tbl_name ENGINE=INNODB` を実行すると、「null」`ALTER TABLE` 操作が実行されます。これは、[セクション15.11.4「テーブルのデフラグ」](#) で説明されているように、InnoDB テーブルのデフラグに使用できます。InnoDB テーブルに対して `ALTER TABLE tbl_name FORCE` を実行しても、同じ機能が実行されます。
- `ALTER TABLE tbl_name ENGINE=INNODB` および `ALTER TABLE tbl_name FORCE` では、[online DDL](#) を使用します。詳細は、[セクション15.12「InnoDB とオンライン DDL」](#) を参照してください。
- テーブルのストレージエンジンを変更しようとした結果は、[セクション5.1.11「サーバー SQL モード」](#) で説明されているように、目的のストレージエンジンが使用可能かどうか、および `NO_ENGINE_SUBSTITUTION` SQL モードの設定の影響を受けます。
- データが誤って失われることのないように、`ALTER TABLE` を使用して、テーブルのストレージエンジンを `MERGE` または `BLACKHOLE` に変更することはできません。
- 圧縮された行ストレージ形式を使用するように InnoDB テーブルを変更するには:

```
ALTER TABLE t1 ROW_FORMAT = COMPRESSED;
```

- `ENCRYPTION` 句は、InnoDB テーブルのページレベルのデータ暗号化を有効または無効にします。暗号化を有効にするには、キーリングプラグインをインストールして構成する必要があります。

`table_encryption_privilege_check` 変数が有効になっている場合、デフォルトのスキーマ暗号化設定とは異なる設定で `ENCRYPTION` 句を使用するには、`TABLE_ENCRYPTION_ADMIN` 権限が必要です。

MySQL 8.0.16 より前では、`ENCRYPTION` 句は file-per-table テーブルスペースに存在するテーブルを変更する場合にのみサポートされていました。MySQL 8.0.16 では、`ENCRYPTION` 句は一般的なテーブルスペースに存在するテーブルに対してもサポートされています。

一般的なテーブルスペースに存在するテーブルの場合、テーブルとテーブルスペースの暗号化は一致する必要があります。

テーブルを別のテーブルスペースに移動したり、ストレージエンジンを変更したりしてテーブルの暗号化を変更したりするには、`ENCRYPTION` 句を明示的に指定する必要があります。

MySQL 8.0.16 の時点では、暗号化をサポートしていないストレージエンジンがテーブルで使用されている場合、`'N'`または`''`以外の値で `ENCRYPTION` 句を指定することはできません。以前は、条項は受け入れられました。暗号化をサポートしていないストレージエンジンを使用して、暗号化対応スキーマで `ENCRYPTION` 句なしでテーブルを作成しようとすることも許可されていません。

詳細は、[セクション15.13「InnoDB 保存データ暗号化」](#)を参照してください。

- 現在の自動インクリメント値をリセットするには:

```
ALTER TABLE t1 AUTO_INCREMENT = 13;
```

このカウンタを、現在使用されている値以下の値にリセットすることはできません。InnoDB と MyISAM のどちらの場合も、この値が現在 `AUTO_INCREMENT` カラム内にある最大値以下である場合、この値は現在の `AUTO_INCREMENT` カラムの最大値に 1 を加えた値にリセットされます。

- デフォルトのテーブルの文字セットを変更するには:

```
ALTER TABLE t1 CHARACTER SET = utf8;
```

[文字セットの変更](#)も参照してください。

- テーブルコメントを追加 (または変更) するには:

```
ALTER TABLE t1 COMMENT = 'New table comment';
```

- 既存の `general tablespaces`、`file-per-table` テーブルスペースおよび `system tablespace` 間で InnoDB テーブルを移動するには、`TABLESPACE` オプションを指定して `ALTER TABLE` を使用します。[ALTER TABLE を使用したテーブルスペース間のテーブルの移動](#)を参照してください。

- `ALTER TABLE ... TABLESPACE` 操作では、`TABLESPACE` 属性が以前の値から変更されていない場合でも、常に全テーブルが再構築されます。

- `ALTER TABLE ... TABLESPACE` 構文では、一時テーブルスペースから永続テーブルスペースへのテーブルの移動はサポートされていません。

- `CREATE TABLE ... TABLESPACE` でサポートされている `DATA DIRECTORY` 句は、`ALTER TABLE ... TABLESPACE` ではサポートされず、指定されている場合は無視されます。

- `TABLESPACE` オプションの機能および制限の詳細は、[CREATE TABLE](#) を参照してください。

- MySQL NDB Cluster 8.0 は、次の例に示すように、`ALTER TABLE` ステートメントのテーブルコメントの一部として、テーブルパーティションバランス (フラグメントカウントタイプ)、`read-from-any-replica` 機能、フルレプリケーション、またはこれらの任意の組み合わせを `CREATE TABLE` と同じ方法で制御するための `NDB_TABLE` オプションの設定をサポートしています:

```
ALTER TABLE t1 COMMENT = "NDB_TABLE=READ_BACKUP=0,PARTITION_BALANCE=FOR_RA_BY_NODE";
```

`ALTER TABLE ... COMMENT ...` では、テーブルの既存のコメントはすべて破棄されることに注意してください。追加情報および例については、[「NDB_TABLE の設定」オプション](#)を参照してください。

- `ENGINE_ATTRIBUTE` および `SECONDARY_ENGINE_ATTRIBUTE` オプション (MySQL 8.0.21 の時点で使用可能) は、プライマリおよびセカンダリストレージエンジンのテーブル、カラム、およびインデックス属性を指定するために使用されます。オプションは、将来の使用のために予約されています。インデックス属性は変更できません。インデックスを削除し、必要な変更を加えて再度追加する必要があります。これは、単一の `ALTER TABLE` ステートメントで実行できます。

テーブルオプションが意図したとおりに変更されたことを確認するには、`SHOW CREATE TABLE` を使用するか、`INFORMATION_SCHEMA.TABLES` テーブルをクエリーします。

パフォーマンスおよび領域要件

ALTER TABLE 操作は、次のいずれかのアルゴリズムを使用して処理されます:

- **COPY**: 操作は元のテーブルのコピーに対して実行され、テーブルデータは元のテーブルから新しいテーブルの行ごとにコピーされます。同時 DML は許可されません。
- **INPLACE**: 操作ではテーブルデータのコピーは回避されますが、テーブルが適切に再構築される可能性があります。操作の準備フェーズおよび実行フェーズでは、テーブルに対する排他的メタデータロックが短時間で取得される場合があります。通常、同時 DML はサポートされています。
- **INSTANT**: 操作では、データディクショナリ内のメタデータのみが変更されます。準備および実行中にテーブルに対する排他的メタデータロックは行われず、テーブルデータは影響を受けず、操作が即時に行われます。同時 DML が許可されます。(MySQL 8.0.12 で導入)

ALGORITHM 句はオプションです。ALGORITHM 句を省略すると、MySQL では、ストレージエンジンおよびそれをサポートする ALTER TABLE 句に ALGORITHM=INSTANT が使用されます。それ以外の場合は、ALGORITHM=INPLACE が使用されます。ALGORITHM=INPLACE がサポートされていない場合、ALGORITHM=COPY が使用されます。

ALGORITHM 句を指定するには、それをサポートする句およびストレージエンジンに指定されたアルゴリズムを使用する操作が必要です。そうしないと、エラーで失敗します。ALGORITHM=DEFAULT を指定することは、ALGORITHM 句を省略することと同じです。

COPY アルゴリズムを使用する ALTER TABLE 操作は、テーブルを変更している他の操作が完了するまで待機します。変更がテーブルコピーに適用されると、データがコピーされ、元のテーブルが削除され、テーブルコピーの名前が元のテーブルの名前に変更されます。ALTER TABLE 操作の実行中、元のテーブルは他のセッションで読み取り可能です(ただし、すぐに記載されている例外があります)。ALTER TABLE 操作の開始後に開始されたテーブルの更新および書き込みは、新しいテーブルの準備ができるまで停止され、新しいテーブルに自動的にリダイレクトされます。テーブルの一時コピーは、別のディレクトリに存在するデータベースにテーブルを移動する RENAME TO 操作でないかぎり、元のテーブルのデータベースディレクトリに作成されます。

前述の例外は、古いテーブル構造をテーブルおよびテーブル定義キャッシュからクリアする準備が整った時点で、ALTER TABLE が読み取り(書き込みのみではなく)をブロックすることです。この時点で、このステートメントは排他的ロックを取得する必要があります。これを行うには、現在のリーダーが終了するまで待機し、新しい読み取りおよび書き込みをブロックします。

COPY アルゴリズムを使用する ALTER TABLE 操作により、同時 DML 操作が防止されます。並列クエリーは、引き続き許可されます。つまり、テーブルコピー操作には常に、少なくとも LOCK=SHARED (クエリーを許可するが、DML は許可しない)の並列性の制限が含まれます。DML およびクエリーを防止する LOCK=EXCLUSIVE を指定することで、LOCK 句をサポートする操作の同時実行性をさらに制限できます。詳細は、[同時実行性制御](#)を参照してください。

それ以外の場合は使用しない ALTER TABLE 操作に COPY アルゴリズムを強制的に使用するには、ALGORITHM=COPY を指定するか、old_alter_table システム変数を有効にします。old_alter_table 設定と、DEFAULT 以外の値を持つ ALGORITHM 句の間に矛盾がある場合は、ALGORITHM 句が優先されます。

InnoDB テーブルの場合、shared tablespace に存在するテーブルで COPY アルゴリズムを使用する ALTER TABLE 操作によって、テーブルスペースで使用される領域の量が増加する可能性があります。このような操作には、テーブルのデータとインデックスと同じ追加領域が必要です。共有テーブルスペースに存在するテーブルの場合、操作中使用された追加領域は、file-per-table テーブルスペースに存在するテーブル用であるため、オペレーティングシステムに解放されません。

オンライン DDL 操作の領域要件の詳細は、[セクション15.12.3「オンライン DDL 領域の要件」](#)を参照してください。

INPLACE アルゴリズムをサポートする ALTER TABLE 操作には、次のものがあります:

- InnoDB online DDL 機能でサポートされている ALTER TABLE 操作。[セクション15.12.1「オンライン DDL 操作」](#)を参照してください。
- テーブルの名前の変更。MySQL は、コピーを作成せずに、テーブル tbl_name に対応するファイルの名前を変更します。(RENAME TABLE ステートメントを使用してテーブルの名前を変更することもできます。[セクション](#)

13.1.36「[RENAME TABLE ステートメント](#)」を参照してください。)名前を変更したテーブル専用付与された権限は、新しい名前に移行されません。それらは、手動で変更する必要があります。

- テーブルメタデータのみを変更する操作。サーバーがテーブルの内容に触れないため、これらの操作はすぐに行われます。メタデータのみには次のものがあります:
 - カラム名の変更。NDB Cluster 8.0.18 以降では、この操作をオンラインで実行することもできます。
 - カラムのデフォルト値の変更 (NDB テーブルを除く)。
 - データ型の記憶域サイズが変更されないかぎり、新しい列挙を追加するか、有効なメンバー値のリストの end にメンバーを設定して、ENUM または SET カラムの定義を変更します。たとえば、8 つのメンバーを持つ SET カラムにメンバーを追加すると、値ごとに必要な記憶域が 1 バイトから 2 バイトに変更されます。これにはテーブルのコピーが必要です。リストの途中でメンバーを追加すると、既存のメンバーの番号が変更されます。これには、テーブルコピーが必要になります。
 - 空間カラムの定義を変更して、SRID 属性を削除します。(SRID 属性を追加または変更するには再構築が必要であり、サーバーはすべての値に指定された SRID 値があることを確認する必要があるため、再構築を実行できません。)
 - MySQL 8.0.14 では、次の条件が適用される場合にカラムの文字セットを変更します:
 - カラムのデータ型は、CHAR、VARCHAR、TEXT 型または ENUM です。
 - 文字セットの変更は、utf8mb3 から utf8mb4 へ、または任意の文字セットから binary へです。
 - カラムにインデックスがありません。
 - MySQL 8.0.14 では、次の条件が適用される場合、生成されたカラムを変更します:
 - InnoDB テーブルの場合、生成されたストアドカラムを変更するが、その型、式または NULL 値可能性は変更しないステートメント。
 - InnoDB 以外のテーブルの場合、生成されたストアドカラムまたは仮想カラムを変更するが、型、式または NULL 値可能性は変更しないステートメント。

このような変更の例として、カラムコメントの変更があります。

- インデックスの名前変更。
- InnoDB および NDB テーブルのセカンダリインデックスの追加または削除。 [セクション15.12.1「オンライン DDL 操作」](#)を参照してください。
- NDB テーブルの場合、可変幅のカラムに対してインデックスを追加および削除する操作。これらの操作は、テーブルのコピーなし、および同時 DML アクションをブロックせずに、ほとんどの期間オンラインで実行されます。 [セクション23.5.11「NDB Cluster での ALTER TABLE を使用したオンライン操作」](#)を参照してください。
- ALTER INDEX 操作によるインデックスの可視性の変更。
- 変更されたカラムが生成されたカラム式に含まれていない場合に、DEFAULT 値を持つカラムに依存する生成されたカラムを含むテーブルのカラムの変更。たとえば、テーブルを再構築せずに、別のカラムの NULL プロパティを変更できます。

INSTANT アルゴリズムをサポートする ALTER TABLE 操作には、次のものがあります:

- カラムの追加。この機能は、「[「インスタント ADD COLUMN」](#)」と呼ばれます。制限が適用されます。 [セクション15.12.1「オンライン DDL 操作」](#)を参照してください。
- 仮想カラムの追加または削除。
- カラムのデフォルト値の追加または削除。
- ENUM または SET カラムの定義の変更。前述の ALGORITHM=INSTANT の場合と同じ制限が適用されます。
- インデックスタイプの変更。

- テーブルの名前の変更。前述の `ALGORITHM=INSTANT` の場合と同じ制限が適用されます。

`ALGORITHM=INSTANT` をサポートする操作の詳細は、[セクション15.12.1「オンライン DDL 操作」](#) を参照してください。

`ALTER TABLE` は、`ADD COLUMN`、`CHANGE COLUMN`、`MODIFY COLUMN`、`ADD INDEX` および `FORCE` 操作のために、MySQL 5.5 一時カラムを 5.6 形式にアップグレードします。テーブルを再構築しなければならないため、この変換は `INPLACE` アルゴリズムを使用して実行することはできません。そのため、これらの場合に `ALGORITHM=INPLACE` を指定するとエラーになります。必要であれば、`ALGORITHM=COPY` を指定します。

`KEY` によってテーブルをパーティション化するために使用される複数カラムインデックスに対する `ALTER TABLE` 操作によってカラムの順序が変更される場合は、`ALGORITHM=COPY` を使用してのみ実行できます。

`WITHOUT VALIDATION` 句および `WITH VALIDATION` 句は、`ALTER TABLE` が `virtual generated column` の変更に対してインプレース操作を実行するかどうかに影響します。[セクション13.1.9.2「ALTER TABLE および生成されるカラム」](#) を参照してください。

NDB Cluster 8.0 は、標準 MySQL Server で使用されるものと同じ `ALGORITHM=INPLACE` 構文を使用したオンライン操作をサポートします。NDB はオンラインでのテーブルスペースの変更をサポートしていません。NDB 8.0.21 以降では許可されません。詳しくは[セクション23.5.11「NDB Cluster での ALTER TABLE を使用したオンライン操作」](#) をご覧ください。

`DISCARD ... PARTITION ... TABLESPACE` または `IMPORT ... PARTITION ... TABLESPACE` を使用した `ALTER TABLE` では、一時テーブルまたは一時パーティションファイルは作成されません。

`ALTER TABLE` with `ADD PARTITION`、`DROP PARTITION`、`COALESCE PARTITION`、`REBUILD PARTITION` または `REORGANIZE PARTITION` では、一時テーブルは作成されません (NDB テーブルとともに使用する場合を除く)。ただし、これらの操作では一時パーティションファイルを作成でき、作成できます。

`RANGE` または `LIST` パーティションに対する `ADD` または `DROP` 操作は即座の操作か、ほぼ即座の操作です。`HASH` または `KEY` パーティションに対する `ADD` または `COALESCE` 操作では、`LINEAR HASH` または `LINEAR KEY` が使用されていないかぎり、すべてのパーティション間でデータがコピーされます。`ADD` または `COALESCE` 操作はパーティションごとに実行されますが、これは実質的に、新しいテーブルの作成と同じです。`REORGANIZE` 操作では変更されたパーティションのみがコピーされ、変更されていないものはそのままです。

MyISAM テーブルの場合は、`mysam_sort_buffer_size` システム変数を大きな値に設定することによって、インデックスの再作成 (変更プロセスのもっとも遅い部分) を高速化できます。

同時実行性制御

これをサポートする `ALTER TABLE` 操作の場合は、`LOCK` 句を使用して、テーブルの変更中の同時読取りおよび書き込みのレベルを制御できます。この句にデフォルト以外の値を指定すると、変更操作中に特定の量の同時アクセスまたは排他性を必要とし、リクエストされたロックの程度が使用できない場合は操作を停止できます。

`ALGORITHM=INSTANT` を使用する操作には、`LOCK = DEFAULT` のみが許可されます。その他の `LOCK` 句パラメータは適用できません。

`LOCK` 句のパラメータは次のとおりです。

- `LOCK = DEFAULT`

指定された `ALGORITHM` 句 (存在する場合) および `ALTER TABLE` 操作に対する最大レベルの並列性: サポートされている場合は、並列読取りおよび書き込みを許可します。そうでない場合、サポートされている場合は、並列読取りを許可します。そうでない場合は、排他的アクセスを適用します。

- `LOCK = NONE`

サポートされている場合は、並列読取りおよび書き込みを許可します。それ以外の場合は、エラーが発生します。

- `LOCK = SHARED`

サポートされている場合は、並列読取りを許可しますが、書き込みはブロックします。ストレージエンジンが指定された `ALGORITHM` 句 (存在する場合) および `ALTER TABLE` 操作に対して同時書き込みをサポートしている場合でも、書き込みはブロックされます。同時読取りがサポートされていない場合は、エラーが発生します。

- `LOCK = EXCLUSIVE`

排他的アクセスを適用します。これは、指定された `ALGORITHM` 句 (存在する場合) および `ALTER TABLE` 操作について、ストレージエンジンによって同時読み取り/書き込みがサポートされている場合でも実行されます。

カラムの追加および削除

`ADD` を使用してテーブルに新しいカラムを追加し、`DROP` を使用して既存のカラムを削除します。 `DROP col_name` は、標準 SQL に対する MySQL の拡張機能です。

テーブル行内の特定の位置にカラムを追加するには、`FIRST` または `AFTER col_name` を使用します。デフォルトでは、そのカラムを最後に追加します。

テーブルに 1 つのカラムしか含まれていない場合は、そのカラムを削除できません。テーブルを削除する場合は、かわりに `DROP TABLE` ステートメントを使用します。

テーブルからカラムが削除された場合、そのカラムは、それが含まれているすべてのインデックスからも削除されます。インデックスを構成するすべてのカラムが削除された場合は、そのインデックスも削除されます。 `CHANGE` または `MODIFY` を使用して、インデックスが存在するカラムを短くしたときに、結果として得られるカラムの長さがインデックスの長さより短くなった場合、MySQL は自動的にそのインデックスを短くします。

`ALTER TABLE ... ADD` では、カラムに非決定的関数を使用する式のデフォルト値がある場合、ステートメントによって警告またはエラーが生成されることがあります。詳細は、[セクション11.6「データ型デフォルト値」](#) および [セクション17.1.3.7「GTID ベースレプリケーションの制約」](#) を参照してください。

カラムの名前変更、再定義および並替え

`CHANGE`、`MODIFY`、`RENAME COLUMN` 句および `ALTER` 句を使用すると、既存のカラムの名前および定義を変更できます。これらには次のような比較特性があります：

- `CHANGE`:
 - カラムの名前を変更し、その定義を変更するか、その両方を行うことができます。
 - `MODIFY` または `RENAME COLUMN` よりも多くの機能を備えていますが、一部の操作には便宜上役立ちます。 `CHANGE` では、名前を変更しない場合はカラムの名前を 2 回指定する必要があるため、名前の変更のみの場合はカラム定義を再指定する必要があります。
 - `FIRST` または `AFTER` では、カラムを並べ替えることができます。
- `MODIFY`:
 - カラム定義は変更できますが、名前は変更できません。
 - `CHANGE` よりも、名前を変更せずにカラム定義を変更する方が便利です。
 - `FIRST` または `AFTER` では、カラムを並べ替えることができます。
- `RENAME COLUMN`:
 - カラム名は変更できますが、その定義は変更できません。
 - `CHANGE` よりも、定義を変更せずにカラムの名前を変更する方が便利です。
- `ALTER`: カラムのデフォルト値を変更するためにのみ使用されます。

`CHANGE` は、標準 SQL に対する MySQL の拡張機能です。 `MODIFY` および `RENAME COLUMN` は、Oracle との互換性のための MySQL の拡張機能です。

カラムを変更してその名前と定義の両方を変更するには、古い名前と新しい名前、および新しい定義を指定して `CHANGE` を使用します。たとえば、`INT NOT NULL` カラムの名前を `a` から `b` に変更し、`NOT NULL` 属性を保持したまま `BIGINT` データ型を使用するようにその定義を変更するには、次のようにします：

```
ALTER TABLE t1 CHANGE a b BIGINT NOT NULL;
```

名前ではなくカラム定義を変更するには、**CHANGE** または **MODIFY** を使用します。**CHANGE** では、構文に 2 つのカラム名が必要であるため、名前を変更せずに同じ名前を 2 回指定する必要があります。たとえば、カラム **b** の定義を変更するには、次のようにします:

```
ALTER TABLE t1 CHANGE b b INT NOT NULL;
```

MODIFY では、カラム名が必要になるのは一度のみであるため、名前を変更せずに定義を変更する方が便利です:

```
ALTER TABLE t1 MODIFY b INT NOT NULL;
```

カラム名を変更するが、その定義は変更しない場合は、**CHANGE** または **RENAME COLUMN** を使用します。**CHANGE** では、構文にカラム定義が必要であるため、定義を変更しないでおくには、カラムに現在設定されている定義を再指定する必要があります。たとえば、**INT NOT NULL** カラムの名前を **b** から **a** に変更するには、次のようにします:

```
ALTER TABLE t1 CHANGE b a INT NOT NULL;
```

RENAME COLUMN では、古い名前と新しい名前のみが必要なため、定義を変更せずに名前を変更する方が便利です:

```
ALTER TABLE t1 RENAME COLUMN b TO a;
```

一般に、カラムの名前をテーブルにすでに存在する名前に変更することはできません。ただし、名前を入れ替えたりサイクル内で移動したりする場合などには、これは当てはまらないことがあります。テーブルに **a**、**b** および **c** という名前のカラムがある場合、これらは有効な操作です:

```
-- swap a and b
ALTER TABLE t1 RENAME COLUMN a TO b,
                RENAME COLUMN b TO a;
-- "rotate" a, b, c through a cycle
ALTER TABLE t1 RENAME COLUMN a TO b,
                RENAME COLUMN b TO c,
                RENAME COLUMN c TO a;
```

CHANGE または **MODIFY** を使用してカラム定義を変更する場合、**PRIMARY KEY** や **UNIQUE** などのインデックス属性以外の、新しいカラムに適用するデータ型およびすべての属性を定義に含める必要があります。元の定義には存在するが、新しい定義として指定されていない属性は引き継がれません。カラム **col1** が **INT UNSIGNED DEFAULT 1 COMMENT 'my column'** として定義されており、カラムを次のように変更して、**INT** のみを **BIGINT** に変更するとします:

```
ALTER TABLE t1 MODIFY col1 BIGINT;
```

このステートメントは、データ型を **INT** から **BIGINT** に変更しますが、**UNSIGNED**、**DEFAULT** および **COMMENT** 属性も削除します。これらを保持するには、ステートメントに明示的に含める必要があります:

```
ALTER TABLE t1 MODIFY col1 BIGINT UNSIGNED DEFAULT 1 COMMENT 'my column';
```

CHANGE または **MODIFY** を使用してデータ型を変更する場合、MySQL は既存のカラム値を可能なかぎり新しい型に変換しようとします。

警告

この変換によって、データが変更される可能性があります。たとえば、文字列カラムを短くすると、値が切り捨てられる可能性があります。新しいデータ型への変換によってデータが失われる場合は操作が成功しないようにするには、**ALTER TABLE** を使用する前に厳密な SQL モードを有効にします ([セクション 5.1.11 「サーバー SQL モード」](#) を参照してください)。

CHANGE または **MODIFY** を使用して、インデックスが存在するカラムを短くしたときに、結果として得られるカラムの長さがインデックスの長さより短くなった場合、MySQL は自動的にそのインデックスを短くします。

CHANGE または **RENAME COLUMN** によって名前が変更されたカラムの場合、MySQL は、これらの参照の名前を名前が変更されたカラムに自動的に変更します:

- 非表示インデックスや無効化された **MyISAM** インデックスなど、古いカラムを参照するインデックス。
- 古いカラムを参照する外部キー。

`CHANGE` または `RENAME COLUMN` によって名前が変更されたカラムの場合、MySQL は、名前が変更されたカラムへの次の参照の名前を自動的に変更しません:

- 名前が変更されたカラムを参照する生成されたカラムおよびパーティション式。 `CHANGE` を使用して、カラムの名前を変更するものと同じ `ALTER TABLE` ステートメントでこのような式を再定義する必要があります。
- 名前が変更されたカラムを参照するビューおよびストアドプログラム。これらのオブジェクトの定義は、新しいカラム名を参照するように手動で変更する必要があります。

テーブル内のカラムを並べ替えるには、`CHANGE` または `MODIFY` 操作で `FIRST` および `AFTER` を使用します。

`ALTER ... SET DEFAULT` または `ALTER ... DROP DEFAULT` は、それぞれカラムに新しいデフォルト値を指定するか、古いデフォルト値を削除します。古いデフォルトが削除され、かつカラムを `NULL` にできる場合、新しいデフォルトは `NULL` です。カラムを `NULL` にできない場合、MySQL は、[セクション11.6「データ型デフォルト値」](#)で説明されているようにデフォルト値を割り当てます。

MySQL 8.0.23 では、`ALTER ... SET VISIBLE` および `ALTER ... SET INVISIBLE` を使用してカラムの可視性を変更できます。[セクション13.1.20.10「非表示カラム」](#)を参照してください。

主キーとインデックス

`DROP PRIMARY KEY` により、`primary key` が削除されます。主キーが存在しない場合は、エラーが発生します。主キーのパフォーマンス特性 (特に InnoDB テーブルの場合) については、[セクション8.3.2「主キーの最適化」](#)を参照してください。

`sql_require_primary_key` システム変数が有効になっている場合、主キーを削除しようとするエラーが発生します。

テーブルに `UNIQUE INDEX` または `PRIMARY KEY` を追加すると、重複キーをできるだけ早く検出できるようにするために、MySQL はそれを一意でないどのインデックスよりも前に格納します。

`DROP INDEX` はインデックスを削除します。これは、標準 SQL への MySQL 拡張です。[セクション13.1.27「DROP INDEX ステートメント」](#)を参照してください。インデックス名を確認するには、`SHOW INDEX FROM tbl_name` を使用します。

一部のストレージエンジンでは、インデックスの作成時にインデックスタイプを指定できます。`index_type` 指定子の構文は、`USING type_name` です。`USING` の詳細は、[セクション13.1.15「CREATE INDEX ステートメント」](#)を参照してください。推奨される位置は、カラムリストのあとです。将来の MySQL リリースでは、カラムリストの前にオプションの使用がサポートされることが期待されます。

`index_option` 値は、インデックスの追加オプションを指定します。`USING` はそのようなオプションの1つです。許可される `index_option` 値の詳細は、[セクション13.1.15「CREATE INDEX ステートメント」](#)を参照してください。

`RENAME INDEX old_index_name TO new_index_name` は、インデックスの名前を変更します。これは、標準 SQL への MySQL 拡張です。テーブルの内容は変更されません。`old_index_name` は、同じ `ALTER TABLE` ステートメントで削除されないテーブル内の既存のインデックスの名前である必要があります。`new_index_name` は新しいインデックス名で、変更が適用された後に結果テーブルのインデックスの名前を複製することはできません。どちらのインデックス名も `PRIMARY` にできません。

MyISAM テーブルで `ALTER TABLE` を使用する場合は、(`REPAIR TABLE` の場合と同様に) 一意でないすべてのインデックスが個別のバッチで作成されます。多くのインデックスがあるときは、この方法で `ALTER TABLE` はるかに早くなります。

MyISAM テーブルの場合は、キーの更新を明示的に制御できます。`ALTER TABLE ... DISABLE KEYS` を使用して、一意でないインデックスの更新を停止するよう MySQL に指示します。次に、`ALTER TABLE ... ENABLE KEYS` を使用して、不足しているインデックスを再作成します。MyISAM はこれを、キーを1つずつ挿入するのに比べてはるかに高速な特殊なアルゴリズムで実行するため、一括挿入操作を実行する前にキーを無効にすると大幅な高速化が得られます。`ALTER TABLE ... DISABLE KEYS` を使用するには、先に説明した権限に加えて `INDEX` 権限が必要です。

一意でないインデックスは、無効になっている間、有効なときにはこのインデックスを使用する `SELECT` や `EXPLAIN` などのステートメントで無視されます。

`ALTER TABLE` ステートメントのあとに、インデックスカーディナリティー情報を更新するために `ANALYZE TABLE` の実行が必要になることがあります。[セクション13.7.7.22「SHOW INDEX ステートメント」](#)を参照してください。

ALTER INDEX 操作では、インデックスを可視または不可視にできます。不可視インデックスはオプティマイザでは使用されません。インデックスの可視性の変更は、主キー以外のインデックス (明示的または暗黙的) に適用されます。この機能はストレージエンジンに依存しません (すべてのエンジンでサポートされています)。詳細は、[セクション 8.3.12 「不可視のインデックス」](#) を参照してください。

外部キーおよびその他の制約

FOREIGN KEY および **REFERENCES** 句は、**ADD [CONSTRAINT [symbol]] FOREIGN KEY [index_name] (...)** **REFERENCES ... (...)** を実装する InnoDB および NDB ストレージエンジンによってサポートされます。[セクション 13.1.20.5 「FOREIGN KEY の制約」](#) を参照してください。その他のストレージエンジンでは、これらの句は解析されますが、無視されます。

ALTER TABLE では、**CREATE TABLE** とは異なり、**ADD FOREIGN KEY** は *index_name* (指定されている場合) を無視し、自動的に生成された外部キー名を使用します。回避方法として、外部キー名を指定する **CONSTRAINT** 句を含めます。

```
ADD CONSTRAINT name FOREIGN KEY (...)
```

重要

MySQL では、参照がカラム指定の一部として定義されているインライン **REFERENCES** 指定は暗黙的に無視されます。MySQL は、個別の **FOREIGN KEY** 仕様の一部として定義された **REFERENCES** 句のみを受け入れます。

注記

パーティション化された InnoDB テーブルは、外部キーをサポートしていません。この制限は、NDB テーブル (**[LINEAR] KEY** によって明示的にパーティション化されたテーブルを含む) には適用されません。詳細は、[セクション 24.6.2 「ストレージエンジンに関連するパーティショニング制限」](#) を参照してください。

MySQL Server と NDB Cluster はどちらも、**ALTER TABLE** を使用した外部キーの削除をサポートしています:

```
ALTER TABLE tbl_name DROP FOREIGN KEY fk_symbol;
```

同じ **ALTER TABLE** ステートメントでの外部キーの追加および削除は、**ALTER TABLE ... ALGORITHM=INPLACE** ではサポートされていますが、**ALTER TABLE ... ALGORITHM=COPY** ではサポートされていません。

サーバーは、参照整合性が失われる可能性がある外部キーカラムの変更を禁止します。回避方法として、カラム定義を変更する前に **ALTER TABLE ... DROP FOREIGN KEY** を使用し、あとで **ALTER TABLE ... ADD FOREIGN KEY** を使用します。禁止されている変更の例を次に示します:

- 安全でない可能性がある外部キーカラムのデータ型に対する変更。たとえば、**VARCHAR(20)** の **VARCHAR(30)** への変更は許可されますが、それを **VARCHAR(1024)** に変更することは、それによって個々の値を格納するために必要なバイト長の数が増えるため許可されません。
- 非制限モードで **NULL** カラムを **NOT NULL** に変更することは、参照テーブルに対応する値がないデフォルトの **NULL** 以外の値への **NULL** 値の変換を防ぐために禁止されています。この操作は厳密モードでは許可されますが、このような変換が必要な場合はエラーが返されます。

ALTER TABLE tbl_name RENAME new_tbl_name は、内部的に生成された外部キー制約名および文字列「tbl_name_ibfk_」で始まるユーザー定義の外部キー制約名を、新しいテーブル名を反映するように変更します。InnoDB は、文字列「tbl_name_ibfk_」で始まる外部キー制約名を内部的に生成された名前と解釈します。

MySQL 8.0.16 より前の **ALTER TABLE** では、次の限定バージョンの **CHECK** 制約追加構文のみが許可されていました。この構文は解析され、無視されます:

```
ADD CHECK (expr)
```

MySQL 8.0.16 の時点で、**ALTER TABLE** では、既存のテーブルの **CHECK** 制約を追加、削除または変更できます:

- 新しい **CHECK** 制約を追加します:

```
ALTER TABLE tbl_name
```



```
ADD CONSTRAINT [symbol] CHECK (expr) [[NOT] ENFORCED];
```

制約構文要素の意味は、[CREATE TABLE](#) の場合と同じです。 [セクション13.1.20.6「CHECK 制約」](#) を参照してください。

- `symbol` という名前の既存の `CHECK` 制約を削除します:

```
ALTER TABLE tbl_name  
DROP CHECK symbol;
```

- `symbol` という名前の既存の `CHECK` 制約が施行されるかどうかを変更します:

```
ALTER TABLE tbl_name  
ALTER CHECK symbol [NOT] ENFORCED;
```

`DROP CHECK` 句および `ALTER CHECK` 句は、標準 SQL に対する MySQL の拡張機能です。

MySQL 8.0.19 の時点で、`ALTER TABLE` では、制約タイプが制約名から決定される任意のタイプの既存の制約を削除および変更するために、より一般的な (および SQL 標準の) 構文を使用できます:

- `symbol` という名前の既存の制約を削除します:

```
ALTER TABLE tbl_name  
DROP CONSTRAINT symbol;
```

`sql_require_primary_key` システム変数が有効になっている場合、主キーを削除しようとするとエラーが発生します。

- `symbol` という名前の既存の制約を施行するかどうかを変更します:

```
ALTER TABLE tbl_name  
ALTER CONSTRAINT symbol [NOT] ENFORCED;
```

`CHECK` 制約のみ変更して強制終了できます。他のすべての制約タイプは常に適用されます。

SQL 標準では、すべてのタイプの制約 (主キー、一意インデックス、外部キー、チェック) が同じネームスペースに属することが指定されています。MySQL では、各制約タイプにスキーマごとに独自のネームスペースがあります。したがって、各タイプの制約の名前はスキーマごとに一意である必要がありますが、異なるタイプの制約には同じ名前を付けることができます。複数の制約の名前が同じ場合、`DROP CONSTRAINT` と `ADD CONSTRAINT` はあいまいであり、エラーが発生します。このような場合は、制約固有の構文を使用して制約を変更する必要があります。たとえば、主キーまたは外部キーを削除するには、`DROP PRIMARY KEY` または `DROP FOREIGN KEY` を使用します。

テーブルの変更によって `CHECK` 制約の施行違反が発生した場合、エラーが発生し、テーブルは変更されません。エラーが発生した操作の例:

- `CHECK` 制約で使用されるカラムに `AUTO_INCREMENT` 属性を追加しようとします。
- 施行された `CHECK` 制約を追加しようとするか、既存の行が制約条件に違反している非施行 `CHECK` 制約を施行しようとします。
- `CHECK` 制約で使用されているカラムを変更、名前変更または削除しようとします。ただし、その制約が同じステートメントでも削除されている場合を除きます。例外: `CHECK` 制約が単一のカラムのみを参照する場合、そのカラムを削除すると制約が自動的に削除されます。

`ALTER TABLE tbl_name RENAME new_tbl_name` は、新しいテーブル名を反映するために、文字列「tbl_name_chk_」で始まる内部生成およびユーザー定義の `CHECK` 制約名を変更します。MySQL は、文字列「tbl_name_chk_」で始まる `CHECK` 制約名を内部的に生成された名前と解釈します。

文字セットの変更

テーブルのデフォルトの文字セットおよびすべての文字カラム (`CHAR`、`VARCHAR`、`TEXT`) を新しい文字セットに変更するには、次のようなステートメントを使用します。

```
ALTER TABLE tbl_name CONVERT TO CHARACTER SET charset_name;
```

このステートメントでは、すべての文字カラムの照合順序も変更されます。使用する照合順序を示す `COLLATE` 句を指定しない場合、このステートメントは、その文字セットのデフォルトの照合順序を使用します。この照合順序が目

的とするテーブル使用に適していない (たとえば、大文字と小文字が区別される照合順序から大文字と小文字が区別されない照合順序に変更されてしまう) 場合は、照合順序を明示的に指定します。

`VARCHAR` のデータ型または `TEXT` 型のいずれかを持つカラムの場合、`CONVERT TO CHARACTER SET` は必要に応じてデータ型を変更し、新しいカラムが元のカラムと同じ数の文字を格納できる長さになるようにします。たとえば、`TEXT` カラムには、そのカラム内の値のバイト長 (最大 65,535) を格納するための 2 バイト長があります。 `latin1 TEXT` カラムの場合は、各文字に 1 バイトが必要なため、このカラムには最大 65,535 文字を格納できます。このカラムが `utf8` に変換された場合は、各文字に最大 3 バイトが必要になる可能性があるため、可能性のある最大の長さは $3 \times 65,535 = 196,605$ バイトになります。この長さは `TEXT` カラムの長さバイトに収まらないため、MySQL はデータ型を `MEDIUMTEXT` に変換します。これは、長さバイトが 196,605 の値を記録できる最小の文字列型です。同様に、`VARCHAR` カラムは `MEDIUMTEXT` に変換される可能性があります。

今説明した型のデータ型の変更を回避するには、`CONVERT TO CHARACTER SET` を使用しないでください。代わりに、`MODIFY` を使用して個々のカラムを変更します。例:

```
ALTER TABLE t MODIFY latin1_text_col TEXT CHARACTER SET utf8;  
ALTER TABLE t MODIFY latin1_varchar_col VARCHAR(M) CHARACTER SET utf8;
```

`CONVERT TO CHARACTER SET binary` を指定した場合、`CHAR`、`VARCHAR`、および `TEXT` カラムは、それぞれ対応するバイナリ文字列型 (`BINARY`、`VARBINARY`、`BLOB`) に変換されます。つまり、カラムには文字セットがなくなり、後続の `CONVERT TO` 操作は適用されません。

`CONVERT TO CHARACTER SET` 操作で `charset_name` が `DEFAULT` の場合は、`character_set_database` システム変数で指定された文字セットが使用されます。

警告

`CONVERT TO` 操作は、元の文字セットと名前付き文字セットの間でカラム値を変換します。これは、ある文字セット (`latin1` など) のカラムがあるが、格納された値が実際には、ほかの何らかの互換性のない文字セット (`utf8` など) を使用している場合に必要なものではありません。この場合は、このようなカラムごとに、次を実行する必要があります。

```
ALTER TABLE t1 CHANGE c1 c1 BLOB;  
ALTER TABLE t1 CHANGE c1 c1 TEXT CHARACTER SET utf8;
```

これが機能する理由は、`BLOB` カラムとの間で変換する場合は変換が発生しないためです。

テーブルのデフォルトの文字セットのみを変更するには、次のステートメントを使用します。

```
ALTER TABLE tbl_name DEFAULT CHARACTER SET charset_name;
```

ワード `DEFAULT` はオプションです。デフォルトの文字セットは、あとで (たとえば、`ALTER TABLE ... ADD column` で) テーブルに追加するカラムの文字セットを指定しない場合に使用される文字セットです。

`foreign_key_checks` システム変数 (デフォルト設定) が有効な場合、外部キー制約で 사용되는文字列カラムを含むテーブルでは文字セット変換は許可されません。回避策は、文字セット変換を実行する前に `foreign_key_checks` を無効にすることです。 `foreign_key_checks` を再度有効にする前に、外部キー制約に関係する両方のテーブルで変換を実行する必要があります。いずれかのテーブルのみを変換した後に `foreign_key_checks` を再度有効にすると、これらの操作中に暗黙的に変換されるために、`ON DELETE CASCADE` または `ON UPDATE CASCADE` 操作によって参照テーブルのデータが破損する可能性があります (Bug #45290、Bug #74816)。

InnoDB テーブルのインポート

独自の `file-per-table` テーブルスペースで作成された InnoDB テーブルは、`DISCARD TABLESPACE` 句および `IMPORT TABLESPACE` 句を使用して、バックアップまたは別の MySQL サーバーインスタンスからインポートできます。 [セクション 15.6.1.3 「InnoDB テーブルのインポート」](#) を参照してください。

MyISAM テーブルの行順序

`ORDER BY` では、特定の順序で行を含む新しいテーブルを作成できます。このオプションは、ほとんどの場合、特定の順序で行をクエリーすることがわかっている場合に主に役立ちます。このオプションをテーブルの大幅な変更のあとに使用すると、パフォーマンスの向上が得られる可能性があります。場合によっては、テーブルが、あとでその並べ替えに使用するカラムごとの順番になっていれば、MySQL でのソートが簡単になることがあります。

注記

挿入や削除を行うと、このテーブルは指定された順序のままではなくなります。

ORDER BY 構文では、ソートのためのカラム名を 1 つ以上指定できます。その各カラム名に続けて、オプションで、それぞれ昇順または降順のソート順序を示す **ASC** または **DESC** を指定できます。デフォルトは昇順です。ソート条件として許可されるのはカラム名だけです。任意の式は許可されていません。この句は、ほかのどの句よりもあとの最後に指定するようにしてください。

InnoDB は常に、[クラスタ化されたインデックス](#)に従ってテーブル行を並べ替えるため、**ORDER BY** は InnoDB テーブルでは意味がありません。

パーティション化されたテーブルに対して使用されている場合、**ALTER TABLE ... ORDER BY** は、各パーティション内でのみ行を並べ替えます。

パーティショニングオプション

partition_options は、パーティションの再パーティション化、パーティションの追加、削除、破棄、インポート、マージおよび分割、およびパーティション化メンテナンスの実行のためにパーティションテーブルで使用できるオプションを示します。

ALTER TABLE ステートメントには、ほかの変更指定に加えて、**PARTITION BY** または **REMOVE PARTITIONING** 句を含めることができますが、**PARTITION BY** または **REMOVE PARTITIONING** 句は、ほかのどの指定よりもあとの最後に指定する必要があります。リストされているオプションは個々のパーティションに作用するため、**ADD PARTITION**、**DROP PARTITION**、**DISCARD PARTITION**、**IMPORT PARTITION**、**COALESCE PARTITION**、**REORGANIZE PARTITION**、**EXCHANGE PARTITION**、**ANALYZE PARTITION**、**CHECK PARTITION** および **REPAIR PARTITION** オプションを単一の **ALTER TABLE** 内の他の変更指定と組み合わせることはできません。

パーティションのオプションの詳細は、[セクション13.1.20「CREATE TABLE ステートメント」](#) および [セクション13.1.9.1「ALTER TABLE パーティション操作」](#) を参照してください。 **ALTER TABLE ... EXCHANGE PARTITION** ステートメントの詳細および例については、[セクション24.3.3「パーティションとサブパーティションをテーブルと交換する」](#) を参照してください。

13.1.9.1 ALTER TABLE パーティション操作

ALTER TABLE のパーティション関連の句をパーティションテーブルとともに使用して、パーティションの再パーティション化、パーティションの追加、削除、破棄、インポート、マージおよび分割、パーティション化メンテナンスの実行を行うことができます。

- 単に、パーティション化されたテーブルに対して **ALTER TABLE** で **partition_options** 句を使用するだけで、**partition_options** で定義されたパーティション化スキームに従って、そのテーブルが再パーティション化されます。この句は常に **PARTITION BY** で始まり、**CREATE TABLE** の **partition_options** 句に適用されるものと同じ構文およびその他のルールに従い（詳細は、[セクション13.1.20「CREATE TABLE ステートメント」](#) を参照）、まだパーティション化されていない既存のテーブルのパーティション化にも使用できます。たとえば、次に示すように定義された（パーティション化されていない）テーブルを考えてみます。

```
CREATE TABLE t1 (  
  id INT,  
  year_col INT  
);
```

このテーブルは、次のステートメントを使用し、**id** カラムをパーティション化キーとして使用して **HASH** によって 8 つのパーティションにパーティション化できます。

```
ALTER TABLE t1  
  PARTITION BY HASH(id)  
  PARTITIONS 8;
```

MySQL は、**[SUB]PARTITION BY [LINEAR] KEY** で **ALGORITHM** オプションをサポートしています。**ALGORITHM=1** を指定すると、サーバーは、パーティション内の行の配置を計算するときに MySQL 5.1 と同じキーハッシュ関数を使用します。**ALGORITHM=2** は、サーバーが、MySQL 5.5 以降で実装され、**KEY** によってパーティション化された新しいテーブルに対してデフォルトで使用されるキーハッシュ関数を使用することを示します。（MySQL 5.5 以降で採用されたキーハッシュ関数によって作成されたパーティション化されたテーブルを MySQL 5.1 サーバーで使用することはできません。）このオプションを指定しない場合は、**ALGORITHM=2** を使用

するのと同じ効果があります。このオプションは、主に [\[LINEAR\] KEY](#) によってパーティション化されたテーブルを MySQL 5.1 以降の MySQL バージョン間でアップグレードまたはダウングレードするとき使用するか、または MySQL 5.5 以降のサーバー上で、MySQL 5.1 サーバー上で使用できる [KEY](#) または [LINEAR KEY](#) によってパーティション化されたテーブルを作成することを目的としています。

[ALTER TABLE ... PARTITION BY](#) ステートメントを使用して作成されたテーブルは、[CREATE TABLE ... PARTITION BY](#) を使用して作成されたテーブルと同じルールに従う必要があります。これには、そのテーブルに含まれている可能性のあるすべての一意のキー（すべての主キーを含む）と、パーティショニング式で使用されている 1 つまたは複数のカラムの間の関係を管理するルールが含まれます。これについては、[セクション24.6.1「パーティショニングキー、主キー、および一意キー」](#) で説明されています。また、パーティションの数を指定するための [CREATE TABLE ... PARTITION BY](#) のルールも [ALTER TABLE ... PARTITION BY](#) に適用されます。

[ALTER TABLE ADD PARTITION](#) の `partition_definition` 句は、[CREATE TABLE](#) ステートメントの同じ名前の句と同じオプションをサポートしています。（構文と説明については、[セクション13.1.20「CREATE TABLE ステートメント」](#) を参照してください。）次に示すように作成されたパーティション化されたテーブルがあるとします。

```
CREATE TABLE t1 (  
  id INT,  
  year_col INT  
)  
PARTITION BY RANGE (year_col) (  
  PARTITION p0 VALUES LESS THAN (1991),  
  PARTITION p1 VALUES LESS THAN (1995),  
  PARTITION p2 VALUES LESS THAN (1999)  
);
```

このテーブルに、[2002](#) より小さい値を格納するための新しいパーティション `p3` を次のように追加できます。

```
ALTER TABLE t1 ADD PARTITION (PARTITION p3 VALUES LESS THAN (2002));
```

[DROP PARTITION](#) を使用すると、1 つ以上の [RANGE](#) または [LIST](#) パーティションを削除できます。このステートメントは、[HASH](#) または [KEY](#) パーティションでは使用できません。かわりに、[COALESCE PARTITION](#) を使用してください（このセクションの後半を参照）。`partition_names` リストで名前が指定されている削除されたパーティションに格納されていたデータはすべて破棄されます。たとえば、前に定義されたテーブル `t1` の場合は、`p0` および `p1` という名前のパーティションを次に示すように削除できます。

```
ALTER TABLE t1 DROP PARTITION p0, p1;
```

注記

[DROP PARTITION](#) は、[NDB](#) ストレージエンジンを使用するテーブルでは機能しません。[セクション24.3.1「RANGE および LIST パーティションの管理」](#) および [セクション23.1.7「NDB Cluster の既知の制限事項」](#) を参照してください。

[ADD PARTITION](#) と [DROP PARTITION](#) は現在、[IF \[NOT\] EXISTS](#) をサポートしていません。

[DISCARD PARTITION ... TABLESPACE](#) および [IMPORT PARTITION ... TABLESPACE](#) オプションは、[Transportable Tablespace](#) 機能を個々の [InnoDB](#) テーブルパーティションに拡張します。各 [InnoDB](#) テーブルパーティションには、独自のテーブルスペースファイル（`.ibd` ファイル）があります。[Transportable Tablespace](#) 機能を使用すると、実行中の MySQL サーバーインスタンスから別の実行中のインスタンスにテーブルスペースを簡

単にコピーしたり、同じインスタンスでリストアを実行できます。どちらのオプションも、1つ以上のパーティション名のカンマ区切りリストを取ります。例:

```
ALTER TABLE t1 DISCARD PARTITION p2, p3 TABLESPACE;
```

```
ALTER TABLE t1 IMPORT PARTITION p2, p3 TABLESPACE;
```

サブパーティションテーブルで **DISCARD PARTITION ... TABLESPACE** および **IMPORT PARTITION ... TABLESPACE** を実行する場合、パーティション名とサブパーティション名の両方が許可されます。パーティション名を指定すると、そのパーティションのサブパーティションが含まれます。

Transportable Tablespace 機能では、パーティション InnoDB テーブルのコピーまたはリストアもサポートされています。詳細は、[セクション15.6.1.3「InnoDB テーブルのインポート」](#)を参照してください。

パーティションテーブルの名前変更はサポートされています。 **ALTER TABLE ... REORGANIZE PARTITION** を使用して個々のパーティションの名前を間接的に変更できますが、この操作ではパーティションデータがコピーされます。

選択したパーティションから行を削除するには、 **TRUNCATE PARTITION** オプションを使用します。このオプションは、1つ以上のカンマ区切りのパーティション名のリストを取ります。次のステートメントによって作成されたテーブル **t1** について考えてみます:

```
CREATE TABLE t1 (  
  id INT,  
  year_col INT  
)  
PARTITION BY RANGE (year_col) (  
  PARTITION p0 VALUES LESS THAN (1991),  
  PARTITION p1 VALUES LESS THAN (1995),  
  PARTITION p2 VALUES LESS THAN (1999),  
  PARTITION p3 VALUES LESS THAN (2003),  
  PARTITION p4 VALUES LESS THAN (2007)  
);
```

パーティション **p0** からすべての行を削除するには、次のステートメントを使用します:

```
ALTER TABLE t1 TRUNCATE PARTITION p0;
```

今示したステートメントには、次の **DELETE** ステートメントと同じ効果があります。

```
DELETE FROM t1 WHERE year_col < 1991;
```

複数のパーティションを切り詰める場合、パーティションが連続している必要はありません。これにより、通常、**DELETE** ステートメントで実行された場合は非常に複雑な **WHERE** 条件が必要になる、パーティション化されたテーブルでの削除操作が大幅に簡素化される可能性があります。たとえば、次のステートメントは、パーティション **p1** と **p3** のすべての行を削除します。

```
ALTER TABLE t1 TRUNCATE PARTITION p1, p3;
```

同等の **DELETE** ステートメントを次に示します。

```
DELETE FROM t1 WHERE  
  (year_col >= 1991 AND year_col < 1995)  
  OR  
  (year_col >= 2003 AND year_col < 2007);
```

パーティション名のリストのかわりに **ALL** キーワードを使用すると、ステートメントはすべてのテーブルパーティションに対して機能します。

TRUNCATE PARTITION は行を削除するだけです。そのテーブル自体や、どのパーティションの定義も変更されません。

行が削除されたことを確認するには、次のようなクエリーを使用して **INFORMATION_SCHEMA.PARTITIONS** テーブルを確認します:

```
SELECT PARTITION_NAME, TABLE_ROWS  
FROM INFORMATION_SCHEMA.PARTITIONS
```



```
WHERE TABLE_NAME = 't1';
```

HASH または **KEY** によってパーティション化されたテーブルで **COALESCE PARTITION** を使用すると、そのパーティションの数を **number** だけ減らすことができます。次のようにテーブル **t2** を作成したとします:

```
CREATE TABLE t2 (  
  name VARCHAR (30),  
  started DATE  
)  
PARTITION BY HASH( YEAR(started) )  
PARTITIONS 6;
```

t2 で使用されるパーティションの数を 6 から 4 に減らすには、次のステートメントを使用します:

```
ALTER TABLE t2 COALESCE PARTITION 2;
```

最後の **number** パーティションに含まれるデータは、残りのパーティションにマージされます。この場合、パーティション 4 および 5 は最初の 4 つのパーティション (0、1、2 および 3 の番号が付けられたパーティション) にマージされます。

パーティション化されたテーブルで使用される (すべてではなく) 一部のパーティションを変更するには、**REORGANIZE PARTITION** を使用できます。このステートメントは、次のいくつかの方法で使用できます。

- 一連のパーティションを単一パーティションにマージします。これを行うには、**partition_names** リスト内の複数のパーティションに名前を付け、**partition_definition** の単一の定義を指定します。
- 既存のパーティションをいくつかのパーティションに分割します。これを実現するには、**partition_names** の単一のパーティションに名前を付け、複数の **partition_definitions** を指定します。
- VALUES LESS THAN** を使用して、定義されたパーティションのサブセットの範囲を変更するか、または **VALUES IN** を使用して、定義されたパーティションのサブセットの値リストを変更します。

注記

明示的に名前が付けられていないパーティションに対して、MySQL は自動的に **p0**、**p1**、**p2** などのデフォルト名を付けます。同じことがサブパーティションにも当てはまります。

ALTER TABLE ... REORGANIZE PARTITION ステートメントの詳細および例については、[セクション 24.3.1 「RANGE および LIST パーティションの管理」](#) を参照してください。

- テーブルのパーティションまたはサブパーティションをテーブルと交換するには、**ALTER TABLE ... EXCHANGE PARTITION** ステートメントを使用します。つまり、パーティションまたはサブパーティション内の既存の行を非パーティションテーブルに移動し、非パーティションテーブル内の既存の行をテーブルのパーティションまたはサブパーティションに移動します。

使用方法および例については、[セクション 24.3.3 「パーティションとサブパーティションをテーブルと交換する」](#) を参照してください。

- いくつかのオプションでは、**CHECK TABLE** や **REPAIR TABLE** などのステートメントによって非パーティションテーブルに実装されるものと同様のパーティションメンテナンスおよび修復機能が提供されます (パーティションテーブルでもサポートされます。詳細は、[セクション 13.7.3 「テーブル保守ステートメント」](#) を参照してください)。これには、**ANALYZE PARTITION**、**CHECK PARTITION**、**OPTIMIZE PARTITION**、**REBUILD PARTITION**、および **REPAIR PARTITION** が含まれます。これらの各オプションは、1 つ以上のパーティション名から成るカンマで区切られた **partition_names** 句を受け取ります。パーティションはターゲットテーブルにすでに存在している必要があります。 **partition_names** のかわりに **ALL** キーワードを使用することもできます。この場合、ステートメントはすべてのテーブルパーティションで動作します。詳細および例については、[セクション 24.3.4 「パーティションの保守」](#) を参照してください。

InnoDB は現在、パーティションごとの最適化をサポートしていません。**ALTER TABLE ... OPTIMIZE PARTITION** では、テーブル全体が再構築および分析され、適切な警告が発行されます。(Bug #11751825、Bug #42822) こ

の問題を回避するには、かわりに `ALTER TABLE ... REBUILD PARTITION` および `ALTER TABLE ... ANALYZE PARTITION` を使用します。

`ANALYZE PARTITION`、`CHECK PARTITION`、`OPTIMIZE PARTITION` および `REPAIR PARTITION` オプションは、パーティション化されていないテーブルではサポートされていません。

- `REMOVE PARTITIONING` を使用すると、テーブルまたはそのデータに影響を与えることなく、テーブルのパーティション化を削除できます。このオプションは、カラムやインデックスの追加、削除、名前変更などのために使用されるその他の `ALTER TABLE` オプションと組み合わせることができます。
- `ALTER TABLE` で `ENGINE` オプションを使用すると、パーティション化に影響を与えることなく、テーブルで使用されるストレージエンジンが変更されます。ターゲットストレージエンジンは、独自のパーティショニングハンドラを提供する必要があります。ネイティブのパーティショニングハンドラを持つのは、`InnoDB` および `NDB` ストレージエンジンだけです。現在、`NDB` は MySQL 8.0 ではサポートされていません。

`ALTER TABLE` ステートメントには、ほかの変更指定に加えて、`PARTITION BY` または `REMOVE PARTITIONING` 句を含めることができますが、`PARTITION BY` または `REMOVE PARTITIONING` 句は、ほかのどの指定よりもあとの最後に指定する必要があります。

`ADD PARTITION`、`DROP PARTITION`、`COALESCE PARTITION`、`REORGANIZE PARTITION`、`ANALYZE PARTITION`、`CHECK PARTITION`、および `REPAIR PARTITION` オプションは、個々のパーティションに対して機能するため、1つの `ALTER TABLE` 内でほかの変更指定と組み合わせることはできません。詳細は、[セクション 13.1.9.1「ALTER TABLE パーティション操作」](#)を参照してください。

特定の `ALTER TABLE` ステートメントでは、次のいずれかが1つのオプションの単一インスタンスのみを使用できます。`PARTITION BY`、`ADD PARTITION`、`DROP PARTITION`、`TRUNCATE PARTITION`、`EXCHANGE PARTITION`、`REORGANIZE PARTITION`、または `COALESCE PARTITION`、`ANALYZE PARTITION`、`CHECK PARTITION`、`OPTIMIZE PARTITION`、`REBUILD PARTITION`、`REMOVE PARTITIONING`。

たとえば、次の2つのステートメントは無効です。

```
ALTER TABLE t1 ANALYZE PARTITION p1, ANALYZE PARTITION p2;
```

```
ALTER TABLE t1 ANALYZE PARTITION p1, CHECK PARTITION p2;
```

最初のケースでは、次のように、分析される両方のパーティションを一覧表示した1つの `ANALYZE PARTITION` オプションを含む1つのステートメントを使用して、テーブル `t1` のパーティション `p1` と `p2` を同時に分析できます。

```
ALTER TABLE t1 ANALYZE PARTITION p1, p2;
```

2番目のケースでは、同じテーブルの別のパーティションに対する `ANALYZE` 操作と `CHECK` 操作を同時に実行することはできません。代わりに、次のように、2つの個別のステートメントを発行する必要があります。

```
ALTER TABLE t1 ANALYZE PARTITION p1;  
ALTER TABLE t1 CHECK PARTITION p2;
```

サブパーティションに対する `REBUILD` 操作は現在サポートされていません。`REBUILD` キーワードはサブパーティションでは明示的に禁止されており、使用すると `ALTER TABLE` はエラーで失敗します。

チェックまたは修復するパーティションに重複キーエラーが含まれている場合、`CHECK PARTITION` および `REPAIR PARTITION` の操作は失敗します。

これらのステートメントの詳細は、[セクション24.3.4「パーティションの保守」](#)を参照してください。

13.1.9.2 ALTER TABLE および生成されるカラム

生成されるカラムに対して許可される `ALTER TABLE` 操作は、`ADD`、`MODIFY` および `CHANGE` です。

- 生成されたカラムを追加できます。

```
CREATE TABLE t1 (c1 INT);  
ALTER TABLE t1 ADD COLUMN c2 INT GENERATED ALWAYS AS (c1 + 1) STORED;
```

- 生成されたカラムのデータ型および式は変更できます。

```
CREATE TABLE t1 (c1 INT, c2 INT GENERATED ALWAYS AS (c1 + 1) STORED);  
ALTER TABLE t1 MODIFY COLUMN c2 TINYINT GENERATED ALWAYS AS (c1 + 5) STORED;
```

- 他のカラムが参照していない場合は、生成されたカラムの名前を変更または削除できます。

```
CREATE TABLE t1 (c1 INT, c2 INT GENERATED ALWAYS AS (c1 + 1) STORED);
ALTER TABLE t1 CHANGE c2 c3 INT GENERATED ALWAYS AS (c1 + 1) STORED;
ALTER TABLE t1 DROP COLUMN c3;
```

- 仮想生成カラムは、格納された生成カラムに変更できません。その逆も同様です。これを回避するには、カラムを削除してから、新しい定義を追加します。

```
CREATE TABLE t1 (c1 INT, c2 INT GENERATED ALWAYS AS (c1 + 1) VIRTUAL);
ALTER TABLE t1 DROP COLUMN c2;
ALTER TABLE t1 ADD COLUMN c2 INT GENERATED ALWAYS AS (c1 + 1) STORED;
```

- 非生成カラムは、格納済に変更できますが、仮想生成カラムには変更できません。

```
CREATE TABLE t1 (c1 INT, c2 INT);
ALTER TABLE t1 MODIFY COLUMN c2 INT GENERATED ALWAYS AS (c1 + 1) STORED;
```

- 格納されているが、仮想生成カラムは生成されていないカラムに変更できます。格納された生成値は、生成されないカラムの値になります。

```
CREATE TABLE t1 (c1 INT, c2 INT GENERATED ALWAYS AS (c1 + 1) STORED);
ALTER TABLE t1 MODIFY COLUMN c2 INT;
```

- 式はサーバーによって評価される必要があるため、**ADD COLUMN** はストアカラムのインプレース操作ではありません (一時テーブルを使用せずに実行されます)。ストアカラムの場合、インデックス付けの変更はインプレースで行われ、式の変更はインプレースでは行われません。カラムコメントへの変更はインプレースで行われます。
- パーティション化されていないテーブルの場合、**ADD COLUMN** および **DROP COLUMN** は仮想カラムのインプレース操作です。ただし、仮想カラムの追加または削除は、他の **ALTER TABLE** 操作と組み合わせて実行することはできません。

パーティションテーブルの場合、**ADD COLUMN** および **DROP COLUMN** は仮想カラムのインプレース操作ではありません。

- **InnoDB** では、仮想生成カラムのセカンダリインデックスがサポートされます。仮想生成カラムに対するセカンダリインデックスの追加または削除はインプレース操作です。詳細は、[セクション13.1.20.9「セカンダリインデックスと生成されたカラム」](#)を参照してください。
- **VIRTUAL** で生成されたカラムがテーブルに追加または変更された場合、生成されたカラム式によって計算されるデータがカラムの範囲外であることは保証されません。これにより、一貫性のないデータが返され、予期せず失敗したステートメントが発生する可能性があります。このようなカラムに対して検証が行われるかどうかを制御できるように、**ALTER TABLE** では **WITHOUT VALIDATION** 句および **WITH VALIDATION** 句がサポートされています:
 - **WITHOUT VALIDATION** (どちらの句も指定されていない場合のデフォルト) では、インプレース操作が実行され (可能な場合)、データ整合性はチェックされず、ステートメントはより迅速に終了します。ただし、後で値が範囲外の場合は、テーブルからの読取りでカラムの警告またはエラーが報告される可能性があります。
 - **WITH VALIDATION** では、**ALTER TABLE** によってテーブルがコピーされます。範囲外のエラーまたはその他のエラーが発生した場合、ステートメントは失敗します。テーブルのコピーが実行されるため、ステートメントに時間がかかります。

WITHOUT VALIDATION および **WITH VALIDATION** は、**ADD COLUMN**、**CHANGE COLUMN** および **MODIFY COLUMN** 操作でのみ許可されます。それ以外の場合は、**ER_WRONG_USAGE** エラーが発生します。

- 式の評価によって切捨てが発生した場合、または関数への入力が正しくない場合、**ALTER TABLE** ステートメントはエラーで終了し、DDL 操作は拒否されます。
- カラムのデフォルト値を変更する **ALTER TABLE** ステートメントでは、**col_name** を使用してカラムを参照する生成されたカラム式の値を変更することもできます。これにより、**DEFAULT(col_name)** を使用してカラムを参照する生成されたカラム式の値を変更できます。このため、生成されたカラム式で **DEFAULT()** が使用されている場合、カラムの定義を変更する **ALTER TABLE** 操作によってテーブルが再構築されます。

13.1.9.3 ALTER TABLE の例

次に示すように作成されたテーブル **t1** から開始します:

```
CREATE TABLE t1 (a INTEGER, b CHAR(10));
```

テーブルの名前を **t1** から **t2** に変更するには、次のようにします。

```
ALTER TABLE t1 RENAME t2;
```

カラム **a** を **INTEGER** から **TINYINT NOT NULL** に変更し (名前は同じままにします)、またカラム **b** を **CHAR(10)** から **CHAR(20)** に変更し、さらにその名前を **b** から **c** に変更するには、次のようにします。

```
ALTER TABLE t2 MODIFY a TINYINT NOT NULL, CHANGE b c CHAR(20);
```

d という名前の新しい **TIMESTAMP** カラムを追加するには、次のようにします。

```
ALTER TABLE t2 ADD d TIMESTAMP;
```

カラム **d** にインデックスを、またカラム **a** に **UNIQUE** インデックスを追加するには、次のようにします。

```
ALTER TABLE t2 ADD INDEX (d), ADD UNIQUE (a);
```

カラム **c** を削除するには、次のようにします。

```
ALTER TABLE t2 DROP COLUMN c;
```

c という名前の新しい **AUTO_INCREMENT** 整数カラムを追加するには、次のようにします。

```
ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL AUTO_INCREMENT,
ADD PRIMARY KEY (c);
```

AUTO_INCREMENT カラムにはインデックスを設定する必要があるため **c** に (**PRIMARY KEY** として) インデックスを設定し、また主キーカラムは **NULL** にできないため **c** を **NOT NULL** として宣言します。

NDB テーブルの場合は、テーブルまたはカラムに使用されるストレージ型を変更することもできます。たとえば、次に示すように作成された **NDB** テーブルを考えてみます。

```
mysql> CREATE TABLE t1 (c1 INT) TABLESPACE ts_1 ENGINE NDB;
Query OK, 0 rows affected (1.27 sec)
```

このテーブルをディスクベースのストレージに変換するには、次の **ALTER TABLE** ステートメントを使用できます。

```
mysql> ALTER TABLE t1 TABLESPACE ts_1 STORAGE DISK;
Query OK, 0 rows affected (2.99 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

```
mysql> SHOW CREATE TABLE t1\G
***** 1. row *****
Table: t1
Create Table: CREATE TABLE `t1` (
  `c1` int(11) DEFAULT NULL
) /*!50100 TABLESPACE ts_1 STORAGE DISK */
ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.01 sec)
```

テーブルが最初に作成されたときにテーブルスペースが参照されている必要はありませんが、テーブルスペースは **ALTER TABLE** によって参照される必要があります。

```
mysql> CREATE TABLE t2 (c1 INT) ts_1 ENGINE NDB;
Query OK, 0 rows affected (1.00 sec)
```

```
mysql> ALTER TABLE t2 STORAGE DISK;
ERROR 1005 (HY000): Can't create table 'c.#sql-1750_3' (errno: 140)
mysql> ALTER TABLE t2 TABLESPACE ts_1 STORAGE DISK;
Query OK, 0 rows affected (3.42 sec)
Records: 0 Duplicates: 0 Warnings: 0
mysql> SHOW CREATE TABLE t2\G
***** 1. row *****
Table: t1
Create Table: CREATE TABLE `t2` (
  `c1` int(11) DEFAULT NULL
) /*!50100 TABLESPACE ts_1 STORAGE DISK */
ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.01 sec)
```

個々のカラムのストレージ型を変更するには、`ALTER TABLE ... MODIFY [COLUMN]` を使用できます。たとえば、次の `CREATE TABLE` ステートメントを使用して、2 つのカラムを含む「NDB Cluster ディスクデータ」テーブルを作成するとします：

```
mysql> CREATE TABLE t3 (c1 INT, c2 INT)
-> TABLESPACE ts_1 STORAGE DISK ENGINE NDB;
Query OK, 0 rows affected (1.34 sec)
```

カラム `c2` をディスクベースのストレージからインメモリーストレージに変更するには、次に示すように、`ALTER TABLE` ステートメントで使用されるカラム定義に `STORAGE MEMORY` 句を含めます。

```
mysql> ALTER TABLE t3 MODIFY c2 INT STORAGE MEMORY;
Query OK, 0 rows affected (3.14 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

同様の方法で `STORAGE DISK` を使用して、インメモリーカラムをディスクベースのカラムにすることができます。

カラム `c1` は、ディスクベースのストレージを使用します。これが (`CREATE TABLE` ステートメント内のテーブルレベルの `STORAGE DISK` 句によって決定される) テーブルのデフォルトであるためです。ただし、次の `SHOW CREATE TABLE` の出力に示すように、カラム `c2` はインメモリーストレージを使用します。

```
mysql> SHOW CREATE TABLE t3\G
***** 1. row *****
Table: t3
Create Table: CREATE TABLE `t3` (
  `c1` int(11) DEFAULT NULL,
  `c2` int(11) /*!50120 STORAGE MEMORY */ DEFAULT NULL
) /*!50100 TABLESPACE ts_1 STORAGE DISK */ ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.02 sec)
```

`AUTO_INCREMENT` カラムを追加すると、カラム値には、自動的にシーケンス番号が入力されます。MyISAM テーブルの場合は、`ALTER TABLE` の前に `SET INSERT_ID=value` を実行するか、または `AUTO_INCREMENT=value` テーブルオプションを使用することによって、最初のシーケンス番号を設定できます。

MyISAM テーブルでは、`AUTO_INCREMENT` カラムを変更しない場合、シーケンス番号は影響を受けません。`AUTO_INCREMENT` カラムを削除してから、別の `AUTO_INCREMENT` カラムを追加した場合、シーケンス番号は 1 から付け直されます。

レプリケーションを使用する場合、テーブルに `AUTO_INCREMENT` カラムを追加しても、レプリカとソースで同じ順序で行が生成されないことがあります。これが発生するのは、行が番号付けされる順序が、テーブルに使用される固有のストレージエンジンおよび行が挿入された順序に依存するためです。ソースとレプリカで順序が同じであることが重要な場合は、`AUTO_INCREMENT` 番号を割り当てる前に行を順序付ける必要があります。テーブル `t1` に `AUTO_INCREMENT` カラムを追加すると仮定した場合、次のステートメントは、`t1` と同一であるが、`AUTO_INCREMENT` カラムを含む新しいテーブル `t2` を生成します。

```
CREATE TABLE t2 (id INT AUTO_INCREMENT PRIMARY KEY)
SELECT * FROM t1 ORDER BY col1, col2;
```

ここでは、テーブル `t1` にカラム `col1` と `col2` が存在することを前提にしています。

この一連のステートメントでは、`AUTO_INCREMENT` カラムが追加された、`t1` と同一の新しいテーブル `t2` も生成されます：

```
CREATE TABLE t2 LIKE t1;
ALTER TABLE t2 ADD id INT AUTO_INCREMENT PRIMARY KEY;
INSERT INTO t2 SELECT * FROM t1 ORDER BY col1, col2;
```

重要

ソースとレプリカの両方で同じ順序を保証するには、`ORDER BY` 句で `t1` の all カラムを参照する必要があります。

`AUTO_INCREMENT` カラムを持つコピーを作成および移入するために使用する方法にかかわらず、最終手順は元のテーブルを削除してコピーの名前を変更することです。

```
DROP TABLE t1;
ALTER TABLE t2 RENAME t1;
```

13.1.10 ALTER TABLESPACE ステートメント

```
ALTER [UNDO] TABLESPACE tablespace_name
NDB only:
{ADD | DROP} DATAFILE 'file_name'
[INITIAL_SIZE [=] size]
[WAIT]
InnoDB and NDB:
[RENAME TO tablespace_name]
InnoDB only:
[AUTOEXTEND_SIZE [=] 'value']
[SET {ACTIVE | INACTIVE}]
[ENCRYPTION [=] {'Y' | 'N'}]
InnoDB and NDB:
[ENGINE [=] engine_name]
Reserved for future use:
[ENGINE_ATTRIBUTE [=] 'string']
```

このステートメントは、NDB および InnoDB テーブルスペースで使用されます。これを使用して、新しいデータファイルを NDB テーブルスペースに追加したり、NDB テーブルスペースからデータファイルを削除できます。また、「NDB Cluster ディスクデータ」テーブルスペースの名前変更、InnoDB 一般テーブルスペースの名前変更、InnoDB 一般テーブルスペースの暗号化、または InnoDB undo テーブルスペースをアクティブまたは非アクティブとしてマークするためにも使用できます。

MySQL 8.0.14 で導入された UNDO キーワードは、InnoDB undo テーブルスペースをアクティブまたは非アクティブとしてマークするために SET {ACTIVE | INACTIVE} 句とともに使用されます。詳細は、[セクション15.6.3.4「undo テーブルスペース」](#)を参照してください。

ADD DATAFILE バリエーションを使用すると、INITIAL_SIZE 句を使用して NDB 「ディスクデータ」テーブルスペースの初期サイズを指定できます。size はバイト単位で測定され、デフォルト値は 134217728 (128 MB) です。オプションで、my.cnf で使用されているものと同様に、size の後に一文字の略称を付けることもできます。一般に、これは M (M バイト) または G (G バイト) のどちらかの文字です。

32 ビットシステム上では、INITIAL_SIZE のサポートされる最大値は 4294967296 (4G バイト) です。(Bug #29186)

INITIAL_SIZE は、CREATE TABLESPACE と同様に明示的に丸められます。

データファイルが作成されると、そのサイズは変更できませんが、追加の ALTER TABLESPACE ... ADD DATAFILE ステートメントを使用して NDB テーブルスペースにデータファイルを追加できます。

ALTER TABLESPACE ... ADD DATAFILE を ENGINE = NDB とともに使用すると、各クラスターデータノードにデータファイルが作成されますが、INFORMATION_SCHEMA.FILES テーブルには 1 つの行のみが生成されます。詳細は、このテーブルおよび [セクション23.5.10.1「NDB Cluster ディスクデータオブジェクト」](#) の説明を参照してください。ADD DATAFILE は、InnoDB テーブルスペースではサポートされていません。

ALTER TABLESPACE で DROP DATAFILE を使用すると、NDB テーブルスペースからデータファイル'*file_name*'が削除されます。いずれかのテーブルが使用しているテーブルスペースからはデータファイルを削除できません。つまり、そのデータファイルが空である (エクステンツが使用されていない) ことが必要です。[セクション23.5.10.1「NDB Cluster ディスクデータオブジェクト」](#)を参照してください。さらに、削除されるデータファイルはすべて、CREATE TABLESPACE または ALTER TABLESPACE で以前にそのテーブルスペースに追加されている必要があります。DROP DATAFILE は、InnoDB テーブルスペースではサポートされていません。

WAIT は解析されますが、それ以外は無視されます。これは将来の拡張のために用意されています。

テーブルスペースで使用されるストレージエンジンを指定する ENGINE 句は非推奨になりました。将来のリリースで削除される予定です。テーブルスペース記憶域エンジンはデータディクショナリによって認識されるため、ENGINE 句は廃止されています。ストレージエンジンが指定されている場合は、データディクショナリに定義されているテーブルスペースストレージエンジンと一致する必要があります。NDB テーブルスペースと互換性のある engine_name の値は、NDB および NDBCLUSTER のみです。

RENAME TO 操作は、autocommit の設定に関係なく、autocommit モードで暗黙的に実行されます。

テーブルスペースに存在するテーブルに対して LOCK TABLES または FLUSH TABLES WITH READ LOCK が有効になっている間は、RENAME TO 操作を実行できません。

排他的 [metadata locks](#) は、テーブルスペースの名前の変更中に一般的なテーブルスペースに存在するテーブルに対して取得されるため、同時 DDL が回避されます。同時 DML がサポートされています。

InnoDB 一般テーブルスペースの名前を変更するには、[CREATE TABLESPACE](#) 権限が必要です。

[AUTOEXTEND_SIZE](#) オプションは、一杯になったときに InnoDB がテーブルスペースのサイズを拡張する量を定義します。MySQL 8.0.23 で導入されました。設定は 4MB の倍数である必要があります。デフォルト設定は 0 で、暗黙的なデフォルト動作に従ってテーブルスペースが拡張されます。詳細は、[セクション15.6.3.9「テーブルスペースの AUTOEXTEND_SIZE 構成」](#)を参照してください。

[ENCRYPTION](#) 句は、InnoDB 一般テーブルスペースまたは [mysql](#) システムテーブルスペースのページレベルのデータ暗号化を有効または無効にします。一般テーブルスペースの暗号化サポートは、MySQL 8.0.13 で導入されました。[mysql](#) システムテーブルスペースの暗号化サポートは、MySQL 8.0.16 で導入されました。

暗号化を有効にする前に、キーリングプラグインをインストールして構成する必要があります。

MySQL 8.0.16 では、[table_encryption_privilege_check](#) 変数が有効になっている場合、[default_table_encryption](#) の設定とは異なる [ENCRYPTION](#) 句の設定を使用して一般的なテーブルスペースを変更するには、[TABLE_ENCRYPTION_ADMIN](#) 権限が必要です。

テーブルスペース内のいずれかのテーブルが [DEFAULT ENCRYPTION='N'](#) で定義されたスキーマに属している場合、一般テーブルスペースの暗号化の有効化は失敗します。同様に、一般テーブルスペースのいずれかのテーブルが [DEFAULT ENCRYPTION='Y'](#) で定義されたスキーマに属している場合、暗号化の無効化は失敗します。[DEFAULT ENCRYPTION](#) スキーマオプションは、MySQL 8.0.16 で導入されました。

一般的なテーブルスペースで実行される [ALTER TABLESPACE](#) ステートメントに [ENCRYPTION](#) 句が含まれていない場合、[default_table_encryption](#) の設定に関係なく、テーブルスペースは現在の暗号化ステータスを保持します。

一般テーブルスペースまたは [mysql](#) システムテーブルスペースが暗号化されると、テーブルスペースに存在するすべてのテーブルが暗号化されます。同様に、暗号化されたテーブルスペースに作成されたテーブルも暗号化されます。

[INPLACE](#) アルゴリズムは、一般テーブルスペースまたは [mysql](#) システムテーブルスペースの [ENCRYPTION](#) 属性を変更するときに使用されます。[INPLACE](#) アルゴリズムでは、テーブルスペースに存在するテーブルに対する同時 DML が許可されます。同時 DDL はブロックされます。

詳細は、[セクション15.13「InnoDB 保存データ暗号化」](#)を参照してください。

[ENGINE_ATTRIBUTE](#) オプション (MySQL 8.0.21 の時点で使用可能) を使用して、プライマリストレージエンジンのテーブルスペース属性を指定します。このオプションは、将来の使用のために予約されています。

許可される値は、有効な [JSON](#) ドキュメントまたは空の文字列 ("") を含む文字列リテラルです。無効な [JSON](#) が拒否されました。

```
ALTER TABLESPACE ts1 ENGINE_ATTRIBUTE={'"key": "value"};
```

[ENGINE_ATTRIBUTE](#) の値は、エラーなしで繰り返すことができます。この場合、最後に指定した値が使用されます。

[ENGINE_ATTRIBUTE](#) 値はサーバーによってチェックされず、テーブルストレージエンジンが変更されたときにもクリアされません。

JSON 属性値の個々の要素を変更することはできません。追加または置換できるのは属性のみです。

13.1.11 ALTER VIEW ステートメント

```
ALTER
[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
[DEFINER = user]
[SQL SECURITY { DEFINER | INVOKER }]
VIEW view_name [(column_list)]
AS select_statement
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

このステートメントは、ビューの定義を変更します。このビューは存在する必要があります。構文は、[CREATE VIEW](#) の場合と似ています ([セクション13.1.23「CREATE VIEW ステートメント」](#)を参照)。このステートメントに

は、このビューに対する [CREATE VIEW](#) および [DROP](#) 権限と、[SELECT](#) ステートメントで参照される各カラムに対する何らかの権限が必要です。[ALTER VIEW](#) は、定義者または [SET_USER_ID](#) 権限 (または非推奨の [SUPER](#) 権限) を持つユーザーにのみ許可されます。

13.1.12 CREATE DATABASE ステートメント

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
  [create_option] ...

create_option: [DEFAULT] {
  CHARACTER SET [=] charset_name
  | COLLATE [=] collation_name
  | ENCRYPTION [=] {'Y' | 'N'}
}
```

[CREATE DATABASE](#) は、指定された名前を持つデータベースを作成します。このステートメントを使用するには、このデータベースに対する [CREATE](#) 権限が必要です。[CREATE SCHEMA](#) は [CREATE DATABASE](#) のシノニムです。

そのデータベースが存在するときに [IF NOT EXISTS](#) を指定しなかった場合は、エラーが発生します。

[CREATE DATABASE](#) は、アクティブな [LOCK TABLES](#) ステートメントを持つセッション内では許可されません。

各 [create_option](#) は、データベース特性を指定します。データベース特性はデータディクショナリに格納されます。

- [CHARACTER SET](#) オプションは、デフォルトのデータベース文字セットを指定します。[COLLATE](#) オプションは、デフォルトのデータベース照合順序を指定します。文字セットおよび照合順序名の詳細は、[第10章「文字セット、照合順序、Unicode」](#) を参照してください。

使用可能な文字セットと照合順序を確認するには、それぞれ [SHOW CHARACTER SET](#) ステートメントと [SHOW COLLATION](#) ステートメントを使用します。[セクション13.7.7.3「SHOW CHARACTER SET ステートメント」](#) および [セクション13.7.7.4「SHOW COLLATION ステートメント」](#) を参照してください。

- MySQL 8.0.16 で導入された [ENCRYPTION](#) オプションは、データベースで作成されたテーブルによって継承されるデフォルトのデータベース暗号化を定義します。許可される値は、['Y'](#) (暗号化有効) および ['N'](#) (暗号化無効) です。[ENCRYPTION](#) オプションが指定されていない場合、[default_table_encryption](#) システム変数の値によってデフォルトのデータベース暗号化が定義されます。[table_encryption_privilege_check](#) システム変数が有効になっている場合、[default_table_encryption](#) 設定とは異なるデフォルトの暗号化設定を指定するには、[TABLE_ENCRYPTION_ADMIN](#) 権限が必要です。詳細は、[スキーマおよび一般テーブルスペースの暗号化デフォルトの定義](#) を参照してください。

MySQL でのデータベースは、そのデータベース内のテーブルに対応するファイルを含むディレクトリとして実装されます。最初の作成時にはデータベースにテーブルがないため、[CREATE DATABASE](#) ステートメントでは MySQL データディレクトリの下にディレクトリのみが作成されます。許可されるデータベース名のルールは、[セクション9.2「スキーマオブジェクト名」](#) に示されています。データベース名に特殊文字が含まれている場合は、[セクション9.2.4「識別子とファイル名のマッピング」](#) で説明されているように、その文字のエンコードされたバージョンがデータベースディレクトリの名前に含まれます。

データディレクトリの下にディレクトリを手動で作成することによるデータベースディレクトリの作成 ([mkdir](#) などを使用) は、MySQL 8.0 ではサポートされていません。

データベースを作成する場合は、サーバーでディレクトリとその中のファイルを管理します。データベースディレクトリおよびファイルを直接操作すると、不整合や予期しない結果が発生する可能性があります。

MySQL にはデータベース数の制限はありません。ベースとなるファイルシステムによっては、ディレクトリ数に制限がある場合があります。

[mysqldadmin](#) プログラムを使用してデータベースを作成することもできます。[セクション4.5.2「mysqldadmin — MySQL Server 管理プログラム」](#) を参照してください。

13.1.13 CREATE EVENT ステートメント

```
CREATE
  [DEFINER = user]
  EVENT
```

```
[IF NOT EXISTS]
event_name
ON SCHEDULE schedule
[ON COMPLETION [NOT] PRESERVE]
[ENABLE | DISABLE | DISABLE ON SLAVE]
[COMMENT 'string']
DO event_body;

schedule: {
  AT timestamp [+ INTERVAL interval] ...
  | EVERY interval
  [STARTS timestamp [+ INTERVAL interval] ...]
  [ENDS timestamp [+ INTERVAL interval] ...]
}

interval:
quantity {YEAR | QUARTER | MONTH | DAY | HOUR | MINUTE |
WEEK | SECOND | YEAR_MONTH | DAY_HOUR | DAY_MINUTE |
DAY_SECOND | HOUR_MINUTE | HOUR_SECOND | MINUTE_SECOND}
```

このステートメントは、新しいイベントを作成してスケジュールします。イベントスケジューラが有効になっていないかぎり、イベントは実行されません。イベントスケジューラのステータスをチェックし、必要に応じてそれを有効にする方法については、[セクション25.4.2「イベントスケジューラの構成」](#)を参照してください。

`CREATE EVENT` には、イベントが作成されるスキーマに対する `EVENT` 権限が必要です。 `DEFINER` 句が存在する場合、[セクション25.6「ストアオブジェクトのアクセス制御」](#)で説明されているように、必要な権限は `user` の値によって異なります。

有効な `CREATE EVENT` ステートメントの最小要件は次のとおりです。

- キーワード `CREATE EVENT` に加えて、データベーススキーマ内のイベントを一意に識別するイベント名。
- イベントが実行される時期と頻度を決定する `ON SCHEDULE` 句。
- イベントによって実行される SQL ステートメントを含む `DO` 句。

最小限の `CREATE EVENT` ステートメントの例を次に示します。

```
CREATE EVENT myevent
ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 HOUR
DO
  UPDATE myschema.mytable SET mycol = mycol + 1;
```

前のステートメントは、`myevent` という名前のイベントを作成します。このイベントは、`myschema.mytable` テーブルの `mycol` カラムの値を 1 増分する SQL ステートメントを実行することによって (その作成の 1 時間後に) 1 回実行されます。

`event_name` は、最大長が 64 文字の有効な MySQL 識別子である必要があります。イベント名では大/小文字が区別されないため、`myevent` と `MyEvent` という名前のイベントを同じスキーマに含めることはできません。一般に、イベント名を管理するルールは、ストアルーチンの名前の場合と同じです。[セクション9.2「スキーマオブジェクト名」](#)を参照してください。

イベントはスキーマに関連付けられています。`event_name` の一部としてスキーマが示されていない場合は、デフォルトの (現在の) スキーマと見なされます。イベントを特定のスキーマ内に作成するには、`schema_name.event_name` 構文を使用して、そのイベント名をスキーマで修飾します。

`DEFINER` 句は、イベントの実行時にアクセス権限を確認するときに使用される MySQL アカウントを指定します。`DEFINER` 句が存在する場合、`user` 値は `'user_name'@'host_name'`、`CURRENT_USER` または `CURRENT_USER()` として指定された MySQL アカウントである必要があります。許可される `user` 値は、[セクション25.6「ストアオブジェクトのアクセス制御」](#)で説明されているように、保持する権限によって異なります。イベントセキュリティの詳細は、そのセクションも参照してください。

`DEFINER` 句を省略すると、デフォルトの定義者は `CREATE EVENT` ステートメントを実行するユーザーになります。これは、明示的に `DEFINER = CURRENT_USER` を指定するのと同じです。

イベント本体内では、`CURRENT_USER` 関数は、`DEFINER` ユーザーであるイベント実行時の権限のチェックに使用されるアカウントを返します。イベント内のユーザー監査については、[セクション6.2.22「SQL ベースのアカウントアクティビティ監査」](#)を参照してください。

CREATE EVENT での IF NOT EXISTS には、CREATE TABLE での場合と同じ意味があります。event_name という名前のイベントが同じスキーマ内にすでに存在する場合、アクションは実行されず、エラーも発生しません。(ただし、このような場合は警告が生成されます。)

ON SCHEDULE 句は、そのイベントに対して定義された event_body を繰り返す時期、頻度、および期間を決定します。この句は、次の 2 つの形式のいずれかを取ります。

- 1 回限りのイベントには、AT timestamp が使用されます。これは、そのイベントが timestamp で指定された日付と時間に 1 回だけ実行されることを指定します。この値は、日付と時間の両方を含んでいるか、または datetime 値に解決される式である必要があります。この目的には、DATETIME または TIMESTAMP 型のどちらかの値を使用できます。日付が過去の日付である場合は、次に示すように、警告が発生します。

```
mysql> SELECT NOW();
+-----+
| NOW() |
+-----+
| 2006-02-10 23:59:01 |
+-----+
1 row in set (0.04 sec)

mysql> CREATE EVENT e_totals
-> ON SCHEDULE AT '2006-02-10 23:59:00'
-> DO INSERT INTO test.totals VALUES (NOW());
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Note
Code: 1588
Message: Event execution time is in the past and ON COMPLETION NOT
PRESERVE is set. The event was dropped immediately after
creation.
```

どのような理由であれ、それ自体が無効な CREATE EVENT ステートメントはエラーで失敗します。

現在の日付と時間を指定するには、CURRENT_TIMESTAMP を使用できます。このような場合、イベントは、作成されるとすぐに機能します。

現在の日付と時間を基準にした将来のある時点(「今から 3 週間後」というフレーズで表される時点など)に発生するイベントを作成するには、オプションの句 + INTERVAL interval を使用できます。interval 部分は数量と時間単位で構成され、時間間隔で説明されている構文ルールに従いますが、イベントの定義時にマイクロ秒を含む単位キーワードは使用できません。一部の間隔型では、複合の時間単位を使用できます。たとえば、「2 分と 10 秒」は、+ INTERVAL '2:10' MINUTE_SECOND として表すことができます。

また、間隔を組み合わせることもできます。たとえば、AT CURRENT_TIMESTAMP + INTERVAL 3 WEEK + INTERVAL 2 DAY は、「今から 3 週間と 2 日後」と同等です。このような句の各部分は、+ INTERVAL で始まる必要があります。

- アクションを定期的に繰り返すには、EVERY 句を使用します。EVERY キーワードのあとに、前の AT キーワードの説明に示されている interval を指定します。(EVERY では + INTERVAL は使用されません。)たとえば、EVERY 6 WEEK は「6 週間ごと」を示します。

EVERY 句では + INTERVAL 句は許可されていませんが、+ INTERVAL 内で許可されているのと同じ複合の時間単位を使用できます。

EVERY 句には、オプションの STARTS 句を含めることができます。STARTS のあとに、このアクションがいつ繰り返しを開始するかを示す timestamp 値を指定します。また、+ INTERVAL interval を使用して、「今からの」時間を指定することもできます。たとえば、EVERY 3 MONTH STARTS CURRENT_TIMESTAMP + INTERVAL 1 WEEK は、「今から 1 週間後に開始して 3 か月ごと」を示します。同様に、「今から 6 時間と 15 分後から開始して 2 週ごと」を、EVERY 2 WEEK STARTS CURRENT_TIMESTAMP + INTERVAL '6:15' HOUR_MINUTE として表すことができます。STARTS を指定しないことは、STARTS CURRENT_TIMESTAMP を使用することと同じです。つまり、イベントに対して指定されたアクションは、そのイベントが作成されるとただちに繰り返しを開始します。

EVERY 句には、オプションの ENDS 句を含めることができます。ENDS キーワードのあとに、このイベントがいつ繰り返しを停止するかを MySQL に指示する timestamp 値を指定します。また、ENDS とともに + INTERVAL

`interval` を使用することもできます。たとえば、`EVERY 12 HOUR STARTS CURRENT_TIMESTAMP + INTERVAL 30 MINUTE ENDS CURRENT_TIMESTAMP + INTERVAL 4 WEEK` は、「今から 30 分後に開始し、今から 4 週間後に終了するまで 12 時間ごと」と同等です。`ENDS` を使用しないことは、このイベントがいつまでも実行を続けることを示します。

`ENDS` は、複合の時間単位に対して `STARTS` と同じ構文をサポートします。

`EVERY` 句では、`STARTS` または `ENDS`、あるいはその両方を使用できます。また、どちらも使用しないことも可能です。

繰り返しイベントがスケジュール間隔内に終了しない場合は、イベントの複数のインスタンスが同時に実行される可能性があります。これが好ましくない場合は、同時インスタンスを回避するためのメカニズムを設けてください。たとえば、`GET_LOCK()` 関数や、行またはテーブルのロックを使用できます。

`ON SCHEDULE` 句では、組み込みの MySQL 関数やユーザー変数を含む式を使用して、そこに含まれているすべての `timestamp` または `interval` 値を取得できます。このような式でストアードファンクションやユーザー定義関数を使用したり、テーブル参照を使用したりすることはできません。ただし、`SELECT FROM DUAL` は使用できます。これは、`CREATE EVENT` ステートメントと `ALTER EVENT` ステートメントの両方に当てはまります。このような場合のストアードファンクション、ユーザー定義関数、およびテーブルへの参照は明確に禁止されており、エラーで失敗します (Bug #22830 を参照してください)。

`ON SCHEDULE` 句の時間は、現在のセッションの `time_zone` 値を使用して解釈されます。これがイベントのタイムゾーン、つまり、イベントのスケジュールリングに使用され、イベントが実行されるとそのイベント内で有効になるタイムゾーンになります。これらの時間は UTC に変換され、イベントタイムゾーンとともに内部的に格納されます。これにより、サーバータイムゾーンまたはサマータイムの影響に対し生じた変更とは無関係に、定義されたとおりにイベントの実行を処理できます。イベントの時間の表現の詳細は、[セクション 25.4.4 「イベントメタデータ」](#) を参照してください。[セクション 13.7.7.18 「SHOW EVENTS ステートメント」](#) および [セクション 26.14 「INFORMATION_SCHEMA EVENTS テーブル」](#) も参照してください。

通常は、イベントの期限が切れると、そのイベントはただちに削除されます。この動作は、`ON COMPLETION PRESERVE` を指定することによってオーバーライドできます。`ON COMPLETION NOT PRESERVE` を使用すると、単にデフォルトの非持続性の動作が明示的になるだけです。

`DISABLE` キーワードを使用すると、イベントは作成するが、それがアクティブにならないようにすることができます。あるいは、`ENABLE` を使用して、デフォルトステータス (アクティブ) を明示的にすることもできます。これは、`ALTER EVENT` と組み合わせるともっとも有効です ([セクション 13.1.3 「ALTER EVENT ステートメント」](#) を参照してください)。

3 番目の値は、`ENABLE` または `DISABLE` のかわりに表示されることもあります。`DISABLE ON SLAVE` は、イベントがレプリケーションソースサーバー上で作成され、レプリカにレプリケートされたが、レプリカ上では実行されなかったことを示すために、レプリカ上のイベントのステータスに設定されます。[セクション 17.5.1.16 「呼び出される機能のレプリケーション」](#) を参照してください。

`COMMENT` 句を使用して、イベントに対するコメントを指定できます。`comment` には、イベントの説明に使用する、最大 64 文字の任意の文字列を指定できます。コメントテキストは文字列リテラルであるため、引用符で囲む必要があります。

`DO` 句は、イベントによって実行されるアクションを指定するものであり、SQL ステートメントで構成されます。ストアードルーチンで使用できる有効な MySQL ステートメントのほぼすべてを、スケジュールされたイベントのアクションステートメントとしても使用できます。([セクション 25.8 「ストアードプログラムの制約」](#) を参照してください。) たとえば、次のイベント `e_hourly` は、`sessions` テーブルのすべての行を 1 時間に 1 回削除します。ここで、このテーブルは `site_activity` スキーマの一部です。

```
CREATE EVENT e_hourly
ON SCHEDULE
  EVERY 1 HOUR
COMMENT 'Clears out sessions table each hour.'
DO
  DELETE FROM site_activity.sessions;
```

MySQL は、イベントが作成または変更されたときの有効な `sql_mode` システム変数の設定を格納し、イベントが実行を開始したときの現在のサーバー SQL モードには関係なく、常にそのイベントを強制的にこの設定で実行します。

DO 句に ALTER EVENT ステートメントを含む CREATE EVENT ステートメントは成功したように見えます。ただし、結果として得られるスケジュールされたイベントをサーバーが実行しようとする、その実行はエラーで失敗します。

注記

単に結果セットを返す SELECT や SHOW などのステートメントは、イベントで使用されても何の効果もありません。これらのステートメントからの出力は MySQL モニターに送信されず、またどこにも格納されません。ただし、結果を格納する SELECT ... INTO や INSERT INTO ... SELECT などのステートメントは使用できます。(後者の例については、このセクションにある次の例を参照してください。)

イベントが属するスキーマは、DO 句でのテーブル参照のためのデフォルトスキーマです。ほかのスキーマでのテーブルへの参照はすべて、正しいスキーマ名で修飾する必要があります。

次に示すように、ストアルーチンと同様に、BEGIN および END キーワードを使用して DO 句で複合ステートメントの構文を使用できます。

```
delimiter |
CREATE EVENT e_daily
ON SCHEDULE
EVERY 1 DAY
COMMENT 'Saves total number of sessions then clears the table each day'
DO
BEGIN
INSERT INTO site_activity.totals (time, total)
SELECT CURRENT_TIMESTAMP, COUNT(*)
FROM site_activity.sessions;
DELETE FROM site_activity.sessions;
END |
delimiter ;
```

この例では、delimiter コマンドを使用して、ステートメント区切り文字を変更します。 [セクション25.1「ストアドプログラムの定義」](#)を参照してください。

イベントでは、ストアルーチンで使用されているような、より複雑な複合ステートメントを使用できます。この例では、ローカル変数、エラーハンドラ、およびフロー制御構造構文を使用しています。

```
delimiter |
CREATE EVENT e
ON SCHEDULE
EVERY 5 SECOND
DO
BEGIN
DECLARE v INTEGER;
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION BEGIN END;

SET v = 0;

WHILE v < 5 DO
INSERT INTO t1 VALUES (0);
UPDATE t2 SET s1 = s1 + 1;
SET v = v + 1;
END WHILE;
END |
delimiter ;
```

イベントに、またはイベントから直接パラメータを渡す方法はありませんが、パラメータを持つストアルーチンをイベント内で呼び出すことは可能です。

```
CREATE EVENT e_call_myproc
ON SCHEDULE
AT CURRENT_TIMESTAMP + INTERVAL 1 DAY
DO CALL myproc(5, 27);
```


イベント定義者がグローバルシステム変数を設定するのに十分な権限を持っている場合 ([セクション5.1.9.1「システム変数権限」](#) を参照)、イベントはグローバル変数の読取りおよび書き込みを実行できます。このような権限を付与するには不正利用の可能性があるので、十分に注意する必要があります。

一般に、ストアルーチンで有効なすべてのステートメントを、イベントによって実行されるアクションステートメントに使用できます。ストアルーチン内で許可されるステートメントの詳細は、[セクション25.2.1「ストアルーチンの構文」](#) を参照してください。ストアルーチンの一部としてイベントを作成できますが、イベントを別のイベントで作成することはできません。

13.1.14 CREATE FUNCTION ステートメント

`CREATE FUNCTION` ステートメントは、ストアドファンクションやユーザー定義関数 (UDF) を作成するために使用されます。

- ストアドファンクションの作成については、[セクション13.1.17「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」](#) を参照してください。
- ユーザー定義関数の作成については、[セクション13.7.4.1「ユーザー定義関数用の CREATE FUNCTION ステートメント」](#) を参照してください。

13.1.15 CREATE INDEX ステートメント

```
CREATE [UNIQUE | FULLTEXT | SPATIAL] INDEX index_name
[index_type]
ON tbl_name (key_part,...)
[index_option]
[algorithm_option | lock_option] ...

key_part: {col_name [(length)] | (expr)} [ASC | DESC]

index_option: {
    KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'
  | {VISIBLE | INVISIBLE}
  | ENGINE_ATTRIBUTE [=] 'string'
  | SECONDARY_ENGINE_ATTRIBUTE [=] 'string'
}

index_type:
    USING {BTREE | HASH}

algorithm_option:
    ALGORITHM [=] {DEFAULT | INPLACE | COPY}

lock_option:
    LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE}
```

通常、テーブル上のすべてのインデックスは、そのテーブル自体が `CREATE TABLE` で作成された時点で作成します。[セクション13.1.20「CREATE TABLE ステートメント」](#) を参照してください。このガイドラインは、主キーによってデータファイル内の行の物理配列が決定される InnoDB テーブルの場合に特に重要です。`CREATE INDEX` では、既存のテーブルにインデックスを追加できます。

`CREATE INDEX` は、インデックスを作成するために `ALTER TABLE` ステートメントにマップされます。[セクション13.1.9「ALTER TABLE ステートメント」](#) を参照してください。`CREATE INDEX` を使用して `PRIMARY KEY` を作成することはできません。代わりに `ALTER TABLE` を使用します。インデックスの詳細は、[セクション8.3.1「MySQL のインデックスの使用の仕組み」](#) を参照してください。

InnoDB は、仮想カラムのセカンダリインデックスをサポートしています。詳細は、[セクション13.1.20.9「セカンダリインデックスと生成されたカラム」](#) を参照してください。

`innodb_stats_persistent` 設定が有効になっている場合は、InnoDB テーブル上でインデックスを作成したあと、そのテーブルに対して `ANALYZE TABLE` ステートメントを実行します。

MySQL 8.0.17 以降、`key_part` 仕様の `expr` では、`(CAST json_expression AS type ARRAY)` の形式を使用して JSON カラムに複数値インデックスを作成できます。[複数値インデックス](#) を参照してください。

(*key_part1*, *key_part2*, ...) 形式のインデックス指定では、複数のキー部分を持つインデックスが作成されます。インデックスキー値は、指定されたキー部分の値を連結することによって形成されます。たとえば、(*col1*, *col2*, *col3*) では、*col1*、*col2* および *col3* の値で構成されるインデックスキーを持つ複数カラムインデックスを指定します。

key_part 仕様の末尾には、ASC または DESC を使用して、インデックス値を昇順または降順のどちらで格納するかを指定できます。順序指定子が指定されていない場合、デフォルトは昇順です。ASC および DESC は、HASH インデックスには使用できません。ASC および DESC は、複数值インデックスでもサポートされていません。MySQL 8.0.12 では、SPATIAL インデックスに対して ASC および DESC は許可されていません。

次の各セクションでは、CREATE INDEX ステートメントの様々な側面について説明します：

- [カラム接頭辞のキー部分](#)
- [機能キー部品](#)
- [一意インデックス](#)
- [全文インデックス](#)
- [複数值インデックス](#)
- [空間インデックス](#)
- [インデックスオプション](#)
- [テーブルのコピーおよびロックのオプション](#)

カラム接頭辞のキー部分

文字列カラムの場合、*col_name*(*length*) 構文を使用してインデックス接頭辞の長さを指定し、カラム値の先頭部分のみを使用するインデックスを作成できます：

- 接頭辞は、CHAR, VARCHAR, BINARY および VARBINARY のキー部分に指定できます。
- 接頭辞は、BLOB および TEXT のキー部分に指定する必要があります。また、BLOB カラムおよび TEXT カラムは、InnoDB、MyISAM および BLACKHOLE テーブルに対してのみインデックス付けできます。
- 接頭辞 *limits* はバイト単位で測定されます。ただし、CREATE TABLE、ALTER TABLE および CREATE INDEX ステートメントのインデックス指定の接頭辞 *lengths* は、非バイナリ文字列型 (CHAR, VARCHAR, TEXT) の場合は文字数として解釈され、バイナリ文字列型 (BINARY, VARBINARY, BLOB) の場合はバイト数として解釈されます。マルチバイト文字セットを使用する非バイナリ文字列カラムに接頭辞の長さを指定する場合は、これを考慮してください。

プリフィクスのサポートやプリフィクスの長さ (サポートされている場合) は、ストレージエンジンに依存します。たとえば、REDUNDANT または COMPACT の行形式を使用する InnoDB テーブルでは、接頭辞の長さは最大 767 バイトです。DYNAMIC または COMPRESSED の行形式を使用する InnoDB テーブルでは、接頭辞の長さの制限は 3072 バイトです。MyISAM テーブルの場合、接頭辞の長さの制限は 1000 バイトです。NDB ストレージエンジンは接頭辞をサポートしていません ([セクション 23.1.7.6 「NDB Cluster でサポートされない機能または欠落している機能」](#) を参照)。

指定したインデックス接頭辞がカラムの最大データ型サイズを超える場合、CREATE INDEX は次のようにインデックスを処理します：

- 一意でないインデックスの場合は、エラーが発生するか (厳密な SQL モードが有効な場合)、インデックスの長さが最大カラムデータ型サイズ内になるように縮小され、警告が生成されます (厳密な SQL モードが有効でない場合)。
- 一意インデックスの場合、インデックスの長さを短くすると、指定した一意性要件を満たさない一意でないエントリの挿入が可能になるため、SQL モードに関係なくエラーが発生します。

次のステートメントは、*name* カラムの最初の 10 文字を使用してインデックスを作成します (*name* にバイナリ以外の文字列型があると想定しています)：

```
CREATE INDEX part_of_name ON customer (name(10));
```

通常、カラムの名前が最初の 10 文字と異なる場合、このインデックスを使用して実行されるルックアップは、*name* カラム全体から作成されたインデックスを使用する場合よりも遅くなることはありません。また、インデックスにカ

ラムプリフィクスを使用するとインデックスファイルをはるかに小さくできるため、多くのディスク領域が節約されるだけでなく、**INSERT** 操作も高速化される可能性があります。

機能キ一部分

「normal」インデックスは、カラム値またはカラム値の接頭辞をインデックス付けします。たとえば、次のテーブルでは、特定の **t1** 行のインデックスエントリに、最初の 10 文字で構成される完全な **col1** 値と **col2** 値の接頭辞が含まれています：

```
CREATE TABLE t1 (  
  col1 VARCHAR(10),  
  col2 VARCHAR(20),  
  INDEX (col1, col2(10))  
);
```

MySQL 8.0.13 以上では、カラムまたはカラムの接頭辞値ではなく式の値をインデックス付けする関数キ一部分がサポートされています。関数キ一部分を使用すると、テーブルに直接格納されない値のインデックス付けが可能になります。例：

```
CREATE TABLE t1 (col1 INT, col2 INT, INDEX func_index ((ABS(col1))));  
CREATE INDEX idx1 ON t1 ((col1 + col2));  
CREATE INDEX idx2 ON t1 ((col1 + col2), (col1 - col2), col1);  
ALTER TABLE t1 ADD INDEX ((col1 * 40) DESC);
```

複数のキ一部分を持つインデックスでは、非機能キ一部分と機能キ一部分を混在させることができます。

ASC および **DESC** は、機能キ一部分でサポートされています。

機能キ一部分は、次のルールに従う必要があります。キ一部分定義に許可されていない構成が含まれている場合は、エラーが発生します。

- インデックス定義では、式をカッコで囲み、カラムまたはカラムの接頭辞と区別します。たとえば、これは許可されており、式はカッコで囲まれています：

```
INDEX ((col1 + col2), (col3 - col4))
```

これによりエラーが発生します。式はカッコで囲まれません：

```
INDEX (col1 + col2, col3 - col4)
```

- 関数キ一部分は、カラム名のみで構成できません。たとえば、これは許可されていません：

```
INDEX ((col1), (col2))
```

かわりに、キ一部分を機能しないキ一部分としてカッコなしで記述します：

```
INDEX (col1, col2)
```

- 関数キ一部分式はカラム接頭辞を参照できません。回避策については、このセクションで後述する **SUBSTRING()** および **CAST()** の説明を参照してください。
- 外部キ仕様では、機能キ一部分は許可されません。

CREATE TABLE ... LIKE の場合、宛先テーブルは元のテーブルの機能キ一部分を保持します。

関数インデックスは非表示の仮想生成カラムとして実装され、次のような影響があります：

- 各関数キ一部分は、テーブルのカラムの合計数に対する制限に対してカウントされます。[セクション8.4.7「テーブルカラム数と行サイズの制限」](#)を参照してください。
- 機能キ一部分は、生成されたカラムに適用されるすべての制限を継承します。例：
 - 関数キ一部分には、生成されたカラムに許可された関数のみが許可されます。
 - サブクエリー、パラメータ、変数、ストアドファンクションおよびユーザー定義関数は使用できません。

適用可能な制限の詳細は、[セクション13.1.20.8「CREATE TABLE および生成されるカラム」](#) および [セクション13.1.9.2「ALTER TABLE および生成されるカラム」](#)を参照してください。

- ・ 仮想生成カラム自体に記憶域は必要ありません。インデックス自体は、他のインデックスと同様に記憶領域を占有します。

UNIQUE は、関数キー部分を含むインデックスに対してサポートされています。ただし、主キーに機能キー部分を含めることはできません。主キーでは、生成されたカラムを格納する必要がありますが、機能キー部分は、格納された生成カラムではなく、仮想生成カラムとして実装されます。

SPATIAL および **FULLTEXT** インデックスには、関数キー部分を含めることはできません。

テーブルに主キーが含まれていない場合、**InnoDB** は最初の **UNIQUE NOT NULL** インデックスを主キーに自動的に昇格します。これは、関数キー部分を持つ **UNIQUE NOT NULL** インデックスではサポートされません。

インデックスが重複している場合は、非関数インデックスで警告が発生します。関数キー部分を含むインデックスには、この機能はありません。

関数キー部分によって参照されるカラムを削除するには、最初にインデックスを削除する必要があります。それ以外の場合は、エラーが発生します。

非機能キー部分は接頭辞の長さの指定をサポートしていますが、これは機能キー部分では不可能です。解決策は、**SUBSTRING()**(または、このセクションの後半で説明する **CAST()**) を使用することです。クエリーで使用される **SUBSTRING()** 関数を含む関数キー部分の場合、**WHERE** 句には同じ引数を持つ **SUBSTRING()** が含まれている必要があります。次の例では、**SUBSTRING()** への引数がインデックス指定と一致する唯一のクエリーであるため、インデックスを使用できるのは 2 つ目の **SELECT** のみです:

```
CREATE TABLE tbl (  
  col1 LONGTEXT,  
  INDEX idx1 ((SUBSTRING(col1, 1, 10)))  
);  
SELECT * FROM tbl WHERE SUBSTRING(col1, 1, 9) = '123456789';  
SELECT * FROM tbl WHERE SUBSTRING(col1, 1, 10) = '1234567890';
```

関数キーパーツを使用すると、**JSON** 値など、インデックス化できない値のインデックス化が可能です。ただし、目的の効果を得るには、これを正しく行う必要があります。たとえば、次の構文は機能しません:

```
CREATE TABLE employees (  
  data JSON,  
  INDEX ((data->>$.name))  
);
```

構文は、次の理由で失敗します:

- ・ **->>** 演算子は **JSON_UNQUOTE(JSON_EXTRACT(...))** に変換されます。
- ・ **JSON_UNQUOTE()** は、データ型が **LONGTEXT** の値を戻し、非表示の生成されたカラムには同じデータ型が割り当てられます。
- ・ MySQL では、キー部分に接頭辞の長さを指定せずに **LONGTEXT** カラムをインデックス付けすることはできず、機能キー部分では接頭辞の長さを使用できません。

JSON カラムをインデックス付けするには、次のように **CAST()** 関数を使用します:

```
CREATE TABLE employees (  
  data JSON,  
  INDEX ((CAST(data->>$.name AS CHAR(30))))  
);
```

非表示の生成されたカラムには、インデックス付け可能な **VARCHAR(30)** データ型が割り当てられます。ただし、この方法では、インデックスを使用しようとすると新しい問題が発生します:

- ・ **CAST()** は、照合 **utf8mb4_0900_ai_ci** (サーバーのデフォルトの照合) を含む文字列を返します。
- ・ **JSON_UNQUOTE()** は、照合順序が **utf8mb4_bin** (ハードコード) の文字列を返します。

その結果、前述のテーブル定義のインデックス付き式と次のクエリーの **WHERE** 句式の間に照合の不一致があります:

```
SELECT * FROM employees WHERE data->>$.name = 'James';
```

クエリーとインデックスの式が異なるため、インデックスは使用されません。関数キー部分でこのようなシナリオをサポートするために、最適化は使用するインデックスを検索するときに **CAST()** を自動的に削除しますが、

インデックス付き式の照合がクエリー式の照合と一致する場合はのみを削除します。関数キー部分が使用されるインデックスの場合、次の 2 つのソリューションのいずれかが機能します (ただし、ある程度異なります):

- 解決策 1. インデックス付き式に `JSON_UNQUOTE()` と同じ照合を割り当てます:

```
CREATE TABLE employees (
  data JSON,
  INDEX idx ((CAST(data->>"$.name" AS CHAR(30)) COLLATE utf8mb4_bin))
);
INSERT INTO employees VALUES
  ({"name": "james", "salary": 9000 }),
  ({"name": "James", "salary": 10000 }),
  ({"name": "Mary", "salary": 12000 }),
  ({"name": "Peter", "salary": 8000 });
SELECT * FROM employees WHERE data->>"$.name" = 'James';
```

->>演算子は `JSON_UNQUOTE(JSON_EXTRACT(...))` と同じで、`JSON_UNQUOTE()` は照合順序 `utf8mb4_bin` を持つ文字列を返します。したがって、比較では大文字と小文字が区別され、一致する行は 1 つのみです:

```
+-----+
| data          |
+-----+
| {"name": "James", "salary": 10000} |
+-----+
```

- ソリューション 2. クエリーに完全な式を指定します:

```
CREATE TABLE employees (
  data JSON,
  INDEX idx ((CAST(data->>"$.name" AS CHAR(30))))
);
INSERT INTO employees VALUES
  ({"name": "james", "salary": 9000 }),
  ({"name": "James", "salary": 10000 }),
  ({"name": "Mary", "salary": 12000 }),
  ({"name": "Peter", "salary": 8000 });
SELECT * FROM employees WHERE CAST(data->>"$.name" AS CHAR(30)) = 'James';
```

`CAST()` は照合 `utf8mb4_0900_ai_ci` を含む文字列を返すため、比較では大文字と小文字が区別されず、次の 2 つの行が一致します:

```
+-----+
| data          |
+-----+
| {"name": "james", "salary": 9000} |
| {"name": "James", "salary": 10000} |
+-----+
```

オプティマイザではインデックス付けされた生成カラムを含む `CAST()` の自動削除がサポートされていますが、次の方法ではインデックスの有無にかかわらず異なる結果が生成されるため、機能しないことに注意してください (Bug#27337092):

```
mysql> CREATE TABLE employees (
  data JSON,
  generated_col VARCHAR(30) AS (CAST(data->>"$.name" AS CHAR(30)))
);
Query OK, 0 rows affected, 1 warning (0.03 sec)

mysql> INSERT INTO employees (data)
VALUES ({"name": "james"}, {"name": "James"});
Query OK, 2 rows affected, 1 warning (0.01 sec)
Records: 2 Duplicates: 0 Warnings: 1

mysql> SELECT * FROM employees WHERE data->>"$.name" = 'James';
+-----+-----+
| data          | generated_col |
+-----+-----+
| {"name": "James"} | James        |
+-----+-----+
1 row in set (0.00 sec)

mysql> ALTER TABLE employees ADD INDEX idx (generated_col);
```

```
Query OK, 0 rows affected, 1 warning (0.03 sec)
Records: 0 Duplicates: 0 Warnings: 1

mysql> SELECT * FROM employees WHERE data->>$.name = 'James';
+-----+-----+
| data          | generated_col |
+-----+-----+
| {"name": "james"} | james      |
| {"name": "James"} | James     |
+-----+-----+
2 rows in set (0.01 sec)
```

一意インデックス

UNIQUE インデックスは、そのインデックス内のすべての値が異なっている必要があるという制約を作成します。既存の行に一致するキー値を持つ新しい行を追加しようとすると、エラーが発生します。**UNIQUE** インデックスのカラムに接頭辞値を指定する場合、カラム値は接頭辞の長さ内で一意である必要があります。**UNIQUE** インデックスでは、**NULL** を含むことができるカラムに対して複数の **NULL** 値が許可されます。

テーブルに整数型の単一カラムで構成される **PRIMARY KEY** または **UNIQUE NOT NULL** インデックスがある場合は、次のように **_rowid** を使用して **SELECT** ステートメントのインデックス付けされたカラムを参照できます:

- 単一の整数カラムで構成される **PRIMARY KEY** がある場合、**_rowid** は **PRIMARY KEY** カラムを参照します。**PRIMARY KEY** はあるが、単一の整数カラムで構成されていない場合、**_rowid** は使用できません。
- それ以外の場合、**_rowid** は最初の **UNIQUE NOT NULL** インデックスのカラムを参照します (そのインデックスが単一の整数カラムで構成されている場合)。最初の **UNIQUE NOT NULL** インデックスが単一の整数カラムで構成されていない場合、**_rowid** は使用できません。

全文インデックス

FULLTEXT インデックスは **InnoDB** および **MyISAM** テーブルでのみサポートされ、**CHAR**、**VARCHAR**、および **TEXT** カラムのみを含めることができます。インデックス設定は常に、カラム全体に対して実行されます。カラムプリフィクスのインデックス設定はサポートされていないため、プリフィクス長が指定されてもすべて無視されます。操作の詳細は、[セクション12.10「全文検索関数」](#)を参照してください。

複数値インデックス

MySQL 8.0.17 では、**InnoDB** は複数値インデックスをサポートしています。複数値インデックスは、値の配列カラムを格納するカラムに定義されたセカンダリインデックスです。「normal」インデックスには、データレコードごとに1つのインデックスレコードがあります (1:1)。複数値インデックスは、単一のデータレコードに対して複数のインデックスレコードを持つことができます (N:1)。複数値インデックスは **JSON** 配列のインデックス付けを目的としています。たとえば、次の **JSON** ドキュメントの郵便番号の配列に定義された複数値インデックスでは、各インデックスレコードが同じデータレコードを参照するように、郵便番号ごとにインデックスレコードが作成されます。

```
{
  "user": "Bob",
  "user_id": 31,
  "zipcode": [94477, 94536]
}
```

複数値インデックスの作成

CREATE TABLE、**ALTER TABLE** または **CREATE INDEX** ステートメントで複数値インデックスを作成できます。これには、**JSON** 配列内の同じ型のスカラー値を SQL データ型配列にキャストするインデックス定義で **CAST(... AS ... ARRAY)** を使用する必要があります。仮想カラムは、SQL データ型配列の値を使用して透過的に生成されます。最後に、仮想カラムに関数インデックス (仮想インデックスとも呼ばれます) が作成されます。これは、複数値インデックスを形成する SQL データ型配列カラムの値の仮想カラムに定義された関数インデックスです。

次のリストの例は、**customers** という名前のテーブルの **JSON** カラム **custinfo** の配列カラム **\$.zipcode** に複数値インデックス **zips** を作成する方法を示しています。いずれの場合も、**JSON** 配列は **UNSIGNED** 整数値の SQL データ型配列にキャストされます。

- **CREATE TABLE** のみ:

```
CREATE TABLE customers (
```

```
id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,  
modified DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,  
custinfo JSON,  
INDEX zips( (CAST(custinfo->'$.zip' AS UNSIGNED ARRAY)) )  
);
```

- [CREATE TABLE](#) と [ALTER TABLE](#):

```
CREATE TABLE customers (  
  id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,  
  modified DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,  
  custinfo JSON  
);  
  
ALTER TABLE customers ADD INDEX zips( (CAST(custinfo->'$.zip' AS UNSIGNED ARRAY)) );
```

- [CREATE TABLE](#) と [CREATE INDEX](#):

```
CREATE TABLE customers (  
  id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,  
  modified DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,  
  custinfo JSON  
);  
  
CREATE INDEX zips ON customers ( (CAST(custinfo->'$.zip' AS UNSIGNED ARRAY)) );
```

複数値インデックスはコンポジットインデックスの一部として定義することもできます。次の例は、([id](#) および [modified](#) カラムの) 2 つの単一値部分と ([custinfo](#) カラムの) 1 つの複数値部分を含むコンポジットインデックスを示しています:

```
CREATE TABLE customers (  
  id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,  
  modified DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,  
  custinfo JSON  
);  
  
ALTER TABLE customers ADD INDEX comp(id, modified,  
  (CAST(custinfo->'$.zipcode' AS UNSIGNED ARRAY)) );
```

コンポジットインデックスで使用できる複数値キー部分は 1 つのみです。複数値キー部分は、キーの他の部分に対して任意の順序で使用できます。つまり、示されている [ALTER TABLE](#) ステートメントは、[comp\(id, \(CAST\(custinfo->'\\$.zipcode' AS UNSIGNED ARRAY\), modified\)\)](#) (またはその他の順序付け) を使用している可能性があり、引き続き有効です。

複数値インデックスの使用

[WHERE](#) 句で次の関数が指定されている場合、オプティマイザは複数値インデックスを使用してレコードをフェッチします:

- [MEMBER OF\(\)](#)
- [JSON_CONTAINS\(\)](#)
- [JSON_OVERLAPS\(\)](#)

これを示すために、次の [CREATE TABLE](#) および [INSERT](#) ステートメントを使用して [customers](#) テーブルを作成および移入します:

```
mysql> CREATE TABLE customers (  
->  id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,  
->  modified DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,  
->  custinfo JSON  
-> );  
Query OK, 0 rows affected (0.51 sec)  
  
mysql> INSERT INTO customers VALUES  
-> (NULL, NOW(), '{"user":"Jack","user_id":37,"zipcode":[94582,94536]}'),  
-> (NULL, NOW(), '{"user":"Jill","user_id":22,"zipcode":[94568,94507,94582]}'),  
-> (NULL, NOW(), '{"user":"Bob","user_id":31,"zipcode":[94477,94507]}'),  
-> (NULL, NOW(), '{"user":"Mary","user_id":72,"zipcode":[94536]}'),  
-> (NULL, NOW(), '{"user":"Ted","user_id":56,"zipcode":[94507,94582]}');
```



```
Query OK, 5 rows affected (0.07 sec)
Records: 5 Duplicates: 0 Warnings: 0
```

まず、`customers` テーブルに対して 3 つのクエリーを実行します。それぞれが `MEMBER OF()`、`JSON_CONTAINS()` および `JSON_OVERLAPS()` を使用し、次に示す各クエリーの結果が表示されます:

```
mysql> SELECT * FROM customers
-> WHERE 94507 MEMBER OF(custinfo->$.zipcode);
+----+-----+-----+-----+
| id | modified      | custinfo                                     |
+----+-----+-----+-----+
| 2 | 2019-06-29 22:23:12 | {"user": "Jill", "user_id": 22, "zipcode": [94568, 94507, 94582]} |
| 3 | 2019-06-29 22:23:12 | {"user": "Bob", "user_id": 31, "zipcode": [94477, 94507]} |
| 5 | 2019-06-29 22:23:12 | {"user": "Ted", "user_id": 56, "zipcode": [94507, 94582]} |
+----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> SELECT * FROM customers
-> WHERE JSON_CONTAINS(custinfo->$.zipcode, CAST('[94507,94582]' AS JSON));
+----+-----+-----+-----+
| id | modified      | custinfo                                     |
+----+-----+-----+-----+
| 2 | 2019-06-29 22:23:12 | {"user": "Jill", "user_id": 22, "zipcode": [94568, 94507, 94582]} |
| 5 | 2019-06-29 22:23:12 | {"user": "Ted", "user_id": 56, "zipcode": [94507, 94582]} |
+----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> SELECT * FROM customers
-> WHERE JSON_OVERLAPS(custinfo->$.zipcode, CAST('[94507,94582]' AS JSON));
+----+-----+-----+-----+
| id | modified      | custinfo                                     |
+----+-----+-----+-----+
| 1 | 2019-06-29 22:23:12 | {"user": "Jack", "user_id": 37, "zipcode": [94582, 94536]} |
| 2 | 2019-06-29 22:23:12 | {"user": "Jill", "user_id": 22, "zipcode": [94568, 94507, 94582]} |
| 3 | 2019-06-29 22:23:12 | {"user": "Bob", "user_id": 31, "zipcode": [94477, 94507]} |
| 5 | 2019-06-29 22:23:12 | {"user": "Ted", "user_id": 56, "zipcode": [94507, 94582]} |
+----+-----+-----+-----+
4 rows in set (0.00 sec)
```

次に、前述の 3 つのクエリーごとに `EXPLAIN` を実行します:

```
mysql> EXPLAIN SELECT * FROM customers
-> WHERE 94507 MEMBER OF(custinfo->$.zipcode);
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | customers | NULL | ALL | NULL | NULL | NULL | NULL | 5 | 100.00 | Using where |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)

mysql> EXPLAIN SELECT * FROM customers
-> WHERE JSON_CONTAINS(custinfo->$.zipcode, CAST('[94507,94582]' AS JSON));
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | customers | NULL | ALL | NULL | NULL | NULL | NULL | 5 | 100.00 | Using where |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)

mysql> EXPLAIN SELECT * FROM customers
-> WHERE JSON_OVERLAPS(custinfo->$.zipcode, CAST('[94507,94582]' AS JSON));
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | customers | NULL | ALL | NULL | NULL | NULL | NULL | 5 | 100.00 | Using where |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.01 sec)
```

上記の 3 つのクエリーでは、どのキーも使用できません。この問題を解決するには、次のように、`zipcode` 配列カラムの `JSON` カラム (`custinfo`) に複数値インデックスを追加します:

```
mysql> ALTER TABLE customers
-> ADD INDEX zips( (CAST(custinfo->$.zipcode AS UNSIGNED ARRAY)) );
Query OK, 0 rows affected (0.47 sec)
```

Records: 0 Duplicates: 0 Warnings: 0

前の **EXPLAIN** ステートメントを再度実行すると、作成したばかりのインデックス `zips` をクエリーで使用できる (および使用できる) ことがわかります:

```
mysql> EXPLAIN SELECT * FROM customers
-> WHERE 94507 MEMBER OF(custinfo->$.zipcode);
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | customers | NULL | ref | zips | zips | 9 | const | 1 | 100.00 | Using where |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)

mysql> EXPLAIN SELECT * FROM customers
-> WHERE JSON_CONTAINS(custinfo->$.zipcode', CAST('[94507,94582]' AS JSON));
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | customers | NULL | range | zips | zips | 9 | NULL | 6 | 100.00 | Using where |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)

mysql> EXPLAIN SELECT * FROM customers
-> WHERE JSON_OVERLAPS(custinfo->$.zipcode', CAST('[94507,94582]' AS JSON));
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | customers | NULL | range | zips | zips | 9 | NULL | 6 | 100.00 | Using where |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.01 sec)
```

複数値インデックスは一意キーとして定義できます。一意キーとして定義されている場合、複数値インデックスにすでに存在する値を挿入しようとすると、重複キーエラーが返されます。重複する値がすでに存在する場合、次に示すように、一意の複数値インデックスを追加しようとすると失敗します:

```
mysql> ALTER TABLE customers DROP INDEX zips;
Query OK, 0 rows affected (0.55 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> ALTER TABLE customers
-> ADD UNIQUE INDEX zips((CAST(custinfo->$.zipcode' AS UNSIGNED ARRAY)));
ERROR 1062 (23000): Duplicate entry '[94507, ' for key 'customers.zips'
mysql> ALTER TABLE customers
-> ADD INDEX zips((CAST(custinfo->$.zipcode' AS UNSIGNED ARRAY)));
Query OK, 0 rows affected (0.36 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

複数値インデックスの特性

複数値インデックスには、次に示す追加の特性があります:

- 複数値インデックスに影響する DML 操作は、通常のインデックスに影響する DML 操作と同じ方法で処理されますが、唯一の違いは、単一のクラスタインデックスレコードに対して複数の挿入または更新が存在する可能性があることです。
- NULL 値可能性および複数値インデックス:
 - 複数値キー部分に空の配列がある場合、インデックスにエントリは追加されず、データレコードにはインデックススキャンでアクセスできません。
 - 複数値キー部分の生成で NULL 値が返された場合、NULL を含む単一のエントリが複数値インデックスに追加されます。キー部分が NOT NULL として定義されている場合は、エラーが報告されます。
 - 型付き配列カラムが NULL に設定されている場合、ストレージエンジンは、データレコードを指す NULL を含む単一のレコードを格納します。
 - インデックス付き配列では、JSON の NULL 値は許可されません。戻り値が NULL の場合、JSON null として扱われ、「JSON 値が無効です」エラーが報告されます。

- 複数値インデックスは仮想カラムの仮想インデックスであるため、仮想生成カラムのセカンダリインデックスと同じルールに従う必要があります。
- 空の配列のインデックスレコードは追加されません。

複数値インデックスの制限事項

複数値インデックスには、次の制限事項があります:

- 複数値インデックスごとに許可される複数値キー部分は 1 つのみです。ただし、次に示すように、`CAST(... AS ... ARRAY)` 式は `JSON` 文書内の複数の配列を参照できます:

```
CAST(data->$.arr[*] AS UNSIGNED ARRAY)
```

この場合、`JSON` 式に一致するすべての値が単一のフラット配列としてインデックスに格納されます。

- 複数値キー部分を持つインデックスは順序付けをサポートしていないため、主キーとして使用できません。同じ理由で、`ASC` または `DESC` キーワードを使用して複数値インデックスを定義することはできません。
- 複数値インデックスをカバーインデックスにすることはできません。
- 複数値インデックスのレコード当たりの最大値は、単一の undo ログページに格納できるデータ量によって決まります。つまり、65221 バイト (オーバーヘッドの場合は 64K から 315 バイトを引いた値) で、キー値の最大合計長も 65221 バイトです。キーの最大数は様々な要因に依存するため、特定の制限を定義できません。たとえば、テストでは、レコードごとに 1604 個までの整数キーを許可する複数値インデックスが示されています。制限に達すると、次のようなエラーが報告されます: `ERROR 3905 (HY000): 複数値インデックス'idx'のレコード当たりの値の最大数を 1 値超えました。`
- 複数値キー部分で許可されている式のタイプは、`JSON` 式のみです。式は、インデックス付けされたカラムに挿入された `JSON` ドキュメント内の既存の要素を参照する必要はありませんが、構文的に有効である必要があります。
- 同じクラスティンデックスレコードのインデックスレコードは複数値インデックス全体に分散されるため、複数値インデックスではレンジスキャンまたはインデックスのみのスキャンはサポートされません。
- 複数値インデックスは、外部キー指定では使用できません。
- 複数値インデックスにはインデックス接頭辞を定義できません。
- 複数値インデックスは、データキャストで `BINARY` として定義できません (`CAST()` 関数の説明を参照)。
- 複数値インデックスのオンライン作成はサポートされていません。つまり、操作で `ALGORITHM=COPY` が使用されます。パフォーマンスおよび領域要件を参照してください。
- 次の 2 つの文字セットと照合順序の組合せ以外の文字セットと照合順序は、複数値インデックスではサポートされていません:
 1. デフォルトの `binary` 照合順序を持つ `binary` 文字セット
 2. デフォルトの `utf8mb4_0900_as_cs` 照合順序を持つ `utf8mb4` 文字セット。
- `InnoDB` テーブルのカラムに対する他のインデックスと同様に、`USING HASH` では複数値インデックスを作成できません。作成しようとするすると警告が表示されます: `このストレージエンジンは HASH インデックスアルゴリズムをサポートしていません。代わりにストレージエンジンのデフォルトが使用されました。(USING BTREE は通常どおりにサポートされます。)`

空間インデックス

`MyISAM`, `InnoDB`, `NDB` および `ARCHIVE` ストレージエンジンは、`POINT`, `GEOMETRY` などの空間カラムをサポートしています。(セクション 11.4 「空間データ型」では、空間データ型について説明します。) ただし、空間カラムのインデックス設定に対するサポートはエンジンによって異なります。空間カラムの空間インデックスおよび非空間インデックスは、次のルールに従って使用できます。

空間カラムの空間インデックスには、次の特性があります:

- `InnoDB` および `MyISAM` テーブルでのみ使用できます。その他のストレージエンジンに対して `SPATIAL INDEX` を指定すると、エラーが発生します。

- MySQL 8.0.12 では、空間カラムのインデックスは **SPATIAL** インデックスである必要があります。したがって、**SPATIAL** キーワードはオプションですが、空間カラムにインデックスを作成する場合は暗黙的です。
- 単一の空間カラムにのみ使用できます。空間インデックスは、複数の空間カラムに対して作成できません。
- インデックス付きカラムは **NOT NULL** である必要があります。
- カラム接頭辞の長さは禁止されています。各カラムの幅全体にインデックスが設定されます。
- 主キーまたは一意インデックスには使用できません。

(**INDEX**、**UNIQUE** または **PRIMARY KEY** で作成された) 空間カラムの非空間インデックスには、次の特性があります:

- **ARCHIVE** を除く空間カラムをサポートするすべてのストレージエンジンに対して許可されます。
- インデックスが主キーでないかぎり、カラムを **NULL** にすることができます。
- 非 **SPATIAL** インデックスのインデックスタイプは、ストレージエンジンによって異なります。現在は、B ツリーが使用されます。
- **InnoDB**、**MyISAM** および **MEMORY** テーブルに対してのみ **NULL** 値を持つことができるカラムに対して許可されません。

インデックスオプション

キーパートリストの後に、インデックスオプションを指定できます。 **index_option** 値には、次のいずれかを指定できます。

- **KEY_BLOCK_SIZE [=] value**

MyISAM テーブルの場合、**KEY_BLOCK_SIZE** はオプションで、インデックスキーブロックに使用するサイズをバイト単位で指定します。この値はヒントとして扱われます。必要に応じて、異なるサイズが使用される可能性があります。個々のインデックス定義に指定された **KEY_BLOCK_SIZE** 値は、テーブルレベルの **KEY_BLOCK_SIZE** 値をオーバーライドします。

KEY_BLOCK_SIZE は、**InnoDB** テーブルのインデックスレベルではサポートされていません。 [セクション 13.1.20 「CREATE TABLE ステートメント」](#) を参照してください。

- **index_type**

一部のストレージエンジンでは、インデックスの作成時にインデックスタイプを指定できます。例:

```
CREATE TABLE lookup (id INT) ENGINE = MEMORY;
CREATE INDEX id_index ON lookup (id) USING BTREE;
```

[表13.1 「ストレージエンジンあたりのインデックスタイプ」](#) には、様々なストレージエンジンでサポートされている許容インデックスタイプ値が表示されます。複数のインデックスタイプが示されている場合は、最初のもので、インデックスタイプ指示子が指定されないときのデフォルトになります。テーブルに示されていないストレージエンジンは、インデックス定義で **index_type** 句をサポートしていません。

表 13.1 ストレージエンジンあたりのインデックスタイプ

ストレージエンジン	許可されるインデックスタイプ
InnoDB	BTREE
MyISAM	BTREE
MEMORY/HEAP	HASH 、 BTREE
NDB	HASH 、 BTREE (テキストの注を参照してください)

index_type 句は、**FULLTEXT INDEX** または (MySQL 8.0.12 より前の) **SPATIAL INDEX** 仕様には使用できません。フルテキストインデックスの実装は、ストレージエンジンに依存します。空間インデックスは R ツリーインデックスとして実装されます。

特定のストレージエンジンに対して無効なインデックスタイプを指定しても、エンジンがクエリー結果に影響を与えずに使用できる別のインデックスタイプが使用可能な場合、エンジンは使用可能なタイプを使用します。パーサーは、**RTREE** を型名として認識します。MySQL 8.0.12 では、これは **SPATIAL** インデックスに対してのみ許可されます。8.0.12 より前では、どのストレージエンジンにも **RTREE** を指定できません。

BTREE インデックスは、**NDB** ストレージエンジンによって T ツリーインデックスとして実装されます。

注記

NDB テーブルカラム上のインデックスの場合、**USING** オプションは、一意のインデックスまたは主キーに対してのみ指定できます。**USING HASH** では、順序付けされたインデックスは作成されません。それ以外の場合、**NDB** テーブルに一意インデックスまたは主キーを作成すると、順序付けられたインデックスとハッシュインデックスの両方が自動的に作成され、それぞれが同じカラムセットをインデックス付けします。

NDB テーブルの 1 つ以上の **NULL** カラムを含む一意インデックスの場合、ハッシュインデックスはリテラル値の検索にのみ使用できます。つまり、**IS [NOT] NULL** 条件ではテーブルの全体スキャンが必要です。回避策として、このようなテーブルの **NULL** カラムを使用している一意のインデックスが、順序付けられたインデックスを含む方法で常に作成されるようにすることがあります。つまり、インデックスの作成時に **USING HASH** を使用しないようにします。

特定のストレージエンジンに対して無効なインデックスタイプを指定しても、エンジンがクエリー結果に影響を与えずに使用できる別のインデックスタイプが使用可能な場合、エンジンは使用可能なタイプを使用します。パーサーは **RTREE** をタイプ名として認識しますが、現在、これはどのストレージエンジンに対しても指定できません。

注記

ON tbl_name 句の前の **index_type** オプションの使用は非推奨になりました。この位置でのオプションの使用のサポートは、将来の MySQL リリースで削除される予定です。**index_type** オプションが前とあとの両方の位置で指定された場合は、最後のオプションが適用されます。

TYPE type_name は、**USING type_name** のシノニムとして認識されます。ただし、推奨される形式は **USING** です。

次のテーブルに、**index_type** オプションをサポートするストレージエンジンのインデックス特性を示します。

表 13.2 InnoDB ストレージエンジンのインデックス特性

インデックスクラス	インデックスタイプ	NULL VALUES を格納	複数の NULL 値を許可	IS NULL スキャンタイプ	IS NOT NULL スキャンタイプ
主キー	BTREE	いいえ	いいえ	N/A	N/A
Unique	BTREE	はい	はい	インデックス	インデックス
鍵	BTREE	はい	はい	インデックス	インデックス
FULLTEXT	N/A	はい	はい	Table	Table
SPATIAL	N/A	いいえ	いいえ	N/A	N/A

表 13.3 MyISAM ストレージエンジンのインデックス特性

インデックスクラス	インデックスタイプ	NULL VALUES を格納	複数の NULL 値を許可	IS NULL スキャンタイプ	IS NOT NULL スキャンタイプ
主キー	BTREE	いいえ	いいえ	N/A	N/A
Unique	BTREE	はい	はい	インデックス	インデックス
鍵	BTREE	はい	はい	インデックス	インデックス

インデックスクラス	インデックスタイプ	NULL VALUES を格納	複数の NULL 値を許可	IS NULL スキャンタイプ	IS NOT NULL スキャンタイプ
FULLTEXT	N/A	はい	はい	Table	Table
SPATIAL	N/A	いいえ	いいえ	N/A	N/A

表 13.4 MEMORY ストレージエンジンのインデックス特性

インデックスクラス	インデックスタイプ	NULL VALUES を格納	複数の NULL 値を許可	IS NULL スキャンタイプ	IS NOT NULL スキャンタイプ
主キー	BTREE	いいえ	いいえ	N/A	N/A
Unique	BTREE	はい	はい	インデックス	インデックス
鍵	BTREE	はい	はい	インデックス	インデックス
主キー	HASH	いいえ	いいえ	N/A	N/A
Unique	HASH	はい	はい	インデックス	インデックス
鍵	HASH	はい	はい	インデックス	インデックス

表 13.5 NDB ストレージエンジンのインデックス特性

インデックスクラス	インデックスタイプ	NULL VALUES を格納	複数の NULL 値を許可	IS NULL スキャンタイプ	IS NOT NULL スキャンタイプ
主キー	BTREE	いいえ	いいえ	インデックス	インデックス
Unique	BTREE	はい	はい	インデックス	インデックス
鍵	BTREE	はい	はい	インデックス	インデックス
主キー	HASH	いいえ	いいえ	テーブル (ノート 1 を参照)	テーブル (ノート 1 を参照)
Unique	HASH	はい	はい	テーブル (ノート 1 を参照)	テーブル (ノート 1 を参照)
鍵	HASH	はい	はい	テーブル (ノート 1 を参照)	テーブル (ノート 1 を参照)

テーブルノート:

1. [USING HASH](#) では、暗黙的な順序付きインデックスは作成されません。

- [WITH PARSER parser_name](#)

このオプションは、[FULLTEXT](#) インデックスとともにのみ使用できます。これは、全文インデックス設定および検索操作に特殊な処理が必要な場合に、パーサープラグインをインデックスに関連付けます。[InnoDB](#) および [MyISAM](#) は、フルテキストパーサープラグインをサポートしています。フルテキストパーサープラグインが関連付けられた [MyISAM](#) テーブルがある場合は、[ALTER TABLE](#) を使用してテーブルを [InnoDB](#) に変換できます。詳細は、[Full-Text Parser Plugins](#) および [Writing Full-Text Parser Plugins](#) を参照してください。

- [COMMENT 'string'](#)

インデックス定義には、最大 1024 文字のオプションのコメントを含めることができます。

インデックスページ用の [MERGE_THRESHOLD](#) は、[CREATE INDEX](#) ステートメントの `index_option COMMENT` 句を使用して個々のインデックスに対して構成できます。例:

```
CREATE TABLE t1 (id INT);
```



```
CREATE INDEX id_index ON t1 (id) COMMENT 'MERGE_THRESHOLD=40';
```

行が削除されたとき、または更新操作によって行が短縮されたときに、インデックスページのページフル率が `MERGE_THRESHOLD` 値を下回った場合、InnoDB はインデックスページを隣接するインデックスページとマージしようとします。デフォルトの `MERGE_THRESHOLD` 値は 50 で、これは以前にハードコードされた値です。

`MERGE_THRESHOLD` は、`CREATE TABLE` および `ALTER TABLE` ステートメントを使用して、インデックスレベルおよびテーブルレベルで定義することもできます。詳細は、[セクション15.8.11「インデックスページのマージしきい値の構成」](#)を参照してください。

- **VISIBLE, INVISIBLE**

インデックスの可視性を指定します。インデックスはデフォルトで可視化されます。不可視インデックスはオプティマイザでは使用されません。インデックスの可視性の指定は、主キー以外のインデックス (明示的または暗黙的) に適用されます。詳細は、[セクション8.3.12「不可視のインデックス」](#)を参照してください。

- **ENGINE_ATTRIBUTE** および **SECONDARY_ENGINE_ATTRIBUTE** オプション (MySQL 8.0.21 の時点で使用可能) は、プライマリストレージエンジンおよびセカンダリストレージエンジンのインデックス属性を指定するために使用されます。オプションは、将来の使用のために予約されています。

許可される値は、有効な JSON ドキュメントまたは空の文字列 ("") を含む文字列リテラルです。無効な JSON が拒否されました。

```
CREATE INDEX i1 ON t1 (c1) ENGINE_ATTRIBUTE='{ "key": "value" }';
```

`ENGINE_ATTRIBUTE` および `SECONDARY_ENGINE_ATTRIBUTE` の値は、エラーなしで繰り返すことができます。この場合、最後に指定した値が使用されます。

`ENGINE_ATTRIBUTE` および `SECONDARY_ENGINE_ATTRIBUTE` の値は、サーバーによってチェックされず、テーブルストレージエンジンが変更されたときにもクリアされません。

テーブルのコピーおよびロックのオプション

`ALGORITHM` 句および `LOCK` 句を指定して、インデックスの変更中にテーブルの読取りおよび書込みを行うためのテーブルのコピー方法および同時実行性のレベルに影響を与えることができます。これらには、`ALTER TABLE` ステートメントの場合と同じ意味があります。詳細は、[セクション13.1.9「ALTER TABLE ステートメント」](#)を参照してください。

NDB Cluster は、標準の MySQL Server で使用されるものと同じ `ALGORITHM=INPLACE` 構文を使用したオンライン操作をサポートします。詳しくは [セクション23.5.11「NDB Cluster での ALTER TABLE を使用したオンライン操作」](#) をご覧ください。

13.1.16 CREATE LOGFILE GROUP ステートメント

```
CREATE LOGFILE GROUP logfile_group
  ADD UNDOFILE 'undo_file'
  [INITIAL_SIZE [=] initial_size]
  [UNDO_BUFFER_SIZE [=] undo_buffer_size]
  [REDO_BUFFER_SIZE [=] redo_buffer_size]
  [NODEGROUP [=] nodegroup_id]
  [WAIT]
  [COMMENT [=] 'string']
  ENGINE [=] engine_name
```

このステートメントは、'undo_file' という名前の 1 つの UNDO ファイルを持つ `logfile_group` という名前の新しいログファイルグループを作成します。CREATE LOGFILE GROUP ステートメントには、ADD UNDOFILE 句が 1 つだけ存在します。ログファイルグループの命名を管理するルールについては、[セクション9.2「スキーマオブジェクト名」](#)を参照してください。

注記

NDB Cluster ディスクデータオブジェクトはすべて同じ名前空間を共有します。つまり、各ディスクデータオブジェクトは (単に、特定の型の各ディスクデータオブジェクトというだけでなく)、一意の名前が付けられている必要があります。たとえば、テーブルスペースと

ログファイルグループを同じ名前にしたり、テーブルスペースとデータファイルを同じ名前にしたりすることはできません。

NDB Cluster インスタンスごとに一度に 1 つのログファイルグループしか存在できません。

オプションの `INITIAL_SIZE` パラメータは、`UNDO` ファイルの初期サイズを設定します。指定されていない場合は、デフォルトで `128M` (`128M` バイト) になります。オプションの `UNDO_BUFFER_SIZE` パラメータは、ログファイルグループの `UNDO` バッファで使われるサイズを設定します。`UNDO_BUFFER_SIZE` のデフォルト値は `8M` (`8M` バイト) です。この値が、使用可能なシステムメモリーの量を超えることはできません。これらのパラメータは、どちらもバイト単位で指定されます。オプションで、これらのいずれかまたは両方の後に、`my.cnf` で使用されるものと同様に、大きさの順に 1 文字の略称を付けることができます。一般に、これは `M` (`M` バイト) または `G` (`G` バイト) のどちらかの文字です。

`UNDO_BUFFER_SIZE` に使われるメモリーは、サイズが `SharedGlobalMemory` データノード構成パラメータの値によって決定されるグローバルプールから取得されます。これには、`InitialLogFileGroup` データノード構成パラメータの設定により、このオプションに暗黙的に指定されるデフォルト値もすべて含まれます。

`UNDO_BUFFER_SIZE` に許可される最大値は `629145600` (`600M` バイト) です。

32 ビットシステム上では、`INITIAL_SIZE` のサポートされる最大値は `4294967296` (`4G` バイト) です。 (Bug #29186)

`INITIAL_SIZE` の許可される最小値は `1048576` (`1M` バイト) です。

`ENGINE` オプションは、このログファイルグループによって使われるストレージエンジンを決定します。ここで、`engine_name` はそのストレージエンジンの名前です。MySQL 8.0 では、これは `NDB` (または `NDBCLUSTER`) である必要があります。`ENGINE` が設定されていない場合、MySQL は、`default_storage_engine` サーバシステム変数 (以前の `storage_engine`) で指定されたエンジンを使用しようとします。いずれにしても、エンジンが `NDB` または `NDBCLUSTER` として指定されていない場合、`CREATE LOGFILE GROUP` ステートメントは成功したように見えますが、次に示すように、実際にはログファイルグループの作成に失敗します。

```
mysql> CREATE LOGFILE GROUP lg1
-> ADD UNDOFILE 'undo.dat' INITIAL_SIZE = 10M;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Error | 1478 | Table storage engine 'InnoDB' does not support the create option 'TABLESPACE or LOGFILE GROUP' |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> DROP LOGFILE GROUP lg1 ENGINE = NDB;
ERROR 1529 (HY000): Failed to drop LOGFILE GROUP

mysql> CREATE LOGFILE GROUP lg1
-> ADD UNDOFILE 'undo.dat' INITIAL_SIZE = 10M
-> ENGINE = NDB;
Query OK, 0 rows affected (2.97 sec)
```

`NDB` 以外のストレージエンジンに名前が付けられているが、成功したように見える場合、`CREATE LOGFILE GROUP` ステートメントは実際にはエラーを返しません。これは `NDB Cluster` の将来のリリースで対処することを希望している既知の問題です。

`REDO_BUFFER_SIZE`、`NODEGROUP`、`WAIT`、および `COMMENT` は解析されますが、無視されるため、MySQL 8.0 では何の効果もありません。これらのオプションは、将来の拡張のために用意されています。

`ENGINE [=] NDB` とともに使用された場合は、ログファイルグループとそれに関連付けられた `UNDO` ログファイルが各クラスターデータノード上に作成されます。`INFORMATION_SCHEMA.FILES` テーブルをクエリーすることによって、`UNDO` ファイルが作成されたことを確認したり、それらに関する情報を取得したりできます。例:

```
mysql> SELECT LOGFILE_GROUP_NAME, LOGFILE_GROUP_NUMBER, EXTRA
-> FROM INFORMATION_SCHEMA.FILES
-> WHERE FILE_NAME = 'undo_10.dat';
+-----+-----+-----+
| LOGFILE_GROUP_NAME | LOGFILE_GROUP_NUMBER | EXTRA |
+-----+-----+-----+
| lg_3 | 11 | CLUSTER_NODE=3 |
```

```
| lg_3 | 11 | CLUSTER_NODE=4 |
+-----+-----+-----+
2 rows in set (0.06 sec)
```

CREATE LOGFILE GROUP は NDB Cluster のディスクデータストレージでのみ役立ちます。 [セクション 23.5.10「NDB Cluster ディスクデータテーブル」](#) を参照してください。

13.1.17 CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント

```
CREATE
[DEFINER = user]
PROCEDURE sp_name ([proc_parameter[,...]])
[characteristic ...] routine_body

CREATE
[DEFINER = user]
FUNCTION sp_name ([func_parameter[,...]])
RETURNS type
[characteristic ...] routine_body

proc_parameter:
[ IN | OUT | INOUT ] param_name type

func_parameter:
param_name type

type:
Any valid MySQL data type

characteristic: {
COMMENT 'string'
| LANGUAGE SQL
| [NOT] DETERMINISTIC
| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
| SQL SECURITY { DEFINER | INVOKER }
}

routine_body:
Valid SQL routine statement
```

これらのステートメントは、ストアルーチン (ストアプロシージャまたはストアファンクション) の作成に使用されます。つまり、指定されたルーチンがサーバーに認識されます。デフォルトでは、ストアルーチンはデフォルトのデータベースに関連付けられます。ルーチンを明示的に特定のデータベースに関連付けるには、そのルーチンの作成時に、その名前を `db_name.sp_name` として指定します。

CREATE FUNCTION ステートメントはまた、UDF (ユーザー定義関数) をサポートするために MySQL でも使用されます。 [セクション 13.7.4.1「ユーザー定義関数用の CREATE FUNCTION ステートメント」](#) を参照してください。UDF は、外部のストアファンクションと見なすことができます。ストアファンクションは、その名前空間を UDF と共有します。各種の関数への参照をサーバーが解釈する方法を記述したルールについては、 [セクション 9.2.5「関数名の構文解析と解決」](#) を参照してください。

ストアプロシージャを呼び出すには、CALL ステートメントを使用します ([セクション 13.2.1「CALL ステートメント」](#) を参照してください)。ストアファンクションを呼び出すには、式でその関数を参照します。その関数は、式の評価中に値を返します。

CREATE PROCEDURE および CREATE FUNCTION には、CREATE ROUTINE 権限が必要です。DEFINER 句が存在する場合、 [セクション 25.6「ストアオブジェクトのアクセス制御」](#) で説明されているように、必要な権限は user の値によって異なります。バイナリロギングが有効になっている場合は、 [セクション 25.7「ストアプログラムバイナリロギング」](#) で説明されているように、CREATE FUNCTION に SUPER 権限が必要になることがあります。

デフォルトでは、MySQL は、ルーチン作成者に ALTER ROUTINE および EXECUTE 権限を自動的に付与します。この動作は、automatic_sp_privileges システム変数を無効にすることによって変更できます。 [セクション 25.2.2「ストアルーチンと MySQL 権限」](#) を参照してください。

DEFINER および SQL SECURITY 句は、このセクションのあとの方で説明されているように、ルーチンの実行時にアクセス権を確認するときに使用されるセキュリティーコンテキストを指定します。

ルーチン名が組み込みの SQL 関数の名前と同じである場合は、そのルーチンを定義するか、またはあとで呼び出すときに名前とそれに続く括弧の間にスペースを使用しないかぎり、構文エラーが発生します。このため、ユーザー独自のストアドルーチンに既存の SQL 関数の名前を使用することは避けてください。

IGNORE_SPACE SQL モードは、ストアドルーチンではなく、組み込み関数に適用されます。ストアドルーチン名のあとのスペースは、**IGNORE_SPACE** が有効になっているかどうかには関係なく、常に許可されます。

括弧で囲まれたパラメータリストは、常に存在する必要があります。パラメータが存在しない場合は、`()` の空のパラメータリストを使用するようにしてください。パラメータ名では大文字と小文字は区別されません。

各パラメータは、デフォルトでは **IN** パラメータです。それ以外のパラメータを指定するには、パラメータ名の前にキーワード **OUT** または **INOUT** を使用します。

注記

IN、**OUT**、または **INOUT** としてのパラメータの指定は、**PROCEDURE** に対してのみ有効です。**FUNCTION** の場合、パラメータは常に **IN** パラメータと見なされます。

IN パラメータは、プロシージャへの値を渡します。プロシージャはその値を変更する可能性があります、そのプロシージャから戻ったとき、その変更は呼び出し元に表示されません。**OUT** パラメータは、プロシージャから呼び出し元に値を渡します。その初期値はプロシージャ内では **NULL** であり、そのプロシージャから戻ったとき、その値は呼び出し元に表示されます。**INOUT** パラメータは呼び出し元によって初期化され、プロシージャで変更できます。そのプロシージャから戻ったとき、プロシージャによって行われた変更はすべて呼び出し元に表示されます。

OUT または **INOUT** パラメータごとに、プロシージャを呼び出す **CALL** ステートメントでユーザー定義変数を渡して、プロシージャから戻ったときにその値を取得できるようにします。別のストアードプロシージャまたはストアードファンクション内からプロシージャをコールする場合は、ルーチンパラメータまたはローカルルーチン変数を **OUT** または **INOUT** パラメータとして渡すこともできます。トリガー内からプロシージャをコールする場合は、**NEW.col_name** を **OUT** または **INOUT** パラメータとして渡すこともできます。

プロシージャパラメータに対する未処理条件の影響の詳細は、[セクション13.6.7.8「条件の処理と OUT または INOUT パラメータ」](#) を参照してください。

ルーチン内に準備されたステートメントでルーチンパラメータを参照することはできません。[セクション25.8「ストアードプログラムの制約」](#) を参照してください。

次の例は、国コードを指定して、**world** データベースの **city** テーブルに表示されるその国の都市の数をカウントする単純なストアードプロシージャを示しています。国コードは **IN** パラメータを使用して渡され、市区町村数は **OUT** パラメータを使用して返されます:

```
mysql> delimiter //
mysql> CREATE PROCEDURE citycount (IN country CHAR(3), OUT cities INT)
BEGIN
  SELECT COUNT(*) INTO cities FROM world.city
  WHERE CountryCode = country;
END//
Query OK, 0 rows affected (0.01 sec)

mysql> delimiter ;

mysql> CALL citycount('JPN', @cities); -- cities in Japan
Query OK, 1 row affected (0.00 sec)

mysql> SELECT @cities;
+-----+
| @cities |
+-----+
|   248 |
+-----+
1 row in set (0.00 sec)

mysql> CALL citycount('FRA', @cities); -- cities in France
Query OK, 1 row affected (0.00 sec)

mysql> SELECT @cities;
+-----+
| @cities |
```

```
+-----+
| 40 |
+-----+
1 row in set (0.00 sec)
```

この例では、プロシーチャーの定義中に `mysql` クライアントの `delimiter` コマンドを使用して、ステートメント区切り文字を `;` から `//` に変更しています。これにより、プロシーチャー本体で使用される `;` 区切り文字を、`mysql` 自体が解釈するのではなく、サーバーに渡すようにすることができます。 [セクション25.1「ストアードプログラムの定義」](#) を参照してください。

`RETURNS` 句は、`FUNCTION` (これには必須です) に対してのみ指定できます。これは関数の戻り型を示すものであり、関数本体には `RETURN value` ステートメントが含まれている必要があります。 `RETURN` ステートメントが異なる型の値を返した場合、その値は正しい型に強制的に変更されます。たとえば、ある関数が `RETURNS` 句で `ENUM` または `SET` 値を指定しているが、`RETURN` ステートメントが整数を返した場合、その関数から返される値は `SET` メンバーのセットの対応する `ENUM` メンバーを示す文字列になります。

次の関数例はパラメータを受け取り、SQL 関数を使用して操作を実行したあと、結果を返します。この場合は、関数定義に内部の `;` ステートメント区切り文字が含まれていないため、`delimiter` を使用する必要はありません。

```
mysql> CREATE FUNCTION hello (s CHAR(20))
mysql> RETURNS CHAR(50) DETERMINISTIC
RETURN CONCAT("Hello, 's;!");
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT hello('world');
+-----+
| hello('world') |
+-----+
| Hello, world! |
+-----+
1 row in set (0.00 sec)
```

パラメータ型と関数の戻り型は、任意の有効なデータ型を使用するように宣言できます。 `COLLATE` 属性は、`CHARACTER SET` 指定の前にある場合に使用できます。

`routine_body` は、有効な SQL ルーチンステートメントで構成されます。これは `SELECT` や `INSERT` などの単純なステートメントでも、`BEGIN` と `END` を使用して記述された複合ステートメントでもかまいません。複合ステートメントには、宣言、ループ、およびその他の制御構造ステートメントを含めることができます。これらのステートメントの構文については、 [セクション13.6「複合ステートメントの構文」](#) で説明されています。実際には、本体が単一の `RETURN` ステートメントで構成されていないが、ストアードファンクションは複合ステートメントを使用する傾向があります。

MySQL では、ルーチンに `CREATE` や `DROP` などの DDL ステートメントを含めることが許可されます。MySQL ではまた、ストアードプロシーチャーに `COMMIT` などの SQL トランザクションステートメントを含めることも許可されます (ただし、ストアードファンクションには許可されません)。ストアードファンクションには、明示的または暗黙的なコミットまたはロールバックを実行するステートメントを含めることはできません。これらのステートメントのサポートは、SQL 標準では必要ありません。SQL 標準では、各 DBMS ベンダーがこれらのステートメントを許可するかどうかを決められると定めています。

結果セットを返すステートメントはストアードプロシーチャー内で使用できますが、ストアードファンクション内では使用できません。この禁止には、`INTO var_list` 句を含まない `SELECT` ステートメントや、`SHOW`、`EXPLAIN`、`CHECK TABLE` などのその他のステートメントが含まれます。結果セットを返すことを関数の定義時に判定できるステートメントの場合は、`Not allowed to return a result set from a function` エラーが発生します (`ER_SP_NO_RETSET`)。結果セットを返すことを実行時にしか判定できないステートメントの場合は、`PROCEDURE %s can't return a result set in the given context` エラーが発生します (`ER_SP_BADSELECT`)。

ストアードルーチン内の `USE` ステートメントは許可されていません。ルーチンが呼び出されると、暗黙的な `USE db_name` が実行されます (また、そのルーチンが終了すると元に戻されます)。これにより、そのルーチンには実行中、特定のデフォルトデータベースが割り当てられます。ルーチンのデフォルトデータベース以外のデータベース内のオブジェクトへの参照は、適切なデータベース名で修飾するようにしてください。

ストアードルーチン内では許可されないステートメントの詳細は、 [セクション25.8「ストアードプログラムの制約」](#) を参照してください。

MySQL インタフェースを備える言語で記述されたプログラム内からのストアードプロシーチャーの呼び出しについては、 [セクション13.2.1「CALL ステートメント」](#) を参照してください。

MySQL は、ルーチンが作成または変更されたときの有効な `sql_mode` システム変数の設定を格納し、ルーチンが実行を開始したときの現在のサーバー SQL モードには関係なく、常にそのルーチンを強制的にこの設定で実行します。

呼び出し元の SQL モードからそのルーチンの SQL モードへの切り替えは、引数を評価し、結果として得られる値をルーチンパラメータに割り当てたあとに実行されます。あるルーチンを厳密な SQL モードで定義したが、その呼び出しを非厳密モードで行なった場合は、引数のルーチンパラメータへの割り当てが厳密モードで実行されません。ルーチンに渡される式を厳密な SQL モードで割り当てて必要がある場合は、そのルーチンを厳密モードが有効な状態で呼び出すようにしてください。

COMMENT 特性は MySQL 拡張であり、そのストアドルーチンの説明のために使用できます。この情報は、**SHOW CREATE PROCEDURE** および **SHOW CREATE FUNCTION** ステートメントによって表示されます。

LANGUAGE 特性は、そのルーチンが記述されている言語を示します。サーバーはこの特性を無視します。SQL ルーチンのみがサポートされています。

ルーチンは、同じ入力パラメータに対して常に同じ結果を生成する場合は「決定的」と見なされ、それ以外の場合は「非決定的」と見なされます。ルーチン定義で **DETERMINISTIC** と **NOT DETERMINISTIC** のどちらも指定されていない場合、デフォルトは **NOT DETERMINISTIC** になります。関数が決定的であることを宣言するには、明示的に **DETERMINISTIC** を指定する必要があります。

ルーチンの性質の評価は、作成者の「誠実さ」に基づいています。MySQL は、**DETERMINISTIC** と宣言されたルーチンに非決定的な結果を生成するステートメントが含まれていないかどうかをチェックしません。ただし、ルーチンの誤った宣言は、その結果やパフォーマンスに影響を与える可能性があります。非決定的なルーチンを **DETERMINISTIC** として宣言すると、オプティマイザが正しくない実行計画を選択するために、予期しない結果を招くことがあります。決定的なルーチンを **NONDETERMINISTIC** として宣言すると、使用可能な最適化が使用されなくなるために、パフォーマンスが低下することがあります。

バイナリロギングが有効になっている場合、**DETERMINISTIC** 特性は、MySQL がどのルーチン定義を受け入れるかに影響を与えます。[セクション25.7「ストアプログラムバイナリロギング」](#)を参照してください。

NOW() 関数 (または、そのシノニム) あるいは **RAND()** を含むルーチンは非決定的ですが、引き続きレプリケーションに対して安全である可能性があります。**NOW()** の場合、バイナリログにはタイムスタンプが含まれ、正しくレプリケートされます。**RAND()** もまた、ルーチンの実行中に 1 回だけ呼び出されるかぎり、正しくレプリケートされます。(ルーチン実行タイムスタンプおよび乱数シードは、ソースとレプリカで同一の暗黙的な入力とみなすことができます。)

いくつかの特性によって、ルーチンによるデータ使用の性質に関する情報が提供されます。MySQL では、これらの特性はアドバイザーにすぎません。サーバーはこれらを使用して、ルーチンの実行を許可するステートメントの種類を制約しません。

- **CONTAINS SQL** は、そのルーチンに、データの読み取りや書き込みを行うステートメントが含まれていないことを示します。これは、これらのどの特性も明示的に指定されていない場合のデフォルトです。このようなステートメントの例として、実行されてもデータの読み取りや書き込みを行わない **SET @x = 1** または **DO RELEASE_LOCK('abc')** があります。
- **NO SQL** は、そのルーチンに SQL ステートメントが含まれていないことを示します。
- **READS SQL DATA** は、そのルーチンに、データを読み取るステートメント (**SELECT** など) が含まれているが、データを書き込むステートメントは含まれていないことを示します。
- **MODIFIES SQL DATA** は、そのルーチンに、データを書き込む可能性のあるステートメント (**INSERT** や **DELETE** など) が含まれていることを示します。

SQL SECURITY 特性は、セキュリティーコンテキストを指定する **DEFINER** または **INVOKER** のどちらかです。これは、そのルーチンがルーチンの **DEFINER** 句で指定されたアカウント、またはそのルーチンを呼び出すユーザーのどちらの権限を使用して実行されるかを示します。このアカウントには、そのルーチンが関連付けられているデータベースにアクセスするためのアクセス権が必要です。デフォルト値は **DEFINER** です。そのルーチンを呼び出すユーザーには、それに対する **EXECUTE** 権限が必要です。また、そのルーチンが定義者のセキュリティーコンテキストで実行される場合は、**DEFINER** アカウントにもその権限が必要です。

DEFINER 句は、**SQL SECURITY DEFINER** 特性を持つルーチンのルーチン実行時にアクセス権を確認するときに使用される MySQL アカウントを指定します。

DEFINER 句が存在する場合、**user** 値は `'user_name'@'host_name'`、**CURRENT_USER** または **CURRENT_USER()** として指定された MySQL アカウントである必要があります。許可される **user** 値は、[セクション25.6「ストアオブ](#)

「[プロジェクトのアクセス制御](#)」で説明されているように、保持する権限によって異なります。ストアドルーチンのセキュリティに関する追加情報については、そのセクションも参照してください。

DEFINER 句を省略すると、デフォルトの定義者は CREATE PROCEDURE または CREATE FUNCTION ステートメントを実行するユーザーになります。これは、明示的に DEFINER = CURRENT_USER を指定するのと同じです。

SQL SECURITY DEFINER 特性で定義されたストアドルーチンの本体内で、CURRENT_USER 関数はルーチン DEFINER 値を返します。ストアドルーチン内のユーザー監査については、[セクション6.2.22「SQL ベースのアカウントアクティビティ監査」](#)を参照してください。

mysql.user システムテーブルにリストされている MySQL アカウントの数を表示する次の手順について考えてみます:

```
CREATE DEFINER = 'admin'@'localhost' PROCEDURE account_count()
BEGIN
  SELECT 'Number of accounts:', COUNT(*) FROM mysql.user;
END;
```

このプロシージャには、それがどのユーザーによって定義されている場合でも、'admin'@'localhost' の DEFINER アカウントが割り当てられます。また、それがどのユーザーから呼び出された場合でも、そのアカウントの権限で実行されます (デフォルトのセキュリティ特性は DEFINER であるため)。このプロシージャは、呼び出し元にそれに対する EXECUTE 権限があり、かつ 'admin'@'localhost' に mysql.user テーブルに対する SELECT 権限があるかどうかに応じて成功または失敗します。

ここで、このプロシージャが SQL SECURITY INVOKER 特性を使用して定義されているとします。

```
CREATE DEFINER = 'admin'@'localhost' PROCEDURE account_count()
SQL SECURITY INVOKER
BEGIN
  SELECT 'Number of accounts:', COUNT(*) FROM mysql.user;
END;
```

このプロシージャは、依然として 'admin'@'localhost' の DEFINER を持っていますが、この場合は呼び出し元ユーザーの権限で実行されます。そのため、このプロシージャは、呼び出し元にそれに対する EXECUTE 権限と、mysql.user テーブルに対する SELECT 権限があるかどうかに応じて成功または失敗します。

サーバーは、ルーチンパラメータ、DECLARE を使用して作成されたローカルルーチン変数、または関数の戻り値のデータ型を次のように処理します。

- データ型の不一致やオーバーフローがないかどうか割り当てがチェックされます。変換やオーバーフローの問題によって警告が発生するか、または厳密な SQL モードではエラーが発生します。
- スカラー値のみを割り当てることができます。たとえば、SET x = (SELECT 1, 2) などのステートメントは無効です。
- 文字データ型では、CHARACTER SET が宣言に含まれている場合、指定された文字セットとそのデフォルトの照合順序が使用されます。COLLATE 属性も存在する場合は、デフォルトの照合順序ではなく、その照合順序が使用されます。

CHARACTER SET および COLLATE が存在しない場合は、ルーチンの作成時に有効なデータベース文字セットおよび照合順序が使用されます。サーバーでデータベース文字セットおよび照合順序を使用しないようにするには、文字データパラメータに明示的な CHARACTER SET および COLLATE 属性を指定します。

データベースのデフォルトの文字セットまたは照合順序を変更する場合は、新しいデータベースのデフォルトを使用するストアドルーチンを削除して再作成する必要があります。

データベース文字セットおよび照合順序は、character_set_database および collation_database システム変数の値で指定されます。詳細は、[セクション10.3.3「データベース文字セットおよび照合順序」](#)を参照してください。

13.1.18 CREATE SERVER ステートメント

```
CREATE SERVER server_name
  FOREIGN DATA WRAPPER wrapper_name
  OPTIONS (option [, option] ...)
```

```
option: {
  HOST character-literal
  | DATABASE character-literal
```

```

| USER character-literal
| PASSWORD character-literal
| SOCKET character-literal
| OWNER character-literal
| PORT numeric-literal
}

```

このステートメントは、**FEDERATED** ストレージエンジンで使用するためのサーバーの定義を作成します。**CREATE SERVER** ステートメントは、**mysql** データベース内の **servers** テーブルに新しい行を作成します。このステートメントには、**SUPER** 権限が必要です。

server_name は、そのサーバーへの一意の参照にしてください。サーバー定義はサーバーのスコープ内でグローバルであり、サーバー定義を特定のデータベースに対して修飾することはできません。**server_name** の最大長は 64 文字 (64 文字を超える名前は暗黙的に切り捨てられます) で、大/小文字は区別されません。この名前は、引用符で囲まれた文字列として指定できます。

wrapper_name は識別子であり、一重引用符で囲むことができます。

各 **option** について、文字リテラルまたは数値リテラルのどちらかを指定する必要があります。文字リテラルは UTF-8 であり、64 文字の最大長をサポートし、デフォルトではブランク (空) の文字列になります。文字列リテラルは、暗黙のうちに 64 文字に切り捨てられます。数値リテラルは 0 から 9999 までの数字である必要があります、デフォルト値は 0 です。

注記

OWNER オプションは現在、適用されず、作成されるサーバー接続の所有権または操作には影響を与えません。

CREATE SERVER ステートメントは、**mysql.servers** テーブル内にエントリを作成します。これは、あとで **FEDERATED** テーブルを作成するときに **CREATE TABLE** ステートメントで使用できます。指定したオプションは、**mysql.servers** テーブルのカラムの移入に使用されます。テーブルカラムは、**Server_name**、**Host**、**Db**、**Username**、**Password**、**Port**、および **Socket** です。

例:

```

CREATE SERVER s
FOREIGN DATA WRAPPER mysql
OPTIONS (USER 'Remote', HOST '198.51.100.106', DATABASE 'test');

```

サーバーへの接続を確立するために必要なすべてのオプションを指定する必要があります。ユーザー名、ホスト名、およびデータベース名は必須です。パスワードなどの、その他のオプションも必要になる可能性があります。

このテーブルに格納されたデータは、**FEDERATED** テーブルへの接続を作成するときに使用できます。

```

CREATE TABLE t (s1 INT) ENGINE=FEDERATED CONNECTION='s';

```

詳細は、[セクション16.8「FEDERATED ストレージエンジン」](#)を参照してください。

CREATE SERVER によって暗黙的なコミットが発生します。[セクション13.3.3「暗黙的なコミットを発生させるステートメント」](#)を参照してください。

使用されているロギング形式に関係なく、**CREATE SERVER** はバイナリログに書き込まれません。

13.1.19 CREATE SPATIAL REFERENCE SYSTEM ステートメント

```

CREATE OR REPLACE SPATIAL REFERENCE SYSTEM
  srid srs_attribute ...

CREATE SPATIAL REFERENCE SYSTEM
  [IF NOT EXISTS]
  srid srs_attribute ...

srs_attribute: {
  NAME 'srs_name'
| DEFINITION 'definition'
| ORGANIZATION 'org_name' IDENTIFIED BY org_id
| DESCRIPTION 'description'
}

```

`srid, org_id: 32-bit unsigned integer`

このステートメントは、[spatial reference system](#) (SRS) 定義を作成し、データディクショナリに格納します。SUPER 権限が必要です。結果のデータディクショナリエントリは、[INFORMATION_SCHEMA.ST_SPATIAL_REFERENCE_SYSTEMS](#) テーブルを使用して検査できます。

SRID 値は一意である必要があるため、[OR REPLACE](#) も [IF NOT EXISTS](#) も指定されていない場合、指定された `srid` 値を持つ SRS 定義がすでに存在するとエラーが発生します。

[CREATE OR REPLACE](#) 構文では、SRID 値が既存のテーブルのカラムで使用されていないがぎり、同じ SRID 値を持つ既存の SRS 定義が置換されます。その場合、エラーが発生します。例:

```
mysql> CREATE OR REPLACE SPATIAL REFERENCE SYSTEM 4326 ...;
ERROR 3716 (SR005): Can't modify SRID 4326. There is at
least one column depending on it.
```

SRID を使用するカラムを識別するには、次のクエリーを使用し、4326 を作成しようとしている定義の SRID に置き換えます:

```
SELECT * FROM INFORMATION_SCHEMA.ST_GEOMETRY_COLUMNS WHERE SRS_ID=4326;
```

[CREATE ... IF NOT EXISTS](#) 構文では、SRID 値が同じ既存の SRS 定義は無視され、警告が発生します。

SRID 値は 32 ビットの符号なし整数の範囲内である必要がありますが、次の制限があります:

- SRID 0 は有効な SRID ですが、[CREATE SPATIAL REFERENCE SYSTEM](#) では使用できません。
- 値が予約された SRID 範囲内にある場合は、警告が発生します。予約済の範囲は、[0, 32767] (EPSG で予約済)、[60,000,000, 69,999,999] (EPSG で予約済) および [2,000,000,000, 2,147,483,647] (MySQL で予約済) です。EPSG は「[欧州石油調査グループ](#)」を表します。
- ユーザーは、予約された範囲内に SRID を持つ SRS を作成しないでください。これにより、MySQL に配布された将来の SRS 定義と競合する SRID のリスクが実行され、その結果、新しいシステム提供の SRS が MySQL のアップグレード用にインストールされないが、ユーザー定義の SRS が上書きされます。

ステートメントの属性は、次の条件を満たす必要があります:

- 属性は任意の順序で指定できますが、複数回指定することはできません。
- [NAME](#) および [DEFINITION](#) 属性は必須です。
- [NAME srs_name](#) 属性値は一意である必要があります。 [ORGANIZATION org_name](#) と [org_id](#) の属性値の組合せは一意である必要があります。
- [NAME srs_name](#) 属性値および [ORGANIZATION org_name](#) 属性値を空にしたり、先頭または末尾を空白にすることはできません。
- 属性指定の文字列値に、改行を含む制御文字を含めることはできません。
- 次のテーブルに、文字列属性値の最大長を示します。

表 13.6 CREATE SPATIAL REFERENCE SYSTEM 属性長

属性	最大の長さ (文字)
NAME	80
DEFINITION	4096
ORGANIZATION	256
DESCRIPTION	2048

[CREATE SPATIAL REFERENCE SYSTEM](#) ステートメントの例を次に示します。読みやすくするために、[DEFINITION](#) 値は複数の行にわたって再フォーマットされます。(ステートメントを有効にするには、実際に値を単一行に指定する必要があります。)

```
CREATE SPATIAL REFERENCE SYSTEM 4120
NAME 'Greek'
ORGANIZATION 'EPSG' IDENTIFIED BY 4120
```

DEFINITION

```
'GEOGCS["Greek",DATUM["Greek",SPHEROID["Bessel 1841",
6377397.155,299.1528128,AUTHORITY["EPSG","7004"]],
AUTHORITY["EPSG","6120"]],PRIMEM["Greenwich",0,
AUTHORITY["EPSG","8901"]],UNIT["degree",0.017453292519943278,
AUTHORITY["EPSG","9122"]],AXIS["Lat",NORTH],AXIS["Lon",EAST],
AUTHORITY["EPSG","4120"]];
```

SRS 定義の文法は、「OpenGIS 実装仕様: 座標変換サービス」、Revision 1.00、OGC 01-009、2001 年 1 月 12 日、7.2 セクションで定義されている文法に基づきます。この仕様は、<http://www.opengeospatial.org/standards/ct> で入手できます。

MySQL では、次の変更が仕様に組み込まれています:

- `<horz cs>` の本番ルール (つまり、地理的および予測される SRS) のみが実装されます。
- `<parameter>` には、オプションの非標準の `<authority>` 句があります。これにより、投影パラメータを名前ではなく権限で認識できます。
- この指定によって、GEOGCS 空間参照システム定義で `AXIS` 句が必須になることはありません。ただし、`AXIS` 句がない場合、MySQL では、定義に緯度 - 経度順または経度 - 緯度順の軸があるかどうかを判断できません。MySQL では、各 GEOGCS 定義に 2 つの `AXIS` 句を含める必要があるという非標準要件が適用されます。一方は `NORTH` または `SOUTH`、もう一方は `EAST` または `WEST` である必要があります。`AXIS` 句の順序によって、定義に緯度 - 経度順と経度 - 緯度順のどちらの軸があるかが決まります。
- SRS 定義に改行を含めることはできません。

SRS 定義で投影の権限コードが指定されている場合 (推奨)、定義に必須パラメータがないとエラーが発生します。この場合、エラーメッセージに問題の内容が示されます。MySQL でサポートされる投影方法および必須パラメータは、表 13.7 「サポートされている空間参照システム投影方法」 および 表 13.8 「空間参照システム予測パラメータ」に示されています。

MySQL の SRS 定義の記述の詳細は、「MySQL 8.0 での地理空間参照システム」および「MySQL 8.0 での空間参照システムの予測」を参照してください。

次のテーブルに、MySQL でサポートされる投影方法を示します。MySQL では不明な投影方法が許可されますが、必須パラメータの定義をチェックすることはできず、空間データを不明な投影との間で変換することもできません。式を含む各予測の動作の詳細は、「EPSG ガイドンスノート 7-2」を参照してください。

表 13.7 サポートされている空間参照システム投影方法

EPSG コード	見積名	必須パラメータ (EPSG コード)
1024	人気のある視覚化擬似 Mercator	8801, 8802, 8806, 8807
1027	ランバートアジムタル等地區 (球面)	8801, 8802, 8806, 8807
1028	均一円柱状	8823, 8802, 8806, 8807
1029	均一円筒 (球状)	8823, 8802, 8806, 8807
1041	クロヴァク語 (北方向)	8811, 8833, 1036, 8818, 8819, 8806, 8807
1042	Krovak 変更済	8811, 8833, 1036, 8818, 8819, 8806, 8807, 8617, 8618, 1026, 1027, 1028, 1029, 1030, 1031, 1032, 1033, 1034, 1035
1043	Krovak Modified (北向き)	8811, 8833, 1036, 8818, 8819, 8806, 8807, 8617, 8618, 1026, 1027, 1028, 1029, 1030, 1031, 1032, 1033, 1034, 1035
1051	Lambert Conic Conformal (2SP ミシガン)	8821, 8822, 8823, 8824, 8826, 8827, 1038
1052	コロンビア都市	8801, 8802, 8806, 8807, 1039
9801	Lambert Conic Conformal (1SP)	8801, 8802, 8805, 8806, 8807

EPSG コード	見積名	必須パラメータ (EPSG コード)
9802	Lambert Conic Conformal (2SP)	8821, 8822, 8823, 8824, 8826, 8827
9803	Lambert Conic Conformal (2SP ベルギー)	8821, 8822, 8823, 8824, 8826, 8827
9804	Mercator (バリエント A)	8801, 8802, 8805, 8806, 8807
9805	Mercator (バリエント B)	8823, 8802, 8806, 8807
9806	Cassini-Soldner	8801, 8802, 8806, 8807
9807	Transverse Mercator	8801, 8802, 8805, 8806, 8807
9808	Transverse Mercator (南向き)	8801, 8802, 8805, 8806, 8807
9809	液体ステレオグラフィ	8801, 8802, 8805, 8806, 8807
9810	極立体視 (バリエント A)	8801, 8802, 8805, 8806, 8807
9811	ニュージーランド地図グリッド	8801, 8802, 8806, 8807
9812	Hotine Oblique Mercator (バリエント A)	8811, 8812, 8813, 8814, 8815, 8806, 8807
9813	Laborde Oblique Mercator	8811, 8812, 8813, 8815, 8806, 8807
9815	Hotine Oblique Mercator (バリエント B)	8811, 8812, 8813, 8814, 8815, 8816, 8817
9816	チュニジアマイニンググリッド	8821, 8822, 8826, 8827
9817	Lambert Conic Near-Conformal	8801, 8802, 8805, 8806, 8807
9818	アメリカ領ポリコン	8801, 8802, 8806, 8807
9819	Krovak	8811, 8833, 1036, 8818, 8819, 8806, 8807
9820	ランバートアジムタル等圏	8801, 8802, 8806, 8807
9822	Albers Equal Area	8821, 8822, 8823, 8824, 8826, 8827
9824	Transverse Mercator ゾーングリッドシステム	8801, 8830, 8831, 8805, 8806, 8807
9826	ランバートコニック連合 (西方向)	8801, 8802, 8805, 8806, 8807
9828	ボンヌ (南指向)	8801, 8802, 8806, 8807
9829	極立体視 (バリエント B)	8832, 8833, 8806, 8807
9830	極立体視 (バリエント C)	8832, 8833, 8826, 8827
9831	グアム投影	8801, 8802, 8806, 8807
9832	修正済方程式	8801, 8802, 8806, 8807
9833	双曲線カシーニ=ソルドナー	8801, 8802, 8806, 8807
9834	ランバートシリンダル等面 (球面)	8823, 8802, 8806, 8807
9835	ランバートシリンダル等面	8823, 8802, 8806, 8807

次のテーブルに、MySQL で認識される投影パラメータを示します。認識は主に権限コードによって行われます。認証局コードがない場合、MySQL は、パラメータ名に一致する大/小文字を区別しない文字列にフォールバックします。各パラメータの詳細は、「[EPSG オンラインレジストリ](#)」のコードを参照してください。

表 13.8 空間参照システム予測パラメータ

EPSG コード	フォールバック名 (MySQL で認識)	EPSG 名
1026	c1	C1
1027	c2	C2
1028	c3	C3

EPSG コード	フォールバック名 (MySQL で認識)	EPSG 名
1029	c4	C4
1030	c5	C5
1031	c6	C6
1032	c7	C7
1033	c8	C8
1034	c9	C9
1035	c10	C10
1036	方位角	円錐軸の同一レベル
1038	ellipsoid_scale_factor	楕円体スケーリング係数
1039	projection_plane_height_at_origin	投影平面原点の高さ
8617	evaluation_point_ordinate_1	評価ポイントの序数 1
8618	evaluation_point_ordinate_2	評価ポイントの序数 2
8801	latitude_of_origin	自然起点の緯度
8802	central_meridian	自然原点の経度
8805	scale_factor	自然原点のスケール係数
8806	false_easting	偽東座標
8807	false_northing	偽北座標
8811	latitude_of_center	投影センターの緯度
8812	longitude_of_center	投影中心の経度
8813	方位角	最初の行の方位角
8814	rectified_grid_angle	ジオメトリ補正からスキューグリッドまでの角度
8815	scale_factor	初期線の尺度係数
8816	false_easting	投影中心の東座標
8817	false_northing	投影中心の北座標
8818	pseudo_standard_parallel_1	擬似標準並列度
8819	scale_factor	擬似標準パラレルのスケール係数
8821	latitude_of_origin	偽起点の緯度
8822	central_meridian	偽原点の経度
8823	standard_parallel_1, standard_parallel1	第 1 標準平行度
8824	standard_parallel_2, standard_parallel2	第 2 標準並列度
8826	false_easting	偽原点の東座標
8827	false_northing	偽原点での北座標
8830	initial_longitude	初期経度
8831	zone_width	ゾーンの幅
8832	standard_parallel	標準並列度
8833	longitude_of_center	原点の経度

13.1.20 CREATE TABLE ステートメント

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
(create_definition,...)
```



```
[table_options]
[partition_options]

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
[(create_definition,...)]
[table_options]
[partition_options]
[IGNORE | REPLACE]
[AS] query_expression

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
{ LIKE old_tbl_name | (LIKE old_tbl_name) }

create_definition: {
  col_name column_definition
| {INDEX | KEY} [index_name] [index_type] (key_part,...)
  [index_option] ...
| {FULLTEXT | SPATIAL} [INDEX | KEY] [index_name] (key_part,...)
  [index_option] ...
| [CONSTRAINT [symbol]] PRIMARY KEY
  [index_type] (key_part,...)
  [index_option] ...
| [CONSTRAINT [symbol]] UNIQUE [INDEX | KEY]
  [index_name] [index_type] (key_part,...)
  [index_option] ...
| [CONSTRAINT [symbol]] FOREIGN KEY
  [index_name] (col_name,...)
  reference_definition
| check_constraint_definition
}

column_definition: {
  data_type [NOT NULL | NULL] [DEFAULT {literal | (expr)}]
  [VISIBLE | INVISIBLE]
  [AUTO_INCREMENT] [UNIQUE [KEY]] [[PRIMARY] KEY]
  [COMMENT 'string']
  [COLLATE collation_name]
  [COLUMN_FORMAT {FIXED | DYNAMIC | DEFAULT}]
  [ENGINE_ATTRIBUTE [=] 'string']
  [SECONDARY_ENGINE_ATTRIBUTE [=] 'string']
  [STORAGE {DISK | MEMORY}]
  [reference_definition]
  [check_constraint_definition]
| data_type
  [COLLATE collation_name]
  [GENERATED ALWAYS] AS (expr)
  [VIRTUAL | STORED] [NOT NULL | NULL]
  [VISIBLE | INVISIBLE]
  [UNIQUE [KEY]] [[PRIMARY] KEY]
  [COMMENT 'string']
  [reference_definition]
  [check_constraint_definition]
}

data_type:
  (see 第11章「データ型」)

key_part: {col_name [(length)] | (expr)} [ASC | DESC]

index_type:
  USING {BTREE | HASH}

index_option: {
  KEY_BLOCK_SIZE [=] value
| index_type
| WITH PARSER parser_name
| COMMENT 'string'
| {VISIBLE | INVISIBLE}
| ENGINE_ATTRIBUTE [=] 'string'
| SECONDARY_ENGINE_ATTRIBUTE [=] 'string'
}

check_constraint_definition:
  [CONSTRAINT [symbol]] CHECK (expr) [[NOT] ENFORCED]
```

```

reference_definition:
REFERENCES tbl_name (key_part,...)
[MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]
[ON DELETE reference_option]
[ON UPDATE reference_option]

reference_option:
RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT

table_options:
table_option [[,] table_option] ...

table_option: {
  AUTOEXTEND_SIZE [=] value
| AUTO_INCREMENT [=] value
| AVG_ROW_LENGTH [=] value
| [DEFAULT] CHARACTER SET [=] charset_name
| CHECKSUM [=] {0 | 1}
| [DEFAULT] COLLATE [=] collation_name
| COMMENT [=] 'string'
| COMPRESSION [=] {'ZLIB' | 'LZ4' | 'NONE'}
| CONNECTION [=] 'connect_string'
| {DATA | INDEX} DIRECTORY [=] 'absolute path to directory'
| DELAY_KEY_WRITE [=] {0 | 1}
| ENCRYPTION [=] {'Y' | 'N'}
| ENGINE [=] engine_name
| ENGINE_ATTRIBUTE [=] 'string'
| INSERT_METHOD [=] { NO | FIRST | LAST }
| KEY_BLOCK_SIZE [=] value
| MAX_ROWS [=] value
| MIN_ROWS [=] value
| PACK_KEYS [=] {0 | 1 | DEFAULT}
| PASSWORD [=] 'string'
| ROW_FORMAT [=] {DEFAULT | DYNAMIC | FIXED | COMPRESSED | REDUNDANT | COMPACT}
| SECONDARY_ENGINE_ATTRIBUTE [=] 'string'
| STATS_AUTO_RECALC [=] {DEFAULT | 0 | 1}
| STATS_PERSISTENT [=] {DEFAULT | 0 | 1}
| STATS_SAMPLE_PAGES [=] value
| TABLESPACE tablespace_name [STORAGE {DISK | MEMORY}]
| UNION [=] (tbl_name[,tbl_name]...)
}

partition_options:
PARTITION BY
{ [LINEAR] HASH(expr)
| [LINEAR] KEY [ALGORITHM={1 | 2}] (column_list)
| RANGE{(expr) | COLUMNS(column_list)}
| LIST{(expr) | COLUMNS(column_list)} }
[PARTITIONS num]
[SUBPARTITION BY
{ [LINEAR] HASH(expr)
| [LINEAR] KEY [ALGORITHM={1 | 2}] (column_list) }
[SUBPARTITIONS num]
]
[[partition_definition [, partition_definition] ...]]

partition_definition:
PARTITION partition_name
[VALUES
{LESS THAN {(expr | value_list) | MAXVALUE}
|
IN (value_list)}]
[[STORAGE] ENGINE [=] engine_name]
[COMMENT [=] 'string' ]
[DATA DIRECTORY [=] 'data_dir']
[INDEX DIRECTORY [=] 'index_dir']
[MAX_ROWS [=] max_number_of_rows]
[MIN_ROWS [=] min_number_of_rows]
[TABLESPACE [=] tablespace_name]
[[subpartition_definition [, subpartition_definition] ...]]

subpartition_definition:
SUBPARTITION logical_name
[[STORAGE] ENGINE [=] engine_name]
[COMMENT [=] 'string' ]

```

```
[DATA DIRECTORY [=] 'data_dir']  
[INDEX DIRECTORY [=] 'index_dir']  
[MAX_ROWS [=] max_number_of_rows]  
[MIN_ROWS [=] min_number_of_rows]  
[TABLESPACE [=] tablespace_name]
```

query_expression:

```
SELECT ... (Some valid select or union statement)
```

CREATE TABLE は、指定された名前を持つテーブルを作成します。このテーブルに対する **CREATE** 権限が必要です。

デフォルトでは、テーブルは **InnoDB** ストレージエンジンを使用してデフォルトデータベースに作成されます。テーブルがすでに存在する場合、デフォルトデータベースが存在しない場合、またはデータベースが存在しない場合はエラーが発生します。

MySQL にはテーブル数の制限はありません。ベースとなるファイルシステムによっては、テーブルを表すファイル数に制限がある場合があります。個々のストレージエンジンには、エンジン固有の制約が課される場合があります。**InnoDB** では、最大 40 億個のテーブルを使用できます。

テーブルの物理表現の詳細は、[セクション13.1.20.1「CREATE TABLE によって作成されるファイル」](#) を参照してください。

このセクションの次のトピックで説明するように、**CREATE TABLE** ステートメントにはいくつかの側面があります:

- [テーブル名](#)
- [一時テーブル](#)
- [テーブルのクローニングおよびコピー](#)
- [カラムのデータ型および属性](#)
- [インデックス、外部キーおよび CHECK 制約](#)
- [テーブルオプション](#)
- [テーブルのパーティション化](#)

テーブル名

- `tbl_name`

特定のデータベース内にテーブルを作成するには、テーブル名を `db_name.tbl_name` として指定できます。そのデータベースが存在すると仮定すると、これは、デフォルトデータベースが存在するかどうかには関係なく機能します。引用符で囲まれた識別子を使用する場合は、データベース名とテーブル名を個別に引用符で囲みます。たとえば、``mydb.mytbl`` ではなく、``mydb`.`mytbl`` と記述します。

許可されるテーブル名のルールは、[セクション9.2「スキーマオブジェクト名」](#) に示されています。

- **IF NOT EXISTS**

テーブルが存在する場合にエラーが発生しないようにします。ただし、既存のテーブルの構造が **CREATE TABLE** ステートメントによって示されている構造と同一であることの検証は行われません。

一時テーブル

テーブルの作成時に **TEMPORARY** キーワードを使用できます。**TEMPORARY** テーブルは現在のセッション内でのみ表示され、セッションがクローズされると自動的に削除されます。詳細は、[セクション13.1.20.2「CREATE TEMPORARY TABLE ステートメント」](#) を参照してください。

テーブルのクローニングおよびコピー

- **LIKE**

`CREATE TABLE ... LIKE` を使用して、元のテーブルに定義されているカラム属性やインデックスなど、別のテーブルの定義に基づいて空のテーブルを作成します:

```
CREATE TABLE new_tbl LIKE orig_tbl;
```

詳細は、[セクション13.1.20.3「CREATE TABLE ... LIKE ステートメント」](#)を参照してください。

- `[AS] query_expression`

別のテーブルからテーブルを作成するには、`CREATE TABLE` ステートメントの最後に `SELECT` ステートメントを追加します:

```
CREATE TABLE new_tbl AS SELECT * FROM orig_tbl;
```

詳細は、[セクション13.1.20.4「CREATE TABLE ... SELECT ステートメント」](#)を参照してください。

- `IGNORE | REPLACE`

`IGNORE` および `REPLACE` オプションは、`SELECT` ステートメントを使用してテーブルをコピーするときに一意キー値を複製する行の処理方法を示します。

詳細は、[セクション13.1.20.4「CREATE TABLE ... SELECT ステートメント」](#)を参照してください。

カラムのデータ型および属性

テーブルあたり 4096 カラムという強い制限値がありますが、特定のテーブルでは、実際の最大数がこれより少なくなる可能性があります。実際の最大数は、[セクション8.4.7「テーブルカラム数と行サイズの制限」](#)で説明されている要因によって異なります。

- `data_type`

`data_type` は、カラム定義のデータ型を表します。カラムのデータ型の指定に使用できる構文の詳細、および各型のプロパティの詳細は、[第11章「データ型」](#)を参照してください。

- 属性の中には、すべてのデータ型には適用されないものがあります。`AUTO_INCREMENT` は、整数型と浮動小数点型にのみ適用されます。MySQL 8.0.13 より前は、`DEFAULT` は `BLOB`、`TEXT`、`GEOMETRY` および `JSON` タイプには適用されません。
- 文字データ型 (`CHAR`、`VARCHAR`、`TEXT` 型、`ENUM`、`SET` およびシノニム) には、`CHARACTER SET` を含めてカラムの文字セットを指定できます。`CHARSET` は `CHARACTER SET` のシノニムです。文字セットの照合順序は、他の属性とともに `COLLATE` 属性とともに指定できます。詳細は、[第10章「文字セット、照合順序、Unicode」](#)を参照してください。例:

```
CREATE TABLE t (c CHAR(20) CHARACTER SET utf8 COLLATE utf8_bin);
```

MySQL 8.0 は、文字カラム定義内の長さの指定を文字数で解釈します。`BINARY` と `VARBINARY` の長さはバイト単位です。

- `CHAR`、`VARCHAR`、`BINARY`、および `VARBINARY` カラムの場合は、`col_name(length)` 構文を使用してインデックスプリフィクス長を指定することにより、カラム値の先頭の部分のみを使用するインデックスを作成できます。`BLOB` および `TEXT` カラムにもインデックスを設定できますが、プリフィクス長を指定する必要があります。プリフィクス長は、バイナリ以外の文字列型の場合は文字数で、バイナリ文字列型の場合はバイト単位で指定されます。つまり、インデックスエントリは、`CHAR`、`VARCHAR`、および `TEXT` カラムの場合は各カラム値の最初の `length` 文字、`BINARY`、`VARBINARY`、および `BLOB` カラムの場合は各カラム値の最初の `length` バイトで構成されます。このようにカラム値のプリフィクスのみでインデックスを設定すると、インデックスファイルをはるかに小さくできます。インデックス接頭辞の詳細は、[セクション13.1.15「CREATE INDEX ステートメント」](#)を参照してください。

`BLOB` および `TEXT` カラム上のインデックス設定をサポートするのは、`InnoDB` および `MyISAM` ストレージエンジンだけです。例:

```
CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
```

指定したインデックス接頭辞がカラムの最大データ型サイズを超える場合、`CREATE TABLE` は次のようにインデックスを処理します:

- 一意でないインデックスの場合は、エラーが発生するか (厳密な SQL モードが有効な場合)、インデックスの長さが最大カラムデータ型サイズ内になるように縮小され、警告が生成されず (厳密な SQL モードが有効でない場合)。
 - 一意インデックスの場合、インデックスの長さを短くすると、指定した一意性要件を満たさない一意でないエントリの挿入が可能になるため、SQL モードに関係なくエラーが発生します。
 - `JSON` カラムはインデックス付けできません。この制限を回避するには、生成されたカラムに `JSON` カラムからスカラー値を抽出するインデックスを作成します。詳細な例は、[JSON カラムインデックスを提供するための生成されたカラムのインデックス付け](#) を参照してください。
- `NOT NULL | NULL`

`NULL` と `NOT NULL` のどちらも指定されていない場合、そのカラムは `NULL` が指定されたかのように処理されず。

MySQL 8.0 では、`InnoDB`、`MyISAM` および `MEMORY` ストレージエンジンのみが `NULL` 値を持つことができるカラムのインデックスをサポートしています。それ以外の場合は、インデックス付きカラムを `NOT NULL` として宣言する必要があります。そうしないと、エラー結果が発生します。

- `DEFAULT`

カラムのデフォルト値を指定します。カラム定義に明示的な `DEFAULT` 値が含まれていない場合など、デフォルト値の処理の詳細は、[セクション11.6「データ型デフォルト値」](#) を参照してください。

`NO_ZERO_DATE` または `NO_ZERO_IN_DATE` SQL モードが有効になっているときに、日付の値のデフォルトがそのモードに従って正しくない場合、`CREATE TABLE` では厳密な SQL モードが有効になっていない場合は警告を、厳密モードが有効になっている場合はエラーを生成します。たとえば、`NO_ZERO_IN_DATE` が有効になっている場合は、`c1 DATE DEFAULT '2010-00-00'` によって警告が生成されます。

- `VISIBLE, INVISIBLE`

カラムの可視性を指定します。どちらのキーワードも存在しない場合、デフォルトは `VISIBLE` です。テーブルには、少なくとも 1 つの表示可能なカラムが必要です。すべてのカラムを非表示にしようとすると、エラーが発生します。詳細は、[セクション13.1.20.10「非表示カラム」](#) を参照してください。

`VISIBLE` および `INVISIBLE` キーワードは、MySQL 8.0.23 の時点で使用できます。MySQL 8.0.23 より前は、すべてのカラムが表示されます。

- `AUTO_INCREMENT`

整数または浮動小数点のカラムには、追加の属性 `AUTO_INCREMENT` を指定できます。インデックスが設定された `AUTO_INCREMENT` カラムに `NULL` (推奨) または `0` の値を挿入すると、カラムは次のシーケンス値に設定されます。通常、これは `value+1` です。ここで `value` は現在テーブルにあるカラムの最大値です。`AUTO_INCREMENT` シーケンスは `1` で始まります。

行を挿入したあとに `AUTO_INCREMENT` 値を取得するには、`LAST_INSERT_ID()` SQL 関数または `mysql_insert_id()` C API 関数を使用します。[セクション12.16「情報関数」](#) および [mysql_insert_id\(\)](#) を参照してください。

`NO_AUTO_VALUE_ON_ZERO` SQL モードが有効になっている場合は、新しいシーケンス値を生成することなく、`0` を `AUTO_INCREMENT` カラム内に `0` として格納できます。[セクション5.1.11「サーバー SQL モード」](#) を参照してください。

テーブルごとに存在できる `AUTO_INCREMENT` カラムは 1 つだけです。このカラムはインデックス付きである必要があり、`DEFAULT` 値を割り当てることはできません。`AUTO_INCREMENT` カラムは、正の値だけが含まれている場合のみ正しく機能します。負の数や `0` を挿入すると、非常に大きな正の数を挿入したと見なされます。これは、数字が正から負に「ラップする」ときの精度の問題を回避すると同時に、`0` を含む `AUTO_INCREMENT` カラムを誤って取得してしまわないようにするために行われます。

MyISAM テーブルの場合は、マルチカラムキー内の `AUTO_INCREMENT` セカンダリカラムを指定できます。 [セクション3.6.9「AUTO_INCREMENT の使用」](#) を参照してください。

MySQL を一部の ODBC アプリケーションと互換性があるようにするために、次のクエリーを使用して、最後に挿入された行の `AUTO_INCREMENT` 値を見つけることができます。

```
SELECT * FROM tbl_name WHERE auto_col IS NULL
```

このメソッドでは、`sql_auto_is_null` 変数が 0 に設定されていない必要があります。 [セクション5.1.8「サーバーシステム変数」](#) を参照してください。

InnoDB と `AUTO_INCREMENT` については、 [セクション15.6.1.6「InnoDB での AUTO_INCREMENT 処理」](#) を参照してください。 `AUTO_INCREMENT` と MySQL レプリケーションについては、 [セクション17.5.1.1「レプリケーションと AUTO_INCREMENT」](#) を参照してください。

- **COMMENT**

カラムのコメントは、`COMMENT` オプションで 1024 文字以内で指定できます。このコメントは、`SHOW CREATE TABLE` および `SHOW FULL COLUMNS` ステートメントによって表示されます。

- **COLUMN_FORMAT**

NDB Cluster では、`COLUMN_FORMAT` を使用して NDB テーブルの個々のカラムのデータストレージ形式を指定することもできます。許可されるカラムフォーマットは、`FIXED`、`DYNAMIC`、および `DEFAULT` です。`FIXED` は固定幅記憶域の指定に使用され、`DYNAMIC` はカラムを可変幅にすることを許可し、`DEFAULT` はカラムのデータ型によって決定される固定幅記憶域または可変幅記憶域 (`ROW_FORMAT` 指定子によってオーバーライドされる可能性がある) を使用します。

NDB テーブルの場合、`COLUMN_FORMAT` のデフォルト値は `FIXED` です。

NDB Cluster では、`COLUMN_FORMAT=FIXED` で定義されたカラムの可能な最大オフセットは 8188 バイトです。詳細および考えられる回避策については、 [セクション23.1.7.5「NDB Cluster 内のデータベースオブジェクトに関連付けられる制限」](#) を参照してください。

`COLUMN_FORMAT` は現在、NDB 以外のストレージエンジンを使用しているテーブルのカラムには影響を与えません。MySQL 8.0 は、`COLUMN_FORMAT` を暗黙的に無視します。

- **ENGINE_ATTRIBUTE** および **SECONDARY_ENGINE_ATTRIBUTE** オプション (MySQL 8.0.21 の時点で使用可能) を使用して、プライマリおよびセカンダリストレージエンジンのカラム属性を指定します。オプションは、将来の使用のために予約されています。

許可される値は、有効な `JSON` ドキュメントまたは空の文字列 ("") を含む文字列リテラルです。無効な `JSON` が拒否されました。

```
CREATE TABLE t1 (c1 INT ENGINE_ATTRIBUTE='{"key":"value"}');
```

`ENGINE_ATTRIBUTE` および `SECONDARY_ENGINE_ATTRIBUTE` の値は、エラーなしで繰り返すことができます。この場合、最後に指定した値が使用されます。

`ENGINE_ATTRIBUTE` および `SECONDARY_ENGINE_ATTRIBUTE` の値は、サーバーによってチェックされず、テーブルストレージエンジンが変更されたときにもクリアされません。

- **STORAGE**

NDB テーブルの場合、`STORAGE` 句を使用して、カラムをディスクに格納するかメモリーに格納するかを指定できます。`STORAGE DISK` を指定するとカラムはディスク上に格納され、`STORAGE MEMORY` を指定するとインメモリーストレージが使用されます。使用される `CREATE TABLE` ステートメントには、引き続き `TABLESPACE` 句が含まれている必要があります。

```
mysql> CREATE TABLE t1 (  
-> c1 INT STORAGE DISK,  
-> c2 INT STORAGE MEMORY  
->) ENGINE NDB;  
ERROR 1005 (HY000): Can't create table 'c.t1' (errno: 140)
```



```
mysql> CREATE TABLE t1 (  
-> c1 INT STORAGE DISK,  
-> c2 INT STORAGE MEMORY  
->) TABLESPACE ts_1 ENGINE NDB;  
Query OK, 0 rows affected (1.06 sec)
```

NDB テーブルの場合、`STORAGE DEFAULT` は `STORAGE MEMORY` と同等です。

`STORAGE` 句は、NDB 以外のストレージエンジンを使用しているテーブルには影響を与えません。`STORAGE` キーワードは NDB Cluster で提供される `mysql` の構築でのみサポートされます。ほかのバージョンの MySQL では認識されず、`STORAGE` キーワードを使用しようとすると構文エラーが発生します。

- `GENERATED ALWAYS`

生成されたカラム式を指定するために使用します。`generated columns` の詳細は、[セクション13.1.20.8「CREATE TABLE および生成されるカラム」](#) を参照してください。

`Stored generated columns` はインデックス付けできます。InnoDB は、`virtual generated columns` でセカンダリインデックスをサポートしています。[セクション13.1.20.9「セカンダリインデックスと生成されたカラム」](#) を参照してください。

インデックス、外部キーおよび CHECK 制約

インデックス、外部キーおよび `CHECK` 制約の作成には、いくつかのキーワードが適用されます。次の説明に加えて、一般的な背景は、[セクション13.1.15「CREATE INDEX ステートメント」](#)、[セクション13.1.20.5「FOREIGN KEY の制約」](#) および [セクション13.1.20.6「CHECK 制約」](#) を参照してください。

- `CONSTRAINT symbol`

`CONSTRAINT symbol` 句を指定して制約に名前を付けることができます。句が指定されていない場合、または `CONSTRAINT` キーワードの後に `symbol` が含まれていない場合、MySQL では制約名が自動的に生成されますが、次に示す例外があります。`symbol` 値 (使用する場合は、制約タイプごとにスキーマ (データベース) ごとに一意である必要があります。`symbol` が重複すると、エラーになります。[セクション9.2.1「識別子の長さ制限」](#) で生成される制約識別子の長さ制限に関する説明も参照してください。

注記

外部キー定義に `CONSTRAINT symbol` 句が指定されていない場合、または `CONSTRAINT` キーワードの後に `symbol` が含まれていない場合、MySQL は MySQL 8.0.15 までの外部キーインデックス名を使用し、その後に制約名を自動的に生成します。

SQL 標準では、すべてのタイプの制約 (主キー、一意インデックス、外部キー、チェック) が同じ名前スペースに属することが指定されています。MySQL では、各制約タイプにスキーマごとに独自の名前スペースがあります。したがって、各タイプの制約の名前はスキーマごとに一意である必要がありますが、異なるタイプの制約には同じ名前を付けることができます。

- `PRIMARY KEY`

すべてのキーカラムを `NOT NULL` として定義する必要がある一意のインデックス。それらが `NOT NULL` として明示的に宣言されていない場合、MySQL は、それらを暗黙的に (かつ警告なしで) そのように宣言します。テーブルに存在できる `PRIMARY KEY` は 1 つだけです。`PRIMARY KEY` の名前は、常に `PRIMARY` です。そのため、これをその他のどの種類のインデックスの名前としても使用できません。

`PRIMARY KEY` が存在しないときに、アプリケーションがテーブル内の `PRIMARY KEY` を要求した場合、MySQL は、`NULL` カラムのない最初の `UNIQUE` インデックスを `PRIMARY KEY` として返します。

InnoDB テーブルでは、セカンダリインデックスのためのストレージのオーバーヘッドを最小限に抑えるために、`PRIMARY KEY` を短い値に維持してください。各セカンダリインデックスエントリには、対応する行の主キーカラムのコピーが含まれています。([セクション15.6.2.1「クラスティンデックスとセカンダリインデックス」](#) を参照してください。)

作成されたテーブルでは、**PRIMARY KEY** が最初に配置され、そのあとにすべての **UNIQUE** インデックス、さらに一意でないインデックスが続きます。これは、MySQL オプティマイザが、使用するインデックスに優先順位を付けたり、重複した **UNIQUE** キーをよりすばやく検出したりするのに役立ちます。

PRIMARY KEY をマルチカラムインデックスにすることができます。ただし、カラム指定で **PRIMARY KEY** キー属性を使用してマルチカラムインデックスを作成することはできません。それを行なっても、その単一カラムがプライマリとしてマークされるだけです。別の **PRIMARY KEY(key_part, ...)** 句を使用する必要があります。

テーブルに整数型の単一カラムで構成される **PRIMARY KEY** または **UNIQUE NOT NULL** インデックスがある場合、**一意インデックス** で説明されているように、`_rowid` を使用して **SELECT** ステートメントのインデックス付きカラムを参照できます。

MySQL では、**PRIMARY KEY** の名前は **PRIMARY** です。その他のインデックスでは、名前を割り当てなかった場合、そのインデックスには最初のインデックス付きカラムと同じ名前が割り当てられ、それを一意にするためにオプションのサフィクス (`_2`、`_3`、...) が付けられます。テーブルのインデックス名は、**SHOW INDEX FROM tbl_name** を使用して確認できます。 [セクション13.7.7.22「SHOW INDEX ステートメント」](#) を参照してください。

- **KEY | INDEX**

KEY は通常、**INDEX** のシノニムです。キー属性 **PRIMARY KEY** もまた、カラム定義内で指定する場合は、単に **KEY** として指定できます。これは、ほかのデータベースシステムとの互換性のために実装されました。

- **UNIQUE**

UNIQUE インデックスは、そのインデックス内のすべての値が異なっている必要があるという制約を作成します。既存の行に一致するキー値を持つ新しい行を追加しようとすると、エラーが発生します。すべてのエンジンについて、**UNIQUE** インデックスは、**NULL** を含むことができるカラムでの複数の **NULL** 値を許可します。**UNIQUE** インデックスのカラムに接頭辞値を指定する場合、カラム値は接頭辞の長さ内で一意である必要があります。

テーブルに整数型の単一カラムで構成される **PRIMARY KEY** または **UNIQUE NOT NULL** インデックスがある場合、**一意インデックス** で説明されているように、`_rowid` を使用して **SELECT** ステートメントのインデックス付きカラムを参照できます。

- **FULLTEXT**

FULLTEXT インデックスは、全文検索に使用される特別なタイプのインデックスです。**FULLTEXT** インデックスをサポートするのは、**InnoDB** および **MyISAM** だけです。これらは、**CHAR**、**VARCHAR**、および **TEXT** カラムからのみ作成できます。インデックス設定は常に、カラム全体に対して実行されます。カラムプリフィクスのインデックス設定はサポートされていないため、プリフィクス長が指定されてもすべて無視されます。操作の詳細は、[セクション12.10「全文検索関数」](#) を参照してください。**WITH PARSER** 句は、全文インデックス設定および検索操作に特殊な処理が必要な場合にパーサープラグインをインデックスに関連付けるために、`index_option` 値として指定できます。この句は、**FULLTEXT** インデックスに対してのみ有効です。**InnoDB** および **MyISAM** は、フルテキストパーサープラグインをサポートしています。詳細は、[Full-Text Parser Plugins](#) および [Writing Full-Text Parser Plugins](#) を参照してください。

- **SPATIAL**

空間データ型に **SPATIAL** インデックスを作成できます。空間型は **InnoDB** および **MyISAM** テーブルでのみサポートされており、インデックス付けされたカラムは **NOT NULL** として宣言する必要があります。[セクション11.4「空間データ型」](#) を参照してください。

- **FOREIGN KEY**

MySQL は、関連データのテーブルにまたがる相互参照を可能にする外部キーと、この分散したデータの整合性を維持するために役立つ外部キー制約をサポートします。定義およびオプションの情報は、[reference_definition](#) および [reference_option](#) を参照してください。

InnoDB ストレージエンジンを使用するパーティション化されたテーブルは、外部キーをサポートしていません。詳細については、[セクション24.6「パーティショニングの制約と制限」](#) を参照してください。

- CHECK

CHECK 句を使用すると、テーブルの行のデータ値について制約を作成できます。 [セクション13.1.20.6「CHECK 制約」](#)を参照してください。

- key_part

- key_part 仕様の末尾には、ASC または DESC を使用して、インデックス値を昇順または降順のどちらで格納するかを指定できます。順序指定子が指定されていない場合、デフォルトは昇順です。

- length 属性で定義される接頭辞の長さは、REDUNDANT または COMPACT の行形式を使用する InnoDB テーブルでは最大 767 バイトです。DYNAMIC または COMPRESSED の行形式を使用する InnoDB テーブルでは、接頭辞の長さの制限は 3072 バイトです。MyISAM テーブルの場合、接頭辞の長さの制限は 1000 バイトです。

接頭辞 limits はバイト単位で測定されます。ただし、CREATE TABLE、ALTER TABLE および CREATE INDEX ステートメントのインデックス指定の接頭辞 lengths は、非バイナリ文字列型 (CHAR, VARCHAR, TEXT) の場合は文字数として解釈され、バイナリ文字列型 (BINARY, VARBINARY, BLOB) の場合はバイト数として解釈されます。マルチバイト文字セットを使用する非バイナリ文字列カラムに接頭辞の長さを指定する場合は、これを考慮してください。

- MySQL 8.0.17 以降、key_part 仕様の expr では、(CAST json_path AS type ARRAY) の形式を使用して JSON カラムに複数值インデックスを作成できます。複数值インデックスでは、複数值インデックスの作成、使用方法、制限および制限に関する詳細情報を提供します。

- index_type

一部のストレージエンジンでは、インデックスの作成時にインデックスタイプを指定できます。index_type 指定子の構文は、USING type_name です。

例:

```
CREATE TABLE lookup
(id INT, INDEX USING BTREE (id))
ENGINE = MEMORY;
```

USING の推奨される位置は、インデックスカラムリストのあとです。カラムリストの前に指定できますが、その位置でのオプションの使用のサポートは非推奨であるため、将来の MySQL リリースで削除される予定です。

- index_option

index_option 値は、インデックスの追加オプションを指定します。

- KEY_BLOCK_SIZE

MyISAM テーブルの場合、KEY_BLOCK_SIZE はオプションで、インデックスキープロックに使用するサイズをバイト単位で指定します。この値はヒントとして扱われます。必要に応じて、異なるサイズが使用される可能性があります。個々のインデックス定義に指定された KEY_BLOCK_SIZE 値は、テーブルレベルの KEY_BLOCK_SIZE 値をオーバーライドします。

テーブルレベルの KEY_BLOCK_SIZE 属性の詳細は、[テーブルオプション](#)を参照してください。

- WITH PARSER

WITH PARSER オプションは、FULLTEXT インデックスでのみ使用できます。これは、全文インデックス設定および検索操作に特殊な処理が必要な場合に、パーサープラグインをインデックスに関連付けます。InnoDB および MyISAM は、フルテキストパーサープラグインをサポートしています。フルテキストパーサープラグイン

が関連付けられた **MyISAM** テーブルがある場合は、**ALTER TABLE** を使用してテーブルを **InnoDB** に変換できません。

- **COMMENT**

インデックス定義には、最大 1024 文字のオプションのコメントを含めることができます。

index_option COMMENT 句を使用して、個々のインデックスに **InnoDB MERGE_THRESHOLD** 値を設定できます。 [セクション15.8.11「インデックスページのマージしきい値の構成」](#) を参照してください。

- **VISIBLE, INVISIBLE**

インデックスの可視性を指定します。インデックスはデフォルトで可視化されます。不可視インデックスはオプティマイザでは使用されません。インデックスの可視性の指定は、主キー以外のインデックス (明示的または暗黙的) に適用されます。詳細は、 [セクション8.3.12「不可視のインデックス」](#) を参照してください。

- **ENGINE_ATTRIBUTE** および **SECONDARY_ENGINE_ATTRIBUTE** オプション (MySQL 8.0.21 の時点で使用可能) は、プライマリストレージエンジンおよびセカンダリストレージエンジンのインデックス属性を指定するために使用されます。オプションは、将来の使用のために予約されています。

許容される **index_option** 値の詳細は、 [セクション13.1.15「CREATE INDEX ステートメント」](#) を参照してください。インデックスの詳細は、 [セクション8.3.1「MySQL のインデックスの使用の仕組み」](#) を参照してください。

- **reference_definition**

reference_definition 構文の詳細および例は、 [セクション13.1.20.5「FOREIGN KEY の制約」](#) を参照してください。

InnoDB および **NDB** テーブルは、外部キー制約のチェックをサポートしています。参照されるテーブルのカラムには、常に明示的に名前を付ける必要があります。外部キーに対しては **ON DELETE** と **ON UPDATE** の両方のアクションがサポートされています。詳細および例については、 [セクション13.1.20.5「FOREIGN KEY の制約」](#) を参照してください。

その他のストレージエンジンの場合、MySQL Server は、**CREATE TABLE** ステートメント内の **FOREIGN KEY** および **REFERENCES** 構文を解析して無視します。 [セクション1.7.2.3「FOREIGN KEY 制約の違い」](#) を参照してください。

重要

ANSI/ISO SQL 標準に精通しているユーザーの場合は、参照整合性の制約定義で使用される **MATCH** 句を認識または適用するストレージエンジンは (**InnoDB** を含め) 存在しません。明示的な **MATCH** 句を使用しても効果はなく、**ON DELETE** 句および **ON UPDATE** 句も無視されます。これらの理由により、**MATCH** の指定は避けるようにしてください。

SQL 標準での **MATCH** 句は、複合 (マルチカラム) 外部キー内の **NULL** 値が、主キーとの比較時にどのように処理されるかを制御します。**InnoDB** は基本的に、外部キーをすべてまたは部分的に **NULL** にすることが許可される、**MATCH SIMPLE** で定義されるセマンティクスを実装しています。その場合は、このような外部キーを含む (子テーブルの) 行の挿入が許可され、その行は参照される (親) テーブル内のどの行にも一致しません。トリガーを使用して、ほかのセマンティクスを実装できます。

さらに、MySQL ではパフォーマンスのために、参照されるカラムにインデックスを設定する必要があります。ただし、**InnoDB** では、参照カラムを **UNIQUE** または **NOT NULL** として宣言する必要はありません。一意でないキーまたは **NULL** 値を含むキーへの外部キー参照の処理は、**UPDATE** や **DELETE CASCADE** などの操作に対して適切に定義されていません。**UNIQUE** (または **PRIMARY**) と **NOT NULL** の両方であるキーのみを参照する外部キーを使用することをお勧めします。

MySQL は、参照がカラム指定の一部として定義されている「**インライン REFERENCES** 仕様」 (SQL 標準で定義されているもの) を解析しますが、無視します。MySQL は、個別の **FOREIGN KEY** 指定の一部として指定されている場合にのみ **REFERENCES** 句を受け入れます。

- **reference_option**

RESTRICT, CASCADE, SET NULL, NO ACTION および SET DEFAULT オプションの詳細は、[セクション 13.1.20.5 「FOREIGN KEY の制約」](#) を参照してください。

テーブルオプション

テーブルオプションは、テーブルの動作を最適化するために使用します。ほとんどの場合は、それらのうちのどれも指定する必要はありません。特に示されていないかぎり、これらのオプションはすべてのストレージエンジンに適用されます。特定のストレージエンジンに適用されないオプションは、テーブル定義の一部として受け入れられ、記憶される可能性があります。それにより、あとで ALTER TABLE を使用して、別のストレージエンジンを使用するようにテーブルを変換した場合に、このようなオプションが適用されます。

• ENGINE

次のテーブルに示すいずれかの名前を使用して、テーブルのストレージエンジンを指定します。エンジン名は、引用符で囲んでも囲まなくてもかまいません。引用符で囲まれた名前 'DEFAULT' は認識されますが、無視されません。

ストレージエンジン	説明
InnoDB	行ロックと外部キーを備えたトランザクションセーフテーブル。新しいテーブルのためのデフォルトのストレージエンジン。MySQL は経験しているが、InnoDB がはじめてである場合は、 第15章 「InnoDB ストレージエンジン」 、そのなかでも特に セクション 15.1 「InnoDB 入門」 を参照してください。
MyISAM	主に読み取り専用または読み取りが大半のワークロードに使用される、バイナリの移植可能なストレージエンジン。 セクション 16.2 「MyISAM ストレージエンジン」 を参照してください。
MEMORY	このストレージエンジンのデータは、メモリー内のみ格納されます。 セクション 16.3 「MEMORY ストレージエンジン」 を参照してください。
CSV	カンマ区切り値形式で行を格納するテーブル。 セクション 16.4 「CSV ストレージエンジン」 を参照してください。
ARCHIVE	アーカイブストレージエンジン。 セクション 16.5 「ARCHIVE ストレージエンジン」 を参照してください。
EXAMPLE	サンプルのエンジン。 セクション 16.9 「EXAMPLE ストレージエンジン」 を参照してください。
FEDERATED	リモートテーブルにアクセスするストレージエンジン。 セクション 16.8 「FEDERATED ストレージエンジン」 を参照してください。
HEAP	これは MEMORY のシノニムです。
MERGE	1つのテーブルとして使用される MyISAM テーブルのコレクション。MRG_MyISAM と呼ばれます。 セクション 16.7 「MERGE ストレージエンジン」 を参照してください。
NDB	トランザクションと外部キーをサポートする、クラスタ化された、耐障害の、メモリーベースのテーブル。NDBCLUSTER と呼ばれます。 第23章 「MySQL NDB Cluster 8.0」 を参照してください。

デフォルトでは、使用できないストレージエンジンが指定されている場合、ステートメントはエラーで失敗します。この動作をオーバーライドするには、サーバーの SQL モードから NO_ENGINE_SUBSTITUTION を削除します ([セクション 5.1.11 「サーバー SQL モード」](#) を参照)。これにより、MySQL では、指定されたエンジンをデフォ

ルトのストレージエンジンに置き換えることができます。通常、このような場合、これは `default_storage_engine` システム変数のデフォルト値である `InnoDB` です。 `NO_ENGINE_SUBSTITUTION` が無効になっている場合、ストレージエンジンの指定が受け入れられないと警告が発生します。

- **AUTOEXTEND_SIZE**

テーブルスペースが一杯になったときに `InnoDB` がテーブルスペースのサイズを拡張する量を定義します。MySQL 8.0.23 で導入されました。設定は 4MB の倍数である必要があります。デフォルト設定は 0 で、暗黙的なデフォルト動作に従ってテーブルスペースが拡張されます。詳細は、[セクション15.6.3.9「テーブルスペースの AUTOEXTEND_SIZE 構成」](#) を参照してください。

- **AUTO_INCREMENT**

テーブルの初期の `AUTO_INCREMENT` 値。MySQL 8.0 では、これは `MyISAM`、`MEMORY`、`InnoDB`、および `ARCHIVE` テーブルに対して機能します。 `AUTO_INCREMENT` テーブルオプションをサポートしていないエンジンの最初の自動インクリメント値を設定するには、テーブルを作成したあとに目的の値より 1 小さい値を持つ「ダミーの」行を挿入してから、そのダミーの行を削除します。

`CREATE TABLE` ステートメント内の `AUTO_INCREMENT` テーブルオプションをサポートするエンジンの場合は、`ALTER TABLE tbl_name AUTO_INCREMENT = N` を使用して `AUTO_INCREMENT` 値をリセットすることもできます。この値を、現在カラム内にある最大値より小さく設定することはできません。

- **AVG_ROW_LENGTH**

テーブルの平均の行の長さの近似値。これを設定する必要があるのは、可変サイズの行を持つ大きなテーブルの場合だけです。

`MyISAM` テーブルを作成すると、MySQL は `MAX_ROWS` および `AVG_ROW_LENGTH` オプションの積を使用して、結果として得られるテーブルがどれくらいの大きさになるかを判定します。どちらのオプションも指定しない場合、`MyISAM` データおよびインデックスファイルの最大サイズは、デフォルトで 256T バイトになります。(オペレーティングシステムでその大きさのファイルがサポートされていない場合、テーブルサイズはファイルサイズ制限によって制約されます。) インデックスをより小さく、かつ高速にするためにポインタサイズを小さく維持したいと考えており、実際に大きなファイルが必要でない場合は、`myisam_data_pointer_size` システム変数を設定することによってデフォルトのポインタサイズを小さくすることができます。([セクション5.1.8「サーバシステム変数」](#) を参照してください。) すべてのテーブルをデフォルトの制限を超えて拡張できるようにしたいと考えており、テーブルが必要以上に少し遅く、かつ大きくなってもかまわない場合は、この変数を設定することによってデフォルトのポインタサイズを大きくすることができます。この値を 7 に設定すると、最大 65,536T バイトのテーブルサイズが許可されます。

- **[DEFAULT] CHARACTER SET**

テーブルのデフォルトの文字セットを指定します。 `CHARSET` は `CHARACTER SET` のシノニムです。文字セット名が `DEFAULT` である場合は、データベース文字セットが使用されます。

- **CHECKSUM**

MySQL ですべての行のライブチェックサム (つまり、テーブルが変更されると MySQL が自動的に更新するチェックサム) が保持されるようにする場合は、これを 1 に設定します。これにより、テーブルの更新が少し遅くなりますが、破損したテーブルを見つけることが容易になります。 `CHECKSUM TABLE` ステートメントは、このチェックサムをレポートします。(`MyISAM` のみ。)

- **[DEFAULT] COLLATE**

テーブルのデフォルトの照合を指定します。

- **COMMENT**

テーブルのコメントであり、長さは最大 2048 文字です。

`table_option COMMENT` 句を使用して、テーブルの `InnoDB MERGE_THRESHOLD` 値を設定できます。 [セクション15.8.11「インデックスページのマージしきい値の構成」](#) を参照してください。

「`NDB_TABLE` の設定」オプション。 `NDB` テーブルを作成する `CREATE TABLE` のまたは変更する `ALTER TABLE` ステートメントのテーブルコメントは、`NDB_TABLE` オプション `NOLOGGING`、`READ_BACKUP`、

`PARTITION_BALANCE`、または `FULLY_REPLICATED` の 1 つから 4 つを指定するのに使用でき、カンマ区切り (必要ならば) の名前 - 値ペアのセットとして、引用符付きテキストで始まる文字列 `NDB_TABLE=` の直後に続きます。この構文を使用したステートメントの例を次に示します (強調されたテキスト):

```
CREATE TABLE t1 (  
  c1 INT NOT NULL AUTO_INCREMENT PRIMARY KEY,  
  c2 VARCHAR(100),  
  c3 VARCHAR(100))  
ENGINE=NDB  
COMMENT="NDB_TABLE=READ_BACKUP=0,PARTITION_BALANCE=FOR_RP_BY_NODE";
```

引用符で囲まれた文字列内ではスペースを使用できません。文字列では大文字と小文字は区別されません。

コメントは、`SHOW CREATE TABLE` の出力の一部として表示されます。コメントのテキストは、MySQL Information Schema `TABLES` テーブルの `TABLE_COMMENT` カラムとしても使用できます。

このコメント構文は、`NDB` テーブルの `ALTER TABLE` ステートメントでもサポートされています。`ALTER TABLE` で使用されるテーブルコメントは、テーブルにある可能性のある既存のコメントを置き換えることに注意してください。

テーブルコメントでの `MERGE_THRESHOLD` オプションの設定は、`NDB` テーブルではサポートされていません (無視されます)。

完全な構文情報および例は、[セクション13.1.20.11「NDB_TABLE オプションの設定」](#) を参照してください。

- **COMPRESSION**

InnoDB テーブルのページレベル圧縮に使用される圧縮アルゴリズム。サポートされている値は、`Zlib`、`LZ4` および `None` です。`COMPRESSION` 属性は、透過的ページ圧縮機能とともに導入されました。ページ圧縮は、`file-per-table` テーブルスペースに存在する InnoDB テーブルでのみサポートされており、スパースファイルおよびホールパンチをサポートする Linux および Windows プラットフォームでのみ使用できます。詳細は、[セクション15.9.2「InnoDB ページ圧縮」](#) を参照してください。

- **CONNECTION**

`FEDERATED` テーブルの接続文字列。

注記

古いバージョンの MySQL は、接続文字列に `COMMENT` オプションを使用していました。

- **DATA DIRECTORY、INDEX DIRECTORY**

InnoDB の場合、`DATA DIRECTORY='directory'` 句を使用すると、データディレクトリ外にテーブルを作成できます。`DATA DIRECTORY` 句を使用するには、`innodb_file_per_table` 変数を有効にする必要があります。完全なディレクトリパスを指定する必要があります。MySQL 8.0.21 では、指定されたディレクトリは InnoDB で認識されている必要があります。詳細は、[セクション15.6.1.2「外部でのテーブルの作成」](#) を参照してください。

MyISAM テーブルを作成する場合は、`DATA DIRECTORY='directory'` 句、`INDEX DIRECTORY='directory'` 句、またはその両方を使用できます。これらは、それぞれ MyISAM テーブルのデータファイルとインデックスファイルを配置する場所を指定します。InnoDB テーブルとは異なり、`DATA DIRECTORY` または `INDEX DIRECTORY` オプ

ションで **MyISAM** テーブルを作成する場合、MySQL はデータベース名に対応するサブディレクトリを作成しません。各ファイルは、指定されたディレクトリ内に作成されます。

DATA DIRECTORY または **INDEX DIRECTORY** テーブルオプションを使用するには、**FILE** 権限が必要です。

重要

テーブルレベルの **DATA DIRECTORY** および **INDEX DIRECTORY** オプションは、パーティション化されたテーブルでは無視されます。(Bug #32091)

これらのオプションは、`--skip-symbolic-links` オプションを使用していない場合にのみ機能します。また、オペレーティングシステムにも、機能するスレッドに対して安全な `realpath()` 呼び出しが存在する必要があります。詳細は、[セクション8.12.2.2「Unix 上の MyISAM へのシンボリックリンクの使用」](#)を参照してください。

MyISAM テーブルが **DATA DIRECTORY** オプションなしで作成される場合、**.MYD** ファイルがデータベースディレクトリ内に作成されます。デフォルトでは、**MyISAM** が既存の **.MYD** ファイルを検出した場合、そのファイルを上書きします。**INDEX DIRECTORY** オプションを指定せずに作成されたテーブルについて、**.MYI** ファイルに同じことが当てはまります。この動作を抑制するには、`--keep_files_on_create` オプションを使用してサーバーを起動します。この場合、**MyISAM** は既存のファイルを上書きせず、かわりにエラーを返します。

DATA DIRECTORY または **INDEX DIRECTORY** オプションを使用して **MyISAM** テーブルが作成され、既存の **.MYD** または **.MYI** ファイルが見つかった場合、**MyISAM** は常にエラーを返し、指定されたディレクトリ内のファイルを上書きしません。

重要

DATA DIRECTORY または **INDEX DIRECTORY** では、MySQL データディレクトリを含むパス名を使用できません。これには、パーティション化されたテーブルや個々のテーブルパーティションが含まれます。(Bug #32167 を参照してください。)

• **DELAY_KEY_WRITE**

テーブルのキー更新をテーブルが閉じられるまで遅らせる場合は、これを 1 に設定します。[セクション5.1.8「サーバーシステム変数」](#)にある `delay_key_write` システム変数の説明を参照してください。(MyISAM のみ。)

• **ENCRYPTION**

ENCRYPTION 句は、**InnoDB** テーブルのページレベルのデータ暗号化を有効または無効にします。暗号化を有効にする前に、キーリングプラグインをインストールして構成する必要があります。MySQL 8.0.16 より前では、**ENCRYPTION** 句は `file-per-table` テーブルスペースにテーブルを作成する場合にのみ指定できます。MySQL 8.0.16 では、一般的なテーブルスペースにテーブルを作成するときに **ENCRYPTION** 句を指定することもできます。

MySQL 8.0.16 では、**ENCRYPTION** 句が指定されていない場合、テーブルはデフォルトのスキーマ暗号化を継承します。`table_encryption_privilege_check` 変数が有効になっている場合、デフォルトのスキーマ暗号化とは異なる **ENCRYPTION** 句設定を使用してテーブルを作成するには、**TABLE_ENCRYPTION_ADMIN** 権限が必要です。一般的なテーブルスペースにテーブルを作成する場合、テーブルとテーブルスペースの暗号化は一致する必要があります。

MySQL 8.0.16 の時点では、暗号化をサポートしていないストレージエンジンを使用する場合、`'N'` または `"` 以外の値で **ENCRYPTION** 句を指定することはできません。以前は、条項は受け入れられました。

詳細は、[セクション15.13「InnoDB 保存データ暗号化」](#)を参照してください。

- **ENGINE_ATTRIBUTE** および **SECONDARY_ENGINE_ATTRIBUTE** オプション (MySQL 8.0.21 の時点で使用可能) を使用して、プライマリおよびセカンダリストレージエンジンのテーブル属性を指定します。オプションは、将来の使用のために予約されています。

許可される値は、有効な **JSON** ドキュメントまたは空の文字列 (") を含む文字列リテラルです。無効な **JSON** が拒否されました。

```
CREATE TABLE t1 (c1 INT) ENGINE_ATTRIBUTE={'key':"value"};
```

ENGINE_ATTRIBUTE および **SECONDARY_ENGINE_ATTRIBUTE** の値は、エラーなしで繰り返すことができます。この場合、最後に指定した値が使用されます。

ENGINE_ATTRIBUTE および **SECONDARY_ENGINE_ATTRIBUTE** の値は、サーバーによってチェックされず、テーブルストレージエンジンが変更されたときにもクリアされません。

- **INSERT_METHOD**

MERGE テーブルにデータを挿入する場合は、**INSERT_METHOD** を使用して、行を挿入するテーブルを指定する必要があります。**INSERT_METHOD** は、**MERGE** テーブルにのみ役立つオプションです。最初または最後のテーブルに挿入するには **FIRST** または **LAST** の値を、挿入されないようにするには **NO** の値を使用します。[セクション 16.7 「MERGE ストレージエンジン」](#) を参照してください。

- **KEY_BLOCK_SIZE**

MyISAM テーブルの場合、**KEY_BLOCK_SIZE** はオプションで、インデックススキップブロックに使用するサイズをバイト単位で指定します。この値はヒントとして扱われます。必要に応じて、異なるサイズが使用される可能性があります。個々のインデックス定義に指定された **KEY_BLOCK_SIZE** 値は、テーブルレベルの **KEY_BLOCK_SIZE** 値をオーバーライドします。

InnoDB テーブルの場合、**KEY_BLOCK_SIZE** は **compressed InnoDB** テーブルに使用する **page** サイズを KB 単位で指定します。**KEY_BLOCK_SIZE** 値は、ヒントとして処理されます。**InnoDB** では、必要に応じて異なるサイズが使用される可能性があります。**KEY_BLOCK_SIZE** は、**innodb_page_size** 値以下にのみできます。値 0 は、**innodb_page_size** 値の半分であるデフォルトの圧縮済みページサイズを表します。**innodb_page_size** に応じて、可能な **KEY_BLOCK_SIZE** 値には 0、1、2、4、8 および 16 が含まれます。詳しくは [セクション 15.9.1 「InnoDB テーブルの圧縮」](#) をご覧ください。

InnoDB テーブルに **KEY_BLOCK_SIZE** を指定する場合、Oracle では **innodb_strict_mode** を有効にすることをお勧めします。**innodb_strict_mode** が有効な場合、無効な **KEY_BLOCK_SIZE** 値を指定するとエラーが返されます。**innodb_strict_mode** が無効な場合、無効な **KEY_BLOCK_SIZE** 値は警告になり、**KEY_BLOCK_SIZE** オプションは無視されます。

SHOW TABLE STATUS に対するレスポンスの **Create_options** カラムには、**SHOW CREATE TABLE** と同様に、テーブルで使用される実際の **KEY_BLOCK_SIZE** がレポートされます。

InnoDB では、テーブルレベルでの **KEY_BLOCK_SIZE** のみがサポートされます。

KEY_BLOCK_SIZE は、32KB および 64KB の **innodb_page_size** 値ではサポートされていません。**InnoDB** テーブル圧縮では、これらのページサイズはサポートされていません。

InnoDB では、一時テーブルの作成時に **KEY_BLOCK_SIZE** オプションをサポートしていません。

- **MAX_ROWS**

テーブル内に格納することを予定している行の最大数。これは強い制限値ではなく、どちらかと言うと、テーブルが少なくともこの行数を格納できる必要があるという、ストレージエンジンへのヒントです。

重要

NDB テーブルで **MAX_ROWS** を使用してテーブルパーティションの数を制御することは非推奨です。下位互換性のために新しいバージョンでも引き続きサポートされますが、将

来のリリースでは削除される予定です。かわりに `PARTITION_BALANCE` を使用してください。「[NDB_TABLE の設定](#)」オプションを参照してください。

NDB ストレージエンジンは、この値を最大値として扱います。非常に大きな「NDB Cluster」テーブル (数百万行を含む) を作成する場合は、このオプションを使用して、`MAX_ROWS = 2 * rows` を設定することで、NDB がテーブルの主キーのハッシュの格納に使用するハッシュテーブルに十分な数の索引スロットを割り当てるようにする必要があります (`rows` は、テーブルに挿入する予定の行数です)。

`MAX_ROWS` の最大値は 4294967295 です。これを超える値は、この制限に切り捨てられます。

- `MIN_ROWS`

テーブル内に格納することを予定している行の最小数。MEMORY ストレージエンジンは、このオプションをメモリー使用に関するヒントとして使用します。

- `PACK_KEYS`

MyISAM テーブルでのみ有効です。インデックスを小さくする場合は、このオプションを 1 に設定します。通常は、これによって更新は遅く、読み取りは高速になります。このオプションを 0 に設定すると、キーのすべてのパッキングが無効になります。これを `DEFAULT` に設定すると、長い `CHAR`、`VARCHAR`、`BINARY`、または `VARBINARY` カラムのみをパックするようストレージエンジンに指示します。

`PACK_KEYS` を使用しない場合、デフォルトでは文字列をパックしますが、数値はパックしません。`PACK_KEYS=1` を使用した場合は、数値もパックされます。

2 進数のキーをパックする場合、MySQL は次のプリフィクス圧縮を使用します。

- 前のキーの何バイトが次のキーと同じであることを示すために、すべてのキーに 1 バイトが余分に必要になります。
- 行へのポインタは、圧縮率を向上させるために、キーの直後に高位バイトが先に来る順序で格納されます。

つまり、2 つの連続した行に等しいキーが多数存在する場合は、次の「同じ」キーはすべて、通常 (行へのポインタを含め) 2 バイトしか占有しません。これを、次のキーが `storage_size_for_key + pointer_size` (ここで、ポインタサイズは通常 4) を占有する通常のケースと比較してください。逆に言うと、プリフィクス圧縮から大きな利点を得られるのは、同じ数値が多数存在する場合だけです。すべてのキーが完全に異なっている場合は、そのキーが `NULL` 値を持つことができるキーでないかぎり、キーあたり 1 バイト多く使用されます。(この場合、パックされたキーの長さは、キーが `NULL` であるかどうかをマークするために使用されるのと同じバイトに格納されます。)

- `PASSWORD`

このオプションは使用されません。

- `ROW_FORMAT`

行が格納される物理フォーマットを定義します。

`strict mode` を無効にしてテーブルを作成する場合、指定した行フォーマットがサポートされていないと、ストレージエンジンのデフォルトの行フォーマットが使用されます。テーブルの実際の行形式は、`SHOW TABLE STATUS`

に応じて `Row_format` カラムにレポートされます。 `Create_options` カラムには、`SHOW CREATE TABLE` と同様に、`CREATE TABLE` ステートメントで指定された行形式が表示されます。

行形式の選択は、テーブルに使用されるストレージエンジンによって異なります。

InnoDB テーブルの場合:

- デフォルトの行フォーマットは、`DYNAMIC` のデフォルト設定を持つ `innodb_default_row_format` によって定義されます。デフォルトの行フォーマットは、`ROW_FORMAT` オプションが定義されていない場合、または `ROW_FORMAT=DEFAULT` が使用されている場合に使用されます。

`ROW_FORMAT` オプションが定義されていない場合、または `ROW_FORMAT=DEFAULT` が使用されている場合は、テーブルを再構築する操作によって、テーブルの行形式も `innodb_default_row_format` で定義されているデフォルトに暗黙的に変更されます。詳細は、[テーブルの行形式の定義](#)を参照してください。

- データ型 (特に `BLOB` 型) の InnoDB 記憶域をより効率的にするには、`DYNAMIC` を使用します。`DYNAMIC` の行形式に関連する要件については、[DYNAMIC 行フォーマット](#) を参照してください。
- InnoDB テーブルの圧縮を有効にするには、`ROW_FORMAT=COMPRESSED` を指定します。一時テーブルを作成する場合、`ROW_FORMAT=COMPRESSED` オプションはサポートされません。`COMPRESSED` の行形式に関連する要件については、[セクション15.9「InnoDB のテーブルおよびページの圧縮」](#)を参照してください。
- 以前のバージョンの MySQL で使用されていた行形式は、`REDUNDANT` の行形式を指定することで引き続きリクエストできます。
- デフォルト以外の `ROW_FORMAT` 句を指定する場合は、`innodb_strict_mode` 構成オプションも有効にすることを考慮してください。
- `ROW_FORMAT=FIXED` はサポートされていません。`innodb_strict_mode` が無効になっているときに `ROW_FORMAT=FIXED` が指定された場合、InnoDB は警告を発行し、`ROW_FORMAT=DYNAMIC` とみなします。`innodb_strict_mode` が有効になっている間に `ROW_FORMAT=FIXED` が指定されている場合 (デフォルト)、InnoDB はエラーを返します。
- InnoDB 行フォーマットの詳細は、[セクション15.10「InnoDB の行フォーマット」](#)を参照してください。

MyISAM テーブルの場合は、このオプション値を、静的行フォーマットまたは可変長行フォーマットを示す `FIXED` または `DYNAMIC` に設定できます。`mysampack` は、この型を `COMPRESSED` に設定します。[セクション16.2.3「MyISAM テーブルのストレージフォーマット」](#)を参照してください。

NDB テーブルの場合、デフォルトの `ROW_FORMAT` は `DYNAMIC` です。

• `STATS_AUTO_RECALC`

InnoDB テーブルの永続的統計を自動的に再計算するかどうかを指定します。値 `DEFAULT` を指定すると、テーブルの永続的統計設定は `innodb_stats_auto_recalc` 構成オプションによって決定されます。値 `1` を指定すると、統計は、テーブル内のデータの 10% が変更されたときに再計算されます。値 `0` は、このテーブルの自動再計算が行われないようにします。この設定の場合、テーブルへの大幅な変更を行なったあとに統計を再計算するには、`ANALYZE TABLE` ステートメントを発行します。永続的統計機能の詳細は、[セクション15.8.10.1「永続的オプティマイザ統計のパラメータの構成」](#)を参照してください。

• `STATS_PERSISTENT`

InnoDB テーブルの永続的統計を有効にするかどうかを指定します。値 `DEFAULT` を指定すると、テーブルの永続的統計設定は `innodb_stats_persistent` 構成オプションによって決定されます。値 `1` がテーブルの永続的統計を有効にするのに対して、値 `0` はこの機能を無効にします。`CREATE TABLE` または `ALTER TABLE` ステートメントを使用して永続的統計を有効にしたあと、代表的なデータのテーブルへのロード後に統計を計算するには、`ANALYZE TABLE` ステートメントを発行します。永続的統計機能の詳細は、[セクション15.8.10.1「永続的オプティマイザ統計のパラメータの構成」](#)を参照してください。

- **STATS_SAMPLE_PAGES**

インデックス付きカラムのカーディナリティーやその他の統計 (**ANALYZE TABLE** によって計算される統計など) を推定するときにサンプリングするインデックスページの数。詳細は、[セクション15.8.10.1「永続的オブティマイザ統計のパラメータの構成」](#)を参照してください。

- **TABLESPACE**

TABLESPACE 句を使用すると、既存の一般テーブルスペース、file-per-table テーブルスペースまたはシステムテーブルスペースにテーブルを作成できます。

```
CREATE TABLE tbl_name ... TABLESPACE [=] tablespace_name
```

TABLESPACE 句を使用する前に、指定する一般テーブルスペースが存在している必要があります。一般テーブルスペースの詳細は、[セクション15.6.3.3「一般テーブルスペース」](#)を参照してください。

tablespace_name は、大/小文字を区別する識別子です。引用符で囲むことも、引用符で囲まないこともできます。スラッシュ文字 (「/」) は使用できません。「innodb_」で始まる名前は、特別な用途のために予約されています。

システムテーブルスペースにテーブルを作成するには、テーブルスペース名として **innodb_system** を指定します。

```
CREATE TABLE tbl_name ... TABLESPACE [=] innodb_system
```

TABLESPACE [=] innodb_system を使用すると、**innodb_file_per_table** の設定に関係なく、圧縮されていない行形式のテーブルをシステムテーブルスペースに配置できます。たとえば、**TABLESPACE [=] innodb_system** を使用して、**ROW_FORMAT=DYNAMIC** を含むテーブルをシステムテーブルスペースに追加できます。

file-per-table テーブルスペースにテーブルを作成するには、テーブルスペース名として **innodb_file_per_table** を指定します。

```
CREATE TABLE tbl_name ... TABLESPACE [=] innodb_file_per_table
```

注記

innodb_file_per_table が有効な場合、InnoDB file-per-table テーブルスペースを作成するために **TABLESPACE=innodb_file_per_table** を指定する必要はありません。InnoDB テーブルは、**innodb_file_per_table** が有効な場合、デフォルトで file-per-table テーブルスペースに作成されます。

DATA DIRECTORY 句は **CREATE TABLE ... TABLESPACE=innodb_file_per_table** では許可されますが、それ以外の場合は **TABLESPACE** 句との組合せでの使用はサポートされていません。MySQL 8.0.21 では、**DATA DIRECTORY** 句で指定されたディレクトリは InnoDB で認識されている必要があります。詳細は、[DATA DIRECTORY 句の使用](#)を参照してください。

注記

CREATE TEMPORARY TABLE での **TABLESPACE = innodb_file_per_table** 句および **TABLESPACE = innodb_temporary** 句のサポートは、MySQL 8.0.13 で非推奨になりました。MySQL の将来のバージョンで削除される予定です。

STORAGE テーブルオプションは、NDB テーブルでのみ使用されます。**STORAGE** は、使用される記憶域のタイプ (ディスクまたはメモリー) を決定し、**DISK** または **MEMORY** のいずれかになります。

TABLESPACE ... STORAGE DISK は、「NDB Cluster ディスクデータ」テーブルスペースにテーブルを割り当てます。テーブルスペースは、**CREATE TABLESPACE** を使用してすでに作成されている必要があります。詳細は、[セクション23.5.10「NDB Cluster ディスクデータテーブル」](#)を参照してください。

重要

STORAGE 句を、**TABLESPACE** 句のない **CREATE TABLE** ステートメントで使用することはできません。

- **UNION**

同一の **MyISAM** テーブルのコレクションにアクセスするために使用します。これは、**MERGE** テーブルでのみ機能します。 [セクション16.7「MERGE ストレージエンジン」](#) を参照してください。

MERGE テーブルにマップするテーブルに対する **SELECT**、**UPDATE**、および **DELETE** 権限が必要です。

注記

以前は、使用されるすべてのテーブルが **MERGE** テーブル自体と同じデータベース内に存在する必要がありました。この制限は適用されなくなりました。

テーブルのパーティション化

partition_options を使用すると、**CREATE TABLE** で作成されたテーブルのパーティション化を制御できます。

このセクションの最初にある **partition_options** の構文に示されているすべてのオプションが、すべてのパーティショニングタイプに使用できるわけではありません。各タイプに固有の情報については、次の個々のタイプのリストを参照してください。また、MySQL でのパーティション化の動作や使用に関するより詳細な情報、および MySQL のパーティション化に関連したテーブル作成やその他のステートメントの追加の例については、[第24章「パーティション化」](#) を参照してください。

パーティションに対しては変更、マージ、テーブルへの追加、およびテーブルからの削除が可能です。これらのタスクを実行するための MySQL ステートメントに関する基本情報については、[セクション13.1.9「ALTER TABLE ステートメント」](#) を参照してください。より詳細な説明および例については、[セクション24.3「パーティション管理」](#) を参照してください。

• PARTITION BY

partition_options 句が使用される場合、この句は **PARTITION BY** で始まります。この句には、パーティションを決定するために使用される関数が含まれています。この関数は、1 から **num** までの範囲の整数値を返します。ここで、**num** はパーティションの数です。(テーブルに含めることのできるユーザー定義パーティションの最大数は 1024 です。この最大数には、このセクションのあとの方で説明されているサブパーティションの数が含まれていません。)

注記

PARTITION BY 句で使用される式 (**expr**) は、作成されているテーブルにはないどのカラムも参照できません。このような参照は明確に禁止されており、そのステートメントがエラーで失敗する原因になります。(Bug #29444)

• HASH(expr)

1 つ以上のカラムをハッシュして、行を配置および検索するためのキーを作成します。**expr** は、1 つ以上のテーブルのカラムを使用する式です。これは、1 つの整数値が得られる任意の有効な MySQL 式 (MySQL 関数を含む) にすることができます。たとえば、次はどちらも、**PARTITION BY HASH** を使用した有効な **CREATE TABLE** ステートメントです。

```
CREATE TABLE t1 (col1 INT, col2 CHAR(5))
PARTITION BY HASH(col1);

CREATE TABLE t1 (col1 INT, col2 CHAR(5), col3 DATETIME)
PARTITION BY HASH ( YEAR(col3) );
```

PARTITION BY HASH では、**VALUES LESS THAN** または **VALUES IN** のどちらの句も使用できません。

PARTITION BY HASH は、**expr** をパーティションの数で割った余り (つまり、法) を使用します。例および追加情報については、[セクション24.2.4「HASH パーティショニング」](#) を参照してください。

LINEAR キーワードには、いくぶん異なるアルゴリズムが必要になります。この場合、行が格納されるパーティションの数は、1 つ以上の論理的な **AND** 演算の結果として計算されます。線形ハッシュの説明および例については、[セクション24.2.4.1「LINEAR HASH パーティショニング」](#) を参照してください。

• KEY(column_list)

これは [HASH](#) と似ていますが、偶数のデータ分散を保証するために、MySQL がハッシュ関数を提供する点が異なります。 `column_list` 引数は、単純に 1 つ以上のテーブルカラム (最大 16 個) のリストです。この例は、4 つのパーティションを持つ、キーによってパーティション化された単純なテーブルを示しています。

```
CREATE TABLE tk (col1 INT, col2 CHAR(5), col3 DATE)
PARTITION BY KEY(col3)
PARTITIONS 4;
```

キーによってパーティション化されたテーブルの場合は、[LINEAR](#) キーワードを使用して線形パーティション化を採用できます。これには、[HASH](#) によってパーティション化されたテーブルの場合と同じ効果があります。つまり、パーティション番号は法ではなく、`&` 演算子を使用して見つけられます (詳細は、[セクション 24.2.4.1 「LINEAR HASH パーティショニング」](#) および [セクション 24.2.5 「KEY パーティショニング」](#) を参照してください)。この例では、キーによる線形パーティション化を使用して 5 つのパーティション間でデータを分散させます。

```
CREATE TABLE tk (col1 INT, col2 CHAR(5), col3 DATE)
PARTITION BY LINEAR KEY(col3)
PARTITIONS 5;
```

`ALGORITHM={1 | 2}` オプションは、`[SUB]PARTITION BY [LINEAR] KEY` でサポートされます。`ALGORITHM=1` を指定すると、サーバーは MySQL 5.1 と同じキーハッシュ関数を使用します。`ALGORITHM=2` は、サーバーが、MySQL 5.5 以降で実装され、`KEY` によってパーティション化された新しいテーブルに対してデフォルトで使用されるキーハッシュ関数を採用することを示します。(MySQL 5.5 以降で採用されたキーハッシュ関数によって作成されたパーティション化されたテーブルを MySQL 5.1 サーバーで使用することはできません。) このオプションを指定しない場合は、`ALGORITHM=2` を使用するのと同じ効果があります。このオプションは、主に `[LINEAR] KEY` によってパーティション化されたテーブルを MySQL 5.1 以降の MySQL バージョン間でアップグレードまたはダウングレードするときに使用するか、または MySQL 5.5 以降のサーバー上で、MySQL 5.1 サーバー上で使用できる `KEY` または `LINEAR KEY` によってパーティション化されたテーブルを作成することを目的にしています。詳細は、[セクション 13.1.9.1 「ALTER TABLE パーティション操作」](#) を参照してください。

MySQL 5.7 以降の `mysqldump` では、次のようにバージョンングされたコメントにこのオプションが含まれていません:

```
CREATE TABLE t1 (a INT)
/*!50100 PARTITION BY KEY */ /*!50611 ALGORITHM = 1 */ /*!50100 ()
PARTITIONS 3 */
```

これにより、MySQL 5.6.10 以前のサーバーはこのオプションを無視するようになります。これらのバージョンでは、通常であれば構文エラーが発生します。`KEY` によってパーティション化またはサブパーティション化されたテーブルをバージョン 5.6.11 より前の MySQL 5.6 サーバーに使用する MySQL 5.7 サーバーで作成されたダンプをロードする場合は、先に進む前に [Changes in MySQL 5.6](#) に問い合わせてください。(見つかった情報は、MySQL 5.7(事実上 5.6.11 または later-server) から作成された `KEY` パーティションテーブルまたはサブパーティションテーブルを含むダンプを MySQL 5.5.30 以前のサーバーにロードする場合にも適用されます。)

また、MySQL 5.6.11 以降では、`ALGORITHM=1` が `mysqldump` と同じ方法で、バージョン管理されたコメントを使用して `SHOW CREATE TABLE` の出力に必要なに応じて表示されます。`ALGORITHM=2` は、元のテーブルを作成するときにこのオプションが指定された場合でも、`SHOW CREATE TABLE` の出力から常に省略されます。

`PARTITION BY KEY` では、`VALUES LESS THAN` または `VALUES IN` のどちらの句も使用できません。

- `RANGE(expr)`

この場合、`expr` では、一連の `VALUES LESS THAN` 演算子を使用して値の範囲が表示されます。範囲のパーティション化を使用する場合は、`VALUES LESS THAN` を使用して、少なくとも 1 つのパーティションを定義する必要があります。範囲のパーティション化では `VALUES IN` を使用できません。

注記

`RANGE` によってパーティション化されたテーブルでは、`VALUES LESS THAN` を整数リテラル値、または 1 つの整数値に評価される式のどちらかとともに使用する必要があります。

ます。MySQL 8.0 では、このセクションのあとの方で説明されている、[PARTITION BY RANGE COLUMNS](#) を使用して定義されたテーブルでこの制限を克服できます。

次のスキームに従って、年の値を含むカラムに関してパーティション化するテーブルがあるとします。

パーティション番号:	年の範囲:
0	1990 以前
1	1991 から 1994 まで
2	1995 から 1998 まで
3	1999 から 2002 まで
4	2003 から 2005 まで
5	2006 以降

このようなパーティション化スキームを実装するテーブルは、次に示す [CREATE TABLE](#) ステートメントによって実現できます。

```
CREATE TABLE t1 (
  year_col INT,
  some_data INT
)
PARTITION BY RANGE (year_col) (
  PARTITION p0 VALUES LESS THAN (1991),
  PARTITION p1 VALUES LESS THAN (1995),
  PARTITION p2 VALUES LESS THAN (1999),
  PARTITION p3 VALUES LESS THAN (2002),
  PARTITION p4 VALUES LESS THAN (2006),
  PARTITION p5 VALUES LESS THAN MAXVALUE
);
```

[PARTITION ... VALUES LESS THAN ...](#) ステートメントは、連続的に機能します。 [VALUES LESS THAN MAXVALUE](#) は、それ以外で指定されている最大値より大きい「残りの」値を指定するように機能します。

[VALUES LESS THAN](#) 句は、[switch ... case](#) ブロックの [case](#) 部分と同様の方法で順次機能します (C、Java、PHP などの多くのプログラミング言語で使用されています)。つまり、この句は、連続した各 [VALUES LESS THAN](#) で指定されている上限が前の句の上限より大きく、かつ [MAXVALUE](#) を参照している句がリスト内のすべての句の最後に来るような方法で配置されている必要があります。

- [RANGE COLUMNS\(column_list\)](#)

[RANGE](#) のこのバリエーションにより、複数のカラムで範囲条件を使用する (つまり、[WHERE a = 1 AND b < 10](#) や [WHERE a = 1 AND b = 10 AND c < 10](#) などの条件を持つ) クエリーのパーティションブルーニングが容易になります。これにより、[COLUMNS](#) 句内のカラムのリストと、各 [PARTITION ... VALUES LESS THAN \(value_list\)](#) パーティション定義句内のカラム値のセットを使用して、複数のカラム内の値の範囲を指定できるようになります。(もっとも単純なケースでは、このセットは単一カラムで構成されます。) [column_list](#) および [value_list](#) で参照できるカラムの最大数は 16 です。

[COLUMNS](#) 句で使用される [column_list](#) には、カラムの名前のみを含めることができます。リスト内の各カラムは MySQL のデータ型のうち、整数型、文字列型、および時間または日付カラム型のいずれかである必要があります。[BLOB](#)、[TEXT](#)、[SET](#)、[ENUM](#)、[BIT](#)、または空間データ型を使用したカラムは許可されていません。浮動小数点数型を使用するカラムも許可されていません。また、[COLUMNS](#) 句では、関数や演算式も使用できません。

パーティション定義で使用される [VALUES LESS THAN](#) 句は、[COLUMNS\(\)](#) 句に現れるカラムごとにリテラル値を指定する必要があります。つまり、各 [VALUES LESS THAN](#) 句で使用される値のリストには、[COLUMNS](#) 句にリストされているカラムの数と同じ数の値が含まれている必要があります。 [VALUES LESS THAN](#) 句で [COLUMNS](#) 句に存在する数より多いか、または少ない値を使用しようとすると、このステートメントは次のエラーで失敗します。 [Inconsistency in usage of column lists for partitioning...](#) [VALUES LESS THAN](#) に現れるどの値にも [NULL](#) は使用できません。この例に示すように、最初のカラム以外の特定のカラムで [MAXVALUE](#) を複数回使用できます。

```
CREATE TABLE rc (
  a INT NOT NULL,
  b INT NOT NULL
)
```

```
PARTITION BY RANGE COLUMNS(a,b) (  
  PARTITION p0 VALUES LESS THAN (10,5),  
  PARTITION p1 VALUES LESS THAN (20,10),  
  PARTITION p2 VALUES LESS THAN (50,MAXVALUE),  
  PARTITION p3 VALUES LESS THAN (65,MAXVALUE),  
  PARTITION p4 VALUES LESS THAN (MAXVALUE,MAXVALUE)  
);
```

VALUES LESS THAN 値リストで使用されている各値が、対応するカラムの型に正確に一致している必要があります。変換は行われません。たとえば、整数型を使用するカラムに一致する値として文字列 '1' を使用したり (代わりに、数値 1 を使用する必要があります)、文字列型を使用するカラムに一致する値として数値 1 を使用したりすることはできません (このような場合は、引用符で囲まれた文字列 '1' を使用する必要があります)。

詳細は、[セクション24.2.1「RANGE パーティショニング」](#) および [セクション24.4「パーティションプレーニング」](#) を参照してください。

- **LIST(expr)**

これは、州や国コードなど、使用可能な値の制限されたセットを持つテーブルのカラムに基づいてパーティションを割り当てる場合に役立ちます。このような場合は、特定の州または国に関連するすべての行を単一パーティションに割り当てたり、特定の州または国のセットのためにパーティションを予約したりできます。これは **RANGE** に似ていますが、各パーティションに許可される値を指定するために **VALUES IN** しか使用できない点が異なります。

VALUES IN は、一致させる値のリストとともに使用されます。たとえば、次のようなパーティション化スキームを作成できます。

```
CREATE TABLE client_firms (  
  id INT,  
  name VARCHAR(35)  
)  
PARTITION BY LIST (id) (  
  PARTITION r0 VALUES IN (1, 5, 9, 13, 17, 21),  
  PARTITION r1 VALUES IN (2, 6, 10, 14, 18, 22),  
  PARTITION r2 VALUES IN (3, 7, 11, 15, 19, 23),  
  PARTITION r3 VALUES IN (4, 8, 12, 16, 20, 24)  
);
```

リストのパーティション化を使用する場合は、**VALUES IN** を使用して、少なくとも 1 つのパーティションを定義する必要があります。 **PARTITION BY LIST** では **VALUES LESS THAN** を使用できません。

注記

LIST によってパーティション化されたテーブルでは、**VALUES IN** で使用される値リストを整数値のみで構成する必要があります。MySQL 8.0 では、このセクションのあとの方で説明されている、**LIST COLUMNS** によるパーティション化を使用してこの制限を克服できます。

- **LIST COLUMNS(column_list)**

LIST のこのバリエーションにより、複数のカラムで比較条件を使用する (つまり、**WHERE a = 5 AND b = 5** や **WHERE a = 1 AND b = 10 AND c = 5** などの条件を持つ) クエリーのパーティションプレーニングが容易になります。これにより、**COLUMNS** 句内のカラムのリストと、各 **PARTITION ... VALUES IN (value_list)** パーティション定義句内のカラム値のセットを使用して、複数のカラム内の値を指定できるようになります。

LIST COLUMNS(column_list) で使用されるカラムリストと **VALUES IN(value_list)** で使用される値リストに関連したデータ型を管理するルールは、**VALUES IN** 句では **MAXVALUE** が許可されず、**NULL** を使用できる点を除き、それぞれ **RANGE COLUMNS(column_list)** で使用されるカラムリストと **VALUES LESS THAN(value_list)** で使用される値リストの場合のルールと同じです。

PARTITION BY LIST COLUMNS で **VALUES IN** に使用される値のリストには、**PARTITION BY LIST** で使用された場合と比較して重要な違いが 1 つあります。**PARTITION BY LIST COLUMNS** で使用された場合、**VALUES IN** 句内の各要素は、カラム値のセットである必要があります。各セット内の値の数は **COLUMNS** 句で使用されているカラム数と同じである必要があります、これらの値のデータ型はそれらのカラムのデータ型に一致している (しか

も、同じ順序で現れる) 必要があります。もっとも単純なケースでは、このセットは単一カラムで構成されます。`column_list` および `value_list` を構成する各要素で利用できるカラムの最大数は 16 です。

次の `CREATE TABLE` ステートメントで定義されるテーブルは、`LIST COLUMNS` パーティション化を使用したテーブルの例を示しています。

```
CREATE TABLE lc (  
  a INT NULL,  
  b INT NULL  
)  
PARTITION BY LIST COLUMNS(a,b) (  
  PARTITION p0 VALUES IN( (0,0), (NULL,NULL) ),  
  PARTITION p1 VALUES IN( (0,1), (0,2), (0,3), (1,1), (1,2) ),  
  PARTITION p2 VALUES IN( (1,0), (2,0), (2,1), (3,0), (3,1) ),  
  PARTITION p3 VALUES IN( (1,3), (2,2), (2,3), (3,2), (3,3) )  
);
```

- `PARTITIONS num`

オプションで、`PARTITIONS num` 句を使用してパーティションの数を指定できます。ここで、`num` はパーティションの数です。この句とほかのいずれかの `PARTITION` 句の両方が使用されている場合、`num` は、`PARTITION` 句を使用して宣言されているすべてのパーティションの総数と同じである必要があります。

注記

`RANGE` または `LIST` によってパーティション化されたテーブルの作成で `PARTITIONS` 句を使用するかどうかにかかわらず、テーブル定義には引き続き、少なくとも 1 つの `PARTITION VALUES` 句を含める必要があります (下を参照してください)。

- `SUBPARTITION BY`

オプションで、パーティションを複数のサブパーティションに分割できます。これは、オプションの `SUBPARTITION BY` 句を使用して示すことができます。サブパーティション化は、`HASH` または `KEY` によって実行できます。これらのどちらも `LINEAR` にすることができます。これらは、同等のパーティショニングタイプについて前に説明したのと同じように機能します。(`LIST` または `RANGE` によってサブパーティション化することはできません。)

サブパーティションの数は、`SUBPARTITIONS` キーワードと、そのあとの整数値を使用して示すことができます。

- `PARTITIONS` または `SUBPARTITIONS` 句で使用されている値の厳密なチェックが適用され、この値は次のルールに従っている必要があります。
 - この値は 0 以外の正の整数である必要があります。
 - 先頭の 0 は許可されません。
 - この値は整数リテラルである必要があり、式にすることはできません。たとえば、`0.2E+01` が 2 に評価されたとしても、`PARTITIONS 0.2E+01` は許可されません。(Bug #15890)
- `partition_definition`

各パーティションは、`partition_definition` 句を使用して個別に定義できます。この句を構成する個別の部分は次のとおりです。

- `PARTITION partition_name`

パーティションの論理名を指定します。

- `VALUES`

レンジパーティション化では、各パーティションに `VALUES LESS THAN` 句が含まれている必要があります。リストパーティション化では、パーティションごとに `VALUES IN` 句を指定する必要があります。これは、この

パーティションにどの行を格納するかを決定するために使用されます。構文の例については、[第24章「パーティション化」](#)にあるパーティショニングタイプの説明を参照してください。

- [\[STORAGE\] ENGINE](#)

MySQL は、[PARTITION](#) と [SUBPARTITION](#) の両方に対して [\[STORAGE\] ENGINE](#) オプションを受け入れます。現在、このオプションを使用できる唯一の方法は、すべてのパーティションまたはすべてのサブパーティションを同じストレージエンジンに設定し、同じテーブル内のパーティションまたはサブパーティションに異なるストレージエンジンを設定しようとする、[「ERROR 1469 \(HY000\): パーティション内のハンドラの混在は、このバージョンの MySQL では許可されていません」](#) エラーが発生することです。

- [COMMENT](#)

オプションの [COMMENT](#) 句を使用すると、このパーティションを説明する文字列を指定できます。例:

```
COMMENT = 'Data for the years previous to 1999'
```

パーティションコメントの最大長は 1024 文字です。

- [DATA DIRECTORY](#) および [INDEX DIRECTORY](#)

[DATA DIRECTORY](#) と [INDEX DIRECTORY](#) は、それぞれ、このパーティションのデータとインデックスが格納されるディレクトリを示すために使用できます。 [data_dir](#) と [index_dir](#) はどちらも、絶対システムパス名である必要があります。

MySQL 8.0.21 では、[DATA DIRECTORY](#) 句で指定されたディレクトリは [InnoDB](#) で認識されている必要があります。詳細は、[DATA DIRECTORY 句の使用](#)を参照してください。

[DATA DIRECTORY](#) または [INDEX DIRECTORY](#) パーティションオプションを使用するには、[FILE](#) 権限が必要です。

例:

```
CREATE TABLE th (id INT, name VARCHAR(30), adate DATE)
PARTITION BY LIST(YEAR(adata))
(
  PARTITION p1999 VALUES IN (1995, 1999, 2003)
  DATA DIRECTORY = '/var/appdata/95/data'
  INDEX DIRECTORY = '/var/appdata/95/idx',
  PARTITION p2000 VALUES IN (1996, 2000, 2004)
  DATA DIRECTORY = '/var/appdata/96/data'
  INDEX DIRECTORY = '/var/appdata/96/idx',
  PARTITION p2001 VALUES IN (1997, 2001, 2005)
  DATA DIRECTORY = '/var/appdata/97/data'
  INDEX DIRECTORY = '/var/appdata/97/idx',
  PARTITION p2002 VALUES IN (1998, 2002, 2006)
  DATA DIRECTORY = '/var/appdata/98/data'
  INDEX DIRECTORY = '/var/appdata/98/idx'
```


);

DATA DIRECTORY および **INDEX DIRECTORY** は、**MyISAM** テーブルで使用される **CREATE TABLE** ステートメントの **table_option** 句と同じように動作します。

パーティションごとに 1 つのデータディレクトリと 1 つのインデックスディレクトリを指定できます。指定されないままになっている場合、データとインデックスは、デフォルトではそのテーブルのデータベースディレクトリ内に格納されます。

DATA DIRECTORY および **INDEX DIRECTORY** オプションは、**NO_DIR_IN_CREATE** が有効になっている場合、パーティション化されたテーブルの作成では無視されます。

- **MAX_ROWS** および **MIN_ROWS**

パーティションに格納する行の最大数と最小数をそれぞれ指定するために使用できます。 **max_number_of_rows** と **min_number_of_rows** の値は正の整数である必要があります。同じ名前を持つテーブルレベルのオプションと同様に、これらはサーバーへの「提案」としてのみ機能し、強い制限値ではありません。

- **TABLESPACE**

TABLESPACE `innodb_file_per_table` を指定して、パーティションの **InnoDB** file-per-table テーブルスペースを指定するために使用できます。すべてのパーティションは同じストレージエンジンに属している必要があります。

InnoDB テーブルパーティションの共有 **InnoDB** テーブルスペースへの配置はサポートされていません。共有テーブルスペースには、**InnoDB** システムテーブルスペースおよび一般テーブルスペースが含まれます。

- **subpartition_definition**

オプションで、パーティション定義に 1 つ以上の **subpartition_definition** 句を含めることができます。これらの各句は、少なくとも **SUBPARTITION name** で構成されます。ここで、**name** はそのサブパーティションの識別子です。**PARTITION** キーワードが **SUBPARTITION** に置き換えられる点を除き、サブパーティション定義の構文はパーティション定義の構文と同じです。

サブパーティション化は **HASH** または **KEY** によって実行する必要があり、**RANGE** または **LIST** パーティションに対してのみ実行できます。[セクション24.2.6「サブパーティショニング」](#) を参照してください。

生成されたカラムによるパーティション化

生成されたカラムによるパーティション化が許可されます。例:

```
CREATE TABLE t1 (
  s1 INT,
  s2 INT AS (EXP(s1)) STORED
)
PARTITION BY LIST (s2) (
  PARTITION p1 VALUES IN (1)
);
```

パーティション化では、生成されたカラムは通常のカラムとして認識されます。これにより、パーティション化が許可されていない関数の制限の回避策が有効になります ([セクション24.6.3「関数に関連するパーティショニング制限」](#) を参照)。前述の例は、この方法を示しています: **EXP()** は **PARTITION BY** 句で直接使用できませんが、**EXP()** を使用して定義された生成されたカラムは許可されます。

13.1.20.1 CREATE TABLE によって作成されるファイル

file-per-table テーブルスペースまたは一般テーブルスペースに作成された **InnoDB** テーブルの場合、テーブルデータおよび関連するインデックスは、データベースディレクトリの **.ibd file** に格納されます。**InnoDB** テーブルがシステムテーブルスペースに作成されると、システムテーブルスペースを表す **ibdata* files** にテーブルデータおよびインデックスが格納されます。**innodb_file_per_table** オプションは、デフォルトで file-per-table テーブルスペースと system テーブルスペースのどちらにテーブルを作成するかを制御します。**TABLESPACE** オプションを使用すると、**innodb_file_per_table** の設定に関係なく、file-per-table テーブルスペース、一般テーブルスペースまたはシステムテーブルスペースにテーブルを配置できます。

MyISAM テーブルの場合は、ストレージエンジンがデータおよびインデックスファイルを作成します。したがって、**MyISAM** テーブル **tbl_name** ごとに 2 つのディスクファイルがあります。

ファイル	目的
tbl_name.MYD	データファイル
tbl_name.MYI	インデックスファイル

第16章「代替ストレージエンジン」では、テーブルを表すために各ストレージエンジンがどのようなファイルを作成するかについて説明しています。テーブル名に特殊文字が含まれている場合は、[セクション9.2.4「識別子とファイル名のマッピング」](#)で説明されているように、その文字のエンコードされたバージョンがテーブルファイルの名前に含まれます。

13.1.20.2 CREATE TEMPORARY TABLE ステートメント

テーブルの作成時に `TEMPORARY` キーワードを使用できます。`TEMPORARY` テーブルは現在のセッション内のみ表示され、セッションがクローズされると自動的に削除されます。つまり、2つの異なるセッションが同じ一時テーブル名を使用することができ、互いに、または同じ名前の既存の `TEMPORARY` 以外のテーブルと競合することはありません。(既存のテーブルは、一時テーブルが削除されるまで非表示になります。)

InnoDB では、圧縮一時テーブルはサポートされていません。`innodb_strict_mode` が有効な場合 (デフォルト)、`ROW_FORMAT=COMPRESSED` または `KEY_BLOCK_SIZE` が指定されていると、`CREATE TEMPORARY TABLE` はエラーを返します。`innodb_strict_mode` が無効な場合は、警告が発行され、圧縮されていない行形式を使用して一時テーブルが作成されます。`innodb_file_per-table` オプションは、InnoDB 一時テーブルの作成には影響しません。

`CREATE TABLE` では、`TEMPORARY` キーワードとともに使用する場合を除き、暗黙的なコミットが発生します。[セクション13.3.3「暗黙的なコミットを発生させるステートメント」](#)を参照してください。

`TEMPORARY` テーブルは、データベース (スキーマ) と非常に疎な関係を持っています。データベースを削除しても、そのデータベース内で作成されたどの `TEMPORARY` テーブルも自動的に削除されません。

一時テーブルを作成するには、`CREATE TEMPORARY TABLES` 権限が必要です。セッションが一時テーブルを作成したあと、サーバーはそのテーブルに対するそれ以上の権限チェックを実行しません。セッションの作成によって、`DROP TABLE`、`INSERT`、`UPDATE`、`SELECT` などのあらゆる操作をテーブル上で実行できます。

この動作の1つの影響として、現在のユーザーが一時テーブルを作成する権限を持たなくても、セッションが一時テーブルを操作できることがあります。現在のユーザーには `CREATE TEMPORARY TABLES` 権限はありませんが、`CREATE TEMPORARY TABLES` を持ち、一時テーブルを作成するユーザーの権限で実行される定義者コンテキストのストアードプロシージャを実行できるとします。プロシージャの実行中、セッションは定義側ユーザーの権限を使用します。プロシージャが復帰したあと、有効な権限は現在のユーザーの権限に戻り、これによって引き続き一時テーブルを表示し、一時テーブルに対してあらゆる操作を実行できるようになります。

`CREATE TEMPORARY TABLE ... LIKE` を使用して、`mysql` テーブルスペース、InnoDB システムテーブルスペース (`innodb_system`) または一般テーブルスペースに存在するテーブルの定義に基づいて空のテーブルを作成することはできません。このようなテーブルのテーブルスペース定義には、テーブルが存在するテーブルスペースを定義する `TABLESPACE` 属性が含まれており、前述のテーブルスペースは一時テーブルをサポートしていません。このようなテーブルの定義に基づいて一時テーブルを作成するには、かわりに次の構文を使用します:

```
CREATE TEMPORARY TABLE new_tbl SELECT * FROM orig_tbl LIMIT 0;
```

注記

`CREATE TEMPORARY TABLE` での `TABLESPACE = innodb_file_per_table` 句および `TABLESPACE = innodb_temporary` 句のサポートは、MySQL 8.0.13 で非推奨になりました。MySQL の将来のバージョンで削除される予定です。

13.1.20.3 CREATE TABLE ... LIKE ステートメント

`CREATE TABLE ... LIKE` を使用して、元のテーブルに定義されているカラム属性やインデックスなど、別のテーブルの定義に基づいて空のテーブルを作成します:

```
CREATE TABLE new_tbl LIKE orig_tbl;
```

このコピーは、元のテーブルと同じバージョンのテーブルストレージフォーマットを使用して作成されます。元のテーブルに対する `SELECT` 権限が必要です。

LIKE は、ビューに対してではなく、ベーステーブルに対してのみ機能します。

重要

LOCK TABLES ステートメントが有効な間は、CREATE TABLE または CREATE TABLE ... LIKE を実行できません。

CREATE TABLE ... LIKE は、CREATE TABLE と同じチェックを行います。つまり、現在の SQL モードが元のテーブルの作成時に有効なモードと異なる場合、テーブル定義は新しいモードでは無効とみなされ、ステートメントが失敗する可能性があります。

CREATE TABLE ... LIKE の場合、宛先テーブルには元のテーブルから生成されたカラム情報が保持されます。

CREATE TABLE ... LIKE の場合、宛先テーブルには元のテーブルの式のデフォルト値が保持されます。

CREATE TABLE ... LIKE の場合、宛先テーブルでは、すべての制約名が生成されることを除き、元のテーブルの CHECK 制約が保持されます。

CREATE TABLE ... LIKE は、元のテーブルや、すべての外部キー定義に対して指定されたどの DATA DIRECTORY または INDEX DIRECTORY テーブルオプションも保持しません。

元のテーブルが TEMPORARY テーブルである場合、CREATE TABLE ... LIKE は TEMPORARY を保持しません。TEMPORARY 宛先テーブルを作成するには、CREATE TEMPORARY TABLE ... LIKE を使用します。

mysql テーブルスペース、InnoDB システムテーブルスペース (innodb_system) または一般テーブルスペースで作成されたテーブルには、テーブルが存在するテーブルスペースを定義する TABLESPACE 属性がテーブル定義に含まれません。一時的な回帰のため、CREATE TABLE ... LIKE は TABLESPACE 属性を保持し、innodb_file_per_table の設定に関係なく、定義されたテーブルスペースにテーブルを作成します。このようなテーブルの定義に基づいて空のテーブルを作成するときに TABLESPACE 属性を回避するには、かわりに次の構文を使用します:

```
CREATE TABLE new_tbl SELECT * FROM orig_tbl LIMIT 0;
```

CREATE TABLE ... LIKE 操作では、すべての ENGINE_ATTRIBUTE および SECONDARY_ENGINE_ATTRIBUTE の値が新しいテーブルに適用されます。

13.1.20.4 CREATE TABLE ... SELECT ステートメント

CREATE TABLE ステートメントの最後に SELECT ステートメントを追加することによって、あるテーブルを別のテーブルから作成できます。

```
CREATE TABLE new_tbl [AS] SELECT * FROM orig_tbl;
```

MySQL は、SELECT 内のすべての要素に対して新しいカラムを作成します。例:

```
mysql> CREATE TABLE test (a INT NOT NULL AUTO_INCREMENT,
-> PRIMARY KEY (a), KEY(b))
-> ENGINE=MyISAM SELECT b,c FROM test2;
```

これにより、a、b、c の 3 つのカラムを含む MyISAM テーブルが作成されます。ENGINE オプションは CREATE TABLE ステートメントの一部であるため、SELECT のあとに使用してはいけません。これにより、構文エラーが発生します。CHARSET などのその他の CREATE TABLE オプションにも同じことが当てはまります。

SELECT ステートメントからのカラムは、テーブルにオーバーラップされるのではなく、テーブルの右側に付加されます。次の例を考えてみます。

```
mysql> SELECT * FROM foo;
+---+
| n |
+---+
| 1 |
+---+

mysql> CREATE TABLE bar (m INT) SELECT n FROM foo;
Query OK, 1 row affected (0.02 sec)
Records: 1 Duplicates: 0 Warnings: 0
```

```
mysql> SELECT * FROM bar;
+-----+-----+
|m      |n      |
+-----+-----+
|NULL   |1      |
+-----+-----+
1 row in set (0.00 sec)
```

テーブル `foo` 内の行ごとに、`foo` からの値と新しいカラムのデフォルト値を持つ行が `bar` 内に挿入されます。

`CREATE TABLE ... SELECT` の結果として得られるテーブルでは、`CREATE TABLE` 部分でのみ指定されているカラムが最初に来ます。両方の部分で指定されているカラム、または `SELECT` 部分でのみ指定されているカラムがそのあとに来ます。`SELECT` カラムのデータ型は、`CREATE TABLE` 部分にあるカラムも指定することによってオーバーライドできます。

テーブルへのデータのコピー中にエラーが発生した場合、テーブルは自動的に削除され、作成されません。ただし、MySQL 8.0.21 より前では、行ベースレプリケーションが使用されている場合、`CREATE TABLE ... SELECT` ステートメントは、テーブルを作成するトランザクションとデータを挿入するトランザクションの2つとしてバイナリログに記録されます。ステートメントがバイナリログから適用された場合、2つのトランザクション間またはデータのコピー中に障害が発生すると、空のテーブルが複製される可能性があります。この制限は、MySQL 8.0.21 では削除されます。アトミック DDL をサポートするストレージエンジンでは、行ベースレプリケーションが使用されているときに、`CREATE TABLE ... SELECT` が1つのトランザクションとして記録および適用されるようになりました。詳細は、[セクション13.1.1「アトミックデータ定義ステートメントのサポート」](#)を参照してください。

MySQL 8.0.21 の時点では、アトミック DDL 制約と外部キー制約の両方をサポートするストレージエンジンでは、行ベースレプリケーションが使用されている場合、`CREATE TABLE ... SELECT` ステートメントで外部キーの作成は許可されません。外部キー制約は、後で `ALTER TABLE` を使用して追加できます。

一意のキー値を複製する行を処理する方法を示すために、`SELECT` の前に `IGNORE` または `REPLACE` を指定できます。`IGNORE` を指定すると、一意のキー値に関して既存の行を複製する行は破棄されます。`REPLACE` を指定すると、新しい行によって同じ一意のキー値を持つ行が置き換えられます。`IGNORE` と `REPLACE` のどちらも指定されていない場合は、重複した一意のキー値によってエラーが発生します。詳細は、[IGNORE がステートメントの実行に与える影響](#)を参照してください。

MySQL 8.0.19 以降では、`CREATE TABLE ... SELECT` の `SELECT` 部分で `VALUES` ステートメントを使用することもできます。ステートメントの `VALUES` 部分には、`AS` 句を使用してテーブルのエイリアスを含める必要があります。`VALUES` からのカラムに名前を付けるには、テーブルのエイリアスを使用してカラムのエイリアスを指定します。それ以外の場合は、デフォルトのカラム名の `column_0`, `column_1`, `column_2...` が使用されます。

それ以外の場合、作成されるテーブルのカラムのネーミングは、このセクションで前述したものと同一ルールに従います。例:

```
mysql> CREATE TABLE tv1
> SELECT * FROM (VALUES ROW(1,3,5), ROW(2,4,6)) AS v;
mysql> TABLE tv1;
+-----+-----+-----+
|column_0|column_1|column_2|
+-----+-----+-----+
|1       |3       |5       |
|2       |4       |6       |
+-----+-----+-----+

mysql> CREATE TABLE tv2
> SELECT * FROM (VALUES ROW(1,3,5), ROW(2,4,6)) AS v(x,y,z);
mysql> TABLE tv2;
+-----+-----+-----+
|x      |y      |z      |
+-----+-----+-----+
|1      |3      |5      |
|2      |4      |6      |
+-----+-----+-----+

mysql> CREATE TABLE tv3 (a INT, b INT, c INT)
> SELECT * FROM (VALUES ROW(1,3,5), ROW(2,4,6)) AS v(x,y,z);
mysql> TABLE tv3;
+-----+-----+-----+
|a      |b      |c      |
+-----+-----+-----+
```

```

| a | b | c | column_0 | column_1 | column_2 |
+-----+-----+-----+-----+-----+
| NULL | NULL | NULL | 1 | 3 | 5 |
| NULL | NULL | NULL | 2 | 4 | 6 |
+-----+-----+-----+-----+

mysql> CREATE TABLE tv4 (a INT, b INT, c INT)
> SELECT * FROM (VALUES ROW(1,3,5), ROW(2,4,6)) AS v(x,y,z);
mysql> TABLE tv4;
+-----+-----+-----+-----+
| a | b | c | x | y | z |
+-----+-----+-----+-----+
| NULL | NULL | NULL | 1 | 3 | 5 |
| NULL | NULL | NULL | 2 | 4 | 6 |
+-----+-----+-----+-----+

mysql> CREATE TABLE tv5 (a INT, b INT, c INT)
> SELECT * FROM (VALUES ROW(1,3,5), ROW(2,4,6)) AS v(a,b,c);
mysql> TABLE tv5;
+-----+-----+
| a | b | c |
+-----+-----+
| 1 | 3 | 5 |
| 2 | 4 | 6 |
+-----+-----+

```

すべてのカラムを選択し、デフォルトのカラム名を使用する場合、**SELECT *** を省略できるため、テーブル **tv1** の作成に使用したステートメントも次のように記述できます:

```

mysql> CREATE TABLE tv1 VALUES ROW(1,3,5), ROW(2,4,6);
mysql> TABLE tv1;
+-----+-----+-----+
| column_0 | column_1 | column_2 |
+-----+-----+-----+
| 1 | 3 | 5 |
| 2 | 4 | 6 |
+-----+-----+

```

VALUES を **SELECT** のソースとして使用する場合、すべてのカラムが常に新しいテーブルに選択され、名前付きのテーブルから選択する場合と同じように個々のカラムを選択することはできません。次の各ステートメントではエラー (**ER_OPERAND_COLUMNS**) が生成されます:

```

CREATE TABLE tvx
  SELECT (x,z) FROM (VALUES ROW(1,3,5), ROW(2,4,6)) AS v(x,y,z);

CREATE TABLE tvx (a INT, c INT)
  SELECT (x,z) FROM (VALUES ROW(1,3,5), ROW(2,4,6)) AS v(x,y,z);

```

同様に、**SELECT** のかわりに **TABLE** ステートメントを使用できます。これは、**VALUES** の場合と同じルールに従います。ソーステーブルのすべてのカラムとその名前は、常に新しいテーブルに挿入されます。例:

```

mysql> TABLE t1;
+-----+
| a | b |
+-----+
| 1 | 2 |
| 6 | 7 |
| 10 | -4 |
| 14 | 6 |
+-----+

mysql> CREATE TABLE tt1 TABLE t1;
mysql> TABLE tt1;
+-----+
| a | b |
+-----+
| 1 | 2 |
| 6 | 7 |
| 10 | -4 |
| 14 | 6 |
+-----+

mysql> CREATE TABLE tt2 (x INT) TABLE t1;

```

```
mysql> TABLE tt2;
+-----+-----+
|x |a |b |
+-----+-----+
|NULL | 1 | 2 |
|NULL | 6 | 7 |
|NULL |10 |-4 |
|NULL |14 | 6 |
+-----+-----+
```

基礎となる **SELECT** ステートメントの行の順序を常に決定できるわけではないため、**CREATE TABLE ... IGNORE SELECT** および **CREATE TABLE ... REPLACE SELECT** ステートメントには、ステートメントベースのレプリケーションに対して安全でないフラグが付けられます。このようなステートメントは、ステートメントベースのモードの使用時にエラーログに警告を生成し、**MIXED** モードの使用時に行ベースの形式を使用してバイナリログに書き込まれます。[セクション17.2.1.1「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」](#)も参照してください。

CREATE TABLE ... SELECT は、どのインデックスも自動的に作成しません。これは、ステートメントをできるだけ柔軟にするために意図的に行われます。作成されたテーブル内にインデックスを設定する場合は、これらを **SELECT** ステートメントの前に指定するようにしてください。

```
mysql> CREATE TABLE bar (UNIQUE (n)) SELECT n FROM foo;
```

CREATE TABLE ... SELECT の場合、宛先テーブルでは、選択元テーブルのカラムが生成されたカラムであるかどうかに関する情報は保持されません。ステートメントの **SELECT** 部分では、宛先テーブルの生成されたカラムに値を割り当てることはできません。

CREATE TABLE ... SELECT の場合、宛先テーブルは元のテーブルの式のデフォルト値を保持します。

何らかのデータ型の変換が実行される可能性があります。たとえば、**AUTO_INCREMENT** 属性が保持されないため、**VARCHAR** カラムは **CHAR** カラムになることができます。リトレインされる属性は **NULL** (または **NOT NULL**) と、それらを含むカラムの場合は、**CHARACTER SET**、**COLLATION**、**COMMENT**、および **DEFAULT** 句です。

CREATE TABLE ... SELECT を使用してテーブルを作成する場合は、クエリー内のすべての関数呼び出しまたは式にエイリアスを付けるようにしてください。そうしないと、**CREATE** ステートメントが失敗するか、または好ましくないカラム名が生成される可能性があります。

```
CREATE TABLE artists_and_works
SELECT artist.name, COUNT(work.artist_id) AS number_of_works
FROM artist LEFT JOIN work ON artist.id = work.artist_id
GROUP BY artist.id;
```

作成したテーブルのカラムのデータ型を明示的に指定することもできます:

```
CREATE TABLE foo (a TINYINT NOT NULL) SELECT b+1 AS a FROM bar;
```

CREATE TABLE ... SELECT では、**IF NOT EXISTS** が指定され、ターゲットテーブルが存在する場合、宛先テーブルには何も挿入されず、ステートメントはログに記録されません。

バイナリログを使用して元のテーブルを確実に再作成できるようにするために、MySQL では、**CREATE TABLE ... SELECT** 中の並列挿入が許可されません。ただし、MySQL 8.0.21 より前では、行ベースレプリケーションが使用されているときにバイナリログから **CREATE TABLE ... SELECT** 操作が適用されると、データのコピー中にレプリケートされたテーブルでの同時挿入が許可されます。この制限は、アトミック DDL をサポートするストレージエンジン上の MySQL 8.0.21 では削除されます。詳細は、[セクション13.1.1「アトミックデータ定義ステートメントのサポート」](#)を参照してください。

CREATE TABLE new_table SELECT ... FROM old_table ... などのステートメントで **SELECT** の一部として **FOR UPDATE** を使用することはできません。それを行おうとすると、このステートメントは失敗します。

CREATE TABLE ... SELECT 操作では、**ENGINE_ATTRIBUTE** および **SECONDARY_ENGINE_ATTRIBUTE** の値はカラムにのみ適用されます。明示的に指定しないかぎり、テーブルおよびインデックスの **ENGINE_ATTRIBUTE** および **SECONDARY_ENGINE_ATTRIBUTE** の値は新しいテーブルに適用されません。

13.1.20.5 FOREIGN KEY の制約

MySQL では、テーブル間の相互参照関連データを許可する外部キー、および関連データの一貫性を保つための外部キー制約がサポートされています。

外部キー関係には、初期カラム値を保持する親テーブルと、親カラム値を参照するカラム値を持つ子テーブルが含まれます。子テーブルに外部キー制約が定義されています。

CREATE TABLE ステートメントまたは **ALTER TABLE** ステートメントで外部キー制約を定義するために不可欠な構文は次のとおりです:

```
[CONSTRAINT [symbol]] FOREIGN KEY
[index_name] (col_name, ...)
REFERENCES tbl_name (col_name,...)
[ON DELETE reference_option]
[ON UPDATE reference_option]

reference_option:
RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT
```

外部キー制約の使用方法については、このセクションの次のトピックで説明します:

- [識別子](#)
- [条件と制限](#)
- [参照アクション](#)
- [外部キー制約の例](#)
- [外部キー制約の追加](#)
- [外部キー制約の削除](#)
- [外部キーチェック](#)
- [ロック中](#)
- [外部キー定義およびメタデータ](#)
- [外部キーエラー](#)

識別子

外部キー制約のネーミングは、次のルールによって制御されます:

- **CONSTRAINT symbol** 値が使用されます (定義されている場合)。
- **CONSTRAINT symbol** 句が定義されていない場合、または **CONSTRAINT** キーワードの後に記号が含まれていない場合は、制約名が自動的に生成されます。

MySQL 8.0.16 より前では、**CONSTRAINT symbol** 句が定義されていなかった場合、または **CONSTRAINT** キーワードのあとにシンボルが含まれていなかった場合、**InnoDB** と **NDB** の両方のストレージエンジンは **FOREIGN_KEY index_name** を使用します (定義されている場合)。MySQL 8.0.16 以上では、**FOREIGN_KEY index_name** は無視されます。

- 定義されている場合、**CONSTRAINT symbol** 値はデータベース内で一意である必要があります。**symbol** が重複すると、次のようなエラーが発生: **ERROR 1005 (HY000): テーブル'test.fk1'を作成できません (errno: 121)**。
- **NDB Cluster** は、外部名を作成時と同じ大文字/小文字を使用して格納します。8.0.20 より前のバージョンでは、**SELECT** およびその他の SQL ステートメントを処理する際に、**NDB** は、**lower_case_table_names** が 0 に等しい場合に、そのようなステートメントの外部キーの名前を大/小文字を区別して格納された名前と比較していました。NDB 8.0.20 以降では、この値はこのような比較の方法に影響を与えなくなり、大文字と小文字に関係なく常に実行されます。(Bug #30512043)

FOREIGN KEY ... REFERENCES 句内のテーブルとカラムの識別子は、逆引用符 (') で囲むことができます。あるいは、**ANSI_QUOTES** SQL モードが有効になっている場合は、二重引用符 (") を使用できます。**lower_case_table_names** システム変数の設定も考慮されます。

条件と制限

外部キー制約には、次の条件および制限事項があります：

- 親テーブルと子テーブルは同じストレージエンジンを使用する必要があり、一時テーブルとして定義することはできません。
- 外部キー制約を作成するには、親テーブルに対する [REFERENCES](#) 権限が必要です。
- 外部キー内の対応するカラムと、参照されるキーは同様のデータ型を持っている必要があります。「[INTEGER](#) や [DECIMAL](#)」などの固定精度タイプのサイズと符号は同じである必要があります。文字列型の長さが同じである必要はありません。バイナリ以外の (文字の) 文字列カラムの場合、文字セットと照合順序が同じである必要があります。
- MySQL は、1 つのテーブル内のあるカラムと別のカラムの間の外部キー参照をサポートしています。(あるカラムが、それ自体への外部キー参照を持つことはできません。) このような場合、「子テーブルレコード」は同じテーブル内の依存レコードを参照します。
- MySQL では、外部キーチェックを高速に実行でき、かつテーブルスキャンが必要なくなるように、外部キーおよび参照されるキーに関するインデックスが必要です。参照しているテーブルには、外部キーカラムが同じ順序で最初のカラムとしてリストされているインデックスが存在する必要があります。このようなインデックスが存在しない場合は、参照しているテーブル上に自動的に作成されます。外部キー制約の施行に使用できる別のインデックスを作成した場合、このインデックスは後で暗黙的に削除される可能性があります。[index_name](#) が指定されている場合は、前述のように使用されます。
- InnoDB では、外部キーが任意のインデックスカラムまたはカラムのグループを参照することが許可されます。ただし、参照テーブルには、参照カラムが同じ順序の first カラムであるインデックスが必要です。InnoDB がインデックスに追加する非表示カラムも考慮されます ([セクション15.6.2.1「クラスタインデックスとセカンダリインデックス」](#) を参照)。

NDB には、外部キーとして参照されるいずれかのカラム上の明示的な一意のキー (または主キー) が必要です。

InnoDB では、標準 SQL の拡張ではありません。

- 外部キーカラム上のインデックスプリフィクスはサポートされていません。したがって、[BLOB](#) カラムおよび [TEXT](#) カラムは、常に接頭辞の長さを含む必要があるため、外部キーに含めることはできません。
- 現在、InnoDB ではユーザー定義のパーティションを持つテーブルの外部キーがサポートされていません。これには、親テーブルと子テーブルの両方が含まれます。

この制限は、[KEY](#) または [LINEAR KEY](#) によってパーティション化された [NDB](#) テーブル ([NDB](#) ストレージエンジンによってサポートされる唯一のユーザーパーティショニングタイプ) には適用されません。これらは外部キー参照を含むか、またはこのような参照のターゲットになることができます。

- 外部キー関係のテーブルは、別のストレージエンジンを使用するように変更できません。ストレージエンジンを変更するには、まず外部キー制約をすべて削除する必要があります。
- 外部キー制約は、仮想生成カラムを参照できません。

外部キー制約の MySQL 実装と SQL 標準の違いの詳細は、[セクション1.7.2.3「FOREIGN KEY 制約の違い」](#) を参照してください。

参照アクション

[UPDATE](#) または [DELETE](#) 操作が、子テーブルで一一致する行を持つ親テーブルのキー値に影響する場合、結果は [FOREIGN KEY](#) 句の [ON UPDATE](#) および [ON DELETE](#) 副次句で指定された参照アクションによって異なります。参照アクションには次のものがあります：

- [CASCADE](#): 親テーブルから行を削除または更新し、子テーブル内の一致する行を自動的に削除または更新します。[ON DELETE CASCADE](#) と [ON UPDATE CASCADE](#) の両方がサポートされています。2 つのテーブル間で、親テーブルまたは子テーブル内の同じカラムに対して機能する複数の [ON UPDATE CASCADE](#) 句を定義しないでください。

外部キーリレーションシップの両方のテーブルに [FOREIGN KEY](#) 句が定義されている場合、カスケード操作を成功させるには、一方の [FOREIGN KEY](#) 句に定義されている [ON UPDATE CASCADE](#) または [ON DELETE CASCADE](#)

副次句をもう一方の **FOREIGN KEY** 句に定義する必要があります。 **ON UPDATE CASCADE** または **ON DELETE CASCADE** 副次句が **FOREIGN KEY** 句に対してのみ定義されている場合、カスケード操作はエラーで失敗します。

注記

カスケードされた外部キーアクションはトリガーをアクティブ化しません。

- **SET NULL**: 親テーブルから行を削除または更新し、子テーブルの外部キーカラムを **NULL** に設定します。 **ON DELETE SET NULL** 句と **ON UPDATE SET NULL** 句の両方がサポートされています。

SET NULL アクションを指定する場合は、子テーブル内のカラムを **NOT NULL** として宣言していないことを確認してください。

- **RESTRICT**: 親テーブルに対する削除または更新操作を拒否します。 **RESTRICT** (または **NO ACTION**) を指定することは、**ON DELETE** または **ON UPDATE** 句を省略することと同じです。
- **NO ACTION**: 標準 SQL のキーワード。MySQL では、**RESTRICT** と同等です。MySQL Server は、参照されるテーブル内に関連する外部キー値が存在する場合、親テーブルに対する削除または更新操作を拒否します。一部のデータベースシステムは遅延チェックを備えており、その場合、**NO ACTION** は遅延チェックです。MySQL では、外部キー制約はただちにチェックされるため、**NO ACTION** は **RESTRICT** と同じです。
- **SET DEFAULT**: このアクションは MySQL パーサーによって認識されますが、**InnoDB** と **NDB** はどちらも、**ON DELETE SET DEFAULT** または **ON UPDATE SET DEFAULT** 句を含むテーブル定義を拒否します。

外部キーをサポートするストレージエンジンでは、親テーブルに一致する候補キー値がない場合、MySQL は子テーブルに外部キー値を作成しようとする **INSERT** または **UPDATE** 操作を拒否します。

指定されていない **ON DELETE** または **ON UPDATE** の場合、デフォルトのアクションは常に **NO ACTION** です。

デフォルトでは、明示的に指定された **ON DELETE NO ACTION** または **ON UPDATE NO ACTION** 句は、**SHOW CREATE TABLE** 出力または **mysqldump** でダンプされたテーブルには表示されません。同等のデフォルト以外のキーワードである **RESTRICT** は、**SHOW CREATE TABLE** 出力および **mysqldump** でダンプされたテーブルに表示されます。

NDB テーブルでは、参照先が親テーブルの主キーである場合、**ON UPDATE CASCADE** はサポートされません。

NDB 8.0.16 の時点: **NDB** テーブルの場合、子テーブルに **TEXT** 型または **BLOB** 型のいずれかのカラムが含まれる **ON DELETE CASCADE** はサポートされません。(Bug #89511、Bug #27484882)

InnoDB は、外部キー制約に対応するインデックスのレコードに対して、深さ優先検索アルゴリズムを使用してカスケード操作を実行します。

格納された生成カラムに対する外部キー制約では、**CASCADE**、**SET NULL** または **SET DEFAULT** を **ON UPDATE** 参照アクションとして使用することも、**SET NULL** または **SET DEFAULT** を **ON DELETE** 参照アクションとして使用することもできません。

格納された生成カラムのベースカラムに対する外部キー制約では、**CASCADE**、**SET NULL** または **SET DEFAULT** を **ON UPDATE** または **ON DELETE** の参照アクションとして使用できません。

外部キー制約の例

次の簡単な例では、単一カラムの外部キーを使用して **parent** テーブルと **child** テーブルを関連付けます:

```
CREATE TABLE parent (
  id INT NOT NULL,
  PRIMARY KEY (id)
) ENGINE=INNODB;

CREATE TABLE child (
  id INT,
  parent_id INT,
  INDEX par_ind (parent_id),
  FOREIGN KEY (parent_id)
  REFERENCES parent(id)
  ON DELETE CASCADE
```

```
) ENGINE=INNODB;
```

これは、`product_order` テーブルに他の 2 つのテーブルの外部キーがある、より複雑な例です。1 つの外部キーが、`product` テーブル内の 2 カラムのインデックスを参照しています。もう一方の外部キーは、`customer` テーブル内の単一カラムインデックスを参照しています。

```
CREATE TABLE product (  
  category INT NOT NULL, id INT NOT NULL,  
  price DECIMAL,  
  PRIMARY KEY(category, id)  
) ENGINE=INNODB;  
  
CREATE TABLE customer (  
  id INT NOT NULL,  
  PRIMARY KEY (id)  
) ENGINE=INNODB;  
  
CREATE TABLE product_order (  
  no INT NOT NULL AUTO_INCREMENT,  
  product_category INT NOT NULL,  
  product_id INT NOT NULL,  
  customer_id INT NOT NULL,  
  
  PRIMARY KEY(no),  
  INDEX (product_category, product_id),  
  INDEX (customer_id),  
  
  FOREIGN KEY (product_category, product_id)  
  REFERENCES product(category, id)  
  ON UPDATE CASCADE ON DELETE RESTRICT,  
  
  FOREIGN KEY (customer_id)  
  REFERENCES customer(id)  
) ENGINE=INNODB;
```

外部キー制約の追加

次の `ALTER TABLE` 構文を使用して、既存のテーブルに外部キー制約を追加できます:

```
ALTER TABLE tbl_name  
  ADD [CONSTRAINT [symbol]] FOREIGN KEY  
  [index_name] (col_name, ...)  
  REFERENCES tbl_name (col_name,...)  
  [ON DELETE reference_option]  
  [ON UPDATE reference_option]
```

外部キーは、自己参照型にする (同じテーブルを参照する) ことができます。 `ALTER TABLE`、最初に、外部キーによって参照されるカラムにインデックスを作成してご使用してテーブルに外部キー制約を追加する場合。

外部キー制約の削除

次の `ALTER TABLE` 構文を使用して、外部キー制約を削除できます:

```
ALTER TABLE tbl_name DROP FOREIGN KEY fk_symbol;
```

制約の作成時に `FOREIGN KEY` 句で `CONSTRAINT` 名が定義されていた場合は、その名前を参照して外部キー制約を削除できます。それ以外の場合は、制約名が内部的に生成されているため、その値を使用する必要があります。外部キー制約名を確認するには、`SHOW CREATE TABLE` を使用します:

```
mysql> SHOW CREATE TABLE child\G  
***** 1. row *****  
Table: child  
Create Table: CREATE TABLE `child` (  
  `id` int DEFAULT NULL,  
  `parent_id` int DEFAULT NULL,  
  KEY `par_ind` (`parent_id`),  
  CONSTRAINT `child_ibfk_1` FOREIGN KEY (`parent_id`)  
  REFERENCES `parent` (`id`) ON DELETE CASCADE  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci  
  
mysql> ALTER TABLE child DROP FOREIGN KEY `child_ibfk_1`;
```

`ALTER TABLE ... ALGORITHM=INPLACE` では、同じ `ALTER TABLE` ステートメントでの外部キーの追加および削除がサポートされています。 `ALTER TABLE ... ALGORITHM=COPY` ではサポートされていません。

外部キーチェック

外部キーチェックは、デフォルトで有効になっている `foreign_key_checks` 変数によって制御されます。通常、この変数は通常の操作中は有効のままにして、参照整合性を強制します。 `foreign_key_checks` 変数は、InnoDB テーブルの場合と同じ効果を NDB テーブルに与えます。

`foreign_key_checks` 変数は動的であり、グローバルスコープとセッションスコープの両方をサポートします。システム変数の使用の詳細は、[セクション5.1.9「システム変数の使用」](#) を参照してください。

外部キーチェックの無効化は、次の場合に役立ちます：

- 外部キー制約によって参照されるテーブルの削除。参照テーブルは、`foreign_key_checks` が無効化された後にのみ削除できます。テーブルを削除すると、テーブルに定義されている制約も削除されます。
- 外部キー関係に必要な順序とは異なる順序でテーブルをリロードします。たとえば、`mysqldump` では、子テーブルの外部キー制約など、ダンプファイル内のテーブルの正しい定義が生成されます。外部キー関係を持つテーブルのダンプファイルを簡単にリロードできるように、`mysqldump` では、`foreign_key_checks` を無効にするステートメントがダンプ出力に自動的に含まれます。これにより、ダンプファイルに外部キーに対して正しく順序付けされていないテーブルが含まれている場合に、任意の順序でテーブルをインポートできます。 `foreign_key_checks` を無効にすると、外部キーチェックが回避され、インポート操作も高速化されます。
- 外部キーチェックを回避するための `LOAD DATA` 操作の実行。
- 外部キー関係を持つテーブルに対する `ALTER TABLE` 操作の実行。

`foreign_key_checks` が無効な場合、外部キー制約は無視されますが、次の例外があります：

- テーブル定義がテーブルを参照する外部キー制約に準拠していない場合、以前に削除されたテーブルを再作成するとエラーが返されます。テーブルには正しいカラム名およびタイプが必要です。参照キーに対するインデックスも必要です。これらの要件が満たされない場合、MySQL は `errno` を参照するエラー 1005 を返します: 150:外部キー制約が正しく形成されなかったことを意味します。
- テーブルを変更すると、エラーが返されます (`errno: 150`) 変更されたテーブルに対して外部キー定義が正しく構成されていない場合。
- 外部キー制約に必要なインデックスの削除。インデックスを削除する前に、外部キー制約を削除する必要があります。
- カラムが一致しないカラムタイプを参照する外部キー制約の作成。

`foreign_key_checks` を無効にすると、次の追加の影響があります：

- データベースの外部のテーブルによって参照される外部キーを持つテーブルを含むデータベースを削除できます。
- 外部キーが他のテーブルによって参照されているテーブルを削除できます。
- `foreign_key_checks` を有効にしてもテーブルデータのスキャンはトリガーされません。つまり、`foreign_key_checks` が無効になっている間にテーブルに追加された行は、`foreign_key_checks` が再度有効になったときに一貫性がチェックされません。

ロック中

MySQL は、必要に応じて、外部キー制約によって関連付けられたテーブルにメタデータロックを拡張します。メタデータロックを拡張すると、競合する DML 操作および DDL 操作が関連するテーブルで同時に実行されなくなります。この機能を使用すると、親テーブルが変更されたときに外部キーメタデータを更新することもできます。以前の MySQL リリースでは、子テーブルが所有する外部キーメタデータは安全に更新できませんでした。

テーブルが `LOCK TABLES` で明示的にロックされている場合、外部キー制約に関連するテーブルはすべて暗黙的にオープンおよびロックされます。外部キーチェックでは、関連するテーブルに対して共有読み専用ロック (`LOCK TABLES READ`) が取得されます。カスケード更新では、操作に関連する関連テーブルに対してシェアードナッシング書き込みロック (`LOCK TABLES WRITE`) が取得されます。

外部キー定義およびメタデータ

外部キー定義を表示するには、`SHOW CREATE TABLE` を使用します:

```
mysql> SHOW CREATE TABLE child\G
***** 1. row *****
      Table: child
Create Table: CREATE TABLE `child` (
  `id` int DEFAULT NULL,
  `parent_id` int DEFAULT NULL,
  KEY `par_ind` (`parent_id`),
  CONSTRAINT `child_ibfk_1` FOREIGN KEY (`parent_id`)
  REFERENCES `parent` (`id`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

`INFORMATION_SCHEMA.KEY_COLUMN_USAGE` テーブルから、外部キーに関する情報を取得できます。このテーブルに対するクエリーの例を次に示します。

```
mysql> SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME, CONSTRAINT_NAME
FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE
WHERE REFERENCED_TABLE_SCHEMA IS NOT NULL;
+-----+-----+-----+-----+
| TABLE_SCHEMA | TABLE_NAME | COLUMN_NAME | CONSTRAINT_NAME |
+-----+-----+-----+-----+
| test         | child       | parent_id   | child_ibfk_1    |
+-----+-----+-----+-----+
```

InnoDB 外部キーに固有の情報は、`INNODB_FOREIGN` テーブルおよび `INNODB_FOREIGN_COLS` テーブルから取得できます。クエリーの例を次に示します:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FOREIGN\G
***** 1. row *****
      ID: test/child_ibfk_1
FOR_NAME: test/child
REF_NAME: test/parent
  N_COLS: 1
   TYPE: 1

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FOREIGN_COLS\G
***** 1. row *****
      ID: test/child_ibfk_1
FOR_COL_NAME: parent_id
REF_COL_NAME: id
      POS: 0
```

外部キーエラー

InnoDB テーブルに関連する外部キーエラー (通常は MySQL Server のエラー 150) が発生した場合、`SHOW ENGINE INNODB STATUS` 出力をチェックすることで、最新の外部キーエラーに関する情報を取得できます。

```
mysql> SHOW ENGINE INNODB STATUS\G
...
-----
LATEST FOREIGN KEY ERROR
-----
2018-04-12 14:57:24 0x7f97a9c91700 Transaction:
TRANSACTION 7717, ACTIVE 0 sec inserting
mysql tables in use 1, locked 1
4 lock struct(s), heap size 1136, 3 row lock(s), undo log entries 3
MySQL thread id 8, OS thread handle 140289365317376, query id 14 localhost root update
INSERT INTO child VALUES (NULL, 1), (NULL, 2), (NULL, 3), (NULL, 4), (NULL, 5), (NULL, 6)
Foreign key constraint fails for table `test`.`child`:
'
  CONSTRAINT `child_ibfk_1` FOREIGN KEY (`parent_id`) REFERENCES `parent` (`id`) ON DELETE
  CASCADE ON UPDATE CASCADE
Trying to add in child table, in index par_ind tuple:
DATA TUPLE: 2 fields;
0: len 4; hex 80000003; asc ;;
1: len 4; hex 80000003; asc ;;

But in parent table `test`.`parent`, in index PRIMARY,
the closest match we can find is record:
```



```
PHYSICAL RECORD: n_fields 3; compact format; info bits 0
0: len 4; hex 80000004; asc  ;;
1: len 6; hex 000000001e19; asc  ;;
2: len 7; hex 81000001110137; asc  7;;
...
```

警告

ユーザーがすべての親テーブルに対するテーブルレベルの権限を持っている場合、外部キー操作に関する `ER_NO_REFERENCED_ROW_2` および `ER_ROW_IS_REFERENCED_2` のエラーメッセージでは、親テーブルに関する情報が公開されます。ユーザーがすべての親テーブルに対するテーブルレベルの権限を持っていない場合は、かわりにより一般的なエラーメッセージ (`ER_NO_REFERENCED_ROW` および `ER_ROW_IS_REFERENCED`) が表示されます。

例外として、`DEFINER` 権限で実行するように定義されたストアプログラムの場合、権限が評価されるユーザーは、起動するユーザーではなく、プログラムの `DEFINER` 句のユーザーです。そのユーザーがテーブルレベルの親テーブル権限を持っている場合でも、親テーブルの情報は表示されます。この場合、ストアプログラムの作成者は、適切な条件ハンドラを含めて情報を非表示にする必要があります。

13.1.20.6 CHECK 制約

MySQL 8.0.16 より前の `CREATE TABLE` では、次の限定バージョンのテーブル `CHECK` 制約構文のみが許可されていました。この構文は解析され、無視されます:

```
CHECK (expr)
```

MySQL 8.0.16 の時点で、`CREATE TABLE` は、すべてのストレージエンジンに対して、テーブルおよびカラムの `CHECK` 制約のコア機能を許可します。`CREATE TABLE` では、テーブル制約とカラム制約の両方に対して、次の `CHECK` 制約構文を使用できます:

```
[CONSTRAINT [symbol]] CHECK (expr) [[NOT] ENFORCED]
```

オプションの `symbol` では、制約の名前を指定します。省略すると、MySQL はテーブル名、リテラル `_chk` および序数 (1、2、3 など) から名前を生成します。制約名の最大長は 64 文字です。大/小文字は区別されますが、アクセントは区別されません。

`expr` では、制約条件をブール式として指定します。この式は、テーブルの各行に対して `TRUE` または `UNKNOWN` (`NULL` 値の場合) に評価される必要があります。条件が `FALSE` に評価されると、失敗し、制約違反が発生します。違反の影響は、このセクションの後半で説明するように、実行されるステートメントによって異なります。

オプションの施行句は、制約が施行されるかどうかを示します:

- 省略するか、`ENFORCED` として指定すると、制約が作成されて適用されます。
- `NOT ENFORCED` として指定した場合、制約は作成されますが、施行されません。

`CHECK` 制約は、テーブル制約またはカラム制約のいずれかとして指定されます:

- テーブル制約はカラム定義内には表示されず、任意のテーブルのカラムを参照できます。前方参照は、後でテーブル定義に表示されるカラムに対して許可されます。
- カラム制約はカラム定義内に表示され、そのカラムのみを参照できます。

このテーブル定義について考えます。

```
CREATE TABLE t1
(
  CHECK (c1 <> c2),
  c1 INT CHECK (c1 > 10),
  c2 INT CONSTRAINT c2_positive CHECK (c2 > 0),
  c3 INT CHECK (c3 < 100),
  CONSTRAINT c1_nonzero CHECK (c1 <> 0),
  CHECK (c1 > c3)
```

);

定義には、名前付き形式および名前なし形式のテーブル制約およびカラム制約が含まれます:

- 最初の制約はテーブル制約です: カラム定義の外部で発生するため、複数のテーブルのカラムを参照できます (参照することもできます)。この制約には、まだ定義されていないカラムへのフォワード参照が含まれています。制約名が指定されていないため、MySQL は名前を生成します。
- 次の 3 つの制約はカラム制約です: それぞれがカラム定義内で発生するため、参照できるのは定義されているカラムのみです。いずれかの制約に明示的に名前が付けられます。MySQL では、それぞれの名前が生成されます。
- 最後の 2 つの制約はテーブル制約です。これらのいずれかに明示的に名前が付けられます。MySQL により、他方の名前が生成されます。

前述のように、MySQL は、指定されていない **CHECK** 制約の名前を生成します。前述のテーブル定義に対して生成された名前を確認するには、**SHOW CREATE TABLE** を使用します:

```
mysql> SHOW CREATE TABLE t1\G
***** 1. row *****
Table: t1
Create Table: CREATE TABLE `t1` (
  `c1` int(11) DEFAULT NULL,
  `c2` int(11) DEFAULT NULL,
  `c3` int(11) DEFAULT NULL,
  CONSTRAINT `c1_nonzero` CHECK ((`c1` <> 0)),
  CONSTRAINT `c2_positive` CHECK ((`c2` > 0)),
  CONSTRAINT `t1_chk_1` CHECK ((`c1` <> `c2`)),
  CONSTRAINT `t1_chk_2` CHECK ((`c1` > 10)),
  CONSTRAINT `t1_chk_3` CHECK ((`c3` < 100)),
  CONSTRAINT `t1_chk_4` CHECK ((`c1` > `c3`))
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

SQL 標準では、すべてのタイプの制約 (主キー、一意インデックス、外部キー、チェック) が同じネームスペースに属することが指定されています。MySQL では、各制約タイプにスキーマ (データベース) ごとに独自のネームスペースがあります。したがって、**CHECK** 制約名はスキーマごとに一意である必要があります。同じスキーマ内の複数のテーブルで **CHECK** 制約名を共有することはできません。(例外: **TEMPORARY** テーブルは、同じ名前の **TEMPORARY** 以外のテーブルを非表示にするため、同じ **CHECK** 制約名を持つこともできます。)

生成された制約名をテーブル名で開始すると、テーブル名もスキーマ内で一意である必要があるため、スキーマの一意性が保証されます。

CHECK の条件式は、次のルールに従う必要があります。許可されていない構造が式に含まれている場合は、エラーが発生します。

- **AUTO_INCREMENT** 属性を持つカラムおよび他のテーブルのカラムを除き、生成されていないカラムおよび生成されたカラムは許可されます。
- リテラル、決定的組み込み関数および演算子を使用できます。関数は、テーブル内の同じデータが指定された場合、接続ユーザーとは関係なく、複数の起動で同じ結果が生成される場合は決定論的です。非決定的で、この定義に失敗する関数の例: **CONNECTION_ID()**, **CURRENT_USER()**, **NOW()**。
- ストアドファンクションおよびユーザー定義関数は使用できません。
- ストアドプロシージャおよびストアドファンクションのパラメータは使用できません。
- 変数 (システム変数、ユーザー定義変数およびストアドプログラムローカル変数) は使用できません。
- サブクエリーは許可されません。

外部キー参照アクション (**ON UPDATE**、**ON DELETE**) は、**CHECK** 制約で使用されるカラムでは禁止されています。同様に、**CHECK** 制約は、外部キー参照アクションで使用されるカラムでは禁止されています。

CHECK 制約は、**INSERT**、**UPDATE**、**REPLACE**、**LOAD DATA** および **LOAD XML** ステートメントに対して評価され、制約が **FALSE** と評価されるとエラーが発生します。エラーが発生した場合、すでに適用されている変更の処理は、**厳密な SQL モード** で説明されているように、トランザクションストレージエンジンと非トランザクションストレージエンジンで異なり、厳密な SQL モードが有効になっているかどうかによっても異なります。

CHECK 制約は、INSERT IGNORE, UPDATE IGNORE, LOAD DATA ... IGNORE および LOAD XML ... IGNORE ステートメントに対して評価され、制約が FALSE と評価されると警告が発生します。問題のある行の挿入または更新はスキップされます。

制約式が宣言されたカラム型とは異なるデータ型に評価された場合、宣言された型への暗黙的な強制は、通常の MySQL 型変換ルールに従って行われます。セクション12.3「式評価での型変換」を参照してください。型変換が失敗した場合、または精度が失われた場合は、エラーが発生します。

注記

制約式の評価では、評価時に有効な SQL モードが使用されます。式のいずれかのコンポーネントが SQL モードに依存している場合、すべての使用中に SQL モードが同じでないかぎり、テーブルの使用方法によって結果が異なることがあります。

13.1.20.7 暗黙のカラム指定の変更

MySQL は場合によって、カラム指定を CREATE TABLE または ALTER TABLE ステートメントで指定されたものから暗黙のうちに変更することがあります。これらの変更は、データ型、データ型に関連付けられた属性、またはインデックス指定に対して行われる可能性があります。

すべての変更は 65,535 バイトの内部の行サイズ制限に従うため、データ型を変更しようとする一部の試みが失敗する可能性があります。セクション8.4.7「テーブルカラム数と行サイズの制限」を参照してください。

- PRIMARY KEY の一部であるカラムは、そのように宣言されていない場合でも、NOT NULL にされます。
- テーブルが作成されたとき、ENUM および SET メンバー値から末尾のスペースが自動的に削除されます。
- MySQL は、ほかの SQL データベースベンダーによって使用されている特定のデータ型を MySQL 型にマップします。セクション11.9「その他のデータベースエンジンのデータ型の使用」を参照してください。
- 特定のストレージエンジンには許可されないインデックスタイプを指定するために USING 句を含めたが、そのエンジンがクエリー結果に影響を与えずに使用できる使用可能な別のインデックスタイプが存在する場合、エンジンはその使用可能なタイプを使用します。
- 厳密な SQL モードが有効になっていない場合、長さ指定が 65535 より大きい VARCHAR カラムは TEXT に変換され、長さ指定が 65535 より大きい VARBINARY カラムは BLOB に変換されます。そうでない場合は、これらのいずれの場合にもエラーが発生します。
- 文字データ型に CHARACTER SET binary 属性を指定すると、カラムは対応するバイナリデータ型として作成されます。つまり、CHAR は BINARY になり、VARCHAR は VARBINARY になり、TEXT は BLOB になります。ENUM および SET データ型では、これは行われず、宣言されたとおりに作成されます。この定義を使用して、テーブルを指定したとします。

```
CREATE TABLE t
(
  c1 VARCHAR(10) CHARACTER SET binary,
  c2 TEXT CHARACTER SET binary,
  c3 ENUM('a','b','c') CHARACTER SET binary
);
```

結果のテーブルには、この定義が含まれています。

```
CREATE TABLE t
(
  c1 VARBINARY(10),
  c2 BLOB,
  c3 ENUM('a','b','c') CHARACTER SET binary
);
```

MySQL が、指定したものの以外のデータ型を使用したかどうかを確認するには、テーブルを作成または変更したあとに、DESCRIBE または SHOW CREATE TABLE ステートメントを発行します。

mysampack を使用してテーブルを圧縮する場合は、その他の特定のデータ型の変更が発生する場合があります。セクション16.2.3.3「圧縮テーブルの特徴」を参照してください。

13.1.20.8 CREATE TABLE および生成されるカラム

CREATE TABLE では、生成されるカラムの指定がサポートされています。生成されたカラムの値は、カラム定義に含まれる式から計算されます。

生成されたカラムは、**NDB** ストレージエンジンでもサポートされます。

次の単純な例は、**sidea** カラムおよび **sideb** カラムに右側の三角形の長さを格納し、**sidec** のハイフンの長さを計算するテーブルを示しています (もう一方の辺の二乗の合計の平方根):

```
CREATE TABLE triangle (
  sidea DOUBLE,
  sideb DOUBLE,
  sidec DOUBLE AS (SQRT(sidea * sidea + sideb * sideb))
);
INSERT INTO triangle (sidea, sideb) VALUES(1,1),(3,4),(6,8);
```

テーブルから選択すると、次の結果になります:

```
mysql> SELECT * FROM triangle;
+-----+-----+-----+
| sidea | sideb | sidec |
+-----+-----+-----+
| 1 | 1 | 1.4142135623730951 |
| 3 | 4 | 5 |
| 6 | 8 | 10 |
+-----+-----+-----+
```

triangle テーブルを使用するアプリケーションは、それらを計算する式を指定しなくても、ハイフン値にアクセスできます。

生成されるカラム定義の構文は次のとおりです:

```
col_name data_type [GENERATED ALWAYS] AS (expr)
[VIRTUAL | STORED] [NOT NULL | NULL]
[UNIQUE [KEY]] [[PRIMARY] KEY]
[COMMENT 'string']
```

AS (expr) は、カラムが生成されることを示し、カラム値の計算に使用される式を定義します。カラムの生成された性質をより明確にするために、**AS** の前に **GENERATED ALWAYS** を付けることができます。式で許可または禁止されている構造体については、あとで説明します。

VIRTUAL または **STORED** キーワードは、カラム値の格納方法を示します。これは、カラムの使用に影響します:

- **VIRTUAL**: カラム値は格納されませんが、**BEFORE** トリガーの直後に行が読み取られたときに評価されます。仮想カラムは記憶域を取りません。

InnoDB は、仮想カラムのセカンダリインデックスをサポートしています。 [セクション13.1.20.9「セカンダリインデックスと生成されたカラム」](#) を参照してください。

- **STORED**: カラム値は、行の挿入または更新時に評価および格納されます。ストアドカラムには記憶領域が必要であり、インデックス付けできます。

どちらのキーワードも指定されていない場合、デフォルトは **VIRTUAL** です。

テーブル内で **VIRTUAL** カラムと **STORED** カラムを混在させることができます。

カラムがインデックス付けされているか、**NULL** であるか、またはコメントを提供できるかを示す他の属性を指定できます。

生成されるカラム式は、次のルールに従う必要があります。許可されていない構造が式に含まれている場合は、エラーが発生します。

- リテラル、決定的組込み関数および演算子を使用できます。関数は、テーブル内の同じデータが指定された場合、接続ユーザーとは関係なく、複数の起動で同じ結果が生成される場合は決定論的です。非決定的で、この定義に失敗する関数の例: **CONNECTION_ID()**, **CURRENT_USER()**, **NOW()**。
- ストアドファンクションおよびユーザー定義関数は使用できません。

- ストアドプロシージャおよびストアドファンクションのパラメータは使用できません。
- 変数 (システム変数、ユーザー定義変数およびストアドプログラムローカル変数) は使用できません。
- サブクエリーは許可されません。
- 生成されたカラム定義は、生成された他のカラムを参照できますが、テーブル定義で以前に発生したカラムのみを参照できます。生成されたカラム定義は、その定義が以前に発生したか後で発生したかに関係なく、テーブル内の任意のベース (生成されていない) カラムを参照できます。
- `AUTO_INCREMENT` 属性は、生成されたカラム定義では使用できません。
- 生成されたカラム定義で `AUTO_INCREMENT` カラムをベースカラムとして使用することはできません。
- 式の評価によって切捨てが発生した場合、または関数への入力が正しくない場合、`CREATE TABLE` ステートメントはエラーで終了し、DDL 操作は拒否されます。

式が宣言されたカラム型とは異なるデータ型に評価された場合、宣言された型への暗黙的な強制は、通常の MySQL 型変換ルールに従って行われます。 [セクション12.3「式評価での型変換」](#) を参照してください。

生成されたカラムが `TIMESTAMP` データ型を使用している場合、`explicit_defaults_for_timestamp` の設定は無視されます。このような場合、この変数を無効にすると、`NULL` は `CURRENT_TIMESTAMP` に変換されません。MySQL 8.0.22 以降では、カラムが `NOT NULL` としても宣言されている場合、`NULL` を挿入しようとする `ER_BAD_NULL_ERROR` で明示的に拒否されます。

注記

式の評価では、評価時に有効な SQL モードが使用されます。式のいずれかのコンポーネントが SQL モードに依存している場合、すべての使用中に SQL モードが同じでないかぎり、テーブルの使用方法によって結果が異なることがあります。

`CREATE TABLE ... LIKE` の場合、宛先テーブルには元のテーブルから生成されたカラム情報が保持されます。

`CREATE TABLE ... SELECT` の場合、宛先テーブルでは、選択元テーブルのカラムが生成されたカラムであるかどうかに関する情報は保持されません。ステートメントの `SELECT` 部分では、宛先テーブルの生成されたカラムに値を割り当てることはできません。

生成されたカラムによるパーティション化が許可されます。 [テーブルのパーティション化](#) を参照してください。

格納された生成カラムに対する外部キー制約では、`CASCADE`、`SET NULL` または `SET DEFAULT` を `ON UPDATE` 参照アクションとして使用することも、`SET NULL` または `SET DEFAULT` を `ON DELETE` 参照アクションとして使用することもできません。

格納された生成カラムのベースカラムに対する外部キー制約では、`CASCADE`、`SET NULL` または `SET DEFAULT` を `ON UPDATE` または `ON DELETE` の参照アクションとして使用できません。

外部キー制約は、仮想生成カラムを参照できません。

トリガーは、`NEW.col_name` または `OLD.col_name` を使用して生成されたカラムを参照することはできません。

`INSERT`、`REPLACE` および `UPDATE` では、生成されたカラムが明示的に挿入、置換または更新される場合、許可される値は `DEFAULT` のみです。

ビュー内の生成されたカラムは、割り当て可能であるため、更新可能とみなされます。ただし、このようなカラムが明示的に更新される場合、許可される値は `DEFAULT` のみです。

生成されるカラムには、次のようないくつかのユースケースがあります:

- 仮想生成カラムは、クエリーを簡略化および統合する方法として使用できます。複雑な条件を生成されたカラムとして定義し、テーブルに対する複数のクエリーから参照して、すべてのクエリーが完全に同じ条件を使用するようにできます。
- 格納された生成カラムは、即時計算にコストがかかる複雑な条件の実体化キャッシュとして使用できます。

- 生成されたカラムは関数インデックスをシミュレートできます: 生成されたカラムを使用して関数式を定義し、インデックス付けします。これは、JSON カラムなど、直接インデックス付けできない型のカラムを操作する場合に役立ちます。詳細な例は、[JSON カラムインデックスを提供するための生成されたカラムのインデックス付け](#) を参照してください。

格納された生成カラムの場合、このアプローチのデメリットは、値が生成されたカラムの値として 2 回格納され、インデックスに 1 回格納されることです。

- 生成されたカラムがインデックス付けされている場合、オプティマイザはカラム定義に一致するクエリー式を認識し、クエリーがそのカラムを名前でも直接参照しない場合でも、クエリーの実行中にカラムのインデックスを適宜使用します。詳細は、[セクション 8.3.11 「生成されたカラムインデックスのオプティマイザによる使用」](#) を参照してください。

例:

テーブル `t1` に `first_name` カラムと `last_name` カラムが含まれており、アプリケーションが次のような式を使用してフルネームを頻繁に構成するとします:

```
SELECT CONCAT(first_name,' ',last_name) AS full_name FROM t1;
```

式を書き出さないようにするには、`t1` でビュー `v1` を作成します。これにより、式を使用せずに `full_name` を直接選択できるため、アプリケーションが簡略化されます:

```
CREATE VIEW v1 AS
SELECT *, CONCAT(first_name,' ',last_name) AS full_name FROM t1;

SELECT full_name FROM v1;
```

生成されたカラムを使用すると、ビューを定義せずに、アプリケーションで `full_name` を直接選択することもできます:

```
CREATE TABLE t1 (
  first_name VARCHAR(10),
  last_name VARCHAR(10),
  full_name VARCHAR(255) AS (CONCAT(first_name,' ',last_name))
);

SELECT full_name FROM t1;
```

13.1.20.9 セカンダリインデックスと生成されたカラム

InnoDB では、仮想生成カラムのセカンダリインデックスがサポートされます。その他のインデックスタイプはサポートされていません。仮想カラムに定義されたセカンダリインデックスは、「仮想インデックス」と呼ばれることもあります。

セカンダリインデックスは、1 つ以上の仮想カラム、または仮想カラムと通常のカラムまたは格納された生成カラムの組合せに対して作成できます。仮想カラムを含むセカンダリインデックスは、`UNIQUE` として定義できます。

セカンダリインデックスが仮想生成カラムに作成されると、生成されたカラム値はインデックスのレコードで実体化されます。インデックスが `covering index` (クエリーによって取得されたすべてのカラムを含む) の場合、生成されたカラム値は、計算された「その場で」ではなく、インデックス構造の実体化された値から取得されます。

`INSERT` および `UPDATE` の操作中にセカンダリインデックスレコードの仮想カラム値を実体化するときに計算が実行されるため、仮想カラムでセカンダリインデックスを使用する際に考慮する追加の書き込みコストがあります。追加の書き込みコストがあっても、生成される `stored` カラム (クラスタインデックスで実体化される) よりも仮想カラムのセカンダリインデックスの方が望ましい場合があり、その結果、より多くのディスク領域およびメモリーが必要なテーブルが大きくなります。セカンダリインデックスが仮想カラムに定義されていない場合、カラムの行が調査されるたびに仮想カラムの値を計算する必要があるため、読取りに追加のコストがかかります。

インデックス付けされた仮想カラムの値は MVCC ログに記録され、ロールバック中またはページ操作中に生成されたカラム値の不要な再計算を回避します。ログに記録される値のデータ長は、`COMPACT` および `REDUNDANT` の行形式では 767 バイト、`DYNAMIC` および `COMPRESSED` の行形式では 3072 バイトのインデックスキー制限によって制限されます。

仮想カラムに対するセカンダリインデックスの追加または削除はインプレース操作です。

JSON カラムインデックスを提供するための生成されたカラムのインデックス付け

他の場所で説明したように、JSON カラムは直接インデックス付けできません。このようなカラムを間接的に参照するインデックスを作成するには、次の例に示すように、インデックス付けする必要がある情報を抽出する生成カラムを定義し、生成されたカラムにインデックスを作成します：

```
mysql> CREATE TABLE jemp (
-> c JSON,
-> g INT GENERATED ALWAYS AS (c->"$.id"),
-> INDEX i (g)
-> );
Query OK, 0 rows affected (0.28 sec)

mysql> INSERT INTO jemp (c) VALUES
> ({"id": "1", "name": "Fred"}), ({"id": "2", "name": "Wilma"}),
> ({"id": "3", "name": "Barney"}), ({"id": "4", "name": "Betty"});
Query OK, 4 rows affected (0.04 sec)
Records: 4 Duplicates: 0 Warnings: 0

mysql> SELECT c->"$.name" AS name
> FROM jemp WHERE g > 2;
+-----+
| name |
+-----+
| Barney |
| Betty |
+-----+
2 rows in set (0.00 sec)

mysql> EXPLAIN SELECT c->"$.name" AS name
> FROM jemp WHERE g > 2\G
***** 1. row *****
id: 1
select_type: SIMPLE
table: jemp
partitions: NULL
type: range
possible_keys: i
key: i
key_len: 5
ref: NULL
rows: 2
filtered: 100.00
Extra: Using where
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Note
Code: 1003
Message: /* select#1 */ select json_unquote(json_extract(`test`.`jemp`.`c`, '$.name'))
AS `name` from `test`.`jemp` where (`test`.`jemp`.`g` > 2)
1 row in set (0.00 sec)
```

(この例では、表示領域に合うように最後のステートメントの出力をラップしています。)

EXPLAIN を **SELECT** または **->** または **->>** 演算子を使用する 1 つ以上の式を含む他の SQL ステートメントで **EXPLAIN** を使用する場合は、**JSON_EXTRACT()** および (必要に応じて) **JSON_UNQUOTE()** を使用して、この **EXPLAIN** ステートメントに続いてすぐに **SHOW WARNINGS** から出力の通り、これらの式はそれらの相当に変換されます：

```
mysql> EXPLAIN SELECT c->"$.name"
> FROM jemp WHERE g > 2 ORDER BY c->"$.name"\G
***** 1. row *****
id: 1
select_type: SIMPLE
table: jemp
partitions: NULL
type: range
possible_keys: i
key: i
key_len: 5
```

```

ref: NULL
rows: 2
filtered: 100.00
Extra: Using where; Using filesort
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Note
Code: 1003
Message: /* select#1 */ select json_unquote(json_extract(`test`.`jemp`.`c`, '$.name')) AS
`c->$.name` from `test`.`jemp` where (`test`.`jemp`.`g` > 2) order by
json_extract(`test`.`jemp`.`c`, '$.name')
1 row in set (0.00 sec)

```

追加情報および例については、-> および ->> の演算子と、[JSON_EXTRACT\(\)](#) および [JSON_UNQUOTE\(\)](#) の関数の説明を参照してください。

この手法を使用して、[GEOMETRY](#) カラムなど、直接インデックス付けできない他のタイプのカラムを間接的に参照するインデックスを提供することもできます。

MySQL 8.0.21 以降では、式を使用するクエリーの最適化に使用できる式を指定した [JSON_VALUE\(\)](#) 関数を使用して、[JSON](#) カラムにインデックスを作成することもできます。詳細および例については、その関数の説明を参照してください。

NDB Cluster での JSON カラムと間接インデックス

MySQL NDB Cluster では、次の条件に従って JSON カラムの間接インデックスを使用することもできます:

1. [NDB](#) は、[JSON](#) カラムの値を [BLOB](#) として内部的に処理します。つまり、[JSON](#) カラムが 1 つ以上ある [NDB](#) テーブルには主キーが必要であり、それ以外の場合はバイナリログに記録できません。
2. [NDB](#) ストレージエンジンは、仮想カラムのインデックス作成をサポートしていません。生成されるカラムのデフォルトは [VIRTUAL](#) であるため、間接インデックスを [STORED](#) として適用する生成されるカラムを明示的に指定する必要があります。

次に示す [jempn](#) テーブルの作成に使用される [CREATE TABLE](#) ステートメントは、[NDB](#) と互換性があるように変更された、前述の [jemp](#) テーブルのバージョンです:

```

CREATE TABLE jempn (
  a BIGINT(20) NOT NULL AUTO_INCREMENT PRIMARY KEY,
  c JSON DEFAULT NULL,
  g INT GENERATED ALWAYS AS (c->"$.name") STORED,
  INDEX i (g)
) ENGINE=NDB;

```

次の [INSERT](#) ステートメントを使用して、このテーブルに移入できます:

```

INSERT INTO jempn (a, c) VALUES
(NULL, '{"id": "1", "name": "Fred"}'),
(NULL, '{"id": "2", "name": "Wilma"}'),
(NULL, '{"id": "3", "name": "Barney"}'),
(NULL, '{"id": "4", "name": "Betty"}');

```

次に示すように、[NDB](#) でインデックス [i](#) を使用できるようになりました:

```

mysql> EXPLAIN SELECT c->"$.name" AS name
FROM jempn WHERE g > 2\G
***** 1. row *****
id: 1
select_type: SIMPLE
table: jempn
partitions: p0,p1
type: range
possible_keys: i
key: i
key_len: 5
ref: NULL
rows: 3
filtered: 100.00

```

```
Extra: Using where with pushed condition (`test`.`jempn`.`g` > 2)
1 row in set, 1 warning (0.00 sec)
```

```
mysql> SHOW WARNINGS\SIG
***** 1. row *****
Level: Note
Code: 1003
Message: /* select#1 */ select
json_unquote(json_extract(`test`.`jempn`.`c`, '$.name')) AS `name` from
`test`.`jempn` where (`test`.`jempn`.`g` > 2)
1 row in set (0.00 sec)
```

格納された生成カラムは [DataMemory](#) を使用し、このようなカラムのインデックスは [IndexMemory](#) を使用することに注意してください。

13.1.20.10 非表示カラム

MySQL では、MySQL 8.0.23 の時点で非表示カラムがサポートされています。非表示のカラムは通常、クエリーでは非表示ですが、明示的に参照されている場合はアクセスできます。MySQL 8.0.23 より前は、すべてのカラムが表示されます。

不可視のカラムが役立つ場合の図として、アプリケーションが [SELECT *](#) クエリーを使用してテーブルにアクセスし、アプリケーションが予期しない新しいカラムを追加するようにテーブルを変更した場合でも、変更せずに作業を続行する必要があります。 [SELECT *](#) クエリーでは、[*](#) は非表示のカラムを除くすべてのテーブルのカラムに評価されるため、解決策は新しいカラムを非表示のカラムとして追加することです。カラムは [SELECT *](#) クエリーから「非表示」のままであり、アプリケーションは引き続き以前と同様に動作します。アプリケーションの新しいバージョンは、明示的に参照することで、必要に応じて不可視のカラムを参照できます。

次の各セクションでは、MySQL で非表示カラムを処理する方法について説明します。

- [DDL ステートメントと非表示カラム](#)
- [DML ステートメントと非表示カラム](#)
- [非表示カラムのメタデータ](#)
- [バイナリログと不可視のカラム](#)

DDL ステートメントと非表示カラム

カラムはデフォルトで可視化されます。新しいカラムの可視性を明示的に指定するには、[CREATE TABLE](#) または [ALTER TABLE](#) のカラム定義の一部として [VISIBLE](#) または [INVISIBLE](#) キーワードを使用します:

```
CREATE TABLE t1 (
  i INT,
  j DATE INVISIBLE
) ENGINE = InnoDB;
ALTER TABLE t1 ADD COLUMN k INT INVISIBLE;
```

既存のカラムの可視性を変更するには、[VISIBLE](#) または [INVISIBLE](#) キーワードをいずれかの [ALTER TABLE](#) カラム変更句とともに使用します:

```
ALTER TABLE t1 CHANGE COLUMN j j DATE VISIBLE;
ALTER TABLE t1 MODIFY COLUMN j DATE INVISIBLE;
ALTER TABLE t1 ALTER COLUMN j SET VISIBLE;
```

テーブルには、少なくとも 1 つの表示可能なカラムが必要です。すべてのカラムを非表示にしようとすると、エラーが発生します。

非表示カラムでは、通常のカラム属性がサポートされます: [NULL](#), [NOT NULL](#), [AUTO_INCREMENT](#) など。

生成されたカラムは非表示にできます。

インデックス定義では、[PRIMARY KEY](#) インデックスや [UNIQUE](#) インデックスの定義など、不可視のカラムに名前を付けることができます。テーブルには 1 つ以上の可視カラムが必要ですが、インデックス定義には可視カラムは必要ありません。

テーブルから削除された不可視のカラムは、通常の方法で、カラムに名前を付けるインデックス定義から削除されません。

外部キー制約は非表示カラムに定義でき、外部キー制約は非表示カラムを参照できます。

CHECK の制約は、不可視のカラムに対して定義できます。新規または変更された行の場合、非表示カラムに対する **CHECK** 制約に違反するとエラーが発生します。

CREATE TABLE ... LIKE には不可視のカラムが含まれており、新しいテーブルでは不可視です。

CREATE TABLE ... SELECT には、**SELECT** 部分で明示的に参照されないかぎり、不可視のカラムは含まれません。ただし、明示的に参照されている場合でも、既存のテーブルに表示されないカラムは新しいテーブルに表示されます:

```
mysql> CREATE TABLE t1 (col1 INT, col2 INT INVISIBLE);
mysql> CREATE TABLE t2 AS SELECT col1, col2 FROM t1;
mysql> SHOW CREATE TABLE t2\G
***** 1. row *****
      Table: t2
Create Table: CREATE TABLE `t2` (
  `col1` int DEFAULT NULL,
  `col2` int DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

非表示を保持する必要がある場合は、**CREATE TABLE ... SELECT** ステートメントの **CREATE TABLE** 部分で非表示カラムの定義を指定します:

```
mysql> CREATE TABLE t1 (col1 INT, col2 INT INVISIBLE);
mysql> CREATE TABLE t2 (col2 INT INVISIBLE) AS SELECT col1, col2 FROM t1;
mysql> SHOW CREATE TABLE t2\G
***** 1. row *****
      Table: t2
Create Table: CREATE TABLE `t2` (
  `col1` int DEFAULT NULL,
  `col2` int DEFAULT NULL /*!80023 INVISIBLE */
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

ビューは、ビューを定義する **SELECT** ステートメントで非表示カラムを明示的に参照することで、非表示カラムを参照できます。カラムを参照するビューを定義した後にカラムの可視性を変更しても、ビューの動作は変わりません。

DML ステートメントと非表示カラム

SELECT ステートメントでは、選択リストで明示的に参照されないかぎり、非表示カラムは結果セットの一部になりません。選択リストでは、* および `tbl_name.*` の短縮形に非表示カラムは含まれません。自然結合には、非表示のカラムは含まれません。

次のステートメントの順序を考えてみます:

```
mysql> CREATE TABLE t1 (col1 INT, col2 INT INVISIBLE);
mysql> INSERT INTO t1 (col1, col2) VALUES(1, 2), (3, 4);

mysql> SELECT * FROM t1;
+----+
| col1 |
+----+
| 1 |
| 3 |
+----+

mysql> SELECT col1, col2 FROM t1;
+----+-----+
| col1 | col2 |
+----+-----+
| 1 | 2 |
| 3 | 4 |
+----+-----+
```

最初の **SELECT** は、選択リストの非表示カラム `col2` を参照しません (* には非表示カラムが含まれていないため)。したがって、`col2` はステートメントの結果に表示されません。もう一方の **SELECT** は `col2` を参照するため、結果に表示されます。

新しい行を作成するステートメントの場合、明示的に参照されて値が割り当てられないかぎり、非表示カラムには暗黙的なデフォルト値が割り当てられます。暗黙的なデフォルトの詳細は、[暗黙的なデフォルト処理](#)を参照してください。

INSERT (および **REPLACE** の場合、非置換行の場合) では、カラムリストが欠落しているか、空のカラムリスト、または非表示のカラムを含まない空でないカラムリストを使用して、暗黙的なデフォルトの割当てが行われます:

```
CREATE TABLE t1 (col1 INT, col2 INT INVISIBLE);
INSERT INTO t1 VALUES(...);
INSERT INTO t1 () VALUES(...);
INSERT INTO t1 (col1) VALUES(...);
```

最初の 2 つの **INSERT** ステートメントでは、**VALUES()** リストは表示カラムごとに値を指定する必要があり、非表示カラムは指定できません。3 つ目の **INSERT** ステートメントでは、**VALUES()** リストに名前付きカラムの数と同じ数の値を指定する必要があります。

LOAD DATA および **LOAD XML** の場合、暗黙的なデフォルト割当ては、欠落しているカラムリストまたは非表示のカラムを含まない空でないカラムリストで発生します。入力行に非表示カラムの値を含めないでください。

前述のステートメントに暗黙的なデフォルト以外の値を割り当てるには、カラムリストで非表示カラムに明示的に名前を付け、その値を指定します。

* には非表示カラムが含まれていないため、**INSERT INTO ... SELECT *** および **REPLACE INTO ... SELECT *** には非表示カラムは含まれていません。暗黙的なデフォルトの割当ては、前述のとおりに行われます。

新しい行を挿入または無視するステートメント、あるいは **PRIMARY KEY** インデックスまたは **UNIQUE** インデックスの値に基づいて既存の行を置換または変更するステートメントの場合、MySQL は非表示カラムを表示カラムと同じように処理: 非表示カラムはキー値の比較に関与します。具体的には、新しい行が一意キー値の既存の行と同じ値を持つ場合、インデックスカラムが可視か不可視かにかかわらず、次の動作が発生します:

- **IGNORE** 修飾子を使用すると、**INSERT**、**LOAD DATA** および **LOAD XML** は新しい行を無視します。
- **REPLACE** は、既存の行を新しい行に置き換えます。 **REPLACE** 修飾子を使用すると、**LOAD DATA** と **LOAD XML** は同じことを行います。
- **INSERT ... ON DUPLICATE KEY UPDATE** によって既存の行が更新されます。

UPDATE ステートメントの非表示カラムを更新するには、表示カラムの場合と同様に、非表示カラムに名前を付けて値を割り当てます。

非表示カラムのメタデータ

カラムが表示可能か非表示かに関する情報は、**INFORMATION_SCHEMA.COLUMNS** テーブルまたは **SHOW COLUMNS** 出力の **EXTRA** カラムから入手できます。例:

```
mysql> SELECT TABLE_NAME, COLUMN_NAME, EXTRA
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_SCHEMA = 'test' AND TABLE_NAME = 't1';
+-----+-----+-----+
| TABLE_NAME | COLUMN_NAME | EXTRA |
+-----+-----+-----+
| t1         | i           |      |
| t1         | j           |      |
| t1         | k           | INVISIBLE |
+-----+-----+-----+
```

カラムはデフォルトで表示されるため、**EXTRA** では表示情報は表示されません。非表示カラムの場合、**EXTRA** には **INVISIBLE** が表示されます。

SHOW CREATE TABLE では、テーブル定義に非表示のカラムが表示され、バージョン固有のコメントには **INVISIBLE** キーワードが含まれます:

```
mysql> SHOW CREATE TABLE t1\G
***** 1. row *****
Table: t1
Create Table: CREATE TABLE `t1` (
  `i` int DEFAULT NULL,
  `j` int DEFAULT NULL,
```

```
'k' int DEFAULT NULL /*!80023 INVISIBLE */
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

`mysqldump` および `mysqlpump` では `SHOW CREATE TABLE` が使用されるため、ダンプされたテーブル定義に非表示のカラムが含まれます。また、ダンプされたデータには非表示のカラム値も含まれます。

不可視のカラムをサポートしていない古いバージョンの MySQL にダンプファイルをリロードすると、バージョン固有のコメントが無視され、不可視のカラムが可視として作成されます。

バイナリログと不可視のカラム

MySQL では、非表示カラムはバイナリログ内のイベントに関して次のように処理されます:

- テーブル作成イベントには、非表示カラムの `INVISIBLE` 属性が含まれます。
- 不可視のカラムは、行イベントで可視のカラムと同様に扱われます。これらは、必要に応じて `binlog_row_image` システム変数の設定に従って含まれます。
- 行イベントが適用されると、不可視のカラムは行イベントの可視カラムと同様に扱われます。特に、使用するアルゴリズムおよびインデックスは、`slave_rows_search_algorithms` システム変数の設定に従って選択されます。
- 非表示カラムは、書き込みセットの計算時に可視カラムと同様に扱われます。特に、書き込みセットには不可視のカラムに定義されたインデックスが含まれます。
- `mysqlbinlog` コマンドには、カラムメタデータの可視性が含まれます。

13.1.20.11 NDB_TABLE オプションの設定

NDB Cluster では、`CREATE TABLE` または `ALTER TABLE` ステートメントのテーブルコメントを使用して `NDB_TABLE` オプションを指定することもできます。`NDB_TABLE` オプションは、文字列 `NDB_TABLE=` に続いて、必要に応じてカンマで区切り、1 つ以上の名前 - 値ペアで構成されます。名前と値の構文の完全な構文を次に示します:

```
COMMENT="NDB_TABLE=ndb_table_option[,ndb_table_option[,...]]"

ndb_table_option: {
  NOLOGGING={1 | 0}
  | READ_BACKUP={1 | 0}
  | PARTITION_BALANCE={FOR_RP_BY_NODE | FOR_RA_BY_NODE | FOR_RP_BY_LDM
    | FOR_RA_BY_LDM | FOR_RA_BY_LDM_X_2
    | FOR_RA_BY_LDM_X_3 | FOR_RA_BY_LDM_X_4}
  | FULLY_REPLICATED={1 | 0}
}
```

引用符で囲まれた文字列内ではスペースを使用できません。文字列では大文字と小文字は区別されません。

この方法でコメントの一部として設定できる 4 つの `NDB` テーブルオプションについては、次のいくつかの段落で詳しく説明します。

NOLOGGING: 1 を使用することは、`ndb_table_no_logging` を有効にすることに対応していますが、実際の影響はありません。プレースホルダとして提供され、主に `ALTER TABLE` ステートメントの完全性のために使用されます。

READ_BACKUP: このオプションを 1 に設定すると、`ndb_read_backup` が有効になっている場合と同じ効果があり、レプリカからの読取りが可能になります。これにより、テーブルからの読取りのパフォーマンスが大幅に向上しますが、書き込みパフォーマンスに対するコストは比較的低くなります。NDB 8.0.19 以降、1 が `READ_BACKUP` のデフォルトであり、`ndb_read_backup` のデフォルトは `ON` です (以前は、すべてのレプリカからの読み取りがデフォルトで無効になっていました)。

次に示すような `ALTER TABLE` ステートメントを使用して、既存のテーブルに対して `READ_BACKUP` をオンラインで設定できます:

```
ALTER TABLE ... ALGORITHM=INPLACE, COMMENT="NDB_TABLE=READ_BACKUP=1";
ALTER TABLE ... ALGORITHM=INPLACE, COMMENT="NDB_TABLE=READ_BACKUP=0";
```

`ALTER TABLE` の `ALGORITHM` オプションの詳細は、[セクション 23.5.11 「NDB Cluster での ALTER TABLE を使用したオンライン操作」](#) を参照してください。

PARTITION_BALANCE: パーティションの割当ておよび配置をさらに制御します。次の4つのスキームがサポートされています:

1. **FOR_RP_BY_NODE**: ノードごとに1つのパーティション。

プライマリパーティションを格納する LDM はノードごとに1つのみです。各パーティションは、すべてのノードの同じ LDM (同じ ID) に格納されます。

2. **FOR_RA_BY_NODE**: ノードグループごとに1つのパーティション。

各ノードには、単一のパーティション (プライマリレプリカまたはバックアップレプリカのいずれか) が格納されます。各パーティションは、すべてのノードの同じ LDM に格納されます。

3. **FOR_RP_BY_LDM**: 各ノードの LDM ごとに1つのパーティション (デフォルト)。

これは、**READ_BACKUP** が 1 に設定されている場合に使用する設定です。

4. **FOR_RA_BY_LDM**: 各ノードグループの LDM ごとに1つのパーティション。

これらのパーティションには、プライマリパーティションまたはバックアップパーティションを指定できます。

5. **FOR_RA_BY_LDM_X_2**: 各ノードグループの LDM ごとに2つのパーティション。

これらのパーティションには、プライマリパーティションまたはバックアップパーティションを指定できます。

6. **FOR_RA_BY_LDM_X_3**: 各ノードグループの LDM ごとに3つのパーティション。

これらのパーティションには、プライマリパーティションまたはバックアップパーティションを指定できます。

7. **FOR_RA_BY_LDM_X_4**: 各ノードグループの LDM ごとに4つのパーティション。

これらのパーティションには、プライマリパーティションまたはバックアップパーティションを指定できます。

PARTITION_BALANCE は、テーブル当たりのパーティション数を設定するための推奨インターフェースです。**MAX_ROWS** を使用したパーティションの数の強制は非推奨ですが、下位互換性のために引き続きサポートされます。MySQL NDB Cluster の将来のリリースでは削除される予定です。(Bug #81759, Bug #23544301)

FULLY_REPLICATED は、テーブルが完全にレプリケートされるかどうか、つまり各データノードにテーブルの完全なコピーがあるかどうかを制御します。テーブルの完全レプリケーションを有効にするには、**FULLY_REPLICATED=1** を使用します。

この設定は、**ndb_fully_replicated** システム変数を使用して制御することもできます。これを **ON** に設定すると、すべての新しい NDB テーブルに対してデフォルトでオプションが有効になります。デフォルトは **OFF** です。**ndb_data_node_neighbour** システム変数は完全にレプリケートされたテーブルにも使用され、完全にレプリケートされたテーブルにアクセスしたときに、この MySQL Server に対してローカルなデータノードにアクセスするようにします。

NDB テーブルの作成時にこのようなコメントを使用する **CREATE TABLE** ステートメントの例を次に示します:

```
mysql> CREATE TABLE t1 (
>   c1 INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
>   c2 VARCHAR(100),
>   c3 VARCHAR(100) )
> ENGINE=NDB
>
COMMENT="NDB_TABLE=READ_BACKUP=0,PARTITION_BALANCE=FOR_RP_BY_NODE";
```

コメントは、**SHOW CREATE TABLE** の出力の一部として表示されます。コメントのテキストは、次の例に示すように、MySQL Information Schema **TABLES** テーブルのクエリーからも使用できます:

```
mysql> SELECT TABLE_NAME, TABLE_SCHEMA, TABLE_COMMENT
> FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME="t1"
***** 1. row *****
TABLE_NAME: t1
TABLE_SCHEMA: test
TABLE_COMMENT: NDB_TABLE=READ_BACKUP=0,PARTITION_BALANCE=FOR_RP_BY_NODE
1 row in set (0.01 sec)
```

このコメント構文は、次に示すように、NDB テーブルの ALTER TABLE ステートメントでもサポートされます：

```
mysql> ALTER TABLE t1 COMMENT="NDB_TABLE=PARTITION_BALANCE=FOR_RA_BY_NODE";
Query OK, 0 rows affected (0.40 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

NDB 8.0.21 以降、TABLE_COMMENT カラムには、次のように ALTER TABLE ステートメントのあとにテーブルを再作成するために必要なコメントが表示されます：

```
mysql> SELECT TABLE_NAME, TABLE_SCHEMA, TABLE_COMMENT
-> FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME="t1"
***** 1. row *****
TABLE_NAME: t1
TABLE_SCHEMA: test
TABLE_COMMENT: NDB_TABLE=READ_BACKUP=0,PARTITION_BALANCE=FOR_RP_BY_NODE
1 row in set (0.01 sec)
```

```
mysql> SELECT TABLE_NAME, TABLE_SCHEMA, TABLE_COMMENT
> FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME="t1";
+-----+-----+-----+
| TABLE_NAME | TABLE_SCHEMA | TABLE_COMMENT |
+-----+-----+-----+
| t1 | c | NDB_TABLE=PARTITION_BALANCE=FOR_RA_BY_NODE |
| t1 | d | |
+-----+-----+-----+
2 rows in set (0.01 sec)
```

ALTER TABLE で使用されるテーブルコメントは、テーブルに存在する可能性のある既存のコメントを置き換えることに注意してください。

```
mysql> ALTER TABLE t1 COMMENT="NDB_TABLE=PARTITION_BALANCE=FOR_RA_BY_NODE";
Query OK, 0 rows affected (0.40 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> SELECT TABLE_NAME, TABLE_SCHEMA, TABLE_COMMENT
> FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME="t1";
+-----+-----+-----+
| TABLE_NAME | TABLE_SCHEMA | TABLE_COMMENT |
+-----+-----+-----+
| t1 | c | NDB_TABLE=PARTITION_BALANCE=FOR_RA_BY_NODE |
| t1 | d | |
+-----+-----+-----+
2 rows in set (0.01 sec)
```

NDB 8.0.21 より前は、ALTER TABLE で使用されていたテーブルコメントによって、テーブルに存在する可能性のある既存のコメントが置き換えられていました。これは、たとえば、READ_BACKUP 値が ALTER TABLE ステートメントによって設定された新しいコメントに引き継がれず、未指定の値がデフォルトに戻されたことを意味します。(BUG#30428829) そのため、SQL を使用して以前にコメントに設定された値を取得する方法がなくなりました。コメント値がデフォルトに戻らないようにするには、既存のコメント文字列からそのような値を保持し、ALTER TABLE に渡されるコメントに含める必要がありました。

PARTITION_BALANCE オプションの値は、ndb_desc の出力にも表示されます。ndb_desc には、READ_BACKUP および FULLY_REPLICATED オプションがテーブルに設定されているかどうか也表示されます。詳細は、このプログラムの説明を参照してください。

13.1.21 CREATE TABLESPACE ステートメント

```
CREATE [UNDO] TABLESPACE tablespace_name
```

```
InnoDB and NDB:
[ADD DATAFILE 'file_name']
[AUTOEXTEND_SIZE [=] value]
```

```
InnoDB only:
[FILE_BLOCK_SIZE = value]
[ENCRYPTION [=] {'Y' | 'N'}]
```

```
NDB only:
USE LOGFILE GROUP logfile_group
[EXTENT_SIZE [=] extent_size]
```

```
[INITIAL_SIZE [=] initial_size]
[MAX_SIZE [=] max_size]
[NODEGROUP [=] nodegroup_id]
[WAIT]
[COMMENT [=] 'string']

InnoDB and NDB:
[ENGINE [=] engine_name]

Reserved for future use:
[ENGINE_ATTRIBUTE [=] 'string']
```

このステートメントは、テーブルスペースの作成に使用されます。正確な構文とセマンティクスは、使用されるストレージエンジンによって異なります。標準の MySQL リリースでは、これは常に InnoDB テーブルスペースです。MySQL NDB Cluster は、NDB ストレージエンジンを使用したテーブルスペースもサポートしています。

- [InnoDB に関する考慮事項](#)
- [NDB Cluster に関する考慮事項](#)
- [オプション](#)
- [メモ](#)
- [InnoDB の例](#)
- [NDB の例](#)

InnoDB に関する考慮事項

CREATE TABLESPACE 構文は、一般的なテーブルスペースまたは undo テーブルスペースの作成に使用されます。undo テーブルスペースを作成するには、MySQL 8.0.14 で導入された UNDO キーワードを指定する必要があります。

一般的なテーブルスペースは共有テーブルスペースです。複数のテーブルを保持でき、すべてのテーブルの行フォーマットをサポートします。一般テーブルスペースは、データディレクトリに対して相対的または独立した場所に作成できます。

InnoDB の一般テーブルスペースを作成した後、CREATE TABLE tbl_name ... TABLESPACE [=] tablespace_name または ALTER TABLE tbl_name TABLESPACE [=] tablespace_name を使用してテーブルスペースにテーブルを追加します。詳細は、[セクション15.6.3.3 「一般テーブルスペース」](#)を参照してください。

undo テーブルスペースには undo ログが含まれます。undo テーブルスペースは、完全修飾データファイルパスを指定することで、選択した場所に作成できます。詳細は、[セクション15.6.3.4 「undo テーブルスペース」](#)を参照してください。

NDB Cluster に関する考慮事項

このステートメントは、テーブルスペースを作成するために使用します。このテーブルスペースには、「NDB Cluster ディスクデータ」テーブルの記憶領域を提供するデータファイルを 1 つ以上含めることができます ([セクション23.5.10 「NDB Cluster ディスクデータテーブル」](#)を参照)。このステートメントを使用して 1 つのデータファイルが作成され、テーブルスペースに追加されます。ALTER TABLESPACE ステートメントを使用して、テーブルスペースにデータファイルを追加できます ([セクション13.1.10 「ALTER TABLESPACE ステートメント」](#)を参照してください)。

注記

NDB Cluster ディスクデータオブジェクトはすべて同じ名前空間を共有します。つまり、各ディスクデータオブジェクトは (単に、特定の型の各ディスクデータオブジェクトというだけでなく)、一意の名前が付けられている必要があります。たとえば、テーブルスペースとログファイルグループを同じ名前にしたり、テーブルスペースとデータファイルを同じ名前にしたりすることはできません。

作成されるテーブルスペースには、USE LOGFILE GROUP 句を使用して、1 つ以上の UNDO ログファイルのログファイルグループを割り当てる必要があります。logfile_group は、CREATE LOGFILE GROUP で作成された既存の

ログファイルグループである必要があります (セクション13.1.16「CREATE LOGFILE GROUP ステートメント」を参照してください)。複数のテーブルスペースが UNDO ロギングのために同じログファイルグループを使用できます。

EXTENT_SIZE または INITIAL_SIZE を設定する場合は、my.cnf で使用されているものと同様に、数値の後に一文字の略称を付けることもできます。一般に、これは M (M バイト) または G (G バイト) のどちらかの文字です。

INITIAL_SIZE および EXTENT_SIZE は、次のように丸められます:

- EXTENT_SIZE は、最も近い 32K の倍数に切り上げられます。
- INITIAL_SIZE は、down を 32K の最も近い整数の倍数に丸めます。この結果は、EXTENT_SIZE の最も近い整数の倍数に切り上げられます (丸め後)。

注記

NDB は、データノードの再起動操作のためにテーブルスペースの 4% を予約します。この予約領域は、データ記憶域には使用できません。

今説明した丸めは明示的に実行され、このような丸めのいずれかが実行された場合は MySQL Server によって警告が発行されます。丸められた値はまた、INFORMATION_SCHEMA.FILES カラム値の計算やその他の目的のために、NDB カーネルでも使用されます。ただし、予期しない結果が発生しないようにするために、これらのオプションの指定では常に 32K の整数倍を使用することをお勧めします。

CREATE TABLESPACE を ENGINE [=] NDB とともに使用すると、各クラスタデータノードにテーブルスペースおよび関連するデータファイルが作成されます。INFORMATION_SCHEMA.FILES テーブルをクエリーすることによって、データファイルが作成されたことを確認したり、それらに関する情報を取得したりできます。(このセクションの後半の例を参照してください。)

(セクション26.15「INFORMATION_SCHEMA FILES テーブル」を参照してください。)

オプション

- ADD DATAFILE: テーブルスペースデータファイルの名前を定義します。このオプションは、NDB テーブルスペースを作成する場合は常に必要です。MySQL 8.0.14 以降の InnoDB の場合は、undo テーブルスペースを作成する場合にのみ必要です。指定したパスを含む file_name は、一重引用符または二重引用符で囲む必要があります。ファイル名 (ファイル拡張子をカウントしない) およびディレクトリ名は、少なくとも 1 バイトの長さにする必要があります。長さゼロのファイル名およびディレクトリ名はサポートされていません。

InnoDB と NDB によるデータファイルの処理方法にはかなりの違いがあるため、次の説明では 2 つのストレージエンジンについて個別に説明します。

InnoDB データファイル。 InnoDB テーブルスペースでは単一のデータファイルのみがサポートされ、その名前には .ibd 拡張子が含まれている必要があります。

InnoDB 一般テーブルスペースデータファイルをデータディレクトリ外の場所に配置するには、データディレクトリに対する完全修飾パスまたは相対パスを含めます。undo テーブルスペースには、完全修飾パスのみが許可されます。パスを指定しない場合、データディレクトリに一般テーブルスペースが作成されます。パスを指定せずに作成された undo テーブルスペースは、innodb_undo_directory 変数で定義されたディレクトリに作成されます。innodb_undo_directory 変数が定義されていない場合、undo テーブルスペースはデータディレクトリに作成されません。

暗黙的に作成された file-per-table テーブルスペースとの競合を回避するために、データディレクトリの下の子ディレクトリに InnoDB 一般テーブルスペースを作成することはサポートされていません。データディレクトリ外に一般的なテーブルスペースまたは undo テーブルスペースを作成する場合、そのディレクトリが存在し、テーブルスペースを作成する前に InnoDB で認識されている必要があります。ディレクトリを InnoDB で認識できるようにするには、innodb_directories 値または innodb_directories 値に値が追加される変数のいずれかにディレクトリを追加します。innodb_directories は読み取り専用変数です。構成するには、サーバーを再起動する必要があります。

InnoDB テーブルスペースの作成時に ADD DATAFILE 句が指定されていない場合、一意のファイル名を持つテーブルスペースデータファイルが暗黙的に作成されます。一意のファイル名は、ダッシュ (aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee) で区切られた 16 進数の 5 つのグループにフォーマットされた 128 ビット UUID です。ストレージ

エンジンで必要な場合は、ファイル拡張子が追加されます。 `.ibd` ファイル拡張子が InnoDB 一般テーブルスペースデータファイルに追加されます。レプリケーション環境では、レプリケーションソースサーバーに作成されたデータファイル名は、レプリカに作成されたデータファイル名と同じではありません。

MySQL 8.0.17 では、InnoDB テーブルスペースの作成時に `ADD DATAFILE` 句で循環ディレクトリ参照は許可されません。たとえば、次のステートメントの循環ディレクトリ参照 (`././`) は使用できません:

```
CREATE TABLESPACE ts1 ADD DATAFILE ts1.ibd 'any_directory/./ts1.ibd';
```

この制限の例外は Linux に存在し、前述のディレクトリがシンボリックリンクの場合は循環ディレクトリ参照が許可されます。たとえば、`any_directory` がシンボリックリンクの場合、前述の例のデータファイルパスは許可されます。(データファイルパスを `././` で始めることはできます。)

NDB データファイル. NDB テーブルスペースでは、任意の有効なファイル名を持つことができる複数のデータファイルがサポートされています。`ALTER TABLESPACE` ステートメントを使用して、作成後に「NDB Cluster」テーブルスペースにデータファイルを追加できます。

NDB テーブルスペースデータファイルは、デフォルトでデータノードファイルシステムディレクトリ、つまりデータノードデータディレクトリ (`DataDir`) の下の `ndb_nodeid_fs/TS` という名前のディレクトリに作成されます。`nodeid` はデータノード `Nodeid` です。データファイルをデフォルト以外の場所に配置するには、絶対ディレクトリパスまたはデフォルトの場所に対する相対パスを含めます。指定されたディレクトリが存在しない場合、NDB はそれを作成しようとします。そのためには、データノードプロセスが実行されているシステムユーザーアカウントに適切なアクセス権が必要です。

注記

データファイルに使用されるパスを決定する際、NDB は `~` (チルダ) 文字を拡張しません。

複数のデータノードが同じ物理ホスト上で実行されている場合は、次の考慮事項が適用されます:

- データファイルの作成時に絶対パスを指定することはできません。
- 各データノードに個別のデータディレクトリがないかぎり、データノードファイルシステムのディレクトリ外にテーブルスペースデータファイルを作成することはできません。
- 各データノードに独自のデータディレクトリがある場合は、このディレクトリ内の任意の場所にデータファイルを作成できます。
- 各データノードに独自のデータディレクトリがある場合は、そのホスト上で実行されている各データノードのホストファイルシステム上の一意の場所に解決されるかぎり、相対パスを使用してノードデータディレクトリ外にデータファイルを作成することもできます。
- FILE_BLOCK_SIZE:** このオプションは InnoDB の一般的なテーブルスペースに固有で、NDB によって無視され、テーブルスペースデータファイルのブロックサイズが定義されます。値はバイト単位または KB 単位で指定できます。たとえば、8K バイトのファイルブロックサイズは 8192 または 8K と指定できます。このオプションを指定しない場合、`FILE_BLOCK_SIZE` はデフォルトで `innodb_page_size` 値に設定されます。`FILE_BLOCK_SIZE` は、圧縮された InnoDB テーブル (`ROW_FORMAT=COMPRESSED`) の格納にテーブルスペースを使用する場合に必要です。この場合、テーブルスペースの作成時にテーブルスペース `FILE_BLOCK_SIZE` を定義する必要があります。
`FILE_BLOCK_SIZE` が `innodb_page_size` 値と等しい場合、テーブルスペースには、圧縮されていない行形式 (`COMPACT`、`REDUNDANT` および `DYNAMIC`) を持つテーブルのみを含めることができます。`COMPRESSED` 行形式のテーブルの物理ページサイズは、圧縮されていないテーブルとは異なります。したがって、圧縮されたテーブルは、圧縮されていないテーブルと同じテーブルスペースに共存できません。
一般的なテーブルスペースに圧縮テーブルを含めるには、`FILE_BLOCK_SIZE` を指定する必要があります。また、`FILE_BLOCK_SIZE` 値は `innodb_page_size` 値との関連で有効な圧縮ページサイズである必要があります。また、圧縮テーブル (`KEY_BLOCK_SIZE`) の物理ページサイズは `FILE_BLOCK_SIZE/1024` と同じである必要があります。たとえば、`innodb_page_size=16K` および `FILE_BLOCK_SIZE=8K` の場合、テーブルの `KEY_BLOCK_SIZE` は 8 である必要があります。詳細は、[セクション 15.6.3.3 「一般テーブルスペース」](#) を参照してください。
- USE LOGFILE GROUP:** NDB に必要です。これは、`CREATE LOGFILE GROUP` を使用して以前に作成されたログファイルグループの名前です。InnoDB ではサポートされていません。エラーで失敗します。

- **EXTENT_SIZE**: このオプションは NDB に固有であり、エラーで失敗する InnoDB ではサポートされていません。**EXTENT_SIZE** では、テーブルスペースに属するすべてのファイルで使用されるエクステントのサイズがバイト単位で設定されます。デフォルト値は 1M です。最小サイズは 32K であり、理論的な最大サイズは 2G です。ただし、実際の最大サイズはいくつかの要因によって異なります。ほとんどの場合は、エクステントサイズを変更してもパフォーマンスに測定可能な影響を与えることはないため、特別な状況を除き、常にデフォルト値を使用することをお勧めします。

エクステントは、ディスク領域の割り当ての単位です。1 つのエクステントが、そのエクステントに収容できる量のデータでいっぱいになってから、別のエクステントが使用されます。理論上は、データファイルあたり最大 65,535 (64K) 個のエクステントを使用できます。ただし、推奨される最大数は 32,768 (32K) です。1 つのデータファイルの推奨される最大サイズは 32G (つまり、32K 個のエクステント × エクステントあたり 1M バイト) です。さらに、エクステントを特定のパーティションに割り当てたあと、そのエクステントを使用して別のパーティションのデータを格納することはできません。エクステントには、複数のパーティションのデータを格納できません。これは、たとえば、**INITIAL_SIZE**(次の項目を参照) が 256 MB で、**EXTENT_SIZE** が 128M の単一のデータファイルを持つテーブルスペースにはエクステントが 2 つしかないため、最大 2 つの異なるディスクデータテーブルパーティションのデータを格納するために使用できることを意味します。

INFORMATION_SCHEMA.FILES テーブルをクエリーすることによって、特定のデータファイルに未使用のまま残っているエクステントの数を確認できるため、ファイル内の空き容量の概算値を導き出すことができます。それ以上の説明および例については、[セクション 26.15 「INFORMATION_SCHEMA FILES テーブル」](#) を参照してください。

- **INITIAL_SIZE**: このオプションは NDB に固有であり、InnoDB ではサポートされていません。エラーが発生して失敗します。

INITIAL_SIZE パラメータは、**ADD DATATFILE** を使用して特定されたデータファイルの合計サイズをバイト単位で設定します。このファイルが作成されると、そのサイズは変更できませんが、**ALTER TABLESPACE ... ADD DATAFILE** を使用してテーブルスペースにデータファイルを追加できます。

INITIAL_SIZE はオプションです。そのデフォルト値は 134217728 (128M バイト) です。

32 ビットシステム上では、**INITIAL_SIZE** のサポートされる最大値は 4294967296 (4G バイト) です。

- **AUTOEXTEND_SIZE**: MySQL 8.0.23 より前の MySQL では無視されます。MySQL 8.0.23 から、テーブルスペースが一杯になったときに InnoDB がテーブルスペースのサイズを拡張する量を定義します。設定は 4MB の倍数である必要があります。デフォルト設定は 0 で、暗黙的なデフォルト動作に従ってテーブルスペースが拡張されます。詳細は、[セクション 15.6.3.9 「テーブルスペースの AUTOEXTEND_SIZE 構成」](#) を参照してください。

使用しているストレージエンジンに関係なく、MySQL NDB Cluster 8.0 のどのリリースにも影響はありません。

- **MAX_SIZE**: 現在、MySQL では無視されます。将来の使用のために予約されています。使用されているストレージエンジンに関係なく、MySQL 8.0 または MySQL NDB Cluster 8.0 のどのリリースにも影響しません。
- **NODEGROUP**: 現在、MySQL では無視されます。将来の使用のために予約されています。使用されているストレージエンジンに関係なく、MySQL 8.0 または MySQL NDB Cluster 8.0 のどのリリースにも影響しません。
- **WAIT**: 現在、MySQL では無視されます。将来の使用のために予約されています。使用されているストレージエンジンに関係なく、MySQL 8.0 または MySQL NDB Cluster 8.0 のどのリリースにも影響しません。
- **COMMENT**: 現在、MySQL では無視されます。将来の使用のために予約されています。使用されているストレージエンジンに関係なく、MySQL 8.0 または MySQL NDB Cluster 8.0 のどのリリースにも影響しません。
- **ENCRYPTION** 句は、InnoDB 一般テーブルスペースのページレベルのデータ暗号化を有効または無効にします。一般テーブルスペースの暗号化サポートは、MySQL 8.0.13 で導入されました。

MySQL 8.0.16 では、**ENCRYPTION** 句が指定されていない場合、**default_table_encryption** 設定によって暗号化を有効にするかどうかを制御されます。**ENCRYPTION** 句は、**default_table_encryption** 設定をオーバーライドします。

ただし、`table_encryption_privilege_check` 変数が有効になっている場合、`default_table_encryption` 設定とは異なる `ENCRYPTION` 句設定を使用するには、`TABLE_ENCRYPTION_ADMIN` 権限が必要です。

暗号化対応のテーブルスペースを作成する前に、キープラグインをインストールして構成する必要があります。

一般的なテーブルスペースが暗号化されると、テーブルスペースに存在するすべてのテーブルが暗号化されます。同様に、暗号化されたテーブルスペースに作成されたテーブルも暗号化されます。

詳細は、[セクション15.13「InnoDB 保存データ暗号化」](#)を参照してください

- **ENGINE**: テーブルスペースを使用するストレージエンジンを定義します。ここで、`engine_name` はストレージエンジンの名前です。現在、標準の MySQL 8.0 リリースでは、`InnoDB` ストレージエンジンのみがサポートされています。MySQL NDB Cluster は、`NDB` と `InnoDB` の両方のテーブルスペースをサポートしています。このオプションが指定されていない場合、`default_storage_engine` システム変数の値が `ENGINE` に使用されます。
- **ENGINE_ATTRIBUTE** オプション (MySQL 8.0.21 の時点で使用可能) を使用して、プライマリストレージエンジンのテーブルスペース属性を指定します。このオプションは、将来の使用のために予約されています。

許可される値は、有効な `JSON` ドキュメントまたは空の文字列 ("") を含む文字列リテラルです。無効な `JSON` が拒否されました。

```
CREATE TABLESPACE ts1 ENGINE_ATTRIBUTE="{\"key\":\"value\"};
```

`ENGINE_ATTRIBUTE` の値は、エラーなしで繰り返すことができます。この場合、最後に指定した値が使用されません。

`ENGINE_ATTRIBUTE` 値はサーバーによってチェックされず、テーブルストレージエンジンが変更されたときにもクリアされません。

メモ

- MySQL テーブルスペースのネーミングに関するルールは、[セクション9.2「スキーマオブジェクト名」](#)を参照してください。この接頭辞はシステムで使用するために予約されているため、これらのルールに加えて、スラッシュ文字 (「/」) も使用できず、`innodb_` で始まる名前も使用できません。
- 一時一般テーブルスペースの作成はサポートされていません。
- 一般テーブルスペースでは、一時テーブルはサポートされていません。
- **TABLESPACE** オプションを `CREATE TABLE` または `ALTER TABLE` とともに使用して、`InnoDB` テーブルパーティションまたはサブパーティションを `file-per-table` テーブルスペースに割り当てることができます。すべてのパーティションは同じストレージエンジンに属している必要があります。共有 `InnoDB` テーブルスペースへのテーブルパーティションの割り当てはサポートされていません。共有テーブルスペースには、`InnoDB` システムテーブルスペースおよび一般テーブルスペースが含まれます。
- 一般テーブルスペースでは、`CREATE TABLE ... TABLESPACE` を使用した行形式のテーブルの追加がサポートされています。`innodb_file_per_table` を有効にする必要はありません。
- `innodb_strict_mode` は、一般的なテーブルスペースには適用できません。テーブルスペース管理ルールは、`innodb_strict_mode` とは無関係に厳密に適用されます。`CREATE TABLESPACE` パラメータが正しくないか、互換性がない場合、`innodb_strict_mode` の設定に関係なく操作は失敗します。`CREATE TABLE ... TABLESPACE` または `ALTER TABLE ... TABLESPACE` を使用してテーブルを一般テーブルスペースに追加すると、`innodb_strict_mode` は無視されますが、このステートメントは `innodb_strict_mode` が有効になっているかのように評価されます。
- `DROP TABLESPACE` を使用して、テーブルスペースを削除します。テーブルスペースを削除する前に、`DROP TABLE` を使用してテーブルスペースからすべてのテーブルを削除する必要があります。「`NDB Cluster`」テーブルスペースを削除する前に、1 つ以上の `ALTER TABLESPACE ... DROP DATATFILE` ステートメントを使用してすべてのデータファイルを削除する必要があります。[セクション23.5.10.1「NDB Cluster ディスクデータオブジェクト」](#)を参照してください。
- `InnoDB` 一般テーブルスペースに追加された `InnoDB` テーブルのすべての部分は、インデックスや `BLOB` ページなどの一般テーブルスペースに存在します。

テーブルスペースに割り当てられた **NDB** テーブルの場合、インデックス付けされていないカラムのみがディスクに格納され、実際にはテーブルスペースデータファイルが使用されます。すべての **NDB** テーブルのインデックスおよびインデックス付けされたカラムは、常にメモリーに保持されます。

- システムテーブルスペースと同様に、一般テーブルスペースに格納されているテーブルの切捨てまたは削除によって、新しい **InnoDB** データにのみ使用できる空き領域が一般テーブルスペース **.ibd data file** に内部的に作成されます。file-per-table テーブルスペース用であるため、領域はオペレーティングシステムに解放されません。
- 一般テーブルスペースは、どのデータベースまたはスキーマにも関連付けられていません。
- **ALTER TABLE ... DISCARD TABLESPACE** および **ALTER TABLE ... IMPORT TABLESPACE** は、一般テーブルスペースに属するテーブルではサポートされていません。
- サーバーは、一般的なテーブルスペースを参照する DDL に対してテーブルスペースレベルのメタデータロックを使用します。比較すると、サーバーは file-per-table テーブルスペースを参照する DDL に対してテーブルレベルのメタデータロックを使用します。
- 生成されたテーブルスペースまたは既存のテーブルスペースを一般テーブルスペースに変更することはできません。
- 一般的なテーブルスペース名と file-per-table テーブルスペース名の間には競合はありません。file-per-table テーブルスペース名に存在する「/」文字は、一般的なテーブルスペース名では使用できません。
- **mysqldump** および **mysqlpump** は、**InnoDB CREATE TABLESPACE** ステートメントをダンプしません。

InnoDB の例

この例では、一般的なテーブルスペースを作成し、異なる行形式の 3 つの非圧縮テーブルを追加する方法を示します。

```
mysql> CREATE TABLESPACE `ts1` ADD DATAFILE `ts1.ibd` ENGINE=INNODB;
mysql> CREATE TABLE t1 (c1 INT PRIMARY KEY) TABLESPACE ts1 ROW_FORMAT=REDUNDANT;
mysql> CREATE TABLE t2 (c1 INT PRIMARY KEY) TABLESPACE ts1 ROW_FORMAT=COMPACT;
mysql> CREATE TABLE t3 (c1 INT PRIMARY KEY) TABLESPACE ts1 ROW_FORMAT=DYNAMIC;
```

この例では、一般的なテーブルスペースを作成し、圧縮テーブルを追加する方法を示します。この例では、デフォルトの **innodb_page_size** 値が 16K であると想定しています。8192 の **FILE_BLOCK_SIZE** では、圧縮テーブルの **KEY_BLOCK_SIZE** が 8 である必要があります。

```
mysql> CREATE TABLESPACE `ts2` ADD DATAFILE `ts2.ibd` FILE_BLOCK_SIZE = 8192 Engine=InnoDB;
mysql> CREATE TABLE t4 (c1 INT PRIMARY KEY) TABLESPACE ts2 ROW_FORMAT=COMPRESSED KEY_BLOCK_SIZE=8;
```

この例では、MySQL 8.0.14 の時点でオプションの **ADD DATAFILE** 句を指定せずに一般的なテーブルスペースを作成する方法を示します。

```
mysql> CREATE TABLESPACE `ts3` ENGINE=INNODB;
```

この例では、undo テーブルスペースの作成方法を示します。

```
mysql> CREATE UNDO TABLESPACE undo_003 ADD DATAFILE `undo_003.ibu`;
```

NDB の例

mydata-1.dat という名前のデータファイルを使用して、**myts** という名前の「NDB Cluster ディスクデータ」テーブルスペースを作成するとします。NDB テーブルスペースでは、常に 1 つ以上の undo ログファイルで構成されるログファイルグループを使用する必要があります。この例では、まず、次に示す **CREATE LOGFILE GROUP** ステートメントを使用して、**myundo-1.dat** という名前の undo ログファイルを含む **mylg** という名前のログファイルグループを作成します：

```
mysql> CREATE LOGFILE GROUP myg1
-> ADD UNDOFILE `myundo-1.dat`
-> ENGINE=NDB;
Query OK, 0 rows affected (3.29 sec)
```

これで、次のステートメントを使用して、前述のテーブルスペースを作成できます：

```
mysql> CREATE TABLESPACE myts
-> ADD DATAFILE 'mydata-1.dat'
-> USE LOGFILE GROUP mylg
-> ENGINE=NDB;
Query OK, 0 rows affected (2.98 sec)
```

次に示すように、**TABLESPACE** および **STORAGE DISK** オプションを指定した **CREATE TABLE** ステートメントを使用して、「ディスクデータ」テーブルを作成できるようになりました：

```
mysql> CREATE TABLE mytable (
-> id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
-> lname VARCHAR(50) NOT NULL,
-> fname VARCHAR(50) NOT NULL,
-> dob DATE NOT NULL,
-> joined DATE NOT NULL,
-> INDEX(last_name, first_name)
-> )
-> TABLESPACE myts STORAGE DISK
-> ENGINE=NDB;
Query OK, 0 rows affected (1.41 sec)
```

id、**lname** および **fname** カラムはすべてインデックス付けされているため、実際には **mytable** の **dob** および **joined** カラムのみがディスクに格納されることに注意してください。

前述のように、**CREATE TABLESPACE** を **ENGINE [=] NDB** とともに使用すると、テーブルスペースおよび関連するデータファイルが NDB Cluster データノードごとに作成されます。次に示すように、**INFORMATION_SCHEMA.FILES** テーブルをクエリーして、データファイルが作成されたことを確認し、その情報を取得できます：

```
mysql> SELECT FILE_NAME, FILE_TYPE, LOGFILE_GROUP_NAME, STATUS, EXTRA
-> FROM INFORMATION_SCHEMA.FILES
-> WHERE TABLESPACE_NAME = 'myts';

+-----+-----+-----+-----+-----+
| file_name | file_type | logfile_group_name | status | extra |
+-----+-----+-----+-----+-----+
| mydata-1.dat | DATAFILE | mylg | NORMAL | CLUSTER_NODE=5 |
| mydata-1.dat | DATAFILE | mylg | NORMAL | CLUSTER_NODE=6 |
| NULL | TABLESPACE | mylg | NORMAL | NULL |
+-----+-----+-----+-----+-----+
3 rows in set (0.01 sec)
```

追加情報および例については、[セクション23.5.10.1「NDB Cluster ディスクデータオブジェクト」](#)を参照してください。

13.1.22 CREATE TRIGGER ステートメント

```
CREATE
[DEFINER = user]
TRIGGER trigger_name
trigger_time trigger_event
ON tbl_name FOR EACH ROW
[trigger_order]
trigger_body

trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }

trigger_order: { FOLLOWS | PRECEDES } other_trigger_name
```

このステートメントは、新しいトリガーを作成します。トリガーとは、テーブルに関連付けられ、そのテーブルに対して特定のイベントが発生するとアクティブ化される名前付きデータベースオブジェクトのことです。トリガーは、**tbl_name** という名前のテーブルに関連付けられます。これは、永続的なテーブルを指す必要があります。トリガーを **TEMPORARY** テーブルまたはビューに関連付けることはできません。

トリガー名はスキーマの名前空間内に存在します。つまり、すべてのトリガーがスキーマ内で一意の名前を持つ必要があります。異なるスキーマ内のトリガーは同じ名前を持つことができます。

このセクションでは、`CREATE TRIGGER` 構文について説明します。詳細は、[セクション25.3.1「トリガーの構文と例」](#)を参照してください。

`CREATE TRIGGER` には、このトリガーに関連付けられたテーブルに対する `TRIGGER` 権限が必要です。 `DEFINER` 句が存在する場合、[セクション25.6「ストアドオブジェクトのアクセス制御」](#)で説明されているように、必要な権限は `user` の値によって異なります。バイナリログインが有効になっている場合は、[セクション25.7「ストアドプログラムバイナリログイン」](#)で説明されているように、`CREATE TRIGGER` に `SUPER` 権限が必要になることがあります。

`DEFINER` 句は、このセクションのあとの方で説明されているように、トリガーのアクティブ化時にアクセス権限を確認するときを使用されるセキュリティコンテキストを決定します。

`trigger_time` は、このトリガーのアクション時間です。これは、トリガーが各行の変更の前またはあとにアクティブ化されることを示す `BEFORE` または `AFTER` にすることができます。

基本的なカラム値チェックはトリガーのアクティブ化の前に行われるため、`BEFORE` トリガーを使用して、カラムタイプに不適切な値を有効な値に変換することはできません。

`trigger_event` は、このトリガーをアクティブ化する操作の種類を示します。次の `trigger_event` 値が許可されます。

- `INSERT`: トリガーは、新しい行がテーブルに挿入されるたびにアクティブ化されます (たとえば、`INSERT`、`LOAD DATA` および `REPLACE` ステートメントを使用)。
- `UPDATE`: トリガーは、(`UPDATE` ステートメントなどを使用して) 行が変更されるたびにアクティブ化されます。
- `DELETE`: トリガーは、(`DELETE` ステートメントや `REPLACE` ステートメントなどを使用して) 行がテーブルから削除されるたびにアクティブになります。テーブルに対する `DROP TABLE` および `TRUNCATE TABLE` ステートメントは、`DELETE` を使用しないため、このトリガーをアクティブ化しません。また、パーティションを削除しても `DELETE` トリガーはアクティブ化されません。

`trigger_event` は、トリガーをアクティブ化する SQL ステートメントのリテラル型を表しているのではなく、テーブル操作の種類を表しています。たとえば、`INSERT` トリガーは、`INSERT` ステートメントだけでなく、`LOAD DATA` ステートメントでもアクティブ化されます。それは、どちらのステートメントもテーブルに行を挿入するためです。

この混乱を招く可能性がある例として、`INSERT INTO ... ON DUPLICATE KEY UPDATE ...` 構文があります。すべての行で `BEFORE INSERT` トリガーがアクティブ化されたあと、その行に重複キーが存在したかどうかに応じて、`AFTER INSERT` トリガーだけか、または `BEFORE UPDATE` トリガーと `AFTER UPDATE` トリガーの両方がアクティブ化されます。

注記

カスケードされた外部キーアクションはトリガーをアクティブ化しません。

同じトリガーイベントおよびアクション時間を持つ特定のテーブルに対して複数のトリガーを定義できます。たとえば、1つのテーブルに対して2つの `BEFORE UPDATE` トリガーを定義できます。デフォルトでは、同じトリガーイベントおよびアクション時間を持つトリガーは、作成された順序で実行されます。トリガー順序に影響を与えるには、`FOLLOWS` または `PRECEDES` を示す `trigger_order` 句と、同じトリガーイベントおよびアクション時間を持つ既存のトリガーの名前を指定します。 `FOLLOWS` を指定すると、新しいトリガーは既存のトリガーのあとに実行されます。 `PRECEDES` を指定すると、新しいトリガーは既存のトリガーの前に実行されます。

`trigger_body` は、トリガーがアクティブ化されるときに実行されるステートメントです。複数のステートメントを実行するには、`BEGIN ... END` 複合ステートメント構造構文を使用します。これにより、ストアドルーチン内で許可されているものと同じステートメントを使用することもできます。[セクション13.6.1「BEGIN ... END 複合ステートメント」](#)を参照してください。一部のステートメントは、トリガー内では許可されません。[セクション25.8「ストアドプログラムの制約」](#)を参照してください。

トリガー本体内では、エイリアス `OLD` と `NEW` を使用して、対象テーブル (そのトリガーに関連付けられたテーブル) 内のカラムを参照できます。 `OLD.col_name` は、更新または削除される前の既存の行のカラムを示します。 `NEW.col_name` は、挿入された新しい行、または更新されたあとの既存の行のカラムを示します。

トリガーは、`NEW.col_name` または `OLD.col_name` を使用して生成されたカラムを参照することはできません。生成されるカラムの詳細は、[セクション13.1.20.8「CREATE TABLE および生成されるカラム」](#)を参照してください。

MySQL は、トリガーが作成されたときの有効な `sql_mode` システム変数の設定を格納し、トリガーが実行を開始したときの現在のサーバー SQL モードには関係なく、常にそのトリガー本体を強制的にこの設定で実行します。

`DEFINER` 句は、トリガーのアクティブ化時にアクセス権を確認するときに使用される MySQL アカウントを指定します。 `DEFINER` 句が存在する場合、 `user` 値は `'user_name'@'host_name'`、 `CURRENT_USER` または `CURRENT_USER()` として指定された MySQL アカウントである必要があります。許可される `user` 値は、 [セクション25.6「ストアドオブジェクトのアクセス制御」](#) で説明されているように、保持する権限によって異なります。トリガーセキュリティの詳細は、そのセクションも参照してください。

`DEFINER` 句を省略すると、デフォルトの定義者は `CREATE TRIGGER` ステートメントを実行するユーザーになります。これは、明示的に `DEFINER = CURRENT_USER` を指定するのと同じです。

MySQL は、トリガー権限を確認するときに、 `DEFINER` ユーザーを次のように考慮します。

- `CREATE TRIGGER` の時点で、このステートメントを発行するユーザーには `TRIGGER` 権限が必要です。
- トリガーのアクティブ化時、権限は `DEFINER` ユーザーに対して確認されます。このユーザーには、次の権限が必要です。
 - 対象テーブルに対する `TRIGGER` 権限。
 - テーブルカラムへの参照がトリガー本体内の `OLD.col_name` または `NEW.col_name` を使用して発生した場合は、対象テーブルに対する `SELECT` 権限。
 - テーブルカラムがトリガー本体内の `SET NEW.col_name = value` 割り当てのターゲットである場合は、対象テーブルに対する `UPDATE` 権限。
 - その他のどのような権限も、通常、そのトリガーによって実行されるステートメントに必要です。

トリガー本体内で、 `CURRENT_USER` 関数は、トリガーのアクティブ化時に権限を確認するために使用されるアカウントを返します。これは、そのトリガーがアクティブ化される原因となるアクションを実行したユーザーではなく、 `DEFINER` ユーザーです。トリガー内のユーザー監査については、 [セクション6.2.22「SQL ベースのアカウントアクティビティ監査」](#) を参照してください。

`LOCK TABLES` を使用してトリガーを含むテーブルをロックした場合は、 `LOCK TABLES` とトリガーで説明されているように、そのトリガー内で使用されているテーブルもロックされます。

トリガーの使用の詳細は、 [セクション25.3.1「トリガーの構文と例」](#) を参照してください。

13.1.23 CREATE VIEW ステートメント

```
CREATE
[OR REPLACE]
[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
[DEFINER = user]
[SQL SECURITY { DEFINER | INVOKER }]
VIEW view_name [(column_list)]
AS select_statement
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

`CREATE VIEW` ステートメントは、新しいビューを作成するか、 `OR REPLACE` 句が指定されている場合は既存のビューを置き換えます。そのビューが存在しない場合、 `CREATE OR REPLACE VIEW` は `CREATE VIEW` と同じです。ビューが存在する場合は、 `CREATE OR REPLACE VIEW` によって置換されます。

ビューの使用に関する制限の詳細は、 [セクション25.9「ビューの制約」](#) を参照してください。

`select_statement` は、そのビューの定義を提供する `SELECT` ステートメントです。(ビューから選択すると、実質的には `SELECT` ステートメントを使用して選択されます。) `select_statement` では、実テーブルや他のビューから選択できます。MySQL 8.0.19 以降、 `SELECT` ステートメントは `VALUES` ステートメントをソースとして使用することも、 `CREATE TABLE ... SELECT` と同様に `TABLE` ステートメントに置き換えることもできます。

ビュー定義は作成時の「冷凍」であり、基礎となるテーブルの定義に対する後続の変更の影響を受けません。たとえば、ビューがテーブルで `SELECT *` として定義されている場合、後でテーブルに追加された新しいカラムはビューの一部にならず、テーブルから削除されたカラムはビューからの選択時にエラーになります。

ALGORITHM 句は、MySQL によるビューの処理方法に影響を与えます。**DEFINER** および **SQL SECURITY** 句は、ビューの呼び出し時にアクセス権を確認するときに使用されるセキュリティーコンテキストを指定します。**WITH CHECK OPTION** 句を指定すると、ビューによって参照されているテーブル内の行への挿入または更新を制約できます。これらの句については、このセクションのあとの方で説明されています。

CREATE VIEW ステートメントには、このビューに対する **CREATE VIEW** 権限と、**SELECT** ステートメントによって選択される各カラムに対する何らかの権限が必要です。**SELECT** ステートメントの他の場所で使用されるカラムの場合は、**SELECT** 権限が必要です。**OR REPLACE** 句が存在する場合は、このビューに対する **DROP** 権限も必要です。**DEFINER** 句が存在する場合、セクション25.6「ストアオブジェクトのアクセス制御」で説明されているように、必要な権限は **user** の値によって異なります。

ビューが参照されると、このセクションのあとの方で説明されている権限確認が発生します。

ビューはデータベースに属します。デフォルトでは、新しいビューはデフォルトデータベース内に作成されます。特定のデータベースでビューを明示的に作成するには、**db_name.view_name** 構文を使用して、ビュー名をデータベース名で修飾します：

```
CREATE VIEW test.v AS SELECT * FROM t;
```

SELECT ステートメントの修飾されていないテーブルまたはビューの名前も、デフォルトのデータベースに関して解釈されます。ビューは、テーブル名またはビュー名を適切なデータベース名で修飾することで、他のデータベース内のテーブルまたはビューを参照できます。

データベース内で、ベーステーブルとビューは同じ名前空間を共有するため、ベーステーブルとビューが同じ名前を持つことはできません。

SELECT ステートメントによって取得されるカラムは、テーブルのカラムへの単純な参照、または関数、定数値、演算子などを使用する式です。

ビューには、実テーブルと同様に、重複のない一意のカラム名が必要です。デフォルトでは、**SELECT** ステートメントによって取得されるカラムの名前はビューカラム名に使用されます。ビューカラムの明示的な名前を定義するには、カンマ区切りの識別子のリストとしてオプションの **column_list** 句を指定します。**column_list** 内の名前数は、**SELECT** ステートメントによって取得されるカラムの数と同じである必要があります。

ビューは、多くの種類の **SELECT** ステートメントから作成できます。ベーステーブルまたはほかのビューを参照できます。結合、**UNION**、およびサブクエリーを使用できます。**SELECT** では、テーブルを参照する必要はありません：

```
CREATE VIEW v_today (today) AS SELECT CURRENT_DATE;
```

次の例では、別のテーブルから2つのカラムを選択するビューと、それらのカラムから計算された式を定義します：

```
mysql> CREATE TABLE t (qty INT, price INT);
mysql> INSERT INTO t VALUES(3, 50);
mysql> CREATE VIEW v AS SELECT qty, price, qty*price AS value FROM t;
mysql> SELECT * FROM v;
+----+-----+-----+
| qty | price | value |
+----+-----+-----+
| 3   | 50    | 150   |
+----+-----+-----+
```

ビュー定義は、次の制限に従います。

- **SELECT** ステートメントは、システム変数またはユーザー定義変数を参照できません。
- ストアドプログラム内では、**SELECT** ステートメントはプログラムパラメータまたはローカル変数を参照できません。
- **SELECT** ステートメントは、準備済みステートメントのパラメータを参照できません。
- この定義で参照されているテーブルまたはビューは、すべて存在する必要があります。ビューの作成後に、定義が参照するテーブルまたはビューを削除すると、ビューを使用するとエラーになります。この種類の問題に関してビュー定義を確認するには、**CHECK TABLE** ステートメントを使用します。

- この定義は **TEMPORARY** テーブルを参照できないため、**TEMPORARY** ビューは作成できません。
- トリガーをビューに関連付けることはできません。
- **SELECT** ステートメント内のカラム名のエイリアスは (256 文字の別名の最大の長さではなく) 64 文字のカラムの最大の長さに対してチェックされます。

ORDER BY はビュー定義内で許可されていますが、独自の **ORDER BY** を含むステートメントを使用しているビューから選択した場合は無視されます。

この定義内のその他のオプションまたは句の場合は、そのビューを参照しているステートメントのオプションまたは句に追加されますが、その効果は定義されていません。たとえば、ビュー定義に **LIMIT** 句が含まれているときに、独自の **LIMIT** 句を含むステートメントを使用しているビューから選択した場合、どの制限が適用されるかは未定義です。これと同じ原則が、**SELECT** キーワードの後に続くオプション (**ALL**、**DISTINCT**、**SQL_SMALL_RESULT** など)、および **INTO**、**FOR UPDATE**、**FOR SHARE**、**LOCK IN SHARE MODE**、**PROCEDURE** などの句にも適用されません。

システム変数を変更してクエリ処理環境を変更すると、ビューから取得した結果が影響を受ける可能性があります:

```
mysql> CREATE VIEW v (mycol) AS SELECT 'abc';
Query OK, 0 rows affected (0.01 sec)

mysql> SET sql_mode = "";
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT "mycol" FROM v;
+-----+
| mycol |
+-----+
| mycol |
+-----+
1 row in set (0.01 sec)

mysql> SET sql_mode = 'ANSI_QUOTES';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT "mycol" FROM v;
+-----+
| mycol |
+-----+
| abc   |
+-----+
1 row in set (0.00 sec)
```

DEFINER および **SQL SECURITY** 句は、そのビューを参照しているステートメントの実行時に、そのビューに対するアクセス権を確認するときどの MySQL アカウントを使用するかを決定します。有効な **SQL SECURITY** 特性値は、**DEFINER** (デフォルト) および **INVOKER** です。これらは、それぞれ、そのビューを定義したユーザーまたは呼び出したユーザーが必要な権限を持っている必要があることを示します。

DEFINER 句が存在する場合、**user** 値は `'user_name'@'host_name'`、**CURRENT_USER** または **CURRENT_USER()** として指定された MySQL アカウントである必要があります。許可される **user** 値は、[セクション25.6「ストアドオブジェクトのアクセス制御」](#) で説明されているように、保持する権限によって異なります。表示セキュリティの詳細は、そのセクションも参照してください。

DEFINER 句を省略すると、デフォルトの定義者は **CREATE VIEW** ステートメントを実行するユーザーになります。これは、明示的に **DEFINER = CURRENT_USER** を指定するのと同じです。

ビュー定義内では、**CURRENT_USER** 関数はデフォルトでビューの **DEFINER** 値を返します。**SQL SECURITY INVOKER** 特性を使用して定義されたビューの場合、**CURRENT_USER** は、そのビューの呼び出し元のアカウントを返します。ビュー内のユーザー監査については、[セクション6.2.22「SQL ベースのアカウントアクティビティ監査」](#) を参照してください。

SQL SECURITY DEFINER 特性を使用して定義されたストアドルーチン内で、**CURRENT_USER** は、そのルーチンの **DEFINER** 値を返します。ビュー定義に **CURRENT_USER** の **DEFINER** 値が含まれている場合は、これにより、このようなルーチン内で定義されたビューも影響を受けます。

MySQL では、次のような表示権限がチェックされます:

- ビューの定義時に、ビュー作成者は、そのビューによってアクセスされるトップレベルのオブジェクトを使用するために必要な権限を持っている必要があります。たとえば、ビュー定義がテーブルカラムを参照している場合、作成者は、その定義の選択リスト内の各カラムに対する何らかの権限と、その定義内の別の場所で使用されている各カラムに対する **SELECT** 権限を持っている必要があります。この定義がストアドファンクションを参照している場合は、その関数を呼び出すために必要な権限のみを確認できます。関数呼び出し時に必要な権限は、その関数が実行されるときにしか確認できません。別の呼び出しでは、その関数内の別の実行パスが選択される可能性があります。
- ビューを参照するユーザーは、そのビューにアクセスするための適切な権限 (そのビューから選択するための **SELECT** や、そのビューに挿入するための **INSERT** など) を持っている必要があります。
- ビューが参照されると、そのビューによってアクセスされるオブジェクトに対する権限が、**SQL SECURITY** 特性が **DEFINER** または **INVOKER** のどちらであるかに応じて、それぞれ、そのビューの **DEFINER** アカウントによって保持されている権限または呼び出し元に対して確認されます。
- ビューへの参照によってストアドファンクションが実行される場合、その関数内で実行されるステートメントの権限確認は、その関数の **SQL SECURITY** 特性が **DEFINER** または **INVOKER** のどちらであるかによって異なります。セキュリティー特性が **DEFINER** である場合、その関数は **DEFINER** アカウントの権限で実行されます。この特性が **INVOKER** である場合、その関数は、そのビューの **SQL SECURITY** 特性によって決定される権限で実行されます。

例: あるビューがストアドファンクションに依存する可能性があり、さらにその関数がほかのストアドルーチンを呼び出す可能性があります。たとえば、次のビューはストアドファンクション **f()** を呼び出します。

```
CREATE VIEW v AS SELECT * FROM t WHERE t.id = f(t.name);
```

f() に次のようなステートメントが含まれているとします。

```
IF name IS NULL then
  CALL p1();
ELSE
  CALL p2();
END IF;
```

f() が実行される時、**f()** 内のステートメントを実行するために必要な権限を確認する必要があります。これは、**f()** 内の実行パスに応じて、**p1()** または **p2()** に対する権限が必要であることを示します。これらの権限は実行時に確認する必要があります。それらの権限を持っている必要のあるユーザーは、ビュー **v** と関数 **f()** の **SQL SECURITY** 値によって決定されます。

ビューの **DEFINER** および **SQL SECURITY** 句は、標準 SQL への拡張です。標準 SQL では、ビューは **SQL SECURITY DEFINER** のルールを使用して処理されます。標準には、ビューの定義者 (これは、ビューのスキーマの所有者と同じです) はそのビューに対する該当する権限 (**SELECT** など) を取得し、またそれらを付与することができることと記載されています。MySQL にはスキーマの「所有者」という概念がないため、MySQL では定義者を識別するための句が追加されています。 **DEFINER** 句は、標準が備えている機能、つまり、だれがそのビューを定義したかについての永続的なレコードを備えることを目的とした拡張です。 **DEFINER** のデフォルト値がビュー作成者のアカウントになっているのはそのためです。

オプションの **ALGORITHM** 句は、標準 SQL への MySQL 拡張です。これは、MySQL によるビューの処理方法に影響を与えます。 **ALGORITHM** は、 **MERGE**、 **TEMPTABLE**、または **UNDEFINED** の 3 つの値を受け取ります。詳細は、 [セクション 25.5.2 「ビュー処理アルゴリズム」](#) および [セクション 8.2.2.4 「マージまたは実体化を使用した導出テーブル、ビュー参照および共通テーブル式の最適化」](#) を参照してください。

いくつかのビューは更新可能です。つまり、これらのビューを **UPDATE**、 **DELETE**、 **INSERT** などのステートメントで使用して、ベースとなるテーブルの内容を更新できます。ビューが更新可能であるためには、そのビュー内の行とベースとなるテーブル内の行の間に 1 対 1 の関係が存在する必要があります。また、ビューを更新不可能にするその他の特定の構造構文も存在します。

ビュー内の生成されたカラムは、割り当て可能であるため、更新可能とみなされます。ただし、このようなカラムが明示的に更新される場合、許可される値は **DEFAULT** のみです。生成されるカラムの詳細は、 [セクション 13.1.20.8 「CREATE TABLE および生成されるカラム」](#) を参照してください。

更新可能なビューに対して **WITH CHECK OPTION** 句を指定すると、 **select_statement** 内の **WHERE** 句が **true** である行を除く行への挿入または更新を回避できます。

更新可能なビューに対する `WITH CHECK OPTION` 句では、そのビューが別のビューとの関連で定義されている場合、`LOCAL` および `CASCADED` キーワードによってチェックテストの範囲が決定されます。`LOCAL` キーワードは、`CHECK OPTION` を、定義されているビューのみに制限します。`CASCADED` を指定すると、ベースとなるビューに対するチェックも評価されます。どちらのキーワードも指定されていない場合、デフォルトは `CASCADED` になります。

更新可能なビューおよび `WITH CHECK OPTION` 句の詳細は、[セクション25.5.3「更新可能および挿入可能なビュー」](#)、and [セクション25.5.4「WITH CHECK OPTION 句の表示」](#) を参照してください。

13.1.24 DROP DATABASE ステートメント

```
DROP {DATABASE | SCHEMA} [IF EXISTS] db_name
```

`DROP DATABASE` は、データベース内のすべてのテーブルを削除したあと、そのデータベースを削除します。このステートメントには十分に注意してください。`DROP DATABASE` を使用するには、そのデータベースに対する `DROP` 権限が必要です。`DROP SCHEMA` は `DROP DATABASE` のシノニムです。

重要

データベースを削除しても、データベース専用が付与された権限は自動的に削除されません。手動で削除する必要があります。[セクション13.7.1.6「GRANT ステートメント」](#) を参照してください。

`IF EXISTS` は、データベースが存在しない場合にエラーが発生しないようにするために使用されます。

デフォルトデータベースが削除されると、そのデフォルトデータベースは設定解除されます (`DATABASE()` 関数が `NULL` を返します)。

シンボリックリンクされたデータベースに対して `DROP DATABASE` を使用した場合は、そのリンクと元のデータベースの両方が削除されます。

`DROP DATABASE` は、削除されたテーブルの数を返します。

`DROP DATABASE` ステートメントは、通常の操作中に MySQL 自体が作成する可能性のあるファイルおよびディレクトリを、指定されたデータベースディレクトリから削除します。これには、次のリストに示す拡張子を持つすべてのファイルが含まれます:

- `.BAK`
- `.DAT`
- `.HSH`
- `.MRG`
- `.MYD`
- `.MYI`
- `.cfg`
- `.db`
- `.ibd`
- `.ndb`

今一覧表示されたファイルを MySQL が削除したあとに、このデータベースディレクトリ内にほかのファイルやディレクトリが残っている場合は、そのデータベースディレクトリを削除できません。この場合は、残っているすべてのファイルまたはディレクトリを手動で削除してから、再度 `DROP DATABASE` ステートメントを発行する必要があります。

データベースを削除しても、そのデータベース内に作成されたどの `TEMPORARY` テーブルも削除されません。`TEMPORARY` テーブルは、それらを作成したセッションが終了すると自動的に削除されます。[セクション13.1.20.2「CREATE TEMPORARY TABLE ステートメント」](#) を参照してください。

データベースは `mysqladmin` でも削除できます。 [セクション4.5.2「mysqladmin — A MySQL Server 管理プログラム」](#) を参照してください。

13.1.25 DROP EVENT ステートメント

```
DROP EVENT [IF EXISTS] event_name
```

このステートメントは、`event_name` という名前のイベントを削除します。このイベントはただちにアクティブな状態を停止し、サーバーから完全に削除されます。

このイベントが存在しない場合は、エラー `ERROR 1517 (HY000): Unknown event 'event_name'` が発生します。これをオーバーライドし、代わりに `IF EXISTS` を使用して、このステートメントで存在しないイベントに対する警告が生成されるようにできます。

このステートメントには、削除されるイベントが属するスキーマに対する `EVENT` 権限が必要です。

13.1.26 DROP FUNCTION ステートメント

`DROP FUNCTION` ステートメントは、ストアドファンクションやユーザー定義関数 (UDF) を削除するために使用されます。

- ストアドファンクションの削除については、[セクション13.1.29「DROP PROCEDURE および DROP FUNCTION ステートメント」](#) を参照してください。
- ユーザー定義関数の削除については、[セクション13.7.4.2「ユーザー定義関数に対する DROP FUNCTION ステートメント」](#) を参照してください。

13.1.27 DROP INDEX ステートメント

```
DROP INDEX index_name ON tbl_name
[algorithm_option | lock_option] ...

algorithm_option:
ALGORITHM [=] {DEFAULT | INPLACE | COPY}

lock_option:
LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE}
```

`DROP INDEX` は、テーブル `tbl_name` から `index_name` という名前のインデックスを削除します。このステートメントは、このインデックスを削除するために `ALTER TABLE` ステートメントにマップされます。 [セクション13.1.9「ALTER TABLE ステートメント」](#) を参照してください。

主キーを削除するには、インデックス名は常に `PRIMARY` です。これは、`PRIMARY` が予約語であるため、引用符で囲まれた識別子として指定する必要があります。

```
DROP INDEX `PRIMARY` ON t;
```

`NDB` テーブルの可変幅カラム上のインデックスはオンラインで、つまり、テーブルコピーを行うことなく削除されます。テーブルは、ほかの `NDB Cluster API` ノードからのアクセスに対してロックされませんが、操作中は same API ノード上のほかの操作に対してロックされます。これは、サーバーが実行できると判断した場合は常に、そのサーバーによって自動的に実行されます。これを実行するために、特殊な SQL 構文やサーバーオプションを使用する必要はありません。

`ALGORITHM` 句および `LOCK` 句を指定して、インデックスの変更中にテーブルの読取りおよび書込みを行うためのテーブルのコピー方法および同時実行性のレベルに影響を与えることができます。これらには、`ALTER TABLE` ステートメントの場合と同じ意味があります。詳細は、[セクション13.1.9「ALTER TABLE ステートメント」](#) を参照してください。

MySQL `NDB Cluster` は、標準 MySQL Server でサポートされているものと同じ `ALGORITHM=INPLACE` 構文を使用したオンライン操作をサポートします。詳しくは [セクション23.5.11「NDB Cluster での ALTER TABLE を使用したオンライン操作」](#) をご覧ください。

13.1.28 DROP LOGFILE GROUP ステートメント

```
DROP LOGFILE GROUP logfile_group
ENGINE [=] engine_name
```

このステートメントは、`logfile_group` という名前のログファイルグループを削除します。このログファイルグループがすでに存在する必要があります。そうしないと、エラー結果が発生します。(ログファイルグループの作成については、[セクション13.1.16「CREATE LOGFILE GROUP ステートメント」](#)を参照してください。)

重要

ログファイルグループを削除する前に、そのログファイルグループを `UNDO` ロギングのために使用しているすべてのテーブルスペースを削除する必要があります。

必須の `ENGINE` 句は、削除されるログファイルグループによって使用されるストレージエンジンの名前を指定します。現在、`engine_name` に許可される値は `NDB` と `NDBCLUSTER` だけです。

`DROP LOGFILE GROUP` は `NDB Cluster` のディスクデータストレージでのみ役立ちます。[セクション23.5.10「NDB Cluster ディスクデータテーブル」](#)を参照してください。

13.1.29 DROP PROCEDURE および DROP FUNCTION ステートメント

```
DROP {PROCEDURE | FUNCTION} [IF EXISTS] sp_name
```

これらのステートメントは、ストアドルーチン (ストアプロシージャまたはストアファンクション) を削除するために使用されます。つまり、指定されたルーチンがサーバーから削除されます。(`DROP FUNCTION` は、ユーザー定義関数の削除にも使用されます。[セクション13.7.4.2「ユーザー定義関数に対する DROP FUNCTION ステートメント」](#)を参照してください。)

ストアドルーチンを削除するには、そのストアドルーチンに対する `ALTER ROUTINE` 権限が必要です。(`automatic_sp_privileges` システム変数が有効になっている場合は、その権限と `EXECUTE` が自動的に、そのルーチンが作成される時はルーチン作成者に付与され、そのルーチンが削除される時は作成者から削除されます。[セクション25.2.2「ストアドルーチンと MySQL 権限」](#)を参照してください。)

`IF EXISTS` 句は MySQL 拡張です。これは、プロシージャまたは関数が存在しない場合にエラーが発生しないようにします。`SHOW WARNINGS` で表示できる警告が生成されます。

`DROP FUNCTION` はまた、ユーザー定義関数を削除するためにも使用されます([セクション13.7.4.2「ユーザー定義関数に対する DROP FUNCTION ステートメント」](#)を参照してください)。

13.1.30 DROP SERVER ステートメント

```
DROP SERVER [IF EXISTS] server_name
```

`server_name` という名前のサーバーのサーバー定義を削除します。`mysql.servers` テーブル内の対応する行が削除されます。このステートメントには、`SUPER` 権限が必要です。

テーブルのサーバーを削除しても、作成されるときにこの接続情報を使用したどの `FEDERATED` テーブルにも影響を与えません。[セクション13.1.18「CREATE SERVER ステートメント」](#)を参照してください。

`DROP SERVER` によって暗黙的なコミットが発生します。[セクション13.3.3「暗黙的なコミットを発生させるステートメント」](#)を参照してください。

使用されているロギング形式に関係なく、`DROP SERVER` はバイナリログに書き込まれません。

13.1.31 DROP SPATIAL REFERENCE SYSTEM ステートメント

```
DROP SPATIAL REFERENCE SYSTEM
[IF EXISTS]
srid
```

`srid: 32-bit unsigned integer`

このステートメントは、データディクショナリから `spatial reference system` (SRS) 定義を削除します。`SUPER` 権限が必要です。

例:

```
DROP SPATIAL REFERENCE SYSTEM 4120;
```

SRID 値を持つ SRS 定義が存在しない場合は、**IF EXISTS** が指定されていないかぎりエラーが発生します。その場合、エラーではなく警告が発生します。

SRID 値が既存のテーブルのカラムで使用されている場合は、エラーが発生します。例:

```
mysql> DROP SPATIAL REFERENCE SYSTEM 4326;
ERROR 3716 (SR005): Can't modify SRID 4326. There is at
least one column depending on it.
```

SRID を使用するカラムを識別するには、次のクエリーを使用します:

```
SELECT * FROM INFORMATION_SCHEMA.ST_GEOMETRY_COLUMNS WHERE SRS_ID=4326;
```

SRID 値は 32 ビットの符号なし整数の範囲内である必要がありますが、次の制限があります:

- SRID 0 は有効な SRID ですが、**DROP SPATIAL REFERENCE SYSTEM** では使用できません。
- 値が予約された SRID 範囲内にある場合は、警告が発生します。予約済の範囲は、[0, 32767] (EPSG で予約済)、[60,000,000, 69,999,999] (EPSG で予約済) および [2,000,000,000, 2,147,483,647] (MySQL で予約済) です。EPSG は「[欧州石油調査グループ](#)」を表します。
- ユーザーは、予約された範囲内の SRID を持つ SRS を削除しないでください。システムにインストールされた SRS が削除された場合、SRS 定義を MySQL のアップグレード用に再作成できます。

13.1.32 DROP TABLE ステートメント

```
DROP [TEMPORARY] TABLE [IF EXISTS]
tbl_name [, tbl_name] ...
[RESTRICT | CASCADE]
```

DROP TABLE は、1 つ以上のテーブルを削除します。各テーブルに対する **DROP** 権限が必要です。

このステートメントを使用した注意してください。テーブルごとに、テーブル定義およびすべてのテーブルデータが削除されます。テーブルがパーティション化されている場合、ステートメントはテーブル定義、そのすべてのパーティション、それらのパーティションに格納されているすべてのデータ、および削除されたテーブルに関連付けられているすべてのパーティション定義を削除します。

テーブルを削除すると、そのテーブルのトリガーも削除されます。

DROP TABLE では、**TEMPORARY** キーワードとともに使用する場合を除き、暗黙的なコミットが発生します。[セクション 13.3.3 「暗黙的なコミットを発生させるステートメント」](#) を参照してください。

重要

テーブルを削除しても、そのテーブル専用に付与された権限は自動的に削除されません。手動で削除する必要があります。[セクション 13.7.1.6 「GRANT ステートメント」](#) を参照してください。

引数リストに指定されたテーブルが存在しない場合、**DROP TABLE** の動作は **IF EXISTS** 句が指定されているかどうかによって異なります:

- **IF EXISTS** がない場合、ステートメントは失敗し、削除できなかった存在しないテーブルを示すエラーが表示され、変更は行われません。
- **IF EXISTS** では、存在しないテーブルに対してエラーは発生しません。このステートメントは、存在するすべての名前付きテーブルを削除し、存在しないテーブルごとに **NOTE** 診断を生成します。これらのノートは、**SHOW WARNINGS** で表示できます。[セクション 13.7.7.42 「SHOW WARNINGS ステートメント」](#) を参照してください。

IF EXISTS は、データディクショナリにエントリがあり、記憶域エンジンによって管理されるテーブルがない異常な状況でテーブルを削除する場合にも役立ちます。(たとえば、ストレージエンジンからテーブルを削除したあと、データディクショナリエントリを削除する前に異常なサーバーイグジットが発生した場合。)

TEMPORARY キーワードには、次の効果があります。

- このステートメントは、TEMPORARY テーブルのみを削除します。
- このステートメントは暗黙的なコミットを引き起こしません。
- アクセス権は確認されません。TEMPORARY テーブルは、それを作成したセッションでのみ表示されるため、チェックは必要ありません。

TEMPORARY 以外のテーブルを誤って削除しないようにするには、TEMPORARY キーワードを含めることをお勧めします。

RESTRICT および CASCADE キーワードは何も行いません。他のデータベースシステムからの移植を容易にすることができます。

DROP TABLE はすべての innodb_force_recovery 設定でサポートされているわけではありません。セクション 15.21.2 「InnoDB のリカバリの強制的な実行」を参照してください。

13.1.33 DROP TABLESPACE ステートメント

```
DROP [UNDO] TABLESPACE tablespace_name  
[ENGINE [=] engine_name]
```

このステートメントは、CREATE TABLESPACE を使用して以前に作成されたテーブルスペースを削除します。NDB および InnoDB ストレージエンジンでサポートされています。

undo テーブルスペースを削除するには、MySQL 8.0.14 で導入された UNDO キーワードを指定する必要があります。CREATE UNDO TABLESPACE 構文を使用して作成された undo テーブルスペースのみを削除できます。undo テーブルスペースは、削除する前に empty 状態である必要があります。詳細は、セクション 15.6.3.4 「undo テーブルスペース」を参照してください。

ENGINE は、テーブルスペースを使用するストレージエンジンを設定します。ここで、engine_name はストレージエンジンの名前です。現在、InnoDB および NDB の値がサポートされています。設定しない場合、default_storage_engine の値が使用されます。テーブルスペースの作成に使用されたストレージエンジンと同じでない場合、DROP TABLESPACE ステートメントは失敗します。

tablespace_name は、MySQL では大/小文字が区別される識別子です。

InnoDB 一般テーブルスペースの場合、DROP TABLESPACE 操作の前にすべてのテーブルをテーブルスペースから削除する必要があります。テーブルスペースが空でない場合、DROP TABLESPACE はエラーを返します。

削除する NDB テーブルスペースにデータファイルを含めることはできません。つまり、NDB テーブルスペースを削除する前に、まず ALTER TABLESPACE ... DROP DATAFILE を使用して各データファイルを削除する必要があります。

メモ

- 一般的な InnoDB テーブルスペースは、テーブルスペースの最後のテーブルが削除されても自動的に削除されません。テーブルスペースは、DROP TABLESPACE tablespace_name を使用して明示的に削除する必要があります。
- DROP DATABASE 操作では、一般的なテーブルスペースに属するテーブルを削除できますが、そのテーブルスペースに属するすべてのテーブルを削除しても、テーブルスペースは削除できません。テーブルスペースは、DROP TABLESPACE tablespace_name を使用して明示的に削除する必要があります。
- システムテーブルスペースと同様に、一般テーブルスペースに格納されているテーブルの切捨てまたは削除によって、新しい InnoDB データにのみ使用できる空き領域が一般テーブルスペース .ibd data file に内部的に作成されます。file-per-table テーブルスペース用であるため、領域はオペレーティングシステムに解放されません。

InnoDB の例

この例では、InnoDB の一般テーブルスペースを削除する方法を示します。一般的なテーブルスペース ts1 は、単一のテーブルで作成されます。テーブルスペースを削除する前に、テーブルを削除する必要があります。

```
mysql> CREATE TABLESPACE `ts1` ADD DATAFILE 'ts1.ibd' Engine=InnoDB;
mysql> CREATE TABLE t1 (c1 INT PRIMARY KEY) TABLESPACE ts1 Engine=InnoDB;
mysql> DROP TABLE t1;
mysql> DROP TABLESPACE ts1;
```

この例では、undo テーブルスペースの削除を示します。undo テーブルスペースは、削除する前に `empty` 状態である必要があります。詳細は、[セクション15.6.3.4「undo テーブルスペース」](#)を参照してください。

```
mysql> DROP UNDO TABLESPACE undo_003;
```

NDB の例

この例では、最初にテーブルスペースを作成した後に `mydata-1.dat` という名前のデータファイルを持つ NDB テーブルスペース `myts` を削除する方法を示し、`mylg` という名前のログファイルグループが存在することを前提としています ([セクション13.1.16「CREATE LOGFILE GROUP ステートメント」](#)を参照)。

```
mysql> CREATE TABLESPACE myts
-> ADD DATAFILE 'mydata-1.dat'
-> USE LOGFILE GROUP mylg
-> ENGINE=NDB;
```

削除する前に、次に示すように、`ALTER TABLESPACE` を使用してテーブルスペースからすべてのデータファイルを削除する必要があります：

```
mysql> ALTER TABLESPACE myts
-> DROP DATAFILE 'mydata-1.dat'
-> ENGINE=NDB;
```

```
mysql> DROP TABLESPACE myts;
```

13.1.34 DROP TRIGGER ステートメント

```
DROP TRIGGER [IF EXISTS] [schema_name.]trigger_name
```

このステートメントは、トリガーを削除します。スキーマ (データベース) 名はオプションです。スキーマが省略されている場合、このトリガーはデフォルトスキーマから削除されます。`DROP TRIGGER` には、このトリガーに関連付けられたテーブルに対する `TRIGGER` 権限が必要です。

存在しないトリガーに対してエラーが発生しないようにするには、`IF EXISTS` を使用します。`IF EXISTS` を使用している場合は、存在しないトリガーに対して `NOTE` が生成されます。[セクション13.7.7.42「SHOW WARNINGS ステートメント」](#)を参照してください。

テーブルを削除すると、そのテーブルのトリガーも削除されます。

13.1.35 DROP VIEW ステートメント

```
DROP VIEW [IF EXISTS]
view_name [, view_name] ...
[RESTRICT | CASCADE]
```

`DROP VIEW` は、1 つ以上のビューを削除します。各ビューに対する `DROP` 権限が必要です。

引数リストに指定されたビューが存在しない場合、ステートメントは、削除できなかったビューを名前ですすエラーで失敗し、変更は行われません。

注記

MySQL 5.7 以前では、引数リストに指定されたビューが存在しない場合、`DROP VIEW` はエラーを返しますが、存在するリスト内のすべてのビューも削除します。MySQL 8.0 の動作が変更されたため、MySQL 8.0 レプリカでレプリケートすると、MySQL 5.7 レプリケーションソースサーバーで部分的に完了した `DROP VIEW` 操作が失敗します。この失敗のシナリオを回避するには、`DROP VIEW` ステートメントで `IF EXISTS` 構文を使用して、存在

しないビューに対してエラーが発生しないようにします。詳細は、[セクション13.1.1「アトミックデータ定義ステートメントのサポート」](#)を参照してください。

`IF EXISTS` 句は、存在しないビューに対してエラーが発生しないようにします。この句が指定されている場合は、存在しないビューごとに `NOTE` が生成されます。[セクション13.7.7.42「SHOW WARNINGS ステートメント」](#)を参照してください。

`RESTRICT` と `CASCADE` (指定されている場合) は解析されますが、無視されます。

13.1.36 RENAME TABLE ステートメント

```
RENAME TABLE
tbl_name TO new_tbl_name
[, tbl_name2 TO new_tbl_name2] ...
```

`RENAME TABLE` は、1 つ以上のテーブルの名前を変更します。元のテーブルに対する `ALTER` 権限と `DROP` 権限、および新しいテーブルに対する `CREATE` 権限と `INSERT` 権限が必要です。

たとえば、`old_table` というテーブルの名前を `new_table` に変更するには、次のステートメントを使用します:

```
RENAME TABLE old_table TO new_table;
```

このステートメントは、次の `ALTER TABLE` ステートメントと同等です:

```
ALTER TABLE old_table RENAME new_table;
```

`RENAME TABLE` は、`ALTER TABLE` とは異なり、単一のステートメント内で複数のテーブルの名前を変更できます:

```
RENAME TABLE old_table1 TO new_table1,
old_table2 TO new_table2,
old_table3 TO new_table3;
```

名前変更操作は左から右に実行されます。したがって、2 つのテーブル名を入れ替えるには、次の手順を実行します (仲介者名が `tmp_table` のテーブルがまだ存在しないことを前提としています):

```
RENAME TABLE old_table TO tmp_table,
new_table TO old_table,
tmp_table TO new_table;
```

テーブルのメタデータロックは名前順に取得され、場合によっては、複数のトランザクションが同時に実行されるときに操作結果に違いが生じることがあります。[セクション8.11.4「メタデータのロック」](#)を参照してください。

MySQL 8.0.13 では、`WRITE` ロックでロックされているテーブル、または複数テーブルの名前変更操作の前のステップでロックされた `WRITE` テーブルの名前を変更する積である場合、`LOCK TABLES` ステートメントでロックされたテーブルの名前を変更できます。たとえば、次のように指定できます:

```
LOCK TABLE old_table1 WRITE;
RENAME TABLE old_table1 TO new_table1,
new_table1 TO new_table2;
```

次の場合は許可されません。

```
LOCK TABLE old_table1 READ;
RENAME TABLE old_table1 TO new_table1,
new_table1 TO new_table2;
```

MySQL 8.0.13 より前は、`RENAME TABLE` を実行するには、`LOCK TABLES` でロックされたテーブルがないようにする必要があります。

トランザクションテーブルのロック条件が満たされると、名前変更操作はアトミックに実行されます。名前変更の進行中は、他のセッションはどのテーブルにもアクセスできません。

`RENAME TABLE` 中にエラーが発生した場合、ステートメントは失敗し、変更は行われません。

`RENAME TABLE` を使用して、データベース間でテーブルを移動できます:

```
RENAME TABLE current_db.tbl_name TO other_db.tbl_name;
```

この方法を使用して、あるデータベースから別のデータベースにすべてのテーブルを移動すると、実際には、元のデータベースが引き続き存在し、テーブルなしでアルバイトされることを除き、データベースの名前が変更されず (MySQL に単一のステートメントがない操作)。

`RENAME TABLE` と同様に、`ALTER TABLE ... RENAME` を使用してテーブルを別のデータベースに移動することもできます。使用するステートメントに関係なく、名前変更操作によってテーブルが別のファイルシステムにあるデータベースに移動される場合、結果の成功はプラットフォーム固有であり、テーブルファイルの移動に使用される基礎となるオペレーティングシステムコールによって異なります。

テーブルにトリガーがある場合、テーブルの名前を別のデータベースに変更しようとする、「トリガーが間違っただスキーマにあります」 (`ER_TRG_IN_WRONG_SCHEMA`) エラーで失敗します。

暗号化されていないテーブルは暗号化対応データベースに移動でき、その逆も可能です。ただし、`table_encryption_privilege_check` 変数が有効になっている場合、テーブルの暗号化設定がデフォルトのデータベース暗号化と異なると、`TABLE_ENCRYPTION_ADMIN` 権限が必要です。

`TEMPORARY` テーブルの名前を変更する場合、`RENAME TABLE` は機能しません。かわりに `ALTER TABLE` を使用してください。

`RENAME TABLE` はビューに対して機能しますが、ビューの名前を別のデータベースに変更することはできません。

名前を変更したテーブルまたはビュー専用が付与された権限は、新しい名前に移行されません。それらは、手動で変更する必要があります。

`RENAME TABLE tbl_name TO new_tbl_name` は、内部的に生成された外部キー制約名および文字列「tbl_name_ibfk_」で始まるユーザー定義の外部キー制約名を、新しいテーブル名を反映するように変更します。InnoDB は、文字列「tbl_name_ibfk_」で始まる外部キー制約名を内部的に生成された名前と解釈します。

名前が変更されたテーブルを指す外部キー制約名は、競合がないかぎり自動的に更新されます。競合がある場合、ステートメントはエラーで失敗します。名前を変更した制約名がすでに存在する場合は、競合が発生します。このような場合は、外部キーを削除して再作成し、正しく機能させる必要があります。

`RENAME TABLE tbl_name TO new_tbl_name` は、新しいテーブル名を反映するために、文字列「tbl_name_chk_」で始まる内部生成およびユーザー定義の `CHECK` 制約名を変更します。MySQL は、文字列「tbl_name_chk_」で始まる `CHECK` 制約名を内部的に生成された名前と解釈します。例:

```
mysql> SHOW CREATE TABLE t1\G
***** 1. row *****
      Table: t1
Create Table: CREATE TABLE `t1` (
  `i1` int(11) DEFAULT NULL,
  `i2` int(11) DEFAULT NULL,
  CONSTRAINT `t1_chk_1` CHECK ((`i1` > 0)),
  CONSTRAINT `t1_chk_2` CHECK ((`i2` < 0))
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
1 row in set (0.02 sec)

mysql> RENAME TABLE t1 TO t3;
Query OK, 0 rows affected (0.03 sec)

mysql> SHOW CREATE TABLE t3\G
***** 1. row *****
      Table: t3
Create Table: CREATE TABLE `t3` (
  `i1` int(11) DEFAULT NULL,
  `i2` int(11) DEFAULT NULL,
  CONSTRAINT `t3_chk_1` CHECK ((`i1` > 0)),
  CONSTRAINT `t3_chk_2` CHECK ((`i2` < 0))
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
1 row in set (0.01 sec)
```

13.1.37 TRUNCATE TABLE ステートメント

```
TRUNCATE [TABLE] tbl_name
```

TRUNCATE TABLE は、テーブルを完全に空にします。これには **DROP** 権限が必要です。**TRUNCATE TABLE** は論理的に、すべての行を削除する **DELETE** ステートメントや、**DROP TABLE** および **CREATE TABLE** ステートメントのシーケンスに似ています。

高パフォーマンスを実現するために、**TRUNCATE TABLE** はデータを削除する DML メソッドをバイパスします。したがって、**ON DELETE** トリガーは起動せず、親子外部キー関係を持つ **InnoDB** テーブルに対しては実行できず、DML 操作のようにロールバックできません。ただし、アトミック DDL でサポートされているストレージエンジンを使用するテーブルでの **TRUNCATE TABLE** 操作は、その操作中にサーバーが停止すると、完全にコミットまたはロールバックされます。詳細は、[セクション13.1.1「アトミックデータ定義ステートメントのサポート」](#)を参照してください。

TRUNCATE TABLE は **DELETE** に似ているにもかかわらず、DML ステートメントではなく DDL ステートメントとして分類されます。これは、次の点で **DELETE** と異なります:

- 切り捨て操作はテーブルを削除して再作成するため、特に大きなテーブルの場合は、行を1つずつ削除するよりはるかに高速です。
- 切り捨て操作は暗黙的なコミットを発生させるため、ロールバックできません。[セクション13.3.3「暗黙的なコミットを発生させるステートメント」](#)を参照してください。
- セッションがアクティブなテーブルロックを保持している場合は、切り詰め操作を実行できません。
- **InnoDB** テーブルまたは **NDB** テーブルを参照する他のテーブルからの **FOREIGN KEY** 制約がある場合、そのテーブルに対する **TRUNCATE TABLE** は失敗します。同じテーブルのカラム間の外部キー制約が許可されます。
- 切り詰め操作は、削除された行数に対して、意味のある値を返しません。通常の結果は「0 rows affected」ですが、これは「情報がない」として解釈してください。
- テーブル定義が有効であるかぎり、データファイルまたはインデックスファイルが破損した場合でも、**TRUNCATE TABLE** を使用してテーブルを空のテーブルとして再作成できます。
- **AUTO_INCREMENT** 値はすべて、その開始値にリセットされます。これは、通常はシーケンス値を再利用しない **MyISAM** や **InnoDB** にも当てはまります。
- パーティションテーブルで使用した場合、**TRUNCATE TABLE** はパーティション化を保持します。つまり、データファイルおよびインデックスファイルは削除されて再作成されますが、パーティション定義には影響しません。
- **TRUNCATE TABLE** ステートメントは、**ON DELETE** トリガーを起動しません。
- 破損した **InnoDB** テーブルの切捨てがサポートされています。

テーブルに対する **TRUNCATE TABLE** は、**HANDLER OPEN** で開かれたそのテーブルのすべてのハンドラを閉じます。

TRUNCATE TABLE は、バイナリロギングおよびレプリケーション目的のときは、**DROP TABLE** とそれに続く **CREATE TABLE** として、つまり、DML ではなく DDL として扱われます。これは、**InnoDB** またはほかのトランザクションストレージエンジン (そのトランザクション分離レベルがステートメントベースロギングを許可しない (**READ COMMITTED** または **READ UNCOMMITTED**)) を使用するときは、**STATEMENT** または **MIXED** ロギングモード使用時にステートメントがログに記録されず複製されなかった事実によります。(Bug #36763) ただし、前述の方法で **InnoDB** を使用してレプリカに適用されます。

MySQL 5.7 以前では、ラージバッファプールおよび **innodb_adaptive_hash_index** が有効になっているシステムで **TRUNCATE TABLE** 操作を実行すると、テーブル適応ハッシュインデックスエントリの削除時に LRU スキャンが発生したため、システムパフォーマンスが一時的に低下する可能性があります (Bug #68184)。MySQL 8.0 で **TRUNCATE TABLE** を **DROP TABLE** および **CREATE TABLE** に再マッピングすると、LRU スキャンの問題が回避されます。

TRUNCATE TABLE はパフォーマンススキーマのサマリーテーブルで使用できますが、その効果は行の削除ではなく、サマリーカラムを 0 または **NULL** にリセットすることです。[セクション27.12.18「パフォーマンススキーマサマリーテーブル」](#)を参照してください。

file-per-table テーブルスペースに存在する **InnoDB** テーブルを切り捨てると、既存のテーブルスペースが削除され、新しいテーブルスペースが作成されます。MySQL 8.0.21 では、テーブルスペースが以前のバージョンで作成され、不明なディレクトリに存在する場合、**InnoDB** は新しいテーブルスペースをデフォルトの場所に作成し、

次の警告をエラーログに書き込みます: The DATA DIRECTORY の場所は既知のディレクトリにある必要があります。DATA DIRECTORY の場所は無視され、ファイルはデフォルトの datadir location に格納されます。既知のディレクトリは、datadir、innodb_data_home_dir および innodb_directories 変数で定義されているディレクトリです。TRUNCATE TABLE で現在の場所にテーブルスペースを作成するには、TRUNCATE TABLE を実行する前に innodb_directories 設定にディレクトリを追加します。

13.2 データ操作ステートメント

13.2.1 CALL ステートメント

```
CALL sp_name([parameter[,...]])
CALL sp_name()
```

CALL ステートメントは、以前に CREATE PROCEDURE を使用して定義されたストアードプロシージャを呼び出します。

引数を取らないストアードプロシージャは、括弧なしで呼び出すことができます。つまり、CALL p() と CALL p は同等です。

CALL は、OUT または INOUT パラメータとして宣言されたパラメータを使用して、その呼び出し元に値を返すことができます。そのプロシージャから戻るとき、クライアントプログラムは、ルーチン内で実行された最後のステートメントで影響を受けた行数を取得することもできます。SQL レベルでは、ROW_COUNT() 関数を呼び出します。C API からは、mysql_affected_rows() 関数を呼び出します。

プロシージャパラメータに対する未処理条件の影響の詳細は、セクション13.6.7.8「条件の処理と OUT または INOUT パラメータ」を参照してください。

OUT または INOUT パラメータを使用してプロシージャから値を取得するには、ユーザー変数を使用してこのパラメータを渡し、そのプロシージャから戻ったあとに変数の値をチェックします。(そのプロシージャを別のストアードプロシージャまたはストアードファンクション内から呼び出している場合は、IN または INOUT パラメータとしてルーチンパラメータまたはローカルルーチン変数を渡すこともできます。) INOUT パラメータの場合は、プロシージャに渡す前にその値を初期化してください。次のプロシージャには、このプロシージャが現在のサーバーバージョンに設定する OUT パラメータと、このプロシージャがその現在の値から 1 増分する INOUT 値が含まれています。

```
CREATE PROCEDURE p (OUT ver_param VARCHAR(25), INOUT incr_param INT)
BEGIN
# Set value of OUT parameter
SELECT VERSION() INTO ver_param;
# Increment value of INOUT parameter
SET incr_param = incr_param + 1;
END;
```

このプロシージャを呼び出す前に、INOUT パラメータとして渡される変数を初期化します。プロシージャをコールすると、2 つの変数の値が設定または変更されていることがわかります:

```
mysql> SET @increment = 10;
mysql> CALL p(@version, @increment);
mysql> SELECT @version, @increment;
+-----+-----+
| @version      | @increment |
+-----+-----+
| 8.0.3-rc-debug-log |      11 |
+-----+-----+
```

PREPARE および EXECUTE で使用される準備済 CALL ステートメントでは、IN パラメータ、OUT および INOUT パラメータにプレースホルダを使用できます。これらの種類のパラメータは、次のように使用できます。

```
mysql> SET @increment = 10;
mysql> PREPARE s FROM 'CALL p(?, ?)';
mysql> EXECUTE s USING @version, @increment;
mysql> SELECT @version, @increment;
+-----+-----+
| @version      | @increment |
+-----+-----+
| 8.0.3-rc-debug-log |      11 |
+-----+-----+
```


CALL SQL ステートメントを使用して、結果セットを生成するストアードプロシージャを実行する C プログラムを記述するには、`CLIENT_MULTI_RESULTS` フラグが有効になっている必要があります。これは、各 `CALL` によって、プロシージャ内で実行されるステートメントによって返される可能性のある結果セットに加えて、呼び出しステータスを示すための結果が返されるためです。`CLIENT_MULTI_RESULTS` は、`CALL` が、準備済みステートメントを含むストアードプロシージャを実行するために使用される場合にも有効になっている必要があります。これらのステートメントが結果セットを生成するかどうかは、このようなプロシージャがロードされたときに判断できないため、結果セットを生成すると想定する必要があります。

`CLIENT_MULTI_RESULTS` は、`mysql_real_connect()` を呼び出すときに、`CLIENT_MULTI_RESULTS` フラグ自体を渡すことによって明示的に、または `CLIENT_MULTI_STATEMENTS` を渡すことによって暗黙的に有効にする (これによって `CLIENT_MULTI_RESULTS` も有効になります) ことができます。`CLIENT_MULTI_RESULTS` はデフォルトで有効になっています。

`mysql_query()` または `mysql_real_query()` を使用して実行された `CALL` ステートメントの結果を処理するには、それ以上結果が存在するかどうかを判定するために `mysql_next_result()` を呼び出すループを使用してください。例については、[Multiple Statement Execution Support](#) を参照してください。

C プログラムでは、プリペアドステートメントインタフェースを使用して `CALL` ステートメントを実行し、`OUT` および `INOUT` パラメータにアクセスできます。これは、それ以上結果が存在するかどうかを判定するために `mysql_stmt_next_result()` を呼び出すループを使用して `CALL` ステートメントの結果を処理することにより行われます。例については、[Prepared CALL Statement Support](#) を参照してください。MySQL インタフェースを備える言語は、準備済み `CALL` ステートメントを使用して、`OUT` および `INOUT` プロシージャパラメータを直接取得できません。

ストアードプログラムによって参照されるオブジェクトに対するメタデータの変更が検出され、プログラムが次に実行されるときに、影響を受けるステートメントが自動的に再解析されます。詳細については、[セクション 8.10.3 「プリペアドステートメントおよびストアードプログラムのキャッシュ」](#) を参照してください。

13.2.2 DELETE ステートメント

`DELETE` は、テーブルの行を削除する DML ステートメントです。

`DELETE` ステートメントは、`WITH` 句で始まり、`DELETE` 内でアクセス可能な共通テーブル式を定義できます。[セクション 13.2.15 「WITH \(共通テーブル式\)」](#) を参照してください。

単一テーブル構文

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name [[AS] tbl_alias]
  [PARTITION (partition_name [, partition_name] ...)]
  [WHERE where_condition]
  [ORDER BY ...]
  [LIMIT row_count]
```

`DELETE` ステートメントは、`tbl_name` の行を削除し、削除された行数を返します。削除された行数をチェックするには、[セクション 12.16 「情報関数」](#) で説明されている `ROW_COUNT()` 関数を呼び出します。

メインの句

オプションの `WHERE` 句内の条件は、どの行を削除するかを識別します。`WHERE` 句がない場合は、すべての行が削除されます。

`where_condition` は、削除される各行に対して `true` に評価される式です。これは、[セクション 13.2.10 「SELECT ステートメント」](#) で説明されているように指定されます。

`ORDER BY` 句が指定されている場合は、指定されている順序で行が削除されます。`LIMIT` 句は、削除できる行数に制限を設定します。これらの句は単一テーブルの削除に適用されますが、複数テーブルの削除には適用されません。

複数テーブル構文

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
  tbl_name[*] [, tbl_name[*]] ...
  FROM table_references
  [WHERE where_condition]
```

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
FROM tbl_name[*] [, tbl_name[*]] ...
USING table_references
[WHERE where_condition]
```

権限

テーブルから行を削除するには、そのテーブルに対する **DELETE** 権限が必要です。 **WHERE** 句で指定されているカラムなどの、読み取られるだけのカラムに対しては、 **SELECT** 権限のみが必要です。

パフォーマンス

削除された行数を知る必要がない場合、テーブルを空にするには、 **WHERE** 句のない **DELETE** ステートメントより **TRUNCATE TABLE** ステートメントの方が高速です。 **DELETE** とは異なり、 **TRUNCATE TABLE** はトランザクション内で、またはそのテーブルがロックされている場合は使用できません。 [セクション13.1.37「TRUNCATE TABLE ステートメント」](#) および [セクション13.3.6「LOCK TABLES および UNLOCK TABLES ステートメント」](#) を参照してください。

削除操作の速度はまた、 [セクション8.2.5.3「DELETE ステートメントの最適化」](#) で説明されている要因によって影響を受ける可能性もあります。

特定の **DELETE** ステートメントに時間がかかりすぎないようにするために、 **DELETE** の MySQL 固有の **LIMIT row_count** 句は、削除される行の最大数を指定します。削除する行数がこの制限を超えている場合は、影響を受ける行数が **LIMIT** 値を下回るまで **DELETE** ステートメントを繰り返します。

サブクエリー

テーブルから削除して、サブクエリーの同じテーブルから選択することはできません。

パーティションテーブルのサポート

DELETE では、 **PARTITION** オプションを使用した明示的なパーティション選択がサポートされています。このオプションは、削除する行を選択するパーティションまたはサブパーティション (あるいはその両方) のカンマ区切り名のリストを取得します。このリストに含まれていないパーティションは無視されます。 **p0** という名前のパーティションを含むパーティション化されたテーブル **t** がある場合、ステートメント **DELETE FROM t PARTITION (p0)** の実行には、このテーブルに対して **ALTER TABLE t TRUNCATE PARTITION (p0)** を実行するのと同じ効果があります。どちらの場合も、パーティション **p0** 内のすべての行が削除されます。

PARTITION は、 **WHERE** 条件とともに使用できます。その場合、この条件は、リストされているパーティション内の行に対してのみテストされます。たとえば、 **DELETE FROM t PARTITION (p0) WHERE c < 5** は、条件 **c < 5** が true であるパーティション **p0** の行のみを削除します。ほかのパーティション内の行はチェックされないため、 **DELETE** によって影響を受けません。

PARTITION オプションはまた、複数テーブルの **DELETE** ステートメントでも使用できます。このようなオプションを、 **FROM** オプションで指定されているテーブルごとに最大 1 つ使用できます。

詳細および例については、 [セクション24.5「パーティション選択」](#) を参照してください。

自動インクリメントカラム

AUTO_INCREMENT カラムに最大値を含む行を削除した場合、その値は、 **MyISAM** または **InnoDB** テーブルには再利用されません。 **autocommit** モードで **DELETE FROM tbl_name (WHERE 句はなし)** を使用してテーブル内のすべての行を削除した場合、そのシーケンスは、 **InnoDB** と **MyISAM** を除くすべてのストレージエンジンに対して開始されます。 [セクション15.6.1.6「InnoDB での AUTO_INCREMENT 処理」](#) で説明されているように、 **InnoDB** テーブルに対しては、この動作の例外がいくつかあります。

MyISAM テーブルの場合は、マルチカラムキー内の **AUTO_INCREMENT** セカンダリカラムを指定できます。この場合は、シーケンスの先頭から削除された値の再利用が **MyISAM** テーブルに対しても実行されます。 [セクション3.6.9「AUTO_INCREMENT の使用」](#) を参照してください。

修飾子

DELETE ステートメントは、次の修飾子をサポートします。

- **LOW_PRIORITY** 修飾子を指定すると、他のクライアントがテーブルから読み取ることがなくなるまで、サーバーは **DELETE** の実行を遅延します。これは、テーブルレベルロックのみを使用するストレージエンジン (**MyISAM**、**MEMORY**、および **MERGE**) にも影響を与えます。
- **MyISAM** テーブルの場合、**QUICK** 修飾子を使用すると、ストレージエンジンは削除中にインデックスリーフをマージしないため、一部の種類の削除操作が高速になる可能性があります。
- **IGNORE** 修飾子を使用すると、MySQL は行の削除処理中に無視できるエラーを無視します。(解析の段階で検出されたエラーは、通常の方法で処理されます。) **IGNORE** の使用のために無視されたエラーは、警告として返されます。詳細は、**IGNORE がステートメントの実行に与える影響**を参照してください。

削除の順序

DELETE ステートメントに **ORDER BY** 句が含まれている場合は、この句で指定されている順序で行が削除されます。これは、主に **LIMIT** と組み合わせて使用した場合に有効です。たとえば、次のステートメントは **WHERE** 句に一致する行を見つけ、それらを **timestamp_column** でソートしたあと、最初の(もっとも古い)行を削除します。

```
DELETE FROM somelog WHERE user = 'jcole'  
ORDER BY timestamp_column LIMIT 1;
```

ORDER BY はまた、参照整合性の違反を回避するために必要な順序で行を削除する場合も役立ちます。

InnoDB テーブル

大きなテーブルから多数の行を削除する場合は、**InnoDB** テーブルに対するロックテーブルのサイズを超える可能性があります。この問題を回避するために、または単にテーブルがロックされたままになる時間を最小限に抑えるために、**DELETE** をまったく使用しない次の方法が有効な場合があります。

1. 削除されない行を選択して、元のテーブルと同じ構造を持つ空のテーブルに格納します。

```
INSERT INTO t_copy SELECT * FROM t WHERE ... ;
```

2. **RENAME TABLE** を使用して元のテーブルを原子的に移動したあと、コピーの名前を元の名前に変更します。

```
RENAME TABLE t TO t_old, t_copy TO t;
```

3. 元のテーブルを削除します。

```
DROP TABLE t_old;
```

RENAME TABLE が実行されている間、関連するテーブルにはほかのどのセッションからもアクセスできないため、名前変更の操作は並列性の問題に制約されません。 [セクション13.1.36「RENAME TABLE ステートメント」](#)を参照してください。

MyISAM テーブル

MyISAM テーブルでは、削除された行はリンクリスト内に保持され、以降の **INSERT** 操作は古い行の位置を再利用します。未使用領域を再利用し、ファイルサイズを減らすには、**OPTIMIZE TABLE** ステートメントまたは **myisamchk** ユーティリティーを使用してテーブルを再編成します。**OPTIMIZE TABLE** の方が使い方は簡単ですが、**myisamchk** の方が高速です。 [セクション13.7.3.4「OPTIMIZE TABLE ステートメント」](#) および [セクション4.6.4「myisamchk — MyISAM テーブルメンテナンスユーティリティー」](#)を参照してください。

QUICK 修飾子は、削除操作でインデックスリーフがマージされるかどうかに影響を与えます。**DELETE QUICK** は、削除された行のインデックス値が、あとで挿入された行の同様のインデックス値に置き換えられるアプリケーションで、特に役立ちます。この場合、削除された値によって残された穴は再利用されます。

DELETE QUICK は、削除された値によって、新しい挿入が再度発生するインデックス値の範囲全体にわたって空きのあるインデックスブロックが残される場合には役立ちません。この場合は、**QUICK** を使用すると、再利用されないままのインデックスで領域が浪費される可能性があります。このようなシナリオの例を次に示します。

1. インデックス付き **AUTO_INCREMENT** カラムを含むテーブルを作成します。
2. このテーブルに多数の行を挿入します。各挿入によって、インデックスの先頭に追加されるインデックス値が生成されます。

3. **DELETE QUICK** を使用して、カラムの範囲の最後にある行のブロックを削除します。

このシナリオでは、削除されたインデックス値に関連付けられたインデックスブロックに空きができますが、**QUICK** が使用されているため、ほかのインデックスブロックにはマージされません。新しい挿入が発生したとき、新しい行には削除された範囲内のインデックス値が含まれていないため、これらのインデックスブロックは空きがあるままになります。さらに、削除された一部のインデックス値が偶然に空きのあるブロック内か、またはその隣のインデックスブロックに含まれていないかぎり、あとで **QUICK** なしで **DELETE** を使用した場合でも空きがあるままになります。これらの状況で未使用のインデックス領域を再利用するには、**OPTIMIZE TABLE** を使用します。

テーブルから多数の行を削除しようとしている場合は、**DELETE QUICK** に続けて **OPTIMIZE TABLE** を使用した方が高速になることがあります。これにより、インデックスブロックの多数のマージ操作が実行されるのではなく、インデックスが再構築されます。

複数テーブルの削除

WHERE 句内の条件に応じて 1 つ以上のテーブルから行を削除するには、**DELETE** ステートメントで複数のテーブルを指定できます。複数テーブルの **DELETE** では、**ORDER BY** または **LIMIT** を使用できません。[セクション 13.2.10.2 「JOIN 句」](#) で説明されているように、**table_references** 句は、結合に含まれるテーブルをリストします。

最初の複数テーブル構文では、**FROM** 句の前にリストされているテーブルの一致する行のみが削除されます。2 番目の複数テーブル構文では、**USING** 句の前にある **FROM** 句にリストされているテーブルの一致する行のみが削除されます。その効果は、多数のテーブルの行を同時に削除し、さらに検索にのみ使用される追加のテーブルを指定できることです。

```
DELETE t1, t2 FROM t1 INNER JOIN t2 INNER JOIN t3
WHERE t1.id=t2.id AND t2.id=t3.id;
```

または:

```
DELETE FROM t1, t2 USING t1 INNER JOIN t2 INNER JOIN t3
WHERE t1.id=t2.id AND t2.id=t3.id;
```

これらのステートメントは、削除する行を検索するときに 3 つのすべてのテーブルを使用しますが、テーブル **t1** と **t2** の一致する行のみを削除します。

前の例では **INNER JOIN** を使用していますが、複数テーブルの **DELETE** ステートメントは、**SELECT** ステートメント内で許可されているほかの型の結合 (**LEFT JOIN** など) を使用できます。たとえば、**t1** 内に存在する行で **t2** 内に一致するものがない行を削除するには、**LEFT JOIN** を使用します。

```
DELETE t1 FROM t1 LEFT JOIN t2 ON t1.id=t2.id WHERE t2.id IS NULL;
```

この構文では、**Access** との互換性のために、各 **tbl_name** のあとに ***** が許可されます。

外部キー制約が存在する **InnoDB** テーブルを含む、複数テーブルの **DELETE** ステートメントを使用した場合は、MySQL オプティマイザが、それらの親子関係の順序とは異なる順序でテーブルを処理する可能性があります。この場合、このステートメントは失敗し、ロールバックされます。代わりに、1 つのテーブルから削除したあと、**InnoDB** が提供する **ON DELETE** 機能を使用して、ほかのテーブルがそれに応じて変更されるようにしてください。

注記

テーブルのエイリアスを宣言した場合は、テーブルを参照するときはそのエイリアスを使用する必要があります。

```
DELETE t1 FROM test AS t1, test2 WHERE ...
```

複数テーブルの **DELETE** 内のテーブルエイリアスは、そのステートメントの **table_references** 部分でのみ宣言するようにしてください。それ以外の場所では、エイリアス参照が許可されますが、エイリアス宣言は許可されません。

正しい:

```
DELETE a1, a2 FROM t1 AS a1 INNER JOIN t2 AS a2
WHERE a1.id=a2.id;
```

```
DELETE FROM a1, a2 USING t1 AS a1 INNER JOIN t2 AS a2
WHERE a1.id=a2.id;
```

正しくない:

```
DELETE t1 AS a1, t2 AS a2 FROM t1 INNER JOIN t2
WHERE a1.id=a2.id;
```

```
DELETE FROM t1 AS a1, t2 AS a2 USING t1 INNER JOIN t2
WHERE a1.id=a2.id;
```

テーブルのエイリアスは、MySQL 8.0.16 以降の単一テーブルの **DELETE** ステートメントでもサポートされています。(Bug #89410、Bug #27455809)

13.2.3 DO ステートメント

```
DO expr [, expr] ...
```

DO は式を実行しますが、結果は何も返しません。ほとんどの点で、**DO** は **SELECT expr, ...** の短縮形ですが、その結果に関心がない場合は少し高速であるという利点があります。

DO は主に、副作用がある関数 (**RELEASE_LOCK()** など) で役立ちます。

例: この **SELECT** ステートメントは一時停止しますが、結果セットの生成も行います。

```
mysql> SELECT SLEEP(5);
+-----+
| SLEEP(5) |
+-----+
| 0 |
+-----+
1 row in set (5.02 sec)
```

それに対して、**DO** は、結果セットを生成することなく一時停止します。

```
mysql> DO SLEEP(5);
Query OK, 0 rows affected (4.99 sec)
```

これは、たとえば、結果セットを生成するステートメントを禁止しているストアドファンクションまたはトリガーで役立つ場合があります。

DO は式を実行するだけです。**SELECT** を使用できるすべての場合に使用できるわけではありません。たとえば、**DO id FROM t1** は、テーブルを参照するため無効です。

13.2.4 HANDLER ステートメント

```
HANDLER tbl_name OPEN [ [AS] alias]

HANDLER tbl_name READ index_name { = | <= | >= | < | > } (value1,value2,...)
[ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ index_name { FIRST | NEXT | PREV | LAST }
[ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ { FIRST | NEXT }
[ WHERE where_condition ] [LIMIT ... ]

HANDLER tbl_name CLOSE
```

HANDLER ステートメントは、テーブルストレージエンジンインタフェースへの直接アクセスを提供します。これは、**InnoDB** および **MyISAM** テーブルに使用できます。

HANDLER ... OPEN ステートメントはテーブルを開き、それを以降の **HANDLER ... READ** ステートメントを使用してアクセス可能にします。このテーブルオブジェクトはほかのセッションによって共有されておらず、このセッションが **HANDLER ... CLOSE** を呼び出すか、またはこのセッションが終了するまでクローズされません。

エイリアスを使用してテーブルを開いた場合は、その開かれたテーブルへのほかの **HANDLER** ステートメントによるそれ以降の参照は、テーブル名ではなくエイリアスを使用する必要があります。エイリアスを使用せずに、データベース名で修飾されたテーブル名を使用してテーブルを開く場合、以降の参照では修飾されていないテーブル名を使

用する必要があります。たとえば、`mydb.mytable` を使用して開いたテーブルの場合、さらに参照するには `mytable` を使用する必要があります。

最初の `HANDLER ... READ` 構文は、指定されたインデックスが特定の値を満たし、かつ `WHERE` 条件が満たされている行をフェッチします。マルチカラムインデックスがある場合は、インデックスカラム値をカンマ区切りリストとして指定します。インデックス内のすべてのカラムの値を指定するか、またはインデックスカラムの左端のプリフィクスの値を指定します。インデックス `my_idx` に、`col_a`、`col_b`、および `col_c` という名前の 3 つのカラムがその順序で含まれているとします。`HANDLER` ステートメントは、そのインデックス内の 3 つのすべてのカラム、または左端のプリフィクス内のカラムの値を指定できます。例:

```
HANDLER ... READ my_idx = (col_a_val,col_b_val,col_c_val) ...
HANDLER ... READ my_idx = (col_a_val,col_b_val) ...
HANDLER ... READ my_idx = (col_a_val) ...
```

`HANDLER` インタフェースを使用してテーブルの `PRIMARY KEY` を参照するには、引用符で囲まれた識別子 `'PRIMARY'` を使用します。

```
HANDLER tbl_name READ 'PRIMARY' ...
```

2 番目の `HANDLER ... READ` 構文は、`WHERE` 条件に一致するインデックス順序でテーブルの行をフェッチします。

3 番目の `HANDLER ... READ` 構文は、`WHERE` 条件に一致する自然な行順序でテーブルの行をフェッチします。これは、フルテーブルスキャンが望ましい場合は、`HANDLER tbl_name READ index_name` より高速です。自然な行順序とは、行が `MyISAM` テーブルデータファイル内に格納されている順序のことです。このステートメントは `InnoDB` テーブルに対しても機能しますが、個別のデータファイルが存在しないため、このような概念はありません。

`LIMIT` 句を使用しない場合は、すべての形式の `HANDLER ... READ` が単一行 (使用可能な場合) をフェッチします。特定の行数を返すには、`LIMIT` 句を含めます。その構文は、`SELECT` ステートメントの場合と同じです。[セクション 13.2.10 「SELECT ステートメント」](#) を参照してください。

`HANDLER ... CLOSE` は、`HANDLER ... OPEN` でオープンされたテーブルをクローズします。

通常の `SELECT` ステートメントの代わりに `HANDLER` インタフェースを使用する理由として、次のいくつかがあります。

- `HANDLER` は `SELECT` より高速です。
 - `HANDLER ... OPEN` に対して、指定されたストレージエンジンハンドラオブジェクトが割り当てられます。このオブジェクトは、そのテーブルに対する以降の `HANDLER` ステートメントに再利用されます。ステートメントごとに再初期化する必要はありません。
 - 関連する解析が少なくなります。
 - オプティマイザまたはクエリーチェックのオーバーヘッドがありません。
 - ハンドラインタフェースは (たとえば、[ダーティー読み取り](#)が許可されるような) データの整合性のある外観を提供する必要がないため、ストレージエンジンは、`SELECT` が通常は許可しない最適化を使用できます。
- `HANDLER` によって、`ISAM` に似た低レベルのインタフェースを使用する MySQL アプリケーションへの移植が容易になります。(キー値格納パラダイムを使用するアプリケーションを適応させるための代替手段については、[セクション 15.20 「InnoDB memcached プラグイン」](#) を参照してください。)
- `HANDLER` を使用すると、`SELECT` では実現が困難な (または、不可能でさえある) 方法でデータベースをたどることができます。`HANDLER` インタフェースは、データベースに対話型ユーザーインタフェースを提供するアプリケーションの操作時にデータを調べるためのより自然な方法です。

`HANDLER` は、やや低レベルのステートメントです。たとえば、一貫性が提供されません。つまり、`HANDLER ... OPEN` はテーブルのスナップショットを作成せず、テーブルのロックも行いません。これは、`HANDLER ... OPEN` ステートメントが発行されたあと、テーブルデータを (現在のセッションまたはその他のセッションで) 変更することができ、これらの変更が `HANDLER ... NEXT` または `HANDLER ... PREV` スキャンに部分的にしか表示されない可能性があることを示します。

開かれたハンドラを閉じ、再度開くようにマークすることができます。その場合、このハンドラはテーブル内の位置を失います。これは、次の両方の状況が当てはまる場合に発生します。

- このハンドラのテーブルに対して、いずれかのセッションが `FLUSH TABLES` または DDL ステートメントを実行している。
- このハンドラを開いているセッションが、テーブルを使用する `HANDLER` 以外のステートメントを実行している。

テーブルに対する `TRUNCATE TABLE` は、`HANDLER OPEN` で開かれたそのテーブルのすべてのハンドラを閉じます。

`FLUSH TABLES tbl_name WITH READ LOCK` でフラッシュされたテーブルが `HANDLER` で開かれた場合、そのハンドラは暗黙的にフラッシュされ、その位置を失います。

13.2.5 IMPORT TABLE ステートメント

```
IMPORT TABLE FROM sdi_file [, sdi_file] ...
```

`IMPORT TABLE` ステートメントは、`.sdi` (シリアライズディクショナリ情報) メタデータファイルに含まれる情報に基づいて `MyISAM` テーブルをインポートします。 `IMPORT TABLE` には、`.sdi` およびテーブルコンテンツファイルを読み取るための `FILE` 権限と、作成するテーブルに対する `CREATE` 権限が必要です。

`mysqldump` を使用してあるサーバーからテーブルをエクスポートし、SQL ステートメントのファイルを書き込み、`mysql` を使用して別のサーバーにインポートしてダンプファイルを処理できます。 `IMPORT TABLE` は、「raw」テーブルファイルを使用して、より高速な代替方法を提供します。

インポートする前に、テーブルの内容を提供するファイルをインポートサーバーの適切なスキーマディレクトリに配置し、`.sdi` ファイルをサーバーからアクセス可能なディレクトリに配置する必要があります。たとえば、`.sdi` ファイルは、`secure_file_priv` システム変数で指定されたディレクトリ、または (`secure_file_priv` が空の場合は) サーバーデータディレクトリの下のディレクトリに配置できます。

次の例では、`employees` および `managers` という名前の `MyISAM` テーブルをあるサーバーの `hr` スキーマからエクスポートし、別のサーバーの `hr` スキーマにインポートする方法について説明します。この例では、次の前提を使用しています (独自のシステムで同様の操作を実行するには、必要に応じてパス名を変更します):

- エクスポートサーバーの場合、`export_basedir` はベースディレクトリを表し、そのデータディレクトリは `export_basedir/data` です。
- インポートサーバーの場合、`import_basedir` はベースディレクトリを表し、そのデータディレクトリは `import_basedir/data` です。
- テーブルファイルはエクスポートサーバーから `/tmp/export` ディレクトリにエクスポートされ、このディレクトリはセキユアです (他のユーザーはアクセスできません)。
- インポートサーバーは、`secure_file_priv` システム変数で指定されたディレクトリとして `/tmp/mysql-files` を使用します。

エクスポートサーバーからテーブルをエクスポートするには、この手順を使用します:

1. エクスポート中に変更できないように、次のステートメントを実行してテーブルをロックし、一貫性のあるスナップショットを確認します:

```
mysql> FLUSH TABLES hr.employees, hr.managers WITH READ LOCK;
```

ロックが有効な間は、テーブルは引き続き使用できますが、読取りアクセスにのみ使用できます。

2. ファイルシステムレベルで、`.sdi` およびテーブルコンテンツファイルを `hr` スキーマディレクトリからセキユアエクスポートディレクトリにコピーします:
 - `.sdi` ファイルは `hr` スキーマディレクトリにありますが、テーブル名とまったく同じベース名を持っていない可能性があります。たとえば、`employees` テーブルおよび `managers` テーブルの `.sdi` ファイルには、`employees_125.sdi` および `managers_238.sdi` という名前が付けられます。
 - `MyISAM` テーブルの場合、コンテンツファイルはその `.MYD` データファイルおよび `.MYI` インデックスファイルです。

これらのファイル名を指定すると、コピーコマンドは次のようになります:

```
shell> cd export_basedir/data/hr
shell> cp employees_125.sdi /tmp/export
shell> cp managers_238.sdi /tmp/export
shell> cp employees.{MYD,MYI} /tmp/export
shell> cp managers.{MYD,MYI} /tmp/export
```

3. テーブルのロックを解除します:

```
mysql> UNLOCK TABLES;
```

インポートサーバーにテーブルをインポートするには、次の手順を使用します:

1. インポートスキーマが存在する必要があります。必要に応じて、次のステートメントを実行して作成します:

```
mysql> CREATE SCHEMA hr;
```

2. ファイルシステムレベルで、.sdi ファイルをインポートサーバーの `secure_file_priv` ディレクトリ/`tmp/mysql-files` にコピーします。また、テーブルコンテンツファイルを `hr` スキーマディレクトリにコピーします:

```
shell> cd /tmp/export
shell> cp employees_125.sdi /tmp/mysql-files
shell> cp managers_238.sdi /tmp/mysql-files
shell> cp employees.{MYD,MYI} import_basedir/data/hr
shell> cp managers.{MYD,MYI} import_basedir/data/hr
```

3. .sdi ファイルを指定する `IMPORT TABLE` ステートメントを実行して、テーブルをインポートします:

```
mysql> IMPORT TABLE FROM
'/tmp/mysql-files/employees.sdi',
'/tmp/mysql-files/managers.sdi';
```

.sdi ファイルは、その変数が空の場合、`secure_file_priv` システム変数で指定されたインポートサーバーディレクトリに配置する必要はありません。インポートされたテーブルのスキーマディレクトリを含め、サーバーからアクセス可能な任意のディレクトリに配置できます。ただし、.sdi ファイルがそのディレクトリに配置されている場合は、リライトされる可能性があります。インポート操作では、テーブルに対して新しい .sdi ファイルが作成され、操作で新しいファイルと同じファイル名が使用されている場合は古い .sdi ファイルが上書きされます。

各 `sdi_file` 値は、テーブルの .sdi ファイルを指定する文字列リテラルであるが、.sdi ファイルと一致するパターンである必要があります。文字列がパターンの場合は、先頭のディレクトリパスおよび .sdi ファイル名の接尾辞を文字どおりに指定する必要があります。パターン文字は、ファイル名のベース名部分でのみ使用できます:

- `?` は任意の 1 文字に一致します
- `*` は、文字を含まない任意の文字シーケンスに一致

パターンを使用すると、前の `IMPORT TABLE` ステートメントが次のように記述されている可能性があります (`/tmp/mysql-files` ディレクトリにパターンに一致する他の .sdi ファイルが含まれていないことを前提としています):

```
IMPORT TABLE FROM '/tmp/mysql-files/*.sdi';
```

.sdi ファイルパス名の場所を解釈するために、サーバーは `LOAD DATA` のサーバー側ルールと同じルール (`LOCAL` 以外のルール) を `IMPORT TABLE` に使用します。セクション13.2.7「`LOAD DATA` ステートメント」を参照してください。相対パス名の解釈に使用されるルールに特に注意してください。

.sdi またはテーブルファイルが見つからない場合、`IMPORT TABLE` は失敗します。テーブルをインポートすると、サーバーはテーブルを開こうとし、検出された問題を警告として報告します。修復を試行して報告された問題を修正するには、`REPAIR TABLE` を使用します。

`IMPORT TABLE` はバイナリログに書き込まれません。

制約と制限

`IMPORT TABLE` は、`TEMPORARY` 以外の `MyISAM` テーブルにのみ適用されます。トランザクションストレージエンジンで作成されたテーブル、`CREATE TEMPORARY TABLE` で作成されたテーブルまたはビューには適用されません。

インポート操作で使用される `.sdi` ファイルは、インポートサーバーと同じデータディクショナリバージョンおよび `sdi` バージョンのサーバーで生成する必要があります。生成元サーバーのバージョン情報は、`.sdi` ファイルにあります:

```
{
  "mysqlId_version_id":80019,
  "dd_version":80017,
  "sdi_version":80016,
  ...
}
```

インポートサーバーのデータディクショナリおよび `sdi` バージョンを確認するには、インポートサーバーで最近作成されたテーブルの `.sdi` ファイルを確認します。

テーブルデータおよびインデックスファイルは、エクスポートサーバーで定義されているテーブルで `DATA DIRECTORY` または `INDEX DIRECTORY` テーブルオプションが使用されていないかぎり、インポート操作の前にインポートサーバーのスキーマディレクトリに配置する必要があります。その場合は、`IMPORT TABLE` ステートメントを実行する前に、次のいずれかの代替方法を使用してインポートプロシージャを変更します:

- データファイルとインデックスファイルをエクスポートサーバーホストと同じディレクトリに配置し、インポートサーバースキーマディレクトリにそれらのファイルへのシンボリックリンクを作成します。
- データファイルおよびインデックスファイルをエクスポートサーバーホスト上のもとは異なるインポートサーバーホストディレクトリに配置し、それらのファイルへのシンボリックリンクをインポートサーバースキーマディレクトリに作成します。また、異なるファイルの場所を反映するように `.sdi` ファイルを変更します。
- データおよびインデックスファイルをインポートサーバーホストのスキーマディレクトリに配置し、`.sdi` ファイルを変更してデータおよびインデックスディレクトリのテーブルオプションを削除します。

`.sdi` ファイルに格納されている照合 ID は、エクスポートサーバーとインポートサーバーで同じ照合を参照する必要があります。

テーブルのトリガー情報はテーブル `.sdi` ファイルにシリアライズされないため、インポート操作によってトリガーはリストアされません。

`.sdi` ファイルの一部の編集は `IMPORT TABLE` ステートメントの実行前に許可されますが、その他の編集は問題があるか、インポート操作が失敗する可能性があります:

- データディレクトリおよびインデックスディレクトリテーブルオプションの変更は、データファイルとインデックスファイルの場所がエクスポートサーバーとインポートサーバーで異なる場合に必要です。
- エクスポートサーバーとは異なるインポートサーバーのスキーマにテーブルをインポートするには、スキーマ名を変更する必要があります。
- エクスポートサーバーとインポートサーバーでのファイルシステムの大/小文字の区別セマンティクスの違いや、`lower_case_table_names` 設定の違いに対応するために、スキーマ名とテーブル名の変更が必要になる場合があります。`.sdi` ファイルのテーブル名を変更する場合は、テーブルファイルの名前も変更する必要があります。
- 場合によっては、カラム定義の変更が許可されます。データ型を変更すると、問題が発生する可能性があります。

13.2.6 INSERT ステートメント

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name
[PARTITION (partition_name [, partition_name] ...)]
[[col_name [, col_name] ...]]
{ [VALUES | VALUE] (value_list) [, (value_list)] ...
|
VALUES row_constructor_list
}
[AS row_alias[(col_alias [, col_alias] ...)]]
[ON DUPLICATE KEY UPDATE assignment_list]

INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name
[PARTITION (partition_name [, partition_name] ...)]
[AS row_alias[(col_alias [, col_alias] ...)]]
SET assignment_list
```

```
[ON DUPLICATE KEY UPDATE assignment_list]

INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name
[PARTITION (partition_name [, partition_name] ...)]
[(col_name [, col_name] ...)]
[AS row_alias[(col_alias [, col_alias] ...)]]
{SELECT ... | TABLE table_name}
[ON DUPLICATE KEY UPDATE assignment_list]

value:
{expr | DEFAULT}

value_list:
value [, value] ...

row_constructor_list:
ROW(value_list)[, ROW(value_list)][, ...]

assignment:
col_name = [row_alias.]value

assignment_list:
assignment [, assignment] ...
```

INSERT は、既存のテーブルに新しい行を挿入します。INSERT ... VALUES、INSERT ... VALUES ROW() および INSERT ... SET 形式のステートメントは、明示的に指定された値に基づいて行を挿入します。INSERT ... SELECT 形式は、別の 1 つまたは複数のテーブルから選択された行を挿入します。MySQL 8.0.19 以降で INSERT ... TABLE を使用して、単一のテーブルから行を挿入することもできます。INSERT で ON DUPLICATE KEY UPDATE 句を使用すると、行が挿入されて UNIQUE インデックスまたは PRIMARY KEY で値が重複する場合に、既存の行を更新できます。MySQL 8.0.19 以降では、1 つ以上のオプションのカラムアライスを持つ行エイリアスを ON DUPLICATE KEY UPDATE で使用して、挿入する行を参照できます。

INSERT ... SELECT および INSERT ... ON DUPLICATE KEY UPDATE の詳細は、[セクション13.2.6.1「INSERT ... SELECT ステートメント」](#) および [セクション13.2.6.2「INSERT ... ON DUPLICATE KEY UPDATE ステートメント」](#) を参照してください。

MySQL 8.0 では、DELAYED キーワードは受け入れられますが、サーバーでは無視されます。この理由については、[セクション13.2.6.3「INSERT DELAYED ステートメント」](#) を参照してください。

テーブルに挿入するには、そのテーブルに対する INSERT 権限が必要です。ON DUPLICATE KEY UPDATE 句が使用されていて、重複キーのために代わりに UPDATE が実行される場合、このステートメントには、更新されるカラムに対する UPDATE 権限が必要です。読み取られるが、変更されないカラムの場合は、SELECT 権限のみが必要です (ON DUPLICATE KEY UPDATE 句にある col_name=expr 割り当ての右側でのみ参照されるカラムの場合など)。

パーティションテーブルに挿入する場合、新しい行を受け入れるパーティションおよびサブパーティションを制御できます。PARTITION オプションは、テーブルのパーティションまたはサブパーティション (あるいはその両方) のカラム区切りの名前リストを取ります。特定の INSERT ステートメントによって挿入される行がリストされているいずれかのパーティションに一致しない場合、INSERT ステートメントは Found a row not matching the given partition set エラーで失敗します。詳細および例については、[セクション24.5「パーティション選択」](#) を参照してください。

tbl_name は、行が挿入されるテーブルです。ステートメントが値を提供するカラムを次のように指定します:

- テーブル名の後にカンマ区切りのカラム名のカッコ付きリストを指定します。この場合、各名前付きカラムの値は、VALUES リスト、VALUES ROW() リストまたは SELECT ステートメントで指定する必要があります。INSERT TABLE フォームの場合、ソーステーブルのカラム数は挿入されるカラム数と一致する必要があります。
- INSERT ... VALUES または INSERT ... SELECT のカラム名のリストを指定しない場合は、テーブル内のすべてのカラムの値を VALUES リスト、SELECT ステートメントまたは TABLE ステートメントで指定する必要があります。テーブル内のカラムの順序がわからない場合は、DESCRIBE tbl_name を使用して見つけます。
- SET 句は、各カラムに割り当てる値とともに、カラムを名前でも示的に指定します。

カラム値は、次のいくつかの方法で指定できます。

- 厳密な SQL モードが有効になっていない場合、値が明示的に指定されていないカラムはデフォルト (明示的または暗黙的) 値に設定されます。たとえば、テーブル内のすべてのカラムを指定していないカラムリストを指定した場

合、指定されていないカラムはそのデフォルト値に設定されます。デフォルト値の割り当てについては、[セクション11.6「データ型デフォルト値」](#)で説明されています。[セクション1.7.3.3「無効なデータに対する制約の施行」](#)も参照してください。

厳密な SQL モードが有効になっている場合、デフォルト値を持たないすべてのカラムに明示的な値が指定されていないと、`INSERT` ステートメントはエラーを生成します。[セクション5.1.11「サーバー SQL モード」](#)を参照してください。

- カラムリストと `VALUES` リストの両方が空である場合、`INSERT` は、各カラムがそのデフォルト値に設定された行を作成します。

```
INSERT INTO tbl_name () VALUES();
```

厳密モードが有効になっていない場合、MySQL では、デフォルトが明示的に定義されていないカラムに暗黙的なデフォルト値が使用されます。厳密モードが有効な場合、いずれかのカラムにデフォルト値がないとエラーが発生します。

- カラムを明示的にそのデフォルト値に設定するには、キーワード `DEFAULT` を使用します。これにより、テーブル内の各カラムの値が含まれていない不完全な `VALUES` リストを書かなくても済むため、いくつかのカラムを除くすべてのカラムに値を割り当てる `INSERT` ステートメントの記述が容易になります。それ以外の場合は、`VALUES` リストの各値に対応するカラム名のリストを指定する必要があります。
- 生成されたカラムが明示的に挿入される場合、許可される値は `DEFAULT` のみです。生成されるカラムの詳細は、[セクション13.1.20.8「CREATE TABLE および生成されるカラム」](#)を参照してください。
- 式では、`DEFAULT(col_name)` を使用してカラム `col_name` のデフォルト値を生成できます。
- カラム値を提供する式 `expr` の型変換は、式のデータ型がカラムのデータ型と一致しない場合に発生することがあります。特定の値を変換すると、カラムタイプに応じて異なる値が挿入される可能性があります。たとえば、文字列 `'1999.0e-2'` を `INT`、`FLOAT`、`DECIMAL(10,6)` に挿入したり、`YEAR` カラムを挿入すると、値 `1999`、`19.9921`、`19.992100` または `1999` がそれぞれ挿入されます。文字列から数値への変換では、文字列の初期部分のみが有効な整数または年とみなされる可能性があるため、`INT` および `YEAR` カラムに格納される値は `1999` です。`FLOAT` および `DECIMAL` カラムの場合、文字列から数値への変換では、文字列全体が有効な数値とみなされます。
- 式 `expr` は、以前に値リスト内に設定された任意のカラムを参照できます。たとえば、次のステートメントは、`col2` の値が、前に割り当てられている `col1` を参照しているため実行可能です。

```
INSERT INTO tbl_name (col1,col2) VALUES(15,col1*2);
```

ただし、次のステートメントは、`col1` の値が、`col1` のあとに割り当てられている `col2` を参照しているため正当ではありません。

```
INSERT INTO tbl_name (col1,col2) VALUES(col2*2,15);
```

`AUTO_INCREMENT` 値を含むカラムに対して例外が発生します。`AUTO_INCREMENT` 値は他の値の割当て後に生成されるため、割当て内の `AUTO_INCREMENT` カラムへの参照はすべて `0` を返します。

`VALUES` 構文を使用する `INSERT` ステートメントは複数の行を挿入できます。これを行うには、カンマで区切られたカラム値の複数のリストをカッコで囲み、カンマで区切って含めます。例:

```
INSERT INTO tbl_name (a,b,c)
VALUES(1,2,3), (4,5,6), (7,8,9);
```

各値リストには、行ごとに挿入されるのと同じ数の値が含まれている必要があります。次のステートメントは、それぞれ 3 つの値のリストではなく、9 つの値のリストが 1 つ含まれているため、無効です:

```
INSERT INTO tbl_name (a,b,c) VALUES(1,2,3,4,5,6,7,8,9);
```

このコンテキストでは、`VALUE` は `VALUES` のシノニムです。値リストの数やリスト当たりの値の数については何も意味しません。リストごとの値の数に関係なく、単一の値リストまたは複数のリストのいずれかを使用できます。

`VALUES ROW()` 構文を使用する `INSERT` ステートメントでは、複数の行を挿入することもできます。この場合、各値リストは、次のように `ROW()` (行コンストラクタ) 内に含まれている必要があります:

```
INSERT INTO tbl_name (a,b,c)
```



```
VALUES ROW(1,2,3), ROW(4,5,6), ROW(7,8,9);
```

INSERT の影響を受ける行の値は、`ROW_COUNT()` SQL 関数または `mysql_affected_rows()` C API 関数を使用して取得できます。 [セクション12.16「情報関数」](#) および `mysql_affected_rows()` を参照してください。

複数の値リスト、`INSERT ... SELECT` または `INSERT ... TABLE` で `INSERT ... VALUES` または `INSERT ... VALUES ROW()` を使用する場合は、このステートメントは次の形式で情報文字列を返します:

```
Records: N1 Duplicates: N2 Warnings: N3
```

C API を使用している場合は、`mysql_info()` 関数を呼び出すことによって情報文字列を取得できます。 `mysql_info()` を参照してください。

`Records` は、このステートメントによって処理された行数を示します。(これは、`Duplicates` が 0 以外であることがあるため、必ずしも実際に挿入された行数ではありません。) `Duplicates` は、何らかの既存の一意のインデックス値を複製しているために挿入できなかった行数を示します。 `Warnings` は、何らかの点で問題があったカラム値を挿入するための試行回数を示します。警告は、次のいずれかの条件で発生する場合があります。

- `NOT NULL` として宣言されているカラムへの `NULL` の挿入。複数行の `INSERT` ステートメントまたは `INSERT INTO ... SELECT` ステートメントの場合、このカラムは、そのカラムデータ型の暗黙のデフォルト値に設定されます。これは、数値型では 0、文字列型では空の文字列 (")、および日付と時間型では「0」の値です。サーバーは `SELECT` からの結果セットを検査して、それが単一行を返すかどうかを確認しないため、`INSERT INTO ... SELECT` ステートメントは複数行の挿入と同じ方法で処理されます。(単一行の `INSERT` の場合は、`NULL` が `NOT NULL` カラムに挿入されても警告は発生しません。代わりに、このステートメントがエラーで失敗します。)
- 数値カラムの、そのカラムの範囲外にある値への設定。この値は、その範囲のもっとも近い端点にクリップされます。
- 数値カラムへの '10.34 a' などの値の割り当て。後続の非数値のテキストは取り除かれ、残りの数値部分が挿入されます。文字列値に先頭の数値部分が含まれていない場合、このカラムは 0 に設定されます。
- 文字列カラム (`CHAR`、`VARCHAR`、`TEXT`、または `BLOB`) への、そのカラムの最大長を超える文字列の挿入。この値は、そのカラムの最大長に切り捨てられます。
- 日付または時間カラムへの、そのデータ型として不正な値の挿入。このカラムは、その型の適切な 0 の値に設定されます。
- `AUTO_INCREMENT` のカラム値を含む `INSERT` の例は、[セクション3.6.9「AUTO_INCREMENT の使用」](#) を参照してください。

`INSERT` が `AUTO_INCREMENT` カラムを含むテーブルに行を挿入する場合は、`LAST_INSERT_ID()` SQL 関数または `mysql_insert_id()` C API 関数を使用して、そのカラムに使用される値を検索できます。

注記

これらの 2 つの関数が、必ずしも同じ動作を行うとは限りません。 `AUTO_INCREMENT` カラムに関連した `INSERT` ステートメントの動作については、[セクション12.16「情報関数」](#) および `mysql_insert_id()` でさらに詳細に説明されています。

`INSERT` ステートメントは、次の修飾子をサポートします。

- `LOW_PRIORITY` 修飾子を使用すると、ほかのクライアントがテーブルから読み取ることがなくなるまで、`INSERT` の実行が遅延されます。これには、既存のクライアントが読み取っている間や、`INSERT LOW_PRIORITY` ステートメントが待機している間に読み取りを開始したほかのクライアントが含まれます。したがって、`INSERT LOW_PRIORITY` ステートメントを発行するクライアントが非常に長い時間待機する可能性があります。

`LOW_PRIORITY` は、テーブルレベルロック (`MyISAM`、`MEMORY`、`MERGE` など) のみを使用するストレージエンジンにのみ影響します。

注記

同時挿入が無効になるため、`LOW_PRIORITY` は通常、`MyISAM` テーブルでは使用しないでください。 [セクション8.11.3「同時挿入」](#) を参照してください。

- `HIGH_PRIORITY` を指定すると、サーバーが `--low-priority-updates` オプションで起動されている場合に、その効果がオーバーライドされます。また、同時挿入も使用されなくなります。 [セクション8.11.3「同時挿入」](#) を参照してください。

`HIGH_PRIORITY` は、テーブルレベルロック (`MyISAM`、`MEMORY`、`MERGE` など) のみを使用するストレージエンジンにのみ影響します。

- `IGNORE` 修飾子を使用すると、`INSERT` ステートメントの実行中に発生する無視可能なエラーは無視されます。たとえば、`IGNORE` を使用しない場合は、テーブル内の既存の `UNIQUE` インデックスまたは `PRIMARY KEY` 値を複製する行によって重複キーエラーが発生し、このステートメントは中止されます。`IGNORE` を指定すると、その行が破棄され、エラーは発生しません。無視されたエラーでは、かわりに警告が生成されます。

`IGNORE` には、特定の値に一致するパーティションが見つからないパーティション化されたテーブルへの挿入でも同様の効果があります。`IGNORE` がない場合、このような `INSERT` ステートメントはエラーで中断されます。`INSERT IGNORE` を使用すると、一致しない値を含む行に対しては挿入操作が暗黙的に失敗しますが、一致する行は挿入されます。例については、[セクション24.2.2「LIST パーティショニング」](#) を参照してください。

`IGNORE` が指定されていない場合は、エラーをトリガーするデータ変換によってステートメントが中止されます。`IGNORE` を指定すると、無効な値はもっとも近い値に調整されて挿入されます。警告は生成されますが、ステートメントは中止されません。`mysql_info()` C API 関数を使用すると、テーブルに実際に挿入された行数を確認できます。

詳細は、[IGNORE がステートメントの実行に与える影響](#) を参照してください。

古い行を上書きするには、`INSERT` の代わりに `REPLACE` を使用できます。`REPLACE` は、古い行を複製する一意のキー値を含む新しい行の処理における `INSERT IGNORE` の対応する機能です: 新しい行は、破棄されるのではなく、古い行を置き換えます。[セクション13.2.9「REPLACE ステートメント」](#) を参照してください。

- `ON DUPLICATE KEY UPDATE` を指定し、`UNIQUE` インデックスまたは `PRIMARY KEY` で値が重複する原因となる行を挿入すると、古い行の `UPDATE` が発生します。行ごとの影響を受けた行の値は、その行が新しい行として挿入された場合は 1、既存の行が更新された場合は 2、既存の行がその現在の値に設定された場合は 0 です。`mysqld` への接続時に `mysql_real_connect()` C API 関数に `CLIENT_FOUND_ROWS` フラグを指定すると、既存の行が現在の値に設定されている場合、影響を受ける行の値は 1 (0 ではなく) になります。[セクション13.2.6.2「INSERT ... ON DUPLICATE KEY UPDATE ステートメント」](#) を参照してください。
- `INSERT DELAYED` は MySQL 5.6 で非推奨となり、最終的な削除がスケジュールされています。MySQL 8.0 では、`DELAYED` 修飾子は受け入れられますが無視されます。代わりに `INSERT (DELAYED を付けない)` を使用してください。[セクション13.2.6.3「INSERT DELAYED ステートメント」](#) を参照してください。

13.2.6.1 INSERT ... SELECT ステートメント

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name
[PARTITION (partition_name [, partition_name] ...)]
[(col_name [, col_name] ...)]
{SELECT ... | TABLE table_name}
[ON DUPLICATE KEY UPDATE assignment_list]

value:
{expr | DEFAULT}

assignment:
col_name = value

assignment_list:
assignment [, assignment] ...
```

`INSERT ... SELECT` を使用すると、複数のテーブルから選択できる `SELECT` ステートメントの結果から、テーブルに多数の行をすばやく挿入できます。例:

```
INSERT INTO tbl_temp2 (fld_id)
SELECT tbl_temp1.fld_order_id
FROM tbl_temp1 WHERE tbl_temp1.fld_order_id > 100;
```

MySQL 8.0.19 以降、次に示すように、`SELECT` のかわりに `TABLE` ステートメントを使用できます:

```
INSERT INTO ta TABLE tb;
```

`TABLE tb` は、`SELECT * FROM tb` と同等です。これは、ソーステーブルのすべてのカラムをターゲットテーブルに挿入し、`WHERE` によるフィルタリングが不要な場合に便利です。また、`TABLE` の行は `ORDER BY` を使用して 1 つ以上のカラムで順序付けでき、挿入される行数は `LIMIT` 句を使用して制限できます。詳細は、[セクション 13.2.12 「TABLE ステートメント」](#) を参照してください。

次の条件は、`INSERT ... SELECT` ステートメント、および特に明記されている場合を除き、`INSERT ... TABLE` にも適用されます：

- 重複キー違反の原因になる行を無視するには、`IGNORE` を指定します。
- `INSERT` ステートメントのターゲットテーブルは、クエリーの `SELECT` 部分の `FROM` 句に、または `TABLE` によって指定されたテーブルとして指定できます。ただし、テーブルに挿入し、さらにサブクエリーで同じテーブルから選択することはできません。

同じテーブルから選択して同じテーブルに挿入する場合、MySQL は、`SELECT` の行を保持する内部一時テーブルを作成し、それらの行をターゲットテーブルに挿入します。ただし、同じステートメントで `TEMPORARY` テーブルを 2 回参照することはできないため、`t` が `TEMPORARY` テーブルの場合は `INSERT INTO t ... SELECT ... FROM t` を使用できません。同じ理由で、`t` が一時テーブルの場合は `INSERT INTO t ... TABLE t` を使用できません。[セクション 8.4.4 「MySQL での内部一時テーブルの使用」](#) および [セクション B.3.6.2 「TEMPORARY テーブルに関する問題」](#) を参照してください。

- `AUTO_INCREMENT` カラムは、通常どおりに機能します。
- バイナリログを使用して元のテーブルを再作成できるようにするために、MySQL では `INSERT ... SELECT` または `INSERT ... TABLE` ステートメントの同時挿入が許可されていません ([セクション 8.11.3 「同時挿入」](#) を参照)。
- `SELECT` と `INSERT` が同じテーブルを参照している場合のあいまいなカラム参照の問題を回避するには、`SELECT` 部分で使用されている各テーブルの一意のエイリアスを指定し、その部分にあるカラム名を適切なエイリアスで修飾します。

`TABLE` ステートメントはエイリアスをサポートしていません。

ソーステーブルまたはターゲットテーブル (あるいはその両方) のどのパーティションまたはサブパーティション (あるいはその両方) を、テーブルの名前に続く `PARTITION` オプションとともに使用するかを明示的に選択できます。`PARTITION` がこのステートメントの `SELECT` 部分にあるソーステーブルの名前とともに使用されている場合は、そのパーティションリストで指定されているパーティションまたはサブパーティションの行のみが選択されます。`PARTITION` をステートメントの `INSERT` 部分のターゲットテーブルの名前とともに使用する場合、選択したすべての行を、オプションの後のパーティションリストで指定されたパーティションまたはサブパーティションに挿入する必要があります。それ以外の場合、`INSERT ... SELECT` ステートメントは失敗します。詳細および例については、[セクション 24.5 「パーティション選択」](#) を参照してください。

`TABLE` は、`PARTITION` オプションをサポートしていません。

`INSERT ... SELECT` ステートメントの場合、`ON DUPLICATE KEY UPDATE` 句で `SELECT` カラムを参照できる条件については、[セクション 13.2.6.2 「INSERT ... ON DUPLICATE KEY UPDATE ステートメント」](#) を参照してください。これは、`INSERT ... TABLE` でも機能します。

`ORDER BY` 句のない `SELECT` ステートメントまたは `TABLE` ステートメントが行を戻す順序は、非決定的です。つまり、レプリケーションを使用している場合、このような `SELECT` がソースとスレーブで同じ順序で行を返すことは保証されないため、それらの間に不整合が生じる可能性があります。これが発生しないようにするには、ソースとレプリカで同じ行順序を生成する `ORDER BY` 句を使用して、レプリケートされる `INSERT ... SELECT` ステートメントまたは `INSERT ... TABLE` ステートメントを常に記述します。[セクション 17.5.1.18 「レプリケーションと LIMIT」](#) も参照してください。

この問題のため、`INSERT ... SELECT ON DUPLICATE KEY UPDATE` および `INSERT IGNORE ... SELECT` ステートメントには、ステートメントベースレプリケーションに対して安全でないというフラグが付けられます。このようなステートメントは、ステートメントベースのモードの使用時にエラーログに警告を生成し、`MIXED` モードの使用時に行ベースの形式を使用してバイナリログに書き込まれます。(Bug #11758262、Bug #50439)

[セクション 17.2.1.1 「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」](#) も参照してください。

13.2.6.2 INSERT ... ON DUPLICATE KEY UPDATE ステートメント

`ON DUPLICATE KEY UPDATE` 句を指定し、行を挿入すると、`UNIQUE` インデックスまたは `PRIMARY KEY` で値が重複する場合、古い行の `UPDATE` が発生します。たとえば、カラム `a` が `UNIQUE` として宣言され、値 `1` を含んでいる場合、次の 2 つのステートメントには同様の効果があります。

```
INSERT INTO t1 (a,b,c) VALUES (1,2,3)
ON DUPLICATE KEY UPDATE c=c+1;

UPDATE t1 SET c=c+1 WHERE a=1;
```

効果はまったく同じではありません: `a` が自動インクリメントカラムである InnoDB テーブルの場合、`INSERT` ステートメントは自動インクリメント値を増やしますが、`UPDATE` は増やしません。

カラム `b` も一意である場合、`INSERT` は、代わりに次の `UPDATE` ステートメントと同等です。

```
UPDATE t1 SET c=c+1 WHERE a=1 OR b=2 LIMIT 1;
```

`a=1 OR b=2` が複数の行に一致する場合は、1 つの行だけが更新されます。一般に、一意のインデックスが複数含まれているテーブルに対して `ON DUPLICATE KEY UPDATE` 句を使用することは避けるようにしてください。

`ON DUPLICATE KEY UPDATE` を使用した場合、行ごとの影響を受けた行の値は、その行が新しい行として挿入された場合は 1、既存の行が更新された場合は 2、既存の行がその現在の値に設定された場合は 0 です。mysql への接続時に `mysql_real_connect()` C API 関数に `CLIENT_FOUND_ROWS` フラグを指定すると、既存の行が現在の値に設定されている場合、影響を受ける行の値は 1 (0 ではなく) になります。

テーブルに `AUTO_INCREMENT` カラムが含まれているときに、`INSERT ... ON DUPLICATE KEY UPDATE` で行を挿入または更新した場合、`LAST_INSERT_ID()` 関数は `AUTO_INCREMENT` 値を返します。

`ON DUPLICATE KEY UPDATE` 句には、カンマで区切られた、複数のカラム割り当てを含めることができます。

`ON DUPLICATE KEY UPDATE` 句の代入式では、`VALUES(col_name)` 関数を使用して `INSERT ... ON DUPLICATE KEY UPDATE` ステートメントの `INSERT` 部分からカラム値を参照できます。つまり、`ON DUPLICATE KEY UPDATE` 句にある `VALUES(col_name)` は、重複キーの競合が発生していない場合に挿入される `col_name` の値を参照します。この関数は、複数の行を挿入する際に特に役立ちます。`VALUES()` 関数は、`ON DUPLICATE KEY UPDATE` 句または `INSERT` ステートメントでのみ意味があり、それ以外の場合は `NULL` を返します。例:

```
INSERT INTO t1 (a,b,c) VALUES (1,2,3),(4,5,6)
ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

そのステートメントは、次の 2 つのステートメントと同一です。

```
INSERT INTO t1 (a,b,c) VALUES (1,2,3)
ON DUPLICATE KEY UPDATE c=3;
INSERT INTO t1 (a,b,c) VALUES (4,5,6)
ON DUPLICATE KEY UPDATE c=9;
```

注記

新しい行とカラムを参照するための `VALUES()` の使用は、MySQL 8.0.20 以降非推奨になり、将来のバージョンの MySQL で削除される予定です。かわりに、このセクションの次のいくつかの段落で説明するように、行およびカラムのエイリアスを使用します。

MySQL 8.0.19 以降では、`VALUES` または `SET` 句の後に `AS` キーワードを付けて、挿入する行のエイリアス (オプションでそのカラムの 1 つ以上) を使用できます。行エイリアス `new` を使用すると、以前に `VALUES()` を使用して新しいカラム値にアクセスしていたステートメントを、次に示す形式で記述できます:

```
INSERT INTO t1 (a,b,c) VALUES (1,2,3),(4,5,6) AS new
ON DUPLICATE KEY UPDATE c = new.a+new.b;
```

また、カラムエイリアス `m`、`n` および `p` を使用する場合は、代入句で行エイリアスを省略して、次のような同じステートメントを記述できます:

```
INSERT INTO t1 (a,b,c) VALUES (1,2,3),(4,5,6) AS new(m,n,p)
```

```
ON DUPLICATE KEY UPDATE c = m+n;
```

この方法でカラムエイリアスを使用する場合、代入句で直接使用しない場合でも、**VALUES** 句の後に行エイリアスを使用する必要があります。

前述のように、**SET** 句で行およびカラムのエイリアスを使用することもできます。前述の 2 つの **INSERT ... ON DUPLICATE KEY UPDATE** ステートメントで、**VALUES** のかわりに **SET** を使用すると、次のように実行できます：

```
INSERT INTO t1 SET a=1,b=2,c=3 AS new
ON DUPLICATE KEY UPDATE c = new.a+new.b;

INSERT INTO t1 SET a=1,b=2,c=3 AS new(m,n,p)
ON DUPLICATE KEY UPDATE c = m+n;
```

行のエイリアスは、テーブルの名前と同じにできません。カラムエイリアスが使用されていない場合、またはカラム名と同じ場合は、**ON DUPLICATE KEY UPDATE** 句の行エイリアスを使用して区別する必要があります。カラムのエイリアスは、適用される行のエイリアスに関して一意である必要があります（つまり、同じ行のカラムを参照するカラムのエイリアスは同じにできません）。

INSERT ... SELECT ステートメントの場合、**ON DUPLICATE KEY UPDATE** 句で参照できる **SELECT** クエリー式の許容形式に関して、次のルールが適用されます：

- 単一のテーブルに対するクエリーからのカラムへの参照（導出テーブルの場合もあります）。
- 複数のテーブルに対する結合のクエリーからのカラムへのリファレンス。
- **DISTINCT** クエリーからのカラムへのリファレンス。
- **SELECT** が **GROUP BY** を使用しないかぎり、他のテーブルのカラムへのリファレンス。副作用の 1 つは、一意でないカラム名への参照を修飾する必要があることです。

UNION からのカラムの参照はサポートされていません。この制限を回避するには、**UNION** を導出テーブルとして書き換え、その行を単一テーブルの結果セットとして処理できるようにします。たとえば、次のステートメントはエラーを生成します：

```
INSERT INTO t1 (a, b)
SELECT c, d FROM t2
UNION
SELECT e, f FROM t3
ON DUPLICATE KEY UPDATE b = b + c;
```

かわりに、**UNION** を導出テーブルとしてリライトする同等のステートメントを使用します：

```
INSERT INTO t1 (a, b)
SELECT * FROM
(SELECT c, d FROM t2
UNION
SELECT e, f FROM t3) AS dt
ON DUPLICATE KEY UPDATE b = b + c;
```

クエリーを導出テーブルとしてリライトする方法では、**GROUP BY** クエリーからカラムを参照することもできます。

INSERT ... SELECT ステートメントの結果は **SELECT** からの行の順序に依存するため、この順序は常に保証されるわけではないため、ソースとスレーブの **INSERT ... SELECT ON DUPLICATE KEY UPDATE** ステートメントの相違を記録するときに可能です。したがって、ステートメントベースのレプリケーションでは、**INSERT ... SELECT ON DUPLICATE KEY UPDATE** ステートメントに安全でないというフラグが付けられます。このようなステートメントは、ステートメントベースのモードの使用時にエラーログに警告を生成し、**MIXED** モードの使用時に行ベースの形式を使用してバイナリログに書き込まれます。複数の一意キーまたは主キーを持つテーブルに対する **INSERT ... ON DUPLICATE KEY UPDATE** ステートメントも安全でないとマークされます。(Bug #11765650、Bug #58637)

セクション 17.2.1.1 「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」も参照してください。

13.2.6.3 INSERT DELAYED ステートメント

```
INSERT DELAYED ...
```

INSERT ステートメントの **DELAYED** オプションは、標準 SQL に対する MySQL の拡張機能です。以前のバージョンの MySQL では、特定の種類のテーブル (MyISAM など) に使用できるため、クライアントが **INSERT DELAYED** を使用すると、サーバーから OK が一度に取得され、テーブルが他のスレッドで使用されていない場合に挿入されるように行がキューに入れられます。

DELAYED の挿入および置換は、MySQL 5.6 で非推奨になりました。MySQL 8.0 では、**DELAYED** はサポートされません。サーバーは **DELAYED** キーワードを認識しますが、無視し、挿入を遅延なしの挿入として処理し、**ER_WARN_LEGACY_SYNTAX_CONVERTED** 警告を生成: **INSERT DELAYED** はサポートされなくなりました。ステートメントは **INSERT** に変換されました。 **DELAYED** キーワードは、将来のリリースで削除される予定です。

13.2.7 LOAD DATA ステートメント

```
LOAD DATA
[LOW_PRIORITY | CONCURRENT] [LOCAL]
INFILE 'file_name'
[REPLACE | IGNORE]
INTO TABLE tbl_name
[PARTITION (partition_name [, partition_name] ...)]
[CHARACTER SET charset_name]
{{FIELDS | COLUMNS}
  [TERMINATED BY 'string']
  [[OPTIONALLY] ENCLOSED BY 'char']
  [ESCAPED BY 'char']
}
[LINES
  [STARTING BY 'string']
  [TERMINATED BY 'string']
]
[IGNORE number {LINES | ROWS}]
[[col_name_or_user_var
  [, col_name_or_user_var] ...]]
[SET col_name={expr | DEFAULT}
  [, col_name={expr | DEFAULT}] ...]
```

LOAD DATA ステートメントは、テキストファイルからテーブルに非常に高速に行を読み取ります。 **LOAD DATA** は、**SELECT ... INTO OUTFILE** を補完したものです。(セクション13.2.10.1「**SELECT ... INTO** ステートメント」を参照してください。) テーブルからファイルにデータを書き込むには、**SELECT ... INTO OUTFILE** を使用します。 ファイルをテーブルに読み取るには、**LOAD DATA** を使用します。 **FIELDS** および **LINES** 句の構文は、両方のステートメントで同じです。

mysqlimport ユーティリティを使用してデータファイルをロードすることもできます。セクション4.5.5「**mysqlimport — データインポートプログラム**」を参照してください。 **mysqlimport** は、**LOAD DATA** ステートメントをサーバーに送信することによって動作します。

INSERT と **LOAD DATA** の効率および **LOAD DATA** の高速化の詳細は、セクション8.2.5.1「**INSERT ステートメントの最適化**」を参照してください。

- [パーティションテーブルのサポート](#)
- [入力ファイルの名前、場所およびコンテンツの解釈](#)
- [レプリケーションに関する考慮事項](#)
- [並列性に関する考慮事項](#)
- [重複キーの処理](#)
- [インデックス処理](#)
- [フィールドおよび明細の処理](#)
- [カラムリストの指定](#)
- [入力前処理](#)
- [ステートメントの結果情報](#)

- [その他のトピック](#)

パーティションテーブルのサポート

LOAD DATA では、パーティションまたはサブパーティション (あるいはその両方) のカンマ区切り名のリストを含む PARTITION オプションを使用した明示的なパーティション選択がサポートされています。このオプションが使用されているとき、リストで指定されているいずれかのパーティションまたはサブパーティションにファイルからの行を挿入できない場合、このステートメントは Found a row not matching the given partition set エラーで失敗します。詳細および例については、[セクション24.5「パーティション選択」](#)を参照してください。

入力ファイルの名前、場所およびコンテンツの解釈

ファイル名は、リテラル文字列として指定する必要があります。Windows では、パス名内のバックスラッシュをスラッシュまたは二重のバックスラッシュとして指定します。character_set_filesystem システム変数は、ファイル名文字セットの解釈を制御します。

サーバーは、character_set_database システム変数によって示されている文字セットを使用してファイル内の情報を解釈します。SET NAMES や、character_set_client の設定は入力の解釈に影響を与えません。入力ファイルの内容にデフォルトとは異なる文字セットが使用されている場合は、通常、CHARACTER SET 句を使用してそのファイルの文字セットを指定することをお勧めします。binary の文字セットは、「変換なし」を指定します。

LOAD DATA では、フィールド値がロードされるカラムのデータ型に関係なく、ファイル内のすべてのフィールドが同じ文字セットを持つと解釈されます。ファイルの内容が正しく解釈されるように、そのファイルが正しい文字セットで書き込まれていることを確認する必要があります。たとえば、mysqldump -T を使用して、または mysql で SELECT ... INTO OUTFILE ステートメントを発行してデータファイルを書き込む場合は、LOAD DATA でファイルをロードするときに使用される文字セットで出力が書き込まれるように、--default-character-set オプションを使用してください。

注記

ucs2、utf16、utf16le、または utf32 文字セットを使用するデータファイルはロードできません。

LOCAL 修飾子は、後で説明するように、ファイルの予期される場所およびエラー処理に影響します。LOCAL は、サーバーとクライアントの両方がそれを許可するように構成されている場合のみ機能します。たとえば、local_infile システム変数を無効にして mysqld を起動した場合、LOCAL は機能しません。[セクション6.1.6「LOAD DATA LOCAL のセキュリティ上の考慮事項」](#)を参照してください。

LOCAL 修飾子は、ファイルが見つかる場所に影響します:

- LOCAL が指定されている場合、ファイルはクライアントホスト上のクライアントプログラムによって読み取られ、サーバーに送信されます。このファイルは、その正確な場所を指定するためにフルパス名として指定できます。相対パス名として指定されている場合、その名前は、クライアントプログラムが起動されたディレクトリを基準にして解釈されます。

LOCAL を LOAD DATA とともに使用すると、MySQL サーバーが一時ファイルを格納するディレクトリにファイルのコピーが作成されます。[セクションB.3.3.5「MySQL が一時ファイルを格納する場所」](#)を参照してください。このディレクトリ内にコピーのための十分な領域がないと、LOAD DATA LOCAL ステートメントが失敗する場合があります。

- LOCAL が指定されていない場合、ファイルはサーバーホスト上にある必要があり、直接サーバーによって読み取られます。サーバーは、次のルールを使用してファイルを見つけます。
 - ファイル名が絶対パス名である場合、サーバーはそれを指定されたとおりに使用します。
 - ファイル名が1つ以上の先行コンポーネントを含む相対パス名である場合、サーバーは、サーバーのデータディレクトリを基準にしてファイルを検索します。
 - 先行コンポーネントを含まないファイル名が指定されている場合、サーバーは、デフォルトデータベースのデータベースディレクトリ内でそのファイルを探します。

LOCAL 以外のケースでは、これらのルールは、./myfile.txt という名前のファイルがサーバーのデータディレクトリから読み取られるのに対して、myfile.txt として指定されたファイルはデフォルトデータベースのデータベースディレク

トリから読み取られることを示します。たとえば、`db1` がデフォルトデータベースである場合、次の `LOAD DATA` ステートメントは、このステートメントが明示的に `db2` データベース内のテーブルにファイルをロードしているにもかかわらず、`db1` のデータベースディレクトリからファイル `data.txt` を読み取ります。

```
LOAD DATA INFILE 'data.txt' INTO TABLE db2.my_table;
```

注記

また、サーバーは `LOCAL` 以外のルールを使用して、`IMPORT TABLE` ステートメントの `.sdi` ファイルを検索します。

`LOCAL` 以外のロード操作は、サーバーにあるテキストファイルを読み取ります。セキュリティ上の理由から、このような操作には `FILE` 権限が必要です。セクション6.2.2「MySQL で提供される権限」を参照してください。また、`LOCAL` 以外のロード操作は、`secure_file_priv` システム変数設定の影響を受けます。変数値が空でないディレクトリ名の場合、ロードするファイルはそのディレクトリに配置する必要があります。変数値が空 (セキュアでない) の場合、ファイルはサーバーからのみ読み取り可能である必要があります。

`LOCAL` を使用すると、クライアントからサーバーへの接続を介してファイルコンテンツを送信する必要があるため、サーバーがファイルに直接アクセスするより少し時間がかかります。その一方で、ローカルファイルをロードするために `FILE` 権限は必要ありません。

`LOCAL` はまた、エラー処理にも影響を与えます。

- `LOAD DATA` では、データ解釈エラーおよび重複キーエラーによって操作が終了します。
- `LOAD DATA LOCAL` では、データ解釈および重複キーのエラーは警告になり、操作の途中でファイルの転送を停止する方法がサーバーにないため、操作は続行されます。重複キーエラーについては、これは `IGNORE` が指定されている場合と同じです。`IGNORE` については、このセクションのあとの方でさらに詳細に説明されています。

レプリケーションに関する考慮事項

`LOAD DATA` は、ステートメントベースレプリケーションでは安全でないとみなされます。`binlog_format=STATEMENT` が設定されているときに `LOAD DATA` を使用すると、データを含む一時ファイルが、変更が適用されるレプリケーションスレーブ上に作成されます。バイナリログの暗号化がサーバー上でアクティブな場合、この一時ファイルは暗号化されないことに注意してください。暗号化が必要な場合は、一時ファイルを作成しない行ベースまたは混合バイナリロギング形式を使用してください。`LOAD DATA` とレプリケーションの相互作用の詳細は、セクション17.5.1.19「レプリケーションと `LOAD DATA`」を参照してください。

並列性に関する考慮事項

`LOW_PRIORITY` 修飾子を使用すると、ほかのクライアントがテーブルから読み取ることがなくなるまで、`LOAD DATA` ステートメントの実行が遅延されます。これは、テーブルレベルロックのみを使用するストレージエンジン (`MyISAM`、`MEMORY`、および `MERGE`) にも影響を与えます。

同時挿入の条件を満たす (つまり、中央に空きブロックが含まれていない) `MyISAM` テーブルで `CONCURRENT` 修飾子を指定すると、`LOAD DATA` の実行中に他のスレッドがテーブルからデータを取得できます。この修飾子は、ほかのスレッドがそのテーブルを同時に使用していない場合でも、`LOAD DATA` のパフォーマンスに少し影響します。

重複キーの処理

`REPLACE` および `IGNORE` 修飾子は、一意のキー値で既存の行を複製する新しい (入力) 行の処理を制御します:

- `REPLACE` を指定すると、新しい行で既存の行が置き換えられます。つまり、既存の行と同じ主キーまたは一意インデックスの値を持つ行が既存の行を置換します。セクション13.2.9「`REPLACE` ステートメント」を参照してください。
- `IGNORE` を指定すると、一意のキー値で既存の行を複製する新しい行は破棄されます。詳細は、`IGNORE がステートメントの実行に与える影響`を参照してください。
- いずれの修飾子も指定しない場合、動作は `LOCAL` 修飾子が指定されているかどうかによって異なります。`LOCAL` が指定されていない場合は、重複キー値が見つかったらエラーが発生し、テキストファイルの残りは無視されます。

`LOCAL` が指定されている場合、デフォルトの動作は `IGNORE` が指定されている場合と同じです。これは、操作の最中にファイルの転送を停止する方法がサーバーにはないためです。

インデックス処理

ロード操作中に外部キー制約を無視するには、`LOAD DATA` を実行する前に `SET foreign_key_checks = 0` ステートメントを実行します。

空の `MyISAM` テーブルで `LOAD DATA` を使用する場合、一意でないすべてのインデックスが個別のバッチで作成されます (`REPAIR TABLE` の場合など)。通常、これにより、多数のインデックスがある場合に `LOAD DATA` がはるかに高速になります。一部の極端なケースでは、ファイルをテーブルにロードする前に `ALTER TABLE ... DISABLE KEYS` でインデックスを無効にし、ファイルをロードしたあとに `ALTER TABLE ... ENABLE KEYS` を使用してインデックスを再作成することによって、インデックスをさらに高速に作成できます。セクション8.2.5.1「`INSERT` ステートメントの最適化」を参照してください。

フィールドおよび明細の処理

`LOAD DATA` ステートメントと `SELECT ... INTO OUTFILE` ステートメントの両方で、`FIELDS` 句と `LINES` 句の構文は同じです。どちらの句もオプションですが、両方が指定される場合は、`FIELDS` を `LINES` の前に指定する必要があります。

`FIELDS` 句を指定する場合は、その各サブ句 (`TERMINATED BY`、`[OPTIONALLY] ENCLOSED BY`、および `ESCAPED BY`) もオプションです。ただし、そのうちの少なくとも1つを指定する必要があります。これらの句の引数には ASCII 文字のみを含めることができます。

`FIELDS` または `LINES` 句を指定しない場合、そのデフォルトは、次を記述した場合と同じです。

```
FIELDS TERMINATED BY '\t' ENCLOSED BY '"' ESCAPED BY '\\'
LINES TERMINATED BY '\n' STARTING BY ''
```

バックスラッシュは、SQL ステートメントの文字列内の MySQL エスケープ文字です。したがって、リテラルのバックスラッシュを指定するには、値が単一のバックスラッシュとして解釈されるように2つのバックスラッシュを指定する必要があります。エスケープシーケンス `\t` および `\n` では、それぞれタブ文字と改行文字が指定されます。

つまり、デフォルトでは、`LOAD DATA` は入力の読取り時に次のように動作します:

- 改行の位置にある行の境界を探します。
- 行の接頭辞はスキップしないでください。
- タブの位置で行をフィールドに分割します。
- フィールドが引用文字で囲まれていることを期待しません。
- 前にエスケープ文字 `\` がある文字をエスケープシーケンスとして解釈します。たとえば、`\t`、`\n` および `\\` は、それぞれタブ、改行およびバックスラッシュを示します。エスケープシーケンスの完全なリストについては、あとの `FIELDS ESCAPED BY` の説明を参照してください。

逆に、デフォルトでは、出力を書き込むとき `SELECT ... INTO OUTFILE` は次のように機能します。

- フィールド間にタブを書き込みます。
- フィールドを引用文字で囲みません。
- `\` を使用して、フィールド値内で発生するタブ、改行または `\` のインスタンスをエスケープします。
- 行の最後に改行を書き込みます。

注記

Windows システムで生成されたテキストファイルの場合、Windows プログラムでは通常、行終了記号として2文字を使用するため、適切なファイル読取りに `LINES TERMINATED`

BY '\r\n'が必要になることがあります。WordPad などの一部のプログラムは、ファイルを書き込むときに行ターミネータとして \r を使用する可能性があります。このようなファイルを読み取るには、LINES TERMINATED BY '\r' を使用します。

すべての入力行に無視する共通の接頭辞がある場合は、LINES STARTING BY 'prefix_string' を使用して接頭辞およびそれより前のものをスキップできます。行にプリフィクスが含まれていない場合は、行全体がスキップされます。たとえば、次のステートメントを発行するとします。

```
LOAD DATA INFILE '/tmp/test.txt' INTO TABLE test
FIELDS TERMINATED BY ',' LINES STARTING BY 'xxx';
```

データファイルは次のようになっています。

```
xxx"abc",1
something xxx"def",2
"ghi",3
```

結果の行は、("abc",1) および ("def",2) です。ファイル内の 3 行目は、プリフィクスが含まれていないためスキップされます。

IGNORE number LINES オプションを使用すると、ファイルの先頭にある行を無視できます。たとえば、IGNORE 1 LINES を使用して、カラム名を含む最初のヘッダー行をスキップできます：

```
LOAD DATA INFILE '/tmp/test.txt' INTO TABLE test IGNORE 1 LINES;
```

SELECT ... INTO OUTFILE を LOAD DATA とともに使用してデータベースからファイルにデータを書き込み、後でそのファイルをデータベースに読み取る場合、両方のステートメントのフィールド処理オプションと行処理オプションが一致する必要があります。そうしないと、LOAD DATA はファイルの内容を正しく解釈しません。SELECT ... INTO OUTFILE を使用して、カンマで区切られたフィールドを含むファイルを書き込むとします。

```
SELECT * INTO OUTFILE 'data.txt'
FIELDS TERMINATED BY ','
FROM table2;
```

カンマ区切りファイルを読み取るには、正しいステートメントは次のようになります：

```
LOAD DATA INFILE 'data.txt' INTO TABLE table2
FIELDS TERMINATED BY ',';
```

かわりに、次に示すステートメントを使用してファイルを読み取ろうとすると、フィールド間のタブを検索するように LOAD DATA に指示するため、機能しません：

```
LOAD DATA INFILE 'data.txt' INTO TABLE table2
FIELDS TERMINATED BY '\t';
```

その結果、各入力行が 1 つのフィールドとして解釈される可能性があります。

LOAD DATA を使用して、外部ソースから取得したファイルを読み取ることができます。たとえば、多くのプログラムは、各行にカンマで区切られ、二重引用符で囲まれた複数のフィールドが含まれており、かつ開始行がカラム名になっているようなカンマ区切り値 (CSV) 形式でデータをエクスポートできます。このようなファイル内の行が復帰改行と改行のペアで終了している場合、次に示すステートメントは、このファイルをロードするために使用するフィールド処理と行処理のオプションを示しています。

```
LOAD DATA INFILE 'data.txt' INTO TABLE tbl_name
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
IGNORE 1 LINES;
```

入力値が必ずしも引用符で囲まれていない場合は、ENCLOSED BY オプションの前に OPTIONALLY を使用します。

フィールド処理または行処理のどのオプションにも、空の文字列 ("") を指定できます。空でない場合、FIELDS [OPTIONALLY] ENCLOSED BY および FIELDS ESCAPED BY 値は単一の文字である必要があります。FIELDS TERMINATED BY、LINES STARTING BY、および LINES TERMINATED BY 値は、複数の文字にすることができます。たとえば、復帰改行と改行のペアで終了する行を書き込むか、またはこのような行を含むファイルを読み取るには、LINES TERMINATED BY '\r\n' 句を指定します。

%% から成る行で区切られたジョークを含むファイルを読み取るには、次のようにできます。

```
CREATE TABLE jokes
(a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
joke TEXT NOT NULL);
LOAD DATA INFILE '/tmp/jokes.txt' INTO TABLE jokes
FIELDS TERMINATED BY "
LINES TERMINATED BY '\n%%\n' (joke);
```

FIELDS [OPTIONALLY] ENCLOSED BY は、フィールドの引用符を制御します。出力 (**SELECT ... INTO OUTFILE**) でワード **OPTIONALLY** を省略した場合は、すべてのフィールドが **ENCLOSED BY** 文字で囲まれます。フィールド区切り文字としてカンマを使用したこのような出力の例を次に示します。

```
"1","a string","100.20"
"2","a string containing a , comma","102.20"
"3","a string containing a \" quote","102.20"
"4","a string containing a \", quote and comma","102.20"
```

OPTIONALLY を指定した場合、**ENCLOSED BY** 文字は、文字列データ型 (**CHAR**、**BINARY**、**TEXT**、**ENUM** など) を持つカラムの値を囲むためにのみ使用されます。

```
1,"a string",100.20
2,"a string containing a , comma",102.20
3,"a string containing a \" quote",102.20
4,"a string containing a \", quote and comma",102.20
```

フィールド値内の **ENCLOSED BY** 文字の出現は、先頭に **ESCAPED BY** 文字を付けてエスケープされます。また、空の **ESCAPED BY** 値を指定すると、**LOAD DATA** で正しく読み取れない出力が誤って生成される可能性があります。たとえば、エスケープ文字が空である場合、今示した前の出力は次のようになります。4 行目の 2 番目のフィールドに含まれる引用符のあとにカンマが続いていることに注目してください。これにより、このフィールドが (誤って) 終了するように見えます。

```
1,"a string",100.20
2,"a string containing a , comma",102.20
3,"a string containing a " quote",102.20
4,"a string containing a ", quote and comma",102.20
```

入力では、**ENCLOSED BY** 文字 (存在する場合) はフィールド値の最後から取り除かれます。(これは、**OPTIONALLY** が指定されているかどうかには関係しません。**OPTIONALLY** は、入力の解釈には影響を与えません。) **ENCLOSED BY** 文字が **ESCAPED BY** 文字のあとに現れた場合は、現在のフィールド値の一部として解釈されます。

フィールドが **ENCLOSED BY** 文字で始まったとき、その文字のインスタンスがフィールド値の終了として認識されるのは、そのあとにフィールドまたは行の **TERMINATED BY** シーケンスが続いている場合だけです。あいまいさを避けるために、フィールド値の中に **ENCLOSED BY** 文字が現れるときはそれを 2 文字にすることができ、それがその文字の単一インスタンスとして解釈されます。たとえば、**ENCLOSED BY ""** が指定されている場合、引用符は次に示すように処理されます。

```
"The ""BIG"" boss" -> The "BIG" boss
The "BIG" boss    -> The "BIG" boss
The ""BIG"" boss  -> The ""BIG"" boss
```

FIELDS ESCAPED BY は、特殊文字の読み取りまたは書き込みの方法を制御します。

- 入力では、**FIELDS ESCAPED BY** 文字が空でない場合、その文字が現れると取り除かれ、それに続く文字がフィールド値の一部として文字どおりに解釈されます。最初の文字がエスケープ文字である一部の 2 文字シーケンスは例外です。これらのシーケンスを (エスケープ文字に) 使用して) 次の表に示します。NULL 処理のルールについては、このセクションのあとの方で説明されています。

文字	エスケープシーケンス
\0	ASCII NUL (X'00') 文字
\b	バックスペース文字
\n	改行 (ラインフィード) 文字
\r	復帰改行文字
\t	タブ文字。

文字	エスケープシーケンス
\Z	ASCII 26 (Ctrl+Z)
\N	NULL

\でのエスケープ構文の詳細は、[セクション9.1.1「文字列リテラル」](#)を参照してください。

FIELDS ESCAPED BY 文字が空である場合、エスケープシーケンスの解釈は実行されません。

- 出力では、**FIELDS ESCAPED BY** 文字が空でない場合、その文字は、出力上で次の文字の前に付けるために使用されます。
 - FIELDS ESCAPED BY** 文字
 - FIELDS [OPTIONALLY] ENCLOSED BY** 文字
 - ENCLOSED BY** 文字が空または指定されていない場合の、**FIELDS TERMINATED BY** および **LINES TERMINATED BY** 値の最初の文字。
 - ASCII 0 (エスケープ文字のあとに実際に書き込まれる文字は 0 の値のバイトではなく、ASCII の 0 です)

FIELDS ESCAPED BY 文字が空である場合は、どの文字もエスケープされず、**NULL** は **\N** ではなく、**NULL** として出力されます。特に、データ内のフィールド値に今指定したリスト内のいずれかの文字が含まれている場合、空のエスケープ文字を指定することはおそらく適切な方法ではありません。

特定のケースでは、フィールド処理と行処理のオプションは相互に作用します。

- LINES TERMINATED BY** が空の文字列であり、かつ **FIELDS TERMINATED BY** が空以外である場合、行は **FIELDS TERMINATED BY** でも終了します。
- FIELDS TERMINATED BY** と **FIELDS ENCLOSED BY** の値がどちらも空 (") である場合は、固定行 (区切られていない) フォーマットが使用されます。固定行フォーマットでは、フィールド間に区切り文字は使用されませんが (ただし、行ターミネータは引き続き存在できます)。代わりに、カラム値は、そのフィールド内のすべての値を保持するために十分に広いフィールド幅を使用して読み取りと書き込みが行われます。**TINYINT**、**SMALLINT**、**MEDIUMINT**、**INT**、および **BIGINT** では、宣言されている表示幅にかかわらず、フィールド幅はそれぞれ 4、6、8、11、および 20 です。

LINES TERMINATED BY は引き続き、行を区切るために使用されます。ある行にすべてのフィールドが含まれていない場合、カラムの残りの部分はそのデフォルト値に設定されます。行ターミネータが存在しない場合は、これを " に設定してください。この場合は、テキストファイルの各行にすべてのフィールドが含まれている必要があります。

固定行フォーマットはまた、あとで説明されているように、**NULL** 値の処理にも影響を与えます。

注記

マルチバイト文字セットを使用している場合、固定サイズフォーマットは機能しません。

NULL 値の処理は、使用されている **FIELDS** および **LINES** オプションによって異なります。

- デフォルトの **FIELDS** および **LINES** 値では、**NULL** は出力として **\N** のフィールド値として書き込まれ、**\N** のフィールド値は入力として **NULL** として読み取られます (**ESCAPED BY** 文字は \ であると仮定します)。
- FIELDS ENCLOSED BY** が空でない場合、リテラルワード **NULL** をその値として含むフィールドは **NULL** 値として読み取られます。これは、文字列 'NULL' として読み取られる、**FIELDS ENCLOSED BY** 文字で囲まれたワード **NULL** とは異なります。
- FIELDS ESCAPED BY** が空である場合、**NULL** はワード **NULL** として書き込まれます。
- 固定行フォーマット (これは、**FIELDS TERMINATED BY** と **FIELDS ENCLOSED BY** がどちらも空であるときに使用されます) では、**NULL** は空の文字列として書き込まれます。これにより、**NULL** 値とテーブル内の空の文字列の両方が空の文字列として書き込まれるため、両方をファイルに書き込むときに区別できなくなります。ファイルを読み戻したときにこの 2 つを区別できることが必要な場合は、固定行フォーマットを使用すべきではありません。

NULL を NOT NULL カラムにロードしようとする、そのカラムのデータ型の暗黙のデフォルト値の割り当てが行われて警告が発生するか、または厳密な SQL モードではエラーが発生します。暗黙のデフォルト値については、[セクション 11.6 「データ型デフォルト値」](#) で説明されています。

一部のケースは、LOAD DATA でサポートされていません:

- 固定サイズ行 (FIELDS TERMINATED BY と FIELDS ENCLOSED BY がどちらも空) および BLOB または TEXT カラム。
- 別のセパレータと同じセパレータまたは別のセパレータの接頭辞を指定すると、LOAD DATA は入力を正しく解釈できません。たとえば、次の FIELDS 句では問題が発生します。

```
FIELDS TERMINATED BY "" ENCLOSED BY ""
```

- FIELDS ESCAPED BY が空の場合、FIELDS ENCLOSED BY または LINES TERMINATED BY の出現箇所と FIELDS TERMINATED BY 値が含まれるフィールド値によって、LOAD DATA はフィールドまたは行の読取りを早すぎる状態で停止します。これは、フィールドまたは行の値がどこで終了するかを LOAD DATA が適切に判断できないために発生します。

カラムリストの指定

次の例では、persondata テーブルのすべてのカラムをロードします。

```
LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata;
```

デフォルトでは、LOAD DATA ステートメントの最後にカラムリストが指定されていない場合、入力行には各テーブルのカラムのフィールドが含まれている必要があります。テーブルのカラムの一部のみをロードする場合は、カラムリストを指定します。

```
LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata  
(col_name_or_user_var [, col_name_or_user_var] ...);
```

カラムリストはまた、入力ファイル内のフィールドの順序がテーブル内のカラムの順序と異なる場合にも指定する必要があります。そうしないと、MySQL は、入力フィールドとテーブルカラムを一致させる方法がわかりません。

入力前処理

各 col_name_or_user_var 値は、カラム名またはユーザー変数のいずれかです。ユーザー変数を使用すると、結果をカラムに割り当てる前に、SET 句を使用して値に対して前処理変換を実行できます。

SET 句内のユーザー変数は、いくつかの方法で使用できます。次の例では、最初の入力カラムを直接 t1.column1 の値に使用し、2 番目の入力カラムを、t1.column2 の値に使用される前に除算演算の対象になるユーザー変数に割り当てます。

```
LOAD DATA INFILE 'file.txt'  
INTO TABLE t1  
(column1, @var1)  
SET column2 = @var1/100;
```

SET 句を使用すると、入力ファイルからは取得されない値を指定できます。次のステートメントは、column3 を現在の日付と時間に設定します。

```
LOAD DATA INFILE 'file.txt'  
INTO TABLE t1  
(column1, column2)  
SET column3 = CURRENT_TIMESTAMP;
```

入力値をユーザー変数に割り当て、その変数をテーブルカラムには代入しないようにして、その値を破棄することもできます。

```
LOAD DATA INFILE 'file.txt'  
INTO TABLE t1  
(column1, @dummy, column2, @dummy, column3);
```

カラム/変数リストと SET 句の使用は、次の制限に従います。

- SET 句の代入では、割り当て演算子の左側にカラム名のみを置くようにしてください。

- **SET** の代入の右側では、サブクエリーを使用できます。カラムに代入される値を返すサブクエリーとして使用できるのは、スカラーサブクエリーだけです。また、サブクエリーを使用して、ロードされているテーブルから選択することはできません。
- **IGNORE** 句によって無視された行は、カラム/変数リストや **SET** 句では処理されません。
- ユーザー変数には表示幅がないため、固定行フォーマットのデータをロードする場合はユーザー変数を使用できません。

入力行を処理する場合、**LOAD DATA** はそれをフィールドに分割し、カラム/変数リストと **SET** 句に応じた値 (存在する場合) を使用します。そのあと、結果として得られる行がテーブルに挿入されます。そのテーブルに **BEFORE INSERT** または **AFTER INSERT** トリガーが存在する場合、これらのトリガーはそれぞれ、行挿入の前またはあとにアクティブ化されます。

入力行に含まれるフィールドが多すぎる場合は、余分なフィールドが無視され、警告数が 1 増えます。

入力行に含まれるフィールドが少なすぎる場合、入力フィールドがないテーブルカラムはそのデフォルト値に設定されます。デフォルト値の割り当てについては、[セクション11.6「データ型デフォルト値」](#)で説明されています。

空のフィールド値はフィールドがないとは見なされず、次のように解釈されます。

- 文字列型の場合、このカラムは空の文字列に設定されます。
- 数値型の場合、このカラムは **0** に設定されます。
- 日付と時間型の場合、このカラムはその型の適切な「0」の値に設定されます。[セクション11.2「日時データ型」](#)を参照してください。

これらは、**INSERT** または **UPDATE** ステートメントで空の文字列を文字列、数値、日付または時間の各型に明示的に割り当てた場合の結果と同じ値です。

空のフィールド値や正しくないフィールド値の処理は、SQL モードが制限的な値に設定されていると、今説明した処理とは異なってきます。たとえば、`sql_mode` が **TRADITIONAL** に設定されている場合、空の値または数値カラムの 'x' などの値を変換すると、0 に変換されるのではなくエラーになります。(LOCAL または IGNORE では、制限付き `sql_mode` 値であっても、エラーではなく警告が発生し、非制限 SQL モードで使用されるのと同じ最も近い値の動作を使用して行が挿入されます。これは、操作中にサーバーがファイルの転送を停止する方法がないために発生します。)

TIMESTAMP カラムが現在の日付と時間に設定されるのは、そのカラムに **NULL** 値 (つまり、**IN**) が存在し、かつそのカラムが **NULL** 値を許可するように宣言されていない場合、または **TIMESTAMP** カラムのデフォルト値が現在のタイムスタンプであり、かつフィールドリストが指定されたときにこのカラムがフィールドリストから省略されている場合だけです。

LOAD DATA はすべての入力を文字列とみなすため、**INSERT** ステートメントと同様に **ENUM** または **SET** カラムに数値を使用することはできません。**ENUM** および **SET** 値はすべて、文字列として指定する必要があります。

バイナリ表記法 (b'011010' など) を使用して **BIT** 値を直接ロードすることはできません。これを回避するには、**SET** 句を使用して先頭の b' および末尾の ' を削除し、base-2 から base-10 への変換を実行して MySQL が値を **BIT** カラムに適切にロードするようにします:

```
shell> cat /tmp/bit_test.txt
b'10'
b'1111111'
shell> mysql test
mysql> LOAD DATA INFILE '/tmp/bit_test.txt'
      INTO TABLE bit_test (@var1)
      SET b = CAST(CONV(MID(@var1, 3, LENGTH(@var1)-3), 2, 10) AS UNSIGNED);
Query OK, 2 rows affected (0.00 sec)
Records: 2 Deleted: 0 Skipped: 0 Warnings: 0

mysql> SELECT BIN(b+0) FROM bit_test;
+-----+
| BIN(b+0) |
+-----+
| 10      |
| 1111111 |
```

```
+-----+
2 rows in set (0.00 sec)
```

0b バイナリ表記法 (0b011010 など) の BIT 値の場合は、かわりに次の SET 句を使用して、先頭の 0b を削除します:

```
SET b = CAST(CONV(MID(@var1, 3, LENGTH(@var1)-2), 2, 10) AS UNSIGNED)
```

ステートメントの結果情報

LOAD DATA ステートメントが終了すると、次の形式の情報文字列が返されます:

```
Records: 1 Deleted: 0 Skipped: 0 Warnings: 0
```

警告は、INSERT ステートメント (セクション13.2.6「INSERT ステートメント」を参照) を使用して値を挿入する場合と同じ状況で発生しますが、入力行のフィールドが少なすぎるか多すぎる場合にも LOAD DATA によって警告が生成される点が異なります。

SHOW WARNINGS を使用すると、発生した問題に関する情報として最初の max_error_count 警告のリストを取得できます。セクション13.7.7.42「SHOW WARNINGS ステートメント」を参照してください。

C API を使用している場合は、mysql_info() 関数を呼び出すことによって、そのステートメントに関する情報を取得できます。mysql_info() を参照してください。

その他のトピック

Unix では、LOAD DATA でパイプから読み取る必要がある場合は次の手法を使用できます (この例では、/ ディレクトリのリストをテーブル db1.t1 にロードします)。

```
mkfifo /mysql/data/db1/ls.dat
chmod 666 /mysql/data/db1/ls.dat
find / -ls > /mysql/data/db1/ls.dat &
mysql -e "LOAD DATA INFILE 'ls.dat' INTO TABLE t1" db1
```

ここでは、ロードするデータを生成するコマンドと mysql コマンドを別々の端末で実行するか、バックグラウンドでデータ生成プロセスを実行する必要があります (前述の例を参照)。これを行わない場合、パイプは mysql プロセスによってデータが読み取られるまでブロックされます。

13.2.8 LOAD XML ステートメント

```
LOAD XML
[LOW_PRIORITY | CONCURRENT] [LOCAL]
INFILE 'file_name'
[REPLACE | IGNORE]
INTO TABLE [db_name.]tbl_name
[CHARACTER SET charset_name]
[ROWS IDENTIFIED BY '<tagname>']
[IGNORE number {LINES | ROWS}]
[(field_name_or_user_var
[, field_name_or_user_var] ...)]
[SET col_name={expr | DEFAULT}
[, col_name={expr | DEFAULT}] ...]
```

LOAD XML ステートメントは、XML ファイルからテーブルにデータを読み取ります。file_name は、リテラル文字列として指定する必要があります。オプションの ROWS IDENTIFIED BY 句内の tagname もリテラル文字列として指定し、山括弧 (< および >) で囲む必要があります。

LOAD XML は、XML 出力モードでの mysql クライアントの実行 (つまり、--xml オプションを使用したクライアントの起動) を補完するものとして機能します。テーブルから XML ファイルにデータを書き込むには、次に示すように、システムシェルから --xml および -e オプションを指定して mysql クライアントを呼び出すことができます:

```
shell> mysql --xml -e 'SELECT * FROM mydb.mytable' > file.xml
```

ファイルをテーブルに読み取るには、LOAD XML を使用します。デフォルトでは、<row> 要素は、データベーステーブル行と同等であると見なされます。これは、ROWS IDENTIFIED BY 句を使用して変更できます。

このステートメントは、次の 3 つの異なる XML 形式をサポートします。

- 属性としてのカラム名と、属性値としてのカラム値:

```
<row column1="value1" column2="value2" .../>
```

- タグとしてのカラム名と、これらのタグの内容としてのカラム値:

```
<row>
  <column1>value1</column1>
  <column2>value2</column2>
</row>
```

- カラム名は `<field>` タグの `name` 属性で、値はこれらのタグの内容:

```
<row>
  <field name='column1'>value1</field>
  <field name='column2'>value2</field>
</row>
```

これは、`mysqldump` などのほかの MySQL ツールによって使用される形式です。

同じ XML ファイルで 3 つのすべての形式を使用できます。インポートルーチンは各行の形式を自動的に検出し、それを正しく解釈します。タグは、タグまたは属性名とカラム名に基づいて照合されます。

MySQL 8.0.21 より前は、`LOAD XML` はソース XML の `CDATA` セクションをサポートしていませんでした。(Bug #30753708、Bug #98199)

次の句は、基本的に `LOAD XML` に対して `LOAD DATA` に対する場合と同じように機能します。

- `LOW_PRIORITY` または `CONCURRENT`
- `LOCAL`
- `REPLACE` または `IGNORE`
- `CHARACTER SET`
- `SET`

これらの句の詳細は、[セクション13.2.7「LOAD DATA ステートメント」](#)を参照してください。

`(field_name_or_user_var, ...)` は、カンマ区切りの XML フィールドまたはユーザー変数のリストです。この目的で使用されるユーザー変数の名前は、接頭辞 `@` が付いた XML ファイルのフィールドの名前と一致する必要があります。フィールド名を使用して、目的のフィールドのみを選択できます。ユーザー変数を使用して、後続の再利用のために対応するフィールド値を格納できます。

`IGNORE number LINES` または `IGNORE number ROWS` 句を指定すると、XML ファイル内の最初の `number` 行がスキップされます。これは、`LOAD DATA` ステートメントの `IGNORE ... LINES` 句に類似しています。

次に示すように作成された `person` という名前のテーブルがあるとします:

```
USE test;

CREATE TABLE person (
  person_id INT NOT NULL PRIMARY KEY,
  fname VARCHAR(40) NULL,
  lname VARCHAR(40) NULL,
  created TIMESTAMP
);
```

さらに、このテーブルが最初は空であるとします。

ここで、次に示すような内容を持つ単純な XML ファイル `person.xml` があるとします。

```
<list>
  <person person_id="1" fname="Kapek" lname="Sainnouine"/>
  <person person_id="2" fname="Sajon" lname="Rondela"/>
  <person person_id="3"><fname>Likame</fname><lname>Örrtmons</lname></person>
  <person person_id="4"><fname>Slar</fname><lname>Manlanth</lname></person>
```

```
<person><field name="person_id">5</field><field name="fname">Stoma</field>
<field name="lname">Milu</field></person>
<person><field name="person_id">6</field><field name="fname">Nirtam</field>
<field name="lname">Sklöd</field></person>
<person person_id="7"><fname>Sungam</fname><lname>Dulbåd</lname></person>
<person person_id="8" fname="Sreraf" lname="Encmelt"/>
</list>
```

例として示したこのファイルには、前に説明した許可される各 XML 形式が表されています。

`person.xml` 内のデータを `person` テーブルにインポートするには、次のステートメントを使用できます。

```
mysql> LOAD XML LOCAL INFILE 'person.xml'
-> INTO TABLE person
-> ROWS IDENTIFIED BY '<person>';
```

```
Query OK, 8 rows affected (0.00 sec)
Records: 8 Deleted: 0 Skipped: 0 Warnings: 0
```

ここでは、`person.xml` が MySQL データディレクトリ内に存在することを前提としています。このファイルが見つからない場合は、次のエラーが発生します。

```
ERROR 2 (HY000): File '/person.xml' not found (Errcode: 2)
```

`ROWS IDENTIFIED BY '<person>'` 句は、XML ファイル内の各 `<person>` 要素が、このデータがインポートされるテーブル内の各行と同等であると見なされることを示します。この場合、これは `test` データベース内の `person` テーブルです。

サーバーからの応答でわかるように、`test.person` テーブルには 8 行がインポートされました。これは、単純な `SELECT` ステートメントで確認できます。

```
mysql> SELECT * FROM person;
+-----+-----+-----+-----+
| person_id | fname | lname | created |
+-----+-----+-----+-----+
| 1 | Kapek | Sainnouine | 2007-07-13 16:18:47 |
| 2 | Sajon | Rondela | 2007-07-13 16:18:47 |
| 3 | Likame | Örrtmons | 2007-07-13 16:18:47 |
| 4 | Slar | Manlanth | 2007-07-13 16:18:47 |
| 5 | Stoma | Nilu | 2007-07-13 16:18:47 |
| 6 | Nirtam | Sklöd | 2007-07-13 16:18:47 |
| 7 | Sungam | Dulbåd | 2007-07-13 16:18:47 |
| 8 | Sreraf | Encmelt | 2007-07-13 16:18:47 |
+-----+-----+-----+-----+
8 rows in set (0.00 sec)
```

これは、このセクションで前述したように、許可された 3 つの XML 形式のいずれかまたはすべてが単一のファイルに表示され、`LOAD XML` を使用して読み取ることができることを示しています。

インポート操作の逆を示します。つまり、次に示すように、MySQL テーブルデータを XML ファイルにダンプするには、`mysql` クライアントをシステムシェルから使用します：

```
shell> mysql --xml -e "SELECT * FROM test.person" > person-dump.xml
shell> cat person-dump.xml
<?xml version="1.0"?>

<resultset statement="SELECT * FROM test.person" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
    <field name="person_id">1</field>
    <field name="fname">Kapek</field>
    <field name="lname">Sainnouine</field>
  </row>

  <row>
    <field name="person_id">2</field>
    <field name="fname">Sajon</field>
    <field name="lname">Rondela</field>
  </row>

  <row>
    <field name="person_id">3</field>
```

```

<field name="fname">Likema</field>
<field name="lname">Örrtmons</field>
</row>

<row>
<field name="person_id">4</field>
<field name="fname">Slar</field>
<field name="lname">Manlanth</field>
</row>

<row>
<field name="person_id">5</field>
<field name="fname">Stoma</field>
<field name="lname">Nilu</field>
</row>

<row>
<field name="person_id">6</field>
<field name="fname">Nirtam</field>
<field name="lname">Sklöd</field>
</row>

<row>
<field name="person_id">7</field>
<field name="fname">Sungam</field>
<field name="lname">Dulbåd</field>
</row>

<row>
<field name="person_id">8</field>
<field name="fname">Sreraf</field>
<field name="lname">Encmelt</field>
</row>
</resultset>

```

注記

--xml オプションを指定すると、mysql クライアントは、その出力として XML 形式を使用します。-e オプションを指定すると、クライアントはそのオプションの直後にある SQL ステートメントを実行します。セクション4.5.1「mysql — MySQL コマンドラインクライアント」を参照してください。

ダンプが有効であることを確認するには、次のように、person テーブルのコピーを作成し、ダンプファイルを新しいテーブルにインポートします:

```

mysql> USE test;
mysql> CREATE TABLE person2 LIKE person;
Query OK, 0 rows affected (0.00 sec)

mysql> LOAD XML LOCAL INFILE 'person-dump.xml'
-> INTO TABLE person2;
Query OK, 8 rows affected (0.01 sec)
Records: 8 Deleted: 0 Skipped: 0 Warnings: 0

mysql> SELECT * FROM person2;
+-----+-----+-----+-----+
| person_id | fname | lname | created |
+-----+-----+-----+-----+
| 1 | Kapek | Sainnouine | 2007-07-13 16:18:47 |
| 2 | Sajon | Rondela | 2007-07-13 16:18:47 |
| 3 | Likema | Örrtmons | 2007-07-13 16:18:47 |
| 4 | Slar | Manlanth | 2007-07-13 16:18:47 |
| 5 | Stoma | Nilu | 2007-07-13 16:18:47 |
| 6 | Nirtam | Sklöd | 2007-07-13 16:18:47 |
| 7 | Sungam | Dulbåd | 2007-07-13 16:18:47 |
| 8 | Sreraf | Encmelt | 2007-07-13 16:18:47 |
+-----+-----+-----+-----+
8 rows in set (0.00 sec)

```

XML ファイルのすべてのフィールドが、対応するテーブルのカラムと一致する必要はありません。対応するカラムがないフィールドはスキップされます。これを確認するには、まず person2 テーブルを空にして created カラムを削除してから、前に使用したのと同じ LOAD XML ステートメントを次のように使用します:

```
mysql> TRUNCATE person2;
Query OK, 8 rows affected (0.26 sec)

mysql> ALTER TABLE person2 DROP COLUMN created;
Query OK, 0 rows affected (0.52 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> SHOW CREATE TABLE person2\G
***** 1. row *****
      Table: person2
Create Table: CREATE TABLE `person2` (
  `person_id` int(11) NOT NULL,
  `fname` varchar(40) DEFAULT NULL,
  `lname` varchar(40) DEFAULT NULL,
  PRIMARY KEY (`person_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)

mysql> LOAD XML LOCAL INFILE 'person-dump.xml'
-> INTO TABLE person2;
Query OK, 8 rows affected (0.01 sec)
Records: 8 Deleted: 0 Skipped: 0 Warnings: 0

mysql> SELECT * FROM person2;
+-----+-----+-----+
| person_id | fname | lname |
+-----+-----+-----+
| 1 | Kapek | Sainnouine |
| 2 | Sajon | Rondela |
| 3 | Likema | Örrtmons |
| 4 | Slar | Manlanth |
| 5 | Stoma | Nilu |
| 6 | Nirtam | Sklöd |
| 7 | Sungam | Dulbåd |
| 8 | Sreraf | Encmelt |
+-----+-----+-----+
8 rows in set (0.00 sec)
```

XML ファイルの各行でフィールドが指定される順序は、LOAD XML の操作には影響しません。フィールドの順序は行によって異なる場合があり、テーブル内の対応するカラムと同じ順序である必要はありません。

前述のように、XML の 1 つ以上のフィールド (目的のフィールドのみを選択する場合) またはユーザー変数 (後で使用するために対応するフィールド値を格納する場合) の (`field_name_or_user_var, ...`) リストを使用できます。ユーザー変数は、XML ファイルの名前が XML フィールドの名前と一致しないテーブルのカラムにデータを挿入する場合に特に役立ちます。これがどのように機能するかを確認するには、まず、`person` テーブルの構造と一致するが、カラムの名前が異なる `individual` という名前のテーブルを作成します:

```
mysql> CREATE TABLE individual (
-> individual_id INT NOT NULL PRIMARY KEY,
-> name1 VARCHAR(40) NULL,
-> name2 VARCHAR(40) NULL,
-> made TIMESTAMP
-> );
Query OK, 0 rows affected (0.42 sec)
```

この場合、フィールド名とカラム名が一致しないため、XML ファイルをテーブルに直接ロードすることはできません:

```
mysql> LOAD XML INFILE './bin/person-dump.xml' INTO TABLE test.individual;
ERROR 1263 (22004): Column set to default value; NULL supplied to NOT NULL column 'individual_id' at row 1
```

これは、MySQL サーバーがターゲットテーブルのカラム名と一致するフィールド名を検索するために発生します。この問題を回避するには、フィールド値をユーザー変数に選択し、SET を使用してターゲットテーブルのカラムをそれらの変数の値と同じに設定します。次に示すように、これらの操作は両方とも単一のステートメントで実行できます:

```
mysql> LOAD XML INFILE './bin/person-dump.xml'
-> INTO TABLE test.individual (@person_id, @fname, @lname, @created)
-> SET individual_id=@person_id, name1=@fname, name2=@lname, made=@created;
Query OK, 8 rows affected (0.05 sec)
Records: 8 Deleted: 0 Skipped: 0 Warnings: 0
```



```
mysql> SELECT * FROM individual;
+-----+-----+-----+-----+
| individual_id | name1 | name2 | made |
+-----+-----+-----+-----+
| 1 | Kapek | Sainnouine | 2007-07-13 16:18:47 |
| 2 | Sajon | Rondela | 2007-07-13 16:18:47 |
| 3 | Likema | Örrtmons | 2007-07-13 16:18:47 |
| 4 | Slar | Manlanth | 2007-07-13 16:18:47 |
| 5 | Stoma | Nilu | 2007-07-13 16:18:47 |
| 6 | Nirtam | Sklöd | 2007-07-13 16:18:47 |
| 7 | Sungam | Dulbåd | 2007-07-13 16:18:47 |
| 8 | Srraf | Encmelt | 2007-07-13 16:18:47 |
+-----+-----+-----+-----+
8 rows in set (0.00 sec)
```

ユーザー変数の名前は、XML ファイルの対応するフィールドの名前と一致し、変数であることを示す必須の@接頭辞が追加されている必要があります。ユーザー変数は、対応するフィールドと同じ順序でリストまたは割り当てる必要はありません。

ROWS IDENTIFIED BY '<tagname>' 句を使用すると、同じ XML ファイルのデータを定義の異なるデータベーステーブルにインポートできます。この例では、次の XML を含む `address.xml` という名前のファイルがあるとします。

```
<?xml version="1.0"?>
<list>
  <person person_id="1">
    <fname>Robert</fname>
    <lname>Jones</lname>
    <address address_id="1" street="Mill Creek Road" zip="45365" city="Sidney"/>
    <address address_id="2" street="Main Street" zip="28681" city="Taylorsville"/>
  </person>
  <person person_id="2">
    <fname>Mary</fname>
    <lname>Smith</lname>
    <address address_id="3" street="River Road" zip="80239" city="Denver"/>
    <!-- <address address_id="4" street="North Street" zip="37920" city="Knoxville"/> -->
  </person>
</list>
```

ここでも、このセクションで前に定義された `test.person` テーブルを使用できます。テーブルの既存のすべてのレコードをクリアしたあと、次に示すようにその構造を表示します。

```
mysql< TRUNCATE person;
Query OK, 0 rows affected (0.04 sec)

mysql< SHOW CREATE TABLE person\G
***** 1. row *****
      Table: person
Create Table: CREATE TABLE `person` (
  `person_id` int(11) NOT NULL,
  `fname` varchar(40) DEFAULT NULL,
  `lname` varchar(40) DEFAULT NULL,
  `created` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`person_id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4
1 row in set (0.00 sec)
```

次に、次の **CREATE TABLE** ステートメントを使用して、`test` データベース内に `address` テーブルを作成します。

```
CREATE TABLE address (
  address_id INT NOT NULL PRIMARY KEY,
  person_id INT NULL,
  street VARCHAR(40) NULL,
  zip INT NULL,
  city VARCHAR(40) NULL,
  created TIMESTAMP
);
```

XML ファイルのデータを `person` テーブルにインポートするには、次に示すように、行が `<person>` 要素で指定されるように指定する次の **LOAD XML** ステートメントを実行します。

```
mysql> LOAD XML LOCAL INFILE 'address.xml'
-> INTO TABLE person
-> ROWS IDENTIFIED BY '<person>';
Query OK, 2 rows affected (0.00 sec)
Records: 2 Deleted: 0 Skipped: 0 Warnings: 0
```

SELECT ステートメントを使用して、レコードがインポートされたことを確認できます。

```
mysql> SELECT * FROM person;
+-----+-----+-----+-----+
| person_id | fname | lname | created |
+-----+-----+-----+-----+
| 1 | Robert | Jones | 2007-07-24 17:37:06 |
| 2 | Mary | Smith | 2007-07-24 17:37:06 |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

XML ファイル内の **<address>** 要素は、**person** テーブル内に対応するカラムがないためスキップされます。

<address> 要素のデータを **address** テーブルにインポートするには、次に示す **LOAD XML** ステートメントを使用します。

```
mysql> LOAD XML LOCAL INFILE 'address.xml'
-> INTO TABLE address
-> ROWS IDENTIFIED BY '<address>';
Query OK, 3 rows affected (0.00 sec)
Records: 3 Deleted: 0 Skipped: 0 Warnings: 0
```

次のような **SELECT** ステートメントを使用して、データがインポートされたこと確認できます。

```
mysql> SELECT * FROM address;
+-----+-----+-----+-----+-----+-----+
| address_id | person_id | street | zip | city | created |
+-----+-----+-----+-----+-----+-----+
| 1 | 1 | Mill Creek Road | 45365 | Sidney | 2007-07-24 17:37:37 |
| 2 | 1 | Main Street | 28681 | Taylorsville | 2007-07-24 17:37:37 |
| 3 | 2 | River Road | 80239 | Denver | 2007-07-24 17:37:37 |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

<address> 要素のデータのうち、XML コメントで囲まれているものはインポートされません。ただし、**address** テーブルには **person_id** カラムがあるため、各 **<address>** に対する親の **<person>** 要素の **person_id** 属性の値は **address** テーブルにインポートされます。

セキュリティ上の考慮事項。 **LOAD DATA** ステートメントと同様に、クライアントホストからサーバーホストへの XML ファイルの転送は MySQL サーバーによって開始されます。理論上は、**LOAD XML** ステートメント内でクライアントによって指定されたファイルではなく、サーバーが選択したファイルを転送するようにクライアントプログラムに指示する、パッチが適用されたサーバーを構築できます。そのようなサーバーは、クライアントユーザーが読み取りアクセス権を持つクライアントホスト上のすべてのファイルにアクセスできます。

Web 環境では、クライアントは通常、Web サーバーから MySQL に接続します。MySQL サーバーに対して任意のコマンドを実行できるユーザーは、**LOAD XML LOCAL** を使用して、Web サーバープロセスが読み取りアクセス権を持つどのファイルでも読み取ることができます。この環境では、そのクライアントは MySQL サーバーに対して、Web サーバーに接続するユーザーによって実行されているリモートプログラムではなく、実際に Web サーバーです。

--local-infile=0 または **--local-infile=OFF** を使用してサーバーを起動することによって、クライアントからの XML ファイルのロードを無効にすることができます。このオプションはまた、クライアントセッションの間中は **LOAD XML** を無効にするように **mysql** クライアントを起動する場合にも使用できます。

クライアントがサーバーから XML ファイルをロードしないようにするために、対応する MySQL ユーザーアカウントには **FILE** 権限を付与しないようにするか、またはクライアントユーザーアカウントがすでにこの権限を持っている場合は取り消してください。

重要

FILE 権限を取り消す (または最初に付与しない) と、ユーザーは **LOAD XML** ステートメント (および **LOAD_FILE()** 関数) の実行のみを保持し、**LOAD XML LOCAL** の実行を妨げるこ

とはありません。このステートメントを禁止するには、サーバーまたはクライアントを `--local-infile=OFF` で起動する必要があります。

つまり、`FILE` 権限は、そのクライアントがサーバー上のファイルを読み取れるかどうかにのみ影響を与えます。そのクライアントがローカルファイルシステム上のファイルを読み取れるかどうかには関係しません。

13.2.9 REPLACE ステートメント

```
REPLACE [LOW_PRIORITY | DELAYED]
[INTO] tbl_name
[PARTITION (partition_name [, partition_name] ...)]
[(col_name [, col_name] ...)]
{(VALUES | VALUE) (value_list) [, (value_list)] ...
 |
 VALUES row_constructor_list
}

REPLACE [LOW_PRIORITY | DELAYED]
[INTO] tbl_name
[PARTITION (partition_name [, partition_name] ...)]
SET assignment_list

REPLACE [LOW_PRIORITY | DELAYED]
[INTO] tbl_name
[PARTITION (partition_name [, partition_name] ...)]
[(col_name [, col_name] ...)]
{SELECT ... | TABLE table_name}

value:
{expr | DEFAULT}

value_list:
value [, value] ...

row_constructor_list:
ROW(value_list)[, ROW(value_list)][, ...]

assignment:
col_name = value

assignment_list:
assignment [, assignment] ...
```

`REPLACE` は、`INSERT` とまったく同じように機能します。ただし、テーブル内の古い行に、`PRIMARY KEY` または `UNIQUE` インデックスに関して新しい行と同じ値が含まれている場合、その古い行は新しい行が挿入される前に削除されます。 [セクション13.2.6「INSERT ステートメント」](#) を参照してください。

`REPLACE` は、SQL 標準への MySQL 拡張です。これは挿入を行うか、または削除と挿入を行います。標準 SQL への別の MySQL 拡張 (挿入または更新を行います) については、 [セクション13.2.6.2「INSERT ... ON DUPLICATE KEY UPDATE ステートメント」](#) を参照してください。

`DELAYED` の挿入および置換は、MySQL 5.6 で非推奨になりました。MySQL 8.0 では、`DELAYED` はサポートされません。サーバーは `DELAYED` キーワードを認識しますが、無視し、置換を遅延なし置換として処理し、`ER_WARN_LEGACY_SYNTAX_CONVERTED` 警告を生成: `REPLACE DELAYED` はサポートされなくなりました。ステートメントは `REPLACE` に変換されました。 `DELAYED` キーワードは、将来のリリースで削除される予定です。

注記

`REPLACE` は、テーブルに `PRIMARY KEY` または `UNIQUE` インデックスがある場合のみ意味を持ちます。それ以外の場合は、新しい行が別の行と重複するかどうかを判断するために使用されるインデックスがないため、`INSERT` と同等になります。

すべてのカラムの値が `REPLACE` ステートメントで指定されている値から取得されます。カラムがない場合は、`INSERT` での処理と同様に、そのカラムはそのデフォルト値に設定されます。現在の行の値を参照し、それを新しい行で使用することはできません。 `SET col_name = col_name + 1` などの代入を使用した場合、右側にあるカラム

名への参照は `DEFAULT(col_name)` として処理されるため、この代入は `SET col_name = DEFAULT(col_name) + 1` と同等です。

MySQL 8.0.19 以降では、`REPLACE` が `VALUES ROW()` を使用して挿入を試みるカラム値を指定できます。

`REPLACE` を使用するには、このテーブルに対する `INSERT` 権限と `DELETE` 権限の両方が必要です。

生成されたカラムが明示的に置換される場合、許可される値は `DEFAULT` のみです。生成されるカラムの詳細は、[セクション13.1.20.8「CREATE TABLE および生成されるカラム」](#) を参照してください。

`REPLACE` では、パーティションまたはサブパーティション (あるいはその両方) のカンマ区切り名のリストを含む `PARTITION` キーワードを使用した明示的なパーティション選択がサポートされています。 `INSERT` と同様に、これらのいずれかのパーティションまたはサブパーティションに新しい行を挿入できない場合、`REPLACE` ステートメントは `Found a row not matching the given partition set.` エラーで失敗します。詳細および例については、[セクション24.5「パーティション選択」](#) を参照してください。

`REPLACE` は、影響を受けた行数を示す数を返します。これは、削除された行と挿入された行の合計です。この数が単一行の `REPLACE` に対して 1 である場合は、行が挿入され、削除された行はありませんでした。この数が 1 より大きい場合は、新しい行が挿入される前に 1 つ以上の古い行が削除されました。テーブルに複数の一意のインデックスが存在するときに、新しい行が異なる一意のインデックス内の別の古い行の値を複製した場合は、単一行が複数の古い行を置き換えることがあります。

影響を受けた行数により、`REPLACE` が行を追加しただけか、または行の置き換えも行なったかを判定することが容易になります。その数が 1 (追加した) か、またはそれより大きい (置き換えた) かをチェックします。

C API を使用している場合は、`mysql_affected_rows()` 関数を使用して、影響を受けた行数を取得できます。

テーブルに置換して、サブクエリーと同じテーブルから選択することはできません。

MySQL は、`REPLACE` (および `LOAD DATA ... REPLACE`) に次のアルゴリズムを使用します。

1. テーブルへの新しい行の挿入を試みます
2. 主キーまたは一意のインデックスに関する重複キーエラーが発生したために挿入が失敗している間、次のことを行います。
 - a. 重複キー値を含む競合している行をテーブルから削除します
 - b. テーブルへの新しい行の挿入を再試行します

重複キーエラーが発生した場合、ストレージエンジンが削除と挿入ではなく、更新として `REPLACE` を実行する可能性があります。そのセマンティクスは同じです。ストレージエンジンが `Handler_xxx` ステータス変数を増分する方法が異なる可能性がある以外、ユーザーに見える影響はありません。

`REPLACE ... SELECT` ステートメントの結果は `SELECT` からの行の順序に依存し、この順序は常に保証されるわけではないため、ソースとスレーブが相違するようにこれらのステートメントをロギングするときに可能です。このため、`REPLACE ... SELECT` ステートメントにはステートメントベースレプリケーションに対して安全でないフラグが付けられます。このようなステートメントは、ステートメントベースモードの使用時にエラーログに警告を生成し、`MIXED` モードの使用時に行ベースの形式を使用してバイナリログに書き込まれます。[セクション17.2.1.1「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」](#) も参照してください。

MySQL 8.0.19 以降では、`INSERT` と同様に、`TABLE` および `SELECT with REPLACE` がサポートされます。詳細および例については、[セクション13.2.6.1「INSERT ... SELECT ステートメント」](#) を参照してください。

パーティション化されていない既存のテーブルをパーティション化に対応するように変更しているときや、すでにパーティション化されたテーブルのパーティション化を変更しているときに、そのテーブルの主キーの変更を検討する可能性があります ([セクション24.6.1「パーティショニングキー、主キー、および一意キー」](#) を参照してください)。これを行うと、パーティション化されていないテーブルの主キーを変更した場合と同様に、`REPLACE` ステートメントの結果が影響を受ける可能性があります。次の `CREATE TABLE` ステートメントによって作成されたテーブルを考えてみます。

```
CREATE TABLE test (  
  id INT UNSIGNED NOT NULL AUTO_INCREMENT,
```

```
data VARCHAR(64) DEFAULT NULL,
ts TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
PRIMARY KEY (id)
);
```

このテーブルを作成し、mysql クライアントに示されているステートメントを実行すると、結果は次のようになります。

```
mysql> REPLACE INTO test VALUES (1, 'Old', '2014-08-20 18:47:00');
Query OK, 1 row affected (0.04 sec)
```

```
mysql> REPLACE INTO test VALUES (1, 'New', '2014-08-20 18:47:42');
Query OK, 2 rows affected (0.04 sec)
```

```
mysql> SELECT * FROM test;
+----+-----+-----+
| id | data | ts          |
+----+-----+-----+
| 1 | New  | 2014-08-20 18:47:42 |
+----+-----+-----+
1 row in set (0.00 sec)
```

ここで、次に示すように (強調表示されたテキスト) 主キーが 2 つのカラムになっている点を除き、最初のテーブルとほぼ同一の 2 番目のテーブルを作成します。

```
CREATE TABLE test2 (
  id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  data VARCHAR(64) DEFAULT NULL,
  ts TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (id, ts)
);
```

元の `test` テーブルに対して実行したのと同じ 2 つの `REPLACE` ステートメントを `test2` に対して実行すると、異なる結果が得られます。

```
mysql> REPLACE INTO test2 VALUES (1, 'Old', '2014-08-20 18:47:00');
Query OK, 1 row affected (0.05 sec)
```

```
mysql> REPLACE INTO test2 VALUES (1, 'New', '2014-08-20 18:47:42');
Query OK, 1 row affected (0.06 sec)
```

```
mysql> SELECT * FROM test2;
+----+-----+-----+
| id | data | ts          |
+----+-----+-----+
| 1 | Old  | 2014-08-20 18:47:00 |
| 1 | New  | 2014-08-20 18:47:42 |
+----+-----+-----+
2 rows in set (0.00 sec)
```

これは、`test2` に対して実行した場合は `id` カラムと `ts` カラムの両方の値が、置き換えられる行に対する既存の行の値に一致している必要があり、そうでないと行が挿入されるためです。

13.2.10 SELECT ステートメント

```
SELECT
[ALL | DISTINCT | DISTINCTROW ]
[HIGH_PRIORITY]
[STRAIGHT_JOIN]
[SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
[SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
select_expr [, select_expr] ...
[into_option]
[FROM table_references
  [PARTITION partition_list]]
[WHERE where_condition]
[GROUP BY {col_name | expr | position}, ... [WITH ROLLUP]]
[HAVING where_condition]
[WINDOW window_name AS (window_spec)
  [, window_name AS (window_spec)] ...]
[ORDER BY {col_name | expr | position}]
```

```
[ASC | DESC], ... [WITH ROLLUP]
[LIMIT {[offset,] row_count | row_count OFFSET offset}]
[into_option]
[FOR {UPDATE | SHARE}
  [OF tbl_name [, tbl_name] ...]
  [NOWAIT | SKIP LOCKED]
  | LOCK IN SHARE MODE]
[into_option]

into_option: {
  INTO OUTFILE 'file_name'
  [CHARACTER SET charset_name]
  export_options
  | INTO DUMPFILE 'file_name'
  | INTO var_name [, var_name] ...
}
```

SELECT は、1 つ以上のテーブルから選択された行を取得するために使用され、**UNION** ステートメントとサブクエリーを含めることができます。 [セクション13.2.10.3「UNION 句」](#) および [セクション13.2.11「サブクエリー」](#) を参照してください。 **SELECT** ステートメントは、**WITH** 句で始まり、**SELECT** 内でアクセス可能な共通テーブル式を定義できます。 [セクション13.2.15「WITH \(共通テーブル式\)」](#) を参照してください。

SELECT ステートメントのもっとも一般的に使用される句は次のとおりです。

- 各 **select_expr** は、取得するカラムを示します。少なくとも 1 つの **select_expr** が存在する必要があります。
- **table_references** は、行を取得する 1 つまたは複数のテーブルを示します。その構文については、[セクション13.2.10.2「JOIN 句」](#) で説明されています。
- **SELECT** では、**table_reference** のテーブル名の後にパーティションまたはサブパーティション (あるいはその両方) のリストを含む **PARTITION** を使用した明示的なパーティション選択がサポートされています ([セクション13.2.10.2「JOIN 句」](#) を参照)。この場合、行はリストされているパーティションからのみ選択され、テーブルのほかのパーティションはすべて無視されます。詳細および例については、[セクション24.5「パーティション選択」](#) を参照してください。
- **WHERE** 句 (指定されている場合) は、選択されるために行が満たす必要のある 1 つまたは複数の条件を示します。**where_condition** は、選択される各行に対して true に評価される式です。**WHERE** 句がない場合、このステートメントはすべての行を選択します。

WHERE 式では、集計 (グループ) 関数を除き、MySQL でサポートされている任意の関数および演算子を使用できます。 [セクション9.5「式」](#) および [第12章「関数と演算子」](#) を参照してください。

SELECT を使用して、どのテーブルも参照せずに計算された行を取得することもできます。

例:

```
mysql> SELECT 1 + 1;
-> 2
```

テーブルが参照されない状況では、ダミーのテーブル名として **DUAL** を指定することが許可されます。

```
mysql> SELECT 1 + 1 FROM DUAL;
-> 2
```

DUAL は純粋に、すべての **SELECT** ステートメントに **FROM** や、場合によってはその他の句が存在することを要求するユーザーの便宜のために用意されています。MySQL は、これらの句を無視する可能性があります。MySQL では、テーブルが参照されない場合でも **FROM DUAL** は必要ありません。

一般に、使用される句は、正確に構文の説明で示されている順序で指定する必要があります。たとえば、**HAVING** 句は、すべての **GROUP BY** 句のあとで、かつすべての **ORDER BY** 句の前にある必要があります。**INTO** 句は、構文の説明で示されている任意の位置に指定できますが、特定のステートメント内に指定できるのは一度のみで、複数の位置には指定できません。**INTO** の詳細は、[セクション13.2.10.1「SELECT ... INTO ステートメント」](#) を参照してください。

select_expr 項のリストは、どのカラムを取得するかを示す選択リストで構成されています。これらの項はカラムや式を指定するか、または * の短縮形を使用できます。

- 1つの修飾されていない * のみから成る選択リストは、すべてのテーブルのすべてのカラムを選択するための短縮形として使用できます。

```
SELECT * FROM t1 INNER JOIN t2 ...
```

- `tbl_name.*` は、指定されたテーブルのすべてのカラムを選択するための修飾された短縮形として使用できます。

```
SELECT t1.*, t2.* FROM t1 INNER JOIN t2 ...
```

- テーブルに非表示カラムがある場合、* および `tbl_name.*` には非表示カラムは含まれません。非表示カラムを含めるには、明示的に参照する必要があります。
- 修飾されていない * を選択リスト内のほかの項目とともに使用すると、解析エラーが生成される可能性があります。この問題を回避するには、修飾された `tbl_name.*` 参照を使用します。:

```
SELECT AVG(score), t1.* FROM t1 ...
```

次のリストは、その他の `SELECT` 句に関する追加情報を示しています。

- `AS alias_name` を使用して、`select_expr` にエイリアスを指定できます。エイリアスは式のカラム名として使用され、`GROUP BY`、`ORDER BY`、または `HAVING` 句で使用できます。例:

```
SELECT CONCAT(last_name,', 'first_name) AS full_name
FROM mytable ORDER BY full_name;
```

`select_expr` にエイリアスとして識別子を指定する場合、`AS` キーワードはオプションです。前の例は、次のように記述することもできました。

```
SELECT CONCAT(last_name,', 'first_name) full_name
FROM mytable ORDER BY full_name;
```

ただし、`AS` はオプションであるため、2つの `select_expr` 式の間のカンマを忘れると、軽微な問題が発生する可能性があります。MySQL は、2番目の式をエイリアスとして解釈します。たとえば、次のステートメントでは、`columnb` はエイリアスとして処理されます。

```
SELECT columna columnb FROM mytable;
```

このため、カラムのエイリアスを指定するときは `AS` を明示的に使用するようをお勧めします。

`WHERE` 句が実行されるときはまだカラム値が決定されていない可能性があるため、`WHERE` 句内でカラムのエイリアスを参照することは許可されません。[セクションB.3.4.4「カラムエイリアスに関する問題」](#)を参照してください。

- `FROM table_references` 句は、行を取得する1つまたは複数のテーブルを示します。複数のテーブルを指定すると、結合が実行されます。結合構文については、[セクション13.2.10.2「JOIN 句」](#)を参照してください。指定されたテーブルごとに、オプションでエイリアスを指定できます。

```
tbl_name [[AS] alias] [index_hint]
```

インデックスヒントを使用すると、クエリー処理中にインデックスを選択する方法に関する情報がオプティマイザに提供されます。これらのヒントを指定するための構文については、[セクション8.9.4「インデックスヒント」](#)を参照してください。

代替の方法として `SET max_seeks_for_key=value` を使用して、MySQL にテーブルスキャンの代わりにキースキャンを強制的に実行させることができます。[セクション5.1.8「サーバーシステム変数」](#)を参照してください。

- データベースを明示的に指定するために、デフォルトデータベース内でテーブルを `tbl_name` または `db_name.tbl_name` として参照できます。カラムを `col_name`、`tbl_name.col_name` または `db_name.tbl_name.col_name` として参照できます。参照があいまいにならないかぎり、カラム参照のために `tbl_name` または `db_name.tbl_name` プリフィクスを指定する必要はありません。より明示的なカラム参照形式を必要とするあいまいさの例については、[セクション9.2.2「識別子の修飾子」](#)を参照してください。
- テーブル参照は、`tbl_name AS alias_name` または `tbl_name alias_name` を使用してエイリアス設定できます。次のステートメントは同等です。

```
SELECT t1.name, t2.salary FROM employee AS t1, info AS t2
```

```
WHERE t1.name = t2.name;
```

```
SELECT t1.name, t2.salary FROM employee t1, info t2
WHERE t1.name = t2.name;
```

- カラム名、カラムのエイリアス、またはカラム位置を使用して、出力のために選択されたカラムを **ORDER BY** および **GROUP BY** 句で参照できます。カラム位置は整数であり、1 から始まります。

```
SELECT college, region, seed FROM tournament
ORDER BY region, seed;
```

```
SELECT college, region AS r, seed AS s FROM tournament
ORDER BY r, s;
```

```
SELECT college, region, seed FROM tournament
ORDER BY 2, 3;
```

逆の順序でソートするには、ソートに使用する **ORDER BY** 句内のカラムの名前に **DESC** (降順) キーワードを追加します。デフォルトは昇順です。これは、**ASC** キーワードを使用して明示的に指定できます。

ORDER BY がカッコで囲まれたクエリー式内で発生し、外部クエリーにも適用される場合、結果は未定義であり、将来のバージョンの MySQL で変更される可能性があります。

カラム位置の使用は、この構文が SQL 標準から削除されたため非推奨です。

- MySQL 8.0.13 より前では、MySQL は **GROUP BY** カラムの明示的な **ASC** または **DESC** 指定子を許可する非標準の構文拡張をサポートしていました。MySQL 8.0.12 以降では、グループ化関数を使用した **ORDER BY** がサポートされているため、この拡張機能を使用する必要はなくなりました。(Bug #86312、Bug #26073525) これは、**GROUP BY** の使用時に次のように任意のカラムでソートできることも意味します:

```
SELECT a, b, COUNT(c) AS t FROM test_table GROUP BY a,b ORDER BY a,t DESC;
```

MySQL 8.0.13 では、**GROUP BY** 拡張機能はサポートされなくなりました: **GROUP BY** カラムの **ASC** または **DESC** 指定子は許可されていません。

- **ORDER BY** または **GROUP BY** を使用して **SELECT** のカラムをソートする場合、**max_sort_length** システム変数で指定された初期バイト数のみを使用して値がソートされます。
- MySQL では、**GROUP BY** の使用が、**GROUP BY** 句で指定されていないフィールドの選択を許可するように拡張されています。クエリーから期待する結果が得られない場合は、[セクション12.20「集計関数」](#)にある **GROUP BY** の説明を参照してください。
- **GROUP BY** では、**WITH ROLLUP** 修飾子が許可されます。[セクション12.20.2「GROUP BY 修飾子」](#)を参照してください。

以前は、**WITH ROLLUP** 修飾子を持つクエリーで **ORDER BY** を使用することはできませんでした。この制限は、MySQL 8.0.12 の時点でなくなりました。[セクション12.20.2「GROUP BY 修飾子」](#)を参照してください。

- **HAVING** 句は、ほぼ最後 (項目がクライアントに送信される直前) に最適化なしで適用されます。(LIMIT は **HAVING** のあとに適用されます。)

SQL 標準では、**HAVING** は **GROUP BY** 句内のカラムか、または集約関数で使用されるカラムしか参照できません。ただし、MySQL ではこの動作への拡張がサポートされており、**HAVING** が **SELECT** リスト内のカラムや外側サブクエリー内のカラムを参照することも許可されます。

HAVING 句があいまいなカラムを参照している場合は、警告が発生します。次のステートメントにある **col2** は、エイリアスとカラム名の両方として使用されているため、あいまいです。

```
SELECT COUNT(col1) AS col2 FROM t GROUP BY col2 HAVING col2 = 2;
```

標準 SQL の動作の方が優先されるため、**HAVING** のカラム名が **GROUP BY** で使用されると同時に、出力カラムリスト内のエイリアスが指定されたカラムとしても使用されている場合は、**GROUP BY** カラム内のカラムが優先されます。

- **WHERE** 句に含めるべき項目には **HAVING** を使用しないでください。たとえば、次のように記述しないでください。

```
SELECT col_name FROM tbl_name HAVING col_name > 0;
```

代わりに、次のように記述してください。

```
SELECT col_name FROM tbl_name WHERE col_name > 0;
```

- **HAVING** 句は、**WHERE** 句が参照できない集約関数を参照できます。

```
SELECT user, MAX(salary) FROM users  
GROUP BY user HAVING MAX(salary) > 10;
```

(これは、一部の古いバージョンの MySQL では機能しませんでした。)

- MySQL では、重複したカラム名が許可されます。つまり、同じ名前を持つ複数の `select_expr` が存在できます。これは、標準 SQL の拡張です。MySQL では **GROUP BY** や **HAVING** が `select_expr` 値を参照することも許可されるため、これにより、あいまいさが発生する場合があります。

```
SELECT 12 AS a, a FROM t GROUP BY a;
```

このステートメントでは、どちらのカラムの名前も `a` です。グループ化のために正しいカラムが使用されるようにするために、`select_expr` ごとに異なる名前を使用してください。

- **WINDOW** 句が存在する場合は、ウィンドウ関数で参照できる名前付きウィンドウを定義します。詳細は、[セクション12.21.4「名前付きウィンドウ」](#)を参照してください。
- MySQL は、**ORDER BY** 句内の修飾されていないカラムまたはエイリアス参照を、まず `select_expr` 値、次に **FROM** 句内のテーブルのカラム内を検索することによって解決します。**GROUP BY** または **HAVING** 句の場合は、`select_expr` 値内を検索する前に **FROM** 句を検索します。(**GROUP BY** と **HAVING** について、これは、**ORDER BY**) の場合と同じルールを使用していた MySQL 5.0 より前の動作とは異なります。
- **LIMIT** 句を使用すると、**SELECT** ステートメントによって返される行数を制約できます。**LIMIT** は 1 つまたは 2 つの数値引数を受け取ります。これは、どちらも負ではない整数である必要があります。ただし、次の例外があります。
 - 準備済みステートメント内では、`?` プレースホルダマーカを使用して **LIMIT** パラメータを指定できます。
 - ストアドプログラム内では、整数値のルーチンパラメータまたはローカル変数を使用して **LIMIT** パラメータを指定できます。

引数が 2 つの場合、最初の引数は返す先頭行のオフセットを指定し、2 番目の引数は返す行の最大数を指定します。最初の行のオフセットは (1 ではなく) 0 です。

```
SELECT * FROM tbl LIMIT 5,10; # Retrieve rows 6-15
```

特定のオフセットから結果セットの最後まですべての行を取得するために、2 番目のパラメータにある程度大きい数字を使用できます。次のステートメントは、96 行目から最後の行までのすべての行を取得します。

```
SELECT * FROM tbl LIMIT 95,18446744073709551615;
```

引数が 1 つの場合、この値は、結果セットの先頭から返す行数を指定します。

```
SELECT * FROM tbl LIMIT 5; # Retrieve first 5 rows
```

つまり、`LIMIT row_count` は `LIMIT 0, row_count` と同等です。

準備済みステートメントでは、プレースホルダを使用できます。次のステートメントは、`tbl` テーブルから行を戻します:

```
SET @a=1;  
PREPARE STMT FROM 'SELECT * FROM tbl LIMIT ?';  
EXECUTE STMT USING @a;
```

次のステートメントは、`tbl` テーブルから秒から 6 行目を戻します:

```
SET @skip=1; SET @numrows=5;  
PREPARE STMT FROM 'SELECT * FROM tbl LIMIT ?, ?';
```

```
EXECUTE STMT USING @skip, @numrows;
```

PostgreSQL との互換性のために、MySQL は `LIMIT row_count OFFSET offset` 構文もサポートしています。

`LIMIT` がカッコで囲まれたクエリー式内で発生し、外部クエリーにも適用される場合、結果は未定義であり、将来のバージョンの MySQL で変更される可能性があります。

- `SELECT` の `SELECT ... INTO` 形式を使用すると、クエリー結果をファイルに書き込んだり、変数に格納したりできます。詳細は、[セクション13.2.10.1「SELECT ... INTO ステートメント」](#)を参照してください。
- `FOR UPDATE` をページロックまたは行ロックを使用するストレージエンジンとともに使用する場合、クエリーによって検査される行は、現在のトランザクションが終了するまで書き込みロックされます。

`CREATE TABLE new_table SELECT ... FROM old_table ...` などのステートメントで `SELECT` の一部として `FOR UPDATE` を使用することはできません。(それを行おうとすると、このステートメントはエラー `Can't update table 'old_table' while 'new_table' is being created` で拒否されます。)

`FOR SHARE` および `LOCK IN SHARE MODE` では、他のトランザクションが調査された行を読み取ることはできませんが、更新または削除することはできない共有ロックが設定されます。`FOR SHARE` と `LOCK IN SHARE MODE` は同等です。ただし、`FOR UPDATE` と同様に、`FOR SHARE` は `NOWAIT`、`SKIP LOCKED` および `OF tbl_name` オプションをサポートしています。`FOR SHARE` は `LOCK IN SHARE MODE` の代替機能ですが、`LOCK IN SHARE MODE` は下位互換性のために引き続き使用できます。

`NOWAIT` では、`FOR UPDATE` または `FOR SHARE` クエリーがすぐに実行され、別のトランザクションによってロックが保持されているために行ロックを取得できない場合はエラーが返されます。

`SKIP LOCKED` では、別のトランザクションによってロックされている結果セットから行を除外して、`FOR UPDATE` または `FOR SHARE` クエリーがただちに実行されます。

`NOWAIT` および `SKIP LOCKED` オプションは、ステートメントベースレプリケーションでは安全ではありません。

注記

ロックされた行をスキップするクエリーは、データの一致性のないビューを返します。したがって、`SKIP LOCKED` は一般的なトランザクション作業には適していません。ただし、複数のセッションが同じキューに類似したテーブルにアクセスする場合、ロックの競合を回避するために使用できます。

`OF tbl_name` は、`FOR UPDATE` および `FOR SHARE` クエリーを名前付きテーブルに適用します。例:

```
SELECT * FROM t1, t2 FOR SHARE OF t1 FOR UPDATE OF t2;
```

`OF tbl_name` を省略すると、クエリーロックによって参照されるすべてのテーブルがロックされます。したがって、別のロック句と組み合わせて `OF tbl_name` なしでロック句を使用すると、エラーが返されます。複数のロック句で同じテーブルを指定すると、エラーが返されます。エイリアスが `SELECT` ステートメントでテーブル名として指定されている場合、ロック句ではエイリアスのみを使用できます。`SELECT` ステートメントでエイリアスが明示的に指定されていない場合、ロック句では実際のテーブル名のみを指定できます。

`FOR UPDATE` および `FOR SHARE` の詳細は、[セクション15.7.2.4「読取りのロック」](#)を参照してください。

`NOWAIT` および `SKIP LOCKED` オプションの詳細は、[NOWAIT および SKIP LOCKED による読取り同時実行性のロック](#)を参照してください。

`SELECT` キーワードの後に、ステートメントの操作に影響を与える多数の修飾子を使用できます。

`HIGH_PRIORITY`、`STRAIGHT_JOIN` および `SQL_`以降の修飾子は、標準 SQL に対する MySQL の拡張機能です。

- `ALL` 修飾子および `DISTINCT` 修飾子は、重複する行を戻すかどうかを指定します。`ALL` (デフォルト) は、重複を含め、一致するすべての行を戻すように指定します。`DISTINCT` は、重複した行の結果セットからの削除を指定します。両方の修飾子を指定するとエラーになります。`DISTINCTROW` は `DISTINCT` のシノニムです。

MySQL 8.0.12 以降では、`WITH ROLLUP` も使用するクエリーで `DISTINCT` を使用できます。(Bug #87450、Bug #26640100)

- `HIGH_PRIORITY` では、テーブルを更新するステートメントよりも `SELECT` の優先度が高くなります。これは、非常に高速であり、かつただちに実行する必要のあるクエリーにのみ使用するようにしてください。テーブルが読

み取りに対してロックされている間に発行された `SELECT HIGH_PRIORITY` クエリーは、そのテーブルが未使用になるのを待機している更新ステートメントが存在する場合でも実行されます。これは、テーブルレベルロックのみを使用するストレージエンジン (`MyISAM`、`MEMORY`、および `MERGE`) にのみ影響を与えます。

`HIGH_PRIORITY` を、`UNION` の一部である `SELECT` ステートメントで使用することはできません。

- `STRAIGHT_JOIN` では、オプティマイザによって、`FROM` 句にリストされている順序でテーブルが結合されます。オプティマイザがテーブルを最適でない順序で結合する場合は、これを使用してクエリーを高速化できます。`STRAIGHT_JOIN` はまた、`table_references` リストでも使用できます。セクション13.2.10.2「`JOIN` 句」を参照してください。

`STRAIGHT_JOIN` は、オプティマイザが `const` または `system` テーブルとして扱うテーブルには適用されません。このようなテーブルは単一行を生成し、クエリー実行の最適化フェーズ中に読み取られます。また、そのカラムへの参照は、クエリー実行が実行される前に適切なカラム値で置き換えられます。これらのテーブルは、`EXPLAIN` によって表示されるクエリー計画の最初に表示されます。「セクション8.8.1「`EXPLAIN` によるクエリーの最適化」」を参照してください。この例外は、外部結合の `NULL` で補完された側で使用されている `const` テーブルまたは `system` テーブル (つまり、`LEFT JOIN` の右側のテーブルまたは `RIGHT JOIN` の左側のテーブル) には適用されない可能性があります。

- `SQL_BIG_RESULT` または `SQL_SMALL_RESULT` を `GROUP BY` または `DISTINCT` とともに使用して、結果セットに多数の行があるか、それぞれ小さいことをオプティマイザに伝えることができます。`SQL_BIG_RESULT` の場合、MySQL ではディスクベースの一時テーブルが直接使用され (作成されている場合)、`GROUP BY` 要素のキーを使用した一時テーブルよりソートが優先されます。`SQL_SMALL_RESULT` の場合、MySQL では、ソートを使用するかわりにインメモリー一時テーブルを使用して結果テーブルを格納します。これは、通常は必要ないはずです。
- `SQL_BUFFER_RESULT` では、結果が強制的に一時テーブルに格納されます。これは、MySQL がテーブルロックを早期に解放する場合や、結果セットをクライアントに送信するのに長い時間がかかる場合に役立ちます。この修飾子は、トップレベルの `SELECT` ステートメントにのみ使用でき、サブクエリーや `UNION` の後には使用できません。
- `SQL_CALC_FOUND_ROWS` は、`LIMIT` 句に関係なく、結果セットに存在する行数を計算するように MySQL に指示します。そのあと、行数は `SELECT FOUND_ROWS()` を使用して取得できます。セクション12.16「情報関数」を参照してください。

注記

`SQL_CALC_FOUND_ROWS` クエリー修飾子および付随する `FOUND_ROWS()` 関数は、MySQL 8.0.17 で非推奨になりました。これらは、MySQL の将来のバージョンで削除される予定です。代替戦略の詳細は、`FOUND_ROWS()` の説明を参照してください。

- `SQL_CACHE` および `SQL_NO_CACHE` 修飾子は、MySQL 8.0 より前のクエリーキャッシュで使用されてきました。クエリーキャッシュは MySQL 8.0 で削除されました。`SQL_CACHE` 修飾子も削除されました。`SQL_NO_CACHE` は非推奨であり、効果はありません。将来の MySQL リリースで削除される予定です。

13.2.10.1 SELECT ... INTO ステートメント

`SELECT` の `SELECT ... INTO` 形式を使用すると、クエリー結果を変数に格納したり、ファイルに書き込んだりできます。

- `SELECT ... INTO var_list` はカラム値を選択し、それらを変数に格納します。
- `SELECT ... INTO OUTFILE` は、選択された行をファイルに書き込みます。カラムおよび行ターミネータを指定すると、特定の出力形式を生成できます。
- `SELECT ... INTO DUMPFILE` は、単一行をファイルに形式設定なしで書き込みます。

特定の `SELECT` ステートメントには最大で1つの `INTO` 句を含めることができますが、`SELECT` 構文の説明 (セクション13.2.10「`SELECT` ステートメント」を参照) に示されているように、`INTO` は異なる位置に配置できます:

- `FROM` より前。例:

```
SELECT * INTO @myvar FROM t1;
```


- 後続のロック句の前。例:

```
SELECT * FROM t1 INTO @myvar FOR UPDATE;
```

- **SELECT** の最後。例:

```
SELECT * FROM t1 FOR UPDATE INTO @myvar;
```

ステートメントの最後の **INTO** 位置は、MySQL 8.0.20 でサポートされており、推奨位置です。ロック句の前の位置は、MySQL 8.0.20 では非推奨です。将来のバージョンの MySQL ではサポートされなくなる予定です。つまり、**FROM** の後には **INTO** が生成されますが、**SELECT** の最後には生成されません。

ネストされた **SELECT** はその結果を外側のコンテキストに返す必要があるため、このような **SELECT** では **INTO** 句を使用してはいけません。**UNION** ステートメント内での **INTO** の使用には制約もあります。[セクション 13.2.10.3「UNION 句」](#) を参照してください。

INTO var_list バリエーションの場合:

- **var_list** では、1 つ以上の変数のリストに名前を付けます。各変数には、ユーザー定義変数、ストアードプロシージャまたはストアードファンクションパラメータ、ストアードプログラムのローカル変数を指定できます。(準備された **SELECT ... INTO var_list** ステートメント内では、ユーザー定義の変数のみが許可されます。[セクション 13.6.4.2「ローカル変数のスコープと解決」](#) を参照してください。)
- 選択された値は変数に割り当てられます。変数の数がカラム数に一致している必要があります。クエリーは、単一行を返すようにしてください。クエリーが行を返さない場合は、エラーコード 1329 で警告が発生し (**No data**)、変数値は変更されないままになります。クエリーが複数の行を返す場合は、エラー 1172 が発生します (**結果が 2 行以上です**)。このステートメントが複数の行を取得する可能性がある場合は、**LIMIT 1** を使用して結果セットを単一行に制限できます。

```
SELECT id, data INTO @x, @y FROM test.t1 LIMIT 1;
```

INTO var_list は、**TABLE** ステートメントとともに使用することもできますが、次の制限があります:

- 変数の数は、テーブルのカラムの数と一致する必要があります。
- テーブルに複数の行が含まれる場合は、**LIMIT 1** を使用して結果セットを単一行に制限する必要があります。**LIMIT 1** は、**INTO** キーワードの前に指定する必要があります。

このようなステートメントの例を次に示します:

```
TABLE employees ORDER BY lname DESC LIMIT 1  
INTO @id, @fname, @lname, @hired, @separated, @job_code, @store_id;
```

単一行を一連のユーザー変数に生成する **VALUES** ステートメントから値を選択することもできます。この場合、テーブルのエイリアスを使用し、値リストの各値を変数に割り当てる必要があります。ここに示す 2 つのステートメントはそれぞれ、**SET @x=2, @y=4, @z=8** と同等です:

```
SELECT * FROM (VALUES ROW(2,4,8)) AS t INTO @x,@y,@z;
```

```
SELECT * FROM (VALUES ROW(2,4,8)) AS t(a,b,c) INTO @x,@y,@z;
```

ユーザー変数名では大/小文字は区別されません。[セクション 9.4「ユーザー定義変数」](#) を参照してください。

SELECT の **SELECT ... INTO OUTFILE 'file_name'** 形式は、選択された行をファイルに書き込みます。ファイルはサーバーホストに作成されるため、この構文を使用するには **FILE** 権限が必要です。**file_name** は既存のファイルにすることはできません。特に、**/etc/passwd** やデータベーステーブルなどのファイルが変更されるのを防ぎます。**character_set_filesystem** システム変数は、ファイル名の解釈を制御します。

SELECT ... INTO OUTFILE ステートメントは、サーバーホスト上のテキストファイルへのテーブルのダンプを有効にすることを目的としています。結果のファイルを他のホストに作成する場合、サーバーホストファイルシステム上のネットワークマップパスを使用してリモートホスト上のファイルの場所にアクセスできないかぎり、サーバーホストファイルシステムを基準にした相対パスをファイルに書き込む方法がないため、**SELECT ... INTO OUTFILE** は通常は適していません。

または、MySQL クライアントソフトウェアがリモートホストにインストールされている場合は、`mysql -e "SELECT ..." > file_name` などのクライアントコマンドを使用して、そのホストにファイルを生成できます。

`SELECT ... INTO OUTFILE` は、`LOAD DATA` を補完したものです。カラム値は、`CHARACTER SET` 句で指定されている文字セットに変換されて書き込まれます。このような句が存在しない場合、値は `binary` 文字セットを使用してダンプされます。事実上、文字セットの変換は実行されません。結果セットに複数の文字セットのカラムが含まれている場合は、出力データファイルであるため、ファイルを正しくリロードできない可能性があります。

ステートメントの `export_options` 部分の構文は、`LOAD DATA` ステートメントで使用されるものと同じ `FIELDS` 句および `LINES` 句で構成されます。デフォルト値や許容値など、`FIELDS` 句および `LINES` 句の詳細は、[セクション 13.2.7 「LOAD DATA ステートメント」](#) を参照してください。

`FIELDS ESCAPED BY` は、特殊文字を書き込む方法を制御します。`FIELDS ESCAPED BY` 文字が空でない場合、その文字は、出力上で次の文字の前に付けられるプリフィクスとして、あいまいさを避けるために必要な場合に使用されます。

- `FIELDS ESCAPED BY` 文字
- `FIELDS [OPTIONALLY] ENCLOSED BY` 文字
- `FIELDS TERMINATED BY` および `LINES TERMINATED BY` 値の最初の文字
- ASCII NUL (0 の値のバイト。エスケープ文字のあとに実際に書き込まれる文字は 0 の値のバイトではなく、ASCII の 0 です)

`FIELDS TERMINATED BY`、`ENCLOSED BY`、`ESCAPED BY`、または `LINES TERMINATED BY` 文字は、そのファイルを実際に読み戻すことができるように、エスケープする必要があります。ASCII NUL は、一部のページャーで見やすくなるようにエスケープされます。

結果のファイルは SQL 構文に準拠する必要がないため、他にエスケープする必要はありません。

`FIELDS ESCAPED BY` 文字が空である場合は、どの文字もエスケープされず、`NULL` は `\N` ではなく、`NULL` として出力されます。特に、データ内のフィールド値に今指定したリスト内のいずれかの文字が含まれている場合、空のエスケープ文字を指定することはおそらく適切な方法ではありません。

テーブルのすべてのカラムをテキストファイルにダンプする場合は、`INTO OUTFILE` を `TABLE` ステートメントとともに使用することもできます。この場合、`ORDER BY` および `LIMIT` を使用して順序付けと行数を制御できます。これらの句は、`INTO OUTFILE` の前に指定する必要があります。`TABLE ... INTO OUTFILE` と同じ `export_options` がサポートされており、ファイルシステムへの書き込みには同じ制限があります。このようなステートメントの例を次に示します：

```
TABLE employees ORDER BY lname LIMIT 1000
INTO OUTFILE '/tmp/employee_data_1.txt'
FIELDS TERMINATED BY ';' OPTIONALLY ENCLOSED BY '"', ESCAPED BY '\
LINES TERMINATED BY '\n';
```

`SELECT ... INTO OUTFILE` を `VALUES` ステートメントとともに使用して、値をファイルに直接書き込むこともできます。次に例を示します：

```
SELECT * FROM (VALUES ROW(1,2,3),ROW(4,5,6),ROW(7,8,9)) AS t
INTO OUTFILE '/tmp/select-values.txt';
```

テーブルのエイリアスを使用する必要があります。カラムのエイリアスもサポートされており、オプションで目的のカラムからのみ値を書き込むために使用できます。`SELECT ... INTO OUTFILE` でサポートされているエクスポートオプションのいずれかまたはすべてを使用して、出力をファイルにフォーマットすることもできます。

多くのプログラムで使用されているカンマ区切り値 (CSV) 形式のファイルを生成する例を次に示します。

```
SELECT a,b,a+b INTO OUTFILE '/tmp/result.txt'
FIELDS TERMINATED BY ';' OPTIONALLY ENCLOSED BY '"
LINES TERMINATED BY '\n'
FROM test_table;
```

INTO OUTFILE の代わりに INTO DUMPFILE を使用した場合、MySQL はエスケープ処理を実行することなく、カラムや行の終了のない 1 行のみをファイルに書き込みます。これは、BLOB 値を選択してファイルに格納する場合に便利です。

TABLE は、INTO DUMPFILE もサポートしています。テーブルに複数の行が含まれている場合は、LIMIT 1 を使用して出力を単一行に制限する必要もあります。INTO DUMPFILE は、SELECT * FROM (VALUES ROW()([...]) AS table_alias [LIMIT 1]) とともに使用することもできます。セクション13.2.14「VALUES ステートメント」を参照してください。

注記

INTO OUTFILE または INTO DUMPFILE によって作成されたファイルは、アカウント `mysqld` が実行されているオペレーティングシステムユーザーによって所有されます。(これらの理由のために、`mysqld` を `root` としては決して実行しないでください。) MySQL 8.0.17 では、ファイル作成の `umask` は 0640 です。ファイルの内容を操作するには、十分なアクセス権限が必要です。MySQL 8.0.17 より前では、`umask` は 0666 で、ファイルはサーバーホスト上のすべてのユーザーによって書き込み可能です。

`secure_file_priv` システム変数が空以外のディレクトリ名に設定されている場合、書き込まれるファイルはそのディレクトリ内に存在する必要があります。

イベントスケジューラによって実行されるイベントの一部として実行された SELECT ... INTO ステートメントのコンテキストでは、診断メッセージ (エラーだけでなく、警告も含まれます) がエラーログに (Windows ではアプリケーションイベントログにも) 書き込まれます。詳細は、セクション25.4.5「イベントスケジューラのステータス」を参照してください。

MySQL 8.0.22 の時点では、SELECT INTO OUTFILE および SELECT INTO DUMPFILE によって書き込まれた出力ファイルの定期的な同期がサポートされており、そのバージョンで導入された `select_into_disk_sync` サーバースystem変数を設定することで有効になります。出力バッファサイズおよびオプションの遅延は、それぞれ `select_into_buffer_size` および `select_into_disk_sync_delay` を使用して設定できます。詳細は、これらのシステム変数の説明を参照してください。

13.2.10.2 JOIN 句

MySQL では、SELECT ステートメントおよび複数テーブルの DELETE ステートメントと UPDATE ステートメントの `table_references` 部分について、次の JOIN 構文がサポートされています:

```
table_references:
  escaped_table_reference [, escaped_table_reference] ...

escaped_table_reference: {
  table_reference
  | { OJ table_reference }
}

table_reference: {
  table_factor
  | joined_table
}

table_factor: {
  tbl_name [PARTITION (partition_names)]
  [[AS] alias] [index_hint_list]
  | [LATERAL] table_subquery [AS] alias [(col_list)]
  | ( table_references )
}

joined_table: {
  table_reference [(INNER | CROSS) JOIN | STRAIGHT_JOIN] table_factor [join_specification]
  | table_reference {LEFT|RIGHT} [OUTER] JOIN table_reference join_specification
  | table_reference NATURAL [(INNER | {LEFT|RIGHT} [OUTER])] JOIN table_factor
}

join_specification: {
  ON search_condition
  | USING (join_column_list)
}
```

```

join_column_list:
  column_name [, column_name] ...

index_hint_list:
  index_hint [, index_hint] ...

index_hint: {
  USE {INDEX|KEY}
  [FOR {JOIN|ORDER BY|GROUP BY}] ([index_list])
| {IGNORE|FORCE} {INDEX|KEY}
  [FOR {JOIN|ORDER BY|GROUP BY}] (index_list)
}

index_list:
  index_name [, index_name] ...

```

テーブル参照は、結合式とも呼ばれます。

テーブル参照 (パーティションテーブルを参照する場合) には、カンマ区切りのパーティションまたはサブパーティション (あるいはその両方) のリストを含む **PARTITION** オプションを含めることができます。このオプションはテーブルの名前のあとで、かつエイリアス宣言 (存在する場合) の前に指定されます。このオプションの効果は、リストされたパーティションまたはサブパーティションからのみ行が選択されることです。リストに指定されていないパーティションまたはサブパーティションは無視されます。詳細および例については、[セクション24.5「パーティション選択」](#)を参照してください。

table_factor の構文は、標準 SQL と比較して MySQL で拡張されています。標準では、**table_reference** のみを受け入れ、カッコのペア内のリストは受け入れません。

table_reference アイテムのリスト内の各カンマが内部結合と同等とみなされる場合、これは保守的な拡張機能です。例:

```

SELECT * FROM t1 LEFT JOIN (t2, t3, t4)
      ON (t2.a = t1.a AND t3.b = t1.b AND t4.c = t1.c)

```

次と同等です。

```

SELECT * FROM t1 LEFT JOIN (t2 CROSS JOIN t3 CROSS JOIN t4)
      ON (t2.a = t1.a AND t3.b = t1.b AND t4.c = t1.c)

```

MySQL では、**JOIN**、**CROSS JOIN**、および **INNER JOIN** は構文上同等です (互いに置き換えることができます)。標準 SQL では、それらは同等ではありません。**INNER JOIN** は **ON** 句とともに使用され、**CROSS JOIN** はそれ以外のときに使用されます。

一般に、内部結合操作のみを含む結合式内のかっこは無視できます。MySQL では、ネストした結合もサポートされています。[セクション8.2.1.8「ネストした結合の最適化」](#)を参照してください。

インデックスヒントを指定すると、MySQL オプティマイザによるインデックスの使用方法に影響を与えることができます。詳細は、[セクション8.9.4「インデックスヒント」](#)を参照してください。オプティマイザヒントおよび **optimizer_switch** システム変数は、オプティマイザによるインデックスの使用に影響を与える他の方法です。[セクション8.9.3「オプティマイザヒント」](#) および [セクション8.9.2「切り替え可能な最適化」](#)を参照してください。

次のリストでは、結合の書込み時に考慮する一般的な要因について説明します:

- テーブル参照には、**tbl_name AS alias_name** または **tbl_name alias_name** を使用してエイリアスを指定できます。

```

SELECT t1.name, t2.salary
FROM employee AS t1 INNER JOIN info AS t2 ON t1.name = t2.name;

SELECT t1.name, t2.salary
FROM employee t1 INNER JOIN info t2 ON t1.name = t2.name;

```

- **table_subquery** は、**FROM** 句では導出テーブルまたはサブクエリーとも呼ばれます。[セクション13.2.11.8「導出テーブル」](#)を参照してください。このようなサブクエリーには、サブクエリーの結果にテーブル名を指定するエイリアスを含める必要があります。また、オプションで、カッコ内にテーブルのカラム名のリストを含めることもできます。簡単な例を次に示します:

```
SELECT * FROM (SELECT 1, 2, 3) AS t1;
```

- 1つの結合で参照できるテーブルの最大数は61です。これには、FROM句の導出テーブルおよびビューを外部クエリーブロックにマージすることで処理される結合が含まれます(セクション8.2.2.4「マージまたは実体化を使用した導出テーブル、ビュー参照および共通テーブル式の最適化」を参照)。
- 結合条件が存在しない場合、INNER JOIN と ,(カンマ) は意味的に同等です。どちらも、指定されたテーブル間のデカルト積を生成します(つまり、最初のテーブル内のすべての各行が2番目のテーブル内のすべての各行に結合されます)。

ただし、カンマ演算子の優先順位は INNER JOIN, CROSS JOIN, LEFT JOIN の優先順位よりも低くなります。結合条件が存在するときにカンマ結合をほかの結合型と混在させた場合は、「**「**カラム 'col_name' は 'on clause' にはありません**」**という形式のエラーが発生する可能性があります。この問題への対処に関する情報は、このセクションのあとの方で提供します。

- ON で使用される search_condition は、WHERE 句で使用できるフォームの条件式です。通常、ON 句はテーブルの結合方法を指定する条件に使用され、WHERE 句は結果セットに含める行を制限します。
- LEFT JOIN 内の ON または USING 部分にある右側のテーブルに一致する行が存在しない場合は、すべてのカラムが NULL に設定された行が右側のテーブルに使用されます。このことを使用して、別のテーブルに対応する行が存在しないテーブル内の行を検索できます。

```
SELECT left_tbl.*
FROM left_tbl LEFT JOIN right_tbl ON left_tbl.id = right_tbl.id
WHERE right_tbl.id IS NULL;
```

この例では、right_tbl に存在しない id 値を持つ left_tbl 内のすべての行(つまり、right_tbl 内に対応する行のない left_tbl 内のすべての行)を検索します。セクション8.2.1.9「外部結合の最適化」を参照してください。

- USING(join_column_list) 句は、両方のテーブルに存在する必要があるカラムのリストを指定します。テーブル a と b の両方にカラム c1、c2、および c3 が含まれている場合、次の結合は、この2つのテーブルの対応するカラムを比較します。

```
a LEFT JOIN b USING (c1, c2, c3)
```

- 2つのテーブルの NATURAL [LEFT] JOIN は、両方のテーブル内に存在するすべてのカラムを指定する USING 句を含む INNER JOIN または LEFT JOIN と意味的に同等であるとして定義されます。
- RIGHT JOIN は、LEFT JOIN と同じように機能します。コードをデータベース間で移植可能な状態に維持するため、RIGHT JOIN の代わりに LEFT JOIN を使用することをお勧めします。
- 結合構文の説明に示されている {OJ ...} 構文は、ODBC との互換性のためにのみ存在します。構文内のカールした中括弧は文字どおりに書き込まれる必要があります。それらは構文説明の別の部分で利用されているようなメタ構文ではありません。

```
SELECT left_tbl.*
FROM { OJ left_tbl LEFT OUTER JOIN right_tbl
      ON left_tbl.id = right_tbl.id }
WHERE right_tbl.id IS NULL;
```

{OJ ...} 内では、INNER JOIN や RIGHT OUTER JOIN などのほかの型の結合を使用できます。これは、一部のサードパーティー製アプリケーションとの互換性に役立ちますが、正式な ODBC 構文ではありません。

- STRAIGHT_JOIN は、左側のテーブルが常に右側のテーブルの前に読み取られる点を除き、JOIN と同じです。これは、結合最適マイザが最適でない順序でテーブルを処理する(少数の)場合に使用できます。

結合のいくつかの例:

```
SELECT * FROM table1, table2;
SELECT * FROM table1 INNER JOIN table2 ON table1.id = table2.id;
SELECT * FROM table1 LEFT JOIN table2 ON table1.id = table2.id;
SELECT * FROM table1 LEFT JOIN table2 USING (id);
SELECT * FROM table1 LEFT JOIN table2 ON table1.id = table2.id
```

```
LEFT JOIN table3 ON table2.id = table3.id;
```

外部結合バリエーションを含む **USING** との自然結合および結合は、SQL:2003 標準に従って処理されます:

- **NATURAL** 結合の冗長カラムは表示されません。次の一連のステートメントを考えてみます。

```
CREATE TABLE t1 (i INT, j INT);
CREATE TABLE t2 (k INT, j INT);
INSERT INTO t1 VALUES(1, 1);
INSERT INTO t2 VALUES(1, 1);
SELECT * FROM t1 NATURAL JOIN t2;
SELECT * FROM t1 JOIN t2 USING (j);
```

最初の **SELECT** ステートメントでは、カラム **j** は両方のテーブルに現れるため、結合カラムになります。そのため、標準 SQL に従って、出力には 2 回ではなく 1 回だけ表示されるべきです。同様に、2 番目の **SELECT** ステートメントでは、カラム **j** は **USING** 句で指定されているため、出力には 2 回ではなく 1 回だけ表示されるべきです。

したがって、このステートメントは次の出力を生成します:

```
+----+----+----+
|j |i |k |
+----+----+----+
| 1| 1| 1|
+----+----+----+
|j |i |k |
+----+----+----+
| 1| 1| 1|
+----+----+----+
```

冗長なカラムの削除およびカラムの順序付けは、標準 SQL に従って行われ、次の表示順序が生成されます:

- 最初に、結合された 2 つのテーブルの合体した共通カラムが、最初のテーブルに現れた順序で
- 2 番目に、最初のテーブルに一意のカラムが、そのテーブルに現れた順序で
- 3 番目に、2 番目のテーブルに一意のカラムが、そのテーブルに現れた順序で

2 つの共通カラムを置き換える 1 つの結果カラムは、合体操作を使用して定義されます。つまり、**t1.a** と **t2.a** の 2 つに対して、結果として得られる 1 つの結合カラム **a** は **a = COALESCE(t1.a, t2.a)** として定義されます。ここでは:

```
COALESCE(x, y) = (CASE WHEN x IS NOT NULL THEN x ELSE y END)
```

結合操作が他の結合の場合、結合の結果カラムは結合されたテーブルのすべてのカラムの連結で構成されます。

合体したカラムの定義の結果として、外部結合では、2 つのカラムのいずれかが常に **NULL** である場合、合体したカラムには **NULL** 以外のカラムの値が含まれます。どちらのカラムも **NULL** でないが、または両方のカラムがこの値である場合、両方の共通カラムに同じ値が含まれているため、合体したカラムの値としてどちらが選択されるかは問題にはなりません。これを解釈するための簡単な方法として、外部結合の合体したカラムが **JOIN** の内部テーブルの共通カラムによって表されると考えてみます。テーブル **t1(a, b)** と **t2(a, c)** に次の内容が含まれているとします。

```
t1 t2
---
1x 2z
2y 3w
```

この結合では、カラム **a** に **t1.a** の値が含まれます:

```
mysql> SELECT * FROM t1 NATURAL LEFT JOIN t2;
+----+----+----+
|a |b |c |
+----+----+----+
| 1|x |NULL|
| 2|y |z |
+----+----+----+
```

対照的に、この結合では、カラム `a` に `t2.a` の値が含まれます。

```
mysql> SELECT * FROM t1 NATURAL RIGHT JOIN t2;
+----+-----+-----+
| a  | c  | b  |
+----+-----+-----+
| 2  | z  | y  |
| 3  | w  | NULL|
+----+-----+-----+
```

`JOIN ... ON` を使用して、これらの結果をそれ以外の同等のクエリーと比較します:

```
mysql> SELECT * FROM t1 LEFT JOIN t2 ON (t1.a = t2.a);
+----+-----+-----+
| a  | b  | a  | c  |
+----+-----+-----+
| 1  | x  | NULL| NULL|
| 2  | y  | 2  | z  |
+----+-----+-----+
```

```
mysql> SELECT * FROM t1 RIGHT JOIN t2 ON (t1.a = t2.a);
+----+-----+-----+
| a  | b  | a  | c  |
+----+-----+-----+
| 2  | y  | 2  | z  |
| NULL| NULL| 3  | w  |
+----+-----+-----+
```

- `USING` 句は、対応するカラムを比較する `ON` 句としてリライトできます。ただし、`USING` と `ON` は似ていますが、まったく同じではありません。次の 2 つのクエリーについて考えてみます:

```
a LEFT JOIN b USING (c1, c2, c3)
a LEFT JOIN b ON a.c1 = b.c1 AND a.c2 = b.c2 AND a.c3 = b.c3
```

結合条件を満たす行の判別に関して、両方の結合は意味的に同一です。

`SELECT *` の展開に対してどのカラムを表示するかの判定に関しては、この 2 つの結合は意味的に同一ではありません。`USING` 結合が対応するカラムの合体した値を選択するのに対して、`ON` 結合は、すべてのテーブルのすべてのカラムを選択します。`USING` 結合の場合、`SELECT *` は次の値を選択します:

```
COALESCE(a.c1, b.c1), COALESCE(a.c2, b.c2), COALESCE(a.c3, b.c3)
```

`ON` 結合の場合、`SELECT *` は次の値を選択します。

```
a.c1, a.c2, a.c3, b.c1, b.c2, b.c3
```

内部結合では、両方のカラムの値が同じであるため、`COALESCE(a.c1, b.c1)` は `a.c1` または `b.c1` と同じです。外部結合 (`LEFT JOIN` など) では、2 つのカラムのどちらかが `NULL` になる場合があります。そのカラムは結果から省略されます。

- `ON` 句は、そのオペランドのみを参照できます。

例:

```
CREATE TABLE t1 (i1 INT);
CREATE TABLE t2 (i2 INT);
CREATE TABLE t3 (i3 INT);
SELECT * FROM t1 JOIN t2 ON (i1 = i3) JOIN t3;
```

`i3` は `ON` 句のオペランドではない `t3` のカラムであるため、このステートメントは `Unknown column 'i3' in 'on clause'` エラーで失敗します。結合を処理できるようにするには、次のようにステートメントをリライトします:

```
SELECT * FROM t1 JOIN t2 JOIN t3 ON (i1 = i3);
```


- **JOIN** の優先順位はカンマ演算子 (,) より高いため、結合式 `t1, t2 JOIN t3` は `((t1, t2) JOIN t3)` としてではなく `(t1, (t2 JOIN t3))` として解釈されます。これは、**ON** 句を使用するステートメントに影響します。この句は結合のオペランド内のカラムのみを参照でき、優先順位はそれらのオペランドの解釈に影響します。

例:

```
CREATE TABLE t1 (i1 INT, j1 INT);
CREATE TABLE t2 (i2 INT, j2 INT);
CREATE TABLE t3 (i3 INT, j3 INT);
INSERT INTO t1 VALUES(1, 1);
INSERT INTO t2 VALUES(1, 1);
INSERT INTO t3 VALUES(1, 1);
SELECT * FROM t1, t2 JOIN t3 ON (t1.i1 = t3.i3);
```

JOIN はカンマ演算子よりも優先されるため、**ON** 句のオペランドは `t2` および `t3` です。 `t1.i1` はどのオペランドのカラムでもないため、その結果は「カラム 't1.i1' は 'on clause' にはありません」というエラーになります。

結合を処理できるようにするには、次のいずれかの方法を使用します:

- **ON** 句のオペランドが `(t1, t2)` および `t3` になるように、最初の 2 つのテーブルをカッコで明示的にグループ化します:

```
SELECT * FROM (t1, t2) JOIN t3 ON (t1.i1 = t3.i3);
```

- カンマ演算子を使用せずに、かわりに **JOIN** を使用します:

```
SELECT * FROM t1 JOIN t2 JOIN t3 ON (t1.i1 = t3.i3);
```

同じ優先順位解釈が、カンマ演算子と **INNER JOIN**, **CROSS JOIN**, **LEFT JOIN** および **RIGHT JOIN** を混在させるステートメントにも適用されます。これらはすべてカンマ演算子よりも優先順位が高くなります。

- SQL:2003 標準と比較した MySQL 拡張機能では、MySQL では **NATURAL** または **USING** 結合の共通 (結合) カラムを修飾できますが、標準では修飾できません。

13.2.10.3 UNION 句

```
SELECT ...
UNION [ALL | DISTINCT] SELECT ...
[UNION [ALL | DISTINCT] SELECT ...]
```

UNION は、複数の **SELECT** ステートメントの結果を単一の結果セットに結合します。例:

```
mysql> SELECT 1, 2;
+---+---+
| 1 | 2 |
+---+---+
| 1 | 2 |
+---+---+
mysql> SELECT 'a', 'b';
+---+---+
| a | b |
+---+---+
| a | b |
+---+---+
mysql> SELECT 1, 2 UNION SELECT 'a', 'b';
+---+---+
| 1 | 2 |
+---+---+
| 1 | 2 |
| a | b |
+---+---+
```

- [結果セットのカラム名およびデータ型](#)
- [共用体の TABLE](#)
- [UNION DISTINCT および UNION ALL](#)
- [UNION での ORDER BY および LIMIT](#)

- [UNION の制限](#)
- [MySQL 5.7 と比較した MySQL 8.0 での UNION の処理](#)

結果セットのカラム名およびデータ型

UNION 結果セットのカラム名は、最初の **SELECT** ステートメントのカラム名から取得されます。

各 **SELECT** ステートメントの対応する位置にリストされている選択されるカラムは、データ型が同じになるようにしてください。たとえば、最初のステートメントで選択された最初のカラムは、他のステートメントで選択された最初のカラムと同じタイプである必要があります。対応する **SELECT** カラムのデータ型が一致しない場合、**UNION** 結果のカラムの型と長さでは、すべての **SELECT** ステートメントによって取得された値が考慮されます。たとえば、カラムの長さが最初の **SELECT** の値の長さに制約されていない次の例を考えてみます：

```
mysql> SELECT REPEAT('a',1) UNION SELECT REPEAT('b',20);
+-----+
| REPEAT('a',1) |
+-----+
| a              |
| bbbbbbbbbbbbbbbbbbbb |
+-----+
```

共用体の TABLE

MySQL 8.0.19 以降では、同等の **SELECT** ステートメントを使用できる場所であれば、**UNION** で **TABLE** ステートメントまたは **VALUES** ステートメントを使用することもできます。次に示すように、テーブル **t1** および **t2** が作成され、移入されているとします：

```
CREATE TABLE t1 (x INT, y INT);
INSERT INTO t1 VALUES ROW(4,-2),ROW(5,9);

CREATE TABLE t2 (a INT, b INT);
INSERT INTO t2 VALUES ROW(1,2),ROW(3,4);
```

前述の例では、**VALUES** 以降のクエリーの出力のカラム名は無視され、次の **UNION** クエリーはすべて同じ結果になります：

```
SELECT * FROM t1 UNION SELECT * FROM t2;
TABLE t1 UNION SELECT * FROM t2;
VALUES ROW(4,-2), ROW(5,9) UNION SELECT * FROM t2;
SELECT * FROM t1 UNION TABLE t2;
TABLE t1 UNION TABLE t2;
VALUES ROW(4,-2), ROW(5,9) UNION TABLE t2;
SELECT * FROM t1 UNION VALUES ROW(4,-2),ROW(5,9);
TABLE t1 UNION VALUES ROW(4,-2),ROW(5,9);
VALUES ROW(4,-2), ROW(5,9) UNION VALUES ROW(4,-2),ROW(5,9);
```

カラム名を強制的に同じにするには、**VALUES** を **SELECT** の左側にラップし、次のようにエイリアスを使用します：

```
SELECT * FROM (VALUES ROW(4,-2), ROW(5,9)) AS t(x,y)
UNION TABLE t2;
SELECT * FROM (VALUES ROW(4,-2), ROW(5,9)) AS t(x,y)
UNION VALUES ROW(4,-2),ROW(5,9);
```

UNION DISTINCT および UNION ALL

デフォルトでは、重複行は **UNION** の結果から削除されます。オプションの **DISTINCT** キーワードも同じ効果がありますが、明示的になります。オプションの **ALL** キーワードを指定すると、重複した行の削除は実行されず、その結果には、すべての **SELECT** ステートメントからの一致するすべての行が含まれます。

UNION ALL と **UNION DISTINCT** を同じクエリー内で混在させることができます。混在した **UNION** 型は、**DISTINCT** 和集合がその左側にある **ALL** 和集合をすべてオーバーライドするように処理されます。**DISTINCT** 和集合は、**UNION DISTINCT** を使用して明示的に、あるいはそのあとに **DISTINCT** または **ALL** キーワードのない **UNION** を使用して暗黙的に生成できます。

MySQL 8.0.19 以降では、**UNION ALL** および **UNION DISTINCT** は、1 つ以上の **TABLE** ステートメントが共用体で使用されている場合と同じように動作します。

UNION での ORDER BY および LIMIT

ORDER BY または **LIMIT** 句を個々の **SELECT** に適用するには、**SELECT** をカッコで囲み、カッコ内に句を配置します:

```
(SELECT a FROM t1 WHERE a=10 AND B=1 ORDER BY a LIMIT 10)
UNION
(SELECT a FROM t2 WHERE a=11 AND B=2 ORDER BY a LIMIT 10);
```

個々の **SELECT** ステートメントに **ORDER BY** を使用すると、**UNION** では順序付けられていない行のセットがデフォルトで生成されるため、最終結果に行が表示される順序については何も意味しません。したがって、このコンテキストの **ORDER BY** は通常、**SELECT** 用に取得する選択済の行のサブセットを決定するために **LIMIT** とともに使用されますが、最終的な **UNION** 結果の行の順序には必ずしも影響しません。**SELECT** に **LIMIT** がない状態で **ORDER BY** が表示された場合は、どのような場合にも効果がないため最適化されます。

ORDER BY または **LIMIT** 句を使用して **UNION** 結果全体をソートまたは制限するには、個々の **SELECT** ステートメントをカッコで囲み、最後のステートメントの後に **ORDER BY** または **LIMIT** を配置します:

```
(SELECT a FROM t1 WHERE a=10 AND B=1)
UNION
(SELECT a FROM t2 WHERE a=11 AND B=2)
ORDER BY a LIMIT 10;
```

括弧のないステートメントは、今示した括弧で囲まれたステートメントと同等です。

MySQL 8.0.19 以降、**TABLE** は **WHERE** 句をサポートしていないことに注意して、前述の方法と同じ方法で **TABLE** とともに **ORDER BY** および **LIMIT** を使用できます。

この種の **ORDER BY** は、テーブル名 (つまり、`tbl_name.col_name` という形式の名前) を含むカラム参照を使用できません。代わりに、最初の **SELECT** ステートメント内にカラムのエイリアスを指定し、そのエイリアスを **ORDER BY** 内で参照します。(あるいは、**ORDER BY** 内でカラムを、そのカラム位置を使用して参照します。ただし、カラム位置の使用は非推奨です。)

また、ソートされるカラムにエイリアスが指定されている場合、**ORDER BY** 句はそのカラム名ではなく、エイリアスを参照する必要があります。次のステートメントのうち最初のステートメントは許可されていますが、次のステートメントは `Unknown column 'a' in 'order clause'` エラーで失敗します:

```
(SELECT a AS b FROM t) UNION (SELECT ...) ORDER BY b;
(SELECT a AS b FROM t) UNION (SELECT ...) ORDER BY a;
```

UNION 結果の行が各 **SELECT** によって取得された行のセットで構成されるようにするには、ソートカラムとして使用する各 **SELECT** の追加のカラムを選択し、最後の **SELECT** に続くそのカラムでソートする **ORDER BY** を追加します:

```
(SELECT 1 AS sort_col, col1a, col1b, ... FROM t1)
UNION
(SELECT 2, col2a, col2b, ... FROM t2) ORDER BY sort_col;
```

さらに個々の **SELECT** の結果内のソート順序を維持するには、**ORDER BY** 句にセカンダリカラムを追加します。

```
(SELECT 1 AS sort_col, col1a, col1b, ... FROM t1)
UNION
(SELECT 2, col2a, col2b, ... FROM t2) ORDER BY sort_col, col1a;
```

また、追加のカラムを使用すると、各行がどの **SELECT** から取得されるかを決定することもできます。追加のカラムでは、テーブル名を示す文字列などのほかの識別情報も指定できます。

UNION の制限

UNION では、**SELECT** ステートメントは通常の **SELECT** ステートメントですが、次の制限があります:

- 最初の **SELECT** の **HIGH_PRIORITY** は効果がありません。後続の **SELECT** の **HIGH_PRIORITY** では、構文エラーが発生します。
- 最後の **SELECT** ステートメントのみが **INTO** 句を使用できます。ただし、**UNION** の結果全体が **INTO** 出力先に書き込まれます。

MySQL 8.0.20 では、**INTO** を含む次の 2 つの **UNION** バリエーションは非推奨であり、将来のバージョンの MySQL でサポートが削除される予定です:

- クエリー式の後続のクエリーブロックでは、**FROM** の前に **INTO** を使用すると警告が生成されます。例:

```
... UNION SELECT * INTO OUTFILE 'file_name' FROM table_name;
```

- クエリー式のカッコで囲まれた後続ブロックでは、**INTO** を (**FROM** に対する相対位置に関係なく) 使用すると警告が生成されます。例:

```
... UNION (SELECT * INTO OUTFILE 'file_name' FROM table_name);
```

これらのバリエーションは、クエリー式 (**UNION**) 全体ではなく名前付きテーブルから情報を収集するかのようになり、混乱しているため非推奨です。

ORDER BY 句に集計関数を含む **UNION** クエリーは、**ER_AGGREGATE_ORDER_FOR_UNION** エラーで拒否されます。例:

```
SELECT 1 AS foo UNION SELECT 2 ORDER BY MAX(1);
```

MySQL 5.7 と比較した MySQL 8.0 での UNION の処理

MySQL 8.0 では、**SELECT** および **UNION** のパーサールールがリファクタリングされ、より一貫性が保たれ (このような各コンテキストで同じ **SELECT** 構文が均一に適用される)、重複が削減されました。MySQL 5.7 と比較すると、この作業の結果、次のようないくつかのユーザーに見える影響があり、特定のステートメントのリライトが必要になる場合があります:

- NATURAL JOIN** では、標準 SQL に準拠したオプションの **INNER** キーワード (**NATURAL INNER JOIN**) を使用できません。
- 標準 SQL に準拠すると、カッコなしの右重複結合 (... **JOIN** ... **JOIN** ... **ON** ... **ON** ...) が許可されます。
- STRAIGHT_JOIN** では、他の内部結合と同様に **USING** 句が許可されるようになりました。
- パーサーは、クエリー式を囲むカッコを受け入れます。たとえば、(**SELECT** ... **UNION SELECT** ...) は許可されています。セクション 13.2.10.4 「カッコで囲まれたクエリー式」も参照してください。
- パーサーは、**SQL_CACHE** および **SQL_NO_CACHE** クエリー修飾子の許可された配置に準拠しています。
- 以前はサブクエリーでのみ許可されていた共用体の左端のネストが、トップレベルのステートメントで許可されるようになりました。たとえば、次のステートメントは有効として受け入れられます:

```
(SELECT 1 UNION SELECT 1) UNION SELECT 1;
```

- ロック句 (**FOR UPDATE**、**LOCK IN SHARE MODE**) は、**UNION** 以外のクエリーでのみ使用できます。つまり、ロック句を含む **SELECT** ステートメントにはカッコを使用する必要があります。このステートメントは有効ではなくなりました:

```
SELECT 1 FOR UPDATE UNION SELECT 1 FOR UPDATE;
```

かわりに、次のようなステートメントを記述します:

```
(SELECT 1 FOR UPDATE) UNION (SELECT 1 FOR UPDATE);
```

13.2.10.4 カッコで囲まれたクエリー式

```
parenthesized_query_expression:
 ( query_expression [order_by_clause] [limit_clause] )
 [order_by_clause]
 [limit_clause]
 [into_clause]

query_expression:
 query_block [UNION query_block [UNION query_block ...]]
 [order_by_clause]
 [limit_clause]
```

```
[into_clause]

query_block:
  SELECT ...      (see セクション13.2.10 「SELECT ステートメント」)

order_by_clause:
  ORDER BY as for SELECT (see セクション13.2.10 「SELECT ステートメント」)

limit_clause:
  LIMIT as for SELECT (see セクション13.2.10 「SELECT ステートメント」)

into_clause:
  INTO as for SELECT (see セクション13.2.10 「SELECT ステートメント」)
```

MySQL 8.0.22 以上では、前述の構文に従ってカッコで囲まれたクエリー式がサポートされます。最も単純なのは、カッコで囲まれたクエリー式に単一の **SELECT** が含まれ、次のオプション句は含まれないことです:

```
(SELECT 1);
(SELECT * FROM INFORMATION_SCHEMA.SCHEMATA WHERE SCHEMA_NAME = 'mysql');
```

カッコで囲まれたクエリー式には、複数の **SELECT** ステートメントで構成される **UNION** を含めることもでき、オプションの句の一部またはすべてで終わることができます:

```
mysql> (SELECT 1 AS result UNION SELECT 2);
+-----+
| result |
+-----+
| 1 |
| 2 |
+-----+
mysql> (SELECT 1 AS result UNION SELECT 2) LIMIT 1;
+-----+
| result |
+-----+
| 1 |
+-----+
mysql> (SELECT 1 AS result UNION SELECT 2) LIMIT 1 OFFSET 1;
+-----+
| result |
+-----+
| 2 |
+-----+
mysql> (SELECT 1 AS result UNION SELECT 2)
  ORDER BY result DESC LIMIT 1;
+-----+
| result |
+-----+
| 2 |
+-----+
mysql> (SELECT 1 AS result UNION SELECT 2)
  ORDER BY result DESC LIMIT 1 OFFSET 1;
+-----+
| result |
+-----+
| 1 |
+-----+
mysql> (SELECT 1 AS result UNION SELECT 3 UNION SELECT 2)
  ORDER BY result LIMIT 1 OFFSET 1 INTO @var;
mysql> SELECT @var;
+-----+
| @var |
+-----+
| 2 |
+-----+
```

カッコで囲まれたクエリー式はクエリー式としても使用されるため、クエリー式は通常、クエリーブロックで構成され、カッコで囲まれたクエリー式で構成されることもあります:

```
(SELECT * FROM t1 ORDER BY a) UNION (SELECT * FROM t2 ORDER BY b) ORDER BY z;
```

クエリーブロックには、外部の **UNION** および **ORDER BY** と **LIMIT** の前に適用される後続の **ORDER BY** 句および **LIMIT** 句がある場合があります。

末尾に **ORDER BY** または **LIMIT** を含むクエリーブロックをカッコで囲まずに含めることはできませんが、カッコは様々な方法で強制に使用できます:

- 各クエリーブロックに **LIMIT** を適用するには:

```
(SELECT 1 LIMIT 1) UNION (SELECT 2 LIMIT 1);
```

- クエリーブロックとクエリー式全体の両方に **LIMIT** を適用するには:

```
(SELECT 1 LIMIT 1) UNION (SELECT 2 LIMIT 1) LIMIT 1;
```

- クエリー式全体に **LIMIT** を適用するには (カッコなし):

```
SELECT 1 UNION SELECT 2 LIMIT 1;
```

- ハイブリッド強制: 最初のクエリーブロックおよびクエリー式全体に対する **LIMIT**:

```
(SELECT 1 LIMIT 1) UNION SELECT 2 LIMIT 1;
```

このセクションで説明する構文には、特定の制限事項があります:

- ORDER BY** がカッコで囲まれたクエリー式内で発生し、外部クエリーにも適用される場合、結果は未定義であり、将来のバージョンの MySQL で変更される可能性があります。 **LIMIT** がカッコで囲まれたクエリー式内に出現し、外部クエリーにも適用される場合も同様です。
- クエリー式の末尾の **INTO** 句は、カッコ内に別の **INTO** 句がある場合は使用できません。
- カッコで囲まれたクエリー式では、複数レベルの **ORDER BY** または **LIMIT** 操作は許可されません。 例:

```
mysql> (SELECT 'a' UNION SELECT 'b' LIMIT 1) LIMIT 2;
ERROR 1235 (42000): This version of MySQL doesn't yet support 'parenthesized
query expression with more than one external level of ORDER/LIMIT operations'
```

13.2.11 サブクエリー

サブクエリーは、別のステートメント内の **SELECT** ステートメントです。

SQL 標準で必要とされるすべてのサブクエリーフォームと操作、および MySQL 固有のいくつかの機能がサポートされています。

サブクエリーの例を次に示します。

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
```

この例では、**SELECT * FROM t1 ...** が外部クエリー (または 外部ステートメント) であり、**(SELECT column1 FROM t2)** がサブクエリーです。これを、このサブクエリーは外部クエリー内でネストされていると表現し、また実際、サブクエリーをほかのサブクエリー内で (かなりの深さまで) ネストできます。サブクエリーは常に、括弧内に指定する必要があります。

サブクエリーの主な利点は次のとおりです。

- ステートメントの各部分を分離できるように、構造化されたクエリーを可能にします。
- 通常であれば複雑な結合や和集合を必要とする操作を実行するための代替手段を提供します。
- 多くの人びとが、サブクエリーを複雑な結合や和集合より読みやすいと感じています。実際、早期の SQL である「構造化クエリー言語」を呼び出すという元の考え方を人びとに提供したのは、サブクエリーの技術革新でした。

SQL 標準で指定され、MySQL でサポートされているサブクエリー構文に関する主なポイントを示すステートメントの例を次に示します。

```
DELETE FROM t1
WHERE s11 > ANY
(SELECT COUNT(*) /* no hint */ FROM t2
```



```
WHERE NOT EXISTS
(SELECT * FROM t3
 WHERE ROW(5*t2.s1,77)=
 (SELECT 50,11*s1 FROM t4 UNION SELECT 50,77 FROM
 (SELECT * FROM t5) AS t5));
```

サブクエリーは、スカラー (単一値)、単一行、単一カラム、またはテーブル (1 つ以上のカラムの 1 つ以上の行) を返すことができます。これらは、スカラー、カラム、行、およびテーブルサブクエリーと呼ばれます。特定の種類の結果を返すサブクエリーは多くの場合、次の各セクションで説明されているように、特定のコンテキストでのみ使用できます。

サブクエリーを使用できるステートメントのタイプに関する制限はほとんどありません。サブクエリーには、[DISTINCT](#)、[GROUP BY](#)、[ORDER BY](#)、[LIMIT](#)、結合、インデックスヒント、[UNION](#) 構造構文、コメント、関数などの、通常の [SELECT](#) に含めることのできる多くのキーワードや句を含めることができます。

MySQL 8.0.19 以降、[TABLE](#) および [VALUES](#) ステートメントをサブクエリーで使用できます。通常、[VALUES](#) を使用するサブクエリーは、set 表記法を使用するか、[SELECT](#) または [TABLE](#) 構文を使用してよりコンパクトにリライトできる、より詳細なバージョンのサブクエリーです。[CREATE TABLE ts VALUES ROW\(2\), ROW\(4\), ROW\(6\)](#) ステートメントを使用して [ts](#) テーブルを作成する場合、ここに示すステートメントはすべて同等です:

```
SELECT * FROM tt
 WHERE b > ANY (VALUES ROW(2), ROW(4), ROW(6));
```

```
SELECT * FROM tt
 WHERE b > ANY (2, 4, 6);
```

```
SELECT * FROM tt
 WHERE b > ANY (SELECT * FROM ts);
```

```
SELECT * FROM tt
 WHERE b > ANY (TABLE ts);
```

次の各セクションでは、[TABLE](#) サブクエリーの例を示します。

サブクエリーの外部ステートメントは、[SELECT](#)、[INSERT](#)、[UPDATE](#)、[DELETE](#)、[SET](#)、[DO](#) のいずれでもかまいません。

最適化によるサブクエリーの処理方法については、[セクション8.2.2「サブクエリー、導出テーブル、ビュー参照および共通テーブル式の最適化」](#)を参照してください。サブクエリーの使用に関する制限の説明 (特定の形式のサブクエリー構文でのパフォーマンスの問題を含む) については、[セクション13.2.11.12「サブクエリーの制約」](#)を参照してください。

13.2.11.1 スカラーオペランドとしてのサブクエリー

もっとも単純な形式のサブクエリーは、単一値を返すスカラーサブクエリーです。スカラーサブクエリーは単純なオペランドであるため、単一カラム値またはリテラルが正当である場所であればほぼどこでも使用できるほか、データ型、長さ、[NULL](#) にできることの表示などの、すべてのオペランドが持っている特性を持つことを期待できます。例:

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5) NOT NULL);
INSERT INTO t1 VALUES(100, 'abcde');
SELECT (SELECT s2 FROM t1);
```

この [SELECT](#) 内のサブクエリーは、[CHAR](#) のデータ型、5 の長さ、[CREATE TABLE](#) の時点で有効なデフォルトに等しい文字セットと照合順序、およびこのカラム内の値を [NULL](#) にできることの表示を持つ単一値 ('abcde') を返します。サブクエリー結果が空であればその結果は [NULL](#) になるため、スカラーサブクエリーによって選択された値の [NULL](#) 可能性はコピーされません。今示したサブクエリーで [t1](#) が空であった場合は、[s2](#) が [NOT NULL](#) であるにもかかわらず、その結果は [NULL](#) になります。

スカラーサブクエリーを使用できないコンテキストがいくつか存在します。ステートメントでリテラル値のみが許可されている場合は、サブクエリーを使用できません。たとえば、[LIMIT](#) にはリテラル整数指数が必要で、[LOAD DATA](#) にはリテラル文字列ファイル名が必要です。サブクエリーを使用してこれらの値を指定することはできません。

次の各セクションにある、やや簡素な構造構文 ([SELECT column1 FROM t1](#)) が含まれた例を参照するときは、はるかに多様で、かつ複雑な構造構文を含む独自のコードがあるものと考えてください。

次の 2 つのテーブルを作成するとします。

```
CREATE TABLE t1 (s1 INT);
INSERT INTO t1 VALUES (1);
CREATE TABLE t2 (s1 INT);
INSERT INTO t2 VALUES (2);
```

次に、`SELECT` を実行します。

```
SELECT (SELECT s1 FROM t2) FROM t1;
```

`t2` には、`2` の値を持つカラム `s1` が含まれている行が存在するため、その結果は `2` になります。

MySQL 8.0.19 以降では、`TABLE` を使用して前述のクエリーを次のように記述することもできます：

```
SELECT (TABLE t2) FROM t1;
```

スカラーサブクエリーを式の一部にすることはできますが、そのサブクエリーが関数への引数を提供するオペランドである場合でも、括弧を忘れないでください。例：

```
SELECT UPPER((SELECT s1 FROM t1)) FROM t2;
```

同じ結果は、MySQL 8.0.19 以降で `SELECT UPPER((TABLE t1)) FROM t2` を使用して取得できます。

13.2.11.2 サブクエリーを使用した比較

サブクエリーのもっとも一般的な使用の形式は次のとおりです。

```
non_subquery_operand comparison_operator (subquery)
```

ここで、`comparison_operator` は次の演算子のいずれかです。

```
= > < >= <= <> != <=>
```

例：

```
... WHERE 'a' = (SELECT column1 FROM t1)
```

MySQL では、次の構造構文も許可されます。

```
non_subquery_operand LIKE (subquery)
```

以前は、サブクエリーの唯一の正当な場所は比較の右側であり、この方法にこだわったいくつかの古い DBMS がまだ見つかることもあります。

結合では実行できない一般的な形式のサブクエリー比較の例を次に示します。これは、`column1` 値がテーブル `t2` 内の最大値に等しいテーブル `t1` 内のすべての行を検索します。

```
SELECT * FROM t1
WHERE column1 = (SELECT MAX(column2) FROM t2);
```

次に別の例を示します。これもまた、いずれかのテーブルに対する集約が含まれているため、結合では実行できません。これは、特定のカラムに 2 回現れる値を含むテーブル `t1` 内のすべての行を検索します。

```
SELECT * FROM t1 AS t
WHERE 2 = (SELECT COUNT(*) FROM t1 WHERE t1.id = t.id);
```

スカラーに対するサブクエリーの比較の場合、サブクエリーはスカラーを返す必要があります。行コンストラクタに対するサブクエリーの比較の場合、サブクエリーは、その行コンストラクタと同じ数の値を含む行を返す行サブクエリーである必要があります。[セクション 13.2.11.5 「行サブクエリー」](#) を参照してください。

13.2.11.3 ANY、IN、または SOME を使用したサブクエリー

構文：

```
operand comparison_operator ANY (subquery)
operand IN (subquery)
operand comparison_operator SOME (subquery)
```

ここで、`comparison_operator` は次の演算子のいずれかです。

```
= > < >= <= <> !=
```

`ANY` キーワード (これは比較演算子のあとに指定する必要があります) は、「このサブクエリーが返すカラム内の値の `ANY` (いずれか) に対して比較が `TRUE` である場合は `TRUE` を返す」ことを示します。例:

```
SELECT s1 FROM t1 WHERE s1 > ANY (SELECT s1 FROM t2);
```

テーブル `t1` 内に (10) を含む行が存在するとします。テーブル `t2` に (21,14,7) が含まれている場合、`t2` には 10 より小さい値 7 が存在するため、この式は `TRUE` です。テーブル `t2` に (20,10) が含まれている場合、またはテーブル `t2` が空である場合、この式は `FALSE` です。テーブル `t2` に (NULL,NULL,NULL) が含まれている場合、この式は不明 (つまり、`NULL`) です。

サブクエリーで使用されている場合、ワード `IN` は `= ANY` のエイリアスです。そのため、次の 2 つのステートメントは同じです。

```
SELECT s1 FROM t1 WHERE s1 = ANY (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 IN (SELECT s1 FROM t2);
```

式リストで使用されている場合、`IN` と `= ANY` はシノニムではありません。 `IN` は式リストを取得できますが、`= ANY` はできません。 [セクション12.4.2「比較関数と演算子」](#) を参照してください。

`NOT IN` は `<> ANY` ではなく、`<> ALL` のエイリアスです。 [セクション13.2.11.4「ALL を使用したサブクエリー」](#) を参照してください。

ワード `SOME` は `ANY` のエイリアスです。そのため、次の 2 つのステートメントは同じです。

```
SELECT s1 FROM t1 WHERE s1 <> ANY (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 <> SOME (SELECT s1 FROM t2);
```

ワード `SOME` はほとんど使用されませんが、この例は、これがなぜ役立つ可能性があるかを示しています。ほとんどの人びとにとって、「a is not equal to any b」(a はどの b にも等しくない) という英語のフレーズは「there is no b which is equal to a」(a に等しい b は存在しない) を示しますが、それはこの SQL 構文が示す内容とは異なります。この構文は、「there is some b to which a is not equal」(a に等しくない b がいくつか存在する) を示します。代わりに `<> SOME` を使用すると、このクエリーの本当の意味がすべての人に理解されるようにするのに役立ちます。

MySQL 8.0.19 以降、テーブルに単一のカラムのみが含まれている場合は、スカラー `IN`、`ANY` または `SOME` サブクエリーで `TABLE` を使用できます。 `t2` にカラムが 1 つしかない場合は、このセクションで前述したステートメントを次に示すように記述できます。各場合は、`SELECT s1 FROM t2` を `TABLE t2` に置き換えます:

```
SELECT s1 FROM t1 WHERE s1 > ANY (TABLE t2);
SELECT s1 FROM t1 WHERE s1 = ANY (TABLE t2);
SELECT s1 FROM t1 WHERE s1 IN (TABLE t2);
SELECT s1 FROM t1 WHERE s1 <> ANY (TABLE t2);
SELECT s1 FROM t1 WHERE s1 <> SOME (TABLE t2);
```

13.2.11.4 ALL を使用したサブクエリー

構文:

```
operand comparison_operator ALL (subquery)
```

ワード `ALL` (これは比較演算子のあとに指定する必要があります) は、「このサブクエリーが返すカラム内の値の `ALL` (すべて) に対して比較が `TRUE` である場合は `TRUE` を返す」ことを示します。例:

```
SELECT s1 FROM t1 WHERE s1 > ALL (SELECT s1 FROM t2);
```

テーブル *t1* 内に (10) を含む行が存在するとします。テーブル *t2* に (-5,0,+5) が含まれている場合、10 が *t2* 内の 3 つのすべての値より大きいため、この式は **TRUE** です。テーブル *t2* に (12,6,NULL,-100) が含まれている場合、テーブル *t2* には 10 より大きい単一値 12 が存在するため、この式は **FALSE** です。テーブル *t2* に (0,NULL,1) が含まれている場合、この式は不明 (つまり、**NULL**) です。

最後に、テーブル *t2* が空である場合、この式は **TRUE** です。そのため、テーブル *t2* が空であるとき、次の式は **TRUE** です。

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT s1 FROM t2);
```

ただし、テーブル *t2* が空であるとき、次の式は **NULL** です。

```
SELECT * FROM t1 WHERE 1 > (SELECT s1 FROM t2);
```

さらに、テーブル *t2* が空であるとき、次の式は **NULL** です。

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT MAX(s1) FROM t2);
```

一般に、**NULL** 値を含むテーブルと空のテーブルは「エッジケース」です。サブクエリーを記述するときは、常に、これらの 2 つの可能性を考慮に入れたかどうかを考慮してください。

NOT IN は **<> ALL** のエイリアスです。そのため、次の 2 つのステートメントは同じです。

```
SELECT s1 FROM t1 WHERE s1 <> ALL (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 NOT IN (SELECT s1 FROM t2);
```

MySQL 8.0.19 では、**TABLE** ステートメントがサポートされます。**IN**、**ANY** および **SOME** と同様に、次の 2 つの条件が満たされている場合は **TABLE** を **ALL** および **NOT IN** とともに使用できます:

- サブクエリーのテーブルに含まれるカラムは 1 つのみです
- サブクエリーはカラム式に依存していません

たとえば、テーブル *t2* が単一のカラムで構成されていると仮定すると、前述の最後の 2 つのステートメントは、次のように **TABLE t2** を使用して記述できます:

```
SELECT s1 FROM t1 WHERE s1 <> ALL (TABLE t2);
SELECT s1 FROM t1 WHERE s1 NOT IN (TABLE t2);
```

サブクエリーはカラム式に依存するため、**SELECT * FROM t1 WHERE 1 > ALL (SELECT MAX(s1) FROM t2);**などのクエリーは **TABLE t2** を使用して記述できません。

13.2.11.5 行サブクエリー

スカラーサブクエリーまたはカラムサブクエリーは、単一の値または値のカラムを返します。行サブクエリーは、単一行を返し、そのために複数のカラム値を返すことができるサブクエリーバリエーションです。行サブクエリーの比較のための正当な演算子は次のとおりです。

```
= > < >= <= <> != <=>
```

次に、2 つの例を示します。

```
SELECT * FROM t1
  WHERE (col1,col2) = (SELECT col3, col4 FROM t2 WHERE id = 10);
SELECT * FROM t1
  WHERE ROW(col1,col2) = (SELECT col3, col4 FROM t2 WHERE id = 10);
```

どちらのクエリーでも、テーブル *t2* に **id = 10** を持つ単一行が含まれている場合、このサブクエリーは単一行を返します。この行に *t1* 内のいずれかの行の **col1** および **col2** 値に等しい **col3** および **col4** 値が含まれている場合、**WHERE** 式は **TRUE** であり、各クエリーはこれらの *t1* 行を返します。*t2* 行の **col3** および **col4** 値が、いずれの *t1* 行の **col1** および **col2** 値にも等しくない場合、この式は **FALSE** であり、このクエリーは空の結果セットを返します。サブクエリーによって行が生成されない場合、この式は不明 (つまり、**NULL**) です。サブクエリーによって複数の行が生成される場合は、行サブクエリーが最大で 1 行しか返すことができないため、エラーが発生します。

行の比較における各演算子の動作の詳細は、[セクション12.4.2「比較関数と演算子」](#)を参照してください。

式 `(1,2)` や `ROW(1,2)` は、行コンストラクタとも呼ばれます。この2つは同等です。行コンストラクタと、サブクエリーによって返される行には、同じ数の値が含まれている必要があります。

行コンストラクタは、2つ以上のカラムを返すサブクエリーとの比較に使用されます。サブクエリーが単一カラムを返すと、これは行ではなく、スカラー値として見なされるため、少なくとも2つのカラムを返さないサブクエリーで行コンストラクタを使用することはできません。そのため、次のクエリーは構文エラーで失敗します。

```
SELECT * FROM t1 WHERE ROW(1) = (SELECT column1 FROM t2)
```

行コンストラクタは、ほかのコンテキストでも正当です。たとえば、次の2つのステートメントは意味的に同等です(また、オプティマイザによって同じように処理されます)。

```
SELECT * FROM t1 WHERE (column1,column2) = (1,1);
SELECT * FROM t1 WHERE column1 = 1 AND column2 = 1;
```

次のクエリーは、「テーブル `t2` 内にも存在するテーブル `t1` 内のすべての行を検索する」という要求にこたえます。

```
SELECT column1,column2,column3
FROM t1
WHERE (column1,column2,column3) IN
(SELECT column1,column2,column3 FROM t2);
```

オプティマイザおよび行コンストラクタの詳細は、[セクション8.2.1.22「行コンストラクタ式の最適化」](#)を参照してください

13.2.11.6 EXISTS または NOT EXISTS を使用したサブクエリー

サブクエリーが少なくとも1行を返す場合、`EXISTS subquery` は `TRUE` であり、`NOT EXISTS subquery` は `FALSE` です。例:

```
SELECT column1 FROM t1 WHERE EXISTS (SELECT * FROM t2);
```

従来より、`EXISTS` サブクエリーは `SELECT *` で始まりますが、`SELECT 5` や `SELECT column1`、あるいはほかの何で始まってもかまいません。MySQLはこのようなサブクエリー内の `SELECT` リストを無視するため、何も違いは生まれません。

前の例では、`t2` に何らかの行が含まれている場合 (`NULL` 値以外は何も含まれていない行でも)、`EXISTS` 条件は `TRUE` です。`[NOT] EXISTS` サブクエリーには、ほぼ常に相互関係が含まれるため、これは実際にはありそうにもない例です。次に、より現実的な例をいくつか示します。

- 1つ以上の市に存在するのはどのような種類のお店ですか?

```
SELECT DISTINCT store_type FROM stores
WHERE EXISTS (SELECT * FROM cities_stores
WHERE cities_stores.store_type = stores.store_type);
```

- どの市にも存在しないのはどのような種類のお店ですか?

```
SELECT DISTINCT store_type FROM stores
WHERE NOT EXISTS (SELECT * FROM cities_stores
WHERE cities_stores.store_type = stores.store_type);
```

- すべての市に存在するのはどのような種類のお店ですか?

```
SELECT DISTINCT store_type FROM stores s1
WHERE NOT EXISTS (
SELECT * FROM cities WHERE NOT EXISTS (
SELECT * FROM cities_stores
WHERE cities_stores.city = cities.city
AND cities_stores.store_type = stores.store_type));
```

最後の例は、二重にネストされた `NOT EXISTS` クエリーです。つまり、`NOT EXISTS` 句の中に `NOT EXISTS` 句が存在します。これは正式には、「Stores がないお店が含まれている市は存在しますか?」という質問に答えます。ただし、ネストされた `NOT EXISTS` が、「x はすべての y に対して `TRUE` ですか?」という質問に答えるという方が簡単です。

MySQL 8.0.19 以降では、次のように、**NOT EXISTS** または **NOT EXISTS** を **TABLE** とともにサブクエリーで使用することもできます:

```
SELECT column1 FROM t1 WHERE EXISTS (TABLE t2);
```

結果は、サブクエリーで **WHERE** 句を指定せずに **SELECT *** を使用した場合と同じです。

13.2.11.7 関連サブクエリー

関連サブクエリーは、外部クエリーにも現れるテーブルへの参照を含むサブクエリーです。例:

```
SELECT * FROM t1
WHERE column1 = ANY (SELECT column1 FROM t2
WHERE t2.column2 = t1.column2);
```

このサブクエリーには、サブクエリーの **FROM** 句でテーブル **t1** が指定されていない場合でも、**t1** のカラムへの参照が含まれます。そのため、MySQL はこのサブクエリーの外部を探し、外部クエリー内の **t1** を見つけます。

テーブル **t1** に **column1 = 5** かつ **column2 = 6** である行が含まれている一方、テーブル **t2** に **column1 = 5** かつ **column2 = 7** である行が含まれているとします。単純な式 **... WHERE column1 = ANY (SELECT column1 FROM t2)** は **TRUE** になりますが、この例では、サブクエリー内の **WHERE** 句は **((5,6)** が **(5,7)** に等しくないため) **FALSE** です。そのため、全体としての式は **FALSE** です。

スコープルール: MySQL は、内部から外部に評価します。例:

```
SELECT column1 FROM t1 AS x
WHERE x.column1 = (SELECT column1 FROM t2 AS x
WHERE x.column1 = (SELECT column1 FROM t3
WHERE x.column2 = t3.column1));
```

このステートメントでは、**SELECT column1 FROM t2 AS x ...** が **t2** の名前を変更するため、**x.column2** はテーブル **t2** 内のカラムである必要があります。**SELECT column1 FROM t1 ...** がさらに外部にある外部クエリーであるため、これはテーブル **t1** 内のカラムではありません。

13.2.11.8 導出テーブル

このセクションでは、導出テーブルの一般的な特性について説明します。**LATERAL** キーワードで始まるラテラル導出テーブルの詳細は、[セクション13.2.11.9「ラテラル導出テーブル」](#) を参照してください。

導出テーブルは、クエリーの **FROM** 句の有効範囲内にテーブルを生成する式です。たとえば、**SELECT** ステートメントの **FROM** 句のサブクエリーは導出テーブルです:

```
SELECT ... FROM (subquery) [AS] tbl_name ...
```

JSON_TABLE() 関数は、テーブルを生成し、導出テーブルを作成する別の方法を提供します:

```
SELECT * FROM JSON_TABLE(arg_list) [AS] tbl_name ...
```

FROM 句のすべてのテーブルに名前が必要なため、**[AS] tbl_name** 句は必須です。導出テーブルのカラムには一意の名前を付ける必要があります。または、**tbl_name** の後に、導出テーブルのカラムの名前をカッコで囲んだリストを続けることもできます:

```
SELECT ... FROM (subquery) [AS] tbl_name (col_list) ...
```

カラム名の数は、テーブルのカラムの数と同じである必要があります。

説明のために、次のテーブルがあるとします。

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5), s3 FLOAT);
```

このテーブルの例を使用して、**FROM** 句内のサブクエリーを使用する方法を次に示します。

```
INSERT INTO t1 VALUES (1,'1',1.0);
```



```
INSERT INTO t1 VALUES (2,'2',2.0);
SELECT sb1,sb2,sb3
FROM (SELECT s1 AS sb1, s2 AS sb2, s3*2 AS sb3 FROM t1) AS sb
WHERE sb1 > 1;
```

結果:

```
+-----+-----+-----+
| sb1 | sb2 | sb3 |
+-----+-----+-----+
| 2 | 2 | 4 |
+-----+-----+-----+
```

次に別の例を示します。グループ化されたテーブルに関する一連の合計の平均を知りたいとします。次のクエリーは機能しません。

```
SELECT AVG(SUM(column1)) FROM t1 GROUP BY column1;
```

ただし、次のクエリーは目的の情報を提供します。

```
SELECT AVG(sum_column1)
FROM (SELECT SUM(column1) AS sum_column1
FROM t1 GROUP BY column1) AS t1;
```

サブクエリー内で使用されているカラム名 (`sum_column1`) は外部クエリーで認識されます。

導出テーブルのカラム名は、その選択リストから取得されます:

```
mysql> SELECT * FROM (SELECT 1, 2, 3, 4) AS dt;
+-----+-----+-----+
| 1 | 2 | 3 | 4 |
+-----+-----+-----+
| 1 | 2 | 3 | 4 |
+-----+-----+-----+
```

カラム名を明示的に指定するには、導出テーブル名の後にカラム名のカッコ付きリストを付けます:

```
mysql> SELECT * FROM (SELECT 1, 2, 3, 4) AS dt (a, b, c, d);
+-----+-----+-----+
| a | b | c | d |
+-----+-----+-----+
| 1 | 2 | 3 | 4 |
+-----+-----+-----+
```

導出テーブルは、スカラー、カラム、行またはテーブルを戻すことができます。

導出テーブルには、次の制限事項があります:

- 導出テーブルには、同じ `SELECT` の他のテーブルへの参照を含めることはできません (そのためには `LATERAL` 導出テーブルを使用します。 [セクション13.2.11.9「ラテラル導出テーブル」](#) を参照)。
- MySQL 8.0.14 より前は、導出テーブルに外部参照を含めることはできません。これは、SQL 標準の制限ではなく、MySQL 8.0.14 で削除された MySQL 制限です。たとえば、次のクエリーの導出テーブル `dt` には、外部クエリーのテーブル `t1` への参照 `t1.b` が含まれています:

```
SELECT * FROM t1
WHERE t1.d > (SELECT AVG(dt.a)
FROM (SELECT SUM(t2.a) AS a
FROM t2
WHERE t2.b = t1.b GROUP BY t2.c) dt
WHERE dt.a > 10);
```

クエリーは、MySQL 8.0.14 以上で有効です。8.0.14 より前は、エラーが生成されます: `Unknown column 't1.b' in 'where clause'`

オプティマイザは、導出テーブルに関する情報を、`EXPLAIN` が導出テーブルを実体化する必要がないように決定します。 [セクション8.2.2.4「マージまたは実体化を使用した導出テーブル、ビュー参照および共通テーブル式の最適化」](#) を参照してください。

特定の状況下では、**EXPLAIN SELECT** を使用してテーブルデータを変更できます。これは、外部クエリーがいずれかのテーブルにアクセスし、内部クエリーが、テーブルの 1 つ以上の行を変更するストアドファンクションを呼び出す場合に発生する可能性があります。データベース `d1` に 2 つのテーブル `t1` および `t2` があり、次に示すように `t2` を変更するストアドファンクション `f1` が作成されているとします：

```
CREATE DATABASE d1;
USE d1;
CREATE TABLE t1 (c1 INT);
CREATE TABLE t2 (c1 INT);
CREATE FUNCTION f1(p1 INT) RETURNS INT
BEGIN
  INSERT INTO t2 VALUES (p1);
  RETURN p1;
END;
```

次に示すように、**EXPLAIN SELECT** で関数を直接参照しても、`t2` には影響しません：

```
mysql> SELECT * FROM t2;
Empty set (0.02 sec)

mysql> EXPLAIN SELECT f1(5)\G
***** 1. row *****
   id: 1
  select_type: SIMPLE
    table: NULL
  partitions: NULL
     type: NULL
possible_keys: NULL
    key: NULL
   key_len: NULL
     ref: NULL
    rows: NULL
   filtered: NULL
  Extra: No tables used
1 row in set (0.01 sec)

mysql> SELECT * FROM t2;
Empty set (0.01 sec)
```

これは、出力の `table` および `Extra` カラムでわかるように、**SELECT** ステートメントがどのテーブルも参照しなかったためです。これはまた、次のネストされた **SELECT** にも当てはまります。

```
mysql> EXPLAIN SELECT NOW() AS a1, (SELECT f1(5)) AS a2\G
***** 1. row *****
   id: 1
  select_type: PRIMARY
    table: NULL
     type: NULL
possible_keys: NULL
    key: NULL
   key_len: NULL
     ref: NULL
    rows: NULL
   filtered: NULL
  Extra: No tables used
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Note | 1249 | Select 2 was reduced during optimization |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT * FROM t2;
Empty set (0.00 sec)
```

ただし、外部 **SELECT** がいずれかのテーブルを参照する場合、オプティマイザはサブクエリーのステートメントも実行し、`t2` が変更されます：

```
mysql> EXPLAIN SELECT * FROM t1 AS a1, (SELECT f1(5)) AS a2\G
```

```

***** 1. row *****
  id: 1
  select_type: PRIMARY
  table: <derived2>
  partitions: NULL
  type: system
  possible_keys: NULL
  key: NULL
  key_len: NULL
  ref: NULL
  rows: 1
  filtered: 100.00
  Extra: NULL
***** 2. row *****
  id: 1
  select_type: PRIMARY
  table: a1
  partitions: NULL
  type: ALL
  possible_keys: NULL
  key: NULL
  key_len: NULL
  ref: NULL
  rows: 1
  filtered: 100.00
  Extra: NULL
***** 3. row *****
  id: 2
  select_type: DERIVED
  table: NULL
  partitions: NULL
  type: NULL
  possible_keys: NULL
  key: NULL
  key_len: NULL
  ref: NULL
  rows: NULL
  filtered: NULL
  Extra: No tables used
3 rows in set (0.00 sec)

mysql> SELECT * FROM t2;
+-----+
| c1 |
+-----+
| 5 |
+-----+
1 row in set (0.00 sec)

```

これはまた、次に示すような `EXPLAIN SELECT` ステートメントは、`t1` 内の行ごとに 1 回 `BENCHMARK()` 関数が実行されるため、実行に長い時間がかかる可能性があることも示しています。

```
EXPLAIN SELECT * FROM t1 AS a1, (SELECT BENCHMARK(1000000, MD5(NOW())));
```

13.2.11.9 ラテラル導出テーブル

導出テーブルは、通常、同じ `FROM` 句内の前述のテーブルのカラムを参照 (依存) することはできません。MySQL 8.0.14 では、導出テーブルを横導出テーブルとして定義して、このような参照が許可されるように指定できます。

非ラテラル導出テーブルは、[セクション13.2.11.8「導出テーブル」](#) で説明されている構文を使用して指定します。ラテラル導出テーブルの構文は、導出テーブルの指定の前にキーワード `LATERAL` が指定されている点を除き、非ラテラル導出テーブルの構文と同じです。`LATERAL` キーワードは、ラテラル導出テーブルとして使用される各テーブルの前に指定する必要があります。

ラテラル導出テーブルには、次の制限事項があります:

- ラテラル導出テーブルは、カンマで区切られたテーブルのリストまたは結合指定 (`JOIN`, `INNER JOIN`, `CROSS JOIN`, `LEFT [OUTER] JOIN` または `RIGHT [OUTER] JOIN`) のいずれかで、`FROM` 句でのみ使用できます。
- ラテラル導出テーブルが結合句の右オペランドにあり、左オペランドへの参照が含まれている場合、結合操作は `INNER JOIN`、`CROSS JOIN` または `LEFT [OUTER] JOIN` である必要があります。

テーブルが左オペランドにあり、右オペランドへの参照が含まれている場合、結合操作は `INNER JOIN`、`CROSS JOIN` または `RIGHT [OUTER] JOIN` である必要があります。

- ラテラル導出テーブルが集計関数を参照する場合、関数集計クエリーを、ラテラル導出テーブルが発生する `FROM` 句を所有するクエリーにすることはできません。
- SQL 標準に従って、テーブル関数には暗黙的な `LATERAL` があるため、8.0.14 より前の MySQL 8.0 バージョンと同様に動作します。ただし、標準に従って、`LATERAL` ワードは暗黙的であっても `JSON_TABLE()` の前には許可されません。

次の説明では、潜在的導出テーブルによって、非潜在的導出テーブルで実行できない特定の SQL 操作や、より効率的な回避策を必要とする特定の SQL 操作がどのように行われるかを示します。

この問題を解決するとします: 営業部隊内の個人のテーブル (各行に販売部隊のメンバーが記述されている) と、すべての売上のテーブル (各行に販売が記述されている) があるとします: 営業担当、顧客、金額、日付) は、各営業担当の最大販売の規模と顧客を決定します。この問題には 2 つの方法があります。

問題を解決する最初のアプローチ: 各営業担当について、最大販売サイズを計算し、この最大値を指定した顧客も検索します。MySQL では、次のように実行できます:

```
SELECT
  salesperson.name,
  -- find maximum sale size for this salesperson
  (SELECT MAX(amount) AS amount
   FROM all_sales
   WHERE all_sales.salesperson_id = salesperson.id)
  AS amount,
  -- find customer for this maximum size
  (SELECT customer_name
   FROM all_sales
   WHERE all_sales.salesperson_id = salesperson.id
   AND all_sales.amount =
     -- find maximum size, again
     (SELECT MAX(amount) AS amount
      FROM all_sales
      WHERE all_sales.salesperson_id = salesperson.id))
  AS customer_name
FROM
  salesperson;
```

このクエリーでは、営業担当ごとに最大サイズが 2 回 (最初のサブクエリーで 1 回、2 回目) 計算されるため、非効率的です。

次の変更されたクエリーに示すように、営業担当ごとに最大数を計算し、それを導出テーブルで「「キャッシュ」」することで、効率性向上を試みることができます:

```
SELECT
  salesperson.name,
  max_sale.amount,
  max_sale_customer.customer_name
FROM
  salesperson,
  -- calculate maximum size, cache it in transient derived table max_sale
  (SELECT MAX(amount) AS amount
   FROM all_sales
   WHERE all_sales.salesperson_id = salesperson.id)
  AS max_sale,
  -- find customer, reusing cached maximum size
  (SELECT customer_name
   FROM all_sales
   WHERE all_sales.salesperson_id = salesperson.id
   AND all_sales.amount =
     -- the cached maximum size
     max_sale.amount)
  AS max_sale_customer;
```

ただし、導出テーブルは同じ `FROM` 句の他のテーブルに依存できないため、SQL-92 ではクエリーは無効です。導出テーブルは、クエリー期間中は一定である必要があり、他の `FROM` 句テーブルのカラムへの参照は含まれません。前述のとおり、クエリーでは次のエラーが生成されます:

```
ERROR 1054 (42S22): Unknown column 'salesperson.id' in 'where clause'
```

SQL:1999 では、導出テーブルの前に **LATERAL** キーワード (「この導出テーブルは左側の前のテーブルに依存しています」) を意味する) がある場合、クエリーは有効になります:

```
SELECT
  salesperson.name,
  max_sale.amount,
  max_sale_customer.customer_name
FROM
  salesperson,
  -- calculate maximum size, cache it in transient derived table max_sale
  LATERAL
  (SELECT MAX(amount) AS amount
   FROM all_sales
   WHERE all_sales.salesperson_id = salesperson.id)
  AS max_sale,
  -- find customer, reusing cached maximum size
  LATERAL
  (SELECT customer_name
   FROM all_sales
   WHERE all_sales.salesperson_id = salesperson.id
   AND all_sales.amount =
     -- the cached maximum size
     max_sale.amount)
  AS max_sale_customer;
```

ラテラル導出テーブルは定数である必要はなく、それが依存する前のテーブルの新しい行が最上位のクエリーによって処理されるたびに最新になります。

問題を解決するための第 2 のアプローチ: **SELECT** リストのサブクエリーが複数のカラムを返す可能性がある場合は、別の解決策を使用できます:

```
SELECT
  salesperson.name,
  -- find maximum size and customer at same time
  (SELECT amount, customer_name
   FROM all_sales
   WHERE all_sales.salesperson_id = salesperson.id
   ORDER BY amount DESC LIMIT 1)
FROM
  salesperson;
```

これは効率的ですが、不正です。このようなサブクエリーは単一のカラムのみを返すことができるため、機能しません:

```
ERROR 1241 (21000): Operand should contain 1 column(s)
```

クエリーをリライトするには、導出テーブルから複数のカラムを選択します:

```
SELECT
  salesperson.name,
  max_sale.amount,
  max_sale.customer_name
FROM
  salesperson,
  -- find maximum size and customer at same time
  (SELECT amount, customer_name
   FROM all_sales
   WHERE all_sales.salesperson_id = salesperson.id
   ORDER BY amount DESC LIMIT 1)
  AS max_sale;
```

ただし、これも機能しません。導出テーブルは **salesperson** テーブルに依存しているため、**LATERAL** なしで失敗します:

```
ERROR 1054 (42S22): Unknown column 'salesperson.id' in 'where clause'
```

LATERAL キーワードを追加すると、クエリーは有効になります:

```
SELECT
```

```

salesperson.name,
max_sale.amount,
max_sale.customer_name
FROM
salesperson,
-- find maximum size and customer at same time
LATERAL
(SELECT amount, customer_name
FROM all_sales
WHERE all_sales.salesperson_id = salesperson.id
ORDER BY amount DESC LIMIT 1)
AS max_sale;

```

つまり、**LATERAL** は、前述の 2 つのアプローチにおけるすべての欠点に対する効率的なソリューションです。

13.2.11.10 サブクエリーのエラー

サブクエリーにのみ適用されるエラーがいくつか存在します。このセクションでは、これらについて説明します。

- サポートされていないサブクエリー構文:

```

ERROR 1235 (ER_NOT_SUPPORTED_YET)
SQLSTATE = 42000
Message = "This version of MySQL doesn't yet support
'LIMIT & IN/ALL/ANY/SOME subquery'"

```

つまり、MySQL は次のようなステートメントをサポートしていません:

```
SELECT * FROM t1 WHERE s1 IN (SELECT s2 FROM t2 ORDER BY s1 LIMIT 1)
```

- サブクエリーからの正しくないカラム数:

```

ERROR 1241 (ER_OPERAND_COL)
SQLSTATE = 21000
Message = "Operand should contain 1 column(s)"

```

このエラーは、次のような場合に発生します。

```
SELECT (SELECT column1, column2 FROM t2) FROM t1;
```

目的が行の比較である場合は、複数のカラムを返すサブクエリーを使用できます。ほかのコンテキストでは、サブクエリーはスカラーオペランドである必要があります。[セクション13.2.11.5「行サブクエリー」](#)を参照してください。

- サブクエリーからの正しくない行数:

```

ERROR 1242 (ER_SUBSELECT_NO_1_ROW)
SQLSTATE = 21000
Message = "Subquery returns more than 1 row"

```

このエラーは、サブクエリーが最大で 1 行しか返す必要がないにもかかわらず、複数の行を返すステートメントで発生します。次の例を考えてみます。

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
```

SELECT column1 FROM t2 が単一行のみを返す場合、前のクエリーは機能します。サブクエリーが複数の行を返す場合、エラー 1242 が発生します。その場合は、このクエリーを次のように書き換えてください。

```
SELECT * FROM t1 WHERE column1 = ANY (SELECT column1 FROM t2);
```

- サブクエリー内の誤って使用されているテーブル:

```

Error 1093 (ER_UPDATE_TABLE_USED)
SQLSTATE = HY000
Message = "You can't specify target table 'x'
for update in FROM clause"

```

このエラーは、テーブルを変更し、さらにサブクエリーで同じテーブルから選択しようとする次のような場合に発生します。


```
UPDATE t1 SET column2 = (SELECT MAX(column1) FROM t1);
```

これを回避するには、共通テーブル式または導出テーブルを使用できます。 [セクション13.2.11.12「サブクエリーの制約」](#)を参照してください。

MySQL 8.0.19 以降では、このセクションで説明するすべてのエラーは、サブクエリーで `TABLE` を使用する場合にも適用されます。

トランザクションストレージエンジンの場合は、サブクエリーが失敗するとステートメント全体が失敗します。非トランザクションストレージエンジンの場合は、エラーが検出される前に行われたデータ変更が保持されます。

13.2.11.11 サブクエリーの最適化

開発が進行中であるため、長期にわたって信頼できる最適化のヒントはありません。次のリストに、再生に役立つ興味深いトリックをいくつか示します。 [セクション8.2.2「サブクエリー、導出テーブル、ビュー参照および共通テーブル式の最適化」](#)も参照してください。

- 句をサブクエリーの外部から内部に移動します。たとえば、次のクエリーを使用してください:

```
SELECT * FROM t1
WHERE s1 IN (SELECT s1 FROM t1 UNION ALL SELECT s1 FROM t2);
```

次のクエリーの代替として:

```
SELECT * FROM t1
WHERE s1 IN (SELECT s1 FROM t1) OR s1 IN (SELECT s1 FROM t2);
```

別の例として、このクエリーを使用してください:

```
SELECT (SELECT column1 + 5 FROM t1) FROM t2;
```

次のクエリーの代替として:

```
SELECT (SELECT column1 FROM t1) + 5 FROM t2;
```

13.2.11.12 サブクエリーの制約

- 一般に、テーブルを変更することも、サブクエリーの同じテーブルから選択することもできません。たとえば、この制限は次の形式のステートメントに適用されます。

```
DELETE FROM t WHERE ... (SELECT ... FROM t ...);
UPDATE t ... WHERE col = (SELECT ... FROM t ...);
{INSERT|REPLACE} INTO t (SELECT ... FROM t ...);
```

例外: 前述の禁止は、導出テーブルを使用している変更されたテーブルで、その導出テーブルが外部クエリーにマージされるのではなく実体化されている場合には適用されません。([セクション8.2.2.4「マージまたは実体化を使用した導出テーブル、ビュー参照および共通テーブル式の最適化」](#)を参照してください。) 例:

```
UPDATE t ... WHERE col = (SELECT * FROM (SELECT ... FROM t...) AS dt ...);
```

導出テーブルからの結果は一時テーブルとして実体化されるため、`t` への更新が行われるまでに、`t` 内の関連する行がすでに選択されています。

一般に、`NO_MERGE` オプティマイザヒントを追加することで、導出テーブルを実体化するオプティマイザに影響を与えることができます。 [セクション8.9.3「オプティマイザヒント」](#)を参照してください。

- 行比較演算は一部のみサポートされています。
 - `expr [NOT] IN subquery` の場合、`expr` は `n` タプル (行コンストラクタ構文を使用して指定します) にでき、サブクエリーは `n` タプルの行を返すことができます。したがって、許可されている構文は、具体的には `row_constructor [NOT] IN table_subquery` と表されます
 - `expr op {ALL|ANY|SOME} subquery` の場合、`expr` はスカラー値にする必要があり、サブクエリーはカラムサブクエリーにする必要があります。複合カラム行を返すことはできません。

つまり、`n` タプルの行を返すサブクエリーの場合、次のものはサポートされています。

```
(expr_1, ..., expr_n) [NOT] IN table_subquery
```

ただし、次のものはサポートされていません。

```
(expr_1, ..., expr_n) op {ALL|ANY|SOME} subquery
```

`IN` の行比較がサポートされているのに、他はサポートされていない理由は、`IN` が、`=` 比較および `AND` 演算のシーケンスに、これを書き換えることによって実装されているためです。この方法は、`ALL`、`ANY`、`SOME` には使用できません。

- MySQL 8.0.14 より前は、`FROM` 句のサブクエリーを相関サブクエリーにすることはできません。これらは、クエリー実行中にすべて実体化 (結果セットを生成するように評価) されるので、外部クエリーの行ごとに評価できません。オプティマイザは、結果が必要になるまで実体化を遅延します。これにより、実体化を回避できます。[セクション 8.2.2.4 「マージまたは実体化を使用した導出テーブル、ビュー参照および共通テーブル式の最適化」](#) を参照してください。
- MySQL は特定のサブクエリー演算子にサブクエリーの `LIMIT` をサポートしていません。

```
mysql> SELECT * FROM t1
  WHERE s1 IN (SELECT s2 FROM t2 ORDER BY s1 LIMIT 1);
ERROR 1235 (42000): This version of MySQL doesn't yet support
'LIMIT & IN/ALL/ANY/SOME subquery'
```

[セクション 13.2.11.10 「サブクエリーのエラー」](#) を参照してください。

- MySQL では、サブクエリーで行をテーブルに挿入するなどのデータ変更の副作用があるストアードファンクションを参照できます。たとえば、`f()` が行を挿入する場合、次のクエリーはデータを変更できます。

```
SELECT ... WHERE x IN (SELECT f() ...);
```

この動作は、SQL 標準に対する拡張です。MySQL では、オプティマイザによる処理の選択方法に応じて、特定のクエリーの実行ごとに `f()` が異なる回数実行される可能性があるため、非決定的な結果が生成される可能性があります。

ステートメントベースレプリケーションまたは混合形式レプリケーションの場合、このような不確定の影響の 1 つは、このようなクエリーがソースとそのスレーブで異なる結果を生成できることです。

13.2.12 TABLE ステートメント

`TABLE` は、MySQL 8.0.19 で導入された DML ステートメントで、指定されたテーブルの行とカラムを返します。

```
TABLE table_name [ORDER BY column_name] [LIMIT number [OFFSET number]]
```

`TABLE` ステートメントは、いくつかの方法で `SELECT` のように動作します。`t` という名前のテーブルが存在する場合、次の 2 つのステートメントによって同一の出力が生成されます:

```
TABLE t;
SELECT * FROM t;
```

`ORDER BY` 句および `LIMIT` 句をそれぞれ使用して、`TABLE` によって生成される行数を順序付けおよび制限できます。これらの関数は、次に示すように、`SELECT` で使用する場合と同じ句 (`LIMIT` でのオプションの `OFFSET` 句を含む) と同じように機能します:

```
mysql> TABLE t;
+----+----+
| a | b |
+----+----+
| 1 | 2 |
| 6 | 7 |
| 9 | 5 |
|10 | -4 |
|11 | -1 |
```

```
| 13 | 3 |
| 14 | 6 |
+----+
7 rows in set (0.00 sec)

mysql> TABLE t ORDER BY b;
+----+
| a | b |
+----+
| 10 | -4 |
| 11 | -1 |
| 1 | 2 |
| 13 | 3 |
| 9 | 5 |
| 14 | 6 |
| 6 | 7 |
+----+
7 rows in set (0.00 sec)

mysql> TABLE t LIMIT 3;
+----+
| a | b |
+----+
| 1 | 2 |
| 6 | 7 |
| 9 | 5 |
+----+
3 rows in set (0.00 sec)

mysql> TABLE t ORDER BY b LIMIT 3;
+----+
| a | b |
+----+
| 10 | -4 |
| 11 | -1 |
| 1 | 2 |
+----+
3 rows in set (0.00 sec)

mysql> TABLE t ORDER BY b LIMIT 3 OFFSET 2;
+----+
| a | b |
+----+
| 1 | 2 |
| 13 | 3 |
| 9 | 5 |
+----+
3 rows in set (0.00 sec)
```

TABLE は、主に次の点で SELECT と異なります:

- TABLE では、常にテーブルのすべてのコラムが表示されます。
- TABLE では、行の任意のフィルタリングは許可されません。つまり、TABLE では WHERE 句はサポートされません。

返されるテーブルのコラムを制限するには、ORDER BY および LIMIT(あるいはその両方) を使用して達成できる以上の行をフィルタリングし、SELECT を使用します。

TABLE は、一時テーブルとともに使用できます。

TABLE は、SELECT のかわりに、ここにリストされているものを含め、他の多くの構成要素で使用することもできます:

- 次に示すように、UNION を使用します:

```
mysql> TABLE t1;
+----+
| a | b |
+----+
| 2 | 10 |
| 5 | 3 |
```

```
| 7 | 8 |
+---+---+
3 rows in set (0.00 sec)
```

```
mysql> TABLE t2;
+---+---+
| a | b |
+---+---+
| 1 | 2 |
| 3 | 4 |
| 6 | 7 |
+---+---+
3 rows in set (0.00 sec)
```

```
mysql> TABLE t1 UNION TABLE t2;
+---+---+
| a | b |
+---+---+
| 2 | 10 |
| 5 | 3 |
| 7 | 8 |
| 1 | 2 |
| 3 | 4 |
| 6 | 7 |
+---+---+
6 rows in set (0.00 sec)
```

ここで示した **UNION** は、次のステートメントと同等です:

```
mysql> SELECT * FROM t1 UNION SELECT * FROM t2;
+---+---+
| a | b |
+---+---+
| 2 | 10 |
| 5 | 3 |
| 7 | 8 |
| 1 | 2 |
| 3 | 4 |
| 6 | 7 |
+---+---+
6 rows in set (0.00 sec)
```

TABLE は、**SELECT** ステートメントまたは **VALUES** ステートメント (あるいはその両方) と組み合わせて使用することもできます。 [セクション13.2.10.3「UNION 句」](#) を参照してください。

- **INTO** を使用してユーザー変数を移入し、**INTO OUTFILE** または **INTO DUMPFILE** を使用してテーブルデータをファイルに書き込みます。詳細および例は、[セクション13.2.10.1「SELECT ... INTO ステートメント」](#) を参照してください。
- 多くの場合、サブクエリーを使用できます。 **a** という名前のカラムを持つ任意のテーブル **t1** と、単一のカラムを持つ別のテーブル **t2** がある場合、次のようなステートメントが可能です:

```
SELECT * FROM t1 WHERE a IN (TABLE t2);
```

テーブル **ts** の単一カラムの名前が **x** であると仮定すると、前述のステートメントは次に示す各ステートメントと同等です (いずれの場合もまったく同じ結果が生成されます):

```
SELECT * FROM t1 WHERE a IN (SELECT x FROM t2);
```

```
SELECT * FROM t1 WHERE a IN (SELECT * FROM t2);
```

詳しくは [セクション13.2.11「サブクエリー」](#) ,をご覧ください。

- **INSERT** および **REPLACE** ステートメントを使用します。それ以外の場合は、**SELECT *** を使用します。詳細および例については、[セクション13.2.6.1「INSERT ... SELECT ステートメント」](#) を参照してください。
- **TABLE** は、多くの場合、**CREATE TABLE ... SELECT** または **CREATE VIEW ... SELECT** の **SELECT** のかわりに使用することもできます。詳細および例については、これらのステートメントの説明を参照してください。

13.2.13 UPDATE ステートメント

UPDATE は、テーブルの行を変更する DML ステートメントです。

UPDATE ステートメントは、WITH 句で始まり、UPDATE 内でアクセス可能な共通テーブル式を定義できます。セクション13.2.15「WITH (共通テーブル式)」を参照してください。

単一テーブル構文:

```
UPDATE [LOW_PRIORITY] [IGNORE] table_reference
SET assignment_list
[WHERE where_condition]
[ORDER BY ...]
[LIMIT row_count]

value:
{expr | DEFAULT}

assignment:
col_name = value

assignment_list:
assignment [, assignment] ...
```

複数テーブル構文:

```
UPDATE [LOW_PRIORITY] [IGNORE] table_references
SET assignment_list
[WHERE where_condition]
```

単一テーブル構文の場合、UPDATE ステートメントは、指定されたテーブル内の既存の行の列を新しい値に更新します。SET 句は、変更する列と、それらの列に指定される値を示します。各値は式か、または列を明示的にそのデフォルト値に設定するキーワード DEFAULT として指定できます。WHERE 句 (指定されている場合) は、どの行を更新するかを識別する条件を指定します。WHERE 句がない場合は、すべての行が更新されます。ORDER BY 句が指定されている場合は、指定されている順序で行が更新されます。LIMIT 句は、更新できる行数に制限を設定します。

複数テーブル構文の場合、UPDATE は、条件を満たす table_references で指定されている各テーブル内の行を更新します。一致した各行は、条件に複数回一致した場合でも、1 回更新されます。複数テーブル構文の場合は、ORDER BY および LIMIT を使用できません。

パーティション化されたテーブルの場合は、このステートメントの単一テーブルと複数テーブルの両方の形式で、テーブル参照の一部としての PARTITION オプションの使用がサポートされます。このオプションは、1 つ以上のパーティションまたはサブパーティション (またはその両方) のリストを受け取ります。リストされているパーティション (またはサブパーティション) だけが一致をチェックされ、これらのパーティションまたはサブパーティションのいずれにも存在しない行は、where_condition を満たすかどうかにかかわらず更新されません。

注記

INSERT または REPLACE ステートメントで PARTITION を使用している場合とは異なり、それ以外は有効な UPDATE ... PARTITION ステートメントは、リストされているパーティション (またはサブパーティション) 内のどの行も where_condition に一致しない場合でも成功したと見なされます。

詳細および例については、セクション24.5「パーティション選択」を参照してください。

where_condition は、更新される各行に対して true に評価される式です。式の構文については、セクション9.5「式」を参照してください。

table_references と where_condition は、セクション13.2.10「SELECT ステートメント」で説明されているように指定されます。

実際に更新された、UPDATE 内で参照されている列に対してのみ UPDATE 権限が必要です。読み取られるが、変更されない列に対しては、SELECT 権限のみが必要です。

UPDATE ステートメントは、次の修飾子をサポートします。

- **LOW_PRIORITY** 修飾子を使用すると、ほかのクライアントがテーブルから読み取ることがなくなるまで、**UPDATE** の実行が遅延されます。これは、テーブルレベルロックのみを使用するストレージエンジン (**MyISAM**、**MEMORY**、および **MERGE**) にのみ影響を与えます。
- **IGNORE** 修飾子を使用すると、更新中にエラーが発生しても更新ステートメントは中断されません。一意のキー値に関して重複キーの競合が発生した行は更新されません。データ変換エラーの原因になる値に更新された行は、代わりに、もっとも近い有効な値に更新されます。詳細は、[IGNORE がステートメントの実行に与える影響](#)を参照してください。

ORDER BY 句を持つステートメントを含む **UPDATE IGNORE** ステートメントには、ステートメントベースのレプリケーションに対して安全でないというフラグが付けられます。(これは、どの行が無視されるかが、行が更新される順序によって決定されるためです。) このようなステートメントは、ステートメントベースのモードの使用時にエラーログに警告を生成し、**MIXED** モードの使用時に行ベースの形式を使用してバイナリログに書き込まれます。(Bug #11758262、Bug #50439) 詳細は、[セクション17.2.1.3「バイナリロギングでの安全および安全でないステートメントの判断」](#)を参照してください。

式で更新されるテーブルのカラムにアクセスする場合、**UPDATE** はそのカラムの現在の値を使用します。たとえば、次のステートメントは、**col1** をその現在の値より 1 大きい値に設定します。

```
UPDATE t1 SET col1 = col1 + 1;
```

次のステートメントの 2 番目の割り当ては、**col2** を元の **col1** 値ではなく、現在の (更新された) **col1** 値に設定します。この結果、**col1** と **col2** の値が同じになります。この動作は標準 SQL とは異なります。

```
UPDATE t1 SET col1 = col1 + 1, col2 = col1;
```

単一テーブルの **UPDATE** の割り当ては一般に、左から右に評価されます。複数テーブルの更新では、割り当てが特定の順序で実行される保証はありません。

カラムをその現在の値に設定した場合は、MySQL がこれに気づき、その更新を行いません。

NOT NULL として宣言されているカラムを **NULL** に設定することによって更新すると、厳密な SQL モードが有効になっている場合は、エラーが発生します。そうでない場合、カラムはそのカラムデータ型の暗黙のデフォルト値に設定され、警告数が 1 増やされます。暗黙のデフォルト値は、数値型では 0、文字列型では空の文字列 ("")、および日付と時間型では「0」の値です。[セクション11.6「データ型デフォルト値」](#)を参照してください。

生成されたカラムが明示的に更新される場合、許可される値は **DEFAULT** のみです。生成されるカラムの詳細は、[セクション13.1.20.8「CREATE TABLE および生成されるカラム」](#)を参照してください。

UPDATE は、実際に変更された行数を返します。**mysql_info()** C API 関数は、一致して更新された行数と、**UPDATE** 中に発生した警告の数を返します。

LIMIT row_count を使用すると、**UPDATE** のスコープを制限できます。**LIMIT** 句は、一致した行の制限です。このステートメントは、実際に変更されたかどうかにかかわらず、**WHERE** 句を満たす **row_count** 行を見つけるとすぐに停止します。

UPDATE ステートメントに **ORDER BY** 句が含まれている場合は、この句で指定されている順序で行が更新されます。これは、通常であればエラーが発生する可能性のある特定の状況で役立つ場合があります。テーブル **t** に、一意のインデックスを持つカラム **id** が含まれているとします。次のステートメントは、行が更新される順序によっては、重複キーエラーで失敗する可能性があります。

```
UPDATE t SET id = id + 1;
```

たとえば、このテーブルの **id** カラムに 1 と 2 が含まれており、2 が 3 に更新される前に 1 が 2 に更新された場合は、エラーが発生します。この問題を回避するには、大きな **id** 値を持つ行が小さな値を持つ行の前に更新されるように、**ORDER BY** 句を追加します。

```
UPDATE t SET id = id + 1 ORDER BY id DESC;
```

また、複数のテーブルを範囲に含む **UPDATE** 操作を実行することもできます。ただし、複数テーブルの **UPDATE** では **ORDER BY** または **LIMIT** を使用できません。**table_references** 句は、結合に含まれるテーブルをリストします。その構文については、[セクション13.2.10.2「JOIN 句」](#)で説明されています。次に例を示します。

```
UPDATE items,month SET items.price=month.price
```



```
WHERE items.id=month.id;
```

前の例は、カンマ演算子を使用する内部結合を示していますが、複数テーブルの **UPDATE** ステートメントは、**SELECT** ステートメント内で許可されている任意の型の結合 (**LEFT JOIN** など) を使用できます。

外部キー制約が存在する **InnoDB** テーブルを含む、複数テーブルの **UPDATE** ステートメントを使用した場合は、MySQL オプティマイザが、それらの親子関係の順序とは異なる順序でテーブルを処理する可能性があります。この場合、このステートメントは失敗し、ロールバックされます。代わりに、1つのテーブルを更新したあと、**InnoDB** が提供する **ON UPDATE** 機能を使用して、ほかのテーブルがそれに応じて変更されるようにします。セクション13.1.20.5「**FOREIGN KEY** の制約」を参照してください。

テーブルを更新してサブクエリーと同じテーブルから直接選択することはできません。これを回避するには、複数テーブルの更新を使用します。この更新では、いずれかのテーブルが実際に更新するテーブルから導出され、エイリアスを使用して導出テーブルを参照します。次に示すステートメントを使用して定義された **items** という名前のテーブルを更新するとします:

```
CREATE TABLE items (  
  id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,  
  wholesale DECIMAL(6,2) NOT NULL DEFAULT 0.00,  
  retail DECIMAL(6,2) NOT NULL DEFAULT 0.00,  
  quantity BIGINT NOT NULL DEFAULT 0  
);
```

値入れが 30% 以上で在庫が 100 未満のアイテムの小売価格を下げるには、**WHERE** 句でサブクエリーを使用する次のような **UPDATE** ステートメントを使用します。次に示すように、このステートメントは機能しません:

```
mysql> UPDATE items  
  > SET retail = retail * 0.9  
  > WHERE id IN  
  >   (SELECT id FROM items  
  >    WHERE retail / wholesale >= 1.3 AND quantity > 100);  
ERROR 1093 (HY000): You can't specify target table 'items' for update in FROM clause
```

かわりに、次のように、最も外側の **WHERE** 句でエイリアスを使用して参照し、サブクエリーを更新対象のテーブルのリストに移動する複数テーブル更新を使用できます:

```
UPDATE items,  
  (SELECT id FROM items  
   WHERE id IN  
     (SELECT id FROM items  
      WHERE retail / wholesale >= 1.3 AND quantity < 100))  
  AS discounted  
SET items.retail = items.retail * 0.9  
WHERE items.id = discounted.id;
```

オプティマイザはデフォルトで導出テーブル **discounted** を最も外側のクエリーブロックにマージしようとするため、導出テーブルを強制的に実体化する場合にのみ機能します。これを行うには、更新を実行する前に **optimizer_switch** システム変数の **derived_merge** フラグを **off** に設定するか、次のように **NO_MERGE** オプティマイザヒントを使用します:

```
UPDATE /*+ NO_MERGE(discounted) */ items,  
  (SELECT id FROM items  
   WHERE retail / wholesale >= 1.3 AND quantity < 100)  
  AS discounted  
SET items.retail = items.retail * 0.9  
WHERE items.id = discounted.id;
```

このような場合にオプティマイザヒントを使用する利点は、オプティマイザヒントが使用されるクエリーブロック内でのみ適用されるため、**UPDATE** の実行後に **optimizer_switch** の値を再度変更する必要がないことです。

次のように、**IN** または **EXISTS** を使用しないようにサブクエリーをリライトすることもできます:

```
UPDATE items,  
  (SELECT id, retail / wholesale AS markup, quantity FROM items)  
  AS discounted  
SET items.retail = items.retail * 0.9  
WHERE discounted.markup >= 1.3  
  AND discounted.quantity < 100  
  AND items.id = discounted.id;
```

この場合、サブクエリーはマージではなくデフォルトで実体化されるため、導出テーブルのマージを無効にする必要はありません。

13.2.14 VALUES ステートメント

VALUES は、MySQL 8.0.19 で導入された DML ステートメントで、1 つ以上の行のセットをテーブルとして返します。つまり、スタンドアロン SQL ステートメントとしても機能するテーブル値コンストラクタです。

```
VALUES row_constructor_list [ORDER BY column_designator] [LIMIT BY number]
```

```
row_constructor_list:
  ROW(value_list)[, ROW(value_list)][, ...]
```

```
value_list:
  value[, value][, ...]
```

```
column_designator:
  column_index
```

VALUES ステートメントは、**VALUES** キーワードと、カンマで区切られた 1 つ以上の行コンストラクタのリストで構成されます。行コンストラクタは、カッコで囲まれた 1 つ以上のスカラー値の値リストを持つ **ROW()** 行コンストラクタ句で構成されます。値には、任意の MySQL データ型のリテラルまたはスカラー値に解決される式を使用できます。

ROW() は空にできません (ただし、指定された各スカラー値を **NULL** にすることはできます)。同じ **VALUES** ステートメントの各 **ROW()** の値リストには、同じ数の値が含まれている必要があります。

DEFAULT キーワードは **VALUES** でサポートされていないため、**INSERT** ステートメントで値を指定するために使用される場合を除き、構文エラーが発生します。

VALUES の出力はテーブルです:

```
mysql> VALUES ROW(1,-2,3), ROW(5,7,9), ROW(4,6,8);
+-----+-----+-----+
| column_0 | column_1 | column_2 |
+-----+-----+-----+
| 1 | -2 | 3 |
| 5 | 7 | 9 |
| 4 | 6 | 8 |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

VALUES からのテーブル出力のカラムには暗黙的に名前が付けられたカラム **column_0**, **column_1**, **column_2** などがあり、常に 0 で始まります。このファクトを使用すると、次に示すように、**SELECT** ステートメントと同じ方法で、オプションの **ORDER BY** 句を使用してカラムごとに行を並べ替えることができます:

```
mysql> VALUES ROW(1,-2,3), ROW(5,7,9), ROW(4,6,8) ORDER BY column_1;
+-----+-----+-----+
| column_0 | column_1 | column_2 |
+-----+-----+-----+
| 1 | -2 | 3 |
| 4 | 6 | 8 |
| 5 | 7 | 9 |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

VALUES ステートメントでは、出力の行数を制限するための **LIMIT** 句もサポートされています。

VALUES ステートメントは、カラム値のデータ型に関して許可されます。次に示すように、同じカラム内で型を混在させることができます:

```
mysql> VALUES ROW("q", 42, '2019-12-18'),
-> ROW(23, "abc", 98.6),
-> ROW(27.0002, "Mary Smith", '{"a": 10, "b": 25}');
+-----+-----+-----+
| column_0 | column_1 | column_2 |
+-----+-----+-----+
| q | 42 | 2019-12-18 |
```

```

23 | abc | 98.6 |
| 27.0002 | Mary Smith | {"a": 10, "b": 25} |
+-----+-----+
3 rows in set (0.00 sec)

```

重要

`ROW()` の 1 つ以上のインスタンスを持つ `VALUES` は、テーブル値コンストラクタとして機能します。`INSERT` ステートメントまたは `REPLACE` ステートメントで値を指定するために使用できますが、この目的でも使用される `VALUES` キーワードと混同しないでください。また、`INSERT ... ON DUPLICATE KEY UPDATE` のカラム値を参照する `VALUES()` 関数と混同しないでください。

`ROW()` は行値コンストラクタであることにも注意する必要があります ([セクション 13.2.11.5 「行サブクエリー」](#) を参照してくださいが、`VALUES ROW()` はテーブル値コンストラクタであり、両者を同じ意味で使用することはできません)。

`VALUES` は、ここにリストされているものを含め、`SELECT` を使用できる多くの場合に使用できます:

- 次に示すように、`UNION` を使用します:

```

mysql> SELECT 1,2 UNION SELECT 10,15;
+----+----+
| 1 | 2 |
+----+----+
| 1 | 2 |
| 10 | 15 |
+----+----+
2 rows in set (0.00 sec)

mysql> VALUES ROW(1,2) UNION VALUES ROW(10,15);
+-----+-----+
| column_0 | column_1 |
+-----+-----+
| 1 | 2 |
| 10 | 15 |
+-----+-----+
2 rows in set (0.00 sec)

```

この方法では、次のように、複数の行を持つ構築されたテーブルを結合することもできます:

```

mysql> VALUES ROW(1,2), ROW(3,4), ROW(5,6)
> UNION VALUES ROW(10,15), ROW(20,25);
+-----+-----+
| column_0 | column_1 |
+-----+-----+
| 1 | 2 |
| 3 | 4 |
| 5 | 6 |
| 10 | 15 |
| 20 | 25 |
+-----+-----+
5 rows in set (0.00 sec)

```

このような場合は、`UNION` を完全に省略して、次のような単一の `VALUES` ステートメントを使用することもできます (通常はこれをお勧めします):

```

mysql> VALUES ROW(1,2), ROW(3,4), ROW(5,6), ROW(10,15), ROW(20,25);
+-----+-----+
| column_0 | column_1 |
+-----+-----+
| 1 | 2 |
| 3 | 4 |
| 5 | 6 |
| 10 | 15 |
| 20 | 25 |
+-----+-----+

```

`VALUES` は、`SELECT` ステートメントまたは `TABLE` ステートメント (あるいはその両方) と組み合わせて使用することもできます。

UNION の構成されたテーブルには、SELECT を使用している場合と同様に、同じ数のカラムが含まれている必要があります。その他の例については、[セクション13.2.10.3「UNION 句」](#)を参照してください。

- 結合内。詳細および例については、[セクション13.2.10.2「JOIN 句」](#)を参照してください。
- INSERT ステートメントまたは REPLACE ステートメントでの VALUES() のかわりに、そのセマンティクスはここで説明するものと多少異なります。詳細は、[セクション13.2.6「INSERT ステートメント」](#)を参照してください。
- CREATE TABLE ... SELECT および CREATE VIEW ... SELECT のソーステーブルのかわり。詳細および例については、これらのステートメントの説明を参照してください。

13.2.15 WITH (共通テーブル式)

共通テーブル式 (CTE) は、単一ステートメントの範囲内に存在し、あとでそのステートメント内で複数回参照できる名前付き一時結果セットです。次の説明では、CTE を使用するステートメントを記述する方法について説明します。

- [共通テーブル式](#)
- [再帰的な共通テーブル式](#)
- [共通テーブル式の再帰の制限](#)
- [再帰的な共通テーブル式の例](#)
- [類似の構成と比較した共通テーブル式](#)

CTE 最適化の詳細は、[セクション8.2.2.4「マージまたは実体化を使用した導出テーブル、ビュー参照および共通テーブル式の最適化」](#)を参照してください。

追加のリソース

次の記事には、多くの例を含む、MySQL での CTE の使用に関する追加情報が含まれています:

- [MySQL 8.0 ラボ: \[再帰的\] MySQL \(CTE\) の共通テーブル式](#)
- [MySQL 8.0 ラボ: \[再帰的\] MySQL \(CTE\) の共通テーブル式、パート 2 - シリーズの生成方法](#)
- [MySQL 8.0 ラボ: \[再帰的\] MySQL \(CTE\) の共通テーブル式 \(第 3 部\) - 階層](#)
- [MySQL 8.0.1 : \[再帰的\] MySQL \(CTE\) の共通テーブル式、パート 4 - 深さ優先または幅優先トラバース、推移閉鎖、サイクル回避](#)

共通テーブル式

共通テーブル式を指定するには、カンマ区切りの副次句を持つ WITH 句を使用します。各副次句は、結果セットを生成し、名前をサブクエリーに関連付けるサブクエリーを提供します。次の例では、WITH 句で cte1 および cte2 という CTE を定義し、WITH 句に続く最上位 SELECT で CTE を参照します:

```
WITH
cte1 AS (SELECT a, b FROM table1),
cte2 AS (SELECT c, d FROM table2)
SELECT b, d FROM cte1 JOIN cte2
WHERE cte1.a = cte2.c;
```

WITH 句を含むステートメントでは、各 CTE 名を参照して、対応する CTE 結果セットにアクセスできます。

CTE 名は他の CTE で参照できるため、CTE を他の CTE に基づいて定義できます。

CTE は、それ自体を参照して再帰 CTE を定義できます。再帰 CTE の一般的なアプリケーションには、階層データまたはツリー構造化データのシリーズ生成およびトラバースが含まれます。

共通テーブル式は、DML ステートメントの構文のオプション部分です。これらは、WITH 句を使用して定義されます:

```
with_clause:
WITH [RECURSIVE]
cte_name [(col_name [, col_name] ...)] AS (subquery)
[, cte_name [(col_name [, col_name] ...)] AS (subquery)] ...
```

`cte_name` は、単一の共通テーブル式に名前を付け、`WITH` 句を含むステートメントでテーブル参照として使用できません。

`AS (subquery)` の `subquery` 部分は「「CTE のサブクエリー」と呼ばれ、CTE 結果セットを生成します。`AS` の後にカッコが必要です。

サブクエリーが独自の名前を参照する場合、共通テーブル式は再帰的です。`WITH` 句の CTE が再帰的な場合は、`RECURSIVE` キーワードを含める必要があります。詳細は、[再帰的な共通テーブル式](#)を参照してください。

特定の CTE のカラム名の決定は、次のように行われます:

- CTE 名の後にカッコで囲まれた名前のある場合、それらの名前はカラム名になります:

```
WITH cte (col1, col2) AS
(
  SELECT 1, 2
  UNION ALL
  SELECT 3, 4
)
SELECT col1, col2 FROM cte;
```

リスト内の名前数は、結果セット内のカラムの数と同じである必要があります。

- それ以外の場合、カラム名は `AS (subquery)` 部分内の最初の `SELECT` の選択リストから取得されます:

```
WITH cte AS
(
  SELECT 1 AS col1, 2 AS col2
  UNION ALL
  SELECT 3, 4
)
SELECT col1, col2 FROM cte;
```

`WITH` 句は、次のコンテキストで使用できます:

- `SELECT`、`UPDATE` および `DELETE` ステートメントの先頭。

```
WITH ... SELECT ...
WITH ... UPDATE ...
WITH ... DELETE ...
```

- サブクエリーの開始時 (導出テーブルサブクエリーを含む):

```
SELECT ... WHERE id IN (WITH ... SELECT ...) ...
SELECT * FROM (WITH ... SELECT ...) AS dt ...
```

- `SELECT` ステートメントを含むステートメントについては、`SELECT` の直前を参照してください:

```
INSERT ... WITH ... SELECT ...
REPLACE ... WITH ... SELECT ...
CREATE TABLE ... WITH ... SELECT ...
CREATE VIEW ... WITH ... SELECT ...
DECLARE CURSOR ... WITH ... SELECT ...
EXPLAIN ... WITH ... SELECT ...
```

同じレベルで許可される `WITH` 句は 1 つのみです。`WITH` の後に同じレベルの `WITH` が続くことは許可されていないため、これは不正です:

```
WITH cte1 AS (...) WITH cte2 AS (...) SELECT ...
```

ステートメントを有効にするには、副次句をカンマで区切る単一の `WITH` 句を使用します:

```
WITH cte1 AS (...), cte2 AS (...) SELECT ...
```

ただし、異なるレベルで発生する場合は、ステートメントに複数の `WITH` 句を含めることができます:

```
WITH cte1 AS (SELECT 1)
SELECT * FROM (WITH cte2 AS (SELECT 2) SELECT * FROM cte2 JOIN cte1) AS dt;
```

WITH 句では、複数の共通テーブル式を定義できますが、各 CTE 名は句に対して一意である必要があります。これは不正です:

```
WITH cte1 AS (...), cte1 AS (...) SELECT ...
```

ステートメントを有効にするには、CTE を一意の名前で定義します:

```
WITH cte1 AS (...), cte2 AS (...) SELECT ...
```

CTE は、自身または他の CTE を参照できます:

- 自己参照 CTE は再帰的です。
- CTE は、同じ **WITH** 句で以前に定義された CTE を参照できますが、後で定義された CTE は参照できません。

この制約は、**cte1** が **cte2** を参照し、**cte2** が **cte1** を参照する相互再帰 CTE を除外します。これらの参照のいずれかは、後で定義する CTE への参照である必要がありますが、これは許可されていません。

- 特定のクエリーブロック内の CTE は、より外側のレベルのクエリーブロックで定義された CTE を参照できますが、より内側のレベルのクエリーブロックで定義された CTE は参照できません。

同じ名前のオブジェクトへの参照を解決するには、導出テーブルで CTE を非表示にし、CTE で実テーブル、**TEMPORARY** テーブルおよびビューを非表示にします。名前解決は、同じクエリーブロック内のオブジェクトを検索し、その名前のオブジェクトが見つからないときに外部ブロックに順に進むことで行われます。

導出テーブルと同様に、CTE には MySQL 8.0.14 より前の外部参照を含めることはできません。これは、SQL 標準の制限ではなく、MySQL 8.0.14 で削除された MySQL 制限です。再帰 CTE に固有の構文上のその他の考慮事項については、[再帰的な共通テーブル式](#) を参照してください。

再帰的な共通テーブル式

再帰的な共通テーブル式は、独自の名前を参照するサブクエリーを持つ式です。例:

```
WITH RECURSIVE cte (n) AS
(
  SELECT 1
  UNION ALL
  SELECT n + 1 FROM cte WHERE n < 5
)
SELECT * FROM cte;
```

このステートメントを実行すると、単純な線形順序を含む単一の列である次の結果が生成されます:

```
+-----+
| n |
+-----+
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
+-----+
```

再帰 CTE の構造は次のとおりです:

- WITH** 句内の CTE がそれ自体を参照する場合、**WITH** 句は **WITH RECURSIVE** で始まる必要があります。(CTE がそれ自体を参照していない場合、**RECURSIVE** は許可されますが必須ではありません。)

再帰 CTE の **RECURSIVE** を忘れた場合、次のエラーが発生する可能性があります:

```
ERROR 1146 (42S02): Table 'cte_name' doesn't exist
```

- 再帰 CTE サブクエリーには、**UNION [ALL]**または **UNION DISTINCT** で区切られた次の 2 つの部分があります:

```
SELECT ... -- return initial row set
```



```
UNION ALL
SELECT ... -- return additional row sets
```

最初の **SELECT** は CTE の最初の行を生成し、CTE 名を参照しません。2 番目の **SELECT** では、**FROM** 句で CTE 名を参照することで追加の行が生成され、繰り返されます。再帰は、この部分で新しい行が生成されない場合に終了します。したがって、再帰 CTE は非再帰的 **SELECT** 部分とそれに続く再帰的 **SELECT** 部分で構成されます。

各 **SELECT** 部分自体を複数の **SELECT** ステートメントの和集合にすることができます。

- CTE 結果カラムの型は、非再帰的 **SELECT** 部分のカラム型からのみ推測され、カラムはすべて NULL 値可能です。型の決定では、再帰的 **SELECT** 部分は無視されます。
- 非再帰的部分と再帰的部分が **UNION DISTINCT** によって分離されている場合、重複する行は削除されます。これは、無限ループを回避するために推移的な閉鎖を実行するクエリーに役立ちます。
- 再帰部分の各反復は、前の反復によって生成された行に対してのみ動作します。再帰部分に複数のクエリーブロックがある場合、各クエリーブロックの反復は未指定の順序でスケジュールされ、各クエリーブロックは、前の反復または前の反復終了以降に他のクエリーブロックによって生成された行に対して動作します。

前述の再帰 CTE サブクエリーには、最初の行セットを生成するために単一行を取得する次の非再帰部分があります：

```
SELECT 1
```

CTE サブクエリーには、次の再帰的部分もあります：

```
SELECT n + 1 FROM cte WHERE n < 5
```

反復のたびに、その **SELECT** は、前の行セットの n の値より大きい新しい値を持つ行を生成します。最初の反復は初期行セット (1) で動作し、 $1+1=2$ を生成します。2 番目の反復は最初の反復行セット (2) で動作し、 $2+1=3$ などを作成します。これは、 n が 5 未満になったときに発生する再帰が終了するまで続きます。

CTE の再帰的部分で非再帰的部分よりもカラムの値が広い場合は、データの切捨てを回避するために、非再帰的部分のカラムを拡張する必要がある場合があります。次のステートメントがあるとします。

```
WITH RECURSIVE cte AS
(
  SELECT 1 AS n, 'abc' AS str
  UNION ALL
  SELECT n + 1, CONCAT(str, str) FROM cte WHERE n < 3
)
SELECT * FROM cte;
```

非厳密 SQL モードでは、このステートメントは次の出力を生成します：

```
+-----+-----+
| n | str |
+-----+-----+
| 1 | abc |
| 2 | abc |
| 3 | abc |
+-----+-----+
```

非再帰的 **SELECT** によってカラム幅が決定されるため、**str** のカラム値はすべて 'abc' です。したがって、再帰的 **SELECT** によって生成されるより広い **str** 値は切り捨てられます。

厳密な SQL モードでは、このステートメントはエラーを生成します：

```
ERROR 1406 (22001): Data too long for column 'str' at row 1
```

この問題に対処して、ステートメントで切捨てやエラーが発生しないようにするには、非再帰的 **SELECT** で **CAST()** を使用して **str** カラムの幅を広げます：

```
WITH RECURSIVE cte AS
(
  SELECT 1 AS n, CAST('abc' AS CHAR(20)) AS str
  UNION ALL
  SELECT n + 1, CONCAT(str, str) FROM cte WHERE n < 3
)
SELECT * FROM cte;
```

これで、ステートメントは切捨てなしで次の結果を生成します:

```
+-----+
| n | str |
+-----+
| 1 | abc |
| 2 | abcabc |
| 3 | abcabcabc |
+-----+
```

コラムには、位置ではなく名前でアクセスします。つまり、次の CTE に示すように、再帰部分のコラムは、別の位置を持つ非再帰部分のコラムにアクセスできます:

```
WITH RECURSIVE cte AS
(
  SELECT 1 AS n, 1 AS p, -1 AS q
  UNION ALL
  SELECT n + 1, q * 2, p * 2 FROM cte WHERE n < 5
)
SELECT * FROM cte;
```

一方の行の p は前の行の q から導出されるため、正の値と負の値は出力の後続の各行の位置を入れ替えます:

```
+-----+
| n | p | q |
+-----+
| 1 | 1 | -1 |
| 2 | -2 | 2 |
| 3 | 4 | -4 |
| 4 | -8 | 8 |
| 5 | 16 | -16 |
+-----+
```

再帰的 CTE サブクエリー内では、いくつかの構文制約が適用されます:

- 再帰的 **SELECT** 部分には、次の構成を含めることはできません:
 - SUM()** などの集計関数
 - ウィンドウ関数
 - GROUP BY**
 - ORDER BY**
 - DISTINCT**

MySQL 8.0.19 より前は、再帰 CTE の再帰的 **SELECT** 部分で **LIMIT** 句を使用することもできませんでした。この制限は MySQL 8.0.19 で解除され、このような場合、オプションの **OFFSET** 句とともに **LIMIT** がサポートされるようになりました。結果セットへの影響は、最も外側の **SELECT** で **LIMIT** を使用する場合と同じですが、再帰的 **SELECT** とともに使用すると、要求された数の行が生成されるとすぐに次の生成が停止するため、より効率的です。

これらの制約は、再帰 CTE の非再帰的 **SELECT** 部分には適用されません。 **DISTINCT** での禁止は、 **UNION** メンバーにのみ適用されます。 **UNION DISTINCT** は許可されます。

- 再帰的 **SELECT** 部分は CTE を参照する必要があるのは、サブクエリーではなく、その **FROM** 句でのみです。CTE 以外のテーブルを参照して CTE と結合できます。このような結合で CTE を使用する場合、CTE を **LEFT JOIN** の右側に配置しないでください。

これらの制約は、MySQL 固有除外である **ORDER BY**、**LIMIT** (MySQL 8.0.18 以前) および **DISTINCT** 以外は、SQL 標準から取得されます。

再帰 CTE の場合、再帰的 **SELECT** 部品の **EXPLAIN** 出力行の **Extra** コラムに **Recursive** が表示されます。

EXPLAIN によって表示される原価見積りは反復当たりの原価を表しますが、これは合計原価と大きく異なる場合があります。オプティマイザは、**WHERE** 句が **false** になる時点では予測できないため、反復数を予測できません。

CTE の実際のコストは、結果セットのサイズの影響を受ける場合もあります。多くの行を生成する CTE では、メモリ内からディスク上の形式に変換するために十分な大きさの内部一時テーブルが必要になる場合があります。パフォーマンスが低下する可能性があります。その場合、許可されるインメモリー一時テーブルサイズを増やすと、パフォーマンスが向上する可能性があります。セクション 8.4.4 「MySQL での内部一時テーブルの使用」を参照してください。

共通テーブル式の再帰の制限

再帰的 CTE では、再帰的 `SELECT` 部分に再帰を終了する条件が含まれていることが重要です。ランナウェイ再帰 CTE から保護する開発手法として、実行時間に制限を設定することで強制的に終了できます：

- `cte_max_recursion_depth` システム変数は、CTE の再帰レベル数に制限を強制します。サーバーは、この変数の値よりも多くのレベルを繰り返す CTE の実行を終了します。
- `max_execution_time` システム変数は、現在のセッション内で実行される `SELECT` ステートメントの実行タイムアウトを強制します。
- `MAX_EXECUTION_TIME` オプティマイザヒントは、表示されている `SELECT` ステートメントに対してクエリーごとの実行タイムアウトを強制します。

再帰 CTE が誤って書き込まれ、再帰実行の終了条件がないとします：

```
WITH RECURSIVE cte (n) AS
(
  SELECT 1
  UNION ALL
  SELECT n + 1 FROM cte
)
SELECT * FROM cte;
```

デフォルトでは、`cte_max_recursion_depth` の値は 1000 で、CTE は 1000 レベルを超えて繰り返されたときに終了します。アプリケーションでは、セッション値を変更して要件に合わせて調整できます：

```
SET SESSION cte_max_recursion_depth = 10; -- permit only shallow recursion
SET SESSION cte_max_recursion_depth = 1000000; -- permit deeper recursion
```

グローバル `cte_max_recursion_depth` 値を設定して、後で開始するすべてのセッションに影響を与えることもできます。

実行が遅く再帰するクエリー、または `cte_max_recursion_depth` 値を非常に高く設定する理由があるコンテキストでは、深い再帰から保護する別の方法は、セッションタイムアウトを設定することです。これを行うには、CTE ステートメントを実行する前に、次のようなステートメントを実行します：

```
SET max_execution_time = 1000; -- impose one second timeout
```

または、CTE ステートメント自体にオプティマイザヒントを含めます：

```
WITH RECURSIVE cte (n) AS
(
  SELECT 1
  UNION ALL
  SELECT n + 1 FROM cte
)
SELECT /*+ SET_VAR(cte_max_recursion_depth = 1M) */ * FROM cte;

WITH RECURSIVE cte (n) AS
(
  SELECT 1
  UNION ALL
  SELECT n + 1 FROM cte
)
SELECT /*+ MAX_EXECUTION_TIME(1000) */ * FROM cte;
```

MySQL 8.0.19 以降では、再帰的クエリー内で `LIMIT` を使用して、最も外側の `SELECT` に返される行の最大数を設定することもできます。次に例を示します：

```
WITH RECURSIVE cte (n) AS
```

```
(
SELECT 1
UNION ALL
SELECT n + 1 FROM cte LIMIT 10000
)
SELECT * FROM cte;
```

これは、時間制限に加えて、または時間制限を設定するかわりに行うことができます。したがって、次の CTE は、10 千行を返した後、または 1 千秒先に実行された後に終了します:

```
WITH RECURSIVE cte (n) AS
(
SELECT 1
UNION ALL
SELECT n + 1 FROM cte LIMIT 10000
)
SELECT /*+ MAX_EXECUTION_TIME(1000) */ * FROM cte;
```

実行時間制限のない再帰的クエリーが無限ループに入った場合は、[KILL QUERY](#) を使用して別のセッションから再帰的クエリーを終了できます。セッション自体では、クエリーの実行に使用されるクライアントプログラムによってクエリーを強制終了する方法が提供される場合があります。たとえば、[mysql](#) では、Control+C と入力すると現在のステートメントが中断されます。

再帰的な共通テーブル式の例

前述のように、再帰的共通テーブル式 (CTE) は、系列の生成および階層データまたはツリー構造化データのトラバースに頻繁に使用されます。このセクションでは、これらの手法の簡単な例をいくつか示します。

- [フィボナッチシリーズ世代](#)
- [日付シリーズ生成](#)
- [階層データトラバース](#)

フィボナッチシリーズ世代

Fibonacci シリーズは、2 つの数値 0 と 1 (または 1 と 1) で始まり、その後の各数値は前の 2 つの数値の合計です。再帰的共通テーブル式では、再帰的 [SELECT](#) によって生成された各行がシリーズの前の 2 つの数値にアクセスできる場合、Fibonacci シリーズを生成できます。次の CTE は、最初の 2 つの番号として 0 と 1 を使用して 10-number シリーズを生成します:

```
WITH RECURSIVE fibonacci (n, fib_n, next_fib_n) AS
(
SELECT 1, 0, 1
UNION ALL
SELECT n + 1, next_fib_n, fib_n + next_fib_n
FROM fibonacci WHERE n < 10
)
SELECT * FROM fibonacci;
```

CTE は次の結果を生成します:

```
+-----+-----+-----+
| n | fib_n | next_fib_n |
+-----+-----+-----+
| 1 | 0 | 1 |
| 2 | 1 | 1 |
| 3 | 1 | 2 |
| 4 | 2 | 3 |
| 5 | 3 | 5 |
| 6 | 5 | 8 |
| 7 | 8 | 13 |
| 8 | 13 | 21 |
| 9 | 21 | 34 |
| 10 | 34 | 55 |
+-----+-----+-----+
```

CTE の機能:

- `n` は、行に `n` 番目の Fibonacci 番号が含まれていることを示す表示カラムです。たとえば、8 番目の Fibonacci 番号は 13 です。
- `fib_n` カラムには、Fibonacci 番号 `n` が表示されます。
- `next_fib_n` カラムには、数値 `n` の後の次の Fibonacci 番号が表示されます。このカラムは次の行に次の系カラム値を提供するため、行の `fib_n` カラムに前の 2 つの系カラム値の合計を生成できます。
- 再帰は、`n` が 10 に達すると終了します。これは任意の選択肢で、出力を小さな行セットに制限します。

前述の出力は CTE の結果全体を示しています。その一部のみを選択するには、トップレベルの `SELECT` に適切な `WHERE` 句を追加します。たとえば、8 番目のフィボナッチ番号を選択するには、次のようにします：

```
mysql> WITH RECURSIVE fibonacci ...
...
SELECT fib_n FROM fibonacci WHERE n = 8;
+-----+
| fib_n |
+-----+
| 13 |
+-----+
```

日付シリーズ生成

共通テーブル式では、一連の連続した日付を生成できます。これは、サマリーデータにテーブルされていない日付を含む、シリーズ内のすべての日付の行を含むサマリーを生成する場合に役立ちます。

売上番号のテーブルに次の行が含まれているとします：

```
mysql> SELECT * FROM sales ORDER BY date, price;
+-----+-----+
| date   | price |
+-----+-----+
| 2017-01-03 | 100.00 |
| 2017-01-03 | 200.00 |
| 2017-01-06 | 50.00 |
| 2017-01-08 | 10.00 |
| 2017-01-08 | 20.00 |
| 2017-01-08 | 150.00 |
| 2017-01-10 | 5.00 |
+-----+-----+
```

このクエリーでは、日当たりの売上が集計されます：

```
mysql> SELECT date, SUM(price) AS sum_price
FROM sales
GROUP BY date
ORDER BY date;
+-----+-----+
| date   | sum_price |
+-----+-----+
| 2017-01-03 | 300.00 |
| 2017-01-06 | 50.00 |
| 2017-01-08 | 180.00 |
| 2017-01-10 | 5.00 |
+-----+-----+
```

ただし、その結果には、テーブルに含まれる日付の範囲でテーブルされない日付の「穴」が含まれます。範囲内のすべての日付を表す結果を再帰 CTE を使用して生成し、その日付セットを生成して、`LEFT JOIN` と販売データを結合できます。

日付範囲シリーズを生成する CTE は次のとおりです：

```
WITH RECURSIVE dates (date) AS
(
  SELECT MIN(date) FROM sales
  UNION ALL
  SELECT date + INTERVAL 1 DAY FROM dates
  WHERE date + INTERVAL 1 DAY <= (SELECT MAX(date) FROM sales)
)
SELECT * FROM dates;
```

CTE は次の結果を生成します:

```
+-----+
| date |
+-----+
| 2017-01-03 |
| 2017-01-04 |
| 2017-01-05 |
| 2017-01-06 |
| 2017-01-07 |
| 2017-01-08 |
| 2017-01-09 |
| 2017-01-10 |
+-----+
```

CTE の機能:

- 非再帰的 **SELECT** では、**sales** テーブルにまたがる日付範囲内で最も低い日付が生成されます。
- 再帰的 **SELECT** によって生成された各行では、前の行によって生成された日付に 1 日が追加されます。
- 再帰は、日付が **sales** テーブルの範囲内の最も高い日付に達した後に終了します。

CTE を **sales** テーブルに対して **LEFT JOIN** と結合すると、範囲内の各日付の行を含む売上サマリーが生成されます:

```
WITH RECURSIVE dates (date) AS
(
  SELECT MIN(date) FROM sales
  UNION ALL
  SELECT date + INTERVAL 1 DAY FROM dates
  WHERE date + INTERVAL 1 DAY <= (SELECT MAX(date) FROM sales)
)
SELECT dates.date, COALESCE(SUM(price), 0) AS sum_price
FROM dates LEFT JOIN sales ON dates.date = sales.date
GROUP BY dates.date
ORDER BY dates.date;
```

出力は次のようになります:

```
+-----+-----+
| date | sum_price |
+-----+-----+
| 2017-01-03 | 300.00 |
| 2017-01-04 | 0.00 |
| 2017-01-05 | 0.00 |
| 2017-01-06 | 50.00 |
| 2017-01-07 | 0.00 |
| 2017-01-08 | 180.00 |
| 2017-01-09 | 0.00 |
| 2017-01-10 | 5.00 |
+-----+-----+
```

次の点に注意してください:

- クエリーは非効率的ですか。特に、再帰的 **SELECT** の各行に対して **MAX()** サブクエリーが実行されているクエリーですか。 **EXPLAIN** では、**MAX()** を含むサブクエリーが一度のみ評価され、結果がキャッシュされることが示されます。
- **COALESCE()** を使用すると、**sales** テーブルに売上データが発生しない日に、**sum_price** カラムに **NULL** が表示されなくなります。

階層データトラバース

再帰的な共通テーブル式は、階層を形成するデータを横断する場合に役立ちます。会社の各従業員について、従業員名、ID 番号および従業員マネージャの ID を表示する小さなデータセットを作成する次のステートメントについて考えてみます。最上位レベルの従業員 (CEO) のマネージャ ID は **NULL** (マネージャなし) です。

```
CREATE TABLE employees (
  id INT PRIMARY KEY NOT NULL,
  name VARCHAR(100) NOT NULL,
```



```

manager_id INT NULL,
INDEX (manager_id),
FOREIGN KEY (manager_id) REFERENCES employees (id)
);
INSERT INTO employees VALUES
(333, "Yasmina", NULL), # Yasmina is the CEO (manager_id is NULL)
(198, "John", 333), # John has ID 198 and reports to 333 (Yasmina)
(692, "Tarek", 333),
(29, "Pedro", 198),
(4610, "Sarah", 29),
(72, "Pierre", 29),
(123, "Adil", 692);

```

結果のデータセットは次のようになります:

```

mysql> SELECT * FROM employees ORDER BY id;
+----+-----+-----+
| id | name  | manager_id |
+----+-----+-----+
| 29 | Pedro | 198        |
| 72 | Pierre| 29         |
| 123| Adil  | 692        |
| 198| John  | 333        |
| 333| Yasmina| NULL       |
| 692| Tarek | 333        |
| 4610| Sarah | 29         |
+----+-----+-----+

```

各従業員の管理チェーン (つまり、CEO から従業員へのパス) を含む組織図を作成するには、再帰 CTE を使用します:

```

WITH RECURSIVE employee_paths (id, name, path) AS
(
  SELECT id, name, CAST(id AS CHAR(200))
  FROM employees
  WHERE manager_id IS NULL
  UNION ALL
  SELECT e.id, e.name, CONCAT(ep.path, ',', e.id)
  FROM employee_paths AS ep JOIN employees AS e
  ON ep.id = e.manager_id
)
SELECT * FROM employee_paths ORDER BY path;

```

CTE は次の出力を生成します:

```

+----+-----+-----+
| id | name  | path          |
+----+-----+-----+
| 333| Yasmina| 333          |
| 198| John  | 333,198      |
| 29 | Pedro | 333,198,29   |
| 4610| Sarah | 333,198,29,4610 |
| 72 | Pierre| 333,198,29,72 |
| 692| Tarek | 333,692      |
| 123| Adil  | 333,692,123  |
+----+-----+-----+

```

CTE の機能:

- 非再帰的 **SELECT** は、CEO (**NULL** マネージャ ID を持つ行) の行を生成します。

path カラムは、再帰的 **SELECT** によって生成されるより長い **path** 値のための領域が確保されるように、**CHAR(200)** に広がります。

- 再帰的 **SELECT** によって生成された各行では、前の行によって生成された従業員に直接レポートするすべての従業員が検索されます。このような従業員ごとに、行には従業員 ID と従業員名、および従業員管理チェーンが含まれます。チェーンは、従業員 ID が最後に追加されたマネージャチェーンです。
- 再帰は、従業員に他の部下がない場合に終了します。

特定の従業員のパスを検索するには、最上位の **SELECT** に **WHERE** 句を追加します。たとえば、Tarek および Sarah の結果を表示するには、次のように **SELECT** を変更します:

```
mysql> WITH RECURSIVE ...
...
SELECT * FROM employees_extended
WHERE id IN (692, 4610)
ORDER BY path;
+-----+-----+-----+
| id | name | path |
+-----+-----+-----+
| 4610 | Sarah | 333,198,29,4610 |
| 692 | Tarek | 333,692 |
+-----+-----+-----+
```

類似の構成と比較した共通テーブル式

共通テーブル式 (CTE) は、いくつかの点で派生テーブルに似ています:

- 両方の構成メンバーに名前が付けられます。
- 両方の構成は、単一のステートメントの有効範囲に存在します。

これらの類似性のため、CTE と導出テーブルは同じ意味で使用できます。簡単な例として、次のステートメントは同等です:

```
WITH cte AS (SELECT 1) SELECT * FROM cte;
SELECT * FROM (SELECT 1) AS dt;
```

ただし、CTE には導出テーブルと比較していくつかの利点があります:

- 導出テーブルは、クエリー内で一度のみ参照できます。CTE は複数回参照できます。導出テーブルの結果の複数のインスタンスを使用するには、結果を複数回導出する必要があります。
- CTE は自己参照 (再帰的) にすることができます。
- CTE は別の CTE を参照できます。
- CTE 内に埋め込まれるのではなく、CTE の定義がステートメントの先頭に表示されると、CTE が読みやすくなる場合があります。

CTE は、[CREATE \[TEMPORARY\] TABLE](#) で作成されるテーブルに似ていますが、明示的に定義または削除する必要はありません。CTE の場合、テーブルを作成する権限は必要ありません。

13.3 トランザクションステートメントおよびロックステートメント

MySQL は、[SET autocommit](#)、[START TRANSACTION](#)、[COMMIT](#)、[ROLLBACK](#) などのステートメントを介して (特定のクライアントセッション内の) ローカルトランザクションをサポートしています。 [セクション13.3.1「START TRANSACTION、COMMIT および ROLLBACK ステートメント」](#) を参照してください。XA トランザクションサポートにより、MySQL は分散トランザクションにも参加できます。 [セクション13.3.8「XA トランザクション」](#) を参照してください。

13.3.1 START TRANSACTION、COMMIT および ROLLBACK ステートメント

```
START TRANSACTION
[transaction_characteristic [, transaction_characteristic] ...]

transaction_characteristic: {
  WITH CONSISTENT SNAPSHOT
  | READ WRITE
  | READ ONLY
}

BEGIN [WORK]
COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
SET autocommit = {0 | 1}
```

次のステートメントにより、[トランザクション](#)の使用を制御できます。

- **START TRANSACTION** または **BEGIN** は、新しいトランザクションを開始します。
- **COMMIT** は、現在のトランザクションをコミットして、その変更を永続的なものにします。
- **ROLLBACK** は、現在のトランザクションをロールバックして、その変更を取り消します。
- **SET autocommit** は、現在のセッションのデフォルトの自動コミットモードを無効または有効にします。

デフォルトでは、MySQL は **自動コミット** モードが有効になった状態で動作します。つまり、特にトランザクション内がない場合、各ステートメントは **START TRANSACTION** および **COMMIT** で囲まれているかのようにアトミックです。 **ROLLBACK** を使用して効果を元に戻すことはできませんが、ステートメントの実行中にエラーが発生した場合、ステートメントはロールバックされます。

一連のステートメントに対して自動コミットモードを暗黙的に無効にするには、 **START TRANSACTION** ステートメントを使用します。

```
START TRANSACTION;  
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;  
UPDATE table2 SET summary=@A WHERE type=1;  
COMMIT;
```

START TRANSACTION を使用すると、そのトランザクションを **COMMIT** または **ROLLBACK** で終了するまで、自動コミットは無効のままになります。そのあと、自動コミットモードはその以前の状態に戻ります。

START TRANSACTION では、トランザクションの特性を制御するいくつかの修飾子が許可されます。複数の修飾子を指定するには、それらをカンマで区切ります。

- **WITH CONSISTENT SNAPSHOT** 修飾子は、この機能に対応しているストレージエンジンでの **一貫性読み取り** を開始します。これは、InnoDB にのみ適用されます。その効果は、任意の InnoDB テーブルから **START TRANSACTION** に続けて **SELECT** を発行することと同じです。 [セクション15.7.2.3「一貫性非ロック読み取り」](#) を参照してください。 **WITH CONSISTENT SNAPSHOT** 修飾子は、現在のトランザクション **分離レベル** を変更しないため、現在の分離レベルが一貫性読み取りを許可するものである場合にのみ、整合性のあるスナップショットを提供します。一貫性読み取りを許可する分離レベルは、**REPEATABLE READ** だけです。その他のすべての分離レベルの場合、**WITH CONSISTENT SNAPSHOT** 句は無視されます。 **WITH CONSISTENT SNAPSHOT** 句が無視されると、警告が生成されます。
- **READ WRITE** および **READ ONLY** 修飾子は、トランザクションアクセスモードを設定します。これらは、そのトランザクションで使用されるテーブルへの変更を許可または禁止します。 **READ ONLY** の制限は、そのトランザクションが、ほかのトランザクションに表示されるトランザクションテーブルと非トランザクションテーブルの両方を変更またはロックしないようにします。このトランザクションは引き続き、一時テーブルを変更またはロックできます。

MySQL では、トランザクションが読み取り専用であることがわかっている場合、InnoDB テーブルに対するクエリーの追加の最適化が可能です。 **READ ONLY** を指定すると、読み取り専用ステータスを自動的に特定できない場合に、これらの最適化が適用されることが保証されます。詳細は、 [セクション8.5.3「InnoDB の読み取り専用トランザクションの最適化」](#) を参照してください。

アクセスモードが指定されていない場合は、デフォルトモードが適用されます。デフォルトが変更されていない限り、それは読み取り/書き込みです。同じステートメント内で **READ WRITE** と **READ ONLY** の両方を指定することは許可されません。

読み取り専用モードでは、DML ステートメントを使用して **TEMPORARY** キーワードで作成されたテーブルは引き続き変更できます。永続的なテーブルと同様に、DDL ステートメントによって行われる変更は許可されません。

トランザクションアクセスモードの詳細 (デフォルトモードを変更する方法を含む) は、 [セクション13.3.7「SET TRANSACTION ステートメント」](#) を参照してください。

read_only システム変数が有効になっている場合、 **START TRANSACTION READ WRITE** でトランザクションを明示的に開始するには、 **CONNECTION_ADMIN** 権限 (または非推奨の **SUPER** 権限) が必要です。

重要

MySQL クライアントアプリケーションを記述するために使用される多くの API (JDBC など) は、クライアントから **START TRANSACTION** ステートメントを送信する代わりに使用

できる (また、場合によっては使用すべき)、トランザクションを開始するための独自のメソッドを提供しています。詳細は、[第29章「Connector および API」](#) または API のドキュメントを参照してください。

自動コミットモードを明示的に無効にするには、次のステートメントを使用します。

```
SET autocommit=0;
```

`autocommit` 変数を 0 に設定することによって自動コミットモードを無効にしたあと、トランザクションセーフテーブル (InnoDB または NDB のテーブルなど) への変更がただちに永続的になることはありません。COMMIT を使用して変更をディスクに格納するか、または ROLLBACK を使用して変更を無視する必要があります。

`autocommit` はセッション変数であるため、セッションごとに設定する必要があります。新しい接続ごとに自動コミットモードを無効にするには、[セクション5.1.8「サーバーシステム変数」](#)にある `autocommit` システム変数の説明を参照してください。

BEGIN と BEGIN WORK は、トランザクションを開始するための START TRANSACTION のエイリアスとしてサポートされています。標準の SQL 構文である START TRANSACTION は、アドホックトランザクションを開始するための推奨される方法であり、BEGIN では許可されない修飾子が許可されます。

BEGIN ステートメントは、BEGIN ... END 複合ステートメントを開始する BEGIN キーワードの使用とは異なります。後者はトランザクションを開始しません。[セクション13.6.1「BEGIN ... END 複合ステートメント」](#)を参照してください。

注記

すべてのストアードプログラム (ストアードプロシージャとストアードファンクション、トリガー、およびイベント) 内で、パーサーは、BEGIN [WORK] を BEGIN ... END ブロックの開始として扱います。このコンテキストでは、代わりに START TRANSACTION を使用してトランザクションを開始します。

オプションの WORK キーワードは、CHAIN および RELEASE 句と同様に、COMMIT と ROLLBACK に対してサポートされています。CHAIN と RELEASE は、トランザクションの完了に対する追加の制御に使用できます。`completion_type` システム変数の値によって、デフォルトの完了動作が決定されます。[セクション5.1.8「サーバーシステム変数」](#)を参照してください。

AND CHAIN 句を指定すると、現在のトランザクションが終了するとすぐに新しいトランザクションが開始され、新しいトランザクションの分離レベルは終了したばかりのトランザクションと同じになります。新しいトランザクションでは、終了理由トランザクションと同じアクセスモード (READ WRITE または READ ONLY) も使用されます。RELEASE 句を指定すると、サーバーは、現在のトランザクションを終了したあと現在のクライアントセッションを切り離します。NO キーワードを含めると、CHAIN または RELEASE の完了が抑制されます。これは、`completion_type` システム変数がデフォルトで、チェーンまたはリリースの完了が実行されるように設定されている場合に役立つことがあります。

トランザクションを開始すると、保留中のトランザクションはすべてコミットされます。詳細は、[セクション13.3.3「暗黙的なコミットを発生させるステートメント」](#)を参照してください。

また、トランザクションを開始すると、ユーザーが UNLOCK TABLES を実行したかのように、LOCK TABLES によって取得されたテーブルロックも解放されます。トランザクションを開始しても、FLUSH TABLES WITH READ LOCK によって取得されたグローバルな読み取りロックは解放されません。

最適な結果を得るために、トランザクションは、1つのトランザクションセーフストレージエンジンによって管理されているテーブルのみを使用して実行するようにしてください。そうしないと、次の問題が発生する場合があります。

- 複数のトランザクションセーフストレージエンジン (InnoDB など) のテーブルを使用し、トランザクション分離レベルが SERIALIZABLE でない場合、あるトランザクションがコミットされると、同じテーブルを使用する別の進行中のトランザクションには、最初のトランザクションによって行われた変更の一部のみが表示される可能性があります。つまり、混在したエンジンではトランザクションのアトミック性が保証されないため、不整合が発生する場合があります。(混在したエンジンでのトランザクションの頻度が低い場合は、SET TRANSACTION ISOLATION LEVEL を使用して、必要に応じてトランザクションごとに分離レベルを SERIALIZABLE に設定できます。)

- トランザクション内でトランザクションセーフでないテーブルを使用する場合は、自動コミットモードのステータスには関係なく、それらのテーブルへの変更が一度に格納されます。
- トランザクション内で非トランザクションテーブルを更新したあとに **ROLLBACK** ステートメントを発行すると、**ER_WARNING_NOT_COMPLETE_ROLLBACK** 警告が発生します。トランザクションセーフテーブルへの変更はロールバックされますが、非トランザクションセーフテーブルへの変更はロールバックされません。

各トランザクションは、**COMMIT** 時に、1つのまとまりでバイナリログに格納されます。ロールバックされたトランザクションはログに記録されません。(例外: 非トランザクションテーブルへの変更はロールバックできません。ロールバックされるトランザクションに非トランザクションテーブルへの変更が含まれている場合は、非トランザクションテーブルへの変更が確実にレプリケートされるようにするために、最後に **ROLLBACK** ステートメントを使用してトランザクション全体がログに記録されます。) [セクション5.4.4「バイナリログ」](#)を参照してください。

トランザクションの分離レベルまたはアクセスモードは、**SET TRANSACTION** ステートメントを使用して変更できません。 [セクション13.3.7「SET TRANSACTION ステートメント」](#)を参照してください。

ロールバックは、ユーザーが明示的に求めることなく(たとえば、エラーの発生時に) 暗黙的に発生する可能性のある低速な操作になる場合があります。このため、**ROLLBACK** ステートメントを使用して実行された明示的なロールバックに対してだけでなく、暗黙のロールバックに対しても、**SHOW PROCESSLIST** はセッションの **State** カラムに **Rolling back** を表示します。

注記

MySQL 8.0 では、**BEGIN**、**COMMIT**、および **ROLLBACK** は `--replicate-do-db` または `--replicate-ignore-db` ルールによって影響を受けません。

InnoDB でトランザクションの完全なロールバックが実行されると、トランザクションで設定されたすべてのロックが解放されます。重複キーエラーなどのエラーの結果としてトランザクション内の単一の SQL ステートメントがロールバックされた場合、そのステートメントによって設定されたロックは、トランザクションがアクティブなまま保持されます。これが発生する原因は、**InnoDB** では、どの行がどのステートメントで設定されたのかをあとで確認できないような形式で、行ロックが格納されるためです。

トランザクション内の **SELECT** ステートメントがストアドファンクションをコールし、ストアドファンクション内のステートメントが失敗した場合、そのステートメントはロールバックされます。その後、**ROLLBACK** がトランザクションに対して実行されると、トランザクション全体がロールバックされます。

13.3.2 ロールバックできないステートメント

いくつかのステートメントはロールバックできません。これには一般に、データベースを作成または削除したり、テーブルやストアドルーチンを作成、削除、または変更したりするデータ定義言語 (DDL) ステートメントが含まれます。

このようなステートメントを含まないようにトランザクションを設計してください。ロールバックできないステートメントをトランザクション内で早期に発行し、そのあと別のステートメントが失敗したとすると、このような場合に **ROLLBACK** ステートメントを発行してもそのトランザクションのすべての効果をロールバックすることはできません。

13.3.3 暗黙的なコミットを発生させるステートメント

このセクションに示されているステートメント (およびそのすべてのシノニム) は、ユーザーがこのステートメントを実行する前に **COMMIT** を実行したかのように、現在のセッション内でアクティブなすべてのトランザクションを暗黙的に終了します。

これらのステートメントのほとんどは、実行後に暗黙的なコミットを引き起こします。その目的は、そのような各ステートメントを独自の特別なトランザクションで処理することです。トランザクション制御ステートメントおよびロックステートメントは例外です: 実行前に暗黙的なコミットが発生した場合、後に別のコミットは発生しません。

- データベースオブジェクトを定義または変更するデータ定義言語 (DDL) ステートメント。 **ALTER EVENT**, **ALTER FUNCTION**, **ALTER PROCEDURE**, **ALTER SERVER**, **ALTER TABLE**, **ALTER VIEW**, **CREATE DATABASE**, **CREATE EVENT**, **CREATE FUNCTION**, **CREATE INDEX**, **CREATE PROCEDURE**, **CREATE ROLE**, **CREATE SERVER**, **CREATE SPATIAL REFERENCE SYSTEM**, **CREATE TABLE**, **CREATE TRIGGER**, **CREATE VIEW**, **DROP DATABASE**, **DROP EVENT**, **DROP FUNCTION**, **DROP INDEX**, **DROP PROCEDURE**, **DROP ROLE**, **DROP**

SERVER, DROP SPATIAL REFERENCE SYSTEM, DROP TABLE, DROP TRIGGER, DROP VIEW, INSTALL PLUGIN, RENAME TABLE, TRUNCATE TABLE, UNINSTALL PLUGIN。

CREATE TABLE および DROP TABLE ステートメントは、TEMPORARY キーワードが使用されている場合はトランザクションをコミットしません。(これは、コミットを発生させる ALTER TABLE や CREATE INDEX などの、一時テーブルに対するその他の操作には適用されません。)ただし、暗黙的なコミットは発生しませんが、ステートメントのロールバックもできません。つまり、このようなステートメントを使用すると、トランザクションのアトミック性が侵害されます。たとえば、CREATE TEMPORARY TABLE を使用したあとにトランザクションをロールバックしても、そのテーブルは存在し続けます。

InnoDB での CREATE TABLE ステートメントは、1つのトランザクションとして処理されます。つまり、ユーザーが ROLLBACK を発行しても、ユーザーがそのトランザクション中に実行した CREATE TABLE ステートメントは元に戻されません。

CREATE TABLE ... SELECT は、一時テーブル以外のテーブルを作成している場合、そのステートメントが実行される前後に暗黙的なコミットを発生させます。(CREATE TEMPORARY TABLE ... SELECT に対してコミットは発生しません。)

- mysql データベース内のテーブルを暗黙的に使用または変更するステートメント。ALTER USER, CREATE USER, DROP USER, GRANT, RENAME USER, REVOKE, SET PASSWORD。
- トランザクション制御およびロックステートメント。BEGIN, LOCK TABLES, SET autocommit = 1 (この値がまだ 1 でない場合)、START TRANSACTION, UNLOCK TABLES。

UNLOCK TABLES は、非トランザクションテーブルロックを取得するために現在 LOCK TABLES でロックされているテーブルがある場合のみ、トランザクションをコミットします。FLUSH TABLES WITH READ LOCK はテーブルレベルのロックを取得しないため、このステートメントに続く UNLOCK TABLES に対してコミットは発生しません。

トランザクションをネストすることはできません。これは、START TRANSACTION ステートメントまたはそのシノニムのいずれかを発行するときに、現在のすべてのトランザクションに対して実行される暗黙的なコミットの結果です。

XA トランザクションが ACTIVE 状態にある間に、暗黙的なコミットを発生させるステートメントをそのトランザクションで使用することはできません。

BEGIN ステートメントは、BEGIN ... END 複合ステートメントを開始する BEGIN キーワードの使用とは異なります。後者は暗黙的なコミットを発生させません。セクション13.6.1「BEGIN ... END 複合ステートメント」を参照してください。

- データロードステートメント。LOAD DATA。LOAD DATA では、NDB ストレージエンジンを使用するテーブルに対してのみ暗黙的なコミットが発生します。
- 管理ステートメント。ANALYZE TABLE, CACHE INDEX, CHECK TABLE, FLUSH, LOAD INDEX INTO CACHE, OPTIMIZE TABLE, REPAIR TABLE, RESET (ただし、RESET PERSIST)。
- レプリケーション制御ステートメント。START REPLICA | SLAVE, STOP REPLICA | SLAVE, RESET REPLICA | SLAVE, CHANGE REPLICATION SOURCE TO, CHANGE MASTER TO。

13.3.4 SAVEPOINT、ROLLBACK TO SAVEPOINT および RELEASE SAVEPOINT ステートメント

```
SAVEPOINT identifier
ROLLBACK [WORK] TO [SAVEPOINT] identifier
RELEASE SAVEPOINT identifier
```

InnoDB は、SQL ステートメント SAVEPOINT、ROLLBACK TO SAVEPOINT、RELEASE SAVEPOINT のほか、ROLLBACK のオプションの WORK キーワードをサポートしています。

SAVEPOINT ステートメントは、identifier の名前を持つ名前付きのトランザクションセーブポイントを設定します。現在のトランザクションに同じ名前を持つセーブポイントが含まれている場合、古いセーブポイントは削除され、新しいセーブポイントが設定されます。

ROLLBACK TO SAVEPOINT ステートメントは、トランザクションを終了することなく、そのトランザクションを指定されたセーブポイントにロールバックします。セーブポイントが設定されたあとに現在のトランザクションが行に対して行った変更はロールバックで元に戻されますが、InnoDB は、セーブポイントのあとにメモリーに格納された行ロックを解放しません。(新しく挿入された行の場合、ロック情報は、その行に格納されているトランザクション ID によって伝達されます。ロックが個別にメモリーに格納されるわけではありません。この場合、行ロックは Undo で解放されます。) 指定されたセーブポイントよりあとで設定されたセーブポイントは削除されます。

ROLLBACK TO SAVEPOINT ステートメントが次のエラーを返した場合は、指定された名前を持つセーブポイントが存在しないことを示しています。

```
ERROR 1305 (42000): SAVEPOINT identifier does not exist
```

RELEASE SAVEPOINT ステートメントは、指定されたセーブポイントを現在のトランザクションの一連のセーブポイントから削除します。コミットまたはロールバックは発生しません。そのセーブポイントが存在しない場合はエラーになります。

COMMIT、またはセーブポイントを指定しない **ROLLBACK** を実行した場合は、現在のトランザクションのすべてのセーブポイントが削除されます。

ストアドファンクションが呼び出されるか、またはトリガーがアクティブ化されると、新しいセーブポイントレベルが作成されます。以前のレベルにあるセーブポイントは使用できなくなるため、新しいレベルのセーブポイントとは競合しません。関数またはトリガーが終了すると、その関数またはトリガーによって作成されたセーブポイントはすべて解放され、以前のセーブポイントレベルがリストアされます。

13.3.5 LOCK INSTANCE FOR BACKUP および UNLOCK INSTANCE ステートメント

```
LOCK INSTANCE FOR BACKUP
```

```
UNLOCK INSTANCE
```

LOCK INSTANCE FOR BACKUP は、オンラインバックアップ中に DML を許可するインスタンスレベルのバックアップロックを取得し、一貫性のないスナップショットを生成する可能性のある操作を防止します。

LOCK INSTANCE FOR BACKUP ステートメントを実行するには、**BACKUP_ADMIN** 権限が必要です。以前のバージョンから MySQL 8.0 へのインプレースアップグレードを実行すると、**RELOAD** 権限を持つユーザーに **BACKUP_ADMIN** 権限が自動的に付与されます。

複数のセッションで同時にバックアップロックを保持できます。

UNLOCK INSTANCE は、現在のセッションで保持されているバックアップロックを解放します。セッションが終了すると、セッションによって保持されているバックアップロックも解放されます。

LOCK INSTANCE FOR BACKUP では、ファイルの作成、名前変更または削除はできません。**REPAIR TABLE**、**TRUNCATE TABLE**、**OPTIMIZE TABLE** およびアカウント管理ステートメントはブロックされます。[セクション 13.7.1「アカウント管理ステートメント」](#)を参照してください。InnoDB redo ログに記録されていない InnoDB ファイルを変更する操作もブロックされます。

LOCK INSTANCE FOR BACKUP では、ユーザーが作成した一時テーブルにのみ影響する DDL 操作が許可されます。実際には、バックアップロックが保持されている間に、ユーザーが作成した一時テーブルに属するファイルを作成、名前変更または削除できます。バイナリログファイルの作成も許可されます。

LOCK INSTANCE FOR BACKUP によって取得されるバックアップロックは、**FLUSH TABLES *tbl_name* [, *tbl_name*] ... WITH READ LOCK** によって取得されるトランザクションロックおよびロックとは無関係であり、次の一連のステートメントが許可されます:

```
LOCK INSTANCE FOR BACKUP;
FLUSH TABLES tbl_name [, tbl_name] ... WITH READ LOCK;
UNLOCK TABLES;
UNLOCK INSTANCE;
```

```
FLUSH TABLES tbl_name [, tbl_name] ... WITH READ LOCK;
```

```
LOCK INSTANCE FOR BACKUP;
UNLOCK INSTANCE;
UNLOCK TABLES;
```

`lock_wait_timeout` 設定では、`LOCK INSTANCE FOR BACKUP` ステートメントがロックの取得を待機してから放棄する時間を定義します。

13.3.6 LOCK TABLES および UNLOCK TABLES ステートメント

```
LOCK TABLES
tbl_name [[AS] alias] lock_type
[, tbl_name [[AS] alias] lock_type] ...
```

```
lock_type: {
  READ [LOCAL]
  | [LOW_PRIORITY] WRITE
}
```

```
UNLOCK TABLES
```

MySQL では、クライアントセッションは、ほかのセッションと連携してテーブルにアクセスするために、またはそのセッションにテーブルへの排他的アクセスが必要な期間中はほかのセッションによってそのテーブルが変更されないようにするために、明示的にテーブルロックを取得できます。セッションがロックを取得または解放できるのは、それ自体のためだけです。あるセッションが別のセッションのためにロックを取得したり、別のセッションによって保持されているロックを解放したりすることはできません。

ロックを使用すると、トランザクションをエミュレートするか、またはテーブル更新時の速度を向上させることができます。詳細は、[テーブルロックの制限と条件](#) を参照してください。

`LOCK TABLES` は、現在のクライアントセッションのテーブルロックを明示的に取得します。テーブルロックは、ベーステーブルまたはビューに対して取得できます。ロックされる各オブジェクトに対する `LOCK TABLES` 権限と `SELECT` 権限が必要です。

ビューのロックの場合、`LOCK TABLES` は、そのビューで使用されているすべてのベーステーブルをロックされるテーブルのセットに追加し、それらのテーブルを自動的にロックします。ロックされるビューの基礎となるテーブルの場合、`LOCK TABLES` は、ビュー定義者 (`SQL SECURITY DEFINER` ビューの場合) または実行者 (すべてのビューの場合) がテーブルに対する適切な権限を持っていることを確認します。

`LOCK TABLES` とトリガーで説明されているように、`LOCK TABLES` によって明示的にテーブルをロックした場合は、トリガーで使用されているテーブルもすべて暗黙的にロックされます。

`LOCK TABLES` を使用してテーブルを明示的にロックすると、外部キー制約に関連するテーブルが暗黙的にオープンおよびロックされます。外部キーチェックでは、関連するテーブルに対して共有読取り専用ロック (`LOCK TABLES READ`) が取得されます。カスケード更新では、操作に関連する関連テーブルに対してシェアードナッシング書き込みロック (`LOCK TABLES WRITE`) が取得されます。

`UNLOCK TABLES` は、現在のセッションによって保持されているテーブルロックをすべて明示的に解放します。`LOCK TABLES` は、新しいロックを取得する前に、現在のセッションによって保持されているテーブルロックをすべて暗黙的に解放します。

`UNLOCK TABLES` の別の使用方法として、すべてのデータベース内のすべてのテーブルをロックできる `FLUSH TABLES WITH READ LOCK` ステートメントによって取得されたグローバルな読み取り専用ロックの解放があります。[セクション 13.7.8.3 「FLUSH ステートメント」](#) を参照してください。(これは、特定時点のスナップショットを取得できる、Veritas などのファイルシステムがある場合にバックアップを取得するための非常に便利な方法です。)

テーブルロックは、ほかのセッションによる不適切な読み取りまたは書き込みからのみ保護します。`WRITE` ロックを保持しているセッションは、`DROP TABLE` や `TRUNCATE TABLE` などのテーブルレベルの操作を実行できます。`READ` ロックを保持しているセッションの場合、`DROP TABLE` および `TRUNCATE TABLE` 操作は許可されません。

次の説明は、`TEMPORARY` 以外のテーブルにのみ適用されます。`LOCK TABLES` は `TEMPORARY` テーブルに対して許可されます (ただし、無視されます)。テーブルは、ほかのどのようなロックが有効になっているかには関係なく、そのテーブルが作成されたセッションから自由にアクセスできます。ほかのどのセッションもそのテーブルを参照できないため、ロックは必要ありません。

- [テーブルロック取得](#)
- [テーブルロック解除](#)
- [テーブルロックとトランザクションの通信](#)
- [LOCK TABLES とトリガー](#)
- [テーブルロックの制限と条件](#)

テーブルロック取得

現在のセッション内でテーブルロックを取得するには、メタデータロックを取得する `LOCK TABLES` ステートメントを使用します ([セクション8.11.4「メタデータのロック」](#) を参照)。

次のロックタイプを使用できます。

`READ [LOCAL]` ロック:

- このロックを保持しているセッションは、テーブルを読み取ることができます (ただし、書き込みはできません)。
- 複数のセッションが同時にテーブルに対する `READ` ロックを取得できます。
- ほかのセッションは、`READ` ロックを明示的に取得することなく、テーブルを読み取ることができます。
- `LOCAL` 修飾子を使用すると、ロックが保持されている間、ほかのセッションによる競合しない `INSERT` ステートメント (並列挿入) を実行できます。 ([セクション8.11.3「同時挿入」](#) を参照してください。) ただし、ロックを保持している間、サーバーの外部にあるプロセスを使用してデータベースを操作しようとしている場合は、`READ LOCAL` を使用できません。InnoDB テーブルの場合、`READ LOCAL` は `READ` と同じです。

`[LOW_PRIORITY] WRITE` ロック:

- このロックを保持しているセッションは、テーブルの読み取りおよび書き込みが可能です。
- このロックを保持しているセッションだけがテーブルにアクセスできます。ロックが解放されるまで、ほかのどのセッションもアクセスできません。
- `WRITE` ロックが保持されている間、テーブルに対するほかのセッションからのロック要求はブロックされます。
- `LOW_PRIORITY` 修飾子は何の効果もありません。以前のバージョンの MySQL では、ロックの動作に影響を与えましたが、これは当てはまらなくなっています。これは非推奨になり、使用すると警告が生成されます。代わりに、`LOW_PRIORITY` のない `WRITE` を使用してください。

`WRITE` ロックは通常、更新ができるだけ早く処理されるように、`READ` ロックより高い優先度を持っています。つまり、あるセッションが `READ` ロックを取得したあと、別のセッションが `WRITE` ロックを要求した場合は、`WRITE` ロックを要求したセッションがロックを取得して解放するまで、以降の `READ` ロック要求が待たされます。(このポリシーの例外は、`max_write_lock_count` システム変数の小さい値に対して発生する可能性があります。[セクション8.11.4「メタデータのロック」](#) を参照してください。)

`LOCK TABLES` ステートメントが、いずれかのテーブルに対するほかのセッションによって保持されているロックのために待機する必要がある場合、このステートメントはすべてのロックを取得できるまでブロックされます。

ロックが必要なセッションは、必要なすべてのロックを1つの `LOCK TABLES` ステートメントで取得する必要があります。このように取得されたロックが保持されている間、このセッションは、ロックされたテーブルにのみアクセスできます。たとえば、次のステートメントシーケンスでは、`t2` が `LOCK TABLES` ステートメントでロックされていないため、このテーブルにアクセスしようとするとエラーが発生します。

```
mysql> LOCK TABLES t1 READ;
mysql> SELECT COUNT(*) FROM t1;
+-----+
| COUNT(*) |
+-----+
|      3 |
+-----+
mysql> SELECT COUNT(*) FROM t2;
```

```
ERROR 1100 (HY000): Table 't2' was not locked with LOCK TABLES
```

INFORMATION_SCHEMA データベース内のテーブルは例外です。これらのテーブルは、セッションが **LOCK TABLES** によって取得されたテーブルロックを保持している間であっても、明示的にロックされることなくアクセスできます。

ロックされたテーブルを、同じ名前を使用して1つのクエリーで複数回参照することはできません。代わりにエイリアスを使用し、そのテーブルと各エイリアスのための個別のロックを取得します。

```
mysql> LOCK TABLE t WRITE, t AS t1 READ;
mysql> INSERT INTO t SELECT * FROM t;
ERROR 1100: Table 't' was not locked with LOCK TABLES
mysql> INSERT INTO t SELECT * FROM t AS t1;
```

最初の **INSERT** では、ロックされたテーブルに対する同じ名前への参照が2つ存在するため、エラーが発生します。2番目の **INSERT** は、テーブルへの参照で異なる名前が使用されるため、成功します。

ステートメントがエイリアスを使用してテーブルを参照する場合は、その同じエイリアスを使用してテーブルをロックする必要があります。エイリアスを指定しないでテーブルをロックすることはできません。

```
mysql> LOCK TABLE t READ;
mysql> SELECT * FROM t AS myalias;
ERROR 1100: Table 'myalias' was not locked with LOCK TABLES
```

逆に、エイリアスを使用してテーブルをロックする場合は、ステートメント内でそのエイリアスを使用してテーブルを参照する必要があります。

```
mysql> LOCK TABLE t AS myalias READ;
mysql> SELECT * FROM t;
ERROR 1100: Table 't' was not locked with LOCK TABLES
mysql> SELECT * FROM t AS myalias;
```

テーブルロック解除

セッションによって保持されているテーブルロックが解放される場合は、すべてのテーブルロックが一度に解放されます。セッションは明示的にロックを解放できます。また、特定の状況で、ロックが暗黙的に解放される場合もあります。

- セッションは、**UNLOCK TABLES** によって明示的にロックを解放できます。
- セッションがすでにロックを保持している間にロックを取得するために **LOCK TABLES** ステートメントを発行した場合は、新しいロックが付与される前に、その既存のロックが暗黙的に解放されます。
- セッションが (たとえば、**START TRANSACTION** で) トランザクションを開始した場合は、暗黙的な **UNLOCK TABLES** が実行され、既存のロックが解放されます。(テーブルロックとトランザクションの間の通信の詳細は、[テーブルロックとトランザクションの通信](#)を参照してください。)

クライアントセッションの接続が (正常または異常にかかわらず) 終了した場合、サーバーは、そのセッションによって保持されているすべてのテーブルロック (トランザクションおよび非トランザクション) を暗黙的に解放します。クライアントが再接続すると、ロックは無効になります。さらに、クライアントにアクティブなトランザクションがある場合、サーバーは切断時にそのトランザクションをロールバックし、再接続が発生した場合は、自動コミットが有効になった状態で新しいセッションが開始されます。このため、クライアントは自動再接続を無効にすることが必要になる場合があります。自動再接続が有効な場合、再接続が発生してもテーブルロックまたは現在のトランザクションが失われても、クライアントには通知されません。自動再接続が無効になっている場合は、接続が削除されると、発行された次のステートメントに対してエラーが発生します。クライアントはそのエラーを検出し、ロックの再取得やトランザクションの再実行などの適切なアクションを実行できます。[Automatic Reconnection Control](#)を参照してください。

注記

ロックされたテーブル上で **ALTER TABLE** を使用すると、そのテーブルがロック解除される場合があります。たとえば、2番目の **ALTER TABLE** 操作を試みると、エラー「**テーブル 'tbl_name' は LOCK TABLES でロックされていません**」が発生する場合があります。これに対処するには、2番目の変更の前にテーブルを再度ロックします。[セクション B.3.6.1「ALTER TABLE での問題」](#)も参照してください。

テーブルロックとトランザクションの通信

LOCK TABLES および UNLOCK TABLES は、トランザクションの使用との間で次のように通信します。

- LOCK TABLES はトランザクションセーフではないため、テーブルをロックしようとする前に、アクティブなトランザクションをすべて暗黙的にコミットします。
- UNLOCK TABLES は、アクティブなトランザクションをすべて暗黙的にコミットしますが、これが行われるのは、テーブルロックを取得するために LOCK TABLES が使用された場合のみです。たとえば、次の一連のステートメントでは、UNLOCK TABLES がグローバルな読み取りロックを解放しますが、有効なテーブルロックがないためにトランザクションはコミットされません。

```
FLUSH TABLES WITH READ LOCK;
START TRANSACTION;
SELECT ...;
UNLOCK TABLES;
```

- トランザクションを (たとえば、START TRANSACTION で) 開始すると、現在のトランザクションはすべて暗黙的にコミットされ、既存のテーブルロックが解放されます。
- FLUSH TABLES WITH READ LOCK は、グローバルな読み取りロックを取得しますが、テーブルロックは取得しないため、テーブルロックと暗黙的なコミットに関して LOCK TABLES および UNLOCK TABLES と同じ動作には従いません。たとえば、START TRANSACTION は、グローバルな読み取りロックを解放しません。 [セクション 13.7.8.3 「FLUSH ステートメント」](#) を参照してください。
- 暗黙的にトランザクションのコミットを発生させるその他のステートメントは、既存のテーブルロックを解放しません。このようなステートメントのリストについては、 [セクション 13.3.3 「暗黙的なコミットを発生させるステートメント」](#) を参照してください。
- トランザクションテーブル (InnoDB テーブルなど) で LOCK TABLES および UNLOCK TABLES を使用するための正しい方法は、SET autocommit = 0 (START TRANSACTION ではなく) に続けて LOCK TABLES を指定することによってトランザクションを開始し、そのトランザクションを明示的にコミットするまで UNLOCK TABLES を呼び出さないことです。たとえば、テーブル t1 に書き込み、テーブル t2 から読み取る必要がある場合は、次のように実行できます。

```
SET autocommit=0;
LOCK TABLES t1 WRITE, t2 READ, ...;
... do something with tables t1 and t2 here ...
COMMIT;
UNLOCK TABLES;
```

LOCK TABLES を呼び出すと、InnoDB は内部的に独自のテーブルロックを取得し、MySQL は独自のテーブルロックを取得します。InnoDB は次のコミット時に内部のテーブルロックを解放しますが、MySQL でテーブルロックが解放されるようにするには、UNLOCK TABLES を呼び出す必要があります。autocommit = 1 を指定すると、LOCK TABLES の呼び出しの直後に InnoDB によって内部のテーブルロックが解放され、デッドロックが非常に発生しやすくなる場合があるため、この指定は行わないようにしてください。autocommit = 1 が指定された場合、古いアプリケーションが不必要なデッドロックを回避するのに役立つように、InnoDB は内部のテーブルロックをまったく取得しません。

- ROLLBACK は、テーブルロックを解放しません。

LOCK TABLES とトリガー

LOCK TABLES によって明示的にテーブルをロックした場合は、トリガーで使用されているテーブルもすべて暗黙的にロックされます。

- これらのロックは、LOCK TABLES ステートメントによって明示的に取得されるロックと同時に取得されます。
- トリガーで使用されているテーブルに対するロックは、そのテーブルが読み取りのみに使用されているかどうかによって異なります。読み取りのみに使用されている場合は、読み取りロックで十分です。そうでない場合は、書き込みロックが使用されます。
- テーブルが LOCK TABLES によって読み取りに対して明示的にロックされているが、トリガー内で変更される可能性があるために書き込みに対してロックする必要がある場合は、読み取りロックではなく書き込みロックが取得さ

れます。(つまり、トリガー内でのテーブルの表示のために必要な暗黙の書き込みロックによって、テーブルに対する明示的な読み取りロック要求が書き込みロック要求に変換されます。)

次のステートメントを使用して、2つのテーブル `t1` と `t2` をロックするとします。

```
LOCK TABLES t1 WRITE, t2 READ;
```

`t1` または `t2` にトリガーがある場合、トリガー内で使用されるテーブルもロックされます。`t1` に、次のように定義されたトリガーが含まれているとします。

```
CREATE TRIGGER t1_a_ins AFTER INSERT ON t1 FOR EACH ROW
BEGIN
  UPDATE t4 SET count = count+1
  WHERE id = NEW.id AND EXISTS (SELECT a FROM t3);
  INSERT INTO t2 VALUES(1, 2);
END;
```

LOCK TABLES ステートメントの結果として、`t1` と `t2` は、このステートメントに現れるためにロックされます。また、`t3` と `t4` は、トリガー内で使用されているためにロックされます。

- `t1` は、**WRITE** ロック要求ごとに、書き込みに対してロックされます。
- `t2` は、要求が **READ** ロックに対するものであったとしても、書き込みに対してロックされます。これは、トリガー内で `t2` に挿入されるために発生します。したがって、**READ** 要求は **WRITE** 要求に変換されます。
- `t3` は、トリガー内から読み取られるだけであるため、読み取りに対してロックされます。
- `t4` は、トリガー内で更新される可能性があるため、書き込みに対してロックされます。

テーブルロックの制限と条件

テーブルロックを待機しているセッションを終了するために、**KILL** を安全に使用できます。 [セクション 13.7.8.4 「KILL ステートメント」](#) を参照してください。

LOCK TABLES および **UNLOCK TABLES** は、ストアードプログラム内では使用できません。

`performance_schema` データベース内のテーブルは、`setup_xxx` テーブルを除き、**LOCK TABLES** ではロックできません。

LOCK TABLES ステートメントが有効になっている間、次のステートメントは禁止されます。**CREATE TABLE**、**CREATE TABLE ... LIKE**、**CREATE VIEW**、**DROP VIEW**、およびストアードファンクション、ストアードプロシージャ、イベントでの DDL ステートメント。

一部の操作では、`mysql` データベース内のシステムテーブルにアクセスする必要があります。たとえば、**HELP** ステートメントにはサーバー側のヘルプテーブルの内容が必要であり、また **CONVERT_TZ()** はタイムゾーンテーブルの読み取りが必要になる可能性があります。サーバーは、ユーザーが明示的にロックしなくても済むように、必要に応じてシステムテーブルを読み取りに対して暗黙的にロックします。次のテーブルは、今説明したように処理されません。

```
mysql.help_category
mysql.help_keyword
mysql.help_relation
mysql.help_topic
mysql.time_zone
mysql.time_zone_leap_second
mysql.time_zone_name
mysql.time_zone_transition
mysql.time_zone_transition_type
```

これらのテーブルのいずれかに対する **WRITE** ロックを **LOCK TABLES** ステートメントで明示的に設定する場合は、そのテーブルがロックされる唯一のテーブルである必要があります。ほかのどのテーブルも、同じステートメントではロックできません。

1つの **UPDATE** ステートメントはすべてアトミックであるため、通常、テーブルをロックする必要はありません。現在実行中の SQL ステートメントを、ほかのどのセッションも妨げることはできません。ただし、テーブルのロックによって利点が見られる可能性のある場合がいくつかあります。

- 一連の **MyISAM** テーブルに対して多くの操作を実行しようとしている場合は、使用しようとしているテーブルをロックする方がはるかに高速です。 **MyISAM** テーブルをロックすると、MySQL はロックされたテーブルのキーキャッシュを **UNLOCK TABLES** が呼び出されるまでフラッシュしないため、そのテーブルに対する挿入、更新、または削除が高速化されます。通常、キーキャッシュは各 SQL ステートメントのあとでフラッシュされます。

テーブルロックのマイナス面は、**READ** によってロックされたテーブルをどのセッションも更新できず (ロックを保持しているセッションを含む)、ロックを保持しているセッションを除き、**WRITE** によってロックされたテーブルにどのセッションもアクセスできない点です。

- 非トランザクションストレージエンジンに対してテーブルを使用している場合、**SELECT** と **UPDATE** の間にテーブルがほかのセッションによって変更されないようにするには、**LOCK TABLES** を使用する必要があります。次に示す例では、安全に実行するために **LOCK TABLES** が必要です。

```
LOCK TABLES trans READ, customer WRITE;
SELECT SUM(value) FROM trans WHERE customer_id=some_id;
UPDATE customer
  SET total_value=sum_from_previous_statement
  WHERE customer_id=some_id;
UNLOCK TABLES;
```

LOCK TABLES を使用しない場合は、**SELECT** ステートメントと **UPDATE** ステートメントの実行の間に、別のセッションによって **trans** テーブルに新しい行が挿入される可能性があります。

多くの場合は、相対的な更新 (**UPDATE customer SET value=value+new_value**) または **LAST_INSERT_ID()** 関数を使用することによって **LOCK TABLES** の使用を回避できます。

場合によっては、ユーザーレベルのアドバイザリロック関数 **GET_LOCK()** および **RELEASE_LOCK()** を使用してテーブルのロックを回避することもできます。高速化のために、これらのロックはサーバーのハッシュテーブル内に保存され、**pthread_mutex_lock()** と **pthread_mutex_unlock()** で実装されます。 [セクション12.15「ロック関数」](#) を参照してください。

ロックポリシーの詳細は、 [セクション8.11.1「内部ロック方法」](#) を参照してください。

13.3.7 SET TRANSACTION ステートメント

```
SET [GLOBAL | SESSION] TRANSACTION
  transaction_characteristic [, transaction_characteristic] ...

transaction_characteristic: {
  ISOLATION LEVEL level
  | access_mode
}

level: {
  REPEATABLE READ
  | READ COMMITTED
  | READ UNCOMMITTED
  | SERIALIZABLE
}

access_mode: {
  READ WRITE
  | READ ONLY
}
```

このステートメントは、**トランザクション**の特性を指定します。これは、カンマで区切られた1つ以上の特性値のリストを受け取ります。各特性値によって、トランザクション **isolation level** またはアクセスモードが設定されます。分離レベルは、**InnoDB** テーブルに対する操作に使用されます。アクセスモードでは、トランザクションが読取り/書き込みモードと読取り専用モードのどちらで動作するかを指定します。

さらに、**SET TRANSACTION** には、ステートメントのスコープを示すオプションの **GLOBAL** または **SESSION** キーワードを含めることができます。

- [トランザクション分離レベル](#)
- [トランザクションアクセスモード](#)

- [トランザクション特性スコープ](#)

トランザクション分離レベル

トランザクション分離レベルを設定するには、`ISOLATION LEVEL level` 句を使用します。同じ `SET TRANSACTION` ステートメントで複数の `ISOLATION LEVEL` 句を指定することはできません。

デフォルトの分離レベルは `REPEATABLE READ` です。許可されるその他の値は、`READ COMMITTED`、`READ UNCOMMITTED` および `SERIALIZABLE` です。これらの分離レベルの詳細は、[セクション15.7.2.1「トランザクション分離レベル」](#) を参照してください。

トランザクションアクセスモード

トランザクションアクセスモードを設定するには、`READ WRITE` または `READ ONLY` 句を使用します。同じ `SET TRANSACTION` ステートメントで複数のアクセスモード句を指定することはできません。

デフォルトでは、トランザクションは読み取り/書き込みモードで実行され、そのトランザクションで使用されるテーブルに対して読み取りと書き込みの両方が許可されます。このモードは、`READ WRITE` のアクセスモードで `SET TRANSACTION` を使用して明示的に指定できます。

トランザクションアクセスモードが `READ ONLY` に設定されている場合は、テーブルへの変更が禁止されます。これにより、書き込みが許可されていない場合に可能になる、ストレージエンジンのパフォーマンス向上が実現される可能性があります。

読み取り専用モードでは、DML ステートメントを使用して `TEMPORARY` キーワードで作成されたテーブルは引き続き変更できます。永続的なテーブルと同様に、DDL ステートメントによって行われる変更は許可されません。

`READ WRITE` および `READ ONLY` アクセスモードは、`START TRANSACTION` ステートメントを使用して個々のトランザクションに対しても指定できます。

トランザクション特性スコープ

トランザクション特性は、グローバルに設定することも、現在のセッションに対して設定することも、次のトランザクションに対してのみ設定することもできます：

- `GLOBAL` キーワードを使用する場合：
 - ステートメントは、後続のすべてのセッションにグローバルに適用されます。
 - 既存のセッションは影響を受けません。
- `SESSION` キーワードを使用する場合：
 - このステートメントは、現行のセッション内で実行される後続のすべてのトランザクションに適用されます。
 - ステートメントはトランザクション内では許可されますが、現在進行中のトランザクションには影響しません。
 - トランザクション間で実行された場合、このステートメントは、指定された特性の次のトランザクション値を設定する前述のステートメントをオーバーライドします。
- `SESSION` または `GLOBAL` キーワードを使用しない場合：
 - このステートメントは、セッション内で次に実行される単一のトランザクションにのみ適用されます。
 - 後続のトランザクションは、指定された特性のセッション値を使用するように戻ります。
 - ステートメントはトランザクション内では許可されません：

```
mysql> START TRANSACTION;  
Query OK, 0 rows affected (0.02 sec)
```

```
mysql> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
ERROR 1568 (25001): Transaction characteristics can't be changed  
while a transaction is in progress
```

グローバルトランザクション特性を変更するには、`CONNECTION_ADMIN` 権限 (または非推奨の `SUPER` 権限) が必要です。任意のセッションで、セッション特性 (トランザクションの途中であっても) または次のトランザクションの特性 (そのトランザクションの開始前) を自由に変更できます。

サーバーの起動時にグローバル分離レベルを設定するには、コマンド行またはオプションファイルで `--transaction-isolation=level` オプションを使用します。このオプションの `level` の値では、スペースではなくダッシュが使用されるため、許可される値は `READ-UNCOMMITTED`、`READ-COMMITTED`、`REPEATABLE-READ`、または `SERIALIZABLE` です。

同様に、サーバーの起動時にグローバルトランザクションアクセスモードを設定するには、`--transaction-read-only` オプションを使用します。デフォルトは `OFF` (読取り/書込みモード) ですが、読取り専用モードの場合は値を `ON` に設定できます。

たとえば、分離レベルを `REPEATABLE READ` に、アクセスモードを `READ WRITE` に設定するには、オプションファイルの `[mysqld]` セクションで次の行を使用します:

```
[mysqld]
transaction-isolation = REPEATABLE-READ
transaction-read-only = OFF
```

実行時に、前述のように、グローバル、セッションおよび次のトランザクションスコープレベルの特性を `SET TRANSACTION` ステートメントを使用して間接的に設定できます。また、`SET` ステートメントを使用して直接設定し、`transaction_isolation` および `transaction_read_only` システム変数に値を割り当てることもできます:

- `SET TRANSACTION` では、様々なスコープレベルでトランザクション特性を設定するために、オプションの `GLOBAL` および `SESSION` キーワードを使用できます。
- `transaction_isolation` および `transaction_read_only` システム変数に値を割り当てるための `SET` ステートメントには、これらの変数を様々なスコープレベルで設定するための構文があります。

次のテーブルに、各 `SET TRANSACTION` で設定される特性スコープレベルと変数割当て構文を示します。

表 13.9 トランザクション特性の SET TRANSACTION 構文

構文	影響を受ける特性スコープ
<code>SET GLOBAL TRANSACTION transaction_characteristic</code>	グローバル
<code>SET SESSION TRANSACTION transaction_characteristic</code>	セッション
<code>SET TRANSACTION transaction_characteristic</code>	次のトランザクションのみ

表 13.10 トランザクション特性の SET 構文

構文	影響を受ける特性スコープ
<code>SET GLOBAL var_name = value</code>	グローバル
<code>SET @@GLOBAL.var_name = value</code>	グローバル
<code>SET PERSIST var_name = value</code>	グローバル
<code>SET @@PERSIST.var_name = value</code>	グローバル
<code>SET PERSIST_ONLY var_name = value</code>	ランタイムへの影響なし
<code>SET @@PERSIST_ONLY.var_name = value</code>	ランタイムへの影響なし
<code>SET SESSION var_name = value</code>	セッション
<code>SET @@SESSION.var_name = value</code>	セッション
<code>SET var_name = value</code>	セッション
<code>SET @@var_name = value</code>	次のトランザクションのみ

実行時にトランザクション特性のグローバル値およびセッション値を確認できます:

```
SELECT @@GLOBAL.transaction_isolation, @@GLOBAL.transaction_read_only;
SELECT @@SESSION.transaction_isolation, @@SESSION.transaction_read_only;
```

13.3.8 XA トランザクション

XA トランザクションのサポートは、InnoDB ストレージエンジンに対して使用できます。MySQL XA 実装は、X/Open CAE ドキュメント分散トランザクション処理: XA 仕様に基づいています。このドキュメントは The Open Group によって発行されており、<http://www.opengroup.org/public/pubs/catalog/c193.htm> で入手できます。現在の XA 実装の制限については、[セクション13.3.8.3「XA トランザクションの制約」](#)で説明されています。

クライアント側には、特殊な要件は何もありません。MySQL サーバーへの XA インタフェースは、XA キーワードで始まる SQL ステートメントで構成されています。MySQL クライアントプログラムは、SQL ステートメントを送信したり、XA ステートメントインタフェースのセマンティクスを理解したりできる必要があります。これらが、最新のクライアントライブラリに対してリンクされている必要はありません。古いクライアントライブラリも機能します。

MySQL Connector/J 5.0.0 以上では、XA SQL ステートメントインタフェースを処理するクラスインタフェースを使用して XA が直接サポートされます。

XA は分散トランザクション、つまり、複数の個別のトランザクションリソースがグローバルトランザクションに参加することを許可する機能をサポートしています。トランザクションリソースは多くの場合 RDBMS ですが、ほかの種類のリソースであってもかまいません。

グローバルトランザクションには、それ自体でトランザクションである複数のアクションが含まれますが、そのすべてがグループとして正常に完了するか、またはすべてがグループとしてロールバックされるかのどちらかである必要があります。基本的に、これは ACID プロパティを「1 レベル上に」拡張することにより、複数の ACID トランザクションを、同じく ACID プロパティを持つグローバル操作のコンポーネントとして連携して実行できるようにします。(非分散トランザクションと同様に、アプリケーションが読取り現象に敏感な場合は、[SERIALIZABLE](#) をお勧めします。[REPEATABLE READ](#) では、分散トランザクションには不十分な場合があります。)

分散トランザクションのいくつかの例:

- あるアプリケーションが、メッセージングサービスを RDBMS と組み合わせる統合ツールとして機能する場合があります。このアプリケーションは、同じくトランザクションデータベースを含む、メッセージの送信、取得、および処理を行うトランザクションがすべて、確実にグローバルトランザクション内で実行されるようにします。これは、「トランザクション電子メール」と考えることができます。
- アプリケーションが、MySQL サーバーや Oracle サーバー (または複数の MySQL サーバー) などの異なるデータベースサーバーに関連するアクションを実行します。ここで、複数のサーバーに関連するアクションは、各サーバーに対してローカルな個別のトランザクションとしてではなく、グローバルトランザクションの一部として実行する必要があります。
- 銀行が口座情報を RDBMS 内に保持し、現金自動預け払い機 (ATM) を通して現金を出し入れしています。ATM のアクションが口座に正しく反映されるように保証することが必要ですが、これは RDBMS だけでは実行できません。グローバルなトランザクションマネージャーが ATM とデータベースリソースを統合して、財務トランザクションの全体的な一貫性を確保します。

グローバルトランザクションを使用するアプリケーションには、1 つまたは複数のリソースマネージャーと 1 つのトランザクションマネージャーが含まれています。

- リソースマネージャー (RM) は、トランザクションリソースへのアクセスを提供します。データベースサーバーは、1 つの種類のリソースマネージャーです。これは、RM によって管理されているトランザクションをコミットまたはロールバックできる必要があります。
- トランザクションマネージャー (TM) は、グローバルトランザクションの一部であるトランザクションを調整します。これは、これらの各トランザクションを処理する RM と通信します。グローバルトランザクション内の個々のトランザクションは、グローバルトランザクションの「ブランチ」です。グローバルトランザクションとそのブランチは、あとで説明されている名付けスキームによって識別されます。

XA の MySQL 実装により、MySQL サーバーはグローバルトランザクション内の XA トランザクションを処理するリソースマネージャーとして機能できます。MySQL サーバーに接続するクライアントプログラムは、トランザクションマネージャーとして機能します。

グローバルトランザクションを実行するには、どのコンポーネントが関連しているかを知り、各コンポーネントをそのコミットまたはロールバックが可能なポイントに持っていくことが必要です。各コンポーネントが自身の成功する

能力に関してレポートする内容に応じて、それらのすべてが、アトミックグループとしてコミットまたはロールバックする必要があります。つまり、すべてのコンポーネントがコミットするか、またはすべてのコンポーネントがロールバックする必要があります。グローバルトランザクションを管理するには、いずれかのコンポーネントまたは接続しているネットワークが失敗する可能性があることを考慮に入れる必要があります。

グローバルトランザクションを実行するためのプロセスでは、2 フェーズコミット (2PC) が使用されます。これは、グローバルトランザクションのブランチによって実行されるアクションが実行されたあとに行われます。

1. 最初のフェーズでは、すべてのブランチが準備されます。つまり、これらは TM からコミットの準備を行うよう指示されます。これは通常、ブランチを管理する各 RM が、そのブランチのアクションを安定したストレージ内に記録することを示します。これらのブランチはこれを実行できるかどうかを示し、これらの結果が 2 番目のフェーズに使用されます。
2. 2 番目のフェーズでは、TM が RM にコミットまたはロールバックのどちらを行うかを指示します。すべてのブランチが準備されたときにコミット可能であることを示した場合、すべてのブランチにコミットするように指示されます。コミットできなかったことを準備したときにブランチが示された場合は、すべてのブランチにロールバックするように指示されます。

場合によっては、グローバルトランザクションで 1 フェーズコミット (1PC) が使用されることがあります。たとえば、グローバルトランザクションが 1 つのトランザクションリソース (つまり、1 つのブランチ) だけで構成されていることがトランザクションマネージャーによって検出された場合は、そのリソースに準備とコミットを一度に行うよう指示できます。

13.3.8.1 XA トランザクション SQL ステートメント

MySQL で XA トランザクションを実行するには、次のステートメントを使用します。

```
XA (START|BEGIN) xid [JOIN|RESUME]
XA END xid [SUSPEND [FOR MIGRATE]]
XA PREPARE xid
XA COMMIT xid [ONE PHASE]
XA ROLLBACK xid
XA RECOVER [CONVERT XID]
```

XA START の場合、JOIN 句および RESUME 句は認識されますが、効果はありません。

XA END の場合、SUSPEND [FOR MIGRATE] 句は認識されますが、効果はありません。

各 XA ステートメントは XA キーワードで始まり、そのほとんどに *xid* 値が必要です。 *xid* は XA トランザクション識別子です。これは、このステートメントがどのトランザクションに適用されるかを示します。 *xid* 値はクライアントによって指定されるか、または MySQL サーバーによって生成されます。 *xid* 値には、1 つから 3 つの部分が含まれています。

```
xid: gtrid [, bqual [, formatID ]]
```

gtrid はグローバルトランザクション識別子であり、*bqual* はブランチ修飾子であり、*formatID* は、*gtrid* および *bqual* 値で使用される形式を識別する数値です。 構文で示されているように、*bqual* と *formatID* はオプションです。 *bqual* が指定されていない場合、そのデフォルト値は " です。 *formatID* が指定されていない場合、そのデフォルト値は 1 です。

gtrid と *bqual* はそれぞれ、最大 64 バイト長 (64 文字ではありません) の文字列リテラルである必要があります。 *gtrid* と *bqual* は、いくつかの方法で指定できます。 引用符付き文字列 ('ab')、16 進文字列 (X'6162'、0x6162) またはビット値 (b'nnnn') を使用できます。

formatID は符号なし整数です。

gtrid および *bqual* 値は、MySQL サーバーのベースとなる XA サポートルーチンによってバイト単位で解釈されます。 ただし、XA ステートメントを含む SQL ステートメントが解析されている間、サーバーは何からの特定の文字セットで動作します。 安全のために、*gtrid* と *bqual* は 16 進文字列として記述してください。

`xid` 値は通常、トランザクションマネージャーによって生成されます。ある TM によって生成される値は、ほかの TM によって生成される値とは異なっている必要があります。特定の TM は、`XA RECOVER` ステートメントによって返された値のリスト内の自身の `xid` 値を認識できる必要があります。

`XA START xid` は、指定された `xid` 値を使用して XA トランザクションを開始します。各 XA トランザクションが一意的な `xid` 値を持っている必要があるため、その値が現在、別の XA トランザクションによって使用されてはいけません。一意性は、`gtrid` および `bqual` 値を使用して評価されます。XA トランザクションに対する以降のすべての XA ステートメントを、`XA START` ステートメントで指定されたものと同じ `xid` 値を使用して指定する必要があります。これらのステートメントのいずれかを使用しているが、既存の XA トランザクションに対応していない `xid` 値を指定した場合は、エラーが発生します。

1 つ以上の XA トランザクションを同じグローバルトランザクションの一部にすることができます。特定のグローバルトランザクション内のすべての XA トランザクションが `xid` 値内の同じ `gtrid` 値を使用する必要があります。このため、特定の XA トランザクションがどのグローバルトランザクションの一部であるかについてのあいまいさがないように、`gtrid` 値はグローバルに一意である必要があります。`xid` 値の `bqual` 部分は、グローバルトランザクション内の XA トランザクションごとに異なっている必要があります。(`bqual` 値が異なっているという要件は、現在の MySQL XA 実装の制限です。これは XA 仕様の一部ではありません。)

`XA RECOVER` ステートメントは、`PREPARED` 状態にある MySQL サーバー上の XA トランザクションに関する情報を返します。([セクション 13.3.8.2「XA トランザクションの状態」](#) を参照してください。) この出力には、どのクライアントによって開始されたかには関係なく、サーバー上のこのような XA トランザクションごとの行が含まれています。

`XA RECOVER` には、`XA_RECOVER_ADMIN` 権限が必要です。この権限要件により、ユーザーは自分以外の未処理の準備済 XA トランザクションの `XID` 値を検出できなくなります。XA トランザクションを開始したユーザーが `XID` を認識しているため、XA トランザクションの通常のコミットまたはロールバックには影響しません。

`XA RECOVER` の出力行は次のようになります ('abc'、'def'、7 の各部分から成る `xid` 値の例の場合)。

```
mysql> XA RECOVER;
+-----+-----+-----+-----+
| formatID | gtrid_length | bqual_length | data |
+-----+-----+-----+-----+
| 7 | 3 | 3 | abcdef |
+-----+-----+-----+-----+
```

出力カラムには次の意味があります。

- `formatID` は、トランザクション `xid` の `formatID` 部分です。
- `gtrid_length` は、`xid` の `gtrid` 部分の長さ (バイト単位) です。
- `bqual_length` は、`xid` の `bqual` 部分の長さ (バイト単位) です。
- `data` は、`xid` の `gtrid` および `bqual` 部分の連結です。

`XID` 値に印刷不可能な文字が含まれる場合があります。`XA RECOVER` では、クライアントが `XID` 値を 16 進数でリクエストできるように、オプションの `CONVERT XID` 句を使用できます。

13.3.8.2 XA トランザクションの状態

XA トランザクションは、次の各状態を経由して処理されます。

1. `XA START` を使用して、XA トランザクションを開始し、それを `ACTIVE` 状態にします。
2. `ACTIVE` XA トランザクションに対しては、トランザクションを構成する SQL ステートメントを発行したあと、`XA END` ステートメントを発行します。`XA END` は、トランザクションを `IDLE` 状態にします。
3. `IDLE` XA トランザクションに対しては、`XA PREPARE` ステートメントまたは `XA COMMIT ... ONE PHASE` ステートメントのどちらかを発行できます。
 - `XA PREPARE` は、トランザクションを `PREPARED` 状態にします。`XA RECOVER` では `PREPARED` 状態の XA トランザクションがすべてリストされるため、この時点での `XA RECOVER` ステートメントの出力にはトランザクションの `xid` 値が含まれます。

- **XA COMMIT ... ONE PHASE** は、トランザクションの準備とコミットを行います。トランザクションが終了するため、**xid** 値は **XA RECOVER** によってリストされません。
4. **PREPARED XA** トランザクションに対しては、**XA COMMIT** ステートメントを発行してトランザクションをコミットおよび終了するか、または **XA ROLLBACK** を発行してトランザクションをロールバックおよび終了することができます。

グローバルトランザクションの一部としてテーブルに行を挿入する単純な XA トランザクションを次に示します。

```
mysql> XA START 'xatest';
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO mytable (i) VALUES(10);
Query OK, 1 row affected (0.04 sec)

mysql> XA END 'xatest';
Query OK, 0 rows affected (0.00 sec)

mysql> XA PREPARE 'xatest';
Query OK, 0 rows affected (0.00 sec)

mysql> XA COMMIT 'xatest';
Query OK, 0 rows affected (0.00 sec)
```

特定のクライアント接続のコンテキスト内では、XA トランザクションとローカル (非 XA) トランザクションは相互に排他的です。たとえば、XA トランザクションを開始するために **XA START** が発行された場合は、その XA トランザクションがコミットまたはロールバックされるまでローカルトランザクションを開始できません。逆に、**START TRANSACTION** を使用してローカルトランザクションが開始された場合は、そのトランザクションがコミットまたはロールバックされるまで XA ステートメントを使用できません。

XA トランザクションが **ACTIVE** 状態の場合、暗黙的なコミットを引き起こすステートメントは発行できません。その XA トランザクションをロールバックできないため、それを行うことは XA 規約に違反します。このようなステートメントを実行しようとすると、次のエラーが発生します:

```
ERROR 1399 (XAE07): XAER_RMFAIL: The command cannot be executed
when global transaction is in the ACTIVE state
```

前の注意事項が適用されるステートメントは、[セクション13.3.3「暗黙的なコミットを発生させるステートメント」](#)に示されています。

13.3.8.3 XA トランザクションの制約

XA トランザクションのサポートは、**InnoDB** ストレージエンジンに限られています。

「外部 XA」の場合、MySQL Server がリソースマネージャーとして機能し、クライアントプログラムがトランザクションマネージャーとして機能します。「内部 XA」の場合、MySQL サーバー内のストレージエンジンがリソースマネージャーとして機能し、サーバー自体がトランザクションマネージャーとして機能します。内部 XA サポートは、個々のストレージエンジンの機能によって制限されています。内部 XA は、複数のストレージエンジンが関与する XA トランザクションを処理するために必要になります。内部 XA の実装では、ストレージエンジンがテーブルハンドラレベルでの 2 フェーズコミットをサポートしている必要であり、現在これは **InnoDB** にのみ当てはまります。

XA START の場合、**JOIN** 句および **RESUME** 句は認識されませんが、効果はありません。

XA END の場合、**SUSPEND [FOR MIGRATE]** 句は認識されませんが、効果はありません。

xid 値の **bqual** 部分が、グローバルトランザクション内の XA トランザクションごとに異なる必要があるという要件が、現在の MySQL XA 実装の制限です。これは XA の仕様によるものではありません。

XA トランザクションは、2 つの部分でバイナリログに書き込まれます。**XA PREPARE** が発行されると、**XA PREPARE** までのトランザクションの最初の部分が初期 GTID を使用して書き込まれます。**XA_prepare_log_event** は、バイナリログ内のこのようなトランザクションを識別するために使用されます。**XA COMMIT** または **XA ROLLBACK** が発行されると、**XA COMMIT** または **XA ROLLBACK** ステートメントのみを含むトランザクションの別の部分が、別の GTID を使用して書き込まれます。**XA_prepare_log_event** で識別されるトランザクションの最初の部

分の後に、必ずしも **XA COMMIT** または **XA ROLLBACK** が続くわけではありません。これにより、任意の 2 つの XA トランザクションのインターリーブバイナリロギングが発生する可能性があります。XA トランザクションの 2 つの部分は、異なるバイナリログファイルでも使用できます。つまり、**PREPARED** 状態の XA トランザクションは、明示的な **XA COMMIT** ステートメントまたは **XA ROLLBACK** ステートメントが発行されるまで永続的になり、XA トランザクションがレプリケーションと互換性を持つようになります。

レプリカでは、XA トランザクションが準備されるとすぐにレプリケーションアプライヤスレッドからデタッチされ、レプリカ上の任意のスレッドによってコミットまたはロールバックできます。これは、同じ XA トランザクションが、異なるスレッド上の異なる状態で `events_transactions_current` テーブルに表示されることを意味します。`events_transactions_current` テーブルには、スレッド上の最新の監視対象トランザクションイベントの現在のステータスが表示され、スレッドがアイドル状態の場合、このステータスは更新されません。そのため、XA トランザクションは、別のスレッドによって処理された後も、元のアプライヤスレッドの **PREPARED** 状態で表示できます。**PREPARED** 状態のままレプリカが必要となる XA トランザクションを肯定的に識別するには、パフォーマンススキーマ `events_transactions_current` テーブルではなく、**XA RECOVER** ステートメントを使用します。

XA トランザクションの使用には、次の制限があります：

- XA トランザクションは、バイナリログに関して予期しない停止に対する完全な回復力がありません。サーバーが **XA PREPARE**, **XA COMMIT**, **XA ROLLBACK** または **XA COMMIT ... ONE PHASE** ステートメントの実行中に予期しない停止が発生した場合、サーバーは正しい状態に回復できず、サーバーとバイナリログが一貫性のない状態のままになる可能性があります。この状況では、バイナリログに適用されていない追加の XA トランザクションが含まれているか、適用されている XA トランザクションが失われている可能性があります。また、GTID が有効になっている場合は、回復後に、`@@GLOBAL.GTID_EXECUTED` が適用されたトランザクションを正しく記述しないことがあります。**XA PREPARE** の前、**XA PREPARE** と **XA COMMIT** (または **XA ROLLBACK**) の間、または **XA COMMIT** (または **XA ROLLBACK**) の後に予期しない停止が発生した場合、サーバーおよびバイナリログは正しくリカバリされ、一貫性のある状態になります。
- XA トランザクションと組み合わせたレプリケーションフィルタまたはバイナリログフィルタの使用はサポートされていません。テーブルをフィルタリングすると、レプリカで XA トランザクションが空になる可能性があり、空の XA トランザクションはサポートされません。また、MySQL 8.0 のデフォルトになった **InnoDB** テーブルに格納されているレプリカ接続メタデータリポジトリおよびアプライヤメタデータリポジトリでは、フィルタ処理された XA トランザクションの後にデータエンジントランザクションの内部状態が変更され、レプリケーショントランザクションコンテキストの状態と一貫性がなくなる可能性があります。

XA トランザクションがレプリケーションフィルタの影響を受けるたびに、トランザクションが空であったかどうかに関係なく、エラー **ER_XA_REPLICATION_FILTERS** がログに記録されます。トランザクションが空でない場合、レプリカは実行を続行できますが、潜在的な問題を回避するために XA トランザクションでのレプリケーションフィルタの使用を中止するステップを実行する必要があります。トランザクションが空の場合、レプリカは停止します。その場合、レプリケーションプロセスの一貫性が損なわれる可能性のある不確定な状態にレプリカがある可能性があります。特に、レプリカのレプリカ上の `gtid_executed` セットは、ソース上のものと一貫性がない可能性があります。この状況を解決するには、ソースを分離し、すべてのレプリケーションを停止してから、レプリケーショントポロジ全体で GTID の一貫性をチェックします。エラーメッセージを生成した XA トランザクションを元に戻し、レプリケーションを再開します。

- XA トランザクションは、ステートメントベースレプリケーションでは安全でないとみなされます。ソースでパラレルにコミットされた 2 つの XA トランザクションがレプリカで逆の順序で準備されている場合、安全に解決できないロック依存性が発生する可能性があり、レプリケーションがレプリカのデッドロックで失敗する可能性があります。この状況は、シングルスレッドまたはマルチスレッドレプリカで発生する可能性があります。`binlog_format=STATEMENT` が設定されている場合、XA トランザクション内の DML ステートメントに対して警告が発行されます。`binlog_format=MIXED` または `binlog_format=ROW` が設定されている場合、XA トランザクション内の DML ステートメントは行ベースのレプリケーションを使用して記録され、潜在的な問題は存在しません。

注記

MySQL 5.7.7 より前は、XA トランザクションはレプリケーションとまったく互換性がありませんでした。これは、**PREPARED** 状態の XA トランザクションが、サーバーのクリーンシャットダウンまたはクライアントの切断時にロールバックされるためです。同様に、サーバーが異常停止してから再度起動されたが、トランザクションの内容をバイナリログに書き込めなかった場合、**PREPARED** 状態の XA トランザクションは **PREPARED** 状態のままになります。どちらの状況でも、XA トランザクションを正しくレプリケートできませんでした。

13.4 レプリケーションステートメント

レプリケーションは、このセクションで説明されているステートメントを使用した SQL インタフェースを通して制御できます。ステートメントは、ソースサーバーを制御するグループ、レプリカサーバーを制御するグループ、および任意のレプリケーションサーバーに適用できるグループに分割されます。

13.4.1 ソースサーバーを制御する SQL ステートメント

このセクションでは、レプリケーションソースサーバーを管理するためのステートメントについて説明します。[セクション13.4.2「レプリケーションサーバーを制御するための SQL ステートメント」](#)では、レプリカサーバーを管理するためのステートメントについて説明します。

ここで説明するステートメントに加えて、レプリケーションでは次の `SHOW` ステートメントがソースサーバーで使用されます。これらのステートメントについては、[セクション13.7.7「SHOW ステートメント」](#)を参照してください。

- `SHOW BINARY LOGS`
- `SHOW BINLOG EVENTS`
- `SHOW MASTER STATUS`
- `SHOW REPLICAS | SHOW SLAVE HOSTS`

13.4.1.1 PURGE BINARY LOGS ステートメント

```
PURGE { BINARY | MASTER } LOGS {  
  TO 'log_name'  
  | BEFORE datetime_expr  
}
```

バイナリログは、MySQL サーバーによって行われたデータ変更に関する情報を含む一連のファイルです。このログは一連のバイナリログファイルのほか、インデックスファイルで構成されています ([セクション5.4.4「バイナリログ」](#)を参照してください)。

`PURGE BINARY LOGS` ステートメントは、指定されたログファイル名または日付の前にあるログインデックスファイルにリストされているすべてのバイナリログファイルを削除します。`BINARY` と `MASTER` はシノニムです。削除されたログファイルはインデックスファイル内に記録されているリストからも削除されるため、特定のログファイルがそのリスト内の先頭になります。

`PURGE BINARY LOGS` には、`BINLOG_ADMIN` 権限が必要です。このステートメントは、サーバーがバイナリロギングを有効にする `--log-bin` オプションで起動されていない場合は何の効果もありません。

例:

```
PURGE BINARY LOGS TO 'mysql-bin.010';  
PURGE BINARY LOGS BEFORE '2019-04-02 22:46:26';
```

`BEFORE` バリエント `datetime_expr` 引数は、`DATETIME` 値 ('YYYY-MM-DD hh:mm:ss'形式の値) に評価される必要があります。

このステートメントは、レプリカのレプリケート中に安全に実行できます。それらを停止する必要はありません。現在削除しようとしているログファイルのいずれかを読み取っているアクティブレプリカがある場合、このステートメントでは、使用中のログファイルまたはそのログファイルより後のログファイルは削除されませんが、以前のログファイルは削除されます。この状況では、警告メッセージが発行されます。ただし、レプリカが接続されておらず、まだ読み取られていないログファイルのいずれかをパージする場合、レプリカは再接続後にレプリケートできません。

バイナリログファイルを安全にパージするには、次の手順に従います。

1. 各レプリカで、`SHOW REPLICA | SLAVE STATUS` を使用して読み取るログファイルを確認します。
2. `SHOW BINARY LOGS` を使用して、ソースのバイナリログファイルのリストを取得します。

3. すべてのレプリカの中で最も古いログファイルを確認します。これがターゲットファイルです。すべてのレプリカが最新の場合、これはリストの最後のログファイルです。
4. 削除しようとしているすべてのログファイルのバックアップを作成します。(この手順はオプションですが、常に行うことをお勧めします。)
5. ターゲットファイルの直前までのすべてのログファイルをパージします。

[PURGE BINARY LOGS TO](#) と [PURGE BINARY LOGS BEFORE](#) はどちらも、`.index` ファイルにリストされているバイナリログファイルが、ほかの何らかの手段 (Linux 上での `rm` の使用など) によってシステムから削除されている場合はエラーで失敗します。(Bug #18199、Bug #18453) このようなエラーに対処するには、`.index` ファイル (これは単純なテキストファイルです) を手動で編集して、実際に存在するバイナリログファイルのみがリストされていることを確認したあと、失敗した [PURGE BINARY LOGS](#) ステートメントを再度実行します。

バイナリログファイルは、サーバーのバイナリログの有効期限後に自動的に削除されます。ファイルの削除は、起動時およびバイナリログのフラッシュ時に実行できます。デフォルトのバイナリログの有効期限は 30 日です。`binlog_expire_logs_seconds` システム変数を使用して、別の有効期限を指定できます。レプリケーションを使用している場合は、レプリカがソースより遅れる可能性のある最大時間以下の有効期限を指定する必要があります。

13.4.1.2 RESET MASTER ステートメント

```
RESET MASTER [TO binary\_log\_file\_index\_number]
```

警告

このステートメントは、必要なバイナリログファイルデータおよび GTID 実行履歴が失われないように注意して使用してください。

[RESET MASTER](#) には、[RELOAD](#) 権限が必要です。

バイナリロギングが有効になっている (`log_bin` が `ON`) サーバーの場合、[RESET MASTER](#) は既存のバイナリログファイルをすべて削除し、バイナリログインデックスファイルをリセットして、バイナリロギングが開始される前の状態にサーバーをリセットします。バイナリロギングを再開できるように、新しい空のバイナリログファイルが作成されます。

GTID が使用されている (`gtid_mode` が `ON`) サーバーの場合、[RESET MASTER](#) を発行すると GTID 実行履歴がリセットされます。`gtid_purged` システム変数の値は空の文字列 ("") に設定され、`gtid_executed` システム変数のグローバル値 (セッション値ではない) は空の文字列に設定され、`mysql.gtid_executed` テーブルはクリアされます ([mysql.gtid_executed テーブル](#) を参照)。GTID 対応サーバーでバイナリロギングが有効になっている場合、[RESET MASTER](#) は前述のようにバイナリログもリセットします。GTID 対応サーバーがバイナリロギングが無効になっているレプリカであっても、[RESET MASTER](#) は GTID 実行履歴をリセットする方法であることに注意してください。[RESET REPLICA | SLAVE](#) は GTID 実行履歴に影響しません。GTID 実行履歴のリセットの詳細は、[GTID 実行履歴のリセット](#) を参照してください。

オプションの `TO` 句を指定せずに [RESET MASTER](#) を発行すると、インデックスファイルにリストされているすべてのバイナリログファイルが削除され、バイナリログインデックスファイルが空にリセットされ、1 から始まる新しいバイナリログファイルが作成されます。リセット後に 1 以外の番号からバイナリログファイルのインデックスを開始するには、オプションの `TO` 句を使用します。

[RESET MASTER](#) を `TO` 句とともに使用してバイナリログファイルのインデックス番号を指定すると、[FLUSH BINARY LOGS](#) および [PURGE BINARY LOGS TO](#) ステートメントの代わりに単一のステートメントが提供されるため、フェイルオーバーが簡略化されます。インデックス番号に適切な値を使用していることを確認します。間違った値を入力した場合は、`TO` 句を指定して、または指定せずに別の [RESET MASTER](#) ステートメントを発行することで、これを修正できます。範囲外の値を修正しないと、サーバーを再起動できません。

次の例では、`TO` 句の使用方法を示します:

```
RESET MASTER TO 1234;  
  
SHOW BINARY LOGS;  
+-----+-----+-----+  
| Log_name | File_size | Encrypted |
```

```
+-----+-----+
| source-bin.001234 | 154 | No |
+-----+-----+
```

重要

TO 句を使用しない **RESET MASTER** の効果は、**PURGE BINARY LOGS** の効果と 2 つの重要な点で異なります:

1. **RESET MASTER** が、インデックスファイルにリストされているすべてのバイナリログファイルを削除し、.000001 の数字のサフィクスを持つ 1 つの空のバイナリログファイルだけを残すのに対して、**PURGE BINARY LOGS** では番号はリセットされません。
2. **RESET MASTER** は、レプリカの実行中に使用することを意図していません。レプリカの実行中に **RESET MASTER** を使用した場合の動作は定義されていません (したがってサポートされていません)。一方、レプリカの実行中は **PURGE BINARY LOGS** を安全に使用できます。

[セクション13.4.1.1「PURGE BINARY LOGS ステートメント」](#) も参照してください。

TO 句のない **RESET MASTER** は、ソースおよびレプリカを最初に設定するときに役立つことがあるため、次のように設定を検証できます:

1. ソースとレプリカを起動し、レプリケーションを開始します ([セクション17.1.2「バイナリログファイルの位置ベースのレプリケーションの設定」](#) を参照)。
2. ソースでいくつかのテストクエリーを実行します。
3. クエリーがレプリカにレプリケートされたことを確認します。
4. レプリケーションが正しく実行されている場合は、**STOP REPLICAS | SLAVES** を発行してからレプリカで **RESET REPLICAS | SLAVES** を発行し、テストクエリーからの不要なデータがレプリカに存在しないことを確認します。
5. ソースで **RESET MASTER** を発行して、テストクエリーをクリーンアップします。

設定の確認、ソースおよびレプリカのリセット、およびテストによって生成された不要なデータまたはバイナリログファイルがソースまたはレプリカに残っていないことの確認後、レプリカを起動してレプリケートを開始できます。

13.4.1.3 SET sql_log_bin ステートメント

```
SET sql_log_bin = {OFF|ON}
```

sql_log_bin 変数は、バイナリログへのロギングを現在のセッションで有効にするかどうかを制御します (バイナリログ自体が有効になっていると仮定します)。デフォルト値は **ON** です。現在のセッションのバイナリロギングを無効または有効にするには、セッション **sql_log_bin** 変数を **OFF** または **ON** に設定します。

レプリカにレプリケートしないソースに変更を加えている間にバイナリロギングを一時的に無効にするには、セッションに対してこの変数を **OFF** に設定します。

このシステム変数のセッション値の設定は制限された操作です。セッションユーザーには、制限付きセッション変数を設定するのに十分な権限が必要です。 [セクション5.1.9.1「システム変数権限」](#) を参照してください。

トランザクションまたはサブクエリー内で **sql_log_bin** のセッション値を設定することはできません。

「この変数を **OFF** に設定すると、新しい GTID がバイナリログ内のトランザクションに割り当てられなくなります」。これは、GTID をレプリケーションに使用している場合、バイナリロギングがあとで再度有効になった場合でも、この時点からログに書き込まれる GTID はその意味で発生したトランザクションを考慮しないため、それらのトランザクションは失われることを意味します。

mysqldump は GTID が使用されているサーバーからダンプファイルに **SET @@SESSION.sql_log_bin=0** ステートメントを追加します。これにより、ダンプファイルのリロード中にバイナリロギングが無効になります。このステートメントは、トランザクションの元の GTID が使用されるように、新しい GTID が生成されてダンプファイル内のトランザクションに割り当てられるのを防ぎます。

13.4.2 レプリケーションサーバーを制御するための SQL ステートメント

このセクションでは、レプリカサーバーを管理するためのステートメントについて説明します。[セクション 13.4.1 「ソースサーバーを制御する SQL ステートメント」](#)では、ソースサーバーを管理するためのステートメントについて説明します。

ここで説明するステートメントに加えて、[SHOW REPLICA | SLAVE STATUS](#) および [SHOW RELAYLOG EVENTS](#) もレプリカとともに使用されます。これらのステートメントについては、[セクション 13.7.7.35 「SHOW REPLICA | SLAVE STATUS ステートメント」](#) および [セクション 13.7.7.32 「SHOW RELAYLOG EVENTS ステートメント」](#) を参照してください。

13.4.2.1 CHANGE MASTER TO ステートメント

```
CHANGE MASTER TO option [, option] ... [ channel_option ]

option: {
  MASTER_BIND = 'interface_name'
| MASTER_HOST = 'host_name'
| MASTER_USER = 'user_name'
| MASTER_PASSWORD = 'password'
| MASTER_PORT = port_num
| PRIVILEGE_CHECKS_USER = { 'account' | NULL }
| REQUIRE_ROW_FORMAT = { 0 | 1 }
| REQUIRE_TABLE_PRIMARY_KEY_CHECK = { STREAM | ON | OFF }
| ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS = { OFF | LOCAL | uuid }
| MASTER_LOG_FILE = 'source_log_name'
| MASTER_LOG_POS = source_log_pos
| MASTER_AUTO_POSITION = { 0 | 1 }
| RELAY_LOG_FILE = 'relay_log_name'
| RELAY_LOG_POS = relay_log_pos
| MASTER_HEARTBEAT_PERIOD = interval
| MASTER_CONNECT_RETRY = interval
| MASTER_RETRY_COUNT = count
| SOURCE_CONNECTION_AUTO_FAILOVER = { 0 | 1 }
| MASTER_DELAY = interval
| MASTER_COMPRESSION_ALGORITHMS = 'value'
| MASTER_ZSTD_COMPRESSION_LEVEL = level
| MASTER_SSL = { 0 | 1 }
| MASTER_SSL_CA = 'ca_file_name'
| MASTER_SSL_CAPATH = 'ca_directory_name'
| MASTER_SSL_CERT = 'cert_file_name'
| MASTER_SSL_CRL = 'crl_file_name'
| MASTER_SSL_CRLPATH = 'crl_directory_name'
| MASTER_SSL_KEY = 'key_file_name'
| MASTER_SSL_CIPHER = 'cipher_list'
| MASTER_SSL_VERIFY_SERVER_CERT = { 0 | 1 }
| MASTER_TLS_VERSION = 'protocol_list'
| MASTER_TLS_CIPHERSUITES = 'ciphersuite_list'
| MASTER_PUBLIC_KEY_PATH = 'key_file_name'
| GET_MASTER_PUBLIC_KEY = { 0 | 1 }
| NETWORK_NAMESPACE = 'namespace'
| IGNORE_SERVER_IDS = (server_id_list)
}

channel_option:
  FOR CHANNEL channel

server_id_list:
  [server_id [, server_id] ... ]
```

CHANGE MASTER TO は、レプリカサーバーがソースへの接続およびソースからのデータの読取りに使用するパラメータを変更します。また、レプリケーションメタデータリポジトリの内容も更新されます ([セクション 17.2.4 「リレーログおよびレプリケーションメタデータリポジトリ」](#) を参照)。MySQL 8.0.23 から、**CHANGE MASTER TO** のかわりに **CHANGE REPLICATION SOURCE TO** を使用します。これは、そのリリースから非推奨になりました。MySQL 8.0.23 より前のリリースでは、**CHANGE MASTER TO** を使用します。

レプリケーション SQL スレッドおよびレプリケーション I/O スレッドの状態に応じて、最初に停止せずに、実行中のレプリカに対して **CHANGE MASTER TO** ステートメントを発行できます。このような使用を制御するルールは、このセクションの後半で説明します。**CHANGE MASTER TO** には、**REPLICATION_SLAVE_ADMIN** 権限 (または非推奨の **SUPER** 権限) が必要です。

マルチスレッドのレプリカを使用する場合 (つまり、`slave_parallel_workers` が 0 より大きい場合)、レプリカを停止すると、レプリカが意図的に停止されたかどうかに関係なく、リレーログから実行された一連のトランザクションで「ギャップ」が発生する可能性があります。このようなギャップが存在する場合、`CHANGE MASTER TO` の発行は失敗します。この状況の解決策は、ギャップを確実に閉じる `START REPLICA | SLAVE UNTIL SQL_AFTER_MTS_GAPS` を発行することです。

オプションの `FOR CHANNEL channel` 句を使用すると、ステートメントが適用されるレプリケーションチャンネルの名前を指定できます。`FOR CHANNEL channel` 句を指定すると、`CHANGE MASTER TO` ステートメントが特定のレプリケーションチャンネルに適用され、新しいチャンネルの追加または既存のチャンネルの変更に使用されます。たとえば、`channel2` という新しいチャンネルを追加するには、次のようにします:

```
CHANGE MASTER TO MASTER_HOST=host1, MASTER_PORT=3002 FOR CHANNEL 'channel2'
```

句が指定されておらず、追加のチャンネルが存在しない場合、ステートメントはデフォルトチャンネルに適用されます。

複数のレプリケーションチャンネルを使用する場合、`CHANGE MASTER TO` ステートメントで `FOR CHANNEL channel` 句を使用してチャンネルを指定しないと、エラーが発生します。詳しくは [セクション17.2.2「レプリケーションチャンネル」](#) をご覧ください。

`MASTER_HOST` およびその他の `CHANGE MASTER TO` オプションに使用される値では、改行 (`\n` または `0x0A`) 文字がチェックされます。このような文字がこれらの値に存在すると、`ER_MASTER_INFO` でステートメントが失敗します。

`CHANGE MASTER TO` を起動すると、`MASTER_HOST`、`MASTER_PORT`、`MASTER_LOG_FILE` および `MASTER_LOG_POS` の以前の値が、実行前のレプリカ状態に関するその他の情報とともにエラーログに書き込まれます。

`CHANGE MASTER TO` では、進行中のトランザクションが暗黙的にコミットされます。[セクション13.3.3「暗黙的なコミットを発生させるステートメント」](#) を参照してください。

`CHANGE MASTER TO` ステートメントの一部のオプションでは、`CHANGE MASTER TO` ステートメント (およびその後の `START REPLICA | SLAVE` ステートメント) を発行する前に、`STOP REPLICA | SLAVE` ステートメントを発行する必要があります。場合によっては、レプリケーション SQL スレッドまたはレプリケーション I/O スレッドのどちらか一方のみを停止する必要があります:

- SQL スレッドが停止すると、レプリケーション I/O スレッドが実行されている場合でも、`RELAY_LOG_FILE`、`RELAY_LOG_POS` および `MASTER_DELAY` オプションの任意の組合せを使用して `CHANGE MASTER TO` を実行できます。I/O スレッドの実行中は、このステートメントでほかのオプションを使用できません。
- I/O スレッドが停止すると、SQL スレッドが実行されている場合でも、このステートメント (任意の組合せで) `except RELAY_LOG_FILE, RELAY_LOG_POS, MASTER_DELAY` または `MASTER_AUTO_POSITION = 1` のオプションを使用して `CHANGE MASTER TO` を実行できます。
- `MASTER_AUTO_POSITION = 1` または `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` を使用する `CHANGE MASTER TO` ステートメントを発行する前に、SQL スレッドと I/O スレッドの両方を停止する必要があります。

`SHOW REPLICA | SLAVE STATUS` を使用して、レプリケーション SQL スレッドおよびレプリケーション I/O スレッドの現在の状態を確認できます。Group Replication applier チャンネル (`group_replication_applier`) には I/O スレッドがなく、SQL スレッドのみがあることに注意してください。

詳細は、[セクション17.4.8「フェイルオーバー中のソースの切替え」](#) を参照してください。

ステートメントベースレプリケーションおよび一時テーブルを使用している場合は、`STOP REPLICA | SLAVE` ステートメントのあとの `CHANGE MASTER TO` ステートメントがレプリカ上の一時テーブルの背後に残る可能性があります。これが発生するたびに警告 (`ER_WARN_OPEN_TEMP_TABLES_MUST_BE_ZERO`) が発行されるようになりました。このような場合は、このような `CHANGE MASTER TO` ステートメントを実行する前に、`Slave_open_temp_tables` システムステータス変数の値が 0 であることを確認することで回避できます。

`CHANGE MASTER TO` は、ソースのスナップショットがあり、スナップショットの時刻に対応するソースバイナリログ座標を記録している場合にレプリカを設定する際に役立ちます。スナップショットをレプリカにロードしてソースと同期した後、レプリカで `CHANGE MASTER TO MASTER_LOG_FILE='log_name', MASTER_LOG_POS=log_pos` を実行して、レプリカがソースバイナリログの読取りを開始する座標を指定できます。

次の例では、レプリカが使用するソースサーバーを変更し、レプリカが読取りを開始するソースバイナリログ座標を確立します:

```
CHANGE MASTER TO
MASTER_HOST='source2.example.com',
MASTER_USER='replication',
MASTER_PASSWORD='password',
MASTER_PORT=3306,
MASTER_LOG_FILE='source2-bin.001',
MASTER_LOG_POS=4,
MASTER_CONNECT_RETRY=10;
```

次の例は、使用される頻度の低い操作を示しています。これは、なんらかの理由で再実行するリレーログファイルがレプリカに存在する場合に使用されます。これを行うには、ソースにアクセスできる必要はありません。CHANGE MASTER TO のみを使用し、SQL スレッド (START REPLICA | SLAVE SQL_THREAD) を起動する必要があります:

```
CHANGE MASTER TO
RELAY_LOG_FILE='replica-relay-bin.006',
RELAY_LOG_POS=4025;
```

CHANGE MASTER TO ステートメントで指定しないオプションは、次の説明に示す場合を除き、その値を保持します。そのため、ほとんどの場合、変更されないオプションを指定する必要はありません。

ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS オプションは、チャンネルが GTID を持たないレプリケートされたトランザクションに GTID を割り当て、GTID ベースのレプリケーションを使用しないソースから使用するレプリカへのレプリケーションを有効にします。マルチソースレプリカの場合、ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS を使用するチャンネルと使用しないチャンネルを混在させることができます。デフォルトは OFF で、この機能は使用されません。

LOCAL は、レプリカ独自の UUID (server_uuid 設定) を含む GTID を割り当てます。uuid は、レプリケーションソースサーバーの server_uuid 設定など、指定された UUID を含む GTID を割り当てます。非ローカル UUID を使用すると、レプリカで発生したトランザクションと、ソースで発生したトランザクション、およびマルチソースレプリカの場合は異なるソースで発生したトランザクションを区別できます。選択した UUID は、レプリカ自体での使用にのみ意味があります。ソースによって送信されたトランザクションのいずれかに GTID がすでにある場合、その GTID は保持されます。

Group Replication に固有のチャンネルでは

ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS を使用できませんが、Group Replication グループメンバーであるサーバーインスタンス上の別のソースの非同期レプリケーションチャンネルでは使用できます。その場合、GTID を作成するための UUID として Group Replication グループ名を指定しないでください。

ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS を LOCAL または uuid に設定するには、レプリカに gtid_mode=ON が設定されている必要があり、後で変更することはできません。このオプションは、バイナリログファイルの位置ベースのレプリケーションを持つソースで使用されるため、チャンネルに MASTER_AUTO_POSITION=1 を設定することはできません。このオプションを設定する前に、レプリケーション SQL スレッドとレプリケーション I/O スレッドの両方を停止する必要があります。

重要

フェイルオーバーが必要な場合、どのチャンネルでも ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS を使用して設定されたレプリカを昇格させてレプリケーションサーバーを置き換えることはできず、レプリカから作成されたバックアップを使用してレプリケーションサーバーをリストアすることはできません。同じ制限が、任意のチャンネルで ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS を

使用する他のレプリカの置換またはリストアにも適用されます。

その他の制限および詳細は、[セクション17.1.3.6「GTIDのないソースから GTIDのあるレプリカへのレプリケーション」](#)を参照してください。

GET_MASTER_PUBLIC_KEY

ソースから公開キーをリクエストすることで、RSA キーペアベースのパスワード交換を有効にします。このオプションは、`cached_sha2_password` 認証プラグインで認証されるレプリカに適用されます。このプラグインを使用して認証されるアカウントによる接続の場合、ソースはリクエストされないが公開キーを送信しないため、クライアントでリクエストまたは指定する必要があります。MASTER_PUBLIC_KEY_PATH が指定され、有効な公開キーファイルが指定されている場合は、GET_MASTER_PUBLIC_KEY よりも優先されます。cached_sha2_password プラグイン (MySQL 8.0 のデフォルト) で認証されるレプリケーションユーザーアカウントを使用していて、セキュアな接続を使用していない場合は、このオプションまたは MASTER_PUBLIC_KEY_PATH オプションを指定して、RSA 公開キーをレプリカに提供する必要があります。

IGNORE_SERVER_IDS

レプリカが指定されたサーバーから発生したイベントを無視するようにします。このオプションは、0 個以上のサーバー ID のコンマ区切りリストを取ります。サーバーからのログローテーションおよび削除イベントは無視されず、リレーログに記録されます。

循環レプリケーションでは、発信元のサーバーは通常、独自のイベントのターミネータとして機能するため、これらのイベントが複数回適用されることはありません。そのため、このオプションは、循環内のいずれかのサーバーが削除されたときの循環レプリケーションで役立ちます。1、2、3、および 4 のサーバー ID を持つ 4 台のサーバーを含む循環レプリケーションセットアップが存在するとき、サーバー 3 に障害が発生したとします。サーバー 2 からサーバー 4 へのレプリケーションを開始してギャップを埋める場合、サーバー 4 で発行する CHANGE MASTER TO ステートメントに IGNORE_SERVER_IDS = (3) を含めて、サーバー 3 の代わりにサーバー 2 をソースとして使用するよう指示できます。それにより、サーバー 4 は、使用されなくなっているサーバーで発信されたすべてのステートメントを無視し、伝播しなくなります。

IGNORE_SERVER_IDS にサーバーの独自の ID が含まれているときに、`--replicate-same-server-id` オプションが有効な状態でそのサーバーが起動された場合は、エラーが発生します。

注記

レプリケーションにグローバルトランザクション識別子 (GTID) が使用されている場合、すでに適用されているトランザクションは自動的に無視されるため、IGNORE_SERVER_IDS 関数は必須ではなく、非推奨です。サーバーに `gtid_mode=ON` が設定されている場合、CHANGE MASTER TO ステートメントに IGNORE_SERVER_IDS オプションを含めると、非推奨の警告が発行されます。

ソースメタデータリポジトリおよび SHOW REPLICA | SLAVE STATUS の出力では、現在無視されているサーバーのリストが提供されます。詳細は、[セクション17.2.4.2「レプリケーションメタデータリポジトリ」](#)および[セクション13.7.7.35「SHOW REPLICA | SLAVE STATUS ステートメント」](#)を参照してください。

IGNORE_SERVER_IDS オプションを指定せずに CHANGE MASTER TO ステートメントを発行した場合、既存のリストは保持されます。無視されるサーバーのリストをクリアするには、このオプションを空のリストとともに使用する必要があります。

```
CHANGE MASTER TO IGNORE_SERVER_IDS = ();
```

`RESET REPLICA | SLAVE ALL` により、`IGNORE_SERVER_IDS` がクリアされます。

注記

`IGNORE_SERVER_IDS` で既存のサーバー ID が設定されているチャンネルがある場合、`SET GTID_MODE=ON` が発行されると、非推奨の警告が発行されます。GTID ベースのレプリケーションを開始する前に、関係するサーバー上の無視されたすべてのサーバー ID リストを確認してクリアします。無視された ID がある場合は、`SHOW REPLICA | SLAVE STATUS` ステートメントにリストが表示されます。非推奨の警告が表示された場合でも、空のリストを指定した `IGNORE_SERVER_IDS` オプションを含む `CHANGE MASTER TO` ステートメントを発行することで、`gtid_mode=ON` の設定後にリストをクリアできます。

MASTER_AUTO_POSITION

GTID ベースのレプリケーションを使用してレプリカがソースに接続しようとしています。このオプションは、レプリケーション SQL スレッドとレプリケーション I/O スレッドの両方が停止している場合にのみ、`CHANGE MASTER TO` で使用できません。

レプリカとソースの両方で GTID が有効になっている必要があります (レプリカでは `GTID_MODE=ON`、`ON_PERMISSIVE`、または `OFF_PERMISSIVE`、ソースでは `GTID_MODE=ON`)。接続には自動配置が使用されるため、`MASTER_LOG_FILE` および `MASTER_LOG_POS` で表される座標は使用されず、これらのオプションのいずれかまたは両方を `MASTER_AUTO_POSITION = 1` とともに使用するとエラーが発生します。レプリカでマルチソースレプリケーションが有効になっている場合は、適用可能なレプリケーションチャンネルごとに `MASTER_AUTO_POSITION = 1` オプションを設定する必要があります。

`MASTER_AUTO_POSITION = 1` が設定されている場合、レプリカは初期接続ハンドシェイクで、すでに受信、コミット、またはその両方を行ったトランザクションを含む GTID セットを送信します。ソースは、GTID がレプリカによって送信される GTID セットに含まれていないバイナリログに記録されたすべてのトランザクションを送信することによって応答します。この交換により、レプリカがまだ記録またはコミットしていない GTID を持つトランザクションのみがソースから送信されるようになります。ダイヤモンドトポロジの場合と同様に、レプリカが複数のソースからトランザクションを受信する場合、自動スキップ機能によってトランザクションが 2 回適用されないようにします。レプリカによって送信される GTID セットの計算方法の詳細は、[セクション 17.1.3.3 「GTID 自動配置」](#) を参照してください。

ソースによって送信されるべきトランザクションのいずれかがソースバイナリログからパージされているか、別の方法で `gtid_purged` システム変数の GTID セットに追加されている場合、ソースはエラー `ER_MASTER_HAS_PURGED_REQUIRED_GTIDS` をレプリカに送信し、レプリケーションは開始しません。欠落しているパージ済トランザクションの GTID が識別され、警告メッセージ `ER_FOUND_MISSING_GTIDS` のソースエラーログにリストされます。また、トランザクションの交換中に、レプリカが GTID 内のソース UUID を持つトランザクションを記録またはコミットしたが、ソース自体がそれらをコミットしていないことが判明した場合、ソースはレプリカにエラー `ER_SLAVE_HAS_MORE_GTIDS_THAN_MASTER` を送信し、レプリケーションは開始しません。これらの状況の処理方法の詳細は、[セクション 17.1.3.3 「GTID 自動配置」](#) を参照してください。

レプリケーションが GTID 自動配置を有効にして実行されているかどうかを確認するには、パフォーマンススキーマの `replication_connection_status` テーブルまたは `SHOW REPLICA | SLAVE STATUS` の出力を確認します。

MASTER_AUTO_POSITION オプションを再度無効にすると、レプリカはファイルベースレプリケーションに戻ります。その場合は、**MASTER_LOG_FILE** または **MASTER_LOG_POS** オプションのいずれかまたは両方も指定する必要があります。

MASTER_BIND

複数のネットワークインタフェースを持つレプリカで使用するために、ソースへの接続に選択するレプリカネットワークインタフェースを決定します。このオプションで構成されたアドレスは、**SHOW REPLICA | SLAVE STATUS** からの出力の **Master_Bind** カラムに表示されます (存在する場合)。ソースメタデータリポジトリテーブル **mysql.slave_master_info** では、値は **Master_bind** カラムとして表示されます。レプリカを特定のネットワークインタフェースにバインドする機能は、NDB Cluster でもサポートされています。

MASTER_COMPRESSION_ALGORITHM

レプリケーションソースサーバーへの接続に許可される圧縮アルゴリズムを指定します。使用可能なアルゴリズムは、**protocol_compression_algorithms** システム変数の場合と同じです。デフォルト値は **uncompressed** です。**MASTER_COMPRESSION_ALGORITHMS** は、MySQL 8.0.18 の時点で使用可能です。

MASTER_COMPRESSION_ALGORITHMS の値は、**slave_compressed_protocol** システム変数が無効な場合にのみ適用されます。**slave_compressed_protocol** が有効な場合は、**MASTER_COMPRESSION_ALGORITHMS** よりも優先され、ソースとレプリカの両方がそのアルゴリズムをサポートしている場合は、ソースへの接続で **zlib** 圧縮が使用されます。詳細は、[セクション4.2.8「接続圧縮制御」](#)を参照してください。

binlog_transaction_compression システム変数によってアクティブ化されるバイナリログトランザクション圧縮 (MySQL 8.0.20 で使用可能) を使用して帯域幅を節約することもできます。これを接続圧縮と組み合わせて行くと、接続圧縮ではデータを処理する機会は少なくなります。ヘッダーと、圧縮されていないイベントおよびトランザクションペイロードは圧縮できます。バイナリログのトランザクション圧縮の詳細は、[セクション5.4.4.5「バイナリログトランザクション圧縮」](#)を参照してください。

MASTER_CONNECT_RETRY

ソースへの接続がタイムアウトした後にレプリカが試行する再接続の間隔を指定します。試行は、**MASTER_RETRY_COUNT** オプションによって制限されます。両方のデフォルト設定が使用されている場合、レプリカは再接続試行 (**MASTER_CONNECT_RETRY=60**) の間 60 秒待機し、60 日間 (**MASTER_RETRY_COUNT=86400**) 再接続を試行し続けます。これらの値はソースメタデータリポジトリに記録され、**replication_connection_configuration** 「パフォーマンススキーマ」テーブルに表示されます。

MASTER_DELAY

レプリカが遅延する必要があるソースからの遅延秒数を指定します。ソースから受信したイベントは、ソースでの実行より少なくとも **interval** 秒後に実行されません。デフォルトは 0 です。**interval** が 0 から $2^{31}-1$ までの範囲の負ではない整数でない場合は、エラーが発生します。詳細は、[セクション17.4.11「遅延レプリケーション」](#)を参照してください。**MASTER_DELAY** オプションを使用する **CHANGE MASTER TO** ステートメントは、レプリケーション SQL スレッドの停止時に実行中のレプリカで実行できます。

MASTER_HEARTBEAT_PERIOD

ハートビート間隔を制御します。これにより、接続がまだ良好な場合に、データが存在しないときに接続タイムアウトが発生しなくなります。ハートビートシグナルは、その秒数が経過するとレプリカに送信され、ソースバイナリログがイベントで更新されるたびに待機期間がリセットされます。したがって、ハートビートは、これより長い期間バイナリログファイルに未送信のイベントがない場合にのみ、ソースによって送信されます。

ハートビート間隔 **interval** は、0 から 4294967 秒の範囲の小数値およびミリ秒単位の解像度です。ゼロ以外の最小値は 0.001 です。**interval** を 0 に設定すると、ハートビートが完全に無効になります。ハートビート間隔のデフォルトは、**slave_net_timeout** システム変数の値の半分です。ソースメタデータリポジトリに記録され、**replication_connection_configuration** 「パフォーマンススキーマ」

テーブルに表示されます。 `RESET REPLICA | SLAVE` を発行すると、ハートビート間隔がデフォルト値にリセットされます。

`slave_net_timeout` システム変数は、レプリカがソースからのデータまたはハートビート信号のいずれかを待機する秒数を指定します。この秒数を過ぎると、レプリカは接続が切断されたときとみなし、読取りを中断して再接続を試行します。デフォルト値は 60 秒 (1 分) です。 `slave_net_timeout` の値またはデフォルト設定を変更しても、明示的に設定されているか、以前に計算されたデフォルトを使用しているかにかかわらず、ハートビート間隔は自動的に変更されないことに注意してください。 `@@GLOBAL.slave_net_timeout` を現在のハートビート間隔より小さい値に設定すると、警告が発行されます。 `slave_net_timeout` が変更された場合は、接続タイムアウトの前にハートビート信号が発生するように、 `CHANGE MASTER TO` を発行してハートビート間隔を適切な値に調整する必要があります。これを行わない場合、ハートビート信号は効果がなく、ソースからデータを受信しない場合、レプリカは再接続を繰り返し試行してゾンビダンプスレッドを作成できます。

MASTER_HOST

レプリケーションソースサーバーのホスト名または IP アドレス。レプリカはこれを使用してソースに接続します。

`MASTER_HOST` または `MASTER_PORT` を指定した場合、レプリカは、ソースサーバーが以前とは異なることを前提とします (オプション値が現在の値と同じであっても。) この場合、ソースバイナリログファイルの名前と位置の古い値は適用できなくなるため、ステートメントに `MASTER_LOG_FILE` と `MASTER_LOG_POS` を指定しないと、 `MASTER_LOG_FILE=""` と `MASTER_LOG_POS=4` は暗黙的に追加されます。

`MASTER_HOST=""` を設定する (つまり、その値を明示的に空の文字列に設定する) ことは、 `MASTER_HOST` をまったく設定しないことと同じではありません。 `MASTER_HOST` を空の文字列に設定しようとすると、エラーで失敗します。

MASTER_LOG_FILE, MASTER_LOG_POS

バイナリログファイル名、およびレプリケーション I/O スレッドが次のスレッド起動時にソースバイナリログからの読取りを開始するそのファイル内の場所。バイナリログファイルの位置ベースのレプリケーションを使用している場合は、これらのオプションを指定します。 `MASTER_LOG_FILE` には、ソースサーバーで使用可能な特定のバイナリログファイル (`MASTER_LOG_FILE='binlog.000145'` など) の数値接尾辞が含まれている必要があります。 `MASTER_LOG_POS` は、レプリカがそのファイルの読取りを開始する数値位置です。 `MASTER_LOG_POS=4` は、バイナリログファイルでイベントの開始を表します。

`MASTER_LOG_FILE` または `MASTER_LOG_POS` のどちらかを指定する場合は、 `RELAY_LOG_FILE` や `RELAY_LOG_POS` を指定できません。 `MASTER_LOG_FILE` または `MASTER_LOG_POS` のいずれかを指定する場合は、GTID ベースのレプリケーション用の `MASTER_AUTO_POSITION = 1` も指定できません。

`MASTER_LOG_FILE` も `MASTER_LOG_POS` も指定されていない場合、レプリカは `CHANGE MASTER TO` が発行される前のレプリケーション SQL スレッドの最後の座標を使用します。これにより、レプリケーション SQL スレッドがレプリケーション I/O スレッドと比較して遅延していても、レプリケーションの不連続性がなくなります。

MASTER_PASSWORD

レプリケーションソースサーバーへの接続に使用するレプリケーションユーザーアカウントのパスワード。 `MASTER_PASSWORD` を指定する場合は、 `MASTER_USER` も必要です。

`CHANGE MASTER TO` ステートメントでレプリケーションユーザーアカウントに使用されるパスワードの長さは、32 文字に制限されています。32 文字を超えるパスワードを使用しようとすると、 `CHANGE MASTER TO` で障害が発生します。

MASTER_PORT

レプリカがレプリケーションソースサーバーへの接続に使用する TCP/IP ポート番号。

注記

レプリケーションでは、Unix ソケットファイルを使用できません。TCP/IP を使用してレプリケーションソースサーバーに接続できる必要があります。

MASTER_HOST または MASTER_PORT を指定した場合、レプリカは、ソースサーバーが以前とは異なることを前提とします (オプション値が現在の値と同じであっても)。) この場合、ソースバイナリログファイルの名前と位置の古い値は適用できなくなるため、ステートメントに MASTER_LOG_FILE と MASTER_LOG_POS を指定しないと、MASTER_LOG_FILE="" と MASTER_LOG_POS=4 は暗黙的に追加されます。

- MASTER_PUBLIC_KEY_PATH ソースが必要とする公開キーのレプリカ側のコピーを含むファイルへのパス名を指定することで、RSA キーペアベースのパスワード交換を有効にします。ファイルは PEM 形式である必要があります。このオプションは、sha256_password または caching_sha2_password 認証プラグインで認証されるレプリカに適用されます。(sha256_password の場合、MASTER_PUBLIC_KEY_PATH は、MySQL が OpenSSL を使用して構築された場合にのみ使用できます。) caching_sha2_password プラグイン (MySQL 8.0 のデフォルト) で認証されるレプリケーションユーザーアカウントを使用して、セキュアな接続を使用していない場合は、このオプションまたは GET_MASTER_PUBLIC_KEY=1 オプションを指定して、RSA 公開キーをレプリカに提供する必要があります。
- MASTER_RETRY_COUNT slave_net_timeout システム変数によって決定される、ソースへの接続がタイムアウトした後にレプリカが行う再接続の最大試行回数を設定します。レプリカを再接続する必要がある場合、タイムアウトの直後に最初の再試行が行われます。試行の間隔は、MASTER_CONNECT_RETRY オプションで指定します。両方のデフォルト設定が使用されている場合、レプリカは再接続試行 (MASTER_CONNECT_RETRY=60) の間 60 秒待機し、60 日間 (MASTER_RETRY_COUNT=86400) 再接続を試行し続けます。これらの値はソースメタデータリポジトリに記録され、replication_connection_configuration 「パフォーマンススキーマ」テーブルに表示されます。MASTER_RETRY_COUNT は、--master-retry-count サーバーの起動オプションより優先されます。
- MASTER_SSL_XXX, MASTER_TLS_XXX レプリカが暗号化および暗号化を使用してレプリケーション接続を保護する方法を指定します。これらのオプションは、SSL サポートなしでコンパイルされたレプリカでも変更できます。これらはソースメタデータリポジトリに保存されますが、レプリカで SSL サポートが有効になっていない場合は無視されます。MASTER_SSL_XXX および MASTER_TLS_XXX オプションは、暗号化接続のコマンドオプションで説明されている --ssl-xxx および --tls-xxx クライアントオプションと同じ機能を実行します。2 つのオプションセット間の対応、および MASTER_SSL_XXX オプションと MASTER_TLS_XXX オプションを使用したセキュアな接続の設定については、セクション 17.3.1 「暗号化接続を使用するためのレプリケーションの設定」を参照してください。
- MASTER_USER レプリケーションソースサーバーへの接続に使用するレプリケーションユーザーアカウントのユーザー名。

重要

caching_sha2_password プラグインで認証するレプリケーションユーザーアカウントを使用してソースに接続するには、セクション 17.3.1 「暗号化接続を使用するためのレプリケーションの設定」の説明に従ってセキュアな接続を設定するか、RSA キーペアを使用したパスワード交換をサポートするように暗号化されていない接続を有効にする必要があります。caching_sha2_password 認証プラグインは、MySQL 8.0 から作成された新規ユーザーのデフォルトです (詳細は、セクション 6.4.1.2 「SHA-2 P

「[ラガブル認証のキャッシュ](#)」を参照)。レプリケーション用に作成または使用するユーザーアカウントがこの認証プラグインを使用しており、セキュアな接続を使用していない場合は、正常に接続するために RSA キーペアベースのパスワード交換を有効にする必要があります。これは、`MASTER_PUBLIC_KEY_PATH` オプションまたはこのステートメントの `GET_MASTER_PUBLIC_KEY=1` オプションを使用して実行できます。

`MASTER_USER=""`を指定して空のユーザー名を設定することはできますが、レプリケーションチャンネルは空のユーザー名で開始できません。MySQL 8.0.21 より前のリリースでは、セキュリティ上の目的で以前に使用した資格証明をレプリケーションメタデータリポジトリからクリアする必要がある場合のみ、空の `MASTER_USER` ユーザー名を設定します。これらのリリースでは、リポジトリから空のユーザー名が読み取られた場合 (グループレプリケーションチャンネルの自動再起動時など)、デフォルトのユーザー名を置換できるバグのため、後でチャンネルを使用しないでください。MySQL 8.0.21 からは、レプリケーションチャンネルを開始する `START REPLICA | SLAVE` ステートメントまたは `START GROUP_REPLICATION` ステートメントを使用して常にユーザー資格証明を指定する場合、空の `MASTER_USER` ユーザー名を設定し、後でチャンネルを使用できます。このアプローチでは、レプリケーションチャンネルを再起動するためにオペレータの介入が常に必要ですが、ユーザー資格証明はレプリケーションメタデータリポジトリに記録されません。

実行中の `CHANGE MASTER TO` ステートメントのテキスト (`MASTER_USER` と `MASTER_PASSWORD` の値を含む) は、並列 `SHOW PROCESSLIST` ステートメントの出力で確認できます。(`START REPLICA | SLAVE` ステートメントの完全なテキストは、`SHOW PROCESSLIST` にも表示されます。)

`MASTER_ZSTD_COMPRESSION_LEVEL` は、圧縮アルゴリズムを使用するレプリケーションソースサーバーへの接続に使用する圧縮レベル。許可されるレベルは 1 から 22 で、大きい値は圧縮レベルの増加を示します。デフォルトの `zstd` 圧縮レベルは 3 です。圧縮レベルの設定は、`zstd` 圧縮を使用しない接続には影響しません。`MASTER_ZSTD_COMPRESSION_LEVEL` は、MySQL 8.0.18 の時点で使用可能です。詳細は、[セクション4.2.8「接続圧縮制御」](#)を参照してください。

`NETWORK_NAMESPACE` レプリケーションソースサーバーへの TCP/IP 接続に使用するネットワーク名前空間。このオプションを省略すると、レプリカからの接続にはデフォルト (グローバル) 名前空間が使用されます。ネットワーク名前空間のサポートを実装していないプラットフォームでは、レプリカがソースに接続しようとするとうつらが発生します。ネットワーク名前空間の詳細は、[セクション5.1.14「ネットワーク名前空間のサポート」](#)を参照してください。`NETWORK_NAMESPACE` は、MySQL 8.0.22 の時点で使用可能です。

`PRIVILEGE_CHECKS_USER` 指定されたチャンネルのセキュリティコンテキストを提供するユーザーアカウントを指定します。`NULL` (デフォルト) は、セキュリティコンテキストが使用されないことを意味します。`PRIVILEGE_CHECKS_USER` は、MySQL 8.0.18 の時点で使用可能です。

ユーザーアカウントのユーザー名とホスト名は、[セクション6.2.4「アカウント名の指定」](#)で説明されている構文に従う必要があり、ユーザーは匿名ユーザー (ユーザー名が空白) または `CURRENT_USER` であってはなりません。アカウントには、`REPLICATION_APPLIER` 権限と、チャンネルでレプリケートされたトランザクションを実行するために必要な権限が必要です。アカウントに必要な権限の詳細は、[セクション17.3.3「レプリケーション権限チェック」](#)を参照してください。レプリケーションチャンネルを再起動すると、その時点から権限チェックが適用されます。チャンネルを指定せず、他のチャンネルが存在しない場合は、ステートメントがデフォルトチャンネルに適用されます。

`PRIVILEGE_CHECKS_USER` が設定されており、これを強制するように `REQUIRE_ROW_FORMAT` を設定できる場合は、行ベースのバイナリロギング

を使用することを強くお勧めします。たとえば、実行中のレプリカでチャンネル `channel_1` に対する権限チェックを開始するには、次のステートメントを発行します:

```
mysql> STOP REPLICA | SLAVE FOR CHANNEL 'channel_1';
mysql> CHANGE MASTER TO
  PRIVILEGE_CHECKS_USER = 'priv_rep!'@'%example.com',
  REQUIRE_ROW_FORMAT = 1,
  FOR CHANNEL 'channel_1';
mysql> START REPLICA | SLAVE FOR CHANNEL 'channel_1';
```

RELAY_LOG_FILE, RELAY_LOG_POS

次のスレッド起動時にレプリケーション SQL スレッドがレプリカリレーログからの読取りを開始するリレーログファイル名およびそのファイル内の場所。 `MASTER_LOG_FILE` または `MASTER_LOG_POS` を使用している場合、これらのオプションは使用できません。

`RELAY_LOG_FILE` では、絶対パスまたは相対パスのいずれかを
使用でき、`MASTER_LOG_FILE` と同じベース名を使用します。
`RELAY_LOG_FILE`、`RELAY_LOG_POS` またはその両方のオプションを使用する `CHANGE MASTER TO` ステートメントは、レプリケーション SQL スレッドの停止時に実行中のレプリカで実行できます。少なくともいずれかのレプリケーション SQL スレッドとレプリケーション I/O スレッドが実行されている場合、リレーログは保持されます。両方のスレッドが停止すると、`RELAY_LOG_FILE` または `RELAY_LOG_POS` のいずれかが指定されていないかぎり、すべてのリレーログファイルが削除されます。Group Replication applier チャンネル (`group_replication_applier`) には I/O スレッドがなく、SQL スレッドのみがあることに注意してください。このチャンネルでは、SQL スレッドの停止時にリレーログは保持されません。

REQUIRE_ROW_FORMAT

レプリケーションチャンネルによる行ベースのレプリケーションイベントの処理のみを許可します。このオプションにより、レプリケーションアプライアンスは一時テーブルの作成や `LOAD DATA INFILE` リクエストの実行などのアクションを実行できなくなり、チャンネルのセキュリティが向上します。グループレプリケーションチャンネルは `REQUIRE_ROW_FORMAT` セットで自動的に作成され、これらのチャンネルのオプションは変更できません。詳細は、[セクション17.3.3「レプリケーション権限チェック」](#)を参照してください。 `REQUIRE_ROW_FORMAT` は、MySQL 8.0.19 の時点で使用可能です。

`REQUIRE_TABLE_PRIMARY_KEY_CHECK` の値が主キーチェック用に独自のポリシーを選択できるようにします。レプリケーションチャンネルのオプションが `ON` に設定されている場合、レプリカはレプリケーション操作で常に `sql_require_primary_key` システム変数に値 `ON` を使用し、主キーが必要です。このオプションが `OFF` に設定されている場合、レプリカはレプリケーション操作で `sql_require_primary_key` システム変数に常に値 `OFF` を使用するため、ソースで必要な場合でも主キーは必要ありません。 `REQUIRE_TABLE_PRIMARY_KEY_CHECK` オプションが `STREAM` (デフォルト) に設定されている場合、レプリカは各トランザクションのソースからレプリケートされた値を使用します。 `REQUIRE_TABLE_PRIMARY_KEY_CHECK` は、MySQL 8.0.20 の時点で使用可能です。

マルチソースレプリケーションの場合

`REQUIRE_TABLE_PRIMARY_KEY_CHECK` を `ON` または `OFF` に設定すると、レプリカは異なるソースのレプリケーションチャンネル間で動作を正規化し、`sql_require_primary_key` システム変数の一貫性のある設定を維持できます。 `ON` を使用すると、複数のソースが同じテーブルセットを更新する場合に、主キーの偶発的な損失から保護されます。 `OFF` を使用すると、主キーを操作できるソースを、操作できないソースと連携させることができます。

`PRIVILEGE_CHECKS_USER` が設定されている場合

`REQUIRE_TABLE_PRIMARY_KEY_CHECK` を `ON` または `OFF` に設定することは、制限付きセッション変数を設定するためのセッション管理レベルの権限がユーザーアカウントに必要なことを意味します。これは、`sql_require_primary_key` の値を各トランザクションのソース設定と一致するよ

うに変更するために必要です。詳細は、[セクション17.3.3「レプリケーション権限チェック」](#)を参照してください。

`SOURCE_CONNECTION_AUTO_FAILOVER` 以上の代替レプリケーションサーバーが使用可能な場合 (レプリケートされたデータを共有する複数の MySQL サーバーまたはサーバーグループが存在する場合)、レプリケーションチャンネルの非同期接続フェイルオーバーメカニズムをアクティブ化します。 `SOURCE_CONNECTION_AUTO_FAILOVER` は、MySQL 8.0.22 の時点で使用可能です。非同期接続フェイルオーバーメカニズムは、`MASTER_CONNECT_RETRY` および `MASTER_RETRY_COUNT` によって制御されている再接続試行を使い果たした後に引き継ぎます。 `asynchronous_connection_failover_add_source` および `asynchronous_connection_failover_delete_source` UDF を使用して管理する、指定されたソースリストから選択された代替ソースにレプリカを再接続します。サーバーの管理対象グループを追加および削除するには、かわりに `asynchronous_connection_failover_add_managed` および `asynchronous_connection_failover_delete_managed` UDF を使用します。詳細は、[セクション17.4.9「非同期接続フェイルオーバーによるソースの切替え」](#)を参照してください。

重要

1. GTID 自動配置が使用中 (`MASTER_AUTO_POSITION = 1`) の場合にのみ、`SOURCE_CONNECTION_AUTO_FAILOVER = 1` を設定できます。
2. `SOURCE_CONNECTION_AUTO_FAILOVER = 1` を設定する場合、一時的なネットワークの停止が原因で接続障害が発生した場合に、`MASTER_RETRY_COUNT` および `MASTER_CONNECT_RETRY` を、同じソースでの数回の再試行のみを許可する最小数に設定します。そうしないと、非同期接続フェイルオーバーメカニズムをすぐにアクティブ化できません。適切な値は `MASTER_RETRY_COUNT=3` と `MASTER_CONNECT_RETRY=10` です。これにより、レプリカは 10 秒間隔で接続を 3 回再試行します。
3. `SOURCE_CONNECTION_AUTO_FAILOVER = 1` を設定する場合、レプリケーションメタデータリポジトリには、レプリケーションチャンネルのソースリスト上のすべてのサーバーへの接続に使用できるレプリケーションユーザーアカウントの資格証明が含まれている必要があります。これらの資格証明は、`MASTER_USER` および `MASTER_PASSWORD` オプションを指定した `CHANGE REPLICATION SOURCE TO` ステートメントを使用して設定できます。詳細は、[セクション17.4.9「非同期接続フェイルオーバーによるソースの切替え」](#)を参照してください。

次の表は、文字列値のオプションに許可される最大長を示しています。

オプション	最大長
<code>MASTER_HOST</code>	255 (MySQL 8.0.17 より前の 60)
<code>MASTER_USER</code>	96
<code>MASTER_PASSWORD</code>	32
<code>MASTER_LOG_FILE</code>	511
<code>RELAY_LOG_FILE</code>	511

オプション	最大長
MASTER_SSL_CA	511
MASTER_SSL_CAPATH	511
MASTER_SSL_CERT	511
MASTER_SSL_CRL	511
MASTER_SSL_CRLPATH	511
MASTER_SSL_KEY	511
MASTER_SSL_CIPHER	511
MASTER_TLS_VERSION	511
MASTER_TLS_CIPHERSUITES	4000
MASTER_PUBLIC_KEY_PATH	511
MASTER_COMPRESSION_ALGORITHMS	99
NETWORK_NAMESPACE	64

13.4.2.2 CHANGE REPLICATION FILTER ステートメント

```
CHANGE REPLICATION FILTER filter[, filter]
[, ...] [FOR CHANNEL channel]

filter: {
  REPLICATE_DO_DB = (db_list)
| REPLICATE_IGNORE_DB = (db_list)
| REPLICATE_DO_TABLE = (tbl_list)
| REPLICATE_IGNORE_TABLE = (tbl_list)
| REPLICATE_WILD_DO_TABLE = (wild_tbl_list)
| REPLICATE_WILD_IGNORE_TABLE = (wild_tbl_list)
| REPLICATE_REWRITE_DB = (db_pair_list)
}

db_list:
  db_name[, db_name][, ...]

tbl_list:
  db_name.table_name[, db_name.table_name][, ...]

wild_tbl_list:
  'db_pattern.table_pattern', 'db_pattern.table_pattern'[, ...]

db_pair_list:
  (db_pair)[, (db_pair)][, ...]

db_pair:
  from_db, to_db
```

`CHANGE REPLICATION FILTER` は、`--replicate-do-db` や `--replicate-wild-ignore-table` などのレプリケーションフィルタリングオプションを使用してレプリカ `mysqld` を起動するのと同じ方法で、レプリカに 1 つ以上のレプリケーションフィルタリングルールを設定します。サーバーオプションとは異なり、このステートメントを有効にするためにサーバーを再起動する必要はなく、まず `STOP REPLICA | SLAVE SQL_THREAD` を使用してレプリケーション SQL スレッドを停止します (その後、`START REPLICA | SLAVE SQL_THREAD` を使用して再起動します)。`CHANGE REPLICATION FILTER` には、`REPLICATION_SLAVE_ADMIN` 権限 (または非推奨の `SUPER` 権限) が必要です。 `FOR CHANNEL channel` 句を使用すると、レプリケーションフィルタをマルチソースレプリカなどのレプリケーションチャンネルに固有にできます。特定の `FOR CHANNEL` 句なしで適用されたフィルタはグローバルフィルタとみなされ、すべてのレプリケーションチャンネルに適用されます。

注記

グループレプリケーション用に構成された MySQL サーバーインスタンスでは、グローバルレプリケーションフィルタを設定できません。これは、一部のサーバーでトランザクションをフィルタすると、グループが一貫性のある状態で承諾に到達できなくなるためです。グループメンバーがグループ外のソースへのレプリカとしても機能する場合など、グループレプリケーションに直接関与しないレプリケーションチャンネルにチャネ

ル固有のレプリケーションフィルタを設定できます。 `group_replication_applier` または `group_replication_recovery` チャンネルでは設定できません。

次のリストに、`CHANGE REPLICATION FILTER` のオプションと、それらが `--replicate-*` サーバーのオプションとどのように関連しているかを示します:

- `REPLICATE_DO_DB`: データベース名に基づいた更新を含めます。 `--replicate-do-db` と同等です。
- `REPLICATE_IGNORE_DB`: データベース名に基づいて更新を除外します。 `--replicate-ignore-db` と同等です。
- `REPLICATE_DO_TABLE`: テーブル名に基づく更新を含めます。 `--replicate-do-table` と同等です。
- `REPLICATE_IGNORE_TABLE`: テーブル名に基づいて更新を除外します。 `--replicate-ignore-table` と同等です。
- `REPLICATE_WILD_DO_TABLE`: ワイルドカードパターンマッチングテーブル名に基づく更新を含めます。 `--replicate-wild-do-table` と同等です。
- `REPLICATE_WILD_IGNORE_TABLE`: ワイルドカードパターンマッチングテーブル名に基づいて更新を除外します。 `--replicate-wild-ignore-table` と同等です。
- `REPLICATE_REWRITE_DB`: レプリカ上の新しい名前をソース上の指定されたデータベースに置き換えた後、レプリカで更新を実行します。 `--replicate-rewrite-db` と同等です。

`REPLICATE_DO_DB` および `REPLICATE_IGNORE_DB` フィルタの正確な影響は、ステートメントベースレプリケーションと行ベースレプリケーションのどちらが有効かによって異なります。詳しくは [セクション 17.2.5 「サーバーがレプリケーションフィルタリングルールをどのように評価するか」](#) をご覧ください。

次に示すように、ルールをカンマで区切ることで、単一の `CHANGE REPLICATION FILTER` ステートメントに複数のレプリケーションフィルタリングルールを作成できます:

```
CHANGE REPLICATION FILTER
  REPLICATE_DO_DB = (d1), REPLICATE_IGNORE_DB = (d2);
```

前述のステートメントを発行することは、`--replicate-do-db=d1 --replicate-ignore-db=d2` オプションを指定してレプリカ `mysqld` を起動することと同じです。

複数のレプリケーションチャンネルを使用して異なるソースからのトランザクションを処理するマルチソースレプリカでは、`FOR CHANNEL channel` 句を使用してレプリケーションチャンネルにレプリケーションフィルタを設定します:

```
CHANGE REPLICATION FILTER REPLICATE_DO_DB = (d1) FOR CHANNEL channel_1;
```

これにより、チャンネル固有のレプリケーションフィルタを作成して、選択したデータをソースから除外できます。`FOR CHANNEL` 句が指定されている場合、レプリケーションフィルタステートメントはそのレプリケーションチャンネルで動作し、指定されたレプリケーションフィルタと同じフィルタタイプを持つ既存のレプリケーションフィルタを削除して、指定されたフィルタに置き換えます。ステートメントに明示的にリストされていないフィルタタイプは変更されません。構成されていないレプリケーションチャンネルに対して発行された場合、ステートメントは `ER_SLAVE_CONFIGURATION` エラーで失敗します。グループレプリケーションチャンネルに対して発行すると、ステートメントは `ER_SLAVE_CHANNEL_OPERATION_NOT_ALLOWED` エラーで失敗します。

複数のレプリケーションチャンネルが構成されているレプリカでは、`FOR CHANNEL` 句を指定せずに `CHANGE REPLICATION FILTER` を発行すると、構成されているすべてのレプリケーションチャンネルおよびグローバルレプリケーションフィルタに対してレプリケーションフィルタが構成されます。すべてのフィルタタイプについて、フィルタタイプがステートメントにリストされている場合、そのタイプの既存のフィルタルールは、最後に発行されたステートメントで指定されたフィルタルールに置き換えられます。それ以外の場合は、フィルタタイプの古い値が保持されます。詳細は、[セクション 17.2.5.4 「レプリケーションチャンネルベースのフィルタ」](#) を参照してください。

同じフィルタリングルールが複数回指定されている場合、実際には last のそのようなルールのみが使用されます。たとえば、最初のステートメントの最初の `REPLICATE_DO_DB` ルールは無視されるため、ここに示す 2 つのステートメントはまったく同じ効果があります:

```
CHANGE REPLICATION FILTER
  REPLICATE_DO_DB = (db1, db2), REPLICATE_DO_DB = (db3, db4);
```

```
CHANGE REPLICATION FILTER
  REPLICATE_DO_DB = (db3, db4);
```


注意

この動作は、同じオプションを複数回指定すると複数のフィルタルールが作成される `--replicate-*` フィルタオプションとは異なります。

特殊文字を含まないテーブルおよびデータベースの名前は引用符で囲む必要はありません。

`REPLICATION_WILD_TABLE` および `REPLICATION_WILD_IGNORE_TABLE` で使用される値は文字列式で、(特殊な)ワイルドカード文字が含まれている可能性があるため、引用符で囲む必要があります。これを次のステートメントの例に示します:

```
CHANGE REPLICATION FILTER
  REPLICATE_WILD_DO_TABLE = ('db1.old%');

CHANGE REPLICATION FILTER
  REPLICATE_WILD_IGNORE_TABLE = ('db1.new%', 'db2.new%');
```

`REPLICATE_REWRITE_DB` で使用される値は、データベース名のペアを表します。このような値はそれぞれカッコで囲む必要があります。次のステートメントは、ソースのデータベース `db1` で発生したステートメントをレプリカのデータベース `db2` にリライトします:

```
CHANGE REPLICATION FILTER REPLICATE_REWRITE_DB = ((db1, db2));
```

前述のステートメントには、データベース名のペアを囲むカッコと、リスト全体を囲むカッコの2つのセットが含まれています。これは、データベース `dbA` を `dbB` にリライトし、データベース `dbC` を `dbD` にリライトするという2つの `rewrite-db` ルールを作成する次の例では、より簡単に見られます:

```
CHANGE REPLICATION FILTER
  REPLICATE_REWRITE_DB = ((dbA, dbB), (dbC, dbD));
```

`CHANGE REPLICATION FILTER` ステートメントでは、ステートメントの影響を受けるフィルタタイプおよびレプリケーションチャンネルのレプリケーションフィルタリングルールのみが置換され、他のルールおよびチャンネルは変更されません。特定のタイプのすべてのフィルタの設定を解除する場合は、次の例に示すように、フィルタ値を明示的に空のリストに設定します。これにより、既存のすべての `REPLICATE_DO_DB` および `REPLICATE_IGNORE_DB` ルールが削除されます:

```
CHANGE REPLICATION FILTER
  REPLICATE_DO_DB = (), REPLICATE_IGNORE_DB = ();
```

この方法でフィルタを空に設定すると、既存のすべてのルールが削除され、新しいルールは作成されず、`mysqld` の起動時にコマンド行または構成ファイルで `--replicate-*` オプションを使用して設定されたルールは復元されません。

`RESET REPLICAS | SLAVE ALL` ステートメントは、ステートメントによって削除されたチャンネルに設定されたチャンネル固有のレプリケーションフィルタを削除します。削除されたチャンネルが再作成されると、レプリカに指定されたグローバルレプリケーションフィルタがそれらにコピーされ、チャンネル固有のレプリケーションフィルタは適用されません。

詳細は、[セクション17.2.5「サーバーがレプリケーションフィルタリングルールをどのように評価するか」](#)を参照してください。

13.4.2.3 CHANGE REPLICATION SOURCE TO ステートメント

```
CHANGE REPLICATION SOURCE TO option [, option] ... [ channel_option ]

option: {
  SOURCE_BIND = 'interface_name'
| SOURCE_HOST = 'host_name'
| SOURCE_USER = 'user_name'
| SOURCE_PASSWORD = 'password'
| SOURCE_PORT = port_num
| PRIVILEGE_CHECKS_USER = {'account' | NULL}
| REQUIRE_ROW_FORMAT = {0|1}
| REQUIRE_TABLE_PRIMARY_KEY_CHECK = {STREAM | ON | OFF}
| ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS = {OFF | LOCAL | uuid}
| SOURCE_LOG_FILE = 'source_log_name'
| SOURCE_LOG_POS = source_log_pos
| SOURCE_AUTO_POSITION = {0|1}
| RELAY_LOG_FILE = 'relay_log_name'
```

```

| RELAY_LOG_POS = relay_log_pos
| SOURCE_HEARTBEAT_PERIOD = interval
| SOURCE_CONNECT_RETRY = interval
| SOURCE_RETRY_COUNT = count
| SOURCE_CONNECTION_AUTO_FAILOVER = {0|1}
| SOURCE_DELAY = interval
| SOURCE_COMPRESSION_ALGORITHMS = 'value'
| SOURCE_ZSTD_COMPRESSION_LEVEL = level
| SOURCE_SSL = {0|1}
| SOURCE_SSL_CA = 'ca_file_name'
| SOURCE_SSL_CAPATH = 'ca_directory_name'
| SOURCE_SSL_CERT = 'cert_file_name'
| SOURCE_SSL_CRL = 'crl_file_name'
| SOURCE_SSL_CRLPATH = 'crl_directory_name'
| SOURCE_SSL_KEY = 'key_file_name'
| SOURCE_SSL_CIPHER = 'cipher_list'
| SOURCE_SSL_VERIFY_SERVER_CERT = {0|1}
| SOURCE_TLS_VERSION = 'protocol_list'
| SOURCE_TLS_CIPHERSUITES = 'ciphersuite_list'
| SOURCE_PUBLIC_KEY_PATH = 'key_file_name'
| GET_SOURCE_PUBLIC_KEY = {0|1}
| NETWORK_NAMESPACE = 'namespace'
| IGNORE_SERVER_IDS = (server_id_list)
}

channel_option:
  FOR CHANNEL channel

server_id_list:
  [server_id [, server_id] ... ]

```

CHANGE REPLICATION SOURCE TO は、レプリカサーバーがソースへの接続およびソースからのデータの読取りに使用するパラメータを変更します。また、レプリケーションメタデータリポジトリの内容も更新されます ([セクション17.2.4「リレーログおよびレプリケーションメタデータリポジトリ」](#)を参照)。MySQL 8.0.23 から、**CHANGE MASTER TO** のかわりに **CHANGE REPLICATION SOURCE TO** を使用します。これは、そのリリースから非推奨になりました。MySQL 8.0.23 より前のリリースでは、**CHANGE MASTER TO** を使用します。

レプリケーション SQL スレッドおよびレプリケーション I/O スレッドの状態に応じて、最初に停止せずに、実行中のレプリカに対して **CHANGE REPLICATION SOURCE TO** ステートメントを発行できます。このような使用を制御するルールは、このセクションの後半で説明します。**CHANGE REPLICATION SOURCE TO** には、**REPLICATION_SLAVE_ADMIN** 権限 (または非推奨の **SUPER** 権限) が必要です。

マルチスレッドのレプリカを使用する場合 (つまり、**slave_parallel_workers** が 0 より大きい場合)、レプリカを停止すると、レプリカが意図的に停止されたかどうかに関係なく、リレーログから実行された一連のトランザクションで「ギャップ」が発生する可能性があります。このようなギャップが存在する場合、**CHANGE REPLICATION SOURCE TO** の発行は失敗します。この状況の解決策は、ギャップを確実に閉じる **START REPLICATION SQL_AFTER_MTS_GAPS** を発行することです。

オプションの **FOR CHANNEL channel** 句を使用すると、ステートメントが適用されるレプリケーションチャンネルの名前を指定できます。**FOR CHANNEL channel** 句を指定すると、**CHANGE REPLICATION SOURCE TO** ステートメントが特定のレプリケーションチャンネルに適用され、新しいチャンネルの追加または既存のチャンネルの変更に使用されます。たとえば、**channel2** という新しいチャンネルを追加するには、次のようにします:

```
CHANGE REPLICATION SOURCE TO SOURCE_HOST=host1, SOURCE_PORT=3002 FOR CHANNEL 'channel2'
```

句が指定されておらず、追加のチャンネルが存在しない場合、ステートメントはデフォルトチャンネルに適用されます。

複数のレプリケーションチャンネルを使用する場合、**CHANGE REPLICATION SOURCE TO** ステートメントで **FOR CHANNEL channel** 句を使用してチャンネルを指定しないと、エラーが発生します。詳しくは [セクション17.2.2「レプリケーションチャンネル」](#) をご覧ください。

SOURCE_HOST およびその他の **CHANGE REPLICATION SOURCE TO** オプションに使用される値では、改行 (**\n** または **0x0A**) 文字がチェックされます。このような文字がこれらの値に存在すると、**ER_MASTER_INFO** でステートメントが失敗します。

CHANGE REPLICATION SOURCE TO を起動すると、**SOURCE_HOST**、**SOURCE_PORT**、**SOURCE_LOG_FILE** および **SOURCE_LOG_POS** の以前の値が、実行前のレプリカ状態に関するその他の情報とともにエラーログに書き込まれます。

`CHANGE REPLICATION SOURCE TO` では、進行中のトランザクションが暗黙的にコミットされます。 [セクション 13.3.3 「暗黙的なコミットを発生させるステートメント」](#) を参照してください。

`CHANGE REPLICATION SOURCE TO` ステートメントの一部のオプションでは、`CHANGE REPLICATION SOURCE TO` ステートメント (およびその後の `START REPLICA` ステートメント) を発行する前に、`STOP REPLICA` ステートメントを発行する必要があります。場合によっては、レプリケーション SQL スレッドまたはレプリケーション I/O スレッドのどちらか一方のみを停止する必要があります:

- SQL スレッドが停止すると、レプリケーション I/O スレッドが実行されている場合でも、`RELAY_LOG_FILE`、`RELAY_LOG_POS` および `SOURCE_DELAY` オプションの任意の組合せを使用して `CHANGE REPLICATION SOURCE TO` を実行できます。I/O スレッドの実行中は、このステートメントでほかのオプションを使用できません。
- I/O スレッドが停止すると、SQL スレッドが実行されている場合でも、このステートメント (任意の組合せで) `except RELAY_LOG_FILE, RELAY_LOG_POS, SOURCE_DELAY` または `SOURCE_AUTO_POSITION = 1` のオプションを使用して `CHANGE REPLICATION SOURCE TO` を実行できます。
- `SOURCE_AUTO_POSITION = 1` または `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` を使用する `CHANGE REPLICATION SOURCE TO` ステートメントを発行する前に、SQL スレッドと I/O スレッドの両方を停止する必要があります。

`SHOW REPLICA STATUS` を使用して、レプリケーション SQL スレッドおよびレプリケーション I/O スレッドの現在の状態を確認できます。Group Replication applier チャンネル (`group_replication_applier`) には I/O スレッドがなく、SQL スレッドのみがあることに注意してください。

詳細は、[セクション 17.4.8 「フェイルオーバー中のソースの切替え」](#) を参照してください。

ステートメントベースレプリケーションおよび一時テーブルを使用している場合は、`STOP REPLICA` ステートメントのあとの `CHANGE REPLICATION SOURCE TO` ステートメントがレプリカ上の一時テーブルの背後に残る可能性があります。これが発生するたびに警告 (`ER_WARN_OPEN_TEMP_TABLES_MUST_BE_ZERO`) が発行されるようになります。このような場合は、このような `CHANGE REPLICATION SOURCE TO` ステートメントを実行する前に、`Slave_open_temp_tables` システムステータス変数の値が 0 であることを確認することで回避できます。

`CHANGE REPLICATION SOURCE TO` は、ソースのスナップショットがあり、スナップショットの時刻に対応するソースバイナリログ座標を記録している場合にレプリカを設定する際に役立ちます。スナップショットをレプリカにロードしてソースと同期した後、レプリカで `CHANGE REPLICATION SOURCE TO SOURCE_LOG_FILE='log_name', SOURCE_LOG_POS=log_pos` を実行して、レプリカがソースバイナリログの読取りを開始する座標を指定できます。

次の例では、レプリカが使用するソースサーバーを変更し、レプリカが読取りを開始するソースバイナリログ座標を確立します:

```
CHANGE REPLICATION SOURCE TO
SOURCE_HOST='source2.example.com',
SOURCE_USER='replication',
SOURCE_PASSWORD='password',
SOURCE_PORT=3306,
SOURCE_LOG_FILE='source2-bin.001',
SOURCE_LOG_POS=4,
SOURCE_CONNECT_RETRY=10;
```

次の例は、使用される頻度の低い操作を示しています。これは、なんらかの理由で再実行するリレーログファイルがレプリカに存在する場合に使用されます。これを行うには、ソースにアクセスできる必要はありません。 `CHANGE REPLICATION SOURCE TO` のみを使用し、SQL スレッド (`START REPLICA SQL_THREAD`) を起動する必要があります:

```
CHANGE REPLICATION SOURCE TO
RELAY_LOG_FILE='replica-relay-bin.006',
RELAY_LOG_POS=4025;
```

`CHANGE REPLICATION SOURCE TO` ステートメントで指定しないオプションは、次の説明に示す場合を除き、その値を保持します。そのため、ほとんどの場合、変更されないオプションを指定する必要はありません。

`ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` オプションが GTID を持たないレプリケートされたトランザクションに GTID を割り当て、GTID ベースのレプリケーションを使用しないソースから使用するレプリカへのレプリケーションを有効にします。マルチソースレプリカ

の場合、[ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS](#) を使用するチャンネルと使用しないチャンネルを混在させることができます。デフォルトは `OFF` で、この機能は使用されません。

`LOCAL` は、レプリカ独自の UUID (`server_uuid` 設定) を含む GTID を割り当てます。`uuid` は、レプリケーションソースサーバーの `server_uuid` 設定など、指定された UUID を含む GTID を割り当てます。非ローカル UUID を使用すると、レプリカで発生したトランザクションと、ソースで発生したトランザクション、およびマルチソースレプリカの場合は異なるソースで発生したトランザクションを区別できません。選択した UUID は、レプリカ自体での使用にのみ意味があります。ソースによって送信されたトランザクションのいずれかに GTID がすでにある場合、その GTID は保持されます。

Group Replication に固有のチャンネルでは [ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS](#) を使用できませんが、Group Replication グループメンバーであるサーバーインスタンス上の別のソースの非同期レプリケーションチャンネルでは使用できます。その場合、GTID を作成するための UUID として Group Replication グループ名を指定しないでください。

[ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS](#) を `LOCAL` または `uuid` に設定するには、レプリカに `gtid_mode=ON` が設定されている必要があります。後で変更することはできません。このオプションは、バイナリログファイルの位置ベースのレプリケーションを持つソースで使用されるため、チャンネルに [MASTER_AUTO_POSITION=1](#) を設定することはできません。このオプションを設定する前に、レプリケーション SQL スレッドとレプリケーション I/O スレッドの両方を停止する必要があります。

重要

フェイルオーバーが必要な場合、どのチャンネルでも [ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS](#) を使用して設定されたレプリカを昇格させてレプリケーションサーバーを置き換えることはできず、レプリカから作成されたバックアップを使用してレプリケーションサーバーをリストアすることはできません。同じ制限が、任意のチャンネルで [ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS](#) を使用する他のレプリカの置換またはリストアにも適用されます。

その他の制限および詳細は、[セクション 17.1.3.6 「GTID のないソースから GTID のあるレプリカへのレプリケーション」](#) を参照してください。

[GET_SOURCE_PUBLIC_KEY](#)

ソースから公開キーをリクエストすることで、RSA キーペアベースのパスワード交換を有効にします。このオプションは、[caching_sha2_password](#) 認証プラグインで認証されるレプリカに適用されます。このプラグインを使用して認証されるアカウントによる接続の場合、ソースはリクエストされないうり公開キーを送信しないため、クライアントでリクエストまたは指定する必要があります。[SOURCE_PUBLIC_KEY_PATH](#) が指定され、有効な公開キーファイルが指定されている場合は、[GET_SOURCE_PUBLIC_KEY](#) よりも優先されます。[caching_sha2_password](#) プラグイン (MySQL 8.0 のデフォルト) で認証されるレプリケーションユーザーアカウントを使用していて、セキュアな接続を使用していない場合は、このオプションまたは [SOURCE_PUBLIC_KEY_PATH](#) オプションを指定して、RSA 公開キーをレプリカに提供する必要があります。

[IGNORE_SERVER_IDS](#)

レプリカが指定されたサーバーから発生したイベントを無視するようにします。このオプションは、0 個以上のサーバー ID のコンマ区切りリストを取ります。

サーバーからのログローテーションおよび削除イベントは無視されず、リレーログに記録されます。

循環レプリケーションでは、発信元のサーバーは通常、独自のイベントのターミネータとして機能するため、これらのイベントが複数回適用されることはありません。そのため、このオプションは、循環内のいずれかのサーバーが削除されたときの循環レプリケーションで役立ちます。1、2、3、および4のサーバーIDを持つ4台のサーバーを含む循環レプリケーションセットアップが存在するとき、サーバー3に障害が発生したとします。サーバー2からサーバー4へのレプリケーションを開始してギャップを埋める場合、サーバー4で発行する `CHANGE REPLICATION SOURCE TO` ステートメントに `IGNORE_SERVER_IDS = (3)` を含めて、サーバー3の代わりにサーバー2をソースとして使用するよう指示できます。それにより、サーバー4は、使用されなくなっているサーバーで発信されたすべてのステートメントを無視し、伝播しなくなります。

`IGNORE_SERVER_IDS` にサーバーの独自のIDが含まれているときに、`--replicate-same-server-id` オプションが有効な状態でそのサーバーが起動された場合は、エラーが発生します。

注記

レプリケーションにグローバルトランザクション識別子 (GTID) が使用されている場合、すでに適用されているトランザクションは自動的に無視されるため、`IGNORE_SERVER_IDS` 関数は必須ではなく、非推奨です。サーバーに `gtid_mode=ON` が設定されている場合、`CHANGE REPLICATION SOURCE TO` ステートメントに `IGNORE_SERVER_IDS` オプションを含めると、非推奨の警告が発行されます。

ソースメタデータリポジトリおよび `SHOW REPLICA STATUS` の出力では、現在無視されているサーバーのリストが提供されます。詳細は、[セクション17.2.4.2「レプリケーションメタデータリポジトリ」](#) および [セクション13.7.7.35「SHOW REPLICA | SLAVE STATUS ステートメント」](#) を参照してください。

`IGNORE_SERVER_IDS` オプションを指定せずに `CHANGE REPLICATION SOURCE TO` ステートメントを発行した場合、既存のリストは保持されます。無視されるサーバーのリストをクリアするには、このオプションを空のリストとともに使用する必要があります。

```
CHANGE REPLICATION SOURCE TO IGNORE_SERVER_IDS = ();
```

`RESET REPLICA ALL` により、`IGNORE_SERVER_IDS` がクリアされます。

注記

`IGNORE_SERVER_IDS` で既存のサーバーIDが設定されているチャンネルがある場合、`SET GTID_MODE=ON` が発行されると、非推奨の警告が発行されます。GTIDベースのレプリケーションを開始する前に、関係するサーバー上の無視されたすべてのサーバーIDリストを確認してクリアします。無視されたIDがある場合は、`SHOW REPLICA STATUS` ステートメントにリストが表示されます。非推奨の警告が表示された場合でも、空のリストを指定した `IGNORE_SERVER_IDS` オプションを含む `CHANGE REPLICATION SOURCE TO` ステートメントを発行することで、`gtid_mode=ON` の設定後にリストをクリアできます。

NETWORK_NAMESPACE

レプリケーションソースサーバーへの TCP/IP 接続に使用するネットワークネームスペース。このオプションを省略すると、レプリカからの接続にはデフォルト(グローバル)名前空間が使用されます。ネットワークネームスペースのサポートを実装していないプラットフォームでは、レプリカがソースに接続しようとするとうつらが発生します。ネットワークネームスペースの詳細は、[セクション5.1.14「ネットワークネームスペースのサポート」](#)を参照してください。**NETWORK_NAMESPACE** は、MySQL 8.0.22 の時点で使用可能です。

PRIVILEGE_CHECKS_USER

指定されたチャンネルのセキュリティコンテキストを提供するユーザーアカウントを指定します。**NULL**(デフォルト)は、セキュリティコンテキストが使用されないことを意味します。**PRIVILEGE_CHECKS_USER** は、MySQL 8.0.18 の時点で使用可能です。

ユーザーアカウントのユーザー名とホスト名は、[セクション6.2.4「アカウント名の指定」](#)で説明されている構文に従う必要があり、ユーザーは匿名ユーザー(ユーザー名が空白)または **CURRENT_USER** であってはなりません。アカウントには、**REPLICATION_APPLIER** 権限と、チャンネルでレプリケートされたトランザクションを実行するために必要な権限が必要です。アカウントに必要な権限の詳細は、[セクション17.3.3「レプリケーション権限チェック」](#)を参照してください。レプリケーションチャンネルを再起動すると、その時点から権限チェックが適用されます。チャンネルを指定せず、他のチャンネルが存在しない場合は、ステートメントがデフォルトチャンネルに適用されます。

PRIVILEGE_CHECKS_USER が設定されており、これを強制するように **REQUIRE_ROW_FORMAT** を設定できる場合は、行ベースのバイナリロギングを使用することを強くお勧めします。たとえば、実行中のレプリカでチャンネル **channel_1** に対する権限チェックを開始するには、次のステートメントを発行します:

```
mysql> STOP REPLICA FOR CHANNEL 'channel_1';
mysql> CHANGE REPLICATION SOURCE TO
  PRIVILEGE_CHECKS_USER = 'priv_repl'@'%example.com',
  REQUIRE_ROW_FORMAT = 1,
  FOR CHANNEL 'channel_1';
mysql> START REPLICA FOR CHANNEL 'channel_1';
```

RELAY_LOG_FILE,
RELAY_LOG_POS

次のスレッド起動時にレプリケーション SQL スレッドがレプリカリレーログからの読取りを開始するリレーログファイル名およびそのファイル内の場所。**SOURCE_LOG_FILE** または **SOURCE_LOG_POS** を使用している場合、これらのオプションは使用できません。

RELAY_LOG_FILE では、絶対パスまたは相対パスのいずれかを
使用でき、**SOURCE_LOG_FILE** と同じベース名を使用します。
RELAY_LOG_FILE、**RELAY_LOG_POS** またはその両方のオプションを使用する **CHANGE REPLICATION SOURCE TO** ステートメントは、レプリケーション SQL スレッドの停止時に実行中のレプリカで実行できます。少なくともいづれかのレプリケーション SQL スレッドとレプリケーション I/O スレッドが実行されている場合、リレーログは保持されます。両方のスレッドが停止すると、**RELAY_LOG_FILE** または **RELAY_LOG_POS** のいずれかが指定されていないかぎり、すべてのリレーログファイルが削除されます。Group Replication applier チャンネル (**group_replication_applier**) には I/O スレッドがなく、SQL スレッドのみがあることに注意してください。このチャンネルでは、SQL スレッドの停止時にリレーログは保持されません。

REQUIRE_ROW_FORMAT

レプリケーションチャンネルによる行ベースのレプリケーションイベントの処理のみを許可します。このオプションにより、レプリケーションアプライアンスは一時テーブルの作成や **LOAD DATA INFILE** リクエストの実行などのアクションを実行できなくなり、チャンネルのセキュリティが向上します。グループレプリケーションチャンネルは **REQUIRE_ROW_FORMAT** セットで自動的に作成され、これらのチャンネルのオプションは変更できません。詳細は、[セクション17.3.3「レプリケーション権限チェック」](#)を参照してください。**REQUIRE_ROW_FORMAT** は、MySQL 8.0.19 の時点で使用可能です。

REQUIRE_TABLE_PRIMARY_KEY_CHECK の値が主キーチェック用に独自のポリシーを選択できるようにします。レプリケーションチャンネルのオプションが **ON** に設定されている場合、レプリカはレプリケーション操作で常に **sql_require_primary_key** システム変数に値 **ON** を使用し、主キーが必要です。このオプションが **OFF** に設定されている場合、レプリカはレプリケーション操作で **sql_require_primary_key** システム変数に常に値 **OFF** を使用するため、ソースで必要な場合でも主キーは必要ありません。**REQUIRE_TABLE_PRIMARY_KEY_CHECK** オプションが **STREAM**(デフォルト) に設定されている場合、レプリカは各トランザクションのソースからレプリケートされた値を使用します。**REQUIRE_TABLE_PRIMARY_KEY_CHECK** は、MySQL 8.0.20 の時点で使用可能です。

マルチソースレプリケーションの場合、

REQUIRE_TABLE_PRIMARY_KEY_CHECK を **ON** または **OFF** に設定すると、レプリカは異なるソースのレプリケーションチャンネル間で動作を正規化し、**sql_require_primary_key** システム変数の一貫性のある設定を維持できます。**ON** を使用すると、複数のソースが同じテーブルセットを更新する場合に、主キーの偶発的な損失から保護されます。**OFF** を使用すると、主キーを操作できるソースを、操作できないソースと連携させることができます。

PRIVILEGE_CHECKS_USER が設定されている場合、

REQUIRE_TABLE_PRIMARY_KEY_CHECK を **ON** または **OFF** に設定することは、制限付きセッション変数を設定するためのセッション管理レベルの権限がユーザーアカウントに必要なことを意味します。これは、**sql_require_primary_key** の値を各トランザクションのソース設定と一致するように変更するために必要です。詳細は、[セクション17.3.3「レプリケーション権限チェック」](#)を参照してください。

SOURCE_AUTO_POSITION

GTID ベースのレプリケーションを使用してレプリカがソースに接続しようとする。このオプションは、レプリケーション SQL スレッドとレプリケーション I/O スレッドの両方が停止している場合にのみ、**CHANGE REPLICATION SOURCE TO** で使用できます。

レプリカとソースの両方で GTID が有効になっている必要があります (レプリカでは **GTID_MODE=ON**、**ON_PERMISSIVE**、または **OFF_PERMISSIVE**、ソースでは **GTID_MODE=ON**)。接続には自動配置が使用されるため、**SOURCE_LOG_FILE** および **SOURCE_LOG_POS** で表される座標は使用されず、これらのオプションのいずれかまたは両方を **SOURCE_AUTO_POSITION = 1** とともに使用するとエラーが発生します。レプリカでマルチソースレプリケーションが有効になっている場合は、適用可能なレプリケーションチャンネルごとに **SOURCE_AUTO_POSITION = 1** オプションを設定する必要があります。

SOURCE_AUTO_POSITION = 1 が設定されている場合、レプリカは初期接続ハンドシェイクで、すでに受信、コミット、またはその両方を行ったトランザクションを含む GTID セットを送信します。ソースは、GTID がレプリカによって送信される GTID セットに含まれていないバイナリログに記録されたすべてのトランザクションを送信することによって応答します。この交換により、レプリカがまだ記録またはコミットしていない GTID を持つトランザクションのみがソースから送信されるようになります。ダイヤモンドトポロジの場合と同様に、レプリカが複数のソースからトランザクションを受信する場合、自動スキップ機能によってトランザクションが 2 回適用されないようにします。レプリカによって送信される GTID セットの計算方法の詳細は、[セクション17.1.3.3「GTID 自動配置」](#)を参照してください。

ソースによって送信されるべきトランザクションのいずれかがソースバイナリログからパージされているか、別の方法で **gtid_purged** システム変数の GTID セットに追加されている場合、ソースはエラー **ER_MASTER_HAS_PURGED_REQUIRED_GTIDS** をレプリカに送信し、レプリケーションは開始しません。欠落しているパージ済トランザクションの GTID が識別され、警告メッセージ **ER_FOUND_MISSING_GTIDS** のソースエラーログにリストされます。また、トランザクションの交換中に、レプリカが GTID 内のソース UUID を持つトランザクションを記録またはコミットしたが、ソース自体

がそれらをコミットしていないことが判明した場合、ソースはレプリカにエラー `ER_SLAVE_HAS_MORE_GTIDS_THAN_MASTER` を送信し、レプリケーションは開始しません。これらの状況の処理方法の詳細は、[セクション17.1.3.3「GTID 自動配置」](#) を参照してください。

レプリケーションが GTID 自動配置を有効にして実行されているかどうかを確認するには、パフォーマンススキーマの `replication_connection_status` テーブルまたは `SHOW REPLICA STATUS` の出力を確認します。 `SOURCE_AUTO_POSITION` オプションを再度無効にすると、レプリカはファイルベースレプリケーションに戻ります。その場合は、`SOURCE_LOG_FILE` または `SOURCE_LOG_POS` オプションのいずれかまたは両方も指定する必要があります。

SOURCE_BIND

複数のネットワークインタフェースを持つレプリカで使用するために、ソースへの接続に選択するレプリカネットワークインタフェースを決定します。このオプションで構成されたアドレスは、`SHOW REPLICA STATUS` からの出力の `Source_Bind` カラムに表示されます (存在する場合)。ソースメタデータリポジトリテーブル `mysql.slave_master_info` では、値は `Source_bind` カラムとして表示されます。レプリカを特定のネットワークインタフェースにバインドする機能は、NDB Cluster でもサポートされています。

SOURCE_COMPRESSION_ALGORITHM

レプリケーションソースサーバーへの接続に許可される圧縮アルゴリズムを指定します。使用可能なアルゴリズムは、`protocol_compression_algorithms` システム変数の場合と同じです。デフォルト値は `uncompressed` です。 `SOURCE_COMPRESSION_ALGORITHMS` は、MySQL 8.0.18 の時点で使用可能です。

`SOURCE_COMPRESSION_ALGORITHMS` の値は、`slave_compressed_protocol` システム変数が無効な場合にのみ適用されます。 `slave_compressed_protocol` が有効な場合は、`SOURCE_COMPRESSION_ALGORITHMS` よりも優先され、ソースとレプリカの両方がそのアルゴリズムをサポートしている場合は、ソースへの接続で `zlib` 圧縮が使用されます。詳細は、[セクション4.2.8「接続圧縮制御」](#) を参照してください。

`binlog_transaction_compression` システム変数によってアクティブ化されるバイナリログトランザクション圧縮 (MySQL 8.0.20 で使用可能) を使用して帯域幅を節約することもできます。これを接続圧縮と組み合わせて行くと、接続圧縮ではデータを処理する機会は少なくなりますが、ヘッダーと、圧縮されていないイベントおよびトランザクションペイロードは圧縮できます。バイナリログのトランザクション圧縮の詳細は、[セクション5.4.4.5「バイナリログトランザクション圧縮」](#) を参照してください。

SOURCE_CONNECT_RETRY

ソースへの接続がタイムアウトした後にレプリカが試行する再接続の間隔を指定します。試行は、`SOURCE_RETRY_COUNT` オプションによって制限されます。両方のデフォルト設定が使用されている場合、レプリカは再接続試行 (`SOURCE_CONNECT_RETRY=60`) の間 60 秒待機し、60 日間 (`SOURCE_RETRY_COUNT=86400`) 再接続を試行し続けます。これらの値はソースメタデータリポジトリに記録され、`replication_connection_configuration` 「パフォーマンススキーマ」テーブルに表示されます。

SOURCE_CONNECTION_AUTO_FAILOVER

以下の代替レプリケーションサーバーが使用可能な場合 (レプリケートされたデータを共有する複数の MySQL サーバーまたはサーバーグループが存在する場合)、レプリケーションチャネルの非同期接続フェイルオーバーメカニズムをアクティブ化します。 `SOURCE_CONNECTION_AUTO_FAILOVER` は、MySQL 8.0.22 の時点で使用可能です。非同期接続フェイルオーバーメカニズムは、`SOURCE_CONNECT_RETRY` および `SOURCE_RETRY_COUNT` によって制御されている再接続試行を使い果たした後に引き継ぎます。 `asynchronous_connection_failover_add_source` および `asynchronous_connection_failover_delete_source` UDF を使用して管理する、指定されたソースリストから選択された代替ソースにレプリカを再接続します。サーバーの管理対象グループを追加および削除するには、かわりに `asynchronous_connection_failover_add_managed` および

`asynchronous_connection_failover_delete_managed` UDF を使用します。詳細は、[セクション17.4.9「非同期接続フェイルオーバーによるソースの切替え」](#)を参照してください。

重要

1. GTID 自動配置が使用中
(`SOURCE_AUTO_POSITION = 1`) の場合には、`SOURCE_CONNECTION_AUTO_FAILOVER = 1` を設定できます。
2. `SOURCE_CONNECTION_AUTO_FAILOVER = 1` を設定する場合、一時的なネットワークの停止によって接続障害が発生した場合に備えて、`SOURCE_RETRY_COUNT` および `SOURCE_CONNECT_RETRY` を同じソースでの再試行回数を最小限に抑えるように設定します。そうしないと、非同期接続フェイルオーバーメカニズムをすぐにアクティブ化できません。適切な値は `SOURCE_RETRY_COUNT=3` と `SOURCE_CONNECT_RETRY=10` です。これにより、レプリカは 10 秒間隔で接続を 3 回再試行します。
3. `SOURCE_CONNECTION_AUTO_FAILOVER = 1` を設定する場合、レプリケーションメタデータリポジトリには、レプリケーションチャンネルのソースリスト上のすべてのサーバーへの接続に使用できるレプリケーションユーザーアカウントの資格証明が含まれている必要があります。アカウントには、「パフォーマンススキーマ」テーブルに対する `SELECT` 権限も必要です。これらの資格証明は、`SOURCE_USER` および `SOURCE_PASSWORD` オプションを指定した `CHANGE REPLICATION SOURCE TO` ステートメントを使用して設定できます。詳細は、[セクション17.4.9「非同期接続フェイルオーバーによるソースの切替え」](#)を参照してください。

`SOURCE_DELAY`

レプリカが遅延する必要があるソースからの遅延秒数を指定します。ソースから受信したイベントは、ソースでの実行より少なくとも `interval` 秒後に実行されません。デフォルトは 0 です。`interval` が 0 から $2^{31}-1$ までの範囲の負でない整数でない場合は、エラーが発生します。詳細は、[セクション17.4.11「遅延レプリケーション」](#)を参照してください。`SOURCE_DELAY` オプションを使用する `CHANGE REPLICATION SOURCE TO` ステートメントは、レプリケーション SQL スレッドの停止時に実行中のレプリカで実行できます。

`SOURCE_HEARTBEAT_PERIOD`

ハートビート間隔を制御します。これにより、接続がまだ良好な場合に、データが存在しないときに接続タイムアウトが発生しなくなります。ハートビートシグナルは、その秒数が経過するとレプリカに送信され、ソースバイナリログがイベントで更新されるたびに待機期間がリセットされます。したがって、ハートビートは、これより長い期間バイナリログファイルに未送信のイベントがない場合のみ、ソースによって送信されます。

ハートビート間隔 `interval` は、0 から 4294967 秒の範囲の小数値およびミリ秒単位の解像度です。ゼロ以外の最小値は 0.001 です。`interval` を 0 に設定すると、ハートビートが完全に無効になります。ハートビート間隔のデフォルトは、`slave_net_timeout` システム変数の値の半分です。ソースメタデータリポジトリに記録され、`replication_connection_configuration` 「パフォーマンススキーマ」

テーブルに表示されます。 `RESET REPLICA` を発行すると、ハートビート間隔がデフォルト値にリセットされます。

`slave_net_timeout` システム変数は、レプリカがソースからのデータまたはハートビートシグナルのいずれかを待機する秒数を指定します。この秒数を過ぎると、レプリカは接続が切断されたとみなし、読取りを中断して再接続を試行します。デフォルト値は 60 秒 (1 分) です。 `slave_net_timeout` の値またはデフォルト設定を変更しても、明示的に設定されているか、以前に計算されたデフォルトを使用しているかにかかわらず、ハートビート間隔は自動的に変更されないことに注意してください。 `@@GLOBAL.slave_net_timeout` を現在のハートビート間隔より小さい値に設定すると、警告が発行されます。 `slave_net_timeout` が変更された場合は、接続タイムアウトの前にハートビートシグナルが発生するように、 `CHANGE REPLICATION SOURCE TO` を発行してハートビート間隔を適切な値に調整する必要があります。これを行わない場合、ハートビートシグナルは効果がなく、ソースからデータを受信しない場合、レプリカは再接続を繰り返し試行してゾンビダンプレスレッドを作成できます。

SOURCE_HOST

レプリケーションソースサーバーのホスト名または IP アドレス。レプリカはこれを使用してソースに接続します。

`SOURCE_HOST` または `SOURCE_PORT` を指定した場合、レプリカは、ソースサーバーが以前とは異なることを前提とします (オプション値が現在の値と同じであっても。) この場合、ソースバイナリログファイルの名前と位置の古い値は適用できなくなるため、ステートメントに `SOURCE_LOG_FILE` と `SOURCE_LOG_POS` を指定しないと、`SOURCE_LOG_FILE=""` と `SOURCE_LOG_POS=4` は暗黙的に追加されます。

`SOURCE_HOST=""` を設定する (つまり、その値を明示的に空の文字列に設定する) ことは、`SOURCE_HOST` をまったく設定しないこととは異なります。`SOURCE_HOST` を空の文字列に設定しようとすると、エラーで失敗します。

SOURCE_LOG_FILE, SOURCE_LOG_POS

バイナリログファイル名、およびレプリケーション I/O スレッドが次のスレッド起動時にソースバイナリログからの読取りを開始するそのファイル内の場所。バイナリログファイルの位置ベースのレプリケーションを使用している場合は、これらのオプションを指定します。 `SOURCE_LOG_FILE` には、ソースサーバーで使用可能な特定のバイナリログファイル (`SOURCE_LOG_FILE='binlog.000145'` など) の数値接尾辞が含まれている必要があります。 `SOURCE_LOG_POS` は、レプリカがそのファイルの読取りを開始する数値位置です。 `SOURCE_LOG_POS=4` は、バイナリログファイルでイベントの開始を表します。

`SOURCE_LOG_FILE` または `SOURCE_LOG_POS` のいずれかを指定する場合、`RELAY_LOG_FILE` または `RELAY_LOG_POS` は指定できません。 `SOURCE_LOG_FILE` または `SOURCE_LOG_POS` のいずれかを指定する場合は、GTID ベースのレプリケーション用の `SOURCE_AUTO_POSITION = 1` も指定できません。

`SOURCE_LOG_FILE` も `SOURCE_LOG_POS` も指定されていない場合、レプリカは `CHANGE REPLICATION SOURCE TO` が発行される前のレプリケーション SQL スレッドの最後の座標を使用します。これにより、レプリケーション SQL スレッドがレプリケーション I/O スレッドと比較して遅延していても、レプリケーションの不連続性がなくなります。

SOURCE_PASSWORD

レプリケーションソースサーバーへの接続に使用するレプリケーションユーザーアカウントのパスワード。 `SOURCE_PASSWORD` を指定する場合は、`SOURCE_USER` も必要です。

`CHANGE REPLICATION SOURCE TO` ステートメントでレプリケーションユーザーアカウントに使用されるパスワードの長さは、32 文字に制限されています。32 文字を超えるパスワードを使用しようとすると、`CHANGE REPLICATION SOURCE TO` で障害が発生します。

SOURCE_PORT

レプリカがレプリケーションソースサーバーへの接続に使用する TCP/IP ポート番号。

注記

レプリケーションでは、Unix ソケットファイルを使用できません。TCP/IP を使用してレプリケーションソースサーバーに接続する必要があります。

`SOURCE_HOST` または `SOURCE_PORT` を指定した場合、レプリカは、ソースサーバーが以前とは異なることを前提とします (オプション値が現在の値と同じであっても)。) この場合、ソースバイナリログファイルの名前と位置の古い値は適用できなくなるため、ステートメントに `SOURCE_LOG_FILE` と `SOURCE_LOG_POS` を指定しないと、`SOURCE_LOG_FILE=""` と `SOURCE_LOG_POS=4` は暗黙的に追加されます。

SOURCE_PUBLIC_KEY_PATH

ソースが必要とする公開キーのレプリカ側のコピーを含むファイルへのパス名を指定することで、RSA キーペアベースのパスワード交換を有効にします。ファイルは PEM 形式である必要があります。このオプションは、`sha256_password` または `caching_sha2_password` 認証プラグインで認証されるレプリカに適用されます。(`sha256_password` の場合、`SOURCE_PUBLIC_KEY_PATH` は、MySQL が OpenSSL を使用して構築された場合にのみ使用できます。) `caching_sha2_password` プラグイン (MySQL 8.0 のデフォルト) で認証されるレプリケーションユーザーアカウントを使用していて、セキュアな接続を使用していない場合は、このオプションまたは `GET_SOURCE_PUBLIC_KEY=1` オプションを指定して、RSA 公開キーをレプリカに提供する必要があります。

SOURCE_RETRY_COUNT

`slave_net_timeout` システム変数によって決定される、ソースへの接続がタイムアウトした後にレプリカが行う再接続の最大試行回数を設定します。レプリカを再接続する必要がある場合、タイムアウトの直後に最初の再試行が行われます。試行の間隔は、`SOURCE_CONNECT_RETRY` オプションで指定します。両方のデフォルト設定が使用されている場合、レプリカは再接続試行 (`SOURCE_CONNECT_RETRY=60`) の間 60 秒待機し、60 日間 (`SOURCE_RETRY_COUNT=86400`) 再接続を試行し続けます。これらの値はソースメタデータリポジトリに記録され、`replication_connection_configuration` 「パフォーマンススキーマ」テーブルに表示されます。`SOURCE_RETRY_COUNT` は、`--master-retry-count` サーバーの起動オプションより優先されます。

SOURCE_SSL_XXX, SOURCE_TLS_XXX

レプリカが暗号化および暗号化を使用してレプリケーション接続を保護する方法を指定します。これらのオプションは、SSL サポートなしでコンパイルされたレプリカでも変更できます。これらはソースメタデータリポジトリに保存されますが、レプリカで SSL サポートが有効になっていない場合は無視されます。`SOURCE_SSL_XXX` および `SOURCE_TLS_XXX` オプションは、**暗号化接続のコマンドオプション** で説明されている `--ssl-xxx` および `--tls-xxx` クライアントオプションと同じ機能を実行します。2 つのオプションセット間の対応、および `SOURCE_SSL_XXX` オプションと `SOURCE_TLS_XXX` オプションを使用したセキュアな接続の設定については、**セクション 17.3.1 「暗号化接続を使用するためのレプリケーションの設定」** を参照してください。

SOURCE_USER

レプリケーションソースサーバーへの接続に使用するレプリケーションユーザーアカウントのユーザー名。

重要

`caching_sha2_password` プラグインで認証するレプリケーションユーザーアカウントを使用してソースに接続するには、**セクション 17.3.1 「暗号化接続を使用するためのレプリケーションの設定」** の説明に従ってセキュアな接続を設定するか、RSA キーペアを使用したパスワード交換をサポートするように暗号化されていない接続を有効にする必要があります。`caching_sha2_password` 認証

プラグインは、MySQL 8.0 から作成された新規ユーザーのデフォルトです (詳細は、[セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#) を参照)。レプリケーション用に作成または使用するユーザーアカウントがこの認証プラグインを使用しており、セキュアな接続を使用していない場合は、正常に接続するために RSA キーペアベースのパスワード交換を有効にする必要があります。これは、`SOURCE_PUBLIC_KEY_PATH` オプションまたはこのステートメントの `GET_SOURCE_PUBLIC_KEY=1` オプションを使用して実行できます。

`SOURCE_USER=""` を指定して空のユーザー名を設定することはできますが、レプリケーションチャンネルは空のユーザー名で開始できません。MySQL 8.0.21 より前のリリースでは、セキュリティ上の目的で以前に使用した資格証明をレプリケーションメタデータリポジトリからクリアする必要がある場合にのみ、空の `SOURCE_USER` ユーザー名を設定します。これらのリリースでは、リポジトリから空のユーザー名が読み取られた場合 (グループレプリケーションチャンネルの自動再起動時など)、デフォルトのユーザー名を置換できるバグのため、後でチャンネルを使用しないでください。MySQL 8.0.21 からは、レプリケーションチャンネルを開始する `START REPLICA` ステートメントまたは `START GROUP_REPLICATION` ステートメントを使用して常にユーザー資格証明を指定する場合、空の `SOURCE_USER` ユーザー名を設定し、後でチャンネルを使用できます。このアプローチでは、レプリケーションチャンネルを再起動するためにオペレータの介入が常に必要ですが、ユーザー資格証明はレプリケーションメタデータリポジトリに記録されません。

`SOURCE_USER` および `SOURCE_PASSWORD` の値を含む実行中の `CHANGE REPLICATION SOURCE TO` ステートメントのテキストは、同時 `SHOW PROCESSLIST` ステートメントの出力に表示されます。(`START REPLICA` ステートメントの完全なテキストは、`SHOW PROCESSLIST` にも表示されます。)

`SOURCE_ZSTD_COMPRESSION_LEVEL` は、zstd 圧縮アルゴリズムを使用するレプリケーションソースサーバーへの接続に使用する圧縮レベル。許可されるレベルは 1 から 22 で、大きい値は圧縮レベルの増加を示します。デフォルトの `zstd` 圧縮レベルは 3 です。圧縮レベルの設定は、`zstd` 圧縮を使用しない接続には影響しません。`SOURCE_ZSTD_COMPRESSION_LEVEL` は、MySQL 8.0.18 の時点で使用可能です。詳細は、[セクション4.2.8「接続圧縮制御」](#) を参照してください。

次の表は、文字列値のオプションに許可される最大長を示しています。

オプション	最大長
<code>SOURCE_HOST</code>	255
<code>SOURCE_USER</code>	96
<code>SOURCE_PASSWORD</code>	32
<code>SOURCE_LOG_FILE</code>	511
<code>RELAY_LOG_FILE</code>	511
<code>SOURCE_SSL_CA</code>	511
<code>SOURCE_SSL_CAPATH</code>	511
<code>SOURCE_SSL_CERT</code>	511
<code>SOURCE_SSL_CRL</code>	511
<code>SOURCE_SSL_CRLPATH</code>	511
<code>SOURCE_SSL_KEY</code>	511
<code>SOURCE_SSL_CIPHER</code>	511
<code>SOURCE_TLS_VERSION</code>	511
<code>SOURCE_TLS_CIPHERSUITES</code>	4000

オプション	最大長
<code>SOURCE_PUBLIC_KEY_PATH</code>	511
<code>SOURCE_COMPRESSION_ALGORITHMS</code>	99
<code>NETWORK_NAMESPACE</code>	64

13.4.2.4 MASTER_POS_WAIT() ステートメント

```
SELECT MASTER_POS_WAIT('source_log_file', source_log_pos [, timeout][, channel])
```

これはステートメントではなく、実際には関数です。これは、レプリカがソースバイナリログ内の指定された位置までイベントを読み取って実行したことを確認するために使用されます。完全な説明については、[セクション 12.24 「その他の関数」](#) を参照してください。

13.4.2.5 RESET REPLICA | SLAVE ステートメント

```
RESET {REPLICA | SLAVE} [ALL] [channel_option]
```

`channel_option`:
FOR CHANNEL `channel`

`RESET REPLICA | SLAVE` では、レプリカはソースバイナリログ内の位置を忘れます。MySQL 8.0.22 から、`RESET SLAVE` のかわりに `RESET REPLICA` を使用します。これは、そのリリースから非推奨になりました。MySQL 8.0.22 より前のリリースでは、`RESET SLAVE` を使用します。

このステートメントはクリーンスタートに使用されます。レプリケーションメタデータリポジトリをクリアし、すべてのリレーログファイルを削除して、新しいリレーログファイルを開始します。また、`CHANGE REPLICATION SOURCE TO` ステートメント (MySQL 8.0.23 から) または `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 より前) の `SOURCE_DELAY | MASTER_DELAY` オプションで指定されたレプリケーション遅延を 0 にリセットします。

注記

レプリケーション SQL スレッドによって完全に実行されていない場合でも、リレーログファイルはすべて削除されます。(`STOP REPLICA | SLAVE` ステートメントを発行した場合、またはレプリカの負荷が高い場合、これはレプリカに存在する可能性があります。)

GTID が使用されている (`gtid_mode` が ON) サーバーの場合、`RESET REPLICA | SLAVE` を発行しても GTID 実行履歴には影響しません。このステートメントでは、`gtid_executed`、`gtid_purged` または `mysql.gtid_executed` テーブルの値は変更されません。GTID 実行履歴をリセットする必要がある場合は、GTID 対応サーバーがバイナリロギングが無効になっているレプリカであっても、`RESET MASTER` を使用します。

`RESET REPLICA | SLAVE` には、`RELOAD` 権限が必要です。

`RESET REPLICA | SLAVE` を使用するには、レプリケーション SQL スレッドおよびレプリケーション I/O スレッドを停止する必要があるため、実行中のレプリカでは、`RESET REPLICA | SLAVE` を発行する前に `STOP REPLICA | SLAVE` を使用します。グループレプリケーショングループメンバーで `RESET REPLICA | SLAVE` を使用するには、メンバーステータスが `OFFLINE` である必要があります。つまり、プラグインはロードされますが、メンバーは現在のどのグループにも属していません。グループメンバーは、`STOP GROUP REPLICATION` ステートメントを使用してオフラインにできます。

オプションの `FOR CHANNEL channel` 句を使用すると、ステートメントが適用されるレプリケーションチャンネルの名前を指定できます。`FOR CHANNEL channel` 句を指定すると、`RESET REPLICA | SLAVE` ステートメントが特定のレプリケーションチャンネルに適用されます。`FOR CHANNEL channel` 句を `ALL` オプションと組み合わせると、指定したチャンネルが削除されます。チャンネルが指定されておらず、追加のチャンネルが存在しない場合、ステートメントはデフォルトチャンネルに適用されます。複数のレプリケーションチャンネルが存在する場合に `FOR CHANNEL channel` 句なしで `RESET REPLICA | SLAVE ALL` ステートメントを発行すると、all レプリケーションチャンネルが削除され、デフォルトチャンネルのみが再作成されます。詳しくは [セクション 17.2.2 「レプリケーションチャンネル」](#) をご覧ください。

`RESET REPLICA | SLAVE` では、ソースホスト名とポート、レプリケーションユーザーアカウントとそのパスワード、`PRIVILEGE_CHECKS_USER` アカウント、`REQUIRE_ROW_FORMAT` オプショ

ン、[REQUIRE_TABLE_PRIMARY_KEY_CHECK](#) オプション、[ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS](#) オプションなどのレプリケーション接続パラメータは変更されません。レプリケーション接続パラメータを変更する場合は、サーバーの起動後に [CHANGE REPLICATION SOURCE TO](#) ステートメント (MySQL 8.0.23 の場合) または [CHANGE MASTER TO](#) ステートメント (MySQL 8.0.23 の場合) を使用します。すべてのレプリケーション接続パラメータを削除する場合は、[RESET REPLICA | SLAVE ALL](#) を使用します。[RESET REPLICA | SLAVE ALL](#) では、[CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO](#) によって設定された [IGNORE_SERVER_IDS](#) リストもクリアされます。[RESET REPLICA | SLAVE ALL](#) を使用している場合、インスタンスをレプリカとして再度使用するには、サーバーの起動後に [CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO](#) ステートメントを発行して、新しい接続パラメータを指定する必要があります。

[RESET REPLICA | SLAVE](#) を発行した後、[START REPLICA | SLAVE](#) を発行する前に予期しないサーバーの終了または意図的な再起動が発生した場合、レプリケーション接続パラメータの保存は、レプリケーションメタデータに使用されるリポジトリによって異なります:

- [master_info_repository=TABLE](#) および [relay_log_info_repository=TABLE](#) がサーバーに設定されている場合 (MySQL 8.0 からのデフォルト設定)、レプリケーション接続パラメータは、InnoDB テーブル [mysql.slave_master_info](#) および [mysql.slave_relay_log_info](#) のクラッシュセーフ [RESET REPLICA | SLAVE](#) 操作の一部として保持されます。これらはメモリーにも保持されます。[RESET REPLICA | SLAVE](#) の発行後、[START REPLICA | SLAVE](#) の発行前に予期しないサーバーの終了または意図的な再起動が発生した場合、レプリケーション接続パラメータがテーブルから取得され、チャンネルに再適用されます。この状況は、接続メタデータリポジトリの場合は MySQL 8.0.13 から、アプライマメタデータリポジトリの場合は MySQL 8.0.19 から適用されます。
- MySQL 8.0 から非推奨になった [master_info_repository=FILE](#) および [relay_log_info_repository=FILE](#) がサーバーに設定されている場合、または MySQL Server リリースが前述のリリースより前の場合、レプリケーション接続パラメータはメモリーにのみ保持されます。予期しないサーバーの終了または故意の再起動が原因で、[RESET REPLICA | SLAVE](#) の発行直後にレプリカ [mysqld](#) が再起動された場合、接続パラメータは失われます。その場合は、[START REPLICA | SLAVE](#) を発行する前に、サーバーが接続パラメータの再指定を開始した後で、(MySQL 8.0.23 から) [CHANGE REPLICATION SOURCE TO](#) ステートメントまたは [CHANGE MASTER TO](#) ステートメントを発行する必要があります。

[RESET REPLICA | SLAVE](#) では、ステートメントの影響を受けるチャンネルのレプリケーションフィルタ設定 ([--replicate-ignore-table](#) など) は変更されません。ただし、[RESET REPLICA | SLAVE ALL](#) では、ステートメントによって削除されたチャンネルに設定されたレプリケーションフィルタが削除されます。削除されたチャンネルが再作成されると、レプリカに指定されたグローバルレプリケーションフィルタがそれらにコピーされ、チャンネル固有のレプリケーションフィルタは適用されません。詳細は、[セクション17.2.5.4「レプリケーションチャンネルベースのフィルタ」](#) を参照してください。

[RESET REPLICA | SLAVE](#) では、進行中のトランザクションが暗黙的にコミットされます。[セクション13.3.3「暗黙的なコミットを発生させるステートメント」](#) を参照してください。

レプリケーション SQL スレッドが停止時に一時テーブルのレプリケート中で、[RESET REPLICA | SLAVE](#) が発行された場合、レプリケートされた一時テーブルはレプリカで削除されます。

[RESET REPLICA | SLAVE](#) はハートビート期間または [SSL_VERIFY_SERVER_CERT](#) をリセットしません。

注記

NDB Cluster レプリカ SQL ノードで使用すると、[RESET REPLICA | SLAVE](#) は [mysql.ndb_apply_status](#) テーブルをクリアします。このステートメントを使用する場合、[ndb_apply_status](#) は NDB ストレージエンジンを使用するため、クラスタに接続されているすべての SQL ノードによって共有されることに注意してください。

この動作をオーバーライドするには、[RESET REPLICA | SLAVE](#) を実行する前に [SET GLOBAL @@ndb_clear_apply_status=OFF](#) を発行します。このような場合、レプリカは [ndb_apply_status](#) テーブルをパージしません。

13.4.2.6 RESET SLAVE | REPLICA ステートメント

```
RESET {SLAVE | REPLICA} [ALL] [channel_option]
```

channel_option:

```
FOR CHANNEL channel
```

レプリカがソースバイナリログ内の位置を忘れられるようにします。MySQL 8.0.22 からは、[RESET SLAVE](#) は非推奨であり、かわりにエイリアス [RESET REPLICA](#) を使用する必要があります。MySQL 8.0.22 より前のリリースでは、[RESET SLAVE](#) を使用します。ステートメントは以前と同様に機能し、ステートメントおよびその出力に使用される用語のみが変更されています。どちらのバージョンのステートメントも、使用時に同じステータス変数を更新します。ステートメントの説明は、[RESET REPLICA](#) のドキュメントを参照してください。

13.4.2.7 START REPLICA | SLAVE ステートメント

```
START {REPLICA | SLAVE} [thread_types] [until_option] [connection_options] [channel_option]
```

thread_types:

```
[thread_type [, thread_type] ... ]
```

thread_type:

```
IO_THREAD | SQL_THREAD
```

until_option:

```
UNTIL { {SQL_BEFORE_GTIDS | SQL_AFTER_GTIDS} = gtid_set
| MASTER_LOG_FILE = 'log_name', MASTER_LOG_POS = log_pos
| SOURCE_LOG_FILE = 'log_name', SOURCE_LOG_POS = log_pos
| RELAY_LOG_FILE = 'log_name', RELAY_LOG_POS = log_pos
| SQL_AFTER_MTS_GAPS }
```

connection_options:

```
[USER='user_name'] [PASSWORD='user_pass'] [DEFAULT_AUTH='plugin_name'] [PLUGIN_DIR='plugin_dir']
```

channel_option:

```
FOR CHANNEL channel
```

gtid_set:

```
uuid_set [, uuid_set] ...
| "
```

uuid_set:

```
uuid:interval[:interval]...
```

uuid:

```
hhhhhhhh-hhhh-hhhh-hhhh-hhhhhhhhhhhh
```

h:

```
[0-9,A-F]
```

interval:

```
n[-n]
```

```
(n >= 1)
```

[START REPLICA | SLAVE](#) は、レプリケーションスレッドの一方または両方を起動します。MySQL 8.0.22 から、[START SLAVE](#) のかわりに [START REPLICA](#) を使用します。これは、そのリリースから非推奨になりました。MySQL 8.0.22 より前のリリースでは、[START SLAVE](#) を使用します。

[thread_type](#) オプションを指定しない [START REPLICA | SLAVE](#) は、両方のレプリケーションスレッドを起動します。レプリケーション I/O スレッドは、ソースサーバーからイベントを読み取り、リレーログに格納します。レプリケーション SQL スレッドは、リレーログからイベントを読み取り、それらを実行します。[START REPLICA | SLAVE](#) には、[REPLICATION_SLAVE_ADMIN](#) 権限 (または非推奨の [SUPER](#) 権限) が必要です。

[START REPLICA | SLAVE](#) がレプリケーションスレッドの起動に成功すると、エラーなしで返されます。ただし、その場合でも、レプリケーションスレッドが起動してから後で停止する可能性があります (たとえば、ソースへの接続やバイナリログの読み取りなどの問題が管理されないため)。[START REPLICA | SLAVE](#) では、これについての警告は表示されません。レプリカエラーログでレプリケーションスレッドによって生成されたエラーメッセージを確認するが、[SHOW REPLICA | SLAVE STATUS](#) で正常に実行されていることを確認する必要があります。

[START REPLICA | SLAVE](#) では、進行中のトランザクションが暗黙的にコミットされます。[セクション13.3.3「暗黙的なコミットを発生させるステートメント」](#)を参照してください。

このステートメントを発行する前に、[gtid_next](#) を [AUTOMATIC](#) に設定する必要があります。

オプションの `FOR CHANNEL channel` 句を使用すると、ステートメントが適用されるレプリケーションチャンネルの名前を指定できます。 `FOR CHANNEL channel` 句を指定すると、`START REPLICA | SLAVE` ステートメントが特定のレプリケーションチャンネルに適用されます。句が指定されておらず、追加のチャンネルが存在しない場合、ステートメントはデフォルトチャンネルに適用されます。複数のチャンネルを使用するときに `START REPLICA | SLAVE` ステートメントにチャンネルが定義されていない場合、このステートメントはすべてのチャンネルに対して指定されたスレッドを開始します。このステートメントは `group_replication_recovery` チャンネルでは許可されていません。詳しくは [セクション17.2.2「レプリケーションチャンネル」](#) をご覧ください。

どちらのスレッドを開始するかを指定するために、このステートメントに `IO_THREAD` および `SQL_THREAD` オプションを追加できます。Group Replication applier チャンネル (`group_replication_applier`) には I/O スレッドがなく、SQL スレッドのみがあることに注意してください。このチャンネルの起動時に `IO_THREAD` または `SQL_THREAD` オプションを指定しても利点はありません。

`START REPLICA | SLAVE` では、次のリストに示すように、`USER`、`PASSWORD`、`DEFAULT_AUTH` および `PLUGIN_DIR` オプションを使用したプラグブルユーザーパスワード認証がサポートされます：

- `USER`: ユーザー名。 `PASSWORD` が使用されている場合は、空または NULL 文字列に設定したり、未設定のままにしたりすることはできません。
- `PASSWORD`: パスワード。
- `DEFAULT_AUTH`: プラグインの名前。デフォルトは MySQL ネイティブ認証です。
- `PLUGIN_DIR`: プラグインの場所。

`IO_THREAD` オプションも指定されていないかぎり、`USER`、`PASSWORD`、`DEFAULT_AUTH` または `PLUGIN_DIR` のいずれかを指定する場合は、`SQL_THREAD` オプションを使用できません。

詳細は、[セクション6.2.17「プラグブル認証」](#) を参照してください。

これらのいずれかのオプションとともにセキュアでない接続が使用されている場合、サーバーは次の警告を発行します：[Sending passwords in plain text without SSL/TLS is extremely insecure](#)

`START REPLICA | SLAVE ... UNTIL` では、グローバルトランザクション識別子 (GTID) で使用するための追加オプションが 2 つサポートされています ([セクション17.1.3「グローバルトランザクション識別子を使用したレプリケーション」](#) を参照)。これらの各オプションは、引数として 1 つ以上のグローバルトランザクション識別子のセット `gtid_set` を受け取ります (詳細は、[GTID セット](#) を参照してください)。

`thread_type` が指定されていない場合、`START REPLICA | SLAVE UNTIL SQL_BEFORE_GTIDS` は GTID が `gtid_set` にリストされている first トランザクションに到達するまで、レプリケーション SQL スレッドにトランザクションを処理させます。`START REPLICA | SLAVE UNTIL SQL_AFTER_GTIDS` では、`gtid_set` の last トランザクションが両方のスレッドによって処理されるまで、レプリケーションスレッドがすべてのトランザクションを処理します。つまり、`START REPLICA | SLAVE UNTIL SQL_BEFORE_GTIDS` は、レプリケーション SQL スレッドが `gtid_set` の最初の GTID に到達する前に発生したすべてのトランザクションを処理し、`START REPLICA | SLAVE UNTIL SQL_AFTER_GTIDS` は GTID がセットに含まれていないトランザクションが検出されるまで、レプリケーションスレッドがすべてのトランザクション (GTID が `gtid_set` で見つかったトランザクションを含む) を処理します。`SQL_BEFORE_GTIDS` と `SQL_AFTER_GTIDS` はそれぞれ、`SQL_THREAD` および `IO_THREAD` オプションをサポートしますが、`IO_THREAD` を一緒に使用しても現在は何の効果もありません。

たとえば、`START REPLICA | SLAVE SQL_THREAD UNTIL SQL_BEFORE_GTIDS = 3E11FA47-71CA-11E1-9E33-C80AA9429562:11-56` では、順序番号 11 のトランザクションが見つかるまで、`server_uuid` が `3E11FA47-71CA-11E1-9E33-C80AA9429562` であるソースから発生したすべてのトランザクションがレプリケーション SQL スレッドによって処理され、このトランザクションは処理されずに停止されます。つまり、シーケンス番号 10 を持つトランザクションまでのすべてのトランザクションが処理されます。一方、`START REPLICA | SLAVE SQL_THREAD UNTIL SQL_AFTER_GTIDS = 3E11FA47-71CA-11E1-9E33-C80AA9429562:11-56` を実行すると、レプリケーション SQL スレッドは、順序番号 11 から 56 を持つすべてのトランザクションを含め、ソースから指定されたすべてのトランザクションを取得し、追加のトランザクションを処理せずに停止します。つまり、順序番号 56 を持つトランザクションは、レプリケーション SQL スレッドによって最後にフェッチされたトランザクションになります。

マルチスレッドレプリカを使用する場合 `slave_preserve_commit_order=0` が設定されていると、次の場合にリレーログから実行された一連のトランザクションにギャップが生じる可能性があります：

- コーディネータスレッドの強制終了
- アプライヤスレッドでエラーが発生した後
- `mysqld` が予期せず停止

マルチスレッドレプリカワーカースレッドをリレーログにギャップがなくなるまでのみ実行してから停止するには、`START REPLICATION | SLAVE UNTIL SQL_AFTER_MTS_GAPS` ステートメントを使用します。このステートメントは `SQL_THREAD` オプションを受け取ることができますが、ステートメントの効果は変更されません。レプリケーション I/O スレッドには影響しません (`IO_THREAD` オプションとともに使用することはできません)。

リレーログから実行されるトランザクションのシーケンスにギャップがあるマルチスレッドレプリカで `START REPLICATION | SLAVE` を発行すると、警告が生成されます。このような状況で解決するには、`START REPLICATION | SLAVE UNTIL SQL_AFTER_MTS_GAPS` を使用してから、`RESET REPLICATION | SLAVE` を発行して残りのリレーログを削除します。詳しくは [セクション 17.5.1.34 「レプリケーションとトランザクションの非一貫性」](#) をご覧ください。

失敗したマルチスレッドレプリカをシングルスレッドモードに変更するには、次に示す順序で次の一連のステートメントを発行します:

```
START {REPLICATION | SLAVE} UNTIL SQL_AFTER_MTS_GAPS;
SET @@GLOBAL.slave_parallel_workers = 0;
START {REPLICATION | SLAVE} SQL_THREAD;
```

注記

`SHOW PROCESSLIST` の出力では、実行中の `START REPLICATION | SLAVE` ステートメントのテキスト全体 (使用されている `USER` または `PASSWORD` の値を含む) を表示できます。これは、実行中の `CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO` ステートメントのテキスト (`SOURCE_USER | MASTER_USER` または `SOURCE_PASSWORD | MASTER_PASSWORD` に使用する値を含む) にも当てはまります。

レプリケーション I/O スレッドとレプリケーション SQL スレッドの両方が起動した後、`START REPLICATION | SLAVE` はユーザーに確認を送信します。ただし、レプリケーション I/O スレッドがまだ接続されていない可能性があります。このため、`START REPLICATION | SLAVE` が成功すると `SHOW REPLICATION | SLAVE STATUS` に `Replica_SQL_Running=Yes` が表示されますが、`Replica_IO_Running=Yes` は保証されません (I/O スレッドがおよび接続済を実行している場合のみ `Replica_IO_Running=Yes` であるため)。詳細は、[セクション 13.7.7.35 「SHOW REPLICATION | SLAVE STATUS ステートメント」](#) および [セクション 17.1.7.1 「レプリケーションステータスの確認」](#) を参照してください。

レプリケーション SQL スレッドがソースバイナリログまたはレプリカリレーログ内の特定のポイントに到達するまでレプリカを起動して実行するように指定するために、`UNTIL` 句 (前述の文法では `until_option`) を追加できます。次のいずれかのオプションのペアを使用して、位置を指定します:

- バイナリログ用の `MASTER_LOG_POS` および `MASTER_LOG_FILE` (MySQL 8.0.22 へ)。
- バイナリログ用の `SOURCE_LOG_POS` および `SOURCE_LOG_FILE` (MySQL 8.0.23 から)。
- リレーログ用の `RELAY_LOG_POS` および `RELAY_LOG_FILE`。

圧縮トランザクションペイロードの場合、位置は圧縮された `Transaction_payload_event` に基づいている必要があります。SQL スレッドは、指定されたポイントに達すると停止します。このステートメントで `SQL_THREAD` オプションが指定されている場合、このオプションは SQL スレッドのみを開始します。それ以外の場合は、両方のレプリケーションスレッドを起動します。SQL スレッドが実行中である場合、`UNTIL` 句は無視され、警告が発行されます。`UNTIL` 句を `IO_THREAD` オプションとともに使用することはできません。

このセクションで前述したように、`START REPLICATION | SLAVE UNTIL` では、`SQL_BEFORE_GTIDS` または `SQL_AFTER_GTIDS` のいずれかのオプションを使用して、特定の GTID または GTID のセットに対して相対的な停止ポイントを指定することもできます。これらのオプションのいずれかを使用している場合は、`SQL_THREAD` または `IO_THREAD`、あるいはこれらの両方を指定できます。また、どちらも指定しないことも可能です。`SQL_THREAD` のみを指定した場合、レプリケーション SQL スレッドのみがステートメントの影響を受けます。`IO_THREAD` のみを指定した場合、レプリケーション I/O スレッドのみが影響を受けます。`SQL_THREAD` と `IO_THREAD` の両方が使用

されている場合、またはそのどちらも使用されていない場合は、SQL スレッドと I/O スレッドの両方がこのステートメントの影響を受けます。

UNTIL 句では、次のいずれかを指定する必要があります。

- ログファイル名とそのファイル内の位置の両方
- `SQL_BEFORE_GTIDS` または `SQL_AFTER_GTIDS` のいずれか
- `SQL_AFTER_MTS_GAPS`

ソースとリレーログオプションを混在させないでください。ログファイルオプションと GTID オプションを混在させないでください。

UNTIL 句は、`SQL_AFTER_MTS_GAPS` も使用している場合を除き、マルチスレッドレプリカではサポートされません。UNTIL が `SQL_AFTER_MTS_GAPS` のないマルチスレッドレプリカで使用されている場合、レプリカは、UNTIL 句で指定されたポイントに達するまで、レプリケーションのためにシングルスレッド (順次) モードで動作します。

UNTIL 条件は、後続の `STOP REPLICA | SLAVE` ステートメント、UNTIL 句を含まない `START REPLICA | SLAVE` ステートメント、またはサーバーの再起動によってリセットされます。

ログファイルと位置を指定する場合、このステートメントの影響を受けるのは SQL スレッドのみですが、`START REPLICA | SLAVE ... UNTIL` で `IO_THREAD` オプションを使用できます。このような場合、`IO_THREAD` オプションは無視されます。GTID オプション (`SQL_BEFORE_GTIDS` および `SQL_AFTER_GTIDS`) のいずれかを使用する場合、前述の制限は適用されません。GTID オプションでは、このセクションで前述した `SQL_THREAD` と `IO_THREAD` の両方がサポートされます。

UNTIL 句は、レプリケーションをデバッグしたり、レプリカがイベントをレプリケートしないようにする直前までレプリケーションを続行させる場合に役立ちます。たとえば、ソースで無関係な `DROP TABLE` ステートメントが実行された場合、UNTIL を使用してレプリカにその時点まで実行するが、それ以上実行しないように指示できます。イベントの内容を確認するには、ソースバイナリログまたはレプリカリレーログとともに `mysqlbinlog` を使用するか、`SHOW BINLOG EVENTS` ステートメントを使用します。

UNTIL を使用してレプリカプロセスのレプリケートされたクエリーをセクションに含める場合は、レプリカサーバーの起動時に SQL スレッドが実行されないように、`--skip-slave-start` オプションを指定してレプリカを起動することをお勧めします。このオプションはおそらく、予期しないサーバーの再起動によって忘れてしまうことがないように、コマンド行ではなく、オプションファイルで使うことが最善です。

`SHOW REPLICA | SLAVE STATUS` ステートメントには、UNTIL 条件の現在の値を表示する出力フィールドが含まれます。

13.4.2.8 START SLAVE | REPLICA ステートメント

```
START {SLAVE | REPLICA} [thread_types] [until_option] [connection_options] [channel_option]

thread_types:
  [thread_type [, thread_type] ... ]

thread_type:
  IO_THREAD | SQL_THREAD

until_option:
  UNTIL { {SQL_BEFORE_GTIDS | SQL_AFTER_GTIDS} = gtid_set
  | MASTER_LOG_FILE = 'log_name', MASTER_LOG_POS = log_pos
  | SOURCE_LOG_FILE = 'log_name', SOURCE_LOG_POS = log_pos
  | RELAY_LOG_FILE = 'log_name', RELAY_LOG_POS = log_pos
  | SQL_AFTER_MTS_GAPS }

connection_options:
  [USER='user_name'] [PASSWORD='user_pass'] [DEFAULT_AUTH='plugin_name'] [PLUGIN_DIR='plugin_dir']

channel_option:
  FOR CHANNEL channel

gtid_set:
```



```

uuid_set [, uuid_set] ...
| "
uuid_set:
  uuid:interval[:interval]...

uuid:
  hhhhhhhh-hhhh-hhhh-hhhhhhhhhhhh

h:
  [0-9,A-F]

interval:
  n[-n]

  (n >= 1)

```

レプリケーションスレッドを起動します。MySQL 8.0.22 からは、[START SLAVE](#) は非推奨であり、かわりにエイリアス [START REPLICA](#) を使用する必要があります。ステートメントは以前と同様に機能し、ステートメントおよびその出力に使用される用語のみが変更されています。どちらのバージョンのステートメントも、使用時に同じステータス変数を更新します。ステートメントの説明は、[START REPLICA](#) のドキュメントを参照してください。

13.4.2.9 STOP REPLICA | SLAVE ステートメント

```

STOP {REPLICA | SLAVE} [thread_types] [channel_option]

thread_types:
  [thread_type [, thread_type] ... ]

thread_type: IO_THREAD | SQL_THREAD

channel_option:
  FOR CHANNEL channel

```

レプリケーションスレッドを停止します。MySQL 8.0.22 から、[STOP SLAVE](#) のかわりに [STOP REPLICA](#) を使用します。これは非推奨になりました。MySQL 8.0.22 より前のリリースでは、[STOP SLAVE](#) を使用します。

[STOP REPLICA | SLAVE](#) には、[REPLICATION_SLAVE_ADMIN](#) 権限 (または非推奨の [SUPER](#) 権限) が必要です。推奨されるベストプラクティスは、レプリカサーバーを停止する前にレプリカで [STOP REPLICA | SLAVE](#) を実行することです (詳細は、[セクション5.1.19「サーバーの停止プロセス」](#) を参照してください)。

[START REPLICA | SLAVE](#) と同様に、このステートメントを [IO_THREAD](#) および [SQL_THREAD](#) オプションとともに使用して、停止するレプリケーションスレッドの名前を指定できます。グループレプリケーションアプライヤチャンネル ([group_replication_applier](#)) にはレプリケーション I/O スレッドがなく、レプリケーション SQL スレッドのみがあることに注意してください。したがって、[SQL_THREAD](#) オプションを使用すると、このチャンネルは完全に停止します。

[STOP REPLICA | SLAVE](#) では、進行中のトランザクションが暗黙的にコミットされます。[セクション13.3.3「暗黙的なコミットを発生させるステートメント」](#) を参照してください。

このステートメントを発行する前に、[gtid_next](#) を [AUTOMATIC](#) に設定する必要があります。

[rpl_stop_slave_timeout](#) システム変数を設定することで、[STOP REPLICA | SLAVE](#) がタイムアウトするまでの待機時間を制御できます。これを使用すると、レプリカへの異なるクライアント接続を使用する [STOP REPLICA | SLAVE](#) と他の SQL ステートメントの間のデッドロックを回避できます。タイムアウト値に達すると、発行元クライアントはエラーメッセージを返して待機を停止しますが、[STOP REPLICA | SLAVE](#) 命令は有効なままです。レプリケーションスレッドがビジー状態でなくなると、[STOP REPLICA | SLAVE](#) ステートメントが実行され、レプリカが停止します。

一部の [CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO](#) ステートメントは、レプリケーションスレッドの状態に応じて、レプリカの実行中に許可されます。ただし、このような場合、[CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO](#) ステートメントを実行する前の [STOP REPLICA | SLAVE](#) の使用は引き続きサポートされています。詳細は、[セクション13.4.2.3「CHANGE REPLICATION SOURCE TO ステートメント」](#)、[セクション13.4.2.1「CHANGE MASTER TO ステートメント」](#) および [セクション17.4.8「ファイルオーバー中のソースの切替え」](#) を参照してください。

オプションの `FOR CHANNEL channel` 句を使用すると、ステートメントが適用されるレプリケーションチャンネルの名前を指定できます。 `FOR CHANNEL channel` 句を指定すると、`STOP REPLICATION | SLAVE` ステートメントが特定のレプリケーションチャンネルに適用されます。チャンネルが指定されておらず、追加のチャンネルが存在しない場合、ステートメントはデフォルトチャンネルに適用されます。複数のチャンネルを使用しているときに `STOP REPLICATION | SLAVE` ステートメントがチャンネルを指定しない場合、このステートメントはすべてのチャンネルの指定されたスレッドを停止します。このステートメントは、`group_replication_recovery` チャンネルでは使用できません。詳しくは [セクション 17.2.2 「レプリケーションチャンネル」](#) をご覧ください。

レプリカがマルチスレッド化されている場合 (`slave_parallel_workers` はゼロ以外の値)、リレーログから実行された一連のトランザクションのギャップはワーカースレッドの停止の一環として閉じられます。 `STOP REPLICATION | SLAVE` ステートメントの実行中にレプリカが予期せず (ワーカースレッドのエラーや `KILL` を発行する別のスレッドが原因など) 停止した場合、リレーログから実行されたトランザクションの順序に一貫性がなくなる可能性があります。詳しくは [セクション 17.5.1.34 「レプリケーションとトランザクションの非一貫性」](#) をご覧ください。

ソースが行ベースのバイナリロギング形式を使用している場合、非トランザクションストレージエンジンを使用するテーブルをレプリケートするときは、レプリカサーバーをシャットダウンする前に、レプリカで `STOP REPLICATION | SLAVE` または `STOP REPLICATION | SLAVE SQL_THREAD` を実行するようにしてください。現在のレプリケーションイベントグループが 1 つ以上の非トランザクションテーブルを変更した場合、レプリケーション SQL スレッドに対して `KILL QUERY` または `KILL CONNECTION` ステートメントを発行しないがぎり、`STOP REPLICATION | SLAVE` はイベントグループが完了するまで最大 60 秒待機します。タイムアウト後もイベントグループが不完全なままである場合は、エラーメッセージが記録されます。

ソースがステートメントベースのバイナリロギング形式を使用している場合、開いている一時テーブルがある間にソースを変更することは安全でない可能性があります。これは、一時テーブルのステートメントベースレプリケーションが推奨されない理由の 1 つです。レプリカに一時テーブルがあるかどうかは、`Slave_open_temp_tables` の値で確認できます。ステートメントベースレプリケーションを使用する場合は、`CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO` を実行する前にこの値を 0 にする必要があります。レプリカで開いている一時テーブルがある場合、`STOP REPLICATION | SLAVE` の発行後に `CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO` ステートメントを発行すると、`ER_WARN_OPEN_TEMP_TABLES_MUST_BE_ZERO` 警告が表示されます。

13.4.2.10 STOP SLAVE | REPLICATION ステートメント

```
STOP {SLAVE | REPLICATION} [thread_types] [channel_option]
```

```
thread_types:
  [thread_type [, thread_type] ... ]
```

```
thread_type: IO_THREAD | SQL_THREAD
```

```
channel_option:
  FOR CHANNEL channel
```

レプリケーションスレッドを停止します。MySQL 8.0.22 からは、`STOP SLAVE` は非推奨であり、かわりにエイリアス `STOP REPLICATION` を使用する必要があります。ステートメントは以前と同様に機能し、ステートメントおよびその出力に使用される用語のみが変更されています。どちらのバージョンのステートメントも、使用時に同じステータス変数を更新します。ステートメントの説明は、`STOP REPLICATION` のドキュメントを参照してください。

13.4.2.11 ソースリストを構成する関数

標準ソースからレプリカレプリケーションの場合は MySQL 8.0.22 から、Group Replication の場合は MySQL 8.0.23 から使用可能な次の機能を使用すると、レプリケーションチャンネルのソースリストに対してレプリケーションソースサーバーを追加および削除できます。非同期接続フェイルオーバーメカニズムは、レプリカからソースへの既存の接続が失敗した後、適切なリストから新しいソースへの非同期 (ソースからレプリカへの) レプリケーション接続を自動的に確立します。MySQL 8.0.23 からは、現在接続されているソースの重み付け優先度がグループ内で最も高い場合にも接続が変更されます。管理対象グループの一部として定義されている Group Replication ソースサーバーの場合、現在接続されているソースがグループから離れているか、その大部分ではなくなった場合、接続は別のグループメンバーにもフェイルオーバーされます。メカニズムの詳細は、[セクション 17.4.9 「非同期接続フェイルオーバーによるソースの切替え」](#) を参照してください。

ソースリストは `mysql.replication_asynchronous_connection_failover` および `mysql.replication_asynchronous_connection_failover_managed` テーブルに格納され、「パフォーマンススキーマ」テーブル `replication_asynchronous_connection_failover` で表示できます。

- `asynchronous_connection_failover_add_source()`

レプリケーションソースサーバーの構成情報をレプリケーションチャンネルのソースリストに追加します。

構文:

```
asynchronous_connection_failover_add_source(channel, host, port, network_namespace, weight)
```

引数:

- **channel**: このレプリケーションソースサーバーがソースリストの一部であるレプリケーションチャンネル。
- **host**: このレプリケーションソースサーバーのホスト名。
- **port**: このレプリケーションソースサーバーのポート番号。
- **network_namespace**: このレプリケーションソースサーバーのネットワークネームスペース。このパラメータは将来の使用のために予約されているため、空の文字列を指定してください。
- **weight**: レプリケーションチャンネルソースリスト内のこのレプリケーションソースサーバーの優先度。優先度は 1 から 100 で、100 が最高、50 がデフォルトです。非同期接続フェイルオーバーメカニズムがアクティブ化されると、チャンネルのソースリストにリストされている代替ソースの中で優先度が最も高いソースが最初の接続試行に選択されます。この試行が機能しない場合、レプリカはリストされているすべてのソースを優先度の降順で試行し、優先度の最も高いソースから再開します。複数のソースの優先度が同じ場合、レプリカはそれらをランダムに順序付けします。MySQL 8.0.23 では、現在接続されているソースがグループ内で最も重み付けされていない場合、非同期接続フェイルオーバーメカニズムがアクティブ化されます。

戻り値:

成功したかどうかなど、操作の結果を含む文字列。

例:

```
SELECT asynchronous_connection_failover_add_source('channel2', '127.0.0.1', 3310, '', 80);
+-----+
| asynchronous_connection_failover_add_source('channel2', '127.0.0.1', 3310, '', 80) |
+-----+
| Source configuration details successfully inserted. |
+-----+
```

詳細は、[セクション 17.4.9 「非同期接続フェイルオーバーによるソースの切替え」](#) を参照してください。

- `asynchronous_connection_failover_delete_source()`

レプリケーションチャンネルのソースリストからレプリケーションソースサーバーの構成情報を削除します。

構文:

```
asynchronous_connection_failover_delete_source(channel, host, port, network_namespace)
```

引数:

- **channel**: このレプリケーションソースサーバーがソースリストの一部であったレプリケーションチャンネル。
- **host**: このレプリケーションソースサーバーのホスト名。
- **port**: このレプリケーションソースサーバーのポート番号。
- **network_namespace**: このレプリケーションソースサーバーのネットワークネームスペース。このパラメータは将来の使用のために予約されているため、空の文字列を指定してください。

戻り値:

成功したかどうかなど、操作の結果を含む文字列。

例:

```
SELECT asynchronous_connection_failover_delete_source('channel2', '127.0.0.1', 3310, "");
+-----+
| asynchronous_connection_failover_delete_source('channel2', '127.0.0.1', 3310, "") |
+-----+
| Source configuration details successfully deleted. |
+-----+
```

詳細は、[セクション17.4.9「非同期接続フェイルオーバーによるソースの切替え」](#)を参照してください。

- `asynchronous_connection_failover_add_managed()`

管理対象グループ (Group Replication グループメンバー) の一部であるレプリケーションソースサーバーの構成情報を、レプリケーションチャンネルのソースリストに追加します。追加する必要があるグループメンバーは 1 つだけです。レプリカは、現在のグループメンバーシップから残りを自動的に追加し、メンバーシップの変更に応じてソースリストを更新します。

構文:

```
asynchronous_connection_failover_add_managed(channel, managed_type, managed_name, host, port, network_namespace, primary_weight, secondary_weight)
```

引数:

- `channel`: このレプリケーションソースサーバーがソースリストの一部であるレプリケーションチャンネル。
- `managed_type`: 非同期接続フェイルオーバーメカニズムがこのサーバーに提供する必要がある管理対象サービスのタイプ。現在受け入れられている値は `GroupReplication` のみです。
- `managed_name`: サーバーが属する管理対象グループの識別子。 `GroupReplication` 管理サービスの場合、 `identifier` は `group_replication_group_name` システム変数の値です。
- `host`: このレプリケーションソースサーバーのホスト名。
- `port`: このレプリケーションソースサーバーのポート番号。
- `network_namespace`: このレプリケーションソースサーバーのネットワークネームスペース。このパラメータは将来の使用のために予約されているため、空の文字列を指定してください。
- `primary_weight`: 管理対象グループのプライマリとして機能している場合の、レプリケーションチャンネルソースリスト内のこのレプリケーションソースサーバーの優先度。重みは 1~100 で、100 が最高です。プライマリの場合、80 が適切な重みです。非同期接続フェイルオーバーメカニズムは、現在接続されているソースがグループ内で最も重み付けされていない場合にアクティブになります。プライマリに高い重みを与え、セカンダリに低い重みを与えるように管理対象グループを設定した場合、プライマリが変更されると、その重みが増加し、レプリカは接続を介して変更されます。
- `secondary_weight`: このレプリケーションソースサーバーが管理対象グループのセカンダリとして機能している場合の、レプリケーションチャンネルソースリスト内での優先度。重みは 1~100 で、100 が最高です。セカンダリの場合、60 が適切な重みです。

戻り値:

成功したかどうかなど、操作の結果を含む文字列。

例:

```
SELECT asynchronous_connection_failover_add_managed('channel2', 'GroupReplication', 'aaaaaaaa-aaaa-aaaa-aaaaaaaaaaaa', '127.0.0.1', 3310, '', 80, 60);
+-----+
| asynchronous_connection_failover_add_source('channel2', 'GroupReplication', 'aaaaaaaa-aaaa-aaaa-aaaaaaaaaaaa', '127.0.0.1', 3310, '', 80, 60) |
+-----+
| Source managed configuration details successfully inserted. |
+-----+
```

詳細は、[セクション17.4.9「非同期接続フェイルオーバーによるソースの切替え」](#)を参照してください。

- `asynchronous_connection_failover_delete_managed()`

レプリケーションチャンネルのソースリストから管理対象グループ全体を削除します。この UDF を使用すると、管理対象グループで定義されたすべてのレプリケーションソースサーバーがチャンネルソースリストから削除されます。

構文:

```
asynchronous_connection_failover_delete_managed(channel, managed_name)
```

引数:

- `channel`: このレプリケーションソースサーバーがソースリストの一部であったレプリケーションチャンネル。
- `managed_name`: サーバーが属する管理対象グループの識別子。GroupReplication 管理サービスの場合、`identifier` は `group_replication_group_name` システム変数の値です。

戻り値:

成功したかどうかなど、操作の結果を含む文字列。

例:

```
SELECT asynchronous_connection_failover_delete_managed('channel2', 'aaaaaaaa-aaaa-aaaa-aaaaaaaaaaaa');
+-----+
| asynchronous_connection_failover_delete_managed('channel2', 'aaaaaaaa-aaaa-aaaa-aaaaaaaaaaaa') |
+-----+
| Source managed configuration details successfully deleted. |
+-----+
```

詳細は、[セクション17.4.9「非同期接続フェイルオーバーによるソースの切替え」](#)を参照してください。

13.4.3 グループレプリケーションを制御するための SQL ステートメント

このセクションでは、グループレプリケーションの制御に使用されるステートメントについて説明します。

13.4.3.1 START GROUP_REPLICATION ステートメント

```
START GROUP_REPLICATION
  [USER='user_name']
  [, PASSWORD='user_pass']
  [, DEFAULT_AUTH='plugin_name']
```

グループレプリケーションを開始します。このステートメントには、`GROUP_REPLICATION_ADMIN` 権限 (または非推奨の `SUPER` 権限) が必要です。 `super_read_only=ON` が設定されており、メンバーがプライマリとして参加する必要がある場合、グループレプリケーションが正常に開始されると、`super_read_only` は `OFF` に設定されます。

MySQL 8.0.21 から、次のように `USER`、`PASSWORD` および `DEFAULT_AUTH` オプションを使用して、`START GROUP_REPLICATION` ステートメントで分散リカバリのユーザー資格証明を指定できます:

- `USER`: 分散リカバリのレプリケーションユーザー。このアカウントを設定する手順は、[セクション18.2.1.3「分散リカバリのユーザー資格証明」](#)を参照してください。 `PASSWORD` が指定されている場合、空または `null` の文字列を指定したり、`USER` オプションを省略することはできません。
- `PASSWORD`: レプリケーションユーザーアカウントのパスワード。パスワードは暗号化できませんが、クエリーログでマスクされます。
- `DEFAULT_AUTH`: レプリケーションユーザーアカウントに使用される認証プラグインの名前。このオプションを指定しない場合、MySQL ネイティブ認証 (`mysql_native_password` プラグイン) が想定されます。このオプションはサーバーへのヒントとして機能し、分散リカバリのドナーは、そのサーバー上のユーザーアカウントに別のプラグインが関連付けられている場合、それをオーバーライドします。MySQL 8 でユーザーアカウントを作成するときにデフォルトで使用される認証プラグインは、キャッシュ SHA-2 認証プラグイン (`caching_sha2_password`) です。認証プラグインの詳細は、[セクション6.2.17「プラグイン認証」](#)を参照してください。

これらの資格証明は、`group_replication_recovery` チャンネルでの分散リカバリに使用されます。 `START GROUP_REPLICATION` でユーザー資格証明を指定すると、資格証明はメモリーにのみ保存され、`STOP`

`GROUP_REPLICATION` ステートメントまたはサーバーの停止によって削除されます。資格証明を再度指定するには、`START GROUP_REPLICATION` ステートメントを発行する必要があります。したがって、この方法は、`group_replication_start_on_boot` システム変数で指定されているように、サーバー起動時に Group Replication を自動的に起動する方法とは互換性がありません。

`START GROUP_REPLICATION` で指定されたユーザー資格証明は、`CHANGE REPLICATION SOURCE TO` ステートメント (MySQL 8.0.23) または `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 より前) を使用して `group_replication_recovery` チャンネルに設定されたユーザー資格証明よりも優先されます。これらのステートメントを使用して設定されたユーザー資格証明はレプリケーションメタデータリポジトリに格納され、`group_replication_start_on_boot` システム変数が `ON` に設定されている場合の自動起動など、ユーザー資格証明なしで `START GROUP_REPLICATION` が指定されている場合に使用されます。`START GROUP_REPLICATION` でユーザー資格証明を指定するセキュリティ上の利点を得るには、[セクション18.5.3「分散リカバリ接続の保護」](#)の手順に従って、`group_replication_start_on_boot` が `OFF` (デフォルトは `ON`) に設定されていることを確認し、`group_replication_recovery` チャンネルに以前に設定されたユーザー資格証明をクリアします。

13.4.3.2 STOP GROUP_REPLICATION ステートメント

STOP GROUP_REPLICATION

グループレプリケーションを停止します。このステートメントには、`GROUP_REPLICATION_ADMIN` 権限 (または非推奨の `SUPER` 権限) が必要です。`STOP GROUP_REPLICATION` を発行するとすぐに、メンバーは `super_read_only=ON` に設定され、グループレプリケーションの停止中にメンバーへの書き込みができなくなります。メンバーで実行されている他のレプリケーションチャンネルも停止されます。このメンバーでグループレプリケーションを開始するときに `START GROUP_REPLICATION` ステートメントで指定したユーザー資格証明はメモリーから削除されるため、グループレプリケーションを再度開始するときに指定する必要があります。

警告

このステートメントは、グループからサーバーインスタンスを削除するため、特に注意して使用してください。つまり、グループレプリケーションの一貫性保証メカニズムによって保護されなくなります。完全に安全にするには、失効した読取りの可能性を回避するために、このステートメントを発行する前にアプリケーションがインスタンスに接続できないようにします。

13.4.3.3 グループレプリケーションプライマリを構成する機能

次の関数を使用すると、単一プライマリレプリケーショングループのどのメンバーがプライマリであるかを構成できます。

- `group_replication_set_as_primary()`

グループの特定のメンバーを新しいプライマリとして指名し、選択プロセスを上書きします。新しいプライマリにするメンバーの `server_uuid` である `member_uuid` を渡します。単一プライマリモードで実行されているレプリケーショングループのメンバーに対して発行する必要があります。

構文:

```
STRING group_replication_set_as_primary(member_uuid)
```

戻り値:

成功したかどうかなど、操作の結果を含む文字列。

例:

```
SELECT group_replication_set_as_primary(member_uuid)
```

詳細は、[セクション18.4.1.1「グループプライマリメンバーの変更」](#)を参照してください

13.4.3.4 グループレプリケーションモードを構成する関数

次の関数を使用すると、レプリケーショングループが実行されているモード (単一プライマリモードまたはマルチプライマリモード) を制御できます。

- `group_replication_switch_to_single_primary_mode()`

グループレプリケーションを停止せずに、マルチプライマリモードで実行されているグループを単一プライマリモードに変更します。マルチプライマリモードで実行されているレプリケーショングループのメンバーに対して発行する必要があります。シングルプライマリモードに変更すると、シングルプライマリモード (`group_replication_enforce_update_everywhere_checks=OFF`) での必要に応じて、すべてのグループメンバーで厳密な整合性チェックも無効になります。

構文:

```
STRING group_replication_switch_to_single_primary_mode([str])
```

引数:

- `str`: 新しい単一のプライマリになるグループのメンバーの UUID を含む文字列。グループの他のメンバーはセカンダリになります。

戻り値:

成功したかどうかなど、操作の結果を含む文字列。

例:

```
SELECT group_replication_switch_to_single_primary_mode(member_uuid);
```

詳細は、[セクション18.4.1.2「グループモードの変更」](#)を参照してください

- `group_replication_switch_to_multi_primary_mode()`

シングルプライマリモードで実行されているグループをマルチプライマリモードに変更します。単一プライマリモードで実行されているレプリケーショングループのメンバーに対して発行する必要があります。

構文:

```
STRING group_replication_switch_to_multi_primary_mode()
```

この関数にはパラメータがありません。

戻り値:

成功したかどうかなど、操作の結果を含む文字列。

例:

```
SELECT group_replication_switch_to_multi_primary_mode()
```

グループに属するすべてのメンバーがプライマリになります。

詳細は、[セクション18.4.1.2「グループモードの変更」](#)を参照してください

13.4.3.5 グループの最大コンセンサスインスタンスを検査および構成する関数

次の機能を使用すると、グループがパラレルに実行できるコンセンサスインスタンスの最大数を検査および構成できます。

- `group_replication_get_write_concurrency()`

グループがパラレルに実行できるコンセンサスインスタンスの最大数を確認します。

構文:

```
INT group_replication_get_write_concurrency()
```

この関数にはパラメータがありません。

戻り値:

グループに現在設定されているコンセンサスインスタンスの最大数。

例:

```
SELECT group_replication_get_write_concurrency()
```

詳細は、[セクション18.4.1.3「グループレプリケーショングループ書き込みコンセンサスの使用」](#)を参照してください。

- `group_replication_set_write_concurrency()`

グループがパラレルに実行できるコンセンサスインスタンスの最大数を構成します。この UDF を使用するには、`GROUP_REPLICATION_ADMIN` 権限が必要です。

構文:

```
STRING group_replication_set_write_concurrency(instances)
```

引数:

- `members`: グループがパラレルに実行できるコンセンサスインスタンスの最大数を設定します。デフォルト値は 10 で、有効な値は 10 から 200 の範囲の整数です。

戻り値:

文字列としての結果のエラー。

例:

```
SELECT group_replication_set_write_concurrency(instances);
```

詳細は、[セクション18.4.1.3「グループレプリケーショングループ書き込みコンセンサスの使用」](#)を参照してください。

13.4.3.6 グループレプリケーション通信プロトコルのバージョンを検査および設定する関数

次の機能を使用すると、レプリケーショングループで使用されるグループレプリケーション通信プロトコルのバージョンを検査および構成できます。

- `group_replication_get_communication_protocol()`

グループで現在使用されている Group Replication 通信プロトコルのバージョンを調べます。

構文:

```
STRING group_replication_get_communication_protocol()
```

この関数にはパラメータがありません。

戻り値:

このグループに参加し、グループ通信プロトコルを使用できる最も古い MySQL Server バージョン。MySQL 5.7.14 のバージョンではメッセージを圧縮でき、MySQL 8.0.16 のバージョンではメッセージを断片化することもできます。`group_replication_get_communication_protocol()` UDF は、グループがサポートする MySQL の最小バージョンを返します。これは、`group_replication_set_communication_protocol()` UDF に渡されたバージョン番号、および UDF を使用するメンバーにインストールされている MySQL Server バージョンとは異なる場合があります。

このサーバーインスタンスがレプリケーショングループに属していないためにプロトコルを検査できない場合は、文字列としてエラーが返されます。

例:

```
SELECT group_replication_get_communication_protocol();
+-----+
| group_replication_get_communication_protocol() |
```

```
+-----+
| 8.0.16 |
+-----+
```

詳細は、[セクション18.4.1.4「グループ通信プロトコルバージョンの設定」](#)を参照してください。

- `group_replication_set_communication_protocol()`

グループの Group Replication 通信プロトコルバージョンをダウングレードして、以前のリリースのメンバーがグループに参加できるようにするか、すべてのメンバーで MySQL Server をアップグレードした後に Group Replication 通信プロトコルバージョンをアップグレードします。この UDF を使用するには `GROUP_REPLICATION_ADMIN` 権限が必要です。また、ステートメントを発行するときは、大部分を失うことなく、既存のすべてのグループメンバーがオンラインである必要があります。

注記

MySQL InnoDB クラスタの場合、AdminAPI 操作を使用してクラスタトポロジが変更されるたびに、通信プロトコルバージョンが自動的に管理されます。InnoDB クラスタに対してこれらの UDF を自分で使用する必要はありません。

構文:

```
STRING group_replication_set_communication_protocol(version)
```

引数:

- `version`: ダウングレードの場合は、インストールされているサーバーバージョンが最も古い見込みグループメンバーの MySQL Server バージョンを指定します。この場合、可能であれば、このコマンドによってグループはそのサーバーバージョンと互換性のある通信プロトコルにフォールバックされます。指定できるサーバーの最小バージョンは MySQL 5.7.14 です。アップグレードの場合は、既存のグループメンバーがアップグレードされた新しい MySQL Server バージョンを指定します。

戻り値:

成功したかどうかなど、操作の結果を含む文字列。

例:

```
SELECT group_replication_set_communication_protocol("5.7.25");
```

詳細は、[セクション18.4.1.4「グループ通信プロトコルバージョンの設定」](#)を参照してください。

13.5 プリペアドステートメント

MySQL 8.0 は、サーバー側の準備済みステートメントをサポートしています。このサポートは、効率的なクライアント/サーバーバイナリプロトコルを利用します。パラメータ値のためのプレースホルダを含む準備済みステートメントの使用には、次の利点があります。

- ステートメントを実行のたびに解析するためのオーバーヘッドが少なくなります。通常、データベースアプリケーションは、クエリーや削除の場合の `WHERE`、更新の場合の `SET`、挿入の場合の `VALUES` などの句でリテラルまたは変数値しか変更されていない、ほぼ同一の大量のステートメントを処理します。
- SQL インジェクション攻撃からの保護。パラメータ値には、エスケープされていない SQL 引用符および区切り文字を含めることができます。

次の各セクションでは、プリペアドステートメントの特性の概要について説明します:

- [アプリケーションプログラムでの準備済みステートメント](#)
- [SQL スクリプトでの準備済みステートメント](#)
- [PREPARE、EXECUTE、および DEALLOCATE PREPARE ステートメント](#)
- [プリペアドステートメントで許可される SQL 構文](#)

アプリケーションプログラムでの準備済みステートメント

サーバー側のプリバードステートメントは、[MySQL C API client library](#) for C プログラム、[MySQL Connector/J](#) for Java プログラム、[MySQL Connector/NET](#) for .NET テクノロジーを使用するプログラムなど、クライアントプログラミングインタフェースを介して使用できます。たとえば、C API は、その準備済みステートメント API を構成する一連の関数呼び出しを提供しています。[C API Prepared Statement Interface](#)を参照してください。他の言語インタフェースでは、PHP 5.0 以上で使用可能な「[mysqli 拡張機能](#)」などの C クライアントライブラリにリンクすることで、バイナリプロトコルを使用するプリバードステートメントをサポートできます。

SQL スクリプトでの準備済みステートメント

準備済みステートメントへの代替 SQL インタフェースを使用できます。このインタフェースは、準備済みステートメント API 経由でのバイナリプロトコルの使用ほど効率的ではありませんが、SQL レベルで直接使用できるためプログラミングが必要ありません。

- 使用できるプログラミングインタフェースが存在しない場合でも使用できます。
- `mysql` クライアントプログラムなどの、サーバーに SQL ステートメントを送信して実行させることのできる任意のプログラムから使用できます。
- これは、クライアントが古いバージョンのクライアントライブラリを使用している場合でも使用できます。

準備済みステートメントのための SQL 構文は、次のような状況で使用されるように考慮されています。

- 準備済みステートメントのコーディングの前に、それがアプリケーションでどのように動作するかをテストする場合。
- サポートしているプログラミング API にアクセスできないときに準備済みステートメントを使用する場合。
- 準備済みステートメントに関するアプリケーションの問題を対話的にトラブルシューティングする場合。
- バグレポートを提出できるように、準備済みステートメントに関する問題を再現するテストケースを作成する場合。

PREPARE、EXECUTE、および DEALLOCATE PREPARE ステートメント

準備済みステートメントのための SQL 構文は、次の 3 つの SQL ステートメントに基づいています。

- **PREPARE** は、ステートメントを実行のために準備します ([セクション 13.5.1 「PREPARE ステートメント」](#)を参照してください)。
- **EXECUTE** は、準備済みステートメントを実行します ([セクション 13.5.2 「EXECUTE ステートメント」](#)を参照してください)。
- **DEALLOCATE PREPARE** は、準備済みステートメントを解放します ([セクション 13.5.3 「DEALLOCATE PREPARE ステートメント」](#)を参照してください)。

次の例は、2 辺の長さが与えられた三角形の斜辺を計算するステートメントを準備するための 2 つの同等の方法を示しています。

最初の例は、文字列リテラルを使用してステートメントのテキストを指定することによって準備済みステートメントを作成する方法を示しています。

```
mysql> PREPARE stmt1 FROM 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';
mysql> SET @a = 3;
mysql> SET @b = 4;
mysql> EXECUTE stmt1 USING @a, @b;
+-----+
| hypotenuse |
+-----+
|      5 |
+-----+
mysql> DEALLOCATE PREPARE stmt1;
```

2 番目の例も同様ですが、ステートメントのテキストをユーザー変数として指定します。

```
mysql> SET @s = 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';
mysql> PREPARE stmt2 FROM @s;
mysql> SET @a = 6;
mysql> SET @b = 8;
mysql> EXECUTE stmt2 USING @a, @b;
+-----+
| hypotenuse |
+-----+
|      10 |
+-----+
mysql> DEALLOCATE PREPARE stmt2;
```

次の追加の例は、クエリーを実行する対象となるテーブルの名前をユーザー変数として格納することによって、実行時にそのテーブルを選択する方法を示しています。

```
mysql> USE test;
mysql> CREATE TABLE t1 (a INT NOT NULL);
mysql> INSERT INTO t1 VALUES (4), (8), (11), (32), (80);

mysql> SET @table = 't1';
mysql> SET @s = CONCAT('SELECT * FROM ', @table);

mysql> PREPARE stmt3 FROM @s;
mysql> EXECUTE stmt3;
+----+
| a |
+----+
| 4 |
| 8 |
| 11 |
| 32 |
| 80 |
+----+

mysql> DEALLOCATE PREPARE stmt3;
```

準備済みステートメントは、そのステートメントが作成されたセッションに固有です。以前に作成された準備済みステートメントを解放せずにセッションを終了した場合、そのステートメントはサーバーによって自動的に解放されません。

準備済みステートメントはまた、セッションに対してグローバルでもあります。ストアドルーチン内で準備済みステートメントを作成した場合、そのステートメントはストアドルーチンが終了しても解放されません。

同時に作成される準備済みステートメントが多くなりすぎないようにするには、`max_prepared_stmt_count` システム変数を設定します。準備済みステートメントの使用を回避するには、この値を 0 に設定します。

プリペアドステートメントで許可される SQL 構文

次の SQL ステートメントは、準備済みステートメントとして使用できます。

```
ALTER TABLE
ALTER USER
ANALYZE TABLE
CACHE INDEX
CALL
CHANGE MASTER
CHECKSUM {TABLE | TABLES}
COMMIT
{CREATE | DROP} INDEX
{CREATE | RENAME | DROP} DATABASE
{CREATE | DROP} TABLE
{CREATE | RENAME | DROP} USER
{CREATE | DROP} VIEW
DELETE
DO
FLUSH {TABLE | TABLES | TABLES WITH READ LOCK | HOSTS | PRIVILEGES
      | LOGS | STATUS | MASTER | SLAVE | USER_RESOURCES}
GRANT
INSERT
INSTALL PLUGIN
KILL
```

```
LOAD INDEX INTO CACHE
OPTIMIZE TABLE
RENAME TABLE
REPAIR TABLE
REPLACE
RESET {MASTER | SLAVE}
REVOKE
SELECT
SET
SHOW BINLOG EVENTS
SHOW CREATE {PROCEDURE | FUNCTION | EVENT | TABLE | VIEW}
SHOW {MASTER | BINARY} LOGS
SHOW {MASTER | SLAVE} STATUS
SLAVE {START | STOP}
TRUNCATE TABLE
UNINSTALL PLUGIN
UPDATE
```

その他のステートメントはサポートされていません。

診断ステートメントを準備できないことを示す SQL 標準に準拠するために、MySQL では準備済みのステートメントとして次のものはサポートされていません:

- [SHOW WARNINGS](#), [SHOW COUNT\(*\) WARNINGS](#)
- [SHOW ERRORS](#), [SHOW COUNT\(*\) ERRORS](#)
- `warning_count` または `error_count` システム変数への参照を含むステートメント。

通常、SQL 準備済みステートメントで許可されていないステートメントは、ストアードプログラムでも許可されません。例外については、[セクション25.8「ストアードプログラムの制約」](#)に示されています。

プリペアドステートメントによって参照されているテーブルやビューのメタデータの変更が検出され、それが次に実行されるたびに、ステートメントが自動再準備されます。詳細については、[セクション8.10.3「プリペアドステートメントおよびストアードプログラムのキャッシュ」](#)を参照してください。

準備済みステートメントを使用する場合は、`LIMIT` 句の引数にプレースホルダを使用できます。[セクション13.2.10「SELECT ステートメント」](#)を参照してください。

`PREPARE` および `EXECUTE` とともに使用される準備済み `CALL` ステートメントでは、`OUT` および `INOUT` パラメータに対するプレースホルダのサポートが MySQL 8.0 から使用できます。例および以前のバージョンでの回避方法については、[セクション13.2.1「CALL ステートメント」](#)を参照してください。`IN` パラメータには、バージョンには関係なくプレースホルダを使用できます。

準備済みステートメントのための SQL 構文は、ネストされた方法では使用できません。つまり、`PREPARE` に渡されるステートメント自体を、`PREPARE`、`EXECUTE`、または `DEALLOCATE PREPARE` ステートメントにすることはできません。

準備済みステートメントのための SQL 構文は、準備済みステートメント API 呼び出しの使用とは異なります。たとえば、`mysql_stmt_prepare()` C API 関数を使用して、`PREPARE`、`EXECUTE`、または `DEALLOCATE PREPARE` ステートメントを準備することはできません。

準備済みステートメントのための SQL 構文はストアードプロシージャ内で使用できますが、ストアードファンクションまたはトリガー内では使用できません。ただし、`PREPARE` と `EXECUTE` で準備および実行される動的なステートメントにはカーソルを使用できません。カーソルのステートメントはカーソル作成時にチェックされるため、そのステートメントを動的にすることはできません。

準備済みステートメントのための SQL 構文は、マルチステートメント (つまり、`;`文字で区切られた 1 つの文字列内の複数のステートメント) をサポートしていません。

`CALL` SQL ステートメントを使用して、準備済みステートメントを含むストアードプロシージャを実行する C プログラムを記述するには、`CLIENT_MULTI_RESULTS` フラグが有効になっている必要があります。これは、各 `CALL` によって、プロシージャ内で実行されるステートメントによって返される可能性のある結果セットに加えて、呼び出しステータスを示すための結果が返されるためです。

`CLIENT_MULTI_RESULTS` は、`mysql_real_connect()` を呼び出すときに、`CLIENT_MULTI_RESULTS` フラグ自体を渡すことによって明示的に、または `CLIENT_MULTI_STATEMENTS` を渡すことによって暗黙的に有効にする (これに

よって `CLIENT_MULTI_RESULTS` も有効になります) ことができます。詳細は、[セクション13.2.1「CALL ステートメント」](#)を参照してください。

13.5.1 PREPARE ステートメント

```
PREPARE stmt_name FROM preparable_stmt
```

`PREPARE` ステートメントは SQL ステートメントを準備し、それに名前 `stmt_name` を割り当てます。この名前は、あとでそのステートメントを参照するために使用されます。この準備済みステートメントは `EXECUTE` で実行され、`DEALLOCATE PREPARE` で解放されます。例については、[セクション13.5「プリペアドステートメント」](#)を参照してください。

ステートメント名では大/小文字は区別されません。`preparable_stmt` は、SQL ステートメントのテキストを含む文字列リテラルまたはユーザー変数です。このテキストは複数のステートメントではなく、1つのステートメントを表している必要があります。このステートメント内では、`?` 文字を、あとでクエリーを実行するときに、そのクエリーのどこにデータ値をバインドするかを示すパラメータマーカーとして使用できます。文字列値にバインドしようとしている場合でも、`?` 文字を引用符で囲んではいけません。パラメータマーカーは、SQL キーワードや識別子などではなく、データ値を指定するべき場所にしか使用できません。

指定された名前を持つ準備済みステートメントがすでに存在する場合、そのステートメントは、新しいステートメントが準備される前に暗黙的に解放されます。つまり、新しいステートメントにエラーが含まれていて準備できない場合は、エラーが返され、指定された名前を持つステートメントは存在しなくなります。

準備済みステートメントの範囲は、そのステートメントが作成されたセッションです。これには、次のいくつかの注意点があります。

- あるセッションで作成された準備済みステートメントを別のセッションで使用することはできません。
- セッションが (正常または異常にかかわらず) 終了すると、その準備済みステートメントは存在しなくなります。自動再接続が有効になっていると、クライアントには接続が失われたことが通知されません。このため、クライアントは自動再接続を無効にすることが必要になる場合があります。[Automatic Reconnection Control](#)を参照してください。
- ストアプログラム内で作成された準備済みステートメントは、そのプログラムが実行を完了したあとも引き続き存在し、あとでそのプログラムの外部で実行できます。
- ストアプログラムのコンテキストで準備されたステートメントは、ストアプロシージャやストアファンクションのパラメータまたはローカル変数を参照できません。これらは、そのプログラムが終了すると範囲から外れ、このステートメントがあとでプログラムの外部で実行されたときに使用できなくなるためです。回避方法として、代わりに、同様にセッションスコープを持つユーザー定義変数を参照します。[セクション9.4「ユーザー定義変数」](#)を参照してください。

MySQL 8.0.22 以降、プリペアドステートメントで使用されるパラメータのタイプは、そのステートメントが最初に準備されたときに決定され、このプリペアドステートメントに対して `EXECUTE` が起動されるたびに保持されます (このセクションで後述するようにステートメントが再準備されないかぎり)。パラメータタイプを決定するためのルールを次に示します:

- バイナリ算術演算子のオペランドであるパラメータは、他のオペランドと同じデータ型を持ちます。
- バイナリ算術演算子の両方のオペランドがパラメータの場合、パラメータの型は演算子のコンテキストによって決定されます。
- パラメータが単項算術演算子のオペランドである場合、パラメータタイプは演算子のコンテキストによって決定されます。
- 算術演算子に型決定コンテキストがない場合、関係するパラメータの導出型は `DOUBLE PRECISION` です。これは、たとえば、パラメータが `SELECT` リストの最上位ノードである場合や、比較演算子の一部である場合に発生することがあります。
- 文字列演算子のオペランドであるパラメータは、他のオペランドの集計型と同じ導出型を持ちます。演算子のすべてのオペランドがパラメータの場合、導出タイプは `VARCHAR` で、その照合は `collation_connection` の値によって決定されます。

- 時間演算子のオペランドであるパラメータは、演算子が `DATETIME` を返す場合は `DATETIME` 型、演算子が `TIME` を返す場合は `TIME`、演算子が `DATE` を返す場合は `DATE` 型になります。
- バイナリ比較演算子のオペランドであるパラメータは、比較の他のオペランドと同じ導出型を持ちます。
- `BETWEEN` などの 3 項比較演算子のオペランドであるパラメータは、他のオペランドの集計型と同じ導出型を持ちます。
- 比較演算子のすべてのオペランドがパラメータの場合、各オペランドの導出タイプは `VARCHAR` で、照合は `collation_connection` の値によって決定されます。
- `CASE`, `COALESCE`, `IF`, `IFNULL` または `NULLIF` の出力オペランドであるパラメータは、演算子の集計型と同じ導出型を持ちます。
- `CASE`, `COALESCE`, `IF`, `IFNULL` または `NULLIF` のすべての出力オペランドがパラメータであるか、すべて `NULL` である場合、パラメータのタイプは演算子のコンテキストによって決定されます。
- パラメータが `CASE`, `COALESCE()`, `IF` または `IFNULL` のオペランドであり、型決定コンテキストがない場合、関連する各パラメータの導出型は `VARCHAR` であり、その照合は `collation_connection` の値によって決定されます。
- `CAST()` のオペランドであるパラメータの型は、`CAST()` で指定されたものと同じです。
- パラメータが `INSERT` ステートメントの一部ではない `SELECT` リストの直接のメンバーである場合、パラメータの導出タイプは `VARCHAR` であり、その照合は `collation_connection` の値によって決定されます。
- パラメータが `INSERT` ステートメントの一部である `SELECT` リストの直接のメンバーである場合、パラメータの導出型は、パラメータが挿入される対応するカラムの型になります。
- パラメータが `UPDATE` ステートメントの `SET` 句または `INSERT` ステートメントの `ON DUPLICATE KEY UPDATE` 句で割当てのソースとして使用される場合、パラメータの導出タイプは、`SET` 句または `ON DUPLICATE KEY UPDATE` 句によって更新される対応するカラムのタイプになります。
- パラメータが関数の引数である場合、派生型は関数の戻り型によって異なります。

実際のタイプと導出タイプの一部の組合せでは、ステートメントの自動再準備がトリガーされ、以前のバージョンの MySQL との互換性が確保されます。次のいずれかの条件に該当する場合、再準備は行われません:

- `NULL` は、実際のパラメータ値として使用されます。
- パラメータは、`CAST()` のオペランドです。(かわりに、派生型へのキャストが試行され、キャストが失敗した場合は例外が発生します。)
- パラメータは文字列です。(この場合、暗黙的な `CAST(? AS derived_type)` が実行されます。)
- パラメータの導出タイプと実際のタイプはどちらも `INTEGER` であり、同じ符号を持ちます。
- パラメータ導出タイプは `DECIMAL` で、その実際のタイプは `DECIMAL` または `INTEGER` のいずれかです。
- 導出型は `DOUBLE` で、実際の型は任意の数値型です。
- 導出型と実際の型はどちらも文字列型です。
- 導出された型が temporal で、実際の型が temporal の場合。例外: 導出タイプは `TIME` で、実際のタイプは `TIME` ではありません。導出タイプは `DATE` で、実際のタイプは `DATE` ではありません。
- 導出型は時間的で、実際の型は数値です。

前述以外の場合は、ステートメントが再準備され、導出されたパラメータタイプのかわりに実際のパラメータタイプが使用されます。

これらのルールは、プリペアドステートメントで参照されるユーザー変数にも適用されます。

最初の実行後にステートメントを実行するために、プリペアドステートメント内の特定のパラメータまたはユーザー変数に異なるデータ型を使用すると、ステートメントが再準備されます。これは効率的ではありません。また、パラメータ(または変数)の実際の型が異なる可能性があるため、準備されたステートメントの後続の実行と結果に一貫

性がなくなる可能性があります。このような理由から、プリバードステートメントを再実行する場合は、特定のパラメータに同じデータ型を使用することをお勧めします。

13.5.2 EXECUTE ステートメント

```
EXECUTE stmt_name  
[USING @var_name [, @var_name] ...]
```

PREPARE でステートメントを準備したあと、準備済みステートメント名を参照する **EXECUTE** ステートメントでそのステートメントを実行します。準備済みステートメントにパラメータマーカーが含まれている場合は、そのパラメータにバインドされる値を含むユーザー変数をリストした **USING** 句を指定する必要があります。パラメータ値はユーザー変数でのみ提供することができ、**USING** 句では、このステートメント内のパラメータマーカーの数とまったく同じ数の変数を指定する必要があります。

特定の準備済みステートメントを複数回実行できます。それには、各ステートメントに異なる変数を渡すか、または各実行の前にその変数を異なる値に設定します。

例については、[セクション13.5「プリバードステートメント」](#)を参照してください。

13.5.3 DEALLOCATE PREPARE ステートメント

```
{DEALLOCATE | DROP} PREPARE stmt_name
```

PREPARE で生成された準備済みステートメントを解放するには、その準備済みステートメント名を参照する **DEALLOCATE PREPARE** ステートメントを使用します。準備済みステートメントを解放したあとにそのステートメントを実行しようとする、エラーが発生します。多すぎる準備済みステートメントが作成され、**DEALLOCATE PREPARE** ステートメントまたはセッションの終了のどちらによっても解放されない場合は、`max_prepared_stmt_count` システム変数によって上限が適用されることがあります。

例については、[セクション13.5「プリバードステートメント」](#)を参照してください。

13.6 複合ステートメントの構文

このセクションでは、**BEGIN ... END** 複合ステートメントや、ストアードプログラム (ストアードプロシージャとストアードファンクション、トリガー、およびイベント) の本体で使用できるその他のステートメントの構文について説明します。これらのオブジェクトは、あとで呼び出すためにサーバー上に格納されている SQL コードに対して定義されます ([第25章「ストアードオブジェクト」](#)を参照してください)。

複合ステートメントとは、ほかのブロック、つまり、変数、条件ハンドラ、およびカーソルの宣言、ループや条件付きテストなどのフロー制御構造構文を含めることのできるブロックのことです。

13.6.1 BEGIN ... END 複合ステートメント

```
[begin_label:] BEGIN  
[statement_list]  
END [end_label]
```

BEGIN ... END 構文は、ストアードプログラム (ストアードプロシージャとストアードファンクション、トリガー、およびイベント) 内に指定できる複合ステートメントを記述するために使用されます。複合ステートメントには、**BEGIN** および **END** キーワードで囲まれた複数のステートメントを含めることができます。`statement_list` は、それぞれがセミコロン (;) ステートメント区切り文字で終了する 1 つ以上のステートメントのリストを表します。`statement_list` 自体がオプションであるため、空の複合ステートメント (**BEGIN END**) は正当です。

BEGIN ... END ブロックはネストできます。

複数のステートメントを使用するには、クライアントが ; ステートメント区切り文字を含むステートメント文字列を送信する必要があります。`mysql` コマンド行クライアントでは、これは `delimiter` コマンドで処理されます。ステートメント終了の区切り文字 ; を (たとえば、// に) 変更すると、プログラム本体での ; の使用が許可されます。例については、[セクション25.1「ストアードプログラムの定義」](#)を参照してください。

BEGIN ... END ブロックにはラベルを付けることができます。[セクション13.6.2「ステートメントラベル」](#)を参照してください。

オプションの `[NOT] ATOMIC` 句はサポートされていません。つまり、この命令ブロックの先頭でトランザクションセーブポイントは設定されず、このコンテキストで使用されている `BEGIN` 句は現在のトランザクションに影響を与えません。

注記

すべてのストアプログラム内で、パーサーは、`BEGIN [WORK]` を `BEGIN ... END` ブロックの開始として扱います。このコンテキストでトランザクションを開始するには、代わりに `START TRANSACTION` を使用します。

13.6.2 ステートメントラベル

```
[begin_label:] BEGIN
  [statement_list]
END [end_label]

[begin_label:] LOOP
  statement_list
END LOOP [end_label]

[begin_label:] REPEAT
  statement_list
UNTIL search_condition
END REPEAT [end_label]

[begin_label:] WHILE search_condition DO
  statement_list
END WHILE [end_label]
```

`BEGIN ... END` ブロックや、`LOOP`、`REPEAT`、および `WHILE` ステートメントに対してラベルが許可されます。これらのステートメントに使用されるラベルは、次のルールに従います。

- `begin_label` のあとにコロンを付ける必要があります。
- `begin_label` は、`end_label` なしでも指定できます。 `end_label` が存在する場合、それは `begin_label` と同じである必要があります。
- `end_label` は、`begin_label` なしでは指定できません。
- 同じネストレベルにあるラベルは異なっている必要があります。
- ラベルは最大 16 文字の長さで指定できます。

ラベルが付けられた構造構文内でラベルを参照するには、`ITERATE` または `LEAVE` ステートメントを使用します。次の例では、これらのステートメントを使用して繰り返しを続行するか、またはループを終了します。

```
CREATE PROCEDURE doiterate(p1 INT)
BEGIN
  label1: LOOP
    SET p1 = p1 + 1;
    IF p1 < 10 THEN ITERATE label1; END IF;
    LEAVE label1;
  END LOOP label1;
END;
```

ブロックラベルの範囲には、そのブロック内で宣言されているハンドラのコードは含まれません。詳細は、[セクション13.6.7.2「DECLARE ... HANDLER ステートメント」](#)を参照してください。

13.6.3 DECLARE ステートメント

`DECLARE` ステートメントは、プログラムにローカルな、次のさまざまな項目を定義するために使用されます。

- ローカル変数。 [セクション13.6.4「ストアプログラム内の変数」](#)を参照してください。
- 条件とハンドラ。 [セクション13.6.7「条件の処理」](#)を参照してください。
- カーソル。 [セクション13.6.6「カーソル」](#)を参照してください。

DECLARE は、**BEGIN ... END** 複合ステートメントの内部でのみ許可され、ほかのどのステートメントよりも前の、その複合ステートメントの先頭に存在する必要があります。

宣言は、特定の順序に従う必要があります。カーソル宣言は、ハンドラ宣言の前に指定する必要があります。変数および条件宣言は、カーソルまたはハンドラ宣言の前に指定する必要があります。

13.6.4 ストアドプログラム内の変数

システム変数とユーザー定義変数は、ストアドプログラムのコンテキストの外部で使用できるのと同様に、ストアドプログラム内で使用できます。さらに、ストアドプログラムは **DECLARE** を使用してローカル変数を定義でき、またストアドルーチン (プロシージャおよびファンクション) は、そのルーチンとその呼び出し元の間で値を通信するパラメータを受け取るように宣言できます。

- [セクション13.6.4.1「ローカル変数 DECLARE ステートメント」](#)で説明されているように、ローカル変数を宣言するには、**DECLARE** ステートメントを使用します。
- 変数は、**SET** ステートメントを使用して直接設定できます。 [セクション13.7.6.1「変数代入の SET 構文」](#)を参照してください。
- クエリーからの結果は、**SELECT ... INTO var_list** を使用するか、またはカーソルを開き、**FETCH ... INTO var_list** を使用することによってローカル変数に取得できます。 [セクション13.2.10.1「SELECT ... INTO ステートメント」](#) および [セクション13.6.6「カーソル」](#)を参照してください。

ローカル変数のスコープ、および MySQL があいまいな名前を解決する方法については、 [セクション13.6.4.2「ローカル変数のスコープと解決」](#)を参照してください。

ストアドプロシージャやストアドファンクションのパラメータまたはストアドプログラムのローカル変数に (たとえば、**SET var_name = DEFAULT** ステートメントを使用して) 値 **DEFAULT** を割り当てることは許可されません。MySQL 8.0 では、これにより構文エラーが発生します。

13.6.4.1 ローカル変数 DECLARE ステートメント

```
DECLARE var_name [, var_name] ... type [DEFAULT value]
```

このステートメントは、ストアドプログラム内のローカル変数を宣言します。変数のデフォルト値を指定するには、**DEFAULT** 句を含めます。この値は式として指定できます。定数である必要はありません。**DEFAULT** 句がない場合、初期値は **NULL** になります。

ローカル変数は、データ型やオーバーフローチェックに関して、ストアドルーチンパラメータと同様に処理されます。 [セクション13.1.17「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」](#)を参照してください。

変数宣言は、カーソルまたはハンドラ宣言の前に指定する必要があります。

ローカル変数名では、大/小文字は区別されません。 [セクション9.2「スキーマオブジェクト名」](#)で説明されているように、許可される文字や引用符のルールはほかの識別子の場合と同じです。

ローカル変数のスコープは、それが宣言されている **BEGIN ... END** ブロックです。この変数は、同じ名前を持つ変数を宣言しているブロックを除き、宣言しているブロック内でネストされたブロック内で参照できます。

変数宣言の例は、 [セクション13.6.4.2「ローカル変数のスコープと解決」](#)を参照してください。

13.6.4.2 ローカル変数のスコープと解決

ローカル変数のスコープは、それが宣言されている **BEGIN ... END** ブロックです。この変数は、同じ名前を持つ変数を宣言しているブロックを除き、宣言しているブロック内でネストされたブロック内で参照できます。

ローカル変数はストアドプログラムの実行中のみスコープ内にあるので、これらの参照は、ストアドプログラム内で作成された準備済みステートメントでは許可されていません。準備済みステートメントのスコープは現在のセッションであり、ストアドプログラムではないので、ステートメントはプログラムの終了後に実行でき、この時点で変数はスコープ内に存在しなくなります。たとえば、**SELECT ... INTO local_var** は準備済みステートメントとして使用できません。この制約は、ストアドプロシージャおよびストアドファンクションのパラメータにも適用されます。 [セクション13.5.1「PREPARE ステートメント」](#)を参照してください。

ローカル変数にテーブルカラムと同じ名前を付けてはいけません。 `SELECT ... INTO` ステートメントなどの SQL ステートメントに、カラムおよび同じ名前を持つ宣言されたローカル変数への参照が含まれている場合、MySQL は現在、その参照を変数の名前として解釈します。次のプロシージャ定義を考えてみます。

```
CREATE PROCEDURE sp1 (x VARCHAR(5))
BEGIN
  DECLARE xname VARCHAR(5) DEFAULT 'bob';
  DECLARE newname VARCHAR(5);
  DECLARE xid INT;

  SELECT xname, id INTO newname, xid
    FROM table1 WHERE xname = xname;
  SELECT newname;
END;
```

MySQL は、`SELECT` ステートメント内の `xname` を、`xname` カラムではなく `xname` 変数への参照として解釈します。その結果、プロシージャ `sp1()` が呼び出されると、`table1.xname` カラムの値には関係なく、`newname` 変数は値 `'bob'` を返します。

同様に、次のプロシージャ内のカーソル定義には、`xname` を参照する `SELECT` ステートメントが含まれています。MySQL はこれをカラム参照ではなく、その名前の変数への参照として解釈します。

```
CREATE PROCEDURE sp2 (x VARCHAR(5))
BEGIN
  DECLARE xname VARCHAR(5) DEFAULT 'bob';
  DECLARE newname VARCHAR(5);
  DECLARE xid INT;
  DECLARE done TINYINT DEFAULT 0;
  DECLARE cur1 CURSOR FOR SELECT xname, id FROM table1;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

  OPEN cur1;
read_loop: LOOP
  FETCH FROM cur1 INTO newname, xid;
  IF done THEN LEAVE read_loop; END IF;
  SELECT newname;
END LOOP;
CLOSE cur1;
END;
```

[セクション25.8「ストアプログラムの制約」](#) も参照してください。

13.6.5 フロー制御ステートメント

MySQL は、ストアプログラム内のフロー制御のために、`IF`、`CASE`、`ITERATE`、`LEAVE LOOP`、`WHILE`、および `REPEAT` 構造構文をサポートしています。また、ストアファンクション内の `RETURN` もサポートしています。

これらの構造構文の多くには、次の各セクションの文法仕様に示されているその他のステートメントが含まれています。このような構造構文はネストできます。たとえば、`IF` ステートメントには、それ自体に `CASE` ステートメントを含む `WHILE` ループが含まれている可能性があります。

MySQL は、`FOR` ループをサポートしていません。

13.6.5.1 CASE ステートメント

```
CASE case_value
  WHEN when_value THEN statement_list
  [WHEN when_value THEN statement_list] ...
  [ELSE statement_list]
END CASE
```

または:

```
CASE
  WHEN search_condition THEN statement_list
  [WHEN search_condition THEN statement_list] ...
  [ELSE statement_list]
END CASE
```

ストアプログラムの `CASE` ステートメントは、複雑な条件構造構文を実装します。

注記

ここで説明する **CASE statement** とは異なる **CASE operator** もあります。 [セクション 12.5「フロー制御関数」](#) を参照してください。 **CASE** ステートメントは **ELSE NULL** 句を持つことができず、**END** でなく、**END CASE** で終了します。

最初の構文の場合、**case_value** は式です。この値は、各 **WHEN** 句内の **when_value** 式のいずれかに等しくなるまで、それらの式と比較されます。等しい **when_value** が見つかったら、対応する **THEN** 句の **statement_list** が実行されます。どの **when_value** も等しくない場合は、**ELSE** 句の **statement_list** が実行されます (この句が存在する場合)。

NULL = NULL は **false** であるため、この構文を **NULL** と等しいかどうかのテストに使用することはできません。 [セクション 3.3.4.6「NULL 値の操作」](#) を参照してください。

2 番目の構文の場合、各 **WHEN** 句の **search_condition** 式のいずれかが **true** になるまでそれらの式が評価され、いずれかが **true** になった時点で、それに対応する **THEN** 句の **statement_list** が実行されます。どの **search_condition** も等しくない場合は、**ELSE** 句の **statement_list** が実行されます (この句が存在する場合)。

どの **when_value** も **search_condition** もテストされた値に一致せず、かつ **CASE** ステートメントに **ELSE** 句が含まれていない場合は、**Case not found for CASE statement** エラーになります。

各 **statement_list** は、1 つ以上の SQL ステートメントで構成されます。空の **statement_list** は許可されません。

どの **WHEN** 句でも値が一致しない状況を処理するには、次の例に示すように、空の **BEGIN ... END** ブロックを含む **ELSE** を使用します。(この **ELSE** 句で使用されているインデントは透明性のみを目的にしており、それ以外の意味はありません。)

```
DELIMITER |
CREATE PROCEDURE p()
BEGIN
  DECLARE v INT DEFAULT 1;

  CASE v
    WHEN 2 THEN SELECT v;
    WHEN 3 THEN SELECT 0;
    ELSE
      BEGIN
        END;
    END CASE;
  END;
|
```

13.6.5.2 IF ステートメント

```
IF search_condition THEN statement_list
[ELSEIF search_condition THEN statement_list] ...
[ELSE statement_list]
END IF
```

ストアードプログラムの **IF** ステートメントは、基本的な条件構造構文を実装します。

注記

ここで説明されている **IF** ステートメントとは異なる **IF()** 関数も存在します。 [セクション 12.5「フロー制御関数」](#) を参照してください。 **IF** ステートメントは **THEN**、**ELSE**、および **ELSEIF** 句を含むことができ、**END IF** で終了します。

特定の **search_condition** が **true** と評価された場合、対応する **THEN** 句または **ELSEIF** 句 **statement_list** が実行されます。どの **search_condition** も一致しない場合は、**ELSE** 句の **statement_list** が実行されます。

各 **statement_list** は、1 つ以上の SQL ステートメントで構成されます。空の **statement_list** は許可されません。

IF ... END IF ブロックは、次の例に示すように、ストアードプログラム内で使用されるその他のすべてのフロー制御ブロックと同様にセミコロンで終了する必要があります。

```
DELIMITER //
```

```
CREATE FUNCTION SimpleCompare(n INT, m INT)
RETURNS VARCHAR(20)

BEGIN
  DECLARE s VARCHAR(20);

  IF n > m THEN SET s = '>';
  ELSEIF n = m THEN SET s = '=';
  ELSE SET s = '<';
  END IF;

  SET s = CONCAT(n, '', s, '', m);

  RETURN s;
END //

DELIMITER ;
```

ほかのフロー制御構造構文と同様に、**IF ... END IF** ブロックは、ほかのフロー制御構造構文 (ほかの **IF** ステートメントを含む) 内にネストできます。各 **IF** は、独自の **END IF** とそれに続くセミコロンで終了する必要があります。次に示すように、インデントを使用して、ネストされたフロー制御ブロックを人間が読みやすくすることができます (ただし、これが MySQL に必要なわけではありません)。

```
DELIMITER //

CREATE FUNCTION VerboseCompare (n INT, m INT)
RETURNS VARCHAR(50)

BEGIN
  DECLARE s VARCHAR(50);

  IF n = m THEN SET s = 'equals';
  ELSE
    IF n > m THEN SET s = 'greater';
    ELSE SET s = 'less';
  END IF;

  SET s = CONCAT('is ', s, ' than');
  END IF;

  SET s = CONCAT(n, '', s, '', m, '');

  RETURN s;
END //

DELIMITER ;
```

この例では、内側の **IF** は **n** が **m** に等しくない場合にのみ評価されます。

13.6.5.3 ITERATE ステートメント

```
ITERATE label
```

ITERATE は、**LOOP**、**REPEAT**、および **WHILE** ステートメント内にのみ指定できます。**ITERATE** は、「ループをふたたび開始する」ことを示します。

例については、[セクション13.6.5.5「LOOP ステートメント」](#)を参照してください。

13.6.5.4 LEAVE ステートメント

```
LEAVE label
```

このステートメントは、特定のラベルを持つフロー制御構造構文を終了するために使用されます。そのラベルがもっとも外側のストアードプログラムブロックのものである場合、**LEAVE** はプログラムを終了します。

LEAVE は、**BEGIN ... END** またはループ構造構文 (**LOOP**、**REPEAT**、**WHILE**) 内で使用できます。

例については、[セクション13.6.5.5「LOOP ステートメント」](#)を参照してください。

13.6.5.5 LOOP ステートメント

```
[begin_label:] LOOP  
statement_list  
END LOOP [end_label]
```

LOOP は単純なループ構造構文を実装し、それぞれがセミコロン (;) ステートメント区切り文字で終了する 1 つ以上のステートメントで構成されたステートメントリストの繰り返し実行を可能にします。ループ内の各ステートメントは、そのループが終了するまで繰り返されます。通常、これは **LEAVE** ステートメントで実行されます。ストアドファンクション内では、**RETURN** も使用できます。これにより、そのストアドファンクションが完全に終了します。

ループ終了ステートメントが含まれていない場合は、無限ループが発生します。

LOOP ステートメントにはラベルを付けることができます。ラベルの使用に関連したルールについては、[セクション 13.6.2 「ステートメントラベル」](#) を参照してください。

例:

```
CREATE PROCEDURE doiterate(p1 INT)  
BEGIN  
label1: LOOP  
  SET p1 = p1 + 1;  
  IF p1 < 10 THEN  
    ITERATE label1;  
  END IF;  
  LEAVE label1;  
END LOOP label1;  
SET @x = p1;  
END;
```

13.6.5.6 REPEAT ステートメント

```
[begin_label:] REPEAT  
statement_list  
UNTIL search_condition  
END REPEAT [end_label]
```

REPEAT ステートメント内のステートメントリストは、`search_condition` 式が true になるまで繰り返されます。そのため、**REPEAT** は常に、少なくとも 1 回はループに入ります。`statement_list` は、それぞれがセミコロン (;) ステートメント区切り文字で終了する 1 つ以上のステートメントで構成されます。

REPEAT ステートメントにはラベルを付けることができます。ラベルの使用に関連したルールについては、[セクション 13.6.2 「ステートメントラベル」](#) を参照してください。

例:

```
mysql> delimiter //  
  
mysql> CREATE PROCEDURE dorepeat(p1 INT)  
BEGIN  
  SET @x = 0;  
  REPEAT  
    SET @x = @x + 1;  
  UNTIL @x > p1 END REPEAT;  
END  
//  
Query OK, 0 rows affected (0.00 sec)  
  
mysql> CALL dorepeat(1000)//  
Query OK, 0 rows affected (0.00 sec)  
  
mysql> SELECT @x//  
+-----+  
| @x |  
+-----+  
| 1001 |  
+-----+  
1 row in set (0.00 sec)
```

13.6.5.7 RETURN ステートメント

```
RETURN expr
```

RETURN ステートメントはストアドファンクションの実行を終了し、そのストアドファンクションの呼び出し元に値 `expr` を返します。ストアドファンクション内には、少なくとも 1 つの **RETURN** ステートメントが存在する必要があります。そのストアドファンクションに複数の終了ポイントがある場合は、複数存在してもかまいません。

このステートメントは、ストアドプロシージャ、トリガー、またはイベントでは使用されません。**LEAVE** ステートメントを使用すると、これらのタイプのストアドプログラムを終了できます。

13.6.5.8 WHILE ステートメント

```
[begin_label:] WHILE search_condition DO
  statement_list
END WHILE [end_label]
```

WHILE ステートメント内のステートメントリストは、`search_condition` 式が `true` であるかぎり繰り返されます。`statement_list` は、それぞれがセミコロン (;) ステートメント区切り文字で終了する 1 つ以上の SQL ステートメントで構成されます。

WHILE ステートメントにはラベルを付けることができます。ラベルの使用に関連したルールについては、[セクション 13.6.2 「ステートメントラベル」](#) を参照してください。

例:

```
CREATE PROCEDURE dowhile()
BEGIN
  DECLARE v1 INT DEFAULT 5;

  WHILE v1 > 0 DO
    ...
    SET v1 = v1 - 1;
  END WHILE;
END;
```

13.6.6 カーソル

MySQL は、ストアドプログラム内部のカーソルをサポートします。その構文は、組み込み SQL の場合と同様です。カーソルには次のプロパティがあります。

- Asensitive: サーバーは、結果テーブルのコピーを作成する場合としない場合があります
- 読み取り専用: 更新できません
- スクロール不可: 1 方向にしかトラバースできず、行をスキップできません

カーソル宣言は、ハンドラ宣言の前で、かつ変数および条件宣言のあとに指定する必要があります。

例:

```
CREATE PROCEDURE curdemo()
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE a CHAR(16);
  DECLARE b, c INT;
  DECLARE cur1 CURSOR FOR SELECT id,data FROM test.t1;
  DECLARE cur2 CURSOR FOR SELECT i FROM test.t2;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

  OPEN cur1;
  OPEN cur2;

  read_loop: LOOP
    FETCH cur1 INTO a, b;
    FETCH cur2 INTO c;
    IF done THEN
      LEAVE read_loop;
    END IF;
    IF b < c THEN
      INSERT INTO test.t3 VALUES (a,b);
    ELSE
      INSERT INTO test.t3 VALUES (a,c);
    END IF;
  END LOOP;
```

```
END IF;  
END LOOP;  
  
CLOSE cur1;  
CLOSE cur2;  
END;
```

13.6.6.1 カーソル CLOSE ステートメント

```
CLOSE cursor_name
```

このステートメントは、以前に開かれたカーソルを閉じます。例については、[セクション13.6.6「カーソル」](#)を参照してください。

カーソルが開いていない場合は、エラーが発生します。

明示的に閉じられない場合は、そのカーソルが宣言された `BEGIN ... END` ブロックの最後に閉じられます。

13.6.6.2 カーソル DECLARE ステートメント

```
DECLARE cursor_name CURSOR FOR select_statement
```

このステートメントはカーソルを宣言し、そのカーソルによってトラバースされる行を取得する `SELECT` ステートメントに関連付けます。行をあとでフェッチするには、`FETCH` ステートメントを使用します。`SELECT` ステートメントによって取得されるカラムの数が、`FETCH` ステートメントで指定された出力変数の数に一致している必要があります。

`SELECT` ステートメントに `INTO` 句を含めることはできません。

カーソル宣言は、ハンドラ宣言の前で、かつ変数および条件宣言のあとに指定する必要があります。

ストアプログラムには複数のカーソル宣言を含めることができますが、特定のブロック内で宣言された各カーソルが一意的の名前を持っている必要があります。例については、[セクション13.6.6「カーソル」](#)を参照してください。

`SHOW` ステートメントで入手できる情報については、多くの場合、`INFORMATION_SCHEMA` テーブルでカーソルを使用することによって同等の情報を取得できます。

13.6.6.3 カーソル FETCH ステートメント

```
FETCH [[NEXT] FROM] cursor_name INTO var_name [, var_name] ...
```

このステートメントは、指定されたカーソル (これは開いている必要があります) に関連付けられた `SELECT` ステートメントの次の行をフェッチし、そのカーソルのポインタを進めます。行が存在する場合は、フェッチされたカラムが指定された変数に格納されます。`SELECT` ステートメントによって取得されるカラムの数が、`FETCH` ステートメントで指定された出力変数の数に一致している必要があります。

それ以上の行を取得できない場合は、SQLSTATE 値 '02000' で「データなし」状況が発生します。この状況を検出するには、その状況 (または、`NOT FOUND` 状況) のハンドラを設定できます。例については、[セクション13.6.6「カーソル」](#)を参照してください。

`SELECT` や別の `FETCH` などの別の操作でも、同じ条件が発生させることでハンドラが実行される場合があることに注意してください。条件が発生した操作を区別する必要がある場合は、独自のハンドラに関連付けることができるように、操作を独自の `BEGIN ... END` ブロック内に配置します。

13.6.6.4 カーソル OPEN ステートメント

```
OPEN cursor_name
```

このステートメントは、以前に宣言されたカーソルを開きます。例については、[セクション13.6.6「カーソル」](#)を参照してください。

13.6.6.5 サーバー側のカーソルの制約

サーバー側のカーソルは、`mysql_stmt_attr_set()` 関数を使用して C API に実装されます。ストアルーチンのカーソルにも同じ実装が使用されます。サーバー側のカーソルによって、サーバー側で結果セットを生成できるようになり

ますが、クライアントが要求する行を除いてクライアントに転送することはできません。たとえば、クライアントがクエリーを実行するが、最初の行のみが必要な場合、残りの行は転送されません。

MySQL では、サーバー側のカーソルは内部一時テーブルに実体化されます。最初これは `MEMORY` テーブルですが、そのサイズが `max_heap_table_size` および `tmp_table_size` システム変数の最小値を超えると、`MyISAM` テーブルに変換されます。カーソルの結果セットを保持するために作成された内部一時テーブルには、内部一時テーブルの他の使用と同じ制限が適用されます。[セクション 8.4.4 「MySQL での内部一時テーブルの使用」](#) を参照してください。この実装の制限の 1 つには、大きな結果セットの場合に、カーソルによる行の取得に時間がかかることがあるというものがあります。

カーソルは読み取り専用です。カーソルを使用して行を更新できません。

`UPDATE WHERE CURRENT OF` および `DELETE WHERE CURRENT OF` は、更新可能なカーソルがサポートされていないため実装されません。

カーソルは保持不可能です (コミット後、開いたままにはできません)。

カーソルは非センシティブです。

カーソルはスクロール不可です。

カーソルには名前が付けられません。ステートメントハンドラがカーソル ID として機能します。

準備済みステートメントごとに、カーソルを 1 つだけ開いておくことができます。複数のカーソルが必要な場合は、複数のステートメントを準備する必要があります。

結果セットを生成するステートメントで、準備モードでサポートされていないものにはカーソルを使用できません。このようなステートメントには、`CHECK TABLE`、`HANDLER READ`、`SHOW BINLOG EVENTS` などがあります。

13.6.7 条件の処理

条件は、現在のプログラムブロックの終了や実行の続行などの、特殊な処理が必要なストアプログラムの実行中に発生する可能性があります。警告や例外などの一般的な条件、または特定のエラーコードなどの具体的な条件に対してハンドラを定義できます。具体的な条件には名前を割り当てることのできるため、ハンドラではその名前で参照できます。

条件に名前を付けるには、`DECLARE ... CONDITION` ステートメントを使用します。ハンドラを宣言するには、`DECLARE ... HANDLER` ステートメントを使用します。[セクション 13.6.7.1 「DECLARE ... CONDITION ステートメント」](#) および [セクション 13.6.7.2 「DECLARE ... HANDLER ステートメント」](#) を参照してください。条件が発生したときにサーバーがハンドラを選択する方法については、[セクション 13.6.7.6 「ハンドラのスコープに関するルール」](#) を参照してください。

条件を発生させるには、`SIGNAL` ステートメントを使用します。条件ハンドラ内で条件情報を変更するには、`RESIGNAL` を使用します。[セクション 13.6.7.1 「DECLARE ... CONDITION ステートメント」](#) および [セクション 13.6.7.2 「DECLARE ... HANDLER ステートメント」](#) を参照してください。

診断領域から情報を取得するには、`GET DIAGNOSTICS` ステートメントを使用します ([セクション 13.6.7.3 「GET DIAGNOSTICS ステートメント」](#) を参照してください)。診断領域については、[セクション 13.6.7.7 「MySQL の診断領域」](#) を参照してください。

13.6.7.1 DECLARE ... CONDITION ステートメント

```
DECLARE condition_name CONDITION FOR condition_value
```

```
condition_value: {  
  mysql_error_code  
  | SQLSTATE [VALUE] sqlstate_value  
}
```

`DECLARE ... CONDITION` ステートメントは名前付きエラー条件を宣言し、特定の処理が必要な条件に名前を関連付けます。この名前は、以降の `DECLARE ... HANDLER` ステートメントで参照できます ([セクション 13.6.7.2 「DECLARE ... HANDLER ステートメント」](#) を参照してください)。

条件宣言は、カーソルまたはハンドラ宣言の前に指定する必要があります。

`condition_value` for `DECLARE ... CONDITION` は、条件名に関連付ける特定の条件または条件のクラスを示します。次の形式を使用できます:

- `mysql_error_code`: MySQL エラーコードを示す整数リテラル。

MySQL エラーコード 0 はエラー条件ではなく成功を示すため、使用しないでください。MySQL エラーコードのリストは、[Server Error Message Reference](#) を参照してください。

- `SQLSTATE [VALUE] sqlstate_value`: SQLSTATE 値を示す 5 文字の文字列リテラル。

'00' で始まる SQLSTATE 値は、エラー条件ではなく成功を示すため、使用しないでください。SQLSTATE 値のリストについては、[Server Error Message Reference](#) を参照してください。

`SIGNAL` で参照されるか、または `RESIGNAL` ステートメントで使用される条件名は MySQL エラーコードではなく、SQLSTATE 値に関連付けられている必要があります。

条件に名前を使用すると、ストアプログラムのコードの明確化に役立つ場合があります。たとえば、このハンドラは存在しないテーブルの削除の試行に適用されますが、1051 が「不明なテーブル」の MySQL エラーコードであることがわかっている場合にのみわかります:

```
DECLARE CONTINUE HANDLER FOR 1051
BEGIN
  -- body of handler
END;
```

条件の名前を宣言することによって、このハンドラの目的がより簡単にわかるようになります。

```
DECLARE no_such_table CONDITION FOR 1051;
DECLARE CONTINUE HANDLER FOR no_such_table
BEGIN
  -- body of handler
END;
```

これは、同じ条件の、MySQL エラーコードではなく、対応する SQLSTATE 値に基づく名前付き条件です。

```
DECLARE no_such_table CONDITION FOR SQLSTATE '42S02';
DECLARE CONTINUE HANDLER FOR no_such_table
BEGIN
  -- body of handler
END;
```

13.6.7.2 DECLARE ... HANDLER ステートメント

```
DECLARE handler_action HANDLER
  FOR condition_value [, condition_value] ...
  statement

handler_action: {
  CONTINUE
| EXIT
| UNDO
}

condition_value: {
  mysql_error_code
| SQLSTATE [VALUE] sqlstate_value
| condition_name
| SQLWARNING
| NOT FOUND
| SQLEXCEPTION
}
```

`DECLARE ... HANDLER` ステートメントは、1 つ以上の条件を処理するハンドラを指定します。これらの条件のいずれかが発生した場合は、指定された `statement` が実行されます。`statement` は `SET var_name = value` などの単純なステートメントでも、`BEGIN` と `END` を使用して記述された複合ステートメントでもかまいません ([セクション 13.6.1「BEGIN ... END 複合ステートメント」](#) を参照してください)。

ハンドラ宣言は、変数または条件宣言のあとに指定する必要があります。

`handler_action` 値は、ハンドラステートメントの実行後にハンドラがどのようなアクションを実行するかを示します。

- **CONTINUE**: 現在のプログラムの実行が続行されます。
- **EXIT**: このハンドラが宣言されている **BEGIN ... END** 複合ステートメントの実行が終了します。これは、この条件が内側のブロックで発生した場合にも当てはまります。
- **UNDO**: サポートされていません。

`condition_value` for **DECLARE ... HANDLER** は、ハンドラをアクティブ化する特定の条件または条件のクラスを示します。次の形式を使用できます:

- **mysql_error_code**: MySQL エラーコードを示す整数リテラル (「不明なテーブル」を指定する 1051 など):

```
DECLARE CONTINUE HANDLER FOR 1051
BEGIN
  -- body of handler
END;
```

MySQL エラーコード 0 はエラー条件ではなく成功を示すため、使用しないでください。MySQL エラーコードのリストは、[Server Error Message Reference](#) を参照してください。

- **SQLSTATE [VALUE] sqlstate_value**: SQLSTATE 値を示す 5 文字の文字列リテラル (「不明なテーブル」を指定する '42S01' など):

```
DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02'
BEGIN
  -- body of handler
END;
```

'00' で始まる SQLSTATE 値は、エラー条件ではなく成功を示すため、使用しないでください。SQLSTATE 値のリストについては、[Server Error Message Reference](#) を参照してください。

- **condition_name**: **DECLARE ... CONDITION** で以前に指定された条件名。条件名は MySQL エラーコードまたは SQLSTATE 値に関連付けることができます。[セクション 13.6.7.1 「DECLARE ... CONDITION ステートメント」](#) を参照してください。
- **SQLWARNING**: '01' で始まる SQLSTATE 値のクラスの短縮形。

```
DECLARE CONTINUE HANDLER FOR SQLWARNING
BEGIN
  -- body of handler
END;
```

- **NOT FOUND**: '02' で始まる SQLSTATE 値のクラスの短縮形。これは、カーソルのコンテキストに関係しており、カーソルがデータセットの最後に達したときの動作を制御するために使用します。それ以上の行を取得できない場合は、SQLSTATE 値 '02000' で「データなし」状況が発生します。この条件を検出するには、その条件または **NOT FOUND** 条件のハンドラを設定します。

```
DECLARE CONTINUE HANDLER FOR NOT FOUND
BEGIN
  -- body of handler
END;
```

別の例については、[セクション 13.6.6 「カーソル」](#) を参照してください。**NOT FOUND** 条件は、行を取得しない **SELECT ... INTO var_list** ステートメントにも発生します。

- **SQLEXCEPTION**: '00'、'01' または '02' で始まらない SQLSTATE 値のクラスの短縮形。

```
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
BEGIN
  -- body of handler
END;
```

条件が発生したときにサーバーがハンドラを選択する方法については、[セクション 13.6.7.6 「ハンドラのスコープに関するルール」](#) を参照してください。

対応するハンドラが宣言されていない条件が発生した場合、実行されるアクションはその条件のクラスによって異なります。

- **SQLEXCEPTION** 条件の場合は、**EXIT** ハンドラが存在するかのように、ストアプログラムはその条件を発生させたステートメントで終了します。そのプログラムが別のストアプログラムから呼び出されていた場合は、呼び出し元プログラムが、独自のハンドラに適用されるハンドラ選択ルールを使用してその条件を処理します。
- **SQLWARNING** 条件の場合は、**CONTINUE** ハンドラが存在するかのように、プログラムは実行を続行します。
- **NOT FOUND** 条件では、その条件が正常に発生した場合、アクションは **CONTINUE** です。 **SIGNAL** または **RESIGNAL** によって発生した場合、アクションは **EXIT** です。

次の例では、重複キーエラーに対して発生する **SQLSTATE '23000'** のハンドラを使用します。

```
mysql> CREATE TABLE test.t (s1 INT, PRIMARY KEY (s1));
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter //

mysql> CREATE PROCEDURE handlerdemo ()
BEGIN
  DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;
  SET @x = 1;
  INSERT INTO test.t VALUES (1);
  SET @x = 2;
  INSERT INTO test.t VALUES (1);
  SET @x = 3;
END;
//
Query OK, 0 rows affected (0.00 sec)

mysql> CALL handlerdemo();//
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x//
+-----+
| @x   |
+-----+
| 3    |
+-----+
1 row in set (0.00 sec)
```

このプロシージャの実行後、**@x** が 3 になっていることを確認してください。これは、エラーが発生したあと、プロシージャの最後まで実行が続行されたことを示しています。 **DECLARE ... HANDLER** ステートメントが存在しなかったとすると、**PRIMARY KEY** 制約のために 2 番目の **INSERT** が失敗したあとに MySQL はデフォルトのアクション (**EXIT**) を実行するため、**SELECT @x** は 2 を返していました。

条件を無視するには、その条件の **CONTINUE** ハンドラを宣言し、それを空のブロックに関連付けます。例:

```
DECLARE CONTINUE HANDLER FOR SQLWARNING BEGIN END;
```

ブロックレベルのスコープには、そのブロック内で宣言されているハンドラのコードは含まれません。そのため、ハンドラに関連付けられたステートメントは、**ITERATE** または **LEAVE** を使用して、そのハンドラ宣言を囲むブロックのラベルを参照することができません。 **REPEAT** ブロックに **retry** のラベルが含まれている次の例を考えてみます。

```
CREATE PROCEDURE p ()
BEGIN
  DECLARE i INT DEFAULT 3;
  retry:
  REPEAT
  BEGIN
    DECLARE CONTINUE HANDLER FOR SQLWARNING
    BEGIN
      ITERATE retry; # illegal
    END;
    IF i < 0 THEN
      LEAVE retry; # legal
    END IF;
    SET i = i - 1;
  END;
UNTIL FALSE END REPEAT;
```

```
END;
```

`retry` ラベルは、そのブロック内の `IF` ステートメントのスコープ内にあります。 `CONTINUE` ハンドラのスコープ内にはないため、そこでの参照は無効であり、エラーが発生します。

```
ERROR 1308 (42000): LEAVE with no matching label: retry
```

ハンドラ内の外側のラベルへの参照を回避するには、次の方法のいずれかを使用します。

- このブロックを離れるには、`EXIT` ハンドラを使用します。ブロックのクリーンアップが必要ない場合は、`BEGIN ... END` ハンドラ本体を空にすることができます。

```
DECLARE EXIT HANDLER FOR SQLWARNING BEGIN END;
```

そうでない場合は、ハンドラ本体内にクリーンアップステートメントを配置します。

```
DECLARE EXIT HANDLER FOR SQLWARNING
BEGIN
  block cleanup statements
END;
```

- 実行を続行するには、`CONTINUE` ハンドラ内に、囲んでいるブロック内でチェックすることによってそのハンドラが呼び出されたかどうかを判定できるステータス変数を設定します。次の例では、この目的のために変数 `done` を使用します。

```
CREATE PROCEDURE p ()
BEGIN
  DECLARE i INT DEFAULT 3;
  DECLARE done INT DEFAULT FALSE;
  retry:
  REPEAT
  BEGIN
    DECLARE CONTINUE HANDLER FOR SQLWARNING
    BEGIN
      SET done = TRUE;
    END;
    IF done OR i < 0 THEN
      LEAVE retry;
    END IF;
    SET i = i - 1;
  END;
  UNTIL FALSE END REPEAT;
END;
```

13.6.7.3 GET DIAGNOSTICS ステートメント

```
GET [CURRENT | STACKED] DIAGNOSTICS {
  statement_information_item
  [, statement_information_item] ...
  | CONDITION condition_number
  condition_information_item
  [, condition_information_item] ...
}

statement_information_item:
  target = statement_information_item_name

condition_information_item:
  target = condition_information_item_name

statement_information_item_name: {
  NUMBER
  | ROW_COUNT
}

condition_information_item_name: {
  CLASS_ORIGIN
  | SUBCLASS_ORIGIN
  | RETURNED_SQLSTATE
  | MESSAGE_TEXT
  | MYSQL_ERRNO
  | CONSTRAINT_CATALOG
```

```
| CONSTRAINT_SCHEMA
| CONSTRAINT_NAME
| CATALOG_NAME
| SCHEMA_NAME
| TABLE_NAME
| COLUMN_NAME
| CURSOR_NAME
}
```

condition_number, target:
(see following discussion)

SQL ステートメントは、診断領域を移入する診断情報を生成します。GET DIAGNOSTICS ステートメントを使用すると、アプリケーションでこの情報を検査できます。(SHOW WARNINGS または SHOW ERRORS を使用して、条件またはエラーを確認することもできます。)

GET DIAGNOSTICS を実行するために特殊な権限は必要ありません。

キーワード CURRENT は、現在の診断領域から情報を取得することを示します。キーワード STACKED は、現在のコンテキストが条件ハンドラである場合にのみ使用可能な 2 番目の診断領域から情報を取得することを意味します。どちらのキーワードも指定しない場合、デフォルトでは現在の診断領域が使用されます。

GET DIAGNOSTICS ステートメントは通常、ストアードプログラム内のハンドラで使用されます。これは、任意の SQL ステートメントの実行をチェックするためにハンドラコンテキストの外部で GET [CURRENT] DIAGNOSTICS が許可されている MySQL 拡張機能です。たとえば、mysql クライアントプログラムを呼び出す場合は、プロンプトで次のステートメントを入力できます。

```
mysql> DROP TABLE test.no_such_table;
ERROR 1051 (42S02): Unknown table 'test.no_such_table'
mysql> GET DIAGNOSTICS CONDITION 1
        @p1 = RETURNED_SQLSTATE, @p2 = MESSAGE_TEXT;
mysql> SELECT @p1, @p2;
+-----+-----+
| @p1 | @p2 |
+-----+-----+
| 42S02 | Unknown table 'test.no_such_table' |
+-----+-----+
```

この拡張機能は、現在の診断領域にのみ適用されます。GET STACKED DIAGNOSTICS は現在のコンテキストが条件ハンドラである場合にのみ許可されるため、2 番目の診断領域には適用されません。そうでない場合は、GET STACKED DIAGNOSTICS when handler not active エラーが発生します。

診断領域については、セクション 13.6.7.7 「MySQL の診断領域」を参照してください。簡単に言うと、ここには次の 2 種類の情報が含まれています。

- 発生した条件の数や、影響を受けた行数などのステートメント情報。
- エラーコードやメッセージなどの条件情報。ステートメントが複数の条件を発生させた場合、診断領域のこの部分には条件ごとの条件領域が含まれています。ステートメントがどの条件も発生させない場合、診断領域のこの部分は空です。

3 つの条件を生成するステートメントの場合、診断領域には、次のようなステートメント情報と条件情報が含まれています。

```
Statement information:
  row count
  ... other statement information items ...
Condition area list:
Condition area 1:
  error code for condition 1
  error message for condition 1
  ... other condition information items ...
Condition area 2:
  error code for condition 2:
  error message for condition 2
  ... other condition information items ...
Condition area 3:
  error code for condition 3
  error message for condition 3
  ... other condition information items ...
```

GET DIAGNOSTICS はステートメントまたは条件情報のどちらかを取得できますが、同じステートメントで両方取得することはできません。

- ステートメント情報を取得するには、目的のステートメント項目をターゲット変数に取得します。 **GET DIAGNOSTICS** の次の例では、使用可能な条件の数と影響を受けた行数をユーザー変数 `@p1` と `@p2` に割り当てます。

```
GET DIAGNOSTICS @p1 = NUMBER, @p2 = ROW_COUNT;
```

- 条件情報を取得するには、条件番号を指定し、目的の条件項目をターゲット変数に取得します。 **GET DIAGNOSTICS** の次の例では、SQLSTATE 値とエラーメッセージをユーザー変数 `@p3` と `@p4` に割り当てます。

```
GET DIAGNOSTICS CONDITION 1
  @p3 = RETURNED_SQLSTATE, @p4 = MESSAGE_TEXT;
```

取得リストには、カンマで区切られた 1 つ以上の `target = item_name` 代入を指定します。各代入では、ターゲット変数と、このステートメントがステートメントまたは条件情報のどちらを取得するかに応じて `statement_information_item_name` または `condition_information_item_name` 指示子のどちらかを指定します。

項目情報を格納するための有効な `target` 指示子は、ストアードプロシージャやストアードファンクションのパラメータ、**DECLARE** で宣言されたストアードプログラムのローカル変数、ユーザー定義変数のいずれかです。

有効な `condition_number` 指示子は、ストアードプロシージャやストアードファンクションのパラメータ、**DECLARE** で宣言されたストアードプログラムのローカル変数、ユーザー定義変数、システム変数、リテラルのいずれかです。文字リテラルには、`_charset` イントロデューサを含めることができます。条件番号が 1 から、情報が含まれている条件領域の数までの範囲にない場合は、警告が発生します。この場合、この警告は、診断領域にその領域をクリアすることなく追加されます。

条件が発生した場合、MySQL は、**GET DIAGNOSTICS** で認識されるすべての条件アイテムを移入しません。例:

```
mysql> GET DIAGNOSTICS CONDITION 1
  @p5 = SCHEMA_NAME, @p6 = TABLE_NAME;
mysql> SELECT @p5, @p6;
+-----+-----+
| @p5 | @p6 |
+-----+-----+
|     |     |
+-----+-----+
```

標準 SQL では、複数の条件が存在する場合、最初の条件は前の SQL ステートメントに対して返された `SQLSTATE` 値に関連しています。MySQL では、これが保証されません。メインのエラーを取得するために、次のように行うことはできません。

```
GET DIAGNOSTICS CONDITION 1 @errno = MYSQL_ERRNO;
```

代わりに、まず条件数を取得し、次にそれを使用してどの条件番号を検査するかを指定します。

```
GET DIAGNOSTICS @cno = NUMBER;
GET DIAGNOSTICS CONDITION @cno @errno = MYSQL_ERRNO;
```

許可されるステートメント情報と条件情報の項目、および条件が発生したときにどの項目が移入されるかについては、[診断領域の情報項目](#)を参照してください。

ストアードプロシージャのコンテキストで **GET DIAGNOSTICS** と例外ハンドラを使用して、挿入操作の結果を評価する例を次に示します。挿入が成功した場合、このプロシージャは **GET DIAGNOSTICS** を使用して、影響を受けた行数を取得します。これは、現在の診断領域がクリアされていないがぎり、**GET DIAGNOSTICS** を複数回使用してステートメントに関する情報を取得できることを示しています。

```
CREATE PROCEDURE do_insert(value INT)
BEGIN
  -- Declare variables to hold diagnostics area information
  DECLARE code CHAR(5) DEFAULT '00000';
  DECLARE msg TEXT;
  DECLARE nrows INT;
  DECLARE result TEXT;
  -- Declare exception handler for failed insert
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
  BEGIN
```



```

GET DIAGNOSTICS CONDITION 1
code = RETURNED_SQLSTATE, msg = MESSAGE_TEXT;
END;

-- Perform the insert
INSERT INTO t1 (int_col) VALUES(value);
-- Check whether the insert was successful
IF code = '00000' THEN
  GET DIAGNOSTICS nrows = ROW_COUNT;
  SET result = CONCAT("insert succeeded, row count = ",nrows);
ELSE
  SET result = CONCAT("insert failed, error = ",code,", message = ",msg);
END IF;
-- Say what happened
SELECT result;
END;

```

`t1.int_col` が、**NOT NULL** として宣言された整数カラムであるとします。このプロシージャは、**NULL** 以外の値と **NULL** 値を挿入するために呼び出されると、それぞれ次の結果を生成します。

```

mysql> CALL do_insert(1);
+-----+
| result |
+-----+
| insert succeeded, row count = 1 |
+-----+

mysql> CALL do_insert(NULL);
+-----+
| result |
+-----+
| insert failed, error = 23000, message = Column 'int_col' cannot be null |
+-----+

```

条件ハンドラがアクティブ化されると、診断領域スタックへのプッシュが発生します:

- 最初の (現在の) 診断領域が 2 番目の (スタックされた) 診断領域になり、新しい現在の診断領域がそのコピーとして作成されます。
- ハンドラ内で **GET [CURRENT] DIAGNOSTICS** および **GET STACKED DIAGNOSTICS** を使用して、現在の診断領域およびスタック診断領域の内容にアクセスできます。
- 最初は、両方の診断領域が同じ結果を返すため、ハンドラをアクティブ化した条件に関する情報を現在の診断領域から取得できます。次の期間は、ハンドラ内で現在の診断領域を変更するステートメントを実行しません。
- ただし、ハンドラ内で実行されるステートメントは、現在の診断領域を変更し、通常の規則に従ってその内容をクリアおよび設定できます ([診断領域のクリアおよび移入方法](#) を参照)。

ハンドラのアクティブ化条件に関する情報を取得する信頼性の高い方法は、スタック診断領域を使用することです。スタック診断領域は、**RESIGNAL** 以外のハンドラ内で実行されているステートメントでは変更できません。現在の診断領域が設定およびクリアされるタイミングの詳細は、[セクション13.6.7.7「MySQLの診断領域」](#) を参照してください。

次の例は、ハンドラステートメントによって現在の診断領域が変更された後でも、ハンドラ内で **GET STACKED DIAGNOSTICS** を使用して処理された例外に関する情報を取得する方法を示しています。

ストアードプロシージャ `p()` 内で、**TEXT NOT NULL** カラムを含むテーブルに 2 つの値を挿入しようとした。最初の値は **NULL** 以外の文字列で、次の値は **NULL** です。このカラムでは **NULL** 値が禁止されているため、最初の挿入は成功しますが、次の挿入によって例外が発生します。このプロシージャには、空の文字列の挿入に **NULL** の挿入をマップする例外ハンドラが含まれています:

```

DROP TABLE IF EXISTS t1;
CREATE TABLE t1 (c1 TEXT NOT NULL);
DROP PROCEDURE IF EXISTS p;
delimiter //
CREATE PROCEDURE p ()
BEGIN
  -- Declare variables to hold diagnostics area information
  DECLARE errcount INT;
  DECLARE errno INT;

```

```

DECLARE msg TEXT;
DECLARE EXIT HANDLER FOR SQLEXCEPTION
BEGIN
-- Here the current DA is nonempty because no prior statements
-- executing within the handler have cleared it
GET CURRENT DIAGNOSTICS CONDITION 1
  errno = MYSQL_ERRNO, msg = MESSAGE_TEXT;
SELECT 'current DA before mapped insert' AS op, errno, msg;
GET STACKED DIAGNOSTICS CONDITION 1
  errno = MYSQL_ERRNO, msg = MESSAGE_TEXT;
SELECT 'stacked DA before mapped insert' AS op, errno, msg;

-- Map attempted NULL insert to empty string insert
INSERT INTO t1 (c1) VALUES("");

-- Here the current DA should be empty (if the INSERT succeeded),
-- so check whether there are conditions before attempting to
-- obtain condition information
GET CURRENT DIAGNOSTICS errcount = NUMBER;
IF errcount = 0
THEN
  SELECT 'mapped insert succeeded, current DA is empty' AS op;
ELSE
  GET CURRENT DIAGNOSTICS CONDITION 1
    errno = MYSQL_ERRNO, msg = MESSAGE_TEXT;
  SELECT 'current DA after mapped insert' AS op, errno, msg;
END IF ;
GET STACKED DIAGNOSTICS CONDITION 1
  errno = MYSQL_ERRNO, msg = MESSAGE_TEXT;
SELECT 'stacked DA after mapped insert' AS op, errno, msg;
END;
INSERT INTO t1 (c1) VALUES('string 1');
INSERT INTO t1 (c1) VALUES(NULL);
END;
//
delimiter ;
CALL p();
SELECT * FROM t1;

```

ハンドラがアクティブになると、現在の診断領域のコピーが診断領域スタックにプッシュされます。ハンドラは最初に現在の診断領域とスタック診断領域の内容を表示します。これらはどちらも最初は同じです:

```

+-----+
| op          | errno | msg                               |
+-----+
| current DA before mapped insert | 1048 | Column 'c1' cannot be null |
+-----+

+-----+
| op          | errno | msg                               |
+-----+
| stacked DA before mapped insert | 1048 | Column 'c1' cannot be null |
+-----+

```

GET DIAGNOSTICS ステートメントのあとに実行されるステートメントは、現在の診断領域をリセットできます。ステートメントは、現在の診断領域をリセットできます。たとえば、ハンドラは **NULL** 挿入を空の文字列挿入にマップし、結果を表示します。新しい挿入は成功し、現在の診断領域はクリアされますが、スタックされた診断領域は変更されず、ハンドラをアクティブ化した条件に関する情報が引き続き含まれます:

```

+-----+
| op          |
+-----+
| mapped insert succeeded, current DA is empty |
+-----+

+-----+
| op          | errno | msg                               |
+-----+
| stacked DA after mapped insert | 1048 | Column 'c1' cannot be null |
+-----+

```

条件ハンドラが終了すると、その現在の診断領域がスタックからポップされ、スタックされた診断領域がストアドロージャの現在の診断領域になります。

プロシージャが戻ると、テーブルには 2 つの行が含まれます。空の行は、空の文字列の挿入にマップされた `NULL` を挿入しようとした結果です:

```
+-----+
| c1 |
+-----+
| string 1 |
|      |
+-----+
```

前述の例では、現在の診断領域とスタック診断領域から情報を取得する条件ハンドラ内の最初の 2 つの `GET DIAGNOSTICS` ステートメントが同じ値を返します。これは、現在の診断領域をリセットするステートメントがハンドラ内で以前に実行された場合には当てはまりません。 `p()` が、前ではなくハンドラ定義内に `DECLARE` ステートメントを配置するようにリライトされるとします:

```
CREATE PROCEDURE p ()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    -- Declare variables to hold diagnostics area information
    DECLARE errcount INT;
    DECLARE errno INT;
    DECLARE msg TEXT;
    GET CURRENT DIAGNOSTICS CONDITION 1
      errno = MYSQL_ERRNO, msg = MESSAGE_TEXT;
    SELECT 'current DA before mapped insert' AS op, errno, msg;
    GET STACKED DIAGNOSTICS CONDITION 1
      errno = MYSQL_ERRNO, msg = MESSAGE_TEXT;
    SELECT 'stacked DA before mapped insert' AS op, errno, msg;
  ...

```

この場合、結果はバージョンに依存します:

- MySQL 5.7.2 より前では、`DECLARE` は現在の診断領域を変更しないため、最初の 2 つの `GET DIAGNOSTICS` ステートメントは `p()` の元のバージョンと同じ結果を返します。

MySQL 5.7.2 では、すべての非診断ステートメントが SQL 標準に従って診断領域に移入されるように作業が行われました。`DECLARE` はこれらのいずれかであるため、5.7.2 以上では、ハンドラの先頭で実行されている `DECLARE` ステートメントによって現在の診断領域がクリアされ、`GET DIAGNOSTICS` ステートメントによって異なる結果が生成されます:

```
+-----+-----+-----+
| op          | errno | msg |
+-----+-----+-----+
| current DA before mapped insert | NULL | NULL |
+-----+-----+-----+

+-----+-----+-----+
| op          | errno | msg |
+-----+-----+-----+
| stacked DA before mapped insert | 1048 | Column 'c1' cannot be null |
+-----+-----+-----+
```

ハンドラをアクティブ化した条件に関する情報を取得するときに条件ハンドラ内でこの問題を回避するには、現在の診断領域ではなく、スタック診断領域にアクセスしてください。

13.6.7.4 RESIGNAL ステートメント

```
RESIGNAL [condition_value]
  [SET signal_information_item
  [, signal_information_item] ...]

condition_value: {
  SQLSTATE [VALUE] sqlstate_value
  | condition_name
}

signal_information_item:
  condition_information_item_name = simple_value_specification
```

```
condition_information_item_name: {
  CLASS_ORIGIN
| SUBCLASS_ORIGIN
| MESSAGE_TEXT
| MYSQL_ERRNO
| CONSTRAINT_CATALOG
| CONSTRAINT_SCHEMA
| CONSTRAINT_NAME
| CATALOG_NAME
| SCHEMA_NAME
| TABLE_NAME
| COLUMN_NAME
| CURSOR_NAME
}
```

```
condition_name, simple_value_specification:
(see following discussion)
```

RESIGNAL は、ストアプロシージャやストアファンクションの内部にある複合ステートメント内の条件ハンドラ、トリガー、またはイベントの実行中に使用可能なエラー条件情報を渡します。**RESIGNAL** は、その情報の一部またはすべてを、渡す前に変更する可能性があります。**RESIGNAL** は **SIGNAL** に関連していますが、**SIGNAL** のように条件を発信する代わりに、**RESIGNAL** は既存の条件情報をおそらく、変更してから中継します。

RESIGNAL は、エラーを処理することと、エラー情報を返すことの両方を可能にします。それ以外の場合は、ハンドラ内の SQL ステートメントを実行することによって、そのハンドラのアクティブ化を発生させた情報が破棄されます。**RESIGNAL** は、指定されたハンドラが状況の一部を処理できる場合はプロシージャの一部を短くしてから、条件を「遡って」別のハンドラに渡すことができます。

RESIGNAL ステートメントの実行に必要な権限はありません。

RESIGNAL のすべての形式で、現在のコンテキストが条件ハンドラである必要があります。そうでない場合、**RESIGNAL** は不正であり、**RESIGNAL when handler not active** エラーが発生します。

診断領域から情報を取得するには、**GET DIAGNOSTICS** ステートメントを使用します (セクション13.6.7.3「**GET DIAGNOSTICS ステートメント**」を参照してください)。診断領域については、セクション13.6.7.7「**MySQL の診断領域**」を参照してください。

- [RESIGNAL の概要](#)
- [RESIGNAL のみ](#)
- [新しいシグナル情報を含む RESIGNAL](#)
- [条件値とオプションの新しいシグナル情報を含む RESIGNAL](#)
- [RESIGNAL には条件ハンドラのコンテキストが必要](#)

RESIGNAL の概要

RESIGNAL に対する `condition_value` と `signal_information_item` の定義やルールは、**SIGNAL** に対するものと同じです。たとえば、`condition_value` を `SQLSTATE` 値にすることができ、この値は、エラー、警告、または「見つからない」を示す場合があります。詳細は、セクション13.6.7.5「**SIGNAL ステートメント**」を参照してください。

RESIGNAL ステートメントは、どちらもオプションである `condition_value` と `SET` 句を受け取ります。このため、次のいくつかの使用法が考えられます。

- **RESIGNAL** のみ:

```
RESIGNAL;
```

- 新しいシグナル情報を含む **RESIGNAL**:

```
RESIGNAL SET signal_information_item [, signal_information_item] ...;
```

- 条件値と場合によっては新しいシグナル情報を含む **RESIGNAL**:

```
RESIGNAL condition_value
[SET signal_information_item [, signal_information_item] ...];
```

これらのユースケースはすべて、診断および条件領域の変更を発生させます。

- 診断領域には 1 つ以上の条件領域が含まれています。
- 条件領域には、`SQLSTATE` 値、`MYSQL_ERRNO`、`MESSAGE_TEXT` などの条件情報項目が含まれています。

診断領域のスタックがあります。ハンドラは制御を取得すると、診断領域をスタックの最上部にプッシュするため、ハンドラの実行中に次の 2 つの診断領域があります：

- 最初の (現在の) 診断領域。最後の診断領域のコピーとして開始されますが、ハンドラ内の現在の診断領域を変更する最初のステートメントによって上書きされます。
- ハンドラが制御を取得する前に設定された条件領域を持つ最後の (スタックされた) 診断領域。

診断領域内の条件領域の最大数は、`max_error_count` システム変数の値によって決定されます。 [診断領域関連のシステム変数](#)を参照してください。

RESIGNAL のみ

単純な `RESIGNAL` のみとは、「エラーを変更せずに渡す」ことを示します。これは最後の診断領域をリストアし、それを現在の診断領域にします。つまり、診断領域スタックを「ポップします」。

条件をキャッチする条件ハンドラ内での `RESIGNAL` のみの 1 つの使用法として、ほかのいくつかのアクションを実行したあと、元の条件情報 (ハンドラに入る前に存在していた情報) を変更せずに渡す方法があります。

例:

```
DROP TABLE IF EXISTS xx;
delimiter //
CREATE PROCEDURE p ()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    SET @error_count = @error_count + 1;
    IF @a = 0 THEN RESIGNAL; END IF;
  END;
  DROP TABLE xx;
END//
delimiter ;
SET @error_count = 0;
SET @a = 0;
CALL p();
```

`DROP TABLE xx` ステートメントが失敗したとします。診断領域スタックは次のようになります。

```
DA 1. ERROR 1051 (42S02): Unknown table 'xx'
```

次に、実行が `EXIT` ハンドラに入ります。このハンドラは最初に、診断領域をスタックの先頭にプッシュします。これで、診断領域スタックは次のようになります。

```
DA 1. ERROR 1051 (42S02): Unknown table 'xx'
DA 2. ERROR 1051 (42S02): Unknown table 'xx'
```

この時点で、最初の (現在の) 診断領域と 2 番目の (スタックされた) 診断領域の内容は同じです。最初の診断領域は、そのあとにハンドラ内で実行されるステートメントによって変更される可能性があります。

通常、プロシージャーステートメントは最初の診断領域をクリアします。`BEGIN` は例外です。これはクリアせず、何も行いません。`SET` は例外ではありません。これはクリアし、操作を実行して、「成功」の結果を生成します。これで、診断領域スタックは次のようになります。

```
DA 1. ERROR 0000 (00000): Successful operation
DA 2. ERROR 1051 (42S02): Unknown table 'xx'
```

この時点で、`@a = 0` の場合、`RESIGNAL` は診断領域スタックをポップします。これで、診断領域スタックは次のようになります。

```
DA 1. ERROR 1051 (42S02): Unknown table 'xx'
```

そして、これが呼び出し元に表示される内容です。

@a が 0 でない場合、このハンドラは単純に終了します。つまり、現在の診断領域はそれ以上使用されなくなる（「処理済み」になった）ため、破棄することが可能になり、スタックされた診断領域がふたたび現在の診断領域になります。診断領域スタックは次のようになります。

```
DA 1. ERROR 0000 (00000): Successful operation
```

詳細を調べると複雑に見えますが、最終結果はきわめて有効です。ハンドラは、そのハンドラのアクティブ化を発生させた条件に関する情報を破棄することなく実行できます。

新しいシグナル情報を含む RESIGNAL

SET 句を含む RESIGNAL は新しいシグナル情報を指定するため、このステートメントは、「エラーを変更してから渡す」ことを示します。

```
RESIGNAL SET signal_information_item [, signal_information_item] ...;
```

RESIGNAL のみと同様に、アイデアは、元の情報が出力されるように診断領域スタックをポップすることです。RESIGNAL のみとは異なり、SET 句で指定されたものはすべて変更されます。

例:

```
DROP TABLE IF EXISTS xx;
delimiter //
CREATE PROCEDURE p ()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    SET @error_count = @error_count + 1;
    IF @a = 0 THEN RESIGNAL SET MYSQL_ERRNO = 5; END IF;
  END;
  DROP TABLE xx;
END//
delimiter ;
SET @error_count = 0;
SET @a = 0;
CALL p();
```

前の説明から、RESIGNAL のみによって、診断領域スタックが次のようになることを思い出してください。

```
DA 1. ERROR 1051 (42S02): Unknown table 'xx'
```

RESIGNAL SET MYSQL_ERRNO = 5 ステートメントでは、代わりに、スタックが次のようになります。これが呼び出し元に表示される内容です。

```
DA 1. ERROR 5 (42S02): Unknown table 'xx'
```

つまり、エラー番号だけが変更され、ほかは何も変更されません。

RESIGNAL ステートメントはシグナル情報項目のいずれかまたはすべてを変更できるため、診断領域の最初の条件領域がまったく異なっているように見えます。

条件値とオプションの新しいシグナル情報を含む RESIGNAL

条件値を含む RESIGNAL は、「条件を現在の診断領域にプッシュする」ことを示します。SET 句が存在する場合は、エラー情報も変更されます。

```
RESIGNAL condition_value
[SET signal_information_item [, signal_information_item] ...];
```

この形式の RESIGNAL は最後の診断領域をリストアし、それを現在の診断領域にします。つまり、診断領域スタックを「ポップします」。これは、単純な RESIGNAL のみが行う動作と同じです。ただし、条件値またはシグナル情報に応じて、診断領域も変更されます。

例:

```
DROP TABLE IF EXISTS xx;
delimiter //
CREATE PROCEDURE p ()
BEGIN
```



```

DECLARE EXIT HANDLER FOR SQLEXCEPTION
BEGIN
  SET @error_count = @error_count + 1;
  IF @a = 0 THEN RESIGNAL SQLSTATE '45000' SET MYSQL_ERRNO=5; END IF;
END;
DROP TABLE xx;
END//
delimiter ;
SET @error_count = 0;
SET @a = 0;
SET @@max_error_count = 2;
CALL p();
SHOW ERRORS;

```

これは前の例に似ていて、その効果も同じですが、**RESIGNAL** が発生した場合は、現在の条件領域が最後には異なっているように見える点が異なります。(この条件が既存の条件を置き換えるのではなく、それに追加される理由は、条件値が使用されているためです。)

この **RESIGNAL** ステートメントには条件値 (**SQLSTATE '45000'**) が含まれているため、新しい条件領域が追加され、診断領域スタックは次のようになります。

```

DA 1. (condition 2) ERROR 1051 (42S02): Unknown table 'xx'
      (condition 1) ERROR 5 (45000) Unknown table 'xx'

```

この例での **CALL p()** と **SHOW ERRORS** の結果は次のとおりです。

```

mysql> CALL p();
ERROR 5 (45000): Unknown table 'xx'
mysql> SHOW ERRORS;
+-----+-----+-----+
| Level | Code | Message                |
+-----+-----+-----+
| Error | 1051 | Unknown table 'xx'    |
| Error | 5    | Unknown table 'xx'    |
+-----+-----+-----+

```

RESIGNAL には条件ハンドラのコンテキストが必要

RESIGNAL のすべての形式で、現在のコンテキストが条件ハンドラである必要があります。そうでない場合、**RESIGNAL** は不正であり、**RESIGNAL when handler not active** エラーが発生します。例:

```

mysql> CREATE PROCEDURE p () RESIGNAL;
Query OK, 0 rows affected (0.00 sec)

mysql> CALL p();
ERROR 1645 (0K000): RESIGNAL when handler not active

```

さらに難しい例を次に示します。

```

delimiter //
CREATE FUNCTION f () RETURNS INT
BEGIN
  RESIGNAL;
  RETURN 5;
END//
CREATE PROCEDURE p ()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION SET @a=f();
  SIGNAL SQLSTATE '55555';
END//
delimiter ;
CALL p();

```

RESIGNAL がスタッドファンクション **f()** 内に現れています。**f()** 自体は **EXIT** ハンドラのコンテキスト内で呼び出されますが、**f()** 内での実行は独自のコンテキストを持ち、それはハンドラのコンテキストではありません。そのため、**f()** 内での **RESIGNAL** によって、「handler not active」エラーが発生します。

13.6.7.5 SIGNAL ステートメント

```

SIGNAL condition_value
[SET signal_information_item

```

```
[, signal_information_item] ...]

condition_value: {
  SQLSTATE [VALUE] sqlstate_value
  | condition_name
}

signal_information_item:
  condition_information_item_name = simple_value_specification

condition_information_item_name: {
  CLASS_ORIGIN
  | SUBCLASS_ORIGIN
  | MESSAGE_TEXT
  | MYSQL_ERRNO
  | CONSTRAINT_CATALOG
  | CONSTRAINT_SCHEMA
  | CONSTRAINT_NAME
  | CATALOG_NAME
  | SCHEMA_NAME
  | TABLE_NAME
  | COLUMN_NAME
  | CURSOR_NAME
}

condition_name, simple_value_specification:
  (see following discussion)
```

SIGNAL は、エラー「を返す」ための方法です。**SIGNAL** は、ハンドラ、アプリケーションの外側の部分、またはクライアントにエラー情報を提供します。また、エラーの特性 (エラー番号、**SQLSTATE** 値、メッセージ) に対する制御も提供します。**SIGNAL** を使用しない場合は、存在しないテーブルを意図的に参照してルーチングエラーを返すようにする、などの回避方法に頼る必要があります。

SIGNAL ステートメントの実行に必要な権限はありません。

診断領域から情報を取得するには、**GET DIAGNOSTICS** ステートメントを使用します (セクション13.6.7.3「**GET DIAGNOSTICS ステートメント**」を参照してください)。診断領域については、セクション13.6.7.7「**MySQL の診断領域**」を参照してください。

- [SIGNAL の概要](#)
- [シグナルの条件情報項目](#)
- [ハンドラ、カーソル、およびステートメントに対するシグナルの影響](#)

SIGNAL の概要

SIGNAL ステートメント内の **condition_value** は、返されるエラー値を示します。これは、**SQLSTATE** 値 (5 文字の文字列リテラル) か、または以前に **DECLARE ... CONDITION** で定義された名前付き条件を参照する **condition_name** にすることができます (セクション13.6.7.1「**DECLARE ... CONDITION ステートメント**」を参照してください)。

SQLSTATE 値は、エラー、警告、または「見つからない」を示す場合があります。[シグナルの条件情報項目](#)で説明されているように、この値の最初の 2 文字はそのエラークラスを示します。一部のシグナル値はステートメントを終了させます。[ハンドラ、カーソル、およびステートメントに対するシグナルの影響](#)を参照してください。

'00' は成功を示し、エラーの通知には有効でないため、**SIGNAL** ステートメントの **SQLSTATE** 値をこのような値で始めるべきではありません。これは、**SQLSTATE** 値が **SIGNAL** ステートメントで直接、またはこのステートメントで参照されている名前付き条件のどちらかで指定されている場合にも当てはまります。この値が無効である場合は、**Bad SQLSTATE** エラーが発生します。

一般的な **SQLSTATE** 値を通知するには、'45000' を使用します。これは、「未処理のユーザー定義の例外」を示します。

SIGNAL ステートメントには、オプションで、複数のシグナル項目を含む **SET** 句が **condition_information_item_name = simple_value_specification** 割当てのリストにカンマで区切られて含まれます。

各 **condition_information_item_name** は、**SET** 句内で 1 回だけ指定できます。そうでない場合は、**Duplicate condition information item** エラーが発生します。

有効な `simple_value_specification` 指示子は、ストアードプロシージャやストアードファンクションのパラメータ、`DECLARE` で宣言されたストアードプログラムのローカル変数、ユーザー定義変数、システム変数、またはリテラルを使用して指定できます。文字リテラルには、`_charset` イントロデューサを含めることができます。

許可される `condition_information_item_name` 値については、[シグナルの条件情報項目](#) を参照してください。

次のプロシージャは、その入力パラメータである `pval` の値に応じて、エラーまたは警告を通知します。

```
CREATE PROCEDURE p (pval INT)
BEGIN
  DECLARE specialty CONDITION FOR SQLSTATE '45000';
  IF pval = 0 THEN
    SIGNAL SQLSTATE '01000';
  ELSEIF pval = 1 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'An error occurred';
  ELSEIF pval = 2 THEN
    SIGNAL specialty
    SET MESSAGE_TEXT = 'An error occurred';
  ELSE
    SIGNAL SQLSTATE '01000'
    SET MESSAGE_TEXT = 'A warning occurred', MYSQL_ERRNO = 1000;
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'An error occurred', MYSQL_ERRNO = 1001;
  END IF;
END;
```

`pval` が 0 である場合、`'01'` で始まる `SQLSTATE` 値は警告クラス内のシグナルであるため、`p()` は警告を通知します。この警告はプロシージャを終了させず、そのプロシージャから戻ったあとに `SHOW WARNINGS` で確認できません。

`pval` が 1 である場合、`p()` はエラーを通知し、`MESSAGE_TEXT` 条件情報項目を設定します。このエラーはプロシージャを終了させ、そのテキストがエラー情報とともに返されます。

`pval` が 2 である場合、この場合は名前付き条件を使用して `SQLSTATE` 値が指定されているにもかかわらず、同じエラーが通知されます。

`pval` がその他の任意の値である場合、`p()` は最初に警告を通知し、メッセージテキストとエラー番号の条件情報項目を設定します。この警告はプロシージャを終了させないため、実行が継続され、そのあと `p()` はエラーを通知します。このエラーはプロシージャを終了させます。警告によって設定されたメッセージテキストとエラー番号は、エラーによって設定された値によって置き換えられ、それがエラー情報とともに返されます。

`SIGNAL` は通常、ストアードプログラム内で使用されますが、これはハンドラのコンテキストの外部で許可される MySQL 拡張です。たとえば、`mysql` クライアントプログラムを呼び出す場合は、プロンプトで次のステートメントのいずれかを入力できます。

```
SIGNAL SQLSTATE '77777';

CREATE TRIGGER t_bi BEFORE INSERT ON t
  FOR EACH ROW SIGNAL SQLSTATE '77777';

CREATE EVENT e ON SCHEDULE EVERY 1 SECOND
  DO SIGNAL SQLSTATE '77777';
```

`SIGNAL` は、次のルールに従って実行されます。

`SIGNAL` ステートメントが特定の `SQLSTATE` 値を示している場合、その値は、指定された条件を通知するために使用されます。例:

```
CREATE PROCEDURE p (divisor INT)
BEGIN
  IF divisor = 0 THEN
    SIGNAL SQLSTATE '22012';
  END IF;
END;
```

`SIGNAL` ステートメントが名前付き条件を使用している場合、その条件は `SIGNAL` ステートメントに適用される何らかの範囲内で宣言される必要があり、また MySQL エラー番号ではなく `SQLSTATE` 値を使用して定義される必要があります。例:

```
CREATE PROCEDURE p (divisor INT)
BEGIN
  DECLARE divide_by_zero CONDITION FOR SQLSTATE '22012';
  IF divisor = 0 THEN
    SIGNAL divide_by_zero;
  END IF;
END;
```

その名前付き条件が **SIGNAL** ステートメントの範囲内に存在しない場合は、**Undefined CONDITION** エラーが発生します。

SIGNAL が、**SQLSTATE** 値ではなく MySQL エラー番号で定義された名前付き条件を参照している場合は、**SIGNAL/RESIGNAL can only use a CONDITION defined with SQLSTATE** エラーが発生します。次のステートメントを発行すると、名前付き条件が MySQL エラー番号に関連付けられているため、そのエラーが発生します。

```
DECLARE no_such_table CONDITION FOR 1051;
SIGNAL no_such_table;
```

特定の名前を持つ条件が異なる範囲で複数回宣言されている場合は、もっともローカルな範囲を持つ宣言が適用されます。次のプロシーチャーを考えてみます。

```
CREATE PROCEDURE p (divisor INT)
BEGIN
  DECLARE my_error CONDITION FOR SQLSTATE '45000';
  IF divisor = 0 THEN
    BEGIN
      DECLARE my_error CONDITION FOR SQLSTATE '22012';
      SIGNAL my_error;
    END;
  END IF;
  SIGNAL my_error;
END;
```

divisor が 0 である場合は、最初の **SIGNAL** ステートメントが実行されます。もっとも内側の **my_error** 条件宣言が適用され、**SQLSTATE '22012'** が発生します。

divisor が 0 でない場合は、2 番目の **SIGNAL** ステートメントが実行されます。もっとも外側の **my_error** 条件宣言が適用され、**SQLSTATE '45000'** が発生します。

条件が発生したときにサーバーがハンドラを選択する方法については、[セクション13.6.7.6「ハンドラの範囲に関するルール」](#)を参照してください。

シグナルが例外ハンドラ内で発生する場合があります。

```
CREATE PROCEDURE p ()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    SIGNAL SQLSTATE VALUE '99999'
    SET MESSAGE_TEXT = 'An error occurred';
  END;
  DROP TABLE no_such_table;
END;
```

CALL p() が **DROP TABLE** ステートメントに達します。 **no_such_table** という名前のテーブルが存在しないため、エラーハンドラがアクティブ化されます。エラーハンドラは元のエラー（「このようなテーブルがない」）を破棄し、**SQLSTATE '99999'** の新しいエラーとメッセージ **An error occurred** を作成します。

シグナルの条件情報項目

次の表は、**SIGNAL** (または **RESIGNAL**) ステートメントで設定できる診断領域条件情報項目の名前を一覧表示しています。MySQL 拡張である **MYSQL_ERRNO** を除き、すべての項目が標準 SQL です。これらの項目の詳細は、[セクション13.6.7.7「MySQL の診断領域」](#)を参照してください。

Item Name	Definition
CLASS_ORIGIN	VARCHAR(64)
SUBCLASS_ORIGIN	VARCHAR(64)
CONSTRAINT_CATALOG	VARCHAR(64)
CONSTRAINT_SCHEMA	VARCHAR(64)

```

CONSTRAINT_NAME  VARCHAR(64)
CATALOG_NAME     VARCHAR(64)
SCHEMA_NAME      VARCHAR(64)
TABLE_NAME       VARCHAR(64)
COLUMN_NAME      VARCHAR(64)
CURSOR_NAME      VARCHAR(64)
MESSAGE_TEXT     VARCHAR(128)
MYSQL_ERRNO      SMALLINT UNSIGNED

```

文字項目の文字セットは UTF-8 です。

SIGNAL ステートメント内で条件情報項目に **NULL** を割り当てることはできません。

SIGNAL ステートメントは直接にか、または **SQLSTATE** 値で定義された名前付き条件を参照することによって間接的にかにかかわらず、常に **SQLSTATE** 値を指定します。**SQLSTATE** 値の最初の 2 文字はそのクラスであり、このクラスによって、その条件情報項目のデフォルト値が決定されます。

- クラス = '00' (成功)

不正です。'00' で始まる **SQLSTATE** 値は成功を示すため、**SIGNAL** には有効ではありません。

- クラス = '01' (警告)

```

MESSAGE_TEXT = 'Unhandled user-defined warning condition';
MYSQL_ERRNO = ER_SIGNAL_WARN

```

- クラス = '02' (見つからない)

```

MESSAGE_TEXT = 'Unhandled user-defined not found condition';
MYSQL_ERRNO = ER_SIGNAL_NOT_FOUND

```

- クラス > '02' (例外)

```

MESSAGE_TEXT = 'Unhandled user-defined exception condition';
MYSQL_ERRNO = ER_SIGNAL_EXCEPTION

```

正当なクラスの場合は、その他の条件情報項目が次のように設定されます。

```

CLASS_ORIGIN = SUBCLASS_ORIGIN = ";
CONSTRAINT_CATALOG = CONSTRAINT_SCHEMA = CONSTRAINT_NAME = ";
CATALOG_NAME = SCHEMA_NAME = TABLE_NAME = COLUMN_NAME = ";
CURSOR_NAME = ";

```

SIGNAL が実行されたあとにアクセスできるエラー値は、**SIGNAL** ステートメントによって発生した **SQLSTATE** 値と、**MESSAGE_TEXT** および **MYSQL_ERRNO** 項目です。これらの値は、次の C API から取得できます。

- `mysql_sqlstate()` は、**SQLSTATE** 値を返します。
- `mysql_errno()` は、**MYSQL_ERRNO** 値を返します。
- `mysql_error()` は、**MESSAGE_TEXT** 値を返します。

SQL レベルでは、**SHOW WARNINGS** および **SHOW ERRORS** からの出力に、**Code** および **Message** カラムの **MYSQL_ERRNO** および **MESSAGE_TEXT** の値が表示されます。

診断領域から情報を取得するには、**GET DIAGNOSTICS** ステートメントを使用します ([セクション13.6.7.3「GET DIAGNOSTICS ステートメント」](#)を参照してください)。診断領域については、[セクション13.6.7.7「MySQL の診断領域」](#)を参照してください。

ハンドラ、カーソル、およびステートメントに対するシグナルの影響

ステートメントの実行に対するシグナルの影響は、そのシグナルのクラスによって異なります。このクラスによって、エラーの重大性が決定されます。MySQL は、`sql_mode` システム変数の値を無視します。特に、厳密な SQL モードは問題になりません。MySQL は **IGNORE** も無視します。**SIGNAL** の目的はユーザーが生成したエラーを明示的に発生させることであるため、シグナルが無視されることはありません。

次の説明で、「未処理」は、通知された **SQLSTATE** 値に対するハンドラが **DECLARE ... HANDLER** で定義されていないことを示します。

- クラス = '00' (成功)

不正です。'00' で始まる `SQLSTATE` 値は成功を示すため、`SIGNAL` には有効ではありません。

- クラス = '01' (警告)

`warning_count` システム変数の値が増やされます。 `SHOW WARNINGS` がシグナルを示します。 `SQLWARNING` ハンドラがシグナルをキャッチします。

ストアドファンクションから警告を返すことはできません。関数を戻す `RETURN` ステートメントによって診断領域がクリアされるためです。したがって、このステートメントはそこに存在する可能性のある警告をすべてクリアします (さらに、`warning_count` を 0 にリセットします)。

- クラス = '02' (見つからない)

`NOT FOUND` ハンドラがシグナルをキャッチします。カーソルには影響しません。ストアドファンクションでシグナルが処理されない場合、ステートメントは終了します。

- クラス = '02' (例外)

`SQLEXCEPTION` ハンドラがシグナルをキャッチします。ストアドファンクションでシグナルが処理されない場合、ステートメントは終了します。

- クラス = '40'

通常の例外として処理されます。

13.6.7.6 ハンドラのスコープに関するルール

ストアドプログラムには、そのプログラム内で特定の条件が発生したときに呼び出されるハンドラを含めることができます。各ハンドラの適用性は、プログラム定義の中でのそのハンドラの場所や、そのハンドラが処理する 1 つまたは複数の条件によって異なります。

- `BEGIN ... END` ブロック内で宣言されたハンドラは、そのブロック内でハンドラ宣言のあとにある SQL ステートメントのスコープ内にしかありません。そのハンドラ自体が条件を発生させた場合は、そのハンドラも、そのブロック内で宣言されているほかのどのハンドラもその条件を処理できません。次の例では、ハンドラ `H1` と `H2` は、ステートメント `stmt1` および `stmt2` によって発生した条件のスコープ内にあります。ただし、`H1` も `H2` も、`H1` または `H2` の本体で発生した条件のスコープ内にはありません。

```
BEGIN -- outer block
  DECLARE EXIT HANDLER FOR ...; -- handler H1
  DECLARE EXIT HANDLER FOR ...; -- handler H2
  stmt1;
  stmt2;
END;
```

- ハンドラは、それが宣言されているブロックのスコープ内にしかなく、そのブロックの外部で発生した条件に対してアクティブ化することはできません。次の例では、ハンドラ `H1` は内側のブロックにある `stmt1` のスコープ内にありますが、外側のブロックにある `stmt2` のスコープ内にはありません。

```
BEGIN -- outer block
  BEGIN -- inner block
    DECLARE EXIT HANDLER FOR ...; -- handler H1
    stmt1;
  END;
  stmt2;
END;
```

- ハンドラは、特定のハンドラまたは一般的なハンドラのどちらかです。特定のハンドラとは、MySQL エラーコード、`SQLSTATE` 値、または条件名を処理するためのものです。一般的なハンドラとは、`SQLWARNING`、`SQLEXCEPTION`、または `NOT FOUND` クラス内の条件を処理するためのものです。あとで説明されているように、条件の特定性は条件の優先順位に関連しています。

複数のハンドラを異なるスコープ内で、かつ異なる特定性で宣言できます。たとえば、外側のブロックには特定の MySQL エラーコードハンドラが、また内側のブロックには一般的な `SQLWARNING` ハンドラが存在する可能性があ

ります。あるいは、特定の MySQL エラーコードのハンドラと、一般的な `SQLWARNING` クラスのハンドラが同じブロック内に存在することもあります。

あるハンドラがアクティブ化されるかどうかは、それ自体のスコープや条件値だけでなく、ほかにどのようなハンドラが存在するかによっても異なります。ストアプログラム内で条件が発生すると、サーバーは、適用可能なハンドラを現在のスコープ (現在の `BEGIN ... END` ブロック) 内で検索します。適用可能なハンドラが存在しない場合は、連続した包含する各スコープ (ブロック) 内のハンドラに関して外側に検索を続行します。特定のスコープで適用可能なハンドラを 1 つ以上見つけると、サーバーは、次の条件の優先順位に基づいてそれらのハンドラから選択します。

- MySQL エラーコードハンドラは、`SQLSTATE` 値ハンドラより優先されます。
- `SQLSTATE` 値ハンドラは、一般的な `SQLWARNING`、`SQLEXCEPTION`、または `NOT FOUND` ハンドラより優先されます。
- `SQLEXCEPTION` ハンドラは、`SQLWARNING` ハンドラより優先されます。
- 同じ優先順位を持つ適用可能なハンドラが複数存在する可能性があります。たとえば、ステートメントが、それぞれに対してエラー固有のハンドラが存在する、異なるエラーコードを持つ複数の警告を生成する可能性があります。この場合、サーバーがアクティブ化するハンドラを選択は非決定的であり、条件が発生する状況に応じて変わる可能性があります。

ハンドラ選択ルールの 1 つの側面として、複数の適用可能なハンドラが異なるスコープ内に存在する場合は、もっともローカルなスコープを持つハンドラが外側のスコープにあるハンドラより (それが、より具体的な条件のハンドラであっても) 優先される点があります。

ある条件が発生したときに適切なハンドラが存在しない場合、実行されるアクションはその条件のクラスによって異なります。

- `SQLEXCEPTION` 条件の場合は、`EXIT` ハンドラが存在するかのように、ストアプログラムはその条件を発生させたステートメントで終了します。そのプログラムが別のストアプログラムから呼び出されていた場合は、呼び出し元プログラムが、独自のハンドラに適用されるハンドラ選択ルールを使用してその条件を処理します。
- `SQLWARNING` 条件の場合は、`CONTINUE` ハンドラが存在するかのように、プログラムは実行を続行します。
- `NOT FOUND` 条件では、その条件が正常に発生した場合、アクションは `CONTINUE` です。 `SIGNAL` または `RESIGNAL` によって発生した場合、アクションは `EXIT` です。

次の例は、MySQL によってハンドラ選択ルールがどのように適用されるかを示しています。

次のプロシージャーには 2 つのハンドラが含まれています。つまり、存在しないテーブルを削除しようとする試みに対して発生する特定の `SQLSTATE` 値 ('42S02') 用に 1 つと、一般的な `SQLEXCEPTION` クラス用に 1 つです。

```
CREATE PROCEDURE p1()
BEGIN
  DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02'
    SELECT 'SQLSTATE handler was activated' AS msg;
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
    SELECT 'SQLEXCEPTION handler was activated' AS msg;

  DROP TABLE test.t;
END;
```

両方のハンドラが同じブロック内で宣言され、同じスコープを持っています。ただし、`SQLSTATE` ハンドラは `SQLEXCEPTION` ハンドラより優先されるため、テーブル `t` が存在しない場合、`DROP TABLE` ステートメントは `SQLSTATE` ハンドラをアクティブ化する条件を発生させます。

```
mysql> CALL p1();
+-----+
| msg          |
+-----+
| SQLSTATE handler was activated |
+-----+
```

次のプロシージャーにも、同じ 2 つのハンドラが含まれています。ただし、今回は、`DROP TABLE` ステートメントと `SQLEXCEPTION` ハンドラが `SQLSTATE` ハンドラに対して内側のブロック内にあります。

```
CREATE PROCEDURE p2()
BEGIN -- outer block
```

```

DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02'
  SELECT 'SQLSTATE handler was activated' AS msg;
BEGIN -- inner block
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
    SELECT 'SQLEXCEPTION handler was activated' AS msg;

  DROP TABLE test.t; -- occurs within inner block
END;
END;

```

この場合は、条件が発生した場所に対してよりローカルなハンドラが優先されます。SQLSTATE ハンドラより一般的であるにもかかわらず、SQLEXCEPTION ハンドラがアクティブ化されます。

```

mysql> CALL p2();
+-----+
| msg                |
+-----+
| SQLEXCEPTION handler was activated |
+-----+

```

次のプロシージャでは、ハンドラの 1 つが、DROP TABLE ステートメントのスコープに対して内側のブロック内で宣言されています。

```

CREATE PROCEDURE p3()
BEGIN -- outer block
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
    SELECT 'SQLEXCEPTION handler was activated' AS msg;
  BEGIN -- inner block
    DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02'
      SELECT 'SQLSTATE handler was activated' AS msg;
  END;

  DROP TABLE test.t; -- occurs within outer block
END;

```

もう一方のハンドラが DROP TABLE によって発生した条件のスコープ内にないため、SQLEXCEPTION ハンドラのみが適用されます。

```

mysql> CALL p3();
+-----+
| msg                |
+-----+
| SQLEXCEPTION handler was activated |
+-----+

```

次のプロシージャでは、両方のハンドラが、DROP TABLE ステートメントのスコープに対して内側のブロック内で宣言されています。

```

CREATE PROCEDURE p4()
BEGIN -- outer block
  BEGIN -- inner block
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
      SELECT 'SQLEXCEPTION handler was activated' AS msg;
    DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02'
      SELECT 'SQLSTATE handler was activated' AS msg;
  END;

  DROP TABLE test.t; -- occurs within outer block
END;

```

DROP TABLE のスコープ内にないため、どちらのハンドラも適用されません。このステートメントによって発生した条件は未処理になり、プロシージャをエラーで終了させます。

```

mysql> CALL p4();
ERROR 1051 (42S02): Unknown table 'test.t'

```

13.6.7.7 MySQL の診断領域

SQL ステートメントは、診断領域を移入する診断情報を生成します。標準 SQL には、ネスト化された実行のコンテキストごとの診断領域を含んだ、診断領域スタックがあります。標準 SQL はまた、条件ハンドラの実行中に 2 番目の診断領域を参照するための GET STACKED DIAGNOSTICS 構文もサポートしています。

次の説明では、MySQL の診断領域の構造、MySQL で認識される情報項目、ステートメントが診断領域をクリアおよび設定する方法、およびスタックに対して診断領域をプッシュおよびポップする方法について説明します。

- [診断領域の構造](#)
- [診断領域の情報項目](#)
- [診断領域のクリアおよび移入方法](#)
- [診断領域スタックの動作](#)
- [診断領域関連のシステム変数](#)

診断領域の構造

診断領域には、次の 2 種類の情報が含まれています。

- 発生した条件の数や、影響を受けた行数などのステートメント情報。
- エラーコードやメッセージなどの条件情報。ステートメントが複数の条件を発生させた場合、診断領域のこの部分には条件ごとの条件領域が含まれています。ステートメントがどの条件も発生させない場合、診断領域のこの部分は空です。

3 つの条件を生成するステートメントの場合、診断領域には、次のようなステートメント情報と条件情報が含まれています。

```
Statement information:
row count
... other statement information items ...
Condition area list:
Condition area 1:
error code for condition 1
error message for condition 1
... other condition information items ...
Condition area 2:
error code for condition 2:
error message for condition 2
... other condition information items ...
Condition area 3:
error code for condition 3
error message for condition 3
... other condition information items ...
```

診断領域の情報項目

診断領域には、ステートメント情報と条件情報項目が含まれています。数値項目は整数です。文字項目の文字セットは UTF-8 です。どの項目も `NULL` にはできません。診断領域を移入するステートメントによってステートメントまたは条件項目が設定されていない場合、その値は、項目のデータ型に応じて 0 または空の文字列になります。

診断領域のステートメント情報の部分には、次の項目が含まれています。

- **NUMBER**: 情報が含まれている条件領域の数を示す整数。
- **ROW_COUNT**: このステートメントによって影響を受けた行数を示す整数。**ROW_COUNT** には、**ROW_COUNT()** 関数と同じ値が含まれています ([セクション 12.16 「情報関数」](#) を参照してください)。

診断領域の条件情報の部分には、条件ごとの条件領域が含まれています。条件領域には、1 から **NUMBER** ステートメント条件項目の値の番号が付けられています。**NUMBER** が 0 である場合、条件領域は存在しません。

各条件領域には、次のリスト内の項目が含まれています。MySQL 拡張である **MYSQL_ERRNO** を除き、すべての項目が標準 SQL です。これらの定義は、シグナル以外によって (つまり、**SIGNAL** または **RESIGNAL** ステートメントによって) 生成された条件に適用されます。シグナル以外の条件の場合、MySQL は、常に空であるとは示されていない条件項目のみを移入します。条件領域に対するシグナルの影響については、あとで説明されています。

- **CLASS_ORIGIN RETURNED_SQLSTATE** 値のクラスを含む文字列。**RETURNED_SQLSTATE** 値が SQL 標準のドキュメント ISO 9075-2 (セクション 24.1, **SQLSTATE**) で定義されているクラス値で始まる場合、**CLASS_ORIGIN** は **'ISO 9075'** です。それ以外の場合、**CLASS_ORIGIN** は **'MySQL'** です。

- **SUBCLASS_ORIGIN**: **RETURNED_SQLSTATE** 値のサブクラスを含む文字列。 **CLASS_ORIGIN** が 'ISO 9075' であるか、または **RETURNED_SQLSTATE** が '000' で終わる場合、 **SUBCLASS_ORIGIN** は 'ISO 9075' です。 それ以外の場合、 **SUBCLASS_ORIGIN** は 'MySQL' です。
- **RETURNED_SQLSTATE**: この条件の **SQLSTATE** 値を示す文字列。
- **MESSAGE_TEXT**: この条件のエラーメッセージを示す文字列。
- **MYSQL_ERRNO**: この条件の MySQL エラーコードを示す整数。
- **CONSTRAINT_CATALOG**、 **CONSTRAINT_SCHEMA**、 **CONSTRAINT_NAME**: 違反した制約のカatalog、スキーマ、および名前を示す文字列。 これらは常に空です。
- **CATALOG_NAME**、 **SCHEMA_NAME**、 **TABLE_NAME**、 **COLUMN_NAME**: この条件に関連したCatalog、スキーマ、テーブル、およびカラムを示す文字列。 これらは常に空です。
- **CURSOR_NAME**: カーソル名を示す文字列。 これは常に空です。

RETURNED_SQLSTATE、 **MESSAGE_TEXT**、 および **MYSQL_ERRNO** 値の特定のエラーについては、 [Server Error Message Reference](#) を参照してください。

SIGNAL (または **RESIGNAL**) ステートメントが診断領域を移入する場合、その **SET** 句は、 **RETURNED_SQLSTATE** を除く任意の条件情報項目に、その項目のデータ型に対して正当な任意の値を割り当てることができます。 **SIGNAL** はまた、 **RETURNED_SQLSTATE** 値も設定しますが、その **SET** 句で直接設定するわけではありません。その値は、 **SIGNAL** ステートメントの **SQLSTATE** 引数から取得されます。

SIGNAL は、ステートメント情報項目も設定します。 **NUMBER** を 1 に設定します。エラーの場合は **ROW_COUNT** を -1 に設定し、それ以外の場合は 0 に設定します。

診断領域のクリアおよび移入方法

診断以外の SQL ステートメントは診断領域に自動的に移入され、その内容は **SIGNAL** および **RESIGNAL** ステートメントを使用して明示的に設定できます。診断領域は、特定の項目を抽出するために **GET DIAGNOSTICS** を使用して、あるいは条件またはエラーを確認するために **SHOW WARNINGS** または **SHOW ERRORS** を使用して検査できます。

SQL ステートメントは、次のように診断領域をクリアおよび設定します。

- 解析後にサーバーがステートメントの実行を開始すると、非診断ステートメントの診断領域がクリアされます。診断ステートメントは診断領域をクリアしません。次のステートメントは診断です:
 - **GET DIAGNOSTICS**
 - **SHOW ERRORS**
 - **SHOW WARNINGS**
- ステートメントが条件を発生させた場合は、以前のステートメントに属する条件の診断領域がクリアされます。例外として、 **GET DIAGNOSTICS** および **RESIGNAL** によって発生した条件は、診断領域にその領域をクリアすることなく追加されます。

そのため、通常は実行開始時に診断領域をクリアしないステートメントであっても、そのステートメントが条件を発生させた場合は診断領域をクリアします。

次の例は、診断領域に対するさまざまなステートメントの影響を、 **SHOW WARNINGS** を使用して、そこに格納されている条件に関する情報を表示することによって示しています。

次の **DROP TABLE** ステートメントは、診断領域をクリアし、条件が発生すると診断領域に移入します:

```
mysql> DROP TABLE IF EXISTS test.no_such_table;
Query OK, 0 rows affected, 1 warning (0.01 sec)

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Note | 1051 | Unknown table 'test.no_such_table' |
```

```
+-----+
1 row in set (0.00 sec)
```

次の **SET** ステートメントはエラーを生成するため、診断領域をクリアして移入します。

```
mysql> SET @x = @@x;
ERROR 1193 (HY000): Unknown system variable 'x'
```

```
mysql> SHOW WARNINGS;
+-----+
| Level | Code | Message |
+-----+
| Error | 1193 | Unknown system variable 'x' |
+-----+
1 row in set (0.00 sec)
```

前の **SET** ステートメントは 1 つの条件を生成したため、1 がこの時点での **GET DIAGNOSTICS** の唯一の有効な条件番号です。次のステートメントは 2 の条件番号を使用しています。これにより、診断領域にその領域をクリアすることなく追加される警告が生成されます。

```
mysql> GET DIAGNOSTICS CONDITION 2 @p = MESSAGE_TEXT;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

```
mysql> SHOW WARNINGS;
+-----+
| Level | Code | Message |
+-----+
| Error | 1193 | Unknown system variable 'xx' |
| Error | 1753 | Invalid condition number |
+-----+
2 rows in set (0.00 sec)
```

これで、診断領域には 2 つの条件が存在するようになったため、同じ **GET DIAGNOSTICS** ステートメントが成功します。

```
mysql> GET DIAGNOSTICS CONDITION 2 @p = MESSAGE_TEXT;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT @p;
+-----+
| @p |
+-----+
| Invalid condition number |
+-----+
1 row in set (0.01 sec)
```

診断領域スタックの動作

診断領域スタックへのプッシュが発生すると、最初の (現在の) 診断領域が 2 番目の (スタックされた) 診断領域になり、新しい現在の診断領域がそのコピーとして作成されます。診断領域は、次の状況でスタックにプッシュされ、スタックからポップされます:

- ストアドプログラムの実行

プログラムが実行される前にプッシュが発生し、その後ポップが発生します。ハンドラの実行中にストアドプログラムが終了した場合、ポップする診断領域が複数存在する可能性があります。これは、適切なハンドラがない例外またはハンドラ内の **RETURN** が原因で発生します。

ポップされた診断領域の警告またはエラーの状態は、トリガーの場合はエラーのみが追加されることを除き、現在の診断領域に追加されます。ストアドプログラムが終了すると、呼び出し元の現在の診断領域にこれらの状態が表示されます。

- ストアドプログラム内での条件ハンドラの実行

条件ハンドラのアクティブ化の結果としてプッシュが発生した場合、スタック診断領域はプッシュ前にストアドプログラム内で最新だった領域です。新しい非現在の診断領域は、ハンドラの現在の診断領域です。ハンドラ内で **GET [CURRENT] DIAGNOSTICS** および **GET STACKED DIAGNOSTICS** を使用して、現在 (ハンドラ) およびスタック (ストアドプログラム) の診断領域の内容にアクセスできます。最初は同じ結果が返されますが、ハンドラ内で実行されているステートメントは現在の診断領域を変更し、通常の規則に従ってその内容をクリアおよび設定し

ます (診断領域のクリアおよび移入方法を参照)。スタック診断領域は、`RESIGNAL` を除くハンドラ内で実行されているステートメントでは変更できません。

ハンドラが正常に実行されると、現在の (ハンドラ) 診断領域がポップされ、スタック (ストアプログラム) 診断領域が再度現在の診断領域になります。ハンドラの実行中にハンドラ診断領域に追加された条件は、現在の診断領域に追加されます。

- `RESIGNAL` の実行

`RESIGNAL` ステートメントは、ストアプログラム内の複合ステートメント内で条件ハンドラの実行中に使用可能なエラー条件情報を渡します。`RESIGNAL` は、情報の一部またはすべてを変更してから渡すことができ、[セクション 13.6.7.4 「RESIGNAL ステートメント」](#) の説明に従って診断スタックを変更します。

診断領域関連のシステム変数

特定のシステム変数が診断領域のいくつかの側面を制御するか、またはそれに関連しています。

- `max_error_count` は、診断領域内の条件領域の数を制御します。これより多い条件が発生した場合、MySQL は、超過した条件に関する情報を暗黙のうちに破棄します。(`RESIGNAL` によって追加された条件は、常に追加されます。空きを作るために、古い条件が必要に応じて破棄されます。)
- `warning_count` は、発生した条件の数を示します。これには、エラー、警告、および注意が含まれます。通常、`NUMBER` と `warning_count` は同じです。ただし、生成された条件の数が `max_error_count` を超えると、診断領域にはそれ以上の条件が格納されないため、`warning_count` の値が引き続き増えるのに対して、`NUMBER` は `max_error_count` に上限が設定されたままになります。
- `error_count` は、発生したエラーの数を示します。この値には「見つからない」と例外条件が含まれますが、警告と注意は除外されます。`warning_count` と同様に、その値は `max_error_count` を超えることができます。
- `sql_notes` システム変数が 0 に設定されている場合、注意は格納されず、`warning_count` も増分しません。

例: `max_error_count` が 10 である場合、診断領域には最大 10 個の条件領域を含めることができます。ステートメントが 20 個の条件を発生させ、そのうちの 12 個がエラーであるとして、その場合、診断領域には最初の 10 個の条件が含まれ、`NUMBER` は 10、`warning_count` は 20、`error_count` は 12 です。

`max_error_count` の値を変更しても、次に診断領域を変更しようとするまでは何の効果もありません。診断領域に 10 個の条件領域が含まれているときに `max_error_count` が 5 に設定された場合、その診断領域のサイズまたは内容への即座の影響は何もありません。

13.6.7.8 条件の処理と OUT または INOUT パラメータ

ストアプロシージャが未処理の例外で終了した場合、`OUT` および `INOUT` パラメータの変更された値はコール元に伝播されません。

`RESIGNAL` ステートメントを含む `CONTINUE` または `EXIT` ハンドラによって例外が処理される場合、`RESIGNAL` を実行すると診断領域スタックがポップされ、例外 (ハンドラへの入力前に存在していた情報) が通知されます。例外がエラーの場合、`OUT` および `INOUT` パラメータの値はコール元に伝播されません。

13.6.8 条件処理の制約

`SIGNAL`、`RESIGNAL`、および `GET DIAGNOSTICS` は準備済みのステートメントとして許可されていません。たとえば、次のステートメントは無効です。

```
PREPARE stmt1 FROM 'SIGNAL SQLSTATE "02000";'
```

クラス '04' の `SQLSTATE` 値は特別扱いされません。ほかの例外と同じように扱われます。

標準 SQL では、最初の条件は、以前の SQL ステートメントに対して返される `SQLSTATE` 値に関連します。MySQL ではこれは保証されていないので、メインエラーを取得するために、次のようにはできません。

```
GET DIAGNOSTICS CONDITION 1 @errno = MYSQL_ERRNO;
```

代わりに次のようにします。

```
GET DIAGNOSTICS @cno = NUMBER;
```



```
GET DIAGNOSTICS CONDITION @cno @errno = MYSQL_ERRNO;
```

13.7 データベース管理ステートメント

13.7.1 アカウント管理ステートメント

MySQL アカウント情報は、`mysql` システムスキーマのテーブルに格納されます。このデータベースとアクセス制御システムについては、[第5章「MySQL サーバーの管理」](#)で広範囲にわたって説明されています。詳細は、この章を参照するようにしてください。

重要

一部の MySQL リリースでは、新しい権限または機能を追加するために付与テーブルに変更が導入されています。新しい機能を実際に利用できるようにするには、MySQL をアップグレードするたびに付与テーブルを現在の構造に更新します。[セクション2.11「MySQL のアップグレード」](#)を参照してください。

`read_only` システム変数が有効になっている場合、`account-management` ステートメントには、他の必要な権限に加えて、`CONNECTION_ADMIN` 権限 (または非推奨の `SUPER` 権限) が必要です。これは、`mysql` システムスキーマのテーブルを変更するためです。

アカウント管理ステートメントはアトミックであり、クラッシュセーフです。詳細は、[セクション13.1.1「アトミックデータ定義ステートメントのサポート」](#)を参照してください。

13.7.1.1 ALTER USER ステートメント

```
ALTER USER [IF EXISTS]
  user [auth_option] [, user [auth_option]] ...
  [REQUIRE {NONE | tls_option [[AND] tls_option] ...}]
  [WITH resource_option [resource_option] ...]
  [password_option | lock_option] ...
  [COMMENT 'comment_string' | ATTRIBUTE 'json_object']

ALTER USER [IF EXISTS] USER() user_func_auth_option

ALTER USER [IF EXISTS]
  user DEFAULT ROLE
  {NONE | ALL | role [, role] ...}

user:
  (see セクション6.2.4「アカウント名の指定」)

auth_option: {
  IDENTIFIED BY 'auth_string'
    [REPLACE 'current_auth_string']
    [RETAIN CURRENT PASSWORD]
  | IDENTIFIED BY RANDOM PASSWORD
    [REPLACE 'current_auth_string']
    [RETAIN CURRENT PASSWORD]
  | IDENTIFIED WITH auth_plugin
  | IDENTIFIED WITH auth_plugin BY 'auth_string'
    [REPLACE 'current_auth_string']
    [RETAIN CURRENT PASSWORD]
  | IDENTIFIED WITH auth_plugin BY RANDOM PASSWORD
    [REPLACE 'current_auth_string']
    [RETAIN CURRENT PASSWORD]
  | IDENTIFIED WITH auth_plugin AS 'auth_string'
  | DISCARD OLD PASSWORD
}

user_func_auth_option: {
  IDENTIFIED BY 'auth_string'
    [REPLACE 'current_auth_string']
    [RETAIN CURRENT PASSWORD]
  | DISCARD OLD PASSWORD
}

tls_option: {
  SSL
```

```
| X509
| CIPHER 'cipher'
| ISSUER 'issuer'
| SUBJECT 'subject'
}

resource_option: {
  MAX_QUERIES_PER_HOUR count
| MAX_UPDATES_PER_HOUR count
| MAX_CONNECTIONS_PER_HOUR count
| MAX_USER_CONNECTIONS count
}

password_option: {
  PASSWORD EXPIRE [DEFAULT | NEVER | INTERVAL N DAY]
| PASSWORD HISTORY {DEFAULT | N}
| PASSWORD REUSE INTERVAL {DEFAULT | N DAY}
| PASSWORD REQUIRE CURRENT [DEFAULT | OPTIONAL]
| FAILED_LOGIN_ATTEMPTS N
| PASSWORD_LOCK_TIME {N | UNBOUNDED}
}

lock_option: {
  ACCOUNT LOCK
| ACCOUNT UNLOCK
}
```

ALTER USER ステートメントによって、MySQL アカウントが変更されます。既存のアカウントの認証、ロール、SSL/TLS、リソース制限およびパスワード管理プロパティを変更できます。また、アカウントのロックおよびロック解除にも使用できます。

ほとんどの場合、**ALTER USER** には `mysql` システムスキーマに対するグローバル **CREATE USER** 権限または **UPDATE** 権限が必要です。例外は次のとおりです：

- 匿名以外のアカウントを使用してサーバーに接続したクライアントはすべて、そのアカウントのパスワードを変更できます。(特に、自分のパスワードを変更できます。)サーバーが認証したアカウントを確認するには、`CURRENT_USER()` 関数を呼び出します：

```
SELECT CURRENT_USER();
```

- DEFAULT ROLE** 構文の場合、**ALTER USER** には次の権限が必要です：
 - 別のユーザーのデフォルトロールを設定するには、`mysql.default_roles` システムテーブルに対するグローバル **CREATE USER** 権限または **UPDATE** 権限が必要です。
 - 自分のデフォルトロールを設定する場合、デフォルトとして必要なロールが付与されているかぎり、特別な権限は必要ありません。
- セカンダリパスワードを変更するステートメントには、次の権限が必要です：
 - 自分のアカウントに適用される **ALTER USER** ステートメントに **RETAIN CURRENT PASSWORD** または **DISCARD OLD PASSWORD** 句を使用するには、`APPLICATION_PASSWORD_ADMIN` 権限が必要です。ほとんどのユーザーは 1 つのパスワードのみを必要とするため、自分のセカンダリパスワードを操作するには権限が必要です。
 - アカウントがすべてのアカウントのセカンダリパスワードの操作を許可される場合は、`APPLICATION_PASSWORD_ADMIN` ではなく **CREATE USER** 権限が必要です。

`read_only` システム変数が有効になっている場合、**ALTER USER** にはさらに `CONNECTION_ADMIN` 権限 (または非推奨の `SUPER` 権限) が必要です。

デフォルトでは、存在しないユーザーを変更しようとする、エラーが発生します。`IF EXISTS` 句を指定すると、ステートメントは、存在しない指定ユーザーごとに、エラーではなく、警告を生成します。

重要

状況によっては、**ALTER USER** はサーバーログまたはクライアント側の `~/.mysql_history` などの履歴ファイルに記録される場合があります。つまり、クリアテキストパスワードは、そ

の情報への読取りアクセス権を持つすべてのユーザーが読み取ることができます。これがサーバーログで発生する条件およびこれを制御する方法については、[セクション6.1.2.3「パスワードおよびロギング」](#)を参照してください。クライアント側のロギングに関する同様の情報については、[セクション4.5.1.3「mysql クライアントロギング」](#)を参照してください。

次のトピックで説明するように、`ALTER USER` ステートメントにはいくつかの側面があります:

- [ALTER USER の概要](#)
- [ALTER USER 認証オプション](#)
- [ALTER USER ロールのオプション](#)
- [ALTER USER SSL/TLS オプション](#)
- [ALTER USER リソース制限オプション](#)
- [ALTER USER のパスワード管理オプション](#)
- [ALTER USER アカウントロックオプション](#)
- [ALTER USER バイナリロギング](#)

ALTER USER の概要

影響を受けるアカウントごとに、`ALTER USER` は `mysql.user` システムテーブルの対応する行を変更して、ステートメントで指定されたプロパティを反映します。未指定のプロパティは現在の値を保持します。

各アカウント名には、[セクション6.2.4「アカウント名の指定」](#)で説明されている形式が使用されます。アカウント名のホスト名部分は、省略すると '%' にデフォルト設定されます。`CURRENT_USER` または `CURRENT_USER()` を指定して、現在のセッションに関連付けられているアカウントを参照することもできます。

一方の構文の場合のみ、`USER()` 関数でアカウントを指定できます:

```
ALTER USER USER() IDENTIFIED BY 'auth_string';
```

この構文を使用すると、アカウントに文字どおりの名前を付けずに自分のパスワードを変更できます。(構文では、[ALTER USER 認証オプション](#) で説明されている `REPLACE`、`RETAIN CURRENT PASSWORD` および `DISCARD OLD PASSWORD` 句もサポートされています。)

MySQL 8.0.21 以降では、[セクション13.7.1.3「CREATE USER ステートメント」](#)で説明されているように、ユーザーコメントおよびユーザー属性がサポートされます。これらは、それぞれ `COMMENT` および `ATTRIBUTE` オプションを使用して、`ALTER USER` を使用して変更できます。同じ `ALTER USER` ステートメントで両方のオプションを指定することはできません。指定しようとすると構文エラーが発生します。

ユーザーコメントおよびユーザー属性は、JSON オブジェクトとして `INFORMATION_SCHEMA.USER_ATTRIBUTES` テーブルに格納されます。この説明で後述するように、ユーザーコメントは、このテーブルの `ATTRIBUTE` カラムの `comment` キーの値として格納されます。`COMMENT` テキストは任意の任意の引用符付きテキストで、既存のユーザーコメントを置換できます。`ATTRIBUTE` 値は、JSON オブジェクトの有効な文字列表現である必要があります。これは、`JSON_MERGE_PATCH()` 関数が既存のユーザー属性および新しいユーザー属性で使用されているかのように、既存のユーザー属性とマージされます。再使用されるキーについては、次に示すように、新しい値によって古い値が上書きされます:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.USER_ATTRIBUTES
-> WHERE USER='bill' AND HOST='localhost';
+-----+-----+-----+
| USER | HOST   | ATTRIBUTE |
+-----+-----+-----+
| bill | localhost | {"foo": "bar"} |
+-----+-----+-----+
1 row in set (0.11 sec)

mysql> ALTER USER 'bill'@'localhost' ATTRIBUTE '{"baz": "faz", "foo": "moo"}';
Query OK, 0 rows affected (0.22 sec)

mysql> SELECT * FROM INFORMATION_SCHEMA.USER_ATTRIBUTES
-> WHERE USER='bill' AND HOST='localhost';
+-----+-----+-----+
```

```
| USER | HOST | ATTRIBUTE |
+-----+-----+-----+
| bill | localhost | {"baz": "faz", "foo": "moo"} |
+-----+-----+-----+
1 row in set (0.00 sec)
```

ユーザー属性からキーとその値を削除するには、次のように、キーを JSON `null` に設定します (小文字で引用符で囲まれていない必要があります):

```
mysql> ALTER USER 'bill'@'localhost' ATTRIBUTE '{"foo": null}';
Query OK, 0 rows affected (0.08 sec)

mysql> SELECT * FROM INFORMATION_SCHEMA.USER_ATTRIBUTES
-> WHERE USER='bill' AND HOST='localhost';
+-----+-----+-----+
| USER | HOST | ATTRIBUTE |
+-----+-----+-----+
| bill | localhost | {"baz": "faz"} |
+-----+-----+-----+
1 row in set (0.00 sec)
```

既存のユーザーコメントを空の文字列に設定するには、`ALTER USER ... COMMENT ''`を使用します。これにより、`USER_ATTRIBUTES` テーブルに空の `comment` 値が残されます。ユーザーコメントを完全に削除するには、カラムキーの値を JSON `null` に設定して `ALTER USER ... ATTRIBUTE ...` を使用します (小文字では引用符で囲まれていません)。これは、次の一連の SQL ステートメントで示されています:

```
mysql> ALTER USER 'bill'@'localhost' COMMENT 'Something about Bill';
Query OK, 0 rows affected (0.06 sec)

mysql> SELECT * FROM INFORMATION_SCHEMA.USER_ATTRIBUTES
-> WHERE USER='bill' AND HOST='localhost';
+-----+-----+-----+
| USER | HOST | ATTRIBUTE |
+-----+-----+-----+
| bill | localhost | {"baz": "faz", "comment": "Something about Bill"} |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> ALTER USER 'bill'@'localhost' COMMENT '';
Query OK, 0 rows affected (0.09 sec)

mysql> SELECT * FROM INFORMATION_SCHEMA.USER_ATTRIBUTES
-> WHERE USER='bill' AND HOST='localhost';
+-----+-----+-----+
| USER | HOST | ATTRIBUTE |
+-----+-----+-----+
| bill | localhost | {"baz": "faz", "comment": ""} |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> ALTER USER 'bill'@'localhost' ATTRIBUTE '{"comment": null}';
Query OK, 0 rows affected (0.07 sec)

mysql> SELECT * FROM INFORMATION_SCHEMA.USER_ATTRIBUTES
-> WHERE USER='bill' AND HOST='localhost';
+-----+-----+-----+
| USER | HOST | ATTRIBUTE |
+-----+-----+-----+
| bill | localhost | {"baz": "faz"} |
+-----+-----+-----+
1 row in set (0.00 sec)
```

`auth_option` 値が `user` 値に従うことを許可する `ALTER USER` 構文の場合、`auth_option` は、アカウント認証プラグイン、資格証明 (パスワードなど) またはその両方を指定して、アカウントがどのように認証されるかを示します。各 `auth_option` 値は、直前に指定されたアカウントにのみを適用します。

`user` 仕様に従って、ステートメントに SSL/TLS、リソース制限、パスワード管理およびロックプロパティのオプションを含めることができます。このようなオプションはすべて、ステートメントに対する `global` であり、ステートメントで指定された `all` アカウントに適用されます。

例: アカウントパスワードを変更して期限切れにします。そのため、ユーザーは指定されたパスワードで接続し、次の接続時に新しいパスワードを選択する必要があります:

```
ALTER USER 'jeffrey'@'localhost'  
IDENTIFIED BY 'new_password' PASSWORD EXPIRE;
```

例: `caching_sha2_password` 認証プラグインと指定されたパスワードを使用するようにアカウントを変更します。180 日ごとに新しいパスワードを選択し、ログイン失敗トラッキングを有効にする必要があります。これにより、次の 3 つのパスワードが連続して正しくないと、一時的なアカウントのロックが 2 日間発生します:

```
ALTER USER 'jeffrey'@'localhost'  
IDENTIFIED WITH caching_sha2_password BY 'new_password'  
PASSWORD EXPIRE INTERVAL 180 DAY  
FAILED_LOGIN_ATTEMPTS 3 PASSWORD_LOCK_TIME 2;
```

例: アカウントのロックまたはロック解除:

```
ALTER USER 'jeffrey'@'localhost' ACCOUNT LOCK;  
ALTER USER 'jeffrey'@'localhost' ACCOUNT UNLOCK;
```

例: SSL を使用して接続し、1 時間あたり 20 の接続を確立するためのアカウントが必要です:

```
ALTER USER 'jeffrey'@'localhost'  
REQUIRE SSL WITH MAX_CONNECTIONS_PER_HOUR 20;
```

例: アカウントごとのプロパティおよびグローバルプロパティを指定して、複数のアカウントを変更します:

```
ALTER USER  
'jeffrey'@'localhost'  
IDENTIFIED BY 'jeffrey_new_password',  
'jeanne'@'localhost',  
'josh'@'localhost'  
IDENTIFIED BY 'josh_new_password'  
REPLACE 'josh_current_password'  
RETAIN CURRENT PASSWORD  
REQUIRE SSL WITH MAX_USER_CONNECTIONS 2  
PASSWORD HISTORY 5;
```

`jeffrey` に続く `IDENTIFIED BY` 値は直前のアカウントにのみ適用されるため、パスワードは `jeffrey` の `'jeffrey_new_password'` にのみ変更されます。 `jeanne` の場合、アカウントごとの値はありません (そのため、パスワードは変更されません)。 `josh` の場合、`IDENTIFIED BY` は新しいパスワード (`'josh_new_password'`) を確立し、`ALTER USER` ステートメントを発行するユーザーが現在のパスワード (`'josh_current_password'`) を知っていることを確認するために `REPLACE` が指定され、現在のパスワードもアカウントセカンダリパスワードとして保持されます。(その結果、`josh` はプライマリパスワードまたはセカンダリパスワードのいずれかを使用して接続できます。)

残りのプロパティは、ステートメントで指定されたすべてのアカウントにグローバルに適用されるため、両方のアカウントについて次のようになります:

- SSL を使用するには接続が必要です。
- アカウントは、最大 2 つの同時接続に使用できます。
- パスワードの変更では、最新の 5 つのパスワードを再利用できません。

例: アカウントにプライマリパスワードのみを残して、`josh` のセカンダリパスワードを破棄します:

```
ALTER USER 'josh'@'localhost' DISCARD OLD PASSWORD;
```

特定のタイプのオプションがない場合、アカウントはその点で変更されません。たとえば、ロックオプションを指定しない場合、アカウントのロック状態は変更されません。

ALTER USER 認証オプション

アカウント名の後に、アカウント認証プラグインまたは資格証明 (あるいはその両方) を指定する `auth_option` 認証オプションを続けることができます。また、置換するアカウントの現在のパスワードを指定する `password-verification` 句や、アカウントにセカンダリパスワードがあるかどうかを管理する句も含めることができます。

注記

ランダムパスワード生成、パスワード検証およびセカンダリパスワードの句は、資格証明を MySQL に内部的に格納する認証プラグインを使用するアカウントにのみ適用されます。MySQL の外部にある資格証明システムに対して認証を実行するプラグインを使用するア

ワントの場合、パスワード管理もそのシステムに対して外部で処理する必要があります。内部資格証明記憶域の詳細は、[セクション6.2.15「パスワード管理」](#)を参照してください。

- `auth_plugin` は、認証プラグインに名前を付けます。プラグイン名は、引用符で囲まれた文字列リテラルまたは引用符で囲まれていない名前です。プラグイン名は、`mysql.user` システムテーブルの `plugin` カラムに格納されます。

認証プラグインを指定しない `auth_option` 構文の場合、デフォルトのプラグインは `default_authentication_plugin` システム変数の値で示されます。各プラグインの説明については、[セクション6.4.1「認証プラグイン」](#)を参照してください。

- 内部的に格納される資格証明は、`mysql.user` システムテーブルに格納されます。'`auth_string`'値または `RANDOM PASSWORD` は、アカウント資格証明をクリアテキスト (暗号化されていない) 文字列として指定するか、アカウントに関連付けられた認証プラグインで想定される形式でハッシュします:
 - `BY 'auth_string'` を使用する構文の場合、文字列はクリアテキストであり、ハッシュ化のために認証プラグインに渡されます。プラグインによって返される結果は、`mysql.user` テーブルに格納されます。プラグインは、指定された値を使用できます。この場合、ハッシュは発生しません。
 - `BY RANDOM PASSWORD` を使用する構文の場合、MySQL はランダムパスワードをクリアテキストとして生成し、ハッシュ化のために認証プラグインに渡します。プラグインによって返される結果は、`mysql.user` テーブルに格納されます。プラグインは、指定された値を使用できます。この場合、ハッシュは発生しません。

ランダムに生成されたパスワードは、MySQL 8.0.18 で使用でき、[ランダムパスワード生成](#) で説明されている特性があります。

- `AS 'auth_string'` を使用する構文の場合、文字列はすでに認証プラグインに必要な形式であるとみなされ、`mysql.user` テーブルにそのまま格納されます。プラグインにハッシュ値が必要な場合、その値はプラグインに適した形式ですすでにハッシュされている必要があります。そうでない場合、プラグインはこの値を使用できず、クライアント接続の正しい認証は行われません。

MySQL 8.0.17 の時点では、ハッシュ文字列は文字列リテラルまたは 16 進数値のいずれかになります。後者は、`print_identified_with_as_hex` システム変数が有効になっている場合に、印刷不可能な文字を含むパスワードハッシュに対して `SHOW CREATE USER` によって表示される値のタイプに対応します。

- 認証プラグインが認証文字列のハッシュを実行しない場合、`BY 'auth_string'`句と `AS 'auth_string'`句は同じ効果を持ちます: 認証文字列は、`mysql.user` システムテーブルにそのまま格納されます。
- `REPLACE 'current_auth_string'`句はパスワード検証を実行し、MySQL 8.0.13 の時点で使用できます。指定された場合:
 - `REPLACE` は、置換するアカウントの現在のパスワードをクリアテキスト (暗号化されていない) 文字列として指定します。
 - 変更しようとしているユーザーが実際に現在のパスワードを認識していることを確認するために、現在のパスワードを指定するためにパスワード変更が必要な場合は、句を指定する必要があります。
 - アカウントのパスワードが変更される可能性があるが、現在のパスワードを指定する必要がない場合、句はオプションです。
 - 句が指定されているが、句がオプションであっても現在のパスワードと一致しない場合、ステートメントは失敗します。
 - `REPLACE` は、現在のユーザーのアカウントパスワードを変更する場合にのみ指定できます。

現在のパスワードを指定したパスワード検証の詳細は、[セクション6.2.15「パスワード管理」](#)を参照してください。

- `RETAIN CURRENT PASSWORD` 句および `DISCARD OLD PASSWORD` 句はデュアルパスワード機能を実装し、MySQL 8.0.14 の時点で使用できます。どちらもオプションですが、指定した場合は次の効果があります:
 - `RETAIN CURRENT PASSWORD` は、アカウントの現在のパスワードをセカンダリパスワードとして保持し、既存のセカンダリパスワードを置き換えます。新しいパスワードはプライマリパスワードになりますが、クライアントはアカウントを使用して、プライマリパスワードまたはセカンダリパスワードのいずれかを使用してサー

バーに接続できます。(例外: ALTER USER ステートメントで指定された新しいパスワードが空の場合、RETAIN CURRENT PASSWORD が指定されていても、セカンダリパスワードも空になります。)

- プライマリパスワードが空のアカウントに RETAIN CURRENT PASSWORD を指定すると、ステートメントは失敗します。
- アカウントにセカンダリパスワードがあり、RETAIN CURRENT PASSWORD を指定せずにプライマリパスワードを変更した場合、セカンダリパスワードは変更されません。
- アカウントに割り当てられた認証プラグインを変更すると、セカンダリパスワードは破棄されます。認証プラグインを変更し、RETAIN CURRENT PASSWORD も指定すると、ステートメントは失敗します。
- セカンダリパスワードが存在する場合、DISCARD OLD PASSWORD はセカンダリパスワードを破棄します。アカウントはプライマリパスワードのみを保持し、クライアントはプライマリパスワードのみを使用してサーバーに接続するためにアカウントを使用できます。

デュアルパスワードの使用の詳細は、[セクション6.2.15「パスワード管理」](#)を参照してください。

ALTER USER では、次の auth_option 構文が許可されます:

- IDENTIFIED BY 'auth_string' [REPLACE 'current_auth_string'] [RETAIN CURRENT PASSWORD]

アカウント認証プラグインをデフォルトプラグインに設定し、ハッシュ化のためにクリアテキストの'auth_string'値をプラグインに渡し、その結果を mysql.user システムテーブルのアカウント行に格納します。

このセクションで前述したように、REPLACE 句を指定すると、アカウントの現在のパスワードが指定されます。

RETAIN CURRENT PASSWORD 句を指定すると、このセクションで前述したように、アカウントの現在のパスワードがセカンダリパスワードとして保持されます。

- IDENTIFIED BY RANDOM PASSWORD [REPLACE 'current_auth_string'] [RETAIN CURRENT PASSWORD]

アカウント認証プラグインをデフォルトのプラグインに設定し、ランダムなパスワードを生成して、ハッシュ可能なプラグインにクリアテキストのパスワード値を渡し、その結果を mysql.user システムテーブルのアカウント行に格納します。このステートメントは、ステートメントを実行しているユーザーまたはアプリケーションが使用できるように、クリアテキストのパスワードも結果セットに返します。ランダムに生成されるパスワードの結果セットおよび特性の詳細は、[ランダムパスワード生成](#)を参照してください。

このセクションで前述したように、REPLACE 句を指定すると、アカウントの現在のパスワードが指定されます。

RETAIN CURRENT PASSWORD 句を指定すると、このセクションで前述したように、アカウントの現在のパスワードがセカンダリパスワードとして保持されます。

- IDENTIFIED WITH auth_plugin

アカウント認証プラグインを auth_plugin に設定し、資格証明を空の文字列にクリアし(資格証明は新しい認証プラグインではなく古い認証プラグインに関連付けられます)、その結果を mysql.user システムテーブルのアカウント行に格納します。

また、パスワードは期限切れとマークされます。ユーザーは、次に接続するときに新しいものを選択する必要があります。

- IDENTIFIED WITH auth_plugin BY 'auth_string' [REPLACE 'current_auth_string'] [RETAIN CURRENT PASSWORD]

アカウント認証プラグインを auth_plugin に設定し、ハッシュ化のためにクリアテキストの'auth_string'値をプラグインに渡し、その結果を mysql.user システムテーブルのアカウント行に格納します。

このセクションで前述したように、REPLACE 句を指定すると、アカウントの現在のパスワードが指定されます。

RETAIN CURRENT PASSWORD 句を指定すると、このセクションで前述したように、アカウントの現在のパスワードがセカンダリパスワードとして保持されます。

- IDENTIFIED WITH auth_plugin BY RANDOM PASSWORD [REPLACE 'current_auth_string'] [RETAIN CURRENT PASSWORD]

アカウント認証プラグインを `auth_plugin` に設定し、ランダムパスワードを生成し、ハッシュ化のためにクリアテキストパスワード値をプラグインに渡し、その結果を `mysql.user` システムテーブルのアカウント行に格納します。このステートメントは、ステートメントを実行しているユーザーまたはアプリケーションが使用できるように、クリアテキストのパスワードも結果セットに返します。ランダムに生成されるパスワードの結果セットおよび特性の詳細は、[ランダムパスワード生成](#) を参照してください。

このセクションで前述したように、`REPLACE` 句を指定すると、アカウントの現在のパスワードが指定されます。

`RETAIN CURRENT PASSWORD` 句を指定すると、このセクションで前述したように、アカウントの現在のパスワードがセカンダリパスワードとして保持されます。

- `IDENTIFIED WITH auth_plugin AS 'auth_string'`

アカウント認証プラグインを `auth_plugin` に設定し、`'auth_string'` 値を `mysql.user` アカウント行にそのまま格納します。プラグインにハッシュ文字列が必要な場合、文字列はプラグインに必要な形式ですでにハッシュされているとみなされます。

- `DISCARD OLD PASSWORD`

このセクションで前述したように、アカウントセカンダリパスワードがある場合は破棄します。

例: パスワードをクリアテキストで指定します。デフォルトのプラグインが使用されます:

```
ALTER USER 'jeffrey'@'localhost'  
IDENTIFIED BY 'password';
```

例: 認証プラグインをクリアテキストのパスワード値とともに指定します:

```
ALTER USER 'jeffrey'@'localhost'  
IDENTIFIED WITH mysql_native_password  
BY 'password';
```

例: 前述の例と同様ですが、変更を行ったユーザーがそのパスワードを知っているアカウント要件を満たすために、現在のパスワードをクリアテキスト値として指定します:

```
ALTER USER 'jeffrey'@'localhost'  
IDENTIFIED WITH mysql_native_password  
BY 'password'  
REPLACE 'current_password';
```

`REPLACE` は現在のユーザーパスワードの変更のみを許可されているため、現在のユーザーが `jeffrey` でないかぎり、前述のステートメントは失敗します。

例: 新しいプライマリパスワードを設定し、既存のパスワードをセカンダリパスワードとして保持します:

```
ALTER USER 'jeffrey'@'localhost'  
IDENTIFIED BY 'new_password'  
RETAIN CURRENT PASSWORD;
```

例: セカンダリパスワードを破棄し、アカウントにプライマリパスワードのみを残します:

```
ALTER USER 'jeffrey'@'localhost' DISCARD OLD PASSWORD;
```

例: ハッシュされたパスワード値とともに認証プラグインを指定します:

```
ALTER USER 'jeffrey'@'localhost'  
IDENTIFIED WITH mysql_native_password  
AS '*6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4';
```

パスワードと認証プラグインの設定の詳細は、[セクション6.2.14「アカウントパスワードの割り当て」](#) および [セクション6.2.17「プラグブル認証」](#) を参照してください。

ALTER USER ロールのオプション

`ALTER USER ... DEFAULT ROLE` では、ユーザーがサーバーに接続して認証するとき、またはユーザーがセッション中に `SET ROLE DEFAULT` ステートメントを実行するときにアクティブになるロールを定義します。

`ALTER USER ... DEFAULT ROLE` は、`SET DEFAULT ROLE` の代替構文です (セクション13.7.1.9「`SET DEFAULT ROLE` ステートメント」を参照)。ただし、`ALTER USER` では単一のユーザーに対してのみデフォルトを設定できませんが、`SET DEFAULT ROLE` では複数のユーザーに対してデフォルトを設定できます。一方、`ALTER USER` ステートメントのユーザー名として `CURRENT_USER` を指定できますが、`SET DEFAULT ROLE` のユーザー名は指定できません。

各ユーザーアカウント名には、前述の形式が使用されます。

各ロール名は、セクション6.2.5「`ロール名の指定`」で説明されている形式を使用します。例:

```
ALTER USER 'joe'@'10.0.0.1' DEFAULT ROLE administrator, developer;
```

ロール名のホスト名部分は、省略すると '%' にデフォルト設定されます。

`DEFAULT ROLE` キーワードに続く句では、次の値が許可されます:

- `NONE`: デフォルトを `NONE` (ロールなし) に設定します。
- `ALL`: アカウントに付与されているすべてのロールにデフォルトを設定します。
- `role [, role] ...`: デフォルトを名前付きロールに設定します。このロールは、`ALTER USER ... DEFAULT ROLE` の実行時に存在し、アカウントに付与される必要があります。

ALTER USER SSL/TLS オプション

MySQL では、ユーザー名と資格証明に基づく通常の認証に加えて、X.509 証明書属性をチェックできます。MySQL での SSL/TLS の使用に関する背景情報は、セクション6.3「`暗号化された接続の使用`」を参照してください。

MySQL アカウントの SSL/TLS 関連オプションを指定するには、1 つ以上の `tls_option` 値を指定する `REQUIRE` 句を使用します。

`REQUIRE` オプションの順序は関係ありませんが、オプションを 2 回指定することはできません。 `AND` キーワードは、`REQUIRE` オプション間のオプションです。

`ALTER USER` では、次の `tls_option` 値が許可されます:

- `NONE`

ステートメントで指定されたすべてのアカウントに SSL または X.509 要件がないことを示します。ユーザー名とパスワードが有効であれば、暗号化されていない接続が許可されます。クライアントに適切な証明書および鍵ファイルがある場合は、クライアントオプションで暗号化された接続を使用できます。

```
ALTER USER 'jeffrey'@'localhost' REQUIRE NONE;
```

クライアントは、デフォルトでセキュアな接続を確立しようとします。 `REQUIRE NONE` を持つクライアントでは、セキュアな接続を確立できない場合、接続試行は暗号化されていない接続にフォールバックされます。暗号化された接続を要求するには、クライアントは `--ssl-mode=REQUIRED` オプションのみを指定する必要があります。セキュアな接続を確立できない場合、接続の試行は失敗します。

- `SSL`

ステートメントで指定されたすべてのアカウントに対して暗号化された接続のみを許可するようにサーバーに指示します。

```
ALTER USER 'jeffrey'@'localhost' REQUIRE SSL;
```

クライアントは、デフォルトでセキュアな接続を確立しようとします。 `REQUIRE SSL` を持つアカウントでは、セキュアな接続を確立できない場合、接続の試行は失敗します。

- `X509`

ステートメントで指定されたすべてのアカウントについて、クライアントは有効な証明書を提示する必要がありますが、正確な証明書、発行者およびサブジェクトは関係ありません。唯一の要件は、いずれかの CA 証明書でその

署名を検証できるべきであるということです。X.509 証明書の使用は常に暗号化を意味するため、この場合は **SSL** オプションは必要ありません。

```
ALTER USER 'jeffrey'@'localhost' REQUIRE X509;
```

REQUIRE X509 のアカウントの場合、クライアントは接続する **--ssl-key** および **--ssl-cert** オプションを指定する必要があります。(サーバーによって提供される公開証明書を検証できるように、**--ssl-ca** も指定することをお勧めしますが、必須ではありません。)これらの **REQUIRE** オプションは **X509** の要件を意味するため、これは **ISSUER** および **SUBJECT** にも当てはまります。

- **ISSUER 'issuer'**

ステートメントで指定されたすべてのアカウントについて、CA **'issuer'** によって発行された有効な X.509 証明書をクライアントが提示する必要があります。クライアントが有効だが発行者が異なる証明書を提示した場合、サーバーは接続を拒否します。X.509 証明書の使用は常に暗号化を意味するため、この場合は **SSL** オプションは必要ありません。

```
ALTER USER 'jeffrey'@'localhost'  
REQUIRE ISSUER '/C=SE/ST=Stockholm/L=Stockholm/  
O=MySQL/CN=CA/emailAddress=ca@example.com';
```

ISSUER には **X509** の要件があるため、クライアントは接続するために **--ssl-key** および **--ssl-cert** オプションを指定する必要があります。(サーバーによって提供される公開証明書を検証できるように、**--ssl-ca** も指定することをお勧めしますが、必須ではありません。)

- **SUBJECT 'subject'**

ステートメントで指定されたすべてのアカウントについて、クライアントがサブジェクト **subject** を含む有効な X.509 証明書を提示する必要があります。クライアントが有効だがサブジェクトが異なる証明書を提示した場合、サーバーは接続を拒否します。X.509 証明書の使用は常に暗号化を意味するため、この場合は **SSL** オプションは必要ありません。

```
ALTER USER 'jeffrey'@'localhost'  
REQUIRE SUBJECT '/C=SE/ST=Stockholm/L=Stockholm/  
O=MySQL demo client certificate/  
CN=client/emailAddress=client@example.com';
```

MySQL では、**'subject'** 値と証明書の値との単純な文字列比較が行われるため、大文字と小文字およびコンポーネントの順序は、証明書に存在するものとまったく同じにする必要があります。

SUBJECT には **X509** の要件があるため、クライアントは接続するために **--ssl-key** および **--ssl-cert** オプションを指定する必要があります。(サーバーによって提供される公開証明書を検証できるように、**--ssl-ca** も指定することをお勧めしますが、必須ではありません。)

- **CIPHER 'cipher'**

ステートメントで指定されたすべてのアカウントには、接続を暗号化するための特定の暗号メソッドが必要です。このオプションは、十分な強度の暗号およびキー長が使用されるようにするために必要です。短い暗号化キーを使用する古いアルゴリズムを使用すると、暗号化が弱くなる可能性があります。

```
ALTER USER 'jeffrey'@'localhost'  
REQUIRE CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

SUBJECT、**ISSUER** および **CIPHER** オプションは、**REQUIRE** 句で組み合わせることができます:

```
ALTER USER 'jeffrey'@'localhost'  
REQUIRE SUBJECT '/C=SE/ST=Stockholm/L=Stockholm/  
O=MySQL demo client certificate/  
CN=client/emailAddress=client@example.com'  
AND ISSUER '/C=SE/ST=Stockholm/L=Stockholm/  
O=MySQL/CN=CA/emailAddress=ca@example.com'  
AND CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

ALTER USER リソース制限オプション

[セクション6.2.20「アカウントリソース制限の設定」](#) で説明されているように、アカウントによるサーバーリソースの使用に制限を設定できます。そのためには、1 つ以上の **resource_option** 値を指定する **WITH** 句を使用します。

WITH オプションの順序は重要ではありませんが、特定のリソース制限が複数回指定された場合は、最後のインスタンスが優先されます。

ALTER USER では、次の `resource_option` 値が許可されます:

- `MAX_QUERIES_PER_HOUR count`, `MAX_UPDATES_PER_HOUR count`, `MAX_CONNECTIONS_PER_HOUR count`

ステートメントで指定されたすべてのアカウントについて、これらのオプションは、特定の 1 時間の間に各アカウントに許可されるクエリー、更新、およびサーバーへの接続の数を制限します。count が 0 (デフォルト) である場合、これは、このアカウントに対する制限が存在しないことを示します。

- `MAX_USER_CONNECTIONS count`

ステートメントで指定されたすべてのアカウントについて、各アカウントによるサーバーへの同時接続の最大数を制限します。0 以外の count は、このアカウントに対する制限を明示的に指定します。count が 0 (デフォルト) である場合、サーバーは、`max_user_connections` システム変数のグローバル値からこのアカウントの同時接続の数を決定します。max_user_connections もゼロである場合は、アカウントに制限がありません。

例:

```
ALTER USER 'jeffrey'@'localhost'  
WITH MAX_QUERIES_PER_HOUR 500 MAX_UPDATES_PER_HOUR 100;
```

ALTER USER のパスワード管理オプション

ALTER USER では、パスワード管理用にいくつかの `password_option` 値がサポートされています:

- パスワードの有効期限オプション: アカウントパスワードを手動で期限切れにし、そのパスワード有効期限ポリシーを設定できます。ポリシーオプションによってパスワードが期限切れになることはありません。代わりに、最新のアカウントパスワード変更の日時から評価されるパスワード有効期限に基づいて、サーバーがアカウントに自動期限切れを適用する方法を決定します。
- パスワード再利用オプション: パスワードの再利用は、パスワード変更の数、経過時間、またはその両方に基づいて制限できます。
- パスワード検証必須オプション: 変更しようとしているユーザーが実際に現在のパスワードを認識していることを確認するために、アカウントパスワードの変更を試行する際に現在のパスワードを指定する必要があるかどうかを指定できます。
- 不正解 - パスワード失敗 - ログイントラッキングオプション: サーバーが失敗したログイン試行を追跡し、連続して正しくないパスワードが多すぎるアカウントを一時的にロックするようにすることができます。必要な失敗数とロック時間は構成可能です。

このセクションでは、パスワード管理オプションの構文について説明します。パスワード管理のポリシーの確立の詳細は、[セクション6.2.15「パスワード管理」](#)を参照してください。

特定のタイプの複数のパスワード管理オプションが指定されている場合は、最後のオプションが優先されます。たとえば、`PASSWORD EXPIRE DEFAULT PASSWORD EXPIRE NEVER` は `PASSWORD EXPIRE NEVER` と同じです。

注記

失敗したログイン追跡に関連するオプションを除き、パスワード管理オプションは、資格証明を MySQL に内部的に格納する認証プラグインを使用するアカウントにのみ適用されます。MySQL の外部にある資格証明システムに対して認証を実行するプラグインを使用するアカウントの場合、パスワード管理もそのシステムに対して外部で処理する必要があります。内部資格証明記憶域の詳細は、[セクション6.2.15「パスワード管理」](#)を参照してください。

アカウントパスワードが手動で期限切れになった場合、または自動期限切れポリシーに従ってパスワードの有効期間が許可された存続期間を超えたとみなされた場合、クライアントには期限切れのパスワードがあります。この場合、サーバーはクライアントを切断するか、クライアントに許可されている操作を制限します ([セクション6.2.16「期限切れパスワードのサーバー処理」](#)を参照)。制限付きクライアントによって実行される操作は、ユーザーが新しいアカウントパスワードを確立するまでエラーになります。

注記

期限切れのパスワードは、現在の値に設定することで「reset」で使用できますが、適切なポリシーとして、別のパスワードを選択することをお勧めします。DBAは、適切なパスワード再利用ポリシーを確立することで、非キューを強制できます。[パスワード再利用ポリシー](#)を参照してください。

`ALTER USER` では、パスワードの有効期限を制御するために次の `password_option` 値が許可されます:

- **PASSWORD EXPIRE**

ステートメントで指定されたすべてのアカウントのパスワードをすぐに期限切れとしてマークします。

```
ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE;
```

- **PASSWORD EXPIRE DEFAULT**

`default_password_lifetime` システム変数で指定されたグローバル有効期限ポリシーが適用されるように、ステートメントで指定されたすべてのアカウントを設定します。

```
ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE DEFAULT;
```

- **PASSWORD EXPIRE NEVER**

この有効期限オプションは、ステートメントで指定されたすべてのアカウントのグローバルポリシーをオーバーライドします。それぞれについて、パスワードの有効期限が切れないようにパスワードの有効期限を無効にします。

```
ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE NEVER;
```

- **PASSWORD EXPIRE INTERVAL N DAY**

この有効期限オプションは、ステートメントで指定されたすべてのアカウントのグローバルポリシーをオーバーライドします。それぞれについて、パスワードの存続期間が `N` 日に設定されます。次のステートメントでは、180日ごとにパスワードを変更する必要があります:

```
ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE INTERVAL 180 DAY;
```

`ALTER USER` では、必要な最小パスワード変更数に基づいて以前のパスワードの再利用を制御するために、次の `password_option` 値が許可されています:

- **PASSWORD HISTORY DEFAULT**

`password_history` システム変数で指定された変更数の前にパスワードの再利用を禁止するために、パスワード履歴の長さに関するグローバルポリシーが適用されるように、ステートメントで指定されたすべてのアカウントを設定します。

```
ALTER USER 'jeffrey'@'localhost' PASSWORD HISTORY DEFAULT;
```

- **PASSWORD HISTORY N**

この履歴長オプションは、ステートメントで指定されたすべてのアカウントのグローバルポリシーをオーバーライドします。それぞれについて、最近選択した `N` パスワードの再利用を禁止するために、パスワード履歴の長さを `N` パスワードに設定します。次のステートメントは、以前の6つのパスワードの再利用を禁止します:

```
ALTER USER 'jeffrey'@'localhost' PASSWORD HISTORY 6;
```

`ALTER USER` では、経過時間に基づいて以前のパスワードの再利用を制御するために、次の `password_option` 値が許可されます:

- **PASSWORD REUSE INTERVAL DEFAULT**

経過時間に関するグローバルポリシーが適用され、`password_reuse_interval` システム変数で指定された日数よりも新しいパスワードの再利用が禁止されるように、アカウントで指定されたすべてのステートメントを設定します。

```
ALTER USER 'jeffrey'@'localhost' PASSWORD REUSE INTERVAL DEFAULT;
```

- **PASSWORD REUSE INTERVAL N DAY**

この time-elapsed オプションは、ステートメントで指定されたすべてのアカウントのグローバルポリシーをオーバーライドします。それぞれについて、パスワードの再利用間隔を **N** 日に設定して、その日数より新しいパスワードの再利用を禁止します。次のステートメントは、360 日間のパスワードの再利用を禁止します:

```
ALTER USER 'jeffrey'@'localhost' PASSWORD REUSE INTERVAL 360 DAY;
```

ALTER USER では、アカウントパスワードの変更の試行で現在のパスワードを指定する必要があるかどうかを制御するために、変更しようとしているユーザーが実際に現在のパスワードを知っていることを確認するために、次の **password_option** 値を許可しています:

- **PASSWORD REQUIRE CURRENT**

この検証オプションは、ステートメントで指定されたすべてのアカウントのグローバルポリシーをオーバーライドします。それぞれについて、パスワードの変更で現在のパスワードを指定する必要があります。

```
ALTER USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT;
```

- **PASSWORD REQUIRE CURRENT OPTIONAL**

この検証オプションは、ステートメントで指定されたすべてのアカウントのグローバルポリシーをオーバーライドします。それぞれについて、パスワードを変更して現在のパスワードを指定する必要はありません。(現在のパスワードを指定する必要はありますが、指定する必要はありません。)

```
ALTER USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT OPTIONAL;
```

- **PASSWORD REQUIRE CURRENT DEFAULT**

password_require_current システム変数で指定されたパスワード検証に関するグローバルポリシーが適用されるように、アカウントで指定されたすべてのステートメントを設定します。

```
ALTER USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT DEFAULT;
```

MySQL 8.0.19 の時点で、**ALTER USER** は、失敗したログイン追跡を制御するために次の **password_option** 値を許可します:

- **FAILED_LOGIN_ATTEMPTS N**

不正なパスワードを指定するアカウントログイン試行を追跡するかどうか。 **N** は 0 から 32767 の数値である必要があります。値 0 を指定すると、失敗したログインロックが無効になります。0 より大きい値は、パスワードが何回連続して失敗したために一時アカウントがロックされるかを示します (**PASSWORD_LOCK_TIME** もゼロ以外の場合)。

- **PASSWORD_LOCK_TIME {N | UNBOUNDED}**

連続して何度もログインを試行した後にアカウントをロックする期間。パスワードが正しくありません。 **N** は、0 から 32767 の数値、または **UNBOUNDED** である必要があります。値 0 を指定すると、一時アカウントロックが無効になります。0 より大きい値は、アカウントをロックする期間を日数で示します。値が **UNBOUNDED** の場合、アカウントのロック期間は無制限になります。ロックされると、アカウントはロック解除されるまでロック状態のままになります。ロック解除が発生する条件の詳細は、[失敗したログインロックと一時アカウントロック](#) を参照してください。

ログイン失敗トラッキングと一時ロックを実行するには、アカウントの **FAILED_LOGIN_ATTEMPTS** オプションと **PASSWORD_LOCK_TIME** オプションの両方をゼロ以外にする必要があります。次のステートメントは、パスワードが 4 回連続して失敗した後も 2 日間ロックされたままになるようにアカウントを変更します:

```
ALTER USER 'jeffrey'@'localhost'  
  FAILED_LOGIN_ATTEMPTS 4 PASSWORD_LOCK_TIME 2;
```

ALTER USER アカウントロックオプション

MySQL では、アカウントのロック状態を指定する **ACCOUNT LOCK** および **ACCOUNT UNLOCK** オプションを使用したアカウントのロックおよびロック解除がサポートされています。詳細は、[セクション 6.2.19 「アカウントロック」](#) を参照してください。

複数の account-locking オプションが指定されている場合は、最後のオプションが優先されます。

MySQL 8.0.19 の時点では、ログインの失敗回数が多すぎるために一時的にロックされているステートメントで指定されたアカウントは、`ALTER USER ... UNLOCK` によってロック解除されます。 [セクション6.2.15「パスワード管理」](#) を参照してください。

ALTER USER バイナリロギング

`ALTER USER` は、成功した場合はバイナリログに書き込まれますが、失敗した場合は書き込まれません。その場合、ロールバックが発生し、変更は行われません。バイナリログに書き込まれるステートメントには、指定されたすべてのユーザーが含まれます。 `IF EXISTS` 句が指定されている場合、これには存在せず、変更されなかったユーザーも含まれます。

元のステートメントがユーザーの資格を変更した場合、バイナリログに書き込まれるステートメントは、そのユーザーに適用可能な認証プラグインを次のように指定します：

- 元のステートメントで指定されたプラグイン (指定されている場合)。
- それ以外の場合は、ユーザーアカウントに関連付けられたプラグイン (ユーザーが存在する場合)、またはデフォルトの認証プラグイン (ユーザーが存在しない場合)。(バイナリログに書き込まれたステートメントがユーザーの特定の認証プラグインを指定する必要がある場合は、それを元のステートメントに含めます。)

サーバーは、バイナリログに書き込まれたステートメント内の任意のユーザーのデフォルトの認証プラグインを追加すると、それらのユーザーを指定する警告をエラーログに書き込みます。

元のステートメントで `FAILED_LOGIN_ATTEMPTS` または `PASSWORD_LOCK_TIME` オプションが指定されている場合、バイナリログに書き込まれるステートメントにはそのオプションが含まれます。

13.7.1.2 CREATE ROLE ステートメント

```
CREATE ROLE [IF NOT EXISTS] role [, role ] ...
```

`CREATE ROLE` では、権限の名前付きコレクションである 1 つ以上のロールが作成されます。このステートメントを使用するには、グローバル `CREATE ROLE` または `CREATE USER` 権限が必要です。 `read_only` システム変数が有効になっている場合、`CREATE ROLE` にはさらに `CONNECTION_ADMIN` 権限 (または非推奨の `SUPER` 権限) が必要です。

作成されたロールはロックされ、パスワードがなく、デフォルトの認証プラグインが割り当てられます。(これらのロール属性は、後で `ALTER USER` ステートメントを使用して、グローバル `CREATE USER` 権限を持つユーザーが変更できます。)

`CREATE ROLE` は、すべての名前付きロールに対して成功するか、ロールバックされ、エラーが発生しても効果はありません。デフォルトでは、すでに存在するロールを作成しようとするとエラーが発生します。 `IF NOT EXISTS` 句が指定されている場合、ステートメントは、エラーではなく、すでに存在する名前付きロールごとに警告を生成します。

ステートメントが成功した場合はバイナリログに書き込まれますが、失敗した場合は書き込まれず、ロールバックが発生して変更は行われません。バイナリログに書き込まれるステートメントには、すべての名前付き役割が含まれます。 `IF NOT EXISTS` 句が指定されている場合は、すでに存在していて作成されていないロールも含まれます。

各ロール名は、 [セクション6.2.5「ロール名の指定」](#) で説明されている形式を使用します。例：

```
CREATE ROLE 'administrator', 'developer';
CREATE ROLE 'webapp'@'localhost';
```

ロール名のホスト名部分は、省略すると `'%'` にデフォルト設定されます。

ロールの使用例は、 [セクション6.2.10「ロールの使用」](#) を参照してください。

13.7.1.3 CREATE USER ステートメント

```
CREATE USER [IF NOT EXISTS]
user [auth_option] [, user [auth_option]] ...
DEFAULT ROLE role [, role ] ...
[REQUIRE {NONE | tls_option [[AND] tls_option] ...}]
[WITH resource_option [resource_option] ...]
[password_option | lock_option] ...
[COMMENT 'comment_string' | ATTRIBUTE 'json_object']
```

```
user:
  (see セクション6.2.4「アカウント名の指定」)

auth_option: {
  IDENTIFIED BY 'auth_string'
| IDENTIFIED BY RANDOM PASSWORD
| IDENTIFIED WITH auth_plugin
| IDENTIFIED WITH auth_plugin BY 'auth_string'
| IDENTIFIED WITH auth_plugin BY RANDOM PASSWORD
| IDENTIFIED WITH auth_plugin AS 'auth_string'
}

tls_option: {
  SSL
| X509
| CIPHER 'cipher'
| ISSUER 'issuer'
| SUBJECT 'subject'
}

resource_option: {
  MAX_QUERIES_PER_HOUR count
| MAX_UPDATES_PER_HOUR count
| MAX_CONNECTIONS_PER_HOUR count
| MAX_USER_CONNECTIONS count
}

password_option: {
  PASSWORD EXPIRE [DEFAULT | NEVER | INTERVAL N DAY]
| PASSWORD HISTORY {DEFAULT | N}
| PASSWORD REUSE INTERVAL {DEFAULT | N DAY}
| PASSWORD REQUIRE CURRENT [DEFAULT | OPTIONAL]
| FAILED_LOGIN_ATTEMPTS N
| PASSWORD_LOCK_TIME {N | UNBOUNDED}
}

lock_option: {
  ACCOUNT LOCK
| ACCOUNT UNLOCK
}
```

CREATE USER ステートメントは、新しい MySQL アカウントを作成します。これにより、認証、ロール、SSL/TLS、リソース制限およびパスワード管理プロパティを新しいアカウントに対して確立できます。また、アカウントを最初にロックするかロック解除するかも制御します。

CREATE USER を使用するには、`mysql` システムスキーマに対するグローバル **CREATE USER** 権限または **INSERT** 権限が必要です。 `read_only` システム変数が有効になっている場合、**CREATE USER** にはさらに **CONNECTION_ADMIN** 権限 (または非推奨の **SUPER** 権限) が必要です。

MySQL 8.0.22 では、作成するアカウントの名前がストアオブジェクトの **DEFINER** 属性として指定されている場合、**CREATE USER** はエラーで失敗します。(つまり、アカウントを作成すると、アカウントが現在孤立しているストアオブジェクトを採用する場合、ステートメントは失敗します。) 操作を実行するには、**SET_USER_ID** 権限が必要です。この場合、ステートメントはエラーで失敗するのではなく、警告付きで成功します。**SET_USER_ID** がない場合、ユーザー作成操作を実行するには、孤立したオブジェクトを削除し、アカウントを作成してその権限を付与してから、削除したオブジェクトを再作成します。特定のアカウントを **DEFINER** 属性として指定するオブジェクトの識別方法などの追加情報は、[孤立したストアオブジェクト](#) を参照してください。

CREATE USER は、指定されたすべてのユーザーに対して成功するか、ロールバックされ、エラーが発生しても効果はありません。デフォルトでは、すでに存在するユーザーを作成しようとするエラーが発生します。**IF NOT EXISTS** 句が指定されている場合、ステートメントは、エラーではなく、すでに存在する名前付きユーザーごとに警告を生成します。

重要

状況によっては、**CREATE USER** がサーバーログ、またはクライアント側にある `~/mysql_history` などの履歴ファイル内に記録されることがあります。つまり、平文のパスワードが、その情報に対する読み取りアクセス権を持つ任意のユーザーによって読み取られる可能性があります。これがサーバーログで発生する条件およびこれを制御する方法については、[セクション6.1.2.3「パスワードおよびロギング」](#) を参照してください。クライアン

ト側のログインに関する同様の情報については、[セクション4.5.1.3「mysql クライアントログイン」](#)を参照してください。

次のトピックで説明するように、`CREATE USER` ステートメントにはいくつかの側面があります:

- [CREATE USER の概要](#)
- [CREATE USER 認証オプション](#)
- [CREATE USER ロールのオプション](#)
- [CREATE USER SSL/TLS オプション](#)
- [CREATE USER リソース制限オプション](#)
- [CREATE USER のパスワード管理オプション](#)
- [CREATE USER アカウントロックオプション](#)
- [CREATE USER バイナリログイン](#)

CREATE USER の概要

`CREATE USER` では、アカウントごとに `mysql.user` システムテーブルに新しい行が作成されます。アカウント行には、ステートメントで指定されたプロパティが反映されます。未指定のプロパティはデフォルト値に設定されます:

- 認証: `default_authentication_plugin` システム変数で定義された認証プラグインと空の資格証明
- デフォルトロール: `NONE`
- SSL/TLS: `NONE`
- リソース制限: 無制限
- パスワード管理: `PASSWORD EXPIRE DEFAULT PASSWORD HISTORY DEFAULT PASSWORD REUSE INTERVAL DEFAULT PASSWORD REQUIRE CURRENT DEFAULT`。ログイン失敗トラッキングおよび一時アカウントロックは無効です
- アカウントのロック: `ACCOUNT UNLOCK`

最初に作成されたアカウントには権限がなく、`NONE` のデフォルトロールがあります。権限またはロールを割り当てるには、`GRANT` ステートメントを使用します。

各アカウント名には、[セクション6.2.4「アカウント名の指定」](#)で説明されている形式が使用されます。例:

```
CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'password';
```

アカウント名のホスト名部分は、省略すると '%' にデフォルト設定されます。

アカウントを指定する各 `user` 値の後に、アカウントの認証方法を示すオプションの `auth_option` 値を続けることができます。これらの値を使用すると、アカウント認証プラグインおよび資格証明 (パスワードなど) を指定できます。各 `auth_option` 値は、直前に指定されたアカウントにのみを適用します。

`user` 仕様に従って、ステートメントに SSL/TLS、リソース制限、パスワード管理およびロックプロパティのオプションを含めることができます。このようなオプションはすべて、ステートメントに対する `global` であり、ステートメントで指定された all アカウントに適用されます。

例: デフォルトの認証プラグインと指定されたパスワードを使用するアカウントを作成します。ユーザーがサーバーへの最初の接続時に新しいパスワードを選択する必要があるように、パスワードを期限切れとしてマークします:

```
CREATE USER 'jeffrey'@'localhost'  
IDENTIFIED BY 'new_password' PASSWORD EXPIRE;
```

例: `caching_sha2_password` 認証プラグインと指定されたパスワードを使用するアカウントを作成します。180 日ごとに新しいパスワードを選択し、ログイン失敗トラッキングを有効にする必要があります。これにより、次の 3 つのパスワードが連続して正しくないと、一時的なアカウントのロックが 2 日間発生します:

```
CREATE USER 'jeffrey'@'localhost'
```

```
IDENTIFIED WITH caching_sha2_password BY 'new_password'  
PASSWORD EXPIRE INTERVAL 180 DAY  
FAILED_LOGIN_ATTEMPTS 3 PASSWORD_LOCK_TIME 2;
```

例: アカウントごとのプロパティとグローバルプロパティを指定して、複数のアカウントを作成します:

```
CREATE USER  
'jeffrey'@'localhost' IDENTIFIED WITH mysql_native_password  
BY 'new_password1',  
'jeanne'@'localhost' IDENTIFIED WITH caching_sha2_password  
BY 'new_password2'  
REQUIRE X509 WITH MAX_QUERIES_PER_HOUR 60  
PASSWORD HISTORY 5  
ACCOUNT LOCK;
```

各 `auth_option` 値 (この場合は `IDENTIFIED WITH ... BY`) は、直前に指定されたアカウントにのみ適用されるため、各アカウントはすぐ後の認証プラグインおよびパスワードを使用します。

残りのプロパティは、ステートメントで指定されたすべてのアカウントにグローバルに適用されるため、両方のアカウントについて次のようになります:

- 有効な X.509 証明書を使用して接続する必要があります。
- 1 時間あたり最大 60 個のクエリーが許可されます。
- パスワードの変更では、最新の 5 つのパスワードを再利用できません。
- アカウントは最初にロックされるため、実質的にはプレースホルダであり、管理者がロックを解除するまで使用できません。

MySQL 8.0.21 以降では、ここで説明するように、オプションでユーザーコメントまたはユーザー属性を持つユーザーを作成できます:

- ユーザーコメント

ユーザーコメントを設定するには、`CREATE USER` ステートメントに `COMMENT 'user_comment'` を追加します。ここで、`user_comment` はユーザーコメントのテキストです。

例 (他のオプションは省略):

```
CREATE USER 'jon'@'localhost' COMMENT 'Some information about Jon';
```

- ユーザー属性

ユーザー属性は、1 つ以上のキーと値のペアで構成される JSON オブジェクトで、`ATTRIBUTE 'json_object'` を `CREATE USER` の一部として含むことによって設定されます。`json_object` は有効な JSON オブジェクトである必要があります。

例 (他のオプションは省略):

```
CREATE USER 'jim'@'localhost'  
ATTRIBUTE '{"fname": "James", "lname": "Scott", "phone": "123-456-7890"}';
```

ユーザーコメントとユーザー属性は、`INFORMATION_SCHEMA.USER_ATTRIBUTES` テーブルの `ATTRIBUTE` カラムにまとめて格納されます。このクエリーでは、ユーザー `jin@localhost` を作成するために示したステートメントによって挿入された次のテーブルの行が表示されます:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.USER_ATTRIBUTES  
-> WHERE USER = 'jim' AND HOST = 'localhost'\G  
***** 1. row *****  
USER: jim  
HOST: localhost  
ATTRIBUTE: {"fname": "James", "lname": "Scott", "phone": "123-456-7890"}  
1 row in set (0.00 sec)
```

実際の `COMMENT` オプションは、キーとして `comment` のみを持ち、その値がオプションに指定された引数であるユーザー属性を設定するためのショートカットを提供します。これを確認するには、`CREATE USER 'jon'@'localhost' COMMENT 'Some information about Jon'` ステートメントを実行し、`USER_ATTRIBUTES` テーブルに挿入される行を確認します:

```
mysql> CREATE USER 'jon'@'localhost' COMMENT 'Some information about Jon';
Query OK, 0 rows affected (0.06 sec)
```

```
mysql> SELECT * FROM INFORMATION_SCHEMA.USER_ATTRIBUTES
-> WHERE USER = 'jon' AND HOST = 'localhost';
+-----+-----+-----+
| USER | HOST | ATTRIBUTE |
+-----+-----+-----+
| jon | localhost | {"comment": "Some information about Jon"} |
+-----+-----+-----+
1 row in set (0.00 sec)
```

`COMMENT` と `ATTRIBUTE` を同じ `CREATE USER` ステートメントで一緒に使用することはできません。使用しようとすると、構文エラーが発生します。ユーザー属性の設定と同時にユーザーコメントを設定するには、次のように `ATTRIBUTE` を使用して、その引数に `comment` キーを持つ値を含めます:

```
mysql> CREATE USER 'bill'@'localhost'
-> ATTRIBUTE '{"fname":"William", "lname":"Schmidt",
-> "comment":"Website developer"}';
Query OK, 0 rows affected (0.16 sec)
```

`ATTRIBUTE` 行のコンテンツは JSON オブジェクトであるため、次に示すように、適切な MySQL JSON 関数または演算子を使用して操作できます:

```
mysql> SELECT
-> USER AS User,
-> HOST AS Host,
-> CONCAT(ATTRIBUTE->"$.fname", " ", ATTRIBUTE->"$.lname") AS 'Full Name',
-> ATTRIBUTE->"$.comment" AS Comment
-> FROM INFORMATION_SCHEMA.USER_ATTRIBUTES
-> WHERE USER='bill' AND HOST='localhost';
+-----+-----+-----+-----+
| User | Host | Full Name | Comment |
+-----+-----+-----+-----+
| bill | localhost | William Schmidt | Website developer |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

既存のユーザーのユーザーコメントまたはユーザー属性を設定または変更するには、`ALTER USER` ステートメントで `COMMENT` または `ATTRIBUTE` オプションを使用できます。

ユーザーコメントとユーザー属性は単一の JSON カラムに内部的にまとめて格納されるため、これにより、最大結合サイズの上限が設定されます。詳細は、[JSON 記憶域の要件](#) を参照してください。

詳細および例は、情報スキーマ `USER_ATTRIBUTES` テーブルの説明も参照してください。

CREATE USER 認証オプション

アカウント名の後に、アカウント認証プラグインまたは資格証明 (あるいはその両方) を指定する `auth_option` 認証オプションを続けることができます。

注記

ランダムパスワード生成の句は、資格証明を MySQL に内部的に格納する認証プラグインを使用するアカウントにのみ適用されます。MySQL の外部にある資格証明システムに対して認証を実行するプラグインを使用するアカウントの場合、パスワード管理もそのシステムに対して外部で処理する必要があります。内部資格証明記憶域の詳細は、[セクション 6.2.15 「パスワード管理」](#) を参照してください。

- `auth_plugin` は、認証プラグインに名前を付けます。プラグイン名は、引用符で囲まれた文字列リテラルまたは引用符で囲まれていない名前です。プラグイン名は、`mysql.user` システムテーブルの `plugin` カラムに格納されます。

認証プラグインを指定しない `auth_option` 構文の場合、デフォルトのプラグインは `default_authentication_plugin` システム変数の値で示されます。各プラグインの説明については、[セクション 6.4.1 「認証プラグイン」](#) を参照してください。

- 内部的に格納される資格証明は、`mysql.user` システムテーブルに格納されます。'auth_string'値または `RANDOM PASSWORD` は、アカウント資格証明をクリアテキスト (暗号化されていない) 文字列として指定するか、アカウントに関連付けられた認証プラグインで想定される形式でハッシュします:

- **BY 'auth_string'**を使用する構文の場合、文字列はクリアテキストであり、ハッシュ化のために認証プラグインに渡されます。プラグインによって返される結果は、`mysql.user` テーブルに格納されます。プラグインは、指定された値を使用できます。この場合、ハッシュは発生しません。
- **BY RANDOM PASSWORD** を使用する構文の場合、MySQL はランダムパスワードをクリアテキストとして生成し、ハッシュ化のために認証プラグインに渡します。プラグインによって返される結果は、`mysql.user` テーブルに格納されます。プラグインは、指定された値を使用できます。この場合、ハッシュは発生しません。

ランダムに生成されたパスワードは、MySQL 8.0.18 で使用でき、[ランダムパスワード生成](#) で説明されている特性があります。

- **AS 'auth_string'**を使用する構文の場合、文字列はすでに認証プラグインに必要な形式であるとみなされ、`mysql.user` テーブルにそのまま格納されます。プラグインにハッシュ値が必要な場合、その値はプラグインに適した形式ですすでにハッシュされている必要があります。そうでない場合、プラグインはこの値を使用できず、クライアント接続の正しい認証は行われません。

MySQL 8.0.17 の時点では、ハッシュ文字列は文字列リテラルまたは 16 進数値のいずれかになります。後者は、`print_identified_with_as_hex` システム変数が有効になっている場合に、印刷不可能な文字を含むパスワードハッシュに対して `SHOW CREATE USER` によって表示される値のタイプに対応します。

- 認証プラグインが認証文字列のハッシュを実行しない場合、**BY 'auth_string'**句と **AS 'auth_string'**句は同じ効果を持ちます: 認証文字列は、`mysql.user` システムテーブルにそのまま格納されます。

`CREATE USER` では、次の `auth_option` 構文が許可されます:

- **IDENTIFIED BY 'auth_string'**

アカウント認証プラグインをデフォルトプラグインに設定し、ハッシュ化のためにクリアテキストの `'auth_string'` 値をプラグインに渡し、その結果を `mysql.user` システムテーブルのアカウント行に格納します。

- **IDENTIFIED BY RANDOM PASSWORD**

アカウント認証プラグインをデフォルトのプラグインに設定し、ランダムなパスワードを生成して、ハッシュ可能なプラグインにクリアテキストのパスワード値を渡し、その結果を `mysql.user` システムテーブルのアカウント行に格納します。このステートメントは、ステートメントを実行しているユーザーまたはアプリケーションが使用できるように、クリアテキストのパスワードも結果セットに返します。ランダムに生成されるパスワードの結果セットおよび特性の詳細は、[ランダムパスワード生成](#) を参照してください。

- **IDENTIFIED WITH auth_plugin**

アカウント認証プラグインを `auth_plugin` に設定し、資格証明を空の文字列にクリアして、その結果を `mysql.user` システムテーブルのアカウント行に格納します。

- **IDENTIFIED WITH auth_plugin BY 'auth_string'**

アカウント認証プラグインを `auth_plugin` に設定し、ハッシュ化のためにクリアテキストの `'auth_string'` 値をプラグインに渡し、その結果を `mysql.user` システムテーブルのアカウント行に格納します。

- **IDENTIFIED WITH auth_plugin BY RANDOM PASSWORD**

アカウント認証プラグインを `auth_plugin` に設定し、ランダムパスワードを生成し、ハッシュ化のためにクリアテキストパスワード値をプラグインに渡し、その結果を `mysql.user` システムテーブルのアカウント行に格納します。このステートメントは、ステートメントを実行しているユーザーまたはアプリケーションが使用できるように、クリアテキストのパスワードも結果セットに返します。ランダムに生成されるパスワードの結果セットおよび特性の詳細は、[ランダムパスワード生成](#) を参照してください。

- **IDENTIFIED WITH auth_plugin AS 'auth_string'**

アカウント認証プラグインを `auth_plugin` に設定し、`'auth_string'` 値を `mysql.user` アカウント行にそのまま格納します。プラグインにハッシュ文字列が必要な場合、文字列はプラグインに必要な形式ですすでにハッシュされているとみなされます。

例: パスワードをクリアテキストで指定します。デフォルトのプラグインが使用されます:

```
CREATE USER 'jeffrey'@'localhost'  
IDENTIFIED BY 'password';
```

例: 認証プラグインをクリアテキストのパスワード値とともに指定します:

```
CREATE USER 'jeffrey'@'localhost'  
IDENTIFIED WITH mysql_native_password BY 'password';
```

いずれの場合も、アカウント行に格納されるパスワード値は、アカウントに関連付けられた認証プラグインによってハッシュされたあとのクリアテキスト値'password'です。

パスワードと認証プラグインの設定の詳細は、[セクション6.2.14「アカウントパスワードの割り当て」](#)および[セクション6.2.17「プラグブル認証」](#)を参照してください。

CREATE USER ロールのオプション

DEFAULT ROLE 句は、ユーザーがサーバーに接続して認証したとき、またはセッション中にユーザーが **SET ROLE DEFAULT** ステートメントを実行したときにアクティブになるロールを定義します。

各ロール名は、[セクション6.2.5「ロール名の指定」](#)で説明されている形式を使用します。例:

```
CREATE USER 'joe'@'10.0.0.1' DEFAULT ROLE administrator, developer;
```

ロール名のホスト名部分は、省略すると '%' にデフォルト設定されます。

DEFAULT ROLE 句では、カンマ区切りのロール名のリストを使用できます。これらのロールは、**CREATE USER** の実行時に存在する必要はありません。

CREATE USER SSL/TLS オプション

MySQL では、ユーザー名と資格証明に基づく通常の認証に加えて、X.509 証明書属性をチェックできます。MySQL での SSL/TLS の使用に関する背景情報は、[セクション6.3「暗号化された接続の使用」](#)を参照してください。

MySQL アカウントの SSL/TLS 関連オプションを指定するには、1 つ以上の **tls_option** 値を指定する **REQUIRE** 句を使用します。

REQUIRE オプションの順序は関係ありませんが、オプションを 2 回指定することはできません。 **AND** キーワードは、**REQUIRE** オプション間のオプションです。

CREATE USER では、次の **tls_option** 値が許可されます:

- **NONE**

ステートメントで指定されたすべてのアカウントに SSL または X.509 要件がないことを示します。ユーザー名とパスワードが有効であれば、暗号化されていない接続が許可されます。クライアントに適切な証明書および鍵ファイルがある場合は、クライアントオプションで暗号化された接続を使用できます。

```
CREATE USER 'jeffrey'@'localhost' REQUIRE NONE;
```

クライアントは、デフォルトでセキュアな接続を確立しようとします。 **REQUIRE NONE** を持つクライアントでは、セキュアな接続を確立できない場合、接続試行は暗号化されていない接続にフォールバックされます。暗号化された接続を要求するには、クライアントは **--ssl-mode=REQUIRED** オプションのみを指定する必要があります。セキュアな接続を確立できない場合、接続の試行は失敗します。

SSL 関連の **REQUIRE** オプションが指定されていない場合、**NONE** がデフォルトです。

- **SSL**

ステートメントで指定されたすべてのアカウントに対して暗号化された接続のみを許可するようにサーバーに指示します。

```
CREATE USER 'jeffrey'@'localhost' REQUIRE SSL;
```

クライアントは、デフォルトでセキュアな接続を確立しようとします。 **REQUIRE SSL** を持つアカウントでは、セキュアな接続を確立できない場合、接続の試行は失敗します。

- X509

ステートメントで指定されたすべてのアカウントについて、クライアントは有効な証明書を提示する必要がありますが、正確な証明書、発行者およびサブジェクトは関係ありません。唯一の要件は、いずれかの CA 証明書でその署名を検証できるべきであるということです。X.509 証明書の使用は常に暗号化を意味するため、この場合は SSL オプションは必要ありません。

```
CREATE USER 'jeffrey'@'localhost' REQUIRE X509;
```

REQUIRE X509 のアカウントの場合、クライアントは接続する `--ssl-key` および `--ssl-cert` オプションを指定する必要があります。(サーバーによって提供される公開証明書を検証できるように、`--ssl-ca` も指定することをお勧めしますが、必須ではありません。)これらの REQUIRE オプションは X509 の要件を意味するため、これは ISSUER および SUBJECT にも当てはまります。

- ISSUER 'issuer'

ステートメントで指定されたすべてのアカウントについて、CA 'issuer'によって発行された有効な X.509 証明書をクライアントが提示する必要があります。クライアントが有効だが発行者が異なる証明書を提示した場合、サーバーは接続を拒否します。X.509 証明書の使用は常に暗号化を意味するため、この場合は SSL オプションは必要ありません。

```
CREATE USER 'jeffrey'@'localhost'  
REQUIRE ISSUER '/C=SE/ST=Stockholm/L=Stockholm/  
O=MySQL/CN=CA/emailAddress=ca@example.com';
```

ISSUER には X509 の要件があるため、クライアントは接続するために `--ssl-key` および `--ssl-cert` オプションを指定する必要があります。(サーバーによって提供される公開証明書を検証できるように、`--ssl-ca` も指定することをお勧めしますが、必須ではありません。)

- SUBJECT 'subject'

ステートメントで指定されたすべてのアカウントについて、クライアントがサブジェクト `subject` を含む有効な X.509 証明書を提示する必要があります。クライアントが有効だがサブジェクトが異なる証明書を提示した場合、サーバーは接続を拒否します。X.509 証明書の使用は常に暗号化を意味するため、この場合は SSL オプションは必要ありません。

```
CREATE USER 'jeffrey'@'localhost'  
REQUIRE SUBJECT '/C=SE/ST=Stockholm/L=Stockholm/  
O=MySQL demo client certificate/  
CN=client/emailAddress=client@example.com';
```

MySQL では、'subject' 値と証明書の値との単純な文字列比較が行われるため、大文字と小文字およびコンポーネントの順序は、証明書に存在するものとまったく同じにする必要があります。

SUBJECT には X509 の要件があるため、クライアントは接続するために `--ssl-key` および `--ssl-cert` オプションを指定する必要があります。(サーバーによって提供される公開証明書を検証できるように、`--ssl-ca` も指定することをお勧めしますが、必須ではありません。)

- CIPHER 'cipher'

ステートメントで指定されたすべてのアカウントには、接続を暗号化するための特定の暗号メソッドが必要です。このオプションは、十分な強度の暗号およびキー長が使用されるようにするために必要です。短い暗号化キーを使用する古いアルゴリズムを使用すると、暗号化が弱くなる可能性があります。

```
CREATE USER 'jeffrey'@'localhost'  
REQUIRE CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

SUBJECT、ISSUER および CIPHER オプションは、REQUIRE 句で組み合わせることができます:

```
CREATE USER 'jeffrey'@'localhost'  
REQUIRE SUBJECT '/C=SE/ST=Stockholm/L=Stockholm/  
O=MySQL demo client certificate/  
CN=client/emailAddress=client@example.com'  
AND ISSUER '/C=SE/ST=Stockholm/L=Stockholm/  
O=MySQL/CN=CA/emailAddress=ca@example.com'  
AND CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

CREATE USER リソース制限オプション

セクション6.2.20「アカウントリソース制限の設定」で説明されているように、アカウントによるサーバーリソースの使用に制限を設定できます。そのためには、1つ以上の `resource_option` 値を指定する `WITH` 句を使用します。

`WITH` オプションの順序は重要ではありませんが、特定のリソース制限が複数回指定された場合は、最後のインスタンスが優先されます。

`CREATE USER` では、次の `resource_option` 値が許可されます:

- `MAX_QUERIES_PER_HOUR count`, `MAX_UPDATES_PER_HOUR count`, `MAX_CONNECTIONS_PER_HOUR count`

ステートメントで指定されたすべてのアカウントについて、これらのオプションは、特定の1時間の間に各アカウントに許可されるクエリー、更新、およびサーバーへの接続の数を制限します。`count` が0 (デフォルト) である場合、これは、このアカウントに対する制限が存在しないことを示します。

- `MAX_USER_CONNECTIONS count`

ステートメントで指定されたすべてのアカウントについて、各アカウントによるサーバーへの同時接続の最大数を制限します。0以外の `count` は、このアカウントに対する制限を明示的に指定します。`count` が0 (デフォルト) である場合、サーバーは、`max_user_connections` システム変数のグローバル値からこのアカウントの同時接続の数を決定します。`max_user_connections` もゼロである場合は、アカウントに制限がありません。

例:

```
CREATE USER 'jeffrey'@'localhost'  
WITH MAX_QUERIES_PER_HOUR 500 MAX_UPDATES_PER_HOUR 100;
```

CREATE USER のパスワード管理オプション

`CREATE USER` では、パスワード管理用にいくつかの `password_option` 値がサポートされています:

- パスワードの有効期限オプション: アカウントパスワードを手動で期限切れにし、そのパスワード有効期限ポリシーを設定できます。ポリシーオプションによってパスワードが期限切れになることはありません。代わりに、最新のアカウントパスワード変更の日時から評価されるパスワード有効期限に基づいて、サーバーがアカウントに自動期限切れを適用する方法を決定します。
- パスワード再利用オプション: パスワードの再利用は、パスワード変更の数、経過時間、またはその両方に基づいて制限できます。
- パスワード検証必須オプション: 変更しようとしているユーザーが実際に現在のパスワードを認識していることを確認するために、アカウントパスワードの変更を試行する際に現在のパスワードを指定する必要があるかどうかを指定できます。
- 不正解 - パスワード失敗 - ログイントラッキングオプション: サーバーが失敗したログイン試行を追跡し、連続して正しくないパスワードが多すぎるアカウントを一時的にロックするようにすることができます。必要な失敗数とロック時間は構成可能です。

このセクションでは、パスワード管理オプションの構文について説明します。パスワード管理のポリシーの確立の詳細は、セクション6.2.15「パスワード管理」を参照してください。

特定のタイプの複数のパスワード管理オプションが指定されている場合は、最後のオプションが優先されます。たとえば、`PASSWORD EXPIRE DEFAULT PASSWORD EXPIRE NEVER` は `PASSWORD EXPIRE NEVER` と同じです。

注記

失敗したログイン追跡に関連するオプションを除き、パスワード管理オプションは、資格証明をMySQLに内部的に格納する認証プラグインを使用するアカウントにのみ適用されます。MySQLの外部にある資格証明システムに対して認証を実行するプラグインを使用するアカウントの場合、パスワード管理もそのシステムに対して外部で処理する必要があります。内部資格証明記憶域の詳細は、セクション6.2.15「パスワード管理」を参照してください。

アカウントパスワードが手動で期限切れになった場合、または自動期限切れポリシーに従ってパスワードの有効期間が許可された存続期間を超えたとみなされた場合、クライアントには期限切れのパスワードがあります。この場合、サーバーはクライアントを切断するか、クライアントに許可されている操作を制限します ([セクション6.2.16「期限切れパスワードのサーバー処理」](#)を参照)。制限付きクライアントによって実行される操作は、ユーザーが新しいアカウントパスワードを確立するまでエラーになります。

`CREATE USER` では、パスワードの有効期限を制御するために次の `password_option` 値が許可されます:

- `PASSWORD EXPIRE`

ステートメントで指定されたすべてのアカウントのパスワードをすぐに期限切れとしてマークします。

```
CREATE USER 'jeffrey'@'localhost' PASSWORD EXPIRE;
```

- `PASSWORD EXPIRE DEFAULT`

`default_password_lifetime` システム変数で指定されたグローバル有効期限ポリシーが適用されるように、ステートメントで指定されたすべてのアカウントを設定します。

```
CREATE USER 'jeffrey'@'localhost' PASSWORD EXPIRE DEFAULT;
```

- `PASSWORD EXPIRE NEVER`

この有効期限オプションは、ステートメントで指定されたすべてのアカウントのグローバルポリシーをオーバーライドします。それぞれについて、パスワードの有効期限が切れないようにパスワードの有効期限を無効にします。

```
CREATE USER 'jeffrey'@'localhost' PASSWORD EXPIRE NEVER;
```

- `PASSWORD EXPIRE INTERVAL N DAY`

この有効期限オプションは、ステートメントで指定されたすべてのアカウントのグローバルポリシーをオーバーライドします。それぞれについて、パスワードの存続期間が `N` 日に設定されます。次のステートメントでは、180日ごとにパスワードを変更する必要があります:

```
CREATE USER 'jeffrey'@'localhost' PASSWORD EXPIRE INTERVAL 180 DAY;
```

`CREATE USER` では、必要な最小パスワード変更数に基づいて以前のパスワードの再利用を制御するために、次の `password_option` 値が許可されています:

- `PASSWORD HISTORY DEFAULT`

`password_history` システム変数で指定された変更数の前にパスワードの再利用を禁止するために、パスワード履歴の長さに関するグローバルポリシーが適用されるように、ステートメントで指定されたすべてのアカウントを設定します。

```
CREATE USER 'jeffrey'@'localhost' PASSWORD HISTORY DEFAULT;
```

- `PASSWORD HISTORY N`

この履歴長オプションは、ステートメントで指定されたすべてのアカウントのグローバルポリシーをオーバーライドします。それぞれについて、最近選択した `N` パスワードの再利用を禁止するために、パスワード履歴の長さを `N` パスワードに設定します。次のステートメントは、以前の6つのパスワードの再利用を禁止します:

```
CREATE USER 'jeffrey'@'localhost' PASSWORD HISTORY 6;
```

`CREATE USER` では、経過時間に基づいて以前のパスワードの再利用を制御するために、次の `password_option` 値が許可されます:

- `PASSWORD REUSE INTERVAL DEFAULT`

経過時間に関するグローバルポリシーが適用され、`password_reuse_interval` システム変数で指定された日数よりも新しいパスワードの再利用が禁止されるように、アカウントで指定されたすべてのステートメントを設定します。

```
CREATE USER 'jeffrey'@'localhost' PASSWORD REUSE INTERVAL DEFAULT;
```

- `PASSWORD REUSE INTERVAL N DAY`

この time-elapsed オプションは、ステートメントで指定されたすべてのアカウントのグローバルポリシーをオーバーライドします。それぞれについて、パスワードの再利用間隔を **N** 日に設定して、その日数より新しいパスワードの再利用を禁止します。次のステートメントは、360 日間のパスワードの再利用を禁止します:

```
CREATE USER 'jeffrey'@'localhost' PASSWORD REUSE INTERVAL 360 DAY;
```

CREATE USER では、アカウントパスワードの変更の試行で現在のパスワードを指定する必要があるかどうかを制御するために、変更しようとしているユーザーが実際に現在のパスワードを知っていることを確認するために、次の **password_option** 値を許可しています:

- **PASSWORD REQUIRE CURRENT**

この検証オプションは、ステートメントで指定されたすべてのアカウントのグローバルポリシーをオーバーライドします。それぞれについて、パスワードの変更で現在のパスワードを指定する必要があります。

```
CREATE USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT;
```

- **PASSWORD REQUIRE CURRENT OPTIONAL**

この検証オプションは、ステートメントで指定されたすべてのアカウントのグローバルポリシーをオーバーライドします。それぞれについて、パスワードを変更して現在のパスワードを指定する必要はありません。(現在のパスワードを指定する必要がありますが、指定する必要はありません。)

```
CREATE USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT OPTIONAL;
```

- **PASSWORD REQUIRE CURRENT DEFAULT**

password_require_current システム変数で指定されたパスワード検証に関するグローバルポリシーが適用されるように、アカウントで指定されたすべてのステートメントを設定します。

```
CREATE USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT DEFAULT;
```

MySQL 8.0.19 の時点で、**CREATE USER** は、失敗したログイン追跡を制御するために次の **password_option** 値を許可します:

- **FAILED_LOGIN_ATTEMPTS N**

不正なパスワードを指定するアカウントログイン試行を追跡するかどうか。 **N** は 0 から 32767 の数値である必要があります。値 0 を指定すると、失敗したログイントラッキングが無効になります。0 より大きい値は、パスワードが何回連続して失敗したために一時アカウントがロックされるかを示します (**PASSWORD_LOCK_TIME** もゼロ以外の場合)。

- **PASSWORD_LOCK_TIME {N | UNBOUNDED}**

連続して何度もログインを試行した後にアカウントをロックする期間。パスワードが正しくありません。 **N** は、0 から 32767 の数値、または **UNBOUNDED** である必要があります。値 0 を指定すると、一時アカウントロックが無効になります。0 より大きい値は、アカウントをロックする期間を日数で示します。値が **UNBOUNDED** の場合、アカウントのロック期間は無制限になります。ロックされると、アカウントはロック解除されるまでロック状態のままになります。ロック解除が発生する条件の詳細は、[失敗したログイントラッキングと一時アカウントロック](#) を参照してください。

ログイン失敗トラッキングと一時ロックを実行するには、アカウントの **FAILED_LOGIN_ATTEMPTS** オプションと **PASSWORD_LOCK_TIME** オプションの両方をゼロ以外にする必要があります。次のステートメントは、パスワードが 4 回連続して失敗した後も 2 日間ロックされたままになるアカウントを作成します:

```
CREATE USER 'jeffrey'@'localhost'  
  FAILED_LOGIN_ATTEMPTS 4 PASSWORD_LOCK_TIME 2;
```

CREATE USER アカウントロックオプション

MySQL では、アカウントのロック状態を指定する **ACCOUNT LOCK** および **ACCOUNT UNLOCK** オプションを使用したアカウントのロックおよびロック解除がサポートされています。詳細は、[セクション6.2.19「アカウントロック」](#)を参照してください。

複数の account-locking オプションが指定されている場合は、最後のオプションが優先されます。

CREATE USER バイナリロギング

`CREATE USER` は、成功した場合はバイナリログに書き込まれますが、失敗した場合は書き込まれません。その場合、ロールバックが発生し、変更は行われません。バイナリログに書き込まれるステートメントには、指定されたすべてのユーザーが含まれます。`IF NOT EXISTS` 句が指定されている場合は、すでに存在していて作成されていないユーザーも含まれます。

バイナリログに書き込まれるステートメントは、次のように決定される各ユーザーの認証プラグインを指定します:

- 元のステートメントで指定されたプラグイン (指定されている場合)。
- それ以外の場合は、デフォルトの認証プラグイン。特に、ユーザー `u1` がすでに存在し、デフォルト以外の認証プラグインを使用している場合、`CREATE USER IF NOT EXISTS u1` のバイナリログに書き込まれるステートメントはデフォルトの認証プラグインに名前を付けます。(バイナリログに書き込まれたステートメントがユーザーのデフォルト以外の認証プラグインを指定する必要がある場合は、それを元のステートメントに含めます。)

サーバーは、バイナリログに書き込まれたステートメント内に存在しないユーザーのデフォルトの認証プラグインを追加すると、それらのユーザーを指定する警告をエラーログに書き込みます。

元のステートメントで `FAILED_LOGIN_ATTEMPTS` または `PASSWORD_LOCK_TIME` オプションが指定されている場合、バイナリログに書き込まれるステートメントにはそのオプションが含まれます。

13.7.1.4 DROP ROLE ステートメント

```
DROP ROLE [IF EXISTS] role [, role ] ...
```

`DROP ROLE` により、1 つ以上のロール (権限の名前付きコレクション) が削除されます。このステートメントを使用するには、グローバル `DROP ROLE` または `CREATE USER` 権限が必要です。`read_only` システム変数が有効になっている場合、`DROP ROLE` にはさらに `CONNECTION_ADMIN` 権限 (または非推奨の `SUPER` 権限) が必要です。

MySQL 8.0.16 では、`CREATE USER` 権限を持つユーザーはこのステートメントを使用して、ロックまたはロック解除されたアカウントを削除できます。`DROP ROLE` 権限を持つユーザーは、このステートメントを使用して、ロックされているアカウントのみを削除できます (ロック解除されたアカウントは、ロールとしてではなく、サーバーへのロギングに使用される可能性が高いユーザーアカウントです)。

`mandatory_roles` システム変数値で指定されたロールは削除できません。

`DROP ROLE` は、すべての名前付きロールに対して成功するか、ロールバックされ、エラーが発生しても効果はありません。デフォルトでは、存在しないロールを削除しようとすると、エラーが発生します。`IF EXISTS` 句を指定すると、ステートメントは、存在しない名前付きロールごとにエラーではなく警告を生成します。

ステートメントが成功した場合はバイナリログに書き込まれますが、失敗した場合は書き込まれず、ロールバックが発生して変更は行われません。バイナリログに書き込まれるステートメントには、すべての名前付き役割が含まれます。`IF EXISTS` 句が指定されている場合は、存在せず、削除されなかったロールも含まれます。

各ロール名は、[セクション6.2.5「ロール名の指定」](#) で説明されている形式を使用します。例:

```
DROP ROLE 'administrator', 'developer';  
DROP ROLE 'webapp'@'localhost';
```

ロール名のホスト名部分は、省略すると '%' にデフォルト設定されます。

削除されたロールは、そのロールが付与されたユーザーアカウント (またはロール) から自動的に取り消されます。このようなアカウントの現在のセッション内では、調整された権限は、次に実行されるステートメントから適用されません。

ロールの使用例は、[セクション6.2.10「ロールの使用」](#) を参照してください。

13.7.1.5 DROP USER ステートメント

```
DROP USER [IF EXISTS] user [, user ] ...
```

DROP USER ステートメントは、1 つ以上の MySQL アカウントとその権限を削除します。これにより、そのアカウントの権限行がすべての付与テーブルから削除されます。

mandatory_roles システム変数値で指定されたロールは削除できません。

DROP USER を使用するには、**mysql** システムスキーマに対するグローバル **CREATE USER** 権限または **DELETE** 権限が必要です。**read_only** システム変数が有効になっている場合、**DROP USER** にはさらに **CONNECTION_ADMIN** 権限 (または非推奨の **SUPER** 権限) が必要です。

MySQL 8.0.22 では、削除するアカウントの名前がストアオブジェクトの **DEFINER** 属性として指定されている場合、**DROP USER** はエラーで失敗します。(つまり、アカウントを削除すると、格納されたオブジェクトが孤立する場合、ステートメントは失敗します。) 操作を実行するには、**SET_USER_ID** 権限が必要です。この場合、ステートメントはエラーで失敗するのではなく、警告付きで成功します。特定のアカウントを **DEFINER** 属性として指定するオブジェクトの識別方法などの追加情報は、**孤立したストアオブジェクト** を参照してください。

DROP USER は、指定されたすべてのユーザーに対して成功するか、ロールバックされ、エラーが発生しても効果はありません。デフォルトでは、存在しないユーザーを削除しようとすると、エラーが発生します。**IF EXISTS** 句を指定すると、ステートメントは、存在しない指定ユーザーごとに、エラーではなく、警告を生成します。

ステートメントが成功した場合はバイナリログに書き込まれますが、失敗した場合は書き込まれず、ロールバックが発生して変更は行われません。バイナリログに書き込まれるステートメントには、指定されたすべてのユーザーが含まれます。**IF EXISTS** 句が指定されている場合、これには存在せず、削除されなかったユーザーも含まれます。

各アカウント名には、**セクション6.2.4「アカウント名の指定」**で説明されている形式が使用されます。例:

```
DROP USER 'jeffrey'@'localhost';
```

アカウント名のホスト名部分は、省略すると '%' にデフォルト設定されます。

重要

DROP USER は、開かれたどのユーザーセッションも自動的に閉じません。さらに、開かれたセッションを持つユーザーが削除されても、このステートメントはそのユーザーのセッションが閉じられるまで有効になりません。セッションがクローズされると、ユーザーは削除され、次のログイン試行は失敗します。これは意図的なものです。

DROP USER は、古いユーザーが作成したどのデータベースまたはそれらのデータベース内のどのオブジェクトも自動的に削除したり、無効にしたりしません。これには、**DEFINER** 属性に削除されたユーザーが指定されているストアプログラムまたはビューが含まれます。このようなオブジェクトにアクセスしようとすると、それが定義者のセキュリティコンテキストで実行された場合は、エラーが生成される可能性があります。(セキュリティコンテキストについては、**セクション25.6「ストアオブジェクトのアクセス制御」**を参照してください。)

13.7.1.6 GRANT ステートメント

```
GRANT
  priv_type [(column_list)]
  [, priv_type [(column_list))] ...
ON [object_type] priv_level
TO user_or_role [, user_or_role] ...
[WITH GRANT OPTION]
[AS user
  [WITH ROLE
  DEFAULT
  | NONE
  | ALL
  | ALL EXCEPT role [, role] ...
  | role [, role] ...
  ]
]
}

GRANT PROXY ON user_or_role
TO user_or_role [, user_or_role] ...
[WITH GRANT OPTION]

GRANT role [, role] ...
TO user_or_role [, user_or_role] ...
```

```
[WITH ADMIN OPTION]

object_type: {
  TABLE
  | FUNCTION
  | PROCEDURE
}

priv_level: {
  *
  | *.*
  | db_name.*
  | db_name.tbl_name
  | tbl_name
  | db_name.routine_name
}

user_or_role: {
  user (see セクション6.2.4「アカウント名の指定」)
  | role (see セクション6.2.5「ロール名の指定」)
}
```

GRANT ステートメントは、MySQL のユーザーアカウントおよびロールに権限およびロールを割り当てます。次のトピックで説明するように、GRANT ステートメントにはいくつかの側面があります:

- [GRANT の一般概要](#)
- [オブジェクト見積ガイドライン](#)
- [アカウント名](#)
- [MySQL によってサポートされる権限](#)
- [グローバル権限](#)
- [データベース権限](#)
- [テーブル権限](#)
- [カラム権限](#)
- [ストアドルーチン権限](#)
- [プロキシユーザー権限](#)
- [ロールの付与](#)
- [AS 句および権限の制限事項](#)
- [その他のアカウント特性](#)
- [MySQL バージョンと標準 SQL バージョンの GRANT](#)

GRANT の一般概要

GRANT ステートメントを使用すると、システム管理者は、ユーザーアカウントおよびロールに付与できる権限およびロールを付与できます。次の構文制限が適用されます:

- GRANT では、権限とロールの両方の付与を同じステートメントに混在させることはできません。特定の GRANT ステートメントでは、権限またはロールのいずれかを付与する必要があります。
- ON 句は、ステートメントによって権限が付与されるかロールが付与されるかを区別します:
 - ON では、ステートメントによって権限が付与されます。
 - ON がない場合、ステートメントはロールを付与します。
 - 権限とロールの両方をアカウントに割り当てることはできますが、付与する内容に適した構文を持つ個別の GRANT ステートメントを使用する必要があります。

ロールの詳細は、[セクション6.2.10「ロールの使用」](#)を参照してください。

`GRANT` で権限を付与するには、`GRANT OPTION` 権限および付与する権限が必要です。(または、`mysql` システムスキーマ内の付与テーブルに対する `UPDATE` 権限を持っている場合は、任意のアカウントに任意の権限を付与できます。) `read_only` システム変数が有効になっている場合、`GRANT` にはさらに `CONNECTION_ADMIN` 権限 (または非推奨の `SUPER` 権限) が必要です。

`GRANT` は、指定されたすべてのユーザーおよびロールに対して成功するか、ロールバックされ、エラーが発生しても効果はありません。ステートメントは、指定されたすべてのユーザーおよび役割で成功した場合のみバイナリログに書き込まれます。

`REVOKE` ステートメントは `GRANT` に関連しており、管理者がアカウントの権限を削除できるようにします。[セクション13.7.1.8「REVOKE ステートメント」](#)を参照してください。

各アカウント名には、[セクション6.2.4「アカウント名の指定」](#)で説明されている形式が使用されます。各ロール名は、[セクション6.2.5「ロール名の指定」](#)で説明されている形式を使用します。例:

```
GRANT ALL ON db1.* TO 'jeffrey'@'localhost';
GRANT 'role1', 'role2' TO 'user1'@'localhost', 'user2'@'localhost';
GRANT SELECT ON world.* TO 'role3';
```

アカウント名またはロール名のホスト名部分は、省略すると '%' にデフォルト設定されます。

通常、データベース管理者は最初に `CREATE USER` を使用してアカウントを作成し、その非権限特性 (パスワード、セキュアな接続を使用するかどうか、サーバーリソースへのアクセス制限など) を定義してから、`GRANT` を使用してその権限を定義します。`ALTER USER` を使用して、既存のアカウントの非権限特性を変更できます。例:

```
CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'password';
GRANT ALL ON db1.* TO 'jeffrey'@'localhost';
GRANT SELECT ON db2.invoice TO 'jeffrey'@'localhost';
ALTER USER 'jeffrey'@'localhost' WITH MAX_QUERIES_PER_HOUR 90;
```

正常に実行されると、`GRANT` は `mysql` プログラムから `Query OK, 0 rows affected` で応答します。この操作によってどのような権限が付与されたかを判定するには、`SHOW GRANTS` を使用します。[セクション13.7.7.21「SHOW GRANTS ステートメント」](#)を参照してください。

重要

状況によっては、`GRANT` がサーバーログ、またはクライアント側にある `~/mysql_history` などの履歴ファイル内に記録されることがあります。つまり、平文のパスワードが、その情報に対する読み取りアクセス権を持つ任意のユーザーによって読み取られる可能性があります。これがサーバーログで発生する条件およびこれを制御する方法については、[セクション6.1.2.3「パスワードおよびロギング」](#)を参照してください。クライアント側のロギングに関する同様の情報については、[セクション4.5.1.3「mysql クライアントロギング」](#)を参照してください。

`GRANT` では、255 文字までのホスト名 (MySQL 8.0.17 より前の 60 文字) がサポートされます。ユーザー名には、最大 32 文字を指定できます。データベース、テーブル、カラム、およびルーチン名には、最大 64 文字を指定できます。

警告

`Do` は、`mysql.user` システムテーブルを変更しても、ユーザー名に許可されている長さを変更しようとしません。これを行うと、予期しない動作が発生し、ユーザーが MySQL server にログインできなくなることもあります。[セクション2.11「MySQL のアップグレード」](#)で説明されている手順以外の方法で、`mysql` システムスキーマ内のテーブルの構造を変更しないでください。

オブジェクト見積ガイドライン

多くの場合、引用符はオプションですが、`GRANT` ステートメント内のいくつかのオブジェクトは引用の対象となります: アカウント名、ロール名、データベース名、テーブル名、カラム名およびルーチン名。たとえば、アカウント名の `user_name` または `host_name` の値が引用符で囲まれていない識別子として有効な場合、引用符で囲む必要はあ

りません。ただし、特殊文字 (- など) を含む `user_name` 文字列、または % などの特殊文字やワイルドカード文字を含む `host_name` 文字列 ('test-user'@'%.com' など) を指定するには、引用符が必要です。ユーザー名とホスト名は個別に引用符で囲みます。

引用符で囲まれた値を指定するには:

- データベース名、テーブル名、カラム名およびルーチン名を識別子として引用符で囲みます。
- ユーザー名とホスト名は識別子または文字列として引用符で囲みます。
- パスワードは文字列として引用符で囲みます。

文字列および識別子として引用符で囲む方法のガイドラインについては、[セクション9.1.1「文字列リテラル」](#) および [セクション9.2「スキーマオブジェクト名」](#) を参照してください。

`_` および `%` のワイルドカードは、データベースレベル (`GRANT ... ON db_name.*`) で権限を付与する `GRANT` ステートメントでデータベース名を指定する場合に使用できます。つまり、`_` 文字をデータベース名の一部として使用するには、`GRANT` ステートメントで `_` として指定し、ワイルドカードパターンに一致する追加のデータベース (`GRANT ... ON 'foo_bar'.* TO ...` など) にユーザーがアクセスできないようにします。

データベース名を使用してデータベースレベルで権限を付与せず、テーブルやルーチン (`GRANT ... ON db_name.tbl_name` など) などの他のオブジェクトに権限を付与する修飾子として使用すると、ワイルドカード文字は通常の文字として扱われます。

アカウント名

`GRANT` ステートメントの `user` 値は、そのステートメントが適用される MySQL アカウントを示します。任意のホストからのユーザーへの権限の付与に対応するために、MySQL では、'`user_name`'@'`host_name`'形式での `user` 値の指定がサポートされています。

ホスト名には、ワイルドカードを指定できます。たとえば、'`user_name`'@'%example.com' は `example.com` ドメイン内の任意のホストの `user_name` に適用され、'`user_name`'@'198.51.100.%' は `198.51.100` クラス C サブネット内の任意のホストの `user_name` に適用されます。

単純な形式の '`user_name`' は、'`user_name`'@'%' のシノニムです。

MySQL は、ユーザー名でのワイルドカードをサポートしていません。匿名ユーザーを参照するには、`GRANT` ステートメントで空のユーザー名を含むアカウントを指定します。

```
GRANT ALL ON test.* TO ''@'localhost' ...;
```

この場合、匿名ユーザーの正しいパスワードを使用してローカルホストから接続するユーザーは、匿名ユーザーアカウントに関連付けられた権限を使用してアクセスを許可されます。

アカウント名内のユーザー名とホスト名の値の詳細は、[セクション6.2.4「アカウント名の指定」](#) を参照してください。

警告

ローカル匿名ユーザーに MySQL サーバーへの接続を許可する場合は、'`user_name`'@'`localhost`'としてすべてのローカルユーザーにも権限を付与する必要があります。それ以外の場合は、指定されたユーザーがローカルマシンから MySQL サーバーにログインしようとするとき、`mysql.user` システムテーブルの `localhost` の匿名ユーザーアカウントが使用されます。詳細は、[セクション6.2.6「アクセス制御、ステージ 1: 接続の検証」](#) を参照してください。

この問題が適用されるかどうかを判断するには、匿名ユーザーをリストする次のクエリーを実行します:

```
SELECT Host, User FROM mysql.user WHERE User="";
```

今説明した問題を回避するには、次のステートメントを使用して、ローカルの匿名ユーザーアカウントを削除します。

```
DROP USER '@localhost';
```

MySQL によってサポートされる権限

次のテーブルに、**GRANT** および **REVOKE** ステートメントに指定できる静的および動的な `priv_type` 権限タイプと、各権限を付与できるレベルをまとめます。各権限の詳細は、[セクション6.2.2「MySQLで提供される権限」](#)を参照してください。静的権限と動的権限の違いの詳細は、[静的権限と動的権限](#)を参照してください。

表 13.11 GRANT および REVOKE に許可される静的権限

権限	意味と付与可能なレベル
ALL [PRIVILEGES]	GRANT OPTION および PROXY を除くすべての権限を指定されたアクセスレベルで付与します。
ALTER	ALTER TABLE の使用を有効にします。レベル: グローバル、データベース、テーブル。
ALTER ROUTINE	ストアドルーチンの変更または削除を有効にします。レベル: Global, database, routine.
CREATE	データベースおよびテーブルの作成を有効にします。レベル: グローバル、データベース、テーブル。
CREATE ROLE	ロール作成を有効にします。レベル: グローバル。
CREATE ROUTINE	ストアドルーチンの作成を有効にします。レベル: グローバル、データベース。
CREATE TABLESPACE	テーブルスペースおよびログファイルグループの作成、変更、または削除を有効にします。レベル: グローバル。
CREATE TEMPORARY TABLES	CREATE TEMPORARY TABLE の使用を有効にします。レベル: グローバル、データベース。
CREATE USER	CREATE USER 、 DROP USER 、 RENAME USER 、および REVOKE ALL PRIVILEGES の使用を有効にします。レベル: グローバル。
CREATE VIEW	ビューの作成または変更を有効にします。レベル: グローバル、データベース、テーブル。
DELETE	DELETE の使用を有効にします。レベル: グローバル、データベース、テーブル。
DROP	データベース、テーブル、およびビューの削除を有効にします。レベル: グローバル、データベース、テーブル。
DROP ROLE	ロールの削除を有効にします。レベル: グローバル。
EVENT	イベントスケジューラでのイベントの使用を有効にします。レベル: グローバル、データベース。
EXECUTE	ユーザーがストアドルーチンを実行できるようにします。レベル: Global, database, routine.
FILE	ユーザーがサーバーにファイルを読み取らせたり書き込ませたりできるようにします。レベル: グローバル。
GRANT OPTION	権限のほかのアカウントへの付与、またはほかのアカウントからの削除を有効にします。レベル: Global, database, table, routine, proxy.
INDEX	インデックスの作成または削除を有効にします。レベル: グローバル、データベース、テーブル。
INSERT	INSERT の使用を有効にします。レベル: グローバル、データベース、テーブル、カラム。
LOCK TABLES	ユーザーが SELECT 権限を持っているテーブルに対する LOCK TABLES の使用を有効にします。レベル: グローバル、データベース。

権限	意味と付与可能なレベル
PROCESS	ユーザーが <code>SHOW PROCESSLIST</code> を使用してすべてのプロセスを表示できるようにします。レベル: グローバル。
PROXY	ユーザーのプロキシ設定を有効にします。レベル: ユーザーからユーザーへ。
REFERENCES	外部キーの作成を有効にします。レベル: グローバル、データベース、テーブル、カラム。
RELOAD	<code>FLUSH</code> 操作の使用を有効にします。レベル: グローバル。
REPLICATION CLIENT	ユーザーがソースサーバーまたはレプリカサーバーの場所を尋ねることができるようにします。レベル: グローバル。
REPLICATION SLAVE	レプリカがソースからバイナリログイベントを読み取ることを有効にします。レベル: グローバル。
SELECT	<code>SELECT</code> の使用を有効にします。レベル: グローバル、データベース、テーブル、カラム。
SHOW DATABASES	<code>SHOW DATABASES</code> がすべてのデータベースを表示できるようにします。レベル: グローバル。
SHOW VIEW	<code>SHOW CREATE VIEW</code> の使用を有効にします。レベル: グローバル、データベース、テーブル。
SHUTDOWN	<code>mysqladmin shutdown</code> の使用を有効にします。レベル: グローバル。
SUPER	<code>CHANGE REPLICATION SOURCE TO</code> , <code>CHANGE MASTER TO</code> , <code>KILL</code> , <code>PURGE BINARY LOGS</code> , <code>SET GLOBAL</code> や <code>mysqladmin debug</code> コマンドなどの他の管理操作の使用を有効にします。レベル: グローバル。
TRIGGER	トリガー操作を有効にします。レベル: グローバル、データベース、テーブル。
UPDATE	<code>UPDATE</code> の使用を有効にします。レベル: グローバル、データベース、テーブル、カラム。
USAGE	「権限なし」のシノニムです

表 13.12 GRANT および REVOKE に許可される動的権限

権限	意味と付与可能なレベル
APPLICATION_PASSWORD_ADMIN	デュアルパスワード管理を有効にします。レベル: グローバル。
AUDIT_ADMIN	監査ログ構成を有効にします。レベル: グローバル。
BACKUP_ADMIN	バックアップ管理を有効にします。レベル: グローバル。
BINLOG_ADMIN	バイナリログ制御を有効にします。レベル: グローバル。
BINLOG_ENCRYPTION_ADMIN	バイナリログ暗号化のアクティブ化および非アクティブ化を有効にします。レベル: グローバル。
CLONE_ADMIN	クローン管理を有効にします。レベル: グローバル。
CONNECTION_ADMIN	接続制限/制限制御を有効にします。レベル: グローバル。
ENCRYPTION_KEY_ADMIN	InnoDB キーローテーションを有効にします。レベル: グローバル。
FIREWALL_ADMIN	ファイアウォールルール管理 (任意のユーザー) を有効にします。レベル: グローバル。

権限	意味と付与可能なレベル
FIREWALL_USER	ファイアウォールルール管理を有効にします (self)。レベル: グローバル。
FLUSH_OPTIMIZER_COSTS	オプティマイザコストのリロードを有効にします。レベル: グローバル。
FLUSH_STATUS	ステータスインジケータのフラッシュを有効にします。レベル: グローバル。
FLUSH_TABLES	テーブルのフラッシュを有効にします。レベル: グローバル。
FLUSH_USER_RESOURCES	ユーザーリソースのフラッシュを有効にします。レベル: グローバル。
GROUP_REPLICATION_ADMIN	Group Replication 制御を有効にします。レベル: グローバル。
INNODB_REDO_LOG_ENABLE	redo ロギングを有効または無効にします。レベル: グローバル。
INNODB_REDO_LOG_ARCHIVE	redo ログアーカイブ管理を有効にします。レベル: グローバル。
NDB_STORED_USER	SQL ノード間でのユーザーまたは役割の共有を有効にします (NDB Cluster)。レベル: グローバル。
PERSIST_RO_VARIABLES_ADMIN	読取り専用システム変数の永続化を有効にします。レベル: グローバル。
REPLICATION_APPLIER	レプリケーションチャンネルの PRIVILEGE_CHECKS_USER として機能します。レベル: グローバル。
REPLICATION_SLAVE_ADMIN	通常のレプリケーション制御を有効にします。レベル: グローバル。
RESOURCE_GROUP_ADMIN	リソースグループの管理を有効にします。レベル: グローバル。
RESOURCE_GROUP_USER	リソースグループの管理を有効にします。レベル: グローバル。
ROLE_ADMIN	WITH ADMIN OPTION を使用して、ロールの付与または取消しを可能にします。レベル: グローバル。
SESSION_VARIABLES_ADMIN	制限付きセッションシステム変数の設定を有効にします。レベル: グローバル。
SET_USER_ID	非自己 DEFINER 値の設定を有効にします。レベル: グローバル。
SHOW_ROUTINE	ストアドルーチン定義へのアクセスを有効にします。レベル: グローバル。
SYSTEM_USER	アカウントをシステムアカウントとして指定します。レベル: グローバル。
SYSTEM_VARIABLES_ADMIN	グローバルシステム変数の変更または永続化を有効にします。レベル: グローバル。
TABLE_ENCRYPTION_ADMIN	デフォルトの暗号化設定のオーバーライドを有効にします。レベル: グローバル。
VERSION_TOKEN_ADMIN	バージョントークン UDF の使用を有効にします。レベル: グローバル。
XA_RECOVER_ADMIN	XA RECOVER の実行を有効にします。レベル: グローバル。

トリガーはテーブルに関連付けられます。トリガーを作成または削除するには、トリガーではなくテーブルに対する **TRIGGER** 権限が必要です。

GRANT ステートメントでは、**ALL [PRIVILEGES]** または **PROXY** 権限は単独で指定する必要があり、ほかの権限とともに指定することはできません。**ALL [PRIVILEGES]** は、**GRANT OPTION** および **PROXY** 権限を除き、権限が付与されるレベルで使用可能なすべての権限を表します。

MySQL アカウント情報は、**mysql** システムスキーマのテーブルに格納されます。詳細は、**mysql** システムスキーマおよびアクセス制御システムについて詳しく説明している [セクション6.2「アクセス制御とアカウント管理」](#) を参照してください。

付与テーブルに、大文字と小文字が混在したデータベースまたはテーブル名を含む権限行が保持されており、かつ **lower_case_table_names** システム変数が 0 以外の値に設定されている場合は、**REVOKE** を使用してこれらの権限を取り消すことはできません。このような場合は、付与テーブルを直接操作する必要があります。(**lower_case_table_names** が設定されている場合、**GRANT** はこのような行を作成しませんが、その変数を設定する前にこのような行が作成されている可能性があります。 **lower_case_table_names** 設定は、サーバーの起動時のみ構成できます。)

権限は、**ON** 句に使用される構文に応じて、いくつかのレベルで付与できます。**REVOKE** の場合、同じ **ON** 構文で削除する権限を指定します。

グローバル、データベース、テーブル、およびルーチンレベルの場合、**GRANT ALL** は、付与しようとしているレベルに存在する権限のみを割り当てます。たとえば、**GRANT ALL ON db_name.*** はデータベースレベルのステートメントであるため、**FILE** などのグローバルのみの権限を付与しません。**ALL** を付与しても、**GRANT OPTION** または **PROXY** 権限は割り当てられません。

object_type 句 (存在する場合) は、それに続くオブジェクトがテーブル、ストアドファンクション、またはストアドプロシージャであるときは **TABLE**、**FUNCTION**、または **PROCEDURE** として指定するようにしてください。

ユーザーがデータベース、テーブル、カラムまたはルーチンに対して保持する権限は、グローバルレベルを含む各権限レベルのアカウント権限の論理 **OR** として追加的に形成されます。下位レベルの権限がないため、上位レベルで付与された権限を拒否できません。たとえば、次のステートメントは **SELECT** および **INSERT** 権限をグローバルに付与します:

```
GRANT SELECT, INSERT ON *.* TO u1;
```

グローバルに付与された権限は、これらの下位レベルでは付与されませんが、すべてのデータベース、テーブルおよびカラムに適用されます。

MySQL 8.0.16 では、**partial_revokes** システム変数が有効になっている場合、特定のデータベースに対して権限を取り消すことで、グローバルレベルで付与された権限を明示的に拒否できます:

```
GRANT SELECT, INSERT, UPDATE ON *.* TO u1;  
REVOKE INSERT, UPDATE ON db1.* FROM u1;
```

前述のステートメントの結果、**SELECT** はすべてのテーブルにグローバルに適用されますが、**INSERT** および **UPDATE** は **db1** のテーブルを除くグローバルに適用されます。**db1** へのアカウントアクセスは読取り専用です。

権限確認手順の詳細については、[セクション6.2.7「アクセス制御、ステージ 2: リクエストの確認」](#) で説明されています。

1 人のユーザーのテーブル、カラム、またはルーチン権限を使用している場合でも、サーバーはすべてのユーザーのテーブル、カラム、およびルーチン権限を検査するため、これにより MySQL が少し遅くなります。同様に、いずれかのユーザーのクエリー、更新、または接続の数を制限した場合、サーバーはこれらの値をモニターする必要があります。

MySQL では、存在しないデータベースまたはテーブルに対する権限を付与できます。テーブルの場合は、付与される権限に **CREATE** 権限が含まれている必要があります。この動作は設計によるものであり、データベース管理者がユーザーアカウントと、あとで作成されるデータベースまたはテーブルに対する権限を準備できるようにすることを目的としています。

重要

MySQL では、データベースまたはテーブルを削除しても、どの権限も自動的に取り消されません。ただし、ルーチンを削除した場合は、そのルーチンに付与されたルーチンレベルの権限がすべて取り消されます。

グローバル権限

グローバル権限は管理権限です。つまり、特定のサーバー上のすべてのデータベースに適用されます。グローバル権限を割り当てるには、`ON *.*` 構文を使用します。

```
GRANT ALL ON *.* TO 'someuser'@'somehost';
GRANT SELECT, INSERT ON *.* TO 'someuser'@'somehost';
```

`CREATE TABLESPACE`, `CREATE USER`, `FILE`, `PROCESS`, `RELOAD`, `REPLICATION CLIENT`, `REPLICATION SLAVE`, `SHOW DATABASES`, `SHUTDOWN` および `SUPER` の静的権限は管理権限であり、グローバルにのみ付与できます。

動的権限はすべてグローバルであり、グローバルにのみ付与できます。

その他の権限はグローバルに、またはより具体的なレベルで付与できます。

グローバルレベルで付与される `GRANT OPTION` の影響は、静的権限と動的権限で異なります：

- 静的グローバル権限に付与された `GRANT OPTION` は、すべての静的グローバル権限に適用されます。
- 動的権限に付与された `GRANT OPTION` は、その動的権限にのみ適用されます。

グローバルレベルの `GRANT ALL` では、すべての静的グローバル権限および現在登録されているすべての動的権限が付与されます。`GRANT` ステートメントの実行後に登録された動的権限は、どのアカウントにも遡及的に付与されません。

MySQL では、グローバル権限は `mysql.user` システムテーブルに格納されます。

データベース権限

データベース権限は、特定のデータベース内のすべてのオブジェクトに適用されます。データベースレベルの権限を割り当てるには、`ON db_name.*` 構文を使用します。

```
GRANT ALL ON mydb.* TO 'someuser'@'somehost';
GRANT SELECT, INSERT ON mydb.* TO 'someuser'@'somehost';
```

(`ON *.*` ではなく) `ON *` 構文を使用する場合、権限はデフォルトデータベースのデータベースレベルで割り当てられません。デフォルトデータベースが存在しない場合は、エラーが発生します。

`CREATE`, `DROP`, `EVENT`, `GRANT OPTION`, `LOCK TABLES` および `REFERENCES` 権限は、データベースレベルで指定できます。また、テーブルまたはルーチン権限もデータベースレベルで指定できます。この場合、これらの権限はデータベース内のすべてのテーブルまたはルーチンに適用されます。

MySQL では、データベース権限は `mysql.db` システムテーブルに格納されます。

テーブル権限

テーブル権限は、特定のテーブル内のすべてのカラムに適用されます。テーブルレベルの権限を割り当てるには、`ON db_name.tbl_name` 構文を使用します。

```
GRANT ALL ON mydb.mytbl TO 'someuser'@'somehost';
GRANT SELECT, INSERT ON mydb.mytbl TO 'someuser'@'somehost';
```

`db_name.tbl_name` ではなく `tbl_name` を指定した場合、このステートメントは、デフォルトデータベース内の `tbl_name` に適用されます。デフォルトデータベースが存在しない場合は、エラーが発生します。

テーブルレベルで許可される `priv_type` 値は、`ALTER`, `CREATE VIEW`, `CREATE`, `DELETE`, `DROP`, `GRANT OPTION`, `INDEX`, `INSERT`, `REFERENCES`, `SELECT`, `SHOW VIEW`, `TRIGGER` および `UPDATE` です。

テーブルレベルの権限は、実テーブルおよびビューに適用されます。テーブル名が一致していても、[CREATE TEMPORARY TABLE](#) で作成されたテーブルには適用されません。[TEMPORARY](#) テーブルの権限の詳細は、[セクション13.1.20.2「CREATE TEMPORARY TABLE ステートメント」](#)を参照してください。

MySQL では、[mysql.tables_priv](#) システムテーブルにテーブル権限が格納されます。

カラム権限

カラム権限は、特定のテーブル内の単一カラムに適用されます。カラムレベルで付与される各権限のあとに、括弧で囲まれた 1 つまたは複数のカラムを指定する必要があります。

```
GRANT SELECT (col1), INSERT (col1, col2) ON mydb.mytbl TO 'someuser'@'somehost';
```

カラムに許可される [priv_type](#) 値 (つまり、[column_list](#) 句を使用する場合は、[INSERT](#)、[REFERENCES](#)、[SELECT](#) および [UPDATE](#) です。

MySQL では、[mysql.columns_priv](#) システムテーブルにカラム権限が格納されます。

ストアドルーチン権限

[ALTER ROUTINE](#)、[CREATE ROUTINE](#)、[EXECUTE](#)、および [GRANT OPTION](#) 権限は、ストアドルーチン (プロシージャおよびファンクション) に適用されます。これらの権限は、グローバルおよびデータベースレベルで付与できます。[CREATE ROUTINE](#) を除き、これらの権限は、個々のルーチンに対してルーチンレベルで付与できます。

```
GRANT CREATE ROUTINE ON mydb.* TO 'someuser'@'somehost';  
GRANT EXECUTE ON PROCEDURE mydb.myproc TO 'someuser'@'somehost';
```

ルーチンレベルで許可される [priv_type](#) 値は、[ALTER ROUTINE](#)、[EXECUTE](#)、および [GRANT OPTION](#) です。ルーチンを最初に作成するには、グローバルレベルまたはデータベースレベルの権限が必要であるため、[CREATE ROUTINE](#) はルーチンレベルの権限ではありません。

MySQL では、[mysql.procs_priv](#) システムテーブルにルーチンレベルの権限が格納されます。

プロキシユーザー権限

[PROXY](#) 権限は、あるユーザーを別のユーザーのプロキシにできるようにします。プロキシユーザーは、プロキシユーザーのアイデンティティを偽装または取得します。つまり、プロキシユーザーの権限を引き継ぎます。

```
GRANT PROXY ON 'localuser'@'localhost' TO 'externaluser'@'somehost';
```

[PROXY](#) が付与されている場合、[GRANT](#) ステートメントで指定されている唯一の権限である必要があり、許可されている [WITH](#) オプションは [WITH GRANT OPTION](#) のみです。

プロキシ設定では、プロキシユーザーが、接続時にプロキシ設定されたユーザーの名前をサーバーに返すプラグイン経由で認証すること、およびプロキシユーザーがプロキシ設定されたユーザーに対する [PROXY](#) 権限を持っていることが必要です。詳細および例については、[セクション6.2.18「プロキシユーザー」](#)を参照してください。

MySQL では、[mysql.proxies_priv](#) システムテーブルにプロキシ権限が格納されます。

ロールの付与

[ON](#) 句のない [GRANT](#) 構文では、個々の権限ではなくロールが付与されます。ロールは、権限の名前付きコレクションです。[セクション6.2.10「ロールの使用」](#)を参照してください。例:

```
GRANT 'role1', 'role2' TO 'user1'@'localhost', 'user2'@'localhost';
```

付与する各ロールと、そのロールが付与される各ユーザーアカウントまたはロールが存在する必要があります。MySQL 8.0.16 では、匿名ユーザーにロールを付与できません。

ロールを付与しても、ロールは自動的にアクティブになりません。ロールのアクティブ化および非アクティブ化の詳細は、[ロールのアクティブ化](#)を参照してください。

ロールを付与するには、次の権限が必要です:

- **ROLE_ADMIN** 権限 (または非推奨の **SUPER** 権限) を持っている場合は、ユーザーまたはロールに対して任意のロールを付与または取り消すことができます。
- **WITH ADMIN OPTION** 句を含む **GRANT** ステートメントを使用してロールを付与された場合、そのロールを他のユーザーまたはロールに付与したり、他のユーザーまたはロールから取り消すことができます。ただし、後で付与または取消しを行うときにロールがアクティブになっている必要があります。これには、**WITH ADMIN OPTION** 自体を使用する機能が含まれます。
- **SYSTEM_USER** 権限を持つロールを付与するには、**SYSTEM_USER** 権限が必要です。

GRANT を使用して循環参照を作成できます。例:

```
CREATE USER 'u1', 'u2';
CREATE ROLE 'r1', 'r2';

GRANT 'u1' TO 'u1'; -- simple loop: u1 => u1
GRANT 'r1' TO 'r1'; -- simple loop: r1 => r1

GRANT 'r2' TO 'u2';
GRANT 'u2' TO 'r2'; -- mixed user/role loop: u2 => r2 => u2
```

循環付与参照は許可されますが、ユーザーまたはロールにはすでに権限およびロールがあるため、権限受領者に新しい権限またはロールは追加されません。

AS 句および権限の制限事項

MySQL 8.0.16 の時点で、**GRANT** には、ステートメントの実行に使用する権限コンテキストに関する追加情報を指定する **AS user [WITH ROLE]** 句があります。この構文は SQL レベルで表示できますが、主な目的は、部分的な取消しによって課される権限付与者権制限のすべてのノード間で均一なレプリケーションを有効にすることです。これにより、これらの制限がバイナリログに表示されます。部分取消しの詳細は、[セクション6.2.12「部分取消しを使用した権限の制限」](#) を参照してください。

AS user 句が指定されている場合、ステートメントの実行では、**WITH ROLE** によって指定されたすべてのロール (存在する場合) を含む、指定されたユーザーに関連付けられた権限の制限が考慮されます。その結果、ステートメントによって実際に付与される権限は、指定された権限と比較して削減される可能性があります。

AS user 句には、次の条件が適用されます:

- **AS** は、指定された **user** に権限制限がある場合にのみ有効です (**partial_revokes** システム変数が有効になっていることを意味します)。
- **WITH ROLE** が指定されている場合は、指定されたすべてのロールを指定された **user** に付与する必要があります。
- 名前付き **user** は、'**user_name**'@'**host_name**'、**CURRENT_USER** または **CURRENT_USER()** として指定された MySQL アカウントである必要があります。現在のセッション内でアクティブなロールとは異なるロールのセットを適用して **GRANT** を実行する場合は、現在のユーザーに **WITH ROLE** との名前を付けることができます。
- **AS** を使用して、**GRANT** ステートメントを実行するユーザーが所有していない権限を取得することはできません。実行中のユーザーには少なくとも付与される権限が必要ですが、**AS** 句では付与される権限のみを制限でき、エスカレートはできません。
- 付与される権限に関して、**AS** では、**GRANT** ステートメントを実行するユーザーよりも多くの権限 (制限が少ない) を持つユーザー/ロールの組合せを指定できません。**AS** ユーザー/ロールの組合せは、実行中のユーザーよりも多くの権限を持つことが許可されていますが、これらの追加の権限がステートメントで付与されていない場合のみです。
- **AS** は、グローバル権限 (**ON *.***) の付与でのみサポートされています。
- **AS** は、**PROXY** 権限付与ではサポートされていません。

次の例に、**AS** 句の影響を示します。いくつかのグローバル権限およびそれらの権限の制限を持つユーザー **u1** を作成します:

```
CREATE USER u1;
```



```
GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO u1;
REVOKE INSERT, UPDATE ON schema1.* FROM u1;
REVOKE SELECT ON schema2.* FROM u1;
```

また、権限制限の一部を引き上げてロールを `u1` に付与するロール `r1` を作成します:

```
CREATE ROLE r1;
GRANT INSERT ON schema1.* TO r1;
GRANT SELECT ON schema2.* TO r1;
GRANT r1 TO u1;
```

権限制限のないアカウントを使用して、複数のユーザーに同じグローバル権限のセットを付与しますが、それぞれに `AS` 句によって課される異なる制限があり、実際に付与されている権限を確認します。

- ここでの `GRANT` ステートメントには `AS` 句がないため、付与される権限は指定したものとまったく同じです:

```
mysql> CREATE USER u2;
mysql> GRANT SELECT, INSERT, UPDATE ON *.* TO u2;
mysql> SHOW GRANTS FOR u2;
+-----+
| Grants for u2@%                |
+-----+
| GRANT SELECT, INSERT, UPDATE ON *.* TO `u2`@`%` |
+-----+
```

- ここでの `GRANT` ステートメントには `AS` 句があるため、付与される権限は指定された権限ですが、`u1` からの制限が適用されます:

```
mysql> CREATE USER u3;
mysql> GRANT SELECT, INSERT, UPDATE ON *.* TO u3 AS u1;
mysql> SHOW GRANTS FOR u3;
+-----+
| Grants for u3@%                |
+-----+
| GRANT SELECT, INSERT, UPDATE ON *.* TO `u3`@`%` |
| REVOKE INSERT, UPDATE ON `schema1`.* FROM `u3`@`%` |
| REVOKE SELECT ON `schema2`.* FROM `u3`@`%`      |
+-----+
```

前述のように、`AS` 句で追加できるのは権限制限のみで、権限をエスカレートすることはできません。したがって、`u1` には `DELETE` 権限がありますが、このステートメントでは `DELETE` の付与が指定されていないため、付与された権限には含まれません。

- ここで `GRANT` ステートメントの `AS` 句を使用すると、`u1` に対して `r1` ロールがアクティブになります。そのロールにより、`u1` の制限の一部が解除されます。したがって、付与される権限にはいくつかの制限がありますが、前の `GRANT` ステートメントと同じ数ではありません:

```
mysql> CREATE USER u4;
mysql> GRANT SELECT, INSERT, UPDATE ON *.* TO u4 AS u1 WITH ROLE r1;
mysql> SHOW GRANTS FOR u4;
+-----+
| Grants for u4@%                |
+-----+
| GRANT SELECT, INSERT, UPDATE ON *.* TO `u4`@`%` |
| REVOKE UPDATE ON `schema1`.* FROM `u4`@`%`      |
+-----+
```

`GRANT` ステートメントに `AS user` 句が含まれている場合、そのステートメントを実行するユーザーに対する権限制限は無視されます (`AS` 句がない場合と同様に適用されません)。

その他のアカウント特性

オプションの `WITH` 句を使用すると、ユーザーは他のユーザーに権限を付与できます。 `WITH GRANT OPTION` 句は、ユーザーが、そのユーザーの持つ指定された権限レベルにある任意の権限をほかのユーザーに与えることができますようにします。

他の方法で権限を変更せずに `GRANT OPTION` 権限をアカウントに付与するには、次の手順を実行します:

```
GRANT USAGE ON *.* TO 'someuser'@'somehost' WITH GRANT OPTION;
```

異なる権限を持つ 2 人のユーザーが権限を組み合わせることができる可能性があるため、**GRANT OPTION** 権限を付与するユーザーには注意してください。

自分が保持していない権限を別のユーザーに付与することはできません。**GRANT OPTION** 権限を使用して割り当てることができるのは、自分が保持している権限だけです。

あるユーザーに特定の権限レベルにある **GRANT OPTION** 権限を付与すると、そのユーザーがそのレベルに保持している (または、将来与えられる可能性のある) すべての権限も、そのユーザーからほかのユーザーに付与される場合があることに注意してください。あるユーザーに、データベースに対する **INSERT** 権限を付与するとします。次に、そのデータベースに対する **SELECT** 権限を付与し、**WITH GRANT OPTION** を指定した場合、そのユーザーはほかのユーザーに **SELECT** 権限だけでなく、**INSERT** 権限も与えることができます。そのあと、そのユーザーにデータベースに対する **UPDATE** 権限を付与すると、そのユーザーは **INSERT**、**SELECT**、および **UPDATE** を付与できます。

非管理ユーザーの場合は、**ALTER** 権限をグローバルまたは **mysql** システムスキーマに付与しないでください。それを行うと、そのユーザーはテーブルの名前を変更することによって権限システムの破壊を試みることができます。

特定の権限に関連付けられたセキュリティリスクの詳細は、[セクション6.2.2「MySQL で提供される権限」](#)を参照してください。

MySQL バージョンと標準 SQL バージョンの GRANT

MySQL バージョンと標準 SQL バージョンの **GRANT** の最大の違いは次のとおりです。

- MySQL は、権限をユーザー名だけでなく、ホスト名とユーザー名の組み合わせに関連付けます。
- 標準 SQL はグローバルまたはデータベースレベルの権限を持たず、また MySQL がサポートするすべての権限タイプをサポートしているわけでもありません。
- MySQL は、標準 SQL の **UNDER** 権限をサポートしていません。
- 標準 SQL の権限は、階層的な方法で構造化されています。ユーザーを削除した場合は、そのユーザーに付与されていたすべての権限が取り消されます。これはまた、**DROP USER** を使用した場合の MySQL にも当てはまります。[セクション13.7.1.5「DROP USER ステートメント」](#)を参照してください。
- 標準 SQL では、テーブルを削除すると、そのテーブルに対するすべての権限が取り消されます。標準 SQL では、権限を取り消すと、その権限に基づいて付与されていたすべての権限も取り消されます。MySQL では、**DROP USER** ステートメントまたは **REVOKE** ステートメントを使用して権限を削除できます。
- MySQL では、テーブル内の一部の列に対してのみ **INSERT** 権限を持つことができます。この場合、**INSERT** 権限を持っている列の値のみを挿入するのであれば、そのテーブルに対して引き続き **INSERT** ステートメントを実行できます。厳密な SQL モードが有効になっていない場合、省略された列はその暗黙のデフォルト値に設定されます。厳密モードでは、省略された列のいずれかにデフォルト値がない場合、このステートメントは拒否されます。(標準 SQL では、すべての列に対する **INSERT** 権限が必要です。) 厳密な SQL モードおよび暗黙的なデフォルト値の詳細は、[セクション5.1.11「サーバー SQL モード」](#) および [セクション11.6「データ型デフォルト値」](#)を参照してください。

13.7.1.7 RENAME USER ステートメント

```
RENAME USER old_user TO new_user  
[, old_user TO new_user] ...
```

RENAME USER ステートメントは、既存の MySQL アカウントの名前を変更します。存在しない古いアカウント、またはすでに存在する新しいアカウントに対しては、エラーが発生します。

RENAME USER を使用するには、**mysql** システムスキーマに対するグローバル **CREATE USER** 権限または **UPDATE** 権限が必要です。**read_only** システム変数が有効になっている場合、**RENAME USER** にはさらに **CONNECTION_ADMIN** 権限 (または非推奨の **SUPER** 権限) が必要です。

MySQL 8.0.22 では、名前を変更するアカウントがストアオブジェクトの **DEFINER** 属性として指定されている場合、**RENAME USER** はエラーで失敗します。(つまり、アカウントの名前を変更すると、格納されているオブジェクトが孤立する場合、ステートメントは失敗します。) 操作を実行するには、**SET_USER_ID** 権限が必要です。この場合、ステートメントはエラーで失敗するのではなく、警告付きで成功します。特定のアカウントを **DEFINER** 属性として指定するオブジェクトの識別方法などの追加情報は、[孤立したストアオブジェクト](#)を参照してください。

各アカウント名には、[セクション6.2.4「アカウント名の指定」](#)で説明されている形式が使用されます。例:

```
RENAME USER 'jeffrey'@'localhost' TO 'jeff'@'127.0.0.1';
```

アカウント名のホスト名部分は、省略すると '%' にデフォルト設定されます。

`RENAME USER` により、古いユーザーによって保持されていた権限は新しいユーザーによって保持される権限になります。ただし、`RENAME USER` は、古いユーザーが作成したどのデータベースまたはそれらのデータベース内のどのオブジェクトも自動的に削除したり、無効にしたりしません。これには、`DEFINER` 属性に古いユーザーが指定されているストアドプログラムまたはビューが含まれます。このようなオブジェクトにアクセスしようとすると、それが定義者のセキュリティーコンテキストで実行された場合は、エラーが生成される可能性があります。(セキュリティーコンテキストについては、[セクション25.6「ストアドオブジェクトのアクセス制御」](#)を参照してください。)

権限の変更は、[セクション6.2.13「権限変更が有効化される時期」](#)に示されているように有効になります。

13.7.1.8 REVOKE ステートメント

```
REVOKE
  priv_type [(column_list)]
  [, priv_type [(column_list)] ...]
  ON [object_type] priv_level
  FROM user_or_role [, user_or_role] ...

REVOKE ALL [PRIVILEGES], GRANT OPTION
  FROM user_or_role [, user_or_role] ...

REVOKE PROXY ON user_or_role
  FROM user_or_role [, user_or_role] ...

REVOKE role [, role] ...
  FROM user_or_role [, user_or_role] ...

user_or_role: {
  user (see セクション6.2.4「アカウント名の指定」)
  | role (see セクション6.2.5「ロール名の指定」)
}
```

`REVOKE` ステートメントを使用すると、システム管理者は、ユーザーアカウントおよびロールから取り消すことができる権限およびロールを取り消すことができます。

権限が存在するレベル、許可される `priv_type`、`priv_level` および `object_type` の値、およびユーザーとパスワードを指定する構文の詳細は、[セクション13.7.1.6「GRANT ステートメント」](#)を参照してください。

ロールの詳細は、[セクション6.2.10「ロールの使用」](#)を参照してください。

`read_only` システム変数が有効になっている場合、`REVOKE` には、次の説明で説明する他の必要な権限に加えて、`CONNECTION_ADMIN` または権限 (または非推奨の `SUPER` 権限) が必要です。

`REVOKE` は、指定されたすべてのユーザーおよびロールに対して成功するか、ロールバックされ、エラーが発生しても効果はありません。ステートメントは、指定されたすべてのユーザーおよび役割で成功した場合にのみバイナリログに書き込まれます。

各アカウント名には、[セクション6.2.4「アカウント名の指定」](#)で説明されている形式が使用されます。各ロール名は、[セクション6.2.5「ロール名の指定」](#)で説明されている形式を使用します。例:

```
REVOKE INSERT ON *.* FROM 'jeffrey'@'localhost';
REVOKE 'role1', 'role2' FROM 'user1'@'localhost', 'user2'@'localhost';
REVOKE SELECT ON world.* FROM 'role3';
```

アカウント名またはロール名のホスト名部分は、省略すると '%' にデフォルト設定されます。

最初の `REVOKE` 構文を使用するには、`GRANT OPTION` 権限が必要であり、かつ取り消そうとしている権限を持っている必要があります。

すべての権限を取り消すには、2番目の構文を使用します。この構文では、指定したユーザーまたはロールのすべてのグローバル、データベース、テーブル、カラムおよびルーチン権限が削除されます:

```
REVOKE ALL PRIVILEGES, GRANT OPTION  
FROM user_or_role [, user_or_role] ...
```

REVOKE ALL PRIVILEGES, GRANT OPTION はロールを取り消しません。

この REVOKE 構文を使用するには、mysql システムスキーマに対するグローバル CREATE USER 権限または UPDATE 権限が必要です。

REVOKE キーワードの後に 1 つ以上のロール名が続く構文では、ロールを取り消す 1 つ以上のユーザーまたはロールを示す FROM 句を使用します。

mandatory_roles システム変数値で指定されたロールは取り消すことができません。

取り消されたロールは、取り消されたすべてのユーザーアカウントに即時に影響し、アカウントの現在のセッション内で、次に実行されるステートメントの権限が調整されます。

ロールを取り消すと、ロール自体が取り消され、ロールが表す権限は取り消されません。アカウントに特定の権限を含むロールが付与され、その権限が明示的に付与されているか、その権限を含む別のロールが付与されているとします。この場合、最初のロールが取り消されても、アカウントにはその権限があります。たとえば、アカウントにそれぞれ SELECT を含む 2 つのロールが付与されている場合、そのアカウントはいずれかのロールが取り消された後も選択できます。

REVOKE ALL ON *.* (グローバルレベル) は、付与されたすべての静的グローバル権限および付与されたすべての動的権限を取り消します。

権限およびロールを取り消すユーザーアカウントおよびロールは存在する必要がありますが、取り消す権限およびロールは現在付与されている必要はありません。

サーバーに付与されているが既知ではない取り消された権限は、警告付きで取り消されます。この状況は、動的権限に対して発生する可能性があります。たとえば、動的権限は、それを登録するコンポーネントがインストールされている間に付与できますが、その後そのコンポーネントがアンインストールされると、権限を所有しているアカウントはまだその権限を所有しており、取り消すことができます。

REVOKE では権限は削除されますが、mysql.user システムテーブルから行は削除されません。ユーザーアカウント全体を削除するには、DROP USER を使用します。セクション13.7.1.5「DROP USER ステートメント」を参照してください。

付与テーブルに、大文字と小文字が混在したデータベースまたはテーブル名を含む権限行が保持されており、かつ lower_case_table_names システム変数が 0 以外の値に設定されている場合は、REVOKE を使用してこれらの権限を取り消すことはできません。このような場合は、付与テーブルを直接操作する必要があります。(lower_case_table_names が設定されている場合、GRANT はこのような行を作成しませんが、変数を設定する前にこのような行が作成されている可能性があります。lower_case_table_names 設定は、サーバーの初期化時にもみ構成できます。)

mysql プログラムから正常に実行された場合、REVOKE は Query OK, 0 rows affected で応答します。操作後に残っている権限を確認するには、SHOW GRANTS を使用します。セクション13.7.7.21「SHOW GRANTS ステートメント」を参照してください。

13.7.1.9 SET DEFAULT ROLE ステートメント

```
SET DEFAULT ROLE  
{NONE | ALL | role [, role] ...}  
TO user [, user] ...
```

このステートメントは、TO キーワードの直後に指定された user ごとに、ユーザーがサーバーに接続して認証したとき、またはユーザーがセッション中に SET ROLE DEFAULT ステートメントを実行したときにアクティブになる役割を定義します。

SET DEFAULT ROLE は、ALTER USER ... DEFAULT ROLE の代替構文です (セクション13.7.1.1「ALTER USER ステートメント」を参照)。ただし、ALTER USER では単一のユーザーに対してのみデフォルトを設定できますが、SET DEFAULT ROLE では複数のユーザーに対してデフォルトを設定できます。一方、ALTER USER ステートメントのユーザー名として CURRENT_USER を指定できますが、SET DEFAULT ROLE のユーザー名は指定できません。

SET DEFAULT ROLE には、次の権限が必要です:

- 別のユーザーのデフォルトロールを設定するには、`mysql.default_roles` システムテーブルに対するグローバル `CREATE USER` 権限または `UPDATE` 権限が必要です。
- 自分のデフォルトロールを設定する場合、デフォルトとして必要なロールが付与されているかぎり、特別な権限は必要ありません。

各ロール名は、[セクション6.2.5「ロール名の指定」](#) で説明されている形式を使用します。例:

```
SET DEFAULT ROLE administrator, developer TO 'joe'@'10.0.0.1';
```

ロール名のホスト名部分は、省略すると '%' にデフォルト設定されます。

DEFAULT ROLE キーワードに続く句では、次の値が許可されます:

- `NONE`: デフォルトを `NONE` (ロールなし) に設定します。
- `ALL`: アカウントに付与されているすべてのロールにデフォルトを設定します。
- `role [, role] ...`: デフォルトを名前付きロールに設定します。このロールは、`SET DEFAULT ROLE` の実行時に存在し、アカウントに付与される必要があります。

注記

`SET DEFAULT ROLE` と `SET ROLE DEFAULT` は異なるステートメントです:

- `SET DEFAULT ROLE` では、アカウントセッション内でデフォルトでアクティブ化するアカウントロールを定義します。
- `SET ROLE DEFAULT` は、現在のセッション内のアクティブロールを現在のアカウントのデフォルトロールに設定します。

ロールの使用例は、[セクション6.2.10「ロールの使用」](#) を参照してください。

13.7.1.10 SET PASSWORD ステートメント

```
SET PASSWORD [FOR user] auth_option
[REPLACE 'current_auth_string']
[RETAIN CURRENT PASSWORD]

auth_option: {
  = 'auth_string'
  | TO RANDOM
}
```

`SET PASSWORD` ステートメントは、MySQL ユーザーアカウントにパスワードを割り当てます。パスワードは、ステートメントで明示的に指定するか、MySQL によってランダムに生成されます。このステートメントには、置換するアカウントの現在のパスワードを指定する `password-verification` 句と、アカウントにセカンダリパスワードがあるかどうかを管理する句を含めることもできます。'auth_string'および'current_auth_string'はそれぞれクリアテキスト(暗号化されていない)パスワードを表します。

注記

`SET PASSWORD` を使用してパスワードを割り当てるのではなく、`ALTER USER` はアカウントの変更(パスワードの割当てを含む)に優先ステートメントです。例:

```
ALTER USER user IDENTIFIED BY 'auth_string';
```

注記

ランダムパスワード生成、パスワード検証およびセカンダリパスワードの句は、資格証明をMySQLに内部的に格納する認証プラグインを使用するアカウントにのみ適用されます。MySQLの外部にある資格証明システムに対して認証を実行するプラグインを使用するア

アカウントの場合、パスワード管理もそのシステムに対して外部で処理する必要があります。内部資格証明記憶域の詳細は、[セクション6.2.15「パスワード管理」](#)を参照してください。

`REPLACE 'current_auth_string'`句はパスワード検証を実行し、MySQL 8.0.13 の時点で使用できます。指定された場合:

- `REPLACE` は、置換するアカウントの現在のパスワードをクリアテキスト (暗号化されていない) 文字列として指定します。
- 現在のパスワードを指定するためにアカウントのパスワード変更が必要な場合は、変更しようとしているユーザーが実際に現在のパスワードを知っていることを確認するために、句を指定する必要があります。
- アカウントのパスワードが変更される可能性があるが、現在のパスワードを指定する必要がない場合、句はオプションです。
- 句が指定されているが、句がオプションであっても現在のパスワードと一致しない場合、ステートメントは失敗します。
- `REPLACE` は、現在のユーザーのアカウントパスワードを変更する場合にのみ指定できます。

現在のパスワードを指定したパスワード検証の詳細は、[セクション6.2.15「パスワード管理」](#)を参照してください。

`RETAIN CURRENT PASSWORD` 句はデュアルパスワード機能を実装し、MySQL 8.0.14 の時点で使用できます。指定された場合:

- `RETAIN CURRENT PASSWORD` は、アカウントの現在のパスワードをセカンダリパスワードとして保持し、既存のセカンダリパスワードを置き換えます。新しいパスワードはプライマリパスワードになりますが、クライアントはアカウントを使用して、プライマリパスワードまたはセカンダリパスワードのいずれかを使用してサーバーに接続できます。(例外: `SET PASSWORD` ステートメントで指定された新しいパスワードが空の場合、`RETAIN CURRENT PASSWORD` が指定されていても、セカンダリパスワードも空になります。)
- プライマリパスワードが空のアカウントに `RETAIN CURRENT PASSWORD` を指定すると、ステートメントは失敗します。
- アカウントにセカンダリパスワードがあり、`RETAIN CURRENT PASSWORD` を指定せずにプライマリパスワードを変更した場合、セカンダリパスワードは変更されません。

デュアルパスワードの使用の詳細は、[セクション6.2.15「パスワード管理」](#)を参照してください。

`SET PASSWORD` では、次の `auth_option` 構文が許可されます:

- `= 'auth_string'`

アカウントに指定されたリテラルパスワードを割り当てます。

- `TO RANDOM`

MySQL によってランダムに生成されたパスワードをアカウントに割り当てます。このステートメントは、ステートメントを実行しているユーザーまたはアプリケーションが使用できるように、クリアテキストのパスワードも結果セットに返します。

ランダムに生成されるパスワードの結果セットおよび特性の詳細は、[ランダムパスワード生成](#)を参照してください。

ランダムパスワード生成は、MySQL 8.0.18 の時点で使用できます。

重要

状況によっては、`SET PASSWORD` がサーバーログ、またはクライアント側にある `~/mysql_history` などの履歴ファイル内に記録されることがあります。つまり、平文のパスワードが、その情報に対する読み取りアクセス権を持つ任意のユーザーによって読み取られる可能性があります。これがサーバーログで発生する条件およびこれを制御する方法については、[セクション6.1.2.3「パスワードおよびロギング」](#)を参照してください。クライアント

ト側のログインに関する同様の情報については、[セクション4.5.1.3「mysql クライアントログイン」](#)を参照してください。

`SET PASSWORD` は、ユーザーアカウントを明示的に指定する `FOR` 句の有無にかかわらず使用できます：

- このステートメントでは、`FOR user` 句を使用して、存在する必要がある指定アカウントのパスワードを設定します：

```
SET PASSWORD FOR 'jeffrey'@'localhost' = 'auth_string';
```

- `FOR user` 句を指定しない場合、ステートメントは現在のユーザーのパスワードを設定します：

```
SET PASSWORD = 'auth_string';
```

匿名以外のアカウントを使用してサーバーに接続したクライアントはすべて、そのアカウントのパスワードを変更できます。(特に、自分のパスワードを変更できます。)サーバーが認証したアカウントを確認するには、`CURRENT_USER()` 関数を呼び出します：

```
SELECT CURRENT_USER();
```

`FOR user` 句が指定されている場合、このアカウント名には、[セクション6.2.4「アカウント名の指定」](#)で説明されている形式が使用されます。例：

```
SET PASSWORD FOR 'bob'@'%.example.org' = 'auth_string';
```

アカウント名のホスト名部分は、省略すると '%' にデフォルト設定されます。

`SET PASSWORD` は、この文字列をクリアテキスト文字列として解釈し、アカウントに関連付けられた認証プラグインに渡して、プラグインによって返された結果を `mysql.user` システムテーブルのアカウント行に格納します。(プラグインには、期待される暗号化形式に値をハッシュする機会が与えられます。プラグインは、指定された値を使用できます。この場合、ハッシュは発生しません。)

(`FOR` 句を使用して) 名前付きアカウントのパスワードを設定するには、`mysql` システムスキーマに対する `UPDATE` 権限が必要です。(`FOR` 句のない匿名以外のアカウントの場合) 自分のパスワードを設定する場合、特別な権限は必要ありません。

セカンダリパスワードを変更するステートメントには、次の権限が必要です：

- 自分のアカウントに適用される `SET PASSWORD` ステートメントに `RETAIN CURRENT PASSWORD` 句を使用するには、`APPLICATION_PASSWORD_ADMIN` 権限が必要です。ほとんどのユーザーは1つのパスワードのみを必要とするため、自分のセカンダリパスワードを操作するには権限が必要です。
- アカウントがすべてのアカウントのセカンダリパスワードの操作を許可される場合は、`APPLICATION_PASSWORD_ADMIN` ではなく `CREATE USER` 権限を付与する必要があります。

`read_only` システム変数が有効になっている場合、`SET PASSWORD` には、他の必要な権限に加えて、`CONNECTION_ADMIN` 権限(または非推奨の `SUPER` 権限)が必要です。

パスワードと認証プラグインの設定の詳細は、[セクション6.2.14「アカウントパスワードの割り当て」](#) および [セクション6.2.17「プラグブル認証」](#)を参照してください。

13.7.1.11 SET ROLE ステートメント

```
SET ROLE {  
  DEFAULT  
  | NONE  
  | ALL  
  | ALL EXCEPT role [, role ] ...  
  | role [, role ] ...  
}
```

`SET ROLE` では、付与されたロールのうちアクティブなロールを指定することで、現行セッション内の現行ユーザーの有効な権限が変更されます。付与されるロールには、ユーザーに明示的に付与されるロールと、`mandatory_roles` システム変数値で指定されるロールが含まれます。

例:

```
SET ROLE DEFAULT;  
SET ROLE 'role1', 'role2';  
SET ROLE ALL;  
SET ROLE ALL EXCEPT 'role1', 'role2';
```

各ロール名は、[セクション6.2.5「ロール名の指定」](#)で説明されている形式を使用します。ロール名のホスト名部分は、省略すると '%' にデフォルト設定されます。

ユーザーに (ロールを介してではなく) 直接付与された権限は、アクティブなロールに対する変更の影響を受けません。

このステートメントは、次の役割指定子を許可します:

- **DEFAULT**: アカウントのデフォルトロールをアクティブ化します。デフォルトのロールは、[SET DEFAULT ROLE](#) で指定されたロールです。

ユーザーがサーバーに接続して正常に認証されると、サーバーはデフォルトロールとしてアクティブ化するロールを決定します。[activate_all_roles_on_login](#) システム変数が有効な場合、サーバーは付与されたすべてのロールをアクティブ化します。それ以外の場合、サーバーは [SET ROLE DEFAULT](#) を暗黙的に実行します。サーバーは、アクティブ化できるデフォルトのロールのみをアクティブ化します。サーバーは、アクティブ化できないデフォルトロールのエラーログに警告を書き込みますが、クライアントは警告を受け取りません。

ユーザーがセッション中に [SET ROLE DEFAULT](#) を実行する場合、デフォルトロールをアクティブ化できないと (存在しない場合やユーザーに付与されていない場合など)、エラーが発生します。この場合、現在アクティブなロールは変更されません。

- **NONE**: アクティブなロールを **NONE** に設定します (アクティブなロールはありません)。
- **ALL**: アカウントに付与されているすべてのロールをアクティブ化します。
- **ALL EXCEPT role [, role] ...**: 指定されたロールを除く、アカウントに付与されているすべてのロールをアクティブ化します。指定したロールが存在しているか、アカウントに付与されている必要はありません。
- **role [, role] ...**: アカウントに付与する必要がある名前付きロールをアクティブ化します。

注記

[SET DEFAULT ROLE](#) と [SET ROLE DEFAULT](#) は異なるステートメントです:

- [SET DEFAULT ROLE](#) では、アカウントセッション内でデフォルトでアクティブ化するアカウントロールを定義します。
- [SET ROLE DEFAULT](#) は、現在のセッション内のアクティブロールを現在のアカウントのデフォルトロールに設定します。

ロールの使用例は、[セクション6.2.10「ロールの使用」](#)を参照してください。

13.7.2 リソースグループ管理ステートメント

MySQL では、リソースグループの作成および管理がサポートされており、グループで使用可能なリソースに従ってスレッドが実行されるように、サーバー内で実行されているスレッドを特定のグループに割り当てることができま。このセクションでは、リソースグループ管理に使用できる SQL ステートメントについて説明します。リソースグループ機能の一般的な説明については、[セクション5.1.16「リソースグループ」](#)を参照してください。

13.7.2.1 ALTER RESOURCE GROUP ステートメント

```
ALTER RESOURCE GROUP group_name  
[VCPU [=] vcpu_spec [, vcpu_spec] ...]  
[THREAD_PRIORITY [=] N]  
[ENABLE|DISABLE [FORCE]]  
  
vcpu_spec: {N | M - N}
```

`ALTER RESOURCE GROUP` は、リソースグループの管理に使用されます (セクション5.1.16「リソースグループ」を参照)。このステートメントは、既存のリソースグループの変更可能な属性を変更します。`RESOURCE_GROUP_ADMIN` 権限が必要です。

`group_name` は、変更するリソースグループを識別します。グループが存在しない場合は、エラーが発生します。

CPU アフィニティ、優先度、およびグループが有効かどうかの属性は、`ALTER RESOURCE GROUP` で変更できます。これらの属性は、`CREATE RESOURCE GROUP` の場合と同じ方法で指定します (セクション13.7.2.2「`CREATE RESOURCE GROUP` ステートメント」を参照)。指定された属性のみが変更されます。未指定の属性は現在の値を保持します。

`FORCE` 修飾子は、`DISABLE` で使用されます。リソースグループにスレッドが割り当てられている場合のステートメントの動作を決定します:

- `FORCE` が指定されていない場合、グループ内の既存のスレッドは終了するまで実行され続けますが、新しいスレッドをグループに割り当ててはできません。
- `FORCE` が指定されている場合、グループ内の既存のスレッドはそれぞれのデフォルトグループ (`SYS_default` へのシステムスレッド、`USR_default` へのユーザースレッド) に移動されます。

`name` および `type` 属性はグループの作成時に設定され、`ALTER RESOURCE GROUP` で変更することはできません。

例:

- グループ CPU アフィニティを変更します:

```
ALTER RESOURCE GROUP rg1 VCPU = 0-63;
```

- グループスレッドの優先度を変更します:

```
ALTER RESOURCE GROUP rg2 THREAD_PRIORITY = 5;
```

- グループを無効にし、そのグループに割り当てられているスレッドをデフォルトグループに移動します:

```
ALTER RESOURCE GROUP rg3 DISABLE FORCE;
```

リソースグループ管理は、リソースグループが発生したサーバーに対してローカルです。`ALTER RESOURCE GROUP` ステートメントはバイナリログに書き込まれず、レプリケートされません。

13.7.2.2 CREATE RESOURCE GROUP ステートメント

```
CREATE RESOURCE GROUP group_name
  TYPE = {SYSTEM|USER}
  [VCPU [=] vcpu_spec [, vcpu_spec] ...]
  [THREAD_PRIORITY [=] N]
  [ENABLE|DISABLE]

vcpu_spec: {N | M - N}
```

`CREATE RESOURCE GROUP` は、リソースグループの管理に使用されます (セクション5.1.16「リソースグループ」を参照)。このステートメントは、新しいリソースグループを作成し、その初期属性値を割り当てます。`RESOURCE_GROUP_ADMIN` 権限が必要です。

`group_name` は、作成するリソースグループを識別します。グループがすでに存在する場合は、エラーが発生します。

`TYPE` 属性は必須です。システムリソースグループの場合は `SYSTEM`、ユーザーリソースグループの場合は `USER` である必要があります。グループタイプは、後で説明するように、許可される `THREAD_PRIORITY` 値に影響しません。

`VCPU` 属性は、CPU アフィニティ、つまりグループが使用できる仮想 CPU のセットを示します:

- `VCPU` が指定されていない場合、リソースグループは CPU アフィニティを持たず、使用可能なすべての CPU を使用できます。
- `VCPU` が指定されている場合、属性値はカンマ区切りの CPU 番号または範囲のリストです:

- 各数値は、0 から CPU の数 -1 までの範囲の整数である必要があります。たとえば、64 個の CPU を持つシステムでは、0 から 63 の範囲の数値を指定できます。
- 範囲は `M N` の形式で指定します。ここで、`M` は `N` 以下で、両方の数値は CPU 範囲内にあります。
- CPU 番号が許容範囲外の整数であるか、整数でない場合は、エラーが発生します。

VCPU 指定子の例 (これらはすべて同等です):

```
VCPU = 0,1,2,3,9,10
VCPU = 0-3,9-10
VCPU = 9,10,0-3
VCPU = 0,10,1,9,3,2
```

THREAD_PRIORITY 属性は、グループに割り当てられたスレッドの優先度を示します:

- THREAD_PRIORITY が指定されていない場合、デフォルトの優先度は 0 です。
- THREAD_PRIORITY が指定されている場合、属性値は -20 (最高優先度) から 19 (最低優先度) の範囲内である必要があります。システムリソースグループの優先順位は -20 から 0 の範囲内である必要があります。ユーザーリソースグループの優先順位は、0 から 19 の範囲内である必要があります。システムグループとユーザーグループに異なる範囲を使用すると、ユーザースレッドの優先度がシステムスレッドより高くなることはありません。

ENABLE および DISABLE は、リソースグループを最初に有効または無効にすることを指定します。どちらも指定しない場合、グループはデフォルトで有効になります。無効化されたグループにスレッドを割り当てることはできません。

例:

- 単一の CPU と最も低い優先度を持つ有効なユーザーグループを作成します:

```
CREATE RESOURCE GROUP rg1
TYPE = USER
VCPU = 0
THREAD_PRIORITY = 19;
```

- CPU アフィニティがなく (すべての CPU を使用できる)、優先度が最も高い無効なシステムグループを作成します:

```
CREATE RESOURCE GROUP rg2
TYPE = SYSTEM
THREAD_PRIORITY = -20
DISABLE;
```

リソースグループ管理は、リソースグループが発生したサーバーに対してローカルです。CREATE RESOURCE GROUP ステートメントはバイナリログに書き込まれず、レプリケートされません。

13.7.2.3 DROP RESOURCE GROUP ステートメント

```
DROP RESOURCE GROUP group_name [FORCE]
```

DROP RESOURCE GROUP は、リソースグループの管理に使用されます (セクション 5.1.16 「リソースグループ」を参照)。このステートメントは、リソースグループを削除します。RESOURCE_GROUP_ADMIN 権限が必要です。

`group_name` は、削除するリソースグループを識別します。グループが存在しない場合は、エラーが発生します。

FORCE 修飾子は、リソースグループにスレッドが割り当てられている場合のステートメントの動作を決定します:

- FORCE が指定されておらず、スレッドがグループに割り当てられている場合は、エラーが発生します。
- FORCE が指定されている場合、グループ内の既存のスレッドはそれぞれのデフォルトグループ (SYS_default へのシステムスレッド、USR_default へのユーザースレッド) に移動されます。

例:

- グループを削除します。グループにスレッドが含まれている場合は失敗します:

```
DROP RESOURCE GROUP rg1;
```

- グループを削除し、既存のスレッドをデフォルトグループに移動します:

```
DROP RESOURCE GROUP rg2 FORCE;
```

リソースグループ管理は、リソースグループが発生したサーバーに対してローカルです。 `DROP RESOURCE GROUP` ステートメントはバイナリログに書き込まれず、レプリケートされません。

13.7.2.4 SET RESOURCE GROUP ステートメント

```
SET RESOURCE GROUP group_name
[FOR thread_id [, thread_id] ...]
```

`SET RESOURCE GROUP` は、リソースグループの管理に使用されます ([セクション5.1.16「リソースグループ」](#) を参照)。このステートメントは、スレッドをリソースグループに割り当てます。 `RESOURCE_GROUP_ADMIN` または `RESOURCE_GROUP_USER` 権限が必要です。

`group_name` は、割り当てるリソースグループを識別します。 `thread_id` 値は、グループに割り当てるスレッドを示します。スレッド ID は、パフォーマンススキーマ `threads` テーブルから決定できます。リソースグループまたは名前付きスレッド ID が存在しない場合は、エラーが発生します。

`FOR` 句を指定しない場合、ステートメントはセッションの現在のスレッドをリソースグループに割り当てます。

スレッド ID を指定する `FOR` 句を使用して、ステートメントはこれらのスレッドをリソースグループに割り当てます。

システムスレッドをユーザーリソースグループに割り当てようとするか、ユーザースレッドをシステムリソースグループに割り当てようとするか、警告が発生します。

例:

- 現在のセッションスレッドをグループに割り当てます:

```
SET RESOURCE GROUP rg1;
```

- 名前付きスレッドをグループに割り当てます:

```
SET RESOURCE GROUP rg2 FOR 14, 78, 4;
```

リソースグループ管理は、リソースグループが発生したサーバーに対してローカルです。 `SET RESOURCE GROUP` ステートメントはバイナリログに書き込まれず、レプリケートされません。

`SET RESOURCE GROUP` のかわりに、個々のステートメントをリソースグループに割り当てる `RESOURCE_GROUP` オプティマイザヒントがあります。 [セクション8.9.3「オプティマイザヒント」](#) を参照してください。

13.7.3 テーブル保守ステートメント

13.7.3.1 ANALYZE TABLE ステートメント

```
ANALYZE [NO_WRITE_TO_BINLOG | LOCAL]
TABLE tbl_name [, tbl_name] ...

ANALYZE [NO_WRITE_TO_BINLOG | LOCAL]
TABLE tbl_name
UPDATE HISTOGRAM ON col_name [, col_name] ...
[WITH N BUCKETS]

ANALYZE [NO_WRITE_TO_BINLOG | LOCAL]
TABLE tbl_name
DROP HISTOGRAM ON col_name [, col_name] ...
```

ANALYZE TABLE では、テーブル統計が生成されます:

- どちらの **HISTOGRAM** 句も指定しない **ANALYZE TABLE** では、キー分散分析が実行され、指定したテーブルの分散が格納されます。MyISAM テーブルの場合、キー配布分析用の **ANALYZE TABLE** は `myisamchk --analyze` の使用と同等です。
- **ANALYZE TABLE** で **UPDATE HISTOGRAM** 句を使用すると、指定したテーブルのカラムのヒストグラム統計が生成され、データディクショナリに格納されます。この構文に使用できるテーブル名は 1 つだけです。
- **ANALYZE TABLE** で **DROP HISTOGRAM** 句を使用すると、指定したテーブルのカラムのヒストグラム統計がデータディクショナリから削除されます。この構文に使用できるテーブル名は 1 つだけです。

このステートメントには、このテーブルに対する **SELECT** および **INSERT** 権限が必要です。

ANALYZE TABLE は、InnoDB、NDB および MyISAM テーブルで動作します。ビューでは機能しません。

`innodb_read_only` システム変数が有効になっている場合、InnoDB を使用するデータディクショナリの統計テーブルを更新できないため、**ANALYZE TABLE** が失敗することがあります。キー分散を更新する **ANALYZE TABLE** 操作では、操作によってテーブル自体が更新された場合でも (MyISAM テーブルの場合など)、障害が発生する可能性があります。更新された分散統計を取得するには、`information_schema_stats_expiry=0` を設定します。

ANALYZE TABLE はパーティションテーブルでサポートされており、**ALTER TABLE ... ANALYZE PARTITION** を使用して 1 つ以上のパーティションを分析できます。詳細は、[セクション13.1.9「ALTER TABLE ステートメント」](#) および [セクション24.3.4「パーティションの保守」](#) を参照してください。

分析中、そのテーブルは InnoDB および MyISAM に対する読み取りロックでロックされます。

ANALYZE TABLE は、フラッシュロックを必要とするテーブル定義キャッシュからテーブルを削除します。長時間実行されているステートメントまたはトランザクションがまだテーブルを使用している場合、後続のステートメントおよびトランザクションは、フラッシュロックが解放される前にこれらの操作が終了するまで待機する必要があります。通常、**ANALYZE TABLE** 自体は迅速に終了するため、同じテーブルを含む遅延トランザクションまたは遅延ステートメントが残りのフラッシュロックによるものであることは明らかではない場合があります。

デフォルトでは、**ANALYZE TABLE** ステートメントはレプリカにレプリケートされるようにバイナリログに書き込まれます。ロギングを抑制するには、オプションの `NO_WRITE_TO_BINLOG` キーワード、またはそのエイリアス `LOCAL` を指定します。

- [ANALYZE TABLE の出力](#)
- [キー分布分析](#)
- [ヒストグラム統計分析](#)
- [その他の考慮事項](#)

ANALYZE TABLE の出力

ANALYZE TABLE は、次のテーブルに示すカラムを含む結果セットを返します。

カラム	値
Table	テーブル名
Op	analyze または histogram
Msg_type	status、error、info、note、または warning
Msg_text	情報メッセージ

キー分布分析

どちらの **HISTOGRAM** 句も指定しない **ANALYZE TABLE** では、キー分散分析が実行され、テーブルの分散が格納されます。既存のヒストグラム統計は影響を受けません。

前回のキー分散分析以降にテーブルが変更されていない場合、テーブルは再度分析されません。

MySQL では、格納されたキーの分散を使用して、定数以外の結合でテーブルを結合する順序を決定します。さらに、クエリー内の特定のテーブルにどのインデックスを使用するかを決定する場合は、キー分布を使用できます。

格納されているキー分散カーディナリティを確認するには、[SHOW INDEX](#) ステートメントまたは [INFORMATION_SCHEMA STATISTICS](#) テーブルを使用します。 [セクション13.7.7.22「SHOW INDEX ステートメント」](#) および [セクション26.34「INFORMATION_SCHEMA STATISTICS テーブル」](#) を参照してください。

InnoDB テーブルの場合、[ANALYZE TABLE](#) は、各インデックスツリーでランダムな除算を実行し、それに応じてインデックスカーディナリティの見積りを更新することで、インデックスカーディナリティを決定します。これらは単なる見積もりであるため、[ANALYZE TABLE](#) を繰り返し実行すると、別の数値が生成される可能性があります。これによって [ANALYZE TABLE](#) の InnoDB テーブル上での速度は速くなりますが、すべての行が考慮されているわけではないため、100% 正確とは言えません。

[セクション15.8.10.1「永続的オプティマイザ統計のパラメータの構成」](#) で説明されているように、[innodb_stats_persistent](#) を有効にすることで、[ANALYZE TABLE](#) によって収集された [statistics](#) をより正確かつ安定させることができます。 [innodb_stats_persistent](#) が有効になっている場合、統計は定期的に再計算されないため（サーバーの再起動後など）、インデックスカラムデータに対する大きな変更後に [ANALYZE TABLE](#) を実行することが重要です。

[innodb_stats_persistent](#) が有効になっている場合は、[innodb_stats_persistent_sample_pages](#) システム変数を変更することで、ランダムな除算の数を変更できます。 [innodb_stats_persistent](#) が無効になっている場合は、かわりに [innodb_stats_transient_sample_pages](#) を変更します。

InnoDB でのキー配布分析の詳細は、[セクション15.8.10.1「永続的オプティマイザ統計のパラメータの構成」](#) および [セクション15.8.10.3「InnoDB テーブルに対する ANALYZE TABLE の複雑さの推定」](#) を参照してください。

MySQL では、結合の最適化にインデックスカーディナリティの見積りが使用されます。結合が適切な方法で最適化されていない場合は、[ANALYZE TABLE](#) を実行してみてください。 [ANALYZE TABLE](#) では特定のテーブルに十分な値が生成されない場合は、特定のインデックスの使用を強制するクエリーとともに [FORCE INDEX](#) を使用するが、または MySQL でテーブルスキャンよりもインデックス検索が優先されるように [max_seeks_for_key](#) システム変数を設定してください。 [セクションB.3.5「オプティマイザ関連の問題」](#) を参照してください。

ヒストグラム統計分析

[ANALYZE TABLE](#) で [HISTOGRAM](#) 句を使用すると、テーブルのカラム値のヒストグラム統計を管理できます。ヒストグラム統計の詳細は、[セクション8.9.6「オプティマイザ統計」](#) を参照してください。

次のヒストグラム操作を使用できます：

- [ANALYZE TABLE](#) で [UPDATE HISTOGRAM](#) 句を使用すると、指定したテーブルのカラムのヒストグラム統計が生成され、データディクショナリに格納されます。この構文に使用できるテーブル名は 1 つだけです。

オプションの [WITH N BUCKETS](#) 句では、ヒストグラムのバケット数を指定します。 [N](#) の値は、1 から 1024 の範囲の整数である必要があります。この句を省略すると、バケットの数は 100 になります。

- [ANALYZE TABLE](#) で [DROP HISTOGRAM](#) 句を使用すると、指定したテーブルのカラムのヒストグラム統計がデータディクショナリから削除されます。この構文に使用できるテーブル名は 1 つだけです。

ストアドヒストグラム管理ステートメントは、名前付きのカラムにのみ影響します。これらのステートメントを考慮してください。

```
ANALYZE TABLE t UPDATE HISTOGRAM ON c1, c2, c3 WITH 10 BUCKETS;  
ANALYZE TABLE t UPDATE HISTOGRAM ON c1, c3 WITH 10 BUCKETS;  
ANALYZE TABLE t DROP HISTOGRAM ON c2;
```

最初のステートメントは、[c1](#)、[c2](#) および [c3](#) カラムのヒストグラムを更新し、それらのカラムの既存のヒストグラムを置き換えます。2 番目のステートメントは、[c1](#) および [c3](#) のヒストグラムを更新し、[c2](#) ヒストグラムは影響を受けません。3 番目のステートメントは、[c2](#) のヒストグラムを削除し、[c1](#) および [c3](#) のヒストグラムは影響を受けません。

ヒストグラム生成は、暗号化されたテーブル (統計内のデータの公開を回避するため) または **TEMPORARY** テーブルではサポートされていません。

ヒストグラム生成は、ジオメトリタイプ (空間データ) および **JSON** を除くすべてのデータ型のカラムに適用されません。

ヒストグラムは、格納されたカラムおよび仮想生成されたカラムに対して生成できます。

ヒストグラムは、単一カラムの一意インデックスでカバーされるカラムに対しては生成できません。

ヒストグラム管理ステートメントは、リクエストされた操作をできるだけ多く実行しようとし、残りの診断メッセージをレポートします。たとえば、**UPDATE HISTOGRAM** ステートメントで複数のカラムを指定したが、その一部が存在しないか、サポートされていないデータ型を持つ場合、他のカラムに対してヒストグラムが生成され、無効なカラムに対してメッセージが生成されます。

ヒストグラムは、次の DDL ステートメントの影響を受けます:

- **DROP TABLE** では、削除されたテーブルのカラムのヒストグラムが削除されます。
- **DROP DATABASE** では、データベース内のすべてのテーブルが削除されるため、ステートメントは削除されたデータベース内のすべてのテーブルのヒストグラムを削除します。
- **RENAME TABLE** ではヒストグラムは削除されません。かわりに、名前を変更したテーブルが新しいテーブル名に関連付けられるようにヒストグラムの名前を変更します。
- カラムを削除または変更する **ALTER TABLE** ステートメントは、そのカラムのヒストグラムを削除します。
- **ALTER TABLE ... CONVERT TO CHARACTER SET** では、文字セットの変更の影響を受けるため、文字カラムのヒストグラムは削除されます。非文字カラムのヒストグラムは影響を受けません。

histogram_generation_max_mem_size システム変数は、ヒストグラム生成に使用できるメモリの最大量を制御します。グローバル値とセッション値は、実行時に設定できます。

グローバル **histogram_generation_max_mem_size** 値を変更するには、グローバルシステム変数を設定するのに十分な権限が必要です。セッションの **histogram_generation_max_mem_size** 値を変更するには、制限付きセッションシステム変数を設定するのに十分な権限が必要です。セクション 5.1.9.1 「システム変数権限」を参照してください。

ヒストグラム生成のためにメモリに読み込まれるデータの推定量が **histogram_generation_max_mem_size** で定義されている制限を超える場合、MySQL はデータをすべてメモリに読み取るのではなくサンプリングします。サンプリングは、テーブル全体に均等に分散されます。MySQL では、ページレベルのサンプリング方法である **SYSTEM** サンプリングを使用します。

INFORMATION_SCHEMA.COLUMN_STATISTICS テーブルの **HISTOGRAM** カラムの **sampling-rate** 値をクエリーして、ヒストグラムを作成するためにサンプリングされたデータの割合を判断できます。 **sampling-rate** は、0.0 と 1.0 の間の数値です。値 1 は、すべてのデータが読み取られたことを意味します (サンプリングなし)。

次の例では、サンプリングを示します。この例では、データ量が **histogram_generation_max_mem_size** の制限を超えていることを確認するために、**employees** テーブルの **birth_date** カラムのヒストグラム統計を生成する前に、制限は低い値 (2000000 バイト) に設定されます。

```
mysql> SET histogram_generation_max_mem_size = 2000000;

mysql> USE employees;

mysql> ANALYZE TABLE employees UPDATE HISTOGRAM ON birth_date WITH 16 BUCKETS\G
***** 1. row *****
Table: employees.employees
Op: histogram
Msg_type: status
Msg_text: Histogram statistics created for column 'birth_date'.

mysql> SELECT HISTOGRAM->>$. "sampling-rate"
FROM INFORMATION_SCHEMA.COLUMN_STATISTICS
WHERE TABLE_NAME = "employees"
```

```

AND COLUMN_NAME = "birth_date";
+-----+
| HISTOGRAM->>$. "sampling-rate" |
+-----+
| 0.0491431208869665          |
+-----+

```

`sampling-rate` 値が 0.0491431208869665 の場合、`birth_date` カラムのデータの約 4.9% がヒストグラム統計を生成するためにメモリーに読み取られたことを意味します。

MySQL 8.0.19 の時点で、`InnoDB` ストレージエンジンは、`InnoDB` テーブルに格納されているデータに対して独自のサンプリング実装を提供します。ストレージエンジンが独自のサンプリング実装を提供しない場合に MySQL で使用されるデフォルトのサンプリング実装では、大規模なテーブルに対してコストがかかる全テーブルスキャンが必要です。`InnoDB` サンプリング実装では、全テーブルスキャンを回避することでサンプリングパフォーマンスが向上します。

`sampled_pages_read` および `sampled_pages_skipped` `INNODB_METRICS` カウンタを使用して、`InnoDB` データページのサンプリングを監視できます。(`INNODB_METRICS` カウンタの一般的な使用方法については、[セクション 26.51.22 「INFORMATION_SCHEMA INNODB_METRICS テーブル」](#) を参照してください。)

次の例は、ヒストグラム統計を生成する前にカウンタを有効にする必要があるサンプリングカウンタの使用法を示しています。

```

mysql> SET GLOBAL innodb_monitor_enable = 'sampled%';

mysql> USE employees;

mysql> ANALYZE TABLE employees UPDATE HISTOGRAM ON birth_date WITH 16 BUCKETS\G
***** 1. row *****
  Table: employees.employees
    Op: histogram
Msg_type: status
Msg_text: Histogram statistics created for column 'birth_date'.

mysql> USE INFORMATION_SCHEMA;

mysql> SELECT NAME, COUNT FROM INNODB_METRICS WHERE NAME LIKE 'sampled%\G
***** 1. row *****
NAME: sampled_pages_read
COUNT: 43
***** 2. row *****
NAME: sampled_pages_skipped
COUNT: 843

```

この式は、サンプリングカウンタデータに基づいてサンプリングレートを概算します:

```
sampling rate = sampled_page_read/(sampled_pages_read + sampled_pages_skipped)
```

サンプリングカウンタデータに基づくサンプリングレートは、`INFORMATION_SCHEMA.COLUMN_STATISTICS` テーブルの `HISTOGRAM` カラムの `sampling-rate` 値とほぼ同じです。

ヒストグラム生成のために実行されるメモリー割り当てについては、パフォーマンススキーマ `memory/sql/histograms` インストゥルメントをモニターしてください。[セクション 27.12.18.10 「メモリーサマリーテーブル」](#) を参照してください。

その他の考慮事項

`ANALYZE TABLE` は、`INFORMATION_SCHEMA.INNODB_TABLESTATS` テーブルからテーブル統計をクリアし、`STATS_INITIALIZED` カラムを `Uninitialized` に設定します。統計は、次回テーブルにアクセスしたときに再度収集されます。

13.7.3.2 CHECK TABLE ステートメント

```

CHECK TABLE tbl_name [, tbl_name] ... [option] ...

option: {

```

```

FOR UPGRADE
| QUICK
| FAST
| MEDIUM
| EXTENDED
| CHANGED
}

```

CHECK TABLE は、1 つまたは複数のテーブルをエラーがないかどうかチェックします。**CHECK TABLE** はまた、ビューをチェックして、そのビュー定義で参照されているテーブルが存在しなくなっているなどの問題がないかどうかを調べることもできます。

テーブルをチェックするには、それに対する何らかの権限が必要です。

CHECK TABLE は、InnoDB、MyISAM、ARCHIVE、および CSV テーブルに対して機能します。

InnoDB テーブルで **CHECK TABLE** を実行する前に、[InnoDB テーブルに対する CHECK TABLE の使用上のノート](#) を参照してください。

CHECK TABLE はパーティションテーブルでサポートされており、**ALTER TABLE ... CHECK PARTITION** を使用して 1 つ以上のパーティションをチェックできます。詳細は、[セクション13.1.9「ALTER TABLE ステートメント」](#) および [セクション24.3.4「パーティションの保守」](#) を参照してください。

CHECK TABLE は、インデックス付けされていない仮想生成カラムを無視します。

- [CHECK TABLE の出力](#)
- [バージョン互換性のチェック](#)
- [データ一貫性のチェック](#)
- [InnoDB テーブルに対する CHECK TABLE の使用上のノート](#)
- [MyISAM テーブルに対する CHECK TABLE の使用上のノート](#)

CHECK TABLE の出力

CHECK TABLE は、次のテーブルに示すカラムを含む結果セットを返します。

カラム	値
Table	テーブル名
Op	常に <code>check</code>
Msg_type	<code>status</code> 、 <code>error</code> 、 <code>info</code> 、 <code>note</code> 、または <code>warning</code>
Msg_text	情報メッセージ

このステートメントによって、チェックされたテーブルごとに多くの情報行が生成される場合があります。最後の行には `status` の `Msg_type` 値が含まれ、`Msg_text` は通常 `OK` になります。`Table is already up to date` は、そのテーブルのストレージエンジンがテーブルのチェックは必要ないと判断したことを示します。

バージョン互換性のチェック

FOR UPGRADE オプションは、指定されたテーブルが現在のバージョンの MySQL と互換性があるかどうかをチェックします。**FOR UPGRADE** を指定すると、サーバーは各テーブルをチェックして、テーブルの作成後にそのテーブルのいずれかのデータ型またはインデックスで互換性のない変更が発生しているかどうかを判定します。発生していない場合は、チェックが成功します。それ以外で、非互換性の可能性がある場合、サーバーはそのテーブルに対して完全なチェックを実行します (これには、ある程度時間がかかることがあります)。

データ型のストレージフォーマットが変更されたか、またはそのソート順序が変更されたために非互換性が発生する可能性があります。弊社の目的はそれらの変更を避けることですが、各リリースの間の非適合性よりもさらに深刻な問題を修正するために必要である場合もあります。

FOR UPGRADE は、次の非互換性を検出します:

- InnoDB および MyISAM テーブルの TEXT カラム内の最後の領域のインデックス順序が MySQL 4.1 と 5.0 の間で変更されました。
- 新しい DECIMAL データ型のストレージ方法が MySQL 5.0.3 と 5.0.5 の間で変更されました。
- 文字セットまたは照合順序に対して、テーブルインデックスの再構築が必要な変更が加えられる場合があります。このような変更の詳細は、[セクション2.11.4「MySQL 8.0 での変更」](#)を参照してください。テーブルの再構築の詳細は、[セクション2.11.13「テーブルまたはインデックスの再作成または修復」](#)を参照してください。
- MySQL 8.0 では、古いバージョンの MySQL で許可されている 2 桁の YEAR(2) データ型はサポートされていません。YEAR(2) カラムを含むテーブルの場合、CHECK TABLE では、2 桁の YEAR(2) カラムを 4 桁の YEAR カラムに変換する REPAIR TABLE をお勧めします。
- トリガー作成時間が保持されます。
- 5.6.4 より前の形式 (TIME、DATETIME および TIMESTAMP カラムで小数秒精度をサポートしていない) の古い時間的カラムが含まれており、avoid_temporal_upgrade システム変数が無効になっている場合、テーブルは再構築が必要であると報告されます。これは、MySQL のアップグレード手順で古い時間的カラムを含むテーブルを検出してアップグレードするのに役立ちます。avoid_temporal_upgrade が有効な場合、FOR UPGRADE はテーブルに存在する古い時間的カラムを無視するため、アップグレード手順はそれらをアップグレードしません。

このような時間的カラムを含み、再構築が必要なテーブルをチェックするには、CHECK TABLE ... FOR UPGRADE を実行する前に avoid_temporal_upgrade を無効にします。
- 非ネイティブパーティション化は MySQL 8.0 で削除されるため、非ネイティブパーティション化を使用するテーブルに対して警告が発行されます。第24章「パーティション化」を参照してください。

データ一貫性のチェック

次のテーブルに、指定できるその他のチェックオプションを示します。これらのオプションはストレージエンジンに渡されますが、これらのオプションは使用することも無視することもできます。

型	意味
QUICK	正しくないリンクをチェックするための行のスキャンを行いません。InnoDB および MyISAM テーブルとビューに適用されます。
FAST	正しく閉じられていないテーブルのみを検査します。InnoDB では無視されます。MyISAM のテーブルおよびビューにのみ適用されます。
CHANGED	最後のチェック以降に変更されたか、または正しく閉じられていないテーブルのみをチェックします。InnoDB では無視されます。MyISAM のテーブルおよびビューにのみ適用されます。
MEDIUM	削除されたリンクが有効であることを検証するために行をスキャンします。また、行のキーチェックサムも計算し、キーの計算されたチェックサムを使用してこれを検証します。InnoDB では無視されます。MyISAM のテーブルおよびビューにのみ適用されます。
EXTENDED	行ごとにすべてのキーの完全なキールックアップを実行します。これにより、テーブルの 100% の整合性が保証されますが、長い時間がかかります。InnoDB では無視されます。MyISAM のテーブルおよびビューにのみ適用されます。

チェックオプションは、次の例のように組み合わせることができます。この例では、テーブルが正しく閉じられたかどうかを判定するために、そのテーブルに対してすばやいチェックを実行します。

```
CHECK TABLE test_table FAST QUICK;
```


注記

CHECK TABLE で、「破損」または「正しく閉じられていません」としてマークされているテーブルに問題が検出されない場合、**CHECK TABLE** によってマークが削除されることがあります。

テーブルが破損している場合、データ部分ではなくインデックスに問題がある可能性があります。前のチェックタイプはすべて、インデックスを徹底的にチェックするため、ほとんどのエラーが見つかるはずですが。

正常であると想定しているテーブルをチェックするには、チェックオプションまたは **QUICK** オプションを使用しません。後者は、急いでおり、かつ **QUICK** でデータファイル内のエラーが見つからないという非常に小さなリスクを負える場合に使用するようにしてください。(ほとんどの場合、通常の使用状況では、MySQL でデータファイル内のどのようなエラーも見つかります。見つかった場合、そのテーブルは「破損している」としてマークされ、修復されるまで使用できなくなります。)

FAST および **CHANGED** は主に、テーブルを定期的にチェックするためにスクリプトから使用されます(たとえば、**cron** から実行されます)。ほとんどの場合、**FAST** は **CHANGED** より優先されます。(優先されない唯一の場合は、**MyISAM** コード内にバグが見つかったのではないかと疑われるときです。)

EXTENDED は、通常のチェックを実行した後にのみ使用されますが、MySQL が行を更新しようとするか、キーで行を検索しようすると、引き続きテーブルからエラーが発生します。通常のチェックが成功した場合、これはめったに発生しません。

CHECK TABLE ... EXTENDED を使用すると、クエリーオプティマイザによって生成される実行計画に影響する可能性があります。

CHECK TABLE によってレポートされる次のいくつかの問題は、自動的に修正できません。

- Found row where the auto_increment column has the value 0.

これは、**AUTO_INCREMENT** インデックスカラムに値 0 が含まれている行がテーブル内に存在することを示します。(**AUTO_INCREMENT** カラムが 0 である行は、**UPDATE** ステートメントを使用してそのカラムを明示的に 0 に設定することによって作成できます。)

これは、それ自体エラーではありませんが、そのテーブルをダンプしてリストアするか、またはそのテーブルに対して **ALTER TABLE** を実行しようとした場合に問題が発生する可能性があります。この場合、**AUTO_INCREMENT** カラムはその **AUTO_INCREMENT** カラムのルールに従って値を変更するため、重複キーエラーなどの問題が発生する可能性があります。

警告を取り除くには、**UPDATE** ステートメントを実行してカラムを 0 以外の値に設定します。

InnoDB テーブルに対する CHECK TABLE の使用上のノート

次の注意事項は、**InnoDB** テーブルに適用されます。

- **CHECK TABLE** で破損したページが検出された場合、サーバーはエラー伝播を防ぐために終了します (Bug #10132)。セカンダリインデックスで破損が発生しても、テーブルデータが読み取り可能な場合は、**CHECK TABLE** を実行するとサーバーが終了する可能性があります。
- **CHECK TABLE** でクラスタインデックスに破損した **DB_TRX_ID** または **DB_ROLL_PTR** フィールドが検出されると、**InnoDB** が無効な undo ログレコードにアクセスし、**MVCC** 関連のサーバーが終了する可能性があります。
- **CHECK TABLE** では、**InnoDB** のテーブルまたはインデックスでエラーが発生した場合、エラーがレポートされ、通常はインデックスがマークされ、場合によってはテーブルに破損のマークが付けられ、インデックスまたはテーブルをさらに使用できなくなります。このようなエラーには、セカンダリインデックス内のエントリの数が正しくないか、リンクが正しくないなどがあります。
- **CHECK TABLE** は、セカンダリインデックスで不正な数のエントリを検出した場合、エラーを報告しますが、サーバーの終了やファイルへのアクセスの防止は行いません。
- **CHECK TABLE** はインデックスページの構造を調査してから、各キーエントリを調査します。キーポインタをクラスタ化されたレコードに対して検証したり、**BLOB** ポインタのバスに従ったりはしません。

- InnoDB テーブルが独自の `.ibd file` に格納されている場合、`.ibd` ファイルの最初の 3 つの `pages` には、テーブルまたはインデックスデータではなくヘッダー情報が含まれます。CHECK TABLE ステートメントでは、ヘッダーデータにのみ影響する非一貫性は検出されません。InnoDB `.ibd` ファイルの内容全体を検証するには、`innochecksum` コマンドを使用します。
- 大規模な InnoDB テーブルで CHECK TABLE を実行する場合、CHECK TABLE の実行中に他のスレッドがブロックされることがあります。タイムアウトを回避するために、CHECK TABLE 操作の場合は、セマフォ待機しきい値 (600 秒) が 2 時間 (7200 秒) 延長されます。InnoDB は、240 秒以上のセマフォ待機を検出すると、エラーログへの InnoDB モニター出力の出力を開始します。ロックリクエストがセマフォ待機しきい値を超えると、InnoDB はプロセスを中断します。セマフォ待機タイムアウトの可能性を完全に回避するには、CHECK TABLE のかわりに CHECK TABLE QUICK を実行します。
- InnoDB SPATIAL インデックスの CHECK TABLE 機能には、R ツリーの妥当性チェックと、R ツリーの行数がクラスタインデックスと一致することを確認するチェックが含まれます。
- CHECK TABLE では、InnoDB でサポートされている仮想生成カラムのセカンダリインデックスがサポートされません。
- MySQL 8.0.14 では、InnoDB はクラスタ化されたパラレルインデックス読取りをサポートしているため、CHECK TABLE のパフォーマンスを向上させることができます。InnoDB は、CHECK TABLE 操作中にクラスタ化されたインデックスを 2 回読み取ります。2 番目の読取りはパラレルで実行できます。パラレルクラスタインデックス読取りを実行するには、`innodb_parallel_read_threads` セッション変数を 1 より大きい値に設定する必要があります。デフォルト値は 4 です。パラレルクラスタインデックス読取りの実行に使用されるスレッドの実際の数は、`innodb_parallel_read_threads` 設定またはスキャンするインデックスサブツリーの数 (いずれか小さい方) によって決まります。

MyISAM テーブルに対する CHECK TABLE の使用上のノート

MyISAM テーブルには、次のノートが適用されます:

- CHECK TABLE は、MyISAM テーブルの主要な統計を更新します。
- CHECK TABLE 出力で OK または `Table is already up to date` が返されない場合は、通常、テーブルの修復を実行する必要があります。セクション 7.6 「MyISAM テーブルの保守とクラッシュリカバリ」を参照してください。
- QUICK、MEDIUM または EXTENDED のいずれの CHECK TABLE オプションも指定されていない場合、動的書式 MyISAM テーブルのデフォルトのチェックタイプは MEDIUM です。これにより、そのテーブルに対して `myisamchk --medium-check tbl_name` を実行したのと同じ結果が得られます。また、CHANGED または FAST が指定されていないかぎり、静的フォーマットの MyISAM テーブルに対するデフォルトのチェックタイプも MEDIUM です。指定されている場合、デフォルトは QUICK です。CHANGED および FAST の場合、行はめったに破損しないため、行スキャンはスキップされます。

13.7.3.3 CHECKSUM TABLE ステートメント

```
CHECKSUM TABLE tbl_name [, tbl_name] ... [QUICK | EXTENDED]
```

CHECKSUM TABLE は、テーブルの内容に対するチェックサムをレポートします。このステートメントを使用すると、その内容が、バックアップ、ロールバック、またはデータを元の既知の状態に戻すことを目的としたその他の操作の前後で同じであることを検証できます。

このステートメントには、このテーブルに対する SELECT 権限が必要です。

このステートメントはビューではサポートされていません。ビューに対して CHECKSUM TABLE を実行すると、Checksum 値は常に NULL になり、警告が返されます。

存在しないテーブルに対しては、CHECKSUM TABLE は NULL を返し、警告を生成します。

チェックサム操作中、そのテーブルは InnoDB および MyISAM に対する読み取りロックでロックされます。

パフォーマンスに関する考慮事項

デフォルトでは、テーブル全体が 1 行ごとに読み取られ、チェックサムが計算されます。大きなテーブルでは長い時間がかかる可能性があるため、この操作は、状況に応じてのみ実行されます。この 1 行ごとの計算は、InnoDB や

MyISAM 以外のその他のすべてのストレージエンジン、および CHECKSUM=1 句で作成されていない MyISAM テーブルの場合に EXTENDED 句で得られるものと同じです。

CHECKSUM=1 句を使用して作成された MyISAM テーブルの場合、CHECKSUM TABLE または CHECKSUM TABLE ... QUICK は、非常に高速に返すことができる「ライブ」テーブルチェックサムを返します。テーブルがこれらのすべての条件を満たさない場合、QUICK による方法は NULL を返します。QUICK メソッドは、InnoDB テーブルではサポートされません。CHECKSUM 句の構文については、[セクション13.1.20「CREATE TABLE ステートメント」](#)を参照してください。

チェックサム値は、テーブル行フォーマットによって異なります。行フォーマットが変更された場合は、チェックサムも変更されます。たとえば、MySQL より前の MySQL 5.6.5 で TIME、DATETIME、TIMESTAMP などの時間型の記憶域形式が変更されたため、5.5 テーブルが MySQL 5.6 にアップグレードされると、チェックサム値が変更される可能性があります。

重要

2 つのテーブルのチェックサムが異なる場合は、それらのテーブルが何らかの点で異なることがほぼ確実です。ただし、CHECKSUM TABLE によって使用されるハッシュ関数は衝突がないことは保証されないため、同一でない 2 つのテーブルが同じチェックサムを生成する可能性があります。

13.7.3.4 OPTIMIZE TABLE ステートメント

```
OPTIMIZE [NO_WRITE_TO_BINLOG | LOCAL]
TABLE tbl_name [, tbl_name] ...
```

OPTIMIZE TABLE では、テーブルデータおよび関連するインデックスデータの物理記憶域が再編成され、記憶領域が削減され、テーブルへのアクセス時の I/O の効率が向上します。各テーブルに加えられる正確な変更は、そのテーブルによって使用されている[ストレージエンジン](#)によって異なります。

OPTIMIZE TABLE は、テーブルのタイプに応じて次の場合に使用します。

- `innodb_file_per_table` オプションが有効な状態で作成されたために独自の `.ibd ファイル` を含む InnoDB テーブルに対して大量の挿入、更新、または削除操作を行なったあと。テーブルとインデックスが再編成されるため、ディスク領域をオペレーティングシステムによる使用のために再利用できます。
- InnoDB テーブル内の FULLTEXT インデックスの一部であるカラムに対して大量の挿入、更新、または削除操作を行なったあと。最初に、構成オプション `innodb_optimize_fulltext_only=1` を設定します。インデックスの保守期間を妥当な時間に維持するために、検索インデックスで更新するワード数を指定する `innodb_ft_num_word_optimize` オプションを設定し、検索インデックスが完全に更新されるまで OPTIMIZE TABLE ステートメントのシーケンスを実行します。
- MyISAM または ARCHIVE テーブルの大きな部分を削除するか、あるいは可変長行を含む MyISAM または ARCHIVE テーブル (VARCHAR、VARBINARY、BLOB、または TEXT カラムを含むテーブル) に多くの変更を行なったあと。削除された行はリンクリスト内に保持され、以降の INSERT 操作は古い行の位置を再利用します。OPTIMIZE TABLE を使用すると、未使用領域を再利用したり、データファイルをデフラグしたりできます。テーブルを大幅に変更したあとは、このステートメントにより、そのテーブルを使用するステートメントのパフォーマンスを (場合によっては大幅に) 向上させることができます。

このステートメントには、このテーブルに対する SELECT および INSERT 権限が必要です。

OPTIMIZE TABLE は、InnoDB、MyISAM、および ARCHIVE テーブルに対して機能します。OPTIMIZE TABLE は、インメモリー NDB テーブルの動的なカラムに対してもサポートされます。インメモリーテーブルの固定幅カラムに対しては機能せず、「ディスクデータ」テーブルに対しても機能しません。「NDB Cluster」テーブルの OPTIMIZE のパフォーマンスは、OPTIMIZE TABLE による行のバッチ処理間の待機時間を制御する `--ndb-optimization-delay` を使用してチューニングできます。詳細は、[セクション23.1.7.11「前 NDB Cluster 8.0 で解決される NDB Cluster の問題」](#)を参照してください。

「NDB Cluster の場合」テーブルの OPTIMIZE TABLE は、OPTIMIZE 操作を実行している SQL スレッドを強制終了することで中断できます。

デフォルトでは、OPTIMIZE TABLE はその他のストレージエンジンを使用して作成されたテーブルに対しては機能せず、このサポートがないことを示す結果を返します。 `--skip-new` オプションを使用して `mysqld` を起動する

ことによって、その他のストレージエンジンに対して `OPTIMIZE TABLE` を機能させることができます。この場合、`OPTIMIZE TABLE` は単に `ALTER TABLE` にマップされます。

このステートメントはビューでは機能しません。

`OPTIMIZE TABLE` は、パーティションテーブルでサポートされています。このステートメントのパーティション化されたテーブルでの使用やテーブルパーティションについては、[セクション24.3.4「パーティションの保守」](#)を参照してください。

デフォルトでは、`OPTIMIZE TABLE` ステートメントはレプリカにレプリケートされるようにバイナリログに書き込まれます。ロギングを抑制するには、オプションの `NO_WRITE_TO_BINLOG` キーワード、またはそのエイリアス `LOCAL` を指定します。

- [OPTIMIZE TABLE の出力](#)
- [InnoDB の詳細](#)
- [MyISAM の詳細](#)
- [その他の考慮事項](#)

OPTIMIZE TABLE の出力

`OPTIMIZE TABLE` は、次のテーブルに示すカラムを含む結果セットを返します。

カラム	値
Table	テーブル名
Op	常に <code>optimize</code>
Msg_type	<code>status</code> 、 <code>error</code> 、 <code>info</code> 、 <code>note</code> 、または <code>warning</code>
Msg_text	情報メッセージ

`OPTIMIZE TABLE` テーブルは、古いファイルから新しく作成されたファイルへのテーブル統計のコピー中に発生したすべてのエラーをキャッチしてスローします。たとえば、`.MYD` または `.MYI` ファイルの所有者のユーザー ID が `mysqld` プロセスのユーザー ID と異なる場合、`OPTIMIZE TABLE` では、`root` ユーザーが `mysqld` を起動しないかぎり、「ファイルの所有権を変更できません」というエラーが生成されます。

InnoDB の詳細

InnoDB テーブルの場合、`OPTIMIZE TABLE` は `ALTER TABLE ... FORCE` にマップされます。これは、インデックス統計を更新し、クラスタ化されたインデックス内の未使用領域を解放するためにテーブルを再構築します。これは、次に示すように、InnoDB テーブルに対して実行したときに `OPTIMIZE TABLE` の出力に表示されます。

```
mysql> OPTIMIZE TABLE foo;
+-----+-----+-----+-----+
| Table | Op   | Msg_type | Msg_text |
+-----+-----+-----+-----+
| test.foo | optimize | note   | Table does not support optimize, doing recreate + analyze instead |
| test.foo | optimize | status  | OK       |
+-----+-----+-----+-----+
```

`OPTIMIZE TABLE` では、通常のパーティション化された InnoDB テーブルに `online DDL` が使用されるため、同時 DML 操作の停止時間が短縮されます。`OPTIMIZE TABLE` によってトリガーされたテーブルの再構築が適切に完了します。排他テーブルロックは、操作の準備フェーズおよびコミットフェーズでのみ短時間実行されます。準備フェーズでは、メタデータが更新され、中間テーブルが作成されます。コミットフェーズでは、テーブルメタデータの変更がコミットされます。

`OPTIMIZE TABLE` では、次の条件下でテーブルのコピー方法を使用してテーブルを再構築します:

- `old_alter_table` システム変数が有効な場合。
- `--skip-new` オプションを使用してサーバーを起動したとき。

`online DDL` を使用する `OPTIMIZE TABLE` は、`FULLTEXT` インデックスを含む InnoDB テーブルではサポートされていません。かわりにテーブルのコピー方法が使用されます。

InnoDB は、ページの割当て方法を使用してデータを格納し、レガシーストレージエンジン (MyISAM など) と同様に断片化の影響を受けません。最適化を実行するかどうかを検討する場合は、サーバーが処理すると予想されるトランザクションのワークロードを考慮してください:

- ある程度の断片化は予測されます。InnoDB は、ページを分割しなくても更新できる余地を残すために、ページを 93% までしかいっぱいにしません。
- 削除操作によってギャップが残され、ページの空きが目的より多くなる場合があります。これにより、テーブルを最適化する価値が生まれる可能性があります。
- 行を更新すると通常、十分な領域が使用可能であれば、データ型と行フォーマットに応じて同じページ内のデータが書き換えられます。セクション 15.9.1.5 「InnoDB テーブルでの圧縮の動作」およびセクション 15.10 「InnoDB の行フォーマット」を参照してください。
- InnoDB はその MVCC メカニズムのために同じデータの複数のバージョンを保持するため、並列性の高いワークロードでは、時間の経過とともにインデックス内にギャップが残される可能性があります。セクション 15.3 「InnoDB マルチバージョン」を参照してください。

MyISAM の詳細

MyISAM テーブルの場合、`OPTIMIZE TABLE` は次のように機能します。

1. テーブルが行を削除または分割した場合は、そのテーブルを修復します。
2. インデックスページがソートされていない場合は、それをソートします。
3. テーブルの統計が最新でない (そのため、インデックスのソートによって修復を実行できない) 場合は、それを更新します。

その他の考慮事項

`OPTIMIZE TABLE` は、通常のパーティション化された InnoDB テーブルに対してオンラインで実行されます。それ以外の場合は、`OPTIMIZE TABLE` の実行中に MySQL `locks the table` が実行されます。

`OPTIMIZE TABLE` は、`POINT` カラム上の空間インデックスなどの R ツリーインデックスをソートしません。(Bug #23578)

13.7.3.5 REPAIR TABLE ステートメント

```
REPAIR [NO_WRITE_TO_BINLOG | LOCAL]
TABLE tbl_name [, tbl_name] ...
[QUICK] [EXTENDED] [USE_FRM]
```

`REPAIR TABLE` は、特定のストレージエンジンに対してのみ、破損している可能性のあるテーブルを修復します。

このステートメントには、このテーブルに対する `SELECT` および `INSERT` 権限が必要です。

通常、`REPAIR TABLE` を実行する必要はありませんが、災害が発生した場合は、このステートメントを使用すると MyISAM テーブルからすべてのデータをリストアできる可能性があります。テーブルが頻繁に破損する場合は、その原因を見つけることにより、`REPAIR TABLE` を使用する必要がなくなるようにしてください。セクション B.3.3.3 「MySQL が繰り返しクラッシュする場合の対処方法」およびセクション 16.2.4 「MyISAM テーブルの問題点」を参照してください。

`REPAIR TABLE` はテーブルをチェックして、アップグレードが必要かどうかを確認します。アップグレードが必要な場合は、`CHECK TABLE ... FOR UPGRADE` と同じルールに従ってアップグレードを実行します。詳細は、セクション 13.7.3.2 「`CHECK TABLE` ステートメント」を参照してください。

重要

- テーブルの修復操作を実行する前に、そのテーブルのバックアップを作成してください。状況によっては、この操作のためにデータ損失が発生することがあります。考えられる

原因としては、ファイルシステムのエラーなどがありますがこれに限りません。第7章「バックアップとリカバリ」を参照してください。

- **REPAIR TABLE** 操作中にサーバーが終了した場合は、再起動後、テーブルに対して別の **REPAIR TABLE** ステートメントをすぐに実行してから、ほかの操作を実行することが不可欠です。最悪の場合は、データファイルに関する情報のない新しいクリーンなインデックスファイルが生成されており、実行する次の操作によってデータファイルが上書きされる可能性があります。この状況はめったに発生しませんが、最初にバックアップを作成することの価値を強調している、可能性のあるシナリオです。
 - ソース上のテーブルが破損し、そのテーブルで **REPAIR TABLE** を実行した場合、元のテーブルに対する変更はレプリカに伝播されません。
- [REPAIR TABLE ストレージエンジンおよびパーティショニングサポート](#)
 - [REPAIR TABLE のオプション](#)
 - [REPAIR TABLE の出力](#)
 - [テーブルの修復に関する考慮事項](#)

REPAIR TABLE ストレージエンジンおよびパーティショニングサポート

REPAIR TABLE は、**MyISAM**、**ARCHIVE** および **CSV** テーブルに対して機能します。**MyISAM** テーブルの場合、デフォルトでは `myisamchk --recover tbl_name` と同じ効果があります。このステートメントはビューでは機能しません。

REPAIR TABLE は、パーティション化されたテーブルに対してサポートされています。ただし、パーティション化されたテーブルに対して、このステートメントで **USE_FRM** オプションを使用することはできません。

ALTER TABLE ... REPAIR PARTITION を使用すると、1 つ以上のパーティションを修復できます。詳細は、[セクション13.1.9「ALTER TABLE ステートメント」](#) および [セクション24.3.4「パーティションの保守」](#) を参照してください。

REPAIR TABLE のオプション

- **NO_WRITE_TO_BINLOG** または **LOCAL**

デフォルトでは、**REPAIR TABLE** ステートメントはレプリカにレプリケートされるようにバイナリログに書き込まれます。ロギングを抑制するには、オプションの **NO_WRITE_TO_BINLOG** キーワード、またはそのエイリアス **LOCAL** を指定します。

- **QUICK**

QUICK オプションを使用した場合、**REPAIR TABLE** はデータファイルではなく、インデックスファイルのみを修復しようとします。このタイプの修復は、`myisamchk --recover --quick` によって実行される修復と同様です。

- **EXTENDED**

EXTENDED オプションを使用した場合、MySQL はソートしながら 1 回につき 1 つのインデックスを作成する代わりに、1 行ごとにインデックスを作成します。このタイプの修復は、`myisamchk --safe-recover` によって実行される修復と同様です。

- **USE_FRM**

USE_FRM オプションは、**.MYI** インデックスファイルがない場合や、そのヘッダーが破損している場合に使用できます。このオプションは、**.MYI** ファイルヘッダーの情報を信頼せず、データディクショナリの情報を使用して再作成するように MySQL に指示します。この種類の修復は、`myisamchk` では実行できません。

注意

通常の **REPAIR** モードを使用できない場合は、**USE_FRM** オプションのみを使用します。サーバーに **.MYI** ファイルを無視するよう指示すると、**.MYI** に格納されている重要なデー

ブルメタデータが修復プロセスから使用できなくなるため、有害な結果を招く場合があります。

- 現在の `AUTO_INCREMENT` 値は失われます。
- テーブル内の削除されたレコードへのリンクは失われます。つまり、削除されたレコードの空き領域はその後も占有されません。
- `.MYI` ヘッダーは、テーブルが圧縮されているかどうかを示します。サーバーがこの情報を無視すると、テーブルが圧縮されていることがわからないため、修復によってテーブルの内容の変更または損失が発生する場合があります。つまり、圧縮テーブルでは `USE_FRM` を使用しないようにしてください。いずれにしても、これは必須ではありません。圧縮テーブルは読み取り専用であるため、破損することはありません。

現在実行しているものとは異なるバージョンの MySQL サーバーで作成されたテーブルに `USE_FRM` を使用する場合、`REPAIR TABLE` はテーブルの修復を試行しません。この場合、`REPAIR TABLE` によって返される結果セットには、`Msg_type` 値が `error` で、`Msg_text` 値が `Failed repairing incompatible .FRM file` である行が含まれています。

`USE_FRM` が使用されている場合、`REPAIR TABLE` はテーブルをチェックして、アップグレードが必要かどうかを確認しません。

REPAIR TABLE の出力

`REPAIR TABLE` は、次のテーブルに示すカラムを含む結果セットを返します。

カラム	値
<code>Table</code>	テーブル名
<code>Op</code>	常に <code>repair</code>
<code>Msg_type</code>	<code>status</code> 、 <code>error</code> 、 <code>info</code> 、 <code>note</code> 、または <code>warning</code>
<code>Msg_text</code>	情報メッセージ

`REPAIR TABLE` ステートメントによって、修復されたテーブルごとに多数の情報行が生成される可能性があります。最後の行には `status` の `Msg_type` 値が含まれ、`Msg_text` は通常 `OK` になります。MyISAM テーブルの場合、`OK` が取得されない場合は、`myisamchk --safe-recover` を使用して修復する必要があります。(`REPAIR TABLE` は、`myisamchk` のすべてのオプションを実装しているわけではありません。 `myisamchk --safe-recover` では、`--max-record-length` など、`REPAIR TABLE` でサポートされていないオプションを使用することもできます。)

`REPAIR TABLE` テーブルは、古い破損したファイルから新しく作成されたファイルへのテーブル統計のコピー中に発生したすべてのエラーをキャッチしてスローします。たとえば、`.MYD` または `.MYI` ファイルの所有者のユーザー ID が `mysqld` プロセスのユーザー ID と異なる場合、`REPAIR TABLE` では、`root` ユーザーが `mysqld` を起動しないかぎり、「ファイルの所有権を変更できません」というエラーが生成されます。

テーブルの修復に関する考慮事項

`REPAIR TABLE` では、5.6.4 より前の形式 (`TIME`、`DATETIME` および `TIMESTAMP` カラムで小数秒精度をサポートしていない) の古い時間的カラムが含まれており、`avoid_temporal_upgrade` システム変数が無効になっている場合、テーブルがアップグレードされます。`avoid_temporal_upgrade` が有効な場合、`REPAIR TABLE` はテーブルに存在する古い時間的カラムを無視し、それらをアップグレードしません。

このような時間的カラムを含むテーブルをアップグレードするには、`REPAIR TABLE` を実行する前に `avoid_temporal_upgrade` を無効にします。

特定のシステム変数を設定することによって、`REPAIR TABLE` のパフォーマンスを向上させることができる可能性があります。セクション8.6.3「`REPAIR TABLE` ステートメントの最適化」を参照してください。

13.7.4 コンポーネント、プラグインおよびユーザー定義関数のステートメント

13.7.4.1 ユーザー定義関数用の CREATE FUNCTION ステートメント


```
CREATE [AGGREGATE] FUNCTION function_name
  RETURNS {STRING|INTEGER|REAL|DECIMAL}
  SONAME shared_library_name
```

このステートメントは、`function_name` という名前のユーザー定義関数 (UDF) をロードします。(CREATE FUNCTION は、ストアードファンクションの作成にも使用されます。セクション13.1.17「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」を参照してください。)

ユーザー定義関数は、ABS() や CONCAT() などのネイティブ (組込み) MySQL 関数のように機能する新しい関数を使用して MySQL を拡張する方法です。Adding a Loadable Function を参照してください。

`function_name` は、この関数を呼び出すために SQL ステートメントで使用される名前です。RETURNS 句は、この関数の戻り値の型を示します。DECIMAL は RETURNS のあとの正当な値ですが、現在 DECIMAL 関数は文字列値を返すため、STRING 関数のように記述してください。

AGGREGATE キーワード (指定されている場合) は、UDF が集計 (グループ) 関数であることを示します。集計 UDF は、SUM() や COUNT() などのネイティブ MySQL 集計関数とまったく同じように機能します。

`shared_library_name` は、関数を実装するコードを含む共有ライブラリファイルのベース名です。このファイルは、プラグインディレクトリに存在する必要があります。このディレクトリは、`plugin_dir` システム変数の値から取得できます。詳細は、セクション5.7.1「ユーザー定義関数のインストールおよびアンインストール」を参照してください。

CREATE FUNCTION では、関数を登録するために `mysql.func` システムテーブルに行が追加されるため、`mysql` システムスキーマに対する INSERT 権限が必要です。

CREATE FUNCTION は、インストールされている UDF に関する実行時情報を提供するパフォーマンススキーマ `user_defined_functions` テーブルにもこの関数を追加します。セクション27.12.19.12「`user_defined_functions` テーブル」を参照してください。

注記

`mysql.func` システムテーブルと同様に、パフォーマンススキーマ `user_defined_functions` テーブルには、CREATE FUNCTION を使用してインストールされた UDF が一覧表示されます。`mysql.func` テーブルとは異なり、`user_defined_functions` テーブルには、サーバーコンポーネントまたはプラグインによって自動的にインストールされる UDF もリストされます。この違いにより、どの UDF がインストールされているかを `mysql.func` より `user_defined_functions` で確認することをお勧めします。

通常の起動シーケンスでは、サーバーは `mysql.func` テーブルに登録されている UDF をロードします。`--skip-grant-tables` オプションを使用してサーバーを起動した場合、テーブルに登録されている UDF はロードされず、使用できません。

注記

UDF に関連付けられた共有ライブラリをアップグレードするには、DROP FUNCTION ステートメントを発行し、共有ライブラリをアップグレードしたあと、CREATE FUNCTION ステートメントを発行します。最初に共有ライブラリをアップグレードしてから DROP FUNCTION を使用すると、サーバーが予期せず停止する可能性があります。

13.7.4.2 ユーザー定義関数に対する DROP FUNCTION ステートメント

```
DROP FUNCTION [IF EXISTS] function_name
```

このステートメントは、`function_name` という名前のユーザー定義関数 (UDF) を削除します。(DROP FUNCTION は、ストアードファンクションの削除にも使用されます。セクション13.1.29「DROP PROCEDURE および DROP FUNCTION ステートメント」を参照してください。)

DROP FUNCTION は、CREATE FUNCTION を補完したものです。関数を登録する `mysql.func` システムテーブルから行が削除されるため、`mysql` システムスキーマに対する DELETE 権限が必要です。

DROP FUNCTION は、インストールされている UDF に関する実行時情報を提供するパフォーマンススキーマ `user_defined_functions` テーブルからも関数を削除します。セクション27.12.19.12「`user_defined_functions` テーブル」を参照してください。

通常の起動シーケンスでは、サーバーは `mysql.func` テーブルに登録されている UDF をロードします。 `DROP FUNCTION` は削除された関数の `mysql.func` 行を削除するため、サーバーはその後の再起動時に関数をロードしません。

`DROP FUNCTION` を使用して、`CREATE FUNCTION` を使用するのではなく、コンポーネントまたはプラグインによって自動的にインストールされる UDF を削除することはできません。このような UDF は、UDF をインストールしたコンポーネントまたはプラグインがアンインストールされると、自動的に削除されます。

注記

UDF に関連付けられた共有ライブラリをアップグレードするには、`DROP FUNCTION` ステートメントを発行し、共有ライブラリをアップグレードしたあと、`CREATE FUNCTION` ステートメントを発行します。最初に共有ライブラリをアップグレードしてから `DROP FUNCTION` を使用すると、サーバーが予期せず停止する可能性があります。

13.7.4.3 INSTALL COMPONENT ステートメント

```
INSTALL COMPONENT component_name [, component_name ] ...
```

このステートメントは、すぐにアクティブになる 1 つ以上のコンポーネントをインストールします。コンポーネントは、サーバーおよびその他のコンポーネントで使用可能なサービスを提供します。 [セクション 5.5 「MySQL のコンポーネント」](#) を参照してください。 `INSTALL COMPONENT` では、コンポーネントに登録するために `mysql.component` システムテーブルに行が追加されるため、そのテーブルに対する `INSERT` 権限が必要です。

例:

```
INSTALL COMPONENT 'file://component1', 'file://component2';
```

コンポーネント名は、`file://` で始まり、`plugin_dir` システム変数で指定されたディレクトリにある、コンポーネントを実装するファイルのベース名を示す URN です。コンポーネント名には、`.so` や `.dll` などのプラットフォーム依存のファイル名接尾辞は含まれません。(コンポーネント名の解釈自体がサービスによって実行され、コンポーネントインフラストラクチャによってデフォルトのサービス実装を代替実装に置き換えることができるため、これらのネーミング詳細は変更される可能性があります。)

エラーが発生した場合、ステートメントは失敗し、効果はありません。たとえば、コンポーネント名が間違っている場合、名前付きコンポーネントが存在しないかすでにインストールされている場合、またはコンポーネントの初期化が失敗した場合に発生します。

ローダーサービスはコンポーネントのロードを処理します。これには、レジストリとして機能する `mysql.component` システムテーブルへのインストール済コンポーネントの追加が含まれます。その後のサーバーの再起動では、`mysql.component` にリストされているコンポーネントは、起動シーケンス中にローダーサービスによってロードされます。これは、サーバーが `--skip-grant-tables` オプションを使用して起動された場合でも発生します。

コンポーネントがレジストリに存在しないサービスに依存しており、依存するサービスを提供するコンポーネントもインストールせずにコンポーネントをインストールしようとすると、エラーが発生します:

```
ERROR 3527 (HY000): Cannot satisfy dependency for service 'component_a'
required by component 'component_b'.
```

この問題を回避するには、すべてのコンポーネントを同じステートメントでインストールするか、依存先のコンポーネントをインストールした後に依存コンポーネントをインストールします。

13.7.4.4 INSTALL PLUGIN ステートメント

```
INSTALL PLUGIN plugin_name SONAME 'shared_library_name'
```

このステートメントは、サーバープラグインをインストールします。プラグインに登録するために `mysql.plugin` システムテーブルに行が追加されるため、このテーブルに対する `INSERT` 権限が必要です。

`plugin_name` は、ライブラリファイルに含まれているプラグインディスクリプタ構造で定義されているプラグインの名前です ([Plugin Data Structures](#) を参照してください)。プラグイン名では大文字と小文字は区別されません。プラグイン名は C ソースファイル、シェルコマンド行、M4 および Bourne シェルスクリプト、SQL 環境などで使用される

ため、最大化の互換性のために、プラグイン名は ASCII 文字、数字、およびアンダースコアに制限するようにしてください。

`shared_library_name` は、プラグインコードを含む共有ライブラリの名前です。この名前には、ファイル名拡張子が含まれています (`libmyplugin.so`、`libmyplugin.dll`、`libmyplugin.dylib` など)。

共有ライブラリは、プラグインディレクトリ (`plugin_dir` システム変数で指定されているディレクトリ) 内に存在する必要があります。このライブラリは、サブディレクトリ内ではなく、プラグインディレクトリ自体に存在する必要があります。デフォルトでは、`plugin_dir` は `pkglibdir` 構成変数で指定されているディレクトリの下にある `plugin` ディレクトリですが、サーバーの起動時に `plugin_dir` の値を設定することによって変更できます。たとえば、`my.cnf` ファイル内でその値を設定します。

```
[mysqlq]
plugin_dir=/path/to/plugin/directory
```

`plugin_dir` の値が相対パス名である場合は、MySQL ベースディレクトリ (`basedir` システム変数の値) を基準にしていると見なされます。

INSTALL PLUGIN は、プラグインを使用可能にするために、そのプラグインコードをロードして初期化します。プラグインは、使用可能になる前にそのプラグインが実行する必要があるすべての設定を処理するその初期化関数を実行することによって初期化されます。サーバーがシャットダウンすると、ロードされるプラグインごとに初期化解除関数が実行されるため、プラグインは最終的なクリーンアップを実行できます。

また、**INSTALL PLUGIN** は、プラグイン名とライブラリファイル名を示す行を `mysql.plugin` システムテーブルに追加して、プラグインを登録します。通常の起動シーケンスでは、サーバーは `mysql.plugin` に登録されているプラグインをロードして初期化します。つまり、プラグインはサーバーが起動するたびにではなく、1 回だけ **INSTALL PLUGIN** によってインストールされます。サーバーが `--skip-grant-tables` オプションで起動された場合、`mysql.plugin` テーブルに登録されているプラグインはロードされず、使用できません。

プラグインライブラリには、複数のプラグインを含めることができます。各プラグインをインストールするには、個別の **INSTALL PLUGIN** ステートメントを使用します。各ステートメントは異なるプラグインを指定しますが、そのすべてが同じライブラリ名を指定します。

INSTALL PLUGIN を指定すると、サーバーは、サーバーの起動中と同様にオプション (`my.cnf`) ファイルを読み取ります。これにより、プラグインは、これらのファイルからすべての関連オプションを取得できるようになります。プラグインをロードする前でも、オプションファイルにプラグインオプションを追加できます (`loose` プリフィクスが使用されている場合)。また、プラグインをアンインストールしたり、`my.cnf` を編集したり、プラグインを再度インストールしたりすることもできます。プラグインをこの方法で再起動すると、サーバーを再起動することなく新しいオプション値を指定できます。

サーバーの起動時に個々のプラグインロードを制御するオプションについては、[セクション5.6.1「プラグインのインストールおよびアンインストール」](#)を参照してください。サーバーにシステムテーブルを読み取らないよう指示する `--skip-grant-tables` オプションが指定されたとき、1 回のサーバー起動時にプラグインをロードする必要がある場合は、`--plugin-load` オプションを使用します。[セクション5.1.7「サーバーコマンドオプション」](#)を参照してください。

プラグインを削除するには、**UNINSTALL PLUGIN** ステートメントを使用します。

プラグインのロードについての追加情報は、[セクション5.6.1「プラグインのインストールおよびアンインストール」](#)を参照してください。

インストールされているプラグインを確認するには、**SHOW PLUGINS** ステートメントを使用するか、`INFORMATION_SCHEMA` に `PLUGINS` テーブルをクエリーします。

プラグインライブラリを再コンパイルするとき、それを再インストールする必要がある場合は、次の方法のいずれかを使用できます。

- **UNINSTALL PLUGIN** を使用してライブラリ内のすべてのプラグインをアンインストールし、新しいプラグインライブラリファイルプラグインディレクトリにインストールしてから、**INSTALL PLUGIN** を使用してすべてのプラグインをライブラリにインストールします。この手順には、サーバーを停止することなく使用できるという利点があります。ただし、プラグインライブラリに多数のプラグインが含まれている場合は、多数の **INSTALL PLUGIN** および **UNINSTALL PLUGIN** ステートメントを発行する必要があります。
- サーバーを停止し、新しいプラグインライブラリファイルプラグインディレクトリにインストールしてから、サーバーを再起動します。

13.7.4.5 UNINSTALL COMPONENT ステートメント

```
UNINSTALL COMPONENT component_name [, component_name ] ...
```

このステートメントは、1つ以上のコンポーネントを非アクティブ化およびアンインストールします。コンポーネントは、サーバーおよびその他のコンポーネントで使用可能なサービスを提供します。[セクション5.5「MySQLのコンポーネント」](#)を参照してください。`UNINSTALL COMPONENT`は、`INSTALL COMPONENT`を補完したものです。コンポーネントを登録するテーブルから行が削除されるため、`mysql.component` システムテーブルに対する `DELETE` 権限が必要です。

例:

```
UNINSTALL COMPONENT 'file://component1', 'file://component2';
```

コンポーネントのネーミングの詳細は、[セクション13.7.4.3「INSTALL COMPONENT ステートメント」](#)を参照してください。

エラーが発生した場合、ステートメントは失敗し、効果はありません。たとえば、コンポーネント名が誤っている場合、名前付きコンポーネントがインストールされていない場合、または他のインストール済コンポーネントが依存しているためアンインストールできない場合に発生します。

ローダーサービスはコンポーネントのアンロードを処理します。これには、レジストリとして機能する `mysql.component` システムテーブルからのアンインストールされたコンポーネントの削除が含まれます。その結果、アンロードされたコンポーネントは、後続のサーバー再起動のために起動シーケンス中にロードされません。

13.7.4.6 UNINSTALL PLUGIN ステートメント

```
UNINSTALL PLUGIN plugin_name
```

このステートメントは、インストールされているサーバープラグインを削除します。`UNINSTALL PLUGIN`は、`INSTALL PLUGIN`を補完したものです。プラグインを登録するテーブルから行が削除されるため、`mysql.plugin` システムテーブルに対する `DELETE` 権限が必要です。

`plugin_name` は、`mysql.plugin` テーブルにリストされている何らかのプラグインの名前である必要があります。サーバーはプラグイン初期化解除関数を実行し、プラグインの行を `mysql.plugin` システムテーブルから削除するため、その後のサーバーの再起動でプラグインがロードおよび初期化されることはありません。`UNINSTALL PLUGIN`は、プラグイン共有ライブラリファイルを削除しません。

プラグインを使用しているテーブルが開いている場合は、そのプラグインをアンインストールできません。

プラグインの削除は、関連付けられたテーブルの使用に影響を与えます。たとえば、全文パーサープラグインがテーブル上の `FULLTEXT` インデックスに関連付けられている場合は、そのプラグインをアンインストールするとそのテーブルが使用できなくなります。そのテーブルにアクセスしようとすると、エラーが発生します。そのテーブルを開くこともできないため、そのプラグインが使用されているインデックスを削除できません。つまり、テーブルの内容が必要であるかぎり、プラグインのアンインストールは慎重に行う必要があります。あとで再インストールする予定のないプラグインをアンインストールしており、テーブルの内容が必要である場合は、あとでそのテーブルをリロードできるように、そのテーブルを `mysqldump` でダンプし、ダンプされた `CREATE TABLE` ステートメントから `WITH PARSER` 句を削除するようにしてください。テーブルの内容が必要でない場合は、そのテーブルに関連付けられたいずれかのプラグインがない場合でも `DROP TABLE` を使用できます。

プラグインのロードについての追加情報は、[セクション5.6.1「プラグインのインストールおよびアンインストール」](#)を参照してください。

13.7.5 CLONE ステートメント

```
CLONE clone_action
```

```
clone_action: {  
  LOCAL DATA DIRECTORY [=] 'clone_dir';  
  | INSTANCE FROM 'user'@'host':port  
  IDENTIFIED BY 'password'  
  [DATA DIRECTORY [=] 'clone_dir']  
  [REQUIRE [NO] SSL]
```


}

CLONE ステートメントは、ローカルまたはリモートの MySQL サーバーインスタンスからデータをクローニングするために使用されます。CLONE 構文を使用するには、クローンプラグインをインストールする必要があります。 [セクション5.6.7「クローンプラグイン」](#)を参照してください。

CLONE LOCAL DATA DIRECTORY 構文は、ローカルの MySQL データディレクトリから、MySQL サーバーインスタンスが実行されているのと同じサーバーまたはノード上のディレクトリにデータをクローニングします。'clone_dir'ディレクトリは、データのクローニング先のローカルディレクトリのフルパスです。絶対パスが必要です。指定されたディレクトリは存在してはいませんが、指定されたパスは存在するパスである必要があります。MySQL サーバーには、指定されたディレクトリを作成するために必要な書込みアクセス権が必要です。詳細は、 [セクション5.6.7.2「ローカルでのデータのクローニング」](#)を参照してください。

CLONE INSTANCE 構文は、リモート MySQL サーバーインスタンス (ドナー) からデータをクローニングし、クローニング操作が開始された MySQL インスタンス (受信者) に転送します。

- **user** は、ドナー MySQL サーバーインスタンス上のクローンユーザーです。
- **host** は、ドナー MySQL サーバーインスタンスの **hostname** アドレスです。インターネットプロトコルバージョン 6 (IPv6) アドレス形式はサポートされていません。かわりに、IPv6 アドレスのエイリアスを使用できます。IPv4 アドレスはそのまま使用できます。
- **port** は、ドナー MySQL サーバーインスタンスの **port** 番号です。 (**mysqlx_port** で指定された X プロトコル ポートはサポートされていません。MySQL Router を介したドナー MySQL サーバーインスタンスへの接続もサポートされていません。)
- **IDENTIFIED BY 'password'** は、ドナー MySQL サーバーインスタンス上のクローンユーザーのパスワードを指定します。
- **DATA DIRECTORY [=] 'clone_dir'** は、クローニングするデータの受信者上のディレクトリを指定するために使用するオプションの句です。このオプションは、受信者データディレクトリ内の既存のデータを削除しない場合に使用します。絶対パスが必要であり、ディレクトリが存在していない必要があります。MySQL サーバーには、ディレクトリの作成に必要な書込みアクセス権が必要です。

オプションの **DATA DIRECTORY [=] 'clone_dir'** 句を使用しない場合、クローニング操作では、受信者データディレクトリ内の既存のデータが削除され、クローニングされたデータに置き換えられ、その後サーバーが自動的に再起動されます。

- **[REQUIRE [NO] SSL]** は、クローニングされたデータをネットワーク経由で転送するときに、暗号化された接続を使用するかどうかを明示的に指定します。明示的な指定が満たされない場合は、エラーが返されます。SSL 句が指定されていない場合、クローンはデフォルトで暗号化された接続を確立しようとし、セキュアな接続試行が失敗した場合はセキュアでない接続にフォールバックします。暗号化データをクローニングする場合は、この句が指定されているかどうかに関係なく、セキュアな接続が必要です。詳細は、 [クローニング用の暗号化された接続の構成](#)を参照してください。

リモート MySQL サーバーインスタンスからのデータのクローニングの詳細は、 [セクション5.6.7.3「リモートデータのクローニング」](#)を参照してください。

13.7.6 SET ステートメント

SET ステートメントには複数の形式があります。特定のサーバー機能に関連付けられていないフォームの説明は、このセクションのサブセクションにあります:

- **SET var_name = value** では、サーバーまたはクライアントの操作に影響する変数に値を割り当てることができません。 [セクション13.7.6.1「変数代入の SET 構文」](#)を参照してください。
- **SET CHARACTER SET** および **SET NAMES** は、サーバーへの現在の接続に関連付けられている文字セットおよび照合変数に値を割り当てます。 [セクション13.7.6.2「SET CHARACTER SET ステートメント」](#) および [セクション13.7.6.3「SET NAMES ステートメント」](#)を参照してください。

他のフォームの説明は他の場所に表示され、実装に役立つ機能に関連する他のステートメントとともにグループ化されます:

- [SET DEFAULT ROLE](#) および [SET ROLE](#) は、ユーザーアカウントのデフォルトロールおよび現在のロールを設定します。 [セクション13.7.1.9「SET DEFAULT ROLE ステートメント」](#) および [セクション13.7.1.11「SET ROLE ステートメント」](#) を参照してください。
- [SET PASSWORD](#) は、アカウントのパスワードを割り当てます。 [セクション13.7.1.10「SET PASSWORD ステートメント」](#) を参照してください。
- [SET RESOURCE GROUP](#) は、スレッドをリソースグループに割り当てます。 [セクション13.7.2.4「SET RESOURCE GROUP ステートメント」](#) を参照してください。
- [SET TRANSACTION ISOLATION LEVEL](#) は、トランザクション処理の分離レベルを設定します。 [セクション13.3.7「SET TRANSACTION ステートメント」](#) を参照してください。

13.7.6.1 変数代入の SET 構文

```
SET variable = expr [, variable = expr] ...
```

```
variable: {  
  user_var_name  
  | param_name  
  | local_var_name  
  | {GLOBAL | @@GLOBAL.} system_var_name  
  | {PERSIST | @@PERSIST.} system_var_name  
  | {PERSIST_ONLY | @@PERSIST_ONLY.} system_var_name  
  | {SESSION | @@SESSION. | @@} system_var_name  
}
```

変数割当ての [SET](#) 構文を使用すると、サーバーまたはクライアントの操作に影響する様々なタイプの変数に値を割り当てることができます:

- ユーザー定義変数。 [セクション9.4「ユーザー定義変数」](#) を参照してください。
- ストアドプロシージャやストアドファンクションのパラメータ、およびストアドプログラムのローカル変数。 [セクション13.6.4「ストアドプログラム内の変数」](#) を参照してください。
- システム変数。 [セクション5.1.8「サーバーシステム変数」](#) を参照してください。システム変数はまた、[セクション5.1.9「システム変数の使用」](#) で説明されているように、サーバーの起動時にも設定できます。

変数値を割り当てる [SET](#) ステートメントはバイナリログに書き込まれないため、レプリケーションシナリオでは、変数値を実行するホストにのみ影響します。すべてのレプリケーションホストに影響を与えるには、各ホストでステートメントを実行します。

次の各セクションでは、変数を設定するための [SET](#) 構文について説明します。 = 代入演算子を使用しますが、この目的で := 代入演算子も許可されています。

- [ユーザー定義変数の割当て](#)
- [パラメータおよびローカル変数の割当て](#)
- [システム変数の割当て](#)
- [SET エラー処理](#)
- [複数変数の割当て](#)
- [式でのシステム変数参照](#)

ユーザー定義変数の割当て

ユーザー定義変数はセッション内でローカルに作成され、そのセッションのコンテキスト内にのみ存在します。 [セクション9.4「ユーザー定義変数」](#) を参照してください。

ユーザー定義変数は `@var_name` として書き込まれ、次のように式の値が割り当てられます:

```
SET @var_name = expr;
```

例:


```
SET @name = 43;  
SET @total_tax = (SELECT SUM(tax) FROM taxable_transactions);
```

これらのステートメントで示されているように、`expr` では単純 (リテラル値) から複雑 (スカラーサブクエリーによって返される値) までの範囲を指定できます。

パフォーマンススキーマ `user_variables_by_thread` テーブルには、ユーザー定義変数に関する情報が含まれています。セクション 27.12.10 「パフォーマンススキーマのユーザー定義変数テーブル」を参照してください。

パラメータおよびローカル変数の割当て

`SET` は、定義されているストアドオブジェクトのコンテキストで、パラメータおよびローカル変数に適用されます。次の手順では、`increment` プロシージャパラメータおよび `counter` ローカル変数を使用します:

```
CREATE PROCEDURE p(increment INT)  
BEGIN  
  DECLARE counter INT DEFAULT 0;  
  WHILE counter < 10 DO  
    -- ... do work ...  
    SET counter = counter + increment;  
  END WHILE;  
END;
```

システム変数の割当て

MySQL サーバーは、その操作を構成するシステム変数を保持します。システム変数には、サーバー操作全体に影響するグローバル値、現在のセッションに影響するセッション値、またはその両方を指定できます。多くのシステム変数は動的であり、`SET` ステートメントを使用して実行時に変更し、現在のサーバーインスタンスの操作に影響を与えることができます。`SET` を使用して、特定のシステム変数をデータディレクトリ内の `mysqld-auto.cnf` ファイルに永続化し、後続の起動のためのサーバー操作に影響を与えることもできます。

セッションシステム変数を変更しても、変数を別の値に変更するか、セッションが終了するまで、その値はセッション内で有効なままです。この変更は、他のセッションには影響しません。

グローバルシステム変数を変更した場合、値は記憶され、変数を別の値に変更するかサーバーが終了するまで、新しいセッションのセッション値の初期化に使用されます。変更は、グローバル値にアクセスするすべてのクライアントに表示されます。ただし、変更は、変更後に接続するクライアントの対応するセッション値にのみ影響します。グローバル変数の変更は、現在のクライアントセッションのセッション値には影響しません (グローバル値の変更が発生したセッションにも影響しません)。

グローバルシステム変数設定をサーバーの再起動後も適用されるように永続化するには、データディレクトリ内の `mysqld-auto.cnf` ファイルに永続化します。`my.cnf` オプションファイルを手動で変更して永続的な構成変更を行うこともできますが、これは面倒であり、手動で入力した設定のエラーは後で検出されない可能性があります。構文エラーのある設定は成功せず、サーバー構成を変更しないため、システム変数を永続化する `SET` ステートメントはより便利であり、不正な設定が発生する可能性を回避します。システム変数および `mysqld-auto.cnf` ファイルの永続化の詳細は、セクション 5.1.9.3 「永続化されるシステム変数」を参照してください。

注記

グローバルシステム変数値を設定または永続化するには、常に特別な権限が必要です。通常、セッションシステム変数の値を設定するには特別な権限は必要なく、例外がありますが、すべてのユーザーが設定できます。詳細は、セクション 5.1.9.1 「システム変数権限」を参照してください。

次の説明では、システム変数を設定および永続化するための構文オプションについて説明します:

- グローバルシステム変数に値を割り当てるには、変数名の前に `GLOBAL` キーワードまたは `@@GLOBAL.` 修飾子を付けます:

```
SET GLOBAL max_connections = 1000;  
SET @@GLOBAL.max_connections = 1000;
```

- セッションシステム変数に値を割り当てるには、変数名の前に `SESSION` または `LOCAL` キーワード、`@@SESSION.`、`@@LOCAL.` または `@@` 修飾子、あるいはキーワードなしまたは修飾子なしを付けます:

```
SET SESSION sql_mode = 'TRADITIONAL';
SET LOCAL sql_mode = 'TRADITIONAL';
SET @@SESSION.sql_mode = 'TRADITIONAL';
SET @@LOCAL.sql_mode = 'TRADITIONAL';
SET @@sql_mode = 'TRADITIONAL';
SET sql_mode = 'TRADITIONAL';
```

クライアントは自分のセッション変数を変更できますが、ほかのどのクライアントのセッション変数も変更できません。

- グローバルシステム変数をデータディレクトリ内の `mysqld-auto.cnf` オプションファイルに永続化するには、変数名の前に `PERSIST` キーワードまたは `@@PERSIST` 修飾子を付けます:

```
SET PERSIST max_connections = 1000;
SET @@PERSIST.max_connections = 1000;
```

この `SET` 構文を使用すると、サーバーの再起動後も保持される構成を実行時に変更できます。 `SET GLOBAL` と同様に、`SET PERSIST` はグローバル変数のランタイム値を設定しますが、変数設定も `mysqld-auto.cnf` ファイルに書き込みます (既存の変数設定がある場合は置き換えます)。

- グローバル変数のランタイム値を設定せずにグローバルシステム変数を `mysqld-auto.cnf` ファイルに永続化するには、変数名の前に `PERSIST_ONLY` キーワードまたは `@@PERSIST_ONLY` 修飾子を付けます:

```
SET PERSIST_ONLY back_log = 100;
SET @@PERSIST_ONLY.back_log = 100;
```

`PERSIST` と同様に、`PERSIST_ONLY` は変数設定を `mysqld-auto.cnf` に書き込みます。ただし、`PERSIST` とは異なり、`PERSIST_ONLY` はグローバル変数のランタイム値を変更しません。これにより、`PERSIST_ONLY` は、サーバーの起動時にのみ設定できる読み取り専用システム変数の構成に適しています。

グローバルシステム変数値をコンパイル済の MySQL デフォルト値に設定するか、セッションシステム変数を現在対応するグローバル値に設定するには、変数値を `DEFAULT` に設定します。たとえば、次の 2 つのステートメントは、`max_join_size` のセッション値を現在のグローバル値に設定する場合と同じです:

```
SET @@SESSION.max_join_size = DEFAULT;
SET @@SESSION.max_join_size = @@GLOBAL.max_join_size;
```

`SET` を使用してグローバルシステム変数を `DEFAULT` の値またはリテラルのデフォルト値に永続化すると、変数のデフォルト値が割り当てられ、変数の設定が `mysqld-auto.cnf` に追加されます。ファイルから変数を削除するには、`RESET PERSIST` を使用します。

一部のシステム変数は永続化できないか、永続的に制限されています。 [セクション5.1.9.4「永続的で永続的に制限されないシステム変数」](#) を参照してください。

プラグインによって実装されたシステム変数は、`SET` ステートメントの実行時にプラグインがインストールされた場合に永続化できます。永続化されたプラグイン変数の割り当ては、プラグインがまだインストールされている場合、それ以降のサーバーの再起動で有効になります。プラグインがインストールされなくなった場合、サーバーが `mysqld-auto.cnf` ファイルを読み取るときにプラグイン変数は存在しなくなります。この場合、サーバーはエラーログに警告を書き込み、続行します:

```
currently unknown variable 'var_name'
was read from the persisted config file
```

システム変数の名前と値を表示するには:

- `SHOW VARIABLES` ステートメントを使用します。 [セクション13.7.7.41「SHOW VARIABLES ステートメント」](#) を参照してください。
- 「複数のパフォーマンススキーマ」テーブルは、システム変数情報を提供します。 [セクション27.12.14「パフォーマンススキーマシステム変数テーブル」](#) を参照してください。
- パフォーマンススキーマ `variables_info` テーブルには、各システム変数が最後に設定された時期とユーザーを示す情報が含まれています。 [セクション27.12.14.2「パフォーマンススキーマ variables_info テーブル」](#) を参照してください。

- パフォーマンススキーマ `persisted_variables` テーブルは、`mysqld-auto.cnf` ファイルへの SQL インタフェースを提供し、`SELECT` ステートメントを使用して実行時にその内容を検査できるようにします。セクション 27.12.14.1 「パフォーマンススキーマ `persisted_variables` テーブル」を参照してください。

SET エラー処理

SET ステートメントの変数割当てが失敗した場合、ステートメント全体が失敗し、変数は変更されず、`mysqld-auto.cnf` ファイルも変更されません。

SET では、ここで説明する状況下でエラーが発生します。ほとんどの例は、キーワード構文 (`GLOBAL` や `SESSION` など) を使用する SET ステートメントを示していますが、対応する修飾子 (`@@GLOBAL.` や `@@SESSION.` など) を使用するステートメントにも原則が当てはまります。

- SET (任意のバリエーション) を使用した読み取り専用変数の設定:

```
mysql> SET GLOBAL version = 'abc';
ERROR 1238 (HY000): Variable 'version' is a read only variable
```

- GLOBAL、PERSIST または PERSIST_ONLY を使用して、セッション値のみを持つ変数を設定します:

```
mysql> SET GLOBAL sql_log_bin = ON;
ERROR 1228 (HY000): Variable 'sql_log_bin' is a SESSION
variable and can't be used with SET GLOBAL
```

- SESSION を使用して、グローバル値のみを持つ変数を設定します:

```
mysql> SET SESSION max_connections = 1000;
ERROR 1229 (HY000): Variable 'max_connections' is a
GLOBAL variable and should be set with SET GLOBAL
```

- グローバル値のみを持つ変数を設定するための GLOBAL、PERSIST または PERSIST_ONLY の省略:

```
mysql> SET max_connections = 1000;
ERROR 1229 (HY000): Variable 'max_connections' is a
GLOBAL variable and should be set with SET GLOBAL
```

- PERSIST または PERSIST_ONLY を使用して、永続化できない変数を設定します:

```
mysql> SET PERSIST port = 3307;
ERROR 1238 (HY000): Variable 'port' is a read only variable
mysql> SET PERSIST_ONLY port = 3307;
ERROR 1238 (HY000): Variable 'port' is a non persistent read only variable
```

- @@GLOBAL., @@PERSIST., @@PERSIST_ONLY., @@SESSION. および @@修飾子は、システム変数にのみ適用されます。ユーザー定義変数、ストアードプロシージャまたはストアードファンクションのパラメータ、またはストアードプログラムのローカル変数に適用しようとすると、エラーが発生します。

- すべてのシステム変数を DEFAULT に設定できるわけではありません。このような場合、DEFAULT を割り当てるとエラーになります。

- ユーザー定義変数、ストアードプロシージャまたはストアードファンクションのパラメータ、またはストアードプログラムのローカル変数に DEFAULT を割り当てようとすると、エラーが発生します。

複数変数の割当て

SET ステートメントは、カンマで区切られた複数の変数割り当てを含むことができます。次のステートメントは、ユーザー定義変数とシステム変数に値を代入します:

```
SET @x = 1, SESSION sql_mode = '';
```

単一のステートメントで複数のシステム変数を設定した場合、ステートメントの最新の GLOBAL、PERSIST、PERSIST_ONLY または SESSION キーワードが、キーワードが指定されていない次の割当てに使用されます。

複数変数割当ての例:

```
SET GLOBAL sort_buffer_size = 1000000, SESSION sort_buffer_size = 1000000;
```

```
SET @@GLOBAL.sort_buffer_size = 1000000, @@LOCAL.sort_buffer_size = 1000000;
SET GLOBAL max_connections = 1000, sort_buffer_size = 1000000;
```

`@@GLOBAL.`, `@@PERSIST.`, `@@PERSIST_ONLY.`, `@@SESSION.` および `@@` 修飾子は、直後のシステム変数にのみ適用され、残りのシステム変数には適用されません。次のステートメントは、`sort_buffer_size` グローバル値を 50000 に、セッション値を 1000000 に設定します:

```
SET @@GLOBAL.sort_buffer_size = 50000, sort_buffer_size = 1000000;
```

式でのシステム変数参照

式でシステム変数の値を参照するには、いずれかの `@@`-modifiers を使用します (式では許可されていない `@@PERSIST.` および `@@PERSIST_ONLY.` を除く)。たとえば、次のような `SELECT` ステートメントでシステム変数値を取得できます:

```
SELECT @@GLOBAL.sql_mode, @@SESSION.sql_mode, @@sql_mode;
```

注記

式内のシステム変数への参照が `@@var_name` (`@@GLOBAL.` または `@@SESSION.` ではなく `@@` を使用) として存在する場合はセッション値を返し、それ以外の場合はグローバル値を返します。これは、常にセッション値を参照する `SET @@var_name = expr` とは異なります。

13.7.6.2 SET CHARACTER SET ステートメント

```
SET {CHARACTER SET | CHARSET}
{'charset_name' | DEFAULT}
```

このステートメントは、サーバーと現在のクライアントの間で送信されたすべての文字列を、指定されたマッピングにマップします。 `SET CHARACTER SET` は、3 つのセッションシステム変数を設定します。 `character_set_client` と `character_set_results` は指定された文字セットに設定され、 `character_set_connection` は `character_set_database` の値に設定されます。 [セクション10.4「接続文字セットおよび照合順序」](#) を参照してください。

`charset_name` は、引用符で囲むことも引用符で囲まないこともできます。

デフォルトの文字セットマッピングは、値 `DEFAULT` を使用してリストアできます。このデフォルトは、サーバー構成によって異なります。

一部の文字セットは、クライアントの文字セットとして使用できません。 `SET CHARACTER SET` でこれらを使用しようとする、エラーが発生します。 [許可されていないクライアント文字セット](#) を参照してください。

13.7.6.3 SET NAMES ステートメント

```
SET NAMES {'charset_name'
[COLLATE 'collation_name'] | DEFAULT}
```

このステートメントは、 `character_set_client`、 `character_set_connection` および `character_set_results` の 3 つのセッションシステム変数を特定の文字セットに設定します。 `character_set_connection` を `charset_name` に設定すると、 `collation_connection` も `charset_name` のデフォルトの照合順序に設定されます。 [セクション10.4「接続文字セットおよび照合順序」](#) を参照してください。

オプションの `COLLATE` 句を使用すると、照合順序を明示的に指定できます。指定する場合、照合順序は `charset_name` で許可されている照合順序のいずれかである必要があります。

`charset_name` および `collation_name` は、引用符で囲むことも引用符で囲まないこともできます。

デフォルトのマッピングは、 `DEFAULT` の値を使用してリストアできます。このデフォルトは、サーバー構成によって異なります。

一部の文字セットは、クライアントの文字セットとして使用できません。 `SET NAMES` でこれらを使用しようとする、エラーが発生します。 [許可されていないクライアント文字セット](#) を参照してください。

13.7.7 SHOW ステートメント

SHOW には、データベース、テーブル、カラムに関する情報、またはサーバーに関するステータス情報を提供するための多くの形式があります。このセクションでは、次のものについて説明します。

```
SHOW {BINARY | MASTER} LOGS
SHOW BINLOG EVENTS [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
SHOW CHARACTER SET [like_or_where]
SHOW COLLATION [like_or_where]
SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [like_or_where]
SHOW CREATE DATABASE db_name
SHOW CREATE EVENT event_name
SHOW CREATE FUNCTION func_name
SHOW CREATE PROCEDURE proc_name
SHOW CREATE TABLE tbl_name
SHOW CREATE TRIGGER trigger_name
SHOW CREATE VIEW view_name
SHOW DATABASES [like_or_where]
SHOW ENGINE engine_name {STATUS | MUTEX}
SHOW [STORAGE] ENGINES
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW EVENTS
SHOW FUNCTION CODE func_name
SHOW FUNCTION STATUS [like_or_where]
SHOW GRANTS FOR user
SHOW INDEX FROM tbl_name [FROM db_name]
SHOW MASTER STATUS
SHOW OPEN TABLES [FROM db_name] [like_or_where]
SHOW PLUGINS
SHOW PROCEDURE CODE proc_name
SHOW PROCEDURE STATUS [like_or_where]
SHOW PRIVILEGES
SHOW [FULL] PROCESSLIST
SHOW PROFILE [types] [FOR QUERY n] [OFFSET n] [LIMIT n]
SHOW PROFILES
SHOW RELAYLOG EVENTS [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
SHOW {REPLICAS | SLAVE HOSTS}
SHOW {REPLICA | SLAVE} STATUS [FOR CHANNEL channel]
SHOW [GLOBAL | SESSION] STATUS [like_or_where]
SHOW TABLE STATUS [FROM db_name] [like_or_where]
SHOW [FULL] TABLES [FROM db_name] [like_or_where]
SHOW TRIGGERS [FROM db_name] [like_or_where]
SHOW [GLOBAL | SESSION] VARIABLES [like_or_where]
SHOW WARNINGS [LIMIT [offset,] row_count]

like_or_where: {
  LIKE 'pattern'
  | WHERE expr
}
```

特定の **SHOW** ステートメントの構文に **LIKE 'pattern'** 部分が含まれている場合、**'pattern'** は SQL **%** および **_** ワイルドカード文字を含むことができる文字列です。このパターンは、ステートメント出力を一致する値に制限するために役立ちます。

いくつかの **SHOW** ステートメントは、どの行を表示するかをより柔軟に指定できる **WHERE** 句も受け入れます。 [セクション26.55「SHOW ステートメントの拡張」](#) を参照してください。

多くの MySQL API (PHP など) では、**SHOW** ステートメントから返された結果を **SELECT** からの結果セットのように処理できます。詳細は、 [第29章「Connector および API」](#) または API のドキュメントを参照してください。さらに、SQL では、**INFORMATION_SCHEMA** データベース内のテーブルに対するクエリーからの結果を操作できます。これは、**SHOW** ステートメントからの結果では簡単にはできません。 [第26章「INFORMATION_SCHEMA テーブル」](#) を参照してください。

13.7.7.1 SHOW BINARY LOGS ステートメント

```
SHOW BINARY LOGS
SHOW MASTER LOGS
```

サーバー上のバイナリログファイルを一覧表示します。このステートメントは、どのログをページできるかを決定する方法を示す、 [セクション13.4.1.1「PURGE BINARY LOGS ステートメント」](#) で説明されている手順の一部として

使用されます。SHOW BINARY LOGS には、REPLICATION CLIENT 権限 (または非推奨の SUPER 権限) が必要です。

暗号化バイナリログファイルには、ファイルの暗号化と復号化に必要な情報を格納する 512 バイトファイルヘッダーがあります。これは、SHOW BINARY LOGS によって表示されるファイルサイズに含まれます。Encrypted カラムには、バイナリログファイルが暗号化されているかどうかが表示されます。binlog_encryption=ON がサーバーに設定されている場合、バイナリログの暗号化はアクティブです。サーバーの実行中にバイナリログ暗号化がアクティブ化または非アクティブ化された場合、既存のバイナリログファイルは暗号化または復号化されません。

```
mysql> SHOW BINARY LOGS;
+-----+-----+-----+
| Log_name | File_size | Encrypted |
+-----+-----+-----+
| binlog.000015 | 724935 | Yes |
| binlog.000016 | 733481 | Yes |
+-----+-----+-----+
```

SHOW MASTER LOGS は SHOW BINARY LOGS と同等です。

13.7.7.2 SHOW BINLOG EVENTS ステートメント

```
SHOW BINLOG EVENTS
[IN 'log_name']
[FROM pos]
[LIMIT [offset,] row_count]
```

バイナリログ内のイベントを表示します。'log_name' を指定しない場合は、最初のバイナリログが表示されます。SHOW BINLOG EVENTS には、REPLICATION SLAVE 権限が必要です。

LIMIT 句の構文は、SELECT ステートメントの場合と同じです。セクション13.2.10「SELECT ステートメント」を参照してください。

注記

SHOW BINLOG EVENTS を LIMIT 句なしで発行すると、サーバーはクライアントに (サーバーによって実行された、データを変更するすべてのステートメントを含む) バイナリログの完全な内容を返すため、時間とリソースを大量に消費するプロセスが開始される可能性があります。あとの調査や分析のためにバイナリログをテキストファイルに保存するには、SHOW BINLOG EVENTS の代わりに mysqlbinlog ユーティリティを使用してください。セクション4.6.8「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」を参照してください。

SHOW BINLOG EVENTS では、バイナリログのイベントごとに次のフィールドが表示されます:

- **Log_name**
リストされるファイルの名前。
- **Pos**
イベントが発生する位置。
- **Event_type**
イベントタイプを説明する識別子。
- **Server_id**
イベントが発生したサーバーのサーバー ID。
- **End_log_pos**
次のイベントが開始される位置。これは、Pos にイベントのサイズを加えたものです。
- **Info**

イベントタイプの詳細情報。この情報の形式は、イベントタイプによって異なります。

圧縮されたトランザクションペイロードの場合、`Transaction_payload_event` は最初に単一のユニットとして印刷されてから解凍され、その内部の各イベントが印刷されます。

`SHOW BINLOG EVENTS` からの出力には、ユーザーおよびシステム変数の設定に関連した一部のイベントが含まれていません。バイナリログ内のイベントを完全に取得するには、`mysqlbinlog` を使用します。

`SHOW BINLOG EVENTS` は、リレーログファイルを操作しません。この目的には、`SHOW RELAYLOG EVENTS` を使用できます。

13.7.7.3 SHOW CHARACTER SET ステートメント

```
SHOW CHARACTER SET
[LIKE 'pattern' | WHERE expr]
```

`SHOW CHARACTER SET` ステートメントは使用可能な文字セットをすべて表示します。LIKE 句 (存在する場合) は、どの文字セット名と照合するかを示します。セクション26.55「SHOW ステートメントの拡張」で説明されているように、WHERE 句を指定すると、より一般的な条件を使用して行を選択できます。例:

```
mysql> SHOW CHARACTER SET LIKE 'latin%';
+-----+-----+-----+-----+
| Charset | Description          | Default collation | Maxlen |
+-----+-----+-----+-----+
| latin1  | cp1252 West European | latin1_swedish_ci | 1      |
| latin2  | ISO 8859-2 Central European | latin2_general_ci | 1      |
| latin5  | ISO 8859-9 Turkish   | latin5_turkish_ci  | 1      |
| latin7  | ISO 8859-13 Baltic   | latin7_general_ci  | 1      |
+-----+-----+-----+-----+
```

`SHOW CHARACTER SET` 出力には、次のカラムがあります:

- **Charset**
文字セット名。
- **説明**
文字セットの説明。
- **Default collation**
文字セットのデフォルトの照合順序。
- **Maxlen**
1 文字の格納に必要な最大バイト数。

`filename` 文字セットは、内部でのみ使用されます。そのため、`SHOW CHARACTER SET` では表示されません。

文字セット情報は、`INFORMATION_SCHEMA.CHARACTER_SETS` テーブルからも入手できます。セクション26.4「`INFORMATION_SCHEMA.CHARACTER_SETS` テーブル」を参照してください。

13.7.7.4 SHOW COLLATION ステートメント

```
SHOW COLLATION
[LIKE 'pattern' | WHERE expr]
```

このステートメントは、サーバーによってサポートされる照合順序を一覧表示します。デフォルトでは、`SHOW COLLATION` からの出力には、使用可能なすべての照合順序が含まれます。LIKE 句 (存在する場合) は、どの照合順序名と照合するかを示します。セクション26.55「SHOW ステートメントの拡張」で説明されているように、WHERE 句を指定すると、より一般的な条件を使用して行を選択できます。例:

```
mysql> SHOW COLLATION WHERE Charset = 'latin1';
```

```

+-----+-----+-----+-----+-----+
| Collation | Charset | Id | Default | Compiled | Sortlen |
+-----+-----+-----+-----+-----+
| latin1_german1_ci | latin1 | 5 | | Yes | 1 |
| latin1_swedish_ci | latin1 | 8 | Yes | Yes | 1 |
| latin1_danish_ci | latin1 | 15 | | Yes | 1 |
| latin1_german2_ci | latin1 | 31 | | Yes | 2 |
| latin1_bin | latin1 | 47 | | Yes | 1 |
| latin1_general_ci | latin1 | 48 | | Yes | 1 |
| latin1_general_cs | latin1 | 49 | | Yes | 1 |
| latin1_spanish_ci | latin1 | 94 | | Yes | 1 |
+-----+-----+-----+-----+-----+

```

SHOW COLLATION 出力には、次のカラムがあります:

- **Collation**

照合名。

- **Charset**

照合順序が関連付けられる文字セットの名前。

- **Id**

照合 ID。

- **Default**

照合順序がその文字セットのデフォルトであるかどうか。

- **Compiled**

文字セットがサーバーにコンパイルされるかどうか。

- **Sortlen**

これは、文字セットで表される文字列のソートに必要なメモリー量に関連します。

各文字セットのデフォルトの照合順序を表示するには、次のステートメントを使用します。Default は予約語であるため、それを識別子として使用するには、次のように引用符で囲む必要があります。

```

mysql> SHOW COLLATION WHERE `Default` = 'Yes';
+-----+-----+-----+-----+-----+
| Collation | Charset | Id | Default | Compiled | Sortlen |
+-----+-----+-----+-----+-----+
| big5_chinese_ci | big5 | 1 | Yes | Yes | 1 |
| dec8_swedish_ci | dec8 | 3 | Yes | Yes | 1 |
| cp850_general_ci | cp850 | 4 | Yes | Yes | 1 |
| hp8_english_ci | hp8 | 6 | Yes | Yes | 1 |
| koi8r_general_ci | koi8r | 7 | Yes | Yes | 1 |
| latin1_swedish_ci | latin1 | 8 | Yes | Yes | 1 |
...

```

照合情報は、[INFORMATION_SCHEMA COLLATIONS](#) テーブルからも入手できます。 [セクション 26.6 「INFORMATION_SCHEMA COLLATIONS テーブル」](#) を参照してください。

13.7.7.5 SHOW COLUMNS ステートメント

```

SHOW [EXTENDED] [FULL] {COLUMNS | FIELDS}
  {FROM | IN} tbl_name
  [{FROM | IN} db_name]
  [LIKE 'pattern' | WHERE expr]

```

SHOW COLUMNS は、特定のテーブル内のカラムに関する情報を表示します。これはビューに対しても機能します。SHOW COLUMNS は、ユーザーが何らかの権限を持っているカラムの情報のみを表示します。

```

mysql> SHOW COLUMNS FROM City;

```

Field	Type	Null	Key	Default	Extra
ID	int(11)	NO	PRI	NULL	auto_increment
Name	char(35)	NO			
CountryCode	char(3)	NO	MUL		
District	char(20)	NO			
Population	int(11)	NO		0	

`tbl_name FROM db_name` 構文のかわりに、`db_name.tbl_name` を使用することもできます。次の 2 つのステートメントは同等です。

```
SHOW COLUMNS FROM mytable FROM mydb;
SHOW COLUMNS FROM mydb.mytable;
```

オプションの **EXTENDED** キーワードを使用すると、MySQL が内部的に使用し、ユーザーがアクセスできない非表示カラムに関する情報が出力に含まれます。

オプションの **FULL** キーワードを指定すると、出力には、カラムの照合およびコメントに加えて、各カラムに付与されている権限が含まれます。

LIKE 句 (存在する場合) は、どのカラム名と照合するかを示します。 [セクション26.55「SHOW ステートメントの拡張」](#) で説明されているように、**WHERE** 句を指定すると、より一般的な条件を使用して行を選択できます。

テーブルの作成または変更時に MySQL によってデータ型が変更される場合があるため、データ型は **CREATE TABLE** ステートメントに基づくものとは異なる場合があります。この状態が発生する条件は、 [セクション13.1.20.7「暗黙のカラム指定の変更」](#) で説明されています。

SHOW COLUMNS は、テーブルカラムごとに次の値を表示します。

- **Field**

カラムの名前。

- **Type**

カラムのデータ型。

- **Collation**

非バイナリ文字列カラムの照合順序、またはほかのカラムの場合は **NULL**。この値は、**FULL** キーワードを使用した場合にのみ表示されます。

- **Null**

カラムの **NULL** 値可能性。この値は、**NULL** 値をカラムに格納できる場合は **YES** で、格納できない場合は **NO** です。

- **Key**

カラムがインデックス付けされているかどうか:

- **Key** が空の場合、このカラムはインデックス設定されていないか、またはマルチカラム内のセカンダリカラム (一意でないインデックス) としてのみインデックス設定されているかのどちらかです。
- **Key** が **PRI** の場合、このカラムは **PRIMARY KEY** であるか、またはマルチカラム **PRIMARY KEY** 内のいずれかのカラムです。
- **Key** が **UNI** の場合、このカラムは **UNIQUE** インデックスの最初のカラムです。(**UNIQUE** インデックスは複数の **NULL** 値を許可しますが、そのカラムが **NULL** を許可するかどうかは **Null** フィールドをチェックすることによってわかります。)
- **Key** が **MUL** の場合、このカラムは、特定の値がカラム内に複数回現れることが許可されている一意でないインデックスの最初のカラムです。

テーブルの特定の列に複数の **Key** 値が適用される場合、**Key** には、もっとも優先度の高い値が **PRI**、**UNI**、**MUL** の順序で表示されます。

UNIQUE インデックスは、**NULL** 値を含むことができず、かつテーブル内に **PRIMARY KEY** が存在しない場合は **PRI** として表示される可能性があります。**UNIQUE** インデックスは、複数の列が複合 **UNIQUE** インデックスを形成している場合は **MUL** として表示される可能性があります。この列の組み合わせは一意であるにもかかわらず、各列には引き続き、特定の値が複数回現れることがあります。

- **Default**

列のデフォルト値。これは、列のデフォルトが明示的に **NULL** に設定されている場合、または列定義に **DEFAULT** 句が含まれていない場合の **NULL** です。

- **Extra**

特定の列について使用可能な追加情報。次の場合、値は空ではありません:

- **AUTO_INCREMENT** 属性を持つ列の **auto_increment**。
- **ON UPDATE CURRENT_TIMESTAMP** 属性を持つ **on update CURRENT_TIMESTAMP for TIMESTAMP** または **DATETIME** の列。
- 生成された列の **VIRTUAL GENERATED** または **VIRTUAL STORED**。
- 式のデフォルト値を持つ列の **DEFAULT_GENERATED**。

- **権限**

列に対して持っている権限。この値は、**FULL** キーワードを使用した場合にのみ表示されます。

- **Comment**

列定義に含まれるコメント。この値は、**FULL** キーワードを使用した場合にのみ表示されます。

テーブルの列情報は、**INFORMATION_SCHEMA COLUMNS** テーブルからも入手できます。[セクション 26.8 「INFORMATION_SCHEMA COLUMNS テーブル」](#) を参照してください。非表示列に関する拡張情報は、**SHOW EXTENDED COLUMNS** のみを使用して使用できます。**COLUMNS** テーブルからは取得できません。

`mysqlshow db_name tbl_name` コマンドを使用してテーブルの列をリストできます。

DESCRIBE ステートメントは、**SHOW COLUMNS** と同様の情報を提供します。[セクション 13.8.1 「DESCRIBE ステートメント」](#) を参照してください。

また、**SHOW CREATE TABLE**、**SHOW TABLE STATUS**、および **SHOW INDEX** ステートメントでは、テーブルに関する情報も提供されます。[セクション 13.7.7 「SHOW ステートメント」](#) を参照してください。

13.7.7.6 SHOW CREATE DATABASE ステートメント

```
SHOW CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
```

指定されたデータベースを作成する **CREATE DATABASE** ステートメントを表示します。**SHOW** ステートメントに **IF NOT EXISTS** 句が含まれている場合は、このような句が出力にも含まれます。**SHOW CREATE SCHEMA** は **SHOW CREATE DATABASE** のシノニムです。

```
mysql> SHOW CREATE DATABASE test\G
***** 1. row *****
Database: test
Create Database: CREATE DATABASE `test` /*!40100 DEFAULT CHARACTER SET utf8mb4
COLLATE utf8mb4_0900_ai_ci */ /*!80014 DEFAULT ENCRYPTION='N' */

mysql> SHOW CREATE SCHEMA test\G
***** 1. row *****
Database: test
Create Database: CREATE DATABASE `test` /*!40100 DEFAULT CHARACTER SET utf8mb4
```

```
COLLATE utf8mb4_0900_ai_ci */ /*!80014 DEFAULT ENCRYPTION='N' */
```

`SHOW CREATE DATABASE` は、`sql_quote_show_create` オプションの値に従って、テーブル名とカラム名を引用符で囲みます。 [セクション5.1.8「サーバーシステム変数」](#) を参照してください。

13.7.7.7 SHOW CREATE EVENT ステートメント

```
SHOW CREATE EVENT event_name
```

このステートメントは、特定のイベントを再作成するために必要な `CREATE EVENT` ステートメントを表示します。これには、このイベントが示される元のデータベースに対する `EVENT` 権限が必要です。例 ([セクション13.7.7.18「SHOW EVENTS ステートメント」](#)) で定義され、あとで変更された同じイベント `e_daily` を使用しています):

```
mysql> SHOW CREATE EVENT myschema.e_dailyG
***** 1. row *****
      Event: e_daily
      sql_mode: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,
               NO_ZERO_IN_DATE,NO_ZERO_DATE,
               ERROR_FOR_DIVISION_BY_ZERO,
               NO_ENGINE_SUBSTITUTION
      time_zone: SYSTEM
      Create Event: CREATE DEFINER='jon'@'ghidora' EVENT `e_daily`
                   ON SCHEDULE EVERY 1 DAY
                   STARTS CURRENT_TIMESTAMP + INTERVAL 6 HOUR
                   ON COMPLETION NOT PRESERVE
                   ENABLE
                   COMMENT 'Saves total number of sessions then
                           clears the table each day'
                   DO BEGIN
                     INSERT INTO site_activity.totals (time, total)
                       SELECT CURRENT_TIMESTAMP, COUNT(*)
                         FROM site_activity.sessions;
                     DELETE FROM site_activity.sessions;
                   END
      character_set_client: utf8mb4
      collation_connection: utf8mb4_0900_ai_ci
      Database Collation: utf8mb4_0900_ai_ci
```

`character_set_client` は、このイベントが作成されたときの `character_set_client` システム変数のセッション値です。`collation_connection` は、このイベントが作成されたときの `collation_connection` システム変数のセッション値です。`Database Collation` は、このイベントが関連付けられているデータベースの照合順序です。

出力には、イベントが作成されたステータスではなく、イベント (`ENABLE`) の現在のステータスが反映されます。

13.7.7.8 SHOW CREATE FUNCTION ステートメント

```
SHOW CREATE FUNCTION func_name
```

このステートメントは、ストアドファンクションである点を除き、`SHOW CREATE PROCEDURE` と同じです。 [セクション13.7.7.9「SHOW CREATE PROCEDURE ステートメント」](#) を参照してください。

13.7.7.9 SHOW CREATE PROCEDURE ステートメント

```
SHOW CREATE PROCEDURE proc_name
```

このステートメントは、MySQL 拡張です。これは、指定されたストアドプロシージャを再作成するために使用できる正確な文字列を返します。同様のステートメントである `SHOW CREATE FUNCTION` は、ストアドファンクションに関する情報を表示します ([セクション13.7.7.8「SHOW CREATE FUNCTION ステートメント」](#) を参照してください)。

いずれかのステートメントを使用するには、ルーチン `DEFINER` として指定されたユーザー、`SHOW_ROUTINE` 権限、グローバルレベルでの `SELECT` 権限、またはルーチンを含むスコープで付与された `CREATE ROUTINE`、`ALTER ROUTINE` または `EXECUTE` 権限を持っている必要があります。`CREATE ROUTINE`、`ALTER ROUTINE` または `EXECUTE` のみがある場合、`Create Procedure` または `Create Function` フィールドに表示される値は `NULL` です。

```
mysql> SHOW CREATE PROCEDURE test.citycount\G
***** 1. row *****
Procedure: citycount
sql_mode: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,
NO_ZERO_IN_DATE,NO_ZERO_DATE,
ERROR_FOR_DIVISION_BY_ZERO,
NO_ENGINE_SUBSTITUTION
Create Procedure: CREATE DEFINER='me'@'localhost'
PROCEDURE `citycount` (IN country CHAR(3), OUT cities INT)
BEGIN
SELECT COUNT(*) INTO cities FROM world.city
WHERE CountryCode = country;
END
character_set_client: utf8mb4
collation_connection: utf8mb4_0900_ai_ci
Database Collation: utf8mb4_0900_ai_ci

mysql> SHOW CREATE FUNCTION test.hello\G
***** 1. row *****
Function: hello
sql_mode: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,
NO_ZERO_IN_DATE,NO_ZERO_DATE,
ERROR_FOR_DIVISION_BY_ZERO,
NO_ENGINE_SUBSTITUTION
Create Function: CREATE DEFINER='me'@'localhost'
FUNCTION `hello` (s CHAR(20))
RETURNS char(50) CHARSET utf8mb4
DETERMINISTIC
RETURN CONCAT('Hello, 's, '!')
character_set_client: utf8mb4
collation_connection: utf8mb4_0900_ai_ci
Database Collation: utf8mb4_0900_ai_ci
```

`character_set_client` は、このルーチンが作成されたときの `character_set_client` システム変数のセッション値です。`collation_connection` は、このルーチンが作成されたときの `collation_connection` システム変数のセッション値です。`Database Collation` は、このルーチンが関連付けられているデータベースの照合順序です。

13.7.7.10 SHOW CREATE TABLE ステートメント

```
SHOW CREATE TABLE tbl_name
```

指定されたテーブルを作成する `CREATE TABLE` ステートメントを表示します。このステートメントを使用するには、そのテーブルに対する何らかの権限が必要です。また、このステートメントはビューでも機能します。

```
mysql> SHOW CREATE TABLE t\G
***** 1. row *****
Table: t
Create Table: CREATE TABLE `t` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`s` char(60) DEFAULT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```

MySQL 8.0.16 では、MySQL によって `CHECK` 制約が実装され、`SHOW CREATE TABLE` によって表示されます。すべての `CHECK` 制約がテーブル制約として表示されます。つまり、カラム定義の一部として最初に指定された `CHECK` 制約は、カラム定義の一部ではなく別の句として表示されます。例:

```
mysql> CREATE TABLE t1 (
i1 INT CHECK (i1 <> 0), -- column constraint
i2 INT,
CHECK (i2 > i1), -- table constraint
CHECK (i2 <> 0) NOT ENFORCED -- table constraint, not enforced
);

mysql> SHOW CREATE TABLE t1\G
***** 1. row *****
Table: t1
Create Table: CREATE TABLE `t1` (
`i1` int(11) DEFAULT NULL,
`i2` int(11) DEFAULT NULL,
CONSTRAINT `t1_chk_1` CHECK ((`i1` <> 0)),
```



```
CONSTRAINT `t1_chk_2` CHECK ((`i2` > `i1`)),
CONSTRAINT `t1_chk_3` CHECK ((`i2` <= 0)) /*!80016 NOT ENFORCED */
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

`SHOW CREATE TABLE` は、`sql_quote_show_create` オプションの値に従って、テーブル名とカラム名を引用符で囲みます。 [セクション5.1.8「サーバーシステム変数」](#) を参照してください。

テーブルのストレージエンジンを変更する場合、新しいストレージエンジンに適用できないテーブルオプションはテーブル定義に保持され、必要に応じて、以前に定義されたオプションを持つテーブルを元のストレージエンジンに戻すことができます。たとえば、ストレージエンジンを InnoDB から MyISAM に変更する場合、`ROW_FORMAT=COMPACT` などの InnoDB 固有のオプションは保持されます。

```
mysql> CREATE TABLE t1 (c1 INT PRIMARY KEY) ROW_FORMAT=COMPACT ENGINE=InnoDB;
mysql> ALTER TABLE t1 ENGINE=MyISAM;
mysql> SHOW CREATE TABLE t1\G
***** 1. row *****
Table: t1
Create Table: CREATE TABLE `t1` (
  `c1` int NOT NULL,
  PRIMARY KEY (`c1`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci ROW_FORMAT=COMPACT
```

`strict mode` を無効にしてテーブルを作成する場合、指定した行フォーマットがサポートされていないと、ストレージエンジンのデフォルトの行フォーマットが使用されます。テーブルの実際の行形式は、`SHOW TABLE STATUS` に応じて `Row_format` カラムにレポートされます。`SHOW CREATE TABLE` には、`CREATE TABLE` ステートメントで指定された行形式が表示されます。

13.7.7.11 SHOW CREATE TRIGGER ステートメント

```
SHOW CREATE TRIGGER trigger_name
```

このステートメントは、指定されたトリガーを作成する `CREATE TRIGGER` ステートメントを表示します。このステートメントには、トリガーに関連付けられたテーブルに対する `TRIGGER` 権限が必要です。

```
mysql> SHOW CREATE TRIGGER ins_sum\G
***** 1. row *****
Trigger: ins_sum
sql_mode: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,
NO_ZERO_IN_DATE,NO_ZERO_DATE,
ERROR_FOR_DIVISION_BY_ZERO,
NO_ENGINE_SUBSTITUTION
SQL Original Statement: CREATE DEFINER='me'@'localhost' TRIGGER `ins_sum`
BEFORE INSERT ON `account`
FOR EACH ROW SET @sum = @sum + NEW.amount
character_set_client: utf8mb4
collation_connection: utf8mb4_0900_ai_ci
Database Collation: utf8mb4_0900_ai_ci
Created: 2018-08-08 10:10:12.61
```

`SHOW CREATE TRIGGER` 出力には、次のカラムがあります:

- **Trigger:** トリガー名。
- **sql_mode:** このトリガーが実行されるときに有効な SQL モード。
- **SQL Original Statement:** このトリガーを定義する `CREATE TRIGGER` ステートメント。
- **character_set_client:** このトリガーが作成されたときの `character_set_client` システム変数のセッション値。
- **collation_connection:** このトリガーが作成されたときの `collation_connection` システム変数のセッション値。
- **Database Collation:** このトリガーが関連付けられているデータベースの照合順序。
- **Created:** トリガーが作成された日時。これは、トリガーの `TIMESTAMP(2)` 値 (小数部は数秒) です。

トリガー情報は、`INFORMATION_SCHEMA TRIGGERS` テーブルからも入手できます。 [セクション26.45「INFORMATION_SCHEMA TRIGGERS テーブル」](#) を参照してください。

13.7.7.12 SHOW CREATE USER ステートメント

```
SHOW CREATE USER user
```

このステートメントは、指定されたユーザーを作成する `CREATE USER` ステートメントを示します。ユーザーが存在しない場合は、エラーが発生します。このステートメントには、`mysql` システムスキーマに対する `SELECT` 権限が必要です (現在のユーザーの情報を表示する場合を除く)。現在のユーザーの場合、`IDENTIFIED AS` 句でパスワードハッシュを表示するには、`mysql.user` システムテーブルに対する `SELECT` 権限が必要です。それ以外の場合、ハッシュは `<secret>` として表示されます。

アカウントに名前を付けるには、[セクション6.2.4「アカウント名の指定」](#)で説明されている形式を使用します。アカウント名のホスト名部分は、省略すると `%` にデフォルト設定されます。`CURRENT_USER` または `CURRENT_USER()` を指定して、現在のセッションに関連付けられているアカウントを参照することもできます。

`SHOW CREATE USER` からの出力の `IDENTIFIED WITH` 句に表示されるパスワードハッシュ値には、端末表示やその他の環境に悪影響を与える印刷不可能な文字が含まれている可能性があります。`print_identified_with_as_hex` システム変数 (MySQL 8.0.17 で使用可能) を有効にすると、`SHOW CREATE USER` では、このようなハッシュ値が通常の文字列リテラルとしてではなく 16 進数文字列として表示されます。印刷できない文字を含まないハッシュ値は、この変数が有効になっていても、通常の文字列リテラルとして表示されます。

```
mysql> CREATE USER 'u1'@'localhost' IDENTIFIED BY 'secret';
mysql> SET print_identified_with_as_hex = ON;
mysql> SHOW CREATE USER 'u1'@'localhost'\G
***** 1. row *****
CREATE USER for u1@localhost: CREATE USER 'u1'@'localhost'
IDENTIFIED WITH 'caching_sha2_password'
AS 0x244124303035240C7745603626313D613C4C10633E0A104B1E14135A544A7871567245614F4872344643546336546F624F6C7861326932752F45622F4F47
REQUIRE NONE PASSWORD EXPIRE DEFAULT ACCOUNT UNLOCK
PASSWORD HISTORY DEFAULT PASSWORD REUSE INTERVAL DEFAULT
PASSWORD REQUIRE CURRENT DEFAULT
```

アカウントに付与されている権限を表示するには、`SHOW GRANTS` ステートメントを使用します。[セクション13.7.7.21「SHOW GRANTS ステートメント」](#)を参照してください。

13.7.7.13 SHOW CREATE VIEW ステートメント

```
SHOW CREATE VIEW view_name
```

このステートメントは、指定されたビューを作成する `CREATE VIEW` ステートメントを表示します。

```
mysql> SHOW CREATE VIEW v\G
***** 1. row *****
View: v
Create View: CREATE ALGORITHM=UNDEFINED
DEFINER='bob'@'localhost'
SQL SECURITY DEFINER VIEW
`v` AS select 1 AS `a`,2 AS `b`
character_set_client: utf8mb4
collation_connection: utf8mb4_0900_ai_ci
```

`character_set_client` は、このビューが作成されたときの `character_set_client` システム変数のセッション値です。`collation_connection` は、このビューが作成されたときの `collation_connection` システム変数のセッション値です。

`SHOW CREATE VIEW` を使用するには、該当するビューに対する `SHOW VIEW` 権限および `SELECT` 権限が必要です。

ビュー情報は、`INFORMATION_SCHEMA VIEWS` テーブルからも参照できます。[セクション26.48「INFORMATION_SCHEMA VIEWS テーブル」](#)を参照してください。

MySQL では、異なる `sql_mode` 設定を使用すると、サポートする SQL 構文のタイプをサーバーに指示できます。たとえば、`ANSI SQL` モードを使用すると、クエリーで、MySQL で標準 SQL 連結演算子の二重バー (`||`) が正しく解釈されます。その後、項目を連結するビューを作成した場合、`sql_mode` 設定を `ANSI` とは別の値に変更すると、そのビューが無効になるという懸念がある場合があります。ただし、そのようなことはありません。MySQL は、記

述方法には関係なく、常にビュー定義を正規の形式で同じ方法で格納します。サーバーが二重バーの連結演算子を `CONCAT()` 関数にどのように変更するかを示す例を次に示します。

```
mysql> SET sql_mode = 'ANSI';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE VIEW test.v AS SELECT 'a' || 'b' as col1;
Query OK, 0 rows affected (0.01 sec)

mysql> SHOW CREATE VIEW test.v\G
***** 1. row *****
      View: v
      Create View: CREATE VIEW "v" AS select concat('a','b') AS "col1"
...
1 row in set (0.00 sec)
```

ビュー定義を正規の形式で格納する利点は、後で `sql_mode` の値を変更してもビューの結果に影響しないことです。ただし、`SELECT` の前にあるコメントが、サーバーによって定義から取り除かれるというその他の影響があります。

13.7.7.14 SHOW DATABASES ステートメント

```
SHOW {DATABASES | SCHEMAS}
[LIKE 'pattern' | WHERE expr]
```

`SHOW DATABASES` は、MySQL サーバーホスト上のデータベースを一覧表示します。 `SHOW SCHEMAS` は `SHOW DATABASES` のシノニムです。 `LIKE` 句 (存在する場合) は、どのデータベース名と照合するかを示します。 [セクション 26.55 「SHOW ステートメントの拡張」](#) で説明されているように、 `WHERE` 句を指定すると、より一般的な条件を使用して行を選択できます。

グローバルな `SHOW DATABASES` 権限を持っていないかぎり、何らかの種類の権限を持っているデータベースしか表示できません。このリストはまた、 `mysqlshow` コマンドを使用して取得することもできます。

サーバーが `--skip-show-database` オプションで起動された場合は、 `SHOW DATABASES` 権限を持っていないかぎり、このステートメントをまったく使用できません。

MySQL はデータベースをデータディレクトリ内のディレクトリとして実装するため、このステートメントは単純に、その場所にあるディレクトリを一覧表示します。ただし、実際のデータベースには対応しないディレクトリの名前が出力に含まれる可能性があります。

データベース情報は、 `INFORMATION_SCHEMA SCHEMATA` テーブルからも入手できます。 [セクション 26.31 「INFORMATION_SCHEMA SCHEMATA テーブル」](#) を参照してください。

注意

静的グローバル権限はすべてのデータベースに対する権限とみなされるため、静的グローバル権限を使用すると、ユーザーは、部分的な取消しによってデータベースレベルで制限されているデータベースを除き、 `SHOW DATABASES` を使用するが、 `INFORMATION_SCHEMA` の `SCHEMATA` テーブルを調べることで、すべてのデータベース名を表示できます。

13.7.7.15 SHOW ENGINE ステートメント

```
SHOW ENGINE engine_name {STATUS | MUTEX}
```

`SHOW ENGINE` は、ストレージエンジンに関する動作情報を表示します。これには `PROCESS` 権限が必要です。このステートメントは、次のバリエーションがあります。

```
SHOW ENGINE INNODB STATUS
SHOW ENGINE INNODB MUTEX
SHOW ENGINE PERFORMANCE_SCHEMA STATUS
```

`SHOW ENGINE INNODB STATUS` は、InnoDB ストレージエンジンの状態に関する InnoDB 標準モニターからの広範囲にわたる情報を表示します。InnoDB の処理に関する情報を提供する標準モニターやその他の InnoDB モニターについては、 [セクション 15.17 「InnoDB モニター」](#) を参照してください。

SHOW ENGINE INNODB MUTEX は、InnoDB 相互排他ロックおよび読み書きロックの統計を表示します。

注記

InnoDB mutex および rwlocks は、Performance Schema テーブルを使用してモニターすることもできます。セクション15.16.2「パフォーマンススキーマを使用した InnoDB Mutex 待機のモニタリング」を参照してください。

相互排他統計収集は、次のオプションを使用して動的に構成されます:

- mutex 統計の収集を有効にするには、次のコマンドを実行します:

```
SET GLOBAL innodb_monitor_enable='latch';
```

- mutex 統計をリセットするには、次のコマンドを実行します:

```
SET GLOBAL innodb_monitor_reset='latch';
```

- mutex 統計の収集を無効にするには、次のコマンドを実行します:

```
SET GLOBAL innodb_monitor_disable='latch';
```

SHOW ENGINE INNODB MUTEX の mutex 統計の収集は、innodb_monitor_enable='all'を設定して有効にすることも、innodb_monitor_disable='all'を設定して無効にすることもできます。

SHOW ENGINE INNODB MUTEX 出力には、次のカラムがあります:

- Type

常に InnoDB です。

- Name

mutex の場合、Name フィールドには mutex 名のみがレポートされます。rwlocks の場合、Name フィールドは rlock が実装されているソースファイルと rlock が作成されたファイル内の行番号を報告します。この行番号は、使用している MySQL のバージョンに固有です。

- Status

相互排他ロックのステータス。このフィールドには、スピン、待機、およびコールの数が報告されます。InnoDB の外部で実装される低レベルのオペレーティングシステム相互排他ロックの統計はレポートされません。

- spins はスピンの数を示します。
- waits は相互排他ロック待機の数を示します。
- calls は、mutex がリクエストされた回数を示します。

バッファプールが大きいシステムでは出力量が過度に多いため、SHOW ENGINE INNODB MUTEX では各バッファプールブロックの mutex および rw ロックはリストされません。ただし、SHOW ENGINE INNODB MUTEX はバッファプールブロック相互排他ロックおよび rw-lock の集約 BUF_BLOCK_MUTEX スピン、待機、および呼び出しの値を出力します。SHOW ENGINE INNODB MUTEX はまた、待機されなかった (os_waits=0) 相互排他ロックまたは読み書きロックも一覧表示しません。そのため、SHOW ENGINE INNODB MUTEX は、OS レベルの待機を少なくとも 1 回は発生させた、バッファプールの外部の相互排他ロックと読み書きロックに関する情報のみを表示します。

SHOW ENGINE PERFORMANCE_SCHEMA STATUS を使用して、パフォーマンススキーマコードの内部操作を検査します。

```
mysql> SHOW ENGINE PERFORMANCE_SCHEMA STATUS\G
...
***** 3. row *****
Type: performance_schema
Name: events_waits_history.size
Status: 76
***** 4. row *****
Type: performance_schema
```

```

Name: events_waits_history.count
Status: 10000
***** 5. row *****
Type: performance_schema
Name: events_waits_history.memory
Status: 760000
...
***** 57. row *****
Type: performance_schema
Name: performance_schema.memory
Status: 26459600
...

```

このステートメントは、さまざまなパフォーマンススキーマオプションがメモリー要件に与える効果について、DBA が理解できるようにすることを目的としています。

Name 値は、それぞれ、内部バッファとバッファ属性を指定する 2 つの部分で構成されます。バッファ名は、次のように解釈します。

- テーブルとして公開されていない内部バッファは括弧内に指定されます。例: `(pfs_cond_class).size`、`(pfs_mutex_class).memory`。
- `performance_schema` データベース内のテーブルとして公開されている内部バッファは、そのテーブル名で (括弧なしで) 指定されます。例: `events_waits_history.size`、`mutex_instances.count`。
- 全体としてのパフォーマンススキーマに適用される値は、`performance_schema` で始まります。例: `performance_schema.memory`。

バッファ属性には、次の意味があります。

- **size** は、テーブル内の行のサイズなど、実装で使用される内部レコードのサイズです。size の値は変更できません。
- **count** は、テーブルの行数などの内部レコードの数です。count の値は、パフォーマンススキーマの構成オプションを使用して変更できます。
- テーブルの場合、`tbl_name.memory` は **size** および **count** の製品です。全体としてのパフォーマンススキーマの場合、`performance_schema.memory` は、使用されているすべてのメモリーの合計 (ほかのすべての **memory** 値の合計) です。

場合によっては、パフォーマンススキーマの構成パラメータと `SHOW ENGINE` 値の間に直接の関係が存在します。たとえば、`events_waits_history_long.count` は `performance_schema_events_waits_history_long_size` に対応します。その他の場合、この関係はより複雑です。たとえば、`events_waits_history.count` は、`performance_schema_events_waits_history_size` (スレッド当たりの行数) に `performance_schema_max_thread_instances` (スレッド数) を掛けた値に対応します。

`SHOW ENGINE NDB STATUS`. サーバーで `NDB` ストレージエンジンが有効になっている場合、`SHOW ENGINE NDB STATUS` は、接続されているデータノードの数、クラスタの接続文字列、クラスタバイナリログのエポックや、クラスタに接続したときに MySQL Server によって作成されたさまざまなクラスタ API オブジェクトの数などのクラスタステータス情報を表示します。このステートメントからのサンプル出力を次に示します。

```

mysql> SHOW ENGINE NDB STATUS;
+-----+-----+-----+
| Type  | Name          | Status          |
+-----+-----+-----+
| ndbcluster | connection      | cluster_node_id=7,
connected_host=198.51.100.103, connected_port=1186, number_of_data_nodes=4,
number_of_ready_data_nodes=3, connect_count=0
| ndbcluster | NdbTransaction  | created=6, free=0, sizeof=212
| ndbcluster | NdbOperation    | created=8, free=8, sizeof=660
| ndbcluster | NdbIndexScanOperation | created=1, free=1, sizeof=744
| ndbcluster | NdbIndexOperation | created=0, free=0, sizeof=664
| ndbcluster | NdbRecAttr      | created=1285, free=1285, sizeof=60
| ndbcluster | NdbApiSignal    | created=16, free=16, sizeof=136
| ndbcluster | NdbLabel        | created=0, free=0, sizeof=196
| ndbcluster | NdbBranch       | created=0, free=0, sizeof=24
| ndbcluster | NdbSubroutine   | created=0, free=0, sizeof=68

```

SHOW ステートメント

```

| ndbcluster | NdbCall          | created=0, free=0, sizeof=16          |
| ndbcluster | NdbBlob          | created=1, free=1, sizeof=264        |
| ndbcluster | NdbReceiver      | created=4, free=0, sizeof=68         |
| ndbcluster | binlog           | latest_epoch=155467, latest_trans_epoch=148126,
latest_received_binlog_epoch=0, latest_handled_binlog_epoch=0,
latest_applied_binlog_epoch=0         |
+-----+-----+-----+

```

これらの各行の **Status** カラムには、クラスタへの MySQL サーバー接続とクラスタバイナリログステータスに関する情報がそれぞれ表示されます。 **Status** 情報は、カンマで区切られた一連の名前と値のペアの形式をしています。

connection の行の **Status** カラムには、次のテーブルで説明する名前と値のペアが含まれます。

名前	値
cluster_node_id	クラスタ内の MySQL サーバーのノード ID
connected_host	MySQL サーバーが接続されているクラスタ管理サーバーのホスト名または IP アドレス
connected_port	MySQL サーバーが管理サーバー (connected_host) に接続するために使用するポート
number_of_data_nodes	クラスタのために構成されているデータノードの数 (つまり、そのクラスタの config.ini ファイル内の [ndbd] セクションの数)
number_of_ready_data_nodes	実際に実行されているクラスタ内のデータノードの数
connect_count	この mysqld がクラスタデータノードに接続または再接続した回数

binlog 行の **Status** カラムには、NDB Cluster レプリケーションに関する情報が含まれています。そこに含まれている名前と値のペアについて、次の表で説明します。

名前	値
latest_epoch	この MySQL サーバー上で直近で実行された最新の工ポック (つまり、このサーバー上で実行された最新のトランザクションのシーケンス番号)
latest_trans_epoch	クラスタのデータノードによって処理された最新の工ポック
latest_received_binlog_epoch	バイナリログスレッドによって受信された最新の工ポック
latest_handled_binlog_epoch	(バイナリログへの書き込みのために) バイナリログスレッドによって処理された最新の工ポック
latest_applied_binlog_epoch	実際にバイナリログに書き込まれた最新の工ポック

詳細は、[セクション23.6「NDB Cluster レプリケーション」](#)を参照してください。

クラスタのモニタリングにもっとも役立つ可能性のある **SHOW ENGINE NDB STATUS** の出力の残りの行を、次に **Name** で一覧表示します。

- **NdbTransaction**: 作成された **NdbTransaction** オブジェクトの数とサイズ。 **NdbTransaction** は、NDB テーブル上で (**CREATE TABLE** や **ALTER TABLE** などの) テーブルスキーマ操作が実行されるたびに作成されます。
- **NdbOperation**: 作成された **NdbOperation** オブジェクトの数とサイズ。
- **NdbIndexScanOperation**: 作成された **NdbIndexScanOperation** オブジェクトの数とサイズ。
- **NdbIndexOperation**: 作成された **NdbIndexOperation** オブジェクトの数とサイズ。
- **NdbRecAttr**: 作成された **NdbRecAttr** オブジェクトの数とサイズ。一般に、これらのいずれかは、SQL ノードによってデータ操作ステートメントが実行されるたびに作成されます。

- **NdbBlob**: 作成された **NdbBlob** オブジェクトの数とサイズ。 **NdbBlob** は、 **NDB** テーブル内の **BLOB** カラムに関連する新しい操作が実行されるたびに作成されます。
- **NdbReceiver**: 作成されたすべての **NdbReceiver** オブジェクトの数とサイズ。 **created** カラム内の数は、MySQL サーバーが接続されているクラスタ内のデータノードの数と同じです。

注記

現在のセッション中に、このステートメントが実行されている SQL ノードにアクセスしている MySQL クライアントによって **NDB** テーブルに関連する操作が実行されていない場合、 **SHOW ENGINE NDB STATUS** は空の結果を返します。

13.7.7.16 SHOW ENGINES ステートメント

```
SHOW [STORAGE] ENGINES
```

SHOW ENGINES は、サーバーのストレージエンジンに関するステータス情報を表示します。これは、ストレージエンジンがサポートされているかどうかをチェックしたり、デフォルトのエンジンが何であるかを確認したりするために特に役立ちます。

MySQL ストレージエンジンについては、 [第15章「InnoDB ストレージエンジン」](#) および [第16章「代替ストレージエンジン」](#) を参照してください。

```
mysql> SHOW ENGINES\G
***** 1. row *****
  Engine: ARCHIVE
  Support: YES
  Comment: Archive storage engine
  Transactions: NO
    XA: NO
  Savepoints: NO
***** 2. row *****
  Engine: BLACKHOLE
  Support: YES
  Comment: /dev/null storage engine (anything you write to it disappears)
  Transactions: NO
    XA: NO
  Savepoints: NO
***** 3. row *****
  Engine: MRG_MYISAM
  Support: YES
  Comment: Collection of identical MyISAM tables
  Transactions: NO
    XA: NO
  Savepoints: NO
***** 4. row *****
  Engine: FEDERATED
  Support: NO
  Comment: Federated MySQL storage engine
  Transactions: NULL
    XA: NULL
  Savepoints: NULL
***** 5. row *****
  Engine: MyISAM
  Support: YES
  Comment: MyISAM storage engine
  Transactions: NO
    XA: NO
  Savepoints: NO
***** 6. row *****
  Engine: PERFORMANCE_SCHEMA
  Support: YES
  Comment: Performance Schema
  Transactions: NO
    XA: NO
  Savepoints: NO
***** 7. row *****
  Engine: InnoDB
  Support: DEFAULT
  Comment: Supports transactions, row-level locking, and foreign keys
  Transactions: YES
```

```

XA: YES
Savepoints: YES
***** 8. row *****
Engine: MEMORY
Support: YES
Comment: Hash based, stored in memory, useful for temporary tables
Transactions: NO
XA: NO
Savepoints: NO
***** 9. row *****
Engine: CSV
Support: YES
Comment: CSV storage engine
Transactions: NO
XA: NO
Savepoints: NO

```

SHOW ENGINES からの出力は、使用されている MySQL バージョンやその他の要因によって異なる可能性があります。

SHOW ENGINES 出力には、次のカラムがあります:

- **Engine**

ストレージエンジンの名前。

- **Support**

次のテーブルに示すストレージエンジンのサーバーレベルのサポート。

値	意味
行う	このエンジンはサポートされており、アクティブです
DEFAULT	YES と同様であることに加え、これがデフォルトのエンジンです
NO	このエンジンはサポートされていません
DISABLED	このエンジンはサポートされていますが、無効になっています

NO の値は、サーバーがこのエンジンに対するサポートなしでコンパイルされたことを示すため、実行時にこのエンジンを有効にすることはできません。

DISABLED の値は、サーバーがこのエンジンを無効にするオプションを使用して起動されたか、またはこのエンジンを有効にするために必要な一部のオプションが指定されなかったために発生します。後者の場合、エラーログにはオプションが無効になっている理由が含まれている必要があります。[セクション5.4.2「エラーログ」](#)を参照してください。

ストレージエンジンに対する **DISABLED** はまた、サーバーがそれをサポートするようにコンパイルされたが、`--skip-engine_name` オプションで起動された場合にも表示される可能性があります。**NDB** ストレージエンジンの場合、**DISABLED** はサーバーが **NDB Cluster** をサポートしてコンパイルされたが、`--ndbcluster` オプションで起動されなかったことを意味します。

すべての MySQL サーバーで **MyISAM** テーブルがサポートされます。**MyISAM** を無効にすることはできません。

- **Comment**

ストレージエンジンの簡単な説明。

- **トランザクション**

ストレージエンジンがトランザクションをサポートするかどうか。

- **XA**

ストレージエンジンが XA トランザクションをサポートするかどうか。

- [Savepoints](#)

ストレージエンジンがセーブポイントをサポートするかどうか。

ストレージエンジンの情報は、[INFORMATION_SCHEMA ENGINES](#) テーブルからも入手できます。 [セクション 26.13 「INFORMATION_SCHEMA ENGINES テーブル」](#) を参照してください。

13.7.7.17 SHOW ERRORS ステートメント

```
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW COUNT(*) ERRORS
```

[SHOW ERRORS](#) は [SHOW WARNINGS](#) に似た診断ステートメントですが、エラー、警告、および注意ではなく、エラーに関する情報のみを表示する点が異なります。

[LIMIT](#) 句の構文は、[SELECT](#) ステートメントの場合と同じです。 [セクション 13.2.10 「SELECT ステートメント」](#) を参照してください。

[SHOW COUNT\(*\) ERRORS](#) ステートメントは、エラーの数を表示します。この数はまた、[error_count](#) 変数からも取得できます。

```
SHOW COUNT(*) ERRORS;
SELECT @@error_count;
```

[SHOW ERRORS](#) および [error_count](#) は、警告や注意ではなく、エラーにのみ適用されます。その他の点では、[SHOW WARNINGS](#) および [warning_count](#) と同様です。特に、[SHOW ERRORS](#) が [max_error_count](#) 個を超えるメッセージに関する情報を表示できないのに対して、[error_count](#) は、エラーの数が [max_error_count](#) を超えた場合は [max_error_count](#) の値を超えることができます。

詳細は、[セクション 13.7.7.42 「SHOW WARNINGS ステートメント」](#) を参照してください。

13.7.7.18 SHOW EVENTS ステートメント

```
SHOW EVENTS
  [(FROM | IN) schema_name]
  [(LIKE 'pattern' | WHERE expr)]
```

このステートメントは、[セクション 25.4 「イベントスケジューラの使用」](#) で説明されているイベントマネージャイベントに関する情報を表示します。これには、これらのイベントが示される元のデータベースに対する [EVENT](#) 権限が必要です。

[SHOW EVENTS](#) は、そのもっとも単純な形式では、現在のスキーマ内のすべてのイベントを一覧表示します。

```
mysql> SELECT CURRENT_USER(), SCHEMA();
+-----+-----+
| CURRENT_USER() | SCHEMA() |
+-----+-----+
| jon@ghidora   | myschema |
+-----+-----+
1 row in set (0.00 sec)

mysql> SHOW EVENTS\G
***** 1. row *****
      Db: myschema
      Name: e_daily
      Definer: jon@ghidora
      Time zone: SYSTEM
      Type: RECURRING
      Execute at: NULL
      Interval value: 1
      Interval field: DAY
      Starts: 2018-08-08 11:06:34
      Ends: NULL
      Status: ENABLED
      Originator: 1
character_set_client: utf8mb4
collation_connection: utf8mb4_0900_ai_ci
Database Collation: utf8mb4_0900_ai_ci
```

特定のスキーマのイベントを表示するには、**FROM** 句を使用します。たとえば、**test** スキーマのイベントを表示するには、次のステートメントを使用します。

```
SHOW EVENTS FROM test;
```

LIKE 句 (存在する場合) は、どのイベント名と照合するかを示します。セクション26.55「**SHOW ステートメントの拡張**」で説明されているように、**WHERE** 句を指定すると、より一般的な条件を使用して行を選択できます。

SHOW EVENTS 出力には、次のカラムがあります：

- **Db**

イベントが属するスキーマ (データベース) の名前。

- **Name**

イベントの名前。

- **Definer**

イベントを作成したユーザーのアカウント ('user_name'@'host_name'形式)。

- **Time zone**

イベントのタイムゾーン。イベントのスケジュールに使用され、イベントの実行時にイベント内で有効なタイムゾーンです。デフォルト値は **SYSTEM** です。

- **Type**

イベントの繰り返しタイプ。 **ONE TIME** (一時) または **RECURRING** (繰り返し)。

- **Execute At**

ワンタイムイベントの場合、これは、イベントの作成に使用される **CREATE EVENT** ステートメントの **AT** 句、またはイベントを変更した最後の **ALTER EVENT** ステートメントで指定された **DATETIME** 値です。このカラムに表示された値は、イベントの **AT** 句に含まれた、**INTERVAL** 値の加算または減算に影響します。たとえば、イベントが **ON SCHEDULE AT CURRENT_TIMESTAMP + '1:6' DAY_HOUR** を使用して作成され、イベントが 2018-02-09 の 14:05:30 に作成された場合、カラムに表示される値は '2018-02-10 20:05:30' になります。イベントのタイミングが **AT** 句ではなく **EVERY** 句で決定される場合 (つまり、イベントが繰り返しである場合)、このカラムの値は **NULL** になります。

- **Interval Value**

繰り返しイベントの場合、イベント実行間で待機する間隔の数。一時的なイベントの場合、このカラムの値は常に **NULL** です。

- **Interval Field**

繰り返しイベントが繰り返される前に待機する間隔に使用される時間単位。一時的なイベントの場合、このカラムの値は常に **NULL** です。

- **Starts**

繰り返しイベントの開始日時。これは **DATETIME** 値として表示され、このイベントの開始日付と開始時間が定義されていない場合は **NULL** です。一時的なイベントの場合、このカラムは常に **NULL** です。定義に **STARTS** 句が含まれる繰り返しイベントの場合、このカラムには対応する **DATETIME** 値が含まれます。 **Execute At** カラムと同様に、この値は使用される式を解決します。イベントのタイミングに影響する **STARTS** 句がない場合、このカラムは **NULL** です。

- **Ends**

定義に **ENDS** 句が含まれる繰り返しイベントの場合、このカラムには対応する **DATETIME** 値が含まれます。 **Execute At** カラムと同様に、この値は使用される式を解決します。イベントのタイミングに影響する **ENDS** 句がない場合、このカラムは **NULL** です。

- **Status**

イベントステータス。ENABLED、DISABLED、SLAVESIDE_DISABLED のいずれか。SLAVESIDE_DISABLED は、イベントの作成が、レプリケーションソースとして機能する別の MySQL サーバーで発生し、レプリカとして機能している現在の MySQL サーバーにレプリケートされたが、そのイベントがレプリカで現在実行されていないことを示します。詳細は、[セクション17.5.1.16「呼び出される機能のレプリケーション」](#) の情報を参照してください。

- **Originator**

イベントが作成された MySQL サーバーのサーバー ID。レプリケーションで使用されます。この値は、ソースサーバーで実行された場合、ALTER EVENT によって、そのステートメントが発生したサーバーのサーバー ID に更新されることがあります。デフォルト値は 0 です。

- **character_set_client**

イベント作成時の character_set_client システム変数のセッション値。

- **collation_connection**

イベント作成時の collation_connection システム変数のセッション値。

- **Database Collation**

イベントが関連付けられているデータベースの照合。

SLAVESIDE_DISABLED および Originator カラムの詳細は、[セクション17.5.1.16「呼び出される機能のレプリケーション」](#) を参照してください。

SHOW EVENTS によって表示される時間は、[セクション25.4.4「イベントメタデータ」](#) で説明されているように、このイベントのタイムゾーンで示されます。

イベント情報は、INFORMATION_SCHEMA EVENTS テーブルからも入手できます。[セクション26.14「INFORMATION_SCHEMA EVENTS テーブル」](#) を参照してください。

イベントのアクションステートメントは、SHOW EVENTS の出力には表示されません。SHOW CREATE EVENT または INFORMATION_SCHEMA EVENTS テーブルを使用します。

13.7.7.19 SHOW FUNCTION CODE ステートメント

```
SHOW FUNCTION CODE func_name
```

このステートメントは、ストアドファンクションである点を除き、SHOW PROCEDURE CODE と同じです。[セクション13.7.7.27「SHOW PROCEDURE CODE ステートメント」](#) を参照してください。

13.7.7.20 SHOW FUNCTION STATUS ステートメント

```
SHOW FUNCTION STATUS  
[LIKE 'pattern' | WHERE expr]
```

このステートメントは、ストアドファンクションである点を除き、SHOW PROCEDURE STATUS と同じです。[セクション13.7.7.28「SHOW PROCEDURE STATUS ステートメント」](#) を参照してください。

13.7.7.21 SHOW GRANTS ステートメント

```
SHOW GRANTS  
[FOR user_or_role  
 [USING role [, role] ...]]  
  
user_or_role: {  
  user (see セクション6.2.4「アカウント名の指定」)  
 | role (see セクション6.2.5「ロール名の指定」)  
 }
```

このステートメントは、MySQL ユーザーアカウントまたはロールに割り当てられている権限およびロールを、権限およびロールの割当てを複製するために実行する必要がある GRANT ステートメントの形式で表示します。

注記

MySQL アカウントの非権限情報を表示するには、[SHOW CREATE USER](#) ステートメントを使用します。[セクション13.7.7.12「SHOW CREATE USER ステートメント」](#)を参照してください。

[SHOW GRANTS](#) には、`mysql` システムスキーマに対する [SELECT](#) 権限が必要ですが、現行ユーザーの権限およびロールは表示されません。

[SHOW GRANTS](#) のアカウントまたはロールに名前を付けるには、[GRANT](#) ステートメントと同じ形式 (`'jeffrey'@'localhost'` など) を使用します:

```
mysql> SHOW GRANTS FOR 'jeffrey'@'localhost';
+-----+
| Grants for jeffrey@localhost |
+-----+
| GRANT USAGE ON *.* TO 'jeffrey'@'localhost' |
| GRANT SELECT, INSERT, UPDATE ON `db1`.* TO 'jeffrey'@'localhost' |
+-----+
```

ホスト部分を省略すると、デフォルトで `'%'` に設定されます。アカウント名およびロール名の指定の詳細は、[セクション6.2.4「アカウント名の指定」](#) および [セクション6.2.5「ロール名の指定」](#) を参照してください。

現在のユーザー (サーバーへの接続に使用しているアカウント) に付与されている権限を表示するには、次のいずれかのステートメントを使用できます:

```
SHOW GRANTS;
SHOW GRANTS FOR CURRENT_USER;
SHOW GRANTS FOR CURRENT_USER();
```

実行者権限ではなく定義者権限で実行されるストアードプロシージャ内など、定義者コンテキストで [SHOW GRANTS FOR CURRENT_USER](#) (または同等の構文) が使用されている場合、表示される権限は実行者ではなく定義者の権限付与です。

以前のシリーズと比較した MySQL 8.0 では、[SHOW GRANTS](#) のグローバル権限出力に [ALL PRIVILEGES](#) が表示されなくなりました。これは、グローバルレベルの [ALL PRIVILEGES](#) の意味が、定義されている動的権限によって異なるためです。かわりに、[SHOW GRANTS](#) では、付与されている各グローバル権限が明示的にリストされます:

```
mysql> SHOW GRANTS FOR 'root'@'localhost';
+-----+
| Grants for root@localhost |
+-----+
| GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, RELOAD, |
| SHUTDOWN, PROCESS, FILE, REFERENCES, INDEX, ALTER, SHOW DATABASES, |
| SUPER, CREATE TEMPORARY TABLES, LOCK TABLES, EXECUTE, REPLICATION |
| SLAVE, REPLICATION CLIENT, CREATE VIEW, SHOW VIEW, CREATE ROUTINE, |
| ALTER ROUTINE, CREATE USER, EVENT, TRIGGER, CREATE TABLESPACE, |
| CREATE ROLE, DROP ROLE ON *.* TO 'root'@'localhost' WITH GRANT |
| OPTION |
| GRANT PROXY ON "@" TO 'root'@'localhost' WITH GRANT OPTION |
+-----+
```

[SHOW GRANTS](#) 出力を処理するアプリケーションは、それに応じて調整する必要があります。

グローバルレベルでは、[GRANT OPTION](#) は付与されているすべての静的グローバル権限に適用されますが (いずれかに付与されている場合)、付与されている動的権限に個別に適用されます。[SHOW GRANTS](#) では、グローバル権限は次のように表示されます:

- 付与されているすべての静的権限 (存在する場合) をリストする行 (該当する場合は [WITH GRANT OPTION](#) を含む)。
- [GRANT OPTION](#) が付与されているすべての付与された動的権限 ([WITH GRANT OPTION](#) を含む) がリストされた行。
- [GRANT OPTION](#) が付与されていない、付与されているすべての動的権限が [WITH GRANT OPTION](#) なしでリストされた行。

オプションの **USING** 句を使用すると、**SHOW GRANTS** でユーザーのロールに関連付けられている権限を調べることができます。 **USING** 句で指定された各ロールをユーザーに付与する必要があります。

次のように、ユーザー **u1** に **r1** および **r2** のロールが割り当てられているとします：

```
CREATE ROLE `r1`, `r2`;
GRANT SELECT ON db1.* TO `r1`;
GRANT INSERT, UPDATE, DELETE ON db1.* TO `r2`;
CREATE USER `u1`@`localhost` IDENTIFIED BY `u1pass`;
GRANT `r1`, `r2` TO `u1`@`localhost`;
```

USING を使用しない **SHOW GRANTS** には、付与されたロールが表示されます：

```
mysql> SHOW GRANTS FOR `u1`@`localhost`;
+-----+
| Grants for u1@localhost |
+-----+
| GRANT USAGE ON *.* TO `u1`@`localhost` |
| GRANT `r1`@`%`,`r2`@`%` TO `u1`@`localhost` |
+-----+
```

USING 句を追加すると、その句で指定された各ロールに関連付けられた権限もステートメントに表示されます：

```
mysql> SHOW GRANTS FOR `u1`@`localhost` USING `r1`;
+-----+
| Grants for u1@localhost |
+-----+
| GRANT USAGE ON *.* TO `u1`@`localhost` |
| GRANT SELECT ON `db1`.* TO `u1`@`localhost` |
| GRANT `r1`@`%`,`r2`@`%` TO `u1`@`localhost` |
+-----+
mysql> SHOW GRANTS FOR `u1`@`localhost` USING `r2`;
+-----+
| Grants for u1@localhost |
+-----+
| GRANT USAGE ON *.* TO `u1`@`localhost` |
| GRANT INSERT, UPDATE, DELETE ON `db1`.* TO `u1`@`localhost` |
| GRANT `r1`@`%`,`r2`@`%` TO `u1`@`localhost` |
+-----+
mysql> SHOW GRANTS FOR `u1`@`localhost` USING `r1`, `r2`;
+-----+
| Grants for u1@localhost |
+-----+
| GRANT USAGE ON *.* TO `u1`@`localhost` |
| GRANT SELECT, INSERT, UPDATE, DELETE ON `db1`.* TO `u1`@`localhost` |
| GRANT `r1`@`%`,`r2`@`%` TO `u1`@`localhost` |
+-----+
```

注記

アカウントに付与された権限は常に有効ですが、ロールは有効ではありません。アカウントのアクティブロールは、[activate_all_roles_on_login](#) システム変数の値、アカウントのデフォルトロール、および **SET ROLE** がセッション内で実行されたかどうかによって、セッション間で異なる場合があります。

MySQL 8.0.16 以上では、グローバル権限を特定のスキーマに適用できないように制限できるように、グローバル権限の部分的な取消しがサポートされています ([セクション6.2.12「部分取消しを使用した権限の制限」](#)を参照)。特定のスキーマに対して取り消されたグローバルスキーマ権限を示すために、**SHOW GRANTS** 出力には **REVOKE** ステートメントが含まれています：

```
mysql> SET PERSIST partial_revokes = ON;
mysql> CREATE USER u1;
mysql> GRANT SELECT, INSERT, DELETE ON *.* TO u1;
mysql> REVOKE SELECT, INSERT ON mysql.* FROM u1;
mysql> REVOKE DELETE ON world.* FROM u1;
mysql> SHOW GRANTS FOR u1;
+-----+
| Grants for u1@% |
+-----+
| GRANT SELECT, INSERT, DELETE ON *.* TO `u1`@`%` |
| REVOKE SELECT, INSERT ON `mysql`.* FROM `u1`@`%` |
| REVOKE DELETE ON `world`.* FROM `u1`@`%` |
+-----+
```

`SHOW GRANTS` では、指定されたアカウントで使用できるが、別のアカウントに付与されている権限は表示されません。たとえば、匿名アカウントが存在する場合、名前付きアカウントはその権限を使用できますが、`SHOW GRANTS` では表示されません。

`SHOW GRANTS` では、`mandatory_roles` システム変数値で指定された必須ロールが次のように表示されます：

- `FOR` 句を指定しない `SHOW GRANTS` では、現行ユーザーの権限が表示され、必須ロールが含まれます。
- `SHOW GRANTS FOR user` には、指定したユーザーの権限が表示され、必須ロールは含まれません。

この動作は、`SHOW GRANTS FOR user` の出力を使用して、指定されたユーザーに明示的に付与される権限を決定するアプリケーションの利点です。その出力に必須ロールが含まれていたため、ユーザーに明示的に付与されたロールを必須ロールと区別することは困難です。

現行ユーザーの場合、アプリケーションでは、`SHOW GRANTS` または `SHOW GRANTS FOR CURRENT_USER` をそれぞれ使用して、必須ロールの有無にかかわらず権限を決定できます。

13.7.7.22 SHOW INDEX ステートメント

```
SHOW [EXTENDED] {INDEX | INDEXES | KEYS}
{FROM | IN} tbl_name
[[(FROM | IN) db_name]
[WHERE expr]
```

`SHOW INDEX` は、テーブルインデックス情報を返します。この形式は、ODBC での `SQLStatistics` 呼び出しの形式に似ています。このステートメントには、このテーブル内のいずれかのカラムに対する何らかの権限が必要です。

```
mysql> SHOW INDEX FROM CityG
***** 1. row *****
    Table: city
  Non_unique: 0
    Key_name: PRIMARY
  Seq_in_index: 1
  Column_name: ID
    Collation: A
  Cardinality: 4188
    Sub_part: NULL
     Packed: NULL
       Null:
  Index_type: BTREE
    Comment:
  Index_comment:
    Visible: YES
  Expression: NULL
***** 2. row *****
    Table: city
  Non_unique: 1
    Key_name: CountryCode
  Seq_in_index: 1
  Column_name: CountryCode
    Collation: A
  Cardinality: 232
    Sub_part: NULL
     Packed: NULL
       Null:
  Index_type: BTREE
    Comment:
  Index_comment:
    Visible: YES
  Expression: NULL
```

`tbl_name FROM db_name` 構文のかわりに、`db_name` を使用することもできます。`tbl_name`。次の 2 つのステートメントは同等です。

```
SHOW INDEX FROM mytable FROM mydb;
SHOW INDEX FROM mydb.mytable;
```

オプションの `EXTENDED` キーワードを指定すると、MySQL が内部的に使用し、ユーザーがアクセスできない非表示インデックスに関する情報が出力に含まれます。

セクション26.55「SHOW ステートメントの拡張」で説明されているように、WHERE 句を指定すると、より一般的な条件を使用して行を選択できます。

SHOW INDEX は、次のフィールドを返します。

- **Table**

テーブルの名前。

- **Non_unique**

このインデックスが重複を含むことができない場合は 0、できる場合は 1。

- **Key_name**

インデックスの名前。このインデックスが主キーである場合、その名前は常に PRIMARY です。

- **Seq_in_index**

インデックス内のコラムシーケンス番号であり、1 から始まります。

- **Column_name**

コラム名。Expression コラムの説明も参照してください。

- **Collation**

インデックス内でのコラムのソート方法。これには、A (昇順)、D (降順) または NULL (ソートなし) の値を指定できます。

- **Cardinality**

このインデックス内の一意の値の数の推定値。この数を更新するには、ANALYZE TABLE または (MyISAM テーブルの場合は)myisamchk -a を実行します。

Cardinality は整数として格納された統計に基づいてカウントされるため、この値は、小さなテーブルの場合でも必ずしも正確であるとはかぎりません。カーディナリティーが高いほど、MySQL が結合を実行するときにこのインデックスを使用する可能性は高くなります。

- **Sub_part**

インデックス接頭辞。つまり、コラムが部分的にのみインデックス付けされている場合はインデックス付けされた文字の数で、コラム全体がインデックス付けされている場合は NULL です。

注記

接頭辞 limits はバイト単位で測定されます。ただし、CREATE TABLE、ALTER TABLE および CREATE INDEX ステートメントのインデックス指定の接頭辞 lengths は、非バイナリ文字列型 (CHAR, VARCHAR, TEXT) の場合は文字数として解釈され、バイナリ文字列型 (BINARY, VARBINARY, BLOB) の場合はバイト数として解釈されます。マルチバイト文字セットを使用する非バイナリ文字列コラムに接頭辞の長さを指定する場合は、これを考慮してください。

インデックス接頭辞の詳細は、セクション8.3.5「コラムインデックス」および セクション13.1.15「CREATE INDEX ステートメント」を参照してください。

- **Packed**

キーがパックされる方法を示します。パックされない場合は NULL。

- **Null**

このコラムに NULL 値を含めることができる場合は YES が、できない場合は " が含まれます。

- **Index_type**

使用されるインデックス方法 (BTREE、FULLTEXT、HASH、RTREE)。

- **Comment**

各カラムで説明されていないこのインデックスに関する情報 (このインデックスが無効になっている場合の `disabled` など)。

- **Index_comment**

このインデックスが作成されたときに `COMMENT` 属性でインデックスに対して提供された任意のコメント。

- **Visible**

オプティマイザがインデックスを参照できるかどうか。 [セクション8.3.12「不可視のインデックス」](#) を参照してください。

- **式**

MySQL 8.0.13 以上では、`Column_name` カラムと `Expression` カラムの両方に影響する機能キー部分 (機能キー部品を参照) がサポートされています:

- 機能しないキー部分の場合、`Column_name` はキー部分でインデックス付けされたカラムを示し、`Expression` は `NULL` です。
- 関数キー部分の場合、`Column_name` カラムは `NULL` で、`Expression` はキー部分の式を示します。

テーブルインデックスに関する情報は、`INFORMATION_SCHEMA STATISTICS` テーブルからも入手できます。 [セクション26.34「INFORMATION_SCHEMA STATISTICS テーブル」](#) を参照してください。非表示インデックスに関する拡張情報は、`SHOW EXTENDED INDEX` のみを使用して使用できます。`STATISTICS` テーブルからは取得できません。

`mysqlshow -k db_name tbl_name` コマンドを使用してテーブルのインデックスをリストできます。

13.7.7.23 SHOW MASTER STATUS ステートメント

```
SHOW MASTER STATUS
```

このステートメントは、ソースサーバーのバイナリログファイルに関するステータス情報を提供します。`REPLICATION CLIENT` 権限 (または非推奨の `SUPER` 権限) が必要です。

例:

```
mysql> SHOW MASTER STATUS\G
***** 1. row *****
      File: source-bin.000002
      Position: 1307
      Binlog_Do_DB: test
      Binlog_Ignore_DB: manual, mysql
      Executed_Gtid_Set: 3E11FA47-71CA-11E1-9E33-C80AA9429562:1-5
      1 row in set (0.00 sec)
```

グローバルトランザクション ID が使用されている場合、`Executed_Gtid_Set` には、ソースで実行されたトランザクションの GTID のセットが表示されます。これは、このサーバーの `gtid_executed` システム変数の値、およびこのサーバーの `SHOW REPLICAS | SLAVE STATUS` の出力における `Executed_Gtid_Set` の値と同じです。

13.7.7.24 SHOW OPEN TABLES ステートメント

```
SHOW OPEN TABLES
  [(FROM | IN) db_name]
  [(LIKE 'pattern' | WHERE expr)]
```

`SHOW OPEN TABLES` は、現在テーブルキャッシュ内で開いている `TEMPORARY` 以外のテーブルを一覧表示します。 [セクション8.4.3.1「MySQL でのテーブルのオープンとクローズの方法」](#) を参照してください。`FROM` 句 (存在する場合) は、表示されるテーブルを `db_name` データベース内に存在するテーブルに制限します。`LIKE` 句 (存在する場合) は、どのテーブル名と照合するかを示します。 [セクション26.55「SHOW ステートメントの拡張」](#) で説明されているように、`WHERE` 句を指定すると、より一般的な条件を使用して行を選択できます。

SHOW OPEN TABLES 出力には、次のカラムがあります:

- Database

このテーブルを含むデータベース。

- Table

テーブル名

- In_use

このテーブルのために存在するテーブルロックまたはロック要求の数。たとえば、あるクライアントが `LOCK TABLE t1 WRITE` を使用してテーブルのロックを取得した場合、`In_use` は 1 です。テーブルがロックされている間に別のクライアントが `LOCK TABLE t1 WRITE` を発行すると、クライアントはブロックされ、ロックを待機しますが、ロックリクエストによって `In_use` は 2 になります。このカウントが 0 の場合、このテーブルは開いています。現在使用されていません。`In_use` はまた、`HANDLER ... OPEN` ステートメントによって増加し、`HANDLER ... CLOSE` ステートメントによって減少します。

- Name_locked

テーブル名がロックされているかどうか。名前のロックは、テーブルの削除や名前の変更などの操作に使用されません。

テーブルに対する権限を持っていない場合、そのテーブルは `SHOW OPEN TABLES` の出力に表示されません。

13.7.7.25 SHOW PLUGINS ステートメント

```
SHOW PLUGINS
```

`SHOW PLUGINS` は、サーバープラグインについての情報を表示します。

`SHOW PLUGINS` の出力の例:

```
mysql> SHOW PLUGINS
***** 1. row *****
  Name: binlog
  Status: ACTIVE
  Type: STORAGE ENGINE
  Library: NULL
  License: GPL
***** 2. row *****
  Name: CSV
  Status: ACTIVE
  Type: STORAGE ENGINE
  Library: NULL
  License: GPL
***** 3. row *****
  Name: MEMORY
  Status: ACTIVE
  Type: STORAGE ENGINE
  Library: NULL
  License: GPL
***** 4. row *****
  Name: MyISAM
  Status: ACTIVE
  Type: STORAGE ENGINE
  Library: NULL
  License: GPL
...
```

`SHOW PLUGINS` 出力には、次のカラムがあります:

- Name

`INSTALL PLUGIN` や `UNINSTALL PLUGIN` などのステートメントでプラグインを参照するために使用される名前。

- Status

プラグインステータス ([ACTIVE](#), [INACTIVE](#), [DISABLED](#), [DELETING](#) のいずれか、または [DELETED](#))。

- [Type](#)

プラグインのタイプ ([STORAGE ENGINE](#)、[INFORMATION_SCHEMA](#)、[AUTHENTICATION](#) など)。

- [Library](#)

プラグイン共有ライブラリファイルの名前。これは、[INSTALL PLUGIN](#) や [UNINSTALL PLUGIN](#) などのステートメントでプラグインファイルを参照するために使用される名前です。このファイルは、[plugin_dir](#) システム変数によって指名されたディレクトリに置かれます。ライブラリ名が [NULL](#) である場合、プラグインはコンパイルされませんが、[UNINSTALL PLUGIN](#) でアンインストールできません。

- [License](#)

プラグインのライセンス方法 ([GPL](#) など)。

[INSTALL PLUGIN](#) とともにインストールされたプラグインの場合、[Name](#) および [Library](#) の値も [mysql.plugin](#) システムテーブルに登録されます。

[SHOW PLUGINS](#) によって表示される情報の基礎を形成するプラグインのデータ構造については、[The MySQL Plugin API](#)を参照してください。

プラグイン情報は、[INFORMATION_SCHEMA.PLUGINS](#) テーブルからも入手できます。 [セクション 26.22 「INFORMATION_SCHEMA PLUGINS テーブル」](#) を参照してください。

13.7.7.26 SHOW PRIVILEGES ステートメント

```
SHOW PRIVILEGES
```

[SHOW PRIVILEGES](#) は、MySQL サーバーがサポートするシステム権限のリストを表示します。表示される権限には、すべての静的権限と、現在登録されているすべての動的権限が含まれます。

```
mysql> SHOW PRIVILEGES\G
***** 1. row *****
Privilege: Alter
Context: Tables
Comment: To alter the table
***** 2. row *****
Privilege: Alter routine
Context: Functions,Procedures
Comment: To alter or drop stored functions/procedures
***** 3. row *****
Privilege: Create
Context: Databases,Tables,Indexes
Comment: To create new databases and tables
***** 4. row *****
Privilege: Create routine
Context: Databases
Comment: To use CREATE FUNCTION/PROCEDURE
***** 5. row *****
Privilege: Create temporary tables
Context: Databases
Comment: To use CREATE TEMPORARY TABLE
...
```

特定のユーザーに属する権限は、[SHOW GRANTS](#) ステートメントによって表示されます。詳細は、[セクション 13.7.7.21 「SHOW GRANTS ステートメント」](#) を参照してください。

13.7.7.27 SHOW PROCEDURE CODE ステートメント

```
SHOW PROCEDURE CODE proc_name
```

このステートメントは、デバッグサポート付きで構築されたサーバーでのみ使用可能な MySQL 拡張です。これは、指定されたストアードプロシージャの内部実装の表現を表示します。同様のステートメントである [SHOW FUNCTION CODE](#) は、ストアードファンクションに関する情報を表示します ([セクション 13.7.7.19 「SHOW FUNCTION CODE ステートメント」](#) を参照してください)。

いずれかのステートメントを使用するには、ルーチン `DEFINER` として指定されたユーザーであるか、`SHOW_ROUTINE` 権限を持っているか、グローバルレベルの `SELECT` 権限を持っている必要があります。

指定されたルーチンが使用可能な場合、各ステートメントは結果セットを生成します。結果セット内の各行は、このルーチン内の 1 つの「命令」に対応します。最初の列は、0 で始まる順序番号である `Pos` です。2 番目の列は `Instruction` であり、SQL ステートメント (通常は、元のソースから変更されています)、またはストアードルーチンのハンドラに対してのみ意味を持つディレクティブが含まれています。

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE p1 ()
  BEGIN
    DECLARE fanta INT DEFAULT 55;
    DROP TABLE t2;
    LOOP
      INSERT INTO t3 VALUES (fanta);
    END LOOP;
  END//
Query OK, 0 rows affected (0.01 sec)

mysql> SHOW PROCEDURE CODE p1//
+----+-----+
| Pos | Instruction |
+----+-----+
| 0 | set fanta@0 55 |
| 1 | stmt 9 "DROP TABLE t2" |
| 2 | stmt 5 "INSERT INTO t3 VALUES (fanta)" |
| 3 | jump 2 |
+----+-----+
4 rows in set (0.00 sec)

mysql> CREATE FUNCTION test.hello (s CHAR(20))
  RETURNS CHAR(50) DETERMINISTIC
  RETURN CONCAT("Hello, 's,!");
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW FUNCTION CODE test.hello;
+----+-----+
| Pos | Instruction |
+----+-----+
| 0 | freturn 254 concat("Hello, 's@0,!") |
+----+-----+
1 row in set (0.00 sec)
```

この例では、実行不可能な `BEGIN` および `END` ステートメントが消えており、`DECLARE variable_name` ステートメントでは、実行可能ファイルの部分 (デフォルトが割り当てられている部分) のみが表示されています。ソースから取得されたステートメントごとに、コードワード `stmt` とそれに続くタイプ (`DROP` を示す 9、`INSERT` を示す 5 など) が存在します。最終行には、`GOTO instruction #2` を示す命令 `jump 2` が含まれています。

13.7.7.28 SHOW PROCEDURE STATUS ステートメント

```
SHOW PROCEDURE STATUS
[LIKE 'pattern' | WHERE expr]
```

このステートメントは、MySQL 拡張です。これは、ストアードプロシージャの特性 (データベース、名前、型、作成者、作成日と変更日、文字セット情報など) を返します。同様のステートメントである `SHOW FUNCTION STATUS` は、ストアードファンクションに関する情報を表示します ([セクション 13.7.7.20 「SHOW FUNCTION STATUS ステートメント」](#) を参照してください)。

いずれかのステートメントを使用するには、ルーチン `DEFINER` として指定されたユーザー、`SHOW_ROUTINE` 権限、グローバルレベルでの `SELECT` 権限、またはルーチンを含むスコープで付与された `CREATE ROUTINE`、`ALTER ROUTINE` または `EXECUTE` 権限を持っている必要があります。

`LIKE` 句 (存在する場合) は、どのプロシージャまたは関数名と照合するかを示します。 [セクション 26.55 「SHOW ステートメントの拡張」](#) で説明されているように、`WHERE` 句を指定すると、より一般的な条件を使用して行を選択できます。

```
mysql> SHOW PROCEDURE STATUS LIKE 'sp1\G
***** 1. row *****
Db: test
```

```

Name: sp1
Type: PROCEDURE
Definer: testuser@localhost
Modified: 2018-08-08 13:54:11
Created: 2018-08-08 13:54:11
Security_type: DEFINER
Comment:
character_set_client: utf8mb4
collation_connection: utf8mb4_0900_ai_ci
Database Collation: utf8mb4_0900_ai_ci

mysql> SHOW FUNCTION STATUS LIKE 'hello\G
***** 1. row *****
      Db: test
      Name: hello
      Type: FUNCTION
      Definer: testuser@localhost
      Modified: 2020-03-10 11:10:03
      Created: 2020-03-10 11:10:03
      Security_type: DEFINER
      Comment:
character_set_client: utf8mb4
collation_connection: utf8mb4_0900_ai_ci
Database Collation: utf8mb4_0900_ai_ci

```

`character_set_client` は、このルーチンが作成されたときの `character_set_client` システム変数のセッション値です。`collation_connection` は、このルーチンが作成されたときの `collation_connection` システム変数のセッション値です。`Database Collation` は、このルーチンが関連付けられているデータベースの照合順序です。

ストアルーチン情報は、`INFORMATION_SCHEMA PARAMETERS` および `ROUTINES` テーブルからも入手できます。[セクション26.20「INFORMATION_SCHEMA PARAMETERS テーブル」](#) および [セクション26.30「INFORMATION_SCHEMA ROUTINES テーブル」](#) を参照してください。

13.7.7.29 SHOW PROCESSLIST ステートメント

```
SHOW [FULL] PROCESSLIST
```

MySQL プロセスリストには、サーバー内で実行されているスレッドのセットによって現在実行されている操作が示されます。`SHOW PROCESSLIST` ステートメントは、プロセス情報のソースです。このステートメントと他のソースの比較については、[プロセス情報のソース](#) を参照してください。

注記

MySQL 8.0.22 の時点では、`SHOW PROCESSLIST` の代替実装はパフォーマンススキーマ `processlist` テーブルに基づいて使用できます。これは、デフォルトの `SHOW PROCESSLIST` 実装とは異なり、`mutex` を必要とせず、パフォーマンス特性が向上します。詳細は、[セクション27.12.19.9「processlist テーブル」](#) を参照してください。

`PROCESS` 権限を持っている場合は、他のユーザーに属するスレッドも含めて、すべてのスレッドを表示できます。それ以外の場合 (`PROCESS` 権限なし)、非匿名ユーザーは自分のスレッドに関する情報にはアクセスできますが、他のユーザーのスレッドにはアクセスできず、匿名ユーザーはスレッド情報にアクセスできません。

`FULL` キーワードを指定しない場合、`SHOW PROCESSLIST` では、`Info` フィールドに各ステートメントの最初の 100 文字のみが表示されます。

`SHOW PROCESSLIST` ステートメントは、「接続が多すぎます」というエラーメッセージが表示されるために、何が発生しているかを突き止めたい場合に非常に役立ちます。MySQL では、管理者が常にシステムに接続してチェックできるように、`CONNECTION_ADMIN` 権限 (または非推奨の `SUPER` 権限) を持つアカウントで使用される追加接続が予約されています (この権限をすべてのユーザーに付与していないことを前提としています)。

スレッドは、`KILL` ステートメントを使用して強制終了できます。[セクション13.7.8.4「KILL ステートメント」](#) を参照してください。

`SHOW PROCESSLIST` 出力の例:

```

mysql> SHOW FULL PROCESSLIST\G
***** 1. row *****

```

```
Id: 1
User: system user
Host:
  db: NULL
Command: Connect
Time: 1030455
State: Waiting for master to send event
Info: NULL
***** 2. row *****
Id: 2
User: system user
Host:
  db: NULL
Command: Connect
Time: 1004
State: Has read all relay log; waiting for the slave
      I/O thread to update it
Info: NULL
***** 3. row *****
Id: 3112
User: replikator
Host: artemis:2204
  db: NULL
Command: Binlog Dump
Time: 2144
State: Has sent all binlog to slave; waiting for binlog to be updated
Info: NULL
***** 4. row *****
Id: 3113
User: replikator
Host: iconnect2:45781
  db: NULL
Command: Binlog Dump
Time: 2086
State: Has sent all binlog to slave; waiting for binlog to be updated
Info: NULL
***** 5. row *****
Id: 3123
User: stefan
Host: localhost
  db: apollon
Command: Query
Time: 0
State: NULL
Info: SHOW FULL PROCESSLIST
```

SHOW PROCESSLIST 出力には、次のカラムがあります:

- **Id**

接続識別子。これは、`INFORMATION_SCHEMA.PROCESSLIST` テーブルの `ID` カラムに表示される値と同じで、パフォーマンススキーマ `threads` テーブルの `PROCESSLIST_ID` カラムに表示され、スレッド内で `CONNECTION_ID()` 関数によって返されます。

- **User**

このステートメントを発行した MySQL ユーザー。 `system user` の値は、遅延行ハンドラスレッド、レプリカホストで使用される I/O または SQL スレッドなど、タスクを内部的に処理するためにサーバーによって起動される非クライアントスレッドを指します。 `system user` の場合、`Host` カラムにホストが指定されていません。 `unauthenticated user` は、クライアント接続に関連付けられたが、クライアントユーザーの認証がまだ行われていないスレッドを参照します。 `event_scheduler` は、スケジュールされたイベントをモニターするスレッドを指します (セクション25.4「イベントスケジューラの使用」を参照)。

注記

`system user` の `User` 値は、`SYSTEM_USER` 権限とは異なります。前者は内部スレッドを指定します。後者は、システムユーザーと通常のユーザーアカウントカテゴリを区別します (セクション6.2.11「アカウントカテゴリ」を参照)。

- **Host**

ステートメントを発行するクライアントのホスト名 (ホストがない `system user` を除く)。TCP/IP 接続のホスト名は、`host_name:client_port` 形式でレポートされるため、どのクライアントが何を実行しているかを簡単に判別できます。

- db

スレッドのデフォルトデータベース。選択されていない場合は `NULL`。

- コマンド

スレッドがクライアントのかわりに実行しているコマンドのタイプ。セッションがアイドル状態の場合は `Sleep`。スレッドコマンドの説明については、[セクション8.14「サーバスレッド \(プロセス\) 情報の確認」](#)を参照してください。このコラムの値は、クライアント/サーバープロトコルの `COM_xxx` コマンドと `Com_xxx` ステータス変数に対応します。[セクション5.1.10「サーバスステータス変数」](#)を参照してください。

- 時間

スレッドが現在の状態になってからの秒数。レプリカ SQL スレッドの場合、この値は、最後にレプリケートされたイベントのタイムスタンプとレプリカホストのリアルタイムの間の秒数です。[セクション17.2.3「レプリケーションスレッド」](#)を参照してください。

- State

スレッドが行なっていることを示すアクション、イベント、または状態。State の値の詳細は、[セクション8.14「サーバスレッド \(プロセス\) 情報の確認」](#)を参照してください。

ほとんどの状態がきわめてすばやい操作に対応します。スレッドの状態が何秒間も特定の状態にとどまっている場合は、調査が必要な問題が発生している可能性があります。

- Info

スレッドが実行しているステートメント。ステートメントを実行していない場合は `NULL`。このステートメントは、サーバーに送信されるステートメント、またはこのステートメントがほかのステートメントを実行する場合は、もっとも内側のステートメントである可能性があります。たとえば、`CALL` ステートメントが、`SELECT` ステートメントを実行しているストアードプロシージャを実行する場合、Info 値はその `SELECT` ステートメントを示します。

13.7.7.30 SHOW PROFILE ステートメント

```
SHOW PROFILE [type [, type] ... ]
[FOR QUERY n]
[LIMIT row_count [OFFSET offset]]

type: {
  ALL
  | BLOCK IO
  | CONTEXT SWITCHES
  | CPU
  | IPC
  | MEMORY
  | PAGE FAULTS
  | SOURCE
  | SWAPS
}
```

`SHOW PROFILE` および `SHOW PROFILES` ステートメントは、現在のセッションの過程で実行されたステートメントのリソース使用状況を示すプロファイリング情報を表示します。

注記

`SHOW PROFILE` および `SHOW PROFILES` ステートメントは非推奨になりました。将来の MySQL リリースで削除される予定です。かわりに `Performance Schema` を使用します。[セクション27.19.1「パフォーマンススキーマを使用したクエリープロファイリング」](#)を参照してください。

プロファイリングを制御するには、`profiling` セッション変数を使用します。この変数のデフォルト値は 0 (OFF) です。`profiling` を 1 または ON に設定してプロファイリングを有効にします:

```
mysql> SET profiling = 1;
```

`SHOW PROFILES` は、サーバーに送信された最新のステートメントのリストを表示します。このリストのサイズは、`profiling_history_size` セッション変数によって制御されます。このデフォルト値は 15 です。最大値は 100 です。この値を 0 に設定すると、実質的にプロファイリングが無効になります。

`SHOW PROFILE` および `SHOW PROFILES` を除くすべてのステートメントがプロファイルされるため、これらのステートメントはプロファイルリストに表示されません。不正な形式のステートメントはプロファイルされます。たとえば、`SHOW PROFILING` は不正なステートメントであり、実行しようとする構文エラーが発生しますが、プロファイリングリストには表示されません。

`SHOW PROFILE` は、1 つのステートメントに関する詳細情報を表示します。`FOR QUERY n` 句を指定しない場合、出力は、直近で実行されたステートメントに関連したものになります。`FOR QUERY n` が含まれている場合、`SHOW PROFILE` は、ステートメント `n` に関する情報を表示します。`n` の値は、`SHOW PROFILES` によって表示される `Query_ID` 値に対応します。

`LIMIT row_count` 句を指定すると、出力を `row_count` 行に制限できます。`LIMIT` が指定されている場合は、`OFFSET offset` を追加することで、行セット全体が `offset` 行分オフセットされた状態で出力を開始できます。

デフォルトでは、`SHOW PROFILE` は `Status` および `Duration` カラムを表示します。この `Status` 値は `SHOW PROCESSLIST` によって表示される `State` 値に似ていますが、一部のステータス値では、この 2 つのステートメントの解釈にわずかな違いがいくつか存在する可能性があります (セクション 8.14 「サーバースレッド (プロセス) 情報の確認」を参照してください)。

オプションの `type` 値を指定すると、次のその他の特定のタイプの情報を表示できます。

- `ALL` は、すべての情報を表示します
- `BLOCK IO` は、ブロック入力および出力操作の数を表示します
- `CONTEXT SWITCHES` は、自発的および非自発的コンテキストスイッチの数を表示します
- `CPU` は、ユーザーとシステムの CPU 使用時間を表示します
- `IPC` は、送受信されたメッセージの数を表示します
- `MEMORY` は現在、実装されていません
- `PAGE FAULTS` は、メジャーおよびマイナーページフォルトの数を表示します
- `SOURCE` は、ソースコードの関数の名前を、その関数が含まれているファイルの名前および行番号とともに表示します
- `SWAPS` は、スワップ数を表示します

プロファイリングは、セッション単位で有効になります。セッションが終了すると、そのプロファイリング情報は失われます。

```
mysql> SELECT @@profiling;
+-----+
| @@profiling |
+-----+
|          0 |
+-----+
1 row in set (0.00 sec)

mysql> SET profiling = 1;
Query OK, 0 rows affected (0.00 sec)

mysql> DROP TABLE IF EXISTS t1;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> CREATE TABLE T1 (id INT);
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> SHOW PROFILES;
+-----+-----+
| Query_ID | Duration | Query |
+-----+-----+
| 0 | 0.000088 | SET PROFILING = 1 |
| 1 | 0.000136 | DROP TABLE IF EXISTS t1 |
| 2 | 0.011947 | CREATE TABLE t1 (id INT) |
+-----+-----+
3 rows in set (0.00 sec)

mysql> SHOW PROFILE;
+-----+-----+
| Status | Duration |
+-----+-----+
| checking permissions | 0.000040 |
| creating table | 0.000056 |
| After create | 0.011363 |
| query end | 0.000375 |
| freeing items | 0.000089 |
| logging slow query | 0.000019 |
| cleaning up | 0.000005 |
+-----+-----+
7 rows in set (0.00 sec)

mysql> SHOW PROFILE FOR QUERY 1;
+-----+-----+
| Status | Duration |
+-----+-----+
| query end | 0.000107 |
| freeing items | 0.000008 |
| logging slow query | 0.000015 |
| cleaning up | 0.000006 |
+-----+-----+
4 rows in set (0.00 sec)

mysql> SHOW PROFILE CPU FOR QUERY 2;
+-----+-----+-----+-----+
| Status | Duration | CPU_user | CPU_system |
+-----+-----+-----+-----+
| checking permissions | 0.000040 | 0.000038 | 0.000002 |
| creating table | 0.000056 | 0.000028 | 0.000028 |
| After create | 0.011363 | 0.000217 | 0.001571 |
| query end | 0.000375 | 0.000013 | 0.000028 |
| freeing items | 0.000089 | 0.000010 | 0.000014 |
| logging slow query | 0.000019 | 0.000009 | 0.000010 |
| cleaning up | 0.000005 | 0.000003 | 0.000002 |
+-----+-----+-----+-----+
7 rows in set (0.00 sec)
```

注記

一部のアーキテクチャーでは、プロファイリングが部分的にしか機能しません。
[getrusage\(\)](#) システムコールに依存する値の場合、このシステムコールをサポートしていない Windows などのシステムでは `NULL` が返されます。さらに、プロファイリングはスレッド単位ではなく、プロセス単位です。つまり、サーバー内の、ユーザー独自のスレッド以外のスレッド上のアクティビティーが、ユーザーに表示されるタイミング情報に影響を与える可能性があります。

プロファイリング情報は、`INFORMATION_SCHEMA` の `PROFILING` テーブルからも入手できます。[セクション 26.24 「INFORMATION_SCHEMA PROFILING テーブル」](#) を参照してください。たとえば、次のクエリーは同等です:

```
SHOW PROFILE FOR QUERY 2;

SELECT STATE, FORMAT(DURATION, 6) AS DURATION
FROM INFORMATION_SCHEMA.PROFILING
WHERE QUERY_ID = 2 ORDER BY SEQ;
```

13.7.7.31 SHOW PROFILES ステートメント

```
SHOW PROFILES
```


`SHOW PROFILES` ステートメントは、`SHOW PROFILE` とともに、現在のセッションの過程で実行されたステートメントのリソース使用状況を示すプロファイリング情報を表示します。詳細は、[セクション13.7.7.30「SHOW PROFILE ステートメント」](#)を参照してください。

注記

`SHOW PROFILE` および `SHOW PROFILES` ステートメントは非推奨になりました。将来の MySQL リリースで削除される予定です。かわりに `Performance Schema` を使用します。[セクション27.19.1「パフォーマンススキーマを使用したクエリープロファイリング」](#)を参照してください。

13.7.7.32 SHOW RELAYLOG EVENTS ステートメント

```
SHOW RELAYLOG EVENTS
[IN 'log_name']
[FROM pos]
[LIMIT [offset,] row_count]
[channel_option]
```

channel_option:
FOR CHANNEL channel

レプリカのリレーログ内のイベントを表示します。'log_name' を指定しない場合は、最初のリレーログが表示されます。このステートメントはソースには影響しません。`SHOW RELAYLOG EVENTS` には、`REPLICATION SLAVE` 権限が必要です。

`LIMIT` 句の構文は、`SELECT` ステートメントの場合と同じです。[セクション13.2.10「SELECT ステートメント」](#)を参照してください。

注記

`LIMIT` 句を指定せずに `SHOW RELAYLOG EVENTS` を発行すると、リレーログの完全な内容 (レプリカによって受信されたデータを変更するすべてのステートメントを含む) がサーバーからクライアントに返されるため、非常に時間のかかるリソース消費プロセスが開始される可能性があります。

オプションの `FOR CHANNEL channel` 句を使用すると、ステートメントが適用されるレプリケーションチャンネルの名前を指定できます。`FOR CHANNEL channel` 句を指定すると、特定のレプリケーションチャンネルにステートメントが適用されます。チャンネルが指定されておらず、追加のチャンネルが存在しない場合、ステートメントはデフォルトチャンネルに適用されます。

複数のレプリケーションチャンネルを使用する場合、`SHOW RELAYLOG EVENTS` ステートメントに `FOR CHANNEL channel` 句を使用して定義されたチャンネルがないと、エラーが生成されます。詳しくは[セクション17.2.2「レプリケーションチャンネル」](#)をご覧ください。

`SHOW RELAYLOG EVENTS` では、リレーログ内のイベントごとに次のフィールドが表示されます:

- `Log_name`
リストされるファイルの名前。
- `Pos`
イベントが発生する位置。
- `Event_type`
イベントタイプを説明する識別子。
- `Server_id`
イベントが発生したサーバーのサーバー ID。
- `End_log_pos`
ソースバイナリログ内のこのイベントの `End_log_pos` の値。

- Info

イベントタイプの詳細情報。この情報の形式は、イベントタイプによって異なります。

圧縮されたトランザクションペイロードの場合、`Transaction_payload_event` は最初に単一のユニットとして印刷された後から解凍され、その内部の各イベントが印刷されます。

`SHOW RELAYLOG EVENTS` からの出力には、ユーザーおよびシステム変数の設定に関連した一部のイベントが含まれていません。リレーログ内のイベントを完全に取得するには、`mysqlbinlog` を使用します。

13.7.7.33 SHOW REPLICAS | SHOW SLAVE HOSTS ステートメント

```
{SHOW REPLICAS | SHOW SLAVE HOSTS}
```

ソースに現在登録されているレプリカのリストを表示します。MySQL 8.0.22 から、`SHOW SLAVE HOSTS` のかわりに `SHOW REPLICAS` を使用します。これは、そのリリースから非推奨になりました。MySQL 8.0.22 より前のリリースでは、`SHOW SLAVE HOSTS` を使用します。`SHOW REPLICAS | SHOW SLAVE HOSTS` には、`REPLICATION SLAVE` 権限が必要です。

`SHOW REPLICAS | SHOW SLAVE HOSTS` は、レプリケーションソースとして機能するサーバーで実行する必要があります。このステートメントは、レプリカとして接続されているサーバーまたは接続されているサーバーに関する情報を、次に示すように、1つの複製サーバーに対応する結果の各行とともに表示します:

```
mysql> SHOW REPLICAS;
+-----+-----+-----+-----+-----+
| Server_id | Host      | Port | Source_id | Replica_UUID |
+-----+-----+-----+-----+-----+
| 10 | iconnect2 | 3306 | 3 | 14cb6624-7f93-11e0-b2c0-c80aa9429562 |
| 21 | athena    | 3306 | 3 | 07af4990-f41f-11df-a566-7ac56fdaf645 |
+-----+-----+-----+-----+-----+
```

- **Server_id**: レプリカサーバーオプションファイルまたは `--server-id=value` を使用したコマンドラインで構成された、レプリカサーバーの一意のサーバー ID。
- **Host**: `--report-host` オプションを使用してレプリカに指定されたレプリカサーバーのホスト名。これは、オペレーティングシステムで構成されているマシン名とは異なる場合があります。
- **User**: `--report-user` オプションを使用してレプリカに指定されたレプリカサーバーのユーザー名。ステートメントの出力にこのコラムが含まれるのは、ソースサーバーが `--show-slave-auth-info` オプションで起動された場合のみです。
- **Password**: `--report-password` オプションを使用してレプリカに指定されたレプリカサーバーのパスワード。ステートメントの出力にこのコラムが含まれるのは、ソースサーバーが `--show-slave-auth-info` オプションで起動された場合のみです。

- **Port**: レプリカで `--report-port` オプションを使用して指定された、レプリカサーバーがリスニングしているソース上のポート。

このコラムのゼロは、レプリカポート (`--report-port`) が設定されていないことを意味します。

- **Source_id**: レプリカサーバーのレプリケート元のソースサーバーの一意のサーバー ID。これは、`SHOW REPLICAS | SHOW SLAVE HOSTS` が実行されるサーバーのサーバー ID であるため、結果の各行に同じ値がリストされます。
- **Replica_UUID**: レプリカで生成され、レプリカ `auto.cnf` ファイルで検出された、このレプリカのグローバルに一意の ID。

13.7.7.34 SHOW SLAVE HOSTS | SHOW REPLICAS ステートメント

```
{SHOW SLAVE HOSTS | SHOW REPLICAS}
```

ソースに現在登録されているレプリカのリストを表示します。MySQL 8.0.22 からは、`SHOW SLAVE HOSTS` は非推奨であり、かわりにエイリアス `SHOW REPLICAS` を使用する必要があります。ステートメントは以前と同様に機能し、ステートメントおよびその出力に使用される用語のみが変更されています。どちらのバージョンのステートメン

とも、使用時に同じステータス変数を更新します。ステートメントの説明は、[SHOW REPLICAS](#) のドキュメントを参照してください。

13.7.7.35 SHOW REPLICA | SLAVE STATUS ステートメント

```
SHOW {REPLICA | SLAVE} STATUS [FOR CHANNEL channel]
```

このステートメントは、レプリカスレッドの必須パラメータに関するステータス情報を提供します。MySQL 8.0.22 から、[SHOW SLAVE STATUS](#) のかわりに [SHOW REPLICA STATUS](#) を使用します。これは、そのリリースから非推奨になりました。MySQL 8.0.22 より前のリリースでは、[SHOW SLAVE STATUS](#) を使用します。このステートメントには、[REPLICATION CLIENT](#) 権限 (または非推奨の [SUPER](#) 権限) が必要です。

[SHOW REPLICA | SLAVE STATUS](#) は非ブロッキングです。[STOP REPLICA | SLAVE](#) と同時に実行すると、[SHOW REPLICA | SLAVE STATUS](#) は、[STOP REPLICA | SLAVE](#) がレプリケーション SQL スレッドまたはレプリケーション I/O スレッド (あるいはその両方) の停止を完了するのを待たずに戻ります。これにより、最新のデータが返されるようにするよりも、[SHOW REPLICA | SLAVE STATUS](#) から即時レスポンスを取得するモニタリングおよびその他のアプリケーションでの使用が重要になります。

`mysql` クライアントを使用してこのステートメントを発行する場合は、セミコロンの代わりに `\G` ステートメントターミネータを使用すると、より読みやすい縦のレイアウトが得られます。

```
mysql> SHOW REPLICA STATUS\G
***** 1. row *****
      Replica_IO_State: Waiting for source to send event
      Source_Host: localhost
      Source_User: repl
      Source_Port: 13000
      Connect_Retry: 60
      Source_Log_File: source-bin.000002
      Read_Source_Log_Pos: 1307
      Relay_Log_File: replica-relay-bin.000003
      Relay_Log_Pos: 1508
      Relay_Source_Log_File: source-bin.000002
      Replica_IO_Running: Yes
      Replica_SQL_Running: Yes
      Replicate_Do_DB:
      Replicate_Ignore_DB:
      Replicate_Do_Table:
      Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
      Replicate_Wild_Ignore_Table:
      Last_Errno: 0
      Last_Error:
      Skip_Counter: 0
      Exec_Source_Log_Pos: 1307
      Relay_Log_Space: 1858
      Until_Condition: None
      Until_Log_File:
      Until_Log_Pos: 0
      Source_SSL_Allowed: No
      Source_SSL_CA_File:
      Source_SSL_CA_Path:
      Source_SSL_Cert:
      Source_SSL_Cipher:
      Source_SSL_Key:
      Seconds_Behind_Source: 0
      Source_SSL_Verify_Server_Cert: No
      Last_IO_Errno: 0
      Last_IO_Error:
      Last_SQL_Errno: 0
      Last_SQL_Error:
      Replicate_Ignore_Server_Ids:
      Source_Server_Id: 1
      Source_UUID: 3e11fa47-71ca-11e1-9e33-c80aa9429562
      Source_Info_File:
      SQL_Delay: 0
      SQL_Remaining_Delay: NULL
      Replica_SQL_Running_State: Reading event from the relay log
      Source_Retry_Count: 10
      Source_Bind:
      Last_IO_Error_Timestamp:
```

```
Last_SQL_Error_Timestamp:
  Source_SSL_Crl:
  Source_SSL_Cripath:
  Retrieved_Gtid_Set: 3e11fa47-71ca-11e1-9e33-c80aa9429562:1-5
  Executed_Gtid_Set: 3e11fa47-71ca-11e1-9e33-c80aa9429562:1-5
  Auto_Position: 1
  Replicate_Rewrite_DB:
  Channel_name:
  Source_TLS_Version: TLSv1.2
  Source_public_key_path: public_key.pem
  Get_source_public_key: 0
  Network_Namespace:
```

パフォーマンススキーマは、レプリケーション情報を公開するテーブルを提供します。これは、[SHOW REPLICATION | SLAVE STATUS](#) ステートメントから使用できる情報に似ていますが、テーブル形式で表されます。詳細は、[セクション27.12.11「パフォーマンススキーマレプリケーションテーブル」](#)を参照してください。

次のリストでは、[SHOW REPLICATION | SLAVE STATUS](#) によって返されるフィールドについて説明します。これらの意味の解釈の詳細は、[セクション17.1.7.1「レプリケーションステータスの確認」](#)を参照してください。

- [Replica_IO_State](#)

レプリカ I/O スレッドの [SHOW PROCESSLIST](#) 出力の `State` フィールドのコピー。これは、スレッドが何をしているかを示します: ソースへの接続、ソースからのイベントの待機、ソースへの再接続などを試行します。可能性のある状態のリストについては、[セクション8.14.5「レプリケーション I/O スレッドの状態」](#)を参照してください。

- [Source_Host](#)

レプリカが接続されているソースホスト。

- [Source_User](#)

ソースへの接続に使用されるアカウントのユーザー名。

- [Source_Port](#)

ソースへの接続に使用されるポート。

- [Connect_Retry](#)

接続再試行の間の秒数 (デフォルトは 60)。これは、[CHANGE REPLICATION SOURCE TO](#) ステートメント (MySQL 8.0.23 の場合) または [CHANGE MASTER TO](#) ステートメント (MySQL 8.0.23 の場合) で設定できます。

- [Source_Log_File](#)

I/O スレッドが現在読み取っているソースバイナリログファイルの名前。

- [Read_Source_Log_Pos](#)

I/O スレッドが読み取った現在のソースバイナリログファイル内の位置。

- [Relay_Log_File](#)

SQL スレッドが現在読み取って実行している元のリレーログファイルの名前。

- [Relay_Log_Pos](#)

現在のリレーログファイル内の SQL スレッドが最後に読み取って実行した位置。

- [Relay_Source_Log_File](#)

SQL スレッドによって実行された最新のイベントを含むソースバイナリログファイルの名前。

- [Replica_IO_Running](#)

レプリケーション I/O スレッドが開始され、ソースに正常に接続されたかどうか。内部的には、このスレッドの状態は次の 3 つの値のいずれかによって表されます。

- `MYSQL_REPLICA_NOT_RUN`. レプリケーション I/O スレッドが実行されていません。この状態では、`Replica_IO_Running` は `No` です。
 - `MYSQL_REPLICA_RUN_NOT_CONNECT`. レプリケーション I/O スレッドは実行中ですが、レプリケーションソースに接続されていません。この状態では、`Replica_IO_Running` は `Connecting` です。
 - `MYSQL_REPLICA_RUN_CONNECT`. レプリケーション I/O スレッドは実行中で、レプリケーションソースに接続されています。この状態では、`Replica_IO_Running` は `Yes` です。
 - `Replica_SQL_Running`
レプリケーション SQL スレッドが開始されているかどうか。
 - `Replicate_Do_DB`、`Replicate_Ignore_DB`
`--replicate-do-db` および `--replicate-ignore-db` オプションまたは `CHANGE REPLICATION FILTER` ステートメントで指定されたデータベースの名前。 `FOR CHANNEL` 句が使用された場合は、チャンネル固有のレプリケーションフィルタが表示されます。それ以外の場合は、すべてのレプリケーションチャンネルのレプリケーションフィルタが表示されます。
 - `Replicate_Do_Table`、`Replicate_Ignore_Table`、`Replicate_Wild_Do_Table`、`Replicate_Wild_Ignore_Table`
`--replicate-do-table`、`--replicate-ignore-table`、`--replicate-wild-do-table` オプション、`--replicate-wild-ignore-table` オプションまたは `CHANGE REPLICATION FILTER` ステートメントで指定されたテーブルの名前。 `FOR CHANNEL` 句が使用された場合は、チャンネル固有のレプリケーションフィルタが表示されます。それ以外の場合は、すべてのレプリケーションチャンネルのレプリケーションフィルタが表示されます。
 - `Last_Errno`、`Last_Error`
これらのカラムは、`Last_SQL_Errno` および `Last_SQL_Error` のエイリアスです。
`RESET MASTER` または `RESET REPLICATION | SLAVE` を発行すると、これらのカラムに表示される値がリセットされます。
- 注記**
- レプリケーション SQL スレッドは、エラーを受信すると、最初にエラーを報告してから SQL スレッドを停止します。これは、`Replica_SQL_Running` に `Yes` が表示されている場合でも、`SHOW REPLICATION | SLAVE STATUS` で `Last_SQL_Errno` にゼロ以外の値が表示される短い期間があることを意味します。
- `Skip_Counter`
`sql_slave_skip_counter` システム変数の現在の値。 `SET GLOBAL sql_slave_skip_counter Statement` を参照してください。
 - `Exec_Source_Log_Pos`
レプリケーション SQL スレッドが読み取って実行した現在のソースバイナリログファイル内の位置で、次に処理されるトランザクションまたはイベントの開始をマークします。この値は、新しいレプリカがこの時点から読み取るように、既存のレプリカから新しいレプリカを開始するときに、(MySQL 8.0.23 の) `CHANGE REPLICATION SOURCE TO` ステートメントの `SOURCE_LOG_POS` オプションまたは (MySQL 8.0.23 の前の) `CHANGE MASTER TO` ステートメントの `MASTER_LOG_POS` オプションとともに使用できます。ソースバイナリログ内の (`Relay_Source_Log_File`、`Exec_Source_Log_Pos`) によって指定された座標は、リレーログ内の (`Relay_Log_File`、`Relay_Log_Pos`) によって指定された座標に対応します。

実行されたリレーログからのトランザクションの順序に一貫性がないと、この値が「最低水位標」になる可能性があります。つまり、位置の前に表示されるトランザクションはコミットされていることが保証されますが、位置の後のトランザクションはコミットされているかどうかは保証されません。これらのギャップを修正する必要がある場合は、`START REPLICATION | SLAVE UNTIL SQL_AFTER_MTS_GAPS` を使用します。詳しくは [セクション 17.5.1.34 「レプリケーションとトランザクションの非一貫性」](#) をご覧ください。
 - `Relay_Log_Space`

既存のすべてのリレーログファイルの合計サイズ。

- [Until_Condition](#)、[Until_Log_File](#)、[Until_Log_Pos](#)

[START REPLICA | SLAVE](#) ステートメントの [UNTIL](#) 句で指定された値。

[Until_Condition](#) の値は次のとおりです。

- [UNTIL](#) 句が指定されなかった場合は [None](#)
- レプリカがソースバイナリログ内の指定された位置まで読み取っている場合は [Source](#)。
- レプリカがリレーログ内の指定された位置まで読み取っている場合は [Relay](#)。
- [gtid_set](#) に GTID がリストされている最初のトランザクションに到達するまで、レプリケーション SQL スレッドがトランザクションを処理している場合は [SQL_BEFORE_GTIDS](#)。
- [gtid_set](#) の最後のトランザクションが両方のスレッドによって処理されるまで、レプリケーションスレッドがすべてのトランザクションを処理している場合は [SQL_AFTER_GTIDS](#)。
- マルチスレッドレプリカ SQL スレッドがリレーログにギャップがなくなるまで実行されている場合は [SQL_AFTER_MTS_GAPS](#)。

[Until_Log_File](#) および [Until_Log_Pos](#) は、レプリケーション SQL スレッドが実行を停止する座標を定義するログファイルの名前と位置を示します。

[UNTIL](#) 句の詳細は、[セクション13.4.2.8「START SLAVE | REPLICA ステートメント」](#)を参照してください。

- [Source_SSL_Allowed](#)、[Source_SSL_CA_File](#)、[Source_SSL_CA_Path](#)、[Source_SSL_Cert](#)、[Source_SSL_Cipher](#)、[Source_SSL_CRL_File](#)、[Source_SSL_CRL_Path](#)、[Source_SSL_Key](#)、[Source_SSL_Verify_Server_Cert](#)

これらのフィールドには、レプリカがソースに接続するために使用する SSL パラメータが表示されます (存在する場合)。

[Source_SSL_Allowed](#) には次の値があります:

- ソースへの SSL 接続が許可されている場合は [Yes](#)。
- ソースへの SSL 接続が許可されていない場合は [No](#)。
- SSL 接続が許可されているが、レプリカサーバーで SSL サポートが有効になっていない場合は [Ignored](#)。

その他の SSL 関連フィールドの値は、[CHANGE REPLICATION SOURCE TO](#) ステートメント (MySQL 8.0.23 の場合) の [SOURCE_SSL_*](#) オプションまたは [CHANGE MASTER TO](#) ステートメント (MySQL 8.0.23 の場合) の [MASTER_SSL_*](#) オプションの値に対応します。[セクション13.4.2.1「CHANGE MASTER TO ステートメント」](#)を参照してください。

- [Seconds_Behind_Source](#)

このフィールドは、レプリカがどのように「late」であるかを示します:

- レプリカが更新をアクティブに処理している場合、このフィールドには、レプリカの現在のタイムスタンプと、レプリカで現在処理されているイベントのソースに記録されている元のタイムスタンプの差異が表示されます。
- レプリカで現在処理されているイベントがない場合、この値は 0 です。

基本的に、このフィールドはレプリケーション SQL スレッドとレプリケーション I/O スレッドの間の時間差を秒単位で測定します。ソースとレプリカ間のネットワーク接続が高速である場合、レプリケーション I/O スレッドはソースに非常に近いので、このフィールドはレプリケーション SQL スレッドがソースと比較される遅延の概算値になります。ネットワークが低速の場合、これは適切な概算ではありません。レプリケーション SQL スレッドは、多くの場合、低速なレプリケーション I/O スレッドで捕捉される可能性があるため、レプリケーション I/O スレッドがソースと比較して遅延している場合でも、[Seconds_Behind_Source](#) は多くの場合 0 の値を表示します。つまり、このカラムは高速ネットワークの場合にのみ有効です。

この時間差計算は、レプリカ I/O スレッドの起動時に計算された差がそれ以降も一定である場合に、ソースとレプリカのクロック時間が同一でない場合でも機能します。NTP の更新を含むすべての変更により、`Seconds_Behind_Source` の計算の信頼性が低下する可能性があるクロックスキューが発生する可能性があります。

MySQL 8.0 では、レプリケーション SQL スレッドが実行されていない場合、または SQL スレッドがすべてのリレーログを消費し、レプリケーション I/O スレッドが実行されていない場合、このフィールドは `NULL` (未定義または不明) になります。(旧バージョンの MySQL では、レプリケーション SQL スレッドまたはレプリケーション I/O スレッドが実行されていなかったか、ソースに接続されていなかった場合、このフィールドは `NULL` でした。) レプリケーション I/O スレッドは実行されているが、リレーログがいっぱいになった場合、`Seconds_Behind_Source` は 0 に設定されます。

`Seconds_Behind_Source` の値は、イベントに格納されているタイムスタンプに基づき、レプリケーションによって保持されます。つまり、ソース M1 自体が M0 のレプリカである場合、M0 バイナリログから発生した M1 バイナリログのすべてのイベントには、そのイベントの M0 タイムスタンプが設定されます。これにより、MySQL は `TIMESTAMP` を正常にレプリケートできます。ただし、`Seconds_Behind_Source` の問題は、M1 がクライアントから直接更新を受信した場合、M1 からの最後のイベントが M0 から発生し、場合によっては M1 での直接更新の結果であるため、`Seconds_Behind_Source` 値がランダムに変動することです。

マルチスレッドのレプリカを使用する場合、この値は `Exec_Source_Log_Pos` に基づいているため、最後にコミットされたトランザクションの位置が反映されない可能性があることに注意してください。

- `Last_IO_Errno`、`Last_IO_Error`

レプリケーション I/O スレッドの停止の原因となった最新のエラーのエラー番号およびエラーメッセージ。0 のエラー番号および空の文字列のメッセージは、「エラーなし」を示します。`Last_IO_Error` 値が空でない場合、エラー値はレプリカエラーログにも表示されます。

I/O エラー情報には、最新の I/O スレッドエラーが発生した日時を示すタイムスタンプが含まれます。このタイムスタンプは `YYMMDD hh:mm:ss` の形式を使用し、`Last_IO_Error_Timestamp` カラムに表示されます。

`RESET MASTER` または `RESET REPLICAS | SLAVE` を発行すると、これらのカラムに表示される値がリセットされます。

- `Last_SQL_Errno`、`Last_SQL_Error`

レプリケーション SQL スレッドの停止の原因となった最新のエラーのエラー番号およびエラーメッセージ。0 のエラー番号および空の文字列のメッセージは、「エラーなし」を示します。`Last_SQL_Error` 値が空でない場合、エラー値はレプリカエラーログにも表示されます。

レプリカがマルチスレッド化されている場合、レプリケーション SQL スレッドはワーカースレッドのコーディネータです。この場合、`Last_SQL_Error` フィールドには、パフォーマンススキーマ `replication_applier_status_by_coordinator` テーブルの `Last_Error_Message` カラムに表示される内容が正確に表示されます。このフィールド値は、各ワーカースレッドステータスを示す `replication_applier_status_by_worker` テーブルに表示される他のワーカースレッドでさらに障害が発生する可能性があることを示すように変更されます。そのテーブルが使用できない場合は、レプリカエラーログを使用できます。ログまたは `replication_applier_status_by_worker` テーブルを使用して、`SHOW REPLICAS | SLAVE STATUS` またはコーディネータテーブルに表示される障害の詳細を確認する必要があります。

SQL エラー情報には、最新の SQL スレッドエラーが発生した日時を示すタイムスタンプが含まれます。このタイムスタンプは `YYMMDD hh:mm:ss` の形式を使用し、`Last_SQL_Error_Timestamp` カラムに表示されます。

`RESET MASTER` または `RESET REPLICAS | SLAVE` を発行すると、これらのカラムに表示される値がリセットされます。

MySQL 8.0 では、`Last_SQL_Errno` および `Last_SQL_Error` カラムに表示されるすべてのエラーコードおよびメッセージは、`Server Error Message Reference` にリストされているエラー値に対応しています。これは、以前のバージョンでは常に当てはまるわけではありませんでした。(Bug #11760365、Bug #52768)

- [Replicate_Ignore_Server_Ids](#)

レプリカがこれらのサーバーからのイベントを無視するように、[CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO](#) ステートメントの [IGNORE_SERVER_IDS](#) オプションを使用して指定されたサーバー ID。このオプションは、いずれかのサーバーが削除されたときに循環またはその他のマルチソースレプリケーション設定で使用されます。この方法でサーバー ID が設定されている場合は、1 つ以上の数値のカンマ区切りリストが表示されます。サーバー ID が設定されていない場合、このフィールドは空白です。

注記

[slave_master_info](#) テーブルの [Ignored_server_ids](#) 値には、無視されるサーバー ID も空白区切りリストとして表示され、無視されるサーバー ID の合計数が先頭に付きます。たとえば、サーバー ID が 2、6 または 9 のソースを無視するようレプリカに指示するために、[IGNORE_SERVER_IDS = \(2,6,9\)](#) オプションを含む [CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO](#) ステートメントが発行された場合、その情報は次のように表示されます:

```
Replicate_Ignore_Server_Ids: 2, 6, 9
```

```
Ignored_server_ids: 3, 2, 6, 9
```

[Replicate_Ignore_Server_Ids](#) のフィルタリングは SQL スレッドではなく、I/O スレッドによって実行されます。つまり、フィルタで除外されるイベントはリレーログに書き込まれません。これは、SQL スレッドに適用される `--replicate-do-table` などのサーバーオプションによって実行されるフィルタリングアクションとは異なります。

注記

MySQL 8.0 から、[IGNORE_SERVER_IDS](#) で既存のサーバー ID が設定されているチャンネルがある場合に [SET GTID_MODE=ON](#) が発行されると、非推奨の警告が発行されます。GTID ベースのレプリケーションを開始する前に、[SHOW REPLICA | SLAVE STATUS](#) を使用して、関係するサーバー上の無視されたすべてのサーバー ID リストを確認してクリアします。リストをクリアするには、空のリストで [IGNORE_SERVER_IDS](#) オプションを含む [CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO](#) ステートメントを発行します。

- [Source_Server_Id](#)

ソースからの `server_id` 値。

- [Source_UUID](#)

ソースからの `server_uuid` 値。

- [Source_Info_File](#)

`master.info` ファイルの場所。その使用は非推奨になりました。デフォルトでは、MySQL 8.0 から、レプリカ接続メタデータリポジトリのかわりにテーブルが使用されます。

- [SQL_Delay](#)

レプリカがソースを遅らせる必要がある秒数。

- [SQL_Remaining_Delay](#)

[Replica_SQL_Running_State](#) が `Waiting until MASTER_DELAY seconds after source executed event` の場合、このフィールドには残りの遅延秒数が含まれます。ほかのときは、このフィールドは `NULL` です。

- [Replica_SQL_Running_State](#)

SQL スレッドの状態 ([Replica_IO_State](#) に類似)。この値は、[SHOW PROCESSLIST](#) で表示される SQL スレッドの `State` 値と同じです。[セクション 8.14.6 「レプリケーション SQL スレッドの状態」](#) には、考えられる状態のリストが表示されます。

- [Source_Retry_Count](#)

接続が失われた場合にレプリカがソースへの再接続を試行できる回数。 This value can be set using the `SOURCE_RETRY_COUNT` | `MASTER_RETRY_COUNT` option of the `CHANGE REPLICATION SOURCE TO` statement (from MySQL 8.0.23) or `CHANGE MASTER TO` statement (before MySQL 8.0.23), or the older `--master-retry-count` server option (still supported for backward compatibility).

- `Source_Bind`

レプリカがバインドされているネットワークインタフェース (ある場合)。これは、`CHANGE REPLICATION SOURCE TO` ステートメント (MySQL 8.0.23) または `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 より前) の `SOURCE_BIND` | `MASTER_BIND` オプションを使用して設定します。

- `Last_IO_Error_Timestamp`

最新の I/O エラーがいつ発生したかを示す `YYMMDD hh:mm:ss` 形式のタイムスタンプ。

- `Last_SQL_Error_Timestamp`

最新の SQL エラーがいつ発生したかを示す `YYMMDD hh:mm:ss` 形式のタイムスタンプ。

- `Retrieved_Gtid_Set`

このレプリカによって受信されたすべてのトランザクションに対応するグローバルトランザクション ID のセット。GTID が使用されていない場合は空です。詳しくは [GTID セット](#) をご覧ください。

これは、リレーログ内に存在するか、またはこれまでに存在したすべての GTID のセットです。各 GTID は、`Gtid_log_event` が受信されるとすぐに追加されます。そのため、このセットには、部分的に転送されたトランザクションの GTID も含まれる場合があります。

`RESET REPLICA` | `SLAVE` または `CHANGE REPLICATION SOURCE TO` | `CHANGE MASTER TO` の実行、あるいは `--relay-log-recovery` オプションの影響が原因ですべてのリレーログが失われると、そのセットはクリアされます。`relay_log_purge = 1` のときは、最新のリレーログが常に保持されるため、このセットはクリアされません。

- `Executed_Gtid_Set`

バイナリログに書き込まれたグローバルトランザクション ID のセット。これは、このサーバー上のグローバル `gtid_executed` システム変数の値、およびこのサーバー上の `SHOW MASTER STATUS` の出力における `Executed_Gtid_Set` の値と同じです。GTID が使用されていない場合は空です。詳しくは [GTID セット](#) をご覧ください。

- `Auto_Position`

GTID 自動位置決めがチャンネルに使用されている場合は 1、それ以外の場合は 0。

- `Replicate_Rewrite_DB`

`Replicate_Rewrite_DB` 値には、指定されたレプリケーションフィルタリングルールが表示されます。たとえば、次のレプリケーションフィルタリングルールが設定されているとします:

```
CHANGE REPLICATION FILTER REPLICATE_REWRITE_DB=((db1,db2), (db3,db4));
```

`Replicate_Rewrite_DB` 値は次のように表示されます:

```
Replicate_Rewrite_DB: (db1,db2),(db3,db4)
```

詳細は、[セクション13.4.2.2 「CHANGE REPLICATION FILTER ステートメント」](#) を参照してください。

- `Channel_name`

表示されるレプリケーションチャンネル。常にデフォルトのレプリケーションチャンネルがあり、さらにレプリケーションチャンネルを追加できます。詳しくは [セクション17.2.2 「レプリケーションチャンネル」](#) をご覧ください。

- `Master_TLS_Version`

ソースで使用される TLS バージョン。TLS バージョン情報については、[セクション6.3.2 「暗号化された接続 TLS プロトコルおよび暗号」](#) を参照してください。

- [Source_public_key_path](#)

RSA キーペアベースのパスワード交換のソースに必要な公開キーのレプリカ側コピーを含むファイルへのパス名。ファイルは PEM 形式である必要があります。このカラムは、[sha256_password](#) または [caching_sha2_password](#) 認証プラグインで認証されるレプリカに適用されます。

[Source_public_key_path](#) が指定され、有効な公開キーファイルが指定されている場合は、[Get_source_public_key](#) よりも優先されます。

- [Get_source_public_key](#)

RSA キーペアベースのパスワード交換に必要な公開キーをソースからリクエストするかどうか。このカラムは、[caching_sha2_password](#) 認証プラグインで認証されるレプリカに適用されます。そのプラグインの場合、ソースは要求されないかぎり公開鍵を送信しません。

[Source_public_key_path](#) が指定され、有効な公開キーファイルが指定されている場合は、[Get_source_public_key](#) よりも優先されます。

- [Network_Namespace](#)

ネットワークネームスペース名。接続でデフォルト (グローバル) ネームスペースを使用する場合は空です。ネットワークネームスペースの詳細は、[セクション5.1.14「ネットワークネームスペースのサポート」](#) を参照してください。このカラムは、MySQL 8.0.22 で追加されました。

13.7.7.36 SHOW SLAVE | REPLICAS STATUS ステートメント

```
SHOW {SLAVE | REPLICAS} STATUS [FOR CHANNEL channel]
```

このステートメントは、レプリカスレッドの必須パラメータに関するステータス情報を提供します。MySQL 8.0.22 からは、[SHOW SLAVE STATUS](#) は非推奨であり、かわりにエイリアス [SHOW REPLICAS STATUS](#) を使用する必要があります。ステートメントは以前と同様に機能し、ステートメントおよびその出力に使用される用語のみが変更されています。どちらのバージョンのステートメントも、使用時に同じステータス変数を更新します。ステートメントの説明は、[SHOW REPLICAS STATUS](#) のドキュメントを参照してください。

13.7.7.37 SHOW STATUS ステートメント

```
SHOW [GLOBAL | SESSION] STATUS  
[LIKE 'pattern' | WHERE expr]
```

[SHOW STATUS](#) は、サーバーのステータス情報を提供します ([セクション5.1.10「サーバーステータス変数」](#) を参照)。このステートメントにはどの権限も必要ありません。これには、サーバーに接続できることのみが必要です。

ステータス変数情報は、次のソースからも入手できます:

- パフォーマンススキーマ tables. [セクション27.12.15「パフォーマンススキーマのステータス変数のテーブル」](#) を参照してください。
- `mysqladmin extended-status` コマンド。 [セクション4.5.2「mysqladmin — A MySQL Server 管理プログラム」](#) を参照してください。

[SHOW STATUS](#) の場合、`LIKE` 句 (存在する場合) は一致させる変数名を示します。 [セクション26.55「SHOW ステートメントの拡張」](#) で説明されているように、より一般的な条件を使用して行を選択するために `WHERE` 句を指定できます。

[SHOW STATUS](#) は、オプションの `GLOBAL` または `SESSION` 変数スコープ修飾子を受け入れます:

- `GLOBAL` 修飾子を使用すると、ステートメントはグローバルステータス値を表示します。グローバルステータス変数は、サーバー自体の一部の側面 (`Aborted_connects` など) のステータス、または MySQL へのすべての接続の集計ステータス (`Bytes_received` や `Bytes_sent` など) を表す場合があります。変数にグローバル値がない場合は、セッション値が表示されます。
- `SESSION` 修飾子を使用すると、ステートメントは現在の接続のステータス変数値を表示します。変数にセッション値がない場合は、グローバル値が表示されます。 `LOCAL` は `SESSION` のシノニムです。

- 修飾子が存在しない場合、デフォルトは `SESSION` です。

各ステータス変数のスコープは、[セクション5.1.10「サーバーステータス変数」](#)に示されています。

`SHOW STATUS` ステートメントを呼び出すたびに内部一時テーブルが使用され、グローバルの `Created_tmp_tables` 値が増加します。

部分的な出力を次に示します。名前と値のリストは、サーバーによって異なる場合があります。各変数の意味は、[セクション5.1.10「サーバーステータス変数」](#)に示されています。

```
mysql> SHOW STATUS;
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Aborted_clients | 0 |
| Aborted_connects | 0 |
| Bytes_received | 155372598 |
| Bytes_sent | 1176560426 |
| Connections | 30023 |
| Created_tmp_disk_tables | 0 |
| Created_tmp_tables | 8340 |
| Created_tmp_files | 60 |
| ... |
| Open_tables | 1 |
| Open_files | 2 |
| Open_streams | 0 |
| Opened_tables | 44600 |
| Questions | 2026873 |
| ... |
| Table_locks_immediate | 1920382 |
| Table_locks_waited | 0 |
| Threads_cached | 0 |
| Threads_created | 30022 |
| Threads_connected | 1 |
| Threads_running | 1 |
| Uptime | 80380 |
+-----+-----+
```

`LIKE` 句を指定すると、このステートメントは、そのパターンに一致する名前を持つ変数の行のみを表示します。

```
mysql> SHOW STATUS LIKE 'Key%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Key_blocks_used | 14955 |
| Key_read_requests | 96854827 |
| Key_reads | 162040 |
| Key_write_requests | 7589728 |
| Key_writes | 3813196 |
+-----+-----+
```

13.7.7.38 SHOW TABLE STATUS ステートメント

```
SHOW TABLE STATUS
[[FROM | IN] db_name]
[LIKE 'pattern' | WHERE expr]
```

`SHOW TABLE STATUS` は `SHOW TABLES` のように機能しますが、`TEMPORARY` 以外の各テーブルに関する多くの情報を提供します。このリストはまた、`mysqlshow --status db_name` コマンドを使用して取得することもできます。`LIKE` 句 (存在する場合) は、どのテーブル名と照合するかを示します。[セクション26.55「SHOW ステートメントの拡張」](#)で説明されているように、`WHERE` 句を指定すると、より一般的な条件を使用して行を選択できます。

このステートメントはまた、ビューに関する情報も表示します。

`SHOW TABLE STATUS` 出力には、次のカラムがあります:

- **Name**

テーブルの名前。

- **Engine**

このテーブルのストレージエンジン。第15章「InnoDB ストレージエンジン」および第16章「代替ストレージエンジン」を参照してください。

パーティション化されたテーブルの場合、Engine には、すべてのパーティションで使用されるストレージエンジンの名前が表示されます。

- **バージョン**

このカラムは未使用です。MySQL 8.0 で .frm ファイルを削除すると、このカラムには、MySQL 5.7 で最後に使用された .frm ファイルバージョンである 10 のハードコードされた値がレポートされるようになりました。

- **Row_format**

行ストレージフォーマット (Fixed、Dynamic、Compressed、Redundant、Compact)。MyISAM テーブルの場合、Dynamic は、myisamchk -dv が Packed としてレポートする内容に対応します。

- **Rows**

行数。MyISAM などの一部のストレージエンジンは、正確な数を格納します。InnoDB などのほかのストレージエンジンの場合、この値は概算であり、実際の値と 40% から 50% まで異なる可能性があります。このような場合、正確な数を取得するには SELECT COUNT(*) を使用します。

INFORMATION_SCHEMA テーブルの場合、Rows 値は NULL です。

InnoDB テーブルの場合、行カウントは SQL 最適化で 사용되는単なる概算です。(InnoDB テーブルがパーティション化されている場合も、これは当てはまります。)

- **Avg_row_length**

平均行長。

- **Data_length**

MyISAM の場合、Data_length はデータファイルの長さ (バイト単位) です。

InnoDB の場合、Data_length は、クラスタ化されたインデックスに割り当てられるおおよその容量 (バイト単位) です。具体的には、クラスタ化されたインデックスサイズ (ページ単位) に InnoDB ページサイズを乗算したものです。

その他のストレージエンジンについては、このセクションの最後にある注を参照してください。

- **Max_data_length**

MyISAM の場合、Max_data_length はデータファイルの最大長です。これは、このテーブル内に格納できるデータの合計バイト数です (使用されるデータポインタサイズが指定された場合)。

InnoDB では未使用です。

その他のストレージエンジンについては、このセクションの最後にある注を参照してください。

- **Index_length**

MyISAM の場合、Index_length はインデックスファイルの長さ (バイト単位) です。

InnoDB の場合、Index_length は、クラスタ化されていないインデックスに割り当てられる領域の概算量 (バイト単位) です。具体的には、クラスタ化されていないインデックスサイズの合計 (ページ数) に InnoDB ページサイズを乗算した値です。

その他のストレージエンジンについては、このセクションの最後にある注を参照してください。

- **Data_free**

割り当てられているが、使用されていないバイト数。

InnoDB テーブルは、このテーブルが属するテーブルスペースの空き領域をレポートします。共有テーブルスペース内に存在するテーブルの場合、これはその共有テーブルスペースの空き領域です。複数のテーブルスペースを使用していて、このテーブルに独自のテーブルスペースがある場合は、そのテーブルのみの空き領域になります。空き領域とは、完全な空きエクステントから安全上のマージンを引いたバイト数を示します。空き領域が 0 として表示されている場合でも、新しいエクステントを割り当てる必要がないかぎり、行を挿入できる可能性があります。

NDB Cluster の場合、`Data_free` は、ディスク上の「ディスクデータ」テーブルまたはフラグメント用にディスクに割り当てられたが使用されていない領域を表示します。(メモリー内データリソース使用率は、`Data_length` カラムによってレポートされます。)

パーティション化されたテーブルの場合、この値は推定値にすぎず、絶対的に正しいとはかぎりません。このような場合にこの情報を取得するより正確な方法は、次の例に示すように、`INFORMATION_SCHEMA PARTITIONS` テーブルをクエリーすることです:

```
SELECT SUM(DATA_FREE)
FROM INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_SCHEMA = 'mydb'
AND TABLE_NAME = 'mytable';
```

詳細は、[セクション26.21「INFORMATION_SCHEMA PARTITIONS テーブル」](#)を参照してください。

- [Auto_increment](#)

次の `AUTO_INCREMENT` 値。

- [Create_time](#)

いつテーブルが作成されたか。

- [Update_time](#)

いつデータファイルが最後に更新されたか。一部のストレージエンジンでは、この値は `NULL` です。たとえば、InnoDB はそのシステムテーブルスペース内に複数のテーブルを格納するため、データファイルのタイムスタンプは適用されません。各 InnoDB テーブルが個別の `.ibd` ファイル内に存在する `file-per-table` モードの場合でも、[変更バッファリング](#)によってデータファイルへの書き込みが遅延される可能性があるため、ファイルの変更時間は最後の挿入、更新、または削除の時間とは異なります。MyISAM の場合、データファイルのタイムスタンプが使用されますが、Windows ではタイムスタンプは更新によって更新されないため、値は正確ではありません。

`Update_time` には、パーティション化されていない InnoDB テーブルに対して最後に実行された `UPDATE`、`INSERT` または `DELETE` のタイムスタンプ値が表示されます。MVCC の場合、タイムスタンプ値は最終更新時間とみなされる `COMMIT` 時間を反映します。タイムスタンプは、サーバーの再起動時、またはテーブルが InnoDB データディクショナリキャッシュから削除されたときに永続化されません。

- [Check_time](#)

いつテーブルが最後にチェックされたか。すべてのストレージエンジンがこの時間を更新するわけではありません。この場合、値は常に `NULL` です。

パーティション化された InnoDB テーブルの場合、`Check_time` は常に `NULL` です。

- [Collation](#)

テーブルのデフォルトの照合。出力にはテーブルのデフォルトの文字セットは明示的にリストされませんが、照合順序名は文字セット名で始まります。

- [Checksum](#)

ライブチェックサム値 (存在する場合)。

- [Create_options](#)

`CREATE TABLE` で使用される追加のオプション。

`Create_options` には、パーティションテーブルの `partitioned` が表示されます。

MySQL 8.0.16 より前の [Create_options](#) では、file-per-table テーブルスペースに作成されたテーブルに指定された [ENCRYPTION](#) 句が表示されます。MySQL 8.0.16 では、テーブルが暗号化されている場合、または指定された暗号化がスキーマ暗号化と異なる場合、file-per-table テーブルスペースの暗号化句が表示されます。暗号化句は、一般テーブルスペースに作成されたテーブルには表示されません。暗号化された file-per-table および一般的なテーブルスペースを識別するには、[INNOODB_TABLESPACES ENCRYPTION](#) カラムをクエリーします。

[strict mode](#) を無効にしてテーブルを作成する場合、指定した行フォーマットがサポートされていないと、ストレージエンジンのデフォルトの行フォーマットが使用されます。テーブルの実際の行形式は、[Row_format](#) カラムにレポートされます。[Create_options](#) には、[CREATE TABLE](#) ステートメントで指定された行形式が表示されます。

テーブルのストレージエンジンを変更する場合、新しいストレージエンジンに適用できないテーブルオプションはテーブル定義に保持され、必要に応じて、以前に定義されたオプションを持つテーブルを元のストレージエンジンに戻すことができます。[Create_options](#) には、保持されているオプションが表示される場合があります。

- [Comment](#)

このテーブルを作成するときに使用されたコメント (または、MySQL がテーブル情報にアクセスできなかった理由に関する情報)。

メモ

- InnoDB テーブルの場合、[SHOW TABLE STATUS](#) では、テーブルで予約されている物理サイズを除き、正確な統計は提供されません。行カウントは、単に SQL 最適化で使用される概算見積もりです。
- NDB テーブルの場合、このステートメントの出力は [Avg_row_length](#) および [Data_length](#) カラムの適切な値を示しますが、例外として [BLOB](#) カラムは考慮に入れられません。
- NDB テーブルの場合、[Data_length](#) にはメインメモリーに格納されているデータのみが含まれます。[Max_data_length](#) および [Data_free](#) カラムはディスクデータに適用されます。
- 「NDB Cluster ディスクデータの場合」テーブル、[Max_data_length](#) には、「ディスクデータ」テーブルまたはフラグメントのディスク部分に割り当てられた領域が表示されます。(メモリー内データリソース使用率は、[Data_length](#) カラムによってレポートされます。)
- MEMORY テーブルの場合、[Data_length](#)、[Max_data_length](#)、および [Index_length](#) 値はほぼ、割り当てられているメモリーの実際の量を表します。割り当てアルゴリズムは、割り当て操作の数を減らすために、大量のメモリーを確保します。
- ビューの場合、[Name](#) がビュー名を示し、[Create_time](#) が作成時間を示し、[Comment](#) が [VIEW](#) を示すことを除き、[SHOW TABLE STATUS](#) によって表示されるほとんどのカラムは 0 または [NULL](#) です。

テーブル情報は、[INFORMATION_SCHEMA TABLES](#) テーブルからも入手できます。[セクション 26.38 「INFORMATION_SCHEMA TABLES テーブル」](#) を参照してください。

13.7.7.39 SHOW TABLES ステートメント

```
SHOW [EXTENDED] [FULL] TABLES
  [(FROM | IN) db_name]
  [(LIKE 'pattern' | WHERE expr)]
```

[SHOW TABLES](#) は、特定のデータベース内の [TEMPORARY](#) 以外のテーブルを一覧表示します。このリストはまた、[mysqlshow db_name](#) コマンドを使用して取得することもできます。[LIKE](#) 句 (存在する場合) は、どのテーブル名と照合するかを示します。[セクション 26.55 「SHOW ステートメントの拡張」](#) で説明されているように、[WHERE](#) 句を指定すると、より一般的な条件を使用して行を選択できます。

[LIKE](#) 句によって実行される照合は、[lower_case_table_names](#) システム変数の設定に依存します。

オプションの [EXTENDED](#) 修飾子を使用すると、失敗した [ALTER TABLE](#) ステートメントによって作成された非表示のテーブルが [SHOW TABLES](#) にリストされます。これらの一時テーブルは、[#sql](#) で始まる名前を持ち、[DROP TABLE](#) を使用して削除できます。

このステートメントはまた、このデータベース内のビューもすべて一覧表示します。オプションの [FULL](#) 修飾子を使用すると、[SHOW TABLES](#) では、テーブルの場合は [BASE TABLE](#)、ビューの場合は

VIEW、INFORMATION_SCHEMA テーブルの場合は SYSTEM VIEW の値を含む 2 番目の出力カラムが表示されません。

ベーステーブルまたはビューに対する権限を持っていない場合、そのテーブルまたはビューは SHOW TABLES または mysqlshow db_name の出力に表示されません。

テーブル情報は、INFORMATION_SCHEMA TABLES テーブルからも入手できます。 [セクション 26.38 「INFORMATION_SCHEMA TABLES テーブル」](#) を参照してください。

13.7.7.40 SHOW TRIGGERS ステートメント

```
SHOW TRIGGERS
  [(FROM | IN) db_name]
  [LIKE 'pattern' | WHERE expr]
```

SHOW TRIGGERS は、データベース (FROM 句が指定されていないかぎり、デフォルトデータベース) 内のテーブルに対して現在定義されているトリガーを一覧表示します。このステートメントは、ユーザーが TRIGGER 権限を持っているデータベースとテーブルに対してのみ結果を返します。LIKE 句が存在する場合は、一致させるテーブル名 (トリガー名ではない) を指定し、そのテーブルのトリガーをステートメントに表示させます。 [セクション 26.55 「SHOW ステートメントの拡張」](#) で説明されているように、WHERE 句を指定すると、より一般的な条件を使用して行を選択できます。

[セクション 25.3 「トリガーの使用」](#) で定義された ins_sum トリガーの場合、SHOW TRIGGERS の出力は次のようになります:

```
mysql> SHOW TRIGGERS LIKE 'acc%\G
***** 1. row *****
      Trigger: ins_sum
      Event: INSERT
      Table: account
      Statement: SET @sum = @sum + NEW.amount
      Timing: BEFORE
      Created: 2018-08-08 10:10:12.61
      sql_mode: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,
                NO_ZERO_IN_DATE,NO_ZERO_DATE,
                ERROR_FOR_DIVISION_BY_ZERO,
                NO_ENGINE_SUBSTITUTION
      Definer: me@localhost
character_set_client: utf8mb4
collation_connection: utf8mb4_0900_ai_ci
Database Collation: utf8mb4_0900_ai_ci
```

SHOW TRIGGERS 出力には、次のカラムがあります:

- **Trigger**
トリガーの名前。
- **イベント**
トリガーイベント。これは、トリガーが有効になる、関連付けられたテーブルに対する操作の種類です。値は、INSERT (行が挿入された場合)、DELETE (行が削除された場合) または UPDATE (行が変更された場合) です。
- **Table**
トリガーが定義されているテーブル。
- **ステートメント**
トリガー本体 (トリガーがアクティブ化されたときに実行されるステートメント)。
- **Timing**
トリガーを起動するイベントの前または後にトリガーをアクティブ化するかどうか。値は BEFORE または AFTER です。
- **Created**

トリガーが作成された日時。これは、トリガーの `TIMESTAMP(2)` 値 (小数部は数百秒) です。

- `sql_mode`

トリガーの作成時およびトリガーの実行時に有効な SQL モード。指定可能な値については、[セクション 5.1.11 「サーバー SQL モード」](#) を参照してください。

- `Definer`

トリガーを作成したユーザーのアカウント ('`user_name`'@'`host_name`'形式)。

- `character_set_client`

トリガー作成時の `character_set_client` システム変数のセッション値。

- `collation_connection`

トリガー作成時の `collation_connection` システム変数のセッション値。

- `Database Collation`

トリガーが関連付けられているデータベースの照合。

トリガー情報は、`INFORMATION_SCHEMA TRIGGERS` テーブルからも入手できます。[セクション 26.45 「INFORMATION_SCHEMA TRIGGERS テーブル」](#) を参照してください。

13.7.7.41 SHOW VARIABLES ステートメント

```
SHOW [GLOBAL | SESSION] VARIABLES  
[LIKE 'pattern' | WHERE expr]
```

`SHOW VARIABLES` には、MySQL システム変数の値が表示されます ([セクション 5.1.8 「サーバーシステム変数」](#) を参照)。このステートメントにはどの権限も必要ありません。これには、サーバーに接続できることのみが必要です。

システム変数情報は、次のソースからも入手できます:

- パフォーマンススキーマ tables. [セクション 27.12.14 「パフォーマンススキーマシステム変数テーブル」](#) を参照してください。
- `mysqladmin variables` コマンド。 [セクション 4.5.2 「mysqladmin — A MySQL Server 管理プログラム」](#) を参照してください。

`SHOW VARIABLES` の場合、`LIKE` 句 (存在する場合) は一致させる変数名を示します。[セクション 26.55 「SHOW ステートメントの拡張」](#) で説明されているように、より一般的な条件を使用して行を選択するために `WHERE` 句を指定できます。

`SHOW VARIABLES` は、オプションの `GLOBAL` または `SESSION` 変数スコープ修飾子を受け入れます:

- `GLOBAL` 修飾子を使用すると、ステートメントはグローバルシステム変数値を表示します。これらは、MySQL への新規接続に対応するセッション変数の初期化に使用される値です。変数にグローバル値がない場合、値は表示されません。
- `SESSION` 修飾子を使用すると、ステートメントは現在の接続で有効なシステム変数値を表示します。変数にセッション値がない場合は、グローバル値が表示されます。`LOCAL` は `SESSION` のシノニムです。
- 修飾子が存在しない場合、デフォルトは `SESSION` です。

各システム変数のスコープは、[セクション 5.1.8 「サーバーシステム変数」](#) にリストされています。

`SHOW VARIABLES` は、バージョンに依存する表示幅の制限に従います。完全には表示されない非常に長い値を持つ変数の場合、回避策として `SELECT` を使用します。例:

```
SELECT @@GLOBAL.innodb_data_file_path;
```

ほとんどのシステム変数は、サーバーの起動時に設定できます (`version_comment` などの読取り専用変数は例外です)。多くは、`SET` ステートメントを使用して実行時に変更できます。セクション5.1.9「システム変数の使用」およびセクション13.7.6.1「変数代入の `SET` 構文」を参照してください。

部分的な出力を次に示します。名前と値のリストは、サーバーによって異なる場合があります。セクション5.1.8「サーバーシステム変数」では、各変数の意味について説明し、セクション5.1.1「サーバーの構成」では、それらのチューニングに関する情報を提供します。

```
mysql> SHOW VARIABLES;
+-----+-----+
| Variable_name | Value |
+-----+-----+
| activate_all_roles_on_login | OFF |
| auto_generate_certs | ON |
| auto_increment_increment | 1 |
| auto_increment_offset | 1 |
| autocommit | ON |
| automatic_sp_privileges | ON |
| avoid_temporal_upgrade | OFF |
| back_log | 151 |
| basedir | /usr/ |
| big_tables | OFF |
| bind_address | * |
| binlog_cache_size | 32768 |
| binlog_checksum | CRC32 |
| binlog_direct_non_transactional_updates | OFF |
| binlog_error_action | ABORT_SERVER |
| binlog_expire_logs_seconds | 2592000 |
| binlog_format | ROW |
| binlog_group_commit_sync_delay | 0 |
| binlog_group_commit_sync_no_delay_count | 0 |
| binlog_gtid_simple_recovery | ON |
| binlog_max_flush_queue_time | 0 |
| binlog_order_commits | ON |
| binlog_row_image | FULL |
| binlog_row_metadata | MINIMAL |
| binlog_row_value_options | |
| binlog_rows_query_log_events | OFF |
| binlog_stmt_cache_size | 32768 |
| binlog_transaction_dependency_history_size | 25000 |
| binlog_transaction_dependency_tracking | COMMIT_ORDER |
| block_encryption_mode | aes-128-ecb |
| bulk_insert_buffer_size | 8388608 |
...
| max_allowed_packet | 67108864 |
| max_binlog_cache_size | 18446744073709547520 |
| max_binlog_size | 1073741824 |
| max_binlog_stmt_cache_size | 18446744073709547520 |
| max_connect_errors | 100 |
| max_connections | 151 |
| max_delayed_threads | 20 |
| max_digest_length | 1024 |
| max_error_count | 1024 |
| max_execution_time | 0 |
| max_heap_table_size | 16777216 |
| max_insert_delayed_threads | 20 |
| max_join_size | 18446744073709551615 |
...
| thread_handling | one-thread-per-connection |
| thread_stack | 286720 |
| time_zone | SYSTEM |
| timestamp | 1530906638.765316 |
| tls_version | TLSv1,TLSv1.1,TLSv1.2 |
| tmp_table_size | 16777216 |
| tmpdir | /tmp |
| transaction_alloc_block_size | 8192 |
| transaction_allow_batching | OFF |
```

```

| transaction_isolation      | REPEATABLE-READ      |
| transaction_prealloc_size | 4096                  |
| transaction_read_only     | OFF                   |
| transaction_write_set_extraction | XXHASH64             |
| unique_checks             | ON                    |
| updatable_views_with_limit | YES                   |
| version                   | 8.0.12                |
| version_comment           | MySQL Community Server - GPL |
| version_compile_machine   | x86_64                |
| version_compile_os        | Linux                 |
| version_compile_zlib      | 1.2.11                |
| wait_timeout              | 28800                 |
| warning_count              | 0                     |
| windowing_use_high_precision | ON                    |
+-----+-----+-----+

```

LIKE 句を指定すると、このステートメントは、そのパターンに一致する名前を持つ変数の行のみを表示します。特定の変数の行を取得するには、LIKE 句を次に示すように使用します。

```

SHOW VARIABLES LIKE 'max_join_size';
SHOW SESSION VARIABLES LIKE 'max_join_size';

```

名前がパターンと一致する変数のリストを取得するには、LIKE 句の中で % のワイルドカード文字を使用します。

```

SHOW VARIABLES LIKE '%size%';
SHOW GLOBAL VARIABLES LIKE '%size%';

```

ワイルドカード文字は、照合されるパターン内のどの場所でも利用できます。厳密に言えば、_ は単一の文字と一致するワイルドカードであるため、文字どおりに一致するように _ としてエスケープする必要があります。実際には、これはほとんど必要ありません。

13.7.7.42 SHOW WARNINGS ステートメント

```

SHOW WARNINGS [LIMIT [offset,] row_count]
SHOW COUNT(*) WARNINGS

```

SHOW WARNINGS は、現在のセッションでのステートメントの実行の結果として得られた条件 (エラー、警告、および注意) に関する情報を表示する診断ステートメントです。INSERT、UPDATE、LOAD DATA などの DML ステートメント、および CREATE TABLE や ALTER TABLE などの DDL ステートメントに対して警告が生成されます。

LIMIT 句の構文は、SELECT ステートメントの場合と同じです。セクション13.2.10「SELECT ステートメント」を参照してください。

SHOW WARNINGS は、EXPLAIN によって生成された拡張情報を表示するために、EXPLAIN の後にも使用されます。セクション8.8.3「拡張 EXPLAIN 出力形式」を参照してください。

SHOW WARNINGS では、現行のセッションで最新の非診断ステートメントを実行した結果の条件に関する情報が表示されます。解析中に最新のステートメントでエラーが発生した場合、SHOW WARNINGS では、ステートメントのタイプ (診断または非診断) に関係なく、結果の条件が表示されます。

SHOW COUNT(*) WARNINGS 診断ステートメントは、エラー、警告、および注意の総数を表示します。この数はまた、warning_count システム変数からも取得できます。

```

SHOW COUNT(*) WARNINGS;
SELECT @@warning_count;

```

これらのステートメントの違いは、最初のステートメントがメッセージリストをクリアしない診断ステートメントであることです。もう一方は SELECT ステートメントであるため、非診断とみなされ、メッセージリストはクリアされます。

関連する診断ステートメント SHOW ERRORS はエラー状態のみ (警告と注意は除外されます) を表示し、SHOW COUNT(*) ERRORS ステートメントはエラーの総数を表示します。セクション13.7.7.17「SHOW ERRORS ステートメント」を参照してください。GET DIAGNOSTICS を使用すると、個々の条件に関する情報を検査できます。セクション13.6.7.3「GET DIAGNOSTICS ステートメント」を参照してください。

次に、INSERT のデータ変換の警告を示す簡単な例を示します。この例では、厳密な SQL モードが無効であることを前提としています。厳密モードを有効にすると、警告がエラーになり、INSERT が終了します。

```

mysql> CREATE TABLE t1 (a TINYINT NOT NULL, b CHAR(4));

```



```
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO t1 VALUES(10,'mysql'), (NULL,'test'), (300,'xyz');
Query OK, 3 rows affected, 3 warnings (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 3

mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Warning
Code: 1265
Message: Data truncated for column 'b' at row 1
***** 2. row *****
Level: Warning
Code: 1048
Message: Column 'a' cannot be null
***** 3. row *****
Level: Warning
Code: 1264
Message: Out of range value for column 'a' at row 3
3 rows in set (0.00 sec)
```

`max_error_count` システム変数は、サーバーが情報を格納する対象となるエラー、警告、および注意メッセージの最大数、したがって `SHOW WARNINGS` が表示するメッセージの数を制御します。サーバーが格納できるメッセージの数を変更するには、`max_error_count` の値を変更します。

`max_error_count` は、カウントされるメッセージの数ではなく、格納されるメッセージの数のみを制御します。生成されたメッセージの数が `max_error_count` を超えた場合でも、`warning_count` の値は `max_error_count` によって制限されません。この点について次の例で説明します。この `ALTER TABLE` ステートメントは、3 つの警告メッセージを生成します (この例では、変換の問題が 1 つ発生したあとにエラーが発生しないように、厳密な SQL モードが無効になっています)。 `max_error_count` が 1 に設定されたため、格納されて表示されたメッセージは 1 つですが、`warning_count` の値で示されているように 3 つすべてがカウントされています。

```
mysql> SHOW VARIABLES LIKE 'max_error_count';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| max_error_count | 1024 |
+-----+-----+
1 row in set (0.00 sec)

mysql> SET max_error_count=1, sql_mode = "";
Query OK, 0 rows affected (0.00 sec)

mysql> ALTER TABLE t1 MODIFY b CHAR;
Query OK, 3 rows affected, 3 warnings (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 3

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Warning | 1263 | Data truncated for column 'b' at row 1 |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT @@warning_count;
+-----+
| @@warning_count |
+-----+
| 3 |
+-----+
1 row in set (0.01 sec)
```

メッセージの格納を無効にするには、`max_error_count` を 0 に設定します。この場合、`warning_count` は引き続き、発生した警告の数を示しますが、メッセージは格納されないため表示できません。

`sql_notes` システム変数は、注意メッセージで `warning_count` が増分されるかどうか、またサーバーがそれらを格納するかどうかを制御します。デフォルトでは、`sql_notes` は 1 ですが、0 に設定されている場合は、注意で `warning_count` が増分されず、またサーバーはそれらを格納しません。

```
mysql> SET sql_notes = 1;
```

```
mysql> DROP TABLE IF EXISTS test.no_such_table;
Query OK, 0 rows affected, 1 warning (0.00 sec)
mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Note | 1051 | Unknown table 'test.no_such_table' |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> SET sql_notes = 0;
mysql> DROP TABLE IF EXISTS test.no_such_table;
Query OK, 0 rows affected (0.00 sec)
mysql> SHOW WARNINGS;
Empty set (0.00 sec)
```

MySQL サーバーは、各クライアントに、そのクライアントによって実行された最新のステートメントの結果として得られたエラー、警告、および注意の総数を示す数を送信します。C API からは、この値は `mysql_warning_count()` を呼び出すことによって取得できます。 `mysql_warning_count()` を参照してください。

`mysql` クライアントでは、`warnings` コマンドと `nowarning` コマンド、またはそれらのショートカット `\W` と `\w` をそれぞれ使用して、自動警告表示を有効または無効にできます ([セクション4.5.1.2「mysql クライアントコマンド」](#) を参照)。例:

```
mysql> \W
Show warnings enabled.
mysql> SELECT 1/0;
+-----+
| 1/0 |
+-----+
| NULL |
+-----+
1 row in set, 1 warning (0.03 sec)

Warning (Code 1365): Division by 0
mysql> \w
Show warnings disabled.
```

13.7.8 その他の管理ステートメント

13.7.8.1 BINLOG ステートメント

```
BINLOG 'str'
```

`BINLOG` は、内部で使用されるステートメントです。これは、バイナリログファイル内の特定のイベントの印刷可能な表現として `mysqlbinlog` プログラムによって生成されます。([セクション4.6.8「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」](#) を参照してください。) `'str'` 値は、サーバーが、対応するイベントによって示されているデータ変更を判定するためにデコードする、base 64 でエンコードされた文字列です。

`mysqlbinlog` 出力の適用時に `BINLOG` ステートメントを実行するには、ユーザーアカウントに `BINLOG_ADMIN` 権限 (または非推奨の `SUPER` 権限) または `REPLICATION_APPLIER` 権限と、各ロギイベントを実行するための適切な権限が必要です。

このステートメントは、フォーマット記述イベントと行イベントのみを実行できます。

13.7.8.2 CACHE INDEX ステートメント

```
CACHE INDEX {
  tbl_index_list [, tbl_index_list] ...
  | tbl_name PARTITION (partition_list)
}
IN key_cache_name

tbl_index_list:
tbl_name [(INDEX|KEY) (index_name[, index_name] ...)]

partition_list: {
  partition_name[, partition_name] ...
  | ALL
}
```

CACHE INDEX ステートメントは、テーブルインデックスを特定のキーキャッシュに割り当てます。パーティション化された **MyISAM** テーブルを含む **MyISAM** テーブルにのみ適用されます。インデックスが割り当てられたら、これらのインデックスを、必要に応じて **LOAD INDEX INTO CACHE** でキャッシュにプリロードできます。

次のステートメントは、テーブル **t1**、**t2**、および **t3** のインデックスを **hot_cache** という名前のキーキャッシュに割り当てます。

```
mysql> CACHE INDEX t1, t2, t3 IN hot_cache;
+-----+-----+-----+
| Table | Op          | Msg_type | Msg_text |
+-----+-----+-----+
| test.t1 | assign_to_keycache | status | OK |
| test.t2 | assign_to_keycache | status | OK |
| test.t3 | assign_to_keycache | status | OK |
+-----+-----+-----+
```

CACHE INDEX の構文では、テーブルの特定のインデックスのみをキャッシュに割り当てるように指定できます。ただし、実装ではすべてのテーブルインデックスがキャッシュに割り当てられるため、テーブル名以外を指定する理由はありません。

CACHE INDEX ステートメントで参照されるキーキャッシュは、パラメータ設定ステートメントを使用して、またはサーバーのパラメータ設定でそのサイズを設定することによって作成できます。例:

```
SET GLOBAL keycache1.key_buffer_size=128*1024;
```

キーキャッシュパラメータには、構造化システム変数のメンバーとしてアクセスします。[セクション5.1.9.5「構造化システム変数」](#)を参照してください。

インデックスを割り当てる前に、キーキャッシュが存在している必要があります。存在していない場合は、エラーが発生します:

```
mysql> CACHE INDEX t1 IN non_existent_cache;
ERROR 1284 (HY000): Unknown key cache 'non_existent_cache'
```

デフォルトで、テーブルインデックスは、サーバー起動時に作成されるメイン (デフォルト) キーキャッシュに割り当てられます。キーキャッシュが破棄されると、それに割り当てられたすべてのインデックスはデフォルトのキーキャッシュに再割り当てされます。

インデックスの割り当ては、サーバーにグローバルに影響を与えます。あるクライアントがインデックスを特定のキャッシュに割り当てると、どのクライアントがクエリーを発行したかには関係なく、このキャッシュはそのインデックスに関連するすべてのクエリーに使用されます。

CACHE INDEX は、パーティション化された **MyISAM** テーブルでサポートされています。1つ、複数、またはすべてのパーティションの1つ以上のインデックスを特定のキーキャッシュに割り当てることができます。たとえば、次のステートメントを実行できます。

```
CREATE TABLE pt (c1 INT, c2 VARCHAR(50), INDEX i(c1))
ENGINE=MyISAM
PARTITION BY HASH(c1)
PARTITIONS 4;

SET GLOBAL kc_fast.key_buffer_size = 128 * 1024;
SET GLOBAL kc_slow.key_buffer_size = 128 * 1024;

CACHE INDEX pt PARTITION (p0) IN kc_fast;
CACHE INDEX pt PARTITION (p1, p3) IN kc_slow;
```

前の一連のステートメントは、次のアクションを実行します。

- 4つのパーティションを含むパーティション化されたテーブルを作成します。これらのパーティションには、自動的に **p0**、**...**、**p3** という名前が付けられます。このテーブルには、カラム **c1** 上に **i** という名前のインデックスが含まれています。
- kc_fast** と **kc_slow** という名前の2つのキーキャッシュを作成します。
- パーティション **p0** のインデックスを **kc_fast** キーキャッシュに、パーティション **p1** と **p3** のインデックスを **kc_slow** キーキャッシュに割り当てます。残りのパーティション (**p2**) のインデックスは、サーバーのデフォルトのキーキャッシュを使用します。

かわりに、テーブル `pt` のすべてのパーティションのインデックスを `kc_all` という名前の単一のキーキャッシュに割り当てる場合は、次のいずれかのステートメントを使用できます:

```
CACHE INDEX pt PARTITION (ALL) IN kc_all;
```

```
CACHE INDEX pt IN kc_all;
```

ここで示した 2 つのステートメントは同等であり、どちらか一方を発行してもまったく同じ効果があります。つまり、パーティションテーブルのすべてのパーティションのインデックスを同じキーキャッシュに割り当てる場合、`PARTITION (ALL)` 句はオプションです。

複数のパーティションのインデックスをキーキャッシュに割り当てる場合、パーティションは連続している必要はなく、それらの名前を特定の順序でリストする必要はありません。キーキャッシュに明示的に割り当てられていないパーティションのインデックスは、サーバーのデフォルトのキーキャッシュを自動的に使用します。

インデックスのプリロードは、パーティション化された `MyISAM` テーブルでもサポートされています。詳細は、[セクション 13.7.8.5 「LOAD INDEX INTO CACHE ステートメント」](#) を参照してください。

13.7.8.3 FLUSH ステートメント

```
FLUSH [NO_WRITE_TO_BINLOG | LOCAL] {  
  flush_option [, flush_option] ...  
  | tables_option  
}  
  
flush_option: {  
  BINARY LOGS  
  | ENGINE LOGS  
  | ERROR LOGS  
  | GENERAL LOGS  
  | HOSTS  
  | LOGS  
  | PRIVILEGES  
  | OPTIMIZER_COSTS  
  | RELAY LOGS [FOR CHANNEL channel]  
  | SLOW LOGS  
  | STATUS  
  | USER_RESOURCES  
}  
  
tables_option: {  
  TABLES  
  | TABLES tbl_name [, tbl_name] ...  
  | TABLES WITH READ LOCK  
  | TABLES tbl_name [, tbl_name] ... WITH READ LOCK  
  | TABLES tbl_name [, tbl_name] ... FOR EXPORT  
}
```

`FLUSH` ステートメントには、さまざまな内部キャッシュをクリアまたはリロードしたり、テーブルをフラッシュしたり、ロックを取得したりするいくつかのバリエーション形式があります。各 `FLUSH` 操作には、その説明に示されている権限が必要です。

注記

ストアドファンクションまたはトリガー内で `FLUSH` ステートメントを発行することはできません。ただし、ストアドプロシージャでは、それがストアドファンクションまたはトリガーから呼び出されないかぎり、`FLUSH` を使用できます。[セクション 25.8 「ストアドプログラムの制約」](#) を参照してください。

デフォルトでは、`FLUSH` ステートメントはレプリカにレプリケートされるようにバイナリログに書き込まれます。ロギングを抑制するには、オプションの `NO_WRITE_TO_BINLOG` キーワード、またはそのエイリアス `LOCAL` を指定します。

注記

レプリカにレプリケートすると問題が発生するため、`FLUSH LOGS`、`FLUSH BINARY LOGS`、`FLUSH TABLES WITH READ LOCK` (テーブルリストの有無にかかわらず)、および `FLUSH TABLES tbl_name ... FOR EXPORT` はバイナリログに書き込まれません。

FLUSH ステートメントは暗黙的なコミットを発生させます。 [セクション13.3.3「暗黙的なコミットを発生させるステートメント」](#) を参照してください。

mysqladmin ユーティリティは、[flush-hosts](#)、[flush-logs](#)、[flush-privileges](#)、[flush-status](#)、[flush-tables](#) などのコマンドを使用して、いくつかのフラッシュ操作へのコマンド行インターフェースを提供します。 [セクション4.5.2「mysqladmin — A MySQL Server 管理プログラム」](#) を参照してください。

SIGHUP または SIGUSR1 シグナルをサーバーに送信すると、FLUSH ステートメントのさまざまな形式に似たフラッシュ操作がいくつか発生します。シグナルは、root システムアカウントまたはサーバープロセスを所有するシステムアカウントによって送信できます。これにより、サーバーに接続せずにフラッシュ操作を実行できるようになり、その操作に十分な権限を持つ MySQL アカウントが必要になります。 [セクション4.10「MySQL での Unix シグナル処理」](#) を参照してください。

RESET ステートメントは、FLUSH に似ています。レプリケーションでの RESET の使用の詳細は、 [セクション13.7.8.6「RESET ステートメント」](#) を参照してください。

次のリストに、許可される FLUSH ステートメントの `flush_option` 値を示します。許可される `tables_option` 値の説明は、 [FLUSH TABLES 構文](#) を参照してください。

- **FLUSH BINARY LOGS**

サーバーが書き込んでいるバイナリログファイルを閉じてから再度開きます。バイナリロギングが有効になっている場合は、バイナリログファイルのシーケンス番号が、前のファイルを基準にして 1 増分されます。

この操作には、RELOAD 権限が必要です。

- **FLUSH ENGINE LOGS**

インストールされているストレージエンジンのフラッシュ可能なログを閉じて再度開きます。これにより、InnoDB はログをディスクにフラッシュします。

この操作には、RELOAD 権限が必要です。

- **FLUSH ERROR LOGS**

サーバーが書き込んでいるエラーログファイルを閉じて再度開きます。

この操作には、RELOAD 権限が必要です。

- **FLUSH GENERAL LOGS**

サーバーが書き込んでいる一般クエリーログファイルを閉じて再度開きます。

この操作には、RELOAD 権限が必要です。

この操作は、一般クエリーログに使用されるテーブルには影響しません ([セクション5.4.1「一般クエリーログおよびスロークエリーログの出力先の選択」](#) を参照)。

- **FLUSH HOSTS**

キャッシュの内容を公開するホストキャッシュとパフォーマンススキーマ `host_cache` テーブルを空にし、ブロックされたホストをブロック解除します。

この操作には、RELOAD 権限が必要です。

ホストキャッシュのフラッシュが推奨または望ましい理由については、 [セクション5.1.12.3「DNS ルックアップとホストキャッシュ」](#) を参照してください。

注記

FLUSH HOSTS は、MySQL 8.0.23 では非推奨です。将来の MySQL リリースでは削除される予定です。代わりに、パフォーマンススキーマ `host_cache` テーブルを切り捨てます:

```
TRUNCATE TABLE performance_schema.host_cache;
```

TRUNCATE TABLE 操作には、**RELOAD** 権限ではなく、テーブルに対する **DROP** 権限が必要です。

• **FLUSH LOGS**

サーバーが書き込んでいるログファイルを閉じて再度開きます。

この操作には、**RELOAD** 権限が必要です。

この操作の効果は、次の操作を組み合わせた効果と同等です:

```
FLUSH BINARY LOGS
FLUSH ENGINE LOGS
FLUSH ERROR LOGS
FLUSH GENERAL LOGS
FLUSH RELAY LOGS
FLUSH SLOW LOGS
```

• **FLUSH OPTIMIZER_COSTS**

コストモデルテーブルに格納されている現在のコスト見積りの使用をオプティマイザが開始するように、コストモデルテーブルを再読取りします。

この操作には、**FLUSH_OPTIMIZER_COSTS** または **RELOAD** 権限が必要です。

サーバーは、認識できないコストモデルテーブルエントリについて、エラーログに警告を書き込みます。これらのテーブルの詳細は、[セクション8.9.5「オプティマイザコストモデル」](#)を参照してください。この操作は、フラッシュ後に開始されるセッションにのみ影響します。既存のセッションでは、開始時の現行のコスト見積りが引き続き使用されます。

• **FLUSH PRIVILEGES**

mysql システムスキーマの付与テーブルから権限を再度読み取ります。この操作の一環として、サーバーは動的権限割当てを含む **global_grants** テーブルを読み取り、そこで見つかった未登録の権限を登録します。

この操作には、**RELOAD** 権限が必要です。

MySQL 権限システムを無効にするためにサーバーの起動時に **--skip-grant-tables** オプションが指定された場合、**FLUSH PRIVILEGES** は実行時に権限システムを有効にする方法を提供します。

失敗したログイントラッキングをリセットし (またはサーバーが **--skip-grant-tables** で起動された場合は有効にします)、一時的にロックされたアカウントのロックを解除します。[セクション6.2.15「パスワード管理」](#)を参照してください。

GRANT, **CREATE USER**, **CREATE SERVER** および **INSTALL PLUGIN** ステートメントの結果としてサーバーによってキャッシュされたメモリーを解放します。このメモリーは、対応する **REVOKE**, **DROP USER**, **DROP SERVER** ステートメントおよび **UNINSTALL PLUGIN** ステートメントによって解放されないため、キャッシュを引き起こすステートメントの多くのインスタンスを実行するサーバーでは、**FLUSH PRIVILEGES** で解放されないが、キャッシュされたメモリーの使用量が増加します。

cached_sha2_password 認証プラグインで使用されるインメモリーキャッシュをクリアします。[SHA-2 プラグイン認証のキャッシュ操作](#)を参照してください。

• **FLUSH RELAY LOGS [FOR CHANNEL channel]**

サーバーが書き込んでいるリレーログファイルを閉じてから再度開きます。リレーロギングが有効になっている場合、リレーログファイルのシーケンス番号は、前のファイルを基準にして1ずつ増分されます。

この操作には、**RELOAD** 権限が必要です。

FOR CHANNEL channel 句を使用すると、操作を適用するレプリケーションチャンネルの名前を指定できます。**FLUSH RELAY LOGS FOR CHANNEL channel** を実行して、特定のレプリケーションチャンネルのリレーログをフラッシュします。チャンネルが指定されておらず、追加のレプリケーションチャンネルが存在しない場合、操作はデフォルトチャンネルに適用されます。チャンネルが指定されておらず、複数のレプリケーションチャンネルが存在する

場合、操作はすべてのレプリケーションチャンネルに適用されます。詳細は、[セクション17.2.2「レプリケーションチャンネル」](#)を参照してください。

- [FLUSH SLOW LOGS](#)

サーバーが書き込んでいるスロークエリーログファイルを閉じてから再度開きます。

この操作には、[RELOAD](#) 権限が必要です。

この操作は、スロークエリーログに使用されるテーブルには影響しません ([セクション5.4.1「一般クエリーログおよびスロークエリーログの出力先の選択」](#)を参照)。

- [FLUSH STATUS](#)

ステータスインジケータをフラッシュします。

この操作には、[FLUSH_STATUS](#) または [RELOAD](#) 権限が必要です。

この操作により、すべてのアクティブセッションのセッションステータスがグローバルステータス変数に追加され、すべてのアクティブセッションのステータスがリセットされ、切断されたセッションから集計されたアカウント、ホストおよびユーザーステータス値がリセットされます。[セクション27.12.15「パフォーマンススキーマのステータス変数のテーブル」](#)を参照してください。この情報は、クエリーのデバッグ時に使用できます。[セクション1.6「質問またはバグをレポートする方法」](#)を参照してください。

- [FLUSH USER_RESOURCES](#)

時間当たりのすべてのユーザーリソースインジケータをゼロにリセットします。

この操作には、[FLUSH_USER_RESOURCES](#) または [RELOAD](#) 権限が必要です。

リソースインジケータをリセットすると、毎時の接続、クエリーまたは更新の制限に達したクライアントは、アクティビティをすぐに再開できます。[FLUSH_USER_RESOURCES](#) は、[max_user_connections](#) システム変数によって制御される同時接続の最大数の制限には適用されません。[セクション6.2.20「アカウントリソース制限の設定」](#)を参照してください。

FLUSH TABLES 構文

[FLUSH TABLES](#) はテーブルをフラッシュし、使用されているバリエーションに応じてロックを取得します。[FLUSH](#) ステートメントで使用される [TABLES](#) バリエーションは、使用される唯一のオプションである必要があります。[FLUSH TABLE](#) は、[FLUSH TABLES](#) のシノニムです。

注記

ここでは、テーブルを閉じることによってフラッシュされることを示す説明は、テーブルの内容をディスクにフラッシュし、開いたままにする [InnoDB](#) には異なる方法で適用されません。これにより、他のアクティビティによって変更されないかぎり、テーブルが開いている間もテーブルファイルをコピーできます。

- [FLUSH TABLES](#)

オープンしているすべてのテーブルをクローズし、使用中のすべてのテーブルを強制的にクローズし、プリペアドステートメントキャッシュをフラッシュします。

この操作には、[FLUSH_TABLES](#) または [RELOAD](#) 権限が必要です。

プリペアドステートメントキャッシュの詳細は、[セクション8.10.3「プリペアドステートメントおよびストアードプログラムのキャッシュ」](#)を参照してください。

アクティブな [LOCK TABLES ... READ](#) がある場合、[FLUSH TABLES](#) は許可されません。テーブルをフラッシュしてロックするには、代わりに [FLUSH TABLES tbl_name ... WITH READ LOCK](#) を使用します。

- [FLUSH TABLES tbl_name \[, tbl_name\] ...](#)

カンマ区切りのテーブル名のリストでは、サーバーが名前付きのテーブルのみをフラッシュする点を除き、この操作は名前のない `FLUSH TABLES` に似ています。指定したテーブルが存在しない場合、エラーは発生しません。

この操作には、`FLUSH_TABLES` または `RELOAD` 権限が必要です。

• `FLUSH TABLES WITH READ LOCK`

開かれているすべてのテーブルを閉じ、グローバルな読み取りロックを保持しているすべてのデータベースのすべてのテーブルをロックします。

この操作には、`FLUSH_TABLES` または `RELOAD` 権限が必要です。

この操作は、時間内にスナップショットを取得できる Veritas や ZFS などのファイルシステムがある場合に、バックアップを取得するための非常に便利な方法です。このロックを解放するには、`UNLOCK TABLES` を使用しません。

`FLUSH TABLES WITH READ LOCK` は、テーブルロックではなくグローバル読み取りロックを取得するため、テーブルロックおよび暗黙的コミットに関して `LOCK TABLES` および `UNLOCK TABLES` と同じ動作を受けません:

- `UNLOCK TABLES` は、現在 `LOCK TABLES` でロックされているテーブルがある場合にのみ、アクティブなトランザクションをすべて暗黙的にコミットします。`FLUSH TABLES WITH READ LOCK` はテーブルロックを取得しないため、このステートメントに続く `UNLOCK TABLES` に対してコミットは発生しません。
- トランザクションを開始すると、ユーザーが `UNLOCK TABLES` を実行したかのように、`LOCK TABLES` によって取得されたテーブルロックが解放されます。トランザクションを開始しても、`FLUSH TABLES WITH READ LOCK` によって取得されたグローバルな読み取りロックは解放されません。

`FLUSH TABLES WITH READ LOCK` では、サーバーがログテーブルに行を挿入することは妨げられません ([セクション 5.4.1 「一般クエリーログおよびスロークエリーログの出力先の選択」](#) を参照してください)。

• `FLUSH TABLES tbl_name [, tbl_name] ... WITH READ LOCK`

指定されたテーブルの読み取りロックをフラッシュおよび取得します。

この操作には、`FLUSH_TABLES` または `RELOAD` 権限が必要です。テーブルロックを取得するため、各テーブルに対する `LOCK TABLES` 権限も必要です。

この操作では、最初にテーブルの排他的メタデータロックが取得されるため、これらのテーブルが開いているトランザクションが完了するまで待機します。次に、操作によってテーブルキャッシュからテーブルがフラッシュされ、テーブルが再オープンされ、テーブルロック (`LOCK TABLES ... READ` など) が取得され、メタデータロックが排他から共有にダウングレードされます。操作によってロックが取得され、メタデータロックがダウングレードされた後、他のセッションはテーブルの読み取りはできますが、変更はできません。

この操作は、既存の実テーブル (`TEMPORARY` 以外のテーブル) にのみ適用されます。名前がベーステーブルを参照している場合は、そのテーブルが使用されます。`TEMPORARY` テーブルを参照している場合、その名前は無視されます。名前がビューに適用される場合は、`ER_WRONG_OBJECT` エラーが発生します。それ以外の場合は、`ER_NO_SUCH_TABLE` エラーが発生します。

ロックを解放するには `UNLOCK TABLES` を、ロックを解放し、ほかのロックを取得するには `LOCK TABLES` を、またはロックを解放し、新しいトランザクションを開始するには `START TRANSACTION` を使用します。

この `FLUSH TABLES` バリエーションを使用すると、単一の操作でテーブルをフラッシュおよびロックできます。これにより、アクティブな `LOCK TABLES ... READ` がある場合に `FLUSH TABLES` が許可されないという制限の回避策が提供されます。

この操作では暗黙的な `UNLOCK TABLES` は実行されないため、アクティブな `LOCK TABLES` がある間に操作を実行するか、最初に取得したロックを解放せずに再度使用すると、エラーが発生します。

フラッシュされたテーブルが `HANDLER` で開かれた場合、そのハンドラは暗黙的にフラッシュされ、その位置を失います。

- `FLUSH TABLES tbl_name [, tbl_name] ... FOR EXPORT`

この `FLUSH TABLES` バリエーションは、InnoDB テーブルに適用されます。これにより、指定されたテーブルへの変更がディスクにフラッシュされ、サーバーの実行中にバイナリテーブルのコピーを作成できるようになります。

この操作には、`FLUSH TABLES` または `RELOAD` 権限が必要です。エクスポートの準備としてテーブルのロックを取得するため、テーブルごとに `LOCK TABLES` および `SELECT` 権限も必要です。

この操作は次のように機能します:

1. 指定されたテーブルに対する共有メタデータロックを取得します。他のセッションに、それらのテーブルを変更したアクティブなトランザクションがあるか、それらのテーブルのロックを保持しているかぎり、操作はブロックされます。ロックが取得されると、読取り専用操作の続行を許可しながら、テーブルの更新を試行するトランザクションがブロックされます。
2. これらのテーブルのすべてのストレージエンジンが `FOR EXPORT` をサポートしているかどうかをチェックします。存在しない場合は、`ER_ILLEGAL HA` エラーが発生し、操作は失敗します。
3. この操作は、各テーブルのストレージエンジンに、テーブルをエクスポートできるように通知します。そのストレージエンジンは、保留中の変更がすべてディスクに書き込まれるようにする必要があります。
4. この操作によってセッションがロックテーブルモードになり、以前に取得したメタデータロックが `FOR EXPORT` 操作の完了時に解放されなくなります。

この操作は、既存の (`TEMPORARY` 以外の) 実テーブルにのみ適用されます。名前がベーステーブルを参照している場合は、そのテーブルが使用されます。`TEMPORARY` テーブルを参照している場合、その名前は無視されます。名前がビューに適用される場合は、`ER_WRONG_OBJECT` エラーが発生します。それ以外の場合は、`ER_NO_SUCH_TABLE` エラーが発生します。

InnoDB は、独自の `.ibd file` ファイル (`innodb_file_per_table` 設定を有効にして作成されたテーブル) を持つテーブルに対して `FOR EXPORT` をサポートしています。InnoDB では、`FOR EXPORT` 操作によって変更がディスクにフラッシュされたことが通知されます。これにより、`.ibd` ファイルはトランザクションの一貫性があり、サーバーの実行中にコピーできるため、`FOR EXPORT` 操作が有効な間にテーブルの内容のバイナリコピーを作成できます。`FOR EXPORT` は、InnoDB システムテーブルスペースファイル、または `FULLTEXT` インデックスを持つ InnoDB テーブルには適用されません。

`FLUSH TABLES ...FOR EXPORT` は、パーティション化された InnoDB テーブルでサポートされています。

`FOR EXPORT` から通知されると、InnoDB は、通常はメモリー内か、またはテーブルスペースファイルの外部にある個別のディスクバッファに保持される特定の種類のデータをディスクに書き込みます。InnoDB はまた、テーブルごとに、そのテーブルと同じデータベースディレクトリ内に `table_name.cfg` という名前のファイルも生成します。`.cfg` ファイルには、あとでテーブルスペースファイルを同じサーバーまたは別のサーバーに再インポートするために必要なメタデータが含まれています。

`FOR EXPORT` 操作が完了すると、InnoDB はすべての `dirty pages` をテーブルデータファイルにフラッシュしました。`変更バッファ`のエントリはすべて、フラッシュの前にマージされます。この時点で、テーブルはロックされ、静止します。これらのテーブルはディスク上でトランザクション的に一貫性のある状態にあるため、`.ibd` テーブルスペースファイルに対応する `.cfg` ファイルとともにコピーすることによって、これらのテーブルの整合性のあるスナップショットを取得できます。

コピーされたテーブルデータを MySQL インスタンスに再インポートする手順については、[セクション 15.6.1.3 「InnoDB テーブルのインポート」](#) を参照してください。

テーブルの処理を完了したあと、ロックを解放するには `UNLOCK TABLES` を、ロックを解放し、ほかのロックを取得するには `LOCK TABLES` を、またはロックを解放し、新しいトランザクションを開始するには `START TRANSACTION` を使用します。

セッション内で次のいずれかのステートメントが有効になっている間は、`FLUSH TABLES ... FOR EXPORT` を使用しようとするとエラーが生成されます。

```
FLUSH TABLES ... WITH READ LOCK
FLUSH TABLES ... FOR EXPORT
LOCK TABLES ... READ
```

```
LOCK TABLES ... WRITE
```

セッション内で `FLUSH TABLES ... FOR EXPORT` が有効になっている間は、次のいずれかのステートメントを使用しようとするエラーが生成されます。

```
FLUSH TABLES WITH READ LOCK  
FLUSH TABLES ... WITH READ LOCK  
FLUSH TABLES ... FOR EXPORT
```

13.7.8.4 KILL ステートメント

```
KILL [CONNECTION | QUERY] processlist_id
```

`mysqld` への各接続は、個別のスレッドで実行されます。スレッドは、`KILL processlist_id` ステートメントで強制終了できます。

スレッドプロセスリスト識別子は、`INFORMATION_SCHEMA PROCESLIST` テーブルの `ID` カラム、`SHOW PROCESLIST` 出力の `Id` カラム、およびパフォーマンススキーマ `threads` テーブルの `PROCESSLIST_ID` カラムから決定できます。現在のスレッドの値は、`CONNECTION_ID()` 関数によって返されます。

`KILL` では、オプションの `CONNECTION` または `QUERY` 修飾子が許可されます。

- `KILL CONNECTION` は修飾子のない `KILL` と同じです: 接続が実行されているステートメントを終了すると、指定された `processlist_id` に関連付けられている接続が終了します。
- `KILL QUERY` は、接続が現在実行されているステートメントを終了しますが、接続自体はそのままになります。

強制終了できるスレッドを表示できるかどうかは、`PROCESS` 権限によって異なります:

- `PROCESS` がない場合は、自分のスレッドのみを表示できます。
- `PROCESS` では、すべてのスレッドを表示できます。

スレッドおよびステートメントを強制終了できるかどうかは、`CONNECTION_ADMIN` 権限および非推奨の `SUPER` 権限によって異なります:

- `CONNECTION_ADMIN` または `SUPER` がない場合は、独自のスレッドおよびステートメントのみを強制終了できます。
- `CONNECTION_ADMIN` または `SUPER` を使用すると、`SYSTEM_USER` 権限で実行されているスレッドまたはステートメントに影響を与える場合を除き、すべてのスレッドおよびステートメントを強制終了できます。独自のセッションには、`SYSTEM_USER` 権限も必要です。

`mysqladmin processlist` および `mysqladmin kill` コマンドを使用して、スレッドを検査および強制終了することもできます。

`KILL` を使用すると、そのスレッドのスレッド固有の強制終了フラグが設定されます。強制終了フラグは次の一定の間隔でしかチェックされないため、ほとんどの場合、スレッドが終了するまでにある程度時間がかかることがあります。

- `SELECT` 操作中、`ORDER BY` および `GROUP BY` ループでは、このフラグは行ブロックの読み取りのあとにチェックされます。強制終了フラグが設定されている場合、このステートメントは中止されます。
- テーブルのコピーを作成する `ALTER TABLE` 操作では、元のテーブルから読み取られたいくつかのコピーされた行について、強制終了フラグが定期的にチェックされます。強制終了フラグが設定されていた場合、このステートメントは中止され、一時テーブルが削除されます。

`KILL` ステートメントは確認を待機せずに戻りますが、強制終了フラグチェックにより、妥当な時間内に操作が中断されます。必要なクリーンアップを実行する操作を中止するには、時間もかかります。

- `UPDATE` または `DELETE` 操作中、強制終了フラグは、ブロックが読み取られるたび、および行が更新または削除されるたびにチェックされます。強制終了フラグが設定されている場合、このステートメントは中止されます。トランザクションを使用していない場合、変更はロールバックされません。
- `GET_LOCK()` は中止され、`NULL` を返します。

- このスレッドがテーブルロックハンドラ内にある場合 (状態: **Locked**)、そのテーブルロックはすばやく中止されま
- このスレッドが書き込みコールでディスクの空き容量を待機している場合、その書き込みは「ディスク領域不足」というエラーメッセージで中止されます。
- **EXPLAIN ANALYZE** は異常終了し、出力の最初の行を出力します。これは、MySQL 8.0.20 以降で機能します。

警告

MyISAM テーブルに対する **REPAIR TABLE** または **OPTIMIZE TABLE** 操作を強制終了すると、テーブルが破損して使用できなくなります。このようなテーブルに対する読み取りまたは書き込みはすべて、そのテーブルをふたたび最適化または修復するまで失敗します (割り込みはなし)。

13.7.8.5 **LOAD INDEX INTO CACHE** ステートメント

```
LOAD INDEX INTO CACHE
tbl_index_list [, tbl_index_list] ...

tbl_index_list:
tbl_name
[PARTITION (partition_list)]
{[(INDEX|KEY) (index_name[, index_name] ...)]
[IGNORE LEAVES]}

partition_list: {
partition_name[, partition_name] ...
| ALL
}
```

LOAD INDEX INTO CACHE ステートメントは、明示的な **CACHE INDEX** ステートメントによって割り当てられたキーキャッシュ、またはそれ以外の場合はデフォルトのキーキャッシュにテーブルインデックスをプリロードします。

LOAD INDEX INTO CACHE は、パーティション化された **MyISAM** テーブルを含む **MyISAM** テーブルにのみ適用されます。また、パーティションテーブルのインデックスは、1つ、複数またはすべてのパーティションに対して事前ロードできます。

IGNORE LEAVES 修飾子によって、インデックスの非リーフノードのブロックのみがプリロードされます。

IGNORE LEAVES は、パーティション化された **MyISAM** テーブルに対してもサポートされます。

次のステートメントは、テーブル **t1** と **t2** のインデックスのノード (インデックスブロック) をプリロードします。

```
mysql> LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES;
+-----+-----+-----+-----+
| Table | Op      | Msg_type | Msg_text |
+-----+-----+-----+-----+
| test.t1 | preload_keys | status | OK      |
| test.t2 | preload_keys | status | OK      |
+-----+-----+-----+-----+
```

このステートメントは、**t1** からすべてのインデックスブロックをプリロードします。**t2** からは、非リーフノードのブロックのみをプリロードします。

LOAD INDEX INTO CACHE の構文では、テーブルの特定のインデックスのみをプリロードするように指定できます。ただし、実装ではすべてのテーブルインデックスがキャッシュに事前ロードされるため、テーブル名以外は指定する必要はありません。

パーティション化された **MyISAM** テーブルの特定のパーティションでインデックスを事前ロードできます。たとえば、次の2つのステートメントでは、最初が、パーティション化されたテーブル **pt** のパーティション **p0** のインデックスをプリロードするのに対して、2番目は同じテーブルのパーティション **p1** と **p3** のインデックスをプリロードします。

```
LOAD INDEX INTO CACHE pt PARTITION (p0);
LOAD INDEX INTO CACHE pt PARTITION (p1, p3);
```


テーブル `pt` のすべてのパーティションのインデックスを事前ロードするには、次のいずれかのステートメントを使用します:

```
LOAD INDEX INTO CACHE pt PARTITION (ALL);
```

```
LOAD INDEX INTO CACHE pt;
```

ここで示した 2 つのステートメントは同等であり、どちらか一方を発行してもまったく同じ効果があります。つまり、パーティションテーブルのすべてのパーティションのインデックスを事前ロードする場合、`PARTITION (ALL)` 句はオプションです。

複数のパーティションのインデックスを事前ロードする場合、パーティションは連続している必要はなく、パーティション名を特定の順序でリストする必要はありません。

`LOAD INDEX INTO CACHE ... IGNORE LEAVES` は、テーブル内のすべてのインデックスのブロックサイズが同じでないかぎり失敗します。テーブルのインデックスブロックサイズを決定するには、`myisamchk -dv` を使用して `Blocksize` カラムを確認します。

13.7.8.6 RESET ステートメント

```
RESET reset_option [, reset_option] ...
```

```
reset_option: {  
  MASTER  
  | REPLIC  
  | SLAVE  
}
```

`RESET` ステートメントは、さまざまなサーバー操作の状態をクリアするために使用されます。`RESET` を実行するには、`RELOAD` 権限が必要です。

永続的なグローバルシステム変数を削除する `RESET PERSIST` ステートメントの詳細は、[セクション 13.7.8.7 「RESET PERSIST ステートメント」](#) を参照してください。

`RESET` は、`FLUSH` ステートメントのより強力なバージョンとして機能します。[セクション 13.7.8.3 「FLUSH ステートメント」](#) を参照してください。

`RESET` ステートメントは暗黙的なコミットを発生させます。[セクション 13.3.3 「暗黙的なコミットを発生させるステートメント」](#) を参照してください。

次のリストに、許可される `RESET` ステートメントの `reset_option` 値を示します:

- `RESET MASTER`

インデックスファイルにリストされているすべてのバイナリログを削除し、バイナリログインデックスファイルを空にリセットして、新しいバイナリログファイルを作成します。

- `RESET REPLIC | SLAVE`

レプリカがソースバイナリログ内のレプリケーション位置を忘れられるようにします。また、既存のリレーログファイルをすべて削除し、新しいリレーログファイルを開始することによってリレーログもリセットします。MySQL 8.0.22 の `RESET SLAVE` のかわりに `RESET REPLIC` を使用します。

13.7.8.7 RESET PERSIST ステートメント

```
RESET PERSIST [[IF EXISTS] system_var_name]
```

`RESET PERSIST` は、永続化されたグローバルシステム変数設定をデータディレクトリの `mysqld-auto.cnf` オプションファイルから削除します。永続化されたシステム変数を削除すると、その変数はサーバーの起動時に `mysqld-auto.cnf` から初期化されなくなります。システム変数および `mysqld-auto.cnf` ファイルの永続化の詳細は、[セクション 5.1.9.3 「永続化されるシステム変数」](#) を参照してください。

`RESET PERSIST` に必要な権限は、削除するシステム変数のタイプによって異なります:

- 動的システム変数の場合、このステートメントには `SYSTEM_VARIABLES_ADMIN` 権限 (または非推奨の `SUPER` 権限) が必要です。

- 読取り専用システム変数の場合、このステートメントには `SYSTEM_VARIABLES_ADMIN` および `PERSIST_RO_VARIABLES_ADMIN` 権限が必要です。

セクション5.1.9.1「システム変数権限」を参照してください。

変数名と `IF EXISTS` 句が存在するかどうかに応じて、`RESET PERSIST` ステートメントの形式は次のようになります:

- `mysqld-auto.cnf` からすべての永続変数を削除するには、システム変数に名前を付けずに `RESET PERSIST` を使用します:

```
RESET PERSIST;
```

`mysqld-auto.cnf` に両方の種類の変数が含まれている場合は、動的システム変数と読取り専用システム変数の両方を削除する権限が必要です。

- `mysqld-auto.cnf` から特定の永続変数を削除するには、ステートメントで名前を付けます:

```
RESET PERSIST system_var_name;
```

これには、プラグインが現在インストールされていない場合でも、プラグインシステム変数が含まれます。変数がファイルに存在しない場合は、エラーが発生します。

- `mysqld-auto.cnf` から特定の永続変数を削除し、ファイルに変数が存在しない場合にエラーではなく警告を生成するには、前の構文に `IF EXISTS` 句を追加します:

```
RESET PERSIST IF EXISTS system_var_name;
```

`RESET PERSIST` は、`persisted_globals_load` システム変数の値の影響を受けません。

テーブルの内容は `mysqld-auto.cnf` ファイルの内容に対応しているため、`RESET PERSIST` はパフォーマンススキーマ `persisted_variables` テーブルの内容に影響します。一方、`RESET PERSIST` は変数値を変更しないため、サーバーが再起動されるまでパフォーマンススキーマ `variables_info` テーブルの内容には影響しません。

ほかのサーバー操作の状態をクリアする `RESET` ステートメントのバリエーションについては、セクション13.7.8.6「`RESET` ステートメント」を参照してください。

13.7.8.8 RESTART ステートメント

```
RESTART
```

このステートメントは、MySQL サーバーを停止して再起動します。 `SHUTDOWN` 権限が必要です。

`RESTART` の使用例としては、サーバーホスト上の MySQL サーバーを再起動するためのコマンドラインアクセスが不可能な場合や便利な場合があります。たとえば、`SET PERSIST_ONLY` を実行時に使用して、サーバーの起動時にのみ設定できるシステム変数に構成変更を加えることができますが、これらの変更を有効にするには、サーバーを再起動する必要があります。 `RESTART` ステートメントは、サーバーホストでのコマンド行アクセスを必要とせずに、クライアントセッション内からこれを実行する方法を提供します。

注記

`RESTART` ステートメントの実行後、クライアントは現在の接続が失われることを期待できません。自動再接続が有効な場合、サーバーの再起動後に接続が再確立されます。それ以外の場合は、接続を手動で再確立する必要があります。

`RESTART` 操作が成功するには、再起動のために実行されたサーバーの停止を検出するための監視プロセスが使用可能な環境で `mysqld` が実行されている必要があります:

- モニタリングプロセスが存在する場合、`RESTART` は、モニタリングプロセスが新しい `mysqld` インスタンスを起動する必要があると判断できるように `mysqld` を終了させます。
- 監視プロセスが存在しない場合、`RESTART` はエラーで失敗します。

これらのプラットフォームは、`RESTART` ステートメントに必要な監視サポートを提供します:

- Windows (`mysqld` が Windows サービスまたはスタンドアロンとして起動された場合)。(`mysqld` がフォークし、一方のプロセスが他方のプロセスのモニターとして機能し、サーバーとして機能します。)
- `systemd` または `mysqld_safe` を使用して `mysqld` を管理する Unix および Unix に似たシステム。

`mysqld` で `RESTART` ステートメントが有効になるように監視環境を構成するには:

1. `mysqld` を起動する前に、`MYSQLD_PARENT_PID` 環境変数を、`mysqld` を起動するプロセスのプロセス ID の値に設定します。
2. `RESTART` ステートメントを使用して `mysqld` が停止を実行すると、終了コード 16 が返されます。
3. モニタリングプロセスは、16 の終了コードを検出すると、`mysqld` を再起動します。それ以外の場合は終了します。

`bash` Shell に実装されている最小限の例を次に示します:

```
#!/bin/bash

export MYSQLD_PARENT_PID=$$

export MYSQLD_RESTART_EXIT=16

while true ; do
  bin/mysqld mysqld options here
  if [ $? -ne $MYSQLD_RESTART_EXIT ]; then
    break
  fi
done
```

Windows では、`RESTART` の実装に使用されるフォーキングにより、デバッグを困難にするためにアタッチするサーバープロセスを決定できます。これを軽減するために、`--gdb` でサーバーを起動すると、デバッグ環境を設定するために実行されるその他のアクションに加えて、フォーキングが抑制されます。デバッグ以外の設定では、モニタープロセスの強制を抑制する唯一の目的で `--no-monitor` を使用できます。`--gdb` または `--no-monitor` を使用して起動されたサーバーの場合、`RESTART` を実行すると、サーバーは再起動せずに単に終了します。

`Com_restart` ステータス変数は、`RESTART` ステートメントの数を追跡します。ステータス変数はサーバーの起動ごとに初期化され、再起動後も保持されないため、`Com_restart` の値は通常ゼロですが、`RESTART` ステートメントが実行されたが失敗した場合はゼロ以外にすることができます。

13.7.8.9 SHUTDOWN ステートメント

SHUTDOWN

このステートメントは、MySQL サーバーを停止します。 `SHUTDOWN` 権限が必要です。

`SHUTDOWN` には、`mysqladmin shutdown` コマンドまたは `mysql_shutdown()` C API 関数を使用して使用可能な同じ機能に対する SQL レベルのインタフェースが用意されています。

`Com_shutdown` ステータス変数は、`SHUTDOWN` ステートメントの数を追跡します。ステータス変数はサーバーの起動ごとに初期化され、再起動後も保持されないため、`Com_shutdown` の値は通常ゼロですが、`SHUTDOWN` ステートメントが実行されたが失敗した場合はゼロ以外にすることができます。

サーバーを停止する別の方法は、サーバーに `SIGTERM` シグナルを送信することです。これは、`root` またはサーバープロセスを所有するアカウントによって実行できます。`SIGTERM` を使用すると、サーバーに接続せずにサーバーの停止を実行できます。[セクション4.10「MySQLでのUnixシグナル処理」](#)を参照してください。

13.8 ユーティリティステートメント

13.8.1 DESCRIBE ステートメント

`DESCRIBE` ステートメントと `EXPLAIN` ステートメントはシノニムであり、テーブル構造またはクエリー実行計画に関する情報を取得するために使用されます。詳細は、[セクション13.7.7.5「SHOW COLUMNS ステートメント」](#) および [セクション13.8.2「EXPLAIN ステートメント」](#) を参照してください。

13.8.2 EXPLAIN ステートメント

```
{EXPLAIN | DESCRIBE | DESC}
tbl_name [col_name | wild]

{EXPLAIN | DESCRIBE | DESC}
[explain_type]
{explainable_stmt | FOR CONNECTION connection_id}

{EXPLAIN | DESCRIBE | DESC} ANALYZE [FORMAT = TREE] select_statement

explain_type: {
  FORMAT = format_name
}

format_name: {
  TRADITIONAL
  | JSON
  | TREE
}

explainable_stmt: {
  SELECT statement
  | TABLE statement
  | DELETE statement
  | INSERT statement
  | REPLACE statement
  | UPDATE statement
}
```

DESCRIBE ステートメントと **EXPLAIN** ステートメントはシノニムです。実際には、**DESCRIBE** キーワードがテーブル構造に関する情報を取得するためにより頻繁に使用されるのに対して、**EXPLAIN** は、クエリー実行計画 (つまり、MySQL がクエリーをどのように実行するかの説明) を取得するために使用されます。

次の説明では、**DESCRIBE** および **EXPLAIN** キーワードをそのような用途に従って使用しますが、MySQL パーサーはこれらを完全にシノニムとして処理します。

- [テーブル構造に関する情報の取得](#)
- [実行計画に関する情報の取得](#)
- [EXPLAIN ANALYZE による情報の取得](#)

テーブル構造に関する情報の取得

DESCRIBE は、テーブル内のカラムに関する情報を提供します。

```
mysql> DESCRIBE City;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| Id    | int(11) | NO | PRI | NULL | auto_increment |
| Name  | char(35) | NO | | | |
| Country | char(3) | NO | UNI | | |
| District | char(20) | YES | MUL | | |
| Population | int(11) | NO | | 0 | |
+-----+-----+-----+-----+-----+-----+
```

DESCRIBE は **SHOW COLUMNS** のショートカットです。これらのステートメントはまた、ビューに関する情報も表示します。**SHOW COLUMNS** の説明では、出力カラムに関してより多くの情報が提供されます。[セクション 13.7.7.5 「SHOW COLUMNS ステートメント」](#) を参照してください。

デフォルトでは、**DESCRIBE** は、そのテーブル内のすべてのカラムに関する情報を表示します。**col_name** (指定されている場合) は、そのテーブル内のカラムの名前です。この場合、このステートメントは、指定されたカラムの情報のみを表示します。**wild** (指定されている場合) は、パターン文字列です。SQL **%** および **_** ワイルドカード文字を含めることができます。この場合、このステートメントは、その文字列に一致する名前を持つカラムの出力のみを表示します。スペースやその他の特殊文字が含まれていないかぎり、この文字列を引用符で囲む必要はありません。

DESCRIBE ステートメントは、Oracle との互換性のために提供されています。

また、[SHOW CREATE TABLE](#)、[SHOW TABLE STATUS](#)、および [SHOW INDEX](#) ステートメントでは、テーブルに関する情報も提供されます。[セクション13.7.7「SHOW ステートメント」](#)を参照してください。

実行計画に関する情報の取得

[EXPLAIN](#) ステートメントは、MySQL がステートメントをどのように実行するかに関する情報を提供します。

- [EXPLAIN](#) は、[SELECT](#)、[DELETE](#)、[INSERT](#)、[REPLACE](#) および [UPDATE](#) ステートメントで動作します。MySQL 8.0.19 以降では、[TABLE](#) ステートメントでも動作します。
- 説明可能なステートメントで [EXPLAIN](#) を使用すると、MySQL は、オプティマイザからのステートメント実行プランに関する情報を表示します。つまり、MySQL はテーブルがどのように、どんな順番で結合されているかに関する情報を含む、ステートメントを処理する方法を説明します。[EXPLAIN](#) を使用して、実行プラン情報を取得することについては、[セクション8.8.2「EXPLAIN 出力フォーマット」](#)を参照してください。
- [EXPLAIN](#) を説明可能なステートメントではなく [FOR CONNECTION connection_id](#) とともに使用すると、名前付き接続で実行されているステートメントの実行計画が表示されます。[セクション8.8.4「名前付き接続の実行計画情報の取得」](#)を参照してください。
- 説明可能なステートメントの場合、[EXPLAIN](#) は、[SHOW WARNINGS](#) を使用して表示できる追加の実行計画情報を生成します。[セクション8.8.3「拡張 EXPLAIN 出力形式」](#)を参照してください。
- [EXPLAIN](#) は、パーティションテーブルを含むクエリーの調査に役立ちます。[セクション24.3.5「パーティションに関する情報を取得する」](#)を参照してください。
- [FORMAT](#) オプションを使用して、出力形式を選択できます。[TRADITIONAL](#) は表形式で出力を表示します。[FORMAT](#) オプションが存在しない場合、これはデフォルトです。[JSON](#) フォーマットは [JSON](#) フォーマットで情報を表示します。MySQL 8.0.16 以降では、[TREE](#) は [TRADITIONAL](#) 形式よりも正確なクエリー処理の説明を含むツリーのような出力を提供します。これは、ハッシュ結合の使用方法を示す唯一の形式 ([セクション8.2.1.4「ハッシュ結合の最適化」](#)を参照) であり、[EXPLAIN ANALYZE](#) に常に使用されます。

[EXPLAIN](#) では、説明されているステートメントの実行に必要な権限と同じ権限が必要です。また、[EXPLAIN](#) には、説明されているビューに対する [SHOW VIEW](#) 権限も必要です。指定した接続が別のユーザーに属している場合、[EXPLAIN ... FOR CONNECTION](#) には [PROCESS](#) 権限も必要です。

[EXPLAIN](#) を使用すると、インデックスを使用して行を検索することでステートメントがより高速に実行されるように、テーブルにインデックスを追加する場所を確認できます。また、[EXPLAIN](#) を使用して、オプティマイザがテーブルを最適な順序で結合しているかどうかを確認することもできます。[SELECT](#) ステートメントでテーブルが指定されている順序に対応する結合順序を使用するように、オプティマイザにヒントを提供するには、ステートメントを [SELECT](#) だけでなく、[SELECT STRAIGHT_JOIN](#) で始めます。([セクション13.2.10「SELECT ステートメント」](#)を参照してください。)

オプティマイザトレースは、[EXPLAIN](#) のトレースを補完する情報を提供する場合があります。ただし、オプティマイザのトレース形式と内容はバージョン間で変更される可能性があります。詳細については、「[MySQL Internals: Tracing the Optimizer](#)」を参照してください。

インデックスが使われるはずであると思うタイミングでそれらが使われていない問題がある場合、[ANALYZE TABLE](#) を実行して、オプティマイザが行う選択に影響する可能性があるキーのカーディナリティーなどのテーブル統計を更新します。[セクション13.7.3.1「ANALYZE TABLE ステートメント」](#)を参照してください。

注記

MySQL Workbench には、[EXPLAIN](#) 出力を視覚的に表現する Visual Explain 機能があります。[Tutorial: Using Explain to Improve Query Performance](#)を参照してください。

EXPLAIN ANALYZE による情報の取得

MySQL 8.0.18 では、[EXPLAIN ANALYZE](#) が導入されています。この [EXPLAIN ANALYZE](#) は、ステートメントを実行し、タイミングおよび追加のイテレータベースの情報とともに、オプティマイザの期待が実際の実行とどのように一致したかに関する [EXPLAIN](#) 出力を生成します。イテレータごとに、次の情報が表示されます:

- 推定実行コスト

(一部のイテレータはコストモデルで考慮されないため、見積りには含まれません。)

- 戻された行の推定数
- 最初の行を返す時間
- すべての行 (実際のコスト) を返す時間 (ミリ秒)

(複数のループがある場合、この図はループ当たりの平均時間を示しています。)

- イテレータによって返された行数
- ループ数

クエリー実行情報は、ノードがイテレータを表す **TREE** 出力形式を使用して表示されます。 **EXPLAIN ANALYZE** では、常に **TREE** 出力形式が使用されます。 MySQL 8.0.21 以降では、これはオプションで **FORMAT=TREE** を使用して明示的に指定できます。 **TREE** 以外の形式はサポートされません。

EXPLAIN ANALYZE は、 **SELECT** ステートメント、複数テーブルの **UPDATE** ステートメントおよび **DELETE** ステートメントとともに使用できます。 MySQL 8.0.19 以降では、 **TABLE** ステートメントでも使用できます。

MySQL 8.0.20 以降、 **KILL QUERY** または **CTRL-C** を使用してこのステートメントを終了できます。

EXPLAIN ANALYZE は、 **FOR CONNECTION** では使用できません。

出力例:

```
mysql> EXPLAIN ANALYZE SELECT * FROM t1 JOIN t2 ON (t1.c1 = t2.c2)\G
***** 1. row *****
EXPLAIN: -> Inner hash join (t2.c2 = t1.c1) (cost=4.70 rows=6)
(actual time=0.032..0.035 rows=6 loops=1)
  -> Table scan on t2 (cost=0.06 rows=6)
(actual time=0.003..0.005 rows=6 loops=1)
  -> Hash
      -> Table scan on t1 (cost=0.85 rows=6)
(actual time=0.018..0.022 rows=6 loops=1)

mysql> EXPLAIN ANALYZE SELECT * FROM t3 WHERE i > 8\G
***** 1. row *****
EXPLAIN: -> Filter: (t3.i > 8) (cost=1.75 rows=5)
(actual time=0.019..0.021 rows=6 loops=1)
  -> Table scan on t3 (cost=1.75 rows=15)
(actual time=0.017..0.019 rows=15 loops=1)

mysql> EXPLAIN ANALYZE SELECT * FROM t3 WHERE pk > 17\G
***** 1. row *****
EXPLAIN: -> Filter: (t3.pk > 17) (cost=1.26 rows=5)
(actual time=0.013..0.016 rows=5 loops=1)
  -> Index range scan on t3 using PRIMARY (cost=1.26 rows=5)
(actual time=0.012..0.014 rows=5 loops=1)
```

出力例で使用されるテーブルは、次に示すステートメントによって作成されています:

```
CREATE TABLE t1 (
  c1 INTEGER DEFAULT NULL,
  c2 INTEGER DEFAULT NULL
);

CREATE TABLE t2 (
  c1 INTEGER DEFAULT NULL,
  c2 INTEGER DEFAULT NULL
);

CREATE TABLE t3 (
  pk INTEGER NOT NULL PRIMARY KEY,
  i INTEGER DEFAULT NULL
);
```

このステートメントの出力で **actual time** に表示される値は、ミリ秒単位で表されます。

13.8.3 HELP ステートメント

HELP 'search_string'

HELP ステートメントは、『MySQL リファレンスマニュアル』からオンライン情報を返します。これが正しく動作するには、mysql データベース内のヘルプテーブルがヘルプトピック情報で初期化されている必要があります ([セクション5.1.17「サーバー側ヘルプのサポート」](#)を参照してください)。

HELP ステートメントは、ヘルプテーブル内の指定された検索文字列を検索し、その検索の結果を表示します。検索文字列では大文字と小文字は区別されません。

検索文字列には、ワイルドカード文字 % および _ を含めることができます。これらは LIKE 演算子で実行されるパターンマッチング演算と同じ意味を持ちます。たとえば、HELP 'rep%' は rep で始まるトピックのリストを返します。

HELP ステートメントは、次のいくつかの種類の検索文字列を理解します。

- もっとも一般的なレベルでは、トップレベルのヘルプカテゴリのリストを取得するには `contents` を使用します。

HELP 'contents'

- `Data Types` などの、特定のヘルプカテゴリ内のトピックのリストを取得するには、そのカテゴリ名を使用します。

HELP 'data types'

- `ASCII()` 関数や `CREATE TABLE` ステートメントなどの、特定のヘルプトピックに関するヘルプを表示するには、関連する 1 つまたは複数のキーワードを使用します。

HELP 'ascii'
HELP 'create table'

つまり、検索文字列はカテゴリ、多数のトピック、または 1 つのトピックに一致します。特定の検索文字列がアイテムのリストを返すか、単一のヘルプトピックのヘルプ情報を返すかを事前に判断することはできません。ただし、結果セット内の行数やカラム数を検査することによって、HELP がどのような種類の応答を返したかがわかります。

次の説明は、結果セットの可能性のある形式を示しています。この例のステートメントの出力は、mysql クライアントの使用時に表示される使い慣れた「tabular」または「vertical」形式を使用して表示されますが、mysql 自体は HELP 結果セットを別の方法で再フォーマットすることに注意してください。

- 空の結果セット

検索文字列に一致するものが見つかりませんでした。

- 3 つのカラムを含む単一行が含まれた結果セット

これは、検索文字列が 1 つのヘルプトピックに一致したことを示します。この結果には 3 つのカラムが含まれています。

- **name:** トピック名。
- **description:** トピックの説明的なヘルプテキスト。
- **example:** 使用例または例。このカラムはブランクである可能性があります。

例: HELP 'replace'

生成される結果:

```
name: REPLACE
description: Syntax:
REPLACE(str,from_str,to_str)
```

```
Returns the string str with all occurrences of the string from_str
replaced by the string to_str. REPLACE() performs a case-sensitive
match when searching for from_str.
example: mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
-> 'WwWwWw.mysql.com'
```


- 2つのカラムを含む複数の行が含まれた結果セット

これは、検索文字列が多数のヘルプトピックに一致したことを示します。この結果セットは、ヘルプトピック名を示します。

- **name**: ヘルプトピック名。
- **is_it_category**: この名前がヘルプカテゴリを表す場合は **Y**、それ以外の場合は **N**。それ以外の場合、**name** 値は、**HELP** ステートメントへの引数として指定されると、指定された項目の説明が含まれた単一行の結果セットを生成するはずですが。

例: **HELP 'status'**

生成される結果:

```
+-----+-----+
| name          | is_it_category |
+-----+-----+
| SHOW          | N              |
| SHOW ENGINE   | N              |
| SHOW MASTER STATUS | N              |
| SHOW PROCEDURE STATUS | N              |
| SHOW SLAVE STATUS | N              |
| SHOW STATUS   | N              |
| SHOW TABLE STATUS | N              |
+-----+-----+
```

- 3つのカラムを含む複数の行が含まれた結果セット

これは、検索文字列がカテゴリに一致したことを示します。この結果セットには、カテゴリエントリが含まれています。

- **source_category_name**: ヘルプカテゴリ名。
- **name**: カテゴリまたはトピック名
- **is_it_category**: この名前がヘルプカテゴリを表す場合は **Y**、それ以外の場合は **N**。それ以外の場合、**name** 値は、**HELP** ステートメントへの引数として指定されると、指定された項目の説明が含まれた単一行の結果セットを生成するはずですが。

例: **HELP 'functions'**

生成される結果:

```
+-----+-----+-----+
| source_category_name | name          | is_it_category |
+-----+-----+-----+
| Functions           | CREATE FUNCTION | N              |
| Functions           | DROP FUNCTION  | N              |
| Functions           | Bit Functions  | Y              |
| Functions           | Comparison operators | Y              |
| Functions           | Control flow functions | Y              |
| Functions           | Date and Time Functions | Y              |
| Functions           | Encryption Functions | Y              |
| Functions           | Information Functions | Y              |
| Functions           | Logical operators | Y              |
| Functions           | Miscellaneous Functions | Y              |
| Functions           | Numeric Functions | Y              |
| Functions           | String Functions | Y              |
+-----+-----+-----+
```

13.8.4 USE ステートメント

USE db_name

USE ステートメントは、指定されたデータベースを後続のステートメントのデフォルト (現行) データベースとして使用するよう MySQL に指示します。このステートメントには、データベースまたはその中のオブジェクトに対するなんらかの権限が必要です。

指定したデータベースは、セッションの終了または別の **USE** ステートメントが発行されるまでデフォルトのままです:

```
USE db1;
SELECT COUNT(*) FROM mytable; # selects from db1.mytable
USE db2;
SELECT COUNT(*) FROM mytable; # selects from db2.mytable
```

データベース名は単一行で指定する必要があります。データベース名の改行はサポートされていません。

USE ステートメントを使用して特定のデータベースをデフォルトにしても、他のデータベースのテーブルへのアクセスは禁止されません。次の例では、**db1** データベースの **author** テーブルと、**db2** データベースの **editor** テーブルにアクセスします。

```
USE db1;
SELECT author_name,editor_name FROM author,db2.editor
WHERE author.editor_id = db2.editor.editor_id;
```

第 14 章 MySQL データディクショナリ

目次

14.1 データディクシヨナリスキーマ	2617
14.2 ファイルベースのメタデータ記憶域の削除	2618
14.3 ディクシヨナリデータのトランザクション記憶域	2619
14.4 ディクシヨナリオブジェクトキャッシュ	2619
14.5 INFORMATION_SCHEMA とデータディクシヨナリの統合	2620
14.6 シリアライズディクシヨナリ情報 (SDI)	2622
14.7 データディクシヨナリの使用方法の違い	2622
14.8 データディクシヨナリの制限事項	2624

MySQL Server には、データベースオブジェクトに関する情報を格納するトランザクションデータディクシヨナリが組み込まれています。以前の MySQL リリースでは、ディクシヨナリデータはメタデータファイル、非トランザクションテーブルおよびストレージエンジン固有のデータディクシヨナリに格納されていました。

この章では、データディクシヨナリの主な機能、利点、使用上の違いおよび制限について説明します。データディクシヨナリ機能のその他の影響については、「MySQL 8.0 リリースノート」の「データディクシヨナリのノート」のセクションを参照してください。

MySQL データディクシヨナリの利点は次のとおりです:

- ディクシヨナリデータを均一に格納する集中化されたデータディクシヨナリスキーマの単純性。 [セクション 14.1「データディクシヨナリスキーマ」](#)を参照してください。
- ファイルベースのメタデータ記憶域の削除。 [セクション 14.2「ファイルベースのメタデータ記憶域の削除」](#)を参照してください。
- ディクシヨナリデータのトランザクション対応のクラッシュセーフ記憶域。 [セクション 14.3「ディクシヨナリデータのトランザクション記憶域」](#)を参照してください。
- ディクシヨナリオブジェクトの共通キャッシュおよび集中キャッシュ。 [セクション 14.4「ディクシヨナリオブジェクトキャッシュ」](#)を参照してください。
- 一部の `INFORMATION_SCHEMA` テーブルの実装が単純かつ改善されました。 [セクション 14.5「INFORMATION_SCHEMA とデータディクシヨナリの統合」](#)を参照してください。
- アトミック DDL。 [セクション 13.1.1「アトミックデータ定義ステートメントのサポート」](#)を参照してください。

重要

データディクシヨナリが有効なサーバーには、データディクシヨナリがないサーバーと比較して、一般的な操作上の違いがいくつかあります。 [セクション 14.7「データディクシヨナリ使用方法の違い」](#)を参照してください。また、MySQL 8.0 へのアップグレードの場合、アップグレード手順は以前の MySQL リリースと多少異なり、特定の前提条件を確認してインストールのアップグレード準備状況を確認する必要があります。詳細は、 [セクション 2.11「MySQL のアップグレード」](#) (特に [セクション 2.11.5「アップグレード用のインストールの準備」](#)) を参照してください。

14.1 データディクシヨナリスキーマ

データディクシヨナリテーブルは保護されており、MySQL のデバッグビルドでのみアクセスできます。ただし、MySQL では、 `INFORMATION_SCHEMA` テーブルおよび `SHOW` ステートメントを介したデータディクシヨナリテーブルに格納されたデータへのアクセスがサポートされています。データディクシヨナリを構成するテーブルの概要は、 [データディクシヨナリテーブル](#) を参照してください。

MySQL システムテーブルは MySQL 8.0 にまだ存在し、 `mysql` システムデータベースで `SHOW TABLES` ステートメントを発行することで表示できます。通常、MySQL データディクシヨナリテーブルとシステムテーブルの違いは、

データディクショナリテーブルには SQL クエリーの実行に必要なメタデータが含まれるのに対し、システムテーブルにはタイムゾーンやヘルプ情報などの補助データが含まれることです。MySQL システムテーブルとデータディクショナリテーブルも、アップグレード方法が異なります。MySQL サーバーは、データディクショナリのアップグレードを管理します。SQL サーバー。 [データディクショナリのアップグレード方法を参照してください](#)。MySQL システムテーブルをアップグレードするには、完全な MySQL アップグレード手順を実行する必要があります。 [セクション 2.11.3「MySQL のアップグレードプロセスの内容」](#) を参照してください。

データディクショナリのアップグレード方法

MySQL の新しいバージョンには、データディクショナリテーブル定義の変更が含まれる場合があります。このような変更は、新しくインストールされたバージョンの MySQL に存在しますが、MySQL バイナリのインプレースアップグレードを実行すると、新しいバイナリを使用して MySQL サーバーを再起動したときに変更が適用されます。起動時に、データディクショナリテーブルをアップグレードする必要があるかどうかを判断するために、サーバーのデータディクショナリのバージョンがデータディクショナリに格納されているバージョン情報と比較されます。アップグレードが必要でサポートされている場合、サーバーは、更新された定義を含むデータディクショナリテーブルを作成し、永続化されたメタデータを新しいテーブルにコピーし、古いテーブルを新しいテーブルに原子的に置き換え、データディクショナリを再初期化します。アップグレードが不要な場合、データディクショナリテーブルを更新せずに起動が続行されます。

データディクショナリテーブルのアップグレードはアトミック操作で、すべてのデータディクショナリテーブルが必要に応じてアップグレードされるか、操作が失敗することを意味します。アップグレード操作が失敗した場合、サーバーの起動はエラーで失敗します。この場合、古いサーバーバイナリを古いデータディレクトリとともに使用してサーバーを起動できます。新しいサーバーバイナリを再度使用してサーバーを起動すると、データディクショナリのアップグレードが再試行されます。

通常、データディクショナリテーブルが正常にアップグレードされた後は、古いサーバーバイナリを使用してサーバーを再起動することはできません。その結果、データディクショナリテーブルのアップグレード後、MySQL サーバーバイナリの以前の MySQL バージョンへのダウングレードはサポートされません。

`mysqld --no-dd-upgrade` オプションを使用すると、起動時のデータディクショナリテーブルの自動アップグレードを回避できます。 `--no-dd-upgrade` が指定され、サーバーのデータディクショナリのバージョンがデータディクショナリに格納されているバージョンと異なることがサーバーで検出されると、データディクショナリのアップグレードが禁止されていることを示すエラーで起動が失敗します。

MySQL のデバッグビルドを使用したデータディクショナリテーブルの表示

データディクショナリテーブルはデフォルトで保護されていますが、デバッグサポートを使用して (`-DWITH_DEBUG=1 CMake` オプションを使用して)MySQL をコンパイルし、`+d,skip_dd_table_access_check debug` オプションおよび修飾子を指定することでアクセスできます。デバッグビルドのコンパイルの詳細は、 [セクション 5.9.1.1「デバッグのための MySQL のコンパイル」](#) を参照してください。

警告

データディクショナリテーブルを直接変更または書き込むことはお薦めしません。また、MySQL インスタンスが動作不能になる可能性があります。

デバッグサポート付きで MySQL をコンパイルした後、次の `SET` ステートメントを使用して、`mysql` クライアントセッションでデータディクショナリテーブルを表示できるようにします:

```
mysql> SET SESSION debug='+d,skip_dd_table_access_check';
```

データディクショナリテーブルのリストを取得するには、次のクエリーを使用します:

```
mysql> SELECT name, schema_id, hidden, type FROM mysql.tables where schema_id=1 AND hidden='System';
```

データディクショナリテーブル定義を表示するには、`SHOW CREATE TABLE` を使用します。例:

```
mysql> SHOW CREATE TABLE mysql.catalogs\G
```

14.2 ファイルベースのメタデータ記憶域の削除

以前の MySQL リリースでは、ディクショナリデータは部分的にメタデータファイルに格納されていました。ファイルベースのメタデータ記憶域の問題には、高価なファイルスキャン、ファイルシステム関連の不具合の疑わしい

問題、レプリケーションおよびクラッシュリカバリの失敗状態を処理するための複雑なコード、新機能およびリレーショナルオブジェクトのメタデータの追加を困難にする拡張性の欠如などがあります。

次に示すメタデータファイルが MySQL から削除されます。特に明記されていないが、メタデータファイルに以前格納されたデータはデータディクショナリテーブルに格納されるようになりました。

- **.frm** ファイル: テーブルメタデータファイル。 **.frm** ファイルを削除すると、次のようになります:
 - **.frm** ファイル構造によって課される 64KB のテーブル定義サイズ制限が削除されます。
 - **INFORMATION_SCHEMA.TABLES VERSION** カラムには、MySQL 5.7 で最後に使用された **.frm** ファイルバージョンである **10** のハードコードされた値がレポートされます。
- **.par** ファイル: パーティション定義ファイル。 MySQL 5.7 のパーティション定義ファイルを使用して InnoDB が停止し、InnoDB テーブルのネイティブパーティション化サポートが導入されました。
- **.TRN** ファイル: トリガーネームスペースファイル。
- **.TRG** ファイル: トリガーパラメータファイル。
- **.isl** ファイル: データディレクトリの外部で作成された **file-per-table** テーブルスペースファイルの場所を含む InnoDB シンボリックリンクファイル。
- **db.opt** ファイル: データベース構成ファイル。これらのファイル(データベースディレクトリごとに1つ)には、データベースのデフォルト文字セット属性が含まれていました。
- **ddl_log.log** ファイル: ファイルには、**DROP TABLE** や **ALTER TABLE** などのデータ定義ステートメントによって生成されたメタデータ操作のレコードが含まれていました。

14.3 ディクショナリデータのトランザクション記憶域

データディクショナリビューは、トランザクション (InnoDB) テーブルにディクショナリデータを格納します。データディクショナリテーブルは、非データディクショナリシステムテーブルとともに **mysql** データベースに配置されます。

データディクショナリテーブルは、MySQL データディレクトリにある **mysql.ibd** という名前の単一の InnoDB テーブルスペースで作成されます。 **mysql.ibd** テーブルスペースファイルは MySQL データディレクトリに存在する必要があるため、その名前を変更したり、別のテーブルスペースで使用することはできません。

ディクショナリデータは、InnoDB テーブルに格納されているユーザーデータを保護する同じコミット、ロールバックおよびクラッシュリカバリ機能によって保護されます。

14.4 ディクショナリオブジェクトキャッシュ

ディクショナリオブジェクトキャッシュは、以前にアクセスしたデータディクショナリオブジェクトをメモリーに格納して、オブジェクトの再利用を可能にし、ディスク I/O を最小限に抑える共有グローバルキャッシュです。MySQL で使用される他のキャッシュメカニズムと同様に、ディクショナリオブジェクトキャッシュでは LRU ベースの削除戦略を使用して、メモリーから最も使用頻度の低いオブジェクトを削除します。

ディクショナリオブジェクトキャッシュは、様々なオブジェクト型を格納するキャッシュパーティションで構成されます。一部のキャッシュパーティションサイズ制限は構成可能ですが、その他の制限はハードコードされています。

- **テーブルスペース定義キャッシュパーティション**: テーブルスペース定義オブジェクトを格納します。 **tablespace_definition_cache** オプションでは、ディクショナリオブジェクトキャッシュに格納できるテーブルスペース定義オブジェクトの数の制限を設定します。デフォルト値は 256 です。
- **スキーマ定義キャッシュパーティション**: スキーマ定義オブジェクトを格納します。 **schema_definition_cache** オプションでは、ディクショナリオブジェクトキャッシュに格納できるスキーマ定義オブジェクトの数の制限を設定します。デフォルト値は 256 です。
- **テーブル定義キャッシュパーティション**: テーブル定義オブジェクトを格納します。オブジェクト制限は **max_connections** の値に設定され、デフォルト値は 151 です。

テーブル定義キャッシュパーティションは、[table_definition_cache](#) 構成オプションを使用して構成されたテーブル定義キャッシュと並行して存在します。どちらのキャッシュにもテーブル定義が格納されますが、MySQL サーバーの様々な部分に対応します。一方のキャッシュ内のオブジェクトは、他方のキャッシュ内のオブジェクトの存在に依存しません。

- ストアドプログラム定義キャッシュパーティション: ストアドプログラム定義オブジェクトを格納します。[stored_program_definition_cache](#) オプションでは、ディクショナリオブジェクトキャッシュに格納できるストアドプログラム定義オブジェクトの数の制限を設定します。デフォルト値は 256 です。

ストアドプログラム定義キャッシュパーティションは、[stored_program_cache](#) オプションを使用して構成されたストアドプロシージャおよびストアドファンクションキャッシュと並行して存在します。

[stored_program_cache](#) オプションは、接続ごとにキャッシュされるストアドプロシージャまたはファンクションの数の弱い上限を設定し、接続がストアドプロシージャまたはファンクションを実行するたびに制限がチェックされます。一方、ストアドプログラム定義キャッシュパーティションは、他の目的でストアドプログラム定義オブジェクトを格納する共有キャッシュです。ストアドプログラム定義キャッシュパーティションにオブジェクトが存在しても、ストアドプロシージャキャッシュまたはストアドファンクションキャッシュにオブジェクトが存在するかどうかには依存しません。その逆も同様です。

- 文字セット定義キャッシュパーティション: 文字セット定義オブジェクトを格納し、256 のハードコードされたオブジェクト制限を持ちます。
- 照合定義キャッシュパーティション: 照合定義オブジェクトを格納し、256 のハードコードされたオブジェクト制限を持ちます。

ディクショナリオブジェクトキャッシュ構成オプションの有効な値の詳細は、[セクション5.1.8「サーバースystem変数」](#)を参照してください。

14.5 INFORMATION_SCHEMA とデータディクショナリの統合

データディクショナリの導入により、次の [INFORMATION_SCHEMA](#) テーブルがデータディクショナリテーブルのビューとして実装されます:

- [CHARACTER_SETS](#)
- [CHECK_CONSTRAINTS](#)
- [COLLATIONS](#)
- [COLLATION_CHARACTER_SET_APPLICABILITY](#)
- [COLUMNS](#)
- [COLUMN_STATISTICS](#)
- [EVENTS](#)
- [FILES](#)
- [INNODB_COLUMNS](#)
- [INNODB_DATAFILES](#)
- [INNODB_FIELDS](#)
- [INNODB_FOREIGN](#)
- [INNODB_FOREIGN_COLS](#)
- [INNODB_INDEXES](#)
- [INNODB_TABLES](#)
- [INNODB_TABLESPACES](#)
- [INNODB_TABLESPACES_BRIEF](#)

- [INNODB_TABLESTATS](#)
- [KEY_COLUMN_USAGE](#)
- [KEYWORDS](#)
- [PARAMETERS](#)
- [PARTITIONS](#)
- [REFERENTIAL_CONSTRAINTS](#)
- [RESOURCE_GROUPS](#)
- [ROUTINES](#)
- [SCHEMATA](#)
- [STATISTICS](#)
- [ST_GEOMETRY_COLUMNS](#)
- [ST_SPATIAL_REFERENCE_SYSTEMS](#)
- [TABLES](#)
- [TABLE_CONSTRAINTS](#)
- [TRIGGERS](#)
- [VIEWS](#)
- [VIEW_ROUTINE_USAGE](#)
- [VIEW_TABLE_USAGE](#)

これらのテーブルに対するクエリーは、他の低速な方法ではなくデータディクショナリテーブルから情報を取得するため、より効率的になりました。特に、データディクショナリテーブルのビューである [INFORMATION_SCHEMA](#) テーブルごとに、次のようにします:

- サーバーは、[INFORMATION_SCHEMA](#) テーブルのクエリーごとに一時テーブルを作成する必要がなくなりました。
- 基礎となるデータディクショナリテーブルに、ディレクトリスキャンによって以前に取得された値 (データベース名やデータベース内のテーブル名を列挙する場合など) またはファイルオープン操作 (.frm ファイルから情報を取り出す場合など) が格納されている場合、これらの値に対する [INFORMATION_SCHEMA](#) クエリーでは、かわりにテーブルルックアップが使用されるようになりました。(また、ビュー以外の [INFORMATION_SCHEMA](#) テーブルの場合でも、データベース名やテーブル名などの値はデータディクショナリから参照によって取得されるため、ディレクトリまたはファイルのスキャンは必要ありません。)
- 基礎となるデータディクショナリテーブルのインデックスを使用すると、オプティマイザは、クエリーごとに一時テーブルを使用して [INFORMATION_SCHEMA](#) テーブルを処理した以前の実装に当てはまらない効率的なクエリー実行計画を作成できます。

前述の改善は、データディクショナリテーブルのビューである [INFORMATION_SCHEMA](#) テーブルに対応する情報を表示する [SHOW](#) ステートメントにも適用されます。たとえば、[SHOW DATABASES](#) では、[SCHEMATA](#) テーブルと同じ情報が表示されます。

データディクショナリテーブルのビューの導入に加えて、[STATISTICS](#) テーブルおよび [TABLES](#) テーブルに含まれるテーブル統計がキャッシュされ、[INFORMATION_SCHEMA](#) クエリーのパフォーマンスが向上しました。[information_schema_stats_expiry](#) システム変数は、キャッシュされたテーブルの統計が期限切れになるまでの期間を定義します。デフォルトは 86400 秒 (24 時間) です。キャッシュされた統計がない場合、または統計が期限切れになっている場合は、テーブル統計カラムのクエリー時にストレージエンジンから統計が取得されます。特定のテーブルのキャッシュされた値をいつでも更新するには、[ANALYZE TABLE](#) を使用

[information_schema_stats_expiry](#) を 0 に設定すると、[INFORMATION_SCHEMA](#) クエリーでストレージエンジンから直接最新の統計を取得できます。これは、キャッシュされた統計の取得ほど高速ではありません。

詳細は、[セクション8.2.3「INFORMATION_SCHEMA クエリーの最適化」](#)を参照してください。

MySQL 8.0 の [INFORMATION_SCHEMA](#) テーブルはデータディクショナリに密接に関連付けられているため、いくつかの使用方法が異なります。[セクション14.7「データディクショナリの使用法の違い」](#)を参照してください。

14.6 シリアライズディクショナリ情報 (SDI)

データベースオブジェクトに関するメタデータをデータディクショナリに格納する以外に、MySQL ではシリアライズされた形式で格納します。このデータは、シリアライズディクショナリ情報 (SDI) と呼ばれます。[InnoDB](#) では、SDI データはテーブルスペースファイル内に格納されます。[NDBCLUSTER](#) は、SDI データを NDB ディクショナリに格納します。その他のストレージエンジンは、テーブルデータベースディレクトリ内の特定のテーブルに対して作成された SDI データを `.sdi` ファイルに格納します。SDI データは、コンパクトな [JSON](#) 形式で生成されます。

シリアライズディクショナリ情報 (SDI) は、一時テーブルスペースおよび undo テーブルスペースファイルを除くすべての [InnoDB](#) テーブルスペースファイルに存在します。[InnoDB](#) テーブルスペースファイルの SDI レコードには、テーブルスペース内に含まれるテーブルおよびテーブルスペースオブジェクトのみが記述されます。

SDI データは、テーブルまたは [CHECK TABLE FOR UPGRADE](#) に対する DDL 操作によって更新されます。SDI データは、MySQL サーバーを新しいリリースまたはバージョンにアップグレードしても更新されません。

SDI データが存在すると、メタデータの冗長性が提供されます。たとえば、データディクショナリが使用できなくなった場合、`ibd2sdi` ツールを使用して、オブジェクトメタデータを [InnoDB](#) テーブルスペースファイルから直接抽出できます。

[InnoDB](#) の場合、SDI レコードには単一のインデックスページ (デフォルトでは 16KB) が必要です。ただし、SDI データは、ストレージフットプリントを削減するために圧縮されます。

複数のテーブルスペースで構成されるパーティション化された [InnoDB](#) テーブルの場合、SDI データは最初のパーティションのテーブルスペースファイルに格納されます。

MySQL サーバーは、DDL 操作中にアクセスされる内部 API を使用して SDI レコードを作成および保守します。

`IMPORT TABLE` ステートメントは、`.sdi` ファイルに含まれる情報に基づいて [MyISAM](#) テーブルをインポートします。詳細は、[セクション13.2.5「IMPORT TABLE ステートメント」](#)を参照してください。

14.7 データディクショナリの使用法の違い

データディクショナリが有効な MySQL サーバーの使用には、データディクショナリがないサーバーと比較して、いくつかの操作上の違いがあります:

- 以前は、`innodb_read_only` システム変数を有効にすると、[InnoDB](#) ストレージエンジンのテーブルの作成および削除のみができなくなりました。MySQL 8.0 の時点では、`innodb_read_only` を有効にすると、すべてのストレージエンジンでこれらの操作が防止されます。ストレージエンジンのテーブルの作成および削除操作では、`mysql` システムデータベース内のデータディクショナリテーブルが変更されますが、これらのテーブルは [InnoDB](#) ストレージエンジンを使用するため、`innodb_read_only` が有効になっている場合は変更できません。データディクショナリテーブルの変更を必要とする他のテーブル操作にも、同じ原則が適用されます。例:
 - データディクショナリに格納されているテーブル統計が更新されるため、`ANALYZE TABLE` は失敗します。
 - データディクショナリに格納されているストレージエンジンの指定が更新されるため、`ALTER TABLE tbl_name ENGINE=engine_name` は失敗します。

注記

`innodb_read_only` を有効にすると、`mysql` システムデータベースのデータディクショナリ以外のテーブルにも重要な影響があります。詳細は、[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)の `innodb_read_only` の説明を参照してください

- 以前は、`mysql` システムデータベース内のテーブルは DML ステートメントおよび DDL ステートメントから参照できました。MySQL 8.0 では、データディクショナリテーブルは非表示であり、直接変更またはクエリーできません。ただし、ほとんどの場合、かわりにクエリーすることができる対応する [INFORMATION_SCHEMA](#) テーブルがあります。これにより、アプリケーションで使用する安定した [INFORMATION_SCHEMA](#) インタフェースを維持しながら、基礎となるデータディクショナリテーブルをサーバー開発の進行中に更新できます。

- MySQL 8.0 の `INFORMATION_SCHEMA` テーブルはデータディクショナリに密接に関連付けられているため、いくつかの使用法が異なります:
 - 以前は、`INFORMATION_SCHEMA` は `STATISTICS` および `TABLES` テーブルのテーブル統計をクエリーして、ストレージエンジンから直接統計を取得していました。MySQL 8.0 では、キャッシュされたテーブルの統計がデフォルトで使用されます。`information_schema_stats_expiry` システム変数は、キャッシュされたテーブルの統計が期限切れになるまでの期間を定義します。デフォルトは 86400 秒 (24 時間) です。(特定のテーブルのキャッシュされた値をいつでも更新するには、`ANALYZE TABLE` を使用します。) キャッシュされた統計がない場合、または統計が期限切れになっている場合は、テーブル統計カラムのクエリー時にストレージエンジンから統計が取得されます。常に最新の統計をストレージエンジンから直接取得するには、`information_schema_stats_expiry` を 0 に設定します。詳細は、[セクション 8.2.3 「INFORMATION_SCHEMA クエリーの最適化」](#) を参照してください。
 - いくつかの `INFORMATION_SCHEMA` テーブルはデータディクショナリテーブルのビューであり、オプティマイザはこれらの基礎となるテーブルでインデックスを使用できます。したがって、オプティマイザの選択に応じて、`INFORMATION_SCHEMA` クエリーの結果の行順序が以前の結果と異なる場合があります。クエリー結果に特定の行順序付け特性が必要な場合は、`ORDER BY` 句を含めます。
 - `INFORMATION_SCHEMA` テーブルに対するクエリーでは、以前の MySQL シリーズとは異なる大文字と小文字でカラム名が返される場合があります。アプリケーションでは、大/小文字を区別しない方法で結果セットのカラム名をテストする必要があります。実行できない場合は、必要な文字のカラム名を返す選択リストでカラムのエイリアスを使用します。例:

```
SELECT TABLE_SCHEMA AS table_schema, TABLE_NAME AS table_name
FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME = 'users';
```

- `mysqldump` および `mysqlpump` は、コマンドラインで明示的に指定されていても、`INFORMATION_SCHEMA` データベースをダンプしなくなりました。
- `CREATE TABLE dst_tbl LIKE src_tbl` では、`src_tbl` が実テーブルである必要があり、データディクショナリテーブルのビューである `INFORMATION_SCHEMA` テーブルである場合は失敗します。
- 以前は、`INFORMATION_SCHEMA` テーブルから選択されたカラムの結果セットヘッダーでは、クエリーで指定された大/小文字が使用されていました。このクエリーでは、`table_name` のヘッダーを含む結果セットが生成されます:

```
SELECT table_name FROM INFORMATION_SCHEMA.TABLES;
```

MySQL 8.0 では、これらのヘッダーは大文字になります。前述のクエリーでは、`TABLE_NAME` のヘッダーを含む結果セットが生成されます。必要に応じて、カラムのエイリアスを使用して別の大文字と小文字を実現できます。例:

```
SELECT table_name AS 'table_name' FROM INFORMATION_SCHEMA.TABLES;
```

- データディレクトリは、`mysqldump` および `mysqlpump` が `mysql` システムデータベースから情報をダンプする方法に影響します:
 - 以前は、`mysql` システムデータベース内のすべてのテーブルをダンプできました。MySQL 8.0 の時点では、`mysqldump` および `mysqlpump` はそのデータベース内のデータディクショナリ以外のテーブルのみをダンプします。
 - 以前は、`--routines` および `--events` オプションは、`--all-databases` オプションの使用時にストアルーチンおよびイベントを含める必要はありませんでした: ダンプには `mysql` システムデータベースが含まれていたため、ストアルーチンおよびイベント定義を含む `proc` および `event` テーブルも含まれていました。MySQL 8.0 では、`event` テーブルおよび `proc` テーブルは使用されません。対応するオブジェクトの定義はデータディクショナリテーブルに格納されますが、これらのテーブルはダンプされません。`--all-databases` を使用して作成されたダンプにストアルーチンおよびイベントを含めるには、`--routines` および `--events` オプションを明示的に使用します。
 - 以前は、`--routines` オプションに `proc` テーブルに対する `SELECT` 権限が必要でした。MySQL 8.0 では、このテーブルは使用されません。かわりに、`--routines` にはグローバル `SELECT` 権限が必要です。

- 以前は、`proc` および `event` テーブルをダンプすることで、ストアルーチンおよびイベント定義をそれらの作成および変更タイムスタンプとともにダンプできました。MySQL 8.0 では、これらのテーブルは使用されないため、タイムスタンプをダンプすることはできません。
- 以前は、不正な文字を含むストアルーチンを作成すると、警告が生成されました。MySQL 8.0 では、これはエラーです。

14.8 データディクショナリの制限事項

このセクションでは、MySQL データディクショナリで導入された一時的な制限について説明します。

- データディレクトリ (`mkdir` など) の下でのデータベースディレクトリの手動作成はサポートされていません。手動で作成されたデータベースディレクトリは、MySQL Server で認識されません。
- `.frm` ファイルのかわりに記憶域、undo ログおよび redo ログに書き込むため、DDL 操作に時間がかかります。

第 15 章 InnoDB ストレージエンジン

目次

15.1 InnoDB 入門	2626
15.1.1 InnoDB テーブルを使用する利点	2628
15.1.2 InnoDB テーブルのベストプラクティス	2629
15.1.3 InnoDB がデフォルトのストレージエンジンであるかどうかの確認	2629
15.1.4 InnoDB を使用したテストおよびベンチマーク	2629
15.2 InnoDB および ACID モデル	2630
15.3 InnoDB マルチバージョン	2631
15.4 InnoDB のアーキテクチャ	2632
15.5 InnoDB インメモリー構造	2633
15.5.1 バッファプール	2633
15.5.2 変更バッファ	2638
15.5.3 適応型ハッシュインデックス	2641
15.5.4 ログバッファ	2642
15.6 InnoDB オンディスク構造	2642
15.6.1 テーブル	2642
15.6.2 インデックス	2665
15.6.3 テーブルスペース	2672
15.6.4 二重書き込みバッファ	2693
15.6.5 redo ログ	2695
15.6.6 undo ログ	2699
15.7 InnoDB のロックおよびトランザクションモデル	2700
15.7.1 InnoDB ロック	2701
15.7.2 InnoDB トランザクションモデル	2705
15.7.3 InnoDB のさまざまな SQL ステートメントで設定されたロック	2713
15.7.4 ファントム行	2716
15.7.5 InnoDB のデッドロック	2717
15.7.6 トランザクションスケジューリング	2720
15.8 InnoDB の構成	2721
15.8.1 InnoDB の起動構成	2721
15.8.2 読み取り専用操作の InnoDB の構成	2727
15.8.3 InnoDB バッファプールの構成	2728
15.8.4 InnoDB のスレッド並列性の構成	2741
15.8.5 InnoDB バックグラウンド I/O スレッドの数の構成	2742
15.8.6 Linux での非同期 I/O の使用	2742
15.8.7 InnoDB I/O Capacity の構成	2743
15.8.8 スピンロックのポーリングの構成	2744
15.8.9 ページ構成	2745
15.8.10 InnoDB のオプティマイザ統計の構成	2747
15.8.11 インデックスページのマージしきい値の構成	2756
15.8.12 専用 MySQL Server の自動構成の有効化	2759
15.9 InnoDB のテーブルおよびページの圧縮	2761
15.9.1 InnoDB テーブルの圧縮	2761
15.9.2 InnoDB ページ圧縮	2774
15.10 InnoDB の行フォーマット	2777
15.11 InnoDB のディスク I/O とファイル領域管理	2783
15.11.1 InnoDB ディスク I/O	2783
15.11.2 ファイル領域管理	2784
15.11.3 InnoDB チェックポイント	2785
15.11.4 テーブルのデフラグ	2785
15.11.5 TRUNCATE TABLE によるディスク領域の再利用	2786
15.12 InnoDB とオンライン DDL	2786
15.12.1 オンライン DDL 操作	2787
15.12.2 オンライン DDL のパフォーマンスと同時実行性	2799

15.12.3	オンライン DDL 領域の要件	2802
15.12.4	オンライン DDL を使用した DDL ステートメントの簡略化	2803
15.12.5	オンライン DDL 失敗条件	2803
15.12.6	オンライン DDL の制限事項	2804
15.13	InnoDB 保存データ暗号化	2805
15.14	InnoDB の起動オプションおよびシステム変数	2813
15.15	InnoDB INFORMATION_SCHEMA テーブル	2894
15.15.1	圧縮に関する InnoDB INFORMATION_SCHEMA テーブル	2895
15.15.2	InnoDB INFORMATION_SCHEMA トランザクションおよびロック情報	2896
15.15.3	InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル	2903
15.15.4	InnoDB INFORMATION_SCHEMA FULLTEXT インデックステーブル	2908
15.15.5	InnoDB INFORMATION_SCHEMA バッファプールテーブル	2911
15.15.6	InnoDB INFORMATION_SCHEMA メトリックテーブル	2915
15.15.7	InnoDB INFORMATION_SCHEMA 一時テーブル情報テーブル	2922
15.15.8	INFORMATION_SCHEMA.FILES からの InnoDB テーブルスペースメタデータの取得	2923
15.16	InnoDB の MySQL パフォーマンススキーマとの統合	2924
15.16.1	パフォーマンススキーマを使用した InnoDB テーブルの ALTER TABLE の進行状況のモニタリング	2926
15.16.2	パフォーマンススキーマを使用した InnoDB Mutex 待機のモニタリング	2928
15.17	InnoDB モニター	2931
15.17.1	InnoDB モニターのタイプ	2931
15.17.2	InnoDB モニターの有効化	2932
15.17.3	InnoDB 標準モニターおよびロックモニターの出力	2933
15.18	InnoDB のバックアップとリカバリ	2937
15.18.1	InnoDB バックアップ	2938
15.18.2	InnoDB のリカバリ	2938
15.19	InnoDB と MySQL レプリケーション	2941
15.20	InnoDB memcached プラグイン	2942
15.20.1	InnoDB memcached プラグインの利点	2943
15.20.2	InnoDB memcached のアーキテクチャー	2944
15.20.3	InnoDB memcached プラグインの設定	2947
15.20.4	InnoDB memcached の複数の get および Range クエリーのサポート	2952
15.20.5	InnoDB memcached プラグインのセキュリティーに関する考慮事項	2954
15.20.6	InnoDB memcached プラグイン用のアプリケーションの記述	2956
15.20.7	InnoDB memcached プラグインとレプリケーション	2967
15.20.8	InnoDB memcached プラグインの内部	2970
15.20.9	InnoDB memcached プラグインのトラブルシューティング	2974
15.21	InnoDB のトラブルシューティング	2976
15.21.1	InnoDB の I/O に関する問題のトラブルシューティング	2977
15.21.2	InnoDB のリカバリの強制的な実行	2977
15.21.3	InnoDB データディクショナリの操作のトラブルシューティング	2979
15.21.4	InnoDB のエラー処理	2980
15.22	InnoDB の制限	2980
15.23	InnoDB の制限および制限事項	2981

15.1 InnoDB 入門

InnoDB は、高い信頼性と高いパフォーマンスとのバランスをとる汎用のストレージエンジンです。MySQL 8.0 では、InnoDB がデフォルトの MySQL ストレージエンジンです。別のデフォルトのストレージエンジンを構成していないがぎり、ENGINE= 句を指定せずに CREATE TABLE ステートメントを発行すると、InnoDB テーブルが作成されます。

InnoDB の主要な利点

- その DML 操作は、トランザクションにユーザーデータを保護するためのコミット、ロールバック、およびクラッシュリカバリ機能が備わっている ACID モデルに従っています。詳しくは[セクション15.2「InnoDB および ACID モデル」](#)をご覧ください。
- 行レベルのロックと Oracle スタイルの一貫性読み取りを使用すると、複数ユーザーの並列性およびパフォーマンスが向上します。詳しくは[セクション15.7「InnoDB のロックおよびトランザクションモデル」](#)をご覧ください。

- InnoDB テーブルでは、**主キー**に基づいてクエリーが最適化されるように、ディスク上のデータが整列されます。各 InnoDB テーブルには、主キー検索用の I/O を最小限に抑えるためにデータを編成する **clustered index** と呼ばれる主キーインデックスがあります。詳しくは[セクション15.6.2.1「クラスタインデックスとセカンダリインデックス」](#)をご覧ください。
- データ **integrity** をメンテナンスするために、InnoDB では **FOREIGN KEY** 制約がサポートされています。外部キーでは、挿入、更新および削除がチェックされ、異なるテーブル間で不整合が発生しないことが確認されます。詳しくは[セクション13.1.20.5「FOREIGN KEY の制約」](#)をご覧ください。

表 15.1 「InnoDB ストレージエンジンの機能」

機能	Support
B ツリーインデックス	はい
MVCC	はい
T ツリーインデックス	いいえ
インデックスキャッシュ	はい
クラスタデータベースのサポート	いいえ
クラスタ化されたインデックス	はい
ストレージの制限	64TB
データキャッシュ	はい
データディクショナリ向け更新統計	はい
トランザクション	はい
ハッシュインデックス	いいえ (InnoDB は、アダプティブハッシュインデックス機能に対して、内部的にハッシュインデックスを利用します。)
バックアップ/ポイントインタイムリカバリ (ストレージエンジン内ではなくサーバー内で実装されています。)	はい
レプリケーションのサポート (ストレージエンジン内ではなくサーバー内で実装されています。)	はい
ロック粒度	行
全文検索インデックス	はい (FULLTEXT インデックスに対する InnoDB サポートは、MySQL 5.6 以降で使用できます。)
圧縮データ	はい
地理空間インデックスのサポート	はい (InnoDB での地理空間インデックス付けのサポートは、MySQL 5.7 以降で使用できます。)
地理空間データ型のサポート	はい
外部キーのサポート	はい
暗号化データ	はい (暗号化機能を介してサーバーに実装されません。MySQL 5.7 以降では、保存データのテーブルスペース暗号化がサポートされます。)

InnoDB の機能と MySQL で提供されている他のストレージエンジンを比較する方法については、[第16章「代替ストレージエンジン」](#)の「ストレージエンジンの機能」表を参照してください。

InnoDB の拡張機能と新機能

InnoDB の拡張機能および新機能の詳細は、次を参照してください:

- [セクション1.3「MySQL 8.0 の新機能」](#) の InnoDB 拡張機能のリスト。
- 「[リリースノート](#)」。

追加の InnoDB 情報およびリソース

- InnoDB 関連の用語および定義については、[MySQL 用語集](#) を参照してください。
- InnoDB ストレージエンジン専用のフォーラムについては、[MySQL Forums::InnoDB](#) を参照してください。
- InnoDB は、MySQL と同じ GNU GPL ライセンスバージョン 2 (1991 年 6 月) によって発行されています。MySQL ライセンスの詳細は、<http://www.mysql.com/company/legal/licensing/> を参照してください。

15.1.1 InnoDB テーブルを使用する利点

InnoDB テーブルは、次の理由で役立ちます:

- ハードウェアまたはソフトウェアの問題が原因でサーバーが予期せず終了した場合は、その時点でデータベースで何が起こっていたかに関係なく、データベースの再起動後に特別な操作を行う必要はありません。InnoDB の [クラッシュリカバリ](#) を使用すると自動的に、クラッシュ時の前にコミットされた変更はすべて完了し、処理中だったがコミットされなかった変更はすべて取り消されます。単に再起動し、終了した場所から続行するだけです。
- InnoDB ストレージエンジンは、データがアクセスされたときにテーブルおよびインデックスデータをメインメモリーにキャッシュする独自の [buffer pool](#) を保持します。頻繁に使用されるデータは、直接メモリーから処理されます。このキャッシュは多くのタイプの情報に適用され、処理が高速化されます。専用データベースサーバーでは、多くの場合、最大 80% の物理メモリーがバッファプールに割り当てられます。
- 関連データをさまざまなテーブルに分割すると、強制的に [参照整合性](#) が適用される [外部キー](#) を設定できます。データを更新または削除すると、ほかのテーブル内の関連データも自動的に更新または削除されます。プライマリテーブル内に対応するデータが存在しないセカンダリテーブルにデータを挿入しようとすると、自動的に不正なデータが除外されます。
- ディスク上またはメモリー内のデータが破損した場合は、偽のデータを使用する前に、[チェックサム](#) メカニズムによって警告が発行されます。
- テーブルごとに適切な [主キー](#) カラムを持つデータベースを設計すると、これらのカラムが関与する操作が自動的に最適化されます。WHERE 句、ORDER BY 句、GROUP BY 句、および [結合](#) 操作では、主キーカラムへの参照が非常に高速です。
- 挿入、更新および削除は、[change buffering](#) と呼ばれる自動メカニズムによって最適化されます。InnoDB では、同じテーブルへの並列読み取りおよび書き込みアクセスが許可されているだけでなく、ディスク I/O が効率化されるように変更されたデータがキャッシュに入れられます。
- パフォーマンスの利点は、長時間実行されるクエリーを含む巨大なテーブルだけに限定されません。同じ行が 1 つのテーブルから何度もアクセスされると、[適応型ハッシュインデックス](#) と呼ばれる機能に引き継がれ、ハッシュテーブルから読み取られたかのように、これらの検索がさらに高速になります。
- テーブルおよび関連付けられたインデックスを圧縮できます。
- パフォーマンスおよび可用性への影響を大幅に少なくして、インデックスを作成および削除できます。
- [file-per-table](#) テーブルスペースの切捨ては非常に高速で、InnoDB のみが再利用できる [system tablespace](#) 内の領域を解放するのではなく、オペレーティングシステムが再利用できるようにディスク領域を解放できます。
- テーブルデータの記憶域レイアウトは、[DYNAMIC](#) 行形式の [BLOB](#) および長いテキストフィールドより効率的です。
- [INFORMATION_SCHEMA](#) テーブルでクエリーを実行することで、ストレージエンジンの内部動作をモニターできます。
- [Performance Schema](#) テーブルをクエリーすることによって、ストレージエンジンのパフォーマンスの詳細をモニターできます。
- 同じステートメント内でも、InnoDB のテーブルと別の MySQL ストレージエンジンのテーブルを混在させることができます。たとえば、[結合](#) 操作を使用すると、単一のクエリーで InnoDB テーブルと [MEMORY](#) テーブルのデータを結合できます。

- InnoDB は、大きなデータボリュームを処理する際に、高い CPU の効率性と最大のパフォーマンスが実現されるように設計されています。
- InnoDB テーブルは、ファイルサイズが 2G バイトに制限されているオペレーティングシステム上でも、大量のデータを処理できます。

アプリケーションコードで適用できる InnoDB 固有のチューニング技術については、[セクション 8.5 「InnoDB テーブルの最適化」](#)を参照してください。

15.1.2 InnoDB テーブルのベストプラクティス

このセクションでは、InnoDB テーブルを使用する場合のベストプラクティスについて説明します。

- 最も頻繁にクエリーするカラムを使用してすべてのテーブルに `primary key` を指定するか、明らかな主キーがない場合は `auto-increment` 値を指定します。
- これらのテーブルの同一の ID 値に基づいて複数のテーブルからデータが取得される場合は、`joins` を使用します。結合のパフォーマンスを高速にするには、結合カラム上に外部キーを定義し、各テーブル内でそれらのカラムを同じデータ型で宣言します。外部キーを追加すると、参照カラムが確実にインデックス付けされ、パフォーマンスを向上させることができます。また、外部キーは、影響を受けるすべてのテーブルに削除または更新を伝播し、対応する ID が親テーブルに存在しない場合に子テーブルへのデータの挿入を防止します。
- `autocommit` をオフにします。1 秒間に何百回もコミットすると、パフォーマンスに上限が設定されます (これは、ストレージデバイスの書き込み速度で制限されます)。
- 関連する DML 操作のセットを、`START TRANSACTION` および `COMMIT` ステートメントでカッコで囲んで `transactions` にグループ化します。頻繁にはコミットしたくない一方で、コミットなしで何時間も実行される `INSERT`、`UPDATE`、または `DELETE` ステートメントの巨大なバッチも発生させたくありません。
- `LOCK TABLES` ステートメントを使用しません。InnoDB は、一度に同じテーブルへのすべての読み取りおよび書き込みを行うことで、信頼性や高パフォーマンスを犠牲にせずに、複数のセッションを処理できます。行のセットへの排他的な書き込みアクセス権を取得するには、`SELECT ... FOR UPDATE` という構文を使用して、更新対象の行のみをロックします。
- `innodb_file_per_table` オプションを有効にするか、一般的なテーブルスペースを使用して、テーブルのデータおよびインデックスを `system tablespace` ではなく別々のファイルに配置します。
`innodb_file_per_table` オプションはデフォルトで有効になっています。
- データおよびアクセスパターンが InnoDB のテーブルまたはページの `compression` 機能を利用できるかどうかを評価します。読み取りおよび書き込みの機能を犠牲にせずに、InnoDB テーブルを圧縮できます。
- オプション `--sql_mode=NO_ENGINE_SUBSTITUTION` を指定してサーバーを実行し、`CREATE TABLE` の `ENGINE=` 句で指定されたエンジンに問題がある場合に、別のストレージエンジンでテーブルが作成されないようにします。

15.1.3 InnoDB がデフォルトのストレージエンジンであるかどうかの確認

`SHOW ENGINES` ステートメントを発行して、使用可能な MySQL ストレージエンジンを表示します。InnoDB 行で `DEFAULT` を探します。

```
mysql> SHOW ENGINES;
```

または、`INFORMATION_SCHEMA.ENGINES` テーブルをクエリーします。

```
mysql> SELECT * FROM INFORMATION_SCHEMA.ENGINES;
```

15.1.4 InnoDB を使用したテストおよびベンチマーク

InnoDB がデフォルトのストレージエンジンでない場合は、コマンド行で `--default-storage-engine=InnoDB` を定義してサーバーを再起動するか、MySQL サーバーオプションファイルの `[mysqld]` セクションで `default-storage-engine=innodb` を定義してサーバーを再起動することによって、データベースサーバーまたはアプリケーションが InnoDB で正しく動作するかどうかを判断できます。

デフォルトのストレージエンジンを変更しても、新たに作成されたテーブルしか影響を受けないため、アプリケーションのインストールおよび設定ステップをすべて実行して、すべてが正しくインストールされたことを確認します。次に、すべてのアプリケーション機能を実行して、データのロード、編集、およびクエリー機能がすべて動作することを確認します。テーブルが別のストレージエンジンに固有の機能に依存している場合は、エラーが発生します。エラーを回避するには、`CREATE TABLE` ステートメントに `ENGINE=other_engine_name` 句を追加します。

ストレージエンジンに関する意図的な決定を行わず、InnoDB を使用して作成された特定のテーブルの動作をプレビューする場合は、テーブルごとにコマンド `ALTER TABLE table_name ENGINE=InnoDB;` を発行します。または、元のテーブルを妨げずにテストクエリーおよびその他のステートメントを実行するには、コピーを作成します：

```
CREATE TABLE InnoDB_Table (...) ENGINE=InnoDB AS SELECT * FROM other_engine_table;
```

現実的なワークロードで完全なアプリケーションを使用してパフォーマンスを評価するには、最新の MySQL サーバーをインストールし、ベンチマークを実行します。

完全なアプリケーションのライフサイクル (インストールから頻繁な使用まで)、およびサーバーの再起動をテストします。電源障害のシミュレーションを行うために、データベースの負荷が高いときにサーバープロセスを強制終了し、サーバーの再起動時にデータが正常にリカバリされるかどうかを確認します。

特に、ソースサーバーとレプリカで異なる MySQL バージョンとオプションを使用する場合は、レプリケーション構成をテストします。

15.2 InnoDB および ACID モデル

ACID モデルは、ビジネスデータおよびミッションクリティカルなアプリケーションで重要となる信頼性の側面が強調されたデータベース設計原則のセットです。ソフトウェアのクラッシュやハードウェアの誤動作などの例外的な状況でも、データが破損せず、結果が歪曲されないように、MySQL には、ACID モデルに厳密に準拠した InnoDB ストレージエンジンなどのコンポーネントが含まれています。ACID に準拠した機能に依存していれば、一貫性チェックおよびクラッシュリカバリのメカニズムを再開発する必要がありません。追加のソフトウェアの保護手段、信頼性が最高のハードウェア、または少量のデータ損失や不整合に耐えることができるアプリケーションが備わっている場合は、ACID の信頼性の一部と引き換えに、パフォーマンスやスループットが向上するように MySQL の設定を調整できます。

次のセクションでは、どのように MySQL の機能 (特に InnoDB ストレージエンジン) が ACID モデルのカテゴリとやりとりするのかについて説明します。

- A: 原子性。
- C: 一貫性。
- I: 分離性。
- D: 持続性。

原子性

ACID モデルの原子性の側面には、主に InnoDB の [トランザクション](#) が関与しています。関連する MySQL の機能は次のとおりです。

- 自動コミット設定。
- `COMMIT` ステートメント。
- `ROLLBACK` ステートメント。
- `INFORMATION_SCHEMA` テーブルの運用データ。

一貫性

ACID モデルの一貫性の側面には、主にクラッシュからデータを保護するための内部的な InnoDB 処理が関与しています。関連する MySQL の機能は次のとおりです。

- InnoDB 二重書き込みバッファ。
- InnoDB クラッシュリカバリ。

分離性

ACID モデルの分離性の側面には、主に InnoDB のトランザクション (特に、各トランザクションに適用される分離レベル) が関与しています。関連する MySQL の機能は次のとおりです。

- 自動コミット設定。
- SET ISOLATION LEVEL ステートメント。
- InnoDB ロックの低レベルの詳細。これらの詳細は、パフォーマンスチューニング時に INFORMATION_SCHEMA テーブルから参照します。

持続性

ACID モデルの持続性の側面には、特定のハードウェア構成とやりとりする MySQL ソフトウェアの機能が関与しています。CPU、ネットワーク、およびストレージデバイスの性能に応じて多くの可能性が考えられるため、具体的なガイドラインを示す際は、この側面がもっとも複雑になります。(これらのガイドラインに従うことは、「新しいハードウェア」を購入するという形になる場合があります。) 関連する MySQL の機能は次のとおりです。

- innodb_doublewrite 構成オプションでオンとオフが切り替えられる InnoDB の二重書き込みバッファ。
- innodb_flush_log_at_trx_commit 構成オプション。
- sync_binlog 構成オプション。
- innodb_file_per_table 構成オプション。
- ストレージデバイス内の書き込みバッファ (ディスクドライブ、SSD、RAID アレイなど)。
- ストレージデバイス内のバッテリーでバックアップされるキャッシュ。
- MySQL を実行する際に使用されるオペレーティングシステム (特に、fsync() システムコールでのサポート)。
- MySQL サーバーを実行し、MySQL データを格納するすべてのコンピュータサーバーおよびストレージデバイスへの電力を保護する無停電電源装置 (UPS)。
- バックアップ方針 (頻度、バックアップのタイプ、バックアップの保存期間など)。
- 分散型またはホスト型のデータアプリケーションの場合、MySQL サーバー用のハードウェアが配置されているデータセンター、およびデータセンター間のネットワーク接続の特定の特性。

15.3 InnoDB マルチバージョン

InnoDB はマルチバージョンストレージエンジンです。並列実行やロールバックなどのトランザクション機能をサポートするために、変更された行の古いバージョンに関する情報が保持されます。この情報は、テーブルスペース内にロールバックセグメントと呼ばれるデータ構造で (Oracle では類似したデータ構造のあとに) 格納されます。InnoDB では、トランザクションのロールバックが必要となる取り消し操作を実行するために、ロールバックセグメント内の情報が使用されます。また、この情報は、一貫性読み取りのために行の初期バージョンを構築する際にも使用されます。

InnoDB は内部的に、データベース内に格納された各行に 3 つのフィールドを追加します。6 バイトの DB_TRX_ID フィールドは、行を挿入または更新した最後のトランザクションに対して、トランザクション識別子を指示します。また、行内の特別ビットが削除されたマークするように設定されている場合、削除は内部的に更新として処理されます。各行には、ロールポインタと呼ばれる 7 バイトの DB_ROLL_PTR フィールドも含まれています。ロールポインタは、ロールバックセグメントに書き込まれた Undo ログレコードを示しています。行が更新された場合は、Undo ログレコードに、更新される前の行の内容を再構築するために必要な情報が含まれます。6 バイトの DB_ROW_ID フィールドには、新しい行が挿入されると単調に増加する行 ID が含まれています。InnoDB によって自動生成された

クラスタ化されたインデックスには、行 ID 値が含まれます。それ以外の場合、インデックスに `DB_ROW_ID` カラムが含まれることはありません。

ロールバックセグメント内の Undo ログは、挿入および更新 Undo ログに分割されます。挿入 Undo ログはトランザクションロールバックでのみ必要であるため、トランザクションのコミット直後に破棄できます。更新 Undo ログも一貫性読み取りで使用されますが、InnoDB によってスナップショットが割り当てられたトランザクションが存在しなくなったあとでのみ破棄できます。更新 Undo ログ内のスナップショット情報は、データベース行の以前のバージョンを構築する際に一貫性読み取りが必要となる可能性があります。

トランザクション (一貫性読み取りのみを発行するトランザクションを含む) を定期的にコミットしてください。それ以外の場合、InnoDB は更新 Undo ログからデータを破棄できないため、ロールバックセグメントが大きくなり過ぎてテーブルスペースがいっぱいになる可能性があります。

一般に、ロールバックセグメント内の Undo ログレコードの物理的サイズは、それに対応する挿入された行や更新された行よりも小さいです。この情報を使用すると、ロールバックセグメントで必要となる領域を計算できます。

InnoDB マルチバージョンスキームでは、SQL ステートメントで行を削除しても、その行はすぐにデータベースから物理的に削除されません。InnoDB は、削除用書き込まれた更新 Undo ログレコードが破棄されたときにのみ、対応する行およびそのインデックスレコードを物理的に削除します。このような削除操作は **ページ** と呼ばれ、非常に高速です。通常は、削除が行われなかった SQL ステートメントと同じ時系列順で実行されます。

テーブル内で小さなバッチの行をほぼ同じ速度で挿入および削除すると、すべての「デッド」行が原因で、ページスレッドが遅延し始め、増加し続ける可能性があります。これにより、すべてにおいてディスクが抑制され、非常に低速になります。このような場合は、新たな行操作を抑制し、`innodb_max_purge_lag` システム変数を調整することで、より多くのリソースをページスレッドに割り当てます。詳細は、[セクション 15.14 「InnoDB の起動オプションおよびシステム変数」](#) を参照してください。

マルチバージョンングおよびセカンダリインデックス

InnoDB multiversion concurrency control (MVCC) は、セカンダリインデックスをクラスタ化されたインデックスとは異なる方法で扱います。クラスタ化されたインデックス内のレコードはインプレースで更新され、非表示のシステムカラムは undo ログエントリを指し、このエントリから以前のバージョンのレコードを再構築できます。クラスタ化されたインデックスレコードとは異なり、セカンダリインデックスレコードには非表示のシステムカラムは含まれず、インプレースで更新されません。

セカンダリインデックスカラムが更新されると、古いセカンダリインデックスレコードが削除マークされ、新しいレコードが挿入され、削除マークが付けられたレコードが最終的にページされます。セカンダリインデックスレコードが削除マークされるか、新しいトランザクションによってセカンダリインデックスページが更新されると、InnoDB はクラスタインデックスでデータベースレコードを検索します。クラスタインデックスでは、読み取りトランザクションの開始後にレコードが変更された場合、レコード `DB_TRX_ID` がチェックされ、正しいバージョンのレコードが undo ログから取得されます。

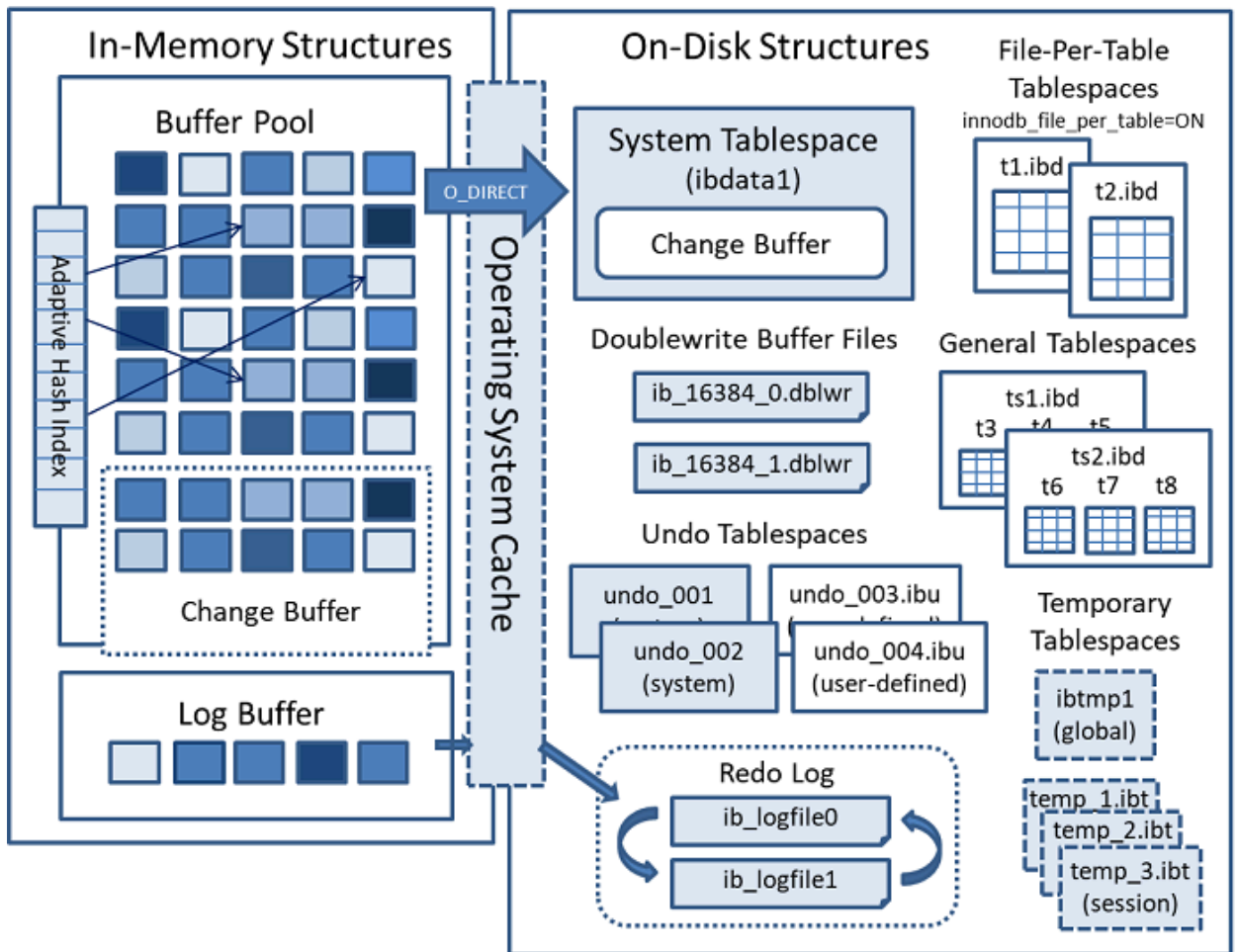
セカンダリインデックスレコードが削除対象としてマークされている場合、またはセカンダリインデックスページが新しいトランザクションによって更新されている場合、**covering index** 手法は使用されません。InnoDB は、インデックス構造から値を戻すかわりに、クラスタ化されたインデックス内のレコードを検索します。

ただし、**index condition pushdown (ICP)** の最適化が有効で、**WHERE** 条件の一部をインデックスのフィールドのみを使用して評価できる場合、MySQL サーバーは **WHERE** 条件のこの部分を、インデックスを使用して評価されるストレージエンジンにプッシュダウンします。一致するレコードが見つからない場合、クラスタインデックスルックアップは回避されます。一致するレコードが見つかった場合、削除マークが付けられたレコードの中でも、InnoDB はクラスタ化されたインデックス内のレコードを検索します。

15.4 InnoDB のアーキテクチャ

次の図は、InnoDB ストレージエンジンアーキテクチャーを構成するインメモリおよびディスク上の構造を示しています。各構造の詳細は、[セクション 15.5 「InnoDB インメモリ構造」](#) および [セクション 15.6 「InnoDB オンディスク構造」](#) を参照してください。

図 15.1 InnoDB のアーキテクチャ



15.5 InnoDB インメモリ構造

このセクションでは、InnoDB のインメモリ構造および関連トピックについて説明します。

15.5.1 バッファプール

バッファプールは、InnoDB がアクセス時にテーブルおよびインデックスデータをキャッシュするメインメモリ内の領域です。バッファプールを使用すると、頻繁に使用されるデータをメモリから直接処理できるため、処理速度が向上します。専用サーバーでは、多くの場合、最大 80% の物理メモリがバッファプールに割り当てられます。

大容量読み取り操作の効率を高めるため、バッファプールは複数行を保持できるページに分割されます。キャッシュ管理を効率化するために、バッファプールはページのリンクリストとして実装されます。使用頻度が低いデータは、LRU アルゴリズムのバリエーションを使用してキャッシュから削除されます。

バッファプールを利用して頻繁にアクセスされるデータをメモリに保持する方法を理解することは、MySQL チューニングの重要な側面です。

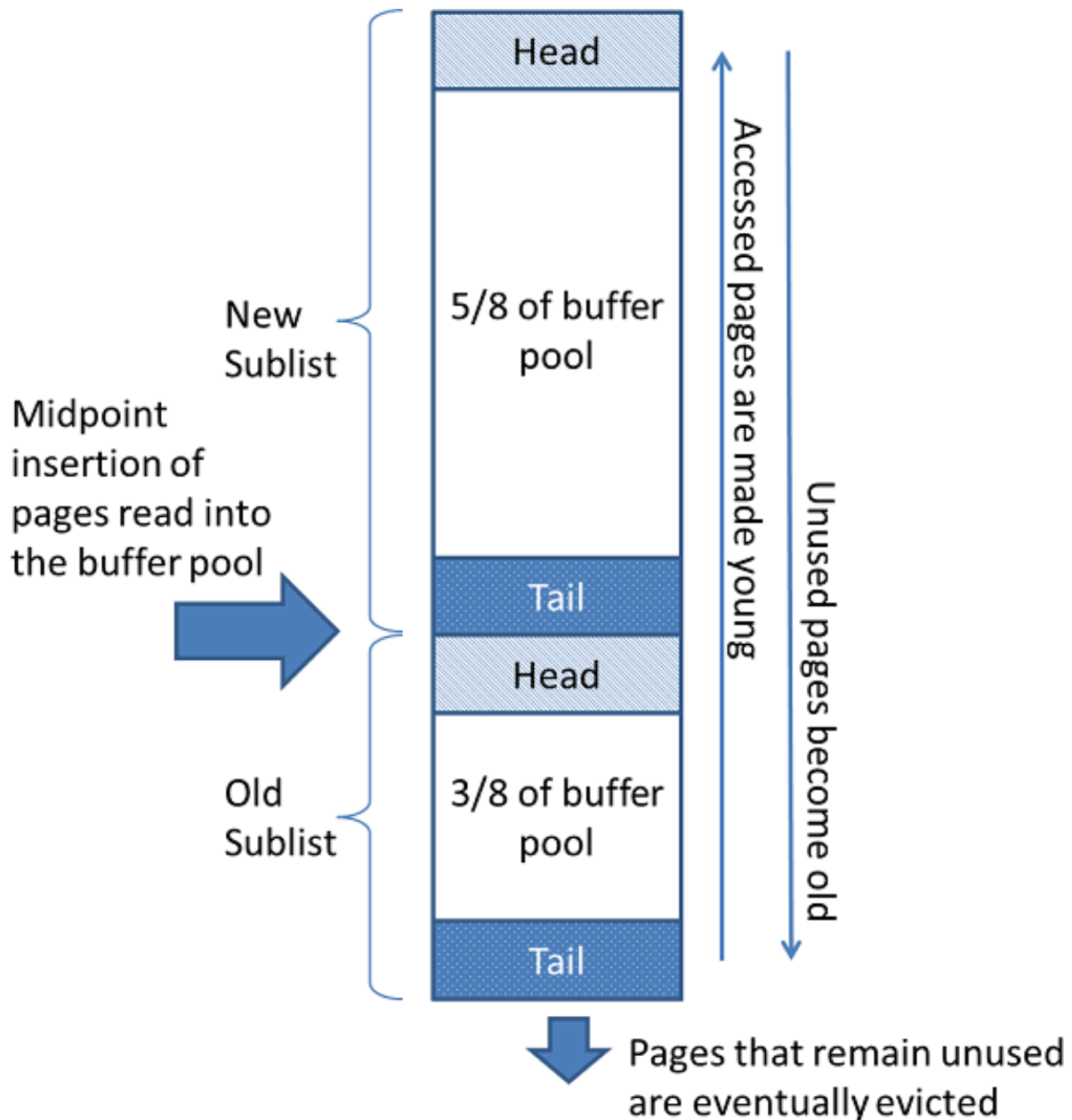
バッファプール LRU アルゴリズム

バッファプールは、最低使用頻度 (LRU) アルゴリズムのバリエーションを使用してリストとして管理されます。バッファプールに新しいページを追加するための領域が必要な場合は、最も最近使用されていないページが削除され、新しいページがリストの中央に追加されます。このミッドポイント挿入戦略はリストを 2 つのサブリストとして扱います。

- 先頭は、最近アクセスされた「新しい」(「若い」) ページのサブリスト

- 末尾は、最近アクセスされていない「古い」ページのサブリスト

図 15.2 バッファプールリスト



このアルゴリズムは、頻繁に使用されるページを新しいサブリストに保持します。古いサブリストには、あまり頻繁に使用されないページが含まれています。これらのページは [eviction](#) の候補です。

デフォルトでは、アルゴリズムは次のように動作します:

- バッファプールの 3/8 が古いサブリストに割り振られます。
- リストのミッドポイントは、新しいサブリストの末尾と古いサブリストの先頭が接する境界です。
- InnoDB は、バッファプールにページを読み取るときに、最初に中間ポイント (古いサブリストの先頭) にページを挿入します。ページの読み込みは、SQL クエリーなどのユーザーが開始する操作、または InnoDB によって自動的に実行される [先読み](#) 操作が要求した場合に発生する可能性があります。

- 古いサブリストのページにアクセスすると、「若い」になり、新しいサブリストの先頭に移動します。ユーザーが開始した操作の要求によるページが読み込まれた場合、最初のアクセスがただちに行われ、ページは若いページになります。先読み操作のためにページが読み込まれた場合、最初のアクセスはすぐに発生せず、ページが削除されるまでまったく行われない可能性もあります。
- データベースが動作するにつれて、アクセスされないバッファプール内のページは、リストの後方に移動し「age」なります。新しいサブリストと古いサブリストの両方のページは、他のページが新しいページになるとエージングされます。古いサブリスト内のページも、中間点にページが挿入されると有効になります。最終的に、未使用のままのページは古いサブリストの末尾に到達し、削除されます。

デフォルトでは、クエリーによって読み取られたページはすぐに新しいサブリストに移動されます。つまり、これらのページはバッファプールに保持されます。たとえば、`mysqldump` または `WHERE` 句のない `SELECT` ステートメントなどにおいて実行されるテーブルスキャンは、新しいデータが再度使用されない場合でも、大量のデータをバッファプールに取り込み、同等の量の古いデータを削除する可能性があります。同様に、先読みバックグラウンドスレッドによってロードされ、一度のみアクセスされるページは、新しいリストの先頭に移動されます。これらの状況では、頻繁に使用されるページが、削除の対象となる古いサブリストに移動される可能性があります。この動作の最適化の詳細は、[セクション15.8.3.3「バッファプールをスキャンに耐えられるようにする」](#) および [セクション15.8.3.4「InnoDB バッファプールのプリフェッチ \(先読み\) の構成」](#) を参照してください。

InnoDB 標準モニターの出力には、バッファプール LRU アルゴリズムの操作に関する `BUFFER POOL AND MEMORY` セクションのいくつかのフィールドが含まれています。詳細は、[InnoDB 標準モニターを使用したバッファプールのモニタリング](#) を参照してください。

バッファプール構成

バッファプールの様々な側面を構成して、パフォーマンスを向上させることができます。

- 理想的には、バッファプールのサイズをできるだけ大きな値に設定して、サーバー上のほかのプロセスが過剰なページングなく実行するように、十分なメモリーを残します。バッファプールが大きいほど、InnoDB はさらにインメモリーデータベースのように動作し、ディスクから1回データを読み取り、後続の読み取り時に、メモリーからデータにアクセスします。[セクション15.8.3.1「InnoDB バッファプールサイズの構成」](#) を参照してください。
- 十分なメモリーがある64ビットシステムでは、バッファプールを複数の部分に分割して、同時操作間のメモリー構造の競合を最小限に抑えることができます。詳細は、[セクション15.8.3.2「複数のバッファプールインスタンスの構成」](#) を参照してください。
- アクセス頻度が低い大量のデータをバッファプールに取り込むような突然のスパイクに関係なく、頻繁にアクセスされるデータをメモリーに保持することができます。詳細は、[セクション15.8.3.3「バッファプールをスキャンに耐えられるようにする」](#) を参照してください。
- 先読みリクエストを実行してバッファプールにページを非同期的にプリフェッチする方法とタイミングを制御して、ページのニーズが低下することを予測できます。詳細は、[セクション15.8.3.4「InnoDB バッファプールのプリフェッチ \(先読み\) の構成」](#) を参照してください。
- バックグラウンドフラッシュが発生するタイミング、およびフラッシュ率をワークロードに基づいて動的に調整するかどうかを制御できます。詳細は、[セクション15.8.3.5「バッファプールのフラッシュの構成」](#) を参照してください。
- サーバーの再起動後の長いウォームアップ期間を回避するために、InnoDB が現在のバッファプールの状態を保持する方法を構成できます。詳細は、[セクション15.8.3.6「バッファプールの状態の保存と復元」](#) を参照してください。

InnoDB 標準モニターを使用したバッファプールのモニタリング

`SHOW ENGINE INNODB STATUS` を使用してアクセスできる InnoDB 標準モニターの出力では、バッファプールの操作に関するメトリックが提供されます。バッファプールメトリックは、InnoDB 標準モニター出力の `BUFFER POOL AND MEMORY` セクションにあり、次のように表示されます:

```
-----  
BUFFER POOL AND MEMORY  
-----  
Total large memory allocated 2198863872  
Dictionary memory allocated 776332  
Buffer pool size 131072  
Free buffers 124908
```

```

Database pages 5720
Old database pages 2071
Modified db pages 910
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 4, not young 0
0.10 youngs/s, 0.00 non-youngs/s
Pages read 197, created 5523, written 5060
0.00 reads/s, 190.89 creates/s, 244.94 writes/s
Buffer pool hit rate 1000 / 1000, young-making rate 0 / 1000 not
0 / 1000
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read
ahead 0.00/s
LRU len: 5720, unzip_LRU len: 0
I/O sum[0]:cur[0], unzip sum[0]:cur[0]

```

次のテーブルでは、**InnoDB** 標準モニターによってレポートされるバッファプールメトリックについて説明します。

注記

InnoDB 標準モニター出力で提供される秒当たりの平均は、**InnoDB** 標準モニター出力が最後に出力されてからの経過時間に基づきます。

表 15.2 InnoDB バッファプールメトリック

名前	説明
割り当てられた合計メモリー	バッファプールに割り当てられた合計メモリー (バイト)。
割り当てられたディクショナリメモリー	InnoDB データディクショナリに割り当てられた合計メモリー (バイト)。
バッファプールサイズ	バッファプールに割り当てられたページ単位の合計サイズ。
空きバッファ	バッファプール空きリストの合計サイズ (ページ数)。
データベースページ	バッファプール LRU リストの合計サイズ (ページ数)。
古いデータベースページ	バッファプールの古い LRU サブリストの合計サイズ (ページ数)。
変更された DB ページ	バッファプールで変更された現在のページ数。
保留検針	バッファプールへの読み取りを待機しているバッファプールページの数。
保留中の書込み LRU	LRU リストの下部から書き込まれるバッファプール内の古いデータページの数。
保留中の書込みフラッシュリスト	チェックポイント中にフラッシュされるバッファプールページの数。
保留中の書込み単一ページ	バッファプール内の保留中の独立したページ書込みの数。
若いページ	バッファプール LRU リストで若くなったページの合計数 (「new」 ページのサブリストの先頭に移動)。
作成されたページが若くない	バッファプール LRU リストで若くなっていないページ (若くない「old」 サブリストに残っているページ) の合計数。
youngs/s	ページを若くしたバッファプール LRU リスト内の古いページへのアクセスの秒当たりの平均。詳細は、このテーブルの後のノートを参照してください。
non-youngs/s	バッファプール LRU リスト内の、ページを若くしなかった古いページへのアクセスの秒当たりの平均。詳細は、このテーブルの後のノートを参照してください。
読み取られたページ	バッファプールから読み取られたページの合計数。

名前	説明
作成されたページ	バッファプール内に作成されたページの合計数。
書き込まれたページ	バッファプールから書き込まれたページの合計数。
reads/s	バッファプールページ読取り/秒の平均数。
creates/s	作成されたバッファプールページの秒当たりの平均数/秒。
writes/s	バッファプールページ書き込みの秒当たりの平均数。
バッファプールヒット率	バッファプールメモリーから読み取られたページとディスク記憶域から読み取られたページのバッファプールページヒット率。
若いマーキング率	ページアクセスによってページが若くなった平均ヒット率。詳細は、このテーブルの後のノートを参照してください。
not (若いマーキング率)	ページアクセスによってページが若くなっていない平均ヒット率。詳細は、このテーブルの後のノートを参照してください。
先読みされたページ	先読み操作の秒当たりの平均。
アクセス権なしで削除されたページ	バッファプールからアクセスせずに削除されたページの秒当たり平均。
ランダム先読み	ランダム先読み操作の秒当たりの平均。
LRU len	バッファプール LRU リストの合計サイズ (ページ数)。
unzip_LRU len	バッファプールの unzip_LRU リストの合計サイズ (ページ数)。
I/O 合計	過去 50 秒間にアクセスされたバッファプール LRU リストページの合計数。
I/O cur	アクセスしたバッファプール LRU リストページの合計数。
I/O unzip sum	アクセスされたバッファプール unzip_LRU リストページの合計数。
I/O unzip cur	アクセスされたバッファプール unzip_LRU リストページの合計数。

メモ:

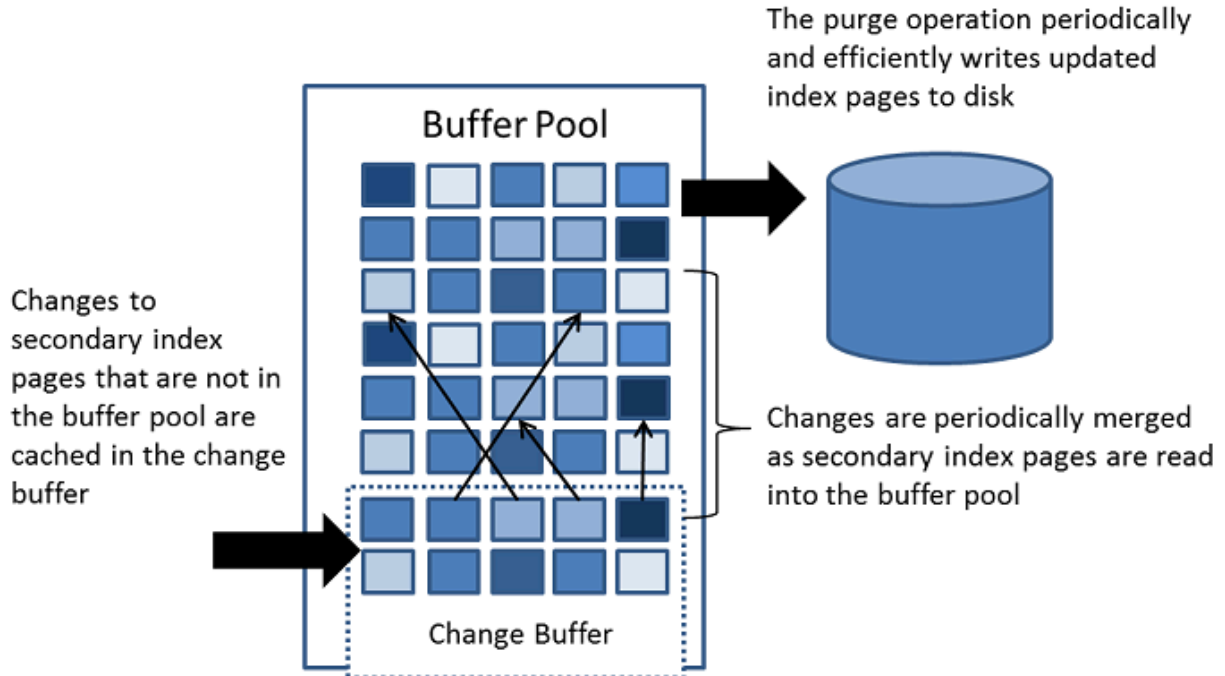
- **young/s** メトリックは、古いページにのみ適用できます。これは、ページ数ではなく、ページへのアクセス数に基づきます。特定のページへの複数のアクセスが可能で、そのすべてがカウントされます。大規模なスキャンが発生していないときに非常に低い **young/s** 値が表示される場合は、遅延時間を短縮するか、古いサブリストに使用されるバッファプールの割合を増やす必要がある場合があります。パーセンテージを増やすと古いサブリストが大きくなるため、そのサブリスト内のページが末尾に移動するのに時間がかかり、これらのページに再度アクセスして若くなる可能性が高くなります。
- **non-young/s** メトリックは、古いページにのみ適用できます。これは、ページ数ではなく、ページへのアクセス数に基づきます。特定のページへの複数のアクセスが可能で、そのすべてがカウントされます。大規模なテーブルスキャンの実行時に **non-young/s** 値が高くない (および **young/s** 値が大きい) 場合は、遅延値を増やします。
- **young-making** レートは、古いサブリスト内のページへのアクセスだけでなく、すべてのバッファプールページへのアクセスを考慮します。 **young-making** レートおよび **not** レートは、通常、全体的なバッファプールヒット率に加算されません。古いサブリストのページヒットによってページが新しいサブリストに移動しますが、新しいサブリストのページヒットによってリストの先頭に移動されるのは、先頭から一定の距離がある場合のみです。
- **not (young-making rate)** は、 **innodb_old_blocks_time** で定義された遅延が満たされていないか、ページがヘッドに移動されなかった新しいサブリストのページヒットが原因で、ページアクセスの結果ページが若くなっていない平均ヒット率です。このレートは、古いサブリスト内のページへのアクセスだけでなく、すべてのバッファプールページへのアクセスを考慮します。

バッファプール `server status variables` と `INNODB_BUFFER_POOL_STATS` テーブルには、InnoDB Standard Monitor の出力と同じバッファプールメトリックが多数用意されています。詳細は、例 15.10 「`INNODB_BUFFER_POOL_STATS` テーブルのクエリー」を参照してください。

15.5.2 変更バッファ

変更バッファは、`secondary index` ページが `buffer pool` がない場合に、そのページに対する変更をキャッシュする特別なデータ構造です。バッファされた変更は、`INSERT`、`UPDATE` または `DELETE` 操作 (DML) によって発生する可能性があり、後で他の読取り操作によってページがバッファプールにロードされるときにマージされます。

図 15.3 変更バッファ



`clustered indexes` とは異なり、セカンダリインデックスは通常一意ではなく、セカンダリインデックスへの挿入は比較的ランダムな順序で行われます。同様に、削除および更新は、インデックスツリーに隣接して配置されていないセカンダリインデックスページに影響する可能性があります。キャッシュされた変更を後でマージすると、影響を受けるページが他の操作によってバッファプールに読み込まれるときに、セカンダリインデックスページをディスクからバッファプールに読み込むために必要な大量のランダムアクセス I/O が回避されます。

システムがほとんどアイドル状態のとき、または低速シャットダウン中に実行されるページ操作は、定期的に更新されたインデックスページをディスクに書き込みます。ページ操作では、各値がすぐにディスクに書き込まれた場合よりも効率的に、一連のインデックス値のディスクブロックを書き込むことができます。

変更バッファのマージでは、影響を受ける行と更新するセカンダリインデックスが多数ある場合、数時間かかることがあります。この期間中、ディスク I/O が増加し、ディスクバインドされたクエリーの速度が大幅に低下する可能性があります。変更バッファのマージは、トランザクションがコミットされた後、およびサーバーが停止して再起動された後でも継続して発生する場合があります (詳細は、[セクション 15.21.2 「InnoDB のリカバリの強制的な実行」](#)を参照してください)。

メモリー内では、変更バッファはバッファプールの一部を占有します。ディスクでは、変更バッファはシステムテーブルスペースの一部であり、データベースサーバーの停止時にインデックス変更がバッファされます。

変更バッファにキャッシュされるデータのタイプは、`innodb_change_buffering` 変数によって制御されます。詳細は、[変更バッファリングの構成](#)を参照してください。最大変更バッファサイズを構成することもできます。詳細は、[変更バッファの最大サイズの構成](#)を参照してください。

インデックスに降順のインデックスカラムが含まれている場合、または主キーに降順のインデックスカラムが含まれている場合、セカンダリインデックスでは変更バッファリングはサポートされません。

変更バッファに関するよくある質問への回答は、[セクションA.16「MySQL 8.0 FAQ : InnoDB 変更バッファ」](#)を参照してください。

変更バッファリングの構成

`INSERT`、`UPDATE` および `DELETE` 操作がテーブルに対して実行される場合、インデックス付けされたカラムの値 (特にセカンダリキーの値) はソートされていない順序であることがよくあり、セカンダリインデックスを最新の状態にするにはかなりの I/O が必要です。関連する `page` が `buffer pool` に存在しない場合、`change buffer` はセカンダリインデックスエントリへの変更をキャッシュするため、ディスクからすぐにページを読み取ることなく、高コストの I/O 操作を回避できます。バッファされた変更は、ページがバッファプールにロードされ、更新されたページが後でディスクにフラッシュされるときにマージされます。InnoDB のメインスレッドは、それらのバッファリングされた変更を、サーバーがほぼアイドル状態にあるときと `低速シャットダウン` 中にマージします。

ディスクの読取りおよび書き込みが少なくなる可能性があるため、変更バッファ機能は、バルク挿入などの大量の DML 操作を使用するアプリケーションなど、I/O-bound、であるワークロードに最も役立ちます。

ただし、変更バッファはバッファプールの一部を占有するため、データページのキャッシュに使用できるメモリーが削減されます。ワーキングセットがバッファプールにほぼ収まる場合、またはテーブルのセカンダリインデックスが比較的少ない場合は、変更バッファリングを無効にすると便利です。作業データセットがバッファプール内に完全に収まる場合、変更バッファリングはバッファプールにないページにのみ適用されるため、余分なオーバーヘッドは発生しません。

`innodb_change_buffering` 構成パラメータを使用して、InnoDB が変更バッファリングを実行する範囲を制御できます。挿入、削除操作 (インデックスレコードが最初に削除対象としてマークされている場合) およびパージ操作 (インデックスレコードが物理的に削除されている場合) のバッファリングを有効または無効にできます。更新操作は、挿入と削除の組合せです。デフォルトの `innodb_change_buffering` 値は `all` です。

許可される `innodb_change_buffering` 値は次のとおりです:

- `all`
デフォルト値: バッファの挿入、削除のマーキング操作、およびパージ。
- `none`
どの操作もバッファリングしません。
- `inserts`
挿入操作をバッファリングします。
- `deletes`
削除のマーキング操作をバッファリングします。
- `changes`
挿入操作と削除マーキング操作の両方をバッファリングします。
- `purges`
バックグラウンドで実行される物理的な削除操作をバッファリングします。

`innodb_change_buffering` パラメータは、MySQL オプションファイル (`my.cnf` または `my.ini`) で設定するか、`SET GLOBAL` ステートメントを使用して動的に変更できます。これには、グローバルシステム変数の設定に十分な権限が必要です。[セクション5.1.9.1「システム変数権限」](#)を参照してください。設定を変更しても、新しい操作のバッファリングに影響します。既存のバッファエントリのマージは影響を受けません。

変更バッファの最大サイズの構成

`innodb_change_buffer_max_size` 変数を使用すると、変更バッファの最大サイズをバッファプールの合計サイズに対する割合として構成できます。デフォルトでは、`innodb_change_buffer_max_size` は 25 に設定されます。最大設定は 50 です。

大量の挿入、更新および削除アクティビティがある MySQL サーバーで `innodb_change_buffer_max_size` を増やすことを検討してください。この場合、変更バッファのマージは新しい変更バッファエントリに対応しないため、変更バッファは最大サイズ制限に達します。

レポートに使用される静的データを使用して MySQL サーバー上の `innodb_change_buffer_max_size` を減らすことを検討してください。または、変更バッファがバッファプールと共有されるメモリー領域を大量に消費し、ページが必要以上に早くバッファプールからエージアウトされるようにすることを検討してください。

代表的なワークロードを使用して様々な設定をテストし、最適な構成を決定します。
`innodb_change_buffer_max_size` 設定は動的で、サーバーを再起動せずに設定を変更できます。

変更バッファの監視

変更バッファの監視には、次のオプションを使用できます:

- `InnoDB` 標準モニターの出力には、変更バッファのステータス情報が含まれます。モニターデータを表示するには、`SHOW ENGINE INNODB STATUS` ステートメントを発行します。

```
mysql> SHOW ENGINE INNODB STATUS\G
```

バッファステータスの変更情報は、`INSERT BUFFER AND ADAPTIVE HASH INDEX` ヘッダーの下にあり、次のように表示されます:

```
-----
INSERT BUFFER AND ADAPTIVE HASH INDEX
-----
lbuf: size 1, free list len 0, seg size 2, 0 merges
merged operations:
  insert 0, delete mark 0, delete 0
discarded operations:
  insert 0, delete mark 0, delete 0
Hash table size 4425293, used cells 32, node heap has 1 buffer(s)
13577.57 hash searches/s, 202.47 non-hash searches/s
```

詳細は、[セクション15.17.3「InnoDB 標準モニターおよびロックモニターの出力」](#)を参照してください。

- `INFORMATION_SCHEMA.INNODB_METRICS` テーブルには、`InnoDB` 標準モニターの出力にあるデータポイントのほとんどと、その他のデータポイントが表示されます。変更バッファメトリックおよびそれぞれの説明を表示するには、次のクエリーを発行します:

```
mysql> SELECT NAME, COMMENT FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME LIKE '%ibuf%'\G
```

`INNODB_METRICS` テーブルの使用の詳細は、[セクション15.15.6「InnoDB INFORMATION_SCHEMA メトリックテーブル」](#)を参照してください。

- `INFORMATION_SCHEMA.INNODB_BUFFER_PAGE` テーブルには、バッファインデックスの変更やバッファビットマップの変更など、バッファプール内の各ページに関するメタデータが表示されます。変更バッファページは、`PAGE_TYPE` によって識別されます。`IBUF_INDEX` は変更バッファインデックスページのページタイプで、`IBUF_BITMAP` は変更バッファビットマップページのページタイプです。

警告

`INNODB_BUFFER_PAGE` テーブルをクエリーすると、大幅なパフォーマンスのオーバーヘッドが生じる可能性があります。パフォーマンスへの影響を回避するために、調査しようとしている問題をテストインスタンスで再現し、テストインスタンスでクエリーを実行してください。

たとえば、`INNODB_BUFFER_PAGE` テーブルをクエリーして、合計バッファプールページ数に対する `IBUF_INDEX` および `IBUF_BITMAP` ページの概数を確認できます。

```
mysql> SELECT (SELECT COUNT(*) FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
  WHERE PAGE_TYPE LIKE 'IBUF%') AS change_buffer_pages,
  (SELECT COUNT(*) FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE) AS total_pages,
  (SELECT ((change_buffer_pages/total_pages)*100))
  AS change_buffer_page_percentage;
+-----+-----+-----+
| change_buffer_pages | total_pages | change_buffer_page_percentage |
```

25	8192	0.3052
----	------	--------

INNODB_BUFFER_PAGE テーブルで提供されるその他のデータの詳細は、[セクション](#)

26.51.1「[INFORMATION_SCHEMA INNODB_BUFFER_PAGE テーブル](#)」を参照してください。関連する使用方法については、[セクション15.15.5「InnoDB INFORMATION_SCHEMA バッファプールテーブル](#)」を参照してください。

- [Performance Schema](#) には、高度なパフォーマンス監視のための変更バッファ相互排他ロック待機インストゥルメンテーションが用意されています。変更バッファインストゥルメンテーションを表示するには、次のクエリーを発行します:

```
mysql> SELECT * FROM performance_schema.setup_instruments
      WHERE NAME LIKE '%wait/synch/mutex/innodb/ibuf%';
+-----+-----+-----+
| NAME                                     | ENABLED | TIMED |
+-----+-----+-----+
| wait/synch/mutex/innodb/ibuf_bitmap_mutex | YES     | YES   |
| wait/synch/mutex/innodb/ibuf_mutex       | YES     | YES   |
| wait/synch/mutex/innodb/ibuf_pessimistic_insert_mutex | YES     | YES   |
+-----+-----+-----+
```

InnoDB mutex 待機の監視の詳細は、[セクション15.16.2「パフォーマンススキーマを使用した InnoDB Mutex 待機のモニタリング](#)」を参照してください。

15.5.3 適応型ハッシュインデックス

適応型ハッシュインデックス機能を使用すると、InnoDB は、トランザクション機能や信頼性を犠牲にすることなく、ワークロードとバッファプールに十分なメモリーを適切に組み合わせたシステム上で、インメモリーデータベースのように実行できます。適応ハッシュインデックス機能は、[innodb_adaptive_hash_index](#) オプションによって有効化され、または [--skip-innodb-adaptive-hash-index](#) オプションにより無効化されます。

検出された検索パターンに基づいて、インデックスキーの接頭辞を使用してハッシュインデックスが作成されます。接頭辞は任意の長さででき、B ツリーの一部の値のみがハッシュインデックスに表示される場合があります。ハッシュインデックスは、頻繁にアクセスされるインデックスのページに対してオンデマンドで作成されます。

テーブルがメインメモリー内にほぼ完全に収容されている場合は、任意の要素の直接検索を有効にし、インデックス値をポインタの一種に変換すると、ハッシュインデックスを使用してクエリーを高速にすることができます。InnoDB には、インデックスの検索をモニターするメカニズムが備わっています。ハッシュインデックスの構築がクエリーにとって有益であると InnoDB が判断した場合は、自動的にそのインデックスが構築されます。

一部のワークロードでは、ハッシュインデックスの検索による高速化の方が、インデックスの検索をモニターしたり、ハッシュインデックスの構造を保持したりする追加の作業よりも重要です。適応型ハッシュインデックスへのアクセスは、複数の同時結合など、負荷の高いワークロードで競合の原因になる場合があります。LIKE 演算子および % ワイルドカードを使用したクエリーもメリットが得られない傾向があります。適応型ハッシュインデックス機能のメリットが得られないワークロードの場合、これをオフにすると不要なパフォーマンスオーバーヘッドが軽減されます。適応型ハッシュインデックス機能が特定のシステムおよびワークロードに適しているかどうかを事前に予測することは困難であるため、有効化および無効化してベンチマークを実行することを検討してください。MySQL 5.6 のアーキテクチャの変更により、適応型ハッシュインデックス機能を以前のリリースよりも無効にする方が適しています。

適応ハッシュインデックス機能はパーティション化されています。各インデックスは特定のパーティションにバインドされ、各パーティションは個別のラッチによって保護されます。パーティション化は、[innodb_adaptive_hash_index_parts](#) 変数によって制御されます。[innodb_adaptive_hash_index_parts](#) 変数はデフォルトで 8 に設定されています。最大設定は 512 です。

[SHOW ENGINE INNODB STATUS](#) 出力の [SEMAPHORES](#) セクションで、適応型ハッシュインデックスの使用および競合を監視できます。[btr0sea.c](#) で作成された RW バッチを待機しているスレッドが多数ある場合は、適応ハッシュインデックスパーティションの数を増やすか、適応ハッシュインデックス機能を無効にすることを検討してください。

ハッシュインデックスのパフォーマンス特性の詳細は、[セクション8.3.9「B ツリーインデックスとハッシュインデックスの比較](#)」を参照してください。

15.5.4 ログバッファ

ログバッファは、ディスク上のログファイルに書き込まれるデータを保持するメモリー領域です。ログバッファサイズは、`innodb_log_buffer_size` 変数によって定義されます。デフォルトのサイズは 16M バイトです。ログバッファの内容は定期的にディスクにフラッシュされます。大きいログバッファを使用すると、トランザクションがコミットされる前に redo ログデータをディスクに書き込むことなく、大きなトランザクションを実行できます。したがって、多数の行を更新、挿入または削除するトランザクションがある場合は、ログバッファのサイズを大きくするとディスク I/O が節約されます。

`innodb_flush_log_at_trx_commit` 変数は、ログバッファの内容をディスクに書き込む方法およびフラッシュする方法を制御します。`innodb_flush_log_at_timeout` 変数は、ログのフラッシュ頻度を制御します。

関連情報については、[メモリー構成](#)、および[セクション 8.5.4 「InnoDB redo ログの最適化」](#)を参照してください。

15.6 InnoDB オンディスク構造

このセクションでは、InnoDB のディスク上の構造および関連項目について説明します。

15.6.1 テーブル

このセクションでは、InnoDB テーブルに関連するトピックについて説明します。

15.6.1.1 InnoDB テーブルの作成

InnoDB テーブルを作成するには、`CREATE TABLE` ステートメントを使用します。

```
CREATE TABLE t1 (a INT, b CHAR (20), PRIMARY KEY (a)) ENGINE=InnoDB;
```

InnoDB がデフォルトのストレージエンジン (デフォルト) として定義されている場合は、`ENGINE=InnoDB` 句を指定する必要はありません。デフォルトのストレージエンジンを確認するには、次のステートメントを発行します:

```
mysql> SELECT @@default_storage_engine;
+-----+
| @@default_storage_engine |
+-----+
| InnoDB                    |
+-----+
```

`mysqldump` またはレプリケーションを使用して、デフォルトのストレージエンジンが InnoDB ではないサーバー上で `CREATE TABLE` ステートメントをリプレイする場合は、引き続き `ENGINE=InnoDB` 句を使用できます。

InnoDB テーブルとそのインデックスは、`system tablespace`、`file-per-table` テーブルスペースまたは `general tablespace` で作成できます。`innodb_file_per_table` が有効な場合 (デフォルト)、InnoDB テーブルは個々の `file-per-table` テーブルスペースに暗黙的に作成されます。逆に、`innodb_file_per_table` を無効にすると、InnoDB システム テーブルスペースに InnoDB テーブルが暗黙的に作成されます。一般的なテーブルスペースにテーブルを作成するには、`CREATE TABLE ... TABLESPACE` 構文を使用します。詳細は、[セクション 15.6.3.3 「一般テーブルスペース」](#)を参照してください。

`file-per-table` テーブルスペースにテーブルを作成すると、MySQL はデフォルトで、MySQL データディレクトリの下にデータベースディレクトリに `.ibd` テーブルスペースファイルを作成します。InnoDB システムテーブルスペースに作成されたテーブルは、MySQL データディレクトリに存在する既存の `ibdata file` に作成されます。一般テーブルスペースで作成されたテーブルは、既存の一般テーブルスペース `.ibd file` に作成されます。一般的なテーブルスペースファイルは、MySQL データディレクトリの内外に作成できます。詳細は、[セクション 15.6.3.3 「一般テーブルスペース」](#)を参照してください。

InnoDB では、内部的に各テーブルのエントリがデータディクショナリに追加されます。このエントリには、データベース名が含まれます。たとえば、テーブル `t1` が `test` データベースに作成されている場合、データベース名のデータディクショナリエントリは `'test/t1'` です。これは、同じ名前 (`t1`) のテーブルを別のデータベースに作成でき、テーブル名が InnoDB 内で衝突しないことを意味します。

InnoDB のテーブルと行の形式

InnoDB テーブルのデフォルトの行形式は、[DYNAMIC](#) のデフォルト値を持つ `innodb_default_row_format` 構成オプションによって定義されます。[Dynamic](#) および [Compressed](#) の行形式を使用すると、テーブル圧縮や長いカラム値の効率的なオフページストレージなどの InnoDB 機能を利用できます。これらの行フォーマットを使用するには、`innodb_file_per_table` を有効 (デフォルト) にする必要があります。

```
SET GLOBAL innodb_file_per_table=1;
CREATE TABLE t3 (a INT, b CHAR (20), PRIMARY KEY (a)) ROW_FORMAT=DYNAMIC;
CREATE TABLE t4 (a INT, b CHAR (20), PRIMARY KEY (a)) ROW_FORMAT=COMPRESSED;
```

または、`CREATE TABLE ... TABLESPACE` 構文を使用して、一般的なテーブルスペースに InnoDB テーブルを作成することもできます。一般テーブルスペースでは、すべての行形式がサポートされます。詳細は、[セクション 15.6.3.3 「一般テーブルスペース」](#) を参照してください。

```
CREATE TABLE t1 (c1 INT PRIMARY KEY) TABLESPACE ts1 ROW_FORMAT=DYNAMIC;
```

`CREATE TABLE ... TABLESPACE` 構文を使用して、[Compact](#) または [Redundant](#) 行形式のテーブルとともに、[Dynamic](#) 行形式の InnoDB テーブルをシステムテーブルスペースに作成することもできます。

```
CREATE TABLE t1 (c1 INT PRIMARY KEY) TABLESPACE = innodb_system ROW_FORMAT=DYNAMIC;
```

InnoDB の行フォーマットの詳細は、[セクション 15.10 「InnoDB の行フォーマット」](#) を参照してください。InnoDB テーブルの行フォーマットおよび InnoDB の行フォーマットの物理特性を決定する方法は、[セクション 15.10 「InnoDB の行フォーマット」](#) を参照してください。

InnoDB テーブルおよび主キー

InnoDB テーブルに対して常に [primary key](#) を定義し、次のようなカラムを指定します:

- もっとも重要なクエリーで参照される。
- ブランクのままになっていない。
- 重複する値がない。
- 挿入後に値が変更されるとしても、きわめてまれである。

たとえば、人に関する情報を含むテーブルでは、複数の人が同じ名前を持つ可能性もあり、名字をブランクにしたリ、名前を変更したりする人もいるため、[\(名、姓\)](#) 上には主キーを作成しません。制約が非常に多く、主キーとして使用する明確なカラムセットがないことも多い場合には、主キーの全部または一部として機能する数値 ID の新しいカラムを作成してください。行が挿入されると自動的に昇順の値が入力されるように、[自動インクリメントカラム](#)を宣言できます。

```
# The value of ID can act like a pointer between related items in different tables.
CREATE TABLE t5 (id INT AUTO_INCREMENT, b CHAR (20), PRIMARY KEY (id));
```

```
# The primary key can consist of more than one column. Any autoinc column must come first.
CREATE TABLE t6 (id INT AUTO_INCREMENT, a INT, b CHAR (20), PRIMARY KEY (id,a));
```

主キーを定義しなくてもテーブルは正しく機能しますが、主キーはパフォーマンスの多くの側面に関係し、大規模または頻繁に使用されるテーブルにとって重要な設計面です。`CREATE TABLE` ステートメントでは、常に主キーを指定することをお勧めします。テーブルを作成し、データをロードしてから、後で `ALTER TABLE` を実行して主キーを追加すると、テーブルの作成時に主キーを定義するよりも操作速度が大幅に遅くなります。

InnoDB テーブルのプロパティの表示

InnoDB テーブルのプロパティを表示するには、`SHOW TABLE STATUS` ステートメントを発行します:

```
mysql> SHOW TABLE STATUS FROM test LIKE '%'\G;
***** 1. row *****
      Name: t1
      Engine: InnoDB
      Version: 10
      Row_format: Compact
      Rows: 0
      Avg_row_length: 0
      Data_length: 16384
      Max_data_length: 0
```



```

Index_length: 0
Data_free: 0
Auto_increment: NULL
Create_time: 2015-03-16 15:13:31
Update_time: NULL
Check_time: NULL
Collation: utf8mb4_0900_ai_ci
Checksum: NULL
Create_options:
Comment:

```

SHOW TABLE STATUS 出力の詳細は、[セクション13.7.7.38「SHOW TABLE STATUS ステートメント」](#) を参照してください。

InnoDB テーブルのプロパティは、**InnoDB Information Schema** システムテーブルを使用してクエリーすることもできます:

```

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TABLES WHERE NAME='test/t1' \G
***** 1. row *****
TABLE_ID: 45
NAME: test/t1
FLAG: 1
N_COLS: 5
SPACE: 35
ROW_FORMAT: Compact
ZIP_PAGE_SIZE: 0
SPACE_TYPE: Single

```

詳細は、[セクション15.15.3「InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル」](#) を参照してください。

15.6.1.2 外部でのテーブルの作成

InnoDB テーブルを外部で作成する (つまり、データディレクトリ外でテーブルを作成する) 理由は様々です。このような理由には、領域管理、I/O の最適化、特定のパフォーマンスまたは容量特性を持つストレージデバイスへのテーブルの配置などがあります。

InnoDB では、テーブルを外部で作成するために次の方法がサポートされています:

- [DATA DIRECTORY 句の使用](#)
- [CREATE TABLE ... TABLESPACE 構文の使用](#)
- [外部一般テーブルスペースでのテーブルの作成](#)

DATA DIRECTORY 句の使用

外部ディレクトリに **InnoDB** テーブルを作成するには、**CREATE TABLE** ステートメントで **DATA DIRECTORY** 句を指定します。

```
CREATE TABLE t1 (c1 INT PRIMARY KEY) DATA DIRECTORY = '/external/directory';
```

file-per-table テーブルスペースに作成されたテーブルでは、**DATA DIRECTORY** 句がサポートされています。**innodb_file_per_table** 変数が有効になっている場合 (デフォルト)、テーブルは file-per-table テーブルスペースに自動的に作成されます。

```

mysql> SELECT @@innodb_file_per_table;
+-----+
| @@innodb_file_per_table |
+-----+
| 1 |
+-----+

```

file-per-table テーブルスペースの詳細は、[セクション15.6.3.2「File-Per-Table テーブルスペース」](#) を参照してください。

CREATE TABLE ステートメントで **DATA DIRECTORY** 句を指定すると、指定したディレクトリの下スキーマディレクトリにテーブルデータファイル (**table_name.ibd**) が作成されます。

MySQL 8.0.21 では、`DATA DIRECTORY` 句を使用してデータディレクトリの外部で作成されるテーブルおよびテーブルパーティションは、InnoDB で認識されるディレクトリに制限されます。この要件により、データベース管理者はテーブルスペースデータファイルが作成される場所を制御し、リカバリ中にデータファイルを検出できるようになります ([クラッシュリカバリ中のテーブルスペースの検出](#) を参照)。既知のディレクトリは、`datadir`、`innodb_data_home_dir` および `innodb_directories` 変数で定義されているディレクトリです。次のステートメントを使用して、これらの設定を確認できます:

```
mysql> SELECT @@datadir,@@innodb_data_home_dir,@@innodb_directories;
```

使用するディレクトリが不明な場合は、テーブルを作成する前に `innodb_directories` 設定に追加します。`innodb_directories` 変数は読み取り専用です。構成するには、サーバーを再起動する必要があります。システム変数の設定に関する一般情報は、[セクション5.1.9「システム変数の使用」](#) を参照してください。

次の例では、`DATA DIRECTORY` 句を使用して外部ディレクトリにテーブルを作成する方法を示します。`innodb_file_per_table` 変数が有効で、ディレクトリが InnoDB に認識されていることを前提としています。

```
mysql> USE test;
Database changed

mysql> CREATE TABLE t1 (c1 INT PRIMARY KEY) DATA DIRECTORY = '/external/directory';

# MySQL creates the table's data file in a schema directory
# under the external directory

shell> cd /external/directory/test
shell> ls
t1.ibd
```

使用上の注意:

- MySQL では、最初にテーブルスペースデータファイルが開いた状態で保持されるため、デバイスをデスマウントできませんが、サーバーがビジー状態の場合は最終的にファイルが閉じられる可能性があります。MySQL の実行中に外部デバイスを誤ってデスマウントしたり、デバイスの切断中に MySQL を起動したりしないように注意してください。関連付けられたデータファイルがないときにテーブルにアクセスしようとすると、重大なエラーが発生し、サーバーの再起動が必要になります。

必要なパスにデータファイルが見つからない場合、サーバーの再起動が失敗することがあります。この場合、テーブルスペースデータファイルをバックアップからリストアするか、テーブルを削除して `data dictionary` から削除できます。

- NFS マウントされたボリュームにテーブルを配置する前に、[MySQL での NFS の使用](#) で概説されている潜在的な問題を確認します。
- LVM スナップショット、ファイルコピーまたはその他のファイルベースのメカニズムを使用してテーブルデータファイルをバックアップする場合は、バックアップが発生する前に、必ず `FLUSH TABLES ... FOR EXPORT` ステートメントを使用して、メモリーにバッファリングされるすべての変更が `flushed` からディスクになるようにします。
- `DATA DIRECTORY` 句を使用して外部ディレクトリにテーブルを作成する方法は、InnoDB でサポートされていない `symbolic links` を使用するかわりに使用する方法です。
- `DATA DIRECTORY` 句は、ソースとレプリカが同じホストに存在するレプリケーション環境ではサポートされていません。`DATA DIRECTORY` 句には完全なディレクトリパスが必要です。この場合、パスをレプリケートすると、ソースとレプリカは同じ場所にテーブルを作成します。
- MySQL 8.0.21 の時点では、file-per-table テーブルスペースで作成されたテーブルは、InnoDB で直接認識されていないかぎり、undo テーブルスペースディレクトリ (`innodb_undo_directory`) に作成できなくなりました。既知のディレクトリは、`datadir`、`innodb_data_home_dir` および `innodb_directories` 変数で定義されているディレクトリです。

CREATE TABLE ... TABLESPACE 構文の使用

`CREATE TABLE ... TABLESPACE` 構文を `DATA DIRECTORY` 句と組み合わせて使用すると、外部ディレクトリにテーブルを作成できます。これを行うには、テーブルスペース名として `innodb_file_per_table` を指定します。

```
mysql> CREATE TABLE t2 (c1 INT PRIMARY KEY) TABLESPACE = innodb_file_per_table  
DATA DIRECTORY = '/external/directory';
```

この方法は、file-per-table テーブルスペースで作成されたテーブルでのみサポートされますが、`innodb_file_per_table` 変数を有効にする必要はありません。他のすべての点では、このメソッドは前述の `CREATE TABLE ... DATA DIRECTORY` メソッドと同等です。同じ使用上のノートが適用されます。

外部一般テーブルスペースでのテーブルの作成

外部ディレクトリに存在する一般的なテーブルスペースにテーブルを作成できます。

- 外部ディレクトリでの一般的なテーブルスペースの作成の詳細は、[一般的なテーブルスペースの作成](#) を参照してください。
- 一般的なテーブルスペースでのテーブルの作成の詳細は、[一般テーブルスペースへのテーブルの追加](#) を参照してください。

15.6.1.3 InnoDB テーブルのインポート

このセクションでは、file-per-table テーブルスペースに存在するテーブル、パーティションテーブルまたは個々のテーブルパーティションのインポートを許可するトランスポータブルテーブルスペース機能を使用してテーブルをインポートする方法について説明します。テーブルをインポートする理由は多数あります：

- 本番以外の MySQL サーバーインスタンスでレポートを実行して、本番サーバーに余分な負荷をかけないようにします。
- 新しいレプリカサーバーにデータをコピーします。
- バックアップされたテーブルスペースファイルからテーブルをリストアします。
- ダンプファイルをインポートするよりも高速にデータを移動できるため、データを再挿入してインデックスを再構築する必要があります。
- ストレージ要件に適したストレージメディアを備えたサーバーにデータを移動する場合。たとえば、ビジー状態のテーブルを SSD デバイスに移動したり、大容量のテーブルを大容量 HDD デバイスに移動したりできます。

トランスポータブルテーブルスペース機能については、このセクションの次のトピックで説明します：

- [前提条件](#)
- [テーブルのインポート](#)
- [パーティションテーブルのインポート](#)
- [テーブルパーティションのインポート](#)
- [制限](#)
- [使用上の注意](#)
- [内部情報](#)

前提条件

- `innodb_file_per_table` 変数は、デフォルトで有効になっている必要があります。
- テーブルスペースのページサイズは、宛先 MySQL サーバーインスタンスのページサイズと一致する必要があります。InnoDB のページサイズは、MySQL サーバーインスタンスの初期化時に構成される `innodb_page_size` 変数によって定義されます。
- テーブルが外部キー関係にある場合は、`DISCARD TABLESPACE` を実行する前に `foreign_key_checks` を無効にする必要があります。また、`ALTER TABLE ... IMPORT TABLESPACE` ではインポートされたデータに外部キー制約が強制されないため、すべての外部キー関連テーブルを同じ論理的な時点でエクスポートする必要があります。これを行うには、関連するテーブルの更新を停止し、すべてのトランザクションをコミットし、テーブルの共有ロックを取得して、エクスポート操作を実行します。

- 別の MySQL サーバーインスタンスからテーブルをインポートする場合、両方の MySQL サーバーインスタンスのステータスは GA (General Availability) であり、同じバージョンである必要があります。それ以外の場合は、インポート先と同じ MySQL サーバーインスタンスにテーブルを作成する必要があります。
- `CREATE TABLE` ステートメントで `DATA DIRECTORY` 句を指定して外部ディレクトリにテーブルを作成した場合は、宛先インスタンスで置換するテーブルを同じ `DATA DIRECTORY` 句で定義する必要があります。句が一致しない場合は、スキーマの不一致エラーが報告されます。ソーステーブルが `DATA DIRECTORY` 句で定義されているかどうかを確認するには、`SHOW CREATE TABLE` を使用してテーブル定義を表示します。`DATA DIRECTORY` 句の使用の詳細は、[セクション15.6.1.2「外部でのテーブルの作成」](#) を参照してください。
- `ROW_FORMAT` オプションがテーブル定義で明示的に定義されていない場合、または `ROW_FORMAT=DEFAULT` が使用されている場合、`innodb_default_row_format` 設定はソースインスタンスと宛先インスタンスで同じである必要があります。そうしないと、インポート操作を試行したときにスキーマの不一致エラーが報告されます。`SHOW CREATE TABLE` を使用してテーブル定義を確認します。`SHOW VARIABLES` を使用して、`innodb_default_row_format` 設定を確認します。関連情報については、[テーブルの行形式の定義](#) を参照してください。

テーブルのインポート

この例では、file-per-table テーブルスペースに存在する通常の非パーティションテーブルをインポートする方法を示します。

- 宛先インスタンスで、インポートするテーブルと同じ定義でテーブルを作成します。(テーブル定義は、`SHOW CREATE TABLE` 構文を使用して取得できます。) テーブル定義が一致しない場合は、インポート操作を試行するとスキーマの不一致エラーが報告されます。

```
mysql> USE test;
mysql> CREATE TABLE t1 (c1 INT) ENGINE=INNODB;
```

- 宛先インスタンスで、作成したテーブルのテーブルスペースを破棄します。(インポートする前に、受信側のテーブルのテーブルスペースを破棄する必要があります。)

```
mysql> ALTER TABLE t1 DISCARD TABLESPACE;
```

- ソースインスタンスで、`FLUSH TABLES ... FOR EXPORT` を実行して、インポートするテーブルを静止します。テーブルが静止している場合、テーブルでは読み取り専用トランザクションのみが許可されます。

```
mysql> USE test;
mysql> FLUSH TABLES t1 FOR EXPORT;
```

`FLUSH TABLES ... FOR EXPORT` は、サーバーの実行中にバイナリテーブルのコピーを作成できるように、指定されたテーブルへの変更がディスクにフラッシュされていることを確認します。`FLUSH TABLES ... FOR EXPORT` を実行すると、InnoDB によって、テーブルのスキーマディレクトリに `.cfg` メタデータファイルが生成されます。`.cfg` ファイルには、インポート操作中のスキーマ検証に使用されるメタデータが含まれています。

- `.ibd` ファイルおよび `.cfg` メタデータファイルをソースインスタンスから宛先インスタンスにコピーします。例:

```
shell> scp /path/to/datadir/test/t1.{ibd,cfg} destination-server:/path/to/datadir/test
```

`.ibd` ファイルおよび `.cfg` ファイルは、次の手順で示すように、共有ロックを解放する前にコピーする必要があります。

注記

暗号化されたテーブルスペースからテーブルをインポートする場合、InnoDB は `.cfg` メタデータファイルに加えて `.cfp` ファイルを生成します。`.cfp` ファイルは、`.cfg` ファイルとともに宛先インスタンスにコピーする必要があります。`.cfp` ファイルには、転送キーと暗号化されたテーブルスペースキーが含まれます。インポート時に、InnoDB は転送キーを使用してテーブルスペースキーを復号化します。関連情報については、[セクション15.13「InnoDB 保存データ暗号化」](#) を参照してください。

- ソースインスタンスで、`UNLOCK TABLES` を使用して、`FLUSH TABLES ... FOR EXPORT` ステートメントで取得したロックを解放します:

```
mysql> USE test;
```

```
mysql> UNLOCK TABLES;
```

6. 宛先インスタンスで、テーブルスペースをインポートします:

```
mysql> USE test;  
mysql> ALTER TABLE t1 IMPORT TABLESPACE;
```

パーティションテーブルのインポート

この例では、各テーブルパーティションが file-per-table テーブルスペースに存在するパーティションテーブルをインポートする方法を示します。

1. 宛先インスタンスで、インポートするパーティションテーブルと同じ定義でパーティションテーブルを作成します。(テーブル定義は、[SHOW CREATE TABLE](#) 構文を使用して取得できます。) テーブル定義が一致しない場合は、インポート操作を試行するとスキーマの不一致エラーが報告されます。

```
mysql> USE test;  
mysql> CREATE TABLE t1 (i int) ENGINE = InnoDB PARTITION BY KEY (i) PARTITIONS 3;
```

`/datadir/test` ディレクトリには、3つのパーティションごとにテーブルスペースの `.ibd` ファイルがあります。

```
mysql> \! ls /path/to/datadir/test/  
t1.frm t1#p#p0.ibd t1#p#p1.ibd t1#p#p2.ibd
```

2. 宛先インスタンスで、パーティションテーブルのテーブルスペースを破棄します。(インポート操作の前に、受信側のテーブルのテーブルスペースを破棄する必要があります。)

```
mysql> ALTER TABLE t1 DISCARD TABLESPACE;
```

パーティションテーブルの3つのテーブルスペース `.ibd` ファイルが `/datadir/test` ディレクトリから破棄され、次のファイルが残されます:

```
mysql> \! ls /path/to/datadir/test/  
t1.frm
```

3. ソースインスタンスで、[FLUSH TABLES ... FOR EXPORT](#) を実行して、インポートするパーティションテーブルを静止します。テーブルが静止している場合、テーブルでは読み取り専用トランザクションのみが許可されます。

```
mysql> USE test;  
mysql> FLUSH TABLES t1 FOR EXPORT;
```

[FLUSH TABLES ... FOR EXPORT](#) では、サーバーの実行中にバイナリテーブルのコピーを作成できるように、指定されたテーブルへの変更がディスクにフラッシュされます。[FLUSH TABLES ... FOR EXPORT](#) を実行すると、InnoDB によって、各テーブルスペースファイルのテーブルのスキーマディレクトリに `.cfg` メタデータファイルが生成されます。

```
mysql> \! ls /path/to/datadir/test/  
t1#p#p0.ibd t1#p#p1.ibd t1#p#p2.ibd  
t1.frm t1#p#p0.cfg t1#p#p1.cfg t1#p#p2.cfg
```

`.cfg` ファイルには、テーブルスペースのインポート時にスキーマ検証に使用されるメタデータが含まれています。[FLUSH TABLES ... FOR EXPORT](#) は、個々のテーブルパーティションではなく、テーブルでのみ実行できます。

4. `.ibd` および `.cfg` ファイルをソースインスタンスのスキーマディレクトリから宛先インスタンスのスキーマディレクトリにコピーします。例:

```
shell> scp /path/to/datadir/test/t1*.{ibd,cfg} destination-server:/path/to/datadir/test
```

次のステップで説明するように、共有ロックを解放する前に `.ibd` および `.cfg` ファイルをコピーする必要があります。

注記

暗号化されたテーブルスペースからテーブルをインポートする場合、InnoDB は `.cfg` メタデータファイルに加えて `.cpk` ファイルを生成します。`.cpk` ファイルは、`.cfg` ファイルとともに宛先インスタンスにコピーする必要があります。`.cpk` ファイルには、転送キー

と暗号化されたテーブルスペースキーが含まれます。インポート時に、InnoDB は転送キーを使用してテーブルスペースキーを復号化します。関連情報については、[セクション15.13「InnoDB 保存データ暗号化」](#)を参照してください。

5. ソースインスタンスで、`UNLOCK TABLES` を使用して、`FLUSH TABLES ... FOR EXPORT` によって取得されたロックを解放します:

```
mysql> USE test;
mysql> UNLOCK TABLES;
```

6. 宛先インスタンスで、パーティションテーブルのテーブルスペースをインポートします:

```
mysql> USE test;
mysql> ALTER TABLE t1 IMPORT TABLESPACE;
```

テーブルパーティションのインポート

この例では、各パーティションが file-per-table テーブルスペースファイルに存在する個々のテーブルパーティションをインポートする方法を示します。

次の例では、4 つのパーティションテーブルの 2 つのパーティション (`p2` および `p3`) がインポートされます。

1. 宛先インスタンスで、パーティションのインポート元のパーティションテーブルと同じ定義を使用してパーティションテーブルを作成します。(テーブル定義は、`SHOW CREATE TABLE` 構文を使用して取得できます。) テーブル定義が一致しない場合は、インポート操作を試行するとスキーマの不一致エラーが報告されます。

```
mysql> USE test;
mysql> CREATE TABLE t1 (i int) ENGINE = InnoDB PARTITION BY KEY (i) PARTITIONS 4;
```

`/datadir/test` ディレクトリには、4 つのパーティションごとにテーブルスペースの `.ibd` ファイルがあります。

```
mysql> \! ls /path/to/datadir/test/
t1.frm t1#p#p0.ibd t1#p#p1.ibd t1#p#p2.ibd t1#p#p3.ibd
```

2. 宛先インスタンスで、ソースインスタンスからインポートするパーティションを破棄します。(パーティションをインポートする前に、受信側のパーティションテーブルから対応するパーティションを破棄する必要があります。)

```
mysql> ALTER TABLE t1 DISCARD PARTITION p2, p3 TABLESPACE;
```

破棄された 2 つのパーティションのテーブルスペース `.ibd` ファイルが宛先インスタンスの `/datadir/test` ディレクトリから削除され、次のファイルが残されます:

```
mysql> \! ls /path/to/datadir/test/
t1.frm t1#p#p0.ibd t1#p#p1.ibd
```

注記

サブパーティションテーブルで `ALTER TABLE ... DISCARD PARTITION ... TABLESPACE` を実行する場合、パーティションテーブル名とサブパーティションテーブル名の両方が許可されます。パーティション名を指定すると、そのパーティションのサブパーティションが操作に含まれます。

3. ソースインスタンスで、`FLUSH TABLES ... FOR EXPORT` を実行してパーティションテーブルを静止します。テーブルが静止している場合、テーブルでは読み取り専用トランザクションのみが許可されます。

```
mysql> USE test;
mysql> FLUSH TABLES t1 FOR EXPORT;
```

`FLUSH TABLES ... FOR EXPORT` では、インスタンスの実行中にバイナリテーブルのコピーを作成できるように、指定されたテーブルへの変更がディスクにフラッシュされます。`FLUSH TABLES ... FOR EXPORT` を実行すると、InnoDB によって、テーブルのスキーマディレクトリ内のテーブルスペースファイルごとに `.cfg` メタデータファイルが生成されます。

```
mysql> \! ls /path/to/datadir/test/
t1#p#p0.ibd t1#p#p1.ibd t1#p#p2.ibd t1#p#p3.ibd
```

```
t1.frm t1#p#p0.cfg t1#p#p1.cfg t1#p#p2.cfg t1#p#p3.cfg
```

`.cfg` ファイルには、インポート操作中のスキーマ検証に使用されるメタデータが含まれています。`FLUSH TABLES ... FOR EXPORT` は、個々のテーブルパーティションではなく、テーブルでのみ実行できます。

- パーティション `p2` およびパーティション `p3` の `.ibd` および `.cfg` ファイルを、ソースインスタンスのスキーマディレクトリから宛先インスタンスのスキーマディレクトリにコピーします。

```
shell> scp t1#p#p2.ibd t1#p#p2.cfg t1#p#p3.ibd t1#p#p3.cfg destination-server:/path/to/datadir/test
```

次のステップで説明するように、共有ロックを解放する前に `.ibd` および `.cfg` ファイルをコピーする必要があります。

注記

暗号化されたテーブルスペースからパーティションをインポートする場合、InnoDB は `.cfg` メタデータファイルに加えて `.cfp` ファイルを生成します。`.cfp` ファイルは、`.cfg` ファイルとともに宛先インスタンスにコピーする必要があります。`.cfp` ファイルには、転送キーと暗号化されたテーブルスペースキーが含まれます。インポート時に、InnoDB は転送キーを使用してテーブルスペースキーを復号化します。関連情報については、[セクション15.13「InnoDB 保存データ暗号化」](#)を参照してください。

- ソースインスタンスで、`UNLOCK TABLES` を使用して、`FLUSH TABLES ... FOR EXPORT` によって取得されたロックを解放します:

```
mysql> USE test;
mysql> UNLOCK TABLES;
```

- 宛先インスタンスで、テーブルパーティション `p2` および `p3` をインポートします:

```
mysql> USE test;
mysql> ALTER TABLE t1 IMPORT PARTITION p2, p3 TABLESPACE;
```

注記

サブパーティションテーブルで `ALTER TABLE ... IMPORT PARTITION ... TABLESPACE` を実行する場合、パーティションテーブル名とサブパーティションテーブル名の両方が許可されます。パーティション名を指定すると、そのパーティションのサブパーティションが操作に含まれます。

制限

- トランスポータブルテーブルスペース機能は、file-per-table テーブルスペースに存在するテーブルでのみサポートされます。システムテーブルスペースまたは一般テーブルスペースに存在するテーブルではサポートされていません。共有テーブルスペースのテーブルは静止できません。
- 全文検索補助テーブルはフラッシュできないため、`FLUSH TABLES ... FOR EXPORT` は `FULLTEXT` インデックスのあるテーブルではサポートされていません。`FULLTEXT` インデックスを含むテーブルをインポートした後、`OPTIMIZE TABLE` を実行して `FULLTEXT` インデックスを再構築します。または、エクスポート操作の前に `FULLTEXT` インデックスを削除し、宛先インスタンスにテーブルをインポートした後にインデックスを再作成します。
- `.cfg` メタデータファイルの制限により、パーティションテーブルのインポート時にパーティションタイプまたはパーティション定義の違いについてスキーマの不一致は報告されません。カラムの差異がレポートされます。
- MySQL 8.0.19 より前では、インデックスキー部分のソート順序情報は、テーブルスペースのインポート操作中に使用される `.cfg` メタデータファイルに格納されません。したがって、インデックスキー部分のソート順序は昇順(デフォルト)とみなされます。その結果、インポート操作に関係するテーブルが `DESC` インデックスキー部分のソート順序で定義されていて、他のテーブルが意図しない順序でレコードがソートされることがあります。回避策は、影響を受けるインデックスを削除して再作成することです。インデックスキー部分のソート順序の詳細は、[セクション13.1.15「CREATE INDEX ステートメント」](#)を参照してください。

MySQL 8.0.19 で `.cfg` ファイル形式が更新され、インデックスキー部分のソート順情報が含まれるようになりました。前述の問題は、MySQL 8.0.19 サーバーインスタンス間のインポート操作には影響しません。

使用上の注意

- `ALTER TABLE ... IMPORT TABLESPACE` では、テーブルをインポートするために `.cfg` メタデータファイルは必要ありません。ただし、`.cfg` ファイルなしでインポートする場合、メタデータチェックは実行されず、次のような警告が発行されます:

```
Message: InnoDB: IO Read error: (2, No such file or directory) Error opening '\  
test\t.cfg', will attempt to import without schema verification  
1 row in set (0.00 sec)
```

`.cfg` メタデータファイルを使用しないテーブルのインポートは、スキーマの不一致が予想されない場合にのみ考慮する必要があります。`.cfg` ファイルなしでインポートする機能は、メタデータにアクセスできないクラッシュリカバリシナリオで役立ちます。

- Windows では、`InnoDB` はデータベース、テーブルスペース、およびテーブル名を内部的に小文字で格納します。Linux や Unix などの大/小文字が区別されるオペレーティングシステムでのインポートの問題を回避するには、小文字の名前を使用してすべてのデータベース、テーブルスペースおよびテーブルを作成します。名前が小文字で作成されるようにする便利な方法は、サーバーを初期化する前に `lower_case_table_names` を 1 に設定することです。(サーバーの初期化時に使用された設定とは異なる `lower_case_table_names` 設定でサーバーを起動することは禁止されています。)

```
[mysqld]  
lower_case_table_names=1
```

- サブパーティションテーブルで `ALTER TABLE ... DISCARD PARTITION ... TABLESPACE` および `ALTER TABLE ... IMPORT PARTITION ... TABLESPACE` を実行する場合、パーティションテーブル名とサブパーティションテーブル名の両方が許可されます。パーティション名を指定すると、そのパーティションのサブパーティションが操作に含まれます。

内部情報

次の情報では、テーブルのインポート手順中にエラーログに書き込まれる内部およびメッセージについて説明します。

`ALTER TABLE ... DISCARD TABLESPACE` が目的のインスタンスで実行された場合。

- テーブルは X モードでロックされています。
- テーブルスペースがテーブルから切り離されています。

`FLUSH TABLES ... FOR EXPORT` がソースインスタンスで実行された場合。

- エクスポートのためにフラッシュされたテーブルが共有モードでロックされています。
- パージコーディネータのスレッドが停止しています。
- ダーティーページがディスクに同期しています。
- テーブルのメタデータがバイナリの `.cfg` ファイルに書き込まれました。

この操作で予想されるエラーログメッセージです。

```
[Note] InnoDB: Sync to disk of "test"."t1" started.  
[Note] InnoDB: Stopping purge  
[Note] InnoDB: Writing table metadata to './test/t1.cfg'  
[Note] InnoDB: Table "test"."t1" flushed to disk
```

`UNLOCK TABLES` がソースインスタンスで実行された場合。

- バイナリ `.cfg` ファイルが削除されます。
- インポートされたテーブル (または複数のテーブル) の共有ロックが解放され、パージコーディネータのスレッドが再起動されました。

この操作で予想されるエラーログメッセージです。

```
[Note] InnoDB: Deleting the meta-data file './test/t1.cfg'  
[Note] InnoDB: Resuming purge
```

`ALTER TABLE ... IMPORT TABLESPACE` が目的のインスタンスで実行されると、インポートのアルゴリズムはインポートされたテーブルスペースごとに次の操作を実行します。

- テーブルスペースの各ページに破損があるかどうかをチェックします。
- 各ページのスペース ID とログシーケンス番号 (LSN) が更新されます。
- フラグが検証され、ヘッダーページの LSN が更新されます。
- B ツリーページが更新されます。
- ページの状態は、ディスクに書き込まれるようにダーティに設定されます。

この操作で予想されるエラーログメッセージです。

```
[Note] InnoDB: Importing tablespace for table 'test/t1' that was exported  
from host 'host_name'  
[Note] InnoDB: Phase I - Update all pages  
[Note] InnoDB: Sync to disk  
[Note] InnoDB: Sync to disk - done!  
[Note] InnoDB: Phase III - Flush changes to disk  
[Note] InnoDB: Phase IV - Flush complete
```

注記

テーブルスペースが破棄されたこと (目的のテーブルのテーブルスペースを破棄した場合) を伝える警告、および `.ibd` ファイルがないために統計値が計算できなかったことを伝えるメッセージも受け取る場合があります。

```
[Warning] InnoDB: Table "test"."t1" tablespace is set as discarded.  
7f34d9a37700 InnoDB: cannot calculate statistics for table  
"test"."t1" because the .ibd file is missing. For help, please refer to  
http://dev.mysql.com/doc/refman/8.0/en/innodb-troubleshooting.html
```

15.6.1.4 InnoDB テーブルの移動またはコピー

このセクションでは、一部またはすべての InnoDB テーブルを別のサーバーまたはインスタンスに移動またはコピーする方法について説明します。たとえば、MySQL インスタンス全体をより大きい高速なサーバーに移動したり、MySQL インスタンス全体を新しいレプリカサーバーにクローニングしたり、個々のテーブルを別のインスタンスにコピーしてアプリケーションを開発およびテストしたり、データウェアハウスサーバーにコピーしてレポートを生成したりできます。

Windows 上の InnoDB では常に、データベース名およびテーブル名が内部的に小文字で格納されます。バイナリ形式のデータベースを Unix から Windows に、または Windows から Unix に移動するには、すべてのデータベースおよびテーブルを小文字の名前を使用して作成します。これを実現する便利な方法は、データベースやテーブルを作成する前に、`my.cnf` または `my.ini` ファイルの `[mysqld]` セクションに次の行を追加することです。

```
[mysqld]  
lower_case_table_names=1
```

注記

サーバーの初期化時に使用された設定とは異なる `lower_case_table_names` 設定でサーバーを起動することは禁止されています。

InnoDB テーブルを移動またはコピーするための方法は、次のとおりです。

- [テーブルのインポート](#)
- [MySQL Enterprise Backup](#)
- [データファイルのコピー \(コールドバックアップ方式\)](#)
- [論理バックアップからのリストア](#)

テーブルのインポート

file-per-table テーブルスペースに存在するテーブルは、別の MySQL サーバーインスタンスから、またはトランスポートテーブルスペース機能を使用してバックアップからインポートできます。[セクション15.6.1.3「InnoDB テーブルのインポート」](#)を参照してください。

MySQL Enterprise Backup

MySQL Enterprise Backup 製品を使用すると、操作の中断を最小限に抑えながら、実行中の MySQL データベースをバックアップし、データベースの一貫したスナップショットを生成できます。MySQL Enterprise Backup がテーブルをコピーしている場合、読取りおよび書き込みを続行できます。また、MySQL Enterprise Backup では、圧縮バックアップファイルを作成し、テーブルのサブセットをバックアップできます。MySQL のバイナリログと組み合わせると、ポイントインタイムリカバリを実行できます。MySQL Enterprise Backup は、MySQL Enterprise サブスクリプションの一部として含まれています。

MySQL Enterprise Backup についての詳細は、[セクション30.2「MySQL Enterprise Backup の概要」](#)を参照してください。

データファイルのコピー (コールドバックアップ方式)

単に、[セクション15.18.1「InnoDB バックアップ」](#)の「コールドバックアップ」で一覧表示した関連ファイルをすべてコピーするだけで、InnoDB データベースを移動できます。

InnoDB のデータファイルとログファイルは、同じ浮動小数点数形式を持つすべてのプラットフォームでバイナリ互換です。浮動小数点形式が異なっている場合でも、テーブル内で `FLOAT` または `DOUBLE` データ型を使用していなければ、手順は同じです。単に、関連するファイルをコピーするだけです。

file-per-table `.ibd` ファイルを移動またはコピーする場合、データベースディレクトリ名はソースシステムと宛先システムで同じである必要があります。データベース名は、InnoDB の共有テーブルスペース内に格納されているテーブル定義に含まれています。テーブルスペースファイル内に格納されているトランザクション ID およびログシーケンス番号も、データベース間で異なります。

あるデータベースから別のデータベースに `.ibd` ファイルとそれに関連付けられたテーブルを移動するには、`RENAME TABLE` ステートメントを使用します。

```
RENAME TABLE db1.tbl_name TO db2.tbl_name;
```

`.ibd` ファイルの「クリーンな」バックアップがある場合は、次のように、そのバックアップが生成された MySQL インストールにリストアできます。

1. `.ibd` ファイルをコピーすると、テーブルスペース内に格納されたテーブル ID が変更されるため、それ以降はテーブルの削除または切り捨ては実行されなかったはずです。
2. 次の `ALTER TABLE` ステートメントを発行して、現在の `.ibd` ファイルを削除します。

```
ALTER TABLE tbl_name DISCARD TABLESPACE;
```

3. バックアップ `.ibd` ファイルを適切なデータベースディレクトリにコピーします。
4. 次の `ALTER TABLE` ステートメントを発行して、このテーブルで新しい `.ibd` ファイルを使用するように InnoDB に指示します。

```
ALTER TABLE tbl_name IMPORT TABLESPACE;
```

注記

`ALTER TABLE ... IMPORT TABLESPACE` 機能は、インポートされたデータに対して外部キー制約を課しません。

このコンテキストでは、「クリーンな」`.ibd` バックアップファイルとは、次の要件を満たすファイルです。

- `.ibd` ファイル内には、トランザクションによってコミットされていない変更はありません。
- `.ibd` ファイル内にマージされていない挿入バッファエントリはありません。

- パージによって、`.ibd` ファイルから削除マークが付けられたすべてのインデックスレコードが削除されました。
- `mysqld` によって、`.ibd` ファイルの変更されたページがすべてバッファプールからファイルにフラッシュされました。

次の方法を使用すると、クリーンなバックアップ `.ibd` ファイルを作成できます。

1. `mysqld` サーバーからのすべてのアクティビティを停止し、すべてのトランザクションをコミットします。
2. `SHOW ENGINE INNODB STATUS` でデータベース内にアクティブなトランザクションがないことが表示され、`InnoDB` のメインスレッドステータスが「`Waiting for server activity`」になるまで待機します。これにより、`.ibd` ファイルのコピーを作成できるようになります。

`.ibd` ファイルのクリーンなコピーを作成するためのもう 1 つの方法は、MySQL Enterprise Backup 製品を使用することです。

1. MySQL Enterprise Backup を使用して、`InnoDB` インストールをバックアップします。
2. 2 番目の `mysqld` サーバーをバックアップ上で起動します。そのサーバーで、バックアップ内の `.ibd` ファイルがクリーンアップされます。

論理バックアップからのリストア

`mysqldump` などのユーティリティを使用して論理バックアップを実行できます。これにより、別の SQL サーバーに転送するために元のデータベースオブジェクト定義およびテーブルデータを再現するために実行できる一連の SQL ステートメントが生成されます。この方式を使用すれば、形式が異なっているかどうかや、テーブルに浮動小数点データが含まれているかどうかは関係ありません。

この方法のパフォーマンスを向上させるには、データのインポート時に `autocommit` を無効にします。コミットは、テーブル全体またはテーブルのセグメントをインポートした後のみ実行します。

15.6.1.5 MyISAM から InnoDB へのテーブルの変換

信頼性とスケーラビリティを向上させるために `InnoDB` に変換する `MyISAM` テーブルがある場合は、変換する前に次のガイドラインとヒントを確認してください。

注記

以前のバージョンの MySQL で作成されたパーティション化された `MyISAM` テーブルは、MySQL 8.0 と互換性がありません。このようなテーブルは、パーティション化を削除するか、`InnoDB` に変換して、アップグレード前に準備する必要があります。詳細は、[セクション 24.6.2 「ストレージエンジンに関連するパーティショニング制限」](#) を参照してください。

- [MyISAM および InnoDB のメモリー使用量の調整](#)
- [Too-Long または Too-Short トランザクションの処理](#)
- [デッドロックの処理](#)
- [記憶域レイアウトの計画](#)
- [既存テーブルの変換](#)
- [テーブル構造のクローニング](#)
- [既存データの転送](#)
- [ストレージ要件](#)
- [各テーブルに対する PRIMARY KEY の定義](#)
- [アプリケーションのパフォーマンスに関する考慮事項](#)
- [InnoDB テーブルに関連付けられたファイルの理解](#)

MyISAM および InnoDB のメモリー使用量の調整

MyISAM テーブルから移行するときに、結果をキャッシュする際に必要でなくなったメモリーが解放されるように、`key_buffer_size` 構成オプションの値を小さくします。InnoDB テーブル用のキャッシュメモリー割り当てと同様の役割を担う `innodb_buffer_pool_size` 構成オプションの値を大きくします。InnoDB `buffer pool` では、テーブルデータとインデックスデータの両方がキャッシュされるため、クエリーのルックアップが高速化され、再利用のためにクエリー結果がメモリーに保持されます。バッファプールサイズの構成に関するガイダンスについては、[セクション 8.12.3.1「MySQL のメモリーの使用方法」](#) を参照してください。

Too-Long または Too-Short トランザクションの処理

MyISAM テーブルではトランザクションがサポートされていないため、`autocommit` 構成オプションと、`COMMIT` および `ROLLBACK` ステートメントに多くの注意が払われていない可能性があります。これらのキーワードは、複数のセッションが並列して InnoDB テーブルの読み取りおよび書き込みを行うことを許可する際に重要となります。これにより、書き込み負荷の高いワークロードで十分な拡張性の利点が得られます。

トランザクションが開いている間は、トランザクションの開始時に見られるようなデータのスナップショットがシステムで保持されます。これにより、未処理のトランザクションが動作し続けている間に、システムで数百万行の挿入、更新、および削除が行われると、相当なオーバーヘッドが発生する可能性があります。そのため、動作時間が長すぎるトランザクションは回避するように注意してください。

- インタラクティブな実験で `mysql` セッションを使用している場合は、完了後に必ず、(変更を完了させる場合は) `COMMIT`、または (変更を取り消す場合は) `ROLLBACK` を実行します。トランザクションを誤って長期間オープンしたままにしないように、対話型セッションを長期間オープンしたままにしておくのではなく、クローズします。
- アプリケーション内の任意のエラーハンドラでも、不完全な変更の `ROLLBACK` が実行されるか、完了した変更の `COMMIT` が実行されることを確認します。
- `INSERT`、`UPDATE` および `DELETE` の各操作は `COMMIT` より前に InnoDB テーブルに書き込まれるため、`ROLLBACK` は比較的高コストの操作ですが、ほとんどの変更は正常にコミットされ、ロールバックはまれです。大量のデータを使用して実験する際は、多数の行に変更を加えてから、それらの変更をロールバックすることは回避してください。
- 一連の `INSERT` ステートメントを使用して大量のデータをロードする際は、トランザクションが数時間継続することを回避するために、定期的に結果の `COMMIT` を実行します。データウェアハウスの一般的なロード操作では、なんらかの問題が発生した場合、`ROLLBACK` を実行するのではなく、(`TRUNCATE TABLE` を使用して) テーブルを切り捨てて最初からやり直します。

前述のヒントを使用すると、長すぎるトランザクション中に無駄になる可能性のあるメモリーおよびディスク容量を節約できます。トランザクションが本来よりも短い場合は、過剰な I/O が問題となります。MySQL では、`COMMIT` が実行されるたびに、各変更が安全にディスクに記録されていることが確認されます。これには、多少の I/O が伴います。

- InnoDB テーブル上のほとんどの操作では、`autocommit=0` の設定を使用するようにしてください。効率性の観点から見ると、これにより、多数の連続した `INSERT`、`UPDATE`、または `DELETE` ステートメントを発行したときの不要な I/O が回避されます。安全性の観点から見ると、これにより、`mysql` コマンド行またはアプリケーションの例外ハンドラに誤りがあった場合に、`ROLLBACK` ステートメントを発行することで、失ったデータや文字化けしたデータをリカバリできます。
- InnoDB テーブルに `autocommit=1` を設定することが適している状況は、レポートの生成または統計の分析を行うために一連のクエリーを実行するときです。このような状況では、`COMMIT` または `ROLLBACK` に関連する I/O ペナルティーが発生せず、InnoDB は自動的に読み取り専用のワークロードを最適化できます。
- 一連の関連する変更を行う場合は、最後に単一の `COMMIT` を使用して、すべての変更を一度に確定します。たとえば、情報の関連部分を複数のテーブルに挿入する場合は、すべての変更を行なったあとに、`COMMIT` を 1 回実行します。また、連続する多数の `INSERT` ステートメントを実行する場合は、すべてのデータがロードされたあとに、`COMMIT` を 1 回実行します。何百万もの `INSERT` ステートメントを実行する場合は、一万または一千レコードごとに `COMMIT` を発行することで、巨大なトランザクションを分割することがあります。
- `SELECT` ステートメントでもトランザクションが開かれるため、インタラクティブな `mysql` セッションで一部のレポートを実行したり、クエリーをデバッグしたりしたあとは、`COMMIT` を発行するか、または `mysql` セッションを閉じます。

デッドロックの処理

MySQL のエラーログまたは `SHOW ENGINE INNODB STATUS` の出力に、「デッドロック」に言及する警告メッセージが表示されることがあります。デッドロックは、恐ろしい響きの名前にもかかわらず、InnoDB テーブルにとっては重大な問題でなく、修正アクションは何も必要ありません。2つのトランザクションが複数のテーブルを変更し、そのテーブルに別々の順序でアクセスし始めると、各トランザクションが相互に待機し合っ、どちらも処理できない状態に達する可能性があります。deadlock detection が有効になっている場合 (デフォルト)、MySQL はこの条件をただちに検出し、「小さい」トランザクションを取り消して他のトランザクションを続行できるようにします (rolls back)。innodb_deadlock_detect 構成オプションを使用してデッドロック検出が無効になっている場合、InnoDB は、デッドロックの場合にトランザクションをロールバックするために innodb_lock_wait_timeout 設定に依存します。

どちらの方法でも、デッドロックのために強制的に取り消されたトランザクションを再起動するには、アプリケーションにエラー処理ロジックが必要です。以前と同じ SQL ステートメントを再発行すると、元のタイミングの問題は適用されなくなります。他のトランザクションがすでに終了して続行できるか、他のトランザクションがまだ進行中で、トランザクションは終了するまで待機します。

デッドロックの警告が常に発生する場合は、アプリケーションコードを再確認して、一貫性のある方法で SQL 操作を再指示したり、トランザクションを短くしたりすることがあります。innodb_print_all_deadlocks オプションを有効にしてテストすれば、SHOW ENGINE INNODB STATUS 出力の最後の警告だけでなく、MySQL のエラーログにもすべてのデッドロックの警告を表示できます。

詳細は、[セクション15.7.5「InnoDB のデッドロック」](#)を参照してください。

記憶域レイアウトの計画

InnoDB テーブルから最高のパフォーマンスを引き出すために、ストレージレイアウトに関連する数多くのパラメータを調整できます。

大規模で頻繁にアクセスされる MyISAM テーブルを変換し、重要なデータを保持する場合は、CREATE TABLE ステートメントの innodb_file_per_table および innodb_page_size の構成オプションと ROW_FORMAT and KEY_BLOCK_SIZE clauses を調査して検討します。

初期の実験時に、もっとも重要となる設定は innodb_file_per_table です。この設定を有効にすると (デフォルト)、新しい InnoDB テーブルが file-per-table テーブルスペースに暗黙的に作成されます。InnoDB システムテーブルスペースとは対照的に、file-per-table テーブルスペースを使用すると、テーブルの切捨てまたは削除時にオペレーティングシステムでディスク領域を再利用できます。File-per-table テーブルスペースでは、テーブル圧縮、長い可変長カラムの効率的なオフページストレージ、大規模なインデックス接頭辞など、DYNAMIC および COMPRESSED の行形式および関連する機能もサポートされます。詳細は、[セクション15.6.3.2「File-Per-Table テーブルスペース」](#)を参照してください。

複数のテーブルおよびすべての行形式をサポートする共有一般テーブルスペースに InnoDB テーブルを格納することもできます。詳細は、[セクション15.6.3.3「一般テーブルスペース」](#)を参照してください。

既存テーブルの変換

InnoDB を使用するように InnoDB 以外のテーブルを変換するには、ALTER TABLE を使用します。

```
ALTER TABLE table_name ENGINE=InnoDB;
```

テーブル構造のクローニング

切り替える前に、ALTER TABLE を使用して変換を実行するのではなく、MyISAM テーブルのクローンである InnoDB テーブルを作成して、古いテーブルと新しいテーブルを並べてテストできます。

同じカラムとインデックスの定義を持つ空の InnoDB テーブルを作成します。SHOW CREATE TABLE table_name\G を使用して、使用する完全な CREATE TABLE ステートメントを確認します。ENGINE 句を ENGINE=INNODB に変更します。

既存データの転送

前のセクションで示したように、作成された空の InnoDB テーブルに大量のデータを転送するには、INSERT INTO innodb_table SELECT * FROM myisam_table ORDER BY primary_key_columns を使用して行を挿入します。

データを挿入したあとに、InnoDB テーブル用のインデックスを作成することもできます。従来、新しいセカンダリインデックスを作成することは、InnoDB にとって低速な操作でしたが、現在は、インデックスの作成ステップで比較的小さいオーバーヘッドでデータがロードされたあとに、インデックスを作成できるようになりました。

副キー上に **UNIQUE** 制約がある場合は、インポート操作中に一貫性チェックを一時的にオフにすることで、テーブルインポートの速度を上げることができます。

```
SET unique_checks=0;
... import operation ...
SET unique_checks=1;
```

大きなテーブルの場合、InnoDB は **change buffer** を使用してセカンダリインデックスレコードをバッチとして書き込むことができるため、これによりディスク I/O が節約されます。データに重複キーが含まれないようにします。**unique_checks** では、ストレージエンジンが重複キーを無視することが許可されていますが、必須ではありません。

挿入プロセスをより適切に制御するために、大きなテーブルをピース単位で挿入できます：

```
INSERT INTO newtable SELECT * FROM oldtable
WHERE yourkey > something AND yourkey <= somethingelse;
```

すべてのレコードを挿入した後、テーブルの名前を変更できます。

ディスク I/O を削減するには、大きなテーブルの変換時に、最大で物理メモリーの 80% まで InnoDB バッファープールのサイズを大きくします。InnoDB ログファイルのサイズを増やすこともできます。

ストレージ要件

変換プロセス中に InnoDB テーブルのデータの一時コピーを複数作成する場合は、テーブルの削除時にディスク領域を再利用できるように、file-per-table テーブルスペースにテーブルを作成することをお勧めします。**innodb_file_per_table** 構成オプションが有効な場合 (デフォルト)、新しく作成された InnoDB テーブルは file-per-table テーブルスペースに暗黙的に作成されます。

MyISAM テーブルを直接変換するのか、クローンの InnoDB テーブルを作成するのには関係なく、プロセス中に古いテーブルと新しいテーブルの両方を保持するのに十分なディスク領域があることを確認します。InnoDB テーブルには、MyISAM テーブルよりも多くのディスク領域が必要です。**ALTER TABLE** 操作によって領域が使い果たされると、ロールバックが開始されますが、ディスクバウンドの場合は、数時間かかる可能性があります。挿入の場合、InnoDB はバッチ内のインデックスにセカンダリインデックスレコードをマージする際に、挿入バッファを使用します。これにより、大量のディスク I/O が節約されます。ロールバックでは、このようなメカニズムは使用されません。ロールバックは挿入よりも、30 倍長い時間がかかる可能性があります。

ランナウェイロールバックの場合は、データベースに貴重なデータがなければ、何百万ものディスク I/O 操作が完了するまで待機するのではなく、データベースプロセスを強制終了することをお勧めします。完全な手順については、[セクション 15.21.2 「InnoDB のリカバリの強制的な実行」](#) を参照してください。

各テーブルに対する PRIMARY KEY の定義

PRIMARY KEY 句は、MySQL クエリーのパフォーマンスや、テーブルおよびインデックス用の領域使用量に影響を与える重要な要素です。主キーは、テーブル内の行を一意に識別します。テーブル内のすべて行が主キー値を持っている必要があり、2 つの行が同じ主キー値を持つことはできません。

これらは主キーのガイドラインで、その後に詳細な説明が続きます。

- テーブルごとに **PRIMARY KEY** を宣言します。一般に、単一の行を検索するときに参照される **WHERE** 句内のカラムの中で、もっとも重要なものです。
- あとで **ALTER TABLE** ステートメントを使用して追加するのではなく、元の **CREATE TABLE** ステートメントで **PRIMARY KEY** 句を宣言します。
- カラムとそのデータ型は慎重に選択してください。文字または文字列のカラムよりも、数値のカラムを優先してください。
- 別の安定していて、一意で、非 NULL で、数値のカラムが使用できない場合は、自動インクリメントカラムを使用することを検討してください。

- 主キーカラムの値が変更されたかどうか疑わしい場合にも、自動インクリメントは適切な選択です。主キーカラムの値を変更することは、負荷の高い操作であり、テーブル内および各セカンダリインデックス内でデータの再編成が伴う可能性があります。

主キーがまだ存在しないテーブルには、追加することを検討してください。計画されたテーブルの最大サイズに基づいて、現実的な最小の数値型を使用します。これにより、各行をわずかにコンパクトにすることができ、大きなテーブル用に相当な領域を節約できます。主キー値は、セカンダリインデックスが入力されるたびに繰り返されるため、テーブルが任意のセカンダリインデックスを持っている場合は、領域の節約も倍増します。小さな主キーを使用すると、ディスク上のデータサイズが削減されることに加えて、より多くのデータをバッファプール内に収容できるため、すべての種類の操作の速度が上がり、並列性が改善されます。

すでにテーブルの多少長いカラム (VARCHAR など) 上に主キーが存在する場合は、そのカラムがクエリーで参照されていなくても、新しい符号なし AUTO_INCREMENT カラムを追加し、主キーをそのカラムに切り替えることを検討してください。このような設計の変更によって、セカンダリインデックス内の相当な領域を節約できます。以前の主キーカラムを UNIQUE NOT NULL として指定すると、PRIMARY KEY 句と同じ制約を強制的に適用できます (つまり、これらのすべてのカラムにわたって重複する値や NULL 値を回避できます)。

関連する情報を複数のテーブルに分散させる場合は、一般に各テーブルで、その主キー用に同じカラムが使用されます。たとえば、人事部のデータベースには複数のテーブルが含まれ、各テーブルには従業員番号の主キーが含まれている場合があります。営業部のデータベースには、顧客番号の主キーを含むテーブルや、注文番号の主キーを含むテーブルが含まれている場合があります。主キーを使用した検索は非常に高速であるため、このようなテーブルには効率的な結合クエリーを構築できます。

PRIMARY KEY 句を完全に削除すると、MySQL によって自動的に非表示の主キーが作成されます。これは、必要以上に長くなる可能性のある 6 バイトの値であるため、領域が無駄になります。これは非表示であるため、クエリーで参照できません。

アプリケーションのパフォーマンスに関する考慮事項

InnoDB の信頼性およびスケーラビリティ機能には、同等の MyISAM テーブルよりも多くのディスク記憶域が必要です。領域の使用率を改善し、結果セットを処理する際の I/O およびメモリーの消費を削減し、インデックス検索を効率的に使用するクエリーの最適化計画を改善するために、カラムおよびインデックスの定義をわずかに変更することがあります。

主キーに数値の ID カラムを設定する場合 (特に、結合クエリーの場合) は、その値を使用して、その他の任意のテーブル内の関連する値と相互参照します。たとえば、入力として国名を受け入れ、同じ名前を検索するクエリーを実行するのではなく、国 ID を確認するための検索を 1 回実行してから、複数のテーブルにわたって関連情報を検索するための別のクエリー (または 1 回の結合クエリー) を実行します。顧客番号またはカタログ項目番号を数字の文字列として格納すると、数バイトを使い果たす可能性があるため、その代わりに、格納およびクエリー用に数値の ID に変換します。4 バイトの符号なし INT カラムでは、40 億を超える項目 (アメリカ合衆国での billion の意味: 10 億) にインデックスを付けることができます。さまざまな整数型の範囲については、セクション 11.1.2 「整数型 (真数値) - INTEGER、INT、SMALLINT、TINYINT、MEDIUMINT、BIGINT」を参照してください。

InnoDB テーブルに関連付けられたファイルの理解

InnoDB ファイルには、MyISAM ファイルよりも多くの注意と計画が必要です。

- InnoDB のシステムテーブルスペースを表す `ibdata` ファイルは削除しないでください。
- InnoDB テーブルを別のサーバーに移動またはコピーする方法については、セクション 15.6.1.4 「InnoDB テーブルの移動またはコピー」を参照してください。

15.6.1.6 InnoDB での AUTO_INCREMENT 処理

InnoDB には、AUTO_INCREMENT カラムを含むテーブルに行を追加する SQL ステートメントのスケーラビリティおよびパフォーマンスを大幅に向上させる構成可能なロックメカニズムが用意されています。InnoDB テーブルで AUTO_INCREMENT メカニズムを使用するには、AUTO_INCREMENT カラムをインデックスの一部として定義して、最大カラム値を取得するためにテーブルでインデックス付けされた SELECT MAX(ai_col) 参照と同等の操作を実行できるようにする必要があります。一般に、これはカラムをどこかのテーブルインデックスの 1 番目のカラムにすることで実現されます。

このセクションでは、AUTO_INCREMENT ロックモードの動作、様々な AUTO_INCREMENT ロックモード設定の使用上の影響、および InnoDB による AUTO_INCREMENT カウンタの初期化方法について説明します。

- [InnoDB AUTO_INCREMENT のロックモード](#)
- [InnoDB AUTO_INCREMENT ロックモードの使用上の意味](#)
- [InnoDB AUTO_INCREMENT カウンタの初期化](#)
- [メモ](#)

InnoDB AUTO_INCREMENT のロックモード

このセクションでは、自動増分値の生成に使用される `AUTO_INCREMENT` ロックモードの動作と、各ロックモードがレプリケーションに与える影響について説明します。自動増分ロックモードは、起動時に `innodb_autoinc_lock_mode` 構成パラメータを使用して構成されます。

`innodb_autoinc_lock_mode` 設定の説明では、次の用語が使用されます:

- 「INSERT のような」ステートメント

`INSERT`、`INSERT ... SELECT`、`REPLACE`、`REPLACE ... SELECT`、`LOAD DATA` など、テーブル内に新しい行を生成するすべてのステートメントです。「simple-inserts」、「bulk-inserts」および「mixed-mode」の挿入が含まれます。

- 「単純挿入」

挿入行数を事前に (ステートメントの初期処理時に) 決定できるステートメントです。これには、ネストしたサブクエリーを持たない単一行および複数行の `INSERT` および `REPLACE` ステートメントが含まれますが、`INSERT ... ON DUPLICATE KEY UPDATE` は含まれません。

- 「一括挿入」

挿入行数 (および必要な自動インクリメント値の数) が事前にわからないステートメントです。これには、`INSERT ... SELECT`、`REPLACE ... SELECT`、および `LOAD DATA` ステートメントが含まれますが、単純な `INSERT` は含まれません。InnoDB では、各行が処理されるたびに `AUTO_INCREMENT` カラムに新しい値が割り当てられます。

- 「混在モード挿入」

これらは、新しい行の一部 (全部ではない) の自動インクリメント値を指定する「単純挿入」ステートメントです。次の例を示します。c1 はテーブル t1 の `AUTO_INCREMENT` カラムです。

```
INSERT INTO t1 (c1,c2) VALUES (1,'a'), (NULL,'b'), (5,'c'), (NULL,'d');
```

`INSERT ... ON DUPLICATE KEY UPDATE` は別のタイプの「混在モード挿入」で、最悪の場合には実質 `INSERT` のあとに `UPDATE` を実行することに相当しますが、`AUTO_INCREMENT` カラムに割り当てられた値は、更新フェーズで使用される可能性も使用されない可能性もあります。

`innodb_autoinc_lock_mode` 構成パラメータには、3つの設定が可能です。「従来型」、「連続」または「インターリーブ」ロックモードの設定は、それぞれ 0、1 または 2 です。MySQL 8.0 の時点では、インターリーブロックモード (`innodb_autoinc_lock_mode=2`) がデフォルト設定です。MySQL 8.0 より前は、連続ロックモードがデフォルト (`innodb_autoinc_lock_mode=1`) です。

MySQL 8.0 のインターリーブロックモードのデフォルト設定は、デフォルトのレプリケーションタイプとして、ステートメントベースのレプリケーションから行ベースのレプリケーションへの変更を反映しています。ステートメントベースレプリケーションでは、SQL ステートメントの特定のシーケンスに対して自動インクリメント値が予測可能かつ繰り返し可能な順序で割り当てられるように、連続した自動インクリメントロックモードが必要ですが、行ベースレプリケーションは SQL ステートメントの実行順序には影響しません。

- `innodb_autoinc_lock_mode = 0` (「従来」ロックモード)

従来のロックモードでは、`innodb_autoinc_lock_mode` 構成パラメータが MySQL 5.1 で導入される前と同じ動作が提供されます。従来のロックモードオプションは、セマンティックに違いがある可能性があるため、下位互換性、パフォーマンステストおよび混合モードの挿入に関する問題の回避のために提供されています。

このロックモードでは、すべての「INSERT-like」ステートメントは、`AUTO_INCREMENT` カラムを含むテーブルに挿入するための特別なテーブルレベルの `AUTO-INC` ロックを取得します。通常、このロックは (トランザクシ

ンの最後ではなく) ステートメントの最後に保持され、特定の一連の **INSERT** ステートメントに対して予測可能で繰返し可能な順序で自動増分値が割り当てられ、特定のステートメントによって割り当てられた自動増分値が連続していることを確認します。

ステートメントベースレプリケーションの場合、これは、SQL ステートメントがレプリカサーバー上でレプリケートされるときに、ソースサーバー上と同じ値が自動インクリメントカラムに使用されることを意味します。複数の **INSERT** ステートメントの実行結果は決定的で、レプリカはソースと同じデータを再現します。複数の **INSERT** ステートメントによって生成された自動インクリメント値がインターリーブされた場合、2つの同時 **INSERT** ステートメントの結果は非決定的になり、ステートメントベースのレプリケーションを使用してレプリカサーバーに確実に伝播できませんでした。

この点が明確になるように、次のテーブルを使用する例を考えてみましょう。

```
CREATE TABLE t1 (
  c1 INT(11) NOT NULL AUTO_INCREMENT,
  c2 VARCHAR(10) DEFAULT NULL,
  PRIMARY KEY (c1)
) ENGINE=InnoDB;
```

実行中のトランザクションが2つ存在しており、それぞれ **AUTO_INCREMENT** カラムを含むテーブル内に行を挿入しているものとします。1つのトランザクションは1000行を挿入する **INSERT ... SELECT** ステートメントを使用しており、もう1つのトランザクションは1行を挿入する単純な **INSERT** ステートメントを使用しています。

```
Tx1: INSERT INTO t1 (c2) SELECT 1000 rows from another table ...
Tx2: INSERT INTO t1 (c2) VALUES ('xxx');
```

InnoDB では、Tx1 の **INSERT** ステートメントで **SELECT** から取得される行数を事前に確認できず、ステートメントの進行に応じて一度に自動増分値が割り当てられます。ステートメントの終了まで保持されるテーブルレベルロックが存在しているため、ある時点で実行可能な **INSERT** ステートメントはテーブル **t1** を参照している1つのステートメントだけであり、複数ステートメントによって自動インクリメント番号の生成がインターリーブされることはありません。Tx1 **INSERT ... SELECT** ステートメントによって生成される自動インクリメント値は連続しており、Tx2 の **INSERT** ステートメントで使用される(単一の)自動インクリメント値は、最初に実行されるステートメントに応じて、Tx1 で使用されるすべての値より小さくなるか大きくなります。

SQL ステートメントがバイナリログからリプレイされたときに(ステートメントベースレプリケーションを使用している場合、または復旧シナリオで) 同じ順序で実行されているかぎり、結果は Tx1 および Tx2 が最初に実行されたときと同じです。したがって、ステートメントの終了まで保持されるテーブルレベルロックが存在することで、自動インクリメントを使用する **INSERT** ステートメントをステートメントベースのレプリケーションで安全に使用できるようになります。ただし、複数のトランザクションが同時に **INSERT** ステートメントを実行している場合、これらのテーブルレベルのロックによって同時実行性およびスケーラビリティが制限されます。

前述の例でテーブルレベルロックが存在しなかった場合、Tx2 の **INSERT** で使用される自動インクリメントカラムの値は、ステートメントが実際に実行されるタイミングに応じて変更されます。Tx1 の **INSERT** の(実行前や完了後ではなく)実行中に、Tx2 の **INSERT** が実行された場合、その2つの **INSERT** ステートメントで割り当てられる具体的な自動インクリメント値は非決定的となり、実行するたびに値が異なる可能性があります。

consecutive ロックモードでは、InnoDB は、行数が事前にわかっている「単純挿入」ステートメントに対してテーブルレベルの **AUTO-INC** ロックを使用せずに、ステートメントベースのレプリケーションの決定論的な実行および安全性を維持できます。

バイナリログを使用して SQL ステートメントを回復またはレプリケーションの一部としてリプレイしない場合、**interleaved** ロックモードを使用すると、ステートメントによって割り当てられた自動インクリメント番号のギャップを許可し、同時に実行するステートメントによって割り当てられた番号を保持する可能性があります。並列性とパフォーマンスを向上させるために、テーブルレベルの **AUTO-INC** ロックのすべての使用を排除できません。

- `innodb_autoinc_lock_mode = 1` (「連続」ロックモード)

このモードでは、「一括挿入」は特殊な **AUTO-INC** テーブルレベルロックを使用し、そのロックをステートメントの終了まで保持します。これは、**INSERT ... SELECT**、**REPLACE ... SELECT**、**LOAD DATA** のすべてのステートメントに当てはまります。一度に実行できるステートメントは、**AUTO-INC** ロックを保持している1つのステートメントだけです。一括挿入操作のソーステーブルがターゲットテーブルと異なる場合は、ソーステーブルから選択された最初の行で共有ロックが取得された後に、ターゲットテーブルの **AUTO-INC** ロックが取得されます。バルク

挿入操作のソースとターゲットが同じテーブルの場合、選択したすべての行で共有ロックが取得された後に **AUTO-INC** ロックが取得されます。

「単純な挿入」(事前に挿入される行数がわかっている)では、ステートメントが完了するまでではなく、割当てプロセスの間のみ保持される mutex (軽量ロック) の制御下で必要な数の自動増分値を取得することで、テーブルレベルの **AUTO-INC** ロックを回避します。 **AUTO-INC** ロックが別のトランザクションによって保持されていないかぎり、テーブルレベルの **AUTO-INC** ロックは使用されません。別のトランザクションが **AUTO-INC** ロックを保持している場合、「単純挿入」は「一括挿入」であるかのように **AUTO-INC** ロックを待機します。

このロックモードでは、行数が事前にわからない(したがってステートメントの処理中に自動インクリメント番号が割り当てられる) **INSERT** ステートメントが存在する場合には、任意の「**INSERT** のような」ステートメントによって割り当てられたすべての自動インクリメント値が必ず連続した値になるため、その処理は、ステートメントベースのレプリケーションで使用しても安全です。

このロックモードを使用すると、ステートメントベースのレプリケーションで安全に使用できるため、スケーラビリティが大幅に向上します。さらに、「従来」ロックモードの場合と同じく、任意のステートメントによって割り当てられた自動インクリメント番号が連続的になります。自動増分を使用するステートメントには、「従来型」モードと比較してセマンティクスの変更なしがありますが、重要な例外があります。

例外は「混在モード挿入」です。この挿入では、ユーザーは複数行の「単純挿入」で、明示的な値を全部ではなく、一部の行の **AUTO_INCREMENT** カラムに指定します。このような挿入の場合、InnoDB は挿入される行数より多くの自動インクリメント値を割り当てます。ただし、自動的に割り当てられる値はすべて連続的に生成されるため、直前に実行されたステートメントによって生成された自動インクリメント値よりも値が大きくなります。「余分」な番号は失われます。

- `innodb_autoinc_lock_mode = 2` (「インターリーブ」ロックモード)

このロックモードでは、テーブルレベル **AUTO-INC** ロックを使用する「**INSERT** のような」ステートメントは 1 つも存在しないため、複数のステートメントを同時に実行できます。これはもっとも高速で、もっとも拡張性の高いロックモードです。ただし、ステートメントベースのレプリケーションを使用する場合や、リカバリシナリオでバイナリログから SQL ステートメントを再現する際には、安全ではありません。

このロックモードでは、自動インクリメント値は一意であり、並列実行されているすべての「**INSERT** のような」ステートメントにわたって単調に増加することが保証されます。ただし、複数のステートメントが同時に番号を生成している(つまり番号の割り当てが複数のステートメント間でインターリーブされている)可能性があるため、任意のステートメントによって挿入される行に対して生成された値が連続的でない可能性があります。

実行中のステートメントが「単純な挿入」のみで、挿入される行数が事前にわかっている場合、「混合モードの挿入」を除き、単一のステートメントに対して生成される番号にギャップはありません。ただし、「一括挿入」が実行されると、特定のステートメントで割り当てられた自動インクリメント値にギャップが発生する可能性があります。

InnoDB AUTO_INCREMENT ロックモードの使用上の意味

- レプリケーションでの自動インクリメントの使用

ステートメントベースレプリケーションを使用している場合は、`innodb_autoinc_lock_mode` を 0 または 1 に設定し、ソースとその複製で同じ値を使用します。`innodb_autoinc_lock_mode = 2` (「インターリーブ」) またはソースとレプリカが同じロックモードを使用しない構成を使用する場合、レプリカとソースで自動インクリメント値が同じであることは保証されません。

行ベースレプリケーションは SQL ステートメントの実行順序に左右されない(混在形式は、ステートメントベースレプリケーションでは安全でないステートメントで行ベースレプリケーションを使用する)ため、行ベースまたは混在形式レプリケーションを使用している場合は、すべての自動インクリメントロックモードが安全です。

- 「失われた」自動インクリメント値とシーケンスギャップ

すべてのロックモード(0、1、および 2)では、自動インクリメント値を生成したトランザクションがロールバックされると、これらの自動インクリメント値が「失われます」。「**INSERT** のような」ステートメントが完了したかどうか、およびそれを含むトランザクションがロールバックされたかどうかに関係なく、自動インクリメントカラムの値は一度生成されたら、ロールバックできません。このような失われた値は再使用されません。したがって、テーブルの **AUTO_INCREMENT** カラムに格納されている値にはギャップが存在する可能性があります。

- `AUTO_INCREMENT` カラムに `NULL` または `0` を指定

すべてのロックモード (0、1 および 2) で、ユーザーが `INSERT` の `AUTO_INCREMENT` カラムに `NULL` または `0` を指定すると、`InnoDB` はその行を値が指定されていないかのように処理し、新しい値を生成します。

- `AUTO_INCREMENT` カラムへの負の値の割当て

すべてのロックモード (0、1 および 2) では、`AUTO_INCREMENT` カラムに負の値を割り当てる場合、自動増分メカニズムの動作は定義されません。

- `AUTO_INCREMENT` 値が指定された整数型の最大整数より大きい場合

すべてのロックモード (0、1、および 2) では、値が指定された整数型に格納できる最大整数を超えると、自動インクリメントメカニズムの動作は定義されません。

- 「一括挿入」の自動インクリメント値のギャップ

`innodb_autoinc_lock_mode` が 0 (「従来型」) または 1 (「連続」) に設定されている場合、テーブルレベルの `AUTO-INC` ロックはステートメントの最後まで保持され、そのようなステートメントのみを一度に実行できるため、特定のステートメントによって生成される自動増分値はギャップなしで連続しています。

`innodb_autoinc_lock_mode` が 2 (「インターリーブ」) に設定されている場合、「一括挿入」によって生成された自動インクリメント値にギャップが存在する可能性があります。並列実行中の「`INSERT` のような」ステートメントが存在する場合に限ります。

一括挿入では、各ステートメントで必要となる自動インクリメント値の正確な数がわからず、過大評価される可能性があるため、ロックモードが 1 または 2 の場合は、連続したステートメント間でギャップが発生する可能性があります。

- 「混在モード挿入」によって割り当てられる自動インクリメント値

「単純挿入」が (全部ではなく) 一部の結果行の自動インクリメント値を指定する「混在モード挿入」を検討します。このようなステートメントの動作は、ロックモード 0、1、および 2 で異なります。たとえば、`c1` はテーブル `t1` の `AUTO_INCREMENT` カラムで、自動生成されたシーケンス番号の最新値が 100 であるとしてみます。

```
mysql> CREATE TABLE t1 (
-> c1 INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
-> c2 CHAR(1)
-> ) ENGINE = INNODB;
```

ここで、次の「混合モードの挿入」ステートメントについて考えてみます:

```
mysql> INSERT INTO t1 (c1,c2) VALUES (1,'a'), (NULL,'b'), (5,'c'), (NULL,'d');
```

`innodb_autoinc_lock_mode` が 0 (「従来型」) に設定されている場合、4 つの新しい行は次のとおりです:

```
mysql> SELECT c1, c2 FROM t1 ORDER BY c2;
+----+-----+
| c1 | c2 |
+----+-----+
| 1  | a  |
| 101| b  |
| 5  | c  |
| 102| d  |
+----+-----+
```

次に使用可能な自動インクリメント値は 103 です。これは、自動インクリメント値が一度に 1 つずつ割り当てられ、ステートメントの実行開始時に一度に割り当てられるわけではないためです。この結果は、並列実行中の (任意の型の) 「`INSERT` のような」ステートメントが存在するかどうかにかかわらず左右されません。

`innodb_autoinc_lock_mode` を 1 (「連続」) に設定すると、次の 4 つの新しい行も表示されます:

```
mysql> SELECT c1, c2 FROM t1 ORDER BY c2;
+----+-----+
| c1 | c2 |
+----+-----+
| 1  | a  |
```



```
| 101 | b |
| 5 | c |
| 102 | d |
+----+----+
```

ただし、この場合、次に使用可能な自動増分値は 103 ではなく 105 です。これは、ステートメントの処理時に 4 つの自動増分値が割り当てられるためですが、使用されるのは 2 つのみであるためです。この結果は、並列実行中の (任意の型の) 「INSERT のような」ステートメントが存在するかどうかに関係ありません。

`innodb_autoinc_lock_mode` がモード 2 (「インターリーブ」) に設定されている場合、4 つの新しい行は次のとおりです:

```
mysql> SELECT c1, c2 FROM t1 ORDER BY c2;
+----+----+
| c1 | c2 |
+----+----+
| 1 | a |
| x | b |
| 5 | c |
| y | d |
+----+----+
```

`x` および `y` の値は一意で、以前に生成された行より大きくなります。ただし、`x` および `y` の特定の値は、同時に実行するステートメントによって生成される自動インクリメント値の数によって異なります。

最後に、最後に生成された順序番号が 100 の場合に発行される次のステートメントについて考えてみます:

```
mysql> INSERT INTO t1 (c1,c2) VALUES (1,'a'), (NULL,'b'), (101,'c'), (NULL,'d');
```

`innodb_autoinc_lock_mode` 設定では、101 が行 (NULL, 'b') に割り当てられ、行 (101, 'c') の挿入が失敗するため、このステートメントによって重複キーエラー 23000 (Can't write; duplicate key in table) が生成されます。

- 一連の INSERT ステートメントの途中での AUTO_INCREMENT カラム値の変更

MySQL 5.7 以前では、一連の INSERT ステートメントの途中で AUTO_INCREMENT カラムの値を変更すると、「重複エントリ」エラーが発生する可能性があります。たとえば、AUTO_INCREMENT カラムの値を現在の最大自動増分値より大きい値に変更する UPDATE 操作を実行した場合、未使用の自動増分値を指定しなかった後続の INSERT 操作では「重複エントリ」エラーが発生する可能性があります。MySQL 8.0 以降では、AUTO_INCREMENT カラムの値を現在の最大自動増分値より大きい値に変更すると、新しい値が永続化され、後続の INSERT 操作では、新しい大きい値から始まる自動増分値が割り当てられます。この動作を次の例に示します。

```
mysql> CREATE TABLE t1 (
-> c1 INT NOT NULL AUTO_INCREMENT,
-> PRIMARY KEY (c1)
-> ) ENGINE = InnoDB;

mysql> INSERT INTO t1 VALUES(0), (0), (3);

mysql> SELECT c1 FROM t1;
+----+
| c1 |
+----+
| 1 |
| 2 |
| 3 |
+----+

mysql> UPDATE t1 SET c1 = 4 WHERE c1 = 1;

mysql> SELECT c1 FROM t1;
+----+
| c1 |
+----+
| 2 |
| 3 |
| 4 |
+----+

mysql> INSERT INTO t1 VALUES(0);
```

```
mysql> SELECT c1 FROM t1;
+----+
| c1 |
+----+
| 2 |
| 3 |
| 4 |
| 5 |
+----+
```

InnoDB AUTO_INCREMENT カウンタの初期化

このセクションでは、InnoDB が AUTO_INCREMENT カウンタを初期化する方法について説明します。

InnoDB テーブルに AUTO_INCREMENT カラムを指定した場合、インメモリーテーブルオブジェクトには、カラムに新しい値を割り当てるときに使用される自動増分カウンタと呼ばれる特別なカウンタが含まれます。

MySQL 5.7 以前では、自動インクリメントカウンタはメインメモリーにのみ格納され、ディスクには格納されません。サーバーの再起動後に自動インクリメントカウンタを初期化するために、InnoDB は、AUTO_INCREMENT カラムを含むテーブルへの最初の挿入で次のステートメントと同等のステートメントを実行します。

```
SELECT MAX(ai_col) FROM table_name FOR UPDATE;
```

MySQL 8.0 では、この動作は変更されています。現在の最大自動増分カウンタ値は、変更されるたびに redo ログに書き込まれ、各チェックポイントのエンジン専用システムテーブルに保存されます。これらの変更により、現在の最大自動インクリメントカウンタ値がサーバーの再起動後も保持されます。

通常の停止後のサーバーの再起動時に、InnoDB は、データディクショナリのシステムテーブルに格納されている現在の最大自動増分値を使用して、インメモリー自動増分カウンタを初期化します。

クラッシュリカバリ中のサーバーの再起動時に、InnoDB は、データディクショナリのシステムテーブルに格納されている現在の最大自動増分値を使用してインメモリー自動増分カウンタを初期化し、最後のチェックポイント以降に書き込まれた自動増分カウンタ値の redo ログをスキップします。redo ログ値がインメモリーカウンタ値より大きい場合は、redo ログ値が適用されます。ただし、予期しないサーバー終了の場合、以前に割り当てられた自動インクリメント値の再利用は保証できません。INSERT または UPDATE 操作のために現在の最大自動増分値が変更されるたびに、新しい値が redo ログに書き込まれますが、redo ログがディスクにフラッシュされる前に予期しない終了が発生した場合は、サーバーの再起動後に自動増分カウンタが初期化されたときに、以前に割り当てられた値を再利用できません。

InnoDB が SELECT MAX(ai_col) FROM table_name FOR UPDATE ステートメントと同等のものを使用して自動増分カウンタを初期化する唯一の状況は、.cfg メタデータファイルを使用しない importing a table の場合です。それ以外の場合は、現在の最大自動インクリメントカウンタ値が .cfg メタデータファイルから読み取られます (存在する場合)。カウンタ値の初期化とは別に、ALTER TABLE ... AUTO_INCREMENT = N FOR UPDATE ステートメントを使用してカウンタ値を永続カウンタ値以下に設定しようとすると、SELECT MAX(ai_col) FROM table_name ステートメントと同等のものを使用して、テーブルの現在の最大自動増分カウンタ値が決定されます。たとえば、一部のレコードを削除した後で、カウンタ値を小さい値に設定しようとする場合があります。この場合、テーブルを検索して、新しいカウンタ値が実際の現在の最大カウンタ値以下でないことを確認する必要があります。

MySQL 5.7 以前では、サーバーを再起動すると AUTO_INCREMENT = N テーブルオプションの影響が取り消され、CREATE TABLE または ALTER TABLE ステートメントで使用して、それぞれ初期カウンタ値を設定したり、既存のカウンタ値を変更したりできます。MySQL 8.0 では、サーバーを再起動しても AUTO_INCREMENT = N テーブルオプションの影響は取り消されません。自動インクリメントカウンタを特定の値に初期化した場合、または自動インクリメントカウンタ値を大きな値に変更した場合、新しい値はサーバーの再起動後も保持されます。

注記

ALTER TABLE ... AUTO_INCREMENT = N では、自動増分カウンタ値を現在の最大値より大きい値にのみ変更できます。

MySQL 5.7 以前では、ROLLBACK 操作の直後にサーバーを再起動すると、以前にロールバックされたトランザクションに割り当てられた自動増分値が再利用され、現在の最大自動増分値が事実上ロールバックされる可能性があります。MySQL 8.0 では、現在の最大自動増分値が永続化され、以前に割り当てられた値が再利用されなくなります。

自動増分カウンタが初期化される前に `SHOW TABLE STATUS` ステートメントがテーブルを調査する場合、InnoDB はテーブルを開き、データディクショナリのシステムテーブルに格納されている現在の最大自動増分値を使用してカウンタ値を初期化します。この値は、後で挿入または更新するためにメモリーに格納されます。カウンタ値の初期化では、トランザクションの最後まで続くテーブルに対する通常の排他ロック読取りが使用されます。InnoDB は、より大きいユーザー指定の自動インクリメント値を持つ新しく作成されたテーブルの自動インクリメントカウンタを初期化する場合と同じ手順に従います。

自動インクリメントカウンタの初期化後、行の挿入時に自動インクリメント値を明示的に指定しない場合、InnoDB は暗黙的にカウンタを増分し、新しい値をカラムに割り当てます。自動インクリメントカラム値を明示的に指定する行を挿入し、その値が現在の最大カウンタ値より大きい場合、カウンタは指定された値に設定されます。

InnoDB では、サーバーが実行されていれば、インメモリーの自動インクリメントカウンタが使用されます。サーバーが停止して再起動されると、InnoDB は前述のように自動インクリメントカウンタを再初期化します。

`auto_increment_offset` 構成オプションによって、`AUTO_INCREMENT` カラム値の開始点が決まります。デフォルト設定は 1 です。

`auto_increment_increment` 構成オプションは、連続するカラム値の間隔を制御します。デフォルト設定は 1 です。

メモ

`AUTO_INCREMENT` 整数カラムの値を使い果たすと、後続の `INSERT` 操作で重複キーエラーが返されます。これは、MySQL の一般的な動作です。

15.6.2 インデックス

このセクションでは、InnoDB インデックスに関連するトピックについて説明します。

15.6.2.1 クラスタインデックスとセカンダリインデックス

すべての InnoDB テーブルは、行のデータが格納されている **クラスタ化されたインデックス** と呼ばれる特別なインデックスを持っています。一般に、クラスタ化されたインデックスは **主キー** のシノニムです。クエリー、挿入およびその他のデータベース操作から最高のパフォーマンスを得るには、InnoDB がクラスタインデックスを使用して各テーブルの最も一般的な参照および DML 操作を最適化する方法を理解する必要があります。

- テーブル上で **PRIMARY KEY** を定義すると、InnoDB ではそれがクラスタ化されたインデックスとして使用されます。作成するテーブルごとに主キーを定義します。論理的に一意で、Null 以外のカラムまたはカラムのセットが存在しない場合は、自動的に値が入力される新しい **自動インクリメントカラム** を追加します。
- テーブルに **PRIMARY KEY** が定義されていない場合、MySQL はすべてのキーカラムが **NOT NULL** の **UNIQUE** インデックスを最初に検索し、InnoDB はそれをクラスタ化されたインデックスとして使用します。
- テーブルに **PRIMARY KEY** インデックスまたは適切な **UNIQUE** インデックスがない場合、InnoDB は、**GEN_CLUST_INDEX** という名前の非表示のクラスタインデックスを、行 ID 値を含む合成カラムに内部的に生成します。そのようなテーブルでは、InnoDB が行に割り当てる ID に基づいて行の順序付けが行われます。行 ID は、新しい行が挿入されると単調に増加する 6 バイトのフィールドです。したがって、行 ID で順序付けられた行が物理的な挿入順になります。

クラスタ化されたインデックスでクエリーを高速にする方法

クラスタ化されたインデックスから行にアクセスすると、インデックス検索がすべての行データを持つページで直接実行されるため、高速になります。多くの場合、テーブルのサイズが大きい場合にクラスタ化されたインデックスアーキテクチャーを使用すれば、インデックスレコードとは別のページに行データを格納するストレージ編成と比べて、ディスク I/O 操作を節約できます。

セカンダリインデックスとクラスタ化されたインデックスとの関係

クラスタ化されたインデックス以外のインデックスは、すべて **セカンダリインデックス** と呼ばれます。InnoDB では、セカンダリインデックス内の各レコードに、行の主キーカラム、およびセカンダリインデックスに指定されたカラムが含まれます。InnoDB では、クラスタ化されたインデックス内で行を検索する際に、この主キー値が使用されます。

主キーが長くなると、セカンダリインデックスで使用される領域も多くなるため、主キーは短い方が利点があります。

InnoDB のクラスティンデックスおよびセカンダリインデックスを利用するためのガイドラインは、[セクション 8.3「最適化とインデックス」](#)を参照してください。

15.6.2.2 InnoDB インデックスの物理構造

空間インデックスを除き、InnoDB インデックスは [B-tree](#) データ構造です。空間インデックスでは、多次元データのインデックス付けに特化したデータ構造である [R-trees](#) を使用します。インデックスレコードは、B ツリーまたは R ツリーデータ構造のリーフページに格納されます。インデックスページのデフォルトサイズは 16K バイトです。

新しいレコードが [InnoDB clustered index](#) に挿入されると、InnoDB は将来のインデックスレコードの挿入および更新のために 1/16 のページを解放しようとしています。インデックスレコードが順次 (昇順または降順) に挿入されると、インデックスページの約 15/16 までがいっぱいになります。レコードがランダムに挿入された場合は、ページの 1/2 から 15/16 までがいっぱいになります。

InnoDB では、B ツリーインデックスの作成または再構築時にバルクロードが実行されます。このインデックス作成方法は、ソートされたインデックス作成と呼ばれます。`innodb_fill_factor` 構成オプションは、ソートされたインデックスの作成時に入力される各 B ツリーページの領域の割合を定義します。残りの領域は、将来のインデックスの増加のために予約されています。ソートされたインデックス構築は、空間インデックスではサポートされていません。詳細は、[セクション 15.6.2.3「ソートされたインデックス構築」](#)を参照してください。`innodb_fill_factor` を 100 に設定すると、クラスタ化されたインデックスページの領域の 1/16 は将来のインデックスの増加に備えて解放されます。

InnoDB インデックスページの塗りつぶし係数が `MERGE_THRESHOLD` の下にドロップされた場合 (指定されていない場合、デフォルトで 50%)、InnoDB はインデックスツリーを縮小してページを解放しようとしています。`MERGE_THRESHOLD` 設定は、B ツリーインデックスと R ツリーインデックスの両方に適用されます。詳細は、[セクション 15.8.11「インデックスページのマージしきい値の構成」](#)を参照してください。

MySQL インスタンスを初期化する前に `innodb_page_size` 構成オプションを設定することで、MySQL インスタンスのすべての InnoDB テーブルスペースに対して `page size` を定義できます。インスタンスのページサイズを定義した後は、インスタンスを再初期化しないと変更できません。サポートされているサイズは、64KB、32KB、16KB (デフォルト)、8KB および 4KB です。

特定の InnoDB ページサイズを使用している MySQL インスタンスは、別のページサイズを使用するインスタンスのデータファイルやログファイルを使用できません。

15.6.2.3 ソートされたインデックス構築

InnoDB では、インデックスの作成または再構築時に、インデックスレコードを一度に挿入するかわりにバルクロードが実行されます。このインデックス作成方法は、ソートされたインデックス作成とも呼ばれます。ソートされたインデックス構築は、空間インデックスではサポートされていません。

インデックス作成には 3 つのフェーズがあります。最初のフェーズでは、[clustered index](#) がスキャンされ、インデックスエントリが生成されてソートバッファに追加されます。`sort buffer` がいっぱいになると、エントリはソートされ、一時的なファイルに書き込まれます。このプロセスは、「run」とも呼ばれます。2 番目のフェーズでは、1 つ以上の実行が一時的なファイルに書き込まれ、ファイル内のすべてのエントリに対してマージソートが実行されます。3 番目と最後のフェーズでは、ソートされたエントリが [B-tree](#) に挿入されます。

ソートされたインデックス構築が導入される前に、インデックスエントリは挿入 API を使用して一度に 1 つのレコードを B ツリーに挿入されました。この方法では、B ツリー `cursor` を開いて挿入位置を検索し、`optimistic` 挿入を使用して B ツリーページにエントリを挿入します。ページがいっぱいであるために挿入が失敗した場合は、`pessimistic` 挿入が実行されます。これには、B ツリーカーソルをオープンし、必要に応じて B ツリーノードを分割してマージし、エントリの領域を見つけます。インデックスを作成するこの「top-down」メソッドの欠点は、挿入位置の検索と、B ツリーノードの一定の分割およびマージのコストです。

ソートされたインデックス構築では、「bottom-up」アプローチを使用してインデックスを構築します。このアプローチでは、B ツリーのすべてのレベルで右端のリーフページへの参照が保持されます。必要な B ツリー深度の右端のリーフページが割り当てられ、ソート順に従ってエントリが挿入されます。リーフページがいっぱいになると、ノードポインタが親ページに追加され、兄弟リーフページが次の挿入用に割り当てられます。このプロセスは、すべてのエントリが挿入されるまで続行され、ルートレベルまで挿入される可能性があります。兄弟ページが割り当てら

れると、以前に固定されたリーフページへの参照が解放され、新しく割り当てられたリーフページが最も右側のリーフページおよび新しいデフォルトの挿入場所になります。

将来のインデックス増加のための B ツリーページ領域の予約

将来のインデックス増加のために領域を確保するには、`innodb_fill_factor` 構成オプションを使用して、B ツリーページ領域の割合を確保します。たとえば、`innodb_fill_factor` を 80 に設定すると、ソートされたインデックスの作成時に B ツリーページの領域の 20% が予約されます。この設定は、B ツリーリーフページと非リーフページの両方に適用されます。`TEXT` または `BLOB` エントリに使用される外部ページには適用されません。予約される領域の量は、`innodb_fill_factor` 値が強い制限ではなくヒントとして解釈されるため、正確には構成されていない場合があります。

ソートされたインデックス構築および全文インデックスのサポート

ソートされたインデックス構築は、`fulltext indexes` でサポートされています。以前は、SQL を使用して全文インデックスにエントリが挿入されていました。

ソートされたインデックス作成および圧縮されたテーブル

`compressed tables` の場合、以前のインデックス作成方法では、圧縮ページと非圧縮ページの両方にエントリが追加されました。変更ログ (圧縮されたページの空き領域を表す) がいっぱいになると、圧縮されたページが再圧縮されます。領域不足のために圧縮に失敗した場合、ページは分割されます。ソートされたインデックス構築では、エントリは圧縮されていないページにのみ追加されます。圧縮されていないページがいっぱいになると、圧縮されます。適応パディングは、ほとんどの場合に圧縮が成功するようにするために使用されますが、圧縮が失敗した場合は、ページが分割され、圧縮が再試行されます。このプロセスは、圧縮が成功するまで続行されます。B ツリーページの圧縮の詳細は、[セクション 15.9.1.5 「InnoDB テーブルでの圧縮の動作」](#) を参照してください。

ソートされたインデックスの作成および redo ロギング

`Redo logging` は、ソートされたインデックスの作成中は無効になります。かわりに、インデックス作成が予期しない終了または障害に耐えることができるようにする `checkpoint` があります。チェックポイントにより、すべてのデータページが強制的にディスクに書き込まれます。ソートされたインデックスの作成中に、チェックポイント操作を迅速に処理できるように、`dirty pages` をフラッシュするように `page cleaner` スレッドに定期的にシグナルが送信されます。通常、クリーンページ数が設定されたしきい値を下回ると、ページクリーナスレッドはデータページをフラッシュします。ソートされたインデックス構築の場合、データページはチェックポイントのオーバーヘッドを削減し、I/O および CPU アクティビティをパラレル化するためにすぐにフラッシュされます。

ソートされたインデックス構築およびオプティマイザ統計

ソートされたインデックス構築では、`optimizer` 統計が、前述のインデックス作成方法で生成された統計と異なる場合があります。ワークロードのパフォーマンスに影響しない統計の違いは、インデックスの移入に使用されるアルゴリズムが異なることが原因です。

15.6.2.4 InnoDB FULLTEXT インデックス

`FULLTEXT` インデックスは、テキストベースのカラム (`CHAR`、`VARCHAR` または `TEXT` カラム) に作成され、それらのカラムに含まれるデータに対するクエリーおよび DML 操作を高速化し、ストップワードとして定義されている単語を省略します。

`FULLTEXT` インデックスは、`CREATE TABLE` ステートメントの一部として定義されるか、`ALTER TABLE` または `CREATE INDEX` を使用して既存のテーブルに追加されます。

全文検索は、`MATCH() ... AGAINST` 構文を使用して実行されます。用法については、[セクション 12.10 「全文検索関数」](#) を参照してください。

InnoDB `FULLTEXT` インデックスについては、このセクションの次のトピックで説明します:

- [InnoDB 全文インデックスの設計](#)
- [InnoDB 全文インデックステーブル](#)

- InnoDB 全文インデックスキャッシュ
- InnoDB 全文インデックスドキュメント ID および FTS_DOC_ID カラム
- InnoDB による全文インデックスの削除処理
- InnoDB による全文インデックスのトランザクション処理
- InnoDB による全文インデックスのモニター

InnoDB 全文インデックスの設計

InnoDB の FULLTEXT インデックスでは、転置インデックスの設計が使用されています。転置インデックスには、単語のリスト、および単語ごとに、その単語が出現するドキュメントのリストが格納されます。近接検索をサポートするために、単語ごとの位置情報もバイトオフセットとして格納されます。

InnoDB 全文インデックステーブル

InnoDB FULLTEXT インデックスを作成すると、次の例に示すように、インデックステーブルのセットが作成されます:

```
mysql> CREATE TABLE opening_lines (
  id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
  opening_line TEXT(500),
  author VARCHAR(200),
  title VARCHAR(200),
  FULLTEXT idx (opening_line)
) ENGINE=InnoDB;

mysql> SELECT table_id, name, space from INFORMATION_SCHEMA.INNOODB_TABLES
WHERE name LIKE 'test%';
```

table_id	name	space
333	test/fts_0000000000000147_0000000000001c9_index_1	289
334	test/fts_0000000000000147_0000000000001c9_index_2	290
335	test/fts_0000000000000147_0000000000001c9_index_3	291
336	test/fts_0000000000000147_0000000000001c9_index_4	292
337	test/fts_0000000000000147_0000000000001c9_index_5	293
338	test/fts_0000000000000147_0000000000001c9_index_6	294
330	test/fts_0000000000000147_being_deleted	286
331	test/fts_0000000000000147_being_deleted_cache	287
332	test/fts_0000000000000147_config	288
328	test/fts_0000000000000147_deleted	284
329	test/fts_0000000000000147_deleted_cache	285
327	test/opening_lines	283

最初の 6 つのテーブルは転置インデックスを表し、近接検索インデックステーブルと呼ばれます。受信ドキュメントがトークン化されると、個々の単語(「トークン」とも呼ばれる)が、位置情報および関連するドキュメント ID (DOC_ID) とともにインデックステーブルに挿入されます。単語は、最初の文字の文字セットソートの重みに基づいて、6 つのインデックステーブル間で完全にソートおよびパーティション化されます。

転置インデックスは、インデックスの並列作成をサポートするために、6 つの補助インデックステーブルにパーティション化されます。デフォルトでは、2 つのスレッドを使用して、単語および関連するデータのトークン化、ソート、およびインデックステーブルへの挿入が実行されます。スレッドの数は、`innodb_ft_sort_pll_degree` オプションを使用することで構成可能です。大規模なテーブルに FULLTEXT インデックスを作成する場合は、スレッド数を増やすことを検討してください。

補助インデックステーブル名には接頭辞として `fts_` が付き、ポストフィックスとして `index_*` が付きます。各インデックステーブルは、インデックス付きのテーブルの `table_id` と一致するインデックステーブル名に含まれる 16 進値によって、インデックス付きのテーブルに関連付けられます。たとえば、`test/opening_lines` テーブルの `table_id` は 327 (16 進値は 0x147) です。前述の例で示したように、16 進値の「147」は、`test/opening_lines` テーブルに関連付けられたインデックステーブルの名前に表示されます。

FULLTEXT インデックスの `index_id` を表す hex 値は、補助インデックステーブル名にも表示されます。たとえば、補助テーブル名 `test/fts_0000000000000147_0000000000001c9_index_1` では、16 進値 `1c9` の小数値は 457 です。

`opening_lines` テーブル (`idx`) に定義されているインデックスは、`INFORMATION_SCHEMA.INNOODB_INDEXES` テーブルにこの値をクエリーして識別できます (457)。

```
mysql> SELECT index_id, name, table_id, space from INFORMATION_SCHEMA.INNOODB_INDEXES
WHERE index_id=457;
+-----+-----+-----+-----+
| index_id | name | table_id | space |
+-----+-----+-----+-----+
| 457 | idx | 327 | 283 |
+-----+-----+-----+-----+
```

プライマリテーブルが `file-per-table` テーブルスペースに作成される場合、インデックステーブルは独自のテーブルスペースに格納されます。

前述の例に示されているその他のインデックステーブルは、共通インデックステーブルと呼ばれ、`FULLTEXT` インデックスの削除処理および内部状態の格納に使用されます。全文インデックスごとに作成される逆インデックステーブルとは異なり、このテーブルのセットは、特定のテーブルに作成されるすべての全文インデックスに共通です。

全文インデックスが削除されても、共通の補助テーブルは保持されます。全文インデックスを削除すると、`FTS_DOC_ID` カラムの削除にはテーブルの再構築が必要になるため、インデックスに対して作成された `FTS_DOC_ID` カラムは保持されます。`FTS_DOC_ID` カラムを管理するには、共通の軸テーブルが必要です。

- `fts_*_deleted` および `fts_*_deleted_cache`

削除されたが、まだ全文インデックスからデータが削除されていないドキュメントのドキュメント ID (`DOC_ID`) が含まれます。`fts_*_deleted_cache` は、`fts_*_deleted` テーブルのインメモリーバージョンです。

- `fts_*_being_deleted` および `fts_*_being_deleted_cache`

削除され、現在全文インデックスからデータが削除されているドキュメントのドキュメント ID (`DOC_ID`) が含まれます。`fts_*_being_deleted_cache` テーブルは、`fts_*_being_deleted` テーブルのインメモリーバージョンです。

- `fts_*_config`

`FULLTEXT` インデックスの内部状態に関する情報を格納します。もっとも重要な点は、解析され、ディスクにフラッシュされたドキュメントを識別する `FTS_SYNCED_DOC_ID` が格納されることです。クラッシュリカバリの場合、ドキュメントを再解析し、`FULLTEXT` インデックスキャッシュに追加し直すことができるように、ディスクにフラッシュされていないドキュメントを識別する際に、`FTS_SYNCED_DOC_ID` 値が使用されます。このテーブル内のデータを表示するには、`INFORMATION_SCHEMA.INNOODB_FT_CONFIG` テーブルでクエリーを実行します。

InnoDB 全文インデックスキャッシュ

ドキュメントが挿入されると、トークン化され、各単語および関連付けられたデータが `FULLTEXT` インデックスに挿入されます。このプロセスが実行されると、小さなドキュメントの場合でも、補助インデックステーブルへの多数の小規模な挿入が発生します。これにより、競合の発生時に、これらのテーブルへの並列アクセスが発生する可能性があります。この問題を回避するために、`InnoDB` では、最近挿入された行に対するインデックステーブルの挿入を一時的にキャッシュに入れるために、`FULLTEXT` インデックスキャッシュが使用されます。このインメモリーキャッシュの構造では、キャッシュがいっぱいになるまで挿入が保持され、そのあと、ディスク (補助インデックステーブル) にバッチフラッシュされます。`INFORMATION_SCHEMA.INNOODB_FT_INDEX_CACHE` テーブルでクエリーを実行すると、最近挿入された行のトークン化されたデータを表示できます。

キャッシュおよびバッチフラッシュの動作によって、補助インデックステーブルへの頻繁な更新が回避されますが、負荷の高い挿入時および更新時に並列アクセスの問題が発生する可能性があります。また、バッチ技術を使用すると、同じ単語への挿入が複数回発生することも回避され、重複エントリも最小限になります。各単語を個別にフラッシュする代わりに、同じ単語の挿入がマージされ、単一のエントリとしてディスクにフラッシュされるため、補助インデックステーブルのサイズをできるかぎり小さく保ちながら、挿入の効率性が改善されます。

全文インデックスキャッシュのサイズを (テーブルごとに) 構成するには、`innodb_ft_cache_size` 変数が使用されます。これにより、全文インデックスキャッシュがフラッシュされる頻度が影響を受けます。特定のインスタンスで `innodb_ft_total_cache_size` オプションを使用すれば、すべてのテーブルに対応したグローバルな全文インデックスキャッシュのサイズ制限を定義することもできます。

全文インデックスキャッシュには、補助インデックステーブルと同じ情報が格納されます。ただし、全文インデックスキャッシュでは、最近挿入された行のトークン化されたデータのみがキャッシュに入れられます。すでにディスク

(全文補助テーブル)にフラッシュされているデータは、クエリー時に全文インデックスキャッシュに戻りません。補助インデックステーブル内のデータは、直接クエリーが実行されます。補助インデックステーブルからの結果は、全文インデックスキャッシュからの結果とマージされてから返されます。

InnoDB 全文インデックスドキュメント ID および FTS_DOC_ID カラム

InnoDB では、全文インデックス内の単語をその単語が出現するドキュメントレコードとマップする際に、ドキュメント ID (DOC_ID) と呼ばれる一意のドキュメント識別子が使用されます。このマッピングには、インデックス付きテーブル上の FTS_DOC_ID カラムが必要です。FTS_DOC_ID カラムが定義されていない場合は、全文インデックスの作成時に、InnoDB によって自動的に非表示の FTS_DOC_ID カラムが追加されます。次の例で、この動作を実演します。

次のテーブル定義には、FTS_DOC_ID カラムが含まれていません。

```
mysql> CREATE TABLE opening_lines (
  id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
  opening_line TEXT(500),
  author VARCHAR(200),
  title VARCHAR(200)
) ENGINE=InnoDB;
```

CREATE FULLTEXT INDEX 構文を使用して、テーブル上に全文インデックスを作成すると、FTS_DOC_ID カラムが追加されるように InnoDB がテーブルを再構築していることをレポートする警告が返されます。

```
mysql> CREATE FULLTEXT INDEX idx ON opening_lines(opening_line);
Query OK, 0 rows affected, 1 warning (0.19 sec)
Records: 0 Duplicates: 0 Warnings: 1
```

```
mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message                                     |
+-----+-----+-----+
| Warning | 124 | InnoDB rebuilding table to add column FTS_DOC_ID |
+-----+-----+-----+
```

ALTER TABLE を使用して、FTS_DOC_ID カラムが存在しないテーブルに全文インデックスを追加するときにも、同じ警告が返されます。CREATE TABLE の実行時に全文インデックスを作成する場合に、FTS_DOC_ID カラムを定義しないと、InnoDB によって警告なしで、非表示の FTS_DOC_ID カラムが追加されます。

CREATE TABLE 時に FTS_DOC_ID カラムを定義すると、すでにデータがロードされているテーブルに全文インデックスを作成するよりもコストがかかりません。データをロードする前に、テーブル上に FTS_DOC_ID カラムが定義されている場合は、新しいカラムが追加されるようにテーブルおよびそのインデックスを再構築する必要がありません。CREATE FULLTEXT INDEX のパフォーマンスに関心がない場合は、InnoDB で自動的に作成されるように、FTS_DOC_ID カラムを除外します。InnoDB では、FTS_DOC_ID カラムに一意インデックス (FTS_DOC_ID_INDEX) とともに非表示の FTS_DOC_ID カラムが作成されます。独自の FTS_DOC_ID カラムを作成する場合は、次の例のように、カラムを BIGINT UNSIGNED NOT NULL として定義し、FTS_DOC_ID (すべて大文字) という名前を付ける必要があります:

注記

FTS_DOC_ID カラムを AUTO_INCREMENT カラムとして定義する必要はありませんが、AUTO_INCREMENT を使用するとデータのロードが容易になります。

```
mysql> CREATE TABLE opening_lines (
  FTS_DOC_ID BIGINT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
  opening_line TEXT(500),
  author VARCHAR(200),
  title VARCHAR(200)
) ENGINE=InnoDB;
```

FTS_DOC_ID カラムをユーザー自身で定義するように決定した場合は、空の値や重複する値が回避されるようにカラムを管理することがユーザーの責任となります。FTS_DOC_ID 値は再使用できません。つまり、FTS_DOC_ID 値は増加し続けます。

オプションで、FTS_DOC_ID カラムに必要な一意の FTS_DOC_ID_INDEX (すべて大文字) を作成できます。

```
mysql> CREATE UNIQUE INDEX FTS_DOC_ID_INDEX on opening_lines(FTS_DOC_ID);
```

`FTS_DOC_ID_INDEX` を作成しない場合は、InnoDB によって自動的に作成されます。

注記

InnoDB SQL パーサーは降順インデックスを使用しないため、`FTS_DOC_ID_INDEX` は降順インデックスとして定義できません。

最大使用 `FTS_DOC_ID` 値と新しい `FTS_DOC_ID` 値の間の許容ギャップは 65535 です。

テーブルの再構築を回避するために、全文インデックスの削除時に `FTS_DOC_ID` カラムが保持されます。

InnoDB による全文インデックスの削除処理

全文インデックスカラムを含むレコードを削除すると、補助インデックステーブルで多数の小さな削除が発生し、これらのテーブルへの同時アクセスが競合ポイントになる可能性があります。この問題を回避するために、削除されたドキュメントのドキュメント ID (`DOC_ID`) は、インデックス付きテーブルからレコードが削除されるたびに特別な `FTS_*_DELETED` テーブルに記録され、インデックス付きレコードは全文インデックスに残ります。クエリー結果を返す前に、`FTS_*_DELETED` テーブルの情報を使用して、削除されたドキュメント ID を除外します。この設計の利点は、削除が高速で、低負荷であることです。欠点は、レコードの削除後に、すぐにインデックスのサイズが削減されないことです。削除されたレコードの全文インデックスエントリを削除するには、`innodb_optimize_fulltext_only=ON` を使用してインデックス付けされたテーブルで `OPTIMIZE TABLE` を実行し、全文インデックスを再構築します。詳細は、[InnoDB 全文インデックスの最適化](#)を参照してください。

InnoDB による全文インデックスのトランザクション処理

InnoDB の `FULLTEXT` インデックスには、そのキャッシュおよびバッチ処理の動作のために、特別なトランザクション処理の特性が備わっています。特に、`FULLTEXT` インデックス上の更新および挿入は、トランザクションのコミット時に処理されます。つまり、`FULLTEXT` 検索では、コミットされたデータのみを表示できます。次の例で、この動作を実演します。`FULLTEXT` 検索では、挿入された行がコミットされたあとにはじめて、結果が返されます。

```
mysql> CREATE TABLE opening_lines (
  id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
  opening_line TEXT(500),
  author VARCHAR(200),
  title VARCHAR(200),
  FULLTEXT idx (opening_line)
) ENGINE=InnoDB;

mysql> BEGIN;

mysql> INSERT INTO opening_lines(opening_line,author,title) VALUES
('Call me Ishmael.','Herman Melville','Moby-Dick'),
('A screaming comes across the sky.','Thomas Pynchon','Gravity's Rainbow'),
('I am an invisible man.','Ralph Ellison','Invisible Man'),
('Where now? Who now? When now?','Samuel Beckett','The Unnamable'),
('It was love at first sight.','Joseph Heller','Catch-22'),
('All this happened, more or less.','Kurt Vonnegut','Slaughterhouse-Five'),
('Mrs. Dalloway said she would buy the flowers herself.','Virginia Woolf','Mrs. Dalloway'),
('It was a pleasure to burn.','Ray Bradbury','Fahrenheit 451');

mysql> SELECT COUNT(*) FROM opening_lines WHERE MATCH(opening_line) AGAINST('Ishmael');
+-----+
| COUNT(*) |
+-----+
|      0 |
+-----+

mysql> COMMIT;

mysql> SELECT COUNT(*) FROM opening_lines WHERE MATCH(opening_line) AGAINST('Ishmael');
+-----+
| COUNT(*) |
+-----+
|      1 |
+-----+
```

InnoDB による全文インデックスのモニター

次の `INFORMATION_SCHEMA` テーブルでクエリーを実行すると、InnoDB の `FULLTEXT` インデックスの特別なテキスト処理の側面をモニターおよび調査できます。

- `INNODB_FT_CONFIG`
- `INNODB_FT_INDEX_TABLE`
- `INNODB_FT_INDEX_CACHE`
- `INNODB_FT_DEFAULT_STOPWORD`
- `INNODB_FT_DELETED`
- `INNODB_FT_BEING_DELETED`

`INNODB_INDEXES` および `INNODB_TABLES` をクエリーして、`FULLTEXT` のインデックスおよびテーブルの基本情報を表示することもできます。

詳細は、[セクション 15.15.4 「InnoDB INFORMATION_SCHEMA FULLTEXT インデックステーブル」](#) を参照してください。

15.6.3 テーブルスペース

このセクションでは、InnoDB テーブルスペースに関連するトピックについて説明します。

15.6.3.1 システムテーブルスペース

システムテーブルスペースは、変更バッファの記憶域です。file-per-table または一般的なテーブルスペースではなく system テーブルスペースにテーブルが作成されている場合は、テーブルおよびインデックスデータが含まれることもあります。以前のバージョンの MySQL では、システムテーブルスペースに InnoDB データディクショナリが含まれていました。MySQL 8.0 では、InnoDB は MySQL データディクショナリにメタデータを格納します。[第 14 章 「MySQL データディクショナリ」](#) を参照してください。以前の MySQL リリースでは、システムテーブルスペースに二重書き込みバッファ記憶域も含まれていました。この記憶域は、MySQL 8.0.20 の時点で別々の二重書き込みファイルに存在します。[セクション 15.6.4 「二重書き込みバッファ」](#) を参照してください。

システムテーブルスペースには、1 つ以上のデータファイルを含めることができます。デフォルトでは、`ibdata1` という名前の単一のシステムテーブルスペースデータファイルがデータディレクトリに作成されます。システムテーブルスペースデータファイルのサイズと数は、`innodb_data_file_path` 起動オプションによって定義されます。構成情報については、[システムテーブルスペースデータファイル構成](#) を参照してください。

システムテーブルスペースの詳細は、このセクションの次のトピックを参照してください：

- [システムテーブルスペースのサイズ変更](#)
- [システムテーブルスペースに対する RAW ディスクパーティションの使用](#)

システムテーブルスペースのサイズ変更

このセクションでは、システムテーブルスペースのサイズを増減する方法について説明します。

システムテーブルスペースのサイズの増加

システムテーブルスペースのサイズを増やす最も簡単な方法は、自動拡張するように構成することです。これを行うには、`innodb_data_file_path` 設定で最後のデータファイルの `autoextend` 属性を指定し、サーバーを再起動します。

例：

```
innodb_data_file_path=ibdata1:10M:autoextend
```

`autoextend` 属性が指定されている場合、データファイルのサイズは、領域が必要になると 8MB ずつ自動的に増加します。`innodb_autoextend_increment` 変数は増分サイズを制御します。

別のデータファイルを追加して、システムテーブルスペースのサイズを増やすこともできます。これを行うには:

1. MySQL Server を停止させます。
2. `innodb_data_file_path` 設定の最後のデータファイルが `autoextend` 属性で定義されている場合は、それを削除し、現在のデータファイルサイズを反映するように `size` 属性を変更します。指定する適切なデータファイルサイズを決定するには、ファイルシステムでファイルサイズを確認し、その値を最も近い MB 値 (MB は 1024 x 1024) に切り捨てます。
3. `innodb_data_file_path` 設定に新しいデータファイルを追加し、オプションで `autoextend` 属性を指定します。`autoextend` 属性は、`innodb_data_file_path` 設定の最後のデータファイルに対してのみ指定できます。
4. MySQL Server を起動する

たとえば、このテーブルスペースには自動拡張データファイルが 1 つあります:

```
innodb_data_home_dir =  
innodb_data_file_path = ./ibdata1:10M:autoextend
```

データファイルが時間の経過とともに 988MB に増加したとします。これは、現在のデータファイルサイズを反映するように `size` 属性を変更した後、および新しい 50MB の自動拡張データファイルを指定した後の `innodb_data_file_path` 設定です:

```
innodb_data_home_dir =  
innodb_data_file_path = ./ibdata1:988M:/disk2/ibdata2:50M:autoextend
```

新しいデータファイルを追加する場合は、既存のファイル名を指定しないでください。InnoDB では、サーバーの起動時に新しいデータファイルが作成され、初期化されます。

注記

`size` 属性を変更して、既存のシステムテーブルスペースデータファイルのサイズを増やすことはできません。たとえば、`innodb_data_file_path` 設定を `ibdata1:10M:autoextend` から `ibdata1:12M:autoextend` に変更すると、サーバーの起動時に次のエラーが発生します:

```
[ERROR] [MY-012263] [InnoDB] The Auto-extending innodb_system  
data file './ibdata1' is of a different size 640 pages (rounded down to MB) than  
specified in the .cnf file: initial 768 pages, max 0 (relevant if non-zero) pages!
```

このエラーは、既存のデータファイルサイズ (InnoDB ページで表示) が構成ファイルで指定されたサイズと異なることを示しています。このエラーが発生した場合は、以前の `innodb_data_file_path` 設定をリストアし、システムテーブルスペースのサイズ変更手順を参照してください。

InnoDB のページサイズは、`innodb_page_size` 変数によって定義されます。デフォルトは 16384 バイトです。

InnoDB システムテーブルスペースのサイズの縮小

既存のシステムテーブルスペースのサイズの縮小はサポートされていません。小さいシステムテーブルスペースを実現する唯一のオプションは、バックアップから目的のシステムテーブルスペース構成で作成された新しい MySQL インスタンスにデータをリストアすることです。

バックアップの作成の詳細は、[セクション15.18.1「InnoDB バックアップ」](#) を参照してください。

新しいシステムテーブルスペースのデータファイルの構成の詳細。 [システムテーブルスペースデータファイル構成](#) を参照してください。

大規模なシステムテーブルスペースを回避するには、データに file-per-table テーブルスペースを使用することを検討してください。File-per-table テーブルスペースはデフォルトのテーブルスペースタイプで、InnoDB テーブルの作成時に暗黙的に使用されます。システムテーブルスペースとは異なり、file-per-table テーブルスペースで作成されたテーブルを切り捨てたり削除すると、ディスク領域がオペレーティングシステムに戻されます。詳細は、[セクション15.6.3.2「File-Per-Table テーブルスペース」](#) を参照してください。

システムテーブルスペースに対する RAW ディスクパーティションの使用

InnoDB のシステムテーブルスペースでは、データファイルとして RAW ディスクパーティションを使用できます。この方法を使用すると、ファイルシステムのオーバーヘッドが発生せずに、Windows 上および一部の Linux と Unix 上でバッファーに入れられない I/O が有効になります。RAW パーティションを使用する場合と使用しない場合でテストを実行して、この変更によって実際にシステム上のパフォーマンスが改善されるかどうかを確認します。

RAW ディスクパーティションを使用する場合は、MySQL サーバーを実行しているユーザー ID がそのパーティションに対する読み取り権限および書き込み権限を持っていることを確認します。たとえば、mysql ユーザーとしてサーバーを実行する場合は、そのパーティションが mysql によって読み取り可能および書き込み可能である必要があります。--memlock オプションを付けてサーバーを実行する場合は、サーバーを root として実行する必要があるため、パーティションが root によって読み取り可能および書き込み可能である必要があります。

次で説明する手順には、オプションファイルの変更が伴います。追加情報については、[セクション4.2.2.2「オプションファイルの使用」](#)を参照してください。

Linux および Unix システムでの RAW ディスクパーティションの割り当て

1. 新しいデータファイルを作成する際は、innodb_data_file_path オプションのデータファイルサイズの直後に、newraw というキーワードを指定します。パーティションは、少なくとも指定したサイズと同じである必要があります。ディスク指定の 1M バイトは通常 1,000,000 バイトを意味するのに対して、InnoDB 内の 1M バイトは 1024 × 1024 バイトであることに注意してください。

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=/dev/hdd1:3Gnewraw;/dev/hdd2:2Gnewraw
```

2. サーバーを再起動します。InnoDB によって newraw キーワードが認識され、新しいパーティションが初期化されます。ただし、まだ InnoDB テーブルを作成したり変更したりしないでください。そうしなければ、サーバーを次に再起動したときに InnoDB によってパーティションが再初期化され、変更がすべて失われます。(安全策として、InnoDB では、newraw を含むパーティションが指定されたときにユーザーがデータを更新することが回避されます。)
3. InnoDB によって新しいパーティションが初期化されたら、サーバーを停止し、データファイルの指定で newraw を raw に変更します。

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=/dev/hdd1:3Graw;/dev/hdd2:2Graw
```

4. サーバーを再起動します。これにより、InnoDB で変更を行うことが許可されます。

Windows での RAW ディスクパーティションの割り当て

Windows システムでは、Linux および Unix システムで説明したものと手順および付随するガイドラインが適用されます。ただし、Windows では innodb_data_file_path の設定がわずかに異なります。

1. 新しいデータファイルを作成する際は、innodb_data_file_path オプションのデータファイルサイズの直後に、newraw というキーワードを指定します。

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=//./D::10Gnewraw
```

//./ は、物理ドライブにアクセスするための Windows の構文 \\.\ に対応しています。前述の例では、D: がパーティションのドライブ文字です。

2. サーバーを再起動します。InnoDB によって newraw キーワードが認識され、新しいパーティションが初期化されます。
3. InnoDB によって新しいパーティションが初期化されたら、サーバーを停止し、データファイルの指定で newraw を raw に変更します。

```
[mysqld]
```



```
innodb_data_home_dir=  
innodb_data_file_path=//.D::10Graw
```

4. サーバーを再起動します。これにより、InnoDB で変更を行うことが許可されます。

15.6.3.2 File-Per-Table テーブルスペース

file-per-table テーブルスペースには、単一の InnoDB テーブルのデータとインデックスが含まれ、ファイルシステム上の独自のデータファイルに格納されます。

File-per-table テーブルスペースの特性については、このセクションの次のトピックで説明します:

- [File-Per-Table テーブルスペースの構成](#)
- [File-Per-Table テーブルスペースデータファイル](#)
- [テーブルスペースごとのファイルの利点](#)
- [File-Per-Table テーブルスペースのデメリット](#)

File-Per-Table テーブルスペースの構成

InnoDB は、デフォルトで file-per-table テーブルスペースにテーブルを作成します。この動作は、`innodb_file_per_table` 変数によって制御されます。`innodb_file_per_table` を無効にすると、InnoDB によってシステムテーブルスペースにテーブルが作成されます。

`innodb_file_per_table` 設定は、オプションファイルで指定するか、[SET GLOBAL](#) ステートメントを使用して実行時に構成できます。実行時に設定を変更するには、グローバルシステム変数を設定するのに十分な権限が必要です。[セクション5.1.9.1「システム変数権限」](#)を参照してください。

オプションファイル:

```
[mysqld]  
innodb_file_per_table=ON
```

実行時の [SET GLOBAL](#) の使用:

```
mysql> SET GLOBAL innodb_file_per_table=ON;
```

File-Per-Table テーブルスペースデータファイル

file-per-table テーブルスペースは、MySQL データディレクトリの下スキーマディレクトリにある `.ibd` データファイルに作成されます。`.ibd` ファイルには、テーブル (`table_name.ibd`) の名前が付けられます。たとえば、テーブル `test.t1` のデータファイルは、MySQL データディレクトリの下 `test` ディレクトリに作成されます:

```
mysql> USE test;  
  
mysql> CREATE TABLE t1 (  
  id INT PRIMARY KEY AUTO_INCREMENT,  
  name VARCHAR(100)  
) ENGINE = InnoDB;  
  
shell> cd /path/to/mysql/data/test  
shell> ls  
t1.ibd
```

`CREATE TABLE` ステートメントの `DATA DIRECTORY` 句を使用して、データディレクトリ外にファイルごとのテーブルスペースデータファイルを暗黙的に作成できます。詳細は、[セクション15.6.1.2「外部でのテーブルの作成」](#)を参照してください。

テーブルスペースごとのファイルの利点

File-per-table テーブルスペースには、システムテーブルスペースや一般テーブルスペースなどの共有テーブルスペースに比べて次の利点があります。

- file-per-table テーブルスペースで作成されたテーブルを切り捨てるか削除すると、ディスク領域がオペレーティングシステムに戻されます。共有テーブルスペースに格納されているテーブルを切り捨てるか削除すると、共有テーブルスペースデータファイル内に空き領域が作成され、これは **InnoDB** データにのみ使用できます。つまり、テーブルの切捨てまたは削除後、共有テーブルスペースデータファイルのサイズは縮小されません。
- 共有テーブルスペースに存在するテーブルに対するテーブルコピー **ALTER TABLE** 操作では、テーブルスペースが占有するディスク領域の量を増やすことができます。このような操作には、テーブル内のデータとインデックスの追加領域が必要になる場合があります。この領域は file-per-table テーブルスペース用であるため、オペレーティングシステムに解放されません。
- file-per-table テーブルスペースに存在するテーブルで実行すると、**TRUNCATE TABLE** のパフォーマンスが向上します。
- File-per-table テーブルスペースデータファイルは、I/O の最適化、領域管理またはバックアップのために、別々のストレージデバイスに作成できます。 [セクション15.6.1.2「外部でのテーブルの作成」](#) を参照してください。
- file-per-table テーブルスペースに存在するテーブルを別の MySQL インスタンスからインポートできます。 [セクション15.6.1.3「InnoDB テーブルのインポート」](#) を参照してください。
- file-per-table テーブルスペースで作成されたテーブルは、システムテーブルスペースでサポートされていない **DYNAMIC** および **COMPRESSED** の行フォーマットに関連付けられた機能をサポートします。 [セクション15.10「InnoDB の行フォーマット」](#) を参照してください。
- 個々のテーブルスペースデータファイルに格納されたテーブルでは、データ破損が発生した場合、バックアップまたはバイナリログが使用できない場合、または MySQL サーバーインスタンスを再起動できない場合に、時間を節約し、リカバリが成功する可能性を向上させることができます。
- MySQL Enterprise Backup を使用すると、他の **InnoDB** テーブルの使用を中断することなく、file-per-table テーブルスペースに作成されたテーブルを迅速にバックアップまたはリストアできます。これは、様々なバックアップスケジュールに関するテーブルや、バックアップの頻度が低いテーブルに役立ちます。詳細は、[Making a Partial Backup](#) を参照してください。
- File-per-table テーブルスペースでは、テーブルスペースデータファイルのサイズをモニタリングすることによって、ファイルシステム上のテーブルサイズをモニタリングできます。
- **innodb_flush_method** が **O_DIRECT** に設定されている場合、共通の Linux ファイルシステムでは、共有テーブルスペースデータファイルなどの単一ファイルへの同時書き込みは許可されません。その結果、file-per-table テーブルスペースをこの設定と組み合わせて使用すると、パフォーマンスが向上する可能性があります。
- 共有テーブルスペース内のテーブルのサイズは、64TB のテーブルスペースサイズ制限によって制限されます。比較すると、各 file-per-table テーブルスペースには 64TB のサイズ制限があり、個々のテーブルのサイズを大きくするための十分な容量が提供されます。

File-Per-Table テーブルスペースのデメリット

File-per-table テーブルスペースには、システムテーブルスペースや一般テーブルスペースなどの共有テーブルスペースと比較して、次のデメリットがあります。

- file-per-table テーブルスペースでは、同じテーブルの行によってのみ利用できる未使用の領域が各テーブルに存在する可能性があるため、適切に管理されていない場合は領域が無駄になる可能性があります。
- **fsync** 操作は、単一の共有テーブルスペースデータファイルではなく、複数のファイルごとのデータファイルに対して実行されます。 **fsync** 操作はファイル単位であるため、複数のテーブルに対する書き込み操作を組み合わせることはできないため、 **fsync** 操作の合計数が増加する可能性があります。
- **mysqld** では、file-per-table テーブルスペースごとにオープンファイルハンドルを保持する必要があります。file-per-table テーブルスペースに多数のテーブルがある場合、パフォーマンスに影響する可能性があります。
- 各テーブルに独自のデータファイルがある場合は、さらにファイル記述子が必要です。
- 断片化が増える可能性があるため、**DROP TABLE** およびテーブルスキャンのパフォーマンスが低下する可能性があります。ただし、断片化が管理されている場合、file-per-table テーブルスペースを使用すると、これらの操作のパフォーマンスを向上させることができます。

- file-per-table テーブルスペースに存在するテーブルを削除すると、バッファプールがスキャンされます。これは、バッファプールが大きい場合には数秒かかることがあります。スキャンは広範囲に内部ロックをかけて実行されるため、他の操作を遅らせる場合があります。
- 自動拡張共有テーブルスペースファイルがいっぱいになったときにそのサイズを拡張する増分サイズを定義する `innodb_autoextend_increment` 変数は、`innodb_autoextend_increment` 設定に関係なく自動拡張される file-per-table テーブルスペースファイルには適用されません。初期 file-per-table テーブルスペース拡張は少量であり、その後、拡張は 4MB 単位で行われます。

15.6.3.3 一般テーブルスペース

一般的なテーブルスペースは、`CREATE TABLESPACE` 構文を使用して作成される共有 InnoDB テーブルスペースです。テーブルスペースの一般的な機能については、このセクションの次のトピックで説明します：

- [一般的なテーブルスペース機能](#)
- [一般的なテーブルスペースの作成](#)
- [一般テーブルスペースへのテーブルの追加](#)
- [一般的なテーブルスペースの行形式のサポート](#)
- [ALTER TABLE を使用したテーブルスペース間のテーブルの移動](#)
- [一般テーブルスペースの名前の変更](#)
- [一般テーブルスペースの削除](#)
- [テーブルスペースの一般的な制限事項](#)

一般的なテーブルスペース機能

一般的なテーブルスペース機能には、次の機能があります：

- システムテーブルスペースと同様に、一般テーブルスペースは、複数のテーブルのデータを格納できる共有テーブルスペースです。
- 一般的なテーブルスペースには、[file-per-table tablespaces](#) よりも潜在的なメモリー上の利点があります。サーバーは、テーブルスペースの存続期間中、テーブルスペースメタデータをメモリーに保持します。一般的なテーブルスペースが少ない複数のテーブルは、個別の file-per-table テーブルスペース内の同じ数のテーブルよりも、テーブルスペースメタデータのメモリー消費量が少なくなります。
- 一般的なテーブルスペースデータファイルは、MySQL データディレクトリに対して相対的または独立したディレクトリに配置できます。これにより、[file-per-table tablespaces](#) のデータファイルおよび記憶域管理機能の多くが提供されます。file-per-table テーブルスペースと同様に、MySQL データディレクトリ外にデータファイルを配置する機能を使用すると、重要なテーブルのパフォーマンスを個別に管理したり、特定のテーブル用に RAID または DRBD を設定したり、テーブルを特定のディスクにバインドしたりできます。
- 一般テーブルスペースでは、すべてのテーブルの行フォーマットおよび関連機能がサポートされています。
- `TABLESPACE` オプションを `CREATE TABLE` とともに使用すると、一般的なテーブルスペース、file-per-table テーブルスペースまたは system テーブルスペースにテーブルを作成できます。
- `TABLESPACE` オプションを `ALTER TABLE` とともに使用すると、一般的なテーブルスペース、file-per-table テーブルスペースおよびシステムテーブルスペース間でテーブルを移動できます。以前は、file-per-table テーブルスペースから system テーブルスペースにテーブルを移動できませんでした。一般テーブルスペース機能を使用して、これを実行できるようになりました。

一般的なテーブルスペースの作成

一般テーブルスペースは、`CREATE TABLESPACE` 構文を使用して作成されます。

```
CREATE TABLESPACE tablespace\_name
```

```
[ADD DATAFILE 'file_name']  
[FILE_BLOCK_SIZE = value]  
[ENGINE [=] engine_name]
```

一般的なテーブルスペースは、データディレクトリ内またはその外部に作成できます。暗黙的に作成された file-per-table テーブルスペースとの競合を回避するために、データディレクトリの下の子ディレクトリに一般テーブルスペースを作成することはサポートされていません。データディレクトリ外に一般的なテーブルスペースを作成する場合、そのディレクトリが存在し、テーブルスペースを作成する前に InnoDB で認識されている必要があります。不明なディレクトリを InnoDB で認識できるようにするには、ディレクトリを `innodb_directories` 引数値に追加します。`innodb_directories` は読み取り専用の起動オプションです。構成するには、サーバーを再起動する必要があります。

例:

データディレクトリに一般的なテーブルスペースを作成します:

```
mysql> CREATE TABLESPACE `ts1` ADD DATAFILE 'ts1.ibd' Engine=InnoDB;
```

または

```
mysql> CREATE TABLESPACE `ts1` Engine=InnoDB;
```

`ADD DATAFILE` 句は、MySQL 8.0.14 の時点ではオプションであり、その前に必要です。テーブルスペースの作成時に `ADD DATAFILE` 句が指定されていない場合、一意のファイル名を持つテーブルスペースデータファイルが暗黙的に作成されます。一意のファイル名は、ダッシュ (aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee) で区切られた 16 進数の 5 つのグループにフォーマットされた 128 ビット UUID です。一般的なテーブルスペースデータファイルには、`.ibd` ファイル拡張子が含まれます。レプリケーション環境では、ソースで作成されたデータファイル名は、レプリカで作成されたデータファイル名と同じではありません。

データディレクトリ外のディレクトリに一般テーブルスペースを作成します:

```
mysql> CREATE TABLESPACE `ts1` ADD DATAFILE '/my/tablespace/directory/ts1.ibd' Engine=InnoDB;
```

テーブルスペースディレクトリがデータディレクトリの下にないかぎり、データディレクトリへの相対パスを指定できます。この例では、`my_tablespace` ディレクトリはデータディレクトリと同じレベルにあります:

```
mysql> CREATE TABLESPACE `ts1` ADD DATAFILE './my_tablespace/ts1.ibd' Engine=InnoDB;
```

注記

`ENGINE = InnoDB` 句を `CREATE TABLESPACE` ステートメントの一部として定義するか、`InnoDB` をデフォルトのストレージエンジン (`default_storage_engine=InnoDB`) として定義する必要があります。

一般テーブルスペースへのテーブルの追加

InnoDB の一般テーブルスペースを作成した後、次の例に示すように、`CREATE TABLE tbl_name ... TABLESPACE [=] tablespace_name` または `ALTER TABLE tbl_name TABLESPACE [=] tablespace_name` を使用してテーブルスペースにテーブルを追加できます:

CREATE TABLE:

```
mysql> CREATE TABLE t1 (c1 INT PRIMARY KEY) TABLESPACE ts1;
```

ALTER TABLE:

```
mysql> ALTER TABLE t2 TABLESPACE ts1;
```

注記

共有テーブルスペースへのテーブルパーティションの追加のサポートは、MySQL 5.7.24 で非推奨になり、MySQL 8.0.13 で削除されました。共有テーブルスペースには、`InnoDB` システムテーブルスペースおよび一般テーブルスペースが含まれます。

構文の詳細は、[CREATE TABLE](#) および [ALTER TABLE](#) を参照してください。

一般的なテーブルスペースの行形式のサポート

一般テーブルスペースでは、物理ページサイズが異なるために圧縮テーブルと非圧縮テーブルを同じ一般テーブルスペースに共存できないという注意事項があるすべてのテーブルの行形式 ([REDUNDANT](#), [COMPACT](#), [DYNAMIC](#), [COMPRESSED](#)) がサポートされています。

一般的なテーブルスペースに圧縮テーブル ([ROW_FORMAT=COMPRESSED](#)) を含めるには、[FILE_BLOCK_SIZE](#) を指定する必要があり、[FILE_BLOCK_SIZE](#) 値は [innodb_page_size](#) 値と比較した有効な圧縮ページサイズである必要があります。また、圧縮テーブル ([KEY_BLOCK_SIZE](#)) の物理ページサイズは [FILE_BLOCK_SIZE/1024](#) と同じである必要があります。たとえば、[innodb_page_size=16KB](#) および [FILE_BLOCK_SIZE=8K](#) の場合、テーブルの [KEY_BLOCK_SIZE](#) は 8 である必要があります。

次のテーブルに、許可される [innodb_page_size](#)、[FILE_BLOCK_SIZE](#) および [KEY_BLOCK_SIZE](#) の組合せを示します。[FILE_BLOCK_SIZE](#) 値はバイト単位で指定することもできます。特定の [FILE_BLOCK_SIZE](#) に対して有効な [KEY_BLOCK_SIZE](#) 値を決定するには、[FILE_BLOCK_SIZE](#) 値を 1024 で除算します。テーブル圧縮では、32K および 64K の InnoDB ページサイズはサポートされていません。[KEY_BLOCK_SIZE](#) の詳細は、[CREATE TABLE](#) および [セクション15.9.1.2「圧縮テーブルの作成」](#) を参照してください。

表 15.3 圧縮テーブルで許可されるページサイズ、[FILE_BLOCK_SIZE](#) および [KEY_BLOCK_SIZE](#) の組合せ

InnoDB ページサイズ (innodb_page_size)	許可される FILE_BLOCK_SIZE 値	許可される KEY_BLOCK_SIZE 値
64KB	64K (65536)	圧縮はサポートされません
32KB	32K (32768)	圧縮はサポートされません
16KB	16K (16384)	N/A: innodb_page_size が FILE_BLOCK_SIZE と等しい場合、テーブルスペースに圧縮テーブルを含めることはできません。
16KB	8K (8192)	8
16KB	4K (4096)	4
16KB	2K (2048)	2
16KB	1K (1024)	1
8KB	8K (8192)	N/A: innodb_page_size が FILE_BLOCK_SIZE と等しい場合、テーブルスペースに圧縮テーブルを含めることはできません。
8KB	4K (4096)	4
8KB	2K (2048)	2
8KB	1K (1024)	1
4KB	4K (4096)	N/A: innodb_page_size が FILE_BLOCK_SIZE と等しい場合、テーブルスペースに圧縮テーブルを含めることはできません。
4KB	2K (2048)	2
4KB	1K (1024)	1

この例では、一般的なテーブルスペースを作成し、圧縮テーブルを追加する方法を示します。この例では、デフォルトの [innodb_page_size](#) が 16KB であると想定しています。8192 の [FILE_BLOCK_SIZE](#) では、圧縮テーブルの [KEY_BLOCK_SIZE](#) が 8 である必要があります。

```
mysql> CREATE TABLESPACE `ts2` ADD DATAFILE `ts2.ibd` FILE_BLOCK_SIZE = 8192 Engine=InnoDB;
```

```
mysql> CREATE TABLE t4 (c1 INT PRIMARY KEY) TABLESPACE ts2 ROW_FORMAT=COMPRESSED KEY_BLOCK_SIZE=8;
```

一般テーブルスペースの作成時に `FILE_BLOCK_SIZE` を指定しない場合、`FILE_BLOCK_SIZE` はデフォルトで `innodb_page_size` に設定されます。`FILE_BLOCK_SIZE` が `innodb_page_size` と等しい場合、テーブルスペースには、圧縮されていない行形式 (`COMPACT`、`REDUNDANT` および `DYNAMIC` の行形式) のテーブルのみを含めることができます。

ALTER TABLE を使用したテーブルスペース間のテーブルの移動

`ALTER TABLE` を `TABLESPACE` オプションとともに使用して、テーブルを既存の一般テーブルスペース、新しい file-per-table テーブルスペースまたはシステムテーブルスペースに移動できます。

注記

共有テーブルスペースへのテーブルパーティションの配置のサポートは、MySQL 5.7.24 で非推奨になり、MySQL 8.0.13 が削除されました。共有テーブルスペースには、`InnoDB` システムテーブルスペースおよび一般テーブルスペースが含まれます。

file-per-table テーブルスペースまたは system テーブルスペースから general テーブルスペースにテーブルを移動するには、general テーブルスペースの名前を指定します。一般テーブルスペースが存在する必要があります。詳細は、`CREATE TABLESPACE` を参照してください。

```
ALTER TABLE tbl_name TABLESPACE [=] tablespace_name;
```

一般的なテーブルスペースまたはファイルごとのテーブルスペースからシステムテーブルスペースにテーブルを移動するには、テーブルスペース名として `innodb_system` を指定します。

```
ALTER TABLE tbl_name TABLESPACE [=] innodb_system;
```

システムテーブルスペースまたは一般テーブルスペースから file-per-table テーブルスペースにテーブルを移動するには、テーブルスペース名として `innodb_file_per_table` を指定します。

```
ALTER TABLE tbl_name TABLESPACE [=] innodb_file_per_table;
```

`ALTER TABLE ... TABLESPACE` 操作では、`TABLESPACE` 属性が以前の値から変更されていない場合でも、常に全テーブルが再構築されます。

`ALTER TABLE ... TABLESPACE` 構文では、一時テーブルスペースから永続テーブルスペースへのテーブルの移動はサポートされていません。

`DATA DIRECTORY` 句は `CREATE TABLE ... TABLESPACE=innodb_file_per_table` で使用できますが、それ以外の場合は `TABLESPACE` オプションと組み合わせて使用することはサポートされていません。MySQL 8.0.21 では、`DATA DIRECTORY` 句で指定されたディレクトリは `InnoDB` で認識されている必要があります。詳細は、`DATA DIRECTORY 句の使用` を参照してください。

暗号化されたテーブルスペースからテーブルを移動する場合は、制限が適用されます。`暗号化の制限事項` を参照してください。

一般テーブルスペースの名前の変更

一般的なテーブルスペースの名前変更は、`ALTER TABLESPACE ... RENAME TO` 構文を使用してサポートされます。

```
ALTER TABLESPACE s1 RENAME TO s2;
```

一般的なテーブルスペースの名前を変更するには、`CREATE TABLESPACE` 権限が必要です。

`RENAME TO` 操作は、`autocommit` の設定に関係なく、`autocommit` モードで暗黙的に実行されます。

テーブルスペースに存在するテーブルに対して `LOCK TABLES` または `FLUSH TABLES WITH READ LOCK` が有効になっている間は、`RENAME TO` 操作を実行できません。

排他的 `metadata locks` は、テーブルスペースの名前が変更されている間、一般的なテーブルスペース内のテーブルに対して取得されるため、同時 DDL が回避されます。同時 DML がサポートされています。

一般テーブルスペースの削除

`DROP TABLESPACE` ステートメントを使用して、InnoDB の一般テーブルスペースを削除します。

`DROP TABLESPACE` 操作の前に、すべてのテーブルをテーブルスペースから削除する必要があります。テーブルスペースが空でない場合、`DROP TABLESPACE` はエラーを返します。

一般的なテーブルスペースのテーブルを識別するには、次のようなクエリーを使用します。

```
mysql> SELECT a.NAME AS space_name, b.NAME AS table_name FROM INFORMATION_SCHEMA.INNODB_TABLESPACES a,
INFORMATION_SCHEMA.INNODB_TABLES b WHERE a.SPACE=b.SPACE AND a.NAME LIKE 'ts1';
+-----+-----+
| space_name | table_name |
+-----+-----+
| ts1       | test/t1   |
| ts1       | test/t2   |
| ts1       | test/t3   |
+-----+-----+
```

一般的な InnoDB テーブルスペースは、テーブルスペースの最後のテーブルが削除されても自動的に削除されません。テーブルスペースは、`DROP TABLESPACE tablespace_name` を使用して明示的に削除する必要があります。

一般テーブルスペースは、特定のデータベースに属していません。`DROP DATABASE` 操作では、一般的なテーブルスペースに属するテーブルを削除できますが、`DROP DATABASE` 操作でテーブルスペースに属するすべてのテーブルを削除しても、テーブルスペースは削除できません。一般テーブルスペースは、`DROP TABLESPACE tablespace_name` を使用して明示的に削除する必要があります。

システムテーブルスペースと同様に、一般テーブルスペースに格納されているテーブルの切捨てまたは削除によって、新しい InnoDB データにのみ使用できる空き領域が一般テーブルスペース `.ibd data file` に内部的に作成されます。`DROP TABLE` 操作中に file-per-table テーブルスペースが削除された場合とは異なり、領域はオペレーティングシステムに解放されません。

この例では、InnoDB の一般テーブルスペースを削除する方法を示します。一般的なテーブルスペース `ts1` は、単一のテーブルで作成されます。テーブルスペースを削除する前に、テーブルを削除する必要があります。

```
mysql> CREATE TABLESPACE 'ts1' ADD DATAFILE 'ts1.ibd' Engine=InnoDB;
mysql> CREATE TABLE t1 (c1 INT PRIMARY KEY) TABLESPACE ts1 Engine=InnoDB;
mysql> DROP TABLE t1;
mysql> DROP TABLESPACE ts1;
```

注記

`tablespace_name` は、MySQL では大/小文字が区別される識別子です。

テーブルスペースの一般的な制限事項

- 生成されたテーブルスペースまたは既存のテーブルスペースを一般テーブルスペースに変更することはできません。
- 一時一般テーブルスペースの作成はサポートされていません。
- 一般テーブルスペースでは、一時テーブルはサポートされていません。
- システムテーブルスペースと同様に、一般テーブルスペースに格納されているテーブルの切捨てまたは削除によって、新しい InnoDB データにのみ使用できる空き領域が一般テーブルスペース `.ibd data file` に内部的に作成されます。領域は、`file-per-table` テーブルスペース用であるため、オペレーティングシステムに解放されません。

また、共有テーブルスペース (一般テーブルスペースまたはシステムテーブルスペース) に存在するテーブルに対するテーブルコピー `ALTER TABLE` 操作によって、テーブルスペースで使用される領域の量を増やすことができます。このような操作には、テーブルのデータとインデックスと同じ追加領域が必要です。テーブルのコピー `ALTER TABLE` 操作に必要な追加領域は、file-per-table テーブルスペース用であるため、オペレーティングシステムに解放されません。

- `ALTER TABLE ... DISCARD TABLESPACE` および `ALTER TABLE ... IMPORT TABLESPACE` は、一般テーブルスペースに属するテーブルではサポートされていません。
- 一般的なテーブルスペースへのテーブルパーティションの配置のサポートは、MySQL 5.7.24 で非推奨になり、MySQL 8.0.13 で削除されました。
- `ADD DATAFILE` 句は、ソースとレプリカが同じホストに存在するレプリケーション環境ではサポートされません。これは、ソースとレプリカが同じ場所に同じ名前のテーブルスペースを作成するためですが、これはサポートされていないためです。ただし、`ADD DATAFILE` 句を省略した場合、テーブルスペースは一意的生成済ファイル名でデータディレクトリに作成されますが、これは許可されます。
- MySQL 8.0.21 では、InnoDB で直接認識されていないが、undo テーブルスペースディレクトリ (`innodb_undo_directory`) に一般テーブルスペースを作成できません。既知のディレクトリは、`datadir`、`innodb_data_home_dir` および `innodb_directories` 変数で定義されているディレクトリです。

15.6.3.4 undo テーブルスペース

undo テーブルスペースには undo ログが含まれます。undo ログレコードには、クラスタ化されたインデックスレコードに対するトランザクションによる最新の変更を取り消す方法に関する情報が含まれます。undo ログは、ロールバックセグメント内に含まれる undo ログセグメント内に存在します。`innodb_rollback_segments` 変数は、各 undo テーブルスペースに割り当てられるロールバックセグメントの数を定義します。

MySQL インスタンスが初期化されると、2 つのデフォルト undo テーブルスペースが作成されます。SQL ステートメントを受け入れる前に存在する必要があるロールバックセグメントの場所を提供するために、初期化時にデフォルトの undo テーブルスペースが作成されます。undo テーブルスペースの自動切捨てをサポートするには、2 つ以上の undo テーブルスペースが必要です。[undo テーブルスペースの切捨て](#) を参照してください。

デフォルトの undo テーブルスペースは、`innodb_undo_directory` 変数で定義された場所に作成されます。`innodb_undo_directory` 変数が定義されていない場合、デフォルトの undo テーブルスペースがデータディレクトリに作成されます。デフォルトの undo テーブルスペースデータファイルには、`undo_001` および `undo_002` という名前が付けられます。データディレクトリに定義されている対応する undo テーブルスペース名は、`innodb_undo_001` および `innodb_undo_002` です。

MySQL 8.0.14 では、SQL を使用して実行時に追加の undo テーブルスペースを作成できます。[undo テーブルスペースの追加](#) を参照してください。

MySQL 8.0.23 より前のリリースでは、UNDO テーブルスペースの初期サイズは `innodb_page_size` の値によって異なります。デフォルトの 16KB ページサイズの場合、undo テーブルスペースの初期ファイルサイズは 10MiB です。4KB、8KB、32KB および 64KB のページサイズの場合、初期 undo テーブルスペースファイルサイズはそれぞれ 7MiB、8MiB、20MiB および 40MiB です。MySQL 8.0.23 では、初期 UNDO テーブルスペースサイズは通常 16MiB です。切捨て操作によって新しい UNDO テーブルスペースが作成される場合、初期サイズが異なることがあります。この場合、ファイル拡張子のサイズが 16MB を超え、前のファイル拡張子が最後の秒以内に発生した場合、新しい UNDO テーブルスペースは `innodb_max_undo_log_size` 変数で定義されたサイズの四半期に作成されます。

MySQL 8.0.23 より前は、UNDO テーブルスペースは一度に 4 エクステント拡張されます。MySQL 8.0.23 からは、UNDO テーブルスペースは 16MB 以上拡張されます。積極的な増加に対処するために、以前のファイル拡張子が 0.1 秒未満になった場合、ファイル拡張子のサイズは倍増します。拡張サイズの倍増は、最大 256MB まで複数回発生する可能性があります。以前のファイル拡張子が 0.1 秒より前に発生した場合、拡張子のサイズは半分に縮小されます。これも、複数回出現する可能性があります (16MB 以上)。UNDO テーブルスペースに `AUTOEXTEND_SIZE` オプションが定義されている場合、前述のロジックで決定された `AUTOEXTEND_SIZE` 設定および拡張サイズの大きい方によって拡張されます。`AUTOEXTEND_SIZE` オプションの詳細は、[セクション 15.6.3.9 「テーブルスペースの AUTOEXTEND_SIZE 構成」](#) を参照してください。

undo テーブルスペースの追加

長時間実行トランザクション中に undo ログが大きくなる可能性があるため、追加の undo テーブルスペースを作成すると、個々の undo テーブルスペースが大きくなりすぎるのを防ぐことができます。MySQL 8.0.14 では、`CREATE UNDO TABLESPACE` 構文を使用して、実行時に追加の undo テーブルスペースを作成できます。

```
CREATE UNDO TABLESPACE tablespace_name ADD DATAFILE 'file_name.ibu';
```

undo テーブルスペースのファイル名には、`.ibu` 拡張子が必要です。undo テーブルスペースファイル名の定義時に相対パスを指定することはできません。完全修飾パスは許可されますが、パスは InnoDB で認識されている必要があります。

ます。既知のパスは、`innodb_directories` 変数で定義されたパスです。データの移動またはクローニング時に潜在的なファイル名の競合を回避するために、一意の undo テーブルスペースファイル名をお勧めします。

注記

レプリケーション環境では、ソースと各レプリカに独自の undo テーブルスペースファイルディレクトリが必要です。undo テーブルスペースファイルの作成を共通ディレクトリにレプリケートすると、ファイル名の競合が発生します。

起動時に、`innodb_directories` 変数で定義されたディレクトリで undo テーブルスペースファイルがスキャンされます。(スキャンはサブディレクトリも走査します。) `innodb_data_home_dir`、`innodb_undo_directory` および `datadir` 変数で定義されたディレクトリは、`innodb_directories` 変数が明示的に定義されているかどうかに関係なく、`innodb_directories` 値に自動的に追加されます。したがって、undo テーブルスペースは、これらの変数のいずれかで定義されたパスに存在できます。

undo テーブルスペースのファイル名にパスが含まれていない場合、undo テーブルスペースは `innodb_undo_directory` 変数で定義されたディレクトリに作成されます。この変数が定義されていない場合、undo テーブルスペースはデータディレクトリに作成されます。

注記

InnoDB リカバリプロセスでは、undo テーブルスペースファイルが既知のディレクトリに存在する必要があります。コミットされていないトランザクションおよびデータディクショナリの変更をロールバックできるようにするには、undo テーブルスペースファイルを検出してオープンしてから redo リカバリを実行し、他のデータファイルをオープンする必要があります。リカバリを使用する前に undo テーブルスペースが見つからないため、データベースの不整合が発生する可能性があります。データディクショナリで認識されている undo テーブルスペースが見つからない場合は、起動時にエラーメッセージがレポートされます。既知のディレクトリ要件では、undo テーブルスペースの移植性もサポートされます。[undo テーブルスペースの移動](#)を参照してください。

データディレクトリからの相対パスに undo テーブルスペースを作成するには、`innodb_undo_directory` 変数を相対パスに設定し、undo テーブルスペースの作成時にのみファイル名を指定します。

undo テーブルスペースの名前とパスを表示するには、`INFORMATION_SCHEMA.FILES` をクエリーします：

```
SELECT TABLESPACE_NAME, FILE_NAME FROM INFORMATION_SCHEMA.FILES
WHERE FILE_TYPE LIKE 'UNDO LOG';
```

MySQL インスタンスでは、MySQL インスタンスの初期化時に作成される 2 つのデフォルト undo テーブルスペースを含む、最大 127 個の undo テーブルスペースがサポートされます。

注記

MySQL 8.0.14 より前は、`innodb_undo_tablespaces` 起動変数を構成することで、追加の undo テーブルスペースが作成されます。この変数は非推奨であり、MySQL 8.0.14 では構成できなくなりました。

MySQL 8.0.14 より前は、`innodb_undo_tablespaces` 設定を増やすと、指定した数の undo テーブルスペースが作成され、アクティブな undo テーブルスペースのリストに追加されます。`innodb_undo_tablespaces` 設定を小さくすると、アクティブな undo テーブルスペースのリストから undo テーブルスペースが削除されます。アクティブリストから削除された undo テーブルスペースは、既存のトランザクションで使用されなくなるまでアクティブなままです。`innodb_undo_tablespaces` 変数は、`SET` ステートメントを使用して実行時に構成するか、構成ファイルで定義できます。

MySQL 8.0.14 より前は、非アクティブ化された undo テーブルスペースは削除できません。undo テーブルスペースファイルは、低速な停止後に手動で削除できますが、非アクティブ化された undo テーブルスペースには、サーバーの停止時にオープントランザクションが存在していた場合、サーバーの再起動後のしばらくの間アクティブ undo ログが含まれる可能性があるため、お勧めしません。MySQL 8.0.14 では、`DROP UNDO TABLESPACE` 構文を使用して undo テーブルスペースを削除できます。[undo テーブルスペースの削除](#)を参照してください。

undo テーブルスペースの削除

MySQL 8.0.14 では、[CREATE UNDO TABLESPACE](#) 構文を使用して作成された undo テーブルスペースは、[DROP UNDO TABLESPACE](#) 構文を使用して実行時に削除できます。

undo テーブルスペースは、削除する前に空にする必要があります。undo テーブルスペースを空にするには、ロールバックセグメントを新しいトランザクションに割り当てるためにテーブルスペースが使用されなくなるように、まず [ALTER UNDO TABLESPACE](#) 構文を使用して undo テーブルスペースを非アクティブとしてマークする必要があります。

```
ALTER UNDO TABLESPACE tablespace_name SET INACTIVE;
```

undo テーブルスペースが非アクティブとしてマークされると、undo テーブルスペースのロールバックセグメントを現在使用しているトランザクションは、それらのトランザクションが完了する前に開始されたトランザクションと同様に終了できます。トランザクションが完了すると、バージョンシステムによって undo テーブルスペースのロールバックセグメントが解放され、undo テーブルスペースは初期サイズに切り捨てられます。(undo テーブルスペースを切り捨てる場合も同じプロセスが使用されます。 [undo テーブルスペースの切捨て](#) を参照してください。) undo テーブルスペースが空の場合は、削除できます。

```
DROP UNDO TABLESPACE tablespace_name;
```

注記

または、undo テーブルスペースを空の状態のままにし、必要に応じて [ALTER UNDO TABLESPACE *tablespace_name* SET ACTIVE](#) ステートメントを発行して後で再アクティブ化することもできます。

undo テーブルスペースの状態は、[INFORMATION_SCHEMA.INNODB_TABLESPACES](#) テーブルをクエリーすることで監視できます。

```
SELECT NAME, STATE FROM INFORMATION_SCHEMA.INNODB_TABLESPACES  
WHERE NAME LIKE 'tablespace_name';
```

[inactive](#) の状態は、undo テーブルスペースのロールバックセグメントが新しいトランザクションで使用されなくなったことを示します。[empty](#) の状態は、undo テーブルスペースが空であり、削除の準備ができていないか、[ALTER UNDO TABLESPACE *tablespace_name* SET ACTIVE](#) ステートメントを使用して再度アクティブになったことを示します。空でない undo テーブルスペースを削除しようとすると、エラーが返されます。

MySQL インスタンスの初期化時に作成されたデフォルトの undo テーブルスペース ([innodb_undo_001](#) および [innodb_undo_002](#)) は削除できません。ただし、[ALTER UNDO TABLESPACE *tablespace_name* SET INACTIVE](#) ステートメントを使用して非アクティブにすることはできます。デフォルトの undo テーブルスペースを非アクティブにするには、その前に undo テーブルスペースが必要です。undo テーブルスペースの自動切捨てをサポートするには、常に 2 つ以上のアクティブ undo テーブルスペースが必要です。

undo テーブルスペースの移動

[CREATE UNDO TABLESPACE](#) 構文で作成された undo テーブルスペースは、サーバーがオフラインのときに既知のディレクトリに移動できます。既知のディレクトリは、[innodb_directories](#) 変数で定義されたディレクトリです。[innodb_data_home_dir](#)、[innodb_undo_directory](#) および [datadir](#) によって定義されたディレクトリは、[innodb_directories](#) 変数が明示的に定義されているかどうかに関係なく、[innodb_directories](#) 値に自動的に追加されます。これらのディレクトリとそのサブディレクトリでは、undo テーブルスペースファイルが起動時にスキャンされます。これらのディレクトリに移動された undo テーブルスペースファイルは、起動時に検出され、移動された undo テーブルスペースとみなされます。

MySQL インスタンスの初期化時に作成されるデフォルトの undo テーブルスペース ([innodb_undo_001](#) および [innodb_undo_002](#)) は、[innodb_undo_directory](#) 変数で定義されたディレクトリに常に存在する必要があります。[innodb_undo_directory](#) 変数が定義されていない場合、デフォルトの undo テーブルスペースはデータディレクトリに存在します。サーバーがオフラインのときにデフォルトの undo テーブルスペースを移動する場合は、新しいディレクトリに構成された [innodb_undo_directory](#) 変数を使用してサーバーを起動する必要があります。

undo ログの I/O パターンにより、undo テーブルスペースは [SSD](#) 記憶域の適切な候補になります。

ロールバックセグメント数の構成

`innodb_rollback_segments` 変数は、各 undo テーブルスペースおよびグローバル一時テーブルスペースに割り当てられる `rollback segments` の数を定義します。 `innodb_rollback_segments` 変数は、起動時またはサーバーの実行中に構成できます。

`innodb_rollback_segments` のデフォルト設定は 128 で、これは最大値でもあります。ロールバックセグメントがサポートするトランザクションの数の詳細は、[セクション15.6.6「undo ログ」](#) を参照してください。

undo テーブルスペースの切捨て

undo テーブルスペースを切り捨てる方法は 2 つあり、undo テーブルスペースのサイズを個別に、または組み合わせて管理できます。1 つの方法は自動化され、構成変数を使用して有効化されます。その他の方法は手動で、SQL ステートメントを使用して実行されます。

自動化された方法では undo テーブルスペースサイズを監視する必要はなく、一度有効にすると、手動操作なしで undo テーブルスペースの非アクティブ化、切捨ておよび再アクティブ化が実行されます。切捨てのために undo テーブルスペースをオフラインにするタイミングを制御する場合は、手動切捨て方法をお勧めします。たとえば、ワークロードのピーク時に undo テーブルスペースを切り捨てないようにすることが必要な場合があります。

自動切捨て

undo テーブルスペースの自動切捨てには、少なくとも 2 つのアクティブな undo テーブルスペースが必要です。これにより、一方の undo テーブルスペースはアクティブなままになり、もう一方の undo テーブルスペースは切り捨てられます。デフォルトでは、MySQL インスタンスの初期化時に 2 つの undo テーブルスペースが作成されます。

undo テーブルスペースを自動的に切り捨てるには、`innodb_undo_log_truncate` 変数を有効にします。例:

```
mysql> SET GLOBAL innodb_undo_log_truncate=ON;
```

`innodb_undo_log_truncate` 変数が有効な場合、`innodb_max_undo_log_size` 変数で定義されたサイズ制限を超える undo テーブルスペースは切り捨てられる可能性があります。 `innodb_max_undo_log_size` 変数は動的で、デフォルト値は 1073741824 バイト (1024 MiB) です。

```
mysql> SELECT @@innodb_max_undo_log_size;
+-----+
| @@innodb_max_undo_log_size |
+-----+
|          1073741824 |
+-----+
```

`innodb_undo_log_truncate` 変数を有効にすると、次のようになります:

1. `innodb_max_undo_log_size` 設定を超えるデフォルトおよびユーザー定義の undo テーブルスペースは、切捨て対象としてマークされます。切捨て用の undo テーブルスペースの選択は循環方式で実行され、毎回同じ undo テーブルスペースが切り捨てられないようにします。
2. 選択した undo テーブルスペースに存在するロールバックセグメントは、新しいトランザクションに割り当てられないように非アクティブになります。ロールバックセグメントを現在使用している既存のトランザクションは終了できます。
3. `purge` システムは、使用されなくなった undo ログを解放することで、ロールバックセグメントを空にします。
4. undo テーブルスペースのすべてのロールバックセグメントが解放されると、切捨て操作が実行され、undo テーブルスペースが初期サイズに切り捨てられます。

切捨て操作後の undo テーブルスペースのサイズは、操作が完了した直後に使用されるため、初期サイズより大きくなる場合があります。

`innodb_undo_directory` 変数は、デフォルトの undo テーブルスペースファイルの場所を定義します。

`innodb_undo_directory` 変数が定義されていない場合、デフォルトの undo テーブルスペースはデータディレクトリに存在します。 `CREATE UNDO TABLESPACE` 構文を使用して作成されたユーザー定義の undo テーブルスペースを含むすべての undo テーブルスペースファイルの場所は、`INFORMATION_SCHEMA.FILES` テーブルをクエリーすることで確認できます:

```
SELECT TABLESPACE_NAME, FILE_NAME FROM INFORMATION_SCHEMA.FILES WHERE FILE_TYPE LIKE 'UNDO LOG';
```

5. ロールバックセグメントは、新しいトランザクションに割り当てることができるように再アクティブ化されます。

手動切捨て

undo テーブルスペースを手動で切り捨てるには、少なくとも 3 つのアクティブな undo テーブルスペースが必要です。自動切捨てを有効にするには、常に 2 つのアクティブ undo テーブルスペースが必要です。undo テーブルスペースを手動でオフラインにすることを許可しながら、少なくとも 3 つの undo テーブルスペースがこの要件を満たしています。

undo テーブルスペースの切捨てを手動で開始するには、次のステートメントを発行して undo テーブルスペースを非アクティブ化します:

```
ALTER UNDO TABLESPACE tablespace_name SET INACTIVE;
```

undo テーブルスペースが非アクティブとしてマークされると、undo テーブルスペースのロールバックセグメントを現在使用しているトランザクションは、それらのトランザクションが完了する前に開始されたトランザクションと同様に終了できます。トランザクションが完了すると、ページシステムによって undo テーブルスペースのロールバックセグメントが解放され、undo テーブルスペースが初期サイズに切り捨てられ、undo テーブルスペースの状態が `inactive` から `empty` に変更されます。

注記

`ALTER UNDO TABLESPACE tablespace_name SET INACTIVE` ステートメントによって undo テーブルスペースが非アクティブ化されると、パージスレッドは次の機会でその undo テーブルスペースを検索します。undo テーブルスペースが検出され、切捨てのマークが付けられると、パージスレッドは頻度を上げて戻され、undo テーブルスペースがすぐに空になり、切り捨てられます。

undo テーブルスペースの状態を確認するには、`INFORMATION_SCHEMA.INNODB_TABLESPACES` テーブルをクエリーします。

```
SELECT NAME, STATE FROM INFORMATION_SCHEMA.INNODB_TABLESPACES
WHERE NAME LIKE 'tablespace_name';
```

undo テーブルスペースが `empty` 状態になると、次のステートメントを発行して undo テーブルスペースを再アクティブ化できます:

```
ALTER UNDO TABLESPACE tablespace_name SET ACTIVE;
```

`empty` 状態の undo テーブルスペースも削除できます。undo テーブルスペースの削除を参照してください。

undo テーブルスペースの自動切捨ての回避

パージスレッドは、undo テーブルスペースを空にしたり切り捨てたりします。デフォルトでは、パージスレッドは undo テーブルスペースを検索し、パージが起動されるたびに 128 回切り捨てます。パージスレッドが切り捨てる undo テーブルスペースを検索する頻度は、`innodb_purge_rseg_truncate_frequency` 変数 (デフォルト設定は 128) によって制御されます。

```
mysql> SELECT @@innodb_purge_rseg_truncate_frequency;
+-----+
| @@innodb_purge_rseg_truncate_frequency |
+-----+
|                128 |
+-----+
```

この頻度を上げるには、`innodb_purge_rseg_truncate_frequency` 設定を減らします。たとえば、パージスレッドで undo テーブルスペースを 32 回起動するたびに検索するには、`innodb_purge_rseg_truncate_frequency` を 32 に設定します。

```
mysql> SET GLOBAL innodb_purge_rseg_truncate_frequency=32;
```


undo テーブルスペースファイルの切捨てによるパフォーマンスへの影響

undo テーブルスペースが切り捨てられると、undo テーブルスペースのロールバックセグメントは非アクティブ化されます。他の undo テーブルスペースのアクティブロールバックセグメントは、システム全体の負荷を前提としているため、パフォーマンスがわずかに低下する可能性があります。パフォーマンスの低下の量は、いくつかの要因によって異なります:

- undo テーブルスペースの数
- undo ログの数
- undo テーブルスペースサイズ
- I/O 疑わしいシステムの数
- 既存の長時間実行トランザクション
- システムロード

この潜在的なパフォーマンスの問題を回避する最も簡単な方法は、undo テーブルスペースの数を増やすことです。

また、MySQL 8.0.21 より前は、undo テーブルスペースの切捨て操作中に 2 つのフラッシュ操作が実行されます。最初のフラッシュ操作では、バッファプールから古い undo テーブルスペースページが削除されます。2 回目のフラッシュ操作では、新しい undo テーブルスペースの初期ページがディスクに書き込まれます。ビジー状態のシステムでは、削除するページが多数ある場合、特に最初のフラッシュ操作がシステムのパフォーマンスに一時的に影響を与える可能性があります。MySQL 8.0.21 の時点では、両方のフラッシュ操作が削除されます。古い undo テーブルスペースページは、最近最も使用されていない状態になるか、次のフルチェックポイントで解放されるため、パッシブに解放されます。新しい undo テーブルスペースページの初期ページは、切捨て操作中にディスクにフラッシュされるのではなく、redo ログに記録されます。

undo テーブルスペースの切捨ての監視

MySQL 8.0.16 では、undo ログの切捨てに関連付けられたバックグラウンドアクティビティを監視するために、[undo](#) および [purge](#) の疑わしいシステムカウンタが提供されています。カウンタ名と説明については、[INFORMATION_SCHEMA.INNODB_METRICS](#) テーブルをクエリーします。

```
SELECT NAME, SUBSYSTEM, COMMENT FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME LIKE '%truncate%';
```

カウンタの有効化およびカウンタデータのクエリーの詳細は、[セクション 15.15.6 「InnoDB INFORMATION_SCHEMA メトリックテーブル」](#) を参照してください。

undo テーブルスペースの切捨て制限

MySQL 8.0.21 の時点では、チェックポイント間の同じ undo テーブルスペースに対する切捨て操作の数は 64 に制限されています。この制限により、ビジー状態のシステムで `innodb_max_undo_log_size` の設定が低すぎる場合などに発生する可能性のある undo テーブルスペースの切捨て操作の数が多すぎることが原因で発生する潜在的な問題が回避されます。この制限を超えると、undo テーブルスペースは非アクティブにできますが、次のチェックポイントまで切り捨てられません。MySQL 8.0.22 では、制限は 64 から 50,000 に引き上げられました。

undo テーブルスペースの切捨てリカバリ

undo テーブルスペースの切捨て操作では、一時 `undo_space_number_trunc.log` ファイルがサーバーログディレクトリに作成されます。そのログディレクトリは、`innodb_log_group_home_dir` によって定義されます。切捨て操作中にシステム障害が発生した場合、一時ログファイルを使用すると、起動プロセスで切り捨てられていた undo テーブルスペースを識別し、操作を続行できます。

undo テーブルスペースのステータス変数

次のステータス変数を使用すると、undo テーブルスペースの合計数、暗黙的 (InnoDB 作成) undo テーブルスペース、明示的 (ユーザー作成) undo テーブルスペースおよびアクティブ undo テーブルスペースの数を追跡できます:

```
mysql> SHOW STATUS LIKE 'InnoDB_undo_tablespace%';
```

Variable_name	Value
InnoDB_undo_tablespaces_total	2
InnoDB_undo_tablespaces_implicit	2
InnoDB_undo_tablespaces_explicit	0
InnoDB_undo_tablespaces_active	2

ステータス変数の説明については、[セクション5.1.10「サーバーステータス変数」](#)を参照してください。

15.6.3.5 一時テーブルスペース

InnoDB では、セッション一時テーブルスペースおよびグローバル一時テーブルスペースが使用されます。

セッション一時テーブルスペース

セッション一時テーブルスペースには、InnoDB がディスク上の内部一時テーブルの記憶域エンジンとして構成されている場合に最適化によって作成されるユーザー作成一時テーブルおよび内部一時テーブルが格納されます。MySQL 8.0.16 以降、ディスク上の内部一時テーブルに使用されるストレージエンジンは常に InnoDB です。(以前は、ストレージエンジンは `internal_tmp_disk_storage_engine` の値によって決定されていました。)

セッション一時テーブルスペースは、ディスク上の一時テーブルを作成する最初のリクエストで、一時テーブルスペースのプールからセッションに割り当てられます。セッションには最大 2 つのテーブルスペースが割り当てられます。1 つはユーザーが作成した一時テーブル用で、もう 1 つは最適化によって作成された内部一時テーブル用です。セッションに割り当てられた一時テーブルスペースは、セッションによって作成されたディスク上のすべての一時テーブルに使用されます。セッションが切断されると、その一時テーブルスペースは切り捨てられ、プールに解放されます。サーバーの起動時に、10 個の一時テーブルスペースのプールが作成されます。プールのサイズは縮小されず、必要に応じてテーブルスペースがプールに自動的に追加されます。一時テーブルスペースのプールは、通常の停止時または中断された初期化時に削除されます。セッション一時テーブルスペースファイルは、作成時にサイズが 5 ページで、`.ibt` ファイル名拡張子が付いています。

400 万の領域 ID の範囲は、セッション一時テーブルスペース用に予約されています。セッション一時テーブルスペースのプールはサーバーが起動するたびに再作成されるため、セッション一時テーブルスペースの領域 ID はサーバーの停止時に永続化されず、再利用できます。

`innodb_temp_tablespaces_dir` 変数は、セッション一時テーブルスペースが作成される場所を定義します。デフォルトの場所は、データディレクトリ内の `#innodb_temp` ディレクトリです。一時テーブルスペースのプールを作成できない場合、起動は拒否されます。

```
shell> cd BASEDIR/data/#innodb_temp
shell> ls
temp_10.ibt temp_2.ibt temp_4.ibt temp_6.ibt temp_8.ibt
temp_1.ibt temp_3.ibt temp_5.ibt temp_7.ibt temp_9.ibt
```

ステートメントベースのレプリケーション (SBR) モードでは、レプリカに作成された一時テーブルは、MySQL サーバーの停止時にのみ切り捨てられる単一セッション一時テーブルスペースに存在します。

`INNODB_SESSION_TEMP_TABLESPACES` テーブルは、セッション一時テーブルスペースに関するメタデータを提供します。

`INFORMATION_SCHEMA.INNODB_TEMP_TABLE_INFO` テーブルは、InnoDB インスタンスでアクティブなユーザー作成一時テーブルに関するメタデータを提供します。

グローバル一時テーブルスペース

グローバル一時テーブルスペース (`ibtmp1`) には、ユーザーが作成した一時テーブルに対する変更のロールバックセグメントが格納されます。

`innodb_temp_data_file_path` 変数は、グローバル一時テーブルスペースデータファイルの相対パス、名前、サイズおよび属性を定義します。 `innodb_temp_data_file_path` に値が指定されていない場合、デフォルトの動作では、`ibtmp1` という名前の単一の自動拡張データファイルが `innodb_data_home_dir` ディレクトリに作成されます。初期ファイルサイズは 12MB を少し超えています。

グローバル一時テーブルスペースは、通常の停止時または中断された初期化時に削除され、サーバーが起動するたびに再作成されます。グローバル一時テーブルスペースは、作成時に動的に生成された領域 ID を受け取ります。グローバル一時テーブルスペースを作成できない場合、起動は拒否されます。サーバーが予期せず停止した場合、グローバル一時テーブルスペースは削除されません。この場合、データベース管理者はグローバル一時テーブルスペースを手動で削除するか、MySQL サーバーを再起動できます。MySQL サーバーを再起動すると、グローバル一時テーブルスペースが自動的に削除され、再作成されます。

グローバル一時テーブルスペースは RAW デバイ스에 配置できません。

`INFORMATION_SCHEMA.FILES` は、グローバル一時テーブルスペースに関するメタデータを提供します。次のようなクエリを発行して、グローバル一時テーブルスペースメタデータを表示します:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.FILES WHERE TABLESPACE_NAME='innodb_temporary'G
```

デフォルトでは、グローバル一時テーブルスペースデータファイルは自動拡張され、必要に応じてサイズが大きくなります。

グローバル一時テーブルスペースデータファイルが自動拡張されているかどうかを確認するには、`innodb_temp_data_file_path` 設定を確認します:

```
mysql> SELECT @@innodb_temp_data_file_path;
+-----+
| @@innodb_temp_data_file_path |
+-----+
| ibtmp1:12M:autoextend      |
+-----+
```

グローバル一時テーブルスペースデータファイルのサイズを確認するには、次のようなクエリを使用して `INFORMATION_SCHEMA.FILES` テーブルをクエリーします:

```
mysql> SELECT FILE_NAME, TABLESPACE_NAME, ENGINE, INITIAL_SIZE, TOTAL_EXTENTS*EXTENT_SIZE
AS TotalSizeBytes, DATA_FREE, MAXIMUM_SIZE FROM INFORMATION_SCHEMA.FILES
WHERE TABLESPACE_NAME = 'innodb_temporary'G
***** 1. row *****
FILE_NAME: ./ibtmp1
TABLESPACE_NAME: innodb_temporary
ENGINE: InnoDB
INITIAL_SIZE: 12582912
TotalSizeBytes: 12582912
DATA_FREE: 6291456
MAXIMUM_SIZE: NULL
```

`TotalSizeBytes` には、グローバル一時テーブルスペースデータファイルの現在のサイズが表示されます。その他のフィールド値の詳細は、[セクション26.15「INFORMATION_SCHEMA FILES テーブル」](#)を参照してください。

または、オペレーティングシステムのグローバル一時テーブルスペースのデータファイルサイズを確認します。グローバル一時テーブルスペースデータファイルは、`innodb_temp_data_file_path` 変数で定義されたディレクトリにあります。

グローバル一時テーブルスペースデータファイルが占有しているディスク領域を再利用するには、MySQL サーバーを再起動します。サーバーを再起動すると、`innodb_temp_data_file_path` で定義された属性に従ってグローバル一時テーブルスペースデータファイルが削除され、再作成されます。

グローバル一時テーブルスペースデータファイルのサイズを制限するには、最大ファイルサイズを指定するように `innodb_temp_data_file_path` を構成します。例:

```
[mysqld]
innodb_temp_data_file_path=ibtmp1:12M:autoextend:max:500M
```

`innodb_temp_data_file_path` を構成するには、サーバーの再起動が必要です。

15.6.3.6 サーバーがオフラインのときのテーブルスペースファイルの移動

起動時にテーブルスペースファイルのスキャンするディレクトリを定義する `innodb_directories` オプションでは、サーバーがオフラインのときにテーブルスペースファイルを新しい場所に移動またはリストアできます。起動時に

は、検出されたテーブルスペースファイルがデータディクショナリで参照されるファイルのかわりに使用され、再配置されたファイルを参照するようにデータディクショナリが更新されます。スキャンによって重複するテーブルスペースファイルが検出された場合、起動は失敗し、同じテーブルスペース ID に対して複数のファイルが見つかったことを示すエラーが表示されます。

`innodb_data_home_dir`、`innodb_undo_directory` および `datadir` の構成オプションで定義されたディレクトリは、`innodb_directories` 引数値に自動的に追加されます。これらのディレクトリは、`innodb_directories` オプションが明示的に指定されているかどうかに関係なく、起動時にスキャンされます。これらのディレクトリを暗黙的に追加すると、`innodb_directories` 設定を構成せずに、システムテーブルスペースファイル、データディレクトリまたは undo テーブルスペースファイルを移動できます。ただし、ディレクトリが変更された場合は設定を更新する必要があります。たとえば、データディレクトリを再配置した後、サーバーを再起動する前に `--datadir` 設定を更新する必要があります。

`innodb_directories` オプションは、起動コマンドまたは MySQL オプションファイルで指定できます。一部のコマンドインタプリタではセミコロン (;) は特殊文字として解釈されるため、引数値の前後に引用符が使用されます。(たとえば UNIX シェルでは、これはコマンド終端記号として扱われます。)

起動コマンド:

```
mysqld --innodb-directories="directory_path_1;directory_path_2"
```

MySQL オプションファイル:

```
[mysqld]
innodb_directories="directory_path_1;directory_path_2"
```

次の手順は、個々の `file-per-table` および `general tablespace` ファイル、`system tablespace` ファイル、`undo tablespace` ファイルまたはデータディレクトリの移動に適用できます。ファイルまたはディレクトリを移動する前に、次の使用上のノートを確認してください。

1. サーバーを停止します。
2. テーブルスペースファイルまたはディレクトリを移動します。
3. 新しいディレクトリを InnoDB で認識できるようにします。
 - 個々の `file-per-table` または `general tablespace` ファイルを移動する場合は、`innodb_directories` 値に不明なディレクトリを追加します。
 - `innodb_data_home_dir`、`innodb_undo_directory` および `datadir` の構成オプションで定義されたディレクトリは、`innodb_directories` 引数値に自動的に追加されるため、これらを指定する必要はありません。
 - `file-per-table` テーブルスペースファイルは、スキーマと同じ名前のディレクトリにのみ移動できます。たとえば、`actor` テーブルが `sakila` スキーマに属している場合、`actor.ibd` データファイルは `sakila` というディレクトリにのみ移動できます。
 - 一般テーブルスペースファイルは、データディレクトリまたはデータディレクトリのサブディレクトリに移動できません。
 - システムテーブルスペースファイル、undo テーブルスペースまたはデータディレクトリを移動する場合は、必要に応じて `innodb_data_home_dir`、`innodb_undo_directory` および `datadir` の設定を更新します。
4. サーバーを再起動します。

使用上の注意

- ワイルドカード式は、`innodb_directories` 引数値では使用できません。
- `innodb_directories` スキャンは、指定されたディレクトリのサブディレクトリも走査します。重複するディレクトリおよびサブディレクトリは、スキャンされるディレクトリのリストから破棄されます。
- `innodb_directories` オプションでは、InnoDB テーブルスペースファイルの移動のみがサポートされます。InnoDB 以外のストレージエンジンに属するファイルの移動はサポートされていません。この制限は、データディレクトリ全体を移動する場合にも適用されます。

- `innodb_directories` オプションでは、スキャンしたディレクトリにファイルを移動する際のテーブルスペースファイルの名前変更がサポートされています。また、サポートされている他のオペレーティングシステムへのテーブルスペースファイルの移動もサポートしています。
- テーブルスペースファイルを別のオペレーティングシステムに移動する場合は、宛先システムで禁止されている文字または特殊文字がテーブルスペースファイル名に含まれていないことを確認してください。
- データディレクトリを Windows オペレーティングシステムから Linux オペレーティングシステムに移動する場合は、バイナリログインデックスファイルのバイナリログファイルパスを変更して、スラッシュではなくバックスラッシュを使用します。デフォルトでは、バイナリログインデックスファイルのベース名はバイナリログファイルと同じで、拡張子は `.index` です。バイナリログインデックスファイルの場所は、`--log-bin` によって定義されます。デフォルトの場所はデータディレクトリです。
- テーブルスペースファイルを別のオペレーティングシステムに移動すると、クロスプラットフォームのレプリケーションが導入されます。データベース管理者は、プラットフォーム固有のディレクトリを含む DDL ステートメントを適切にレプリケーションする必要があります。ディレクトリの指定を許可するステートメントには、`CREATE TABLE ... DATA DIRECTORY` および `CREATE TABLESPACE ... ADD DATAFILE` があります。
- 絶対パスを使用して、またはデータディレクトリ外の場所に作成された file-per-table および general テーブルスペースのディレクトリを `innodb_directories` 設定に追加します。そうしないと、InnoDB はリカバリ中にファイルを検出できません。関連情報については、[クラッシュリカバリ中のテーブルスペースの検出](#)を参照してください。

テーブルスペースファイルの場所を表示するには、`INFORMATION_SCHEMA.FILES` テーブルをクエリーします:

```
mysql> SELECT TABLESPACE_NAME, FILE_NAME FROM INFORMATION_SCHEMA.FILES \G
```

15.6.3.7 テーブルスペースパス検証の無効化

起動時に、InnoDB は `innodb_directories` 変数で定義されたディレクトリでテーブルスペースファイルをスキャンします。検出されたテーブルスペースファイルのパスは、データディクショナリに記録されているパスに対して検証されます。パスが一致しない場合は、データディクショナリ内のパスが更新されます。

MySQL 8.0.21 で導入された `innodb_validate_tablespace_paths` 変数を使用すると、テーブルスペースパス検証を無効にできます。この機能は、テーブルスペースファイルを移動しない環境を対象としています。パス検証を無効にすると、多数のテーブルスペースファイルがあるシステムでの起動時間が短縮されます。`log_error_verbosity` が 3 に設定されている場合、テーブルスペースパスの検証が無効になると、起動時に次のメッセージが出力されます:

```
[InnoDB] Skipping InnoDB tablespace path validation.
Manually moved tablespace files will not be detected!
```

警告

テーブルスペースファイルの移動後にテーブルスペースパス検証を無効にしてサーバーを起動すると、動作が未定義になる可能性があります。

15.6.3.8 Linux でのテーブルスペースの領域割当ての最適化

MySQL 8.0.22 では、InnoDB が Linux の file-per-table および general テーブルスペースに領域を割り当てる方法を最適化できます。デフォルトでは、追加の領域が必要な場合、InnoDB はテーブルスペースにページを割り当て、それらのページに NULL を物理的に書き込みます。新しいページが頻繁に割り当てられる場合、この動作はパフォーマンスに影響を与える可能性があります。MySQL 8.0.22 では、Linux システムで `innodb_extend_and_initialize` を無効にして、新しく割り当てられたテーブルスペースページに NULL が物理的に書き込まないようにできます。`innodb_extend_and_initialize` を無効にすると、`posix_fallocate()` コールを使用してテーブルスペースファイルに領域が割り当てられ、物理的に NULL を書き込まずに領域が予約されます。

`posix_fallocate()` 操作はアトミックではないため、テーブルスペースファイルへの領域の割当てとファイルメタデータの更新の間に障害が発生する可能性があります。このような障害が発生すると、新しく割り当てられたページは初期化されていない状態のままになり、InnoDB がこれらのページにアクセスしようとしたときに障害が発生する可能性があります。このシナリオを回避するために、InnoDB は新しいテーブルスペースページを割り当てる前に redo ログレコードを書き込みます。ページ割当て操作が中断されると、リカバリ中に redo ログレコードから操作がリプレイされます。(redo ログレコードからリプレイされたページ割当て操作は、新しく割り当てられたページに NULL を物理的に書き込みます。) redo ログレコードは、`innodb_extend_and_initialize` の設定に関係なく、ページを割り当てる前に書き込まれます。

Linux 以外のシステムおよび Windows では、InnoDB はテーブルスペースに新しいページを割り当て、それらのページに NULL を物理的に書き込みます (デフォルトの動作)。これらのシステムで `innodb_extend_and_initialize` を無効にしようとすると、次のエラーが返されます:

`Changing innodb_extend_and_initialize` はこのプラットフォームではサポートされていません。default. へのフォールバック

MySQL 8.0.23 で導入された `AUTOEXTEND_SIZE` オプションは、`posix_fallocate()` コールによって割り当てられる領域の量を定義します。領域を大量に割り当てると、断片化の回避に役立ち、大量のデータの収集が容易になります。詳細は、[セクション15.6.3.9「テーブルスペースの AUTOEXTEND_SIZE 構成」](#)を参照してください。

15.6.3.9 テーブルスペースの AUTOEXTEND_SIZE 構成

デフォルトでは、file-per-table または general テーブルスペースに追加の領域が必要な場合、テーブルスペースは次の規則に従って増分的に拡張されます:

- テーブルスペースのサイズがエクステントより小さい場合は、一度に 1 ページずつ拡張されます。
- テーブルスペースが 1 エクステントより大きい、サイズが 32 エクステントより小さい場合は、一度に 1 エクステントずつ拡張されます。
- テーブルスペースのサイズが 32 エクステントを超える場合は、一度に 4 エクステント拡張されます。

エクステントサイズの詳細は、[セクション15.11.2「ファイル領域管理」](#)を参照してください。

MySQL 8.0.23 では、file-per-table または general テーブルスペースを拡張する量は、`AUTOEXTEND_SIZE` オプションを指定することで構成できます。拡張サイズを大きく構成すると、断片化を回避し、大量のデータの収集を容易にすることができます。

file-per-table テーブルスペースの拡張子サイズを構成するには、`CREATE TABLE` または `ALTER TABLE` ステートメントで `AUTOEXTEND_SIZE` サイズを指定します:

```
CREATE TABLE t1 (c1 INT) AUTOEXTEND_SIZE = 4M;
```

```
ALTER TABLE t1 AUTOEXTEND_SIZE = 8M;
```

一般的なテーブルスペースの拡張サイズを構成するには、`CREATE TABLESPACE` ステートメントまたは `ALTER TABLESPACE` ステートメントで `AUTOEXTEND_SIZE` サイズを指定します:

```
CREATE TABLESPACE ts1 AUTOEXTEND_SIZE = 4M;
```

```
ALTER TABLESPACE ts1 AUTOEXTEND_SIZE = 8M;
```

注記

`AUTOEXTEND_SIZE` オプションは、UNDO テーブルスペースの作成時にも使用できますが、UNDO テーブルスペースの拡張動作は異なります。詳細は、[セクション15.6.3.4「undo テーブルスペース」](#)を参照してください。

`AUTOEXTEND_SIZE` 設定は 4M の倍数である必要があります。4M の倍数ではない `AUTOEXTEND_SIZE` 設定を指定すると、エラーが返されます。

`AUTOEXTEND_SIZE` のデフォルト設定は 0 で、テーブルスペースは前述のデフォルトの動作に従って拡張されます。

`AUTOEXTEND_SIZE` の最大設定は 64M です。

次のテーブルに示すように、`AUTOEXTEND_SIZE` の最小設定は InnoDB ページサイズによって異なります:

InnoDB ページサイズ	最小 AUTOEXTEND_SIZE
4K	4M

InnoDB ページサイズ	最小 AUTOEXTEND_SIZE
8K	4M
16K	4M
32K	8M
64K	16M

デフォルトの InnoDB ページサイズは 16K (16384 バイト) です。MySQL インスタンスの InnoDB ページサイズを確認するには、`innodb_page_size` 設定をクエリーします:

```
mysql> SELECT @@GLOBAL.innodb_page_size;
+-----+
| @@GLOBAL.innodb_page_size |
+-----+
| 16384 |
+-----+
```

テーブルスペースの `AUTOEXTEND_SIZE` 設定が変更されると、後で最初に拡張されるときに、テーブルスペースサイズが `AUTOEXTEND_SIZE` 設定の倍数に増加します。後続の拡張機能は、構成されたサイズです。

ゼロ以外の `AUTOEXTEND_SIZE` 設定で `file-per-table` または `general` テーブルスペースが作成されると、テーブルスペースは指定された `AUTOEXTEND_SIZE` サイズで初期化されます。

`ALTER TABLESPACE` を使用して `file-per-table` テーブルスペースの `AUTOEXTEND_SIZE` を構成することはできません。`ALTER TABLE` を使用する必要があります。

`file-per-table` テーブルスペースに作成されたテーブルの場合、`SHOW CREATE TABLE` で `AUTOEXTEND_SIZE` オプションが表示されるのは、ゼロ以外の値に構成されている場合のみです。

InnoDB テーブルスペースの `AUTOEXTEND_SIZE` を確認するには、`INFORMATION_SCHEMA.INNODB_TABLESPACES` テーブルをクエリーします。例:

```
mysql> SELECT NAME, AUTOEXTEND_SIZE FROM INFORMATION_SCHEMA.INNODB_TABLESPACES
        WHERE NAME LIKE 'test/t1';
+-----+-----+
| NAME | AUTOEXTEND_SIZE |
+-----+-----+
| test/t1 | 4194304 |
+-----+-----+

mysql> SELECT NAME, AUTOEXTEND_SIZE FROM INFORMATION_SCHEMA.INNODB_TABLESPACES
        WHERE NAME LIKE 'ts1';
+-----+-----+
| NAME | AUTOEXTEND_SIZE |
+-----+-----+
| ts1 | 4194304 |
+-----+-----+
```

注記

`AUTOEXTEND_SIZE` のデフォルト設定である 0 は、テーブルスペースが前述のデフォルトのテーブルスペース拡張動作に従って拡張されることを意味します。

15.6.4 二重書き込みバッファ

二重書き込みバッファは、InnoDB データファイル内の適切な位置にページを書き込む前に、バッファプールからフラッシュされたページを InnoDB が書き込む記憶域です。ページ書き込み中にオペレーティングシステム、ストレージサブシステムまたは予期しない `mysqld` プロセスが終了した場合、InnoDB はクラッシュリカバリ中に二重書き込みバッファからページの適切なコピーを見つけることができます。

データは 2 回書き込まれますが、二重書き込みバッファには I/O オーバーヘッドの 2 倍や I/O 操作の 2 倍は必要ありません。データは、オペレーティングシステムへの単一の `fsync()` コールを使用して、大きいシーケンシャルチャンクで二重書き込みバッファに書き込まれます (`innodb_flush_method` が `O_DIRECT_NO_FSYNC` に設定されている場合を除く)。

MySQL 8.0.20 より前は、二重書き込みバッファ記憶域は InnoDB システムテーブルスペースにありました。MySQL 8.0.20 では、二重書き込みバッファ記憶域は二重書き込みファイルにあります。

二重書き込みバッファ構成には、次の変数が用意されています:

- `innodb_doublewrite`

`innodb_doublewrite` 変数は、二重書き込みバッファを有効にするかどうかを制御します。ほとんどの場合、デフォルトで有効になっています。二重書き込みバッファを無効にするには、`innodb_doublewrite` を 0 に設定するか、`--skip-innodb-doublewrite` でサーバーを起動します。たとえば、ベンチマークの実行時などのように、データ整合性よりもパフォーマンスに関心がある場合は、二重書き込みバッファを無効にすることを検討してください。

二重書き込みバッファがアトミック書き込みをサポートする Fusion-io デバイス上にある場合、二重書き込みバッファは自動的に無効になり、代わりに Fusion-io アトミック書き込みを使用してデータファイル書き込みが実行されます。ただし、`innodb_doublewrite` 設定はグローバルであることに注意してください。二重書き込みバッファが無効になっている場合、Fusion-io ハードウェア上に存在しないデータファイルを含むすべてのデータファイルに対して無効になります。この機能は Fusion-io ハードウェアでのみサポートされ、Linux の Fusion-io NVMFS でのみ有効になります。この機能を最大限に活用するには、`O_DIRECT` の `innodb_flush_method` 設定をお勧めします。

- `innodb_doublewrite_dir`

`innodb_doublewrite_dir` 変数 (MySQL 8.0.20 で導入) は、InnoDB が二重書き込みファイルを作成するディレクトリを定義します。ディレクトリが指定されていない場合、二重書き込みファイルが `innodb_data_home_dir` ディレクトリに作成され、指定されていない場合はデータディレクトリにデフォルト設定されます。

スキーマ名との競合を避けるために、指定されたディレクトリ名の前にハッシュ記号 '#' が自動的に付加されます。ただし、ディレクトリ名に '.', '#', または '/' 接頭辞が明示的に指定されている場合、ディレクトリ名の前にハッシュ記号 '#' は付けられません。

二重書き込みディレクトリは、使用可能な最も高速なストレージメディアに配置することが理想的です。

- `innodb_doublewrite_files`

`innodb_doublewrite_files` 変数は、二重書き込みファイルの数を定義します。デフォルトでは、バッファプールインスタンスごとに 2 つの二重書き込みファイルが作成されます: フラッシュリスト二重書き込みファイルおよび LRU リスト二重書き込みファイル。

フラッシュリスト二重書き込みファイルは、バッファプールのフラッシュリストからフラッシュされたページ用です。フラッシュリストの二重書き込みファイルのデフォルトサイズは、InnoDB ページサイズ * 二重書き込みページバイトです。

LRU リストの二重書き込みファイルは、バッファプール LRU リストからフラッシュされたページ用です。また、単一ページフラッシュ用のスロットも含まれます。LRU リスト二重書き込みファイルのデフォルトサイズは、InnoDB ページサイズ * (二重書き込みページ + (512 / バッファプールインスタンスの数)) です。512 は、単一ページフラッシュ用に予約されたスロットの合計数です。

少なくとも 2 つの二重書き込みファイルがあります。二重書き込みファイルの最大数は、バッファプールインスタンスの 2 倍です。(バッファプールインスタンスの数は、`innodb_buffer_pool_instances` 変数によって制御されます。)

二重書き込みファイル名の形式は次のとおりです: `#ib_page_size_file_number.dblwr`。たとえば、次の二重書き込みファイルは、InnoDB ページサイズが 16KB で単一のバッファプールの MySQL インスタンスに対して作成されます:

```
#ib_16384_0.dblwr
#ib_16384_1.dblwr
```

`innodb_doublewrite_files` 変数は、高度なパフォーマンスチューニングを目的としています。デフォルト設定は、ほとんどのユーザーに適しています。

- `innodb_doublewrite_pages`

`innodb_doublewrite_pages` 変数 (MySQL 8.0.20 で導入) は、スレッド当たりの二重書き込みページの最大数を制御します。値が指定されていない場合、`innodb_doublewrite_pages` は `innodb_write_io_threads` 値に設定されます。この変数は高度なパフォーマンスチューニングを目的としています。デフォルト値は、ほとんどのユーザーに適しています。

- [innodb_doublewrite_batch_size](#)

[innodb_doublewrite_batch_size](#) 変数 (MySQL 8.0.20 で導入) は、バッチで書き込む二重書き込みページの数を制御します。この変数は高度なパフォーマンスチューニングを目的としています。デフォルト値は、ほとんどのユーザーに適しています。

MySQL 8.0.23 では、[InnoDB](#) は暗号化されたテーブルスペースに属する二重書き込みファイルページを自動的に暗号化します ([セクション15.13「InnoDB 保存データ暗号化」](#) を参照)。同様に、ページ圧縮テーブルスペースに属する二重書き込みファイルページも圧縮されます。その結果、二重書き込みファイルには、暗号化されていないページと圧縮されていないページ、暗号化されたページ、圧縮されたページ、暗号化と圧縮の両方を含む様々なページタイプを含めることができます。

15.6.5 redo ログ

redo ログは、不完全なトランザクションによって書き込まれたデータを修正するためにクラッシュリカバリ中に使用されるディスクベースのデータ構造です。通常の操作中、redo ログは、SQL ステートメントまたは低レベルの API コールによって発生したテーブルデータを変更するリクエストをエンコードします。予期しないシャットダウンの前にデータファイルの更新を終了しなかった変更は、初期化中、および接続が受け入れられる前に自動的にリプレイされます。クラッシュリカバリにおける redo ログの役割の詳細は、[セクション15.18.2「InnoDB のリカバリ」](#) を参照してください。

デフォルトでは、redo ログはディスク上で [ib_logfile0](#) および [ib_logfile1](#) という名前の 2 つのファイルによって物理的に表されます。MySQL は、redo ログファイルに循環して書き込みます。redo ログ内のデータは、影響を受けるレコードに関してエンコードされます。このデータはまとめて redo と呼ばれます。redo ログを介したデータの受渡しは、増加する [LSN](#) 値で表されます。

関連情報については、[redo ログファイル構成](#)、および [セクション8.5.4「InnoDB redo ログの最適化」](#) を参照してください。

redo ログの保存データ暗号化の詳細は、[redo ログの暗号化](#) を参照してください。

redo ログファイルの数またはサイズの変更

[redo log](#) ファイルの数またはサイズを変更するには、次のステップを実行します:

1. MySQL サーバーを停止し、エラーなしでシャットダウンされることを確認します。
2. [my.cnf](#) を編集して、ログファイルの構成を変更します。ログファイルのサイズを変更するには、[innodb_log_file_size](#) を構成します。ログファイルの数を多くするには、[innodb_log_files_in_group](#) を構成します。
3. MySQL サーバーを再起動します。

[InnoDB](#) は、[innodb_log_file_size](#) と redo ログファイルのサイズが異なることを検出すると、ログチェックポイントを書き込み、古いログファイルを閉じて削除し、リクエストされたサイズで新しいログファイルを作成し、新しいログファイルを開きます。

redo ログフラッシュのグループコミット

[InnoDB](#) は、他の [ACID](#) 準拠のデータベースエンジンと同様に、コミット前にトランザクションの [redo log](#) をフラッシュします。[InnoDB](#) では、[group commit](#) 機能を使用してこのような複数のフラッシュリクエストをグループ化し、コミットごとに 1 つのフラッシュを回避します。グループコミットでは、[InnoDB](#) はログファイルに単一の書き込みを発行して、ほぼ同時にコミットする複数のユーザートランザクションに対してコミットアクションを実行し、スループットを大幅に向上させます。

[COMMIT](#) やその他のトランザクション操作のパフォーマンスの詳細は、[セクション8.5.2「InnoDB トランザクション管理の最適化」](#) を参照してください。

redo ログのアーカイブ

redo ログレコードをコピーするバックアップユーティリティは、バックアップ操作の進行中に redo ログの生成に対応できない場合があり、その結果、それらのレコードが上書きされるために redo ログレコードが失われます。この

問題は、多くの場合、バックアップ操作中に重大な MySQL サーバークิจกรรมが発生し、redo ログファイルストレージメディアがバックアップストレージメディアより高速に動作する場合に発生します。MySQL 8.0.17 で導入された redo ログアーカイブ機能は、redo ログファイルに加えて redo ログレコードをアーカイブファイルに順次書き込むことで、この問題に対処します。バックアップユーティリティでは、必要に応じてアーカイブファイルから redo ログレコードをコピーできるため、データが失われる可能性を回避できます。

redo ログアーカイブがサーバで構成されている場合、MySQL Enterprise Edition で使用可能な MySQL Enterprise Backup は、MySQL サーバのバックアップ時に redo ログアーカイブ機能を使用します。

サーバで redo ログアーカイブを有効にするには、`innodb_redo_log_archive_dirs` システム変数の値を設定する必要があります。この値は、ラベル付き redo ログアーカイブディレクトリのセミコロン区切りリストとして指定されます。`label:directory` ペアはコロン (:) で区切られます。例:

```
mysql> SET GLOBAL innodb_redo_log_archive_dirs='label1:directory_path1[:label2:directory_path2:...];'
```

`label` は、アーカイブディレクトリの任意の識別子です。コロン (:) を除いて、任意の文字列を指定できますが、これは許可されません。空のラベルも使用できますが、この場合もコロン (:) が必要です。`directory_path` を指定する必要があります。redo ログアーカイブがアクティブ化されている場合は、redo ログアーカイブファイル用に選択されたディレクトリが存在する必要があります。存在しない場合は、エラーが返されます。パスにはコロン (:) を含めることができますが、セミコロン (;) は使用できません。

redo ログアーカイブをアクティブ化する前に、`innodb_redo_log_archive_dirs` 変数を構成する必要があります。デフォルト値は `NULL` で、redo ログアーカイブのアクティブ化は許可されません。

メモ

指定するアーカイブディレクトリは、次の要件を満たす必要があります。(要件は、redo ログアーカイブがアクティブ化されたときに適用されます。):

- ディレクトリが存在する必要があります。redo ログアーカイブプロセスでは、ディレクトリは作成されません。それ以外の場合は、次のエラーが返されます:

```
ERROR 3844 (HY000): redo ログアーカイブディレクトリ'directory_path1'が存在しないか、ディレクトリではありません
```

- ディレクトリはワールドアクセス可能にしないでください。これは、redo ログデータがシステム上の権限のないユーザーに公開されないようにするためです。それ以外の場合は、次のエラーが返されます:

```
ERROR 3846 (HY000): redo ログアーカイブディレクトリ'directory_path1'には、すべてのOSユーザーがアクセスできます
```

- ディレクトリには、`datadir`、`innodb_data_home_dir`、`innodb_directories`、`innodb_log_group_home_dir`、`innodb_temp_tablespaces_dir`、`innodb_tmpdir`、`innodb_undo_directory` または `secure_file_priv` で定義されたディレクトリや、それらのディレクトリの親ディレクトリまたはサブディレクトリを指定できません。それ以外の場合は、次のようなエラーが返されます:

```
ERROR 3845 (HY000): redo ログアーカイブディレクトリ'directory_path1'は、サーバディレクトリ'datadir'の下または上にあります - '/path/to/data_directory'
```

redo ログアーカイブをサポートするバックアップユーティリティがバックアップを開始すると、バックアップユーティリティは `innodb_redo_log_archive_start()` ユーザー定義関数を起動して redo ログアーカイブをアクティブ化します。

redo ログのアーカイブをサポートするバックアップユーティリティを使用していない場合は、次に示すように、redo ログのアーカイブを手動でアクティブ化することもできます:

```
mysql> SELECT innodb_redo_log_archive_start('label', 'subdir');
```

```
+-----+
| innodb_redo_log_archive_start('label') |
+-----+
| 0 |
+-----+
```

または:

```
mysql> DO innodb_redo_log_archive_start('label', 'subdir');
Query OK, 0 rows affected (0.09 sec)
```

注記

(`innodb_redo_log_archive_start()` を使用して) redo ログアーカイブをアクティブ化する MySQL セッションは、アーカイブ中はオープンのままである必要があります。同じセッションで (`innodb_redo_log_archive_stop()` を使用して) redo ログアーカイブを非アクティブ化する必要があります。redo ログアーカイブが明示的に非アクティブ化される前にセッションが終了した場合、サーバーは redo ログアーカイブを暗黙的に非アクティブ化し、redo ログアーカイブファイルを削除します。

ここで、`label` は `innodb_redo_log_archive_dirs` によって定義されたラベルです。`subdir` は、アーカイブファイルを保存するために `label` によって識別されるディレクトリのサブディレクトリを指定するためのオプションの引数です。単純なディレクトリ名である必要があります (スラッシュ (/)、バックスラッシュ (\)、またはコロン (:)) は使用できません。`subdir` は空または null にすることも、省略することもできます。

`innodb_redo_log_archive_start()` を起動して redo ログアーカイブをアクティブ化するか、`innodb_redo_log_archive_stop()` を使用して非アクティブ化できるのは、`INNODB_REDO_LOG_ARCHIVE` 権限を持つユーザーのみです。バックアップユーティリティを実行している MySQL ユーザーまたは redo ログアーカイブを手動でアクティブ化および非アクティブ化する MySQL ユーザーには、この権限が必要です。

redo ログアーカイブファイルのパスは `directory_identified_by_label/[subdir]archive.serverUUID.000001.log` です。ここで、`directory_identified_by_label` は `innodb_redo_log_archive_start()` の `label` 引数で識別されるアーカイブディレクトリです。`subdir` は、`innodb_redo_log_archive_start()` に使用されるオプションの引数です。

たとえば、redo ログアーカイブファイルのフルパスと名前は次のようになります:

```
/directory_path/subdirectory/archive.e71a47dc-61f8-11e9-a3cb-080027154b4d.000001.log
```

バックアップユーティリティは、InnoDB データファイルのコピーを終了した後、`innodb_redo_log_archive_stop()` ユーザー定義関数をコールして redo ログアーカイブを非アクティブ化します。

redo ログのアーカイブをサポートするバックアップユーティリティを使用していない場合は、次に示すように、redo ログのアーカイブを手動で非アクティブ化することもできます:

```
mysql> SELECT innodb_redo_log_archive_stop();
+-----+
| innodb_redo_log_archive_stop() |
+-----+
| 0 |
+-----+
```

または:

```
mysql> DO innodb_redo_log_archive_stop();
Query OK, 0 rows affected (0.01 sec)
```

停止機能が正常に完了すると、バックアップユーティリティはアーカイブファイルから redo ログデータの関連セクションを検索し、それをバックアップにコピーします。

バックアップユーティリティが redo ログデータのコピーを終了し、redo ログアーカイブファイルが不要になると、アーカイブファイルは削除されます。

アーカイブファイルの削除は、通常の場合ではバックアップユーティリティの役割を果たします。ただし、`innodb_redo_log_archive_stop()` がコールされる前に redo ログのアーカイブ操作が予期せず終了した場合、MySQL サーバーはファイルを削除します。

パフォーマンスに関する考慮事項

通常、redo ログアーカイブをアクティブ化すると、書込みアクティビティが追加されるため、パフォーマンスが若干低下します。

Unix および Unix に似たオペレーティングシステムでは、パフォーマンスへの影響は、通常、高い更新率が持続しないことを前提としています。Windows では、通常、パフォーマンスへの影響は少し大きくなります (同じことを前提としています)。

継続的に高い更新率があり、redo ログアーカイブファイルが redo ログファイルと同じ記憶域メディアにある場合、複合書き込みアクティビティが原因でパフォーマンスへの影響が大きくなる可能性があります。

継続的に高い更新率があり、redo ログアーカイブファイルが redo ログファイルより低速なストレージメディア上にある場合、パフォーマンスには任意の影響があります。

redo ログアーカイブファイルへの書き込みでは、redo ログアーカイブファイルの記憶域メディアが redo ログファイルの記憶域メディアよりもはるかに低速で動作し、redo ログアーカイブファイルへの書き込みを待機している永続 redo ログブロックの大きなバックログがある場合を除き、通常のトランザクションロギングは妨げられません。この場合、トランザクションロギング率は、redo ログアーカイブファイルが存在する低速の記憶域メディアで管理できるレベルに削減されます。

redo ロギングの無効化

MySQL 8.0.21 では、`ALTER INSTANCE DISABLE INNODB REDO_LOG` ステートメントを使用して redo ロギングを無効にできます。この機能は、新しい MySQL インスタンスにデータをロードするためのものです。redo ロギングを無効にすると、redo ログの書き込みおよび二重書き込みバッファリングが回避され、データのロードが高速化されます。

警告

この機能は、新しい MySQL インスタンスへのデータのロードのみを目的としています。本番システムで redo ロギングを無効にしないでください。redo ロギングが無効化されている間はサーバーを停止して再起動できますが、redo ロギングが無効化されている間に予期しないサーバーストップページが発生すると、データが失われ、インスタンスが破損する可能性があります。

redo ロギングが無効化されている間に予期しないサーバー停止ページの後にサーバーを再起動しようとすると、次のエラーで拒否されます:

```
[ERROR] [MY-013578] [InnoDB] Server was killed when InnoDB Redo logging was disabled. Data files could be corrupt. You can try to restart the database with innodb_force_recovery=6
```

この場合、新しい MySQL インスタンスを初期化し、データロードプロセスを再度開始します。

redo ロギングを有効化および無効化するには、`INNODB_REDO_LOG_ENABLE` 権限が必要です。

`Innodb_redo_log_enabled` ステータス変数を使用すると、redo ロギングステータスを監視できます。

redo ロギングが無効化されている間は、クローニング操作および redo ログアーカイブは許可されません。その逆も同様です。

`ALTER INSTANCE [ENABLE|DISABLE] INNODB REDO_LOG` 操作には排他的バックアップメタデータロックが必要で、これにより他の `ALTER INSTANCE` 操作が同時に実行されなくなります。その他の `ALTER INSTANCE` 操作は、ロックが解放されるまで待機してから実行する必要があります。

次の手順は、新しい MySQL インスタンスにデータをロードするときに redo ロギングを無効にする方法を示しています。

1. 新しい MySQL インスタンスで、redo ロギングを無効にするユーザーアカウントに `INNODB_REDO_LOG_ENABLE` 権限を付与します。

```
mysql> GRANT INNODB_REDO_LOG_ENABLE ON *.* to 'data_load_admin';
```

2. `data_load_admin` ユーザーとして、redo ロギングを無効にします:

```
mysql> ALTER INSTANCE DISABLE INNODB REDO_LOG;
```


3. `InnoDB_redo_log_enabled` ステータス変数をチェックして、redo ログが無効になっていることを確認します。

```
mysql> SHOW GLOBAL STATUS LIKE 'InnoDB_redo_log_enabled';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| InnoDB_redo_log_enabled | OFF |
+-----+-----+
```

4. データロード操作を実行します。
5. `data_load_admin` ユーザーとして、データロード操作の終了後に redo ログを有効にします:

```
mysql> ALTER INSTANCE ENABLE INNODB REDO_LOG;
```

6. `InnoDB_redo_log_enabled` ステータス変数をチェックして、redo ログが有効になっていることを確認します。

```
mysql> SHOW GLOBAL STATUS LIKE 'InnoDB_redo_log_enabled';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| InnoDB_redo_log_enabled | ON |
+-----+-----+
```

15.6.6 undo ログ

undo ログは、単一の読み取り/書き込みトランザクションに関連付けられた undo ログレコードの集合です。undo ログレコードには、トランザクションによる [clustered index](#) レコードへの最新の変更を元に戻す方法に関する情報が含まれます。別のトランザクションで読み取り一貫性操作の一部として元のデータを参照する必要がある場合、未変更のデータは undo ログレコードから取得されます。undo ログは、[rollback segments](#) 内に含まれる [undo log segments](#) 内に存在します。ロールバックセグメントは、[undo tablespaces](#) および [global temporary tablespace](#) にあります。

グローバル一時テーブルスペースに存在する undo ログは、ユーザー定義一時テーブルのデータを変更するトランザクションで使用されます。これらの undo ログは、クラッシュリカバリに必要なため、redo ログには記録されません。これらは、サーバーの実行中のロールバックにのみ使用されます。このタイプの undo ログは、redo ログिंग//O を回避することでパフォーマンスを向上させます。

undo ログの保存データ暗号化の詳細は、[undo ログの暗号化](#) を参照してください。

各 undo テーブルスペースとグローバル一時テーブルスペースは、それぞれ最大 128 個のロールバックセグメントをサポートします。`innodb_rollback_segments` 変数は、ロールバックセグメントの数を定義します。

ロールバックセグメントでサポートされるトランザクションの数は、ロールバックセグメント内の undo スロットの数および各トランザクションに必要な undo ログの数によって異なります。

ロールバックセグメントの undo スロットの数は、InnoDB のページサイズによって異なります。

InnoDB ページサイズ	ロールバックセグメント内の undo スロット数 (InnoDB ページサイズ / 16)
4096 (4KB)	256
8192 (8KB)	512
16384 (16KB)	1024
32768 (32KB)	2048
65536 (64KB)	4096

トランザクションには、次の操作タイプごとに 1 つずつ、最大 4 つの undo ログが割り当てられます:

1. ユーザー定義テーブルに対する `INSERT` 操作
2. ユーザー定義テーブルに対する `UPDATE` および `DELETE` 操作
3. ユーザー定義一時テーブルに対する `INSERT` 操作
4. ユーザー定義一時テーブルに対する `UPDATE` および `DELETE` 操作

undo ログは必要に応じて割り当てられます。たとえば、通常のテーブルおよび一時テーブルに対して **INSERT**、**UPDATE** および **DELETE** 操作を実行するトランザクションには、4 つの undo ログの完全な割当てが必要です。通常のテーブルに対して **INSERT** 操作のみを実行するトランザクションには、単一の undo ログが必要です。

通常のテーブルに対して操作を実行するトランザクションには、割り当てられた undo テーブルスペースロールバックセグメントから undo ログが割り当てられます。一時テーブルに対して操作を実行するトランザクションには、割り当てられたグローバル一時テーブルスペースロールバックセグメントから undo ログが割り当てられます。

トランザクションに割り当てられた undo ログは、その期間中トランザクションに関連付けられたままになります。たとえば、通常のテーブルに対する **INSERT** 操作のトランザクションに割り当てられた undo ログは、そのトランザクションによって実行される通常のテーブルに対するすべての **INSERT** 操作に使用されます。

前述の要因に基づき、次の式を使用して、InnoDB がサポートできる同時読取り/書き込みトランザクションの数を見積もることができます。

注記

InnoDB がサポートできる同時読取り/書き込みトランザクションの数に達する前に、トランザクションで同時トランザクション制限エラーが発生する場合があります。これは、トランザクションに割り当てられたロールバックセグメントで undo スロットが不足した場合に発生します。このような場合は、トランザクションを再実行してください。

トランザクションが一時テーブルに対して操作を実行する場合、InnoDB がサポートできる同時読取り/書き込みトランザクションの数は、グローバル一時テーブルスペース (デフォルトで 128) に割り当てられたロールバックセグメントの数によって制約されます。

- 各トランザクションが **INSERT**、**UPDATE** または **DELETE** のいずれかの操作を実行する場合、InnoDB がサポートできる同時読取り/書き込みトランザクションの数は次のとおりです:

$$(\text{innodb_page_size} / 16) * \text{innodb_rollback_segments} * \text{number of undo tablespaces}$$

- 各トランザクションが **INSERT** および **UPDATE** または **DELETE** 操作を実行する場合、InnoDB がサポートできる同時読取り/書き込みトランザクションの数は次のとおりです:

$$(\text{innodb_page_size} / 16 / 2) * \text{innodb_rollback_segments} * \text{number of undo tablespaces}$$

- 各トランザクションが一時テーブルに対して **INSERT** 操作を実行する場合、InnoDB がサポートできる同時読取り/書き込みトランザクションの数は次のとおりです:

$$(\text{innodb_page_size} / 16) * \text{innodb_rollback_segments}$$

- 各トランザクションが一時テーブルに対して **INSERT** および **UPDATE** または **DELETE** 操作を実行する場合、InnoDB がサポートできる同時読取り/書き込みトランザクションの数は次のとおりです:

$$(\text{innodb_page_size} / 16 / 2) * \text{innodb_rollback_segments}$$

15.7 InnoDB のロックおよびトランザクションモデル

大規模、ビジーまたは信頼性の高いデータベースアプリケーションを実装して、異なるデータベースシステムから大幅なコードを移植したり、MySQL のパフォーマンスをチューニングするには、InnoDB のロックおよび InnoDB トランザクションモデルを理解することが重要です。

このセクションでは、InnoDB のロックおよび理解しておく必要がある InnoDB トランザクションモデルに関連するいくつかのトピックについて説明します。

- セクション 15.7.1 「InnoDB ロック」では、InnoDB で使用されるロックタイプについて説明します。
- セクション 15.7.2 「InnoDB トランザクションモデル」では、トランザクション分離レベルおよびそれぞれで 사용되는ロック戦略について説明します。また、`autocommit`、一貫性非ロック読取りおよびロック読取りの使用についても説明します。
- セクション 15.7.3 「InnoDB のさまざまな SQL ステートメントで設定されたロック」では、様々なステートメントに対して InnoDB で設定される特定のタイプのロックについて説明します。

- [セクション15.7.4「ファントム行」](#) では、InnoDB が次のキーロックを使用してファントム行を回避する方法について説明します。
- [セクション15.7.5「InnoDB のデッドロック」](#) にはデッドロックの例が用意されており、デッドロックの検出について説明し、InnoDB でデッドロックを最小化および処理するためのヒントを提供しています。

15.7.1 InnoDB ロック

このセクションでは、InnoDB で使用されるロックタイプについて説明します。

- [共有ロックと排他ロック](#)
- [インテンションロック](#)
- [レコードロック](#)
- [ギャップロック](#)
- [ネクストキーロック](#)
- [インテンションロックの挿入](#)
- [AUTO-INC ロック](#)
- [空間インデックスの述語ロック](#)

共有ロックと排他ロック

InnoDB では、2つのロックタイプ ([共有 \(S\) ロック](#)と[排他 \(X\) ロック](#)) がある標準の行レベルロックが実装されます。

- [共有 \(S\) ロック](#)では、ロックを保持するトランザクションによる行の読み取りが許可されます。
- [排他 \(X\) ロック](#)では、ロックを保持するトランザクションによる行の更新または削除が許可されます。

トランザクション T_1 が行 r に対する共有 (S) ロックを保持している場合、別のトランザクション T_2 からの行 r に対するロック要求は次のように処理されます。

- T_2 による S ロックに対するリクエストは、すぐに付与できます。結果として、 T_1 と T_2 の両方が r 上で S ロックを保持します。
- T_2 による X ロックに対するリクエストは、すぐに付与できません。

トランザクション T_1 が行 r 上で排他 (X) ロックを保持している場合は、 r 上のいずれかのタイプのロックに対する一部の個別のトランザクション T_2 からのリクエストは、すぐに付与できません。代わりに、トランザクション T_2 は、行 r 上でトランザクション T_1 のロックが解放されるまで待機する必要があります。

インテンションロック

InnoDB では、行ロックとテーブルロックの共存を許可する複数粒度ロックがサポートされています。たとえば、`LOCK TABLES ... WRITE` などのステートメントは、指定されたテーブルに対して排他ロック (X ロック) を取得します。複数の粒度レベルでロックするには、InnoDB で [intention locks](#) を使用します。インテントロックは、トランザクションが後でテーブルの行に必要とするロックのタイプ (共有または排他) を示すテーブルレベルのロックです。インテントロックには、次の2種類があります:

- [intention shared lock \(IS\)](#) は、トランザクションがテーブルの個々の行に shared ロックを設定することを示します。
- [intention exclusive lock \(IX\)](#) は、トランザクションがテーブル内の個々の行に排他ロックを設定することを示します。

たとえば、`SELECT ... FOR SHARE` は IS ロックを設定し、`SELECT ... FOR UPDATE` は IX ロックを設定します。

インテンションロックの手順は次のとおりです。

- トランザクションは、テーブル内の行に対する共有ロックを取得する前に、まず IS ロックを取得するか、テーブルに対して強いロックを取得する必要があります。
- トランザクションは、テーブル内の行に対する排他ロックを取得する前に、まずテーブルに対する IX ロックを取得する必要があります。

次のマトリックスに、テーブルレベルのロックタイプの互換性の概要を示します。

	X	IX	S	IS
X	競合	競合	競合	競合
IX	競合	互換	競合	互換
S	競合	競合	互換	互換
IS	競合	互換	互換	互換

ロックに既存のロックとの互換性がある場合は、リクエスト元のトランザクションにロックが付与されますが、既存のロックと競合している場合は、ロックが付与されません。トランザクションは、競合している既存のロックが解放されるまで待機します。ロックリクエストが既存のロックと競合し、**デッドロック**が発生するために付与できない場合は、エラーが発生します。

意図的ロックでは、完全なテーブルリクエスト (**LOCK TABLES ... WRITE** など) 以外はブロックされません。意図的ロックの主な目的は、誰かが行をロックしていること、またはテーブル内の行をロックしていることを示すことです。

_intent ロックのトランザクションデータは、**SHOW ENGINE INNODB STATUS** および **InnoDB monitor** の出力に次のように表示されます:

```
TABLE LOCK table `test`.`t` trx id 10080 lock mode IX
```

レコードロック

レコードロックは、インデックスレコードのロックです。たとえば、**SELECT c1 FROM t WHERE c1 = 10 FOR UPDATE;**では、**t.c1** の値が **10** の場合、他のトランザクションによる行の挿入、更新または削除が防止されます。

レコードロックでは、テーブルにインデックスが定義されていなくても必ず、インデックスレコードがロックされます。このような場合は、**InnoDB** によって非表示のクラスタ化されたインデックスが作成され、このインデックスを使用してレコードロックが行われます。**セクション15.6.2.1「クラスタインデックスとセカンダリインデックス」**を参照してください。

レコードロックのトランザクションデータは、**SHOW ENGINE INNODB STATUS** および **InnoDB monitor** 出力に次のように表示されます:

```
RECORD LOCKS space id 58 page no 3 n bits 72 index `PRIMARY` of table `test`.`t`
trx id 10078 lock_mode X locks rec but not gap
Record lock, heap no 2 PHYSICAL RECORD: n_fields 3; compact format; info bits 0
0: len 4; hex 8000000a; asc  ;;
1: len 6; hex 00000000274f; asc  'O';
2: len 7; hex b60000019d0110; asc  ;;
```

ギャップロック

ギャップロックは、インデックスレコード間のギャップのロック、または最初のインデックスレコードの前または最後のインデックスレコードの後のギャップのロックです。たとえば、**SELECT c1 FROM t WHERE c1 BETWEEN 10 and 20 FOR UPDATE;**では、範囲内の既存のすべての値間のギャップがロックされているため、カラムにそのような値がすでに存在するかどうかにかかわらず、他のトランザクションが **15** の値をカラム **t.c1** に挿入できなくなります。

ギャップは、単一のインデックス値、複数のインデックス値にまたがることも、空にすることもできます。

ギャップロックは、パフォーマンスと並列性とのトレードオフの一環であり、一部のトランザクション分離レベルで使用され、ほかでは使用されません。

一意のインデックスを使用して一意の行を検索することで行をロックするステートメントでは、ギャップロックは必要ありません。(これには、検索条件に複数カラムの一意のインデックスの一部のカラムのみが含まれるケースは含まれません。この場合は、ギャップロックが発生します。)たとえば、`id` カラムに一意のインデックスが設定されている場合、次のステートメントで使用されるのは `id` の値が 100 の行に対するインデックスレコードロックだけとなり、ほかのセッションがそのレコードの前にあるギャップに行を挿入するかどうかは問題ではなくなります。

```
SELECT * FROM child WHERE id = 100;
```

`id` にインデックスが設定されていなかったり、一意でないインデックスが設定されていたりすると、このステートメントで先行するギャップがロックされます。

さまざまなトランザクションによってギャップ上に競合するロックを保持できることも、ここで注目すべき点です。たとえば、トランザクション A はギャップ上に共有ギャップロック (ギャップ S ロック) を保持できる一方で、トランザクション B は同じギャップ上に排他ギャップロック (ギャップ X ロック) を保持します。競合するギャップロックが許可される理由は、レコードがインデックスからバージョンされる場合に、さまざまなトランザクションによってレコード上に保持されたギャップロックをマージする必要があるためです。

InnoDB のギャップロックは「純粋に阻害」です。つまり、その唯一の目的は、他のトランザクションがギャップに挿入されないようにすることです。ギャップロックは共存できます。あるトランザクションによって取得されたギャップロックによって、別のトランザクションが同じギャップに対してギャップロックを取得することが妨げられることはありません。共有ギャップロックと排他ギャップロックに違いはありません。これらは互いに競合せず、同じ機能を実行します。

ギャップロックは明示的に無効化できます。これは、トランザクション分離レベルを `READ COMMITTED` に変更した場合に発生します。このような状況では、ギャップロックは検索およびインデックススキャン時に無効化され、外部キー制約チェックおよび重複キーチェック時のみ使用されます。

`READ COMMITTED` 分離レベルの使用には、他にも影響があります。一致しなかった行のレコードロックは、MySQL による `WHERE` 条件の評価後に解除されます。`UPDATE` ステートメントの場合、InnoDB は最後にコミットされたバージョンが MySQL に返されるように、「半一貫性」読み取りを実行します。これにより、MySQL はその行が `UPDATE` の `WHERE` 条件に一致するかどうかを判断できます。

ネクストキーロック

次のキーロックは、インデックスレコードのレコードロックと、インデックスレコードの前のギャップのギャップロックの組み合わせです。

InnoDB は、テーブルインデックスを検索またはスキャンするときに、生成されたインデックスレコード上に共有ロックまたは排他ロックを設定するという方法で、行レベルロックを実行します。したがって、行レベルロックは、実際にはインデックスレコードロックです。インデックスレコードに対する次のキーロックは、そのインデックスレコードの前の「gap」にも影響します。つまり、ネクストキーロックは、インデックスレコードロックと、そのインデックスレコードの前のギャップに対するギャップロックとを組み合わせたものです。あるセッションがインデックス内のレコード `R` 上に共有ロックまたは排他ロックを持っている場合は、別のセッションがインデックスの順番で `R` の直前にあるギャップに新しいインデックスレコードを挿入できません。

あるインデックスに値 10、11、13、20 が含まれているとします。このインデックスで使用可能な次のキーロックは、次の間隔を対象としています。丸カッコは間隔エンドポイントの除外を示し、角カッコはエンドポイントの包含を示します:

```
(negative infinity, 10]
(10, 11]
(11, 13]
(13, 20]
(20, positive infinity)
```

最後の間隔ではネクストキーロックによって、インデックス内の最大値を上回るギャップ、およびインデックス内の実際のどの値よりも大きい値を持つ「最小上限」の擬似レコードがロックされます。最小上限は実際のインデックスレコードではないため、事実上、このネクストキーロックによってロックされるのは、最大インデックス値のあとにあるギャップのみです。

デフォルトでは、InnoDB は `REPEATABLE READ` トランザクション分離レベルで動作します。この場合、InnoDB はネクストキーロックを使用して検索およびインデックススキャンを行うため、ファントム行の発生を回避できます ([セクション 15.7.4 「ファントム行」](#) を参照)。

次のキーロックのトランザクションデータは、`SHOW ENGINE INNODB STATUS` および `InnoDB monitor` の出力に次のように表示されます:

```
RECORD LOCKS space id 58 page no 3 n bits 72 index `PRIMARY` of table `test`.`t`
trx id 10080 lock_mode X
Record lock, heap no 1 PHYSICAL RECORD: n_fields 1; compact format; info bits 0
0: len 8; hex 73757072656d756d; asc supremum;;

Record lock, heap no 2 PHYSICAL RECORD: n_fields 3; compact format; info bits 0
0: len 4; hex 8000000a; asc ;;
1: len 6; hex 00000000274f; asc 'O';
2: len 7; hex b60000019d0110; asc ;;
```

インテンションロックの挿入

挿入意図ロックは、行の挿入前に `INSERT` 操作によって設定されるギャップロックのタイプです。このロックは、同じインデックスギャップに挿入する複数のトランザクションは、そのギャップ内の同じ場所に挿入しなければ相互に待機する必要がないように、意図的に挿入することを示しています。値が 4 と 7 のインデックスレコードが存在すると仮定します。5 と 6 の値をそれぞれ挿入しようとする個別のトランザクションでは、挿入された行の排他ロックを取得する前に、挿入意図ロックを使用して 4 と 7 のギャップがロックされますが、行が競合していないため相互にブロックされません。

次の例は、挿入されたレコードの排他ロックを取得する前に挿入意図ロックを取得するトランザクションを示しています。この例には、A と B の 2 つのクライアントが登場します。

クライアント A は、2 つのインデックスレコード (90 および 102) を含むテーブルを作成し、100 を超える ID を持つインデックスレコードに排他ロックを設定するトランザクションを開始します。排他ロックには、レコード 102 の前にギャップロックが含まれます:

```
mysql> CREATE TABLE child (id int(11) NOT NULL, PRIMARY KEY(id)) ENGINE=InnoDB;
mysql> INSERT INTO child (id) values (90),(102);

mysql> START TRANSACTION;
mysql> SELECT * FROM child WHERE id > 100 FOR UPDATE;
+----+
| id |
+----+
| 102 |
+----+
```

クライアント B はトランザクションを開始して、ギャップにレコードを挿入します。トランザクションは、排他ロックの取得を待機している間、挿入意図ロックを取得します。

```
mysql> START TRANSACTION;
mysql> INSERT INTO child (id) VALUES (101);
```

挿入意図ロックのトランザクションデータは、`SHOW ENGINE INNODB STATUS` および `InnoDB monitor` 出力に次のように表示されます:

```
RECORD LOCKS space id 31 page no 3 n bits 72 index `PRIMARY` of table `test`.`child`
trx id 8731 lock_mode X locks gap before rec insert intention waiting
Record lock, heap no 3 PHYSICAL RECORD: n_fields 3; compact format; info bits 0
0: len 4; hex 80000066; asc f;;
1: len 6; hex 000000002215; asc " ;;
2: len 7; hex 9000000172011c; asc 'r' ;;...
```

AUTO-INC ロック

`AUTO-INC` ロックは、`AUTO_INCREMENT` カラムを含むテーブルに挿入されるトランザクションによって取得される特別なテーブルレベルロックです。もっとも単純なケースでは、あるトランザクションがテーブルに値を挿入している場合に、ほかのトランザクションはそのテーブルへのそれぞれの挿入を待機する必要があるため、最初のトランザクションによって挿入された行が、連続する主キー値を受け取ります。

`innodb_autoinc_lock_mode` 構成オプションは、自動増分ロックに使用されるアルゴリズムを制御します。これにより、自動増分値の予測可能なシーケンスと挿入操作の最大同時実行性の間のトレードオフ方法を選択できます。

詳細は、[セクション 15.6.1.6 「InnoDB での AUTO_INCREMENT 処理」](#) を参照してください。

空間インデックスの述語ロック

InnoDB では、空間カラムを含むカラムの **SPATIAL** インデックス付けがサポートされています ([セクション11.4.9「空間分析の最適化」](#) を参照)。

SPATIAL インデックスを含む操作のロックを処理するために、次キーロックは **REPEATABLE READ** または **SERIALIZABLE** のトランザクション分離レベルをサポートするのに適切に機能しません。マルチディメンショナルデータには絶対順序付けの概念がないため、「next」キーは明確ではありません。

SPATIAL インデックスを含むテーブルの分離レベルのサポートを有効にするために、InnoDB では述語ロックを使用します。**SPATIAL** インデックスには最小境界矩形 (MBR) 値が含まれているため、InnoDB では、クエリーに使用される MBR 値に述語ロックを設定することで、インデックスに対する読取り一貫性が強制されます。他のトランザクションでは、クエリー条件に一致する行を挿入または変更できません。

15.7.2 InnoDB トランザクションモデル

InnoDB トランザクションモデルでの目標は、**multi-versioning** データベースの最適なプロパティを従来の 2 フェーズロックと組み合わせることです。InnoDB は、行レベルでロックを実行し、Oracle のスタイルでクエリーを非ロック **consistent reads** としてデフォルトで実行します。InnoDB のロック情報は領域効率に優れて格納されるため、ロックのエスカレーションは必要ありません。通常、いくつかのユーザーは、InnoDB メモリーを使い果たさずに、InnoDB テーブルのすべての行または行のランダムなサブセットをロックできます。

15.7.2.1 トランザクション分離レベル

トランザクションの分離は、データベース処理の基礎の 1 つです。分離とは、頭字語の **ACID** における I です。分離レベルは、複数のトランザクションで変更が行われ、クエリーが同時に実行される場合に、パフォーマンスと信頼性、一貫性および結果の再現性のバランスを微調整する設定です。

InnoDB では、SQL:1992 標準に記載された 4 つのトランザクション分離レベル (**READ UNCOMMITTED**、**READ COMMITTED**、**REPEATABLE READ**、**SERIALIZABLE**) がすべて提供されます。InnoDB のデフォルトの分離レベルは **REPEATABLE READ** です。

ユーザーは **SET TRANSACTION** ステートメントを使用して単一のセッションまたは後続のすべての接続の分離レベルを変更できます。すべての接続に対するサーバーのデフォルトの分離レベルを設定するには、コマンド行上、またはオプションファイル内で **--transaction-isolation** オプションを使用します。分離レベルおよびレベル設定構文についての詳細は、[セクション13.3.7「SET TRANSACTION ステートメント」](#) を参照してください。

InnoDB は、ここで説明されている各トランザクション分離レベルを、異なる**ロック**の方法を使用してサポートしています。**ACID** 準拠が重要な要件である重要なデータに対する操作の場合は、デフォルトの **REPEATABLE READ** レベルを使用して高度な一貫性を適用できます。あるいは、正確な一貫性や繰り返し可能な結果がロックのためのオーバーヘッドの量の最少化ほど重要でない一括レポートなどの状況では、**READ COMMITTED** や場合によっては **READ UNCOMMITTED** を使用して一貫性のルールを緩和できます。**SERIALIZABLE** は **REPEATABLE READ** よりさらに厳密なルールを適用し、主に **XA** トランザクションのほか、並列性や**デッドロック**に関する問題のトラブルシューティングなどの特殊な状況で使用されます。

次のリストは、MySQL が各種のトランザクションレベルをどのようにサポートするかについて説明しています。このリストは、もっとも一般的に使用されるレベルから使用頻度の低い順に並べられています。

• **REPEATABLE READ**

これが InnoDB のデフォルトの分離レベルです。同じトランザクション内の **Consistent reads** は、最初の読取りによって確立された **snapshot** を読み取ります。つまり、同じトランザクション内で複数のプレーン (非ロック) **SELECT** ステートメントを発行すると、これらの **SELECT** ステートメントも互いに一貫性が保たれます。[セクション15.7.2.3「一貫性非ロック読み取り」](#) を参照してください。

locking reads (**FOR UPDATE** または **FOR SHARE** のある **SELECT**)、**UPDATE** および **DELETE** ステートメントの場合、ロックはステートメントが一意的な検索条件を持つ一意のインデックスを使用するか、範囲タイプの検索条件を使用するかによって異なります。

- 一意の検索条件を使用した一意のインデックスの場合、InnoDB は見つかったインデックスレコードのみをロックし、その前にある**ギャップ**はロックしません。

- その他の検索条件の場合、InnoDB は、**ギャップロック**または**ネクストキーロック**を使用して、範囲に含まれるギャップへのほかのセッションによる挿入をブロックすることによって、スキャンされたインデックス範囲をロックします。ギャップロックおよびネクストキーロックについては、[セクション15.7.1「InnoDB ロック」](#)を参照してください。
- **READ COMMITTED**

各読み取り一貫性は、同じトランザクション内であっても、独自の新しいスナップショットを設定して読み取ります。読み取り一貫性の詳細は、[セクション15.7.2.3「一貫性非ロック読み取り」](#)を参照してください。

ロック読み取り (**FOR UPDATE** または **FOR SHARE** を使用した **SELECT**)、**UPDATE** ステートメントおよび **DELETE** ステートメントの場合、InnoDB はインデックスレコードのみをロックし、その前のギャップはロックしないため、ロックされたレコードの横に新しいレコードを自由に挿入できます。ギャップロックは、外部キー制約チェックおよび重複キーチェックにのみ使用されます。

ギャップロックが無効になっているため、他のセッションがギャップに新しい行を挿入できるため、ファントムの問題が発生する可能性があります。ファントムの詳細は、[セクション15.7.4「ファントム行」](#)を参照してください。

READ COMMITTED 分離レベルでは、行ベースのバイナリロギングのみがサポートされます。**READ COMMITTED** を **binlog_format=MIXED** とともに使用する場合、サーバーは自動的に行ベースのロギングを使用しません。

READ COMMITTED の使用には、追加の効果があります:

- **UPDATE** または **DELETE** ステートメントでは、InnoDB は更新または削除の対象となる行に対してのみ、ロックを保持します。一致しなかった行のレコードロックは、MySQL による **WHERE** 条件の評価後に解除されます。これにより、デッドロックの可能性が大幅に低くなりますが、まだ発生する可能性があります。
- **UPDATE** ステートメントである行がすでにロックされていた場合、InnoDB は「半一貫性」読み取りを実行し、最後にコミットされたバージョンを MySQL に返すため、MySQL はその行が **UPDATE** の **WHERE** 条件に一致するかどうかを判断できます。その行が一致した場合 (その行を更新する必要がある場合)、MySQL はその行を再度読み取り、InnoDB は今度はその行をロックするか、その行のロックが解除されるまで待機します。

次のような例について、このテーブルから検討します。

```
CREATE TABLE t (a INT NOT NULL, b INT) ENGINE = InnoDB;  
INSERT INTO t VALUES (1,2),(2,3),(3,2),(4,3),(5,2);  
COMMIT;
```

この場合、テーブルにはインデックスがないため、検索およびインデックススキャンでは、インデックス付けされたカラムではなく、非表示のクラスタインデックスをレコードロックに使用します ([セクション15.6.2.1「クラスタインデックスとセカンダリインデックス」](#)を参照)。

あるセッションで、次のステートメントを使用して **UPDATE** を実行するとします:

```
# Session A  
START TRANSACTION;  
UPDATE t SET b = 5 WHERE b = 3;
```

また、最初のセッションのステートメントの後に次のステートメントを実行して、別のセッションで **UPDATE** を実行するとします:

```
# Session B  
UPDATE t SET b = 4 WHERE b = 2;
```

InnoDB は各 **UPDATE** を実行する際に、まず各行の排他ロックを取得し、次にその行を変更するかどうかを判断します。InnoDB は、行を変更しない場合、ロックを解放します。それ以外の場合、トランザクションが終了するまで InnoDB はそのロックを保持します。これにより、トランザクション処理が次のような影響を受けます。

デフォルトの **REPEATABLE READ** 分離レベルを使用する場合、最初の **UPDATE** は読み取り対象の各行に対して x ロックを取得し、それらのいずれも解放しません:

```
x-lock(1,2); retain x-lock
```

```
x-lock(2,3); update(2,3) to (2,5); retain x-lock
x-lock(3,2); retain x-lock
x-lock(4,3); update(4,3) to (4,5); retain x-lock
x-lock(5,2); retain x-lock
```

次のように、2 番目の **UPDATE** は (1 番目の更新がすべての行のロックを保持しているため)、ロックを取得しようとしてもすぐにブロックされ、1 番目の **UPDATE** がコミットまたはロールバックを実行するまで続行されません。

```
x-lock(1,2); block and wait for first UPDATE to commit or roll back
```

かわりに **READ COMMITTED** が使用されている場合、最初の **UPDATE** は読み取る各行で x ロックを取得し、変更しない行の X ロックを解放します:

```
x-lock(1,2); unlock(1,2)
x-lock(2,3); update(2,3) to (2,5); retain x-lock
x-lock(3,2); unlock(3,2)
x-lock(4,3); update(4,3) to (4,5); retain x-lock
x-lock(5,2); unlock(5,2)
```

2 番目の **UPDATE** では、InnoDB は「semi-consistent」読取りを実行し、MySQL に読み取られた各行の最新のコミット済バージョンを返して、MySQL がその行が **UPDATE** の **WHERE** 条件に一致するかどうかを判断できるようにします:

```
x-lock(1,2); update(1,2) to (1,4); retain x-lock
x-lock(2,3); unlock(2,3)
x-lock(3,2); update(3,2) to (3,4); retain x-lock
x-lock(4,3); unlock(4,3)
x-lock(5,2); update(5,2) to (5,4); retain x-lock
```

ただし、**WHERE** 条件にインデックス付けされたカラムが含まれており、InnoDB でインデックスが使用されている場合は、レコードロックを取得および保持するときにインデックス付けされたカラムのみが考慮されます。次の例では、最初の **UPDATE** は、b = 2 の各行で x ロックを取得して保持します。2 番目の **UPDATE** は、カラム b に定義されたインデックスも使用するため、同じレコードで x ロックを取得しようとするブロックされます。

```
CREATE TABLE t (a INT NOT NULL, b INT, c INT, INDEX (b)) ENGINE = InnoDB;
INSERT INTO t VALUES (1,2,3),(2,2,4);
COMMIT;

# Session A
START TRANSACTION;
UPDATE t SET b = 3 WHERE b = 2 AND c = 3;

# Session B
UPDATE t SET b = 4 WHERE b = 2 AND c = 4;
```

READ COMMITTED 分離レベルは、起動時に設定することも、実行時に変更することもできます。実行時に、すべてのセッションに対してグローバルに設定することも、セッションごとに個別に設定することもできます。

- **READ UNCOMMITTED**

SELECT ステートメントは非ロックの方法で実行されますが、以前のバージョンの行が使用される可能性もあります。そのため、この分離レベルを使用すると、このような読み取りには一貫性がありません。これは、**ダーティー読み取り**とも呼ばれます。そうでなければ、この分離レベルは **READ COMMITTED** のように機能します。

- **SERIALIZABLE**

このレベルは **REPEATABLE READ** と似ていますが、**autocommit** が無効になっている場合、InnoDB はすべてのプレーン **SELECT** ステートメントを **SELECT ... FOR SHARE** に暗黙的に変換します。**autocommit** が有効な場合、**SELECT** は独自のトランザクションです。したがって、読み取り専用であることがわかっているため、一貫性のある (非ロック) 読み取りとして実行された場合は直列化することができ、ほかのトランザクションのためのブ

ロックは必要ありません。(選択した行が他のトランザクションによって変更された場合にプレーン `SELECT` を強制的にブロックするには、`autocommit` を無効にします。)

注記

MySQL 8.0.22 の時点では、MySQL からデータを読み取る DML 操作では、分離レベルに関係なく、(結合リストまたはサブクエリーを介して) テーブルが付与されますが、変更はされません。詳細は、[テーブル同時実行性の付与](#)を参照してください。

15.7.2.2 自動コミット、コミットおよびロールバック

InnoDB では、すべてのユーザーアクティビティーがトランザクション内部で発生します。`autocommit` モードが有効な場合、各 SQL ステートメントは単独で単一のトランザクションを形成します。デフォルトでは、MySQL は、`autocommit` が有効になっている新しい接続ごとにセッションを開始するため、MySQL は、SQL ステートメントがエラーを戻さなかった場合に、各 SQL ステートメントの後にコミットを実行します。ステートメントからエラーが返された場合、コミットまたはロールバックの動作はそのエラーによって異なります。[セクション 15.21.4「InnoDB のエラー処理」](#)を参照してください。

`autocommit` が有効になっているセッションは、明示的な `START TRANSACTION` ステートメントまたは `BEGIN` ステートメントで開始し、`COMMIT` ステートメントまたは `ROLLBACK` ステートメントで終了することで、複数ステートメントのトランザクションを実行できます。[セクション 13.3.1「START TRANSACTION、COMMIT および ROLLBACK ステートメント」](#)を参照してください。

`SET autocommit = 0` とのセッション内で `autocommit` モードが無効になっている場合、セッションでは常にトランザクションがオープンされています。`COMMIT` または `ROLLBACK` ステートメントは現在のトランザクションを終了し、新しいセッションを開始します。

`autocommit` が無効になっているセッションが、最終トランザクションを明示的にコミットせずに終了した場合、MySQL はそのトランザクションをロールバックします。

一部のステートメントは、ユーザーがそのステートメントの実行前に `COMMIT` を実行した場合と同様に、暗黙的にトランザクションを終了します。詳細は、[セクション 13.3.3「暗黙的なコミットを発生させるステートメント」](#)を参照してください。

`COMMIT` は、現在のトランザクション内で行われた変更は永続的であり、その他のセッションから表示できることを意味します。反対に、`ROLLBACK` ステートメントは、現在のトランザクションによって行われたすべての変更を取り消します。`COMMIT` と `ROLLBACK` は両方とも、現在のトランザクション中に設定されたすべての InnoDB ロックを解除します。

トランザクションを使用した DML 操作のグループ化

デフォルトでは、MySQL サーバーへの接続は、[自動コミット](#)モードが有効になっている状態で開始されるため、SQL ステートメントは実行するたびに自動的にコミットされます。一連の DML ステートメントを発行し、すべてまとめてコミットまたはロールバックすることが標準操作となっているほかのデータベースシステムの使用経験がある場合は、この操作モードに馴染みがないかもしれません。

複数ステートメントのトランザクションを使用するには、SQL ステートメント `SET autocommit = 0` を使用して自動コミットをオフにして、必要に応じて `COMMIT` または `ROLLBACK` を使用して各トランザクションを終了します。自動コミットをオンのままにするには、`START TRANSACTION` を使用して各トランザクションを開始し、`COMMIT` または `ROLLBACK` を使用して終了します。次の例は 2 つのトランザクションを表しています。1 番目はコミットされ、2 番目はロールバックされています。

```
shell> mysql test
```

```
mysql> CREATE TABLE customer (a INT, b CHAR (20), INDEX (a));
Query OK, 0 rows affected (0.00 sec)
mysql> -- Do a transaction with autocommit turned on.
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO customer VALUES (10, 'Heikki');
Query OK, 1 row affected (0.00 sec)
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> -- Do another transaction with autocommit turned off.
mysql> SET autocommit=0;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO customer VALUES (15, 'John');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO customer VALUES (20, 'Paul');
Query OK, 1 row affected (0.00 sec)
mysql> DELETE FROM customer WHERE b = 'Heikki';
Query OK, 1 row affected (0.00 sec)
mysql> -- Now we undo those last 2 inserts and the delete.
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT * FROM customer;
+----+-----+
| a  | b    |
+----+-----+
| 10 | Heikki |
+----+-----+
1 row in set (0.00 sec)
mysql>
```

クライアント側言語でのトランザクション

PHP、Perl DBI、JDBC、ODBC などの API または MySQL の標準 C 呼び出しインタフェースでは、**COMMIT** などのトランザクション制御ステートメントを **SELECT** や **INSERT** などのその他の SQL ステートメントと同様の文字列として、MySQL サーバーに送信できます。一部の API では、別個の特別なトランザクションコミットおよびロールバックの関数やメソッドも提供されています。

15.7.2.3 一貫性非ロック読み取り

一貫性読み取りとは、InnoDB がマルチバージョンを使用して、ある時点でのデータベースのスナップショットをクエリーに提供することを意味します。クエリーには、その時点よりも前にコミットされたトランザクションによる変更のみが表示され、その時点よりもあとのトランザクションまたはコミットされていないトランザクションによる変更は表示されません。このルールの例外として、同じトランザクション内の以前のステートメントによる変更はクエリーに表示されます。この例外によって、次のような異常が発生します。テーブル内の一部の行を更新すると、更新された行の最新バージョンが **SELECT** に表示されますが、いずれかの行の旧バージョンも表示される可能性があります。その他のセッションで同じテーブルが同時に更新される場合、その異常は、データベースに存在しない状態でテーブルが表示される可能性があることを意味します。

トランザクション分離レベルが **REPEATABLE READ** (デフォルトのレベル) である場合は、同じトランザクション内のすべての一貫性読み取りで、そのトランザクション内の最初のこのような読み取りで確立されたスナップショットが読み取られます。現在のトランザクションをコミットしたあとに、新しいクエリーを発行すると、クエリーの新しいスナップショットを取得できます。

分離レベルが **READ COMMITTED** の場合は、トランザクション内の各一貫性読み取りで、独自の新しいスナップショットが設定され、読み取られます。

一貫性読み取りは、InnoDB が **READ COMMITTED** および **REPEATABLE READ** 分離レベルで **SELECT** ステートメントを処理する際のデフォルトモードです。一貫性読み取りではアクセスされるテーブル上にロックが設定されないため、その他のセッションも、そのテーブル上で一貫性読み取りが実行されるときと同時に、それらのテーブルを自由に変更できます。

デフォルトの **REPEATABLE READ** 分離レベルで実行していると仮定します。一貫性読み取り (つまり、通常の **SELECT** ステートメント) を発行すると、InnoDB は、クエリーがデータベースを参照するときの基準となるタイムポイントにトランザクションに付与します。タイムポイントが割り当てられたあとに、別のトランザクションが行を削除してコミットした場合は、その行が削除済みとして表示されません。挿入および更新も同様に処理されます。

注記

データベースの状態のスナップショットは、トランザクション内の **SELECT** ステートメントに適用されますが、**DML** ステートメントには必ずしも適用されるとは限りません。一部の行を挿入または変更してから、そのトランザクションをコミットする場合は、そのセッションでクエリーが実行される可能性がない場合でも、別の並列実行 **REPEATABLE READ** トランザクションから発行された **DELETE** または **UPDATE** ステートメントによって、コミットされたばかりの行が影響を受ける可能性があります。トランザクションによって別の

トランザクションでコミットされた行が更新または削除されると、これらの変更を現在のトランザクションに表示できるようになります。たとえば、次のような状況が発生する可能性があります。

```
SELECT COUNT(c1) FROM t1 WHERE c1 = 'xyz';
-- Returns 0: no rows match.
DELETE FROM t1 WHERE c1 = 'xyz';
-- Deletes several rows recently committed by other transaction.

SELECT COUNT(c2) FROM t1 WHERE c2 = 'abc';
-- Returns 0: no rows match.
UPDATE t1 SET c2 = 'cba' WHERE c2 = 'abc';
-- Affects 10 rows: another txn just committed 10 rows with 'abc' values.
SELECT COUNT(c2) FROM t1 WHERE c2 = 'cba';
-- Returns 10: this txn can now see the rows it just updated.
```

トランザクションをコミットしてから、別の [SELECT](#) または [START TRANSACTION WITH CONSISTENT SNAPSHOT](#) を実行すると、タイムポイントを進めることができます。

これは、マルチバージョン並列処理制御と呼ばれます。

次の例では、セッション B が挿入をコミットし、セッション A も同様にコミットした場合にのみ、B によって挿入された行が A に表示されます。これにより、タイムポイントが B のコミットよりも先に進みます。

	Session A	Session B
time	SET autocommit=0;	SET autocommit=0;
	SELECT * FROM t;	
	empty set	
		INSERT INTO t VALUES (1, 2);
v	SELECT * FROM t;	
	empty set	
		COMMIT;
	SELECT * FROM t;	
	empty set	
	COMMIT;	
	SELECT * FROM t;	

	1 2	

データベースの「最新」状態を確認する場合は、[READ COMMITTED](#) 分離レベルと[ロック読み取り](#)のいずれかを使用します。

```
SELECT * FROM t FOR SHARE;
```

分離レベルが [READ COMMITTED](#) の場合は、トランザクション内の各一貫性読み取りで、独自の新しいスナップショットが設定され、読み取られます。[FOR SHARE](#) では、かわりにロック読み取りが発生: [SELECT](#) は、最新の行を含むトランザクションが終了するまでブロックされます ([セクション15.7.2.4「読み取りのロック」](#) を参照)。

特定の DDL ステートメントでは、一貫性読み取りが機能しません。

- [DROP TABLE](#) では、MySQL が削除されたテーブルを使用できず、そのテーブルは [InnoDB](#) によって破棄されるため、一貫性読み取りが機能しません。
- 読み取り一貫性は、元のテーブルの一時コピーを作成し、一時コピーの作成時に元のテーブルを削除する [ALTER TABLE](#) 操作では機能しません。トランザクション内で一貫性読み取りを再発行しても、新しいテーブル内の行はトランザクションのスナップショット取得されたときには存在していなかったため、表示できません。この場合、トランザクションはエラーを返します: [ER_TABLE_DEF_CHANGED](#)、「テーブル定義が変更されました。トランザクションを再試行してください」。

読み取りのタイプは、[FOR UPDATE](#) または [FOR SHARE](#) を指定しない [INSERT INTO ... SELECT](#)、[UPDATE ... \(SELECT\)](#) および [CREATE TABLE ... SELECT](#) などの句での選択によって異なります:

- デフォルトでは、InnoDB はこれらのステートメントに対してより強力なロックを使用し、SELECT 部分は READ COMMITTED のように動作します。この場合、各読取り一貫性は、同じトランザクション内であっても、独自の新しいスナップショットを設定および読み取ります。
- このような場合に非ロック読取りを実行するには、トランザクションの分離レベルを READ UNCOMMITTED または READ COMMITTED に設定して、選択したテーブルから読み取られた行にロックを設定しないようにします。

15.7.2.4 読取りのロック

データのクエリーを実行してから、同じトランザクション内で関連データを挿入または更新する場合は、通常の SELECT ステートメントで十分な保護が提供されません。ほかのトランザクションは、クエリーが実行されたばかりの同じ行を更新または削除できます。InnoDB では、追加の安全性が提供される 2 つのタイプの **ロック読取り** がサポートされています。

- SELECT ... FOR SHARE

読み取られる行に共有モードロックを設定します。ほかのセッションもその行を読み取ることができますが、トランザクションがコミットするまで変更することはできません。これらの行のいずれかがコミットされていない別のトランザクションによって変更された場合、クエリーはそのトランザクションが終了するまで待機してから、最新の値を使用します。

注記

SELECT ... FOR SHARE は SELECT ... LOCK IN SHARE MODE の代替機能ですが、LOCK IN SHARE MODE は下位互換性のために引き続き使用できます。ステートメントは同等です。ただし、FOR SHARE は OF table_name、NOWAIT および SKIP LOCKED オプションをサポートしています。NOWAIT および SKIP LOCKED による読取り同時実行性のロックを参照してください。

MySQL 8.0.22 より前は、SELECT ... FOR SHARE には SELECT 権限と、DELETE、LOCK TABLES または UPDATE のいずれかの権限が必要です。MySQL 8.0.22 では、SELECT 権限のみが必要です。

MySQL 8.0.22 の時点では、SELECT ... FOR SHARE ステートメントは MySQL 付与テーブルの読取りロックを取得しません。詳細は、[テーブル同時実行性の付与](#)を参照してください。

- SELECT ... FOR UPDATE

検索で検出されたインデックスレコードについては、それらの行に対して UPDATE ステートメントを発行した場合と同じように、行および関連するインデックスエントリがロックされます。他のトランザクションは、これらの行の更新、SELECT ... FOR SHARE の実行、または特定のトランザクション分離レベルでのデータの読取りをブロックされます。一貫性読取りでは、読み取られたビュー内に存在するレコードに設定されたロックはすべて無視されます。(古いバージョンのレコードはロックできません。レコードのインメモリーコピー上の Undo ログに適用することで、再構築されます。)

SELECT ... FOR UPDATE には、SELECT 権限と、DELETE、LOCK TABLES または UPDATE のいずれかの権限が必要です。

これらの句は、主に、単一のテーブル内または複数のテーブルに分割された状態で、ツリー構造またはグラフ構造のデータを処理する際に役立ちます。エッジまたはツリー分岐のある場所から別の場所にトラバースしても、これらの「ポインタ」に戻ってその値を変更する権利を保有しています。

FOR SHARE および FOR UPDATE クエリーによって設定されたすべてのロックは、トランザクションがコミットまたはロールバックされると解放されます。

注記

ロック読取りは、(START TRANSACTION でトランザクションを開始するか、autocommit を 0 に設定することで) 自動コミットが無効になっている場合にのみ可能です。

外部ステートメントのロック読取り句では、サブクエリーにロック読取り句も指定されていないかぎり、ネストしたサブクエリーのテーブルの行はロックされません。たとえば、次のステートメントでは、テーブル t2 の行はロックされません。

```
SELECT * FROM t1 WHERE c1 = (SELECT c1 FROM t2) FOR UPDATE;
```

テーブル `t2` の行をロックするには、サブクエリーにロック読取り句を追加します:

```
SELECT * FROM t1 WHERE c1 = (SELECT c1 FROM t2 FOR UPDATE) FOR UPDATE;
```

読取りロックの例

`child` テーブルに新しい行を挿入し、子の行が `parent` テーブル内に親の行を持っていることを確認すると仮定します。アプリケーションコードを使用して、この操作シーケンス全体の参照整合性を確保できます。

まず、一貫性読み取りを使用して、`PARENT` テーブルでクエリーを実行し、親の行が存在することを確認します。`CHILD` テーブルに子の行を安全に挿入できますか。気付かないうちに、その他の一部のセッションで、`SELECT` と `INSERT` との間に親の行が削除された可能性もあるため、できません。

この潜在的な問題を回避するには、`FOR SHARE` を使用して `SELECT` を実行します:

```
SELECT * FROM parent WHERE NAME = 'Jones' FOR SHARE;
```

`FOR SHARE` クエリーで親'`Jones`'が返された後、子レコードを `CHILD` テーブルに安全に追加し、トランザクションをコミットできます。 `PARENT` テーブルの該当する行で排他ロックを取得しようとするトランザクションは、終了するまで、つまりすべてのテーブルのデータが一貫性のある状態になるまで待機します。

もう 1 つの例では、`CHILD` テーブルに追加された各子に一意の識別子を割り当てる際に使用される `CHILD_CODES` テーブル内の整数カウンタフィールドを検討します。一貫性読み取りまたは共有モード読み取りを使用すると、データベースの 2 人のユーザーが同じカウンタ値を参照する可能性があり、2 つのトランザクションが同じ識別子を持つ行を `CHILD` テーブルに追加しようすると、重複キーのエラーが発生するため、カウンタの現在の値を読み取る際には使用しないでください。

ここでは、2 人のユーザーがカウンタを同時に読み取った場合、カウンタを更新しようすると少なくとも 1 人のユーザーがデッドロックになるため、`FOR SHARE` は適切なソリューションではありません。

カウンタの読み取りおよび増分を実装するには、まず `FOR UPDATE` を使用してカウンタのロック読み取りを実行してから、カウンタを増分します。例:

```
SELECT counter_field FROM child_codes FOR UPDATE;  
UPDATE child_codes SET counter_field = counter_field + 1;
```

`SELECT ... FOR UPDATE` は使用可能な最新データを読み取り、読み取られる各行上に排他ロックを設定します。したがって、検索された SQL `UPDATE` によって行上に設定される場合と同じロックが設定されます。

前述の説明は、単に `SELECT ... FOR UPDATE` がどのように機能するのかを示した例です。MySQL では、テーブルへの単一アクセスを使用するだけで、一意の識別子を生成する特定のタスクを実現できます。

```
UPDATE child_codes SET counter_field = LAST_INSERT_ID(counter_field + 1);  
SELECT LAST_INSERT_ID();
```

この `SELECT` ステートメントは、単に (現在の接続に固有の) 識別子情報を取得するだけです。どのテーブルにもアクセスしません。

NOWAIT および SKIP LOCKED による読取り同時実行性のロック

行がトランザクションによってロックされている場合、同じロックされた行をリクエストする `SELECT ... FOR UPDATE` または `SELECT ... FOR SHARE` トランザクションは、ブロックしているトランザクションが行ロックを解放するまで待機する必要があります。この動作により、トランザクションは、他のトランザクションによる更新をクエリーする行を更新または削除できなくなります。ただし、リクエストされた行がロックされたときにすぐにクエリーを戻す場合、またはロックされた行を結果セットから除外できる場合は、行ロックの解放を待機する必要はありません。

他のトランザクションによる行ロックの解放を待機しないように、`SELECT ... FOR UPDATE` または `SELECT ... FOR SHARE` のロック読取りステートメントで `NOWAIT` および `SKIP LOCKED` オプションを使用できます。

- `NOWAIT`

`NOWAIT` を使用するロック読取りは、行ロックの取得を待機しません。クエリーはただちに実行され、リクエストされた行がロックされている場合はエラーで失敗します。

- SKIP LOCKED

SKIP LOCKED を使用するロック読取りは、行ロックの取得を待機しません。クエリーはただちに実行され、ロックされた行が結果セットから削除されます。

注記

ロックされた行をスキップするクエリーは、データの一貫性のないビューを返します。したがって、SKIP LOCKED は一般的なトランザクション作業には適していません。ただし、複数のセッションが同じキューに類似したテーブルにアクセスする場合、ロックの競合を回避するために使用できます。

NOWAIT および SKIP LOCKED は、行レベルロックにのみ適用されます。

NOWAIT または SKIP LOCKED を使用するステートメントは、ステートメントベースのレプリケーションでは安全ではありません。

次の例は、NOWAIT および SKIP LOCKED を示しています。セッション 1 は、単一のレコードで行ロックを取得するトランザクションを開始します。セッション 2 は、NOWAIT オプションを使用して、同じレコードに対するロック読取りを試行します。リクエストされた行はセッション 1 によってロックされているため、ロック読取りはすぐにエラーとともに返されます。セッション 3 では、SKIP LOCKED で読み取られたロックは、セッション 1 でロックされている行を除いて、リクエストされた行を返します。

```
# Session 1:
mysql> CREATE TABLE t (i INT, PRIMARY KEY (i)) ENGINE = InnoDB;
mysql> INSERT INTO t (i) VALUES(1),(2),(3);
mysql> START TRANSACTION;
mysql> SELECT * FROM t WHERE i = 2 FOR UPDATE;
+---+
|i|
+---+
|2|
+---+

# Session 2:
mysql> START TRANSACTION;
mysql> SELECT * FROM t WHERE i = 2 FOR UPDATE NOWAIT;
ERROR 3572 (HY000): Do not wait for lock.

# Session 3:
mysql> START TRANSACTION;
mysql> SELECT * FROM t FOR UPDATE SKIP LOCKED;
+---+
|i|
+---+
|1|
|3|
+---+
```

15.7.3 InnoDB のさまざまな SQL ステートメントで設定されたロック

一般に、ロック読取り、UPDATE、または DELETE では、SQL ステートメントの処理時にスキャンされるすべてのインデックスレコード上に、レコードロックが設定されます。行を除外する WHERE 条件がステートメント内に存在するかどうかは、関係ありません。InnoDB には正確な WHERE 条件が記憶されませんが、スキャンされたインデックスの範囲は認識されます。通常、ロックはレコードの直前にある「ギャップ」への挿入もブロックするネクストキーロックです。ただし、ギャップロックは明示的に無効にすることができます。これにより、ネクストキーロックが使用されなくなります。詳細は、セクション 15.7.1 「InnoDB ロック」を参照してください。トランザクション分離レベルによって、どのロックが設定されるのかも影響を受けます。セクション 15.7.2.1 「トランザクション分離レベル」を参照してください。

検索でセカンダリインデックスが使用され、設定されるインデックスレコードのロックが排他的である場合、InnoDB は対応するクラスタ化されたインデックスレコードを取得し、それらにロックを設定することも行います。

ステートメントに適したインデックスがなく、MySQL がステートメントを処理するためにテーブル全体をスキャンする必要がある場合は、テーブルのすべての行がロックされます。その結果、そのテーブルへのほかのユーザーによるすべての挿入がブロックされます。クエリーで不必要に複数の行がスキャンされないように、適切なインデックスを作成することが重要です。

InnoDB は、次のように特定のロックタイプを設定します。

- **SELECT ... FROM** は一貫性読み取りであり、データベースのスナップショットを読み取り、トランザクションの分離レベルが **SERIALIZABLE** に設定されなければロックを設定しません。 **SERIALIZABLE** レベルの場合、検索で見つかったインデックスレコード上に共有ネクストキーロックが設定されます。ただし、一意の行を検索するために一意のインデックスを使用して行をロックするステートメントには、インデックスレコードのロックのみが必要です。
- 一意インデックスを使用する **SELECT ... FOR UPDATE** および **SELECT ... FOR SHARE** ステートメントは、スキャンされた行のロックを取得し、結果セットに含まれない行のロックを解除します (たとえば、**WHERE** 句で指定された基準を満たさない場合)。ただし場合によっては、クエリーの実行中に結果行とその元のソースとの関係が失われたために、行のロックがすぐに解除されない可能性もあります。たとえば **UNION** では、スキャン (およびロック) されたテーブル内の行が、結果セットに含める対象となるかどうかの評価前に、一時テーブルに挿入される可能性があります。この状況では、一時テーブル内の行と元のテーブル内の行との関係は失われているため、クエリー実行が終了するまで後者の行のロックは解除されません。
- **locking reads** (**SELECT** と **FOR UPDATE** または **FOR SHARE**)、**UPDATE** および **DELETE** ステートメントの場合、実行されるロックは、ステートメントが一意の検索条件を持つ一意のインデックスを使用するか、範囲タイプの検索条件を使用するかによって異なります。
 - 一意の検索条件を使用した一意のインデックスの場合、InnoDB は見つかったインデックスレコードのみをロックし、その前にある**ギャップ**はロックしません。
 - 他の検索条件および一意でないインデックスの場合、InnoDB は、**gap locks** または **next-key locks** を使用してスキャンされたインデックス範囲をロックし、他のセッションによる挿入を範囲の対象となるギャップにブロックします。ギャップロックおよびネクストキーロックについては、[セクション15.7.1「InnoDB ロック」](#)を参照してください。
- 検索で検出されたインデックスレコードの場合、**SELECT ... FOR UPDATE** は、他のセッションによる **SELECT ... FOR SHARE** の実行または特定のトランザクション分離レベルでの読み取りをブロックします。一貫性読み取りでは、読み取られたビュー内に存在するレコードに設定されたロックはすべて無視されます。
- **UPDATE ... WHERE ...** は、検索で見つかったすべてのレコード上に排他ネクストキーロックを設定します。ただし、一意の行を検索するために一意のインデックスを使用して行をロックするステートメントには、インデックスレコードのロックのみが必要です。
- **UPDATE** がクラスタ化されたインデックスレコードを変更すると、影響を受けるセカンダリインデックスレコードが暗黙的にロックされます。 **UPDATE** 操作では、新しいセカンダリインデックスレコードを挿入する前に重複チェックスキャンを実行するとき、および新しいセカンダリインデックスレコードを挿入するとき、影響を受けるセカンダリインデックスレコードの共有ロックも取得されます。
- **DELETE FROM ... WHERE ...** は、検索で見つかったすべてのレコード上に排他ネクストキーロックを設定します。ただし、一意の行を検索するために一意のインデックスを使用して行をロックするステートメントには、インデックスレコードのロックのみが必要です。
- **INSERT** は、挿入される行に排他ロックを設定します。このロックは、ネクストキーロックではなくインデックスレコードロックである (つまり、ギャップロックが存在しない) ため、ほかのセッションが挿入された行の前にあるギャップに挿入することは回避されません。

行を挿入する前に、挿入意図ギャップロックと呼ばれるギャップロックのタイプが設定されます。このロックは、同じインデックスギャップに挿入する複数のトランザクションは、そのギャップ内の同じ場所に挿入しなければ相互に待機する必要がないように、意図的に挿入することを示しています。値が 4 と 7 のインデックスレコードが存在すると仮定します。それぞれ値 5 と 6 の挿入を試みる別々のトランザクションは、挿入される行の排他ロックを取得する前に挿入意図ロックを使用して、4 と 7 の間にあるギャップをロックしますが、行の競合が発生しないため相互にブロックされません。

重複キーエラーが発生すると、重複インデックスレコード上の共有ロックが設定されます。複数のセッションが同じ行を挿入しようとしているときに、別のセッションがすでに排他ロックを取得していた場合は、このように共有ロックを使用することでデッドロックが発生する可能性があります。これは、別のセッションがその行を削除した場合に発生する可能性があります。InnoDB テーブル `t1` の構造が次のようになっています。

```
CREATE TABLE t1 (i INT, PRIMARY KEY (i)) ENGINE = InnoDB;
```

次に、3つのセッションが次の処理を順番に実行するものとします。

セッション 1:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

セッション 2:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

セッション 3:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

セッション 1:

```
ROLLBACK;
```

セッション 1 による最初の処理では、行の排他ロックが取得されます。セッション 2 と 3 の処理ではどちらも重複キーエラーが発生し、どちらのセッションも行の共有ロックをリクエストします。セッション 1 はロールバック時に行の排他ロックを解放し、キュー内のセッション 2 と 3 の共有ロックリクエストが付与されます。この時点でセッション 2 と 3 でデッドロックが発生します。どちらも他方が保持している共有ロックのために、行の排他ロックを取得できません。

キー値が 1 の行がテーブルに含まれている場合も似たような状況が発生し、3つのセッションが次の処理を順番に実行します。

セッション 1:

```
START TRANSACTION;
DELETE FROM t1 WHERE i = 1;
```

セッション 2:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

セッション 3:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

セッション 1:

```
COMMIT;
```

セッション 1 による最初の処理では、行の排他ロックが取得されます。セッション 2 と 3 の処理ではどちらも重複キーエラーが発生し、どちらのセッションも行の共有ロックをリクエストします。セッション 1 はコミット時に行の排他ロックを解放し、キュー内のセッション 2 と 3 の共有ロックリクエストが付与されます。この時点でセッション 2 と 3 でデッドロックが発生します。どちらも他方が保持している共有ロックのために、行の排他ロックを取得できません。

- **INSERT ... ON DUPLICATE KEY UPDATE** は、重複キーエラーが発生したときに更新される行に共有ロックではなく排他ロックが設定されるという点で、単純な **INSERT** と異なります。重複する主キー値に対して排他的なイン

デックスレコードロックが取得されます。重複する一意キー値に対して排他的なネクストキーロックが取得されません。

- **REPLACE** は、一意のキーが競合していなければ、**INSERT** と同様に動作します。それ以外の場合は、置換される行に排他ネクストキーロックが配置されます。
- **INSERT INTO T SELECT ... FROM S WHERE ...** では、**T** に挿入された各行に排他的インデックスレコードロック (ギャップロックなし) が設定されます。トランザクション分離レベルが **READ COMMITTED** の場合、**InnoDB** は **S** で一貫性読取り (ロックなし) として検索を実行します。それ以外の場合、**InnoDB** は **S** から取得した行に共有ネクストキーロックを設定します。後者の場合、**InnoDB** はロックを設定する必要があります: ステートメントベースのバイナリログを使用したロールフォワードリカバリ中は、すべての SQL ステートメントを、最初に行われたのとまったく同じ方法で実行する必要があります。

CREATE TABLE ... SELECT ... は、**INSERT ... SELECT** の場合と同様に、**SELECT** を共有ネクストキーロックを使用して実行するか、一貫性読み取りとして実行します。

構造文 **REPLACE INTO t SELECT ... FROM s WHERE ...** または **UPDATE t ... WHERE col IN (SELECT ... FROM s ...)** で **SELECT** が使用されると、**InnoDB** はテーブル **s** の行に共有ネクストキーロックを設定します。

- **InnoDB** は、テーブルで以前に指定された **AUTO_INCREMENT** カラムを初期化する際に、**AUTO_INCREMENT** カラムに関連付けられたインデックスの最後に排他ロックを設定します。

innodb_autoinc_lock_mode=0 では、**InnoDB** は特別な **AUTO-INC** テーブルロックモードを使用します。このモードでは、自動増分カウンタにアクセスしながら、ロックが取得され、(トランザクション全体の最後ではなく) 現在の SQL ステートメントの最後まで保持されます。**AUTO-INC** テーブルロックが保持されている間は、ほかのクライアントはそのテーブルに挿入できません。**innodb_autoinc_lock_mode=1** を使用した「一括挿入」でも同じ動作が発生します。テーブルレベルの **AUTO-INC** ロックは、**innodb_autoinc_lock_mode=2** では使用されません。詳細は、[セクション15.6.1.6「InnoDBでのAUTO_INCREMENT処理」](#)を参照してください。

InnoDB は、ロックを設定せずに、事前に初期化された **AUTO_INCREMENT** カラムの値をフェッチします。

- **FOREIGN KEY** 制約がテーブル上で定義されている場合は、制約条件をチェックする必要がある挿入、更新、または削除が行われると、制約をチェックするために、参照されるレコード上に共有レコードレベルロックが設定されます。**InnoDB** は、制約が失敗する場合に備えて、これらのロックの設定も行います。
- **LOCK TABLES** はテーブルロックを設定しますが、これらのロックを設定する **InnoDB** レイヤーよりも上位の MySQL レイヤーです。**InnoDB** は、**innodb_table_locks = 1** (デフォルト) かつ **autocommit = 0** の場合にテーブルロックを認識し、**InnoDB** よりも上位の MySQL レイヤーは、行レベルロックを識別します。

それ以外の場合は、**InnoDB** の自動デッドロック検出では、このようなテーブルロックが関与するデッドロックを検出できません。また、この場合には上位の MySQL レイヤーは行レベルロックを識別しないため、現在別のセッションが行レベルロックを保持しているテーブル上でテーブルロックを取得できません。ただし、[セクション15.7.5.2「デッドロック検出」](#)で説明したように、これによりトランザクションの完全性が危険にさらされることはありません。

- **innodb_table_locks=1** (デフォルト) の場合、**LOCK TABLES** で各テーブル上に 2 つのロックが取得されます。MySQL レイヤーでのテーブルロックに加えて、**InnoDB** テーブルロックも取得されます。バージョン 4.1.2 よりも前の MySQL では、**InnoDB** テーブルロックが取得されませんでした。この古い動作は、**innodb_table_locks=0** を設定すれば選択できます。**InnoDB** テーブルロックが取得されない場合は、テーブルの一部のレコードがほかのトランザクションによってロックされなくても、**LOCK TABLES** が完了します。

MySQL 8.0 では、**LOCK TABLES ... WRITE** を使用して明示的にロックされたテーブルには、**innodb_table_locks=0** が無効です。**LOCK TABLES ... WRITE** で暗黙的に (たとえば、トリガーを使用して)、または **LOCK TABLES ... READ** によって、読み取りまたは書き込み用にロックされたテーブルには有効です。

- トランザクションで保持されているすべての **InnoDB** ロックは、トランザクションがコミットまたは中止されると解放されます。したがって、**autocommit=1** モードの **InnoDB** テーブル上で **LOCK TABLES** を呼び出しても、取得された **InnoDB** テーブルロックはすぐに解放されてしまうため、まったく意味がありません。
- **LOCK TABLES** では暗黙的な **COMMIT** および **UNLOCK TABLES** が実行されるため、トランザクションの実行中に追加のテーブルをロックできません。

15.7.4 ファントム行

同じクエリーでさまざまな時間にさまざまな行のセットが生成されると、いわゆるファントムの問題がトランザクション内で発生します。たとえば、`SELECT` が 2 回実行されたが、1 回目には返されなかった行が 2 回目には返された場合、その行が「ファントム」行です。

`child` テーブルの `id` カラム上にインデックスがあり、識別子の値が 100 よりも大きいすべての行をテーブルから読み取り、選択された行の一部のカラムをあとで更新するという意図でロックすると仮定します。

```
SELECT * FROM child WHERE id > 100 FOR UPDATE;
```

クエリーでは、`id` が 100 よりも大きい最初のレコードからインデックスがスキャンされます。このテーブルには `id` の値が 90 と 102 の行が格納されているものとします。スキャン範囲内のインデックスレコード上に設定されたロックによって、ギャップ (この場合のギャップは 90 から 102 まで) への挿入がロックアウトされていない場合は、別のセッションが `id` が 101 の新しい行をそのテーブルに挿入できます。同じトランザクション内で同じ `SELECT` を実行すると、クエリーから返された結果セット内に、`id` が 101 の新しい行 (「ファントム」) が含まれています。一連の行をデータ項目とみなせば、この新しいファントムの子は、「トランザクション中は読み取られるデータが変更されないようにトランザクションを実行できるべきである」というトランザクションの分離原則に違反しています。

ファントムの発生を回避できるように、InnoDB では通常、インデックス行ロックとギャップロックを組み合わせたネクストキーロックと呼ばれるアルゴリズムが使用されます。InnoDB は、テーブルインデックスを検索またはスキャンするときに、生成されたインデックスレコード上に共有ロックまたは排他ロックを設定するという方法で、行レベルロックを実行します。したがって、行レベルロックは、実際にはインデックスレコードロックです。さらに、あるインデックスレコードに対するネクストキーロックによって、そのインデックスレコードの前の「ギャップ」も影響を受けます。つまり、ネクストキーロックは、インデックスレコードロックと、そのインデックスレコードの前のギャップに対するギャップロックとを組み合わせたものです。あるセッションがインデックス内のレコード `R` 上に共有ロックまたは排他ロックを持っている場合は、別のセッションがインデックスの順番で `R` の直前にあるギャップに新しいインデックスレコードを挿入できません。

InnoDB はインデックスをスキャンするときに、インデックス内の最後のレコードのあとのギャップをロックすることもできます。前述の例では、まさにそれが行われています。`id` が 100 よりも大きいテーブルへの挿入が回避されるように、InnoDB で設定されたロックには、`id` 値 102 のあとのギャップに対するロックが含まれています。

ネクストキーロックを使用すると、アプリケーションに一貫性チェックを実装できます。共有モードでデータを読み取るときに、挿入される行の重複が見られなければ、行を安全に挿入でき、読み取り中に後続の行に設定されたネクストキーロックによって、任意のユーザーによる重複行の挿入が回避されることを確認できます。したがって、ネクストキーロックを使用すれば、テーブル内に存在しないものも「ロック」できます。

ギャップロックは、[セクション 15.7.1 「InnoDB ロック」](#) で説明した方法で無効にすることができます。ギャップロックが無効になっていると、ほかのセッションが新しい行をギャップに挿入できるため、ファントムの問題が発生する可能性があります。

15.7.5 InnoDB のデッドロック

デッドロックとは、それぞれが他の必要なロックを保持しているために、異なるトランザクションを続行できない状況です。両方のトランザクションがリソースが使用可能になるのを待機しているため、保持しているロックは解放されません。

デッドロックは、(`UPDATE` や `SELECT ... FOR UPDATE` などのステートメントを使用して) 複数のテーブルの行をトランザクションがロックするときに発生する可能性がありますが、逆の順序で発生します。デッドロックは、このようなステートメントがインデックスレコードとギャップの範囲をロックし、各トランザクションが一部のロックを取得するけれども、タイミングの問題によりほかを取得しない場合にも発生することがあります。デッドロックの例については、[セクション 15.7.5.1 「InnoDB デッドロックの例」](#) を参照してください。

デッドロックの可能性を減らすには、`LOCK TABLES` ステートメントではなくトランザクションを使用します。データを挿入または更新するトランザクションのサイズは、長期間オープンしたままにしないでください。異なるトランザクションが複数のテーブルまたは行の範囲を更新する場合は、各トランザクションで同じ順序 (`SELECT ... FOR UPDATE` など) を使用し、`SELECT ... FOR UPDATE` ステートメントおよび `UPDATE ... WHERE` ステートメントで使用されるカラムにインデックスを作成します。デッドロックの可能性は、分離レベルに影響を受けません。分離レベルは読み取り操作の動作を変更し、デッドロックは書き込み操作のために発生するからです。デッドロック状態からの回避およびリカバリの詳細は、[セクション 15.7.5.3 「デッドロックを最小化および処理する方法」](#) を参照してください。

デッドロック検出が有効 (デフォルト) でデッドロックが発生した場合、InnoDB は条件を検出し、いずれかのトランザクション (被害者) をロールバックします。innodb_deadlock_detect 構成オプションを使用してデッドロック検出が無効になっている場合、InnoDB は、デッドロックの場合にトランザクションをロールバックするために innodb_lock_wait_timeout 設定に依存します。したがって、アプリケーションロジックが正しい場合でも、トランザクションを再試行する必要があるケースを処理する必要があります。InnoDB ユーザートランザクションの最後のデッドロックを表示するには、SHOW ENGINE INNODB STATUS コマンドを使用します。デッドロックが頻繁に発生して、トランザクション構造やアプリケーションエラー処理に問題があるらしいと思われる場合は、mysqld エラーログにすべてのデッドロックに関する情報を出力するために、innodb_print_all_deadlocks 設定を有効にした状態で実行してください。デッドロックが自動的に検出および処理される方法の詳細は、セクション15.7.5.2「デッドロック検出」を参照してください。

15.7.5.1 InnoDB デッドロックの例

次の例は、ロックリクエストによってデッドロックが発生したときに、どのようにエラーが発生するのかを示しています。この例には、A と B の 2 つのクライアントが登場します。

最初に、クライアント A が行を 1 つ含むテーブルを作成し、トランザクションを開始します。トランザクション内で、A は共有モードで選択した行で S ロックを取得します。

```
mysql> CREATE TABLE t (i INT) ENGINE = InnoDB;
Query OK, 0 rows affected (1.07 sec)

mysql> INSERT INTO t (i) VALUES(1);
Query OK, 1 row affected (0.09 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM t WHERE i = 1 FOR SHARE;
+-----+
| i |
+-----+
| 1 |
+-----+
```

次に、クライアント B がトランザクションを開始し、テーブルから行を削除しようとしています。

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> DELETE FROM t WHERE i = 1;
```

削除操作を行うには、X ロックが必要です。クライアント A が保持している S ロックとの互換性がないために、ロックを付与できません。そのため、リクエストはその行のロックリクエストのキューに入れられ、クライアント B はブロックされます。

最後に、クライアント A もテーブルから行を削除しようとしています。

```
mysql> DELETE FROM t WHERE i = 1;
ERROR 1213 (40001): Deadlock found when trying to get lock;
try restarting transaction
```

クライアント A は行を削除するために X ロックが必要であるため、ここでデッドロックが発生します。ただし、クライアント B はすでに X ロックに対するリクエストを持っていて、クライアント A がその S ロックを解放するまで待機しているため、そのロックリクエストを付与することはできません。B による X ロックに対する以前のリクエストが原因で、A が保持している S ロックを X ロックにアップグレードすることもできません。その結果、InnoDB はクライアントのいずれかに対してエラーを生成し、そのロックを解放します。クライアントは、次のエラーを返します。

```
ERROR 1213 (40001): Deadlock found when trying to get lock;
try restarting transaction
```

この時点で、ほかのクライアントに対するロックリクエストを付与できるようになり、テーブルから行が削除されません。

15.7.5.2 デッドロック検出

[deadlock detection](#) が有効な場合 (デフォルト)、InnoDB はトランザクション [deadlocks](#) を自動的に検出し、デッドロックを解消するためにトランザクションをロールバックします。InnoDB は、小さいトランザクションを選択してロールバックしようと試みます。トランザクションのサイズは、挿入、更新、または削除された行数によって決定されます。

InnoDB は、`innodb_table_locks = 1` (デフォルト) かつ `autocommit = 0` の場合にテーブルロックを認識し、それよりも上位の MySQL レイヤーは、行レベルロックを識別します。それ以外の場合、InnoDB は、MySQL `LOCK TABLES` ステートメントで設定されたテーブルロックまたは InnoDB 以外のストレージエンジンで設定されたロックが関連しているデッドロックを検出できません。このような状況を解決するには、`innodb_lock_wait_timeout` システム変数の値を設定します。

InnoDB Monitor 出力の `LATEST DETECTED DEADLOCK` セクションに「グラフのロックテーブルの検索が深すぎるが長すぎます。次のトランザクションをロールバック」というメッセージが含まれている場合、これは、キャンセル待ちリストのトランザクション数が 200 の制限に達したことを示します。200 個のトランザクションを超える待機リストはデッドロックとして処理され、待機リストをチェックしようとするトランザクションはロールバックされます。ロックスレッドが待機リストのトランザクションが所有する 1,000,000 を超えるロックを参照する必要がある場合も、同じエラーが発生する可能性があります。

デッドロックを回避するためにデータベース操作を編成する方法については、[セクション 15.7.5 「InnoDB のデッドロック」](#) を参照してください。

デッドロック検出の無効化

同時実行性の高いシステムでは、多数のスレッドが同じロックを待機している場合、デッドロック検出によって速度が低下する可能性があります。デッドロック検出を無効にし、デッドロック発生時のトランザクションロールバックの `innodb_lock_wait_timeout` 設定に依存する方が効率的な場合があります。デッドロック検出は、`innodb_deadlock_detect` 構成オプションを使用して無効にできます。

15.7.5.3 デッドロックを最小化および処理する方法

このセクションは、[セクション 15.7.5.2 「デッドロック検出」](#) に示したデッドロックに関する概念情報に基づいています。ここでは、デッドロックが最小限になるようにデータベース操作を編成する方法、およびアプリケーションで必要となる後続のエラー処理について説明します。

デッドロック は、トランザクションデータベースの古典的な問題ですが、特定のトランザクションをまったく実行できないほど発生頻度が高くなければ、危険ではありません。通常は、デッドロックが発生したためにトランザクションがロールバックされた場合に、それを再発行できる準備が常にできているようにアプリケーションを作成する必要があります。

InnoDB では自動行レベルロックが使用されます。単一の行を挿入または削除するだけのトランザクションの場合でも、デッドロックが発生する可能性があります。その原因は、これらの操作が実際には「原子的」でないためです。これらの操作では自動的に、挿入または削除される行のインデックスレコード (複数の可能性あり) にロックが設定されます。

次の方法を使用すれば、デッドロックに対処し、発生の可能性を減らすことができます。

- いつでも、`SHOW ENGINE INNODB STATUS` コマンドを発行して、最近のデッドロックの原因を特定してください。これは、デッドロックが回避されるようにアプリケーションを調整する際に役立ちます。
- デッドロック警告が頻繁に発生することが懸念される場合は、`innodb_print_all_deadlocks` 構成オプションを有効にして、より広範なデバッグ情報を収集します。MySQL の [エラーログ](#) には、最近のデッドロックだけでなく、各デッドロックに関する情報が記録されます。デバッグが完了したら、このオプションを無効にします。
- デッドロックが原因でトランザクションに失敗した場合に、そのトランザクションを再発行できるように常に準備しておきます。デッドロックは危険ではありません。再度試してください。
- トランザクションが競合する可能性を低くするために、トランザクションのサイズを小さく、期間を短く保ってください。
- トランザクションが競合する可能性を低くするために、関連する一連の変更を行なった直後にトランザクションをコミットしてください。特に、コミットされていないトランザクションを含むインタラクティブな `mysql` セッションは、長時間開いたままにしないでください。

- **locking reads** (`SELECT ... FOR UPDATE` または `SELECT ... FOR SHARE`) を使用する場合は、**READ COMMITTED** などのより低い分離レベルを使用してみてください。
- トランザクション内の複数のテーブルを変更する場合や、同じテーブル内のさまざまな行のセットを変更する場合は、毎回、これらの操作を一貫性のある順序で実行してください。その結果、トランザクションで明示的に定義されたキューが生成され、デッドロックは発生しません。たとえば、さまざまな場所で同様の **INSERT**、**UPDATE**、および **DELETE** ステートメントのシーケンスを複数回コーディングするのではなく、データベース操作をアプリケーション内の関数に編成したり、ストアルーチンを呼び出したりします。
- テーブルに適切なインデックスを追加してください。これにより、クエリーでスキャンする必要があるインデックスレコード数が減少するため、ロックの設定も減少します。MySQL サーバーがクエリーに最適であるとみなすインデックスを特定するために、**EXPLAIN SELECT** を使用してください。
- ロックの使用を減らしてください。**SELECT** が古いスナップショットからデータを返すことを許可できる場合は、**FOR UPDATE** または **FOR SHARE** 句を追加しないでください。同じトランザクション内の各一貫性読み取りでは、独自の新しいスナップショットから読み取られるため、**READ COMMITTED** 分離レベルを使用することが適切な方法です。
- ほかに方法がなければ、テーブルレベルロックを使用してトランザクションを直列化してください。**InnoDB** テーブルなどのトランザクションテーブルで **LOCK TABLES** を使用する正しい方法は、(**START TRANSACTION** ではなく) **SET autocommit = 0** でトランザクションを開始し、そのあと **LOCK TABLES** を実行し、**UNLOCK TABLES** を呼び出す前にそのトランザクションを明示的にコミットすることです。たとえば、テーブル **t1** に書き込み、テーブル **t2** から読み取る必要がある場合は、次のように実行できます。

```
SET autocommit=0;
LOCK TABLES t1 WRITE, t2 READ, ...;
... do something with tables t1 and t2 here ...
COMMIT;
UNLOCK TABLES;
```

テーブルレベルロックを使用すると、テーブルへの並列更新が抑制されるため、デッドロックが回避されますが、負荷の高いシステムで応答性が低くなるという犠牲が伴います。

- トランザクションを直列化する別の方法は、単一行だけを含む補助「セマフォ」テーブルを作成することです。ほかのテーブルにアクセスする前に、各トランザクションでその行を更新してください。これにより、すべてのトランザクションが直列方式で発生します。直列化ロックは行レベルロックであるため、この場合、**InnoDB** のインスタントデッドロック検出アルゴリズムも機能することに注意してください。MySQL のテーブルレベルロックを使用してデッドロックを解決するには、タイムアウト方式を使用する必要があります。

15.7.6 トランザクションスケジューリング

InnoDB では、競合対応トランザクションスケジューリング (CATS) アルゴリズムを使用して、ロックを待機しているトランザクションに優先順位を付けます。複数のトランザクションが同じオブジェクトのロックを待機している場合、CATS アルゴリズムは最初にロックを受信するトランザクションを決定します。

CATS アルゴリズムは、トランザクションがブロックするトランザクションの数に基づいて計算されるスケジューリング加重を割り当てることで、待機中のトランザクションに優先順位を付けます。たとえば、2つのトランザクションが同じオブジェクトのロックを待機している場合、ほとんどのトランザクションをブロックするトランザクションには、より大きなスケジューリング加重が割り当てられます。重みが等しい場合、最も長い待機トランザクションに優先度が与えられます。

注記

MySQL 8.0.20 より前の **InnoDB** では、トランザクションのスケジュールに先入れ先出し (FIFO) アルゴリズムも使用され、CATS アルゴリズムはロック競合が多い場合にのみ使用されていました。MySQL 8.0.20 の CATS アルゴリズムの拡張により、FIFO アルゴリズムが冗長になり、削除が許可されました。FIFO アルゴリズムによって以前に実行されたトランザクションスケジューリングは、MySQL 8.0.20 の時点で CATS アルゴリズムによって実行されます。場合によっては、この変更がトランザクションにロックが付与される順序に影響することがあります。

INFORMATION_SCHEMA.INNODB_TRX テーブルの **TRX_SCHEDULE_WEIGHT** カラムをクエリーすることで、トランザクションスケジューリング加重を表示できます。重みは、待機中のトランザクションに対してのみ計算されま

す。待機中のトランザクションは、`TRX_STATE` カラムでレポートされる `LOCK WAIT` トランザクション実行状態のトランザクションです。ロックを待機していないトランザクションは、`NULL` の `TRX_SCHEDULE_WEIGHT` 値を報告します。

`INNODB_METRICS` カウンタは、コードレベルのトランザクションスケジューリングイベントを監視するために用意されています。`INNODB_METRICS` カウンタの使用の詳細は、[セクション15.15.6「InnoDB INFORMATION_SCHEMA メトリックテーブル」](#) を参照してください。

- `lock_rec_release_attempts`

レコードロックの解放の試行回数。単一の構造にゼロ個以上のレコードロックが存在する可能性があるため、単一の試行によってゼロ個以上のレコードロックが解放される場合があります。

- `lock_rec_grant_attempts`

レコードロックの付与を試行する回数。単一回試行すると、ゼロ個以上のレコードロックが付与される場合があります。

- `lock_schedule_refreshes`

スケジュール済トランザクションの重みを更新するために待機グラフが分析された回数。

15.8 InnoDB の構成

このセクションでは、`InnoDB` の初期化、起動、および `InnoDB` ストレージエンジンのさまざまなコンポーネントと機能の構成情報と手順について説明します。`InnoDB` テーブルのデータベース操作の最適化の詳細は、[セクション 8.5「InnoDB テーブルの最適化」](#) を参照してください。

15.8.1 InnoDB の起動構成

`InnoDB` 構成に関する最初の決定には、データファイル、ログファイル、ページサイズおよびメモリーバッファの構成が含まれます。`InnoDB` インスタンスを作成する前に、データファイル、ログファイルおよびページサイズの構成を定義することをお勧めします。`InnoDB` インスタンスの作成後にデータファイルまたはログファイルの構成を変更するには、簡単でない手順が必要になる場合があります、ページサイズを定義できるのは、`InnoDB` インスタンスが最初に初期化されたときのみです。

これらのトピックに加えて、このセクションでは、構成ファイルでの `InnoDB` オプションの指定、`InnoDB` 初期化情報の表示、および記憶域に関する重要な考慮事項について説明します。

- [MySQL 構成ファイルでのオプションの指定](#)
- [InnoDB 初期化情報の表示](#)
- [記憶域に関する重要な考慮事項](#)
- [システムテーブルスペースデータファイル構成](#)
- [InnoDB 二重書込みバッファファイルの構成](#)
- [redo ログファイル構成](#)
- [undo テーブルスペースの構成](#)
- [グローバル一時テーブルスペース構成](#)
- [セッション一時テーブルスペース構成](#)
- [ページサイズ構成](#)
- [メモリー構成](#)

MySQL 構成ファイルでのオプションの指定

MySQL はデータファイル、ログファイルおよびページサイズの構成設定を使用して `InnoDB` インスタンスを初期化するため、`InnoDB` を初めて初期化する前に、MySQL が起動時に読み取る構成ファイルでこれらの設定を定義すること

をお勧めします。InnoDB は MySQL サーバーの起動時に初期化され、InnoDB の最初の初期化は通常、MySQL サーバーの初回起動時に行われます。

サーバーの起動時に読み取られる任意のオプションファイルの `[mysqld]` グループ内に、InnoDB オプションを配置できます。MySQL オプションファイルの場所は、[セクション4.2.2.2「オプションファイルの使用」](#) で説明されています。

`mysqld` が特定のファイル (および `mysqld-auto.cnf`) からのみオプションを読み取るようにするには、サーバーの起動時にコマンド行で最初のオプションとして `--defaults-file` オプションを使用します:

```
mysqld --defaults-file=path_to_configuration_file
```

InnoDB 初期化情報の表示

起動時に InnoDB の初期化情報を表示するには、コマンドプロンプトから `mysqld` を起動します。コマンドプロンプトから `mysqld` を起動すると、初期化情報がコンソールに出力されます。

たとえば、Windows では、`mysqld` が `C:\Program Files\MySQL\MySQL Server 8.0\bin` にある場合、次のように MySQL サーバーを起動します:

```
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqld" --console
```

Unix に似たシステムでは、`mysqld` は MySQL インストールの `bin` ディレクトリにあります:

```
shell> bin/mysqld --user=mysql &
```

サーバー出力をコンソールに送信しない場合は、起動後にエラーログを確認して、起動プロセス中に InnoDB によって出力された初期化情報を確認します。

他の方法を使用した MySQL の起動の詳細は、[セクション2.10.5「MySQL を自動的に起動および停止する」](#) を参照してください。

注記

InnoDB では、起動時にすべてのユーザーテーブルおよび関連データファイルが開かれるわけではありません。ただし、InnoDB では、データディクショナリで参照されるテーブルスペースファイル (`*.ibd` ファイル) の存在がチェックされます。テーブルスペースファイルが見つからない場合、InnoDB はエラーをログに記録し、起動順序を続行します。redo ログで参照されるテーブルスペースファイルは、redo アプリケーションのクラッシュリカバリ中に開くことができます。

記憶域に関する重要な考慮事項

起動構成を続行する前に、次の記憶域関連の考慮事項を確認してください。

- 場合によっては、一部のデータが同じ物理ディスク上に配置されていない場合に、データベースのパフォーマンスが改善されることがあります。非常に多くの場合、ログファイルをデータとは別のディスク上に配置すると、パフォーマンスの改善に役立ちます。たとえば、システムテーブルスペースデータファイルとログファイルを異なるディスクに配置できます。InnoDB データファイルに RAW ディスクパーティション (RAW デバイス) を使用して、I/O を高速化することもできます。[システムテーブルスペースに対する RAW ディスクパーティションの使用](#) を参照してください。
- InnoDB はトランザクションセーフな (ACID に準拠した) MySQL 用のストレージエンジンであり、ユーザーデータを保護するためのコミット、ロールバック、およびクラッシュリカバリ機能を備えています。ただし、ベースとなるオペレーティングシステムやハードウェアが公表どおりに機能しない場合は、実行できません。多くのオペレーティングシステムやディスクサブシステムでは、パフォーマンスを改善するために書き込み操作が遅延したり、再指示されたりする可能性があります。一部のオペレーティングシステムでは、まさに `fsync()` システムコールは、ファイルのすべての未書き込みデータがフラッシュされるまで待機するべきですが、実際には、データが安定したストレージにフラッシュされる前に返される可能性があります。このため、オペレーティングシステムのクラッシュや停電によって最近コミットされたデータが破損したり、さらに最悪の場合、書き込み操作が再指示されたためにデータベースが破損したりすることもあります。データの完全性が重要である場合は、本番環境で何かを使用する前に、何らかの形で「電源プラグを抜く」テストを実行してください。macOS では、InnoDB は特別な `fcntl()`

ファイルフラッシュ方法を使用します。Linux では、ライトバックキャッシュを無効にすることが推奨されています。

ATA/SATA ディスクドライブ上で `hdparm -W0 /dev/hda` のようなコマンドを使用すると、ライトバックキャッシュを無効にできる場合があります。一部のドライブやディスクコントローラでは、ライトバックキャッシュを無効にできない可能性があることに注意してください。

- ユーザーを保護する InnoDB のリカバリ機能に関しては、InnoDB では **二重書き込みバッファ**と呼ばれる構造に関連したファイルフラッシュ技術が使用されています。これは、デフォルトで有効になっています (`innodb_doublewrite=ON`)。二重書き込みバッファを使用すると、予期しない終了または停電後のリカバリの安全性が向上し、`fsync()` 操作の必要性を減らすことで、ほとんどの Unix のパフォーマンスが向上します。データの完全性またはエラーの可能性に関心がある場合は、`innodb_doublewrite` オプションを有効のままにすることが推奨されています。二重書き込みバッファの追加情報については、[セクション15.11.1「InnoDB ディスク I/O」](#)を参照してください。
- InnoDB で NFS を使用する前に、[MySQL での NFS の使用](#) で概説されている潜在的な問題を確認してください。

システムテーブルスペースデータファイル構成

`innodb_data_file_path` の起動オプションでは、InnoDB システムテーブルスペースデータファイルの名前、サイズおよび属性を定義します。MySQL サーバーを初期化する前にこのオプションを構成しない場合、デフォルトの動作では、12MB を少し超える単一の自動拡張データファイルが `ibdata1` という名前で作成されます：

```
mysql> SHOW VARIABLES LIKE 'innodb_data_file_path';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| innodb_data_file_path | ibdata1:12M:autoextend |
+-----+-----+
```

完全なデータファイル指定構文には、ファイル名、ファイルサイズ、`autoextend` 属性および `max` 属性が含まれます：

```
file_name:file_size[:autoextend[:max:max_file_size]]
```

ファイルサイズは、**K**、**M** または **G** をサイズ値に追加することで、KB、MB または GB 単位で指定します。データファイルのサイズを KB 単位で指定する場合は、1024 の倍数で指定します。それ以外の場合、キロバイト値はもっとも近いメガバイト (MB) 境界に丸められます。ファイルサイズの合計は、12MB 以上である必要があります。

セミコロンで区切られたリストを使用して、複数のデータファイルを指定できます。例：

```
[mysql]
innodb_data_file_path=ibdata1:50M;ibdata2:50M:autoextend
```

`autoextend` および `max` 属性は、最後に指定されたデータファイルにのみ使用できます。

`autoextend` 属性が指定されている場合、データファイルのサイズは領域が必要になると 64MB ずつ自動的に増加します。`innodb_autoextend_increment` 変数は増分サイズを制御します。

自動拡張データファイルの最大サイズを指定するには、`autoextend` 属性のあとに `max` 属性を使用してください。`max` 属性は、ディスク使用量の制約が重要な場合にものみ使用します。次の構成では、`ibdata1` を 500MB の制限まで拡大できます：

```
[mysql]
innodb_data_file_path=ibdata1:12M:autoextend:max:500M
```

二重書き込みバッファページ用の十分な領域が確保されるように、first システムのテーブルスペースデータファイルには最小ファイルサイズが適用されます。次のテーブルに、各 InnoDB ページサイズの最小ファイルサイズを示します。デフォルトの InnoDB ページサイズは 16384 (16KB) です。

ページサイズ (innodb_page_size)	最小ファイルサイズ
16384 (16KB) 以下	3MB
32768 (32KB)	6MB

ページサイズ (innodb_page_size)	最小ファイルサイズ
65536 (64KB)	12MB

ディスクがいっぱいになった場合は、別のディスクにデータファイルを追加できます。その手順は、[システムテーブルスペースのサイズ変更](#)を参照してください。

個々のファイルのサイズ制限は、オペレーティングシステムによって決まります。大規模ファイルをサポートするオペレーティングシステムでは、ファイルサイズを 4GB を超える値に設定できます。データファイルとして生のディスクパーティションを使用することもできます。[システムテーブルスペースに対する RAW ディスクパーティションの使用](#)を参照してください。

InnoDB ではファイルシステムの最大ファイルサイズが認識されないため、最大ファイルサイズが 2G バイトのような小さい値になっているファイルシステムでは注意してください。

システムテーブルスペースファイルは、デフォルトでデータディレクトリ (`datadir`) に作成されます。別の場所を指定するには、`innodb_data_home_dir` オプションを使用できます。たとえば、`myibdata` という名前のディレクトリにシステムテーブルスペースデータファイルを作成するには、次の構成を使用します：

```
[mysqld]
innodb_data_home_dir = /myibdata/
innodb_data_file_path=ibdata1:50M:autoextend
```

`innodb_data_home_dir` の値を指定する場合は、末尾にスラッシュが必要です。InnoDB ではディレクトリが作成されないため、サーバーを起動する前に、指定したディレクトリが存在することを確認してください。また、MySQL サーバーに、ディレクトリにファイルを作成するための適切なアクセス権があることを確認してください。

InnoDB は、`innodb_data_home_dir` の値をデータファイル名にテキストで連結することで、各データファイルのディレクトリパスを形成します。`innodb_data_home_dir` が定義されていない場合、デフォルト値は「./」(データディレクトリ)です。(MySQL サーバーは、実行を開始すると現在の作業ディレクトリをデータディレクトリに変更します。)

または、システムテーブルスペースデータファイルの絶対パスを指定することもできます。次の構成は、前述の構成と同等です：

```
[mysqld]
innodb_data_file_path=/myibdata/ibdata1:50M:autoextend
```

`innodb_data_file_path` の絶対パスを指定すると、設定は `innodb_data_home_dir` 設定と連結されません。システムテーブルスペースファイルは、指定した絶対パスに作成されます。サーバーを起動する前に、指定したディレクトリが存在している必要があります。

InnoDB 二重書込みバッファファイルの構成

MySQL 8.0.20 の時点では、二重書込みバッファ記憶域は二重書込みファイルに存在し、二重書込みページの記憶域の場所に対する柔軟性を提供します。以前のリリースでは、二重書込みバッファ記憶域はシステムテーブルスペースに存在していました。`innodb_doublewrite_dir` 変数は、InnoDB が起動時に二重書込みファイルを作成するディレクトリを定義します。ディレクトリが指定されていない場合、二重書込みファイルが `innodb_data_home_dir` ディレクトリに作成され、指定されていない場合はデータディレクトリにデフォルト設定されます。

二重書込みファイルを `innodb_data_home_dir` ディレクトリ以外の場所に作成するには、`innodb_doublewrite_dir` 変数を構成します。例：

```
innodb_doublewrite_dir=/path/to/doublewrite_directory
```

その他の二重書込みバッファ変数では、二重書込みファイルの数、スレッド当たりのページ数および二重書込みバッファサイズを定義できます。二重書込みバッファ構成の詳細は、[セクション15.6.4「二重書き込みバッファ」](#)を参照してください。

redo ログファイル構成

デフォルトでは、InnoDB は `ib_logfile0` および `ib_logfile1` という名前のデータディレクトリに 2 つの 5MB redo ログファイルを作成します。

次のオプションを使用して、デフォルトの構成を変更できます：

- `innodb_log_group_home_dir` は、InnoDB ログファイル (redo ログ) へのディレクトリパスを定義します。このオプションが構成されていない場合、InnoDB ログファイルは MySQL データディレクトリ (`datadir`) に作成されます。

このオプションを使用すると、潜在的な I/O リソースの競合を回避するために、InnoDB ログファイルを InnoDB データファイルとは異なる物理記憶域の場所に配置できます。例:

```
[mysqld]
innodb_log_group_home_dir = /dr3/iblogs
```

注記

InnoDB ではディレクトリが作成されないため、サーバーを起動する前にログディレクトリが存在することを確認してください。必要なディレクトリを作成するには、Unix または DOS の `mkdir` コマンドを使用します。

MySQL サーバーに、ログディレクトリにファイルを作成するための適切なアクセス権があることを確認します。より一般的に、サーバーは、ログファイルを作成する必要があるディレクトリでアクセス権を持っている必要があります。

- `innodb_log_files_in_group` は、ロググループ内のログファイルの数を定義します。デフォルトおよび推奨値は 2 です。
- `innodb_log_file_size` は、ロググループ内の各ログファイルのサイズをバイト単位で定義します。ログファイルを結合したサイズ (`innodb_log_file_size * innodb_log_files_in_group`) は、512G バイトよりもわずかに小さい最大値を上回ることができません。たとえば、255 GB のログファイルのペアは制限に近づいていますが、それを超えていません。デフォルトのログファイルサイズは 48MB です。一般に、ログファイルの合計サイズは、サーバーがワークロードアクティビティのピークおよびトラブルをスムーズにできる十分な大きさである必要があります。これは、書込みアクティビティを 1 時間以上処理するための十分な redo ログ領域があることを意味することがよくあります。値を大きくするほど、バッファプール内で必要となるチェックポイントフラッシュアクティビティの数が少なくなるため、ディスク I/O を節約できます。追加情報については [セクション 8.5.4 「InnoDB redo ロギングの最適化」](#) を参照してください。

undo テーブルスペースの構成

デフォルトでは、undo ログは、MySQL インスタンスの初期化時に作成される 2 つの undo テーブルスペースに存在します。undo ログの I/O パターンにより、undo テーブルスペースは SSD 記憶域の適切な候補になります。

`innodb_undo_directory` 変数は、InnoDB がデフォルトの undo テーブルスペースを作成するパスを定義します。この変数が定義されていない場合、デフォルトの undo テーブルスペースがデータディレクトリに作成されます。

`innodb_undo_directory` 変数は動的ではありません。構成するには、サーバーを再起動する必要があります。

追加の undo テーブルスペースの構成の詳細は、[セクション 15.6.3.4 「undo テーブルスペース」](#) を参照してください。

グローバル一時テーブルスペース構成

グローバル一時テーブルスペースには、ユーザー作成一時テーブルに対する変更のロールバックセグメントが格納されます。

デフォルトでは、InnoDB は、`ibtmp1` という名前の単一の自動拡張グローバル一時テーブルスペースデータファイルを `innodb_data_home_dir` ディレクトリに作成します。初期ファイルサイズは 12MB を少し超えています。

`innodb_temp_data_file_path` 変数は、グローバル一時テーブルスペースデータファイルのパス、ファイル名およびファイルサイズを指定します。ファイルサイズは、サイズ値に K、M または G を追加して KB、MB または GB で指定します。ファイルのサイズの合計は、12MB より少し大きくする必要があります。

グローバル一時テーブルスペースデータファイルの代替の場所を指定するには、起動時に `innodb_temp_data_file_path` 変数を構成します。

セッション一時テーブルスペース構成

MySQL 8.0.15 以前では、InnoDB が内部一時テーブル (`internal_tmp_disk_storage_engine=InnoDB`) のディスク上の記憶域エンジンとして構成されている場合、セッション一時テーブルスペースには、最適化によって作成され

たユーザー作成一時テーブルおよび内部一時テーブルが格納されます。MySQL 8.0.16 以降では、InnoDB ストレージエンジンは常にディスク上の内部一時テーブルに使用されます。

`innodb_temp_tablespaces_dir` 変数は、InnoDB がセッション一時テーブルスペースを作成する場所を定義します。デフォルトの場所は、データディレクトリ内の `#innodb_temp` ディレクトリです。

セッション一時テーブルスペースに別の場所を指定するには、起動時に `innodb_temp_tablespaces_dir` 変数を構成します。データディレクトリに対する完全修飾パスまたは相対パスが許可されます。

ページサイズ構成

`innodb_page_size` オプションでは、MySQL インスタンス内のすべての InnoDB テーブルスペースのページサイズを指定します。この値は、インスタンスの作成時に設定され、その後は一定のままです。有効な値は、64KB、32KB、16KB (デフォルト)、8KB および 4KB です。または、ページサイズをバイト単位で指定できます (65536、32768、16384、8192、4096)。

16KB のデフォルトページサイズは、広範囲のワークロードに適しています。特に、バルク更新を伴うテーブルスキャンおよび DML 操作を含むクエリに適しています。ページサイズが小さいほど、多くの小規模な書き込みを伴う OLTP ワークロードの効率性が高くなる可能性があります。その一方で、単一のページに数多くの行が含まれる場合は、競合の問題が発生する可能性もあります。ページを小さくすると、一般に小さなブロックサイズが使用される SSD ストレージデバイスの効率性が高くなる可能性があります。InnoDB のページサイズをストレージデバイスのブロックサイズに近づけると、ディスクに再度書き込まれる未変更データの量が最小限になります。

メモリー構成

MySQL は、様々なキャッシュおよびバッファにメモリーを割り当てて、データベース操作のパフォーマンスを向上させます。InnoDB にメモリーを割り当てる場合は、常にオペレーティングシステムに必要なメモリー、他のアプリケーションに割り当てられたメモリー、および他の MySQL バッファとキャッシュに割り当てられたメモリーを考慮してください。たとえば、MyISAM テーブルを使用する場合は、キーバッファ (`key_buffer_size`) に割り当てられるメモリー量を考慮してください。MySQL バッファおよびキャッシュの概要は、[セクション 8.12.3.1 「MySQL のメモリーの使用方法」](#) を参照してください。

InnoDB に固有のバッファは、次のパラメータを使用して構成されます:

- `innodb_buffer_pool_size` は、バッファプールのサイズを定義します。バッファプールは、InnoDB テーブル、インデックスおよびその他の補助バッファのキャッシュデータを保持するメモリー領域です。バッファプールのサイズはシステムパフォーマンスにとって重要であり、通常、`innodb_buffer_pool_size` はシステムメモリーの 50 から 75% に構成することをお勧めします。デフォルトのバッファプールサイズは 128MB です。その他のガイドラインは、[セクション 8.12.3.1 「MySQL のメモリーの使用方法」](#) を参照してください。InnoDB バッファプールサイズを構成する方法については、[セクション 15.8.3.1 「InnoDB バッファプールサイズの構成」](#) を参照してください。バッファプールサイズは、起動時または動的に構成できます。

メモリーが大量にあるシステムでは、バッファプールを複数のバッファプールインスタンスに分割することで同時実行性を向上させることができます。バッファプールインスタンスの数は、`innodb_buffer_pool_instances` オプションによって制御されます。デフォルトでは、InnoDB はバッファプールインスタンスを 1 つ作成します。バッファプールインスタンスの数は起動時に構成できます。詳細は、[セクション 15.8.3.2 「複数のバッファプールインスタンスの構成」](#) を参照してください。

- `innodb_log_buffer_size` は、InnoDB がディスク上のログファイルへの書き込みに使用するバッファのサイズをバイト単位で定義します。デフォルトのサイズは 16M バイトです。ログバッファを大きくすると、トランザクションがコミットする前にディスクにログを書き込まなくても、大規模なトランザクションを実行できます。多数の行を更新、挿入または削除するトランザクションがある場合は、ログバッファのサイズを増やしてディスク I/O を節約することを検討してください。`innodb_log_buffer_size` は起動時に構成できます。関連情報については、[セクション 8.5.4 「InnoDB redo ロギングの最適化」](#) を参照してください。

警告

32 ビット版の GNU/Linux x86 では、高すぎるメモリー使用率を設定しないように注意してください。`glibc` では、プロセスヒープがスレッドスタック上で増加することが許可されている可能性があるため、サーバーがクラッシュします。グローバルおよびスレッドごとの

バッファおよびキャッシュ用に `mysqld` プロセスに割り当てられたメモリーが 2GB に近いが、それを超えるとリスクがあります。

MySQL のグローバルおよびスレッドごとのメモリー割当てを計算する次のような式を使用して、MySQL メモリー使用量を見積もることができます。MySQL のバージョンおよび構成でバッファおよびキャッシュを考慮するように式を変更する必要がある場合があります。MySQL バッファおよびキャッシュの概要は、[セクション 8.12.3.1 「MySQL のメモリーの使用方法」](#) を参照してください。

```
innodb_buffer_pool_size
+ key_buffer_size
+ max_connections*(sort_buffer_size+read_buffer_size+binlog_cache_size)
+ max_connections*2MB
```

各スレッドではスタックが使用され(多くの場合は 2M バイトですが、Oracle Corporation が提供する MySQL バイナリでは 256K バイトだけです)、最悪のケースでは、`sort_buffer_size` + `read_buffer_size` の追加メモリーも使用されます。

Linux では、カーネルでラージページサポートが有効になっている場合、InnoDB はラージページを使用してバッファプールにメモリーを割り当てることができます。[セクション 8.12.3.2 「ラージページのサポートの有効化」](#) を参照してください。

15.8.2 読み取り専用操作用の InnoDB の構成

サーバーの起動時に `--innodb-read-only` 構成オプションを有効にすることで、MySQL データディレクトリが読み取り専用メディアにある InnoDB テーブルをクエリーすることができます。

有効にする方法

読み取り操作用にインスタンスを準備するには、必要なすべての情報が読み取り専用メディア上に格納される前に、データファイルにフラッシュされることを確認します。変更バッファが無効になっている (`innodb_change_buffering=0`) サーバーを実行し、[低速シャットダウン](#) を実行します。

MySQL インスタンス全体にわたって読み取り専用モードを有効にするには、サーバーの起動時に次の構成オプションを指定します。

- `--innodb-read-only=1`
- インスタンスが DVD や CD などの読み取り専用メディア上にある場合、または `/var` ディレクトリがすべてのユーザーから書き込み可能でない場合: `--pid-file=path_on_writeable_media` および `--event-scheduler=disabled`
- `--innodb-temp-data-file-path`。このオプションは、InnoDB 一時テーブルスペースデータファイルのパス、ファイル名およびファイルサイズを指定します。デフォルト設定は `ibtmp1:12M:autoextend` で、`ibtmp1` 一時テーブルスペースデータファイルがデータディレクトリに作成されます。読み取り専用操作のためにインスタンスを準備するには、`innodb_temp_data_file_path` をデータディレクトリ外の場所に設定します。パスは、データディレクトリに対する相対パスである必要があります。例:

```
--innodb-temp-data-file-path=../../tmp/ibtmp1:12M:autoextend
```

MySQL 8.0 の時点では、`innodb_read_only` を有効にすると、すべてのストレージエンジンのテーブルの作成および削除操作が防止されます。これらの操作により、`mysql` システムデータベースのデータディクショナリテーブルが変更されますが、これらのテーブルは InnoDB ストレージエンジンを使用し、`innodb_read_only` が有効な場合は変更できません。`ANALYZE TABLE`、`ALTER TABLE tbl_name ENGINE=engine_name` など、データディクショナリテーブルを変更する操作にも同じ制限が適用されます。

また、`mysql` システムデータベースの他のテーブルでは、MySQL 8.0 の InnoDB ストレージエンジンが使用されます。これらのテーブルを読み取り専用にすると、テーブルを変更する操作が制限されます。たとえば、`CREATE USER`、`GRANT`、`REVOKE` および `INSTALL PLUGIN` 操作は読み取り専用モードでは許可されません。

使用シナリオ

この操作モードは、次のような状況に適しています。

- DVD や CD などの読み取り専用ストレージメディア上に、MySQL アプリケーションまたは MySQL データセットを配布する。
- (一般に、データウェアハウス構成で) 同じデータディレクトリで同時にクエリーを実行する複数の MySQL インスタンス。この方法を使用すれば、負荷の高い MySQL インスタンスで発生する可能性のある **ボトルネック** を回避したり、さまざまなインスタンスに対してさまざまな構成オプションを使用して、特定の種類のクエリーを個別に調整したりできる場合があります。
- 安全性またはデータの完全性の理由により読み取り専用の状態になったデータ (アーカイブされたバックアップデータなど) のクエリーを実行する。

注記

この機能の目的は、読み取り専用の側面に基づいた生のパフォーマンスではなく、主に配布および配備する際の柔軟性です。サーバー全体を読み取り専用にする必要なしで、読み取り専用クエリーのパフォーマンスを調整する方法については、[セクション 8.5.3 「InnoDB の読み取り専用トランザクションの最適化」](#) を参照してください。

動作

`--innodb-read-only` オプションを使用して、サーバーが読み取り専用モードで実行されると、特定の InnoDB 機能およびコンポーネントが減少したり、完全に無効になったりします。

- **変更バッファ** (特に、変更バッファからのマージ) は実行されません。読み取り専用操作にインスタンスを準備するときに、変更バッファが空になっていることを確認するには、変更バッファを無効にして (`innodb_change_buffering=0`)、まず **低速シャットダウン** を実行します。
- 起動時には **クラッシュリカバリフェーズ** がありません。インスタンスを読み取り専用状態にする前に、**低速シャットダウン** が実行されている必要があります。
- 読み取り専用操作では **Redo ログ** が使用されないため、インスタンスを読み取り専用にする前に、`innodb_log_file_size` を最小限のサイズ (1M バイト) に設定できます。
- ほとんどのバックグラウンドスレッドはオフになります。I/O の読取りスレッドと、読取り専用モードで許可される一時ファイルへの書き込み用の I/O 書き込みスレッドおよびページフラッシュコーデイネータスレッドは残ります。バッファプールのサイズ変更スレッドもアクティブなままになり、バッファプールのオンラインサイズ変更が可能になります。
- デッドロックに関する情報やモニターの出力などは、一時ファイルに書き込まれません。その結果、**SHOW ENGINE INNODB STATUS** で出力が生成されなくなります。
- 構成オプションの設定を変更すると、通常は書き込み操作の動作が変更されますが、サーバーが読み取り専用モードになっている場合は影響がありません。
- **分離レベル** を強制的に適用する **MVCC** 処理が無効になります。更新も削除もできないため、すべてのクエリーで最新バージョンのレコードが読み取られます。
- **Undo ログ** は使用されません。 `innodb_undo_tablespaces` および `innodb_undo_directory` 構成オプションの設定を無効にします。

15.8.3 InnoDB バッファプールの構成

このセクションでは、InnoDB バッファプールの構成およびチューニングについて説明します。

15.8.3.1 InnoDB バッファプールサイズの構成

InnoDB バッファプールのサイズは、オフラインまたはサーバーの実行中に構成できます。このセクションで説明する動作は、両方の方法に適用されます。バッファプールサイズをオンラインで構成する方法の詳細は、[オンラインでの InnoDB バッファプールサイズの構成](#) を参照してください。

`innodb_buffer_pool_size` を増減すると、操作はチャンク単位で実行されます。チャンクサイズは、デフォルトの 128M を持つ `innodb_buffer_pool_chunk_size` 構成オプションによって定義されます。詳細は、[InnoDB バッファプールのチャンクサイズの構成](#) を参照してください。

バッファプールサイズは、常に `innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances` と等しいか倍数である必要があります。 `innodb_buffer_pool_size` を `innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances` と等しくない値または倍数に構成すると、バッファプールサイズは `innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances` と等しいか倍数の値に自動的に調整されます。

次の例では、 `innodb_buffer_pool_size` が 8G に設定され、 `innodb_buffer_pool_instances` が 16 に設定されます。 `innodb_buffer_pool_chunk_size` は 128M で、これがデフォルト値です。

8G は `innodb_buffer_pool_instances=16 * innodb_buffer_pool_chunk_size=128M` の倍数 (2G) であるため、8G は有効な `innodb_buffer_pool_size` 値です。

```
shell> mysql -innodb-buffer-pool-size=8G --innodb-buffer-pool-instances=16
```

```
mysql> SELECT @@innodb_buffer_pool_size/1024/1024/1024;
+-----+
| @@innodb_buffer_pool_size/1024/1024/1024 |
+-----+
| 8.000000000000000 |
+-----+
```

この例では、 `innodb_buffer_pool_size` は 9G に設定され、 `innodb_buffer_pool_instances` は 16 に設定されます。 `innodb_buffer_pool_chunk_size` は 128M で、これがデフォルト値です。この場合、9G は `innodb_buffer_pool_instances=16 * innodb_buffer_pool_chunk_size=128M` の倍数ではないため、 `innodb_buffer_pool_size` は 10G (`innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances` の倍数) に調整されます。

```
shell> mysql -innodb-buffer-pool-size=9G --innodb-buffer-pool-instances=16
```

```
mysql> SELECT @@innodb_buffer_pool_size/1024/1024/1024;
+-----+
| @@innodb_buffer_pool_size/1024/1024/1024 |
+-----+
| 10.000000000000000 |
+-----+
```

InnoDB バッファプールのチャンクサイズの構成

`innodb_buffer_pool_chunk_size` は 1MB (1048576 バイト) 単位で増減できますが、起動時、コマンドライン文字列または MySQL 構成ファイルでのみ変更できます。

コマンドライン:

```
shell> mysql -innodb-buffer-pool-chunk-size=134217728
```

構成ファイル:

```
[mysqld]
innodb_buffer_pool_chunk_size=134217728
```

`innodb_buffer_pool_chunk_size` を変更する場合は、次の条件が適用されます:

- バッファプールの初期化時に、新しい `innodb_buffer_pool_chunk_size` 値 * `innodb_buffer_pool_instances` が現在のバッファプールサイズより大きい場合、 `innodb_buffer_pool_chunk_size` は `innodb_buffer_pool_size / innodb_buffer_pool_instances` に切り捨てられます。

たとえば、バッファプールが 2GB (2147483648 バイト)、4 バッファプールインスタンスおよびチャンクサイズ 1GB (1073741824 バイト) で初期化されている場合、次に示すようにチャンクサイズは `innodb_buffer_pool_size / innodb_buffer_pool_instances` と等しい値に切り捨てられます:

```
shell> mysql -innodb-buffer-pool-size=2147483648 --innodb-buffer-pool-instances=4
--innodb-buffer-pool-chunk-size=1073741824;
```

```
mysql> SELECT @@innodb_buffer_pool_size;
+-----+
| @@innodb_buffer_pool_size |
```

```
+-----+
|      2147483648 |
+-----+

mysql> SELECT @@innodb_buffer_pool_instances;
+-----+
| @@innodb_buffer_pool_instances |
+-----+
|              4 |
+-----+

# Chunk size was set to 1GB (1073741824 bytes) on startup but was
# truncated to innodb_buffer_pool_size / innodb_buffer_pool_instances

mysql> SELECT @@innodb_buffer_pool_chunk_size;
+-----+
| @@innodb_buffer_pool_chunk_size |
+-----+
|      536870912 |
+-----+
```

- バッファプールサイズは、常に `innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances` と等しいか倍数である必要があります。 `innodb_buffer_pool_chunk_size` を変更すると、`innodb_buffer_pool_size` は `innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances` と等しいか倍数の値に自動的に調整されます。調整は、バッファプールが初期化されたときに行われます。この動作を次の例に示します:

```
# The buffer pool has a default size of 128MB (134217728 bytes)

mysql> SELECT @@innodb_buffer_pool_size;
+-----+
| @@innodb_buffer_pool_size |
+-----+
|      134217728 |
+-----+

# The chunk size is also 128MB (134217728 bytes)

mysql> SELECT @@innodb_buffer_pool_chunk_size;
+-----+
| @@innodb_buffer_pool_chunk_size |
+-----+
|      134217728 |
+-----+

# There is a single buffer pool instance

mysql> SELECT @@innodb_buffer_pool_instances;
+-----+
| @@innodb_buffer_pool_instances |
+-----+
|              1 |
+-----+

# Chunk size is decreased by 1MB (1048576 bytes) at startup
# (134217728 - 1048576 = 133169152):

shell> mysqld --innodb-buffer-pool-chunk-size=133169152

mysql> SELECT @@innodb_buffer_pool_chunk_size;
+-----+
| @@innodb_buffer_pool_chunk_size |
+-----+
|      133169152 |
+-----+

# Buffer pool size increases from 134217728 to 266338304
# Buffer pool size is automatically adjusted to a value that is equal to
# or a multiple of innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances

mysql> SELECT @@innodb_buffer_pool_size;
+-----+
| @@innodb_buffer_pool_size |
+-----+
|      266338304 |
+-----+
```

```
+-----+
```

この例では、同じ動作を示しますが、複数のバッファプールインスタンスがあります：

```
# The buffer pool has a default size of 2GB (2147483648 bytes)

mysql> SELECT @@innodb_buffer_pool_size;
+-----+
| @@innodb_buffer_pool_size |
+-----+
|          2147483648 |
+-----+

# The chunk size is .5 GB (536870912 bytes)

mysql> SELECT @@innodb_buffer_pool_chunk_size;
+-----+
| @@innodb_buffer_pool_chunk_size |
+-----+
|          536870912 |
+-----+

# There are 4 buffer pool instances

mysql> SELECT @@innodb_buffer_pool_instances;
+-----+
| @@innodb_buffer_pool_instances |
+-----+
|                4 |
+-----+

# Chunk size is decreased by 1MB (1048576 bytes) at startup
# (536870912 - 1048576 = 535822336):

shell> mysqld --innodb-buffer-pool-chunk-size=535822336

mysql> SELECT @@innodb_buffer_pool_chunk_size;
+-----+
| @@innodb_buffer_pool_chunk_size |
+-----+
|          535822336 |
+-----+

# Buffer pool size increases from 2147483648 to 4286578688
# Buffer pool size is automatically adjusted to a value that is equal to
# or a multiple of innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances

mysql> SELECT @@innodb_buffer_pool_size;
+-----+
| @@innodb_buffer_pool_size |
+-----+
|          4286578688 |
+-----+
```

前述の例に示すように、この値を変更するとバッファプールのサイズが増加する可能性があるため、`innodb_buffer_pool_chunk_size` を変更する場合は注意が必要です。`innodb_buffer_pool_chunk_size` を変更する前に、`innodb_buffer_pool_size` への影響を計算して、結果のバッファプールサイズが許容範囲内であることを確認します。

注記

潜在的なパフォーマンスの問題を回避するには、チャンク (`innodb_buffer_pool_size / innodb_buffer_pool_chunk_size`) の数が 1000 を超えないようにする必要があります。

オンラインでの InnoDB バッファプールサイズの構成

`innodb_buffer_pool_size` 構成オプションは、`SET` ステートメントを使用して動的に設定でき、サーバーを再起動せずにバッファプールのサイズを変更できます。例：

```
mysql> SET GLOBAL innodb_buffer_pool_size=402653184;
```

注記

バッファプールサイズは `innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances` と等しいか、倍数である必要があります。これらの変数設定を変更するには、サーバーを再起動する必要があります。

バッファプールのサイズを変更する前に、InnoDB API を介して実行されるアクティブなトランザクションおよび操作を完了する必要があります。サイズ変更操作を開始しても、すべてのアクティブなトランザクションが完了するまで操作は開始されません。サイズ変更操作が進行中になると、バッファプールへのアクセスを必要とする新しいトランザクションおよび操作は、サイズ変更操作が終了するまで待機する必要があります。ルールの例外は、バッファプールがデフラグされている間はバッファプールへの同時アクセスが許可され、バッファプールサイズが小さくなるとページが取り下げられることです。同時アクセスを許可するという欠点は、ページの取下げ中に一時的に使用可能なページが不足する可能性があることです。

注記

バッファプールのサイズ変更操作の開始後に開始された場合、ネストされたトランザクションは失敗する可能性があります。

オンラインバッファプールのサイズ変更の進行状況の監視

`InnoDB_buffer_pool_resize_status` では、バッファプールのサイズ変更の進行状況がレポートされます。例:

```
mysql> SHOW STATUS WHERE Variable_name='InnoDB_buffer_pool_resize_status';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| InnoDB_buffer_pool_resize_status | Resizing also other hash tables. |
+-----+-----+
```

バッファプールのサイズ変更の進行状況も、サーバーエラーログに記録されます。次の例は、バッファプールのサイズを増やすとログに記録されるノートを示しています:

```
[Note] InnoDB: Resizing buffer pool from 134217728 to 4294967296. (unit=134217728)
[Note] InnoDB: disabled adaptive hash index.
[Note] InnoDB: buffer pool 0 : 31 chunks (253952 blocks) was added.
[Note] InnoDB: buffer pool 0 : hash tables were resized.
[Note] InnoDB: Resized hash tables at lock_sys, adaptive hash index, dictionary.
[Note] InnoDB: completed to resize buffer pool from 134217728 to 4294967296.
[Note] InnoDB: re-enabled adaptive hash index.
```

次の例は、バッファプールのサイズを小さくしたときに記録されるノートを示しています:

```
[Note] InnoDB: Resizing buffer pool from 4294967296 to 134217728. (unit=134217728)
[Note] InnoDB: disabled adaptive hash index.
[Note] InnoDB: buffer pool 0 : start to withdraw the last 253952 blocks.
[Note] InnoDB: buffer pool 0 : withdrew 253952 blocks from free list. tried to relocate 0 pages.
(253952/253952)
[Note] InnoDB: buffer pool 0 : withdrawn target 253952 blocks.
[Note] InnoDB: buffer pool 0 : 31 chunks (253952 blocks) was freed.
[Note] InnoDB: buffer pool 0 : hash tables were resized.
[Note] InnoDB: Resized hash tables at lock_sys, adaptive hash index, dictionary.
[Note] InnoDB: completed to resize buffer pool from 4294967296 to 134217728.
[Note] InnoDB: re-enabled adaptive hash index.
```

オンラインバッファプールの内部サイズ変更

サイズ変更操作はバックグラウンドスレッドによって実行されます。バッファプールのサイズを増やすと、サイズ変更操作は次のようになります:

- `chunks` にページを追加します (チャンクサイズは `innodb_buffer_pool_chunk_size` によって定義されます)
- メモリー内の新しいアドレスを使用するためのハッシュテーブル、リスト、およびポインタをカバー
- 空きリストに新規ページを追加

これらの操作の進行中、他のスレッドはバッファプールへのアクセスをブロックされます。

バッファプールのサイズを小さくすると、サイズ変更操作は次のようになります:

- バッファプールをデフラグし、ページを取り下げます (解放します)
- `chunks` 内のページを削除します (チャンクサイズは `innodb_buffer_pool_chunk_size` によって定義されます)
- ハッシュテーブル、リストおよびポインタをメモリー内の新しいアドレスを使用するように変換

これらの操作のうち、バッファプールをデフラグしてページを取り下げるだけで、他のスレッドがバッファプールに同時にアクセスできます。

15.8.3.2 複数のバッファプールインスタンスの構成

バッファプールが数 G バイトの範囲にあるシステムでは、バッファプールを個別のインスタンスに分割すると、キャッシュされたページに対して異なるスレッドが読み取りおよび書き込みを行うときの競合が減るため、並列性が向上する場合があります。この機能は通常、**バッファプールのサイズ**が数 G バイトの範囲にあるシステムを対象にしています。複数のバッファプールインスタンスは `innodb_buffer_pool_instances` 構成オプションを使用して構成され、また `innodb_buffer_pool_size` 値を調整することもできます。

InnoDB バッファプールが大きい場合、メモリーから取得することで多くのデータリクエストを満たすことができます。複数のスレッドが一度にバッファプールにアクセスしようとした場合は、ボトルネックが発生する可能性があります。この競合を最小限に抑えるために、複数のバッファプールを有効にすることができます。バッファプールに格納されるか、またはバッファプールから読み取られる各ページは、ハッシュ関数を使用して、いずれかのバッファプールにランダムに割り当てられます。各バッファプールは、バッファプールに接続されている独自の空きリスト、フラッシュリスト、LRU およびその他のすべてのデータ構造を管理します。MySQL 8.0 より前は、各バッファプールは独自のバッファプール相互排他ロックによって保護されていました。MySQL 8.0 以降では、競合を減らすために、バッファプール `mutex` が複数のリストおよびハッシュ保護 `mutex` に置き換えられました。

複数のバッファプールインスタンスを有効にするには、`innodb_buffer_pool_instances` 構成オプションを 1 (デフォルト) より大きく 64 (最大) までの値に設定します。このオプションは、`innodb_buffer_pool_size` を 1GB 以上のサイズに設定した場合にのみ有効になります。指定した合計サイズは、すべてのバッファプール間で分割されます。最高の効率を得るには、`innodb_buffer_pool_instances` と `innodb_buffer_pool_size` の組み合わせを、各バッファプールインスタンスが少なくとも 1G バイトになるように指定します。

InnoDB バッファプールサイズの変更については、[セクション15.8.3.1「InnoDB バッファプールサイズの構成」](#) を参照してください。

15.8.3.3 バッファプールをスキャンに耐えられるようにする

厳密な LRU アルゴリズムを使用するかわりに、InnoDB では、`buffer pool` に取り込まれ、再度アクセスされないデータ量を最小限に抑える手法を使用します。目標は、**先読み**や**フルテーブルスキャン**によって、その後アクセスされるかどうか分からない新しいブロックが読み取られた場合でも、頻繁にアクセスされるページ (「ホットページ」) が確実にバッファプール内に残るようにすることです。

新しく読み取られたブロックは、LRU リストの途中に挿入されます。新しく読み取られたページはすべて、デフォルトでは LRU リストの末尾から $3/8$ にあたる場所に挿入されます。これらのページは、はじめてバッファプール内でアクセスされたときに、リストの前面 (直近で使用された端) に移動されます。したがって、アクセスされないページでは LRU リストの先頭部分にアクセスされず、「エージアウト」は厳密な LRU アプローチより早くアクセスされます。この配置では、LRU リストが 2 つのセグメントに分割されます。つまり、挿入ポイントの下流にあるページは「古い」とみなされ、LRU のエビクションの望ましい対象になります。

InnoDB バッファプールの内部動作および LRU アルゴリズムの詳細は、[セクション15.5.1「バッファプール」](#) を参照してください。

LRU リストの挿入ポイントを制御し、テーブルスキャンまたはインデックススキャンによってバッファプールに入れられたブロックに InnoDB が同じ最適化を適用するかどうかを選択できます。構成パラメータ `innodb_old_blocks_pct` は、LRU リスト内の「古い」ブロックの割合 (%) を制御します。`innodb_old_blocks_pct` のデフォルト値は 37 であり、元の固定された $3/8$ の比率に対応します。この値の範囲は、5 (バッファプール内の新しいページが非常に早く古くなります) から 95 (バッファプールの 5% しかホットページとして予約されないため、アルゴリズムがなじみのある LRU の方法に近くなります) までです。

バッファープールを先読みによって混乱した状態にならないように維持する最適化は、テーブルまたはインデックススキャンによる同様の問題も回避できます。これらのスキャンでは通常、データページはすばやく連続して数回アクセスされ、それ以降は二度とアクセスされません。構成パラメータ `innodb_old_blocks_time` は、あるページにはじめてアクセスしたあと、そのページが LRU リストの前面 (直近で使用された端) に移動されることなくアクセス可能になっている時間ウィンドウ (ミリ秒単位) を指定します。 `innodb_old_blocks_time` のデフォルト値は 1000 です。この値を大きくすると、より多くのブロックがバッファープールから早く古くなる可能性があります。

`innodb_old_blocks_pct` と `innodb_old_blocks_time` の両方を MySQL オプションファイル (`my.cnf` または `my.ini`) で指定するか、`SET GLOBAL` ステートメントを使用して実行時に変更できます。実行時に値を変更するには、グローバルシステム変数を設定するのに十分な権限が必要です。 [セクション 5.1.9.1 「システム変数権限」](#) を参照してください。

これらのパラメータの設定の影響を測定するために、`SHOW ENGINE INNODB STATUS` コマンドはバッファープール統計をレポートします。詳細は、[InnoDB 標準モニターを使用したバッファープールのモニタリング](#) を参照してください。

これらのパラメータの効果はハードウェア構成、使用しているデータ、およびワークロードの詳細によって大幅に異なる場合があるため、パフォーマンスが重要な環境や本番環境でこれらの設定を変更する前には、常にベンチマークによってその有効性を確認してください。

ほとんどのアクティビティーが、大規模なスキャンにつながる定期的なバッチレポートクエリーを含む OLTP タイプである混在ワークロード環境では、バッチの実行中に `innodb_old_blocks_time` の値を設定すると、通常のワークロードのワーキングセットをバッファープール内に維持するのに役立つ場合があります。

バッファープール内に完全に収まらない大きなテーブルをスキャンする場合は、`innodb_old_blocks_pct` を小さな値に設定すると、1 回しか読み取られないデータがバッファープールの大きな部分を消費することはなくなります。たとえば、`innodb_old_blocks_pct=5` を設定すると、1 回しか読み取られないこのデータがバッファープールの 5% に制限されます。

メモリーに収まる小さなテーブルをスキャンする場合は、バッファープール内でページを移動するためのオーバーヘッドが低いいため、`innodb_old_blocks_pct` をデフォルト値のままにするか、あるいは場合によっては (`innodb_old_blocks_pct=50` など) 増やすこともできます。

`innodb_old_blocks_time` パラメータの効果は、比較的効果の小さい `innodb_old_blocks_pct` パラメータに比べて予測が困難であり、ワークロードによる変動も大きくなります。最適な値に到達するには、`innodb_old_blocks_pct` の調整によるパフォーマンス向上が不十分な場合は独自のベンチマークを実施してください。

15.8.3.4 InnoDB バッファープールのプリフェッチ (先読み) の構成

`read-ahead` リクエストは、`buffer pool` の複数のページを非同期にプリフェッチするための I/O リクエストであり、これらのページの必要性が低下することが予想されます。これらの要求によって、すべてのページが 1 つの `エクステン`ト内に移動されます。InnoDB では、I/O のパフォーマンスを向上させるために、次の 2 つの先読みアルゴリズムを使用します:

線形先読みは、順次にアクセスされているバッファープール内のページに基づいて、どのページがすぐに必要になる可能性があるかを予測する手法です。InnoDB が先読み操作を実行するタイミングを制御するには、構成パラメータ `innodb_read_ahead_threshold` を使用して、非同期読取りリクエストのトリガーに必要な順次ページアクセスの数を調整します。このパラメータが追加される前に、InnoDB は、現行エクステン

トの最後のページを読み取ったときに、次のエクステン

ト全体に対して非同期プリフェッチリクエストを発行するかどうかのみを計算します。

構成パラメータ `innodb_read_ahead_threshold` は、InnoDB が順次ページアクセスのパターンを検出する方法を制御します。エクステン

トから順次読み取られるページ数が `innodb_read_ahead_threshold` 以上の場合、InnoDB は次のエクステン

ト全体の非同期先読み操作を開始します。 `innodb_read_ahead_threshold` は、0-64 の任意の値に設定できます。デフォルト値は 56 です。この値が大きいほど、アクセスパターンのチェックは厳密になります。たとえば、値を 48 に設定すると、現在のエクステン

ト内の 48 ページが順次アクセスされた場合にのみ、InnoDB によって線形先読みリクエストがトリガーされます。値が 8 の場合、エクステン

ト内の 8 ページが順次アクセスされていても、InnoDB は非同期先読みをトリガーします。このパラメータの値は、MySQL [configuration file](#) で設定することも、グローバルシステム変数を設定するのに十分な権限を必要とする `SET GLOBAL` ステートメントを使用して動的に変更することもできます。 [セクション 5.1.9.1 「システム変数権限」](#) を参照してください。

ランダム先読みは、すでにバッファープール内に存在するページに基づいて、これらのページが読み取られた順序には関係なく、ページがいつ必要になる可能性があるかを予測する手法です。同じエクステン

ジガバッファプールに見つかった場合、InnoDB はエクステントの残りのページをプリフェッチするリクエストを非同期的に発行します。この機能を有効にするには、構成変数 `innodb_random_read_ahead` を ON に設定します。

`SHOW ENGINE INNODB STATUS` コマンドは、先読みアルゴリズムの有効性を評価するのに役立つ統計を表示します。統計には、次のグローバルステータス変数のカウンタ情報が含まれます:

- `Innodb_buffer_pool_read_ahead`
- `Innodb_buffer_pool_read_ahead_evicted`
- `Innodb_buffer_pool_read_ahead_rnd`

これらの情報は、`innodb_random_read_ahead` 設定を微調整する場合に役立つことがあります。

I/O パフォーマンスの詳細は、[セクション8.5.8「InnoDB ディスク I/O の最適化」](#) および [セクション8.12.1「ディスク I/O の最適化」](#) を参照してください。

15.8.3.5 バッファプールのフラッシュの構成

InnoDB は、バッファプールからのダーティページのフラッシュなど、特定のタスクをバックグラウンドで実行します。ダーティページは、変更されたが、まだディスク上のデータファイルに書き込まれていないページです。

MySQL 8.0 では、バッファプールのフラッシュはページクリーナスレッドによって実行されます。ページクリーナスレッドの数は、`innodb_page_cleaners` 変数 (デフォルト値は 4) によって制御されます。ただし、ページクリーナスレッドの数がバッファプールインスタンスの数を超えると、`innodb_page_cleaners` は自動的に `innodb_buffer_pool_instances` と同じ値に設定されます。

ダーティページの割合が `innodb_max_dirty_pages_pct_lwm` 変数で定義された最低水位標値に達すると、バッファプールのフラッシュが開始されます。デフォルトの最低水位標はバッファプールページの 10% です。`innodb_max_dirty_pages_pct_lwm` 値が 0 の場合、この早期フラッシュ動作は無効になります。

`innodb_max_dirty_pages_pct_lwm` のしきい値の目的は、バッファプール内のダーティページの割合を制御し、ダーティページの量が `innodb_max_dirty_pages_pct` 変数で定義されたしきい値 (デフォルト値 90) に到達しないようにすることです。バッファプール内のダーティページの割合が `innodb_max_dirty_pages_pct` しきい値に達した場合、InnoDB はバッファプールページを積極的にフラッシュします。

追加の変数により、バッファプールのフラッシュ動作を微調整できます:

- `innodb_flush_neighbors` 変数は、バッファプールからページをフラッシュすると、同じエクステント内のほかのダーティページもフラッシュするかどうかを定義します。
 - デフォルト設定の 0 は、`innodb_flush_neighbors` を無効にします。同じエクステント内のダーティページはフラッシュされません。シーク時間が重要な要因ではない非定期ストレージ (SSD) デバイスには、この設定をお勧めします。
 - 1 に設定すると、連続したダーティページが同じエクステントにフラッシュされます。
 - 2 に設定すると、ダーティページが同じエクステントでフラッシュされます。

テーブルデータが従来の HDD ストレージデバイスに格納されている場合、隣接するページをある操作でフラッシュすると、個々のページを異なるタイミングでフラッシュする場合と比較して、I/O のオーバーヘッド (主にディスクシーク操作) が削減されます。SSD に格納されているテーブルデータの場合、シーク時間は重要な要因ではなく、この設定を無効にして書き込み操作を分散できます。

- `innodb_lru_scan_depth` 変数は、バッファプールインスタンスごとに、フラッシュするダーティページを検索するページクリーナスレッドスキャンをバッファプール LRU リストのどれだけ下に表示するかを指定します。これは、ページクリーナスレッドによって毎秒 1 回実行されるバックグラウンド操作です。

デフォルトより小さい設定は、通常、ほとんどのワークロードに適しています。必要以上の値を指定すると、パフォーマンスに影響する可能性があります。通常のワークロードでスベア I/O 容量がある場合のみ、値を増やすことを検討してください。逆に、書き込み集中型のワークロードが I/O の容量を満たしている場合は、特に大きなバッファプールの場合に値を減らします。

`innodb_lru_scan_depth` をチューニングする場合は、小さい値から始めて、ゼロの空きページが表示されることがほとんどないという目標で設定を上方に構成します。また、`innodb_lru_scan_depth * innodb_buffer_pool_instances` は毎秒ページクリーナスレッドによって実行される作業量を定義するため、バッファプールインスタンスの数を変更するときに `innodb_lru_scan_depth` を調整することを検討してください。

`innodb_flush_neighbors` および `innodb_lru_scan_depth` 変数は、主に書き込み集中型のワークロードを対象としています。DML アクティビティが重い場合、フラッシュが十分に積極的でないとフラッシュが遅れる可能性があります。または、フラッシュが過度に積極的であると、ディスク書き込みが I/O の容量を満たす可能性があります。理想的な設定は、ワークロード、データのアクセスパターン、およびストレージ構成（たとえば、データが HDD または SSD デバイスのどちらに格納されているか）によって異なります。

適応型フラッシュ

InnoDB では、適応型フラッシュアルゴリズムを使用して、redo ログ生成の速度および現在のフラッシュ率に基づいてフラッシュ率を動的に調整します。この目的は、フラッシュアクティビティが現在のワークロードに対応できるようにすることで、全体的なパフォーマンスをスムーズ化することです。フラッシュレートを自動的に調整すると、バッファプールのフラッシュによる I/O アクティビティのバーストが通常の読み取りおよび書き込みアクティビティで使用可能な I/O 容量に影響する場合に発生する可能性があるスループットの急激なディップを回避するのに役立ちます。

多くの redo エントリを生成する書き込み集中型のワークロードに通常関連付けられているシャープなチェックポイントは、スループットの急激な変更を引き起こす可能性があります。シャープなチェックポイントは、InnoDB がログファイルの一部を再利用する場合に発生します。これを行う前に、ログファイルのその部分に redo エントリがあるすべてのデータページをフラッシュする必要があります。ログファイルがいっぱいになると、シャープなチェックポイントが発生し、スループットが一時的に低下します。このシナリオは、`innodb_max_dirty_pages_pct` のしきい値に達していない場合でも発生する可能性があります。

適応型フラッシュアルゴリズムは、バッファプール内のデータページの数と redo ログレコードが生成される割合を追跡することで、このようなシナリオを回避するのに役立ちます。この情報に基づいて、バッファプールから毎秒フラッシュするデータページの数決定され、ワークロードの突然の変更を管理できます。

`innodb_adaptive_flushing_lwm` 変数は、redo ログ容量の最低水位標を定義します。このしきい値を超えると、`innodb_adaptive_flushing` 変数が無効になっていても適応型フラッシュが有効になります。

内部ベンチマークは、アルゴリズムが一定期間スループットを維持するだけでなく、全体的なスループットを大幅に向上させることもできることを示しています。ただし、適応型フラッシュはワークロードの I/O パターンに大きく影響を与える可能性があり、すべての場合に適切であるとはかぎりません。これがもっとも大きな利点をもたらすのは、Redo ログがいっぱいになるおそれがある場合です。適応型フラッシュがワークロードの特性に適していない場合は、無効にできます。適応型フラッシュは、デフォルトで有効になっている `innodb_adaptive_flushing` 変数によって制御されます。

`innodb_flushing_avg_loops` は、InnoDB が以前に計算したフラッシュ状態のスナップショットを保持する反復回数を定義し、適応型フラッシュがフォアグラウンドのワークロード変更に応じてどの程度迅速に回答するかを制御します。`innodb_flushing_avg_loops` 値が高いほど、InnoDB は以前に計算されたスナップショットを長く保持するため、適応型フラッシュの応答が遅くなります。高い値を設定する場合は、redo ログ使用率が 75% (非同期フラッシュが開始されるハードコードされた制限) に達しないようにし、`innodb_max_dirty_pages_pct` のしきい値によってデータページの数ワークロードに適したレベルに維持されるようにすることが重要です。

一貫性のあるワークロードを持つシステム、大きなログファイルサイズ (`innodb_log_file_size`) および 75% のログ領域使用率に達しない小さなスパイクでは、フラッシュをできるだけスムーズに保つために高い `innodb_flushing_avg_loops` 値を使用する必要があります。負荷が非常に高いスパイクまたはログファイルが領域を多く提供しないシステムでは、フラッシュを小さくするとワークロードの変更を密接に追跡でき、75% のログ領域使用率への到達を回避できます。

フラッシュが遅れた場合、バッファプールのフラッシュ率は、`innodb_io_capacity` 設定で定義されているように、InnoDB で使用可能な I/O 容量を超えることがあります。このような状況では、I/O アクティビティのスパイクがサーバーの I/O 容量全体を消費しないように、`innodb_io_capacity_max` 値によって I/O 容量の上限が定義されます。

`innodb_io_capacity` 設定は、すべてのバッファプールインスタンスに適用できます。データページがフラッシュされると、I/O 容量はバッファプールインスタンス間で均等に分割されます。

アイドル期間中のバッファフラッシュの制限

MySQL 8.0.18 の時点では、`innodb_idle_flush_pct` 変数を使用して、アイドル期間 (データベースページが変更されない期間) 中のバッファプールのフラッシュ率を制限できます。`innodb_idle_flush_pct` 値は、InnoDB で使用可能な I/O 操作数/秒を定義する `innodb_io_capacity` 設定の割合です。デフォルトの `innodb_idle_flush_pct` 値は 100 で、これは `innodb_io_capacity` 設定の 100% です。アイドル期間中のフラッシュを制限するには、100 未満の `innodb_idle_flush_pct` 値を定義します。

アイドル期間中のページフラッシュの制限は、ソリッドステートストレージデバイスの寿命を延長するのに役立ちます。アイドル期間中のページフラッシュの制限の副作用には、長時間のアイドル期間後の停止時間が長くなり、サーバー障害が発生した場合のリカバリ期間が長くなることがあります。

15.8.3.6 バッファプールの状態の保存と復元

サーバーの再起動後の `warmup` 期間を短縮するために、InnoDB では、サーバーの停止時にバッファプールごとに最近使用されたページの割合が保存され、サーバーの起動時にこれらのページがリストアされます。最近使用されたページのうち、格納されたページの割合は、`innodb_buffer_pool_dump_pct` 構成オプションによって定義されます。

ビジー状態のサーバーを再起動した後、バッファプール内にあったディスクページがメモリーに戻されるため (同じデータがクエリー、更新などされるため)、通常はスループットが急激に向上するウォームアップ期間があります。起動時にバッファプールをリストアする機能により、DML 操作が対応する行にアクセスするのを待機するのではなく、再起動前にバッファプールにあったディスクページをリロードすることでウォームアップ期間が短縮されます。また、I/O リクエストは大規模なバッチで実行できるため、I/O 全体が高速になります。ページのロードはバックグラウンドで行われ、データベースの起動は遅延しません。

停止時にバッファプールの状態を保存して起動時にリストアする以外に、サーバーの実行中にいつでもバッファプールの状態を保存およびリストアできます。たとえば、安定したスループットに達した後、安定したワークロードでバッファプールの状態を保存できます。また、レポートまたはメンテナンスジョブを実行した後、これらの操作のみに必要なデータページをバッファプールに移動した後、またはその他の非標準ワークロードを実行した後、以前のバッファプールの状態をリストアすることもできます。

バッファプールのサイズは GB ですが、InnoDB がディスクに保存するバッファプールデータは比較によって小さいです。該当するページを見つけるために必要なテーブルスペース ID とページ ID だけがディスクに保存されます。この情報は、`INNODB_BUFFER_PAGE_LRU_INFORMATION_SCHEMA` テーブルから取得されます。デフォルトでは、テーブルスペース ID とページ ID のデータは、InnoDB データディレクトリに保存される `ib_buffer_pool` という名前のファイル内に保存されます。ファイル名と場所は、`innodb_buffer_pool_filename` 構成パラメータを使用して変更できます。

データは通常のデータベース操作と同様にキャッシュされ、バッファプールからエージアウトされるため、ディスクページが最近更新された場合や、ロードされていないデータが DML 操作に含まれている場合は、問題はありません。ロードメカニズムは、すでに存在しないリクエストされたページをスキップします。

ベースとなるメカニズムには、ダンプおよびロード操作を実行するためにディスパッチされるバックグラウンドスレッドが含まれています。

圧縮テーブルからのディスクページは、その圧縮された形式でバッファプールにロードされます。DML 操作中にページコンテンツにアクセスすると、ページは通常どおりに圧縮解除されます。ページの圧縮解除は CPU を大量に消費するプロセスであるため、バッファプールのリストア操作を実行する単一のスレッドではなく、接続スレッドで操作を実行すると同時実行性が向上します。

バッファプールの状態の保存と復元に関連する操作については、次のトピックで説明します:

- [バッファプールページのダンプ率の構成](#)
- [シャットダウン時のバッファプールの状態の保存と起動時の復元](#)
- [バッファプールの状態をオンラインで保存および復元](#)
- [バッファプールのダンプの進行状況の表示](#)
- [バッファプールのロードの進行状況の表示](#)

- [バッファプールのロード操作の中止](#)
- [パフォーマンススキーマを使用したバッファプールのロード進行状況の監視](#)

バッファプールページのダンプ率の構成

バッファプールからページをダンプする前に、`innodb_buffer_pool_dump_pct` オプションを設定することによって、ダンプする最後に使用されたバッファプールページの割合を構成できます。サーバーの実行中にバッファプールページをダンプする場合は、このオプションを動的に構成できます:

```
SET GLOBAL innodb_buffer_pool_dump_pct=40;
```

サーバーの停止時にバッファプールページをダンプする場合は、構成ファイルで `innodb_buffer_pool_dump_pct` を設定します。

```
[mysqld]  
innodb_buffer_pool_dump_pct=40
```

`innodb_buffer_pool_dump_pct` のデフォルト値は 25 です (最近使用されたページの 25% をダンプします)。

シャットダウン時のバッファプールの状態の保存と起動時の復元

サーバーの停止時にバッファプールの状態を保存するには、サーバーを停止する前に次のステートメントを発行します:

```
SET GLOBAL innodb_buffer_pool_dump_at_shutdown=ON;
```

`innodb_buffer_pool_dump_at_shutdown` はデフォルトで有効になっています。

サーバーの起動時にバッファプールの状態を復元するには、サーバーの起動時に `--innodb-buffer-pool-load-at-startup` オプションを指定します:

```
mysqld --innodb-buffer-pool-load-at-startup=ON;
```

`innodb_buffer_pool_load_at_startup` はデフォルトで有効になっています。

バッファプールの状態をオンラインで保存および復元

MySQL サーバーの実行中にバッファプールの状態を保存するには、次のステートメントを発行します:

```
SET GLOBAL innodb_buffer_pool_dump_now=ON;
```

MySQL の実行中にバッファプールの状態を復元するには、次のステートメントを発行します:

```
SET GLOBAL innodb_buffer_pool_load_now=ON;
```

バッファプールのダンプの進行状況の表示

バッファプールの状態をディスクに保存するときの進行状況を表示するには、次のステートメントを発行します:

```
SHOW STATUS LIKE 'InnoDB_buffer_pool_dump_status';
```

操作がまだ開始されていない場合は、「not started」が返されます。操作が完了している場合は、完了時間が出力されます (たとえば、Finished at 110505 12:18:02)。操作が進行中である場合は、ステータス情報が表示されます (たとえば、Dumping buffer pool 5/7, page 237/2873)。

バッファプールのロードの進行状況の表示

バッファプールのロード時に進行状況を表示するには、次のステートメントを発行します:

```
SHOW STATUS LIKE 'InnoDB_buffer_pool_load_status';
```

操作がまだ開始されていない場合は、「not started」が返されます。操作が完了している場合は、完了時間が出力されます (たとえば、Finished at 110505 12:23:24)。操作が進行中である場合は、ステータス情報が表示されます (たとえば、Loaded 123/22301 pages)。

バッファプールのロード操作の中止

バッファプールのロード操作を中止するには、次のステートメントを発行します:

```
SET GLOBAL innodb_buffer_pool_load_abort=ON;
```

パフォーマンススキーマを使用したバッファプールのロード進行状況の監視

[Performance Schema](#) を使用して、バッファプールのロードの進行状況をモニターできます。

次の例は、[stage/innodb/buffer pool load](#) ステージイベントインストゥルメントおよび関連するコンシューマテーブルを有効にして、バッファプールのロード進行状況を監視する方法を示しています。

この例で使用されるバッファプールのダンプおよびロード手順については、[セクション15.8.3.6「バッファプールの状態の保存と復元」](#)を参照してください。パフォーマンススキーマステージイベントインストゥルメントおよび関連コンシューマについては、[セクション27.12.5「パフォーマンススキーマステージイベントテーブル」](#)を参照してください。

1. [stage/innodb/buffer pool load](#) インストゥルメントを有効にします:

```
mysql> UPDATE performance_schema.setup_instruments SET ENABLED = 'YES'
WHERE NAME LIKE 'stage/innodb/buffer%';
```

2. ステージイベントコンシューマテーブル ([events_stages_current](#)、[events_stages_history](#) および [events_stages_history_long](#) を含む) を有効にします。

```
mysql> UPDATE performance_schema.setup_consumers SET ENABLED = 'YES'
WHERE NAME LIKE '%stages%';
```

3. [innodb_buffer_pool_dump_now](#) を有効にして、現在のバッファプールの状態をダンプします。

```
mysql> SET GLOBAL innodb_buffer_pool_dump_now=ON;
```

4. バッファプールダンプステータスをチェックして、操作が完了したことを確認します。

```
mysql> SHOW STATUS LIKE 'InnoDB_buffer_pool_dump_status\G
***** 1. row *****
Variable_name: InnoDB_buffer_pool_dump_status
Value: Buffer pool(s) dump completed at 150202 16:38:58
```

5. [innodb_buffer_pool_load_now](#) を有効にしてバッファプールをロードします:

```
mysql> SET GLOBAL innodb_buffer_pool_load_now=ON;
```

6. パフォーマンススキーマ [events_stages_current](#) テーブルをクエリーして、バッファプールのロード操作の現在のステータスを確認します。 [WORK_COMPLETED](#) カラムには、ロードされたバッファプールページの数が表示されます。 [WORK_ESTIMATED](#) カラムには、残りの作業の推定がページ単位で表示されます。

```
mysql> SELECT EVENT_NAME, WORK_COMPLETED, WORK_ESTIMATED
FROM performance_schema.events_stages_current;
+-----+-----+-----+
| EVENT_NAME          | WORK_COMPLETED | WORK_ESTIMATED |
+-----+-----+-----+
| stage/innodb/buffer pool load |          5353 |          7167 |
+-----+-----+-----+
```

バッファプールのロード操作が完了すると、[events_stages_current](#) テーブルは空のセットを返します。この場合、[events_stages_history](#) テーブルをチェックして、完了したイベントのデータを表示できます。例:

```
mysql> SELECT EVENT_NAME, WORK_COMPLETED, WORK_ESTIMATED
FROM performance_schema.events_stages_history;
+-----+-----+-----+
| EVENT_NAME          | WORK_COMPLETED | WORK_ESTIMATED |
+-----+-----+-----+
| stage/innodb/buffer pool load |          7167 |          7167 |
+-----+-----+-----+
```


注記

`innodb_buffer_pool_load_at_startup` を使用して起動時にバッファプールをロードするときに、パフォーマンススキーマを使用してバッファプールのロードの進行状況をモニターすることもできます。この場合、起動時に `stage/innodb/buffer pool load` インストールメントおよび関連コンシューマを有効にする必要があります。詳細については、[セクション 27.3「パフォーマンススキーマ起動構成」](#)を参照してください。

15.8.3.7 コアファイルからのバッファプールページの除外

コアファイルには、実行中のプロセスのステータスとメモリーイメージが記録されます。バッファプールはメインメモリー内に存在し、実行中のプロセスのメモリーイメージがコアファイルにダンプされるため、`mysqld` プロセスが終了すると、バッファプールが大きいシステムで大きなコアファイルが生成される可能性があります。

大規模なコアファイルは、書込みにかかる時間、それらが消費するディスク領域の量、大規模なファイルの転送に関連する課題など、様々な理由で問題になる可能性があります。

コアファイルサイズを減らすには、`innodb_buffer_pool_in_core_file` 変数を無効にして、コアダンプからバッファプールページを省略します。`innodb_buffer_pool_in_core_file` 変数は MySQL 8.0.14 で導入され、デフォルトで有効になっています。

デバッグ目的で組織内外で共有される可能性のあるコアファイルにデータベースページをダンプすることに懸念がある場合は、セキュリティの観点からバッファプールページを除外することも望ましい場合があります。

注記

一部のデバッグシナリオでは、`mysqld` プロセスの停止時にバッファプールページに存在するデータへのアクセスが有益な場合があります。バッファプールページを含めるが除外するかが疑わしい場合は、MySQL サポートに問い合わせてください。

`innodb_buffer_pool_in_core_file` の無効化は、`core_file` 変数が有効で、オペレーティングシステムが `madvise()` システムコールに対する `MADV_DONTDUMP` の POSIX 以外の拡張機能をサポートしている場合にのみ有効になります。これは Linux 3.4 以降でサポートされています。`MADV_DONTDUMP` 拡張機能を使用すると、指定した範囲のページがコアダンプから除外されます。

オペレーティングシステムで `MADV_DONTDUMP` 拡張機能がサポートされている場合は、`--core-file` および `--innodb-buffer-pool-in-core-file=OFF` オプションを使用してサーバーを起動し、バッファプールページなしでコアファイルを生成します。

```
shell> mysqld --core-file --innodb-buffer-pool-in-core-file=OFF
```

`core_file` 変数は読み取り専用で、デフォルトで無効になっています。これを有効にするには、起動時に `--core-file` オプションを指定します。`innodb_buffer_pool_in_core_file` 変数は動的です。起動時に指定するか、`SET` ステートメントを使用して実行時に構成できます。

```
mysql> SET GLOBAL innodb_buffer_pool_in_core_file=OFF;
```

`innodb_buffer_pool_in_core_file` 変数が無効になっているが、`MADV_DONTDUMP` がオペレーティングシステムでサポートされていない場合、または `madvise()` 障害が発生した場合は、MySQL サーバーのエラーログに警告が書き込まれ、`core_file` 変数は、意図せずバッファプールページを含むコアファイルが書き込まれないように無効になります。読み取り専用 `core_file` 変数が無効になった場合は、サーバーを再起動して再度有効にする必要があります。

次のテーブルに、コアファイルが生成されるかどうか、およびコアファイルにバッファプールページが含まれるかどうかを決定する構成シナリオと `MADV_DONTDUMP` サポートシナリオを示します。

表 15.4 コアファイルの構成シナリオ

<code>core_file</code> 変数	<code>innodb_buffer_pool_in_core_file</code> 変数	<code>madvise()</code> <code>MADV_DONTDUMP</code> のサポート	結果
OFF (デフォルト)	結果に関連しない	結果に関連しない	コアファイルは生成されません

core_file 変数	innodb_buffer_pool_in_core_file 変数	madvise() MADV_DONTDUMP のサポート	結果
ON	ON (デフォルト)	結果に関連しない	コアファイルはバッファプールページで生成されます
ON	OFF	はい	コアファイルはバッファプールページなしで生成されます
ON	OFF	いいえ	コアファイルが生成されず、core_file が無効になり、サーバーエラーログに警告が書き込まれます

innodb_buffer_pool_in_core_file 変数を無効にして達成されるコアファイルサイズの縮小は、バッファプールのサイズによって異なりますが、InnoDB ページサイズの影響も受けます。ページサイズが小さいほど、同じ量のデータに必要なページが増え、ページが多いほどページメタデータが増えます。次のテーブルに、ページサイズが異なる 1GB バッファプールで表示されるサイズ削減の例を示します。

表 15.5 バッファプールページが含まれ、除外されているコアファイルのサイズ

innodb_page_size 設定	含まれるバッファプールページ (innodb_buffer_pool_in_core_file=ON)	除外されたバッファプールページ (innodb_buffer_pool_in_core_file=OFF)
4KB	2.1GB	0.9GB
64KB	1.7GB	0.7GB

15.8.4 InnoDB のスレッド並列性の構成

InnoDB は、オペレーティングシステムスレッドを使用して、ユーザートランザクションからの要求を処理します。(トランザクションは、コミットまたはロールバックする前に、InnoDB に多数の要求を発行する可能性があります。) コンテキストスイッチングが効率的な、マルチコアプロセッサを備えた最新のオペレーティングシステムおよびサーバーでは、並列スレッドの数を制限することなく、ほとんどのワークロードが適切に動作します。

スレッド間のコンテキストスイッチングを最小限に抑えることが役立つ状況では、InnoDB はいくつかの手法を使用して、並列実行中のオペレーティングシステムスレッドの数(したがって、一度に処理される要求の数)を制限できます。InnoDB がユーザーセッションからの新しい要求を受信したとき、並列実行中のスレッドの数が事前に定義された制限に達している場合、その新しい要求は再試行の前に短時間だけスリープします。スリープのあとに再スケジューリングできない要求は先入れ先出しキューに入れられ、最終的に処理されます。ロックを待機しているスレッドは、並列実行中のスレッドの数にカウントされません。

並列スレッドの数は、構成パラメータ innodb_thread_concurrency を設定することによって制限できます。実行中のスレッドの数がこの制限に達すると、追加のスレッドはキューに入れられる前に、構成パラメータ innodb_thread_sleep_delay で設定されたマイクロ秒数だけスリープします。

構成オプション innodb_adaptive_max_sleep_delay を innodb_thread_sleep_delay に許可する最大値に設定すると、InnoDB は現在のスレッドスケジューリングアクティビティに応じて innodb_thread_sleep_delay を自動的に上下に調整します。この動的な調整は、システムにかかる負荷が軽い期間や、システムがほぼ容量いっぱい動作している期間に、スレッドスケジューリングメカニズムがスムーズに機能するのに役立ちます。

innodb_thread_concurrency のデフォルト値や、並列スレッドの数に対する暗黙的なデフォルトの制限は、MySQL および InnoDB のさまざまなリリースで変更されてきました。innodb_thread_concurrency のデフォルト値は 0 であるため、デフォルトでは同時に実行するスレッドの数に制限はありません。

InnoDB では、同時スレッド数が制限されている場合にのみスレッドがスリープします。スレッドの数に対して制限がない場合は、すべてが均等に競合してスケジューリングされます。つまり、innodb_thread_concurrency が 0 である場合は、innodb_thread_sleep_delay の値は無視されます。

スレッドの数に対して制限がある (innodb_thread_concurrency > 0 である) 場合、InnoDB は、1 つの SQL ステートメントの実行中に発行された複数の要求が innodb_thread_concurrency で設定された制限に従うことなく InnoDB に入

ることを許可することによって、コンテキストスイッチングのオーバーヘッドを削減します。SQL ステートメント (結合など) は InnoDB 内の複数の行操作で構成されている可能性があるため、InnoDB は、スレッドが最小限のオーバーヘッドで繰り返しスケジュールされることを許可する指定された数の「チケット」を割り当てます。

新しい SQL ステートメントが開始されたとき、スレッドにはチケットがないため、`innodb_thread_concurrency` に従う必要があります。スレッドが InnoDB に入るときを許可されると、そのスレッドには、行操作を実行するためにあとで InnoDB に入るときに使用できる複数のチケットが割り当てられます。それらのチケットが使い果たされた場合、スレッドは削除され、ふたたび `innodb_thread_concurrency` に従います。それにより、そのスレッドが、待機中のスレッドの先入れ先出しキューに戻される可能性があります。スレッドがふたたび InnoDB に入るときを許可されると、チケットが再度割り当てられます。割り当てられるチケットの数は、グローバルオプション `innodb_concurrency_tickets` (デフォルトは 5000) によって指定されます。ロックを待機しているスレッドには、そのロックが使用可能になるとチケットが 1 つ与えられます。

これらの変数の正しい値は、環境やワークロードによって異なります。アプリケーションでどのような値が機能するかを確認するには、さまざまな値を試してください。並列実行中のスレッドの数を制限する前に、マルチコアおよびマルチプロセッサコンピュータ上の InnoDB のパフォーマンスを向上させる可能性のある構成オプション (`innodb_adaptive_hash_index` など) を確認してください。

MySQL のスレッド処理に関する一般的なパフォーマンス情報については、[セクション 5.1.12.1 「接続インタフェース」](#) を参照してください。

15.8.5 InnoDB バックグラウンド I/O スレッドの数の構成

InnoDB では、バックグラウンド `threads` を使用して様々なタイプの I/O リクエストを処理します。`innodb_read_io_threads` および `innodb_write_io_threads` 構成パラメータを使用して、データページで I/O の読取りおよび書き込みを行うバックグラウンドスレッドの数を構成できます。これらのパラメータは、読取りおよび書き込みリクエストにそれぞれ使用されるバックグラウンドスレッドの数を示します。これらは、サポートされるすべてのプラットフォームで有効です。これらのパラメータの値は、MySQL オプションファイル (`my.cnf` または `my.ini`) で設定できます。値を動的に変更することはできません。これらのパラメータのデフォルト値は 4 で、許容値の範囲は 1-64 です。

ハイエンドシステムで InnoDB をよりスケラブルにするためのこれらの構成オプションの目的。各バックグラウンドスレッドは、保留中の I/O 要求を最大 256 個処理できます。バックグラウンド I/O の主なソースは、`read-ahead` リクエストです。InnoDB は、ほとんどのバックグラウンドスレッドが同じように動作するように、受信リクエストのロードバランシングを試みます。また、InnoDB は、同じエクステンツから同じスレッドに読取りリクエストを割り当てようとし、リクエストを結合する可能性を高めます。ハイエンドの I/O サブシステムがあり、`SHOW ENGINE INNODB STATUS` 出力に $64 \times \text{innodb_read_io_threads}$ を超える保留中の読取りリクエストが表示される場合、`innodb_read_io_threads` の値を増やすことでパフォーマンスが向上する可能性があります。

Linux システムでは、InnoDB はデフォルトで非同期 I/O サブシステムを使用して、データファイルページの先読みおよび書き込みリクエストを実行します。これにより、InnoDB バックグラウンドスレッドがこれらのタイプの I/O リクエストを処理する方法が変更されます。詳細は、[セクション 15.8.6 「Linux での非同期 I/O の使用」](#) を参照してください。

InnoDB I/O のパフォーマンスの詳細は、[セクション 8.5.8 「InnoDB ディスク I/O の最適化」](#) を参照してください。

15.8.6 Linux での非同期 I/O の使用

InnoDB は、Linux で非同期 I/O サブシステム (ネイティブ AIO) を使用して、データファイルページの先読みおよび書き込みリクエストを実行します。この動作は、Linux システムにのみ適用され、デフォルトで有効になっている `innodb_use_native_aio` 構成オプションによって制御されます。他の Unix に似たシステムでは、InnoDB は同期 I/O のみを使用します。従来、InnoDB では、Windows システムで非同期 I/O のみが使用されていました。Linux で非同期 I/O サブシステムを使用するには、`libaio` ライブラリが必要です。

同期 I/O、クエリースレッドは I/O リクエストをキューに入れ、InnoDB バックグラウンドスレッドはキューに入れたリクエストを一度に 1 つずつ取得し、それぞれに対して同期 I/O コールを発行します。I/O リクエストが完了し、I/O コールが返されると、リクエストを処理している InnoDB バックグラウンドスレッドは I/O 完了ルーチンをコールし、次のリクエストを処理するために戻ります。パラレルに処理できるリクエストの数は n で、 n は InnoDB バックグラウンドスレッドの数です。InnoDB バックグラウンドスレッドの数は、`innodb_read_io_threads` および

`innodb_write_io_threads` によって制御されます。 [セクション15.8.5「InnoDB バックグラウンド I/O スレッドの数の構成」](#) を参照してください。

ネイティブ AIO では、クエリースレッドは I/O リクエストをオペレーティングシステムに直接ディスパッチするため、バックグラウンドスレッドの数によって課される制限は削除されます。 InnoDB バックグラウンドスレッドは、I/O イベントが完了したリクエストを通知するのを待機します。 リクエストが完了すると、バックグラウンドスレッドは I/O 完了ルーチンを呼び出し、I/O イベントの待機を再開します。

ネイティブ AIO の利点は、多くの保留中の読取り/書き込みを `SHOW ENGINE INNODB STATUS\G` 出力に表示する I/O-bound システムのスケーラビリティです。 ネイティブ AIO を使用する場合の並列処理の増加は、I/O スケジューラのタイプまたはディスクアレイコントローラのプロパティが I/O のパフォーマンスに大きく影響することを意味します。

大量の I/O-bound システムに対するネイティブ AIO の潜在的なデメリットは、オペレーティングシステムに一度にディスパッチされる I/O 書き込みリクエストの数を制御できないことです。 パラレル処理のためにオペレーティングシステムにディスパッチされた I/O 書き込みリクエストが多すぎると、I/O アクティビティおよびシステム機能の量に応じて、I/O 読取り文が発生する場合があります。

OS の非同期 I/O サブシステムに問題があるために InnoDB を起動できない場合は、`innodb_use_native_aio=0` を使用してサーバーを起動できます。 このオプションは、`tmpfs` で非同期 I/O をサポートしていない `tmpdir` の場所、`tmpfs` ファイルシステム、Linux カーネルの組合せなどの潜在的な問題が InnoDB によって検出された場合にも、起動時に自動的に無効になることがあります。

15.8.7 InnoDB I/O Capacity の構成

InnoDB マスタースレッドおよびその他のスレッドは、さまざまなタスクをバックグラウンドで実行します。これらのほとんどは、バッファプールからデータページをフラッシュしたり、変更バッファから適切なセカンダリインデックスに変更を書き込みたりするなど、I/O に関連しています。 InnoDB は、サーバーの通常の動作に悪影響を与えない方法でこれらのタスクを実行しようとします。 使用可能な I/O 帯域幅を見積もり、そのアクティビティをチューニングして使用可能な容量を利用しようとします。

`innodb_io_capacity` 変数は、InnoDB で使用可能な全体的な I/O 容量を定義します。 これは、システムが秒当たりに実行できる I/O 操作の数 (IOPS) におよそ設定する必要があります。 `innodb_io_capacity` が設定されている場合、InnoDB は、設定された値に基づいてバックグラウンドタスクに使用可能な I/O 帯域幅を見積もります。

`innodb_io_capacity` は 100 以上の値に設定できます。 デフォルト値は 200 です。 通常、100 付近の値は、最大 7200 RPM のハードドライブなどのコンシューマレベルのストレージデバイスに適しています。 より高速なハードドライブ、RAID 構成、およびソリッドステートドライブ (SSD) は、より高い値からメリットを得られます。

理想的には、設定を実用的なものとして低く保ちますが、バックグラウンドアクティビティが遅れるほど低くしないでください。 値が大きすぎる場合、データはバッファプールから削除され、キャッシュするにはバッファの変更が早すぎるため、大きな利点があります。 I/O レートが高いビジネシステムの場合は、高い値を設定すると、高い行変更率に関連付けられたバックグラウンドメンテナンス作業をサーバーが処理できるようになります。 通常、この値は、InnoDB I/O に使用されるドライブ数の関数として増やすことができます。 たとえば、複数のディスクまたは SSD を使用するシステムで値を増やすことができます。

通常、デフォルト設定の 200 は、ローエンド SSD には十分です。 高エンドのバス接続 SSD の場合は、1000 などのより高い設定を検討してください。 個々の 5400 RPM または 7200 RPM ドライブを搭載したシステムでは、値を 100 に減らすことができます。これは、約 100 IOPS を実行できる古い世代のディスクドライブで使用可能な秒あたりの I/O 操作数 (IOPS) の推定比率を表します。

百万などの高い値を指定できますが、実際にはこのような大きな値の利点はほとんどありません。 通常、ワークロードに対して低い値が不十分であることが確実でないかぎり、20000 より大きい値はお勧めしません。

`innodb_io_capacity` をチューニングする場合は、書き込みワークロードを検討してください。 書き込みワークロードが大きいシステムでは、設定を大きくするとメリットが得られる可能性があります。 書き込みワークロードが小さいシステムでは、低い設定で十分な場合があります。

`innodb_io_capacity` 設定はバッファプールごとのインスタンス設定ではありません。 使用可能な I/O 容量は、アクティビティをフラッシュするためにバッファプールインスタンス間で均等に分散されます。

`innodb_io_capacity` の値は、MySQL オプションファイル (`my.cnf` または `my.ini`) で設定するか、`SET GLOBAL` ステートメントを使用して実行時に変更できます。これには、グローバルシステム変数の設定に十分な権限が必要です。セクション 5.1.9.1 「システム変数権限」を参照してください。

チェックポイントでの I/O 容量の無視

`innodb_flush_sync` 変数はデフォルトで有効になっており、`checkpoints` で発生する I/O アクティビティのバースト中に `innodb_io_capacity` 設定が無視されます。 `innodb_io_capacity` 設定で定義された I/O レートに準拠するには、`innodb_flush_sync` を無効にします。

`innodb_flush_sync` の値は、MySQL オプションファイル (`my.cnf` または `my.ini`) で設定するか、`SET GLOBAL` ステートメントを使用して実行時に変更できます。これには、グローバルシステム変数の設定に十分な権限が必要です。セクション 5.1.9.1 「システム変数権限」を参照してください。

I/O 容量の最大値の構成

フラッシュアクティビティが遅れている場合、InnoDB は `innodb_io_capacity` 変数で定義されているよりも高い速度の I/O 操作/秒 (IOPS) で、より積極的にフラッシュできます。 `innodb_io_capacity_max` 変数は、このような状況で InnoDB バックグラウンドタスクによって実行される IOPS の最大数を定義します。

起動時に `innodb_io_capacity` 設定を指定し、`innodb_io_capacity_max` の値を指定しない場合、`innodb_io_capacity_max` のデフォルトは `innodb_io_capacity` または 2000 のどちらか大きい方の値の 2 倍になります。

`innodb_io_capacity_max` を構成する場合、多くの場合、`innodb_io_capacity` が適切な開始点になります。デフォルト値 2000 は、SSD または複数の通常のディスクドライブを使用するワークロードを対象としています。SSD または複数のディスクドライブを使用しないワークロードでは、2000 の設定が高すぎる可能性があり、フラッシュが多すぎる可能性があります。単一の標準ディスクドライブの場合は、200 から 400 の間を設定をお勧めします。ハイエンドのバス接続 SSD の場合は、2500 などの高い設定を検討してください。 `innodb_io_capacity` 設定と同様に、この設定は実用的ではなく、InnoDB が IOPS の速度を `innodb_io_capacity` 設定を超えて十分に拡張できないように低くしてください。

`innodb_io_capacity_max` をチューニングする場合は、書込みワークロードを検討してください。書込みワークロードが大きいシステムでは、設定を大きくするとメリットが得られる場合があります。書込みワークロードが小さいシステムでは、低い設定で十分な場合があります。

`innodb_io_capacity_max` は、`innodb_io_capacity` 値より小さい値に設定できません。

`SET` ステートメント (`SET GLOBAL innodb_io_capacity_max=DEFAULT`) を使用して `innodb_io_capacity_max` を `DEFAULT` に設定すると、`innodb_io_capacity_max` が最大値に設定されます。

`innodb_io_capacity_max` 制限は、すべてのバッファプールインスタンスに適用されます。バッファプールごとのインスタンス設定ではありません。

15.8.8 スピンロックのポーリングの構成

InnoDB `mutexes` および `rw-locks` は通常、短い間隔で予約されています。マルチコアシステムでは、スレッドがスリープする前に `mutex` または `rw-lock` を一定期間取得できるかどうかを継続的にチェックする方が効率的です。 `mutex` または `rw-lock` がこの期間中に使用可能になった場合、スレッドは同じタイムスライスですぐに続行できます。ただし、相互排他ロックや `rw-lock` などの共有オブジェクトを複数のスレッドで頻りにポーリングすると、「cache ping pong」が発生する可能性があり、その結果プロセスは互いの一部を無効にします。InnoDB では、ポーリング間のランダムな遅延を強制してポーリングアクティビティを非同期化することで、この問題を最小限に抑えます。ランダム遅延はスピン待機ループとして実装されます。

スピン待機ループの継続時間は、ループ内で発生する `PAUSE` 命令の数によって決まります。その数値を生成するには、0 から `innodb_spin_wait_delay` 値までの整数をランダムに選択し、その値に 50 を掛けます。(乗数値 50 は、MySQL 8.0.16 の前にハードコードされ、その後構成できます。) たとえば、`innodb_spin_wait_delay` 設定が 6 の場合、整数は次の範囲からランダムに選択されます:

```
{0,1,2,3,4,5}
```


選択した整数に 50 が乗算されるため、PAUSE 命令の値は 6 つのいずれかになります：

```
{0,50,100,150,200,250}
```

この値セットの場合、250 はスピン待機ループで発生する PAUSE 命令の最大数です。 `innodb_spin_wait_delay` を 5 に設定すると、使用可能な 5 つの値の `{0,50,100,150,200}` のセットになります。200 は PAUSE 命令の最大数などです。このように、`innodb_spin_wait_delay` 設定はスピンロックポーリング間の最大遅延を制御します。

すべてのプロセッサコアが高速なキャッシュメモリーを共有するシステムでは、この最大の遅延を短くするか、または `innodb_spin_wait_delay=0` を設定してビジーループを完全に無効にすることができます。複数のプロセッサチップを備えたシステムでは、キャッシュを無効にすると重大な影響を与える場合があるため、最大の遅延を増やすことができます。

100MHz Pentium 紀元では、`innodb_spin_wait_delay` ユニットはマイクロ秒に相当するように調整されました。この時間等価は保持されませんでした。PAUSE 命令が比較的長いプロセッサの Skylake 生成が導入されるまで、ほかの CPU 命令と比較してプロセッササイクルの観点から PAUSE 命令の期間はかなり一定のままでした。`innodb_spin_wait_pause_multiplier` 変数は、PAUSE 命令の期間の違いを説明する方法を提供するために、MySQL 8.0.16 で導入されました。

`innodb_spin_wait_pause_multiplier` 変数は、PAUSE 命令の値のサイズを制御します。たとえば、`innodb_spin_wait_delay` 設定が 6 の場合、`innodb_spin_wait_pause_multiplier` 値を 50 (デフォルトおよび以前にハードコードされた値) から 5 に減らすと、小さい PAUSE 命令値のセットが生成されます：

```
{0,5,10,15,20,25}
```

PAUSE 命令の値を増減できるため、プロセッサアーキテクチャーごとに InnoDB を微調整できます。たとえば、PAUSE 命令の値が小さいほど、PAUSE 命令が比較的長いプロセッサアーキテクチャーに適しています。

`innodb_spin_wait_delay` および `innodb_spin_wait_pause_multiplier` 変数は動的です。これらは、MySQL オプションファイルで指定することも、`SET GLOBAL` ステートメントを使用して実行時に変更することもできます。実行時に変数を変更するには、グローバルシステム変数を設定するのに十分な権限が必要です。[セクション5.1.9.1「システム変数権限」](#)を参照してください。

15.8.9 ページ構成

InnoDB では、SQL ステートメントを使用して行を削除しても、データベースからすぐには行が物理的に削除されません。行とそのインデックスレコードは、削除のために書き込まれた undo ログレコードが InnoDB によって破棄された場合にのみ物理的に削除されます。この削除操作は、行がマルチバージョン同時実行制御 (MVCC) またはローバックに不要になった後にのみ実行され、ページと呼ばれます。

ページは定期的に行われます。履歴リストから undo ログページを解析して処理します。これは、InnoDB トランザクションシステムによってメンテナンスされるコミット済トランザクションの undo ログページのリストです。ページすると、undo ログページは処理後に履歴リストから解放されます。

ページスレッドの構成

ページ操作は、1 つ以上のページスレッドによってバックグラウンドで実行されます。ページスレッドの数は、`innodb_purge_threads` 変数によって制御されます。デフォルト値は 4 です。DML アクションが単一のテーブルまたはいくつかのテーブルに集中している場合は、スレッドがテーブルにアクセスするために互いに競合しないように、設定を低くしておきます。DML 操作が多数のテーブルに分散している場合は、この設定を増やします。ページスレッドの最大数は 32 です。

`innodb_purge_threads` 設定は、許可されるページスレッドの最大数です。ページシステムは、必要に応じてページスレッドの数を自動的に調整します。

ページバッチサイズの構成

`innodb_purge_batch_size` 変数は、履歴リストから一度に解析および処理をページする undo ログページの数を定義します。デフォルト値は 300 です。マルチスレッドページ構成では、コーディネータページスレッドは `innodb_purge_batch_size` を `innodb_purge_threads` で除算し、その数のページを各ページスレッドに割り当てます。

ページシステムでは、不要になった undo ログページも解放されます。undo ログを使用して 128 回ずつ反復します。innodb_purge_batch_size 変数では、バッチで解析および処理される undo ログページの数の定義に加えて、undo ログを介して 128 回反復するたびにページによって解放される undo ログページの数を定義します。

innodb_purge_batch_size 変数は、高度なパフォーマンスチューニングおよび実験を目的としています。ほとんどのユーザーは、innodb_purge_batch_size をデフォルト値から変更する必要はありません。

最大ページラグの構成

innodb_max_purge_lag 変数は、必要な最大ページラグを定義します。ページラグが innodb_max_purge_lag のしきい値を超えると、INSERT、UPDATE および DELETE 操作に遅延が課され、ページ操作がキャッチアップされる時間が許可されます。デフォルト値は 0 です。これは、最大ページラグおよび遅延がないことを意味します。

InnoDB トランザクションシステムでは、UPDATE または DELETE 操作で削除のマークが付けられたインデックスレコードを含むトランザクションのリストが保持されます。リストの長さはページラグです。MySQL 8.0.14 より前では、ページラグ遅延は次の式によって計算されるため、最小遅延は 5000 マイクロ秒になりました：

```
(purge_lag/innodb_max_purge_lag - 0.5) * 10000
```

MySQL 8.0.14 では、ページラグ遅延は次の改訂された式によって計算され、最小遅延が 5 マイクロ秒に削減されます。5 マイクロ秒の遅延は、最新のシステムに適しています。

```
(purge_lag/innodb_max_purge_lag - 0.9995) * 10000
```

遅延は、ページバッチの開始時に計算されます。

問題のあるワークロードの一般的な innodb_max_purge_lag 設定は 1000000 (100 万) で、トランザクションが小さく、サイズが 100 バイトのみで、100MB のページされていないテーブルの行を持つことができると想定されます。

ページラグは、SHOW ENGINE INNODB STATUS 出力の TRANSACTIONS セクションに History list length 値として表示されます。

```
mysql> SHOW ENGINE INNODB STATUS;
...
-----
TRANSACTIONS
-----
Trx id counter 0 290328385
Purge done for trx's n:o < 0 290315608 undo n:o < 0 17
History list length 20
```

通常、History list length は低い値で、通常は数千未満ですが、書き込み負荷の高いワークロードまたは長時間実行中のトランザクションは、読取り専用のトランザクションの場合でも増加する可能性があります。長時間実行トランザクションが原因で History list length が増加する理由は、REPEATABLE READ などの読取り一貫性トランザクション分離レベルでは、トランザクションの読取りビューが作成されたときと同じ結果を返す必要があるためです。したがって、InnoDB マルチバージョン同時実行性制御 (MVCC) システムは、そのデータに依存するすべてのトランザクションが完了するまで、undo ログにデータのコピーを保持する必要があります。次に、History list length が増加する可能性のある長時間実行トランザクションの例を示します：

- 大量の同時 DML があるときに `--single-transaction` オプションを使用する `mysqldump` 操作。
- `autocommit` を無効にし、明示的な `COMMIT` または `ROLLBACK` の発行を忘れた後の `SELECT` クエリーの実行。

ページラグが膨大になる極端な状況での過剰な遅延を防ぐために、innodb_max_purge_lag_delay 変数を設定して遅延を制限できます。innodb_max_purge_lag_delay 変数は、innodb_max_purge_lag しきい値を超えた場合に課される遅延の最大遅延をマイクロ秒単位で指定します。指定された innodb_max_purge_lag_delay 値は、innodb_max_purge_lag 式で計算された遅延期間の上限です。

ページおよび undo テーブルスペースの切捨て

ページシステムは、undo テーブルスペースの切捨ても行います。ページシステムが切り捨てる undo テーブルスペースを検索する頻度を制御するように innodb_purge_rseg_truncate_frequency 変数を構成できます。詳細は、undo テーブルスペースの切捨てを参照してください。

15.8.10 InnoDB のオプティマイザ統計の構成

このセクションでは、InnoDB テーブルの永続オプティマイザ統計および非永続オプティマイザ統計を構成する方法について説明します。

永続オプティマイザ統計はサーバーの再起動後も永続化されるため、[plan stability](#) の向上とクエリーのパフォーマンスの一貫性が向上します。永続オプティマイザ統計には、次のような利点があり、制御と柔軟性もあります：

- `innodb_stats_auto_recalc` 構成オプションを使用して、テーブルに対する大幅な変更後に統計を自動的に更新するかどうかを制御できます。
- `STATS_PERSISTENT`、`STATS_AUTO_RECALC` および `STATS_SAMPLE_PAGES` 句を `CREATE TABLE` および `ALTER TABLE` ステートメントとともに使用して、個々のテーブルのオプティマイザ統計を構成できます。
- `mysql.innodb_table_stats` テーブルおよび `mysql.innodb_index_stats` テーブルのオプティマイザ統計データをクエリーすることができます。
- `mysql.innodb_table_stats` テーブルおよび `mysql.innodb_index_stats` テーブルの `last_update` カラムを表示して、統計が最後に更新された日時を確認できます。
- `mysql.innodb_table_stats` テーブルおよび `mysql.innodb_index_stats` テーブルを手動で変更して、特定のクエリー最適化計画を強制したり、データベースを変更せずに代替計画をテストできます。

永続オプティマイザ統計機能は、デフォルトで有効になっています (`innodb_stats_persistent=ON`)。

非永続オプティマイザ統計は、各サーバーの再起動時および他の操作後にクリアされ、次のテーブルアクセスで再計算されます。その結果、統計の再計算時に異なる見積りが生成され、実行計画の選択肢やクエリーパフォーマンスの変動が生じる可能性があります。

このセクションでは、正確な統計と `ANALYZE TABLE` 実行時間のバランスを取る場合に役立つ可能性のある、`ANALYZE TABLE` の複雑さの見積りについても説明します。

15.8.10.1 永続的オプティマイザ統計のパラメータの構成

永続オプティマイザ統計機能は、統計をディスクに格納し、サーバーの再起動後も永続させることで [plan stability](#) を改善し、[optimizer](#) が特定のクエリーに対して一貫性のある選択を行う可能性を高めます。

オプティマイザ統計は、`innodb_stats_persistent=ON` または個々のテーブルが `STATS_PERSISTENT=1` で定義されている場合、ディスクに永続化されます。`innodb_stats_persistent` はデフォルトで有効になっています。

以前は、サーバーの再起動時および他のタイプの操作後にオプティマイザ統計がクリアされ、次のテーブルアクセスで再計算されていました。したがって、統計を再計算すると、クエリー実行計画の選択肢やクエリーパフォーマンスの変動につながる様々な見積りが生成される可能性があります。

永続統計は、`mysql.innodb_table_stats` テーブルおよび `mysql.innodb_index_stats` テーブルに格納されます。[InnoDB 永続的統計テーブル](#) を参照してください。

オプティマイザ統計をディスクに永続化しない場合は、[セクション15.8.10.2「非永続的オプティマイザ統計のパラメータの構成」](#) を参照してください

永続オプティマイザ統計の自動統計計算の構成

デフォルトで有効になっている `innodb_stats_auto_recalc` 変数は、テーブルが 10% を超える行に変更された場合に統計を自動的に計算するかどうかを制御します。テーブルの作成または変更時に `STATS_AUTO_RECALC` 句を指定して、個々のテーブルの自動統計再計算を構成することもできます。

自動統計再計算はバックグラウンドで行われるため、`innodb_stats_auto_recalc` が有効な場合でも、10% を超えるテーブルに影響を与える DML 操作を実行した直後に統計が再計算されないことがあります。場合によっては、統計の再計算が数秒遅れることがあります。最新の統計がすぐに必要な場合は、`ANALYZE TABLE` を実行して統計の同期 (フォアグラウンド) 再計算を開始します。

`innodb_stats_auto_recalc` が無効になっている場合は、インデックス付けされたカラムを大幅に変更した後に `ANALYZE TABLE` ステートメントを実行することで、オプティマイザ統計の正確性を確保できます。データのロー

ド後に実行する設定スクリプトに `ANALYZE TABLE` を追加し、アクティビティが少ないときにスケジュールで `ANALYZE TABLE` を実行することも検討できます。

既存のテーブルにインデックスを追加する場合、またはカラムを追加または削除する場合、`innodb_stats_auto_recalc` の値に関係なく、インデックス統計が計算されて `innodb_index_stats` テーブルに追加されます。

個々のテーブルのオプティマイザ統計パラメータの構成

`innodb_stats_persistent`、`innodb_stats_auto_recalc` および `innodb_stats_persistent_sample_pages` はグローバル変数です。これらのシステム全体の設定をオーバーライドし、個々のテーブルのオプティマイザ統計パラメータを構成するには、`CREATE TABLE` ステートメントまたは `ALTER TABLE` ステートメントで `STATS_PERSISTENT`、`STATS_AUTO_RECALC` および `STATS_SAMPLE_PAGES` 句を定義します。

- `STATS_PERSISTENT` では、InnoDB テーブルに対して `persistent statistics` を有効にするかどうかを指定します。値が `DEFAULT` の場合、テーブルの永続統計設定は `innodb_stats_persistent` 設定によって決定されます。1 の値を指定するとテーブルの永続統計が有効になり、0 の値を指定するとこの機能は無効になります。個々のテーブルの永続統計を有効にした後、`ANALYZE TABLE` を使用して、テーブルデータのロード後に統計を計算します。
- `STATS_AUTO_RECALC` では、`persistent statistics` を自動的に再計算するかどうかを指定します。値が `DEFAULT` の場合、テーブルの永続統計設定は `innodb_stats_auto_recalc` 設定によって決定されます。1 の値を指定すると、テーブルデータの 10% が変更されたときに統計が再計算されます。値が 0 の場合、テーブルの自動再計算は行われません。値 0 を使用する場合は、テーブルに大幅な変更を加えた後、`ANALYZE TABLE` を使用して統計を再計算します。
- `STATS_SAMPLE_PAGES` では、`ANALYZE TABLE` 操作などによって、インデックス付けされたカラムのカーディナリティおよびその他の統計が計算される場合にサンプリングするインデックスページの数を指定します。

次の `CREATE TABLE` の例では、3 つの句がすべて指定されています:

```
CREATE TABLE `t1` (  
  `id` int(8) NOT NULL auto_increment,  
  `data` varchar(255),  
  `date` datetime,  
  PRIMARY KEY (`id`),  
  INDEX `DATE_IX` (`date`)  
) ENGINE=InnoDB,  
  STATS_PERSISTENT=1,  
  STATS_AUTO_RECALC=1,  
  STATS_SAMPLE_PAGES=25;
```

InnoDB オプティマイザ統計でサンプリングされるページの数の構成

オプティマイザは、キー配分に関する推定 `statistics` を使用して、インデックスの相対 `selectivity` に基づいて実行計画のインデックスを選択します。`ANALYZE TABLE` などの操作を行うと、InnoDB は、インデックスの `カーディナリティ` を推定するためにテーブル上の各インデックスからランダムなページをサンプリングします。このサンプリング手法は、`random dive` と呼ばれます。

`innodb_stats_persistent_sample_pages` は、サンプリングされるページの数を制御します。実行時に設定を調整して、オプティマイザで使用される統計の見積りの品質を管理できます。デフォルト値は 20 です。次の問題が発生した場合は、設定の変更を検討してください:

1. 統計が十分ではなく、オプティマイザが最適でない計画を選択しています (`EXPLAIN` 出力を参照)。インデックスの実際のカーディナリティ (インデックスカラムで `SELECT DISTINCT` を実行して決定) を `mysql.innodb_index_stats` テーブルの見積りと比較することで、統計の正確性をチェックできます。

統計の精度が十分でないことが確認された場合は、統計の推定値が十分な精度になるまで `innodb_stats_persistent_sample_pages` の値を増やすようにしてください。ただし、`innodb_stats_persistent_sample_pages` を大きくしすぎると、`ANALYZE TABLE` の実行が遅くなる可能性があります。
2. `ANALYZE TABLE` が遅すぎる。この場合は、`ANALYZE TABLE` の実行時間が許容可能になるまで `innodb_stats_persistent_sample_pages` を減らすようにしてください。ただし、この値を小さくしすぎると、精度の低い統計および次善のクエリ実行計画という最初の問題につながる可能性があります。

統計の精度と `ANALYZE TABLE` の実行時間のバランスをとることができない場合は、`ANALYZE TABLE` の複雑さを減らすためにテーブル内のインデックス付きカラムの数を減らすか、またはパーティションの数を制限することを考慮してください。主キーカラムは一意でない各インデックスに追加されるため、テーブルの主キーのカラム数も考慮することが重要です。

関連情報については、[セクション15.8.10.3「InnoDB テーブルに対する ANALYZE TABLE の複雑さの推定」](#)を参照してください。

永続統計計算への削除マーク付きレコードの組込み

デフォルトでは、InnoDB は統計の計算時にコミットされていないデータを読み取ります。テーブルから行を削除するコミットされていないトランザクションの場合、行の見積りおよびインデックス統計の計算時に削除マークが付けられたレコードが除外されるため、`READ UNCOMMITTED` 以外のトランザクション分離レベルを使用してテーブルで同時に操作している他のトランザクションの実行計画が最適でなくなる可能性があります。このシナリオを回避するために、`innodb_stats_include_delete_marked` を有効にして、永続オプティマイザ統計の計算時に削除マーク付きレコードが含まれるようにできます。

`innodb_stats_include_delete_marked` が有効な場合、`ANALYZE TABLE` では、統計の再計算時に削除マークが付けられたレコードが考慮されます。

`innodb_stats_include_delete_marked` は、すべての InnoDB テーブルに影響するグローバル設定で、永続オプティマイザ統計にのみ適用されます。

InnoDB 永続的統計テーブル

永続的統計機能は、`innodb_table_stats` および `innodb_index_stats` という名前の、`mysql` データベース内の内部的に管理されているテーブルに依存します。これらのテーブルは、すべてのインストール、アップグレード、およびソースからのビルド手順で自動的に設定されます。

表 15.6 `innodb_table_stats` のカラム

カラム名	説明
<code>database_name</code>	データベース名
<code>table_name</code>	テーブル名、パーティション名、またはサブパーティション名
<code>last_update</code>	InnoDB が最後にこの行を更新した時間を示すタイムスタンプ
<code>n_rows</code>	テーブル内の行数
<code>clustered_index_size</code>	プライマリインデックスのサイズ (ページ数)
<code>sum_of_other_index_sizes</code>	その他の (プライマリ以外の) インデックスの合計サイズ (ページ数)

表 15.7 `innodb_index_stats` のカラム

カラム名	説明
<code>database_name</code>	データベース名
<code>table_name</code>	テーブル名、パーティション名、またはサブパーティション名
<code>index_name</code>	インデックス名
<code>last_update</code>	行が最後に更新された時刻を示すタイムスタンプ
<code>stat_name</code>	<code>stat_value</code> カラムに値がレポートされている統計の名前
<code>stat_value</code>	<code>stat_name</code> カラムで名前が指定されている統計の値
<code>sample_size</code>	<code>stat_value</code> カラムに示されている推定値のサンプリングされるページの数

カラム名	説明
stat_description	stat_name カラムで名前が指定されている統計の説明

`innodb_table_stats` テーブルおよび `innodb_index_stats` テーブルには、インデックス統計が最後に更新された日時を示す `last_update` カラムが含まれます:

```
mysql> SELECT * FROM innodb_table_stats \G
***** 1. row *****
  database_name: sakila
  table_name: actor
  last_update: 2014-05-28 16:16:44
  n_rows: 200
  clustered_index_size: 1
  sum_of_other_index_sizes: 1
  ...
```

```
mysql> SELECT * FROM innodb_index_stats \G
***** 1. row *****
  database_name: sakila
  table_name: actor
  index_name: PRIMARY
  last_update: 2014-05-28 16:16:44
  stat_name: n_diff_pfx01
  stat_value: 200
  sample_size: 1
  ...
```

`innodb_table_stats` テーブルおよび `innodb_index_stats` テーブルは手動で更新できるため、データベースを変更せずに特定のクエリー最適化計画を強制的に実行したり、代替計画をテストできます。統計を手動で更新する場合は、`FLUSH TABLE tbl_name` ステートメントを使用して更新された統計をロードします。

永続統計はサーバーインスタンスに関連するため、ローカル情報とみなされます。したがって、自動統計再計算が行われた場合、`innodb_table_stats` テーブルおよび `innodb_index_stats` テーブルはレプリケートされません。`ANALYZE TABLE` を実行して統計の同期再計算を開始すると、ステートメントはレプリケートされ (ロギングを抑制していないかぎり)、レプリカで再計算が行われます。

InnoDB 永続的統計テーブルの例

`innodb_table_stats` テーブルには、テーブルごとに 1 つの行が含まれます。次の例は、収集されるデータのタイプを示しています。

テーブル `t1` には、プライマリインデックス (カラム `a`、`b`)、セカンダリインデックス (カラム `c`、`d`)、および一意のインデックス (カラム `e`、`f`) が含まれています。

```
CREATE TABLE t1 (
  a INT, b INT, c INT, d INT, e INT, f INT,
  PRIMARY KEY (a, b), KEY i1 (c, d), UNIQUE KEY i2uniq (e, f)
) ENGINE=INNODB;
```

5 行のサンプルデータを挿入すると、テーブル `t1` は次のように表示されます:

```
mysql> SELECT * FROM t1;
+----+-----+-----+-----+
| a | b | c | d | e | f |
+----+-----+-----+-----+
| 1 | 1 | 10 | 11 | 100 | 101 |
| 1 | 2 | 10 | 11 | 200 | 102 |
| 1 | 3 | 10 | 11 | 100 | 103 |
| 1 | 4 | 10 | 12 | 200 | 104 |
| 1 | 5 | 10 | 12 | 100 | 105 |
+----+-----+-----+-----+
```

統計をただちに更新するには、`ANALYZE TABLE` を実行します (`innodb_stats_auto_recalc` が有効になっている場合、変更されるテーブル行の 10% のしきい値に達したと仮定すると、統計は数秒以内に自動的に更新されます)。

```
mysql> ANALYZE TABLE t1;
+----+-----+-----+-----+
```

```
| Table | Op | Msg_type | Msg_text |
+-----+----+-----+-----+
| test.t1 | analyze | status | OK |
+-----+----+-----+-----+
```

テーブル **t1** のテーブル統計には、InnoDB が最後にテーブル統計を更新した時間 (2014-03-14 14:36:34)、テーブル内の行数 (5)、クラスタ化されたインデックスのサイズ (1 ページ)、およびほかのインデックスの合計サイズ (2 ページ) が示されます。

```
mysql> SELECT * FROM mysql.innodb_table_stats WHERE table_name like 't1'\G
***** 1. row *****
  database_name: test
  table_name: t1
  last_update: 2014-03-14 14:36:34
  n_rows: 5
  clustered_index_size: 1
  sum_of_other_index_sizes: 2
```

innodb_index_stats テーブルには、インデックスごとに複数の行が含まれています。 **innodb_index_stats** テーブル内の各行は、**stat_name** カラムで名前が指定され、**stat_description** カラムで説明されている特定のインデックス統計に関連したデータを示します。例:

```
mysql> SELECT index_name, stat_name, stat_value, stat_description
FROM mysql.innodb_index_stats WHERE table_name like 't1';
+-----+-----+-----+-----+
| index_name | stat_name | stat_value | stat_description |
+-----+-----+-----+-----+
| PRIMARY | n_diff_pfx01 | 1 | a |
| PRIMARY | n_diff_pfx02 | 5 | a,b |
| PRIMARY | n_leaf_pages | 1 | Number of leaf pages in the index |
| PRIMARY | size | 1 | Number of pages in the index |
| i1 | n_diff_pfx01 | 1 | c |
| i1 | n_diff_pfx02 | 2 | c,d |
| i1 | n_diff_pfx03 | 2 | c,d,a |
| i1 | n_diff_pfx04 | 5 | c,d,a,b |
| i1 | n_leaf_pages | 1 | Number of leaf pages in the index |
| i1 | size | 1 | Number of pages in the index |
| i2uniq | n_diff_pfx01 | 2 | e |
| i2uniq | n_diff_pfx02 | 5 | e,f |
| i2uniq | n_leaf_pages | 1 | Number of leaf pages in the index |
| i2uniq | size | 1 | Number of pages in the index |
+-----+-----+-----+-----+
```

stat_name カラムには、次のタイプの統計が表示されます。

- **size**: **stat_name=size** である場合、**stat_value** カラムには、インデックス内のページの総数が表示されます。
- **n_leaf_pages**: **stat_name=n_leaf_pages** である場合、**stat_value** カラムには、インデックス内のリーフページの数が表示されます。
- **n_diff_pfxNN**: **stat_name=n_diff_pfx01** である場合、**stat_value** カラムには、インデックスの最初のカラム内の固有の値の数が表示されます。 **stat_name=n_diff_pfx02** である場合、**stat_value** カラムには、インデックスの最初の2つのカラム内の固有の値の数が表示されます。以下も同様です。 **stat_name=n_diff_pfxNN** の場合、**stat_description** カラムには、カウントされるインデックスカラムのカンマ区切りリストが表示されます。

カーディナリティデータを提供する **n_diff_pfxNN** 統計をさらに詳しく説明するために、前に紹介した **t1** テーブルの例をもう一度検討してください。次に示すように、**t1** テーブルは、プライマリインデックス (カラム **a**、**b**)、セカンダリインデックス (カラム **c**、**d**)、および一意のインデックス (カラム **e**、**f**) で作成されます。

```
CREATE TABLE t1 (
  a INT, b INT, c INT, d INT, e INT, f INT,
  PRIMARY KEY (a, b), KEY i1 (c, d), UNIQUE KEY i2uniq (e, f)
) ENGINE=INNODB;
```

5 行のサンプルデータを挿入すると、テーブル **t1** は次のように表示されます:

```
mysql> SELECT * FROM t1;
+----+----+----+----+----+----+
| a | b | c | d | e | f |
+----+----+----+----+----+----+
```


1	1	10	11	100	101
1	2	10	11	200	102
1	3	10	11	100	103
1	4	10	12	200	104
1	5	10	12	100	105

`stat_name LIKE 'n_diff%'` である `index_name`、`stat_name`、`stat_value`、および `stat_description`、をクエリーすると、次の結果セットが返されます。

```
mysql> SELECT index_name, stat_name, stat_value, stat_description
FROM mysql.innodb_index_stats
WHERE table_name like 't1' AND stat_name LIKE 'n_diff%';
```

index_name	stat_name	stat_value	stat_description
PRIMARY	n_diff_pfx01	1	a
PRIMARY	n_diff_pfx02	5	a,b
i1	n_diff_pfx01	1	c
i1	n_diff_pfx02	2	c,d
i1	n_diff_pfx03	2	c,d,a
i1	n_diff_pfx04	5	c,d,a,b
i2uniq	n_diff_pfx01	2	e
i2uniq	n_diff_pfx02	5	e,f

PRIMARY インデックスの場合は、2 つの `n_diff%` 行があります。行数は、インデックス内のコラム数に等しくなります。

注記

一意でないインデックスの場合、InnoDB は主キーのコラムを追加します。

- `index_name=PRIMARY` および `stat_name=n_diff_pfx01` である場合、`stat_value` は 1 です。これは、インデックスの最初のコラム (コラム a) 内に固有の値が 1 つ存在することを示します。コラム a 内の固有の値の数は、テーブル `t1` 内のコラム a のデータを表示することによって確認されます。ここでは、固有の値が 1 つ存在します (1)。カウントされるコラム (a) は、結果セットの `stat_description` コラムに示されています。
- `index_name=PRIMARY` および `stat_name=n_diff_pfx02` である場合、`stat_value` は 5 です。これは、インデックスの 2 つのコラム (a,b) 内に固有の値が 5 つ存在することを示します。コラム a および b 内の固有の値の数は、テーブル `t1` 内のコラム a および b のデータを表示することによって確認されます。ここでは、固有の値が 5 つ存在します: (1,1)、(1,2)、(1,3)、(1,4)、および (1,5)。カウントされるコラム (a,b) は、結果セットの `stat_description` コラムに示されています。

セカンダリインデックス (i1) の場合は、4 つの `n_diff%` 行があります。セカンダリインデックス (c,d) には 2 つのコラムのみが定義されていますが、InnoDB では一意でないすべてのインデックスに主キーが接尾辞として付加されるため、セカンダリインデックスには 4 つの `n_diff%` 行があります。その結果、セカンダリインデックスコラム (c,d) と主キーコラム (a,b) の両方を反映して、2 つではなく 4 つの `n_diff%` 行があります。

- `index_name=i1` および `stat_name=n_diff_pfx01` である場合、`stat_value` は 1 です。これは、インデックスの最初のコラム (コラム c) 内に固有の値が 1 つ存在することを示します。コラム c 内の固有の値の数は、テーブル `t1` 内のコラム c のデータを表示することによって確認されます。ここでは、固有の値が 1 つ存在します: (10)。カウントされるコラム (c) は、結果セットの `stat_description` コラムに示されています。
- `index_name=i1` および `stat_name=n_diff_pfx02` である場合、`stat_value` は 2 です。これは、インデックスの最初の 2 つのコラム (c,d) 内に固有の値が 2 つ存在することを示します。コラム c および d 内の固有の値の数は、テーブル `t1` 内のコラム c および d のデータを表示することによって確認されます。ここでは、固有の値が 2 つ存在します: (10,11) および (10,12)。カウントされるコラム (c,d) は、結果セットの `stat_description` コラムに示されています。
- `index_name=i1` および `stat_name=n_diff_pfx03` である場合、`stat_value` は 2 です。これは、インデックスの最初の 3 つのコラム (c,d,a) 内に固有の値が 2 つ存在することを示します。コラム c、d、および a 内の固有の値の数は、テーブル `t1` 内のコラム c、d、および a のデータを表示することによって確認されます。ここでは、固有の値が 2 つ存在します: (10,11,1) および (10,12,1)。カウントされるコラム (c,d,a) は、結果セットの `stat_description` コラムに示されています。

- `index_name=i1` および `stat_name=n_diff_pfx04` である場合、`stat_value` は 5 です。これは、インデックスの 4 つのカラム (c,d,a,b) 内に固有の値が 5 つ存在することを示します。カラム c, d, a および b の個別値の数を確認するには、カラム c, d, a のデータおよびテーブル t1 の b を表示します。これらには 5 つの個別値があります: (10,11,1,1)、(10,11,1,2)、(10,11,1,3)、(10,12,1,4) および (10,12,1,5)。カウントされるカラム (c,d,a,b) は、結果セットの `stat_description` カラムに示されています。
- 一意のインデックス (i2uniq) の場合は、2 つの `n_diff%` 行があります。
- `index_name=i2uniq` および `stat_name=n_diff_pfx01` である場合、`stat_value` は 2 です。これは、インデックスの最初のカラム (カラム e) 内に固有の値が 2 つ存在することを示します。カラム e 内の固有の値の数は、テーブル t1 内のカラム e のデータを表示することによって確認されます。ここには、固有の値が 2 つ存在します: (100) および (200)。カウントされるカラム (e) は、結果セットの `stat_description` カラムに示されています。
- `index_name=i2uniq` および `stat_name=n_diff_pfx02` である場合、`stat_value` は 5 です。これは、インデックスの 2 つのカラム (e,f) 内に固有の値が 5 つ存在することを示します。カラム e および f の個別値の数を確認するには、テーブル t1 のカラム e および f のデータを表示します。これらには 5 つの個別値があります: (100,101)、(200,102)、(100,103)、(200,104) および (100,105)。カウントされるカラム (e,f) は、結果セットの `stat_description` カラムに示されています。

innodb_index_stats テーブルを使用したインデックスサイズの取得

`innodb_index_stats` テーブルを使用して、テーブル、パーティションまたはサブパーティションのインデックスサイズを取得できます。次の例では、テーブル t1 のインデックスサイズが取得されています。テーブル t1 の定義および対応するインデックス統計については、[InnoDB 永続的統計テーブルの例](#) を参照してください。

```
mysql> SELECT SUM(stat_value) pages, index_name,
SUM(stat_value)*@@innodb_page_size size
FROM mysql.innodb_index_stats WHERE table_name='t1'
AND stat_name = 'size' GROUP BY index_name;
```

```
+-----+-----+-----+
| pages | index_name | size |
+-----+-----+-----+
| 1 | PRIMARY | 16384 |
| 1 | i1 | 16384 |
| 1 | i2uniq | 16384 |
+-----+-----+-----+
```

パーティションまたはサブパーティションの場合は、変更された `WHERE` 句で同じクエリーを使用してインデックスサイズを取得できます。たとえば、次のクエリーは、テーブル t1 のパーティションのインデックスサイズを取得しません。

```
mysql> SELECT SUM(stat_value) pages, index_name,
SUM(stat_value)*@@innodb_page_size size
FROM mysql.innodb_index_stats WHERE table_name like 't1#P%'
AND stat_name = 'size' GROUP BY index_name;
```

15.8.10.2 非永続的オプティマイザ統計のパラメータの構成

このセクションでは、非永続オプティマイザ統計を構成する方法について説明します。オプティマイザ統計は、`innodb_stats_persistent=OFF` の場合、または個々のテーブルが `STATS_PERSISTENT=0` で作成または変更された場合、ディスクに永続化されません。かわりに、統計はメモリーに格納され、サーバーの停止時に失われます。統計は、特定の操作および特定の条件下で定期的に更新されます。

オプティマイザ統計はデフォルトでディスクに永続化され、`innodb_stats_persistent` 構成オプションによって有効化されます。永続的オプティマイザ統計については、[セクション 15.8.10.1 「永続的オプティマイザ統計のパラメータの構成」](#) を参照してください。

オプティマイザ統計の更新

非永続オプティマイザ統計は、次の場合に更新されます:

- `ANALYZE TABLE` の実行。
- `SHOW TABLE STATUS`、`SHOW INDEX` を実行するか、`innodb_stats_on_metadata` オプションを有効にして `INFORMATION_SCHEMA.TABLES` または `INFORMATION_SCHEMA.STATISTICS` テーブルをクエリーします。

`innodb_stats_on_metadata` のデフォルト設定は `OFF` です。 `innodb_stats_on_metadata` を有効にすると、多数のテーブルまたはインデックスを持つスキーマのアクセス速度が低下し、InnoDB テーブルを含むクエリーの実行計画の安定性が低下する可能性があります。 `innodb_stats_on_metadata` は、`SET` ステートメントを使用してグローバルに構成されます。

```
SET GLOBAL innodb_stats_on_metadata=ON
```

注記

`innodb_stats_on_metadata` は、オプティマイザ `statistics` が非永続として構成されている場合 (`innodb_stats_persistent` が無効な場合) にのみ適用されます。

- デフォルトの `--auto-rehash` オプションを有効にして `mysql` クライアントを起動します。 `auto-rehash` オプションを使用すると、すべての InnoDB テーブルがオープンされ、オープンしているテーブルの操作によって統計が再計算されます。

`mysql` クライアントの起動時間を改善し、統計を更新するには、`--disable-auto-rehash` オプションを使用して `auto-rehash` をオフにします。 `auto-rehash` 機能は、対話ユーザーのためのデータベース、テーブル、およびカラム名の自動名前補完を有効にします。

- 最初にテーブルが開かれます。
- InnoDB は、統計が最後に更新されてから 1 / 16 のテーブルが変更されたことを検出します。

サンプルページ数の構成

MySQL クエリーオプティマイザは、インデックスの相対的な**選択性**に基づいて、キー分布に関する推定された**統計**を使用して実行計画のためのインデックスを選択します。 InnoDB でオプティマイザ統計が更新されると、テーブルの各インデックスからランダムページがサンプリングされ、インデックスの `cardinality` が見積もられます。(この手法は、**ランダムダイブ**と呼ばれます。)

統計の推定値の品質を制御する (それにより、クエリーオプティマイザへの情報を改善する) ために、パラメータ `innodb_stats_transient_sample_pages` を使用して、サンプリングされるページの数を変更できます。 サンプリングされるページのデフォルト数は 8 です。これは、正確な推定値を生成するには十分ではなく、クエリーオプティマイザによる不適切なインデックス選択につながる可能性があります。 この手法は、大きなテーブルや、**結合**で使用されるテーブルの場合に特に重要です。 このようなテーブルに対する不必要な**フルテーブルスキャン**が、パフォーマンスの重大な問題になる場合があります。 このようなクエリーのチューニングのヒントは、**セクション 8.2.1.23 「全テーブルスキャンの回避」**を参照してください。 `innodb_stats_transient_sample_pages` は、実行時に設定できるグローバルパラメータです。

`innodb_stats_persistent=0` である場合は、`innodb_stats_transient_sample_pages` の値がすべての InnoDB テーブルおよびインデックスのインデックスサンプリングに影響を与えます。 インデックスサンプルサイズを変更する場合、次のような大きな影響がある可能性があることに注意してください:

- 1 や 2 などの小さな値では、カーディナリティーの不正確な推定値が生成される可能性があります。
- `innodb_stats_transient_sample_pages` 値を大きくすると、必要なディスク読み取りが増える可能性があります。 8 よりもはるかに大きい値 (100 など) を指定すると、テーブルのオープンまたは `SHOW TABLE STATUS` の実行にかかる時間が大幅に遅くなる可能性があります。
- オプティマイザが、インデックスの選択性の異なる推定値に基づいて、非常に異なるクエリー計画を選択する可能性があります。

あるシステムで `innodb_stats_transient_sample_pages` のどのような値が最適に機能したとしても、このオプションを設定し、その値のままにします。 過剰な I/O を必要とせず、データベース内のすべてのテーブルに対して適度に正確な推定値を生成する値を選択してください。 統計は `ANALYZE TABLE` の実行時以外のさまざまな時間に自動的に再計算されるため、インデックスのサンプルサイズを増やし、`ANALYZE TABLE` を実行してから、サンプルサイズをふたたび減らしても意味がありません。

通常、小さいテーブルでは、大きいテーブルよりも必要なインデックスサンプルが少なくなります。 データベースに多数の大きなテーブルが含まれている場合は、ほとんどが小さなテーブルである場合より大きな `innodb_stats_transient_sample_pages` 値を使用することを考慮してください。

15.8.10.3 InnoDB テーブルに対する ANALYZE TABLE の複雑さの推定

InnoDB テーブルに対する ANALYZE TABLE の複雑さは、次のものに依存します。

- `innodb_stats_persistent_sample_pages` で定義される、サンプリングされるページの数。
- テーブル内のインデックス付きカラムの数
- パーティションの数。テーブルにパーティションが存在しない場合、パーティションの数は 1 であるとみなされま

これらのパラメータを使用すると、ANALYZE TABLE の複雑さを推定するための概略の計算式は次のようになります。

`innodb_stats_persistent_sample_pages` の値 * テーブル内のインデックス付きカラムの数 * パーティションの数

通常は、この結果の値が大きいくほど、ANALYZE TABLE の実行時間も大きくなります。

注記

`innodb_stats_persistent_sample_pages` は、グローバルレベルでサンプリングされるページの数を定義します。個々のテーブルのサンプリングされるページの設定するには、CREATE TABLE または ALTER TABLE で `STATS_SAMPLE_PAGES` オプションを使用します。詳細は、[セクション15.8.10.1「永続的オプティマイザ統計のパラメータの構成」](#)を参照してください。

`innodb_stats_persistent=OFF` である場合、サンプリングされるページの数 `innodb_stats_transient_sample_pages` で定義されます。詳細は、[セクション15.8.10.2「非永続的オプティマイザ統計のパラメータの構成」](#)を参照してください。

ANALYZE TABLE の複雑さを推定するためのより詳細なアプローチを示すために、次の例を考えてみます。

ビッグオー表記では、ANALYZE TABLE の複雑さは次のように記述されます。

```
O(n_sample
  * (n_cols_in_uniq_i
    + n_cols_in_non_uniq_i
    + n_cols_in_pk * (1 + n_non_uniq_i))
  * n_part)
```

ここでは:

- `n_sample` は、サンプリングされるページの数 (`innodb_stats_persistent_sample_pages` で定義されます)
- `n_cols_in_uniq_i` は、すべての一意のインデックス内のすべてのカラムの総数 (主キーカラムはカウントしない)
- `n_cols_in_non_uniq_i` は、すべての非一意インデックスのすべてのカラムの合計数です
- `n_cols_in_pk` は、主キー内のカラム数 (主キーが定義されていない場合、InnoDB は単一カラムの主キーを内部的に作成します)
- `n_non_uniq_i` は、テーブル内の一意でないインデックスの数です
- `n_part` は、パーティションの数。パーティションが定義されていない場合、そのテーブルは単一パーティションであるとみなされます。

ここで、主キー (2 つのカラム)、一意インデックス (2 つのカラム) および 2 つの非一意インデックス (それぞれ 2 つのカラム) を持つ次のテーブル (テーブル t) について考えてみます:

```
CREATE TABLE t (
  a INT,
  b INT,
  c INT,
  d INT,
  e INT,
  f INT,
```

```
g INT,  
h INT,  
PRIMARY KEY (a, b),  
UNIQUE KEY i1uniq (c, d),  
KEY i2nonuniq (e, f),  
KEY i3nonuniq (g, h)  
);
```

上で説明したアルゴリズムに必要なカラムとインデックスデータについて、テーブル `t` の `mysql.innodb_index_stats` 永続的インデックス統計テーブルにクエリーします。 `n_diff_pfx%` の統計には、各インデックスに対してカウントされるカラムが示されます。たとえば、カラム `a` および `b` は、主キーのインデックスに対してカウントされます。一意でないインデックスの場合、主キーカラム (a,b) は、ユーザー定義カラムに加えてカウントされます。

注記

InnoDB 永続的統計テーブルの詳細は、[セクション15.8.10.1「永続的オプティマイザ統計のパラメータの構成」](#)を参照してください。

```
mysql> SELECT index_name, stat_name, stat_description  
FROM mysql.innodb_index_stats WHERE  
database_name='test' AND  
table_name='t' AND  
stat_name like 'n_diff_pfx%';
```

index_name	stat_name	stat_description
PRIMARY	n_diff_pfx01	a
PRIMARY	n_diff_pfx02	a,b
i1uniq	n_diff_pfx01	c
i1uniq	n_diff_pfx02	c,d
i2nonuniq	n_diff_pfx01	e
i2nonuniq	n_diff_pfx02	e,f
i2nonuniq	n_diff_pfx03	e,f,a
i2nonuniq	n_diff_pfx04	e,f,a,b
i3nonuniq	n_diff_pfx01	g
i3nonuniq	n_diff_pfx02	g,h
i3nonuniq	n_diff_pfx03	g,h,a
i3nonuniq	n_diff_pfx04	g,h,a,b

上に示したインデックス統計データとテーブル定義に基づいて、次の値を確認できます。

- `n_cols_in_uniq_i` (すべての一意のインデックス内のすべてのカラムの総数、主キーカラムはカウントしない) は 2 (c および d)
- `n_cols_in_non_uniq_i`、一意でないすべてのインデックスのすべてのカラムの合計数は 4 (e, f, g および h) です
- `n_cols_in_pk` (主キー内のカラム数) は 2 (a および b)
- テーブル内の一意でないインデックスの数である `n_non_uniq_i` は 2 (i2nonuniq および i3nonuniq) です
- `n_part` (パーティションの数) は 1。

これで、スキャンされるリーフページの数を決めるために `innodb_stats_persistent_sample_pages * (2 + 4 + 2 * (1 + 2)) * 1` を計算できます。 `innodb_stats_persistent_sample_pages` が 20 のデフォルト値に設定されており、かつページサイズがデフォルトの 16 KiB (`innodb_page_size=16384`) である場合は、テーブル `t` に対して `20 * 12 * 16384` バイト、つまり約 4 MiB が読み取られると推定できます。

注記

一部のリーフページはすでにバッファプール内にキャッシュされている可能性があるため、4 MiB のすべてがディスクから読み取られるとは限りません。

15.8.11 インデックスページのマージしきい値の構成

インデックスページの `MERGE_THRESHOLD` 値を構成できます。行が削除されたとき、または `UPDATE` 操作によって行が短縮されたときに、インデックスページの「page-full」割合が `MERGE_THRESHOLD` 値を下回っ

た場合、InnoDB はインデックスページを隣接するインデックスページとマージしようとします。デフォルトの `MERGE_THRESHOLD` 値は 50 で、これは以前にハードコードされた値です。 `MERGE_THRESHOLD` の最小値は 1 で、最大値は 50 です。

インデックスページの「page-full」割合がデフォルトの `MERGE_THRESHOLD` 設定である 50% を下回ると、InnoDB はインデックスページを隣接するページとマージしようとします。両方のページが 50% に近い場合、ページがマージされた直後にページ分割が発生する可能性があります。このマージ分割動作が頻繁に発生する場合は、パフォーマンスに悪影響を与える可能性があります。頻繁なマージスプリットを回避するには、InnoDB が「page-full」の低い割合でページマージを試行するように、 `MERGE_THRESHOLD` 値を小さくします。ページフルの割合が低いページをマージすると、インデックスページの空き領域が増え、マージ分割の動作を減らすことができます。

インデックスページ用の `MERGE_THRESHOLD` は、テーブルまたは個々のインデックスに対して定義できます。個々のインデックスに定義された `MERGE_THRESHOLD` 値は、テーブルに定義された `MERGE_THRESHOLD` 値よりも優先されます。未定義の場合、 `MERGE_THRESHOLD` 値はデフォルトで 50 に設定されます。

テーブルに対する `MERGE_THRESHOLD` の設定

`CREATE TABLE` ステートメントの `table_option COMMENT` 句を使用して、テーブルの `MERGE_THRESHOLD` 値を設定できます。例:

```
CREATE TABLE t1 (  
  id INT,  
  KEY id_index (id)  
) COMMENT='MERGE_THRESHOLD=45';
```

`ALTER TABLE` で `table_option COMMENT` 句を使用して、既存のテーブルの `MERGE_THRESHOLD` 値を設定することもできます:

```
CREATE TABLE t1 (  
  id INT,  
  KEY id_index (id)  
);  
  
ALTER TABLE t1 COMMENT='MERGE_THRESHOLD=40';
```

個々のインデックスに対する `MERGE_THRESHOLD` の設定

個々のインデックスの `MERGE_THRESHOLD` 値を設定するには、次の例に示すように、 `CREATE TABLE`、 `ALTER TABLE` または `CREATE INDEX` で `index_option COMMENT` 句を使用できます:

- `CREATE TABLE` を使用した個々のインデックスに対する `MERGE_THRESHOLD` の設定:

```
CREATE TABLE t1 (  
  id INT,  
  KEY id_index (id) COMMENT 'MERGE_THRESHOLD=40'  
);
```

- `ALTER TABLE` を使用した個々のインデックスに対する `MERGE_THRESHOLD` の設定:

```
CREATE TABLE t1 (  
  id INT,  
  KEY id_index (id)  
);  
  
ALTER TABLE t1 DROP KEY id_index;  
ALTER TABLE t1 ADD KEY id_index (id) COMMENT 'MERGE_THRESHOLD=40';
```

- `CREATE INDEX` を使用した個々のインデックスに対する `MERGE_THRESHOLD` の設定:

```
CREATE TABLE t1 (id INT);  
CREATE INDEX id_index ON t1 (id) COMMENT 'MERGE_THRESHOLD=40';
```

注記

`GEN_CLUST_INDEX` のインデックスレベルで `MERGE_THRESHOLD` 値を変更することはできません。これは、InnoDB テーブルが主キーまたは一意キーインデックスなしで作成さ

れたときに InnoDB によって作成されるクラスタインデックスです。GEN_CLUST_INDEX の MERGE_THRESHOLD 値は、テーブルに MERGE_THRESHOLD を設定することによってのみ変更できます。

インデックスの MERGE_THRESHOLD 値のクエリー

インデックスの現在の MERGE_THRESHOLD 値は、INNODB_INDEXES テーブルをクエリーすることで取得できます。例:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_INDEXES WHERE NAME='id_index' \G
***** 1. row *****
INDEX_ID: 91
NAME: id_index
TABLE_ID: 68
TYPE: 0
N_FIELDS: 1
PAGE_NO: 4
SPACE: 57
MERGE_THRESHOLD: 40
```

table_option COMMENT 句を使用して明示的に定義されている場合は、SHOW CREATE TABLE を使用してテーブルの MERGE_THRESHOLD 値を表示できます:

```
mysql> SHOW CREATE TABLE t2 \G
***** 1. row *****
Table: t2
Create Table: CREATE TABLE `t2` (
  `id` int(11) DEFAULT NULL,
  KEY `id_index` (`id`) COMMENT 'MERGE_THRESHOLD=40'
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```

注記

インデックスレベルで定義された MERGE_THRESHOLD 値は、テーブルに定義された MERGE_THRESHOLD 値よりも優先されます。未定義の場合、MERGE_THRESHOLD はデフォルトで 50% (MERGE_THRESHOLD=50、以前にハードコードされた値) に設定されます。

同様に、index_option COMMENT 句を使用して明示的に定義されている場合は、SHOW INDEX を使用してインデックスの MERGE_THRESHOLD 値を表示できます:

```
mysql> SHOW INDEX FROM t2 \G
***** 1. row *****
Table: t2
Non_unique: 1
Key_name: id_index
Seq_in_index: 1
Column_name: id
Collation: A
Cardinality: 0
Sub_part: NULL
Packed: NULL
Null: YES
Index_type: BTREE
Comment:
Index_comment: MERGE_THRESHOLD=40
```

MERGE_THRESHOLD 設定の影響の測定

INNODB_METRICS テーブルには、インデックスページのマージに対する MERGE_THRESHOLD 設定の影響を測定するために使用できる 2 つのカウンタが用意されています。

```
mysql> SELECT NAME, COMMENT FROM INFORMATION_SCHEMA.INNODB_METRICS
WHERE NAME like '%index_page_merge%';
+-----+-----+
| NAME                | COMMENT                                     |
+-----+-----+
| index_page_merge_attempts | Number of index page merge attempts |
| index_page_merge_successful | Number of successful index page merges |
```


MERGE_THRESHOLD 値を下げる場合の目標は次のとおりです:

- ページマージの試行回数が少なく、ページマージが成功しました
- 同様の数のページマージ試行と成功したページマージ

MERGE_THRESHOLD 設定が小さすぎると、空のページ領域が過剰になるため、データファイルが大きくなる可能性があります。

INNODB_METRICS カウンタの使用の詳細は、[セクション15.15.6「InnoDB INFORMATION_SCHEMA メトリックテーブル」](#)を参照してください。

15.8.12 専用 MySQL Server の自動構成の有効化

innodb_dedicated_server が有効な場合、InnoDB は次の変数を自動的に構成します:

- [innodb_buffer_pool_size](#)
- [innodb_log_file_size](#)
- [innodb_log_files_in_group](#) (MySQL 8.0.14 の時点)
- [innodb_flush_method](#)

MySQL インスタンスが、使用可能なすべてのシステムリソースを使用できる専用サーバーに存在する場合にのみ、[innodb_dedicated_server](#) を有効にすることを検討してください。たとえば、MySQL のみを実行する Docker コンテナまたは専用 VM で MySQL Server を実行する場合は、有効化を検討してください。MySQL インスタンスが他のアプリケーションとシステムリソースを共有している場合、[innodb_dedicated_server](#) を有効にすることはお薦めしません。

次の情報では、各変数が自動的に構成される方法について説明します。

- [innodb_buffer_pool_size](#)

バッファプールサイズは、サーバーで検出されたメモリー量に従って構成されます。

表 15.8 自動的に構成されるバッファプールサイズ

検出されたサーバーメモリー	バッファプールサイズ
1GB 未満	128MiB (デフォルト値)
1GB から 4GB	detected server memory * 0.5
4GB を超える	detected server memory * 0.75

- [innodb_log_file_size](#)

MySQL 8.0.14 では、ログファイルサイズは自動的に構成されたバッファプールサイズに従って構成されます。

表 15.9 自動的に構成されるログファイルサイズ

バッファプールサイズ	ログファイルのサイズ
8GB 未満	512MiB
8GB から 128GB	1024MiB
128GB を超える	2048MiB

注記

MySQL 8.0.14 より前は、次に示すように、[innodb_log_file_size](#) 変数はサーバーで検出されたメモリー量に従って自動的に構成されていました:

表 15.10 自動的に構成されるログファイルサイズ (MySQL 8.0.13 以前)

検出されたサーバーメモリー	ログファイルのサイズ
< 1GB	48MiB (デフォルト値)
<= 4GB	128MiB
<= 8GB	512MiB
<= 16GB	1024MiB
> 16GB	2048MiB

- [innodb_log_files_in_group](#)

ログファイルの数は、自動的に構成されたバッファプールサイズ (GB) に従って構成されます。
[innodb_log_files_in_group](#) 変数の自動構成が MySQL 8.0.14 に追加されました。

表 15.11 自動的に構成されるログファイルの数

バッファプールサイズ	ログファイル数
8GB 未満	ROUND(buffer pool size)
8GB から 128GB	ROUND(buffer pool size * 0.75)
128GB を超える	64

注記

丸められたバッファプールサイズの値が 2G バイト未満の場合、[innodb_log_files_in_group](#) の最小値 2 が適用されます。

- [innodb_flush_method](#)

[innodb_dedicated_server](#) が有効になっている場合、flush メソッドは [O_DIRECT_NO_FSYNC](#) に設定されます。
[O_DIRECT_NO_FSYNC](#) 設定を使用できない場合は、デフォルトの [innodb_flush_method](#) 設定が使用されます。

InnoDB は、I/O のフラッシュ中に [O_DIRECT](#) を使用しますが、書き込み操作のたびに [fsync\(\)](#) システムコールをスキップします。

警告

MySQL 8.0.14 より前では、この設定は XFS や EXT4 などのファイルシステムには適していません。これらのファイルシステムでは、[fsync\(\)](#) システムコールを使用してファイルシステムメタデータの変更を同期する必要があります。

MySQL 8.0.14 の時点では、[fsync\(\)](#) は、新しいファイルの作成後、ファイルサイズの増加後およびファイルのクローズ後にコールされ、ファイルシステムメタデータの変更が確実に同期されます。各書き込み操作の後、[fsync\(\)](#) システムコールはスキップされます。

redo ログファイルとデータファイルが異なるストレージデバイスに存在し、データファイルの書き込みがバッテリバックされていないデバイスキャッシュからフラッシュされる前に予期しない終了が発生した場合、データが失われる可能性があります。redo ログファイルおよびデータファイルに別の記憶域デバイスを使用する場合、およびデータファイルがバッテリバックアップされていないキャッシュを持つデバイスに存在する場合は、かわりに [O_DIRECT](#) を使用します。

自動的に構成されたオプションがオプションファイルまたは他の場所で明示的に構成されている場合は、明示的に指定された設定が使用され、次のような起動警告が [stderr](#) に出力されます:

[警告] [000000] InnoDB: [innodb_buffer_pool_size=134217728](#) が明示的に指定されているため、[innodb_buffer_pool_size](#) ではオプション [innodb_dedicated_server](#) は無視されます。

あるオプションを明示的に構成しても、他のオプションの自動構成は妨げられません。

`innodb_dedicated_server` が有効で、`innodb_buffer_pool_size` がオプションファイルで明示的に構成されている場合でも、`innodb_log_file_size` および `innodb_log_files_in_group` は、バッファプールのサイズの構成に使用されていなくても、サーバーで検出されたメモリー量に基づいて計算されたバッファプールサイズ値に基づいて自動的に構成されます。

自動的に構成された設定は、MySQL サーバーが起動されるたびに必要に応じて評価および再構成されます。

15.9 InnoDB のテーブルおよびページの圧縮

このセクションでは、InnoDB テーブルの圧縮および InnoDB ページの圧縮機能について説明します。ページ圧縮機能は、[transparent page compression](#) と呼ばれます。

InnoDB の圧縮機能を使用すると、データが圧縮形式で格納されるテーブルを作成できます。圧縮を使用すると、生のパフォーマンスと拡張性の両方を改善する際に役立つことがあります。圧縮とは、ディスクとメモリー間で転送されるデータの量が少なくなり、ディスク上とメモリー内で占有される領域の量が少なくなることを意味します。インデックスデータも圧縮されるため、[セカンダリインデックス](#)を含むテーブルでは利点も増幅されます。SSD ストレージデバイスは、HDD デバイスよりも容量が小さくなる傾向があるため、圧縮が特に重要となる可能性があります。

15.9.1 InnoDB テーブルの圧縮

このセクションでは、`file_per_table` テーブルスペースまたは `general tablespaces` に存在する InnoDB テーブルでサポートされる InnoDB テーブルの圧縮について説明します。テーブル圧縮は、`CREATE TABLE` または `ALTER TABLE` で `ROW_FORMAT=COMPRESSED` 属性を使用して有効にします。

15.9.1.1 テーブル圧縮の概要

プロセッサおよびキャッシュメモリーは、ディスクストレージデバイスよりも速度が上昇しているため、多くのワークロードが[ディスクバウンド](#)になります。データ圧縮を使用すると、データベースのサイズが小さくなり、I/O が削減され、スループットが改善されますが、CPU 使用率が上昇するという少しの犠牲が伴います。圧縮は、頻繁に使用されるデータをメモリー内に保持するために十分な RAM が搭載されたシステム上で、読み取り負荷の高いアプリケーションを実行する際に、特に有効です。

`ROW_FORMAT=COMPRESSED` で作成された InnoDB テーブルでは、構成された `innodb_page_size` 値より小さいディスク上の `page size` を使用できます。ページが小さいほど、ディスクから読み取られる I/O とディスクに書き込まれる I/O が少なくなるため、SSD デバイスを使用する際に、特に有効です。

圧縮されたページサイズは、`CREATE TABLE` または `ALTER TABLE KEY_BLOCK_SIZE` パラメータを使用して指定します。システムテーブルスペースに圧縮テーブルを格納できないため、異なるページサイズでは、テーブルを `system tablespace` ではなく `file-per-table` テーブルスペースまたは `general tablespace` に配置する必要があります。詳細は、[セクション 15.6.3.2 「File-Per-Table テーブルスペース」](#) および [セクション 15.6.3.3 「一般テーブルスペース」](#) を参照してください。

圧縮レベルは、`KEY_BLOCK_SIZE` の値に関係なく同じです。`KEY_BLOCK_SIZE` に小さい値を指定するほど、徐々にページが小さくなるという I/O の利点が得られます。ただし、小さすぎる値を指定すると、各ページ内に複数の行を収容できるほど十分にデータ値を圧縮できない場合に、ページを再編成するための追加のオーバーヘッドが発生します。そのインデックスごとのキーカラムの長さに基づいて、どのくらい小さい `KEY_BLOCK_SIZE` をテーブルに指定できるのかについて、ハード制限が課されています。小さすぎる値を指定すると、`CREATE TABLE` または `ALTER TABLE` ステートメントが失敗します。

バッファプールには、圧縮済みデータが `KEY_BLOCK_SIZE` の値に基づいたページサイズの小さなページで保持されます。MySQL では、カラム値を抽出または更新するために、圧縮されていないデータを含む未圧縮のページもバッファプールに作成されます。バッファプール内では、非圧縮ページへの更新が同等の圧縮済みページに再度書き込まれます。圧縮済みページと非圧縮ページの両方の追加データが収容されるように、バッファページのサイズを変更する必要がある場合もあります。ただし、非圧縮のページは、領域が必要になるとバッファプールから解放され、次のアクセス時に再度圧縮が解除されます。

15.9.1.2 圧縮テーブルの作成

圧縮テーブルは、`file-per-table` テーブルスペースまたは `general tablespaces` で作成できます。テーブル圧縮は、InnoDB `system tablespace` では使用できません。システムテーブルスペース (領域 0、`.ibdata files`) には、ユー

ザーが作成したテーブルを含めることができますが、圧縮されない内部システムデータも含まれます。したがって、圧縮は file-per-table または general テーブルスペースに格納されているテーブル (およびインデックス) にのみ適用されます。

File-Per-Table テーブルスペースでの圧縮テーブルの作成

file-per-table テーブルスペースに圧縮テーブルを作成するには、`innodb_file_per_table` を有効にする必要があります (デフォルト)。このパラメータは、MySQL 構成ファイル (`my.cnf` または `my.ini`) で設定するか、`SET` ステートメントを使用して動的に設定できます。

`innodb_file_per_table` オプションの構成後、`CREATE TABLE` ステートメントまたは `ALTER TABLE` ステートメントで `ROW_FORMAT=COMPRESSED` 句または `KEY_BLOCK_SIZE` 句 (あるいはその両方) を指定して、file-per-table テーブルスペースに圧縮テーブルを作成します。

たとえば、次のステートメントを使用できます:

```
SET GLOBAL innodb_file_per_table=1;
CREATE TABLE t1
(c1 INT PRIMARY KEY)
ROW_FORMAT=COMPRESSED
KEY_BLOCK_SIZE=8;
```

一般テーブルスペースでの圧縮テーブルの作成

一般的なテーブルスペースに圧縮テーブルを作成するには、テーブルスペースの作成時に指定される一般的なテーブルスペースに対して `FILE_BLOCK_SIZE` を定義する必要があります。 `FILE_BLOCK_SIZE` 値は、`innodb_page_size` 値に関連する有効な圧縮ページサイズである必要があります。 `CREATE TABLE` または `ALTER TABLE KEY_BLOCK_SIZE` 句で定義された圧縮テーブルのページサイズは `FILE_BLOCK_SIZE/1024` と同じである必要があります。たとえば、`innodb_page_size=16384` および `FILE_BLOCK_SIZE=8192` の場合、テーブルの `KEY_BLOCK_SIZE` は 8 である必要があります。詳細は、[セクション15.6.3.3「一般テーブルスペース」](#)を参照してください。

次の例は、一般的なテーブルスペースの作成および圧縮テーブルの追加を示しています。この例では、デフォルトの `innodb_page_size` が 16K であると想定しています。8192 の `FILE_BLOCK_SIZE` では、圧縮テーブルの `KEY_BLOCK_SIZE` が 8 である必要があります。

```
mysql> CREATE TABLESPACE `ts2` ADD DATAFILE `ts2.ibd` FILE_BLOCK_SIZE = 8192 Engine=InnoDB;
mysql> CREATE TABLE t4 (c1 INT PRIMARY KEY) TABLESPACE ts2 ROW_FORMAT=COMPRESSED KEY_BLOCK_SIZE=8;
```

メモ

- MySQL 8.0 では、圧縮テーブルのテーブルスペースファイルは InnoDB ページサイズではなく物理ページサイズを使用して作成されるため、空の圧縮テーブルのテーブルスペースファイルの初期サイズは以前の MySQL リリースより小さくなります。
- `ROW_FORMAT=COMPRESSED` を指定する場合は、`KEY_BLOCK_SIZE` を省略できます。`KEY_BLOCK_SIZE` 設定のデフォルトは `innodb_page_size` 値の半分です。
- 有効な `KEY_BLOCK_SIZE` 値を指定する場合は、`ROW_FORMAT=COMPRESSED` を省略できます。圧縮は自動的に有効になります。
- `KEY_BLOCK_SIZE` の最適な値を決定するには、通常、この句に異なる値を指定して同じテーブルの複数のコピーを作成し、生成される `.ibd` ファイルのサイズを測定して、各ファイルが現実的な `workload` でどのように動作するかを確認します。一般的なテーブルスペースの場合、テーブルを削除しても、一般的なテーブルスペースの `.ibd` ファイルのサイズが小さくなることはなく、ディスク領域がオペレーティングシステムに戻されることもないことに注意してください。詳細は、[セクション15.6.3.3「一般テーブルスペース」](#)を参照してください。
- `KEY_BLOCK_SIZE` 値は、ヒントとして処理されます。InnoDB では、必要に応じて異なるサイズが使用される可能性があります。file-per-table テーブルスペースの場合、`KEY_BLOCK_SIZE` は `innodb_page_size` 値以下にのみできます。`innodb_page_size` 値を超える値を指定した場合は、指定された値が無視され、警告が発行されます。また、`KEY_BLOCK_SIZE` は `innodb_page_size` 値の半分に設定されます。`innodb_strict_mode=ON` の場合、無効な `KEY_BLOCK_SIZE` 値を指定するとエラーが返されます。一般的なテーブルスペースの場合、有効な `KEY_BLOCK_SIZE` 値はテーブルスペースの `FILE_BLOCK_SIZE` 設定によって異なります。詳細は、[セクション15.6.3.3「一般テーブルスペース」](#)を参照してください。

- InnoDB は 32KB および 64KB のページサイズをサポートしていますが、これらのページサイズは圧縮をサポートしていません。詳細は、`innodb_page_size` のドキュメントを参照してください。
- InnoDB データページのデフォルトの非圧縮サイズは、16K バイトです。オプション値の組合せに応じて、MySQL では、テーブルスペースデータファイル (.ibd ファイル) に 1KB、2KB、4KB、8KB または 16KB のページサイズが使用されます。実際の圧縮アルゴリズムは、`KEY_BLOCK_SIZE` 値の影響を受けません。この値によって、各圧縮済みチャンクの大きさが決定されるため、各圧縮済みページに詰め込むことができる行数が影響を受けます。
- file-per-table テーブルスペースに圧縮テーブルを作成する場合、`KEY_BLOCK_SIZE` を InnoDB page size と同等に設定しても、通常は圧縮があまり発生しません。たとえば、InnoDB のページサイズは 16K バイトであるため、一般に `KEY_BLOCK_SIZE=16` を設定しても、大量の圧縮は発生しません。多くの場合、このような値で適切に圧縮されるため、この設定は多くの長い BLOB、VARCHAR、または TEXT カラムを持つテーブルで引き続き役立つことがあります。したがって、[セクション15.9.1.5「InnoDB テーブルでの圧縮の動作」](#)で説明したように、必要となる **オーバーフローページ** が少なくなる可能性もあります。一般的なテーブルスペースの場合、InnoDB ページサイズと等しい `KEY_BLOCK_SIZE` 値は許可されません。詳細は、[セクション15.6.3.3「一般テーブルスペース」](#)を参照してください。
- テーブルのすべてのインデックス ([クラスタ化されたインデックス](#)を含む) は、`CREATE TABLE` または `ALTER TABLE` ステートメントで指定されたものと同じページサイズを使用して圧縮されます。`ROW_FORMAT` や `KEY_BLOCK_SIZE` などのテーブル属性は、InnoDB テーブルの `CREATE INDEX` 構文の一部ではなく、指定されている場合は無視されます (ただし、指定されている場合は `SHOW CREATE TABLE` ステートメントの出力に表示されます)。
- パフォーマンス関連の構成オプションについては、[セクション15.9.1.3「InnoDB テーブルの圧縮の調整」](#)を参照してください。

圧縮テーブル上の制約

- 圧縮テーブルは、InnoDB システムテーブルスペースに格納できません。
- 一般テーブルスペースには複数のテーブルを含めることができますが、圧縮テーブルと非圧縮テーブルを同じ一般テーブルスペース内に共存させることはできません。
- 句の名前が `ROW_FORMAT` であるにもかかわらず、圧縮は個別の行にではなく、テーブル全体およびそれに関連付けられたすべてのインデックスに適用されます。
- InnoDB では、圧縮一時テーブルはサポートされていません。`innodb_strict_mode` が有効な場合 (デフォルト)、`ROW_FORMAT=COMPRESSED` または `KEY_BLOCK_SIZE` が指定されていると、`CREATE TEMPORARY TABLE` はエラーを返します。`innodb_strict_mode` が無効な場合は、警告が発行され、圧縮されていない行形式を使用して一時テーブルが作成されます。一時テーブルに対する `ALTER TABLE` 操作にも同じ制限が適用されます。

15.9.1.3 InnoDB テーブルの圧縮の調整

ほとんどの場合、[InnoDB データストレージと圧縮](#)で説明した内部的な最適化によって、圧縮済みデータを使用してもシステムは適切に動作します。ただし、圧縮の効率性はデータの特性によって異なるため、圧縮テーブルのパフォーマンスに影響を与える決定を行うことができます。

- 圧縮するテーブル。
- 使用する圧縮済みページサイズ。
- 実行時のパフォーマンス特性 (システムでデータの圧縮および圧縮解除に要する時間など) に基づいて、バッファプールのサイズを調整するかどうか。ワークロードが [データウェアハウス](#) (主にクエリー) または [OLTP](#) システム (クエリーと DML の混在) に似ているかどうか。
- システムの圧縮テーブル上で DML 操作が実行されているときに、データを配布する方法によって実行時に負荷の高い **圧縮が失敗** する場合は、追加の高度な構成オプションを調整することがあります。

このセクションのガイドラインを使用すると、このようなアーキテクチャー上および構成上の選択を行う際に役立ちます。長期間のテストを実施し、圧縮テーブルを本番環境に移行する準備ができたなら、これらの選択を現実の状況で行なった場合の効率性を検証する方法について、[セクション15.9.1.4「実行時の InnoDB テーブル圧縮の監視」](#)を参照してください。

圧縮を使用するタイミング

一般に、圧縮は、適当な数の文字列カラムが含まれ、データの書き込みよりも読み取りの頻度の方がはるかに高いテーブルで最適に動作します。特定の状況で圧縮の利点が得られるかどうかを予測するための保証された方法はないため、必ず、代表的な構成で実行する特定のワークロードおよびデータセットをテストしてください。圧縮するテーブルを決定する際は、次の要素を検討してください。

データの特性と圧縮

データファイルのサイズを削減する際に圧縮の効率性の決定要因となるものは、データ自体の特性です。圧縮は、データのブロックで繰り返されるバイト文字列を識別することで動作していることを思い出してください。完全にランダム化されたデータは、最悪のケースです。多くの場合、一般的なデータには繰り返し値が含まれているため、効率的に圧縮されます。CHAR、VARCHAR、TEXT、または BLOB のいずれのカラムに定義されているのに関係なく、多くの場合、文字列は効率的に圧縮されます。その一方で、一般に、ほとんどがバイナリデータ (整数または浮動小数) や以前に圧縮されたデータ (JPEG または PNG イメージなど) を含むテーブルは、大幅にまたはまったく効率的に圧縮されない可能性があります。

InnoDB テーブルごとに圧縮を有効にするかどうかを選択します。テーブルおよびそのすべてのインデックスでは、同じ (圧縮済み) ページサイズが使用されます。すべてのテーブルカラムのデータを含む主キー (クラスタ化) インデックスは、セカンダリインデックスよりも効率的に圧縮される可能性があります。長い行が存在する場合に圧縮を使用すると、DYNAMIC 行フォーマットで説明したように、長いカラム値が「オフページ」に格納される可能性があります。このようなオーバーフローページは、効率的に圧縮される可能性があります。これらの検討事項を考慮すると、多くのアプリケーションでは、一部のテーブルがその他よりも効率的に圧縮され、圧縮されたテーブルのサブセットを含むワークロードのみが最適に動作する場合もあります。

特定のテーブルを圧縮するかどうかを決定するには、実験を行います。非圧縮テーブルの .ibd ファイルのコピー上に、LZ77 圧縮 (gzip や WinZip など) が実装されたユーティリティを使用すると、データを圧縮する際の効率性の概算見積もりを取得できます。MySQL ではページサイズ (デフォルトは 16K バイト) に基づいたチャンク単位でデータが圧縮されるため、MySQL で圧縮されたテーブルからは、ファイルベースの圧縮ツールよりも低い圧縮率が得られると予測できます。ページ形式には、ユーザーデータに加えて、圧縮されていない内部システムデータもいくつか含まれます。ファイルベースの圧縮ユーティリティでは、さらに大きなデータチャンクを調査できるため、MySQL の各ページで見つかるよりも多くの繰り返し文字列が巨大なファイルで見つかる可能性があります。

特定のテーブルの圧縮をテストする別の方法は、圧縮されていないテーブルの一部のデータを file-per-table テーブルスペース内の類似した圧縮テーブル (すべて同じインデックスを持つ) にコピーし、結果の .ibd ファイルのサイズを確認することです。例:

```
USE test;
SET GLOBAL innodb_file_per_table=1;
SET GLOBAL autocommit=0;

-- Create an uncompressed table with a million or two rows.
CREATE TABLE big_table AS SELECT * FROM information_schema.columns;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
COMMIT;
ALTER TABLE big_table ADD id int unsigned NOT NULL PRIMARY KEY auto_increment;

SHOW CREATE TABLE big_table\G

select count(id) from big_table;

-- Check how much space is needed for the uncompressed table.
! ls -l data/test/big_table.ibd

CREATE TABLE key_block_size_4 LIKE big_table;
ALTER TABLE key_block_size_4 key_block_size=4 row_format=compressed;
```



```
INSERT INTO key_block_size_4 SELECT * FROM big_table;
commit;

-- Check how much space is needed for a compressed table
-- with particular compression settings.
\! ls -l data/test/key_block_size_4.ibd
```

この実験では、次のような数値が生成されました。当然、テーブル構造やデータによって、数値が大幅に異なる可能性があります。

```
-rw-rw---- 1 cirrus staff 310378496 Jan 9 13:44 data/test/big_table.ibd
-rw-rw---- 1 cirrus staff 83886080 Jan 9 15:10 data/test/key_block_size_4.ibd
```

特定の**ワークロード**で圧縮が効率的かどうかを確認するには:

- 単純なテストでは、その他の圧縮テーブルが含まれない MySQL インスタンスを使用し、`INFORMATION_SCHEMA.INNODB_CMP` テーブルに対してクエリーを実行します。
- 複数の圧縮テーブルが含まれるワークロードが関与するより詳細なテストでは、`INFORMATION_SCHEMA.INNODB_CMP_PER_INDEX` テーブルに対してクエリーを実行します。`INNODB_CMP_PER_INDEX` テーブルの統計は収集にコストがかかるため、そのテーブルをクエリーする前に構成オプション `innodb_cmp_per_index_enabled` を有効にする必要があります、そのようなテストを開発サーバーまたはクリティカルでないレプリカサーバーに制限できます。
- テスト中の圧縮テーブルに対して、一般的な SQL ステートメントをいくつか実行します。
- `INFORMATION_SCHEMA.INNODB_CMP` または `INFORMATION_SCHEMA.INNODB_CMP_PER_INDEX` テーブルのクエリーを実行し、`COMPRESS_OPS` と `COMPRESS_OPS_OK` を比較することで、圧縮操作全体に対する正常な圧縮操作の比率を調査します。
- 圧縮操作が正常に完了した比率が高い場合は、そのテーブルが圧縮対象の候補である可能性が高くなります。
- **圧縮が失敗**する比率が高い場合は、[セクション15.9.1.6「OLTP ワークロードの圧縮」](#)で説明したように、`innodb_compression_level`、`innodb_compression_failure_threshold_pct`、および `innodb_compression_pad_pct_max` オプションを調整すれば、さらに詳細なテストを試すことができます。

データベースの圧縮とアプリケーションの圧縮

アプリケーション内とテーブル内のどちらでデータを圧縮するかどうかを決定します。同じデータで両方のタイプの圧縮を使用しないでください。アプリケーション内でデータを圧縮し、その結果を圧縮テーブルに格納すると、追加の領域が節約される可能性は大幅に低くなり、二重圧縮によって単に CPU サイクルが無駄になるだけです。

データベース内での圧縮

これを有効にすると、MySQL テーブルの圧縮は自動的にになり、すべてのカラムおよびインデックス値に適用されます。`LIKE` などの演算子を含むカラムも引き続きテストでき、インデックス値が圧縮されている場合でも、ソート操作でインデックスを引き続き使用できます。多くの場合、インデックスがデータベースの合計サイズの相当な割合を占めるため、圧縮を使用すると、ストレージ、I/O、またはプロセッサ時間が大幅に節約される可能性があります。圧縮および圧縮解除の操作は、予期される負荷を処理できるようにサイズ変更された強力なシステムとなる可能性が高いデータベースサーバー上で発生します。

アプリケーション内での圧縮

テキストなどのデータをアプリケーション内で圧縮してから、データベースに挿入する場合は、一部のカラムは圧縮されるが、その他は圧縮されないことで効率的に圧縮されないデータで、オーバーヘッドが節約される可能性があります。このアプローチでは、圧縮および圧縮解除用の CPU サイクルがデータベースサーバー上ではなく、クライアントマシン上で使用されるため、多数のクライアントが含まれる分散アプリケーションや、予備の CPU サイクルを備えたクライアントマシンに適している場合があります。

ハイブリッドアプローチ

当然、これらのアプローチは組み合わせることができます。一部のアプリケーションでは、いくつかの圧縮テーブルといくつかの非圧縮テーブルを使用することが適切である場合があります。一部のデータを外部で圧縮して(それを

非圧縮テーブルに格納して)、アプリケーション内のその他のテーブル (の一部) を MySQL で圧縮できるようにすることが最適な方法である場合もあります。通常どおり、適切な決定に達するには、事前の設計および現実のテストが重要となります。

ワークロードの特性と圧縮

圧縮するテーブル (およびページサイズ) を選択することに加えて、ワークロードはもう 1 つのパフォーマンスの主要な決定要因でもあります。アプリケーションが更新ではなく、読み取りで占有されている場合は、圧縮済みデータ用に MySQL で保持されるページごとの「変更ログ」用の空き領域がインデックスページによって使い果たされたあとに、再編成および再圧縮する必要のあるページが少なくなります。更新によって、インデックスなしのカラムまたはそれらが含まれている BLOB や、偶然に「オフページ」に格納される大きな文字列が主に変更される場合は、圧縮のオーバーヘッドが許容可能になる可能性があります。単調に増加する主キーを使用する INSERT がテーブルへの唯一の変更であり、セカンダリインデックスがほとんどない場合は、インデックスページを再編成および再圧縮する必要もほとんどありません。MySQL では、非圧縮データを変更することで、「適切に」、圧縮済みページ上のデータに「削除マークを付け」てから削除できるため、テーブル上の DELETE 操作は比較的効率的に行われます。

環境によっては、データのロードに要する時間がリアルタイム検索と同じくらいに重要である場合があります。特にデータウェアハウス環境では、数多くのテーブルが読み取り専用または読み取りが大半になっている可能性があります。このような場合、結果として少数のディスク読み取りとストレージコストの節約が重要である場合を除いて、ロード時間が長くなるという点で圧縮の犠牲を払うことが許容できる場合と、許容できない場合があります。

本来は、データを圧縮および圧縮解除する際に CPU 時間を使用できるときに、圧縮が最適に動作します。そのため、ワークロードが CPU バウンドではなく、I/O バウンドである場合に、圧縮を使用することで全体的なパフォーマンスを改善できることがわかるでしょう。さまざまな圧縮構成でアプリケーションのパフォーマンスをテストする際は、計画した本番システム構成と同様のプラットフォーム上でテストしてください。

構成の特性と圧縮

データベースページのディスクからの読み取りとディスクへの書き込みは、システムパフォーマンスのもっとも低速な側面です。圧縮では、CPU 時間を使用してデータを圧縮および圧縮解除することで I/O の削減が試みられるため、プロセスサイクルと比べて、I/O が比較的少ないリソースであるときに、もっとも効率性が高くなります。

多くの場合、これは特に、高速のマルチコア CPU が搭載された複数ユーザー環境で動作しているときに当てはまります。圧縮テーブルのページがメモリー内にあるときは、MySQL では多くの場合、ページの非圧縮コピー用の **バッファプール** 内で追加のメモリー (一般に 16K バイト) が使用されます。適応型 LRU アルゴリズムでは、I/O バウンドと CPU バウンドのどちらの方式でワークロードが動作しているのに関係なく、考慮される圧縮済みページと非圧縮ページ間でメモリー使用のバランスを調整しようと試みられます。メモリーが非常に制約されている構成よりも、バッファプール専用のメモリーがより多く搭載された構成の方が、圧縮テーブルを使用するときに適切に動作する傾向があります。

圧縮済みページサイズの選択

圧縮済みページサイズの最適な設定は、テーブルおよびそのインデックスに含まれるデータの型および分布によって異なります。圧縮済みページのサイズは、常に最大のレコードサイズよりも大きくするようにしてください。そうでなければ、**B ツリーページの圧縮** で注記したように、操作に失敗する可能性があります。

圧縮済みページサイズの設定が大きすぎると、一部の領域が無駄になりますが、頻繁にページを圧縮する必要はなくなります。圧縮されたページサイズが小さすぎる場合、挿入または更新に時間のかかる再圧縮が必要になることがあり、**B-tree** ノードをより頻繁に分割する必要があるため、データファイルが大きくなり、インデックス付けの効率が低下する可能性があります。

一般に、圧縮済みページサイズは 8K バイトまたは 4K バイトに設定されます。InnoDB テーブルの最大行サイズが約 8K とすれば、通常、**KEY_BLOCK_SIZE=8** は安全な選択です。

15.9.1.4 実行時の InnoDB テーブル圧縮の監視

アプリケーション全体のパフォーマンス、CPU と I/O の使用率、およびディスクファイルのサイズは、アプリケーションでの圧縮の効率性を示す適切な指標です。このセクションは、**セクション 15.9.1.3 「InnoDB テーブルの圧縮の調整」** に示したパフォーマンスチューニングのアドバイスに基づいて構成され、初期のテスト時には発生する可能性のない問題を見つける方法を示しています。

圧縮テーブルのパフォーマンス上の考慮事項をさらに深く掘り下げるには、例15.1「圧縮情報スキーマテーブルの使用」に記載した「情報スキーマ」テーブルを使用すれば、実行時に圧縮のパフォーマンスをモニターできます。これらのテーブルは、メモリーの内部使用および全体的に使用される圧縮の比率を反映しています。

`INNODB_CMP` テーブルには、使用中の圧縮済みページサイズ (`KEY_BLOCK_SIZE`) ごとに、圧縮アクティビティーに関する情報がレポートされます。これらのテーブル内の情報は、システム全体のものであり、データベース内のすべての圧縮テーブルにわたる圧縮の統計を集約したものです。このデータを使用すると、その他の圧縮テーブルがアクセスしていないときに、これらのテーブルを調査することでテーブルを圧縮するかどうかを決定する際に役立ちます。これには、サーバー上で比較的小さいオーバーヘッドが伴うため、圧縮失敗の全体的な効率性をチェックするために、本番環境サーバー上で定期的にクエリーを実行することがあります。

`INNODB_CMP_PER_INDEX` テーブルには、個別のテーブルおよびインデックスごとに、圧縮アクティビティーに関する情報がレポートされます。この情報は、圧縮の効率性を評価し、一度に1つのテーブルまたはインデックスのパフォーマンス問題を診断する際に、よりの絞ることができ、より役立ちます。(各 InnoDB テーブルはクラスタ化されたインデックスとして表されるため、このコンテキストでは、MySQL でテーブルとインデックス間で大きな区別が行われません。) `INNODB_CMP_PER_INDEX` テーブルには大量のオーバーヘッドが伴うため、さまざまなワークロード、データ、および圧縮設定の効果を分離して比較できる開発サーバーにより適しています。このモニタリングのオーバーヘッドが誤って課されることを防ぐには、`INNODB_CMP_PER_INDEX` テーブルのクエリーを実行する前に、`innodb_cmp_per_index_enabled` 構成オプションを有効にする必要があります。

考慮すべき主要な統計は、圧縮および圧縮解除操作の数、および実行に要する時間数です。MySQL では、変更後に B-tree ノードがいっぱいになりすぎて圧縮データを含めることができない場合に分割されるため、「成功」圧縮操作の数とそのような操作の数を全体で比較します。`INNODB_CMP` および `INNODB_CMP_PER_INDEX` テーブル内の情報、およびアプリケーション全体のパフォーマンスとハードウェアリソースの使用率に基づいて、ハードウェア構成の変更を行ったり、バッファプールのサイズを調整したり、別のページサイズを選択したり、圧縮する別のテーブルセットを選択したりすることがあります。

圧縮および圧縮解除するために必要な CPU 時間の合計が大きい場合は、高速またはマルチコアの CPU に変更すると、同じデータ、アプリケーションのワークロード、および圧縮テーブルのセットを使用してパフォーマンスを改善する際に役立つことがあります。バッファプールのサイズを大きくすると、パフォーマンスの改善に役立つこともあります。これにより、より多くの非圧縮ページをメモリー内に滞在できるようになるため、圧縮形式でのみメモリー内に存在するページを圧縮解除する必要が少なくなります。

(アプリケーションでの `INSERT`、`UPDATE`、および `DELETE` 操作の数、およびデータベースのサイズと比較して) 圧縮操作全体の数が大きい場合は、効率的な圧縮としては、圧縮テーブルの一部が更新される頻度が高すぎることを示している可能性があります。その場合は、より大きなページサイズを選択するか、圧縮するテーブルをより慎重に選択してください。

「正常な」圧縮操作の数 (`COMPRESS_OPS_OK`) が圧縮操作の合計数 (`COMPRESS_OPS`) の高い比率を占めている場合は、システムが正常に実行されている可能性が高くなります。比率が低い場合は、MySQL によって理想よりも頻繁に、B ツリーノードの再編成、再圧縮、および分割が行われます。この場合、一部のテーブルの圧縮を回避するか、圧縮テーブルの一部で `KEY_BLOCK_SIZE` を大きくしてください。テーブルの圧縮をオフにすると、アプリケーション内での「圧縮失敗」の数が合計の 1% または 2% を上回る可能性があります。(このような失敗の比率は、データのロードなどの一時的な操作時には許容範囲内である場合もあります)。

15.9.1.5 InnoDB テーブルでの圧縮の動作

このセクションでは、InnoDB テーブルの圧縮に関する一部の内部実装について詳細に説明します。ここで示す情報は、パフォーマンスを調整する際に役立つがありますが、圧縮の基本的な使用を理解する必要はありません。

圧縮アルゴリズム

一部のオペレーティングシステムでは、ファイルシステムのレベルで圧縮が実装されています。一般に、ファイルは、可変サイズのブロックに圧縮される固定サイズのブロックに分割されるため、簡単に断片化されます。ブロック内部で何かの変更されるたびに、ブロック全体が再圧縮されてからディスクに書き込まれます。これらのプロパティーを使用すると、この圧縮方法が更新の多いデータベースシステムでの使用には適さなくなります。

MySQL では、LZ77 圧縮アルゴリズムが実装されている有名な `zlib` ライブラリの支援を得て、圧縮が実装されています。この圧縮アルゴリズムは十分に発達し、強固であり、CPU の使用率とデータサイズの削減の両方の点で効率的です。このアルゴリズムは「損失なし」であるため、常に、元の非圧縮データを圧縮形式から再構築できます。LZ77

圧縮は、圧縮されるデータ内で繰り返される一連のデータを見つけることで動作します。データ内の値のパターンによって、圧縮の効率性が決定されますが、多くの場合、一般的なユーザーデータは 50% 以上圧縮されます。

注記

InnoDB は、MySQL 8.0 にバンドルされているバージョンであるバージョン 1.2.11 までの `zlib` ライブラリをサポートしています。

アプリケーションで実行される圧縮や、その他の一部のデータベース管理システムの圧縮機能とは異なり、InnoDB の圧縮は、ユーザーデータとインデックスの両方に適用されます。多くの場合、インデックスがデータベースの合計サイズの 40-50% 以上を占める可能性があるため、この相違点は重要です。データセットの圧縮が適切に機能している場合、InnoDB データファイル (`file-per-table` テーブルスペースまたは `general_tablespace .ibd` ファイル) のサイズは、圧縮されていないサイズの 25% から 50% 以下になります。ワークロードによっては、このようにデータベースを小さくすることにより、CPU 使用率を少し増加させるだけで I/O を削減してスループットを増加できます。`innodb_compression_level` 構成オプションを変更すると、圧縮のレベルと CPU のオーバーヘッド間のバランスを調整できます。

InnoDB データストレージと圧縮

InnoDB テーブル内のすべてのユーザーデータは、**B ツリーインデックス (クラスタ化されたインデックス)** を構成しているページに格納されます。その他の一部のデータベースシステムでは、このタイプのインデックスは「インデックス編成テーブル」と呼ばれます。インデックスノード内の各行には、(ユーザーが指定した、またはシステムで生成された) **主キー** の値およびテーブルのその他のすべてのカラムが含まれています。

InnoDB テーブル内の **セカンダリインデックス** は、値のペア (インデックスキーと、クラスタ化されたインデックス内の行へのポインタ) を含む B ツリーでもあります。実際は、ポインタはテーブルの主キーの値であり、インデックスキーおよび主キー以外のカラムが必要な場合に、クラスタ化されたインデックスにアクセスする際に使用されます。常に、セカンダリインデックスのレコードは、B ツリーページ上に収容される必要があります。

次のセクションで説明するように、(クラスタ化インデックスとセカンダリインデックスの両方の) B ツリーノードの圧縮は、長い `VARCHAR`、`BLOB`、または `TEXT` カラムを格納するために使用される **オーバーフロー** の圧縮とは異なる方法で処理されます。

B ツリーページの圧縮

B ツリーページは頻繁に更新されるため、特別な処理が必要です。B ツリーノードが分割される回数を最小限にし、それらの内容を圧縮解除および再圧縮する必要性も最小限にすることが重要となります。

MySQL で使用される技術の 1 つでは、一部のシステム情報が非圧縮形式で B ツリーノード内に保持されるため、特定のインプレース更新が容易になります。たとえば、これにより、圧縮操作なしで行に削除のマークを付け、その行を削除できます。

さらに、MySQL では、インデックスページが変更されたときに、不要な圧縮解除および再圧縮を回避しようと試みられます。システムの各 B ツリーページ内には、ページに行われた変更を記録するための非圧縮の「変更ログ」が保持されます。小さいレコードの更新および挿入は、ページ全体を完全に再構築する必要なしで、この変更ログに書き込まれる場合があります。

変更ログ用の領域を使い果たすと、InnoDB によってページが圧縮解除され、変更が適用され、ページが再圧縮されます。再圧縮に失敗すると (**圧縮の失敗** と呼ばれる状況)、B ツリーノードが分割され、更新または挿入に成功するまでプロセスが繰り返されます。

OLTP アプリケーションなどで、書き込み負荷の高いワークロードでの頻繁な圧縮の失敗を回避するために、MySQL では、ページ内にいくつかの空のスペース (パディング) が予約されている場合があります。これにより、変更ログがより早く埋められ、分割を回避するための十分な空き領域がまだある間にページが再圧縮されます。各ページに残されるパディングスペースの量は、システムでページ分割の頻度が追跡されるにつれて変化します。圧縮テーブルへの書き込みが頻繁に行われる高負荷のサーバー上では、`innodb_compression_failure_threshold_pct` および `innodb_compression_pad_pct_max` 構成オプションを調整すると、このメカニズムを微調整できます。

一般に、MySQL では、InnoDB テーブル内の各 B ツリーページに 2 つ以上のレコードを収容できます。圧縮テーブルに対しては、この要件が緩和されました。B ツリーノードのリーフページには (主キーとセカンダリインデックスのどちらでも)、1 つのレコードのみが収容される必要がありますが、そのレコードはページごとの変更ログに非圧縮形式で収まる必要があります。`innodb_strict_mode` が `ON` の場合は、`CREATE TABLE` または `CREATE INDEX` の実

行中に、MySQL によって行の最大サイズがチェックされます。行が収まらない場合は、「[ERROR HY000: Too big row](#)」というエラーメッセージが発行されます。

`innodb_strict_mode` が OFF のときにテーブルを作成した場合に、後続の `INSERT` または `UPDATE` ステートメントで圧縮済みページのサイズに収まらないインデックスエントリの作成が試みられると、その操作に失敗し、「[ERROR 42000: Row size too large](#)」というエラーが表示されます。(このエラーメッセージは、レコードが長すぎるインデックスの名前を示すものでも、その特定のインデックスページ上のインデックスレコードの長さや最大レコードサイズを示すものでもありません。)この問題を解決するには、`ALTER TABLE` を使用してテーブルを再構築し、より大きな圧縮済みページサイズ (`KEY_BLOCK_SIZE`) を選択して、任意のカラムプリフィックスのインデックスを短くするか、`ROW_FORMAT=DYNAMIC` または `ROW_FORMAT=COMPACT` を使用して圧縮を完全に無効にします。

`innodb_strict_mode` は、圧縮テーブルもサポートする一般的なテーブルスペースには適用できません。一般的なテーブルスペースのテーブルスペース管理ルールは、`innodb_strict_mode` とは無関係に厳密に適用されます。詳細は、[セクション 13.1.21「CREATE TABLESPACE ステートメント」](#) を参照してください。

BLOB、VARCHAR、および TEXT カラムの圧縮

InnoDB テーブルでは、主キーの一部ではない `BLOB`、`VARCHAR`、および `TEXT` カラムが、個別に割り当てられた [オーバーフローページ](#) に格納される場合があります。このようなカラムは、[オフページカラム](#) と呼ばれています。これらの値は、オーバーフローページの片方向リストに格納されます。

`ROW_FORMAT=DYNAMIC` または `ROW_FORMAT=COMPRESSED` で作成されたテーブルでは、カラムの長さおよび行全体の長さによっては、`BLOB`、`TEXT`、または `VARCHAR` カラムの値が完全にオフページに格納される場合もあります。オフページに格納されるカラムでは、クラスタ化されたインデックスのレコードに、オーバーフローページへの 20 バイトのポインタのみがカラムごとに 1 つずつ含まれます。カラムがオフページに格納されるかどうかは、ページサイズおよび行の合計サイズによって異なります。行がクラスタ化されたインデックスのページ内に完全に収まらないほど長い場合は、クラスタ化されたインデックスページ上に行が収まるまで、MySQL によってオフページストレージに合った最長のカラムが選択されます。前述の注で示したように、行自体が圧縮済みページ上に収まらない場合は、エラーが発生します。

注記

`ROW_FORMAT=DYNAMIC` または `ROW_FORMAT=COMPRESSED` で作成されたテーブルでは、40 バイト以下の `TEXT` および `BLOB` カラムは、常にインラインに格納されます。

`ROW_FORMAT=REDUNDANT` および `ROW_FORMAT=COMPACT` を使用するテーブルでは、`BLOB`、`VARCHAR` および `TEXT` カラムの最初の 768 バイトが主キーとともにクラスタインデックスレコードに格納されます。768 バイトのプリフィックスのあとには、残りのカラム値を含むオーバーフローページへの 20 バイトのポインタが続きます。

テーブルの形式が `COMPRESSED` である場合は、オーバーフローページに書き込まれるすべてのデータが「そのまま」圧縮されます。つまり、MySQL では、データ項目全体に `zlib` 圧縮アルゴリズムが適用されます。圧縮済みのオーバーフローページには、データ以外では特に、ページチェックサムを構成する非圧縮のヘッダーとトレーラ、および次のオーバーフローページへのリンクが含まれます。したがって、テキストデータを使用した場合に多く見られるように、データの圧縮性が高い場合は、長い `BLOB`、`TEXT`、または `VARCHAR` カラムで非常に大幅なストレージの節約が実現されます。一般に、`JPEG` などのイメージデータはすでに圧縮されているため、圧縮テーブルに格納される利点がほとんど得られません。領域の節約がほとんどない、またはまったくない場合は、二重圧縮によって CPU サイクルが無駄になる可能性があります。

オーバーフローページのサイズは、その他のページと同じです。カラムの合計長が 8K バイトのみである場合でも、オフページに格納される 10 個のカラムを含む行で、10 個のオーバーフローページが占有されます。非圧縮テーブルでは、10 個の非圧縮オーバーフローページで 160K バイトが占有されます。ページサイズが 8K の圧縮テーブルでは、80K バイトのみが占有されます。そのため、長いカラム値を含むテーブルでは、圧縮テーブル形式を使用すると効率性が高くなることが多くあります。

`file-per-table` テーブルスペースでは、`BLOB`、`VARCHAR` または `TEXT` カラムの記憶域および I/O コストを 16K 圧縮ページサイズを使用すると削減できます。これは、これらのデータが圧縮されることが多いため、B ツリーノード自体が圧縮されていない形式と同じ数のページを使用しても、オーバーフローページが必要になる場合があるためです。一般テーブルスペースでは、16K 圧縮ページサイズ (`KEY_BLOCK_SIZE`) はサポートされていません。詳細は、[セクション 15.6.3.3「一般テーブルスペース」](#) を参照してください。

圧縮と InnoDB バッファプール

圧縮された InnoDB テーブルでは、すべての圧縮ページ (1K、2K、4K または 8K) が 16K バイト (`innodb_page_size` が設定されている場合は小さいサイズ) の圧縮されていないページに対応します。ページ内のデータにアクセスするために、MySQL は、圧縮済みページが **バッファプール** 内にすでに存在しない場合、そのページをディスクから読み取ってから、その元の形式に圧縮解除します。このセクションでは、InnoDB が圧縮テーブルのページに関して **バッファプール** を管理する方法について説明します。

I/O を最小限にして、ページを圧縮解除する必要性を削減するために、**バッファプール** に圧縮済み形式と非圧縮形式の両方のデータベースページが含まれることがあります。その他の必要なデータベースページ用の空き領域を作成するために、MySQL ではメモリー内に圧縮済みページを残しながら、**バッファプール** から非圧縮ページを **エビクション** できます。また、しばらくの間ページがアクセスされていない場合は、その他のデータ用に領域を解放するために、圧縮形式のページがディスクに書き込まれることもあります。したがって、そのときどきで、**バッファプール** に圧縮形式と非圧縮形式の両方のページが含まれている場合、圧縮形式のページのみが含まれている場合、どちらも含まれていない場合があります。

MySQL では、**ホット** (頻繁にアクセスされる) データがメモリー内に滞在する傾向となるように、最近もつとも使用されていない (LRU) リストを使用して、メモリー内に保持されるページおよび削除されるページが追跡されます。圧縮テーブルにアクセスすると、MySQL は適応型 LRU アルゴリズムを使用して、メモリー内の圧縮済みページと非圧縮ページの適切なバランスを実現します。この適応型アルゴリズムは、システムが **I/O バウンド** と **CPU バウンド** のどちらの方式で実行されているかどうかの影響を受けやすくなります。この目的は、CPU の負荷が高いときにページを圧縮解除するために要する処理時間が長くなりすぎることを回避すること、および (メモリー内にすでに存在する可能性のある) 圧縮済みページを圧縮解除するために使用できる予備のサイクルが CPU に備わっているときに過剰な I/O が発生することを回避することです。システムが I/O バウンドの場合、このアルゴリズムでは、その他のディスクページ用により多くの空き領域を作成することでメモリーが常駐になるように、ページの両方のコピーではなく、非圧縮コピーを削除することが優先されます。システムが CPU バウンドの場合、MySQL では、「ホット」ページ用に使用できるメモリーが多くなり、圧縮形式でのみメモリー内のデータを圧縮解除する必要性が少なくなるように、圧縮済みページと非圧縮ページの両方を削除することが優先されます。

圧縮と InnoDB の Redo ログファイル

圧縮済みページが **データファイル** に書き込まれる前に、MySQL によってページのコピーが Redo ログに書き込まれます (最後にデータベースに書き込まれた以降に再圧縮された場合)。これは、`zlib` ライブラリがアップグレードされ、その変更によって圧縮済みデータとの互換性の問題が発生する可能性が低い場合でも、**クラッシュリカバリ** 時に Redo ログを使用できるかどうかを確認するために行われます。したがって、圧縮の使用時に、**ログファイル** のサイズを多少大きくすること、またはより頻繁に **チェックポイント** を発生させる必要性を多少多くすることが要求される可能性があります。ログファイルのサイズを大きくする量またはチェックポイントの頻度を多くする数は、再構成および再圧縮が必要となる方法で圧縮済みページが変更される回数によって異なります。

file-per-table テーブルスペースに圧縮テーブルを作成するには、`innodb_file_per_table` が有効になっている必要があります。一般的なテーブルスペースに圧縮テーブルを作成する場合、`innodb_file_per_table` 設定には依存しません。詳細は、**セクション 15.6.3.3 「一般テーブルスペース」** を参照してください。

15.9.1.6 OLTP ワークロードの圧縮

従来、InnoDB の **圧縮機能** は、**データウェアハウス** 構成などで、主に読み取り専用または読み取りが大半の **ワークロード** に対して使用することが推奨されていました。高速だが比較的小規模でコストがかかる **SSD** ストレージデバイスの増加により、**OLTP** ワークロードにも圧縮が魅力的になります: トラフィック量の多い対話型 web サイトでは、**INSERT**、**UPDATE** および **DELETE** の操作を頻繁に行うアプリケーションで圧縮テーブルを使用することで、ストレージ要件および秒当たりの I/O 操作 (**IOPS**) を減らすことができます。

これらの構成オプションを使用すると、書き込み集中型操作のパフォーマンスとスケーラビリティに重点を置いて、特定の MySQL インスタンスの圧縮方法を調整できます:

- `innodb_compression_level` を使用すると、圧縮の程度を上げたり、下げたりできます。値を大きくすると、ストレージデバイス上に収容できるデータ量が多くなりますが、圧縮時の CPU オーバーヘッドも多くなるという犠牲が伴います。値を小さくすると、ストレージ領域がクリティカルでない場合に、CPU のオーバーヘッドを削減できます。それ以外の場合は、データが特に圧縮可能でないと予測されます。
- `innodb_compression_failure_threshold_pct` には、圧縮テーブルへの更新時に **圧縮が失敗** したときの **カットオフポイント** が指定されます。このしきい値を超えると、MySQL は、最大で `innodb_compression_pad_pct_max` で指定されたページサイズの割合まで空き領域の量を動的に調整することで、新しい各圧縮済みページ内に追加の空き領域を残し始めます。

- `innodb_compression_pad_pct_max` を使用すると、ページ全体を再度圧縮する必要なしで、変更を圧縮済み行に記録するための各ページ内に予約されている領域の最大量を調整できます。値を大きくすると、ページを再度圧縮せずに記録できる変更の量が多くなります。MySQL では、実行時に指定した割合の圧縮操作に「失敗した」ときのみ、各圧縮テーブル内にあるページ用に可変量の空き領域が使用されますが、圧縮済みページを分割するために負荷の高い操作が必要となります。
- `innodb_log_compressed_pages` では、redo log への re-compressed pages のイメージの書き込みを無効にできます。圧縮されたデータが変更されると、再圧縮が発生する場合があります。このオプションは、リカバリ時に異なるバージョンの zlib 圧縮アルゴリズムが使用された場合に発生する可能性がある破損を防ぐために、デフォルトで有効になっています。zlib のバージョンが変更されないことが確実な場合は、`innodb_log_compressed_pages` を無効にして、圧縮データを変更するワークロードの redo ログ生成を減らします。

圧縮済みデータを操作すると、圧縮済みと非圧縮の両方のバージョンのページが同時にメモリー内に保持されるため、OLTP スタイルのワークロードで圧縮を使用するときは、`innodb_buffer_pool_size` 構成オプションの値を大きくする準備をしてください。

15.9.1.7 SQL 圧縮構文の警告とエラー

このセクションでは、`file-per-table` テーブルスペースおよび `general tablespaces` でテーブル圧縮機能を使用する際に発生する可能性がある構文の警告およびエラーについて説明します。

File-Per-Table テーブルスペースに対する SQL 圧縮構文の警告およびエラー

`innodb_strict_mode` が有効な場合 (デフォルト)、`CREATE TABLE` ステートメントまたは `ALTER TABLE` ステートメントで `ROW_FORMAT=COMPRESSED` または `KEY_BLOCK_SIZE` を指定すると、`innodb_file_per_table` が無効な場合に次のエラーが生成されます。

```
ERROR 1031 (HY000): Table storage engine for 't1' doesn't have this option
```

注記

現在の構成では圧縮テーブルの使用が許可されていないため、テーブルは作成されません。

`innodb_strict_mode` が無効になっている場合、`CREATE TABLE` または `ALTER TABLE` ステートメントで `ROW_FORMAT=COMPRESSED` または `KEY_BLOCK_SIZE` を指定すると、`innodb_file_per_table` が無効になっていると次の警告が生成されます。

```
mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Warning | 1478 | InnoDB: KEY_BLOCK_SIZE requires innodb_file_per_table. |
| Warning | 1478 | InnoDB: ignoring KEY_BLOCK_SIZE=4. |
| Warning | 1478 | InnoDB: ROW_FORMAT=COMPRESSED requires innodb_file_per_table. |
| Warning | 1478 | InnoDB: assuming ROW_FORMAT=DYNAMIC. |
+-----+-----+-----+
```

注記

これらのメッセージは単なる警告であり、エラーではありません。オプションが指定されていない場合と同様に、テーブルは圧縮なしで作成されます。

「厳密でない」動作を使用すると、ソースデータベースに圧縮テーブルが含まれていない場合でも、圧縮テーブルがサポートされていないデータベースに `mysqldump` ファイルをインポートできます。その場合、MySQL は、操作を妨げるかわりに `ROW_FORMAT=DYNAMIC` にテーブルを作成します。

ダンプファイルを新しいデータベースにインポートし、元のデータベースに存在するとおりにテーブルを再作成するには、サーバーに `innodb_file_per_table` 構成パラメータの適切な設定があることを確認します。

`KEY_BLOCK_SIZE` 属性は、`ROW_FORMAT` が `COMPRESSED` として指定されているか、省略されている場合のみ許可されます。その他の `ROW_FORMAT` とともに `KEY_BLOCK_SIZE` を指定すると、`SHOW WARNINGS` を使用して表示できる警告が生成されます。ただし、テーブルは非圧縮です。つまり、指定された `KEY_BLOCK_SIZE` は無視されます。

レベル	コード	メッセージ
警告	1478	InnoDB: ignoring KEY_BLOCK_SIZE=n unless ROW_FORMAT=COMPRESSED.

`innodb_strict_mode` が有効になっている状態で実行している場合は、`COMPRESSED` 以外の任意の `ROW_FORMAT` と `KEY_BLOCK_SIZE` を組み合わせると警告ではなく、エラーが生成され、テーブルは作成されません。

表15.12 「`ROW_FORMAT` および `KEY_BLOCK_SIZE` のオプション」では、`CREATE TABLE` または `ALTER TABLE` で使用される `ROW_FORMAT` および `KEY_BLOCK_SIZE` のオプションの概要を示します。

表 15.12 `ROW_FORMAT` および `KEY_BLOCK_SIZE` のオプション

オプション	使用上の注意	説明
<code>ROW_FORMAT=REDUNDANT</code>	MySQL 5.0.3 よりも前で使用されていたストレージフォーマット	<code>ROW_FORMAT=COMPACT</code> よりも効率性が低く、下位互換性を保つためのものです。
<code>ROW_FORMAT=COMPACT</code>	MySQL 5.0.3 以降でのデフォルトのストレージフォーマット	クラスタ化されたインデックスページに、768 バイトの長いカラム値のプリフィクスが格納され、残りのバイトはオーバーフローページに格納されます。
<code>ROW_FORMAT=DYNAMIC</code>		クラスタ化されたインデックスページ内に収まる場合は、そのページ内に値が保存されます。収まらない場合は、オーバーフローページへの 20 バイトのポインタのみが (プリフィクスなしで) 格納されます。
<code>ROW_FORMAT=COMPRESSED</code>		<code>zlib</code> を使用してテーブルとインデックスを圧縮
<code>KEY_BLOCK_SIZE=n</code>		圧縮されたページサイズとして 1、2、4、8 または 16 KB を指定します。これは <code>ROW_FORMAT=COMPRESSED</code> を意味します。一般的なテーブルスペースの場合、InnoDB ページサイズと等しい <code>KEY_BLOCK_SIZE</code> 値は許可されません。

表15.13 「InnoDB 厳密モードがオフになっているときの `CREATE/ALTER TABLE` の警告とエラー」では、`CREATE TABLE` または `ALTER TABLE` ステートメント上で、構成パラメータとオプションの特定の組み合わせで発生するエラー状況、およびオプションが `SHOW TABLE STATUS` の出力に表示される方法について簡単に説明しています。

`innodb_strict_mode` が `OFF` の場合、MySQL によってテーブルが作成または変更されますが、次に示すように特定の設定は無視されます。警告メッセージは、MySQL エラーログで確認できます。`innodb_strict_mode` が `ON` の場合、このような特定のオプションの組み合わせでエラーが生成され、テーブルは作成または変更されません。エラー状況の完全な説明を参照するには、次に示すように、`SHOW ERRORS` ステートメントを発行します。

```
mysql> CREATE TABLE x (id INT PRIMARY KEY, c INT)
-> ENGINE=INNODB KEY_BLOCK_SIZE=33333;
ERROR 1005 (HY000): Can't create table 'test.x' (errno: 1478)

mysql> SHOW ERRORS;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Error | 1478 | InnoDB: invalid KEY_BLOCK_SIZE=33333. |
| Error | 1005 | Can't create table 'test.x' (errno: 1478) |
+-----+-----+-----+
```

表 15.13 InnoDB 厳密モードがオフになっているときの CREATE/ALTER TABLE の警告とエラー

構文	警告またはエラーの状況	結果として SHOW TABLE STATUS に表示される ROW_FORMAT
ROW_FORMAT=REDUNDANT	なし	REDUNDANT
ROW_FORMAT=COMPACT	なし	COMPACT
ROW_FORMAT=COMPRESSED または ROW_FORMAT=DYNAMIC、または KEY_BLOCK_SIZE が指定されている	innodb_file_per_table が有効になっていないかぎり、file-per-table テーブルスペースでは無視されます。一般テーブルスペースでは、すべての行形式がサポートされます。セクション 15.6.3.3 「一般テーブルスペース」を参照してください。	the default row format for file-per-table tablespaces; the specified row format for general tablespaces
無効な KEY_BLOCK_SIZE (1、2、4、8、または 16 以外) が指定されている	KEY_BLOCK_SIZE が無視されます。	指定された行フォーマットまたはデフォルトの行フォーマット
ROW_FORMAT=COMPRESSED および有効な KEY_BLOCK_SIZE が指定されている	なし。指定した KEY_BLOCK_SIZE が使用されます	COMPRESSED
REDUNDANT、COMPACT、または DYNAMIC 行フォーマットを使用して KEY_BLOCK_SIZE が指定されている	KEY_BLOCK_SIZE が無視されます。	REDUNDANT、COMPACT、または DYNAMIC
ROW_FORMAT が REDUNDANT、COMPACT、DYNAMIC または COMPRESSED のいずれでもない	MySQL パーサーで認識される場合は無視されます。その他の場合は、エラーが発行されます。	デフォルトの行フォーマットまたは N/A

innodb_strict_mode が ON の場合、MySQL は無効な ROW_FORMAT または KEY_BLOCK_SIZE パラメータを拒否し、エラーを発行します。厳密モードはデフォルトで ON です。innodb_strict_mode が OFF の場合、MySQL は無視された無効なパラメータに対してエラーではなく警告を発行します。

SHOW TABLE STATUS を使用して選択した KEY_BLOCK_SIZE を表示できません。SHOW CREATE TABLE ステートメントでは、(テーブルの作成時に無視された場合でも) KEY_BLOCK_SIZE が表示されます。テーブルの実際の圧縮済みページサイズは、MySQL では表示できません。

一般的なテーブルスペースに対する SQL 圧縮構文の警告およびエラー

- テーブルスペースの作成時に一般テーブルスペースに対して FILE_BLOCK_SIZE が定義されていない場合、テーブルスペースに圧縮テーブルを含めることはできません。圧縮テーブルを追加しようとすると、次の例に示すようにエラーが返されます:

```
mysql> CREATE TABLESPACE `ts1` ADD DATAFILE 'ts1.ibd' Engine=InnoDB;

mysql> CREATE TABLE t1 (c1 INT PRIMARY KEY) TABLESPACE ts1 ROW_FORMAT=COMPRESSED
  KEY_BLOCK_SIZE=8;
ERROR 1478 (HY000): InnoDB: Tablespace `ts1` cannot contain a COMPRESSED table
```

- 無効な KEY_BLOCK_SIZE を含むテーブルを一般テーブルスペースに追加しようとすると、次の例に示すようにエラーが返されます:

```
mysql> CREATE TABLESPACE `ts2` ADD DATAFILE 'ts2.ibd' FILE_BLOCK_SIZE = 8192 Engine=InnoDB;

mysql> CREATE TABLE t2 (c1 INT PRIMARY KEY) TABLESPACE ts2 ROW_FORMAT=COMPRESSED
  KEY_BLOCK_SIZE=4;
ERROR 1478 (HY000): InnoDB: Tablespace `ts2` uses block size 8192 and cannot
contain a table with physical page size 4096
```

一般的なテーブルスペースの場合、テーブルの KEY_BLOCK_SIZE は、テーブルスペースの FILE_BLOCK_SIZE を 1024 で割ったものである必要があります。たとえば、テーブルスペースの FILE_BLOCK_SIZE が 8192 の場合、テーブルの KEY_BLOCK_SIZE は 8 である必要があります。

- 圧縮されていない行形式のテーブルを、圧縮テーブルを格納するように構成された一般的なテーブルスペースに追加しようとすると、次の例に示すようにエラーが返されます:

```
mysql> CREATE TABLESPACE `ts3` ADD DATAFILE 'ts3.ibd' FILE_BLOCK_SIZE = 8192 Engine=InnoDB;
mysql> CREATE TABLE t3 (c1 INT PRIMARY KEY) TABLESPACE ts3 ROW_FORMAT=COMPACT;
ERROR 1478 (HY000): InnoDB: Tablespace `ts3` uses block size 8192 and cannot
contain a table with physical page size 16384
```

`innodb_strict_mode` は、一般的なテーブルスペースには適用できません。一般的なテーブルスペースのテーブルスペース管理ルールは、`innodb_strict_mode` とは無関係に厳密に適用されます。詳細は、[セクション13.1.21「CREATE TABLESPACE ステートメント」](#)を参照してください。

一般的なテーブルスペースでの圧縮テーブルの使用の詳細は、[セクション15.6.3.3「一般テーブルスペース」](#)を参照してください。

15.9.2 InnoDB ページ圧縮

InnoDB では、`file-per-table` テーブルスペースに存在するテーブルのページレベルの圧縮がサポートされます。この機能は、透過的ページ圧縮と呼ばれます。ページ圧縮を有効にするには、`CREATE TABLE` または `ALTER TABLE` で `COMPRESSION` 属性を指定します。サポートされている圧縮アルゴリズムには、`Zlib` および `LZ4` があります。

サポートされるプラットフォーム

ページ圧縮には、スパーズファイルおよびホールパンチのサポートが必要です。ページ圧縮は、NTFS を使用する Windows、およびカーネルレベルでホールパンチングサポートが提供される MySQL-supported Linux プラットフォームの次のサブセットでサポートされます:

- RHEL 7 およびカーネルバージョン 3.10.0-123 以降を使用する派生ディストリビューション
- OEL 5.10 (UEK2) カーネルバージョン 2.6.39 以上
- OEL 6.5 (UEK3) カーネルバージョン 3.8.13 以上
- OEL 7.0 カーネルバージョン 3.8.13 以上
- SLE11 カーネルバージョン 3.0-x
- SLE12 カーネルバージョン 3.12-x
- OES11 カーネルバージョン 3.0-x
- Ubuntu 14.0.4 LTS カーネルバージョン 3.13 以上
- Ubuntu 12.0.4 LTS カーネルバージョン 3.2 以上
- Debian 7 カーネルバージョン 3.2 以上

注記

特定の Linux ディストリビューションで使用可能なすべてのファイルシステムでホールパンチがサポートされていない場合があります。

ページ圧縮の仕組み

ページが書き込まれると、指定された圧縮アルゴリズムを使用して圧縮されます。圧縮されたデータはディスクに書き込まれ、ホールパンチングメカニズムによってページの最後から空のブロックが解放されます。圧縮に失敗すると、データはそのまま書き出されます。

Linux の穴パンチサイズ

Linux システムでは、ファイルシステムのブロックサイズはホールパンチに使用される単位サイズです。したがって、ページ圧縮は、InnoDB ページサイズからファイルシステムのブロックサイズを引いたサイズ以下のサイズにペー

ジデータを圧縮できる場合にのみ機能します。たとえば、`innodb_page_size=16K` でファイルシステムのブロックサイズが 4K の場合、ホールパンチを可能にするには、ページデータを 12K 以下に圧縮する必要があります。

Windows でのホールパンチングサイズ

Windows システムでは、疎ファイルの基盤となるインフラストラクチャは NTFS 圧縮に基づいています。ホールパンチングサイズは NTFS 圧縮ユニットで、NTFS クラスタサイズの 16 倍です。次のテーブルに、クラスタサイズとその圧縮単位を示します:

表 15.14 Windows NTFS クラスタのサイズと圧縮単位

クラスタサイズ	圧縮単位
512 バイト	8 KB
1 KB	16 KB
2 KB	32 KB
4 KB	64 KB

Windows システムでのページ圧縮は、ページデータを InnoDB ページサイズから圧縮単位サイズを引いた値以下のサイズに圧縮できる場合にのみ機能します。

NTFS クラスタのデフォルトサイズは 4KB で、圧縮単位のサイズは 64KB です。つまり、デフォルトの Windows NTFS 構成では、最大 `innodb_page_size` も 64KB であるため、ページ圧縮にメリットはありません。

ページ圧縮を Windows で機能させるには、ファイルシステムを 4K より小さいクラスタサイズで作成し、`innodb_page_size` を圧縮ユニットの 2 倍以上のサイズにする必要があります。たとえば、ページ圧縮が Windows で機能するようにするには、512 バイト (圧縮単位は 8KB) のクラスタサイズでファイルシステムを構築し、16K 以上の `innodb_page_size` 値で InnoDB を初期化します。

ページ圧縮の有効化

ページ圧縮を有効にするには、`CREATE TABLE` ステートメントで `COMPRESSION` 属性を指定します。例:

```
CREATE TABLE t1 (c1 INT) COMPRESSION="zlib";
```

`ALTER TABLE` ステートメントでページ圧縮を有効にすることもできます。ただし、`ALTER TABLE ... COMPRESSION` ではテーブルスペース圧縮属性のみが更新されます。新しい圧縮アルゴリズムの設定後に発生したテーブルスペースへの書き込みでは新しい設定が使用されますが、新しい圧縮アルゴリズムを既存のページに適用するには、`OPTIMIZE TABLE` を使用してテーブルを再構築する必要があります。

```
ALTER TABLE t1 COMPRESSION="zlib";
OPTIMIZE TABLE t1;
```

ページ圧縮の無効化

ページ圧縮を無効にするには、`ALTER TABLE` を使用して `COMPRESSION=None` を設定します。`COMPRESSION=None` の設定後に発生するテーブルスペースへの書き込みでは、ページ圧縮は使用されなくなりました。既存のページを解凍するには、`COMPRESSION=None` の設定後に `OPTIMIZE TABLE` を使用してテーブルを再構築する必要があります。

```
ALTER TABLE t1 COMPRESSION="None";
OPTIMIZE TABLE t1;
```

ページ圧縮メタデータ

ページ圧縮メタデータは、`INFORMATION_SCHEMA.INNODB_TABLESPACES` テーブルの次のカラムにあります:

- `FS_BLOCK_SIZE`: ホールパンチングに使用される単位サイズであるファイルシステムのブロックサイズ。
- `FILE_SIZE`: 圧縮解除されたファイルの最大サイズを表す、ファイルの見かけ上のサイズ。

- **ALLOCATED_SIZE**: ファイルの実際のサイズ。これは、ディスクに割り当てられた領域の量です。

注記

Unix に似たシステムでは、`ls -l tablespace_name.ibd` は明らかなファイルサイズ (**FILE_SIZE** と同等) をバイト単位で表示します。ディスクに割り当てられている実際の領域の量 (**ALLOCATED_SIZE** と同等) を表示するには、`du --block-size=1 tablespace_name.ibd` を使用します。`--block-size=1` オプションは、割り当てられた領域をブロックではなくバイト単位で出力するため、`ls -l` 出力と比較できます。

SHOW CREATE TABLE を使用して、現在のページ圧縮設定 (**Zlib**、**Lz4** または **None**) を表示します。テーブルには、異なる圧縮設定を持つページが混在する場合があります。

次の例では、`employees` テーブルのページ圧縮メタデータが `INFORMATION_SCHEMA.INNODB_TABLESPACES` テーブルから取得されます。

```
# Create the employees table with Zlib page compression

CREATE TABLE employees (
  emp_no INT NOT NULL,
  birth_date DATE NOT NULL,
  first_name VARCHAR(14) NOT NULL,
  last_name VARCHAR(16) NOT NULL,
  gender ENUM ('M','F') NOT NULL,
  hire_date DATE NOT NULL,
  PRIMARY KEY (emp_no)
) COMPRESSION="zlib";

# Insert data (not shown)

# Query page compression metadata in INFORMATION_SCHEMA.INNODB_TABLESPACES

mysql> SELECT SPACE, NAME, FS_BLOCK_SIZE, FILE_SIZE, ALLOCATED_SIZE FROM
INFORMATION_SCHEMA.INNODB_TABLESPACES WHERE NAME='employees/employees'G
***** 1. row *****
SPACE: 45
NAME: employees/employees
FS_BLOCK_SIZE: 4096
FILE_SIZE: 23068672
ALLOCATED_SIZE: 19415040
```

`employees` テーブルのページ圧縮メタデータは、明らかなファイルサイズが 23068672 バイトで、実際のファイルサイズ (ページ圧縮あり) が 19415040 バイトであることを示しています。ファイルシステムのブロックサイズは 4096 バイトで、ホールパンチに使用されるブロックサイズです。

ページ圧縮を使用したテーブルの識別

ページ圧縮が有効になっているテーブルを識別するには、**COMPRESSION** 属性で定義されているテーブルの `INFORMATION_SCHEMA.TABLES.CREATE_OPTIONS` カラムをクエリーします:

```
mysql> SELECT TABLE_NAME, TABLE_SCHEMA, CREATE_OPTIONS FROM INFORMATION_SCHEMA.TABLES
WHERE CREATE_OPTIONS LIKE '%COMPRESSION=%';
+-----+-----+-----+
| TABLE_NAME | TABLE_SCHEMA | CREATE_OPTIONS |
+-----+-----+-----+
| employees | test | COMPRESSION="zlib" |
+-----+-----+-----+
```

SHOW CREATE TABLE では、**COMPRESSION** 属性も表示されます (使用されている場合)。

ページ圧縮の制限事項と使用上のノート

- ファイルシステムのブロックサイズ (または Windows の場合は圧縮単位サイズ) * 2 > `innodb_page_size` の場合、ページ圧縮は無効になります。
- システムテーブルスペース、一時テーブルスペースおよび一般テーブルスペースを含む共有テーブルスペースに存在するテーブルでは、ページ圧縮はサポートされていません。

- undo ログテーブルスペースではページ圧縮はサポートされていません。
- redo ログページでは、ページ圧縮はサポートされていません。
- 空間インデックスに使用される R ツリーページは圧縮されません。
- 圧縮テーブル (`ROW_FORMAT=COMPRESSED`) に属するページはそのまま残ります。
- リカバリ中、更新されたページは圧縮されていない形式で書き出されます。
- 使用された圧縮アルゴリズムをサポートしていないサーバーにページ圧縮テーブルスペースをロードすると、I/O エラーが発生します。
- ページ圧縮をサポートしていない以前のバージョンの MySQL にダウングレードする前に、ページ圧縮機能を使用するテーブルを解凍します。テーブルを解凍するには、`ALTER TABLE ... COMPRESSION=None` および `OPTIMIZE TABLE` を実行します。
- ページ圧縮されたテーブルスペースは、使用された圧縮アルゴリズムが両方のサーバーで使用可能な場合、Linux サーバーと Windows サーバーの間でコピーできます。
- ページ圧縮されたテーブルスペースファイルがあるホストから別のホストに移動する際にページ圧縮を保持するには、スパーファイルを保持するユーティリティが必要です。
- NVMFS はバンチ穴機能を利用するように設計されているため、NVMFS を搭載した Fusion-io ハードウェアでは、他のプラットフォームよりも優れたページ圧縮を実現できます。
- InnoDB ページサイズが大きく、ファイルシステムのブロックサイズが比較的小さいページ圧縮機能を使用すると、書き込み増幅が発生する可能性があります。たとえば、ファイルシステムのブロックサイズが 4KB の 64KB の最大 InnoDB ページサイズでは、圧縮が改善されますが、バッファプールに対する需要が増加し、I/O および書き込み増幅が増加する可能性があります。

15.10 InnoDB の行フォーマット

テーブルの行形式によって、その行が物理的に格納される方法が決まり、クエリーおよび DML 操作のパフォーマンスに影響を与える可能性があります。単一のディスクページに収まる行数が多いほど、クエリーおよびインデックス検索の動作が速くなり、バッファプールに必要なキャッシュメモリーが少なくなり、更新された値を書き出すために必要な I/O が少なくなります。

各テーブルのデータは複数のページに分かれています。各テーブルを構成するページは、B ツリーインデックスと呼ばれるツリーデータ構造で配置されます。テーブルデータとセカンダリインデックスはどちらも、このタイプの構造を使用します。テーブル全体を表す B ツリーインデックスは、クラスタ化されたインデックスと呼ばれます。これは、主キーカラムに従って編成されます。クラスタ化されたインデックスデータ構造のノードには、行のすべてのカラムの値が含まれます。セカンダリインデックス構造のノードには、インデックスカラムと主キーカラムの値が含まれます。

可変長カラムは、カラム値が B ツリーインデックスノードに格納されるというルールの例外です。長すぎて B ツリーページに収まらない可変長カラムは、オーバーフローページと呼ばれる個別に割り当てられたディスクページに格納されます。このようなカラムは、オフページカラムと呼ばれます。オフページカラムの値は、オーバーフローページの単一リンクリストに格納され、このような各カラムには 1 つ以上のオーバーフローページの独自のリストがあります。カラムの長さに応じて、可変長のカラム値のすべてまたは接頭辞が B ツリーに格納され、記憶域が無駄になり、別のページを読み取る必要がなくなります。

InnoDB ストレージエンジンは 4 つの行フォーマットをサポートしています: `REDUNDANT`, `COMPACT`, `DYNAMIC` および `COMPRESSED`。

表 15.15 InnoDB の行フォーマットの概要

行フォーマット	コンパクトなストレージ特性	可変長カラム記憶域の拡張	大きいインデックスキー接頭辞のサポート	圧縮のサポート	サポートされるテーブルスペースタイプ
<code>REDUNDANT</code>	いいえ	いいえ	いいえ	いいえ	システム、file-per-table、一般

行フォーマット	コンパクトなストレージ特性	可変長カラム記憶域の拡張	大きいインデックスキー接頭辞のサポート	圧縮のサポート	サポートされるテーブルスペースタイプ
COMPACT	はい	いいえ	いいえ	いいえ	システム、file-per-table、一般
DYNAMIC	はい	はい	はい	いいえ	システム、file-per-table、一般
COMPRESSED	はい	はい	はい	はい	file-per-table、general

次のトピックでは、行形式の格納特性と、テーブルの行形式を定義および決定する方法について説明します。

- [REDUNDANT 行形式](#)
- [COMPACT 行フォーマット](#)
- [DYNAMIC 行フォーマット](#)
- [COMPRESSED 行形式](#)
- [テーブルの行形式の定義](#)
- [テーブルの行形式の決定](#)

REDUNDANT 行形式

REDUNDANT 形式は、古いバージョンの MySQL との互換性を提供します。

REDUNDANT 行フォーマットを使用するテーブルでは、可変長カラム値 (VARCHAR、VARBINARY、BLOB および TEXT タイプ) の最初の 768 バイトが B ツリーノード内のインデックスレコードに格納され、残りはオーバーフローページに格納されます。768 バイト以上の固定長カラムは可変長カラムとしてエンコードされ、オフページに格納できます。たとえば、utf8mb4 と同様に、文字セットの最大バイト長が 3 より大きい場合、CHAR(255) カラムは 768 バイトを超えることがあります。

カラムの値が 768 バイト以下の場合、オーバーフローページは使用されず、値は完全に B ツリーノードに格納されるため、I/O である程度節約できます。これは比較的短い BLOB カラム値には適切に機能しますが、B ツリーノードがキー値ではなくデータを埋めるため、効率が低下する可能性があります。BLOB カラムが多数あるテーブルでは、B ツリーノードがいっぱいになりすぎて行が少なすぎるため、行が短い場合やカラム値がオフページに格納された場合よりもインデックス全体の効率が低下する可能性があります。

REDUNDANT 行形式の記憶特性

REDUNDANT の行形式には、次のような記憶特性があります:

- 各インデックスレコードには、6 バイトのヘッダーが含まれています。ヘッダーは、連続するレコードをリンクし、行レベルロックに使用されます。
- クラスタ化されたインデックス内のレコードには、すべてのユーザー定義カラムのフィールドが含まれます。さらに、6 バイトのトランザクション ID フィールドと 7 バイトのロールポインタフィールドも含まれています。
- テーブルに主キーが定義されていない場合は、クラスタ化された各インデックスレコードにも 6 バイトの行 ID フィールドが含まれます。
- 各セカンダリインデックスレコードには、セカンダリインデックスにないクラスタ化されたインデックスキーに定義されたすべての主キーカラムが含まれます。
- レコードには、そのレコードの各フィールドへのポインタが含まれます。レコード内のフィールド長の合計が 128 バイト未満の場合はポインタが 1 バイト、128 バイト以上の場合はポインタが 2 バイトになります。ポインタの配列はレコードディレクトリと呼ばれます。ポインタが指す領域は、レコードのデータ部分です。
- CHAR(10) などの固定長文字カラムは、内部的に固定長形式で格納されます。末尾の空白は、VARCHAR カラムから切り捨てられません。

- 768 バイト以上の固定長カラムは可変長カラムとしてエンコードされ、オフページに格納できます。たとえば、`utf8mb4` と同様に、文字セットの最大バイト長が 3 より大きい場合、`CHAR(255)` カラムは 768 バイトを超えることがあります。
- SQL の `NULL` 値では、レコードディレクトリに 1 バイトまたは 2 バイトが予約されます。SQL `NULL` 値は、可変長カラムに格納されている場合、レコードのデータ部分にゼロバイトを予約します。固定長カラムの場合、カラムの固定長はレコードのデータ部分で予約されます。`NULL` 値の固定領域を予約すると、インデックスページの断片化を発生させることなく、`NULL` から `NULL` 以外の値にカラムをインプレースで更新できます。

COMPACT 行フォーマット

`COMPACT` の行形式では、`REDUNDANT` の行形式と比べて行の記憶領域が約 20% 削減されますが、一部の操作で CPU 使用率が増加します。ワークロードが、キャッシュヒット率とディスク速度によって制限される通常のワークロードであれば、`COMPACT` 形式が高速になる可能性があります。ワークロードが CPU 速度によって制限されている場合、圧縮形式が遅くなる可能性があります。

`COMPACT` 行フォーマットを使用するテーブルでは、可変長カラム値 (`VARCHAR`、`VARBINARY`、`BLOB` および `TEXT` タイプ) の最初の 768 バイトが `B-tree` ノード内のインデックスレコードに格納され、残りはオーバーフローページに格納されます。768 バイト以上の固定長カラムは可変長カラムとしてエンコードされ、オフページに格納できます。たとえば、`utf8mb4` と同様に、文字セットの最大バイト長が 3 より大きい場合、`CHAR(255)` カラムは 768 バイトを超えることがあります。

カラムの値が 768 バイト以下の場合、オーバーフローページは使用されず、値は完全に B ツリーノードに格納されるため、I/O である程度節約できます。これは比較的短い `BLOB` カラム値には適切に機能しますが、B ツリーノードがキー値ではなくデータを埋めるため、効率が低下する可能性があります。`BLOB` カラムが多数あるテーブルでは、B ツリーノードがいっぱいになりすぎて行が少なすぎるため、行が短い場合やカラム値がオフページに格納された場合よりもインデックス全体の効率が低下する可能性があります。

COMPACT 行形式の格納特性

`COMPACT` の行形式には、次のような記憶特性があります：

- 各インデックスレコードには、前に可変長ヘッダーが付く可能性のある 5 バイトのヘッダーが含まれています。ヘッダーは、連続するレコードをリンクし、行レベルロックに使用されます。
- レコードヘッダーの可変長部分には、`NULL` カラムを示すビットベクトルが含まれています。`NULL` にすることができるインデックス内のカラム数が `N` である場合は、ビットベクトルで `CEILING(N/8)` バイトが占有されます。(たとえば、`NULL` にすることができるカラムが 9 から 16 までの任意の数だけ存在する場合は、ビットベクトルで 2 バイトが使用されます。) このベクトル内のビット以外の領域は、`NULL` のカラムで占有されません。ヘッダーの可変長部分には、可変長カラムの長さも含まれています。各長さは、カラムの最大長に応じて、1 バイトと 2 バイトのいずれかになります。インデックス内のすべてのカラムが `NOT NULL` でかつ固定長である場合、レコードヘッダーには可変長部分が含まれません。
- 非 `NULL` 可変長フィールドごとに、レコードヘッダーに 1 バイトまたは 2 バイトのカラム長が含まれます。2 バイトが必要なのは、カラムの一部がオーバーフローページに外部的に格納されている場合、または最大長が 255 バイトを超え、実際の長さが 127 バイトを超える場合のみです。カラムが外部に格納された場合、2 バイトの長さは、内部に格納された部分の長さ、外部に格納された部分への 20 バイトのポインタを加えた長さを示します。内部の部分は 768 バイトであるため、長さは `768+20` になります。20 バイトのポインタには、そのカラムの実際の長さが格納されます。
- レコードヘッダーの後に、`NULL` 以外のカラムのデータコンテンツが続きます。
- クラスタ化されたインデックス内のレコードには、すべてのユーザー定義カラムのフィールドが含まれます。さらに、6 バイトのトランザクション ID フィールドと 7 バイトのロールポインタフィールドも含まれています。
- テーブルに主キーが定義されていない場合は、クラスタ化された各インデックスレコードにも 6 バイトの行 ID フィールドが含まれます。
- 各セカンダリインデックスレコードには、セカンダリインデックスにないクラスタ化されたインデックスキーに定義されたすべての主キーカラムが含まれます。主キーカラムのいずれかが可変長の場合、セカンダリインデックスが固定長カラムに定義されていても、各セカンダリインデックスのレコードヘッダーには長さを記録する可変長部分があります。

- 内部的には、可変長文字セットの場合、[CHAR\(10\)](#) などの固定長文字カラムは固定長形式で格納されます。

末尾の空白は、[VARCHAR](#) カラムから切り捨てられません。

- 内部的に、[utf8mb3](#) や [utf8mb4](#) などの可変長文字セットの場合、[InnoDB](#) は末尾の空白を切り捨てて [CHAR\(N\)](#) を N バイトに格納しようとしています。 [CHAR\(N\)](#) カラム値のバイト長が N バイトを超える場合、後続の空白はカラム値のバイト長の最小値に切り捨てられます。 [CHAR\(N\)](#) カラムの最大長は、最大文字バイト長 $\times N$ です。

N の最小バイト数は [CHAR\(N\)](#) 用に予約されています。多くの場合、 N の最小領域を予約すると、インデックスページの断片化を発生させずにカラムの更新を実行できます。比較すると、[CHAR\(N\)](#) カラムは、[REDUNDANT](#) 行形式を使用している場合、最大文字バイト長の N を占有します。

768 バイト以上の固定長カラムは可変長フィールドとしてエンコードされ、オフページに格納できます。たとえば、[utf8mb4](#) と同様に、文字セットの最大バイト長が 3 より大きい場合、[CHAR\(255\)](#) カラムは 768 バイトを超えることがあります。

DYNAMIC 行フォーマット

[DYNAMIC](#) の行形式では、[COMPACT](#) の行形式と同じ記憶特性が提供されますが、長い可変長カラムの拡張記憶域機能が追加され、大規模なインデックスキー接頭辞がサポートされます。

[ROW_FORMAT=DYNAMIC](#) を使用してテーブルを作成する場合、[InnoDB](#) では、オーバーフローページへの 20 バイトポインタのみを含むクラスタインデックスレコードを使用して、長い可変長のカラム値 ([VARCHAR](#)、[VARBINARY](#)、[BLOB](#) および [TEXT](#) タイプの場合) を完全にオフページに格納できます。768 バイト以上の固定長フィールドは、可変長フィールドとしてエンコードされます。たとえば、[utf8mb4](#) と同様に、文字セットの最大バイト長が 3 より大きい場合、[CHAR\(255\)](#) カラムは 768 バイトを超えることがあります。

カラムがオフページに格納されるかどうかは、ページサイズおよび行の合計サイズによって異なります。行が長すぎる場合、クラスタ化されたインデックスレコードが [B-tree](#) ページに収まるまで、最も長いカラムがオフページ記憶域として選択されます。40 バイト以下の [TEXT](#) および [BLOB](#) カラムは、行に格納されます。

[DYNAMIC](#) の行形式では、([COMPACT](#) および [REDUNDANT](#) 形式の場合と同様に) インデックスノードに行全体を格納する効率が維持されますが、[DYNAMIC](#) の行形式では、B ツリーノードに大量の長いカラムのデータバイトを入力する問題が回避されます。[DYNAMIC](#) の行形式は、長いデータ値の一部がオフページに格納されている場合、通常は値全体をオフページに格納する方が効率的です。[DYNAMIC](#) 形式では、B ツリーノードに短いカラムが残る可能性があるため、特定の行に必要なオーバーフローページの数も最小限に抑えられます。

[DYNAMIC](#) の行形式では、3072 バイトまでのインデックスキー接頭辞がサポートされます。

[DYNAMIC](#) 行形式を使用するテーブルは、システムテーブルスペース、file-per-table テーブルスペースおよび一般テーブルスペースに格納できます。[DYNAMIC](#) テーブルをシステムテーブルスペースに格納するには、[innodb_file_per_table](#) を無効にして通常の [CREATE TABLE](#) ステートメントまたは [ALTER TABLE](#) ステートメントを使用するか、[CREATE TABLE](#) または [ALTER TABLE](#) で [TABLESPACE \[=\] innodb_system](#) テーブルオプションを使用します。[innodb_file_per_table](#) 変数は、一般的なテーブルスペースには適用されず、[TABLESPACE \[=\] innodb_system](#) テーブルオプションを使用して [DYNAMIC](#) テーブルをシステムテーブルスペースに格納する場合にも適用されません。

DYNAMIC 行フォーマットの格納特性

[DYNAMIC](#) の行フォーマットは、[COMPACT](#) の行フォーマットのバリエーションです。記憶特性については、[COMPACT 行形式の格納特性](#) を参照してください。

COMPRESSED 行形式

[COMPRESSED](#) の行形式は、[DYNAMIC](#) の行形式と同じ記憶特性および機能を提供しますが、テーブルおよびインデックスのデータ圧縮のサポートが追加されています。

[COMPRESSED](#) 行フォーマットは、オフページストレージに関して [DYNAMIC](#) 行フォーマットと同様の内部の詳細を使用するほか、追加のストレージ、圧縮されるテーブルおよびインデックスデータからのパフォーマンスの考慮事項、および小さいページサイズを使用します。[COMPRESSED](#) の行形式では、[KEY_BLOCK_SIZE](#) オプションによ

て、クラスタインデックスに格納されるカラムデータの量およびオーバーフローページに配置される量が制御されます。[COMPRESSED](#) の行形式の詳細は、[セクション15.9「InnoDB のテーブルおよびページの圧縮」](#)を参照してください。

[COMPRESSED](#) の行形式では、3072 バイトまでのインデックスキー接頭辞がサポートされます。

[COMPRESSED](#) 行形式を使用するテーブルは、file-per-table テーブルスペースまたは一般テーブルスペースに作成できます。システムテーブルスペースは、[COMPRESSED](#) の行形式をサポートしていません。file-per-table テーブルスペースに [COMPRESSED](#) テーブルを格納するには、[innodb_file_per_table](#) 変数を有効にする必要があります。[innodb_file_per_table](#) 変数は、一般的なテーブルスペースには適用できません。一般テーブルスペースでは、物理ページサイズが異なるために圧縮テーブルと非圧縮テーブルを同じ一般テーブルスペースに共存できないという注意事項を含むすべての行形式がサポートされています。詳細は、[セクション15.6.3.3「一般テーブルスペース」](#)を参照してください。

圧縮された行形式の格納特性

[COMPRESSED](#) の行フォーマットは、[COMPACT](#) の行フォーマットのバリエーションです。記憶特性については、[COMPACT 行形式の格納特性](#)を参照してください。

テーブルの行形式の定義

InnoDB テーブルのデフォルトの行形式は、[DYNAMIC](#) のデフォルト値を持つ [innodb_default_row_format](#) 変数によって定義されます。デフォルトの行フォーマットは、[ROW_FORMAT](#) テーブルオプションが明示的に定義されていない場合、または [ROW_FORMAT=DEFAULT](#) が指定されている場合に使用されます。

テーブルの行形式は、[CREATE TABLE](#) ステートメントまたは [ALTER TABLE](#) ステートメントの [ROW_FORMAT](#) テーブルオプションを使用して明示的に定義できます。例:

```
CREATE TABLE t1 (c1 INT) ROW_FORMAT=DYNAMIC;
```

明示的に定義された [ROW_FORMAT](#) 設定は、デフォルトの行フォーマットをオーバーライドします。[ROW_FORMAT=DEFAULT](#) を指定することは、暗黙のデフォルトを使用することと同じです。

[innodb_default_row_format](#) 変数は動的に設定できます:

```
mysql> SET GLOBAL innodb_default_row_format=DYNAMIC;
```

有効な [innodb_default_row_format](#) オプションには、[DYNAMIC](#)、[COMPACT](#) および [REDUNDANT](#) があります。システムテーブルスペースでの使用がサポートされていない [COMPRESSED](#) 行形式は、デフォルトとして定義できません。これは、[CREATE TABLE](#) ステートメントまたは [ALTER TABLE](#) ステートメントでのみ明示的に指定できます。[innodb_default_row_format](#) 変数を [COMPRESSED](#) に設定しようとすると、エラーが返されます:

```
mysql> SET GLOBAL innodb_default_row_format=COMPRESSED;
ERROR 1231 (42000): Variable 'innodb_default_row_format'
can't be set to the value of 'COMPRESSED'
```

新しく作成されたテーブルでは、[ROW_FORMAT](#) オプションが明示的に指定されていない場合、または [ROW_FORMAT=DEFAULT](#) が使用されている場合に、[innodb_default_row_format](#) 変数で定義された行形式が使用されます。たとえば、次の [CREATE TABLE](#) ステートメントでは、[innodb_default_row_format](#) 変数で定義された行形式が使用されます。

```
CREATE TABLE t1 (c1 INT);
```

```
CREATE TABLE t2 (c1 INT) ROW_FORMAT=DEFAULT;
```

[ROW_FORMAT](#) オプションが明示的に指定されていない場合、または [ROW_FORMAT=DEFAULT](#) が使用されている場合、テーブルを再構築する操作によって、テーブルの行形式が [innodb_default_row_format](#) 変数で定義された形式に暗黙的に変更されます。

テーブルの再構築操作には、テーブルの再構築が必要な [ALGORITHM=COPY](#) または [ALGORITHM=INPLACE](#) を使用する [ALTER TABLE](#) 操作が含まれます。詳しくは[セクション15.12.1「オンライン DDL 操作」](#)をご覧ください。[OPTIMIZE TABLE](#) は、テーブルの再構築操作でもあります。

次の例は、明示的に定義された行形式なしで作成されたテーブルの行形式を暗黙的に変更するテーブル再構築操作を示しています。

```
mysql> SELECT @@innodb_default_row_format;
+-----+
| @@innodb_default_row_format |
+-----+
| dynamic |
+-----+

mysql> CREATE TABLE t1 (c1 INT);

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TABLES WHERE NAME LIKE 'test/t1' \G
***** 1. row *****
TABLE_ID: 54
NAME: test/t1
FLAG: 33
N_COLS: 4
SPACE: 35
ROW_FORMAT: Dynamic
ZIP_PAGE_SIZE: 0
SPACE_TYPE: Single

mysql> SET GLOBAL innodb_default_row_format=COMPACT;

mysql> ALTER TABLE t1 ADD COLUMN (c2 INT);

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TABLES WHERE NAME LIKE 'test/t1' \G
***** 1. row *****
TABLE_ID: 55
NAME: test/t1
FLAG: 1
N_COLS: 5
SPACE: 36
ROW_FORMAT: Compact
ZIP_PAGE_SIZE: 0
SPACE_TYPE: Single
```

既存のテーブルの行形式を **REDUNDANT** または **COMPACT** から **DYNAMIC** に変更する前に、次の潜在的な問題を考慮してください。

- **REDUNDANT** および **COMPACT** の行形式では 767 バイトのインデックスキー接頭辞の最大長がサポートされますが、**DYNAMIC** および **COMPRESSED** の行形式では 3072 バイトのインデックスキー接頭辞の長さがサポートされます。レプリケーション環境では、ソースで `innodb_default_row_format` 変数が **DYNAMIC** に設定され、レプリカで **COMPACT** に設定されている場合、行形式を明示的に定義しない次の DDL ステートメントはソースで成功しますが、レプリカでは失敗します:

```
CREATE TABLE t1 (c1 INT PRIMARY KEY, c2 VARCHAR(5000), KEY i1(c2(3070)));
```

関連情報については、[セクション 15.22 「InnoDB の制限」](#) を参照してください。

- 行フォーマットを明示的に定義しないテーブルをインポートすると、ソースサーバーの `innodb_default_row_format` 設定が宛先サーバーの設定と異なる場合にスキーマの不一致エラーが発生します。詳細は、[セクション 15.6.1.3 「InnoDB テーブルのインポート」](#) を参照してください。

テーブルの行形式の決定

テーブルの行形式を確認するには、`SHOW TABLE STATUS` を使用します:

```
mysql> SHOW TABLE STATUS IN test1\G
***** 1. row *****
Name: t1
Engine: InnoDB
Version: 10
Row_format: Dynamic
Rows: 0
Avg_row_length: 0
Data_length: 16384
Max_data_length: 0
Index_length: 16384
```



```
Data_free: 0
Auto_increment: 1
Create_time: 2016-09-14 16:29:38
Update_time: NULL
Check_time: NULL
Collation: utf8mb4_0900_ai_ci
Checksum: NULL
Create_options:
Comment:
```

または、`INFORMATION_SCHEMA.INNOODB_TABLES` テーブルをクエリーします:

```
mysql> SELECT NAME, ROW_FORMAT FROM INFORMATION_SCHEMA.INNOODB_TABLES WHERE NAME='test1/t1';
+-----+-----+
| NAME   | ROW_FORMAT |
+-----+-----+
| test1/t1 | Dynamic   |
+-----+-----+
```

15.11 InnoDB のディスク I/O とファイル領域管理

DBA は、I/O サブシステムが飽和状態にならないようにディスク I/O を管理するとともに、ストレージデバイスがいっぱいにならないようにディスク領域を管理する必要があります。ACID 設計モデルには、冗長に見える可能性はあっても、データの信頼性の確保に役立つ、ある一定の量の I/O が必要です。これらの制約の中で、InnoDB は、ディスク I/O の量を最小限に抑えるためにデータベースの動作やディスクファイルの編成を最適化しようとします。場合によっては、I/O は、データベースがビジー状態でなくなるまで、または高速シャットダウンのあとのデータベースの再起動中など、すべてを一貫性のある状態に移行することが必要になるまで延期されます。

このセクションでは、デフォルトの種類の MySQL テーブル (InnoDB テーブルとも呼ばれます) での I/O とディスク領域に関する主な考慮事項について説明します。

- クエリーパフォーマンスを向上させるために使用されるバックグラウンド I/O の量の制御。
- 追加の I/O を削減する代わりに耐久性を向上させる機能の有効化または無効化。
- テーブルの、多数の小さなファイル、いくつかのより大きなファイル、またはその両方の組み合わせへの編成。
- Redo ログファイルのサイズと、ログファイルがいっぱいになったときに発生する I/O アクティビティーとのバランス。
- テーブルを最適なクエリーパフォーマンスのために再編成する方法。

15.11.1 InnoDB ディスク I/O

InnoDB は、可能であれば、I/O 操作を処理するための複数のスレッドを作成することによって非同期ディスク I/O を使用します。それにより、その I/O がまだ進行中の間もほかのデータベース操作を続行できるようにします。Linux および Windows プラットフォームでは、InnoDB は使用可能な OS およびライブラリ関数を使用して「native」非同期 I/O を実行します。他のプラットフォームでは、InnoDB は引き続き I/O スレッドを使用しますが、スレッドは実際には I/O リクエストの完了を待機する場合があります。この手法は「シミュレーション」非同期 I/O と呼ばれます。

先読み

InnoDB は、データがすぐに必要になる可能性が高いと判断できる場合、先読み操作を実行して、そのデータをメモリー内で使用できるようにバッファプールに取り込みます。連続したデータに対しては、いくつかの大きな読み取り要求を作成する方が、複数の拡散した小さな要求を作成するより効率的である場合があります。InnoDB には、2 つの先読みヒューリスティックがあります:

- シーケンシャル先読みでは、テーブルスペース内のセグメントへのアクセスパターンがシーケンシャルであることに気付くと、InnoDB はデータベースページの読み取りのバッチを I/O システムにあらかじめ送信します。
- ランダム先読みでは、テーブルスペース内の一部の領域がバッファプールに完全に読み取られている最中であることに気付くと、InnoDB は残りの読み取りを I/O システムに送信します。

先読みヒューリスティックの構成の詳細は、[セクション15.8.3.4「InnoDB バッファープールのプリフェッチ \(先読み\)の構成」](#)を参照してください。

二重書き込みバッファ

InnoDB では、[doublewrite buffer](#) と呼ばれる構造を含む新しいファイルフラッシュ技術を使用します。これは、ほとんどの場合 ([innodb_doublewrite=ON](#)) でデフォルトで有効になっています。これにより、予期しない終了または停電後のリカバリの安全性が向上し、[fsync\(\)](#) 操作の必要性が減少するため、ほとんどの種類の Unix のパフォーマンスが向上します。

データファイルにページを書き込む前に、InnoDB はまず二重書き込みバッファと呼ばれる記憶域にページを書き込みます。二重書き込みバッファへの書き込みとフラッシュが完了したあとはじめて、InnoDB はそれらのページをデータファイル内の適切な位置に書き込みます。ページ書き込み中にオペレーティングシステム、ストレージサブシステムまたは予期しない [mysqld](#) プロセスが終了した場合 ([torn page](#) 条件の原因)、InnoDB は後でリカバリ中に二重書き込みバッファからページの適切なコピーを見つけることができます。

二重書き込みバッファの詳細は、[セクション15.6.4「二重書き込みバッファ」](#)を参照してください。

15.11.2 ファイル領域管理

[innodb_data_file_path](#) 構成オプションを使用して構成ファイルに定義するデータファイルは、[InnoDB system tablespace](#) を形成します。ファイルは論理的に連結され、システムテーブルスペースを形成します。ストライピングは使用されません。システムテーブルスペース内のどこにテーブルを割り当てるかは定義できません。新しく作成されたシステムテーブルスペースでは、InnoDB は最初のデータファイルから始まる領域を割り当てます。

システムテーブルスペース内にすべてのテーブルおよびインデックスを格納する際の問題を回避するために、[innodb_file_per_table](#) 構成オプション (デフォルト) を有効にして、新しく作成された各テーブルを個別のテーブルスペースファイル (拡張子 [.ibd](#)) に格納できます。この方法で格納されたテーブルの場合、ディスクファイル内の断片化は減少し、テーブルが切り捨てられると、その領域は InnoDB によって引き続きシステムテーブルスペース内に予約されるのではなく、オペレーティングシステムに返されます。詳細は、[セクション15.6.3.2「File-Per-Table テーブルスペース」](#)を参照してください。

[general tablespaces](#) にテーブルを格納することもできます。一般テーブルスペースは、[CREATE TABLESPACE](#) 構文を使用して作成される共有テーブルスペースです。これらは MySQL データディレクトリの外部で作成でき、複数のテーブルを保持でき、すべての行形式のテーブルをサポートします。詳細は、[セクション15.6.3.3「一般テーブルスペース」](#)を参照してください。

ページ、エクステント、セグメント、およびテーブルスペース

各テーブルスペースは、データベースページで構成されます。MySQL インスタンス内のテーブルスペースはすべて、同じ[ページサイズ](#)を持っています。デフォルトでは、すべてのテーブルスペースが 16K バイトのページサイズを持っています。このページサイズを 8K バイトまたは 4K バイトに減らすには、MySQL インスタンスを作成するときに [innodb_page_size](#) オプションを指定します。ページサイズを 32KB または 64KB に増やすこともできます。詳細は、[innodb_page_size](#) のドキュメントを参照してください。

ページは、最大 16K バイトのサイズ (16K バイトの連続した 64 ページ、128 8K バイトのページ、または 256 4K バイトのページ) でサイズ 1M バイトの [extents](#) にグループ化されます。32KB のページサイズの場合、エクステントサイズは 2MB です。64KB のページサイズの場合、エクステントサイズは 4MB です。InnoDB では、テーブルスペース内部の「ファイル」を[セグメント](#)と呼びます。(これらのセグメントは、実際に多数のテーブルスペースセグメントが含まれている[ロールバックセグメント](#)とは異なります。)

セグメントがテーブルスペース内部で拡張される場合、InnoDB は、そのセグメントに最初の 32 ページを一度に割り当てます。そのあと、InnoDB は、そのセグメントへのすべてのエクステントの割り当てを開始します。InnoDB は、データの良好な連続性を保証するために、大きなセグメントには 1 回につき最大 4 つのエクステントを追加できます。

InnoDB では、各インデックスに 2 つのセグメントが割り当てられます。一方は [B-tree](#) の非リーフノード用で、もう一方はリーフノード用です。リーフノードをディスク上で連続した状態に維持すると、これらのリーフノードには実際のテーブルデータが含まれているため、シーケンシャル I/O 操作の性能が向上します。

テーブルスペース内の一部のページにはほかのページのビットマップが含まれているため、InnoDB テーブルスペース内のいくつかのエクステントは全体としてではなく、個々のページとしてのみセグメントに割り当てることができません。

SHOW TABLE STATUS ステートメントを発行することによってテーブルスペース内の使用可能な空き領域を求めると、InnoDB は、テーブルスペース内の確実に空いているエクステントをレポートします。InnoDB は、常にいくつかのエクステントをクリーンアップやその他の内部の目的のために予約します。これらの予約されたエクステントは空き領域に含まれません。

テーブルからデータを削除すると、InnoDB は、対応する B ツリーインデックスを短くします。解放された領域をほかのユーザーが使用できるようになるかどうかは、削除のパターンがテーブルスペースに対して個々のページまたはエクステントのどちらを解放するかによって異なります。テーブルを削除したりテーブルのすべての行を削除したりすると、その領域は確実にほかのユーザーに解放されますが、それらの削除された行は、その行がトランザクションロールバックまたは一貫性読み取りに必要なくなったあと、しばらくして自動的に発生する **ページ** 操作によってのみ物理的に削除されることに注意してください。(セクション 15.3 「InnoDB マルチバージョン」を参照してください。)

ページのテーブル行への関連付け

行の最大長は、4KB、8KB、16KB および 32KB の `innodb_page_size` 設定のデータベースページの半分未満です。たとえば、デフォルトの 16KB の InnoDB ページサイズでは、行の最大長は 8KB 未満です。64KB ページの場合、行の最大長は 16KB 未満です。

行が最大行長を超えない場合、すべての行はページ内にローカルに格納されます。行が最大行長を超えると、行が最大行長制限内に収まるまで、外部オフページストレージ用に `variable-length columns` が選択されます。可変長カラムの外部オフページストレージは、行形式によって異なります:

- COMPACT および REDUNDANT 行フォーマット

可変長カラムが外部オフページストレージに選択されると、InnoDB では最初の 768 バイトが行にローカルに格納され、残りはオーバーフローページに外部的に格納されます。このような各カラムには、オーバーフローページの独自のリストがあります。768 バイトのプリフィクスには、そのカラムの実際の長さを格納し、値の残りの部分が格納されているオーバーフローページリストを指す 20 バイトの値が付随します。セクション 15.10 「InnoDB の行フォーマット」を参照してください。

- DYNAMIC および COMPRESSED 行フォーマット

可変長カラムが外部オフページストレージに選択されると、InnoDB では 20 バイトのポインタが行にローカルに格納され、残りはオーバーフローページに外部的に格納されます。セクション 15.10 「InnoDB の行フォーマット」を参照してください。

LONGBLOB および **LONGTEXT** カラムは 4G バイト未満である必要があり、**BLOB** および **TEXT** カラムを含む行全体の長さは 4G バイト未満である必要があります。

15.11.3 InnoDB チェックポイント

ログファイル を非常に大きくすると、**チェックポイント設定** 中のディスク I/O が少なくなる可能性があります。ログファイルの合計サイズは多くの場合、バッファプールと同じか、またはそれより大きい設定が適切です。

チェックポイント処理の動作のしくみ

InnoDB は、**ファジーチェックポイント設定** と呼ばれる **チェックポイント** メカニズムを実装しています。InnoDB は、変更されたデータベースページをバッファプールから小さなバッチにフラッシュします。バッファプールを 1 つのバッチにフラッシュする必要はありません。それを行うと、チェックポイント設定プロセス中にユーザーの SQL ステートメントの処理が中断されます。

クラッシュリカバリ 中に、InnoDB は、ログファイルに書き込まれたチェックポイントラベルを探します。それは、そのラベルの前にあるデータベースへのすべての変更がデータベースのディスクイメージ内に存在することを知っています。次に、InnoDB はそのチェックポイントから前方にログファイルをスキャンしながら、ログに記録された変更をデータベースに適用します。

15.11.4 テーブルのデフラグ

セカンダリインデックスへのランダムな挿入やセカンダリインデックスからのランダムな削除によって、インデックスが断片化される場合があります。断片化とは、ディスク上のインデックスページの物理的な順序がページ上のレコードのインデックス順序とかけ離れているか、またはインデックスに割り当てられた 64 ページのブロック内に未使用のページが多数存在することを示します。

断片化の 1 つの現象として、あるテーブルが占めている領域が、本来占めている「はずの」領域より大きいことがあります。それが正確にどの程度かを判定するのは困難です。すべての InnoDB データおよびインデックスは B-trees に格納され、その fill factor は 50% から 100% まで異なる場合があります。断片化の別の現象として、次のようなテーブルスキャンにかかる時間が、本来かかる「はずの」時間より長いことがあります。

```
SELECT COUNT(*) FROM t WHERE non_indexed_column <> 12345;
```

前のクエリーでは、MySQL が、大きなテーブルに対してもっとも遅いタイプのクエリーであるフルテーブルスキャンを実行する必要があります。

インデックススキャンを高速化するために、MySQL にテーブルを再構築させる次の「null」ALTER TABLE 操作を定期的に行うことができます。

```
ALTER TABLE tbl_name ENGINE=INNODB
```

ALTER TABLE tbl_name FORCE を使用して、テーブルを再構築する「null」変更操作を実行することもできます。

ALTER TABLE tbl_name ENGINE=INNODB と ALTER TABLE tbl_name FORCE はどちらも online DDL を使用します。詳細は、セクション 15.12 「InnoDB とオンライン DDL」を参照してください。

デフラグ操作を実行するための別の方法として、mysqldump を使用してテーブルをテキストファイルにダンプし、テーブルを削除してから、それをダンプファイルからリロードする方法があります。

インデックスへの挿入が常に昇順であり、かつレコードが末尾からしか削除されない場合は、InnoDB のファイル領域管理アルゴリズムにより、インデックス内の断片化は発生しないことが保証されます。

15.11.5 TRUNCATE TABLE によるディスク領域の再利用

InnoDB テーブルを切り捨てるときにオペレーティングシステムのディスク領域を再利用するには、そのテーブルが独自の .ibd ファイルに格納されている必要があります。独自の .ibd ファイルに格納されるテーブルの場合は、そのテーブルを作成するときに innodb_file_per_table を有効にする必要があります。さらに、切り捨てられるテーブルとその他のテーブルの間に外部キー制約が存在してはいけません。そうしないと、TRUNCATE TABLE 操作は失敗します。ただし、同じテーブル内の 2 つのカラム間の外部キー制約は許可されます。

テーブルが切り捨てられると、そのテーブルが削除されて新しい .ibd ファイル内に再作成され、解放された領域はオペレーティングシステムに返されます。これは、InnoDB system tablespace 内に格納されている InnoDB テーブル (innodb_file_per_table=OFF 時に作成されたテーブル) および共有 general tablespaces に格納されているテーブルを切り捨てることとは対照的で、InnoDB のみがテーブルの切捨て後に解放された領域を使用できます。

テーブルを切り捨て、そのディスク領域をオペレーティングシステムに返す機能はまた、物理バックアップを小さくすることもできます。システムテーブルスペース (innodb_file_per_table=OFF で作成されたテーブル) または一般テーブルスペースに格納されているテーブルを切り捨てると、テーブルスペース内に未使用領域のブロックが残されます。

15.12 InnoDB とオンライン DDL

オンライン DDL 機能では、即時およびインプレースのテーブル変更および同時 DML がサポートされます。この機能の利点は次のとおりです：

- ビジーな本番環境での応答性と可用性が向上し、テーブルを数分間または数時間使用できなくなります。
- インプレース操作の場合、LOCK 句を使用して DDL 操作中のパフォーマンスと同時実行性のバランスを調整する機能。LOCK 句を参照してください。
- テーブルコピー方法よりもディスク領域の使用量と I/O のオーバーヘッドが少なくなります。

注記

`ALGORITHM=INSTANT` のサポートは、`ADD COLUMN` および MySQL 8.0.12 のその他の操作で使用できます。

通常、オンライン DDL を有効にするために特別な操作を行う必要はありません。デフォルトでは、MySQL は操作を許可されているとおりに即時またはインプレースで実行しますが、ロックはできるかぎり少なくなります。

`ALTER TABLE` ステートメントの `ALGORITHM` 句および `LOCK` 句を使用して、DDL 操作の側面を制御できます。これらの句は、テーブルおよびカラムの指定からカンマで区切ってステートメントの最後に配置されます。例:

```
ALTER TABLE tbl_name ADD PRIMARY KEY (column), ALGORITHM=INPLACE, LOCK=NONE;
```

`LOCK` 句は、インプレースで実行される操作に使用でき、操作中のテーブルへの同時アクセスの程度を微調整する場に役立ちます。即時に実行される操作では、`LOCK=DEFAULT` のみがサポートされます。`ALGORITHM` 句は、主にパフォーマンスの比較と、問題が発生した場合の古いテーブルコピー動作へのフォールバックを目的としています。例:

- インプレースの `ALTER TABLE` 操作中に、誤ってテーブルを読み取りまたは書き込み (あるいはその両方) に使用できないようにするには、`LOCK=NONE` (読み取りおよび書き込みの許可) や `LOCK=SHARED` (読み取りの許可) などの句を `ALTER TABLE` ステートメントに指定します。要求されたレベルの並列性が使用できない場合、操作はただちに停止します。
- アルゴリズム間でパフォーマンスを比較するには、`ALGORITHM=INSTANT`、`ALGORITHM=INPLACE` および `ALGORITHM=COPY` でステートメントを実行します。`old_alter_table` 構成オプションを有効にしてステートメントを実行し、`ALGORITHM=COPY` を強制的に使用することもできます。
- テーブルをコピーする `ALTER TABLE` 操作でサーバーがタイアップされないようにするには、`ALGORITHM=INSTANT` または `ALGORITHM=INPLACE` を含めます。指定されたアルゴリズムを使用できない場合、ステートメントはただちに停止します。

15.12.1 オンライン DDL 操作

DDL 操作のオンラインサポートの詳細、構文例および使用上のノートは、このセクションの次のトピックで説明します。

- [インデックス操作](#)
- [主キーの操作](#)
- [カラム操作](#)
- [生成されたカラム操作](#)
- [外部キー操作](#)
- [テーブルの操作](#)
- [テーブルスペースの操作](#)
- [パーティション化操作](#)

インデックス操作

次のテーブルに、インデックス操作のオンライン DDL サポートの概要を示します。アスタリスクは、追加情報、例外または依存関係を示します。詳細は、[構文および使用上のノート](#)を参照してください。

表 15.16 インデックス操作のオンライン DDL サポート

操作	インスタント	インプレース	テーブルの再構築	同時 DML の許可	メタデータの変更のみ
セカンダリインデックスの作成または追加	いいえ	はい	いいえ	はい	いいえ

操作	インスタント	インプレース	テーブルの再構築	同時 DML の許可	メタデータの変更のみ
インデックスの削除	いいえ	はい	いいえ	はい	はい
インデックスの名前変更	いいえ	はい	いいえ	はい	はい
FULLTEXT インデックスの追加	いいえ	はい*	いいえ*	いいえ	いいえ
SPATIAL インデックスの追加	いいえ	はい	いいえ	いいえ	いいえ
インデックスタイプの変更	はい	はい	いいえ	はい	はい

構文および使用上のノート

- セカンダリインデックスの作成または追加

```
CREATE INDEX name ON table (col_list);
```

```
ALTER TABLE tbl_name ADD INDEX name (col_list);
```

このテーブルは、インデックスの作成中も読取りおよび書込み操作に使用できます。 **CREATE INDEX** ステートメントは、テーブルにアクセスしているすべてのトランザクションが完了した後にのみ終了するため、インデックスの初期状態にはテーブルの最新の内容が反映されます。

セカンダリインデックスを追加するためのオンライン DDL サポートとは、通常、セカンダリインデックスのないテーブルを作成してからデータのロード後にセカンダリインデックスを追加することで、テーブルおよび関連するインデックスの作成およびロードのプロセス全体を高速化できることを意味します。

新しく作成されたセカンダリインデックスには、**CREATE INDEX** または **ALTER TABLE** ステートメントの実行が終了した時点でテーブルにコミットされたデータのみが含まれます。コミットされていない値や古いバージョンの値、または削除対象としてマークされているが、まだ古いインデックスから削除されていない値は含まれていません。

この操作のパフォーマンス、領域使用量およびセマンティクスに影響する要因もあります。詳細は、[セクション 15.12.6 「オンライン DDL の制限事項」](#) を参照してください。

- インデックスの削除

```
DROP INDEX name ON table;
```

```
ALTER TABLE tbl_name DROP INDEX name;
```

このテーブルは、インデックスの削除中も読取りおよび書込み操作に使用できます。 **DROP INDEX** ステートメントは、テーブルにアクセスしているすべてのトランザクションが完了した後にのみ終了するため、インデックスの初期状態にはテーブルの最新の内容が反映されます。

- インデックスの名前変更

```
ALTER TABLE tbl_name RENAME INDEX old_index_name TO new_index_name, ALGORITHM=INPLACE, LOCK=NONE;
```

- FULLTEXT** インデックスの追加

```
CREATE FULLTEXT INDEX name ON table(column);
```

ユーザー定義の **FTS_DOC_ID** カラムがない場合は、最初の **FULLTEXT** インデックスを追加するとテーブルが再構築されます。テーブルを再構築せずに、**FULLTEXT** インデックスを追加できます。

- SPATIAL** インデックスの追加

```
CREATE TABLE geom (g GEOMETRY NOT NULL);
ALTER TABLE geom ADD SPATIAL INDEX(g), ALGORITHM=INPLACE, LOCK=SHARED;
```


- インデックスタイプの変更 (USING {BTREE | HASH})

```
ALTER TABLE tbl_name DROP INDEX i1, ADD INDEX i1(key_part,...) USING BTREE, ALGORITHM=INSTANT;
```

主キーの操作

次のテーブルに、主キー操作のオンライン DDL サポートの概要を示します。アスタリスクは、追加情報、例外または依存関係を示します。 [構文および使用上のノート](#) を参照してください。

表 15.17 主キー操作のオンライン DDL サポート

操作	インスタント	インプレース	テーブルの再構築	同時 DML の許可	メタデータの変更のみ
主キーの追加	いいえ	はい*	はい*	はい	いいえ
主キーの削除	いいえ	いいえ	はい	いいえ	いいえ
主キーの削除および別の主キーの追加	いいえ	はい	はい	はい	いいえ

構文および使用上のノート

- 主キーの追加

```
ALTER TABLE tbl_name ADD PRIMARY KEY (column), ALGORITHM=INPLACE, LOCK=NONE;
```

テーブルを適切に再構築します。データは大幅に再編成され、コストのかかる操作になります。カラムを **NOT NULL** に変換する必要がある場合、特定の条件下で **ALGORITHM=INPLACE** は許可されません。

clustered index を再構築するには、常にテーブルデータのコピーが必要です。したがって、後で **ALTER TABLE ... ADD PRIMARY KEY** を発行するのではなく、テーブルの作成時に **primary key** を定義することをお勧めします。

UNIQUE または **PRIMARY KEY** インデックスを作成したとき、MySQL は、いくつかの追加の作業を行う必要があります。**UNIQUE** インデックスの場合、MySQL は、テーブルに重複したキーの値が含まれていないことをチェックします。**PRIMARY KEY** インデックスの場合も、MySQL は、どの **PRIMARY KEY** カラムにも **NULL** が含まれていないことをチェックします。

ALGORITHM=COPY 句を使用して主キーを追加すると、MySQL は関連付けられたカラムの **NULL** 値をデフォルト値に変換: 数値の場合は 0、文字ベースのカラムおよび BLOB の場合は空の文字列、**DATETIME** の場合は 0000-00-00 00:00:00。これは非標準の動作であるため、これに依存しないようにすることをお勧めします。**ALGORITHM=INPLACE** を使用した主キーの追加は、**SQL_MODE** 設定に **strict_trans_tables** または **strict_all_tables** フラグが含まれている場合のみ許可されます。**SQL_MODE** 設定が厳密な場合、**ALGORITHM=INPLACE** は許可されますが、リクエストされた主キーカラムに **NULL** 値が含まれている場合、ステートメントは失敗する可能性があります。**ALGORITHM=INPLACE** の動作は、より標準に準拠しています。

主キーなしでテーブルを作成すると、**InnoDB** によってテーブルが選択されます。これは、**NOT NULL** カラムに定義されている最初の **UNIQUE** キーまたはシステム生成キーです。余分な非表示カラムの不確実性および潜在的な領域要件を回避するには、**CREATE TABLE** ステートメントの一部として **PRIMARY KEY** 句を指定します。

MySQL では、既存のデータを元のテーブルから目的のインデックス構造を持つ一時テーブルにコピーすることで、新しいクラスタインデックスが作成されます。データが一時テーブルに完全にコピーされると、元のテーブルの名前は別の一時テーブル名に変更されます。新しいクラスタ化されたインデックスで構成される一時テーブルの名前が元のテーブルの名前に変更され、元のテーブルはデータベースから削除されます。

セカンダリインデックスでの操作に適用されるオンラインパフォーマンスの拡張は、主キーインデックスには適用されません。**InnoDB** テーブルの行は、主キーに基づいて編成された **クラスタ化されたインデックス** に格納されます。これにより、一部のデータベースシステムで「インデックス編成テーブル」と呼ばれるものが形成されます。テーブル構造は主キーに密接に関連付けられているため、主キーを再定義するには引き続きデータをコピーする必要があります。

主キーに対する操作で **ALGORITHM=INPLACE** が使用される場合は、データが引き続きコピーされるにもかかわらず、次の理由で **ALGORITHM=COPY** を使用するより効率的です。

- `ALGORITHM=INPLACE` には、Undo ロギングやそれに関連する Redo ロギングが必要ありません。これらの操作は、`ALGORITHM=COPY` を使用する DDL ステートメントのオーバーヘッドを増やします。
- セカンダリインデックスエントリは事前にソートされているため、順番にロードできます。
- セカンダリインデックスへのランダムアクセス挿入は存在しないため、変更バッファは使用されません。
- 主キーの削除

```
ALTER TABLE tbl_name DROP PRIMARY KEY, ALGORITHM=COPY;
```

同じ `ALTER TABLE` ステートメントに新しい主キーを追加せずに主キーを削除できるのは、`ALGORITHM=COPY` のみです。

- 主キーの削除および別の主キーの追加

```
ALTER TABLE tbl_name DROP PRIMARY KEY, ADD PRIMARY KEY (column), ALGORITHM=INPLACE, LOCK=NONE;
```

データは大幅に再編成され、コストのかかる操作になります。

カラム操作

次のテーブルに、カラム操作のオンライン DDL サポートの概要を示します。アスタリスクは、追加情報、例外または依存関係を示します。詳細は、[構文および使用上のノート](#)を参照してください。

表 15.18 カラム操作のオンライン DDL サポート

操作	インスタント	インプレース	テーブルの再構築	同時 DML の許可	メタデータの変更のみ
カラムの追加	はい*	はい	いいえ*	はい*	いいえ
カラムの削除	いいえ	はい	はい	はい	いいえ
カラム名の変更	いいえ	はい	いいえ	はい*	はい
カラムの並替え	いいえ	はい	はい	はい	いいえ
カラムのデフォルト値の設定	はい	はい	いいえ	はい	はい
カラムのデータ型の変更	いいえ	いいえ	はい	いいえ	いいえ
<code>VARCHAR</code> カラムサイズの拡張	いいえ	はい	いいえ	はい	はい
カラムのデフォルト値の削除	はい	はい	いいえ	はい	はい
自動インクリメント値の変更	いいえ	はい	いいえ	はい	いいえ*
カラムの <code>NULL</code> 化	いいえ	はい	はい*	はい	いいえ
カラムの <code>NOT NULL</code> 化	いいえ	はい*	はい*	はい	いいえ
<code>ENUM</code> または <code>SET</code> カラムの定義の変更	はい	はい	いいえ	はい	はい

構文および使用上のノート

- カラムの追加

```
ALTER TABLE tbl_name ADD COLUMN column_name column_definition, ALGORITHM=INSTANT;
```

`INSTANT` アルゴリズムを使用してカラムを追加する場合は、次の制限が適用されます:

- カラムの追加は、`ALGORITHM=INSTANT` をサポートしない他の `ALTER TABLE` アクションと同じステートメントで組み合わせることはできません。
- カラムは、テーブルの最後のカラムとしてのみ追加できます。他のカラム間の他の位置へのカラムの追加はサポートされていません。
- `ROW_FORMAT=COMPRESSED` を使用するテーブルにはカラムを追加できません。
- `FULLTEXT` インデックスを含むテーブルにはカラムを追加できません。
- カラムは一時テーブルに追加できません。一時テーブルでは、`ALGORITHM=COPY` のみがサポートされます。
- データディクショナリテーブルスペースに存在するテーブルにはカラムを追加できません。
- 行サイズ制限は、カラムの追加時には評価されません。ただし、行サイズ制限は、テーブルの行を挿入および更新する DML 操作中にチェックされます。

同じ `ALTER TABLE` ステートメントに複数のカラムを追加できます。例:

```
ALTER TABLE t1 ADD COLUMN c2 INT, ADD COLUMN c3 INT, ALGORITHM=INSTANT;
```

`INFORMATION_SCHEMA.INNODB_TABLES` および `INFORMATION_SCHEMA.INNODB_COLUMNS` は、即時に追加されたカラムのメタデータを提供します。 `INFORMATION_SCHEMA.INNODB_TABLES.INSTANT_COLS` では、最初のインスタントカラムを追加する前に、テーブルのカラム数が表示されます。

`INFORMATION_SCHEMA.INNODB_COLUMNS.HAS_DEFAULT` および `DEFAULT_VALUE` は、即時に追加されたカラムのデフォルト値に関するメタデータを提供します。

`auto-increment` カラムを追加する場合、同時 DML は許可されません。データは大幅に再編成され、コストのかかる操作になります。少なくとも、`ALGORITHM=INPLACE`、`LOCK=SHARED` が必要です。

`ALGORITHM=INPLACE` を使用してカラムを追加すると、テーブルが再構築されます。

- カラムの削除

```
ALTER TABLE tbl_name DROP COLUMN column_name, ALGORITHM=INPLACE, LOCK=NONE;
```

データは大幅に再編成され、コストのかかる操作になります。

- カラム名の変更

```
ALTER TABLE tbl CHANGE old_col_name new_col_name data_type, ALGORITHM=INPLACE, LOCK=NONE;
```

同時 DML を許可するには、同じデータ型を保持し、カラム名のみを変更します。

同じデータ型と `[NOT] NULL` 属性を保持し、カラム名の変更のみを行う場合、操作は常にオンラインで実行できます。

外部キー制約の一部であるカラムの名前を変更することもできます。外部キー定義は、新しいカラム名を使用するように自動的に更新されます。外部キーに参加するカラムの名前の変更は、`ALGORITHM=INPLACE` でのみ機能します。`ALGORITHM=COPY` 句を使用した場合、または他のなんらかの状況でコマンドがバックグラウンドで `ALGORITHM=COPY` を使用する場合、`ALTER TABLE` ステートメントは失敗します。

`ALGORITHM=INPLACE` では、`generated column` の名前の変更はサポートされていません。

- カラムの並替え

カラムの順序を変更するには、`CHANGE` または `MODIFY` 操作で `FIRST` または `AFTER` を使用します。

```
ALTER TABLE tbl_name MODIFY COLUMN col_name column_definition FIRST, ALGORITHM=INPLACE, LOCK=NONE;
```

データは大幅に再編成され、コストのかかる操作になります。

- カラムのデータ型の変更

```
ALTER TABLE tbl_name CHANGE c1 c1 BIGINT, ALGORITHM=COPY;
```

カラムのデータ型の変更は、**ALGORITHM=COPY** でのみサポートされます。

- **VARCHAR** カラムサイズの拡張

```
ALTER TABLE tbl_name CHANGE COLUMN c1 c1 VARCHAR(255), ALGORITHM=INPLACE, LOCK=NONE;
```

VARCHAR カラムに必要な長さバイト数は、同じままにする必要があります。サイズが 0 から 255 バイトの **VARCHAR** カラムの場合、値のエンコードには長さバイトが必要です。256 バイト以上の **VARCHAR** カラムの場合は、長さが 2 バイト必要です。その結果、インプレース **ALTER TABLE** では、**VARCHAR** カラムサイズの 0 から 255 バイト、または 256 バイトからそれより大きいサイズへの増加のみがサポートされます。インプレース **ALTER TABLE** では、**VARCHAR** カラムのサイズを 256 バイト未満から 256 バイト以上に増やすことはサポートされていません。この場合、必要な長さバイト数は 1 から 2 に変更され、テーブルコピー (**ALGORITHM=COPY**) でのみサポートされます。たとえば、シングルバイト文字セットの **VARCHAR** カラムサイズを、インプレース **ALTER TABLE** を使用して **VARCHAR(255)** から **VARCHAR(256)** に変更しようとする、次のエラーが返されます:

```
ALTER TABLE tbl_name ALGORITHM=INPLACE, CHANGE COLUMN c1 c1 VARCHAR(256);
ERROR 0A000: ALGORITHM=INPLACE is not supported. Reason: Cannot change
column type INPLACE. Try ALGORITHM=COPY.
```

注記

VARCHAR カラムのバイト長は、文字セットのバイト長によって異なります。

インプレース **ALTER TABLE** を使用した **VARCHAR** サイズの縮小はサポートされていません。**VARCHAR** サイズを小さくするには、テーブルコピー (**ALGORITHM=COPY**) が必要です。

- カラムのデフォルト値の設定

```
ALTER TABLE tbl_name ALTER COLUMN col SET DEFAULT literal, ALGORITHM=INSTANT;
```

テーブルメタデータのみを変更します。デフォルトのカラム値は **data dictionary** に格納されます。

- カラムのデフォルト値の削除

```
ALTER TABLE tbl ALTER COLUMN col DROP DEFAULT, ALGORITHM=INSTANT;
```

- 自動インクリメント値の変更

```
ALTER TABLE table AUTO_INCREMENT=next_value, ALGORITHM=INPLACE, LOCK=NONE;
```

データファイルではなく、メモリーに格納された値を変更します。

レプリケーションまたはシャーディングを使用する分散システムでは、テーブルの自動増分カウンタを特定の値にリセットすることがあります。テーブルに挿入された次の行は、その自動インクリメントカラムの指定された値を使用します。この方法は、すべてのテーブルを定期的に空にしてリロードし、自動増分順序を 1 から再開するデータウェアハウス環境でも使用できます。

- カラムの **NULL** 化

```
ALTER TABLE tbl_name MODIFY COLUMN column_name data_type NULL, ALGORITHM=INPLACE, LOCK=NONE;
```

テーブルを適切に再構築します。データは大幅に再編成され、コストのかかる操作になります。

- カラムの **NOT NULL** 化

```
ALTER TABLE tbl_name MODIFY COLUMN column_name data_type NOT NULL, ALGORITHM=INPLACE, LOCK=NONE;
```

テーブルを適切に再構築します。操作を成功させるには、**STRICT_ALL_TABLES** または **STRICT_TRANS_TABLES SQL_MODE** が必要です。カラムに **NULL** 値が含まれている場合、操作は失敗します。サーバーは、参照整合性が失われる可能性がある外部キーカラムの変更を禁止します。[セクション 13.1.9 「ALTER TABLE ステートメント」](#) を参照してください。データは大幅に再編成され、コストのかかる操作になります。

- **ENUM** または **SET** カラムの定義の変更

```
CREATE TABLE t1 (c1 ENUM('a', 'b', 'c'));
```

```
ALTER TABLE t1 MODIFY COLUMN c1 ENUM('a', 'b', 'c', 'd'), ALGORITHM=INSTANT;
```

新しい列挙を追加するか、有効なメンバー値のリストの end にメンバーを設定して、**ENUM** または **SET** カラムの定義を変更すると、データ型の記憶域サイズが変更されないがぎり、すぐに実行することも、その場で実行することもできます。たとえば、8 つのメンバーを持つ **SET** カラムにメンバーを追加すると、値ごとに必要な記憶域が 1 バイトから 2 バイトに変更されます。これにはテーブルのコピーが必要です。リストの途中でメンバーを追加すると、既存のメンバーの番号が変更されます。これには、テーブルコピーが必要になります。

生成されたカラム操作

次のテーブルに、生成されるカラム操作のオンライン DDL サポートの概要を示します。詳細は、[構文および使用上のノート](#)を参照してください。

表 15.19 生成されたカラム操作のオンライン DDL サポート

操作	インスタント	インプレース	テーブルの再構築	同時 DML の許可	メタデータの変更のみ
STORED カラムの追加	いいえ	いいえ	はい	いいえ	いいえ
STORED カラムの順序の変更	いいえ	いいえ	はい	いいえ	いいえ
STORED カラムの削除	いいえ	はい	はい	はい	いいえ
VIRTUAL カラムの追加	はい	はい	いいえ	はい	はい
VIRTUAL カラムの順序の変更	いいえ	いいえ	はい	いいえ	いいえ
VIRTUAL カラムの削除	はい	はい	いいえ	はい	はい

構文および使用上のノート

- **STORED** カラムの追加

```
ALTER TABLE t1 ADD COLUMN (c2 INT GENERATED ALWAYS AS (c1 + 1) STORED), ALGORITHM=COPY;
```

式はサーバーによって評価される必要があるため、**ADD COLUMN** はストアドカラムのインプレース操作ではありません（一時テーブルを使用せずに実行されます）。

- **STORED** カラムの順序の変更

```
ALTER TABLE t1 MODIFY COLUMN c2 INT GENERATED ALWAYS AS (c1 + 1) STORED FIRST, ALGORITHM=COPY;
```

テーブルを適切に再構築します。

- **STORED** カラムの削除

```
ALTER TABLE t1 DROP COLUMN c2, ALGORITHM=INPLACE, LOCK=NONE;
```

テーブルを適切に再構築します。

- **VIRTUAL** カラムの追加

```
ALTER TABLE t1 ADD COLUMN (c2 INT GENERATED ALWAYS AS (c1 + 1) VIRTUAL), ALGORITHM=INSTANT;
```

仮想カラムの追加は、即時に実行することも、パーティション化されていないテーブルに対して適切に実行することもできます。

VIRTUAL の追加は、パーティションテーブルのインプレース操作ではありません。

- **VIRTUAL** カラムの順序の変更

```
ALTER TABLE t1 MODIFY COLUMN c2 INT GENERATED ALWAYS AS (c1 + 1) VIRTUAL FIRST, ALGORITHM=COPY;
```

- **VIRTUAL** カラムの削除

```
ALTER TABLE t1 DROP COLUMN c2, ALGORITHM=INSTANT;
```

VIRTUAL カラムの削除は、即時に実行することも、パーティション化されていないテーブルに対して適切に実行することもできます。

外部キー操作

次のテーブルに、外部キー操作のオンライン DDL サポートの概要を示します。アスタリスクは、追加情報、例外または依存関係を示します。詳細は、[構文および使用上のノート](#)を参照してください。

表 15.20 外部キー操作のオンライン DDL サポート

操作	インスタント	インプレース	テーブルの再構築	同時 DML の許可	メタデータの変更のみ
外部キー制約の追加	いいえ	はい*	いいえ	はい	はい
外部キー制約の削除	いいえ	はい	いいえ	はい	はい

構文および使用上のノート

- 外部キー制約の追加

INPLACE アルゴリズムは、[foreign_key_checks](#) が無効な場合にサポートされます。それ以外の場合は、**COPY** アルゴリズムのみがサポートされます。

```
ALTER TABLE tbl1 ADD CONSTRAINT fk_name FOREIGN KEY index (col1)
REFERENCES tbl2(col2) referential_actions;
```

- 外部キー制約の削除

```
ALTER TABLE tbl DROP FOREIGN KEY fk_name;
```

外部キーの削除は、[foreign_key_checks](#) オプションが有効または無効になった状態でオンラインで実行できます。

特定のテーブル上の外部キー制約の名前がわからない場合は、次のステートメントを発行し、各外部キーに対する **CONSTRAINT** 句で制約名を見つけます。

```
SHOW CREATE TABLE tableG
```

または、**INFORMATION_SCHEMA.TABLE_CONSTRAINTS** テーブルをクエリーして、**CONSTRAINT_NAME** カラムおよび **CONSTRAINT_TYPE** カラムを使用して外部キー名を識別します。

単一のステートメントで外部キーとそれに関連付けられたインデックスを削除することもできます:

```
ALTER TABLE table DROP FOREIGN KEY constraint, DROP INDEX index;
```

注記

変更対象のテーブルに **foreign keys** がすでに存在する場合 (つまり、**FOREIGN KEY ... REFERENCE** 句を含む **child table** である場合)、外部キーカラムが直接関係していない場合でも、オンライン DDL 操作に追加の制限が適用されます:

- 親テーブルに対する変更によって、**CASCADE** または **SET NULL** パラメータを使用した **ON UPDATE** または **ON DELETE** 句を介して子テーブルの関連する変更が発生した場合、子テーブルの **ALTER TABLE** は別のトランザクションがコミットされるのを待機できます。
- 同様に、テーブルが外部キー関係の **parent table** である場合、**FOREIGN KEY** 句が含まれていなくても、**INSERT**、**UPDATE** または **DELETE** ステートメントによって子テーブルの **ON UPDATE** または **ON DELETE** アクションが完了するまで **ALTER TABLE** を待機できます。

テーブルの操作

次のテーブルに、テーブル操作のオンライン DDL サポートの概要を示します。アスタリスクは、追加情報、例外または依存関係を示します。詳細は、[構文および使用上のノート](#)を参照してください。

表 15.21 テーブル操作のオンライン DDL サポート

操作	インスタント	インプレース	テーブルの再構築	同時 DML の許可	メタデータの変更のみ
ROW_FORMAT の変更	いいえ	はい	はい	はい	いいえ
KEY_BLOCK_SIZE の変更	いいえ	はい	はい	はい	いいえ
永続テーブルの統計の設定	いいえ	はい	いいえ	はい	はい
文字セットの指定	いいえ	はい	はい*	いいえ	いいえ
文字セットの変換	いいえ	いいえ	はい*	いいえ	いいえ
テーブルの最適化	いいえ	はい*	はい	はい	いいえ
FORCE オプションを使用した再構築	いいえ	はい*	はい	はい	いいえ
null の再構築の実行	いいえ	はい*	はい	はい	いいえ
テーブル名の変更	はい	はい	いいえ	はい	はい

構文および使用上のノート

- [ROW_FORMAT](#) の変更

```
ALTER TABLE tbl_name ROW_FORMAT = row_format, ALGORITHM=INPLACE, LOCK=NONE;
```

データは大幅に再編成され、コストのかかる操作になります。

[ROW_FORMAT](#) オプションの詳細は、[テーブルオプション](#)を参照してください。

- [KEY_BLOCK_SIZE](#) の変更

```
ALTER TABLE tbl_name KEY_BLOCK_SIZE = value, ALGORITHM=INPLACE, LOCK=NONE;
```

データは大幅に再編成され、コストのかかる操作になります。

[KEY_BLOCK_SIZE](#) オプションの詳細は、[テーブルオプション](#)を参照してください。

- 永続テーブル統計オプションの設定

```
ALTER TABLE tbl_name STATS_PERSISTENT=0, STATS_SAMPLE_PAGES=20, STATS_AUTO_RECALC=1, ALGORITHM=INPLACE, LOCK=NONE;
```

テーブルメタデータのみを変更します。

永続統計には、[STATS_PERSISTENT](#)、[STATS_AUTO_RECALC](#) および [STATS_SAMPLE_PAGES](#) が含まれます。詳細は、[セクション 15.8.10.1 「永続的オブティマイザ統計のパラメータの構成」](#)を参照してください。

- 文字セットの指定

```
ALTER TABLE tbl_name CHARACTER SET = charset_name, ALGORITHM=INPLACE, LOCK=NONE;
```

新しい文字エンコーディングが別のものである場合は、テーブルを再構築します。

- 文字セットの変換

```
ALTER TABLE tbl_name CONVERT TO CHARACTER SET charset_name, ALGORITHM=COPY;
```

新しい文字エンコーディングが別のものである場合は、テーブルを再構築します。

- テーブルの最適化

```
OPTIMIZE TABLE tbl_name;
```

インプレース操作は、**FULLTEXT** インデックスのあるテーブルではサポートされていません。この操作では **INPLACE** アルゴリズムを使用しますが、**ALGORITHM** および **LOCK** 構文は許可されていません。

- **FORCE** オプションを使用したテーブルの再構築

```
ALTER TABLE tbl_name FORCE, ALGORITHM=INPLACE, LOCK=NONE;
```

MySQL 5.6.17 . **ALGORITHM=INPLACE** is not supported for tables with **FULLTEXT** インデックスの時点で **ALGORITHM=INPLACE** を使用します。

- 「null」再構築の実行

```
ALTER TABLE tbl_name ENGINE=InnoDB, ALGORITHM=INPLACE, LOCK=NONE;
```

MySQL 5.6.17 の時点では、**ALGORITHM=INPLACE** を使用します。**ALGORITHM=INPLACE** は、**FULLTEXT** インデックスのあるテーブルではサポートされていません。

- テーブル名の変更

```
ALTER TABLE old_tbl_name RENAME TO new_tbl_name, ALGORITHM=INSTANT;
```

テーブルの名前変更は、即時に実行することも、インプレースで実行することもできます。MySQL は、コピーを作成せずに、テーブル **tbl_name** に対応するファイルの名前を変更します。(**RENAME TABLE** ステートメントを使用してテーブルの名前を変更することもできます。 [セクション13.1.36「RENAME TABLE ステートメント」](#) を参照してください。) 名前を変更したテーブル専用には付与された権限は、新しい名前に移行されません。それらは、手動で変更する必要があります。

テーブルスペースの操作

次のテーブルに、テーブルスペース操作のオンライン DDL サポートの概要を示します。詳細は、[構文および使用上のノート](#)を参照してください。

表 15.22 テーブルスペース操作のオンライン DDL サポート

操作	インスタント	インプレース	テーブルの再構築	同時 DML の許可	メタデータの変更のみ
一般テーブルスペースの名前の変更	いいえ	はい	いいえ	はい	はい
一般的なテーブルスペース暗号化の有効化または無効化	いいえ	はい	いいえ	はい	いいえ
file-per-table テーブルスペース暗号化の有効化または無効化	いいえ	いいえ	はい	いいえ	いいえ

構文および使用上のノート

- 一般テーブルスペースの名前の変更

```
ALTER TABLESPACE tablespace_name RENAME TO new_tablespace_name;
```

ALTER TABLESPACE ... RENAME TO は **INPLACE** アルゴリズムを使用しますが、**ALGORITHM** 句はサポートしていません。

- 一般的なテーブルスペース暗号化の有効化または無効化

```
ALTER TABLESPACE tablespace_name ENCRYPTION='Y';
```

`ALTER TABLESPACE ... ENCRYPTION` は `INPLACE` アルゴリズムを使用しますが、`ALGORITHM` 句はサポートしていません。

関連情報については、[セクション15.13「InnoDB 保存データ暗号化」](#)を参照してください。

- file-per-table テーブルスペース暗号化の有効化または無効化

```
ALTER TABLE tbl_name ENCRYPTION='Y', ALGORITHM=COPY;
```

関連情報については、[セクション15.13「InnoDB 保存データ暗号化」](#)を参照してください。

パーティション化操作

一部の `ALTER TABLE` パーティション化句を除き、パーティション化された InnoDB テーブルのオンライン DDL 操作は、通常の InnoDB テーブルに適用されるのと同じルールに従います。

一部の `ALTER TABLE` パーティション化句は、通常非パーティション InnoDB テーブルと同じ内部オンライン DDL API を経由しません。その結果、`ALTER TABLE` パーティション化句のオンラインサポートは異なります。

次のテーブルに、各 `ALTER TABLE` パーティション化ステートメントのオンラインステータスを示します。使用されるオンライン DDL API に関係なく、MySQL は可能な場合はデータのコピーおよびロックを最小限に抑えようとしません。

`ALGORITHM=COPY` を使用するか、「`ALGORITHM=DEFAULT, LOCK=DEFAULT`」のみを許可する `ALTER TABLE` パーティション化オプションでは、`COPY` アルゴリズムを使用してテーブルを再パーティション化します。つまり、新しいパーティション化されたテーブルは、新しいパーティション化スキームで作成されます。新しく作成されたテーブルには、`ALTER TABLE` ステートメントによって適用された変更が含まれ、テーブルデータが新しいテーブル構造にコピーされます。

表 15.23 パーティション化操作のオンライン DDL サポート

Partitioning 句	インスタント	インプレース	DML を許可	メモ
<code>PARTITION BY</code>	いいえ	いいえ	いいえ	<code>ALGORITHM=COPY, LOCK={SHARED EXCLUSIVE}</code> を許可
<code>ADD PARTITION</code>	いいえ	はい*	はい*	<code>ALGORITHM=INPLACE, LOCK={DEFAULT NONE SHARED EXCLUSIVE}</code> は、 <code>RANGE</code> および <code>LIST</code> パーティション、 <code>HASH</code> および <code>KEY</code> パーティションの <code>ALGORITHM=INPLACE, LOCK={DEFAULT SHARED EXCLUSIVE}</code> およびすべてのパーティションタイプの <code>ALGORITHM=COPY, LOCK={SHARED EXCLUSIVE}</code> でサポートされています。 <code>RANGE</code> または <code>LIST</code> によってパーティション化されたテーブルの既存のデータはコピー

Partitioning 句	インスタント	インプレース	DML を許可	メモ
				しません。MySQL は共有ロックを保持しながらデータをコピーするため、HASH または LIST によってパーティション化されたテーブルに対して <code>ALGORITHM=COPY</code> で同時クエリーが許可されます。
<code>DROP PARTITION</code>	いいえ	はい*	はい*	<p><code>ALGORITHM=INPLACE</code>、<code>LOCK={DEFAULT NONE SHARED EXCLUSIVE}</code> がサポートされています。<code>RANGE</code> または <code>LIST</code> によってパーティション化されたテーブルのデータはコピーしません。</p> <p><code>ALGORITHM=INPLACE</code> を使用した <code>DROP PARTITION</code> は、パーティションに格納されているデータを削除し、パーティションを削除します。ただし、<code>ALGORITHM=COPY</code> または <code>old_alter_table=ON</code> を使用した <code>DROP PARTITION</code> では、パーティションテーブルが再構築され、削除されたパーティションから互換性のある <code>PARTITION ... VALUES</code> 定義を持つ別のパーティションへのデータの移動が試行されます。別のパーティションに移動できないデータは削除されます。</p>
<code>DISCARD PARTITION</code>	いいえ	いいえ	いいえ	<code>ALGORITHM=DEFAULT</code> 、 <code>LOCK=DEFAULT</code> のみを許可
<code>IMPORT PARTITION</code>	いいえ	いいえ	いいえ	<code>ALGORITHM=DEFAULT</code> 、 <code>LOCK=DEFAULT</code> のみを許可
<code>TRUNCATE PARTITION</code>	いいえ	はい	はい	既存のデータをコピーしません。行を削除するだけで、テーブル自体またはそのパーティションの定義は変更されません。

Partitioning 句	インスタント	インプレース	DML を許可	メモ
COALESCE PARTITION	いいえ	はい*	いいえ	ALGORITHM=INPLACE、LOCK={DEFAULT SHARED EXCLUSIVE} がサポートされています。
REORGANIZE PARTITION	いいえ	はい*	いいえ	ALGORITHM=INPLACE、LOCK={DEFAULT SHARED EXCLUSIVE} がサポートされています。
EXCHANGE PARTITION	いいえ	はい	はい	
ANALYZE PARTITION	いいえ	はい	はい	
CHECK PARTITION	いいえ	はい	はい	
OPTIMIZE PARTITION	いいえ	いいえ	いいえ	ALGORITHM 句および LOCK 句は無視されます。テーブル全体を再構築します。セクション 24.3.4 「パーティションの保守」を参照してください。
REBUILD PARTITION	いいえ	はい*	いいえ	ALGORITHM=INPLACE、LOCK={DEFAULT SHARED EXCLUSIVE} がサポートされています。
REPAIR PARTITION	いいえ	はい	はい	
REMOVE PARTITIONING	いいえ	いいえ	いいえ	ALGORITHM=COPY、LOCK={SHARED EXCLUSIVE} を許可

パーティションテーブルに対する非パーティション化オンライン ALTER TABLE 操作は、通常のテーブルに適用されるのと同じルールに従います。ただし、ALTER TABLE は各テーブルパーティションに対してオンライン操作を実行するため、複数のパーティションで操作が実行されるため、システムリソースに対する需要が増加します。

ALTER TABLE パーティション化句の詳細は、パーティショニングオプション および セクション 13.1.9.1 「ALTER TABLE パーティション操作」を参照してください。一般的なパーティション化については、第 24 章 「パーティション化」を参照してください。

15.12.2 オンライン DDL のパフォーマンスと同時実行性

オンライン DDL は、MySQL 操作のいくつかの側面を改善します：

- DDL 操作の進行中にテーブルに対するクエリーおよび DML 操作を続行できるため、テーブルにアクセスするアプリケーションの応答性が向上します。ロックを削減し、MySQL サーバリソースを待機すると、DDL 操作に関係しない操作でもスケーラビリティが向上します。
- 即時操作では、データディクショナリのメタデータのみが変更されます。テーブルに対するメタデータロックは行われず、テーブルデータは影響を受けず、操作が即時に行われます。同時 DML は影響を受けません。
- オンライン操作により、テーブルコピー方法に関連付けられたディスク I/O および CPU サイクルが回避され、データベースの全体的な負荷が最小限に抑えられます。負荷を最小限に抑えると、DDL 操作中に良好なパフォーマンスと高スループットを維持できます。

- オンライン操作は、テーブルコピー操作より少ないデータをバッファプールに読み取り、頻繁にアクセスされるデータのメモリーからのページを削減します。頻繁にアクセスされるデータをページすると、DDL 操作後に一時的なパフォーマンスが低下する可能性があります。

LOCK 句

デフォルトでは、MySQL は DDL 操作中にできるだけ少ないロックを使用します。必要に応じて、**LOCK** 句をインプレース操作および一部のコピー操作に指定して、より限定的なロックを強制できます。**LOCK** 句で、特定の DDL 操作に許可されている制限レベルより低いロックが指定されている場合、ステートメントはエラーで失敗します。**LOCK** 句については、次に、最も制限の少ないものから順に説明します:

- **LOCK=NONE:**

同時クエリーおよび DML を許可します。

たとえば、長い DDL 操作中にテーブルを使用できないようにするには、顧客のサインアップまたは購入を含むテーブルに対してこの句を使用します。

- **LOCK=SHARED:**

同時クエリーは許可されますが、DML はブロックされます。

たとえば、データウェアハウステーブルでこの句を使用すると、DDL 操作が終了するまでデータロード操作を遅延できますが、クエリーを長期間遅延することはできません。

- **LOCK=DEFAULT:**

可能なかぎり多くの同時実行性を許可します (同時クエリーまたは DML、あるいはその両方)。**LOCK** 句を省略することは、**LOCK=DEFAULT** を指定することと同じです。

DDL ステートメントのデフォルトのロックレベルでテーブルの可用性の問題が発生することが予想されない場合は、この句を使用します。

- **LOCK=EXCLUSIVE:**

同時クエリーおよび DML をブロックします。

この句は、主な懸念事項が可能なかぎり短い時間で DDL 操作を終了することで、同時クエリーおよび DML アクセスが不要な場合に使用します。また、予期しないテーブルアクセスを避けるために、サーバーがアイドル状態であると想定される場合にも、この句を使用できます。

オンライン DDL およびメタデータロック

オンライン DDL 操作は、次の 3 つのフェーズを持つものとして表示できます:

- フェーズ 1: 初期化

初期化フェーズでは、サーバーは、ストレージエンジンの機能、ステートメントで指定された操作、およびユーザー指定の **ALGORITHM** オプションと **LOCK** オプションを考慮して、操作中に許可される同時実行性を決定します。このフェーズでは、現在のテーブル定義を保護するために、アップグレード可能な共有メタデータロックが取得されます。

- フェーズ 2: Execution

このフェーズでは、ステートメントが準備されて実行されます。メタデータロックが排他的にアップグレードされるかどうかは、初期化フェーズで評価される要因によって異なります。排他的メタデータロックが必要な場合は、ステートメントの準備中のみ簡単に取得されます。

- フェーズ 3: テーブル定義のコミット

テーブル定義のコミットフェーズでは、メタデータロックが排他的にアップグレードされ、古いテーブル定義が削除されて新しい定義がコミットされます。付与されると、排他的メタデータロックの期間が短くなります。

前述の排他的メタデータロック要件のため、オンライン DDL 操作では、テーブルのメタデータロックを保持する同時トランザクションがコミットまたはロールバックされるまで待機する必要がある場合があります。DDL 操作の前また

は実行中に開始されたトランザクションは、変更されるテーブルのメタデータロックを保持できます。長時間実行中または非アクティブなトランザクションの場合、オンライン DDL 操作は排他的メタデータロックの待機中にタイムアウトすることがあります。また、オンライン DDL 操作によってリクエストされた保留中の排他的メタデータロックによって、テーブルの後続のトランザクションがブロックされます。

次の例は、排他的メタデータロックを待機しているオンライン DDL 操作と、保留中のメタデータロックがテーブルの後続のトランザクションをブロックする方法を示しています。

セッション 1:

```
mysql> CREATE TABLE t1 (c1 INT) ENGINE=InnoDB;
mysql> START TRANSACTION;
mysql> SELECT * FROM t1;
```

セッション 1 の **SELECT** ステートメントは、テーブル t1 で共有メタデータロックを取得します。

セッション 2:

```
mysql> ALTER TABLE t1 ADD COLUMN x INT, ALGORITHM=INPLACE, LOCK=NONE;
```

テーブル定義の変更をコミットするためにテーブル t1 の排他的メタデータロックを必要とするセッション 2 のオンライン DDL 操作は、セッション 1 のトランザクションがコミットまたはロールバックされるまで待機する必要があります。

セッション 3:

```
mysql> SELECT * FROM t1;
```

セッション 3 で発行された **SELECT** ステートメントは、セッション 2 の **ALTER TABLE** 操作によってリクエストされた排他的メタデータロックが付与されるのを待機してブロックされます。

SHOW FULL PROCESSLIST を使用して、トランザクションがメタデータロックを待機しているかどうかを確認できます。

```
mysql> SHOW FULL PROCESSLISTG
...
***** 2. row *****
  Id: 5
  User: root
  Host: localhost
  db: test
  Command: Query
  Time: 44
  State: Waiting for table metadata lock
  Info: ALTER TABLE t1 ADD COLUMN x INT, ALGORITHM=INPLACE, LOCK=NONE
...
***** 4. row *****
  Id: 7
  User: root
  Host: localhost
  db: test
  Command: Query
  Time: 5
  State: Waiting for table metadata lock
  Info: SELECT * FROM t1
4 rows in set (0.00 sec)
```

メタデータロック情報は、セッション間のメタデータロックの依存関係、セッションが待機しているメタデータロック、および現在メタデータロックを保持しているセッションに関する情報を提供するパフォーマンススキーマ **metadata_locks** テーブルを介しても公開されます。詳細は、[セクション 27.12.13.3 「metadata_locks テーブル」](#) を参照してください。

オンライン DDL パフォーマンス

DDL 操作のパフォーマンスは、操作が即時に実行されるかどうか、インプレースで実行されるかどうか、およびテーブルを再構築するかどうかによって主に決定されます。

DDL 操作の相対パフォーマンスを評価するには、`ALGORITHM=INSTANT`、`ALGORITHM=INPLACE` および `ALGORITHM=COPY` を使用して結果を比較します。 `old_alter_table` を有効にしてステートメントを実行し、`ALGORITHM=COPY` を強制的に使用することもできます。

テーブルデータを変更する DDL 操作の場合は、コマンドの終了後に表示される「影響を受ける行」値を参照して、DDL 操作で変更を実行するか、テーブルのコピーを実行するかを決定できます。例:

- カラムのデフォルト値の変更 (高速、テーブルデータへの影響なし):

```
Query OK, 0 rows affected (0.07 sec)
```

- インデックスの追加 (時間はかかりますが、0 rows affected はテーブルがコピーされないことを示しています):

```
Query OK, 0 rows affected (21.42 sec)
```

- カラムのデータ型の変更 (かなりの時間がかかり、テーブルのすべての行を再構築する必要があります):

```
Query OK, 1671168 rows affected (1 min 35.54 sec)
```

大規模なテーブルに対して DDL 操作を実行する前に、次のように操作が高速か低速かを確認します:

1. テーブル構造をクローニングします。
2. クローンテーブルに少量のデータを移入します。
3. クローニングされたテーブルで DDL 操作を実行します。
4. 「rows affected」の値が 0 かどうかをチェックします。ゼロ以外の値は、特別な計画を必要とする可能性があるテーブルデータがコピーされることを意味します。たとえば、スケジュールされた停止時間中に DDL 操作を実行したり、各レプリカサーバーで一度に 1 つずつ DDL 操作を実行できます。

注記

DDL 操作に関連する MySQL 処理をより深く理解するには、DDL 操作の前後に InnoDB に関連するパフォーマンススキーマおよび `INFORMATION_SCHEMA` テーブルを調べて、物理読み取り、書き込み、メモリー割当てなどの数を確認します。

パフォーマンススキーマのステージイベントを使用して、`ALTER TABLE` の進行状況をモニターできます。 [セクション 15.16.1 「パフォーマンススキーマを使用した InnoDB テーブルの ALTER TABLE の進行状況のモニタリング」](#) を参照してください。

同時 DML 操作によって行われた変更の記録、最後へのそれらの変更の適用に関連する処理作業がいくつかあるため、オンライン DDL 操作は、他のセッションからのテーブルのアクセスをブロックするテーブルコピーメカニズムよりも全体的に時間がかかる可能性があります。raw パフォーマンスの低下は、そのテーブルを使用するアプリケーションの応答性の向上とバランスがとれています。テーブル構造を変更する手法を評価する場合は、web ページのロード時間などの要因に基づいて、エンドユーザーがパフォーマンスを認識することを検討してください。

15.12.3 オンライン DDL 領域の要件

インプレースのオンライン DDL 操作の領域要件の概要を次に示します。領域要件は、即時に実行される操作には適用されません。

- 一時ログファイル用の領域

一時ログファイルには、オンライン DDL 操作によってインデックスが作成されるか、テーブルが変更されると、同時 DML が記録されます。一時ログファイルは、`innodb_sort_buffer_size` の値によって必要に応じて、`innodb_online_alter_log_max_size` で指定された最大値まで拡張されます。一時ログファイルがサイズ制限を超えると、オンライン DDL 操作は失敗し、コミットされていない同時 DML 操作がロールバックされます。大規模な `innodb_online_alter_log_max_size` 設定では、オンライン DDL 操作中により多くの DML が許可されますが、ログに記録された DML を適用するためにテーブルがロックされている場合、DDL 操作の終了時の期間も延長されません。

操作に時間がかかり、一時ログファイルのサイズが `innodb_online_alter_log_max_size` の値を超えるように同時 DML によってテーブルが変更された場合、オンライン DDL 操作は `DB_ONLINE_LOG_TOO_BIG` エラーで失敗します。

- 一時ソートファイル用の領域

テーブルを再構築するオンライン DDL 操作では、インデックスの作成時に一時ソートファイルが MySQL 一時ディレクトリ (Unix の場合は `$TMPDIR`、Windows の場合は `%TEMP%`、`--tmpdir` で指定されたディレクトリ) に書き込まれます。一時ソートファイルは、元のテーブルを含むディレクトリには作成されません。各一時ソートファイルは、1 つのデータカラムを保持するのに十分な大きさであり、各ソートファイルは、そのデータが最終的なテーブルまたはインデックスにマージされると削除されます。一時ソートファイルを使用する操作には、テーブルのデータ量にインデックスを加えたものと同じ一時領域が必要になる場合があります。オンライン DDL 操作で、データディレクトリが存在するファイルシステム上の使用可能なすべてのディスク領域が使用されている場合は、エラーが報告されます。

MySQL 一時ディレクトリがソートファイルを保持するのに十分な大きさでない場合は、`tmpdir` を別のディレクトリに設定します。または、`innodb_tmpdir` を使用して、オンライン DDL 操作用に個別の一時ディレクトリを定義します。このオプションは、大規模な一時ソートファイルの結果として発生する可能性のある一時ディレクトリのオーバーフローを回避するために導入されました。

- 中間テーブルファイル用の領域

テーブルを再構築する一部のオンライン DDL 操作では、元のテーブルと同じディレクトリに一時中間テーブルファイルが作成されます。中間テーブルファイルには、元のテーブルのサイズと等しい領域が必要な場合があります。中間テーブルのファイル名は `#sql-ib` 接頭辞で始まり、オンライン DDL 操作中にのみ簡単に表示されます。

`innodb_tmpdir` オプションは、中間テーブルファイルには適用されません。

15.12.4 オンライン DDL を使用した DDL ステートメントの簡略化

オンライン DDL が導入される前は、多くの DDL 操作を 1 つの `ALTER TABLE` ステートメントに結合することが一般的な習慣でした。各 `ALTER TABLE` ステートメントにはテーブルのコピーと再構築が含まれていたため、テーブルに対するすべての変更を 1 回の再構築操作で実行できたことから、同じテーブルへのいくつかの変更を一度に行う方が効率的でした。マイナス面としては、DDL 操作に関連する SQL コードが保守しにくく、別のスクリプトでの再利用も難しい点がありました。特定の変更が毎回異なっていたとすると、少し異なるシナリオごとに、新しい複雑な `ALTER TABLE` の構築が必要になる可能性があります。

オンラインで実行できる DDL 操作の場合は、効率を犠牲にすることなく、スクリプトおよびメンテナンスを容易にするために個々の `ALTER TABLE` ステートメントに分割できます。たとえば、次のような複雑なステートメントを取り上げ、

```
ALTER TABLE t1 ADD INDEX i1(c1), ADD UNIQUE INDEX i2(c2),  
CHANGE c4_old_name c4_new_name INTEGER UNSIGNED;
```

それを独立してテストおよび実行できる、次のようなより簡単な部分に分解することができます。

```
ALTER TABLE t1 ADD INDEX i1(c1);  
ALTER TABLE t1 ADD UNIQUE INDEX i2(c2);  
ALTER TABLE t1 CHANGE c4_old_name c4_new_name INTEGER UNSIGNED NOT NULL;
```

複数の部分からなる `ALTER TABLE` ステートメントは、次の目的に引き続き使用できます。

- 特定のシーケンスで実行する必要のある操作。たとえば、インデックスの作成に続けて、そのインデックスを使用する外部キー制約を作成する場合など。
- グループとして成功または失敗するようにしたい、すべてが同じ特定の `LOCK` 句を使用している操作。
- オンラインで実行できない (つまり、引き続き `table-copy` メソッドを使用する) 操作。
- 特殊なシナリオでの正確な下位互換性のために必要な場合に強制的にテーブルコピー動作を行うために、`ALGORITHM=COPY` または `old_alter_table=1` を指定する操作。

15.12.5 オンライン DDL 失敗条件

オンライン DDL 操作の失敗は、通常、次のいずれかの状況が原因です:

- `ALGORITHM` 句では、特定のタイプの DDL 操作またはストレージエンジンと互換性のないアルゴリズムを指定します。

- `LOCK` 句では、特定のタイプの DDL 操作と互換性のない低レベルのロック (`SHARED` または `NONE`) を指定します。
- テーブルでの `exclusive lock` の待機中にタイムアウトが発生し、DDL 操作の初期フェーズおよび最終フェーズで短時間必要になる場合があります。
- `tmpdir` または `innodb_tmpdir` ファイルシステムのディスク領域が不足していますが、MySQL はインデックスの作成中に一時ソートファイルをディスクに書き込みます。詳細は、[セクション15.12.3「オンライン DDL 領域の要件」](#)を参照してください。
- 操作には時間がかかり、同時 DML は一時オンラインログのサイズが `innodb_online_alter_log_max_size` 構成オプションの値を超えるようにテーブルを変更します。この状態は `DB_ONLINE_LOG_TOO_BIG` エラーの原因になります。
- 同時 DML は、元のテーブル定義では許可されているが、新しいテーブル定義では許可されていないテーブルに変更を加えます。この操作は、MySQL がいちばん最後に、並列 DML ステートメントからのすべての変更を適用しようとしたときにのみ失敗します。たとえば、一意のインデックスの作成中にカラムに重複した値を挿入したり、そのカラムでの主キーのインデックスの作成中にカラムに `NULL` 値を挿入したりすることがあります。並列 DML によって行われた変更が優先され、`ALTER TABLE` 操作は実質的にロールバックされます。

15.12.6 オンライン DDL の制限事項

オンライン DDL 操作には、次の制限が適用されます：

- このテーブルは、`TEMPORARY TABLE` でインデックスを作成するときにコピーされます。
- テーブルに `ON...CASCADE` または `ON...SET NULL` 制約がある場合、`ALTER TABLE` 句の `LOCK=NONE` は使用できません。
- インプレースのオンライン DDL 操作を終了する前に、テーブルのメタデータロックを保持するトランザクションがコミットまたはロールバックされるまで待機する必要があります。オンライン DDL 操作では、実行フェーズ中にテーブルに対する排他的メタデータロックが短時間必要になる場合があります。テーブル定義の更新時には常に操作の最終フェーズで必要になります。その結果、テーブルのメタデータロックを保持しているトランザクションによって、オンライン DDL 操作がブロックされる可能性があります。テーブルのメタデータロックを保持するトランザクションは、オンライン DDL 操作の前または実行中に開始されている可能性があります。テーブルのメタデータロックを保持する長時間実行中または非アクティブなトランザクションによって、オンライン DDL 操作がタイムアウトする可能性があります。
- インプレースのオンライン DDL 操作を実行する場合、`ALTER TABLE` ステートメントを実行するスレッドは、他の接続スレッドから同じテーブルに対して同時に実行された DML 操作のオンラインログを適用します。これらの DML 操作が適用されると、重複したキーエントリのエラー (`ERROR 1062 (23000): 重複したエントリ`) が発生する可能性があります。これは、重複したエントリが一時的なだけで、オンラインログのあとの方のエントリによって元に戻されるとしても同じです。これは、トランザクション中は制約を保持する必要のある、`InnoDB` での外部キー制約チェックの考え方に似ています。
- `InnoDB` テーブルに対する `OPTIMIZE TABLE` は、テーブルを再構築して、インデックス統計を更新し、クラスタ化されたインデックス内の未使用領域を解放するための `ALTER TABLE` 操作にマップされます。主キーに現れる順序でキーが挿入されるため、セカンダリインデックスはそれほど効率的に作成されません。`OPTIMIZE TABLE` は、通常の `InnoDB` テーブルおよびパーティション化された `InnoDB` テーブルを再構築するためのオンライン DDL サポートが追加されてサポートされています。
- 一時カラム (`DATE`、`DATETIME` または `TIMESTAMP`) を含み、`ALGORITHM=COPY` を使用して再構築されていない MySQL 5.6 より前に作成されたテーブルは、`ALGORITHM=INPLACE` をサポートしていません。この場合は、`ALTER TABLE ... ALGORITHM=INPLACE` 操作によって次のエラーが返されます。

```
ERROR 1846 (0A000): ALGORITHM=INPLACE is not supported.  
Reason: Cannot change column type INPLACE. Try ALGORITHM=COPY.
```

- 通常、テーブルの再構築を伴う大規模なテーブルに対するオンライン DDL 操作には、次の制限が適用されます：
 - オンライン DDL 操作を一時停止したり、オンライン DDL 操作の I/O または CPU 使用率を抑制するメカニズムはありません。
 - オンライン DDL 操作のロールバックは、操作が失敗した場合にコストがかかる可能性があります。

- オンライン DDL 操作を長時間実行すると、レプリケーションラグが発生する可能性があります。オンライン DDL 操作は、レプリカで実行する前にソースで実行を終了する必要があります。また、ソースで同時に処理された DML は、レプリカでの DDL 操作が完了した後にのみレプリカで処理されます。

大規模なテーブルに対するオンライン DDL 操作の実行に関連する追加情報は、[セクション15.12.2「オンライン DDL のパフォーマンスと同時実行性」](#) を参照してください。

15.13 InnoDB 保存データ暗号化

InnoDB では、[file-per-table](#) テーブルスペース、[general](#) テーブルスペース、[mysql](#) システムテーブルスペース、redo ログおよび undo ログの保存データ暗号化がサポートされています。

MySQL 8.0.16 では、スキーマおよび一般テーブルスペースの暗号化デフォルトの設定もサポートされているため、DBA はこれらのスキーマおよびテーブルスペースで作成されたテーブルを暗号化するかどうかを制御できます。

InnoDB の保存データ暗号化の機能については、このセクションの次のトピックで説明します。

- [保存データの暗号化について](#)
- [暗号化の前提条件](#)
- [スキーマおよび一般テーブルスペースの暗号化デフォルトの定義](#)
- [テーブルごとのファイルテーブルスペースの暗号化](#)
- [一般的なテーブルスペース暗号化](#)
- [二重書き込みファイル暗号化](#)
- [mysql システムテーブルスペースの暗号化](#)
- [redo ログの暗号化](#)
- [undo ログの暗号化](#)
- [マスターキーのローテーション](#)
- [暗号化とリカバリ](#)
- [暗号化されたテーブルスペースのエクスポート](#)
- [暗号化とレプリケーション](#)
- [暗号化されたテーブルスペースおよびスキーマの識別](#)
- [暗号化の進行状況の監視](#)
- [暗号化の使用上のノート](#)
- [暗号化の制限事項](#)

保存データの暗号化について

InnoDB では、マスター暗号化キーとテーブルスペースキーで構成される 2 層暗号化キーアーキテクチャを使用します。テーブルスペースが暗号化されると、テーブルスペースキーが暗号化され、テーブルスペースヘッダーに格納されます。アプリケーションまたは認証済ユーザーが暗号化されたテーブルスペースデータにアクセスする場合、InnoDB はマスター暗号化キーを使用してテーブルスペースキーを復号化します。復号化されたバージョンのテーブルスペースキーは変更されませんが、必要に応じてマスター暗号化キーを変更できます。このアクションはマスターキーのローテーションと呼ばれます。

保存データ暗号化機能は、マスター暗号化キー管理のためにキーリングプラグインに依存します。

すべての MySQL エディションには、サーバーホストに対してローカルなファイルにキーリングデータを格納する [keyring_file](#) プラグインが用意されています。

MySQL Enterprise Edition には、追加のキーリングプラグインが用意されています:

- `keyring_encrypted_file` は、サーバーホストに対してローカルな暗号化されたファイルにキーリングデータを格納します。
- `keyring_okv` には、KMIP 互換製品を鍵リングストレージのバックエンドとして使用する KMIP クライアント (KMIP 1.1) が含まれています。サポートされている KMIP 互換製品には、Oracle Key Vault、Gemalto KeySecure、Thales Vormetric キー管理サーバー、Fornetix Key Orchestration などの一元化された鍵管理ソリューションが含まれます。
- `keyring_aws` は、キー生成のバックエンドとして Amazon Web Services Key Management Service (AWS KMS) と通信し、キーの格納にローカルファイルを使用します。
- `keyring_hashicorp` は、バックエンドストレージのために HashiCorp Vault と通信します。

警告

`keyring_file` および `keyring_encrypted_file` プラグインは、規制コンプライアンスソリューションとしては意図されていません。PCI、FIPS などのセキュリティ標準では、キーポルトまたはハードウェアセキュリティモジュール (HSM) 内の暗号化キーを保護、管理および保護するためにキー管理システムを使用する必要があります。

セキュアで堅牢な暗号化キー管理ソリューションは、セキュリティおよび様々なセキュリティ標準への準拠に不可欠です。保存データ暗号化機能で一元化されたキー管理ソリューションを使用する場合、この機能は「MySQL Enterprise Transparent Data Encryption (TDE)」と呼ばれます。

保存データ暗号化機能は、Advanced Encryption Standard (AES) ブロックベースの暗号化アルゴリズムをサポートしています。テーブルスペースキー暗号化には電子コードブック (ECB) ブロック暗号化モードを使用し、データ暗号化には暗号ブロックチェーン (CBC) ブロック暗号化モードを使用します。

保存データ暗号化機能に関するよくある質問については、[セクションA.17「MySQL 8.0 FAQ : InnoDB 保存データ暗号化」](#)を参照してください。

暗号化の前提条件

- キーリングプラグインをインストールして構成する必要があります。キーリングプラグインのインストールは、起動時に `early-plugin-load` オプションを使用して実行されます。早期ロードにより、`InnoDB` ストレージエンジンを初期化する前にプラグインが使用可能になります。プラグインのインストールと構成の手順については、[セクション6.4.4「MySQL キーリング」](#)を参照してください。

一度に有効にできるキーリングプラグインは 1 つだけです。複数のキーリングプラグインの有効化はサポートされていません。

重要

暗号化されたテーブルスペースが MySQL インスタンスで作成されたら、暗号化されたテーブルスペースの作成時にロードされたキーリングプラグインは、`early-plugin-load` オプションを使用して起動時に引き続きロードされる必要があります。そうしないと、サーバーの起動時および `InnoDB` のリカバリ時にエラーが発生します。

キーリングプラグインがアクティブであることを確認するには、`SHOW PLUGINS` ステートメントを使用するか、`INFORMATION_SCHEMA.PLUGINS` テーブルをクエリーします。例:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
        FROM INFORMATION_SCHEMA.PLUGINS
        WHERE PLUGIN_NAME LIKE 'keyring%';
+-----+-----+
| PLUGIN_NAME | PLUGIN_STATUS |
+-----+-----+
| keyring_file | ACTIVE        |
+-----+-----+
```

- 本番データを暗号化する場合は、マスター暗号化キーが失われないようにするステップを実行してください。マスター暗号化キーが失われた場合、暗号化されたテーブルスペースファイルに格納されているデータはリカバ

りできません。 `keyring_file` または `keyring_encrypted_file` プラグインを使用する場合は、最初の暗号化されたテーブルスペースの作成直後、マスターキーのローテーションの前、およびマスターキーのローテーションの後に、キーリングデータファイルのバックアップを作成します。 `keyring_file_data` 構成オプションは、`keyring_file` プラグインのキーリングデータファイルの場所を定義します。 `keyring_encrypted_file_data` 構成オプションは、`keyring_encrypted_file` プラグインのキーリングデータファイルの場所を定義します。 `keyring_okv` または `keyring_aws` プラグインを使用する場合は、必要な構成が実行されていることを確認します。 その手順は、[セクション6.4.4「MySQL キーリング」](#)を参照してください。

スキーマおよび一般テーブルスペースの暗号化デフォルトの定義

MySQL 8.0.16 では、`default_table_encryption` システム変数によってスキーマおよび一般テーブルスペースのデフォルトの暗号化設定が定義されます。 `ENCRYPTION` 句が明示的に指定されていない場合、`CREATE TABLESPACE` および `CREATE SCHEMA` 操作によって `default_table_encryption` 設定が適用されます。

`ALTER SCHEMA` および `ALTER TABLESPACE` の操作では、`default_table_encryption` 設定は適用されません。 既存のスキーマまたは一般テーブルスペースの暗号化を変更するには、`ENCRYPTION` 句を明示的に指定する必要があります。

`default_table_encryption` 変数は、個々のクライアント接続に対して設定することも、`SET` 構文を使用してグローバルに設定することもできます。 たとえば、次のステートメントは、デフォルトのスキーマおよびテーブルスペースの暗号化をグローバルに有効にします：

```
mysql> SET GLOBAL default_table_encryption=ON;
```

スキーマのデフォルトの暗号化設定は、次の例に示すように、スキーマの作成または変更時に `DEFAULT ENCRYPTION` 句を使用して定義することもできます：

```
mysql> CREATE SCHEMA test DEFAULT ENCRYPTION = 'Y';
```

スキーマの作成時に `DEFAULT ENCRYPTION` 句が指定されていない場合、`default_table_encryption` 設定が適用されます。 既存のスキーマのデフォルト暗号化を変更するには、`DEFAULT ENCRYPTION` 句を指定する必要があります。 それ以外の場合、スキーマは現在の暗号化設定を保持します。

デフォルトでは、テーブルは作成されたスキーマまたは一般テーブルスペースの暗号化設定を継承します。 たとえば、暗号化対応スキーマで作成されたテーブルは、デフォルトで暗号化されます。 この動作により、DBA は、スキーマおよび一般的なテーブルスペース暗号化のデフォルトを定義して強制することで、テーブル暗号化の使用を制御できます。

暗号化のデフォルトは、`table_encryption_privilege_check` システム変数を有効にすることで適用されます。 `table_encryption_privilege_check` が有効な場合、`default_table_encryption` 設定とは異なる暗号化設定を使用してスキーマまたは一般テーブルスペースを作成または変更するとき、またはデフォルトのスキーマ暗号化とは異なる暗号化設定を使用してテーブルを作成または変更するとき、権限チェックが発生します。 `table_encryption_privilege_check` が無効 (デフォルト) の場合、権限チェックは実行されず、前述の操作は警告付きで続行できます。

`table_encryption_privilege_check` が有効な場合、デフォルトの暗号化設定をオーバーライドするには、`TABLE_ENCRYPTION_ADMIN` 権限が必要です。 DBA は、この権限を付与して、スキーマまたは一般テーブルスペースの作成または変更時にユーザーが `default_table_encryption` 設定から逸脱したり、テーブルの作成または変更時にデフォルトのスキーマ暗号化から逸脱できるようにすることができます。 この権限では、テーブルの作成または変更時に一般テーブルスペースの暗号化から逸脱することはできません。 テーブルの暗号化設定は、テーブルが存在する一般テーブルスペースと同じである必要があります。

テーブルごとのファイルテーブルスペースの暗号化

MySQL 8.0.16 の時点では、file-per-table テーブルスペースは、`CREATE TABLE` ステートメントで `ENCRYPTION` 句が明示的に指定されていない限り、テーブルが作成されるスキーマのデフォルトの暗号化を継承します。 MySQL 8.0.16 より前は、暗号化を有効にするために `ENCRYPTION` 句を指定する必要があります。

```
mysql> CREATE TABLE t1 (c1 INT) ENCRYPTION = 'Y';
```

既存の file-per-table テーブルスペースの暗号化を変更するには、`ENCRYPTION` 句を指定する必要があります。

```
mysql> ALTER TABLE t1 ENCRYPTION = 'Y';
```

MySQL 8.0.16 では、`table_encryption_privilege_check` 変数が有効になっている場合、デフォルトのスキーマ暗号化とは異なる設定で `ENCRYPTION` 句を指定するには、`TABLE_ENCRYPTION_ADMIN` 権限が必要です。 [スキーマおよび一般テーブルスペースの暗号化デフォルトの定義](#)を参照してください。

一般的なテーブルスペース暗号化

MySQL 8.0.16 では、`CREATE TABLESPACE` ステートメントで `ENCRYPTION` 句が明示的に指定されていないが、`default_table_encryption` 変数によって、新しく作成された一般テーブルスペースの暗号化が決定されます。MySQL 8.0.16 より前は、暗号化を有効にするために `ENCRYPTION` 句を指定する必要があります。

```
mysql> CREATE TABLESPACE `ts1` ADD DATAFILE 'ts1.ibd' ENCRYPTION = 'Y' Engine=InnoDB;
```

既存の一般テーブルスペースの暗号化を変更するには、`ENCRYPTION` 句を指定する必要があります。

```
mysql> ALTER TABLESPACE ts1 ENCRYPTION = 'Y';
```

MySQL 8.0.16 では、`table_encryption_privilege_check` 変数が有効になっている場合、`default_table_encryption` 設定とは異なる設定で `ENCRYPTION` 句を指定するには `TABLE_ENCRYPTION_ADMIN` 権限が必要です。 [スキーマおよび一般テーブルスペースの暗号化デフォルトの定義](#)を参照してください。

二重書込みファイル暗号化

二重書込みファイルの暗号化サポートは、MySQL 8.0.23 の時点で使用できます。InnoDB では、暗号化されたテーブルスペースに属する二重書込みファイルページが自動的に暗号化されます。必要なアクションはありません。二重書込みファイルページは、関連付けられたテーブルスペースの暗号化キーを使用して暗号化されます。テーブルスペースデータファイルに書き込まれた同じ暗号化ページも二重書込みファイルに書き込まれます。暗号化されていないテーブルスペースに属するファイルの二重書込みページは、暗号化されません。

リカバリ中、暗号化された二重書込みファイルページは暗号化されず、破損がないかどうかチェックされます。

mysql システムテーブルスペースの暗号化

mysql システムテーブルスペースの暗号化サポートは、MySQL 8.0.16 の時点で使用できます。

mysql システムテーブルスペースには、mysql システムデータベースおよび MySQL データディクショナリテーブルが含まれます。デフォルトでは暗号化されていません。mysql システムテーブルスペースの暗号化を有効にするには、`ALTER TABLESPACE` ステートメントでテーブルスペース名と `ENCRYPTION` オプションを指定します。

```
mysql> ALTER TABLESPACE mysql ENCRYPTION = 'Y';
```

mysql システムテーブルスペースの暗号化を無効にするには、`ALTER TABLESPACE` ステートメントを使用して `ENCRYPTION = 'N'`を設定します。

```
mysql> ALTER TABLESPACE mysql ENCRYPTION = 'N';
```

mysql システムテーブルスペースの暗号化を有効または無効にするには、インスタンス内のすべてのテーブル (`CREATE TABLESPACE on *.*`) に対する `CREATE TABLESPACE` 権限が必要です。

redo ログの暗号化

redo ログデータの暗号化は、`innodb_redo_log_encrypt` 構成オプションを使用して有効にします。redo ログの暗号化はデフォルトで無効になっています。

テーブルスペースデータと同様に、redo ログデータの暗号化は redo ログデータがディスクに書き込まれるときに行われ、復号化は redo ログデータがディスクから読み取られるときに行われます。redo ログデータがメモリーに読み込まれると、暗号化されていない形式になります。redo ログデータは、テーブルスペース暗号化キーを使用して暗号化および復号化されます。

`innodb_redo_log_encrypt` が有効な場合、ディスクに存在する暗号化されていない redo ログページは暗号化されずに残り、新しい redo ログページは暗号化された形式でディスクに書き込まれます。同様に、`innodb_redo_log_encrypt`

が無効な場合、ディスクに存在する暗号化された redo ログページは暗号化されたままになり、新しい redo ログページは暗号化されていない形式でディスクに書き込まれます。

テーブルスペース暗号化キーを含む redo ログ暗号化メタデータは、最初の redo ログファイル (`ib_logfile0`) のヘッダーに格納されます。このファイルを削除すると、redo ログの暗号化は無効になります。

redo ログの暗号化が有効になると、InnoDB は起動時に redo ページをスキャンできる必要があります。redo ログページが暗号化されている場合はスキャンできないため、キーリングプラグインなしまたは暗号化キーなしで通常の再起動はできません。キーリングプラグインまたは暗号化鍵がない場合は、redo ログ (`SRV_FORCE_NO_LOG_REDO`) を使用しない強制的な起動のみが可能です。セクション15.21.2「InnoDB のリカバリの強制的な実行」を参照してください。

undo ログの暗号化

undo ログデータの暗号化は、`innodb_undo_log_encrypt` 構成オプションを使用して有効にします。undo ログの暗号化は、`undo tablespaces` に存在する undo ログに適用されます。セクション15.6.3.4「undo テーブルスペース」を参照してください。undo ログデータの暗号化は、デフォルトで無効になっています。

テーブルスペースデータと同様に、undo ログデータの暗号化は undo ログデータがディスクに書き込まれるときに行われ、復号化は undo ログデータがディスクから読み取られるときに行われます。undo ログデータがメモリーに読み込まれると、暗号化されていない形式になります。undo ログデータは、テーブルスペース暗号化キーを使用して暗号化および復号化されます。

`innodb_undo_log_encrypt` が有効な場合、ディスクに存在する暗号化されていない undo ログページは暗号化されずに残り、新しい undo ログページは暗号化された形式でディスクに書き込まれます。同様に、`innodb_undo_log_encrypt` が無効になっている場合、ディスクに存在する暗号化された undo ログページは暗号化されたままになり、新しい undo ログページは暗号化されていない形式でディスクに書き込まれます。

undo ログ暗号化メタデータ (テーブルスペース暗号化キーを含む) は、undo ログファイルのヘッダーに格納されません。

注記

undo ログの暗号化が無効になっている場合、サーバーは、暗号化された undo ログデータを含む undo テーブルスペースが切り捨てられるまで、undo ログデータの暗号化に使用されたキーリングプラグインを引き続き必要とします。(暗号化ヘッダーは、undo テーブルスペースが切り捨てられた場合にのみ undo テーブルスペースから削除されます。) undo テーブルスペースの切捨手の詳細は、[undo テーブルスペースの切捨て](#) を参照してください。

マスターキーのローテーション

マスター暗号化キーは、定期的およびキーが危険にさらされた疑いがある場合は常にローテーションする必要があります。

マスターキーローテーションは、アトミックなインスタンスレベルの操作です。マスター暗号化キーがローテーションされるたびに、MySQL インスタンスのすべてのテーブルスペースキーが再暗号化され、それぞれのテーブルスペースヘッダーに保存されます。アトミック操作として、ローテーション操作が開始されたら、すべてのテーブルスペースキーに対して再暗号化が成功する必要があります。サーバー障害によってマスターキーのローテーションが中断された場合、InnoDB はサーバーの再起動時に操作をロールフォワードします。詳細は、[暗号化とリカバリ](#) を参照してください。

マスター暗号化キーをローテーションすると、マスター暗号化キーのみが変更され、テーブルスペースキーが再暗号化されます。関連付けられたテーブルスペースデータは復号化または再暗号化されません。

マスター暗号化キーをローテーションするには、`ENCRYPTION_KEY_ADMIN` 権限 (または非推奨の `SUPER` 権限) が必要です。

マスター暗号化キーをローテーションするには、次のコマンドを実行します:

```
mysql> ALTER INSTANCE ROTATE INNODB MASTER KEY;
```

`ALTER INSTANCE ROTATE INNODB MASTER KEY` では、同時 DML がサポートされます。ただし、テーブルスペースの暗号化操作と同時に実行することはできず、同時実行によって発生する可能性のある競合を防ぐためにロックが取得されます。`ALTER INSTANCE ROTATE INNODB MASTER KEY` 操作が実行中の場合は、テーブルスペースの暗号化操作を続行する前に操作を終了する必要があります、その逆も同様です。

暗号化とリカバリ

暗号化操作中にサーバー障害が発生した場合、サーバーの再起動時に操作がロールフォワードされます。一般的なテーブルスペースの場合、暗号化操作は最後に処理されたページからバックグラウンドスレッドで再開されます。

マスターキーのローテーション中にサーバー障害が発生した場合、InnoDB はサーバーの再起動時に操作を続行しません。

InnoDB の初期化およびリカバリアクティビティがテーブルスペースデータにアクセスする前に、テーブルスペースデータページの復号化に必要な情報をテーブルスペースヘッダーから取得できるように、ストレージエンジンの初期化の前にキープラグインをロードする必要があります。(暗号化の前提条件を参照してください。)

InnoDB の初期化およびリカバリが開始されると、マスターキーのローテーション操作が再開されます。サーバー障害のため、一部のテーブルスペースキーは新しいマスター暗号化キーを使用してすでに暗号化されている可能性があります。InnoDB は各テーブルスペースヘッダーから暗号化データを読み取り、データが古いマスター暗号化キーを使用してテーブルスペースキーが暗号化されていることを示している場合、InnoDB はキーリングから古いキーを取得し、それを使用してテーブルスペースキーを復号化します。次に、InnoDB は新しいマスター暗号化キーを使用してテーブルスペースキーを再暗号化し、再暗号化されたテーブルスペースキーをテーブルスペースヘッダーに保存します。

暗号化されたテーブルスペースのエクスポート

テーブルスペースのエクスポートは、file-per-table テーブルスペースでのみサポートされます。

暗号化されたテーブルスペースがエクスポートされると、InnoDB によって、テーブルスペースキーの暗号化に使用される転送キーが生成されます。暗号化されたテーブルスペースキーおよび転送キーは、`tablespace_name.cfp` ファイルに格納されます。インポート操作を実行するには、このファイルと暗号化されたテーブルスペースファイルが必要です。インポート時に、InnoDB は転送キーを使用して `tablespace_name.cfp` ファイルのテーブルスペースキーを復号化します。関連情報については、[セクション15.6.1.3「InnoDB テーブルのインポート」](#)を参照してください。

暗号化とレプリケーション

- `ALTER INSTANCE ROTATE INNODB MASTER KEY` ステートメントは、ソースおよびレプリカがテーブルスペースの暗号化をサポートするバージョンの MySQL を実行するレプリケーション環境でのみサポートされます。
- 成功した `ALTER INSTANCE ROTATE INNODB MASTER KEY` ステートメントは、レプリカ上のレプリケーションのためにバイナリログに書き込まれます。
- `ALTER INSTANCE ROTATE INNODB MASTER KEY` ステートメントが失敗した場合、バイナリログに記録されず、レプリカにレプリケートされません。
- キーリングプラグインがソースにインストールされているが、レプリカにはインストールされていない場合、`ALTER INSTANCE ROTATE INNODB MASTER KEY` 操作のレプリケーションは失敗します。
- `keyring_file` または `keyring_encrypted_file` プラグインがソースとレプリカの両方にインストールされているが、レプリカにキーリングデータファイルがない場合、レプリケートされた `ALTER INSTANCE ROTATE INNODB MASTER KEY` ステートメントは、キーリングファイルデータがメモリーにキャッシュされていないと想定して、レプリカにキーリングデータファイルを作成します。`ALTER INSTANCE ROTATE INNODB MASTER KEY` では、メモリーにキャッシュされているキーリングファイルデータが使用されます(使用可能な場合)。

暗号化されたテーブルスペースおよびスキーマの識別

MySQL 8.0.13 で導入された `INFORMATION_SCHEMA.INNODB_TABLESPACES` テーブルには、暗号化されたテーブルスペースの識別に使用できる `ENCRYPTION` カラムが含まれています。

```
mysql> SELECT SPACE, NAME, SPACE_TYPE, ENCRYPTION FROM INFORMATION_SCHEMA.INNODB_TABLESPACES
```

```

WHERE ENCRYPTION=Y\G
***** 1. row *****
SPACE: 4294967294
NAME: mysql
SPACE_TYPE: General
ENCRYPTION: Y
***** 2. row *****
SPACE: 2
NAME: test/t1
SPACE_TYPE: Single
ENCRYPTION: Y
***** 3. row *****
SPACE: 3
NAME: ts1
SPACE_TYPE: General
ENCRYPTION: Y

```

CREATE TABLE または ALTER TABLE ステートメントで ENCRYPTION オプションが指定されている場合、INFORMATION_SCHEMA.TABLES の CREATE_OPTIONS カラムに記録されます。このカラムをクエリーすると、暗号化された file-per-table テーブルスペースに存在するテーブルを識別できます。

```

mysql> SELECT TABLE_SCHEMA, TABLE_NAME, CREATE_OPTIONS FROM INFORMATION_SCHEMA.TABLES
WHERE CREATE_OPTIONS LIKE '%ENCRYPTION%';
+-----+-----+-----+
| TABLE_SCHEMA | TABLE_NAME | CREATE_OPTIONS |
+-----+-----+-----+
| test        | t1          | ENCRYPTION="Y" |
+-----+-----+-----+

```

INFORMATION_SCHEMA.INNODB_TABLESPACES をクエリーして、特定のスキーマおよびテーブルに関連付けられているテーブルスペースに関する情報を取得します。

```

mysql> SELECT SPACE, NAME, SPACE_TYPE FROM INFORMATION_SCHEMA.INNODB_TABLESPACES WHERE NAME='test/t1';
+-----+-----+-----+
| SPACE | NAME   | SPACE_TYPE |
+-----+-----+-----+
| 3     | test/t1 | Single     |
+-----+-----+-----+

```

INFORMATION_SCHEMA.SCHEMATA テーブルをクエリーすることで、暗号化対応のスキーマを識別できます。

```

mysql> SELECT SCHEMA_NAME, DEFAULT_ENCRYPTION FROM INFORMATION_SCHEMA.SCHEMATA
WHERE DEFAULT_ENCRYPTION='YES';
+-----+-----+
| SCHEMA_NAME | DEFAULT_ENCRYPTION |
+-----+-----+
| test        | YES                 |
+-----+-----+

```

SHOW CREATE SCHEMA には、DEFAULT ENCRYPTION 句も表示されます。

暗号化の進行状況の監視

Performance Schema を使用して、一般的なテーブルスペースおよび mysql システムテーブルスペースの暗号化の進行状況を監視できます。

stage/innodb/alter tablespace (encryption) ステージイベントインストゥルメントは、一般的なテーブルスペース暗号化操作に関する WORK_ESTIMATED および WORK_COMPLETED の情報をレポートします。

次の例は、stage/innodb/alter tablespace (encryption) ステージイベントインストゥルメントおよび関連するコンシューマテーブルを有効にして、一般的なテーブルスペースまたは mysql システムテーブルスペースの暗号化の進行状況を監視する方法を示しています。パフォーマンススキーマステージイベントインストゥルメントおよび関連コンシューマについては、[セクション27.12.5「パフォーマンススキーマステージイベントテーブル」](#)を参照してください。

1. stage/innodb/alter tablespace (encryption) インストゥルメントを有効にします:

```
mysql> USE performance_schema;
```



```
mysql> UPDATE setup_instruments SET ENABLED = 'YES'
WHERE NAME LIKE 'stage/innodb/alter tablespace (encryption)';
```

2. ステージイベントコンシューマテーブル (`events_stages_current`、`events_stages_history` および `events_stages_history_long` を含む) を有効にします。

```
mysql> UPDATE setup_consumers SET ENABLED = 'YES' WHERE NAME LIKE '%stages%';
```

3. テーブルスペース暗号化操作を実行します。この例では、`ts1` という一般的なテーブルスペースが暗号化されません。

```
mysql> ALTER TABLESPACE ts1 ENCRYPTION = 'Y';
```

4. パフォーマンススキーマ `events_stages_current` テーブルをクエリーして、暗号化操作の進行状況を確認します。`WORK_ESTIMATED` では、テーブルスペース内のページの合計数がレポートされます。`WORK_COMPLETED` では、処理されたページ数がレポートされます。

```
mysql> SELECT EVENT_NAME, WORK_ESTIMATED, WORK_COMPLETED FROM events_stages_current;
+-----+-----+-----+
| EVENT_NAME                | WORK_COMPLETED | WORK_ESTIMATED |
+-----+-----+-----+
| stage/innodb/alter tablespace (encryption) |          1056 |          1407 |
+-----+-----+-----+
```

暗号化操作が完了すると、`events_stages_current` テーブルは空のセットを返します。この場合、`events_stages_history` テーブルをチェックして、完了した操作のイベントデータを表示できます。例:

```
mysql> SELECT EVENT_NAME, WORK_COMPLETED, WORK_ESTIMATED FROM events_stages_history;
+-----+-----+-----+
| EVENT_NAME                | WORK_COMPLETED | WORK_ESTIMATED |
+-----+-----+-----+
| stage/innodb/alter tablespace (encryption) |          1407 |          1407 |
+-----+-----+-----+
```

暗号化の使用上のノート

- **ENCRYPTION** オプションを使用して既存の file-per-table テーブルスペースを変更する場合は、適切に計画します。file-per-table テーブルスペースに存在するテーブルは、**COPY** アルゴリズムを使用して再構築されます。**INPLACE** アルゴリズムは、一般テーブルスペースまたは `mysql` システムテーブルスペースの **ENCRYPTION** 属性を変更するときに使用されます。**INPLACE** アルゴリズムでは、一般テーブルスペースに存在するテーブルに対する同時 DML が許可されます。同時 DDL はブロックされます。
- 一般テーブルスペースまたは `mysql` システムテーブルスペースが暗号化されると、テーブルスペースに存在するすべてのテーブルが暗号化されます。同様に、暗号化されたテーブルスペースに作成されたテーブルも暗号化されます。
- 通常の操作中にサーバーが終了または停止した場合は、以前に構成したものと同じ暗号化設定を使用してサーバーを再起動することをお勧めします。
- 最初のマスター暗号化キーは、最初の新規または既存のテーブルスペースが暗号化されるときに生成されます。
- マスターキーローテーションでは、テーブルスペースキーは再暗号化されますが、テーブルスペースキー自体は変更されません。テーブルスペースキーを変更するには、暗号化を無効にして再度有効にする必要があります。file-per-table テーブルスペースの場合、テーブルスペースの再暗号化はテーブルを再構築する **ALGORITHM=COPY** 操作です。一般テーブルスペースおよび `mysql` システムテーブルスペースの場合、これは **ALGORITHM=INPLACE** 操作であり、テーブルスペースに存在するテーブルを再構築する必要はありません。
- **COMPRESSION** オプションと **ENCRYPTION** オプションの両方を使用してテーブルが作成された場合、圧縮はテーブルスペースデータが暗号化される前に実行されます。
- キーリングデータファイル (`keyring_file_data` または `keyring_encrypted_file_data` で指定されたファイル) が空であるか欠落している場合、**ALTER INSTANCE ROTATE INNODB MASTER KEY** の最初の実行でマスター暗号化キーが作成されます。
- `keyring_file` または `keyring_encrypted_file` プラグインをアンインストールしても、既存のキーリングデータファイルは削除されません。

- ・ キーリングデータファイルは、テーブルスペースデータファイルと同じディレクトリに配置しないことをお勧めします。
- ・ 実行時またはサーバーの再起動時に `keyring_file_data` または `keyring_encrypted_file_data` の設定を変更すると、以前に暗号化されたテーブルスペースにアクセスできなくなり、データが失われる可能性があります。
- ・ 暗号化は、`FULLTEXT` インデックスの追加時に暗黙的に作成される `InnoDB FULLTEXT` インデックステーブルでサポートされますが、暗号化された一般テーブルスペースに存在するテーブルに `FULLTEXT` インデックスが作成される場合にのみサポートされます。この場合、`FULLTEXT` インデックステーブルは、同じ暗号化された一般テーブルスペースに作成されます。関連情報については、[InnoDB 全文インデックステーブル](#) を参照してください。

暗号化の制限事項

- ・ Advanced Encryption Standard (AES) は、サポートされる唯一の暗号化アルゴリズムです。 `InnoDB` テーブルスペース暗号化では、テーブルスペースキー暗号化に電子コードブック (ECB) ブロック暗号化モードを使用し、データ暗号化に暗号ブロックチェーン (CBC) ブロック暗号化モードを使用します。パディングは CBC ブロック暗号化モードでは使用されません。かわりに、`InnoDB` は暗号化されるテキストがブロックサイズの倍数であることを確認します。
- ・ 暗号化は、`file-per-table` テーブルスペース、`general` テーブルスペースおよび `mysql` システムテーブルスペースでのみサポートされます。一般テーブルスペースの暗号化サポートは、MySQL 8.0.13 で導入されました。 `mysql` システムテーブルスペースの暗号化サポートは、MySQL 8.0.16 の時点で使用できます。暗号化は、`InnoDB system tablespace` を含む他のテーブルスペースタイプではサポートされていません。
- ・ 暗号化された `file-per-table` テーブルスペース、`general` テーブルスペースまたは `mysql` システムテーブルスペースから、暗号化をサポートしないテーブルスペースタイプにテーブルを移動またはコピーすることはできません。
- ・ 暗号化されたテーブルスペースから暗号化されていないテーブルスペースにテーブルを移動またはコピーすることはできません。ただし、暗号化されていないテーブルスペースから暗号化されたテーブルスペースへのテーブルの移動は許可されています。たとえば、暗号化されていない `file-per-table` または `general` テーブルスペースから暗号化された一般テーブルスペースにテーブルを移動またはコピーできます。
- ・ デフォルトでは、テーブルスペースの暗号化はテーブルスペースのデータにのみ適用されます。redo ログおよび undo ログデータは、`innodb_redo_log_encrypt` および `innodb_undo_log_encrypt` を有効にすることで暗号化できます。redo ログの暗号化および undo ログの暗号化を参照してください。バイナリログファイルとリレーログファイルの暗号化については、[セクション17.3.2「バイナリログファイルとリレーログファイルの暗号化」](#) を参照してください。
- ・ 暗号化されたテーブルスペースに存在する、または以前に存在していたテーブルのストレージエンジンを変更することはできません。

15.14 InnoDB の起動オプションおよびシステム変数

- ・ true または false であるシステム変数は、サーバー起動時に変数の名前を指定することで有効にすることができ、`--skip-` プリフィクスを使用することで無効にすることができます。たとえば、`InnoDB` 適応ハッシュインデックスを有効または無効にするには、コマンドラインで `--innodb-adaptive-hash-index` または `--skip-innodb-adaptive-hash-index` を使用するか、オプションファイルで `innodb_adaptive_hash_index` または `skip_innodb_adaptive_hash_index` を使用します。
- ・ 数値が指定されるシステム変数は、コマンド行で `--var_name=value` として指定するか、オプションファイルで `var_name=value` として指定できます。
- ・ 多くのシステム変数は、実行時に変更できます ([セクション5.1.9.2「動的システム変数」](#) を参照してください)。
- ・ `GLOBAL` および `SESSION` 変数スコープ修飾子については、[SET ステートメントのドキュメント](#) を参照してください。
- ・ 特定のオプションでは、`InnoDB` データファイルの場所およびレイアウトが制御されます。[セクション15.8.1「InnoDB の起動構成」](#) では、これらのオプションを使用する方法について説明します。
- ・ 初期段階では使用しないような一部のオプションは、マシンの処理能力やデータベースのワークロードに基づいて、`InnoDB` のパフォーマンス特性を調整する際に役立ちます。

- オプションおよびシステム変数の指定に関する詳細は、[セクション4.2.2「プログラムオプションの指定」](#)を参照してください。

表 15.24 「InnoDB オプションおよび変数リファレンス」

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
daemon_memcached_enable_binlog	はい	はい	はい		グローバル	いいえ
daemon_memcached_engine_lib_path	はい	はい	はい		グローバル	いいえ
daemon_memcached_engine_lib_path	はい	はい	はい		グローバル	いいえ
daemon_memcached_option	はい	はい	はい		グローバル	いいえ
daemon_memcached_r_batch_size	はい	はい	はい		グローバル	いいえ
daemon_memcached_w_batch_size	はい	はい	はい		グローバル	いいえ
foreign_key_checks			はい		両方	はい
innodb	はい	はい				
innodb_adaptive_flushing	はい	はい	はい		グローバル	はい
innodb_adaptive_flushing_lwm	はい	はい	はい		グローバル	はい
innodb_adaptive_hash_index	はい	はい	はい		グローバル	はい
innodb_adaptive_hash_index_parts	はい	はい	はい		グローバル	いいえ
innodb_adaptive_mutex_sleep_delay	はい	はい	はい		グローバル	はい
innodb_api_bk_commit_interval	はい	はい	はい		グローバル	はい
innodb_api_disable_rowlock	はい	はい	はい		グローバル	いいえ
innodb_api_enable_binlog	はい	はい	はい		グローバル	いいえ
innodb_api_enable_mdll	はい	はい	はい		グローバル	いいえ
innodb_api_trx_level	はい	はい	はい		グローバル	はい
innodb_autoextend_increment	はい	はい	はい		グローバル	はい
innodb_autoinc_lock_mode	はい	はい	はい		グローバル	いいえ
innodb_background_drop_list_empty	はい	はい	はい		グローバル	はい
Innodb_buffer_pool_bytes_data				はい	グローバル	いいえ
Innodb_buffer_pool_bytes_dirty				はい	グローバル	いいえ
innodb_buffer_pool_chunk_size	はい	はい	はい		グローバル	いいえ
innodb_buffer_pool_debug	はい	はい	はい		グローバル	いいえ
innodb_buffer_pool_dump_at_shutdown	はい	はい	はい		グローバル	はい
innodb_buffer_pool_dump_now	はい	はい	はい		グローバル	はい
innodb_buffer_pool_dump_pct	はい	はい	はい		グローバル	はい
Innodb_buffer_pool_dump_status				はい	グローバル	いいえ
innodb_buffer_pool_filename	はい	はい	はい		グローバル	はい
innodb_buffer_pool_in_core_file	はい	はい	はい		グローバル	はい
innodb_buffer_pool_instances	はい	はい	はい		グローバル	いいえ
innodb_buffer_pool_load_abort	はい	はい	はい		グローバル	はい
innodb_buffer_pool_load_at_startup	はい	はい	はい		グローバル	いいえ
innodb_buffer_pool_load_now	はい	はい	はい		グローバル	はい
Innodb_buffer_pool_load_status				はい	グローバル	いいえ
Innodb_buffer_pool_pages_data				はい	グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
InnoDB_buffer_pool_pages_dirty				はい	グローバル	いいえ
InnoDB_buffer_pool_pages_flushed				はい	グローバル	いいえ
InnoDB_buffer_pool_pages_free				はい	グローバル	いいえ
InnoDB_buffer_pool_pages_latched				はい	グローバル	いいえ
InnoDB_buffer_pool_pages_misc				はい	グローバル	いいえ
InnoDB_buffer_pool_pages_total				はい	グローバル	いいえ
InnoDB_buffer_pool_read_ahead				はい	グローバル	いいえ
InnoDB_buffer_pool_read_ahead_evicted				はい	グローバル	いいえ
InnoDB_buffer_pool_read_ahead_rnd				はい	グローバル	いいえ
InnoDB_buffer_pool_read_requests				はい	グローバル	いいえ
InnoDB_buffer_pool_reads				はい	グローバル	いいえ
InnoDB_buffer_pool_resize_status				はい	グローバル	いいえ
innodb_buffer_pool_resize	はい	はい	はい		グローバル	はい
InnoDB_buffer_pool_wait_free				はい	グローバル	いいえ
InnoDB_buffer_pool_write_requests				はい	グローバル	いいえ
innodb_change_buffer_max_size	はい	はい	はい		グローバル	はい
innodb_change_buffering	はい	はい	はい		グローバル	はい
innodb_change_buffering_debug	はい	はい	はい		グローバル	はい
innodb_checkpoint_disabled	はい	はい	はい		グローバル	はい
innodb_checksum_algorithm	はい	はい	はい		グローバル	はい
innodb_cmp_per_index_enabled	はい	はい	はい		グローバル	はい
innodb_commit_ordering	はい	はい	はい		グローバル	はい
innodb_compression_debug	はい	はい	はい		グローバル	はい
innodb_compression_failure_threshold_pct	はい	はい	はい		グローバル	はい
innodb_compression_level	はい	はい	はい		グローバル	はい
innodb_compression_pad_pct_max	はい	はい	はい		グローバル	はい
innodb_concurrency_tickets	はい	はい	はい		グローバル	はい
innodb_data_file_path	はい	はい	はい		グローバル	いいえ
InnoDB_data_fsyncs				はい	グローバル	いいえ
innodb_data_home_dir	はい	はい	はい		グローバル	いいえ
InnoDB_data_pending_fsyncs				はい	グローバル	いいえ
InnoDB_data_pending_reads				はい	グローバル	いいえ
InnoDB_data_pending_writes				はい	グローバル	いいえ
InnoDB_data_read				はい	グローバル	いいえ
InnoDB_data_reads				はい	グローバル	いいえ
InnoDB_data_writes				はい	グローバル	いいえ
InnoDB_data_written				はい	グローバル	いいえ
InnoDB_dblwr_pages_written				はい	グローバル	いいえ
InnoDB_dblwr_writes				はい	グローバル	いいえ
innodb_ddl_log_reset_debug	はい	はい	はい		グローバル	はい

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
innodb_deadlock_detect	はい	はい	はい		グローバル	はい
innodb_dedicated_server	はい	はい	はい		グローバル	いいえ
innodb_default_row_format	はい	はい	はい		グローバル	はい
innodb_directories	はい	はい	はい		グローバル	いいえ
innodb_disable_sort_file_cache	はい	はい	はい		グローバル	はい
innodb_doublewrite	はい	はい	はい		グローバル	いいえ
innodb_doublewrite_batch_size	はい	はい	はい		グローバル	いいえ
innodb_doublewrite_dir	はい	はい	はい		グローバル	いいえ
innodb_doublewrite_files	はい	はい	はい		グローバル	いいえ
innodb_doublewrite_pages	はい	はい	はい		グローバル	いいえ
innodb_fast_shutdown	はい	はい	はい		グローバル	はい
innodb_file_make_page_dirty_debug	はい	はい	はい		グローバル	はい
innodb_file_per_table	はい	はい	はい		グローバル	はい
innodb_fill_factor	はい	はい	はい		グローバル	はい
innodb_flush_log_timeout	はい	はい	はい		グローバル	はい
innodb_flush_log_trx_commit	はい	はい	はい		グローバル	はい
innodb_flush_method	はい	はい	はい		グローバル	いいえ
innodb_flush_neighbors	はい	はい	はい		グローバル	はい
innodb_flush_sync	はい	はい	はい		グローバル	はい
innodb_flushing_loops	はい	はい	はい		グローバル	はい
innodb_force_load_corrupted	はい	はい	はい		グローバル	いいえ
innodb_force_recovery	はい	はい	はい		グローバル	いいえ
innodb_fsync_threshold	はい	はい	はい		グローバル	はい
innodb_ft_aux_table			はい		グローバル	はい
innodb_ft_cache_size	はい	はい	はい		グローバル	いいえ
innodb_ft_enable_big_print	はい	はい	はい		グローバル	はい
innodb_ft_enable_stopword	はい	はい	はい		両方	はい
innodb_ft_max_token_size	はい	はい	はい		グローバル	いいえ
innodb_ft_min_token_size	はい	はい	はい		グローバル	いいえ
innodb_ft_num_wdopt	はい	はい	はい		グローバル	はい
innodb_ft_result_cache_limit	はい	はい	はい		グローバル	はい
innodb_ft_server_stopword_table	はい	はい	はい		グローバル	はい
innodb_ft_sort_pll_degree	はい	はい	はい		グローバル	いいえ
innodb_ft_total_cache_size	はい	はい	はい		グローバル	いいえ
innodb_ft_user_stopword_table	はい	はい	はい		両方	はい
InnoDB_have_atomic_builtins				はい	グローバル	いいえ
innodb_idle_flush_pct	はい	はい	はい		グローバル	はい
innodb_io_capacity	はい	はい	はい		グローバル	はい
innodb_io_capacity_max	はい	はい	はい		グローバル	はい
innodb_limit_optimizations	はい	はい	はい		グローバル	はい

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
innodb_lock_wait_timeout	はい	はい	はい		両方	はい
innodb_log_buffer_size	はい	はい	はい		グローバル	異なる
innodb_log_checkpoint_fuzzy_nowait	はい	はい	はい		グローバル	はい
innodb_log_checkpoint_nowait	はい	はい	はい		グローバル	はい
innodb_log_checkpoint_syncs	はい	はい	はい		グローバル	はい
innodb_log_compressed_pages	はい	はい	はい		グローバル	はい
innodb_log_file_size	はい	はい	はい		グローバル	いいえ
innodb_log_files_in_group	はい	はい	はい		グローバル	いいえ
innodb_log_group_home_dir	はい	はい	はい		グローバル	いいえ
innodb_log_spin_abs_lwm	はい	はい	はい		グローバル	はい
innodb_log_spin_pct_hwm	はい	はい	はい		グローバル	はい
innodb_log_wait_flush_spin_hwm	はい	はい	はい		グローバル	はい
InnoDB_log_waits				はい	グローバル	いいえ
innodb_log_write_ahead_size	はい	はい	はい		グローバル	はい
InnoDB_log_write_requests				はい	グローバル	いいえ
innodb_log_write_ahead_threads	はい	はい	はい		グローバル	はい
InnoDB_log_writes				はい	グローバル	いいえ
innodb_lru_scan_depth	はい	はい	はい		グローバル	はい
innodb_max_dirty_pages_pct	はい	はい	はい		グローバル	はい
innodb_max_dirty_pages_pct_lwm	はい	はい	はい		グローバル	はい
innodb_max_purge_lag	はい	はい	はい		グローバル	はい
innodb_max_purge_lag_delay	はい	はい	はい		グローバル	はい
innodb_max_undo_log_size	はい	はい	はい		グローバル	はい
innodb_merge_threads_hold_set_all_disabled	はい	はい	はい		グローバル	はい
innodb_monitor_disable	はい	はい	はい		グローバル	はい
innodb_monitor_enable	はい	はい	はい		グローバル	はい
innodb_monitor_reset	はい	はい	はい		グローバル	はい
innodb_monitor_reset_all	はい	はい	はい		グローバル	はい
InnoDB_num_open_files				はい	グローバル	いいえ
innodb_numa_interleave	はい	はい	はい		グローバル	いいえ
innodb_old_blocks_pct	はい	はい	はい		グローバル	はい
innodb_old_blocks_time	はい	はい	はい		グローバル	はい
innodb_online_alter_log_max_size	はい	はい	はい		グローバル	はい
innodb_open_files	はい	はい	はい		グローバル	いいえ
innodb_optimize_text_only	はい	はい	はい		グローバル	はい
InnoDB_os_log_fsyncs				はい	グローバル	いいえ
InnoDB_os_log_pending_fsyncs				はい	グローバル	いいえ
InnoDB_os_log_pending_writes				はい	グローバル	いいえ
InnoDB_os_log_written				はい	グローバル	いいえ
innodb_page_cleaners	はい	はい	はい		グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
InnoDB_page_size				はい	グローバル	いいえ
innodb_page_size	はい	はい	はい		グローバル	いいえ
InnoDB_pages_created				はい	グローバル	いいえ
InnoDB_pages_read				はい	グローバル	いいえ
InnoDB_pages_written				はい	グローバル	いいえ
innodb_parallel_read_threads	はい	はい	はい		セッション	はい
innodb_print_all_deadlocks	はい	はい	はい		グローバル	はい
innodb_print_ddl_logs	はい	はい	はい		グローバル	はい
innodb_purge_batch_size	はい	はい	はい		グローバル	はい
innodb_purge_reuse_oldest_truncate_frequency	はい	はい	はい		グローバル	はい
innodb_purge_threads	はい	はい	はい		グローバル	いいえ
innodb_random_read_ahead	はい	はい	はい		グローバル	はい
innodb_read_ahead_threshold	はい	はい	はい		グローバル	はい
innodb_read_io_threads	はい	はい	はい		グローバル	いいえ
innodb_read_only	はい	はい	はい		グローバル	いいえ
innodb_redo_log_archive_dirs	はい	はい	はい		グローバル	はい
InnoDB_redo_log_enabled				はい	グローバル	いいえ
innodb_redo_log_encrypt	はい	はい	はい		グローバル	はい
innodb_replication_delay	はい	はい	はい		グローバル	はい
innodb_rollback_timeout	はい	はい	はい		グローバル	いいえ
innodb_rollback_segments	はい	はい	はい		グローバル	はい
InnoDB_row_lock_current_waits				はい	グローバル	いいえ
InnoDB_row_lock_time				はい	グローバル	いいえ
InnoDB_row_lock_time_avg				はい	グローバル	いいえ
InnoDB_row_lock_time_max				はい	グローバル	いいえ
InnoDB_row_lock_waits				はい	グローバル	いいえ
InnoDB_rows_deleted				はい	グローバル	いいえ
InnoDB_rows_inserted				はい	グローバル	いいえ
InnoDB_rows_read				はい	グローバル	いいえ
InnoDB_rows_updated				はい	グローバル	いいえ
innodb_saved_page_number_debug	はい	はい	はい		グローバル	はい
innodb_sort_buffer_size	はい	はい	はい		グローバル	いいえ
innodb_spin_wait_delay	はい	はい	はい		グローバル	はい
innodb_spin_wait_noop_multiplier	はい	はい	はい		グローバル	はい
innodb_stats_auto_calc	はい	はい	はい		グローバル	はい
innodb_stats_include_delete_marked	はい	はい	はい		グローバル	はい
innodb_stats_method	はい	はい	はい		グローバル	はい
innodb_stats_on_metadata	はい	はい	はい		グローバル	はい
innodb_stats_percent	はい	はい	はい		グローバル	はい
innodb_stats_percent_sample_pages	はい	はい	はい		グローバル	はい

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
<code>innodb_stats_truncate</code>	<code>innodb_stats_truncate_sample_pages</code>	はい	はい		グローバル	はい
<code>innodb-status-file</code>	はい	はい				
<code>innodb_status_output</code>	はい	はい	はい		グローバル	はい
<code>innodb_status_output_locks</code>	はい	はい	はい		グローバル	はい
<code>innodb_strict_mode</code>	はい	はい	はい		両方	はい
<code>innodb_sync_array_size</code>	はい	はい	はい		グローバル	いいえ
<code>innodb_sync_debug</code>	はい	はい	はい		グローバル	いいえ
<code>innodb_sync_spin_loops</code>	はい	はい	はい		グローバル	はい
<code>InnoDB_system_rows_deleted</code>				はい	グローバル	いいえ
<code>InnoDB_system_rows_inserted</code>				はい	グローバル	いいえ
<code>InnoDB_system_rows_read</code>				はい	グローバル	いいえ
<code>innodb_table_locks</code>	はい	はい	はい		両方	はい
<code>innodb_temp_data_home_path</code>	はい	はい	はい		グローバル	いいえ
<code>innodb_temp_tablespace_dir</code>	はい	はい	はい		グローバル	いいえ
<code>innodb_thread_concurrency</code>	はい	はい	はい		グローバル	はい
<code>innodb_thread_sleep_delay</code>	はい	はい	はい		グローバル	はい
<code>innodb_tmpdir</code>	はい	はい	はい		両方	はい
<code>InnoDB_truncated_status_writes</code>				はい	グローバル	いいえ
<code>innodb_trx_purge_new_update_order</code>	はい	はい	はい		グローバル	はい
<code>innodb_trx_rseg_slots_debug</code>	はい	はい	はい		グローバル	はい
<code>innodb_undo_directory</code>	はい	はい	はい		グローバル	いいえ
<code>innodb_undo_log_encrypt</code>	はい	はい	はい		グローバル	はい
<code>innodb_undo_log_truncate</code>	はい	はい	はい		グローバル	はい
<code>innodb_undo_tablespace</code>	はい	はい	はい		グローバル	異なる
<code>InnoDB_undo_tablespace_active</code>				はい	グローバル	いいえ
<code>InnoDB_undo_tablespace_explicit</code>				はい	グローバル	いいえ
<code>InnoDB_undo_tablespace_implicit</code>				はい	グローバル	いいえ
<code>InnoDB_undo_tablespace_total</code>				はい	グローバル	いいえ
<code>innodb_use_native_aio</code>	はい	はい	はい		グローバル	いいえ
<code>innodb_validate_innodb_space_paths</code>	はい	はい	はい		グローバル	いいえ
<code>innodb_version</code>			はい		グローバル	いいえ
<code>innodb_write_io_threads</code>	はい	はい	はい		グローバル	いいえ
<code>unique_checks</code>			はい		両方	はい

InnoDB コマンドオプション

- `--innodb[=value]`

コマンド行形式	<code>--innodb[=value]</code>
非推奨	はい
型	列挙

デフォルト値	ON
有効な値	OFF ON FORCE

サーバーが InnoDB サポートでコンパイルされた場合に、InnoDB ストレージエンジンのロードを制御します。このオプションの形式はトライステートであり、指定可能な値は OFF、ON、または FORCE です。 [セクション 5.6.1「プラグインのインストールおよびアンインストール」](#) を参照してください。

InnoDB を無効にするには、`--innodb=OFF` または `--skip-innodb` を使用します。この場合、デフォルトのストレージエンジンは InnoDB であるため、`--default-storage-engine` および `--default-tmp-storage-engine` を使用して永続テーブルと TEMPORARY テーブルの両方のデフォルトをほかのエンジンに設定しないかぎり、サーバーは起動しません。

InnoDB ストレージエンジンを無効にすることはできなくなり、`--innodb=OFF` および `--skip-innodb` オプションは非推奨であり、効果はありません。使用すると警告が表示されます。これらのオプションは、将来の MySQL リリースで削除される予定です。

- `--innodb-status-file`

コマンド行形式	<code>--innodb-status-file[={OFF ON}]</code>
型	Boolean
デフォルト値	OFF

`--innodb-status-file` の起動オプションは、InnoDB が `innodb_status.pid` という名前のファイルをデータディレクトリに作成し、`SHOW ENGINE INNODB STATUS` 出力を 15 秒ごとにおよそ書き込むかどうかを制御します。

`innodb_status.pid` ファイルはデフォルトでは作成されません。これを作成するには、`--innodb-status-file` オプションを指定して `mysqld` を起動します。サーバーが正常に停止すると、InnoDB によってファイルが削除されます。異常停止が発生した場合は、ステータスファイルを手動で削除する必要がある場合があります。

`--innodb-status-file` オプションは一時的な使用を目的としています。これは、`SHOW ENGINE INNODB STATUS` の出力生成がパフォーマンスに影響し、`innodb_status.pid` ファイルが時間の経過とともに非常に大きくなる可能性があるためです。

関連情報については、[セクション 15.17.2「InnoDB モニターの有効化」](#) を参照してください。

- `--skip-innodb`

InnoDB ストレージエンジンを無効にします。 `--innodb` の説明を参照してください。

InnoDB システム変数

- `daemon_memcached_enable_binlog`

コマンド行形式	<code>--daemon-memcached-enable-binlog[={OFF ON}]</code>
システム変数	<code>daemon_memcached_enable_binlog</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

MySQL `binary log` で InnoDB `memcached` プラグイン (`daemon_memcached`) を使用するには、ソースサーバーでこのオプションを有効にします。このオプションは、サーバーの起動時にのみ設定できます。 `--log-bin` オプションを使用して、ソースサーバーで MySQL バイナリログを有効にする必要もあります。

詳細は、[セクション15.20.7「InnoDB memcached プラグインとレプリケーション」](#)を参照してください。

- `daemon_memcached_engine_lib_name`

コマンド行形式	<code>--daemon-memcached-engine-lib-name=file_name</code>
システム変数	<code>daemon_memcached_engine_lib_name</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	<code>innodb_engine.so</code>

InnoDB `memcached` プラグインを実装する共有ライブラリを指定します。

詳細は、[セクション15.20.3「InnoDB memcached プラグインの設定」](#)を参照してください。

- `daemon_memcached_engine_lib_path`

コマンド行形式	<code>--daemon-memcached-engine-lib-path=dir_name</code>
システム変数	<code>daemon_memcached_engine_lib_path</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ディレクトリ名
デフォルト値	NULL

InnoDB `memcached` プラグインを実装する共有ライブラリを含むディレクトリのパスです。デフォルト値は、MySQL プラグインディレクトリを表す NULL です。MySQL プラグインディレクトリの外部にある別のストレージエンジン用の `memcached` プラグインを指定しないかぎり、このパラメータを変更する必要はありません。

詳細は、[セクション15.20.3「InnoDB memcached プラグインの設定」](#)を参照してください。

- `daemon_memcached_option`

コマンド行形式	<code>--daemon-memcached-option=options</code>
システム変数	<code>daemon_memcached_option</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	

起動時に、空白文字で区切られた `memcached` オプションをベースとなる `memcached` メモリーオブジェクトのキャッシュデーモンに渡すために使用されます。たとえば、`memcached` がリスニングするポートの変更、同時接続の最大数の削減、キーと値のペアの最大メモリーサイズの変更、またはエラーログのデバッグメッセージの有効化を行うことができます。

使用法の詳細は、[セクション15.20.3「InnoDB memcached プラグインの設定」](#)を参照してください。`memcached` オプションについては、`memcached` のマニュアルページを参照してください。

- [daemon_memcached_r_batch_size](#)

コマンド行形式	<code>--daemon-memcached-r-batch-size=#</code>
システム変数	<code>daemon_memcached_r_batch_size</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1

COMMIT を実行して新しいトランザクションを開始する前に実行する `memcached` 読み取り操作 (`get` 操作) の数を指定します。 `daemon_memcached_w_batch_size` の対の片方です。

この値はデフォルトで 1 に設定されているため、SQL ステートメントを介してテーブルに加えられた変更は、`memcached` 操作からすぐに参照できます。ベースとなるテーブルが `memcached` インタフェースからのみアクセスされているシステム上で、頻繁なコミットによるオーバーヘッドを削減するために、これを大きくすることがあります。大きすぎる値を設定すると、Undo データまたは Redo データの量によっては、長時間実行されるトランザクションの場合と同様に、一部のストレージでオーバーヘッドが発生する可能性があります。

詳細は、[セクション15.20.3「InnoDB memcached プラグインの設定」](#)を参照してください。

- [daemon_memcached_w_batch_size](#)

コマンド行形式	<code>--daemon-memcached-w-batch-size=#</code>
システム変数	<code>daemon_memcached_w_batch_size</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1

COMMIT を実行して新しいトランザクションを開始する前に実行する、`add`、`set`、`incr` などの `memcached` 書き込み操作の数を指定します。 `daemon_memcached_r_batch_size` の対の一方です。

格納されるデータは停止時に保持することが重要であり、すぐにコミットする必要があると仮定すると、この値はデフォルトで 1 に設定されます。クリティカルでないデータを格納する場合、頻繁なコミットによるオーバーヘッドを削減するためにこの値を増やすことができますが、予期しない終了が発生すると、最後の `N-1` のコミットされていない書き込み操作が失われる可能性があります。

詳細は、[セクション15.20.3「InnoDB memcached プラグインの設定」](#)を参照してください。

- [innodb_adaptive_flushing](#)

コマンド行形式	<code>--innodb-adaptive-flushing[={OFF ON}]</code>
システム変数	<code>innodb_adaptive_flushing</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean

デフォルト値	ON
--------	----

ワークロードに基づいて、[InnoDB バッファプール](#)内の**ダーティーページ**をフラッシュする比率を動的に調整するかどうかを指定します。フラッシュ比率を動的に調整する目的は、I/O アクティビティーのバーストを回避することです。この設定はデフォルトで有効になっています。詳しくは[セクション15.8.3.5「バッファプールのフラッシュの構成」](#)をご覧ください。一般的な I/O チューニングのアドバイスについては、[セクション8.5.8「InnoDB ディスク I/O の最適化」](#)を参照してください。

- [innodb_adaptive_flushing_lwm](#)

コマンド行形式	<code>--innodb-adaptive-flushing-lwm=#</code>
システム変数	innodb_adaptive_flushing_lwm
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	10
最小値	0
最大値	70

[adaptive flushing](#) が有効な [redo log](#) 容量の割合を表す最低水位標を定義します。詳細は、[セクション15.8.3.5「バッファプールのフラッシュの構成」](#)を参照してください。

- [innodb_adaptive_hash_index](#)

コマンド行形式	<code>--innodb-adaptive-hash-index[={OFF ON}]</code>
システム変数	innodb_adaptive_hash_index
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

InnoDB [適応型ハッシュインデックス](#)が有効と無効のどちらになっているのかを示します。ワークロードに応じて、[適応型ハッシュインデックスの作成](#)を動的に有効または無効にして、クエリーのパフォーマンスを改善することが望ましい場合があります。適応型ハッシュインデックスがすべてのワークロードに役立つとは限らないため、現実的なワークロードを使用して、有効と無効の両方でベンチマークを実施してください。詳細は、[セクション15.5.3「適応型ハッシュインデックス」](#)を参照してください。

この変数はデフォルトで有効になっています。SET GLOBAL ステートメントを使用すると、サーバーを再起動せずに、このパラメータを変更できます。実行時に設定を変更するには、グローバルシステム変数を設定するのに十分な権限が必要です。[セクション5.1.9.1「システム変数権限」](#)を参照してください。サーバーの起動時に `--skip-innodb-adaptive-hash-index` を使用して無効にすることもできます。

適応型ハッシュインデックスを無効にすると、すぐにハッシュテーブルが空になります。ハッシュテーブルが空になっても通常の操作は続行でき、ハッシュテーブルを使用していた実行中のクエリーは、代わりにインデックスの B ツリーに直接アクセスします。適応型ハッシュインデックスを再度有効にすると、通常の操作時にハッシュテーブルが再度移入されます。

- [innodb_adaptive_hash_index_parts](#)

コマンド行形式	<code>--innodb-adaptive-hash-index-parts=#</code>	
システム変数	innodb_adaptive_hash_index_parts	2823

スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	数値
デフォルト値	8
最小値	1
最大値	512

適応ハッシュインデックス検索システムをパーティション化します。各インデックスは特定のパーティションにバインドされ、各パーティションは個別のラッチで保護されます。

適応ハッシュインデックス検索システムは、デフォルトで 8 つの部分にパーティション化されています。最大設定は 512 です。

関連情報については、[セクション15.5.3「適応型ハッシュインデックス」](#)を参照してください。

- [innodb_adaptive_max_sleep_delay](#)

コマンド行形式	<code>--innodb-adaptive-max-sleep-delay=#</code>
システム変数	innodb_adaptive_max_sleep_delay
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	150000
最小値	0
最大値	1000000

InnoDB で、現在のワークロードに応じて [innodb_thread_sleep_delay](#) の値を自動的に調整できます。ゼロ以外の値を指定すると、[innodb_adaptive_max_sleep_delay](#) オプションで指定した最大値まで、[innodb_thread_sleep_delay](#) 値の動的な自動調整が可能になります。値はマイクロ秒数を表しています。このオプションは、InnoDB スレッド数が 16 個を上回る高負荷のシステムで役立つことがあります。(実際には、同時接続数が数百または数千になる MySQL システムの大部分の変数です。)

詳細は、[セクション15.8.4「InnoDB のスレッド並列性の構成」](#)を参照してください。

- [innodb_api_bk_commit_interval](#)

コマンド行形式	<code>--innodb-api-bk-commit-interval=#</code>
システム変数	innodb_api_bk_commit_interval
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	5
最小値	1
最大値	1073741824

InnoDB memcached インタフェースが使用されるアイドル状態の接続が自動コミットされる頻度 (秒単位) です。詳細は、[セクション15.20.6.4「InnoDB memcached プラグインのトランザクション動作の制御」](#)を参照してください。

- innodb_api_disable_rowlock

コマンド行形式	<code>--innodb-api-disable-rowlock[={OFF ON}]</code>
システム変数	<code>innodb_api_disable_rowlock</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

InnoDB memcached が DML 操作を実行するときに行ロックを無効にするには、このオプションを使用します。デフォルトでは、`innodb_api_disable_rowlock` は無効になっています。これは、`memcached` が `get` および `set` 操作の行ロックを要求することを意味します。`innodb_api_disable_rowlock` が有効な場合、`memcached` は行ロックではなくテーブルロックを要求します。

`innodb_api_disable_rowlock` は動的ではありません。これは `mysqld` コマンド行で指定するか、または MySQL 構成ファイルに入力する必要があります。構成は、MySQL サーバーの起動時に発生するプラグインのインストール時に有効になります。

詳細は、[セクション15.20.6.4「InnoDB memcached プラグインのトランザクション動作の制御」](#)を参照してください。

- innodb_api_enable_binlog

コマンド行形式	<code>--innodb-api-enable-binlog[={OFF ON}]</code>
システム変数	<code>innodb_api_enable_binlog</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

MySQL バイナリログとともに、InnoDB memcached プラグインを使用できます。詳細は、[InnoDB memcached バイナリログの有効化](#)を参照してください。

- innodb_api_enable_md1

コマンド行形式	<code>--innodb-api-enable-md1[={OFF ON}]</code>
システム変数	<code>innodb_api_enable_md1</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

InnoDB memcached プラグインで使用されるテーブルをロックします。これにより、SQL インタフェースから DDL によって削除または変更できなくなります。詳細は、[セクション15.20.6.4「InnoDB memcached プラグインのトランザクション動作の制御」](#)を参照してください。

- innodb_api_trx_level

コマンド行形式	<code>--innodb-api-trx-level=#</code>	2825
---------	---------------------------------------	------

システム変数	innodb_api_trx_level
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0

[memcached](#) インタフェースによって処理されるクエリーのトランザクション [isolation level](#) を制御します。よく聞く名前に対応する定数は、次のとおりです。

- 0 = [READ UNCOMMITTED](#)
- 1 = [READ COMMITTED](#)
- 2 = [REPEATABLE READ](#)
- 3 = [SERIALIZABLE](#)

詳細は、[セクション15.20.6.4「InnoDB memcached プラグインのトランザクション動作の制御」](#)を参照してください。

- [innodb_autoextend_increment](#)

コマンド行形式	<code>--innodb-autoextend-increment=#</code>
システム変数	innodb_autoextend_increment
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	64
最小値	1
最大値	1000

自動拡張 [InnoDB system tablespace](#) ファイルがいっぱいになったときにサイズを拡張するための増分サイズ (MB)。デフォルト値は 64 です。関連情報については、[システムテーブルスペースデータファイル構成](#),および[システムテーブルスペースのサイズ変更](#)を参照してください。

[innodb_autoextend_increment](#) 設定は、[file-per-table](#) テーブルスペースファイルまたは [general tablespace](#) ファイルには影響しません。これらのファイルは、[innodb_autoextend_increment](#) の設定に関係なく自動拡張されます。拡張は少量で始まり、その後の拡張は増分が 4MB で発生します。

- [innodb_autoinc_lock_mode](#)

コマンド行形式	<code>--innodb-autoinc-lock-mode=#</code>
システム変数	innodb_autoinc_lock_mode
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	2
有効な値	0 1

自動インクリメント値を生成する際に使用される**ロックモード**です。許容値は、従来型、連続型またはインターリーブ型の場合、それぞれ 0、1 または 2 です。

デフォルト設定は、MySQL 8.0 の時点では 2 (インターリーブ)、それより前では 1 (連続) です。デフォルト設定としてインターリーブロックモードを変更すると、MySQL 5.7 で発生したデフォルトのレプリケーションタイプとして、ステートメントベースから行ベースのレプリケーションへの変更が反映されます。ステートメントベースレプリケーションでは、SQL ステートメントの特定のシーケンスに対して自動インクリメント値が予測可能かつ繰り返し可能な順序で割り当てられるように、連続した自動インクリメントロックモードが必要ですが、行ベースレプリケーションは SQL ステートメントの実行順序には影響しません。

各ロックモードの特性については、[InnoDB AUTO_INCREMENT のロックモード](#) を参照してください。

- [innodb_background_drop_list_empty](#)

コマンド行形式	<code>--innodb-background-drop-list-empty[={OFF ON}]</code>
システム変数	innodb_background_drop_list_empty
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

[innodb_background_drop_list_empty](#) デバッグオプションを有効にすると、バックグラウンドドロップリストが空になるまでテーブルの作成が遅延されるため、テストケースの失敗を回避できます。たとえば、テストケース A がテーブル `t1` をバックグラウンドドロップリストに配置する場合、テストケース B はバックグラウンドドロップリストが空になるまで待機してから、テーブル `t1` を作成します。

- [innodb_buffer_pool_chunk_size](#)

コマンド行形式	<code>--innodb-buffer-pool-chunk-size=#</code>
システム変数	innodb_buffer_pool_chunk_size
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	134217728
最小値	1048576
最大値	<code>innodb_buffer_pool_size / innodb_buffer_pool_instances</code>

[innodb_buffer_pool_chunk_size](#) は、InnoDB バッファプールのサイズ変更操作のチャンクサイズを定義します。

サイズ変更操作中にすべてのバッファプールページがコピーされないようにするために、この操作は「chunks」で実行されます。デフォルトでは、[innodb_buffer_pool_chunk_size](#) は 128MB (134217728 バイト) です。チャンクに含まれるページ数は、[innodb_page_size](#) の値によって異なります。[innodb_buffer_pool_chunk_size](#) は、1MB (1048576 バイト) 単位で増減できます。

[innodb_buffer_pool_chunk_size](#) 値を変更する場合は、次の条件が適用されます:

- バッファプールの初期化時に `innodb_buffer_pool_chunk_size*innodb_buffer_pool_instances` が現在のバッファプールサイズより大きい場合、[innodb_buffer_pool_chunk_size](#) は `innodb_buffer_pool_size / innodb_buffer_pool_instances` に切り捨てられます。

- バッファプールサイズは、常に `innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances` と等しいか倍数である必要があります。 `innodb_buffer_pool_chunk_size` を変更すると、 `innodb_buffer_pool_size` は `innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances` と等しいか倍数の値に自動的に丸められます。調整は、バッファプールが初期化されたときに行われます。

重要

この値を変更するとバッファプールのサイズが自動的に増加する可能性があるため、 `innodb_buffer_pool_chunk_size` を変更する場合は注意が必要です。 `innodb_buffer_pool_chunk_size` を変更する前に、 `innodb_buffer_pool_size` への影響を計算して、生成されるバッファプールサイズが受け入れ可能であることを確認します。

潜在的なパフォーマンスの問題を回避するには、チャンク (`innodb_buffer_pool_size / innodb_buffer_pool_chunk_size`) の数が 1000 を超えないようにする必要があります。

`innodb_buffer_pool_size` 変数は動的で、サーバーがオンラインのときにバッファプールのサイズを変更できます。ただし、バッファプールサイズは `innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances` の倍数である必要があり、これらの変数設定のいずれかを変更するにはサーバーを再起動する必要があります。

詳しくは [セクション15.8.3.1「InnoDB バッファプールサイズの構成」](#) をご覧ください。

- `innodb_buffer_pool_debug`

コマンド行形式	<code>--innodb-buffer-pool-debug[={OFF ON}]</code>
システム変数	<code>innodb_buffer_pool_debug</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

このオプションを有効にすると、バッファプールのサイズが 1GB 未満の場合に複数のバッファプールインスタンスが許可され、 `innodb_buffer_pool_instances` に設定されている 1GB の最小バッファプールサイズ制約は無視されます。 `innodb_buffer_pool_debug` オプションは、デバッグサポートが `WITH_DEBUG CMake` オプションを使用してコンパイルされている場合にのみ使用できます。

- `innodb_buffer_pool_dump_at_shutdown`

コマンド行形式	<code>--innodb-buffer-pool-dump-at-shutdown[={OFF ON}]</code>
システム変数	<code>innodb_buffer_pool_dump_at_shutdown</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

MySQL サーバーの停止時に `InnoDB buffer pool` にキャッシュされたページを記録して、次の再起動時に `warmup` プロセスを短縮するかどうかを指定します。一般に、 `innodb_buffer_pool_load_at_startup` と組み合わせて使用されます。 `innodb_buffer_pool_dump_pct` オプションは、ダンプする最後に使用されたバッファプールページの割合を定義します。

`innodb_buffer_pool_dump_at_shutdown` と `innodb_buffer_pool_load_at_startup` の両方がデフォルトで有効になっています。

詳細は、 [セクション15.8.3.6「バッファプールの状態の保存と復元」](#) を参照してください。

- innodb_buffer_pool_dump_now

コマンド行形式	<code>--innodb-buffer-pool-dump-now[={OFF ON}]</code>
システム変数	<code>innodb_buffer_pool_dump_now</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

InnoDB buffer pool にキャッシュされたページをすぐに記録します。一般に、`innodb_buffer_pool_load_now` と組み合わせて使用されます。

詳細は、[セクション15.8.3.6「バッファプールの状態の保存と復元」](#)を参照してください。

- innodb_buffer_pool_dump_pct

コマンド行形式	<code>--innodb-buffer-pool-dump-pct=#</code>
システム変数	<code>innodb_buffer_pool_dump_pct</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	25
最小値	1
最大値	100

各バッファプールが読み出してダンプする直近で使用されたページの割合を指定。指定できる範囲は 1 ~ 100 です。デフォルト値は 25 です。たとえば、100 ページのバッファプールが 4 つあり、`innodb_buffer_pool_dump_pct` が 25 に設定されている場合、各バッファプールから最近使用された 25 ページがダンプされます。

- innodb_buffer_pool_filename

コマンド行形式	<code>--innodb-buffer-pool-filename=file_name</code>
システム変数	<code>innodb_buffer_pool_filename</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	<code>ib_buffer_pool</code>

`innodb_buffer_pool_dump_at_shutdown` または `innodb_buffer_pool_dump_now` で生成されるテーブルスペース ID およびページ ID のリストを保持するファイルの名前を指定します。テーブルスペース ID およびページ ID

は、`space, page_id` という形式で保存されます。デフォルトでは、ファイルの名前は `ib_buffer_pool` で、InnoDB データディレクトリにあります。データディレクトリに対してデフォルト以外の場所を指定する必要があります。

SET ステートメントを使用して、実行時にファイル名を指定できます:

```
SET GLOBAL innodb_buffer_pool_filename='file_name';
```

起動時に、起動文字列または MySQL 構成ファイルでファイル名を指定することもできます。起動時にファイル名を指定する場合は、ファイルが存在する必要があります。存在しない場合は、そのようなファイルまたはディレクトリがないことを示す起動エラーが InnoDB によって返されます。

詳細は、[セクション15.8.3.6「バッファープールの状態の保存と復元」](#)を参照してください。

- [innodb_buffer_pool_in_core_file](#)

コマンド行形式	<code>--innodb-buffer-pool-in-core-file[={OFF ON}]</code>
導入	8.0.14
システム変数	innodb_buffer_pool_in_core_file
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

`innodb_buffer_pool_in_core_file` 変数を無効にすると、InnoDB バッファープールページが除外され、コアファイルのサイズが小さくなります。この変数を使用するには、`core_file` 変数を有効にし、オペレーティングシステムで `madvise()` に対する `MADV_DONTDUMP` の POSIX 以外の拡張機能をサポートする必要があります。これは Linux 3.4 以降でサポートされています。詳細は、[セクション15.8.3.7「コアファイルからのバッファープールページの除外」](#)を参照してください。

- [innodb_buffer_pool_instances](#)

コマンド行形式	<code>--innodb-buffer-pool-instances=#</code>
システム変数	innodb_buffer_pool_instances
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値 (Windows, 32 ビットプラットフォーム)	(autosized)
デフォルト値 (その他)	8 (or 1 if <code>innodb_buffer_pool_size < 1GB</code>)
最小値	1
最大値	64

InnoDB のバッファープールが分割される領域の数です。バッファープールが数 G バイトの範囲にあるシステムでは、バッファープールを個別のインスタンスに分割すると、キャッシュされたページに対して異なるスレッドが読み取りおよび書き込みを行うときの競合が減るため、並列性が向上する場合があります。バッファープールに格納される各ページまたはバッファープールから読み取られる各ページは、ハッシュ関数を使用して、バッファープールインスタンスのいずれかにランダムに割り当てられます。各バッファープールは、独自の空きリスト、フラッシュリスト、LRU、およびバッファープールに接続されたその他のすべてのデータ構造を管理し、独自のバッファープール相互排他ロックによって保護されます。

このオプションは、`innodb_buffer_pool_size` を 1GB 以上に設定する場合にのみ有効になります。バッファープールの合計サイズは、すべてのバッファープールに分割されます。最高の効率を得るには、`innodb_buffer_pool_instances`

と `innodb_buffer_pool_size` の組み合わせを、各バッファプールインスタンスが少なくとも 1G バイトになるように指定します。

32-bit Windows システムのデフォルト値は、次に説明するように、`innodb_buffer_pool_size` の値によって異なります:

- `innodb_buffer_pool_size` が 1.3G バイトよりも大きい場合は、`innodb_buffer_pool_instances` のデフォルトが `innodb_buffer_pool_size/128M` バイトになり、チャンクごとに個別のメモリー割り当てリクエストを持ちます。32 ビット版 Windows で単一のバッファプールで必要となる連続したアドレス空間を割り当てることができないという重大なリスクが存在する境界として、1.3G バイトが選択されました。
- それ以外の場合、デフォルトは 1 です。

他のすべてのプラットフォームでは、`innodb_buffer_pool_size` が 1GB 以上の場合、デフォルト値は 8 です。それ以外の場合、デフォルトは 1 です。

関連情報については、[セクション15.8.3.1「InnoDB バッファプールサイズの構成」](#)を参照してください。

- `innodb_buffer_pool_load_abort`

コマンド行形式	<code>--innodb-buffer-pool-load-abort[={OFF ON}]</code>
システム変数	<code>innodb_buffer_pool_load_abort</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

`innodb_buffer_pool_load_at_startup` または `innodb_buffer_pool_load_now` によってトリガーされた InnoDB buffer pool コンテンツをリストアするプロセスを中断します。

詳細は、[セクション15.8.3.6「バッファプールの状態の保存と復元」](#)を参照してください。

- `innodb_buffer_pool_load_at_startup`

コマンド行形式	<code>--innodb-buffer-pool-load-at-startup[={OFF ON}]</code>
システム変数	<code>innodb_buffer_pool_load_at_startup</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

MySQL サーバーの起動時に、以前に保持していたものと同じページをロードすることで、InnoDB buffer pool が自動的に `warmed up` になるように指定します。一般に、`innodb_buffer_pool_dump_at_shutdown` と組み合わせて使用されます。

`innodb_buffer_pool_dump_at_shutdown` と `innodb_buffer_pool_load_at_startup` の両方がデフォルトで有効になっています。

詳細は、[セクション15.8.3.6「バッファプールの状態の保存と復元」](#)を参照してください。

- `innodb_buffer_pool_load_now`

コマンド行形式	<code>--innodb-buffer-pool-load-now[={OFF ON}]</code>
システム変数	<code>innodb_buffer_pool_load_now</code>

スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

サーバーの再起動を待たずに一連のデータページをロードすることで、InnoDB buffer pool を即時に **warms up** します。ベンチマーク時にキャッシュメモリーを既知の状態に戻したり、レポートやメンテナンスのためにクエリーを実行したあとに、MySQL サーバーの通常のワークロードを再開する準備をしたりする際に役立ちます。

詳細は、[セクション15.8.3.6「バッファープールの状態の保存と復元」](#)を参照してください。

- [innodb_buffer_pool_size](#)

コマンド行形式	--innodb-buffer-pool-size=#
システム変数	innodb_buffer_pool_size
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	134217728
最小値	5242880
最大値 (64 ビットプラットフォーム)	2**64-1
最大値 (32 ビットプラットフォーム)	2**32-1

InnoDB がテーブルおよびインデックスのデータをキャッシュするメモリー領域である **バッファープール** のサイズ (バイト単位) です。デフォルト値は 134217728 バイト (128MB) です。最大値は、CPU アーキテクチャーによって異なります。最大値は、32 ビットシステムでは 4294967295 ($2^{32}-1$)、64 ビットシステムでは 18446744073709551615 ($2^{64}-1$) です。32 ビットシステムでは、CPU アーキテクチャーおよびオペレーティングシステムに、指定された最大値よりも小さい実用的な最大サイズが課されている可能性があります。バッファープールのサイズが 1G バイトよりも大きい場合に、[innodb_buffer_pool_instances](#) を 1 よりも大きい値に設定すると、高負荷のサーバーで拡張性を改善できます。

バッファープールを大きくすると、同じテーブルデータに複数回アクセスするために必要なディスク I/O が少なくなります。専用データベースサーバーでは、バッファープールサイズをマシンの物理メモリーサイズの 80% に設定できます。バッファープールサイズを構成するときは、次の潜在的な問題に注意し、必要に応じてバッファープールのサイズをスケールバックする準備をしてください。

- 物理メモリーの競合により、オペレーティングシステムでページングが発生する可能性があります。
- InnoDB では、バッファおよび制御構造体用に追加のメモリーが予約されるため、割り当てられる領域の合計は、指定されたバッファープールサイズよりも約 10% 大きくなります。
- バッファープールのアドレス空間は連続している必要があります。これは、特定のアドレスで DLL をロードする Windows システムで問題になる可能性があります。
- バッファープールを初期化する時間は、ほぼそのサイズに比例しています。バッファープールが大きいインスタンスでは、初期化にかなりの時間がかかる場合があります。初期化期間を短縮するには、サーバーの停止時にバッ

ファイルの状態を保存し、サーバーの起動時にリストアします。 [セクション15.8.3.6「バッファープールの状態の保存と復元」](#)を参照してください。

バッファープールサイズを増減すると、操作はチャンク単位で実行されます。チャンクサイズは、`innodb_buffer_pool_chunk_size` 変数 (デフォルトは 128 MB) によって定義されます。

バッファープールサイズは、常に `innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances` と等しいか倍数である必要があります。バッファープールサイズを `innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances` と等しくない値または倍数に変更すると、バッファープールサイズは `innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances` と等しいか倍数の値に自動的に調整されます。

`innodb_buffer_pool_size` は動的に設定できるため、サーバーを再起動せずにバッファープールのサイズを変更できます。`innodb_buffer_pool_resize_status` ステータス変数は、オンラインバッファープールのサイズ変更操作のステータスを報告します。詳しくは [セクション15.8.3.1「InnoDB バッファープールサイズの構成」](#) をご覧ください。

`innodb_dedicated_server` が有効な場合、`innodb_buffer_pool_size` 値は明示的に定義されていなければ自動的に構成されます。詳細は、[セクション15.8.12「専用 MySQL Server の自動構成の有効化」](#) を参照してください。

- `innodb_change_buffer_max_size`

コマンド行形式	<code>--innodb-change-buffer-max-size=#</code>
システム変数	<code>innodb_change_buffer_max_size</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	25
最小値	0
最大値	50

`buffer pool` の合計サイズに対する InnoDB `change buffer` の最大サイズの割合。この値は、MySQL サーバーで頻繁に挿入、更新、および削除アクティビティーが発生する場合は大きくし、MySQL サーバーでレポート用に使用されるデータが変更されない場合は小さくするとよいでしょう。詳細は、[セクション15.5.2「変更バッファ」](#) を参照してください。一般的な I/O チューニングのアドバイスについては、[セクション8.5.8「InnoDB ディスク I/O の最適化」](#) を参照してください。

- `innodb_change_buffering`

コマンド行形式	<code>--innodb-change-buffering=value</code>
システム変数	<code>innodb_change_buffering</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	all
有効な値	none inserts deletes changes purges

all

InnoDB が **バッファリングの変更** (I/O 操作を連続して実行できるように、セカンダリインデックスへの書き込み操作を遅延させる最適化) を実行するかどうかを指定します。次のテーブルに、許可される値を示します。値は数値で指定することもできます。

表 15.25 innodb_change_buffering に許可される値

値	数値	説明
none	0	どの操作もバッファリングしません。
inserts	1	挿入操作をバッファリングします。
deletes	2	バッファ削除マーキング操作。厳密に言えば、ページ操作中に後で削除するためにインデックスレコードをマークする書き込み。
changes	3	バッファの挿入および削除マーク操作。
purges	4	バックグラウンドで実行される物理的な削除操作をバッファリングしません。
all	5	デフォルト。バッファの挿入、削除マーク操作およびページ。

詳細は、[セクション15.5.2「変更バッファ」](#)を参照してください。一般的な I/O チューニングのアドバイスについては、[セクション8.5.8「InnoDB ディスク I/O の最適化」](#)を参照してください。

- [innodb_change_buffering_debug](#)

コマンド行形式	<code>--innodb-change-buffering-debug=#</code>
システム変数	<code>innodb_change_buffering_debug</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最大値	2

InnoDB 変更バッファリングのデバッグフラグを設定します。値 1 を指定すると、変更バッファに対するすべての変更が強制されます。値 2 を指定すると、マージ時に予期しない終了が発生します。デフォルト値の 0 は、変更バッファリングデバッグフラグが設定されていないことを示します。このオプションは、デバッグサポートが `WITH_DEBUG CMake` オプションを使用してコンパイルされている場合にのみ使用できます。

- [innodb_checkpoint_disabled](#)

コマンド行形式	<code>--innodb-checkpoint-disabled[={OFF ON}]</code>
システム変数	<code>innodb_checkpoint_disabled</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean

デフォルト値	OFF
--------	-----

これは、エキスパートによるデバッグのみを目的としたデバッグオプションです。チェックポイントを無効にして、故意のサーバイグジットが常に InnoDB リカバリを開始するようにします。通常は、サーバーの終了後にリカバリが必要な redo ログエントリを書き込む DML 操作を実行する前に、短い間隔でのみ有効にする必要があります。このオプションは、デバッグサポートが `WITH_DEBUG CMake` オプションを使用してコンパイルされている場合にのみ使用できます。

- `innodb_checksum_algorithm`

コマンド行形式	<code>--innodb-checksum-algorithm=value</code>
システム変数	<code>innodb_checksum_algorithm</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	<code>crc32</code>
有効な値	<code>innodb</code> <code>crc32</code> <code>none</code> <code>strict_innodb</code> <code>strict_crc32</code> <code>strict_none</code>

InnoDB `tablespaces` のディスクブロックに格納されている `checksum` を生成および検証する方法を指定します。`innodb_checksum_algorithm` のデフォルト値は `crc32` です。

3.8.0 までのバージョンの `MySQL Enterprise Backup` は、CRC32 チェックサムを使用するテーブルスペースのバックアップをサポートしていません。`MySQL Enterprise Backup` は、CRC32 チェックサムのサポートを 3.8.1 で (いくつかの制限付きで) 追加しています。詳細は、`MySQL Enterprise Backup 3.8.1` の変更履歴を参照してください。

値 `innodb` は、以前のバージョンの MySQL と下位互換性があります。値 `crc32` では、より高速に、変更されたすべてのブロックのチェックサムを計算し、ディスク読み取りごとにチェックサムをチェックするアルゴリズムが使用されます。ブロックを一度に 32 ビットスキャンします。これは、ブロックを一度に 8 ビットスキャンする `innodb` チェックサムアルゴリズムより高速です。値 `none` では、ブロックデータに基づいて値が計算されるのではなく、チェックサムフィールドに定数値が書き込まれます。テーブルスペース内のブロックは、データの変更時に徐々に更新される古いチェックサム値と新しいチェックサム値を混在させて使用できます。テーブルスペース内のブロックが `crc32` アルゴリズムを使用するように変更されると、関連付けられたテーブルを以前のバージョンの MySQL で読み取ることはできません。

チェックサムアルゴリズムの厳密な形式では、テーブルスペースで有効だが一致しないチェックサム値が検出されると、エラーが報告されます。テーブルスペースを初めて設定する場合は、新しいインスタンスでのみ厳密な設定を使用することをお勧めします。厳密な設定は、ディスク読み取り時にすべてのチェックサム値を計算する必要がないため、多少高速です。

次のテーブルに、`none`、`innodb` および `crc32` オプションの値とそれぞれに対応する厳密な値の違いを示します。`none`、`innodb` および `crc32` は、指定されたタイプのチェックサム値を各データブロックに書き込みますが、互換性のために、読み取り操作中にブロックを検証するときに他のチェックサム値を受け入れます。厳密な設定も有効なチェックサム値を受け入れますが、一致しない有効なチェックサム値が検出されるとエラーメッセージ

セージを出力します。厳密な形式を使用すると、インスタンス内のすべての InnoDB データファイルが同一の `innodb_checksum_algorithm` 値で作成される場合に、検証が高速になります。

表 15.26 許可される `innodb_checksum_algorithm` 値

値	生成されるチェックサム (書き込み時)	許可されたチェックサム (読取り時)
<code>none</code>	定数。	<code>none</code> 、 <code>innodb</code> 、または <code>crc32</code> で生成されるチェックサムのいずれか。
<code>innodb</code>	ソフトウェアで InnoDB の元のアルゴリズムを使用して計算されたチェックサム。	<code>none</code> 、 <code>innodb</code> 、または <code>crc32</code> で生成されるチェックサムのいずれか。
<code>crc32</code>	<code>crc32</code> アルゴリズムを使用して計算されたチェックサム (ハードウェアの支援を得て実行される可能性もあります)。	<code>none</code> 、 <code>innodb</code> 、または <code>crc32</code> で生成されるチェックサムのいずれか。
<code>strict_none</code>	定数	<code>none</code> 、 <code>innodb</code> 、または <code>crc32</code> で生成されるチェックサムのいずれか。有効だが一致しないチェックサムが検出されると、InnoDB はエラーメッセージを出力します。
<code>strict_innodb</code>	ソフトウェアで InnoDB の元のアルゴリズムを使用して計算されたチェックサム。	<code>none</code> 、 <code>innodb</code> 、または <code>crc32</code> で生成されるチェックサムのいずれか。有効だが一致しないチェックサムが検出されると、InnoDB はエラーメッセージを出力します。
<code>strict_crc32</code>	<code>crc32</code> アルゴリズムを使用して計算されたチェックサム (ハードウェアの支援を得て実行される可能性もあります)。	<code>none</code> 、 <code>innodb</code> 、または <code>crc32</code> で生成されるチェックサムのいずれか。有効だが一致しないチェックサムが検出されると、InnoDB はエラーメッセージを出力します。

- `innodb_cmp_per_index_enabled`

コマンド行形式	<code>--innodb-cmp-per-index-enabled[={OFF ON}]</code>
システム変数	<code>innodb_cmp_per_index_enabled</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

`INFORMATION_SCHEMA.INNODB_CMP_PER_INDEX` テーブルでインデックスごとの圧縮関連の統計を有効にします。これらの統計は収集にコストがかかる可能性があるため、このオプションは、InnoDB `compressed` テーブルに関連するパフォーマンスチューニング中に開発、テストまたはレプリカインスタンスでのみ有効にします。

詳細は、[セクション26.51.7「INFORMATION_SCHEMA INNODB_CMP_PER_INDEX および INNODB_CMP_PER_INDEX_RESET テーブル」](#) および [セクション15.9.1.4「実行時の InnoDB テーブル圧縮の監視」](#) を参照してください。

- `innodb_commit_concurrency`

コマンド行形式	<code>--innodb-commit-concurrency=#</code>
システム変数	<code>innodb_commit_concurrency</code>

スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	1000

同時にコミットできるスレッドの数です。値を 0 (デフォルト) にすると、任意の数のトランザクションを同時にコミットすることが許可されます。

`innodb_commit_concurrency` の値は、実行時にゼロからゼロ以外 (またはその逆) に変更できません。ゼロ以外の値から別のゼロ以外の値に変更することはできます。

- `innodb_compress_debug`

コマンド行形式	<code>--innodb-compress-debug=value</code>
システム変数	<code>innodb_compress_debug</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	<code>none</code>
有効な値	<code>none</code> <code>zlib</code> <code>lz4</code> <code>lz4hc</code>

テーブルごとに `COMPRESSION` 属性を定義せずに、指定された圧縮アルゴリズムを使用してすべてのテーブルを圧縮します。このオプションは、デバッグサポートが `WITH_DEBUG CMake` オプションを使用してコンパイルされている場合にのみ使用できます。

関連情報については、[セクション15.9.2「InnoDB ページ圧縮」](#)を参照してください。

- `innodb_compression_failure_threshold_pct`

コマンド行形式	<code>--innodb-compression-failure-threshold-pct=#</code>
システム変数	<code>innodb_compression_failure_threshold_pct</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	5
最小値	0
最大値	100

高コストの `compression failures` を回避するために MySQL が `compressed` ページ内でパディングの追加を開始するテーブルの圧縮失敗率のしきい値をパーセンテージで定義します。このしきい値を超えると、MySQL は、最大で `innodb_compression_pad_pct_max` で指定されたページサイズの割合まで空き領域の量を動的に調整することで、

新しい各圧縮済みページ内に追加の空き領域を残し始めます。値をゼロにすると、圧縮の効率性をモニターするメカニズムが無効になり、パディングの量が動的に調整されます。

詳細は、[セクション15.9.1.6「OLTP ワークロードの圧縮」](#)を参照してください。

- [innodb_compression_level](#)

コマンド行形式	<code>--innodb-compression-level=#</code>
システム変数	innodb_compression_level
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	6
最小値	0
最大値	9

InnoDB の圧縮されたテーブルおよびインデックスで使用される zlib 圧縮のレベルを指定します。値を大きくすると、ストレージデバイス上に収容できるデータ量が多くなりますが、圧縮時の CPU オーバーヘッドも多くなるという犠牲が伴います。値を小さくすると、ストレージ領域がクリティカルでない場合に、CPU のオーバーヘッドを削減できます。それ以外の場合は、データが特に圧縮可能でないと予測されます。

詳細は、[セクション15.9.1.6「OLTP ワークロードの圧縮」](#)を参照してください。

- [innodb_compression_pad_pct_max](#)

コマンド行形式	<code>--innodb-compression-pad-pct-max=#</code>
システム変数	innodb_compression_pad_pct_max
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	50
最小値	0
最大値	75

圧縮された各ページ内の空き領域として予約できる最大の割合を指定します。これにより、圧縮されたテーブルまたはインデックスが更新され、データが再度圧縮される可能性があるときに、ページ内のデータおよび変更ログを再編成する余地が得られます。[innodb_compression_failure_threshold_pct](#) がゼロ以外の値に設定され、[compression failures](#) のレートがカットオフポイントを通過する場合にのみ適用されます。

詳細は、[セクション15.9.1.6「OLTP ワークロードの圧縮」](#)を参照してください。

- [innodb_concurrency_tickets](#)

コマンド行形式	<code>--innodb-concurrency-tickets=#</code>
システム変数	innodb_concurrency_tickets
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer

デフォルト値	5000
最小値	1
最大値	4294967295

同時に InnoDB に入ることができるスレッドの数を決定します。スレッドが InnoDB に入ろうとしたときに、すでにスレッド数が並列実行の制限に達している場合は、そのスレッドがキューに配置されます。スレッドが InnoDB に入ることが許可されている場合は、`innodb_concurrency_tickets` の値と同じ数の「チケット」が与えられ、チケットを使い切るまでスレッドは InnoDB に自由に入ることができます。それ以降は、スレッドが次に InnoDB に入ろうとしたときに、再度並列実行チェックの対象となります (キューに入る対象となる可能性もあります)。デフォルト値は 5000 です。

`innodb_concurrency_tickets` 値を小さくすると、1、2 行しか処理する必要のない小規模なトランザクションと、多数の行を処理する大規模なトランザクションが競合する可能性が高くなります。小さい `innodb_concurrency_tickets` 値のデメリットは、大規模なトランザクションが完了する前にキューを何度もループする必要があり、これによりタスクの完了に必要な時間が長くなることです。

`innodb_concurrency_tickets` 値を大きくすると、大規模なトランザクションで (`innodb_thread_concurrency` で制御される) キューの終了時の位置を待機する時間が短くなり、行を取得する時間が長くなります。また、大規模なトランザクションでは、タスクを完了するために必要なキューとの間の移動も少なくなります。`innodb_concurrency_tickets` 値を大きくする欠点は、同時に実行する大規模なトランザクションの数が非常に多くなることで、小規模なトランザクションが実行されるまでの待機時間が長くなるため、枯渇する可能性がある点です。

ゼロ以外の `innodb_thread_concurrency` 値では、`innodb_concurrency_tickets` 値を上下に調整して、大規模なトランザクションと小規模なトランザクションの間の最適なバランスを見つける必要がある場合があります。`SHOW ENGINE INNODB STATUS` レポートには、キューを通過する現時点で実行中のトランザクション用に残されているチケットの数が表示されます。このデータは、`INFORMATION_SCHEMA.INNODB_TRX` テーブルの `TRX_CONCURRENCY_TICKETS` カラムから取得することもできます。

詳細は、[セクション 15.8.4 「InnoDB のスレッド並列性の構成」](#) を参照してください。

- `innodb_data_file_path`

コマンド行形式	<code>--innodb-data-file-path=file_name</code>
システム変数	<code>innodb_data_file_path</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	<code>ibdata1:12M:autoextend</code>

InnoDB システムテーブルスペースデータファイルの名前、サイズおよび属性を定義します。`innodb_data_file_path` の値を指定しない場合、デフォルトの動作では、12MB を少し超える単一の自動拡張データファイルが `ibdata1` という名前で作成されます。

データファイル指定の完全な構文には、ファイル名、ファイルサイズ、`autoextend` 属性および `max` 属性が含まれます:

```
file_name:file_size[:autoextend[:max:max_file_size]]
```

ファイルサイズは、**K**、**M** または **G** をサイズ値に追加することで、KB、MB または GB 単位で指定します。データファイルのサイズを KB 単位で指定する場合は、1024 の倍数で指定します。それ以外の場合、KB 値は最も近いメガバイト (MB) 境界に丸められます。ファイルサイズの合計は、12MB 以上である必要があります。

その他の構成情報については、[システムテーブルスペースデータファイル構成](#) を参照してください。サイズ変更の手順は、[システムテーブルスペースのサイズ変更](#) を参照してください。

- innodb_data_home_dir

コマンド行形式	<code>--innodb-data-home-dir=dir_name</code>
システム変数	<code>innodb_data_home_dir</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ディレクトリ名

InnoDB `system tablespace` データファイルのディレクトリパスの共通部分。デフォルト値は、MySQL の `data` ディレクトリです。設定が絶対パスで定義されていない限り、この設定は `innodb_data_file_path` 設定と連結されません。

`innodb_data_home_dir` の値を指定する場合は、末尾にスラッシュが必要です。例:

```
[mysqld]
innodb_data_home_dir = /path/to/myibdata/
```

この設定は、`file-per-table` テーブルスペースの場所には影響しません。

関連情報については、[セクション15.8.1「InnoDB の起動構成」](#)を参照してください。

- innodb_ddl_log_crash_reset_debug

コマンド行形式	<code>--innodb-ddl-log-crash-reset-debug[={OFF ON}]</code>
システム変数	<code>innodb_ddl_log_crash_reset_debug</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

DDL ログクラッシュインサーションカウンタを 1 にリセットするには、このデバッグオプションを有効にします。このオプションは、デバッグサポートが `WITH_DEBUG CMake` オプションを使用してコンパイルされている場合のみ使用できます。

- innodb_deadlock_detect

コマンド行形式	<code>--innodb-deadlock-detect[={OFF ON}]</code>
システム変数	<code>innodb_deadlock_detect</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

このオプションは、デッドロック検出を無効にするために使用します。同時実行性の高いシステムでは、多数のスレッドが同じロックを待機している場合、デッドロック検出によって速度が低下する可能性があります。デッドロック検出を無効にし、デッドロック発生時のトランザクションロールバックの `innodb_lock_wait_timeout` 設定に依存する方が効率的な場合があります。

関連情報については、[セクション15.7.5.2「デッドロック検出」](#)を参照してください。

- [innodb_dedicated_server](#)

コマンド行形式	<code>--innodb-dedicated-server[={OFF ON}]</code>
システム変数	<code>innodb_dedicated_server</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

`innodb_dedicated_server` が有効な場合、InnoDB は次の変数を自動的に構成します:

- [innodb_buffer_pool_size](#)
- [innodb_log_file_size](#)
- [innodb_log_files_in_group](#) (MySQL 8.0.14 の時点)
- [innodb_flush_method](#)

MySQL インスタンスが、使用可能なすべてのシステムリソースを使用できる専用サーバーに存在する場合にのみ、`innodb_dedicated_server` を有効にすることを検討してください。MySQL インスタンスが他のアプリケーションとシステムリソースを共有している場合、`innodb_dedicated_server` を有効にすることはお薦めしません。

詳細は、[セクション15.8.12「専用 MySQL Server の自動構成の有効化」](#)を参照してください。

- [innodb_default_row_format](#)

コマンド行形式	<code>--innodb-default-row-format=value</code>
システム変数	<code>innodb_default_row_format</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	DYNAMIC
有効な値	DYNAMIC COMPACT REDUNDANT

`innodb_default_row_format` オプションは、InnoDB テーブルおよびユーザー作成一時テーブルのデフォルトの行形式を定義します。デフォルト設定は DYNAMIC です。許可されるその他の値は、COMPACT および REDUNDANT です。`system tablespace` での使用がサポートされていない COMPRESSED 行フォーマットは、デフォルトとして定義できません。

新しく作成されたテーブルでは、`ROW_FORMAT` オプションが明示的に指定されていない場合、または `ROW_FORMAT=DEFAULT` が使用されている場合に、`innodb_default_row_format` で定義された行形式が使用されます。

`ROW_FORMAT` オプションが明示的に指定されていない場合、または `ROW_FORMAT=DEFAULT` が使用されている場合は、テーブルを再構築する操作によって、テーブルの行形式も `innodb_default_row_format` で定義された形式に暗黙的に変更されます。詳細は、[テーブルの行形式の定義](#)を参照してください。

クエリーを処理するためにサーバーによって作成された内部 InnoDB 一時テーブルは、`innodb_default_row_format` の設定に関係なく、DYNAMIC 行形式を使用します。

- innodb_directories

コマンド行形式	--innodb-directories=dir_name
システム変数	innodb_directories
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ディレクトリ名

起動時にテーブルスペースファイルをスキャンするディレクトリを定義します。このオプションは、サーバーがオフラインのときにテーブルスペースファイルを新しい場所に移動またはリストアする場合に使用します。また、絶対パスを使用して作成されたテーブルスペースファイル、またはデータディレクトリの外部にあるテーブルスペースファイルのディレクトリを指定するためにも使用されます。

クラッシュリカバリ中のテーブルスペースの検出は、redo ログで参照されるテーブルスペースを識別するために `innodb_directories` 設定に依存します。詳細は、[クラッシュリカバリ中のテーブルスペースの検出](#)を参照してください。

`innodb_data_home_dir`、`innodb_undo_directory` および `datadir` によって定義されたディレクトリは、`innodb_directories` オプションが明示的に指定されているかどうかに関係なく、起動時にスキャンするディレクトリのリストを作成するときに `innodb_directories` 引数値に自動的に追加されます。

`innodb_directories` は、起動コマンドまたは MySQL オプションファイルでオプションとして指定できます。一部のコマンドインタプリタではセミコロン (;) は特殊文字として解釈されるため、引数値の前後に引用符が使用されます。(たとえば UNIX シェルでは、これはコマンド終端記号として扱われます。)

起動コマンド:

```
mysqld --innodb-directories="directory_path_1;directory_path_2"
```

MySQL オプションファイル:

```
[mysqld]
innodb_directories="directory_path_1;directory_path_2"
```

ワイルドカード式は、ディレクトリの指定には使用できません。

`innodb_directories` スキャンは、指定されたディレクトリのサブディレクトリも走査します。重複するディレクトリおよびサブディレクトリは、スキャンされるディレクトリのリストから破棄されます。

詳細は、[セクション15.6.3.6「サーバーがオフラインのときのテーブルスペースファイルの移動」](#)を参照してください。

- innodb_disable_sort_file_cache

コマンド行形式	--innodb-disable-sort-file-cache[={OFF ON}]
システム変数	innodb_disable_sort_file_cache
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

マージソート一時ファイルのオペレーティングシステムファイルシステムキャッシュを無効にします。その結果、このようなファイルが `O_DIRECT` の同等のものとともに開きます。

- innodb_doublewrite

コマンド行形式	<code>--innodb-doublewrite[={OFF ON}]</code>
システム変数	<code>innodb_doublewrite</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

`innodb_doublewrite` 変数は、二重書き込みバッファを有効にするかどうかを制御します。ほとんどの場合、デフォルトで有効になっています。二重書き込みバッファを無効にするには、`innodb_doublewrite` を 0 に設定するか、`--skip-innodb-doublewrite` でサーバーを起動します。たとえば、ベンチマークの実行時などのように、データ整合性よりもパフォーマンスに関心がある場合は、二重書き込みバッファを無効にすることを検討してください。

二重書き込みバッファがアトミック書き込みをサポートする Fusion-io デバイス上にある場合、二重書き込みバッファは自動的に無効になり、代わりに Fusion-io アトミック書き込みを使用してデータファイル書き込みが実行されます。ただし、`innodb_doublewrite` 設定はグローバルであることに注意してください。二重書き込みバッファが無効になっている場合、Fusion-io ハードウェア上に存在しないデータファイルを含むすべてのデータファイルに対して無効になります。この機能は Fusion-io ハードウェアでのみサポートされ、Linux の Fusion-io NVMFS でのみ有効になります。この機能を最大限に活用するには、`O_DIRECT` の `innodb_flush_method` 設定をお勧めします。

関連情報については、[セクション15.6.4「二重書き込みバッファ」](#)を参照してください。

- innodb_doublewrite_batch_size

コマンド行形式	<code>--innodb-doublewrite-batch-size=#</code>
導入	8.0.20
システム変数	<code>innodb_doublewrite_batch_size</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	256

バッチで書き込む二重書き込みページの数を実験します。

詳細は、[セクション15.6.4「二重書き込みバッファ」](#)を参照してください。

- innodb_doublewrite_dir

コマンド行形式	<code>--innodb-doublewrite-dir=dir_name</code>
導入	8.0.20
システム変数	<code>innodb_doublewrite_dir</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ

型	ディレクトリ名
---	---------

二重書き込みファイルのディレクトリを定義します。ディレクトリが指定されていない場合、二重書き込みファイルが `innodb_data_home_dir` ディレクトリに作成され、指定されていない場合はデータディレクトリにデフォルト設定されます。

詳細は、[セクション15.6.4「二重書き込みバッファ」](#)を参照してください。

- [innodb_doublewrite_files](#)

コマンド行形式	<code>--innodb-doublewrite-files=#</code>
導入	8.0.20
システム変数	innodb_doublewrite_files
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	<code>innodb_buffer_pool_instances * 2</code>
最小値	2
最大値	256

二重書き込みファイルの数を定義します。デフォルトでは、バッファプールインスタンスごとに2つの二重書き込みファイルが作成されます。

少なくとも2つの二重書き込みファイルがあります。二重書き込みファイルの最大数は、バッファプールインスタンスの2倍です。(バッファプールインスタンスの数は、[innodb_buffer_pool_instances](#) 変数によって制御されます。)

詳細は、[セクション15.6.4「二重書き込みバッファ」](#)を参照してください。

- [innodb_doublewrite_pages](#)

コマンド行形式	<code>--innodb-doublewrite-pages=#</code>
導入	8.0.20
システム変数	innodb_doublewrite_pages
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	<code>innodb_write_io_threads value</code>
最小値	<code>innodb_write_io_threads value</code>
最大値	512

バッチ書き込みのスレッド当たりの二重書き込みページの最大数を定義します。値が指定されていない場合、[innodb_doublewrite_pages](#) は `innodb_write_io_threads` 値に設定されます。

詳細は、[セクション15.6.4「二重書き込みバッファ」](#)を参照してください。

- [innodb_extend_and_initialize](#)

コマンド行形式	<code>--innodb=extend-and-initialize[={OFF ON}]</code>
導入	8.0.22
システム変数	innodb_extend_and_initialize

スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

Linux システムの file-per-table テーブルスペースおよび一般テーブルスペースへの領域の割当て方法を制御します。

有効にすると、InnoDB は新しく割り当てられたページに NULL を書き込みます。無効にすると、領域は `posix_fallocate()` コールを使用して割り当てられます。このコールは、物理的に NULL を書き込まずに領域を予約します。

詳細は、[セクション15.6.3.8「Linux でのテーブルスペースの領域割当ての最適化」](#)を参照してください。

- [innodb_fast_shutdown](#)

コマンド行形式	<code>--innodb-fast-shutdown=#</code>
システム変数	<code>innodb_fast_shutdown</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1
有効な値	0 1 2

InnoDB のシャットダウンモードです。値が 0 の場合、InnoDB は、停止前に `slow shutdown`、完全な `purge` および変更バッファのマージを実行します。この値を 1 (デフォルト) にすると、InnoDB はシャットダウン時に、これらの操作をスキップします。このプロセスは、`高速シャットダウン`と呼ばれます。この値を 2 にすると、InnoDB は MySQL がクラッシュした場合と同様に、そのログをフラッシュし、コールドシャットダウンを実行します。コミットされていないトランザクションは失われませんが、`クラッシュリカバリ`操作によって次の起動時間が長くなります。

低速シャットダウンには数分間かかる可能性があり、大量のデータがバッファに存在する極端なケースでは、数時間かかる可能性もあります。MySQL のメジャーリリース間でアップグレードまたはダウングレードを行う前には、アップグレードプロセスによってファイル形式が更新される場合に備えて、すべてのデータファイルが完全に準備されるように、低速シャットダウン技術を使用してください。

データが破損するリスクがある場合に、完全な最速のシャットダウンを行うには、緊急事態またはトラブルシューティングの状況で `innodb_fast_shutdown=2` を使用してください。

- [innodb_fil_make_page_dirty_debug](#)

コマンド行形式	<code>--innodb-fil-make-page-dirty-debug=#</code>
システム変数	<code>innodb_fil_make_page_dirty_debug</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0

最大値	2**32-1
-----	---------

デフォルトでは、`innodb_fil_make_page_dirty_debug` をテーブルスペースの ID に設定すると、テーブルスペースの最初のページがすぐに使用済になります。 `innodb_saved_page_number_debug` がデフォルト以外の値に設定されている場合、`innodb_fil_make_page_dirty_debug` を設定すると、指定したページがダーティになります。 `innodb_fil_make_page_dirty_debug` オプションは、デバッグサポートが `WITH_DEBUG CMake` オプションを使用してコンパイルされている場合にのみ使用できます。

- `innodb_file_per_table`

コマンド行形式	<code>--innodb-file-per-table[={OFF ON}]</code>
システム変数	<code>innodb_file_per_table</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

`innodb_file_per_table` が有効な場合、テーブルはデフォルトで file-per-table テーブルスペースに作成されます。無効にすると、デフォルトでシステムテーブルスペースにテーブルが作成されます。file-per-table テーブルスペースについては、[セクション15.6.3.2「File-Per-Table テーブルスペース」](#)を参照してください。InnoDB システムテーブルスペースの詳細は、[セクション15.6.3.1「システムテーブルスペース」](#)を参照してください。

`innodb_file_per_table` 変数は、実行時に `SET GLOBAL` ステートメントを使用して構成するか、起動時にコマンドラインで指定するか、またはオプションファイルで指定できます。実行時の構成には、グローバルシステム変数を設定するのに十分な権限 ([セクション5.1.9.1「システム変数権限」](#)を参照) が必要で、すべての接続の操作にすぐに影響します。

file-per-table テーブルスペースに存在するテーブルが切り捨てられるか削除されると、解放された領域がオペレーティングシステムに戻されます。システムテーブルスペースに存在するテーブルの切捨てまたは削除では、システムテーブルスペースの領域のみが解放されます。システムテーブルスペースのデータファイルは縮小しないため、システムテーブルスペースの空き領域は InnoDB データに再度使用できますが、オペレーティングシステムには戻されません。

`innodb_file_per-table` 設定は、一時テーブルの作成には影響しません。MySQL 8.0.14 では、一時テーブルはセッション一時テーブルスペースに作成され、その前にグローバル一時テーブルスペースに作成されます。[セクション15.6.3.5「一時テーブルスペース」](#)を参照してください。

- `innodb_fill_factor`

コマンド行形式	<code>--innodb-fill-factor=#</code>
システム変数	<code>innodb_fill_factor</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	100
最小値	10
最大値	100

InnoDB では、インデックスの作成または再構築時にバルクロードが実行されます。このインデックス作成方法は、「ソートされたインデックス構築」と呼ばれます。

`innodb_fill_factor` では、ソートされたインデックスの作成時に入力される各 B ツリーページ上の領域の割合が定義され、将来のインデックスの増加のために予約されている残りの領域が使用されます。たとえ

ば、`innodb_fill_factor` を 80 に設定すると、将来のインデックス増加のために各 B ツリーページの領域の 20% が予約されます。実際の割合は異なる場合があります。`innodb_fill_factor` 設定は、強い制限ではなくヒントとして解釈されます。

`innodb_fill_factor` を 100 に設定すると、クラスタ化されたインデックスページの領域の 1/16 は将来のインデックスの増加に備えて解放されます。

`innodb_fill_factor` は、B ツリーリーフページと非リーフページの両方に適用されます。`TEXT` または `BLOB` エントリに使用される外部ページには適用されません。

詳細は、[セクション15.6.2.3「ソートされたインデックス構築」](#)を参照してください。

- `innodb_flush_log_at_timeout`

コマンド行形式	<code>--innodb-flush-log-at-timeout=#</code>
システム変数	<code>innodb_flush_log_at_timeout</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1
最小値	1
最大値	2700

N 秒ごとにログを書き込み、フラッシュします。`innodb_flush_log_at_timeout` では、フラッシュを減らし、バイナリロググループのコミットのパフォーマンスへの影響を回避するために、フラッシュ間のタイムアウト期間を増やすことができます。`innodb_flush_log_at_timeout` のデフォルト設定は 1 秒に 1 回です。

- `innodb_flush_log_at_trx_commit`

コマンド行形式	<code>--innodb-flush-log-at-trx-commit=#</code>
システム変数	<code>innodb_flush_log_at_trx_commit</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	1
有効な値	0 1 2

`commit` 操作に対する厳密な `ACID` コンプライアンスと、コミット関連の I/O 操作がバッチで再配置および実行される場合に可能なパフォーマンスのバランスを制御します。デフォルト値を変更することでパフォーマンスを向上できますが、クラッシュ時にトランザクションが失われる可能性があります。

- `ACID` に完全に準拠するには、デフォルト設定の 1 が必要です。ログは、トランザクションのコミットごとにディスクに書き込まれ、フラッシュされます。
- 0 に設定すると、ログは 1 秒に 1 回書き込まれ、ディスクにフラッシュされます。ログがフラッシュされていないトランザクションはクラッシュ時に失われる可能性があります。
- 2 に設定すると、各トランザクションのコミット後にログが書き込まれ、1 秒に 1 回ディスクにフラッシュされます。ログがフラッシュされていないトランザクションはクラッシュ時に失われる可能性があります。

- 設定 0 および 2 の場合、秒単位のフラッシュは 100% 保証されません。フラッシュは、DDL 変更や、`innodb_flush_log_at_trx_commit` 設定とは関係なくログがフラッシュされる原因となるその他の内部 InnoDB アクティビティが原因で頻繁に発生し、スケジューリングの問題が原因で頻繁に発生しない場合があります。ログが 1 秒に 1 回フラッシュされると、クラッシュ時に最大 1 秒のトランザクションが失われる可能性があります。ログが 1 秒に 1 回以上フラッシュされるか、それほど頻繁にフラッシュされない場合、失われる可能性のあるトランザクションの量はそれに依りて異なります。
- ログのフラッシュ頻度は `innodb_flush_log_at_timeout` によって制御されます。これにより、ログのフラッシュ頻度を N 秒 (N は 1 ... 2700 で、デフォルト値は 1) に設定できます。ただし、予期しない `mysqld` プロセスの終了によって、最大 N 秒のトランザクションが消去される可能性があります。
- DDL 変更およびその他の内部 InnoDB アクティビティは、`innodb_flush_log_at_trx_commit` 設定とは関係なくログをフラッシュします。
- InnoDB crash recovery は、`innodb_flush_log_at_trx_commit` の設定に関係なく機能します。トランザクションは完全に適用されるか、完全に消去されるかのいずれかです。

トランザクションで InnoDB が使用されるレプリケーションセットアップの持続性および一貫性を保つ場合:

- バイナリロギングが有効になっている場合は、`sync_binlog=1` を設定します。
- 常に `innodb_flush_log_at_trx_commit=1` を設定します。

予期しない停止に対して最も回復可能なレプリカの設定の組合せの詳細は、[セクション 17.4.2 「レプリカの予期しない停止の処理」](#) を参照してください。

注意

多くのオペレーティングシステムや一部のディスクハードウェアは、ディスクへのフラッシュ操作を行なったと欺きます。フラッシュが行われていなくても、行われたと `mysqld` に通知される可能性があります。この場合、推奨設定であってもトランザクションの永続性は保証されず、最悪の場合は停電によって InnoDB データが破損する可能性があります。バッテリーバックアップのディスクキャッシュを SCSI ディスクコントローラ内やディスク自体で使用すると、ファイルフラッシュの速度が上がり、操作が安全になります。ハードウェアキャッシュ内のディスク書き込みのキャッシュを無効にすることもできます。

- `innodb_flush_method`

コマンド行形式	<code>--innodb-flush-method=value</code>
システム変数	<code>innodb_flush_method</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値 (Unix)	<code>fsync</code>
デフォルト値 (Windows)	<code>unbuffered</code>
有効な値 (Unix)	<code>fsync</code> <code>O_DSYNC</code> <code>littlesync</code> <code>nosync</code> <code>O_DIRECT</code> <code>O_DIRECT_NO_FSYNC</code>

有効な値 (Windows)	unbuffered normal
----------------	----------------------

InnoDB data files および log files への flush データに使用される方法を定義します。これは I/O スループットに影響する可能性があります。

Unix に似たシステムでは、デフォルト値は `fsync` です。Windows では、デフォルト値は `unbuffered` です。

注記

MySQL 8.0 では、`innodb_flush_method` オプションを数値で指定できます。

Unix に似たシステムの `innodb_flush_method` オプションには、次のものがあります:

- `fsync` または `0`: InnoDB は、`fsync()` システムコールを使用して、データファイルとログファイルの両方をフラッシュします。`fsync` がデフォルト設定です。
- `O_DSYNC` または `1`: InnoDB では、`O_SYNC` を使用してログファイルをオープンおよびフラッシュし、`fsync()` を使用してデータファイルをフラッシュします。さまざまな種類の Unix で問題が発生しているため、InnoDB では直接 `O_DSYNC` が使用されません。
- `littlesync` または `2`: このオプションは内部パフォーマンステストに使用され、現在はサポートされていません。独自のリスクで使用します。
- `nosync` または `3`: このオプションは内部パフォーマンステストに使用され、現在はサポートされていません。独自のリスクで使用します。
- `O_DIRECT` または `4`: InnoDB では、`O_DIRECT` (または Solaris 上の `directio()`) を使用してデータファイルを開き、`fsync()` を使用してデータファイルとログファイルの両方をフラッシュします。このオプションは、一部の GNU/Linux バージョン、FreeBSD、および Solaris で使用可能です。
- `O_DIRECT_NO_FSYNC`: InnoDB は、I/O のフラッシュ中に `O_DIRECT` を使用しますが、書き込み操作のたびに `fsync()` システムコールをスキップします。

MySQL 8.0.14 より前では、この設定は XFS や EXT4 などのファイルシステムには適していません。これらのファイルシステムでは、`fsync()` システムコールを使用してファイルシステムメタデータの変更を同期する必要があります。ファイルシステムのメタデータ変更を同期するためにファイルシステムで `fsync()` システムコールが必要かどうかかわからない場合は、かわりに `O_DIRECT` を使用します。

MySQL 8.0.14 の時点では、`fsync()` は、新しいファイルの作成後、ファイルサイズの増加後およびファイルのクローズ後にコールされ、ファイルシステムメタデータの変更が確実に同期されます。各書き込み操作の後、`fsync()` システムコールはスキップされます。

redo ログファイルとデータファイルが異なるストレージデバイスに存在し、データファイルの書き込みがバッテリーバックされていないデバイスキャッシュからフラッシュされる前に予期しない終了が発生した場合、データが失われる可能性があります。redo ログファイルおよびデータファイルに別の記憶域デバイスを使用する場合、およびデータファイルがバッテリーバックアップされていないキャッシュを持つデバイスに存在する場合は、かわりに `O_DIRECT` を使用します。

Windows システム用の `innodb_flush_method` オプションには、次のものがあります:

- `unbuffered` または `0`: InnoDB は、シミュレートされた非同期 I/O およびバッファなし I/O を使用します。
- `normal` または `1`: InnoDB は、シミュレートされた非同期 I/O およびバッファされた I/O を使用します。

各設定がパフォーマンスに与える影響は、ハードウェア構成およびワークロードによって異なります。使用する設定を決定したり、デフォルト設定のままにするかどうかを決定したりするには、特定の構成でベンチマークを実施します。設定ごとに `fsync()` 呼び出しの全体数を確認するには、`Innodb_data_fsyncs` ステータス変数を調査します。ワークロードに読み取り操作と書き込み操作を混在させると、一部の設定での実行が影響を受ける可能性があります。たとえば、ハードウェア RAID コントローラとバッテリーバックアップ式書き込みキャッシュを備えたシステムでは、`O_DIRECT` は、InnoDB バッファプールとオペレーティングシステムのファイルシステムキャッシュ

の間の二重バッファリングを回避するのに役立ちます。InnoDB のデータファイルとログファイルが SAN 上に配置されている一部のシステムでは、大部分の `SELECT` ステートメントを含む読み取り負荷の高いワークロードで、デフォルト値または `O_DSYNC` の速度が速くなる可能性があります。このパラメータは、必ず、本番環境が反映されたハードウェアおよびワークロードでテストしてください。一般的な I/O チューニングのアドバイスについては、[セクション 8.5.8 「InnoDB ディスク I/O の最適化」](#) を参照してください。

`innodb_dedicated_server` が有効な場合、`innodb_flush_method` 値は明示的に定義されていなければ自動的に構成されます。詳細は、[セクション 15.8.12 「専用 MySQL Server の自動構成の有効化」](#) を参照してください。

- `innodb_flush_neighbors`

コマンド行形式	<code>--innodb-flush-neighbors=#</code>
システム変数	<code>innodb_flush_neighbors</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	0
有効な値	0 1 2

`flushing` で、InnoDB buffer pool のページが同じ extent の他の dirty pages もフラッシュするかどうかを指定します。

- 0 に設定すると、`innodb_flush_neighbors` が無効になります。同じエクステント内のダーティページはフラッシュされません。
- 1 に設定すると、連続したダーティページが同じエクステントにフラッシュされます。
- 2 に設定すると、ダーティページが同じエクステントでフラッシュされます。

テーブルデータが従来の HDD ストレージデバイスに格納されている場合は、1 回の操作でこのような隣接ページをフラッシュすると、さまざまな時間に個々のページをフラッシュする場合と比較して、(主にディスクシーク操作の) I/O オーバーヘッドが削減されます。SSD に格納されているテーブルデータの場合、シーク時間は重要な要因ではなく、このオプションを 0 に設定して書き込み操作を分散できます。関連情報については、[セクション 15.8.3.5 「バッファプールのフラッシュの構成」](#) を参照してください。

- `innodb_flush_sync`

コマンド行形式	<code>--innodb-flush-sync[={OFF ON}]</code>
システム変数	<code>innodb_flush_sync</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

`innodb_flush_sync` 変数はデフォルトで有効になっており、`checkpoints` で発生する I/O アクティビティのバースト中に `innodb_io_capacity` 設定が無視されます。`innodb_io_capacity` 設定で定義された I/O レートに準拠するには、`innodb_flush_sync` を無効にします。

`innodb_flush_sync` 変数の構成の詳細は、[セクション 15.8.7 「InnoDB I/O Capacity の構成」](#) を参照してください。

- innodb_flushing_avg_loops

コマンド行形式	<code>--innodb-flushing-avg-loops=#</code>
システム変数	<code>innodb_flushing_avg_loops</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	30
最小値	1
最大値	1000

InnoDB が以前に計算されたフラッシュ状態のスナップショットを保持し、[adaptive flushing](#) が [workloads](#) の変更に応じ、どのくらい迅速に回答するかを制御する反復の数。この値を大きくすると、ワークロードが変化するにつれて、[フラッシュ](#)操作の速度が円滑かつ徐々に変化します。この値を小さくすると、適応型フラッシュがワークロードの変化にすばやく適応します。これにより、ワークロードが突然に増減した場合に、フラッシュアクティビティが急増する可能性があります。

関連情報については、[セクション15.8.3.5「バッファープールのフラッシュの構成」](#)を参照してください。

- innodb_force_load_corrupted

コマンド行形式	<code>--innodb-force-load-corrupted[={OFF ON}]</code>
システム変数	<code>innodb_force_load_corrupted</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

破損としてマークされたテーブルを起動時に InnoDB がロードできるようにします。トラブルシューティング時に、何も対処しなければアクセスできないデータをリカバリする際にのみ使用してください。トラブルシューティングが完了したら、この設定を無効にしてサーバーを再起動します。

- innodb_force_recovery

コマンド行形式	<code>--innodb-force-recovery=#</code>
システム変数	<code>innodb_force_recovery</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0

最大値	6
-----	---

クラッシュリカバリモードです。一般に、重大なトラブルシューティングの状況でのみ変更されます。指定可能な値は 0 から 6 までです。これらの値の意味および `innodb_force_recovery` に関する重要な情報については、[セクション 15.21.2 「InnoDB のリカバリの強制的な実行」](#) を参照してください。

警告

InnoDB を起動してテーブルをダンプできるように、緊急時にはこの変数を 0 より大きい値にのみ設定してください。安全策として、`innodb_force_recovery` が 0 より大きい場合、InnoDB は `INSERT`、`UPDATE`、または `DELETE` 操作を回避します。`innodb_force_recovery` 設定が 4 以上の場合、InnoDB は読取り専用モードになります。

レプリケーションではレプリカステータスログが InnoDB テーブルに格納されるため、これらの制限により、レプリケーション管理コマンドがエラーで失敗する場合があります。

• `innodb_fsync_threshold`

コマンド行形式	<code>--innodb-fsync-threshold=#</code>
導入	8.0.13
システム変数	<code>innodb_fsync_threshold</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	$2^{*}64-1$

デフォルトでは、InnoDB が新しいログファイルやテーブルスペースファイルなどの新しいデータファイルを作成すると、ファイルはディスクにフラッシュされる前にオペレーティングシステムキャッシュに完全に書き込まれるため、大量のディスク書き込みアクティビティが一度に発生する可能性があります。オペレーティングシステムキャッシュから定期的にデータを強制的に小さいフラッシュするには、`innodb_fsync_threshold` 変数を使用してしきい値をバイト単位で定義します。バイトしきい値に達すると、オペレーティングシステムキャッシュの内容がディスクにフラッシュされます。デフォルト値の 0 では、デフォルトの動作が強制されます。つまり、ファイルがキャッシュに完全に書き込まれた後にのみ、データがディスクにフラッシュされます。

複数の MySQL インスタンスが同じストレージデバイスを使用している場合は、より小さい定期的なフラッシュを強制的に実行するためのしきい値を指定すると有益です。たとえば、新しい MySQL インスタンスとそれに関連付けられたデータファイルを作成すると、ディスク書き込みアクティビティが大きくなり、同じストレージデバイスを使用する他の MySQL インスタンスのパフォーマンスが低下する可能性があります。しきい値を構成すると、書き込みアクティビティでのこのようなサージの回避に役立ちます。

• `innodb_ft_aux_table`

システム変数	<code>innodb_ft_aux_table</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ

型	文字列
---	-----

FULLTEXT インデックスを含む **InnoDB** テーブルの修飾名を指定します。この変数は診断のために使用され、実行時にのみ設定できます。例:

```
SET GLOBAL innodb_ft_aux_table = 'test/t1';
```

この変数を `db_name/table_name` 形式の名前に設定すると、**INFORMATION_SCHEMA** テーブル **INNODB_FT_INDEX_TABLE**、**INNODB_FT_INDEX_CACHE**、**INNODB_FT_CONFIG**、**INNODB_FT_DELETED** および **INNODB_FT_BEING_DELETED** に、指定したテーブルの検索インデックスに関する情報が表示されます。

詳細は、[セクション15.15.4「InnoDB INFORMATION_SCHEMA FULLTEXT インデックステーブル」](#)を参照してください。

- [innodb_ft_cache_size](#)

コマンド行形式	<code>--innodb-ft-cache-size=#</code>
システム変数	innodb_ft_cache_size
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	8000000
最小値	1600000
最大値	80000000

InnoDB FULLTEXT 検索インデックスキャッシュに割り当てられたメモリー (バイト単位)。これは、**InnoDB FULLTEXT** インデックスの作成時にメモリー内に解析済ドキュメントを保持します。[innodb_ft_cache_size](#) のサイズ制限に達すると、インデックスの挿入および更新のみがディスクにコミットされます。[innodb_ft_cache_size](#) では、キャッシュサイズがテーブルごとに定義されます。すべてのテーブルにグローバルな制限を設定する方法については、[innodb_ft_total_cache_size](#) を参照してください。

詳細は、[InnoDB 全文インデックスキャッシュ](#)を参照してください。

- [innodb_ft_enable_diag_print](#)

コマンド行形式	<code>--innodb-ft-enable-diag-print[={OFF ON}]</code>
システム変数	innodb_ft_enable_diag_print
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

追加の全文検索 (FTS) 診断の出力を有効にするかどうかを指定します。このオプションは主に高度な FTS デバッグを目的としており、ほとんどのユーザーにとって重要ではありません。出力はエラーログに記録され、次のような情報が含まれています。

- FTS インデックス同期の進行状況 (FTS キャッシュ制限に達したとき)。例:

```
FTS SYNC for table test, deleted count: 100 size: 10000 bytes
SYNC words: 100
```

- FTS 最適化の進行状況。例:

```
FTS start optimize test
```

```
FTS_OPTIMIZE: optimize "mysql"
FTS_OPTIMIZE: processed "mysql"
```

- FTS インデックス構築の進行状況。例:

```
Number of doc processed: 1000
```

- FTS クエリーでは、クエリー解析のツリー、単語の重み、クエリーの処理時間、およびメモリーの使用状況が出力されます。例:

```
FTS Search Processing time: 1 secs: 100 millisec: row(s) 10000
Full Search Memory: 245666 (bytes), Row: 10000
```

- [innodb_ft_enable_stopword](#)

コマンド行形式	<code>--innodb-ft-enable-stopword[={OFF ON}]</code>
システム変数	innodb_ft_enable_stopword
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

インデックスの作成時に、一連のストップワードが InnoDB の FULLTEXT インデックスに関連付けられることを指定します。 [innodb_ft_user_stopword_table](#) オプションが設定されている場合は、そのテーブルからストップワードが取得されます。そうでなければ、[innodb_ft_server_stopword_table](#) オプションが設定されている場合は、そのテーブルからストップワードが取得されます。それ以外の場合は、組み込みのデフォルトストップワードセットが使用されます。

詳細は、[セクション12.10.4「全文ストップワード」](#)を参照してください。

- [innodb_ft_max_token_size](#)

コマンド行形式	<code>--innodb-ft-max-token-size=#</code>
システム変数	innodb_ft_max_token_size
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	84
最小値	10
最大値	84

InnoDB FULLTEXT インデックスに格納される単語の最大文字長。この値に制限を設定すると、実在の単語ではなく、検索語句になる可能性の低い英字の任意のコレクションや長いキーワードが省略されることで、インデックスのサイズが削減されるため、クエリーの速度が上がります。

詳細は、[セクション12.10.6「MySQL の全文検索の微調整」](#)を参照してください。

- [innodb_ft_min_token_size](#)

コマンド行形式	<code>--innodb-ft-min-token-size=#</code>
システム変数	innodb_ft_min_token_size
スコープ	グローバル
動的	いいえ

SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	3
最小値	0
最大値	16

InnoDB FULLTEXT インデックスに格納される単語の最小長。この値を大きくすると、英語の「a」や「to」などの検索コンテキストで重要ではない一般的な単語が省略されるため、インデックスのサイズが減り、クエリーが高速化されます。内容でCJK (中国語、日本語、韓国語) 文字セットが使用されている場合は、値 1 を指定します。

詳細は、[セクション12.10.6「MySQLの全文検索の微調整」](#)を参照してください。

- [innodb_ft_num_word_optimize](#)

コマンド行形式	--innodb-ft-num-word-optimize=#
システム変数	innodb_ft_num_word_optimize
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	2000
最小値	1000
最大値	10000

InnoDB の FULLTEXT インデックスでの各 OPTIMIZE TABLE 操作時に処理される単語数です。全文検索インデックスを含むテーブルへの一括挿入または一括更新操作では、すべての変更を組み込むために大量のインデックスのメンテナンスが必要となる可能性があるため、それぞれが最後に終了した場所から再開する一連の OPTIMIZE TABLE ステートメントを実行するとよいでしょう。

詳細は、[セクション12.10.6「MySQLの全文検索の微調整」](#)を参照してください。

- [innodb_ft_result_cache_limit](#)

コマンド行形式	--innodb-ft-result-cache-limit=#
システム変数	innodb_ft_result_cache_limit
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	2000000000
最小値	1000000
最大値	2**32-1

全文検索クエリーまたはスレッド当たりの InnoDB 全文検索クエリーの結果キャッシュ制限 (バイト単位で定義)。中間および最終的な InnoDB 全文検索クエリー結果はメモリー内で処理されます。innodb_ft_result_cache_limit を使用して全文検索のクエリー結果キャッシュにサイズ制限を設定し、InnoDB 全文検索のクエリー結果が非常に大きい場合 (数百万行や数百万行など) にメモリーを過剰に消費しないようにします。全文検索クエリーの処理時に、必要に応じてメモリーが割り当てられます。結果のキャッシュサイズ制限に達すると、クエリーで最大限に許可されるメモリー量を超えたことを示すエラーが返されます。

すべてのプラットフォームタイプおよびビットサイズに対する innodb_ft_result_cache_limit の最大値は、2**32-1 です。

- innodb_ft_server_stopword_table

コマンド行形式	<code>--innodb-ft-server-stopword-table=db_name/table_name</code>
システム変数	<code>innodb_ft_server_stopword_table</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	NULL

このオプションは、すべての InnoDB テーブルに対応した独自の InnoDB の FULLTEXT インデックスストップワードリストを指定する際に使用されます。特定の InnoDB テーブルに独自のストップワードリストを構成するには、`innodb_ft_user_stopword_table` を使用します。

`db_name/table_name` の形式で、`innodb_ft_server_stopword_table` をストップワードリストを含むテーブルの名前に設定します。

`innodb_ft_server_stopword_table` を構成する前に、ストップワードテーブルが存在する必要があります。FULLTEXT インデックスを作成する前に、`innodb_ft_enable_stopword` を有効にし、`innodb_ft_server_stopword_table` オプションを構成する必要があります。

ストップワードテーブルは、`value` という名前の単一の VARCHAR カラムを含む InnoDB テーブルである必要があります。

詳細は、[セクション12.10.4「全文ストップワード」](#) を参照してください。

- innodb_ft_sort_pll_degree

コマンド行形式	<code>--innodb-ft-sort-pll-degree=#</code>
システム変数	<code>innodb_ft_sort_pll_degree</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	2
最小値	1
最大値	32

`search index` の構築時に InnoDB FULLTEXT インデックスのテキストをインデックス付けおよびトークン化するためにパラレルで使用されるスレッドの数。

関連情報は、[セクション15.6.2.4「InnoDB FULLTEXT インデックス」](#) および `innodb_sort_buffer_size` を参照してください。

- innodb_ft_total_cache_size

コマンド行形式	<code>--innodb-ft-total-cache-size=#</code>
システム変数	<code>innodb_ft_total_cache_size</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer

デフォルト値	640000000
最小値	32000000
最大値	1600000000

すべてのテーブルの InnoDB 全文検索インデックスキャッシュに割り当てられた合計メモリー (バイト)。
FULLTEXT 検索インデックスを使用して多数のテーブルを作成すると、使用可能なメモリーの大部分が消費される可能性があります。**innodb_ft_total_cache_size** では、過剰なメモリー消費を回避するために、すべての全文検索インデックスに対してグローバルメモリー制限を定義します。インデックス操作によってグローバル制限に達すると、強制同期がトリガーされます。

詳細は、[InnoDB 全文インデックスキャッシュ](#)を参照してください。

- [innodb_ft_user_stopword_table](#)

コマンド行形式	--innodb-ft-user-stopword-table=db_name/table_name
システム変数	innodb_ft_user_stopword_table
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	NULL

このオプションは、特定のテーブルに独自の InnoDB の **FULLTEXT** インデックスストップワードリストを指定する際に使用されます。すべての InnoDB テーブル用に独自のストップワードリストを構成するには、[innodb_ft_server_stopword_table](#) を使用します。

db_name/table_name の形式で、[innodb_ft_user_stopword_table](#) をストップワードリストを含むテーブルの名前に設定します。

[innodb_ft_user_stopword_table](#) を構成する前に、ストップワードテーブルが存在する必要があります。**FULLTEXT** インデックスを作成する前に、[innodb_ft_enable_stopword](#) を有効にし、[innodb_ft_user_stopword_table](#) を構成する必要があります。

ストップワードテーブルは、**value** という名前の単一の **VARCHAR** カラムを含む InnoDB テーブルである必要があります。

詳細は、[セクション12.10.4「全文ストップワード」](#)を参照してください。

- [innodb_idle_flush_pct](#)

コマンド行形式	--innodb-idle-flush-pct=#
導入	8.0.18
システム変数	innodb_idle_flush_pct
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	100
最小値	0
最大値	100

InnoDB がアイドル状態の場合のページフラッシュを制限します。[innodb_idle_flush_pct](#) 値は、InnoDB で使用可能な I/O 操作数/秒を定義する [innodb_io_capacity](#) 設定の割合です。詳細は、[アイドル期間中のバッファフラッシュの制限](#)を参照してください。

- innodb_io_capacity

コマンド行形式	--innodb-io-capacity=#
システム変数	innodb_io_capacity
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	200
最小値	100
最大値 (64 ビットプラットフォーム)	2**64-1
最大値 (32 ビットプラットフォーム)	2**32-1

innodb_io_capacity 変数は、buffer pool からの flushing ページや change buffer からのデータのマージなど、InnoDB バックグラウンドタスクで使用可能な秒当たりの I/O 操作数 (IOPS) を定義します。

innodb_io_capacity 変数の構成の詳細は、[セクション15.8.7「InnoDB I/O Capacity の構成」](#) を参照してください。

- innodb_io_capacity_max

コマンド行形式	--innodb-io-capacity-max=#
システム変数	innodb_io_capacity_max
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	see description
最小値	100
最大値 (32 ビットプラットフォーム)	2**32-1
最大値 (Unix, 64 ビットプラットフォーム)	2**64-1
最大値 (Windows, 64 ビットプラットフォーム)	2**32-1

フラッシュアクティビティが遅れている場合、InnoDB は innodb_io_capacity 変数で定義されているよりも高い速度の I/O 操作/秒 (IOPS) で、より積極的にフラッシュできます。innodb_io_capacity_max 変数は、このような状況で InnoDB バックグラウンドタスクによって実行される IOPS の最大数を定義します。

innodb_io_capacity_max 変数の構成の詳細は、[セクション15.8.7「InnoDB I/O Capacity の構成」](#) を参照してください。

- innodb_limit_optimistic_insert_debug

コマンド行形式	--innodb-limit-optimistic-insert-debug=#
システム変数	innodb_limit_optimistic_insert_debug
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0

最大値	2**32-1
-----	---------

B-tree ページ当たりのレコード数を制限します。デフォルト値 0 は、制限が課されないことを意味します。このオプションは、デバッグサポートが `WITH_DEBUG CMake` オプションを使用してコンパイルされている場合にのみ使用できます。

- [innodb_lock_wait_timeout](#)

コマンド行形式	<code>--innodb-lock-wait-timeout=#</code>
システム変数	innodb_lock_wait_timeout
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	50
最小値	1
最大値	1073741824

行ロックが解除されるまで InnoDB トランザクションが待機する時間の長さ (秒単位) です。デフォルト値は 50 秒です。別の InnoDB トランザクションでロックされている行へのアクセスを試みるトランザクションは、行への書き込みアクセスを最大でこの秒数間待機してから、次のエラーを発行します。

```
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

ロック待機のタイムアウトが発生すると、(トランザクション全体ではなく) 現在のステートメントが **ロールバック** されます。トランザクション全体をロールバックするには、`--innodb-rollback-on-timeout` オプションを使用してサーバーを起動します。 [セクション15.21.4「InnoDB のエラー処理」](#) も参照してください。

高度にインタラクティブなアプリケーションまたは OLTP システムでは、ユーザーのフィードバックをすばやく表示したり、あとで処理するために更新をキューに入れたりするために、この値を小さくするとよいでしょう。長時間実行されるバックエンド操作 (その他の大規模な挿入操作や更新操作が完了するまで待機するデータウェアハウスでの変換ステップなど) では、この値を大きくするとよいでしょう。

`innodb_lock_wait_timeout` は、InnoDB の行ロックに適用されます。MySQL の **テーブルロック** は InnoDB 内部では発生せず、このタイムアウトはテーブルロックの待機には適用されません。

InnoDB ではデッドロックが即時に検出され、デッドロックされたトランザクションのいずれかがロールバックされるため、`innodb_deadlock_detect` が有効な場合 (デフォルト)、ロック待機タイムアウト値は `deadlocks` には適用されません。`innodb_deadlock_detect` が無効になっている場合、InnoDB はデッドロック発生時のトランザクションロールバックを `innodb_lock_wait_timeout` に依存します。 [セクション15.7.5.2「デッドロック検出」](#) を参照してください。

`innodb_lock_wait_timeout` は、実行時に `SET GLOBAL` または `SET SESSION` ステートメントとともに設定できます。`GLOBAL` 設定を変更するには、グローバルシステム変数を設定するのに十分な権限 ([セクション5.1.9.1「システム変数権限」](#) を参照) が必要であり、その後接続するすべてのクライアントの操作に影響します。任意のクライアントが `innodb_lock_wait_timeout` の `SESSION` 設定を変更でき、そのクライアントのみが影響を受けます。

- [innodb_log_buffer_size](#)

コマンド行形式	<code>--innodb-log-buffer-size=#</code>
システム変数	innodb_log_buffer_size
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer

デフォルト値	16777216
最小値	1048576
最大値	4294967295

ディスク上のログファイルに書き込む際に InnoDB で使用されるバッファのサイズ (バイト単位) です。デフォルトは 16M バイトです。大規模な `log buffer` では、トランザクション `commit` の前にログをディスクに書き込むことなく、大規模な `transactions` を実行できます。したがって、多数の行を更新、挿入、または削除するトランザクションの場合、ログバッファを大きくすると、ディスク I/O を節約できます。関連情報については、[メモリ構成](#)および[セクション8.5.4「InnoDB redo ロギングの最適化」](#)を参照してください。一般的な I/O チューニングのアドバイスについては、[セクション8.5.8「InnoDB ディスク I/O の最適化」](#)を参照してください。

- [innodb_log_checkpoint_fuzzy_now](#)

コマンド行形式	<code>--innodb-log-checkpoint-fuzzy-now[={OFF ON}]</code>
導入	8.0.13
システム変数	innodb_log_checkpoint_fuzzy_now
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

InnoDB にファジーチェックポイントの書き込みを強制するには、このデバッグオプションを有効にします。このオプションは、デバッグサポートが `WITH_DEBUG CMake` オプションを使用してコンパイルされている場合にのみ使用できます。

- [innodb_log_checkpoint_now](#)

コマンド行形式	<code>--innodb-log-checkpoint-now[={OFF ON}]</code>
システム変数	innodb_log_checkpoint_now
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

InnoDB にチェックポイントの書き込みを強制するには、このデバッグオプションを有効にします。このオプションは、デバッグサポートが `WITH_DEBUG CMake` オプションを使用してコンパイルされている場合にのみ使用できません。

- [innodb_log_checksums](#)

コマンド行形式	<code>--innodb-log-checksums[={OFF ON}]</code>
システム変数	innodb_log_checksums
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean

デフォルト値	ON
--------	----

redo ログページのチェックサムを有効または無効にします。

`innodb_log_checksums=ON` では、redo ログページの CRC-32C チェックサムアルゴリズムを有効にします。`innodb_log_checksums` が無効な場合、redo ログページのチェックサムフィールドの内容は無視されます。

redo ログヘッダーページおよび redo ログチェックポイントページのチェックサムは無効化されません。

- `innodb_log_compressed_pages`

コマンド行形式	<code>--innodb-log-compressed-pages[={OFF ON}]</code>
システム変数	<code>innodb_log_compressed_pages</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

`re-compressed pages` のイメージを redo log に書き込むかどうかを指定します。圧縮されたデータが変更されると、再圧縮が発生する場合があります。

`innodb_log_compressed_pages` は、リカバリ時に異なるバージョンの zlib 圧縮アルゴリズムが使用された場合に発生する可能性がある破損を防ぐために、デフォルトで有効になっています。zlib のバージョンが変更されないことが確実な場合は、`innodb_log_compressed_pages` を無効にして、圧縮データを変更するワークロードの redo ログ生成を減らすことができます。

`innodb_log_compressed_pages` の有効化または無効化の影響を測定するには、同じワークロードで両方の設定の redo ログ生成を比較します。redo ログ生成の測定オプションには、`SHOW ENGINE INNODB STATUS` 出力の LOG セクションでの `Log sequence number (LSN)` の監視、または redo ログファイルに書き込まれたバイト数の `InnoDB_os_log_written` ステータスの監視が含まれます。

関連情報については、[セクション15.9.1.6「OLTP ワークロードの圧縮」](#)を参照してください。

- `innodb_log_file_size`

コマンド行形式	<code>--innodb-log-file-size=#</code>
システム変数	<code>innodb_log_file_size</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	50331648
最小値	4194304
最大値	512GB / <code>innodb_log_files_in_group</code>

ロググループ内の各ログファイルのサイズ (バイト単位) です。ログファイルを結合したサイズ (`innodb_log_file_size * innodb_log_files_in_group`) は、512G バイトよりもわずかに小さい最大値を上回ることができません。たとえば、255 GB のログファイルのペアは制限に近づいていますが、それを超えていません。デフォルト値は 48M バイトです。

一般に、ログファイルの合計サイズは、サーバーがワークロードアクティビティのピークおよびトラブルをスムーズにできる十分な大きさである必要があります。これは、書き込みアクティビティを 1 時間以上処理するための十分な redo ログ領域があることを意味することがよくあります。この値が大きいくほど、バッファプールに必要な

チェックポイントフラッシュアクティビティが少なくなり、ディスク I/O が節約されます。 ログファイルが大きいほど、[crash recovery](#) も遅くなります。

最小の `innodb_log_file_size` は 4MB です。

関連情報については、[redo ログファイル構成](#)を参照してください。 一般的な I/O チューニングのアドバイスについては、[セクション8.5.8「InnoDB ディスク I/O の最適化」](#)を参照してください。

`innodb_dedicated_server` が有効な場合、`innodb_log_file_size` 値は明示的に定義されていなければ自動的に構成されます。 詳細は、[セクション15.8.12「専用 MySQL Server の自動構成の有効化」](#)を参照してください。

- `innodb_log_files_in_group`

コマンド行形式	<code>--innodb-log-files-in-group=#</code>
システム変数	<code>innodb_log_files_in_group</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	2
最小値	2
最大値	100

[ロググループ](#)内の[ログファイル](#)の数です。 InnoDB はファイルに輪状に書き込みをします。 デフォルト (推奨) 値は 2 です。 ファイルの場所は、`innodb_log_group_home_dir` によって指定されます。 ログファイルを結合したサイズ (`innodb_log_file_size * innodb_log_files_in_group`) は、最大で 512G バイトにすることができます。

関連情報については、[redo ログファイル構成](#)を参照してください。

- `innodb_log_group_home_dir`

コマンド行形式	<code>--innodb-log-group-home-dir=dir_name</code>
システム変数	<code>innodb_log_group_home_dir</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ディレクトリ名

InnoDB の [Redo ログファイル](#)へのディレクトリパスです。 この数は、`innodb_log_files_in_group` で指定されます。 どの InnoDB ログ変数も指定しない場合は、デフォルトで、MySQL データディレクトリ内に `ib_logfile0` および `ib_logfile1` という名前の 2 つのファイルが作成されます。 ログファイルのサイズは、`innodb_log_file_size` システム変数によって指定されます。

関連情報については、[redo ログファイル構成](#)を参照してください。

- `innodb_log_spin_cpu_abs_lwm`

コマンド行形式	<code>--innodb-log-spin-cpu-abs-lwm=#</code>
システム変数	<code>innodb_log_spin_cpu_abs_lwm</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer

デフォルト値	80
最小値	0
最大値	4294967295

フラッシュされた redo の待機中にユーザースレッドがスピンしなくなる CPU 使用率の最小量を定義します。この値は、CPU コア使用率の合計として表されます。たとえば、80 のデフォルト値は、単一の CPU コアの 80% です。マルチコアプロセッサを搭載したシステムでは、150 の値は、1 つの CPU コアの 100% 使用率と 2 つ目の CPU コアの 50% 使用率を表します。

関連情報については、[セクション8.5.4「InnoDB redo ロギングの最適化」](#)を参照してください。

- [innodb_log_spin_cpu_pct_hwm](#)

コマンド行形式	<code>--innodb-log-spin-cpu-pct-hwm=#</code>
システム変数	innodb_log_spin_cpu_pct_hwm
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	50
最小値	0
最大値	100

フラッシュされた redo の待機中にユーザースレッドがスピンしなくなる CPU 使用率の最大量を定義します。この値は、すべての CPU コアの合計処理能力の割合として表されます。デフォルト値は 50% です。たとえば、2 つの CPU コアの 100% 使用率は、4 つの CPU コアを持つサーバーでの CPU 処理能力の合計の 50% です。

[innodb_log_spin_cpu_pct_hwm](#) 変数は、プロセッサアフィニティを考慮します。たとえば、サーバーに 48 個のコアがあり、`mysqld` プロセスが 4 個の CPU コアにのみ固定されている場合、他の 44 個の CPU コアは無視されません。

関連情報については、[セクション8.5.4「InnoDB redo ロギングの最適化」](#)を参照してください。

- [innodb_log_wait_for_flush_spin_hwm](#)

コマンド行形式	<code>--innodb-log-wait-for-flush-spin-hwm=#</code>
システム変数	innodb_log_wait_for_flush_spin_hwm
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	400
最小値	0
最大値 (64 ビットプラットフォーム)	$2^{64}-1$
最大値 (32 ビットプラットフォーム)	$2^{32}-1$

フラッシュされた redo の待機中にユーザースレッドがスピンしなくなる最大平均ログフラッシュ時間を定義します。デフォルト値は 400 マイクロ秒です。

関連情報については、[セクション8.5.4「InnoDB redo ロギングの最適化」](#)を参照してください。

- [innodb_log_write_ahead_size](#)

コマンド行形式	<code>--innodb-log-write-ahead-size=#</code>
システム変数	innodb_log_write_ahead_size
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	8192
最小値	512 (log file block size)
最大値	Equal to innodb_page_size

redo ログの先行書込みブロックサイズをバイト単位で定義します。「read-on-write」を回避するには、オペレーティングシステムまたはファイルシステムのキャッシュブロックサイズと一致するように [innodb_log_write_ahead_size](#) を設定します。デフォルト設定は 8192 バイトです。読取り/書込みは、redo ログの先行書込みブロックサイズとオペレーティングシステムまたはファイルシステムのキャッシュブロックサイズが一致しないために、redo ログブロックがオペレーティングシステムまたはファイルシステムに完全にキャッシュされない場合に発生します。

[innodb_log_write_ahead_size](#) の有効な値は、InnoDB ログファイルのブロックサイズ (2^n) の倍数です。最小値は、InnoDB ログファイルのブロックサイズ (512) です。最小値が指定されている場合、ライトアヘッドは発生しません。最大値は [innodb_page_size](#) 値と同じです。[innodb_log_write_ahead_size](#) に [innodb_page_size](#) 値より大きい値を指定すると、[innodb_log_write_ahead_size](#) 設定は [innodb_page_size](#) 値に切り捨てられます。

オペレーティングシステムまたはファイルシステムのキャッシュブロックサイズに対する [innodb_log_write_ahead_size](#) 値の設定が低すぎると、「read-on-write」が発生します。値を高く設定しすぎると、一度に複数のブロックが書き込まれるため、ログファイル書込みの `fsync` パフォーマンスにわずかな影響を与える可能性があります。

関連情報については、[セクション8.5.4「InnoDB redo ロギングの最適化」](#)を参照してください。

- [innodb_log_writer_threads](#)

コマンド行形式	<code>--innodb-log-writer-threads[={OFF ON}]</code>
導入	8.0.22
システム変数	innodb_log_writer_threads
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

ログバッファからシステムバッファに redo ログレコードを書き込み、システムバッファを redo ログファイルにフラッシュするための専用のログライタースレッドを有効にします。専用ログライタースレッドを使用すると、同時実行性の高いシステムのパフォーマンスを向上させることができますが、同時実行性の低いシステムでは、専用ログライタースレッドを無効にすると、パフォーマンスが向上します。

詳細は、[セクション8.5.4「InnoDB redo ロギングの最適化」](#)を参照してください。

- [innodb_lru_scan_depth](#)

コマンド行形式	<code>--innodb-lru-scan-depth=#</code>
システム変数	innodb_lru_scan_depth

スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1024
最小値	100
最大値 (64 ビットプラットフォーム)	2**64-1
最大値 (32 ビットプラットフォーム)	2**32-1

InnoDB のバッファプールでのフラッシュ操作のアルゴリズムおよびヒューリスティクスに影響を与えるパラメータです。主に、I/O インテンシブなワークロードを調整するパフォーマンスの専門家が関心を持つものです。バッファプールインスタンスごとに、フラッシュする dirty pages を検索するページクリーナスレッドスキャンをバッファプール LRU ページにリストする距離を指定します。これは、1 秒に 1 回実行されるバックグラウンド操作です。

デフォルトより小さい設定は、通常、ほとんどのワークロードに適しています。必要以上の値を指定すると、パフォーマンスに影響する可能性があります。通常のワークロードでスベア I/O 容量がある場合のみ、値を増やすことを検討してください。逆に、書き込み集中型のワークロードが I/O の容量を満たしている場合は、特に大きなバッファプールの場合に値を減らします。

`innodb_lru_scan_depth` をチューニングする場合は、小さい値から始めて、ゼロの空きページが表示されることがほとんどないという目標で設定を上方に構成します。また、`innodb_lru_scan_depth * innodb_buffer_pool_instances` は毎秒ページクリーナスレッドによって実行される作業量を定義するため、バッファプールインスタンスの数を変更するときに `innodb_lru_scan_depth` を調整することを検討してください。

関連情報については、[セクション15.8.3.5「バッファプールのフラッシュの構成」](#)を参照してください。一般的な I/O チューニングのアドバイスについては、[セクション8.5.8「InnoDB ディスク I/O の最適化」](#)を参照してください。

- `innodb_max_dirty_pages_pct`

コマンド行形式	<code>--innodb-max-dirty-pages-pct=#</code>
システム変数	<code>innodb_max_dirty_pages_pct</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	数値
デフォルト値	90
最小値	0
最大値	99.99

InnoDB は、**ダーティーページ**の割合がこの値を超えないように、**バッファプール**からデータを**フラッシュ**しようと試みます。

`innodb_max_dirty_pages_pct` 設定は、フラッシュアクティビティーのターゲットを確立します。フラッシュの頻度には影響を与えません。フラッシュの頻度の管理については、[セクション15.8.3.5「バッファプールのフラッシュの構成」](#)を参照してください。

関連情報については、[セクション15.8.3.5「バッファプールのフラッシュの構成」](#)を参照してください。一般的な I/O チューニングのアドバイスについては、[セクション8.5.8「InnoDB ディスク I/O の最適化」](#)を参照してください。

- innodb_max_dirty_pages_pct_lwm

コマンド行形式	--innodb-max-dirty-pages-pct-lwm=#
システム変数	innodb_max_dirty_pages_pct_lwm
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	数値
デフォルト値	10
最小値	0
最大値	99.99

ダーティページ率を制御するために事前フラッシュが有効になる [dirty pages](#) の割合を表す最低水位標を定義します。値 0 を指定すると、事前フラッシュ動作が完全に無効になります。詳細は、[セクション15.8.3.5「バッファープールのフラッシュの構成」](#)を参照してください。

- innodb_max_purge_lag

コマンド行形式	--innodb-max-purge-lag=#
システム変数	innodb_max_purge_lag
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	4294967295

必要な最大パーシラグを定義します。この値を超えると、INSERT、UPDATE および DELETE 操作に遅延が課され、ページが捕捉されるまでの時間が許可されます。デフォルト値は 0 です。これは、最大パーシラグおよび遅延がないことを意味します。

詳細は、[セクション15.8.9「パーシ構成」](#)を参照してください。

- innodb_max_purge_lag_delay

コマンド行形式	--innodb-max-purge-lag-delay=#
システム変数	innodb_max_purge_lag_delay
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	10000000

[innodb_max_purge_lag](#) しきい値を超えた場合に課される遅延の最大遅延をマイクロ秒単位で指定します。指定された [innodb_max_purge_lag_delay](#) 値は、[innodb_max_purge_lag](#) 式で計算された遅延期間の上限です。

詳細は、[セクション15.8.9「パーシ構成」](#)を参照してください。

- innodb_max_undo_log_size

コマンド行形式	--innodb-max-undo-log-size=#
システム変数	innodb_max_undo_log_size
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1073741824
最小値	10485760
最大値	2**64-1

undo テーブルスペースのしきい値サイズを定義します。undo テーブルスペースがしきい値を超えると、[innodb_undo_log_truncate](#) が有効になっているときに切り捨てられるようにマークできます。デフォルト値は 1073741824 バイト (1024 MiB) です。

詳細は、[undo テーブルスペースの切捨て](#) を参照してください。

- innodb_merge_threshold_set_all_debug

コマンド行形式	--innodb-merge-threshold-set-all-debug=#
システム変数	innodb_merge_threshold_set_all_debug
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	50
最小値	1
最大値	50

ディクショナリキャッシュに現在存在するすべてのインデックスの現在の [MERGE_THRESHOLD](#) 設定をオーバーライドするインデックスページのページフルパーセント値を定義します。このオプションは、デバッグサポートが [WITH_DEBUG CMake](#) オプションを使用してコンパイルされている場合にのみ使用できます。関連情報については、[セクション15.8.11「インデックスページのマージしきい値の構成」](#) を参照してください。

- innodb_monitor_disable

コマンド行形式	--innodb-monitor-disable={counter module pattern all}
システム変数	innodb_monitor_disable
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列

InnoDB [metrics counters](#) を無効にします。カウンタデータは [INFORMATION_SCHEMA.INNODB_METRICS](#) テーブルを使用してクエリーすることができます。使用法については、[セクション15.15.6「InnoDB INFORMATION_SCHEMA メトリックテーブル」](#) を参照してください。

[innodb_monitor_disable='latch'](#) は、[SHOW ENGINE INNODB MUTEX](#) の統計収集を無効にします。詳細は、[セクション13.7.7.15「SHOW ENGINE ステートメント」](#) を参照してください。

- innodb_monitor_enable

コマンド行形式	<code>--innodb-monitor-enable={counter module pattern all}</code>
システム変数	innodb_monitor_enable
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列

[InnoDB metrics counters](#) を有効にします。カウンタデータは `INFORMATION_SCHEMA.INNODB_METRICS` テーブルを使用してクエリーすることができます。用法については、[セクション15.15.6「InnoDB INFORMATION_SCHEMA メトリックテーブル」](#)を参照してください。

`innodb_monitor_enable='latch'`では、`SHOW ENGINE INNODB MUTEX`の統計収集が可能です。詳細は、[セクション13.7.7.15「SHOW ENGINE ステートメント」](#)を参照してください。

- [innodb_monitor_reset](#)

コマンド行形式	<code>--innodb-monitor-reset={counter module pattern all}</code>
システム変数	innodb_monitor_reset
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	<code>empty string</code>
有効な値	<code>counter</code> <code>module</code> <code>pattern</code> <code>all</code>

[InnoDB metrics counters](#) のカウント値をゼロにリセットします。カウンタデータは `INFORMATION_SCHEMA.INNODB_METRICS` テーブルを使用してクエリーすることができます。用法については、[セクション15.15.6「InnoDB INFORMATION_SCHEMA メトリックテーブル」](#)を参照してください。

`innodb_monitor_reset='latch'`は、`SHOW ENGINE INNODB MUTEX`によって報告された統計をリセットします。詳細は、[セクション13.7.7.15「SHOW ENGINE ステートメント」](#)を参照してください。

- [innodb_monitor_reset_all](#)

コマンド行形式	<code>--innodb-monitor-reset-all={counter module pattern all}</code>
システム変数	innodb_monitor_reset_all
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	<code>empty string</code>
有効な値	<code>counter</code> <code>module</code> <code>pattern</code>

all

InnoDB [metrics counters](#) のすべての値 (最小、最大など) をリセットします。カウンタデータは `INFORMATION_SCHEMA.INNODB_METRICS` テーブルを使用してクエリーすることができます。用法については、[セクション15.15.6「InnoDB INFORMATION_SCHEMA メトリックテーブル」](#) を参照してください。

- [innodb_numa_interleave](#)

コマンド行形式	<code>--innodb-numa-interleave[={OFF ON}]</code>
システム変数	innodb_numa_interleave
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

NUMA インターリーブメモリーポリシーを有効にして、InnoDB バッファプールを割り当てます。[innodb_numa_interleave](#) が有効な場合、NUMA メモリーポリシーは `mysqld` プロセスに対して `MPOL_INTERLEAVE` に設定されます。InnoDB バッファプールが割り当てられると、NUMA メモリーポリシーは `MPOL_DEFAULT` に戻されます。[innodb_numa_interleave](#) オプションを使用できるようにするには、NUMA 対応の Linux システムで MySQL をコンパイルする必要があります。

CMake では、現在のプラットフォームに NUMA サポートがあるかどうかに基づいて、デフォルトの `WITH_NUMA` 値が設定されます。詳細は、[セクション2.9.7「MySQL ソース構成オプション」](#) を参照してください。

- [innodb_old_blocks_pct](#)

コマンド行形式	<code>--innodb-old-blocks-pct=#</code>
システム変数	innodb_old_blocks_pct
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	37
最小値	5
最大値	95

古いブロックサブリストで使用される InnoDB のバッファプールの概算割合を指定します。値の範囲は 5 から 95 です。デフォルト値は 37 (つまり、プールの 3/8) です。多くの場合、[innodb_old_blocks_time](#) と組み合わせて使用されます。

詳細は、[セクション15.8.3.3「バッファプールをスキャンに耐えられるようにする」](#) を参照してください。バッファプールの管理、LRU アルゴリズム、および [eviction](#) ポリシーについては、[セクション15.5.1「バッファプール」](#) を参照してください。

- [innodb_old_blocks_time](#)

コマンド行形式	<code>--innodb-old-blocks-time=#</code>
システム変数	innodb_old_blocks_time
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ

型	Integer
デフォルト値	1000
最小値	0
最大値	2**32-1

ゼロ以外の値は、[full table scan](#) の実行中など、短い期間のみ参照されるデータによって入力される [buffer pool](#) から保護します。この値を大きくすると、テーブルの完全スキャンがバッファプール内にキャッシュされたデータとやりとりすることからさらに保護されます。

新しいサブリストに移動する前に、古い [sublist](#) に挿入されたブロックが最初のアクセス後も保持される必要がある時間をミリ秒単位で指定します。値を 0 にすると、古いサブリストに挿入されたブロックは、挿入後にどのくらいの期間でアクセスが発生するのには関係なく、最初のアクセスの直後に新しいサブリストに移動します。値が 0 より大きい場合、最初のアクセス後に少なくとも何ミリ秒もアクセスが発生するまで、ブロックは古いサブリストに残ります。たとえば、1000 の値では、ブロックは最初のアクセス後、それらが新しいサブリストに移動される資格を得るまで、1 秒間古いサブリストにとどまります。

デフォルト値は 1000 です。

多くの場合、この変数は [innodb_old_blocks_pct](#) と組み合わせて使用されます。詳細は、[セクション15.8.3.3「バッファプールをスキャンに耐えられるようにする」](#) を参照してください。バッファプールの管理、[LRU アルゴリズム](#)、および [eviction](#) ポリシーについては、[セクション15.5.1「バッファプール」](#) を参照してください。

- [innodb_online_alter_log_max_size](#)

コマンド行形式	<code>--innodb-online-alter-log-max-size=#</code>
システム変数	innodb_online_alter_log_max_size
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	134217728
最小値	65536
最大値	2**64-1

InnoDB テーブルの [online DDL](#) 操作中に使用される一時ログファイルのサイズの上限をバイト単位で指定します。作成されるインデックスまたは変更されるテーブルごとに、このようなログファイルが 1 つ存在します。このログファイルには、DDL 操作時にテーブルで挿入、更新、または削除されたデータが格納されます。一時ログファイルは、[innodb_sort_buffer_size](#) の値で必要になったときに、最大で [innodb_online_alter_log_max_size](#) で指定された最大値まで拡張されます。一時ログファイルが上限サイズを超えると、[ALTER TABLE](#) 操作は失敗し、コミットされていないすべての同時 DML 操作がロールバックされます。したがって、このオプションの値を大きくすると、オンライン DDL 操作中に発生する DML が増えますが、ログからデータを適用するためにテーブルがロックされているときの DDL 操作の終了時の期間も長くなります。

- [innodb_open_files](#)

コマンド行形式	<code>--innodb-open-files=#</code>
システム変数	innodb_open_files
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動サイズ設定を示します。このリテラル値を割り当てないでください)

最小値	10
最大値	4294967295

この変数は、複数の [InnoDB tablespaces](#) を使用する場合にのみ関連します。MySQL で一度に開いたままにできる [.ibd ファイル](#) の最大数が指定されます。最小値は 10 です。デフォルト値は、[innodb_file_per_table](#) が有効になっていない場合は 300 で、それ以外の場合は 300 以上および [table_open_cache](#) です。

[.ibd](#) ファイルで使用されるファイルディスクリプタは、[InnoDB](#) テーブルでのみ使用されます。これらは、[open_files_limit](#) システム変数で指定されたものとは独立しており、テーブルキャッシュの操作には影響しません。一般的な I/O チューニングのアドバイスについては、[セクション8.5.8「InnoDB ディスク I/O の最適化」](#)を参照してください。

- [innodb_optimize_fulltext_only](#)

コマンド行形式	<code>--innodb-optimize-fulltext-only[={OFF ON}]</code>
システム変数	innodb_optimize_fulltext_only
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

[InnoDB](#) テーブルでの [OPTIMIZE TABLE](#) の動作方法を変更します。[FULLTEXT](#) インデックスを含む [InnoDB](#) テーブルのメンテナンス操作時に、一時的に有効にするために使用されます。

デフォルトでは、[OPTIMIZE TABLE](#) はテーブルの [clustered index](#) のデータを再編成します。このオプションを有効にすると、[OPTIMIZE TABLE](#) はテーブルデータの再編成をスキップし、かわりに [InnoDB FULLTEXT](#) インデックスに対して新しく追加、削除および更新されたトークンデータを処理します。詳細は、[InnoDB 全文インデックスの最適化](#)を参照してください。

- [innodb_page_cleaners](#)

コマンド行形式	<code>--innodb-page-cleaners=#</code>
システム変数	innodb_page_cleaners
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	4
最小値	1
最大値	64

バッファプールインスタンスからダーティーページをフラッシュするページクリーナースレッドの数。ページクリーナースレッドは、フラッシュリストおよび LRU フラッシュを実行します。ページクリーナースレッドが複数ある場合、バッファプールインスタンスごとにバッファプールのフラッシュタスクがアイドル状態のページクリーナースレッドにディスパッチされます。[innodb_page_cleaners](#) のデフォルト値は 4 です。ページクリーナースレッドの数

がバッファプールインスタンスの数を超えると、`innodb_page_cleaners` は自動的に `innodb_buffer_pool_instances` と同じ値に設定されます。

ダーティページをバッファプールインスタンスからデータファイルにフラッシュするときにワークロードが書き込み IO バインドされている場合、およびシステムハードウェアに使用可能な容量がある場合は、ページクリーナスレッドの数を増やすと書き込み IO スループットの向上に役立つことがあります。

マルチスレッドページクリーナのサポートは、停止フェーズおよびリカバリフェーズまで拡張されています。

`setpriority()` システムコールは、サポートされている Linux プラットフォームで使用され、`mysqld` 実行ユーザーが `page_cleaner` スレッドに他の MySQL および InnoDB スレッドよりも優先順位を与えることを認可されている場合、ページフラッシュが現在のワークロードに対応できるようにします。`setpriority()` のサポートは、次の InnoDB 起動メッセージで示されます:

```
[Note] InnoDB: If the mysqld execution user is authorized, page cleaner
thread priority can be changed. See the man page of setpriority().
```

サーバーの起動および停止が `systemd` によって管理されていないシステムでは、`/etc/security/limits.conf` で `mysqld` 実行ユーザー認可を構成できます。たとえば、`mysqld` が `mysql` ユーザーで実行されている場合、次の行を `/etc/security/limits.conf` に追加することで `mysql` ユーザーを認可できます:

```
mysql      hard nice -20
mysql      soft nice -20
```

`systemd` 管理対象システムの場合は、ローカライズされた `systemd` 構成ファイルで `LimitNICE=-20` を指定することで同じことを実現できます。たとえば、`/etc/systemd/system/mysqld.service.d/override.conf` で `override.conf` という名前のファイルを作成し、次のエントリを追加します:

```
[Service]
LimitNICE=-20
```

`override.conf` を作成または変更した後、`systemd` 構成をリロードし、MySQL サービスを再起動するように `systemd` に指示します:

```
systemctl daemon-reload
systemctl restart mysqld # RPM platforms
systemctl restart mysql  # Debian platforms
```

ローカライズされた `systemd` 構成ファイルの使用の詳細は、MySQL の `systemd` の構成 を参照してください。

`mysqld` 実行ユーザーを認可した後、`cat` コマンドを使用して、`mysqld` プロセスに構成されている `Nice` 制限を確認します:

```
shell> cat /proc/mysqld_pid/limits | grep nice
Max nice priority      18446744073709551596 18446744073709551596
```

- `innodb_page_size`

コマンド行形式	<code>--innodb-page-size=#</code>
システム変数	<code>innodb_page_size</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	16384
有効な値	4096
	8192
	16384
	32768

65536

InnoDB tablespaces 用の `page size` を指定します。値はバイト単位または KB 単位で指定できます。たとえば、16K バイトのページサイズ値は 16384、16K バイト、または 16K と指定できます。

`innodb_page_size` は、MySQL インスタンスの初期化前にもみ構成でき、後で変更することはできません。値を指定しない場合、インスタンスはデフォルトのページサイズを使用して初期化されます。セクション15.8.1「InnoDB の起動構成」を参照してください。

32KB と 64KB の両方のページサイズで、行の最大長は約 16000 バイトです。`innodb_page_size` が 32KB または 64KB に設定されている場合、`ROW_FORMAT=COMPRESSED` はサポートされません。`innodb_page_size=32KB` の場合、エクステントサイズは 2MB です。`innodb_page_size=64KB` の場合、エクステントサイズは 4MB です。32KB または 64KB のページサイズを使用する場合は、`innodb_log_buffer_size` を 16M (デフォルト) 以上に設定する必要があります。

デフォルトの 16KB ページサイズ以上は、`workloads` の広範囲、特にバルク更新を伴うテーブルスキャンおよび DML 操作を含むクエリに適しています。単一ページに多数の行が含まれている場合、競合が問題になる可能性がある多数の小さい書込みを含む OLTP ワークロードでは、ページサイズを小さくする方が効率的です。ページを小さくすると、一般に小さなブロックサイズが使用される SSD ストレージデバイスの効率性が高くなる可能性もあります。InnoDB のページサイズをストレージデバイスのブロックサイズに近づけると、ディスクに再度書き込まれる未変更データの量が最小限になります。

最初のシステムテーブルスペースデータファイル (`ibdata1`) の最小ファイルサイズは、`innodb_page_size` の値によって異なります。詳細は、`innodb_data_file_path` オプションの説明を参照してください。

一般的な I/O チューニングのアドバイスについては、セクション8.5.8「InnoDB ディスク I/O の最適化」を参照してください。

- `innodb_parallel_read_threads`

コマンド行形式	<code>--innodb-parallel-read-threads=#</code>
導入	8.0.14
システム変数	<code>innodb_parallel_read_threads</code>
スコープ	セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	4
最小値	1
最大値	256

パラレルクラスタインデックス読取りに使用できるスレッドの数を定義します。パーティションのパラレルスキャンは、MySQL 8.0.17 でサポートされています。パラレル読取りスレッドを使用すると、`CHECK TABLE` のパフォーマンスを向上できます。InnoDB は、`CHECK TABLE` 操作中にクラスタ化されたインデックスを 2 回読み取ります。2 番目の読取りはパラレルで実行できます。この機能は、セカンダリインデックススキャンには適用されません。パラレルクラスタインデックス読取りを実行するには、`innodb_parallel_read_threads` セッション変数を 1 より大きい値に設定する必要があります。パラレルクラスタインデックス読取りの実行に使用されるスレッドの実際の数は、`innodb_parallel_read_threads` 設定またはスキャンするインデックスサブツリーの数 (いずれか小さい方) によって決まります。スキャン中にバッファプールに読み取られたページは、空きバッファプールページが必要となるときにすぐに破棄できるように、バッファプール LRU リストの末尾に保持されます。

MySQL 8.0.17 では、パラレル読取りスレッドの最大数 (256) は、すべてのクライアント接続のスレッドの合計数です。スレッド制限に達すると、接続は単一スレッドの使用にフォールバックします。

- [innodb_print_all_deadlocks](#)

コマンド行形式	<code>--innodb-print-all-deadlocks[={OFF ON}]</code>
システム変数	innodb_print_all_deadlocks
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

このオプションを有効にすると、[mysqld エラーログ](#)に、InnoDB のユーザートランザクション内のすべてのデッドロックに関する情報が記録されます。それ以外の場合は、[SHOW ENGINE INNODB STATUS](#) コマンドを使用すると、最後のデッドロックに関する情報のみが表示されます。状況は InnoDB によってただちに検出され、いずれかのトランザクションが自動的にロールバックされるため、場合によっては InnoDB デッドロックは必ずしも問題ではありません。このオプションを使用して、ロールバックを検出してその操作を再試行するための適切なエラー処理ロジックがアプリケーションにない場合にデッドロックが発生する理由をトラブルシューティングできます。多数のデッドロックが発生する場合は、各トランザクションが同じ順序でテーブルにアクセスするように (これにより、デッドロックの状況が回避されます)、複数のテーブルに対して [DML](#) または [SELECT ... FOR UPDATE](#) ステートメントを発行するトランザクションを再構築する必要があることを示している可能性があります。

関連情報については、[セクション15.7.5「InnoDB のデッドロック」](#)を参照してください。

- [innodb_print_ddl_logs](#)

コマンド行形式	<code>--innodb-print-ddl-logs[={OFF ON}]</code>
システム変数	innodb_print_ddl_logs
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

このオプションを有効にすると、MySQL は DDL ログを `stderr` に書き込みます。詳細は、[DDL ログの表示](#)を参照してください。

- [innodb_purge_batch_size](#)

コマンド行形式	<code>--innodb-purge-batch-size=#</code>
システム変数	innodb_purge_batch_size
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	300
最小値	1
最大値	5000

[history list](#) から一度に解析および処理をページする undo ログページの数を実験的に定義します。マルチスレッドページ構成では、コーディネータページスレッドは [innodb_purge_batch_size](#) を [innodb_purge_threads](#) で除算し、その数の

ページを各パーシスレッドに割り当てます。 `innodb_purge_batch_size` 変数では、undo ログを 128 回反復するたびに消去する undo ログページの数も定義されます。

`innodb_purge_batch_size` オプションは、`innodb_purge_threads` 設定と組み合わせた高度なパフォーマンスチューニングを目的としています。ほとんどのユーザーは、`innodb_purge_batch_size` をデフォルト値から変更する必要はありません。

関連情報については、[セクション15.8.9「ページ構成」](#)を参照してください。

- `innodb_purge_threads`

コマンド行形式	<code>--innodb-purge-threads=#</code>
システム変数	<code>innodb_purge_threads</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	4
最小値	1
最大値	32

InnoDB purge 操作専用のバックグラウンドスレッドの数。この値を大きくすると、追加のパーシスレッドが作成されるため、DML 操作が複数のテーブルで実行されるシステムの効率が向上します。

関連情報については、[セクション15.8.9「ページ構成」](#)を参照してください。

- `innodb_purge_rseg_truncate_frequency`

コマンド行形式	<code>--innodb-purge-rseg-truncate-frequency=#</code>
システム変数	<code>innodb_purge_rseg_truncate_frequency</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	128
最小値	1
最大値	128

ページが起動された回数に関して、パーシシステムがロールバックセグメントを解放する頻度を定義します。undo テーブルスペースは、ロールバックセグメントが解放されるまで切り捨てられません。通常、パーシシステムは、ページが起動される 128 回ごとにロールバックセグメントを解放します。デフォルト値は 128 です。この値を減らすと、パーシスレッドがロールバックセグメントを解放する頻度が高くなります。

`innodb_purge_rseg_truncate_frequency` は、`innodb_undo_log_truncate` での使用を目的としています。詳細は、[undo テーブルスペースの切捨て](#)を参照してください。

- `innodb_random_read_ahead`

コマンド行形式	<code>--innodb-random-read-ahead[={OFF ON}]</code>
システム変数	<code>innodb_random_read_ahead</code>
スコープ	グローバル
動的	はい

SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

InnoDB の I/O を最適化するために、ランダムな先読み技術を有効にします。

様々なタイプの先読みリクエストのパフォーマンスに関する考慮事項の詳細は、[セクション15.8.3.4「InnoDB バッファープールのプリフェッチ \(先読み\) の構成」](#)を参照してください。一般的な I/O チューニングのアドバイスについては、[セクション8.5.8「InnoDB ディスク I/O の最適化」](#)を参照してください。

- innodb_read_ahead_threshold

コマンド行形式	--innodb-read-ahead-threshold=#
システム変数	innodb_read_ahead_threshold
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	56
最小値	0
最大値	64

バッファープールにページをプリフェッチする際に InnoDB で使用される線形の先読みの感度を制御します。InnoDB が少なくとも `innodb_read_ahead_threshold` ページをエクステント (64 ページ) から連続して読み取る場合は、次のエクステント全体の非同期読み取りが開始されます。許可される値の範囲は 0 から 64 までです。値 0 は先読みを無効にします。デフォルトの 56 では、InnoDB は次のエクステント全体の非同期読み取りを開始するために、少なくとも 56 ページをエクステントから連続して読み取る必要があります。

先読みメカニズムを使用して読み取られるページの数と、アクセスされずにバッファープールから削除されるページ数を把握しておくこと、`innodb_read_ahead_threshold` 設定を微調整する場合に役立ちます。SHOW ENGINE INNODB STATUS 出力には、`InnoDB_buffer_pool_read_ahead` および `InnoDB_buffer_pool_read_ahead_evicted` のグローバルステータス変数からのカウンタ情報が表示されます。これらの変数は、先読みリクエストによって `buffer pool` に取り込まれたページ数と、アクセスされたことなくバッファープールから `evicted` のそのようなページ数をそれぞれレポートします。ステータス変数は、最後のサーバー再起動以降のグローバル値を報告します。

SHOW ENGINE INNODB STATUS には、先読みページが読み取られる速度と、そのようなページがアクセスされずに削除される速度も表示されます。秒当たりの平均は、SHOW ENGINE INNODB STATUS の最後の呼出し以降に収集された統計に基づき、SHOW ENGINE INNODB STATUS 出力の BUFFER POOL AND MEMORY セクションに表示されます。

詳細は、[セクション15.8.3.4「InnoDB バッファープールのプリフェッチ \(先読み\) の構成」](#)を参照してください。一般的な I/O チューニングのアドバイスについては、[セクション8.5.8「InnoDB ディスク I/O の最適化」](#)を参照してください。

- innodb_read_io_threads

コマンド行形式	--innodb-read-io-threads=#
システム変数	innodb_read_io_threads
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	4

最小値	1
最大値	64

InnoDB での読み取り操作で使用される I/O スレッドの数です。書き込みスレッドで対応するものは、`innodb_write_io_threads` です。詳細は、[セクション15.8.5「InnoDB バックグラウンド I/O スレッドの数の構成」](#)を参照してください。一般的な I/O チューニングのアドバイスについては、[セクション8.5.8「InnoDB ディスク I/O の最適化」](#)を参照してください。

注記

Linux システムでは、デフォルトの `innodb_read_io_threads` 設定で複数 (一般には 12 台よりも多く) の MySQL サーバーを実行すると、`innodb_write_io_threads` および Linux の `aio-max-nr` 設定がシステムの制限を超過する可能性があります。理想的には、`aio-max-nr` 設定を増やします。回避策として、いずれかまたは両方の MySQL 変数の設定を減らすことができます。

• `innodb_read_only`

コマンド行形式	<code>--innodb-read-only[={OFF ON}]</code>
システム変数	<code>innodb_read_only</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

InnoDB を読み取り専用モードで起動します。読み取り専用メディア上のデータベースアプリケーションまたはデータセットを配布するために使用されます。複数のインスタンス間で同じデータディレクトリを共有する際に、データウェアハウスで使用することもできます。詳細は、[セクション15.8.2「読み取り専用操作の InnoDB の構成」](#)を参照してください。

以前は、`innodb_read_only` システム変数を有効にすると、InnoDB ストレージエンジンのテーブルの作成および削除のみができなくなりました。MySQL 8.0 の時点では、`innodb_read_only` を有効にすると、すべてのストレージエンジンでこれらの操作が防止されます。ストレージエンジンのテーブルの作成および削除操作では、`mysql` システムデータベース内のデータディクショナリテーブルが変更されますが、これらのテーブルは InnoDB ストレージエンジンを使用するため、`innodb_read_only` が有効になっている場合は変更できません。データディクショナリテーブルの変更を必要とする他のテーブル操作にも、同じ原則が適用されます。例:

- `innodb_read_only` システム変数がある場合、InnoDB を使用するデータディクショナリの統計テーブルを更新できないため、`ANALYZE TABLE` が失敗することがあります。キー分散を更新する `ANALYZE TABLE` 操作では、操作によってテーブル自体が更新された場合でも (MyISAM テーブルの場合など)、障害が発生する可能性があります。更新された分散統計を取得するには、`information_schema_stats_expiry=0` を設定します。
- データディクショナリに格納されているストレージエンジンの指定が更新されるため、`ALTER TABLE tbl_name ENGINE=engine_name` は失敗します。

また、`mysql` システムデータベースの他のテーブルでは、MySQL 8.0 の InnoDB ストレージエンジンが使用されます。これらのテーブルを読み取り専用にする、テーブルを変更する操作が制限されます。例:

- 付与テーブルで InnoDB が使用されているため、`CREATE USER` や `GRANT` などのアカウント管理ステートメントは失敗します。
- `mysql.plugin` システムテーブルで InnoDB が使用されているため、`INSTALL PLUGIN` および `UNINSTALL PLUGIN` プラグイン管理ステートメントは失敗します。
- `mysql.func` システムテーブルで InnoDB が使用されているため、`CREATE FUNCTION` および `DROP FUNCTION` UDF 管理ステートメントは失敗します。

- [innodb_redo_log_archive_dirs](#)

コマンド行形式	<code>--innodb-redo-log-archive-dirs</code>
導入	8.0.17
システム変数	innodb_redo_log_archive_dirs
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	NULL

redo ログアーカイブファイルを作成できるラベル付きディレクトリを定義します。複数のラベル付きディレクトリをセミコロン区切りリストで定義できます。例:

```
innodb_redo_log_archive_dirs='label1:/backups1;label2:/backups2'
```

ラベルには任意の文字列を指定できますが、コロン (:) は使用できません。空のラベルも使用できますが、この場合もコロン (:) が必要です。

パスを指定する必要がある、ディレクトリが存在する必要があります。パスにはコロン (:) を含めることができませんが、セミコロン (;) は使用できません。

- [innodb_redo_log_encrypt](#)

コマンド行形式	<code>--innodb-redo-log-encrypt[={OFF ON}]</code>
システム変数	innodb_redo_log_encrypt
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

[InnoDB data-at-rest encryption feature](#) を使用して暗号化されたテーブルの redo ログデータの暗号化を制御します。redo ログデータの暗号化は、デフォルトで無効になっています。詳細は、[redo ログの暗号化](#)を参照してください。

- [innodb_replication_delay](#)

コマンド行形式	<code>--innodb-replication-delay=#</code>
システム変数	innodb_replication_delay
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	4294967295

[innodb_thread_concurrency](#) に到達した場合のレプリカサーバーのレプリケーションスレッド遅延 (ミリ秒)。

- innodb_rollback_on_timeout

コマンド行形式	<code>--innodb-rollback-on-timeout[={OFF ON}]</code>
システム変数	<code>innodb_rollback_on_timeout</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

InnoDB rolls back では、トランザクションタイムアウトの最後のステートメントのみがデフォルトで実行されません。 `--innodb-rollback-on-timeout` が指定されている場合、トランザクションタイムアウトにより、InnoDB はトランザクション全体を中断およびロールバックします。

詳細は、[セクション15.21.4「InnoDBのエラー処理」](#)を参照してください。

- innodb_rollback_segments

コマンド行形式	<code>--innodb-rollback-segments=#</code>
システム変数	<code>innodb_rollback_segments</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	128
最小値	1
最大値	128

`innodb_rollback_segments` では、undo レコードを生成するトランザクションに対して、各 undo テーブルスペースおよびグローバル一時テーブルスペースに割り当てられる `rollback segments` の数を定義します。各ロールバックセグメントでサポートされるトランザクションの数は、InnoDB のページサイズおよび各トランザクションに割り当てられた undo ログの数によって異なります。詳細は、[セクション15.6.6「undo ログ」](#)を参照してください。

関連情報については、[セクション15.3「InnoDB マルチバージョン」](#)を参照してください。undo テーブルスペースの詳細は、[セクション15.6.3.4「undo テーブルスペース」](#)を参照してください。

- innodb_saved_page_number_debug

コマンド行形式	<code>--innodb-saved-page-number-debug=#</code>
システム変数	<code>innodb_saved_page_number_debug</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最大値	$2^{*}23-1$

ページ番号を保存します。 `innodb_fil_make_page_dirty_debug` オプションを設定すると、`innodb_saved_page_number_debug` で定義されたページがダーティになります。`innodb_saved_page_number_debug` オプションは、デバッグサポートが `WITH_DEBUG_CMake` オプションを使用してコンパイルされている場合にのみ使用できます。

- innodb_sort_buffer_size

コマンド行形式	<code>--innodb-sort-buffer-size=#</code>
システム変数	<code>innodb_sort_buffer_size</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1048576
最小値	65536
最大値	67108864

InnoDB インデックスの作成時にデータのソートに使用されるソートバッファのサイズを指定します。指定されたサイズは、内部ソートのためにメモリーに読み取られ、ディスクに書き込まれるデータの量を定義します。このプロセスは「run」と呼ばれます。マージフェーズでは、指定したサイズのバッファのペアが読み取られ、マージされます。設定が大きいほど、実行数が少なくなり、マージされます。

このソート領域は、後続のインデックスのメンテナンス操作時ではなく、インデックスの作成時のマージソートでのみ使用されます。インデックスの作成が完了すると、バッファの割り当てが解除されます。

このオプションの値は、[online DDL](#) 操作中に同時 DML を記録するために一時ログファイルを拡張する量も制御します。

この設定を構成可能にする前は、サイズは 1048576 バイト (1MB) にハードコードされていましたが、これはデフォルトのままです。

インデックスを作成する [ALTER TABLE](#) または [CREATE TABLE](#) ステートメントの実行時に、それぞれが、このオプションで定義されたサイズを持つ 3 つのバッファが割り当てられます。さらに、ポインタ上でソートを実行できるように、ソートバッファ内の行に補助ポインタが割り当てられます (これは、ソート操作時の行の移動とは異なります)。

一般的なソート操作では、次のような式を使用してメモリー消費量を見積もることができます:

```
(6 /*FTS_NUM_AUX_INDEX*/ * (3*@@GLOBAL.innodb_sort_buffer_size)
+ 2 * number_of_partitions * number_of_secondary_indexes_created
* (@@GLOBAL.innodb_sort_buffer_size/dict_index_get_min_size(index)*/)
* 8 /*64-bit sizeof *buf->tuples*/)
```

`@@GLOBAL.innodb_sort_buffer_size/dict_index_get_min_size(index)` は、保持される最大タプル数を示します。 `2 * (@@GLOBAL.innodb_sort_buffer_size/*dict_index_get_min_size(index)*/) * 8 /*64-bit size of *buf->tuples*/` は、割り当てられた補助ポインタを示します。

注記

32 ビットの場合は、8 の代わりに 4 で乗算します。

全文インデックスでの並列ソートでは、`innodb_ft_sort_pll_degree` の設定で乗算します。

```
(6 /*FTS_NUM_AUX_INDEX*/ * @@GLOBAL.innodb_ft_sort_pll_degree)
```

- innodb_spin_wait_delay

コマンド行形式	<code>--innodb-spin-wait-delay=#</code>
システム変数	<code>innodb_spin_wait_delay</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ

型	Integer
デフォルト値	6
最小値	0
最大値 (64 ビットプラットフォーム, ≤ 8.0.13)	2**64-1
最大値 (32 ビットプラットフォーム, ≤ 8.0.13)	2**32-1
最大値 (≥ 8.0.14)	1000

スピンロックでのポーリング間の最大遅延です。このメカニズムの低レベルの実装は、ハードウェアとオペレーティングシステムの組み合わせによって異なるため、遅延は一定の時間間隔に対応しません。

スピンロックポーリング遅延の期間をより詳細に制御するために、`innodb_spin_wait_pause_multiplier` 変数と組み合わせて使用できます。

詳細は、[セクション15.8.8「スピンロックのポーリングの構成」](#)を参照してください。

- [innodb_spin_wait_pause_multiplier](#)

コマンド行形式	<code>--innodb-spin-wait-pause-multiplier=#</code>
導入	8.0.16
システム変数	innodb_spin_wait_pause_multiplier
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	50
最小値	1
最大値	100

スレッドが mutex または rw-lock の取得を待機するときに発生するスピン待機ループ内の PAUSE 命令の数を決定するために使用される乗数値を定義します。

詳細は、[セクション15.8.8「スピンロックのポーリングの構成」](#)を参照してください。

- [innodb_stats_auto_recalc](#)

コマンド行形式	<code>--innodb-stats-auto-recalc[={OFF ON}]</code>
システム変数	innodb_stats_auto_recalc
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

テーブル内のデータが大幅に変更されたあとは、InnoDB によって自動的に永続的統計が再計算されます。しきい値は、テーブルの行の 10% です。この設定は、`innodb_stats_persistent` オプションが有効な場合に作成されるテーブルに適用されます。自動統計再計算は、`CREATE TABLE` ステートメントまたは `ALTER TABLE` ステートメントで `STATS_PERSISTENT=1` を指定して構成することもできます。統計を生成するためにサンプリングされるデータの量は、`innodb_stats_persistent_sample_pages` 変数によって制御されます。

詳細は、[セクション15.8.10.1「永続的オプティマイザ統計のパラメータの構成」](#)を参照してください。

- innodb_stats_include_delete_marked

コマンド行形式	--innodb-stats-include-delete-marked[={OFF ON}]
システム変数	innodb_stats_include_delete_marked
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

デフォルトでは、InnoDB は統計の計算時にコミットされていないデータを読み取ります。テーブルから行を削除するコミットされていないトランザクションの場合、InnoDB では、行の見積りおよびインデックス統計の計算時に削除マークが付けられたレコードが除外されるため、[READ UNCOMMITTED](#) 以外のトランザクション分離レベルを使用してテーブルで同時に操作している他のトランザクションの実行計画が最適でなくなる可能性があります。このシナリオを回避するために、[innodb_stats_include_delete_marked](#) を有効にして、永続オプティマイザ統計の計算時に InnoDB に削除マーク付きレコードが含まれるようになります。

[innodb_stats_include_delete_marked](#) が有効な場合、[ANALYZE TABLE](#) では、統計の再計算時に削除マークが付けられたレコードが考慮されます。

[innodb_stats_include_delete_marked](#) は、すべての InnoDB テーブルに影響するグローバル設定です。永続オプティマイザ統計にのみ適用されます。

関連情報については、[セクション15.8.10.1「永続的オプティマイザ統計のパラメータの構成」](#)を参照してください。

- innodb_stats_method

コマンド行形式	--innodb-stats-method=value
システム変数	innodb_stats_method
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	nulls_equal
有効な値	nulls_equal nulls_unequal nulls_ignored

InnoDB テーブルのインデックス値の分布に関する統計を収集するときに、サーバーが NULL 値を処理する方法です。許可される値は、[nulls_equal](#)、[nulls_unequal](#) および [nulls_ignored](#) です。[nulls_equal](#) の場合、すべての NULL インデックス値は等しいとみなされ、NULL 値の数と等しいサイズの単一の値グループを形成します。[nulls_unequal](#) の場合、NULL 値同士を同等として扱わず、それぞれの NULL はサイズが 1 の別個のグループを生成します。[nulls_ignored](#) の場合、NULL 値は無視されます。

テーブル統計の生成に使用される方法は、[セクション8.3.8「InnoDB および MyISAM インデックス統計コレクション」](#)で説明されているように、オプティマイザがクエリーを実行するためにインデックスを選択する方法に影響します。

- innodb_stats_on_metadata

コマンド行形式	--innodb-stats-on-metadata[={OFF ON}]
システム変数	innodb_stats_on_metadata

スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

このオプションは、オプティマイザ `statistics` が非永続として構成されている場合にのみ適用されます。オプティマイザ統計は、`innodb_stats_persistent` が無効な場合、または `STATS_PERSISTENT=0` を使用して個々のテーブルが作成または変更された場合、ディスクに永続化されません。詳細は、[セクション15.8.10.2「非永続的オプティマイザ統計のパラメータの構成」](#)を参照してください。

`innodb_stats_on_metadata` が有効になっている場合、`SHOW TABLE STATUS` などのメタデータステートメントの場合、または `INFORMATION_SCHEMA.TABLES` テーブルまたは `INFORMATION_SCHEMA.STATISTICS` テーブルにアクセスする場合、InnoDB は非永続 `statistics` を更新します。(これらの更新は、`ANALYZE TABLE` で実行されるものに似ています。)無効にすると、これらの操作時に InnoDB によって統計が更新されません。この設定を無効のままにすると、多数のテーブルまたはインデックスを持つスキーマのアクセス速度を向上させることができます。InnoDB テーブルが関与するクエリーの**実行計画**の安定性も改善できます。

設定を変更するには、`SET GLOBAL innodb_stats_on_metadata=mode` ステートメントを発行します。ここで、`mode` は `ON` と `OFF` のいずれか (または `1` と `0` のいずれか) です。設定を変更するには、グローバルシステム変数を設定するのに十分な権限 ([セクション5.1.9.1「システム変数権限」](#)を参照) が必要で、すべての接続の操作にすぐに影響します。

- `innodb_stats_persistent`

コマンド行形式	<code>--innodb-stats-persistent[={OFF ON}]</code>
システム変数	<code>innodb_stats_persistent</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

InnoDB インデックス統計をディスクに永続化するかどうかを指定します。それ以外の場合は、頻繁に統計が再計算される可能性があります。これにより、**クエリーの実行計画**が変化する可能性があります。テーブルが作成されると、この設定が各テーブルとともに格納されます。テーブルを作成する前にグローバルレベルで `innodb_stats_persistent` を設定することも、`CREATE TABLE` および `ALTER TABLE` ステートメントで `STATS_PERSISTENT` 句を使用して、システム全体の設定をオーバーライドし、個々のテーブルの永続的統計を構成することもできます。

詳細は、[セクション15.8.10.1「永続的オプティマイザ統計のパラメータの構成」](#)を参照してください。

- `innodb_stats_persistent_sample_pages`

コマンド行形式	<code>--innodb-stats-persistent-sample-pages=#</code>
システム変数	<code>innodb_stats_persistent_sample_pages</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	20

インデックス付きカラムの**カーディナリティー**やその他の**統計** (`ANALYZE TABLE` で計算された統計など) を見積もるときに、サンプルとして取得されるインデックス**ページ**の数です。値を大きくすると、**クエリーの実行計画**を改

善するインデックス統計の精度が改善されますが、InnoDB テーブルに対する `ANALYZE TABLE` の実行時に I/O が増加することになります。詳細は、[セクション15.8.10.1「永続的オプティマイザ統計のパラメータの構成」](#)を参照してください。

注記

`innodb_stats_persistent_sample_pages` に大きな値を設定すると、`ANALYZE TABLE` の実行時間が長くなる可能性があります。`ANALYZE TABLE` によってアクセスされるデータベースページの数を見積もるには、[セクション15.8.10.3「InnoDB テーブルに対する ANALYZE TABLE の複雑さの推定」](#)を参照してください。

`innodb_stats_persistent_sample_pages` は、テーブルに対して `innodb_stats_persistent` が有効になっている場合のみ適用され、`innodb_stats_persistent` が無効になっている場合は、かわりに `innodb_stats_transient_sample_pages` が適用されます。

• `innodb_stats_transient_sample_pages`

コマンド行形式	<code>--innodb-stats-transient-sample-pages=#</code>
システム変数	<code>innodb_stats_transient_sample_pages</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	8

インデックス付きカラムのカーディナリティーやその他の統計 (`ANALYZE TABLE` で計算された統計など) を見積もるときに、サンプルとして取得されるインデックスページの数です。デフォルト値は 8 です。値を大きくすると、インデックス統計の精度が改善されます。これにより、クエリーの実行計画を改善できますが、InnoDB テーブルを開くときや統計を再計算するときに I/O が増加するという犠牲が伴います。詳細は、[セクション15.8.10.2「非永続的オプティマイザ統計のパラメータの構成」](#)を参照してください。

注記

`innodb_stats_transient_sample_pages` に大きな値を設定すると、`ANALYZE TABLE` の実行時間が長くなる可能性があります。`ANALYZE TABLE` によってアクセスされるデータベースページの数を見積もるには、[セクション15.8.10.3「InnoDB テーブルに対する ANALYZE TABLE の複雑さの推定」](#)を参照してください。

`innodb_stats_transient_sample_pages` は、テーブルに対して `innodb_stats_persistent` が無効になっている場合のみ適用され、`innodb_stats_persistent` が有効になっている場合は、かわりに `innodb_stats_persistent_sample_pages` が適用されます。`innodb_stats_sample_pages` のかわりに使用します。詳細は、[セクション15.8.10.2「非永続的オプティマイザ統計のパラメータの構成」](#)を参照してください。

• `innodb_status_output`

コマンド行形式	<code>--innodb-status-output[={OFF ON}]</code>
システム変数	<code>innodb_status_output</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

標準 InnoDB モニターの定期出力を有効または無効にします。また、InnoDB Lock Monitor の定期的な出力を有効または無効にする際に、`innodb_status_output_locks` と組み合わせて使用されます。詳細は、[セクション15.17.2「InnoDB モニターの有効化」](#)を参照してください。

- [innodb_status_output_locks](#)

コマンド行形式	<code>--innodb-status-output-locks[={OFF ON}]</code>
システム変数	innodb_status_output_locks
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

InnoDB ロックモニターを有効または無効にします。有効にすると、InnoDB ロックモニターは、[SHOW ENGINE INNODB STATUS](#) 出力および MySQL エラーログに出力される定期的な出力にロックに関する追加情報を出力します。InnoDB ロックモニターの定期的な出力は、標準の InnoDB モニター出力の一部として出力されます。したがって、InnoDB ロックモニターで MySQL エラーログに定期的にデータを出力するには、標準の InnoDB モニターを有効にする必要があります。詳細は、[セクション 15.17.2 「InnoDB モニターの有効化」](#) を参照してください。

- [innodb_strict_mode](#)

コマンド行形式	<code>--innodb-strict-mode[={OFF ON}]</code>
システム変数	innodb_strict_mode
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

[innodb_strict_mode](#) が有効になっている場合、InnoDB は特定の条件に対して警告ではなくエラーを返します。

厳密モードは、SQL 内の無視できる誤字や構文エラー、または操作モードと SQL ステートメントのさまざまな組み合わせによる意図しないその他の結果から保護する際に役立ちます。[innodb_strict_mode](#) が有効になっている場合、InnoDB では、警告を発行して指定されたステートメントを処理するのではなく (おそらく意図しない動作で)、特定のケースでエラー状態が発生します。これは、MySQL で受け入れられる SQL 構文を制御し、警告なしでエラーを無視するのか、入力構文とデータ値を検証するのかを決定する MySQL の [sql_mode](#) と類似しています。

[innodb_strict_mode](#) 設定は、[CREATE TABLE](#)、[ALTER TABLE](#)、[CREATE INDEX](#) および [OPTIMIZE TABLE](#) ステートメントの構文エラーの処理に影響します。[innodb_strict_mode](#) ではレコードサイズチェックも有効になるため、[INSERT](#) または [UPDATE](#) は、選択したページサイズに対してレコードが大きすぎるために失敗することはありません。

[CREATE TABLE](#) ステートメント、[ALTER TABLE](#) ステートメントおよび [CREATE INDEX](#) ステートメントで [ROW_FORMAT](#) 句および [KEY_BLOCK_SIZE](#) 句を使用する場合は、Oracle で [innodb_strict_mode](#) を有効にすることをお勧めします。[innodb_strict_mode](#) が無効になっている場合、InnoDB は競合する句を無視し、メッセージログに警告のみを表示してテーブルまたはインデックスを作成します。結果のテーブルには、圧縮テーブルを作成しようとしたときの圧縮サポートの不足など、意図したものとは異なる特性があります。[innodb_strict_mode](#) が有効な場合、このような問題により即時エラーが生成され、テーブルまたはインデックスは作成されません。

[innodb_strict_mode](#) は、[mysqld](#) の起動時にコマンドラインで、または MySQL [configuration file](#) で有効または無効にできます。[SET \[GLOBAL|SESSION\] innodb_strict_mode=mode](#) ステートメントを使用して、実行時に [innodb_strict_mode](#) を有効または無効にすることもできます。ここで、[mode](#) は ON または OFF です。[GLOBAL](#) 設定を変更するには、グローバルシステム変数を設定するのに十分な権限 ([セクション 5.1.9.1 「システム変数権](#)

限」を参照) が必要であり、その後接続するすべてのクライアントの操作に影響します。任意のクライアントが `innodb_strict_mode` の `SESSION` 設定を変更でき、そのクライアントのみが設定の影響を受けます。

`innodb_strict_mode` は、`general tablespaces` には適用できません。一般的なテーブルスペースのテーブルスペース管理ルールは、`innodb_strict_mode` とは無関係に厳密に適用されます。詳細は、[セクション13.1.21「CREATE TABLESPACE ステートメント」](#)を参照してください。

- `innodb_sync_array_size`

コマンド行形式	<code>--innodb-sync-array-size=#</code>
システム変数	<code>innodb_sync_array_size</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1
最小値	1
最大値	1024

`mutex/lock` 待機配列のサイズを定義します。値を大きくすると、スレッドの調整に使用される内部データ構造が分割され、多数の待機スレッドを持つワークロードの同時実行性が向上します。この設定は MySQL インスタンスの起動時に構成する必要があり、あとで変更することはできません。頻繁に多数の待機スレッドを生成するワークロード (通常は 768 を超える) では、値を増やすことをお勧めします。

- `innodb_sync_spin_loops`

コマンド行形式	<code>--innodb-sync-spin-loops=#</code>
システム変数	<code>innodb_sync_spin_loops</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	30
最小値	0
最大値	4294967295

スレッドが中断される前に、InnoDB 相互排他ロックが開放されるまでスレッドが待機する回数です。

- `innodb_sync_debug`

コマンド行形式	<code>--innodb-sync-debug[={OFF ON}]</code>
システム変数	<code>innodb_sync_debug</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

InnoDB ストレージエンジンの同期デバッグ検査を有効にします。このオプションは、デバッグサポートが `WITH_DEBUG CMake` オプションを使用してコンパイルされている場合のみ使用できます。

- `innodb_table_locks`

コマンド行形式	<code>--innodb-table-locks[={OFF ON}]</code>
システム変数	<code>innodb_table_locks</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

`autocommit = 0` の場合、InnoDB は LOCK TABLES の要求を受け入れます。MySQL はすべてのスレッドがテーブルに対するすべてのロックを解放するまで、LOCK TABLES ... WRITE から戻りません。innodb_table_locks のデフォルト値は 1 です。これは、autocommit = 0. の場合、LOCK TABLES によって InnoDB がテーブルを内部的にロックすることを意味します。

innodb_table_locks = 0 は、LOCK TABLES ... WRITE で明示的にロックされたテーブルには影響しません。LOCK TABLES ... WRITE で暗黙的に (たとえば、トリガーを使用して)、または LOCK TABLES ... READ によって、読み取りまたは書き込み用にロックされたテーブルには有効です。

関連情報については、セクション 15.7 「InnoDB のロックおよびトランザクションモデル」を参照してください。

- innodb_temp_data_file_path

コマンド行形式	<code>--innodb-temp-data-file-path=file_name</code>
システム変数	<code>innodb_temp_data_file_path</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	<code>ibtmp1:12M:autoextend</code>

グローバル一時テーブルスペースデータファイルの相対パス、名前、サイズおよび属性を定義します。グローバル一時テーブルスペースには、ユーザー作成一時テーブルに対する変更のロールバックセグメントが格納されます。

innodb_temp_data_file_path に値が指定されていない場合、デフォルトの動作では、ibtmp1 という名前の単一の自動拡張データファイルが innodb_data_home_dir ディレクトリに作成されます。初期ファイルサイズは 12MB を少し超えています。

グローバル一時テーブルスペースのデータファイル指定の構文には、ファイル名、ファイルサイズ、autoextend および max 属性が含まれます:

```
file_name:file_size[:autoextend[:max:max_file_size]]
```

グローバル一時テーブルスペースデータファイルには、別の InnoDB データファイルと同じ名前を付けることはできません。グローバル一時テーブルスペースデータファイルを作成できない場合やエラーが発生した場合は、致命的として扱われ、サーバーの起動は拒否されます。

ファイルサイズは、K、M または G をサイズ値に追加することで、KB、MB または GB で指定します。ファイルサイズの合計は、12MB より少し大きくする必要があります。

個々のファイルのサイズ制限は、オペレーティングシステムによって決まります。大規模ファイルをサポートするオペレーティングシステムでは、ファイルサイズが 4GB を超える場合があります。グローバル一時テーブルスペースデータファイルに対する RAW ディスクパーティションの使用はサポートされていません。

autoextend および max 属性は、innodb_temp_data_file_path 設定で最後に指定されたデータファイルにのみ使用できます。例:

```
[mysqld]
```

```
innodb_temp_data_file_path=ibtmp1:50M;ibtmp2:12M:autoextend:max:500MB
```

`autoextend` オプションを使用すると、データファイルの空き領域がなくなると、データファイルのサイズが自動的に増加します。デフォルトでは、`autoextend` の増分は 64MB です。増分を変更するには、`innodb_autoextend_increment` 変数の設定を変更します。

グローバル一時テーブルスペースデータファイルのディレクトリパスは、`innodb_data_home_dir` および `innodb_temp_data_file_path` で定義されたパスを連結することによって形成されます。

InnoDB を読み取り専用モードで実行する前に、`innodb_temp_data_file_path` をデータディレクトリ外の場所に設定します。パスは、データディレクトリに対する相対パスである必要があります。例:

```
--innodb-temp-data-file-path=../../tmp/ibtmp1:12M:autoextend
```

詳細は、[グローバル一時テーブルスペース](#)を参照してください。

- `innodb_temp_tablespaces_dir`

コマンド行形式	<code>--innodb-temp-tablespaces-dir=dir_name</code>
導入	8.0.13
システム変数	<code>innodb_temp_tablespaces_dir</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ディレクトリ名
デフォルト値	<code>#innodb_temp</code>

起動時に InnoDB がセッション一時テーブルスペースのプールを作成する場所を定義します。デフォルトの場所は、データディレクトリ内の `#innodb_temp` ディレクトリです。データディレクトリに対する完全修飾パスまたは相対パスが許可されます。

MySQL 8.0.16 では、セッション一時テーブルスペースには常に、InnoDB を使用して最適化によって作成されたユーザー作成一時テーブルおよび内部一時テーブルが格納されます。(以前は、内部一時テーブルのディスク上のストレージエンジンは、サポートされなくなった `internal_tmp_disk_storage_engine` システム変数によって決定されていました。[オンディスク内部一時テーブルのストレージエンジン](#)を参照してください。)

詳細は、[セッション一時テーブルスペース](#)を参照してください。

- `innodb_thread_concurrency`

コマンド行形式	<code>--innodb-thread-concurrency=#</code>
システム変数	<code>innodb_thread_concurrency</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0

最大値	1000
-----	------

InnoDB 内で許可されるスレッドの最大数を定義します。値 0 (デフォルト) は、無限同時実行性 (制限なし) として解釈されます。この変数は、高同時実行性システムでのパフォーマンスチューニングを目的としています。

InnoDB は、InnoDB 内のスレッド数を `innodb_thread_concurrency` の制限以下に保つことを試みます。制限に達すると、待機スレッドの「先入れ先出し」(FIFO) キューに追加のスレッドが配置されます。ロックを待機しているスレッドは、並列実行中のスレッドの数にカウントされません。

正しい設定は、ワークロードおよびコンピューティング環境によって異なります。MySQL インスタンスが CPU リソースを他のアプリケーションと共有している場合、またはワークロードや同時ユーザー数が増加している場合は、この変数の設定を検討してください。値の範囲をテストして、最適なパフォーマンスを提供する設定を決定します。`innodb_thread_concurrency` は動的変数で、ライブテストシステムで様々な設定を試すことができます。特定の設定でパフォーマンスが低下した場合は、すぐに `innodb_thread_concurrency` を 0 に戻してください。

次のガイドラインに従うと、適切な設定を見つけて保持する際に役立ちます。

- ワークロードの同時ユーザースレッドの数が一貫して小さく、パフォーマンスに影響しない場合は、`innodb_thread_concurrency=0` を設定します (制限なし)。
- ワークロードが一貫して大きく、または時々スパイクする場合は、`innodb_thread_concurrency` 値を設定し、最適なパフォーマンスを提供するスレッドの数が見つかるまで調整します。たとえば、システムに通常 40 から 50 人のユーザーがいるが、定期的に 60、70 以上に増加するとします。テストにより、同時ユーザー数は 80 に制限され、パフォーマンスはほとんど安定したままであることがわかります。この場合、`innodb_thread_concurrency` を 80 に設定します。
- InnoDB でユーザースレッドに特定の数を超える仮想 CPU (たとえば、20 個の仮想 CPU) を使用しない場合は、`innodb_thread_concurrency` をこの数に設定します (パフォーマンステストによっては小さくなる可能性があります)。MySQL を他のアプリケーションから分離することを目的としている場合は、`mysqld` プロセスを仮想 CPU のみにバインドすることを確認してください。ただし、排他的バインドを使用すると、`mysqld` プロセスが一貫してビジー状態でない場合に最適でないハードウェア使用量になる可能性があることに注意してください。この場合、`mysqld` プロセスを仮想 CPU にバインドできますが、他のアプリケーションが一部またはすべての仮想 CPU を使用できるようになります。

注記

オペレーティングシステムの観点からは、リソース管理ソリューションを使用して、`mysqld` プロセスをバインドするよりもアプリケーション間で CPU 時間がどのように共有されるかを管理することをお勧めします。たとえば、他のクリティカルプロセスが実行されていないときに特定のアプリケーションに 90% の仮想 CPU 時間を割り当て、他のクリティカルプロセスが実行されているときにその値を 40% にスケールバックできます。

- 場合によっては、最適な `innodb_thread_concurrency` 設定が仮想 CPU の数より小さいことがあります。
- `innodb_thread_concurrency` 値が高すぎると、システム内部およびリソースの競合が増加するため、パフォーマンスが低下する可能性があります。
- 定期的にシステムをモニターし、分析してください。ワークロード、ユーザー数、またはコンピューティング環境を変更するために、`innodb_thread_concurrency` 設定の調整が必要なことがあります。

値 0 を指定すると、`SHOW ENGINE INNODB STATUS` 出力の `ROW OPERATIONS` セクションの `queries inside InnoDB` および `queries in queue` カウンタが無効になります。

関連情報については、[セクション 15.8.4 「InnoDB のスレッド並列性の構成」](#) を参照してください。

- `innodb_thread_sleep_delay`

コマンド行形式	<code>--innodb-thread-sleep-delay=#</code>
システム変数	<code>innodb_thread_sleep_delay</code>
スコープ	グローバル

動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	10000
最小値	0
最大値	1000000

InnoDB キューに参加するまでに、InnoDB スレッドがスリープ状態になる期間 (マイクロ秒単位) です。デフォルト値は 10000 です。0 の値はスリープを無効にします。innodb_adaptive_max_sleep_delay を innodb_thread_sleep_delay に許可する最大値に設定すると、InnoDB は現在のスレッドスケジューリングアクティビティに応じて innodb_thread_sleep_delay を自動的に上下に調整します。この動的調整は、システムが軽くロードされているとき、またはほぼ全容量で動作しているときに、スレッドスケジューリングメカニズムが円滑に機能するのに役立ちます。

詳細は、[セクション15.8.4「InnoDB のスレッド並列性の構成」](#)を参照してください。

- innodb_tmpdir

コマンド行形式	--innodb-tmpdir=dir_name
システム変数	innodb_tmpdir
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	ディレクトリ名
デフォルト値	NULL

テーブルを再構築するオンライン ALTER TABLE 操作中に作成される一時ソートファイルの代替ディレクトリを定義するために使用します。

テーブルを再構築するオンライン ALTER TABLE 操作では、元のテーブルと同じディレクトリに中間テーブルファイルも作成されます。innodb_tmpdir オプションは、中間テーブルファイルには適用されません。

有効な値は、MySQL データディレクトリパス以外の任意のディレクトリパスです。値が NULL の場合 (デフォルト)、一時ファイルは MySQL 一時ディレクトリ (Unix の場合は \$TMPDIR、Windows の場合は %TEMP%、--tmpdir 構成オプションで指定されたディレクトリ)、作成されます。ディレクトリが指定されている場合、SET ステートメントを使用して innodb_tmpdir が構成されている場合にのみ、ディレクトリの存在と権限がチェックされます。symlink がディレクトリ文字列に指定されている場合、symlink は解決され、絶対パスとして格納されます。パスは 512 バイトを超えることはできません。innodb_tmpdir が無効なディレクトリに設定されている場合、オンラインの ALTER TABLE 操作でエラーが報告されます。innodb_tmpdir は、MySQL tmpdir 設定をオーバーライドしますが、オンラインの ALTER TABLE 操作の場合のみです。

innodb_tmpdir を構成するには、FILE 権限が必要です。

tmpfs ファイルシステムにある一時ファイルディレクトリのオーバーフローを回避するために、innodb_tmpdir オプションが導入されました。このようなオーバーフローは、テーブルを再構築するオンライン ALTER TABLE 操作中に作成された大規模な一時ソートファイルの結果として発生する可能性があります。

レプリケーション環境では、すべてのサーバーに同じオペレーティングシステム環境がある場合のみ、innodb_tmpdir 設定のレプリケートを検討してください。それ以外の場合、innodb_tmpdir 設定をレプリケートすると、テーブルを再構築するオンライン ALTER TABLE 操作の実行時にレプリケーションが失敗する可能性があります。サーバーの動作環境が異なる場合は、各サーバーで個別に innodb_tmpdir を構成することをお勧めします。

詳細は、[セクション15.12.3「オンライン DDL 領域の要件」](#)を参照してください。ALTER TABLE のオンライン操作の詳細は、[セクション15.12「InnoDB とオンライン DDL」](#)を参照してください。

- innodb_trx_purge_view_update_only_debug

コマンド行形式	--innodb-trx-purge-view-update-only-debug[={OFF ON}]
システム変数	innodb_trx_purge_view_update_only_debug
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

ページビューの更新を許可しながら、削除マーク付きレコードのページを一時停止します。このオプションでは、ページビューは更新されますが、ページはまだ実行されていない状況が人為的に作成されます。このオプションは、デバッグサポートが [WITH_DEBUG CMake](#) オプションを使用してコンパイルされている場合にのみ使用できません。

- innodb_trx_rseg_n_slots_debug

コマンド行形式	--innodb-trx-rseg-n-slots-debug=#
システム変数	innodb_trx_rseg_n_slots_debug
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最大値	1024

[TRX_RSEG_N_SLOTS](#) を、undo ログセグメントの空きスロットを検索する [trx_rsegf_undo_find_free](#) 関数の特定の値に制限するデバッグフラグを設定します。このオプションは、デバッグサポートが [WITH_DEBUG CMake](#) オプションを使用してコンパイルされている場合にのみ使用できます。

- innodb_undo_directory

コマンド行形式	--innodb-undo-directory=dir_name
システム変数	innodb_undo_directory
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ディレクトリ名

InnoDB が undo テーブルスペースを作成するパス。通常、undo テーブルスペースを別のストレージデバイスに配置するために使用されます。

デフォルト値はありません (NULL)。[innodb_undo_directory](#) 変数が定義されていない場合、undo テーブルスペースはデータディレクトリに作成されます。

MySQL インスタンスの初期化時に作成されるデフォルトの undo テーブルスペース ([innodb_undo_001](#) および [innodb_undo_002](#)) は、[innodb_undo_directory](#) 変数で定義されたディレクトリに常に存在します。

別のパスが指定されていない場合、[CREATE UNDO TABLESPACE](#) 構文を使用して作成された undo テーブルスペースは、[innodb_undo_directory](#) 変数で定義されたディレクトリに作成されます。

詳細は、[セクション15.6.3.4「undo テーブルスペース」](#)を参照してください。

- innodb_undo_log_encrypt

コマンド行形式	<code>--innodb-undo-log-encrypt[={OFF ON}]</code>
システム変数	innodb_undo_log_encrypt
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

InnoDB [data-at-rest encryption feature](#) を使用して暗号化されたテーブルの undo ログデータの暗号化を制御します。個別の [undo tablespaces](#) に存在する undo ログにのみ適用されます。 [セクション15.6.3.4「undo テーブルスペース」](#) を参照してください。システムテーブルスペースに存在する undo ログデータの暗号化はサポートされていません。詳細は、[undo ログの暗号化](#) を参照してください。

- innodb_undo_log_truncate

コマンド行形式	<code>--innodb-undo-log-truncate[={OFF ON}]</code>
システム変数	innodb_undo_log_truncate
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

有効にすると、[innodb_max_undo_log_size](#) で定義されたしきい値を超える undo テーブルスペースに切捨てるマークが付けられます。undo テーブルスペースのみ切り捨てられます。システムテーブルスペースに存在する undo ログの切捨てはサポートされていません。切捨てるを実行するには、少なくとも2つの undo テーブルスペースが必要です。

[innodb_purge_rseg_truncate_frequency](#) 変数を使用すると、undo テーブルスペースの切捨てるを迅速に実行できます。

詳細は、[undo テーブルスペースの切捨てる](#) を参照してください。

- innodb_undo_tablespaces

コマンド行形式	<code>--innodb-undo-tablespaces=#</code>
非推奨	はい
システム変数	innodb_undo_tablespaces
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	2
最小値	2

最大値	127
-----	-----

InnoDB で使用される `undo tablespaces` の数を定義します。デフォルト値と最小値は 2 です。

注記

`innodb_undo_tablespaces` 変数は非推奨であり、MySQL 8.0.14 の時点では構成できなくなりました。将来のリリースで削除される予定です。

詳細は、[セクション15.6.3.4「undo テーブルスペース」](#)を参照してください。

- [innodb_use_native_aio](#)

コマンド行形式	<code>--innodb-use-native-aio[={OFF ON}]</code>
システム変数	innodb_use_native_aio
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

Linux の非同期 I/O サブシステムを使用するかどうかを指定します。この変数は Linux システムにのみ適用され、サーバーの実行中は変更できません。通常、このオプションはデフォルトで有効になっているため、構成する必要はありません。

Windows システムで InnoDB が持つ [asynchronous I/O](#) 機能は、Linux システムで使用できます。(その他の Unix に似たシステムでは、引き続き同期 I/O 呼び出しが使用されます。) この機能により、[SHOW ENGINE INNODB STATUS](#) の出力に多くの保留中の読取り/書き込みが通常表示される、大量の I/O-bound システムのスケラビリティが向上します。

大量の InnoDB I/O スレッドとともに実行すると (特に、同じサーバーマシン上で複数のこのようなインスタンスを実行すると)、Linux システムの能力制限を超える可能性があります。この場合、次のエラーを受信する可能性があります。

```
EAGAIN: The specified maxevents exceeds the user's limit of available events.
```

一般に、`/proc/sys/fs/aio-max-nr` により大きな制限を記述すれば、このエラーに対処できます。

ただし、OS の非同期 I/O サブシステムに問題があるために InnoDB を起動できない場合は、`innodb_use_native_aio=0` を使用してサーバーを起動できます。このオプションは、`tmpfs` で AIO をサポートしていない `tmpdir` の場所、`tmpfs` ファイルシステム、Linux カーネルの組合せなどの潜在的な問題が InnoDB によって検出された場合にも、起動時に自動的に無効になることがあります。

詳細は、[セクション15.8.6「Linux での非同期 I/O の使用」](#)を参照してください。

- [innodb_validate_tablespace_paths](#)

コマンド行形式	<code>--innodb-validate-tablespace-paths[={OFF ON}]</code>
導入	8.0.21
システム変数	innodb_validate_tablespace_paths
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean

デフォルト値	ON
--------	----

テーブルスペースファイルパスの検証を制御します。起動時に、InnoDB は、テーブルスペースファイルが別の場所に移動された場合に備えて、データディクショナリに格納されているテーブルスペースファイルパスに対して既知のテーブルスペースファイルのパスを検証します。 `innodb_validate_tablespace_paths` 変数を使用すると、テーブルスペースパスの検証を無効にできます。この機能は、テーブルスペースファイルを移動しない環境を対象としています。パス検証を無効にすると、多数のテーブルスペースファイルがあるシステムでの起動時間が短縮されません。

警告

テーブルスペースファイルの移動後にテーブルスペースパス検証を無効にしてサーバーを起動すると、動作が未定義になる可能性があります。

詳細は、[セクション15.6.3.7「テーブルスペースパス検証の無効化」](#)を参照してください。

• `innodb_version`

InnoDB のバージョン番号です。MySQL 8.0 では、InnoDB の個別のバージョン番号は適用されず、この値はサーバーの `version` 番号と同じです。

• `innodb_write_io_threads`

コマンド行形式	<code>--innodb-write-io-threads=#</code>
システム変数	<code>innodb_write_io_threads</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	4
最小値	1
最大値	64

InnoDB の書き込み操作で使用される I/O スレッドの数です。デフォルト値は 4 です。読み取りスレッドで対応するものは、`innodb_read_io_threads` です。詳細は、[セクション15.8.5「InnoDB バックグラウンド I/O スレッドの数の構成」](#)を参照してください。一般的な I/O チューニングのアドバイスについては、[セクション8.5.8「InnoDB ディスク I/O の最適化」](#)を参照してください。

注記

Linux システムでは、デフォルトの `innodb_read_io_threads` 設定で複数 (一般には 12 台よりも多く) の MySQL サーバーを実行すると、`innodb_write_io_threads` および Linux の `aio-max-nr` 設定がシステムの制限を超過する可能性があります。理想的には、`aio-max-nr` 設定を増やします。回避策として、いずれかまたは両方の MySQL 変数の設定を減らすことができます。

また、バイナリログとディスクの同期を制御する `sync_binlog` の値も考慮してください。

一般的な I/O チューニングのアドバイスについては、[セクション8.5.8「InnoDB ディスク I/O の最適化」](#)を参照してください。

15.15 InnoDB INFORMATION_SCHEMA テーブル

このセクションでは、InnoDB INFORMATION_SCHEMA テーブルについて、その使用例とともに説明します。

InnoDB INFORMATION_SCHEMA テーブルは、InnoDB ストレージエンジンのさまざまな側面に関するメタデータ、ステータス情報、および統計を提供します。INFORMATION_SCHEMA データベースで `SHOW TABLES` ステートメントを発行することによって、InnoDB INFORMATION_SCHEMA テーブルのリストを表示できます。

```
mysql> SHOW TABLES FROM INFORMATION_SCHEMA LIKE 'INNODB%';
```

テーブル定義については、[セクション26.51「INFORMATION_SCHEMA InnoDB テーブル」](#)を参照してください。MySQL INFORMATION_SCHEMA データベースに関連した一般的な情報については、[第26章「INFORMATION_SCHEMA テーブル」](#)を参照してください。

15.15.1 圧縮に関する InnoDB INFORMATION_SCHEMA テーブル

圧縮が全体としてどれだけ適切に機能しているかを把握するための、圧縮に関する InnoDB INFORMATION_SCHEMA テーブルのペアとして、次の2つがあります。

- `INNODB_CMP` および `INNODB_CMP_RESET` は、圧縮操作の数および圧縮の実行に費やされた時間に関する情報を提供します。
- `INNODB_CMPMEM` および `INNODB_CMPMEM_RESET` は、圧縮のためのメモリーの割当て方法に関する情報を提供します。

15.15.1.1 INNODB_CMP および INNODB_CMP_RESET

`INNODB_CMP` テーブルおよび `INNODB_CMP_RESET` テーブルは、[セクション15.9「InnoDB のテーブルおよびページの圧縮」](#)で説明されている圧縮テーブルに関連する操作のステータス情報を提供します。`PAGE_SIZE` カラムは、圧縮されたページサイズをレポートします。

これらの2つのテーブルの内容は同じですが、`INNODB_CMP_RESET` から読み取ると、圧縮および圧縮解除操作に関する統計がリセットされます。たとえば、`INNODB_CMP_RESET` の出力を60分に1回アーカイブした場合は、1時間ごとの統計が表示されます。`INNODB_CMP` の出力を監視する場合 (`INNODB_CMP_RESET` を読み取らないことを確認)、InnoDB の起動以降の累積統計が表示されます。

テーブル定義については、[セクション26.51.5「INFORMATION_SCHEMA INNODB_CMP および INNODB_CMP_RESET テーブル」](#)を参照してください。

15.15.1.2 INNODB_CMPMEM および INNODB_CMPMEM_RESET

`INNODB_CMPMEM` および `INNODB_CMPMEM_RESET` テーブルは、バッファプール内に存在する圧縮されたページに関するステータス情報を提供します。圧縮テーブルおよびバッファプールの使用の詳細は、[セクション15.9「InnoDB のテーブルおよびページの圧縮」](#)を参照してください。`INNODB_CMP` および `INNODB_CMP_RESET` テーブルでは、圧縮に関するより役立つ統計が提供されます。

内部の詳細

InnoDB は、[バディアアロケータシステム](#)を使用して、1Kバイトから16Kバイトまでのさまざまなサイズのページに割り当てられたメモリーを管理します。ここで説明されている2つのテーブルの各行は、1つのページサイズに対応します。

`INNODB_CMPMEM` および `INNODB_CMPMEM_RESET` テーブルの内容は同じですが、`INNODB_CMPMEM_RESET` から読み取ると、再配置操作に関する統計がリセットされます。たとえば、`INNODB_CMPMEM_RESET` の出力を60分に1回アーカイブした場合は、1時間ごとの統計が表示されます。`INNODB_CMPMEM_RESET` を読み取らずに `INNODB_CMPMEM` の出力を監視した場合、InnoDB の起動後の累積統計が表示されます。

テーブル定義については、[セクション26.51.6「INFORMATION_SCHEMA INNODB_CMPMEM および INNODB_CMPMEM_RESET テーブル」](#)を参照してください。

15.15.1.3 圧縮情報スキーマテーブルの使用

例 15.1 圧縮情報スキーマテーブルの使用

圧縮テーブルを含むデータベースからのサンプル出力を次に示します ([セクション15.9「InnoDB のテーブルおよびページの圧縮」](#)、`INNODB_CMP`、`INNODB_CMP_PER_INDEX`、および `INNODB_CMPMEM` を参照してください)。

次の表は、軽いワークロード下にある `INFORMATION_SCHEMA.INNODB_CMP` の内容を示しています。バッファプールに含まれている唯一の圧縮ページサイズは8Kです。カラム `COMPRESS_TIME` および `UNCOMPRESS_TIME` が0であるため、ページの圧縮または圧縮解除で消費された時間は統計がリセットされてから1秒未満でした。

page size	compress ops	compress ops ok	compress time	uncompress ops	uncompress time
1024	0	0	0	0	0
2048	0	0	0	0	0
4096	0	0	0	0	0
8192	1048	921	0	61	0
16384	0	0	0	0	0

INNOODB_CMPMEM によると、バッファプール内には 6169 個の圧縮された 8K バイトページが存在します。割り当てられているほかのブロックサイズは 64 バイトだけです。INNOODB_CMPMEM 内のもっとも小さい PAGE_SIZE は、対応する圧縮解除されたページがバッファプール内に存在しない圧縮ページのブロックディスクリプタとして使用されます。このようなページが 5910 個存在することがわかります。また、間接的には、259 (6169-5910) 個の圧縮ページもバッファプール内に圧縮解除された形式で存在することがわかります。

次の表は、軽いワークロード下にある INFORMATION_SCHEMA.INNOODB_CMPMEM の内容を示しています。圧縮ページのためのメモリアロケータの断片化のために、一部のメモリー $SUM(PAGE_SIZE * PAGES_FREE) = 6784$ は使用できません。これは、小さなメモリー割り当て要求が、バディアロケーションシステムを使用して (メインのバッファプールから割り当てられる 16K ブロックから始めて) より大きなブロックを分割することによって満たされるためです。断片化がこのように少ないのは、より大きな隣接した空きブロックを形成するために、割り当てられた一部のブロックが再配置 (コピー) されたためです。この $SUM(PAGE_SIZE * RELOCATION_OPS)$ バイトのコピーで消費された時間は 1 秒未満でした ($SUM(RELOCATION_TIME) = 0$)。

page size	pages used	pages free	relocation ops	relocation time
64	5910	0	2436	0
128	0	1	0	0
256	0	0	0	0
512	0	1	0	0
1024	0	0	0	0
2048	0	1	0	0
4096	0	1	0	0
8192	6169	0	5	0
16384	0	0	0	0

15.15.2 InnoDB INFORMATION_SCHEMA トランザクションおよびロック情報

注記

このセクションでは、MySQL 8.0 内の INFORMATION_SCHEMA.INNOODB_LOCKS および INNOODB_LOCK_WAITS テーブルよりも優先される、パフォーマンススキーマ data_locks および data_lock_waits テーブルによって公開されるロック情報について説明します。古い INFORMATION_SCHEMA テーブルに関して記述されている同様の説明は、MySQL 5.7 Reference Manual の InnoDB INFORMATION_SCHEMA Transaction and Locking Information を参照してください。

一方の INFORMATION_SCHEMA テーブルと 2 つの「パフォーマンススキーマ」テーブルを使用すると、InnoDB トランザクションを監視し、潜在的なロックの問題を診断できます:

- **INNOODB_TRX**: この INFORMATION_SCHEMA テーブルには、InnoDB 内で現在実行されているすべてのトランザクションに関する情報が表示されます。これには、トランザクションの状態 (実行中かロック待機中かなど)、トランザクションの開始時期、トランザクションが実行されている特定の SQL ステートメントなどが含まれます。
- **data_locks**: 「このパフォーマンススキーマ」テーブルには、各保留ロックの行と、保留ロックの解放を待機してブロックされる各ロックリクエストが含まれています:
 - ロックを保持しているトランザクションの状態 (INNOODB_TRX.TRX_STATE が RUNNING, LOCK WAIT, ROLLING BACK または COMMITTING) にかかわらず、保持されているロックごとに 1 つの行があります。

- 別のトランザクションがロックを解放するのを待機している InnoDB 内の各トランザクション (INNODB_TRX.TRX_STATE は LOCK WAIT) は、単一のブロッキングロックリクエストによってブロックされます。そのブロックしているロック要求は、互換性がないモードにある別のトランザクションによって保持されている行ロックまたはテーブルロックに対するものです。ロック要求には常に、要求をブロックする保持ロックのモード (読み取りと書き込み、共有と排他) と互換性のないモードがあります。

ブロックされたトランザクションは、他のトランザクションがコミットまたはロールバックされ、リクエストされたロックが解放されるまで続行できません。ブロックされたトランザクションごとに、`data_locks` には、トランザクションがリクエストしたロックと待機しているロックを示す行が 1 つ含まれます。

- `data_lock_waits`: 「このパフォーマンススキーマ」テーブルには、特定のロックを待機しているトランザクション、または特定のトランザクションが待機しているロックが示されます。このテーブルには、ブロックされているトランザクションごとに、そのトランザクションが要求したロックと、その要求をブロックしているロックを示す 1 つ以上の行が含まれています。REQUESTING_ENGINE_LOCK_ID 値はトランザクションによってリクエストされたロックを参照し、BLOCKING_ENGINE_LOCK_ID 値は最初のトランザクションの続行を妨げる (別のトランザクションによって保持されている) ロックを参照します。特定のブロックされたトランザクションについて、`data_lock_waits` のすべての行の値は、REQUESTING_ENGINE_LOCK_ID では同じで、BLOCKING_ENGINE_LOCK_ID では異なる値になります。

前述のテーブルの詳細は、[セクション26.51.29「INFORMATION_SCHEMA INNODB_TRX テーブル」](#)、[セクション27.12.13.1「data_locks テーブル」](#) および [セクション27.12.13.2「data_lock_waits テーブル」](#) を参照してください。

15.15.2.1 InnoDB トランザクションの使用および情報のロック

注記

このセクションでは、MySQL 8.0 内の INFORMATION_SCHEMA INNODB_LOCKS および INNODB_LOCK_WAITS テーブルよりも優先される、パフォーマンススキーマ `data_locks` および `data_lock_waits` テーブルによって公開されるロック情報について説明します。古い INFORMATION_SCHEMA テーブルに関して記述されている同様の説明は、[MySQL 5.7 Reference Manual](#) の [Using InnoDB Transaction and Locking Information](#) を参照してください。

ブロックしているトランザクションの識別

どのトランザクションが別のトランザクションをブロックしているかを識別すると役立つ場合があります。InnoDB トランザクションおよびデータロックに関する情報を含むテーブルを使用すると、どのトランザクションが別のトランザクションを待機しているか、およびどのリソースがリクエストされているかを判別できます。(これらのテーブルの詳細は、[セクション15.15.2「InnoDB INFORMATION_SCHEMA トランザクションおよびロック情報」](#) を参照してください。)

3つのセッションが同時に実行されているとします。各セッションは MySQL スレッドに対応し、あるトランザクションを別のトランザクションの後に実行します。これらのセッションが次のステートメントを発行したが、まだトランザクションをコミットしていない場合は、システムの状態を考慮してください:

- セッション A:

```
BEGIN;
SELECT a FROM t FOR UPDATE;
SELECT SLEEP(100);
```

- セッション B:

```
SELECT b FROM t FOR UPDATE;
```

- セッション C:

```
SELECT c FROM t FOR UPDATE;
```

このシナリオでは、次のクエリーを使用して、待機中のトランザクションおよびブロックしているトランザクションを確認します:

```
SELECT
```

```

r.trx_id waiting_trx_id,
r.trx_mysql_thread_id waiting_thread,
r.trx_query waiting_query,
b.trx_id blocking_trx_id,
b.trx_mysql_thread_id blocking_thread,
b.trx_query blocking_query
FROM performance_schema.data_lock_waits w
INNER JOIN information_schema.innodb_trx b
  ON b.trx_id = w.blocking_engine_transaction_id
INNER JOIN information_schema.innodb_trx r
  ON r.trx_id = w.requesting_engine_transaction_id;

```

または、より単純に `sys.innodb_lock_waits` ビューを使用します:

```

SELECT
  waiting_trx_id,
  waiting_pid,
  waiting_query,
  blocking_trx_id,
  blocking_pid,
  blocking_query
FROM sys.innodb_lock_waits;

```

ブロッキングクエリーに対して NULL 値がレポートされる場合は、[発行セッションがアイドル状態になった後のブロッキングクエリーの識別](#) を参照してください。

waiting trx id	waiting thread	waiting query	blocking trx id	blocking thread	blocking query
A4	6	SELECT b FROM t FOR UPDATE	A3	5	SELECT SLEEP(100)
A5	7	SELECT c FROM t FOR UPDATE	A3	5	SELECT SLEEP(100)
A5	7	SELECT c FROM t FOR UPDATE	A4	6	SELECT b FROM t FOR UPDATE

前述のテーブルでは、「待機中のクエリー」カラムまたは「ブロッキングクエリー」カラムでセッションを識別できます。次のことがわかります。

- セッション B (trx id A4、スレッド 6) とセッション C (trx id A5、スレッド 7) はどちらもセッション A (trx id A3、スレッド 5) を待機しています。
- セッション C はセッション B およびセッション A を待機しています。

基礎となるデータは、`INFORMATION_SCHEMA.INNODB_TRX` テーブルおよびパフォーマンススキーマの `data_locks` および `data_lock_waits` テーブルに表示されます。

次のテーブルに、`INNODB_TRX` テーブルのサンプルコンテンツを示します。

trx id	trx state	trx started	trx requested lock id	trx wait started	trx weight	trx mysql thread id	trx query
A3	RUNNING	2008-01-15 16:44:54	NULL	NULL	2	5	SELECT SLEEP(100)
A4	LOCK WAIT	2008-01-15 16:45:09	A4:1:3:2	2008-01-15 16:45:09	2	6	SELECT b FROM t FOR UPDATE
A5	LOCK WAIT	2008-01-15 16:45:14	A5:1:3:2	2008-01-15 16:45:14	2	7	SELECT c FROM t FOR UPDATE

次のテーブルに、`data_locks` テーブルのサンプルコンテンツを示します。

lock id	lock trx id	lock mode	lock type	スキーマのロック	lock table	lock index	lock data
A3:1:3:2	A3	X	RECORD	test	t	PRIMARY	0x0200

lock id	lock trx id	lock mode	lock type	スキーマの ロック	lock table	lock index	lock data
A4:1:3:2	A4	X	RECORD	test	t	PRIMARY	0x0200
A5:1:3:2	A5	X	RECORD	test	t	PRIMARY	0x0200

次のテーブルに、`data_lock_waits` テーブルのサンプルコンテンツを示します。

requesting trx id	requested lock id	blocking trx id	blocking lock id
A4	A4:1:3:2	A3	A3:1:3:2
A5	A5:1:3:2	A3	A3:1:3:2
A5	A5:1:3:2	A4	A4:1:3:2

発行セッションがアイドル状態になった後のブロッキングクエリーの識別

ブロッキングトランザクションを識別するときに、クエリーを発行したセッションがアイドル状態になった場合は、ブロッキングクエリーに対して NULL 値が報告されます。この場合は、次のステップを使用してブロッキングクエリーを決定します:

1. ブロッキングトランザクションのプロセスリスト ID を識別します。 `sys.innodb_lock_waits` テーブルでは、ブロッキングトランザクションのプロセスリスト ID は `blocking_pid` 値です。
2. `blocking_pid` を使用して、MySQL パフォーマンススキーマの `threads` テーブルをクエリーし、ブロックしているトランザクションの `THREAD_ID` を判別します。たとえば、`blocking_pid` が 6 の場合は、次のクエリーを発行します:

```
SELECT THREAD_ID FROM performance_schema.threads WHERE PROCESSLIST_ID = 6;
```

3. `THREAD_ID` を使用して、パフォーマンススキーマ `events_statements_current` テーブルをクエリーし、スレッドによって最後に実行されたクエリーを確認します。たとえば、`THREAD_ID` が 28 の場合は、次のクエリーを発行します:

```
SELECT THREAD_ID, SQL_TEXT FROM performance_schema.events_statements_current WHERE THREAD_ID = 28\G
```

4. スレッドによって実行された最後のクエリーが、ロックが保持されている理由を判断するのに十分な情報でない場合は、パフォーマンススキーマ `events_statements_history` テーブルをクエリーして、スレッドによって実行された最後の 10 個のステートメントを表示できます。

```
SELECT THREAD_ID, SQL_TEXT FROM performance_schema.events_statements_history WHERE THREAD_ID = 28 ORDER BY EVENT_ID;
```

InnoDB トランザクションと MySQL セッションの関連付け

内部 InnoDB ロック情報を、MySQL によって保持されるセッションレベルの情報と関連付けると便利な場合があります。たとえば、特定の InnoDB トランザクション ID について、対応する MySQL セッション ID とロックを保持している可能性があるセッションの名前を把握し、他のトランザクションをブロックする場合があります。

`INFORMATION_SCHEMA INNODB_TRX` テーブル、パフォーマンススキーマ `data_locks` および `data_lock_waits` テーブルからの次の出力は、ある程度ロードされたシステムから取得されます。表示されているように、複数のトランザクションが実行されています。

次の `data_locks` テーブルおよび `data_lock_waits` テーブルは、次のことを示しています:

- トランザクション 77F (INSERT を実行中) は、トランザクション 77E、77D、および 77B がコミットするのを待機しています。
- トランザクション 77E (INSERT を実行) は、トランザクション 77D および 77B のコミットを待機しています。
- トランザクション 77D (INSERT を実行) は、トランザクション 77B のコミットを待機しています。
- トランザクション 77B (INSERT を実行) は、トランザクション 77A のコミットを待機しています。

- トランザクション **77A** は実行中であり、現在 **SELECT** を実行しています。
- トランザクション **E56** (**INSERT** を実行中) は、トランザクション **E55** がコミットするのを待機しています。
- トランザクション **E55** (**INSERT** を実行中) は、トランザクション **19C** がコミットするのを待機しています。
- トランザクション **19C** は実行中であり、現在 **INSERT** を実行しています。

注記

INFORMATION_SCHEMA PROCESSLIST テーブルと **INNODB_TRX** テーブルに表示されるクエリーの間には不整合がある可能性があります。詳細は、[セクション15.15.2.3「InnoDB トランザクションおよびロック情報の永続性と一貫性」](#)を参照してください。

次のテーブルに、重い **workload** を実行しているシステムの **PROCESSLIST** テーブルの内容を示します。

ID	USER	HOST	DB	COMMAND	TIME	STATE	INFO
384	root	localhost	test	Query	10	update	INSERT INTO t2 VALUES ...
257	root	localhost	test	Query	3	update	INSERT INTO t2 VALUES ...
130	root	localhost	test	Query	0	update	INSERT INTO t2 VALUES ...
61	root	localhost	test	Query	1	update	INSERT INTO t2 VALUES ...
8	root	localhost	test	Query	1	update	INSERT INTO t2 VALUES ...
4	root	localhost	test	Query	0	preparing	SELECT * FROM PROCESSLIST
2	root	localhost	test	Sleep	566		NULL

次のテーブルに、重い **workload** を実行しているシステムの **INNODB_TRX** テーブルの内容を示します。

trx id	trx state	trx started	trx requested lock id	trx wait started	trx weight	trx mysql thread id	trx query
77F	LOCK WAIT	2008-01-15 13:10:16	77F	2008-01-15 13:10:16	1	876	INSERT INTO t09 (D, B, C) VALUES ...
77E	LOCK WAIT	2008-01-15 13:10:16	77E	2008-01-15 13:10:16	1	875	INSERT INTO t09 (D, B, C) VALUES ...
77D	LOCK WAIT	2008-01-15 13:10:16	77D	2008-01-15 13:10:16	1	874	INSERT INTO t09 (D, B, C) VALUES ...
77B	LOCK WAIT	2008-01-15 13:10:16	77B:733:12:1	2008-01-15 13:10:16	4	873	INSERT INTO t09

trx id	trx state	trx started	trx requested lock id	trx wait started	trx weight	trx mysql thread id	trx query
							(D, B, C) VALUES ...
77A	RUNNING	2008-01-15 13:10:16	NULL	NULL	4	872	SELECT b, c FROM t09 WHERE ...
E56	LOCK WAIT	2008-01-15 13:10:06	E56:743:6:2	2008-01-15 13:10:06	5	384	INSERT INTO t2 VALUES ...
E55	LOCK WAIT	2008-01-15 13:10:06	E55:743:38:2	2008-01-15 13:10:13	965	257	INSERT INTO t2 VALUES ...
19C	RUNNING	2008-01-15 13:09:10	NULL	NULL	2900	130	INSERT INTO t2 VALUES ...
E15	RUNNING	2008-01-15 13:08:59	NULL	NULL	5395	61	INSERT INTO t2 VALUES ...
51D	RUNNING	2008-01-15 13:08:47	NULL	NULL	9807	8	INSERT INTO t2 VALUES ...

次のテーブルに、重い workload を実行しているシステムの data_lock_waits テーブルの内容を示します。

requesting trx id	requested lock id	blocking trx id	blocking lock id
77F	77F:806	77E	77E:806
77F	77F:806	77D	77D:806
77F	77F:806	77B	77B:806
77E	77E:806	77D	77D:806
77E	77E:806	77B	77B:806
77D	77D:806	77B	77B:806
77B	77B:733:12:1	77A	77A:733:12:1
E56	E56:743:6:2	E55	E55:743:6:2
E55	E55:743:38:2	19C	19C:743:38:2

次のテーブルに、重い workload を実行しているシステムの data_locks テーブルの内容を示します。

lock id	lock trx id	lock mode	lock type	スキーマの ロック	lock table	lock index	lock data
77F:806	77F	AUTO_INC	TABLE	test	t09	NULL	NULL
77E:806	77E	AUTO_INC	TABLE	test	t09	NULL	NULL
77D:806	77D	AUTO_INC	TABLE	test	t09	NULL	NULL
77B:806	77B	AUTO_INC	TABLE	test	t09	NULL	NULL
77B:733:12:1	77B	X	RECORD	test	t09	PRIMARY	supremum pseudo- record
77A:733:12:1	77A	X	RECORD	test	t09	PRIMARY	supremum pseudo- record

lock id	lock trx id	lock mode	lock type	スキーマの ロック	lock table	lock index	lock data
E56:743:6:2	E56	S	RECORD	test	t2	PRIMARY	0, 0
E55:743:6:2	E55	X	RECORD	test	t2	PRIMARY	0, 0
E55:743:38:2	E55	S	RECORD	test	t2	PRIMARY	1922, 1922
19C:743:38:2	19C	X	RECORD	test	t2	PRIMARY	1922, 1922

15.15.2.2 InnoDB のロックおよびロック待機情報

注記

このセクションでは、MySQL 8.0 内の [INFORMATION_SCHEMA INNODB_LOCKS](#) および [INNODB_LOCK_WAITS](#) テーブルよりも優先される、パフォーマンススキーマ [data_locks](#) および [data_lock_waits](#) テーブルによって公開されるロック情報について説明します。古い [INFORMATION_SCHEMA](#) テーブルに関して記述されている同様の説明は、[MySQL 5.7 Reference Manual](#) の [InnoDB Lock and Lock-Wait Information](#) を参照してください。

トランザクションがテーブル内の行を更新するか、または [SELECT FOR UPDATE](#) でロックする場合、InnoDB はその行に関するロックのリストまたはキューを確立します。同様に、テーブルレベルのロックの場合、InnoDB はテーブルに関するロックのリストを保持します。2 番目のトランザクションが、互換性がないモードにある以前のトランザクションによってすでにロックされている行の更新またはテーブルのロックを行おうとした場合、InnoDB はその行に対するロック要求を対応するキューに追加します。トランザクションによってロックを取得するには、その行またはテーブルのロックキューに以前に入力されたすべての互換性のないロックリクエストを削除する必要があります（これらのロックを保持またはリクエストしているトランザクションがコミットまたはロールバックしたときに発生します）。

トランザクションは、異なる行またはテーブルに対する任意の数のロック要求を保持できます。トランザクションはいつでも、別のトランザクションによって保持されているロックを要求できますが、そのロックは、その別のトランザクションによってブロックされます。リクエスト側トランザクションは、ブロッキングロックを保持するトランザクションがコミットまたはロールバックされるまで待機する必要があります。トランザクションがロックを待機していない場合は、[RUNNING](#) 状態になります。トランザクションがロックを待機している場合は、[LOCK WAIT](#) 状態になります。（[INFORMATION_SCHEMA INNODB_TRX](#) テーブルは、トランザクションの状態の値を示します。）

パフォーマンススキーマ [data_locks](#) テーブルには、[LOCK WAIT](#) トランザクションごとに 1 つ以上の行が保持され、その進行を妨げるロック要求があることを示します。このテーブルにはまた、特定の行またはテーブルに対して保留されているロックのキュー内の各ロックを記述した 1 行も含まれています。パフォーマンススキーマ [data_lock_waits](#) テーブルには、ほかのトランザクションによって要求されたロックをブロックしているトランザクションによってすでに保持されているロックが表示されます。

15.15.2.3 InnoDB トランザクションおよびロック情報の永続性と一貫性

注記

このセクションでは、MySQL 8.0 内の [INFORMATION_SCHEMA INNODB_LOCKS](#) および [INNODB_LOCK_WAITS](#) テーブルよりも優先される、パフォーマンススキーマ [data_locks](#) および [data_lock_waits](#) テーブルによって公開されるロック情報について説明します。古い [INFORMATION_SCHEMA](#) テーブルに関して記述されている同様の説明は、[MySQL 5.7 Reference Manual](#) の [Persistence and Consistency of InnoDB Transaction and Locking Information](#) を参照してください。

トランザクションテーブルとロックテーブル ([INFORMATION_SCHEMA INNODB_TRX](#) テーブル、パフォーマンススキーマ [data_locks](#) テーブル、および [data_lock_waits](#) テーブル) によって公開されるデータは、高速変更データの概要を表します。これは、アプリケーションによって開始された更新が発生した場合にのみデータが変更されるユーザーテーブルとは異なります。基礎となるデータはシステム管理の内部データであり、非常に迅速に変更できます：

- データは、[INNODB_TRX](#)、[data_locks](#) および [data_lock_waits](#) テーブル間で一貫性がない場合があります。

[data_locks](#) および [data_lock_waits](#) テーブルは、[INNODB_TRX](#) テーブル内のトランザクションに関するロック情報を提供するために、InnoDB ストレージエンジンからライブデータを公開します。ロックテーブルから取得された

データは、`SELECT` の実行時に存在しますが、クエリー結果がクライアントによって消費されるまでに削除または変更される場合があります。

`data_locks` を `data_lock_waits` と結合すると、存在しない、またはまだ存在しない `data_locks` の親行を識別する `data_lock_waits` の行を表示できます。

- トランザクションテーブルおよびロックテーブルのデータは、`INFORMATION_SCHEMA PROCESLIST` テーブルまたはパフォーマンススキーマ `threads` テーブルのデータと整合性がとれていない可能性があります。

たとえば、InnoDB トランザクションのデータを比較し、テーブルを `PROCESLIST` テーブルのデータとロックする場合は注意が必要です。1つの `SELECT` (たとえば、`INNODB_TRX` と `PROCESLIST` の結合) を発行した場合でも、一般に、これらのテーブルの内容には整合性がありません。`INNODB_TRX` では、`PROCESLIST` に存在しない行や、`INNODB_TRX.TRX_QUERY` に表示されているトランザクションの現在実行中の SQL クエリーが `PROCESLIST.INFO` のものと異なる行を参照できます。

15.15.3 InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル

InnoDB `INFORMATION_SCHEMA` テーブルを使用して、InnoDB で管理されるスキーマオブジェクトに関するメタデータを抽出できます。この情報はデータディクショナリから取得されます。従来、このタイプの情報は、[セクション 15.17 「InnoDB モニター」](#) の手法を使用して取得し、InnoDB モニターを設定して、`SHOW ENGINE INNODB STATUS` ステートメントからの出力を解析します。InnoDB `INFORMATION_SCHEMA` テーブルのインターフェースを使用すると、SQL を使用してこのデータをクエリーできます。

InnoDB `INFORMATION_SCHEMA` スキーマオブジェクトテーブルには、次のテーブルが含まれます。

```
INNODB_DATAFILES
INNODB_TABLESTATS
INNODB_FOREIGN
INNODB_COLUMNS
INNODB_INDEXES
INNODB_FIELDS
INNODB_TABLESPACES
INNODB_TABLESPACES_BRIEF
INNODB_FOREIGN_COLS
INNODB_TABLES
```

これらのテーブル名は、提供されるデータのタイプを示しています。

- `INNODB_TABLES` は、InnoDB テーブルに関するメタデータを提供します。
- `INNODB_COLUMNS` は、InnoDB テーブルのカラムに関するメタデータを提供します。
- `INNODB_INDEXES` は、InnoDB インデックスに関するメタデータを提供します。
- `INNODB_FIELDS` では、InnoDB インデックスのキーカラム (フィールド) に関するメタデータが提供されます。
- `INNODB_TABLESTATS` では、メモリー内データ構造から導出された InnoDB テーブルに関する低レベルのステータス情報のビューが提供されます。
- `INNODB_DATAFILES` では、InnoDB file-per-table および一般テーブルスペースのデータファイルパス情報が提供されます。
- `INNODB_TABLESPACES` は、InnoDB file-per-table、general および undo テーブルスペースに関するメタデータを提供します。
- `INNODB_TABLESPACES_BRIEF` では、InnoDB テーブルスペースに関するメタデータのサブセットが提供されます。
- `INNODB_FOREIGN` は、InnoDB テーブルに定義されている外部キーに関するメタデータを提供します。
- `INNODB_FOREIGN_COLS` では、InnoDB テーブルに定義されている外部キーのカラムに関するメタデータが提供されます。

InnoDB `INFORMATION_SCHEMA` スキーマオブジェクトテーブルは、`TABLE_ID`、`INDEX_ID`、`SPACE` などのフィールドを使用して結合できるため、調査または監視するオブジェクトに使用可能なすべてのデータを簡単に取得できます。

各テーブルのカラムについては、[InnoDB INFORMATION_SCHEMA](#) のドキュメントを参照してください。

例 15.2 InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル

この例では、単一のインデックス (*i1*) を持つ単純なテーブル (*t1*) を使用して、[InnoDB INFORMATION_SCHEMA](#) スキーマオブジェクトテーブルにあるメタデータのタイプを示します。

1. テストデータベースとテーブル *t1* を作成します。

```
mysql> CREATE DATABASE test;

mysql> USE test;

mysql> CREATE TABLE t1 (
  col1 INT,
  col2 CHAR(10),
  col3 VARCHAR(10))
ENGINE = InnoDB;

mysql> CREATE INDEX i1 ON t1(col1);
```

2. テーブル *t1* を作成した後、[INNODB_TABLES](#) をクエリーして *test/t1* のメタデータを検索します:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TABLES WHERE NAME='test/t1'\G
***** 1. row *****
  TABLE_ID: 71
    NAME: test/t1
    FLAG: 1
    N_COLS: 6
    SPACE: 57
  ROW_FORMAT: Compact
  ZIP_PAGE_SIZE: 0
  INSTANT_COLS: 0
```

テーブル *t1* の [TABLE_ID](#) は 71 です。 [FLAG](#) フィールドは、テーブルの形式とストレージの特性に関するビットレベルの情報を提供します。6 つのカラムがあり、そのうちの 3 つが [InnoDB](#) によって作成された非表示のカラム ([DB_ROW_ID](#)、[DB_TRX_ID](#)、および [DB_ROLL_PTR](#)) です。このテーブルの [SPACE](#) の ID は 57 です (0 の値は、テーブルがシステムテーブルスペース内に存在することを示します)。 [ROW_FORMAT](#) はコンパクトです。 [ZIP_PAGE_SIZE](#) は、[Compressed](#) 行フォーマットのテーブルにのみ適用されます。 [INSTANT_COLS](#) では、[ALGORITHM=INSTANT](#) で [ALTER TABLE ... ADD COLUMN](#) を使用して最初のインスタントカラムを追加する前に、テーブルのカラム数が表示されます。

3. [INNODB_TABLES](#) の [TABLE_ID](#) 情報を使用して、[INNODB_COLUMNS](#) テーブルにテーブルのカラムに関する情報をクエリーします。

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_COLUMNS where TABLE_ID = 71\G
***** 1. row *****
  TABLE_ID: 71
    NAME: col1
    POS: 0
    MTYPE: 6
    PRYPE: 1027
    LEN: 4
  HAS_DEFAULT: 0
  DEFAULT_VALUE: NULL
***** 2. row *****
  TABLE_ID: 71
    NAME: col2
    POS: 1
    MTYPE: 2
    PRYPE: 524542
    LEN: 10
  HAS_DEFAULT: 0
  DEFAULT_VALUE: NULL
***** 3. row *****
  TABLE_ID: 71
    NAME: col3
    POS: 2
    MTYPE: 1
    PRYPE: 524303
    LEN: 10
```

```
HAS_DEFAULT: 0
DEFAULT_VALUE: NULL
```

TABLE_ID および **NAME** カラムに加えて、**INNODB_COLUMNS** は、(0 から始まり、順次増分する) 各カラムの順序位置 (**POS**)、**MTYPE** または「メインタイプ」(6 = INT, 2 = CHAR, 1 = VARCHAR)、**PRTYPE** または「正確な型」(MySQL データセット、文字セットコード、およびヌル可能性を示すビットを持つバイナリ値) およびコード長を表すリテラル (**LEN**) を提供します。**HAS_DEFAULT** および **DEFAULT_VALUE** のカラムは、**ALGORITHM=INSTANT** とともに **ALTER TABLE ... ADD COLUMN** を使用して即時に追加されたカラムのみ適用されます。

- INNODB_TABLES** の **TABLE_ID** 情報を再度使用して、テーブル **t1** に関連付けられたインデックスに関する情報を **INNODB_INDEXES** にクエリーします。

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_INDEXES WHERE TABLE_ID = 71 \G
***** 1. row *****
INDEX_ID: 111
NAME: GEN_CLUST_INDEX
TABLE_ID: 71
TYPE: 1
N_FIELDS: 0
PAGE_NO: 3
SPACE: 57
MERGE_THRESHOLD: 50
***** 2. row *****
INDEX_ID: 112
NAME: i1
TABLE_ID: 71
TYPE: 0
N_FIELDS: 1
PAGE_NO: 4
SPACE: 57
MERGE_THRESHOLD: 50
```

INNODB_INDEXES は、2 つのインデックスのデータを返します。最初のインデックスは **GEN_CLUST_INDEX** です。これは、テーブルにユーザー定義のクラスタ化されたインデックスが存在しない場合に **InnoDB** によって作成されたクラスタ化されたインデックスです。2 番目のインデックス (**i1**) は、ユーザー定義のセカンダリインデックスです。

INDEX_ID は、インスタンス内のすべてのデータベースにわたって一意であるインデックスの識別子です。**TABLE_ID** は、そのインデックスが関連付けられているテーブルを識別します。インデックスの **TYPE** 値は、インデックスのタイプ (1 = クラスタ化されたインデックス、0 = セカンダリインデックス) を示します。**N_FIELDS** 値は、このインデックスを構成するフィールドの数です。**PAGE_NO** はインデックスの B ツリーのルートページ番号であり、**SPACE** はインデックスが存在するテーブルスペースの ID です。ゼロ以外の値は、インデックスがシステムテーブルスペースに存在しないことを示します。**MERGE_THRESHOLD** では、インデックスページのデータ量のパーセンテージしきい値を定義します。行が削除されたとき、または更新操作によって行が短縮されたときに、インデックスページのデータ量がこの値 (デフォルトは 50%) を下回った場合、**InnoDB** はインデックススペースを隣接するインデックスページとマージしようとします。

- INNODB_INDEXES** の **INDEX_ID** 情報を使用して、**INNODB_FIELDS** にインデックス **i1** のフィールドに関する情報をクエリーします。

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FIELDS where INDEX_ID = 112 \G
***** 1. row *****
INDEX_ID: 112
NAME: col1
POS: 0
```

INNODB_FIELDS には、インデックス付きフィールドの **NAME** と、インデックス内での順序位置が用意されています。インデックス (**i1**) が複数のフィールドに定義されている場合、**INNODB_FIELDS** はインデックス付けされた各フィールドのメタデータを提供します。

- INNODB_TABLES** の **SPACE** 情報を使用して、**INNODB_TABLESPACES** テーブルにテーブルのテーブルスペースに関する情報をクエリーします。

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TABLESPACES WHERE SPACE = 57 \G
***** 1. row *****
SPACE: 57
NAME: test/t1
```

```

FLAG: 16417
ROW_FORMAT: Dynamic
PAGE_SIZE: 16384
ZIP_PAGE_SIZE: 0
SPACE_TYPE: Single
FS_BLOCK_SIZE: 4096
FILE_SIZE: 114688
ALLOCATED_SIZE: 98304
AUTOEXTEND_SIZE: 0
SERVER_VERSION: 8.0.23
SPACE_VERSION: 1
ENCRYPTION: N
STATE: normal

```

INNODB_TABLESPACES では、テーブルスペースの **SPACE ID** および関連付けられたテーブルの **NAME** に加えて、テーブルスペースのフォーマットおよび記憶特性に関するビットレベルの情報であるテーブルスペース **FLAG** データが提供されます。テーブルスペース **ROW_FORMAT**、**PAGE_SIZE** およびその他のいくつかのテーブルスペースメタデータ項目も用意されています。

7. **INNODB_TABLES** の **SPACE** 情報を再度使用して、**INNODB_DATAFILES** にテーブルスペースデータファイルの場所をクエリーします。

```

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_DATAFILES WHERE SPACE = 57 \G
***** 1. row *****
SPACE: 57
PATH: ./test/t1.ibd

```

データファイルは、MySQL の **data** ディレクトリの下での **test** ディレクトリにあります。 **file-per-table** テーブルスペースが **CREATE TABLE** ステートメントの **DATA DIRECTORY** 句を使用して MySQL データディレクトリ以外の場所に作成された場合、テーブルスペースの **PATH** は完全修飾のディレクトリパスになります。

8. 最後のステップとして、テーブル **t1** (**TABLE_ID = 71**) に行を挿入し、**INNODB_TABLESTATS** テーブルのデータを表示します。このテーブル内のデータは、**InnoDB** テーブルのクエリー時に使用するインデックスを決定するために MySQL オプティマイザによって使用されます。この情報は、インメモリーデータ構造から取得されます。

```

mysql> INSERT INTO t1 VALUES(5, 'abc', 'def');
Query OK, 1 row affected (0.06 sec)

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TABLESTATS where TABLE_ID = 71 \G
***** 1. row *****
TABLE_ID: 71
NAME: test/t1
STATS_INITIALIZED: Initialized
NUM_ROWS: 1
CLUST_INDEX_SIZE: 1
OTHER_INDEX_SIZE: 0
MODIFIED_COUNTER: 1
AUTOINC: 0
REF_COUNT: 1

```

STATS_INITIALIZED フィールドは、このテーブルの統計が収集されているかどうかを示します。 **NUM_ROWS** は、現在の推定されるテーブル内の行数です。 **CLUST_INDEX_SIZE** および **OTHER_INDEX_SIZE** フィールドはそれぞれ、テーブルのクラスタ化されたインデックスとセカンダリインデックスを格納するディスク上のページの数レポートします。 **MODIFIED_COUNTER** 値は、外部キーからの DML 操作およびカスケード操作によって変更された行数を示します。 **AUTOINC** 値は、自動インクリメントベースの操作に対して発行される次の番号です。テーブル **t1** では自動インクリメントカラムが定義されていないため、この値は 0 です。 **REF_COUNT** 値はカウンタです。このカウンタが 0 に達すると、テーブルキャッシュからテーブルメタデータを削除できることを示します。

例 15.3 外部キー INFORMATION_SCHEMA スキーマオブジェクトテーブル

INNODB_FOREIGN テーブルおよび **INNODB_FOREIGN_COLS** テーブルは、外部キー関係に関するデータを提供します。この例では、外部キー関係を持つ親テーブルと子テーブルを使用して、**INNODB_FOREIGN** テーブルと **INNODB_FOREIGN_COLS** テーブルで検出されたデータを示します。

1. テストデータベースおよび親テーブルと子テーブルを作成します。

```
mysql> CREATE DATABASE test;
```

```
mysql> USE test;

mysql> CREATE TABLE parent (id INT NOT NULL,
    PRIMARY KEY (id)) ENGINE=INNODB;

mysql> CREATE TABLE child (id INT, parent_id INT,
    INDEX par_ind (parent_id),
    CONSTRAINT fk1
    FOREIGN KEY (parent_id) REFERENCES parent(id)
    ON DELETE CASCADE) ENGINE=INNODB;
```

2. 親テーブルと子テーブルが作成されたら、`INNODB_FOREIGN` をクエリーして、`test/child` と `test/parent` の外部キー関係の外部キーデータを見つけます:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FOREIGN \G
***** 1. row *****
    ID: test/fk1
   FOR_NAME: test/child
   REF_NAME: test/parent
    N_COLS: 1
     TYPE: 1
```

メタデータには、子テーブルで定義された `CONSTRAINT` として指定されている外部キー ID (`fk1`) が含まれています。 `FOR_NAME` は、外部キーが定義されている子テーブルの名前です。 `REF_NAME` は、親テーブル(「参照される」テーブル)の名前です。 `N_COLS` は、外部キーのインデックス内のコラム数です。 `TYPE` は、外部キーコラムに関する追加情報を提供するビットフラグを表す数値です。この場合、`TYPE` 値は 1 です。これは、外部キーに対して `ON DELETE CASCADE` オプションが指定されたことを示します。 `TYPE` 値の詳細は、`INNODB_FOREIGN` テーブルの定義を参照してください。

3. 外部キー ID を使用して、`INNODB_FOREIGN_COLS` をクエリーして、外部キーのコラムに関するデータを表示します。

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FOREIGN_COLS WHERE ID = 'test/fk1' \G
***** 1. row *****
    ID: test/fk1
   FOR_COL_NAME: parent_id
   REF_COL_NAME: id
        POS: 0
```

`FOR_COL_NAME` は子テーブル内の外部キーコラムの名前であり、`REF_COL_NAME` は親テーブル内の参照されるコラムの名前です。 `POS` 値は、外部キーのインデックス内のキーフィールドの序数位置です (0 から始まります)。

例 15.4 InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブルの結合

この例では、`employees` サンプルデータベースのテーブルに関するファイル形式、行形式、ページサイズおよびインデックスサイズ情報を収集するために、3 つの `INFORMATION_SCHEMA` スキーマオブジェクトテーブル (`INNODB_TABLES`、`INNODB_TABLESPACES` および `INNODB_TABLESTATS`) を結合する方法を示します。

クエリー文字列を短くするために、次のテーブル名のエイリアスが使用されます。

- `INFORMATION_SCHEMA.INNODB_TABLES: a`
- `INFORMATION_SCHEMA.INNODB_TABLESPACES: b`
- `INFORMATION_SCHEMA.INNODB_TABLESTATS: c`

圧縮テーブルに対応するために、`IF()` 制御フロー関数が使用されています。テーブルが圧縮されている場合、インデックスサイズは `PAGE_SIZE` ではなく、`ZIP_PAGE_SIZE` を使用して計算されます。バイト単位でレポートされる `CLUST_INDEX_SIZE` および `OTHER_INDEX_SIZE` を `1024*1024` で割ると、M バイト (MB) 単位のインデックスサイズが得られます。MB 値は、`ROUND()` 関数を使用して小数点以下 0 桁に丸められます。

```
mysql> SELECT a.NAME, a.ROW_FORMAT,
    @page_size :=
    IF(a.ROW_FORMAT='Compressed',
    b.ZIP_PAGE_SIZE, b.PAGE_SIZE)
    AS page_size,
```

```

ROUND((@page_size * c.CLUST_INDEX_SIZE)
/(1024*1024)) AS pk_mb,
ROUND((@page_size * c.OTHER_INDEX_SIZE)
/(1024*1024)) AS secidx_mb
FROM INFORMATION_SCHEMA.INNODB_TABLES a
INNER JOIN INFORMATION_SCHEMA.INNODB_TABLESPACES b on a.NAME = b.NAME
INNER JOIN INFORMATION_SCHEMA.INNODB_TABLESTATS c on b.NAME = c.NAME
WHERE a.NAME LIKE 'employees/%'
ORDER BY a.NAME DESC;
+-----+-----+-----+-----+
| NAME          | ROW_FORMAT | page_size | pk_mb | secidx_mb |
+-----+-----+-----+-----+
| employees/titles | Dynamic   | 16384 | 20 | 11 |
| employees/salaries | Dynamic  | 16384 | 93 | 34 |
| employees/employees | Dynamic  | 16384 | 15 | 0 |
| employees/dept_manager | Dynamic | 16384 | 0 | 0 |
| employees/dept_emp | Dynamic  | 16384 | 12 | 10 |
| employees/departments | Dynamic | 16384 | 0 | 0 |
+-----+-----+-----+-----+

```

15.15.4 InnoDB INFORMATION_SCHEMA FULLTEXT インデックステーブル

次のテーブルに、[FULLTEXT](#) インデックスのメタデータを示します:

```

mysql> SHOW TABLES FROM INFORMATION_SCHEMA LIKE 'INNODB_FT%';
+-----+-----+
| Tables_in_INFORMATION_SCHEMA (INNODB_FT%) |
+-----+-----+
| INNODB_FT_CONFIG | |
| INNODB_FT_BEING_DELETED | |
| INNODB_FT_DELETED | |
| INNODB_FT_DEFAULT_STOPWORD | |
| INNODB_FT_INDEX_TABLE | |
| INNODB_FT_INDEX_CACHE | |
+-----+-----+

```

テーブルの概要

- **INNODB_FT_CONFIG**: InnoDB テーブルの **FULLTEXT** インデックスおよび関連する処理に関するメタデータを提供します。
- **INNODB_FT_BEING_DELETED**: **INNODB_FT_DELETED** テーブルのスナップショットを提供します。これは、**OPTIMIZE TABLE** メンテナンス操作中にのみ使用されます。**OPTIMIZE TABLE** を実行すると、**INNODB_FT_BEING_DELETED** テーブルが空になり、**INNODB_FT_DELETED** テーブルから **DOC_ID** 値が削除されます。**INNODB_FT_BEING_DELETED** の内容は一般に有効期間が短いため、モニタリングやデバッグでのこのテーブルの有用性は限られます。**FULLTEXT** インデックスを持つテーブルでの **OPTIMIZE TABLE** の実行の詳細は、[セクション12.10.6「MySQLの全文検索の微調整」](#)を参照してください。
- **INNODB_FT_DELETED**: InnoDB テーブルの **FULLTEXT** インデックスから削除された行を格納します。InnoDB **FULLTEXT** インデックスに対する DML 操作中の高コストのインデックス再編成を回避するために、新しく削除された単語に関する情報は個別に格納され、テキスト検索の実行時に検索結果から除外され、InnoDB テーブルに対して **OPTIMIZE TABLE** ステートメントを発行した場合にのみメイン検索インデックスから削除されます。
- **INNODB_FT_DEFAULT_STOPWORD**: InnoDB テーブルに **FULLTEXT** インデックスを作成するときにデフォルトで使用される **stopwords** のリストを保持します。
INNODB_FT_DEFAULT_STOPWORD テーブルについては、[セクション12.10.4「全文ストップワード」](#)を参照してください。
- **INNODB_FT_INDEX_TABLE**: InnoDB テーブルの **FULLTEXT** インデックスに対するテキスト検索の処理に使用される逆インデックスに関する情報を提供します。
- **INNODB_FT_INDEX_CACHE**: **FULLTEXT** インデックスに新しく挿入された行に関するトークン情報を提供します。DML 操作中の高コストのインデックス再編成を回避するために、新しくインデックス付けされたワードに関する情報は個別に格納され、**OPTIMIZE TABLE** の実行時、サーバーの停止時、またはキャッシュサイズが **innodb_ft_cache_size** または **innodb_ft_total_cache_size** システム変数で定義された制限を超えた場合にのみメイン検索インデックスと結合されます。

注記

INNODB_FT_DEFAULT_STOPWORD テーブルを除き、これらのテーブルは最初は空です。これらのいずれかをクエリーする前に、**innodb_ft_aux_table** システム変数の値を、**FULLTEXT** インデックスを含むテーブルの名前 (**test/articles** など) に設定します。

例 15.5 InnoDB FULLTEXT インデックスの INFORMATION_SCHEMA テーブル

この例では、**FULLTEXT** インデックスを含むテーブルを使用して、**FULLTEXT** インデックスの **INFORMATION_SCHEMA** テーブルに含まれているデータを示します。

1. **FULLTEXT** インデックスを含むテーブルを作成し、一部のデータを挿入します。

```
mysql> CREATE TABLE articles (
  id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
  title VARCHAR(200),
  body TEXT,
  FULLTEXT (title,body)
) ENGINE=InnoDB;

mysql> INSERT INTO articles (title,body) VALUES
('MySQL Tutorial','DBMS stands for DataBase ...'),
('How To Use MySQL Well','After you went through a ...'),
('Optimizing MySQL','In this tutorial we show ...'),
('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
('MySQL vs. YourSQL','In the following database comparison ...'),
('MySQL Security','When configured properly, MySQL ...');
```

2. **innodb_ft_aux_table** 変数を **FULLTEXT** インデックスを含むテーブルの名前に設定します。この変数が設定されていない場合、**INNODB_FT_DEFAULT_STOPWORD** を除き、**InnoDB FULLTEXT INFORMATION_SCHEMA** テーブルは空です。

```
mysql> SET GLOBAL innodb_ft_aux_table = 'test/articles';
```

3. **INNODB_FT_INDEX_CACHE** テーブルをクエリーします。これにより、**FULLTEXT** インデックス内の新しく挿入された行に関する情報が示されます。DML 操作中の高コストのインデックス再編成を回避するために、新しく挿入された行のデータは、**OPTIMIZE TABLE** が実行されるまで (またはサーバーが停止するか、キャッシュ制限を超えるまで)、**FULLTEXT** インデックスキャッシュに残ります。

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_INDEX_CACHE LIMIT 5;
+-----+-----+-----+-----+-----+-----+
| WORD   | FIRST_DOC_ID | LAST_DOC_ID | DOC_COUNT | DOC_ID | POSITION |
+-----+-----+-----+-----+-----+-----+
| 1001   | 5 | 5 | 1 | 5 | 0 |
| after  | 3 | 3 | 1 | 3 | 22 |
| comparison | 6 | 6 | 1 | 6 | 44 |
| configured | 7 | 7 | 1 | 7 | 20 |
| database | 2 | 6 | 2 | 2 | 31 |
+-----+-----+-----+-----+-----+-----+
```

4. **innodb_optimize_fulltext_only** システム変数を有効にし、**FULLTEXT** インデックスを含むテーブルで **OPTIMIZE TABLE** を実行します。この操作により、**FULLTEXT** インデックスキャッシュの内容がメインの **FULLTEXT** インデックスにフラッシュされます。**innodb_optimize_fulltext_only** は、**InnoDB** テーブルでの **OPTIMIZE TABLE** ステートメントの動作方法を変更するものであり、**FULLTEXT** インデックスを含む **InnoDB** テーブルでの保守操作中に一時的に有効にすることを目的としています。

```
mysql> SET GLOBAL innodb_optimize_fulltext_only=ON;
```

```
mysql> OPTIMIZE TABLE articles;
+-----+-----+-----+-----+
| Table   | Op   | Msg_type | Msg_text |
+-----+-----+-----+-----+
| test.articles | optimize | status | OK |
+-----+-----+-----+-----+
```

5. **INNODB_FT_INDEX_TABLE** テーブルにクエリーして、メインの **FULLTEXT** インデックス内のデータに関する情報 (**FULLTEXT** インデックスキャッシュからフラッシュされたばかりのデータに関する情報を含む) を表示します。

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNOODB_FT_INDEX_TABLE LIMIT 5;
+-----+-----+-----+-----+-----+
| WORD      | FIRST_DOC_ID | LAST_DOC_ID | DOC_COUNT | POSITION |
+-----+-----+-----+-----+-----+
| 1001     | 5 | 5 | 1 | 5 | 0 |
| after    | 3 | 3 | 1 | 3 | 22 |
| comparison | 6 | 6 | 1 | 6 | 44 |
| configured | 7 | 7 | 1 | 7 | 20 |
| database | 2 | 6 | 2 | 2 | 31 |
+-----+-----+-----+-----+-----+
```

OPTIMIZE TABLE 操作によって FULLTEXT インデックスキャッシュがフラッシュされたため、INNOODB_FT_INDEX_CACHE テーブルは空になっています。

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNOODB_FT_INDEX_CACHE LIMIT 5;
Empty set (0.00 sec)
```

6. test/articles テーブルからいくつかのレコードを削除します。

```
mysql> DELETE FROM test.articles WHERE id < 4;
```

7. INNOODB_FT_DELETED テーブルをクエリーします。このテーブルには、FULLTEXT インデックスから削除された行が記録されます。DML 操作中にコストの高いインデックス再編成が行われないようにするために、新しく削除されたレコードに関する情報は個別に格納され、テキスト検索を実行すると検索結果からフィルタで除外され、OPTIMIZE TABLE を実行するとメインの検索インデックスから削除されます。

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNOODB_FT_DELETED;
+-----+
| DOC_ID |
+-----+
| 2 |
| 3 |
| 4 |
+-----+
```

8. OPTIMIZE TABLE を実行して、削除されたレコードを消去します。

```
mysql> OPTIMIZE TABLE articles;
+-----+-----+-----+-----+
| Table      | Op      | Msg_type | Msg_text |
+-----+-----+-----+-----+
| test.articles | optimize | status   | OK       |
+-----+-----+-----+-----+
```

これで、INNOODB_FT_DELETED テーブルは空になります。

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNOODB_FT_DELETED;
Empty set (0.00 sec)
```

9. INNOODB_FT_CONFIG テーブルをクエリーします。このテーブルには、FULLTEXT インデックスに関するメタデータとそれに関連する処理が含まれています。

- optimize_checkpoint_limit: OPTIMIZE TABLE の実行が停止するまでの秒数。
- synced_doc_id: 次に発行される DOC_ID です。
- stopword_table_name: ユーザー定義のストップワードテーブルの database/table 名。ユーザー定義のストップワードテーブルがない場合、VALUE カラムは空です。
- use_stopword: FULLTEXT インデックスの作成時に定義されるストップワードテーブルを使用するかどうかを示します。

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNOODB_FT_CONFIG;
+-----+-----+
| KEY          | VALUE |
+-----+-----+
| optimize_checkpoint_limit | 180 |
| synced_doc_id      | 8 |
| stopword_table_name |      |
+-----+-----+
```

```
| use_stopword | 1 |
+-----+-----+
```

10. `innodb_optimize_fulltext_only` は一時的にのみ有効にすることを意図しているため、無効にします:

```
mysql> SET GLOBAL innodb_optimize_fulltext_only=OFF;
```

15.15.5 InnoDB INFORMATION_SCHEMA バッファプールテーブル

InnoDB INFORMATION_SCHEMA バッファプールテーブルは、バッファプールのステータス情報、および InnoDB バッファプール内のページに関するメタデータを提供します。

InnoDB INFORMATION_SCHEMA バッファプールテーブルには、下に一覧表示されているものが含まれます。

```
mysql> SHOW TABLES FROM INFORMATION_SCHEMA LIKE 'INNODB_BUFFER%';
```

```
+-----+
| Tables_in_INFORMATION_SCHEMA (INNODB_BUFFER%) |
+-----+
| INNODB_BUFFER_PAGE_LRU |
| INNODB_BUFFER_PAGE |
| INNODB_BUFFER_POOL_STATS |
+-----+
```

テーブルの概要

- `INNODB_BUFFER_PAGE`: InnoDB バッファプール内の各ページに関する情報を保持します。
- `INNODB_BUFFER_PAGE_LRU`: InnoDB バッファプール内のページに関する情報、特に、いっぱいになったときにバッファプールからどのページを削除するかを決定する LRU リスト内の各ページの順序を保持します。 `INNODB_BUFFER_PAGE_LRU` テーブルには、 `INNODB_BUFFER_PAGE` テーブルと同じカラムがありますが、 `INNODB_BUFFER_PAGE_LRU` テーブルには `BLOCK_ID` カラムではなく `LRU_POSITION` カラムがある点が異なります。
- `INNODB_BUFFER_POOL_STATS`: バッファプールのステータス情報を提供します。 同じ情報のほとんどは、 `SHOW ENGINE INNODB STATUS` の出力で提供されるか、または InnoDB バッファプールのサーバーステータス変数を使用して取得できます。

警告

`INNODB_BUFFER_PAGE` または `INNODB_BUFFER_PAGE_LRU` テーブルをクエリーすると、パフォーマンスに影響する可能性があります。 パフォーマンスへの影響を認識し、許容できると判断した場合を除き、本番システムでこれらのテーブルをクエリーしないでください。 本番システムのパフォーマンスへの影響を回避するには、調査する問題を再現し、テストインスタンスのバッファプール統計をクエリーします。

例 15.6 INNODB_BUFFER_PAGE テーブル内のシステムデータのクエリー

このクエリーでは、 `TABLE_NAME` 値が `NULL` であるページ、またはユーザー定義テーブルを示すスラッシュ/またはピリオド . がテーブル名に含まれるページを除外することで、システムデータを含むページの概算数が提供されます。

```
mysql> SELECT COUNT(*) FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
WHERE TABLE_NAME IS NULL OR (INSTR(TABLE_NAME, '/') = 0 AND INSTR(TABLE_NAME, '.') = 0);
+-----+
| COUNT(*) |
+-----+
| 1516 |
+-----+
```

このクエリーは、システムデータを含むページの概数、バッファプールページの総数、およびシステムデータを含むページの概略の割合 (%) を返します。

```
mysql> SELECT
(SELECT COUNT(*) FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
WHERE TABLE_NAME IS NULL OR (INSTR(TABLE_NAME, '/') = 0 AND INSTR(TABLE_NAME, '.') = 0)
```

```

) AS system_pages,
(
SELECT COUNT(*)
FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
) AS total_pages,
(
SELECT ROUND((system_pages/total_pages) * 100)
) AS system_page_percentage;
+-----+-----+-----+
| system_pages | total_pages | system_page_percentage |
+-----+-----+-----+
|      295 |      8192 |           4 |
+-----+-----+-----+

```

バッファプール内のシステムデータのタイプは、`PAGE_TYPE` 値をクエリーすることによって確認できます。たとえば、次のクエリーは、システムデータを含むページ間の 8 つの個別の `PAGE_TYPE` 値を返します。

```

mysql> SELECT DISTINCT PAGE_TYPE FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
WHERE TABLE_NAME IS NULL OR (INSTR(TABLE_NAME, '/') = 0 AND INSTR(TABLE_NAME, '.') = 0);
+-----+
| PAGE_TYPE |
+-----+
| SYSTEM    |
| IBUF_BITMAP |
| UNKNOWN   |
| FILE_SPACE_HEADER |
| INODE     |
| UNDO_LOG  |
| ALLOCATED |
+-----+

```

例 15.7 INNODB_BUFFER_PAGE テーブル内のユーザーデータのクエリー

このクエリーでは、`TABLE_NAME` 値が `NOT NULL` および `NOT LIKE '%INNODB_TABLES%'` であるページをカウントすることで、ユーザーデータを含むページの概算数が提供されます。

```

mysql> SELECT COUNT(*) FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
WHERE TABLE_NAME IS NOT NULL AND TABLE_NAME NOT LIKE '%INNODB_TABLES%';
+-----+
| COUNT(*) |
+-----+
|      7897 |
+-----+

```

このクエリーは、ユーザーデータを含むページの概数、バッファプールページの総数、およびユーザーデータを含むページの概略の割合 (%) を返します。

```

mysql> SELECT
(SELECT COUNT(*) FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
WHERE TABLE_NAME IS NOT NULL AND (INSTR(TABLE_NAME, '/') > 0 OR INSTR(TABLE_NAME, '.') > 0)
) AS user_pages,
(
SELECT COUNT(*)
FROM information_schema.INNODB_BUFFER_PAGE
) AS total_pages,
(
SELECT ROUND((user_pages/total_pages) * 100)
) AS user_page_percentage;
+-----+-----+-----+
| user_pages | total_pages | user_page_percentage |
+-----+-----+-----+
|      7897 |      8192 |           96 |
+-----+-----+-----+

```

このクエリーは、バッファプール内のページを含むユーザー定義のテーブルを識別します。

```

mysql> SELECT DISTINCT TABLE_NAME FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
WHERE TABLE_NAME IS NOT NULL AND (INSTR(TABLE_NAME, '/') > 0 OR INSTR(TABLE_NAME, '.') > 0)
AND TABLE_NAME NOT LIKE "mysql`.`innodb_%";
+-----+
| TABLE_NAME |
+-----+

```

```
| `employees`.`salaries` |
| `employees`.`employees` |
+-----+
```

例 15.8 INNODB_BUFFER_PAGE テーブル内のインデックスデータのクエリー

インデックスページに関する情報を取得するには、そのインデックスの名前を使用して `INDEX_NAME` カラムをクエリーします。たとえば、次のクエリーは、`employees.salaries` テーブルで定義されている `emp_no` インデックスのページの数とページの合計データサイズを返します。

```
mysql> SELECT INDEX_NAME, COUNT(*) AS Pages,
ROUND(SUM(IF(COMPRESSED_SIZE = 0, @@GLOBAL.innodb_page_size, COMPRESSED_SIZE))/1024/1024)
AS 'Total Data (MB)'
FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
WHERE INDEX_NAME='emp_no' AND TABLE_NAME = 'employees`.`salaries`;
+-----+-----+-----+
| INDEX_NAME | Pages | Total Data (MB) |
+-----+-----+-----+
| emp_no    | 1609 | 25              |
+-----+-----+-----+
```

このクエリーは、`employees.salaries` テーブルで定義されているすべてのインデックスのページの数とページの合計データサイズを返します。

```
mysql> SELECT INDEX_NAME, COUNT(*) AS Pages,
ROUND(SUM(IF(COMPRESSED_SIZE = 0, @@GLOBAL.innodb_page_size, COMPRESSED_SIZE))/1024/1024)
AS 'Total Data (MB)'
FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
WHERE TABLE_NAME = 'employees`.`salaries'
GROUP BY INDEX_NAME;
+-----+-----+-----+
| INDEX_NAME | Pages | Total Data (MB) |
+-----+-----+-----+
| emp_no    | 1608 | 25              |
| PRIMARY   | 6086 | 95              |
+-----+-----+-----+
```

例 15.9 INNODB_BUFFER_PAGE_LRU テーブル内の LRU_POSITION データのクエリー

`INNODB_BUFFER_PAGE_LRU` テーブルは、InnoDB バッファプール内のページに関する情報、特に、いっぱいになったときにバッファプールからどのページを削除するかを決定する各ページの順序を保持しています。このページの定義は、このテーブルには `BLOCK_ID` カラムの代わりに `LRU_POSITION` カラムがある点を除き、`INNODB_BUFFER_PAGE` の場合と同じです。

このクエリーは、`employees.employees` テーブルの各ページによって占有されている LRU リスト内の特定の場所にある位置の数をカウントします。

```
mysql> SELECT COUNT(LRU_POSITION) FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE_LRU
WHERE TABLE_NAME='employees`.`employees' AND LRU_POSITION < 3072;
+-----+
| COUNT(LRU_POSITION) |
+-----+
| 548                  |
+-----+
```

例 15.10 INNODB_BUFFER_POOL_STATS テーブルのクエリー

`INNODB_BUFFER_POOL_STATS` テーブルは、`SHOW ENGINE INNODB STATUS` および InnoDB バッファプールのステータス変数と同様の情報を提供します。

```
mysql> SELECT * FROM information_schema.INNODB_BUFFER_POOL_STATS \G
***** 1. row *****
      POOL_ID: 0
      POOL_SIZE: 8192
      FREE_BUFFERS: 1
      DATABASE_PAGES: 8173
      OLD_DATABASE_PAGES: 3014
      MODIFIED_DATABASE_PAGES: 0
```

```

PENDING_DECOMPRESS: 0
  PENDING_READS: 0
  PENDING_FLUSH_LRU: 0
  PENDING_FLUSH_LIST: 0
  PAGES_MADE_YOUNG: 15907
  PAGES_NOT_MADE_YOUNG: 3803101
  PAGES_MADE_YOUNG_RATE: 0
  PAGES_MADE_NOT_YOUNG_RATE: 0
  NUMBER_PAGES_READ: 3270
  NUMBER_PAGES_CREATED: 13176
  NUMBER_PAGES_WRITTEN: 15109
  PAGES_READ_RATE: 0
  PAGES_CREATE_RATE: 0
  PAGES_WRITTEN_RATE: 0
  NUMBER_PAGES_GET: 33069332
  HIT_RATE: 0
  YOUNG_MAKE_PER_THOUSAND_GETS: 0
  NOT_YOUNG_MAKE_PER_THOUSAND_GETS: 0
  NUMBER_PAGES_READ_AHEAD: 2713
  NUMBER_READ_AHEAD_EVICTED: 0
  READ_AHEAD_RATE: 0
  READ_AHEAD_EVICTED_RATE: 0
  LRU_IO_TOTAL: 0
  LRU_IO_CURRENT: 0
  UNCOMPRESS_TOTAL: 0
  UNCOMPRESS_CURRENT: 0

```

比較のために、同じデータセットに基づいた `SHOW ENGINE INNODB STATUS` の出力および InnoDB バッファプールのステータス変数の出力を次に示します。

`SHOW ENGINE INNODB STATUS` の出力の詳細は、[セクション15.17.3「InnoDB 標準モニターおよびロックモニターの出力」](#)を参照してください。

```

mysql> SHOW ENGINE INNODB STATUS \G
...
-----
BUFFER POOL AND MEMORY
-----
Total large memory allocated 137428992
Dictionary memory allocated 579084
Buffer pool size 8192
Free buffers 1
Database pages 8173
Old database pages 3014
Modified db pages 0
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 15907, not young 3803101
0.00 young/s, 0.00 non-young/s
Pages read 3270, created 13176, written 15109
0.00 reads/s, 0.00 creates/s, 0.00 writes/s
No buffer pool page gets since the last printout
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s
LRU len: 8173, unzip_LRU len: 0
I/O sum[0]:cur[0], unzip sum[0]:cur[0]
...

```

ステータス変数の説明については、[セクション5.1.10「サーバーステータス変数」](#)を参照してください。

```

mysql> SHOW STATUS LIKE 'Innodb_buffer%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Innodb_buffer_pool_dump_status | not started |
| Innodb_buffer_pool_load_status | not started |
| Innodb_buffer_pool_resize_status | not started |
| Innodb_buffer_pool_pages_data | 8173 |
| Innodb_buffer_pool_bytes_data | 133906432 |
| Innodb_buffer_pool_pages_dirty | 0 |
| Innodb_buffer_pool_bytes_dirty | 0 |
| Innodb_buffer_pool_pages_flushed | 15109 |
| Innodb_buffer_pool_pages_free | 1 |
| Innodb_buffer_pool_pages_misc | 18 |

```



```

| InnoDB_buffer_pool_pages_total | 8192 |
| InnoDB_buffer_pool_read_ahead_rnd | 0 |
| InnoDB_buffer_pool_read_ahead | 2713 |
| InnoDB_buffer_pool_read_ahead_evicted | 0 |
| InnoDB_buffer_pool_read_requests | 33069332 |
| InnoDB_buffer_pool_reads | 558 |
| InnoDB_buffer_pool_wait_free | 0 |
| InnoDB_buffer_pool_write_requests | 11985961 |
+-----+-----+

```

15.15.6 InnoDB INFORMATION_SCHEMA メトリックテーブル

`INNODB_METRICS` のテーブルには、`InnoDB` のパフォーマンスおよびリソース関連のカウンタに関する情報が表示されます。

`INNODB_METRICS` テーブルのカラムを次に示します。カラムの説明は、[セクション 26.51.22 「INFORMATION_SCHEMA INNODB_METRICS テーブル」](#) を参照してください。

```

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME="dml_inserts" \G
***** 1. row *****
      NAME: dml_inserts
  SUBSYSTEM: dml
     COUNT: 46273
  MAX_COUNT: 46273
  MIN_COUNT: NULL
  AVG_COUNT: 492.2659574468085
  COUNT_RESET: 46273
  MAX_COUNT_RESET: 46273
  MIN_COUNT_RESET: NULL
  AVG_COUNT_RESET: NULL
  TIME_ENABLED: 2014-11-28 16:07:53
  TIME_DISABLED: NULL
  TIME_ELAPSED: 94
  TIME_RESET: NULL
     STATUS: enabled
     TYPE: status_counter
  COMMENT: Number of rows inserted

```

カウンタの有効化、無効化、およびリセット

次の変数を使用して、カウンタを有効化、無効化およびリセットできます：

- `innodb_monitor_enable`: カウンタを有効にします。

```
SET GLOBAL innodb_monitor_enable = [counter-name|module_name|pattern|all];
```

- `innodb_monitor_disable`: カウンタを無効にします。

```
SET GLOBAL innodb_monitor_disable = [counter-name|module_name|pattern|all];
```

- `innodb_monitor_reset`: カウンタ値をゼロにリセットします。

```
SET GLOBAL innodb_monitor_reset = [counter-name|module_name|pattern|all];
```

- `innodb_monitor_reset_all`: すべてのカウンタ値をリセットします。 `innodb_monitor_reset_all` を使用する前にカウンタを無効にする必要があります。

```
SET GLOBAL innodb_monitor_reset_all = [counter-name|module_name|pattern|all];
```

カウンタおよびカウンタモジュールは、起動時に MySQL サーバー構成ファイルを使用して有効にすることもできます。たとえば、`log` モジュール、`metadata_table_handles_opened` および `metadata_table_handles_closed` カウンタを有効にするには、MySQL サーバー構成ファイルの `[mysqld]` セクションに次の行を入力します。

```

[mysqld]
innodb_monitor_enable = module_recovery,metadata_table_handles_opened,metadata_table_handles_closed

```

構成ファイルで複数のカウンタまたはモジュールを有効にする場合は、前述のように、`innodb_monitor_enable` 変数の後にカウンタ名とモジュール名をカンマで区切って指定します。構成ファイルで使用できるのは

`innodb_monitor_enable` 変数のみです。 `innodb_monitor_disable` および `innodb_monitor_reset` 変数は、コマンドラインでのみサポートされています。

注記

各カウンタはランタイムオーバーヘッドの程度を追加するため、本番サーバーでカウンタを保守的に使用して、特定の問題を診断したり、特定の機能を監視します。カウンタをより広範囲に使用するには、テストサーバーまたは開発サーバーをお勧めします。

カウンタ

使用可能なカウンタのリストは変更される可能性があります。使用している MySQL サーバーバージョンで使用可能なカウンタを `INFORMATION_SCHEMA.INNODB_METRICS` テーブルにクエリーします。

デフォルトで有効になっているカウンタは、`SHOW ENGINE INNODB STATUS` 出力に表示されるカウンタに対応しています。 `SHOW ENGINE INNODB STATUS` 出力に表示されるカウンタは、常にシステムレベルで有効になりますが、`INNODB_METRICS` テーブルでは無効にできます。カウンタステータスは永続的ではありません。特に構成されていないかぎり、カウンタはサーバーの再起動時にデフォルトの有効または無効ステータスに戻ります。

カウンタの追加または削除の影響を受けるプログラムを実行する場合は、リリースノートを確認し、`INNODB_METRICS` テーブルをクエリーして、それらの変更をアップグレードプロセスの一部として識別することをお勧めします。

```
mysql> SELECT name, subsystem, status FROM INFORMATION_SCHEMA.INNODB_METRICS ORDER BY NAME;
```

name	subsystem	status
adaptive_hash_pages_added	adaptive_hash_index	disabled
adaptive_hash_pages_removed	adaptive_hash_index	disabled
adaptive_hash_rows_added	adaptive_hash_index	disabled
adaptive_hash_rows_deleted_no_hash_entry	adaptive_hash_index	disabled
adaptive_hash_rows_removed	adaptive_hash_index	disabled
adaptive_hash_rows_updated	adaptive_hash_index	disabled
adaptive_hash_searches	adaptive_hash_index	enabled
adaptive_hash_searches_btree	adaptive_hash_index	enabled
buffer_data_reads	buffer	enabled
buffer_data_written	buffer	enabled
buffer_flush_adaptive	buffer	disabled
buffer_flush_adaptive_avg_pass	buffer	disabled
buffer_flush_adaptive_avg_time_est	buffer	disabled
buffer_flush_adaptive_avg_time_slot	buffer	disabled
buffer_flush_adaptive_avg_time_thread	buffer	disabled
buffer_flush_adaptive_pages	buffer	disabled
buffer_flush_adaptive_total_pages	buffer	disabled
buffer_flush_avg_page_rate	buffer	disabled
buffer_flush_avg_pass	buffer	disabled
buffer_flush_avg_time	buffer	disabled
buffer_flush_background	buffer	disabled
buffer_flush_background_pages	buffer	disabled
buffer_flush_background_total_pages	buffer	disabled
buffer_flush_batches	buffer	disabled
buffer_flush_batch_num_scan	buffer	disabled
buffer_flush_batch_pages	buffer	disabled
buffer_flush_batch_scanned	buffer	disabled
buffer_flush_batch_scanned_per_call	buffer	disabled
buffer_flush_batch_total_pages	buffer	disabled
buffer_flush_lsn_avg_rate	buffer	disabled
buffer_flush_neighbor	buffer	disabled
buffer_flush_neighbor_pages	buffer	disabled
buffer_flush_neighbor_total_pages	buffer	disabled
buffer_flush_n_to_flush_by_age	buffer	disabled
buffer_flush_n_to_flush_requested	buffer	disabled
buffer_flush_pct_for_dirty	buffer	disabled
buffer_flush_pct_for_lsn	buffer	disabled
buffer_flush_sync	buffer	disabled
buffer_flush_sync_pages	buffer	disabled
buffer_flush_sync_total_pages	buffer	disabled
buffer_flush_sync_waits	buffer	disabled
buffer_LRU_batches_evict	buffer	disabled
buffer_LRU_batches_flush	buffer	disabled
buffer_LRU_batch_evict_pages	buffer	disabled

buffer_LRU_batch_evict_total_pages	buffer	disabled
buffer_LRU_batch_flush_avg_pass	buffer	disabled
buffer_LRU_batch_flush_avg_time_est	buffer	disabled
buffer_LRU_batch_flush_avg_time_slot	buffer	disabled
buffer_LRU_batch_flush_avg_time_thread	buffer	disabled
buffer_LRU_batch_flush_pages	buffer	disabled
buffer_LRU_batch_flush_total_pages	buffer	disabled
buffer_LRU_batch_num_scan	buffer	disabled
buffer_LRU_batch_scanned	buffer	disabled
buffer_LRU_batch_scanned_per_call	buffer	disabled
buffer_LRU_get_free_loops	buffer	disabled
buffer_LRU_get_free_search	Buffer	disabled
buffer_LRU_get_free_waits	buffer	disabled
buffer_LRU_search_num_scan	buffer	disabled
buffer_LRU_search_scanned	buffer	disabled
buffer_LRU_search_scanned_per_call	buffer	disabled
buffer_LRU_single_flush_failure_count	Buffer	disabled
buffer_LRU_single_flush_num_scan	buffer	disabled
buffer_LRU_single_flush_scanned	buffer	disabled
buffer_LRU_single_flush_scanned_per_call	buffer	disabled
buffer_LRU_unzip_search_num_scan	buffer	disabled
buffer_LRU_unzip_search_scanned	buffer	disabled
buffer_LRU_unzip_search_scanned_per_call	buffer	disabled
buffer_pages_created	buffer	enabled
buffer_pages_read	buffer	enabled
buffer_pages_written	buffer	enabled
buffer_page_read_blob	buffer_page_io	disabled
buffer_page_read_read_fsp_hdr	buffer_page_io	disabled
buffer_page_read_ibuf_bitmap	buffer_page_io	disabled
buffer_page_read_ibuf_free_list	buffer_page_io	disabled
buffer_page_read_index_ibuf_leaf	buffer_page_io	disabled
buffer_page_read_index_ibuf_non_leaf	buffer_page_io	disabled
buffer_page_read_index_inode	buffer_page_io	disabled
buffer_page_read_index_leaf	buffer_page_io	disabled
buffer_page_read_index_non_leaf	buffer_page_io	disabled
buffer_page_read_other	buffer_page_io	disabled
buffer_page_read_system_page	buffer_page_io	disabled
buffer_page_read_trx_system	buffer_page_io	disabled
buffer_page_read_undo_log	buffer_page_io	disabled
buffer_page_read_xdes	buffer_page_io	disabled
buffer_page_read_zblob	buffer_page_io	disabled
buffer_page_read_zblob2	buffer_page_io	disabled
buffer_page_written_blob	buffer_page_io	disabled
buffer_page_written_fsp_hdr	buffer_page_io	disabled
buffer_page_written_ibuf_bitmap	buffer_page_io	disabled
buffer_page_written_ibuf_free_list	buffer_page_io	disabled
buffer_page_written_index_ibuf_leaf	buffer_page_io	disabled
buffer_page_written_index_ibuf_non_leaf	buffer_page_io	disabled
buffer_page_written_index_inode	buffer_page_io	disabled
buffer_page_written_index_leaf	buffer_page_io	disabled
buffer_page_written_index_non_leaf	buffer_page_io	disabled
buffer_page_written_other	buffer_page_io	disabled
buffer_page_written_system_page	buffer_page_io	disabled
buffer_page_written_trx_system	buffer_page_io	disabled
buffer_page_written_undo_log	buffer_page_io	disabled
buffer_page_written_xdes	buffer_page_io	disabled
buffer_page_written_zblob	buffer_page_io	disabled
buffer_page_written_zblob2	buffer_page_io	disabled
buffer_pool_bytes_data	buffer	enabled
buffer_pool_bytes_dirty	buffer	enabled
buffer_pool_pages_data	buffer	enabled
buffer_pool_pages_dirty	buffer	enabled
buffer_pool_pages_free	buffer	enabled
buffer_pool_pages_misc	buffer	enabled
buffer_pool_pages_total	buffer	enabled
buffer_pool_reads	buffer	enabled
buffer_pool_read_ahead	buffer	enabled
buffer_pool_read_ahead_evicted	buffer	enabled
buffer_pool_read_requests	buffer	enabled
buffer_pool_size	server	enabled
buffer_pool_wait_free	buffer	enabled
buffer_pool_write_requests	buffer	enabled
compression_pad_decrements	compression	disabled
compression_pad_increments	compression	disabled
compress_pages_compressed	compression	disabled

compress_pages_decompressed	compression	disabled
ddl_background_drop_indexes	ddl	disabled
ddl_background_drop_tables	ddl	disabled
ddl_log_file_alter_table	ddl	disabled
ddl_online_create_index	ddl	disabled
ddl_pending_alter_table	ddl	disabled
ddl_sort_file_alter_table	ddl	disabled
dml_deletes	dml	enabled
dml_inserts	dml	enabled
dml_reads	dml	disabled
dml_updates	dml	enabled
file_num_open_files	file_system	enabled
ibuf_merges	change_buffer	enabled
ibuf_merges_delete	change_buffer	enabled
ibuf_merges_delete_mark	change_buffer	enabled
ibuf_merges_discard_delete	change_buffer	enabled
ibuf_merges_discard_delete_mark	change_buffer	enabled
ibuf_merges_discard_insert	change_buffer	enabled
ibuf_merges_insert	change_buffer	enabled
ibuf_size	change_buffer	enabled
icp_attempts	icp	disabled
icp_match	icp	disabled
icp_no_match	icp	disabled
icp_out_of_range	icp	disabled
index_page_discards	index	disabled
index_page_merge_attempts	index	disabled
index_page_merge_successful	index	disabled
index_page_reorg_attempts	index	disabled
index_page_reorg_successful	index	disabled
index_page_splits	index	disabled
innodb_activity_count	server	enabled
innodb_background_drop_table_usec	server	disabled
innodb_checkpoint_usec	server	disabled
innodb_dblwr_pages_written	server	enabled
innodb_dblwr_writes	server	enabled
innodb_dict_lru_count	server	disabled
innodb_dict_lru_usec	server	disabled
innodb_ibuf_merge_usec	server	disabled
innodb_log_flush_usec	server	disabled
innodb_master_active_loops	server	disabled
innodb_master_idle_loops	server	disabled
innodb_master_purge_usec	server	disabled
innodb_master_thread_sleeps	server	disabled
innodb_mem_validate_usec	server	disabled
innodb_page_size	server	enabled
innodb_rwlock_sx_os_waits	server	enabled
innodb_rwlock_sx_spin_rounds	server	enabled
innodb_rwlock_sx_spin_waits	server	enabled
innodb_rwlock_s_os_waits	server	enabled
innodb_rwlock_s_spin_rounds	server	enabled
innodb_rwlock_s_spin_waits	server	enabled
innodb_rwlock_x_os_waits	server	enabled
innodb_rwlock_x_spin_rounds	server	enabled
innodb_rwlock_x_spin_waits	server	enabled
lock_deadlocks	lock	enabled
lock_rec_locks	lock	disabled
lock_rec_lock_created	lock	disabled
lock_rec_lock_removed	lock	disabled
lock_rec_lock_requests	lock	disabled
lock_rec_lock_waits	lock	disabled
lock_row_lock_current_waits	lock	enabled
lock_row_lock_time	lock	enabled
lock_row_lock_time_avg	lock	enabled
lock_row_lock_time_max	lock	enabled
lock_row_lock_waits	lock	enabled
lock_table_locks	lock	disabled
lock_table_lock_created	lock	disabled
lock_table_lock_removed	lock	disabled
lock_table_lock_waits	lock	disabled
lock_timeouts	lock	enabled
log_checkpoints	recovery	disabled
log_isn_buf_pool_oldest	recovery	disabled
log_isn_checkpoint_age	recovery	disabled
log_isn_current	recovery	disabled
log_isn_last_checkpoint	recovery	disabled

```

log_lsnn_last_flush      | recovery      | disabled |
log_max_modified_age_async | recovery      | disabled |
log_max_modified_age_sync | recovery      | disabled |
log_num_log_io           | recovery      | disabled |
log_padded                | recovery      | enabled  |
log_pending_checkpoint_writes | recovery      | disabled |
log_pending_log_flushes  | recovery      | disabled |
log_waits                 | recovery      | enabled  |
log_writes                | recovery      | enabled  |
log_write_requests       | recovery      | enabled  |
metadata_table_handles_closed | metadata      | disabled |
metadata_table_handles_opened | metadata      | disabled |
metadata_table_reference_count | metadata      | disabled |
os_data_fsyncs           | os            | enabled  |
os_data_reads            | os            | enabled  |
os_data_writes           | os            | enabled  |
os_log_bytes_written     | os            | enabled  |
os_log_fsyncs            | os            | enabled  |
os_log_pending_fsyncs    | os            | enabled  |
os_log_pending_writes    | os            | enabled  |
os_pending_reads         | os            | disabled |
os_pending_writes        | os            | disabled |
purge_del_mark_records   | purge         | disabled |
purge_dml_delay_usec     | purge         | disabled |
purge_invoked            | purge         | disabled |
purge_resume_count       | purge         | disabled |
purge_stop_count         | purge         | disabled |
purge_undo_log_pages     | purge         | disabled |
purge_upd_exist_or_extern_records | purge         | disabled |
trx_active_transactions  | transaction   | disabled |
trx_commits_insert_update | transaction   | disabled |
trx_nl_ro_commits        | transaction   | disabled |
trx_rollback             | transaction   | disabled |
trx_rollback_savepoint  | transaction   | disabled |
trx_rollback_active      | transaction   | disabled |
trx_ro_commits           | transaction   | disabled |
trx_rseg_current_size    | transaction   | disabled |
trx_rseg_history_len     | transaction   | enabled  |
trx_rw_commits           | transaction   | disabled |
trx_undo_slots_cached    | transaction   | disabled |
trx_undo_slots_used      | transaction   | disabled |
+-----+-----+-----+
235 rows in set (0.01 sec)

```

カウンタモジュール

各カウンタは特定のモジュールに関連付けられています。モジュール名を使用すると、特定のサブシステムのすべてのカウンタを有効化、無効化、またはリセットできます。たとえば、`dml` サブシステムに関連付けられたすべてのカウンタを有効にするには、`module_dml` を使用します。

```

mysql> SET GLOBAL innodb_monitor_enable = module_dml;

mysql> SELECT name, subsystem, status FROM INFORMATION_SCHEMA.INNODB_METRICS
        WHERE subsystem = 'dml';
+-----+-----+-----+
| name      | subsystem | status |
+-----+-----+-----+
| dml_reads | dml       | enabled |
| dml_inserts | dml       | enabled |
| dml_deletes | dml       | enabled |
| dml_updates | dml       | enabled |
+-----+-----+-----+

```

モジュール名は、`innodb_monitor_enable` および関連する変数とともに使用できます。

モジュール名および対応する `SUBSYSTEM` 名を次に示します。

- `module_adaptive_hash` (サブシステム = `adaptive_hash_index`)
- `module_buffer` (サブシステム = `buffer`)
- `module_buffer_page` (subsystem = `buffer_page_io`)

- `module_compress` (サブシステム = `compression`)
- `module_ddl` (サブシステム = `ddl`)
- `module_dml` (サブシステム = `dml`)
- `module_file` (サブシステム = `file_system`)
- `module_ibuf_system` (サブシステム = `change_buffer`)
- `module_icp` (サブシステム = `icp`)
- `module_index` (サブシステム = `index`)
- `module_innodb` (subsystem = `innodb`)
- `module_lock` (サブシステム = `lock`)
- `module_log` (サブシステム = `recovery`)
- `module_metadata` (サブシステム = `metadata`)
- `module_os` (サブシステム = `os`)
- `module_purge` (サブシステム = `purge`)
- `module_trx` (サブシステム = `transaction`)
- `module_undo` (subsystem = `undo`)

例 15.11 INNODB_METRICS テーブルのカウンタの操作

この例では、カウンタの有効化、無効化、およびリセットと、`INNODB_METRICS` テーブル内のカウンタデータのクエリーを示します。

1. 単純な InnoDB テーブルを作成します。

```
mysql> USE test;
Database changed

mysql> CREATE TABLE t1 (c1 INT) ENGINE=INNODB;
Query OK, 0 rows affected (0.02 sec)
```

2. `dml_inserts` カウンタを有効にします。

```
mysql> SET GLOBAL innodb_monitor_enable = dml_inserts;
Query OK, 0 rows affected (0.01 sec)
```

`dml_inserts` カウンタの説明は、`INNODB_METRICS` テーブルの `COMMENT` カラムで見つけることができます。

```
mysql> SELECT NAME, COMMENT FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME="dml_inserts";
+-----+-----+
| NAME   | COMMENT                               |
+-----+-----+
| dml_inserts | Number of rows inserted |
+-----+-----+
```

3. `dml_inserts` カウンタデータを取得するために `INNODB_METRICS` テーブルをクエリーします。DML 操作が実行されていないため、カウンタ値は 0 または NULL です。 `TIME_ENABLED` および `TIME_ELAPSED` の値は、カウンタが最後に有効になった時間と、その時間から経過した秒数を示します。

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME="dml_inserts" \G
***** 1. row *****
      NAME: dml_inserts
     SUBSYSTEM: dml
        COUNT: 0
     MAX_COUNT: 0
     MIN_COUNT: NULL
     AVG_COUNT: 0
```



```

COUNT_RESET: 0
MAX_COUNT_RESET: 0
MIN_COUNT_RESET: NULL
AVG_COUNT_RESET: NULL
TIME_ENABLED: 2014-12-04 14:18:28
TIME_DISABLED: NULL
TIME_ELAPSED: 28
TIME_RESET: NULL
STATUS: enabled
TYPE: status_counter
COMMENT: Number of rows inserted

```

4. テーブルに 3 行のデータを挿入します。

```

mysql> INSERT INTO t1 values(1);
Query OK, 1 row affected (0.00 sec)

```

```

mysql> INSERT INTO t1 values(2);
Query OK, 1 row affected (0.00 sec)

```

```

mysql> INSERT INTO t1 values(3);
Query OK, 1 row affected (0.00 sec)

```

5. `dml_inserts` カウンタデータを取得するために再度 `INNODB_METRICS` テーブルをクエリーします。`COUNT`、`MAX_COUNT`、`AVG_COUNT`、`COUNT_RESET` など、いくつかのカウンタ値が増分されています。これらの値の説明については、`INNODB_METRICS` テーブルの定義を参照してください。

```

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME="dml_inserts"
***** 1. row *****
      NAME: dml_inserts
     SUBSYSTEM: dml
        COUNT: 3
     MAX_COUNT: 3
     MIN_COUNT: NULL
     AVG_COUNT: 0.046153846153846156
     COUNT_RESET: 3
     MAX_COUNT_RESET: 3
     MIN_COUNT_RESET: NULL
     AVG_COUNT_RESET: NULL
     TIME_ENABLED: 2014-12-04 14:18:28
     TIME_DISABLED: NULL
     TIME_ELAPSED: 65
     TIME_RESET: NULL
     STATUS: enabled
     TYPE: status_counter
     COMMENT: Number of rows inserted

```

6. `dml_inserts` カウンタをリセットし、`INNODB_METRICS` テーブルで `dml_inserts` カウンタデータを再度クエリーします。`COUNT_RESET` や `MAX_RESET` などの、前にレポートされた `%_RESET` 値が 0 に戻っています。カウンタが有効になった時点から累積してデータを収集する `COUNT`、`MAX_COUNT`、`AVG_COUNT` などの値はリセットの影響を受けません。

```

mysql> SET GLOBAL innodb_monitor_reset = dml_inserts;
Query OK, 0 rows affected (0.00 sec)

```

```

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME="dml_inserts"
***** 1. row *****
      NAME: dml_inserts
     SUBSYSTEM: dml
        COUNT: 3
     MAX_COUNT: 3
     MIN_COUNT: NULL
     AVG_COUNT: 0.03529411764705882
     COUNT_RESET: 0
     MAX_COUNT_RESET: 0
     MIN_COUNT_RESET: NULL
     AVG_COUNT_RESET: 0
     TIME_ENABLED: 2014-12-04 14:18:28
     TIME_DISABLED: NULL
     TIME_ELAPSED: 85
     TIME_RESET: 2014-12-04 14:19:44
     STATUS: enabled
     TYPE: status_counter

```

```
COMMENT: Number of rows inserted
```

7. すべてのカウンタ値をリセットするには、まずそのカウンタを無効にする必要があります。カウンタを無効にすると、**STATUS** 値が **disabled** に設定されます。

```
mysql> SET GLOBAL innodb_monitor_disable = dml_inserts;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME="dml_inserts"\G
***** 1. row *****
      NAME: dml_inserts
  SUBSYSTEM: dml
      COUNT: 3
   MAX_COUNT: 3
  MIN_COUNT: NULL
   AVG_COUNT: 0.030612244897959183
  COUNT_RESET: 0
  MAX_COUNT_RESET: 0
  MIN_COUNT_RESET: NULL
  AVG_COUNT_RESET: 0
  TIME_ENABLED: 2014-12-04 14:18:28
  TIME_DISABLED: 2014-12-04 14:20:06
  TIME_ELAPSED: 98
  TIME_RESET: NULL
      STATUS: disabled
      TYPE: status_counter
  COMMENT: Number of rows inserted
```

注記

カウンタおよびモジュール名にはワイルドカードマッチングがサポートされています。たとえば、**dml_inserts** カウンタの完全な名前を指定する代わりに、**dml_i%** を指定できます。また、ワイルドカードマッチングを使用して、複数のカウンタまたはモジュールを一度に有効または無効にしたり、リセットしたりすることもできます。たとえば、**dml_** で始まるすべてのカウンタを有効化、無効化またはリセットするには、**dml_%** を指定します。

8. カウンタが無効になったら、**innodb_monitor_reset_all** オプションを使用して、すべてのカウンタ値をリセットできます。すべての値が 0 または NULL に設定されます。

```
mysql> SET GLOBAL innodb_monitor_reset_all = dml_inserts;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME="dml_inserts"\G
***** 1. row *****
      NAME: dml_inserts
  SUBSYSTEM: dml
      COUNT: 0
   MAX_COUNT: NULL
  MIN_COUNT: NULL
   AVG_COUNT: NULL
  COUNT_RESET: 0
  MAX_COUNT_RESET: NULL
  MIN_COUNT_RESET: NULL
  AVG_COUNT_RESET: NULL
  TIME_ENABLED: NULL
  TIME_DISABLED: NULL
  TIME_ELAPSED: NULL
  TIME_RESET: NULL
      STATUS: disabled
      TYPE: status_counter
  COMMENT: Number of rows inserted
```

15.15.7 InnoDB INFORMATION_SCHEMA 一時テーブル情報テーブル

INNODB_TEMP_TABLE_INFO は、InnoDB インスタンスでアクティブなユーザー作成の InnoDB 一時テーブルに関する情報を提供します。オプティマイザで使用する内部 InnoDB 一時テーブルに関する情報は提供されません。

```
mysql> SHOW TABLES FROM INFORMATION_SCHEMA LIKE 'INNODB_TEMP%';
+-----+
```

```
| Tables_in_INFORMATION_SCHEMA (INNODB_TEMP%) |
+-----+
| INNODB_TEMP_TABLE_INFO |
+-----+
```

テーブル定義については、[セクション26.51.28「INFORMATION_SCHEMA INNODB_TEMP_TABLE_INFO テーブル」](#)を参照してください。

例 15.12 INNODB_TEMP_TABLE_INFO

この例では、`INNODB_TEMP_TABLE_INFO` テーブルの特性を示します。

1. 単純な InnoDB 一時テーブルを作成します:

```
mysql> CREATE TEMPORARY TABLE t1 (c1 INT PRIMARY KEY) ENGINE=INNODB;
```

2. `INNODB_TEMP_TABLE_INFO` をクエリーして、一時テーブルのメタデータを表示します。

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TEMP_TABLE_INFO\G
***** 1. row *****
TABLE_ID: 194
NAME: #sql7a79_1_0
N_COLS: 4
SPACE: 182
```

`TABLE_ID` は、一時テーブルの一意の識別子です。 `NAME` カラムには、「#sql」という接頭辞が付いた一時テーブルのシステム生成名が表示されます。 InnoDB では常に 3 つの非表示のテーブルのカラム (`DB_ROW_ID`、`DB_TRX_ID` および `DB_ROLL_PTR`) が作成されるため、カラム数 (`N_COLS`) は 1 ではなく 4 です。

3. MySQL を再起動し、`INNODB_TEMP_TABLE_INFO` をクエリーします。

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TEMP_TABLE_INFO\G
```

サーバーの停止時に `INNODB_TEMP_TABLE_INFO` とそのデータがディスクに永続化されないため、空のセットが返されます。

4. 新しい一時テーブルを作成します。

```
mysql> CREATE TEMPORARY TABLE t1 (c1 INT PRIMARY KEY) ENGINE=INNODB;
```

5. `INNODB_TEMP_TABLE_INFO` をクエリーして、一時テーブルのメタデータを表示します。

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TEMP_TABLE_INFO\G
***** 1. row *****
TABLE_ID: 196
NAME: #sql7b0e_1_0
N_COLS: 4
SPACE: 184
```

`SPACE ID` は、サーバーの起動時に動的に生成されるため、異なる場合があります。

15.15.8 INFORMATION_SCHEMA.FILES からの InnoDB テーブルスペースメタデータの取得

`INFORMATION_SCHEMA.FILES` テーブルは、[file-per-table tablespaces](#)、[general tablespaces](#)、[system tablespace](#)、[temporary table tablespaces](#)、[undo tablespaces](#) (存在する場合) など、すべての InnoDB テーブルスペースタイプに関するメタデータを提供します。

このセクションでは、InnoDB 固有の使用例を示します。 `INFORMATION_SCHEMA.FILES` テーブルで提供されるデータの詳細は、[セクション26.15「INFORMATION_SCHEMA FILES テーブル」](#)を参照してください。

注記

`INNODB_TABLESPACES` テーブルおよび `INNODB_DATAFILES` テーブルでは、InnoDB テーブルスペースに関するメタデータも提供されますが、データは `file-per-table`、`general` および `undo` テーブルスペースに制限されます。

このクエリーは、InnoDB システムテーブルスペースに関するメタデータを、InnoDB テーブルスペースに関連する `INFORMATION_SCHEMA.FILES` テーブルのフィールドから取得します。InnoDB に関連しない `INFORMATION_SCHEMA.FILES` フィールドは、常に `NULL` を返し、クエリーから除外されます。

```
mysql> SELECT FILE_ID, FILE_NAME, FILE_TYPE, TABLESPACE_NAME, FREE_EXTENTS,
  TOTAL_EXTENTS, EXTENT_SIZE, INITIAL_SIZE, MAXIMUM_SIZE, AUTOEXTEND_SIZE, DATA_FREE, STATUS ENGINE
  FROM INFORMATION_SCHEMA.FILES WHERE TABLESPACE_NAME LIKE 'innodb_system' \G
***** 1. row *****
  FILE_ID: 0
  FILE_NAME: ./ibdata1
  FILE_TYPE: TABLESPACE
TABLESPACE_NAME: innodb_system
  FREE_EXTENTS: 0
  TOTAL_EXTENTS: 12
  EXTENT_SIZE: 1048576
  INITIAL_SIZE: 12582912
  MAXIMUM_SIZE: NULL
  AUTOEXTEND_SIZE: 67108864
  DATA_FREE: 4194304
  ENGINE: NORMAL
```

このクエリーは、InnoDB file-per-table および一般テーブルスペースの `FILE_ID` (スペース ID と同等) および `FILE_NAME` (パス情報を含む) を取得します。File-per-table および general テーブルスペースには、`.ibd` ファイル拡張子が付いています。

```
mysql> SELECT FILE_ID, FILE_NAME FROM INFORMATION_SCHEMA.FILES
  WHERE FILE_NAME LIKE '%.ibd%' ORDER BY FILE_ID;
+-----+-----+
| FILE_ID | FILE_NAME |
+-----+-----+
| 2 | ./mysql/plugin.ibd |
| 3 | ./mysql/servers.ibd |
| 4 | ./mysql/help_topic.ibd |
| 5 | ./mysql/help_category.ibd |
| 6 | ./mysql/help_relation.ibd |
| 7 | ./mysql/help_keyword.ibd |
| 8 | ./mysql/time_zone_name.ibd |
| 9 | ./mysql/time_zone.ibd |
| 10 | ./mysql/time_zone_transition.ibd |
| 11 | ./mysql/time_zone_transition_type.ibd |
| 12 | ./mysql/time_zone_leap_second.ibd |
| 13 | ./mysql/innodb_table_stats.ibd |
| 14 | ./mysql/innodb_index_stats.ibd |
| 15 | ./mysql/slave_relay_log_info.ibd |
| 16 | ./mysql/slave_master_info.ibd |
| 17 | ./mysql/slave_worker_info.ibd |
| 18 | ./mysql/gtid_executed.ibd |
| 19 | ./mysql/server_cost.ibd |
| 20 | ./mysql/engine_cost.ibd |
| 21 | ./sys/sys_config.ibd |
| 23 | ./test/t1.ibd |
| 26 | ./home/user/test/test/t2.ibd |
+-----+-----+
```

このクエリーは、InnoDB グローバル一時テーブルスペースの `FILE_ID` および `FILE_NAME` を取得します。グローバル一時テーブルスペースのファイル名には、接頭辞として `ibtmp` が付きます。

```
mysql> SELECT FILE_ID, FILE_NAME FROM INFORMATION_SCHEMA.FILES
  WHERE FILE_NAME LIKE '%ibtmp%';
+-----+-----+
| FILE_ID | FILE_NAME |
+-----+-----+
| 22 | ./ibtmp1 |
+-----+-----+
```

同様に、InnoDB undo テーブルスペースのファイル名には `undo` という接頭辞が付きます。次のクエリーは、InnoDB undo テーブルスペースの `FILE_ID` および `FILE_NAME` を戻します。

```
mysql> SELECT FILE_ID, FILE_NAME FROM INFORMATION_SCHEMA.FILES
  WHERE FILE_NAME LIKE '%undo%';
```

15.16 InnoDB の MySQL パフォーマンススキーマとの統合

このセクションでは、[InnoDB](#) とパフォーマンススキーマの統合について簡単に説明します。包括的なパフォーマンススキーマドキュメントについては、[第27章「MySQL パフォーマンススキーマ」](#)を参照してください。

MySQL [Performance Schema feature](#) を使用して、特定の内部 [InnoDB](#) 操作をプロファイルできます。このタイプのチューニングは、主に最適化戦略を評価してパフォーマンスボトルネックを克服するエキスパートユーザーを対象としています。DBA はまた、この機能を容量計画に使用することにより、標準的なワークロードのときに CPU、RAM、およびディスクストレージの特定の組み合わせでパフォーマンスのボトルネックが発生するかどうかを確認し、発生する場合は、システムの一部の容量を増やすことでパフォーマンスを向上させることができるかどうかを判断することもできます。

この機能を使用して [InnoDB](#) のパフォーマンスを調べるには:

- 通常、[Performance Schema feature](#) の使用方法に精通している必要があります。たとえば、インストゥルメントとコンシューマを有効にする方法、および `performance_schema` テーブルをクエリーしてデータを取得する方法を理解する必要があります。概要については、[セクション27.1「パフォーマンススキーマクイックスタート」](#)を参照してください。
- [InnoDB](#) で使用可能なパフォーマンススキーマインストゥルメントに精通している必要があります。[InnoDB](#) 関連のインストゥルメントを表示するには、`innodb` を含むインストゥルメント名を `setup_instruments` テーブルにクエリーすることができます。

```
mysql> SELECT *
      FROM performance_schema.setup_instruments
      WHERE NAME LIKE '%innodb%';
+-----+-----+-----+
| NAME                                     | ENABLED | TIMED |
+-----+-----+-----+
| wait/synch/mutex/innodb/commit_cond_mutex | NO      | NO    |
| wait/synch/mutex/innodb/innobase_share_mutex | NO      | NO    |
| wait/synch/mutex/innodb/autoinc_mutex      | NO      | NO    |
| wait/synch/mutex/innodb/buf_pool_mutex     | NO      | NO    |
| wait/synch/mutex/innodb/buf_pool_zip_mutex | NO      | NO    |
| wait/synch/mutex/innodb/cache_last_read_mutex | NO      | NO    |
| wait/synch/mutex/innodb/dict_foreign_err_mutex | NO      | NO    |
| wait/synch/mutex/innodb/dict_sys_mutex     | NO      | NO    |
| wait/synch/mutex/innodb/recalc_pool_mutex  | NO      | NO    |
| ...                                         |         |       |
| wait/io/file/innodb/innodb_data_file       | YES     | YES   |
| wait/io/file/innodb/innodb_log_file        | YES     | YES   |
| wait/io/file/innodb/innodb_temp_file       | YES     | YES   |
| stage/innodb/alter table (end)             | YES     | YES   |
| stage/innodb/alter table (flush)           | YES     | YES   |
| stage/innodb/alter table (insert)          | YES     | YES   |
| stage/innodb/alter table (log apply index) | YES     | YES   |
| stage/innodb/alter table (log apply table) | YES     | YES   |
| stage/innodb/alter table (merge sort)      | YES     | YES   |
| stage/innodb/alter table (read PK and internal sort) | YES     | YES   |
| stage/innodb/buffer pool load              | YES     | YES   |
| memory/innodb/buf_buf_pool                 | NO      | NO    |
| memory/innodb/dict_stats_bg_recalc_pool_t  | NO      | NO    |
| memory/innodb/dict_stats_index_map_t       | NO      | NO    |
| memory/innodb/dict_stats_n_diff_on_level   | NO      | NO    |
| memory/innodb/other                         | NO      | NO    |
| memory/innodb/row_log_buf                   | NO      | NO    |
| memory/innodb/row_merge_sort                | NO      | NO    |
| memory/innodb/std                           | NO      | NO    |
| memory/innodb/sync_debug_latches           | NO      | NO    |
| memory/innodb/trx_sys_t::rw_trx_ids        | NO      | NO    |
| ...                                         |         |       |
+-----+-----+-----+
155 rows in set (0.00 sec)
```

インストゥルメントされた [InnoDB](#) オブジェクトに関する追加情報については、インストゥルメントされたオブジェクトに関する追加情報を提供するパフォーマンススキーマ [instances tables](#) をクエリーできます。[InnoDB](#) に関連するインスタンステーブルは次のとおりです:

- [mutex_instances](#) テーブル
- [rlock_instances](#) テーブル

- [cond_instances](#) テーブル
- [file_instances](#) テーブル

注記

InnoDB バッファプールに関連する相互排他ロックおよび RW ロックは、このカバレッジには含まれません。これは、[SHOW ENGINE INNODB MUTEX](#) コマンドの出力にも当てはまります。

たとえば、ファイル I/O インストールメンテーションの実行時にパフォーマンススキーマに表示されるインストールされた InnoDB ファイルオブジェクトに関する情報を表示するには、次のクエリーを発行します:

```
mysql> SELECT *
      FROM performance_schema.file_instances
      WHERE EVENT_NAME LIKE '%innodb%\G
***** 1. row *****
FILE_NAME: /path/to/mysql-8.0/data/ibdata1
EVENT_NAME: wait/io/file/innodb/innodb_data_file
OPEN_COUNT: 3
***** 2. row *****
FILE_NAME: /path/to/mysql-8.0/data/ib_logfile0
EVENT_NAME: wait/io/file/innodb/innodb_log_file
OPEN_COUNT: 2
***** 3. row *****
FILE_NAME: /path/to/mysql-8.0/data/ib_logfile1
EVENT_NAME: wait/io/file/innodb/innodb_log_file
OPEN_COUNT: 2
***** 4. row *****
FILE_NAME: /path/to/mysql-8.0/data/mysql/engine_cost.ibd
EVENT_NAME: wait/io/file/innodb/innodb_data_file
OPEN_COUNT: 3
...
```

- InnoDB イベントデータを格納する [performance_schema](#) テーブルに精通している必要があります。InnoDB 関連のイベントに関連するテーブルは次のとおりです:
 - 待機イベントを格納する [Wait Event](#) テーブル。
 - [Summary](#) テーブル。時間の経過とともに終了したイベントの集計情報を提供します。サマリーテーブルには、I/O 操作に関する情報を集計する [file I/O summary tables](#) が含まれます。
 - [Stage Event](#) テーブル: InnoDB ALTER TABLE およびバッファプールロード操作のイベントデータを格納します。詳細は、[セクション15.16.1「パフォーマンススキーマを使用した InnoDB テーブルの ALTER TABLE の進行状況のモニタリング」](#) および [パフォーマンススキーマを使用したバッファプールのロード進行状況の監視](#) を参照してください。

InnoDB 関連のオブジェクトのみに関心がある場合は、これらのテーブルをクエリーするときに [WHERE EVENT_NAME LIKE '%innodb%'](#) 句または [WHERE NAME LIKE '%innodb%'](#) 句を (必要に応じて) 使用します。

15.16.1 パフォーマンススキーマを使用した InnoDB テーブルの ALTER TABLE の進行状況のモニタリング

[Performance Schema](#) を使用して、InnoDB テーブルの ALTER TABLE 進捗を監視できます。

ALTER TABLE の様々なフェーズを表す 7 つのステージイベントがあります。各ステージイベントでは、ALTER TABLE 操作全体の様々なフェーズの進行に応じて、[WORK_COMPLETED](#) および [WORK_ESTIMATED](#) の累積合計がレポートされます。[WORK_ESTIMATED](#) は、ALTER TABLE が実行するすべての作業を考慮した式を使用して計算され、ALTER TABLE の処理中に改訂できます。[WORK_COMPLETED](#) および [WORK_ESTIMATED](#) の値は、ALTER TABLE によって実行されるすべての作業の抽象表現です。

発生順に、ALTER TABLE ステージイベントには次のものが含まれます:

- [stage/innodb/alter table \(read PK and internal sort\)](#): このステージは、ALTER TABLE が読み取り - 主キーフェーズにある場合にアクティブになります。これは、主キーの推定ページ数に設定された [WORK_COMPLETED=0](#) および

`WORK_ESTIMATED` から始まります。ステージが完了すると、`WORK_ESTIMATED` は主キーの実際のページ数に更新されます。

- `stage/innodb/alter table (merge sort)`: このステージは、ALTER TABLE 操作によって追加されたインデックスごとに繰り返されます。
- `stage/innodb/alter table (insert)`: このステージは、ALTER TABLE 操作によって追加されたインデックスごとに繰り返されます。
- `stage/innodb/alter table (log apply index)`: このステージには、ALTER TABLE の実行中に生成された DML ログの適用が含まれます。
- `stage/innodb/alter table (flush)`: このステージが開始される前に、フラッシュリストの長さに基づいて、より正確な見積りで `WORK_ESTIMATED` が更新されます。
- `stage/innodb/alter table (log apply table)`: このステージには、ALTER TABLE の実行中に生成された同時 DML ログの適用が含まれます。このフェーズの期間は、テーブルの変更の程度によって異なります。テーブルに対して同時 DML が実行されなかった場合、このフェーズは即時です。
- `stage/innodb/alter table (end)`: ALTER TABLE の実行中にテーブルに対して実行された DML の再適用など、フラッシュフェーズ後に表示された残りの作業が含まれます。

注記

InnoDB ALTER TABLE ステージイベントでは、現在空間インデックスの追加は考慮されていません。

パフォーマンススキーマを使用した ALTER TABLE のモニタリングの例

次の例は、`stage/innodb/alter table%` ステージイベントインストゥルメントおよび関連するコンシューマテーブルを有効にして ALTER TABLE の進行状況を監視する方法を示しています。パフォーマンススキーマステージイベントインストゥルメントおよび関連コンシューマについては、[セクション27.12.5「パフォーマンススキーマステージイベントテーブル」](#)を参照してください。

1. `stage/innodb/alter%` インストゥルメントを有効にします:

```
mysql> UPDATE performance_schema.setup_instruments
      SET ENABLED = 'YES'
      WHERE NAME LIKE 'stage/innodb/alter%';
Query OK, 7 rows affected (0.00 sec)
Rows matched: 7  Changed: 7  Warnings: 0
```

2. ステージイベントコンシューマテーブル (`events_stages_current`、`events_stages_history` および `events_stages_history_long` を含む) を有効にします。

```
mysql> UPDATE performance_schema.setup_consumers
      SET ENABLED = 'YES'
      WHERE NAME LIKE '%stages%';
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0
```

3. ALTER TABLE 操作を実行します。この例では、employees サンプルデータベースの employees テーブルに `middle_name` カラムが追加されます。

```
mysql> ALTER TABLE employees.employees ADD COLUMN middle_name varchar(14) AFTER first_name;
Query OK, 0 rows affected (9.27 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

4. パフォーマンススキーマ `events_stages_current` テーブルをクエリーして、ALTER TABLE 操作の進行状況を確認します。表示されるステージイベントは、現在進行中の ALTER TABLE フェーズによって異なります。`WORK_COMPLETED` カラムには、完了した作業が表示されます。`WORK_ESTIMATED` カラムには、残りの作業の見積りが表示されます。

```
mysql> SELECT EVENT_NAME, WORK_COMPLETED, WORK_ESTIMATED
      FROM performance_schema.events_stages_current;
+-----+-----+-----+
| EVENT_NAME                | WORK_COMPLETED | WORK_ESTIMATED |
+-----+-----+-----+
```

```
| stage/innodb/alter table (read PK and internal sort) |      280 |      1245 |
+-----+-----+-----+
1 row in set (0.01 sec)
```

ALTER TABLE 操作が完了すると、`events_stages_current` テーブルは空のセットを返します。この場合、`events_stages_history` テーブルをチェックして、完了した操作のイベントデータを表示できます。例:

```
mysql> SELECT EVENT_NAME, WORK_COMPLETED, WORK_ESTIMATED
FROM performance_schema.events_stages_history;
+-----+-----+-----+
| EVENT_NAME                                | WORK_COMPLETED | WORK_ESTIMATED |
+-----+-----+-----+
| stage/innodb/alter table (read PK and internal sort) |      886 |      1213 |
| stage/innodb/alter table (flush)                |     1213 |     1213 |
| stage/innodb/alter table (log apply table)       |     1597 |     1597 |
| stage/innodb/alter table (end)                  |     1597 |     1597 |
| stage/innodb/alter table (log apply table)       |     1981 |     1981 |
+-----+-----+-----+
5 rows in set (0.00 sec)
```

前述のように、`WORK_ESTIMATED` 値は ALTER TABLE 処理中に改訂されました。初期ステージの完了後の見積作業は 1213 です。ALTER TABLE の処理が完了すると、`WORK_ESTIMATED` は実際の値 (1981) に設定されました。

15.16.2 パフォーマンススキーマを使用した InnoDB Mutex 待機のモニタリング

mutex は、特定の時間に 1 つのスレッドのみが共通リソースにアクセスできるように強制するために、コードで使用される同期メカニズムです。サーバーで実行されている複数のスレッドが同じリソースにアクセスする必要がある場合、スレッドは互いに競合します。mutex のロックを取得する最初のスレッドは、ロックが解放されるまで他のスレッドを待機します。

インストールされた InnoDB mutex の場合、mutex 待機は Performance Schema を使用してモニターできます。「パフォーマンススキーマ」テーブルで収集された待機イベントデータは、たとえば、待機時間が最も多い相互排他ロックや合計待機時間が最も長い相互排他ロックの識別に役立ちます。

次の例は、InnoDB mutex 待機インストールを有効にする方法、関連付けられたコンシューマを有効にする方法、および待機イベントデータをクエリーする方法を示しています。

1. 使用可能な InnoDB mutex 待機インストールを表示するには、パフォーマンススキーマ `setup_instruments` テーブルをクエリーします。すべての InnoDB mutex 待機インストールはデフォルトで無効になっています。

```
mysql> SELECT *
FROM performance_schema.setup_instruments
WHERE NAME LIKE '%wait/synch/mutex/innodb%';
+-----+-----+-----+
| NAME                                | ENABLED | TIMED |
+-----+-----+-----+
| wait/synch/mutex/innodb/commit_cond_mutex | NO      | NO    |
| wait/synch/mutex/innodb/innobase_share_mutex | NO      | NO    |
| wait/synch/mutex/innodb/autoinc_mutex      | NO      | NO    |
| wait/synch/mutex/innodb/autoinc_persisted_mutex | NO      | NO    |
| wait/synch/mutex/innodb/buf_pool_flush_state_mutex | NO      | NO    |
| wait/synch/mutex/innodb/buf_pool_LRU_list_mutex | NO      | NO    |
| wait/synch/mutex/innodb/buf_pool_free_list_mutex | NO      | NO    |
| wait/synch/mutex/innodb/buf_pool_zip_free_mutex | NO      | NO    |
| wait/synch/mutex/innodb/buf_pool_zip_hash_mutex | NO      | NO    |
| wait/synch/mutex/innodb/buf_pool_zip_mutex | NO      | NO    |
| wait/synch/mutex/innodb/cache_last_read_mutex | NO      | NO    |
| wait/synch/mutex/innodb/dict_foreign_err_mutex | NO      | NO    |
| wait/synch/mutex/innodb/dict_persist_dirty_tables_mutex | NO      | NO    |
| wait/synch/mutex/innodb/dict_sys_mutex      | NO      | NO    |
| wait/synch/mutex/innodb/recalc_pool_mutex   | NO      | NO    |
| wait/synch/mutex/innodb/fil_system_mutex    | NO      | NO    |
| wait/synch/mutex/innodb/flush_list_mutex    | NO      | NO    |
| wait/synch/mutex/innodb/fts_bg_threads_mutex | NO      | NO    |
| wait/synch/mutex/innodb/fts_delete_mutex    | NO      | NO    |
| wait/synch/mutex/innodb/fts_optimize_mutex  | NO      | NO    |
| wait/synch/mutex/innodb/fts_doc_id_mutex    | NO      | NO    |
| wait/synch/mutex/innodb/log_flush_order_mutex | NO      | NO    |
```

wait/synch/mutex/innodb/hash_table_mutex	NO	NO	
wait/synch/mutex/innodb/ibuf_bitmap_mutex	NO	NO	
wait/synch/mutex/innodb/ibuf_mutex	NO	NO	
wait/synch/mutex/innodb/ibuf_pessimistic_insert_mutex	NO	NO	
wait/synch/mutex/innodb/log_sys_mutex	NO	NO	
wait/synch/mutex/innodb/log_sys_write_mutex	NO	NO	
wait/synch/mutex/innodb/mutex_list_mutex	NO	NO	
wait/synch/mutex/innodb/page_zip_stat_per_index_mutex	NO	NO	
wait/synch/mutex/innodb/purge_sys_pq_mutex	NO	NO	
wait/synch/mutex/innodb/recv_sys_mutex	NO	NO	
wait/synch/mutex/innodb/recv_writer_mutex	NO	NO	
wait/synch/mutex/innodb/redo_rseg_mutex	NO	NO	
wait/synch/mutex/innodb/noredoreg_mutex	NO	NO	
wait/synch/mutex/innodb/rw_lock_list_mutex	NO	NO	
wait/synch/mutex/innodb/rw_lock_mutex	NO	NO	
wait/synch/mutex/innodb/srv_dict_tmpfile_mutex	NO	NO	
wait/synch/mutex/innodb/srv_innodb_monitor_mutex	NO	NO	
wait/synch/mutex/innodb/srv_misc_tmpfile_mutex	NO	NO	
wait/synch/mutex/innodb/srv_monitor_file_mutex	NO	NO	
wait/synch/mutex/innodb/buf_dblwr_mutex	NO	NO	
wait/synch/mutex/innodb/trx_undo_mutex	NO	NO	
wait/synch/mutex/innodb/trx_pool_mutex	NO	NO	
wait/synch/mutex/innodb/trx_pool_manager_mutex	NO	NO	
wait/synch/mutex/innodb/srv_sys_mutex	NO	NO	
wait/synch/mutex/innodb/lock_mutex	NO	NO	
wait/synch/mutex/innodb/lock_wait_mutex	NO	NO	
wait/synch/mutex/innodb/trx_mutex	NO	NO	
wait/synch/mutex/innodb/srv_threads_mutex	NO	NO	
wait/synch/mutex/innodb/rtr_active_mutex	NO	NO	
wait/synch/mutex/innodb/rtr_match_mutex	NO	NO	
wait/synch/mutex/innodb/rtr_path_mutex	NO	NO	
wait/synch/mutex/innodb/rtr_ssn_mutex	NO	NO	
wait/synch/mutex/innodb/trx_sys_mutex	NO	NO	
wait/synch/mutex/innodb/zip_pad_mutex	NO	NO	
wait/synch/mutex/innodb/master_key_id_mutex	NO	NO	

2. 一部の InnoDB mutex インスタンスはサーバーの起動時に作成され、関連付けられたインストゥルメントがサーバーの起動時にも有効になっている場合のみインストゥルメントされます。すべての InnoDB mutex インスタンスがインストゥルメントされ、有効になっていることを確認するには、次の [performance-schema-instrument](#) ルールを MySQL 構成ファイルに追加します:

```
performance-schema-instrument='wait/synch/mutex/innodb/%=ON'
```

すべての InnoDB mutex の待機イベントデータが不要な場合は、MySQL 構成ファイルに [performance-schema-instrument](#) ルールを追加することで、特定のインストゥルメントを無効にできます。たとえば、全文検索に関連する InnoDB mutex 待機イベントインストゥルメントを無効にするには、次のルールを追加します:

```
performance-schema-instrument='wait/synch/mutex/innodb/fts%=OFF'
```

注記

`wait/synch/mutex/innodb/fts%` などの長い接頭辞を持つルールは、`wait/synch/mutex/innodb/%` などの短い接頭辞を持つルールよりも優先されます。

[performance-schema-instrument](#) ルールを構成ファイルに追加した後、サーバーを再起動します。全文検索に関連するものを除くすべての InnoDB mutex が有効になります。確認するには、[setup_instruments](#) テーブルをクエリーします。有効にしたインストゥルメントの `ENABLED` および `TIMED` カラムを `YES` に設定する必要があります。

```
mysql> SELECT *
FROM performance_schema.setup_instruments
WHERE NAME LIKE '%wait/synch/mutex/innodb%';
+-----+-----+-----+
| NAME                                     | ENABLED | TIMED |
+-----+-----+-----+
| wait/synch/mutex/innodb/commit_cond_mutex | YES     | YES   |
| wait/synch/mutex/innodb/innobase_share_mutex | YES     | YES   |
| wait/synch/mutex/innodb/autoinc_mutex      | YES     | YES   |
| ...                                         | ...     | ...   |
| wait/synch/mutex/innodb/master_key_id_mutex | YES     | YES   |
```

```
+-----+-----+-----+
49 rows in set (0.00 sec)
```

3. `setup_consumers` テーブルを更新して待機イベントコンシューマを有効にします。待機イベントコンシューマはデフォルトで無効になっています。

```
mysql> UPDATE performance_schema.setup_consumers
      SET enabled = 'YES'
      WHERE name like 'events_waits%';
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0
```

待機イベントコンシューマが有効になっていることを確認するには、`setup_consumers` テーブルをクエリーします。`events_waits_current`、`events_waits_history` および `events_waits_history_long` コンシューマを有効にする必要があります。

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+-----+-----+
| NAME                | ENABLED |
+-----+-----+
| events_stages_current | NO      |
| events_stages_history | NO      |
| events_stages_history_long | NO      |
| events_statements_current | YES     |
| events_statements_history | YES     |
| events_statements_history_long | NO      |
| events_transactions_current | YES     |
| events_transactions_history | YES     |
| events_transactions_history_long | NO      |
| events_waits_current    | YES     |
| events_waits_history    | YES     |
| events_waits_history_long | YES     |
| global_instrumentation  | YES     |
| thread_instrumentation  | YES     |
| statements_digest       | YES     |
+-----+-----+
15 rows in set (0.00 sec)
```

4. インストゥルメントおよびコンシューマが有効になったら、監視するワークロードを実行します。この例では、`mysqlslap` ロードエミュレーションクライアントを使用してワークロードをシミュレートします。

```
shell> ./mysqlslap --auto-generate-sql --concurrency=100 --iterations=10
      --number-of-queries=1000 --number-char-cols=6 --number-int-cols=6;
```

5. 待機イベントデータをクエリーします。この例では、`events_waits_current`、`events_waits_history` および `events_waits_history_long` テーブルで見つかったデータを集計する `events_waits_summary_global_by_event_name` テーブルから待機イベントデータをクエリーします。データは、イベントを生成したインストゥルメントの名前であるイベント名 (`EVENT_NAME`) 別に要約されます。要約されたデータには次のものが含まれます:

- `COUNT_STAR`
要約された待機イベントの数。
- `SUM_TIMER_WAIT`
要約された時間指定待機イベントの合計待機時間。
- `MIN_TIMER_WAIT`
要約された時間指定待機イベントの最小待機時間。
- `AVG_TIMER_WAIT`
要約された時間指定待機イベントの平均待機時間。
- `MAX_TIMER_WAIT`
要約された時間指定待機イベントの最大待機時間。

次のクエリーは、インストゥルメント名 (`EVENT_NAME`)、待機イベントの数 (`COUNT_STAR`) およびそのインストゥルメントのイベントの合計待機時間 (`SUM_TIMER_WAIT`) を返します。待機はデフォルトでピコ秒 (1 秒に 1 兆) で時間がかかるため、待機時間は 1000000000 で除算され、待機時間がミリ秒単位で表示されます。データは、集計された待機イベントの数 (`COUNT_STAR`) の降順で表示されます。 `ORDER BY` 句を調整して、合計待機時間でデータを順序付けできます。

```
mysql> SELECT EVENT_NAME, COUNT_STAR, SUM_TIMER_WAIT/1000000000 SUM_TIMER_WAIT_MS
FROM performance_schema.events_waits_summary_global_by_event_name
WHERE SUM_TIMER_WAIT > 0 AND EVENT_NAME LIKE 'wait/synch/mutex/innodb/%'
ORDER BY COUNT_STAR DESC;
```

EVENT_NAME	COUNT_STAR	SUM_TIMER_WAIT_MS
wait/synch/mutex/innodb/trx_mutex	201111	23.4719
wait/synch/mutex/innodb/fil_system_mutex	62244	9.6426
wait/synch/mutex/innodb/redo_rseg_mutex	48238	3.1135
wait/synch/mutex/innodb/log_sys_mutex	46113	2.0434
wait/synch/mutex/innodb/trx_sys_mutex	35134	1068.1588
wait/synch/mutex/innodb/lock_mutex	34872	1039.2589
wait/synch/mutex/innodb/log_sys_write_mutex	17805	1526.0490
wait/synch/mutex/innodb/dict_sys_mutex	14912	1606.7348
wait/synch/mutex/innodb/trx_undo_mutex	10634	1.1424
wait/synch/mutex/innodb/rw_lock_list_mutex	8538	0.1960
wait/synch/mutex/innodb/buf_pool_free_list_mutex	5961	0.6473
wait/synch/mutex/innodb/trx_pool_mutex	4885	8821.7496
wait/synch/mutex/innodb/buf_pool_LRU_list_mutex	4364	0.2077
wait/synch/mutex/innodb/innobase_share_mutex	3212	0.2650
wait/synch/mutex/innodb/flush_list_mutex	3178	0.2349
wait/synch/mutex/innodb/trx_pool_manager_mutex	2495	0.1310
wait/synch/mutex/innodb/buf_pool_flush_state_mutex	1318	0.2161
wait/synch/mutex/innodb/log_flush_order_mutex	1250	0.0893
wait/synch/mutex/innodb/buf_dblwr_mutex	951	0.0918
wait/synch/mutex/innodb/recalc_pool_mutex	670	0.0942
wait/synch/mutex/innodb/dict_persist_dirty_tables_mutex	345	0.0414
wait/synch/mutex/innodb/lock_wait_mutex	303	0.1565
wait/synch/mutex/innodb/autoinc_mutex	196	0.0213
wait/synch/mutex/innodb/autoinc_persisted_mutex	196	0.0175
wait/synch/mutex/innodb/purge_sys_pq_mutex	117	0.0308
wait/synch/mutex/innodb/srv_sys_mutex	94	0.0077
wait/synch/mutex/innodb/ibuf_mutex	22	0.0086
wait/synch/mutex/innodb/recv_sys_mutex	12	0.0008
wait/synch/mutex/innodb/srv_innodb_monitor_mutex	4	0.0009
wait/synch/mutex/innodb/recv_writer_mutex	1	0.0005

注記

前述の結果セットには、起動プロセス中に生成された待機イベントデータが含まれます。このデータを除外するには、起動直後およびワークロードの実行前に `events_waits_summary_global_by_event_name` テーブルを切り捨てることができます。ただし、切捨て操作自体では、少量の待機イベントデータが生成される場合があります。

```
mysql> TRUNCATE performance_schema.events_waits_summary_global_by_event_name;
```

15.17 InnoDB モニター

InnoDB モニターは、InnoDB の内部状態に関する情報を提供します。この情報は、パフォーマンスチューニングに役立ちます。

15.17.1 InnoDB モニターのタイプ

InnoDB モニターには次の 2 つのタイプがあります:

- InnoDB 標準モニターは、次のタイプの情報を表示します。
 - メインバックグラウンドスレッドによって実行される作業

- セマフォ待機
 - 最新の外部キーおよびデッドロックエラーに関するデータ
 - トランザクションのロック待機
 - アクティブなトランザクションによって保持されているテーブルおよびレコードのロック
 - 保留中の I/O 操作および関連する統計
 - 挿入バッファおよび適応ハッシュインデックスの統計
 - redo ログデータ
 - バッファープールの統計
 - 行操作データ
- InnoDB ロックモニターは、標準の InnoDB モニター出力の一部として追加のロック情報を出力します。

15.17.2 InnoDB モニターの有効化

InnoDB モニターで定期出力が有効になっている場合、InnoDB は約 15 秒ごとに `mysqld` サーバーの標準エラー出力 (`stderr`) に出力を書き込みます。

InnoDB は、潜在的なバッファオーバーフローを回避するために、モニター出力を `stdout` または固定サイズのメモリーバッファではなく `stderr` に送信します。

Windows では、特に構成されていないかぎり、`stderr` はデフォルトのログファイルに転送されます。出力をエラーログではなくコンソールウィンドウに送る場合は、コンソールウィンドウで `--console` オプションを使用してコマンドプロンプトからサーバーを起動します。詳細は、[Windows のデフォルトのエラーログの保存先](#)を参照してください。

Unix および Unix に似たシステムでは、特に構成されていないかぎり、`stderr` は通常端末に送信されます。詳細は、[Unix および Unix-Like システムでのデフォルトのエラーログの保存先](#)を参照してください。

InnoDB モニターは、出力生成によってパフォーマンスが低下するため、実際にモニター情報を表示する場合にのみ有効にする必要があります。また、モニター出力がエラーログに送られた場合、後でモニターを無効にしないと、ログが非常に大きくなる可能性があります。

注記

トラブルシューティングを支援するために、InnoDB は、特定の状況で InnoDB 標準モニターの出力を一時的に有効にします。詳細は、[セクション15.21「InnoDB のトラブルシューティング」](#)を参照してください。

InnoDB モニターの出力は、タイムスタンプとモニター名を含むヘッダーで始まります。例:

```
=====
2014-10-16 18:37:29 0x7fc2a95c1700 INNODB MONITOR OUTPUT
=====
```

ロックモニターでは、追加のロック情報が付加された同じ出力が生成されるため、InnoDB 標準モニターのヘッダー (INNODB MONITOR OUTPUT) はロックモニターにも使用されます。

`innodb_status_output` および `innodb_status_output_locks` システム変数は、標準の InnoDB モニターおよび InnoDB ロックモニターを有効にするために使用されます。

InnoDB モニターを有効または無効にするには、`PROCESS` 権限が必要です。

InnoDB 標準モニターの有効化

`innodb_status_output` システム変数を `ON` に設定して、標準の InnoDB モニターを有効にします。

```
SET GLOBAL innodb_status_output=ON;
```


InnoDB 標準モニターを無効にするには、`innodb_status_output` を `OFF` に設定します。

サーバーをシャットダウンすると、`innodb_status_output` 変数がデフォルトの `OFF` 値に設定されます。

InnoDB ロックモニターの有効化

InnoDB ロックモニターのデータは、InnoDB 標準モニターの出力とともに出力されます。InnoDB ロックモニターデータを定期的に印刷するには、InnoDB 標準モニターと InnoDB ロックモニターの両方を有効にする必要があります。

InnoDB ロックモニターを有効にするには、`innodb_status_output_locks` システム変数を `ON` に設定します。InnoDB ロックモニターデータを定期的に印刷するには、InnoDB 標準モニターと InnoDB ロックモニターの両方を有効にする必要があります。

```
SET GLOBAL innodb_status_output=ON;
SET GLOBAL innodb_status_output_locks=ON;
```

InnoDB ロックモニターを無効にするには、`innodb_status_output_locks` を `OFF` に設定します。InnoDB Standard Monitor も無効にするには、`innodb_status_output` を `OFF` に設定します。

サーバーをシャットダウンすると、`innodb_status_output` および `innodb_status_output_locks` 変数がデフォルトの `OFF` 値に設定されます。

注記

`SHOW ENGINE INNODB STATUS` 出力の InnoDB ロックモニターを有効にするには、`innodb_status_output_locks` を有効にする必要があります。

オンデマンドでの InnoDB 標準モニターの出力の取得

InnoDB 標準モニターでの定期的な出力を有効にする代わりに、出力をクライアントプログラムにフェッチする `SHOW ENGINE INNODB STATUS` SQL ステートメントを使用して、オンデマンドで InnoDB 標準モニターの出力を取得できます。`mysql` 対話型クライアントを使用している場合は、通常のセミコロンのステートメントターミネータを `\G` に置き換えると、出力が読み取りやすくなります。

```
mysql> SHOW ENGINE INNODB STATUS\G
```

InnoDB ロックモニターが有効になっている場合、`SHOW ENGINE INNODB STATUS` 出力には InnoDB ロックモニターデータも含まれます。

標準の InnoDB モニター出力のステータスファイルへの送信

起動時に `--innodb-status-file` オプションを指定すると、標準の InnoDB モニター出力を有効にしてステータスファイルに送ることができます。このオプションを使用すると、InnoDB はデータディレクトリに `innodb_status.pid` という名前のファイルを作成し、約 15 秒ごとに出力を書き込みます。

サーバーが正常に停止されると、InnoDB によってステータスファイルが削除されます。異常停止が発生した場合は、ステータスファイルを手動で削除する必要がある場合があります。

`--innodb-status-file` オプションは一時的な使用を目的としています。出力生成はパフォーマンスに影響を与える可能性があり、`innodb_status.pid` ファイルは時間の経過とともに非常に大きくなる可能性があるためです。

15.17.3 InnoDB 標準モニターおよびロックモニターの出力

ロックモニタは、追加のロック情報が含まれている点を除き、標準モニタと同じです。どちらのモニターの定期的な出力を有効にしても、同じ出力ストリームが有効になりますが、ロックモニターが有効になっている場合は、そのストリームに追加の情報が含まれます。たとえば、標準モニタとロックモニタをイネーブルにすると、1つの出力ストリームがオンになります。ロックモニターを無効にするまで、そのストリームには追加のロック情報が含まれます。

`SHOW ENGINE INNODB STATUS` ステートメントを使用して生成される場合、標準モニター出力は 1MB に制限されます。この制限は、サーバー標準エラー出力 (`stderr`) に書き込まれる出力には適用されません。

標準モニターの出力例:

```
mysql> SHOW ENGINE INNODB STATUS\G
***** 1. row *****
```

```
Type: InnoDB
Name:
Status:
=====
2018-04-12 15:14:08 0x7f971c063700 INNODB MONITOR OUTPUT
=====
Per second averages calculated from the last 4 seconds
-----
BACKGROUND THREAD
-----
srv_master_thread loops: 15 srv_active, 0 srv_shutdown, 1122 srv_idle
srv_master_thread log flush and writes: 0
-----
SEMAPHORES
-----
OS WAIT ARRAY INFO: reservation count 24
OS WAIT ARRAY INFO: signal count 24
RW-shared spins 4, rounds 8, OS waits 4
RW-excl spins 2, rounds 60, OS waits 2
RW-sx spins 0, rounds 0, OS waits 0
Spin rounds per wait: 2.00 RW-shared, 30.00 RW-excl, 0.00 RW-sx
-----
LATEST FOREIGN KEY ERROR
-----
2018-04-12 14:57:24 0x7f97a9c91700 Transaction:
TRANSACTION 7717, ACTIVE 0 sec inserting
mysql tables in use 1, locked 1
4 lock struct(s), heap size 1136, 3 row lock(s), undo log entries 3
MySQL thread id 8, OS thread handle 140289365317376, query id 14 localhost root update
INSERT INTO child VALUES (NULL, 1), (NULL, 2), (NULL, 3), (NULL, 4), (NULL, 5), (NULL, 6)
Foreign key constraint fails for table `test`.`child`:
'
CONSTRAINT `child_ibfk_1` FOREIGN KEY (`parent_id`) REFERENCES `parent` (`id`) ON DELETE
CASCADE ON UPDATE CASCADE
Trying to add in child table, in index par_ind tuple:
DATA TUPLE: 2 fields;
0: len 4; hex 80000003; asc ;;
1: len 4; hex 80000003; asc ;;

But in parent table `test`.`parent`, in index PRIMARY,
the closest match we can find is record:
PHYSICAL RECORD: n_fields 3; compact format; info bits 0
0: len 4; hex 80000004; asc ;;
1: len 6; hex 000000001e19; asc ;;
2: len 7; hex 81000001110137; asc 7;;

-----
TRANSACTIONS
-----
Trx id counter 7748
Purge done for trx's n:o < 7747 undo n:o < 0 state: running but idle
History list length 19
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 421764459790000, not started
0 lock struct(s), heap size 1136, 0 row lock(s)
---TRANSACTION 7747, ACTIVE 23 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 1136, 1 row lock(s)
MySQL thread id 9, OS thread handle 140286987249408, query id 51 localhost root updating
DELETE FROM t WHERE i = 1
----- TRX HAS BEEN WAITING 23 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 4 page no 4 n bits 72 index GEN_CLUST_INDEX of table `test`.`t`
trx id 7747 lock_mode X waiting
Record lock, heap no 3 PHYSICAL RECORD: n_fields 4; compact format; info bits 0
0: len 6; hex 000000000202; asc ;;
1: len 6; hex 000000001e41; asc A;;
2: len 7; hex 820000008b0110; asc ;;
3: len 4; hex 80000001; asc ;;

-----
TABLE LOCK table `test`.`t` trx id 7747 lock mode IX
RECORD LOCKS space id 4 page no 4 n bits 72 index GEN_CLUST_INDEX of table `test`.`t`
trx id 7747 lock_mode X waiting
Record lock, heap no 3 PHYSICAL RECORD: n_fields 4; compact format; info bits 0
0: len 6; hex 000000000202; asc ;;
```

```
1: len 6; hex 000000001e41; asc  A;;
2: len 7; hex 820000008b0110; asc  ;;
3: len 4; hex 80000001; asc  ;;

-----
FILE I/O
-----
I/O thread 0 state: waiting for i/o request (insert buffer thread)
I/O thread 1 state: waiting for i/o request (log thread)
I/O thread 2 state: waiting for i/o request (read thread)
I/O thread 3 state: waiting for i/o request (read thread)
I/O thread 4 state: waiting for i/o request (read thread)
I/O thread 5 state: waiting for i/o request (read thread)
I/O thread 6 state: waiting for i/o request (write thread)
I/O thread 7 state: waiting for i/o request (write thread)
I/O thread 8 state: waiting for i/o request (write thread)
I/O thread 9 state: waiting for i/o request (write thread)
Pending normal aio reads: [0, 0, 0, 0] , aio writes: [0, 0, 0, 0] ,
ibuf aio reads:, log i/o's:, sync i/o's:
Pending flushes (fsync) log: 0; buffer pool: 0
833 OS file reads, 605 OS file writes, 208 OS fsyncs
0.00 reads/s, 0 avg bytes/read, 0.00 writes/s, 0.00 fsyncs/s

-----
INSERT BUFFER AND ADAPTIVE HASH INDEX
-----
ibuf: size 1, free list len 0, seg size 2, 0 merges
merged operations:
insert 0, delete mark 0, delete 0
discarded operations:
insert 0, delete mark 0, delete 0
Hash table size 553253, node heap has 0 buffer(s)
Hash table size 553253, node heap has 1 buffer(s)
Hash table size 553253, node heap has 3 buffer(s)
Hash table size 553253, node heap has 0 buffer(s)
Hash table size 553253, node heap has 0 buffer(s)
Hash table size 553253, node heap has 0 buffer(s)
Hash table size 553253, node heap has 0 buffer(s)
Hash table size 553253, node heap has 0 buffer(s)
0.00 hash searches/s, 0.00 non-hash searches/s

---
LOG
---
Log sequence number      19643450
Log buffer assigned up to 19643450
Log buffer completed up to 19643450
Log written up to        19643450
Log flushed up to        19643450
Added dirty pages up to  19643450
Pages flushed up to      19643450
Last checkpoint at       19643450
129 log i/o's done, 0.00 log i/o's/second

-----
BUFFER POOL AND MEMORY
-----
Total large memory allocated 2198863872
Dictionary memory allocated 409606
Buffer pool size 131072
Free buffers 130095
Database pages 973
Old database pages 0
Modified db pages 0
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 0, not young 0
0.00 youngs/s, 0.00 non-youngs/s
Pages read 810, created 163, written 404
0.00 reads/s, 0.00 creates/s, 0.00 writes/s
Buffer pool hit rate 1000 / 1000, young-making rate 0 / 1000 not 0 / 1000
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s
LRU len: 973, unzip_LRU len: 0
I/O sum[0]:cur[0], unzip sum[0]:cur[0]

-----
INDIVIDUAL BUFFER POOL INFO
-----
---BUFFER POOL 0
```

```
Buffer pool size 65536
Free buffers 65043
Database pages 491
Old database pages 0
Modified db pages 0
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 0, not young 0
0.00 young/s, 0.00 non-young/s
Pages read 411, created 80, written 210
0.00 reads/s, 0.00 creates/s, 0.00 writes/s
Buffer pool hit rate 1000 / 1000, young-making rate 0 / 1000 not 0 / 1000
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s
LRU len: 491, unzip_LRU len: 0
I/O sum[0]:cur[0], unzip sum[0]:cur[0]
---BUFFER POOL 1
Buffer pool size 65536
Free buffers 65052
Database pages 482
Old database pages 0
Modified db pages 0
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 0, not young 0
0.00 young/s, 0.00 non-young/s
Pages read 399, created 83, written 194
0.00 reads/s, 0.00 creates/s, 0.00 writes/s
No buffer pool page gets since the last printout
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s
LRU len: 482, unzip_LRU len: 0
I/O sum[0]:cur[0], unzip sum[0]:cur[0]
-----
ROW OPERATIONS
-----
0 queries inside InnoDB, 0 queries in queue
0 read views open inside InnoDB
Process ID=5772, Main thread ID=140286437054208, state=sleeping
Number of rows inserted 57, updated 354, deleted 4, read 4421
0.00 inserts/s, 0.00 updates/s, 0.00 deletes/s, 0.00 reads/s
-----
END OF INNODB MONITOR OUTPUT
=====
```

標準モニター出力セクション

標準モニターによってレポートされる各メトリックの詳細は、「[Oracle Enterprise Manager for MySQL データベース ユーザーガイド](#)」の「[メトリック](#)」の章を参照してください。

- [Status](#)

このセクションは、タイムスタンプ、モニター名、および 1 秒あたりの平均の基になる秒数を示します。この秒数は、現在の時間と InnoDB モニターの出力が最後に出力された時間の間の経過時間です。

- [BACKGROUND THREAD](#)

[srv_master_thread](#) 行は、メインのバックグラウンドスレッドによって実行された作業を示します。

- [SEMAPHORES](#)

このセクションは、セマフォを待機しているスレッド、およびスレッドが相互排他ロックまたは読み書きロックセマフォでスピンまたは待機を必要とした回数に関する統計をレポートします。多数のスレッドがセマフォを待機している場合は、ディスク I/O または InnoDB 内部の競合の問題の結果である可能性があります。競合は、クエリーの高い並列性、またはオペレーティングシステムのスレッドスケジューリングでの問題が原因である場合があります。このような状況では、[innodb_thread_concurrency](#) システム変数をデフォルト値より小さい値に設定すると役立つことがあります。[Spin rounds per wait](#) 行は、相互排他ロックでの OS ウェイトあたりのスピンロックラウンドの数を示します。

相互排他メトリックは、[SHOW ENGINE INNODB MUTEX](#) によってレポートされます。

- [LATEST FOREIGN KEY ERROR](#)

このセクションは、最新の外部キー制約エラーに関する情報を提供します。このようなエラーが発生していない場合は存在しません。その内容には、失敗したステートメントのほか、失敗した制約や、参照されるテーブルと参照するテーブルに関する情報が含まれます。

- [LATEST DETECTED DEADLOCK](#)

このセクションは、最新のデッドロックに関する情報を提供します。デッドロックが発生していない場合は存在しません。その内容には、関連しているトランザクション、各トランザクションが実行しようとしていたステートメント、それぞれが保持しているロックと必要なロック、およびデッドロックを解消するために InnoDB がロールバックすることを決定したトランザクションが示されます。このセクションでレポートされるロックモードについては、[セクション15.7.1「InnoDB ロック」](#)で説明されています。

- [TRANSACTIONS](#)

このセクションでロック待機がレポートされている場合は、アプリケーションでロック競合が発生している可能性があります。この出力はまた、トランザクションデッドロックの原因の追跡にも役立つことがあります。

- [FILE I/O](#)

このセクションは、InnoDB がさまざまなタイプの I/O を実行するために使用するスレッドに関する情報を提供します。このうちの最初の数行は、InnoDB の一般的な処理に専用で使用されます。この内容には、保留中の I/O 操作や I/O パフォーマンスの統計に関する情報も表示されます。

これらのスレッドの数は、`innodb_read_io_threads` および `innodb_write_io_threads` パラメータによって制御されます。[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)を参照してください。

- [INSERT BUFFER AND ADAPTIVE HASH INDEX](#)

このセクションでは、InnoDB 挿入バッファ (`change buffer` と呼ばれる) および適応ハッシュインデックスのステータスを示します。

関連情報については、[セクション15.5.2「変更バッファ」](#)、および[セクション15.5.3「適応型ハッシュインデックス」](#)を参照してください。

- [LOG](#)

このセクションには、InnoDB のログに関する情報が表示されます。その内容には、現在のログシーケンス番号、ログがディスクにフラッシュされた範囲、および InnoDB が最後にチェックポイントを取得した位置が含まれます。([セクション15.11.3「InnoDB チェックポイント」](#)を参照してください。) このセクションには、保留中の書き込みや書き込みパフォーマンスの統計に関する情報も表示されます。

- [BUFFER POOL AND MEMORY](#)

このセクションは、読み取られたページと書き込まれたページに関する統計を提供します。これらの数値から、現在クエリーが実行しているデータファイル I/O 操作の数を計算できます。

バッファープールの統計情報については、[InnoDB 標準モニターを使用したバッファープールのモニタリング](#)を参照してください。バッファープールの操作の詳細は、[セクション15.5.1「バッファプール」](#)を参照してください。

- [ROW OPERATIONS](#)

このセクションは、メインスレッドが実行している内容 (各タイプの行操作の数とパフォーマンスレートを含む) を示します。

15.18 InnoDB のバックアップとリカバリ

このセクションでは、InnoDB のバックアップおよびリカバリに関連するトピックについて説明します。

- InnoDB に適用可能なバックアップ方法の詳細は、[セクション15.18.1「InnoDB バックアップ」](#)を参照してください。
- point-in-time リカバリ、ディスク障害または破損からのリカバリ、および InnoDB によるクラッシュリカバリの実行方法の詳細は、[セクション15.18.2「InnoDB のリカバリ」](#)を参照してください。

15.18.1 InnoDB バックアップ

安全なデータベース管理の鍵は、定期的なバックアップを作成することです。データ量、MySQL サーバーの数およびデータベースワークロードに応じて、これらのバックアップ手法を単独または組み合わせて使用できます: MySQL サーバーの停止中にファイルをコピーして [hot backup with MySQL Enterprise Backup](#); [cold backup](#); [logical backup with mysqldump for small data volumes or record the structure of schema objects](#). ホットバックアップとコールドバックアップは、実際のデータファイルをコピーする [physical backups](#) です。これは、リストアを高速化するために [mysqld](#) サーバーで直接使用できます。

InnoDB データをバックアップするには、MySQL Enterprise Backup を使用することをお勧めします。

注記

InnoDB では、サードパーティのバックアップツールを使用してリストアされるデータベースはサポートされていません。

ホットバックアップ

MySQL Enterprise Backup コンポーネントの一部である [mysqlbackup](#) コマンドを使用すると、実行中の MySQL インスタンス (InnoDB テーブルを含む) を操作の中断を最小限に抑えながらバックアップし、データベースの一貫性のあるスナップショットを生成できます。mysqlbackup が InnoDB テーブルをコピーする場合、InnoDB テーブルに対する読み取りおよび書き込みを続行できます。MySQL Enterprise Backup はまた、圧縮バックアップファイルを作成したり、テーブルやデータベースのサブセットをバックアップしたりすることもできます。ユーザーは、MySQL バイナリログと組み合わせてポイントインタイムリカバリを実行できます。MySQL Enterprise Backup は、MySQL Enterprise サブスクリプションの一部です。詳細は、[セクション30.2「MySQL Enterprise Backup の概要」](#)を参照してください。

コールドバックアップ

MySQL サーバーを停止できる場合は、InnoDB がテーブルの管理に使用するすべてのファイルで構成される物理バックアップを作成できます。次の手順を使用します。

1. MySQL サーバーの [slow shutdown](#) を実行し、エラーなしで停止していることを確認します。
2. すべての InnoDB データファイル (ibdata ファイルおよび .ibd ファイル) を安全な場所にコピーします。
3. すべての InnoDB ログファイル (ib_logfile ファイル) を安全な場所にコピーします。
4. 1 つまたは複数の my.cnf 構成ファイルを安全な場所にコピーします。

mysqldump を使用した論理バックアップ

物理バックアップに加えて、[mysqldump](#) を使用してテーブルをダンプすることで、論理バックアップを定期的を作成することをお勧めします。バイナリファイルは、気付かないうちに破損することがあります。ダンプされたテーブルは人間が読むことのできるテキストファイルに格納されるため、テーブルの破損を見つけることが容易になります。また、形式が単純であるため、重大なデータ破損につながる可能性も少なくなります。mysqldump には、ほかのクライアントをロックすることなく、整合性のあるスナップショットを作成するための [--single-transaction](#) オプションも用意されています。[セクション7.3.1「バックアップポリシーの確立」](#)を参照してください。

レプリケーションは InnoDB テーブルと連携して動作するため、MySQL のレプリケーション機能を使用して、データベースのコピーを高可用性が必要なデータベースサイトに保持できます。[セクション15.19「InnoDB と MySQL レプリケーション」](#)を参照してください。

15.18.2 InnoDB のリカバリ

このセクションでは、InnoDB のリカバリについて説明します。内容は次のとおりです:

- [Point-in-Time リカバリ](#)
- [データ破損またはディスク障害からのリカバリ](#)
- [InnoDB のクラッシュリカバリ](#)
- [クラッシュリカバリ中のテーブルスペースの検出](#)

Point-in-Time リカバリ

物理バックアップが作成された時点から InnoDB データベースをリカバリするには、バックアップを取得する前でも、バイナリロギングを有効にして MySQL サーバーを実行する必要があります。バックアップをリストアしたあとにポイントインタイムリカバリを実現するには、バックアップが作成されたあとに発生した変更をバイナリログから適用できます。 [セクション7.5「Point-in-Time \(増分\) リカバリ」](#) を参照してください。

データ破損またはディスク障害からのリカバリ

データベースが破損するか、またはディスク障害が発生した場合は、バックアップを使用してリカバリを実行する必要があります。破損の場合は、まず、破損していないバックアップを見つけます。ベースバックアップをリストアしたあと、`mysqlbinlog` および `mysql` を使用してバイナリログファイルからポイントインタイムリカバリを実行することにより、バックアップが作成されたあとに発生した変更をリストアします。

データベース破損の場合によっては、破損したテーブルの 1 つまたはいくつかをダンプ、削除および再作成するだけで十分です。 `CHECK TABLE` ステートメントを使用して、テーブルが破損しているかどうかを確認できますが、`CHECK TABLE` では、すべての種類の破損を自然に検出できません。

場合によっては、見た目はデータベースページの破損だが、実際にはオペレーティングシステムによる独自のファイルキャッシュの破損であり、ディスク上のデータは正常であることがあります。最初にコンピュータを再起動することをお勧めします。それにより、データベースページの破損に見えたエラーが解消される可能性があります。 [InnoDB の一貫性の問題](#) が原因で MySQL の起動にまだ問題がある場合は、 [セクション15.21.2「InnoDB のリカバリの強制的な実行」](#) でインスタンスをリカバリモードで起動するステップを参照してください。これにより、データをダンプできます。

InnoDB のクラッシュリカバリ

予期しない MySQL サーバーの終了からリカバリするには、MySQL サーバーを再起動する必要があります。InnoDB はログを自動的にチェックし、データベースの現時点へのロールフォワードを実行します。InnoDB は、クラッシュの時点で存在していたコミットされていないトランザクションを自動的にロールバックします。リカバリ中、`mysqld` には次のような出力が表示されます:

```
InnoDB: The log sequence number 664050266 in the system tablespace does not match
the log sequence number 685111586 in the ib_logfiles!
InnoDB: Database was not shutdown normally!
InnoDB: Starting crash recovery.
InnoDB: Using 'tablespaces.open.2' max LSN: 664075228
InnoDB: Doing recovery: scanned up to log sequence number 690354176
InnoDB: Doing recovery: scanned up to log sequence number 695597056
InnoDB: Doing recovery: scanned up to log sequence number 700839936
InnoDB: Doing recovery: scanned up to log sequence number 706082816
InnoDB: Doing recovery: scanned up to log sequence number 711325696
InnoDB: Doing recovery: scanned up to log sequence number 713458156
InnoDB: Applying a batch of 1467 redo log records ...
InnoDB: 10%
InnoDB: 20%
InnoDB: 30%
InnoDB: 40%
InnoDB: 50%
InnoDB: 60%
InnoDB: 70%
InnoDB: 80%
InnoDB: 90%
InnoDB: 100%
InnoDB: Apply batch completed!
InnoDB: 1 transaction(s) which must be rolled back or cleaned up in total 561887 row
operations to undo
InnoDB: Trx id counter is 4096
...
InnoDB: 8.0.1 started; log sequence number 713458156
InnoDB: Waiting for purge to start
InnoDB: Starting in background the rollback of uncommitted transactions
InnoDB: Rolling back trx with id 3596, 561887 rows to undo
...
./mysqld: ready for connections...
```

InnoDB のクラッシュリカバリは、次のいくつかのステップで構成されます。

- テーブルスペースの検出

テーブルスペース検出は、redo ログアプリケーションを必要とするテーブルスペースを識別するために InnoDB で使用されるプロセスです。クラッシュリカバリ中のテーブルスペースの検出を参照してください。

- Redo log アプリケーション

redo ログアプリケーションは、接続を受け入れる前に初期化中に実行されます。停止またはクラッシュ時にすべての変更が buffer pool から tablespaces (ibdata* および *.ibd ファイル) にフラッシュされた場合、redo ログアプリケーションはスキップされます。起動時に redo ログファイルが欠落している場合、InnoDB は redo ログアプリケーションもスキップします。

- 現在の最大自動増分カウンタ値は、値が変更されるたびに redo ログに書き込まれるため、クラッシュが安全になります。リカバリ中に、InnoDB は redo ログをスキャンしてカウンタ値の変更を収集し、インメモリーテーブルオブジェクトに変更を適用します。

InnoDB による自動増分値の処理方法の詳細は、[セクション15.6.1.6 「InnoDB での AUTO_INCREMENT 処理」](#) および [InnoDB AUTO_INCREMENT カウンタの初期化](#) を参照してください。

- インデックスツリーの破損が発生すると、InnoDB は破損フラグを redo ログに書き込み、破損フラグをクラッシュセーフにします。また、InnoDB は、インメモリー破損フラグデータを各チェックポイントのエンジン専用システムテーブルに書き込みます。リカバリ中、InnoDB は、インメモリーテーブルおよびインデックスオブジェクトを破損としてマークする前に、両方の場所から破損フラグを読み取り、結果をマージします。

- redo ログを削除してリカバリを高速化することは、一部のデータ損失が許容される場合でもお薦めしません。redo ログの削除は、`innodb_fast_shutdown` を 0 または 1 に設定してクリーンシャットダウンした後にのみ考慮する必要があります。

- 不完全な transactions の Roll back

未完了のトランザクションは、予期しない終了時または fast shutdown でアクティブだったトランザクションです。未完了のトランザクションをロールバックするためにかかる時間は、サーバーの負荷に応じて、そのトランザクションが中断される前にアクティブであった期間の 3 または 4 倍になる場合があります。

ロールバックされているトランザクションは取り消せません。極端なケースとして、トランザクションのロールバックに膨大な時間がかかると予測される場合は、`innodb_force_recovery` の設定を 3 以上にして InnoDB を起動した方が速いことがあります。[セクション15.21.2 「InnoDB のリカバリの強制的な実行」](#) を参照してください。

- Change buffer マージ

インデックスページがバッファプールに読み込まれるため、変更バッファ (system tablespace の一部) からセカンダリインデックスのリーフページに変更を適用します。

- Purge

アクティブなトランザクションに表示されなくなった削除マーク付きレコードを削除しています。

Redo ログの適用に続く各ステップは (書き込みのロギングを除き) Redo ログには依存しないため、通常の処理では並列に実行されます。これらのうち、クラッシュリカバリに固有なのは未完了のトランザクションのロールバックだけです。挿入バッファのマージとパージは、通常の処理中に実行されます。

Redo ログの適用のあと、InnoDB は、ダウンタイムを短縮するために接続をできるだけ早く受け入れようとします。クラッシュリカバリの一環として、InnoDB は、サーバーの終了時にコミットされなかったトランザクションまたは XA PREPARE 状態のトランザクションをロールバックします。このロールバックは、新しい接続からのトランザクションと並列に実行されているバックグラウンドスレッドによって実行されます。新しい接続では、ロールバック操作が完了するまで、リカバリされるトランザクションとのロック競合が発生する可能性があります。

ほとんどの状況では、重いアクティビティの途中で MySQL サーバーが予期せず強制終了された場合でも、リカバリプロセスは自動的に実行され、DBA によるアクションは必要ありません。ハードウェア障害や重大なシステムエラーのために InnoDB データが破損した場合は、MySQL が起動を拒否する可能性があります。この場合は、[セクション15.21.2 「InnoDB のリカバリの強制的な実行」](#) を参照してください。

バイナリログおよび InnoDB のクラッシュリカバリについては、[セクション5.4.4 「バイナリログ」](#) を参照してください。

クラッシュリカバリ中のテーブルスペースの検出

リカバリ中に、最後のチェックポイント以降に書き込まれた redo ログが InnoDB で検出された場合は、影響を受けるテーブルスペースに redo ログを適用する必要があります。リカバリ中に影響を受けるテーブルスペースを識別するプロセスは、テーブルスペースの検出と呼ばれます。

テーブルスペースの検出は、起動時にテーブルスペースファイルのスキャンするディレクトリを定義する `innodb_directories` 設定に依存します。 `innodb_data_home_dir`、`innodb_undo_directory` および `datadir` によって定義されたディレクトリは、`innodb_directories` オプションが明示的に構成されているかどうかに関係なく、起動時にスキャンするディレクトリのリストを作成するときに `innodb_directories` 引数値に自動的に追加されます。絶対パスで定義されたテーブルスペースファイル、または `innodb_directories` 設定に自動的に追加されたディレクトリの外部に存在するテーブルスペースファイルは、`innodb_directories` 設定に追加する必要があります。redo ログで参照されているテーブルスペースファイルがまだ検出されていない場合、リカバリは終了します。

15.19 InnoDB と MySQL レプリケーション

レプリカ上のストレージエンジンがソース上のストレージエンジンと同じでない方法でレプリケーションを使用できません。たとえば、ソースの InnoDB テーブルに対する変更をレプリカの MyISAM テーブルにレプリケートできます。詳細は、[セクション17.4.4「異なるソースおよびレプリカのストレージエンジンでのレプリケーションの使用」](#)を参照してください。

レプリカの設定の詳細は、[セクション17.1.2.6「レプリカの設定」](#) および [セクション17.1.2.5「データスナップショットの方法の選択」](#)を参照してください。ソースまたは既存のレプリカを停止せずに新しいレプリカを作成するには、[MySQL Enterprise Backup](#) 製品を使用します。

ソースで失敗したトランザクションはレプリケーションに影響しません。MySQL レプリケーションは、データを変更する SQL ステートメントが MySQL によって書き込まれたバイナリログに基づいています。失敗したトランザクション (外部キー違反やロールバックされたためなど) はバイナリログに書き込まれないため、レプリカには送信されません。[セクション13.3.1「START TRANSACTION、COMMIT および ROLLBACK ステートメント」](#)を参照してください。

レプリケーションと CASCADE. 外部キーリレーションを共有するテーブルがソースとレプリカの両方で InnoDB を使用する場合、ソース上の InnoDB テーブルのカスケードアクションはレプリカのみでレプリケートされます。これは、ステートメントベースのレプリケーションと行ベースのレプリケーションのどちらを使用している場合にも当てはまります。レプリケーションを開始し、次の CREATE TABLE ステートメントを使用して、InnoDB がデフォルトのストレージエンジンとして定義されているソースに 2 つのテーブルを作成するとします:

```
CREATE TABLE fc1 (
  i INT PRIMARY KEY,
  j INT
);

CREATE TABLE fc2 (
  m INT PRIMARY KEY,
  n INT,
  FOREIGN KEY ni (n) REFERENCES fc1 (i)
  ON DELETE CASCADE
);
```

レプリカにデフォルトのストレージエンジンとして定義された MyISAM がある場合、レプリカ上に同じテーブルが作成されますが、それらは MyISAM ストレージエンジンを使用し、FOREIGN KEY オプションは無視されます。次に、ソースのテーブルにいくつかの行を挿入します:

```
source> INSERT INTO fc1 VALUES (1, 1), (2, 2);
Query OK, 2 rows affected (0.09 sec)
Records: 2 Duplicates: 0 Warnings: 0

source> INSERT INTO fc2 VALUES (1, 1), (2, 2), (3, 1);
Query OK, 3 rows affected (0.19 sec)
Records: 3 Duplicates: 0 Warnings: 0
```

この時点で、次に示すように、ソースとレプリカの両方で、テーブル fc1 には 2 行、テーブル fc2 には 3 行が含まれます:

```
source> SELECT * FROM fc1;
+---+-----+
|i|j |
```

```

+---+---+
| 1 | 1 |
| 2 | 2 |
+---+---+
2 rows in set (0.00 sec)

source> SELECT * FROM fc2;
+---+---+
| m | n |
+---+---+
| 1 | 1 |
| 2 | 2 |
| 3 | 1 |
+---+---+
3 rows in set (0.00 sec)

replica> SELECT * FROM fc1;
+---+---+
| i | j |
+---+---+
| 1 | 1 |
| 2 | 2 |
+---+---+
2 rows in set (0.00 sec)

replica> SELECT * FROM fc2;
+---+---+
| m | n |
+---+---+
| 1 | 1 |
| 2 | 2 |
| 3 | 1 |
+---+---+
3 rows in set (0.00 sec)

```

ここで、ソースで次の **DELETE** ステートメントを実行するとします:

```

source> DELETE FROM fc1 WHERE i=1;
Query OK, 1 row affected (0.09 sec)

```

カスケードのため、ソースのテーブル **fc2** には 1 行のみが含まれるようになりました:

```

source> SELECT * FROM fc2;
+---+---+
| m | n |
+---+---+
| 2 | 2 |
+---+---+
1 row in set (0.00 sec)

```

ただし、レプリカでは **DELETE** for **fc1** は **fc2** から行を削除しないため、カスケードはレプリカに伝播されません。**fc2** のレプリカコピーには、最初に挿入されたすべての行が引き続き含まれます:

```

replica> SELECT * FROM fc2;
+---+---+
| m | n |
+---+---+
| 1 | 1 |
| 3 | 1 |
| 2 | 2 |
+---+---+
3 rows in set (0.00 sec)

```

この違いは、カスケード削除が、実際には **InnoDB** ストレージエンジンによって内部的に処理されることから来ています。つまり、どの変更もログに記録されません。

15.20 InnoDB memcached プラグイン

注記

InnoDB memcached プラグインは、MySQL 8.0.22 の時点では非推奨です。将来のバージョンの MySQL ではサポートされなくなる予定です。

InnoDB memcached プラグイン (`daemon_memcached`) には、InnoDB テーブルのデータを自動的に格納および取得し、MySQL サーバーを高速「キー/値ストア」にする統合 memcached デーモンが用意されています。SQL でクエリを作成するかわりに、SQL 解析およびクエリ最適化計画の作成に関連するパフォーマンスオーバーヘッドを回避する単純な `get`、`set` および `incr` 操作を使用できます。また、便宜上、複雑なクエリ、バルク操作および従来のデータベースソフトウェアのその他の長所のために、SQL を使用して同じ InnoDB テーブルにアクセスすることもできます。

この「NoSQL スタイルの」インターフェースは、memcached API を使用してデータベース操作を高速化し、InnoDB がそのバッファプールメカニズムを使用してメモリーキャッシュを処理します。add、set、incr などの memcached 操作によって変更されたデータは、InnoDB テーブルのディスクに格納されます。memcached の簡便性と InnoDB の信頼性および一貫性の組み合わせにより、セクション15.20.1「InnoDB memcached プラグインの利点」で説明されている両方の優れた点がユーザーに提供されます。アーキテクチャの概要は、セクション15.20.2「InnoDB memcached のアーキテクチャー」を参照してください。

15.20.1 InnoDB memcached プラグインの利点

このセクションでは、`daemon_memcached` プラグインの利点について概説します。InnoDB テーブルと memcached を組み合わせると、いずれか一方を単独で使用するよりも利点があります。

- InnoDB ストレージエンジンに直接アクセスすることによって、SQL の構文解析およびプランニングのオーバーヘッドを回避できます。
- memcached を MySQL Server と同じプロセス空間で実行することにより、リクエストを受け渡すことによるネットワークオーバーヘッドが回避されます。
- memcached プロトコルを使用して書き込まれたデータは、MySQL SQL レイヤーを経由せずに InnoDB テーブルに透過的に書き込まれます。クリティカルでないデータの更新時に RAW パフォーマンスを向上させるために、書き込みの頻度を制御できます。
- memcached プロトコルを介してリクエストされたデータは、MySQL SQL レイヤーを経由せずに、InnoDB テーブルから透過的にクエリーされます。
- 同じデータに対する後続のリクエストは InnoDB バッファプールから提供されます。バッファプールはインメモリーキャッシュを処理します。InnoDB 構成オプションを使用して、データ集中型の操作のパフォーマンスをチューニングできます。
- アプリケーションのタイプに応じて、非構造化データまたは構造化データを使用できます。データ用に新しいテーブルを作成するか、既存のテーブルを使用できます。
- InnoDB は複数カラムの値を単一の memcached 項目値に連結したり分解したりできるため、アプリケーションで必要な文字列の構文解釈および連結の量が削減されます。たとえば、文字列値 `2|4|6|8` を memcached キャッシュに格納し、InnoDB でセパレータ文字に基づいて値を分割してから、4 つの数値カラムに結果を格納できます。
- メモリーとディスク間の転送は自動的に処理されるため、アプリケーションロジックが簡素化されます。
- データは MySQL データベースに格納されることで、クラッシュ、機能停止、および破損から保護されます。
- 基礎となる InnoDB テーブルには、レポート、分析、非定型クエリ、バルクロード、マルチステップトランザクション計算、論理和や交差などの集合演算、および SQL の表現と柔軟性に適したその他の操作のために、SQL を介してアクセスできます。
- MySQL レプリケーションと組み合わせてソースサーバーで `daemon_memcached` プラグインを使用することで、高可用性を確保できます。
- memcached と MySQL の統合により、インメモリーデータを永続的にする方法が提供されるため、より重要な種類のデータに使用できます。データが失われる可能性があることを気にすることなく、アプリケーションでより多くの `add`、`incr` および同様の書き込み操作を使用できます。キャッシュされたデータに対する更新を失わずに、memcached サーバーを停止および起動できます。予期しない停止から保護するために、InnoDB のクラッシュリカバリ、レプリケーションおよびバックアップ機能を利用できます。
- InnoDB が高速な主キー検索を実行する方法では、通常 memcached の単一項目クエリに適合します。`daemon_memcached` プラグインで使用される直接の低レベルのデータベースアクセスパスは、同等の SQL クエリよりもキー値ルックアップの方がはるかに効率的です。

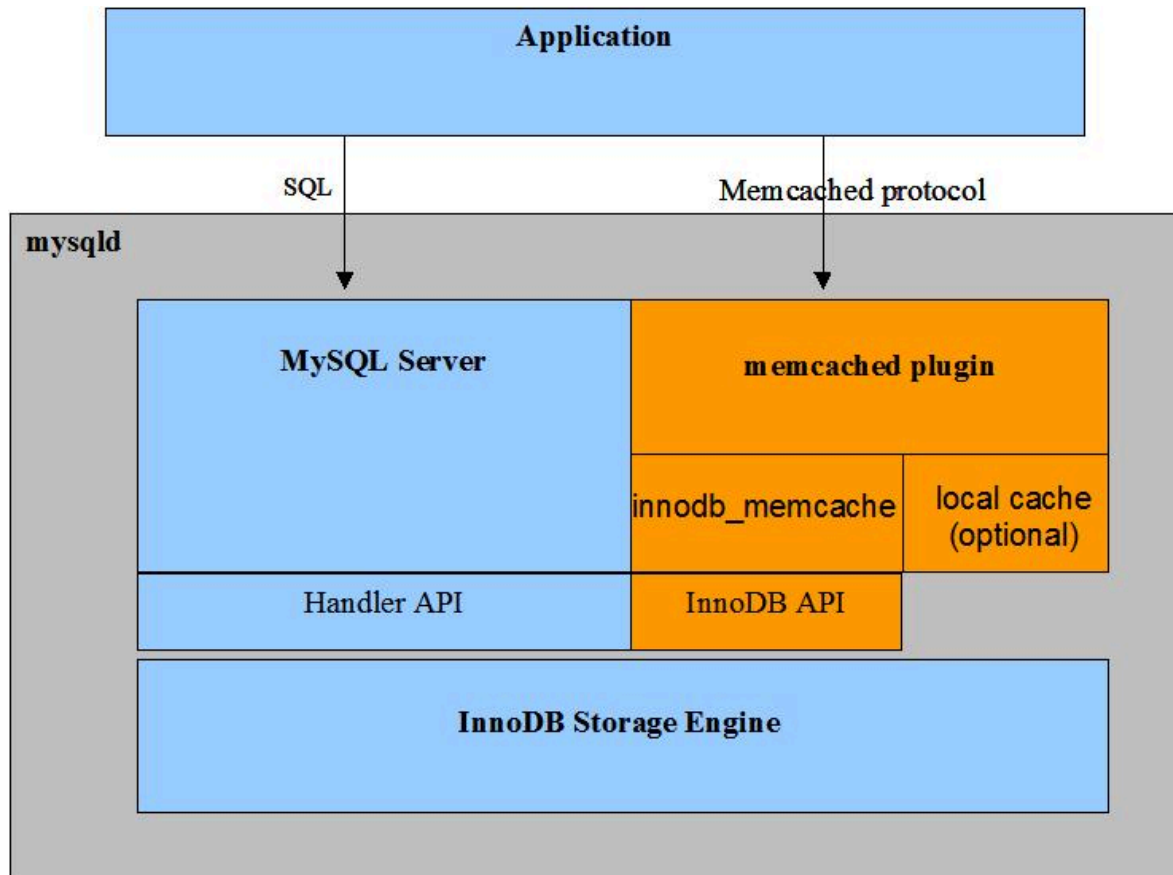
- 複雑なデータ構造、バイナリファイル、またはコードブロックも格納可能な文字列に変換できる **memcached** のシリアライズ機能によって、このようなオブジェクトをデータベースに格納する簡単な方法が提供されます。
- 基礎となるデータには SQL を介してアクセスできるため、レポートの作成、複数のキーにわたる検索または更新、**memcached** データに対する **AVG()** や **MAX()** などの関数のコールが可能です。これらの操作はすべて、**memcached** を単独で使用するとコストがかかり、複雑になります。
- 起動時に **memcached** にデータを手動でロードする必要はありません。特定のキーがアプリケーションによってリクエストされると、値はデータベースから自動的に取得され、**InnoDB buffer pool** を使用してメモリーにキャッシュされます。
- **memcached** は CPU の消費が比較的少なく、メモリーフットプリントを管理しやすいため、同じシステム上で MySQL インスタンスとともに快適に実行できます。
- データ整合性は通常の **InnoDB** テーブルに使用されるメカニズムによって強制されるため、キーが欠落している場合にデータベースをクエリーするために、失効した **memcached** データやフォールバックロジックについて心配する必要はありません。

15.20.2 InnoDB memcached のアーキテクチャー

InnoDB memcached プラグインは、MySQL SQL レイヤーをバイパスして **InnoDB** ストレージエンジンに直接アクセスする MySQL プラグインデーモンとして **memcached** を実装します。

次の図は、アプリケーションが SQL と比較して **daemon_memcached** プラグインを介してデータにアクセスする方法を示しています。

図 15.4 統合 **memcached** Server を使用した MySQL Server



daemon_memcached プラグインの機能:

- `mysqld` のデーモンプラグインとしての `memcached`。 `mysqld` と `memcached` の両方が同じプロセス領域で実行され、データへの待機時間が非常に短くなります。
- SQL パーサー、オプティマイザ、さらにハンドラ API レイヤーもバイパスして、InnoDB テーブルに直接アクセスします。
- テキストベースのプロトコルおよびバイナリプロトコルを含む標準の `memcached` プロトコル。
`daemon_memcached` プラグインは、`memcapable` コマンドの 55 の互換性テストすべてに合格します。
- Multi-column support. ユーザー指定のセパレータ文字で区切られたカラム値を使用して、複数のカラムをキー値ストアの「value」部分にマップできます。
- デフォルトでは、`memcached` プロトコルを使用してデータの読取りおよび InnoDB への書き込みが直接行われるため、MySQL は InnoDB buffer pool を使用してインメモリーキャッシュを管理できます。 デフォルト設定は、データベースアプリケーションの高い信頼性と最も低い驚きの組合せを表します。 たとえば、デフォルト設定では、データベース側のコミットされていないデータや、`memcached get` リクエストに対して返される失効データは回避されます。
- 上級ユーザーは、システムを従来の `memcached` サーバーとして構成し、すべてのデータを `memcached` エンジン (メモリーキャッシング) にのみキャッシュすることも、「`memcached` エンジン」 (メモリーキャッシング) と InnoDB memcached エンジン (InnoDB をバックエンド永続ストレージとして) の組合せを使用することもできます。
- `innodb_api_bk_commit_interval`、`daemon_memcached_r_batch_size` および `daemon_memcached_w_batch_size` 構成オプションを使用して、InnoDB 操作と `memcached` 操作の間でデータがやり取りされる頻度を制御します。 信頼性を最大化するために、バッチサイズオプションのデフォルト値は 1 です。
- `daemon_memcached_option` 構成パラメータを使用して `memcached` オプションを指定する機能。 たとえば、`memcached` がリスニングするポートの変更、同時接続の最大数の削減、キーと値のペアの最大メモリーサイズの変更、またはエラーログのデバッグメッセージの有効化を行うことができます。
- `innodb_api_trx_level` 構成オプションは、`memcached` で処理されるクエリーのトランザクション `isolation level` を制御します。 `memcached` には `transactions` の概念はありませんが、このオプションを使用し、`daemon_memcached` プラグインで使用されるテーブルに対して発行された SQL ステートメントによって発生した変更を `memcached` がすぐに確認する方法を制御できます。 デフォルトでは、`innodb_api_trx_level` は `READ UNCOMMITTED` に設定されます。
- `innodb_api_enable_mdll` オプションを使用すると、MySQL レベルでテーブルをロックできるため、マップされたテーブルは `DDL` で SQL インタフェースを介して削除または変更できません。 ロックがない場合、テーブルは MySQL レイヤーから削除できますが、`memcached` または他のユーザーが使用を停止するまで InnoDB 記憶域に保持されます。「MDL」は「メタデータロック」を表します。

InnoDB memcached と従来の memcached の違い

Using MySQL with memcached で説明されているように、MySQL での `memcached` の使用にすでに慣れている場合があります。 このセクションでは、統合された InnoDB memcached プラグインの機能が従来の `memcached` とどのように異なるかについて説明します。

- インストール: `memcached` ライブラリには MySQL サーバーが付属しているため、インストールと設定が比較的簡単です。 インストールには、`memcached` で使用する `demo_test` テーブルを作成するための `innodb_memcached_config.sql` スクリプトの実行、`daemon_memcached` プラグインを有効にするための `INSTALL PLUGIN` ステートメントの発行、MySQL 構成ファイルまたは起動スクリプトへの必要な `memcached` オプションの追加が含まれます。 `memcp`、`memcat`、`memcapable` などの追加ユーティリティ用の従来の `memcached` ディストリビューションをインストールすることもできます。

従来の `memcached` との比較は、[Installing memcached](#) を参照してください。

- デプロイメント: 従来の `memcached` では、通常、大量の低容量 `memcached` サーバーを実行します。 ただし、`daemon_memcached` プラグインの一般的なデプロイメントでは、MySQL がすでに実行されている中程度または高電力のサーバーの数が少なくなります。 この構成の利点は、未使用のメモリーを利用したり、多数のサーバーにルックアップを分散したりするのではなく、個々のデータベースサーバーの効率を向上させることです。 デフォルト構成では、`memcached` に使用されるメモリーはほとんどなく、メモリー内ルックアップは InnoDB buffer pool から提供されます。 これにより、最近頻繁に使用されたデータが自動的にキャッシュされます。 従来の MySQL

サーバーインスタンスと同様に、`innodb_buffer_pool_size` 構成オプションの値は、メモリー内で可能なかぎり多くの作業が実行されるように、(OS レベルでページングを発生させずに) 実用的な値にしてください。

従来の `memcached` との比較は、[memcached Deployment](#) を参照してください。

- 有効期限: デフォルトでは (`innodb_only` キャッシュポリシーを使用)、InnoDB テーブルの最新データが常に返されるため、有効期限オプションは実用的な効果がありません。キャッシュポリシーを `caching` または `cache_only` に変更すると、有効期限オプションは通常どおりに機能しますが、メモリーキャッシュから期限切れになる前に基礎となるテーブルで更新された場合、リクエストされたデータは失効する可能性があります。

従来の `memcached` との比較は、[Data Expiry](#) を参照してください。

- ネームスペース: `memcached` は、ファイルが競合しないようにするために、接頭辞と接尾辞を含む複雑な名前をファイルに付ける大規模なディレクトリに似ています。`daemon_memcached` プラグインでは、キーに対して同様の命名規則を使用できますが、追加の命名規則もあります。`@@table_id.key` 形式のキー名。`table_id` は、`innodb_memcache.containers` テーブルのマッピングデータを使用して、特定のテーブルを参照するようにデコードされます。`key` は指定されたテーブル内で参照されるが、このテーブルに書き込まれます。

`@@`表記法は、`get`、`add` および `set` 関数への個々のコールに対してのみ機能し、`incr` や `delete` などの他のコールに対しては機能しません。セッション内の後続の `memcached` 操作のデフォルトテーブルを指定するには、`table_id` でキー部分を指定せずに `@@` テーブル記を使用して `get` リクエストを実行します。例:

```
get @@table_id
```

後続の `get`、`set`、`incr`、`delete` およびその他の操作では、`table_id` によって `innodb_memcache.containers.name` カラムに指定されたテーブルが使用されます。

従来の `memcached` との比較は、[Using Namespaces](#) を参照してください。

- ハッシングと分散: `innodb_only` キャッシュポリシーを使用するデフォルト構成は、レプリカサーバーのセットなど、すべてのサーバーですべてのデータを使用できる従来のデプロイメント構成に適しています。

シャード構成と同様にデータを物理的に分割する場合は、`daemon_memcached` プラグインを実行している複数のマシンにデータを分割し、従来の `memcached` ハッシュメカニズムを使用してリクエストを特定のマシンにルーティングできます。通常、MySQL 側では、適切な値が適切なサーバーのデータベースに格納されるように、`add` リクエストによってすべてのデータを `memcached` に挿入できます。

従来の `memcached` との比較は、[memcached Hashing/Distribution Types](#) を参照してください。

- メモリー使用量: デフォルトでは (`innodb_only` キャッシュポリシーを使用)、`memcached` プロトコルは InnoDB テーブルとの間で情報をやり取りし、InnoDB バッファプールは `memcached` メモリー使用量の増加および縮小のかわりにメモリー内ロックアップを処理します。相対的には、`memcached` 側ではメモリーをほとんど使用しません。

キャッシングポリシーを `caching` または `cache_only` に切り替えると、`memcached` メモリー使用量の通常のルールが適用されます。`memcached` データ値のメモリーは、「スラブ」の観点から割り当てられます。`memcached` に使用されるスラブサイズおよび最大メモリーを制御できます。

どちらの方法でも、`telnet` セッションなどを介して標準プロトコルを介してアクセスされる使い慣れた `statistics` システムを使用して、`daemon_memcached` プラグインをモニターおよびトラブルシューティングできます。`daemon_memcached` プラグインには、追加のユーティリティは含まれていません。`memcached-tool script` を使用して、完全な `memcached` ディストリビューションをインストールできます。

従来の `memcached` との比較は、[Memory Allocation within memcached](#) を参照してください。

- スレッド使用率: MySQL スレッドと `memcached` スレッドは同じサーバー上に共存します。オペレーティングシステムによってスレッドに課される制限は、スレッドの合計数に適用されます。

従来の `memcached` との比較は、[memcached Thread Support](#) を参照してください。

- ログの使用状況: `memcached` デーモンは MySQL サーバーとともに実行され、`stderr`、`-v`、`-vv`、`-vvv` の各オプションを使用して、書き込み出力を MySQL `error log` に書き込みます。

従来の `memcached` との比較は、[memcached Logs](#) を参照してください。

- **memcached 操作:** `get`, `set`, `add`, `delete` などの使い慣れた **memcached** 操作を使用できます。シリアライズ (複雑なデータ構造を表す正確な文字列形式) は、言語インタフェースによって異なります。

従来の **memcached** との比較は、[Basic memcached Operations](#) を参照してください。

- **MySQL フロントエンドとしての memcached の使用:** これは、**InnoDB memcached** プラグインの主な目的です。統合 **memcached** デーモンを使用すると、アプリケーションのパフォーマンスが向上し、**InnoDB** でメモリーとディスク間のデータ転送を処理できるため、アプリケーションロジックが簡略化されます。

従来の **memcached** との比較は、[Using memcached as a MySQL Caching Layer](#) を参照してください。

- **ユーティリティ:** MySQL サーバーには **libmemcached** ライブラリが含まれていますが、追加のコマンドラインユーティリティは含まれていません。**memcp**、**memcat**、**memcapable** コマンドなどのコマンドを使用するには、完全な **memcached** ディストリビューションをインストールします。**memrm** および **memflush** がキャッシュからアイテムを削除すると、基礎となる **InnoDB** テーブルからもアイテムが削除されます。

従来の **memcached** との比較は、[libmemcached Command-Line Utilities](#) を参照してください。

- **プログラミングインタフェース:** サポートされているすべての言語を使用して、**daemon_memcached** プラグインを介して MySQL サーバーにアクセスできます: **C and C++**, **Java**, **Perl**, **Python**, **PHP** および **Ruby**。従来の **memcached** サーバーと同様に、サーバーのホスト名とポートを指定します。デフォルトでは、**daemon_memcached** プラグインはポート **11211** でリスニングします。**テキストプロトコルとバイナリプロトコル**の両方を使用できます。**memcached** 関数の **behavior** は、実行時にカスタマイズできます。シリアライズ (複雑なデータ構造を表す正確な文字列形式) は、言語インタフェースによって異なります。

従来の **memcached** との比較は、[Developing a memcached Application](#) を参照してください。

- **よくある質問:** MySQL には、従来の **memcached** に関する広範な FAQ があります。FAQ は、**memcached** データの記憶域メディアとして **InnoDB** テーブルを使用することを除いて、読取り専用キャッシュとしてではなく、以前よりも多くの書込み集中型アプリケーションに **memcached** を使用できることを意味します。

[memcached FAQ](#)を参照してください。

15.20.3 InnoDB memcached プラグインの設定

このセクションでは、MySQL サーバーで **daemon_memcached** プラグインを設定する方法について説明します。ネットワークトラフィックを回避し、待機時間を最小限に抑えるために、**memcached** デーモンは MySQL サーバーと緊密に統合されているため、この機能を使用する各 MySQL インスタンスでこのプロセスを実行します。

注記

daemon_memcached プラグインを設定する前に、[セクション15.20.5「InnoDB memcached プラグインのセキュリティーに関する考慮事項」](#)を参照して、不正アクセスを防ぐために必要なセキュリティー手順を理解してください。

前提条件

- **daemon_memcached** プラグインは、Linux、Solaris および macOS プラットフォームでのみサポートされます。その他のオペレーティングシステムはサポートされません。
- ソースから MySQL を構築する場合は、**-DWITH_INNODB_MEMCACHED=ON** を使用して構築する必要があります。このビルドオプションでは、**daemon_memcached** プラグインの実行に必要な 2 つの共有ライブラリが MySQL プラグインディレクトリ (**plugin_dir**) に生成されます:
 - **libmemcached.so:** MySQL に対する **memcached** デーモンプラグイン。
 - **innodb_engine.so:** **memcached** に対する **InnoDB** API プラグイン。
- **libevent** をインストールする必要があります。
 - ソースから MySQL をビルドしなかった場合、**libevent** ライブラリはインストールに含まれません。オペレーティングシステムのインストール方法を使用して、**libevent** 1.4.12 以上をインストールします。たとえば、オペレーティングシステムに応じて、**apt-get**、**yum** または **port install** を使用できます。たとえば、Ubuntu Linux では、次を使用します:

```
sudo apt-get install libevent-dev
```

- MySQL をソースコードリリースからインストールした場合、[libevent 1.4.12](#) はパッケージにバンドルされ、MySQL ソースコードディレクトリの最上位にあります。バンドルされているバージョンの [libevent](#) を使用する場合は、アクションは必要ありません。ローカルシステムバージョンの [libevent](#) を使用する場合は、`-DWITH_LIBEVENT` ビルドオプションを `system` または `yes` に設定して MySQL をビルドする必要があります。

InnoDB memcached プラグインのインストールおよび構成

- `MYSQL_HOME/share` にある `innodb_memcached_config.sql` 構成スクリプトを実行して、InnoDB テーブルと対話できるように `daemon_memcached` プラグインを構成します。このスクリプトは、`innodb_memcache` データベースに必要な 3 つのテーブル (`cache_policies`、`config_options` および `containers`) とともにインストールします。また、`demo_test` サンプルテーブルが `test` データベースにインストールされます。

```
mysql> source MYSQL_HOME/share/innodb_memcached_config.sql
```

`innodb_memcached_config.sql` スクリプトの実行は単発操作です。後で `daemon_memcached` プラグインをアンインストールして再インストールした場合、テーブルはそのまま残ります。

```
mysql> USE innodb_memcache;
mysql> SHOW TABLES;
+-----+
| Tables_in_innodb_memcache |
+-----+
| cache_policies           |
| config_options          |
| containers               |
+-----+

mysql> USE test;
mysql> SHOW TABLES;
+-----+
| Tables_in_test |
+-----+
| demo_test      |
+-----+
```

これらのテーブルの中で、`innodb_memcache.containers` テーブルが最も重要です。`containers` テーブルのエントリは、InnoDB テーブルのカラムへのマッピングを提供します。`daemon_memcached` プラグインで使用される各 InnoDB テーブルには、`containers` テーブルのエントリが必要です。

`innodb_memcached_config.sql` スクリプトは、`demo_test` テーブルのマッピングを提供する単一のエントリを `containers` テーブルに挿入します。また、単一行のデータを `demo_test` テーブルに挿入します。このデータを使用すると、設定の完了後すぐにインストールを検証できます。

```
mysql> SELECT * FROM innodb_memcache.containers\G
***** 1. row *****
      name: aaa
    db_schema: test
    db_table: demo_test
  key_columns: c1
value_columns: c2
      flags: c3
    cas_column: c4
expire_time_column: c5
unique_idx_name_on_key: PRIMARY

mysql> SELECT * FROM test.demo_test;
+----+-----+-----+-----+-----+
| c1 | c2      | c3 | c4 | c5 |
+----+-----+-----+-----+
| AA | HELLO, HELLO | 8 | 0 | 0 |
+----+-----+-----+-----+
```

`innodb_memcache` テーブルおよび `demo_test` サンプルテーブルの詳細は、[セクション15.20.8「InnoDB memcached プラグインの内部」](#) を参照してください。

- `INSTALL PLUGIN` ステートメントを実行して、`daemon_memcached` プラグインをアクティブにします:

```
mysql> INSTALL PLUGIN daemon_memcached soname "libmemcached.so";
```

プラグインがインストールされると、MySQL サーバーが再起動されるたびに自動的にアクティブ化されます。

InnoDB および memcached 設定の検証

`daemon_memcached` プラグインの設定を確認するには、`telnet` セッションを使用して `memcached` コマンドを発行します。デフォルトでは、`memcached` デーモンはポート 11211 でリスニングします。

1. `test.demo_test` テーブルからデータを取得します。 `demo_test` テーブルの単一行のデータのキー値は `AA` です。

```
telnet localhost 11211
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
get AA
VALUE AA 8 12
HELLO, HELLO
END
```

2. `set` コマンドを使用してデータを挿入します。

```
set BB 10 0 16
GOODBYE, GOODBYE
STORED
```

ここでは:

- `set` は、値を格納するコマンドです
 - `BB` がキーです
 - `10` は操作のフラグです。 `memcached` では無視されますが、任意のタイプの情報を示すためにクライアントで使用できます。未使用の場合は `0` を指定
 - `0` は有効期限 (TTL) です。未使用の場合は `0` を指定
 - `16` は、指定された値ブロックの長さ (バイト) です
 - `GOODBYE, GOODBYE` は、格納される値です
3. MySQL サーバーに接続し、`test.demo_test` テーブルをクエリーして、挿入されたデータが MySQL に格納されていることを確認します。

```
mysql> SELECT * FROM test.demo_test;
+----+-----+-----+-----+
| c1 | c2      | c3 | c4 | c5 |
+----+-----+-----+-----+
| AA | HELLO, HELLO | 8 | 0 | 0 |
| BB | GOODBYE, GOODBYE | 10 | 1 | 0 |
+----+-----+-----+-----+
```

4. `telnet` セッションに戻り、キー `BB` を使用して前に挿入したデータを取得します。

```
get BB
VALUE BB 10 16
GOODBYE, GOODBYE
END
quit
```

統合 `memcached` サーバーも停止する MySQL サーバーを停止した場合、`memcached` データへのアクセスの試行は接続エラーで失敗します。通常、この時点では `memcached` データも表示されなくなるため、`memcached` の再起動時にアプリケーションロジックでデータをメモリーにロードしなおす必要があります。ただし、このプロセスは `InnoDB memcached` プラグインによって自動化されます。

MySQL を再起動すると、`get` 操作によって、以前の `memcached` セッションに格納したキーと値のペアが再度返されます。キーがリクエストされ、関連付けられた値がメモリーキャッシュに存在しない場合、その値は MySQL `test.demo_test` テーブルから自動的に問い合わせられます。

新しいテーブルとカラムのマッピングの作成

この例では、`daemon_memcached` プラグインを使用して独自の InnoDB テーブルを設定する方法を示します。

1. InnoDB テーブルを作成します。テーブルには一意のインデックスを持つキーカラムが必要です。市区町村テーブルのキーカラムは、主キーとして定義されている `city_id` です。テーブルには、`flags`、`cas` および `expiry` 値のカラムも含まれている必要があります。1 つ以上の値カラムがある場合があります。city テーブルには、3 つの値カラム (`name`, `state`, `country`) があります。

注記

有効なマッピングが `innodb_memcache.containers` テーブルに追加されるため、カラム名に関して特別な要件はありません。

```
mysql> CREATE TABLE city (  
  city_id VARCHAR(32),  
  name VARCHAR(1024),  
  state VARCHAR(1024),  
  country VARCHAR(1024),  
  flags INT,  
  cas BIGINT UNSIGNED,  
  expiry INT,  
  primary key(city_id)  
) ENGINE=InnoDB;
```

2. `daemon_memcached` プラグインが InnoDB テーブルへのアクセス方法を認識できるように、`innodb_memcache.containers` テーブルにエントリを追加します。エントリは、`innodb_memcache.containers` テーブル定義を満たす必要があります。各フィールドの説明は、[セクション 15.20.8 「InnoDB memcached プラグインの内部」](#) を参照してください。

```
mysql> DESCRIBE innodb_memcache.containers;  
+-----+-----+-----+-----+-----+-----+  
| Field      | Type      | Null | Key | Default | Extra |  
+-----+-----+-----+-----+-----+-----+  
| name       | varchar(50) | NO   | PRI | NULL    |      |  
| db_schema  | varchar(250) | NO   |     | NULL    |      |  
| db_table   | varchar(250) | NO   |     | NULL    |      |  
| key_columns | varchar(250) | NO   |     | NULL    |      |  
| value_columns | varchar(250) | YES  |     | NULL    |      |  
| flags      | varchar(250) | NO   |     | 0       |      |  
| cas_column | varchar(250) | YES  |     | NULL    |      |  
| expire_time_column | varchar(250) | YES  |     | NULL    |      |  
| unique_idx_name_on_key | varchar(250) | NO   |     | NULL    |      |  
+-----+-----+-----+-----+-----+-----+
```

city テーブルの `innodb_memcache.containers` テーブルエントリは、次のように定義されます：

```
mysql> INSERT INTO `innodb_memcache`.`containers` (  
  `name`, `db_schema`, `db_table`, `key_columns`, `value_columns`,  
  `flags`, `cas_column`, `expire_time_column`, `unique_idx_name_on_key`)  
VALUES ('default', 'test', 'city', 'city_id', 'name|state|country',  
  'flags', 'cas', 'expiry', 'PRIMARY');
```

- `containers.name` カラムに `default` が指定され、`daemon_memcached` プラグインで使用されるデフォルトの InnoDB テーブルとして city テーブルが構成されます。
 - 「|」 デリミタを使用して、複数の InnoDB テーブルのカラム (`name`, `state`, `country`) が `containers.value_columns` にマップされます。
 - `innodb_memcache.containers` テーブルの `flags`、`cas_column` および `expire_time_column` フィールドは、通常、`daemon_memcached` プラグインを使用するアプリケーションでは重要ではありません。ただし、それに指定された InnoDB テーブルのカラムが必要です。データを挿入するときに、これらのカラムが未使用の場合は 0 を指定します。
3. `innodb_memcache.containers` テーブルを更新した後、`daemon_memcache` プラグインを再起動して変更を適用します。

```
mysql> UNINSTALL PLUGIN daemon_memcached;
```



```
mysql> INSTALL PLUGIN daemon_memcached soname "libmemcached.so";
```

4. telnet を使用して、memcached set コマンドを使用して city テーブルにデータを挿入します。

```
telnet localhost 11211
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
set B 0 0 22
BANGALORE|BANGALORE|IN
STORED
```

5. MySQL を使用して、test.city テーブルをクエリーして、挿入したデータが格納されていることを確認します。

```
mysql> SELECT * FROM test.city;
+-----+-----+-----+-----+-----+-----+
| city_id | name   | state   | country | flags | cas | expiry |
+-----+-----+-----+-----+-----+-----+
| B       | BANGALORE | BANGALORE | IN      |      | 0 | 3 | 0 |
+-----+-----+-----+-----+-----+-----+
```

6. MySQL を使用して、test.city テーブルに追加データを挿入します。

```
mysql> INSERT INTO city VALUES ('C','CHENNAI','TAMIL NADU','IN', 0, 0, 0);
mysql> INSERT INTO city VALUES ('D','DELHI','DELHI','IN', 0, 0, 0);
mysql> INSERT INTO city VALUES ('H','HYDERABAD','TELANGANA','IN', 0, 0, 0);
mysql> INSERT INTO city VALUES ('M','MUMBAI','MAHARASHTRA','IN', 0, 0, 0);
```

注記

flags、cas_column および expire_time_column フィールドが使用されていない場合は、これらのフィールドに 0 の値を指定することをお勧めします。

7. telnet を使用して、memcached get コマンドを発行し、MySQL を使用して挿入したデータを取得します。

```
get H
VALUE H 0 22
HYDERABAD|TELANGANA|IN
END
```

InnoDB memcached プラグインの構成

従来の memcached 構成オプションは、daemon_memcached_option 構成パラメータの引数でエンコードされた MySQL 構成ファイルまたは mysqld 起動文字列で指定できます。memcached 構成オプションは、MySQL サーバーが起動されるたびに発生するプラグインがロードされる時に有効になります。

たとえば、memcached がデフォルトポート 11211 ではなくポート 11222 でリスニングするようにするには、daemon_memcached_option 構成オプションの引数として -p11222 を指定します：

```
mysqld .... --daemon_memcached_option="-p11222"
```

その他の memcached オプションは、daemon_memcached_option 文字列でエンコードできます。たとえば、同時接続の最大数の削減、キーと値のペアの最大メモリーサイズの変更、エラーログのデバッグメッセージの有効化などのオプションを指定できます。

daemon_memcached プラグインに固有の構成オプションもあります。これには次のものが含まれます。

- **daemon_memcached_engine_lib_name:** InnoDB memcached プラグインを実装する共有ライブラリを指定します。デフォルト設定は innodb_engine.so です。
- **daemon_memcached_engine_lib_path:** InnoDB memcached プラグインを実装する共有ライブラリを含むディレクトリのパス。デフォルトは NULL で、プラグインディレクトリを表します。
- **daemon_memcached_r_batch_size:** 読取り操作 (get) のバッチコミットサイズを定義します。commit が発生するまでの memcached 読取り操作の数を指定します。daemon_memcached_r_batch_size はデフォルトで 1 に設定されているため、データが memcached を介して更新されたか SQL によって更新されたかに関係なく、すべての get リクエストが InnoDB テーブルで最後にコミットされたデータにアクセスします。値が 1 より大きい場合、読取り操

作のカウンタは `get` コールごとに増分されます。 `flush_all` コールは、読取りカウンタと書き込みカウンタの両方をリセットします。

- `daemon_memcached_w_batch_size`: 書き込み操作 (`set`, `replace`, `append`, `prepend`, `incr`, `decr` など) のバッチコミットサイズを定義します。 `daemon_memcached_w_batch_size` はデフォルトで 1 に設定されているため、停止時にコミットされていないデータは失われず、基礎となるテーブルに対する SQL クエリーが最新のデータにアクセスします。値が 1 より大きい場合、書き込み操作のカウンタは `add`, `set`, `incr`, `decr` および `delete` コールごとに増分されます。 `flush_all` コールは、読取りカウンタと書き込みカウンタの両方をリセットします。

デフォルトでは、 `daemon_memcached_engine_lib_name` または `daemon_memcached_engine_lib_path` を変更する必要はありません。たとえば、 `memcached` に別のストレージエンジン (NDB `memcached` エンジンなど) を使用する場合は、これらのオプションを構成できます。

`daemon_memcached` プラグイン構成パラメータは、MySQL 構成ファイルまたは `mysqld` 起動文字列で指定できます。これらは、 `daemon_memcached` プラグインをロードすると有効になります。

`daemon_memcached` プラグイン構成を変更する場合は、プラグインをリロードして変更を適用します。これを行うには、次のステートメントを発行します:

```
mysql> UNINSTALL PLUGIN daemon_memcached;
mysql> INSTALL PLUGIN daemon_memcached soname "libmemcached.so";
```

構成設定、必要なテーブルおよびデータは、プラグインの再起動時に保持されます。

プラグインの有効化および無効化についてのその他の情報は、 [セクション5.6.1「プラグインのインストールおよびアンインストール」](#) を参照してください。

15.20.4 InnoDB memcached の複数の get および Range クエリーのサポート

`daemon_memcached` プラグインは、複数の `get` 操作 (単一の `memcached` クエリーで複数のキーと値のペアをフェッチ) および範囲クエリーをサポートしています。

複数の get 操作

単一の `memcached` クエリーで複数のキーと値のペアをフェッチする機能により、クライアントとサーバー間の通信トラフィックが削減され、読取りパフォーマンスが向上します。InnoDB の場合、トランザクションおよびオープンテーブル操作が少なくなります。

次の例は、複数データセットのサポートを示しています。この例では、 [新しいテーブルとカラムのマッピングの作成](#) で説明されている `test.city` テーブルを使用します。

```
mysql> USE test;
mysql> SELECT * FROM test.city;
+-----+-----+-----+-----+-----+-----+
| city_id | name   | state   | country | flags | cas | expiry |
+-----+-----+-----+-----+-----+
| B       | BANGALORE | BANGALORE | IN      | 0 | 1 | 0 |
| C       | CHENNAI  | TAMIL NADU | IN      | 0 | 0 | 0 |
| D       | DELHI    | DELHI     | IN      | 0 | 0 | 0 |
| H       | HYDERABAD | TELANGANA | IN      | 0 | 0 | 0 |
| M       | MUMBAI   | MAHARASHTRA | IN      | 0 | 0 | 0 |
+-----+-----+-----+-----+-----+

```

`get` コマンドを実行して、 `city` テーブルからすべての値を取得します。結果はキーと値のペアの順序で返されます。

```
telnet 127.0.0.1 11211
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
get B C D H M
VALUE B 0 22
BANGALORE|BANGALORE|IN
VALUE C 0 21
CHENNAI|TAMIL NADU|IN
VALUE D 0 14
DELHI|DELHI|IN
VALUE H 0 22
HYDERABAD|TELANGANA|IN
```

```
VALUE M 0 21
MUMBAI|MAHARASHTRA|IN
END
```

単一の `get` コマンドで複数の値を取得する場合、(`@@containers.name` テーブル記法を使用して) テーブルを切り替えて最初のキーの値を取得できますが、後続のキーのテーブルを切り替えることはできません。たとえば、この例のテーブルスイッチは有効です:

```
get @@aaa.AA BB
VALUE @@aaa.AA 8 12
HELLO, HELLO
VALUE BB 10 16
GOODBYE, GOODBYE
END
```

同じ `get` コマンドでテーブルの切替えを再試行して別のテーブルからキー値を取得することはサポートされていません。

複数の `get` 操作で取得できるキーの数に制限はありませんが、結果を格納するための 128MB のメモリ制限があります。

範囲クエリー

範囲クエリーの場合、`daemon_memcached` プラグインでは次の比較演算子がサポートされます: `<`, `>`, `<=`, `>=`。演算子の前に `@` 記号を付ける必要があります。範囲クエリーで複数の一致するキーと値のペアが見つかった場合、結果はキーと値のペアの順序で返されます。

次の例は、範囲クエリーのサポートを示しています。この例では、[新しいテーブルとカラムのマッピングの作成](#) で説明されている `test.city` テーブルを使用します。

```
mysql> SELECT * FROM test.city;
+-----+-----+-----+-----+-----+-----+
| city_id | name   | state   | country | flags | cas | expiry |
+-----+-----+-----+-----+-----+-----+
| B       | BANGALORE | BANGALORE | IN      | 0     | 1   | 0       |
| C       | CHENNAI  | TAMIL NADU | IN      | 0     | 0   | 0       |
| D       | DELHI    | DELHI     | IN      | 0     | 0   | 0       |
| H       | HYDERABAD | TELANGANA | IN      | 0     | 0   | 0       |
| M       | MUMBAI   | MAHARASHTRA | IN      | 0     | 0   | 0       |
+-----+-----+-----+-----+-----+-----+
```

telnet セッションを開きます:

```
telnet 127.0.0.1 11211
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^'.
```

B より大きいすべての値を取得するには、`get @>B` を入力します:

```
get @>B
VALUE C 0 21
CHENNAI|TAMIL NADU|IN
VALUE D 0 14
DELHI|DELHI|IN
VALUE H 0 22
HYDERABAD|TELANGANA|IN
VALUE M 0 21
MUMBAI|MAHARASHTRA|IN
END
```

M より小さいすべての値を取得するには、`get @<M` を入力します:

```
get @<M
VALUE B 0 22
BANGALORE|BANGALORE|IN
VALUE C 0 21
CHENNAI|TAMIL NADU|IN
VALUE D 0 14
DELHI|DELHI|IN
VALUE H 0 22
HYDERABAD|TELANGANA|IN
```

```
END
```

M 以下のすべての値を取得するには、`get @<=M` を入力します:

```
get @<=M
VALUE B 0 22
BANGALORE|BANGALORE|IN
VALUE C 0 21
CHENNAI|TAMIL NADU|IN
VALUE D 0 14
DELHI|DELHI|IN
VALUE H 0 22
HYDERABAD|TELANGANA|IN
VALUE M 0 21
MUMBAI|MAHARASHTRA|IN
```

B より大きく、M より小さい値を取得するには、`get @>B@<M` を入力します:

```
get @>B@<M
VALUE C 0 21
CHENNAI|TAMIL NADU|IN
VALUE D 0 14
DELHI|DELHI|IN
VALUE H 0 22
HYDERABAD|TELANGANA|IN
END
```

最大 2 つの比較演算子を解析できます。1 つは次より小さい (`@<`) 演算子または次以下 (`@<=`) 演算子で、もう 1 つは次より大きい (`@>`) 演算子または次以上 (`@>=`) 演算子です。追加の演算子はキーの一部とみなされます。たとえば、3 つの演算子を使用して `get` コマンドを発行すると、3 つ目の演算子 (`@>C`) はキーの一部として扱われ、`get` コマンドは M より小さく B@>C より大きい値を検索します。

```
get @<M@>B@>C
VALUE C 0 21
CHENNAI|TAMIL NADU|IN
VALUE D 0 14
DELHI|DELHI|IN
VALUE H 0 22
HYDERABAD|TELANGANA|IN
```

15.20.5 InnoDB memcached プラグインのセキュリティーに関する考慮事項

注意

本番サーバー、または MySQL インスタンスに機密データが含まれている場合はテストサーバーに `daemon_memcached` プラグインをデプロイする前に、このセクションを参照してください。

`memcached` ではデフォルトで認証メカニズムが使用されず、オプションの SASL 認証は従来の DBMS セキュリティー対策ほど強力ではないため、`daemon_memcached` プラグインを使用する MySQL インスタンス内の機密性のないデータのみを保持し、潜在的な侵入者からこの構成を使用するすべてのサーバーからウォールオフにします。インターネットからこれらのサーバーへの `memcached` アクセスを許可しないでください。ファイアウォール付きイントラネット内 (メンバーシップを制限できるサブネットからのみ) からのアクセスを許可してください。

SASL を使用した memcached のパスワード保護

SASL サポートは、`memcached` クライアントを介した認証されていないアクセスから MySQL データベースを保護する機能を提供します。このセクションでは、`daemon_memcached` プラグインを使用して SASL を有効にする方法について説明します。これらのステップは、従来の `memcached` サーバーで SASL を有効にするために実行されるステップとほぼ同じです。

SASL は、接続ベースのプロトコルに認証サポートを追加するための標準である「Simple Authentication and Security レイヤー」を表します。`memcached` は、バージョン 1.4.3 で SASL サポートを追加しました。

SASL 認証はバイナリプロトコルでのみサポートされます。

`memcached` クライアントは、`innodb_memcache.containers` テーブルに登録されている InnoDB テーブルにのみアクセスできます。DBA はこのようなテーブルにアクセス制限を設定できますが、`memcached` アプリケーションを介

したアクセスは制御できません。このため、SASL サポートは、`daemon_memcached` プラグインに関連付けられた InnoDB テーブルへのアクセスを制御するために提供されます。

次のセクションでは、SASL 対応 `daemon_memcached` プラグインを構築、有効化、およびテストする方法を示します。

InnoDB memcached プラグインを使用した SASL の構築と有効化

SASL 対応の `daemon_memcached` プラグインでは SASL ライブラリを使用して `memcached` を構築する必要があるため、デフォルトでは、SASL 対応の `daemon_memcached` プラグインは MySQL リリースパッケージに含まれていません。SASL サポートを有効にするには、MySQL ソースをダウンロードし、SASL ライブラリのダウンロード後に `daemon_memcached` プラグインを再構築します:

1. SASL 開発およびユーティリティライブラリをインストールします。たとえば、Ubuntu では、`apt-get` を使用してライブラリを取得します:

```
sudo apt-get -f install libsasl2-2 sasl2-bin libsasl2-2 libsasl2-dev libsasl2-modules
```

2. `cmake` オプションに `ENABLE_MEMCACHED_SASL=1` を追加して、SASL 機能を持つ `daemon_memcached` プラグイン共有ライブラリを構築します。`memcached` には、テストを容易にする簡易クリアテキストパスワードのサポートも用意されています。簡易クリアテキストパスワードサポートを有効にするには、`ENABLE_MEMCACHED_SASL_PWDB=1` `cmake` オプションを指定します。

要約すると、次の 3 つの `cmake` オプションを追加します:

```
cmake ... -DWITH_INNOODB_MEMCACHED=1 -DENABLE_MEMCACHED_SASL=1 -DENABLE_MEMCACHED_SASL_PWDB=1
```

3. セクション 15.20.3 「InnoDB memcached プラグインの設定」の説明に従って、`daemon_memcached` プラグインをインストールします。

4. ユーザー名とパスワードファイルを構成します。(この例では、`memcached` の簡易クリアテキストパスワードサポートを使用します。)

- a. ファイルで、`testname` という名前のユーザーを作成し、パスワードを `testpasswd` として定義します:

```
echo "testname:testpasswd:::::::" >/home/jy/memcached-sasl-db
```

- b. `memcached` にユーザー名とパスワードファイルを通知するように、`MEMCACHED_SASL_PWDB` 環境変数を構成します:

```
export MEMCACHED_SASL_PWDB=/home/jy/memcached-sasl-db
```

- c. クリアテキストパスワードが使用されていることを `memcached` に通知します:

```
echo "mech_list: plain" > /home/jy/work2/msasl/clients/memcached.conf
export SASL_CONF_PATH=/home/jy/work2/msasl/clients
```

5. `daemon_memcached_option` 構成パラメータでエンコードされた `memcached -S` オプションを使用して MySQL サーバーを再起動し、SASL を有効にします:

```
mysqld ... --daemon_memcached_option="-S"
```

6. 設定をテストするには、「SASL 対応 libmemcached」などの SASL 対応クライアントを使用します。

```
memcp --servers=localhost:11211 --binary --username=testname
--password=password myfile.txt
```

```
memcat --servers=localhost:11211 --binary --username=testname
--password=password myfile.txt
```

間違ったユーザー名またはパスワードを指定すると、操作は拒否され、`memcache error AUTHENTICATION FAILURE` メッセージが表示されます。この場合、`memcached-sasl-db` ファイルに設定されているクリアテキストパスワードを調べて、指定した資格証明が正しいことを確認します。

memcached を使用して SASL 認証をテストする方法は他にもありますが、前述の方法は最も簡単です。

15.20.6 InnoDB memcached プラグイン用のアプリケーションの記述

通常、InnoDB memcached プラグイン用のアプリケーションを記述するには、MySQL または memcached API を使用する既存のコードをある程度書き換えるか適応させます。

- `daemon_memcached` プラグインを使用すると、多くの従来の memcached サーバーが低消費マシン上で実行されるのではなく、MySQL サーバーと同じ数の memcached サーバーが、大規模なディスク記憶域およびメモリーを備えた比較的高消費マシン上で実行されます。memcached API で動作する既存のコードを再利用できますが、サーバー構成が異なるために適応が必要になる可能性があります。
- `daemon_memcached` プラグインを介して格納されたデータは、`VARCHAR`、`TEXT` または `BLOB` カラムに格納され、数値操作を実行するために変換する必要があります。変換は、アプリケーション側で実行することも、クエリーで `CAST()` 関数を使用して実行することもできます。
- データベースバックグラウンドから使用する場合、多くのカラムを備えた汎用の SQL テーブルを使用する場合があります。memcached コードによってアクセスされるテーブルには、データ値を保持する単一または少数のカラムしかない可能性があります。
- 単一行のクエリー、挿入、更新または削除を実行するアプリケーションの一部を調整して、コードの重要なセクションのパフォーマンスを向上させることができます。InnoDB memcached インタフェースを介して実行すると、`queries` (読取り) 操作と `DML` (書込み) 操作の両方が大幅に高速になります。書込みのパフォーマンスの向上は通常、読取りのパフォーマンスの向上よりも大きいため、ロギングを実行するコードの適応や、web サイトでの対話型の選択の記録に焦点を当てる場合があります。

次の各セクションでは、これらの点について詳しく説明します。

15.20.6.1 InnoDB memcached プラグイン用の既存の MySQL スキーマの適応

既存の MySQL スキーマまたはアプリケーションを `daemon_memcached` プラグインを使用するように適応させる場合は、memcached アプリケーションの次の側面を考慮してください:

- memcached キーに空白や改行を含めることはできません。これらの文字は ASCII プロトコルでセパレーターとして使用されるためです。スペースを含む検索値を使用する場合は、`add()`、`set()`、`get()`、などの呼び出しでこれらをキーとして使用する前に、スペースのない値に変換またはハッシュします。理論上、これらの文字はバイナリプロトコルを使用するプログラム内の鍵で使用できますが、広範囲のクライアントとの互換性を確保するために、鍵で使用される文字を制限するようにしてください。
- InnoDB テーブルに短い数値の `primary key` カラムがある場合は、整数を文字列値に変換して、memcached の一意の参照キーとして使用します。memcached サーバーが複数のアプリケーションに使用されている場合、または複数の InnoDB テーブルで使用されている場合は、一意になるように名前を変更することを検討してください。たとえば、数値の前にテーブル名またはデータベース名とテーブル名を付加します。

注記

`daemon_memcached` プラグインは、`INTEGER` が主キーとして定義されているマップ済 InnoDB テーブルでの挿入および読取りをサポートしています。

- memcached を使用してクエリーまたは格納されたデータには、パーティションテーブルを使用できません。
- memcached プロトコルは数値を文字列として渡します。`SUM()` や `AVG()` などの SQL 関数で使えるカウンタを実装するために、基礎となる InnoDB テーブルに数値を格納するには、次のようにします:
 - 予想される最大数のすべての桁 (さらに該当する場合はマイナス符号、小数点またはその両方に対する追加の文字) を保持するために十分な文字がある `VARCHAR` カラムを使用します。
 - カラム値を使用して算術を実行するクエリーでは、`CAST()` 関数を使用して、値を文字列から整数または他の数値型に変換します。例:

```
# Alphanumeric entries are returned as zero.
SELECT CAST(c2 as unsigned integer) FROM demo_test;
```



```
# Since there could be numeric values of 0, can't disqualify them.
# Test the string values to find the ones that are integers, and average only those.

SELECT AVG(cast(c2 as unsigned integer)) FROM demo_test
WHERE c2 BETWEEN '0' and '9999999999';

# Views let you hide the complexity of queries. The results are already converted;
# no need to repeat conversion functions and WHERE clauses each time.

CREATE VIEW numbers AS SELECT c1 KEY, CAST(c2 AS UNSIGNED INTEGER) val
FROM demo_test WHERE c2 BETWEEN '0' and '9999999999';
SELECT SUM(val) FROM numbers;
```

注記

結果セット内のアルファベット値は、`CAST()` のコールによって 0 に変換されます。結果セット内の行数に依存する `AVG()` などの関数を使用する場合は、数値以外の値を除外するための `WHERE` 句を含めます。

- キーとして使用される InnoDB カラムの値が 250 バイトを超える可能性がある場合は、250 バイト未満にハッシュします。
- `daemon_memcached` プラグインで既存のテーブルを使用するには、`innodb_memcache.containers` テーブルにそのテーブルのエントリを定義します。このテーブルをすべての `memcached` リクエストのデフォルトにするには、`name` カラムに `default` の値を指定し、MySQL サーバーを再起動して変更を有効にします。異なるクラスの `memcached` データに複数のテーブルを使用する場合は、選択した `name` 値を使用して `innodb_memcache.containers` テーブルに複数のエントリを設定し、アプリケーション内で `get @@name` または `set @@name` の形式で `memcached` リクエストを発行して、後続の `memcached` リクエストに使用するテーブルを指定します。

事前定義された `test.demo_test` テーブル以外のテーブルを使用する例については、例 15.13 「InnoDB memcached アプリケーションでの独自のテーブルの使用」を参照してください。必要なテーブルレイアウトについては、セクション 15.20.8 「InnoDB memcached プラグインの内部」を参照してください。

- `memcached` キーと値のペアで複数の InnoDB テーブルのカラム値を使用するには、InnoDB テーブルの `innodb_memcache.containers` エントリの `value_columns` フィールドに、カンマ、セミコロン、空白またはパイプ文字で区切られたカラム名を指定します。たとえば、`value_columns` フィールドに `col1,col2,col3` または `col1|col2|col3` を指定します。

`memcached add` または `set` コールに文字列を渡す前に、パイプ文字をセパレータとして使用して、カラム値を単一の文字列に連結します。文字列は、正しいカラムに自動的に解凍されます。各 `get` コールは、パイプ文字で区切られたカラム値を含む単一の文字列を戻します。適切なアプリケーション言語構文を使用して、値を解凍できます。

例 15.13 InnoDB memcached アプリケーションでの独自のテーブルの使用

この例では、データ操作に `memcached` を使用するサンプル Python アプリケーションで独自のテーブルを使用する方法を示します。

この例では、セクション 15.20.3 「InnoDB memcached プラグインの設定」の説明に従って `daemon_memcached` プラグインがインストールされていることを前提としています。また、`python-memcache` モジュールを使用する Python スクリプトを実行するようにシステムが構成されていることも前提としています。

1. 母集団、地域およびドライバ側のデータを含む国情報を格納する `multicol` テーブルを作成します (右側の場合は 'R'、左側の場合は 'L')。

```
mysql> USE test;

mysql> CREATE TABLE `multicol` (
  `country` varchar(128) NOT NULL DEFAULT "",
  `population` varchar(10) DEFAULT NULL,
  `area_sq_km` varchar(9) DEFAULT NULL,
  `drive_side` varchar(1) DEFAULT NULL,
  `c3` int(11) DEFAULT NULL,
  `c4` bigint(20) unsigned DEFAULT NULL,
  `c5` int(11) DEFAULT NULL,
  PRIMARY KEY (`country`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

2. `daemon_memcached` プラグインが `multicol` テーブルにアクセスできるように、`innodb_memcache.containers` テーブルにレコードを挿入します。

```
mysql> INSERT INTO innodb_memcache.containers
  (name,db_schema,db_table,key_columns,value_columns,flags,cas_column,
  expire_time_column,unique_idx_name_on_key)
VALUES
('bbb','test','multicol','country','population,area_sq_km,drive_side',
'c3','c4','c5','PRIMARY');

mysql> COMMIT;
```

- `multicol` テーブルの `innodb_memcache.containers` レコードは、'bbb' の `name` 値 (テーブル識別子) を指定します。

注記

すべての `memcached` アプリケーションに単一の `InnoDB` テーブルが使用されている場合、`@@` テーブル記法を使用してテーブルを切り替えるのを避けるために、`name` 値を `default` に設定できます。

- `db_schema` カラムは、`multicol` テーブルが存在するデータベースの名前である `test` に設定されます。
 - `db_table` カラムは、`InnoDB` テーブルの名前である `multicol` に設定されます。
 - `key_columns` は、一意の `country` カラムに設定されます。 `country` カラムは、`multicol` テーブル定義で主キーとして定義されます。
 - 複合データ値を保持する単一の `InnoDB` テーブルのカラムではなく、データは 3 つのテーブルのカラム (`population`、`area_sq_km` および `drive_side`) に分割されます。 複数の値カラムに対応するには、`value_columns` フィールドにカンマ区切りのカラムリストを指定します。 `value_columns` フィールドに定義されているカラムは、値の格納または取得時に使用されるカラムです。
 - `flags`、`expire_time` および `cas_column` の各フィールドの値は、`demo.test` サンプルテーブルで使用されている値に基づきます。 これらのフィールドは通常、`daemon_memcached` プラグインを使用するアプリケーションでは重要ではありません。これは、MySQL がデータの同期を維持し、データの期限切れや失効を心配する必要がないためです。
 - `unique_idx_name_on_key` フィールドが `PRIMARY` に設定され、`multicol` テーブルの一意の `country` カラムに定義されているプライマリインデックスを参照します。
3. サンプル Python アプリケーションをファイルにコピーします。 この例では、サンプルスクリプトが `multicol.py` という名前のファイルにコピーされます。

サンプル Python アプリケーションは、`daemon_memcached` プラグインを介して `InnoDB` テーブルにアクセスする方法を示す、`multicol` テーブルにデータを挿入し、すべてのキーのデータを取得します。

```
import sys, os
import memcache

def connect_to_memcached():
    memc = memcache.Client(['127.0.0.1:11211'], debug=0);
    print "Connected to memcached."
    return memc

def banner(message):
    print
    print "=" * len(message)
    print message
    print "=" * len(message)

country_data = [
("Canada","34820000","9984670","R"),
("USA","314242000","9826675","R"),
("Ireland","6399152","84421","L"),
("UK","62262000","243610","L"),
("Mexico","113910608","1972550","R"),
("Denmark","5543453","43094","R"),
```

```
("Norway","5002942","385252","R"),
("UAE","8264070","83600","R"),
("India","1210193422","3287263","L"),
("China","1347350000","9640821","R"),
]

def switch_table(memc,table):
    key = "@@" + table
    print "Switching default table to " + table + " by issuing GET for " + key + "."
    result = memc.get(key)

def insert_country_data(memc):
    banner("Inserting initial data via memcached interface")
    for item in country_data:
        country = item[0]
        population = item[1]
        area = item[2]
        drive_side = item[3]

        key = country
        value = "".join([population,area,drive_side])
        print "Key = " + key
        print "Value = " + value

        if memc.add(key,value):
            print "Added new key, value pair."
        else:
            print "Updating value for existing key."
            memc.set(key,value)

def query_country_data(memc):
    banner("Retrieving data for all keys (country names)")
    for item in country_data:
        key = item[0]
        result = memc.get(key)
        print "Here is the result retrieved from the database for key " + key + ":"
        print result
        (m_population, m_area, m_drive_side) = result.split("|")
        print "Unpacked population value: " + m_population
        print "Unpacked area value      : " + m_area
        print "Unpacked drive side value: " + m_drive_side

if __name__ == '__main__':

    memc = connect_to_memcached()
    switch_table(memc,"bbb")
    insert_country_data(memc)
    query_country_data(memc)

    sys.exit(0)
```

サンプル Python アプリケーションのノート:

- データ操作は `memcached` インタフェースを介して実行されるため、アプリケーションの実行にデータベース認可は必要ありません。必要な情報は、`memcached` デーモンがリスニングするローカルシステム上のポート番号のみです。
- アプリケーションで `multicol` テーブルが使用されるようにするために、`switch_table()` 関数がコールされ、`@@` テーブル記法を使用してダミーの `get` または `set` リクエストが実行されます。リクエストの `name` 値は、`innodb_memcache.containers.name` フィールドで定義されている `multicol` テーブル識別子である `bbb` です。

実際のアプリケーションでは、より説明的な `name` 値を使用できます。この例は、`get @@...` リクエストでテーブル名ではなくテーブル識別子が指定されていることを示しています。
- データの挿入およびクエリーに使用されるユーティリティ関数は、`add` または `set` リクエストを使用して MySQL にデータを送信するために Python データ構造をパイプ区切りの値に変換する方法、および `get` リクエストによって返されるパイプ区切りの値を解凍する方法を示しています。この追加処理は、単一の `memcached` 値を複数の MySQL テーブルのカラムにマップする場合にのみ必要です。

4. サンプル Python アプリケーションを実行します。

```
shell> python multicol.py
```

成功した場合、サンプルアプリケーションは次の出力を返します:

```
Connected to memcached.
Switching default table to 'bbb' by issuing GET for '@@bbb'.

=====
Inserting initial data via memcached interface
=====
Key = Canada
Value = 34820000|9984670|R
Added new key, value pair.
Key = USA
Value = 314242000|9826675|R
Added new key, value pair.
Key = Ireland
Value = 6399152|84421|L
Added new key, value pair.
Key = UK
Value = 62262000|243610|L
Added new key, value pair.
Key = Mexico
Value = 113910608|1972550|R
Added new key, value pair.
Key = Denmark
Value = 5543453|43094|R
Added new key, value pair.
Key = Norway
Value = 5002942|385252|R
Added new key, value pair.
Key = UAE
Value = 8264070|83600|R
Added new key, value pair.
Key = India
Value = 1210193422|3287263|L
Added new key, value pair.
Key = China
Value = 1347350000|9640821|R
Added new key, value pair.

=====
Retrieving data for all keys (country names)
=====
Here is the result retrieved from the database for key Canada:
34820000|9984670|R
Unpacked population value: 34820000
Unpacked area value : 9984670
Unpacked drive side value: R
Here is the result retrieved from the database for key USA:
314242000|9826675|R
Unpacked population value: 314242000
Unpacked area value : 9826675
Unpacked drive side value: R
Here is the result retrieved from the database for key Ireland:
6399152|84421|L
Unpacked population value: 6399152
Unpacked area value : 84421
Unpacked drive side value: L
Here is the result retrieved from the database for key UK:
62262000|243610|L
Unpacked population value: 62262000
Unpacked area value : 243610
Unpacked drive side value: L
Here is the result retrieved from the database for key Mexico:
113910608|1972550|R
Unpacked population value: 113910608
Unpacked area value : 1972550
Unpacked drive side value: R
Here is the result retrieved from the database for key Denmark:
5543453|43094|R
Unpacked population value: 5543453
Unpacked area value : 43094
```

```

Unpacked drive side value: R
Here is the result retrieved from the database for key Norway:
5002942|385252|R
Unpacked population value: 5002942
Unpacked area value   : 385252
Unpacked drive side value: R
Here is the result retrieved from the database for key UAE:
8264070|83600|R
Unpacked population value: 8264070
Unpacked area value   : 83600
Unpacked drive side value: R
Here is the result retrieved from the database for key India:
1210193422|3287263|L
Unpacked population value: 1210193422
Unpacked area value   : 3287263
Unpacked drive side value: L
Here is the result retrieved from the database for key China:
1347350000|9640821|R
Unpacked population value: 1347350000
Unpacked area value   : 9640821
Unpacked drive side value: R

```

5. `innodb_memcache.containers` テーブルをクエリーして、`multicol` テーブルに対して前に挿入したレコードを表示します。最初のレコードは、`daemon_memcached` プラグインの初期設定時に作成される `demo_test` テーブルのサンプルエントリです。2 番目のレコードは、`multicol` テーブルに挿入したエントリです。

```

mysql> SELECT * FROM innodb_memcache.containers\G
***** 1. row *****
      name: aaa
      db_schema: test
      db_table: demo_test
      key_columns: c1
      value_columns: c2
      flags: c3
      cas_column: c4
      expire_time_column: c5
      unique_idx_name_on_key: PRIMARY
***** 2. row *****
      name: bbb
      db_schema: test
      db_table: multicol
      key_columns: country
      value_columns: population,area_sq_km,drive_side
      flags: c3
      cas_column: c4
      expire_time_column: c5
      unique_idx_name_on_key: PRIMARY

```

6. `multicol` テーブルをクエリーして、サンプル Python アプリケーションによって挿入されたデータを表示します。データは MySQL `queries` で使用でき、SQL またはアプリケーション (適切な `MySQL Connector or API` を使用) を介して同じデータにアクセスする方法を示します。

```

mysql> SELECT * FROM test.multicol;
+-----+-----+-----+-----+-----+-----+
| country | population | area_sq_km | drive_side | c3 | c4 | c5 |
+-----+-----+-----+-----+-----+-----+
| Canada | 34820000 | 9984670 | R | 0 | 11 | 0 |
| China | 1347350000 | 9640821 | R | 0 | 20 | 0 |
| Denmark | 5543453 | 43094 | R | 0 | 16 | 0 |
| India | 1210193422 | 3287263 | L | 0 | 19 | 0 |
| Ireland | 6399152 | 84421 | L | 0 | 13 | 0 |
| Mexico | 113910608 | 1972550 | R | 0 | 15 | 0 |
| Norway | 5002942 | 385252 | R | 0 | 17 | 0 |
| UAE | 8264070 | 83600 | R | 0 | 18 | 0 |
| UK | 62262000 | 243610 | L | 0 | 14 | 0 |
| USA | 314242000 | 9826675 | R | 0 | 12 | 0 |
+-----+-----+-----+-----+-----+-----+

```

注記

数値として扱われるカラムの長さを定義する場合は、必要な桁数、小数点、符号文字、先頭のゼロなどを保持するのに十分なサイズを常に許可してください。 `VARCHAR` など

の文字列カラムの Too-long 値は、意味のない数値を生成する可能性のある一部の文字を削除することによって切り捨てられます。

- オプションで、memcached データを格納する InnoDB テーブルに対してレポートタイプのクエリーを実行します。

country キーカラムのみでなく、任意のカラムに対して計算およびテストを実行して、SQL クエリーを介してレポートを生成できます。(次の例では、少数の国のデータのみを使用しているため、数字は説明のみを目的としています。) 次のクエリーは、人々が右側を運転する国の平均人口、および名前が「U」で始まる国の平均サイズを返します:

```
mysql> SELECT AVG(population) FROM multicol WHERE drive_side = 'R';
+-----+
| avg(population) |
+-----+
| 261304724.7142857 |
+-----+

mysql> SELECT SUM(area_sq_km) FROM multicol WHERE country LIKE 'U%';
+-----+
| sum(area_sq_km) |
+-----+
| 10153885 |
+-----+
```

population および area_sq_km のカラムには強い型指定の数値データではなく文字データが格納されるため、AVG() や SUM() などの関数は、最初に各値を数値に変換することによって機能します。このアプローチでは、< や > などの演算子に機能しないを使用します。たとえば、ORDER BY population DESC などの句から想定されていない文字ベースの値 9 > 1000 を比較する場合などです。もっとも正確な型処理を行うには、数値カラムを適切な型にキャストするビューに対してクエリーを実行します。この手法を使用すると、キャスト、フィルタリングおよび順序付けが正しいことを確認しながら、データベースアプリケーションから単純な SELECT * クエリーを発行できます。次の例は、母集団の降順で上位 3 か国を検索するためにクエリーすることができるビューを示しています。結果には、multicol テーブルの最新データが反映され、母集団と面積の数値が数値として扱われます:

```
mysql> CREATE VIEW populous_countries AS
SELECT
  country,
  cast(population as unsigned integer) population,
  cast(area_sq_km as unsigned integer) area_sq_km,
  drive_side FROM multicol
ORDER BY CAST(population as unsigned integer) DESC
LIMIT 3;

mysql> SELECT * FROM populous_countries;
+-----+-----+-----+-----+
| country | population | area_sq_km | drive_side |
+-----+-----+-----+-----+
| China | 1347350000 | 9640821 | R |
| India | 1210193422 | 3287263 | L |
| USA | 314242000 | 9826675 | R |
+-----+-----+-----+-----+

mysql> DESC populous_countries;
+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+
| country | varchar(128) | NO | | | |
| population | bigint(10) unsigned | YES | | NULL | |
| area_sq_km | int(9) unsigned | YES | | NULL | |
| drive_side | varchar(1) | YES | | NULL | |
+-----+-----+-----+-----+
```

15.20.6.2 InnoDB memcached プラグインに対する memcached アプリケーションの適応

daemon_memcached プラグインを使用するように既存の memcached アプリケーションを適応させる場合は、MySQL および InnoDB テーブルの次の側面を考慮してください:

- 数バイトを超えるキー値がある場合は、InnoDB テーブルの **primary key** として数値自動増分カラムを使用し、**memcached** キー値を含むカラムに一意の **secondary index** を作成する方が効率的です。これは、主キー値が (自動増分値と同様に) ソートされた順序で追加されている場合、InnoDB が大規模な挿入に最適なパフォーマンスを発揮するためです。主キー値はセカンダリインデックスに含まれ、主キーが長い文字列値の場合は不要な領域を占有します。
- **memcached** を使用して複数の異なるクラスの情報を格納する場合は、データのタイプごとに個別の InnoDB テーブルを設定することを検討してください。 `innodb_memcache.containers` テーブルに追加のテーブル識別子を定義し、 `@@table_id.key` テーブル記を使用して異なるテーブルのアイテムを格納および取得します。様々なタイプの情報を物理的に分割することで、最適な領域使用率、パフォーマンスおよび信頼性のために各テーブルの特性をチューニングできます。たとえば、ブログ投稿を保持するテーブルに対しては **compression** を有効にできますが、サムネイルイメージを保持するテーブルに対しては有効にできません。非常に重要なデータが保持されているテーブルは、別のテーブルよりも頻繁にバックアップする場合があります。SQL を使用してレポートを生成するために頻繁に使用されるテーブルに追加の **secondary indexes** を作成できます。
- できれば、 `daemon_memcached` プラグインで使用する安定したテーブル定義のセットを構成し、テーブルを永続的に配置したままにします。 `innodb_memcache.containers` テーブルへの変更は、 `innodb_memcache.containers` テーブルへの次のクエリー時に有効になります。コンテナテーブルのエントリは起動時に処理され、認識されないテーブル識別子 (`containers.name` で定義) が `@@` テーブル記を使用してリクエストされるたびに参照されます。したがって、関連するテーブル識別子を使用するとすぐに新しいエントリが表示されますが、既存のエントリへの変更を有効にするには、サーバーの再起動が必要です。
- デフォルトの `innodb_only` キャッシュポリシーを使用すると、 `add()`, `set()`, `incr()` のコールは成功しますが、 `while expecting 'STORED', got unexpected response 'NOT_STORED'` などのデバッグメッセージはトリガーされます。デバッグメッセージが発生するのは、新しい値と更新された値が、 `innodb_only` キャッシュポリシーのためにメモリーキャッシュに保存されずに InnoDB テーブルに直接送信されるためです。

15.20.6.3 InnoDB memcached プラグインのパフォーマンスのチューニング

InnoDB を **memcached** と組み合わせて使用すると、すぐに行うか後で行うかにかかわらず、すべてのデータをディスクに書き込む必要があるため、RAW パフォーマンスはそれ自体で **memcached** を使用するより多少遅くなることが予想されます。 **InnoDB memcached** プラグインを使用する場合、同等の SQL 操作よりも優れたパフォーマンスを実現するために、 **memcached** 操作のチューニング目標に焦点を当てます。

ベンチマークは、 **memcached** インタフェースを使用するクエリーおよび **DML** 操作 (挿入、更新および削除) が従来の SQL より高速であることを示しています。DML 操作では、通常、改善点が大きくなります。したがって、最初に **memcached** インタフェースを使用するように書き込み集中型アプリケーションを調整することを検討してください。また、信頼性のない高速で軽量なメカニズムを使用する書き込み集中型アプリケーションの適応に優先順位を付けることも検討してください。

SQL クエリーの改変

単純な **GET** リクエストに最も適したクエリーのタイプは、 **WHERE** 句に単一の句または **AND** 条件のセットを含むクエリーです:

```
SQL:
SELECT col FROM tbl WHERE key = 'key_value';

memcached:
get key_value

SQL:
SELECT col FROM tbl WHERE col1 = val1 and col2 = val2 and col3 = val3;

memcached:
# Since you must always know these 3 values to look up the key,
# combine them into a unique string and use that as the key
# for all ADD, SET, and GET operations.
key_value = val1 + ":" + val2 + ":" + val3
get key_value

SQL:
SELECT 'key exists!' FROM tbl
WHERE EXISTS (SELECT col1 FROM tbl WHERE KEY = 'key_value') LIMIT 1;
```

```
memcached:
# Test for existence of key by asking for its value and checking if the call succeeds,
# ignoring the value itself. For existence checking, you typically only store a very
# short value such as "1".
get key_value
```

システムメモリーの使用

最高のパフォーマンスを得るには、`innodb_buffer_pool_size` 構成オプションを使用して、システム RAM の大部分が `InnoDB buffer pool` 専用である一般的なデータベースサーバーとして構成されているマシンに `daemon_memcached` プラグインをデプロイします。マルチギガバイトバッファプールがあるシステムでは、ほとんどの操作にすでにメモリーにキャッシュされているデータが含まれる場合、スループットを最大化するために `innodb_buffer_pool_instances` の値を増やすことを検討してください。

冗長 I/O の削減

InnoDB には、高い信頼性 (クラッシュの場合) と高い書き込みワークロード中の I/O オーバーヘッドの量のバランスを選択できる多数の設定があります。たとえば、`innodb_doublewrite` を 0 に、`innodb_flush_log_at_trx_commit` を 2 に設定することを検討してください。様々な `innodb_flush_method` 設定でパフォーマンスを測定します。

テーブル操作の I/O を削減したりチューニングしたりするその他の方法については、[セクション 8.5.8 「InnoDB ディスク I/O の最適化」](#) を参照してください。

トランザクションオーバーヘッドの削減

`daemon_memcached_r_batch_size` および `daemon_memcached_w_batch_size` のデフォルト値の 1 は、格納または更新されたデータの結果の信頼性と安全性を最大限に高めるためのものです。

アプリケーションのタイプによっては、これらの設定の 1 つまたは両方を増やすと、頻繁な `コミット` 操作によるオーバーヘッドを削減できる場合もあります。ビジー状態のシステムでは、SQL を介して行われたデータへの変更が `memcached` にすぐに表示されないことを知っている (つまり、`N` の `get` 操作がさらに処理されるまで) `daemon_memcached_r_batch_size` を増やすことができます。すべての書き込み操作を確実に格納する必要があるデータを処理する場合は、`daemon_memcached_w_batch_size` を 1 に設定したままにします。統計分析のみを目的とした大量の更新を処理する場合は、予期しない終了での最後の `N` 更新の消失が許容されるリスクである設定を増やします。

たとえば、ビジーブリッジを通過するトラフィックを監視し、毎日約 100,000 台の車両のデータを記録するシステムについて考えてみます。アプリケーションがトラフィックパターンを分析するために様々なタイプの車両をカウントする場合、`daemon_memcached_w_batch_size` を 1 から 100 に変更すると、`コミット` 操作の I/O オーバーヘッドが 99% 削減されます。停止の場合、最大 100 個のレコードが失われます。これはエラーの許容マージンである可能性があります。かわりに、アプリケーションが自動車ごとに自動通話回収を実行した場合は、`daemon_memcached_w_batch_size` を 1 に設定して、各通話レコードがすぐにディスクに保存されるようにします。

InnoDB が `memcached` キー値をディスク上に編成する方法のため、作成するキーが多数ある場合は、キーを任意の順序で作成するよりも、アプリケーションのキー値でデータ項目をソートし、`add` をソート順でソートの方が高速な場合があります。

通常の `memcached` ディストリビューションの一部であるが、`daemon_memcached` プラグインには含まれていない `memslap` コマンドは、異なる構成をベンチマークする場合に役立ちます。また、独自のベンチマークで使用するサンプルのキーと値のペアを生成するためにも使用できます。詳細は、[libmemcached Command-Line Utilities](#) を参照してください。

15.20.6.4 InnoDB memcached プラグインのトランザクション動作の制御

従来の `memcached` とは異なり、`daemon_memcached` プラグインを使用すると、`add`、`set`、`incr` のコールによって生成されるデータ値の永続性などを制御できます。デフォルトでは、`memcached` インタフェースを介して書き込まれたデータはディスクに格納され、`get` のコールはディスクから最新の値を返します。デフォルトの動作では最適な RAW パフォーマンスは提供されませんが、InnoDB テーブルの SQL インタフェースと比べて高速です。

`daemon_memcached` プラグインの使用経験があるため、重要でないデータクラスの永続性設定を緩和することを検討できます。ただし、停止時に一部の更新された値が失われたり、若干古いデータが返されるリスクがあります。

コミットの頻度

永続性と本来のパフォーマンスを両立させる 1 つの条件は、新しいデータや変更されたデータが **コミット** される頻度です。データが重要な場合は、予期しない終了または停止が発生した場合に安全になるように、すぐにコミットする必要があります。予期しない終了後にリセットされるカウンタや失われる可能性のあるロギングデータなど、データのクリティカル性が低い場合は、コミットの頻度を低くして使用可能な RAW スループットを高くすることをお勧めします。

memcached 操作によって基礎となる InnoDB テーブルのデータが挿入、更新または削除される場合、変更は InnoDB テーブルに即座に (`daemon_memcached_w_batch_size=1` の場合) または後で (`daemon_memcached_w_batch_size` 値が 1 より大きい場合) コミットされる可能性があります。いずれの場合も、変更はロールバックできません。ピーク時間中に高い I/O オーバーヘッドを回避するために `daemon_memcached_w_batch_size` の値を増やすと、ワークロードが減少したときにコミットの頻度が低下する可能性があります。安全策として、バックグラウンドスレッドで、**memcached** API 経由で行なった変更を一定の間隔で自動的にコミットします。間隔は、5 秒のデフォルト設定を持つ `innodb_api_bk_commit_interval` 構成オプションによって制御されます。

memcached 操作によって基礎となる InnoDB テーブルのデータが挿入または更新されると、MySQL 側でまだコミットされていない場合でも、新しい値はメモリーキャッシュに残されるため、変更されたデータは他の **memcached** リクエストにすぐに表示されます。

トランザクションの分離

`get` や `incr` などの **memcached** 操作によって基礎となる InnoDB テーブルに対するクエリーまたは DML 操作が発生した場合、操作でテーブルに書き込まれた最新のデータ、コミットされたデータのみ、またはトランザクション `isolation level` のその他のバリエーションを表示するかどうかを制御できます。この機能を制御するには、`innodb_api_trx_level` 構成オプションを使用します。このオプションに指定された数値は、**REPEATABLE READ** などの分離レベルに対応します。その他の設定の詳細は、`innodb_api_trx_level` オプションの説明を参照してください。

厳密な分離レベルでは、取得したデータが突然ロールバックまたは変更されず、後続のクエリーで異なる値が返されることはありません。ただし、厳密な分離レベルでは、待機を引き起こす可能性のある **locking** オーバーヘッドの増加が必要になります。長時間実行トランザクションを使用しない NoSQL 形式のアプリケーションの場合は、通常、デフォルトの分離レベルを使用するか、より厳しくない分離レベルに切り替えることができます。

memcached DML 操作の行ロックの無効化

`innodb_api_disable_rowlock` オプションを使用すると、`daemon_memcached` プラグインを介した **memcached** リクエストによって DML 操作が発生した場合に行ロックを無効にできます。デフォルトでは、`innodb_api_disable_rowlock` は **OFF** に設定されており、これは **memcached** が `get` および `set` 操作の行ロックを要求することを意味します。`innodb_api_disable_rowlock` を **ON** に設定すると、**memcached** は行ロックの代わりに、テーブルロックをリクエストします。

`innodb_api_disable_rowlock` オプションは動的ではありません。起動時に `mysqld` コマンドラインで指定するか、MySQL 構成ファイルに入力する必要があります。

DDL の許可または禁止

デフォルトでは、`daemon_memcached` プラグインで使用されるテーブルに対して **ALTER TABLE** などの **DDL** 操作を実行できます。これらのテーブルが高スループットアプリケーションに使用される場合の潜在的な速度低下を回避するには、起動時に `innodb_api_enable_mdll` を有効にして、これらのテーブルに対する **DDL** 操作を無効にします。このオプションは、**memcached** と SQL の両方を介して同じテーブルにアクセスする場合にはあまり適切ではありません。これは、テーブルに対する **CREATE INDEX** ステートメントがブロックされるためです。これは、レポートクエリーの実行に重要な場合があるためです。

ディスク、メモリーまたはその両方へのデータの格納

`innodb_memcache.cache_policies` テーブルでは、**memcached** インタフェースを介してディスクに書き込まれたデータを格納するか (`innodb_only`、デフォルト)、従来の **memcached** (`cache_only`) と同様にメモリー内だけに格納するか、またはその両方 (`caching`) を指定します。

`caching` 設定では、**memcached** がメモリー内からキーを検出できない場合、InnoDB テーブル内から値を検索します。InnoDB テーブルのディスクで値が更新されたが、まだメモリーキャッシュから期限切れになっていない場合、`caching` 設定で `get` コールから返される値は期限切れになる可能性があります。

キャッシュポリシーは、`get`、`set` (`incr` および `decr` を含む)、`delete`、および `flush` 操作で個々に設定できます。

たとえば、`get` および `set` 操作で、テーブルを、および同時に (`cache_only` 設定を使用して) `memcached` メモリーキャッシュをクエリーまたは更新でき、一方、`delete`、`flush`、またはその両方が (`cache_only` 設定を使用して) メモリー内コピーでのみ動作するようにします。このようにして、アイテムを削除またはフラッシュすると、そのアイテムはキャッシュからのみ期限切れになり、アイテムが次回リクエストされたときに InnoDB テーブルから最新の値が返されます。

```
mysql> SELECT * FROM innodb_memcache.cache_policies;
+-----+-----+-----+-----+-----+
| policy_name | get_policy | set_policy | delete_policy | flush_policy |
+-----+-----+-----+-----+-----+
| cache_policy | innodb_only | innodb_only | innodb_only | innodb_only |
+-----+-----+-----+-----+-----+

mysql> UPDATE innodb_memcache.cache_policies SET set_policy = 'caching'
WHERE policy_name = 'cache_policy';
```

`innodb_memcache.cache_policies` の値は、起動時のみ読み取られます。このテーブルの値を変更したら、`daemon_memcached` プラグインをアンインストールして再インストールし、変更が有効になるようにします。

```
mysql> UNINSTALL PLUGIN daemon_memcached;

mysql> INSTALL PLUGIN daemon_memcached soname "libmemcached.so";
```

15.20.6.5 memcached 操作に合わせた DML ステートメントの改変

ベンチマークは、`daemon_memcached` プラグインがクエリーを高速化するよりも DML 操作 (挿入、更新および削除) を高速化することを示しています。したがって、I/O-bound、である書き込み集中型アプリケーションに初期開発作業に焦点を当て、新しい書き込み集中型アプリケーション用に `daemon_memcached` プラグインとともに MySQL を使用する機会を探すことを検討してください。

単一行 DML ステートメントは、`memcached` 操作に変換する最も簡単なタイプのステートメントです。INSERT は `add` になり、UPDATE は `set`、`incr` または `decr` になり、DELETE は `delete` になります。key はテーブル内で一意であるため、これらの操作は、`memcached` インタフェースを介して発行された場合にのみ影響を受けることが保証されます。

次の SQL の例では、`t1` は、`innodb_memcache.containers` テーブルの構成に基づいて、`memcached` 操作に使用されるテーブルを参照します。key は `key_columns` の下にリストされているカラムを示し、val は `value_columns` の下にリストされているカラムを示します。

```
INSERT INTO t1 (key,val) VALUES (some_key,some_value);
SELECT val FROM t1 WHERE key = some_key;
UPDATE t1 SET val = new_value WHERE key = some_key;
UPDATE t1 SET val = val + x WHERE key = some_key;
DELETE FROM t1 WHERE key = some_key;
```

テーブルからすべての行を削除する次の `TRUNCATE TABLE` および `DELETE` ステートメントは、前の例のように `t1` が `memcached` 操作のテーブルとして構成されている `flush_all` 操作に対応しています。

```
TRUNCATE TABLE t1;
DELETE FROM t1;
```

15.20.6.6 ベースとなる InnoDB テーブルでの DML および DDL ステートメントの実行

基礎となる InnoDB テーブル (デフォルトでは `test.demo_test`) には、標準 SQL インタフェースを介してアクセスできます。ただし、いくつかの制約があります。

- `memcached` インタフェースを介してもアクセスされるテーブルをクエリーする場合は、すべての書き込み操作の後ではなく、定期的にコミットされるように `memcached` 操作を構成することに注意してください。この動作は、`daemon_memcached_w_batch_size` オプションによって制御されます。このオプションが 1 より大きい値に設定されている場合は、`READ UNCOMMITTED` クエリーを使用して、挿入されたばかりの行を検索します。

```
mysql> SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

mysql> SELECT * FROM demo_test;
+-----+-----+-----+-----+-----+-----+-----+-----+
| cx | cy | c1 | cz | c2 | ca | CB | c3 | cu | c4 | C5 |
+-----+-----+-----+-----+-----+-----+-----+-----+
```



```

+-----+-----+-----+-----+-----+-----+-----+-----+
| NULL | NULL | a11 | NULL | 123456789 | NULL | NULL | 10 | NULL | 3 | NULL |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

- **memcached** インタフェースを介してもアクセスされる SQL を使用してテーブルを変更する場合、読取り操作ごとではなく、新しいトランザクションを定期的を開始するように **memcached** 操作を構成できます。この動作は、**daemon_memcached_r_batch_size** オプションによって制御されます。このオプションが 1 より大きい値に設定されている場合、SQL を使用してテーブルに加えられた変更は、**memcached** 操作ですぐには表示されません。
- **InnoDB** テーブルは、IS (インテント共有) または IX (インテント排他) のいずれかで、トランザクション内のすべての操作に対してロックされています。**daemon_memcached_r_batch_size** および **daemon_memcached_w_batch_size** を 1 のデフォルト値から大幅に増やすと、各操作間でテーブルがロックされる可能性が高くなり、テーブルに対する **DDL** ステートメントが防止されます。

15.20.7 InnoDB memcached プラグインとレプリケーション

daemon_memcached プラグインは MySQL **binary log** をサポートしているため、**memcached** インタフェースを介してソースサーバーをレプリケートし、バックアップ、集中型の読取りワークロードおよび高可用性のバランスを取ることができます。バイナリロギングでは、すべての **memcached** コマンドがサポートされます。

レプリカサーバーに **daemon_memcached** プラグインを設定する必要はありません。この構成の主な利点は、ソースでの書き込みスループットの向上です。レプリケーションメカニズムの速度は影響を受けません。

次のセクションでは、MySQL レプリケーションで **daemon_memcached** プラグインを使用するときバイナリログ機能を使用する方法について説明します。[セクション15.20.3「InnoDB memcached プラグインの設定」](#) で説明されている設定が完了していることを前提としています。

InnoDB memcached バイナリログの有効化

1. MySQL **binary log** で **daemon_memcached** プラグインを使用するには、ソースサーバーで **innodb_api_enable_binlog** 構成オプションを有効にします。このオプションは、サーバーの起動時にのみ設定できます。**--log-bin** オプションを使用して、ソースサーバーで MySQL バイナリログを有効にする必要もあります。これらのオプションは、MySQL 構成ファイルまたは **mysqld** コマンドラインに追加できます。

```
mysqld ... --log-bin --innodb_api_enable_binlog=1
```

2. [セクション17.1.2「バイナリログファイルの位置ベースのレプリケーションの設定」](#) の説明に従って、ソースサーバーとレプリカサーバーを構成します。
3. **mysqldump** を使用して、ソースデータスナップショットを作成し、スナップショットをレプリカサーバーに同期します。

```
source shell> mysqldump --all-databases --lock-all-tables > dbdump.db
replica shell> mysql < dbdump.db
```

4. ソースサーバーで、**SHOW MASTER STATUS** を発行してソースバイナリログ座標を取得します。

```
mysql> SHOW MASTER STATUS;
```

5. レプリカサーバーで、(MySQL 8.0.23 の) **CHANGE REPLICATION SOURCE TO** ステートメントまたは (MySQL 8.0.23 の前の) **CHANGE MASTER TO** ステートメントを使用して、ソースバイナリログ座標を使用するレプリカサーバーを設定します。

```
mysql> CHANGE MASTER TO
  MASTER_HOST='localhost',
  MASTER_USER='root',
  MASTER_PASSWORD='',
  MASTER_PORT = 13000,
  MASTER_LOG_FILE='0.000001',
  MASTER_LOG_POS=114;
```

Or from MySQL 8.0.23:

```
mysql> CHANGE REPLICATION SOURCE TO
  SOURCE_HOST='localhost',
  SOURCE_USER='root',
  SOURCE_PASSWORD='',
  SOURCE_PORT = 13000,
```

```
SOURCE_LOG_FILE='0.000001',
SOURCE_LOG_POS=114;
```

- レプリカを起動します。

```
mysql> START SLAVE;
Or from MySQL 8.0.22:
mysql> START REPLICA;
```

エラーログに次のような出力が出力された場合、レプリカはレプリケーションの準備ができています。

```
2013-09-24T13:04:38.639684Z 49 [Note] Replication I/O thread: connected to
source 'root@localhost:13000', replication started in log '0.000001'
at position 114
```

InnoDB memcached レプリケーション構成のテスト

この例では、`memcached` および `telnet` を使用して `InnoDB memcached` レプリケーション構成をテストし、データを挿入、更新および削除する方法を示します。MySQL クライアントは、ソースサーバーとレプリカサーバーの結果を検証するために使用されます。

この例では、`daemon_memcached` プラグインの初期設定時に `innodb_memcached_config.sql` 構成スクリプトによって作成された `demo_test` テーブルを使用します。`demo_test` テーブルには、単一のサンプルレコードが含まれます。

- `set` コマンドを使用して、キーが `test1`、フラグ値が `10`、有効期限値が `0`、`cas` 値が `1` および `t1` のレコードを挿入します。

```
telnet 127.0.0.1 11211
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
set test1 10 0 1
t1
STORED
```

- ソースサーバーで、レコードが `demo_test` テーブルに挿入されたことを確認します。`demo_test` テーブルが以前に変更されていないと仮定すると、2つのレコードがあります。キーが `AA` で、キーが `test1` のレコードの例を示します。`c1` カラムはキーに、`c2` カラムは値に、`c3` カラムはフラグ値に、`c4` カラムは `cas` 値に、`c5` カラムは有効期限にマップされます。有効期限は未使用であるため、`0` に設定されました。

```
mysql> SELECT * FROM test.demo_test;
+-----+-----+-----+-----+-----+
| c1 | c2      | c3 | c4 | c5 |
+-----+-----+-----+-----+
| AA | HELLO, HELLO | 8 | 0 | 0 |
| test1 | t1      | 10 | 1 | 0 |
+-----+-----+-----+-----+
```

- 同じレコードがレプリカサーバーにレプリケートされたことを確認する場合は選択します。

```
mysql> SELECT * FROM test.demo_test;
+-----+-----+-----+-----+-----+
| c1 | c2      | c3 | c4 | c5 |
+-----+-----+-----+-----+
| AA | HELLO, HELLO | 8 | 0 | 0 |
| test1 | t1      | 10 | 1 | 0 |
+-----+-----+-----+-----+
```

- `set` コマンドを使用して、キーを `new` の値に更新します。

```
telnet 127.0.0.1 11211
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
set test1 10 0 2
new
STORED
```

更新はレプリカサーバーにレプリケートされます (`cas` 値も更新されます)。

```
mysql> SELECT * FROM test.demo_test;
```



```
+-----+-----+-----+-----+
| c1 | c2      | c3 | c4 | c5 |
+-----+-----+-----+-----+
| AA | HELLO, HELLO | 8 | 0 | 0 |
| test1 | new      | 10 | 2 | 0 |
+-----+-----+-----+-----+
```

5. `delete` コマンドを使用して `test1` レコードを削除します。

```
telnet 127.0.0.1 11211
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
delete test1
DELETED
```

`delete` 操作がレプリカにレプリケートされると、レプリカ上の `test1` レコードも削除されます。

```
mysql> SELECT * FROM test.demo_test;
+-----+-----+-----+-----+
| c1 | c2      | c3 | c4 | c5 |
+-----+-----+-----+-----+
| AA | HELLO, HELLO | 8 | 0 | 0 |
+-----+-----+-----+-----+
```

6. `flush_all` コマンドを使用して、テーブルからすべての行を削除します。

```
telnet 127.0.0.1 11211
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
flush_all
OK
```

```
mysql> SELECT * FROM test.demo_test;
Empty set (0.00 sec)
```

7. ソースサーバーに Telnet し、2 つの新しいレコードを入力します。

```
telnet 127.0.0.1 11211
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
set test2 10 0 4
again
STORED
set test3 10 0 5
again1
STORED
```

8. 2 つのレコードがレプリカサーバーにレプリケートされたことを確認します。

```
mysql> SELECT * FROM test.demo_test;
+-----+-----+-----+-----+
| c1 | c2      | c3 | c4 | c5 |
+-----+-----+-----+-----+
| test2 | again  | 10 | 4 | 0 |
| test3 | again1 | 10 | 5 | 0 |
+-----+-----+-----+-----+
```

9. `flush_all` コマンドを使用して、テーブルからすべての行を削除します。

```
telnet 127.0.0.1 11211
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
flush_all
OK
```

10. `flush_all` 操作がレプリカサーバーでレプリケートされたことを確認します。

```
mysql> SELECT * FROM test.demo_test;
Empty set (0.00 sec)
```

InnoDB memcached バイナリログノート

バイナリログ形式:

- ほとんどの `memcached` 操作は (挿入、削除、更新に類似した) `DML` ステートメントにマップされます。MySQL サーバーでは実際の SQL ステートメントが処理されないため、すべての `memcached` コマンド (`flush_all` を除く) で行ベースのレプリケーション (RBR) ログギングが使用され、これはサーバーの `binlog_format` 設定とは無関係です。
- `memcached flush_all` コマンドは、MySQL 5.7 以前の `TRUNCATE TABLE` コマンドにマップされます。DDL コマンドはステートメントベースのログギングのみを使用できるため、`flush_all` コマンドは `TRUNCATE TABLE` ステートメントを送信することによってレプリケートされます。MySQL 8.0 以降では、`flush_all` は `DELETE` にマップされますが、`TRUNCATE TABLE` ステートメントを送信することでレプリケートされます。

トランザクション:

- トランザクション** の概念は、これまで通常は `memcached` アプリケーションの一部をなすものではありませんでした。パフォーマンスを考慮するために、`daemon_memcached_r_batch_size` および `daemon_memcached_w_batch_size` を使用して、読取りおよび書き込みトランザクションのバッチサイズを制御します。これらの設定はレプリケーションには影響しません。基礎となる InnoDB テーブルに対する各 SQL 操作は、正常に完了した後にレプリケートされます。
- `daemon_memcached_w_batch_size` のデフォルト値は 1 です。これは、各 `memcached` 書き込み操作がただちにコミットされることを意味します。このデフォルト設定では、ソースサーバーとレプリカサーバーに表示されるデータの不整合を回避するために、一定量のパフォーマンスオーバーヘッドが発生します。レプリケートされたレコードは、常にレプリカサーバーですぐに使用できます。`daemon_memcached_w_batch_size` を 1 より大きい値に設定すると、`memcached` を介して挿入または更新されたレコードはすぐにソースサーバーに表示されません。コミット前にソースサーバー上のレコードを表示するには、`SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED` を発行します。

15.20.8 InnoDB memcached プラグインの内部

InnoDB memcached プラグイン用の InnoDB API

InnoDB memcached エンジンは、埋込み InnoDB から直接採用された InnoDB API を介して InnoDB にアクセスします。InnoDB API 関数は、コールバック関数として InnoDB memcached エンジンに渡されます。InnoDB API 関数は、InnoDB テーブルに直接アクセスします。ほとんどの場合、DML 操作ですが、`TRUNCATE TABLE` は例外です。

`memcached` コマンドは、InnoDB memcached API を介して実装されます。次のテーブルに、`memcached` コマンドが DML または DDL 操作にどのようにマップされるかを示します。

表 15.27 memcached コマンドおよび関連する DML または DDL 操作

memcached コマンド	DML または DDL 操作
<code>get</code>	読取り/フェッチコマンド
<code>set</code>	<code>INSERT</code> または <code>UPDATE</code> が続く検索 (キーが存在するかどうかによる)
<code>add</code>	<code>INSERT</code> または <code>UPDATE</code> が続く検索
<code>replace</code>	<code>UPDATE</code> が続く検索
<code>append</code>	検索の後に <code>UPDATE</code> が続きます (<code>UPDATE</code> の前に結果にデータを追加します)
<code>prepend</code>	検索の後に <code>UPDATE</code> が続く (<code>UPDATE</code> の前にデータを結果の先頭に付加する)
<code>incr</code>	<code>UPDATE</code> が続く検索
<code>decr</code>	<code>UPDATE</code> が続く検索
<code>delete</code>	<code>DELETE</code> が続く検索
<code>flush_all</code>	<code>TRUNCATE TABLE</code> (DDL)

InnoDB memcached プラグインの構成テーブル

このセクションでは、`daemon_memcached` プラグインで使用される構成テーブルについて説明します。`cache_policies` テーブル、`config_options` テーブルおよび `containers` テーブルは、`innodb_memcache` データベースの `innodb_memcached_config.sql` 構成スクリプトによって作成されます。

```
mysql> USE innodb_memcache;
Database changed
mysql> SHOW TABLES;
+-----+
| Tables_in_innodb_memcache |
+-----+
| cache_policies             |
| config_options             |
| containers                  |
+-----+
```

cache_policies テーブル

`cache_policies` テーブルでは、InnoDB memcached インストールのキャッシュポリシーを定義します。単一のキャッシュポリシー内で、`get`、`set`、`delete` および `flush` 操作の個々のポリシーを指定できます。すべての操作のデフォルト設定は `innodb_only` です。

- `innodb_only`: InnoDB をデータストアとして使用します。
- `cache_only`: memcached エンジンを実データストアとして使用します。
- `caching`: InnoDB と memcached エンジンの両方をデータストアとして使用します。この状況で、memcached がメモリー内からキーを検出できない場合、InnoDB テーブル内から値を検索します。
- `disable`: キャッシュを無効にします。

表 15.28 cache_policies カラム

カラム	説明
<code>policy_name</code>	キャッシュポリシーの名前。デフォルトのキャッシュポリシー名は <code>cache_policy</code> です。
<code>get_policy</code>	<code>get</code> 操作のキャッシュポリシー。有効な値は、 <code>innodb_only</code> 、 <code>cache_only</code> 、 <code>caching</code> または <code>disabled</code> です。デフォルト設定は <code>innodb_only</code> です。
<code>set_policy</code>	<code>set</code> 操作のキャッシュポリシー。有効な値は、 <code>innodb_only</code> 、 <code>cache_only</code> 、 <code>caching</code> または <code>disabled</code> です。デフォルト設定は <code>innodb_only</code> です。
<code>delete_policy</code>	<code>delete</code> 操作のキャッシュポリシー。有効な値は、 <code>innodb_only</code> 、 <code>cache_only</code> 、 <code>caching</code> または <code>disabled</code> です。デフォルト設定は <code>innodb_only</code> です。
<code>flush_policy</code>	<code>flush</code> 操作のキャッシュポリシー。有効な値は、 <code>innodb_only</code> 、 <code>cache_only</code> 、 <code>caching</code> または <code>disabled</code> です。デフォルト設定は <code>innodb_only</code> です。

config_options テーブル

`config_options` テーブルには、SQL を使用して実行時に変更できる memcached 関連の設定が格納されます。サポートされる構成オプションは、`separator` および `table_map_delimiter` です。

表 15.29 config_options カラム

カラム	説明
<code>Name</code>	memcached 関連の構成オプションの名前。 <code>config_options</code> テーブルでは、次の構成オプションがサポートされています:

カラム	説明
	<ul style="list-style-type: none"> separator: 複数の <code>value_columns</code> が定義されている場合に、1 つの長い文字列の値を別々の値に分離するために使用します。デフォルトでは、<code>separator</code> は <code> </code> 文字です。たとえば、<code>col1</code>, <code>col2</code> を値カラムとして定義し、<code> </code> をセパレータとして定義する場合、次の <code>memcached</code> コマンドを発行して、<code>col1</code> および <code>col2</code> にそれぞれ値を挿入できます: <pre>set keyx 10 0 19 valuecolx valuecoly</pre> <code>valuecol1x</code> は <code>col1</code> に格納され、<code>valuecoly</code> は <code>col2</code> に格納されます。 table_map_delimiter: 特定のテーブル内のキーにアクセスするために、キー名に <code>@@</code> 表記を使用するとき、スキーマ名とテーブル名を区切る文字。たとえば、<code>@@t1.some_key</code> と <code>@@t2.some_key</code> は同じキー値を持っていますが、異なるテーブルに格納されます。
値	<code>memcached</code> 関連の構成オプションに割り当てられた値。

containers テーブル

`containers` テーブルは、3 つの構成テーブルの中で最も重要です。 `memcached` 値の格納に使用される各 InnoDB テーブルには、`containers` テーブルのエントリが必要です。このエントリは、InnoDB テーブルのカラムとコンテナテーブルのカラムの間のマッピングを提供します。これは、`memcached` が InnoDB テーブルを操作するために必要です。

`containers` テーブルには、`innodb_memcached_config.sql` 構成スクリプトによって作成される `test.demo_test` テーブルのデフォルトエントリが含まれています。独自の InnoDB テーブルで `daemon_memcached` プラグインを使用するには、`containers` テーブルにエントリを作成する必要があります。

表 15.30 containers のカラム

カラム	説明
<code>name</code>	コンテナに付与された名前。 <code>@@</code> テーブル記法を使用して名前が InnoDB テーブルがリクエストされない場合、 <code>daemon_memcached</code> プラグインは <code>containers.name</code> 値が <code>default</code> の InnoDB テーブルを使用します。このようなエントリがない場合は、 <code>containers</code> テーブルの最初のエントリが <code>name</code> (昇順) でアルファベット順に並べられて、デフォルトの InnoDB テーブルが決定されます。
<code>db_schema</code>	InnoDB テーブルが存在するデータベースの名前。これは必須の値です。
<code>db_table</code>	<code>memcached</code> 値を格納する InnoDB テーブルの名前。これは必須の値です。
<code>key_columns</code>	<code>memcached</code> 操作のための検索キー値を格納する InnoDB テーブルのカラム。これは必須の値です。
<code>value_columns</code>	<code>memcached</code> データを格納する InnoDB テーブルのカラム (1 つ以上)。 <code>innodb_memcached_config_options</code> テーブルで指定されたセパレータ文字を使用して、複数のカラムを指定できます。デフォルトでは、区切り文字はパイプ文字 (<code> </code>) です。複数カラムを指定するには、定義された区切り文字でカラムを区切ります。たとえば、 <code>col1 col2 col3</code> となります。これは必須の値です。
<code>flags</code>	<code>memcached</code> のフラグ (メイン値とともに格納および取得されるユーザー定義の数値) として使用される

カラム	説明
	InnoDB テーブルのカラム。 <code>memcached</code> 値が複数のカラムにマップされている場合、フラグ値を一部の操作 (<code>incr</code> 、 <code>prepend</code> など) のカラム指定子として使用して、指定したカラムに対して操作を実行できます。たとえば、 <code>value_columns</code> を 3 つの InnoDB テーブルのカラムにマップし、一方のカラムに対してのみ増分操作を実行する場合は、 <code>flags</code> カラムを使用してカラムを指定します。 <code>flags</code> カラムを使用しない場合は、未使用であることを示す 0 の値を設定します。
<code>cas_column</code>	比較およびスワップ (<code>cas</code>) 値を格納する InnoDB テーブルのカラム。 <code>cas_column</code> の値は、 <code>memcached</code> が異なるサーバーにリクエストをハッシュし、データをメモリーにキャッシュする方法に関連しています。 InnoDB <code>memcached</code> プラグインは単一の <code>memcached</code> デーモンと緊密に統合されており、インメモリーキャッシュメカニズムは MySQL および InnoDB <code>buffer pool</code> によって処理されるため、このカラムはほとんど必要ありません。このカラムを使用しない場合は、0 の値を設定して未使用であることを示します。
<code>expire_time_column</code>	有効期限の値を格納する InnoDB テーブルのカラム。 <code>expire_time_column</code> の値は、 <code>memcached</code> が異なるサーバーにリクエストをハッシュし、データをメモリーにキャッシュする方法に関連しています。 InnoDB <code>memcached</code> プラグインは単一の <code>memcached</code> デーモンと緊密に統合されており、インメモリーキャッシュメカニズムは MySQL および InnoDB <code>buffer pool</code> によって処理されるため、このカラムはほとんど必要ありません。このカラムを使用しない場合は、0 の値を設定して、カラムが未使用であることを示します。最大有効期限は、 <code>INT_MAX32</code> または 2147483647 秒 (約 68 年) として定義されます。
<code>unique_idx_name_on_key</code>	キーカラムのインデックスの名前。これは一意のインデックスである必要があります。これは主キーまたはセカンダリインデックスにできます。できれば、InnoDB テーブルの主キーを使用してください。主キーを使用すると、セカンダリインデックスの使用時に実行されるロックアップが回避されます。 <code>memcached</code> 参照のためのカバリングインデックスは作成できません。キーカラムおよび値カラムの両方に複合セカンダリインデックスを定義しようとすると、InnoDB はエラーを返します。

containers テーブルカラムの制約

- `db_schema`、`db_name`、`key_columns`、`value_columns`、および `unique_idx_name_on_key` の値を指定する必要があります。 `flags`、`cas_column`、および `expire_time_column` が使用されない場合、これらに 0 を指定します。 そうしないと、セットアップが失敗する場合があります。
- `key_columns`: `memcached` で強制される、`memcached` キーの最大長は 250 文字です。 マップ済みのキーは、Null 以外の `CHAR` または `VARCHAR` タイプである必要があります。
- `value_columns`: `CHAR`、`VARCHAR`、または `BLOB` カラムにマップされる必要があります。 長さに制約はなく、値を NULL に指定できます。
- `cas_column`: `cas` 値は 64 ビットの整数です。 これは少なくとも 8 バイトの `BIGINT` にマップされる必要があります。 このカラムを使用しない場合は、0 の値を設定して未使用であることを示します。
- `expiration_time_column`: 少なくとも 4 バイトの `INTEGER` にマップされる必要があります。 有効期限は、Unix 時間の 32 ビット整数 (1970 年 1 月 1 日からの秒数の 32 ビット値) として、または現在時間から開始する秒数として定

義されます。後者の場合、秒数は $60 \times 60 \times 24 \times 30$ (30 日間の秒数) を超えないようにしてください。クライアントから送信される数が多い場合、サーバーはそれを現在の時間からのオフセットではなく、実際の Unix 時間値とみなします。このカラムを使用しない場合は、0 の値を設定して未使用であることを示します。

- **flags**: 少なくとも 32 ビットの **INTEGER** にマップする必要があり、NULL に指定できます。このカラムを使用しない場合は、0 の値を設定して未使用であることを示します。

カラム制約を強制するために、プラグインのロード時に事前検査が行われます。不一致が見つかった場合、プラグインはロードされません。

複数値カラムマッピング

- プラグインの初期化時に、InnoDB memcached が **containers** テーブルで定義された情報で構成されている場合、**containers.value_columns** で定義された各マップ済カラムは、マップ済 InnoDB テーブルに対して検証されます。複数の InnoDB テーブルのカラムがマップされている場合は、各カラムが存在し、正しい型であることを確認するチェックがあります。
- 実行時に、**memcached** の挿入操作では、マップされたカラムの数よりもデリミタ付きの値が多い場合、マップされた値の数のみが取得されます。たとえば、マッピングされたカラムが 6 つあり、7 つの区切り値が指定されている場合、最初の 6 つの区切り値のみが使用されます。7 番目の区切り値は無視されます。
- マップされたカラムより区切られた値の方が少ない場合、入力値のないカラムは NULL に設定されます。未入力のカラムを NULL に設定できない場合、挿入操作は失敗します。
- テーブルにマップされた値より多くのカラムがある場合、余分なカラムは結果に影響しません。

demo_test のサンプルテーブル

innodb_memcached_config.sql 構成スクリプトにより、**test** データベースに **demo_test** テーブルが作成され、これを使用して、設定後すぐに InnoDB memcached プラグインのインストールを検証できます。

innodb_memcached_config.sql 構成スクリプトでは、**innodb_memcache.containers** テーブルに **demo_test** テーブルのエントリも作成されます。

```
mysql> SELECT * FROM innodb_memcache.containers\G
***** 1. row *****
      name: aaa
     db_schema: test
      db_table: demo_test
   key_columns: c1
  value_columns: c2
         flags: c3
     cas_column: c4
  expire_time_column: c5
unique_idx_name_on_key: PRIMARY

mysql> SELECT * FROM test.demo_test;
+-----+-----+-----+
| c1 | c2 | c3 | c4 | c5 |
+-----+-----+-----+
| AA | HELLO, HELLO | 8 | 0 | 0 |
+-----+-----+-----+
```

15.20.9 InnoDB memcached プラグインのトラブルシューティング

このセクションでは、InnoDB memcached プラグインの使用時に発生する可能性のある問題について説明します。

- MySQL エラーログで次のエラーが発生した場合、サーバーの起動に失敗する可能性があります:

開いているファイルの **rlimit** を設定する **failed**。root として実行するか、小さい **maxconns value** をリクエストしてください

エラーメッセージは、**memcached** デーモンからのものです。1 つの解決策は、開くファイルの数について OS での制限を引き上げることです。オープンファイル制限をチェックして増やすためのコマンドは、オペレーティングシステムによって異なります。この例は、Linux および macOS のコマンドを示しています:

```
# Linux
```



```
shell> ulimit -n
1024
shell> ulimit -n 4096
shell> ulimit -n
4096

# macOS
shell> ulimit -n
256
shell> ulimit -n 4096
shell> ulimit -n
4096
```

もう 1 つの解決策は、`memcached` デーモンで許可される同時接続の数を減らすことです。これを行うには、MySQL 構成ファイルの `daemon_memcached_option` 構成パラメータで `-c memcached` オプションをエンコードします。`-c` オプションのデフォルト値は 1024 です。

```
[mysqld]
...
loose-daemon_memcached_option='-c 64'
```

- `memcached` デーモンが InnoDB テーブルデータを格納または取得できない問題をトラブルシューティングするには、MySQL 構成ファイルの `daemon_memcached_option` 構成パラメータで `-vvv memcached` オプションをエンコードします。MySQL エラーログを調べて、`memcached` 操作に関するデバッグ出力がないか検査します。

```
[mysqld]
...
loose-daemon_memcached_option='-vvv'
```

- `memcached` 値を保持するために指定されたカラムのデータ型が間違っている場合 (文字列型ではなく数値型など)、キーと値のペアを格納しようとするとう失敗し、特定のエラーコードまたはメッセージが表示されません。
- `daemon_memcached` プラグインによって MySQL サーバーの起動の問題が発生した場合は、MySQL 構成ファイルの `[mysqld]` グループの下に次の行を追加することで、トラブルシューティング中に `daemon_memcached` プラグインを一時的に無効にできます:

```
daemon_memcached=OFF
```

たとえば、`innodb_memcached_config.sql` 構成スクリプトを実行して必要なデータベースとテーブルを設定する前に `INSTALL PLUGIN` ステートメントを実行すると、サーバーが予期せず終了し、起動に失敗することがあります。`innodb_memcache.containers` テーブルのエントリを誤って構成すると、サーバーの起動に失敗する可能性もあります。

MySQL インスタンスの `memcached` プラグインをアンインストールするには、次のステートメントを発行します:

```
mysql> UNINSTALL PLUGIN daemon_memcached;
```

- 各インスタンスで `daemon_memcached` プラグインが有効になっている同じマシンで MySQL の複数のインスタンスを実行する場合は、`daemon_memcached_option` 構成パラメータを使用して、`daemon_memcached` プラグインごとに一意の `memcached` ポートを指定します。
- SQL ステートメントで InnoDB テーブルが見つからない場合、またはテーブルにデータが見つからず、`memcached` API コールに必要なデータが取得される場合、`innodb_memcache.containers` テーブルの InnoDB テーブルのエントリが欠落しているか、`@@table_id` テーブル記法を使用して `get` または `set` リクエストを発行して正しい InnoDB テーブルに切り替えられていない可能性があります。この問題は、後で MySQL サーバーを再起動せずに `innodb_memcache.containers` テーブルの既存のエントリを変更した場合にも発生する可能性があります。フリーフォーム記憶域メカニズムは柔軟性があるため、デーモンが単一のカラムに値を格納する `test.demo_test` テーブルを使用している場合でも、`col1|col2|col3` などの複数カラム値を格納または取得するリクエストは引き続き機能します。
- `daemon_memcached` プラグインで使用する独自の InnoDB テーブルを定義し、テーブルのカラムが `NOT NULL` として定義されている場合は、`innodb_memcache.containers` テーブルにテーブルのレコードを挿入するときに `NOT NULL` カラムに値が指定されていることを確認します。`innodb_memcache.containers` レコードの `INSERT` ステートメントに含まれるデリミタ付きの値が、マップされたカラムより少ない場合、未入力のカラムは `NULL` に設定されます。`NULL` 値を `NOT NULL` カラムに挿入しようとする `INSERT` が失敗し、`daemon_memcached` プラグインを再初期化して `innodb_memcache.containers` テーブルに変更を適用した後にのみ明らかになる場合があります。

- `innodb_memcached.containers` テーブルの `cas_column` および `expire_time_column` フィールドが `NULL` に設定されている場合、`memcached` プラグインをロードしようとすると次のエラーが返されます:

```
InnoDB_Memcached: column 6 in the entry for config table 'containers' in
database 'innodb_memcache' has an invalid NULL value.
```

`memcached` プラグインは、`cas_column` および `expire_time_column` カラムでの `NULL` の使用を拒否します。カラムが使用されていない場合は、これらのカラムの値を `0` に設定します。

- `memcached` のキーと値の長さが増加するにつれて、サイズと長さの制限が生じる場合があります。
 - キーが 250 バイトを超えると、`memcached` 操作はエラーを返します。これは `memcached` 内での現在の固定制限値です。
 - InnoDB テーブルの制限は、サイズが 768 バイト、サイズが 3072 バイトまたは `innodb_page_size` 値の半分を超える場合に発生することがあります。これらの制限は主に、SQL を使用して値カラムにインデックスを作成し、そのカラムに対してレポート生成クエリを実行する場合に適用されます。詳細は、[セクション15.22「InnoDBの制限」](#) を参照してください。
 - キーと値の組合せの最大サイズは 1 MB です。
- 異なるバージョンの MySQL サーバー間で構成ファイルを共有する場合、`daemon_memcached` プラグインの最新の構成オプションを使用すると、古い MySQL バージョンで起動エラーが発生する可能性があります。互換性の問題を回避するには、オプション名とともに `loose` 接頭辞を使用します。たとえば、`daemon_memcached_option='-c 64'` のかわりに `loose-daemon_memcached_option='-c 64'` を使用します。
- 文字セットの設定を検証するための制約もチェックありません。`memcached` は、キーおよび値をバイト形式で格納および取得するため、文字セットの違いは区別されません。ただし、`memcached` クライアントと MySQL テーブルで、同じ文字セットを使用する必要があります。
- `memcached` 接続は、インデックス付けされた仮想カラムを含むテーブルへのアクセスをブロックされます。インデックス付き仮想カラムにアクセスするにはサーバーへのコールバックが必要ですが、`memcached` 接続にはサーバーコードへのアクセス権がありません。

15.21 InnoDB のトラブルシューティング

InnoDB の問題のトラブルシューティングには、次の一般的なガイドラインが適用されます。

- 操作が失敗した場合、またはバグが疑われる場合は、MySQL サーバーのエラーログを参照してください ([セクション5.4.2「エラーログ」](#) を参照)。[Server Error Message Reference](#) では、発生する可能性のある InnoDB 固有の一般的なエラーのトラブルシューティング情報が提供されます。
- 障害が `deadlock` に関連している場合は、各デッドロックの詳細が MySQL サーバーのエラーログに出力されるように、`innodb_print_all_deadlocks` オプションを有効にして実行します。デッドロックの詳細は、[セクション15.7.5「InnoDBのデッドロック」](#) を参照してください。
- 問題が InnoDB データディクショナリに関連している場合は、[セクション15.21.3「InnoDB データディクショナリの操作のトラブルシューティング」](#) を参照してください。
- トラブルシューティング時は通常、`mysqld_safe` 経由、または Windows サービスとしてではなく、コマンドプロンプトから MySQL サーバーを実行することが最善です。それにより、`mysqld` がコンソールに出力する内容を確認できるため、何が発生しているかをよりの確に把握できます。Windows では、出力先がコンソールウィンドウになるように、`--console` オプションを付けて `mysqld` を起動します。
- InnoDB モニターを有効にして、問題に関する情報を取得します ([セクション15.17「InnoDB モニター」](#) を参照)。その問題がパフォーマンスに関するものか、またはサーバーがハングアップしているように見える場合は、InnoDB の内部状態に関する情報を出力するために、標準モニターを有効にするようにしてください。問題がロックに関するものである場合は、ロックモニターを有効にします。テーブルの作成、テーブルスペースまたはデータディクショナリ操作に問題がある場合は、[InnoDB Information Schema system tables](#) を参照して InnoDB 内部データディクショナリの内容を調べます。

InnoDB は、次の条件の下で InnoDB 標準モニターの出力を一時的に有効にします。

- 長いセマフォ待機

- InnoDB がバッファプール内に空きブロックを見つけることができない
- ロックヒープまたはアダプティブハッシュインデックスによってバッファプールの 67% を超える領域が占有されている
- テーブルが破損していると思われる場合は、そのテーブルに対して `CHECK TABLE` を実行します。

15.21.1 InnoDB の I/O に関する問題のトラブルシューティング

InnoDB の I/O に関する問題のトラブルシューティング手順は、その問題がいつ、つまり MySQL サーバーの起動中か、あるいは通常の動作中にファイルシステムレベルの問題で DML または DDL ステートメントが失敗したときのどちらで発生したかによって異なります。

初期化の問題

InnoDB がそのテーブルスペースまたはログファイルを初期化しようとしたときに問題が発生した場合は、InnoDB によって作成されたすべてのファイル、つまりすべての `ibdata` ファイルおよびすべての `ib_logfile` ファイルを削除します。いくつかの InnoDB テーブルをすでに作成している場合は、MySQL データベースディレクトリから `.ibd` ファイルも削除します。次に、再度 InnoDB データベースを作成してみてください。もっとも簡単なトラブルシューティングとして、何が発生しているかがわかるように、コマンドプロンプトから MySQL サーバーを起動してください。

実行時の問題

InnoDB がファイル操作中にオペレーティングシステムのエラーを出力する場合、通常、この問題には次のいずれかの解決方法があります。

- InnoDB データファイルディレクトリと InnoDB ログディレクトリが存在することを確認します。
- `mysqld` に、これらのディレクトリ内にファイルを作成するためのアクセス権があることを確認します。
- 指定したオプションで起動できるように、`mysqld` が正しい `my.cnf` または `my.ini` オプションファイルを読み取れることを確認します。
- ディスクがいっぱいでなく、かつどのディスク割り当て制限も超えていないことを確認します。
- サブディレクトリとデータファイルに指定した名前が衝突していないことを確認します。
- `innodb_data_home_dir` および `innodb_data_file_path` 値の構文を再確認します。特に、`innodb_data_file_path` オプション内の `MAX` 値はすべて強い制限値であるため、その制限を超えると致命的エラーが発生します。

15.21.2 InnoDB のリカバリの強制的な実行

データベースページの破損を調査するために、`SELECT ... INTO OUTFILE` を使用して、データベースからテーブルをダンプできます。通常は、この方法で取得されたデータのほとんどが完全な状態にあります。重大な破損により、`SELECT * FROM tbl_name` ステートメントまたは InnoDB バックグラウンド操作が予期せず終了またはアサートされたり、InnoDB ロールフォワードリカバリがクラッシュする可能性があります。このような場合は、テーブルをダンプできるように、`innodb_force_recovery` オプションを使用して、バックグラウンド操作が実行されないようにして InnoDB ストレージエンジンを強制的に起動させることができます。たとえば、サーバーを再起動する前に、オプションファイルの `[mysqld]` セクションに次の行を追加できます。

```
[mysqld]
innodb_force_recovery = 1
```

オプションファイルの使用の詳細は、[セクション4.2.2.2「オプションファイルの使用」](#)を参照してください。

警告

`innodb_force_recovery` を 0 を超える値に設定するのは、緊急の状況で InnoDB を起動し、テーブルをダンプできるようにする場合だけにしてください。それを行う前に、データベースの再作成が必要になった場合に備えて、データベースのバックアップコピーがあることを確認してください。4 以上の値を指定すると、データファイルが永続的に破損する場合があります。本番サーバーインスタンスで 4 以上の `innodb_force_recovery` 設定を使用するの

は、データベースの個別の物理コピーで設定を正常にテストした後のみです。InnoDB のリカバ리를強制的に実行する場合は、常に `innodb_force_recovery=1` から始め、必要がある場合にのみこの値を 1 ずつ増やすようにしてください。

`innodb_force_recovery` は、デフォルトでは 0 です (リカバリが強制的に実行されない通常の起動)。

`innodb_force_recovery` の許可される 0 以外の値は 1 から 6 までです。大きい方の値には、小さい方の値の機能が含まれています。たとえば、3 の値には、値 1 と 2 のすべての機能が含まれています。

3 以下の `innodb_force_recovery` 値を使用してテーブルをダンプできる場合は、破損した個々のページ上の一部のデータしか失われなため、比較的安全です。4 以上の値は、データファイルが永続的に破損する場合があるため、危険であるとみなされます。値 6 は、データベースページが廃止された状態のままであり、B-trees およびその他のデータベース構造が破損する可能性があるため、劇的とみなされます。

安全策として、`innodb_force_recovery` が 0 より大きい場合、InnoDB は INSERT、UPDATE、または DELETE 操作を回避します。`innodb_force_recovery` 設定が 4 以上の場合、InnoDB は読取り専用モードになります。

- 1 (SRV_FORCE_IGNORE_CORRUPT)

破損したページを検出した場合でも、サーバーが動作できるようにします。SELECT * FROM tbl_name での破損したインデックスレコードおよびページの飛び越しを試行します。これが、テーブルのダンプに役立ちます。

- 2 (SRV_FORCE_NO_BACKGROUND)

マスタースレッドや、すべてのパージスレッドが実行されないようにします。purge 操作中に予期しない終了が発生した場合、このリカバリ値によってそれが防止されます。

- 3 (SRV_FORCE_NO_TRX_UNDO)

クラッシュリカバリのあとにトランザクションロールバックを実行しません。

- 4 (SRV_FORCE_NO_IBUF_MERGE)

挿入バッファのマージ操作を回避します。その操作によってクラッシュが発生しそうになった場合は、それが回避されます。テーブル統計を計算しません。この値を指定すると、データファイルが永続的に破損する場合があります。この値を使用したあと、すべてのセカンダリインデックスを削除して再作成するように準備してください。InnoDB を読取り専用を設定します。

- 5 (SRV_FORCE_NO_UNDO_LOG_SCAN)

データベースを起動するときに、Undo ログを参照しません。InnoDB は、未完了のトランザクションでさえコミット済みとして処理します。この値を指定すると、データファイルが永続的に破損する場合があります。InnoDB を読取り専用を設定します。

- 6 (SRV_FORCE_NO_LOG_REDO)

リカバリに関連した Redo ログのロールフォワードを実行しません。この値を指定すると、データファイルが永続的に破損する場合があります。データベースページを廃止された状態のままにし、それによって B ツリーやその他のデータベース構造にさらに多くの破損が発生する可能性があります。InnoDB を読取り専用を設定します。

テーブルから SELECT を使用してダンプできます。`innodb_force_recovery` 値が 3 以下の場合、DROP テーブルまたは CREATE テーブルを使用できます。DROP TABLE は、3 より大きい `innodb_force_recovery` 値でもサポートされています。`innodb_force_recovery` 値が 4 より大きい場合、DROP TABLE は許可されません。

特定のテーブルがロールバック時に予期しない終了を引き起こしていることがわかっている場合は、そのテーブルを削除できます。失敗した大量のインポートまたは ALTER TABLE によってロールバックの暴走が発生する場合は、mysqld プロセスを強制終了し、`innodb_force_recovery` を 3 に設定してロールバックなしでデータベースを起動したあと、ロールバックの暴走の原因になっているテーブルの DROP を実行することができます。

テーブルデータ内の破損のためにテーブルの内容全体をダンプできない場合は、ORDER BY primary_key DESC 句を含むクエリで、破損した部分のあとにあるテーブルの部分の部分をダンプできる可能性があります。

InnoDB を起動するために `innodb_force_recovery` を大きな値にする必要がある場合は、複雑なクエリ (WHERE、ORDER BY、またはその他の句を含むクエリ) を失敗させることがある破損したデータ構造が存在する可能性があります。この場合は、基本的な SELECT * FROM t クエリしか実行できない可能性があります。

15.21.3 InnoDB データディクショナリの操作のトラブルシューティング

テーブル定義に関する情報は、InnoDB [data dictionary](#) に格納されます。データファイルを移動すると、ディクショナリデータに一貫性がなくなる可能性があります。

データディクショナリの破損や一貫性の問題によって InnoDB を起動できない場合は、手動のリカバリに関する情報について、[セクション15.21.2「InnoDB のリカバリの強制的な実行」](#)を参照してください。

データファイルを開けません

[innodb_file_per_table](#) が有効な場合 (デフォルト)、[file-per-table](#) テーブルスペースファイル (.ibd ファイル) が欠落していると、起動時に次のメッセージが表示されることがあります:

```
[ERROR] InnoDB: Operating system error number 2 in a file operation.
[ERROR] InnoDB: The error means the system cannot find the path specified.
[ERROR] InnoDB: Cannot open datafile for read-only: './test/t1.ibd' OS error: 71
[Warning] InnoDB: Ignoring tablespace `test/t1` because it could not be opened.
```

これらのメッセージに対処するには、[DROP TABLE](#) ステートメントを発行して、欠落しているテーブルに関するデータをデータディクショナリから削除します。

孤立したファイル/テーブル ibd ファイルの復元

この手順では、孤立した [file-per-table .ibd](#) ファイルを別の MySQL インスタンスにリストアする方法について説明します。システムテーブルスペースが消失またはリカバリ不能で、新しい MySQL インスタンスで .ibd ファイルのバックアップをリストアする場合は、このプロシージャを使用できます。

プロシージャは、[general tablespace .ibd](#) ファイルではサポートされていません。

この手順では、.ibd ファイルのバックアップのみがあり、孤立した .ibd ファイルを最初に作成したのと同じバージョンの MySQL にリカバリしており、.ibd ファイルのバックアップがクリーンであることを前提としています。クリーンバックアップの作成の詳細は、[セクション15.6.1.4「InnoDB テーブルの移動またはコピー」](#)を参照してください。

この手順には、[セクション15.6.1.3「InnoDB テーブルのインポート」](#)で概説されているテーブルのインポート制限が適用されます。

1. 新しい MySQL インスタンスで、同じ名前のデータベースにテーブルを再作成します。

```
mysql> CREATE DATABASE sakila;

mysql> USE sakila;

mysql> CREATE TABLE actor (
  actor_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
  first_name VARCHAR(45) NOT NULL,
  last_name VARCHAR(45) NOT NULL,
  last_update TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (actor_id),
  KEY idx_actor_last_name (last_name)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

2. 新しく作成したテーブルのテーブルスペースを破棄します。

```
mysql> ALTER TABLE sakila.actor DISCARD TABLESPACE;
```

3. 孤立した .ibd ファイルをバックアップディレクトリから新しいデータベースディレクトリにコピーします。

```
shell> cp /backup_directory/actor.ibd path/to/mysql-5.7/data/sakila/
```

4. .ibd ファイルに必要なファイル権限があることを確認します。
5. 孤立した .ibd ファイルをインポートします。InnoDB がスキーマ検証なしでファイルをインポートしようとしていることを示す警告が発行されます。

```
mysql> ALTER TABLE sakila.actor IMPORT TABLESPACE; SHOW WARNINGS;
Query OK, 0 rows affected, 1 warning (0.15 sec)
```



```
Warning | 1810 | InnoDB: IO Read error: (2, No such file or directory)
Error opening './sakila/actor.cfg', will attempt to import
without schema verification
```

6. テーブルをクエリーして、`.ibd` ファイルが正常にリストアされたことを確認します。

```
mysql> SELECT COUNT(*) FROM sakila.actor;
+-----+
| count(*) |
+-----+
|      200 |
+-----+
```

15.21.4 InnoDB のエラー処理

次の項目では、**InnoDB** がエラー処理を実行する方法について説明します。**InnoDB** では、失敗したステートメントのみがロールバックされ、それ以外の場合はトランザクション全体がロールバックされることがあります。

- **テーブルスペース**内のファイル領域が不足した場合は、MySQL の **Table is full** エラーが発生し、**InnoDB** は SQL ステートメントをロールバックします。
- トランザクション**デッドロック**が発生すると、**InnoDB** は**トランザクション**全体を**ロールバック**します。これが発生したら、トランザクション全体を再試行します。

ロック待機タイムアウトが発生すると、**InnoDB** は現在のステートメント (ロックを待機していてタイムアウトが発生したステートメント) をロールバックします。トランザクション全体をロールバックするには、`--innodb-rollback-on-timeout` を有効にしてサーバーを起動します。デフォルトの動作を使用する場合はステートメントを再試行し、`--innodb-rollback-on-timeout` が有効な場合はトランザクション全体を再試行します。

デッドロックとロック待機のタイムアウトはどちらもビジー状態のサーバーでは通常のことであり、アプリケーションはそれらが発生する可能性を認識し、発生した場合は再試行によって処理する必要があります。トランザクション中の最初のデータ変更からコミットまでの間に行う作業をできるだけ少なくして、ロックが可能性のある最短の時間、可能性のある最少の行数に対して保持されるようにすることにより、それらが発生する可能性を少なくすることができます。場合によっては、異なるトランザクション間での作業の分割が実際的で、かつ役立つことがあります。

- ステートメントで **IGNORE** オプションを指定していない場合、重複キーエラーは SQL ステートメントをロールバックします。
- **row too long error** は、SQL ステートメントをロールバックします。
- その他のエラーはほとんど (**InnoDB** ストレージエンジンレベルの上にある) コードの MySQL レイヤーによって検出され、対応する SQL ステートメントをロールバックします。1 つの SQL ステートメントのロールバックでは、ロックは解放されません。

暗黙的なロールバック中や、明示的な **ROLLBACK** SQL ステートメントの実行中に、**SHOW PROCESSLIST** は、関連する接続の **State** カラムに **Rolling back** を表示します。

15.22 InnoDB の制限

このセクションでは、**InnoDB** ストレージエンジンの **InnoDB** テーブル、インデックス、テーブルスペース、およびその他の側面の制限について説明します。

- テーブルには、最大 1017 カラムを含めることができます。仮想生成カラムはこの制限に含まれます。
- テーブルには、最大で 64 個の**セカンダリインデックス**を含めることができます。
- **DYNAMIC** または **COMPRESSED** の行形式を使用する **InnoDB** テーブルでは、インデックスキーの接頭辞の長さの制限は 3072 バイトです。

REDUNDANT または **COMPACT** の行形式を使用する **InnoDB** テーブルのインデックスキー接頭辞の長さ制限は 767 バイトです。たとえば、**utf8mb4** 文字セットおよび各文字の最大 4 バイトを想定して、**TEXT** または **VARCHAR** カラムで 191 文字を超える **column prefix** インデックスを使用して、この制限に達する場合があります。

制限を超えるインデックスキー接頭辞の長さを使用しようとすると、エラーが返されます。

MySQL インスタンスの作成時に `innodb_page_size` オプションを指定して、InnoDB のページサイズを 8K バイトまたは 4K バイトまで小さくすると、16K バイトのページサイズに対応する 3072 バイトの制限に基づいて、比例的にインデックスキーの最大長も短くなります。つまり、インデックスキーの最大長は、ページサイズが 8K バイトのときは 1536 バイト、ページサイズが 4K バイトのときは 768 バイトになります。

インデックスキー接頭辞に適用される制限は、フルカラムインデックスキーにも適用されます。

- 複数カラムインデックスには最大 16 カラムを使用できます。制限を超えると、エラーが返されます。

```
ERROR 1070 (42000): Too many key parts specified; max 16 parts allowed
```

- オフページに格納されている可変長カラムを除く最大行サイズは、4K バイト、8K バイト、16K バイト、および 32K バイトのページサイズではページの半分よりわずかに小さくなります。たとえば、デフォルトの `innodb_page_size` 16KB の最大行サイズは約 8000 バイトです。ただし、64KB の InnoDB ページサイズの場合、最大行サイズは約 16000 バイトです。LONGBLOB および LONGTEXT のカラムは 4GB 未満である必要があり、BLOB および TEXT のカラムを含む合計行サイズは 4GB 未満である必要があります。

行の長さが 1 ページの半分より短い場合は、行全体がそのページ内にローカルに格納されます。セクション 15.11.2 「ファイル領域管理」で説明したように、半ページを超える行では、その行が半ページ以内に収まるように、可変長カラムが外部オフページストレージの対象として選択されます。

- InnoDB では内部的に 65,535 バイトを超える行サイズがサポートされませんが、MySQL 自体では、すべてのカラムの合計サイズに 65,535 の行サイズ制限が課されます。セクション 8.4.7 「テーブルカラム数と行サイズの制限」を参照してください。
- 一部の古いオペレーティングシステムでは、ファイルは 2G バイトよりも小さくする必要があります。これは InnoDB の制限ではありません。大規模なシステムテーブルスペースが必要な場合は、1 つの大規模なデータファイルではなく複数の小規模なデータファイルを使用して構成するか、file-per-table および一般的なテーブルスペースデータファイルにテーブルデータを分散します。
- InnoDB ログファイルの最大サイズの合計は 512GB です。
- テーブルスペースの最小サイズは、10M バイトをわずかに超える大きさです。テーブルスペースの最大サイズは、InnoDB のページサイズによって異なります。

表 15.31 InnoDB テーブルスペースの最大サイズ

InnoDB ページサイズ	最大テーブルスペースサイズ
4KB	16TB
8KB	32TB
16KB	64TB
32KB	128TB
64KB	256TB

最大テーブルスペースサイズは、テーブルの最大サイズでもあります。

- ファイル名を含むテーブルスペースファイルのパスは、Windows での `MAX_PATH` 制限を超えることはできません。Windows 10 より前では、`MAX_PATH` の制限は 260 文字です。Windows 10 バージョン 1607 では、`MAX_PATH` の制限は共通の Win32 ファイルおよびディレクトリ機能から削除されていますが、新しい動作を有効にする必要があります。
- 同時読取り / 書き込みトランザクションに関連する制限については、セクション 15.6.6 「undo ログ」を参照してください。

15.23 InnoDB の制限および制限事項

このセクションでは、InnoDB ストレージエンジンの制限事項と制限事項について説明します。

- 内部 InnoDB カラム (`DB_ROW_ID`、`DB_TRX_ID` および `DB_ROLL_PTR` を含む) の名前と一致するカラム名を持つテーブルは作成できません。この制限は、任意の大文字と小文字での名前の使用に適用されます。

```
mysql> CREATE TABLE t1 (c1 INT, db_row_id INT) ENGINE=INNODB;  
ERROR 1166 (42000): Incorrect column name 'db_row_id'
```

- `SHOW TABLE STATUS` では、テーブルで予約されている物理サイズを除き、InnoDB テーブルの正確な統計は提供されません。行カウントは、単に SQL 最適化で使用される概算見積もりです。
- 並列トランザクションでは同時にさまざまな数の行が「参照」される可能性があるため、InnoDB のテーブルには、行の内部的なカウントが保持されません。したがって、`SELECT COUNT(*)` ステートメントでは、現在のトランザクションで参照可能な行のみがカウントされます。

InnoDB による `SELECT COUNT(*)` ステートメントの処理方法の詳細は、[セクション12.20.1「集計関数の説明」](#) の `COUNT()` の説明を参照してください。

- `ROW_FORMAT=COMPRESSED` は、16KB を超えるページサイズではサポートされていません。
- 特定の InnoDB ページサイズ (`innodb_page_size`) を使用する MySQL インスタンスでは、異なるページサイズを使用するインスタンスのデータファイルまたはログファイルを使用できません。
- トランスポータブルテーブルスペース機能を使用したテーブルのインポートに関連する制限については、[Table Import Limitations](#) を参照してください。
- オンライン DDL に関連する制限については、[セクション15.12.6「オンライン DDL の制限事項」](#) を参照してください。
- 一般的なテーブルスペースに関連する制限については、[テーブルスペースの一般的な制限事項](#) を参照してください。
- 保存データ暗号化に関連する制限については、[暗号化の制限事項](#) を参照してください。

第 16 章 代替ストレージエンジン

目次

16.1 ストレージエンジンの設定	2986
16.2 MyISAM ストレージエンジン	2987
16.2.1 MyISAM 起動オプション	2989
16.2.2 キーに必要な容量	2991
16.2.3 MyISAM テーブルのストレージフォーマット	2991
16.2.4 MyISAM テーブルの問題点	2994
16.3 MEMORY ストレージエンジン	2995
16.4 CSV ストレージエンジン	2999
16.4.1 CSV テーブルの修復と確認	3000
16.4.2 CSV の制限	3000
16.5 ARCHIVE ストレージエンジン	3001
16.6 BLACKHOLE ストレージエンジン	3002
16.7 MERGE ストレージエンジン	3004
16.7.1 MERGE テーブルの長所と短所	3007
16.7.2 MERGE テーブルの問題点	3008
16.8 FEDERATED ストレージエンジン	3009
16.8.1 FEDERATED ストレージエンジンの概要	3009
16.8.2 FEDERATED テーブルの作成方法	3010
16.8.3 FEDERATED ストレージエンジンの注記とヒント	3013
16.8.4 FEDERATED ストレージエンジンのリソース	3014
16.9 EXAMPLE ストレージエンジン	3014
16.10 ほかのストレージエンジン	3015
16.11 MySQL ストレージエンジンアーキテクチャーの概要	3015
16.11.1 プラガブルストレージエンジンのアーキテクチャー	3016
16.11.2 共通データベースサーバーレイヤー	3016

ストレージエンジンは、さまざまなテーブル型に対する SQL 操作を処理する MySQL コンポーネントです。InnoDB はデフォルトでもっとも汎用のストレージエンジンであり、Oracle は、特別なユースケースを除くテーブルについては、このエンジンの使用を推奨します。(デフォルトでは、MySQL 8.0 の `CREATE TABLE` ステートメントは InnoDB テーブルを作成します。)

MySQL Server は、ストレージエンジンが、動作中の MySQL サーバーにロードされたり、MySQL サーバーからアンロードされたりできる、プラガブルストレージエンジンアーキテクチャーを採用しています。

ご使用のサーバーがサポートするストレージエンジンを調べるには、`SHOW ENGINES` ステートメントを使用します。サポートカラムの値は、エンジンを使用できるかどうかを示します。YES、NO、または DEFAULT の値は、エンジンが「使用可能」、「使用可能でない」、または「デフォルトのストレージエンジンとして使用可能であり、現在設定されている」を表しています。

```
mysql> SHOW ENGINES\G
***** 1. row *****
  Engine: PERFORMANCE_SCHEMA
  Support: YES
  Comment: Performance Schema
  Transactions: NO
    XA: NO
  Savepoints: NO
***** 2. row *****
  Engine: InnoDB
  Support: DEFAULT
  Comment: Supports transactions, row-level locking, and foreign keys
  Transactions: YES
    XA: YES
  Savepoints: YES
***** 3. row *****
  Engine: MRG_MYISAM
```

```
Support: YES
Comment: Collection of identical MyISAM tables
Transactions: NO
  XA: NO
Savepoints: NO
***** 4. row *****
  Engine: BLACKHOLE
  Support: YES
  Comment: /dev/null storage engine (anything you write to it disappears)
Transactions: NO
  XA: NO
  Savepoints: NO
***** 5. row *****
  Engine: MyISAM
  Support: YES
  Comment: MyISAM storage engine
Transactions: NO
  XA: NO
  Savepoints: NO
...
```

この章では、特別な目的の MySQL ストレージエンジンのユースケースについて説明します。第15章「InnoDB ストレージエンジン」および第23章「MySQL NDB Cluster 8.0」で説明するデフォルトの InnoDB ストレージエンジンまたは NDB ストレージエンジンについては説明しません。上級ユーザーの場合は、プラグインストレージエンジンアーキテクチャの説明も含まれています(セクション16.11「MySQL ストレージエンジンアーキテクチャーの概要」を参照)。

商用 MySQL Server バイナリで提供される機能の詳細は、MySQL web サイトの「MySQL エディション」を参照してください。使用可能なストレージエンジンは、使用している MySQL のエディションによって異なります。

MySQL ストレージエンジンに関するよくある質問の回答については、セクションA.2「MySQL 8.0 FAQ: ストレージエンジン」を参照してください。

MySQL 8.0 がサポートするストレージエンジン

- **InnoDB:** MySQL 8.0 のデフォルトのストレージエンジン。InnoDB はトランザクションセーフな (ACID に準拠した) MySQL 用のストレージエンジンであり、ユーザーデータを保護するためのコミット、ロールバック、およびクラッシュリカバリ機能を備えています。InnoDB の行レベルロック (より粒度の粗いロックへのエスカレーションは行わない) と Oracle スタイルの一貫性非ロック読み取りにより、マルチユーザーの並列性とパフォーマンスが向上します。InnoDB では、主キーに基づいた一般的なクエリーの入出力を低減するため、クラスタ化されたインデックス内にユーザーデータが格納されます。InnoDB ではデータの整合性を維持できるように、FOREIGN KEY 参照整合性制約もサポートされています。InnoDB の詳細については、第15章「InnoDB ストレージエンジン」を参照してください。
- **MyISAM:** これらのテーブルのフットプリントは小さくなります。テーブルレベルのロックでは、読み取り/書き込みの作業負荷でのパフォーマンスが抑えられるため、Web およびデータウェアハウス構成の読み取り専用または読み取りが大半の作業負荷の場合に使用されるのが一般的です。
- **メモリー:** すべてのデータを RAM に格納します (重要でないデータの短時間での検索が必要な環境で高速にアクセスするため)。このエンジンは以前は HEAP エンジンとして知られていました。このユースケースは減少しています。バッファプールのメモリー領域を持つ InnoDB は、ほとんどのデータまたはすべてのデータをメモリーに保持する汎用的で永続的な方法を提供し、NDBCLUSTER は大規模な分散データセットでキー値の高速な検索ができます。
- **CSV:** このテーブルは、カンマ区切り値を持つ実際のテキストファイルです。CSV テーブルにより、CSV フォーマットでデータをインポートしたりダンプしたりして、同じフォーマットを読み込んだり書き込んだりするスクリプトおよびアプリケーションとデータを交換できます。CSV テーブルはインデックス化されないため、通常の操作時はデータを InnoDB テーブルに保持し、インポートまたはエクスポートの段階でのみ CSV テーブルを使用するのが一般的です。
- **アーカイブ:** これらのインデックス化されていないコンパクトなテーブルは、ほとんど参照されない大量の履歴情報、アーカイブされた情報、またはセキュリティー監査情報を格納したり、検索したりするためのテーブルです。
- **Blackhole:** Blackhole ストレージエンジンはデータを受け付けますが、Unix /dev/null デバイスと同じように、格納しません。クエリーは常に空のセットを返します。これらのテーブルは、DML ステートメントがレプリカサー

バーに送信されるレプリケーション構成で使用できますが、ソースサーバーはデータの独自のコピーを保持しません。

- **NDB (NDBCLUSTER と呼ばれる)**: このクラスタデータベースエンジンは、可能な限り高いアップタイムと可用性を必要とするアプリケーションに特に適しています。
- **マージ**: MySQL DBA または開発者は、一連のまったく同じ **MyISAM** テーブルを論理的にグループ分けして、それらを 1 つのオブジェクトとして参照します。データウェアハウスなどの VLDB 環境に適しています。
- **Federated**: 多くの物理サーバーから 1 つの論理サーバーを作成するために別々の MySQL サーバーをリンクする機能を提供します。分散またはデータマート環境に非常に適しています。
- **例**: このエンジンは、新しいストレージエンジンの書き込みを開始する方法を示す MySQL ソースコードの例として機能します。これは、主に開発者が対象です。ストレージエンジンは何もしない「stub」です。このエンジンでテーブルを作成できますが、それらにデータを格納したり、それらからデータを取り出したりすることはできません。

サーバー全体またはスキーマ全体に同じストレージエンジンを使用するという制限はありません。いずれのテーブルにもストレージエンジンを指定できます。たとえばアプリケーションでは、**InnoDB** テーブルを使用している場合がほとんどであり、データをスプレッドシートにエクスポートするための **CSV** テーブルを 1 つ、テンポラリワークスペース用に **MEMORY** テーブルをいくつか持っています。

ストレージエンジンの選択

MySQL が提供するさまざまなストレージエンジンは、異なるユースケースで使用されることを想定して設計されています。次のテーブルに、MySQL で提供される一部のストレージエンジンの概要と、そのあとのノートを示します。

表 16.1 「ストレージエンジン機能のサマリー」

機能	MyISAM	メモリー	InnoDB	アーカイブ	NDB
B ツリーインデックス	はい	はい	はい	いいえ	いいえ
MVCC	いいえ	いいえ	はい	いいえ	いいえ
T ツリーインデックス	いいえ	いいえ	いいえ	いいえ	はい
インデックス キャッシュ	はい	N/A	はい	いいえ	はい
クラスタデータベースのサポート	いいえ	いいえ	いいえ	いいえ	はい
クラスタ化されたインデックス	いいえ	いいえ	はい	いいえ	いいえ
ストレージの制限	256TB	RAM	64TB	なし	384EB
データキャッシュ	いいえ	N/A	はい	いいえ	はい
データディクショナリ向け更新統計	はい	はい	はい	はい	はい
トランザクション	いいえ	いいえ	はい	いいえ	はい
ハッシュインデックス	いいえ	はい	いいえ (note 1)	いいえ	はい
バックアップ/ポイントインタイムリカバリ (note 2)	はい	はい	はい	はい	はい
レプリケーションのサポート (note 2)	はい	制限付き (note 3)	はい	はい	はい
ロック粒度	Table	Table	行	行	行

機能	MyISAM	メモリー	InnoDB	アーカイブ	NDB
全文検索インデックス	はい	いいえ	はい (note 4)	いいえ	いいえ
圧縮データ	はい (note 5)	いいえ	はい	はい	いいえ
地理空間インデックスのサポート	はい	いいえ	はい (note 6)	いいえ	いいえ
地理空間データ型のサポート	はい	いいえ	はい	はい	はい
外部キーのサポート	いいえ	いいえ	はい	いいえ	はい (note 7)
暗号化データ	はい (note 8)	はい (note 8)	はい (note 9)	はい (note 8)	はい (note 8)

Notes:

1. InnoDB は、アダプティブハッシュインデックス機能に対して、内部的にハッシュインデックスを利用します。
2. ストレージエンジン内ではなくサーバー内で実装されています。
3. このセクションの後半の説明を参照してください。
4. FULLTEXT インデックスに対する InnoDB サポートは、MySQL 5.6 以降で使用できます。
5. 圧縮された MyISAM テーブルがサポートされているのは、圧縮行フォーマットを使用している場合だけです。MyISAM で圧縮行フォーマットを使用するテーブルは、読み取り専用です。
6. InnoDB での地理空間インデックス付けのサポートは、MySQL 5.7 以降で使用できます。
7. 外部キーのサポートは、MySQL Cluster NDB 7.3 以降で使用できます。
8. 暗号化機能を介してサーバーに実装されます。
9. 暗号化機能を介してサーバーに実装されます。MySQL 5.7 以降では、保存データのテーブルスペース暗号化がサポートされます。

16.1 ストレージエンジンの設定

新しいテーブルを作成するときに、**ENGINE** テーブルオプションを **CREATE TABLE** ステートメントに加えることによって、どのストレージエンジンを利用するかを指定できます。

```
-- ENGINE=INNODB not needed unless you have set a different
-- default storage engine.
CREATE TABLE t1 (i INT) ENGINE = INNODB;
-- Simple table definitions can be switched from one to another.
CREATE TABLE t2 (i INT) ENGINE = CSV;
CREATE TABLE t3 (i INT) ENGINE = MEMORY;
```

ENGINE オプションを省略した場合、デフォルトのストレージエンジンが使用されます。デフォルトのエンジンは MySQL 8.0 の **InnoDB** です。デフォルトのエンジンを指定するには、**--default-storage-engine** サーバースタートアップオプションを使用するか、**my.cnf** 構成ファイルにある **default-storage-engine** オプションを設定するかします。

現在のセッションにデフォルトのストレージエンジンを設定するには、**default_storage_engine** 変数を設定します。

```
SET default_storage_engine=NDBCLUSTER;
```

CREATE TEMPORARY TABLE で作成された **TEMPORARY** テーブルのストレージエンジンは、起動時または実行時に **default_tmp_storage_engine** を設定することによって、永続テーブルのエンジンとは別に設定できます。

テーブルを別のストレージエンジンに変換するには、新しいエンジンを指定する **ALTER TABLE** ステートメントを使用します。


```
ALTER TABLE t ENGINE = InnoDB;
```

セクション13.1.20「CREATE TABLE ステートメント」およびセクション13.1.9「ALTER TABLE ステートメント」を参照してください。

コンパイルされていないストレージエンジン、またはコンパイルされているが無効化されたストレージエンジンを使用する場合、MySQL はその代わりに、デフォルトのストレージエンジンを使用してテーブルを作成します。たとえば、レプリケーション設定では、おそらくソースサーバーは安全性を最大化するために InnoDB テーブルを使用しますが、レプリカサーバーは永続性または同時実行性を犠牲にして速度を高めるために他のストレージエンジンを使用します。

デフォルトでは、CREATE TABLE または ALTER TABLE がデフォルトのストレージエンジンを使用できない場合は、常に警告が生成されます。目的のエンジンが使用できない場合に、混乱を起こす意図しない動作をしないようにするには、NO_ENGINE_SUBSTITUTION SQL モードを有効にします。目的のエンジンが使用できない場合、この設定によって、警告の代わりにエラーが起り、テーブルが作成されたり変更されたりしません。セクション 5.1.11「サーバー SQL モード」を参照してください。

MySQL では、ストレージエンジンに応じて、テーブルのインデックスおよびデータを他の 1 つ以上のファイルに格納できます。テーブルおよびカラムの定義は、MySQL データディクショナリに格納されます。個々のストレージエンジンは、それらが管理するテーブルに必要なファイルをさらに作成します。テーブル名に特殊文字が含まれている場合は、セクション9.2.4「識別子とファイル名のマッピング」で説明されているように、その文字のエンコードされたバージョンがテーブルファイルの名前に含まれます。

16.2 MyISAM ストレージエンジン

MyISAM は古い (そしてすでに使用できない) ISAM ストレージエンジンに基づいていますが、多くの役に立つ拡張機能を持っています。

表 16.2 「MyISAM ストレージエンジンの機能」

機能	Support
B ツリーインデックス	はい
MVCC	いいえ
T ツリーインデックス	いいえ
インデックスキャッシュ	はい
クラスタデータベースのサポート	いいえ
クラスタ化されたインデックス	いいえ
ストレージの制限	256TB
データキャッシュ	いいえ
データディクショナリ向け更新統計	はい
トランザクション	いいえ
ハッシュインデックス	いいえ
バックアップ/ポイントインタイムリカバリ (ストレージエンジン内ではなくサーバー内で実装されています。)	はい
レプリケーションのサポート (ストレージエンジン内ではなくサーバー内で実装されています。)	はい
ロック粒度	Table
全文検索インデックス	はい
圧縮データ	はい (圧縮された MyISAM テーブルがサポートされているのは、圧縮行フォーマットを使用している場合だけです。MyISAM で圧縮行フォーマットを使用するテーブルは、読み取り専用です。)
地理空間インデックスのサポート	はい

機能	Support
地理空間データ型のサポート	はい
外部キーのサポート	いいえ
暗号化データ	はい (暗号化機能を介してサーバーに実装されます。)

各 MyISAM テーブルは、ディスク上の 2 つのファイルに格納されます。そのファイル名はテーブル名で始まり、ファイルタイプを示す拡張子が付きます。データファイルには `.MYD (MYData)` 拡張子が付きます。インデックスファイルには `.MYI (MYIndex)` 拡張子が付きます。テーブル定義は、MySQL データディクショナリに格納されます。

MyISAM テーブルが必要であることを明示的に指定するには、`ENGINE` テーブルオプションで指定します。

```
CREATE TABLE t (i INT) ENGINE = MYISAM;
```

MySQL 8.0 では通常、`InnoDB` がデフォルトエンジンであるため、`ENGINE` を使用して MyISAM ストレージエンジンを指定する必要があります。

`mysqlcheck` クライアントが `myisamchk` コーティリティーで MyISAM テーブルをチェックしたり修正したりできます。容量を節約するために `myisampack` を使って MyISAM テーブルを圧縮することもできます。セクション 4.5.3 「`mysqlcheck` — テーブル保守プログラム」、セクション 4.6.4 「`myisamchk` — MyISAM テーブルメンテナンスユーティリティー」、およびセクション 4.6.6 「`myisampack` — 圧縮された読み取り専用の MyISAM テーブルの生成」を参照してください。

MySQL 8.0 では、MyISAM ストレージエンジンはパーティション分割をサポートしていません。「以前のバージョンの MySQL で作成されたパーティション MyISAM テーブルは、MySQL 8.0 では使用できません」。詳細は、セクション 24.6.2 「ストレージエンジンに関連するパーティショニング制限」を参照してください。このようなテーブルを MySQL 8.0 で使用できるようにアップグレードする方法の詳細は、セクション 2.11.4 「MySQL 8.0 での変更」を参照してください。

MyISAM テーブルには次のような特徴があります。

- すべてのデータ値は、下位バイトから順に格納されます。これにより、データマシンとオペレーティングシステムは依存しなくなります。バイナリポータビリティのための唯一の要件は、2 の補数の符号付き整数と IEEE 浮動小数点フォーマットを使用することです。これらの要件は主流のマシンで幅広く使用されています。バイナリポータビリティは、組み込みシステムには適用されない可能性があります。特別のプロセッサを使用している場合があるためです。

下位バイトから順にデータを格納するため、大きな速度低下はありません。通常、テーブル行のバイトは整列しておらず、順番に未整列のバイトを読み込む処理は逆の順番に読み込む処理より時間がかかりません。また、カラム値をフェッチするサーバーのコードは、ほかのコードに比べて速度は重視されません。

- インデックスを効率良く圧縮ができるため、すべての数値キー値は上位バイトから順に格納されます。
- 大きなファイル (最大 63 ビットのファイル長) は、大きなファイルをサポートするファイルシステムとオペレーティングシステムでサポートされます。
- MyISAM テーブルの行数は、 $(2^{32})^2$ (1.844E+19) の制限があります。
- 1 つの MyISAM テーブルの最大インデックス数は 64 です。
1 つのインデックスの最大カラム数は 16 です。
- 最大キー長は 1000 バイトです。これは、ソースを変更して再コンパイルしても変えることができます。キーが 250 バイトより長いと、キーのブロックサイズはデフォルト値の 1024 バイトより大きい値が使用されます。
- ソートされた順番で行が挿入されたとき (`AUTO_INCREMENT` カラムを使用しているときと同様に)、上位のノードが 1 つのキーだけを含むように、インデックスツリーが分割されます。これにより、インデックスツリーの領域の利用率が向上します。
- テーブルごとに 1 つの `AUTO_INCREMENT` カラムの内部処理がサポートされます。MyISAM は `INSERT` 操作と `UPDATE` 操作でこのカラムを自動的に更新します。これにより、`AUTO_INCREMENT` カラムは速くなります (少なくとも 10%)。シーケンスの一番上の値は、削除されると、再利用されません。(`AUTO_INCREMENT` カラムが

マルチカラムインデックスの最後のカラムとして定義された場合、シーケンスの最上部から削除された値が再利用されます。) `AUTO_INCREMENT` 値は `ALTER TABLE` や `myisamchk` でリセットできます。

- 動的サイズの行は、削除を更新および挿入と併用すると、フラグメント化がかなり減少します。これは、削除された隣接ブロックを自動的に結合し、次のブロックが削除されたときにブロックを拡張することで行われます。
- `MyISAM` は同時挿入をサポートしています。テーブルのデータファイルの途中に空きブロックがなければ、ほかのスレッドがテーブルから読み取るのと同時に新しい行をそれに `INSERT` できます。行を削除した結果として、または動的長の行を現在の内容より多くのデータで更新した結果として、空きブロックが発生する可能性があります。すべての空きブロックが完全に使用されると(埋まると)、その後の挿入はふたたび並列になります。 [セクション 8.11.3 「同時挿入」](#) を参照してください。
- データファイルとインデックスファイルを異なる物理デバイス上の異なるディレクトリに置き、 `DATA DIRECTORY` および `INDEX DIRECTORY` テーブルオプションを `CREATE TABLE` に付けて速度を上げることができます。 [セクション 13.1.20 「CREATE TABLE ステートメント」](#) を参照してください。
- `BLOB` と `TEXT` カラムはインデックスを付けることができます。
- インデックスを付けたカラムでは `NULL` 値が許可されます。これには、キー当たり 0-1 バイトが必要です。
- 文字カラムごとに異なる文字セットを持つことができます。 [第 10 章 「文字セット、照合順序、Unicode」](#) を参照してください。
- `MyISAM` インデックスファイルの中に、テーブルが正しく閉じられたかどうかを表すフラグがあります。 `myisam_recover_options` システム変数を設定して `mysqld` を起動すると、`MyISAM` テーブルはオープン時に自動的にチェックされ、テーブルが正しくクローズされなかった場合は修復されます。
- `myisamchk` は `--update-state` オプションを付けて実行したかどうかをテーブルにマークします。 `myisamchk --fast` はこのマークがないテーブルだけを確認します。
- `myisamchk --analyze` はキー全体に対してするのと同様に、キーの一部に対する統計データを格納します。
- `myisampack` は `BLOB` と `VARCHAR` カラムを圧縮できます。

`MyISAM` は次のような機能もサポートしています。

- 真の `VARCHAR` 型をサポートしています。 `VARCHAR` カラムは 1 バイトか 2 バイトで格納される長さから始まります。
- `VARCHAR` カラムを持つテーブルの行の長さは固定でも動的でもかまいません。
- テーブル内の `VARCHAR` と `CHAR` カラムの長さの合計は、最大で 64K バイトになる場合があります。
- 任意の長さの `UNIQUE` 制約。

追加のリソース

- `MyISAM` ストレージエンジンに特化したフォーラムは <https://forums.mysql.com/list.php?21> で参照できます。

16.2.1 MyISAM 起動オプション

`MyISAM` テーブルの振る舞いを変えるために、次の `mysqld` オプションを使用できます。追加情報については [セクション 5.1.7 「サーバーコマンドオプション」](#) を参照してください。

表 16.3 「MyISAM オプションおよび変数リファレンス」

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
<code>bulk_insert_buffer_size</code>	はい	はい	はい		両方	はい
<code>concurrent_insert</code>	はい	はい	はい		グローバル	はい
<code>delay_key_write</code>	はい	はい	はい		グローバル	はい
<code>have_rtree_keys</code>			はい		グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
key_buffer_size	はい	はい	はい		グローバル	はい
log-isam	はい	はい				
myisam-block-size	はい	はい				
myisam_data_pointer_size	はい	はい	はい		グローバル	はい
myisam_max_sort_file_size	はい	はい	はい		グローバル	はい
myisam_mmap_size	はい	はい	はい		グローバル	いいえ
myisam_recover_options	はい	はい	はい		グローバル	いいえ
myisam_repair_threads	はい	はい	はい		両方	はい
myisam_sort_buffer_size	はい	はい	はい		両方	はい
myisam_stats_method	はい	はい	はい		両方	はい
myisam_use_mmap	はい	はい	はい		グローバル	はい
tmp_table_size	はい	はい	はい		両方	はい

次のシステム変数は MyISAM テーブルの振る舞いに影響を与えます。追加情報については [セクション5.1.8「サーバーシステム変数」](#) を参照してください。

- [bulk_insert_buffer_size](#)

大量挿入の最適化に使用されるツリーキャッシュのサイズです。

注記

これは、スレッド当たりの制限値です。

- [delay_key_write=ALL](#)

MyISAM テーブルへの書き込みの間にキーバッファをフラッシュしないでください。

注記

これを行う場合、MyISAM テーブルの使用中に別のプログラムから (別の MySQL サーバーから、[myisamchk](#) を使用して、など)、このテーブルにアクセスしないでください。そのようにすると、インデックスが破損するおそれがあります。 [--external-locking](#) を利用しても、このリスクは回避されません。

- [myisam_max_sort_file_size](#)

MyISAM インデックスの再作成時 ([REPAIR TABLE](#)、[ALTER TABLE](#) または [LOAD DATA](#)) に MySQL で使用できる一時ファイルの最大サイズ。ファイルサイズがこの値より大きい場合、さらに低速なキーキャッシュを代わりに使用してインデックスが作成されます。値はバイト単位で指定されます。

- [myisam_recover_options=mode](#)

クラッシュした MyISAM テーブルの自動リカバリにモードを設定します。

- [myisam_sort_buffer_size](#)

テーブルのリカバリ時に使用されるバッファのサイズを設定します。

[myisam_recover_options](#) システム変数を設定して [mysqld](#) を起動すると、自動リカバリがアクティブ化されます。この場合、サーバーが MyISAM テーブルを開いたときに、テーブルにクラッシュのマークが付いているかどうかや、テーブルのオープンカウント変数が 0 でないかどうか、そして外部ロックが使用不可能な状態でサーバーを起動させているかどうかを確認します。これらの条件のいずれかが true である場合、次のことが起こります。

- サーバーは、テーブルにエラーがあるかどうかを確認します。

- ・ サーバーがエラーを検出した場合、迅速なテーブル修復を行います (データファイルのソートは行いますが、再作成は行いません)。
- ・ データファイルの中にエラーがあるために (たとえば、重複キーエラーなど) 修復が失敗した場合、サーバーは再試行して、今度はデータファイルを再作成します。
- ・ それでも修復が失敗した場合、サーバーはもう一度古い修復オプション方式で試行します (ソートをせずに行ごとに書き込みます)。この方法は、どのタイプのエラーも修復できるはずであり、ディスク容量の要件は低くなっています。

リカバリで以前に完了したステートメントのすべての行をリカバリできず、`myisam_recover_options` システム変数の値に `FORCE` を指定しなかった場合、自動修復はエラーログにエラーメッセージとともに中断されます:

```
Error: Couldn't repair table: test.g00pages
```

`FORCE` を指定すると、代わりにこのような警告が書かれます。

```
Warning: Found 344 of 354 rows when repairing ./test/g00pages
```

自動リカバリ値に `BACKUP` が含まれている場合、リカバリプロセスでは、`tbl_name-datetime.BAK` という形式の名前のファイルが作成されます。これらのファイルを自動的にデータベースディレクトリからバックアップメディアに移動する `cron` スクリプトを持つことをお勧めします。

16.2.2 キーに必要な容量

`MyISAM` テーブルは B ツリーインデックスを使用します。インデックスファイルのサイズは、キーを $(key_length + 4) \times 0.67$ と計算し、すべてのキーに対してその値を合計して概算できます。これは、すべてのキーがソート順に挿入され、テーブル内のキーが圧縮されていないときの最悪なケースです。

文字列のインデックスはスペース圧縮されています。最初のインデックス部が文字列の場合、プリフィクスも圧縮されています。文字列カラムに含まれる後続の空白が長い場合、またはそのカラムが `VARCHAR` カラムであるために、必ずしもその長さがフルに使用されることがない場合は、スペース圧縮によってインデックスファイルが最悪の数値よりも小さくなります。プリフィクスの圧縮は文字列から始まるキーで使用されます。多くの文字列が同一のプリフィクスで始まる場合、プリフィクスの圧縮が役に立ちます。

`MyISAM` テーブルでは、テーブルの作成時に `PACK_KEYS=1` テーブルオプションを指定することで、数値のプリフィクスを圧縮することもできます。数値は上位バイトから順に格納されるため、同一のプリフィクスを持つ整数キーが多数あるときに役立ちます。

16.2.3 MyISAM テーブルのストレージフォーマット

`MyISAM` は 3 つの異なるストレージフォーマットをサポートします。使用するカラムの型によって、固定フォーマットと動的フォーマットの 2 つが自動的に選択されます。3 つ目は圧縮フォーマットで、`myisampack` ユーティリティを使用した場合にのみ作成できます (セクション 4.6.6 「`myisampack` — 圧縮された読み取り専用の `MyISAM` テーブルの生成」を参照してください)。

`BLOB` または `TEXT` カラムを持たないテーブルに対して、`CREATE TABLE` または `ALTER TABLE` を使用する場合、`ROW_FORMAT` テーブルオプションで、テーブルフォーマットを強制的に `FIXED` または `DYNAMIC` にできます。

`ROW_FORMAT` についての情報は、セクション 13.1.20 「`CREATE TABLE` ステートメント」を参照してください。

`myisamchk --unpack` を使用して、圧縮された `MyISAM` テーブルを解凍 (解凍) できます。詳細は、セクション 4.6.4 「`myisamchk` — `MyISAM` テーブルメンテナンスユーティリティ」を参照してください。

16.2.3.1 静的 (固定長) テーブルの特長

`MyISAM` テーブルでは、静的フォーマットがデフォルトです。テーブルに可変長のカラムが含まれない場合に使用されます (`VARCHAR`、`VARBINARY`、`BLOB`、または `TEXT`)。各行は固定バイト数で格納されます。

3 つの `MyISAM` ストレージフォーマットの中で、静的フォーマットが一番シンプルで安全です (一番破損しにくい)。またこれは、ディスク上でデータファイルの中の行が容易に検出できるという理由で、オンディスクフォーマットの中でもっとも高速です。インデックスの中の行数に基づいて行を検索するには、行数に行長を掛けて行の位置を計算

します。また、テーブルをスキャンするときに、ディスクの読み込み操作ごとに一定の行数を読み込むことが容易です。

MySQL サーバーが固定フォーマットの MyISAM ファイルに書き込んでいる最中にコンピュータがクラッシュした場合、その安全性が証明されます。この場合、`myisamchk` はそれぞれの行がどこで始まりどこで終わるかを簡単に判断できるため、通常、一部が書き込まれた行を除くすべての行を再利用できます。MyISAM テーブルのインデックスは、データ行に基づいていつでも再構築できます。

注記

固定長の行形式は、`BLOB` または `TEXT` カラムを持たないテーブルでのみ使用できます。明示的な `ROW_FORMAT` 句を使用してこのようなカラムを含むテーブルを作成しても、エラーや警告は発生せず、書式指定は無視されます。

静的フォーマットのテーブルには、次の特徴があります。

- `CHAR` および `VARCHAR` カラムには、特定の列幅にスペースが埋め込まれます (しかし、列の型は変わりません)。`BINARY` および `VARBINARY` カラムには、列幅に `0x00` バイトが埋め込まれます。
- `NULL` カラムの値が `NULL` であるかどうかを記録するには、行に追加の領域が必要です。各 `NULL` カラムは 1 ビット余分に占め、もっとも近いバイトまで丸められます。
- 非常に高速です。
- キャッシュが容易です。
- 行が固定位置にあるため、クラッシュしたあとも再構築が容易です。
- 大量の行を削除して、空きディスク容量をオペレーティングシステムに戻す場合を除いて、再編成の必要はありません。これを行うには、`OPTIMIZE TABLE` または `myisamchk -r` を使用します。
- 通常は、動的フォーマットテーブルよりも多くのディスク容量を必要とします。
- 静的サイズの行に必要な行の長さ (バイト単位) は、次の式を使用して計算されます:

```
row length = 1
+ (sum of column lengths)
+ (number of NULL columns + delete_flag + 7)/8
+ (number of variable-length columns)
```

`delete_flag` は静的行フォーマットのテーブルに対しては 1 です。静的テーブルは、行が削除されているかどうかを示すフラグとして、行レコード内の 1 ビットを使用します。このフラグは動的行ヘッダーに格納されるので、動的テーブルの場合、`delete_flag` は 0 です。

16.2.3.2 動的テーブルの特徴

MyISAM テーブルが可変長カラムを含んでいる場合 (`VARCHAR`、`VARBINARY`、`BLOB`、または `TEXT`)、またはテーブルが `ROW_FORMAT=DYNAMIC` テーブルオプションで作成された場合は、動的ストレージフォーマットが使用されます。

動的フォーマットは、それぞれの行に行の長さを示すヘッダーがあるため、静的フォーマットよりも少し複雑です。更新の結果として行が長くなった場合、行がフラグメント化される可能性があります (非連続的な断片で格納されず)。

`OPTIMIZE TABLE` または `myisamchk -r` を使用して、テーブルをデフラグできます。可変長カラムも含んだテーブルで、固定長カラムに頻繁にアクセスしたり変更したりする場合、可変長カラムを他のテーブルに移動して、フラグメンテーションを回避する方法が良い場合があります。

動的フォーマットのテーブルには、次の特徴があります。

- 長さが 4 未満のカラムを除くすべての文字列カラムは動的です。
- それぞれの行の先頭には、どのカラムが空の文字列 (文字列カラムの場合) またはゼロ (数値カラムの場合) を含むかを示すビットマップが付いています。これには、`NULL` 値を含むカラムは含まれません。後続スペースを削除したあとに文字列カラムの長さがゼロであったり、数値カラムの値がゼロであったりした場合、ビットマップの中で

マークが付きますが、ディスクには保存されません。空ではない文字列は、長さバイトに文字列コンテンツを加えて保存されます。

- **NULL** カラムの値が **NULL** であるかどうかを記録するには、行に追加の領域が必要です。各 **NULL** カラムは 1 ビット余分に占め、もっとも近いバイトまで丸められます。
- 通常、固定長テーブルに比べると、必要なディスク容量がかなり少なくなります。
- それぞれの行は、必要とする容量だけを使用します。ただし、行がさらに大きくなると、必要な数の断片に分割され、行のフラグメンテーションが起こることになります。たとえば、行の長さを延長する情報を使って行を更新すると、その行はフラグメント化されます。このような場合、パフォーマンスを上げるために、ときどき **OPTIMIZE TABLE** または **myisamchk -r** を実行しなければいけないかもしれません。 **myisamchk -ei** を使用して、テーブルの統計を取得します。
- 行がいくつもの断片にフラグメント化されている場合や、リンク (フラグメント) が失われている場合があるため、クラッシュ後の再構築は、静的フォーマットテーブルよりも難しくなります。
- 動的サイズの行の予想される行長は、次の式で計算されます。

```
3  
+ (number of columns + 7) / 8  
+ (number of char columns)  
+ (packed size of numeric columns)  
+ (length of strings)  
+ (number of NULL columns + 7) / 8
```

それぞれのリンクには 6 バイトのペナルティーがあります。更新によって行が拡大される場合は、必ず動的行がリンクされます。新しいリンクはそれぞれ少なくとも 20 バイトであるため、おそらく次の拡張は同じリンクになります。そうでない場合、別のリンクが作成されます。 **myisamchk -ed** を利用してリンク数を確認できます。 **OPTIMIZE TABLE** または **myisamchk -r** を使用すると、すべてのリンクを削除できます。

16.2.3.3 圧縮テーブルの特徴

圧縮ストレージフォーマットは、**mysampack** ツールで生成される読み取り専用のフォーマットです。圧縮テーブルは **myisamchk** を使って解凍できます。

圧縮テーブルには次のような特徴があります。

- 圧縮テーブルに必要なディスク容量はごくわずかです。これによりディスクの使用量は最少になるため、低速のディスクを使用する場合に役立ちます (CD-ROM など)。
- それぞれの行は個々に圧縮されるため、アクセスのオーバーヘッドはごくわずかです。行のヘッダーに必要なバイト数は、テーブル中の一番大きい行によって異なりますが、1-3 バイトです。各カラムは別々に圧縮されます。カラムごとに異なる Huffman ツリーがあるのが一般的です。圧縮タイプのいくつかは次のとおりです。
 - サフィクススペース圧縮。
 - プリフィクススペース圧縮。
 - 値が 0 の数値は 1 ビットで格納されます。
 - 値の範囲が小さい整数カラムは、可能な限り小さな型を使って格納されます。たとえば、**BIGINT** カラム (8 バイト) のすべての値が **-128** から **127** の範囲内にある場合は、このカラムを **TINYINT** カラム (1 バイト) として格納できます。
 - カラムの可能値が少ない場合は、データの型を **ENUM** に変換します。
 - カラムに、上記の圧縮型を組み合わせ使用してもかまいません。
- 固定長または動的長の行を使用できます。

注記

圧縮テーブルは読み取り専用なので、テーブルの行を更新したり、行を追加したりはできませんが、DDL (データ定義言語) 操作は有効です。たとえば、**DROP** を使用してテーブルを削除しても、**TRUNCATE TABLE** を使用してテーブルを空にしてもかまいません。

16.2.4 MyISAM テーブルの問題点

MySQL がデータの格納に使用するファイルフォーマットは幅広い検査を受けていますが、データベーステーブルの破損を招きかねない状況は常に存在します。次に、これがどのようにして起こるのか、またどのように対処すればよいのかについて説明します。

16.2.4.1 MyISAM テーブルの破損

MyISAM のテーブルフォーマットは、きわめて信頼性の高いフォーマットです (SQL ステートメントが行うテーブルに対するすべての変更は、そのステートメントが戻る前に書き込まれます) が、それでも次の状況が発生した場合、テーブルが破損するおそれがあります。

- `mysqld` プロセスは、書き込みの最中に強制終了されます。
- コンピュータが予期せずシャットダウンされます (たとえば、コンピューターの電源が切られた場合など)。
- ハードウェア障害。
- サーバーが修正中のテーブルを、外部プログラム (`myisamchk` など) を使用して同時に修正しています。
- MySQL または MyISAM コードのソフトウェアバグです。

テーブルが破損した場合の典型的な兆候は、次のとおりです。

- テーブルからデータを選択するときに、次のエラーが表示されます。

```
Incorrect key file for table: '...'. Try to repair it
```

- クエリーが、テーブル内で行を検出しない、または不完全な結果を返します。

MyISAM テーブルのヘルスを `CHECK TABLE` ステートメントを利用して確認でき、破損した MyISAM テーブルを `REPAIR TABLE` を利用して修復できます。 `mysqld` が動作していない場合は、 `myisamchk` コマンドを利用してテーブルを確認したり修復したりすることもできます。 [セクション13.7.3.2「CHECK TABLE ステートメント」](#)、 [セクション13.7.3.5「REPAIR TABLE ステートメント」](#)、および [セクション4.6.4「myisamchk — MyISAM テーブルメンテナンスユーティリティー」](#) を参照してください。

テーブルが頻繁に破損する場合は、その原因を突き止めるようにしてください。最も重要なことは、予期しないサーバー終了の結果としてテーブルが破損したかどうかです。エラーログの最新の `restarted mysqld` メッセージを探すと、簡単に検証できます。このようなメッセージがある場合、テーブルの破損はサーバーのダウンによる可能性が高くなります。そうでなければ、破損は通常作業の最中に起きた可能性があります。これはバグです。問題点を明らかにする再現可能なテストケースを作成するべきです。 [セクションB.3.3.3「MySQL が繰り返しクラッシュする場合の対処方法」](#) および [セクション5.9「MySQL のデバッグ」](#) を参照してください。

16.2.4.2 適切に閉じられなかったテーブルの問題

各 MyISAM インデックスファイル (`.MYI` ファイル) には、テーブルが適切に閉じられたかどうかをチェックするために使用できるカウンタがヘッダーの中にあります。 `CHECK TABLE` または `myisamchk` から次のような警告が表示された場合、このカウンタの同期が取れていないことを示しています。

```
clients are using or haven't closed the table properly
```

この警告は、必ずしもテーブルが破損されたという意味ではありませんが、少なくともテーブルを確認したほうがよいでしょう。

カウンターは次のように機能します。

- MySQL でテーブルが最初に更新されるときに、インデックスファイルのヘッダー内にあるカウンタが増えます。
- その後の更新ではカウンタは変更されません。
- テーブルの最後のインスタンスが閉じられるとき (`FLUSH TABLES` 操作が行われたため、またはテーブルキャッシュの中に場所がないため) に、それまでにテーブルが更新されていると、カウンタの値が減少します。
- テーブルを修復するか、チェックして問題がなかった場合は、カウンタがゼロにリセットされます。

- テーブルを検査する可能性のあるほかのプロセスとの相互作用の問題を回避するため、カウンタがゼロである場合は、テーブルを閉じる際にカウンタの値は減りません。

つまり、カウンタが不正確になる可能性があるのは、次のような場合だけです。

- **MyISAM** テーブルのコピーが、最初に **LOCK TABLES** と **FLUSH TABLES** を発行しないで行われる。
- MySQL が更新されてから閉じられるまでの間にクラッシュした。(MySQL は常に各ステートメントの間のすべてに対して書き込みを発行するため、テーブルは問題ない可能性があります。)
- **mysqld** と同時に使用した **myisamchk --recover** が **myisamchk --update-state** によって、テーブルが修正された。
- 別のサーバーによって使用されている最中に、複数の **mysqld** サーバーがテーブルを使用し、1つのサーバーが **REPAIR TABLE** または **CHECK TABLE** をテーブルで実行した。このセットアップでは、ほかのサーバーから警告を受ける可能性があります。しかし、**CHECK TABLE** の使用が安全です。しかし、あるサーバーがデータファイルを新しいファイルに置き換えた場合、別のサーバーには通知されないため、**REPAIR TABLE** は避けるべきです。

一般的に、複数のサーバー間でデータディレクトリを共有することは推奨されません。追加情報については、[セクション5.8「1つのマシン上での複数のMySQLインスタンスの実行」](#)を参照してください。

16.3 MEMORY ストレージエンジン

MEMORY ストレージエンジン (従来は **HEAP** と呼ばれていました) は、メモリーに格納された内容で特定用途のテーブルを作成します。データは、クラッシュ、ハードウェア問題、または電源停止に弱いため、これらのテーブルは、一時的な作業領域またはほかのテーブルから抽出されたデータの読み取り専用キャッシュとして使用されるだけです。

表 16.4 「MEMORY ストレージエンジンの機能」

機能	Support
B ツリーインデックス	はい
MVCC	いいえ
T ツリーインデックス	いいえ
インデックスキャッシュ	N/A
クラスタデータベースのサポート	いいえ
クラスタ化されたインデックス	いいえ
ストレージの制限	RAM
データキャッシュ	N/A
データディクショナリ向け更新統計	はい
トランザクション	いいえ
ハッシュインデックス	はい
バックアップ/ポイントインタイムリカバリ (ストレージエンジン内ではなくサーバー内で実装されています。)	はい
レプリケーションのサポート (ストレージエンジン内ではなくサーバー内で実装されています。)	制限付き (このセクションの後半の説明を参照してください。)
ロック粒度	Table
全文検索インデックス	いいえ
圧縮データ	いいえ
地理空間インデックスのサポート	いいえ
地理空間データ型のサポート	いいえ
外部キーのサポート	いいえ
暗号化データ	はい (暗号化機能を介してサーバーに実装されます。)

- MEMORY または NDB Cluster を使用する場合
- パーティション化
- パフォーマンスの特徴
- MEMORY テーブルの特性
- MEMORY テーブルへの DDL 操作
- インデックス
- ユーザー作成の一時テーブル
- データのロード
- MEMORY テーブルとレプリケーション
- メモリー使用量の管理
- 追加のリソース

MEMORY または NDB Cluster を使用する場合

重要で可用性の高い、または頻繁に更新されるデータのために MEMORY ストレージエンジンを使用するアプリケーションを配備する開発者は、NDB Cluster の方がよいかどうかを考慮するようにしてください。MEMORY エンジンの典型的なユースケースには、次の特徴があります。

- セッション管理やキャッシングなどの一時的で重要でないデータに関連する操作。MySQL サーバーが停止または再起動したときに、MEMORY テーブルのデータは失われます。
- 高速アクセスおよび低待機時間のためのインメモリー保存。データボリュームはメモリー内に完全に収まり、オペレーティングシステムによる仮想メモリーページのスワップアウトはありません。
- 読み取り専用または読み取りが大半のデータのアクセスパターン (更新が制限されています)。

NDB Cluster は、より高いパフォーマンスレベルの MEMORY エンジンと同じ機能を提供し、MEMORY では使用できない追加機能を提供します:

- クライアント間で競合の少ない行レベルロックとマルチスレッド操作。
- 書き込みを含むステートメント混在時の拡張性。
- データ持続性のためのディスクバックアップ式操作 (オプション)。
- 単一障害点がない、シェアードナッシングアーキテクチャーと複数ホスト操作。99.999% の可用性を実現できます。
- ノードをまたがる自動データ分散。アプリケーションの開発者はカスタムの共有またはパーティション化ソリューションを作る必要がありません。
- 可変長データ型 (MEMORY がサポートしない BLOB および TEXT を含みます) をサポートします。

パーティション化

MEMORY テーブルはパーティション化できません。

パフォーマンスの特徴

MEMORY のパフォーマンスは、更新処理時のシングルスレッド実行とテーブルロックオーバーヘッドが原因の競合によって抑制されます。このため、負荷が増えたときに拡張性が制限されます (特に、書き込みを含むステートメント混在時)。

MEMORY テーブルのインメモリー処理にかかわらず、それらは、汎用目的クエリーのために、または読み取り/書き込み負荷では、必ずしもビジーサーバーの InnoDB テーブルより高速である必要はありません。特に、更新実行に関与するテーブルロックは、複数セッションからの MEMORY テーブルの並列使用の速度を低下させる可能性があります。

MEMORY テーブルで実行されるクエリーの種類によっては、デフォルトのハッシュデータ構造 (一意キーで 1 つの値を検索する場合)、または汎用目的の B ツリーデータ構造 (等号、不等号、未満または「- を超える」などの範囲演算子などを含むすべての種類のクエリーの場合) のいずれかとしてインデックスを作成する場合があります。次のセクションでは、両方の種類のインデックスを作成するための構文について説明します。パフォーマンス面でよくある問題は、B ツリーインデックスがより効率的な作業負荷で、デフォルトのハッシュインデックスを使用していることです。

MEMORY テーブルの特性

MEMORY ストレージエンジンは、ディスク上にファイルを作成しません。テーブル定義は、MySQL データディクショナリに格納されます。

MEMORY テーブルには次のような特徴があります。

- MEMORY テーブルの領域は小さなブロックに割り当てられます。テーブルは、挿入に 100% 動的ハッシュを使用します。オーバーフロー領域や余分なキー領域は必要ありません。フリーリスト用の余分な領域は必要ありません。削除された行はリンクリストに置かれ、新しいデータをテーブルに挿入するときに再利用されます。MEMORY テーブルでは、ハッシュテーブルで一般的に削除 + 挿入に関連付けられる問題も起こりません。
- MEMORY テーブルは固定長の行ストレージフォーマットを使用します。VARCHAR などの可変長型は、固定長を使用して格納されます。
- MEMORY テーブルは BLOB または TEXT カラムを含むことができません。
- MEMORY は AUTO_INCREMENT カラムのサポートを含みます。
- TEMPORARY MEMORY でないテーブルは、ほかの TEMPORARY でないテーブルと同様に、すべてのクライアントで共有されます。

MEMORY テーブルへの DDL 操作

MEMORY テーブルを作成するには、CREATE TABLE ステートメントで ENGINE=MEMORY 句を指定します。

```
CREATE TABLE t (i INT) ENGINE = MEMORY;
```

エンジン名が表すように、MEMORY テーブルはメモリーに格納されます。デフォルトではハッシュインデックスを使用するため、単一値の検索には非常に高速であり、一時テーブルの作成には非常に役立ちます。ただし、サーバーがシャットダウンすると、MEMORY テーブルに格納されたすべての行が失われます。定義は MySQL データディクショナリに格納されますが、サーバーの再起動時には空であるため、テーブル自体は引き続き存在します。

この例は、MEMORY テーブルをどのように作成、使用、および削除できるかを示しています。

```
mysql> CREATE TABLE test ENGINE=MEMORY
      SELECT ip,SUM(downloads) AS down
      FROM log_table GROUP BY ip;
mysql> SELECT COUNT(ip),AVG(down) FROM test;
mysql> DROP TABLE test;
```

MEMORY テーブルの最大サイズは max_heap_table_size システム変数によって制限されます (デフォルト値は 16M バイト)。MEMORY テーブルに異なるサイズ制限を適用するには、この変数値を変更します。CREATE TABLE、それに続く ALTER TABLE または TRUNCATE TABLE の実質的な値は、テーブルの有効期限に使用される値です。サーバーを再起動しても、既存の MEMORY テーブルの最大サイズがグローバルの max_heap_table_size 値に設定されます。各テーブルのサイズをこのセクションの後半で説明するように設定できます。

インデックス

MEMORY ストレージエンジンは HASH と BTREE の両方のインデックスをサポートしています。ここに示すように USING 句を追加することによりどちらであるかを指定できます。


```
CREATE TABLE lookup
(id INT, INDEX USING HASH (id))
ENGINE = MEMORY;
CREATE TABLE lookup
(id INT, INDEX USING BTREE (id))
ENGINE = MEMORY;
```

B ツリーとハッシュインデックスの一般的な特徴については、[セクション8.3.1「MySQL のインデックスの使用の仕組み」](#)を参照してください。

MEMORY テーブルでは、テーブル当たり最大で 64 個のインデックス、インデックス当たり 16 個のカラム、3072 バイトの最大キー長を持つことができます。

MEMORY テーブルのハッシュインデックスでキーの重複の程度が高いと (同じ値を含むインデックスエントリが多い)、キー値に影響を与えるテーブルへの更新処理とすべての削除処理の速度が大きく低下します。この低下の程度は重複の程度に比例します (または、インデックスカーディナリティーに反比例します)。**BTREE** インデックスを使用することで、この問題を回避できます。

MEMORY テーブルには、非一意キーを持つことができます。(これは、ハッシュインデックスの実装ではまれな特徴です。)

インデックスが付けられたカラムに **NULL** 値を含むことができます。

ユーザー作成の一時テーブル

MEMORY テーブルの内容はメモリーに格納されます。これは **MEMORY** テーブルが、クエリーの処理中にその場でサーバーが作成する内部一時テーブルと共有する特性です。ただし、2 つのタイプのテーブルには違いがあり、**MEMORY** テーブルはストレージ変換の影響を受けませんが、内部一時テーブルは次のような影響があります。

- 内部一時テーブルが大きくなりすぎると、[セクション8.4.4「MySQL での内部一時テーブルの使用」](#)で説明したように、サーバーは自動的にオンディスクストレージに変換します。
- ユーザー作成の **MEMORY** テーブルは、決してディスクテーブルに変換されません。

データのロード

MySQL サーバーの起動時に **MEMORY** テーブルに移入するには、`init_file` システム変数を使用できます。たとえば、`INSERT INTO ... SELECT` や `LOAD DATA` などのステートメントをファイルに挿入して永続データソースからテーブルをロードし、`init_file` を使用してファイルに名前を付けることができます。[セクション5.1.8「サーバーシステム変数」](#) および [セクション13.2.7「LOAD DATA ステートメント」](#) を参照してください。

MEMORY テーブルとレプリケーション

レプリケーションソースサーバーが停止して再起動すると、その **MEMORY** テーブルは空になります。この効果をレプリカにレプリケートするには、ソースが起動後に最初に特定の **MEMORY** テーブルを使用するときに、そのテーブルの `DELETE` ステートメントまたは (MySQL 8.0.22 から) `TRUNCATE TABLE` ステートメントをバイナリログに書き込むことによってテーブルを空にする必要があることをレプリカに通知するイベントをログに記録します。レプリカサーバーを停止して再起動すると、その **MEMORY** テーブルも空になり、`DELETE` または (MySQL 8.0.22 から) `TRUNCATE TABLE` ステートメントが独自のバイナリログに書き込まれ、ダウンストリームレプリカに渡されます。

レプリケーショントポロジで **MEMORY** テーブルを使用する場合、状況によっては、ソースのテーブルとレプリカのテーブルが異なることがあります。失効した読み取りまたはエラーを防ぐためのこれらの各状況の処理の詳細は、[セクション17.5.1.21「レプリケーションと MEMORY テーブル」](#) を参照してください。

メモリー使用量の管理

サーバーには、同時に使用されるすべての **MEMORY** テーブルを保持するための十分なメモリーが必要です。

MEMORY テーブルから各行を削除しても、メモリーは再利用されません。テーブル全体が削除された場合のみ、メモリーが再利用されます。削除された行に以前に使用されたメモリーは同じテーブル内の新しい行に再利用され

ます。MEMORY テーブルの内容が必要でなくなったときに、それが使用していたすべてのメモリーを解放するには、DELETE または TRUNCATE TABLE を実行してすべての行を削除するか、DROP TABLE を使用してテーブルを完全に削除します。削除された行が使用していたメモリーを解放するには、ALTER TABLE ENGINE=MEMORY を使用してテーブルを強制的に再作成します。

MEMORY テーブルで 1 つの行に必要なメモリーは、次の式で計算されます。

```
SUM_OVER_ALL_BTREE_KEYS(max_length_of_key + sizeof(char*) * 4)
+ SUM_OVER_ALL_HASH_KEYS(sizeof(char*) * 2)
+ ALIGN(length_of_row+1, sizeof(char*))
```

ALIGN() は行の長さを char ポインタサイズのちょうど倍数にするための切り上げ係数を表します。sizeof(char*) は 32 ビットマシンでは 4、64 ビットマシンでは 8 です。

前に述べたように、max_heap_table_size システム変数は MEMORY テーブルの最大サイズの制限値を設定します。各テーブルの最大サイズを制御するには、各テーブルを作成する前に、この変数のセッション値を設定します。(すべてのクライアントが作成した MEMORY テーブルに、グローバル max_heap_table_size 値を使用するのであれば、この値を変更しないでください。) 次は、2 つの MEMORY テーブル (最大サイズがそれぞれ 1M バイトと 2M バイト) を作成する例です。

```
mysql> SET max_heap_table_size = 1024*1024;
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t1 (id INT, UNIQUE(id)) ENGINE = MEMORY;
Query OK, 0 rows affected (0.01 sec)

mysql> SET max_heap_table_size = 1024*1024*2;
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t2 (id INT, UNIQUE(id)) ENGINE = MEMORY;
Query OK, 0 rows affected (0.00 sec)
```

両方のテーブルは、サーバーが再起動した場合、サーバーのグローバル max_heap_table_size 値に戻ります。

MEMORY テーブルに対して CREATE TABLE ステートメントの MAX_ROWS テーブルオプションを指定して、テーブルに格納する予定の行数に関するヒントを提供することもできます。これによって max_heap_table_size 値 (引き続き最大テーブルサイズの制約として機能) を超えてテーブルが拡大できなくなります。MAX_ROWS を使用できるだけの最大限の柔軟性を得るには、少なくとも各 MEMORY テーブルが拡大できる値程度に max_heap_table_size を設定してください。

追加のリソース

MEMORY ストレージエンジンに特化したフォーラムは、<https://forums.mysql.com/list.php?92>で参照できます。

16.4 CSV ストレージエンジン

CSV ストレージエンジンは、カンマ区切り値形式を使用してデータをテキストファイルに保存します。

CSV ストレージエンジンは、常に MySQL サーバーにコンパイルされます。

CSV エンジンのソースを調べるには、MySQL ソース配布の storage/csv ディレクトリを検索します。

CSV テーブルを作成すると、サーバーは、名前がテーブル名で始まり .CSV 拡張子を持つプレーンテキストデータファイルを作成します。データをテーブルに保存するとき、ストレージエンジンはデータファイルにカンマ区切り値形式で保存します。

```
mysql> CREATE TABLE test (i INT NOT NULL, c CHAR(10) NOT NULL)
ENGINE = CSV;
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO test VALUES(1,'record one'),(2,'record two');
Query OK, 2 rows affected (0.05 sec)
Records: 2 Duplicates: 0 Warnings: 0
```

```
mysql> SELECT * FROM test;
+----+-----+
| i | c |
+----+-----+
| 1 | record one |
| 2 | record two |
+----+-----+
2 rows in set (0.00 sec)
```

CSV テーブルを作成すると、テーブルの状態およびテーブルに存在する行数を格納する対応するメタファイルも作成されます。このファイルの名前は **CSM** 拡張子のついたテーブル名と同じです。

前のステートメントの実行で作成されたデータベースディレクトリにある **test.CSV** ファイルを調べると、その内容は次のようであるはずですが、

```
"1","record one"
"2","record two"
```

この形式は、Microsoft Excel などのスプレッドシートアプリケーションによって読み取りおよび書き込みが可能です。

16.4.1 CSV テーブルの修復と確認

CSV ストレージエンジンは、破損した CSV テーブルを検証し、可能な場合は修復するための **CHECK TABLE** および **REPAIR TABLE** ステートメントをサポートしています。

CHECK TABLE ステートメントを実行すると、正しいフィールドセパレータ、エスケープされたフィールド (一致する引用符または欠落している引用符)、テーブル定義と比較した正しいフィールド数および対応する CSV メタファイルの存在を検索することで、CSV ファイルの妥当性がチェックされます。最初に検出された無効な行が原因でエラーが発生します。有効なテーブルをチェックすると、次に示すような出力が生成されます:

```
mysql> CHECK TABLE csvtest;
+-----+-----+-----+-----+
| Table | Op | Msg_type | Msg_text |
+-----+-----+-----+-----+
| test.csvtest | check | status | OK |
+-----+-----+-----+-----+
```

破損したテーブルをチェックすると、次のようなフォルトが返されます

```
mysql> CHECK TABLE csvtest;
+-----+-----+-----+-----+
| Table | Op | Msg_type | Msg_text |
+-----+-----+-----+-----+
| test.csvtest | check | error | Corrupt |
+-----+-----+-----+-----+
```

テーブルを修復するには、**REPAIR TABLE** を使用します。これにより、既存の CSV データから可能な限り多くの有効な行がコピーされ、既存の CSV ファイルがリカバリされた行で置換されます。破損したデータ以降のすべての行は失われます。

```
mysql> REPAIR TABLE csvtest;
+-----+-----+-----+-----+
| Table | Op | Msg_type | Msg_text |
+-----+-----+-----+-----+
| test.csvtest | repair | status | OK |
+-----+-----+-----+-----+
```

警告

修復中、破損した最初の行までの CSV ファイルの行のみが新しいテーブルにコピーされません。破損した最初の行からテーブルの最後までほかのすべての行は、有効な行であっても削除されます。

16.4.2 CSV の制限

CSV ストレージエンジンはインデックスをサポートしていません。

CSV ストレージエンジンはパーティション分割をサポートしていません。

CSV ストレージエンジンを使用して作成したすべてのテーブルには、すべてのカラムに **NOT NULL** 属性が必要です。

16.5 ARCHIVE ストレージエンジン

ARCHIVE ストレージエンジンは、非常に小さなフットプリントに大量のインデックス化されていないデータを格納する、特別な目的のテーブルを作成します。

表 16.5 「ARCHIVE ストレージエンジンの機能」

機能	Support
B ツリーインデックス	いいえ
MVCC	いいえ
T ツリーインデックス	いいえ
インデックスキャッシュ	いいえ
クラスタデータベースのサポート	いいえ
クラスタ化されたインデックス	いいえ
ストレージの制限	なし
データキャッシュ	いいえ
データディクショナリ向け更新統計	はい
トランザクション	いいえ
ハッシュインデックス	いいえ
バックアップ/ポイントインタイムリカバリ (ストレージエンジン内ではなくサーバー内で実装されています。)	はい
レプリケーションのサポート (ストレージエンジン内ではなくサーバー内で実装されています。)	はい
ロック粒度	行
全文検索インデックス	いいえ
圧縮データ	はい
地理空間インデックスのサポート	いいえ
地理空間データ型のサポート	はい
外部キーのサポート	いいえ
暗号化データ	はい (暗号化機能を介してサーバーに実装されます。)

ARCHIVE ストレージエンジンは MySQL バイナリ配布に含まれています。ソースから MySQL を構築する場合にこのストレージエンジンを有効にするには、**CMake** を **-DWITH_ARCHIVE_STORAGE_ENGINE** オプションで呼び出します。

ARCHIVE エンジンのソースを調べるには、MySQL ソース配布の `storage/archive` ディレクトリを検索します。

ARCHIVE ストレージエンジンが **SHOW ENGINES** ステートメントで使用できるかどうかを確認できます。

ARCHIVE テーブルを作成すると、ストレージエンジンはテーブル名で始まる名前のファイルを作成します。データファイルの拡張子は **.ARZ** です。最適化操作中に **.ARN** ファイルが現れる場合があります。

ARCHIVE エンジンでは、**INSERT**、**REPLACE** および **SELECT** はサポートされますが、**DELETE** または **UPDATE** はサポートされません。**ORDER BY** 操作、**BLOB** カラムおよび空間データ型はサポートされています (**セクション 11.4.1 「空間データ型」** を参照)。地理空間参照システムはサポートされていません。ARCHIVE エンジンは低レベルロックを使用します。

ARCHIVE エンジンは `AUTO_INCREMENT` カラム属性をサポートしています。ARCHIVE カラムには、一意のインデックスまたは一意でないインデックスのどちらかを付けることができます。ほかのカラムにインデックスを作成しようとすると、エラーになります。ARCHIVE エンジンは、それぞれ、新しいテーブルの最初のシーケンス値を指定したり、既存テーブルのシーケンス値をリセットしたりする `CREATE TABLE` ステートメントの `AUTO_INCREMENT` テーブルオプションもサポートしています。

ARCHIVE は、現在の最大カラム値未満の値を `AUTO_INCREMENT` カラムに挿入する機能をサポートしていません。そのようにしようとすると、`ER_DUP_KEY` エラーになります。

ARCHIVE エンジンは `BLOB` カラムが要求されない場合はそれらを見逃して、読み取り中にそれらを通り過ぎてスキップします。

ARCHIVE ストレージエンジンはパーティション分割をサポートしていません。

ストレージ: 行は挿入されるときに圧縮されます。ARCHIVE エンジンは `zlib` ロスレスデータ圧縮を使用します (<http://www.zlib.net/> を参照してください)。OPTIMIZE TABLE を使用してテーブルを解析したり、より小さいフォーマットにテーブルを圧縮したりできます (OPTIMIZE TABLE を利用する理由については、このセクションの後半を参照してください)。このエンジンは `CHECK TABLE` もサポートしています。使用される挿入のタイプはいくつかあります。

- `INSERT` ステートメントは行を圧縮バッファに単純に入れ、バッファは必要に応じてフラッシュします。バッファへの挿入はロックで保護されています。SELECT はフラッシュを強制的に実行します。
- 大量挿入は、ほかの挿入が同時に発生した場合を除いて (その場合は部分的に可視になります)、完了後にのみ可視になります。SELECT は、ロード中に通常の挿入が発生した場合を除いて、大量挿入をフラッシュすることはありません。

取り出し: 取り出しの際、要求によって行が圧縮解除され、行キャッシュはありません。SELECT 操作によって完全なテーブルスキャンが実行されます。SELECT が発生すると、現在使用できる行数を検出し、その行数を読み取ります。SELECT は一貫性読み取りとして実行されます。バルク挿入のみを使用しない限り、挿入中に多くの SELECT ステートメントが圧縮を妨げる可能性があることに注意してください。圧縮品質を高めるために、OPTIMIZE TABLE または REPAIR TABLE を使用できます。SHOW TABLE STATUS によって報告される ARCHIVE テーブルの行数は常に正確です。セクション 13.7.3.4 「OPTIMIZE TABLE ステートメント」、セクション 13.7.3.5 「REPAIR TABLE ステートメント」、およびセクション 13.7.7.38 「SHOW TABLE STATUS ステートメント」を参照してください。

追加のリソース

- ARCHIVE ストレージエンジンに特化したフォーラムは <https://forums.mysql.com/list.php?112> で参照できます。

16.6 BLACKHOLE ストレージエンジン

BLACKHOLE ストレージエンジンは、データを受け取るけれども破棄して格納しない「ブラックホール」として機能します。検索は、常に空の結果を返します。

```
mysql> CREATE TABLE test(i INT, c CHAR(10)) ENGINE = BLACKHOLE;
Query OK, 0 rows affected (0.03 sec)

mysql> INSERT INTO test VALUES(1,'record one'),(2,'record two');
Query OK, 2 rows affected (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM test;
Empty set (0.00 sec)
```

ソースから MySQL を構築する場合に BLACKHOLE ストレージエンジンを有効にするには、CMake を `-DWITH_BLACKHOLE_STORAGE_ENGINE` オプションで呼び出します。

BLACKHOLE エンジンのソースを調べるには、MySQL ソース配布の `sql` ディレクトリを検索します。

BLACKHOLE テーブルを作成すると、サーバーはグローバルデータディクショナリにテーブル定義を作成します。テーブルに関連付けられたファイルがありません。

BLACKHOLE ストレージエンジンはすべての種類のインデックスをサポートしています。すなわち、テーブル定義にインデックス宣言を含めることができます。

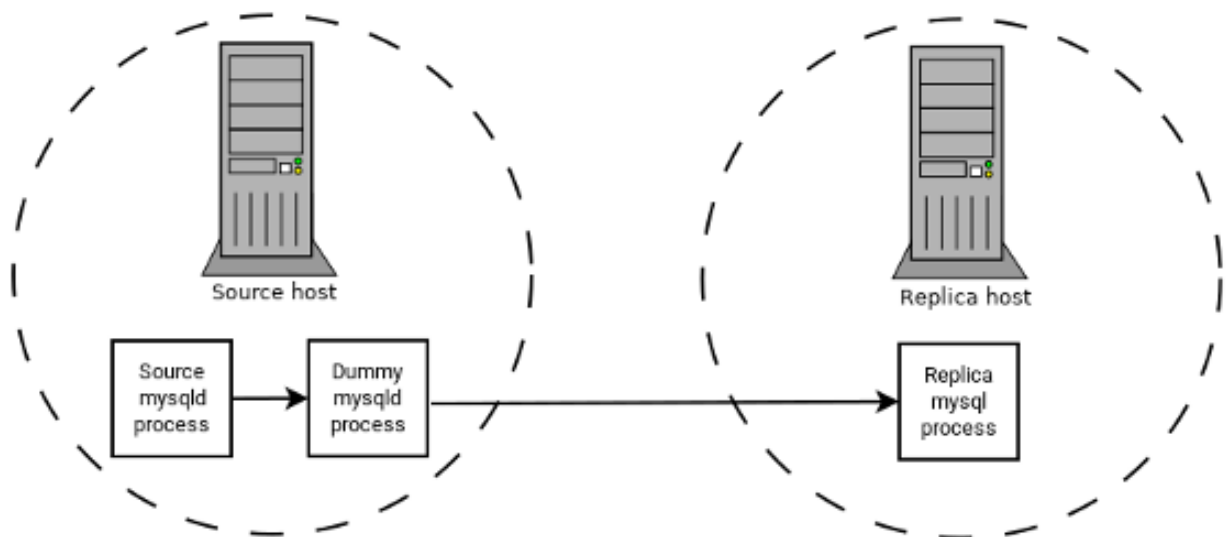
BLACKHOLE ストレージエンジンはパーティション分割をサポートしていません。

BLACKHOLE ストレージエンジンが `SHOW ENGINES` ステートメントで使用できるかどうかを確認できます。

BLACKHOLE テーブルへの挿入にはデータは格納されませんが、ステートメントベースのバイナリロギングが有効になっている場合は、SQL ステートメントがログに記録され、複製サーバーに複製されます。これは、繰り返しまたはフィルタメカニズムとして役立つ場合があります。

アプリケーションでレプリカ側のフィルタリングルールが必要だが、最初にすべてのバイナリログデータをレプリカに転送すると、トラフィックが多すぎるとします。このような場合は、レプリケーションソースサーバー上で、デフォルトのストレージエンジンが BLACKHOLE である「ダミー」レプリカプロセスを次のように設定できます：

図 16.1 フィルタリングに BLACKHOLE を使用したレプリケーション



ソースはバイナリログに書き込みます。「ダミー」mysql プロセスはレプリカとして機能し、`replicate-do-*` ルールと `replicate-ignore-*` ルールの目的の組合せを適用して、独自のフィルタ処理された新しいバイナリログを書き込みます。セクション 17.1.6 「レプリケーションおよびバイナリロギングのオプションと変数」を参照してください。このフィルタ処理されたログはレプリカに提供されます。

ダミープロセスは実際にはデータを格納しないため、レプリケーションソースサーバーで追加の mysql プロセスを実行することで発生する処理オーバーヘッドはほとんどありません。このタイプの設定は、追加のレプリカを使用して繰り返すことができます。

BLACKHOLE テーブルの `INSERT` トリガーは期待どおりに機能します。しかし、実際には BLACKHOLE テーブルはデータを格納しないため、`UPDATE` および `DELETE` トリガーは有効ではありません。トリガー定義の `FOR EACH ROW` 句は、行がないために適用されません。

BLACKHOLE ストレージエンジンのその他の利用方法は、次のとおりです。

- ダンプファイル構文の検証。
- バイナリロギングのオーバーヘッドを測定 (バイナリロギングが有効である場合と有効でない場合のパフォーマンスを BLACKHOLE を利用して比較することで)。
- BLACKHOLE は本質的には「no-op」ストレージエンジンであるため、ストレージエンジン自体には関係ないパフォーマンスボトルネックの検出に使用される場合があります。

コミットされたトランザクションはバイナリログに書き込まれ、ロールバックされたトランザクションは書き込まれないという意味で、**BLACKHOLE** エンジンはトランザクション対応です。

Blackhole エンジンと自動インクリメントカラム

BLACKHOLE エンジンは **no-op** エンジンです。 **BLACKHOLE** を使用してテーブルに対して実行される操作は影響を受けません。これは、自動増分される主キーカラムの動作を考慮する際に注意する必要があります。エンジンはフィールド値を自動的に増分せず、自動増分フィールドの状態を保持しません。これは、レプリケーションで重要な意味を持ちます。

次の 3 つの条件がすべて適用される次のレプリケーションシナリオを検討します。

1. ソースサーバーには、主キーである自動増分フィールドを持つブラックホールテーブルがあります。
2. レプリカには同じテーブルが存在しますが、MyISAM エンジンを使用します。
3. ソーステーブルへの挿入は、**INSERT** ステートメント自体で自動増分値を明示的に設定せずに、または **SET INSERT_ID** ステートメントを使用して実行されます。

このシナリオでは、主キーカラムで重複エントリエラーが発生してレプリケーションが失敗します。

ステートメントベースのレプリケーションでは、コンテキストイベントの **INSERT_ID** の値は常に同じです。したがって、主キーカラムの値が重複する行を挿入しようとする、レプリケーションは失敗します。

行ベースのレプリケーションでは、エンジンが戻す行の値は、各挿入で常に同じです。この結果、レプリカは主キーカラムに同じ値を使用して 2 つの挿入ログエントリをリプレイしようとするため、レプリケーションは失敗します。

カラムのフィルタリング

行ベースのレプリケーション (**binlog_format=ROW**) を使用する場合は、[セクション 17.5.1.9 「ソースとレプリカで異なるテーブル定義を使用したレプリケーション」](#) のセクションで説明されているように、最後のカラムがテーブルから欠落しているレプリカがサポートされます。

このフィルタリングはレプリカ側で機能します。つまり、カラムはフィルタで除外される前にレプリカにコピーされます。カラムをレプリカにコピーすることは望ましくないケースが 2 つ以上あります：

1. データが機密である場合、レプリカサーバーはそのデータにアクセスできません。
2. ソースに多数のレプリカがある場合、レプリカに送信する前にフィルタリングすると、ネットワークトラフィックが削減される可能性があります。

ソースカラムのフィルタリングは、**BLACKHOLE** エンジンを使用して実行できます。これは、ソーステーブルのフィルタリングの実現方法と同様の方法で実行されます - **BLACKHOLE** エンジンおよび **--replicate-do-table** または **--replicate-ignore-table** オプションを使用します。

ソースの設定は次のとおりです：

```
CREATE TABLE t1 (public_col_1, ..., public_col_N,  
secret_col_1, ..., secret_col_M) ENGINE=MyISAM;
```

信頼できるレプリカの設定は次のとおりです：

```
CREATE TABLE t1 (public_col_1, ..., public_col_N) ENGINE=BLACKHOLE;
```

信頼できないレプリカの設定は次のとおりです：

```
CREATE TABLE t1 (public_col_1, ..., public_col_N) ENGINE=MyISAM;
```

16.7 MERGE ストレージエンジン

MRG_MyISAM エンジンとしても知られている **MERGE** ストレージエンジンは、1 つのテーブルとして使用できる同一の **MyISAM** テーブルの集まりです。「同一」は、すべてのテーブルのカラムデータ型とインデックス情報が同一であることを意味します。カラムが異なる順序でリストされている **MyISAM** テーブル、対応するカラムにまったく同じデータ型がない **MyISAM** テーブル、またはインデックスが異なる順序である **MyISAM** テーブルはマー

じできません。しかし、MyISAM テーブルのすべてまたはいずれかを `mysampack` で圧縮できます。セクション 4.6.6 「`mysampack` — 圧縮された読み取り専用の MyISAM テーブルの生成」を参照してください。次のようなテーブルの違いは関係ありません:

- 対応するカラムおよびインデックスの名前は異なる場合があります。
- テーブル、カラムおよびインデックスのコメントは異なる場合があります。
- `AVG_ROW_LENGTH`、`MAX_ROWS`、`PACK_KEYS` などのテーブルオプションは異なる場合があります。

MERGE テーブルのかわりにパーティションテーブルを使用すると、単一のテーブルのパーティションを別々のファイルに格納し、一部の操作をより効率的に実行できます。詳細については、第24章「パーティション化」を参照してください。

MERGE テーブルを作成すると、MySQL によって、基礎となる MyISAM テーブルの名前を含む `.MRG` ファイルがディスク上に作成されます。MERGE テーブルのテーブル形式は、MySQL データディクショナリに格納されます。基礎となるテーブルは、MERGE テーブルと同じデータベース内にある必要はありません。

MERGE テーブルでは、`SELECT`、`DELETE`、`UPDATE`、および `INSERT` を使用できます。MERGE テーブルにマッピングする MyISAM テーブルに対して `SELECT`、`DELETE`、および `UPDATE` 権限が必要です。

注記

MERGE テーブルの利用は、次のセキュリティに関する問題を引き起こします。ユーザーが MyISAM テーブル `t` に対するアクセス権限を持っていると、そのユーザーは `t` にアクセスできる MERGE テーブル `m` を作成できます。しかし、`t` に対するユーザーの権限があつて破棄された場合、ユーザーは `m` を介してアクセスすることで `t` にアクセスを続けることができます。

`DROP TABLE` を MERGE テーブルに使用すると、MERGE 指定だけが削除されます。基礎テーブルは影響を受けません。

MERGE テーブルを作成するには、どの MyISAM テーブルを使用するかを示す `UNION=(list-of-tables)` オプションを指定する必要があります。オプションとして、`INSERT_METHOD` オプションを指定して MERGE テーブルへの挿入方法を制御できます。`FIRST` または `LAST` の値を使用すると、それぞれ最初のまたは最後の基礎テーブルで挿入が実行されることとなります。`INSERT_METHOD` オプションを指定しないか、または値 `NO` 付きでこのオプションを指定すると、MERGE テーブルへの挿入は許可されず、挿入の試みはエラーとなります。

次の例は、MERGE テーブルの作成方法を紹介しています。

```
mysql> CREATE TABLE t1 (  
-> a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,  
-> message CHAR(20)) ENGINE=MyISAM;  
mysql> CREATE TABLE t2 (  
-> a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,  
-> message CHAR(20)) ENGINE=MyISAM;  
mysql> INSERT INTO t1 (message) VALUES ('Testing'),('table'),('t1');  
mysql> INSERT INTO t2 (message) VALUES ('Testing'),('table'),('t2');  
mysql> CREATE TABLE total (  
-> a INT NOT NULL AUTO_INCREMENT,  
-> message CHAR(20), INDEX(a))  
-> ENGINE=MERGE UNION=(t1,t2) INSERT_METHOD=LAST;
```

カラム `a` は、基礎となる MyISAM テーブルでは `PRIMARY KEY` としてインデックス付けされますが、MERGE テーブルではインデックス付けされません。MERGE テーブルは基礎テーブルセットに一意性を適用できないため、インデックスは設定されますが、`PRIMARY KEY` としては設定されません。(同様に、基礎テーブルで `UNIQUE` インデックスを持つカラムには、MERGE テーブルでインデックスが付けられますが、`UNIQUE` インデックスとしては付けられないはずです。)

MERGE テーブルを作成したあと、このテーブルを使用して、テーブルのグループにまとめて操作を行うクエリーを発行できます。

```
mysql> SELECT * FROM total;  
+--+-----+  
| a | message |
```

```
+--+-----+
| 1 | Testing |
| 2 | table  |
| 3 | t1    |
| 1 | Testing |
| 2 | table  |
| 3 | t2    |
+--+-----+
```

MERGE テーブルを **MyISAM** テーブルの別のコレクションに対して再マッピングするには、次のいずれかの方法を利用できます。

- MERGE テーブルを **DROP** して、再作成する。
- 基礎テーブルのリストを変更するために、**ALTER TABLE tbl_name UNION=(...)** を利用する。

ALTER TABLE ... UNION=() (つまり、空の **UNION** 句) を使用してすべての基礎テーブルを削除することもできます。ただしこの場合、テーブルは実質的には空であり、新しい行を取得する基礎テーブルがないために挿入は失敗します。このようなテーブルは、新しい **MERGE** テーブルを **CREATE TABLE ... LIKE** で作成するためのテンプレートとして役立つ場合があります。

基礎テーブルの定義とインデックスは、**MERGE** テーブルの定義と厳密に一致する必要があります。一致がチェックされるのは、**MERGE** テーブルが作成されたときではなく、**MERGE** テーブルの一部のテーブルが開いたときです。いずれのテーブルも一致チェックに失敗した場合、テーブルのオープンをトリガーした操作は失敗します。すなわち、**MERGE** 内のテーブルの定義を変更すると、**MERGE** テーブルがアクセスされたときに失敗の原因となる可能性があります。それぞれのテーブルに適用される一致チェックは次のとおりです。

- 基礎テーブルと **MERGE** テーブルのカラム数は同じでなければいけません。
- 基礎テーブルと **MERGE** テーブルのカラムの順番は一致する必要があります。
- また、親の **MERGE** テーブル内の対応する各カラムの指定と基礎テーブルの指定を比較して、次のチェック内容を満たす必要があります。
 - 基礎テーブルと **MERGE** テーブルのカラムの型は一致する必要がある。
 - 基礎テーブルと **MERGE** テーブルのカラムの長さは一致する必要がある。
 - 基礎テーブルと **MERGE** テーブルのカラムは **NULL** でもかまわない。
- 基礎テーブルは、少なくとも **MERGE** テーブルと同じ数のインデックスを持つ必要がある。基礎テーブルのインデックスの数は **MERGE** テーブルより多くてもかまわないが、少なくすることはできない。

注記

同じカラムのインデックスは、**MERGE** テーブルと基礎 **MyISAM** テーブルの両方でまったく同じ順番でなければならないという、既知の問題が存在します。バグ #33653 を参照してください。

各インデックスは次のチェック内容を満たす必要があります。

- 基礎テーブルと **MERGE** テーブルのインデックスの型は一致する必要がある。
- 基礎テーブルと **MERGE** テーブルのインデックス定義でのインデックス部の数 (すなわち、複合インデックス内に複数のカラム) は一致する必要があります。
- 各インデックス部について。
 - インデックス部の長さは同じでなければいけない。
 - インデックス部の型は同じでなければいけない。
 - インデックス部の言語は同じでなければいけない。
 - インデックス部が **NULL** でかまわないかどうかをチェックする。

MERGE テーブルが基礎テーブルの問題のために、開いたり使用したりできない場合、CHECK TABLE は問題の原因となったテーブルに関する情報を表示します。

追加のリソース

- MERGE ストレージエンジンに特化したフォーラムは、<https://forums.mysql.com/list.php?93>で参照できます。

16.7.1 MERGE テーブルの長所と短所

MERGE テーブルは、次のような問題を解決するのに役立つことがあります。

- ログテーブルセットを簡単に管理する。たとえば、異なる月のデータを別々のテーブルに入力し、`myisampack` を利用してそれらの一部を圧縮してから、1つのものとして利用するために MERGE テーブルを作成できます。
- スピードを上げる。大きな読み取り専用テーブルを同じ基準で分割し、個々のテーブルを異なるディスクに置くことができます。このように構成された MERGE テーブルは、1つの大きなテーブルを使用するよりも、速度がかなり速くなる可能性があります。
- より効率的に検索を行う。検索する対象が正確にわかっている場合、あるクエリーで基礎テーブルの1つだけを検索し、別のクエリーで MERGE テーブルを使用できます。重複するテーブルセットを使用する、多数の異なる MERGE テーブルを持つこともできます。
- より効率的な修復を行う。1つの大きなテーブルを修復するよりも、MERGE テーブルにマッピングされた個々の小さいテーブルを修復する方が簡単です。
- 多くのテーブルを瞬時に1つのテーブルとしてマッピングする。MERGE テーブルは個々のテーブルのインデックスを利用するので、それ自体のインデックスを保守する必要はありません。その結果、MERGE テーブルコレクションは、作成や再マッピングを非常に速く行うことができます。(MERGE テーブルを作成するときは、インデックスが作成されない場合でも、インデックスの定義を指定する必要があります。)
- テーブルセットがあり、それらからオンデマンドで大きなテーブルを作成する場合は、代わりにそれらからオンデマンドで MERGE テーブルを作成できます。この方が、速度がかなり速くなり、多くのディスク容量が節約されます。
- オペレーティングシステムのファイルサイズ制限を超える。個々の MyISAM テーブルはこの制限に制約されますが、MyISAM テーブルのコレクションは制約されません。
- MyISAM テーブルにマッピングする MERGE テーブルを定義することで、その単一テーブルのエイリアスやシノニムを作成できます。これを行っても、特に顕著なパフォーマンス面の影響はないはずですが(個々の読み取りのためにいくつかの間接呼び出しや `memcpy()` 呼び出しがあるだけです)。

MERGE テーブルの短所は次のとおりです。

- MERGE テーブルに対して、同一の MyISAM テーブルしか使用できません。
- MyISAM 機能のいくつかは MERGE テーブルでは使用できません。たとえば、MERGE テーブルに FULLTEXT インデックスを作成できません。(基礎 MyISAM テーブルに FULLTEXT インデックスを作成できますが、MERGE テーブルを全文検索で検索できません。)
- MERGE テーブルが非一時的である場合、すべての基礎 MyISAM テーブルは非一時的である必要があります。MERGE テーブルが一時的である場合、MyISAM テーブルは一時的なテーブルと非一時的なテーブルが混在してもかまいません。
- MERGE テーブルは MyISAM テーブルより多くのファイルディスクリプタを使用します。10個のクライアントが、10個のテーブルにマッピングする MERGE テーブルを使用する場合、サーバーは $(10 \times 10) + 10$ 個のファイルディスクリプタを使用します。(10個のクライアントに対してそれぞれ10個のデータファイルディスクリプタに加えて、クライアント間で共有される10個のインデックスファイルディスクリプタです。)
- インデックスの読み取りは低下します。インデックスを読み取るときに、MERGE ストレージエンジンはすべての基礎テーブルに読み取りを発行して、渡されたインデックス値に厳密に一致するかをチェックする必要があります。次のインデックス値を読み取るために、MERGE ストレージエンジンは読み取りバッファを検索して次の値を探す必要があります。1つのインデックスバッファが使果たされていた場合にのみ、ストレージエンジンは次のインデックスブロックを読み取る必要があります。これで、MERGE インデックスは `eq_ref` 検索を

かなり遅くしますが、`ref` 検索ではそれほど低下しません。 `eq_ref` および `ref` の詳細情報については、[セクション 13.8.2 「EXPLAIN ステートメント」](#) を参照してください。

16.7.2 MERGE テーブルの問題点

MERGE テーブルの既知の問題点は次のとおりです。

- 5.1.23 バージョンより前の MySQL Server では、MyISAM の非一時的な子供のテーブルを持つ一時的なマージテーブルを作成できました。

バージョン 5.1.23 からは、MERGE の子供は親のテーブルを介してロックされました。親が一時的であった場合、親がロックされなかったため、子供もロックされませんでした。MyISAM テーブルを同時に使用すると、テーブルが破損しました。

- `ALTER TABLE` を使用して MERGE テーブルを別のストレージエンジンに変えると、基礎テーブルへのマッピングが失われます。その代わりに、変更されたテーブルに基礎 MyISAM テーブルの行がコピーされ、そのときに、指定されたストレージエンジンを使用します。
- MERGE テーブルの `INSERT_METHOD` テーブルオプションは、MERGE テーブルへの挿入にどの基礎 MyISAM テーブルを使用するかを示します。ただし、その MyISAM テーブルに `AUTO_INCREMENT` テーブルオプションを使用しても、1 つ以上の行が MyISAM テーブルに直接挿入されるまで、MERGE テーブルへの挿入は有効になりません。
- MERGE テーブルは、テーブル全体に一意制約を保持できません。 `INSERT` を実行すると、データは最初または最後の MyISAM テーブル (`INSERT_METHOD` オプションで指定されます) に入ります。MySQL は一意のキー値が MyISAM テーブル内で一意のままであることを保証しますが、コレクション内のすべての基礎テーブルには保証しません。
- MERGE エンジンは基本テーブルセットに一意性を適用できないため、`REPLACE` は期待どおりに機能しません。次の 2 つの重要な事実があります。
 - `REPLACE` は、書き込もうとする基礎テーブルでのみ一意キー違反を検出できます (`INSERT_METHOD` オプションで指定されます)。これは MERGE テーブル自体の違反とは異なります。
 - `REPLACE` は、一意キー違反を検出すると、書込み先の基礎となるテーブル (つまり、`INSERT_METHOD` オプションで決定された最初または最後のテーブル) の対応する行のみを変更します。

`INSERT ... ON DUPLICATE KEY UPDATE` についても同様な考慮が適用されます。

- MERGE テーブルはパーティション化をサポートしていません。つまり、MERGE テーブルも、MERGE テーブルの基礎 MyISAM テーブルもパーティション化できません。
- 開いた MERGE テーブルにマッピングされたどのテーブルにも、`ANALYZE TABLE`、`REPAIR TABLE`、`OPTIMIZE TABLE`、`ALTER TABLE`、`DROP TABLE`、`DELETE (WHERE 句なし)`、または `TRUNCATE TABLE` を使用するべきではありません。そうする場合、MERGE テーブルは引き続き元のテーブルを参照しているため、予期しない結果となる可能性があります。この問題に対処するには、名前付きの操作を行う前に `FLUSH TABLES` ステートメントを発行することで、確実に MERGE テーブルが開いたままにならないようにします。

予期しない結果には、MERGE テーブルに対する操作でテーブルの破損が報告される可能性があります。基礎 MyISAM テーブルで名前付き操作のあとにこれが発生した場合、破損メッセージは偽りです。これに対処するには、MyISAM テーブルを変更したあとで `FLUSH TABLES` ステートメントを発行します。

- MERGE ストレージエンジンのテーブルマッピングは MySQL の上位レイヤーから隠れているので、MERGE テーブルによって使用されているテーブルでの `DROP TABLE` は Windows では機能しません。Windows では開いているファイルの削除を許可しないため、最初にすべての MERGE テーブルをフラッシュするか (`FLUSH TABLES` を使用します)、テーブルを削除する前に MERGE テーブルを削除する必要があります。
- MyISAM テーブルと MERGE テーブルの定義は、テーブルにアクセスするときにチェックされます (たとえば、`SELECT` または `INSERT` ステートメントの一部として)。このチェックは、テーブルの定義と親の MERGE テーブルの定義が、カラムの順番、タイプ、サイズ、および関連するインデックスを比較することで一致することを保証します。テーブル間で違いがある場合、エラーが戻され、ステートメントは失敗します。これらのチェックはテーブルが開かれたときに行われるため、カラムの変更、カラムの順序付け、エンジンの変更など、単一のテーブルの定義を変更すると、ステートメントが失敗します。

- **MERGE** テーブルと、その基礎テーブルのインデックスの順番は同一でなければいけません。 **ALTER TABLE** を使用して、**MERGE** テーブル内で使用されるテーブルに **UNIQUE** インデックスを追加し、次に **ALTER TABLE** を使用して **MERGE** テーブル上に一意でないインデックスを追加した場合、基礎テーブル内に一意でないインデックスがすでに存在していると、それらのテーブルのインデックス順序は異なります。(これが発生するのは、重複キーをすばやく検出できるように **ALTER TABLE** が一意でないインデックスの前に **UNIQUE** インデックスを配置するためです。) その結果、このようなインデックスを持つテーブルに対するクエリーは予想外の結果をもたらす可能性があります。
- **ERROR 1017 (HY000): Can't find file: 'tbl_name.MRG' (errno: 2)** エラーメッセージが表示された場合、一般的に、いくつかの基礎テーブルが **MyISAM** ストレージエンジンを使用していないことを表しています。これらのテーブルがすべて **MyISAM** であることを確認してください。
- **MERGE** テーブルの行の最大値は 2^{64} です (~1.844E+19 で、**MyISAM** テーブルの場合と同じ)。複数の **MyISAM** テーブルを、この数よりも多くの行を含む単一の **MERGE** テーブルにマージできません。
- 親の **MERGE** テーブルを持つ、異なる行フォーマットの基礎 **MyISAM** テーブルを使用すると、現在失敗することが知られています。バグ #32364 を参照してください。
- **LOCK TABLES** が実施されているときは、非一時的な **MERGE** テーブルの結合リストを変更できません。次は動作しません。

```
CREATE TABLE m1 ... ENGINE=MRG_MYISAM ...;  
LOCK TABLES t1 WRITE, t2 WRITE, m1 WRITE;  
ALTER TABLE m1 ... UNION=(t1,t2) ...;
```

ただし、一時的な **MERGE** テーブルではこれを行うことができます。

- 一時的な **MERGE** としても、非一時的な **MERGE** テーブルとしても、**CREATE ... SELECT** で **MERGE** テーブルを作成できません。例:

```
CREATE TABLE m1 ... ENGINE=MRG_MYISAM ... SELECT ...;
```

これを試みると、**tbl_name** は **BASE TABLE** ではないというエラーとなります。

- あるケースでは、**MERGE** と基礎テーブル間で **PACK_KEYS** テーブルオプション値が異なると、基礎テーブルに **CHAR** または **BINARY** カラムが含まれている場合、予期しない結果になります。回避策として、**ALTER TABLE** を使用して、関係するすべてのテーブルの **PACK_KEYS** 値が同じであることを保証します。(Bug #50646)

16.8 FEDERATED ストレージエンジン

FEDERATED ストレージエンジンを使用すると、レプリケーションまたはクラスタの技術を使用しないで、リモートの MySQL データベースのデータにアクセスできます。ローカルの **FEDERATED** テーブルにクエリーを発行すると、リモート(連合)テーブルからデータを自動的に取得します。データはローカルテーブルに格納されません。

ソースから MySQL を構築する場合に **FEDERATED** ストレージエンジンを含めるには、**CMake** を **-DWITH_FEDERATED_STORAGE_ENGINE** オプションで呼び出します。

FEDERATED ストレージエンジンは、デフォルトでは動作中のサーバーで有効になっていません。**FEDERATED** を有効にするには、**--federated** オプションを使用して MySQL サーバーバイナリを起動する必要があります。

FEDERATED エンジンのソースを調べるには、MySQL ソース配布の **storage/federated** ディレクトリを検索します。

16.8.1 FEDERATED ストレージエンジンの概要

標準のストレージエンジン (**MyISAM**、**CSV**、**InnoDB** など) のいずれかを使用してテーブルを作成すると、そのテーブルはテーブルの定義と関連データで構成されます。**FEDERATED** テーブルを作成すると、テーブル定義は同じですが、データの物理ストレージはリモートサーバーで処理されます。

FEDERATED テーブルは 2 つの要素で構成されます。

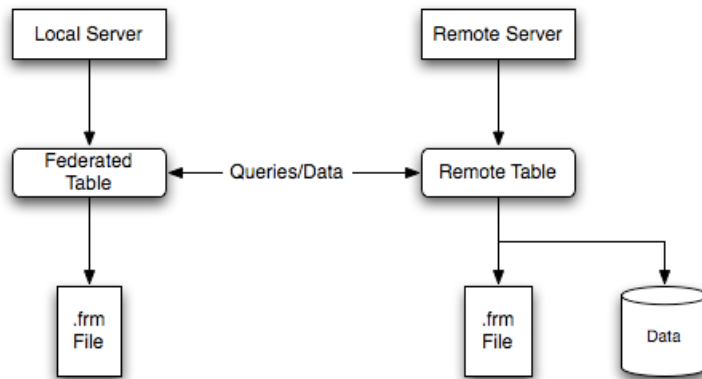
- データベーステーブルを含むリモートサーバー。これは、(MySQL データディクショナリに格納されている) テーブル定義と関連するテーブルで構成されます。リモートテーブルのテーブルタイプは、**MyISAM** や **InnoDB** を含む、リモート **mysqld** サーバーがサポートするいずれのタイプであってもかまいません。

- データベーステーブルを持つローカルサーバー。テーブルの定義は、リモートサーバー上の対応するテーブルの定義に一致します。テーブル定義はデータディクショナリに格納されます。ローカルサーバーにデータファイルがありません。その代わりに、テーブル定義にはリモートテーブルをポイントする接続文字列が含まれています。

ローカルサーバーで **FEDERATED** テーブルにクエリーやステートメントを実行すると、一般的にローカルデータファイルの情報を挿入、更新、または削除する操作は、実行するために代わりにリモートサーバーに送られ、そこでリモートサーバーのデータファイルを更新したり、リモートサーバーから一致する行を戻したりします。

FEDERATED テーブルのセットアップの基本的な構造は 図16.2 「**FEDERATED テーブルの構造**」で示します。

図 16.2 FEDERATED テーブルの構造



FEDERATED テーブルを参照する SQL ステートメントをクライアントが発行する場合、ローカルサーバー (SQL ステートメントが実行される場所) とリモートサーバー (データが物理的に格納される場所) の間の情報の流れは次のとおりです。

- ストレージエンジンは **FEDERATED** テーブルが持つ各カラムを調べて、リモートテーブルを参照する適切な SQL ステートメントを構築します。
- ステートメントは MySQL クライアント API を使用してリモートサーバーに送られます。
- リモートサーバーはステートメントを処理し、ローカルサーバーはステートメントが作成した結果 (影響を受けた行の数や結果セット) を取得します。
- ステートメントが結果セットを作成する場合、各カラムは **FEDERATED** エンジンが求める内部ストレージエンジン形式に変換され、元のステートメントを発行したクライアントに結果を表示するために使用できます。

ローカルサーバーは、MySQL クライアントの C API 関数を使用してリモートサーバーと通信します。 `mysql_real_query()` を呼び出して、ステートメントを送信します。結果セットを読み取るために、 `mysql_store_result()` を使用し、 `mysql_fetch_row()` を使用して 1 つずつ行をフェッチします。

16.8.2 FEDERATED テーブルの作成方法

FEDERATED テーブルを作成するときは、次の手順に従うようにしてください。

- リモートサーバーにテーブルを作成します。または、 `SHOW CREATE TABLE` ステートメントを使用するなどして、既存テーブルのテーブル定義のメモを取ります。
- 同一のテーブル定義でローカルサーバーにテーブルを作成しますが、ローカルテーブルをリモートテーブルにリンクする接続情報を追加してください。

たとえば、リモートサーバーに次のテーブルを作成できます。

```
CREATE TABLE test_table (
  id INT(20) NOT NULL AUTO_INCREMENT,
  name VARCHAR(32) NOT NULL DEFAULT "",
```



```
other INT(20) NOT NULL DEFAULT '0',
PRIMARY KEY (id),
INDEX name (name),
INDEX other_key (other)
)
ENGINE=MyISAM
DEFAULT CHARSET=utf8mb4;
```

リモートテーブルにフェデレートされるローカルテーブルを作成するには、2つのオプションを使用できます。ローカルテーブルを作成し、**CONNECTION** を使用してリモートテーブルへの接続に使用される接続文字列 (サーバー名、ログイン、パスワードを含みます) を指定するか、**CREATE SERVER** ステートメントを使用してすでに作成された既存の接続を使用できます。

重要

ローカルテーブルを作成する場合、リモートテーブルに同一のフィールド定義を持つ必要があります。

注記

インデックスをホストのテーブルに追加することで、**FEDERATED** テーブルのパフォーマンスを向上できます。最適化は、リモートサーバーに送信されるクエリーに **WHERE** 句の内容が含まれ、リモートサーバーに送信されてからローカルで実行されるために行われます。これにより、そうしないとローカル処理のためにサーバーからテーブル全体を要求することになるネットワークトラフィックが削減されます。

16.8.2.1 CONNECTION を使用した FEDERATED テーブルの作成

最初の方法を使用するには、**CREATE TABLE** ステートメントのエンジンタイプの後ろに **CONNECTION** 文字列を指定する必要があります。例:

```
CREATE TABLE federated_table (
  id INT(20) NOT NULL AUTO_INCREMENT,
  name VARCHAR(32) NOT NULL DEFAULT "",
  other INT(20) NOT NULL DEFAULT '0',
  PRIMARY KEY (id),
  INDEX name (name),
  INDEX other_key (other)
)
ENGINE=FEDERATED
DEFAULT CHARSET=utf8mb4
CONNECTION='mysql://fed_user@remote_host:9306/federated/test_table';
```

注記

CONNECTION は MySQL の以前のバージョンで使われた **COMMENT** を置き換えるものです。

CONNECTION 文字列には、データが物理的に存在するテーブルを含むリモートサーバーへの接続に必要な情報が含まれます。接続文字列には、サーバー名、ログイン資格証明、ポート番号、およびデータベース/テーブル情報を指定します。この例では、リモートテーブルはサーバー **remote_host** 上にあり、ポート 9306 を使用します。名前とポート番号は、リモートテーブルとして使用するリモート MySQL サーバーのホスト名 (または IP アドレス) とポート番号に一致する必要があります。

接続文字列の書式は次のとおりです。

```
scheme://user_name[:password]@host_name[:port_num]/db_name/tbl_name
```

ここでは:

- **scheme**: 認識された接続プロトコル。この時点では、**mysql** だけが **scheme** 値としてサポートされています。
- **user_name**: 接続のためのユーザー名。このユーザーは、リモートサーバー上に作成されている必要があります、リモートテーブルで必要なアクション (**SELECT**、**INSERT**、**UPDATE** など) を実行するのに適した権限を持つ必要があります。

- **password:** (オプション) `user_name` に対応するパスワード。
- **host_name:** リモートサーバーのホスト名または IP アドレス。
- **port_num:** (オプション) リモートサーバーのポート番号。デフォルトは 3306 です。
- **db_name:** リモートテーブルを保持するデータベースの名前。
- **tbl_name:** リモートテーブルの名前。ローカルテーブルとリモートテーブルの名前が一致する必要はありません。

接続文字列の例は次のとおりです。

```
CONNECTION='mysql://username:password@hostname:port/database/tablename'  
CONNECTION='mysql://username@hostname/database/tablename'  
CONNECTION='mysql://username:password@hostname/database/tablename'
```

16.8.2.2 CREATE SERVER を使用した FEDERATED テーブルの作成

多くの **FEDERATED** テーブルを同じサーバーに作成する場合、または **FEDERATED** テーブルの作成プロセスを単純化する場合、**CREATE SERVER** ステートメントを使用してサーバー接続パラメータを定義できます (**CONNECTION** 文字列の場合と同様)。

CREATE SERVER ステートメントの書式は次のとおりです。

```
CREATE SERVER  
server_name  
FOREIGN DATA WRAPPER wrapper_name  
OPTIONS (option [, option] ...)
```

`server_name` は **FEDERATED** テーブルを作成するときに接続文字列で使用されます。

たとえば **CONNECTION** 文字列と同一のサーバー接続を作成するには、次のとおりです。

```
CONNECTION='mysql://fed_user@remote_host:9306/federated/test_table';
```

次のステートメントを使用することになります。

```
CREATE SERVER fedlink  
FOREIGN DATA WRAPPER mysql  
OPTIONS (USER 'fed_user', HOST 'remote_host', PORT 9306, DATABASE 'federated');
```

この接続を使用する **FEDERATED** テーブルを作成するには、**CONNECTION** キーワードも使用しますが、**CREATE SERVER** ステートメントで使用した名前を指定します。

```
CREATE TABLE test_table (  
  id INT(20) NOT NULL AUTO_INCREMENT,  
  name VARCHAR(32) NOT NULL DEFAULT "",  
  other INT(20) NOT NULL DEFAULT '0',  
  PRIMARY KEY (id),  
  INDEX name (name),  
  INDEX other_key (other)  
)  
ENGINE=FEDERATED  
DEFAULT CHARSET=utf8mb4  
CONNECTION='fedlink/test_table';
```

この例の接続名には、接続の名前 (`fedlink`) とリンクするテーブルの名前 (`test_table`) が含まれます (区切りはスラッシュ)。テーブル名なしで接続名だけを指定した場合、代わりにローカルテーブルのテーブル名が使用されます。

CREATE SERVER の詳細情報については、[セクション13.1.18「CREATE SERVER ステートメント」](#)を参照してください。

CREATE SERVER ステートメントは、**CONNECTION** 文字列と同じ引数を受け入れます。**CREATE SERVER** ステートメントは `mysql.servers` テーブルの中の行を更新します。接続文字列のパラメータ間の通信、**CREATE SERVER** ステートメントのオプション、および `mysql.servers` テーブルのカラムに関する情報については、次の表を参照してください。参考までに、**CONNECTION** 文字列の書式は次のとおりです。

```
scheme://user_name[:password]@host_name[:port_num]/db_name/tbl_name
```

説明	CONNECTION 文字列	CREATE SERVER オプション	mysql.servers カラム
接続スキーム	scheme	wrapper_name	Wrapper
リモートユーザー	user_name	USER	Username
リモートパスワード	password	PASSWORD	Password
リモートホスト	host_name	HOST	Host
リモートポート	port_num	PORT	Port
リモートデータベース	db_name	DATABASE	Db

16.8.3 FEDERATED ストレージエンジンの注記とヒント

FEDERATED ストレージエンジンを使用するときは、次の点に注意することをお勧めします。

- FEDERATED テーブルは他のレプリカにレプリケートできますが、レプリカサーバーが CONNECTION 文字列 (または mysql.servers テーブルの行) に定義されているユーザー/パスワードの組合せを使用してリモートサーバーに接続できることを確認する必要があります。

次の項目は、FEDERATED ストレージエンジンがサポートしている機能とサポートしていない機能を示します。

- リモートサーバーは MySQL サーバーでなくてはなりません。
- FEDERATED テーブルがポイントするリモートテーブルは、FEDERATED テーブルを介してそのテーブルにアクセスを試みる前に、存在している必要があります。
- ある FEDERATED テーブルがほかのテーブルをポイントすることは可能ですが、ループを作らないように注意する必要があります。
- FEDERATED テーブルは通常の意味でインデックスをサポートしていません。テーブルデータへのアクセスはリモートで処理されるため、実際にはインデックスを使用するリモートテーブルです。つまり、インデックスを使用できないために全テーブルスキャンが必要なクエリーの場合、サーバーはリモートテーブルからすべての行をフェッチし、ローカルでフィルタします。これは、この SELECT ステートメントで使用される WHERE または LIMIT に関係なく発生します。これらの句は、戻される行にローカルに適用されます。

したがって、インデックスの使用に失敗したクエリーは、パフォーマンスおよびネットワークオーバーロードの低下を引き起こす可能性があります。また、返される行はメモリーに格納する必要があるため、このようなクエリーによってローカルサーバーのスワッピングやハングアップが発生することもあります。

- 同等の MyISAM やほかのテーブルからのインデックス定義がサポートされていない可能性があるため、FEDERATED テーブルを作成するときは注意を払うようにしてください。たとえば、テーブルが VARCHAR、TEXT または BLOB カラムでインデックス接頭辞を使用している場合、FEDERATED テーブルの作成は失敗します。MyISAM を使用した次の定義は有効です:

```
CREATE TABLE `T1`(`A` VARCHAR(100),UNIQUE KEY(`A` (30))) ENGINE=MYISAM;
```

この例のキー接頭辞は FEDERATED エンジンと互換性がなく、同等のステートメントが失敗します:

```
CREATE TABLE `T1`(`A` VARCHAR(100),UNIQUE KEY(`A` (30))) ENGINE=FEDERATED  
CONNECTION='MYSQL://127.0.0.1:3306/TEST/T1';
```

可能であれば、これらのインデックスの問題を回避するため、リモートサーバーとローカルサーバーの両方にテーブルを作成する場合、カラムとインデックスの定義を分けるようにしてください。

- 内部的に、実装は SELECT、INSERT、UPDATE、および DELETE を使用しますが、HANDLER は使用しません。
- FEDERATED ストレージエンジンは、SELECT、INSERT、UPDATE、DELETE、TRUNCATE TABLE、およびインデックスをサポートしています。DROP TABLE を除いて、ALTER TABLE や、テーブルの構造に直接影響を与えるデータ定義言語ステートメントをサポートしていません。現在の実装は、プリペアドステートメントを使用しません。

- **FEDERATED** は `INSERT ... ON DUPLICATE KEY UPDATE` ステートメントを受け入れますが、重複キー違反が起こった場合、ステートメントはエラーで失敗します。
- トランザクションはサポートされていません。
- **FEDERATED** は、複数の行がバッチでリモートテーブルに送信されるように一括挿入処理を実行するため、パフォーマンスが向上します。また、リモートテーブルがトランザクション対応の場合、エラーが発生したときにリモートストレージエンジンはステートメントロールバックを適切に実行できます。この機能には次の制限があります。
 - 挿入のサイズは、サーバー間の最大パケットサイズを超えることはできません。挿入がこのサイズを超えた場合、複数のパケットに分割され、ロールバック問題が発生する可能性があります。
 - 大量挿入処理は `INSERT ... ON DUPLICATE KEY UPDATE` では起こりません。
- **FEDERATED** エンジンは、リモートテーブルが変わったかどうかを知る方法がありません。その理由は、このテーブルが、データベースシステム以外の何かによって決して書き込まれることのないデータファイルのように動作する必要があるためです。リモートデータベースに変更が加えられた場合に、ローカルテーブルのデータの完全性が損なわれる可能性があります。
- **CONNECTION** 文字列を使用する場合、パスワードに '@' 文字を使用できません。 `CREATE SERVER` ステートメントを使用してサーバー接続を作成することで、この制限を回避できます。
- `insert_id` および `timestamp` オプションはデータプロバイダには伝達されません。
- **FEDERATED** テーブルに対して発行された `DROP TABLE` ステートメントは、ローカルテーブルだけを削除し、リモートテーブルは削除しません。
- **FEDERATED** テーブルはクエリーキャッシュでは機能しません。
- ユーザー定義のパーティション化は、**FEDERATED** テーブルではサポートされていません。

16.8.4 FEDERATED ストレージエンジンのリソース

次の追加リソースは、**FEDERATED** ストレージエンジンで利用できます。

- **FEDERATED** ストレージエンジンに特化したフォーラムは <https://forums.mysql.com/list.php?105> で参照できます。

16.9 EXAMPLE ストレージエンジン

EXAMPLE ストレージエンジンは、何もしないスタブエンジンです。その目的は、新しいストレージエンジンの書き込みを開始する方法を示す MySQL ソースコードの例として機能することです。このため、主に開発者を対象としています。

ソースから MySQL を構築する場合に **EXAMPLE** ストレージエンジンを有効にするには、`CMake` を `-DWITH_EXAMPLE_STORAGE_ENGINE` オプションで呼び出します。

EXAMPLE エンジンのソースを調べるには、MySQL ソース配布の `storage/example` ディレクトリを検索します。

EXAMPLE テーブルを作成する場合、ファイルは作成されません。データをテーブルに格納できません。検索は空の結果を返します。

```
mysql> CREATE TABLE test (i INT) ENGINE = EXAMPLE;
Query OK, 0 rows affected (0.78 sec)

mysql> INSERT INTO test VALUES(1),(2),(3);
ERROR 1031 (HY000): Table storage engine for 'test' doesn't »
      have this option

mysql> SELECT * FROM test;
Empty set (0.31 sec)
```

EXAMPLE ストレージエンジンはインデックスをサポートしていません。

EXAMPLE ストレージエンジンはパーティション分割をサポートしていません。

16.10 ほかのストレージエンジン

ストレージエンジンは、カスタムストレージエンジンのインターフェースを使用したサードパーティーおよびコミュニティメンバーの別のストレージエンジンを使用できる場合があります。

サードパーティーのエンジンは MySQL によってサポートされていません。これらのエンジンに関する詳細情報、ドキュメント、インストールガイド、バグレポート、ヘルプや支援については、そのエンジンの開発者に直接お問い合わせください。

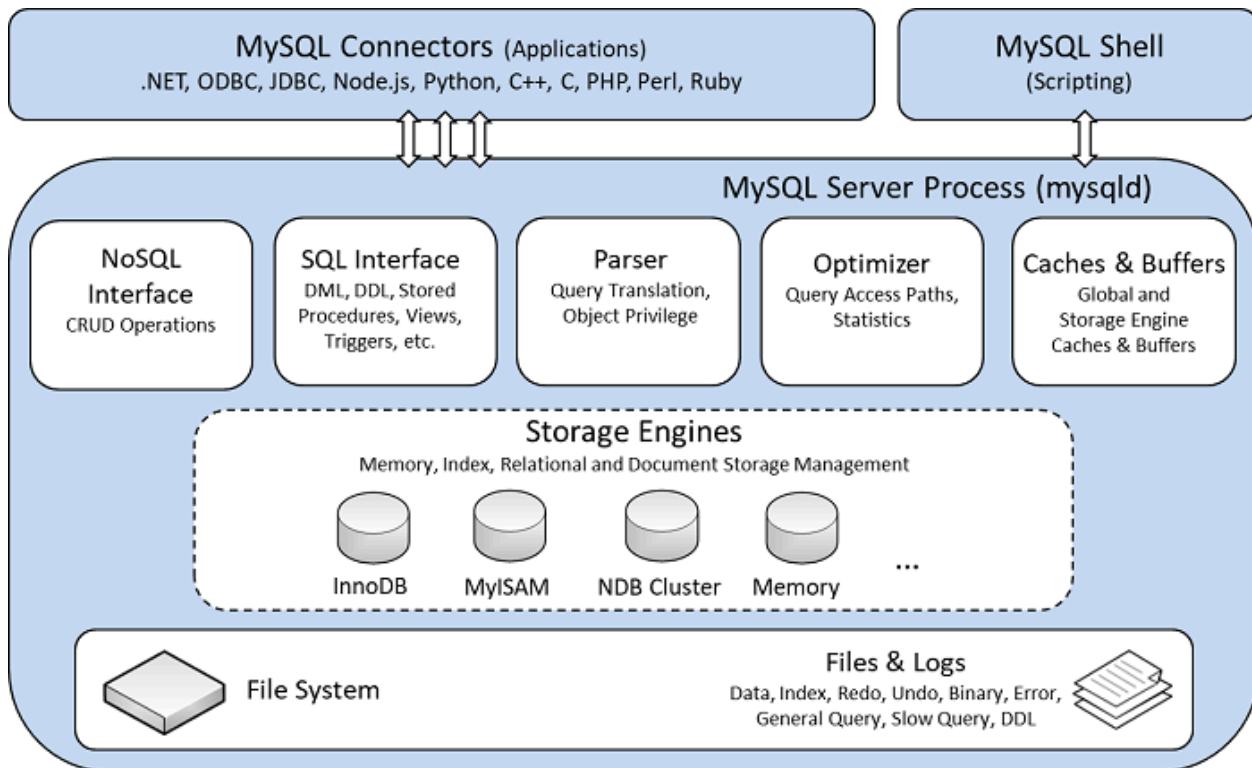
プラグブルストレージエンジンアーキテクチャーで使用できるお客様のストレージエンジンの開発に関する詳細については、「[MySQL Internals: Writing a Custom Storage Engine](#)」を参照してください。

16.11 MySQL ストレージエンジンアーキテクチャーの概要

MySQL プラグブルストレージエンジンアーキテクチャーを採用すると、データベースの専門家は、特定のアプリケーションニーズに特化したストレージエンジンを選択でき、さらにアプリケーションの特定のコーディング要件を管理する必要が完全なくなります。MySQL サーバーのアーキテクチャーにより、アプリケーションプログラマーと DBA はストレージレベルのすべての実装詳細から解放され、一貫した容易なアプリケーションモデルと API が得られます。したがって、異なるストレージエンジンの機能には違いがありますが、アプリケーションはその違いから解放されます。

MySQL プラグブルストレージエンジンのアーキテクチャは、[図16.3「プラグブルストレージエンジンを使用した MySQL アーキテクチャ」](#)に示されています。

図 16.3 プラグブルストレージエンジンを使用した MySQL アーキテクチャ



プラグブルストレージエンジンのアーキテクチャーは、すべての基になるストレージエンジンに共通の管理およびサポートサービスの標準セットを提供します。ストレージエンジン自身は、物理サーバーレベルで保守される基になるデータに対してアクションを実際に行うデータベースサーバーのコンポーネントです。

この効率的なモジュール形式のアーキテクチャーは、データウェアハウス、トランザクション処理、高可用性状況など、特別なアプリケーションニーズを特に対象にしたい人、またどれか1つのストレージエンジンに依存しないインターフェースとサービスのセットを利用するメリットを享受したい人にとって、大きなメリットが得られます。

アプリケーションプログラマと DBA は、コネクタ API およびストレージエンジンの上位にあるサービスレイヤーを介して MySQL データベースと対話します。アプリケーションの変更によって、基になるストレージエンジンの変更を求める要件が発生した場合、または新しいニーズをサポートするために 1 つ以上のストレージエンジンが追加された場合、これをうまく行うためにコーディングやプロセスを大幅に変更する必要はありません。MySQL サーバーのアーキテクチャーは、ストレージエンジンに適用される、一貫して使いやすい API を提供することで、ストレージエンジンの内在する複雑さからアプリケーションを解放します。

16.11.1 プラグブルストレージエンジンのアーキテクチャー

MySQL Server は、ストレージエンジンが、動作中の MySQL サーバーにロードされたり、MySQL サーバーからアンロードされたりできる、プラグブルストレージエンジンアーキテクチャーを採用しています。

ストレージエンジンのプラグイン

ストレージエンジンを使用する前に、`INSTALL PLUGIN` ステートメントを利用してストレージエンジンのプラグイン共有ライブラリを MySQL にロードする必要があります。たとえば、`EXAMPLE` エンジンのプラグインの名前が `example` で、共有ライブラリの名前が `ha_example.so` である場合、次のステートメントを使用してロードします。

```
INSTALL PLUGIN example SONAME 'ha_example.so';
```

プラグブルストレージエンジンをインストールするには、プラグインファイルは MySQL プラグインディレクトリにある必要があり、`INSTALL PLUGIN` ステートメントを発行するユーザーには、`mysql.plugin` テーブルの `INSERT` 権限が必要です。

共有ライブラリは MySQL サーバーのプラグインディレクトリの中にある必要があり、その場所は `plugin_dir` システム変数によって指示されます。

ストレージエンジンのアンプラグ

ストレージエンジンをアンプラグするには、`UNINSTALL PLUGIN` ステートメントを利用します。

```
UNINSTALL PLUGIN example;
```

既存のテーブルに必要なストレージエンジンを切断すると、それらのテーブルにはアクセスできなくなりますが、ディスクにはまだ存在します (該当する場合)。ストレージエンジンをアンプラグする前に、そのストレージエンジンを使用しているテーブルがないことを確認してください。

16.11.2 共通データベースサーバーレイヤー

MySQL プラグブルストレージエンジンは、データベースに対して実データの I/O 操作を行ったり、特定のアプリケーションニーズを対象とする機能セットを有効にしたり適用したりする役割を担う、MySQL データベースサーバーのコンポーネントです。特定のストレージエンジンを使用する主なメリットは、特定のアプリケーションに必要な機能だけが配布されるため、データベースのオーバーヘッドが小さくなり、データベースパフォーマンスが効率的になり向上します。これは、MySQL がこのように高パフォーマンスであると以前から知られてきた理由の 1 つであり、業界標準ベンチマークで独占的な地位を占める強力なデータベースに匹敵または対抗できる要因になっています。

技術的に見て、ストレージエンジンを支える独自のインフラストラクチャー要素は何でしょうか。機能を差別化している主な要素は次のとおりです。

- 並列性: いくつかのアプリケーションは、ほかのアプリケーションよりさらに粒度の細かいロック要件 (低レベルロックなど) を持ちます。適切なロック方式を選択すると、オーバーヘッドが低減されるため、全体のパフォーマンスが向上します。また、この分野はマルチバージョンの並列処理制御や「スナップショット」の読み込みのような機能もサポートします。
- トランザクションサポート: 必ずしもすべてのアプリケーションがトランザクションを必要としていませんが、必要な場合、ACID 準拠などの非常に明確な要件があります。
- 参照整合性: サーバーは DDL 定義の外部キーでリレーショナルデータベースの参照整合性を適用する必要があります。
- 物理ストレージ: これには、テーブルとインデックスの全体ページサイズやデータの物理ディスクへの格納に使用されるフォーマットのすべてが関係します。

- インデックスサポート: アプリケーションシナリオによっては、別のインデックス方式からメリットが得られる傾向があります。通常、各ストレージエンジンには独自のインデックス方式がありますが、ほぼすべてのエンジンに共通の方式 (B ツリーインデックスなど) もあります。
- メモリーキャッシュ: 一部のメモリーキャッシュ戦略では、一部のアプリケーションがほかのアプリケーションよりも適切に応答するため、一部のメモリーキャッシュはすべてのストレージエンジン (ユーザー接続に使用されるものなど) に共通ですが、その他のアプリケーションは特定のストレージエンジンが再生されるときにのみ一意に定義されます。
- パフォーマンスエイド: これには、並列操作のマルチ I/O スレッド、スレッド並列処理、データベースチェックポイント、大量の挿入処理などが含まれます。
- その他のターゲット機能: これには、地理空間操作、データ操作のセキュリティ面の制限事項、およびその他の類似機能に対するサポートが含まれます。

プラグブルストレージエンジンの各インフラストラクチャーコンポーネントセットは、特定のアプリケーション向けの利点を提供できるように設計されています。反対に、あるコンポーネント機能セットを回避することは、不必要なオーバーヘッドを削減するのに役立ちます。特定アプリケーションの要件セットを理解して、適切な MySQL ストレージエンジンを選択することは、当然のことながらシステム全体の効率とパフォーマンスに大きな影響を与える可能性があります。

第 17 章 レプリケーション

目次

17.1 レプリケーションの構成	3020
17.1.1 バイナリログファイルの位置ベースのレプリケーション構成の概要	3021
17.1.2 バイナリログファイルの位置ベースのレプリケーションの設定	3021
17.1.3 グローバルトランザクション識別子を使用したレプリケーション	3032
17.1.4 オンラインサーバーでの GTID モードの変更	3053
17.1.5 MySQL マルチソースレプリケーション	3059
17.1.6 レプリケーションおよびバイナリロギングのオプションと変数	3065
17.1.7 一般的なレプリケーション管理タスク	3144
17.2 レプリケーションの実装	3150
17.2.1 レプリケーション形式	3150
17.2.2 レプリケーションチャンネル	3157
17.2.3 レプリケーションスレッド	3161
17.2.4 リレーログおよびレプリケーションメタデータリポジトリ	3163
17.2.5 サーバーがレプリケーションフィルタリングルールをどのように評価するか	3169
17.3 レプリケーションのセキュリティ	3176
17.3.1 暗号化接続を使用するためのレプリケーションの設定	3177
17.3.2 バイナリログファイルとリレーログファイルの暗号化	3179
17.3.3 レプリケーション権限チェック	3183
17.4 レプリケーションソリューション	3189
17.4.1 バックアップ用にレプリケーションを使用する	3189
17.4.2 レプリカの予期しない停止の処理	3193
17.4.3 行ベースのレプリケーションの監視	3195
17.4.4 異なるソースおよびレプリカのストレージエンジンでのレプリケーションの使用	3195
17.4.5 スケールアウトのためにレプリケーションを使用する	3196
17.4.6 異なるレプリカへの異なるデータベースのレプリケート	3198
17.4.7 レプリケーションパフォーマンスを改善する	3199
17.4.8 フェイルオーバー中のソースの切替え	3200
17.4.9 非同期接続フェイルオーバーによるソースの切替え	3202
17.4.10 準同期レプリケーション	3204
17.4.11 遅延レプリケーション	3209
17.5 レプリケーションの注釈とヒント	3212
17.5.1 レプリケーションの機能と問題	3212
17.5.2 MySQL バージョン間のレプリケーション互換性	3236
17.5.3 レプリケーションセットアップをアップグレードする	3237
17.5.4 レプリケーションのトラブルシューティング	3238
17.5.5 レプリケーションバグまたは問題を報告する方法	3239

レプリケーションを使用すると、ある MySQL データベースサーバー (ソースと呼ばれる) のデータを、複数の MySQL データベースサーバー (レプリカと呼ばれる) にコピーできます。レプリケーションはデフォルトで非同期です。ソースから更新を受信するためにレプリカを永続的に接続する必要はありません。構成に応じて、すべてのデータベース、選択したデータベース、さらにデータベース内の選択したテーブルを複製できます。

MySQL のレプリケーションの長所は次のとおりです。

- **スケールアウトソリューション** - 複数のレプリカに負荷を分散して、パフォーマンスを向上させます。この環境では、すべての書込みおよび更新がソースサーバーで実行される必要があります。ただし、1 つまたは複数のレプリカで読み取りが行われる場合があります。このモデルでは、(ソースが更新専用であるため) 書込みのパフォーマンスを向上させる一方で、レプリカの数の増加に伴って読取り速度を大幅に向上させることができます。
- **データセキュリティ** - レプリカはレプリケーションプロセスを一時停止できるため、対応するソースデータを破損させることなくレプリカでバックアップサービスを実行できます。
- **アナリティクス** - ライブデータはソースで作成できますが、情報の分析はソースのパフォーマンスに影響を与えることなくレプリカで実行できます。

- 長距離データ分散 - レプリケーションを使用すると、ソースに永続的にアクセスせずに、リモートサイトで使用するデータのローカルコピーを作成できます。

このようなシナリオでレプリケーションを使用する方法については、[セクション17.4「レプリケーションソリューション」](#)を参照してください。

MySQL 8.0 は、様々なレプリケーション方法をサポートしています。従来の方法は、ソースバイナリログからのイベントのレプリケートに基づいており、ソースとレプリカ間で同期するためにログファイルとその中の位置が必要です。グローバルトランザクション識別子 (GTID) に基づく新しい方法はトランザクションであるため、これらのファイル内のログファイルや位置を操作する必要はなく、多くの一般的なレプリケーションタスクが大幅に簡略化されます。GTID を使用したレプリケーションでは、ソースでコミットされたすべてのトランザクションがレプリカにも適用されているが、ソースとレプリカの間の一貫性が保証されます。MySQL での GTID および GTID ベースレプリケーションの詳細は、[セクション17.1.3「グローバルトランザクション識別子を使用したレプリケーション」](#)を参照してください。バイナリログファイルの位置ベースのレプリケーションの使用方法については、[セクション17.1「レプリケーションの構成」](#)を参照してください。

MySQL でのレプリケーションでは、様々なタイプの同期がサポートされます。元のタイプの同期は、一方の非同期レプリケーションであり、一方のサーバーはソースとして機能し、もう一方のサーバーはレプリカとして機能します。これは NDB Cluster の特性である同期レプリケーションとは対照的です ([第23章「MySQL NDB Cluster 8.0」](#)を参照)。MySQL 8.0 では、組込み非同期レプリケーションに加えて、準同期レプリケーションがサポートされています。準同期レプリケーションでは、トランザクションを実行したセッションに戻る前に、ソースで実行されたコミットは、トランザクションのイベントを受信して記録したことを少なくとも 1 つのレプリカが確認するまでブロックされます。[セクション17.4.10「準同期レプリケーション」](#)を参照してください。MySQL 8.0 は、レプリカが少なくとも指定された時間だけソースから意図的に遅れるような遅延レプリケーションもサポートしています。[セクション17.4.11「遅延レプリケーション」](#)を参照してください。同期レプリケーションが必要なシナリオでは、NDB Cluster を使用します ([第23章「MySQL NDB Cluster 8.0」](#)を参照)。

サーバー間のレプリケーションを設定するために使用できるソリューションは多数あり、使用する最適な方法は、使用しているデータの存在とエンジンタイプによって異なります。利用可能なオプションの詳細については、[セクション17.1.2「バイナリログファイルの位置ベースのレプリケーションの設定」](#)を参照してください。

レプリケーション形式の主要なタイプは 2 つあり、1 つは、SQL ステートメント全体を複製する Statement Based Replication (SBR: ステートメントベースレプリケーション)、もう 1 つは変更があった行だけを複製する Row Based Replication (RBR: 行ベースレプリケーション) です。また、3 種類目の混合ベースのレプリケーション (MBR) を使用することもできます。さまざまなレプリケーション形式の詳細については、[セクション17.2.1「レプリケーション形式」](#)を参照してください。

レプリケーションは、いくつかのオプションと変数によって制御されます。詳細は、[セクション17.1.6「レプリケーションおよびバイナリロギングのオプションと変数」](#)を参照してください。[セクション17.3「レプリケーションのセキュリティ」](#)で説明されているように、レプリケーショントポロジに追加のセキュリティ対策を適用できます。

レプリケーションを使用すると、パフォーマンス、様々なデータベースのバックアップのサポート、システム障害を軽減する大規模なソリューションの一部など、様々な問題を解決できます。これらの問題の対処方法については、[セクション17.4「レプリケーションソリューション」](#)を参照してください。

レプリケーション機能の詳細、バージョンの互換性、アップグレード、潜在的な問題とその解決方法など、レプリケーション中の様々なデータ型およびステートメントの処理方法に関するノートおよびヒントは、[セクション17.5「レプリケーションの注釈とヒント」](#)を参照してください。MySQL Replication をはじめて使用する人がよくする質問の回答については、[セクションA.14「MySQL 8.0 FAQ: レプリケーション」](#)を参照してください。

レプリケーションの実装、レプリケーションの動作、バイナリログのプロセスと内容、バックグラウンドスレッド、およびステートメントの記録とレプリケート方法を決定するために使用される規則の詳細は、[セクション17.2「レプリケーションの実装」](#)を参照してください。

17.1 レプリケーションの構成

このセクションでは、MySQL で使用可能な様々なタイプのレプリケーションを構成する方法について説明し、レプリケーション環境に必要な設定および構成 (新しいレプリケーション環境を作成するステップを含む) について説明します。このセクションの主な内容は次のとおりです。

- バイナリログファイルの位置を使用してレプリケーション用に複数のサーバーを設定するためのガイドとして、[セクション17.1.2「バイナリログファイルの位置ベースのレプリケーションの設定」](#)はサーバーの構成を処理し、ソースとレプリカ間でデータをコピーする方法を提供します。

- GTID トランザクションを使用したレプリケーション用に複数のサーバーを設定するためのガイドとして、[セクション17.1.3「グローバルトランザクション識別子を使用したレプリケーション」](#)はサーバーの構成を処理します。
- バイナリログ内のイベントはいくつかの形式で記録されます。これらは、ステートメントベースレプリケーション (SBR) または行ベースレプリケーション (RBR) と呼ばれます。3つ目のタイプ、混合形式レプリケーション (MIXED) は、SBR または RBR レプリケーションを自動的に使用し、必要に応じて SBR と RBR の両方の形式の利点を活用します。さまざまな形式については、[セクション17.2.1「レプリケーション形式」](#)を参照してください。
- レプリケーションに適用するさまざまな構成のオプションと変数に関する詳細は、[セクション17.1.6「レプリケーションおよびバイナリロギングのオプションと変数」](#)を参照してください。
- レプリケーションプロセスが開始されると、管理または監視はほとんど必要ありません。ただし、実行することが望ましい一般的なタスクに関するアドバイスについては、[セクション17.1.7「一般的なレプリケーション管理タスク」](#)を参照してください。

17.1.1 バイナリログファイルの位置ベースのレプリケーション構成の概要

このセクションでは、バイナリログファイルの位置方式に基づく MySQL サーバー間のレプリケーションについて説明します。この方法では、(データベースの変更が行われる) ソースとして動作する MySQL インスタンスが、更新および変更を「events」としてバイナリログに書き込みます。バイナリログ内の情報は、記録されているデータベース変更に応じて異なるロギング形式で格納されます。レプリカは、ソースからバイナリログを読み取り、レプリカローカルデータベース上のバイナリログ内のイベントを実行するように構成されます。

各レプリカは、バイナリログの内容全体のコピーを受け取ります。バイナリログ内のどのステートメントを実行すべきかを決定するのは、レプリカの役割です。特に指定しないかぎり、ソースバイナリログ内のすべてのイベントがレプリカで実行されます。必要に応じて、特定のデータベースまたはテーブルに適用されるイベントのみを処理するようにレプリカを構成できます。

重要

特定のイベントのみを記録するようにソースを構成することはできません。

各レプリカはバイナリログ座標のレコードを保持: ソースから読み取って処理したファイル内のファイル名と位置。つまり、複数のレプリカをソースに接続し、同じバイナリログの異なる部分を実行できます。レプリカはこのプロセスを制御するため、ソース操作に影響を与えることなく、個々のレプリカを接続してサーバーから切断できます。また、各レプリカはバイナリログ内の現在の位置を記録するため、レプリカを切断し、再接続してから処理を再開できます。

ソースと各レプリカは、(`server_id` システム変数を使用して) 一意の ID で構成する必要があります。また、各レプリカは、ソースホスト名、ログファイル名およびそのファイル内の位置に関する情報で構成する必要があります。これらの詳細は、レプリカに対する `CHANGE REPLICATION SOURCE TO` ステートメント (MySQL 8.0.23) または `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 より前) を使用して、MySQL セッション内から制御できます。詳細は、レプリカ接続メタデータリポジトリ内に格納されます ([セクション17.2.4「リレーログおよびレプリケーションメタデータリポジトリ」](#)を参照)。

17.1.2 バイナリログファイルの位置ベースのレプリケーションの設定

このセクションでは、バイナリログファイルの位置ベースのレプリケーションを使用するように MySQL サーバーを設定する方法について説明します。レプリケーションを設定するには様々な方法があり、使用する正確な方法はレプリケーションの設定方法、およびレプリケートするソース上のデータベースにデータがすでにあるかどうかによって異なります。

ヒント

MySQL の複数のインスタンスをデプロイするには、[MySQL Shell](#) で MySQL サーバーインスタンスのグループを簡単に管理できるようにする [InnoDB クラスタ](#) を使用できます。InnoDB クラスタは MySQL Group Replication をプログラム環境でラップするため、MySQL インスタンスのクラスタを簡単にデプロイして高可用性を実現できます。また、InnoDB クラスタは [MySQL Router](#) とシームレスにインタフェースするため、アプリケーションは独自のフェイルオーバープロセスを記述せずにクラスタに接続できます。ただし、高可用性を必要としない同様のユースケースでは、[InnoDB ReplicaSet](#) を使用できます。MySQL Shell のインストール手順は、[here](#) にあります。

すべての設定に共通の汎用タスクがいくつかあります:

- ソースで、バイナリロギングが有効になっていることを確認し、一意のサーバー ID を構成する必要があります。これには、サーバーの再起動が必要となる場合があります。 [セクション17.1.2.1「レプリケーションソース構成の設定」](#)を参照してください。
- ソースに接続する各レプリカで、一意のサーバー ID を構成する必要があります。これには、サーバーの再起動が必要となる場合があります。 [セクション17.1.2.2「レプリカ構成の設定」](#)を参照してください。
- 必要に応じて、レプリケーション用のバイナリログを読み取るときに、ソースとの認証中にレプリカで使用する別のユーザーを作成します。 [セクション17.1.2.3「レプリケーション用ユーザーの作成」](#)を参照してください。
- データスナップショットを作成したり、レプリケーションプロセスを開始したりする前に、ソースで現在の位置をバイナリログに記録するようにしてください。レプリカがイベントの実行を開始するバイナリログ内の場所を認識できるように、レプリカを構成するときにこの情報が必要になります。 [セクション17.1.2.4「レプリケーションソースのバイナリログ座標の取得」](#)を参照してください。
- すでにソースにデータがあり、それを使用してレプリカを同期する場合は、データスナップショットを作成してレプリカにデータをコピーする必要があります。使用しているストレージエンジンは、スナップショットの作成方法に影響します。 [MyISAM](#) を使用している場合は、ソース上のステートメントの処理を停止して読み取りロックを取得し、その現在のバイナリログ座標を取得してそのデータをダンプしてから、ソースがステートメントの実行を続行できるようにする必要があります。ステートメントの実行を停止しないと、データダンプとソースステータス情報が一致なくなり、レプリカ上のデータベースの一貫性または破損が発生します。 [MyISAM](#) ソースのレプリケートの詳細は、 [セクション17.1.2.4「レプリケーションソースのバイナリログ座標の取得」](#)を参照してください。 [InnoDB](#) を使用している場合、読み取りロックは必要なく、データスナップショットの転送に十分な長さのトランザクションで十分です。詳細は、 [セクション15.19「InnoDB と MySQL レプリケーション」](#)を参照してください。
- ホスト名、ログイン資格証明、バイナリログファイルの名前と位置など、ソースに接続するための設定でレプリカを構成します。 [セクション17.1.2.7「レプリカでのソース構成の設定」](#)を参照してください。
- システムに応じて、ソースおよびレプリカにレプリケーション固有のセキュリティ対策を実装します。 [セクション17.3「レプリケーションのセキュリティ」](#)を参照してください。

注記

セットアッププロセスのあるステップでは、[SUPER](#) 権限が必要です。この権限がないと、レプリケーションを有効にできない可能性があります。

基本オプションを構成したあとは、次のシナリオを選択します。

- データを含まないソースおよびレプリカのフレッシュインストールのレプリケーションを設定するには、 [新しいソースおよびレプリカを使用したレプリケーションの設定](#)を参照してください。
- 既存の MySQL サーバーのデータを使用して新しいソースのレプリケーションを設定するには、 [既存のデータによるレプリケーションのセットアップ](#)を参照してください。
- 既存のレプリケーション環境にレプリカを追加するには、 [セクション17.1.2.8「レプリケーション環境へのレプリカの追加」](#)を参照してください。

MySQL レプリケーションサーバーを管理する前に、この章全体を読み、 [セクション13.4.1「ソースサーバーを制御する SQL ステートメント」](#)と [セクション13.4.2「レプリケーションサーバーを制御するための SQL ステートメント」](#)で説明したすべてのステートメントを試みてください。また、 [セクション17.1.6「レプリケーションおよびバイナリロギングのオプションと変数」](#)で説明されたレプリケーションの起動オプションについても習得してください。

17.1.2.1 レプリケーションソース構成の設定

バイナリログファイルの位置ベースのレプリケーションを使用するようにソースを構成するには、バイナリロギングが有効になっていることを確認し、一意のサーバー ID を確立する必要があります。

レプリケーショントポロジ内の各サーバーは、 [server_id](#) システム変数を使用して指定できる一意のサーバー ID で構成する必要があります。このサーバー ID は、レプリケーショントポロジ内の個々のサーバーを識別するために使用され、1 から $(2^{32}-1)$ までの正の整数である必要があります。MySQL 8.0 のデフォルトの [server_id](#) 値は 1 です。次のようなステートメントを発行して、 [server_id](#) 値を動的に変更できます:


```
SET GLOBAL server_id = 2;
```

各サーバー ID がレプリケーショントポロジ内の他のサーバーで使用されている他のすべてのサーバー ID と異なるが、サーバー ID を編成して選択する方法が選択されます。サーバー ID に 0 (以前のリリースではデフォルト) の値が以前に設定されていた場合は、サーバーを再起動して、ソースを新しいゼロ以外のサーバー ID で初期化する必要があります。それ以外の場合は、サーバー ID を変更するときにサーバーを再起動する必要はありません。ただし、サーバー ID を必要とする他の構成を変更する場合は除きます。

バイナリログはソースからそのレプリカに変更をレプリケートするための基礎であるため、ソースではバイナリロギングが必要です。バイナリロギングはデフォルトで有効になっています (`log_bin` システム変数は ON に設定されています)。`--log-bin` オプションは、バイナリログファイルに使用するベース名をサーバーに指示します。ホスト名が変更された場合でも同じバイナリログファイル名を簡単に使用できるように、バイナリログファイルにデフォルト以外のベース名を付けるには、このオプションを指定することをお勧めします ([セクション B.3.7 「MySQL の既知の問題」](#) を参照)。バイナリロギングが以前に `--skip-log-bin` オプションを使用してソースで無効になっていた場合は、このオプションを指定せずにサーバーを再起動して有効にする必要があります。

注記

次のオプションもソースに影響します:

- InnoDB とトランザクションを使用したレプリケーション設定で永続性と一貫性を最大限に高めるには、ソース `my.cnf` ファイルで `innodb_flush_log_at_trx_commit=1` および `sync_binlog=1` を使用する必要があります。
- ソースで `skip_networking` システム変数が有効になっていないことを確認します。ネットワークワーキングが無効になっている場合、レプリカはソースと通信できず、レプリケーションは失敗します。

17.1.2.2 レプリカ構成の設定

各レプリカには、`server_id` システム変数で指定された一意のサーバー ID が必要です。複数のレプリカを設定する場合、各レプリカの `server_id` 値は、ソースと他のレプリカの値とは異なる一意である必要があります。レプリカサーバー ID がまだ設定されていない場合、または現在の値がソースまたは別のレプリカに選択した値と競合する場合は、それを変更する必要があります。

デフォルトの `server_id` 値は 1 です。次のようなステートメントを発行して、`server_id` 値を動的に変更できます:

```
SET GLOBAL server_id = 21;
```

サーバー ID の値が 0 の場合、レプリカはソースに接続できません。そのサーバー ID 値 (以前のリリースではデフォルト) が以前に設定されていた場合は、サーバーを再起動して、新しいゼロ以外のサーバー ID でレプリカを初期化する必要があります。それ以外の場合は、サーバー ID を変更するときにサーバーを再起動する必要はありません。ただし、サーバー ID を必要とする他の構成を変更する場合は除きます。たとえば、バイナリロギングがサーバーで無効になっていて、それをレプリカに対して有効にする場合、これを有効にするにはサーバーの再起動が必要です。

レプリカサーバーを停止する場合は、構成ファイルの `[mysqld]` セクションを編集して、一意のサーバー ID を指定できます。例:

```
[mysqld]
server-id=21
```

バイナリロギングは、すべてのサーバーでデフォルトで有効になっています。レプリケーションを実行するために、レプリカでバイナリロギングを有効にする必要はありません。ただし、レプリカのバイナリロギングとは、レプリカバイナリログをデータバックアップおよびクラッシュ回復に使用できることを意味します。バイナリロギングが有効になっているレプリカは、より複雑なレプリケーショントポロジの一部としても使用できます。たとえば、次の連鎖配置を使用してレプリケーションサーバーを設定できます:

```
A -> B -> C
```

ここで、A はレプリカ B のソースとして機能し、B はレプリカ C のソースとして機能します。これが機能するには、B がソースとレプリカの両方である必要があります。A から受信した更新を C に渡すには、B がバイナリログに記録する必要があります。バイナリロギングに加えて、このレプリケーショントポロジでは `log_slave_updates` システム変数を有効にする必要があります。レプリカ更新が有効になっている場合、レプリカはソースから受信し、レプ

リカ SQL スレッドによって実行された更新をレプリカ独自のバイナリログに書き込みます。 `log_slave_updates` システム変数はデフォルトで有効になっています。

レプリカでバイナリロギングまたはレプリカ更新ロギングを無効にする必要がある場合は、レプリカの `--skip-log-bin` および `--log-slave-updates=OFF` オプションを指定することでこれを行うことができます。レプリカでこれらの機能を再度有効にする場合は、関連するオプションを削除してサーバーを再起動します。

17.1.2.3 レプリケーション用ユーザーの作成

各レプリカは MySQL のユーザー名とパスワードを使用してソースに接続するため、レプリカが接続に使用できるユーザーアカウントがソースに存在する必要があります。ユーザー名は、レプリカの設定時に、`CHANGE REPLICATION SOURCE TO` ステートメント (MySQL 8.0.23 から) または `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 から) の `SOURCE_USER` | `MASTER_USER` オプションによって指定されます。この操作には、`REPLICATION SLAVE` 権限が付与されているすべてのアカウントを使用できます。レプリカごとに異なるアカウントを作成することも、レプリカごとに同じアカウントを使用してソースに接続することもできます。

レプリケーション専用のアカウントを作成する必要はありませんが、レプリケーションユーザー名とパスワードはレプリカ接続メタデータリポジトリ `mysql.slave_master_info` にプレーンテキストで格納されることに注意してください (セクション 17.2.4.2 「レプリケーションメタデータリポジトリ」を参照)。このため、ほかのアカウントのセキュリティを損なう可能性を最小限に抑えるため、レプリケーションプロセスにのみ権限を持つ別のアカウントを作成することをお勧めします。

新しいアカウントを作成するには、`CREATE USER` を使用します。レプリケーションに必要な権限をこのアカウントに付与するには、`GRANT` ステートメントを使用します。レプリケーションの目的にだけアカウントを作成する場合、そのアカウントには `REPLICATION SLAVE` 権限だけが必要です。たとえば、`example.com` ドメイン内の任意のホストからレプリケーション用に接続できる新しいユーザー `repl` を設定するには、ソースで次のステートメントを発行します:

```
mysql> CREATE USER 'repl'@'%example.com' IDENTIFIED BY 'password';
mysql> GRANT REPLICATION SLAVE ON *.* TO 'repl'@'%example.com';
```

ユーザーアカウントを操作するためのステートメントの詳細については、セクション 13.7.1 「アカウント管理ステートメント」を参照してください。

重要

`caching_sha2_password` プラグインで認証するユーザーアカウントを使用してソースに接続するには、セクション 17.3.1 「暗号化接続を使用するためのレプリケーションの設定」の説明に従ってセキュアな接続を設定するか、RSA キーペアを使用したパスワード交換をサポートするように暗号化されていない接続を有効にする必要があります。`caching_sha2_password` 認証プラグインは、MySQL 8.0 から作成された新規ユーザーのデフォルトです (詳細は、セクション 6.4.1.2 「SHA-2 プラガブル認証のキャッシュ」を参照)。(MASTER_USER オプションで指定された)レプリケーション用に作成または使用するユーザーアカウントがこの認証プラグインを使用し、セキュアな接続を使用していない場合は、接続を成功させるために RSA 鍵ペアベースのパスワード交換を有効にする必要があります。

17.1.2.4 レプリケーションソースのバイナリログ座標の取得

レプリケーションプロセスを正しい時点で開始するようにレプリカを構成するには、そのバイナリログ内のソースの現在の座標を書き留める必要があります。

警告

このプロシージャでは、InnoDB テーブルの `COMMIT` 操作をブロックする `FLUSH TABLES WITH READ LOCK` を使用します。

ソースを停止してデータスナップショットを作成する場合は、オプションでこの手順をスキップし、かわりにバイナリログインデックスファイルのコピーをデータスナップショットとともに格納できます。その場合、ソースは再起動時に新しいバイナリログファイルを作成します。したがって、レプリカがレプリケーションプロセスを開始すべきソースのバイナリログ座標は、その新しいファイルの開始点であり、コピーされたバイナリログインデックスファイルにリストされているファイルの次のソースのバイナリログファイルとなります。

ソースバイナリログ座標を取得するには、次のステップに従います:

1. コマンドラインクライアントを使用してソースに接続してセッションを開始し、[FLUSH TABLES WITH READ LOCK](#) ステートメントを実行してすべてのテーブルおよびブロック書き込みステートメントをフラッシュします:

```
mysql> FLUSH TABLES WITH READ LOCK;
```

警告

読み取りロックを有効のままにするため、[FLUSH TABLES](#) ステートメントを発行したクライアントを実行中のままにしてください。クライアントを終了すると、ロックは解除されます。

2. ソース上の別のセッションで、[SHOW MASTER STATUS](#) ステートメントを使用して現在のバイナリログファイルの名前と位置を確認します:

```
mysql > SHOW MASTER STATUS;
+-----+-----+-----+-----+
| File          | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----+-----+-----+-----+
| mysql-bin.000003 | 73      | test         | manual,mysql      |
+-----+-----+-----+-----+
```

File カラムにはログファイルの名前が表示され、**Position** カラムにはファイル内の位置が表示されます。この例では、バイナリログファイルは `mysql-bin.000003` で、位置は 73 です。これらの値を記録します。これらは、後でレプリカを設定するときが必要になります。これらは、レプリカがソースからの新しい更新の処理を開始するレプリケーション座標を表します。

ソースがそれまでバイナリロギングを無効にして実行されていた場合、[SHOW MASTER STATUS](#) または [mysqldump --master-data](#) によって表示されるログファイルの名前と位置の値は空です。その場合、あとでソースバイナリログファイルと位置を指定するときを使用する必要がある値は、空の文字列 ("") と 4 です。

これで、レプリカがレプリケーションを開始するための正しい場所でソースバイナリログからの読み取りを開始できるようにするために必要な情報が得られました。

次のステップは、ソースに既存のデータがあるかどうかによって異なります。次のいずれかのオプションを選択します:

- レプリケーションを開始する前にレプリカと同期する必要がある既存のデータがある場合は、ロックが維持されるようにクライアントを実行したままにします。これにより、レプリカにコピーされたデータがソースと同期されるように、それ以上の変更が行われなくなります。[セクション17.1.2.5「データスナップショットの方法の選択」](#)に進みます。
- 新しいソースとレプリカの組合せを設定する場合は、最初のセッションを終了して読み取りロックを解放できます。手順については、[新しいソースおよびレプリカを使用したレプリケーションの設定](#) を参照してください。

17.1.2.5 データスナップショットの方法の選択

ソースデータベースに既存のデータが含まれている場合、このデータを各レプリカにコピーする必要があります。ソースデータベースからデータをダンプするには、様々な方法があります。次の各セクションでは、使用可能なオプションについて説明します。

データベースをダンプする適切な方法を選択するには、次のいずれかのオプションを選択します:

- [mysqldump](#) ツールを使用して、レプリケートするすべてのデータベースのダンプを作成します。これは、特に [InnoDB](#) を使用する場合に推奨される方法です。
- データベースがバイナリポータブルファイルに格納されている場合は、RAW データファイルをレプリカにコピーできます。これは、[INSERT](#) ステートメントのリプレイ時にインデックスの更新のオーバーヘッドがスキップされるため、[mysqldump](#) を使用して各レプリカにファイルをインポートするよりも効率的です。[InnoDB](#) などのストレージエンジンでは、これはお勧めしません。
- MySQL Server クローンプラグインを使用して、既存のレプリカからクローンにすべてのデータを転送します。この方法の使用手順は、[セクション5.6.7.6「レプリケーション用のクローニング」](#) を参照してください。

ヒント

MySQL の複数のインスタンスをデプロイするには、[MySQL Shell](#) で MySQL サーバーインスタンスのグループを簡単に管理できるようにする [InnoDB クラスタ](#) を使用できます。InnoDB クラスタは MySQL Group Replication をプログラム環境でラップするため、MySQL インスタンスのクラスタを簡単にデプロイして高可用性を実現できます。また、InnoDB クラスタは [MySQL Router](#) とシームレスにインターフェースするため、アプリケーションは独自のフェイルオーバープロセスを記述せずにクラスタに接続できます。ただし、高可用性を必要としない同様のユースケースでは、[InnoDB ReplicaSet](#) を使用できます。MySQL Shell のインストール手順は、[here](#) にあります。

mysqldump を使用したデータスナップショットの作成

既存のソースデータベースにデータの snapshots を作成するには、[mysqldump](#) ツールを使用します。データダンプが完了したら、レプリケーションプロセスを開始する前に、このデータをレプリカにインポートします。

次の例では、すべてのデータベースを `dbdump.db` という名前のファイルにダンプし、レプリケーションプロセスを開始するためにレプリカに必要な [CHANGE REPLICATION SOURCE TO](#)/[CHANGE MASTER TO](#) ステートメントを自動的に追加する `--master-data` オプションを含めます:

```
shell> mysqldump --all-databases --master-data > dbdump.db
```

注記

`--master-data` を使用しない場合は、別のセッションですべてのテーブルを手動でロックする必要があります。 [セクション 17.1.2.4 「レプリケーションソースのバイナリログ座標の取得」](#) を参照してください。

[mysqldump](#) ツールを使用して、特定のデータベースをダンプから除外できます。ダンプに含めるデータベースを選択する場合は、`--all-databases` を使用しないでください。次のいずれかのオプションを選択します:

- `--ignore-table` オプションを使用して、データベース内のすべてのテーブルを除外します。
- `--databases` オプションを使用してダンプするデータベースのみに名前を付けます。

注記

デフォルトでは、GTID がソース (`gtid_mode=ON`) で使用されている場合、[mysqldump](#) はソース上の `gtid_executed` セットの GTID をダンプ出力に含めて、レプリカ上の `gtid_purged` セットに追加します。特定のデータベースまたはテーブルのみをダンプする場合、[mysqldump](#) に含まれる値には、データベースの抑制された部分を変更したトランザクションや、部分ダンプに含まれていないサーバー上のその他のデータベースであっても、ソース上の `gtid_executed` セット内のすべてのトランザクションの GTID が含まれることに注意してください。 `mysqldump --set-gtid-purged` オプションの説明を確認して、使用している MySQL Server バージョンのデフォルト動作の結果、およびこの結果が状況に適していない場合の動作の変更方法を確認します。

詳細は、[セクション 4.5.4 「mysqldump — データベースバックアッププログラム」](#) を参照してください。

データをインポートするには、ダンプファイルをレプリカにコピーするか、レプリカにリモート接続するときソースからファイルにアクセスします。

ローデータファイルを使用したデータスナップショットの作成

このセクションでは、データベースを構成する RAW ファイルを使用してデータスナップショットを作成する方法について説明します。複雑なキャッシュまたはロギングアルゴリズムを持つストレージエンジンを使用するテーブルでこの方法を使用するには、完全な「ポイントインタイム」スナップショットを生成するための追加のステップが必要です: 最初のコピーコマンドでは、グローバル読み取りロックを取得した場合でも、キャッシュ情報およびロギング更新を除外できます。ストレージエンジンがこれにどのように反応するかは、そのクラッシュリカバリ能力によります。

InnoDB テーブルを使用する場合、MySQL Enterprise Backup コンポーネントから `mysqlbackup` コマンドを使用して、一貫性のあるスナップショットを作成できます。このコマンドは、レプリカで使用されるスナップショットに対

応するログ名とオフセットを記録します。MySQL Enterprise Backup は MySQL Enterprise サブスクリプションの一部として同梱される製品です。詳細は、[セクション30.2「MySQL Enterprise Backup の概要」](#)を参照してください。

この方法は、ソースとレプリカの値が `ft_stopword_file`、`ft_min_word_len` または `ft_max_word_len` で異なり、全文インデックスを持つテーブルをコピーしている場合にも確実に機能しません。

前述の例外がデータベースに適用されない場合は、[cold backup](#) 手法を使用して、[InnoDB](#) テーブルの信頼性のあるバイナリスナップショットを取得: MySQL Server の [slow shutdown](#) を実行してから、データファイルを手動でコピーします。

MySQL データファイルが単一のファイルシステムに存在する場合に [MyISAM](#) テーブルの RAW データスナップショットを作成するには、`cp` や `copy` などのアーカイブツール、`scp` や `rsync` などのリモートコピーツール、または `zip` または `tar`、または `dump` のようなファイルシステムスナップショットツールを使用できます。特定のデータベースだけを複製する場合、それらのテーブルに関するファイルだけをコピーします。[InnoDB](#) の場合、`innodb_file_per_table` オプションを有効にしないかぎり、すべてのデータベースのすべてのテーブルが `system tablespace` ファイルに格納されます。

次のファイルはレプリケーションには必要ありません:

- `mysql` データベースに関連するファイル。
- レプリカ接続メタデータリポジトリファイル `master.info` が使用されている場合、このファイルの使用は非推奨になりました ([セクション17.2.4「リレーログおよびレプリケーションメタデータリポジトリ」](#)を参照)。
- これを使用してレプリカのソースバイナリログ座標を検索する場合は、バイナリログインデックスファイルを除き、ソースログファイル。
- リレーログファイル。

[InnoDB](#) テーブルを使用しているかどうかに応じて、次のいずれかを選択します:

[InnoDB](#) テーブルを使用していて、RAW データスナップショットと最も一貫性のある結果を得るには、プロセス中に次のようにソースサーバーを停止します:

1. 読み取りロックを取得し、ソースステータスを取得します。 [セクション17.1.2.4「レプリケーションソースのバイナリログ座標の取得」](#)を参照してください。
2. 別のセッションで、ソースサーバーを停止します:

```
shell> mysqladmin shutdown
```

3. MySQL データファイルのコピーを作成します。次の例では、これを行うための一般的な方法を示します。この中の1つだけを選択する必要があります。

```
shell> tar cf /tmp/db.tar ./data
shell> zip -r /tmp/db.zip ./data
shell> rsync --recursive ./data /tmp/dbdata
```

4. ソースサーバーを再起動します。

[InnoDB](#) テーブルを使用していない場合は、次の手順で説明するように、サーバーをシャットダウンせずにソースからシステムのスナップショットを取得できます:

1. 読み取りロックを取得し、ソースステータスを取得します。 [セクション17.1.2.4「レプリケーションソースのバイナリログ座標の取得」](#)を参照してください。
2. MySQL データファイルのコピーを作成します。次の例では、これを行うための一般的な方法を示します。この中の1つだけを選択する必要があります。

```
shell> tar cf /tmp/db.tar ./data
shell> zip -r /tmp/db.zip ./data
shell> rsync --recursive ./data /tmp/dbdata
```

3. 読み取りロックを獲得したクライアントでは、ロックを解除します。

```
mysql> UNLOCK TABLES;
```


データベースのアーカイブまたはコピーを作成したら、レプリケーションプロセスを開始する前に各レプリカにファイルをコピーします。

17.1.2.6 レプリカの設定

次の各セクションでは、レプリカの設定方法について説明します。続行する前に、次のことを確認してください：

- 必要な構成プロパティを使用してソースを構成しました。 [セクション17.1.2.1「レプリケーションソース構成の設定」](#)を参照してください。
- データスナップショットのシャットダウン中に作成されたソースステータス情報またはソースバイナリログインデックスファイルのコピーを取得しました。 [セクション17.1.2.4「レプリケーションソースのバイナリログ座標の取得」](#)を参照してください。
- ソースで、読み取りロックを解放します：

```
mysql> UNLOCK TABLES;
```

- レプリカで、MySQL 構成を編集しました。 [セクション17.1.2.2「レプリカ構成の設定」](#)を参照してください。

次のステップは、レプリカにインポートする既存のデータがあるかどうかによって異なります。詳しくは [セクション17.1.2.5「データスナップショットの方法の選択」](#) をご覧ください。次のいずれかを選択します：

- インポートするデータベースのスナップショットがない場合は、 [新しいソースおよびレプリカを使用したレプリケーションの設定](#) を参照してください。
- インポートするデータベースのスナップショットがある場合は、 [既存のデータによるレプリケーションのセットアップ](#) を参照してください。

新しいソースおよびレプリカを使用したレプリケーションの設定

インポートする以前のデータベースのスナップショットがない場合は、新しいソースからレプリケーションを開始するようにレプリカを構成します。

ソースと新しいレプリカ間のレプリケーションを設定するには：

1. レプリカを起動します。
2. レプリカに対して `CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO` ステートメントを実行し、ソース構成を設定します。 [セクション17.1.2.7「レプリカでのソース構成の設定」](#)を参照してください。

各レプリカで次のレプリカ設定ステップを実行します。

この方法は、新しいサーバーを設定しているが、レプリケーション構成にロードする別のサーバーのデータベースの既存のダンプがある場合にも使用できます。データを新しいソースにロードすると、データはレプリカに自動的にレプリケートされます。

別の既存のデータベースサーバーのデータを使用して新しいレプリケーション環境を設定して新しいソースを作成する場合は、そのサーバーから生成されたダンプファイルを新しいソースで実行します。データベース更新はレプリカに自動的に伝播されます：

```
shell> mysql -h source < fulldb.dump
```

既存のデータによるレプリケーションのセットアップ

既存のデータを使用してレプリケーションを設定する場合は、レプリケーションを開始する前に、スナップショットをソースからレプリカに転送します。レプリカにデータをインポートするプロセスは、ソースでのデータのスナップショットの作成方法によって異なります。

ヒント

MySQL の複数のインスタンスをデプロイするには、[MySQL Shell](#) で MySQL サーバーインスタンスのグループを簡単に管理できるようにする [InnoDB クラスタ](#) を使用でき

ます。InnoDB クラスタは MySQL Group Replication をプログラム環境でラップするため、MySQL インスタンスのクラスタを簡単にデプロイして高可用性を実現できます。また、InnoDB クラスタは [MySQL Router](#) とシームレスにインタフェースするため、アプリケーションは独自のフェイルオーバープロセスを記述せずにクラスタに接続できます。ただし、高可用性を必要としない同様のユースケースでは、[InnoDB ReplicaSet](#) を使用できます。MySQL Shell のインストール手順は、[here](#) にあります。

注記

新しいレプリカを作成するためにコピーするレプリケーションソースサーバーまたは既存のレプリカにスケジュールされたイベントがある場合は、それらが新しいレプリカで無効になっていることを確認してから開始してください。ソースですでに実行されている新しいレプリカでイベントが実行されると、複製された操作によってエラーが発生します。イベントスケジューラは、`event_scheduler` システム変数によって制御されます。このシステム変数のデフォルトは MySQL 8.0 の `ON` であるため、元のサーバーでアクティブなイベントは、新しいレプリカの起動時にデフォルトで実行されます。新しいレプリカでのすべてのイベントの実行を停止するには、新しいレプリカで `event_scheduler` システム変数を `OFF` または `DISABLED` に設定します。または、`ALTER EVENT` ステートメントを使用して個々のイベントを `DISABLE` または `DISABLE ON SLAVE` に設定し、新しいレプリカで実行されないようにすることもできます。`SHOW` ステートメントまたは情報スキーマ `EVENTS` テーブルを使用して、サーバー上のイベントをリストできます。詳細は、[セクション 17.5.1.16 「呼び出される機能のレプリケーション」](#) を参照してください。

既存のレプリカからクローンにすべてのデータを転送します。この方法を使用する手順については、次のいずれかの手順を選択してください。

1. MySQL Server クローンプラグインを使用して既存のレプリカからクローンを作成した場合 ([セクション 5.6.7.6 「レプリケーション用のクローニング」](#) を参照)、データはすでに転送されています。それ以外の場合は、次のいずれかの方法を使用してレプリカにデータをインポートします。
 - a. `mysqldump` を使用した場合は、レプリケーションが開始されないように、`--skip-slave-start` オプションを使用してレプリカを起動します。次に、ダンプファイルをインポートします:

```
shell> mysql < fulldb.dump
```
 - b. RAW データファイルを使用してスナップショットを作成した場合は、データファイルをレプリカデータディレクトリに抽出します。例:

```
shell> tar xvf dbdump.tar
```

レプリカサーバーがファイルにアクセスして変更できるように、ファイルに対する権限と所有権の設定が必要になる場合があります。次に、レプリケーションが開始されないように、`--skip-slave-start` オプションを使用してレプリカを起動します。
2. ソースのレプリケーション座標を使用してレプリカを構成します。これにより、レプリケーションを開始する必要があるバイナリログファイルおよびファイル内の位置がレプリカに通知されます。また、ソースのログイン資格証明とホスト名を使用してレプリカを構成します。必要な `CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO` ステートメントの詳細は、[セクション 17.1.2.7 「レプリカでのソース構成の設定」](#) を参照してください。
3. `START REPLICATION | SLAVE` ステートメントを発行してレプリケーションスレッドを起動します。

この手順を実行すると、レプリカはソースに接続し、スナップショットの取得後にソースで発生した更新をレプリケートします。なんらかの理由でレプリケートできない場合、エラーメッセージがレプリカエラーログに発行されません。

レプリカは、接続メタデータリポジトリおよび適用者メタデータリポジトリに記録された情報を使用して、処理されたソースバイナリログの量を追跡します。MySQL 8.0 からは、デフォルトで、これらのリポジトリは `mysql` データベース内の `slave_master_info` および `slave_relay_log_info` という名前のテーブルです。実行内容を正確に把握し、その影響を完全に理解していないかぎり、これらのテーブルを削除または編集しないでください。その場合でも、レプリケーションパラメータを変更するには、`CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO` ステートメントを使用することをお勧めします。レプリカは、ステートメントで指定された値を使用して、レプリケーション

メタデータリポジトリを自動的に更新します。詳しくは[セクション17.2.4「リレーログおよびレプリケーションメタデータリポジトリ」](#),をご覧ください。

注記

レプリカ接続メタデータリポジトリの内容は、コマンドラインまたは `my.cnf` で指定されたサーバーオプションの一部をオーバーライドします。詳細については、[セクション17.1.6「レプリケーションおよびバイナリログのオプションと変数」](#)を参照してください。

ソースの単一のスナップショットでは、複数のレプリカで十分です。追加のレプリカを設定するには、同じソーススナップショットを使用して、前述の手順のレプリカ部分に従います。

17.1.2.7 レプリカでのソース構成の設定

レプリケーションのソースと通信するようにレプリカを設定するには、必要な接続情報を使用してレプリカを構成します。これを行うには、レプリカで、[CHANGE REPLICATION SOURCE TO](#) ステートメント (MySQL 8.0.23 の場合) または [CHANGE MASTER TO](#) ステートメント (MySQL 8.0.23 の場合) を実行し、オプション値をシステムに関連する実際の値に置き換えます:

```
mysql> CHANGE MASTER TO
-> MASTER_HOST='source_host_name',
-> MASTER_USER='replication_user_name',
-> MASTER_PASSWORD='replication_password',
-> MASTER_LOG_FILE='recorded_log_file_name',
-> MASTER_LOG_POS=recorded_log_position;
```

Or from MySQL 8.0.23:

```
mysql> CHANGE REPLICATION SOURCE TO
-> SOURCE_HOST='source_host_name',
-> SOURCE_USER='replication_user_name',
-> SOURCE_PASSWORD='replication_password',
-> SOURCE_LOG_FILE='recorded_log_file_name',
-> SOURCE_LOG_POS=recorded_log_position;
```

注記

レプリケーションでは、Unix ソケットファイルを使用できません。TCP/IP を使用してソース MySQL サーバーに接続できる必要があります。

[CHANGE REPLICATION SOURCE TO](#) | [CHANGE MASTER TO](#) ステートメントには、その他のオプションもあります。たとえば、SSL を使用してセキュアなレプリケーションをセットアップできます。オプションの完全なリスト、および文字列値オプションに許可された最大長に関する情報については、[セクション13.4.2.1「CHANGE MASTER TO ステートメント」](#)を参照してください。

重要

[セクション17.1.2.3「レプリケーション用ユーザーの作成」](#)で説明したように、セキュアな接続を使用せず、`SOURCE_USER` | `MASTER_USER` オプションで指定されたユーザーアカウントが `caching_sha2_password` プラグイン (MySQL 8.0 からのデフォルト) を使用して認証を行う場合は、[CHANGE REPLICATION SOURCE TO](#) | [CHANGE MASTER TO](#) ステートメントの `SOURCE_PUBLIC_KEY_PATH` | `MASTER_PUBLIC_KEY_PATH` または `GET_SOURCE_PUBLIC_KEY` | `GET_MASTER_PUBLIC_KEY` オプションを指定して RSA キーペアベースパスワード交換を使用可能にします。

17.1.2.8 レプリケーション環境へのレプリカの追加

ソースサーバーを停止せずに、既存のレプリケーション構成に別のレプリカを追加できます。これを行うには、既存のレプリカのデータディレクトリをコピーし、新しいレプリカに別のサーバー ID (ユーザー指定) およびサーバー UUID (起動時に生成) を指定して、新しいレプリカを設定します。

注記

新しいレプリカを作成するためにコピーするレプリケーションソースサーバーまたは既存のレプリカにスケジュールされたイベントがある場合は、それらが新しいレプリカで無効に

なっていることを確認してから開始してください。ソースですでに実行されている新しいレプリカでイベントが実行されると、複製された操作によってエラーが発生します。イベントスケジューラは、`event_scheduler` システム変数によって制御されます。このシステム変数のデフォルトは MySQL 8.0 の `ON` であるため、元のサーバーでアクティブなイベントは、新しいレプリカの起動時にデフォルトで実行されます。新しいレプリカでのすべてのイベントの実行を停止するには、新しいレプリカで `event_scheduler` システム変数を `OFF` または `DISABLED` に設定します。または、`ALTER EVENT` ステートメントを使用して個々のイベントを `DISABLE` または `DISABLE ON SLAVE` に設定し、新しいレプリカで実行されないようにすることもできます。`SHOW` ステートメントまたは情報スキーマ `EVENTS` テーブルを使用して、サーバー上のイベントをリストできます。詳細は、[セクション17.5.1.16「呼び出される機能のレプリケーション」](#)を参照してください。

この方法で新しいレプリカを作成するかわりに、MySQL Server クローンプラグインを使用して、既存のレプリカからクローンにすべてのデータおよびレプリケーション設定を転送できます。この方法の使用手順は、[セクション5.6.7.6「レプリケーション用のクローニング」](#)を参照してください。

クローニングせずに既存のレプリカを複製するには、次のステップに従います:

1. 既存のレプリカを停止し、レプリカステータス情報 (特にソースバイナリログファイルとリレーログファイルの位置) を記録します。レプリカステータスは、パフォーマンススキーマレプリケーションテーブル ([セクション27.12.11「パフォーマンススキーマレプリケーションテーブル」](#)を参照) で表示するか、次のように `SHOW REPLICAS | SLAVE STATUS` を発行して表示できます:

```
mysql> STOP SLAVE;
mysql> SHOW SLAVE STATUS\G
Or from MySQL 8.0.22:
mysql> STOP REPLICAS;
mysql> SHOW REPLICAS STATUS\G
```

2. 既存のレプリカを停止します:

```
shell> mysqladmin shutdown
```

3. ログファイルやリレーログファイルなど、既存のレプリカから新しいレプリカにデータディレクトリをコピーします。これを行うには、`tar` または `WinZip` を使用してアーカイブを作成するか、`cp`、`rsync` などのツールを使用して直接コピーを実行します。

重要

- コピーする前に、既存のレプリカに関連するすべてのファイルが実際にデータディレクトリに格納されていることを確認します。たとえば、`InnoDB` のシステムテーブルスペース、`undo` テーブルスペースおよび `redo` ログを別の場所に格納できます。`InnoDB` テーブルスペースファイルおよび `file-per-table` テーブルスペースが他のディレクトリに作成されている可能性があります。レプリカのバイナリログおよびリレーログは、データディレクトリ外の独自のディレクトリに存在する場合があります。既存のレプリカに設定されているシステム変数を確認し、指定されている代替パスを探します。見つかった場合は、これらのディレクトリもコピーします。
- コピー中に、レプリケーションメタデータリポジトリにファイルが使用されている場合 ([セクション17.2.4「リレーログおよびレプリケーションメタデータリポジトリ」](#)を参照)、これらのファイルも既存のレプリカから新しいレプリカにコピーしてください。リポジトリにテーブルが使用されている場合 (MySQL 8.0 のデフォルト)、テーブルはデータディレクトリにあります。
- コピー後、新しいレプリカのデータディレクトリのコピーから `auto.cnf` ファイルを削除して、生成された別のサーバー UUID で新しいレプリカが開始されるようにします。サーバー UUID は一意である必要があります。

新しいレプリカの追加時に発生する一般的な問題は、新しいレプリカが次のような一連の警告およびエラーメッセージで失敗することです:

```
071118 16:44:10 [Warning] Neither --relay-log nor --relay-log-index were used; so
replication may break when this MySQL server acts as a replica and has his hostname
changed!! Please use '--relay-log=new_replica_hostname-relay-bin' to avoid this problem.
```

```
071118 16:44:10 [ERROR] Failed to open the relay log './old_replica_hostname-relay-bin.003525'  
(relay_log_pos 22940879)  
071118 16:44:10 [ERROR] Could not find target log during relay log initialization  
071118 16:44:10 [ERROR] Failed to initialize the master info structure
```

リレーログファイルにはファイル名の一部としてホスト名が含まれているため、この状況は `relay_log` システム変数が指定されていない場合に発生することがあります。これは、`relay_log_index` システム変数が使用されない場合のリレーログインデックスファイルにも当てはまります。これらの変数の詳細は、[セクション17.1.6「レプリケーションおよびバイナリロギングのオプションと変数」](#)を参照してください。

この問題を回避するには、既存のレプリカで使用された新しいレプリカの `relay_log` に同じ値を使用します。このオプションが既存のレプリカで明示的に設定されていない場合は、`existing_replica_hostname-relay-bin` を使用します。これが不可能な場合は、既存のレプリカリレーログインデックスファイルを新しいレプリカにコピーし、既存のレプリカで使用されたものと一致するように新しいレプリカの `relay_log_index` システム変数を設定します。このオプションが既存のレプリカで明示的に設定されていない場合は、`existing_replica_hostname-relay-bin.index` を使用します。または、このセクションの残りのステップに従って新しいレプリカを起動しようとし、前述のようなエラーが発生した場合は、次のステップを実行します：

- a. まだ実行していない場合は、新しいレプリカで `STOP REPLICHA | SLAVE` を発行します。

既存のレプリカをすでに再起動している場合は、既存のレプリカでも `STOP REPLICHA | SLAVE` を発行します。
 - b. 既存のレプリカリレーログインデックスファイルの内容を新しいレプリカリレーログインデックスファイルにコピーし、ファイル内にすでに存在する内容を上書きしてください。
 - c. このセクションの残りの手順に進みます。
4. コピーが完了したら、既存のレプリカを再起動します。
 5. 新しいレプリカで構成を編集し、新しいレプリカに、ソースまたは既存のレプリカで使用されていない一意のサーバー ID (`server_id` システム変数を使用) を指定します。
 6. レプリケーションがまだ開始されないように、`--skip-slave-start` オプションを指定して新しいレプリカサーバーを起動します。パフォーマンススキーマレプリケーションテーブルを使用するか、`SHOW REPLICHA | SLAVE STATUS` を発行して、既存のレプリカと比較して新しいレプリカの設定が正しいことを確認します。また、サーバー ID とサーバー UUID を表示し、これらが正しいことと、新しいレプリカに対して一意であることを確認します。
 7. `START REPLICHA | SLAVE` ステートメントを発行してレプリカスレッドを起動します。これで、新しいレプリカは接続メタデータリポジトリの情報を使用してレプリケーションプロセスを開始します。

17.1.3 グローバルトランザクション識別子を使用したレプリケーション

このセクションでは、グローバルトランザクション識別子 (GTID) を使用したトランザクションベースのレプリケーションについて説明します。GTID を使用している場合、各トランザクションは元のサーバーでコミットされ、レプリカによって適用されるため、識別および追跡できます。つまり、GTID を使用して新しいレプリカの起動時または新しいソースへのフェイルオーバー時に、それらのファイル内のログファイルまたは位置を参照する場合は必要ありません。これにより、これらのタスクが大幅に簡略化されます。GTID ベースのレプリケーションは完全にトランザクションベースであるため、ソースとレプリカに一貫性があるかどうかを簡単に判断できます。ソースでコミットされたすべてのトランザクションもレプリカでコミットされているかぎり、両者の間の一貫性が保証されます。ステートメントベースまたは行ベースレプリケーションを GTID に基づいて使用できます ([セクション17.2.1「レプリケーション形式」](#)を参照してください)。ただし、最善の結果を得るには、行ベース形式を使用することをお勧めします。

GTID は常にソースとレプリカの間で保持されます。つまり、バイナリログを調べることによって、レプリカに適用されているトランザクションのソースをいつでも判別できます。また、ある GTID のトランザクションがあるサーバーでコミットされると、同じ GTID のそれ以降のトランザクションはそのサーバーで無視されます。したがって、ソースでコミットされたトランザクションはレプリカに複数回適用でき、一貫性の保証に役立ちます。

このセクションでは、次のトピックについて説明します。

- GTID の定義方法と作成方法、および GTID が MySQL サーバーでどのように表されるか ([セクション 17.1.3.1「GTID 形式および格納」](#)を参照)。

- GTID のライフサイクル ([セクション17.1.3.2「GTID ライフサイクル」](#) を参照)。
- GTID を使用するレプリカとソースを同期するための自動配置機能 ([セクション17.1.3.3「GTID 自動配置」](#) を参照)。
- GTID ベースレプリケーションをセットアップおよび起動するための一般的な手順 ([セクション17.1.3.4「GTID を使用したレプリケーションのセットアップ」](#) を参照してください)。
- GTID を使用するとき新しいレプリケーションサーバーをプロビジョニングするために推奨される方法 ([セクション17.1.3.5「フェイルオーバーおよびスケールアウトでの GTID の使用」](#) を参照してください)。
- GTID ベースレプリケーションを使用するとき留意すべき制約と制限 ([セクション17.1.3.7「GTID ベースレプリケーションの制約」](#) を参照してください)。
- GTID の操作に使用できるストアドファンクション ([セクション17.1.3.8「GTID を操作するストアドファンクションの例」](#) を参照)。

GTID ベースレプリケーションに関する MySQL Server オプションおよび変数については、[セクション17.1.6.5「グローバルトランザクション ID システム変数」](#) を参照してください。GTID と一緒に使用するために MySQL 8.0 がサポートする SQL 関数については、[セクション12.19「グローバルトランザクション識別子 \(GTID\) で使用される機能」](#) も参照してください。

17.1.3.1 GTID 形式および格納

グローバルトランザクション識別子 (GTID) は、発生元のサーバー (ソース) でコミットされた各トランザクションに作成および関連付けられる一意の識別子です。この識別子は、それが発生したサーバーに対して一意であるだけでなく、特定のレプリケーショントポロジ内のすべてのサーバーで一意です。

GTID 割当では、ソースでコミットされるクライアントトランザクションと、レプリカで再現されるレプリケートトランザクションが区別されます。クライアントトランザクションがソースでコミットされると、そのトランザクションがバイナリログに書き込まれた場合、新しい GTID が割り当てられます。クライアントトランザクションでは、生成された番号間のギャップなしで GTID が単調に増加することが保証されます。クライアントトランザクションがバイナリログに書き込まれない場合 (たとえば、トランザクションがフィルタで除外されたか、またはトランザクションが読み取り専用だったため)、そのトランザクションには起点のサーバー上の GTID は割り当てられません。

レプリケートされたトランザクションは、オリジンのサーバー上のトランザクションに割り当てられた GTID と同じ GTID を保持します。GTID は、レプリケートされたトランザクションの実行が開始される前に存在し、レプリケートされたトランザクションがレプリカのバイナリログに書き込まれていない場合や、レプリカでフィルタで除外されている場合でも保持されます。MySQL システムテーブル `mysql.gtid_executed` は、現在アクティブなバイナリログファイルに格納されているトランザクションを除き、MySQL サーバーに適用されたすべてのトランザクションの割り当てられた GTID を保持するために使用されます。

GTID の自動スキップ機能は、ソースでコミットされたトランザクションをレプリカに複数回適用でき、一貫性の保証に役立ちます。指定された GTID を持つトランザクションが指定されたサーバーでコミットされると、同じ GTID を持つ後続のトランザクションを実行しようとする試みはそのサーバーによって無視されます。エラーは発生せず、トランザクション内のステートメントは実行されません。

指定された GTID を持つトランザクションがサーバー上で実行を開始したが、まだコミットまたはロールバックされていない場合、同じ GTID ブロックを持つサーバー上で同時トランザクションを開始しようとする試みはすべて行われます。サーバーは、同時トランザクションの実行を開始せず、クライアントに制御を返しません。トランザクションでの最初の試行がコミットまたはロールバックされると、同じ GTID でブロックしていた同時セッションが継続される可能性があります。最初の試行がロールバックされた場合、1つのコンカレントセッションがトランザクションの試行を継続し、同じ GTID でブロックしていた他のコンカレントセッションはブロックされたままになります。最初の試行がコミットされると、すべての同時セッションがブロックされなくなり、トランザクションのすべてのステートメントが自動スキップされます。

GTID は座標のペアとして表現され、次に示すように、コロン文字 (:) で区切られます。

```
GTID = source_id:transaction_id
```

`source_id` は発生元サーバーを識別します。通常、この目的にはソース `server_uuid` が使用されます。`transaction_id` は、ソースでトランザクションがコミットされた順序によって決定される順序番号です。たとえば、コミットされる

最初のトランザクションには `transaction_id` として 1 があり、同じオリジンサーバーでコミットされる 10 番目のトランザクションには 10 の `transaction_id` が割り当てられます。トランザクションに、GTID のシーケンス番号として 0 を割り当てることはできません。たとえば、UUID が `3E11FA47-71CA-11E1-9E33-C80AA9429562` の発生元サーバーでコミットされた 23 番目のトランザクションの GTID は次のとおりです。

```
3E11FA47-71CA-11E1-9E33-C80AA9429562:23
```

サーバーインスタンス上の GTID の順序番号の上限は、符号付き 64 ビット整数 (2 から 63-1 の累乗、つまり 9,223,372,036,854,775,807) の負でない値の数です。GTID が不足すると、サーバーは `binlog_error_action` によって指定されたアクションを実行します。MySQL 8.0.23 からは、サーバーインスタンスが制限に近づいたときに警告メッセージが発行されます。

トランザクションの GTID は、`mysqlbinlog` からの出力に表示され、`replication_applier_status_by_worker` などのパフォーマンススキーマレプリケーションステータステーブル内の個々のトランザクションを識別するために使用されます。`gtid_next` システム変数 (`@@GLOBAL.gtid_next`) によって格納される値は単一 GTID です。

GTID セット

GTID セットは、1 つ以上の GTID または GTID の範囲で構成されるセットです。GTID セットは、いくつかの方法で MySQL サーバーで使用されます。たとえば、`gtid_executed` および `gtid_purged` システム変数によって格納される値は GTID セットです。`START REPLICAS | SLAVE` 句 `UNTIL SQL_BEFORE_GTIDS` および `UNTIL SQL_AFTER_GTIDS` を使用すると、GTID セット内の最初の GTID までのレプリカプロセストランザクションのみを作成したり、GTID セット内の最後の GTID の後に停止したりできます。`GTID_SUBSET()` および `GTID_SUBTRACT()` の組み込み関数には、GTID セットが入力として必要です。

次に示すように、同じサーバーから発生した GTID の範囲を単一の式に縮小できます：

```
3E11FA47-71CA-11E1-9E33-C80AA9429562:1-5
```

前述の例は、`server_uuid` が `3E11FA47-71CA-11E1-9E33-C80AA9429562` である MySQL サーバーで発生した最初から 5 番目のトランザクションを表しています。次の例のように、GTID または範囲をコロンで区切って、同じサーバーから発生した GTID の複数の単一 GTID または範囲を単一の式に含めることもできます：

```
3E11FA47-71CA-11E1-9E33-C80AA9429562:1-3:11:47-49
```

GTID セットには、単一 GTID と GTID の範囲の任意の組合せを含めることができ、異なるサーバーから発生した GTID を含めることができます。この例は、複数のソースからトランザクションを適用したレプリカの `gtid_executed` システム変数 (`@@GLOBAL.gtid_executed`) に格納されている GTID セットを示しています：

```
2174B383-5441-11E8-B90A-C80AA9429562:1-3, 24DA167-0C0C-11E8-8442-00059A3C7B00:1-19
```

GTID セットがサーバー変数から返されると、UUID はアルファベット順になり、数値間隔がマージされて昇順になります。

GTID セットの構文は次のとおりです：

```
gtid_set:
  uuid_set [, uuid_set] ...
  |"

uuid_set:
  uuid:interval[:interval]...

uuid:
  hhhhhhhh-hhhh-hhhh-hhhh-hhhhhhhhhhhh

h:
  [0-9|A-F]

interval:
  n[-n]

(n >= 1)
```

mysql.gtid_executed テーブル

GTID は、`mysql` データベースの `gtid_executed` という名前のテーブルに格納されます。このテーブルの行には、GTID または GTID が表す GTID のセットごとに、元のサーバーの UUID、およびセットの開始トランザクション ID と終了トランザクション ID が含まれます。単一 GTID のみを参照する行の場合、これらの最後の 2 つの値は同じです。

`mysql.gtid_executed` テーブルは、MySQL Server のインストールまたはアップグレード時に、次に示すような `CREATE TABLE` ステートメントを使用して作成されます (まだ存在しない場合):

```
CREATE TABLE gtid_executed (  
  source_uuid CHAR(36) NOT NULL,  
  interval_start BIGINT(20) NOT NULL,  
  interval_end BIGINT(20) NOT NULL,  
  PRIMARY KEY (source_uuid, interval_start)  
)
```

警告

他の MySQL システムテーブルと同様に、このテーブルを自分で作成または変更しないでください。

`mysql.gtid_executed` テーブルは、MySQL サーバーによる内部使用のために提供されています。これにより、レプリカでバイナリロギングが無効になっているときにレプリカが GTID を使用できるようになり、バイナリログが失われたときに GTID 状態の保持が有効になります。 `RESET MASTER` を発行すると、`mysql.gtid_executed` テーブルがクリアされることに注意してください。

GTID は、`gtid_mode` が `ON` または `ON_PERMISSIVE` の場合にのみ `mysql.gtid_executed` テーブルに格納されます。バイナリロギングが無効になっている (`log_bin` が `OFF`)、または `log_slave_updates` が無効になっている場合、サーバーは、トランザクションのコミット時に、各トランザクションに属する GTID を `mysql.gtid_executed` テーブルにトランザクションとともに格納します。また、`mysql.gtid_executed` テーブル圧縮 で説明されているように、テーブルはユーザーが構成可能なレートで定期的に圧縮されます。

バイナリロギングが有効になっている (`log_bin` が `ON`)、`InnoDB` ストレージエンジンの MySQL 8.0.17 からのみ、サーバーはバイナリロギングまたはレプリカ更新ロギングが無効になっている場合と同じ方法で `mysql.gtid_executed` テーブルを更新し、トランザクションのコミット時にトランザクションごとに GTID を格納します。ただし、MySQL 8.0.17 より前のリリースやその他のストレージエンジンでは、バイナリログがローテーションされるか、サーバーがシャットダウンされたときにのみ、サーバーは `mysql.gtid_executed` テーブルを更新します。この時点で、サーバーは、前のバイナリログに書き込まれたすべてのトランザクションの GTID を `mysql.gtid_executed` テーブルに書き込みます。この状況は、MySQL 8.0.17 より前のソース、バイナリロギングが有効になっている MySQL 8.0.17 より前のレプリカ、または `InnoDB` 以外のストレージエンジンでは、次のような結果になります:

- サーバーが予期せず停止した場合、現在のバイナリログファイルから GTID のセットは `mysql.gtid_executed` テーブルに保存されません。これらの GTID は、レプリケーションを続行できるように、回復中にバイナリログファイルからテーブルに追加されます。ただし、(`--skip-log-bin` または `--disable-log-bin` を使用して) サーバーの再起動時にバイナリロギングが無効にした場合は例外です。その場合、サーバーはバイナリログファイルにアクセスして GTID を回復できないため、レプリケーションを開始できません。
- `mysql.gtid_executed` テーブルには、実行されたすべてのトランザクションの GTID の完全なレコードは保持されません。この情報は、`gtid_executed` システム変数のグローバル値によって提供されます。MySQL 8.0.17 より前のリリースおよび `InnoDB` 以外のストレージエンジンでは、`mysql.gtid_executed` テーブルをクエリーする代わりに、コミットのたびに更新される `@@GLOBAL.gtid_executed` を使用して、MySQL サーバーの GTID 状態を表します。

MySQL サーバーは、サーバーが読み取り専用モードまたはスーパー読み取り専用モードの場合でも、`mysql.gtid_executed` テーブルに書き込むことができます。MySQL 8.0.17 より前のリリースでは、これらのモードでバイナリログファイルをローテーションできます。書き込みのために `mysql.gtid_executed` テーブルにアクセスできず、バイナリログファイルが最大ファイルサイズ (`max_binlog_size`) に達しない理由でローテーションされた場合、現在のバイナリログファイルが引き続き使用されます。ローテーションをリクエストしたクライアントにエラーメッセージが返され、サーバーに警告が記録されます。書き込みのために `mysql.gtid_executed` テーブルにアクセスできず、`max_binlog_size` に到達した場合、サーバーは `binlog_error_action` 設定に従って応答します。 `IGNORE_ERROR` が設定されている場合、サーバーにエラーが記録され、バイナリロギングが停止されます。または、`ABORT_SERVER` が設定されている場合、サーバーはシャットダウンします。

mysql.gtid_executed テーブル圧縮

この間、`mysql.gtid_executed` テーブルには、同じサーバー上で発生し、次に示すような範囲を構成するトランザクション ID を持つ個々の GTID を参照する多数の行を入力できます:

```
+-----+-----+-----+
| source_uuid          | interval_start | interval_end |
+-----+-----+-----+
| 3E11FA47-71CA-11E1-9E33-C80AA9429562 | 37             | 37             |
| 3E11FA47-71CA-11E1-9E33-C80AA9429562 | 38             | 38             |
| 3E11FA47-71CA-11E1-9E33-C80AA9429562 | 39             | 39             |
| 3E11FA47-71CA-11E1-9E33-C80AA9429562 | 40             | 40             |
| 3E11FA47-71CA-11E1-9E33-C80AA9429562 | 41             | 41             |
| 3E11FA47-71CA-11E1-9E33-C80AA9429562 | 42             | 42             |
| 3E11FA47-71CA-11E1-9E33-C80AA9429562 | 43             | 43             |
| ...                                     | ...           | ...           |
```

領域を節約するために、MySQL サーバーでは、`mysql.gtid_executed` テーブルを定期的に圧縮できます。このような行の各セットを、次のようなトランザクション識別子の間隔全体にわたる単一行に置き換えます:

```
+-----+-----+-----+
| source_uuid          | interval_start | interval_end |
+-----+-----+-----+
| 3E11FA47-71CA-11E1-9E33-C80AA9429562 | 37             | 43             |
| ...                                     | ...           | ...           |
```

サーバーは、`thread/sql/compress_gtid_table` という専用のフォアグラウンドスレッドを使用して圧縮を実行できます。このスレッドは `SHOW PROCESSLIST` の出力にはリストされませんが、次に示すように、`threads` テーブルの行として表示できます:

```
mysql> SELECT * FROM performance_schema.threads WHERE NAME LIKE '%gtid%'G
***** 1. row *****
  THREAD_ID: 26
    NAME: thread/sql/compress_gtid_table
    TYPE: FOREGROUND
  PROCESSLIST_ID: 1
  PROCESSLIST_USER: NULL
  PROCESSLIST_HOST: NULL
  PROCESSLIST_DB: NULL
  PROCESSLIST_COMMAND: Daemon
  PROCESSLIST_TIME: 1509
  PROCESSLIST_STATE: Suspending
  PROCESSLIST_INFO: NULL
  PARENT_THREAD_ID: 1
    ROLE: NULL
  INSTRUMENTED: YES
    HISTORY: YES
  CONNECTION_TYPE: NULL
  THREAD_OS_ID: 18677
```

サーバーでバイナリロギングが有効になっている場合、この圧縮方法は使用されず、代わりに `mysql.gtid_executed` テーブルはバイナリログのローテーションごとに圧縮されます。ただし、バイナリロギングがサーバーで無効になっている場合、`thread/sql/compress_gtid_table` スレッドは指定された数のトランザクションが実行されるまでスリープしてから、`mysql.gtid_executed` テーブルの圧縮を実行するためにウェイクアップします。その後、同じ数のトランザクションが発生するまでスリープしてから、圧縮を再度実行するためにウェイクアップし、このループを無期限に繰り返します。テーブルが圧縮される前に経過したトランザクションの数、つまり圧縮率は、`gtid_executed_compression_period` システム変数の値によって制御されます。この値を 0 に設定すると、スレッドはウェイクアップしません。つまり、この明示的な圧縮方法は使用されません。かわりに、圧縮は必要に応じて暗黙的に行われます。

MySQL 8.0.17 から、`InnoDB` トランザクションは `InnoDB` 以外のトランザクションに対する個別のプロセスによって `mysql.gtid_executed` テーブルに書き込まれます。このプロセスは、別のスレッドである `innodb/clone_gtid_thread` によって制御されます。この GTID 永続スレッドは GTID をグループ単位で収集し、`mysql.gtid_executed` テーブルにフラッシュしてから、テーブルを圧縮します。サーバーに、`mysql.gtid_executed` テーブルに個別に書き込まれる `InnoDB` トランザクションと `InnoDB` 以外のトランザクションが混在している場合、`compress_gtid_table` スレッドによって実行される圧縮は GTID 永続スレッドの作業を妨げ、大幅に遅くなる可能性があります。このため、このリリースからは、`compress_gtid_table` スレッドがアクティブ化されないように、`gtid_executed_compression_period` を 0 に設定することをお勧めします。

MySQL 8.0.23 からは、`gtid_executed_compression_period` のデフォルト値は 0 で、InnoDB トランザクションと InnoDB 以外のトランザクションの両方が GTID 永続性スレッドによって `mysql.gtid_executed` テーブルに書き込まれます。

MySQL 8.0.17 より前のリリースでは、`gtid_executed_compression_period` のデフォルト値 1000 を使用できます。つまり、1000 トランザクションごとにテーブルの圧縮が実行されるか、別の値を選択できます。これらのリリースでは、値 0 を設定し、バイナリロギングが無効になっている場合、`mysql.gtid_executed` テーブルで明示的な圧縮は実行されないため、これを行うと、テーブルで必要になる可能性のあるディスク容量が大幅に増加する可能性がある準備をするようにしてください。

サーバーインスタンスの起動時に、`gtid_executed_compression_period` がゼロ以外の値に設定され、`thread/sql/compress_gtid_table` スレッドが起動された場合、ほとんどのサーバー構成では、`mysql.gtid_executed` テーブルに対して明示的な圧縮が実行されます。MySQL 8.0.17 より前のリリースでは、バイナリロギングが有効になっている場合、圧縮は起動時にローテーションされるバイナリログによってトリガーされます。MySQL 8.0.20 からのリリースでは、圧縮はスレッドの起動によってトリガーされます。介在するリリースでは、圧縮は起動時に行われません。

17.1.3.2 GTID ライフサイクル

GTID のライフサイクルは、次のステップで構成されます：

1. トランザクションが実行され、ソースでコミットされます。このクライアントトランザクションには、ソース UUID と、このサーバーでまだ使用されていないゼロ以外の最小のトランザクションシーケンス番号で構成される GTID が割り当てられます。GTID はソースバイナリログ (ログ内のトランザクション自体の直前) に書き込まれます。クライアントトランザクションがバイナリログに書き込まれない場合 (たとえば、トランザクションがフィルタで除外されたか、トランザクションが読み取り専用だったため)、GTID は割り当てられません。
2. GTID がトランザクションに割り当てられている場合、GTID はトランザクションの開始時にバイナリログに (`Gtid_log_event` として) 書き込むことによって、コミット時に原子的に永続化されます。バイナリログがローテーションされるか、サーバーがシャットダウンされるたびに、サーバーは以前のバイナリログファイルに書き込まれたすべてのトランザクションの GTID を `mysql.gtid_executed` テーブルに書き込みます。
3. GTID がトランザクションに割り当てられている場合、GTID は `gtid_executed` システム変数 (`@@GLOBAL.gtid_executed`) の GTID のセットに追加することで、非原子的に (トランザクションのコミット直後に) 外部化されます。この GTID セットには、コミットされたすべての GTID トランザクションのセットの表現が含まれ、サーバーの状態を表すトークンとしてレプリケーションで使用されます。バイナリロギングが有効になっている (ソースに必要な) 場合、`gtid_executed` システム変数内の GTID のセットは適用されるトランザクションの完全なレコードですが、最新の履歴がまだ現在のバイナリログファイル内にあるため、`mysql.gtid_executed` テーブルは適用されません。
4. バイナリログデータがレプリカに転送され、レプリカリレーログに格納されたあと (このプロセスで確立されたメカニズムを使用して、[セクション 17.2 「レプリケーションの実装」](#) を参照)、レプリカは GTID を読み取り、その `gtid_next` システム変数の値を GTID として設定します。これは、この GTID を使用して次のトランザクションをログに記録する必要があることをレプリカに通知します。レプリカはセッションコンテキストで `gtid_next` を設定することに注意してください。
5. レプリカは、トランザクションを処理するために、`gtid_next` で GTID の所有権を取得しているスレッドがないことを検証します。レプリケートされたトランザクション GTID を最初に読み取ってチェックすることで、トランザクション自体を処理する前に、レプリカは、この GTID を持つ以前のトランザクションがレプリカに適用されていないことだけでなく、この GTID をまだ読み取っていないが、関連付けられたトランザクションをまだコミットしていないことも保証します。そのため、複数のクライアントが同時に同じトランザクションを適用しようとする、サーバーはいずれか一方のクライアントのみを実行できるようにしてこれを解決します。レプリカの `gtid_owned` システム変数 (`@@GLOBAL.gtid_owned`) には、現在使用中の各 GTID とそれを所有するスレッドの ID が表示されます。GTID がすでに使用されている場合、エラーは発生せず、自動スキップ機能を使用してトランザクションが無視されます。
6. GTID が使用されていない場合、レプリカはレプリケートされたトランザクションを適用します。`gtid_next` はソースによってすでに割り当てられている GTID に設定されているため、レプリカはこのトランザクションに対して新しい GTID を生成しようとせず、かわりに `gtid_next` に格納されている GTID を使用します。
7. バイナリロギングがレプリカで有効になっている場合、GTID はトランザクションの開始時にバイナリログに (`Gtid_log_event` として) 書き込むことによって、コミット時に原子的に永続化されます。バイナリログがロー

ーションされるか、サーバーがシャットダウンされるたびに、サーバーは以前のバイナリログファイルに書き込まれたすべてのトランザクションの GTID を `mysql.gtid_executed` テーブルに書き込みます。

- バイナリロギングがレプリカで無効になっている場合、GTID は `mysql.gtid_executed` テーブルに直接書き込むことによって原子的に永続化されます。MySQL は、GTID をテーブルに挿入するステートメントをトランザクションに追加します。MySQL 8.0 からは、この操作は DDL ステートメントおよび DML ステートメントに対してアトミックです。この状況では、`mysql.gtid_executed` テーブルはレプリカに適用されるトランザクションの完全なレコードです。
- レプリケートされたトランザクションがレプリカでコミットされるとすぐに、GTID はレプリカの `gtid_executed` システム変数 (`@@GLOBAL.gtid_executed`) 内の GTID のセットに追加され、非原子的に外部化されます。ソースに関して、この GTID セットには、コミットされた GTID トランザクションのセットの表現が含まれます。レプリカでバイナリロギングが無効になっている場合、`mysql.gtid_executed` テーブルはレプリカに適用されたトランザクションの完全なレコードでもあります。バイナリロギングがレプリカで有効になっている場合、つまり一部の GTID がバイナリログにのみ記録される場合、`gtid_executed` システム変数内の GTID のセットのみが完全なレコードになります。

ソースで完全にフィルタで除外されたクライアントトランザクションに GTID が割り当てられていないため、`gtid_executed` システム変数のトランザクションセットに追加されたり、`mysql.gtid_executed` テーブルに追加されることはありません。ただし、レプリカで完全にフィルタで除外されたレプリケートされたトランザクションの GTID は永続化されます。バイナリロギングがレプリカで有効になっている場合、フィルタリングされたトランザクションは `Gtid_log_event` としてバイナリログに書き込まれ、その後 `BEGIN` および `COMMIT` ステートメントのみを含む空のトランザクションが続きます。バイナリロギングが無効になっている場合は、フィルタ処理されたトランザクションの GTID が `mysql.gtid_executed` テーブルに書き込まれます。フィルタ処理されたトランザクションの GTID を保持することで、`mysql.gtid_executed` テーブルおよび GTID のセットを `gtid_executed` システム変数に確実に圧縮できます。また、[セクション 17.1.3.3 「GTID 自動配置」](#) で説明されているように、レプリカがソースに再接続した場合、フィルタで除外されたトランザクションが再度取得されないようにします。

マルチスレッドレプリカ (`slave_parallel_workers > 0` を使用) では、トランザクションをパラレルに適用できるため、レプリケートされたトランザクションは順序どおりにコミットできません (`slave_preserve_commit_order=1` が設定されていない場合)。その場合、`gtid_executed` システム変数内の GTID のセットには、GTID 間にギャップがある複数の GTID 範囲が含まれます。(ソースまたはシングルスレッドレプリカでは、数値間のギャップなしで GTID が単調に増加します。) マルチスレッドのレプリカのギャップは、最後に適用されたトランザクション間でのみ発生し、レプリケーションの進行に応じて埋められます。 `STOP REPLICAS | SLAVE` ステートメントを使用してレプリケーションスレッドが正常に停止されると、ギャップが埋められるように進行中のトランザクションが適用されます。サーバー障害や `KILL` ステートメントを使用してレプリケーションスレッドを停止した場合、ギャップが残ることがあります。

GTID にはどのような変更が割り当てられますか。

一般的なシナリオは、サーバーがコミットされたトランザクションに対して新しい GTID を生成することです。ただし、GTID はトランザクション以外の他の変更にも割り当てることができ、場合によっては単一のトランザクションに複数の GTID を割り当てることができます。

バイナリログに書き込まれるすべてのデータベース変更 (DDL または DML) に GTID が割り当てられます。これには、自動コミットされる変更と、`BEGIN` および `COMMIT` または `START TRANSACTION` ステートメントを使用してコミットされる変更が含まれます。GTID は、データベースの作成、変更または削除、およびプロシージャ、ファンクション、トリガー、イベント、ビュー、ユーザー、ロールまたは付与などのテーブル以外のデータベースオブジェクトにも割り当てられます。

非トランザクション更新およびトランザクション更新に GTID が割り当てられます。また、非トランザクション更新では、バイナリログキャッシュへの書き込み中にディスク書き込み障害が発生し、そのためバイナリログにギャップが作成された場合、生成されるインシデントログイベントに GTID が割り当てられます。

バイナリログ内の生成されたステートメントによってテーブルが自動的に削除されると、GTID がステートメントに割り当てられます。レプリカが開始したばかりのソースからイベントの適用を開始し、ステートメントベースレプリケーションが使用中 (`binlog_format=STATEMENT`) で、開いている一時テーブルを持つユーザーセッションが切断されると、一時テーブルは自動的に削除されます。 `MEMORY` ストレージエンジンを使用するテーブルは、サーバーの起動後にはじめてアクセスされたときに自動的に削除されます。これは、シャットダウン中に行が失われた可能性があるためです。

トランザクションが起点のサーバー上のバイナリログに書き込まれない場合、サーバーは GTID を割り当てません。これには、ロールバックされたトランザクションと、バイナリロギング中に実行されたトランザクションがオリジ

ンのサーバーでグローバルに (サーバー構成で `--skip-log-bin` が指定された状態で) またはセッションに対して (`SET @@SESSION.sql_log_bin = 0`) 無効化された状態で含まれます。これには、行ベースレプリケーションが使用されている場合の `no-op` トランザクションも含まれます (`binlog_format=ROW`)。

XA トランザクションには、トランザクションの `XA PREPARE` フェーズおよびトランザクションの `XA COMMIT` または `XA ROLLBACK` フェーズ用に個別の GTID が割り当てられます。XA トランザクションは、障害発生時にユーザーがコミットまたはロールバックできるように永続的に準備されます (レプリケーショントポロジでは、別のサーバーへのフェイルオーバーが含まれる場合があります)。したがって、トランザクションの 2 つの部分は個別にレプリケートされるため、ロールバックされる非 XA トランザクションに GTID がなくても、独自の GTID が必要です。

次の特殊なケースでは、単一のステートメントで複数のトランザクションを生成できるため、複数の GTID を割り当てることができます:

- 複数のトランザクションをコミットするストアードプロシージャが起動されます。プロシージャがコミットするトランザクションごとに GTID が 1 つ生成されます。
- 複数テーブルの `DROP TABLE` ステートメントは、異なるタイプのテーブルを削除します。いずれかのテーブルがアトミック DDL をサポートしていないストレージエンジンを使用している場合、またはいずれかのテーブルが一時的テーブルである場合は、複数の GTID を生成できます。
- `CREATE TABLE ... SELECT` ステートメントは、行ベースのレプリケーションが使用中 (`binlog_format=ROW`) の場合に発行されます。`CREATE TABLE` 処理に対して GTID が生成され、行挿入処理に対して GTID が生成されません。

gtid_next システム変数

デフォルトでは、ユーザーセッションでコミットされた新しいトランザクションの場合、サーバーは自動的に新しい GTID を生成して割り当てます。トランザクションがレプリカに適用されると、オリジンのサーバーからの GTID が保持されます。この動作は、`gtid_next` システム変数のセッション値を設定することで変更できます:

- `gtid_next` が `AUTOMATIC` (デフォルト) に設定され、トランザクションがコミットされてバイナリログに書き込まれると、サーバーは自動的に新しい GTID を生成して割り当てます。トランザクションが別の理由でロールバックされるか、バイナリログに書き込まれない場合、サーバーは GTID を生成して割り当てません。
- `gtid_next` を有効な GTID (コロンで区切られた UUID とトランザクション順序番号で構成) に設定すると、サーバーはその GTID をトランザクションに割り当てます。この GTID は、トランザクションがバイナリログに書き込まれない場合や、トランザクションが空の場合でも、`gtid_executed` に割り当てられて追加されます。

`gtid_next` を特定の GTID に設定し、トランザクションがコミットまたはロールバックされた後、明示的な `SET @@SESSION.gtid_next` ステートメントを他のステートメントの前に発行する必要があります。GTID を明示的に割り当てない場合は、GTID 値を `AUTOMATIC` に戻すためにこれを使用できます。

レプリケーションアプライヤスレッドがレプリケートされたトランザクションを適用する場合、この手法を使用して、`@@SESSION.gtid_next` をオリジンサーバーに割り当てられているレプリケートされたトランザクションの GTID に明示的に設定します。これは、レプリカによって生成および割り当てられる新しい GTID ではなく、起点のサーバーからの GTID が保持されることを意味します。また、バイナリロギングまたはレプリカ更新ロギングがレプリカで無効になっている場合、またはトランザクションが `no-op` であるかレプリカでフィルタで除外されている場合でも、GTID がレプリカ上の `gtid_executed` に追加されることを意味します。

クライアントは、トランザクションを実行する前に `@@SESSION.gtid_next` を特定の GTID に設定することで、レプリケートされたトランザクションをシミュレートできます。この手法は、GTID を保持するためにクライアントがリプレイできるバイナリログのダンプを生成するために、`mysqlbinlog` によって使用されます。クライアントを介してコミットされたシミュレートされたレプリケートされたトランザクションは、レプリケーションアプライヤスレッドを介してコミットされたレプリケートされたトランザクションと完全に同等であり、実際には区別できません。

gtid_purged システム変数

`gtid_purged` システム変数 (`@@GLOBAL.gtid_purged`) 内の GTID のセットには、サーバー上でコミットされたが、サーバー上のバイナリログファイルには存在しないすべてのトランザクションの GTID が含まれています。`gtid_purged` は、`gtid_executed` のサブセットです。GTID の次のカテゴリが `gtid_purged` にあります:

- レプリカでバイナリロギングを無効にしてコミットされたレプリケートされたトランザクションの GTID。

- 現在パージされているバイナリログファイルに書き込まれたトランザクションの GTID。
- ステートメント `SET @@GLOBAL.gtid_purged` によってセットに明示的に追加された GTID。

特定の GTID セット内のトランザクションが適用されたことをサーバーに記録するために、`gtid_purged` の値を変更できますが、それらはサーバー上のバイナリログには存在しません。GTID を `gtid_purged` に追加すると、`gtid_executed` にも追加されます。このアクションのユースケースの例は、サーバー上の 1 つ以上のデータベースのバックアップをリストアするが、サーバー上のトランザクションを含む関連するバイナリログがない場合です。MySQL 8.0 より前は、`gtid_executed` (および `gtid_purged`) が空の場合にのみ、`gtid_purged` の値を変更できました。MySQL 8.0 からは、この制限は適用されず、`gtid_purged` 内の GTID セット全体を指定された GTID セットに置き換えるか、指定された GTID セットを `gtid_purged` 内の GTID に追加するかを選択することもできます。これを行う方法の詳細は、`gtid_purged` の説明を参照してください。

`gtid_executed` および `gtid_purged` システム変数内の GTID のセットは、サーバーの起動時に初期化されます。すべてのバイナリログファイルは、以前のすべてのバイナリログファイル (前のファイル `Previous_gtid_log_event` の GTID および前のファイル自体のすべての `Gtid_log_event` の GTID から構成される) 内の GTID のセットを含むイベント `Previous_gtid_log_event` から始まります。もっとも古いバイナリログファイルと最新のバイナリログファイル内の `Previous_gtid_log_event` の内容は、サーバーの起動時に `gtid_executed` および `gtid_purged` セットを計算するために使用されます:

- `gtid_executed` は、最新のバイナリログファイル内の `Previous_gtid_log_event` 内の GTID、そのバイナリログファイル内のトランザクションの GTID、および `mysql.gtid_executed` テーブルに格納されている GTID の結合として計算されます。この GTID セットには、現在サーバー上のバイナリログファイル内にあるかどうかに関係なく、サーバー上で使用された (または `gtid_purged` に明示的に追加された) GTID がすべて含まれます。サーバー (@@GLOBAL.gtid_owned) で現在処理されているトランザクションの GTID は含まれません。
- `gtid_purged` は、まず最新のバイナリログファイルに `Previous_gtid_log_event` の GTID を追加し、そのバイナリログファイルにトランザクションの GTID を追加することによって計算されます。この手順では、サーバー (`gtids_in_binlog`) のバイナリログに現在記録されている GTID、または一度も記録されていた GTID のセットを提供します。次に、もっとも古いバイナリログファイル内の `Previous_gtid_log_event` 内の GTID が `gtids_in_binlog` から差し引かれます。この手順では、サーバー (`gtids_in_binlog_not_purged`) のバイナリログに現在記録されている GTID のセットを提供します。最後に、`gtids_in_binlog_not_purged` が `gtid_executed` から減算されます。結果は、サーバー上で使用されているが、現在サーバー上のバイナリログファイルに記録されていない GTID のセットであり、この結果は `gtid_purged` の初期化に使用されます。

これらの計算に MySQL 5.7.7 以前のバイナリログが含まれている場合は、`gtid_executed` および `gtid_purged` に対して不正な GTID セットを計算でき、サーバーがあとで再起動されても正しくありません。詳細は、GTID セットを計算するためにバイナリログを繰り返す方法を制御する `binlog_gtid_simple_recovery` システム変数の説明を参照してください。説明されているいずれかの状況がサーバーに当てはまる場合は、サーバー構成ファイルで `binlog_gtid_simple_recovery=FALSE` を設定してから起動します。この設定により、サーバーは (もっとも新しいものともっとも古いものだけでなく) すべてのバイナリログファイルを反復して GTID イベントが表示される場所を見つけます。GTID イベントのないバイナリログファイルがサーバーに多数ある場合、このプロセスには時間がかかることがあります。

GTID 実行履歴のリセット

サーバーで GTID 実行履歴をリセットする必要がある場合は、`RESET MASTER` ステートメントを使用します。たとえば、テストクエリーを実行して新しい GTID 対応サーバーでレプリケーション設定を検証した後、または新しいサーバーをレプリケーショングループに結合するが、グループレプリケーションで受け入れられない不要なローカルトランザクションが含まれている場合に、これを行う必要があります。

警告

必要な GTID 実行履歴およびバイナリログファイルが失われないように、`RESET MASTER` を慎重に使用してください。

`RESET MASTER` を発行する前に、サーバーのバイナリログファイルとバイナリログインデックスファイル (ある場合) のバックアップがあることを確認し、`gtid_executed` システム変数のグローバル値に保持されている GTID セットを取得して保存します (たとえば、`SELECT @@GLOBAL.gtid_executed` ステートメントを発行して結果を保存します)。その GTID セットから不要なトランザクションを削除する場合は、`mysqlbinlog` を使用してトランザクションの内容を調べ、値がなく、保存またはレプリケートが必要なデータがなく、サーバーでデータが変更されなかったことを確認します。

`RESET MASTER` を発行すると、次のリセット操作が実行されます:

- `gtid_purged` システム変数の値は空の文字列 ("") に設定されます。
- `gtid_executed` システム変数のグローバル値 (セッション値ではない) が空の文字列に設定されています。
- `mysql.gtid_executed` テーブルがクリアされます (`mysql.gtid_executed` テーブル を参照)。
- サーバーでバイナリロギングが有効になっている場合、既存のバイナリログファイルは削除され、バイナリログインデックスファイルはクリアされます。

サーバーがバイナリロギングが無効になっているレプリカであっても、`RESET MASTER` は GTID 実行履歴をリセットする方法であることに注意してください。 `RESET REPLICA | SLAVE` は GTID 実行履歴には影響しません。

17.1.3.3 GTID 自動配置

GTID は、ソースとレプリカ間のデータフローを開始、停止、または再開するためのポイントを決定するために以前に必要なファイルオフセットペアを置き換えます。GTID が使用されている場合、レプリカがソースと同期するために必要なすべての情報は、レプリケーションデータストリームから直接取得されます。

GTID ベースのレプリケーションを使用してレプリカを開始するには、`CHANGE REPLICATION SOURCE TO` ステートメント (MySQL 8.0.23 から) または `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 より前) で `SOURCE_AUTO_POSITION | MASTER_AUTO_POSITION` オプションを有効にする必要があります。代替の `SOURCE_LOG_FILE | MASTER_LOG_FILE` および `SOURCE_LOG_POS | MASTER_LOG_POS` オプションでは、ログファイルの名前とファイル内の開始位置を指定しますが、GTID ではレプリカにこの非ローカルデータは必要ありません。GTID ベースのレプリケーションを使用してソースおよびレプリカを構成および起動する完全な手順については、[セクション 17.1.3.4 「GTID を使用したレプリケーションのセットアップ」](#) を参照してください。

`SOURCE_AUTO_POSITION | MASTER_AUTO_POSITION` オプションはデフォルトで無効になっています。レプリカでマルチソースレプリケーションが有効になっている場合は、該当するレプリケーションチャンネルごとにオプションを設定する必要があります。 `SOURCE_AUTO_POSITION | MASTER_AUTO_POSITION` オプションを再度無効にすると、レプリカはファイルベースレプリケーションに戻ります。この場合、`SOURCE_LOG_FILE | MASTER_LOG_FILE` または `SOURCE_LOG_POS | MASTER_LOG_POS` オプションのいずれかまたは両方も指定する必要があります。

レプリカで GTID が有効 (`GTID_MODE=ON`、`ON_PERMISSIVE`、または `OFF_PERMISSIVE`) で `MASTER_AUTO_POSITION` オプションが有効になっている場合、ソースへの接続に対して自動配置がアクティブ化されます。接続を成功させるには、ソースに `GTID_MODE=ON` が設定されている必要があります。初期ハンドシェイクでは、レプリカは、すでに受信、コミット、またはその両方を行ったトランザクションを含む GTID セットを送信します。この GTID セットは、`gtid_executed` システム変数 (`@@GLOBAL.gtid_executed`) 内の GTID のセットの結合、および受信したトランザクションとしてパフォーマンススキーマ `replication_connection_status` テーブルに記録された GTID のセット (`SELECT RECEIVED_TRANSACTION_SET FROM PERFORMANCE_SCHEMA.replication_connection_status` ステートメントの結果) と等しくなります。

ソースは、GTID がレプリカによって送信される GTID セットに含まれていないバイナリログに記録されたすべてのトランザクションを送信することによって応答します。これを行うには、ソースはまず、各バイナリログファイルのヘッダーにある `Previous_gtid_log_event` を最新のものからチェックして、作業を開始する適切なバイナリログファイルを識別します。ソースは、レプリカが欠落しているトランザクションを含まない最初の `Previous_gtid_log_event` を検出すると、そのバイナリログファイルから始まります。この方法は効率的で、大量のバイナリログファイルによってレプリカがソースの背後にある場合にのみかなりの時間がかかります。次に、ソースはそのバイナリログファイル内のトランザクションとそれ以降のファイルを現在のファイルまで読み取り、レプリカが欠落している GTID を含むトランザクションを送信し、レプリカによって送信された GTID セット内のトランザクションをスキップします。レプリカが最初に欠落しているトランザクションを受信するまでの経過時間は、バイナリログファイル内のオフセットによって異なります。この交換により、レプリカがまだ受信またはコミットしていない GTID を持つトランザクションのみがソースから送信されるようになります。ダイヤモンドトポロジの場合と同様に、レプリカが複数のソースからトランザクションを受信する場合、自動スキップ機能によってトランザクションが 2 回適用されないようにします。

ソースによって送信されるべきトランザクションのいずれかがソースバイナリログからパーズされているか、別の方法で `gtid_purged` システム変数の GTID セットに追加されている場合、ソースはエラー `ER_MASTER_HAS_PURGED_REQUIRED_GTIDS` をレプリカに送信し、レプリケーションは開始しません。欠落

しているパージ済トランザクションの GTID が識別され、警告メッセージ `ER_FOUND_MISSING_GTIDS` のソースエラーログにリストされます。ソースのキャッチアップに必要なトランザクション履歴の一部がパージされているため、レプリカはこのエラーから自動的にリカバリできません。 `MASTER_AUTO_POSITION` オプションを有効にせずに再接続しようとする、レプリカ上のパージされたトランザクションが失われます。この状況からリカバリする正しいアプローチは、レプリカが `ER_FOUND_MISSING_GTIDS` メッセージにリストされている欠落トランザクションを別のソースからレプリケートするか、レプリカをより新しいバックアップから作成された新しいレプリカに置き換えることです。状況が再度発生しないように、ソースでバイナリログの有効期限 (`binlog_expire_logs_seconds`) を変更することを検討してください。

トランザクションの交換中に、GTID 内のソース UUID を持つトランザクションをレプリカが受信またはコミットしたが、ソース自体にそれらのレコードがないことが判明した場合、ソースはレプリカにエラー `ER_SLAVE_HAS_MORE_GTIDS_THAN_MASTER` を送信し、レプリケーションは開始しません。この状況は、`sync_binlog=1` セットを持たないソースで電源障害またはオペレーティングシステムクラッシュが発生し、まだバイナリログファイルに同期されていないがレプリカによって受信されたコミット済トランザクションが失われた場合に発生する可能性があります。再起動後にクライアントがソースでトランザクションをコミットすると、ソースとレプリカが相違する可能性があります。これは、ソースとレプリカが異なるトランザクションに同じ GTID を使用している状況につながる可能性があります。この状況からリカバリする正しい方法は、ソースとレプリカが相違しているかどうかを手動で確認することです。同じ GTID が異なるトランザクションに使用されている場合は、必要に応じて個々のトランザクションに対して手動による競合解決を実行するか、レプリケーショントポロジからソースまたはレプリカを削除する必要があります。問題がソースで欠落しているトランザクションのみである場合は、かわりにソースをレプリカにし、レプリケーショントポロジ内の他のサーバーで捕捉できるようにしてから、必要に応じて再度ソースにすることができます。

ダイヤモンドトポロジ内のマルチソースレプリカ (レプリカが複数のソースからレプリケートされ、共通ソースからレプリケートされる) の場合、GTID ベースのレプリケーションが使用されているときは、マルチソースレプリカ上のすべてのチャンネルでレプリケーションフィルタまたはその他のチャンネル構成が同一であることを確認してください。GTID ベースのレプリケーションでは、フィルタはトランザクションデータにのみ適用され、GTID はフィルタで除外されません。これは、レプリカの GTID セットがソースと一貫性が保たれるようにするためです。つまり、毎回フィルタで除外されたトランザクションを再取得せずに GTID 自動配置を使用できます。ダウンストリームレプリカがマルチソースで、ダイヤモンドトポロジの複数のソースから同じトランザクションを受信する場合、ダウンストリームレプリカには複数のバージョンのトランザクションが含まれるようになり、結果はトランザクションを最初に適用するチャンネルによって異なります。トランザクションの GTID が最初のチャンネルによって `gtid_executed` セットに追加されたため、GTID 自動スキップを使用してトランザクションをスキップしようとする 2 つ目のチャンネル。チャンネルのフィルタリングが同一の場合、トランザクションのすべてのバージョンに同じデータが含まれているため、結果は同じであるため、問題はありません。ただし、チャンネルのフィルタリングが異なると、データベースに一貫性がなくなる可能性があり、レプリケーションがハングする可能性があります。

17.1.3.4 GTID を使用したレプリケーションのセットアップ

このセクションでは、MySQL 8.0 で GTID ベースレプリケーションを構成および起動するためのプロセスについて説明します。これは、ソースサーバーを初めて起動するか、停止できることを前提とした「コールドスタート」プロジェクトです。GTID を使用した実行中のソースサーバーからのレプリカのプロジェクトの詳細は、[セクション 17.1.3.5 「フェイルオーバーおよびスケールアウトでの GTID の使用」](#) を参照してください。オンラインでの GTID モードの変更については、[セクション 17.1.4 「オンラインサーバーでの GTID モードの変更」](#) を参照してください。

1 つのソースと 1 つのレプリカで構成される、最も単純な GTID レプリケーショントポロジのためのこの起動プロセスの主なステップは、次のとおりです:

1. レプリケーションがすでに動作している場合、両方のサーバーを読み取り専用にするのでそれらを同期します。
2. 両方のサーバーを停止します。
3. GTID を有効にして両方のサーバーを再起動し、正しいオプションを構成します。

説明したサーバーを起動するために必要な `mysqld` オプションについては、このセクションの後半の例で説明します。

4. ソースをレプリケーションデータソースとして使用し、自動配置を使用するようレプリカに指示します。この手順の実施に必要な SQL ステートメントは、このセクションの後半の例で説明します。
5. 新しいバックアップを作成します。GTID のないトランザクションを含むバイナリログは GTID が有効になっているサーバーでは使用できないため、この時点より前に作成されたバックアップは新しい構成では使用できません。

- レプリカを起動し、両方のサーバーで読み専用モードを無効にして、更新を受け入れることができるようにします。

次の例では、MySQL バイナリログの位置ベースのレプリケーションプロトコルを使用して、2つのサーバーがすでにソースおよびレプリカとして実行されています。新しいサーバーから開始する場合、レプリケーション接続用の特定のユーザーの追加の詳細は [セクション17.1.2.3「レプリケーション用ユーザーの作成」](#) を、`server_id` 変数の設定の詳細は [セクション17.1.2.1「レプリケーションソース構成の設定」](#) を参照してください。次の例は、`mysqld` 起動オプションをサーバーオプションファイルに格納する方法を示しています。詳細は、[セクション4.2.2.2「オプションファイルの使用」](#) を参照してください。または、`mysqld` の実行時に起動オプションを使用することもできます。

後続のほとんどの手順では、`SUPER` 権限を持つ MySQL `root` アカウントまたは別の MySQL ユーザーアカウントを使用する必要があります。`mysqladmin shutdown` には、`SUPER` 権限または `SHUTDOWN` 権限が必要です。

手順 1: サーバーを同期します。このステップは、GTID を使用せずにすでにレプリケートされているサーバーを操作する場合にのみ必要です。新しいサーバーの場合は、ステップ 3 に進みます。次のコマンドを発行して、各サーバーで `read_only` システム変数を `ON` に設定し、サーバーを読み専用にします:

```
mysql> SET @@GLOBAL.read_only = ON;
```

進行中のすべてのトランザクションがコミットまたはロールバックされるまで待機します。その後、レプリカがソースをキャッチアップできるようにします。続行する前にレプリカがすべての更新を処理したことを確認することが非常に重要です。

ポイントインタイムのバックアップおよびリストアなど、レプリケーション以外にバイナリログを使用する場合は、GTID のないトランザクションを含む古いバイナリログが不要になるまで待機します。理想的には、サーバーがすべてのバイナリログをパージし、既存のバックアップが期限切れになるまで待機します。

重要

GTID が有効になっているサーバーでは GTID のないトランザクションを含むログを使用できないことを理解することが重要です。続行する前に、GTID のないトランザクションがトポロジ内のどこにも存在しないことを確認する必要があります。

手順 2: 両方のサーバーを停止します。ここで示すように、`mysqladmin` を使用して各サーバーを停止します。ここで、`username` はサーバーをシャットダウンするのに十分な権限を持つ MySQL ユーザーのユーザー名です。

```
shell> mysqladmin -uusername -p shutdown
```

次に、プロンプトにこのユーザーのパスワードを指定します。

ステップ 3: GTID が有効な両方のサーバーを起動。GTID ベースのレプリケーションを有効にするには、GTID ベースのレプリケーションで安全なステートメントのみがログに記録されるように、`gtid_mode` 変数を `ON` に設定し、`enforce_gtid_consistency` 変数を有効にして GTID モードで各サーバーを起動する必要があります。例:

```
gtid_mode=ON
enforce-gtid-consistency=ON
```

また、レプリカ設定を構成する前に、`--skip-slave-start` オプションを使用してレプリカを開始する必要があります。GTID 関連のオプションおよび変数の詳細は、[セクション17.1.6.5「グローバルトランザクション ID システム変数」](#) を参照してください。

`mysql.gtid_executed` テーブルの使用時に GTID を使用するためにバイナリロギングを有効にする必要はありません。レプリケートを可能にするには、ソースサーバーで常にバイナリロギングが有効になっている必要があります。ただし、複製サーバーは GTID を使用できませんが、バイナリロギングは使用できません。レプリカサーバーでバイナリロギングを無効にする必要がある場合は、レプリカの `--skip-log-bin` および `--log-slave-updates=OFF` オプションを指定して無効にできます。

ステップ 4: GTID ベースの自動配置を使用するようにレプリカを構成。GTID ベースのトランザクションを含むソースをレプリケーションデータソースとして使用し、ファイルベースの配置ではなく GTID ベースの自動配置を使用するようにレプリカに指示します。レプリカに対して (MySQL 8.0.23 の) `CHANGE REPLICATION SOURCE TO` ステートメントまたは (MySQL 8.0.23 の前の) `CHANGE MASTER TO` ステートメントを発行し、ステートメントに `SOURCE_AUTO_POSITION` | `MASTER_AUTO_POSITION` オプションを含めて、ソーストランザクションが GTID によって識別されることをレプリカに伝えます。

また、ソースホスト名とポート番号に適切な値を指定し、レプリカがソースに接続するために使用できるレプリケーションユーザーアカウントのユーザー名とパスワードを指定する必要がある場合もあります。これらがステップ 1 より前にすでに設定されており、それ以上変更する必要がない場合は、ここに示すステートメントから対応するオプションを安全に省略できます。

```
mysql> CHANGE MASTER TO
> MASTER_HOST = host,
> MASTER_PORT = port,
> MASTER_USER = user,
> MASTER_PASSWORD = password,
> MASTER_AUTO_POSITION = 1;
```

Or from MySQL 8.0.23:

```
mysql> CHANGE REPLICATION SOURCE TO
> SOURCE_HOST = host,
> SOURCE_PORT = port,
> SOURCE_USER = user,
> SOURCE_PASSWORD = password,
> SOURCE_AUTO_POSITION = 1;
```

ステップ 5: 新しいバックアップの作成。 GTID を有効にする前に作成された既存のバックアップは、GTID を有効にしたこれらのサーバーでは使用できなくなりました。この時点で新しいバックアップを作成して、使用可能なバックアップなしで残されないようにします。

たとえば、バックアップを作成しているサーバーで `FLUSH LOGS` を実行できます。次に、明示的にバックアップを取るか、設定した定期バックアップルーチンの次の反復を待機します。

ステップ 6: レプリカを起動し、読取り専用モードを無効にします。 次のようにレプリカを起動します:

```
mysql> START SLAVE;
Or from MySQL 8.0.22:
mysql> START REPLICA;
```

次のステップは、ステップ 1 でサーバーを読取り専用構成した場合にのみ必要です。サーバーが更新の受け入れを再度開始できるようにするには、次のステートメントを発行します:

```
mysql> SET @@GLOBAL.read_only = OFF;
```

GTID ベースのレプリケーションが実行され、以前と同様にソースでアクティビティを開始 (または再開) できます。 [セクション 17.1.3.5 「フェイルオーバーおよびスケールアウトでの GTID の使用」](#) では、GTID 使用時の新しいレプリカの作成について説明します。

17.1.3.5 フェイルオーバーおよびスケールアウトでの GTID の使用

MySQL レプリケーションをグローバルトランザクション識別子 (GTID) とともに使用して新しいレプリカをプロビジョニングする場合、スケールアウトに使用し、フェイルオーバーのために必要に応じてソースに昇格する方法が多数あります。このセクションでは、次の手法について説明します:

- [単純なレプリケーション](#)
- [レプリカへのデータおよびトランザクションのコピー](#)
- [空のトランザクションの注入](#)
- [gtid_purged によるトランザクションの除外](#)
- [GTID モードの複製の復元](#)

グローバルトランザクション識別子は、特にレプリケーションデータフローおよびフェイルオーバーアクティビティの一般管理を簡易化するために、MySQL Replication に追加されました。各識別子は、全体でトランザクションを構成するバイナリログイベントセットを一意に識別します。GTID はデータベースに変更を適用する際に重要な役割を果たします。サーバーは、以前に処理済みと認識している識別子のトランザクションを自動的にスキップします。この動作は、自動レプリケーションポジショニングおよび正確なフェイルオーバーのために重要です。

トランザクションを構成する識別子とイベントセットとの間のマッピングは、バイナリログで取得されます。このことは、別の既存のサーバーからのデータで新しいサーバーをプロビジョニングする際に、いくつかの課題を提起しま

す。新しいサーバーに設定された識別子を再現するには、古いサーバーから新しいサーバーに識別子をコピーし、識別子と実際のイベントの関係を保持する必要があります。これは、フェイルオーバーまたはスイッチオーバー時に新しいソースになる候補としてすぐに使用可能なレプリカをリストアするために必要です。

単純なレプリケーション。 新しいサーバーですべての識別子とトランザクションを再現する最も簡単な方法は、新しいサーバーを実行履歴全体を持つソースのレプリカにし、両方のサーバーでグローバルトランザクション識別子を有効にすることです。詳細については、[セクション17.1.3.4「GTIDを使用したレプリケーションのセットアップ」](#)を参照してください。

レプリケーションが開始されると、新しいサーバーはバイナリログ全体をソースからコピーするため、すべての GTID に関するすべての情報を取得します。

この方法は単純で効果的ですが、レプリカがソースからバイナリログを読み取る必要があります。新しいレプリカがソースに追いつくまでに比較的長い時間がかかる場合があるため、この方法は高速フェイルオーバーやバックアップからの復元には適していません。このセクションでは、バイナリログファイルを新しいサーバーにコピーして、ソースからすべての実行履歴をフェッチしない方法について説明します。

レプリカへのデータおよびトランザクションのコピー。 ソースサーバーが以前に多数のトランザクションを処理している場合、トランザクション履歴全体の実行に時間がかかることがあり、これは新しいレプリカの設定時の大きなボトルネックを表している可能性があります。この要件をなくすために、データセットのスナップショット、バイナリログおよびソースサーバーに含まれるグローバルトランザクション情報を新しいレプリカにインポートできます。スナップショットが作成されるサーバーは、ソースまたはそのレプリカのいずれかになりますが、データをコピーする前に、サーバーが必要なすべてのトランザクションを処理していることを確認する必要があります。

この方法にはいくつかのバリエーションがあります。違いは、データダンプとバイナリログからのトランザクションがレプリカに転送される方法です。次に概要を示します：

- データセット
1. ソースサーバーで `mysqldump` を使用してダンプファイルを作成します。バイナリロギング情報を含む `CHANGE REPLICATION SOURCE TO|CHANGE MASTER TO` ステートメントを含めるように、`mysqldump` オプション `--master-data` (デフォルト値は 1) を設定します。実行されたトランザクションに関する情報をダンプに含めるには、`--set-gtid-purged` オプションを `AUTO` (デフォルト) または `ON` に設定します。次に、`mysql` クライアントを使用して、ダンプファイルをターゲットサーバーにインポートします。
 2. または、RAW データファイルを使用してソースサーバーのデータスナップショットを作成し、[セクション17.1.2.5「データスナップショットの方法の選択」](#) の手順に従ってこれらのファイルをターゲットサーバーにコピーします。InnoDB テーブルを使用する場合、MySQL Enterprise Backup コンポーネントから `mysqlbackup` コマンドを使用して、一貫性のあるスナップショットを作成できます。このコマンドは、レプリカで使用されるスナップショットに対応するログ名とオフセットを記録します。MySQL Enterprise Backup は MySQL Enterprise サブスクリプションの一部として同梱される製品です。詳細は、[セクション30.2「MySQL Enterprise Backup の概要」](#) を参照してください。
 3. または、ソースサーバーとターゲットサーバーの両方を停止し、ソースデータディレクトリの内容を新しいレプリカデータディレクトリにコピーしてから、レプリカを再起動します。この方法を使用する場合は、GTID ベースのレプリケーション、つまり `gtid_mode=ON` でレプリカを構成する必要があります。この方法の手順および重要な情報については、[セクション17.1.2.8「レプリケーション環境へのレプリカの追加」](#) を参照してください。

トランザクション履歴

ソースサーバーのバイナリログに完全なトランザクション履歴がある (GTID セット `@@GLOBAL.gtid_purged` が空である) 場合は、次の方法を使用できます。

1. `--read-from-remote-server` および `--read-from-remote-master` オプションを指定して、`mysqlbinlog` を使用してバイナリログをソースサーバーから新しいレプリカにインポートします。
2. または、ソースサーバーのバイナリログファイルをレプリカにコピーします。`--read-from-remote-server` および `--raw` オプションを指定した `mysqlbinlog` を使用して、レプリカからコピーを作成できます。これらは、`mysqlbinlog > file` (`--raw` オプションなし) を使用してバイナリログファイルを SQL ファイルにエクスポートし、これらのファイルを `mysql` クライアントに渡して処理することでレプリカに読み取ることができます。すべてのバイナリログファイルが、複数の接続ではなく単一の `mysql` プロセスを使用して処理されていることを確認します。例：

```
shell> mysqlbinlog copied-binlog.000001 copied-binlog.000002 | mysql -u root -p
```

詳細は、[セクション4.6.8.3「バイナリログファイルのバックアップのための mysqlbinlog の使用」](#)を参照してください。

この方法には、ほとんどすぐに新しいサーバーを使用できるという利点があります。スナップショットまたはダンブファイルのリプレイ中にコミットされたトランザクションのみ、既存のソースから取得する必要があります。つまり、レプリカの可用性は瞬時ではありませんが、レプリカがこれらの少数の残りのトランザクションに追いつくには比較的短い時間しか必要ありません。

通常、バイナリログをターゲットサーバーに事前にコピーする方が、ソースからリアルタイムでトランザクション実行履歴全体を読み取るより高速です。ただし、サイズやその他の考慮事項により、必要なときにこれらのファイルをターゲットに移動することが常に実現できるとはかぎらない場合があります。このセクションで説明する新しいレプリカをプロビジョニングするための残りの2つの方法では、他の方法を使用してトランザクションに関する情報を新しいレプリカに転送します。

空のトランザクションの注入。ソースグローバル `gtid_executed` 変数には、ソースで実行されたすべてのトランザクションのセットが含まれます。新しいサーバーをプロビジョニングするためにスナップショットを作成するときにバイナリログをコピーする代わりに、スナップショットが作成されたサーバーで `gtid_executed` の内容に注目できます。新しいサーバーをレプリケーションチェーンに追加する前に、次のように、ソース `gtid_executed` に含まれるトランザクション識別子ごとに新しいサーバーで空のトランザクションをコミットします：

```
SET GTID_NEXT='aaa-bbb-ccc-ddd:N';  
  
BEGIN;  
COMMIT;  
  
SET GTID_NEXT='AUTOMATIC';
```

空のトランザクションを使用してすべてのトランザクション識別子をこの方法で回復したら、次に示すようにレプリカバイナリログをフラッシュしてパージする必要があります。ここで、`N` は現在のバイナリログファイル名のゼロ以外の接尾辞です：

```
FLUSH LOGS;  
PURGE BINARY LOGS TO 'source-bin.00000N';
```

後でソースに昇格された場合に、このサーバーが `false` トランザクションでレプリケーションストリームをフラッシュしないようにするには、これを行うようにしてください。(FLUSH LOGS ステートメントは強制的に新しいバイナリログファイルを作成します。PURGE BINARY LOGS は空のトランザクションをパージしますが、その識別子を保持します。)

このメソッドは、基本的にスナップショットであるサーバーを作成しますが、バイナリログ履歴がレプリケーションストリームのバイナリログ履歴と収束する(つまり、ソースとキャッチアップする)と同時にソースになることができます。この結果は、残りのプロビジョニング方法を使用して得られる結果に実質的に似ています(次のいくつかの段落で説明します)。

`gtid_purged` によるトランザクションの除外。ソースグローバル `gtid_purged` 変数には、ソースバイナリログからパージされたすべてのトランザクションのセットが含まれます。前に説明した方法と同様に(空のトランザクションの注入を参照してください)、スナップショットが作成されたサーバーで(バイナリログを新しいサーバーにコピーする代わりに) `gtid_executed` の値を記録できます。前述の方法とは異なり、空のトランザクションをコミットする(または PURGE BINARY LOGS を発行する)必要はありません。かわりに、バックアップまたはスナップショットが作成されたサーバー上の `gtid_executed` の値に基づいて、レプリカに `gtid_purged` を直接設定できます。

空のトランザクションを使用する方法と同様に、この方法では、機能的にスナップショットであるサーバーが作成されますが、バイナリログ履歴がソースおよびほかのレプリカのものと同様に収束すると、時間内にソースになることができます。

GTID モードの複製の復元。エラーが発生した GTID ベースのレプリケーション設定でレプリカをリストアする場合、イベントに GTID がないため、空のトランザクションをインジェクトしても問題が解決しないことがあります。

`mysqlbinlog` を使用して、次のトランザクション(イベント後の次のログファイルの最初のトランザクション)を検索します。SET @@SESSION.gtid_next が含まれていることを確認して、そのトランザクションの COMMIT までのすべてをコピーします。行ベースレプリケーションを使用していない場合でも、コマンドラインクライアントでバイナリログ行イベントを実行できます。

レプリカを停止し、コピーしたトランザクションを実行します。 `mysqlbinlog` 出力ではデリミタが `/*!*/;` に設定されるため、再度設定します:

```
mysql> DELIMITER ;
```

正しい位置からレプリケーションを自動的に再開します:

```
mysql> SET GTID_NEXT=automatic;
mysql> RESET SLAVE;
mysql> START SLAVE;
Or from MySQL 8.0.22:
mysql> SET GTID_NEXT=automatic;
mysql> RESET REPLICAS;
mysql> START REPLICAS;
```

17.1.3.6 GTID のないソースから GTID のあるレプリカへのレプリケーション

MySQL 8.0.23 から、GTID がまだないレプリケートされたトランザクションに GTID を割り当てるようにレプリケーションチャンネルを設定できます。この機能により、GTID が有効になっておらず GTID ベースのレプリケーションを使用しないソースサーバーから GTID が有効になっているレプリカへのレプリケーションが可能になります。 [セクション 17.1.4 「オンラインサーバーでの GTID モードの変更」](#) で説明されているように、レプリケーションソースサーバーで GTID を有効にできる場合は、代わりにそのアプローチを使用します。この機能は、GTID を有効にできないレプリケーションソースサーバー用に設計されています。

`CHANGE REPLICATION SOURCE TO` ステートメントの `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` オプションを使用して、レプリケーションチャンネルで GTID 割当てを有効にできます。 `LOCAL` は、レプリカ独自の UUID (`server_uuid` 設定) を含む GTID を割り当てます。 `uuid` は、レプリケーションソースサーバーの `server_uuid` 設定など、指定された UUID を含む GTID を割り当てます。非ローカル UUID を使用すると、レプリカで発生したトランザクションと、ソースで発生したトランザクション、およびマルチソースレプリカの場合は異なるソースで発生したトランザクションを区別できます。ソースによって送信されたトランザクションのいずれかに GTID がすでにある場合、その GTID は保持されます。

重要

フェイルオーバーが必要な場合、どのチャンネルでも

`ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` を使用して設定されたレプリカを昇格させてレプリケーションサーバーを置き換えることはできず、レプリカから作成されたバックアップを使用してレプリケーションサーバーをリストアすることはできません。同じ制限が、任意のチャンネルで `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` を使用する他のレプリカの置換またはリストアにも適用されます。

レプリカには `gtid_mode=ON` セットが必要であり、後で変更することはできません。GTID を有効にせずにレプリカサーバーを起動し、レプリケーションチャンネルに `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` を設定した場合、設定は変更されませんが、状況の変更方法を説明する警告メッセージがエラーログに書き込まれます。

マルチソースレプリカの場合、`ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` を使用するチャンネルと使用しないチャンネルを混在させることができます。Group Replication に固有のチャンネルでは `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` を使用できませんが、Group Replication グループメンバーであるサーバーインスタンス上の別のソースの非同期レプリケーションチャンネルでは使用できます。Group Replication グループメンバー上のチャンネルの場合、GTID を作成するための UUID として Group Replication グループ名を指定しないでください。

レプリケーションチャンネルで `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` を使用することは、チャンネルに GTID ベースのレプリケーションを導入することとは異なります。`ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` で設定されたレプリカセットから GTID セット (`gtid_executed`) を別のサーバーに転送したり、別のサーバー `gtid_executed` セットと比較したりしないでください。匿名トランザクションに割り当てられている GTID と、それらに対して選択する UUID は、そのレプリカ独自の使用にのみ意味があります。

`ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` を使用するレプリケーションチャンネルには、GTID ベースのレプリケーションと次の動作の違いがあります:

- GTID は、レプリケートされたトランザクションの適用時に割り当てられます (GTID がすでにある場合を除く)。GTID は通常、トランザクションのコミット時にレプリケーションソースサーバーに割り当てられ、トランザク

ションとともにレプリカに送信されます。マルチスレッドレプリカでは、これは GTID の順序が、`slave-preserve-commit-order=1` が設定されている場合でもトランザクションの順序と一致しないことを意味します。

- `MASTER_AUTO_POSITION` オプションではなく、`CHANGE REPLICATION SOURCE TO` ステートメントの `SOURCE_LOG_FILE` および `SOURCE_LOG_POS` オプションを使用してレプリケーション I/O スレッドを配置します。
- `SET GLOBAL sql_slave_skip_counter` ステートメントは、`CHANGE REPLICATION SOURCE TO` ステートメントではなく、`ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` で設定されたレプリケーションチャンネルでトランザクションをスキップするために使用されます。その手順は、[セクション17.1.7.3「トランザクションのスキップ」](#)を参照してください。
- `START REPLICA` ステートメントの `UNTIL SQL_BEFORE_GTIDS` および `UNTIL SQL_AFTER_GTIDS` オプションは、チャンネルには使用できません。
- MySQL 8.0.18 から非推奨になった関数 `WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS()` は、チャンネルでは使用できません。サーバー全体で動作する交換用の `WAIT_FOR_EXECUTED_GTID_SET()` を使用できます。

「パフォーマンススキーマ」テーブル `replication_applier_configuration` は、GTID がレプリケーションチャンネル上の匿名トランザクションに割り当てられているかどうか、UUID とは何か、およびそれがレプリカサーバー (LOCAL) とユーザー指定 UUID (MANUAL) のどちらであるかを示します。この情報は、アプライヤメタデータリポジトリにも記録されます。`RESET SLAVE ALL` ステートメントは `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` 設定をリセットしますが、`RESET SLAVE` ステートメントはリセットしません。

17.1.3.7 GTID ベースレプリケーションの制約

GTID ベースレプリケーションはトランザクションに依存しているため、そうでなければ MySQL で使用できるいくつかの機能が、それを使用するときにサポートされません。このセクションでは、GTID ベースレプリケーションの制約と制限についての情報を提供します。

非トランザクションストレージエンジンに関係する更新。 GTID を使用する場合、`MyISAM` などの非トランザクションストレージエンジンを使用するテーブルへの更新は、`InnoDB` などのトランザクションストレージエンジンを使用するテーブルへの更新と同じステートメントまたはトランザクションで実行できません。

この制約は、非トランザクションストレージエンジンを使用するテーブルへの更新とトランザクションストレージエンジンを使用するテーブルへの更新が、同じトランザクション内に混在していると、複数の GTID が同じトランザクションに割り当てられる可能性があるためです。

このような問題は、一方のストレージエンジンがトランザクショナルで、もう一方がトランザクショナルでない場合に、ソースとレプリカがそれぞれのバージョンの同じテーブルに対して異なるストレージエンジンを使用している場合にも発生することがあります。また、非トランザクションテーブルで動作するように定義されているトリガーが、これらの問題の原因になる可能性があることにも注意してください。

今挙げたいずれの場合も、トランザクションと GTID との間の 1 対 1 対応が壊れていて、GTID ベースレプリケーションは正しく機能できません。

`CREATE TABLE ... SELECT` ステートメント。 MySQL 8.0.21 より前では、GTID ベースのレプリケーションを使用する場合、`CREATE TABLE ... SELECT` ステートメントは許可されません。`binlog_format` が `STATEMENT` に設定されている場合、`CREATE TABLE ... SELECT` ステートメントはバイナリログに 1 つの GTID を持つ 1 つのトランザクションとして記録されますが、`ROW` 形式が使用されている場合、ステートメントは 2 つの GTID を持つ 2 つのトランザクションとして記録されます。ソースが `STATEMENT` 形式を使用し、レプリカが `ROW` 形式を使用した場合、レプリカはトランザクションを正しく処理できないため、GTID で `CREATE TABLE ... SELECT` ステートメントを使用してこのシナリオを回避することはできません。この制限は、アトミック DDL をサポートするストレージエンジン上の MySQL 8.0.21 で解除されています。この場合、`CREATE TABLE ... SELECT` はバイナリログに 1 つのトランザクションとして記録されます。詳細は、[セクション13.1.1「アトミックデータ定義ステートメントのサポート」](#)を参照してください。

一時テーブル。 `binlog_format` が `STATEMENT` に設定されている場合、GTID がサーバーで使用されているとき (`enforce_gtid_consistency` システム変数が `ON` に設定されているとき) は、トランザクション、プロシージャ、関数およびトリガー内で `CREATE TEMPORARY TABLE` および `DROP TEMPORARY TABLE` ステートメントを使用できません。GTID が使用されている場合、`autocommit=1` が設定されていれば、これらのコンテキストの外部で使用できます。MySQL 8.0.13 から、`binlog_format` が `ROW` または `MIXED` に設定されている場合、GTID が使用されているとき

に、[CREATE TEMPORARY TABLE](#) および [DROP TEMPORARY TABLE](#) ステートメントをトランザクション、プロシージャ、関数またはトリガー内で使用できます。ステートメントはバイナリログに書き込まれないため、複製に複製されません。行ベースのレプリケーションを使用することは、一時テーブルをレプリケートする必要なく、レプリカの同期が維持されることを意味します。トランザクションからこれらのステートメントを削除した結果、空のトランザクションが発生した場合、そのトランザクションはバイナリログに書き込まれません。

サポートされないステートメントの実行の回避。 GTID ベースのレプリケーションが失敗する原因となるステートメントの実行を防ぐには、GTID を有効にするときに `--enforce-gtid-consistency` オプションを使用してすべてのサーバーを起動する必要があります。これにより、このセクションですでに説明したタイプのステートメントはエラーで失敗します。

`--enforce-gtid-consistency` が有効になるのは、ステートメントに対してバイナリロギングが行われる場合だけです。バイナリロギングがサーバーで無効になっている場合、またはステートメントがフィルタによって削除されたためにバイナリログに書き込まれない場合、GTID の整合性は、ログに記録されていないステートメントに対してチェックまたは適用されません。

GTID を有効にするときに必要なほかの起動オプションについては、[セクション 17.1.3.4 「GTID を使用したレプリケーションのセットアップ」](#) を参照してください。

トランザクションのスキップ。 GTID ベースのレプリケーションを使用している場合、`sql_slave_skip_counter` は使用できません。トランザクションをスキップする必要がある場合は、代わりにソース `gtid_executed` 変数の値を使用します。[CHANGE REPLICATION SOURCE TO](#) ステートメントの `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` オプションを使用してレプリケーションチャンネルで GTID 割当てを有効にした場合、`sql_slave_skip_counter` を使用できます。詳細は、[セクション 17.1.7.3 「トランザクションのスキップ」](#) を参照してください。

サーバーの無視。 GTID を使用している場合、[CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO](#) ステートメントの「The IGNORE_SERVER_IDS」オプションは非推奨です。すでに適用されているトランザクションは自動的に無視されるためです。GTID ベースのレプリケーションを開始する前に、関係するサーバーで以前に設定されたすべての無視されたサーバー ID リストを確認してクリアします。個々のチャンネルに対して発行できる [SHOW REPLICA | SLAVE STATUS](#) ステートメントには、無視されたサーバー ID のリストが表示されます (存在する場合)。リストがない場合、`Replicate_Ignore_Server_Ids` フィールドは空白です。

GTID モードと `mysql_upgrade`。 MySQL 8.0.16 より前では、グローバルトランザクション識別子 (GTID) を有効にして (`gtid_mode=ON`) サーバーを実行している場合、`mysql_upgrade` (`--write-binlog` オプション) によるバイナリロギングを有効にしないでください。MySQL 8.0.16 の時点では、サーバーは MySQL アップグレード手順全体を実行しますが、アップグレード中はバイナリロギングを無効にするため、問題はありません。

17.1.3.8 GTID を操作するストアドファンクションの例

MySQL には、GTID ベースのレプリケーションで使用する組込み (ネイティブ) 関数がいくつか含まれています。これらの関数は次のとおりです:

`GTID_SUBSET(set1,set2)` グローバルトランザクション識別子 `set1` および `set2` の 2 つのセットが指定されている場合、`set1` 内のすべての GTID も `set2` 内にあると true を返します。それ以外の場合は、false を返します。

`GTID_SUBTRACT(set1,set2)` グローバルトランザクション識別子 `set1` および `set2` の 2 つのセットがある場合、`set2` がない GTID のみを `set1` から返します。

`WAIT_FOR_EXECUTED_GTID_SET(gtids, timeout)` `gtids` が、グローバルトランザクション識別子が `gtid_set` に含まれているすべてのトランザクションを適用するまで待機します。オプションのタイムアウトは、指定された秒数が経過すると、関数の待機を停止します。

これらの関数の詳細は、[セクション 12.19 「グローバルトランザクション識別子 \(GTID\) で使用される機能」](#) を参照してください。

GTID を操作する独自のストアドファンクションを定義できます。ストアドファンクションの定義の詳細は、[第 25 章 「ストアドオブジェクト」](#) を参照してください。次の例は、組込みの `GTID_SUBSET()` および `GTID_SUBTRACT()` 関数に基づいて作成できる便利なストアドファンクションを示しています。

これらのストアドファンクションでは、次のように `delimiter` コマンドを使用して MySQL ステートメントデリミタを縦棒に変更しています:

```
mysql> delimiter |
```

これらの関数はすべて GTID セットの文字列表現を引数として取るため、GTID セットと一緒に使用する場合は常に引用符で囲む必要があります。

この関数は、2 つの GTID セットが同じセットである場合、同じ方法で書式設定されていなくても、ゼロ以外 (true) を返します。

```
CREATE FUNCTION GTID_IS_EQUAL(gtid_set_1 LONGTEXT, gtid_set_2 LONGTEXT)
RETURNS INT
RETURN GTID_SUBSET(gtid_set_1, gtid_set_2) AND GTID_SUBSET(gtid_set_2, gtid_set_1)
```

この関数は、2 つの GTID セットが非結合の場合、ゼロ以外 (true) を返します。

```
CREATE FUNCTION GTID_IS_DISJOINT(gtid_set_1 LONGTEXT, gtid_set_2 LONGTEXT)
RETURNS INT
RETURN GTID_SUBSET(gtid_set_1, GTID_SUBTRACT(gtid_set_1, gtid_set_2))
```

この関数は、2 つの GTID セットが非結合で、**sum** が 2 つのセットの和集合である場合、ゼロ以外 (true) を返します。

```
CREATE FUNCTION GTID_IS_DISJOINT_UNION(gtid_set_1 LONGTEXT, gtid_set_2 LONGTEXT, sum LONGTEXT)
RETURNS INT
RETURN GTID_IS_EQUAL(GTID_SUBTRACT(sum, gtid_set_1), gtid_set_2) AND
GTID_IS_EQUAL(GTID_SUBTRACT(sum, gtid_set_2), gtid_set_1)
```

この関数は、GTID セットの正規化された形式をすべて大文字で返します。空白はなく、重複もありません。UUID はアルファベット順に配置され、間隔は数値順に配置されます。

```
CREATE FUNCTION GTID_NORMALIZE(g LONGTEXT)
RETURNS LONGTEXT
RETURN GTID_SUBTRACT(g, "")
```

この関数は、2 つの GTID セットの和集合を返します。

```
CREATE FUNCTION GTID_UNION(gtid_set_1 LONGTEXT, gtid_set_2 LONGTEXT)
RETURNS LONGTEXT
RETURN GTID_NORMALIZE(CONCAT(gtid_set_1, ',', gtid_set_2))
```

この関数は、2 つの GTID セットの交差を返します。

```
CREATE FUNCTION GTID_INTERSECTION(gtid_set_1 LONGTEXT, gtid_set_2 LONGTEXT)
RETURNS LONGTEXT
RETURN GTID_SUBTRACT(gtid_set_1, GTID_SUBTRACT(gtid_set_1, gtid_set_2))
```

この関数は、2 つの GTID セット (**gtid_set_1** には存在するが **gtid_set_2** には存在しない GTID) と、**gtid_set_2** には存在するが **gtid_set_1** には存在しない GTID の対称差異を戻します。

```
CREATE FUNCTION GTID_SYMMETRIC_DIFFERENCE(gtid_set_1 LONGTEXT, gtid_set_2 LONGTEXT)
RETURNS LONGTEXT
RETURN GTID_SUBTRACT(CONCAT(gtid_set_1, ',', gtid_set_2), GTID_INTERSECTION(gtid_set_1, gtid_set_2))
```

この関数は GTID セットから、指定された起点からすべての GTID を削除し、残りの GTID があればそれを返します。UUID は、トランザクションが発生したサーバーで使用される識別子で、通常は **server_uuid** 値です。

```
CREATE FUNCTION GTID_SUBTRACT_UUID(gtid_set LONGTEXT, uuid TEXT)
RETURNS LONGTEXT
RETURN GTID_SUBTRACT(gtid_set, CONCAT(UUID, ':1-', (1 << 63) - 2))
```

この関数は、指定された識別子 (UUID) を持つサーバーから発生した GTID セットの GTID のみを返すように、以前に一覧表示された関数を逆にします。

```
CREATE FUNCTION GTID_INTERSECTION_WITH_UUID(gtid_set LONGTEXT, uuid TEXT)
RETURNS LONGTEXT
RETURN GTID_SUBTRACT(gtid_set, GTID_SUBTRACT_UUID(gtid_set, uuid))
```

例 17.1 レプリカが最新であることの確認

組み込み関数 **GTID_SUBSET** および **GTID_SUBTRACT** を使用すると、ソースが適用したすべてのトランザクションにレプリカが適用されているかどうかを確認できます。

`GTID_SUBSET` でこのチェックを実行するには、レプリカで次のステートメントを実行します:

```
SELECT GTID_SUBSET(source_gtid_executed, replica_gtid_executed)
```

これが 0 (false) を返す場合、`source_gtid_executed` の GTID の一部が `replica_gtid_executed` に存在しないため、レプリカが適用されていないトランザクションがソースによって適用されているため、レプリカは最新ではありません。

`GTID_SUBTRACT` でチェックを実行するには、レプリカで次のステートメントを実行します:

```
SELECT GTID_SUBTRACT(source_gtid_executed, replica_gtid_executed)
```

このステートメントは、`source_gtid_executed` にはあるが `replica_gtid_executed` にはない GTID を返します。GTID が返された場合、ソースはレプリカが適用していない一部のトランザクションを適用しているため、レプリカは最新ではありません。

例 17.2 バックアップおよびリストアのシナリオ

ストアドファンクション `GTID_IS_EQUAL`、`GTID_IS_DISJOINT` および `GTID_IS_DISJOINT_UNION` を使用して、複数のデータベースおよびサーバーに関連するバックアップおよびリストア操作を検証できます。この例のシナリオでは、`server1` にデータベース `db1` が含まれ、`server2` にデータベース `db2` が含まれています。目的は、データベース `db2` を `server1` にコピーし、`server1` での結果を 2 つのデータベースの和集合にすることです。使用する手順は、`mysqldump` または `mysqlpump` を使用して `server2` をバックアップし、`server1` でこのバックアップをリストアすることです。

バックアッププログラムオプション `--set-gtid-purged` が `ON` に設定されているか、`AUTO` のデフォルトに設定されている場合、プログラム出力には `SET @@GLOBAL.gtid_purged` ステートメントが含まれ、`gtid_executed` セットを `server2` から `server1` の `gtid_purged` に追加します。`gtid_purged` セットには、サーバー上でコミットされたが、サーバー上のバイナリログファイルに存在しないすべてのトランザクションの GTID が含まれています。データベース `db2` を `server1` にコピーする場合、`server1` のバイナリログファイルにない `server2` でコミットされたトランザクションの GTID を `server1` の `gtid_purged` セットに追加して、セットを完成させる必要があります。

ストアドファンクションを使用すると、このシナリオの次のステップを実行できます:

- `GTID_IS_EQUAL` を使用して、バックアップ操作によって `SET @@GLOBAL.gtid_purged` ステートメントの正しい GTID セットが計算されたことを確認します。`server2` では、`mysqlpump` または `mysqldump` の出力からそのステートメントを抽出し、GTID セットを `$gtid_purged_set` などのローカル変数に格納します。次に、次のステートメントを実行します:

```
server2> SELECT GTID_IS_EQUAL($gtid_purged_set, @@GLOBAL.gtid_executed);
```

結果が 1 の場合、2 つの GTID セットは等しく、セットは正しく計算されています。

- `GTID_IS_DISJOINT` を使用して、`mysqlpump` または `mysqldump` 出力の GTID セットが `server1` の `gtid_executed` セットと重複していないことを確認します。両方のサーバーに同一の GTID が存在すると、データベース `db2` を `server1` にコピーするときにエラーが発生します。確認するには、`server1` で、前述のように出力から `gtid_purged` セットを抽出してローカル変数に格納し、次のステートメントを実行します:

```
server1> SELECT GTID_IS_DISJOINT($gtid_purged_set, @@GLOBAL.gtid_executed);
```

結果が 1 の場合、2 つの GTID セット間に重複がないため、重複する GTID は存在しません。

- `GTID_IS_DISJOINT_UNION` を使用して、リストア操作が `server1` で正しい GTID 状態になったことを確認します。バックアップをリストアする前に、`server1` で次のステートメントを実行して既存の `gtid_executed` セットを取得します:

```
server1> SELECT @@GLOBAL.gtid_executed;
```

結果をローカル変数 `$original_gtid_executed` に格納します。また、前述のように、`gtid_purged` セットをローカル変数に格納します。`server2` からのバックアップが `server1` にリストアされたら、次のステートメントを実行して GTID 状態を確認します:

```
server1> SELECT GTID_IS_DISJOINT_UNION($original_gtid_executed,
    $gtid_purged_set,
    @@GLOBAL.gtid_executed);
```

結果が 1 の場合、ストアドファンクションは、`server1` (`$original_gtid_executed`) からの元の `gtid_executed` セットと `server2` (`$gtid_purged_set`) から追加された `gtid_purged` セットが重複していないことを検証し、`server1` で更新

された `gtid_executed` セットが、`server1` からの以前の `gtid_executed` セットと `server2` からの `gtid_purged` セットで構成されるようになりました。これが目的の結果です。 `server1` でこれ以上のトランザクションが実行される前に、このチェックが実行されていることを確認してください。実行されていない場合、`gtid_executed` セットの新しいトランザクションが失敗します。

例 17.3 手動フェイルオーバー用の最新レプリカの選択

ストアドファンクション `GTID_UNION` を使用すると、ソースサーバーが予期せず停止した後に手動フェイルオーバー操作を実行するために、レプリカのセットから最新のレプリカを識別できます。一部のレプリカでレプリケーションラグが発生している場合、このストアドファンクションを使用すると、すべてのレプリカが既存のリレーログを適用するのを待機せずに最新のレプリカを計算できるため、フェイルオーバー時間を最小限に抑えることができます。この関数は、「パフォーマンススキーマ」テーブル `replication_connection_status` に記録されているレプリカが受信したトランザクションのセットを使用して、各レプリカの `gtid_executed` セットの和集合を戻すことができます。これらの結果を比較して、すべてのトランザクションがまだコミットされていない場合でも、最新のトランザクションのレプリカレコードを検索できます。

各レプリカで、次のステートメントを発行してトランザクションの完全なレコードを計算します:

```
SELECT GTID_UNION(RECEIVED_TRANSACTION_SET, @@GLOBAL.gtid_executed)
FROM performance_schema.replication_connection_status
WHERE channel_name = 'name';
```

その後、各レプリカの結果を比較してトランザクションの最新レコードを確認し、このレプリカを新しいソースとして使用できます。

例 17.4 レプリカ上の無関係なトランザクションのチェック

ストアドファンクション `GTID_SUBTRACT_UUID` を使用すると、指定されたソースから発生しなかったトランザクションをレプリカが受信したかどうかを確認できます。その場合は、レプリケーションの設定、またはプロキシ、ルーターまたはロードバランサに問題がある可能性があります。この機能は、GTID セットから指定された発信元サーバーからすべての GTID を削除し、残りの GTID があればそれを返すことによって機能します。

単一のソースからレプリケートするレプリカの場合、次のステートメントを発行し、元のソースの識別子 (通常は `server_uuid` 値) を指定します:

```
SELECT GTID_SUBTRACT_UUID(@@GLOBAL.gtid_executed, server_uuid_of_source);
```

結果が空でない場合、返されるトランザクションは、指定されたソースから発生しなかった余分なトランザクションです。

マルチソースレプリケーショントポロジのレプリカの場合は、次の例のように機能を繰り返します:

```
SELECT GTID_SUBTRACT_UUID(GTID_SUBTRACT_UUID(@@GLOBAL.gtid_executed,
server_uuid_of_source_1),
server_uuid_of_source_2);
```

結果が空でない場合、返されるトランザクションは、指定されたどのソースからも発生しなかった余分なトランザクションです。

例 17.5 レプリケーショントポロジ内のサーバーが読み取り専用であることの確認

ストアドファンクション `GTID_INTERSECTION_WITH_UUID` を使用すると、サーバーが GTID を発生させておらず、読み取り専用状態であることを検証できます。この関数は、指定された識別子を持つサーバーから発生した GTID セットから GTID のみを返します。サーバー `gtid_executed` セット内のいずれかのトランザクションにサーバー独自の識別子がある場合、サーバー自身がそれらのトランザクションを開始しています。サーバーで次のステートメントを発行して確認できます:

```
SELECT GTID_INTERSECTION_WITH_UUID(@@GLOBAL.gtid_executed, my_server_uuid);
```

例 17.6 マルチソースレプリケーション設定での追加レプリカの検証

ストアドファンクション `GTID_INTERSECTION_WITH_UUID` を使用すると、マルチソースレプリケーション設定にアタッチされたレプリカが、特定のソースから発生したすべてのトランザクションを適用したかどうかを確認できます。このシナリオでは、`source1` と `source2` の両方がソースとレプリカであり、相互にレプリケートされます。`source2` には、独自のレプリカもあります。また、`source2` が `log_slave_updates=ON` で構成されている場合、レプリカはソース `source1` からトランザクションを受信して適用しますが、`source2` が `log_slave_updates=OFF` を

使用している場合は行いません。いずれの場合も、現在、レプリカが `source2` で最新であるかどうかのみを確認します。この状況では、ストアドファンクション `GTID_INTERSECTION_WITH_UUID` を使用して、`source2` が発生させたトランザクションを識別し、`source2` が `source1` からレプリケートしたトランザクションを破棄できます。その後、組み込み関数 `GTID_SUBSET` を使用して、結果をレプリカ上の `gtid_executed` セットと比較できます。レプリカが `source2` で最新の場合、レプリカに設定されている `gtid_executed` には交差セット (`source2` から発生したトランザクション) 内のすべてのトランザクションが含まれます。

このチェックを実行するには、次のように、`source2` の `gtid_executed` セット、`source2` のサーバー UUID およびレプリカ `gtid_executed` セットをクライアント側の変数に格納します:

```
$source2_gtid_executed :=  
source2> SELECT @@GLOBAL.gtid_executed;  
$source2_server_uuid :=  
source2> SELECT @@GLOBAL.server_uuid;  
$replica_gtid_executed :=  
replica> SELECT @@GLOBAL.gtid_executed;
```

次に、次のように、これらの変数を入力として `GTID_INTERSECTION_WITH_UUID` および `GTID_SUBSET` を使用します:

```
SELECT GTID_SUBSET(GTID_INTERSECTION_WITH_UUID($source2_gtid_executed,  
$source2_server_uuid),  
$replica_gtid_executed);
```

`source2` (`$source2_server_uuid`) のサーバー識別子は、`source1` で発生した GTID を除外して、`source2` の `gtid_executed` セットからの GTID のみを識別して返すために `GTID_INTERSECTION_WITH_UUID` とともに使用されます。結果の GTID セットは、`GTID_SUBSET` を使用して、レプリカで実行されたすべての GTID のセットと比較されます。このステートメントがゼロ以外 (`true`) を返す場合、`source2` から識別された GTID (最初のセット入力) もすべてレプリカ `gtid_executed` セット (2 番目のセット入力) に含まれます。つまり、レプリカは `source2` から発生したすべてのトランザクションをレプリケートしています。

17.1.4 オンラインサーバーでの GTID モードの変更

このセクションでは、サーバーをオフラインにせず GTID モードとの間でレプリケーションのモードを変更する方法について説明します。

17.1.4.1 レプリケーションモードの概念

オンラインサーバーのレプリケーションモードを安全に構成できるようにするには、レプリケーションのいくつかの重要な概念を理解することが重要です。このセクションでは、オンラインサーバーのレプリケーションモードを変更する前に、これらの概念について説明し、不可欠な読み取り値を示します。

MySQL で使用可能なレプリケーションのモードは、ログに記録されるトランザクションを識別するための様々な手法に依存します。レプリケーションで使用されるトランザクションのタイプは次のとおりです:

- GTID トランザクションは、`UUID:NUMBER` 形式のグローバルトランザクション識別子 (GTID) によって識別されます。ログ内の GTID トランザクションはすべて、常に `Gtid_log_event` で始まります。GTID トランザクションは GTID を使用するが、ファイル名とポジションを使用して対処できます。
- 匿名トランザクションに GTID が割り当てられておらず、MySQL はログ内のすべての匿名トランザクションの前に `Anonymous_gtid_log_event` があることを確認します。以前のバージョンでは、匿名トランザクションの前に特定のイベントはありませんでした。匿名トランザクションは、ファイル名と位置を使用してのみ処理できます。

GTID を使用する場合は、GTID 自動配置および自動フェイルオーバーを利用したり、`WAIT_FOR_EXECUTED_GTID_SET()`、`session_track_gtid` を使用したり、「パフォーマンススキーマ」テーブルを使用してレプリケートされたトランザクションをモニターしたりできます。

以前のバージョンの MySQL を実行しているソースから受信されたリレーログ内のトランザクションの前には、特定のイベントがまったく表示されない場合がありますが、リプレイされてレプリカバイナリログに記録されたあとは、それらのトランザクションの前に `Anonymous_gtid_log_event` が付きます。

レプリケーションモードをオンラインで構成する機能は、`gtid_mode` 変数と `enforce_gtid_consistency` 変数の両方が動的になり、グローバルシステム変数の設定に十分な権限を持つアカウントによって最上位レベルのステートメントから設定できることを意味します。セクション 5.1.9.1 「システム変数権限」を参照してください。MySQL 5.6 以前

では、これらの変数は両方とも、サーバーの起動時に適切なオプションを使用してのみ構成できました。つまり、レプリケーションモードへの変更にはサーバーの再起動が必要でした。すべてのバージョンで、`gtid_mode` を `ON` または `OFF` に設定でき、GTID がトランザクションの識別に使用されたかどうかに対応します。`gtid_mode=ON` で匿名トランザクションをレプリケートできない場合、および `gtid_mode=OFF` でレプリケートできるのは匿名トランザクションのみです。`gtid_mode=OFF_PERMISSIVE` の場合、new トランザクションは匿名ですが、レプリケートされたトランザクションを GTID または匿名トランザクションにすることを許可します。`gtid_mode=ON_PERMISSIVE` の場合、new トランザクションは GTID を使用しますが、レプリケートされたトランザクションを GTID または匿名トランザクションにすることを許可します。つまり、匿名トランザクションと GTID トランザクションの両方を使用するサーバーを持つレプリケーショントポロジを使用できます。たとえば、`gtid_mode=ON` を含むソースは、`gtid_mode=ON_PERMISSIVE` を使用してレプリカにレプリケートできます。`gtid_mode` の有効な値は次のとおりで、順序は次のとおりです:

- `OFF`
- `OFF_PERMISSIVE`
- `ON_PERMISSIVE`
- `ON`

`gtid_mode` の状態は、前述の順序に基づいて一度に 1 つのステップでのみ変更できることに注意してください。たとえば、`gtid_mode` が現在 `OFF_PERMISSIVE` に設定されている場合、`OFF` または `ON_PERMISSIVE` に変更できますが、`ON` には変更できません。これは、匿名トランザクションから GTID トランザクションへの変更プロセスがサーバーによって正しく処理されるようにするためです。`gtid_mode=ON` と `gtid_mode=OFF` を切り替えると、GTID 状態 (つまり、`gtid_executed` の値) は永続的になります。これにより、`gtid_mode` のタイプ間の変更に関係なく、サーバーによって適用された GTID セットが常に保持されます。

GTID に関連するフィールドには、現在選択されている `gtid_mode` に関係なく、正しい情報が表示されます。つまり、GTID セットを表示するフィールド (`replication_connection_status` 「パフォーマンススキーマ」テーブルの `gtid_executed`、`gtid_purged`、`RECEIVED_TRANSACTION_SET` など) および `SHOW REPLICA | SLAVE STATUS` の GTID 関連の結果は、GTID が存在しない場合に空の文字列を返すようになりました。GTID トランザクションが使用されていないときに、パフォーマンススキーマ `replication_applier_status_by_worker` テーブル内の `CURRENT_TRANSACTION` などの単一 GTID を表示するフィールドに `ANONYMOUS` が表示されるようになりました。

`gtid_mode=ON` を使用するソースからのレプリケーションでは、`CHANGE REPLICATION SOURCE TO` ステートメント (MySQL 8.0.23 から) の `SOURCE_AUTO_POSITION`、または `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 より前) の `MASTER_AUTO_POSITION` を使用して構成された GTID 自動配置を使用できます。使用されるレプリケーショントポロジは、自動配置を有効にできるかどうかに影響します。この機能は GTID に依存し、匿名トランザクションと互換性がないためです。自動配置を有効にする前に、トポロジに匿名トランザクションが残っていないことを確認することを強くお勧めします。[セクション 17.1.4.2 「GTID トランザクションのオンラインでの有効化」](#) を参照してください。

次のテーブルに、`gtid_mode` とソースおよびレプリカでの自動配置の有効な組合せを示します。ここで、ソース `gtid_mode` は水平方向に、レプリカ `gtid_mode` は垂直方向に表示されます。各エントリの意味は次のとおりです:

- `Y`: ソースとレプリカの `gtid_mode` に互換性があります
- `N`: ソースとレプリカの `gtid_mode` に互換性ありません
- `*`: この組合せでは自動配置を使用できます

表 17.1 ソースとレプリカ `gtid_mode` の有効な組合せ

<code>gtid_mode</code>	ソース <code>OFF</code>	ソース <code>OFF_PERMISSIVE</code>	ソース <code>ON_PERMISSIVE</code>	ソース <code>ON</code>
レプリカ <code>OFF</code>	Y	Y	N	N
レプリカ <code>OFF_PERMISSIVE</code>	Y	Y	Y	Y*
レプリカ <code>ON_PERMISSIVE</code>	Y	Y	Y	Y*

gtid_mode	ソース OFF	ソース OFF_PERMISSIVE	ソース ON_PERMISSIVE	ソース ON
レプリカ ON	N	N	Y	Y*

現在選択されている `gtid_mode` は、`gtid_next` 変数にも影響します。次のテーブルに、`gtid_mode` および `gtid_next` の様々な値に対するサーバーの動作を示します。各エントリの意味は次のとおりです：

- **ANONYMOUS**: 匿名トランザクションを生成します。
- **Error**: エラーを生成し、`SET GTID_NEXT` の実行に失敗します。
- **UUID:NUMBER**: 指定された UUID:NUMBER の GTID を生成します。
- **New GTID**: 自動生成された番号を使用して GTID を生成します。

表 17.2 gtid_mode と gtid_next の有効な組合せ

	gtid_next AUTOMATIC バイナリログオン	gtid_next AUTOMATIC バイナリログオフ	gtid_next ANONYMOUS	gtid_next UUID:NUMBER
gtid_mode OFF	ANONYMOUS	ANONYMOUS	ANONYMOUS	エラー
gtid_mode OFF_PERMISSIVE	ANONYMOUS	ANONYMOUS	ANONYMOUS	UUID:NUMBER
gtid_mode ON_PERMISSIVE	新規 GTID	ANONYMOUS	ANONYMOUS	UUID:NUMBER
gtid_mode ON	新規 GTID	ANONYMOUS	エラー	UUID:NUMBER

バイナリログがオフで、`gtid_next` が `AUTOMATIC` に設定されている場合、GTID は生成されません。これは、以前のバージョンの動作と一貫性があります。

17.1.4.2 GTID トランザクションのオンラインでの有効化

このセクションでは、すでにオンラインで匿名トランザクションを使用しているサーバー上で GTID トランザクションを有効にし、オプションで自動配置を有効にする方法について説明します。この手順では、サーバーをオフラインにする必要はなく、本番での使用に適しています。ただし、GTID トランザクションを有効にするときにサーバーをオフラインにできる可能性がある場合は、プロセスが容易です。

MySQL 8.0.23 から、GTID がまだないレプリケートされたトランザクションに GTID を割り当てるようにレプリケーションチャンネルを設定できます。この機能により、GTID ベースのレプリケーションを使用しないソースサーバーから使用するレプリカへのレプリケーションが可能になります。この手順で説明されているように、レプリケーションソースサーバーで GTID を有効にできる場合は、代わりにこの方法を使用します。GTID の割当ては、GTID を有効にできないレプリケーションソースサーバー用に設計されています。このオプションの詳細は、[セクション 17.1.3.6 「GTID のないソースから GTID のあるレプリカへのレプリケーション」](#) を参照してください。

起動する前に、サーバーが次の事前条件を満たしていることを確認します：

- トポロジ内の All サーバーは、MySQL 5.7.6 以上を使用する必要があります。トポロジ内の all サーバーがこのバージョンを使用していないかぎり、GTID トランザクションを単一のサーバーでオンラインで有効にすることはできません。
- すべてのサーバーで、`gtid_mode` がデフォルト値の `OFF` に設定されています。

次の手順は、GTID を無効にするオンライン手順である [セクション 17.1.4.3 「GTID トランザクションのオンラインでの無効化」](#) の対応する手順にジャンプすることで、いつでも一時停止してあとで再開できます。これにより、プロセスの途中で発生する関連のない問題は通常どおりに処理でき、その後、プロセスは停止した場所で続行されるため、プロセスはフォルトトレラントになります。

注記

次のステップに進む前に、すべてのステップを完了することが重要です。

GTID トランザクションを使用可能にする手順は、次のとおりです:

1. 各サーバーで、次のコマンドを実行します:

```
SET @@GLOBAL.ENFORCE_GTID_CONSISTENCY = WARN;
```

通常のワークロードでサーバーをしばらく実行し、ログを監視します。このステップでログに警告が発生した場合は、GTID 互換機能のみを使用し、警告を生成しないようにアプリケーションを調整します。

重要

これは最初の重要なステップです。次のステップに進む前に、エラーログに警告が生成されていないことを確認する必要があります。

2. 各サーバーで、次のコマンドを実行します:

```
SET @@GLOBAL.ENFORCE_GTID_CONSISTENCY = ON;
```

3. 各サーバーで、次のコマンドを実行します:

```
SET @@GLOBAL.GTID_MODE = OFF_PERMISSIVE;
```

どのサーバーがこのステートメントを最初に実行するかは関係ありませんが、すべてのサーバーが次のステップを開始する前にこのステップを完了することが重要です。

4. 各サーバーで、次のコマンドを実行します:

```
SET @@GLOBAL.GTID_MODE = ON_PERMISSIVE;
```

どのサーバーがこのステートメントを最初に実行するかは関係ありません。

5. 各サーバーで、ステータス変数 `ONGOING_ANONYMOUS_TRANSACTION_COUNT` がゼロになるまで待機します。これは、次を使用してチェックできます:

```
SHOW STATUS LIKE 'ONGOING_ANONYMOUS_TRANSACTION_COUNT';
```

注記

レプリカでは、理論的にはゼロを表示してから再度ゼロ以外を表示することが可能です。これは問題ではなく、一度はゼロと表示されます。

6. ステップ 5 までに生成されたすべてのトランザクションがすべてのサーバーにレプリケートされるまで待機します。これは、更新を停止せずに実行できます: 唯一重要なのは、すべての匿名トランザクションがレプリケートされることです。

すべての匿名トランザクションがすべてのサーバーにレプリケートされたことを確認する方法については、[セクション 17.1.4.4 「匿名トランザクションのレプリケーションの検証」](#) を参照してください。

7. ポイントインタイムバックアップやリストアなど、レプリケーション以外にバイナリログを使用する場合は、GTID のないトランザクションを含む古いバイナリログが不要になるまで待機します。

たとえば、ステップ 6 の完了後、バックアップを作成しているサーバーで `FLUSH LOGS` を実行できます。次に、明示的にバックアップを取るか、設定した定期バックアップルーチンの次の反復を待機します。

理想的には、ステップ 6 の完了時に存在していたすべてのバイナリログがサーバーによってパーージされるまで待機します。また、ステップ 6 より前に作成したバックアップが期限切れになるまで待機します。

重要

これは 2 番目の重要な点です。GTID のない匿名トランザクションを含むバイナリログは、次の手順のあとに使用できないことを理解することが不可欠です。このステップの後、GTID のないトランザクションがトポロジ内のどこにも存在しないことを確認する必要があります。

8. 各サーバーで、次のコマンドを実行します:

```
SET @@GLOBAL.GTID_MODE = ON;
```

9. 各サーバーで、`gtid_mode=ON` および `enforce_gtid_consistency=ON` を `my.cnf` に追加します。

すべてのトランザクションに GTID があることが保証されるようになりました (ステップ 5 以前で生成され、すでに処理されたトランザクションを除く)。GTID プロトコルの使用を開始して後で自動フェイルオーバーを実行できるようにするには、各レプリカで次を実行します。オプションで、マルチソースレプリケーションを使用する場合は、チャンネルごとにこれを行い、`FOR CHANNEL channel` 句を含めます:

```
STOP SLAVE [FOR CHANNEL 'channel'];
CHANGE MASTER TO MASTER_AUTO_POSITION = 1 [FOR CHANNEL 'channel'];
START SLAVE [FOR CHANNEL 'channel'];

Or from MySQL 8.0.22 / 8.0.23:
STOP REPLICA [FOR CHANNEL 'channel'];
CHANGE REPLICATION SOURCE TO SOURCE_AUTO_POSITION = 1 [FOR CHANNEL 'channel'];
START REPLICA [FOR CHANNEL 'channel'];
```

17.1.4.3 GTID トランザクションのオンラインでの無効化

このセクションでは、すでにオンラインになっているサーバーで GTID トランザクションを無効にする方法について説明します。この手順では、サーバーをオフラインにする必要はなく、本番での使用に適しています。ただし、GTID モードを無効にするときにサーバーをオフラインにする可能性がある場合は、そのプロセスが容易になります。

このプロセスは、サーバーがオンラインのときに GTID トランザクションを有効にする場合と似ていますが、手順は逆になります。異なるのは、ログに記録されたトランザクションがレプリケートされるまで待機するポイントのみです。

起動する前に、サーバーが次の事前条件を満たしていることを確認します:

- トポロジ内の All サーバーは、MySQL 5.7.6 以上を使用する必要があります。トポロジ内の all サーバーがこのバージョンを使用していないかぎり、GTID トランザクションを単一のサーバーでオンラインで無効にすることはできません。
- すべてのサーバーで、`gtid_mode` が `ON` に設定されています。
- `--replicate-same-server-id` オプションがどのサーバーにも設定されていません。このオプションが `--log-slave-updates` オプション (デフォルト) とともに設定され、バイナリロギングが有効 (デフォルト) である場合は、GTID トランザクションを無効にできません。GTID を使用しない場合、このオプションの組み合わせによって循環レプリケーションで無限ループが発生します。

1. 各レプリカで次を実行し、マルチソースレプリケーションを使用している場合は、チャンネルごとに実行し、`FOR CHANNEL` チャンネル句を含めます:

```
STOP SLAVE [FOR CHANNEL 'channel'];
CHANGE MASTER TO MASTER_AUTO_POSITION = 0, MASTER_LOG_FILE = file, \
MASTER_LOG_POS = position [FOR CHANNEL 'channel'];
START SLAVE [FOR CHANNEL 'channel'];

Or from MySQL 8.0.22 / 8.0.23:
STOP REPLICA [FOR CHANNEL 'channel'];
CHANGE REPLICATION SOURCE TO SOURCE_AUTO_POSITION = 0, SOURCE_LOG_FILE = file, \
SOURCE_LOG_POS = position [FOR CHANNEL 'channel'];
START REPLICA [FOR CHANNEL 'channel'];
```

2. 各サーバーで、次のコマンドを実行します:

```
SET @@GLOBAL.GTID_MODE = ON_PERMISSIVE;
```

3. 各サーバーで、次のコマンドを実行します:

```
SET @@GLOBAL.GTID_MODE = OFF_PERMISSIVE;
```

4. 各サーバーで、変数 `@@GLOBAL.GTID_OWNED` が空の文字列と等しくなるまで待機します。これは、次を使用してチェックできます:


```
SELECT @@GLOBAL.GTID_OWNED;
```

レプリカでは、理論上、これが空になってから再度空でない可能性があります。これは問題ではありません。一度空にすれば十分です。

- バイナリログに現在存在するすべてのトランザクションがすべてのレプリカにレプリケートされるのを待ちます。すべての匿名トランザクションがすべてのサーバーにレプリケートされたことを確認する方法については、[セクション 17.1.4.4 「匿名トランザクションのレプリケーションの検証」](#) を参照してください。
- レプリケーション以外にバイナリログを使用する場合 (たとえば、ポイントインタイムバックアップまたはリストアを実行する場合): GTID トランザクションを含む古いバイナリログが不要になるまで待ちます。

たとえば、ステップ 5 の完了後、バックアップを作成しているサーバーで **FLUSH LOGS** を実行できます。次に、明示的にバックアップを取るか、設定した定期バックアップルーチンの次の反復を待機します。

理想的には、ステップ 5 の完了時に存在していたすべてのバイナリログがサーバーによってパージされるまで待機します。また、ステップ 5 の前に作成したバックアップが期限切れになるまで待機します。

重要

これは、この手順で重要な点です。GTID トランザクションを含むログを次のステップの後に使用できないことを理解することが重要です。続行する前に、GTID トランザクションがトポロジ内のどこにも存在しないことを確認する必要があります。

- 各サーバーで、次のコマンドを実行します:

```
SET @@GLOBAL.GTID_MODE = OFF;
```

- 各サーバーで、`my.cnf` で `gtid_mode=OFF` を設定します。

`enforce_gtid_consistency=OFF` を設定する場合は、ここで行うことができます。設定後、`enforce_gtid_consistency=OFF` を構成ファイルに追加する必要があります。

以前のバージョンの MySQL にダウングレードする場合は、通常のダウングレード手順を使用してここでダウングレードできます。

17.1.4.4 匿名トランザクションのレプリケーションの検証

このセクションでは、レプリケーショントポロジを監視し、すべての匿名トランザクションがレプリケートされていることを確認する方法について説明します。GTID トランザクションに安全に変更できることを確認できるため、これはレプリケーションモードをオンラインに変更するとき役に立ちます。

トランザクションのレプリケートを待機するには、いくつかの方法があります:

トポロジに関係なく動作しますが、タイミングに依存する最も単純な方法は次のとおりです: レプリカが N 秒を超えないことが確実な場合は、N 秒を少し待機します。または、1 日またはデブロイメントの安全性を考慮する期間を待機します。

タイミングに依存しないという意味で安全な方法: 1 つまたは複数のレプリカを持つソースしかない場合は、次の手順を実行します:

- ソースで、次のコマンドを実行します:

```
SHOW MASTER STATUS;
```

File および **Position** カラムの値をノートにとります。

- すべてのレプリカで、ソースのファイルおよび位置情報を使用して実行します:

```
SELECT MASTER_POS_WAIT(file, position);
```

ソースと複数レベルのレプリカがある場合、またはレプリカのレプリカがある場合は、ソースから開始して各レベルでステップ 2 を繰り返し、次にすべてのダイレクトレプリカ、レプリカのすべてのレプリカなどを繰り返します。

複数のサーバーに書き込みクライアントが存在する循環レプリケーショントポロジを使用する場合は、円全体が完成するまで、ソースレプリケーション接続ごとにステップ 2 を実行します。完全な円 2 回を実行するように、プロセス全体を繰り返します。

たとえば、サーバー A、B および C が 3 つあり、A → B → C → A のように円でレプリケートしているとします。その後、プロセスは次のようになります：

- A でステップ 1 を実行し、B でステップ 2 を実行します。
- B でステップ 1 を実行し、C でステップ 2 を実行します。
- C でステップ 1 を実行し、A でステップ 2 を実行します。
- A でステップ 1 を実行し、B でステップ 2 を実行します。
- B でステップ 1 を実行し、C でステップ 2 を実行します。
- C でステップ 1 を実行し、A でステップ 2 を実行します。

17.1.5 MySQL マルチソースレプリケーション

MySQL マルチソースレプリケーションを使用すると、レプリカは複数の即時ソースからトランザクションをパラレルに受信できます。マルチソースレプリケーショントポロジでは、レプリカはトランザクションの受信元となるソースごとにレプリケーションチャンネルを作成します。レプリケーションチャンネルの機能の詳細は、[セクション 17.2.2 「レプリケーションチャンネル」](#) を参照してください。

マルチソースレプリケーションを実装して、次のような目標を達成できます：

- 複数のサーバーの単一サーバーへのバックアップ。
- テーブルシャードをマージしています。
- 複数のサーバーのデータを単一のサーバーに統合します。

マルチソースレプリケーションでは、トランザクションの適用時に競合検出または解消は実装されず、必要に応じてこれらのタスクがアプリケーションに残されます。

注記

マルチソースレプリカ上の各チャンネルは、異なるソースからレプリケートする必要があります。単一のレプリカから単一のソースに複数のレプリケーションチャンネルを設定することはできません。これは、レプリカのサーバー ID がレプリケーショントポロジ内で一意である必要があるためです。ソースはレプリカをサーバー ID でのみ区別し、レプリケーションチャンネルの名前では区別しないため、同じレプリカとは異なるレプリケーションチャンネルを認識できません。

multi-source レプリカは、`slave_parallel_workers` システム変数を 0 より大きい値に設定することで、マルチスレッドレプリカとして設定することもできます。マルチソースレプリカでこれを行う場合、レプリカ上の各チャンネルには、指定された数のアプライアスレッドと、それらを管理するためのコーディネータスレッドがあります。個々のチャンネルのアプライアスレッドの数は構成できません。

MySQL 8.0 から、特定のレプリケーションチャンネルでレプリケーションフィルタを使用してマルチソースレプリカを構成できます。チャンネル固有のレプリケーションフィルタは、同じデータベースまたはテーブルが複数のソースに存在し、単一のソースからレプリケートする場合にのみ使用できます。GTID ベースのレプリケーションでは、(ダイアモンドトポロジなどの) 複数のソースから同じトランザクションが到着する可能性がある場合、フィルタリング設定がすべてのチャンネルで同じであることを確認する必要があります。詳細は、[セクション 17.2.5.4 「レプリケーションチャンネルベースのフィルタ」](#) を参照してください。

このセクションでは、マルチソースレプリケーションのソースおよびレプリカの構成方法、マルチソースレプリカの起動、停止およびリセット方法、およびマルチソースレプリケーションの監視方法に関するチュートリアルを示します。

17.1.5.1 マルチソースレプリケーションの構成

マルチソースレプリケーショントポロジでは、少なくとも 2 つのソースと 1 つのレプリカが構成されている必要があります。これらのチュートリアルでは、2 つのソース `source1` と `source2` およびレプリカ `replicahost` があると想定しています。レプリカは、`source1` の `db1` および `source2` の `db2` の各ソースから 1 つのデータベースをレプリケートします。

マルチソースレプリケーショントポロジのソースは、GTID ベースのレプリケーションまたはバイナリログ位置ベースのレプリケーションのいずれかを使用するように構成できます。GTID ベースのレプリケーションを使用してソースを構成する方法については、[セクション 17.1.3.4 「GTID を使用したレプリケーションのセットアップ」](#) を参照してください。ファイル位置ベースのレプリケーションを使用してソースを構成する方法は、[セクション 17.1.2.1 「レプリケーションソース構成の設定」](#) を参照してください。

マルチソースレプリケーショントポロジのレプリカには、MySQL 8.0 のデフォルトであるレプリカ接続メタデータリポジトリおよびアプライヤメタデータリポジトリ用の `TABLE` リポジトリが必要です。マルチソースレプリケーションは、非推奨の代替ファイルリポジトリと互換性がありません。

レプリカが接続に使用できるすべてのソースで適切なユーザーアカウントを作成します。すべてのソースで同じアカウントを使用することも、それぞれで異なるアカウントを使用することもできます。レプリケーションの目的にだけアカウントを作成する場合、そのアカウントには `REPLICATION SLAVE` 権限だけがが必要です。たとえば、レプリカ `replicahost` から接続できる新しいユーザー `ted` を設定するには、`mysql` クライアントを使用して各ソースで次のステートメントを発行します：

```
mysql> CREATE USER 'ted'@'replicahost' IDENTIFIED BY 'password';
mysql> GRANT REPLICATION SLAVE ON *.* TO 'ted'@'replicahost';
```

MySQL 8.0 からの新規ユーザーのデフォルト認証プラグインの詳細および重要な情報は、[セクション 17.1.2.3 「レプリケーション用ユーザーの作成」](#) を参照してください。

17.1.5.2 GTID ベースのレプリケーション用のマルチソースレプリカのプロビジョニング

マルチソースレプリケーショントポロジのソースに既存のデータがある場合、レプリケーションを開始する前にレプリカに関連データをプロビジョニングする時間を節約できます。マルチソースレプリケーショントポロジでは、データディレクトリのクローニングまたはコピーを使用して、レプリカにすべてのソースのデータをプロビジョニングすることはできません。また、各ソースから特定のデータベースのみをレプリケートすることもできます。したがって、このようなレプリカをプロビジョニングする最善の戦略は、`mysqldump` を使用して各ソースに適切なダンプファイルを作成し、`mysql` クライアントを使用してレプリカにダンプファイルをインポートすることです。

GTID ベースのレプリケーションを使用している場合は、`mysqldump` がダンプ出力に配置する `SET @@GLOBAL.gtid_purged` ステートメントに注意する必要があります。このステートメントは、ソースで実行されたトランザクションの GTID をレプリカに転送し、レプリカはこの情報を必要とします。ただし、あるソースから新しい空のレプリカをプロビジョニングするよりも複雑な場合は、レプリカで使用される MySQL のバージョンでステートメントの影響を確認し、それに従ってステートメントを処理する必要があります。次のガイダンスは、適切なアクションの概要を示していますが、詳細は、`mysqldump` のドキュメントを参照してください。

`mysqldump` によって記述された `SET @@GLOBAL.gtid_purged` ステートメントの動作は、MySQL 8.0 と MySQL 5.6 および 5.7 のリリースでは異なります。MySQL 5.6 および 5.7 では、このステートメントによってレプリカ上の `gtid_purged` の値が置き換えられ、それらのリリースでは GTID (`gtid_executed` セット) を含むトランザクションのレプリカレコードが空の場合にのみその値を変更できます。したがって、マルチソースレプリケーショントポロジでは、ダンプファイルをリプレイする前に、ダンプ出力から `SET @@GLOBAL.gtid_purged` ステートメントを削除する必要があります。これは、このステートメントを含む別のダンプファイルまたは後続のダンプファイルを適用できないためです。また、MySQL 5.6 および 5.7 の場合、この制限は、ソースのすべてのダンプファイルを空の `gtid_executed` セットを持つレプリカに対する単一の操作で適用する必要があることを意味します。レプリカ上で `RESET MASTER` を発行することでレプリカ GTID 実行履歴をクリアできますが、レプリカ上に GTID との別のトランザクションが必要な場合は、[セクション 17.1.3.5 「フェイルオーバーおよびスケールアウトでの GTID の使用」](#) で説明されているプロビジョニング方法から別の方法を選択します。

MySQL 8.0 から、`SET @@GLOBAL.gtid_purged` ステートメントは GTID セットをダンプファイルからレプリカ上の既存の `gtid_purged` セットに追加します。したがって、レプリカでダンプファイルをリプレイすると、ダンプ出力にステートメントが残る可能性があり、ダンプファイルは異なるタイミングでリプレイできます。ただし、`SET @@GLOBAL.gtid_purged` ステートメントの `mysqldump` に含まれる値には、ソース上の `gtid_executed` セット内のすべてのトランザクションの GTID (データベースの抑制された部分を変更したトランザクションや、部分ダンプに含まれていなかったサーバー上のその他のデータベースも含む) が含まれることに注意してください。同じ

GTID (たとえば、同じソースからの別の部分ダンプ、または重複するトランザクションを持つ別のソースからのダンプ) を含むレプリカ上の 2 番目または後続のダンプファイルをリプレイすると、2 番目のダンプファイル内の `SET @@GLOBAL.gtid_purged` ステートメントは失敗するため、ダンプ出力から削除する必要があります。

MySQL 8.0.17 からのソースの場合、`SET @@GLOBAL.gtid_purged` ステートメントを削除するかわりに、`mysqldump` の `--set-gtid-purged` オプションを `COMMENTED` に設定してステートメントを含め、コメントアウトして、ダンプファイルのロード時にアクションが実行されないようにできます。同じソースから 2 つの部分ダンプを使用してレプリカをプロビジョニングし、2 つ目のダンプで設定された GTID が最初のダンプと同じである場合 (そのため、ダンプ間でソースで新しいトランザクションが実行されていない場合)、2 つ目のダンプファイルを出力するときに `mysqldump` の `--set-gtid-purged` オプションを `OFF` に設定して、ステートメントを省略できます。

次のプロビジョニング例では、`SET @@GLOBAL.gtid_purged` ステートメントをダンプ出力に残すことができず、ファイルから削除して手動で処理する必要があると想定しています。また、プロビジョニングを開始する前に、GTID を含む必要なトランザクションがレプリカに存在しないことも前提としています。

1. `db1` という名前のデータベース (`source1` 上) および `db2` という名前のデータベース (`source2`) のダンプファイルを作成するには、次のように `mysqldump for source1` を実行します:

```
mysqldump -u<user> -p<password> --single-transaction --triggers --routines --set-gtid-purged=ON --databases db1 > dumpM1.sql
```

次に、`mysqldump for source2` を次のように実行します:

```
mysqldump -u<user> -p<password> --single-transaction --triggers --routines --set-gtid-purged=ON --databases db2 > dumpM2.sql
```

2. `mysqldump` が各ダンプファイルに追加した `gtid_purged` 値を記録します。たとえば、MySQL 5.6 または 5.7 で作成されたダンプファイルの場合、次のような値を抽出できます:

```
cat dumpM1.sql | grep GTID_PURGED | cut -f2 -d=' ' | cut -f2 -d$'\n'
cat dumpM2.sql | grep GTID_PURGED | cut -f2 -d=' ' | cut -f2 -d$'\n'
```

フォーマットが変更された MySQL 8.0 から、次のような値を抽出できます:

```
cat dumpM1.sql | grep GTID_PURGED | perl -p0 -e 's#\^.*?/\##sg' | cut -f2 -d=' ' | cut -f2 -d$'\n'
cat dumpM2.sql | grep GTID_PURGED | perl -p0 -e 's#\^.*?/\##sg' | cut -f2 -d=' ' | cut -f2 -d$'\n'
```

各ケースの結果は GTID セットである必要があります。次に例を示します:

```
source1: 2174B383-5441-11E8-B90A-C80AA9429562:1-1029
source2: 224DA167-0C0C-11E8-8442-00059A3C7B00:1-2695
```

3. `SET @@GLOBAL.gtid_purged` ステートメントを含む各ダンプファイルから行を削除します。例:

```
sed '/GTID_PURGED/d' dumpM1.sql > dumpM1_nopurge.sql
sed '/GTID_PURGED/d' dumpM2.sql > dumpM2_nopurge.sql
```

4. `mysql` クライアントを使用して、編集した各ダンプファイルをレプリカにインポートします。例:

```
mysql -u<user> -p<password> < dumpM1_nopurge.sql
mysql -u<user> -p<password> < dumpM2_nopurge.sql
```

5. レプリカで、`RESET MASTER` を発行して GTID 実行履歴をクリアします (前述のように、すべてのダンプファイルがインポートされており、GTID を含む必要なトランザクションがレプリカにないことを前提としています)。次に、ステップ 2 で記録したように、`SET @@GLOBAL.gtid_purged` ステートメントを発行して、`gtid_purged` 値をすべてのダンプファイルからのすべての GTID セットの和集合に設定します。例:

```
mysql> RESET MASTER;
mysql> SET @@GLOBAL.gtid_purged = "2174B383-5441-11E8-B90A-C80AA9429562:1-1029, 224DA167-0C0C-11E8-8442-00059A3C7B00:1-2695";
```

ダンプファイル内の GTID セット間に重複するトランザクションがあるか、または重複している可能性がある場合は、[セクション 17.1.3.8 「GTID を操作するストアドファンクションの例」](#) で説明されているストアドファンクションを使用して、これを事前にチェックし、すべての GTID セットの結合を計算できます。

17.1.5.3 マルチソースレプリカへの GTID ベースのソースの追加

これらのステップでは、`gtid_mode=ON` を使用してソースのトランザクションに対して GTID を有効にし、レプリケーションユーザーを作成し、レプリカが `TABLE` ベースのレプリケーションアプライマメタデータリポジトリを使

用していることを確認し、必要に応じてレプリカにソースのデータをプロビジョニングしていることを前提としています。

`CHANGE REPLICATION SOURCE TO` ステートメント (MySQL 8.0.23 の場合) または `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 の場合) を使用して、レプリカ上の各ソースのレプリケーションチャンネルを構成します ([セクション17.2.2「レプリケーションチャンネル」](#) を参照)。 `FOR CHANNEL` 句を使用してチャンネルを指定します。 GTID ベースのレプリケーションでは、GTID 自動配置を使用してソースと同期されます ([セクション17.1.3.3「GTID 自動配置」](#) を参照)。 `SOURCE_AUTO_POSITION` | `MASTER_AUTO_POSITION` オプションは、自動配置の使用を指定するために設定されます。

たとえば、`source1` および `source2` をソースとしてレプリカに追加するには、`mysql` クライアントを使用して、次のようにレプリカでステートメントを 2 回発行します:

```
mysql> CHANGE MASTER TO MASTER_HOST="source1", MASTER_USER="ted", \
MASTER_PASSWORD="password", MASTER_AUTO_POSITION=1 FOR CHANNEL "source_1";
mysql> CHANGE MASTER TO MASTER_HOST="source2", MASTER_USER="ted", \
MASTER_PASSWORD="password", MASTER_AUTO_POSITION=1 FOR CHANNEL "source_2";
```

Or from MySQL 8.0.23:

```
mysql> CHANGE REPLICATION SOURCE TO SOURCE_HOST="source1", SOURCE_USER="ted", \
SOURCE_PASSWORD="password", SOURCE_AUTO_POSITION=1 FOR CHANNEL "source_1";
mysql> CHANGE REPLICATION SOURCE TO SOURCE_HOST="source2", SOURCE_USER="ted", \
SOURCE_PASSWORD="password", SOURCE_AUTO_POSITION=1 FOR CHANNEL "source_2";
```

レプリカが `source1` からデータベース `db1` のみをレプリケートし、`source2` からデータベース `db2` のみをレプリケートするようにするには、`mysql` クライアントを使用して、次のように各チャンネルに対して `CHANGE REPLICATION FILTER` ステートメントを発行します:

```
mysql> CHANGE REPLICATION FILTER REPLICATE_WILD_DO_TABLE = ('db1.%') FOR CHANNEL "source_1";
mysql> CHANGE REPLICATION FILTER REPLICATE_WILD_DO_TABLE = ('db2.%') FOR CHANNEL "source_2";
```

`CHANGE REPLICATION FILTER` ステートメントの完全な構文およびその他の使用可能なオプションについては、[セクション13.4.2.2「CHANGE REPLICATION FILTER ステートメント」](#) を参照してください。

17.1.5.4 マルチソースレプリカへのバイナリログベースレプリケーションソースの追加

これらのステップでは、バイナリロギングがソース (デフォルト) で有効になっており、レプリカが `TABLE` ベースのレプリケーションアプライアンスメタデータリポジトリ (MySQL 8.0 のデフォルト) を使用しており、レプリケーションユーザーを有効にして現在のバイナリログファイルの名前と位置をメモしていることを前提としています。

`CHANGE REPLICATION SOURCE TO` ステートメント (MySQL 8.0.23 の場合) または `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 の場合) を使用して、レプリカ上の各ソースのレプリケーションチャンネルを構成します ([セクション17.2.2「レプリケーションチャンネル」](#) を参照)。 `FOR CHANNEL` 句を使用してチャンネルを指定します。たとえば、`source1` および `source2` をソースとしてレプリカに追加するには、`mysql` クライアントを使用して、次のようにレプリカでステートメントを 2 回発行します:

```
mysql> CHANGE MASTER TO MASTER_HOST="source1", MASTER_USER="ted", MASTER_PASSWORD="password", \
MASTER_LOG_FILE='source1-bin.000006', MASTER_LOG_POS=628 FOR CHANNEL "source_1";
mysql> CHANGE MASTER TO MASTER_HOST="source2", MASTER_USER="ted", MASTER_PASSWORD="password", \
MASTER_LOG_FILE='source2-bin.000018', MASTER_LOG_POS=104 FOR CHANNEL "source_2";
```

Or from MySQL 8.0.23:

```
mysql> CHANGE REPLICATION SOURCE TO SOURCE_HOST="source1", SOURCE_USER="ted", SOURCE_PASSWORD="password", \
SOURCE_LOG_FILE='source1-bin.000006', SOURCE_LOG_POS=628 FOR CHANNEL "source_1";
mysql> CHANGE REPLICATION SOURCE TO SOURCE_HOST="source2", SOURCE_USER="ted", SOURCE_PASSWORD="password", \
SOURCE_LOG_FILE='source2-bin.000018', SOURCE_LOG_POS=104 FOR CHANNEL "source_2";
```

レプリカが `source1` からデータベース `db1` のみをレプリケートし、`source2` からデータベース `db2` のみをレプリケートするようにするには、`mysql` クライアントを使用して、次のように各チャンネルに対して `CHANGE REPLICATION FILTER` ステートメントを発行します:

```
mysql> CHANGE REPLICATION FILTER REPLICATE_WILD_DO_TABLE = ('db1.%') FOR CHANNEL "source_1";
mysql> CHANGE REPLICATION FILTER REPLICATE_WILD_DO_TABLE = ('db2.%') FOR CHANNEL "source_2";
```

`CHANGE REPLICATION FILTER` ステートメントの完全な構文およびその他の使用可能なオプションについては、[セクション13.4.2.2「CHANGE REPLICATION FILTER ステートメント」](#) を参照してください。

17.1.5.5 マルチソースレプリカの開始

すべてのレプリケーションソースにチャンネルを追加したら、[START REPLICA | SLAVE](#) ステートメントを発行してレプリケーションを開始します。レプリカで複数のチャンネルを有効にした場合は、すべてのチャンネルを起動するか、開始する特定のチャンネルを選択できます。たとえば、2つのチャンネルを個別に起動するには、`mysql` クライアントを使用して次のステートメントを発行します:

```
mysql> START SLAVE FOR CHANNEL "source_1";
mysql> START SLAVE FOR CHANNEL "source_2";
Or from MySQL 8.0.22:
mysql> START REPLICA FOR CHANNEL "source_1";
mysql> START REPLICA FOR CHANNEL "source_2";
```

[START REPLICA | SLAVE](#) コマンドの完全な構文およびその他の使用可能なオプションについては、[セクション 13.4.2.7 「START REPLICA | SLAVE ステートメント」](#) を参照してください。

両方のチャンネルが起動し、正しく動作していることを確認するには、レプリカで次のような [SHOW REPLICA | SLAVE STATUS](#) ステートメントを発行します:

```
mysql> SHOW SLAVE STATUS FOR CHANNEL "source_1"\G
mysql> SHOW SLAVE STATUS FOR CHANNEL "source_2"\G
Or from MySQL 8.0.22:
mysql> SHOW REPLICA STATUS FOR CHANNEL "source_1"\G
mysql> SHOW REPLICA STATUS FOR CHANNEL "source_2"\G
```

17.1.5.6 マルチソースレプリカの停止

[STOP REPLICA | SLAVE](#) ステートメントを使用して、マルチソースレプリカを停止できます。デフォルトでは、マルチソースレプリカで [STOP REPLICA | SLAVE](#) ステートメントを使用すると、すべてのチャンネルが停止します。オプションで、[FOR CHANNEL channel](#) 句を使用して、特定のチャンネルのみを停止します。

- 現在構成されているすべてのレプリケーションチャンネルを停止するには:

```
mysql> STOP SLAVE;
Or from MySQL 8.0.22:
mysql> STOP REPLICA;
```

- 名前付きチャンネルのみを停止するには、[FOR CHANNEL channel](#) 句を使用します:

```
mysql> STOP SLAVE FOR CHANNEL "source_1";
Or from MySQL 8.0.22:
mysql> STOP REPLICA FOR CHANNEL "source_1";
```

[STOP REPLICA | SLAVE](#) コマンドの完全な構文およびその他の使用可能なオプションについては、[セクション 13.4.2.9 「STOP REPLICA | SLAVE ステートメント」](#) を参照してください。

17.1.5.7 マルチソースレプリカのリセット

[RESET REPLICA | SLAVE](#) ステートメントを使用して、マルチソースレプリカをリセットできます。デフォルトでは、マルチソースレプリカで [RESET REPLICA | SLAVE](#) ステートメントを使用すると、すべてのチャンネルがリセットされます。オプションで、[FOR CHANNEL channel](#) 句を使用して特定のチャンネルのみをリセットします。

- 現在構成されているすべてのレプリケーションチャンネルをリセットするには:

```
mysql> RESET SLAVE;
Or from MySQL 8.0.22:
mysql> RESET REPLICA;
```

- 名前付きチャンネルのみをリセットするには、[FOR CHANNEL channel](#) 句を使用します:

```
mysql> RESET SLAVE FOR CHANNEL "source_1";
Or from MySQL 8.0.22:
mysql> RESET REPLICA FOR CHANNEL "source_1";
```

GTID ベースのレプリケーションの場合、[RESET REPLICA | SLAVE](#) はレプリカ GTID 実行履歴に影響しないことに注意してください。これをクリアする場合は、レプリカで [RESET MASTER](#) を発行します。

[RESET REPLICA | SLAVE](#) はレプリカをレプリケーション位置を忘れ、リレーログをクリアしますが、レプリケーション接続パラメータ (ソースホスト名など) やレプリケーションフィルタは変更しません。チャンネルのこれらを削除する場合は、[RESET REPLICA | SLAVE ALL](#) を発行します。

[RESET REPLICA | SLAVE](#) コマンドの完全な構文およびその他の使用可能なオプションについては、[セクション 13.4.2.5 「RESET REPLICA | SLAVE ステートメント」](#) を参照してください。

17.1.5.8 マルチソースレプリケーションの監視

レプリケーションチャンネルのステータスを監視するには、次のオプションがあります:

- レプリケーション「パフォーマンススキーマ」テーブルの使用。これらのテーブルの最初の列は `Channel_Name` です。これにより、`Channel_Name` に基づく複雑なクエリをキーとして記述できます。[セクション 27.12.11 「パフォーマンススキーマレプリケーションテーブル」](#) を参照してください。
- [SHOW REPLICA | SLAVE STATUS FOR CHANNEL channel](#) の使用。デフォルトでは、`FOR CHANNEL channel` 句を使用しない場合、このステートメントはすべてのチャンネルのレプリカステータスをチャンネルごとに 1 行ずつ表示します。識別子 `Channel_name` が結果セットの列として追加されます。`FOR CHANNEL channel` 句が指定されている場合、結果には名前付きレプリケーションチャンネルのステータスのみが表示されます。

注記

[SHOW VARIABLES](#) ステートメントは、複数のレプリケーションチャンネルでは機能しません。これらの変数を介して使用可能だった情報は、レプリケーションパフォーマンステーブルに移行されています。複数のチャンネルを含むトポロジで [SHOW VARIABLES](#) ステートメントを使用すると、デフォルトチャンネルのステータスのみが表示されます。

マルチソースレプリケーションが有効な場合に発行されるエラーコードおよびメッセージは、エラーを生成したチャンネルを指定します。

パフォーマンススキーマテーブルを使用したチャンネルの監視

このセクションでは、レプリケーション「パフォーマンススキーマ」テーブルを使用してチャンネルを監視する方法について説明します。すべてのチャンネルを監視するか、既存のチャンネルのサブセットを監視するかを選択できます。

すべてのチャンネルの接続ステータスを監視するには:

```
mysql> SELECT * FROM replication_connection_status\G;
***** 1. row *****
CHANNEL_NAME: source_1
GROUP_NAME:
SOURCE_UUID: 046e41f8-a223-11e4-a975-0811960cc264
THREAD_ID: 24
SERVICE_STATE: ON
COUNT_RECEIVED_HEARTBEATS: 0
LAST_HEARTBEAT_TIMESTAMP: 0000-00-00 00:00:00
RECEIVED_TRANSACTION_SET: 046e41f8-a223-11e4-a975-0811960cc264:4-37
LAST_ERROR_NUMBER: 0
LAST_ERROR_MESSAGE:
LAST_ERROR_TIMESTAMP: 0000-00-00 00:00:00
***** 2. row *****
CHANNEL_NAME: source_2
GROUP_NAME:
SOURCE_UUID: 7475e474-a223-11e4-a978-0811960cc264
THREAD_ID: 26
SERVICE_STATE: ON
COUNT_RECEIVED_HEARTBEATS: 0
LAST_HEARTBEAT_TIMESTAMP: 0000-00-00 00:00:00
RECEIVED_TRANSACTION_SET: 7475e474-a223-11e4-a978-0811960cc264:4-6
LAST_ERROR_NUMBER: 0
LAST_ERROR_MESSAGE:
LAST_ERROR_TIMESTAMP: 0000-00-00 00:00:00
2 rows in set (0.00 sec)
```

前述の出力では、2 つのチャンネルが有効になっており、`CHANNEL_NAME` フィールドで示されているように、これらは `source_1` および `source_2` と呼ばれます。

`CHANNEL_NAME` フィールドを追加すると、特定のチャンネルの「パフォーマンススキーマ」テーブルをクエリーすることができます。名前付きチャンネルの接続ステータスを監視するには、`WHERE CHANNEL_NAME=channel` 句を使用します:

```
mysql> SELECT * FROM replication_connection_status WHERE CHANNEL_NAME='source_1'\G
***** 1. row *****
CHANNEL_NAME: source_1
GROUP_NAME:
SOURCE_UUID: 046e41f8-a223-11e4-a975-0811960cc264
THREAD_ID: 24
SERVICE_STATE: ON
COUNT_RECEIVED_HEARTBEATS: 0
LAST_HEARTBEAT_TIMESTAMP: 0000-00-00 00:00:00
RECEIVED_TRANSACTION_SET: 046e41f8-a223-11e4-a975-0811960cc264:4-37
LAST_ERROR_NUMBER: 0
LAST_ERROR_MESSAGE:
LAST_ERROR_TIMESTAMP: 0000-00-00 00:00:00
1 row in set (0.00 sec)
```

同様に、`WHERE CHANNEL_NAME=channel` 句を使用して、特定のチャンネルの他のレプリケーション「パフォーマンススキーマ」テーブルを監視できます。詳細は、[セクション27.12.11「パフォーマンススキーマレプリケーションテーブル」](#)を参照してください。

17.1.6 レプリケーションおよびバイナリロギングのオプションと変数

以降のセクションでは、`mysqld` オプション、およびレプリケーションで使用されてバイナリログを制御するためのサーバー変数の情報について説明します。ソースおよびレプリカで使用されるオプションおよび変数は、バイナリロギングおよびグローバルトランザクション識別子 (GTID) に関連するオプションおよび変数と同様に個別にカバーされます。これらのオプションと変数に関する基本情報するクイックリファレンス表のセットも含まれています。

特に重要なのは、`server_id` システム変数です。

コマンド行形式	<code>--server-id=#</code>
システム変数	<code>server_id</code>
スコープ	グローバル
動的	はい
<code>SET_VAR</code> ヒントの適用	いいえ
型	Integer
デフォルト値	1
最小値	0
最大値	4294967295

この変数は、サーバー ID を指定します。`server_id` はデフォルトで 1 に設定されています。このデフォルト ID を使用してサーバーを起動できますが、バイナリロギングが有効になっているときに、サーバー ID を指定するように `server_id` を明示的に設定しなかった場合は、情報メッセージが発行されます。

レプリケーショントポロジで使用されるサーバーの場合、レプリケーションサーバーごとに一意のサーバー ID を 1 から $2^{32}-1$ の範囲で指定する必要があります。「Unique」は、各 ID がレプリケーショントポロジ内の他のソースまたはレプリカで使用されている他のすべての ID と異なる必要があることを意味します。詳細については、[セクション17.1.6.2「レプリケーションソースのオプションと変数」](#)、および[セクション17.1.6.3「Replica Server のオプションと変数」](#)を参照してください。

サーバー ID が 0 に設定されている場合、バイナリロギングは行われますが、サーバー ID が 0 のソースはレプリカからの接続を拒否し、サーバー ID が 0 のレプリカはソースへの接続を拒否します。サーバー ID は動的にゼロ以外の値に変更できますが、変更してもレプリケーションはすぐに開始されません。レプリカを初期化するには、サーバー ID を変更してからサーバーを再起動する必要があります。

詳細については、[セクション17.1.2.2「レプリカ構成の設定」](#)を参照してください。

`server_uuid`

MySQL サーバーは、`server_id` システム変数に設定されているデフォルトまたはユーザー指定のサーバー ID に加えて、真の UUID を生成します。これは、グローバルな読み取り専用変数 `server_uuid` として使用できます。

注記

このセクションで前述したように、`server_uuid` システム変数が存在しても、MySQL レプリケーションの準備および実行の一環として MySQL サーバーごとに一意の `server_id` 値を設定するための要件は変更されません。

システム変数	<code>server_uuid</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

起動時、MySQL サーバーは次のように自動的に UUID を取得します。

1. ファイル `data_dir/auto.cnf` に書かれている UUID を読み取って使用しようとします (ここで、`data_dir` はサーバーのデータディレクトリ)。
2. `data_dir/auto.cnf` が見つからない場合、新しい UUID を生成してこのファイルに保存します (必要に応じてファイルを作成します)。

`auto.cnf` ファイルの形式は、`my.cnf` または `my.ini` ファイルに使用される形式と似ています。`auto.cnf` には、単一の `server_uuid` 設定および値を含む単一の `[auto]` セクションのみがあります。ファイルの内容は次のように表示されます:

```
[auto]
server_uuid=8a94f357-aab4-11df-86ab-c80aa9429562
```

重要

`auto.cnf` ファイルは自動的に生成されます。このファイルを書き込んだり修正したりしなうとしないでください。

MySQL レプリケーションを使用する場合、ソースとレプリカは相互に UUID を認識します。レプリカ UUID の値は、`SHOW REPLICAS | SHOW SLAVE HOSTS` の出力に表示されます。`START REPLICA | SLAVE` が実行されると、ソース UUID の値が `SHOW REPLICA | SLAVE STATUS` の出力のレプリカで使用可能になります。

注記

`STOP REPLICA | SLAVE` または `RESET REPLICA | SLAVE` ステートメントを発行しても、レプリカで使用されているソース UUID はリセットされません。

サーバー `server_uuid` は GTID でも、そのサーバーで発生したトランザクションに使用されます。詳細は、[セクション 17.1.3 「グローバルトランザクション識別子を使用したレプリケーション」](#) を参照してください。

起動時に、`--replicate-same-server-id` オプションが設定されていないかぎり、レプリケーション I/O スレッドはエラーを生成し、そのソース UUID がそれ自体と等しい場合は中断します。また、次のいずれかに該当する場合、レプリケーション I/O スレッドは警告を生成します:

- 必要な `server_uuid` を持つソースが存在しません。
- `CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO` ステートメントは実行されていませんが、ソース `server_uuid` は変更されました。

17.1.6.1 レプリケーション、バイナリロギングオプション、および変数のリファレンス

次の 2 つのセクションでは、レプリケーションおよびバイナリログに適用可能な MySQL コマンド行オプションとシステム変数に関する基本情報を提供します。

レプリケーションオプションと変数

次のリストのコマンド行オプションとシステム変数は、レプリケーションソースサーバーと複製に関連しています。セクション17.1.6.2「レプリケーションソースのオプションと変数」では、レプリケーションソースサーバーに関連するオプションおよび変数の詳細が提供されます。レプリカに関連するオプションおよび変数の詳細は、セクション17.1.6.3「Replica Server のオプションと変数」を参照してください。

- `abort-slave-event-count`: `mysql-test` がレプリケーションのデバッグとテストに使用するオプション。
- `auto_increment_increment`: `AUTO_INCREMENT` カラムはこの値でインクリメント。
- `auto_increment_offset`: `AUTO_INCREMENT` カラムに追加されるオフセット。
- `binlog_expire_logs_seconds`: この秒数後にバイナリログをパージ。
- `binlog_gtid_simple_recovery`: GTID リカバリでのバイナリログの回復方法を制御。
- `Com_change_master`: `CHANGE MASTER TO` ステートメントの数。
- `Com_replica_start`: `START REPLICA` および `START SLAVE` ステートメントの数。
- `Com_replica_stop`: `STOP REPLICA` および `STOP SLAVE` ステートメントの数。
- `Com_show_master_status`: `SHOW MASTER STATUS` ステートメントの数。
- `Com_show_replica_status`: `SHOW REPLICA STATUS` および `SHOW SLAVE STATUS` ステートメントの数。
- `Com_show_replicas`: `SHOW REPLICAS` および `SHOW SLAVE HOSTS` ステートメントの数。
- `Com_show_slave_hosts`: `SHOW REPLICAS` および `SHOW SLAVE HOSTS` ステートメントの数。
- `Com_show_slave_status`: `SHOW REPLICA STATUS` および `SHOW SLAVE STATUS` ステートメントの数。
- `Com_slave_start`: `START REPLICA` および `START SLAVE` ステートメントの数。
- `Com_slave_stop`: `STOP REPLICA` および `STOP SLAVE` ステートメントの数。
- `disconnect-slave-event-count`: `mysql-test` がレプリケーションのデバッグとテストに使用するオプション。
- `enforce_gtid_consistency`: トランザクションで安全な方法でログに記録できないステートメントの実行を防止。
- `expire_logs_days`: この日数後にバイナリログをパージ。
- `gtid_executed`: グローバル: バイナリログ (グローバル) または現在のトランザクション (セッション) 内のすべての GTID。読み取り専用。
- `gtid_executed_compression_period`: この数のトランザクションが発生するたびに `gtid_executed` テーブルを圧縮します。0 はこのテーブルを圧縮しないことを意味します。バイナリロギングが無効な場合にかぎり適用。
- `gtid_mode`: GTID ベースのロギングを有効にするかどうか、およびどのタイプのトランザクションログを含めることができるかを制御。
- `gtid_next`: 次のステートメントを実行する GTID を指定します。詳細は、ドキュメントを参照してください。
- `gtid_owned`: このクライアント (セッション) またはすべてのクライアントが所有する GTID のセットと、所有者 (グローバル) のスレッド ID。読み取り専用。
- `gtid_purged`: バイナリログからパージされたすべての GTID のセット。
- `immediate_server_version`: 即時レプリケーションソースであるサーバーの MySQL Server リリース番号。
- `init_slave`: レプリカがソースに接続したときに実行されるステートメント。
- `log_bin_trust_function_creators`: 0 (デフォルト) に設定すると、`--log-bin` を使用した場合、ストアドファンクションの作成は SUPER 権限を持つユーザーに対してのみ許可され、作成された関数がバイナリロギングを中断しない場合のみ許可されます。

- `log_statements_unsafe_for_binlog`: エラー 1592 警告がエラーログに書き込まないようにします。
- `master-info-file`: ソースを記憶し、I/O レプリケーションスレッドがソースバイナリログ内にあるファイルの場所と名前。
- `master-retry-count`: レプリカがソースへの接続を試行してから切断する回数。
- `master_info_repository`: ソースバイナリログ内のソース情報およびレプリケーション I/O スレッドの場所を含む接続メタデータリポジトリをファイルまたはテーブルに書き込むかどうか。
- `max_relay_log_size`: ゼロでない場合、サイズがこの値を超えたときにリレーログは自動的に交替します。ゼロの場合、ローテーションが発生するサイズは `max_binlog_size` の値によって決まります。
- `original_commit_timestamp`: トランザクションが元のソースでコミットされた時刻。
- `original_server_version`: トランザクションが最初にコミットされたサーバーの MySQL Server リリース番号。
- `relay_log`: リレーログに使用する場所とベース名。
- `relay_log_basename`: ファイル名を含むリレーログへの完全パス。
- `relay_log_index`: 最後のリレーログのリストを保持するファイルに使用する場所と名前。
- `relay_log_info_file`: レプリカがリレーログに関する情報を記録するアプライアンスのメタデータリポジトリのファイル名。
- `relay_log_info_repository`: リレーログ内のレプリケーション SQL スレッドの場所をファイルまたはテーブルに書き込むかどうか。
- `relay_log_purge`: リレーログをパージするかどうかを決定。
- `relay_log_recovery`: 起動時のソースからのリレーログファイルの自動回復を有効にするかどうか。クラッシュセーフレプリカに対して有効にする必要があります。
- `relay_log_space_limit`: すべてのリレーログに使用する最大スペース。
- `replicate-do-db`: レプリケーションを指定されたデータベースに制限するようにレプリケーション SQL スレッドに指示。
- `replicate-do-table`: レプリケーションを指定されたテーブルに制限するようにレプリケーション SQL スレッドに指示。
- `replicate-ignore-db`: 指定されたデータベースにレプリケートしないようにレプリケーション SQL スレッドに指示。
- `replicate-ignore-table`: レプリケーション SQL スレッドに、指定されたテーブルにレプリケートしないように指示。
- `replicate-rewrite-db`: 元のデータベースとは異なる名前のデータベースへの更新。
- `replicate-same-server-id`: レプリケーションでは、有効な場合、サーバー ID を持つイベントをスキップしないでください。
- `replicate-wild-do-table`: レプリケーション SQL スレッドに、指定されたワイルドカードパターンに一致するテーブルにレプリケーションを制限するように指示。
- `replicate-wild-ignore-table`: 指定されたワイルドカードパターンに一致するテーブルにレプリケートしないようにレプリケーション SQL スレッドに指示。
- `replication_optimize_for_static_plugin_config`: 準同期レプリケーションの共有ロック。
- `replication_sender_observe_commit_only`: 準同期レプリケーションのための制限付きコールバック。
- `report_host`: レプリカ登録中にソースにレポートされるレプリカのホスト名または IP。
- `report_password`: レプリカサーバーがソースにレポートする任意のパスワード。レプリケーションユーザーアカウントのパスワードと同じではありません。
- `report_port`: レプリカ登録中にソースに報告されたレプリカに接続するためのポート。

- `report_user`: レプリカサーバーがソースにレポートする任意のユーザー名。レプリケーションユーザーアカウントに使用される名前と同じではありません。
- `rpl_read_size`: バイナリログファイルおよびリレーログファイルから読み取られるデータの最小量をバイト単位で設定。
- `Rpl_semi_sync_master_clients`: 準同期レプリカの数。
- `rpl_semi_sync_master_enabled`: 準同期レプリケーションがソースで有効かどうか。
- `Rpl_semi_sync_master_net_avg_wait_time`: ソースがレプリカからの返信を待機した平均時間。
- `Rpl_semi_sync_master_net_wait_time`: ソースがレプリカからの返信を待機した合計時間。
- `Rpl_semi_sync_master_net_waits`: ソースがレプリカからの返信を待機した合計回数。
- `Rpl_semi_sync_master_no_times`: ソースが準同期レプリケーションをオフにした回数。
- `Rpl_semi_sync_master_no_tx`: 肯定応答が成功しなかったコミットの数。
- `Rpl_semi_sync_master_status`: 準同期レプリケーションがソースで動作しているかどうか。
- `Rpl_semi_sync_master_timefunc_failures`: 時間関数のコール時にソースが失敗した回数。
- `rpl_semi_sync_master_timeout`: レプリカ確認応答を待機するミリ秒数。
- `rpl_semi_sync_master_trace_level`: ソースの準同期レプリケーションデバッグトレースレベル。
- `Rpl_semi_sync_master_tx_avg_wait_time`: ソースが各トランザクションを待機した平均時間。
- `Rpl_semi_sync_master_tx_wait_time`: ソースがトランザクションを待機した合計時間。
- `Rpl_semi_sync_master_tx_waits`: ソースがトランザクションを待機した合計回数。
- `rpl_semi_sync_master_wait_for_slave_count`: 続行する前に受信する必要があるレプリカ確認ソースの数 (トランザクション当たり)。
- `rpl_semi_sync_master_wait_no_slave`: レプリカがなくてもソースがタイムアウトを待機するかどうか。
- `rpl_semi_sync_master_wait_point`: レプリカトランザクション受信確認の待機ポイント。
- `Rpl_semi_sync_master_wait_pos_backtraverse`: バイナリ座標が以前に待機したイベントより小さいイベントをソースが待機した合計回数。
- `Rpl_semi_sync_master_wait_sessions`: レプリカ応答を現在待機しているセッションの数。
- `Rpl_semi_sync_master_yes_tx`: 肯定応答が成功したコミットの数。
- `rpl_semi_sync_slave_enabled`: 準同期レプリケーションがレプリカで有効かどうか。
- `Rpl_semi_sync_slave_status`: 準同期レプリケーションがレプリカで動作しているかどうか。
- `rpl_semi_sync_slave_trace_level`: レプリカの準同期レプリケーションデバッグトレースレベル。
- `rpl_stop_slave_timeout`: STOP REPLICAS または STOP SLAVE がタイムアウトするまで待機する秒数。
- `server_uuid`: サーバーの起動時に自動的に (再) 生成されるサーバーグローバル一意 ID。
- `show-slave-auth-info`: このソースの SHOW REPLICAS および SHOW SLAVE HOSTS のユーザー名とパスワードを表示。
- `skip-slave-start`: 設定されている場合、レプリケーションサーバーの起動時にレプリケーションは自動起動されません。
- `slave-skip-errors`: 指定されたリストからクエリーでエラーが返された場合にレプリケーションを続行するようレプリケーションスレッドに指示。

- `slave_checkpoint_group`: 進行状況ステータスを更新するためにチェックポイント操作がコールされる前にマルチスレッドレプリカによって処理されるトランザクションの最大数。NDB Cluster ではサポートされていません。
- `slave_checkpoint_period`: マルチスレッドレプリカの進行状況を更新し、このミリ秒後にリレーログ情報をディスクにフラッシュします。NDB Cluster ではサポートされていません。
- `slave_compressed_protocol`: ソース/レプリカプロトコルの圧縮の使用。
- `slave_exec_mode`: IDEMPOTENT モード (キーおよびその他のいくつかのエラーを抑制) と STRICT モードの間でレプリケーションスレッドを切り替えることができます。STRICT モードは NDB Cluster を除き、IDEMPOTENT が常に使用されます。
- `slave_load_tmpdir`: LOAD DATA ステートメントのレプリケート時にレプリカが一時ファイルを配置する場所。
- `slave_max_allowed_packet`: レプリケーションソースサーバーからレプリカに送信できるパケットの最大サイズ (バイト単位)。max_allowed_packet をオーバーライド。
- `slave_net_timeout`: 読取りを中断する前にソース/レプリカ接続からの追加データを待機する秒数。
- `Slave_open_temp_tables`: レプリケーション SQL スレッドが現在オープンしている一時テーブルの数。
- `slave_parallel_type`: タイムスタンプ情報 (LOGICAL_CLOCK) またはデータベースパーティション化 (DATABASE) を使用してトランザクションをパラレル化するようにレプリカに指示。
- `slave_parallel_workers`: レプリケーショントランザクションをパラレルに実行するためのアプライヤスレッドの数。0 はレプリカマルチスレッドを無効にします。MySQL クラスタではサポートされていません。
- `slave_pending_jobs_size_max`: まだ適用されていないイベントを保持するレプリカワーカーキューの最大サイズ。
- `slave_preserve_commit_order`: パラレルアプライヤスレッドの使用時に一貫性を維持するために、レプリカワーカーによるすべてのコミットがソースと同じ順序で発生するようにします。
- `Slave_rows_last_search_algorithm_used`: このレプリカが行ベースのレプリケーション (インデックス、テーブルまたはハッシュスキャン) のために最後に使用した検索アルゴリズム。
- `slave_rows_search_algorithms`: レプリカのバッチ更新に使用される検索アルゴリズムを決定します。このリストの任意の 2 または 3: INDEX_SEARCH, TABLE_SCAN, HASH_SCAN。
- `slave_transaction_retries`: デッドロックまたは経過ロック待機タイムアウトで失敗した場合に、レプリケーション SQL スレッドがトランザクションを再試行してから停止する回数。
- `slave_type_conversions`: レプリカの型変換モードを制御します。値は、このリストのゼロ個以上の要素のリストです: ALL_LOSSY, ALL_NON_LOSSY。ソースとレプリカ間の型変換を禁止するには、空の文字列に設定。
- `sql_log_bin`: 現在のセッションのバイナリロギングを制御。
- `sql_slave_skip_counter`: レプリカがスキップするソースからのイベント数。GTID レプリケーションと互換性はありません。
- `sync_master_info`: # 番目のイベントごとに、master.info とディスクの同期を取ります。
- `sync_relay_log`: # 番目のイベントごとに、リレーログとディスクの同期を取ります。
- `sync_relay_log_info`: # 番目のイベントごとに、relay.info ファイルとディスクの同期を取ります。
- `transaction_write_set_extraction`: トランザクション中に抽出された書込みのハッシュに使用されるアルゴリズムを定義。

mysqld で使用されるすべてのコマンドラインオプション、システム変数およびステータス変数のリストは、[セクション 5.1.4 「サーバーオプション、システム変数およびステータス変数リファレンス」](#) を参照してください。

バイナリロギングのオプションと変数

次のリストのコマンド行オプションとシステム変数は、バイナリログに関連しています。[セクション 17.1.6.4 「バイナリロギングのオプションと変数」](#) では、バイナリロギングに関連するオプションと変数について詳しく説明します。バイナリログに関するその他の一般情報については、[セクション 5.4.4 「バイナリログ」](#) を参照してください。

- [binlog-checksum](#): バイナリログチェックサムを有効化/無効化.
- [binlog-do-db](#): バイナリロギングを特定のデータベースに限定.
- [binlog-ignore-db](#): 指定されたデータベースへの更新をバイナリログに書き込まないようにソースに指示.
- [binlog-row-event-max-size](#): バイナリログの最大イベントサイズ.
- [Binlog_cache_disk_use](#): バイナリログキャッシュの代わりに一時ファイルを使用したトランザクションの数.
- [binlog_cache_size](#): トランザクション中にバイナリログの SQL ステートメントを保持するキャッシュのサイズ.
- [Binlog_cache_use](#): 一時バイナリログキャッシュを使用したトランザクションの数.
- [binlog_checksum](#): バイナリログチェックサムを有効化/無効化.
- [binlog_direct_non_transactional_updates](#): 非トランザクションエンジンへの、ステートメント形式を使用する更新が、直接バイナリログに書き込まれるようになります。使用する前にドキュメントを参照してください.
- [binlog_encryption](#): このサーバー上のバイナリログファイルおよびリレーログファイルの暗号化を有効にします.
- [binlog_error_action](#): サーバーがバイナリログに書き込めない場合の動作を制御.
- [binlog_format](#): バイナリログの形式を指定.
- [binlog_group_commit_sync_delay](#): トランザクションをディスクに同期する前に待機するマイクロ秒数を設定.
- [binlog_group_commit_sync_no_delay_count](#): [binlog_group_commit_sync_delay](#) で指定された現在の遅延を中断する前に待機するトランザクションの最大数を設定.
- [binlog_max_flush_queue_time](#): バイナリログにフラッシュするまでにどれくらいトランザクションを読み取るか.
- [binlog_order_commits](#): バイナリログへの書き込みと同じ順序でコミットするかどうか.
- [binlog_rotate_encryption_master_key_at_startup](#): サーバー起動時のバイナリログマスターキーのローテーション.
- [binlog_row_image](#): 行の変更のロギングの際にフルイメージまたは最少イメージを使用.
- [binlog_row_metadata](#): 行ベースのロギングを使用している場合に、テーブル関連のすべてのメタデータをバイナリログに記録するか、最小限のメタデータのみを記録するか.
- [binlog_row_value_options](#): 行ベースレプリケーションの部分 JSON 更新のバイナリロギングを有効にします.
- [binlog_rows_query_log_events](#): 有効にすると、行ベースロギングの使用時に行クエリーログイベントのロギングが有効になります。デフォルトで無効になっています。5.6 より前のレプリカリーダーのログの生成時に有効にしないでください.
- [Binlog_stmt_cache_disk_use](#): バイナリログステートメントキャッシュの代わりに一時ファイルを使用した非トランザクションステートメントの数.
- [binlog_stmt_cache_size](#): トランザクション中にバイナリログの非トランザクションステートメントを保持するキャッシュのサイズ.
- [Binlog_stmt_cache_use](#): 一時バイナリログステートメントキャッシュを使用したステートメントの数.
- [binlog_transaction_compression](#): バイナリログファイル内のトランザクションペイロードの圧縮を有効にします.
- [binlog_transaction_compression_level_zstd](#): バイナリログファイル内のトランザクションペイロードの圧縮レベル.
- [binlog_transaction_dependency_history_size](#): 一部の行を最後に更新したトランザクションを参照するために保持される行ハッシュの数.
- [binlog_transaction_dependency_tracking](#): レプリカマルチスレッドアプリケーションでパラレルに実行できるトランザクションを評価する依存性情報 (コミットタイムスタンプまたはトランザクション書き込みセット) のソース.
- [Com_show_binlog_events](#): SHOW BINLOG EVENTS ステートメントの数.

- `Com_show_binlogs`: SHOW BINLOGS ステートメントの数.
- `log-bin`: バイナリログファイルのベース名.
- `log-bin-index`: バイナリログインデックスファイル名.
- `log_bin`: バイナリログが有効かどうか.
- `log_bin_basename`: バイナリログファイルのパスとベース名.
- `log_bin_use_v1_row_events`: サーバーがバージョン 1 バイナリログ行イベントを使用しているかどうか.
- `log_slave_updates`: レプリケーション SQL スレッドによって実行された更新を独自のバイナリログに記録するかどうか.
- `master_verify_checksum`: バイナリログからの読み取り時にソースがチェックサムを検査するようにします.
- `max-binlog-dump-events`: mysql-test がレプリケーションのデバッグとテストに使用するオプション.
- `max_binlog_cache_size`: 複数ステートメントトランザクションのキャッシュに使用する合計サイズを制限するために使用できます.
- `max_binlog_size`: バイナリログは、サイズがこの値を超えると自動的にローテーションされます.
- `max_binlog_stmt_cache_size`: トランザクション中にすべての非トランザクションステートメントのキャッシュに使用される合計サイズを制限するために使用できます.
- `slave-sql-verify-checksum`: リレーログからの読み取り時にレプリカがチェックサムを検査するようにします.
- `slave_sql_verify_checksum`: リレーログからの読み取り時にレプリカがチェックサムを検査するようにします.
- `sporadic-binlog-dump-fail`: mysql-test がレプリケーションのデバッグとテストに使用するオプション.
- `sync_binlog`: # 番目のイベントごとに、同期してバイナリログをディスクにフラッシュ.

mysqld で使用されるすべてのコマンドラインオプション、システム変数およびステータス変数のリストは、[セクション 5.1.4 「サーバーオプション、システム変数およびステータス変数リファレンス」](#) を参照してください。

17.1.6.2 レプリケーションソースのオプションと変数

このセクションでは、レプリケーションソースサーバーで使用できるサーバーオプションおよびシステム変数について説明します。オプションは[コマンド行](#)または[オプションファイル](#)で指定できます。システム変数値はSET を使用して指定できます。

ソースおよび各レプリカで、`server_id` システム変数を設定して一意のレプリケーション ID を確立する必要があります。サーバーごとに、1 から $2^{32}-1$ の範囲の一意の正の整数を選択する必要があります。各 ID はレプリケーショントポロジ内の他のソースまたはレプリカで使用されている他のすべての ID と異なる必要があります。例: `server-id=3`。

バイナリロギングを制御するためにソースで使用されるオプションについては、[セクション 17.1.6.4 「バイナリロギングのオプションと変数」](#) を参照してください。

レプリケーションソースサーバーの起動オプション

次のリストでは、レプリケーションソースサーバーを制御するための起動オプションについて説明します。レプリケーションに関連するシステム変数はこのセクションの後半で説明します。

- `--show-slave-auth-info`

コマンド行形式	<code>--show-slave-auth-info[={OFF ON}]</code>
型	Boolean
デフォルト値	OFF

`--report-user` および `--report-password` オプションを使用して開始されたレプリカのソース上の `SHOW REPLICAS | SHOW SLAVE HOSTS` の出力に、レプリケーションユーザー名とパスワードを表示します。

レプリケーションソースサーバーで使用されるシステム変数

次のシステム変数は、レプリケーションソースサーバーに対して、またはレプリケーションソースサーバーによって使用されます:

- `auto_increment_increment`

コマンド行形式	<code>--auto-increment-increment=#</code>
システム変数	<code>auto_increment_increment</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	はい
型	Integer
デフォルト値	1
最小値	1
最大値	65535

`auto_increment_increment` および `auto_increment_offset` は、循環 (ソースからソース) レプリケーションでの使用を目的としており、`AUTO_INCREMENT` カラムの操作を制御するために使用できます。両方の変数はグローバル値とセッション値を持ち、各値は 1 から 65,535 (1 と 65,535 を含みます) の間の整数値を取ることができます。これらの 2 つの変数のいずれかの値を 0 に設定すると、代わりにその値は 1 に設定されます。これらの 2 つの変数のいずれかの値を 65,535 より大きな整数または 0 より小さい整数に設定しようとする、代わりにその値は 65,535 に設定されます。`auto_increment_increment` または `auto_increment_offset` の値を整数でない値に設定しようとする、エラーが発生し、変数の実際の値は変化しません。

注記

`auto_increment_increment` は NDB テーブルで使用する場合にもサポートされます。

MySQL 8.0.18 では、このシステム変数のセッション値の設定は制限付き操作ではなくなりました。

グループレプリケーションがサーバーで開始されると、`auto_increment_increment` の値は `group_replication_auto_increment_increment` の値に変更され (デフォルトは 7)、`auto_increment_offset` の値はサーバー ID に変更されます。グループレプリケーションが停止すると、変更は元に戻されます。これらの変更は、`auto_increment_increment` および `auto_increment_offset` のそれぞれのデフォルト値が 1 の場合にのみ行われ、元に戻されます。これらの値がすでにデフォルトから変更されている場合、Group Replication はそれらを変更しません。MySQL 8.0 からは、グループレプリケーションが単一プライマリモードで、サーバー書き込みが 1 つのみの場合も、システム変数は変更されません。

`auto_increment_increment` および `auto_increment_offset` は、`AUTO_INCREMENT` のカラムの動作に次のように影響します:

- `auto_increment_increment` は、連続するカラム値の間隔を制御します。例:

```
mysql> SHOW VARIABLES LIKE 'auto_inc%';
+-----+-----+
| Variable_name      | Value |
+-----+-----+
| auto_increment_increment | 1    |
| auto_increment_offset  | 1    |
+-----+-----+
2 rows in set (0.00 sec)

mysql> CREATE TABLE autoinc1
-> (col INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
Query OK, 0 rows affected (0.04 sec)

mysql> SET @@auto_increment_increment=10;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'auto_inc%';
```

```

+-----+-----+
| Variable_name | Value |
+-----+-----+
| auto_increment_increment | 10 |
| auto_increment_offset | 1 |
+-----+-----+
2 rows in set (0.01 sec)

mysql> INSERT INTO autoinc1 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4 Duplicates: 0 Warnings: 0

mysql> SELECT col FROM autoinc1;
+----+
| col |
+----+
| 1 |
| 11 |
| 21 |
| 31 |
+----+
4 rows in set (0.00 sec)

```

- `auto_increment_offset` は `AUTO_INCREMENT` カラム値の開始点を指定します。次のことは、`auto_increment_increment` の記述で示した例のように、同じセッション中にこれらのステートメントが実行されるものと仮定しています。

```

mysql> SET @@auto_increment_offset=5;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'auto_inc%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| auto_increment_increment | 10 |
| auto_increment_offset | 5 |
+-----+-----+
2 rows in set (0.00 sec)

mysql> CREATE TABLE autoinc2
-> (col INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO autoinc2 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4 Duplicates: 0 Warnings: 0

mysql> SELECT col FROM autoinc2;
+----+
| col |
+----+
| 5 |
| 15 |
| 25 |
| 35 |
+----+
4 rows in set (0.02 sec)

```

`auto_increment_offset` の値が `auto_increment_increment` の値よりも大きいと、`auto_increment_offset` の値は無視されます。

これらの変数のいずれかが変更されてから、`AUTO_INCREMENT` カラムを含むテーブルに新しい行が挿入される場合、結果は反直感的に見える場合があります。`AUTO_INCREMENT` 値のシリーズがカラムにすでに存在する値に関係なく計算され、挿入される次の値が `AUTO_INCREMENT` カラムに存在する最大値よりも大きなシリーズ内最小値であるためです。シリーズは次のように計算されます。

`auto_increment_offset + N × auto_increment_increment`

ここで、`N` はシリーズ内正の整数値 [1, 2, 3, ...] です。例:

```
mysql> SHOW VARIABLES LIKE 'auto_inc%';
```

```

+-----+-----+
| Variable_name | Value |
+-----+-----+
| auto_increment_increment | 10 |
| auto_increment_offset | 5 |
+-----+-----+
2 rows in set (0.00 sec)

mysql> SELECT col FROM autoinc1;
+----+
| col |
+----+
| 1 |
| 11 |
| 21 |
| 31 |
+----+
4 rows in set (0.00 sec)

mysql> INSERT INTO autoinc1 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4 Duplicates: 0 Warnings: 0

mysql> SELECT col FROM autoinc1;
+----+
| col |
+----+
| 1 |
| 11 |
| 21 |
| 31 |
| 35 |
| 45 |
| 55 |
| 65 |
+----+
8 rows in set (0.00 sec)

```

`auto_increment_increment` と `auto_increment_offset` に示される値は、シリーズ $5 + N \times 10$ 、つまり [5, 15, 25, 35, 45, ...] を生成します。INSERT より前に `col` カラムに存在する最高値は 31、`AUTO_INCREMENT` シリーズ内で次に使用できる値は 35 なので、`col` に挿入される値はそのポイントで始まり、結果は `SELECT` クエリーで示されるようになります。

これらの 2 つの変数の影響を単一テーブルに制限することはできません。これらの変数は MySQL サーバーのすべてのテーブルのすべての `AUTO_INCREMENT` カラムの動作を制御します。どちらかの変数のグローバル値が設定されると、グローバル値が変更されるか、セッション値の設定によってオーバーライドされるまで、または `mysqld` が再起動されるまでその効果は持続します。ローカル値が設定されると、新しい値は、セッションの期間に現在のユーザーが新しい行を挿入したすべてのテーブルの `AUTO_INCREMENT` カラムに影響します (そのセッション中にそれらの値が変更される場合を除く)。

`auto_increment_increment` のデフォルト値は 1 です。 [セクション17.5.1.1「レプリケーションと AUTO_INCREMENT」](#) を参照してください。

- `auto_increment_offset`

コマンド行形式	<code>--auto-increment-offset=#</code>
システム変数	<code>auto_increment_offset</code>
スコープ	グローバル、セッション
動的	はい
<code>SET_VAR</code> ヒントの適用	はい
型	Integer
デフォルト値	1
最小値	1

最大値	65535
-----	-------

この変数のデフォルト値は 1 です。デフォルト値のままにしておくと、グループレプリケーションがマルチプライマリモードでサーバーで起動され、サーバー ID に変更されます。詳細については、[auto_increment_increment](#) の説明を参照してください。

注記

[auto_increment_offset](#) は NDB テーブルで使用する場合にもサポートされます。

MySQL 8.0.18 では、このシステム変数のセッション値の設定は制限付き操作ではなくなりました。

• [immediate_server_version](#)

導入	8.0.14
システム変数	immediate_server_version
スコープ	セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer

レプリケーションによる内部使用。このセッションシステム変数は、レプリケーショントポロジ内の直接のソースであるサーバーの MySQL Server リリース番号 (MySQL 8.0.14 サーバーインスタンスの [80014](#) など) を保持します。この即時サーバーがセッションシステム変数をサポートしていないリリースの場合、変数の値は 0 ([UNKNOWN_SERVER_VERSION](#)) に設定されます。

変数の値は、ソースからレプリカにレプリケートされます。この情報を使用すると、関係するリリース間で構文変更またはセマンティック変更が発生した場所を認識し、それらを適切に処理することで、レプリカは古いリリースのソースから発生したデータを正しく処理できます。この情報は、レプリケーショングループの 1 つ以上のメンバーが他のメンバーより新しいリリースであるグループレプリケーション環境でも使用できます。変数の値は、([Gtid_log_event](#) または GTID がサーバーで使用されていない場合は [Anonymous_gtid_log_event](#) の一部として) 各トランザクションのバイナリログに表示でき、バージョン間レプリケーションの問題のデバッグに役立つことがあります。

このシステム変数のセッション値の設定は制限された操作です。セッションユーザーには、[REPLICATION_APPLIER](#) 権限 ([セクション 17.3.3 「レプリケーション権限チェック」](#) を参照) または制限付きセッション変数の設定に十分な権限 ([セクション 5.1.9.1 「システム変数権限」](#) を参照) が必要です。ただし、この変数はユーザーが設定するためのものではなく、レプリケーションインフラストラクチャによって自動的に設定されることに注意してください。

• [original_server_version](#)

導入	8.0.14
システム変数	original_server_version
スコープ	セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer

レプリケーションによる内部使用。このセッションシステム変数は、トランザクションが最初にコミットされたサーバーの MySQL Server リリース番号 (MySQL 8.0.14 サーバーインスタンスの [80014](#) など) を保持します。この元のサーバーがセッションシステム変数をサポートしていないリリースの場合、変数の値は 0 ([UNKNOWN_SERVER_VERSION](#)) に設定されます。リリース番号が元のサーバーによって設定されている場合、

レプリケーショントポロジ内の即時サーバーまたはその他の介在するサーバーがセッションシステム変数をサポートしていないため、その値をレプリケートしないと、変数の値は 0 にリセットされます。

変数の値は、`immediate_server_version` システム変数と同じ方法で設定および使用されます。変数の値が `immediate_server_version` システム変数の値と同じである場合、後者のみがバイナリログに記録され、元のサーバーのバージョンが同じであることが示されます。

グループレプリケーション環境では、変更ロギイベントを表示します。変更ロギイベントは、新しいメンバーがグループに参加したときに各グループメンバーによってキューに入れられる特別なトランザクションであり、トランザクションをキューに入れるグループメンバーのサーバーバージョンでタグ付けされます。これにより、元のドナーのサーバーバージョンが参加メンバーに認識されるようになります。特定のビュー変更に対してキューに入れられたビュー変更ロギイベントは、すべてのメンバーで同じ GTID を持つため、この場合のみ、同じ GTID のインスタンスの元のサーバーバージョンが異なる可能性があります。

このシステム変数のセッション値の設定は制限された操作です。セッションユーザーには、`REPLICATION_APPLIER` 権限 (セクション 17.3.3 「レプリケーション権限チェック」を参照) または制限付きセッション変数の設定に十分な権限 (セクション 5.1.9.1 「システム変数権限」を参照) が必要です。ただし、この変数はユーザーが設定するためのものではなく、レプリケーションインフラストラクチャによって自動的に設定されることに注意してください。

- `rpl_semi_sync_master_enabled`

コマンド行形式	<code>--rpl-semi-sync-master-enabled[={OFF ON}]</code>
システム変数	<code>rpl_semi_sync_master_enabled</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

ソースサーバーで準同期レプリケーションを有効にするかどうかを制御します。プラグインを有効または無効にするには、この変数を ON または OFF (あるいは 1 または 0) にそれぞれ設定します。デフォルトは OFF です。

この変数は、ソース側の準同期レプリケーションプラグインがインストールされている場合にのみ使用できます。

- `rpl_semi_sync_master_timeout`

コマンド行形式	<code>--rpl-semi-sync-master-timeout=#</code>
システム変数	<code>rpl_semi_sync_master_timeout</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	10000

ソースがタイムアウトして非同期レプリケーションに戻る前にレプリカからの確認応答をコミットで待機する時間を制御するミリ秒単位の値。デフォルト値は 10000 (10 秒) です。

この変数は、ソース側の準同期レプリケーションプラグインがインストールされている場合にのみ使用できます。

- `rpl_semi_sync_master_trace_level`

コマンド行形式	<code>--rpl-semi-sync-master-trace-level=#</code>
システム変数	<code>rpl_semi_sync_master_trace_level</code>
スコープ	グローバル

動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	32

ソースサーバーの準同期レプリケーションのデバッグトレースレベル。次の4つのレベルが定義されます:

- 1 = 一般レベル (時間関数の失敗など)
- 16 = 詳細レベル (詳細情報)
- 32 = ネット待機レベル (ネットワーク待機についての詳細情報)
- 64 = 関数レベル (関数の入口および出口についての情報)

この変数は、ソース側の準同期レプリケーションプラグインがインストールされている場合にのみ使用できます。

- [rpl_semi_sync_master_wait_for_slave_count](#)

コマンド行形式	<code>--rpl-semi-sync-master-wait-for-slave-count=#</code>
システム変数	<code>rpl_semi_sync_master_wait_for_slave_count</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1
最小値	1
最大値	65535

続行する前にソースがトランザクションごとに受信する必要があるレプリカ確認の数。デフォルトでは、`rpl_semi_sync_master_wait_for_slave_count` は 1 です。つまり、準同期レプリケーションは、単一のレプリカ確認応答を受信した後に続行されます。パフォーマンスは、この変数の小さい値に最適です。

たとえば、`rpl_semi_sync_master_wait_for_slave_count` が 2 の場合、準同期レプリケーションを続行するには、`rpl_semi_sync_master_timeout` によって構成されたタイムアウト期間の前に、2 つのレプリカがトランザクションの受信を確認する必要があります。タイムアウト期間中にトランザクションの受信を確認するレプリカの数がない場合、ソースは通常のレプリケーションに戻ります。

注記

この動作は `rpl_semi_sync_master_wait_no_slave` にも依存

この変数は、ソース側の準同期レプリケーションプラグインがインストールされている場合にのみ使用できます。

- [rpl_semi_sync_master_wait_no_slave](#)

コマンド行形式	<code>--rpl-semi-sync-master-wait-no-slave[={OFF ON}]</code>
システム変数	<code>rpl_semi_sync_master_wait_no_slave</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean

デフォルト値	ON
--------	----

レプリカ数がタイムアウト期間中に `rpl_semi_sync_master_wait_for_slave_count` によって構成されたレプリカの数を下回る場合でも、`rpl_semi_sync_master_timeout` によって構成されたタイムアウト期間が経過するまでソースが待機するかどうかを制御します。

`rpl_semi_sync_master_wait_no_slave` の値が ON (デフォルト) の場合、レプリカ数はタイムアウト期間中に `rpl_semi_sync_master_wait_for_slave_count` 未満にドロップできます。タイムアウト期間が経過する前に十分なレプリカがトランザクションを確認するがぎり、準同期レプリケーションは続行されます。

`rpl_semi_sync_master_wait_no_slave` の値が OFF の場合、`rpl_semi_sync_master_timeout` で構成されたタイムアウト期間中に `rpl_semi_sync_master_wait_for_slave_count` で構成された数より少ない数にレプリカ数がドロップされると、ソースは通常のレプリケーションに戻ります。

この変数は、ソース側の準同期レプリケーションプラグインがインストールされている場合にのみ使用できます。

- `rpl_semi_sync_master_wait_point`

コマンド行形式	<code>--rpl-semi-sync-master-wait-point=value</code>
システム変数	<code>rpl_semi_sync_master_wait_point</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	AFTER_SYNC
有効な値	AFTER_SYNC AFTER_COMMIT

この変数は、準同期レプリケーションソースサーバーがトランザクションをコミットしたクライアントにステータスを返す前に、トランザクション受信のレプリカ確認を待機するポイントを制御します。次の値を使用できます:

- **AFTER_SYNC** (デフォルト): ソースは、各トランザクションをバイナリログとレプリカに書き込み、バイナリログをディスクに同期します。ソースは、同期後にトランザクション受信のレプリカ確認を待機します。確認応答を受信すると、ソースはトランザクションをストレージエンジンにコミットし、クライアントに結果を返してから続行できます。
- **AFTER_COMMIT**: ソースは、各トランザクションをバイナリログとレプリカに書き込み、バイナリログを同期し、トランザクションをストレージエンジンにコミットします。ソースは、コミット後にトランザクション受信のレプリカ確認を待機します。確認を受信すると、ソースは結果をクライアントに返し、クライアントは続行できます。

これらの設定のレプリケーション特性は、次のように異なります:

- **AFTER_SYNC** では、すべてのクライアントに同時にコミットされたトランザクションが表示されます: レプリカによって確認され、ソース上のストレージエンジンにコミットされたあと。したがって、すべてのクライアントにソース上の同じデータが表示されます。

ソース障害が発生した場合、ソースでコミットされたすべてのトランザクションがレプリカにレプリケートされます (リレーログに保存されます)。レプリカが最新であるため、ソースサーバーの予期しない終了およびレプリカへのフェイルオーバーは失われません。ただし、バイナリログレプリカ後に外部化された場合にレプリカとの競争を引き起こすコミットされていないトランザクションがバイナリログに含まれる可能性があるため、このシナリオではソースを再起動できず、破棄する必要があることに注意してください。

- **AFTER_COMMIT** では、サーバーがストレージエンジンにコミットしてレプリカの確認応答を受信したあとのみ、トランザクションを発行するクライアントは戻りステータスを取得します。コミット後およびレプリカの確認前に、他のクライアントはコミット中のクライアントの前にコミット済トランザクションを確認できます。

レプリカがトランザクションを処理しないなどの問題が発生した場合は、予期しないソースサーバーが終了してレプリカにフェイルオーバーしたときに、そのようなクライアントがソースで見た内容と比較してデータが失われる可能性があります。

この変数は、ソース側の準同期レプリケーションプラグインがインストールされている場合にのみ使用できます。

MySQL 5.7 での `rpl_semi_sync_master_wait_point` の追加により、準同期インタフェースバージョンが増分されるため、バージョン互換性制約が作成されました: MySQL 5.7 以上のサーバーは、旧バージョンの準同期レプリケーションプラグインでは動作しません。また、旧バージョンのサーバーは、MySQL 5.7 以上の準同期レプリケーションプラグインでは動作しません。

17.1.6.3 Replica Server のオプションと変数

このセクションでは、レプリカサーバーに適用されるサーバーオプションおよびシステム変数について説明します。このセクションの内容は次のとおりです:

- [Replica Server の起動オプション](#)
- [レプリカサーバーで使用されるシステム変数](#)

オプションは**コマンド行**または**オプションファイル**で指定します。サーバーの実行中に、(MySQL 8.0.23 の) `CHANGE REPLICATION SOURCE TO` ステートメントまたは (MySQL 8.0.23 の前の) `CHANGE MASTER TO` ステートメントを使用して、多くのオプションを設定できます。システム変数値は `SET` を使用して指定します。

サーバー ID. ソースおよび各レプリカで、`server_id` システム変数を設定して、1 から $2^{32}-1$ の範囲の一意のレプリケーション ID を確立する必要があります。「Unique」は、各 ID がレプリケーショントポロジ内の他のソースまたはレプリカで使用されている他のすべての ID と異なる必要があることを意味します。 `my.cnf` ファイルの例:

```
[mysqld]
server-id=3
```

Replica Server の起動オプション

このセクションでは、レプリカサーバーを制御するための起動オプションについて説明します。これらのオプションの多くは、(MySQL 8.0.23 の) `CHANGE REPLICATION SOURCE TO` ステートメントまたは (MySQL 8.0.23 の前の) `CHANGE MASTER TO` ステートメントを使用して、サーバーの実行中に設定できます。その他のオプション (`--replicate-*` オプションなど) は、レプリカサーバーの起動時のみ設定できます。レプリケーションに関連するシステム変数はこのセクションの後半で説明します。

- `--master-info-file=file_name`

コマンド行形式	<code>--master-info-file=file_name</code>
非推奨	8.0.18
型	ファイル名
デフォルト値	<code>master.info</code>

このオプションの使用は非推奨になりました。 `master_info_repository=FILE` が設定されている場合は、レプリカ接続メタデータリポジトリのファイル名を設定するために使用されていました。接続メタデータリポジトリのファイルの使用がクラッシュセーフテーブルに置き換えられたため、`--master-info-file` および `master_info_repository` システム変数の使用は非推奨になりました。接続メタデータリポジトリの詳細は、[セクション17.2.4.2「レプリケーションメタデータリポジトリ」](#) を参照してください。

- `--master-retry-count=count`

コマンド行形式	<code>--master-retry-count=#</code>
非推奨	はい

型	Integer
デフォルト値	86400
最小値	0
最大値 (64 ビットプラットフォーム)	18446744073709551615
最大値 (32 ビットプラットフォーム)	4294967295

レプリカがソースへの再接続を試行してから中止する回数。デフォルト値は 86400 回です。値 0 は「無限」を意味し、レプリカは永久に接続を試みます。再接続の試行は、レプリカが (`slave_net_timeout` システム変数で指定された) 接続タイムアウトに達したときに、ソースからデータまたはハートビートシグナルを受信せずにトリガーされます。再接続は、`CHANGE REPLICATION SOURCE TO` | `CHANGE MASTER TO` ステートメントの `SOURCE_CONNECT_RETRY` | `MASTER_CONNECT_RETRY` オプションで設定された間隔 (デフォルトは 60 秒ごと) で試行されます。

このオプションは非推奨です。将来の MySQL リリースで削除される予定です。かわりに、`CHANGE REPLICATION SOURCE TO` | `CHANGE MASTER TO` ステートメントの `SOURCE_RETRY_COUNT` | `MASTER_RETRY_COUNT` オプションを使用してください。

- `--max-relay-log-size=size`

コマンド行形式	<code>--max-relay-log-size=#</code>
システム変数	<code>max_relay_log_size</code>
スコープ	グローバル
動的	はい
<code>SET_VAR</code> ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	1073741824

このサイズで、サーバーはリレーログファイルを自動的にローテーションします。この値がゼロでない場合は、サイズがこの値を超えたときにリレーログは自動的にローテーションされます。この値がゼロ (デフォルト) の場合、リレーログローテーションが発生するサイズは `max_binlog_size` の値によって決められます。詳細は、[セクション 17.2.4.1「リレーログ」](#) を参照してください。

- `--relay-log-purge={0|1}`

コマンド行形式	<code>--relay-log-purge[={OFF ON}]</code>
システム変数	<code>relay_log_purge</code>
スコープ	グローバル
動的	はい
<code>SET_VAR</code> ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

リレーログファイルが不要になるとすぐに自動的にパージすることを無効または有効にします。デフォルト値は 1 (有効) です。これは `SET GLOBAL relay_log_purge = N` で動的に変更できるグローバル変数です。`--relay-log-recovery` オプションを有効にするときにリレーログのパージを無効にすると、データの整合性が損なわれるため、クラッシュセーフではありません。

- `--relay-log-space-limit=size`

コマンド行形式	<code>--relay-log-space-limit=#</code>
---------	--

システム変数	relay_log_space_limit
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値 (64 ビットプラットフォーム)	18446744073709551615
最大値 (32 ビットプラットフォーム)	4294967295

このオプションは、レプリカ上のすべてのリレーログの合計サイズの上限をバイト単位で設定します。値 0 は「制限なし」を表します。これは、ディスク容量が限られたレプリカサーバーホストに役立ちます。制限に達すると、SQL スレッドがいくつかの未使用のリレーログをキャッチアップして削除するまで、I/O スレッドはソースサーバーからのバイナリロギングイベントの読み取りを停止します。この制限は絶対ではありません。SQL スレッドがリレーログを削除する前により多くのイベントを必要とする場合があります。その場合、SQL スレッドが一部のリレーログを削除できるようになるまで I/O スレッドは制限を超えます。そうしないとデッドロックになるためです。--relay-log-space-limit を --max-relay-log-size (または --max-relay-log-size が 0 の場合は --max-binlog-size) の値の 2 倍未満に設定しないでください。その場合、I/O スレッドが空き領域を待機する可能性があります。--relay-log-space-limit を超えたけれども、SQL スレッドはパージするリレーログを持たず、I/O スレッドを満たすことができないためです。この場合、I/O スレッドは強制的に --relay-log-space-limit を一時的に無視します。

- --replicate-do-db=db_name

コマンド行形式	--replicate-do-db=name
型	文字列

データベースの名前を使用してレプリケーションフィルタを作成します。このようなフィルタは、[CHANGE REPLICATION FILTER REPLICATE_DO_DB](#) を使用しても作成できます。

このオプションでは、チャンネル固有のレプリケーションフィルタがサポートされているため、マルチソースレプリカは異なるソースに対して特定のフィルタを使用できます。channel_1 という名前のチャンネルでチャンネル固有のレプリケーションフィルタを構成するには、[--replicate-do-db:channel_1:db_name](#) を使用します。この場合、最初のコロンはセパレータとして解釈され、後続のコロンはリテラルコロンです。詳しくは[セクション 17.2.5.4「レプリケーションチャンネルベースのフィルタ」](#)をご覧ください。

注記

グループレプリケーション用に構成された MySQL サーバーインスタンスでは、グローバルレプリケーションフィルタを使用できません。これは、一部のサーバーでトランザクションをフィルタすると、グループが一貫性のある状態で承諾に到達できなくなるためです。グループメンバーがグループ外のソースへのレプリカとしても機能する場合など、グループレプリケーションに直接関与しないレプリケーションチャンネルでチャンネル固有のレプリケーションフィルタを使用できます。group_replication_applier または group_replication_recovery チャンネルでは使用できません。

このレプリケーションフィルタの正確な効果は、ステートメントベースレプリケーションと行ベースレプリケーションのどちらが使用されているかによって異なります。

ステートメントベースのレプリケーション. レプリケーション SQL スレッドに、デフォルトデータベース (つまり、[USE](#) によって選択されたデータベース) が db_name であるステートメントにレプリケーションを制限するように指示します。複数のデータベースを指定するには、このオプションを複数回 (データベースごとに 1 回) 使用

します。ただし、このようにすると別のデータベースは選択される (またはデータベースが選択されない) けれども、`UPDATE some_db.some_table SET foo='bar'` などのクロスデータベースステートメントを複製しません。

警告

複数のデータベースを指定するには、このオプションの複数インスタンスを使用する必要があります。データベース名にはカンマを含めることができるため、カンマ区切りリストを指定すると、リストは単一のデータベースの名前として扱われます。

ステートメントベースレプリケーションの使用時に予想どおりに機能しないことの例: レプリカが `--replicate-do-db=sales` で起動され、ソースで次のステートメントを発行した場合、`UPDATE` ステートメントはレプリケートされません:

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

この「デフォルトデータベースだけをチェックする」動作の主な理由は、ステートメントだけから複製すべきかどうかを知るのが難しいためです (たとえば、複数のデータベースをまたがって動作する複数テーブル `DELETE` ステートメントまたは `UPDATE` ステートメントを使用する場合)。また、必要がない場合、すべてのデータベースではなくデフォルトデータベースだけをチェックする方が早いです。

行ベースのレプリケーション。レプリケーションをデータベース `db_name` に制限するようにレプリケーション SQL スレッドに指示します。 `db_name` に属するテーブルだけが変更されます。現在のデータベースはこれに影響しません。レプリカが `--replicate-do-db=sales` で開始され、行ベースのレプリケーションが有効になっているとします。その後、次のステートメントがソースで実行されます:

```
USE prices;
UPDATE sales.february SET amount=amount+100;
```

レプリカ上の `sales` データベース内の `february` テーブルは、`UPDATE` ステートメントに従って変更されます。これは、`USE` ステートメントが発行されたかどうかに関係なく発生します。ただし、行ベースのレプリケーションおよび `--replicate-do-db=sales` を使用している場合、ソースで次のステートメントを発行してもレプリカには影響しません:

```
USE prices;
UPDATE prices.march SET amount=amount-25;
```

ステートメント `USE prices` が `USE sales` に変更された場合でも、`UPDATE` ステートメントの結果は複製されません。

`--replicate-do-db` が行ベースレプリケーションとステートメントベースレプリケーションでどのように扱われるかについても 1 つ重要な違いは、複数のデータベースを参照するステートメントで発生します。レプリカが `--replicate-do-db=db1` で起動され、次のステートメントがソースで実行されるとします:

```
USE db1;
UPDATE db1.table1, db2.table2 SET db1.table1.col1 = 10, db2.table2.col2 = 20;
```

ステートメントベースレプリケーションを使用している場合は、両方のテーブルがレプリカで更新されます。ただし、行ベースのレプリケーションを使用している場合、レプリカに対する影響を受けるのは `table1` のみです。 `table2` は別のデータベースにあるため、レプリカ上の `table2` は `UPDATE` によって変更されません。ここで、`USE db1` ステートメントの代わりに、`USE db4` ステートメントが使用されたものとします。

```
USE db4;
UPDATE db1.table1, db2.table2 SET db1.table1.col1 = 10, db2.table2.col2 = 20;
```

この場合、ステートメントベースのレプリケーションを使用しても、`UPDATE` ステートメントはレプリカに影響しません。ただし、行ベースのレプリケーションを使用している場合、`UPDATE` はレプリカ上の `table1` を変更しま

すが、`table2` は変更しません。つまり、`--replicate-do-db` によって指定されたデータベース内のテーブルのみが変更され、デフォルトデータベースの選択はこの動作に影響しません。

クロスデータベース更新を機能させる必要がある場合は、代わりに `--replicate-wild-do-table=db_name.%` を使用してください。セクション17.2.5「サーバーがレプリケーションフィルタリングルールをどのように評価するか」を参照してください。

注記

このオプションは、`--binlog-do-db` がバイナリロギングに影響するのと同じ方法でレプリケーションに影響し、`--replicate-do-db` がレプリケーション動作にどのように影響するかに対してレプリケーション形式がどのように影響するかは、`--binlog-do-db` 動作に対してロギング形式がどのように影響するかと同じです。

このオプションは、`BEGIN`、`COMMIT`、または `ROLLBACK` ステートメントに影響しません。

- `--replicate-ignore-db=db_name`

コマンド行形式	<code>--replicate-ignore-db=name</code>
型	文字列

データベースの名前を使用してレプリケーションフィルタを作成します。このようなフィルタは、`CHANGE REPLICATION FILTER REPLICATE_IGNORE_DB` を使用しても作成できます。

このオプションでは、チャンネル固有のレプリケーションフィルタがサポートされているため、マルチソースレプリカは異なるソースに対して特定のフィルタを使用できます。`channel_1` という名前のチャンネルでチャンネル固有のレプリケーションフィルタを構成するには、`--replicate-ignore-db:channel_1 :db_name` を使用します。この場合、最初のコロンはセパレーターとして解釈され、後続のコロンはリテラルコロンです。詳しくはセクション17.2.5.4「レプリケーションチャンネルベースのフィルタ」をご覧ください。

注記

グループレプリケーション用に構成された MySQL サーバーインスタンスでは、グローバルレプリケーションフィルタを使用できません。これは、一部のサーバーでトランザクションをフィルタすると、グループが一貫性のある状態で承諾に到達できなくなるためです。グループメンバーがグループ外のソースへのレプリカとしても機能する場合など、グループレプリケーションに直接関与しないレプリケーションチャンネルでチャンネル固有のレプリケーションフィルタを使用できます。`group_replication_applier` または `group_replication_recovery` チャンネルでは使用できません。

無視するデータベースを複数指定するには、このオプションを複数回 (データベースごとに 1 回) 使用します。データベース名にはカンマを含めることができるため、カンマ区切りリストを指定すると、単一のデータベースの名前として扱われます。

`--replicate-do-db` と同様に、このフィルタリングの正確な効果は、ステートメントベースと行ベースのどちらのレプリケーションが使用されているかによって異なり、次のいくつかの段落で説明します。

ステートメントベースのレプリケーション。 デフォルトデータベース (つまり、`USE` によって選択されたデータベース) が `db_name` であるステートメントをレプリケートしないようにレプリケーション SQL スレッドに指示します。

行ベースのレプリケーション。 データベース `db_name` 内のテーブルを更新しないようにレプリケーション SQL スレッドに指示します。デフォルトデータベースは影響しません。

ステートメントベースレプリケーションを使用する場合、次の例は予期したとおりに機能しません。レプリカが `--replicate-ignore-db=sales` で起動され、ソースで次のステートメントを発行するとします:

```
USE prices;
```

```
UPDATE sales.january SET amount=amount+1000;
```

このような場合 `UPDATE` ステートメントは複製されます。 `--replicate-ignore-db` が (`USE` ステートメントで指定された) デフォルトデータベースにのみ適用されるためです。 `sales` データベースがステートメントで明示的に指定されたため、ステートメントはフィルタされませんでした。ただし、行ベースのレプリケーションを使用している場合、`UPDATE` ステートメントの効果はレプリカに伝播されず、`sales.january` テーブルのレプリカコピーは変更されません。この場合、`--replicate-ignore-db=sales` によって `sales` データベースのソースコピーのテーブルに対して行われた `all` の変更はレプリカによって無視されます。

クロスデータベース更新を使用していて、これらの更新を複製したくない場合は、このオプションを使用しないでください。 [セクション17.2.5「サーバーがレプリケーションフィルタリングルールをどのように評価するか」](#) を参照してください。

クロスデータベース更新を機能させる必要がある場合、代わりに `--replicate-wild-ignore-table=db_name.%` を使用してください。 [セクション17.2.5「サーバーがレプリケーションフィルタリングルールをどのように評価するか」](#) を参照してください。

注記

このオプションは、`--binlog-ignore-db` がバイナリロギングに影響するのと同じ方法でレプリケーションに影響し、`--replicate-ignore-db` がレプリケーション動作にどのように影響するかに対してレプリケーション形式がどのように影響するかは、`--binlog-ignore-db` 動作に対してロギング形式がどのように影響するかと同じです。

このオプションは、`BEGIN`、`COMMIT`、または `ROLLBACK` ステートメントに影響しません。

- `--replicate-do-table=db_name.tbl_name`

コマンド行形式	<code>--replicate-do-table=name</code>
型	文字列

レプリケーションを特定のテーブルに制限するようにレプリケーション SQL スレッドに指示することで、レプリケーションフィルタを作成します。複数のテーブルを指定するには、このオプションを複数回 (テーブルごとに 1 回) 使用します。 `--replicate-do-db` とは対照的に、これはクロスデータベース更新とデフォルトデータベース更新の両方に機能します。 [セクション17.2.5「サーバーがレプリケーションフィルタリングルールをどのように評価するか」](#) を参照してください。このようなフィルタは、`CHANGE REPLICATION FILTER REPLICATE_DO_TABLE` ステートメントを発行して作成することもできます。

このオプションでは、チャンネル固有のレプリケーションフィルタがサポートされているため、マルチソースレプリカは異なるソースに対して特定のフィルタを使用できます。 `channel_1` という名前のチャンネルでチャンネル固有のレプリケーションフィルタを構成するには、`--replicate-do-table:channel_1 :db_name.tbl_name` を使用します。この場合、最初のコロンはセパレータとして解釈され、後続のコロンはリテラルコロンです。詳しくは [セクション17.2.5.4「レプリケーションチャンネルベースのフィルタ」](#) をご覧ください。

注記

グループレプリケーション用に構成された MySQL サーバーインスタンスでは、グローバルレプリケーションフィルタを使用できません。これは、一部のサーバーでトランザクションをフィルタすると、グループが一貫性のある状態で承諾に到達できなくなるためです。グループメンバーがグループ外のソースへのレプリカとしても機能する場合など、グループレプリケーションに直接関与しないレプリケーションチャンネルでチャンネル固有のレプリケーションフィルタを使用できます。 `group_replication_applier` または `group_replication_recovery` チャンネルでは使用できません。

このオプションは、テーブルに適用されるステートメントにのみ影響します。ストアドルーチンなど、ほかのデータベースオブジェクトにのみ適用されるステートメントには影響しません。ストアドルーチンに作用するステートメントをフィルタするには、1 つまたは複数の `--replicate-*-db` オプションを使用します。

- `--replicate-ignore-table=db_name.tbl_name`

コマンド行形式	<code>--replicate-ignore-table=name</code>
型	文字列

同じステートメントによってほかのテーブルが更新される可能性がある場合でも、指定されたテーブルを更新するステートメントをレプリケートしないようにレプリケーション SQL スレッドに指示することによって、レプリケーションフィルタを作成します。無視するテーブルを複数指定するには、このオプションを複数回 (テーブルごとに 1 回) 使用します。 `--replicate-ignore-db` とは対照的に、これはクロスデータベース更新に機能します。 [セクション 17.2.5 「サーバーがレプリケーションフィルタリングルールをどのように評価するか」](#) を参照してください。このようなフィルタは、 `CHANGE REPLICATION FILTER REPLICATE_IGNORE_TABLE` ステートメントを発行して作成することもできます。

このオプションでは、チャンネル固有のレプリケーションフィルタがサポートされているため、マルチソースレプリカは異なるソースに対して特定のフィルタを使用できます。 `channel_1` という名前のチャンネルでチャンネル固有のレプリケーションフィルタを構成するには、 `--replicate-ignore-table:channel_1:db_name.tbl_name` を使用します。この場合、最初のコロンはセパレータとして解釈され、後続のコロンはリテラルコロンです。詳しくは [セクション 17.2.5.4 「レプリケーションチャンネルベースのフィルタ」](#) をご覧ください。

注記

グループレプリケーション用に構成された MySQL サーバーインスタンスでは、グローバルレプリケーションフィルタを使用できません。これは、一部のサーバーでトランザクションをフィルタすると、グループが一貫性のある状態で承諾に到達できなくなるためです。グループメンバーがグループ外のソースへのレプリカとしても機能する場合など、グループレプリケーションに直接関与しないレプリケーションチャンネルでチャンネル固有のレプリケーションフィルタを使用できます。 `group_replication_applier` または `group_replication_recovery` チャンネルでは使用できません。

このオプションは、テーブルに適用されるステートメントにのみ影響します。ストアルーチンなど、ほかのデータベースオブジェクトにのみ適用されるステートメントには影響しません。ストアルーチンに作用するステートメントをフィルタするには、1 つまたは複数の `--replicate-*db` オプションを使用します。

- `--replicate-rewrite-db=from_name->to_name`

コマンド行形式	<code>--replicate-rewrite-db=old_name->new_name</code>
型	文字列

指定されたデータベースがソース上の `from_name` である場合に、それを `to_name` に変換するレプリケーションフィルタを作成するようレプリカに指示します。影響を受けるのはテーブルを含むステートメントのみで、 `CREATE DATABASE`、 `DROP DATABASE`、 `ALTER DATABASE` などのステートメントは影響を受けません。

複数の書き換えを指定するには、複数回のこのオプションを使用します。サーバーは、一致する `from_name` 値で最初のものを使用します。データベース名変換は、 `--replicate-*` ルールがテストされる前に行われます。このようなフィルタは、 `CHANGE REPLICATION FILTER REPLICATE_REWRITE_DB` ステートメントを発行して作成することもできます。

コマンドラインで `--replicate-rewrite-db` オプションを使用し、 `>` 文字がコマンドインタプリタに特殊な場合は、オプション値を引用符で囲みます。例:

```
shell> mysql --replicate-rewrite-db="olddb->newdb"
```

`--replicate-rewrite-db` オプションの効果は、ステートメントベースと行ベースのどちらのバイナリロギング形式をクエリーに使用するかによって異なります。ステートメントベースの形式では、DML ステートメントは、 `USE` ステートメントで指定された現行のデータベースに基づいて変換されます。行ベースの形式では、DML ステートメントは、変更されたテーブルが存在するデータベースに基づいて変換されます。DDL ステートメントは、バイナリロ

ギング形式に関係なく、常に `USE` ステートメントで指定された現在のデータベースに基づいてフィルタ処理されま

す。
 リライトによって予期した結果が得られるようにするには、特に他のレプリケーションフィルタリングオプションと組み合わせて、`--replicate-rewrite-db` オプションを使用する際に次の推奨事項に従います:

- ソースおよびレプリカに異なる名前 `from_name` および `to_name` データベースを手動で作成します。
- ステートメントベースまたは混合バイナリロギング形式を使用する場合は、クロスデータベースクエリーを使用せず、クエリーでデータベース名を指定しないでください。DDL ステートメントと DML ステートメントの両方で、`USE` ステートメントを使用して現在のデータベースを指定し、クエリーでテーブル名のみを使用します。
- DDL ステートメントで行ベースのバイナリロギング形式を排他的に使用する場合は、`USE` ステートメントを使用して現在のデータベースを指定し、クエリーでテーブル名のみを使用します。DML ステートメントには、完全修飾テーブル名 (`db`) を使用できます。 `table`) (必要な場合)。

これらの推奨事項に従う場合は、`--replicate-rewrite-db` オプションを `--replicate-do-table` などのテーブルレベルのレプリケーションフィルタリングオプションと組み合わせて使用しても安全です。

このオプションでは、チャンネル固有のレプリケーションフィルタがサポートされているため、マルチソースレプリカは異なるソースに対して特定のフィルタを使用できます。チャンネル名に続けてコロンを指定し、その後にフィルタ指定を指定します。最初のコロンはセパレータとして解釈され、後続のコロンはリテラルコロンとして解釈されます。たとえば、`channel_1` という名前のチャンネルでチャンネル固有のレプリケーションフィルタを構成するには、次を使用します:

```
shell> mysqld --replicate-rewrite-db=channel_1:db_name1->db_name2
```

コロンを使用してチャンネル名を指定しない場合、このオプションはデフォルトのレプリケーションチャンネルのレプリケーションフィルタを構成します。詳しくは [セクション 17.2.5.4 「レプリケーションチャンネルベースのフィルタ」](#) をご覧ください。

注記

グループレプリケーション用に構成された MySQL サーバーインスタンスでは、グローバルレプリケーションフィルタを使用できません。これは、一部のサーバーでトランザクションをフィルタすると、グループが一貫性のある状態で承諾に到達できなくなるためです。グループメンバーがグループ外のソースへのレプリカとしても機能する場合など、グループレプリケーションに直接関与しないレプリケーションチャンネルでチャンネル固有のレプリケーションフィルタを使用できます。 `group_replication_applier` または `group_replication_recovery` チャンネルでは使用できません。

- `--replicate-same-server-id`

コマンド行形式	<code>--replicate-same-server-id[={OFF ON}]</code>
型	Boolean
デフォルト値	OFF

このオプションはレプリカで使用します。デフォルトは 0 (`FALSE`) です。このオプションを 1 (`TRUE`) に設定すると、レプリカは独自のサーバー ID を持つイベントをスキップしません。通常、この設定はまれな構成でのみ役立ちます。

レプリカでバイナリロギングが有効になっている場合、サーバーが循環レプリケーショントポロジの一部であると、レプリカ上の `--replicate-same-server-id` オプションと `--log-slave-updates` オプションの組み合わせによってレプリケーションで無限ループが発生する可能性があります。(MySQL 8.0 では、バイナリロギングはデフォルトで有効になっており、バイナリロギングが有効になっている場合はレプリカ更新ロギングがデフォルトになります。) ただし、グローバルトランザクション識別子 (GTID) を使用すると、すでに適用されているトランザクションの実行がスキップされ、この状況が回避されます。レプリカに `gtid_mode=ON` が設定されている場合、このオプションの

組合せでサーバーを起動できますが、サーバーの実行中は他の GTID モードに変更できません。ほかの GTID モードが設定されている場合、サーバーはこのオプションの組み合わせで起動しません。

デフォルトでは、複製サーバー ID を持っている場合、レプリケーション I/O スレッドはバイナリロギングイベントをリレーログに書き込みません (この最適化はディスク使用量の節約に役立ちます)。`--replicate-same-server-id` を使用する場合は、レプリケーション SQL スレッドで実行する独自のイベントをレプリカで読み取る前に、必ずこのオプションを使用してレプリカを起動してください。

- `--replicate-wild-do-table=db_name.tbl_name`

コマンド行形式	<code>--replicate-wild-do-table=name</code>
型	文字列

レプリケーション SQL スレッドに、更新されたテーブルのいずれかが指定されたデータベースおよびテーブル名パターンと一致するステートメントにレプリケーションを制限するように指示することで、レプリケーションフィルタを作成します。パターンには、LIKE パターン一致演算子と同じ意味を持つ `%` および `_` ワイルドカード文字を含めることができます。複数のテーブルを指定するには、このオプションを複数回 (テーブルごとに 1 回) 使用します。これはクロスデータベース更新に役立ちます。セクション 17.2.5 「サーバーがレプリケーションフィルタリングルールをどのように評価するか」を参照してください。このようなフィルタは、`CHANGE REPLICATION FILTER REPLICATE_WILD_DO_TABLE` ステートメントを発行して作成することもできます。

このオプションでは、チャンネル固有のレプリケーションフィルタがサポートされているため、マルチソースレプリカは異なるソースに対して特定のフィルタを使用できます。`channel_1` という名前のチャンネルでチャンネル固有のレプリケーションフィルタを構成するには、`--replicate-wild-do-table:channel_1:db_name.tbl_name` を使用します。この場合、最初のコロンはセパレータとして解釈され、後続のコロンはリテラルコロンです。詳しくはセクション 17.2.5.4 「レプリケーションチャンネルベースのフィルタ」をご覧ください。

注記

グループレプリケーション用に構成された MySQL サーバーインスタンスでは、グローバルレプリケーションフィルタを使用できません。これは、一部のサーバーでトランザクションをフィルタすると、グループが一貫性のある状態で承諾に到達できなくなるためです。グループメンバーがグループ外のソースへのレプリカとしても機能する場合など、グループレプリケーションに直接関与しないレプリケーションチャンネルでチャンネル固有のレプリケーションフィルタを使用できます。`group_replication_applier` または `group_replication_recovery` チャンネルでは使用できません。

このオプションはテーブル、ビュー、およびトリガーに適用されます。ストアードプロシージャと関数、またはイベントには適用されません。後者のオブジェクトで作用するステートメントをフィルタするには、1 つまたは複数の `--replicate-*-db` オプションを使用します。

たとえば、`--replicate-wild-do-table=foo%.bar%` は、データベース名が `foo` で始まり、テーブル名が `bar` で始まるテーブルを使用する更新のみをレプリケートします。

テーブル名パターンが `%` の場合、それは任意のテーブル名に一致し、このオプションはデータベースレベルステートメント (`CREATE DATABASE`、`DROP DATABASE`、および `ALTER DATABASE`) にも適用されます。たとえば、`--replicate-wild-do-table=foo%.%` を使用する場合には、データベース名がパターン `foo%` に一致する場合はデータベースレベルステートメントが複製されます。

リテラルワイルドカード文字をデータベースまたはテーブル名パターンに含めるには、バックスラッシュでそれらをエスケープします。たとえば、`my_own%db` という名前のデータベースのすべてのテーブルをレプリケートし、`my1ownAABCdb` データベースからはレプリケートしない場合は、次のように `_` および `%` 文字をエスケープする必要があります: `--replicate-wild-do-table=my_own%db`。このオプションをコマンド行で使用する場合は、コマンドインタープリターによっては、バックスラッシュを二重にしたりオプション値を引用符で囲んだりする必要があります。たとえば、`bash` シェルでは、`--replicate-wild-do-table=my_own%db` と入力する必要があります。

- `--replicate-wild-ignore-table=db_name.tbl_name`

コマンド行形式	<code>--replicate-wild-ignore-table=name</code>
---------	---

型	文字列
---	-----

レプリケーション SQL スレッドが、任意のテーブルが指定されたワイルドカードパターンと一致するステートメントをレプリケートしないようにするレプリケーションフィルタを作成します。無視するテーブルを複数指定するには、このオプションを複数回 (テーブルごとに 1 回) 使用します。これはクロスデータベース更新に役立ちます。[セクション 17.2.5 「サーバーがレプリケーションフィルタリングルールをどのように評価するか」](#)を参照してください。このようなフィルタは、`CHANGE REPLICATION FILTER REPLICATE_WILD_IGNORE_TABLE` ステートメントを発行して作成することもできます。

このオプションでは、チャンネル固有のレプリケーションフィルタがサポートされているため、マルチソースレプリカは異なるソースに対して特定のフィルタを使用できます。`channel_1` という名前のチャンネルでチャンネル固有のレプリケーションフィルタを構成するには、`--replicate-wild-ignore:channel_1 :db_name.tbl_name` を使用します。この場合、最初のコロンはセパレータとして解釈され、後続のコロンはリテラルコロンです。詳しくは[セクション 17.2.5.4 「レプリケーションチャンネルベースのフィルタ」](#)をご覧ください。

注記

グループレプリケーション用に構成された MySQL サーバーインスタンスでは、グローバルレプリケーションフィルタを使用できません。これは、一部のサーバーでトランザクションをフィルタすると、グループが一貫性のある状態で承諾に到達できなくなるためです。グループメンバーがグループ外のソースへのレプリカとしても機能する場合など、グループレプリケーションに直接関与しないレプリケーションチャンネルでチャンネル固有のレプリケーションフィルタを使用できます。`group_replication_applier` または `group_replication_recovery` チャンネルでは使用できません。

たとえば、`--replicate-wild-ignore-table=foo%.bar%` では、データベース名が `foo` で始まり、テーブル名が `bar` で始まるテーブルを使用する更新はレプリケートされません。照合の仕組みについては、`--replicate-wild-do-table` オプションの説明を参照してください。オプション値にリテラルワイルドカード文字を含めるためのルールは、`--replicate-wild-ignore-table` 場合と同じです。

- `--skip-slave-start`

コマンド行形式	<code>--skip-slave-start[={OFF ON}]</code>
型	Boolean
デフォルト値	OFF

サーバーの起動時にレプリケーション I/O および SQL スレッドを起動しないようにレプリカサーバーに指示します。あとでスレッドを起動するには、`START REPLICA | SLAVE` ステートメントを使用します。

- `--slave-skip-errors=[err_code1,err_code2,...|all|ddl_exist_errors]`

コマンド行形式	<code>--slave-skip-errors=name</code>
システム変数	<code>slave_skip_errors</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	OFF
有効な値	OFF [list of error codes] all

ddl_exist_errors

通常、レプリケーションはレプリカでエラーが発生すると停止するため、データの非一貫性を手動で解決できません。このオプションを指定すると、ステートメントがオプション値にリストされているエラーのいずれかを返したときに、レプリケーション SQL スレッドはレプリケーションを続行します。

このオプションは、エラーが発生している理由を完全に理解しないかぎり使用しないでください。レプリケーションセットアップとクライアントプログラムにバグがなく、MySQL 自体にバグがない場合は、レプリケーションを停止するエラーは発生しないはずですが、これが発生した理由はわかりません。

エラーコードの場合は、レプリカエラーログおよび `SHOW REPLICA | SLAVE STATUS` の出力のエラーメッセージに示されている番号を使用する必要があります。付録B「エラーメッセージと一般的な問題」には、サーバーエラーコードがリストされます。

短縮値 `ddl_exist_errors` は、エラーコードリスト `1007,1008,1050,1051,1054,1060,1061,1068,1094,1146` と同等です。

また、`all` の推奨されない値を使用して、レプリカがすべてのエラーメッセージを無視し、何が発生したかに関係なく処理を続行するようにすることもできます（ただし、推奨されません）。言うまでもなく、`all` を使用した場合、データの完全性に関して保証はありません。この場合、レプリカデータがソース上のどこにも近い場所がない場合は、苦情（またはバグレポートをファイル）しないでください。以上のことを警告しました。

例:

```
--slave-skip-errors=1062,1053
--slave-skip-errors=all
--slave-skip-errors=ddl_exist_errors
```

- `--slave-sql-verify-checksum={0|1}`

コマンド行形式	<code>--slave-sql-verify-checksum[={OFF ON}]</code>
型	Boolean
デフォルト値	ON

このオプションを有効にすると、レプリカはリレーログから読み取られたチェックサムを調べます。不一致が発生した場合、レプリカはエラーで停止します。

次のオプションは、レプリケーションテストおよびデバッグのために MySQL テストスイートによって内部的に使用されます。本番設定での使用は意図していません。

- `--abort-slave-event-count`

コマンド行形式	<code>--abort-slave-event-count=#</code>
型	Integer
デフォルト値	0
最小値	0

このオプションを 0（デフォルト）以外の正の整数 `value` に設定すると、次のようにレプリケーションの動作に影響: レプリケーション SQL スレッドが開始されると、`value` ログイベントの実行が許可されます。その後、レプリケーション SQL スレッドは、ソースからのネットワーク接続が切断された場合と同様に、これ以上イベントを受信しません。レプリケーション SQL スレッドは引き続き実行され、`SHOW REPLICA | SLAVE STATUS` からの出力では、`Replica_IO_Running` と `Replica_SQL_Running` の両方のカラムに `Yes` が表示されますが、リレーログからそれ以上のイベントは読み取られません。

- `--disconnect-slave-event-count`

コマンド行形式	<code>--disconnect-slave-event-count=#</code>
型	Integer

デフォルト値	0
--------	---

レプリカサーバーで使用されるシステム変数

次のリストでは、レプリカサーバーを制御するためのシステム変数について説明します。これらはサーバー起動時に設定でき、それらの一部は `SET` を使用して実行時に変更できます。レプリカで使用されるサーバーオプションは、このセクションで前述しました。

- `init_slave`

コマンド行形式	<code>--init-slave=name</code>
システム変数	<code>init_slave</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列

この変数は `init_connect` と似ていますが、レプリケーション SQL スレッドが開始されるたびにレプリカサーバーによって実行される文字列です。文字列の形式は `init_connect` 変数の場合と同じです。この変数の設定は、後続の `START REPLICIA | SLAVE` ステートメントに対して有効になります。

注記

レプリケーション SQL スレッドは、`init_slave` を実行する前にクライアントに確認を送信します。したがって、`START REPLICIA | SLAVE` が戻ったときに `init_slave` が実行されていることは保証されていません。詳しくは [セクション13.4.2.7「START REPLICIA | SLAVE ステートメント」](#) をご覧ください。

- `log_slow_slave_statements`

コマンド行形式	<code>--log-slow-slave-statements[={OFF ON}]</code>
システム変数	<code>log_slow_slave_statements</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

スロークエリーログが有効になっている場合、この変数は、レプリカでの実行に `long_query_time` 秒を超える時間がかかったクエリーのロギングを有効にします。行ベースのレプリケーションが使用されている (`binlog_format=ROW`) 場合、`log_slow_slave_statements` は効果がないことに注意してください。クエリーがレプリカのスロークエリーログに追加されるのは、バイナリログにステートメント形式で記録されている場合、つまり `binlog_format=STATEMENT` が設定されている場合、または `binlog_format=MIXED` が設定されていてステートメントがステートメント形式で記録されている場合だけです。 `binlog_format=MIXED` の設定時に行形式でログに記録されるスロークエリー、または `binlog_format=ROW` の設定時にログに記録されるスロークエリーは、`log_slow_slave_statements` が有効な場合でもレプリカのスロークエリーログに追加されません。

`log_slow_slave_statements` を設定しても、すぐには影響しません。変数の状態は、後続のすべての `START REPLICIA | SLAVE` ステートメントに適用されます。また、`long_query_time` のグローバル設定は、SQL スレッドの存続期間中に適用されることに注意してください。この設定を変更する場合は、レプリケーション SQL スレッドを停止して再起動し、そこで変更を実装する必要があります (たとえば、`SQL_THREAD` オプションを指定して `STOP REPLICIA | SLAVE` および `START REPLICIA | SLAVE` ステートメントを発行します)。

- `master_info_repository`

コマンド行形式	<code>--master-info-repository={FILE TABLE}</code>
---------	--

非推奨	8.0.23
システム変数	master_info_repository
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	TABLE
有効な値	FILE TABLE

このシステム変数の使用は非推奨になりました。TABLE の設定がデフォルトであり、複数のレプリケーションチャンネルが構成されている場合は必須です。代替設定の FILE は、以前は非推奨でした。

デフォルト設定では、レプリカは、ステータスおよび接続情報で構成されるソースに関するメタデータを、`mysql.slave_master_info` という名前の `mysql` システムデータベースの InnoDB テーブルに記録します。接続メタデータリポジトリの詳細は、[セクション17.2.4 「リレーログおよびレプリケーションメタデータリポジトリ」](#) を参照してください。

FILE 設定では、レプリカ接続メタデータリポジトリがファイルに書き込まれましたが、これはデフォルトで `master.info` という名前でした。この名前は、`--master-info-file` オプションを使用して変更できます。

- [max_relay_log_size](#)

コマンド行形式	<code>--max-relay-log-size=#</code>
システム変数	max_relay_log_size
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	1073741824

レプリカによるリレーログへの書き込みによって、現在のログファイルサイズがこの変数の値を超える場合、レプリカはリレーログをローテーションします (現在のファイルを閉じて次のファイルを開きます)。[max_relay_log_size](#) が 0 の場合、サーバーはバイナリログとリレーログの両方に [max_binlog_size](#) を使用します。[max_relay_log_size](#) が 0 より大きい場合、リレーログのサイズを抑制し、2 つのログに異なるサイズを持たせることが可能になります。[max_relay_log_size](#) を 4096 バイトと 1G バイト (両端の値を含む) の間に設定するか、0 にする必要があります。デフォルト値は 0 です。[セクション17.2.3 「レプリケーションスレッド」](#) を参照してください。

- [relay_log](#)

コマンド行形式	<code>--relay-log=file_name</code>
システム変数	relay_log
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ファイル名

リレーログファイルのベース名。デフォルトのレプリケーションチャンネルの場合、リレーログのデフォルトのベース名は `host_name-relay-bin` です。デフォルト以外のレプリケーションチャンネルの場合、リレーログのデフォルト

のベース名は `host_name-relay-bin-channel` です。ここで、`channel` は、このリレーログに記録されているレプリケーションチャンネルの名前です。

ベース名の先頭に絶対パス名を付けて別のディレクトリを指定しないかぎり、サーバーはファイルをデータディレクトリに書き込みます。サーバーは、ベース名に数値の接尾辞を追加することによって、リレーログファイルを順番に作成します。

レプリケーションサーバーのリレーログおよびリレーログインデックスには、バイナリログおよびバイナリログインデックスと同じ名前を付けることはできません。バイナリログおよびバイナリログインデックスの名前は、`--log-bin` および `--log-bin-index` オプションで指定されます。バイナリログとリレーログファイルのベース名が同じであれば、サーバーはエラーメッセージを発行し、起動しません。

MySQL がサーバーオプションを解析する方法のため、サーバーの起動時にこの変数を指定する場合は、デフォルトのベース名は、オプションが実際に指定されていない場合にのみ使用されますという値を指定する必要があります。サーバーの起動時に値を指定せずに `relay_log` システム変数を指定すると、予期しない動作が発生する可能性があります。この動作は、使用される他のオプション、それらが指定されている順序、およびそれらがコマンド行とオプションファイルのどちらで指定されているかによって異なります。MySQL がサーバーオプションをどのように処理するかについて詳しくは、[セクション4.2.2「プログラムオプションの指定」](#)を参照してください。

この変数を指定すると、指定した値がリレーログインデックスファイルのベース名としても使用されます。この動作をオーバーライドするには、`relay_log_index` システム変数を使用して別のリレーログインデックスファイルのベース名を指定します。

サーバーは、インデックスファイルからエントリを読み取るときに、エントリに相対パスが含まれているかどうかをチェックします。その場合、パスの相対部分は、`relay_log` システム変数を使用して設定された絶対パスに置き換えられます。絶対パスは変わりません。このような場合、使用される新しいパスを有効にするために、インデックスを手動で編集する必要があります。

`relay_log` システム変数は、次のタスクの実行に役立つ場合があります：

- 名前がホスト名に依存しないリレーログを作成する。
- リレーログが非常に大きくなる傾向があり、`max_relay_log_size` を小さくしたくないため、リレーログをデータディレクトリ以外の領域に置く必要がある場合。
- ディスク間のロードバランシングを使用して速度を上げるため。

リレーログファイル名 (およびパス) は、`relay_log_basename` システム変数から取得できます。

• `relay_log_basename`

システム変数	<code>relay_log_basename</code>
スコープ	グローバル
動的	いいえ
<code>SET_VAR</code> ヒントの適用	いいえ
型	ファイル名
デフォルト値	<code>datadir + '/' + hostname + '-relay-bin'</code>

リレーログファイルのベース名と完全パスを保持します。最大可変長は 256 です。この変数はサーバーによって設定され、読取り専用です。

• `relay_log_index`

コマンド行形式	<code>--relay-log-index=file_name</code>
システム変数	<code>relay_log_index</code>
スコープ	グローバル
動的	いいえ
<code>SET_VAR</code> ヒントの適用	いいえ

型	ファイル名
デフォルト値	*host_name*-relay-bin.index

リレーログインデックスファイルの名前。最大可変長は 256 です。この変数を指定しないが、`relay_log` システム変数が指定されている場合、リレーログインデックスファイルのデフォルトのベース名としてその値が使用されます。`relay_log` も指定されていない場合、デフォルトのレプリケーションチャンネルのデフォルト名は、ホストマシンの名前を使用した `host_name-relay-bin.index` です。デフォルト以外のレプリケーションチャンネルの場合、デフォルト名は `host_name-relay-bin-channel.index` で、`channel` はこのリレーログインデックスに記録されているレプリケーションチャンネルの名前です。

リレーログファイルのデフォルトの場所は、データディレクトリ、または `relay_log` システム変数を使用して指定されたその他の場所です。ベース名に先頭の絶対パスを追加して別のディレクトリを指定することで、`relay_log_index` システム変数を使用して別の場所を指定できます。

レプリケーションサーバーのリレーログおよびリレーログインデックスには、バイナリログおよびバイナリログインデックスと同じ名前を付けることはできません。バイナリログおよびバイナリログインデックスの名前は、`--log-bin` および `--log-bin-index` オプションで指定されます。バイナリログとリレーログファイルのベース名が同じであれば、サーバーはエラーメッセージを発行し、起動しません。

MySQL がサーバーオプションを解析する方法のため、サーバーの起動時にこの変数を指定する場合は、デフォルトのベース名は、オプションが実際に指定されていない場合にのみ使用されますという値を指定する必要があります。サーバーの起動時に値を指定せずに `relay_log_index` システム変数を指定すると、予期しない動作が発生する可能性があります。この動作は、使用される他のオプション、それらが指定されている順序、およびそれらがコマンド行とオプションファイルのどちらで指定されているかによって異なります。MySQL がサーバーオプションをどのように処理するかについて詳しくは、[セクション4.2.2「プログラムオプションの指定」](#)を参照してください。

- `relay_log_info_file`

コマンド行形式	<code>--relay-log-info-file=file_name</code>
非推奨	8.0.18
システム変数	<code>relay_log_info_file</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	<code>relay-log.info</code>

このシステム変数の使用は非推奨になりました。`relay_log_info_repository=FILE` が設定されている場合、レプリカアプライアンスメタデータリポジトリのファイル名を設定するために使用されていました。`relay_log_info_file` および `relay_log_info_repository` システム変数の使用は、アプライアンスメタデータリポジトリのファイルの使用がクラッシュセーフテーブルに置き換えられたため、非推奨になりました。適用者メタデータリポジトリの詳細は、[セクション17.2.4.2「レプリケーションメタデータリポジトリ」](#)を参照してください。

- `relay_log_info_repository`

コマンド行形式	<code>--relay-log-info-repository=value</code>
非推奨	8.0.23
システム変数	<code>relay_log_info_repository</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	<code>TABLE</code>
有効な値	<code>FILE</code>

TABLE

このシステム変数の使用は非推奨になりました。TABLE の設定がデフォルトであり、複数のレプリケーションチャンネルが構成されている場合は必須です。レプリカアプライアンスのメタデータリポジトリの TABLE 設定は、予期しない停止に対するレプリケーションの回復性を確保するためにも必要です。詳しくは [セクション17.4.2「レプリカの予期しない停止の処理」](#) をご覧ください。代替設定の FILE は、以前は非推奨でした。

デフォルト設定では、レプリカのアプライアンスメタデータリポジトリは、mysql.slave_relay_log_info という名前の mysql システムデータベースに InnoDB テーブルとして格納されます。適用者メタデータリポジトリの詳細は、[セクション17.2.4「リレーログおよびレプリケーションメタデータリポジトリ」](#) を参照してください。

FILE 設定では、レプリカアプライアンスメタデータリポジトリがファイルに書き込まれましたが、これはデフォルトで relay-log.info という名前でした。名前は、relay_log_info_file システム変数を使用して変更できます。

- [relay_log_purge](#)

コマンド行形式	--relay-log-purge[={OFF ON}]
システム変数	relay_log_purge
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

リレーログファイルが不要になったときに自動的にパージするよう無効または有効にします。デフォルト値は 1 (ON) です。

- [relay_log_recovery](#)

コマンド行形式	--relay-log-recovery[={OFF ON}]
システム変数	relay_log_recovery
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

この変数を有効にすると、サーバーの起動直後にリレーログリカバリが自動的に有効になります。リカバリプロセスでは、新しいリレーログファイルを作成し、SQL スレッド位置をこの新しいリレーログに初期化し、I/O スレッドを SQL スレッド位置に初期化します。その後、ソースからのリレーログの読み取りが続行されます。

このグローバル変数は、実行時に読み取り専用です。この値は、レプリカサーバーの起動時に --relay-log-recovery オプションを使用して設定できます。このオプションは、破損する可能性のあるリレーログが処理されないようにするために、レプリカの予期しない停止後に使用する必要があり、クラッシュセーフなレプリカを保証するために使用する必要があります。デフォルト値は 0 (無効) です。予期しない停止に対して最も回復可能なレプリカの設定の組合せの詳細は、[セクション17.4.2「レプリカの予期しない停止の処理」](#) を参照してください。

マルチスレッドレプリカ (slave_parallel_workers が 0 より大きい) の場合、起動時に --relay-log-recovery を設定すると、リレーログから実行された一連のトランザクションの不整合およびギャップが自動的に処理されます。これらのギャップは、ファイル位置ベースのレプリケーションが使用されている場合に発生することがあります。(詳細は、[セクション17.5.1.34「レプリケーションとトランザクションの非一貫性」](#) を参照してください。) リレーログリカバリプロセスは、START REPLICAS | SLAVE UNTIL SQL_AFTER_MTS_GAPS ステートメントと同じ方法を使用してギャップを処理します。レプリカが一貫性のないギャップのない状態に達すると、リレーログリカバリプロセスが進行し、SQL (適用者) スレッド位置からソースからさらにトランザクションがフェッチされます。GTID ベースのレプリケーションが使用されている場合、このプロセスは不要であり、MASTER_AUTO_POSITION が

ON に設定されている場合、MySQL 8.0.18 からマルチスレッドレプリカはリレーログのリカバリを自動的にスキップするため、`relay_log_recovery` の設定に違いはありません。

注記

この変数は、次のグループレプリケーションチャンネルには影響しません:

- `group_replication_applier`
- `group_replication_recovery`

外部ソースまたは別のグループからレプリケートしているチャンネルなど、グループで実行されている他のチャンネルは影響を受けます。

• `relay_log_space_limit`

コマンド行形式	<code>--relay-log-space-limit=#</code>
システム変数	<code>relay_log_space_limit</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値 (64 ビットプラットフォーム)	18446744073709551615
最大値 (32 ビットプラットフォーム)	4294967295

すべてのリレーログに使用するスペースの最大量。

• `replication_optimize_for_static_plugin_config`

コマンド行形式	<code>--replication-optimize-for-static-plugin-config[={OFF ON}]</code>
導入	8.0.23
システム変数	<code>replication_optimize_for_static_plugin_config</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

準同期レプリケーションのパフォーマンスを向上させるには、共有ロックを使用し、不要なロック取得を回避します。このシステム変数が有効になっている間は準同期レプリケーションプラグインをアンインストールできないため、アンインストールを完了する前にシステム変数を無効にする必要があります。

このシステム変数は、準同期レプリケーションプラグインのインストール前またはインストール後に有効にしたり、レプリケーションの実行中に有効にしたりできます。準同期レプリケーションソースサーバーも、複製と同じロックメカニズムを使用するため、このシステム変数を有効にすることでパフォーマンス上の利点を得ることができます。

• `replication_sender_observe_commit_only`

コマンド行形式	<code>--replication-sender-observe-commit-only[={OFF ON}]</code>
導入	8.0.23

システム変数	replication_sender_observe_commit_only
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

準同期レプリケーションのパフォーマンスを向上させるには、コールバックを制限します。このシステム変数は、準同期レプリケーションプラグインのインストール前またはインストール後に有効にしたり、レプリケーションの実行中に有効にしたりできます。準同期レプリケーションソースサーバーも、複製と同じロックメカニズムを使用するため、このシステム変数を有効にすることでパフォーマンス上の利点を得ることができます。

- [report_host](#)

コマンド行形式	--report-host=host_name
システム変数	report_host
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

レプリカの登録時にソースにレポートされるレプリカのホスト名または IP アドレス。この値は、ソースサーバーの [SHOW REPLICAS | SHOW SLAVE HOSTS](#) の出力に表示されます。レプリカ自体をソースに登録しない場合は、値を未設定のままにします。

注記

レプリカの接続後、ソースが TCP/IP ソケットからレプリカサーバーの IP アドレスを読み取るだけでは不十分です。NAT およびその他のルーティングの問題のため、その IP はソースまたは他のホストからレプリカへの接続に有効でない可能性があります。

- [report_password](#)

コマンド行形式	--report-password=name
システム変数	report_password
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

レプリカの登録時にソースにレポートされるレプリカのアカウントパスワード。この値は、ソースが [--show-slave-auth-info](#) で起動された場合に、ソースサーバー上の [SHOW REPLICAS | SHOW SLAVE HOSTS](#) の出力に表示されます。

この変数の名前は、それ以外の意味を持つ場合もありますが、[report_password](#) は MySQL ユーザー権限システムに接続されていないため、MySQL レプリケーションユーザーアカウントのパスワードと同じである必要はありません (または同じである可能性があります)。

- [report_port](#)

コマンド行形式	--report-port=port_num
システム変数	report_port
スコープ	グローバル

動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	[slave_port]
最小値	0
最大値	65535

レプリカ登録時にソースにレポートされる、レプリカに接続するための TCP/IP ポート番号。これは、レプリカがデフォルト以外のポートでリスニングしている場合、またはソースまたは他のクライアントからレプリカへの特別なトンネルがある場合にのみ設定します。確実にない場合は、このオプションを使用しないでください。

このオプションのデフォルト値は、レプリカによって実際に使用されるポート番号です。これは、[SHOW REPLICAS](#) | [SHOW SLAVE HOSTS](#) によって表示されるデフォルト値でもあります。

- [report_user](#)

コマンド行形式	--report-user=name
システム変数	report_user
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

レプリカの登録時にソースにレポートされるレプリカのアカウントユーザー名。この値は、ソースが [--show-slave-auth-info](#) で起動された場合に、ソースサーバー上の [SHOW REPLICAS](#) | [SHOW SLAVE HOSTS](#) の出力に表示されます。

この変数の名前はそれ以外を意味する場合がありますが、[report_user](#) は MySQL ユーザー権限システムに接続されていないため、必ずしも MySQL レプリケーションユーザーアカウントの名前と同じである必要はありません。

- [rpl_read_size](#)

コマンド行形式	--rpl-read-size=#
システム変数	rpl_read_size
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	8192
最小値	8192
最大値	4294967295

[rpl_read_size](#) システム変数は、バイナリログファイルおよびリレーログファイルから読み取られるデータの最小量をバイト単位で制御します。これらのファイルに対する大量のディスク I/O アクティビティがデータベースのパフォーマンスを低下させている場合、ファイルデータがオペレーティングシステムによって現在キャッシュされていないと、読み取りサイズを増やすとファイル読み取りが減少し、I/O が停止する可能性があります。

[rpl_read_size](#) の最小値およびデフォルト値は 8192 バイトです。値は 4KB の倍数である必要があります。バイナリログおよびリレーログファイルから読み取るスレッドごとに、この値のバッファが割り当てられます。これには、ソース上のダンプスレッドやレプリカ上のコーディネータスレッドも含まれます。したがって、大きな値を設定すると、サーバーのメモリー消費に影響する可能性があります。

- [rpl_semi_sync_slave_enabled](#)

コマンド行形式	<code>--rpl-semi-sync-slave-enabled[={OFF ON}]</code>
システム変数	rpl_semi_sync_slave_enabled
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

レプリカサーバーで準同期レプリケーションを有効にするかどうかを制御します。プラグインを有効または無効にするには、この変数を ON または OFF (あるいは 1 または 0) にそれぞれ設定します。デフォルトは OFF です。

この変数は、レプリカ側の準同期レプリケーションプラグインがインストールされている場合にのみ使用できません。

- [rpl_semi_sync_slave_trace_level](#)

コマンド行形式	<code>--rpl-semi-sync-slave-trace-level=#</code>
システム変数	rpl_semi_sync_slave_trace_level
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	32

レプリカサーバー上の準同期レプリケーションのデバッグトレースレベル。許可できる値については、[rpl_semi_sync_master_trace_level](#) を参照してください。

この変数は、レプリカ側の準同期レプリケーションプラグインがインストールされている場合にのみ使用できません。

- [rpl_stop_slave_timeout](#)

コマンド行形式	<code>--rpl-stop-slave-timeout=seconds</code>
システム変数	rpl_stop_slave_timeout
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	31536000
最小値	2

最大値	31536000
-----	----------

この変数を設定することで、[STOP REPLICA | SLAVE](#) がタイムアウトするまで待機する時間 (秒) を制御できます。これを使用すると、レプリカへの異なるクライアント接続を使用する [STOP REPLICA | SLAVE](#) と他の SQL ステートメントの間のデッドロックを回避できます。

[rpl_stop_slave_timeout](#) の最大値およびデフォルト値は 31536000 秒 (1 年) です。最小は 2 秒です。この変数への変更は、後続の [STOP REPLICA | SLAVE](#) ステートメントに対して有効になります。

この変数は、[STOP REPLICA | SLAVE](#) ステートメントを発行するクライアントにのみ影響します。タイムアウトに達すると、発行クライアントはコマンドの実行が不完全であることを示すエラーメッセージを返します。その後、クライアントはレプリケーション I/O および SQL スレッドの停止の待機を停止しますが、レプリケーションスレッドは引き続き停止しようとし、[STOP REPLICA | SLAVE](#) 命令は有効なままです。レプリケーションスレッドがビジー状態でなくなると、[STOP REPLICA | SLAVE](#) ステートメントが実行され、レプリカが停止します。

- [slave_checkpoint_group](#)

コマンド行形式	--slave-checkpoint-group=#
システム変数	slave_checkpoint_group
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	512
最小値	32
最大値	524280
ブロックサイズ	8

[SHOW REPLICA | SLAVE STATUS](#) で示されているように、チェックポイント操作がコールされてステータスが更新されるまでにマルチスレッドレプリカで処理できるトランザクションの最大数を設定します。この変数を設定しても、マルチスレッドが有効になっていないレプリカには影響しません。この変数を設定しても、すぐには影響しません。変数の状態は、後続のすべての [START REPLICA | SLAVE](#) コマンドに適用されます。

注記

マルチスレッドレプリカは現在 NDB Cluster でサポートされていないため、この変数の設定は暗黙的に無視されます。詳しくは[セクション23.6.3「NDB Cluster レプリケーションの既知の問題」](#)をご覧ください。

この変数は、[slave_checkpoint_period](#) システム変数との組み合わせで、どちらかの制限を超えたときに、チェックポイントが実行され、トランザクションの数と最後のチェックポイントから経過した時間の両方を追跡するカウンタがリセットされる、という方法で機能します。

この変数の最小許容値は 32 です (サーバーが [-DWITH_DEBUG](#) を使用して構築された場合を除く、この場合の最小値は 1)。効果的な値は常に 8 の倍数です。そのような倍数でない値に設定することもできますが、サーバーは値を格納する前に次に小さい 8 の倍数に丸めます。(例外: このような丸めはデバッグサーバーでは実行されません。) サーバーの構築方法にかかわらず、デフォルト値は 512 であり、最大許容値は 524280 です。

- [slave_checkpoint_period](#)

コマンド行形式	--slave-checkpoint-period=#
システム変数	slave_checkpoint_period
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ

型	Integer
デフォルト値	300
最小値	1
最大値 (64 ビットプラットフォーム)	18446744073709551615
最大値 (32 ビットプラットフォーム)	4294967295
単位	milliseconds

[SHOW REPLICA | SLAVE STATUS](#) で示されるように、マルチスレッドレプリカのステータスを更新するためにチェックポイント操作がコールされるまでに許容される最大時間(ミリ秒)を設定します。この変数を設定しても、マルチスレッドが有効になっていないレプリカには影響しません。この変数の設定は、実行中のチャンネルを含め、すべてのレプリケーションチャンネルに対してただちに有効になります。

注記

マルチスレッドレプリカは現在 NDB Cluster でサポートされていないため、この変数の設定は暗黙的に無視されます。詳しくは[セクション23.6.3「NDB Cluster レプリケーションの既知の問題」](#)をご覧ください。

この変数は、[slave_checkpoint_group](#) システム変数との組み合わせで、どちらかの制限を超えたときに、チェックポイントが実行され、トランザクションの数と最後のチェックポイントから経過した時間の両方を追跡するカウンタがリセットされる、という方法で機能します。

この変数の最小許容値は 1 です (サーバーが [-DWITH_DEBUG](#) を使用して構築された場合を除く、この場合の最小値は 0)。サーバーの構築方法にかかわらず、デフォルト値は 300 であり、最大可能値は 4294967296 (4G バイト) です。

- [slave_compressed_protocol](#)

コマンド行形式	<code>--slave-compressed-protocol[={OFF ON}]</code>
非推奨	8.0.18
システム変数	slave_compressed_protocol
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

ソースとレプリカの両方でサポートされている場合に、ソース/レプリカ接続プロトコルの圧縮を使用するかどうか。この変数が無効(デフォルト)の場合、接続は圧縮解除されます。この変数への変更は、後続の接続試行時に有効になります。これには、[START REPLICA | SLAVE](#) ステートメントの発行後、および実行中のレプリケーション I/O スレッドによって行われた再接続が含まれます。

[binlog_transaction_compression](#) システム変数によってアクティブ化されるバイナリログトランザクション圧縮(MySQL 8.0.20 で使用可能)を使用して帯域幅を節約することもできます。バイナリログトランザクション圧縮をプロトコル圧縮と組み合わせて使用する場合、プロトコル圧縮ではデータを処理する機会は少なくなりますが、ヘッダーと、圧縮されていないイベントおよびトランザクションペイロードは圧縮できます。バイナリログのトランザクション圧縮の詳細は、[セクション5.4.4.5「バイナリログトランザクション圧縮」](#)を参照してください。

MySQL 8.0.18 の時点では、[slave_compressed_protocol](#) が有効な場合、[CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO](#) ステートメントに指定された [SOURCE_COMPRESSION_ALGORITHMS | MASTER_COMPRESSION_ALGORITHMS](#) オプションより優先されます。この場合、ソースとレプリカの両方がそのアルゴリズムをサポートしていれば、ソースへの接続で [zlib](#) 圧縮が使用されます。[slave_compressed_protocol](#) が無効な場合、[SOURCE_COMPRESSION_ALGORITHMS |](#)

MASTER_COMPRESSION_ALGORITHMS の値が適用されます。詳細は、[セクション4.2.8「接続圧縮制御」](#)を参照してください。

MySQL 8.0.18 では、このシステム変数は非推奨です。MySQL の将来のバージョンで削除される予定です。[レガシー接続圧縮の構成](#)を参照してください。

- [slave_exec_mode](#)

コマンド行形式	<code>--slave-exec-mode=mode</code>
システム変数	<code>slave_exec_mode</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	IDEMPOTENT (NDB) STRICT (その他)
有効な値	IDEMPOTENT STRICT

レプリケーション中にレプリケーションスレッドが競合およびエラーを解決する方法を制御します。IDEMPOTENT モードでは、重複キーおよびキーが見つからないエラーが抑止されます。STRICT では、このような抑止は行われません。

IDEMPOTENT モードは、マルチソースレプリケーション、循環レプリケーション、および NDB Cluster レプリケーションのその他の特殊なレプリケーションシナリオで使用することを目的としています。(詳しくは、[セクション23.6.10「NDB Cluster レプリケーション: 双方向および循環レプリケーション」](#)および[セクション23.6.11「NDB Cluster レプリケーションの競合解決」](#)を参照してください。) NDB Cluster は、`slave_exec_mode` に明示的に設定された値を無視し、常に IDEMPOTENT として扱います。

MySQL Server 8.0 では、STRICT モードがデフォルト値です。

この変数を設定すると、実行中のチャンネルを含むすべてのレプリケーションチャンネルに対して即時に有効になります。

NDB、「IDEMPOTENT モードは、重複キーエラーおよびキーが見つからないエラーが無視される可能性があることを絶対に確認している場合にのみ使用してください」以外のストレージエンジンの場合。マルチソースレプリケーションまたは循環レプリケーションが採用されている NDB Cluster のフェイルオーバーシナリオで使用されることを意図しており、ほかの場合には使用しないことをお勧めします。

- [slave_load_tmpdir](#)

コマンド行形式	<code>--slave-load-tmpdir=dir_name</code>
システム変数	<code>slave_load_tmpdir</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ディレクトリ名

デフォルト値	Value of <code>--tmpdir</code>
--------	--------------------------------

レプリカが一時ファイルを作成するディレクトリの名前。この変数の設定は、実行中のチャンネルを含め、すべてのレプリケーションチャンネルに対してただちに有効になります。変数値は、デフォルトでは `tmpdir` システム変数の値、またはそのシステム変数が指定されていない場合に適用されるデフォルトと等しくなります。

レプリケーション SQL スレッドは、`LOAD DATA` ステートメントをレプリケートするときに、リレーログから一時ファイルにロードされるファイルを抽出し、それらをテーブルにロードします。ソースにロードされたファイルが膨大な場合、レプリカ上の一時ファイルも膨大になります。したがって、このオプションを使用して、使用可能な領域が大量にある一部のファイルシステムにあるディレクトリに一時ファイルを配置するようレプリカに指示することをお勧めします。その場合、リレーログも非常に大きいため、リレーログをそのファイルシステムに配置するように `relay_log` システム変数を設定することもできます。

このオプションで指定するディレクトリは、`LOAD DATA` ステートメントのレプリケートに使用される一時ファイルがマシンの再起動後も存続できるように、メモリーベースのファイルシステムではなくディスクベースのファイルシステムに配置する必要があります。このディレクトリは、システム起動プロセス中にオペレーティングシステムによってクリアされるものはいけません。ただし、一時ファイルが削除されている場合は、再起動後にレプリケーションを続行できるようになりました。

- [slave_max_allowed_packet](#)

コマンド行形式	<code>--slave-max-allowed-packet=#</code>
システム変数	<code>slave_max_allowed_packet</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1073741824
最小値	1024
最大値	1073741824

このオプションは、レプリケーション SQL および I/O スレッドが処理できる最大パケットサイズをバイト単位で設定します。この変数の設定は、実行中のチャンネルを含め、すべてのレプリケーションチャンネルに対してただちに有効になります。イベントヘッダーが追加されると、ソースは `max_allowed_packet` 設定より長いバイナリロギングイベントを書き込むことができます。`slave_max_allowed_packet` の設定は、行ベースのレプリケーションを使用した大規模な更新によってレプリケーションが失敗しないように、ソースの `max_allowed_packet` 設定より大きくする必要があります。

このグローバル変数は常に、1024 の正の整数の倍数である値を持ちます。そうでない何らかの値に設定しても、値は次に大きい 1024 の倍数に自動的に丸められて、格納または使用されます。`slave_max_allowed_packet` を 0 に設定すると、1024 が使用されます。(このような場合、切り捨ての警告が発行されます。) デフォルトおよび最大値は 1073741824 (1G バイト) で、最小は 1024 です。

- [slave_net_timeout](#)

コマンド行形式	<code>--slave-net-timeout=#</code>
システム変数	<code>slave_net_timeout</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	60
最小値	1

最大値	4294967295
-----	------------

レプリカが接続の切断を考慮し、読取りを中断して再接続を試行するまでに、ソースからさらにデータまたはハートビートシグナルを待機する秒数。この変数を設定しても、すぐには影響しません。変数の状態は、後続のすべての `START REPLICA | SLAVE` コマンドに適用されます。

デフォルト値は 60 秒 (1 分) です。最初の再試行はタイムアウトの直後に発生します。再試行の間隔は、`CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO` ステートメントの `SOURCE_CONNECT_RETRY | MASTER_CONNECT_RETRY` オプションによって制御され、再接続の試行回数は `SOURCE_RETRY_COUNT | MASTER_RETRY_COUNT` オプションによって制限されます。

ハートビート間隔は、データが存在しない場合に接続タイムアウトが発生するのを停止します (接続が正常な場合)。ハートビート間隔は、`CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO` ステートメントの `SOURCE_HEARTBEAT_PERIOD | MASTER_HEARTBEAT_PERIOD` オプションによって制御されます。ハートビート間隔はデフォルトで `slave_net_timeout` の半分の値に設定され、レプリカ接続メタデータリポジトリに記録されて `replication_connection_configuration` 「パフォーマンススキーマ」テーブルに表示されます。`slave_net_timeout` の値またはデフォルト設定を変更しても、明示的に設定されているか、以前に計算されたデフォルトを使用しているかにかかわらず、ハートビート間隔は自動的に変更されないことに注意してください。接続タイムアウトが変更された場合は、`CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO` も発行して、接続タイムアウトの前に発生するようにハートビート間隔を適切な値に調整する必要があります。

- `slave_parallel_type`

コマンド行形式	<code>--slave-parallel-type=value</code>
システム変数	<code>slave_parallel_type</code>
スコープ	グローバル
動的	はい
<code>SET_VAR</code> ヒントの適用	いいえ
型	列挙
デフォルト値	<code>DATABASE</code>
有効な値	<code>DATABASE</code> <code>LOGICAL_CLOCK</code>

マルチスレッドのレプリカ (`slave_parallel_workers` が 0 より大きい値に設定されているレプリカ) の場合、`slave_parallel_type` はレプリカでパラレルに実行できるトランザクションを決定するために使用されるポリシーを指定します。この変数は、マルチスレッドが有効になっていないレプリカには影響しません。指定可能な値は次のとおりです。

- `LOGICAL_CLOCK`: ソース上の同じバイナリロググループのコミットの一部であるトランザクションは、レプリカ上で並列に適用されます。可能な場合は、トランザクション間の依存性がタイムスタンプに基づいて追跡され、追加のパラレル化が提供されます。この値を設定すると、`binlog_transaction_dependency_tracking` システム変数をソースで使用して、書き込みセットがトランザクションで使用可能で、タイムスタンプと比較して改善された結果が得られる場合に、タイムスタンプのかわりに書き込みセットがパラレル化に使用されるように指定できます。
- `DATABASE`: 異なるデータベースを更新するトランザクションはパラレルに適用されます。この値は、データがソースで独立して同時に更新される複数のデータベースにパーティション化されている場合にのみ適しています。このような制約はレプリカで違反する可能性があるため、クロスデータベース制約は存在できません。

`slave_preserve_commit_order=1` が設定されている場合、使用できるのは `LOGICAL_CLOCK` のみです。

レプリケーショントポロジで複数レベルのレプリカが使用されている場合、`LOGICAL_CLOCK` ではレプリカがソースから離れている各レベルのパラレル化が実現されないことがあります。可能な場合は、ソースで

[binlog_transaction_dependency_tracking](#) を使用してパラレル化にタイムスタンプのかわりに書き込みセットを使用するように指定することで、この影響を軽減できます。

[binlog_transaction_compression](#) システム変数を使用してバイナリログトランザクション圧縮が有効になっている場合、[slave_parallel_type](#) が `DATABASE` に設定されていると、トランザクションがスケジュールされる前に、トランザクションの影響を受けるすべてのデータベースがマップされます。バイナリログトランザクション圧縮を `DATABASE` ポリシーとともに使用すると、イベントごとにマップおよびスケジュールされる圧縮されていないトランザクションと比較して並列度を減らすことができます。

- [slave_parallel_workers](#)

コマンド行形式	<code>--slave-parallel-workers=#</code>
システム変数	slave_parallel_workers
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	1024

レプリカでマルチスレッドを有効にし、レプリケーショントランザクションをパラレルに実行するためのアプライヤスレッドの数を設定します。値が 0 より大きい場合、レプリカは、指定された数のアプライヤスレッドと、それらを管理するためのコーディネータスレッドを持つマルチスレッドのレプリカです。複数のレプリケーションチャンネルを使用している場合、各チャンネルにはこの数のスレッドがあります。

注記

マルチスレッドレプリカは現在 NDB Cluster でサポートされていないため、この変数の設定は暗黙的に無視されます。詳しくは[セクション23.6.3「NDB Cluster レプリケーションの既知の問題」](#)をご覧ください。

レプリカでマルチスレッドが有効になっている場合、トランザクションの再試行がサポートされます。[slave_preserve_commit_order=1](#) の場合レプリカ上のトランザクションは、レプリカリレーログに表示される順序と同じ順序でレプリカ上で外部化されます。トランザクションがアプライヤスレッド間で分散される方法は、[slave_parallel_type](#) によって構成されます。

パラレル実行を無効にするには、このオプションを 0 に設定します。これにより、レプリカに単一のアプライヤスレッドが提供され、コーディネータスレッドは提供されません。この設定では、[slave_parallel_type](#) および [slave_preserve_commit_order](#) システム変数は効果がなく、無視されます。

[slave_parallel_workers](#) を設定しても、すぐには影響しません。変数の状態は、後続のすべての `START REPLICASlave` ステートメントに適用されます。

- [slave_pending_jobs_size_max](#)

コマンド行形式	<code>--slave-pending-jobs-size-max=#</code>
システム変数	slave_pending_jobs_size_max
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値 (≥ 8.0.12)	128M
デフォルト値 (8.0.11)	16M

最小値	1024
最大値	16EiB
単位	bytes
ブロックサイズ	1024

マルチスレッドレプリカの場合、この変数は、まだ適用されていないイベントを保持するアプライヤキューで使用可能なメモリの最大量 (バイト単位) を設定します。この変数を設定しても、マルチスレッドが有効になっていないレプリカには影響しません。この変数を設定しても、すぐには影響しません。変数の状態は、後続のすべての `START REPLICHA | SLAVE` コマンドに適用されます。

この変数に指定できる最小値は 1024 バイトです。デフォルトは 128MB です。可能な最大値は 18446744073709551615 (16 exbibytes) です。1024 バイトの正確な倍数でない値は、格納される前に 1024 バイトの次の小さい倍数に切り捨てられます。

この変数の値は弱い制限であり、通常のワークロードと一致するように設定できます。異常に大きいイベントがこのサイズを超えると、すべてのワーカースレッドに空のキューが設定されてから処理されるまで、トランザクションは保持されます。後続のすべてのトランザクションは、大規模なトランザクションが完了するまで保持されません。

- `slave_preserve_commit_order`

コマンド行形式	<code>--slave-preserve-commit-order[={OFF ON}]</code>
システム変数	<code>slave_preserve_commit_order</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

マルチスレッドのレプリカ (`slave_parallel_workers` が 0 より大きい値に設定されているレプリカ) の場合、`slave_preserve_commit_order=1` を設定すると、トランザクションはレプリカリレーログと同じ順序でレプリカで実行およびコミットされます。これにより、レプリカリレーログから実行された一連のトランザクションのギャップが回避され、レプリカとソースで同じトランザクション履歴が保持されます (ただし、次の制限があります)。この変数は、マルチスレッドが有効になっていないレプリカには影響しません。

MySQL 8.0.18 まで、`slave_preserve_commit_order=1` を設定するには、バイナリロギング (`log_bin`) およびレプリカ更新ロギング (`log_slave_updates`) がレプリカで有効になっている必要があります。これは、MySQL 8.0 のデフォルト設定です。MySQL 8.0.19 からは、バイナリロギングおよびレプリカ更新ロギングは、`slave_preserve_commit_order=1` を設定するためにレプリカでは必要なく、必要に応じて無効にできます。すべてのリリースで、`slave_preserve_commit_order=1` を設定するには、`slave_parallel_type` がデフォルト設定ではない `LOGICAL_CLOCK` に設定されている必要があります。`slave_preserve_commit_order` および `slave_parallel_type` の値を変更する前に、レプリケーション SQL スレッド (複数のレプリケーションチャンネルを使用している場合はすべてのレプリケーションチャンネル用) を停止する必要があります。

`slave_preserve_commit_order=0` が設定されている場合 (デフォルト)、マルチスレッドレプリカがパラレルで適用するトランザクションは、順序が正しくない場合があります。したがって、最後に実行されたトランザクションのチェックでは、ソースの以前のすべてのトランザクションがレプリカで実行されていることは保証されません。レプリカリレーログから実行された一連のトランザクションにギャップが生じる可能性があります。これは、マルチスレッドレプリカを使用する場合のロギングおよびリカバリに影響します。詳しくは [セクション 17.5.1.34 「レプリケーションとトランザクションの非一貫性」](#) をご覧ください。

`slave_preserve_commit_order=1` が設定されている場合、実行中のワーカースレッドは、前のすべてのトランザクションがコミットされるまで待機してからコミットします。特定のスレッドが他のワーカースレッドによるトランザクションのコミットを待機している間、そのステータスは `Waiting for preceding transaction to commit` としてレポートされます。このモードでは、マルチスレッドレプリカはソースが存在しなかった状態になることはありません。これにより、読み取りスケールアウトでのレプリケーションの使用がサポートされます。 [セクション 17.4.5 「スケールアウトのためにレプリケーションを使用する」](#) を参照してください。

注記

- `slave_preserve_commit_order=1` では、`Exec_master_log_pos` がトランザクションが実行された位置より遅れているソースバイナリログの位置ラグは防止されません。 [セクション17.5.1.34「レプリケーションとトランザクションの非一貫性」](#) を参照してください。
 - レプリカが `--binlog-do-db` などのバイナリログでフィルタを使用する場合、`slave_preserve_commit_order=1` はコミット順序およびトランザクション履歴を保持しません。
 - `slave_preserve_commit_order=1` では、非トランザクション DML 更新の順序は保持されません。これらはリレーログ内で、それらより前のトランザクションの前にコミットされる可能性があるため、レプリカリレーログから実行された一連のトランザクションにギャップが生じる可能性があります。
 - MySQL 8.0.19 より前のリリースでは、関連するオブジェクトが存在しない場合、`slave_preserve_commit_order=1` は `IF EXISTS` 句を含むステートメントの順序を保持しません。これらはリレーログ内で、それらより前のトランザクションの前にコミットされる可能性があるため、レプリカリレーログから実行された一連のトランザクションにギャップが生じる可能性があります。
 - ステートメントベースのレプリケーションが使用中で、トランザクションおよび非トランザクションの両方のストレージエンジンがソースでロールバックされる非 XA トランザクションに参加している場合、レプリカ上のコミット順序を保持する制限が発生することがあります。通常、ソースでロールバックされる非 XA トランザクションはレプリカにレプリケートされませんが、この特定の状況ではトランザクションがレプリカにレプリケートされる可能性があります。これが発生した場合、バイナリロギングのないマルチスレッドレプリカはトランザクションのロールバックを処理しないため、レプリカ上のコミット順序は、その場合のトランザクションのリレーログ順序とは異なります。
- [slave_rows_search_algorithms](#)

コマンド行形式	<code>--slave-rows-search-algorithms=value</code>
非推奨	8.0.18
システム変数	<code>slave_rows_search_algorithms</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Set
デフォルト値	<code>INDEX_SCAN,HASH_SCAN</code>
有効な値	<code>TABLE_SCAN,INDEX_SCAN</code> <code>INDEX_SCAN,HASH_SCAN</code> <code>TABLE_SCAN,HASH_SCAN</code> <code>TABLE_SCAN,INDEX_SCAN,HASH_SCAN</code> (<code>INDEX_SCAN,HASH_SCAN</code> と同等)

行ベースのロギングおよびレプリケーションのために行のバッチを準備する場合、このシステム変数は、行で一致を検索する方法、特にハッシュスキャンを使用するかどうかを制御します。このシステム変数の使用は非推奨になりました。デフォルト設定の `INDEX_SCAN,HASH_SCAN` はパフォーマンスに最適で、すべてのシナリオで正しく機能します。 [セクション17.5.1.27「レプリケーションおよび行検索」](#) を参照してください。

- [slave_skip_errors](#)

コマンド行形式	<code>--slave-skip-errors=name</code>
システム変数	<code>slave_skip_errors</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	OFF
有効な値	OFF [list of error codes] all ddl_exist_errors

通常、レプリケーションはレプリカでエラーが発生すると停止するため、データの非一貫性を手動で解決できません。この変数を指定すると、ステートメントが変数値にリストされているエラーのいずれかを返したときに、レプリケーション SQL スレッドはレプリケーションを続行します。

- [slave_sql_verify_checksum](#)

コマンド行形式	<code>--slave-sql-verify-checksum[={OFF ON}]</code>
システム変数	<code>slave_sql_verify_checksum</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

レプリケーション SQL スレッドがリレーログから読み取られたチェックサムを使用してデータを検証するようにします。不一致が発生した場合、レプリカはエラーで停止します。この変数の設定は、実行中のチャンネルを含め、すべてのレプリケーションチャンネルに対してただちに有効になります。

注記

レプリケーション I/O スレッドは、ネットワーク経由でイベントを受け入れるときに、可能であれば常にチェックサムを読み取ります。

- [slave_transaction_retries](#)

コマンド行形式	<code>--slave-transaction-retries=#</code>
システム変数	<code>slave_transaction_retries</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	10
最小値	0
最大値 (64 ビットプラットフォーム)	18446744073709551615

最大値 (32 ビットプラットフォーム)	4294967295
----------------------	------------

シングルスレッドまたはマルチスレッドレプリカ上のレプリケーション SQL スレッドが、停止前に失敗したトランザクションを自動的に再試行する最大回数を設定します。この変数の設定は、実行中のチャンネルを含め、すべてのレプリケーションチャンネルに対してただちに有効になります。デフォルト値は 10 です。変数を 0 に設定すると、トランザクションの自動再試行が無効になります。

InnoDB デッドロックのため、またはトランザクション実行時間が InnoDB の `innodb_lock_wait_timeout`、NDB の `TransactionDeadlockDetectionTimeout` または `TransactionInactiveTimeout` を超えたためにレプリケーション SQL スレッドがトランザクションの実行に失敗した場合、エラーで停止する前に `slave_transaction_retries` 回自動的に再試行されます。一時的でないエラーのあるトランザクションは再試行されません。

「パフォーマンススキーマ」テーブル `replication_applier_status` の `COUNT_TRANSACTIONS_RETRIES` カラムには、各レプリケーションチャンネルで行われた再試行回数が表示されます。「パフォーマンススキーマ」テーブル `replication_applier_status_by_worker` には、シングルスレッドまたはマルチスレッドレプリカ上の個々のアプライヤースレッドによるトランザクションの再試行に関する詳細情報が表示され、最後のトランザクションの原因となったエラーおよび現在進行中のトランザクションの再試行が識別されます。

- [slave_type_conversions](#)

コマンド行形式	<code>--slave-type-conversions=set</code>
システム変数	slave_type_conversions
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Set
デフォルト値	
有効な値	ALL_LOSSY ALL_NON_LOSSY ALL_SIGNED ALL_UNSIGNED

行ベースレプリケーションの使用時にレプリカで有効な型変換モードを制御します。その値は、リスト内のゼロ個以上の要素のカンマ区切りセットです: [ALL_LOSSY](#), [ALL_NON_LOSSY](#), [ALL_SIGNED](#), [ALL_UNSIGNED](#)。ソースとレプリカ間の型変換を禁止するには、この変数を空の文字列に設定します。この変数の設定は、実行中のチャンネルを含め、すべてのレプリケーションチャンネルに対してただちに有効になります。

行ベースレプリケーションで属性の昇格と降格に適用できるタイプ変換モードの詳細については、[行ベースレプリケーション: 属性の昇格と降格](#)を参照してください。

- [sql_slave_skip_counter](#)

システム変数	sql_slave_skip_counter
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値 (64 ビットプラットフォーム)	18446744073709551615

最大値 (32 ビットプラットフォーム)	4294967295
----------------------	------------

レプリカがスキップするソースからのイベントの数。このオプションを設定しても、すぐには影響しません。変数は次の [START REPLICHA | SLAVE](#) ステートメントに適用され、次の [START REPLICHA | SLAVE](#) ステートメントでも値が 0 に戻ります。この変数がゼロ以外の値に設定され、複数のレプリケーションチャンネルが構成されている場合、[START REPLICHA | SLAVE](#) ステートメントは `FOR CHANNEL channel` 句でのみ使用できます。

このオプションは GTID ベースのレプリケーションと互換性がなく、`gtid_mode=ON` が設定されている場合はゼロ以外の値に設定しないでください。GTID の採用時にトランザクションをスキップする必要がある場合は、かわりにソースから `gtid_executed` を使用します。[CHANGE REPLICATION SOURCE TO](#) ステートメントの [ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS](#) オプションを使用してレプリケーションチャンネルで GTID 割当てを有効にした場合、`sql_slave_skip_counter` を使用できます。[セクション17.1.7.3「トランザクションのスキップ」](#) を参照してください。

重要

この変数を設定して指定されたイベント数をスキップすると、レプリカがイベントグループの途中で開始される場合、レプリカは次のイベントグループの開始を検出し、その時点から開始するまでスキップし続けます。詳細は、[セクション17.1.7.3「トランザクションのスキップ」](#) を参照してください。

- [sync_master_info](#)

コマンド行形式	<code>--sync-master-info=#</code>
システム変数	sync_master_info
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	10000
最小値	0
最大値 (64 ビットプラットフォーム)	18446744073709551615
最大値 (32 ビットプラットフォーム)	4294967295

レプリカが接続メタデータリポジトリを更新するまでのイベント数。接続メタデータリポジトリが MySQL 8.0 のデフォルトである `InnoDB` テーブルとして格納されている場合、この数のイベントの後に更新されます。接続メタデータリポジトリが、MySQL 8.0 から非推奨になったファイルとして格納されている場合、レプリカは、この数のイベントの後に、(`fdatasync()` を使用して) `master.info` ファイルをディスクに同期します。デフォルト値は 10000 で、ゼロ値はリポジトリが更新されないことを意味します。この変数の設定は、実行中のチャンネルを含め、すべてのレプリケーションチャンネルに対してただちに有効になります。

- [sync_relay_log](#)

コマンド行形式	<code>--sync-relay-log=#</code>
システム変数	sync_relay_log
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	10000
最小値	0
最大値 (64 ビットプラットフォーム)	18446744073709551615

最大値 (32 ビットプラットフォーム)	4294967295
----------------------	------------

この変数の値が 0 より大きい場合は、すべての `sync_relay_log` イベントがリレーログに書き込まれたあとに、MySQL サーバーはそのリレーログをディスクに同期します (`fdatsync()` を使用)。この変数の設定は、実行中のチャンネルを含め、すべてのレプリケーションチャンネルに対してただちに有効になります。

`sync_relay_log` を 0 に設定すると、ディスクへの同期は実行されません。この場合、サーバーはオペレーティングシステムに依存してほかのファイルに関してリレーログの内容をときどきフラッシュします。

値 1 は、予期しない停止が発生した場合にリレーログから最大 1 つのイベントが失われるため、もっとも安全な選択です。しかし、一番遅い選択でもあります (ディスクにバッテリ付きキャッシュがある場合を除きます。その場合は同期が非常に速くなります)。予期しない停止に対して最も回復可能なレプリカの設定の組合せの詳細は、[セクション 17.4.2 「レプリカの予期しない停止の処理」](#) を参照してください。

- `sync_relay_log_info`

コマンド行形式	<code>--sync-relay-log-info=#</code>
システム変数	<code>sync_relay_log_info</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	10000
最小値	0
最大値 (64 ビットプラットフォーム)	18446744073709551615
最大値 (32 ビットプラットフォーム)	4294967295

レプリカが適用者メタデータリポジトリを更新するまでのトランザクションの数。アプライヤメタデータリポジトリが MySQL 8.0 のデフォルトである InnoDB テーブルとして格納されている場合、トランザクションのたびに更新され、このシステム変数は無視されます。アプライヤメタデータリポジトリが、MySQL 8.0 で非推奨になったファイルとして格納されている場合、レプリカは、この数のトランザクションの後に、(`fdatsync()` を使用して) `relay-log.info` ファイルをディスクに同期します。 `sync_relay_log_info` のデフォルト値は 10000 で、ゼロ値はファイルの内容がオペレーティングシステムによってのみフラッシュされることを意味します。この変数の設定は、実行中のチャンネルを含め、すべてのレプリケーションチャンネルに対してただちに有効になります。

17.1.6.4 バイナリロギングのオプションと変数

- [バイナリロギングで使用する起動オプション](#)
- [バイナリロギングで使用されるシステム変数](#)

このセクションで説明する `mysqld` オプションおよびシステム変数を使用して、バイナリログの操作に影響を与えたり、バイナリログにどのステートメントが書き込まれたかを制御したりできます。バイナリログの追加情報については、[セクション 5.4.4 「バイナリログ」](#) を参照してください。MySQL サーバーのオプションとシステム変数の使用に関する追加情報については、[セクション 5.1.7 「サーバーコマンドオプション」](#) および [セクション 5.1.8 「サーバーシステム変数」](#) を参照してください。

バイナリロギングで使用する起動オプション

次のリストでは、バイナリログを有効化したり構成したりするための起動オプションについて説明します。バイナリロギングで使用するシステム変数については、このセクションの後半で説明します。

- `--binlog-row-event-max-size=N`

コマンド行形式	<code>--binlog-row-event-max-size=#</code>
システム変数 (≥ 8.0.14)	<code>binlog_row_event_max_size</code>
スコープ (≥ 8.0.14)	グローバル

動的 (≥ 8.0.14)	いいえ
SET_VAR ヒントの適用 (≥ 8.0.14)	いいえ
型	Integer
デフォルト値	8192
最小値	256
最大値 (64 ビットプラットフォーム)	18446744073709551615
最大値 (32 ビットプラットフォーム)	4294967295

行ベースのバイナリロギングを使用する場合、この設定は、行ベースのバイナリログイベントの最大サイズに対する弱い制限(バイト単位)です。可能であれば、バイナリログに格納されている行は、この設定の値を超えないサイズのイベントにグループ化されます。イベントを分割できない場合は、最大サイズを超えることができます。値は256の倍数である必要があります(そうでない場合は切り捨てられます)。デフォルトは8192バイトです。

- `--log-bin[=base_name]`

コマンド行形式	<code>--log-bin=file_name</code>
型	ファイル名

バイナリログファイルに使用するベース名を指定します。バイナリロギングが有効になっている場合、サーバーは、データを変更するすべてのステートメントをバイナリログに記録します。バイナリログは、バックアップとレプリケーションに使用されます。バイナリログは、ベース名と数値拡張子を持つ一連のファイルです。`--log-bin` オプションの値は、ログ順序のベース名です。サーバーは、ベース名に数値の接尾辞を追加して、バイナリログファイルを順番に作成します。

`--log-bin` オプションを指定しない場合、MySQL はバイナリログファイルのデフォルトのベース名として `binlog` を使用します。以前のリリースとの互換性のために、文字列なしまたは空の文字列を指定して `--log-bin` オプションを指定した場合、ベース名はホストマシンの名前を使用して `host_name-bin` にデフォルト設定されます。

バイナリログファイルのデフォルトの場所はデータディレクトリです。ベース名に先頭の絶対パスを追加して別のディレクトリを指定することで、`--log-bin` オプションを使用して別の場所を指定できます。サーバーは、使用されたバイナリログファイルを追跡するバイナリログインデックスファイルからエントリを読み取るときに、エントリに絶対パスが含まれているかどうかを確認します。その場合、パスの相対部分は、`--log-bin` オプションを使用して設定された絶対パスに置き換えられます。バイナリログインデックスファイルに記録された絶対パスは変更されません。このような場合は、新しいパスを使用できるようにインデックスファイルを手動で編集する必要があります。バイナリログファイルのベース名と指定されたパスは、`log_bin_basename` システム変数として使用できません。

以前の MySQL バージョンでは、バイナリロギングはデフォルトで無効になっており、`--log-bin` オプションを指定した場合は有効になっていました。MySQL 8.0 からは、`--log-bin` オプションを指定するかどうかにかかわらず、バイナリロギングはデフォルトで有効になっています。例外は、バイナリロギングがデフォルトで無効になっている場合に、`mysqld` を使用して `--initialize` または `--initialize-insecure` オプションを指定してデータディレクトリを手動で呼び出すことによって、データディレクトリを初期化する場合です。この場合、`--log-bin` オプションを指定することでバイナリロギングを有効にできます。バイナリロギングが有効になっている場合、サーバー上のバイナリロギングのステータスを示す `log_bin` システム変数は ON に設定されます。

バイナリロギングを無効にするには、起動時に `--skip-log-bin` または `--disable-log-bin` オプションを指定できます。これらのオプションのいずれかが指定され、`--log-bin` も指定されている場合は、後で指定するオプションが優先されます。バイナリロギングが無効になっている場合、`log_bin` システム変数は OFF に設定されます。

GTID がサーバーで使用されているときに、異常シャットダウン後にサーバーを再起動するときにバイナリロギングを無効にすると、GTID の一部が失われ、レプリケーションが失敗する可能性があります。通常のシャットダウンでは、現在のバイナリログファイルから GTID のセットが `mysql.gtid_executed` テーブルに保存されます。これが発生しなかった異常なシャットダウンのあと、バイナリロギングがまだ有効になっている場合は、回復中に GTID がバイナリログファイルからテーブルに追加されます。サーバーの再起動に対してバイナリロギングが無効になっ

ている場合、サーバーはバイナリログファイルにアクセスして GTID を回復できないため、レプリケーションを開始できません。バイナリロギングは、通常のシャットダウン後に安全に無効にできます。

`--log-slave-updates` および `--slave-preserve-commit-order` オプションにはバイナリロギングが必要です。バイナリロギングを無効にする場合は、これらのオプションを省略するか、`--log-slave-updates=OFF` および `--skip-slave-preserve-commit-order` を指定します。`--skip-log-bin` または `--disable-log-bin` が指定されている場合、MySQL はこれらのオプションをデフォルトで無効にします。`--log-slave-updates` または `--slave-preserve-commit-order` を `--skip-log-bin` または `--disable-log-bin` とともに指定すると、警告またはエラーメッセージが発行されます。

MySQL 5.7 では、バイナリロギングが有効になっているときにサーバー ID を指定する必要がありました。そうしないと、サーバーが起動しません。MySQL 8.0 では、`server_id` システム変数はデフォルトで 1 に設定されています。バイナリロギングが有効になっている場合、このデフォルトのサーバー ID を使用してサーバーを起動できるようになりましたが、`server_id` システム変数を設定してサーバー ID を明示的に指定しないと、情報メッセージが発行されます。レプリケーショントポロジで使用されるサーバーの場合、サーバーごとにゼロ以外の一意のサーバー ID を指定する必要があります。

バイナリログの形式と管理については、[セクション5.4.4「バイナリログ」](#) を参照してください。

- `--log-bin-index[=file_name]`

コマンド行形式	<code>--log-bin-index=file_name</code>
システム変数	<code>log_bin_index</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ファイル名

バイナリログインデックスファイルの名前。バイナリログファイルの名前が含まれます。デフォルトでは、`--log-bin` オプションと拡張子 `.index` を使用してバイナリログファイルに指定された値と同じ場所とベース名を持ちます。`--log-bin` を指定しない場合、デフォルトのバイナリログインデックスファイル名は `binlog.index` です。文字列または空の文字列を指定せずに `--log-bin` オプションを指定した場合、デフォルトのバイナリログインデックスファイル名は、ホストマシンの名前を使用して `host_name-bin.index` になります。

バイナリログの形式と管理については、[セクション5.4.4「バイナリログ」](#) を参照してください。

ステートメント選択オプション。次のリストのオプションは、バイナリログに書き込まれ、レプリケーションソースサーバーによってその複製に送信されるステートメントに影響します。レプリカには、ソースから受信したどのステートメントを実行または無視するかを制御するオプションもあります。詳細は、[セクション17.1.6.3「Replica Server のオプションと変数」](#) を参照してください。

- `--binlog-do-db=db_name`

コマンド行形式	<code>--binlog-do-db=name</code>
型	文字列

このオプションは、`--replicate-do-db` がレプリケーションに影響するのと同様にバイナリロギングに影響します。

このオプションの影響は、ステートメントベースまたは行ベースロギング形式のどちらが使用されるかに依存します (`--replicate-do-db` の影響がステートメントベースまたは行ベースレプリケーションのどちらが使用されたかに依存すると同じ)。指定されたステートメントのログを記録するために使用される形式が、`binlog_format` の値で示される形式と必ずしも同じではないことに留意してください。たとえば、`CREATE TABLE` や `ALTER TABLE` などの DDL ステートメントは、有効になっているロギング形式にかかわらず、常にステートメントとしてログが記録されるため、`--binlog-do-db` の次のステートメントベースルールはステートメントのログが記録されるかどうかの判断に常に適用されます。

ステートメントベースのロギング。デフォルトデータベース (つまり、`USE` で選択されたもの) が `db_name` であるステートメントだけが、バイナリログに書き込まれます。複数のデータベースを指定するには、このオプションを複数回 (データベースごとに 1 回) 使用します。ただし、このようにしても、別のデータベースが選択されてい

るとき（またはデータベースが選択されていないとき）に、`UPDATE some_db.some_table SET foo='bar'` などのクロスデータベースステートメントのログは記録されません。

警告

複数のデータベースを指定するには、このオプションの複数インスタンスを使用する必要があります。データベース名にはカンマを含めることができるため、カンマ区切りリストを指定すると、リストは単一のデータベースの名前として扱われます。

ステートメントベースロギングを使用するときに想定される、機能しない例: サーバーが `--binlog-do-db=sales` で起動され、次のステートメントを発行する場合、`UPDATE` ステートメントのログは記録されません。

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

この「デフォルトデータベースをチェックするだけ」動作の主な理由は、ステートメントだけから複製すべきかどうかを知ることが難しいことです（たとえば、複数のデータベースをまたがって動作する複数テーブル `DELETE` ステートメントまたは複数テーブル `UPDATE` ステートメントを使用する場合）。また、必要がない場合、すべてのデータベースではなくデフォルトデータベースだけをチェックする方が早いです。

もう 1 つのケースは自明ではないかもしれませんが、オプションを設定するときに指定されなかったけれども所定のデータベースが複製されます。サーバーが `--binlog-do-db=sales` で起動される場合、`--binlog-do-db` の設定時に `prices` が含まれなかったけれども、次の `UPDATE` ステートメントのログが記録されます。

```
USE sales;
UPDATE prices.discounts SET percentage = percentage + 10;
```

`UPDATE` ステートメントが発行されたときに `sales` がデフォルトデータベースであるため、`UPDATE` のログが記録されます。

行ベースのロギング。ロギングはデータベース `db_name` に制限されます。`db_name` に属するテーブルへの変更だけがログに記録されます。デフォルトデータベースはこれに影響しません。サーバーが `--binlog-do-db=sales` で起動され、行ベースロギングが有効であると想定すると、次のステートメントが実行されます。

```
USE prices;
UPDATE sales.february SET amount=amount+100;
```

`sales` データベース内の `february` テーブルへの変更が、`UPDATE` ステートメントに従ってログに記録されます。これは `USE` ステートメントが発行されたかどうかにかかわらず発生します。ただし、行ベースロギング形式および `--binlog-do-db=sales` を使用するときは、次の `UPDATE` によって行われた変更のログは記録されません。

```
USE prices;
UPDATE prices.march SET amount=amount-25;
```

`USE prices` ステートメントが `USE sales` に変更された場合でも、`UPDATE` ステートメントの結果は依然としてバイナリログに書き込まれません。

`--binlog-do-db` 処理でステートメントベースロギングと行ベースロギング間のもう 1 つの重要な違いは、複数のデータベースを参照するステートメントに関して発生します。サーバーが `--binlog-do-db=db1` で起動され、次のステートメントが実行されると想定します。

```
USE db1;
UPDATE db1.table1, db2.table2 SET db1.table1.col1 = 10, db2.table2.col2 = 20;
```

ステートメントベースロギングを使用している場合、両方のテーブルへの更新がバイナリログに書き込まれます。一方、行ベース形式を使用するときは、`table1` への変更だけがログに記録されます。`table2` は別のデータベース内にあり、`UPDATE` によって変更されません。ここで、`USE db1` ステートメントの代わりに、`USE db4` ステートメントが使用されたものとしします。

```
USE db4;
UPDATE db1.table1, db2.table2 SET db1.table1.col1 = 10, db2.table2.col2 = 20;
```

この場合、ステートメントベースロギングを使用するときに `UPDATE` ステートメントはバイナリログに書き込まれません。一方、行ベースロギングを使用するときは、`table1` への変更のログは記録されますが、`table2` へはそうな

りません。つまり、`--binlog-do-db` によって指定されたデータベース内のテーブルへの変更のみがログに記録され、デフォルトデータベースの選択はこの動作に影響しません。

- `--binlog-ignore-db=db_name`

コマンド行形式	<code>--binlog-ignore-db=name</code>
型	文字列

このオプションは、`--replicate-ignore-db` がレプリケーションに影響するように、バイナリロギングに影響します。

このオプションの影響は、ステートメントベースまたは行ベースロギング形式のどちらが使用されるかに依存します (`--replicate-ignore-db` の影響がステートメントベースまたは行ベースレプリケーションのどちらが使用されたかに依存すると同じ)。指定されたステートメントのログを記録するために使用される形式が、`binlog_format` の値で示される形式と必ずしも同じではないことに留意してください。たとえば、`CREATE TABLE` や `ALTER TABLE` などの DDL ステートメントは、有効になっているロギング形式にかかわらず、常にステートメントとしてログが記録されるため、`--binlog-ignore-db` の次のステートメントベースルールはステートメントのログが記録されるかどうかの判断に常に適用されます。

ステートメントベースのロギング。 デフォルトデータベース (つまり、`USE` で選択されたもの) が `db_name` であるステートメントのログを記録しないようにサーバーに指示します。

デフォルトのデータベースがない場合、`--binlog-ignore-db` オプションは適用されず、このようなステートメントは常にログに記録されます。(Bug #11829838, Bug #60188)

行ベース形式。 データベース `db_name` 内のテーブルへの更新のログを記録しないようにサーバーに指示します。現在のデータベースは影響しません。

ステートメントベースロギングを使用するとき、次の例は予期するとおりに機能しません。サーバーが `--binlog-ignore-db=sales` で起動され、次のステートメントを発行すると想定します。

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

`--binlog-ignore-db` がデフォルトデータベース (`USE` ステートメントで決定) にも適用されるため、このような場合は `UPDATE` ステートメントのログが記録されます。`sales` データベースがステートメントで明示的に指定されたため、ステートメントはフィルタされませんでした。ただし、行ベースのロギングを使用している場合、`UPDATE` ステートメントの効果はバイナリログに書き込まれません。つまり、`sales.january` テーブルへの変更は記録されません。この場合、`--binlog-ignore-db=sales` は `sales` データベースのソースコピー内のテーブルに対して行われた all の変更をバイナリロギングのために無視します。

無視するデータベースを複数指定するには、このオプションを複数回 (データベースごとに 1 回) 使用します。データベース名にはカンマを含めることができるため、カンマ区切りリストを指定すると、リストは単一のデータベースの名前として扱われます。

クロスデータベース更新を使用していて、それらの更新ログを記録したくない場合は、このオプションを使用しないでください。

チェックサムオプション。 MySQL では、バイナリログチェックサムの読み取りと書き込みがサポートされています。これらは、ここで示す 2 つのオプションを使用して有効化されます。

- `--binlog-checksum={NONE|CRC32}`

コマンド行形式	<code>--binlog-checksum=type</code>
型	文字列
デフォルト値	CRC32
有効な値	NONE CRC32

このオプションを有効にすると、ソースはバイナリログに書き込まれたイベントのチェックサムを書き込みます。`NONE` に設定して無効にするか、チェックサムの生成に使用するアルゴリズムの名前を指定します。現在、CRC32

チェックサムのみがサポートされており、CRC32 がデフォルトです。トランザクション内のこのオプションの設定は変更できません。

レプリカによる (リレーログからの) チェックサムの読み取りを制御するには、`--slave-sql-verify-checksum` オプションを使用します。

テストおよびデバッグのオプション。次のバイナリログオプションは、レプリケーションテストおよびデバッグで使用されます。これらは通常操作での使用を意図していません。

- `--max-binlog-dump-events=N`

コマンド行形式	<code>--max-binlog-dump-events=#</code>
型	Integer
デフォルト値	0

このオプションは、レプリケーションのテストとデバッグのために、MySQL テストスイートで内部的に使用されません。

- `--sporadic-binlog-dump-fail`

コマンド行形式	<code>--sporadic-binlog-dump-fail[={OFF ON}]</code>
型	Boolean
デフォルト値	OFF

このオプションは、レプリケーションのテストとデバッグのために、MySQL テストスイートで内部的に使用されません。

バイナリロギングで使用されるシステム変数

次の一覧で、バイナリロギングを制御するためのシステム変数について説明します。これらはサーバー起動時に設定でき、それらの一部は `SET` を使用して実行時に変更できます。バイナリロギングを制御するために使用されるサーバーオプションは、このセクションですでにリストされています。

- `binlog_cache_size`

コマンド行形式	<code>--binlog-cache-size=#</code>
システム変数	<code>binlog_cache_size</code>
スコープ	グローバル
動的	はい
<code>SET_VAR</code> ヒントの適用	いいえ
型	Integer
デフォルト値	32768
最小値	4096
最大値 (64 ビットプラットフォーム)	18446744073709551615
最大値 (32 ビットプラットフォーム)	4294967295

トランザクション中にバイナリログに対する変更を保持するメモリーバッファのサイズ。(`log_bin` システム変数を ON に設定して) サーバーでバイナリロギングが有効になっている場合、サーバーがトランザクションストレージエンジンをサポートしていれば、クライアントごとにバイナリログキャッシュが割り当てられます。トランザクションのデータがメモリーバッファ内の領域を超えた場合、余分なデータは一時ファイルに格納されます。バイナリログの暗号化がサーバーでアクティブな場合、メモリーバッファは暗号化されませんが、バイナリログキャッシュの保持に使用される一時ファイルは (MySQL 8.0.17 から) 暗号化されます。各トランザクションがコミットされると、メモリーバッファをクリアし、一時ファイル (使用されている場合) を切り捨てることで、バイナリログキャッシュがリセットされます。

大規模なトランザクションを頻繁に使用する場合は、一時ファイルへの書き込みを削減または排除することで、このキャッシュサイズを増やしてパフォーマンスを向上させることができます。 [Binlog_cache_use](#) および [Binlog_cache_disk_use](#) ステータス変数は、この変数のサイズを調整するために役立つことがあります。 [セクション5.4.4「バイナリログ」](#)を参照してください。

[binlog_cache_size](#) はトランザクションキャッシュのみのサイズを設定します。ステートメントキャッシュのサイズは [binlog_stmt_cache_size](#) システム変数によって管理されます。

- [binlog_checksum](#)

コマンド行形式	--binlog-checksum=name
システム変数	binlog_checksum
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	CRC32
有効な値	NONE CRC32

この変数を有効にすると、ソースは各イベントのチェックサムをバイナリログに書き込みます。 [binlog_checksum](#) は、[NONE](#) (チェックサムを無効にする) および [CRC32](#) の値をサポートしています。デフォルトは [CRC32](#) です。 [binlog_checksum](#) が無効 (値 [NONE](#)) のときは、サーバーは各イベントのイベント長 (チェックサムではなく) を書き込んでチェックすることで、すべてそろったイベントのみをバイナリログに書き込んでいることを検証します。

ソースでこの変数をレプリカで認識されない値に設定すると、レプリカは独自の [binlog_checksum](#) 値を [NONE](#) に設定し、エラーでレプリケーションを停止します。古いレプリカとの下位互換性に問題がある場合は、値を明示的に [NONE](#) に設定できます。

MySQL 8.0.20 まで、グループレプリケーションはチェックサムを使用できず、バイナリログでのチェックサムの存在をサポートしないため、サーバーインスタンスがグループメンバーになるように構成するときに [binlog_checksum=NONE](#) を設定する必要があります。MySQL 8.0.21 からは、グループレプリケーションでチェックサムがサポートされるため、グループメンバーはデフォルト設定を使用できます。

チェックサムはバイナリログファイル全体に対して書き込まれる必要があり、バイナリログの一部に対してのみ書き込まれないため、[binlog_checksum](#) の値を変更するとバイナリログがローテーションされます。トランザクション内の [binlog_checksum](#) の値は変更できません。

[binlog_transaction_compression](#) システム変数を使用してバイナリログのトランザクション圧縮が有効になっている場合、圧縮されたトランザクションペイロード内の個々のイベントのチェックサムは書き込まれません。代わりに、GTID イベントに対してチェックサムが書き込まれ、圧縮された [Transaction_payload_event](#) に対してチェックサムが書き込まれます。

- [binlog_direct_non_transactional_updates](#)

コマンド行形式	--binlog-direct-non-transactional-updates[={OFF ON}]
システム変数	binlog_direct_non_transactional_updates
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean

デフォルト値	OFF
--------	-----

同時実行性の問題のため、トランザクションにトランザクションテーブルと非トランザクションテーブルの両方に対する更新が含まれる場合、レプリカに一貫性がなくなる可能性があります。MySQL は、非トランザクションステートメントをトランザクションキャッシュに書き込むことで（コミット時にフラッシュされる）、これらのステートメント間の因果関係を維持しようとします。ただし、トランザクションに代わって非トランザクションテーブルに行われた変更がほかの接続にただちに可視になるときに、問題が起こります（これらの変更がただちにバイナリログに書き込まれない場合があるため）。

`binlog_direct_non_transactional_updates` 変数は、この問題に対する 1 つの可能な回避策を提供します。デフォルトでは、この変数は無効です。`binlog_direct_non_transactional_updates` を有効にすることで、非トランザクションテーブルへの更新が、トランザクションキャッシュではなく直接バイナリログに書き込まれます。

MySQL 8.0.14 では、このシステム変数のセッション値の設定は制限された操作です。セッションユーザーには、制限付きセッション変数を設定するのに十分な権限が必要です。[セクション 5.1.9.1 「システム変数権限」](#) を参照してください。

`binlog_direct_non_transactional_updates` は、ステートメントベースバイナリロギング形式を使用して複製されるステートメントにのみ機能します。つまり、`binlog_format` の値が `STATEMENT` のとき、または `binlog_format` が `MIXED` で、与えられたステートメントがステートメントベース形式を使用して複製されているときのみ機能します。バイナリログ形式が `ROW` のとき、または `binlog_format` が `MIXED` に設定され、与えられたステートメントが行ベース形式で複製されるときは、この変数は効果がありません。

重要

この変数を有効にする前に、トランザクションおよび非トランザクションテーブルの間に依存関係がないことを確認する必要があります。このような依存関係の例は、ステートメント `INSERT INTO myisam_table SELECT * FROM innodb_table` です。そうしないと、このようなステートメントによってレプリカがソースと相違する可能性があります。

バイナリログ形式が `ROW` または `MIXED` の場合、この変数は無効です。

• `binlog_encryption`

コマンド行形式	<code>--binlog-encryption[={OFF ON}]</code>
導入	8.0.14
システム変数	<code>binlog_encryption</code>
スコープ	グローバル
動的	はい
<code>SET_VAR</code> ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

このサーバー上のバイナリログファイルおよびリレーログファイルの暗号化を有効にします。`OFF` がデフォルトです。`ON` は、バイナリログファイルおよびリレーログファイルの暗号化をオンに設定します。暗号化を有効にするためにバイナリロギングをサーバー上で有効にする必要はないため、バイナリログのないレプリカ上のリレーログファイルを暗号化できます。暗号化を使用するには、MySQL Server 鍵リングサービスを提供するように鍵リングプラグインをインストールして構成する必要があります。これを行う手順は、[セクション 6.4.4 「MySQL キーリング」](#) を参照してください。サポートされている任意のキーリングプラグインを使用して、バイナリログ暗号化鍵を格納できます。

バイナリログ暗号化を有効にしてサーバーをはじめて起動すると、バイナリログとリレーログが初期化される前に新しいバイナリログ暗号化鍵が生成されます。このキーは、各バイナリログファイル（サーバーでバイナリロギングが有効になっている場合）およびリレーログファイル（サーバーにレプリケーションチャンネルがある場合）のファイルパスワードを暗号化するために使用され、ファイルパスワードから生成された以降の鍵を使用してファイル内のデータが暗号化されます。リレーログファイルは、グループレプリケーションアプライヤチャンネルや暗号化のアクティブ化後に作成される新しいチャンネルなど、すべてのチャンネルに対して暗号化されます。バイナリログインデックスファイルとリレーログインデックスファイルは暗号化されません。

サーバーの実行中に暗号化をアクティブ化すると、その時点で新しいバイナリログ暗号化鍵が生成されます。例外は、以前にサーバー上で暗号化がアクティブであり、その後無効になっていた場合です。その場合、以前に使用されていたバイナリログ暗号化鍵が再度使用されます。バイナリログファイルとリレーログファイルはただちにローテーションされ、新しいファイルとそれ以降のすべてのバイナリログファイルおよびリレーログファイルのファイルパスワードは、このバイナリログ暗号化鍵を使用して暗号化されます。既存のバイナリログファイルとリレーログファイルがサーバー上にまだ存在する場合、それらは自動的に暗号化されませんが、不要になった場合はバージョンできません。

`binlog_encryption` システム変数を `OFF` に変更して暗号化を非アクティブにすると、バイナリログファイルとリレーログファイルはただちにローテーションされ、それ以降のすべてのロギングは暗号化されません。以前に暗号化されたファイルは自動的に復号化されませんが、サーバーはそれらを読み取ることができます。サーバーの実行中に暗号化をアクティブ化または非アクティブ化するには、`BINLOG_ENCRYPTION_ADMIN` 権限 (または非推奨の `SUPER` 権限) が必要です。グループレプリケーションアプライヤチャネルはリレーログローテーションリクエストに含まれないため、これらのチャネルの暗号化されていないロギングは、ログが通常の使用でローテーションされるまで開始されません。

バイナリログファイルとリレーログファイルの暗号化の詳細は、[セクション17.3.2「バイナリログファイルとリレーログファイルの暗号化」](#) を参照してください。

- `binlog_error_action`

コマンド行形式	<code>--binlog-error-action[=value]</code>
システム変数	<code>binlog_error_action</code>
スコープ	グローバル
動的	はい
<code>SET_VAR</code> ヒントの適用	いいえ
型	列挙
デフォルト値	<code>ABORT_SERVER</code>
有効な値	<code>IGNORE_ERROR</code> <code>ABORT_SERVER</code>

バイナリログへの書き込み、フラッシュ、同期ができないなどのエラーがサーバーで発生した場合に何が起るかを制御します。これにより、ソースバイナリログが矛盾し、レプリカが同期を失う可能性があります。

この変数のデフォルトは `ABORT_SERVER` で、バイナリログでこのようなエラーが発生するたびに、サーバーはロギングを停止してシャットダウンします。再起動時に、予期しないサーバーが停止した場合と同様にリカバリが続行されます ([セクション17.4.2「レプリカの予期しない停止の処理」](#) を参照)。

`binlog_error_action` が `IGNORE_ERROR` に設定されている場合、サーバーでこのようなエラーが発生すると、進行中のトランザクションが続行され、エラーがログに記録されてからロギングが停止され、更新の実行が続行されます。バイナリロギングを再開するには、`log_bin` を再度有効にする必要があります。これにはサーバーの再起動が必要です。この設定により、古いバージョンの MySQL との下位互換性が提供されます。

- `binlog_expire_logs_seconds`

コマンド行形式	<code>--binlog-expire-logs-seconds=#</code>
システム変数	<code>binlog_expire_logs_seconds</code>
スコープ	グローバル
動的	はい
<code>SET_VAR</code> ヒントの適用	いいえ
型	Integer
デフォルト値	<code>2592000</code>
最小値	<code>0</code>

最大値	4294967295
-----	------------

バイナリログの有効期限を秒単位で設定します。有効期限の終了後、バイナリログファイルを自動的に削除できません。削除は起動時およびバイナリログがフラッシュされる時に発生する可能性があります。ログのフラッシュは、[セクション5.4「MySQL Server ログ」](#)に記載されているように発生します。

デフォルトのバイナリログ有効期限は 2592000 秒で、30 日 (30*24*60*60 秒) です。デフォルトは、[binlog_expire_logs_seconds](#) と非推奨のシステム変数 [expire_logs_days](#) のどちらにも起動時に設定された値がない場合に適用されます。[binlog_expire_logs_seconds](#) または [expire_logs_days](#) のいずれかの変数にゼロ以外の値が起動時に設定されている場合、この値はバイナリログの有効期限として使用されます。これらの両方の変数にゼロ以外の値が起動時に設定されている場合、[binlog_expire_logs_seconds](#) の値がバイナリログの有効期限として使用され、[expire_logs_days](#) の値は警告メッセージとともに無視されます。

バイナリログの自動パージを無効にするには、[binlog_expire_logs_seconds](#) に値 0 を明示的に指定し、[expire_logs_days](#) に値を指定しないでください。以前のリリースとの互換性のために、[expire_logs_days](#) に値 0 を明示的に指定し、[binlog_expire_logs_seconds](#) に値を指定しないと、自動パージも無効になります。その場合、[binlog_expire_logs_seconds](#) のデフォルトは適用されません。

バイナリログファイルを手動で削除するには、[PURGE BINARY LOGS](#) ステートメントを使用します。[セクション13.4.1.1「PURGE BINARY LOGS ステートメント」](#)を参照してください。

- [binlog_format](#)

コマンド行形式	--binlog-format=format
システム変数	binlog_format
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	ROW
有効な値	ROW STATEMENT

MIXED

この変数はバイナリロギング形式を設定し、`STATEMENT`、`ROW`、`MIXED` のいずれかが可能です。 [セクション 17.2.1「レプリケーション形式」](#) を参照してください。

`binlog_format` は起動時または実行時に設定できますが、後で説明するように、一部の条件下ではこの変数を実行時に変更できないか、レプリケーションが失敗することを除きます。

デフォルトは `ROW` です。例外: NDB Cluster では、デフォルトは `MIXED` です。ステートメントベースのレプリケーションは NDB Cluster ではサポートされません。

このシステム変数のセッション値の設定は制限された操作です。セッションユーザーには、制限付きセッション変数を設定するのに十分な権限が必要です。 [セクション 5.1.9.1「システム変数権限」](#) を参照してください。

この変数への変更がいつ有効になり、影響はどのくらい長く続くかを管理するルールは、ほかの MySQL サーバースystem変数の場合と同じです。詳細は、 [セクション 13.7.6.1「変数代入の SET 構文」](#) を参照してください。

`MIXED` が指定されている場合、行ベースレプリケーションだけが適切な結果になることが保証されている場合を除き、ステートメントベースレプリケーションが使用されます。たとえば、これはユーザー定義関数 (UDF) または `UUID()` 関数がステートメントに含まれているときに発生します。

各バイナリロギング形式が設定されている場合のストアードプログラム (ストアードプロシージャとストアードファンクション、トリガー、およびイベント) の処理方法の詳細は、 [セクション 25.7「ストアードプログラムバイナリロギング」](#) を参照してください。

レプリケーション形式を実行時に切り替えることができない例外もあります。

- レプリケーション形式は、ストアードファンクションまたはトリガー内から変更できません。
- セッションにオープン一時テーブルがある場合、セッション (`SET @@SESSION.binlog_format`) のレプリケーション形式は変更できません。
- いずれかのレプリケーションチャンネルにオープン一時テーブルがある場合、レプリケーション形式はグローバルに変更できません (`SET @@GLOBAL.binlog_format` または `SET @@PERSIST.binlog_format`)。
- レプリケーションチャンネルアプライヤスレッドが現在実行されている場合、レプリケーション形式はグローバルに変更できません (`SET @@GLOBAL.binlog_format` または `SET @@PERSIST.binlog_format`)。

これらのいずれかの場合 (または現在のレプリケーション形式を設定しようとする場合) にレプリケーション形式を切り替えようとする、エラーになります。ただし、`PERSIST_ONLY` (`SET @@PERSIST_ONLY.binlog_format`) を使用してレプリケーション形式をいつでも変更できます。これは、このアクションによってランタイムグローバルシステム変数の値が変更されず、サーバの再起動後にのみ有効になるためです。

一時テーブルが存在する場合、実行時にレプリケーション形式を切り替えることはお勧めしません。一時テーブルはステートメントベースレプリケーションの使用時にのみログに記録されますが、行ベースレプリケーションと混在レプリケーションではログに記録されないためです。

レプリケーションソースサーバでロギング形式を変更しても、レプリカのロギング形式は一致するように変更されません。レプリケーションの進行中にレプリケーション形式を切り替えると、レプリカでバイナリロギングが有効になっている場合に問題が発生し、ソースが `ROW` または `MIXED` 形式のロギングを使用している間に、変更によってレプリカで `STATEMENT` 形式のロギングが使用されることがあります。レプリカは、`ROW` ロギング形式で受信したバイナリログエントリを独自のバイナリログで使用するために `STATEMENT` 形式に変換できないため、レ

アプリケーションが失敗する可能性があります。詳細は、[セクション5.4.4.2「バイナリログ形式の設定」](#)を参照してください。

バイナリログ形式は次のサーバーオプションの動作に影響を与えます。

- `--replicate-do-db`
- `--replicate-ignore-db`
- `--binlog-do-db`
- `--binlog-ignore-db`

これらの影響の詳細は、個々のオプションの説明に記載されています。

- `binlog_group_commit_sync_delay`

コマンド行形式	<code>--binlog-group-commit-sync-delay=#</code>
システム変数	<code>binlog_group_commit_sync_delay</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	1000000

バイナリログファイルをディスクに同期する前にバイナリログのコミットが待機するマイクロ秒数を制御します。デフォルトでは、`binlog_group_commit_sync_delay` は 0 に設定されており、これは遅延がないことを意味します。`binlog_group_commit_sync_delay` をマイクロ秒遅延に設定すると、必要なグループがグループごとの時間単位が少なくなるため、一度により多くのトランザクションをディスクに同期でき、トランザクションのグループをコミットするための全体的な時間が短縮されます。

`sync_binlog=0` または `sync_binlog=1` が設定されている場合、`binlog_group_commit_sync_delay` によって指定された遅延は、同期の前 (または `sync_binlog=0` の場合は先に進む前) にすべてのバイナリログコミットグループに適用されます。`sync_binlog` が 1 より大きい値 `n` に設定されている場合、遅延はすべての `n` バイナリログコミットグループのあとに適用されます。

`binlog_group_commit_sync_delay` を設定すると、レプリカを持つ (またはフェイルオーバー後に存在する可能性のある) サーバー上のパラレルコミットトランザクションの数を増やすことができます。この効果を活用するには、レプリカサーバーに `slave_parallel_type=LOGICAL_CLOCK` が設定されている必要があります。また、`binlog_transaction_dependency_tracking=COMMIT_ORDER` も設定されている場合は、この効果が重要になります。`binlog_group_commit_sync_delay` の設定をチューニングする場合は、ソースのスループットとレプリカのスループットの両方を考慮することが重要です。

`binlog_group_commit_sync_delay` を設定すると、バイナリログを持つサーバー (ソースまたはレプリカ) 上のバイナリログへの `fsync()` 呼び出しの数を減らすこともできます。

`binlog_group_commit_sync_delay` を設定すると、サーバー上のトランザクションの待機時間が長くなり、クライアントアプリケーションに影響する可能性があることに注意してください。また、同時実行性の高いワークロードでは、遅延によって競合が増加し、スループットが低下する可能性があります。通常、遅延を設定する利点は欠点を上回っていますが、最適な設定を決定するには、チューニングを常に行う必要があります。

- `binlog_group_commit_sync_no_delay_count`

コマンド行形式	<code>--binlog-group-commit-sync-no-delay-count=#</code>
システム変数	<code>binlog_group_commit_sync_no_delay_count</code>

スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	1000000

`binlog_group_commit_sync_delay` で指定された現在の遅延を中断する前に待機するトランザクションの最大数。`binlog_group_commit_sync_delay` が 0 に設定されている場合、このオプションは無効です。

- `binlog_max_flush_queue_time`

コマンド行形式	<code>--binlog-max-flush-queue-time=#</code>
非推奨	はい
システム変数	<code>binlog_max_flush_queue_time</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	100000

`binlog_max_flush_queue_time` は非推奨であり、将来の MySQL リリースでは最終的に削除対象としてマークされます。以前は、このシステム変数は、グループコミットを続行する前にフラッシュキューからトランザクションの読取りを続行する時間をマイクロ秒単位で制御していました。効果はなくなりました。

- `binlog_order_commits`

コマンド行形式	<code>--binlog-order-commits[={OFF ON}]</code>
システム変数	<code>binlog_order_commits</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

この変数がレプリケーションソースサーバー (デフォルト) で有効になっている場合、ストレージエンジンに発行されるトランザクションコミット命令はシングルスレッド上で直列化されるため、トランザクションは常にバイナリログに書き込まれるのと同じ順序でコミットされます。この変数を無効にすると、複数のスレッドを使用してトランザクションコミット命令を発行できます。バイナリロググループのコミットと組み合わせて使用すると、単一のトランザクションのコミット率がスループットのボトルネックになるのを防ぐため、パフォーマンスが向上する可能性があります。

トランザクションは、関連するすべてのストレージエンジンがトランザクションのコミット準備を完了したことを確認した時点で、バイナリログに書き込まれます。その後、バイナリロググループのコミットロジックは、バイナリログの書き込みが行われたあとにトランザクションのグループをコミットします。`binlog_order_commits` が無効になっている場合、このプロセスに複数のスレッドが使用されているため、コミットグループ内のトランザクションはバイナリログ内の順序とは異なる順序でコミットされる可能性があります。(単一のクライアントからのトランザクションは常に時系列順にコミットされます。) 多くの場合、これは問題ではありません。個別のトランザクシ

ンで実行される操作は一貫した結果を生成する必要があり、そうでない場合は、かわりに単一のトランザクションを使用する必要があります。

ソースとマルチスレッドのレプリカのトランザクション履歴が同一であることを確認する場合は、レプリカに `slave_preserve_commit_order=1` を設定します。

- `binlog_rotate_encryption_master_key_at_startup`

コマンド行形式	<code>--binlog-rotate-encryption-master-key-at-startup[={OFF ON}]</code>
導入	8.0.14
システム変数	<code>binlog_rotate_encryption_master_key_at_startup</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

サーバーの起動時にバイナリログマスターキーをローテーションするかどうかを指定します。バイナリログマスターキーは、バイナリログファイルおよびサーバー上のリレーログファイルのファイルパスワードを暗号化するために使用されるバイナリログ暗号化鍵です。バイナリログ暗号化 (`binlog_encryption=ON`) を有効にしてサーバーをはじめて起動すると、新しいバイナリログ暗号化鍵が生成され、バイナリログマスターキーとして使用されます。`binlog_rotate_encryption_master_key_at_startup` システム変数も ON に設定されている場合、サーバーが再起動されるたびにバイナリログ暗号化鍵が生成され、後続のすべてのバイナリログファイルおよびリレーログファイルのバイナリログマスターキーとして使用されます。`binlog_rotate_encryption_master_key_at_startup` システム変数が OFF (デフォルト) に設定されている場合、サーバーの再起動後に既存のバイナリログマスターキーが再度使用されます。バイナリログ暗号化鍵とバイナリログマスターキーの詳細は、[セクション17.3.2「バイナリログファイルとリレーログファイルの暗号化」](#) を参照してください。

- `binlog_row_event_max_size`

コマンド行形式	<code>--binlog-row-event-max-size=#</code>
システム変数 (≥ 8.0.14)	<code>binlog_row_event_max_size</code>
スコープ (≥ 8.0.14)	グローバル
動的 (≥ 8.0.14)	いいえ
SET_VAR ヒントの適用 (≥ 8.0.14)	いいえ
型	Integer
デフォルト値	8192
最小値	256
最大値 (64 ビットプラットフォーム)	18446744073709551615
最大値 (32 ビットプラットフォーム)	4294967295

行ベースのバイナリロギングを使用する場合、この設定は、行ベースのバイナリログイベントの最大サイズに対する弱い制限 (バイト単位) です。可能であれば、バイナリログに格納されている行は、この設定の値を超えないサイズのイベントにグループ化されます。イベントを分割できない場合は、最大サイズを超えることができます。値は 256 の倍数である必要があります (そうでない場合は切り捨てられます)。デフォルトは 8192 バイトです。

このグローバルシステム変数は読み取り専用で、サーバーの起動時にのみ設定できます。したがって、その値を変更するには、SET ステートメントで `PERSIST_ONLY` キーワードまたは `@@persist_only` 修飾子を使用する必要があります。

- `binlog_row_image`

コマンド行形式	<code>--binlog-row-image=image_type</code>
---------	--

システム変数	binlog_row_image
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	full
有効な値	<p>full (すべてのカラムをログに記録)</p> <p>minimal (変更されたカラムおよび、行を特定するために必要とされたカラムのみをログに記録)</p> <p>noblob (不要な BLOB カラムおよび TEXT カラム以外のすべてのカラムをログに記録)</p>

MySQL 行ベースレプリケーションの場合、この変数は行イメージをバイナリログに書き込む方法を決定します。

このシステム変数のセッション値の設定は制限された操作です。セッションユーザーには、制限付きセッション変数を設定するのに十分な権限が必要です。 [セクション5.1.9.1「システム変数権限」](#)を参照してください。

MySQL 行ベースレプリケーションでは、各行変更イベントに 2 つのイメージ、つまり「前」イメージ (更新される行を検索するときにこれらのカラムが照合される) と「後」イメージ (変更を含む) が含まれます。通常、MySQL は前イメージと後イメージの両方のためにすべての行 (つまり、すべてのカラム) のログを記録します。ただし、両方のイメージにすべてのカラムが厳密に含まれている必要はなく、多くの場合、実際に必要なカラムのログのみを記録することでディスク、メモリー、およびネットワーク使用量を節約できます。

注記

行を削除するときは、削除後に伝達する変更後の値がないため、前イメージのみのログが記録されます。行を挿入するときは、照合される既存の行がないため、後イメージのみのログが記録されます。行を更新するときのみ、前イメージと後イメージの両方が必要で、両方がバイナリログに書き込まれます。

前イメージについては、行を一意に識別するために必要な最小カラムセットのログが記録されることのみが必要です。行を含むテーブルに主キーがある場合、主キーカラムだけがバイナリログに書き込まれます。そうではなく、テーブルに一意キーがあり、そのカラムのすべてが **NOT NULL** の場合は、一意キー内のカラムのログのみを記録する必要があります。(テーブルに **NULL** カラムなしの主キーまたは一意キーがない場合、すべてのカラムが前イメージで使用され、それらのログが記録される必要があります。) 後イメージでは、実際に変更されたカラムのログのみを記録する必要があります。

[binlog_row_image](#) システム変数を使用して、サーバーに完全な行または最小限の行を記録させることができます。この変数は実際には、次のリストで示す 3 つの可能な値の 1 つを取ることができます。

- **full**: 前イメージと後イメージの両方にすべてのカラムのログを記録します。
- **minimal**: 変更する行を識別するために必要なピフォアイメージのカラムのみをログに記録します。SQL ステートメントで値が指定されたか、自動増分によって生成されたアフターイメージのカラムのみをログに記録します。

- **noblob**: すべてのカラム (**full** と同じ) のログを記録しますが、行の識別に必要な、または変更されなかった **BLOB** および **TEXT** カラムは除きます。

注記

この変数は NDB Cluster ではサポートされていません。設定しても **NDB** テーブルのロギングには影響しません。

デフォルト値は **full** です。

minimal または **noblob** を使用するときは、ソーステーブルと宛先テーブルの両方について次の条件が **true** の場合のみ、所定のテーブルに対して削除と更新が正しく機能することが保証されます。

- すべてのカラムが同じ順番で存在する必要があります。それぞれのカラムがもう一方のテーブル内の対応するものと同じデータ型を使用する必要があります。
- これらのテーブルの主キー定義が同じである必要があります。

(つまり、これらのテーブルは同じである必要がありますが、テーブルの主キーの一部でないインデックスがある場合にはそれらは除きます。)

これらの条件が満たされない場合は、宛先テーブル内の主キーカラム値が、削除または更新のための一意一致を提供するために不十分であることが判明する場合があります。この場合、警告やエラーは発行されず、ソースとレプリカは暗黙的に相違するため、一貫性が損なわれます。

バイナリロギング形式が **STATEMENT** のときは、この変数を設定しても効果はありません。 **binlog_format** が **MIXED** の場合、 **binlog_row_image** の設定は行ベースの形式を使用して記録された変更に適用されますが、この設定はステートメントとして記録された変更には影響しません。

グローバルまたはセッションレベルのいずれかで **binlog_row_image** を設定しても、暗黙的にコミットされません。これは、トランザクションの進行中にトランザクションに影響を与えずにこの変数を変更できることを意味します。

- **binlog_row_metadata**

コマンド行形式	<code>--binlog-row-metadata=metadata_type</code>
システム変数	<code>binlog_row_metadata</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	MINIMAL
有効な値	FULL (すべてのメタデータが含まれます) MINIMAL (含まれるメタデータの制限)

行ベースロギングの使用時にバイナリログに追加されるテーブルメタデータの量を構成します。 **MINIMAL** (デフォルト) に設定すると、 **SIGNED** フラグ、カラム文字セットおよびジオメトリタイプに関連するメタデータのみがログに記録されます。 **FULL** に設定すると、カラム名、 **ENUM** または **SET** 文字列値、 **PRIMARY KEY** 情報など、テーブルの完全なメタデータがログに記録されます。

拡張メタデータは、次の目的で使用されます:

- レプリカは、テーブル構造がソースと異なる場合、メタデータを使用してデータを転送します。
- 外部ソフトウェアでは、メタデータを使用して行イベントをデコードし、データウェアハウスなどの外部データベースにデータを格納できます。

- **binlog_row_value_options**

コマンド行形式	<code>--binlog-row-value-options=#</code>
システム変数	<code>binlog_row_value_options</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Set
デフォルト値	"
有効な値	<code>PARTIAL_JSON</code>

`PARTIAL_JSON` に設定すると、JSON ドキュメントの小さい部分のみを変更する更新に領域効率のよいバイナリログ形式を使用できるようになります。これにより、行ベースのレプリケーションでは、JSON ドキュメントの変更された部分のみがバイナリログ内の更新のイメージに書き込まれます (完全なドキュメントは書き込まれません)。これは、`JSON_SET()`、`JSON_REPLACE()` および `JSON_REMOVE()` の任意の順序を使用して JSON カラムを変更する `UPDATE` ステートメントに対して機能します。変更にはドキュメント全体よりも多くの領域が必要な場合、またはサーバーが部分更新を生成できない場合は、かわりにドキュメント全体が使用されます。

このシステム変数のセッション値の設定は制限された操作です。セッションユーザーには、制限付きセッション変数を設定するのに十分な権限が必要です。セクション 5.1.9.1 「システム変数権限」を参照してください。

サポートされている値は `PARTIAL_JSON` のみです。`binlog_row_value_options` の設定を解除するには、その値を空の文字列に設定します。

`binlog_row_value_options=PARTIAL_JSON` は、バイナリロギングが有効で、`binlog_format` が `ROW` または `MIXED` に設定されている場合にのみ有効になります。ステートメントベースレプリケーション `always` は、`binlog_row_value_options` に設定された値に関係なく、JSON ドキュメントの変更された部分のみを記録します。節約される領域の量を最大化するには、このオプションとともに `binlog_row_image=NOBLOB` または `binlog_row_image=MINIMAL` を使用します。完全な JSON ドキュメントはビフォアイメージに格納され、部分更新はイメージにのみ格納されるため、`binlog_row_image=FULL` はこれらのいずれかよりも少ない領域を節約します。

`mysqlbinlog` 出力には、`BINLOG` ステートメントを使用して base-64 文字列としてエンコードされたイベントの形式で部分的な JSON 更新が含まれます。`--verbose` オプションが指定されている場合、`mysqlbinlog` は擬似 SQL ステートメントを使用して部分的な JSON 更新を読み取り可能な JSON として表示します。

レプリカ上の JSON ドキュメントに変更を適用できない場合、MySQL レプリケーションはエラーを生成します。これには、パスの検索の失敗も含まれます。この安全性チェックおよびその他の安全性チェックでも、レプリカ上の JSON ドキュメントがソース上の JSON ドキュメントと相違しており、部分更新が適用されても、理論的にはレプリカ上に有効だが予期しない JSON ドキュメントを生成できることに注意してください。

- `binlog_rows_query_log_events`

コマンド行形式	<code>--binlog-rows-query-log-events[={OFF ON}]</code>
システム変数	<code>binlog_rows_query_log_events</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean

デフォルト値	OFF
--------	-----

このシステム変数は、行ベースのロギングにのみ影響します。有効にすると、サーバーは行クエリーログイベントなどの情報ロギングイベントをバイナリログに書き込みます。この情報は、行の更新から再構築できない場合にソースに対して発行された元のクエリーを取得するなど、デバッグおよび関連する目的で使用できます。

このシステム変数のセッション値の設定は制限された操作です。セッションユーザーには、制限付きセッション変数を設定するのに十分な権限が必要です。 [セクション5.1.9.1「システム変数権限」](#)を参照してください。

これらの情報イベントは通常、バイナリログを読み取る MySQL プログラムによって無視されるため、バックアップからレプリケートまたは復元するときに問題は発生しません。これらを表示するには、`mysqlbinlog --verbose` オプションを `-vv` または `--verbose --verbose` として 2 回使用して、冗長性レベルを上げます。

- [binlog_stmt_cache_size](#)

コマンド行形式	<code>--binlog-stmt-cache-size=#</code>
システム変数	binlog_stmt_cache_size
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	32768
最小値	4096
最大値 (64 ビットプラットフォーム)	18446744073709551615
最大値 (32 ビットプラットフォーム)	4294967295

トランザクション中に発行された非トランザクションステートメントを保持するバイナリログのメモリーバッファのサイズ。 ([log_bin](#) システム変数を ON に設定して) サーバーでバイナリロギングが有効になっている場合、サーバーがトランザクションストレージエンジンをサポートしていれば、クライアントごとに個別のバイナリログトランザクションとステートメントキャッシュが割り当てられます。トランザクションで使用される非トランザクションステートメントのデータがメモリーバッファ内の領域を超えると、余分なデータが一時ファイルに格納されます。バイナリログの暗号化がサーバーでアクティブな場合、メモリーバッファは暗号化されませんが、バイナリログキャッシュの保持に使用される一時ファイルは (MySQL 8.0.17 から) 暗号化されます。各トランザクションがコミットされたあと、バイナリログステートメントキャッシュはメモリーバッファをクリアし、一時ファイルが使用されている場合は切り捨ててリセットされます。

トランザクション中に大規模な非トランザクションステートメントを頻繁に使用する場合は、一時ファイルへの書き込みを削減または排除することで、このキャッシュサイズを増やしてパフォーマンスを向上させることができます。 [Binlog_stmt_cache_use](#) および [Binlog_stmt_cache_disk_use](#) ステータス変数は、この変数のサイズを調整する場合に役立つ場合があります。 [セクション5.4.4「バイナリログ」](#)を参照してください。

[binlog_cache_size](#) システム変数はトランザクションキャッシュのサイズを設定します。

- [binlog_transaction_compression](#)

コマンド行形式	<code>--binlog-transaction-compression[={OFF ON}]</code>
導入	8.0.20
システム変数	binlog_transaction_compression
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean

デフォルト値	OFF
--------	-----

このサーバー上のバイナリログファイルに書き込まれるトランザクションの圧縮を有効にします。OFF がデフォルトです。binlog_transaction_compression_level_zstd システム変数を使用して、圧縮に使用される zstd アルゴリズムのレベルを設定します。

バイナリログのトランザクション圧縮が有効になっている場合、トランザクションペイロードは圧縮され、単一のイベント (Transaction_payload_event) としてバイナリログファイルに書き込まれます。圧縮されたトランザクションペイロードは、レプリケーションストリームでレプリカ、他のグループレプリケーショングループメンバー、または mysqlbinlog などのクライアントに送信されている間、圧縮状態のままであり、圧縮状態のままリレーログに書き込まれます。したがって、バイナリログトランザクション圧縮では、トランザクションの作成者と受信者 (およびそのバックアップ) の両方に記憶領域が節約され、トランザクションがサーバーインスタンス間で送信されるときにネットワーク帯域幅が節約されます。

binlog_transaction_compression=ON を直接有効にするには、サーバーでバイナリロギングを有効にする必要があります。MySQL サーバーインスタンスにバイナリログがない場合、MySQL 8.0.20 からのリリースであれば、binlog_transaction_compression の値に関係なく、圧縮されたトランザクションペイロードを受信、処理および表示できます。このようなサーバーインスタンスによって受信された圧縮トランザクションペイロードは、圧縮された状態でリレーログに書き込まれるため、レプリケーショントポロジ内の他のサーバーによって実行される圧縮から間接的にメリットが得られます。

このシステム変数は、トランザクションのコンテキスト内では変更できません。このシステム変数のセッション値の設定は制限された操作です。セッションユーザーには、制限付きセッション変数を設定するのに十分な権限が必要です。セクション5.1.9.1「システム変数権限」を参照してください。

バイナリログのトランザクション圧縮の詳細 (圧縮されているイベントと圧縮されていないイベントの詳細、トランザクション圧縮が使用されている場合の動作の変更など) は、セクション5.4.4.5「バイナリログトランザクション圧縮」を参照してください。

- binlog_transaction_compression_level_zstd

コマンド行形式	--binlog-transaction-compression-level-zstd=#
導入	8.0.20
システム変数	binlog_transaction_compression_level_zstd
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	3
最小値	1
最大値	22

binlog_transaction_compression システム変数によって有効になる、このサーバー上のバイナリログトランザクション圧縮の圧縮レベルを設定します。この値は、圧縮作業を決定する整数で、1 (最小作業量) から 22 (最大作業量) までです。このシステム変数を指定しない場合、圧縮レベルは 3 に設定されます。

圧縮レベルが高くなると、データ圧縮率が高くなり、トランザクションペイロードに必要なストレージ領域およびネットワーク帯域幅が削減されます。ただし、データ圧縮に必要な労力も増加し、元のサーバーでは時間と CPU およびメモリーリソースがかかります。圧縮作業の増加には、データ圧縮率の増加と線形関係はありません。

このシステム変数は、トランザクションのコンテキスト内では変更できません。このシステム変数のセッション値の設定は制限された操作です。セッションユーザーには、制限付きセッション変数を設定するのに十分な権限が必要です。セクション5.1.9.1「システム変数権限」を参照してください。

- binlog_transaction_dependency_tracking

コマンド行形式	--binlog-transaction-dependency-tracking=value
---------	--

システム変数	binlog_transaction_dependency_tracking
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	COMMIT_ORDER
有効な値	COMMIT_ORDER WRITESET WRITESET_SESSION

マルチスレッドレプリカ ([slave_parallel_workers](#) が 0 より大きい値に設定されているレプリカ) を持つレプリケーションソースサーバーの場合、[binlog_transaction_dependency_tracking](#) は、レプリカが並列で実行できるトランザクションを判断するのに役立つように、バイナリログにソースが記録する依存性情報のソースを指定します。指定可能な値は次のとおりです。

- [COMMIT_ORDER](#) : 依存性情報は、ソースコミットタイムスタンプから生成されます。これはデフォルトです。
- [WRITESET](#) : 依存性情報はソース書き込みセットから生成され、異なるタプルを書き込むトランザクションはパラレル化できます。
- [WRITESET_SESSION](#) : 依存性情報はソース書き込みセットから生成され、異なるタプルを書き込むトランザクションはパラレル化できますが、同じセッションからの 2 つの更新を並べ替えることはできません。

[WRITESET](#) および [WRITESET_SESSION](#) モードでは、[COMMIT_ORDER](#) モードで返されるものよりも最適化されていないトランザクション依存性は提供されません。[WRITESET](#) または [WRITESET_SESSION](#) を値として設定すると、ソースは、空または部分書き込みセットを持つトランザクション、主キーまたは一意キーのないテーブルを更新するトランザクション、および外部キー関係の親テーブルを更新するトランザクションに対して [COMMIT_ORDER](#) モードを使用します。

[WRITESET](#) または [WRITESET_SESSION](#) を [binlog_transaction_dependency_tracking](#) の値として設定するには、(OFF に設定されていない) アルゴリズムを指定するように [transaction_write_set_extraction](#) を設定する必要があります。MySQL 8.0 のデフォルトでは、[transaction_write_set_extraction](#) は [XXHASH64](#) に設定されています。[transaction_write_set_extraction](#) に選択した値は、[binlog_transaction_dependency_tracking](#) の値が [WRITESET](#) または [WRITESET_SESSION](#) のままである間は再度変更できません。

特定の行を変更した最新のトランザクションについて保持およびチェックされる行ハッシュの数は、[binlog_transaction_dependency_history_size](#) の値によって決まります。

グループレプリケーションの場合、[binlog_transaction_dependency_tracking=WRITESET_SESSION](#) を設定すると、グループワークロードに応じてグループメンバーのパフォーマンスを向上させることができます。Group Replication は、[binlog_transaction_dependency_tracking](#) に設定された値とは関係なく、リレーログからトランザクションを適用するときに、証明後に独自のパラレル化を実行します。ただし、[binlog_transaction_dependency_tracking](#) の値は、グループレプリケーションメンバーのバイナリログへのトランザクションの書き込み方法に影響します。これらのログ内の依存関係情報は、メンバーがグループに参加または再参加するたびに発生する分散リカバリのドナーバイナリログからの状態転送プロセスを支援するために使用されません。

- [binlog_transaction_dependency_history_size](#)

コマンド行形式	<code>--binlog-transaction-dependency-history-size=#</code>
システム変数	binlog_transaction_dependency_history_size
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ

型	Integer
デフォルト値	25000
最小値	1
最大値	1000000

メモリーに保持され、特定の行を最後に変更したトランザクションの検索に使用される行ハッシュの数の上限を設定します。このハッシュ数に達すると、履歴がページされます。

- [expire_logs_days](#)

コマンド行形式	<code>--expire-logs-days=#</code>
非推奨	はい
システム変数	expire_logs_days
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	99

バイナリログファイルを自動的に削除するまでの日数を指定します。[expire_logs_days](#) は非推奨であり、将来のリリースで削除される予定です。代わりに、バイナリログの有効期限を秒単位で設定する [binlog_expire_logs_seconds](#) を使用してください。どちらのシステム変数にも値を設定しない場合、デフォルトの有効期限は 30 日です。削除は起動時およびバイナリログがフラッシュされるときに発生する可能性があります。ログのフラッシュは、[セクション5.4「MySQL Server ログ」](#)に記載されているように発生します。

[binlog_expire_logs_seconds](#) も指定した場合、[expire_logs_days](#) にゼロ以外の値を指定すると無視され、かわりにバイナリログの有効期限として [binlog_expire_logs_seconds](#) の値が使用されます。この状況では、警告メッセージが発行されます。[expire_logs_days](#) のゼロ以外の値は、[binlog_expire_logs_seconds](#) が指定されていないか、0 として指定されている場合にのみバイナリログの有効期限として適用されます。

バイナリログの自動ページを無効にするには、[binlog_expire_logs_seconds](#) に値 0 を明示的に指定し、[expire_logs_days](#) に値を指定しないでください。以前のリリースとの互換性のために、[expire_logs_days](#) に値 0 を明示的に指定し、[binlog_expire_logs_seconds](#) に値を指定しないと、自動ページも無効になります。その場合、[binlog_expire_logs_seconds](#) のデフォルトは適用されません。

バイナリログファイルを手動で削除するには、[PURGE BINARY LOGS](#) ステートメントを使用します。[セクション13.4.1.1「PURGE BINARY LOGS ステートメント」](#)を参照してください。

- [log_bin](#)

システム変数	log_bin
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean

サーバー上のバイナリロギングのステータスを表示します。有効 (ON) または無効 (OFF) のいずれかです。バイナリロギングが有効になっている場合、サーバーは、データを変更するすべてのステートメントをバイナリログに記録します。バイナリログは、バックアップとレプリケーションに使用されます。ON はバイナリログが使用可能で

あることを意味し、**OFF** はバイナリログが使用されていないことを意味します。 `--log-bin` オプションを使用すると、バイナリログのベース名と場所を指定できます。

以前の MySQL バージョンでは、バイナリロギングはデフォルトで無効になっており、`--log-bin` オプションを指定した場合は有効になっていました。 MySQL 8.0 からは、`--log-bin` オプションを指定するかどうかに関係なく、`log_bin` システム変数が **ON** に設定されたバイナリロギングがデフォルトで有効になります。 例外は、バイナリロギングがデフォルトで無効になっている場合に、`mysqld` を使用して `--initialize` または `--initialize-insecure` オプションを指定してデータディレクトリを手動で呼び出すことによって、データディレクトリを初期化する場合です。 この場合、`--log-bin` オプションを指定することでバイナリロギングを有効にできます。

`--skip-log-bin` または `--disable-log-bin` オプションが起動時に指定された場合、バイナリロギングは無効になり、`log_bin` システム変数は **OFF** に設定されます。 これらのオプションのいずれかが指定され、`--log-bin` も指定されている場合は、後で指定するオプションが優先されます。

バイナリログの形式と管理については、[セクション5.4.4「バイナリログ」](#) を参照してください。

- `log_bin_basename`

システム変数	<code>log_bin_basename</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ファイル名

バイナリログファイルのベース名とパスを保持します。これらは `--log-bin` サーバーオプションで設定できます。 最大可変長は 256 です。 MySQL 8.0 では、`--log-bin` オプションが指定されていない場合、デフォルトのベース名は `binlog` です。 MySQL 5.7 との互換性のために、`--log-bin` オプションが文字列なしまたは空の文字列とともに指定されている場合、デフォルトのベース名は `host_name-bin` で、ホストマシンの名前が使用されます。 デフォルトの場所はデータディレクトリです。

- `log_bin_index`

コマンド行形式	<code>--log-bin-index=file_name</code>
システム変数	<code>log_bin_index</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ファイル名

バイナリログインデックスファイルのベース名とパスを保持します。これは、`--log-bin-index` サーバーオプションで設定できます。 最大可変長は 256 です。

- `log_bin_trust_function_creators`

コマンド行形式	<code>--log-bin-trust-function-creators[={OFF ON}]</code>
システム変数	<code>log_bin_trust_function_creators</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

この変数は、バイナリロギングが有効な場合に適用されます。 ストアドファンクションの作成者が、安全でないイベントがバイナリログに書き込まれる原因となるストアドファンクションを作成しないことを信頼できるかどうかを制御します。 0 (デフォルト) に設定した場合、ユーザーは `CREATE ROUTINE` または `ALTER ROUTINE` 権限に

加えて `SUPER` 権限を持たないかぎり、ストアドファンクションを作成または変更することが許可されません。0 に設定することで、関数を `DETERMINISTIC` 特性で、あるいは `READS SQL DATA` または `NO SQL` 特性で宣言する必要があるという制約も強制されます。変数が 1 に設定された場合、MySQL はストアドファンクション作成にこれらの制約を強制しません。この変数はトリガー作成にも適用されます。セクション 25.7 「ストアドプログラムバイナリロギング」を参照してください。

- `log_bin_use_v1_row_events`

コマンド行形式	<code>--log-bin-use-v1-row-events[={OFF ON}]</code>
非推奨	8.0.18
システム変数	<code>log_bin_use_v1_row_events</code>
スコープ	グローバル
動的	いいえ
<code>SET_VAR</code> ヒントの適用	いいえ
型	Boolean
デフォルト値	<code>OFF</code>

この読み取り専用システム変数は非推奨です。サーバーの起動時にシステム変数を `ON` に設定すると、MySQL 5.6 の時点でデフォルトであるバージョン 2 バイナリログ行イベントではなく、バージョン 1 バイナリログ行イベントを使用してバイナリログを書き込むことによって、MySQL Server 5.5 以前を実行しているレプリカで行ベースレプリケーションが有効になります。

- `log_slave_updates`

コマンド行形式	<code>--log-slave-updates[={OFF ON}]</code>
システム変数	<code>log_slave_updates</code>
スコープ	グローバル
動的	いいえ
<code>SET_VAR</code> ヒントの適用	いいえ
型	Boolean
デフォルト値	<code>ON</code>

レプリケーションソースサーバーから受信した更新を、レプリカ独自のバイナリログに記録するかどうか。

この変数を有効にすると、レプリカはソースから受信し、レプリケーション SQL スレッドによって実行された更新をレプリカ独自のバイナリログに書き込みます。バイナリロギングは `--log-bin` オプションによって制御され、デフォルトで有効になっていますが、更新をログに記録するにはレプリカでも有効にする必要があります。セクション 17.1.6 「レプリケーションおよびバイナリロギングのオプションと変数」を参照してください。`--skip-log-bin` を指定してバイナリロギングを無効にしないかぎり、`log_slave_updates` はデフォルトで有効になっています。この場合、MySQL はレプリカ更新ロギングもデフォルトで無効にします。バイナリロギングが有効になっているときにレプリカ更新ロギングを無効にする必要がある場合は、レプリカサーバーの起動時に `--log-slave-updates=OFF` を指定します。

`log_slave_updates` を有効にすると、レプリケーションサーバーをチェーンできます。たとえば、このようにレプリケーションサーバーをセットアップするとします。

```
A -> B -> C
```

ここで、`A` はレプリカ `B` のソースとして機能し、`B` はレプリカ `C` のソースとして機能します。これが機能するには、`B` がソースとレプリカの両方である必要があります。バイナリロギングが有効で、`log_slave_updates` が有効になっている場合 (デフォルト設定)、`A` から受信した更新は `B` によってバイナリログに記録されるため、`C` に渡すことができます。

- `log_statements_unsafe_for_binlog`

コマンド行形式	<code>--log-statements-unsafe-for-binlog[={OFF ON}]</code>
---------	--

システム変数	log_statements_unsafe_for_binlog
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

エラー 1592 が発生した場合、生成された警告をエラーログに追加するかどうかを制御します。

- [master_verify_checksum](#)

コマンド行形式	<code>--master-verify-checksum[={OFF ON}]</code>
システム変数	master_verify_checksum
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

この変数を有効にすると、ソースはチェックサムを検査してバイナリログから読み取られたイベントを検証し、不一致が発生した場合はエラーで停止します。[master_verify_checksum](#) はデフォルトで無効になっています。この場合、ソースはバイナリログのイベント長を使用してイベントを検証し、バイナリログから完全なイベントのみが読み取られるようにします。

- [max_binlog_cache_size](#)

コマンド行形式	<code>--max-binlog-cache-size=#</code>
システム変数	max_binlog_cache_size
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	18446744073709551615
最小値	4096
最大値	18446744073709551615

トランザクション内の非トランザクションステートメントがこのバイト数より多くのメモリーを必要とする場合、サーバーは `Multi-statement transaction required more than 'max_binlog_cache_size' bytes of storage` エラーを生成します。最小値は 4096 です。可能な最大値は 16EiB (exbibytes) です。推奨される最大値は 4G バイトです。これは、MySQL が現在 4G バイトより大きなバイナリログ位置で作業できないという事実によります。

[max_binlog_cache_size](#) はトランザクションキャッシュのみのサイズを設定します。ステートメントキャッシュの上限値は [max_binlog_stmt_cache_size](#) システム変数によって管理されます。

[max_binlog_cache_size](#) のセッションの可視性は、[binlog_cache_size](#) システム変数の可視性と一致します。つまり、その値を変更すると、値が変更された後に開始される新しいセッションにのみ影響します。

- [max_binlog_size](#)

コマンド行形式	<code>--max-binlog-size=#</code>
システム変数	max_binlog_size

スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1073741824
最小値	4096
最大値	1073741824

バイナリログへの書き込みによって、現在のログファイルサイズがこの変数の値を超えた場合、サーバーはバイナリログをローテーションします (現在のファイルを閉じて、新しいものを開きます)。最小値は 4096 バイトです。最大値およびデフォルト値は 1G バイトです。暗号化バイナリログファイルには、`max_binlog_size` に含まれる追加の 512 バイトヘッダーがあります。

トランザクションはバイナリログにひとまとまりで書き込まれ、複数のバイナリログ間に分割されることはありません。このため、大きなトランザクションの場合、`max_binlog_size` より大きいバイナリログファイルが見られることがあります。

`max_relay_log_size` が 0 の場合、`max_binlog_size` の値がリレーログにも適用されます。

GTID がサーバーで使用されている場合、`max_binlog_size` に到達したときに、システムテーブル `mysql.gtid_executed` にアクセスして GTID を現在のバイナリログファイルから書き込めないと、バイナリログをローテーションできません。この状況では、サーバーはその `binlog_error_action` 設定に従って応答します。`IGNORE_ERROR` が設定されている場合、サーバーにエラーが記録され、バイナリロギングが停止されます。または、`ABORT_SERVER` が設定されている場合、サーバーはシャットダウンします。

- `max_binlog_stmt_cache_size`

コマンド行形式	<code>--max-binlog-stmt-cache-size=#</code>
システム変数	<code>max_binlog_stmt_cache_size</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	18446744073709547520
最小値	4096
最大値	18446744073709547520

トランザクション内の非トランザクションステートメントがこのバイト数より多くのメモリーを必要とする場合、サーバーはエラーを生成します。最小値は 4096 です。最大値およびデフォルト値は、32 ビットプラットフォームでは 4G バイト、64 ビットプラットフォームでは 16E バイト (エクサバイト) です。

`max_binlog_stmt_cache_size` はステートメントキャッシュのみのサイズを設定します。トランザクションキャッシュの上限値は `max_binlog_cache_size` システム変数によって排他的に管理されます。

- `original_commit_timestamp`

システム変数	<code>original_commit_timestamp</code>
スコープ	セッション
動的	はい
SET_VAR ヒントの適用	いいえ

型	数値
---	----

レプリケーションによる内部使用用。レプリカでトランザクションを再実行する場合、これは元のソースでトランザクションがコミットされた時間に設定され、エポック以降マイクロ秒単位で測定されます。これにより、元のコミットタイムスタンプをレプリケーショントポロジ全体に伝播できます。

このシステム変数のセッション値の設定は制限された操作です。セッションユーザーには、[REPLICATION_APPLIER](#) 権限 ([セクション17.3.3「レプリケーション権限チェック」](#) を参照) または制限付きセッション変数の設定に十分な権限 ([セクション5.1.9.1「システム変数権限」](#) を参照) が必要です。ただし、この変数はユーザーが設定するためのものではなく、レプリケーションインフラストラクチャによって自動的に設定されることに注意してください。

- [sql_log_bin](#)

システム変数	sql_log_bin
スコープ	セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

この変数は、現在のセッションでバイナリログへのロギングを有効にするかどうかを制御します (バイナリログ自体が有効になっていると仮定します)。デフォルト値は [ON](#) です。現在のセッションのバイナリロギングを無効または有効にするには、セッション [sql_log_bin](#) 変数を [OFF](#) または [ON](#) に設定します。

レプリカにレプリケートしないソースに変更を加えている間にバイナリロギングを一時的に無効にするには、セッションに対してこの変数を [OFF](#) に設定します。

このシステム変数のセッション値の設定は制限された操作です。セッションユーザーには、制限付きセッション変数を設定するのに十分な権限が必要です。 [セクション5.1.9.1「システム変数権限」](#) を参照してください。

トランザクションまたはサブクエリー内で [sql_log_bin](#) のセッション値を設定することはできません。

「この変数を [OFF](#) に設定すると、GTID がバイナリログ内のトランザクションに割り当てられなくなります」。これは、GTID をレプリケーションに使用している場合、バイナリロギングがあとで再度有効になった場合でも、この時点からログに書き込まれる GTID はその意味で発生したトランザクションを考慮しないため、それらのトランザクションは失われることを意味します。

- [sync_binlog](#)

コマンド行形式	--sync-binlog=#
システム変数	sync_binlog
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1
最小値	0
最大値	4294967295

MySQL サーバーがバイナリログをディスクに同期する頻度を制御します。

- [sync_binlog=0](#): MySQL サーバーによるバイナリログのディスクへの同期を無効にします。代わりに、MySQL サーバーは、ほかのファイルの場合と同様に、バイナリログをディスクにフラッシュするためにオペレーティングシステムに依存します。この設定は最高のパフォーマンスを提供しますが、電源障害またはオペレーティン

システムがクラッシュした場合は、バイナリログに同期されていないトランザクションがサーバーによってコミットされている可能性があります。

- `sync_binlog=1`: トランザクションがコミットされる前にバイナリログをディスクに同期できるようにします。これは最も安全な設定ですが、ディスク書き込み数が増加したためにパフォーマンスに悪影響を与える可能性があります。電源障害またはオペレーティングシステムがクラッシュした場合、バイナリログから欠落しているトランザクションは準備完了状態にすぎません。これにより、自動回復ルーチンはトランザクションをロールバックでき、バイナリログからトランザクションが失われないことが保証されます。
- `sync_binlog=N` (N は 0 または 1 以外の値): バイナリログは、 N バイナリログコミットグループが収集されたあとにディスクに同期されます。電源障害またはオペレーティングシステムがクラッシュした場合は、バイナリログにフラッシュされていないトランザクションがサーバーによってコミットされている可能性があります。この設定は、ディスク書き込み数が増加したため、パフォーマンスに悪影響を与える可能性があります。値を大きくするとパフォーマンスは向上しますが、データ損失のリスクは高くなります。

トランザクションで `InnoDB` を使用するレプリケーション設定で永続性と一貫性を最大限に高めるには、次の設定を使用します:

- `sync_binlog=1`。
- `innodb_flush_log_at_trx_commit=1`。

注意

多くのオペレーティングシステムや一部のディスクハードウェアは、ディスクへのフラッシュ操作を行なったと欺きます。フラッシュが行われていなくても、行われたと `mysqld` に通知される可能性があります。この場合、推奨設定であってもトランザクションの永続性は保証されず、最悪の場合は停電によって `InnoDB` データが破損する可能性があります。バッテリーバックアップのディスクキャッシュを SCSI ディスクコントローラ内やディスク自体で使用すると、ファイルフラッシュの速度が上がり、操作が安全になります。ハードウェアキャッシュ内のディスク書き込みのキャッシュを無効にすることもできます。

- `transaction_write_set_extraction`

コマンド行形式	<code>--transaction-write-set-extraction[=value]</code>
システム変数	<code>transaction_write_set_extraction</code>
スコープ	グローバル、セッション
動的	はい
<code>SET_VAR</code> ヒントの適用	いいえ
型	列挙
デフォルト値	<code>XXHASH64</code>
有効な値	<code>OFF</code> <code>MURMUR32</code> <code>XXHASH64</code>

マルチスレッドレプリカ (`slave_parallel_workers` が 0 より大きい値に設定されているレプリカ) を持つレプリケーションソースサーバーの場合、`binlog_transaction_dependency_tracking` は `WRITESET` または `WRITESET_SESSION` に設定され、ソース書き込みセットから依存性情報を生成します。`transaction_write_set_extraction` は、トランザクション中に抽出された書き込みのハッシュに使用されるアルゴリズムを指定します。このシステム変数の値を変更するには、`binlog_format` を `ROW` に設定する必要があります。

`WRITESET` または `WRITESET_SESSION` が `binlog_transaction_dependency_tracking` の値として設定されている場合、`transaction_write_set_extraction` を設定してアルゴリズムを指定する必要があります (`OFF` に設定されていません)。MySQL 8.0 のデフォルトでは、`transaction_write_set_extraction` は `XXHASH64` に設定されてい

ます。 `binlog_transaction_dependency_tracking` の現在の値は `WRITESET` または `WRITESET_SESSION` ですが、`transaction_write_set_extraction` の値は変更できません。

グループレプリケーションの場合、`transaction_write_set_extraction` を `XXHASH64` に設定する必要があります。トランザクションから書き込みを抽出するプロセスは、すべてのグループメンバーの競合検出および証明のためにグループレプリケーションで使用されます。 [セクション18.9.1「グループレプリケーションの要件」](#) を参照してください。

MySQL 8.0.14 では、このシステム変数のセッション値の設定は制限された操作です。セッションユーザーには、制限付きセッション変数を設定するのに十分な権限が必要です。 [セクション5.1.9.1「システム変数権限」](#) を参照してください。

17.1.6.5 グローバルトランザクション ID システム変数

このセクションで説明する MySQL Server システム変数は、グローバルトランザクション識別子 (GTID) をモニターおよび制御するために使用されます。追加情報については [セクション17.1.3「グローバルトランザクション識別子を使用したレプリケーション」](#) を参照してください。

- `binlog_gtid_simple_recovery`

コマンド行形式	<code>--binlog-gtid-simple-recovery[={OFF ON}]</code>
システム変数	<code>binlog_gtid_simple_recovery</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

この変数は、MySQL の起動または再起動時に GTID の検索中にバイナリログファイルがどのように繰り返されるかを制御します。

MySQL 8.0 のデフォルトである `binlog_gtid_simple_recovery=TRUE` の場合、`gtid_executed` および `gtid_purged` の値は、最新のバイナリログファイルともっとも古いバイナリログファイルの `Previous_gtids_log_event` の値に基づいて起動時に計算されます。計算の詳細は、[gtid_purged システム変数](#) を参照してください。この設定は、サーバーの再起動時に 2 つのバイナリログファイルにのみアクセスします。サーバー上のすべてのバイナリログが MySQL 5.7.8 以降を使用して生成された場合、`binlog_gtid_simple_recovery=TRUE` は常に安全に使用できます。

MySQL 5.7.7 以前のバイナリログがサーバーに存在する場合 (たとえば、古いサーバーから MySQL 8.0 へのアップグレード後)、`binlog_gtid_simple_recovery=TRUE` を使用すると、次の 2 つの状況で `gtid_executed` および `gtid_purged` が正しく初期化されないことがあります:

- 最新のバイナリログは MySQL 5.7.5 以前によって生成され、`gtid_mode` は一部のバイナリログでは `ON` でしたが、最新のバイナリログでは `OFF` でした。
- MySQL 5.7.7 以前で `SET @@GLOBAL.gtid_purged` ステートメントが発行され、`SET @@GLOBAL.gtid_purged` ステートメントの時点でアクティブだったバイナリログはまだパーシされていません。

いずれかの状況で不正な GTID セットが計算された場合、あとで `binlog_gtid_simple_recovery=FALSE` を使用してサーバーを再起動しても、正しくありません。これらの状況のいずれかが適用されるか、サーバーに適用される可能性がある場合は、サーバーを起動または再起動する前に `binlog_gtid_simple_recovery=FALSE` を設定します。

`binlog_gtid_simple_recovery=FALSE` が設定されている場合、`gtid_purged システム変数` で説明されている `gtid_executed` および `gtid_purged` の計算方法は、次のようにバイナリログファイルを繰り返すように変更されます:

- 最新のバイナリログファイルからの `Previous_gtids_log_event` および GTID ログイベントの値を使用する代わりに、`gtid_executed` の計算は最新のバイナリログファイルから繰り返され、`Previous_gtids_log_event` の値と、`Previous_gtids_log_event` 値が見つかった最初のバイナリログファイルからの GTID ログイベントを使用します。サーバーの最新のバイナリログファイルに GTID ログイベントがない場合 (たとえば、`gtid_mode=ON` が使

用されたが、あとでサーバーが `gtid_mode=OFF` に変更された場合)、このプロセスには時間がかかることがあります。

- もっとも古いバイナリログファイルからの `Previous_gtid_log_event` の値を使用する代わりに、`gtid_purged` の計算はもっとも古いバイナリログファイルから繰り返され、空でない `Previous_gtid_log_event` 値または GTID ログイベント (GTID の使用がその時点から開始されることを示す) が見つかった最初のバイナリログファイルからの `Previous_gtid_log_event` の値を使用します。サーバーの古いバイナリログファイルに GTID ログイベントがない場合 (たとえば、`gtid_mode=ON` がサーバー上で最近のみ設定された場合)、このプロセスには時間がかかることがあります。
- `enforce_gtid_consistency`

コマンド行形式	<code>--enforce-gtid-consistency[=value]</code>
システム変数	<code>enforce_gtid_consistency</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	OFF
有効な値	OFF ON WARN

この変数の値に応じて、GTID を使用して安全にログに記録できるステートメントのみを実行できるようにすることで、サーバーは GTID 整合性を強制します。GTID ベースのレプリケーションを有効にする前に、この変数を `ON` に設定する必要があります。

`enforce_gtid_consistency` で構成できる値は次のとおりです:

- `OFF`: すべてのトランザクションが GTID 整合性に違反することを許可されます。
- `ON`: GTID 整合性に違反するトランザクションは許可されていません。
- `WARN`: すべてのトランザクションは GTID 整合性に違反できますが、この場合は警告が生成されます。

`--enforce-gtid-consistency` は、ステートメントに対してバイナリロギングが行われた場合にのみ有効になります。バイナリロギングがサーバーで無効になっている場合、またはステートメントがフィルタによって削除されたためにバイナリログに書き込まれない場合、GTID の整合性は、ログに記録されていないステートメントに対してチェックまたは適用されません。

`enforce_gtid_consistency` が `ON` に設定されている場合、GTID セーフステートメントを使用してログに記録できるステートメントのみが記録されるため、ここにリストされている操作はこのオプションとともに使用できません:

- トランザクション内の `CREATE TEMPORARY TABLE` または `DROP TEMPORARY TABLE` ステートメント。
- トランザクションおよび非トランザクションテーブルの両方を更新するトランザクションまたはステートメント。すべての非トランザクションテーブルが一時テーブルの場合、非トランザクション DML は同じトランザクションまたはトランザクション DML と同じステートメントで許可されるという例外があります。
- MySQL 8.0.21 より前の `CREATE TABLE ... SELECT` ステートメント。MySQL 8.0.21 からは、アトミック DDL をサポートするストレージエンジンに対して `CREATE TABLE ... SELECT` ステートメントが許可されます。

詳細は、[セクション17.1.3.7「GTID ベースレプリケーションの制約」](#)を参照してください。

MySQL 5.7 より前およびそのリリースシリーズの初期リリースでは、ブール `enforce_gtid_consistency` は `OFF` にデフォルト設定されていました。これらの以前のリリースとの互換性を維持するために、列挙はデフォルトで `OFF` に設定され、値を指定せずに `--enforce-gtid-consistency` を設定すると、

値が **ON** に設定されたと解釈されます。変数には、値に対する複数のテキストエイリアスもあります: **0=OFF=FALSE**、**1=ON=TRUE**、**2=WARN**。これは他の列挙型とは異なりますが、以前のリリースで使用されていたブール型との互換性は維持されます。これらの変更は、変数によって返される内容に影響します。 **SELECT @@ENFORCE_GTID_CONSISTENCY**、**SHOW VARIABLES LIKE 'ENFORCE_GTID_CONSISTENCY'** および **SELECT * FROM INFORMATION_SCHEMA.VARIABLES WHERE 'VARIABLE_NAME' = 'ENFORCE_GTID_CONSISTENCY'** を使用すると、すべて数値フォームではなくテキストフォームが返されます。 **@@ENFORCE_GTID_CONSISTENCY** はブールの数値フォームを返しますが、**SHOW** および情報スキーマのテキストフォームを返すため、これは互換性のない変更です。

- [gtid_executed](#)

システム変数	gtid_executed
システム変数	gtid_executed
スコープ	グローバル
スコープ	グローバル、セッション
動的	いいえ
動的	いいえ
SET_VAR ヒントの適用	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列
単位	set of GTIDs

グローバルスコープで使用する場合、この変数には、**SET gtid_purged** ステートメントによって設定されたサーバーおよび GTID で実行されたすべてのトランザクションのセットの表現が含まれます。これは、**SHOW MASTER STATUS** および **SHOW REPLICA | SLAVE STATUS** の出力の **Executed_Gtid_Set** カラムの値と同じです。この変数の値は GTID セットです。詳細は、[GTID セット](#) を参照してください。

サーバーが起動すると、**@@GLOBAL.gtid_executed** が初期化されます。バイナリログを反復して **gtid_executed** に移入する方法の詳細は、[binlog_gtid_simple_recovery](#) を参照してください。GTID は、トランザクションの実行時、または **SET gtid_purged** ステートメントの実行時にセットに追加されます。

任意の時点でバイナリログに存在できるトランザクションのセットは、**GTID_SUBTRACT(@@GLOBAL.gtid_executed, @@GLOBAL.gtid_purged)** と同等です。つまり、まだパーズされていないバイナリログ内のすべてのトランザクションに相当します。

RESET MASTER を発行することで、この変数のグローバル値（ただし、セッション値ではない）は空の文字列にリセットされます。GTID は、セットが **RESET MASTER** によってクリアされるのを除いてセットから削除されません。

一部の旧リリースでは、この変数はセッションスコープとともに使用することもできます。セッションスコープには、現在のセッションでキャッシュに書き込まれる一連のトランザクションの表現が含まれていました。セッションスコープは非推奨になりました。

- [gtid_executed_compression_period](#)

コマンド行形式	<code>--gtid-executed-compression-period=#</code>
システム変数	gtid_executed_compression_period
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値 (≥ 8.0.23)	0
デフォルト値 (≤ 8.0.22)	1000

最小値	0
最大値	4294967295

この数のトランザクションが処理されるたびに、`mysql.gtid_executed` テーブルを圧縮します。サーバーでバイナリロギングが有効になっている場合、この圧縮方法は使用されず、代わりに `mysql.gtid_executed` テーブルはバイナリログのローテーションごとに圧縮されます。バイナリロギングがサーバーで無効になっている場合、圧縮スレッドは、指定された数のトランザクションが実行されるまでスリープしてから、ウェイクアップして `mysql.gtid_executed` テーブルの圧縮を実行します。このシステム変数の値を 0 に設定すると、スレッドはウェイクアップしないため、この明示的な圧縮方法は使用されません。かわりに、圧縮は必要に応じて暗黙的に行われます。

MySQL 8.0.17 から、InnoDB トランザクションは、InnoDB 以外のトランザクションに対する個別のプロセスによって `mysql.gtid_executed` テーブルに書き込まれます。サーバーに InnoDB トランザクションと InnoDB 以外のトランザクションが混在している場合、このシステム変数によって制御される圧縮はこのプロセスの作業を妨げ、処理速度が大幅に低下する可能性があります。このため、そのリリースから、`gtid_executed_compression_period` を 0 に設定することをお勧めします。

MySQL 8.0.23 から、InnoDB および InnoDB 以外のトランザクションが同じプロセスで `mysql.gtid_executed` テーブルに書き込まれ、`gtid_executed_compression_period` のデフォルト値は 0 です。

詳しくは [mysql.gtid_executed テーブル圧縮](#) をご覧ください。

- `gtid_mode`

コマンド行形式	<code>--gtid-mode=MODE</code>
システム変数	<code>gtid_mode</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	OFF
有効な値	OFF OFF_PERMISSIVE ON_PERMISSIVE ON

GTID ベースのロギングを有効にするかどうか、およびログに含めることができるトランザクションのタイプを制御します。グローバルシステム変数を設定するには、十分な権限が必要です。 [セクション5.1.9.1「システム変数権限」](#) を参照してください。 `gtid_mode=ON` を設定するには、`enforce_gtid_consistency` が true である必要があります。この変数を変更する前に、 [セクション17.1.4「オンラインサーバーでの GTID モードの変更」](#) を参照してください。

ログに記録されるトランザクションは、匿名にすることも GTID を使用することもできます。匿名トランザクションは、特定のトランザクションを識別するためにバイナリログファイルと位置に依存します。GTID トランザクションには、トランザクションの参照に使用される一意の識別子があります。様々なモードは次のとおりです:

- **OFF**: 新規トランザクションとレプリケートされたトランザクションの両方が匿名である必要があります。
- **OFF_PERMISSIVE**: 新しいトランザクションは匿名です。レプリケートされたトランザクションは、匿名トランザクションまたは GTID トランザクションのいずれかです。
- **ON_PERMISSIVE**: 新規トランザクションは GTID トランザクションです。レプリケートされたトランザクションは、匿名トランザクションまたは GTID トランザクションのいずれかです。

- **ON**: 新規トランザクションとレプリケートされたトランザクションの両方が GTID トランザクションである必要があります。

ある値から別の値への変更は、一度に 1 つのステップにしかできません。たとえば、`gtid_mode` が現在 `OFF_PERMISSIVE` に設定されている場合、`OFF` または `ON_PERMISSIVE` に変更できますが、`ON` には変更できません。

`gtid_purged` および `gtid_executed` の値は、`gtid_mode` の値に関係なく永続的です。したがって、`gtid_mode` の値を変更した後も、これらの変数には正しい値が含まれます。

- `gtid_next`

システム変数	<code>gtid_next</code>
スコープ	セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	<code>AUTOMATIC</code>
有効な値	<code>AUTOMATIC</code> <code>ANONYMOUS</code> <code>UUID:NUMBER</code>

この変数は、次の GTID を取得するかどうかとその取得方法を指定するために使用されます。

このシステム変数のセッション値の設定は制限された操作です。セッションユーザーには、`REPLICATION_APPLIER` 権限 (セクション 17.3.3 「レプリケーション権限チェック」を参照) または制限付きセッション変数の設定に十分な権限 (セクション 5.1.9.1 「システム変数権限」を参照) が必要です。

`gtid_next` では、次のいずれかの値を使用できます:

- **AUTOMATIC**: 自動的に生成される次のグローバルトランザクション ID を使用します。
- **ANONYMOUS**: トランザクションはグローバル識別子を持たず、ファイルと位置のみで識別されます。
- **UUID:NUMBER** 形式のグローバルトランザクション ID。

前述のどのオプションが有効かは、`gtid_mode` の設定によって異なります。詳細は、セクション 17.1.4.1 「レプリケーションモードの概念」を参照してください。`gtid_mode` が `OFF` の場合、この変数を設定しても効果はありません。

この変数が `UUID` に設定された後:`NUMBER` では、トランザクションがコミットまたはロールバックされている場合、他のステートメントの前に明示的な `SET GTID_NEXT` ステートメントを再発行する必要があります。

`DROP TABLE` または `DROP TEMPORARY TABLE` は、非一時テーブルと一時テーブルの組み合わせ、または非トランザクションストレージエンジンを使用する一時テーブルとトランザクションストレージエンジンを使用する一時テーブルの組み合わせで使用すると、明示的なエラーで失敗します。

- `gtid_owned`

システム変数	<code>gtid_owned</code>
スコープ	グローバル、セッション
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

単位	set of GTIDs
----	--------------

この読み取り専用変数は、主に内部で使用されます。その内容はスコープによって異なります。

- グローバルスコープで使用される場合、`gtid_owned` は、サーバーで現在使用されているすべての GTID のリストを、それらを所有するスレッドの ID とともに保持します。この変数は主に、マルチスレッドレプリカがトランザクションがすでに別のスレッドに適用されているかどうかをチェックする場合に役立ちます。アップライアスレッドは、トランザクションを処理している間は常にトランザクション GTID の所有権を取得するため、`@@global.gtid_owned` には処理中の GTID と所有者が表示されます。トランザクションがコミット (またはロールバック) されると、適用者スレッドは GTID の所有権を解放します。
- セッションスコープとともに使用された場合、`gtid_owned` は、このセッションによって現在使用されており、所有されている単一の GTID を保持します。この変数は主に、クライアントが `gtid_next` を設定してトランザクションに GTID を明示的に割り当てたときに GTID の使用をテストおよびデバッグする場合に役立ちます。この場合、`@@session.gtid_owned` は、トランザクションがコミット (またはロールバック) されるまで、クライアントがトランザクションを処理している間は常に GTID を表示します。クライアントがトランザクションの処理を終了すると、変数はクリアされます。セッションに `gtid_next=AUTOMATIC` が使用されている場合、`gtid_owned` はトランザクションのコミットステートメントの実行中に短時間のみ移入されるため、適切な時点で `@@global.gtid_owned` が読み取られるかどうかはリストされますが、対象のセッションからは監視できません。セッションでクライアントによって処理される GTID を追跡する必要がある場合は、`session_track_gtid` システム変数によって制御されるセッション状態トラッカを有効にできます。
- `gtid_purged`

システム変数	<code>gtid_purged</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
単位	set of GTIDs

`gtid_purged` システム変数 (`@@GLOBAL.gtid_purged`) のグローバル値は GTID セットで、サーバー上でコミットされたが、サーバー上のバイナリログファイルには存在しないすべてのトランザクションの GTID で構成されます。`gtid_purged` は、`gtid_executed` のサブセットです。GTID の次のカテゴリが `gtid_purged` にあります:

- レプリカでバイナリロギングを無効にしてコミットされたレプリケートされたトランザクションの GTID。
- 現在パージされているバイナリログファイルに書き込まれたトランザクションの GTID。
- ステートメント `SET @@GLOBAL.gtid_purged` によってセットに明示的に追加された GTID。

サーバーが起動すると、`gtid_purged` のグローバル値は GTID のセットに初期化されます。この GTID セットの計算方法については、`gtid_purged` システム変数を参照してください。MySQL 5.7.7 以前のバイナリログがサーバーに存在する場合は、サーバー構成ファイルで `binlog_gtid_simple_recovery=FALSE` を設定して、正しい計算を生成する必要がある場合があります。この設定が必要な状況の詳細は、`binlog_gtid_simple_recovery` の説明を参照してください。

`RESET MASTER` を発行すると、`gtid_purged` の値が空の文字列にリセットされます。

特定の GTID セット内のトランザクションが適用されたことをサーバーに記録するために、`gtid_purged` の値を設定できますが、それらはサーバー上のバイナリログには存在しません。このアクションのユースケースの例は、サーバー上の 1 つ以上のデータベースのバックアップをリストアするが、サーバー上のトランザクションを含む関連するバイナリログがない場合です。

重要

GTID は、符号付き 64 ビット整数 (2 から 63 の累乗から 1 を引いた数) の負でない値までのサーバーインスタンスでのみ使用できます。`gtid_purged` の値をこの制限に近づく数に設定すると、後続のコミットによってサーバーで GTID が不足し、`binlog_error_action` で

指定されたアクションが実行される可能性があります。MySQL 8.0.23 からは、サーバーインスタンスが制限に近づいたときに警告メッセージが発行されます。

MySQL 8.0 からは、`gtid_purged` の値を設定する方法が 2 つあります。`gtid_purged` の値を指定した GTID セットに置き換えるか、`gtid_purged` によってすでに保持されている GTID セットに指定した GTID セットを追加できます。サーバーに既存の GTID がない場合 (たとえば、既存のデータベースのバックアップをプロビジョニングする空のサーバーがある場合)、両方の方法で同じ結果が得られます。サーバー上にすでに存在するトランザクションと重複するバックアップを復元する場合、たとえば、破損したテーブルを `mysqldump` を使用して作成されたソースからの部分的なダンプで置き換える場合 (ダンプが部分的であっても、サーバー上のすべてのトランザクションの GTID を含む)、`gtid_purged` の値を置き換える最初の方法を使用します。サーバー上にすでに存在するトランザクションとは無関係なバックアップをリストアする場合 (たとえば、2 つの異なるサーバーからのダンプを使用してマルチソースレプリカをプロビジョニングする場合)、`gtid_purged` の値に追加する別の方法を使用します。

- `gtid_purged` の値を指定した GTID セットに置き換えるには、次のステートメントを使用します:

```
SET @@GLOBAL.gtid_purged = 'gtid_set'
```

`gtid_set` は、`gtid_purged` の現在の値のスーパーセットである必要があります。つまり、新しい GTID セットには、`gtid_subtract(gtid_executed,gtid_purged)` と交差してはなりません。つまり、新しい GTID セットには、`gtid_purged` にすでに存在する GTID が含まれている必要があります。まだパージされていない GTID を `gtid_executed` に含めることはできません。`gtid_set` には、`@@global.gtid_owned` にある GTID、つまりサーバーで現在処理されているトランザクションの GTID も含めることができません。

その結果、`gtid_purged` のグローバル値は `gtid_set` に設定され、`gtid_executed` の値は `gtid_set` と `gtid_executed` の以前の値の和集合になります。

- 指定した GTID セットを `gtid_purged` に追加するには、GTID セットの前にプラス記号 (+) を付けて次のステートメントを使用します:

```
SET @@GLOBAL.gtid_purged = '+gtid_set'
```

`gtid_set` は、`gtid_executed` の現在の値と交差できません。つまり、新しい GTID セットには、`gtid_purged` にもすでに存在するトランザクションを含め、`gtid_executed` の GTID を含めないでください。`gtid_set` には、`@@global.gtid_owned` にある GTID、つまりサーバーで現在処理されているトランザクションの GTID も含めることができません。

その結果、`gtid_set` が `gtid_executed` と `gtid_purged` の両方に追加されます。

注記

MySQL 5.7.7 以前のバイナリログがサーバーに存在する場合 (たとえば、古いサーバーから MySQL 8.0 へのアップグレード後)、`SET @@GLOBAL.gtid_purged` ステートメントの発行後、サーバーを再起動する前にサーバー構成ファイルで `binlog_gtid_simple_recovery=FALSE` を設定する必要がある場合があります。そうしないと、`gtid_purged` が正しく計算されない可能性があります。この設定が必要な状況の詳細は、`binlog_gtid_simple_recovery` の説明を参照してください。

17.1.7 一般的なレプリケーション管理タスク

レプリケーションが開始されると、定期的に管理しなくても実行されます。このセクションでは、レプリケーションのステータスを確認する方法、レプリカを一時停止する方法、およびレプリカで失敗したトランザクションをスキップする方法について説明します。

ヒント

MySQL の複数のインスタンスをデプロイするには、`MySQL Shell` で MySQL サーバーインスタンスのグループを簡単に管理できるようにする `InnoDB クラスタ` を使用できます。InnoDB クラスタは MySQL Group Replication をプログラム環境でラップするため、MySQL インスタンスのクラスタを簡単にデプロイして高可用性を実現できます。また、InnoDB クラスタは `MySQL Router` とシームレスにインターフェースするため、アプリケーションは独自のフェイルオーバープロセスを記述せずにクラスタに接続できます。た

ただし、高可用性を必要としない同様のユースケースでは、[InnoDB ReplicaSet](#) を使用できません。MySQL Shell のインストール手順は、[here](#) にあります。

17.1.7.1 レプリケーションステータスの確認

レプリケーションプロセスを管理する場合の最も一般的なタスクは、レプリケーションが実行され、レプリカとソースの間にエラーがないことを確認することです。

各レプリカで実行する必要がある `SHOW REPLICA | SLAVE STATUS` ステートメントは、レプリカサーバーとソースサーバー間の接続の構成およびステータスに関する情報を提供します。MySQL 8.0.22 から、`SHOW SLAVE STATUS` は非推奨になり、かわりに `SHOW REPLICA STATUS` を使用できます。パフォーマンススキーマには、この情報をよりアクセスしやすい形式で提供するレプリケーションテーブルがあります。[セクション27.12.11「パフォーマンススキーマレプリケーションテーブル」](#)を参照してください。

パフォーマンススキーマレプリケーションテーブルに表示されるレプリケーションハートビート情報を使用すると、ソースが最近レプリカにイベントを送信していない場合でも、レプリケーション接続がアクティブであることを確認できます。ソースは、ハートビート間隔より長い期間バイナリログに更新がなく、未送信のイベントがない場合に、ハートビート信号をレプリカに送信します。(`CHANGE MASTER TO` ステートメントで設定された) ソースの `MASTER_HEARTBEAT_PERIOD` 設定では、ハートビートの頻度を指定します。これは、レプリカ (`slave_net_timeout`) の接続タイムアウト間隔の半分にデフォルト設定されます。[replication_connection_status](#) 「パフォーマンススキーマ」テーブルには、レプリカが最新のハートビート信号をいつ受信したか、および受信したハートビート信号の数が表示されます。

`SHOW REPLICA | SLAVE STATUS` ステートメントを使用して個々のレプリカのステータスを確認している場合、ステートメントは次の情報を提供します:

```
mysql> SHOW REPLICA STATUS\G
***** 1. row *****
    Replica_IO_State: Waiting for master to send event
      Source_Host: source1
      Source_User: root
      Source_Port: 3306
      Connect_Retry: 60
      Source_Log_File: mysql-bin.000004
      Read_Source_Log_Pos: 931
      Relay_Log_File: replica1-relay-bin.000056
      Relay_Log_Pos: 950
      Relay_Source_Log_File: mysql-bin.000004
      Replica_IO_Running: Yes
      Replica_SQL_Running: Yes
      Replicate_Do_DB:
      Replicate_Ignore_DB:
      Replicate_Do_Table:
      Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
      Replicate_Wild_Ignore_Table:
      Last_Errno: 0
      Last_Error:
      Skip_Counter: 0
      Exec_Source_Log_Pos: 931
      Relay_Log_Space: 1365
      Until_Condition: None
      Until_Log_File:
      Until_Log_Pos: 0
      Source_SSL_Allowed: No
      Source_SSL_CA_File:
      Source_SSL_CA_Path:
      Source_SSL_Cert:
      Source_SSL_Cipher:
      Source_SSL_Key:
      Seconds_Behind_Source: 0
      Source_SSL_Verify_Server_Cert: No
      Last_IO_Errno: 0
      Last_IO_Error:
      Last_SQL_Errno: 0
      Last_SQL_Error:
      Replicate_Ignore_Server_Ids: 0
```

ステータスレポートの中で調査すべき主要フィールドは、次のとおりです。

- **Replica_IO_State**: レプリカの現在のステータス。詳細は、[セクション8.14.5「レプリケーション I/O スレッドの状態」](#) および [セクション8.14.6「レプリケーション SQL スレッドの状態」](#) を参照してください。
- **Replica_IO_Running**: ソースバイナリログを読み取るための I/O スレッドが実行されているかどうか。通常、レプリケーションをまだ開始していないか、`STOP REPLICAS | SLAVE` で明示的に停止していないかぎり、これを **Yes** にします。
- **Replica_SQL_Running**: リレーログ内のイベントを実行するための SQL スレッドが実行中かどうか。I/O スレッドと同様、これは通常は **Yes** にすることをお勧めします。
- **Last_IO_Error**、**Last_SQL_Error**: リレーログを処理するときに I/O および SQL スレッドによって登録された最後のエラー。理想的には、これらはエラーがないことを示すブランクであるべきです。
- **Seconds_Behind_Source**: レプリケーション SQL スレッドがソースバイナリログの処理を遅れている秒数。高い数値 (または増加する数値) は、レプリカがソースからのイベントを適時に処理できないことを示します。

Seconds_Behind_Source の値 0 は通常、レプリカがソースに追いついたことを意味するものとして解釈できますが、厳密には **true** でない場合もあります。たとえば、これは、ソースとレプリカ間のネットワーク接続が切断されたが、レプリケーション I/O スレッドがまだこれに気付いていない (`slave_net_timeout` がまだ経過していない) 場合に発生することがあります。

Seconds_Behind_Source の一時的な値が状況を正確に反映しない場合もあります。レプリケーション SQL スレッドが I/O、**Seconds_Behind_Source** でキャッチアップされると 0 が表示されますが、レプリケーション I/O スレッドがまだ新しいイベントをキューに入れている場合、レプリケーション SQL スレッドが新しいイベントの実行を終了するまで、**Seconds_Behind_Source** に大きな値が表示されることがあります。これは、イベントに古いタイムスタンプがある場合に特に発生する可能性があります。このような場合、比較的短い期間に `SHOW REPLICAS | SLAVE STATUS` を複数回実行すると、この値が 0 から比較的大きい値まで繰り返し変更されることがあります。

フィールドのいくつかのペアは、ソースバイナリログからイベントを読み取り、リレーログでそれらを処理する際のレプリカの進行状況に関する情報を提供します:

- (**Master_Log_File**, **Read_Master_Log_Pos**): レプリケーション I/O スレッドがそのログからイベントを読み取る距離を示す、ソースバイナリログ内の座標。
- (**Relay_Master_Log_File**, **Exec_Master_Log_Pos**): レプリケーション SQL スレッドがそのログから受信したイベントを実行した距離を示すソースバイナリログ内の座標。
- (**Relay_Log_File**, **Relay_Log_Pos**): レプリケーション SQL スレッドがリレーログを実行した距離を示すレプリカリレーログ内の座標。これらは前述の座標に対応していますが、ソースバイナリログ座標ではなくレプリカリレーログ座標で表されます。

ソースでは、`SHOW PROCESSLIST` を使用して接続レプリカのステータスを確認し、実行中のプロセスのリストを調べることができます。レプリカ接続では、**Command** フィールドに **Binlog Dump** があります:

```
mysql> SHOW PROCESSLIST \G;
***** 4. row *****
  Id: 10
  User: root
  Host: replica1:58371
  db: NULL
  Command: Binlog Dump
  Time: 777
  State: Has sent all binlog to slave; waiting for binlog to be updated
  Info: NULL
```

これはレプリケーションプロセスを駆動するレプリカであるため、このレポートで使用できる情報はほとんどありません。

`--report-host` オプションで開始され、ソースに接続されているレプリカの場合、ソースの `SHOW REPLICAS | SHOW SLAVE HOSTS` ステートメントにレプリカに関する基本情報が表示されます。出力には、レプリカサーバーの ID、`--report-host` オプションの値、接続ポートおよびソース ID が含まれます:

```
mysql> SHOW REPLICAS;
+-----+-----+-----+-----+
| Server_id | Host   | Port | Rpl_recovery_rank | Source_id |
+-----+-----+-----+-----+

```

```
| 10 | replica1 | 3306 | 0 | 1 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

17.1.7.2 レプリカでのレプリケーションの一時停止

`STOP REPLICHA | SLAVE` および `START REPLICHA | SLAVE` ステートメントを使用して、レプリカのレプリケーションを停止および開始できます。MySQL 8.0.22 から、`STOP SLAVE` および `START SLAVE` は非推奨になり、`STOP REPLICHA` および `START REPLICHA` をかわりに使用できます。

ソースからのバイナリログの処理を停止するには、`STOP REPLICHA | SLAVE` を使用します:

```
mysql> STOP SLAVE;
Or from MySQL 8.0.22:
mysql> STOP REPLICHA;
```

レプリケーションが停止すると、レプリケーション I/O スレッドはソースバイナリログからのイベントの読み取りとリレーログへの書き込みを停止し、SQL スレッドはリレーログからのイベントの読み取りと実行を停止します。スレッドタイプを指定することで、I/O または SQL スレッドを個別に一時停止できます。

```
mysql> STOP SLAVE IO_THREAD;
mysql> STOP SLAVE SQL_THREAD;
Or from MySQL 8.0.22:
mysql> STOP REPLICHA IO_THREAD;
mysql> STOP REPLICHA SQL_THREAD;
```

実行を再開するには、`START REPLICHA | SLAVE` ステートメントを使用します:

```
mysql> START SLAVE;
Or from MySQL 8.0.22:
mysql> START REPLICHA;
```

特定のスレッドを開始するには、スレッドタイプを指定します。

```
mysql> START SLAVE IO_THREAD;
mysql> START SLAVE SQL_THREAD;
Or from MySQL 8.0.22:
mysql> START REPLICHA IO_THREAD;
mysql> START REPLICHA SQL_THREAD;
```

ソースからのイベントを処理することによってのみ更新を実行するレプリカの場合、SQL スレッドのみを停止すると、バックアップまたはその他のタスクを実行する場合に役立ちます。I/O スレッドはソースからのイベントの読み取りを続行しますが、実行されません。これにより、SQL スレッドの再起動時にレプリカが捕捉されやすくなります。

I/O スレッドだけを停止することで、SQL スレッドはリレーログが終了したポイントまでリレーログ内のイベントを実行できます。これは、ソースからすでに受信したイベントを捕捉するために実行を一時停止する場合、レプリカで管理を実行するが、特定のポイントへのすべての更新が処理されていることを確認する場合に役立ちます。この方法を使用して、ソースでの管理を実行しながら、レプリカでのイベント受信を一時停止することもできます。I/O スレッドは停止するけれども SQL スレッドの実行を許可することで、レプリケーションが再開したときに実行される大量のイベントバックログを確実になくすのに役立ちます。

17.1.7.3 トランザクションのスキップ

レプリケートされたトランザクションのイベントの問題が原因でレプリケーションが停止した場合は、レプリカで失敗したトランザクションをスキップしてレプリケーションを再開できます。トランザクションをスキップする前に、レプリケーション I/O スレッドと SQL スレッドが停止していることを確認します。

まず、エラーの原因となったレプリケートイベントを特定する必要があります。エラーの詳細および最後に正常に適用されたトランザクションは、「パフォーマンススキーマ」テーブル `replication_applier_status_by_worker` に記録されます。 `mysqlbinlog` を使用して、エラー発生時にログに記録されたイベントを取得して表示できます。これを行う手順は、[セクション7.5「Point-in-Time \(増分\) リカバリ」](#) を参照してください。または、レプリカまたはソース上の `SHOW BINLOG EVENTS` で `SHOW RELAYLOG EVENTS` を発行できます。

トランザクションをスキップしてレプリカを再起動する前に、次の点を確認します:

- 不明または信頼できないソースからのレプリケーションを停止したトランザクションですか。その場合は、レプリカを再起動しないことを示すセキュリティ上の考慮事項がある場合に、原因を調査します。

- レプリケーションを停止したトランザクションをレプリカに適用する必要がありますか。その場合は、適切な修正を行ってトランザクションを再適用するか、レプリカ上のデータを手動で調整します。
- レプリケーションを停止したトランザクションをソースに適用する必要がありましたか。そうでない場合は、最初に実行されたサーバーでトランザクションを手動で元に戻します。

トランザクションをスキップするには、必要に応じて次のいずれかの方法を選択します:

- GTID が使用中 (`gtid_mode` が ON) の場合は、[GTID のあるトランザクションのスキップ](#) を参照してください。
- GTID が使用されていないか、段階的に導入されている (`gtid_mode` が OFF、OFF_PERMISSIVE または ON_PERMISSIVE) 場合は、[GTID のないトランザクションのスキップ](#) を参照してください。
- `CHANGE REPLICATION SOURCE TO` ステートメントの `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` オプションを使用してレプリケーションチャンネルで GTID 割り当てを有効にした場合は、[GTID のないトランザクションのスキップ](#) を参照してください。レプリケーションチャンネルで `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` を使用することは、チャンネルに GTID ベースのレプリケーションを導入することとは異なり、GTID ベースのレプリケーションではそれらのチャンネルでトランザクションスキップ方法を使用できません。

トランザクションのスキップ後にレプリケーションを再開するには、レプリカがマルチソースレプリカの場合は、`FOR CHANNEL` 句を指定して `START REPLICA | SLAVE` を発行します。

GTID のあるトランザクションのスキップ

GTID が使用されている場合 (`gtid_mode` が ON)、コミットされたトランザクションの GTID は、トランザクションの内容がフィルタで除外されてもレプリカに保持されます。この機能により、GTID 自動配置を使用してソースに再接続したときに、レプリカが以前にフィルタ処理されたトランザクションを取得できなくなります。また、障害が発生したトランザクションのかわりに空のトランザクションをコミットすることで、レプリカ上のトランザクションをスキップするためにも使用できます。

このトランザクションのスキップ方法は、`CHANGE REPLICATION SOURCE TO` ステートメントの `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` オプションを使用してレプリケーションチャンネルで GTID 割り当てを有効にした場合には適していません。

失敗したトランザクションによってワーカースレッドでエラーが生成された場合は、「パフォーマンススキーマ」テーブル `replication_applier_status_by_worker` の `LAST_SEEN_TRANSACTION` フィールドから GTID を直接取得できます。トランザクションの内容を確認するには、レプリカまたはソース上の `SHOW BINLOG EVENTS` で `SHOW RELAYLOG EVENTS` を発行し、その GTID で始まるトランザクションの出力を検索します。

前述の他の適切なアクション (セキュリティ上の考慮事項など) の失敗したトランザクションを評価した場合、スキップするには、失敗したトランザクションと同じ GTID を持つレプリカで空のトランザクションをコミットします。例:

```
SET GTID_NEXT='aaa-bbb-ccc-ddd:N';
BEGIN;
COMMIT;
SET GTID_NEXT='AUTOMATIC';
```

レプリカ上にこの空のトランザクションが存在するということは、レプリケーションを再開するために `START REPLICA | SLAVE` ステートメントを発行すると、その GTID を持つトランザクションがすでに適用されていることがわかっているため、レプリカは自動スキップ機能を使用して失敗したトランザクションを無視することを意味します。レプリカがマルチソースレプリカの場合、空のトランザクションのコミット時にチャンネル名を指定する必要はありませんが、`START REPLICA | SLAVE` の発行時にチャンネル名を指定する必要があります。

バイナリロギングがこのレプリカで使用されている場合、レプリカが将来ソースまたはプライマリになると、空のトランザクションがレプリケーションストリームに入ります。この可能性を回避する必要がある場合は、次の例のようにレプリカバイナリログをフラッシュおよびパージすることを検討してください:

```
FLUSH LOGS;
PURGE BINARY LOGS TO 'binlog.000146';
```

空のトランザクションの GTID は永続化されますが、トランザクション自体はバイナリログファイルをパージすることによって削除されます。

GTID のないトランザクションのスキップ

GTID が使用されていないか、段階的に導入されている (`gtid_mode` が `OFF`、`OFF_PERMISSIVE` または `ON_PERMISSIVE`) 場合に失敗したトランザクションをスキップするには、`SET GLOBAL sql_slave_skip_counter` ステートメントを発行して、指定した数のイベントをスキップできます。または、`CHANGE MASTER TO` ステートメントを発行してソースバイナリログの位置を前方に移動することによって、イベントを過ぎてスキップすることもできます。

これらの方法は、`CHANGE REPLICATION SOURCE TO` ステートメントの `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` オプションを使用してレプリケーションチャンネルで GTID 割当てを有効にした場合にも適しています。

これらの方法を使用する場合は、前述の GTID ベースの方法の場合と同様に、完全なトランザクションをスキップする必要はないことを理解することが重要です。GTID ベース以外のこれらの方法では、そのようなトランザクションは認識されませんが、かわりにイベントを操作します。バイナリログはイベントグループと呼ばれる一連のグループとして編成され、各イベントグループは一連のイベントで構成されます。

- トランザクションテーブルの場合、イベントグループはトランザクションに対応しています。
- 非トランザクションテーブルの場合、イベントグループは 1 つの SQL ステートメントに対応しています。

1 つのトランザクションに、トランザクションテーブルと非トランザクションテーブルの両方の変更を含めることができます。

`SET GLOBAL sql_slave_skip_counter` ステートメントを使用してイベントをスキップし、結果の位置がイベントグループの途中にある場合、レプリカはグループの最後に達するまでイベントをスキップし続けます。そのあと、次のイベントグループから実行が開始されます。`CHANGE MASTER TO` ステートメントにはこの機能がないため、イベントグループの開始時にレプリケーションを再開する正しい場所を特定するように注意する必要があります。ただし、`CHANGE MASTER TO` を使用すると、`SET GLOBAL sql_slave_skip_counter` と同様にスキップする必要があるイベントをカウントする必要がなく、かわりに再起動する場所を指定できます。

`SET GLOBAL sql_slave_skip_counter` でのトランザクションのスキップ

前述の他の適切なアクション (セキュリティ上の考慮事項など) の失敗したトランザクションを評価したら、スキップする必要があるイベントの数をカウントします。あるイベントは通常、バイナリログ内の 1 つの SQL ステートメントに対応しますが、`AUTO_INCREMENT` または `LAST_INSERT_ID()` を使用するステートメントはバイナリログ内の 2 つのイベントとしてカウントされることに注意してください。バイナリログのトランザクション圧縮が使用されている場合、圧縮されたトランザクションペイロード (`Transaction_payload_event`) は単一のカウンタ値としてカウントされるため、その内部のすべてのイベントは単位としてスキップされます。

完了したトランザクションをスキップする場合は、イベントをトランザクションの最後までカウントすることも、関連するイベントグループをスキップすることもできます。`SET GLOBAL sql_slave_skip_counter` では、レプリカはイベントグループの最後までスキップし続けることに注意してください。スキップがスキップされないように、スキップして次のイベントグループまたはトランザクションに移動しないでください。

次のように `SET` ステートメントを発行します。ここで、`N` はスキップするソースからのイベント数です：

```
SET GLOBAL sql_slave_skip_counter = N
```

このステートメントは、`gtid_mode=ON` が設定されている場合、またはレプリケーション I/O および SQL スレッドが実行されている場合は発行できません。

`SET GLOBAL sql_slave_skip_counter` ステートメントはすぐには影響しません。次の `SET` ステートメントの後に `START REPLICATION | SLAVE` ステートメントを発行すると、システム変数 `sql_slave_skip_counter` の新しい値が適用され、イベントはスキップされます。その `START REPLICATION | SLAVE` ステートメントでは、システム変数の値も自動的に 0 に戻されます。レプリカがマルチソースレプリカの場合は、その `START REPLICATION | SLAVE` ステートメントを発行するときに `FOR CHANNEL` 句が必要です。正しいチャンネルを指定していることを確認してください。そうしないと、間違ったチャンネルでイベントがスキップされます。

`CHANGE MASTER TO` でのトランザクションのスキップ

前述の他の適切なアクション (セキュリティ上の考慮事項など) の失敗したトランザクションを評価したら、レプリケーションを再開するための適切な位置を表すソースバイナリログ内の座標 (ファイルと位置) を特定します。これは、問題の原因となったイベントの後のイベントグループの開始、または次のトランザクションの開始です。レプリ

ケーション I/O スレッドは、次にスレッドが起動したときに、これらの座標でソースからの読取りを開始し、失敗したイベントをスキップします。このステートメントではイベントグループが考慮されないため、ポジションが正確に識別されていることを確認してください。

次のように `CHANGE MASTER TO` ステートメントを発行します。ここで、`source_log_name` は再起動位置を含むバイナリログファイル、`source_log_pos` はバイナリログファイルに記述されている再起動位置を表す番号です：

```
CHANGE MASTER TO MASTER_LOG_FILE='source_log_name', MASTER_LOG_POS=source_log_pos;
```

レプリカがマルチソースレプリカの場合は、`FOR CHANNEL` 句を使用して `CHANGE MASTER TO` ステートメントの適切なチャンネルに名前を付ける必要があります。

このステートメントは、`MASTER_AUTO_POSITION=1` が設定されている場合、またはレプリケーション I/O および SQL スレッドが実行されている場合は発行できません。`MASTER_AUTO_POSITION=1` が通常設定されているときにトランザクションをスキップする方法を使用する必要がある場合は、ステートメントの発行中に設定を `MASTER_AUTO_POSITION=1` に変更してから、変更しなおすことができます。例：

```
CHANGE MASTER TO MASTER_AUTO_POSITION=0, MASTER_LOG_FILE='binlog.000145', MASTER_LOG_POS=235;  
CHANGE MASTER TO MASTER_AUTO_POSITION=1;
```

17.2 レプリケーションの実装

レプリケーションは、バイナリログ内のデータベースに対するすべての変更(更新、削除など)を追跡するソースサーバーに基づいています。バイナリログは、サーバーが起動した瞬間からデータベースの構造または内容(データ)を変更するあらゆるイベントが書き込まれた記録として機能します。`SELECT` ステートメントは通常、データベースの構造および内容を変更しないため記録されません。

ソースに接続する各レプリカは、バイナリログのコピーを要求します。つまり、ソースがレプリカにデータをプッシュするのではなく、ソースからデータをプルします。レプリカは、受信したバイナリログからもイベントを実行します。これは、元の変更をソースで行ったときと同じように繰り返す効果があります。テーブルが作成されるか、その構造が変更され、元のソースで行われた変更に従ってデータが挿入、削除および更新されます。

各レプリカは独立しているため、ソースバイナリログからの変更のリプレイは、ソースに接続されている各レプリカで独立して行われます。また、各レプリカはソースからのリクエストによってのみバイナリログのコピーを受信するため、レプリカは独自のペースでデータベースのコピーを読み取って更新でき、ソース側またはレプリカ側の最新のデータベースステータスに更新する機能に影響を与えることなく、レプリケーションプロセスを任意に開始および停止できます。

レプリケーション実装の仕様に関する詳細は、[セクション17.2.3「レプリケーションスレッド」](#)を参照してください。

ソースサーバーおよびレプリカは、レプリケーションプロセスに関するステータスを定期的にレポートして、それらを監視できるようにします。すべてのレプリケーション関連ステータスの説明については、[セクション8.14「サーバーレッド\(プロセス\)情報の確認」](#)を参照してください。

ソースバイナリログは、レプリカが処理される前に、レプリカ上のローカルリレーログに書き込まれます。また、レプリカは、現在の位置に関する情報をソースバイナリログおよびローカルリレーログとともに記録します。[セクション17.2.4「リレーログおよびレプリケーションメタデータリポジトリ」](#)を参照してください。

データベースの変更は、イベント評価を制御する様々な構成オプションおよび変数に従って適用される一連のルールに従ってレプリカでフィルタ処理されます。これらのルールがどのように適用されるかについての詳細は、[セクション17.2.5「サーバーがレプリケーションフィルタリングルールをどのように評価するか」](#)を参照してください。

17.2.1 レプリケーション形式

レプリケーションは、バイナリログに書き込まれたイベントがソースから読み取られてからレプリカで処理されるため機能します。イベントは、イベントタイプに従ってさまざまな形式でバイナリログに記録されます。使用されるさまざまなレプリケーション形式は、イベントがソースバイナリログに記録されたときに使用されたバイナリロギング形式に対応します。バイナリロギング形式とレプリケーションで使用される用語との関係は次のとおりです。

- ステートメントベースのバイナリロギングを使用する場合、ソースは SQL ステートメントをバイナリログに書き込みます。レプリカへのソースのレプリケーションは、レプリカで SQL ステートメントを実行することで機能します。これは、MySQL ステートメントベースのバイナリロギング形式に対応するステートメントベースレプリケーション(SBRと省略できます)と呼ばれます。

- 行ベースのロギングを使用する場合、ソースは個々のテーブル行の変更方法を示す events をバイナリログに書き込みます。レプリカへのソースのレプリケーションは、テーブルの行に対する変更を表すイベントをレプリカにコピーすることで機能します。これは行ベースのレプリケーションと呼ばれます (RBR と省略できます)。

デフォルトの方法は行ベースのロギングです。

- 変更ログが記録されるときにステートメントベースと行ベースのどちらが適しているかによって、これらのロギングの組み合わせを使用するように MySQL を構成することもできます。これは混合形式のロギングと呼ばれます。混合形式のロギングを使用する場合、デフォルトでステートメントベースのログが使用されます。ステートメントに応じて、また使用されるストレージエンジンに応じて、ログは特定のケースで行ベースに自動的に切りかえられます。混合形式を使用したレプリケーションは、混合ベースのレプリケーションまたは混合形式のレプリケーションと呼ばれます。詳細については、[セクション5.4.4.3「混合形式のバイナリロギング形式」](#)を参照してください。

NDB Cluster. MySQL NDB Cluster 8.0 のデフォルトのバイナリロギング形式は `MIXED` です。NDB Cluster レプリケーションは常に行ベースレプリケーションを使用し、NDB ストレージエンジンはステートメントベースレプリケーションと互換性がないことに注意してください。詳細については、[セクション23.6.2「NDB Cluster レプリケーションの一般的な要件」](#)を参照してください。

混合形式を使用する場合、バイナリロギング形式は、使用されているストレージエンジンと実行されているステートメントによってある程度決定されます。混合形式のロギング、および異なるロギング形式のサポートを管理するルールの詳細は、[セクション5.4.4.3「混合形式のバイナリロギング形式」](#)を参照してください。

実行中 MySQL サーバーのロギング形式は、`binlog_format` サーバースystem変数を設定することで制御されます。この変数はセッションまたはグローバルスコープで設定できます。新しい設定が有効になるタイミングと方法を制御するルールは、他の MySQL サーバースystem変数の場合と同じです。現在のセッションの変数の設定は、そのセッションが終了するまでのみ継続され、変更は他のセッションには表示されません。変数をグローバルに設定すると、変更後に接続するクライアントに対して有効になりますが、変数設定が変更されたセッションを含む現在のクライアントセッションに対しては有効になりません。グローバルsystem変数設定を永続的にしてサーバーの再起動後も適用されるようにするには、オプションファイルで設定する必要があります。詳細は、[セクション13.7.6.1「変数代入のSET構文」](#)を参照してください。

実行時にバイナリロギング形式を変更できない、つまりそのようにするとレプリケーションが失敗する状況があります。[セクション5.4.4.2「バイナリログ形式の設定」](#)を参照してください。

グローバル `binlog_format` 値を変更するには、グローバルsystem変数を設定するのに十分な権限が必要です。セッションの `binlog_format` 値を変更するには、制限付きセッションsystem変数を設定するのに十分な権限が必要です。[セクション5.1.9.1「system変数権限」](#)を参照してください。

ステートメントベースと行ベースのレプリケーション形式には、異なる問題と制限があります。関連するメリットとデメリットの比較は、[セクション17.2.1.1「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」](#)を参照してください。

ステートメントベースのレプリケーションでは、ストアドルーチンまたはトリガーの複製で問題が発生する場合があります。代わりに行ベースのレプリケーションを使用することで、これらの問題を回避できます。詳細については、[セクション25.7「ストアドプログラムバイナリロギング」](#)を参照してください。

17.2.1.1 ステートメントベースおよび行ベースレプリケーションのメリットとデメリット

それぞれのバイナリロギングの形式にメリットとデメリットがあります。ほとんどのユーザーにとって、データの完全性とパフォーマンスの最善の組み合わせが得られるのは、混合レプリケーション形式であるはずですが、ただし、特定のタスクを実行するときにはステートメントベースまたは行ベースレプリケーション形式固有の機能を利用する場合、関連するメリットとデメリットのサマリーを記述したこのセクションの情報を使用して、どちらがニーズに最適であるかを定めることができます。

- [ステートメントベースレプリケーションのメリット](#)
- [ステートメントベースレプリケーションのデメリット](#)
- [行ベースレプリケーションのメリット](#)
- [行ベースレプリケーションのデメリット](#)

ステートメントベースレプリケーションのメリット

- 実績のあるテクノロジー。
- ログファイルに書き込まれるデータが少ないです。更新または削除が多くに行に影響を与える場合、これによってログファイルに必要なストレージ容量がかなり少なくなります。つまり、バックアップの取得とリストアをより短時間で達成できます。
- ログファイルには変更があったすべてのステートメントが含まれるため、データベースの監査に使用できます。

ステートメントベースレプリケーションのデメリット

- SBR にとって安全でないステートメント。
データを変更するすべてのステートメント ([INSERT](#) [DELETE](#)、[UPDATE](#)、[REPLACE](#) ステートメントなど) を、ステートメントベースレプリケーションを使用して複製できるわけではありません。ステートメントベースレプリケーションを使用する場合、非決定的動作は複製が困難です。このようなデータ変更言語 (DML) ステートメントの例を次に示します:
 - 非決定的な UDF またはストアードプログラムに依存するステートメント。そのような UDF またはストアードプログラムによって返される値は、それに提供されるパラメータ以外の要因に依存するため。(ただし、行ベースのレプリケーションでは、UDF またはストアードプログラムによって返される値が単にレプリケートされるため、テーブルの行およびデータに対する影響はソースとレプリカの両方で同じです。) 詳細は、[セクション 17.5.1.16 「呼び出される機能のレプリケーション」](#) を参照してください。
 - [ORDER BY](#) なしで [LIMIT](#) 句を使用する [DELETE](#) および [UPDATE](#) ステートメントは非決定的です。 [セクション 17.5.1.18 「レプリケーションと LIMIT」](#) を参照してください。
 - [NOWAIT](#) または [SKIP LOCKED](#) オプションを使用する読取りステートメント ([SELECT ... FOR UPDATE](#) および [SELECT ... FOR SHARE](#)) のロック。 [NOWAIT](#) および [SKIP LOCKED](#) による読取り同時実行性のロックを参照してください。
- 決定論的 UDF をレプリカに適用する必要があります。
- 次のいずれかの関数を使用するステートメントは、ステートメントベースレプリケーションでは適切に複製できません。
 - [LOAD_FILE\(\)](#)
 - [UUID\(\)](#)、[UUID_SHORT\(\)](#)
 - [USER\(\)](#)
 - [FOUND_ROWS\(\)](#)
 - [SYSDATE\(\)](#) (ソースとレプリカの両方が `--sysdate-is-now` オプションで起動されていない場合)
 - [GET_LOCK\(\)](#)
 - [IS_FREE_LOCK\(\)](#)
 - [IS_USED_LOCK\(\)](#)
 - [MASTER_POS_WAIT\(\)](#)
 - [RAND\(\)](#)
 - [RELEASE_LOCK\(\)](#)
 - [SLEEP\(\)](#)
 - [VERSION\(\)](#)

ただし、[NOW\(\)](#) などを含めてほかのすべての関数はステートメントベースレプリケーションで正しく複製されます。

詳細については、[セクション 17.5.1.14 「レプリケーションとシステム関数」](#) を参照してください。

ステートメントベースレプリケーションで正しく複製できないステートメントは、ここに示すもののような警告でログが記録されます。

```
[Warning] Statement is not safe to log in statement format.
```

このような場合、類似の警告がクライアントにも発行されます。クライアントは `SHOW WARNINGS` を使用してそれを表示できます。

- `INSERT ... SELECT` は、行ベースレプリケーションよりも多くの行レベルロックが必要です。
- `WHERE` 句でインデックスが使用されていないためにテーブルスキャンを必要とする `UPDATE` ステートメントは、行ベースレプリケーションの場合より多くの行をロックする必要があります。
- InnoDB の場合: `AUTO_INCREMENT` を使用する `INSERT` ステートメントは、競合しないほかの `INSERT` ステートメントをブロックします。
- 複雑なステートメントの場合、行が更新または挿入される前に、ステートメントを評価してレプリカで実行する必要があります。行ベースのレプリケーションでは、レプリカは影響を受ける行のみを変更する必要があり、完全なステートメントは実行しません。
- 特に複雑なステートメントの実行時にレプリカの評価でエラーが発生した場合、ステートメントベースのレプリケーションによって、影響を受ける行のエラーのマージンが徐々に増加する可能性があります。 [セクション 17.5.1.29 「レプリケーション中のレプリカエラー」](#) を参照してください。
- ストアドファンクションは、呼び出し元のステートメントと同じ `NOW()` 値で実行します。ただし、これはストアドプロシージャには当てはまりません。
- 決定論的 UDF をレプリカに適用する必要があります。
- テーブル定義は、ソースとレプリカで (ほぼ) 同一である必要があります。詳細については、 [セクション 17.5.1.9 「ソースとレプリカで異なるテーブル定義を使用したレプリケーション」](#) を参照してください。
- MySQL 8.0.22 の時点では、(結合リストまたはサブクエリーを介して) MySQL 付与テーブルからデータを読み取るが、変更しない DML 操作は、MySQL 付与テーブルに対する非ロック読取りとして実行されるため、ステートメントベースのレプリケーションでは安全ではありません。詳細は、 [テーブル同時実行性の付与](#) を参照してください。

行ベースレプリケーションのメリット

- すべての変更を複製できます。これはもっとも安全な形式のレプリケーションです。

注記

`GRANT`、`REVOKE` およびトリガー、ストアドルーチン (ストアドプロシージャを含む) およびビューの操作など、`mysql` システムスキーマ内の情報を更新するステートメントはすべて、ステートメントベースレプリケーションを使用してレプリカにレプリケートされます。

`CREATE TABLE ... SELECT` などのステートメントの場合、`CREATE` ステートメントはテーブル定義から生成されてステートメントベース形式を使用して複製される一方、行挿入は行ベース形式を使用して複製されます。

- 次のタイプのステートメントでは、ソースで必要な行ロックが少なくなるため、同時実行性が高くなります:
 - `INSERT ... SELECT`
 - `AUTO_INCREMENT` 付きの `INSERT` ステートメント
 - キーを使用しないまたは検査された行のほとんどを変更しない `WHERE` 句付きの `UPDATE` または `DELETE` ステートメント。
- `INSERT`、`UPDATE` または `DELETE` ステートメントのレプリカで必要な行ロックが少なくなります。

行ベースレプリケーションのデメリット

- RBR では、ログに書き込む必要があるデータが増える可能性があります。ステートメントベースレプリケーションでは、DML ステートメント (UPDATE、DELETE ステートメントなど) を複製するためにステートメントだけをバイナリログに書き込みます。一方、行ベースレプリケーションでは変更されたすべての行をバイナリログに書き込みます。ステートメントが多くの変更に伴って、行ベースレプリケーションは非常に多くのデータをバイナリログに書き込む可能性があります。このことはロールバックされるステートメントにも当てはまります。これは、バックアップの作成およびリストアにさらに時間がかかる可能性があることも意味します。また、データを書き込むためにバイナリログがロックされる時間が長くなるため、並列性の問題が発生する場合があります。 `binlog_row_image=minimal` を使用すると、デメリットを大幅に削減できます。
- 大きな BLOB 値を生成する決定的 UDF の場合は、ステートメントベースレプリケーションより行ベースレプリケーションの方が複製に時間がかかります。これは、データを生成するステートメントではなく、BLOB カラム値がログに書き込まれるためです。
- レプリカでは、ソースから受信して実行されたステートメントは表示されません。ただし、オプション `--base64-output=DECODE-ROWS` および `--verbose` を付けて `mysqlbinlog` を使用すると、何のデータが変更されたかがわかります。
または、`binlog_rows_query_log_events` 変数を使用します。これを有効にすると、`-vv` オプションが使用されたときに、`Rows_query` イベントとそのステートメントが `mysqlbinlog` 出力に追加されます。
- MyISAM ストレージエンジンを使用するテーブルの場合、INSERT ステートメントをバイナリログに行ベースのイベントとして適用するときは、ステートメントとして適用するときよりも強力なロックが INSERT ステートメントのレプリカに必要です。これは、MyISAM テーブルでの同時挿入が、行ベースレプリケーションを使用するときをサポートされないことを意味します。

17.2.1.2 行ベースロギングおよびレプリケーションの使用

MySQL はステートメントベースロギング (SBL)、行ベースロギング (RBL)、または混合形式ロギングを使用します。使用されるバイナリログのタイプは、ロギングのサイズと効率に影響します。したがって、行ベースレプリケーション (RBR) とステートメントベースレプリケーション (SBR) のどちらを選択するかは、アプリケーションと環境によって異なります。このセクションでは、行ベースのフォーマットログを使用する際の既知の問題と、それをレプリケーションで使用するためのいくつかのベストプラクティスについて説明します。

詳細については、[セクション 17.2.1 「レプリケーション形式」](#) および [セクション 17.2.1.1 「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」](#) を参照してください。

NDB Cluster レプリケーションに固有の問題 (行ベースレプリケーションに依存) については、[セクション 23.6.3 「NDB Cluster レプリケーションの既知の問題」](#) を参照してください。

- 一時テーブルの行ベースロギング。 [セクション 17.5.1.31 「レプリケーションと一時テーブル」](#) で説明されているように、行ベース形式または (MySQL 8.0.4) 混合形式を使用する場合、一時テーブルはレプリケートされません。詳細については、[セクション 17.2.1.1 「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」](#) を参照してください。

行ベースまたは混合形式を使用する場合、一時テーブルはレプリケートされません。これは、必要がないためです。また、一時テーブルはそれらを作成したスレッドからのみ読み取れるため、ステートメントベース形式を使用する場合でも、それらを複製することから得られるメリットはまずありません。

一時テーブルが作成されている場合でも、実行時にステートメントベースから行ベースのバイナリロギング形式に切り替えることができます。ただし、MySQL 8.0 では、前のモードでバイナリログから `CREATE TEMPORARY TABLE` ステートメントが省略されているため、バイナリロギングの行ベースまたは混合形式からステートメントベース形式に切り替えることはできません。

MySQL サーバーは、各一時テーブルが作成されたときに有効だったロギングモードを追跡します。特定のクライアントセッションが終了すると、サーバーは、まだ存在し、ステートメントベースのバイナリロギングが使用されていたときに作成された一時テーブルごとに `DROP TEMPORARY TABLE IF EXISTS` ステートメントをログに記録します。テーブルの作成時に行ベースまたは混合形式のバイナリロギングが使用されていた場合、`DROP TEMPORARY TABLE IF EXISTS` ステートメントはログに記録されません。MySQL 8.0.4 および 5.7.25 より前のリリースでは、`DROP TEMPORARY TABLE IF EXISTS` ステートメントは有効なロギングモードに関係なくログに記録されていました。

一時テーブルに関連する非トランザクション DML ステートメントは、ステートメントの影響を受ける非トランザクションテーブルが一時テーブルであるかぎり、`binlog_format=ROW` の使用時に許可されます (Bug #14272672)。

- 非トランザクションテーブルの RBL と同期。多くの行が影響を受ける場合、変更のセットは複数のイベントに分割されます。ステートメントがコミットすると、これらのイベントのすべてがバイナリログに書き込まれます。レプリカで実行すると、関連するすべてのテーブルでテーブルロックが取得され、行がバッチモードで適用されます。テーブルのレプリカコピーに使用されるエンジンによっては、これが有効な場合と無効な場合があります。
- 待機時間およびバイナリログサイズ。RBL は各行の変更をバイナリログに書き込むため、そのサイズは急激に増える場合があります。これにより、ソースと一致するレプリカの変更に必要な時間が大幅に長くなる可能性があります。アプリケーションでこのような遅延が発生する可能性を意識してください。
- バイナリログの読み取り。mysqlbinlog は、BINLOG ステートメントを使用してバイナリログ内の行ベースイベントを表示します(セクション13.7.8.1「BINLOG ステートメント」を参照してください)。このステートメントは base 64 でエンコードされた文字列(その意味は明白ではありません)としてイベントを表示します。--base64-output=DECODE-ROWS および --verbose オプションを付けて呼び出されたときは、mysqlbinlog はバイナリログの内容を人間が読める形式にします。バイナリログイベントが行ベース形式で書き込まれ、それらを読み取ったりレプリケーションまたはデータベース障害からリカバリしたりしたい場合は、このコマンドでバイナリログの内容を読み取ることができます。詳細については、セクション4.6.8.2「mysqlbinlog 行イベントの表示」を参照してください。
- バイナリログ実行エラーおよびレプリカ実行モード。slave_exec_mode=IDEMPOTENT の使用は一般に、IDEMPOTENT がデフォルト値である MySQL NDB Cluster レプリケーションでのみ役立ちます。(セクション23.6.10「NDB Cluster レプリケーション: 双方向および循環レプリケーション」を参照してください)。slave_exec_mode が IDEMPOTENT の場合、元の行が見つからないため RBL からの変更の適用に失敗しても、エラーはトリガーされず、レプリケーションが失敗します。つまり、ソースとレプリカが同期されなくなるように、更新がレプリカに適用されない可能性があります。slave_exec_mode が IDEMPOTENT の場合、RBR での待機時間の問題および非トランザクションテーブルの使用によって、ソースとレプリカがさらに相違する可能性があります。slave_exec_mode の詳細については、セクション5.1.8「サーバーシステム変数」を参照してください。

その他のシナリオでは、通常、slave_exec_mode を STRICT に設定するだけで十分です。これは、NDB 以外のストレージエンジンのデフォルト値です。
- サーバー ID に基づくフィルタリングはサポートされない。サーバー ID に基づいてフィルタ処理するには、CHANGE REPLICATION SOURCE TO ステートメント(MySQL 8.0.23 の場合)または CHANGE MASTER TO ステートメント(MySQL 8.0.23 の場合)の IGNORE_SERVER_IDS オプションを使用します。このオプションは、ステートメントベースおよび行ベースのロギング形式で機能しますが、GTID_MODE=ON が設定されている場合に使用することは非推奨です。一部のレプリカに対する変更を除外する別の方法は、UPDATE および DELETE ステートメントでリレーション@@server_id <> id_value 句を含む WHERE 句を使用することです。たとえば、WHERE @@server_id <> 1。ただし、これは行ベースロギングでは正しく動作しません。ステートメントフィルタリングに server_id システム変数を使用するには、ステートメントベースロギングを使用します。
- RBL、非トランザクションテーブルおよび停止したレプリカ。行ベースのロギングを使用している場合、レプリカスレッドが非トランザクションテーブルを更新している間にレプリカサーバーが停止すると、レプリカデータベースが一貫性のない状態になる可能性があります。このため、行ベース形式を使用して複製されたすべてのテーブルに、InnoDB などのトランザクションストレージエンジンを使用することをお勧めします。レプリカ MySQL サーバーを停止する前に STOP REPLICATION SLAVE または STOP REPLICATION SLAVE SQL_THREAD を使用すると、問題の発生を防ぐことができ、使用するロギング形式やストレージエンジンに関係なく常に推奨されます。

17.2.1.3 バイナリロギングでの安全および安全でないステートメントの判断

MySQL レプリケーションでのステートメントの「安全性」とは、ステートメントベースの形式を使用してステートメントとその効果を正しくレプリケートできるかどうかを指します。これがステートメントに当てはまる場合、ステートメントは安全と言い、そうでない場合は安全でないと言います。

一般的に、ステートメントが決定的である場合は安全であり、そうでない場合は安全ではありません。ただし、特定の非決定的関数は「安全でない」と見なされません(このセクションの後半の安全でないで見なされない非決定的関数を参照してください)。また、浮動小数点数学関数(ハードウェア依存)からの結果を使用するステートメントは、常に安全でないで見なされます(セクション17.5.1.12「レプリケーションと浮動小数点値」を参照してください)。

安全および安全でないステートメントの処理。ステートメントは、ステートメントが安全と見なされるかどうかに応じて、およびバイナリロギング形式(すなわち、binlog_format の現在の値)に基づいて異なる方法で処理されます。

- 行ベースロギングを使用する場合、安全および安全でないステートメントの扱いに違いはありません。

- 混合形式ロギングを使用する場合、安全でないとフラグされたステートメントは行ベース形式を使用してログが記録され、安全と見なされたステートメントはステートメントベース形式を使用してログが記録されます。
- ステートメントベースロギングを使用する場合、安全でないとフラグされたステートメントはこの結果に警告を生成します。安全なステートメントは通常どおりにログが記録されます。

安全でないとフラグされた各ステートメントは警告を生成します。このようなステートメントがソースで多数実行された場合、エラーログファイルが過剰に大きくなる可能性があります。これを防ぐために、MySQL には警告抑制メカニズムがあります。50 秒間に 50 回を超える最新の `ER_BINLOG_UNSAFE_STATEMENT` 警告が 50 回生成されると、警告抑制が有効になります。有効になっているときは、これによってこのような警告がエラーログに書き込まれることはありません。代わりに、このタイプの警告が 50 個生成されるたびに、「最後の警告が N 回、最近の S 秒間に繰り返されました」との注記がエラーログに書き込まれます。50 個の最近のこのような警告が 50 秒以内に発行されるかぎり、これが続きます。頻度がこのしきい値を下回ると、再度通常どおりに警告ログが記録されます。警告抑止は、ステートメントベースロギングでステートメントの安全がどのように判断されるか、および警告がクライアントにどのように送信されるかに影響しません。MySQL クライアントは引き続きこのようなステートメントごとに 1 つの警告を受け取ります。

詳細については、[セクション 17.2.1 「レプリケーション形式」](#) を参照してください。

安全でないと見なされるステートメント。
次の特徴を持つステートメントは安全でないと見なされます。

- レプリカで異なる値を返す可能性のあるシステム関数を含むステートメント。これらの関数には、`FOUND_ROWS()`、`GET_LOCK()`、`IS_FREE_LOCK()`、`IS_USED_LOCK()`、`LOAD_FILE()`、`MASTER_POS_WAIT()`、`RAND()`、`RELEASE_LOCK()`、`ROW_COUNT()`、`SESSION_USER()`、`SLEEP()`、`SYSDATE()`、`SYSTEM_USER()`、`USER()`、`UUID()` および `UUID_SHORT()` が含まれます。

安全でないと見なされない非決定的関数。これらの関数は決定的ではありませんが、ロギングおよびレプリケーション目的の場合は安全として処理されます：
`CONNECTION_ID()`、`CURDATE()`、`CURRENT_DATE()`、`CURRENT_TIME()`、`CURRENT_TIMESTAMP()`、`CURTIME()`、`LAST_INSERT_ID()` および `UTC_TIMESTAMP()`。

詳細については、[セクション 17.5.1.14 「レプリケーションとシステム関数」](#) を参照してください。

- システム変数への参照。ほとんどのシステム変数は、ステートメントベース形式で正しく複製されません。[セクション 17.5.1.39 「レプリケーションと変数」](#) を参照してください。例外については、[セクション 5.4.4.3 「混合形式のバイナリロギング形式」](#) を参照してください。
- UDF。UDF が何をするかは制御できないため、それが安全でないステートメントを実行していると推定する必要があります。
- 全文プラグイン。このプラグインの動作は MySQL サーバーによって異なる場合があるため、それに基づくステートメントは結果が異なる場合があります。このため、フルテキストプラグインに依存するすべてのステートメントは、MySQL で安全でないものとして扱われます。
- トリガーまたはストアードプログラムは `AUTO_INCREMENT` カラムを持つテーブルを更新する。行の更新順序がソースとレプリカで異なる可能性があるため、これは安全ではありません。

また、複合主キーを持つテーブルに、この複合キーの先頭カラムでない `AUTO_INCREMENT` カラムが含まれるテーブルに `INSERT` することは、安全ではありません。

詳細については、[セクション 17.5.1.1 「レプリケーションと AUTO_INCREMENT」](#) を参照してください。

- 複数の主キーまたは一意キーを持つテーブルでの `INSERT ... ON DUPLICATE KEY UPDATE` ステートメント。複数の主キーまたは一意キーを持つテーブルに対して実行されるときこのステートメントは、安全でないと見なされます (ストレージエンジンがキーをチェックする順番に影響されやすいですが、これは決定的でなく、さらに MySQL Server が更新する行の選択がこれに依存するためです)。

複数の一意キーまたは主キーを持つテーブルに対する `INSERT ... ON DUPLICATE KEY UPDATE` ステートメントは、ステートメントベースのレプリケーションに対して安全でないとマークされます。(Bug #11765650、Bug #58637)

- `LIMIT` を使用する更新。行の取得順序が指定されていないため、安全でないと見なされます。[セクション 17.5.1.18 「レプリケーションと LIMIT」](#) を参照してください。

- ログテーブルへのアクセスまたは参照。 システムログテーブルの内容は、ソースとレプリカで異なる場合があります。
- トランザクション操作後の非トランザクション操作。 トランザクション内で、トランザクション読み取りまたは書き込み後に非トランザクション読み取りまたは書き込みを実行することを許可することは、安全でないと見なされます。
詳細については、[セクション17.5.1.35「レプリケーションとトランザクション」](#)を参照してください。
- セルフロギングテーブルへのアクセスまたは参照。 セルフロギングテーブルへのすべての読み取りと書き込みは、安全でないと見なされます。 トランザクション内で、セルフロギングテーブルへの読み取りまたは書き込みに続くステートメントも、安全でないと見なされます。
- LOAD DATA ステートメント。 `LOAD DATA` は安全でないとみなされ、`binlog_format=MIXED` の場合、ステートメントは行ベースの形式で記録されます。 他の安全でないステートメントとは異なり、`binlog_format=STATEMENT LOAD DATA` で警告が生成されない場合。
- XA トランザクション。 ソースでパラレルにコミットされた2つのXA トランザクションがレプリカで逆の順序で準備されている場合、安全に解決できないステートメントベースのレプリケーションでロック依存関係が発生する可能性があり、レプリケーションがレプリカのデッドロックで失敗する可能性があります。`binlog_format=STATEMENT` が設定されている場合、XA トランザクション内のDML ステートメントに安全でないというフラグが付けられ、警告が生成されます。`binlog_format=MIXED` または `binlog_format=ROW` が設定されている場合、XA トランザクション内のDML ステートメントは行ベースのレプリケーションを使用して記録され、潜在的な問題は存在しません。
- 非決定的関数を参照する `DEFAULT` 句。 式のデフォルト値が非決定的関数を参照している場合、式を評価するステートメントはステートメントベースレプリケーションでは安全ではありません。 これには、`INSERT`、`UPDATE`、`ALTER TABLE` などのステートメントが含まれます。 他のほとんどの安全でないステートメントとは異なり、このカテゴリのステートメントは行ベースの形式で安全にレプリケートできません。`binlog_format` が `STATEMENT` に設定されている場合、ステートメントはログに記録されて実行されますが、エラーログに警告メッセージが書き込まれます。`binlog_format` が `MIXED` または `ROW` に設定されている場合、ステートメントは実行されず、エラーログにエラーメッセージが書き込まれます。 明示的なデフォルトの処理の詳細は、[MySQL 8.0.13 での明示的なデフォルト処理](#)を参照してください。

追加情報については [セクション17.5.1「レプリケーションの機能と問題」](#)を参照してください。

17.2.2 レプリケーションチャンネル

MySQL マルチソースレプリケーションでは、レプリカはソースサーバーごとに複数のレプリケーションチャンネルを開きます。 レプリケーションチャンネルは、ソースからレプリカに流れるトランザクションのパスを表します。 各レプリケーションチャンネルには、独自の受信者 (I/O) スレッド、1つ以上の適用者 (SQL) スレッドおよびリレーログがあります。 ソースからのトランザクションがチャンネルレシーバスレッドによって受信されると、トランザクションはチャンネルリレーログファイルに追加され、チャンネルアプライアスレッドに渡されます。 これにより、各チャンネルは個別に機能できます。

このセクションでは、レプリケーショントポロジでチャンネルを使用する方法と、チャンネルが単一ソースレプリケーションに与える影響について説明します。 マルチソースレプリケーションのソースおよびレプリカの構成、マルチソースレプリカの起動、停止およびリセット、およびマルチソースレプリケーションの監視の手順は、[セクション17.1.5「MySQL マルチソースレプリケーション」](#)を参照してください。

マルチソースレプリケーショントポロジの1つのレプリカサーバーに作成できるチャンネルの最大数は256です。 各レプリケーションチャンネルには、[セクション17.2.2.4「レプリケーションチャンネルのネーミング規則」](#)で説明されている一意の(空でない)名前が必要です。 マルチソースレプリケーションが有効な場合に発行されるエラーコードおよびメッセージは、エラーを生成したチャンネルを指定します。

注記

マルチソースレプリカ上の各チャンネルは、異なるソースからレプリケートする必要があります。 単一のレプリカから単一のソースに複数のレプリケーションチャンネルを設定することはできません。 これは、レプリカのサーバー ID がレプリケーショントポロジ内で一意である必要があるためです。 ソースはレプリカをサーバー ID でのみ区別し、レプリケーション

チャンネルの名前では区別しないため、同じレプリカとは異なるレプリケーションチャンネルを認識できません。

マルチソースレプリカは、`slave_parallel_workers` システム変数を 0 より大きい値に設定することで、マルチスレッドレプリカとして設定することもできます。マルチソースレプリカでこれを行う場合、レプリカ上の各チャンネルには、指定された数のアプライヤスレッドと、それらを管理するためのコーディネータスレッドがあります。個々のチャンネルのアプライヤスレッドの数は構成できません。

MySQL 8.0 から、特定のレプリケーションチャンネルでレプリケーションフィルタを使用してマルチソースレプリカを構成できます。チャンネル固有のレプリケーションフィルタは、同じデータベースまたはテーブルが複数のソースに存在し、単一のソースからレプリケートする場合にのみ使用できます。GTID ベースのレプリケーションでは、(ダイヤモンドポジなどの) 複数のソースから同じトランザクションが到着する可能性がある場合、フィルタリング設定がすべてのチャンネルで同じであることを確認する必要があります。詳細は、[セクション 17.2.5.4 「レプリケーションチャンネルベースのフィルタ」](#) を参照してください。

以前のバージョンとの互換性を提供するために、MySQL サーバーは起動時に、名前が空の文字列 ("") であるデフォルトチャンネルを自動的に作成します。このチャンネルは常に存在します。ユーザーが作成または破棄することはできません。他のチャンネル (空でない名前を持つ) が作成されていない場合、レプリケーションステートメントはデフォルトチャンネルでのみ機能するため、古いレプリカからのすべてのレプリケーションステートメントは予想どおりに機能します ([セクション 17.2.2.2 「以前のレプリケーションステートメントとの互換性」](#) を参照)。このセクションで説明するように、レプリケーションチャンネルに適用するステートメントは、少なくとも 1 つの名前付きチャンネルがある場合にのみ使用できます。

17.2.2.1 単一チャンネルで操作するためのコマンド

MySQL レプリケーション操作で個々のレプリケーションチャンネルを操作できるようにするには、次のレプリケーションステートメントで `FOR CHANNEL channel` 句を使用します:

- [CHANGE REPLICATION SOURCE TO](#)
- [CHANGE MASTER TO](#)
- [START REPLICA | SLAVE](#)
- [STOP REPLICA | SLAVE](#)
- [SHOW RELAYLOG EVENTS](#)
- [FLUSH RELAY LOGS](#)
- [SHOW REPLICA | SLAVE STATUS](#)
- [RESET REPLICA | SLAVE](#)

次の関数に追加の `channel` パラメータが導入されました:

- [MASTER_POS_WAIT\(\)](#)

`group_replication_recovery` チャンネルでは、次のステートメントは許可されません:

- [START REPLICA | SLAVE](#)
- [STOP REPLICA | SLAVE](#)

`group_replication_applier` チャンネルでは、次のステートメントは許可されません:

- [START REPLICA | SLAVE](#)
- [STOP REPLICA | SLAVE](#)
- [SHOW REPLICA | SLAVE STATUS](#)

`FLUSH RELAY LOGS` は `group_replication_applier` チャンネルに対して許可されるようになりましたが、トランザクションの適用中にリクエストを受信した場合、リクエストはトランザクションの終了後に実行されます。リクエストは、トランザクションが完了してローテーションが実行されるまで待機する必要があります。この動作により、グループレプリケーションで許可されていないトランザクションを分割できなくなります。

17.2.2.2 以前のレプリケーションステートメントとの互換性

レプリカに複数のチャンネルがあり、`FOR CHANNEL channel` オプションが指定されていない場合、通常、有効なステートメントは使用可能なすべてのチャンネルに対して動作しますが、特定の例外があります。

たとえば、次のステートメントは、特定のグループレプリケーションチャンネルを除くすべてのステートメントで予想どおりに動作します：

- `START REPLICA | SLAVE` は、`group_replication_recovery` および `group_replication_applier` チャンネルを除くすべてのチャンネルのレプリケーションスレッドを開始します。
- `STOP REPLICA | SLAVE` は、`group_replication_recovery` および `group_replication_applier` チャンネルを除くすべてのチャンネルのレプリケーションスレッドを停止します。
- `SHOW REPLICA | SLAVE STATUS` では、`group_replication_applier` チャンネルを除くすべてのチャンネルのステータスがレポートされます。
- `RESET REPLICA | SLAVE` はすべてのチャンネルをリセットします。

警告

このステートメントは既存のすべてのチャンネルを削除し、リレーログファイルをパージし、デフォルトチャンネルのみを再作成するため、`RESET REPLICA | SLAVE` を慎重に使用してください。

一部のレプリケーションステートメントは、すべてのチャンネルで動作するわけではありません。この場合、エラー 1964 `Multiple channels exist in replica` が生成されるため、チャンネル名を指定してください。次のステートメントおよび関数は、マルチソースレプリケーショントポロジで使用され、`FOR CHANNEL channel` オプションを使用して動作するチャンネルを指定しない場合に、このエラーを生成します：

- `SHOW RELAYLOG EVENTS`
- `CHANGE REPLICATION SOURCE TO`
- `CHANGE MASTER TO`
- `MASTER_POS_WAIT()`

デフォルトチャンネルは常に単一ソースレプリケーショントポロジに存在することに注意してください。このトポロジでは、ステートメントおよび関数は以前のバージョンの MySQL と同様に動作します。

17.2.2.3 起動オプションとレプリケーションチャンネル

このセクションでは、レプリケーションチャンネルの追加によって影響を受ける起動オプションについて説明します。

レプリケーションチャンネルを使用する場合は、`master_info_repository` および `relay_log_info_repository` システム変数を `FILE` に設定しないでください。MySQL 8.0 では、`FILE` 設定は非推奨であり、`TABLE` がデフォルトであるため、システム変数を省略できます。MySQL 8.0.23 では、使用は非推奨であるため、省略する必要があります。これらのシステム変数が `FILE` に設定されている場合、レプリカにソースを追加しようとすると、`ER_SLAVE_NEW_CHANNEL_WRONG_REPOSITORY` で失敗します。

次の起動オプションがレプリケーショントポロジの all チャンネルに影響するようになりました。

- `--log-slave-updates`

レプリカによって受信されたすべてのトランザクション (複数のソースからのトランザクションも含む) は、バイナリログに書き込まれます。

- `--relay-log-purge`

設定すると、各チャンネルは独自のリレーログを自動的にパージします。

- `--slave_transaction_retries`

指定された数のトランザクション再試行は、すべてのチャンネルのすべてのアプライヤスレッドで実行できます。

- `--skip-slave-start`

どのチャンネルでもレプリケーションスレッドは開始しません。

- `--slave-skip-errors`

すべてのチャンネルで実行が続行され、エラーはスキップされます。

次の起動オプションに設定された値は各チャンネルに適用されます。これらは `mysqld` の起動オプションであるため、すべてのチャンネルに適用されます。

- `--max-relay-log-size=size`

各チャンネルの個々のリレーログファイルの最大サイズ。この制限に達すると、ファイルはローテーションされます。

- `--relay-log-space-limit=size`

個々のチャンネルについて、結合されたすべてのリレーログの合計サイズの上限。N チャンネルの場合、これらのログの合計サイズは `relay_log_space_limit * N` に制限されます。

- `--slave-parallel-workers=value`

チャンネル当たりのレプリケーションアプライヤスレッド数。

- `slave_checkpoint_group`

各ソースの I/O スレッドによる待機時間。

- `--relay-log-index=filename`

各チャンネルリレーログインデックスファイルのベース名。 [セクション17.2.2.4「レプリケーションチャンネルのネーミング規則」](#) を参照してください。

- `--relay-log=filename`

各チャンネルリレーログファイルのベース名を示します。 [セクション17.2.2.4「レプリケーションチャンネルのネーミング規則」](#) を参照してください。

- `--slave_net_timeout=N`

この値はチャンネルごとに設定されるため、各チャンネルは N 秒間待機して切断された接続をチェックします。

- `--slave-skip-counter=N`

この値はチャンネルごとに設定されるため、各チャンネルはソースからの N イベントをスキップします。

17.2.2.4 レプリケーションチャンネルのネーミング規則

このセクションでは、レプリケーションチャンネルによるネーミング規則への影響について説明します。

各レプリケーションチャンネルの一意の名前は、最大 64 文字の文字列で、大/小文字は区別されません。チャンネル名はレプリカアプライアンスのメタデータリポジトリテーブルで使用されるため、これらに使用される文字セットは常に UTF-8 です。通常、チャンネルには任意の名前を自由に使用できますが、次の名前は予約されています:

- `group_replication_applier`

- `group_replication_recovery`

レプリケーションチャンネルに選択した名前は、マルチソースレプリカで使用されるファイル名にも影響します。各チャンネルのリレーログファイルとインデックスファイルには `relay_log_basename-channel.xxxxxx` という名前が付けられます。ここで、`relay_log_basename` は `relay_log` システム変数を使用して指定されたベース名、`channel` はこのファイルに記録されたチャンネルの名前です。 `relay_log` システム変数を指定しない場合は、チャンネルの名前も含むデフォルトのファイル名が使用されます。

17.2.3 レプリケーションスレッド

MySQL レプリケーション機能は、3つのメインスレッド(ソースサーバー上のスレッドとレプリカ上のスレッド)を使用して実装されます:

- **バイナリログダンプスレッド.** ソースは、レプリカの接続時にバイナリログの内容をレプリカに送信するスレッドを作成します。このスレッドは、ソース上の `SHOW PROCESSLIST` の出力で `Binlog Dump` スレッドとして識別できます。

バイナリログダンプスレッドは、レプリカに送信される各イベントを読み取るために、ソースバイナリログのロックを取得します。イベントが読み取られるとすぐに、イベントがレプリカに送信される前でもロックが解除されません。

- **レプリケーション I/O スレッド.** レプリカサーバーで `START REPLICHA | SLAVE` ステートメントが発行されると、レプリカは I/O スレッドを作成します。このスレッドはソースに接続し、バイナリログに記録された更新を送信するように要求します。

レプリケーション I/O スレッドは、ソース `Binlog Dump` スレッドが送信した更新を読み取り(前の項目を参照)、レプリカリレーログを構成するローカルファイルにコピーします。

このスレッドの状態は、`SHOW SLAVE STATUS` の出力に `Slave_IO_running` として表示されます。

- **レプリケーション SQL スレッド.** レプリカは、レプリケーション I/O スレッドによって書き込まれるリレーログを読み取り、そこに含まれるトランザクションを実行する SQL スレッドを作成します。

ソース/レプリカ接続ごとに3つのメインスレッドがあります。複数のレプリカを持つソースは、現在接続されているレプリカごとに1つのバイナリログダンプスレッドを作成し、各レプリカには独自のレプリケーション I/O および SQL スレッドがあります。

レプリカは2つのスレッドを使用して、読取り更新をソースから分離し、独立したタスクに実行します。したがって、トランザクションの適用プロセスが遅い場合、トランザクションの読取りタスクは遅くなりません。たとえば、レプリカサーバーがしばらく実行されていない場合、その I/O スレッドは、SQL スレッドが遠く遅れていても、レプリカの起動時にソースからすべてのバイナリログコンテンツをすばやくフェッチできます。SQL スレッドがフェッチされたすべてのステートメントを実行する前にレプリカが停止した場合、I/O スレッドは少なくともすべてをフェッチして、トランザクションの安全なコピーがレプリカリレーログにローカルに格納され、レプリカの次の起動時に実行できるようにします。

`slave_parallel_workers` システム変数を0(デフォルト)より大きい値に設定することで、レプリカのタスクに対してさらにパラレル化を有効にできます。このシステム変数を設定すると、レプリカは、トランザクションを適用するために指定された数のワーカースレッドと、それらを管理するためのコーディネータスレッドを作成します。複数のレプリケーションチャンネルを使用している場合、各チャンネルにはこの数のスレッドがあります。`slave_parallel_workers` が0より大きい値に設定されたレプリカは、マルチスレッドレプリカと呼ばれます。この設定では、失敗したトランザクションを再試行できます。

注記

マルチスレッドレプリカは現在 NDB Cluster でサポートされていないため、この変数の設定は暗黙的に無視されます。詳しくは [セクション23.6.3「NDB Cluster レプリケーションの既知の問題」](#) をご覧ください。

17.2.3.1 レプリケーションメインスレッドの監視

`SHOW PROCESSLIST` ステートメントは、レプリケーションに関してソースおよびレプリカで何が起きているかを示す情報を提供します。ソースの状態の詳細は、[セクション8.14.4「レプリケーションソーススレッドの状態」](#) を参照してください。レプリカの状態については、[セクション8.14.5「レプリケーション I/O スレッドの状態」](#) および [セクション8.14.6「レプリケーション SQL スレッドの状態」](#) を参照してください。

次の例は、バイナリログダンプスレッド、Replicatin I/O スレッド、およびレプリケーション SQL スレッドの3つのメインレプリケーションスレッドが `SHOW PROCESSLIST` からの出力にどのように表示されるかを示しています。

ソースサーバーでは、`SHOW PROCESSLIST` からの出力は次のようになります:

```
mysql> SHOW PROCESSLIST
***** 1. row *****
```

```

Id: 2
User: root
Host: localhost:32931
db: NULL
Command: Binlog Dump
Time: 94
State: Has sent all binlog to slave; waiting for binlog to
be updated
Info: NULL

```

ここで、スレッド 2 は接続レプリカにサービスを提供する **Binlog Dump** スレッドです。State 情報は、すべての未処理の更新がレプリカに送信され、ソースがさらに更新が発生するのを待機していることを示します。ソースサーバーに **Binlog Dump** スレッドが表示されない場合は、レプリケーションが実行されていないことを意味します。つまり、レプリカは現在接続されていません。

レプリカサーバーでは、**SHOW PROCESSLIST** からの出力は次のようになります:

```

mysql> SHOW PROCESSLIST
***** 1. row *****
  Id: 10
  User: system user
  Host:
  db: NULL
  Command: Connect
  Time: 11
  State: Waiting for master to send event
  Info: NULL
***** 2. row *****
  Id: 11
  User: system user
  Host:
  db: NULL
  Command: Connect
  Time: 11
  State: Has read all relay log; waiting for the slave I/O
        thread to update it
  Info: NULL

```

State 情報は、スレッド 10 がソースサーバーと通信しているレプリケーション I/O スレッドであり、スレッド 11 がリレーログに格納されている更新を処理しているレプリケーション SQL スレッドであることを示しています。**SHOW PROCESSLIST** が実行された時点で、両方のスレッドはアイドルで、後続の更新を待機中でした。

Time カラムの値は、レプリカがソースと比較される遅延を示すことができます。 [セクション A.14 「MySQL 8.0 FAQ: レプリケーション」](#) を参照してください。 **Binlog Dump** スレッドのアクティビティなしでソース側で十分な時間が経過すると、ソースはレプリカが接続されなくなったと判断します。ほかのクライアント接続に関して、このタイムアウトは `net_write_timeout` および `net_retry_count` の値によって異なります。これらの詳細については、 [セクション 5.1.8 「サーバーシステム変数」](#) を参照してください。

SHOW REPLICAS | SLAVE STATUS ステートメントは、レプリカサーバーでのレプリケーション処理に関する追加情報を提供します。 [セクション 17.1.7.1 「レプリケーションステータスの確認」](#) を参照してください。

17.2.3.2 レプリケーションアプライアンスワーカースレッドの監視

マルチスレッドレプリカでは、「パフォーマンススキーマ」テーブル `replication_applier_status_by_coordinator` および `replication_applier_status_by_worker` に、レプリカコーディネータスレッドおよびアプライアワーカースレッドのステータス情報がそれぞれ表示されます。複数のチャンネルを持つレプリカの場合、各チャンネルのスレッドが識別されます。

マルチスレッドレプリカコーディネータスレッドは、詳細設定が情報メッセージを表示するように設定されている場合、定期的にレプリカエラーログに統計も出力します。統計は、コーディネータスレッドがアプライアワーカースレッドに割り当てたイベントの量に応じて出力されます。最大頻度は 120 秒ごとに 1 回です。このメッセージには、関連するレプリケーションチャンネルまたはデフォルトのレプリケーションチャンネル(名前のない)に関する次の統計がリストされます:

経過秒数	この情報がエラーログに最後に出力された時間と現在の時間の差(秒)。
イベント割当済	コーディネータスレッドが起動されてから、すべてのアプライアワーカースレッドに対してコーディネータスレッドがキューに入れたイベントの合計数。

オーバーランレベルを超えて入力されたワーカーキュー	オーバーランレベルを超えて、任意のアプライワーカースレッドにキューに入れているイベントの現在の数。これは、16384 イベントの最大キュー長の 90% に設定されます。この値がゼロの場合、アプライワーカースレッドは容量の上限で動作していません。
ワーカーキューがいっぱいであるため待機中	アプライワーカースレッドキューがいっぱいであったために、コーディネータスレッドがイベントのスケジュールを待機する必要があった回数。この値がゼロの場合、アプライワーカースレッドは容量を使い果たしませんでした。
合計サイズのため待機中	<code>slave_pending_jobs_size_max</code> 制限に達したために、コーディネータスレッドがイベントのスケジュールを待機する必要があった回数。このシステム変数は、まだ適用されていないイベントを保持するアプライワーカースレッドキューで使用可能なメモリの最大量 (バイト) を設定します。異常に大きいイベントがこのサイズを超えると、すべてのアプライワーカースレッドに空のキューが設定されてから処理されるまで、トランザクションは保持されます。後続のすべてのトランザクションは、大規模なトランザクションが完了するまで保持されます。
クロック競合で待機中	イベントが依存していたトランザクションがまだコミットされていないために、コーディネータスレッドがイベントのスケジュールを待機する必要があった回数。 <code>slave_parallel_type</code> が (<code>LOGICAL_CLOCK</code> ではなく) <code>DATABASE</code> に設定されている場合、この値は常にゼロです。
ワーカーが占有している場合の待機 (件数)	コーディネータスレッドが短期間スリープした回数。これは 2 つの状況で実行される可能性があります。最初の状況では、コーディネータスレッドがイベントを割り当て、最大キュー長の 10% のアンダーランレベルを超えて適用者ワーカースレッドキューが一杯であることが判明します。この場合、最大 1 ミリ秒スリープします。2 つ目の状況では、 <code>slave_parallel_type</code> が <code>LOGICAL_CLOCK</code> に設定され、コーディネータのスレッドがトランザクションの最初のイベントをアプライアンスのワーカースレッドキューに割り当てる必要があり、これは空のキューを持つワーカーに対してのみ実行されるため、キューが空でない場合、コーディネータスレッドは空になるまでスリープします。
ワーカーが占有している場合に待機	空のアプライワーカースレッドキューを待機している間にコーディネータスレッドがスリープしたナノ秒数 (つまり、前述の 2 つ目の状況では、 <code>slave_parallel_type</code> が <code>LOGICAL_CLOCK</code> に設定され、トランザクションの最初のイベントを割り当てる必要があります)。

17.2.4 リレーログおよびレプリケーションメタデータリポジトリ

レプリカサーバーは、レプリケーションプロセスに使用する情報のリポジトリをいくつか作成します:

- レプリケーション I/O スレッドによって書き込まれるレプリカリレーログには、レプリケーションソースサーバーのバイナリログから読み取られたトランザクションが含まれます。リレーログ内のトランザクションは、レプリケーション SQL スレッドによってレプリカに適用されます。リレーログについては、[セクション 17.2.4.1 「リレーログ」](#) を参照してください。
- レプリカ接続メタデータリポジトリには、レプリケーション I/O スレッドがレプリケーションソースサーバーに接続し、ソースバイナリログからトランザクションを取得するために必要な情報が含まれています。接続メタデータリポジトリが `mysql.slave_master_info` テーブルに書き込まれます。
- レプリカ適用者メタデータリポジトリには、レプリケーション SQL スレッドがレプリカリレーログからトランザクションを読み取り、適用するために必要な情報が含まれています。アプライメタデータリポジトリが `mysql.slave_relay_log_info` テーブルに書き込まれます。

レプリカ接続メタデータリポジトリと適用者メタデータリポジトリは、まとめてレプリケーションメタデータリポジトリと呼ばれます。詳細は、[セクション 17.2.4.2 「レプリケーションメタデータリポジトリ」](#) を参照してください。

予期しない停止に対するレプリケーションの回復性の実現。 `mysql.slave_master_info` および `mysql.slave_relay_log_info` テーブルは、トランザクションストレージエンジン InnoDB を使用して作成されます。レプリカアプライアンスのメタデータリポジトリテーブルへの更新は、トランザクションとともにコミットされます。つまり、予期しないサーバーが停止した場合でも、そのリポジトリに記録されるレプリカ進捗情報は、常にデータベースに適用されているものと一貫性があります。予期しない停止に対して最も回復可能なレプリカの設定の組合せの詳細は、[セクション 17.4.2 「レプリカの予期しない停止の処理」](#) を参照してください。

17.2.4.1 リレーログ

リレーログは、バイナリログと同様に、データベース変更を記述するイベントを含む番号付きファイルのセットと、使用されたすべてのリレーログファイルの名前を含むインデックスファイルとで構成されます。リレーログファイルのデフォルトの場所はデータディレクトリです。

用語「リレーログファイル」は一般的に、データベースイベントを含む個々の番号付きファイルを示します。用語「リレーログ」は、番号付きリレーログファイルとインデックスファイルのセットの総称です。

リレーログファイルはバイナリログファイルと同じ形式で、`mysqlbinlog` を使用して読み取ることができます (セクション4.6.8「`mysqlbinlog` — バイナリログファイルを処理するためのユーティリティ」を参照)。バイナリログのトランザクション圧縮 (MySQL 8.0.20 の時点で使用可能) が使用されている場合、リレーログに書き込まれるトランザクションペイロードはバイナリログと同じ方法で圧縮されます。バイナリログのトランザクション圧縮の詳細は、セクション5.4.4.5「`バイナリログトランザクション圧縮`」を参照してください。

デフォルトのレプリケーションチャンネルの場合、リレーログファイル名のデフォルト形式は `host_name-relay-bin.nnnnnn` です。ここで、`host_name` はレプリカサーバーホストの名前、`nnnnnn` はシーケンス番号です。連続するリレーログファイルは、`000001` で始まる連続シーケンス番号を使用して作成されます。デフォルト以外のレプリケーションチャンネルの場合、デフォルトのベース名は `host_name-relay-bin-channel` です。ここで、`channel` はリレーログに記録されたレプリケーションチャンネルの名前です。

レプリカはインデックスファイルを使用して、現在使用されているリレーログファイルを追跡します。デフォルトのリレーログインデックスファイル名は、デフォルトチャンネルの場合は `host_name-relay-bin.index`、デフォルト以外のレプリケーションチャンネルの場合は `host_name-relay-bin-channel.index` です。

デフォルトのリレーログファイルとリレーログインデックスファイルの名前と場所は、それぞれ `relay_log` および `relay_log_index` システム変数でオーバーライドできます (セクション17.1.6「`レプリケーションおよびバイナリロギングのオプションと変数`」を参照)。

レプリカがデフォルトのホストベースのリレーログファイル名を使用する場合、レプリケーションの設定後にレプリカホスト名を変更すると、「`リレーログのオープンに失敗しました`」および「`リレーログの初期化中にターゲットログが見つかりませんでした`」のエラーでレプリケーションが失敗する可能性があります。これは既知の問題です (Bug #2122 を参照してください)。将来レプリカホスト名が変更される可能性があるとして予想される場合 (たとえば、DHCP を使用してホスト名を変更できるようにレプリカにネットワークが設定されている場合)、レプリカの初期設定時に `relay_log` および `relay_log_index` システム変数を使用してリレーログファイル名を明示的に指定することで、この問題を完全に回避できます。これにより、名前はサーバーのホスト名の変更とは無関係になります。

レプリケーションの開始後に問題が発生した場合に対処するには、レプリカサーバーを停止し、古いリレーログインデックスファイルの内容を新しいファイルの先頭に追加してからレプリカを再起動します。Unix システムでは、ここで示すようにこれを実行できます。

```
shell> cat new_relay_log_name.index >> old_relay_log_name.index
shell> mv old_relay_log_name.index new_relay_log_name.index
```

複製サーバーは、次の条件下で新しいリレーログファイルを作成します:

- レプリケーション I/O スレッドが開始されるたび。
- ログがフラッシュされるタイミング (`FLUSH LOGS` や `mysqladmin flush-logs` などを使用)。
- 現在のリレーログファイルのサイズが大きすぎる場合は、次のように決定されます:
 - `max_relay_log_size` (最大リレーログファイルサイズ) の値が 0 より大きい場合。
 - `max_relay_log_size` の値が 0 の場合、`max_binlog_size` が最大リレーログファイルサイズを判断します。

レプリケーション SQL スレッドは、ファイル内のすべてのイベントを実行し、不要になったあと、各リレーログファイルを自動的に削除します。レプリケーション SQL スレッドがリレーログを削除するための明示的なメカニズムはありません。ただし、`FLUSH LOGS` はリレーログをローテーションするため、レプリケーション SQL スレッドがリレーログを削除するタイミングに影響します。

17.2.4.2 レプリケーションメタデータリポジトリ

レプリカサーバーは、接続メタデータリポジトリと適用者メタデータリポジトリの 2 つのレプリケーションメタデータリポジトリを作成します。レプリケーションメタデータリポジトリは、レプリカサーバーの停止後も存続します。

バイナリログファイルの位置ベースのレプリケーションを使用している場合、レプリカは再起動時に 2 つのリポジトリを読み取り、以前にソースからバイナリログを読み取り、独自のリレーログを処理していた距離を判断します。GTID ベースのレプリケーションが使用されている場合、レプリカはその目的のためにレプリケーションメタデータリポジトリを使用しませんが、含まれている他のメタデータのために必要です。

- レプリカ接続メタデータリポジトリには、レプリケーション I/O スレッドがレプリケーションソースサーバーに接続し、ソースバイナリログからトランザクションを取得するために必要な情報が含まれています。このリポジトリのメタデータには、接続構成、レプリケーションユーザーアカウントの詳細、接続の SSL 設定、レプリケーション I/O スレッドがソースバイナリログから現在読み取っているファイル名と位置が含まれます。
- レプリカ適用者メタデータリポジトリには、レプリケーション SQL スレッドがレプリカリレーログからトランザクションを読み取り、適用するために必要な情報が含まれています。このリポジトリのメタデータには、レプリケーション SQL スレッドがリレーログ内のトランザクションを実行したファイル名と位置、およびソースバイナリログ内の同等の位置が含まれます。また、ワーカースレッドの数やチャンネルの `PRIVILEGE_CHECKS_USER` アカウントなど、トランザクションを適用するプロセスのメタデータも含まれます。

接続メタデータリポジトリは `mysql` システムスキーマの `slave_master_info` テーブルに書き込まれ、適用者メタデータリポジトリは `mysql` システムスキーマの `slave_relay_log_info` テーブルに書き込まれます。mysqld がレプリケーションメタデータリポジトリのテーブルを初期化できないが、レプリカの起動を続行できる場合は、警告メッセージが発行されます。この状況は、リポジトリのテーブルの使用をサポートしていない MySQL のバージョンから、サポートされているバージョンにアップグレードする場合に最も発生する可能性があります。

重要

- `mysql.slave_master_info` または `mysql.slave_relay_log_info` テーブルの行を手動で更新または挿入しないでください。そのようにすることは、未定義の動作になる可能性があります。サポートされていません。レプリケーションの進行中は、`slave_master_info` テーブルと `slave_relay_log_info` テーブルのいずれかまたは両方で書き込みロックを必要とするステートメントの実行は許可されません (ただし、読み取りのみを実行するステートメントはいつでも許可されます)。
- 接続メタデータリポジトリテーブル `mysql.slave_master_info` のアクセス権限は、ソースに接続するためのレプリケーションユーザーアカウント名とパスワードが含まれているため、データベース管理者に制限する必要があります。このテーブルを含むデータベースバックアップを保護するには、制限付きアクセスモードを使用します。MySQL 8.0.21 から、レプリケーションユーザーアカウントの資格証明を接続メタデータリポジトリからクリアし、かわりにレプリケーションチャンネルを起動する `START REPLICA | SLAVE` ステートメントまたは `START GROUP_REPLICATION` ステートメントを使用してそれらを常に指定できます。このアプローチでは、レプリケーションチャンネルを再起動するためにオペレータの介入が常に必要ですが、アカウント名とパスワードはレプリケーションメタデータリポジトリに記録されません。

`RESET REPLICA | SLAVE` は、レプリケーション接続パラメータ (MySQL Server リリースによって異なります) を除いて、レプリケーションメタデータリポジトリ内のデータをクリアします。詳細は、`RESET REPLICA | SLAVE` の説明を参照してください。

MySQL 8.0 より前は、レプリケーションメタデータリポジトリをテーブルとして作成するには、サーバーの起動時に `master_info_repository=TABLE` および `relay_log_info_repository=TABLE` を指定する必要がありました。それ以外の場合、リポジトリは、`master.info` および `relay-log.info` という名前のデータディレクトリにファイルとして作成されたか、`--master-info-file` オプションおよび `relay_log_info_file` システム変数で指定された代替の名前と場所を使用して作成されました。MySQL 8.0 からは、レプリケーションメタデータリポジトリをテーブルとして作成することがデフォルトであり、これらすべてのシステム変数の使用は非推奨になりました。

`mysql.slave_master_info` および `mysql.slave_relay_log_info` テーブルは、InnoDB トランザクションストレージエンジンを使用して作成されます。アプライヤメタデータリポジトリテーブルへの更新は、トランザクションとともにコミットされます。つまり、予期しないサーバーが停止した場合でも、そのリポジトリに記録されるレプリカ進捗情報は、常にデータベースに適用されているものと一貫性があります。予期しない停止に対して最も回復可能なレプリカの設定の組合せの詳細は、[セクション 17.4.2 「レプリカの予期しない停止の処理」](#) を参照してください。

レプリカデータをバックアップするか、そのデータのスナップショットを転送して新しいレプリカを作成する場合は、レプリケーションメタデータリポジトリを含む `mysql.slave_master_info` および `mysql.slave_relay_log_info` テーブルを必ず含めてください。クローニング操作の場合、レプリケーションメタデータリポジトリがテーブルとして

作成されると、クローニング操作中に受信者にコピーされますが、ファイルとして作成されるとコピーされないことに注意してください。バイナリログファイルの位置ベースのレプリケーションが使用されている場合、レプリケーションメタデータリポジトリは、復元、コピー、またはクローニングされたレプリカの再起動後にレプリケーションを再開するために必要です。リレーログファイルはないが、まだアプライヤメタデータリポジトリがある場合は、それをチェックして、レプリケーション SQL スレッドがソースバイナリログで実行された距離を判断できます。その後、[CHANGE REPLICATION SOURCE TO](#) ステートメント (MySQL 8.0.23 から) または [CHANGE MASTER TO](#) ステートメント (MySQL 8.0.23 より前) を [SOURCE_LOG_FILE](#) | [MASTER_LOG_FILE](#) および [SOURCE_LOG_POS](#) | [MASTER_LOG_POS](#) オプションとともに使用して、必要なバイナリログをソースログに再度読み取り、バイナリログからレプリカを読み取ることができます。

追加のリポジトリであるアプライヤワーカーメタデータリポジトリは、主に内部使用のために作成され、マルチスレッドレプリカ上のワーカースレッドに関するステータス情報を保持します。アプライヤワーカーメタデータリポジトリには、リレーログファイルの名前と位置、および各ワーカースレッドのソースバイナリログファイルが含まれます。アプライヤメタデータリポジトリがテーブルとして作成されている場合 (デフォルト)、アプライヤワーカーメタデータリポジトリは [mysql.slave_worker_info](#) テーブルに書き込まれます。適用者メタデータリポジトリがファイルに書き込まれると、適用者ワーカーメタデータリポジトリが [worker-relay-log.info](#) ファイルに書き込まれます。外部で使用するために、ワーカースレッドのステータス情報がパフォーマンススキーマ [replication_applier_status_by_worker](#) テーブルに表示されます。

レプリケーションメタデータリポジトリには、[セクション13.4.2「レプリケーションサーバーを制御するための SQL ステートメント」](#) で説明されている [SHOW REPLICAS | SLAVE STATUS](#) ステートメントの出力に示されているような情報が最初に含まれていました。[SHOW REPLICAS | SLAVE STATUS](#) ステートメントで表示されないレプリケーションメタデータリポジトリに追加された情報。

接続メタデータリポジトリの場合、次のテーブルに、[mysql.slave_master_info](#) テーブルのカラム、[SHOW REPLICAS | SLAVE STATUS](#) によって表示されるカラムおよび非推奨の [master.info](#) ファイルの行の対応関係を示します。

mysql.slave_master_info テーブルのカラム	SHOW REPLICAS SLAVE STATUS カラム	master.info ファイル行	説明
Number_of_lines	[None]	1	テーブル (またはファイル内の行) のカラム数
Master_log_name	Source_Log_File	2	ソースから現在読み取られているバイナリログの名前
Master_log_pos	Read_Source_Log_Pos	3	ソースから読み取られたバイナリログ内の現在の位置
Host	Source_Host	4	レプリケーションソースサーバーのホスト名
User_name	Source_User	5	ソースへの接続に使用されるレプリケーションユーザーアカウント名
User_password	パスワード (SHOW REPLICAS SLAVE STATUS では表示されません)	6	ソースへの接続に使用されるレプリケーションユーザーアカウントのパスワード
Port	Source_Port	7	レプリケーションソースサーバーへの接続に使用されるネットワークポート
Connect_retry	Connect_Retry	8	レプリカがソースへの再接続を試行するまで待機する期間 (秒)
Enabled_ssl	Source_SSL_Allowed	9	レプリカが SSL 接続をサポートするかどうか
Ssl_ca	Source_SSL_CA_File	10	認証局 (CA) 証明書に使用されるファイル
Ssl_capath	Source_SSL_CA_Path	11	認証局 (CA) 証明書へのパス

slave_master_info テーブルの カラム	SHOW REPLICA SLAVE STATUS カラム	master.info ファイル行	説明
Ssl_cert	Source_SSL_Cert	12	SSL 証明書ファイルの名前
Ssl_cipher	Source_SSL_Cipher	13	SSL 接続のハンドシェイク で使用される可能な暗号の リスト
Ssl_key	Source_SSL_Key	14	SSL キーファイルの名前
Ssl_verify_server_cert	Source_SSL_Verify_Server_Cert	15	サーバー証明書を検証する かどうか
Heartbeat	[None]	16	レプリケーションハート ビートの間隔 (秒単位)
Bind	Source_Bind	17	ソースへの接続に使用する レプリカネットワークイン タフェース
Ignored_server_ids	Replicate_Ignore_Server_Ids	18	無視するサーバー ID のリス ト。 Ignored_server_ids の 場合、サーバー ID のリス トの前に、無視するサーバー ID の合計数が付きます。
Uuid	Source_UUID	19	ソースの一意の ID
Retry_count	Source_Retry_Count	20	許容される再接続試行の最 大数
Ssl_crl	[None]	21	SSL 証明書失効リストファ イルへのパス
Ssl_crlpath	[None]	22	SSL 証明書失効リストファ イルを含むディレクトリへ のパス
Enabled_auto_position	Auto_position	23	GTID 自動配置が使用中かど うか
Channel_name	Channel_name	24	レプリケーションチャネル の名前
Tls_version	Source_TLS_Version	25	ソースの TLS バージョン
Public_key_path	Source_public_key_path	26	RSA 公開キーファイルの名 前
Get_public_key	Get_source_public_key	27	ソースから RSA 公開キーを リクエストするかどうか
Network_namespace	Network_namespace	28	ネットワークネームスパー ス
Master_compression_algorithm	[None]	29	ソースへの接続に許可され る圧縮アルゴリズム
Master_zstd_compression_level	[None]	30	zstd 圧縮レベル
Tls_ciphersuites	[None]	31	TLSv1.3 で許可される暗号 スイート
Source_connection_auto_failover	[None]	32	非同期接続フェイルオー バーメカニズムをアクティ ブ化するかどうか

アプライメタデータリポジトリの場合、次のテーブルに、mysql.slave_relay_log_info テーブルのカラム、SHOW REPLICA | SLAVE STATUS によって表示されるカラムおよび非推奨の relay-log.info ファイルの行の対応関係を示します。

slave_relay_log_info テーブルの カラム	SHOW REPLICA SLAVE STATUS カラム	relay-log.info ファイルの行	説明
Number_of_lines	[None]	1	ファイル内のテーブルまたは 行のカラム数
Relay_log_name	Relay_Log_File	2	現在のリレーログファイル の名前
Relay_log_pos	Relay_Log_Pos	3	リレーログファイル内の現 在の位置。この位置までの イベントがレプリカデータ ベース上で実行されていま す
Master_log_name	Relay_Source_Log_File	4	リレーログファイル内のイ ベントが読み取られたソー スバイナリログファイルの 名前
Master_log_pos	Exec_Source_Log_Pos	5	レプリカで実行されたイベ ントのソースバイナリログ ファイル内の同等の位置
Sql_delay	SQL_Delay	6	レプリカがソースを遅らせ る必要がある秒数
Number_of_workers	[None]	7	レプリケーションランザ クションを平行に適用 するワーカースレッドの数
Id	[None]	8	内部目的に使用される ID。 現在は常に 1 です
Channel_name	Channel_name	9	レプリケーションチャネル の名前
Privilege_checks_username	[None]	10	チャネルの PRIVILEGE_CHECKS_USER アカウントのユーザー名
Privilege_checks_hostname	[None]	11	チャネルの PRIVILEGE_CHECKS_USER アカウントのホスト名
Require_row_format	[None]	12	チャネルが行ベースのイベ ントのみを受け入れるかど うか
Require_table_primary_key_c onstraints	[None]	13	CREATE TABLE および ALTER TABLE 操作のため にテーブルに主キーが必要 かどうかに関するチャネル ポリシー
Assign_gtids_to_anonymous_ replicas_type	[None]	14	チャネルが GTID をまだ 持っていないレプリケー トランザクションに割り 当てるかどうか、および割 り当てられている場合はレ プリカローカル UUID を使 用するか手動で設定した UUID を使用するかどうか
Assign_gtids_to_anonymous_ replicas_value	[None]	15	匿名トランザクションに割 り当てられた GTID で使用 される UUID

17.2.5 サーバーがレプリケーションフィルタリングルールをどのように評価するか

レプリケーションソースサーバーがステートメントをバイナリログに書き込まない場合、ステートメントはレプリケートされません。サーバーがステートメントをログに記録する場合、ステートメントはすべてのレプリカに送信され、各レプリカはステートメントを実行するか無視するかを決定します。

ソースでは、`--binlog-do-db` および `--binlog-ignore-db` オプションを使用してバイナリロギングを制御することによって、変更を記録するデータベースを制御できます。これらのオプションを評価するときにサーバーが使用するルールの詳細は、[セクション17.2.5.1「データベースレベルレプリケーションオプションおよびバイナリロギングオプションの評価」](#)を参照してください。複製するデータベースとテーブルを制御するためにこれらのオプションを使用しないでください。代わりに、レプリカでフィルタリングを使用して、レプリカで実行されるイベントを制御します。

レプリカ側では、ソースから受け取ったステートメントを実行するか無視するかに関する決定は、レプリカが起動された `--replicate-*` オプションに従って行われます。[セクション17.1.6「レプリケーションおよびバイナリロギングのオプションと変数」](#)を参照してください。これらのオプションによって制御されるフィルタは、`CHANGE REPLICATION FILTER` ステートメントを使用して動的に設定することもできます。このようなフィルタを制御するルールは、`--replicate-*` オプションを使用して起動時に作成されるか、`CHANGE REPLICATION FILTER` によってレプリカサーバーが実行されている間に作成されるかに関係なく同じです。レプリケーションフィルタは、グループレプリケーション用に構成された MySQL サーバーインスタンス上のグループレプリケーション固有のチャンネルでは使用できません。これは、一部のサーバーでトランザクションをフィルタすると、グループが一貫性のある状態で承諾に到達できなくなるためです。

最も単純なケースでは、`--replicate-*` オプションがない場合、レプリカはソースから受信したすべてのステートメントを実行します。そうでない場合は、結果は指定された特定のオプションに依存します。

データベースレベルオプション (`--replicate-do-db`、`--replicate-ignore-db`) が最初にチェックされます。このプロセスの説明については、[セクション17.2.5.1「データベースレベルレプリケーションオプションおよびバイナリロギングオプションの評価」](#)を参照してください。データベースレベルオプションが使用されない場合は、オプションチェックはテーブルレベルオプション (使用されている場合がある) のチェックに進みます (これらの説明については、[セクション17.2.5.2「テーブルレベルレプリケーションオプションの評価」](#)を参照してください)。1つ以上のデータベースレベルオプションが使用されているけれども、一致するものがない場合は、ステートメントは複製されません。

データベースにのみ影響するステートメントの場合 (つまり、`CREATE DATABASE`、`DROP DATABASE`、および `ALTER DATABASE`)、データベースレベルオプションは `--replicate-wild-do-table` オプションより常に優先されます。つまり、このようなステートメントの場合、適用するデータベースレベルオプションがない場合にかぎり、`--replicate-wild-do-table` オプションがチェックされます。

特定のオプションセットにどのような影響があるかを簡単に判断できるように、`do-*` オプションと `ignore-*` オプション、またはワイルドカードを含むオプションとそれ以外のオプションを混在させないことをお勧めします。

`--replicate-rewrite-db` オプションが指定された場合、それらは `--replicate-*` フィルタリングルールがテストされる前に適用されます。

注記

すべてのレプリケーションフィルタリングオプションは、`lower_case_table_names` システム変数の影響を含め、MySQL サーバーの他の場所にあるデータベースおよびテーブルの名前に適用される大/小文字の区別について同じルールに従います。

17.2.5.1 データベースレベルレプリケーションオプションおよびバイナリロギングオプションの評価

レプリケーションオプションを評価する場合、レプリカはまず、適用される `--replicate-do-db` または `--replicate-ignore-db` オプションがあるかどうかを確認します。`--binlog-do-db` または `--binlog-ignore-db` を使用する場合、プロセスは似ていますが、オプションはソースでチェックされます。

一致がチェックされるデータベースは、処理されるステートメントのバイナリログ形式によって異なります。ステートメントが行形式を使用してログに記録されている場合、データが変更されるデータベースはチェックされるデータベースです。ステートメントがステートメントの形式を使用してログに記録されている場合、デフォルトのデータベース (`USE` ステートメントで指定) がチェックされるデータベースになります。

注記

行形式を使用してログに記録できるのは DML ステートメントのみです。DDL ステートメントは、`binlog_format=ROW` の場合でも、常にステートメントとして記録されます。したがって、すべての DDL ステートメントは、ステートメントベースのレプリケーションのルールに従って常にフィルタ処理されます。つまり、DDL ステートメントを適用するには、`USE` ステートメントを使用してデフォルトのデータベースを明示的に選択する必要があります。

レプリケーションの場合、関連するステップは次のとおりです：

1. どのロギング形式が使用されますか。
 - `STATEMENT`. デフォルトのデータベースをテストします。
 - `ROW`. 変更の影響を受けるデータベースをテストします。
2. `--replicate-do-db` オプションはありますか？
 - はい。 データベースはこれらのいずれかと一致しますか。
 - はい。 ステップ 4 に進みます。
 - いいえ。 更新を無視して終了します。
 - いいえ。 手順 3 に進みます。
3. `--replicate-ignore-db` オプションはありますか？
 - はい。 データベースはこれらのいずれかと一致しますか。
 - はい。 更新を無視して終了します。
 - いいえ。 手順 4 に進みます。
 - いいえ。 手順 4 に進みます。
4. テーブルレベルレプリケーションオプションがある場合、それらの検査に進みます。これらのオプションの検査方法の説明については、[セクション 17.2.5.2 「テーブルレベルレプリケーションオプションの評価」](#) を参照してください。

重要

この段階でまだ許可されているステートメントは、実際にはまだ実行されていません。ステートメントはすべてのテーブルレベルオプション（ある場合）が検査されるまで実行されず、そのプロセスの結果がステートメントの実行を許可します。

バイナリロギングの場合、関連する手順の一覧は次のとおりです。

1. `--binlog-do-db` または `--binlog-ignore-db` オプションはありますか？
 - はい。 手順 2 に進みます。
 - いいえ。 ステートメントのログを記録して終了します。
2. デフォルトデータベースはありますか（データベースが `USE` で選択されていますか）？
 - はい。 手順 3 に進みます。
 - いいえ。 ステートメントを無視して終了します。
3. デフォルトデータベースがあります。 `--binlog-do-db` オプションはありますか？
 - はい。 それらのいずれかがデータベースに一致しますか？

- はい。 ステートメントのログを記録して終了します。
 - いいえ。 ステートメントを無視して終了します。
 - いいえ。 手順 4 に進みます。
4. `--binlog-ignore-db` オプションのいずれかがデータベースに一致しますか？
- はい。 ステートメントを無視して終了します。
 - いいえ。 ステートメントのログを記録して終了します。

重要

ステートメントベースロギングの場合、`CREATE DATABASE`、`ALTER DATABASE`、および `DROP DATABASE` ステートメントに適用されるルールにだけ例外が作成されています。これらの場合には、更新のログを記録または無視するかを判断するときに、作成、変更、またはドロップされるデータベースがデフォルトデータベースを置き換えます。

`--binlog-do-db` は「ほかのデータベースを無視する」ことを意味する場合があります。たとえば、ステートメントベースロギングを使用するときに、`--binlog-do-db=sales` だけで動作するサーバーは、デフォルトデータベースが `sales` ではないバイナリログステートメントに書き込みません。同じオプションで行ベースロギングを使用するときは、サーバーは `sales` 内のデータを変更する更新のみのログを記録します。

17.2.5.2 テーブルレベルレプリケーションオプションの評価

レプリカは、次の 2 つの条件のいずれかに該当する場合にのみ、テーブルオプションをチェックして評価します：

- 一致するデータベースオプションが見つからなかった。
- 1 つ以上のデータベースオプションが見つかり、前のセクションで説明したルール (セクション 17.2.5.1 「データベースレベルレプリケーションオプションおよびバイナリロギングオプションの評価」を参照) に従って、「実行」条件に達していると評価された。

まず、予備的な状態として、レプリカはステートメントベースのレプリケーションが有効かどうかをチェックします。その場合、ステートメントがストアドファンクション内で発生すると、レプリカはステートメントを実行して終了します。行ベースレプリケーションが有効になっている場合、レプリカはソースのストアドファンクション内でステートメントが発生したかどうかを認識しないため、この条件は適用されません。

注記

ステートメントベースレプリケーションの場合、レプリケーションイベントがステートメントを表現します (あるイベントを構成するすべての変更が単一 SQL ステートメントに関連付けられています)。行ベースレプリケーションの場合、各イベントが単一テーブル行内の変更を表現します (このため、`UPDATE mytable SET mycol = 1` などの単一ステートメントが多くの行ベースイベントを生成する場合があります)。イベントの観点から見ると、テーブルオプションを検査するプロセスは行ベースおよびステートメントベースレプリケーションの両方について同じです。

この時点に到達すると、テーブルオプションがない場合、レプリカは単にすべてのイベントを実行します。`--replicate-do-table` または `--replicate-wild-do-table` オプションがある場合は、それが実行すべきイベントの場合、イベントはこれらのいずれかに一致する必要があります。そうでない場合、無視されます。`--replicate-ignore-table` または `--replicate-wild-ignore-table` オプションがある場合、これらのオプションのいずれかに一致するものを除いてすべてのイベントが実行されます。

次のステップでは、この評価について詳しく説明します。開始ポイントは、セクション 17.2.5.1 「データベースレベルレプリケーションオプションおよびバイナリロギングオプションの評価」で説明されているように、データベースレベルのオプションの評価の最後です。

1. テーブルレプリケーションオプションはありますか。
- はい。 手順 2 に進みます。

- いいえ. 更新を実行して終了します。
2. どのロギング形式が使用されますか。
 - STATEMENT. 更新を実行するステートメントごとに残りのステップを実行します。
 - ROW. テーブルの行を更新するたびに残りのステップを実行します。
 3. `--replicate-do-table` オプションはありますか？
 - はい. テーブルはそれらのいずれかに一致しますか？
 - はい. 更新を実行して終了します。
 - いいえ. 手順 4 に進みます。
 - いいえ. 手順 4 に進みます。
 4. `--replicate-ignore-table` オプションはありますか？
 - はい. テーブルはそれらのいずれかに一致しますか？
 - はい. 更新を無視して終了します。
 - いいえ. 手順 5 に進みます。
 - いいえ. 手順 5 に進みます。
 5. `--replicate-wild-do-table` オプションはありますか？
 - はい. テーブルはそれらのいずれかに一致しますか？
 - はい. 更新を実行して終了します。
 - いいえ. 手順 6 に進みます。
 - いいえ. 手順 6 に進みます。
 6. `--replicate-wild-ignore-table` オプションはありますか？
 - はい. テーブルはそれらのいずれかに一致しますか？
 - はい. 更新を無視して終了します。
 - いいえ. 手順 7 に進みます。
 - いいえ. 手順 7 に進みます。
 7. テストする別のテーブルがありますか。
 - はい. ステップ 3 に戻ります。
 - いいえ. 手順 8 に進みます。
 8. `--replicate-do-table` または `--replicate-wild-do-table` オプションはありますか？
 - はい. 更新を無視して終了します。
 - いいえ. 更新を実行して終了します。

注記

単一の SQL ステートメントが `--replicate-do-table` または `--replicate-wild-do-table` オプションに含まれるテーブルと、`--replicate-ignore-table` または `--replicate-wild-ignore-table` オプションで無視される別のテーブルの両方で動作する場合、ステートメントベースのレプリ

ケーションは停止します。レプリカは、(レプリケーションイベントを形成する) 完全なステートメントを実行または無視する必要があり、論理的にはこれを実行できません。DDL ステートメントは常に有効なロギング形式に関係なくステートメントとして記録されるため、これは DDL ステートメントの行ベースのレプリケーションにも適用されます。インクルードされたテーブルと無視されたテーブルの両方を更新でき、正常にレプリケートされる唯一のタイプのステートメントは、`binlog_format=ROW` でログに記録された DML ステートメントです。

17.2.5.3 レプリケーションフィルタリングオプション間の相互作用

データベースレベルとテーブルレベルのレプリケーションフィルタリングオプションの組合せを使用する場合、レプリカは最初にデータベースオプションを使用してイベントを受け入れるか無視し、次にテーブルオプションに従ってそれらのオプションで許可されるすべてのイベントを評価します。これにより、結果が直感的に印刷される場合があります。また、ステートメントベースと行ベースのどちらのバイナリロギング形式を使用して操作をログに記録するかによって結果が異なることにも注意してください。レプリケーションフィルタがバイナリロギング形式とは独立して常に同じ方法で動作するようにする場合は、特に混合バイナリロギング形式を使用する場合に重要です。このトピックのガイダンスに従ってください。

レプリケーションフィルタリングオプションの効果は、データベース名が識別される方法のため、バイナリロギング形式によって異なります。ステートメントベースの形式では、DML ステートメントは、`USE` ステートメントステートメントで指定された現在のデータベースに基づいて処理されます。行ベースの形式では、DML ステートメントは、変更されたテーブルが存在するデータベースに基づいて処理されます。DDL ステートメントは、バイナリロギング形式に関係なく、常に `USE` ステートメントで指定された現在のデータベースに基づいてフィルタ処理されます。

複数のテーブルを含む操作は、バイナリロギング形式に応じて、レプリケーションフィルタリングオプションの影響を受けることもあります。監視する操作には、複数テーブルの `UPDATE` ステートメント、トリガー、外部キーのカスケード、複数のテーブルを更新するストアドファンクション、および 1 つ以上のテーブルを更新するストアドファンクションを起動する DML ステートメントが含まれます。これらの操作によってフィルタ処理されたテーブルとフィルタ処理されたテーブルの両方が更新される場合、結果はバイナリロギング形式によって異なる可能性があります。

特に混合バイナリロギング形式 (`binlog_format=MIXED`) を使用している場合、レプリケーションフィルタがバイナリロギング形式に関係なく一貫して動作することを保証する必要がある場合は、テーブルレベルのレプリケーションフィルタリングオプションのみを使用し、データベースレベルのレプリケーションフィルタリングオプションは使用しないでください。また、フィルタ処理されたテーブルとフィルタ処理されたテーブルの両方を更新する複数テーブル DML ステートメントは使用しないでください。

データベースレベルとテーブルレベルのレプリケーションフィルタの組合せを使用する必要があり、これらを可能なかぎり一貫して動作させる場合は、次のいずれかの方法を選択します:

1. DDL ステートメントに行ベースのバイナリロギング形式 (`binlog_format=ROW`) を使用する場合は、`USE` ステートメントを使用してデータベースを設定し、データベース名を指定しないでください。レプリケーションフィルタリングとの一貫性を向上させるために、行ベースのバイナリロギング形式に変更することを検討できます。バイナリロギング形式の変更に適用される条件については、[セクション 5.4.4.2 「バイナリログ形式の設定」](#) を参照してください。
2. ステートメントベースまたは混合バイナリロギング形式 (`binlog_format=STATEMENT` または `MIXED`) を DML ステートメントと DDL ステートメントの両方に使用する場合は、`USE` ステートメントに依存し、データベース名は使用しません。また、フィルタ処理されたテーブルとフィルタ処理されたテーブルの両方を更新する複数テーブル DML ステートメントは使用しないでください。

例 17.7 `--replicate-ignore-db` オプションおよび `--replicate-do-table` オプション

レプリケーションソースサーバーでは、次のステートメントが発行されます:

```
USE db1;
CREATE TABLE t2 LIKE t1;
INSERT INTO db2.t3 VALUES (1);
```

レプリカには、次のレプリケーションフィルタリングオプションが設定されています:

```
replicate-ignore-db = db1
```

```
replicate-do-table = db2.t3
```

DDL ステートメント `CREATE TABLE` は、前述の `USE` ステートメントで指定されているように、`db1` にテーブルを作成します。`db1` が現在のデータベースであるため、レプリカは `--replicate-ignore-db = db1` オプションに従ってこのステートメントを除外します。この結果は、レプリケーションソースサーバー上のバイナリロギング形式と同じです。ただし、DML `INSERT` ステートメントの結果は、バイナリロギング形式によって異なります:

- 行ベースのバイナリロギング形式がソース (`binlog_format=ROW`) で使用されている場合、レプリカは、`db2` という名前のテーブルが存在するデータベースを使用して `INSERT` 操作を評価します。最初に評価されるデータベースレベルのオプション `--replicate-ignore-db = db1` は適用されません。テーブルレベルのオプション `--replicate-do-table = db2.t3` は適用されるため、レプリカはテーブル `t3` に変更を適用します。
- ステートメントベースのバイナリロギング形式がソース (`binlog_format=STATEMENT`) で使用されている場合、レプリカは、`USE` ステートメントによって `db1` に設定され、変更されていないデフォルトのデータベースを使用して `INSERT` 操作を評価します。したがって、データベースレベルの `--replicate-ignore-db = db1` オプションによると、操作は無視され、変更はテーブル `t3` に適用されません。ステートメントがすでにデータベースレベルのオプションと一致し、無視されたため、テーブルレベルのオプション `--replicate-do-table = db2.t3` はチェックされません。

レプリカで `--replicate-ignore-db = db1` オプションが必要であり、ソースでステートメントベース (または混在) のバイナリロギング形式を使用する必要もある場合は、次のように `INSERT` ステートメントからデータベース名を省略し、代わりに `USE` ステートメントに依存することで、結果を整合させることができます:

```
USE db1;
CREATE TABLE t2 LIKE t1;
USE db2;
INSERT INTO t3 VALUES (1);
```

この場合、レプリカは常にデータベース `db2` に基づいて `INSERT` ステートメントを評価します。操作がステートメントベースまたは行ベースのバイナリ形式のどちらで記録されているかにかかわらず、結果は同じままです。

17.2.5.4 レプリケーションチャンネルベースのフィルタ

このセクションでは、マルチソースレプリケーショントポロジなどに複数のレプリケーションチャンネルが存在する場合にレプリケーションフィルタを使用する方法について説明します。MySQL 8.0 より前は、レプリケーションフィルタはグローバルであったため、すべてのレプリケーションチャンネルにフィルタが適用されていました。MySQL 8.0 からは、レプリケーションフィルタをグローバルまたはチャンネル固有にでき、特定のレプリケーションチャンネルでレプリケーションフィルタを使用してマルチソースレプリカを構成できます。チャンネル固有のレプリケーションフィルタは、同じデータベースまたはテーブルが複数のソースに存在し、レプリカが単一のソースからレプリケートする必要がある場合に、マルチソースレプリケーショントポロジで特に役立ちます。

レプリケーションチャンネルの設定手順は、[セクション 17.1.5 「MySQL マルチソースレプリケーション」](#) を参照してください。レプリケーションチャンネルの動作の詳細は、[セクション 17.2.2 「レプリケーションチャンネル」](#) を参照してください。

重要

マルチソースレプリカ上の各チャンネルは、異なるソースからレプリケートする必要があります。レプリケーションフィルタを使用してチャンネルごとにレプリケートする異なるデータを選択した場合でも、単一のレプリカから単一のソースに複数のレプリケーションチャンネルを設定することはできません。これは、レプリカのサーバー ID がレプリケーショントポロジ内で一意である必要があるためです。ソースはレプリカをサーバー ID でのみ区別し、レプリケーションチャンネルの名前では区別しないため、同じレプリカとは異なるレプリケーションチャンネルを認識できません。

重要

グループレプリケーション用に構成された MySQL サーバーインスタンスでは、グループレプリケーションに直接関係しないレプリケーションチャンネル (グループメンバーがグループ外のソースへのレプリカとしても機能する場合など) でチャンネル固有のレプリケーションフィルタを使用できません。`group_replication_applier` または `group_replication_recovery` チャンネルでは使用できません。これらのチャンネルをフィルタすると、グループは一貫性のある状態でアグリーメントに到達できなくなります。

重要

ダイヤモンドトポロジ内のマルチソースレプリカ (レプリカが複数のソースからレプリケートされ、共通ソースからレプリケートされる) の場合、GTID ベースのレプリケーションが使用されているときは、マルチソースレプリカ上のすべてのチャンネルでレプリケーションフィルタまたはその他のチャンネル構成が同一であることを確認してください。GTID ベースのレプリケーションでは、フィルタはトランザクションデータにのみ適用され、GTID はフィルタで除外されません。これは、レプリカの GTID セットがソースと一貫性が保たれるようにするためです。つまり、毎回フィルタで除外されたトランザクションを再取得せずに GTID 自動配置を使用できます。ダウンストリームレプリカがマルチソースで、ダイヤモンドトポロジの複数のソースから同じトランザクションを受信する場合、ダウンストリームレプリカには複数のバージョンのトランザクションが含まれるようになり、結果はトランザクションを最初に適用するチャンネルによって異なります。トランザクションの GTID が最初のチャンネルによって `gtid_executed` セットに追加されたため、GTID 自動スキップを使用してトランザクションをスキップしようとする 2 つ目のチャンネル。チャンネルのフィルタリングが同一の場合、トランザクションのすべてのバージョンに同じデータが含まれているため、結果は同じであるため、問題はありません。ただし、チャンネルのフィルタリングが異なると、データベースに一貫性がなくなる可能性があり、レプリケーションがハングする可能性があります。

レプリケーションフィルタおよびチャンネルの概要

マルチソースレプリケーショントポロジなどに複数のレプリケーションチャンネルが存在する場合、レプリケーションフィルタは次のように適用されます:

- 指定したグローバルレプリケーションフィルタは、フィルタタイプ (`do_db`、`do_ignore_table` など) のグローバルレプリケーションフィルタに追加されます。
- チャンネル固有のレプリケーションフィルタは、指定されたフィルタタイプの指定されたチャンネルのレプリケーションフィルタにフィルタを追加します。
- このタイプのチャンネル固有のレプリケーションフィルタが構成されていない場合、各レプリケーションチャンネルはグローバルレプリケーションフィルタをチャンネル固有のレプリケーションフィルタにコピーします。
- 各チャンネルは、チャンネル固有のレプリケーションフィルタを使用してレプリケーションストリームをフィルタリングします。

チャンネル固有のレプリケーションフィルタを作成する構文は、既存の SQL ステートメントおよびコマンドオプションを拡張します。レプリケーションチャンネルが指定されていない場合、グローバルレプリケーションフィルタは下位互換性を保証するように構成されます。 `CHANGE REPLICATION FILTER` ステートメントでは、チャンネル固有のフィルタをオンラインで構成する `FOR CHANNEL` 句がサポートされています。フィルタを構成するための `--replicate-*` コマンドオプションでは、`--replicate-filter_type=channel_name:filter_details` の形式を使用してレプリケーションチャンネルを指定できます。たとえば、サーバーの起動前にチャンネル `channel_1` および `channel_2` が存在し、コマンドラインオプション `--replicate-do-db=db1 --replicate-do-db=channel_1:db2 --replicate-do-db=db3 --replicate-ignore-db=db4 --replicate-ignore-db=channel_2:db5` を使用してレプリカを起動すると、次のようになります:

- グローバルレプリケーションフィルタ: `do_db=db1,db3, ignore_db=db4`
- `channel_1` でのチャンネル固有のフィルタ: `do_db=db2 ignore_db=db4`
- `channel_2` でのチャンネル固有のフィルタ: `do_db=db1,db3 ignore_db=db5`

このような設定でレプリケーションフィルタを監視するには、`replication_applier_global_filters` テーブルと `replication_applier_filters` テーブルを使用します。

起動時のチャンネル固有のレプリケーションフィルタの構成

レプリケーションフィルタ関連のコマンドオプションには、オプションの `channel` に続けてコロンを指定し、その後ろにフィルタ指定を指定できます。最初のコロンはセパレータとして解釈され、後続のコロンはリテラルコロンとして解釈されます。次のコマンドオプションは、この形式を使用したチャンネル固有のレプリケーションフィルタをサポートしています:

- `--replicate-do-db=channel:database_id`
- `--replicate-ignore-db=channel:database_id`
- `--replicate-do-table=channel:table_id`
- `--replicate-ignore-table=channel:table_id`
- `--replicate-rewrite-db=channel:db1-db2`
- `--replicate-wild-do-table=channel:table regexid`
- `--replicate-wild-ignore-table=channel:table regexid`

コロンを使用するが、`--replicate-do-db=:database_id`などのフィルタオプションに `channel` を指定しない場合、このオプションはデフォルトのレプリケーションチャンネルのレプリケーションフィルタを構成します。デフォルトのレプリケーションチャンネルは、レプリケーションが開始されると常に存在するレプリケーションチャンネルで、手動で作成するマルチソースレプリケーションチャンネルとは異なります。コロンも `channel` も指定されていない場合は、グローバルレプリケーションフィルタが構成されます。たとえば、`--replicate-do-db=database_id` はグローバル `--replicate-do-db` フィルタを構成します。

同じ `from_name` データベースで複数の `rewrite-db=from_name->to_name` オプションを構成すると、すべてのフィルタがまとめて追加され (`rewrite_do` リストに入れられます)、最初のフィルタが有効になります。

チャンネル固有のレプリケーションフィルタのオンラインでの変更

`--replicate-*` オプションに加えて、`CHANGE REPLICATION FILTER` ステートメントを使用してレプリケーションフィルタを構成できます。これにより、サーバーを再起動する必要がなくなりますが、変更中はレプリケーション SQL スレッドを停止する必要があります。このステートメントで特定のチャンネルにフィルタを適用するには、`FOR CHANNEL channel` 句を使用します。例:

```
CHANGE REPLICATION FILTER REPLICATE_DO_DB=(db1) FOR CHANNEL channel_1;
```

`FOR CHANNEL` 句が指定されている場合、ステートメントは指定されたチャンネルレプリケーションフィルタに対して動作します。複数のタイプのフィルタ (`do_db`, `do_ignore_table`, `wild_do_table` など) が指定されている場合、指定されたフィルタタイプのみがステートメントで置換されます。マルチソースレプリカなどの複数のチャンネルがあるレプリケーショントポロジでは、`FOR CHANNEL` 句が指定されていない場合、`FOR CHANNEL` の場合と同様のロジックを使用して、グローバルレプリケーションフィルタおよびすべてのチャンネルのレプリケーションフィルタに対してステートメントが機能します。詳細は、[セクション13.4.2.2「CHANGE REPLICATION FILTER ステートメント」](#)を参照してください。

チャンネル固有のレプリケーションフィルタの削除

チャンネル固有のレプリケーションフィルタが構成されている場合は、空の `filter type` ステートメントを発行してフィルタを削除できます。たとえば、`channel_1` issue という名前のレプリケーションチャンネルからすべての `REPLICATE_REWRITE_DB` フィルタを削除するには、次のコマンドを発行します:

```
CHANGE REPLICATION FILTER REPLICATE_REWRITE_DB=() FOR CHANNEL channel_1;
```

コマンドオプションまたは `CHANGE REPLICATION FILTER` を使用して以前に構成された `REPLICATE_REWRITE_DB` フィルタは削除されます。

`RESET REPLICA | SLAVE ALL` ステートメントは、ステートメントによって削除されたチャンネルに設定されたチャンネル固有のレプリケーションフィルタを削除します。削除されたチャンネルが再作成されると、レプリカに指定されたグローバルレプリケーションフィルタがそれらにコピーされ、チャンネル固有のレプリケーションフィルタは適用されません。

17.3 レプリケーションのセキュリティ

レプリケーションソースサーバーとレプリカ間で格納および転送されるデータへの不正なアクセスから保護するには、[第6章「セキュリティ」](#)で説明されているように、インストール内の任意の MySQL インスタンスに対して選

択するセキュリティ対策を使用して、関係するすべてのサーバーを設定します。また、レプリケーショントポロジ内のサーバーの場合は、次のセキュリティ対策を実装することを検討してください:

- 暗号化された接続を使用してバイナリログを転送するようにソースとレプリカを設定します。これにより、このデータは移動中に保護されます。これらの接続の暗号化は、暗号化されたネットワーク接続をサポートするようにサーバーを設定するだけでなく、[CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO](#) ステートメントを使用してアクティブ化する必要があります。 [セクション17.3.1「暗号化接続を使用するためのレプリケーションの設定」](#)を参照してください。
- バイナリログファイルを暗号化し、ソースおよびレプリカ上のリレーログファイルを暗号化します。これにより、このデータが保存され、バイナリログキャッシュで使用されているデータも保護されます。バイナリログの暗号化は、`binlog_encryption` システム変数を使用してアクティブ化されます。 [セクション17.3.2「バイナリログファイルとリレーログファイルの暗号化」](#)を参照してください。
- 権限チェックをレプリケーションアプライアンスに適用します。これは、権限のある操作または不要な操作の不正または偶然の使用からレプリケーションチャンネルを保護するのに役立ちます。権限チェックは、[PRIVILEGE_CHECKS_USER](#) アカウントを設定することで実装されます。このアカウントは、そのチャンネルの特定の各トランザクションが認可されていることを確認するために MySQL で使用されます。 [セクション17.3.3「レプリケーション権限チェック」](#)を参照してください。

グループレプリケーションでは、バイナリログの暗号化および権限チェックをレプリケーショングループメンバーのセキュリティ対策として使用できます。また、グループメンバー間の接続の暗号化、グループ通信接続と分散リカバリ接続の構成、および信頼できないホストを除外するための IP アドレスの割当ての適用も検討する必要があります。Group Replication に固有のこれらのセキュリティ対策については、[セクション18.5「グループレプリケーションセキュリティ」](#)を参照してください。

17.3.1 暗号化接続を使用するためのレプリケーションの設定

レプリケーション中に必要なバイナリログの転送に暗号化された接続を使用するには、ソースサーバーと複製サーバーの両方が暗号化されたネットワーク接続をサポートしている必要があります。いずれかのサーバーが暗号化された接続をサポートしていない場合(それらに対してコンパイルまたは構成されていないため)、暗号化された接続を介したレプリケーションはできません。

レプリケーション用の暗号化された接続の設定は、クライアント/サーバー接続の場合と同様です。ソースで使用できる適切なセキュリティ証明書と、各レプリカで(同じ認証局から)同様の証明書を取得(または作成)する必要があります。適切なキーファイルも取得する必要があります。

暗号化された接続のためのサーバーおよびクライアントの設定の詳細は、[セクション6.3.1「暗号化接続を使用するための MySQL の構成」](#)を参照してください。

ソースで暗号化された接続を有効にするには、適切な証明書およびキーファイルを作成または取得し、必要に応じてファイル名を変更して、ソース `my.cnf` ファイルの`[mysqld]`セクション内のソース構成に次の構成パラメータを追加する必要があります:

```
[mysqld]
ssl_ca=cacert.pem
ssl_cert=server-cert.pem
ssl_key=server-key.pem
```

ファイルへのパスは相対パスまたは絶対パスである可能性があります。この目的には、常に完全なパスを使用することをお勧めします。

構成パラメータは次のとおりです:

- `ssl_ca`: 認証局 (CA) 証明書ファイルのパス名。(`ssl_capath` は類似していますが、CA 証明書ファイルのディレクトリのパス名を指定します。)
- `ssl_cert`: サーバー公開キー証明書ファイルのパス名。この証明書はクライアントに送信し、クライアントが持っている CA 証明書に対して認証できます。
- `ssl_key`: サーバー秘密キーファイルのパス名。

レプリカで暗号化された接続を有効にするには、(MySQL 8.0.23 の) [CHANGE REPLICATION SOURCE TO](#) ステートメントまたは (MySQL 8.0.23 の前の) [CHANGE MASTER TO](#) ステートメントを使用します。レプリカ `my.cnf` ファイ

ルの[client]セクションで、暗号化された接続に必要なレプリカ証明書および SSL 秘密キーファイルに名前を付けることも、[CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO](#) ステートメントを使用してその情報を明示的に指定することもできます。

- オプションファイルを使用してレプリカ証明書およびキーファイルに名前を付けるには、レプリカ `my.cnf` ファイルの[client]セクションに次の行を追加し、必要に応じてファイル名を変更します:

```
[client]
ssl-ca=cacert.pem
ssl-cert=client-cert.pem
ssl-key=client-key.pem
```

- レプリカがソースに接続しないようにするには、`--skip-slave-start` オプションを使用してレプリカサーバーを再起動します。[CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO](#) を使用してソース構成を指定し、暗号化を使用して接続するための `MASTER_SSL` オプションを追加します:

```
mysql> CHANGE MASTER TO
-> MASTER_HOST='source_hostname',
-> MASTER_USER='repl',
-> MASTER_PASSWORD='password',
-> MASTER_SSL=1;

Or from MySQL 8.0.23:
mysql> CHANGE REPLICATION SOURCE TO
-> SOURCE_HOST='source_hostname',
-> SOURCE_USER='repl',
-> SOURCE_PASSWORD='password',
-> SOURCE_SSL=1;
```

レプリケーション接続用の `SOURCE_SSL=1 | MASTER_SSL=1` を設定し、それ以上の `SOURCE_SSL_xxx | MASTER_SSL_xxx` オプションを設定しないことは、[暗号化接続のコマンドオプション](#) で説明されているように、クライアント用の `--ssl-mode=REQUIRED` の設定に対応します。`SOURCE_SSL=1 | MASTER_SSL=1` では、暗号化された接続を確立できる場合のみ接続が成功します。レプリケーション接続は暗号化されていない接続にフォールバックしないため、レプリケーションの `--ssl-mode=PREFERRED` 設定に対応する設定はありません。`SOURCE_SSL=0 | MASTER_SSL=0` が設定されている場合、これは `--ssl-mode=DISABLED` に対応します。

- [CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO](#) ステートメントを使用してレプリカ証明書および SSL 秘密キーファイルに名前を付けるには、レプリカ `my.cnf` ファイルで指定していない場合は、適切な `SOURCE_SSL_xxx | MASTER_SSL_xxx` オプションを追加します:

```
-> MASTER_SSL_CA = 'ca_file_name',
-> MASTER_SSL_CAPATH = 'ca_directory_name',
-> MASTER_SSL_CERT = 'cert_file_name',
-> MASTER_SSL_KEY = 'key_file_name';
```

これらのオプションは、[暗号化接続のコマンドオプション](#) で説明されている同じ名前の `--ssl-xxx` オプションに対応しています。これらのオプションを有効にするには、`MASTER_SSL=1` も設定する必要があります。レプリケーション接続の場合、`MASTER_SSL_CA` または `MASTER_SSL_CAPATH` のいずれかの値を指定するか、レプリカ `my.cnf` ファイルでこれらのオプションを指定すると、`--ssl-mode=VERIFY_CA` の設定に対応します。接続試行は、指定された情報を使用して有効な一致する認証局 (CA) 証明書が見つかった場合にのみ成功します。

- ホスト名アイデンティティ検証をアクティブ化するには、`MASTER_SSL_VERIFY_SERVER_CERT` オプションを追加します:

```
-> MASTER_SSL_VERIFY_SERVER_CERT=1;
```

このオプションは、MySQL 5.7 から非推奨になり、MySQL 8.0 で削除された `--ssl-verify-server-cert` オプションに対応します。レプリケーション接続の場合、[暗号化接続のコマンドオプション](#) で説明されているように、`MASTER_SSL_VERIFY_SERVER_CERT=1` の指定は `--ssl-mode=VERIFY_IDENTITY` の設定に対応します。このオプションを有効にするには、`MASTER_SSL=1` も設定する必要があります。ホスト名アイデンティティ検証は、自己署名証明書では機能しません。

- 証明書失効リスト (CRL) チェックをアクティブ化するには、`MASTER_SSL_CRL` または `MASTER_SSL_CRLPATH` オプションを追加します:

```
-> MASTER_SSL_CRL = 'crl_file_name';
```



```
-> MASTER_SSL_CRLPATH = 'crl_directory_name',
```

これらのオプションは、[暗号化接続のコマンドオプション](#) で説明されている同じ名前の `--ssl-xxx` オプションに対応しています。指定しない場合、CRL チェックは行われません。

- レプリケーション接続のレプリカで許可される暗号、暗号スイートおよび暗号化プロトコルのリストを指定するには、[MASTER_SSL_CIPHER](#)、[MASTER_TLS_VERSION](#) および [MASTER_TLS_CIPHERSUITES](#) オプションを使用します:

```
-> MASTER_SSL_CIPHER = 'cipher_list',  
-> MASTER_TLS_VERSION = 'protocol_list',  
-> MASTER_TLS_CIPHERSUITES = 'ciphersuite_list',
```

- [MASTER_SSL_CIPHER](#) オプションは、レプリケーション接続のレプリカで許可される暗号のコロン区切りリストを指定します。
- [MASTER_TLS_VERSION](#) オプションは、レプリケーション接続のレプリカで許可される TLS 暗号化プロトコルのコンマ区切りリストを、`tls_version` サーバースystem変数と同様の形式で指定します。接続手順では、ソースとレプリカの両方で許可されている最上位 TLS バージョンの使用をネゴシエーションします。接続できるようにするには、レプリカにソースと共通の TLS バージョンが少なくとも 1 つ必要です。
- [MASTER_TLS_CIPHERSUITES](#) オプション (MySQL 8.0.19 から使用可能) は、TLSv1.3 が接続に使用されている場合にレプリケーション接続のレプリカで許可される暗号スイートのコロン区切りリストを指定します。TLSv1.3 の使用時にこのオプションが `NULL` に設定されている場合 (オプションを設定しない場合のデフォルト)、デフォルトで有効になっている暗号スイートが許可されます。オプションを空の文字列に設定した場合、暗号スイートは許可されないため、TLSv1.3 は使用されません。

これらのリストで指定できるプロトコル、暗号および暗号スイートは、MySQL のコンパイルに使用される SSL ライブラリによって異なります。フォーマット、許可される値、およびオプションを指定しない場合のデフォルトの詳細は、[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#) を参照してください。

注記

8.0.18 を介した MySQL 8.0.16 では、MySQL は TLSv1.3 をサポートしていますが、[MASTER_TLS_CIPHERSUITES](#) オプションは使用できません。これらのリリースでは、ソースとレプリカ間の接続に TLSv1.3 を使用する場合、デフォルトで有効になっている少なくとも 1 つの TLSv1.3 暗号スイートの使用をソースで許可する必要があります。MySQL 8.0.19 から、オプションを使用して、デフォルト以外の暗号スイートのみを含め、任意の暗号スイートの選択を指定できます。

- ソース情報が更新されたら、レプリカでレプリケーションプロセスを開始します:

```
mysql> START SLAVE;  
Or from MySQL 8.0.22:  
mysql> START REPLICA;
```

[SHOW REPLICA | SLAVE STATUS](#) ステートメントを使用して、暗号化された接続が正常に確立されたことを確認できます。

- レプリカで暗号化された接続を要求しても、ソースがレプリカからの暗号化された接続を必要とすることは保証されません。ソースが暗号化された接続を使用して接続するレプリカのみを受け入れるようにするには、[REQUIRE SSL](#) オプションを使用してソースにレプリケーションユーザーアカウントを作成し、そのユーザーに [REPLICATION SLAVE](#) 権限を付与します。例:

```
mysql> CREATE USER 'repl'@'%example.com' IDENTIFIED BY 'password'  
-> REQUIRE SSL;  
mysql> GRANT REPLICATION SLAVE ON *.*  
-> TO 'repl'@'%example.com';
```

ソースに既存のレプリケーションユーザーアカウントがある場合は、次のステートメントを使用して [REQUIRE SSL](#) を追加できます:

```
mysql> ALTER USER 'repl'@'%example.com' REQUIRE SSL;
```

17.3.2 バイナリログファイルとリレーログファイルの暗号化

MySQL 8.0.14 からは、バイナリログファイルとリレーログファイルを暗号化できるため、これらのファイルとそれらに含まれる潜在的機密データを、外部の攻撃者による誤用から保護したり、格納されているオペレーティングシステムのユーザーによる不正な表示から保護したりできます。ファイルに使用される暗号化アルゴリズム AES (Advanced Encryption Standard) 暗号アルゴリズムは、MySQL Server に組み込まれており、構成できません。

MySQL サーバーでこの暗号化を有効にするには、`binlog_encryption` システム変数を `ON` に設定します。`OFF` がデフォルトです。システム変数は、バイナリログファイルおよびリレーログファイルの暗号化をオンに設定します。暗号化を有効にするためにバイナリロギングをサーバー上で有効にする必要はないため、バイナリログのないレプリカ上のリレーログファイルを暗号化できます。暗号化を使用するには、MySQL Server 鍵リングサービスを提供するように鍵リングプラグインをインストールして構成する必要があります。これを行う手順は、[セクション 6.4.4 「MySQL キーリング」](#) を参照してください。サポートされている任意のキーリングプラグインを使用して、バイナリログ暗号化鍵を格納できます。

暗号化を有効にしてサーバーを初めて起動すると、バイナリログとリレーログが初期化される前に新しいバイナリログ暗号化鍵が生成されます。このキーは、各バイナリログファイル (サーバーでバイナリロギングが有効になっている場合) およびリレーログファイル (サーバーにレプリケーションチャンネルがある場合) のファイルパスワードを暗号化するために使用され、ファイルパスワードから生成された以降の鍵を使用してファイル内のデータが暗号化されます。サーバーで現在使用されているバイナリログ暗号化鍵は、バイナリログマスターキーと呼ばれます。2 層暗号化キーアーキテクチャでは、必要に応じてバイナリログマスターキーをローテーション (新しいマスターキーに置換) でき、各ファイルのファイルパスワードのみを新しいマスターキーで再暗号化する必要があり、ファイル全体ではありません。リレーログファイルは、暗号化がアクティブ化された後に作成される新しいチャンネルを含め、すべてのチャンネルに対して暗号化されます。バイナリログインデックスファイルとリレーログインデックスファイルは暗号化されません。

サーバーの実行中に暗号化をアクティブ化すると、その時点で新しいバイナリログ暗号化鍵が生成されます。例外は、以前にサーバー上で暗号化がアクティブであり、その後無効になっていた場合です。その場合、以前に使用されていたバイナリログ暗号化鍵が再度使用されます。バイナリログファイルとリレーログファイルはただちにローテーションされ、新しいファイルとそれ以降のすべてのバイナリログファイルおよびリレーログファイルのファイルパスワードは、このバイナリログ暗号化鍵を使用して暗号化されます。既存のバイナリログファイルとリレーログファイルはまだサーバー上に存在していても暗号化されませんが、不要になった場合はパージできます。

`binlog_encryption` システム変数を `OFF` に変更して暗号化を非アクティブにすると、バイナリログファイルとリレーログファイルはただちにローテーションされ、それ以降のすべてのロギングは暗号化されません。以前に暗号化されたファイルは自動的に復号化されませんが、サーバーはそれらを読み取ることができます。サーバーの実行中に暗号化をアクティブ化または非アクティブ化するには、`BINLOG_ENCRYPTION_ADMIN` 権限が必要です。

暗号化されたバイナリログファイルと暗号化されていないバイナリログファイルは、暗号化されたログファイル (`0xFD62696E`) のファイルヘッダーの先頭にあるマジック番号を使用して区別できます。これは、暗号化されていないログファイル (`0xFE62696E`) に使用されるものとは異なります。`SHOW BINARY LOGS` ステートメントは、各バイナリログファイルが暗号化されているか暗号化されていないかを示します。

バイナリログファイルが暗号化されている場合、`mysqlbinlog` はそれらを直接読み取ることはできませんが、`--read-from-remote-server` オプションを使用してサーバーから読み取ることができます。MySQL 8.0.14 から、暗号化されたバイナリログファイルを直接読み取ろうとしたが、古いバージョンの `mysqlbinlog` がそのファイルをバイナリログファイルとして認識しない場合、`mysqlbinlog` は適切なエラーを返します。`mysqlbinlog` を使用して暗号化バイナリログファイルをバックアップする場合、`mysqlbinlog` を使用して生成されるファイルのコピーは暗号化されていない形式で格納されることに注意してください。

バイナリログ暗号化は、バイナリログトランザクション圧縮 (MySQL 8.0.20 の時点で使用可能) と組み合わせることができます。バイナリログのトランザクション圧縮の詳細は、[セクション 5.4.4.5 「バイナリログトランザクション圧縮」](#) を参照してください。

17.3.2.1 バイナリログの暗号化の範囲

MySQL サーバーインスタンスのバイナリログ暗号化がアクティブな場合、暗号化の範囲は次のとおりです:

- バイナリログファイルおよびリレーログファイルに書き込まれる保存データは、前述の 2 層暗号化アーキテクチャーを使用して、暗号化が開始された時点から暗号化されます。暗号化を開始したときにサーバーに存在していた既存のバイナリログファイルおよびリレーログファイルは暗号化されません。これらのファイルは、不要になったときにパージできます。

- `mysqlbinlog` を含む MySQL クライアントに送信されるレプリケーションイベントストリーム内の移動中のデータは、転送用に復号化されるため、接続暗号化を使用して転送中に保護する必要があります ([セクション6.3「暗号化された接続の使用」](#) および [セクション17.3.1「暗号化接続を使用するためのレプリケーションの設定」](#) を参照)。
- トランザクション中にバイナリログトランザクションおよびステートメントキャッシュに保持される使用中のデータは、キャッシュを格納するメモリーバッファ内で暗号化されていない形式です。データは、メモリーバッファで使用可能な領域を超えると、ディスク上の一時ファイルに書き込まれます。MySQL 8.0.17 から、バイナリログ暗号化がサーバー上でアクティブな場合、バイナリログキャッシュを保持するために使用される一時ファイルは、ストリーム暗号化に AES-CTR (AES カウンタモード) を使用して暗号化されます。一時ファイルは揮発性であり、単一のプロセスに関連付けられているため、ランダムに生成されたファイルパスワードと初期化ベクトルを使用して単一層暗号化を使用して暗号化され、メモリー内のみ存在し、ディスクまたはキーリングに格納されることはありません。各トランザクションがコミットされると、バイナリログキャッシュがリセットされます: メモリーバッファがクリアされ、バイナリログキャッシュの保持に使用される一時ファイルが切り捨てられ、次のトランザクションで使用するために新しいファイルパスワードおよび初期化ベクトルがランダムに生成されます。このリセットは、通常の停止または予期しない停止の後にサーバーが再起動された場合にも行われます。

注記

`binlog_format=STATEMENT` が設定されているときに `LOAD DATA` を使用する場合 (ステートメントがステートメントベースのレプリケーションで安全でないのみなされるため、お薦めしません)、変更が適用されるレプリカにデータを含む一時ファイルが作成されます。これらの一時ファイルは、バイナリログの暗号化がサーバー上でアクティブな場合は暗号化されません。代わりに、一時ファイルを作成しない行ベースまたは混合バイナリロギング形式を使用してください。

17.3.2.2 バイナリログ暗号化キー

ログファイルのファイルパスワードの暗号化に使用されるバイナリログ暗号化キーは 256 ビットキーで、MySQL Server キーリングサービスを使用して各 MySQL サーバーインスタンス専用生成されます ([セクション6.4.4「MySQL キーリング」](#) を参照)。鍵リングサービスは、バイナリログ暗号化鍵の作成、取得、および削除を処理します。サーバーインスタンスは、自身に対して生成されたキーの作成と削除のみを行います。ファイルコピーによってクローニングされたサーバーインスタンスの場合と同様に、キーリングに格納されている他のインスタンスに対して生成されたキーを読み取ることができます。

重要

MySQL サーバーインスタンスのバイナリログ暗号化キーは、バックアップおよびリカバリ手順に含める必要があります。これは、現在および保持されているバイナリログファイルまたはリレーログファイルのファイルパスワードの復号化に必要なキーが失われた場合、サーバーを起動できない可能性があるためです。

鍵リング内のバイナリログ暗号化鍵の形式は次のとおりです:

```
MySQLReplicationKey_{UUID}_{SEQ_NO}
```

例:

```
MySQLReplicationKey_00508583-b5ce-11e8-a6a5-0010e0734796_1
```

{UUID} は、MySQL サーバーによって生成される真の UUID (`server_uuid` システム変数の値) です。{SEQ_NO} はバイナリログ暗号化キーの順序番号で、サーバーで生成される新しいキーごとに 1 ずつ増分されます。

サーバーで現在使用されているバイナリログ暗号化鍵は、バイナリログマスターキーと呼ばれます。現在のバイナリログマスターキーのシーケンス番号は、鍵リングに格納されます。バイナリログマスターキーは、新しい各ログファイルパスワードを暗号化するために使用されます。これは、ファイルデータの暗号化に使用されるログファイルに固有のランダムに生成された 32 バイトのファイルパスワードです。ファイルパスワードは AES-CBC (AES 暗号ブロックチェーンモード) を使用して 256 ビットバイナリログ暗号化キーおよびランダム初期化ベクトル (IV) とともに暗号化され、ログファイルヘッダーに格納されます。ファイルデータは AES-CTR (AES カウンタモード) を使用して暗号化され、ファイルパスワードから 256 ビット鍵が生成され、nonce もファイルパスワードから生成されます。ファイルパスワードの暗号化に使用されるバイナリログ暗号化キーがわかっている場合は、OpenSSL 暗号化ツールキットで使用可能なツールを使用して、暗号化ファイルを技術的にオフラインで復号化できます。

バイナリログファイルおよびリレーログファイルが暗号化されるように、ファイルコピーを使用して暗号化がアクティブな MySQL サーバーインスタンスをクローニングする場合は、クローンサーバーがソースサーバーからバイナリログ暗号化キーを読み取ることができるように、キーリングもコピーされていることを確認します。クローンサーバーで暗号化がアクティブ化されると(起動時またはその後)、クローンサーバーは、コピーされたファイルで使用されるバイナリログ暗号化キーにソースサーバーの生成済 UUID が含まれていることを認識します。独自に生成された UUID を使用して新しいバイナリログ暗号化鍵を自動的に生成し、これを使用して後続のバイナリログファイルおよびリレーログファイルのファイルパスワードを暗号化します。コピーされたファイルは、ソースサーバーキーを使用して引き続き読み取られます。

17.3.2.3 バイナリログマスターキーのローテーション

バイナリログの暗号化が有効になっている場合は、`ALTER INSTANCE ROTATE BINLOG MASTER KEY` を発行することで、サーバーの実行中いつでもバイナリログマスターキーをローテーションできます。このステートメントを使用してバイナリログマスターキーを手動でローテーションすると、新規および後続のファイルのパスワードは新しいバイナリログマスターキーを使用して暗号化され、既存の暗号化バイナリログファイルおよびリレーログファイルのファイルパスワードも新しいバイナリログマスターキーを使用して再暗号化されるため、暗号化は完全に更新されます。バイナリログマスターキーを定期的にローテーションして、組織のセキュリティポリシーに準拠できます。また、現在のバイナリログマスターキーまたは以前のバイナリログマスターキーのいずれかが危険にさらされている疑いがある場合も同様です。

バイナリログマスターキーを手動でローテーションすると、MySQL Server は次のアクションを順番に実行します:

1. 新しいバイナリログ暗号化鍵は、次に使用可能なシーケンス番号で生成され、鍵リングに格納され、新しいバイナリログマスターキーとして使用されます。
2. バイナリログおよびリレーログファイルは、すべてのチャンネルでローテーションされます。
3. 新しいバイナリログマスターキーは、新しいバイナリログファイル、リレーログファイル、および鍵が再度変更されるまで後続のファイルのファイルパスワードを暗号化するために使用されます。
4. サーバー上の既存の暗号化バイナリログファイルおよびリレーログファイルのファイルパスワードは、新しいバイナリログマスターキーを使用して、最新のファイルから順に再暗号化されます。暗号化されていないファイルはスキップされます。
5. 再暗号化プロセス後にファイルに使用されなくなったバイナリログ暗号化キーは、キーリングから削除されます。

`ALTER INSTANCE ROTATE BINLOG MASTER KEY` を発行するには `BINLOG_ENCRYPTION_ADMIN` 権限が必要であり、`binlog_encryption` システム変数が `OFF` に設定されている場合、このステートメントは使用できません。

バイナリログマスターキーローテーション処理の最終ステップとして、保持されているバイナリログファイルまたはリレーログファイルに適用されなくなったすべてのバイナリログ暗号化鍵が鍵リングからクリーンアップされます。保持されているバイナリログファイルまたはリレーログファイルを再暗号化のために初期化できない場合、将来ファイルを回復できるように、関連するバイナリログ暗号化鍵は削除されません。たとえば、バイナリログインデックスファイルにリストされているファイルが現在読み取れない場合や、チャンネルの初期化に失敗した場合などです。たとえば、MySQL Enterprise Backup を使用して作成されたバックアップを使用して新しいレプリカを設定するためにサーバー UUID が変更された場合、新しいサーバーで `ALTER INSTANCE ROTATE BINLOG MASTER KEY` を発行しても、元のサーバー UUID を含む以前のバイナリログ暗号化キーは削除されません。

バイナリログマスターキーローテーション処理の最初の 4 つの手順のいずれかを正しく完了できない場合は、状況とバイナリログファイルおよびリレーログファイルの暗号化ステータスの結果を説明するエラーメッセージが発行されます。以前に暗号化されたファイルは常に暗号化された状態のままですが、そのファイルのパスワードは古いバイナリログマスターキーを使用して暗号化される場合があります。これらのエラーが表示された場合は、まず `ALTER INSTANCE ROTATE BINLOG MASTER KEY` を再発行してプロセスを再試行してください。次に、個々のファイルのステータスを調べて、特に現在または以前のバイナリログマスターキーのいずれかが危険にさらされている可能性があると思われる場合に、プロセスをブロックしている内容を確認します。

バイナリログマスターキーローテーション処理の最終ステップを正しく完了できない場合は、状況を説明する警告メッセージが発行されます。警告メッセージは、バイナリログマスターキーをローテーションするための鍵リング内の補助鍵をプロセスがクリーンアップできなかったか、または未使用のバイナリログ暗号化鍵をクリーンアップできなかったかを示します。キーが補助キーであるか使用されなくなったため、メッセージを無視するか、`ALTER INSTANCE ROTATE BINLOG MASTER KEY` を再発行してプロセスを再試行できます。

サーバーが停止し、バイナリログマスターキーローテーション処理中にバイナリログ暗号化が ON に設定されたまま再起動された場合、再起動後の新しいバイナリログファイルとリレーログファイルは新しいバイナリログマスターキーを使用して暗号化されます。ただし、既存のファイルの再暗号化は続行されないため、サーバーが停止する前に再暗号化されなかったファイルは、以前のバイナリログマスターキーを使用して暗号化されたままになります。再暗号化を完了し、未使用のバイナリログ暗号化キーをクリーンアップするには、再起動後に `ALTER INSTANCE ROTATE BINLOG MASTER KEY` を再発行します。

`ALTER INSTANCE ROTATE BINLOG MASTER KEY` アクションはバイナリログに書き込まれず、レプリカでは実行されません。したがって、バイナリログマスターキーローテーションは、MySQL バージョンの混在を含むレプリケーション環境で実行できます。適用可能なすべてのソースサーバーおよびレプリカサーバーでバイナリログマスターキーの定期ローテーションをスケジュールするには、各サーバーで MySQL イベントスケジューラを有効にし、`CREATE EVENT` ステートメントを使用して `ALTER INSTANCE ROTATE BINLOG MASTER KEY` ステートメントを発行します。現在または以前のバイナリログマスターキーのいずれかが危険にさらされている可能性があるためにバイナリログマスターキーをローテーションした場合は、該当するすべてのソースおよびレプリカサーバーでステートメントを発行します。個々のサーバーでステートメントを発行すると、遅延しているレプリカ、複数のレプリケーショントポロジに属しているレプリカ、またはレプリケーショントポロジで現在アクティブではないがバイナリログファイルとリレーログファイルがあるレプリカの場合でも、即座にコンプライアンスを検証できます。

`binlog_rotate_encryption_master_key_at_startup` システム変数は、サーバーの再起動時にバイナリログマスターキーを自動的にローテーションするかどうかを制御します。このシステム変数が ON に設定されている場合、新しいバイナリログ暗号化鍵が生成され、サーバーが再起動されるたびに新しいバイナリログマスターキーとして使用されます。OFF (デフォルト) に設定されている場合は、再起動後に既存のバイナリログマスターキーが再度使用されます。バイナリログマスターキーが起動時にローテーションされると、新しいバイナリログおよびリレーログファイルのファイルパスワードは新しいキーを使用して暗号化されます。既存の暗号化バイナリログファイルおよびリレーログファイルのファイルパスワードは再暗号化されないため、鍵リングで引き続き使用できる古い鍵を使用して暗号化されたままになります。

17.3.3 レプリケーション権限チェック

デフォルトでは、別のサーバーによってすでに受け入れられたトランザクションがレプリカまたはグループメンバーに適用されている場合、MySQL レプリケーション (グループレプリケーションを含む) は権限チェックを実行しません。MySQL 8.0.18 から、通常はチャンネルでレプリケートされるトランザクションを適用する適切な権限を持つユーザーアカウントを作成し、`CHANGE REPLICATION SOURCE TO` ステートメント (MySQL 8.0.23 の場合) または `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 の場合) を使用して、これをレプリケーションアプライアンスの `PRIVILEGE_CHECKS_USER` アカウントとして指定できます。次に、MySQL は各トランザクションをユーザーアカウント権限と照合してチェックし、そのチャンネルに対する操作が認可されていることを確認します。管理者は、このアカウントを安全に使用して、チャンネルのレプリケーションエラーからのリカバリなど、`mysqlbinlog` 出力からトランザクションを適用または再適用することもできます。

`PRIVILEGE_CHECKS_USER` アカウントを使用すると、権限のある操作または不要な操作の不正または誤った使用からレプリケーションチャンネルを保護できます。`PRIVILEGE_CHECKS_USER` アカウントは、次のような状況で追加のセキュリティレイヤーを提供します:

- あなたは、組織のネットワーク上のサーバーインスタンスと、クラウドサービスプロバイダによって提供されるインスタンスなど、別のネットワーク上のサーバーインスタンスの間でレプリケートしています。
- すべてのデプロイメントに対して 1 つの管理者アカウント権限を付与せずに、複数のオンプレミスデプロイメントまたはオフサイトデプロイメントを別々のユニットとして管理する場合。
- 管理者がサーバーインスタンスに対する広範囲の権限を持つのではなく、レプリケーションチャンネルおよびレプリケーションチャンネルがレプリケートするデータベースに直接関連する操作のみを実行できるようにする管理者アカウントが必要です。

チャンネルの `PRIVILEGE_CHECKS_USER` アカウントを指定するときに、次のいずれかまたは両方のオプションを `CHANGE REPLICATION SOURCE TO` | `CHANGE MASTER TO` ステートメントに追加することで、権限チェックが適用されるレプリケーションチャンネルのセキュリティを強化できます:

- `REQUIRE_ROW_FORMAT` オプション (MySQL 8.0.19 から使用可能) は、レプリケーションチャンネルが行ベースのレプリケーションイベントのみを受け入れるようにします。`REQUIRE_ROW_FORMAT` が設定されている場合、ソースサーバーで行ベースのバイナリロギング (`binlog_format=ROW`) を使用する必要があります。MySQL 8.0.18 では、`REQUIRE_ROW_FORMAT` は使用できませんが、セキュアなレプリケーションチャンネルに

行ベースのバイナリロギングを使用することを強くお勧めします。ステートメントベースのバイナリロギングでは、`PRIVILEGE_CHECKS_USER` アカウントがトランザクションを正常に実行するために、一部の管理者レベルの権限が必要になる場合があります。

- `REQUIRE_TABLE_PRIMARY_KEY_CHECK` オプション (MySQL 8.0.20 から使用可能) は、レプリケーションチャンネルが主キーチェックに独自のポリシーを使用するようにします。`ON` を設定することは、主キーが常に必要であり、`OFF` を設定することは、主キーが不要であることを意味します。デフォルト設定の `STREAM` では、各トランザクションのソースからレプリケートされた値を使用して、`sql_require_primary_key` システム変数のセッション値が設定されます。`PRIVILEGE_CHECKS_USER` が設定されている場合、`REQUIRE_TABLE_PRIMARY_KEY_CHECK` を `ON` または `OFF` に設定することは、`sql_require_primary_key` の値を変更するために必要な、制限されたセッション変数を設定するためのセッション管理レベルの権限をユーザーアカウントが必要としないことを意味します。また、様々なソースのレプリケーションチャンネル間の動作を正規化します。

レプリケーションアプライヤスレッドの `PRIVILEGE_CHECKS_USER` としてユーザーアカウントを表示できるようにし、`mysqlbinlog` で使用される内部使用 `BINLOG` ステートメントを実行するには、`REPLICATION_APPLIER` 権限を付与します。`PRIVILEGE_CHECKS_USER` アカウントのユーザー名とホスト名は、[セクション6.2.4「アカウント名の指定」](#) で説明されている構文に従う必要があり、ユーザーは匿名ユーザー (ユーザー名が空白) または `CURRENT_USER` であってはなりません。新しいアカウントを作成するには、`CREATE USER` を使用します。このアカウントに `REPLICATION_APPLIER` 権限を付与するには、`GRANT` ステートメントを使用します。たとえば、`example.com` ドメイン内の任意のホストから管理者が手動で使用でき、暗号化された接続が必要なユーザーアカウント `priv_repl` を作成するには、次のステートメントを発行します:

```
mysql> SET sql_log_bin = 0;
mysql> CREATE USER 'priv_repl'@'%example.com' IDENTIFIED BY 'password' REQUIRE SSL;
mysql> GRANT REPLICATION_APPLIER ON *.* TO 'priv_repl'@'%example.com';
mysql> SET sql_log_bin = 1;
```

`SET sql_log_bin` ステートメントは、アカウント管理ステートメントがバイナリログに追加されず、レプリケーションチャンネルに送信されないように使用されます ([セクション13.4.1.3「SET sql_log_bin ステートメント」](#) を参照)。

重要

`caching_sha2_password` 認証プラグインは、MySQL 8.0 から作成された新規ユーザーのデフォルトです (詳細は、[セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#) を参照)。このプラグインで認証するユーザーアカウントを使用してサーバーに接続するには、[セクション17.3.1「暗号化接続を使用するためのレプリケーションの設定」](#) の説明に従って暗号化された接続を設定するか、RSA キーペアを使用したパスワード交換をサポートするように暗号化されていない接続を有効にする必要があります。

ユーザーアカウントを設定した後、`GRANT` ステートメントを使用して追加の権限を付与し、サーバーに保持されている特定のテーブルの更新など、アプライヤスレッドが実行する予定のデータベース変更をユーザーアカウントが実行できるようにします。レプリケーションチャンネルでこれらのトランザクションのいずれかを手動で実行する必要がある場合、管理者はこれらの同じ権限を使用できます。適切な権限を付与しなかった予期しない操作が試行された場合、操作は許可されず、レプリケーションアプライヤスレッドはエラーで停止します。[セクション17.3.3.1「レプリケーション PRIVILEGE_CHECKS_USER アカウントの権限」](#) では、アカウントに必要な追加の権限について説明します。たとえば、`priv_repl` ユーザーアカウントに `db1` の `cust` テーブルに行を追加する `INSERT` 権限を付与するには、次のステートメントを発行します:

```
mysql> GRANT INSERT ON db1.cust TO 'priv_repl'@'%example.com';
```

レプリケーションチャンネルの `PRIVILEGE_CHECKS_USER` アカウントは、`CHANGE REPLICATION SOURCE TO` ステートメント (MySQL 8.0.23 の場合) または `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 の場合) を使用して割り当てます。`PRIVILEGE_CHECKS_USER` が設定されている場合は、行ベースのバイナリロギングを使用することを強くお勧めします。また、MySQL 8.0.19 から、このステートメントを使用して `REQUIRE_ROW_FORMAT` を設定してこれを強制できます。レプリケーションが実行中の場合は、`CHANGE MASTER TO` ステートメントの前に `STOP REPLICAS | SLAVE` を発行し、その後 `START REPLICAS | SLAVE` を発行します。たとえば、実行中のレプリカでチャンネル `channel_1` に対する権限チェックを開始するには、次のステートメントを発行します:

```
mysql> STOP SLAVE FOR CHANNEL 'channel_1';
mysql> CHANGE MASTER TO
  PRIVILEGE_CHECKS_USER = 'priv_repl'@'%example.com',
  REQUIRE_ROW_FORMAT = 1 FOR CHANNEL 'channel_1';
```



```
mysql> START SLAVE FOR CHANNEL 'channel_1';

Or from MySQL 8.0.22 / 8.0.23:
mysql> STOP REPLICA FOR CHANNEL 'channel_1';
mysql> CHANGE REPLICATION SOURCE TO
  PRIVILEGE_CHECKS_USER = 'priv_repl'@'%example.com',
  REQUIRE_ROW_FORMAT = 1 FOR CHANNEL 'channel_1';
mysql> START REPLICA FOR CHANNEL 'channel_1';
```

レプリケーションチャンネルを再起動すると、その時点から権限チェックが適用されます。チャンネルを指定せず、他のチャンネルが存在しない場合は、ステートメントがデフォルトチャンネルに適用されます。チャンネルの `PRIVILEGE_CHECKS_USER` アカウントのユーザー名とホスト名は、パフォーマンススキーマの `replication_applier_configuration` テーブルに表示されます。ここでは、パフォーマンススキーマが適切にエスケープされているため、個々のトランザクションを実行するために SQL ステートメントに直接コピーできます。

レプリケーションチャンネルに `REQUIRE_ROW_FORMAT` が設定されている場合、レプリケーションアプライアンスは一時テーブルを作成または削除しないため、`pseudo_thread_id` セッションシステム変数は設定されません。`LOAD DATA INFILE` 命令は実行されないため、(Format_description_log_event としてログに記録された) データロードに関連付けられた一時ファイルへのファイル操作のアクセスまたは削除は試行されません。`INTVAR`、`RAND` および `USER_VAR` イベントは実行されません。これらは、ステートメントベースレプリケーションのクライアント接続状態を再現するために使用されます。(例外は、実行される DDL クエリーに関連付けられた `USER_VAR` イベントです。) DML トランザクション内に記録されるステートメントは実行されません。トランザクションのキューイングまたは適用の試行中にレプリケーションアプライアンスがこれらのタイプのイベントのいずれかを検出した場合、イベントは適用されず、レプリケーションはエラーで停止します。

`PRIVILEGE_CHECKS_USER` アカウントを設定するかどうかに関係なく、レプリケーションチャンネルに `REQUIRE_ROW_FORMAT` を設定できます。このオプションを設定すると、権限チェックなしでもレプリケーションチャンネルのセキュリティが向上します。`mysqlbinlog` の使用時に `--require-row-format` オプションを指定して、`mysqlbinlog` 出力で行ベースのレプリケーションイベントを強制することもできます。

セキュリティコンテキスト。デフォルトでは、レプリケーションアプライアスレッドが `PRIVILEGE_CHECKS_USER` として指定されたユーザーアカウントで起動されると、セキュリティコンテキストはデフォルトロールを使用して作成されるか、`activate_all_roles_on_login` が `ON` に設定されている場合はすべてのロールを使用して作成されます。

次の例に示すように、ロールを使用して、`PRIVILEGE_CHECKS_USER` アカウントとして使用されるアカウントに一般権限セットを提供できます。ここでは、前述の例のように、`db1.cust` テーブルに対する `INSERT` 権限をユーザーアカウントに直接付与するかわりに、この権限が `REPLICATION_APPLIER` 権限とともに `priv_repl_role` ロールに付与されます。このロールは、権限セットを2つのユーザーアカウントに付与するために使用され、その両方を `PRIVILEGE_CHECKS_USER` アカウントとして使用できるようになります:

```
mysql> SET sql_log_bin = 0;
mysql> CREATE USER 'priv_repa'@'%example.com'
  IDENTIFIED BY 'password'
  REQUIRE SSL;
mysql> CREATE USER 'priv_repb'@'%example.com'
  IDENTIFIED BY 'password'
  REQUIRE SSL;
mysql> CREATE ROLE 'priv_repl_role';
mysql> GRANT REPLICATION_APPLIER TO 'priv_repl_role';
mysql> GRANT INSERT ON db1.cust TO 'priv_repl_role';
mysql> GRANT 'priv_repl_role' TO
  'priv_repa'@'%example.com',
  'priv_repb'@'%example.com';
mysql> SET DEFAULT ROLE 'priv_repl_role' TO
  'priv_repa'@'%example.com',
  'priv_repb'@'%example.com';
mysql> SET sql_log_bin = 1;
```

レプリケーションアプライアスレッドがセキュリティコンテキストを作成する場合、`PRIVILEGE_CHECKS_USER` アカウントの権限はチェックされますが、パスワード検証は実行されず、アカウントがロックされているかどうかのチェックなど、アカウント管理に関連するチェックは実行されません。作成されたセキュリティコンテキストは、レプリケーションアプライアスレッドの存続期間中は変更されません。

制限。MySQL 8.0.18 のみ、`RESET REPLICA | SLAVE` ステートメントを発行した直後にレプリカ `mysqld` が再起動された場合 (予期しないサーバー終了または意図的な再起動のため)、`mysql.slave_relay_log_info` テーブルに保持さ

れている `PRIVILEGE_CHECKS_USER` アカウント設定は失われ、再指定する必要があります。そのリリースで権限チェックを使用する場合は、必ず再起動後に権限チェックが設定されていることを確認し、必要に応じて再指定します。この状況では、MySQL 8.0.19 から `PRIVILEGE_CHECKS_USER` アカウント設定が保持されるため、テーブルから取得され、チャンネルに再適用されます。

17.3.3.1 レプリケーション `PRIVILEGE_CHECKS_USER` アカウントの権限

レプリケーションチャンネルの `PRIVILEGE_CHECKS_USER` アカウントとして `CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO` ステートメントを使用して指定されたユーザーアカウントには、`REPLICATION_APPLIER` 権限が必要です。権限がない場合、レプリケーションアプライアンススレッドは起動しません。 [セクション 17.3.3 「レプリケーション権限チェック」](#) で説明されているように、アカウントには、レプリケーションチャンネルで予想されるすべてのトランザクションを適用するのに十分な追加の権限が必要です。これらの権限は、関連するトランザクションが実行された場合にのみチェックされます。

`PRIVILEGE_CHECKS_USER` アカウントを使用して保護されているレプリケーションチャンネルには、行ベースのバイナリロギング (`binlog_format=ROW`) を使用することを強くお勧めします。ステートメントベースのバイナリロギングでは、`PRIVILEGE_CHECKS_USER` アカウントがトランザクションを正常に実行するために、一部の管理者レベルの権限が必要になる場合があります。MySQL 8.0.19 から、`REQUIRE_ROW_FORMAT` 設定を保護されたチャンネルに適用できます。これにより、チャンネルはこれらの権限を必要とするイベントを実行できなくなります。

`REPLICATION_APPLIER` 権限によって、`PRIVILEGE_CHECKS_USER` アカウントはレプリケーションスレッドが実行する必要がある次の操作を明示的または暗黙的に実行できます:

- システム変数 `gtid_next`, `original_commit_timestamp`, `original_server_version`, `immediate_server_version` および `pseudo_slave_mode` の値を設定して、トランザクションの実行時に適切なメタデータおよび動作を適用します。
- 内部使用の `BINLOG` ステートメントを実行して `mysqlbinlog` 出力を適用します (アカウントにそれらのステートメントのテーブルおよび操作に対する権限もある場合)。
- レプリケーションメタデータを更新するためのシステムテーブル `mysql.gtid_executed`, `mysql.slave_relay_log_info`, `mysql.slave_worker_info` および `mysql.slave_master_info` の更新。(イベントが他の目的でこれらのテーブルに明示的にアクセスする場合は、テーブルに対する適切な権限を付与する必要があります。)
- バイナリログ `Table_map_log_event` を適用します。これは、テーブルメタデータを提供しますが、データベースの変更は行いません。

`CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO` ステートメントの `REQUIRE_TABLE_PRIMARY_KEY_CHECK` オプションがデフォルトの `STREAM` に設定されている場合、`PRIVILEGE_CHECKS_USER` アカウントには、ソースからレプリケートされた設定と一致するようにセッション期間中に `sql_require_primary_key` システム変数の値を変更できるように、制限付きセッション変数を設定するのに十分な権限が必要です。`SESSION_VARIABLES_ADMIN` 権限により、アカウントにこの機能が付与されます。この権限により、アカウントは `--disable-log-bin` オプションを使用して作成された `mysqlbinlog` 出力を適用することもできます。`REQUIRE_TABLE_PRIMARY_KEY_CHECK` を `ON` または `OFF` のいずれかに設定した場合、レプリカはレプリケーション操作で `sql_require_primary_key` システム変数にその値を常に使用するため、これらのセッション管理レベルの権限は必要ありません。

テーブルの暗号化が使用されており、`table_encryption_privilege_check` システム変数が `ON` に設定されており、イベントに関係するテーブルスペースの暗号化設定が (`default_table_encryption` システム変数で指定された) サーバーのデフォルトの暗号化設定の適用と異なる場合、`PRIVILEGE_CHECKS_USER` アカウントでは、デフォルトの暗号化設定をオーバーライドするために `TABLE_ENCRYPTION_ADMIN` 権限が必要です。この権限を付与しないことを強くお勧めします。かわりに、レプリカのデフォルトの暗号化設定がレプリケートするテーブルスペースの暗号化ステータスと一致し、レプリケーショングループメンバーのデフォルトの暗号化設定が同じであることを確認して、権限が不要になるようにします。

特定のレプリケートされたトランザクションをリレーログから実行するか、必要に応じて `mysqlbinlog` 出力から実行するには、`PRIVILEGE_CHECKS_USER` アカウントに次の権限が必要です:

- (`Write_rows_log_event` として記録される) 行形式で記録される行挿入の場合、関連するテーブルに対する `INSERT` 権限。
- (`Update_rows_log_event` として記録される) 行形式で記録された行更新の場合、関連するテーブルに対する `UPDATE` 権限。

- (Delete_rows_log_event として記録される) 行形式で記録された行の削除の場合、関連するテーブルに対する DELETE 権限。

ステートメントベースのバイナリロギングが使用されている (PRIVILEGE_CHECKS_USER アカウントでは推奨されません) 場合、BEGIN、COMMIT、DML logged in statement format (Query_log_event として記録されます) などのトランザクション制御ステートメントでは、PRIVILEGE_CHECKS_USER アカウントにはイベントに含まれるステートメントを実行する権限が必要です。

レプリケーションチャンネルで LOAD DATA 操作を実行する必要がある場合は、行ベースのバイナリロギング (binlog_format=ROW) を使用します。このロギング形式では、イベントの実行に FILE 権限は必要ないため、PRIVILEGE_CHECKS_USER アカウントにこの権限を付与しないでください。PRIVILEGE_CHECKS_USER アカウントを使用して保護されているレプリケーションチャンネルでは、行ベースのバイナリロギングを使用することを強くお勧めします。チャンネルに REQUIRE_ROW_FORMAT が設定されている場合は、行ベースのバイナリロギングが必要です。LOAD DATA イベントによって作成された一時ファイルを削除する Format_description_log_event は、権限チェックなしで処理されます。詳細は、セクション 17.5.1.19 「レプリケーションと LOAD DATA」を参照してください。

レプリケーション SQL スレッドの起動時に実行される 1 つ以上の SQL ステートメントを指定するように init_slave システム変数が設定されている場合、PRIVILEGE_CHECKS_USER アカウントにはこれらのステートメントの実行に必要な権限が必要です。

CREATE USER、CREATE ROLE、DROP ROLE や GRANT OPTION などの ACL 権限を PRIVILEGE_CHECKS_USER アカウントに付与しないこと、およびアカウントによる mysql.user テーブルの更新を許可しないことをお勧めします。これらの権限では、アカウントを使用してサーバー上のユーザーアカウントを作成または変更できます。ソースサーバーで発行された ACL ステートメントが実行のために保護されたチャンネルにレプリケートされないようにするには (これらの権限がない場合は失敗します)、すべての ACL ステートメントの前に SET sql_log_bin = 0 を発行し、その後に SET sql_log_bin = 1 を発行して、ソースバイナリログからステートメントを除外します。または、すべての ACL ステートメントを実行する前に専用の現行データベースを設定し、レプリケーションフィルタ (--binlog-ignore-db) を使用してレプリカでこのデータベースを除外することもできます。

17.3.3.2 グループレプリケーションチャンネルの権限チェック

MySQL 8.0.19 から、非同期レプリケーションおよび準同期レプリケーションを保護するだけでなく、PRIVILEGE_CHECKS_USER アカウントを使用して、Group Replication で使用される 2 つのレプリケーションアプライヤスレッドを保護することもできます。各グループメンバーの group_replication_applier スレッドはグループトランザクションの適用に使用され、各グループメンバーの group_replication_recovery スレッドは、メンバーがグループに参加または再参加したときの分散リカバリの一環として、バイナリログからの状態転送に使用されます。

これらのスレッドのいずれかを保護するには、Group Replication を停止し、PRIVILEGE_CHECKS_USER オプションとともに CHANGE REPLICATION SOURCE TO ステートメント (MySQL 8.0.23) または CHANGE MASTER TO ステートメント (MySQL 8.0.23 より前) を発行し、チャンネル名として group_replication_applier または group_replication_recovery を指定します。。例:

```
mysql> STOP GROUP_REPLICATION;
mysql> CHANGE MASTER TO PRIVILEGE_CHECKS_USER = 'gr_repl'@'%example.com'
FOR CHANNEL 'group_replication_recovery';
mysql> START GROUP_REPLICATION;

Or from MySQL 8.0.23:
mysql> STOP GROUP_REPLICATION;
mysql> CHANGE REPLICATION SOURCE TO PRIVILEGE_CHECKS_USER = 'gr_repl'@'%example.com'
FOR CHANNEL 'group_replication_recovery';
mysql> START GROUP_REPLICATION;
```

グループレプリケーションチャンネルの場合、REQUIRE_ROW_FORMAT 設定はチャンネルの作成時に自動的に有効化され、無効化できないため、指定する必要はありません。

重要

MySQL 8.0.19 では、Group Replication の実行中に PRIVILEGE_CHECKS_USER オプションを指定して CHANGE REPLICATION SOURCE TO|CHANGE MASTER TO ステートメントを発行しないでください。このアクションにより、チャンネルのリレーログファイルがパー

じされ、受信してリレーログにキューに入れられたがまだ適用されていないトランザクションが失われる可能性があります。

グループレプリケーションでは、グループによってレプリケートされるすべてのテーブルに、定義済の主キーまたは同等の主キー (同等のものが NULL 以外の一意キー) がある必要があります。グループレプリケーションには、`sql_require_primary_key` システム変数によって実行されるチェックを使用するかわりに、主キーまたは主キーに相当する独自の組込みチェックセットがあります。グループレプリケーションチャンネルの場合、`CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO` ステートメントの `REQUIRE_TABLE_PRIMARY_KEY_CHECK` オプションを `ON` に設定できます。ただし、グループレプリケーションの組込みチェックで許可されている一部のトランザクションは、`sql_require_primary_key = ON` または `REQUIRE_TABLE_PRIMARY_KEY_CHECK = ON` の設定時に実行されるチェックでは許可されない場合があることに注意してください。このため、MySQL 8.0.20 の新規およびアップグレードされたグループレプリケーションチャンネル (オプションが導入された場合) では、`REQUIRE_TABLE_PRIMARY_KEY_CHECK` が `ON` ではなく `STREAM` のデフォルトに設定されています。

Group Replication (セクション 18.4.3.2 「分散リカバリのためのクローニング」を参照) の分散リカバリにリモートクローニング操作が使用されている場合、MySQL 8.0.19 から、ドナーの `PRIVILEGE_CHECKS_USER` アカウントおよび関連設定が結合メンバーにクローニングされます。参加メンバーがブート時にグループレプリケーションを開始するように設定されている場合、適切なレプリケーションチャンネルの権限チェックにアカウントが自動的に使用されます。

MySQL 8.0.18 では、多くの制限により、グループレプリケーションチャンネルで `PRIVILEGE_CHECKS_USER` アカウントを使用しないことをお勧めします。

17.3.3.3 失敗したレプリケーション権限チェックからのリカバリ

`PRIVILEGE_CHECKS_USER` アカウントに対する権限チェックが失敗した場合、トランザクションは実行されず、チャンネルのレプリケーションは停止します。エラーの詳細と最後に適用されたトランザクションは、パフォーマンススキーマ `replication_applier_status_by_worker` テーブルに記録されます。エラーから回復するには、次の手順に従います:

1. エラーの原因となったレプリケートイベントを特定し、イベントが予想されるかどうか、および信頼できるソースからのものかどうかを確認します。 `mysqlbinlog` を使用して、エラー発生時にログに記録されたイベントを取得して表示できます。これを行う手順は、セクション 7.5 「Point-in-Time (増分) リカバリ」を参照してください。
2. レプリケートされたイベントが予期されていないか、既知の信頼できるソースからのものでない場合は、原因を調査します。イベントが発生した理由がわかり、セキュリティ上の考慮事項がない場合は、次の説明に従ってエラーの修正に進みます。
3. `PRIVILEGE_CHECKS_USER` アカウントがトランザクションの実行を許可されている必要があるが、正しく構成されていない場合は、アカウントに不足している権限を付与し、チャンネルのレプリケーションを再開します。
4. トランザクションを実行する必要があり、信頼できることを確認したが、`PRIVILEGE_CHECKS_USER` アカウントにこの権限を通常どおりに付与しない場合は、`PRIVILEGE_CHECKS_USER` アカウントに必要な権限を一時的に付与できます。レプリケートされたイベントが適用されたら、アカウントから権限を削除し、必要なステップを実行して、回避できる場合にイベントが繰り返されないようにします。
5. トランザクションがソースでのみ実行され、レプリカでは実行されない管理アクションである場合、または単一のレプリケーショングループメンバーでのみ実行される必要がある場合は、レプリケーションを停止したサーバーでトランザクションをスキップし、`START REPLICA | SLAVE` を発行してチャンネルでレプリケーションを再起動します。今後の状況を回避するために、その前に `SET sql_log_bin = 0` を使用し、その後に `SET sql_log_bin = 1` を使用してこのような管理ステートメントを発行して、ソースに記録されないようにすることができます。
6. トランザクションがソースまたはレプリカで行われてはならない DDL ステートメントまたは DML ステートメントである場合は、レプリケーションを停止したサーバー上のトランザクションをスキップし、トランザクションを元のサーバー上で手動で取り消してから、`START REPLICA | SLAVE` を発行してレプリケーションを再起動します。

GTID が使用中の場合にトランザクションをスキップするには、失敗したトランザクションの GTID を持つ空のトランザクションをコミットします。次に例を示します:

```
SET GTID_NEXT='aaa-bbb-ccc-ddd:N';
BEGIN;
```



```
COMMIT;  
SET GTID_NEXT='AUTOMATIC';
```

GTID が使用されていない場合は、[SET GLOBAL sql_slave_skip_counter](#) ステートメントを発行してイベントをスキップします。この代替方法を使用する手順およびトランザクションのスキップの詳細は、[セクション17.1.7.3「トランザクションのスキップ」](#)を参照してください。

17.4 レプリケーションソリューション

レプリケーションは、多くの異なる環境でさまざまな目的に使用できます。このセクションでは、特定のソリューションタイプにレプリケーションを使用する場合の一般的な注意点とアドバイスを提供します。

セットアップに関する注意点、バックアップ手順、バックアップするファイルなど、バックアップ環境でレプリケーションを使用する場合の情報については、[セクション17.4.1「バックアップ用にレプリケーションを使用する」](#)を参照してください。

ソースとレプリカで異なるストレージエンジンを使用する際のアドバイスとヒントについては、[セクション17.4.4「異なるソースおよびレプリカのストレージエンジンでのレプリケーションの使用」](#)を参照してください。

スケールアウトソリューションとしてレプリケーションを使用するには、ソリューションを使用するアプリケーションのロジックとオペレーションで若干の変更が必要になります。[セクション17.4.5「スケールアウトのためにレプリケーションを使用する」](#)を参照してください。

パフォーマンスまたはデータ分散の理由から、異なるデータベースを異なるレプリカにレプリケートできます。[セクション17.4.6「異なるレプリカへの異なるデータベースのレプリケート」](#)を参照してください

レプリカの数が増えると、ソースの負荷が増加し、パフォーマンスが低下する可能性があります(各レプリカにバイナリログをレプリケートする必要があるため)。単一のセカンダリサーバーをソースとして使用するなど、レプリケーションパフォーマンスの向上に関するヒントは、[セクション17.4.7「レプリケーションパフォーマンスを改善する」](#)を参照してください。

緊急フェイルオーバーソリューションの一部としてのソースの切替えまたはレプリカのソースへの変換に関するガイドは、[セクション17.4.8「フェイルオーバー中のソースの切替え」](#)を参照してください。

レプリケーショントポロジ内のサーバーに固有のセキュリティ対策の詳細は、[セクション17.3「レプリケーションのセキュリティ」](#)を参照してください。

17.4.1 バックアップ用にレプリケーションを使用する

レプリケーションをバックアップソリューションとして使用するには、ソースからレプリカにデータをレプリケートしてから、レプリカをバックアップします。レプリカは、ソースの実行操作に影響を与えずに一時停止および停止できるため、ソースの停止を必要とする「ライブ」データの有効なスナップショットを生成できます。

データベースのバックアップ方法は、データベースのサイズと、障害発生時にレプリカを再構築できるようにデータのみをバックアップするか、データとレプリカの状態をバックアップするかによって異なります。つまり、2つの選択肢があります。

- ソース上のデータのバックアップを可能にするソリューションとしてレプリケーションを使用しており、データベースのサイズが大きすぎない場合は、[mysqldump](#) ツールが適している可能性があります。[セクション17.4.1.1「mysqldumpを使用したレプリカのバックアップ」](#)を参照してください。
- より大きなデータベースの場合は、[mysqldump](#) は実用的または効率的でなく、代わりにローデータファイルをバックアップできます。RAW データファイルオプションを使用すると、レプリカに障害が発生した場合にレプリカを再作成できるバイナリログおよびリレーログをバックアップすることもできます。詳細については、[セクション17.4.1.2「レプリカからのRAW データのバックアップ」](#)を参照してください。

ソースサーバーまたはレプリカサーバーに使用できる別のバックアップ計画は、サーバーを読み取り専用状態にすることです。バックアップは読み取り専用サーバーに対して実行され、これが通常の読み取り/書き込み操作ステータスに戻されます。[セクション17.4.1.3「ソースまたはレプリカを読み取り専用にするによるバックアップ」](#)を参照してください。

17.4.1.1 mysqldump を使用したレプリカのバックアップ

`mysqldump` を使用してデータベースのコピーを作成することで、MySQL Server の別のインスタンスに情報をインポートできる形式でデータベース内のすべてのデータを取得できます ([セクション4.5.4 「mysqldump — データベースバックアッププログラム」](#)を参照してください)。情報の形式が SQL ステートメントであるため、緊急時にデータにアクセスする必要がある場合にファイルを実行中サーバーに配布して適用することも簡単にできます。ただし、データセットのサイズが非常に大きい場合は、`mysqldump` が実用的でない場合があります。

ヒント

複数のスレッド、ファイル圧縮、進捗情報の表示、および Oracle Cloud Infrastructure Object Storage ストリーミングや MySQL データベースサービス 互換性チェックおよび変更などのクラウド機能で並列ダンプを提供する [MySQL Shell dump utilities](#) の使用を検討してください。ダンプは、[MySQL Shell load dump utilities](#) を使用して MySQL Server インスタンスまたは MySQL データベースサービス DB システムに簡単にインポートできます。MySQL Shell のインストール手順は、[here](#) にあります。

`mysqldump` を使用する場合は、ダンププロセスを開始する前にレプリカのレプリケーションを停止して、ダンプに一貫性のあるデータセットが含まれていることを確認する必要があります:

1. レプリカによるリクエストの処理を停止します。 `mysqladmin` を使用して、レプリカ上のレプリケーションを完全に停止できます:

```
shell> mysqladmin stop-slave
```

または、レプリケーション SQL スレッドのみを停止してイベントの実行を一時停止することもできます:

```
shell> mysql -e 'STOP SLAVE SQL_THREAD;'
Or from MySQL 8.0.22:
shell> mysql -e 'STOP REPLICHA SQL_THREAD;'
```

これにより、レプリカは引き続きソースバイナリログからデータ変更イベントを受信し、レプリケーション I/O スレッドを使用してリレーログに格納できますが、レプリカはこれらのイベントを実行してデータを変更できなくなります。ビジーなレプリケーション環境では、レプリケーション SQL スレッドを再起動するときに、バックアップ中にレプリケーション I/O スレッドの実行を許可するとキャッチアッププロセスが高速化される場合があります。

2. `mysqldump` を実行してデータベースをダンプします。すべてのデータベースをダンプしたり、ダンプするデータベースを選択したりできます。たとえば、すべてのデータベースをダンプするには:

```
shell> mysqldump --all-databases > fulldb.dump
```

3. ダンプが完了したら、レプリケーションを再度開始します:

```
shell> mysqladmin start-slave
```

前の例では、ログイン資格証明(ユーザー名、パスワード)をコマンドに追加したり、毎日自動的に実行できるプロセスをスクリプトにバンドルしたりすることをお勧めします。

このアプローチを使用する場合は、必ずレプリケーションプロセスを監視して、バックアップの実行に要した時間が、ソースからのイベントを継続するレプリカ機能に影響しないようにしてください。 [セクション17.1.7.1 「レプリケーションステータスの確認」](#)を参照してください。レプリカを維持できない場合は、別のレプリカを追加してバックアッププロセスを分散できます。このシナリオの構成方法の例は、 [セクション17.4.6 「異なるレプリカへの異なるデータベースのレプリケート」](#)を参照してください。

17.4.1.2 レプリカからの RAW データのバックアップ

コピーされるファイルの整合性を保証するには、レプリカサーバーの停止中に MySQL レプリカ上の RAW データファイルのバックアップを実行する必要があります。MySQL サーバーがまだ実行中の場合は、バックグラウンドタスクがデータベースファイルをまだ更新中の可能性があります (特に、[InnoDB](#) などのストレージエンジンをバックグラウンドプロセスで使用するもの)。 [InnoDB](#) では、クラッシュリカバリ中にこれらの問題を解決する必要がありますが、この機能を利用すると、ソースの実行に影響を与えることなく、バックアッププロセス中にレプリカサーバーを停止できるためです。

サーバーをシャットダウンしてファイルをバックアップするには：

1. レプリカ MySQL サーバーを停止します：

```
shell> mysqladmin shutdown
```

2. データファイルをコピーします。cp、tar、WinZip など、コピーまたはアーカイブに適したユーティリティを使用できます。たとえば、データディレクトリが現在のディレクトリの下にある場合、ディレクトリ全体を次のようにアーカイブできます。

```
shell> tar cf /tmp/dbbackup.tar ./data
```

3. MySQL サーバーを再起動します。Unix の場合：

```
shell> mysqld_safe &
```

Windows の場合：

```
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqld"
```

通常、レプリカ MySQL サーバーのデータディレクトリ全体をバックアップする必要があります。データをリストアしてレプリカとして操作できるようにするには (レプリカに障害が発生した場合など)、データに加えて、レプリカ接続メタデータリポジトリと適用者メタデータリポジトリ、およびリレーログファイルが必要です。これらのアイテムは、レプリカデータのリストア後にレプリケーションを再開するために必要です。MySQL 8.0 のデフォルトであるレプリカ接続メタデータリポジトリおよび適用機能メタデータリポジトリ ([セクション 17.2.4 「リレーログおよびレプリケーションメタデータリポジトリ」](#) を参照) にテーブルが使用されている場合、これらのテーブルはデータディレクトリとともにバックアップされます。非推奨のリポジトリにファイルが使用されている場合は、それらを個別にバックアップする必要があります。リレーログファイルがデータディレクトリとは異なる場所に配置されている場合は、個別にバックアップする必要があります。

リレーログを失っても `relay-log.info` ファイルが残っている場合は、それをチェックして、レプリケーション SQL スレッドがソースバイナリログで実行された距離を判断できます。その後、[CHANGE REPLICATION SOURCE TO](#) ステートメント (MySQL 8.0.23 から) または [CHANGE MASTER TO](#) ステートメント (MySQL 8.0.23 より前) を `SOURCE_LOG_FILE` | `MASTER_LOG_FILE` および `SOURCE_LOG_POS` | `MASTER_LOG_POS` オプションとともに使用して、バイナリログを再度読み取り、そのレプリカに記録することができます。これには、バイナリログがソースサーバーにまだ存在している必要があります。

レプリカが `LOAD DATA` ステートメントをレプリケートしている場合は、レプリカがこの目的で使用するディレクトリに存在する `SQL_LOAD-*` ファイルもバックアップする必要があります。レプリカでは、中断された `LOAD DATA` 操作のレプリケーションを再開するために、これらのファイルが必要です。このディレクトリの場所は、`slave_load_tmpdir` システム変数の値です。その変数を設定してサーバーを起動しなかった場合、ディレクトリの場所は `tmpdir` システム変数の値になります。

17.4.1.3 ソースまたはレプリカを読み取り専用にすることによるバックアップ

グローバル読み取りロックを取得し、`read_only` システム変数を操作して、バックアップするサーバーの読み取り専用状態を変更することで、レプリケーション設定のソースサーバーまたはレプリカサーバーをバックアップできます：

1. サーバーを読み取り専用にします (検索のみが処理され、更新はブロックされます)。
2. バックアップを実行します。
3. サーバーを通常の読み取り/書き込み状態に戻します。

注記

このセクションの手順では、バックアップするサーバーを、サーバーからデータを取得するバックアップ方式 (`mysqldump` など) に安全な状態に変換しています ([セクション 4.5.4 「mysqldump — データベースバックアッププログラム」](#) を参照してください)。バイナリバックアップを作成するために、ファイルを直接コピーする方法でこれらの手順の使用を試みてはいけません (サーバーが変更後データをまだメモリー内にキャッシュしていてディスクにフラッシュしていない可能性があるため)。

次の手順では、ソースおよびレプリカに対してこれを実行する方法について説明します。ここで説明する両方のシナリオでは、次のレプリケーションセットアップを想定します。

- ソースサーバー S1
- ソースとして S1 を持つレプリカサーバー R1
- S1 に接続されたクライアント C1
- R1 に接続されたクライアント C2

いずれのシナリオでも、グローバル読み取りロックを取得して `read_only` 変数を操作するステートメントは、バックアップ対象のサーバーで実行され、そのサーバーのレプリカには伝播されません。

シナリオ 1: 読み取り専用ソースを使用したバックアップ

次のステートメントを実行して、ソース S1 を読み取り専用状態にします:

```
mysql> FLUSH TABLES WITH READ LOCK;  
mysql> SET GLOBAL read_only = ON;
```

S1 が読み取り専用状態のときは、次の属性が `true` になります。

- サーバーが読み取り専用モードであるため、C1 によって S1 ブロックに送信される更新のリクエスト。
- C1 から S1 に送信されたクエリー結果のリクエストは成功しました。
- S1 でのバックアップの作成は安全です。
- R1 でバックアップを作成することは安全ではありません。このサーバーはまだ実行中で、バイナリログを処理中であつたり、クライアント C2 から着信する要求を更新したりする可能性があります。

S1 が読み取り専用のときに、バックアップを実行してください。たとえば、`mysqldump` を使用できます。

S1 でのバックアップ操作が完了したあとに、これらのステートメントを実行することで S1 を通常の動作状態に戻します。

```
mysql> SET GLOBAL read_only = OFF;  
mysql> UNLOCK TABLES;
```

S1 でのバックアップの実行は安全ですが (バックアップに関するかぎり)、S1 のクライアントは更新の実行をブロックされるため、パフォーマンスに最適ではありません。

この戦略は、レプリケーション設定でのソースのバックアップに適用されますが、非レプリケーション設定で単一のサーバーに使用することもできます。

シナリオ 2: 読み取り専用レプリカによるバックアップ

次のステートメントを実行して、レプリカ R1 を読み取り専用状態にします:

```
mysql> FLUSH TABLES WITH READ LOCK;  
mysql> SET GLOBAL read_only = ON;
```

R1 が読み取り専用状態の間は、次のプロパティが適用されます:

- ソース S1 は引き続き動作するため、ソースでのバックアップは安全ではありません。
- レプリカ R1 は停止しているため、レプリカ R1 上のバックアップは安全です。

これらのプロパティは、一般的なバックアップシナリオの基礎を提供: バックアップの実行中に 1 つのレプリカがビジー状態になっていても、ネットワーク全体には影響せず、バックアップ中もシステムが実行中であるため、問題は発生しません。特に、クライアントはソースサーバーで更新を実行できますが、レプリカのバックアップアクティビティによる影響は受けません。

R1 は読み取り専用ですが、バックアップを実行します。たとえば、`mysqldump` を使用できます。

R1 でのバックアップ操作が完了したら、次のステートメントを実行して、R1 を通常の操作状態にリストアします:

```
mysql> SET GLOBAL read_only = OFF;  
mysql> UNLOCK TABLES;
```

replca は、通常の操作に復元されたあと、ソースバイナリログからの未処理の更新をキャッチアップすることによって、ソースと再度同期します。

17.4.2 レプリカの予期しない停止の処理

サーバーの予期しない停止(クラッシュセーフとも呼ばれる)に対するレプリケーションを回復できるようにするには、レプリカが停止する前にその状態を回復する必要があります。このセクションでは、レプリケーション中のレプリカの予期しない停止の影響、およびレプリケーションを続行するためのリカバリの最善の機会を得るためのレプリカの構成方法について説明します。

レプリカの予期しない停止後、再起動時に、レプリケーション SQL スレッドは、すでに実行されているトランザクションに関する情報をリカバリする必要があります。リカバリに必要な情報は、レプリカアプライアンスのメタデータリポジトリに格納されます。MySQL 8.0 から、このリポジトリは `mysql.slave_relay_log_info` という名前の InnoDB テーブルとしてデフォルトで作成されます。このトランザクションストレージエンジンを使用すると、再起動時に情報が常に回復可能になります。アプライアメタデータリポジトリへの更新は、トランザクションとともにコミットされます。つまり、予期しないサーバーが停止した場合でも、そのリポジトリに記録されるレプリカ進捗情報は、常にデータベースに適用されているものと一貫性があります。適用者メタデータリポジトリの詳細は、[セクション 17.2.4 「リレーログおよびレプリケーションメタデータリポジトリ」](#) を参照してください。

DML トランザクションおよびアトミック DDL は、アトミック操作としてのデータベースへの変更の適用とともに、`mysql.slave_relay_log_info` テーブルのレプリカアプリケーションメタデータリポジトリ内のレプリケーション位置を更新します。完全にアトミックではない DDL ステートメントや、アトミック DDL をサポートしない除外されたストレージエンジンなど、ほかのすべての場合、サーバーが予期せず停止した場合、レプリケートされたデータに関連付けられた更新が `mysql.slave_relay_log_info` テーブルに失われる可能性があります。この場合、更新のリストアは手動プロセスです。MySQL 8.0 でのアトミック DDL のサポートおよび特定のステートメントのレプリケーションの結果の動作の詳細は、[セクション 13.1.1 「アトミックデータ定義ステートメントのサポート」](#) を参照してください。

予期しない停止からレプリカをリカバリするリカバリプロセスは、レプリカの構成によって異なります。リカバリプロセスの詳細は、レプリケーションの選択した方法、レプリカがシングルスレッドかマルチスレッドか、および関連するシステム変数の設定の影響を受けます。リカバリプロセスの全体的な目的は、予期しない停止が発生する前にレプリカデータベースにすでに適用されているトランザクションを識別し、予期しない停止の後にレプリカが見逃したトランザクションを取得して適用することです。

- GTID ベースのレプリケーションの場合、回復プロセスには、レプリカによってすでに受信またはコミットされたトランザクションの GTID が必要です。GTID 自動配置を使用すると、欠落しているトランザクションをソースから取得できます。GTID 自動配置では、ソーストランザクションがレプリカトランザクションと自動的に比較され、欠落しているトランザクションが識別されます。
- ファイル位置ベースのレプリケーションの場合、リカバリプロセスには、レプリカに適用された最後のトランザクションを示す正確なレプリケーション SQL スレッド(適用者)位置が必要です。その位置に基づいて、レプリケーション I/O スレッド(レシーバ)は、その時点からレプリカに適用する必要があるすべてのトランザクションをソースバイナリログから取得します。

GTID ベースのレプリケーションを使用すると、予期しない停止に対して回復可能になるようにレプリケーションを構成することもっとも簡単になります。GTID 自動配置とは、適用されたトランザクションのシーケンスにギャップがある場合でも、レプリカが欠落しているトランザクションを確実に識別して取得できることを意味します。

次の情報は、レプリケーションの制御下にあるかぎりリカバリを保証するために、様々なタイプのレプリカに適した設定の組合せを提供します。

重要

レプリケーションの制御外の一部の要因は、レプリケーションリカバリプロセスおよびリカバリプロセス後のレプリケーションの全体的な状態に影響を与える可能性があります。特に、個々のストレージエンジンの回復プロセスに影響する設定では、レプリカが予期せず停止したためにトランザクションが失われ、レプリケーションリカバリプロセスで使えなくなる可能性があります。次のリストに示す `innodb_flush_log_at_trx_commit=1` 設定は、トランザクションで InnoDB を使用するレプリケーション設定の主要な設定です。ただ

し、InnoDB またはほかのストレージエンジン (特にフラッシュまたは同期に関連するもの) に固有のほかの設定も影響を与える可能性があります。選択したストレージエンジンによって作成されたクラッシュセーフ設定に関する推奨事項を常にチェックして適用します。

レプリカの次の設定の組合せは、予期しない停止に対する最も回復力があります:

- GTID ベースのレプリケーションが使用されている場合 (`gtid_mode=ON`)、`SOURCE_AUTO_POSITION=1` | `MASTER_AUTO_POSITION=1` を設定します。これにより、ソースへの接続の GTID 自動配置がアクティブ化され、欠落しているトランザクションが自動的に識別および取得されます。このオプションは、`CHANGE REPLICATION SOURCE TO` ステートメント (MySQL 8.0.23 の場合) または `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 の場合) を使用して設定します。レプリカに複数のレプリケーションチャンネルがある場合は、チャンネルごとにこのオプションを個別に設定する必要があります。GTID 自動配置の動作の詳細は、[セクション 17.1.3.3 「GTID 自動配置」](#) を参照してください。ファイル位置ベースのレプリケーションが使用されている場合、`SOURCE_AUTO_POSITION=1` | `MASTER_AUTO_POSITION=1` は使用されず、代わりにバイナリログ位置またはリレーログポジションを使用してレプリケーションの開始位置が制御されます。
- 受信した各トランザクションがディスクに書き込まれた後、リレーログをディスクに同期するようにレプリケーション I/O スレッドに指示する `sync_relay_log=1` を設定します。つまり、ソースバイナリログ (アプライヤメタデータリポジトリ内) から読み取られた現在の位置のレプリカレコードが、リレーログに保存されたトランザクションのレコードより前になることはありません。この設定は最も安全ですが、関連するディスク書き込みの数が原因で最も遅くなることにも注意してください。 `sync_relay_log > 1` または `sync_relay_log=0` (同期がオペレーティングシステムによって処理される) では、レプリカが予期せず停止した場合、コミットされたトランザクションがディスクに同期されていない可能性があります。このようなトランザクションでは、リカバリしているレプリカがディスクに最後に同期されたときのリレーログ内の情報に基づいて、トランザクションをスキップするのではなく、トランザクションの取得および適用を再試行する場合に、リカバリプロセスが失敗する可能性があります。 `sync_relay_log=1` の設定は、マルチスレッドレプリカで特に重要です。マルチスレッドレプリカでは、リレーログの情報を使用して一連のトランザクションのギャップを埋めることができない場合、リカバリプロセスは失敗します。シングルスレッドレプリカの場合、リカバリプロセスでリレーログを使用する必要のあるのは、関連情報がアプライヤメタデータリポジトリで使用できない場合のみです。
- 各トランザクションがコミットされる前に InnoDB ログをディスクに同期する `innodb_flush_log_at_trx_commit=1` を設定します。この設定 (デフォルト) により、InnoDB テーブルおよび InnoDB ログがディスクに保存されるため、トランザクションに関する情報がリレーログに不要になります。この設定を `sync_relay_log=1` の設定と組み合わせると、InnoDB テーブルと InnoDB ログの内容が常にリレーログの内容と一致ようになるため、予期しない停止が発生した場合にリレーログファイルをパーージしてもトランザクションのレプリカ履歴に問題のないギャップが生じることはありません。
- レプリケーション SQL スレッドの位置を InnoDB テーブル `mysql.slave_relay_log_info` に格納する `relay_log_info_repository = TABLE` を設定し、トランザクションコミットとともに更新して、常に正確なレコードを確保します。この設定は MySQL 8.0 のデフォルトであり、`FILE` 設定は非推奨です。MySQL 8.0.23 では、システム変数自体の使用は非推奨であるため、省略してデフォルトにできます。`FILE` 設定 (以前のリリースのデフォルト) を使用した場合、情報は、トランザクションの適用後に更新されるデータディレクトリ内のファイルに格納されます。これにより、レプリカが停止するトランザクションの処理ステージや、ファイル自体の破損に応じて、ソースとの同期が失われるリスクが発生します。`relay_log_info_repository = FILE` の設定では、リカバリは保証されません。
- サーバーの起動直後に自動リレーログリカバリを有効にする `relay_log_recovery = ON` を設定します。このグローバル変数はデフォルトで `OFF` に設定され、実行時には読み取り専用ですが、レプリカの予期しない停止後、レプリカの起動時に `--relay-log-recovery` オプションを使用して `ON` に設定できます。この設定では、既存のリレーログファイルが破損または矛盾している場合に無視されることに注意してください。リレーログリカバリプロセスでは、新しいリレーログファイルが開始され、アプライヤメタデータリポジトリに記録されているレプリケーション SQL スレッド位置からソースからトランザクションがフェッチされます。以前のリレーログファイルは、レプリカの通常のパージメカニズムによって時間の経過とともに削除されます。

マルチスレッドレプリカの場合、`relay_log_recovery = ON` を設定すると、リレーログから実行された一連のトランザクションの不整合およびギャップが自動的に処理されます。これらのギャップは、ファイル位置ベースのレプリケーションが使用されている場合に発生することがあります。(詳細は、[セクション 17.5.1.34 「レプリケーションとトランザクションの非一貫性」](#) を参照してください。) リレーログリカバリプロセスは、`START REPLICATION | SLAVE UNTIL SQL_AFTER_MTS_GAPS` ステートメントと同じ方法を使用してギャップを処理します。レプリカが一貫性のないギャップのない状態に達すると、リレーログリカバリプロセスが進行し、レプリケーション SQL スレッド位置からソースからさらにトランザクションがフェッチされます。GTID ベースのレプリケーションが使用されている場

合、このプロセスは不要であり、`MASTER_AUTO_POSITION` が `ON` に設定されている場合、MySQL 8.0.18 からマルチスレッドレプリカはリレーログのリカバリを自動的にスキップするため、`relay_log_recovery` の設定に違いはありません。

17.4.3 行ベースのレプリケーションの監視

行ベースレプリケーション使用時のレプリケーションアプライアンス (SQL) スレッドの現在の進行状況は、パフォーマンススキーマインストゥルメントステージを介してモニターされるため、操作の処理を追跡したり、完了した作業量や推定作業量を確認したりできます。これらのパフォーマンススキーマインストゥルメントステージが有効になっている場合、`events_stages_current` テーブルにはアプライアンスレプリカのスレッドのステージとその進行状況が表示されます。背景情報については、[セクション27.12.5「パフォーマンススキーマステージイベントテーブル」](#) を参照してください。

3 つのすべての行ベースのレプリケーションイベントタイプ (書き込み、更新、削除) の進行状況を追跡するには:

- 次を発行して、3 つのパフォーマンススキーマステージを有効にします:

```
mysql> UPDATE performance_schema.setup_instruments SET ENABLED = 'YES'  
-> WHERE NAME LIKE 'stage/sql/Applying batch of row changes%';
```

- 一部のイベントがレプリケーションアプライアンスレプリカによって処理されるまで待機し、`events_stages_current` テーブルを参照して進行状況を確認します。たとえば、`update` イベントの進捗を取得するには、次のようにします:

```
mysql> SELECT WORK_COMPLETED, WORK_ESTIMATED FROM performance_schema.events_stages_current  
-> WHERE EVENT_NAME LIKE 'stage/sql/Applying batch of row changes (update)';
```

- `binlog_rows_query_log_events` が有効な場合、クエリーに関する情報はバイナリログに格納され、`processlist_info` フィールドに表示されます。このイベントをトリガーした元のクエリーを表示するには:

```
mysql> SELECT db, processlist_state, processlist_info FROM performance_schema.threads  
-> WHERE processlist_state LIKE 'stage/sql/Applying batch of row changes%' AND thread_id = N;
```

17.4.4 異なるソースおよびレプリカのストレージエンジンでのレプリケーションの使用

レプリケーションプロセスでは、ソース上の元のテーブルとレプリカ上のレプリケートされたテーブルが異なるストレージエンジンタイプを使用するかどうかは関係ありません。実際、`default_storage_engine` システム変数はレプリケートされません。

これは、異なるレプリケーションシナリオに異なるエンジンタイプを利用できるという点で、レプリケーションプロセスにいくつかの利点を提供します。たとえば、典型的なスケールアウトシナリオ ([セクション17.4.5「スケールアウトのためにレプリケーションを使用する」](#) を参照) では、トランザクション機能を利用するためにソースで `InnoDB` テーブルを使用しますが、データの読取りのみであるためトランザクションサポートが不要なレプリカでは `MyISAM` を使用します。データロギング環境でレプリケーションを使用する場合は、レプリカで `Archive` ストレージエンジンを使用できます。

ソースとレプリカで異なるエンジンを構成するかどうかは、初期レプリケーションプロセスの設定方法によって異なります:

- `mysqldump` を使用してソースにデータベーススナップショットを作成した場合は、ダンプファイルのテキストを編集して、各テーブルで使用されるエンジンタイプを変更できます。

`mysqldump` の別の方法は、ダンプを使用してレプリカにデータを構築する前に、レプリカで使用しないエンジンタイプを無効にすることです。たとえば、レプリカに `--skip-federated` オプションを追加して、`FEDERATED` エンジンを無効にできます。作成するテーブルに特定のエンジンが存在しない場合、MySQL はデフォルトのエンジンタイプ (通常は `InnoDB`) を使用します。(これには、`NO_ENGINE_SUBSTITUTION` SQL モードが有効でないことが必要です。) この方法で追加のエンジンを無効にする場合は、必要なエンジンのみをサポートするレプリカで使用する特別なバイナリを構築することを検討してください。

- RAW データファイル (バイナリバックアップ) を使用してレプリカを設定する場合、初期テーブル形式を変更することはできません。かわりに、レプリカの起動後に `ALTER TABLE` を使用してテーブルタイプを変更します。

- 現在ソースにテーブルがない新しいソース/レプリカレプリケーション設定の場合は、新しいテーブルの作成時にエンジンタイプを指定しないでください。

レプリケーションソリューションをすでに実行していて、既存のテーブルを別のエンジンタイプに変更する場合は、これらの手順に従ってください。

1. レプリカによるレプリケーション更新の実行を停止します:

```
mysql> STOP SLAVE;  
Or from MySQL 8.0.22:  
mysql> STOP REPLICA;
```

これにより、中断せずにエンジンタイプを変更できます。

2. 変更するテーブルごとに `ALTER TABLE ... ENGINE='engine_type'` を実行します。

3. レプリケーションプロセスを再度開始します:

```
mysql> START SLAVE;
```

または、MySQL 8.0.22 以降では次のようになります:

```
mysql> START REPLICA;
```

`default_storage_engine` 変数はレプリケートされませんが、エンジン仕様を含む `CREATE TABLE` および `ALTER TABLE` ステートメントはレプリカに正しくレプリケートされることに注意してください。CSV テーブルの場合は、次のステートメントを実行します:

```
mysql> ALTER TABLE csvtable ENGINE='MyISAM';
```

このステートメントはレプリケートされます。レプリカのテーブルエンジンタイプを以前に CSV 以外のエンジンに変更した場合でも、レプリカのテーブルエンジンタイプは InnoDB に変換されます。ソースとレプリカでエンジンの差異を保持する場合は、新しいテーブルの作成時にソースで `default_storage_engine` 変数を使用するように注意する必要があります。たとえば、次の代わりに:

```
mysql> CREATE TABLE tablea (columna int) Engine=MyISAM;
```

この形式を使用してください。

```
mysql> SET default_storage_engine=MyISAM;  
mysql> CREATE TABLE tablea (columna int);
```

レプリケートされると、`default_storage_engine` 変数は無視され、レプリカのデフォルトエンジンを使用してレプリカで `CREATE TABLE` ステートメントが実行されます。

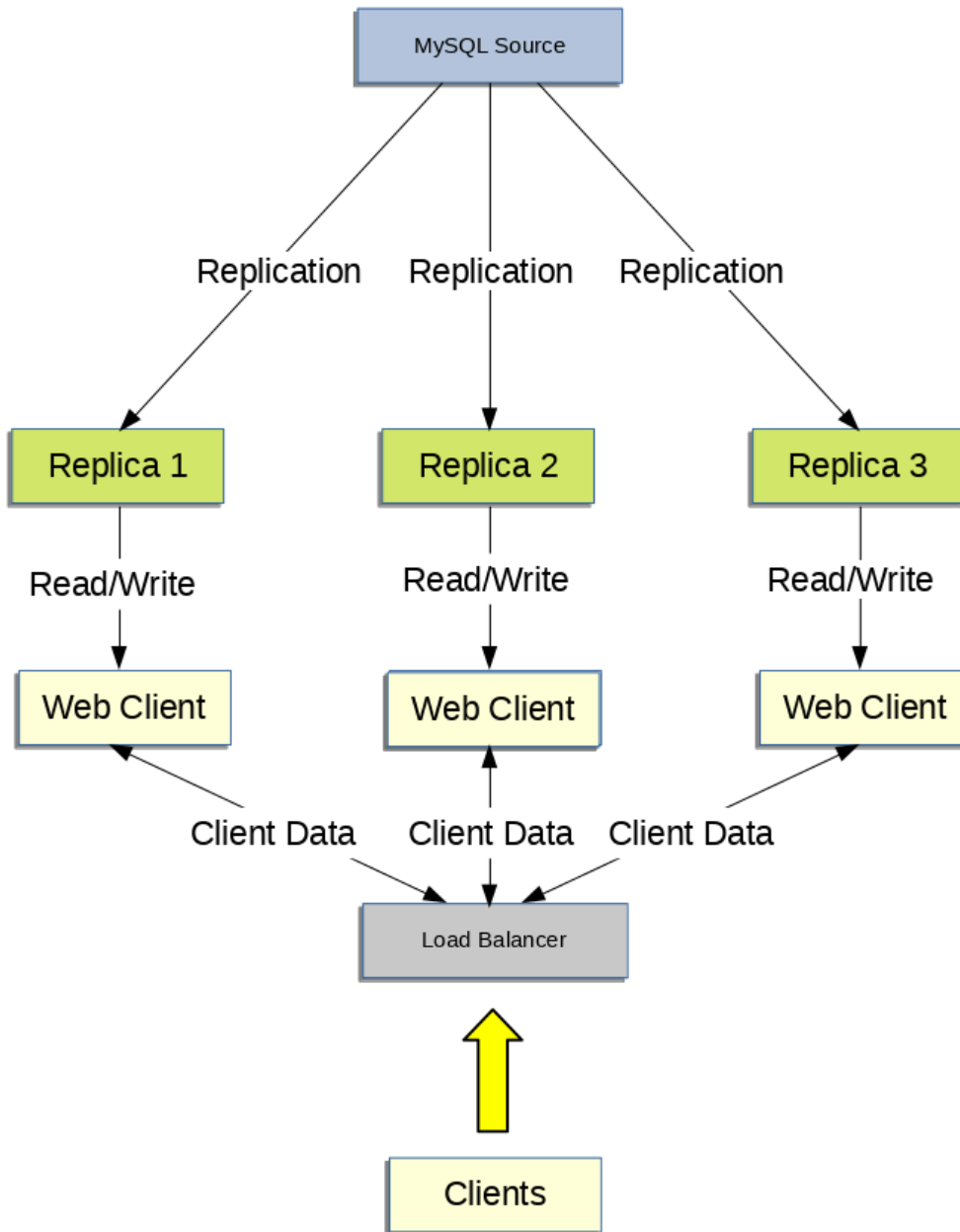
17.4.5 スケールアウトのためにレプリケーションを使用する

レプリケーションをスケールアウトソリューションとして、つまり、いくつかの合理的な制限内でデータベースクエリーの負荷を複数のデータベースサーバーに分割するために使用できます

レプリケーションは1つのソースから1つ以上のレプリカに分散するため、スケールアウトにレプリケーションを使用するのは、読取り数が多く、書込み/更新数が少ない環境で最適です。ほとんどの web サイトは、ユーザーが Web サイトを参照したり、記事を読んだり、投稿したり、製品を表示したりするこのカテゴリに該当します。更新は、セッション管理中、購入するとき、またはフォーラムにコメント/メッセージを追加するときのみ発生します。

この状況でのレプリケーションでは、書込みが必要なときに web サーバーがソースと通信できるようにしながら、レプリカに読取りを分散できます。このシナリオのためのサンプルレプリケーションレイアウトは、[図17.1「スケールアウト中のパフォーマンスを向上するためにレプリケーションを使用する」](#)で見ることができます。

図 17.1 スケールアウト中のパフォーマンスを向上するためにレプリケーションを使用する



データベースにアクセスするコードの一部が適切に抽象化/モジュール化されている場合は、それを複製されたセットアップで動作するように変換することはとても効率的かつ簡単であるはずですが。すべての書き込みをソースに送信し、読み取りをソースまたはレプリカに送信するように、データベースアクセスの実装を変更します。コードがこのレベルの抽象を備えていない場合、複製されたシステムのセットアップは整理するための機会および動機となります。まずは、次の関数を実装するラッパーライブラリまたはモジュールを作成してください。

- `safe_writer_connect()`
- `safe_reader_connect()`
- `safe_reader_statement()`

- `safe_writer_statement()`

各関数名の `safe_` は、その関数がすべてのエラー条件の処理を引き受けることを意味します。関数に別の名前を使用できます。重要なことは、読み取りのための接続、書き込みのための接続、読み取りの実行、および書き込みの実行に対して、統一されたインタフェースを持つことです。

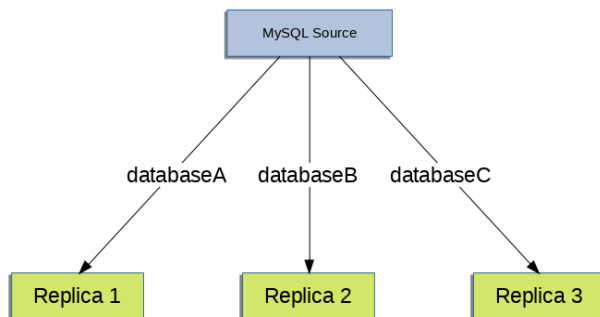
次に、ラッパーライブラリを使用するようにクライアントコードを変換してください。これは、最初は苦しくて怖い工程かもしれませんが、長い目でみるとやるだけの価値があります。ここで説明したアプローチを使用するすべてのアプリケーションは、複数のレプリカを含むアプリケーションであっても、ソース/レプリカ構成を利用できます。こうしたコードは非常に保守しやすく、トラブルシューティングオプションを追加するのも手間がかかりません。変更する必要があるのは、1つまたは2つの関数のみです(たとえば、各ステートメントの所要時間、または発行されたステートメントのうちエラーが発生したステートメントをログに記録する場合)。

多数のコードを記述した場合は、変換スクリプトを記述して変換タスクを自動化できます。理想的には、コードが一貫性のあるプログラミングスタイル規則を使用すべきです。そうでない場合は、一貫性のあるスタイルを使用するために、とにかく書き換えたり、少なくとも詳しく調べて手動で整理したりすることをお勧めします。

17.4.6 異なるレプリカへの異なるデータベースのレプリケート

単一のソースサーバーがあり、異なるデータベースを異なるレプリカにレプリケートする必要がある場合があります。たとえば、データ分析時の負荷を分散するために、異なる売上データを異なる部門に分散したい場合です。このレイアウトの例を図17.2「個別のレプリカへのデータベースのレプリケート」に示します。

図 17.2 個別のレプリカへのデータベースのレプリケート



この分離を実現するには、ソースとレプリカを通常どおりに構成し、各レプリカで `--replicate-wild-do-table` 構成オプションを使用して、各レプリカが処理するバイナリログステートメントを制限します。

重要

ステートメントベースレプリケーションを使用する場合は、`--replicate-do-db` をこの目的に使用しないでください。ステートメントベースレプリケーションを使用すると、このオプションの影響は現在選択されているデータベースによって異なります。このことは、混合形式のレプリケーションにも当てはまります。一部の更新をステートメントベース形式を使用して複製できるためです。

しかし、行ベースレプリケーションだけを使用している場合には、この目的のために `--replicate-do-db` を使用しても安全なはずですが、この場合には、現在選択されているデータベースがオプションの動作に影響しないためです。

たとえば、図17.2「個別のレプリカへのデータベースのレプリケート」に示されている分離をサポートするには、`START REPLICA | SLAVE` を実行する前に、各レプリカを次のように構成する必要があります：

- レプリカ 1 では `--replicate-wild-do-table=databaseA.%` を使用する必要があります。
- レプリカ 2 では `--replicate-wild-do-table=databaseB.%` を使用する必要があります。
- レプリカ 3 では `--replicate-wild-do-table=databaseC.%` を使用する必要があります。

この構成内の各レプリカは、ソースからバイナリログ全体を受け取りますが、そのレプリカで有効な `--replicate-wild-do-table` オプションに含まれるデータベースとテーブルに適用されるバイナリログからのイベントのみを実行します。

レプリケーションを開始する前にレプリカに同期する必要があるデータがある場合は、次のようないくつかの選択肢があります:

- すべてのデータを各レプリカに同期し、保持しないデータベースまたはテーブル、あるいはその両方を削除します。
- `mysqldump` を使用して、データベースごとに個別のダンプファイルを作成し、各レプリカに適切なダンプファイルをロードします。
- RAW データファイルダンプを使用し、各レプリカに必要な特定のファイルおよびデータベースのみを含めます。

注記

これは、`innodb_file_per_table` を使用しないかぎり、InnoDB データベースでは機能しません。

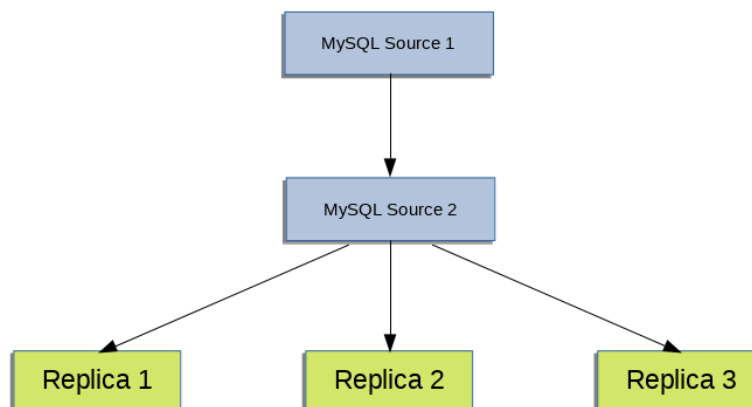
17.4.7 レプリケーションパフォーマンスを改善する

ソースに接続するレプリカの数が増えるにつれて、各レプリカがソースへのクライアント接続を使用するため、負荷も最小限に抑えられます。また、各レプリカはソースバイナリログの完全なコピーを受信する必要があるため、ソースのネットワーク負荷も増加し、ボトルネックが発生する可能性があります。

あるソースに接続されている多数のレプリカを使用しており、そのソースがリクエストの処理にもビジー状態である場合(たとえば、スケールアウトソリューションの一部として)、レプリケーションプロセスのパフォーマンスを向上させることが必要な場合があります。

レプリケーションプロセスのパフォーマンスを向上させる方法の1つは、ソースを1つのレプリカにのみレプリケートし、残りのレプリカが個々のレプリケーション要件のためにこのプライマリレプリカに接続できるようにする、より深いレプリケーション構造を作成することです。この構造のサンプルを [図17.3「追加のレプリケーションソースを使用したパフォーマンスの向上」](#) に示します。

図 17.3 追加のレプリケーションソースを使用したパフォーマンスの向上



これが機能するには、MySQL インスタンスを次のように構成する必要があります。

- ソース 1 は、すべての変更および更新がデータベースに書き込まれるプライマリソースです。バイナリロギングは、両方のソースサーバー(デフォルト)で有効になっています。
- ソース 2 はサーバー Source 1 へのレプリカで、レプリケーション構造内の残りのレプリカにレプリケーション機能を提供します。ソース 2 は、ソース 1 への接続が許可されている唯一のマシンです。ソース 2 では、`--log-slave-`

`updates` オプションが有効 (デフォルト) になっています。このオプションを使用すると、ソース 1 からのレプリケーション命令もソース 2 のバイナリログに書き込まれるため、それらを実際のレプリカにレプリケートできます。

- レプリカ 1、レプリカ 2 およびレプリカ 3 はレプリカとしてソース 2 に機能し、ソース 2 からの情報をレプリケートします。この情報は、実際にはソース 1 に記録されたアップグレードで構成されます。

前述のソリューションを直接データベースソリューションとして使用すると、プライマリソースでのクライアントの負荷とネットワークインターフェースの負荷が軽減されるため、プライマリソースの全体的なパフォーマンスが向上します。

レプリカがソースでのレプリケーションプロセスの維持に問題がある場合は、いくつかのオプションを使用できます:

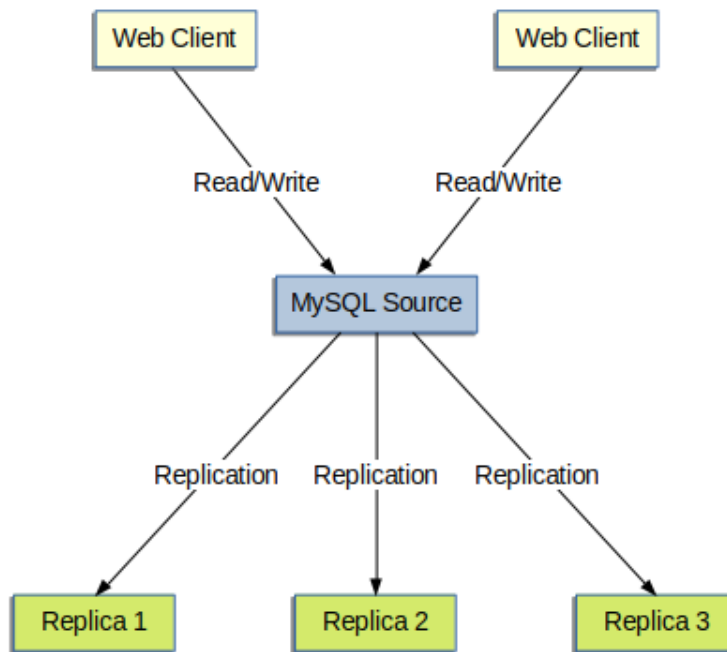
- 可能であれば、リレーログとデータファイルを異なる物理ドライブに置きます。これを行うには、`relay_log` システム変数を設定してリレーログの場所を指定します。
- バイナリログファイルおよびリレーログファイルの読み取りに対するディスク I/O アクティビティーが大きい場合は、`rpl_read_size` システム変数の値を増やすことを検討してください。このシステム変数は、ログファイルから読み取られるデータの最小量を制御し、それを増やすと、ファイルデータがオペレーティングシステムによって現在キャッシュされていない場合にファイルの読み取りおよび I/O の停止が減少する可能性があります。バイナリログおよびリレーログファイルから読み取るスレッドごとに、この値のバッファが割り当てられます。これには、ソース上のダンプスレッドやレプリカ上のコーディネータスレッドも含まれます。したがって、大きな値を設定すると、サーバーのメモリー消費に影響する可能性があります。
- レプリカがソースより大幅に低速な場合は、異なるデータベースを異なるレプリカにレプリケートする責任を分ける必要がある場合があります。セクション 17.4.6 「異なるレプリカへの異なるデータベースのレプリケート」を参照してください。
- ソースでトランザクションを使用していて、レプリカでのトランザクションサポートに関心がない場合は、レプリカで `MyISAM` または別の非トランザクションエンジンを使用します。セクション 17.4.4 「異なるソースおよびレプリカのストレージエンジンでのレプリケーションの使用」を参照してください。
- レプリカがソースとして機能せず、障害発生時にソースを起動できるようにするための潜在的なソリューションがある場合は、レプリカの `log_slave_updates` システム変数を無効にできます。これにより、「dumb」レプリカは、実行したイベントを独自のバイナリログに記録することもできなくなります。

17.4.8 フェイルオーバー中のソースの切替え

(MySQL 8.0.23 の) `CHANGE REPLICATION SOURCE TO` ステートメントまたは `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 の前) を使用して、新しいソースに変更するようレプリカに指示できます。レプリカは、ソース上のデータベースがレプリカ上のデータベースと互換性があるかどうかをチェックしません。単に、新しいソースバイナリログ内の指定された座標からイベントの読み取りおよび実行を開始します。フェイルオーバーの状況では、グループ内のすべてのサーバーが同じバイナリログファイルから同じイベントを実行するのが一般的であるため、イベントのソースを変更しても、変更を加えるときに注意することで、データベースの構造または完全性に影響を与えないはずで

す。レプリカは、バイナリロギングを有効にして実行するようにしてください (`--log-bin` オプション)。これはデフォルトです。GTID をレプリケーションに使用しない場合は、レプリカも `--log-slave-updates=OFF` で実行する必要があります (レプリカ更新のロギングがデフォルトです)。このように、レプリカはレプリカ `mysqld` を再起動せずにソースになる準備ができています。図 17.4 「レプリケーションを使用する冗長性、初期構造」で示す構造を想定してください。

図 17.4 レプリケーションを使用する冗長性、初期構造



この図では、MySQL Source はソースデータベースを保持し、MySQL Replica ホストはレプリカであり、Web Client マシンはデータベースの読み取りおよび書き込みを発行しています。読み取りのみを発行する (通常はレプリカに接続される) Web クライアントは、障害発生時に新しいサーバーに切り替える必要がないため、表示されません。読み取り/書き込みスケーラブルレプリケーション構造の詳細例については、[セクション17.4.5「スケーラブルのためにレプリケーションを使用する」](#)を参照してください。

各 MySQL レプリカ (Replica 1、Replica 2 および Replica 3) は、バイナリロギングを有効にして `--log-slave-updates=OFF` で実行されるレプリカです。 `--log-slave-updates=OFF` が指定されている場合、ソースからレプリカによって受信された更新はバイナリログに記録されないため、各レプリカのバイナリログは最初は空になります。なんらかの理由で MySQL Source が使用できなくなった場合は、いずれかのレプリカを選択して新しいソースにすることができます。たとえば、Replica 1 を選択した場合、すべての Web Clients は Replica 1 にリダイレクトされ、バイナリログに更新が書き込まれます。その後、Replica 2 および Replica 3 は Replica 1 からレプリケートする必要があります。

`--log-slave-updates=OFF` でレプリカを実行する理由は、いずれかのレプリカが新しいソースになった場合に、レプリカが更新を 2 回受信しないようにするためです。Replica 1 で `--log-slave-updates` が有効になっている場合 (デフォルト)、Source から受信した更新が独自のバイナリログに書き込まれます。つまり、Replica 2 が Source から Replica 1 にソースとして変更されると、Source からすでに受信した Replica 1 から更新を受信する可能性があります。

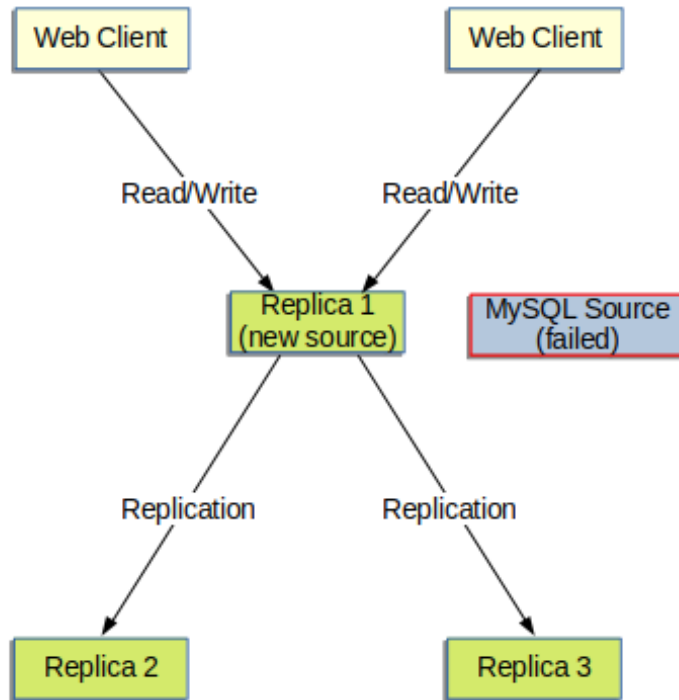
すべてのレプリカがリレーログ内のすべてのステートメントを処理したことを確認します。各レプリカで `STOP REPLICHA | SLAVE_IO_THREAD` を発行し、`Has read all relay log` が表示されるまで `SHOW PROCESSLIST` の出力を確認します。これがすべてのレプリカに当てはまる場合は、新しい設定に再構成できます。ソースになるように昇格されるレプリカ Replica 1 で、`STOP REPLICHA | SLAVE` および `RESET MASTER` を発行します。

他のレプリカ Replica 2 および Replica 3 では、`STOP REPLICHA | SLAVE` および `CHANGE REPLICATION SOURCE TO SOURCE_HOST='Replica1'` または `CHANGE MASTER TO MASTER_HOST='Replica1'` を使用します ('Replica1' は Replica 1 の実際のホスト名を表します)。 `CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO` を使用するには、Replica 2 または Replica 3 (user, password, port) から Replica 1 に接続する方法に関するすべての情報を追加します。このシナリオでは、最初のバイナリログファイルと位置 4 がデフォルトであるため、ステートメントを発行するときに、読み取る Replica 1 バイナリログファイルまたはログ位置の名前を指定する必要はありません。最後に、Replica 2 および Replica 3 で `START REPLICHA | SLAVE` を実行します。

新しいレプリケーションの設定が完了したら、そのステートメントを **Replica 1** に送るように各 **Web Client** に指示する必要があります。この時点から、**Web Client** から **Replica 1** に送信されたすべての update ステートメントが **Replica 1** のバイナリログに書き込まれ、**Source** の停止後に **Replica 1** に送信されたすべての update ステートメントが含まれます。

結果のサーバー構造を図17.5「ソース障害後のレプリケーションを使用した冗長性」に示します。

図 17.5 ソース障害後のレプリケーションを使用した冗長性



Source が再び使用可能になったら、**Replica 1** のレプリカにする必要があります。これを行うには、**Replica 2** および **Replica 3** で発行されたのと同じ **CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO** ステートメントを **Source** で発行します。その後、**Source** は **Replica 1** のレプリカになり、オフライン中に失われた **Web Client** 書き込みを取得します。

Source を再度ソースにするには、**Replica 1** を使用できず、**Source** を新しいソースにする場合と同様に、前述の手順を使用します。この手順では、**Source** の **Replica 1**、**Replica 2** および **Replica 3** レプリカを作成する前に、必ず **Replica 1** で **RESET MASTER** を実行してください。これを実行できない場合、レプリカは、**Source** が使用できなくなった時点より前の **Web Client** アプリケーションから失効した書き込みを取得する可能性があります。

レプリカが同じソースを共有している場合でも、レプリカ間で同期が行われなため、一部のレプリカが他のレプリカよりかなり進んでいる可能性があることに注意してください。これは場合によっては、前の例で説明した手順が期待どおりに機能しない可能性があることを意味します。ただし、実際には、すべてのレプリカのリレーログを比較的近いものにする必要があります。

ソースの場所についてアプリケーションに通知する方法の 1 つは、ソースサーバーの動的 DNS エントリを保持することです。bind で nsupdate を使用することで DNS を動的に更新できます。

17.4.9 非同期接続フェイルオーバーによるソースの切替え

MySQL 8.0.22 以降では、レプリカからソースへの既存の接続が失敗した後、非同期接続フェイルオーバーメカニズムを使用して、新しいソースへの非同期 (ソースからレプリカ) レプリケーション接続を自動的に確立できます。非同期

接続フェイルオーバーメカニズムを使用すると、データを共有する複数の MySQL サーバーまたはサーバーグループとのレプリカの同期を維持できます。潜在的なソースサーバーのリストはレプリカに格納され、接続障害が発生すると、設定した重み付け優先度に基づいて新しいソースがリストから選択されます。

MySQL 8.0.23 から、非同期接続フェイルオーバーメカニズムでは、グループメンバーシップに対する変更を自動的に監視し、プライマリサーバーとセカンダリサーバーを区別することで、グループレプリケーショントポロジもサポートされます。グループメンバーをソースリストに追加し、それを管理対象グループの一部として定義すると、非同期接続フェイルオーバーメカニズムによってソースリストが更新され、メンバーシップの変更にあわせて保持され、グループメンバーの追加または削除が自動的に行われます。接続およびステータスの取得に使用されるのは、大多数のオンライングループメンバーのみです。管理対象グループの最後に残っているメンバーは、グループから移動しても自動的に削除されないため、管理対象グループの構成は保持されますが、管理対象グループが不要になった場合は手動で削除できます。

レプリケーションチャンネルの非同期接続フェイルオーバーをアクティブ化するには、チャンネルの `CHANGE REPLICATION SOURCE TO` ステートメント (MySQL 8.0.23) または `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 より前) で `SOURCE_CONNECTION_AUTO_FAILOVER=1` を設定します。GTID 自動配置は、チャンネルで使用されている必要があります (`SOURCE_AUTO_POSITION = 1 | MASTER_AUTO_POSITION = 1`)。このオプションは、レプリカの実行中に設定できます。

重要

ソースへの既存の接続に失敗した場合、レプリカはまず、`CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO` ステートメントの `SOURCE_RETRY_COUNT | MASTER_RETRY_COUNT` オプションで指定された回数だけ同じ接続を再試行します。試行間隔は、`SOURCE_CONNECT_RETRY | MASTER_CONNECT_RETRY` オプションによって設定されます。これらの試行を使い果たすと、非同期接続フェイルオーバーメカニズムが引き継ぎます。単一ソースへの接続用に設計されたこれらのオプションのデフォルトでは、レプリカは 60 日間同じ接続を再試行します。非同期接続フェイルオーバーメカニズムをすぐにアクティブ化できるようにするには、接続障害の原因が一時的なネットワーク停止である場合に備えて、`SOURCE_RETRY_COUNT | MASTER_RETRY_COUNT` および `SOURCE_CONNECT_RETRY | MASTER_CONNECT_RETRY` を、同じソースでの再試行を数回のみ許可する最小数に設定します。適切な値は、`SOURCE_RETRY_COUNT=3 | MASTER_RETRY_COUNT=3` と `SOURCE_CONNECT_RETRY=10 | MASTER_CONNECT_RETRY=10` です。これにより、レプリカは 10 秒間隔で接続を 3 回再試行します。

また、レプリケーションチャンネルのレプリカにソースリストを設定します。

`asynchronous_connection_failover_add_source` および `asynchronous_connection_failover_delete_source` UDF を使用してソースリストを設定および管理し、単一のレプリケーションサーバーを追加および削除します。サーバーの管理対象グループを追加および削除するには、かわりに `asynchronous_connection_failover_add_managed` および `asynchronous_connection_failover_delete_managed` UDF を使用します。

UDF は、関連するレプリケーションチャンネルに名前を付け、チャンネルソースリストに対して追加または削除する MySQL インスタンスのホスト名、ポート番号、ネットワーク名前スペースおよび重み付け優先度 (1-100、100 が最高の優先度) を指定します。管理対象グループの場合は、管理対象サービスのタイプ (現在使用可能なのは Group Replication のみ) および管理対象グループの識別子 (Group Replication の場合は `group_replication_group_name` システム変数の値) も指定します。管理対象グループを追加する場合、追加する必要があるのは 1 つのグループメンバーのみで、レプリカは現在のグループメンバーシップから残りを自動的に追加します。管理対象グループを削除すると、グループ全体がまとめて削除されます。

MySQL 8.0.22 では、ソースへのレプリカ接続の失敗後に非同期接続フェイルオーバーメカニズムがアクティブ化され、`START REPLICATION | SLAVE` ステートメントが発行されて新しいソースへの接続が試行されます。このリリースでは、ソースの停止またはネットワーク障害が原因でレプリケーション I/O スレッドが停止した場合、接続はフェイルオーバーします。レプリケーションスレッドが `STOP REPLICATION | SLAVE` ステートメントによって停止された場合など、他の状況では接続はフェイルオーバーされません。

MySQL 8.0.23 では、ソースリストで使用可能な別のサーバーの優先度 (重み) 設定が高い場合、非同期接続フェイルオーバーメカニズムによって接続もフェイルオーバーされます。この機能により、レプリカは常に最適なソースサーバーに接続されたままになり、管理対象グループと単一 (非管理対象) サーバーの両方に適用されます。管理対象グループの場合、ソースの重みは、プライマリサーバーとセカンダリサーバーのどちらであるかによって割り当てられます。したがって、プライマリに高い重みを与え、セカンダリに低い重みを与えるように管理対象グループを設定し

た場合、プライマリが変更されると、新しいプライマリに高い重みが割り当てられるため、レプリカは接続を介して変更されます。非同期接続フェイルオーバーメカニズムでは、現在接続している管理対象ソースサーバーが管理対象グループから離れるか、管理対象グループの大部分に存在しなくなった場合にも、接続が変更されます。

接続をフェイルオーバーする場合、チャンネルのソースリストにリストされている代替ソースの中で、優先度(重み)が最も高いソースが最初の接続試行に対して選択されます。レプリカは、まずソースサーバーに接続できることを確認します。管理対象グループの場合は、ソースサーバーがグループ内で **ONLINE** ステータスになっていることを確認します。最も重み付けされたソースが使用できない場合、レプリカは、リストされたすべてのソースを重みの降順で試行し、最も重み付けされたソースから再開します。複数のソースの重みが同じ場合、レプリカはそれらをランダムに順序付けします。レプリカは、リストの操作を再度開始する必要がある場合、元の接続障害が発生したソースを含めて再試行します。

ソースリストは `mysql.replication_asynchronous_connection_failover` および `mysql.replication_asynchronous_connection_failover_managed` テーブルに格納され、「パフォーマンススキーマ」テーブル `replication_asynchronous_connection_failover` で表示できます。レプリカは、モニタースレッドを使用して管理対象グループのメンバーシップを追跡し、ソースリスト (`thread/sql/replica_monitor`) を更新します。 `CHANGE REPLICATION SOURCE TO` | `CHANGE MASTER TO` ステートメントの `SOURCE_CONNECTION_AUTO_FAILOVER` オプションの設定およびソースリストは、リモートクローニング操作中にレプリカのクローンに転送されます。

非同期接続フェイルオーバーメカニズムを使用するための要件は次のとおりです:

- GTID がソースおよびレプリカ (`gtid_mode=ON`) で使用されている必要があり、GTID 自動配置がソースへの接続に使用されるように、`CHANGE REPLICATION SOURCE TO` | `CHANGE MASTER TO` ステートメントの `SOURCE_AUTO_POSITION` | `MASTER_AUTO_POSITION` オプションがレプリカで有効になっている必要があります。
- チャンネルのソースリスト内のすべてのソースサーバーに、同じレプリケーションユーザーアカウントおよびパスワードが存在する必要があります。このアカウントは、各ソースへの接続に使用されます。チャンネルごとに異なるアカウントを設定できます。
- レプリケーションユーザーアカウントには、たとえば `GRANT SELECT ON performance_schema.* TO 'repl_user';` を発行して、「パフォーマンススキーマ」テーブルに対する `SELECT` 権限が付与されている必要があります。
- レプリケーションユーザーアカウントおよびパスワードは、代替ソースへの接続のために自動再起動時に使用可能である必要があるため、レプリケーションの開始に使用するステートメントには指定できません。これらは、レプリカで `CHANGE REPLICATION SOURCE TO` | `CHANGE MASTER TO` ステートメントを使用してチャンネルに設定し、レプリケーションメタデータリポジトリに記録する必要があります。

17.4.10 準同期レプリケーション

MySQL 8.0 は、非同期レプリケーションを内蔵していますが、さらにプラグインによって実装される準同期レプリケーションへのインタフェースをサポートします。このセクションでは、準同期レプリケーションの概要とその動作について説明します。後続のセクションでは、準同期レプリケーションへの管理インタフェース、およびこれをインストール、構成、およびモニターする方法について説明します。

MySQL レプリケーションはデフォルトで非同期です。ソースはイベントをバイナリログに書き込み、レプリカは準備ができたならそれらを要求します。ソースでは、レプリカがトランザクションを取得して処理したかどうか、またはいつ処理したかは認識されず、イベントがレプリカに到達したことは保証されません。非同期レプリケーションでは、ソースがクラッシュした場合、コミットされたトランザクションがレプリカに送信されていない可能性があります。この場合、ソースからレプリカにフェイルオーバーすると、ソースに対して相対的なトランザクションが欠落しているサーバーにフェイルオーバーする可能性があります。

完全同期レプリケーションでは、ソースがトランザクションをコミットすると、ソースがトランザクションを実行したセッションに戻る前に、すべてのレプリカもトランザクションをコミットしています。完全同期レプリケーションは、ソースから任意のレプリカへのフェイルオーバーがいつでも可能であることを意味します。完全同期レプリケーションの欠点は、トランザクションの完了に多くの遅延が発生する可能性があることです。

準同期レプリケーションは、非同期および完全同期レプリケーションの中間です。ソースは、少なくとも1つのレプリカがイベントを受信してログに記録する(必要な数のレプリカが構成可能)まで待機してから、トランザクションをコミットします。ソースは、すべてのレプリカが受信を確認するのを待機するのではなく、レプリカからの確認の

みを必要とし、レプリカ側でイベントが完全に実行およびコミットされたことを必要としません。したがって、準同期レプリケーションでは、ソースがクラッシュした場合、コミットされたすべてのトランザクションが少なくとも1つのレプリカに送信されていることが保証されます。

非同期レプリケーションと比較して、準同期レプリケーションではデータ整合性が向上します。コミットが正常に終了すると、データが2つ以上の場所に存在することがわかっているためです。準同期ソースが必要な数のレプリカから確認を受信するまで、トランザクションは保留中でコミットされません。

完全同期レプリケーションと比較すると、準同期レプリケーションは、データ整合性の要件 (トランザクションの受信を確認するレプリカの数) とコミットの速度のバランスをとるように構成できるため、高速です。これは、レプリカを待機する必要があるため、速度が遅くなります。

重要

準同期レプリケーションでは、ソースがクラッシュし、レプリカへのフェイルオーバーが実行された場合、障害が発生したソースはレプリケーションソースとして再利用されないため、破棄するようにしてください。どのレプリカによっても確認されなかったため、フェイルオーバーの前にコミットされなかったトランザクションがある可能性があります。

すべてのサーバーが同じトランザクションを同じ順序で受信し、クラッシュしたサーバーがグループに再度参加して自動的に最新になるフォルトトレラントレプリケーショントポロジを実装することを目的としている場合は、グループレプリケーションを使用してこれを実現できます。詳細は、[第18章「グループレプリケーション」](#)を参照してください。

非同期レプリケーションと比較した準同期レプリケーションのパフォーマンスへの影響は、データ整合性を向上させるためのトレードオフです。速度低下の量は、少なくともレプリカにコミットを送信し、レプリカによる受信確認を待機する TCP/IP ラウンドトリップ時間です。これは、準同期レプリケーションは高速ネットワーク上で通信する近いサーバーに最適で、低速ネットワークで通信する遠いサーバーに最悪であることを意味します。準同期レプリケーションでは、バイナリログイベントをソースからレプリカに送信できる速度を制限することによって、ビジーセッションの速度制限も設定されます。あるユーザーがビジー状態の場合、これにより処理速度が低下するため、一部のデプロイメント状況で役立ちます。

ソースとそのレプリカ間の準同期レプリケーションは、次のように動作します:

- レプリカは、ソースへの接続時に準同期対応であるかどうかを示します。
- ソース側で準同期レプリケーションが有効になっていて、少なくとも1つの準同期レプリカが存在する場合、ソースブロックでトランザクションコミットを実行し、少なくとも1つの準同期レプリカがトランザクションのすべてのイベントを受信したことを確認するか、タイムアウトが発生するまで待機するスレッド。
- レプリカは、イベントがリレーログに書き込まれてディスクにフラッシュされた後のみ、トランザクションイベントの受信を確認します。
- レプリカがトランザクションを確認せずにタイムアウトが発生した場合、ソースは非同期レプリケーションに戻ります。少なくとも1つの準同期レプリカがキャッチアップすると、ソースは準同期レプリケーションに戻ります。
- ソース側とレプリカ側の両方で、準同期レプリケーションを有効にする必要があります。準同期レプリケーションがソースで無効になっているか、ソースで有効になっているがレプリカで有効になっていない場合、ソースは非同期レプリケーションを使用します。

ソースがブロックしている間 (レプリカからの確認を待機している間)、トランザクションを実行したセッションには戻りません。ブロックが終了すると、ソースはセッションに戻り、他のステートメントの実行に進むことができます。この時点で、トランザクションはソース側でコミットされ、そのイベントの受信は少なくとも1つのレプリカによって確認されています。セッションに戻る前にソースがトランザクションごとに受信する必要があるレプリカ確認の数は、`rpl_semi_sync_master_wait_for_slave_count` システム変数 (デフォルト値は 1) を使用して構成できます。

ブロックはバイナリログに書き込まれるロールバック後にも発生し、これは非トランザクションテーブルを変更するトランザクションがロールバックされる時に発生します。非トランザクションテーブルへの変更はロールバックできず、レプリカに送信する必要があるため、トランザクションテーブルには影響しませんが、ロールバックされたトランザクションはログに記録されます。

トランザクションコンテキストで発生しないステートメントの場合 (つまり、トランザクションが `START TRANSACTION` または `SET autocommit = 0` で起動されなかったとき)、自動コミットが有効になっていて、各ステー

トメントは暗黙的にコミットされます。準同期レプリケーションでは、明示的なトランザクションコミットの場合と同様に、このような各ステートメントのソースブロックが行われます。

`rpl_semi_sync_master_wait_point` システム変数は、準同期ソースサーバーがトランザクションをコミットしたクライアントにステータスを返す前に、トランザクション受信のレプリカ確認を待機するポイントを制御します。次の値を使用できます:

- **AFTER_SYNC** (デフォルト): ソースは、各トランザクションをバイナリログとレプリカに書き込み、バイナリログをディスクに同期します。ソースは、同期後にトランザクション受信のレプリカ確認を待機します。確認応答を受信すると、ソースはトランザクションをストレージエンジンにコミットし、クライアントに結果を返してから続行できます。
- **AFTER_COMMIT**: ソースは、各トランザクションをバイナリログとレプリカに書き込み、バイナリログを同期し、トランザクションをストレージエンジンにコミットします。ソースは、コミット後にトランザクション受信のレプリカ確認を待機します。確認を受信すると、ソースは結果をクライアントに返し、クライアントは続行できます。

これらの設定のレプリケーション特性は、次のように異なります:

- **AFTER_SYNC** では、すべてのクライアントが同時にコミットされたトランザクションを確認できます。これは、レプリカによって確認され、ソース上のストレージエンジンにコミットされたあとです。したがって、すべてのクライアントにソース上の同じデータが表示されます。

ソース障害が発生した場合、ソースでコミットされたすべてのトランザクションがレプリカにレプリケートされず (リレーログに保存されます)。レプリカが最新であるため、ソースの予期しない終了およびレプリカへのフェイルオーバーは失われません。前述のように、フェイルオーバー後にソースを再利用しないでください。

- **AFTER_COMMIT** では、サーバーがストレージエンジンにコミットしてレプリカの確認応答を受信したあとにのみ、トランザクションを発行するクライアントは戻りステータスを取得します。コミット後およびレプリカの確認前に、他のクライアントはコミット中のクライアントの前にコミット済トランザクションを確認できます。

レプリカがトランザクションを処理しないなどの問題が発生した場合は、予期しないソースの終了およびレプリカへのフェイルオーバーが発生したときに、そのようなクライアントがソースで見た内容と比較してデータの損失を確認できる可能性があります。

17.4.10.1 準同期レプリケーション管理インタフェース

準同期レプリケーションへの管理インタフェースにはいくつかのコンポーネントがあります。

- 準同期機能を実装する 2 つのプラグイン。ソース側には 1 つのプラグイン、レプリカ側には 1 つのプラグインがあります。
- プラグインの動作を制御するシステム変数。例:

- `rpl_semi_sync_master_enabled`

ソースサーバーで準同期レプリケーションを有効にするかどうかを制御します。プラグインを有効または無効にするには、この変数をそれぞれ 1 または 0 に設定します。デフォルトは 0 (オフ) です。

- `rpl_semi_sync_master_timeout`

ソースがタイムアウトして非同期レプリケーションに戻る前にレプリカからの確認応答をコミットで待機する時間を制御するミリ秒単位の値。デフォルト値は 10000 (10 秒) です。

- `rpl_semi_sync_slave_enabled`

`rpl_semi_sync_master_enabled` と似ていますが、レプリカプラグインを制御します。

すべての `rpl_semi_sync_xxx` システム変数は、[セクション 17.1.6.2 「レプリケーションソースのオプションと変数」](#) および [セクション 17.1.6.3 「Replica Server のオプションと変数」](#) で説明されています。

- MySQL 8.0.23 から、コールバックを制限するシステム変数 `replication_sender_observe_commit_only` と、共有ロックを追加して不要なロック取得を回避する `replication_optimize_for_static_plugin_config` を有効にすることで、準同期レプリケーションのパフォーマンスを向上させることができます。これらの設定は、ロックの競合によってパ

パフォーマンスが低下する可能性があるため、レプリカの数が増えるにつれて役立ちます。準同期レプリケーションソースサーバーは、複製と同じロックメカニズムを使用するため、これらのシステム変数を有効にすることによってパフォーマンス上の利点を得ることもできます。

- 準同期レプリケーションモニタリングを有効にするステータス変数。例:

- [Rpl_semi_sync_master_clients](#)

準同期レプリカの数。

- [Rpl_semi_sync_master_status](#)

準同期レプリケーションが現在ソースサーバーで動作しているかどうか。プラグインが有効になっていてコミット通知が発生していない場合、値は 1 です。プラグインが有効になっていない場合、またはコミット確認タイムアウトのためにソースが非同期レプリケーションにフォールバックした場合は 0 です。

- [Rpl_semi_sync_master_no_tx](#)

レプリカによって正常に確認されなかったコミットの数。

- [Rpl_semi_sync_master_yes_tx](#)

レプリカによって正常に確認されたコミットの数。

- [Rpl_semi_sync_slave_status](#)

準同期レプリケーションが現在レプリカで動作しているかどうか。プラグインが有効になっていてレプリケーション I/O スレッドが実行中の場合は 1、それ以外の場合は 0 です。

すべての [Rpl_semi_sync_xxx](#) ステータス変数は[セクション 5.1.10 「サーバーステータス変数」](#)で説明されています。

システム変数およびステータス変数は、適切なソースまたはレプリカプラグインが [INSTALL PLUGIN](#) とともにインストールされている場合のみ使用できます。

17.4.10.2 準同期レプリケーションのインストールと構成

準同期レプリケーションはプラグインを使用して実装されるため、プラグインがサーバーにインストールされて利用できる状態である必要があります。プラグインがインストールされたあと、それに関連付けられたシステム変数によって制御します。これらのシステム変数は、関連付けられたプラグインがインストールされるまで利用できません。

このセクションでは、準同期レプリケーションプラグインをインストールする方法について説明します。プラグインのインストールについての一般的な情報は、[セクション 5.6.1 「プラグインのインストールおよびアンインストール」](#)を参照してください。

準同期レプリケーションを使用するには、次の要件を満たす必要があります。

- プラグインをインストールする機能には、動的ローディングをサポートする MySQL サーバーが必要です。これを検証するために、[have_dynamic_loading](#) システム変数の値が **YES** であることを確認してください。バイナリ配布は動的ローディングをサポートしているはずですが。
- レプリケーションがすでに機能している必要があります。[セクション 17.1 「レプリケーションの構成」](#)を参照してください。
- 複数のレプリケーションチャンネルを構成しないでください。準同期レプリケーションは、デフォルトのレプリケーションチャンネルとのみ互換性があります。[セクション 17.2.2 「レプリケーションチャンネル」](#)を参照してください。

準同期レプリケーションをセットアップするには、次の指示を使用してください。ここで説明する [INSTALL PLUGIN, SET GLOBAL, STOP REPLICA | SLAVE](#) および [START REPLICA | SLAVE](#) ステートメントには、[REPLICATION_SLAVE_ADMIN](#) 権限 (または非推奨の [SUPER](#) 権限) が必要です。

MySQL ディストリビューションには、ソース側とレプリカ側の準同期レプリケーションプラグインファイルが含まれます。

ソースサーバーまたはレプリカサーバーで使用できるようにするには、適切なプラグインライブラリファイルを MySQL プラグインディレクトリ (`plugin_dir` システム変数で指定されたディレクトリ) に配置する必要があります。必要に応じて、サーバーの起動時に `plugin_dir` の値を設定してプラグインディレクトリの場所を構成します。

プラグインライブラリファイルのベース名は、ソース用の `semisync_master`、およびレプリカ用の `semisync_slave` です。ファイル名の接尾辞は、プラットフォームごとに異なります (たとえば、`.so` for Unix and Unix-like systems, `.dll` for Windows)。

ソースプラグインライブラリファイルは、ソースサーバーのプラグインディレクトリに存在する必要があります。レプリカプラグインライブラリファイルは、各レプリカサーバーのプラグインディレクトリに存在する必要があります。

プラグインをロードするには、ソースおよび準同期化する各レプリカで `INSTALL PLUGIN` ステートメントを使用し、必要に応じてプラットフォームの `.so` 接尾辞を調整します。

ソースで、次のようにします:

```
INSTALL PLUGIN rpl_semi_sync_master SONAME 'semisync_master.so';
```

各レプリカで、次のようにします:

```
INSTALL PLUGIN rpl_semi_sync_slave SONAME 'semisync_slave.so';
```

プラグインをインストールしようとする、次に示すようなエラーが Linux で発生する場合は、`libimf` をインストールする必要があります:

```
mysql> INSTALL PLUGIN rpl_semi_sync_master SONAME 'semisync_master.so';
ERROR 1126 (HY000): Can't open shared library
'/usr/local/mysql/lib/plugin/semisync_master.so'
(errno: 22 libimf.so: cannot open shared object file:
No such file or directory)
```

`libimf` は <https://dev.mysql.com/downloads/os-linux.html> から取得できます。

どのプラグインがインストールされているかを確認するには、`SHOW PLUGINS` ステートメントを使用するか、`INFORMATION_SCHEMA.PLUGINS` テーブルを照会してください。

プラグインのインストールを確認するには、`INFORMATION_SCHEMA.PLUGINS` テーブルを調べるか、`SHOW PLUGINS` ステートメントを使用します (セクション 5.6.2 「サーバープラグイン情報の取得」 を参照)。例:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
FROM INFORMATION_SCHEMA.PLUGINS
WHERE PLUGIN_NAME LIKE '%semi%';
+-----+-----+
| PLUGIN_NAME | PLUGIN_STATUS |
+-----+-----+
| rpl_semi_sync_master | ACTIVE      |
+-----+-----+
```

プラグインの初期化に失敗した場合は、サーバーエラーログで診断メッセージを確認してください。

準同期レプリケーションプラグインがインストールされたあとは、デフォルトで無効になっています。準同期レプリケーションを有効にするには、ソース側とレプリカ側の両方でプラグインを有効にする必要があります。一方の側のみが有効な場合、レプリケーションは非同期です。

インストールされたプラグインが有効かどうかを制御するには、該当するシステム変数を設定します。これらの変数は実行時に `SET GLOBAL` を使用して、またはコマンド行またはオプションファイルでサーバー起動時に設定できます。

実行時には、次のソース側システム変数を使用できます:

```
SET GLOBAL rpl_semi_sync_master_enabled = {0|1};
SET GLOBAL rpl_semi_sync_master_timeout = N;
```

レプリカ側では、次のシステム変数を使用できます:


```
SET GLOBAL rpl_semi_sync_slave_enabled = {0|1};
```

`rpl_semi_sync_master_enabled` または `rpl_semi_sync_slave_enabled` の場合、準同期レプリケーションを有効にするには値を 1、無効にするには 0 にすべきです。デフォルトでは、これらの変数は 0 に設定されています。

`rpl_semi_sync_master_timeout` の場合、値 `N` はミリ秒で指定されます。デフォルト値は 10000 (10 秒) です。

実行時にレプリカで準同期レプリケーションを有効にする場合は、レプリケーション I/O スレッドも起動 (すでに実行されている場合は最初に停止) して、レプリカをソースに接続し、準同期レプリカとして登録する必要があります。

```
STOP SLAVE IO_THREAD;
START SLAVE IO_THREAD;
Or from MySQL 8.0.22:
STOP REPLICHA IO_THREAD;
START REPLICHA IO_THREAD;
```

レプリケーション I/O スレッドがすでに実行されていて、再起動しない場合、レプリカは非同期レプリケーションを引き続き使用します。

サーバー起動時に、準同期レプリケーションを制御する変数をコマンド行オプションとしてまたはオプションファイルに設定できます。オプションファイルにリストされる設定は、サーバーが起動するたびに有効になります。たとえば、次のようにして、ソースサーバーおよびレプリカサーバー上の `my.cnf` ファイルに変数を設定できます。

ソースで、次のようにします:

```
[mysqld]
rpl_semi_sync_master_enabled=1
rpl_semi_sync_master_timeout=1000 # 1 second
```

各レプリカで、次のようにします:

```
[mysqld]
rpl_semi_sync_slave_enabled=1
```

17.4.10.3 準同期レプリケーションモニタリング

準同期レプリケーション機能用のプラグインはいくつかのシステム変数とステータス変数を公開しており、その構成および運用状態を判断するためにそれらを調べることができます。

システム変数は準同期レプリケーションがどのように構成されているかを反映します。これらの値を確認するには、`SHOW VARIABLES` を使用します。

```
mysql> SHOW VARIABLES LIKE 'rpl_semi_sync%';
```

ステータス変数によって、準同期レプリケーションの動作をモニターできます。これらの値を確認するには、`SHOW STATUS` を使用します。

```
mysql> SHOW STATUS LIKE 'Rpl_semi_sync%';
```

コミットブロックのタイムアウトまたはレプリカのキャッチアップのためにソースが非同期レプリケーションまたは準同期レプリケーションを切り替えると、`Rpl_semi_sync_master_status` ステータス変数の値が適切に設定されます。ソースでの準同期レプリケーションから非同期レプリケーションへの自動フォールバックは、準同期レプリケーションが実際には動作していない場合でも、`rpl_semi_sync_master_enabled` システム変数の値がソース側で 1 になる可能性があることを意味します。`Rpl_semi_sync_master_status` ステータス変数をモニターして、ソースが現在非同期レプリケーションと準同期レプリケーションのどちらを使用しているかを判断できます。

接続されている準同期レプリカの数を確認するには、`Rpl_semi_sync_master_clients` をチェックします。

レプリカによって正常に確認されたコミットまたは正常に確認されなかったコミットの数
は、`Rpl_semi_sync_master_yes_tx` および `Rpl_semi_sync_master_no_tx` 変数によって示されます。

レプリカ側では、`Rpl_semi_sync_slave_status` は準同期レプリケーションが現在動作しているかどうかを示します。

17.4.11 遅延レプリケーション

MySQL では、レプリカサーバーが、少なくとも指定した時間だけソースより後のトランザクションを意図的に実行するように遅延レプリケーションをサポートしています。このセクションでは、レプリカのレプリケーション遅延を構成する方法と、レプリケーション遅延を監視する方法について説明します。

MySQL 8.0 では、レプリケーションを遅延させる方法は、`immediate_commit_timestamp` および `original_commit_timestamp` (レプリケーション遅延タイムスタンプを参照) の 2 つのタイムスタンプによって異なります。レプリケーショントポロジ内のすべてのサーバーで MySQL 8.0 以上が実行されている場合、遅延レプリケーションはこれらのタイムスタンプを使用して測定されます。即時ソースまたはレプリカがこれらのタイムスタンプを使用していない場合は、MySQL 5.7 からの遅延レプリケーションの実装が使用されます (`Delayed Replication` を参照)。このセクションでは、これらのタイムスタンプをすべて使用しているサーバー間の遅延レプリケーションについて説明します。

デフォルトのレプリケーション遅延は 0 秒です。 `CHANGE REPLICATION SOURCE TO SOURCE_DELAY=N` ステートメント (MySQL 8.0.23 の場合) または `CHANGE MASTER TO MASTER_DELAY=N` ステートメント (MySQL 8.0.23 の場合) を使用して、遅延を `N` 秒に設定します。ソースから受信したトランザクションは、`N` 秒以上が即時ソースでのコミットより後になるまで実行されません。遅延はトランザクションごとに発生し (以前の MySQL バージョンとは異なり)、実際の遅延は `gtid_log_event` または `anonymous_gtid_log_event` にのみ適用されます。トランザクション内の他のイベントは、常に待機時間なしでこれらのイベントに従います。

注記

`START REPLICA | SLAVE` および `STOP REPLICA | SLAVE` はただちに有効になり、遅延は無視されます。 `RESET REPLICA | SLAVE` は遅延を 0 にリセットします。

`replication_applier_configuration` 「パフォーマンススキーマ」テーブルには、`SOURCE_DELAY | MASTER_DELAY` オプションを使用して構成された遅延を示す `DESIRED_DELAY` カラムが含まれます。 `replication_applier_status` 「パフォーマンススキーマ」テーブルには、残りの遅延秒数を示す `REMAINING_DELAY` カラムが含まれています。

遅延レプリケーションはいくつかの目的に使用できます。

- ソースでのユーザーミスから保護します。遅延により、遅延レプリカを誤った直前の時点にロールバックできません。
- 遅延があるときにシステムがどのように動作するかをテストするため。たとえば、アプリケーションでは、レプリカの負荷が高いためにラグが発生する場合があります。しかし、この負荷レベルを生成するのが難しい場合があります。遅延レプリケーションは、負荷をシミュレートしなくても遅延をシミュレートできます。また、遅延レプリカに関連する状態のデバッグにも使用できます。
- バックアップを再ロードせずに、データベースが過去にどのように表示されていたかを検査します。たとえば、1 週間の遅延でレプリカを構成することで、過去数日前にデータベースがどのように表示されたかを確認する必要があります。遅延レプリカを検査できます。

レプリケーション遅延タイムスタンプ

MySQL 8.0 には、バイナリログに書き込まれる (各イベントではなく) 各トランザクションの GTID に関連付けられた次のタイムスタンプに依存するレプリケーショントポロジで遅延 (レプリケーションラグとも呼ばれる) を測定するための新しい方法が用意されています。

- `original_commit_timestamp`: トランザクションが元のソースのバイナリログに書き込まれた (コミットされた) ときのエポック以降のマイクロ秒数。
- `immediate_commit_timestamp`: トランザクションが即時ソースのバイナリログに書き込まれた (コミットされた) ときのエポック以降のマイクロ秒数。

`mysqlbinlog` の出力には、これらのタイムスタンプがエポックからのマイクロ秒の形式で表示され、読みやすくするためにユーザー定義のタイムゾーンに基づく `TIMESTAMP` 形式でも表示されます。例:

```
#170404 10:48:05 server id 1 end_log_pos 233 CRC32 0x016ce647 GTID last_committed=0
\sequence_number=1 original_committed_timestamp=1491299285661130 immediate_commit_timestamp=1491299285643771
# original_commit_timestamp=1491299285661130 (2017-04-04 10:48:05.661130 WEST)
# immediate_commit_timestamp=1491299285643771 (2017-04-04 10:48:05.843771 WEST)
/*!80001 SET @@SESSION.original_commit_timestamp=1491299285661130/*!*/;
SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:1/*!*/';
# at 233
```

原則として、`original_commit_timestamp` は、トランザクションが適用されるすべてのレプリカで常に同じです。ソースレプリケーションレプリケーションでは、(元の) ソースのバイナリログ内のトランザクションの `original_commit_timestamp` は、常にその `immediate_commit_timestamp` と同じです。レプリカのリレーログでは、トランザクションの `original_commit_timestamp` と `immediate_commit_timestamp` はソースのバイナリログと同じですが、独自のバイナリログでは、レプリカがトランザクションをコミットした時点でトランザクションの `immediate_commit_timestamp` が対応します。

グループレプリケーション設定では、元のソースがグループのメンバーである場合、トランザクションをコミットする準備が整うと `original_commit_timestamp` が生成されます。つまり、元のソースでの実行が終了し、その書き込みセットを証明のためにグループのすべてのメンバーに送信する準備が整ったときです。したがって、(グループメンバーであるか、メンバーからレプリケートされるグループ外のレプリカであるかに関係なく) すべてのサーバーに同じ `original_commit_timestamp` がレプリケートされ、トランザクションとそのバイナリログ内の各ストアに、`immediate_commit_timestamp` を使用したローカルコミット時間が記録されます。

グループレプリケーション専用の変更イベントの表示は特殊なケースです。これらのイベントを含むトランザクションは各サーバーによって生成されますが、同じ GTID を共有します (そのため、最初にソースで実行されてからグループにレプリケートされるのではなく、グループのすべてのメンバーが同じトランザクションを実行して適用します)。元のソースがないため、これらのトランザクションの `original_commit_timestamp` はゼロに設定されます。

レプリケーション遅延の監視

以前の MySQL バージョンでレプリケーション遅延 (ラグ) を監視する最も一般的な方法の 1 つは、`SHOW REPLICHA | SLAVE STATUS` の出力の `Seconds_Behind_Master` フィールドに依存することでした。ただし、このメトリックは、グループレプリケーションなどの従来のソースレプリケーション設定より複雑なレプリケーショントポロジを使用する場合には適していません。MySQL 8 に `immediate_commit_timestamp` および `original_commit_timestamp` を追加すると、レプリケーション遅延に関するより詳細な情報が提供されます。これらのタイムスタンプをサポートするトポロジでレプリケーション遅延を監視するには、次の「パフォーマンススキーマ」テーブルを使用することをお勧めします。

- `replication_connection_status`: ソースへの接続の現在のステータス。接続スレッドがリレーログにキューに入れた最後のトランザクションと現在のトランザクションに関する情報を提供します。
- `replication_applier_status_by_coordinator`: マルチスレッドレプリカの使用時にのみ情報を表示するコーディネータスレッドの現在のステータスは、コーディネータスレッドによってワーカーキューにバッファリングされた最後のトランザクションに関する情報と、現在バッファリングしているトランザクションに関する情報を提供します。
- `replication_applier_status_by_worker`: ソースから受信したトランザクションを適用しているスレッドの現在のステータス。レプリケーション SQL スレッド、またはマルチスレッドのレプリカを使用している場合は各ワーカースレッドによって適用されたトランザクションに関する情報を提供します。

これらのテーブルを使用して、対応するスレッドが処理した最後のトランザクションおよびスレッドが現在処理しているトランザクションに関する情報を監視できます。この情報は次のもので構成されます:

- トランザクションの GTID
- レプリカのリレーログから取得されたトランザクション `original_commit_timestamp` および `immediate_commit_timestamp`
- スレッドがトランザクションの処理を開始した時刻
- スレッドが最後に処理したトランザクションの処理を終了した時間

「パフォーマンススキーマ」テーブルに加えて、`SHOW REPLICHA | SLAVE STATUS` の出力には次の 3 つのフィールドがあります:

- `SQL_Delay`: `CHANGE REPLICATION SOURCE TO SOURCE_DELAY=N` (MySQL 8.0.23 から) または `CHANGE MASTER TO MASTER_DELAY=N` (MySQL 8.0.23 より前) を使用して構成されたレプリケーション遅延を示す負でない整数 (単位は秒)。
- `SQL_Remaining_Delay`: `Replica_SQL_Running_State` が `Waiting until MASTER_DELAY seconds after master executed event` の場合、このフィールドには遅延の残り秒数を示す整数が含まれます。ほかのときは、このフィールドは `NULL` です。

- `Replica_SQL_Running_State`: SQL スレッドの状態を示す文字列 (`Replica_IO_State` に類似)。値は、`SHOW PROCESSLIST` で表示される、SQL スレッドの `State` 値と同じです。

レプリケーション SQL スレッドがイベントの実行前に遅延が経過するのを待機している場合、`SHOW PROCESSLIST` はその `State` 値を `Waiting until MASTER_DELAY seconds after master executed event` として表示します。

17.5 レプリケーションの注釈とヒント

17.5.1 レプリケーションの機能と問題

以降のセクションでは、MySQL レプリケーションでサポートされていることとされていないことに関する情報、および特定のステートメントの複製時に発生する可能性がある固有の問題と状況に関する情報を提供します。

ステートメントベースレプリケーションは、ソースとレプリカ間の SQL レベルでの互換性に依存します。つまり、ステートメントベースのレプリケーションが成功するには、使用される SQL 機能がソースサーバーとレプリカサーバーの両方でサポートされている必要があります。現在のバージョンの MySQL でのみ使用可能なソースサーバーで機能を使用する場合、以前のバージョンの MySQL を使用するレプリカにレプリケートすることはできません。このような非互換性は、リリースシリーズ内およびバージョン間でも発生する可能性があります。

MySQL 8.0 と以前の MySQL リリースシリーズの間でステートメントベースレプリケーションを使用する場合は、そのシリーズのレプリケーション特性に関する情報について、以前のリリースシリーズに対応する「MySQL リファレンスマニュアル」のエディションを参照することをお勧めします。

MySQL のステートメントベースレプリケーションでは、ストアドルーチンまたはトリガーの複製で問題が発生する場合があります。これらの問題は、代わりに MySQL の行ベースのレプリケーションを使用することで回避できます。問題の詳細な一覧は、[セクション25.7「ストアプログラムバイナリロギング」](#)を参照してください。行ベースロギングおよび行ベースレプリケーションに関する詳細は、[セクション5.4.4.1「バイナリロギング形式」](#)および[セクション17.2.1「レプリケーション形式」](#)を参照してください。

レプリケーションおよび InnoDB に固有の追加情報については、[セクション15.19「InnoDB と MySQL レプリケーション」](#)を参照してください。NDB Cluster でのレプリケーションに関する情報については、[セクション23.6「NDB Cluster レプリケーション」](#)を参照してください。

17.5.1.1 レプリケーションと AUTO_INCREMENT

`AUTO_INCREMENT`、`LAST_INSERT_ID()` および `TIMESTAMP` の値のステートメントベースのレプリケーションは、次の例外の対象となります:

- `AUTO_INCREMENT` カラムを更新するトリガーまたは関数を呼び出すステートメントは、ステートメントベースレプリケーションでは正しく複製されません。これらのステートメントは安全でないとしてマークされます。(Bug #45677)
- 複合主キーを持ち、この複合キーの先頭カラムでない `AUTO_INCREMENT` カラムを含むテーブルに `INSERT` を実行することは、ステートメントベースロギングまたはレプリケーションにとって安全ではありません。これらのステートメントは安全でないとしてマークされます。(Bug #11754117、Bug #45670)

この問題は InnoDB ストレージエンジンを使用するテーブルに影響しません。`AUTO_INCREMENT` カラムを持つ InnoDB テーブルには、自動インクリメントカラムが唯一または左端のカラムであるキーが少なくとも 1 つ必要であるためです。

- `ALTER TABLE` を使用してテーブルに `AUTO_INCREMENT` カラムを追加すると、レプリカとソースで同じ順序で行が生成されない場合があります。これが発生するのは、行が番号付けされる順序が、テーブルに使用される固有のストレージエンジンおよび行が挿入された順序に依存するためです。ソースとレプリカで順序が同じであることが重要な場合は、`AUTO_INCREMENT` 番号を割り当てる前に行を順序付けする必要があります。カラム `col1` と `col2` を持つテーブル `t1` に `AUTO_INCREMENT` カラムを追加するものと仮定すると、次のステートメントは `t1` と同じであるけれども `AUTO_INCREMENT` カラムを持つ新しいテーブル `t2` を生成します。

```
CREATE TABLE t2 LIKE t1;
ALTER TABLE t2 ADD id INT AUTO_INCREMENT PRIMARY KEY;
INSERT INTO t2 SELECT * FROM t1 ORDER BY col1, col2;
```


重要

ソースとレプリカの両方で同じ順序を保証するには、`ORDER BY` 句で `t1` の all カラムを指定する必要があります。

上記の指示には `CREATE TABLE ... LIKE` の制限が適用されます。外部キー定義は `DATA DIRECTORY` および `INDEX DIRECTORY` テーブルオプションと同様に無視されます。テーブル定義がこれらの特性を含む場合、`t1` の作成に使用したのと同じであるけれども `AUTO_INCREMENT` カラムを追加した `CREATE TABLE` ステートメントを使用して、`t2` を作成してください。

`AUTO_INCREMENT` カラムを持つコピーを作成および移入するために使用する方法にかかわらず、最終手順は元のテーブルを削除してコピーの名前を変更することです。

```
DROP t1;  
ALTER TABLE t2 RENAME t1;
```

[セクションB.3.6.1「ALTER TABLE での問題」](#) も参照してください。

17.5.1.2 レプリケーションと BLACKHOLE テーブル

`BLACKHOLE` ストレージエンジンはデータを受け入れますが、それを破棄し、格納しません。バイナリロギングを実行するときは、使用しているロギング形式にかかわらず、このようなテーブルへのすべての挿入は常にログが記録されます。更新と削除は、ステートメントベースまたは行ベースのどちらのロギングが使用されているかによって扱いが異なります。ステートメントベースロギング形式では、`BLACKHOLE` テーブルに影響するすべてのステートメントのログが記録されますが、それらの影響は無視されます。行ベースロギングを使用するときは、このようなテーブルへの更新と削除は単にスキップされ、バイナリログに書き込まれません。これが発生するたびに警告がログに記録されます。

このため、`BLACKHOLE` ストレージエンジンを使用してテーブルに複製するときは、`binlog_format` サーバー変数を `ROW` または `MIXED` ではなく `STATEMENT` に設定することをお勧めします。

17.5.1.3 レプリケーションと文字セット

次のことは、異なる文字セットを使用する MySQL サーバー間でのレプリケーションに適用されます。

- ソースにグローバル `character_set_server` 値とは異なる文字セットのデータベースがある場合は、データベースのデフォルト文字セットに暗黙的に依存しないように `CREATE TABLE` ステートメントを設計する必要があります。推奨される回避策は、`CREATE TABLE` ステートメントに明示的に文字セットと照合順序を指定することです。

17.5.1.4 レプリケーションと CHECKSUM TABLE

`CHECKSUM TABLE` は、テーブルの行の格納形式に依存する方法を使用して、行ごとに計算されるチェックサムを返します。MySQL バージョン間で記憶域形式が同じであることは保証されないため、アップグレード後にチェックサム値が変更される可能性があります。

17.5.1.5 CREATE SERVER、ALTER SERVER、および DROP SERVER のレプリケーション

使用されているバイナリロギング形式に関係なく、ステートメント `CREATE SERVER`、`ALTER SERVER`、および `DROP SERVER` はバイナリログに書き込まれません。

17.5.1.6 CREATE ... IF NOT EXISTS ステートメントのレプリケーション

MySQL は、さまざまな `CREATE ... IF NOT EXISTS` ステートメントが複製されるときにこれらの値を適用します。

- データベースがソースにすでに存在するかどうかにかかわらず、すべての `CREATE DATABASE IF NOT EXISTS` ステートメントがレプリケートされます。
- 同様に、テーブルがソースにすでに存在するかどうかに関係なく、`SELECT` のないすべての `CREATE TABLE IF NOT EXISTS` ステートメントがレプリケートされます。これは `CREATE TABLE IF NOT EXISTS ... LIKE` を含みます。`CREATE TABLE IF NOT EXISTS ... SELECT` のレプリケーションは、多少異なるルールに従います。詳細については、[セクション17.5.1.7「CREATE TABLE ... SELECT ステートメントのレプリケーション」](#)を参照してください。

- ステートメントで指定されたイベントがソースにすでに存在するかどうかにかかわらず、`CREATE EVENT IF NOT EXISTS` は常にレプリケートされます。

17.5.1.7 CREATE TABLE ... SELECT ステートメントのレプリケーション

MySQL では、`CREATE TABLE ... SELECT` ステートメントのレプリケート時に次のルールが適用されます:

- `CREATE TABLE ... SELECT` は常に暗黙的コミットを実行します ([セクション13.3.3「暗黙的なコミットを発生させるステートメント」](#))。
- 宛先テーブルが存在しない場合、ロギングは次のように行われます。 `IF NOT EXISTS` が存在するかどうかは重要ではありません。
 - `STATEMENT` または `MIXED` 形式: ステートメントは、書き込まれたものとしてログに記録されます。
 - `ROW` 形式: ステートメントは、`CREATE TABLE` ステートメントおよび一連の行挿入イベントとしてログに記録されます。

MySQL 8.0.21 より前は、ステートメントは 2 つのトランザクションとしてログに記録されます。 MySQL 8.0.21 の時点では、アトミック DDL をサポートするストレージエンジンでは、1 つのトランザクションとしてログに記録されます。 詳細は、[セクション13.1.1「アトミックデータ定義ステートメントのサポート」](#) を参照してください。

- `CREATE TABLE ... SELECT` ステートメントが失敗した場合、何も記録されません。 これには、宛先テーブルが存在し、`IF NOT EXISTS` が指定されていないケースが含まれます。
- 宛先テーブルが存在し、`IF NOT EXISTS` が指定されている場合、MySQL 8.0 はステートメントを完全に無視します。 何も挿入または記録されません。

MySQL 8.0 では、`CREATE TABLE ... SELECT` ステートメントで、ステートメントによって作成されたテーブル以外のテーブルを変更することはできません。

17.5.1.8 CURRENT_USER() のレプリケーション

次のステートメントは、影響を受けるユーザーまたは定義者の名前 (場合によってはホスト) のかわりに `CURRENT_USER()` 関数の使用をサポートしています:

- `DROP USER`
- `RENAME USER`
- `GRANT`
- `REVOKE`
- `CREATE FUNCTION`
- `CREATE PROCEDURE`
- `CREATE TRIGGER`
- `CREATE EVENT`
- `CREATE VIEW`
- `ALTER EVENT`
- `ALTER VIEW`
- `SET PASSWORD`

バイナリロギングが有効で、これらのステートメントのいずれかで `CURRENT_USER()` または `CURRENT_USER` が定義者として使用されている場合、MySQL Server はステートメントがレプリケートされるときに、ソースとレプリカの両方の同じユーザーにステートメントが適用されることを確認します。 場合によっては、パスワードを変更す

るステートメントなど、ステートメントにユーザー名が含まれるように、関数参照がバイナリログに書き込まれる前に展開されます。他のすべての場合、ソースの現在のユーザーの名前はメタデータとしてレプリカにレプリケートされ、レプリカはレプリカの現在のユーザーではなく、メタデータで指定された現在のユーザーにステートメントを適用します。

17.5.1.9 ソースとレプリカで異なるテーブル定義を使用したレプリケーション

レプリケーションのソースおよびターゲットテーブルは同じである必要はありません。ソース上のテーブルには、テーブルのレプリカコピーより多いカラムまたは少ないカラムを含めることができます。また、ソースおよびレプリカ上の対応するテーブルのカラムでは、特定の条件に応じて異なるデータ型を使用できます。

注記

パーティション化が異なるテーブル間のレプリケーションはサポートされていません。 [セクション17.5.1.24「レプリケーションおよびパーティション化」](#)を参照してください。

ソーステーブルとターゲットテーブルの定義が同じではない場合、データベースとテーブルの名前はソースとレプリカの両方で同じである必要があります。次の2つのセクションで、追加条件について例を示して説明します。

ソースまたはレプリカにカラムが多いレプリケーション

テーブルのソースコピーとレプリカコピーのカラム数が異なるように、次の条件に従ってソースからレプリカにテーブルをレプリケートできます:

- 両方のバージョンのテーブルに共通するカラムは、ソースとレプリカで同じ順序で定義する必要があります。(これは、両方のテーブルのカラム数が同じ場合でも当てはまります。)
- 両方のバージョンのテーブルに共通するカラムは、追加カラムの前に定義する必要があります。

これは、次の例に示すように、両方のテーブルに共通するカラムの範囲内のテーブルに新しいカラムが挿入されるレプリカで `ALTER TABLE` ステートメントを実行すると、レプリケーションが失敗することを意味します:

ソースおよびレプリカに存在するテーブル `t` が、次の `CREATE TABLE` ステートメントによって定義されているとします:

```
CREATE TABLE t (  
  c1 INT,  
  c2 INT,  
  c3 INT  
);
```

ここに示す `ALTER TABLE` ステートメントがレプリカで実行されるとします:

```
ALTER TABLE t ADD COLUMN cnew1 INT AFTER c3;
```

以前の `ALTER TABLE` はレプリカで許可されています。これは、両方のバージョンのテーブル `t` に共通のカラム `c1`、`c2` および `c3` が、異なるカラムの前に両方のバージョンのテーブルでグループ化されたままであるためです。

ただし、レプリケーションを中断させないがぎり、次の `ALTER TABLE` ステートメントはレプリカで実行できません:

```
ALTER TABLE t ADD COLUMN cnew2 INT AFTER c2;
```

新しいカラム `cnew2` は両方のバージョンの `t` に共通のカラムの間にあるため、示されている `ALTER TABLE` ステートメントのレプリカでの実行後にレプリケーションが失敗します。

- カラム数の多いバージョンのテーブルの「追加」カラムごとに、デフォルト値が必要です。

カラムのデフォルト値は、その型、`DEFAULT` オプションで定義されているかどうか、`NULL` として宣言されているかどうか、作成時に有効であったサーバー SQL モードなど、いくつかの要因で決まります。詳細については、[セクション11.6「データ型デフォルト値」](#)を参照してください)。

また、テーブルのレプリカコピーにソースコピーよりも多くのカラムがある場合、テーブルに共通する各カラムは両方のテーブルで同じデータ型を使用する必要があります。

例。 次の例は、有効および無効なテーブル定義をいくつか示します。

ソースのその他のカラム。 次のテーブル定義は有効で、正しく複製されます。

```
source> CREATE TABLE t1 (c1 INT, c2 INT, c3 INT);
replica> CREATE TABLE t1 (c1 INT, c2 INT);
```

次のテーブル定義では、両方のバージョンのテーブルに共通するカラムの定義がソースとは異なる順序でレプリカ上にあるため、エラーが発生します：

```
source> CREATE TABLE t1 (c1 INT, c2 INT, c3 INT);
replica> CREATE TABLE t1 (c2 INT, c1 INT);
```

次のテーブル定義では、両方のバージョンのテーブルに共通するカラムの定義の前にソースの追加カラムの定義が表示されるため、エラーが発生します：

```
source> CREATE TABLE t1 (c3 INT, c1 INT, c2 INT);
replica> CREATE TABLE t1 (c1 INT, c2 INT);
```

レプリカのカラムが増えます。 次のテーブル定義は有効で、正しく複製されます。

```
source> CREATE TABLE t1 (c1 INT, c2 INT);
replica> CREATE TABLE t1 (c1 INT, c2 INT, c3 INT);
```

次の定義では、両方のバージョンのテーブルに共通するカラムがソースとレプリカの両方で同じ順序で定義されていないため、エラーが発生します：

```
source> CREATE TABLE t1 (c1 INT, c2 INT);
replica> CREATE TABLE t1 (c2 INT, c1 INT, c3 INT);
```

レプリカバージョンのテーブルの追加カラムの定義は、両方のバージョンのテーブルに共通するカラムの定義の前に表示されるため、次のテーブル定義でもエラーが発生します：

```
source> CREATE TABLE t1 (c1 INT, c2 INT);
replica> CREATE TABLE t1 (c3 INT, c1 INT, c2 INT);
```

次のテーブル定義は、レプリカバージョンのテーブルにソースバージョンと比較して追加のカラムがあり、2つのバージョンのテーブルで共通カラム `c2` に異なるデータ型が使用されているため、失敗します：

```
source> CREATE TABLE t1 (c1 INT, c2 BIGINT);
replica> CREATE TABLE t1 (c1 INT, c2 INT, c3 INT);
```

データ型が異なるカラムのレプリケーション

ソース上の対応するカラムと同じテーブルのレプリカコピーは、同じデータ型であることが理想的です。ただし、特定の条件が満たされているかぎり、これは必ずしも厳密には強制されません。

通常、特定のデータ型のカラムから、同じサイズまたは幅 (該当する場合) またはそれ以上の同じ型の別のカラムにレプリケートできます。たとえば、`CHAR(10)` カラムから別の `CHAR(10)` に、または `CHAR(10)` カラムから `CHAR(25)` カラムに、問題なく複製できます。場合によっては、あるデータ型 (ソース上) を持つカラムから (レプリカ上の) 異なるデータ型を持つカラムにレプリケートすることもできます。ソースバージョンのカラムのデータ型がレプリカ上で同じサイズ以上の型に昇格される場合、これは属性プロモーションと呼ばれます。

属性昇格は、ステートメントベースと行ベースの両方のレプリケーションで使用でき、ソースまたはレプリカで使用されるストレージエンジンには依存しません。ただし、ロギング形式の選択は許可される型変換に影響します。詳細は、このセクションで後述します。

重要

ステートメントベースまたは行ベースのレプリケーションのどちらを使用する場合でも、属性昇格を採用する場合は、テーブルのレプリカコピーにソースコピーより多くのカラムを含めることはできません。

ステートメントベースのレプリケーション。 ステートメントベースレプリケーションを使用する場合、従うべき簡単な経験則は、「ソースで実行されたステートメントもレプリカで正常に実行される場合は、正常にレプリケート

する必要もあります」」です。つまり、レプリカ上の特定のカラムの型と互換性のある値がステートメントで使用されている場合は、そのステートメントをレプリケートできます。たとえば、**TINYINT** カラムに収まる任意の値を **BIGINT** カラムに挿入することもできます。テーブルのレプリカコピーの **TINYINT** カラムのタイプを **BIGINT** に変更した場合でも、成功したソースのそのカラムへの挿入もレプリカで成功する必要があります。これは、**BIGINT** カラムを超える有効な **TINYINT** 値を持つことができないためです。

行ベースレプリケーション: 属性の昇格と降格. 行ベースのレプリケーションでは、小さいデータ型と大きいデータ型間の属性昇格および降格がサポートされています。このセクションで後述するように、降格されるカラム値の不可逆 (切り捨て) または非不可逆変換を許可するかどうかを指定することもできます。

不可逆および非不可逆変換. ターゲット型が挿入される値を表現できない場合、変換をどのように扱うかについての決定が必要になります。変換を許可するけれども、ターゲットカラムで「適合」を実現するためにソース値を切り捨てる (または変更する) 場合、不可逆変換と呼ばれることを行います。ソースカラム値をターゲットカラムに適合させるために切り捨てまたは同様の変更を必要としない変換は、非不可逆変換です。

型変換モード. `slave_type_conversions` グローバルサーバー変数の設定によって、レプリカで使用される型変換モードが制御されます。この変数は、レプリカタイプ変換動作に対する各モードの影響を説明する次のリストから一連の値を取得します:

<code>ALL_LOSSY</code>	このモードでは、情報の損失を意味する型変換が許可されます。 これは非不可逆変換が許可されることを暗示せず、不可逆変換を必要とするまたは変換をまったく必要としないケースのみが許可されることだけを暗示します。たとえば、このモードのみを有効にした場合、 INT カラムが TINYINT に変換されること (不可逆変換) は許可されますが、 TINYINT カラムが INT カラムに変換されること (非不可逆) は許可されません。この場合、後者の変換を試みると、レプリカでエラーが発生してレプリケーションが停止します。
<code>ALL_NON_LOSSY</code>	このモードは、ソース値の切り捨てまたは特別処理を必要としない変換を許可します。すなわち、ターゲット型の範囲がソース型より広い変換を許可します。 このモードを設定することは、不可逆変換が許可されるかどうかに関係ありません。これは <code>ALL_LOSSY</code> モードで制御されます。 <code>ALL_NON_LOSSY</code> のみが設定され、 <code>ALL_LOSSY</code> は設定されていない場合、データ (INT から TINYINT 、 CHAR(25) から VARCHAR(20) など) が失われる原因となる変換を試みると、レプリカはエラーで停止します。
<code>ALL_LOSSY,ALL_NON_LOSSY</code>	このモードが設定されると、サポートされるすべての型変換が、不可逆変換かどうかにかかわらず、許可されます。
<code>ALL_SIGNED</code>	昇格される整数型を符号付き値として扱います (デフォルト動作)。
<code>ALL_UNSIGNED</code>	昇格される整数型を符号なし値として扱います。
<code>ALL_SIGNED,ALL_UNSIGNED</code>	昇格される整数型を、可能な場合符号付きとして、そうでない場合は符号なしとして扱います。
[empty]	<code>slave_type_conversions</code> が設定されていないときは、属性の昇格または降格は許可されません。これは、ソースおよびターゲットテーブル内のすべてのカラムが同じ型である必要があることを意味します。 このモードがデフォルトです。

整数型が昇格されるときに、符号ありか符号なしかは保持されません。デフォルトでは、レプリカはこのような値をすべて符号付きとして扱います。この動作は、`ALL_SIGNED`、`ALL_UNSIGNED`、またはその両方を使用して制御できます。`ALL_SIGNED` は、昇格されたすべての整数型を符号付きとして処理するようにレプリカに指示します。`ALL_UNSIGNED` は、これらを符号なしとして処理するように指示します。両方を指定すると、レプリカは可能であれば値を符号付きとして処理し、それ以外の場合は符号なしとして処理します。リストされる順序は重要ではありません。少なくとも `ALL_LOSSY` または `ALL_NONLOSSY` のいずれかが使用されていない場合は、`ALL_SIGNED` も `ALL_UNSIGNED` も効果を持ちません。

タイプ変換モードを変更するには、新しい `slave_type_conversions` 設定でレプリカを再起動する必要があります。

サポートされる変換。 違うけれども似ているデータ型の間でサポートされる変換を次のリストに示します。

- 整数型 `TINYINT`、`SMALLINT`、`MEDIUMINT`、`INT`、および `BIGINT` のいずれかの間。

これには、これらの型の符号付きおよび符号なしバージョンの間の変換が含まれます。

不可逆変換は、ソース値をターゲットカラムで許可される最大値(または最小値)に切り捨てることで行われます。符号なし型から符号付き型への変換時に非可逆変換を保証するには、ターゲットカラムがソースカラムの値の範囲を収容できる十分な大きさである必要があります。たとえば、`TINYINT UNSIGNED` は、非可逆に `SMALLINT` に降格できますが、`TINYINT` にはできません。

- 小数点型 `DECIMAL`、`FLOAT`、`DOUBLE`、および `NUMERIC` のいずれかの間。

`FLOAT` から `DOUBLE` へは非不可逆変換です。`DOUBLE` から `FLOAT` へは不可逆にしか扱えません。 $D' \geq D$ および $(M'-D') \geq (M-D)$ が非可逆的である `DECIMAL(M,D)` から `DECIMAL(M',D')` への変換。 $M' < M$ 、 $D' < D$ 、またはその両方の場合、不可逆変換のみ実行できます。

いずれかの小数点型の場合、格納される値をターゲット型に適合させることができない場合は、このマニュアルのほかの場所でサーバーに定義される丸めルールに従って値が切り捨てられます。小数点型でこれがどのように実行されるかについては、[セクション12.25.4「丸め動作」](#)を参照してください。

- 文字列型 `CHAR`、`VARCHAR`、および `TEXT` のいずれかの間(異なる幅の間での変換を含む)。

`CHAR`、`VARCHAR`、または `TEXT` から、同じまたはそれより大きいサイズの `CHAR`、`VARCHAR`、または `TEXT` カラムへの変換は、決して不可逆ではありません。不可逆変換は、レプリカに文字列の最初の `N` 文字のみを挿入することで処理されます。ここで、`N` はターゲットカラムの幅です。

重要

異なる文字セットを使用するカラム間のレプリケーションはサポートされません。

- バイナリデータ型 `BINARY`、`VARBINARY`、および `BLOB` のいずれかの間(異なる幅の間での変換を含む)。

`BINARY`、`VARBINARY`、または `BLOB` から、同じまたはそれより大きいサイズの `BINARY`、`VARBINARY`、または `BLOB` カラムへの変換は、決して不可逆ではありません。不可逆変換は、レプリカに文字列の最初の `N` バイトのみを挿入することで処理されます。ここで、`N` はターゲットカラムの幅です。

- 任意の2つのサイズの任意の2つの `BIT` カラムの間。

`BIT(M)` カラムからの値を `BIT(M')` カラムに挿入するときは $(M' > M)$ 、`BIT(M')` カラムの最上位ビットがクリアされ(ゼロに設定され)、`BIT(M)` 値の `M` ビットが `BIT(M')` カラムの最下位ビットとして設定されます。

ソース `BIT(M)` カラムからの値をターゲット `BIT(M')` カラムに挿入するときは $(M' < M)$ 、`BIT(M')` カラムの可能な最大値が割り当てられます。つまり、「すべてが設定された」値がターゲットカラムに割り当てられます。

前のリストにない型の間の変換は許可されません。

17.5.1.10 レプリケーションと DIRECTORY テーブルオプション

ソースサーバーの `CREATE TABLE` ステートメントで `DATA DIRECTORY` または `INDEX DIRECTORY` テーブルオプションが使用されている場合は、レプリカでもテーブルオプションが使用されます。これにより、対応するディレクトリがレプリカホストファイルシステムに存在しない場合、または存在するがレプリカ MySQL サーバーにアクセスできない場合に問題が発生する可能性があります。これをオーバーライドするには、レプリカで `NO_DIR_IN_CREATE` サーバーの SQL モードを使用します。これにより、`CREATE TABLE` ステートメントのレプリケート時にレプリカで `DATA DIRECTORY` および `INDEX DIRECTORY` テーブルオプションが無視されます。その結果、テーブルのデータベースディレクトリ内に `MyISAM` データおよびインデックスファイルが作成されます。

詳細は、[セクション5.1.11「サーバー SQL モード」](#)を参照してください。

17.5.1.11 DROP ... IF EXISTS ステートメントのレプリケーション

削除するデータベース、テーブルまたはビューがソースに存在しない場合でも、`DROP DATABASE IF EXISTS`、`DROP TABLE IF EXISTS` および `DROP VIEW IF EXISTS` ステートメントは常にレプリケートされます。

これは、レプリカがソースでキャッチアップされた後、削除するオブジェクトがソースまたはレプリカに存在しなくなるようにするためです。

ストアドプログラム (ストアドプロシージャとストアドファンクション、トリガーおよびイベント) の `DROP ... IF EXISTS` ステートメントも、削除するストアドプログラムがソースに存在しない場合でもレプリケートされます。

17.5.1.12 レプリケーションと浮動小数点値

ステートメントベースレプリケーションでは、値は 10 進からバイナリに変換されます。それらの表現を 10 進とバイナリの間で変換すると近似値になる場合があるため、浮動小数点値を含む比較が不正確になります。これは、浮動小数点値を明示的に使用したり、浮動小数点に暗黙的に変換された値を使用したりする演算に当てはまります。浮動小数点値を比較すると、コンピュータアーキテクチャ、MySQL の構築に使用されるコンパイラなどが異なるため、ソースサーバーとレプリカサーバーで異なる結果が生じる可能性があります。 [セクション12.3「式評価での型変換」](#) および [セクションB.3.4.8「浮動小数点値に関する問題」](#) を参照してください。

17.5.1.13 レプリケーションと FLUSH

レプリカにレプリケートすると問題が発生する可能性があるため、一部の形式の `FLUSH` ステートメントはログに記録されません: `FLUSH LOGS` および `FLUSH TABLES WITH READ LOCK`。構文例は、[セクション13.7.8.3「FLUSH ステートメント」](#) を参照してください。 `FLUSH TABLES`, `ANALYZE TABLE`, `OPTIMIZE TABLE` および `REPAIR TABLE` ステートメントはバイナリログに書き込まれるため、複製に複製されます。これらのステートメントはテーブルデータを変更しないため、通常は問題ではありません。

ただし、ある状況では、この動作が問題になる場合があります。 `mysql` データベースで権限テーブルをレプリケートし、`GRANT` を使用せずにそれらのテーブルを直接更新する場合は、レプリカで `FLUSH PRIVILEGES` を発行して新しい権限を有効にする必要があります。また、`MERGE` テーブルの一部である `MyISAM` テーブルの名前を変更するときに `FLUSH TABLES` を使用する場合は、レプリカに対して `FLUSH TABLES` を手動で発行する必要があります。 `NO_WRITE_TO_BINLOG` またはそのエイリアスの `LOCAL` を指定しない場合、これらのステートメントはバイナリログに書き込まれます。

17.5.1.14 レプリケーションとシステム関数

一部の関数は条件によっては適切に複製されません。

- `USER()`、`CURRENT_USER()` (または `CURRENT_USER`)、`UUID()`、`VERSION()` および `LOAD_FILE()` 関数は変更なしでレプリケートされるため、行ベースのレプリケーションが有効になっていないかぎり、レプリカで確実に動作しません。([セクション17.2.1「レプリケーション形式」](#) を参照してください。)

`USER()` および `CURRENT_USER()` は、`MIXED` モード使用時に行ベースレプリケーションを使用して自動的に複製され、`STATEMENT` モードでは警告を生成します。([セクション17.5.1.8「CURRENT_USER\(\) のレプリケーション」](#) も参照してください。) これは、`VERSION()` および `RAND()` にも当てはまります。

- `NOW()` の場合、バイナリログはタイムスタンプを含みます。これは、値ソースでこの関数を呼び出したときに返されますがレプリカにレプリケートされることを意味します。異なるタイムゾーンの MySQL サーバー間でレプリケートするときに予期しない結果が発生しないようにするには、ソースとレプリカの両方でタイムゾーンを設定します。詳細は、[セクション17.5.1.33「レプリケーションとタイムゾーン」](#) を参照してください。

異なるタイムゾーンのサーバー間でレプリケートする際の潜在的な問題を説明するために、ソースがニューヨークにあり、レプリカがストックホルムにあり、両方のサーバーがローカル時間を使用しているとします。さらに、次に示すように、ソースでテーブル `mytable` を作成し、このテーブルに対して `INSERT` ステートメントを実行し、テーブルから選択するとします:

```
mysql> CREATE TABLE mytable (mycol TEXT);
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO mytable VALUES ( NOW() );
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM mytable;
+-----+
| mycol |
+-----+
| 2009-09-01 12:00:00 |
```



```
+-----+
1 row in set (0.00 sec)
```

ストックホルムの現地時間はニューヨークより 6 時間遅れているため、その瞬間にレプリカで `SELECT NOW()` を発行すると、値 `2009-09-01 18:00:00` が返されます。このため、表示された `CREATE TABLE` および `INSERT` ステートメントがレプリケートされた後に、`mytable` のレプリカコピーから選択した場合、`mycol` に `2009-09-01 18:00:00` という値が含まれている可能性があります。ただし、これは当てはまりません。`mytable` のレプリカコピーから選択すると、ソースとまったく同じ結果が得られます:

```
mysql> SELECT * FROM mytable;
+-----+
| mycol |
+-----+
| 2009-09-01 12:00:00 |
+-----+
1 row in set (0.00 sec)
```

`SYSDATE()` 関数は、`NOW()` とは異なり、レプリケーションに安全ではありません。バイナリログ内で `SET TIMESTAMP` ステートメントに影響されず、ステートメントベースロギングが使用される場合は非決定的であるためです。行ベースロギングを使用する場合は、これは問題ではありません。

ほかの方法は `--sysdate-is-now` オプションを使用することで、`SYSDATE()` が `NOW()` のエイリアスになります。これは、ソースおよびレプリカで正しく機能するために必要です。このような場合でも、この関数によって警告が発行されますが、`--sysdate-is-now` がソースとレプリカの両方で使用されているかぎり、無視しても問題ありません。

`SYSDATE()` は、`MIXED` モードの使用時に行ベースのレプリケーションを使用して自動的にレプリケートされ、`STATEMENT` モードで警告を生成します。

セクション 17.5.1.33 「レプリケーションとタイムゾーン」も参照してください。

- 次の制限は、ステートメントベースレプリケーションにのみ適用され、行ベースレプリケーションには適用されません。ユーザーレベルのロックを処理する `GET_LOCK()`、`RELEASE_LOCK()`、`IS_FREE_LOCK()` および `IS_USED_LOCK()` 関数は、ソース上の同時実行性コンテキストを認識していないレプリカでレプリケートされません。したがって、レプリカ上のコンテンツが異なるため、これらの関数を使用してソーステーブルに挿入しないでください。たとえば、`INSERT INTO mytable VALUES(GET_LOCK(...))` などのステートメントを発行しないでください。

これらの関数は、`MIXED` モード使用時に行ベースレプリケーションを使用して自動的に複製され、`STATEMENT` モードで警告を生成します。

ステートメントベースレプリケーションが有効のときに前述の制限に対する回避策として、問題のある関数結果をユーザー変数に保存して、後続のステートメントでその変数を参照する方法を使用できます。たとえば、次の単一行 `INSERT` は、`UUID()` 関数を参照するため問題があります。

```
INSERT INTO t VALUES(UUID());
```

この問題を回避するには、代わりにこれを実行してください。

```
SET @my_uuid = UUID();
INSERT INTO t VALUES(@my_uuid);
```

このステートメントの連続は複製されます。`@my_uuid` の値が `INSERT` ステートメントの前にユーザー変数イベントとしてバイナリログに格納されて `INSERT` で使用できるためです。

同じ概念が複数行挿入に適用されますが、使用するのが面倒です。2 行挿入の場合、このようにできます。

```
SET @my_uuid1 = UUID(); @my_uuid2 = UUID();
INSERT INTO t VALUES(@my_uuid1),(@my_uuid2);
```

ただし、行数が多いか不明の場合、この回避策は困難であるか実用的ではありません。たとえば、次のステートメントを個々のユーザー変数が各行に関連付けられているものに変換することはできません。

```
INSERT INTO t2 SELECT UUID(), * FROM t1;
```

ストアードファンクション内で、`RAND()` は、関数の実行中に 1 回だけ呼び出されるかぎり、正しく複製されます。(関数実行タイムスタンプおよび乱数シードは、ソースとレプリカで同一の暗黙的な入力とみなすことができます。)

`FOUND_ROWS()` と `ROW_COUNT()` 関数がステートメントベースレプリケーションを使用して複製されるときは、信頼性がありません。回避策は、関数呼び出しの結果をユーザー変数に格納してから、`INSERT` ステートメントでこれを使用することです。たとえば、`mytable` という名前のテーブルに結果を格納する場合は、普通は次のように実行するかもしれません。

```
SELECT SQL_CALC_FOUND_ROWS FROM mytable LIMIT 1;
INSERT INTO mytable VALUES( FOUND_ROWS() );
```

しかし、`mytable` を複製する場合は、次のように `SELECT ... INTO` を使用してから変数をテーブルに格納することをお勧めします。

```
SELECT SQL_CALC_FOUND_ROWS INTO @found_rows FROM mytable LIMIT 1;
INSERT INTO mytable VALUES(@found_rows);
```

このように、ユーザー変数はコンテキストの一部としてレプリケートされ、レプリカに正しく適用されます。

これらの関数は、`MIXED` モード使用時に行ベースレプリケーションを使用して自動的に複製され、`STATEMENT` モードで警告を生成します。(Bug #12092、Bug #30244)

17.5.1.15 レプリケーションと小数秒サポート

MySQL 8.0 では、最大マイクロ秒 (6 桁) の精度で `TIME`、`DATETIME` および `TIMESTAMP` 値の小数秒が許可されません。セクション11.2.6「時間値での小数秒」を参照してください。

17.5.1.16 呼び出される機能のレプリケーション

ユーザー定義関数 (UDF) やストアードプログラム (ストアードプロシージャと関数、トリガー、およびイベント) などの呼び出される機能のレプリケーションには、次の特徴があります。

- 機能の影響は常に複製されます。
- 次のステートメントはステートメントベースレプリケーションを使用して複製されます。
 - `CREATE EVENT`
 - `ALTER EVENT`
 - `DROP EVENT`
 - `CREATE PROCEDURE`
 - `DROP PROCEDURE`
 - `CREATE FUNCTION`
 - `DROP FUNCTION`
 - `CREATE TRIGGER`
 - `DROP TRIGGER`

ただし、これらのステートメントを使用して作成、変更、または削除される機能の影響は、行ベースレプリケーションを使用して複製されます。

注記

呼び出される機能をステートメントベースレプリケーションを使用して複製しようとする、警告が生成されます: `Statement is not safe to log in statement format`。たとえば、ステートメントベースレプリケーションで UDF を複製しようとする、MySQL サーバーは現在 UDF が決定的かどうかを判断できないため、この警告が生成されます。呼び出される機能の影響が決定的であることを確実にわかっている場合は、このような警告を安全に無視できます。

- `CREATE EVENT` および `ALTER EVENT` の場合:

- イベントのステータスは、指定された状態に関係なくレプリカ上で `SLAVESIDE_DISABLED` に設定されます (これは `DROP EVENT` には適用されません)。
- イベントが作成されたソースは、レプリカ上でそのサーバー ID によって識別されます。
`INFORMATION_SCHEMA.EVENTS` の `ORIGINATOR` カラムには、この情報が格納されます。詳細は、[セクション26.14「INFORMATION_SCHEMA EVENTS テーブル」](#) および [セクション13.7.7.18「SHOW EVENTS ステートメント」](#) を参照してください。
- 機能実装は、ソースに障害が発生した場合にイベント処理を失うことなくレプリカをソースとして使用できるように、更新可能な状態のレプリカに存在します。

(ソースとして機能していた) 別のサーバーで作成されたスケジュール済イベントが MySQL サーバーにあるかどうかを確認するには、次に示すような方法で `INFORMATION_SCHEMA.EVENTS` テーブルをクエリーします:

```
SELECT EVENT_SCHEMA, EVENT_NAME
FROM INFORMATION_SCHEMA.EVENTS
WHERE STATUS = 'SLAVESIDE_DISABLED';
```

また、`SHOW EVENTS` ステートメントを次のように使用できます。

```
SHOW EVENTS
WHERE STATUS = 'SLAVESIDE_DISABLED';
```

このようなイベントを持つレプリカをソースに昇格する場合は、`ALTER EVENT event_name ENABLE` を使用して各イベントを有効にする必要があります。ここで、`event_name` はイベントの名前です。

このレプリカでのイベントの作成に複数のソースが関与しており、サーバー ID `source_id` を持つ特定のソースでのみ作成されたイベントを識別する場合は、次に示すように、`EVENTS` テーブルで前のクエリーを変更して `ORIGINATOR` カラムを含めます:

```
SELECT EVENT_SCHEMA, EVENT_NAME, ORIGINATOR
FROM INFORMATION_SCHEMA.EVENTS
WHERE STATUS = 'SLAVESIDE_DISABLED'
AND ORIGINATOR = 'source_id'
```

同じような方法で `SHOW EVENTS` ステートメントで `ORIGINATOR` を使用できます。

```
SHOW EVENTS
WHERE STATUS = 'SLAVESIDE_DISABLED'
AND ORIGINATOR = 'source_id'
```

ソースからレプリケートされたイベントを有効にする前に、(`SET GLOBAL event_scheduler = OFF;`などのステートメントを使用して) レプリカで MySQL イベントスケジューラを無効にし、必要な `ALTER EVENT` ステートメントを実行してサーバーを再起動してから、(`SET GLOBAL event_scheduler = ON;`などのステートメントを使用して) レプリカでイベントスケジューラを再度有効にする必要があります

後で新しいソースをレプリカに後退させる場合は、`ALTER EVENT` ステートメントで有効になっているすべてのイベントを手動で無効にする必要があります。これは、前に示した `SELECT` ステートメントからのイベントの名前を別個のテーブルに格納するか、`ALTER EVENT` ステートメントを使用してイベントを識別する共通プリフィクス (`replicated_` など) でそれらの名前を変更することで、行うことができます。

イベントの名前を変更した場合、このサーバーをレプリカに後退させるときに、次に示すように `EVENTS` テーブルをクエリーしてイベントを識別できます:

```
SELECT CONCAT(EVENT_SCHEMA, '.', EVENT_NAME) AS 'Db.Event'
FROM INFORMATION_SCHEMA.EVENTS
WHERE INSTR(EVENT_NAME, 'replicated_') = 1;
```

17.5.1.17 JSON ドキュメントのレプリケーション

MySQL 8.0 より前は、JSON カラムへの更新は常に完全なドキュメントとしてバイナリログに書き込まれていました。MySQL 8.0 では、JSON ドキュメントへの部分的な更新をログに記録できます ([JSON 値の部分更新](#) を参照)。これはより効率的です。ロギングの動作は、次に説明するように、使用する形式によって異なります:

ステートメントベースのレプリケーション。JSON 部分更新は、常に部分更新として記録されます。ステートメントベースのロギングを使用している場合は、これを無効にできません。

行ベースのレプリケーション。JSON 部分更新は、デフォルトではログに記録されませんが、完全なドキュメントとして記録されます。部分更新のロギングを有効にするには、`binlog_row_value_options=PARTIAL_JSON` を設定します。レプリケーションソースにこの変数が設定されている場合、そのソースから受信された部分更新は、その変数のレプリカ自体の設定に関係なく、レプリカによって処理および適用されます。

MySQL 8.0.2 以前を実行しているサーバーは、JSON 部分更新に使用されるログイベントを認識しません。このため、MySQL 8.0.3 以降を実行しているサーバーからこのようなサーバーにレプリケートする場合は、この変数を" (空の文字列) に設定して、ソースで `binlog_row_value_options` を無効にする必要があります。詳細は、この変数の説明を参照してください。

17.5.1.18 レプリケーションと LIMIT

`DELETE`、`UPDATE`、および `INSERT ... SELECT` ステートメント内の `LIMIT` 句のステートメントベースレプリケーションは、影響を受ける行の順序が未定義のため、安全ではありません。(このようなステートメントは、`ORDER BY` 句も含んでいる場合にのみ、ステートメントベースレプリケーションで正しく複製できます。)このようなステートメントに遭遇したときは:

- `STATEMENT` モード使用時は、このステートメントがステートメントベースレプリケーションで安全でないという警告が発行されるようになりました。

`STATEMENT` モードを使用している場合、`LIMIT` を含む DML ステートメントには、`ORDER BY` 句も含まれていても警告が発行されます (決定論的に行われます)。これは既知の問題です。(Bug #42851)

- `MIXED` モード使用時は、このステートメントは行ベースモードを使用して自動的に複製されるようになりました。

17.5.1.19 レプリケーションと LOAD DATA

`LOAD DATA` はステートメントベースのロギングで安全でないとみなされます (を参照してください [セクション 17.2.1.3 「バイナリロギングでの安全および安全でないステートメントの判断」](#))。 `binlog_format=MIXED` が設定されている場合、ステートメントは行ベースの形式で記録されます。 `binlog_format=STATEMENT` が設定されている場合、他の安全でないステートメントとは異なり、`LOAD DATA` では警告が生成されないことに注意してください。

`binlog_format=STATEMENT` が設定されているときに `LOAD DATA` を使用すると、変更が適用されるレプリカにデータを含む一時ファイルが作成されます。次に、レプリカは `LOAD DATA INFILE` ステートメントを使用して変更を適用します。バイナリログの暗号化がサーバー上でアクティブな場合、この一時ファイルは暗号化されないことに注意してください。暗号化が必要な場合は、一時ファイルを作成しない行ベースまたは混合バイナリロギング形式を使用してください。

レプリケーションチャネルの保護に `PRIVILEGE_CHECKS_USER` アカウントが使用されている場合 ([セクション 17.3.3 「レプリケーション権限チェック」](#) を参照)、行ベースのバイナリロギング (`binlog_format=ROW`) を使用して `LOAD DATA` 操作をログに記録することを強くお勧めします。チャネルに `REQUIRE_ROW_FORMAT` が設定されている場合は、行ベースのバイナリロギングが必要です。このロギング形式では、イベントの実行に `FILE` 権限は必要ないため、`PRIVILEGE_CHECKS_USER` アカウントにこの権限を付与しないでください。ステートメント形式で記録された `LOAD DATA INFILE` 操作に関連するレプリケーションエラーからリカバリする必要があり、レプリケートされたイベントが信頼できる場合は、`FILE` 権限を `PRIVILEGE_CHECKS_USER` アカウントに一時的に付与し、レプリケートされたイベントの適用後に削除できます。

`mysqlbinlog` がステートメントベースの形式で記録された `LOAD DATA` ステートメントのログイベントを読み取ると、生成されたローカルファイルが一時ディレクトリに作成されます。これらの一時ファイルは、`mysqlbinlog` およびその他のどの MySQL プログラムによっても自動的に削除されません。ステートメントベースのバイナリロギングで `LOAD DATA` ステートメントを使用する場合は、ステートメントログが不要になったあとに一時ファイルを自分で削除するようにしてください。詳細は、[セクション 4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」](#) を参照してください。

17.5.1.20 レプリケーションと max_allowed_packet

`max_allowed_packet` は、MySQL サーバーとクライアント間の単一のメッセージ (レプリカを含む) のサイズに上限を設定します。大規模なカラム値 (`TEXT` または `BLOB` カラムにある可能性がある) をレプリケートしていて、`max_allowed_packet` がソース上で小さすぎる場合、ソースはエラーで失敗し、レプリカはレプリケーション I/O スレッドを停止します。レプリカ上の `max_allowed_packet` が小さすぎると、レプリカは I/O スレッドを停止します。

現在、行ベースのレプリケーションでは、更新によって実際に変更されなかったカラムの値を含め、更新された行のすべてのカラムおよびカラムの値がソースからレプリカに送信されます。これは、行ベースレプリケーションを使用して大きなカラム値を複製するときに、複製されるテーブル内でもっとも大きい行を格納できるだけの大きさに `max_allowed_packet` を設定するように気をつける必要があります (更新だけを複製したり、比較的小さい値だけを挿入したりする場合でも)。

マルチスレッドレプリカ (`slave_parallel_workers > 0` を使用) では、`slave_pending_jobs_size_max` システム変数がソースの `max_allowed_packet` システム変数の設定以上の値に設定されていることを確認します。`slave_pending_jobs_size_max` のデフォルト設定 128M は、`max_allowed_packet` のデフォルト設定の 64M の 2 倍です。`max_allowed_packet` は、ソースが送信できるパケットサイズを制限しますが、イベントヘッダーを追加すると、このサイズを超えるバイナリログイベントを生成できます。また、行ベースのレプリケーションでは、`max_allowed_packet` の値によってテーブルの各カラムのみが制限されるため、単一のイベントが `max_allowed_packet` サイズより大幅に大きくなる可能性があります。

レプリカは、`slave_max_allowed_packet` 設定で設定された制限までパケットを実際に受け入れます。これは、大きなパケットによるレプリケーションの失敗を防ぐために、デフォルトで最大設定の 1GB に設定されています。ただし、`slave_pending_jobs_size_max` の値によって、受信パケットを保持するためにレプリカで使用可能になるメモリが制御されます。指定されたメモリは、すべてのレプリカワーカーキュー間で共有されます。

`slave_pending_jobs_size_max` の値は弱い制限であり、非常に大きなイベント (1 つまたは複数のパケットで構成される) がこのサイズを超えると、トランザクションはすべてのレプリカワーカーが空のキューを持つまで保持されてから処理されます。後続のすべてのトランザクションは、大規模なトランザクションが完了するまで保持されます。そのため、`slave_pending_jobs_size_max` より大きい異常なイベントを処理できますが、すべてのレプリカワーカーのキューをクリアする遅延および後続のトランザクションをキューに入れる待機によってレプリカが遅れ、レプリカワーカーの同時実行性が低下する可能性があります。そのため、`slave_pending_jobs_size_max` は、予想されるほとんどのイベントサイズに対応できる高さに設定する必要があります。

17.5.1.21 レプリケーションと MEMORY テーブル

レプリケーションソースサーバーが停止して再起動すると、その `MEMORY` テーブルは空になります。この効果をレプリカにレプリケートするには、ソースが起動後に最初に特定の `MEMORY` テーブルを使用するときに、そのテーブルの `DELETE` ステートメントまたは (MySQL 8.0.22 から) `TRUNCATE TABLE` ステートメントをバイナリログに書き込むことによってテーブルを空にする必要があることをレプリカに通知するイベントをログに記録します。この生成されたイベントはバイナリログ内のコメントによって識別でき、GTID がサーバー上で使用されている場合は GTID が割り当てられます。バイナリロギング形式が `ROW` に設定されている場合でも、ステートメントは常にステートメント形式で記録され、`read_only` または `super_read_only` モードがサーバーで設定されていても書き込まれます。ソースの再起動からテーブルの最初の使用までの間、レプリカの `MEMORY` テーブルに古いデータが残っていることに注意してください。レプリカへの直接クエリーで失効したデータが返される可能性がある場合にこの間隔を回避するには、`init_file` システム変数を設定して、起動時にソースの `MEMORY` テーブルに移入するステートメントを含むファイルに名前を付けることができます。

レプリカサーバーを停止して再起動すると、その `MEMORY` テーブルは空になります。これにより、レプリカがソースと同期しなくなり、他の障害が発生したり、レプリカが停止する可能性があります:

- ソースから受信した行形式の更新および削除は、`Can't find record in 'memory_table'` で失敗する場合があります。
- `INSERT INTO ... SELECT FROM memory_table` などのステートメントによって、ソースおよびレプリカに異なる行セットが挿入される場合があります。

レプリカはまた、`DELETE` または (MySQL 8.0.22) `TRUNCATE TABLE` ステートメントを独自のバイナリログに書き込み、それがダウンストリームレプリカに渡されて、独自の `MEMORY` テーブルが空になります。

`MEMORY` テーブルをレプリケートしているレプリカを再起動する安全な方法は、まずソース上の `MEMORY` テーブルからすべての行を削除し、それらの変更がレプリカにレプリケートされるまで待機することです。その後、レプリカを安全に再起動できます。

場合によっては、別の再起動方法を適用できることがあります。`binlog_format=ROW` の場合、レプリカを再起動する前に `slave_exec_mode=IDEMPOTENT` を設定すると、レプリカが停止しないようにできます。これにより、レプリカはレプリケートを続行できますが、その `MEMORY` テーブルはソースのテーブルとは異なります。これは、`MEMORY` テーブルの内容が安全に失われるようにアプリケーションロジックが設定されている場合 (`MEMORY` テーブルがキャッシュに使用されている場合など) に許容されます。`slave_exec_mode=IDEMPOTENT` はすべての

テーブルにグローバルに適用されるため、`MEMORY` 以外のテーブルの他のレプリケーションエラーが非表示になる可能性があります。

(ここで説明した方法は NDB Cluster では適用できません。ここでは、`slave_exec_mode` は常に `IDEMPOTENT` であり、変更できません。)

`MEMORY` テーブルのサイズは、`max_heap_table_size` システム変数の値によって制限され、これは複製されません (セクション 17.5.1.39 「レプリケーションと変数」を参照してください)。`max_heap_table_size` での変更は、変更後に `ALTER TABLE ... ENGINE = MEMORY` または `TRUNCATE TABLE` を使用して作成または更新された `MEMORY` テーブルに、またはサーバー再起動後にすべての `MEMORY` テーブルに反映されます。レプリカで実行せずにソースでこの変数の値を増やすと、ソース上のテーブルがレプリカ上の対応するテーブルより大きくなり、ソースでは正常に挿入されますが、「テーブルがいっぱいです」エラーでレプリカでは失敗します。これは既知の問題です (Bug #48666)。このような場合は、レプリカおよびソースで `max_heap_table_size` のグローバル値を設定してから、レプリケーションを再起動する必要があります。また、ソースとレプリカの両方の MySQL サーバーを再起動して、それぞれに新しい値が完全 (グローバル) に有効になるようにすることをお勧めします。

`MEMORY` テーブルに関する詳細は、セクション 16.3 「`MEMORY` ストレージエンジン」を参照してください。

17.5.1.22 mysql システムスキーマのレプリケーション

`mysql` スキーマのテーブルに対して行われたデータ変更ステートメントは、`binlog_format` の値に従ってレプリケートされます。この値が `MIXED` の場合、これらのステートメントは行ベースの形式を使用してレプリケートされます。ただし、通常はこの情報を間接的に更新するステートメント (`GRANT`、`REVOKE`、およびトリガー、ストアルーチン、およびビューを操作するステートメント) は、ステートメントベースレプリケーションを使用してレプリカにレプリケートされます。

17.5.1.23 レプリケーションとクエリーオプティマイザ

データ変更が非決定的に行われるようにステートメントが書き込まれた場合、つまりクエリーオプティマイザを残した場合は、ソースとレプリカのデータが異なる可能性があります。(一般的に、これはレプリケーション以外であっても良い行動ではありません。) 非決定的なステートメントの例には、`ORDER BY` 句なしの `LIMIT` を使用する `DELETE` または `UPDATE` ステートメントが含まれます。これらの説明の詳細は、セクション 17.5.1.18 「レプリケーションと `LIMIT`」を参照してください。

17.5.1.24 レプリケーションおよびパーティション化

パーティションテーブル間のレプリケーションは、例外が特に許可されている場合を除き、同じパーティション化スキームを使用し、それ以外の場合は同じ構造を持つがざりサポートされます (セクション 17.5.1.9 「ソースとレプリカで異なるテーブル定義を使用したレプリケーション」を参照)。

通常、パーティション化が異なるテーブル間のレプリケーションはサポートされません。これは、このような場合にパーティションに直接作用するステートメント (`ALTER TABLE ... DROP PARTITION` など) がソースとレプリカで異なる結果を生成する可能性があるためです。テーブルがソースでパーティション化されているがレプリカではパーティション化されていない場合、レプリカのソースコピーでパーティションを操作するステートメントはレプリカで失敗します。テーブルのレプリカコピーがパーティション化されているが、ソースコピーがパーティション化されていない場合、パーティションに直接作用するステートメントは、そこでエラーが発生することなくソースで実行できません。レプリケーションを停止したり、ソースとレプリカ間の不整合を作成したりしないようにするには、必ずソースのテーブルとレプリカの対応するレプリケートテーブルが同じ方法でパーティション化されていることを確認します。

17.5.1.25 レプリケーションと REPAIR TABLE

破損または損傷したテーブルで使用されるとき、`REPAIR TABLE` ステートメントはレプリカでできない行を削除できます。ただし、このステートメントによって実行されるテーブルデータのこのような変更はレプリケートされないため、ソースとレプリカの同期が失われる可能性があります。このため、ソース上のテーブルが破損し、`REPAIR TABLE` を使用して修復する場合は、レプリケーションを停止してから (まだ実行中の場合)、`REPAIR TABLE` を使用する必要があります。その後、テーブルのソースコピーとレプリカコピーを比較し、矛盾を手動で修正する準備をしてから、レプリケーションを再起動してください。

17.5.1.26 レプリケーションと予約語

古いソースから新しいレプリカにレプリケートしようとしたときに、レプリカで実行されている新しい MySQL バージョンの予約語であるソースで識別子を使用すると、問題が発生する可能性があります。たとえば、MySQL 8.0 レプリカにレプリケートしている MySQL 5.7 ソース上の `rank` という名前のテーブルのカラムは、`RANK` が MySQL 8.0 で始まる予約語であるため、問題が発生する可能性があります。

このような場合、レプリケーションはエラー 1064 で失敗する可能性があります: `You have an error in your SQL syntax...`。(予約語を使用して名前が付けられたデータベースまたはテーブル、または予約語を使用して名前が付けられたカラムを持つテーブルが、レプリケーションから除外されていても)。これは、レプリカが影響を受けるデータベースオブジェクトを認識できるように、実行前に各 SQL イベントをレプリカで解析する必要があるためです。イベントが解析された後のみ、レプリカは `--replicate-do-db`, `--replicate-do-table`, `--replicate-ignore-db` および `--replicate-ignore-table` で定義されたフィルタリングルールを適用できます。

レプリカによって予約語とみなされるソース上のデータベース、テーブルまたはカラムの名前の問題を回避するには、次のいずれかを実行します:

- ソースで 1 つ以上の `ALTER TABLE` ステートメントを使用して、レプリカでこれらの名前が予約語とみなされるデータベースオブジェクトの名前を変更し、かわりに古い名前を使用する SQL ステートメントを変更して新しい名前を使用します。
- これらのデータベースオブジェクト名を使用する SQL ステートメントで、それらの名前をバッククォート文字 (```) で囲まれた識別子として書いてください。

MySQL バージョン別の予約語の一覧については、「MySQL Server Version Reference」の「[Reserved Words](#)」を参照してください。識別子を囲むルールについては、[セクション 9.2「スキーマオブジェクト名」](#)を参照してください。

17.5.1.27 レプリケーションおよび行検索

行ベースのレプリケーション形式を使用するレプリカが `UPDATE` または `DELETE` 操作を適用する場合、関連するテーブルで一致する行を検索する必要があります。このプロセスの実行に使用されるアルゴリズムでは、いずれかのテーブルインデックスを使用して最初の選択として検索を実行し、適切なインデックスがない場合はハッシュテーブルを使用します。

アルゴリズムは、まずテーブル定義で使用可能なインデックスを評価して、使用する適切なインデックスがあるかどうか、および複数の可能性があるかどうかを確認します。どのインデックスが操作に最適ですか。このアルゴリズムは、次のタイプのインデックスを無視します:

- 全文インデックス。
- 非表示インデックス。
- 生成されたインデックス。
- Multi-valued indexes。
- 行イベントのビフォアイメージにインデックスのすべてのカラムが含まれていないインデックス。

これらのインデックスタイプを除外した後に適切なインデックスがない場合、アルゴリズムは検索にインデックスを使用しません。適切なインデックスがある場合は、次の優先順位で候補から 1 つのインデックスが選択されます:

1. 主キー。
2. インデックス内のすべてのカラムに `NOT NULL` 属性がある一意のインデックス。このようなインデックスが複数使用可能な場合、アルゴリズムはこれらのインデックスの左端を選択します。
3. その他のインデックス。このようなインデックスが複数使用可能な場合、アルゴリズムはこれらのインデックスの左端を選択します。

インデックス内のすべてのカラムに `NOT NULL` 属性がある主キーまたは一意インデックスをアルゴリズムで選択できる場合、このインデックスを使用して `UPDATE` または `DELETE` 操作の行を反復処理します。行イベントの各行について、アルゴリズムはインデックス内の行を検索して、更新するテーブルレコードを検索します。一致するレコードが見つからない場合は、エラー `ER_KEY_NOT_FOUND` が返され、レプリケーションアプライヤスレッドが停止します。

アルゴリズムが適切なインデックスを見つけられなかった場合、または一意でないか `NULL` を含んでいたインデックスのみを見つけることができた場合は、ハッシュテーブルを使用してテーブルレコードを識別します。このアルゴ

リズムは、`UPDATE` または `DELETE` 操作の行を含むハッシュテーブルを作成し、そのキーを行の完全なビフォアイメージとして使用します。アルゴリズムは、検出された場合は選択されたインデックスを使用してターゲットテーブルのすべてのレコードを反復処理し、それ以外の場合は全テーブルスキャンを実行します。ターゲットテーブルのレコードごとに、その行がハッシュテーブルに存在するかどうかを判別されます。ハッシュテーブルで行が見つかった場合、ターゲットテーブルのレコードが更新され、ハッシュテーブルから行が削除されます。ターゲットテーブルのすべてのレコードがチェックされると、このアルゴリズムはハッシュテーブルが空であるかどうかを検証します。ハッシュテーブルに一致しない行が残っている場合、アルゴリズムはエラー `ER_KEY_NOT_FOUND` を返し、レプリケーションアプライヤスレッドを停止します。

`slave_rows_search_algorithms` システム変数は、以前は一致する行の検索方法を制御するために使用されていました。前述のように、インデックススキャンの後にハッシュスキャンを使用するデフォルト設定はパフォーマンスに最適であり、すべてのシナリオで正しく機能するため、このシステム変数の使用は非推奨になりました。

17.5.1.28 レプリケーションとソースまたはレプリカの停止

レプリケーションソースサーバーをシャットダウンして、あとで再起動しても安全です。レプリカがソースへの接続を失うと、レプリカはただちに再接続を試み、失敗した場合は定期的に再試行します。デフォルトでは 60 秒ごとに再試行します。これは、`CHANGE REPLICATION SOURCE TO` ステートメント (MySQL 8.0.23 の場合) または `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 の場合) を使用して変更できます。レプリカは、ネットワーク接続の停止にも対処できます。ただし、レプリカは、ソースから `slave_net_timeout` 秒間データを受信しなかった後のみネットワークの停止に気付きます。停止時間が短い場合は、`slave_net_timeout` を減らすことをお勧めします。セクション 17.4.2 「レプリカの予期しない停止の処理」を参照してください。

ソースバイナリログファイルがフラッシュされていないため、ソース側でクリーンでないシャットダウン (クラッシュなど) を実行すると、ソースバイナリログの最終的な位置がレプリカによって読み取られた最新の位置より小さくなる可能性があります。これにより、ソースの起動時にレプリカをレプリケートできなくなる可能性があります。ソースサーバーの `my.cnf` ファイルで `sync_binlog=1` を設定すると、ソースがより頻繁にバイナリログをフラッシュするため、この問題を最小限に抑えることができます。InnoDB とトランザクションを使用したレプリケーション設定で永続性と一貫性を最大限に高めるには、`innodb_flush_log_at_trx_commit=1` も設定する必要があります。この設定では、各トランザクションのコミット時に InnoDB redo ログバッファの内容がログファイルに書き込まれ、ログファイルがディスクにフラッシュされます。オペレーティングシステムまたはディスクハードウェアは、ディスクへのフラッシュ操作が行われたことを `mysqld` に通知する場合があるため、トランザクションの永続性はまだこの設定で保証されていないことに注意してください。

レプリカの完全なシャットダウンは、レプリカが停止した場所を追跡するため安全です。ただし、レプリカに一時テーブルが開かれていないことに注意してください。セクション 17.5.1.31 「レプリケーションと一時テーブル」を参照してください。予期しないシャットダウンにより問題が発生する場合があります (特に、ディスクキャッシュがディスクにフラッシュされない状態で問題が発生した場合)。

- トランザクションの場合、レプリカは `relay-log.info` をコミットしてから更新します。これらの 2 つの操作の間に予期しない終了が発生した場合、リレーログの処理は情報ファイルに示されているよりも先に進み、レプリカは再起動後にリレーログ内の最後のトランザクションからイベントを再実行します。
- 同様の問題は、レプリカが `relay-log.info` を更新しても、書き込みがディスクにフラッシュされる前にサーバーホストがクラッシュした場合に発生することがあります。これが発生する可能性を最小限に抑えるには、レプリカ `my.cnf` ファイルに `sync_relay_log_info=1` を設定します。 `sync_relay_log_info` を 0 に設定すると、ディスクへの書き込みは強制されず、サーバーはオペレーティングシステムに依存してファイルを随時フラッシュします。

システムのこのような問題に対する耐障害性は、優れた無停電電源があると大幅に向上します。

17.5.1.29 レプリケーション中のレプリカエラー

ステートメントがソースとレプリカの両方で同じエラー (同一のエラーコード) を生成した場合、エラーはログに記録されますが、レプリケーションは続行されます。

ステートメントがソースとレプリカで異なるエラーを生成すると、レプリケーション SQL スレッドは終了し、レプリカはエラーログにメッセージを書き込み、データベース管理者がエラーに関する処理を決定するのを待機します。これには、ステートメントがソースまたはレプリカでエラーを生成する場合がありますが、両方ではありません。この問題に対処するには、レプリカに手動で接続し、問題の原因を特定します。これには、`SHOW REPLICASlave STATUS` が役立ちます。次に、問題を修正し、`START REPLICASlave` を実行します。たとえば、レプリカを再起動する前に、存在しないテーブルを作成する必要がある場合があります。

注記

レプリカエラーログに一時エラーが記録されている場合は、引用符で囲まれたエラーメッセージに示されているアクションを実行する必要はありません。一時エラーは、トランザクションを再試行するクライアントによって処理される必要があります。たとえば、レプリケーション SQL スレッドがデッドロックに関連する一時エラーを記録している場合、レプリケーション SQL スレッドがその後非一時エラーメッセージで終了しないかぎり、レプリカでトランザクションを手動で再起動する必要はありません。

このエラーコード検証動作が望ましくない場合は、`--slave-skip-errors` オプションでエラーの一部またはすべてを隠す(無視する)ことができます。

MyISAM などの非トランザクションストレージエンジンの場合、テーブルを不完全に更新してエラーコードを返すだけのステートメントが存在する場合があります。これはたとえば、1つの行がキー制約に違反する複数行挿入で、または長い更新ステートメントが一部の行を更新したあとに強制終了された場合に発生する可能性があります。これがソースで発生した場合、レプリカはステートメントの実行結果が同じエラーコードになることを想定しています。そうでない場合、レプリケーション SQL スレッドは前述のように停止します。

ソースとレプリカで異なるストレージエンジンを使用するテーブル間でレプリケートする場合は、同じステートメントをテーブルの一方のバージョンに対して実行すると異なるエラーが生成される可能性があります。他方に対しては生成されないか、一方のバージョンのテーブルでエラーが発生する可能性があります。他方のバージョンでは発生しないことに注意してください。たとえば、**MyISAM** は外部キー制約を無視するため、ソース上の **InnoDB** テーブルにアクセスする **INSERT** または **UPDATE** ステートメントは外部キー違反を引き起こす可能性があります。レプリカ上の同じテーブルの **MyISAM** バージョンで同じステートメントを実行してもそのようなエラーは発生せず、レプリケーションは停止します。

17.5.1.30 レプリケーションとサーバー SQL モード

ソースとレプリカで異なるサーバー SQL モード設定を使用すると、ソースとレプリカで同じ **INSERT** ステートメントが異なる方法で処理され、ソースとレプリカが相違する可能性があります。最良の結果を得るには、ソースとレプリカで常に同じサーバー SQL モードを使用する必要があります。このアドバイスは、ステートメントベースまたは行ベースのどちらのレプリケーションを使用しているかにかかわらず、適用されます。

パーティションテーブルをレプリケートする場合、ソースとレプリカで異なる SQL モードを使用すると、問題が発生する可能性があります。少なくとも、特定のテーブルのソースコピーとレプリカコピーでパーティション間のデータ分散が異なる可能性があります。また、ソースで成功したパーティションテーブルへの挿入がレプリカで失敗する可能性もあります。

詳細は、[セクション5.1.11「サーバー SQL モード」](#)を参照してください。

17.5.1.31 レプリケーションと一時テーブル

MySQL 8.0 では、`binlog_format` が **ROW** または **MIXED** に設定されている場合、一時テーブルのみを使用するステートメントはソースに記録されないため、一時テーブルはレプリケートされません。一時テーブルと非一時テーブルが混在するステートメントは、非一時テーブルに対する操作に対してのみソースに記録され、一時テーブルに対する操作は記録されません。これは、レプリカによって計画外停止が行われた場合に失われるレプリカ上の一時テーブルがないことを意味します。行ベースレプリケーションと一時テーブルの詳細は、[一時テーブルの行ベースロギング](#)を参照してください。

`binlog_format` が **STATEMENT** に設定されている場合、一時テーブルに関連するステートメントをステートメントベースの形式を使用して安全にログに記録できるかぎり、一時テーブルに対する操作はソースに記録され、レプリカにレプリケートされます。この状況では、レプリカ上のレプリケートされた一時テーブルの損失が問題になる可能性があります。ステートメントベースレプリケーションモードでは、GTID がサーバー上で使用されている場合(つまり、`enforce_gtid_consistency` システム変数が **ON** に設定されている場合)、**CREATE TEMPORARY TABLE** および **DROP TEMPORARY TABLE** ステートメントをトランザクション、手順、関数、またはトリガー内で使用することはできません。GTID が使用されている場合、`autocommit=1` が設定されていれば、これらのコンテキストの外部で使用できます。

一時テーブルに関して行ベースまたは混合レプリケーションモードとステートメントベースのレプリケーションモードの動作が異なるため、変更がオープン一時テーブルを含むコンテキスト(グローバルまたはセッション)に適用される場合、実行時にレプリケーション形式を切り替えることはできません。詳細は、`binlog_format` オプションの説明を参照してください。

一時テーブル使用時の安全なレプリカの停止。ステートメントベースレプリケーションモードでは、(レプリケーションスレッドだけでなく)複製サーバーを停止し、まだ複製で実行されていない更新で使用するために開かれている複製一時テーブルがある場合を除き、一時テーブルは複製されます。レプリカサーバーを停止すると、レプリカの再起動時に、これらの更新に必要な一時テーブルを使用できなくなります。この問題を回避するには、一時テーブルが開いている間はレプリカを停止しないでください。代わりに、次の手順を使用してください。

1. `STOP REPLICA | SLAVE SQL_THREAD` ステートメントを発行します。
2. `SHOW STATUS` を使用して `Slave_open_temp_tables` 変数の値を確認します。
3. 値が 0 でない場合は、`START REPLICA | SLAVE SQL_THREAD` を使用してレプリケーション SQL スレッドを再起動し、後で手順を繰り返します。
4. 値が 0 の場合は、`mysqladmin shutdown` コマンドを発行してレプリカを停止します。

一時テーブルとレプリケーションオプション。デフォルトでは、ステートメントベースレプリケーションでは、すべての一時テーブルがレプリケートされます。これは、一致する `--replicate-do-db`、`--replicate-do-table`、または `--replicate-wild-do-table` オプションが有効になっているかどうかにかかわらず発生します。ただし、`--replicate-ignore-table` および `--replicate-wild-ignore-table` オプションは一時テーブルで受け付けられます。ただし、セッションの終了時に一時テーブルを正しく削除できるようにするために、レプリカは、通常は指定されたテーブルに適用される除外ルールに関係なく、常に `DROP TEMPORARY TABLE IF EXISTS` ステートメントをレプリケートします。

ステートメントベースレプリケーションを使用するときに推奨される方法は、レプリケートしない一時テーブルの名前付けに排他的に使用する接頭辞を指定し、その接頭辞と一致するように `--replicate-wild-ignore-table` オプションを使用することです。たとえば、このようなすべてのテーブルに `norep` で始まる名前を付けてから (たとえば、`norepmytable`、`norepyourtable` など)、テーブルが複製されることを回避するために `--replicate-wild-ignore-table=norep%` を使用します。

17.5.1.32 レプリケーション再試行とタイムアウト

グローバルシステム変数 `slave_transaction_retries` は、シングルスレッドまたはマルチスレッドレプリカ上の適用者スレッドが、停止前に失敗したトランザクションを自動的に再試行する最大回数を設定します。InnoDB デッドロックのために SQL スレッドがトランザクションの実行に失敗した場合、またはトランザクション実行時間が `InnoDB innodb_lock_wait_timeout` 値を超えた場合、トランザクションは自動的に再試行されます。トランザクションに、成功を妨げる一時的でないエラーがある場合、再試行されません。

`slave_transaction_retries` のデフォルト設定は 10 です。これは、通常一時エラーで失敗したトランザクションが、アプライヤスレッドが停止する前に 10 回再試行されることを意味します。変数を 0 に設定すると、トランザクションの自動再試行が無効になります。マルチスレッドのレプリカでは、すべてのチャンネルのすべてのアプライヤスレッドで、指定された数のトランザクション再試行を実行できます。「パフォーマンススキーマ」テーブル `replication_applier_status` の `COUNT_TRANSACTIONS_RETRIES` カラムには、各レプリケーションチャンネルで行われたトランザクション再試行の合計数が表示されます。

トランザクションを再試行するプロセスによって、レプリカまたはグループレプリケーショングループメンバーでラグが発生する可能性があります。これは、シングルスレッドまたはマルチスレッドのレプリカとして構成できます。「パフォーマンススキーマ」テーブル `replication_applier_status_by_worker` には、シングルスレッドまたはマルチスレッドレプリカでの適用者スレッドによるトランザクションの再試行に関する詳細情報が表示されます。このデータには、アプライヤスレッドが最後のトランザクションの適用にかかった時間 (および現在進行中のトランザクションが開始された時間) と、元のソースおよび即時ソースでのコミット後の時間を示すタイムスタンプが含まれます。データには、最後のトランザクションおよび現在進行中のトランザクションの再試行回数も表示され、トランザクションの再試行の原因となった一時エラーを識別できます。この情報を使用して、トランザクションの再試行がレプリケーションラグの原因であるかどうかを確認し、再試行の原因となった失敗の根本原因を調査できます。

17.5.1.33 レプリケーションとタイムゾーン

デフォルトでは、ソースサーバーとレプリカサーバーは同じタイムゾーンであると想定します。異なるタイムゾーンのサーバー間でレプリケートする場合は、ソースとレプリカの両方でタイムゾーンを設定する必要があります。それ以外の場合、ソースのローカル時間に依存するステートメント (`NOW()`) または `FROM_UNIXTIME()` 関数を使用するステートメントなどは正しくレプリケートされません。

ソースおよびレプリカのシステムタイムゾーン (`system_time_zone`)、サーバーの現在のタイムゾーン (`time_zone` のグローバル値) およびセッションごとのタイムゾーン (`time_zone` のセッション値) の設定の組合せが正しい結果を生成していることを確認します。特に、`time_zone` システム変数が `SYSTEM` の値に設定されていて、サーバーのタイム

ゾーンがシステムのタイムゾーンと同じであることを示している場合は、ソースとレプリカが異なるタイムゾーンを適用する可能性があります。たとえば、ソースはバイナリログに次のステートメントを書き込むことができます:

```
SET @@session.time_zone='SYSTEM';
```

このソースとそのレプリカのシステムタイムゾーンの設定が異なる場合、レプリカグローバル `time_zone` 値がソースと一致するように設定されていても、このステートメントによってレプリカで予期しない結果が生成される可能性があります。MySQL Server のタイムゾーン設定およびその変更方法の詳細は、[セクション5.1.15「MySQL Serverでのタイムゾーンのサポート」](#)を参照してください。

[セクション17.5.1.14「レプリケーションとシステム関数」](#)も参照してください。

17.5.1.34 レプリケーションとトランザクションの非一貫性

リレーログから実行された一連のトランザクションの不整合は、レプリケーション構成によって発生する可能性があります。このセクションでは、不整合を回避し、問題の原因を解決する方法について説明します。

次のタイプの不整合が存在する可能性があります:

- 半消込済トランザクション。非トランザクションテーブルを更新するトランザクションが、そのすべてではなく一部の変更を適用しました。
- ギャップ。外部化されたトランザクションセット内のギャップは、順序付けられたトランザクションのシーケンスが指定された場合に、シーケンス内の後のトランザクションがシーケンス内の他のトランザクションより前に適用されるときに表示されます。ギャップは、マルチスレッドのレプリカを使用している場合にのみ表示されます。ギャップが発生しないようにするには、`slave_preserve_commit_order=1` を設定します。この設定では、MySQL 8.0.18 まで、バイナリロギング (`log_bin`) およびレプリカ更新ロギング (`log_slave_updates`) も有効にする必要があります。これは、MySQL 8.0 のデフォルト設定です。MySQL 8.0.19 からは、バイナリロギングおよびレプリカ更新ロギングは、`slave_preserve_commit_order=1` を設定するためにレプリカでは必要なく、必要に応じて無効にできます。すべてのリリースで、`slave_preserve_commit_order=1` を設定するには、`slave_parallel_type` がデフォルト設定ではない `LOGICAL_CLOCK` に設定されている必要があります。一部の特定の状況では、`slave_preserve_commit_order` の説明に示されているように、`slave_preserve_commit_order=1` を設定してもレプリカのコミット順序を保持できないため、レプリカリレーログから実行された一連のトランザクションにギャップが表示される場合があります。
- ソースバイナリログの位置ラグ。ギャップがなくても、`Exec_master_log_pos` が適用された後のトランザクションが発生する可能性があります。つまり、`N` までのすべてのトランザクションが適用され、`N` が適用された後のトランザクションは適用されませんが、`Exec_master_log_pos` の値は `N` より小さくなります。この場合、`Exec_master_log_pos` は適用されたトランザクションの「最低水位標」であり、最後に適用されたトランザクションの位置より遅れています。これはマルチスレッドのレプリカでのみ発生します。`slave_preserve_commit_order` を有効にしても、ソースバイナリログの位置ラグは妨げられません。

次のシナリオは、半導体で適用されるトランザクション、ギャップ、およびソースバイナリログの位置ラグの存在に関連します:

- レプリケーションスレッドの実行中に、ギャップと半数適用のトランザクションが発生する場合があります。
- `mysqld` が停止します。クリーンな停止とクリーンでない停止の両方により、進行中のトランザクションが中断され、ギャップと半数適用のトランザクションが残る可能性があります。
- レプリケーションスレッドの `KILL` (シングルスレッドレプリカを使用する場合は SQL スレッド、マルチスレッドレプリカを使用する場合はコーディネータスレッド)。これにより、進行中のトランザクションが中断され、ギャップや半数適用のトランザクションが残る可能性があります。
- 適用者スレッドでエラーが発生しました。これによりギャップが生じる可能性があります。エラーが混合トランザクション内にある場合、そのトランザクションは半適用されます。マルチスレッドのレプリカを使用する場合、エラーを受信していないワーカーはキューを完了するため、すべてのスレッドの停止に時間がかかることがあります。
- マルチスレッドのレプリカを使用している場合は `STOP REPLICAS | SLAVE`。 `STOP REPLICAS | SLAVE` の発行後、レプリカはギャップ一杯になるまで待機し、`Exec_master_log_pos` を更新します。これにより、前述のいずれかのケース (つまり、`STOP REPLICAS | SLAVE` が完了する前、エラーが発生する前、別のスレッドが `KILL` を発行する前、またはサーバーが再起動する前) を除き、ギャップまたはソースバイナリログの位置ラグが保持されることはありません。このような場合、`STOP REPLICAS | SLAVE` は正常に戻ります。

6. リレーログ内の最後のトランザクションが半数受信のみで、マルチスレッドレプリカコーディネータスレッドがワーカーへのトランザクションのスケジュールを開始した場合、`STOP REPLICATION | SLAVE` はトランザクションが受信されるまで最大 60 秒待機します。このタイムアウト後、コーディネータはトランザクションを中止します。トランザクションが混在している場合は、半完了のままにすることができます。
7. シングルスレッドレプリカを使用する場合の `STOP REPLICATION | SLAVE`。進行中のトランザクションがトランザクションテーブルのみを更新する場合、トランザクションテーブルはロールバックされ、`STOP REPLICATION | SLAVE` はただちに停止します。進行中のトランザクションが混在している場合、`STOP REPLICATION | SLAVE` はトランザクションが完了するまで最大 60 秒待機します。このタイムアウト後、トランザクションは中断されるため、半完了のままになります。

グローバル変数 `rpl_stop_slave_timeout` は、レプリケーションスレッドを停止するプロセスとは無関係です。`STOP REPLICATION | SLAVE` を発行するクライアントがクライアントに戻るだけですが、レプリケーションスレッドは引き続き停止しようとしています。

レプリケーションチャンネルにギャップがある場合は、次の結果になります：

1. レプリカデータベースは、ソースに存在していない可能性がある状態です。
2. `SHOW REPLICATION | SLAVE STATUS` のフィールド `Exec_master_log_pos` は、「最低水位標」のみです。つまり、位置の前に表示されるトランザクションはコミットされていることが保証されますが、位置の後のトランザクションはコミットされているかどうかは保証されません。
3. そのチャンネルの `CHANGE REPLICATION SOURCE TO` ステートメントおよび `CHANGE MASTER TO` ステートメントは、適用者スレッドが実行中であり、ステートメントで受信者オプションのみが設定されている場合を除き、エラーで失敗します。
4. `--relay-log-recovery` を使用して `mysqld` を起動した場合、そのチャンネルのリカバリは行われず、警告が出力されません。
5. If `mysqldump` is used with `--dump-slave`, it does not record the existence of gaps; thus it prints `CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO` with `RELAY_LOG_POS` set to the 「最低水位標」 position in `Exec_master_log_pos`.

ダンプを別のサーバーに適用し、レプリケーションスレッドを開始すると、位置の後に表示されるトランザクションが再度レプリケートされます。GTID が有効になっている場合、これは無害であることに注意してください (ただし、その場合、`--dump-slave` を使用することはお勧めしません)。

レプリケーションチャンネルにソースバイナリログの位置ラグがあり、ギャップがない場合は、前述のケース 2 から 5 が適用されますが、ケース 1 は適用されません。

ソースバイナリログの位置情報は、内部テーブル `mysql.slave_worker_info` にバイナリ形式で保持されます。`START REPLICATION | SLAVE [SQL_THREAD]` は常にこの情報を参照して、正しいトランザクションのみを適用します。これは、`slave_parallel_workers` が `START REPLICATION | SLAVE` の前に 0 に変更された場合、および `START REPLICATION | SLAVE` が `UNTIL` 句とともに使用された場合でも当てはまります。`START REPLICATION | SLAVE UNTIL SQL_AFTER_MTS_GAPS` では、ギャップを埋めるために必要な数のトランザクションのみが適用されます。`START REPLICATION | SLAVE` が、すべてのギャップを消費する前に停止するように指示する `UNTIL` 句とともに使用されている場合、残りのギャップは残ります。

警告

`RESET REPLICATION | SLAVE` によってリレーログが削除され、レプリケーション位置がリセットされます。したがって、ギャップのあるレプリカで `RESET REPLICATION | SLAVE` を発行すると、レプリカはギャップを修正せずにギャップに関する情報を失います。

17.5.1.35 レプリケーションとトランザクション

同じトランザクション内にトランザクションおよび非トランザクションステートメントを混在させる。一般的に、レプリケーション環境でトランザクションおよび非トランザクションテーブルの両方を更新するトランザクションは避けるべきです。トランザクション (または一時) および非トランザクションテーブルの両方にアクセスしてそれらに書き込むステートメントを使用することも避けるべきです。

サーバーは、バイナリロギングに次の規則を使用します：

- トランザクション内の最初のステートメントが非トランザクションである場合、それらはすぐにバイナリログに書き込まれます。トランザクション内の残りのステートメントはキャッシュされ、トランザクションがコミットされるまでバイナリログに書き込まれません。(トランザクションがロールバックされた場合、キャッシュされたステートメントは、ロールバックできない非トランザクション変更を行う場合にのみバイナリログに書き込まれます。それ以外の場合は、それらは破棄されます。)
- ステートメントベースロギングの場合、非トランザクションステートメントのロギングは `binlog_direct_non_transactional_updates` システム変数によって影響されます。この変数が `OFF` の場合 (デフォルト)、ロギングは上記のとおりになります。この変数が `ON` の場合、非トランザクションステートメントがトランザクション内のどこで発生しても (最初の非トランザクションステートメントではありません)、ただちにロギングが発生します。ほかのステートメントはトランザクションキャッシュに保持され、トランザクションがコミットしたときにログが記録されます。`binlog_direct_non_transactional_updates` は行形式または混合形式バイナリロギングに影響しません。

トランザクション、非トランザクション、および混合ステートメント。

これらのルールを適用する場合、サーバーは、非トランザクションテーブルだけを変更する場合にはステートメントを非トランザクションと見なし、トランザクションテーブルだけを変更する場合にはトランザクションと見なします。非トランザクションテーブルとトランザクションテーブルの両方を参照し、関係するテーブルを更新するステートメントは、「mixed」ステートメントとみなされます。混合ステートメントは、トランザクションステートメントと同様に、キャッシュされ、トランザクションがコミットするときにログが記録されます。

トランザクションテーブルを更新する混合ステートメントは、次のアクションのいずれかを実行する場合も、安全ではないと見なされます。

- 一時テーブルの更新または読取り
- 非トランザクションテーブルを読み取り、トランザクション分離レベルが `REPEATABLE_READ` 未満

トランザクション内でトランザクションテーブル更新に続く混合ステートメントは、次のアクションのいずれかを実行する場合、安全ではないと見なされます。

- テーブルを更新し、一時テーブルから読み取る
- 非トランザクションテーブルを更新し、`binlog_direct_non_transactional_updates` が `OFF` の場合

詳細については、[セクション17.2.1.3「バイナリロギングでの安全および安全でないステートメントの判断」](#)を参照してください。

注記

混合ステートメントは混合バイナリロギング形式とは関係ありません。

トランザクション内でトランザクションおよび非トランザクションテーブルへの更新が混在している状態で、バイナリログ内のステートメントの順序は正しく、必要なすべてのステートメントがバイナリログに書き込まれます (`ROLLBACK` の場合でも)。ただし、最初の接続トランザクションが完了する前に 2 番目の接続が非トランザクションテーブルを更新するときは、ステートメントログが記録される順序が乱れる場合があります。最初の接続によって実行されているトランザクションの状態にかかわらず、2 番目の接続更新が実行された直後に書き込まれるためです。

ソースとレプリカで異なるストレージエンジンを使用。レプリカの非トランザクションテーブルを使用して、ソースでトランザクションテーブルをレプリケートできます。たとえば、`InnoDB` ソーステーブルを `MyISAM` レプリカテーブルとしてレプリケートできます。ただし、これを行うと、レプリカが `BEGIN ... COMMIT` ブロックの先頭から再開されるため、レプリカが `BEGIN ... COMMIT` ブロックの途中で停止した場合に問題が発生します。

また、ソース上の `MyISAM` テーブルからレプリカ上のトランザクションテーブル (`InnoDB` ストレージエンジンを使用するテーブルなど) にトランザクションをレプリケートすることも安全です。このような場合、ソースで発行された `AUTOCOMMIT=1` ステートメントがレプリケートされるため、レプリカで `AUTOCOMMIT` モードが強制されます。

レプリカのストレージエンジンタイプが非トランザクションの場合、トランザクションテーブルと非トランザクションテーブルの更新が混在するソース上のトランザクションは、ソーストランザクションテーブルとレプリカ非トランザクションテーブルの間でデータの不整合を引き起こす可能性があるため、回避するようにしてください。つまり、このようなトランザクションは、ソースストレージエンジン固有の動作につながり、レプリケーションが同期しなくなる可能性があります。MySQL では、これに関する警告は発行されないため、レプリカ上でソーステーブルから非トランザクションテーブルにトランザクションテーブルをレプリケートする場合は、特に注意する必要があります。

トランザクション内でバイナリロギング形式を変更する。 `binlog_format` および `binlog_checksum` システム変数は、トランザクションが進行中であれば読取り専用です。

各トランザクション (`autocommit` トランザクションを含む) は、`BEGIN` ステートメントで始まり、`COMMIT` または `ROLLBACK` ステートメントで終わるかのように、バイナリログに記録されます。これは、非トランザクションストレージエンジン (`MyISAM` など) を使用するテーブルに影響を与えるステートメントにも当てはまります。

注記

XA トランザクションに特に適用される制限については、[セクション13.3.8.3「XA トランザクションの制約」](#) を参照してください。

17.5.1.36 レプリケーションとトリガー

ステートメントベースレプリケーションでは、ソースで実行されたトリガーもレプリカで実行されます。行ベースのレプリケーションでは、ソースで実行されたトリガーはレプリカで実行されません。かわりに、トリガーの実行によって生成されたソースの行変更がレプリケートされ、レプリカに適用されます。

この動作は設計によります。行ベースのレプリケーションでは、レプリカがトリガーおよびトリガーによって発生した行の変更を適用した場合、変更はレプリカに 2 回適用され、ソースとレプリカのデータが異なります。

ソースとレプリカの両方でトリガーを実行する場合は、おそらくソースとレプリカで異なるトリガーがあるため、ステートメントベースのレプリケーションを使用する必要があります。ただし、レプリカ側トリガーを有効にするには、ステートメントベースのレプリケーションを排他的に使用する必要はありません。この効果が必要なステートメントでのみステートメントベースレプリケーションに切り替え、残りの時間は行ベースレプリケーションを使用することで十分です。

`AUTO_INCREMENT` カラムを更新するトリガー (または関数) を呼び出すステートメントは、ステートメントベースレプリケーションを使用して正しく複製されません。MySQL 8.0 はこのようなステートメントを安全でないとマークします。(Bug #45677)

トリガーは、トリガーイベント (`INSERT`, `UPDATE`, `DELETE`) とアクション時間 (`BEFORE`, `AFTER`) の様々な組合せに対してトリガーを持つことができ、複数のトリガーが許可されます。

簡潔にするために、ここでの「複数のトリガー」は「同じトリガーイベントおよびアクション時間を持つ複数のトリガー」の短縮形です。

Upgrades. MySQL 5.7 より前のバージョンでは、複数のトリガーはサポートされていません。MySQL 5.7 より前のバージョンを使用するレプリケーショントポロジのサーバーをアップグレードする場合は、まずレプリカをアップグレードしてから、ソースをアップグレードします。アップグレードされたレプリケーションソースサーバーに、複数のトリガーをサポートしていない MySQL バージョンを使用する古いレプリカがまだ存在する場合、トリガーイベントとアクション時間がすでに同じトリガーを持つテーブルのソースでトリガーが作成されると、それらのレプリカでエラーが発生します。

Downgrades. 複数のトリガーをサポートするサーバーをサポートしていない古いバージョンにダウングレードすると、ダウングレードは次の影響を受けます:

- トリガーを持つテーブルごとに、すべてのトリガー定義はテーブルの `.TRG` ファイルにあります。ただし、トリガーイベントとアクション時間が同じトリガーが複数ある場合、サーバーはトリガーイベントが発生したときにいずれか一方のみを実行します。`.TRG` ファイルの詳細は、<https://dev.mysql.com/doc/index-other.html> で入手可能な MySQL Server Doxygen ドキュメントの「テーブルトリガー記憶域」に関するセクションを参照してください。
- ダウングレード後にテーブルのトリガーが追加または削除されると、サーバーはテーブルの `.TRG` ファイルを書き換えます。リライトされたファイルには、トリガーイベントとアクション時間の組合せごとに 1 つのトリガーのみが保持され、他のトリガーは失われます。

これらの問題を回避するには、ダウングレードする前にトリガーを変更します。トリガーイベントとアクション時間の組合せごとに複数のトリガーを持つテーブルごとに、次のようにトリガーの各セットを単一のトリガーに変換します:

- トリガーごとに、トリガー内のすべてのコードを含むストアドルーチンを作成します。`NEW` および `OLD` を使用してアクセスされる値は、パラメータを使用してルーチンに渡すことができます。トリガーにコードからの単一の結果値が必要な場合は、ストアドファンクションにコードを配置し、その関数で値を戻すことができます。ト

リガーにコードからの複数の結果値が必要な場合は、ストアードプロシージャにコードを配置し、`OUT` パラメータを使用して値を返すことができます。

2. テーブルのすべてのトリガーを削除します。
3. 作成したストアードルーチン呼び出すテーブルに対して、新しいトリガーを作成します。したがって、このトリガーの効果は、トリガーが置換する複数のトリガーと同じです。

17.5.1.37 レプリケーションと TRUNCATE TABLE

`TRUNCATE TABLE` は通常は DML ステートメントと見なされるため、バイナリロギングモードが `ROW` または `MIXED` のときは行ベース形式を使用してログが記録されて複製されることが予想されます。しかしこのことが、`InnoDB` などのトランザクションストレージエンジン (トランザクション分離レベルが `READ COMMITTED` または `READ UNCOMMITTED` (ステートメントベースロギングを排除)) を使用するテーブルを `STATEMENT` または `MIXED` モードでログを記録または複製するときに、問題を発生させました。

`TRUNCATE TABLE` は、ロギングおよびレプリケーション目的のときはステートメントとしてログを記録し複製できるように、DML ではなく DDL として扱われます。ただし、レプリカに対する `InnoDB` および他のトランザクションテーブルに適用可能なステートメントの影響は、そのようなテーブルを制御する [セクション13.1.37「TRUNCATE TABLE ステートメント」](#) で説明されているルールに従います。(Bug #36763)

17.5.1.38 レプリケーションおよびユーザー名の長さ

MySQL 8.0 でのユーザー名の最大長は 32 文字です。16 文字を超えるユーザー名のレプリケーションは、5.7 より前のバージョンの MySQL をレプリカが実行している場合に失敗します。これらのバージョンでは短いユーザー名のみがサポートされるためです。これは、新しいソースから古いレプリカにレプリケートする場合にのみ発生します。これは推奨される構成ではありません。

17.5.1.39 レプリケーションと変数

システム変数は、次の変数を除き (セッションスコープで使用されるとき)、`STATEMENT` モード使用時は正しく複製されません。

- `auto_increment_increment`
- `auto_increment_offset`
- `character_set_client`
- `character_set_connection`
- `character_set_database`
- `character_set_server`
- `collation_connection`
- `collation_database`
- `collation_server`
- `foreign_key_checks`
- `identity`
- `last_insert_id`
- `lc_time_names`
- `pseudo_thread_id`
- `sql_auto_is_null`
- `time_zone`

- [timestamp](#)
- [unique_checks](#)

MIXED モード使用時に、前述のリスト内の変数がセッションスコープで使用される場合はステートメントベースから行ベースロギングに切り替わります。 [セクション5.4.4.3「混合形式のバイナリロギング形式」](#)を参照してください。

[sql_mode](#) も [NO_DIR_IN_CREATE](#) モードを除いてレプリケートされます。レプリカは、ソースでの変更に関係なく、常に [NO_DIR_IN_CREATE](#) の独自の値を保持します。これは、すべてのレプリケーション形式に当てはまりません。

ただし、[mysqlbinlog](#) が `SET @@sql_mode = mode` ステートメントを解析したときに、[NO_DIR_IN_CREATE](#) を含む完全な `mode` 値が受信サーバーに渡されます。このため、このようなステートメントのレプリケーションは、[STATEMENT](#) モード使用時は安全でない場合があります。

ロギングモードに関係なく、[default_storage_engine](#) システム変数はレプリケートされません。これは、異なるストレージエンジン間のレプリケーションを容易にすることを目的としています。

[read_only](#) システム変数は複製されません。さらに、この変数を有効にした場合の一時テーブル、テーブルロック、および [SET PASSWORD](#) ステートメントに関する効果は、MySQL バージョンごとに異なります。

[max_heap_table_size](#) システム変数は複製されません。レプリカでこれを行わずにソースでこの変数の値を増やすと、ソース上の [MEMORY](#) テーブルで [INSERT](#) ステートメントを実行しようとしたときに、最終的にレプリカで「[テーブルがいっぱいです](#)」エラーが発生する可能性があるため、レプリカ上の対応する値より大きくなるのが許可されます。詳細については、[セクション17.5.1.21「レプリケーションとMEMORY テーブル」](#)を参照してください。

ステートメントベースレプリケーションで、セッション変数は、テーブルを更新するステートメントで使用されるときに正しく複製されません。たとえば、次の一連のステートメントでは、ソースとレプリカに同じデータは挿入されません:

```
SET max_join_size=1000;
INSERT INTO mytable VALUES(@@max_join_size);
```

これは、一般的なシーケンスには適用されません。

```
SET time_zone=...;
INSERT INTO mytable VALUES(CONVERT_TZ(..., ..., @@time_zone));
```

セッション変数のレプリケーションは、行ベースレプリケーションが使用されているときは問題ではありません。このケースでは、セッション変数は常に安全に複製されます。 [セクション17.2.1「レプリケーション形式」](#)を参照してください。

次のセッション変数はバイナリログに書き込まれ、ロギング形式に関係なく、バイナリログの解析時にレプリカによって適用されます:

- [sql_mode](#)
- [foreign_key_checks](#)
- [unique_checks](#)
- [character_set_client](#)
- [collation_connection](#)
- [collation_database](#)
- [collation_server](#)
- [sql_auto_is_null](#)

重要

文字セットと照合順序に関するセッション変数はバイナリログに書き込まれるけれども、異なる文字セット間のレプリケーションはサポートされません。

混乱の可能性を低減するために、特にファイルシステムが大/小文字を区別するプラットフォームで MySQL を実行している場合は、ソースとレプリカの両方で `lower_case_table_names` システム変数に常に同じ設定を使用することをお勧めします。 `lower_case_table_names` 設定は、サーバーの初期化時にのみ構成できます。

17.5.1.40 レプリケーションとビュー

ビューは常にレプリカにレプリケートされます。ビューは、それらが参照するテーブルではなく、それら独自の名前でフィルタリングされます。つまり、通常は `replication-ignore-table` ルールによってフィルタで除外されるテーブルがビューに含まれている場合でも、ビューをレプリカにレプリケートできます。このため、通常はセキュリティ上の理由でフィルタリングされるテーブルデータをビューが複製しないように、気を付けるようにしてください。

テーブルから同じ名前のビューへのレプリケーションは、ステートメントベースのロギングを使用してサポートされますが、行ベースのロギングを使用している場合はサポートされません。行ベースのロギングが有効なときに試行すると、エラーが発生します。

17.5.2 MySQL バージョン間のレプリケーション互換性

MySQL は、あるリリースシリーズから次の上位リリースシリーズへのレプリケーションをサポートしています。たとえば、MySQL 5.6 を実行しているソースから MySQL 5.7 を実行しているレプリカ、MySQL 5.7 を実行しているソースから MySQL 8.0 を実行しているレプリカなどにレプリケートできます。ただし、ソースがステートメントを使用しているか、レプリカで使用されている MySQL のバージョンでサポートされなくなった動作に依存している場合、古いソースから新しいレプリカにレプリケートするときに問題が発生することがあります。たとえば、64 文字を超える外部キー名は、MySQL 8.0 からサポートされなくなりました。

複数のソースを含むレプリケーション設定では、ソースまたはレプリカ MySQL サーバーの数に関係なく、複数の MySQL Server バージョンの使用はサポートされていません。この制限は、リリースシリーズだけでなく、同じリリースシリーズ内のバージョン番号にも適用されます。たとえば、連鎖レプリケーション設定または循環レプリケーション設定を使用している場合、MySQL 8.0.1、MySQL 8.0.2 および MySQL 8.0.4 を同時に使用することはできませんが、これらのリリースのいずれかを同時に使用できます。

重要

レプリケーション (およびその他の) 機能は継続的に改善されるため、特定の MySQL リリースシリーズ内で使用可能な最新リリースを使用することを強くお勧めします。また、MySQL のリリースシリーズの初期リリースを使用するソースおよびレプリカを GA (本番) リリースにアップグレードすることもお勧めします (後者がそのリリースシリーズで使用可能になった場合)。

MySQL 8.0.14 から、サーバーのバージョンは、トランザクションを最初にコミットしたサーバー (`original_server_version`) の各トランザクション、およびレプリケーショントポロジ (`immediate_server_version`) の現在のサーバーの即時ソースであるサーバーのバイナリログに記録されます。

新しいソースから古いレプリカへのレプリケーションは可能ですが、通常はサポートされていません。これはいくつかの要因によります。

- **バイナリログ形式の変更。** バイナリログ形式はメジャーリリース間で変わることがあります。下位互換性を維持しようと試みてはいますが、これがいつか可能なわけではありません。ソースでは、バイナリログトランザクション圧縮など、古いレプリカで認識されないオプションの機能が有効になっている場合もあります。バイナリログトランザクション圧縮では、MySQL 8.0.20 より前のリリースでは結果の圧縮トランザクションペイロードをレプリカで読み取ることができません。

これは、レプリケーションサーバーのアップグレードにも密接な関係があります。詳細は、[セクション17.5.3「レプリケーションセットアップをアップグレードする」](#)を参照してください。

- 行ベースレプリケーションの詳細は、[セクション17.2.1「レプリケーション形式」](#)を参照してください。
- **SQL 非互換。** レプリケート対象のステートメントがソースで使用可能でレプリカではなく SQL 機能を使用している場合、ステートメントベースのレプリケーションを使用して新しいソースから古いレプリカにレプリケートすることはできません。

ただし、ソースとレプリカの両方が行ベースのレプリケーションをサポートし、ソースで検出されたがレプリカでは検出されなかった SQL 機能に依存するレプリケート対象のデータ定義ステートメントがない場合は、ソースで実

行された DDL がレプリカでサポートされていない場合、行ベースのレプリケーションを使用してデータ変更ステートメントの影響をレプリケートできます。

潜在的なレプリケーション問題の詳細は、[セクション 17.5.1 「レプリケーションの機能と問題」](#) を参照してください。

17.5.3 レプリケーションセットアップをアップグレードする

レプリケーションセットアップに関するサーバーをアップグレードするときに、アップグレードの手順は現在のサーバーバージョンとアップグレード後のバージョンによって異なります。このセクションでは、アップグレードがレプリケーションに与える影響について説明します。MySQL のアップグレードの一般情報は、[セクション 2.11 「MySQL のアップグレード」](#) を参照してください

以前の MySQL リリースシリーズから 8.0 にソースをアップグレードする場合は、まず、このソースのすべてのレプリカが同じ 8.0.x リリースを使用していることを確認する必要があります。そうでない場合は、まずレプリカをアップグレードする必要があります。各レプリカをアップグレードするには、レプリカを停止し、適切な 8.0.x バージョンにアップグレードして再起動し、レプリケーションを再開します。アップグレード後にレプリカによって作成されるリレーログは、8.0 形式です。

厳密な SQL モード (`STRICT_TRANS_TABLES` または `STRICT_ALL_TABLES`) で操作に影響を与える変更により、アップグレードされたレプリカでレプリケーションが失敗する可能性があります。ステートメントベースのロギング (`binlog_format=STATEMENT`) を使用する場合、ソースの前にレプリカがアップグレードされると、ソースはソースで成功したがレプリカで失敗する可能性があるステートメントを実行するため、レプリケーションが停止します。これに対処するには、ソース上のすべての新しいステートメントを停止し、レプリカがキャッチアップされるまで待機してから、レプリカをアップグレードします。または、新しいステートメントを停止できない場合は、ソース (`binlog_format=ROW`) で一時的に行ベースのロギングに変更し、この変更時点までに生成されたすべてのバイナリログがすべてのレプリカによって処理されるまで待機してから、レプリカをアップグレードします。

MySQL 8.0 で、デフォルトの文字セットが `latin1` から `utf8mb4` に変更されました。レプリケートされた設定では、MySQL 5.7 から 8.0 にアップグレードする場合、アップグレードする前にデフォルトの文字セットを MySQL 5.7 で使用されている文字セットに戻すことをお勧めします。アップグレードの完了後、デフォルトの文字セットを `utf8mb4` に変更できます。以前のデフォルトが使用されていたと仮定すると、これらを保持する方法の 1 つは、`my.cnf` ファイル内の次の行を使用してサーバーを起動することです:

```
[mysqld]
character_set_server=latin1
collation_server=latin1_swedish_ci
```

レプリカがアップグレードされたら、ソースを停止し、レプリカと同じ 8.0.x リリースにアップグレードして再起動します。ソースを一時的に行ベースのロギングに変更した場合は、ステートメントベースのロギングに戻します。8.0 ソースは、アップグレード前に書き込まれた古いバイナリログを読み取り、それらを 8.0 レプリカに送信できます。レプリカは古い形式を認識し、適切に処理します。アップグレード後にソースによって作成されるバイナリログは、8.0 形式です。これらも 8.0 レプリカによって認識されます。

つまり、MySQL 8.0 にアップグレードする場合、ソースを 8.0 にアップグレードする前にレプリカが MySQL 8.0 である必要があります。8.0 から古いバージョンへのダウングレードは、それほど簡単には機能しません。ダウングレードに進む前に 8.0 バイナリログまたはリレーログを削除できるように、それらが完全に処理されたことを確認する必要があります。

アップグレードによっては、ある MySQL シリーズから次のものに移行するときに、データベースオブジェクトの削除と再作成が必要になる場合があります。たとえば、照合順序の変更には、テーブルインデックスの再構築が必要な場合があります。このような操作の詳細は、必要に応じて [セクション 2.11.4 「MySQL 8.0 での変更」](#) を参照してください。レプリカとソースでこれらの操作を個別に実行し、ソースからレプリカへのこれらの操作のレプリケーションを無効にすることが最も安全です。これを実現するには、次の手順を使用してください。

1. すべてのレプリカを停止し、アップグレードします。ソースに接続しないように、`--skip-slave-start` オプションを使用して再起動します。データベースオブジェクトの再作成に必要なテーブル修復または再構築操作 (`REPAIR TABLE`、`ALTER TABLE` の使用など)、またはテーブルまたはトリガーのダンプおよびリロードを実行します。
2. ソースのバイナリログを無効にします。ソースを再起動せずにこれを行うには、`SET sql_log_bin = OFF` ステートメントを実行します。または、ソースを停止し、`--skip-log-bin` オプションを使用して再起動します。ソースを再

起動する場合は、クライアント接続を禁止することもできます。たとえば、すべてのクライアントが TCP/IP を使用して接続する場合は、ソースの再起動時に `skip_networking` システム変数を有効にします。

- バイナリログが無効の状態、データベースオブジェクトの再作成に必要なテーブル修復または再構築操作を実行します。これらの操作がログに記録され、あとでレプリカに送信されないようにするには、この手順でバイナリログを無効にする必要があります。
- ソースでバイナリログを再度有効にします。以前に `sql_log_bin` を OFF に設定した場合は、`SET sql_log_bin = ON` ステートメントを実行します。バイナリログを無効にするためにソースを再起動した場合は、クライアントとレプリカが接続できるように、`--skip-log-bin` なしで `skip_networking` システム変数を有効にせずに再起動します。
- 今回は、`--skip-slave-start` オプションを指定せずにレプリカを再起動します。

グローバルトランザクション識別子をサポートしていない MySQL のバージョンからアップグレードする場合は、GTID ベースのレプリケーションのすべての要件を設定が満たしていることを確認する前に、ソースまたはレプリカで GTID を有効にしないでください。セクション 17.1.3.4 「GTID を使用したレプリケーションのセットアップ」を参照してください。GTID ベースレプリケーションを使用するために既存のレプリケーションセットアップを変換することに関する情報が含まれています。

MySQL 8.0.16 より前では、グローバルトランザクション識別子 (GTID) を有効にして (`gtid_mode=ON`) サーバーを実行している場合、`mysql_upgrade` (`--write-binlog` オプション) によるバイナリロギングを有効にしないでください。MySQL 8.0.16 の時点では、サーバーは MySQL アップグレード手順全体を実行しますが、アップグレード中はバイナリロギングを無効にするため、問題はありません。

ダンプファイルにシステムテーブルが含まれている場合、GTID がサーバー (`gtid_mode=ON`) で有効になっているときにダンプファイルをロードすることはお勧めしません。`mysqldump` は、非トランザクション MyISAM ストレージエンジンを使用するシステムテーブルに対して DML 命令を発行します。GTID が有効になっている場合、この組み合わせは許可されません。GTID が有効になっているサーバーから GTID が有効になっている別のサーバーにダンプファイルをロードすると、異なるトランザクション識別子が生成されることにも注意してください。

17.5.4 レプリケーションのトラブルシューティング

指示に従ってもレプリケーションセットアップが機能しない場合、最初に行うことはエラーログでメッセージを確認することです。多くのユーザーは、問題が発生したあとにこれを十分に実行せずに、時間を失います。

エラーログから何が問題だったのかわからない場合は、次の手法を試してください。

- `SHOW MASTER STATUS` ステートメントを発行して、ソースのバイナリロギングが有効になっていることを確認します。バイナリロギングはデフォルトで有効になっています。バイナリロギングが有効になっている場合、`Position` はゼロ以外です。バイナリロギングが有効になっていない場合は、バイナリロギングを無効にする設定 (`--skip-log-bin` オプションなど) でソースを実行していないことを確認します。
- ソースとレプリカの両方で起動時に `server_id` システム変数が設定され、ID 値が各サーバーで一意であることを確認します。
- レプリカが実行されていることを確認します。`SHOW REPLICA | SLAVE STATUS` を使用して、`Replica_IO_Running` と `Replica_SQL_Running` の両方の値が `Yes` であるかどうかを確認します。そうでない場合は、レプリカサーバーの起動時に使用されたオプションを確認します。たとえば、`--skip-slave-start` では、`START REPLICA | SLAVE` ステートメントを発行するまでレプリケーションスレッドが起動しません。
- レプリカが実行中の場合は、ソースへの接続が確立されているかどうかを確認します。`SHOW PROCESSLIST` を使用して I/O スレッドと SQL スレッドを見つけ、それらの `State` カラムをチェックしてそれらに何が表示されているかを確認してください。セクション 17.2.3 「レプリケーションスレッド」を参照してください。I/O スレッド状態が `Connecting to master` である場合、次のことを確認してください。
 - ソースのレプリケーションユーザーの権限を確認します。
 - ソースのホスト名が正しいこと、およびソースへの接続に正しいポートを使用していることを確認してください。レプリケーションに使用されるポートは、クライアントネットワーク通信に使用されるポートと同じです (デフォルトは `3306` です)。ホスト名の場合、その名前が正しい IP アドレスに解決されることを確認してください。
 - 構成ファイルをチェックして、ネットワークを無効にするためにソースまたはレプリカで `skip_networking` システム変数が有効になっているかどうかを確認します。その場合は、設定をコメント化するか、削除します。

- ソースにファイアウォールまたは IP フィルタリング構成がある場合は、MySQL に使用されているネットワークポートがフィルタ処理されていないことを確認します。
- ping または [traceroute/tracert](#) を使用してホストにアクセスし、ソースに到達できることを確認します。
- レプリカが以前に実行されていたが停止している場合は、通常、ソースで成功した一部のステートメントがレプリカで失敗したためです。これは、ソースの適切なスナップショットを取得し、レプリケーションスレッド外のレプリカのデータを変更していない場合には発生しません。レプリカが予期せず停止した場合は、バグであるか、[セクション 17.5.1 「レプリケーションの機能と問題」](#) で説明されている既知のレプリケーション制限のいずれかが発生しています。バグの場合は、報告方法の説明を [セクション 17.5.5 「レプリケーションバグまたは問題を報告する方法」](#) で参照してください。
- ソースで成功したステートメントがレプリカでの実行を拒否した場合、レプリカデータベースを削除してソースから新しいスナップショットをコピーすることで、完全なデータベース再同期を実行できない場合は、次の手順を実行します:
 1. レプリカ上の影響を受けるテーブルがソーステーブルと異なるかどうかを確認します。これがどのように発生したかを理解しようとしてください。次に、レプリカテーブルをソースと同一にして、[START REPLICASLAVE](#) を実行します。
 2. 前述のステップが機能しないか適用されない場合は、手動で更新を安全に行うかどうか (必要な場合) を理解してから、ソースの次のステートメントを無視してください。
 3. レプリカがソースから次のステートメントをスキップできると判断した場合は、次のステートメントを発行します:

```
mysql> SET GLOBAL sql_slave_skip_counter = N;  
mysql> START SLAVE;  
Or from MySQL 8.0.22:  
mysql> START REPLICAS;
```

ソースの次のステートメントで [AUTO_INCREMENT](#) または [LAST_INSERT_ID\(\)](#) を使用しない場合、N の値は 1 にする必要があります。そうでない場合は、値は 2 であるべきです。 [AUTO_INCREMENT](#) または [LAST_INSERT_ID\(\)](#) を使用するステートメントに値 2 を使用する理由は、ソースのバイナリログで 2 つのイベントを取得するためです。

[SET GLOBAL sql_slave_skip_counter Statement](#) も参照してください。

4. レプリカがソースと完全に同期化を開始し、レプリケーションスレッドの外部に関係するテーブルを誰も更新していないことが確実な場合は、バグの結果として相違が発生している可能性があります。最新バージョンの MySQL を実行している場合は、この問題を報告してください。古いバージョンを実行している場合は、最新の本番環境リリースにアップグレードして問題が持続するかどうかを判断してみてください。

17.5.5 レプリケーションバグまたは問題を報告する方法

関係するユーザーエラーはないと判断したけれども、依然としてレプリケーションがまったく機能しないか安定しない場合は、バグレポートを送る時期です。バグを突き止めるため、できるだけ多くの情報をユーザーから入手する必要があります。優れたバグレポートを準備するために、ある程度の時間と労力を費やしてくださるようお願いいたします。

そのバグをはっきりと示す再現可能なテストケースがある場合は、[セクション 1.6 「質問またはバグをレポートする方法」](#) で示す手順を使用してオラクルのバグデータベースに入力してくださるようお願いいたします。「幽霊のような」問題 (意のままに再現できないもの) の場合は、次の手順を使用してください。

1. ユーザーエラーが関係していないことを確認します。たとえば、レプリケーションスレッド外でレプリカを更新すると、データが同期しなくなり、更新時に一意キー違反が発生する可能性があります。この場合、レプリケーションスレッドは停止し、テーブルを手動でクリーンアップして同期化するまで待機します。これは、レプリケーションの問題ではありません。外部干渉の問題でレプリケーションが失敗しています。
2. バイナリロギングを有効にして ([log_bin](#) システム変数)、[--log-slave-updates](#) オプションを有効にしてレプリカが実行されていることを確認します。これにより、ソースから受信した更新がレプリカによって独自のバイナリログに記録されます。これらの設定はデフォルトです。

- レプリケーション状態をリセットする前のすべての証拠を保存します。情報がなかったり、不完全な情報しかなかったりすると、オラクルでは問題を突き止めることが困難または不可能になります。収集すべき証拠:
 - ソースのすべてのバイナリログファイル
 - レプリカからのすべてのバイナリログファイル
 - 問題を検出した時点でのソースからの `SHOW MASTER STATUS` の出力
 - 問題を検出した時点でのレプリカからの `SHOW REPLICA | SLAVE STATUS` の出力
 - ソースおよびレプリカからのエラーログ
- `mysqlbinlog` を使用してバイナリログを調べます。問題の説明ステートメントを見つけるには、次のことが役立ちます。`log_file` および `log_pos` は、`SHOW REPLICA | SLAVE STATUS` の `Master_Log_File` および `Read_Master_Log_Pos` の値です。

```
shell> mysqlbinlog --start-position=log_pos log_file | head
```

問題の証拠を収集したあとは、まずそれらを個別のテストケースとして切り分けてみてください。そのうえで、[セクション1.6「質問またはバグをレポートする方法」](#)での指示を使用して問題およびできるだけ多くの情報をオラクルのバグデータベースに入力してください。

第 18 章 グループレプリケーション

目次

18.1	グループレプリケーションのバックグラウンド	3242
18.1.1	レプリケーションテクノロジー	3243
18.1.2	グループレプリケーションのユースケース	3245
18.1.3	マルチプライマリモードとシングルプライマリモード	3246
18.1.4	グループレプリケーションサービス	3250
18.1.5	グループレプリケーションプラグインのアーキテクチャ	3252
18.2	はじめに	3254
18.2.1	単一プライマリモードでのグループレプリケーションのデプロイ	3254
18.2.2	グループレプリケーションのローカルでのデプロイ	3265
18.3	グループレプリケーションの監視	3266
18.3.1	グループレプリケーションサーバーの状態	3267
18.3.2	replication_group_members テーブル	3268
18.3.3	replication_group_member_stats テーブル	3268
18.4	グループレプリケーション操作	3269
18.4.1	オンライングループの構成	3269
18.4.2	トランザクション一貫性保証	3273
18.4.3	分散リカバリ	3278
18.4.4	ネットワークパーティション化	3292
18.4.5	IPv6 および IPv6 と IPv4 の混合グループのサポート	3297
18.4.6	グループレプリケーションでの MySQL Enterprise Backup の使用	3299
18.5	グループレプリケーションセキュリティ	3304
18.5.1	グループレプリケーション IP アドレスの権限	3304
18.5.2	Secure Socket Layer (SSL) を使用したグループ通信接続の保護	3306
18.5.3	分散リカバリ接続の保護	3308
18.6	グループレプリケーションのパフォーマンス	3311
18.6.1	グループ通信スレッドの微調整	3311
18.6.2	フロー制御	3312
18.6.3	メッセージ圧縮	3313
18.6.4	メッセージの断片化	3314
18.6.5	XCom キャッシュ管理	3315
18.6.6	障害検出およびネットワークパーティション化へのレスポンス	3316
18.7	グループレプリケーションのアップグレード	3321
18.7.1	グループ内の異なるメンバーバージョンの組合せ	3322
18.7.2	グループレプリケーションのオフラインアップグレード	3324
18.7.3	グループレプリケーションのオンラインアップグレード	3324
18.8	グループレプリケーションシステム変数	3328
18.9	要件と制限事項	3361
18.9.1	グループレプリケーションの要件	3361
18.9.2	グループレプリケーションの制限事項	3363
18.10	よくある質問	3365

この章では、MySQL Group Replication と、グループをインストール、構成および監視する方法について説明します。MySQL Group Replication を使用すると、柔軟で可用性の高いフォルトトレラントレプリケーショントポロジを作成できます。

グループは、一度に 1 つのサーバーのみが更新を受け入れる自動プライマリ選択を使用して、単一プライマリモードで動作できます。または、グループをマルチプライマリモードでデプロイすることもできます。このモードでは、更新が同時に発行されても、すべてのサーバーが更新を受け入れることができます。

グループのビューの一貫性を保ち、任意の時点ですべてのサーバーで使用できるようにする組み込みグループメンバーシップサービスがあります。サーバーはグループから退出して参加でき、それに応じてビューが更新されます。サーバーが予期せずにグループから離れる場合があります。その場合、障害検出メカニズムはこれを検出し、ビューが変更されたことをグループに通知します。これはすべて自動です。

Group Replication は、データベースサービスが継続的に使用可能であることを保証します。ただし、いずれかのグループメンバーが使用できなくなった場合、コネクタ、ロードバランサ、ルーターまたはなんらかの形式のミドルウェアを使用して、そのグループメンバーに接続されているクライアントをグループ内の別のサーバーにリダイレクトまたはフェイルオーバーする必要があることを理解することが重要です。グループレプリケーションには、これを行う組み込みの方法はありません。たとえば、[MySQL Router 8.0](#) を参照してください。

Group Replication は、MySQL Server へのプラグインとして提供されます。この章の手順に従って、グループに含める各サーバーインスタンスでプラグインを構成し、グループを起動し、グループをモニターおよび管理できます。MySQL サーバーインスタンスのグループをデプロイする別の方法は、[InnoDB クラスタ](#) を使用することです。

ヒント

MySQL の複数のインスタンスをデプロイするには、[MySQL Shell](#) で MySQL サーバーインスタンスのグループを簡単に管理できるようにする [InnoDB クラスタ](#) を使用できます。InnoDB クラスタは MySQL Group Replication をプログラム環境でラップするため、MySQL インスタンスのクラスタを簡単にデプロイして高可用性を実現できます。また、InnoDB クラスタは [MySQL Router](#) とシームレスにインタフェースするため、アプリケーションは独自のフェイルオーバープロセスを記述せずにクラスタに接続できます。ただし、高可用性を必要としない同様のユースケースでは、[InnoDB ReplicaSet](#) を使用できます。MySQL Shell のインストール手順は、[here](#) にあります。

この章の構成は次のとおりです：

- [セクション18.1「グループレプリケーションのバックグラウンド」](#) では、グループの概要とグループレプリケーションの仕組みについて説明します。
- [セクション18.2「はじめに」](#) では、複数の MySQL Server インスタンスを構成してグループを作成する方法について説明します。
- [セクション18.3「グループレプリケーションの監視」](#) では、グループをモニターする方法について説明します。
- [セクション18.4「グループレプリケーション操作」](#) では、グループの操作方法について説明します。
- [セクション18.5「グループレプリケーションセキュリティ」](#) では、グループを保護する方法について説明します。
- [セクション18.6「グループレプリケーションのパフォーマンス」](#) では、グループのパフォーマンスを微調整する方法について説明します。
- [セクション18.7「グループレプリケーションのアップグレード」](#) では、グループのアップグレード方法について説明します。
- [セクション18.8「グループレプリケーションシステム変数」](#) は、Group Replication に固有のシステム変数のリファレンスです。
- [セクション18.9「要件と制限事項」](#) では、Group Replication の技術要件および制限事項について説明します。
- [セクション18.10「よくある質問」](#) では、グループレプリケーションのデプロイおよびオペレーティングに関する技術的な質問に対する回答を提供しています。

18.1 グループレプリケーションのバックグラウンド

このセクションでは、MySQL Group Replication のバックグラウンド情報を提供します。

フォルトトレラントシステムを作成する最も一般的な方法は、コンポーネントを冗長にすることです。つまり、コンポーネントを削除でき、システムは引き続き予想どおりに動作する必要があります。これにより、このようなシステムの複雑さを異なるレベル全体に引き上げる一連の課題が作成されます。具体的には、レプリケートされたデータベースは、1つのサーバーではなく複数のサーバーのメンテナンスおよび管理が必要であるという事実に対処する必要があります。さらに、サーバーが連携してグループを作成しているため、ネットワークのパーティション化やスプリットプレーンシナリオなど、他のいくつかのクラシック分散システムの問題を処理する必要があります。

したがって、最終的な課題は、データベースおよびデータレプリケーションのロジックを、一貫性のある簡単な方法で複数のサーバーを調整するロジックと融合することです。つまり、複数のサーバーがシステムの状態と、システムが通過する各変更のデータに同意するようにします。これは、すべてが単一のデータベースとして進行するように、

または最終的に同じ状態に収束するように、サーバーが各データベースの状態遷移で承諾に到達するように要約できます。つまり、(分散) 状態マシンとして動作する必要があります。

MySQL Group Replication は、サーバー間で強かに調整された分散状態マシンのレプリケーションを提供します。サーバーは、同じグループの一部である場合に自動的に調整されます。グループは、一度に 1 つのサーバーのみが更新を受け入れる自動プライマリ選択を使用して、単一プライマリモードで動作できます。または、より上級のユーザーの場合は、グループをマルチプライマリモードでデプロイできます。マルチプライマリモードでは、同時に発行された場合でも、すべてのサーバーが更新を受け入れることができます。この機能は、このようなデプロイメントによって課される制限を回避する必要があるアプリケーションを犠牲にします。

グループのビューの一貫性を保ち、任意の時点ですべてのサーバーで使用できるようにする組込みグループメンバーシップサービスがあります。サーバーはグループから退出して参加でき、それに応じてビューが更新されます。サーバーが予期せずグループから離れる場合があります。その場合、障害検出メカニズムはこれを検出し、ビューが変更されたことをグループに通知します。これはすべて自動です。

トランザクションをコミットするには、グループの大部分が、トランザクションのグローバル順序での特定のトランザクションの順序に合意する必要があります。トランザクションをコミットまたは中断するかどうかは、各サーバーによって個別に決定されますが、すべてのサーバーが同じ決定を行います。ネットワークパーティションがあり、メンバーがアグリーメントに到達できない分割になった場合、この問題が解決されるまでシステムは進行しません。したがって、ビルトインの自動スプリットブレイン保護メカニズムもあります。

これらはすべて、提供されている Group Communication System (GCS) プロトコルによって処理されます。これらは、障害検出メカニズム、グループメンバーシップサービス、および安全で完全に順序付けされたメッセージ配信を提供します。これらのプロパティはすべて、サーバーのグループ間でデータが一貫してレプリケートされるようにするシステムの作成に重要です。このテクノロジーの非常にコアには、Paxos アルゴリズムの実装があります。グループ通信エンジンとして機能します。

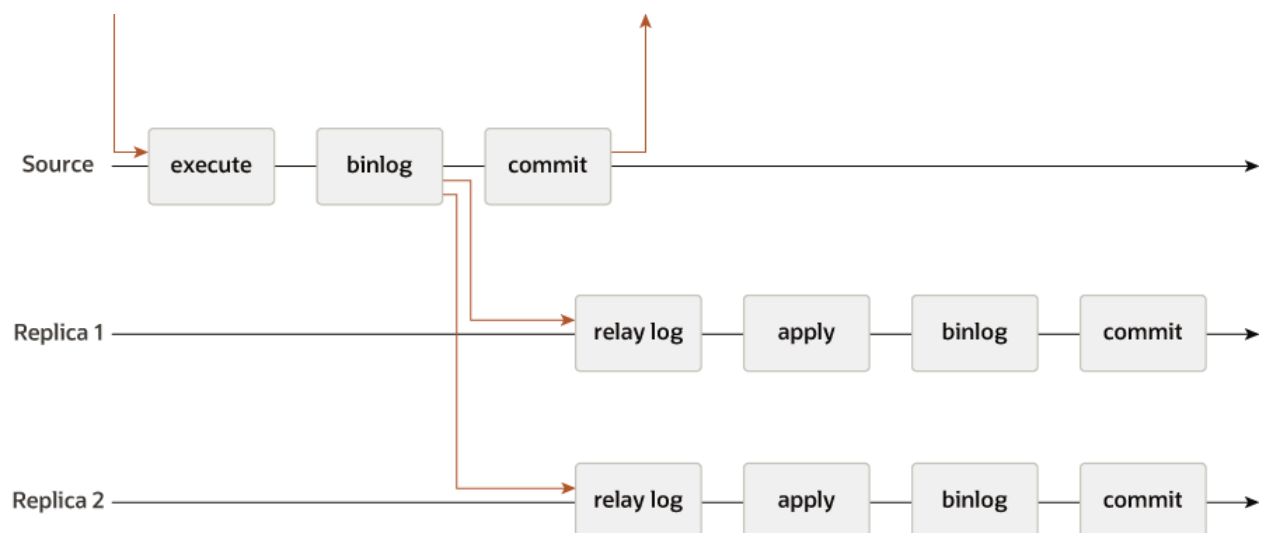
18.1.1 レプリケーションテクノロジー

このセクションでは、MySQL Group Replication の詳細を理解する前に、いくつかのバックグラウンド概念とその仕組みの概要について説明します。これにより、グループレプリケーションに必要なものと、従来の非同期 MySQL レプリケーションとグループレプリケーションの違いを理解するのに役立つコンテキストが提供されます。

18.1.1.1 ソースからレプリカへのレプリケーション

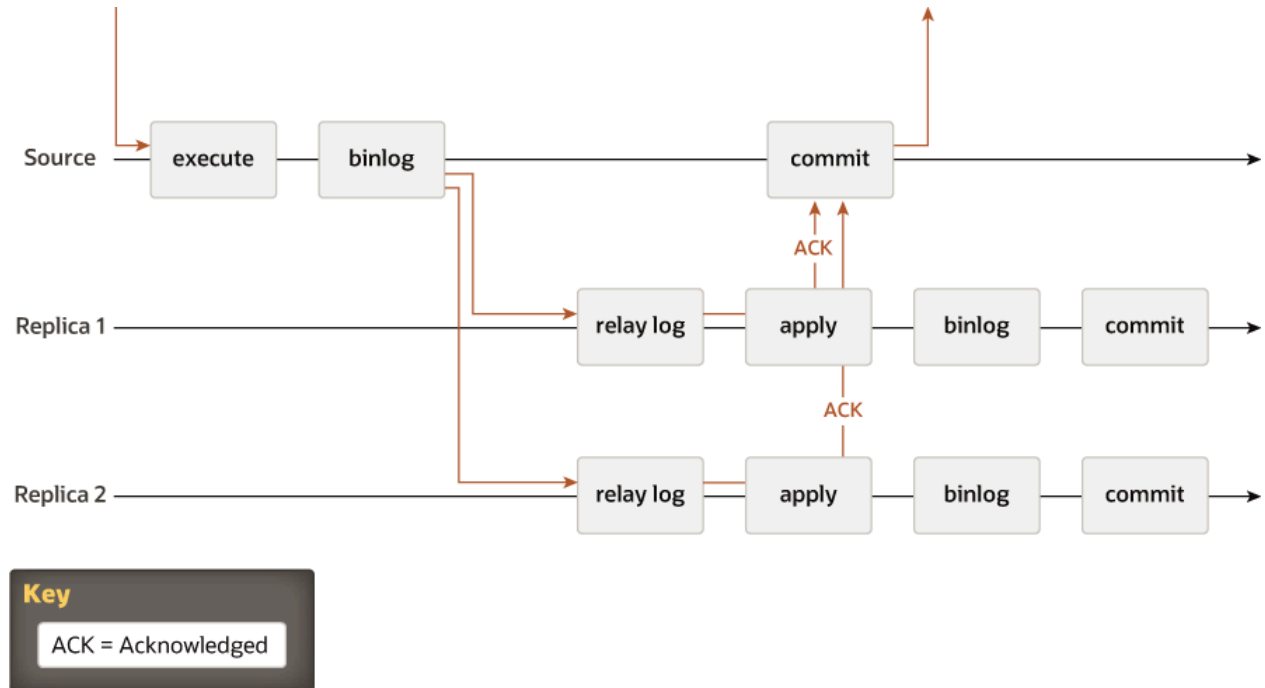
従来の MySQL [Replication](#) は、レプリケーションへの単純なソースからレプリカへのアプローチを提供します。ソースはプライマリであり、セカンダリである 1 つ以上のレプリカがあります。ソースは、トランザクションを適用し、コミットした後、(ステートメントベースレプリケーションで) 再実行されるか、(行ベースレプリケーションで) 適用されるように、あとで (非同期的に) レプリカに送信されます。これはシェアードナッシングシステムであり、すべてのサーバーにデフォルトでデータの完全なコピーがあります。

図 18.1 MySQL 非同期レプリケーション



準同期レプリケーションもあり、プロトコルに1つの同期ステップが追加されます。つまり、プライマリは適用時に、セカンダリがトランザクションを受信済していることを確認するのを待機します。その後のみ、プライマリがコミット操作を再開します。

図 18.2 MySQL 準同期レプリケーション



この2つの図には、従来の非同期MySQLレプリケーションプロトコル(およびその準同期バリエーション)のダイアグラムがあります。異なるインスタンス間の矢印は、サーバー間で交換されるメッセージまたはサーバーとクライアントアプリケーション間で交換されるメッセージを表します。

18.1.1.2 グループレプリケーション

グループレプリケーションは、フォルトトレラントシステムの実装に使用できる手法です。レプリケーショングループは、それぞれ独自のデータの複製全体(共有なしレプリケーションスキーム)を持ち、メッセージを渡して相互作用する一連のサーバーです。通信レイヤーは、アトミックメッセージや合計オーダーメッセージ配信などの一連の保証を提供します。これらは非常に強力なプロパティで、非常に有用な抽象化に変換され、より高度なデータベースレプリケーションソリューションを構築するために再ソートできます。

MySQL Group Replicationは、このようなプロパティおよび抽象化に基づいて構築され、どこにいてもマルチソース更新を実装します。レプリケーショングループは複数のサーバーによって形成され、グループ内の各サーバーはいつでも個別にトランザクションを実行できます。ただし、すべての読み取り/書き込みトランザクションは、グループによって承認された後にのみコミットされます。つまり、読み取り/書き込みトランザクションの場合、グループはコミットするかどうかを決定する必要があるため、コミット操作は元のサーバーからの一方向の決定ではありません。読み取り専用トランザクションでは、グループ内で調整を行う必要はなく、即時にコミットされます。

元のサーバーで読み取り/書き込みトランザクションをコミットする準備が整うと、サーバーは書き込み値(変更された行)および対応する書き込みセット(更新された行の一意の識別子)を原子的にブロードキャストします。トランザクションは原子性ブロードキャストを介して送信されるため、グループ内のすべてのサーバーがトランザクションを受信するか、まったく受信しないかのいずれかです。受信した場合、すべてが以前に送信された他のトランザクションに関して同じ順序で受信します。したがって、すべてのサーバーが同じ順序で同じトランザクションセットを受信し、トランザクションに対してグローバルな合計順序が確立されます。

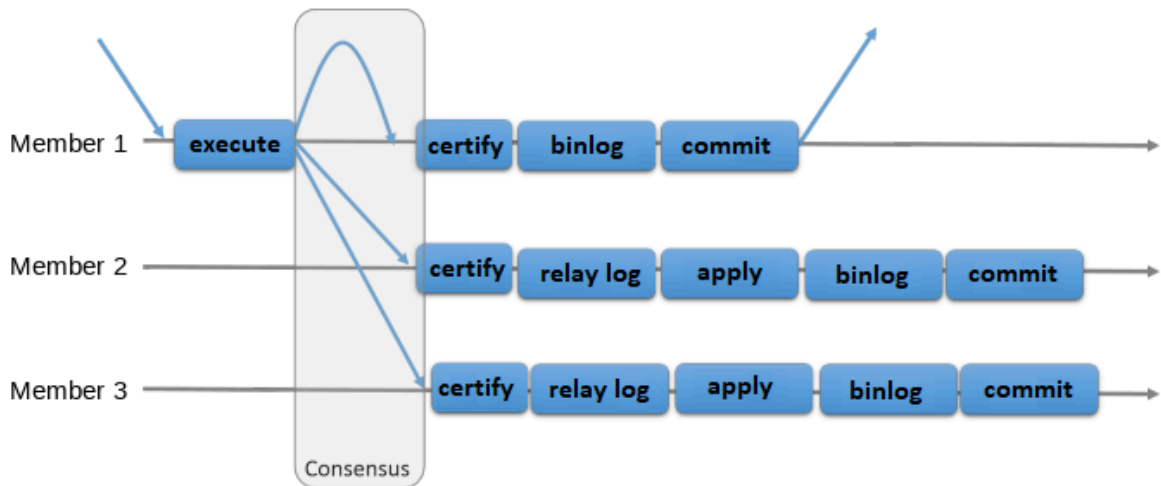
ただし、異なるサーバーで同時に実行されるトランザクション間で競合が発生する可能性があります。このような競合は、動作保証と呼ばれるプロセスで、異なるトランザクションと同時トランザクションの書き込みセットを調べて比較することで検出されます。動作保証中に、競合検出が行レベルで実行されます:異なるサーバーで実行される2つ

の同時トランザクションが同じ行を更新すると、競合が発生します。競合解決手順では、順序付けされたトランザクションが最初にすべてのサーバーでコミットされ、次に順序付けされたトランザクションは中断されるため、元のサーバーでロールバックされ、グループ内の他のサーバーによって削除されます。たとえば、t1 と t2 が異なるサイトで同時に実行され、両方とも同じ行が変更され、t2 が t1 の前に順序付けされている場合、t2 は競合を優先し、t1 はロールバックされます。これは、実際には分散された最初のコミット優先ルールです。2 つのトランザクションが競合の頻度が高くなるようにバインドされている場合は、それらを同じサーバーで起動することをお勧めします。この場合、証明の結果としてロールバックされるのではなく、ローカルロックマネージャで同期化する機会があります。

動作保証されたトランザクションを適用および外部化するために、グループレプリケーションでは、一貫性と有効性が損なわれない場合に、サーバーがトランザクションの合意された順序から逸脱することを許可します。グループレプリケーションは最終的な一貫性システムです。つまり、受信トラフィックの速度が低下または停止するとすぐに、すべてのグループメンバーのデータコンテンツが同じになります。トラフィックのフロー中に、トランザクションを若干異なる順序で外部化したり、一部のメンバーで外部化したりできます。たとえば、マルチプライマリモードでは、証明書の直後にローカルトランザクションが外部化される場合がありますが、以前はグローバル順序になっていたリモートトランザクションはまだ適用されていません。これは、トランザクション間に競合がないことが証明プロセスによって確立された場合に許可されます。単一プライマリモードでは、プライマリサーバーで、競合しない同時ローカルトランザクションが、グループレプリケーションによって合意されたグローバル順序とは異なる順序でコミットおよび外部化される可能性があります。クライアントからの書き込みを受け入れないセカンダリでは、トランザクションは常に合意された順序でコミットおよび外部化されます。

次の図に、MySQL Group Replication プロトコルを示します。これを MySQL Replication (または MySQL 準同期レプリケーション) と比較すると、いくつかの違いがわかります。わかりやすくするために、基礎となるコンセンサスおよび Paxos 関連のメッセージの一部がこの写真にありません。

図 18.3 MySQL グループレプリケーションプロトコル



18.1.2 グループレプリケーションのユースケース

Group Replication を使用すると、システム状態を一連のサーバーにレプリケートすることで、冗長性を備えたフォルトトレラントシステムを作成できます。その後、一部のサーバーで障害が発生した場合でも、すべてでないが大部分であるかぎり、システムは引き続き使用可能です。グループに障害が発生したサーバーの数によっては、パフォーマンスまたはスケーラビリティが低下する可能性があります。サーバー障害は分離され、独立しています。これらは、任意のサーバーがグループから退出したときに通知できる分散障害検出機能に依存するグループメンバーシップサービスによって追跡されます (自発的または予期しない停止のため)。サーバーがグループに参加したときに自動的に最新の状態になるようにする分散リカバリ手順があります。サーバーのフェイルオーバーは必要ありません。また、マルチソース更新では、単一のサーバーに障害が発生した場合にも更新がブロックされないよう

に、あらゆる性質があります。要約すると、MySQL Group Replication では、データベースサービスが継続的に使用可能であることが保証されます。

データベースサービスは使用可能ですが、予期しないサーバーの終了時には、接続しているクライアントを別のサーバーにリダイレクトまたはフェイルオーバーする必要があることを理解することが重要です。これは Group Replication が解決しようとするものではありません。コネクタ、ロードバランサ、ルーターまたはなんらかの形式のミドルウェアは、この問題の処理に適しています。たとえば、[MySQL Router 8.0](#) を参照してください。

要約すると、MySQL Group Replication は、可用性が高く、柔軟性の高い、信頼性の高い MySQL サービスを提供します。

ヒント

MySQL の複数のインスタンスをデプロイするには、[MySQL Shell](#) で MySQL サーバーインスタンスのグループを簡単に管理できるようにする [InnoDB クラスタ](#) を使用できます。InnoDB クラスタは MySQL Group Replication をプログラム環境でラップするため、MySQL インスタンスのクラスタを簡単にデプロイして高可用性を実現できます。また、InnoDB クラスタは [MySQL Router](#) とシームレスにインタフェースするため、アプリケーションは独自のフェイルオーバープロセスを記述せずにクラスタに接続できます。ただし、高可用性を必要としない同様のユースケースでは、[InnoDB ReplicaSet](#) を使用できます。MySQL Shell のインストール手順は、[here](#) にあります。

ユースケースの例

次に、グループレプリケーションの一般的なユースケースの例を示します。

- エラスティックレプリケーション - 非常に流体的なレプリケーションインフラストラクチャを必要とする環境。サーバーの数は、可能なかぎり少ない副作用で動的に増加または縮小する必要があります。たとえば、クラウドのデータベースサービスです。
- 高可用性シャード - シャーディングは、書き込みスケールアウトを実現するための一般的なアプローチです。MySQL Group Replication を使用して、各シャードがレプリケーショングループにマップされる高可用性シャードを実装します。
- 非同期ソース - レプリケーションレプリケーションの代替 - 特定の状況では、単一のソースサーバーを使用すると、単一の競合ポイントになります。グループ全体への書き込みは、特定の状況下でよりスケラブルであることが証明される場合があります。
- 自律システム - また、レプリケーションプロトコルに組み込まれている自動化のためにのみ、MySQL Group Replication をデプロイできます (この章および前の章ですでに説明しています)。

18.1.3 マルチプライマリモードとシングルプライマリモード

グループレプリケーションは、単一プライマリモードまたはマルチプライマリモードで動作します。グループモードは、[group_replication_single_primary_mode](#) システム変数で指定されるグループ全体の構成設定で、すべてのメンバーで同じである必要があります。ON はシングルプライマリモード (デフォルトモード) を意味し、OFF はマルチプライマリモードを意味します。グループのメンバーを異なるモードでデプロイすることはできません。たとえば、あるメンバーをマルチプライマリモードで構成し、別のメンバーをシングルプライマリモードにすることはできません。

グループレプリケーションの実行中は、[group_replication_single_primary_mode](#) の値を手動で変更できません。MySQL 8.0.13 から、[group_replication_switch_to_single_primary_mode\(\)](#) および [group_replication_switch_to_multi_primary_mode\(\)](#) UDF を使用して、グループレプリケーションの実行中にグループをあるモードから別のモードに移動できます。これらの UDF は、グループモードを変更するプロセスを管理し、データの安全性と一貫性を確保します。以前のリリースでは、グループモードを変更するには、グループレプリケーションを停止し、すべてのメンバーで [group_replication_single_primary_mode](#) の値を変更する必要がありました。次に、グループ ([group_replication_bootstrap_group=ON](#) を使用したサーバーによるブートストラップ) の完全再起動を実行して、新しいオペレーティング構成への変更を実装します。サーバーを再起動する必要はありません。

デプロイされたモードに関係なく、グループレプリケーションはクライアント側のフェイルオーバーを処理しません。これは、[MySQL Router 8.0](#)、プロキシ、コネクタまたはアプリケーション自体などのミドルウェアフレームワークによって処理される必要があります。

18.1.3.1 シングルプライマリモード

シングルプライマリモード (`group_replication_single_primary_mode=ON`) では、グループに読み取り/書き込みモードに設定された単一のプライマリサーバーがあります。グループ内の他のすべてのメンバーは読み取り専用モードに設定されます (`super_read_only=ON` を使用)。プライマリは通常、グループをブートストラップする最初のサーバーです。グループに参加する他のすべてのサーバーはプライマリサーバーについて学習し、自動的に読み取り専用モードに設定されます。

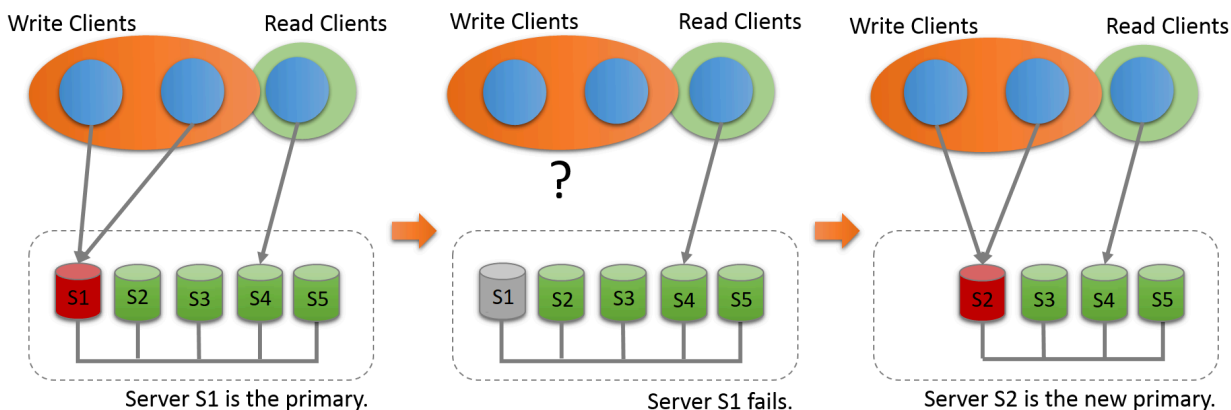
単一プライマリモードでは、グループレプリケーションにより、単一サーバーのみがグループに書き込まれるように強制されるため、マルチプライマリモードと比較すると、一貫性チェックの制限が少なくなり、DDL ステートメントを特別に処理する必要はありません。オプション `group_replication_enforce_update_everywhere_checks` は、グループの厳密な整合性チェックを有効または無効にします。シングルプライマリモードでデプロイする場合、またはグループをシングルプライマリモードに変更する場合は、このシステム変数を `OFF` に設定する必要があります。

プライマリサーバーとして指定されたメンバーは、次の方法で変更できます：

- 既存のプライマリがグループから退出した場合は、自発的か予期せず、新しいプライマリが自動的に選択されます。
- `group_replication_set_as_primary()` UDF を使用して、特定のメンバーを新しいプライマリとして指名できます。
- `group_replication_switch_to_single_primary_mode()` UDF を使用して、マルチプライマリモードで実行されていたグループをシングルプライマリモードで実行するように変更した場合は、新しいプライマリが自動的に選択されるか、UDF で指定して新しいプライマリを指名できます。

UDF は、すべてのグループメンバーが MySQL 8.0.13 以上を実行している場合にのみ使用できます。新しいプライマリサーバーが自動的に選択されるか、手動で指定されると、自動的に読み取り/書き込みモードに設定され、他のグループメンバーはセカンダリとして残り、読み取り専用のままになります。図 18.4 「新規プライマリ選択」にこのプロセスが表示されます。

図 18.4 新規プライマリ選択



新しいプライマリが選択または指名されると、古いプライマリに適用されたが、このサーバーにはまだ適用されていない変更のバックログがある可能性があります。この状況では、新しいプライマリが古いプライマリと捕捉されるまで、読み取り/書き込みトランザクションによって競合が発生してロールバックされ、読み取り専用トランザクションによって読み取りが失効する可能性があります。高速メンバーと低速メンバーの違いを最小限に抑えるグループレプリケーションのフロー制御メカニズムにより、アクティブ化され適切にチューニングされた場合に発生する可能性が軽減されます。フロー制御の詳細は、[セクション 18.6.2 「フロー制御」](#) を参照してください。MySQL 8.0.14 から、`group_replication_consistency` システム変数を使用してトランザクション一貫性のグループレベルを構成し、この問題を回避することもできます。`BEFORE_ON_PRIMARY_FAILOVER` (またはそれ以上の整合性レベル) を設定すると、バックログが適用されるまで、新しく選択されたプライマリに新しいトランザクションが保持されます。トランザクションの一貫性の詳細は、[セクション 18.4.2 「トランザクション一貫性保証」](#) を参照してください。フロー制御およびトランザクションの一貫性保証がグループに使用されていない場合は、クライアントアプリケーションを再ルーティングする前に、新しいプライマリがレプリケーション関連のリレーログを適用するのを待機することをお勧めします。

プライマリ選択アルゴリズム

プライマリメンバーの自動選択プロセスでは、各メンバーがグループの新しいビューを参照し、潜在的な新しいプライマリメンバーを順序付けし、最も適したメンバーを選択します。各メンバーは、MySQL Server リリースのプライマリ選択アルゴリズムに従って、独自の決定をローカルで行います。すべてのメンバーは同じ決定に到達する必要があります。そのため、他のグループメンバーがより低い MySQL Server バージョンを実行している場合、メンバーはプライマリ選択アルゴリズムを調整して、グループ内で最も低い MySQL Server バージョンのメンバーと同じ動作をします。

プライマリを選択するときメンバーが考慮するファクタは、次のとおりです：

1. 最初に考慮される要因は、MySQL Server の最下位バージョンを実行しているメンバーです。すべてのグループメンバーが MySQL 8.0.17 以上を実行している場合、メンバーは最初にそのリリースのバッチバージョンで順序付けされます。MySQL Server 5.7 または MySQL 8.0.16 以下を実行しているメンバーがある場合、メンバーは最初にメジャーバージョンのリリースで順序付けされ、パッチバージョンは無視されます。
2. 複数のメンバーで最低バージョンの MySQL Server が実行されている場合、考慮される 2 番目のファクタは、メンバーの `group_replication_member_weight` システム変数で指定されている各メンバーのメンバーの重みです。このシステム変数を使用できなかった MySQL Server 5.7 をグループのいずれかのメンバーが実行している場合、この係数は無視されます。

`group_replication_member_weight` システム変数は、0-100 の範囲の数値を指定します。すべてのメンバーのデフォルトの重みは 50 であるため、順序を下げるにはこの値より下の重みを設定し、順序を上げるにはその上の重みを設定します。この重み付け機能を使用すると、より適切なハードウェアの使用に優先順位を付けることや、プライマリのスケジュールされたメンテナンス中に特定のメンバーにフェイルオーバーすることができます。

3. 複数のメンバーで最低バージョンの MySQL Server が実行されており、それらのメンバーのうち複数のメンバーの重みが最高である (またはメンバーの重みが無視されている) 場合、3 番目の要因は、`server_uuid` システム変数で指定されているように、各メンバーの生成されたサーバー UUID の辞書順であるとみなされます。サーバー UUID が最も低いメンバーがプライマリとして選択されます。このファクタは、すべてのグループメンバーが重要なファクタによって決定できない場合と同じ決定に到達するように、保証付きで予測可能なタイエリアとして機能します。

プライマリの検索

シングルプライマリモードでデプロイされたときに現在プライマリであるサーバーを確認するには、`performance_schema.replication_group_members` テーブルの `MEMBER_ROLE` カラムを使用します。例：

```
mysql> SELECT MEMBER_HOST, MEMBER_ROLE FROM performance_schema.replication_group_members;
+-----+-----+
| MEMBER_HOST | MEMBER_ROLE |
+-----+-----+
| remote1.example.com | PRIMARY |
| remote2.example.com | SECONDARY |
| remote3.example.com | SECONDARY |
+-----+-----+
```

警告

`group_replication_primary_member` ステータス変数は非推奨になり、将来のバージョンで削除される予定です。

または、`group_replication_primary_member` ステータス変数を使用します。

```
mysql> SHOW STATUS LIKE 'group_replication_primary_member'
```

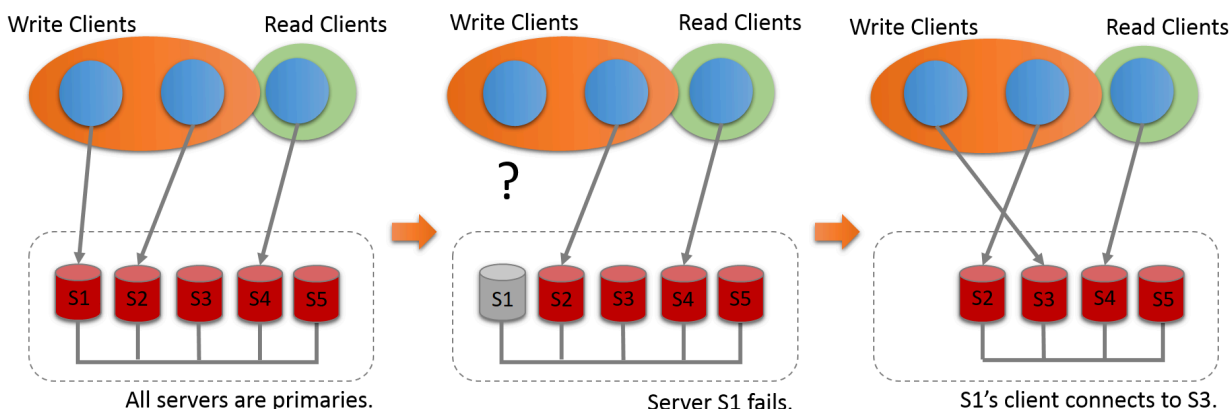
18.1.3.2 マルチプライマリモード

マルチプライマリモード (`group_replication_single_primary_mode=OFF`) では、メンバーに特別なロールはありません。他のグループメンバーと互換性のあるメンバーは、グループへの参加時に読取り/書込みモードに設定され、同時に発行された場合でも書込みトランザクションを処理できます。

たとえば、予期しないサーバーの終了時に書込みトランザクションの受入れを停止したメンバーに接続されているクライアントは、読取り/書込みモードの他のメンバーにリダイレクトまたはフェイルオーバーできます。グループレプリケーションはクライアント側フェイルオーバー自体を処理しないため、MySQL Router 8.0、プロキシ、コネ

クマまたはアプリケーション自体などのミドルウェアフレームワークを使用してこれを調整する必要があります。図 18.5「クライアントフェイルオーバー」は、メンバーがグループを離れた場合に、クライアントが代替グループメンバーに再接続する方法を示します。

図 18.5 クライアントフェイルオーバー



グループレプリケーションは、最終的な一貫性システムです。つまり、受信トラフィックの速度が低下または停止するとすぐに、すべてのグループメンバーのデータコンテンツが同じになります。トラフィックが流れている間、特に一部のメンバーの書き込みスループットが他のメンバーより低い場合、トランザクションを他のメンバーの前に外部化でき、失効した読取りの可能性があります。マルチプライマリモードでは、低速なメンバーがトランザクションの過剰なバックログを構築して認証および適用することもでき、競合および証明の失敗のリスクが高くなります。これらの問題を制限するには、グループレプリケーションのフロー制御メカニズムをアクティブ化およびチューニングして、高速メンバーと低速メンバーの違いを最小限に抑えることができます。フロー制御の詳細は、[セクション 18.6.2「フロー制御」](#)を参照してください。

MySQL 8.0.14 から、グループ内のすべてのトランザクションに対してトランザクションの一貫性を保証する場合は、`group_replication_consistency` システム変数を使用してこれを実行できます。一貫性の向上に必要な同期のパフォーマンスへの影響を考慮して、グループのワークロードおよびデータの読取りおよび書き込みの優先度に応じた設定を選択できます。また、個々のセッションのシステム変数を設定して、特に同時実行に注意が必要なトランザクションを保護することもできます。トランザクションの一貫性の詳細は、[セクション 18.4.2「トランザクション一貫性保証」](#)を参照してください。

トランザクションチェック

グループがマルチプライマリモードでデプロイされている場合、トランザクションはモードと互換性があることを確認するためにチェックされます。グループレプリケーションがマルチプライマリモードでデプロイされると、次の厳密な整合性チェックが行われます：

- `SERIALIZABLE` 分離レベルでトランザクションが実行されると、トランザクション自体をグループと同期するときにコミットが失敗します。
- カスケード制約を持つ外部キーを持つテーブルに対してトランザクションが実行される場合、トランザクション自体をグループと同期するとコミットは失敗します。

チェックは、`group_replication_enforce_update_everywhere_checks` システム変数によって制御されます。マルチプライマリモードでは、システム変数は通常 `ON` に設定する必要がありますが、システム変数を `OFF` に設定することで、オプションでチェックを非アクティブ化できます。シングルプライマリモードでデプロイする場合は、システム変数を `OFF` に設定する必要があります。

データ定義ステートメント

マルチプライマリモードのグループレプリケーショントポロジでは、一般にデータ定義言語 (DDL) と呼ばれるデータ定義ステートメントの実行時に注意する必要があります。

MySQL 8.0 では、完全な DDL ステートメントが単一のアトミックトランザクションとしてコミットまたはロールバックされるアトミックデータ定義言語 (DDL) ステートメントのサポートが導入されています。ただし、DDL ス

テートメント (アトミックまたはそれ以外) は、ステートメントを実行する前に **COMMIT** を実行したかのように、現在のセッションでアクティブなトランザクションを暗黙的に終了します。つまり、DDL ステートメントは、別のトランザクション内、**START TRANSACTION ... COMMIT** などのトランザクション制御ステートメント内、または同じトランザクション内の他のステートメントと組み合わせることはできません。

グループレプリケーションはオプティミスティックレプリケーションパラダイムに基づいており、必要に応じてステートメントがオプティミスティックに実行され、後でロールバックされます。各サーバーは、最初にグループ承諾を保護せずに実行されます。したがって、マルチプライマリモードで DDL ステートメントをレプリケートする場合は、より注意する必要があります。(DDL を使用して) スキーマを変更し、同じオブジェクトに対してオブジェクトに含まれるデータを (DML を使用して) 変更する場合は、スキーマ操作がまだ完了しておらず、どこにもレプリケートされていない間に、同じサーバーを介して変更を処理する必要があります。そうしないと、操作が中断されたとき、または部分的にしか完了しなかったときに、データの不整合が発生する可能性があります。グループがシングルプライマリモードでデプロイされている場合、この問題は発生しません。これは、すべての変更が同じサーバー (プライマリ) を介して実行されるためです。

MySQL 8.0 でのアトミック DDL のサポート、および特定のステートメントのレプリケーションの動作の変更の詳細は、[セクション13.1.1「アトミックデータ定義ステートメントのサポート」](#) を参照してください。

バージョンの互換性

最適な互換性とパフォーマンスを得るには、グループのすべてのメンバーが同じバージョンの MySQL Server を実行する必要があるため、グループレプリケーションを実行する必要があります。マルチプライマリモードでは、通常、すべてのメンバーが読み取り/書き込みモードでグループに参加するため、これはより重要です。グループに複数の MySQL Server バージョンを実行しているメンバーが含まれている場合、一部のメンバーは他のメンバーがサポートしていないか、他のメンバーが持っている関数がないため、他のメンバーと互換性がない可能性があります。これを防ぐために、新しいメンバー (アップグレードおよび再起動された以前のメンバーを含む) が参加すると、メンバーは残りのグループに対して互換性チェックを実行します。

これらの互換性チェックの結果の 1 つは、マルチプライマリモードで特に重要です。参加メンバーが、既存のグループメンバーが実行されている最下位バージョンより上位の MySQL Server バージョンを実行している場合、そのメンバーはグループに参加しますが、読み取り専用モードのままです。(単一プライマリモードで実行されているグループでは、新しく追加されたメンバーは、いずれの場合も読み取り専用でデフォルト設定されます。) MySQL 8.0.17 以上を実行しているメンバーは、互換性をチェックするときにリリースのパッチバージョンを考慮します。MySQL 8.0.16 以下または MySQL 5.7 を実行しているメンバーには、メジャーバージョンのみが考慮されます。

異なる MySQL Server バージョンを使用するメンバーを持つマルチプライマリモードで実行されているグループでは、グループレプリケーションは、MySQL 8.0.17 以上を実行しているメンバーの読み取り/書き込みおよび読み取り専用ステータスを自動的に管理します。メンバーがグループから離れると、現在最も低いバージョンを実行しているメンバーは自動的に読み取り/書き込みモードに設定されます。シングルプライマリモードで実行されていたグループをマルチプライマリモードで実行するように変更すると、`group_replication_switch_to_multi_primary_mode()` UDF を使用して、グループレプリケーションによって自動的にメンバーが正しいモードに設定されます。メンバーは、グループに存在する最低バージョンより上位の MySQL サーバーバージョンを実行しており、最低バージョンを実行しているメンバーが読み取り/書き込みモードになっている場合、自動的に読み取り専用モードになります。

グループ内のバージョンの互換性、およびこれがアップグレードプロセス中にグループの動作に与える影響の詳細は、[セクション18.7.1「グループ内の異なるメンバーバージョンの組合せ」](#) を参照してください。

18.1.4 グループレプリケーションサービス

このセクションでは、Group Replication が構築されるサービスの一部を紹介します。

18.1.4.1 グループメンバーシップ

MySQL Group Replication では、一連のサーバーがレプリケーショングループを形成します。グループには UUID の形式をとる名前があります。このグループは動的であり、サーバーはいつでも (自発的または無関係に) 離脱して参加できます。サーバーが参加または退出するたびに、グループ自体が調整されます。

サーバーがグループに参加すると、既存のサーバーから欠落している状態がフェッチされ、自動的に最新の状態になります。サーバーがグループから離れると (たとえば、メンテナンスのために停止された場合)、残りのサーバーは自動的にグループを離れて再構成したことに気付きます。

グループレプリケーションには、オンラインでグループに参加しているサーバーを定義するグループメンバーシップサービスがあります。オンラインサーバーのリストは view と呼ばれます。グループ内のすべてのサーバーには、特定の時点でグループにアクティブに参加しているサーバーの一貫したビューがあります。

グループメンバーは、トランザクションのコミットのみでなく、現在のビューでも一致する必要があります。既存のメンバーが新しいサーバーをグループの一部にする必要があることに同意すると、そのサーバーを統合するようにグループが再構成され、ビューの変更がトリガーされます。サーバーが自発的にグループから離れるかどうかにかかわらず、グループはその構成を動的に再配置し、ビューの変更がトリガーされます。

メンバーが自発的にグループから離れる場合、最初に動的グループ再構成を開始し、その間、すべてのメンバーはサーバーから離れることなく新しいビューに同意する必要があります。ただし、予期せず停止した場合やネットワーク接続が停止している場合などに、メンバーがグループを離れると、再構成を開始できません。この状況では、グループレプリケーションの障害検出メカニズムは、メンバーが残っている短期間を認識し、障害が発生したメンバーのないグループの再構成を提案します。自発的に退職するメンバーと同様に、再構成にはグループ内の大部分のサーバーからの同意が必要です。ただし、大部分のサーバーがオンラインにならないようにパーティション化されているなどの理由で、グループがアグリーメントに到達できない場合、システムは構成を動的に変更できず、スプリットブレインの状況を防ぐためにブロックされます。この状況では、管理者の介入が必要です。

メンバーが短時間オフラインになってから、障害検出メカニズムによって障害が検出される前、およびグループが再構成されてメンバーが削除される前に、グループへの再参加を再試行できます。この場合、再参加メンバーは以前の状態を忘れますが、他のメンバーがその前の状態を意図したメッセージを送信すると、データの不整合などの問題が発生する可能性があります。この状況のメンバーが XCom コンセンサスプロトコルに参加している場合、障害の前後に異なる決定を行うことで、XCom が同じコンセンサスラウンドに対して異なる値を配信する可能性があります。

この可能性に対処するために、MySQL 5.7.22 および MySQL 8.0 では、グループレプリケーションは、(同じアドレスとポート番号を持つ) 古いインカネーションがメンバーとしてリストされているときに、同じサーバーの新しいインカネーションがグループに参加しようとしている状況をチェックします。新しいインカネーションは、再構成によって古いインカネーションを削除できるようになるまで、グループへの参加をブロックされます。メンバーが削除される前にグループに再接続できるように、待機期間が `group_replication_member_expel_timeout` システム変数によって追加されている場合、疑わしいメンバーが疑わしいタイムアウトの前にグループに再接続すると、疑わしいメンバーが現在のインカネーションとして再度アクティブになる可能性があります。メンバーが `expel` タイムアウトを超えてグループから削除された場合、または `STOP GROUP REPLICATION` ステートメントまたはサーバー障害によってサーバーでグループレプリケーションが停止された場合は、新しいインカネーションとして再結合する必要があります。

18.1.4.2 障害検出

Group Replication には、障害検出メカニズムが含まれています。このメカニズムを使用すると、どのサーバーがサイレントで、どのサーバーが停止していると想定されているかを検出してレポートできます。高レベルでは、障害検出機能は、どのサーバーが停止する可能性があるか(疑わしい)に関する情報を提供する分散サービスです。疑いは、サーバーがミュートされるとトリガーされます。指定した期間中にサーバー A がサーバー B からメッセージを受信しないと、タイムアウトが発生し、疑いが発生します。その後、グループが疑いが真であることに同意した場合、グループは特定のサーバーに実際に障害が発生したと判断します。つまり、グループ内の残りのメンバーは、調整された決定を受けて特定のメンバーを明示します。

サーバーがグループの他の部分から分離されると、他のすべてのサーバーで障害が発生している可能性があります。グループとのアグリーメントを保護できません(クォーラムを保護できないため)。疑いはありません。この方法でサーバーをグループから分離すると、ローカルトランザクションを実行できなくなります。

ネットワークが不安定であり、メンバーが異なる組合せで相互に頻繁に接続を失い、回復する場合、理論的には、グループがすべてのメンバーに強制マークを付けることができます。その後、グループは存在しなくなり、再度設定する必要があります。この可能性に対処するために、MySQL 8.0.20 から、Group Replication Group Communication System (GCS) は説明のためにマークされたグループメンバーを追跡し、大多数があるかどうかを判断する際に疑わしいメンバーのグループ内にあるかのように処理します。これにより、グループに少なくとも 1 つのメンバーが残っていることが保証され、グループは引き続き存在できます。削除されたメンバーがグループから実際に削除されると、GCS は、メンバーに強制マークを付けたレコードを削除して、メンバーがグループに再度参加できるようにします。

障害状況に対する作業グループメンバーのレスポンスを指定するために構成できるグループレプリケーションシステム変数、および障害が疑われるグループメンバーが実行するアクションの詳細は、[セクション 18.6.6 「障害検出およびネットワークパーティション化へのレスポンス」](#) を参照してください。

18.1.4.3 Fault-tolerance

MySQL Group Replication は、Paxos 分散アルゴリズムの実装に基づいて構築され、サーバー間の分散調整を提供します。そのため、定足数に到達して決定するには、大部分のサーバーがアクティブである必要があります。これは、システム自体とその全体的な機能を損なわずに許容できる障害の数に直接影響します。 f 障害を許容するために必要なサーバーの数 (n) は、 $n = 2 \times f + 1$ です。

これは、1つの障害を許容するには、グループに3つのサーバーが含まれている必要があることを意味します。そのため、1台のサーバーで障害が発生した場合でも、大多数のサーバー(3台のうち2台)を形成し、システムが自動的に意思決定を継続して進行できるようにします。ただし、別のサーバーで involuntarily に障害が発生した場合は、決定に到達する大部分がないため、グループがブロックされます(1つのサーバーが残っています)。

次のテーブルに、前述の式を示します。

グループサイズ	過半数	許容される即時障害
1	1	0
2	2	0
3	2	1
4	3	1
5	3	2
6	4	2
7	4	3

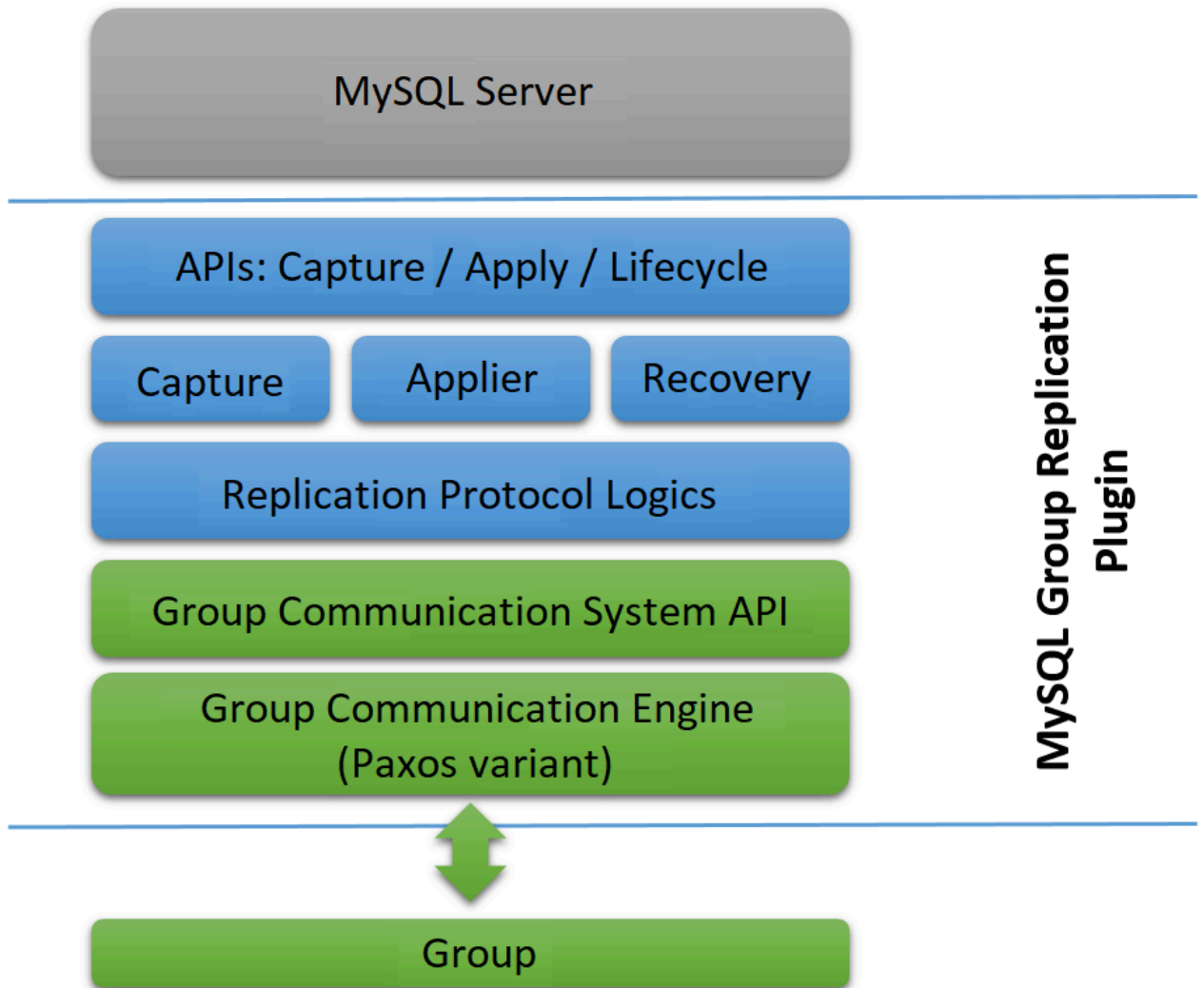
18.1.4.4 可観測性

Group Replication プラグインには多くの自動化が組み込まれています。それでも、バックグラウンドで何が起きているかを理解する必要がある場合があります。ここでは、グループレプリケーションおよびパフォーマンススキーマのインストールメンテーションが重要になります。システム全体の状態(ビュー、競合統計およびサービス状態を含む)は、「パフォーマンススキーマ」テーブルを介してクエリーすることができます。レプリケーションプロトコルの分散特性と、サーバーインスタンスが合意してトランザクションおよびメタデータで同期化されるという事実により、グループの状態を簡単に検査できます。たとえば、グループレプリケーション関連の「パフォーマンススキーマ」テーブルに対して SELECT ステートメントを発行することで、グループ内の単一のサーバーに接続し、ローカル情報とグローバル情報の両方を取得できます。詳細は、[セクション18.3「グループレプリケーションの監視」](#)を参照してください。

18.1.5 グループレプリケーションプラグインのアーキテクチャ

MySQL Group Replication は MySQL プラグインであり、バイナリログ、行ベースのロギング、グローバルトランザクション識別子などの機能を利用して、既存の MySQL レプリケーションインフラストラクチャ上に構築されます。パフォーマンススキーマやプラグイン、サービスインフラストラクチャなど、現在の MySQL フレームワークと統合されます。次の図に、MySQL Group Replication の全体的なアーキテクチャを示すブロック図を示します。

図 18.6 グループ複製プラグインのブロック図



MySQL Group Replication プラグインには、プラグインが MySQL Server とどのように相互作用するかを制御する、取得、適用およびライフサイクル用の一連の API が含まれています。サーバーからプラグインへ、またはその逆への情報フローを行うインタフェースがあります。これらのインタフェースは、MySQL Server コアを Group Replication プラグインから分離し、ほとんどの場合、トランザクション実行パイプラインに配置されるフックです。サーバーからプラグインへの一方向に、サーバーの起動、サーバーのリカバリ、接続を受け入れる準備ができていないサーバー、トランザクションをコミットしようとしているサーバーなどのイベントの通知があります。もう一方の方向では、プラグインは進行中のトランザクションのコミットや中断、リレーログ内のトランザクションのキューイングなどのアクションを実行するようにサーバーに指示します。

Group Replication プラグインアーキテクチャーの次のレイヤーは、通知がルーティングされたときに反応する一連のコンポーネントです。取得コンポーネントは、実行中のトランザクションに関連するコンテキストを追跡します。applier コンポーネントは、データベースでリモートトランザクションを実行します。リカバリコンポーネントは分散リカバリを管理し、ドナーを選択してキャッチアップ手順を管理し、ドナー障害に対応することで、グループに最新の状態で参加しているサーバーを取得します。

スタックを続行すると、レプリケーションプロトコルモジュールにはレプリケーションプロトコルの特定のロジックが含まれます。競合検出を処理し、トランザクションを受信してグループに伝播します。

Group Replication プラグインアーキテクチャーの最後の 2 つのレイヤーは、Group Communication System (GCS) API と Paxos ベースのグループ通信エンジン (XCom) の実装です。GCS API は、レプリケートされた状態マシンの構築

に必要なプロパティを抽象化する高レベルの API です (セクション18.1「グループレプリケーションのバックグラウンド」を参照)。したがって、メッセージングレイヤーの実装がプラグインの残りの上位レイヤーから切り離されます。グループ通信エンジンは、レプリケーショングループのメンバーとの通信を処理します。

18.2 はじめに

MySQL Group Replication は MySQL サーバーへのプラグインとして提供され、グループ内の各サーバーにはプラグインの構成とインストールが必要です。このセクションでは、少なくとも3つのメンバーを含むレプリケーショングループを作成するために必要なステップを含む詳細なチュートリアルを提供します。

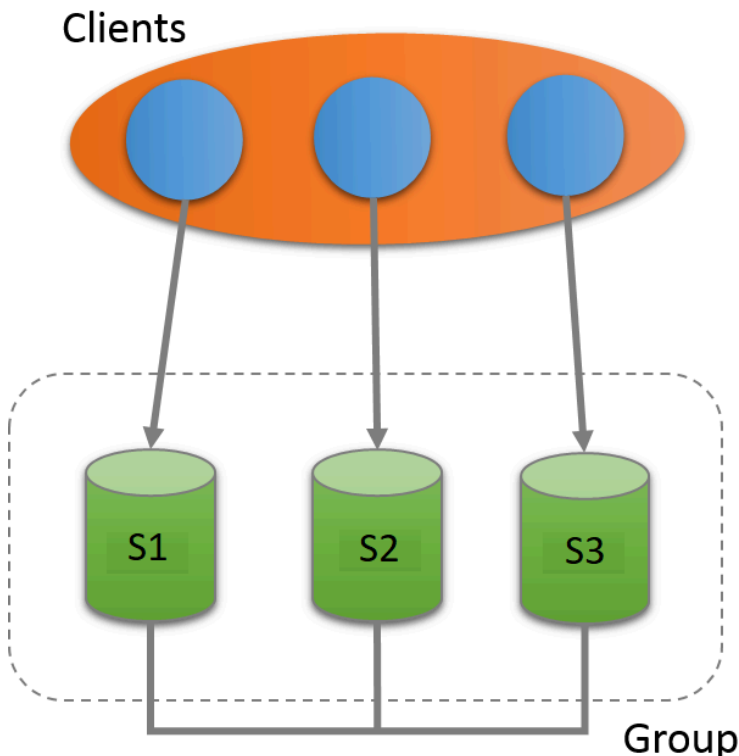
ヒント

MySQL の複数のインスタンスをデプロイするには、MySQL Shell で MySQL サーバーインスタンスのグループを簡単に管理できるようにする [InnoDB クラスター](#) を使用できます。InnoDB クラスターは MySQL Group Replication をプログラム環境でラップするため、MySQL インスタンスのクラスターを簡単にデプロイして高可用性を実現できます。また、InnoDB クラスターは [MySQL Router](#) とシームレスにインターフェースするため、アプリケーションは独自のフェイルオーバープロセスを記述せずにクラスターに接続できます。ただし、高可用性を必要としない同様のユースケースでは、[InnoDB ReplicaSet](#) を使用できます。MySQL Shell のインストール手順は、[here](#) にあります。

18.2.1 単一プライマリモードでのグループレプリケーションのデプロイ

グループ内の各 MySQL サーバーインスタンスは、グループレプリケーションをデプロイするための推奨方法である独立した物理ホストマシン上で実行できます。このセクションでは、それぞれ異なるホストマシンで実行されている3つの MySQL Server インスタンスでレプリケーショングループを作成する方法について説明します。テスト目的など、グループレプリケーションを実行している複数の MySQL サーバーインスタンスを同じホストマシンにデプロイする方法の詳細は、[セクション18.2.2「グループレプリケーションのローカルでのデプロイ」](#) を参照してください。

図 18.7 グループアーキテクチャ



このチュートリアルでは、Group Replication プラグインを使用して MySQL Server を取得および配備する方法、グループを作成する前に各サーバーインスタンスを構成する方法、およびパフォーマンススキーマのモニタリングを使用してすべてが正しく動作していることを確認する方法について説明します。

18.2.1.1 グループレプリケーション用のインスタンスのデプロイ

最初のステップは、MySQL Server の少なくとも 3 つのインスタンスをデプロイすることです。この手順では、s1、s2 および s3 という名前のインスタンスに複数のホストを使用する方法を示します。各ホストに MySQL Server がインストールされていることを前提としています。第2章「MySQL のインストールとアップグレード」を参照してください。Group Replication は、MySQL Server 8.0 で提供される組み込みの MySQL プラグインであるため、追加のインストールは必要ありません。MySQL プラグインの背景情報の詳細は、セクション5.6「MySQL Server プラグイン」を参照してください。

この例では、グループに 3 つのインスタンスが使用されています。これは、グループを作成するための最小インスタンス数です。インスタンスを追加すると、グループのフォルトトレランスが増加します。たとえば、グループが 3 つのメンバーで構成されている場合、1 つのインスタンスで障害が発生しても、グループは続行できます。ただし、別の障害が発生した場合、グループは書き込みトランザクションの処理を続行できなくなります。さらにインスタンスを追加することで、グループがトランザクションの処理を続行している間に失敗する可能性のあるサーバーの数も増加します。グループで使用できるインスタンスの最大数は 9 です。詳細は、セクション18.1.4.2「障害検出」を参照してください。

18.2.1.2 グループレプリケーション用のインスタンスの構成

このセクションでは、グループレプリケーションに使用する MySQL Server インスタンスに必要な構成設定について説明します。背景情報については、セクション18.9「要件と制限事項」を参照してください。

- [Storage Engines \(ストレージエンジン\)](#)
- [レプリケーションフレームワーク](#)
- [グループレプリケーション設定](#)

Storage Engines (ストレージエンジン)

Group Replication の場合、データは InnoDB トランザクションストレージエンジンに格納される必要があります (理由の詳細は、セクション18.9.1「グループレプリケーションの要件」を参照してください)。一時的な MEMORY ストレージエンジンを含むほかのストレージエンジンを使用すると、Group Replication でエラーが発生する可能性があります。disabled_storage_engines システム変数を次のように設定して、使用しないようにします:

```
disabled_storage_engines="MyISAM,BLACKHOLE,FEDERATED,ARCHIVE,MEMORY"
```

MyISAM ストレージエンジンが無効になっている場合、mysql_upgrade がまだ使用されているリリース (MySQL 8.0.16 より前) に MySQL インスタンスをアップグレードすると、mysql_upgrade がエラーで失敗することがあります。これを処理するには、mysql_upgrade の実行中にそのストレージエンジンを再度有効にし、サーバーの再起動時に再度無効にします。詳細は、セクション4.4.5「mysql_upgrade — MySQL テーブルのチェックとアップグレード」を参照してください。

レプリケーションフレームワーク

次の設定では、MySQL Group Replication の要件に従ってレプリケーションを構成します。

```
server_id=1
gtid_mode=ON
enforce_gtid_consistency=ON
```

これらの設定は、一意の識別子番号 1 を使用して [セクション17.1.3「グローバルトランザクション識別子を使用したレプリケーション」](#) を有効にし、GTID を使用して安全にログに記録できるステートメントのみを実行できるようにサーバーを構成します。

MySQL 8.0.20 までは、次の設定も必要です:

```
binlog_checksum=NONE
```

この設定により、バイナリログに書き込まれたイベントのチェックサムが無効になり、デフォルトで有効になります。MySQL 8.0.21 から、Group Replication はバイナリログ内のチェックサムの存在をサポートし、それらを使用し

て一部のチャンネルでのイベントの整合性を検証できるため、デフォルト設定を使用できます。詳細は、[セクション 18.9.2 「グループレプリケーションの制限事項」](#)を参照してください。

レプリケーションのデフォルトが改善された 8.0.3 より前のバージョンの MySQL を使用している場合は、これらの行をメンバーオプションファイルに追加する必要があります。以降のバージョンのオプションファイルにこれらのシステム変数が含まれている場合は、それらが次のように設定されていることを確認します。詳細は、[セクション 18.9.1 「グループレプリケーションの要件」](#)を参照してください。

```
log_bin=binlog
log_slave_updates=ON
binlog_format=ROW
master_info_repository=TABLE
relay_log_info_repository=TABLE
transaction_write_set_extraction=XXHASH64
```

グループレプリケーション設定

この時点で、オプションファイルによってサーバーが構成され、特定の構成でレプリケーションインフラストラクチャをインスタンス化するように指示されます。次のセクションでは、サーバーのグループレプリケーション設定を構成します。

```
plugin_load_add='group_replication.so'
group_replication_group_name="aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa"
group_replication_start_on_boot=off
group_replication_local_address="s1:33061"
group_replication_group_seeds="s1:33061,s2:33061,s3:33061"
group_replication_bootstrap_group=off
```

- `plugin-load-add` は、起動時にサーバーがロードするプラグインのリストに Group Replication プラグインを追加します。これは、プラグインを手動でインストールするよりも本番デプロイメントでお勧めします。
- `group_replication_group_name` の構成は、参加または作成しているグループの名前が "aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa" であることをプラグインに伝えます。

`group_replication_group_name` の値は有効な UUID である必要があります。この UUID は、バイナリログ内のグループレプリケーションイベントの GTID を設定するとき内部的に使用されます。`SELECT UUID()` を使用して UUID を生成できます。

- `off` に `group_replication_start_on_boot` 変数を構成すると、サーバーの起動時に自動的に操作を開始しないようにプラグインに指示されます。これは、プラグインを手動で起動する前にサーバーを構成できるようにするため、Group Replication を設定する際に重要です。メンバーを構成したら、サーバーの起動時にグループレプリケーションが自動的に開始されるように、`group_replication_start_on_boot` を `on` に設定できます。
- `group_replication_local_address` を構成すると、メンバーがグループ内の他のメンバーとの内部通信に使用するネットワークアドレスとポートが設定されます。グループレプリケーションは、グループ通信エンジン (XCom, Paxos バリエーション) のリモートインスタンスを含む内部メンバーからメンバーへの接続にこのアドレスを使用します。

重要

グループレプリケーションのローカルアドレスは、MySQL Server `hostname` および `port` システム変数で定義される SQL クライアント接続に使用されるホスト名およびポートとは異なる必要があります。クライアントアプリケーションには使用しないでください。グループレプリケーションの実行中にグループのメンバー間の内部通信にのみ使用する必要があります。

`group_replication_local_address` によって構成されたネットワークアドレスは、すべてのグループメンバーが解決できる必要があります。たとえば、各サーバーインスタンスが固定ネットワークアドレスを持つ異なるマシン上にある場合、10.0.0.1 などのマシンの IP アドレスを使用できます。ホスト名を使用する場合は、完全修飾名を使用し、DNS、正しく構成された `/etc/hosts` ファイルまたはその他の名前解決プロセスを介して解決できることを確認する必要があります。MySQL 8.0.14 からは、IPv6 アドレス (またはそれらに解決されるホスト名) と IPv4 アドレスを使用できます。グループには、IPv6 を使用するメンバーと IPv4 を使用するメンバーを混在させることができます。IPv6 ネットワークおよび IPv4 と IPv6 の混合グループに対するグループレプリケーションサポートの詳細は、[セクション 18.4.5 「IPv6 および IPv6 と IPv4 の混合グループのサポート」](#)を参照してください。

`group_replication_local_address` の推奨ポートは 33061 です。`group_replication_local_address` は、グループレプリケーションによって、レプリケーショングループ内のグループメンバーの一意的識別子として使用されます。このチュートリアルで示すように、ホスト名または IP アドレスがすべて異なるかぎり、レプリケーショングループのすべてのメンバーに同じポートを使用できます。または、[セクション18.2.2「グループレプリケーションのローカルでのデプロイ」](#) に示すように、ポートがすべて異なるかぎり、すべてのメンバーに同じホスト名または IP アドレスを使用できます。

グループレプリケーション分散リカバリプロセスで既存のメンバーが参加メンバーに提供する接続は、`group_replication_local_address` によって構成されたネットワークアドレスではありません。MySQL 8.0.20 まで、グループメンバーは、MySQL Server `hostname` および `port` システム変数で指定されている分散リカバリのためにメンバーを結合するための標準 SQL クライアント接続を提供します。MySQL 8.0.21 から、グループメンバーは分散リカバリエンドポイントの代替リストをメンバー参加専用のクライアント接続として通知できます。詳細は、[セクション18.4.3.1「分散リカバリの接続」](#) を参照してください。

重要

結合メンバーが、MySQL Server `hostname` システム変数で定義されたホスト名を使用して他のメンバーを正しく識別できない場合、分散リカバリは失敗する可能性があります。MySQL を実行しているオペレーティングシステムでは、DNS またはローカル設定を使用して、一意のホスト名を適切に構成することをお勧めします。サーバーが SQL クライアント接続に使用しているホスト名は、「パフォーマンススキーマ」テーブル `replication_group_members` の `Member_host` カラムで確認できます。複数のグループメンバーがオペレーティングシステムによって設定されたデフォルトのホスト名を外部化する場合、参加メンバーが正しいメンバーアドレスに解決せず、分散リカバリのために接続できない可能性があります。この状況では、MySQL Server `report_host` システム変数を使用して、各サーバーによって外部化される一意のホスト名を構成できます。

- `group_replication_group_seeds` を構成すると、新しいメンバーがグループへの接続を確立するために使用するグループメンバーのホスト名とポートが設定されます。これらのメンバーはシードメンバーと呼ばれます。接続が確立されると、グループメンバーシップ情報が「パフォーマンススキーマ」テーブル `replication_group_members` にリストされます。通常、`group_replication_group_seeds` リストには各グループメンバー `group_replication_local_address` の `hostname:port` が含まれますが、これは義務ではなく、グループメンバーのサブセットをシードとして選択できます。

重要

`group_replication_group_seeds` にリストされている `hostname:port` は、`group_replication_local_address` によって構成されたシードメンバーの内部ネットワークアドレスであり、「パフォーマンススキーマ」テーブル `replication_group_members` などに示されている SQL クライアント接続に使用される `hostname:port` ではありません。

グループを起動するサーバーは、このオプションを使用しません。これは、このオプションが初期サーバーであり、グループのブートストラップを担当しているためです。つまり、グループをブートストラップするサーバー上の既存のデータは、次の結合メンバーのデータとして使用されます。2 つ目のサーバーを結合すると、グループ内の 1 つのメンバーと唯一のメンバーに参加するように要求され、2 つ目のサーバー上の欠落しているデータはブートストラップメンバー上のドナーデータからレプリケートされ、グループが展開されます。3 つ目のサーバー結合

では、これらの2つのいずれかを結合するように要求できます。データは新しいメンバーに同期され、グループが再度展開されます。後続のサーバーは、参加時にこの手順を繰り返します。

警告

複数のサーバーに同時に参加する場合は、グループにすでに存在するシードメンバーを指していることを確認してください。グループに参加しているメンバーは、連絡時にまだグループに属していない可能性があるため、シードとして使用しないでください。

ブートストラップメンバーを最初に起動し、グループを作成することをお勧めします。次に、参加している残りのメンバーのシードメンバーにします。これにより、残りのメンバーを結合するときにグループが形成されます。

グループの作成と複数のメンバーの同時参加はサポートされていません。これは機能する可能性があります。操作が競合してから、グループに参加する操作がエラーまたはタイムアウトで終了する可能性があります。

参加メンバーは、シードメンバーが `group_replication_group_seeds` オプションで通知するプロトコル (IPv4 または IPv6) と同じプロトコルを使用してシードメンバーと通信する必要があります。グループレプリケーションの IP アドレス権限のために、シードメンバーの許可リストには、シードメンバーが提供するプロトコルの参加メンバーの IP アドレス、またはそのプロトコルのアドレスに解決されるホスト名が含まれている必要があります。このアドレスのプロトコルがシードメンバーに通知されたプロトコルと一致しない場合は、参加メンバー `group_replication_local_address` に加えて、このアドレスまたはホスト名を設定して許可する必要があります。参加メンバーに適切なプロトコルの許可されたアドレスがない場合、その接続試行は拒否されます。詳細は、[セクション 18.5.1「グループレプリケーション IP アドレスの権限」](#) を参照してください。

- `group_replication_bootstrap_group` を構成すると、グループをブートストラップするかどうかをプラグインに指示します。この場合、`s1` がグループの最初のメンバーであっても、オプションファイルでこの変数を `off` に設定します。かわりに、インスタンスの実行中に `group_replication_bootstrap_group` を構成して、実際にグループをブートストラップするメンバーが 1 人のみであることを確認します。

重要

`group_replication_bootstrap_group` 変数は、グループに属しているあるサーバーインスタンスでのみ有効にする必要があります。通常は、グループを初めてブートストラップするとき (または、グループ全体を停止して再度起動する場合) に有効にします。グループを複数回ブートストラップする場合 (たとえば、複数のサーバーインスタンスにこのオプションが設定されている場合)、同じ名前の異なる 2 つのグループが存在する人工的な分割プレーンシナリオを作成できます。最初のサーバーインスタンスがオンラインになった後は、必ず `group_replication_bootstrap_group=off` を設定してください。

グループ内のすべてのサーバーの構成は非常に似ています。各サーバー (`server_id`, `datadir`, `group_replication_local_address` など) の詳細を変更する必要があります。これは、このチュートリアルの後半で説明します。

18.2.1.3 分散リカバリのユーザー資格証明

グループレプリケーションでは、分散リカバリプロセスを使用して、グループに参加するときにグループメンバーを同期します。分散リカバリでは、`group_replication_recovery` という名前のレプリケーションチャンネルを使用して、ドナーのバイナリログから参加メンバーにトランザクションを転送します。したがって、グループレプリケーションが直接メンバー間レプリケーションチャンネルを確立できるように、適切な権限を持つレプリケーションユーザーを設定する必要があります。MySQL 8.0.17 から使用可能な分散リカバリの一部としてリモートクローニング操作の使用をサポートするようにグループメンバーが設定されている場合、このレプリケーションユーザーはドナーのクローンユーザーとしても使用され、このロールに対する正しい権限も必要です。分散リカバリの詳細は、[セクション 18.4.3「分散リカバリ」](#) を参照してください。

すべてのグループメンバーで分散リカバリに同じレプリケーションユーザーを使用する必要があります。分散リカバリ用のレプリケーションユーザーを作成するプロセスをバイナリログに取得し、分散リカバリを使用してユーザーの作成に使用されるステートメントをレプリケートできます。または、レプリケーションユーザーを作成する前にバイナリロギングを無効にし、変更が他のサーバーインスタンスに伝播されないようにする場合などは、各メンバーでユーザーを手動で作成することもできます。これを行う場合は、ユーザーを構成したあとでバイナリロギングを再度有効にしてください。

重要

グループの分散リカバリ接続で SSL を使用する場合は、参加メンバーがドナーに接続する前に、各サーバーにレプリケーションユーザーを作成する必要があります。分散リカバリ接続用に SSL を設定し、SSL を必要とするレプリケーションユーザーを作成する手順は、[セクション18.5.3「分散リカバリ接続の保護」](#)を参照してください

重要

デフォルトでは、MySQL 8 で作成されたユーザーは [セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#)を使用します。分散リカバリのレプリケーションユーザーがキャッシュ SHA-2 認証プラグインを使用しており、分散リカバリ接続に SSL を使用していない場合、RSA キーペアがパスワード交換に使用されます。レプリケーションユーザーの公開キーを参加メンバーにコピーするか、リクエスト時に公開キーを提供するようにドナーを構成できます。これを行う手順は、[セクション18.5.3.1「分散リカバリのためのセキュアなユーザー資格証明」](#)を参照してください。

分散リカバリのレプリケーションユーザーを作成するには、次のステップに従います:

1. MySQL サーバーインスタンスを起動し、それにクライアントを接続します。
2. レプリケーションユーザーをインスタンスごとに個別に作成するためにバイナリロギングを無効にする場合は、次のステートメントを発行します:

```
mysql> SET SQL_LOG_BIN=0;
```

3. 分散リカバリに使用する [REPLICATION SLAVE](#) 権限を持つ MySQL ユーザーを作成し、クローニングをサポートするようにサーバーが設定されている場合は、クローニング操作でドナーとして使用する [BACKUP_ADMIN](#) 権限を作成します。この例では、パスワード `password` を持つユーザー `rpl_user` が表示されます。サーバーを構成するときは、適切なユーザー名とパスワードを使用します:

```
mysql> CREATE USER rpl_user@%' IDENTIFIED BY 'password';
mysql> GRANT REPLICATION SLAVE ON *.* TO rpl_user@%';
mysql> GRANT BACKUP_ADMIN ON *.* TO rpl_user@%';
mysql> FLUSH PRIVILEGES;
```

4. バイナリロギングを無効にした場合は、次のステートメントを発行して、ユーザーを作成した直後に再度有効にします:

```
mysql> SET SQL_LOG_BIN=1;
```

5. レプリケーションユーザーを作成したら、分散リカバリで使用するユーザー資格証明をサーバーに指定する必要があります。これを行うには、[CHANGE REPLICATION SOURCE TO](#) ステートメント (MySQL 8.0.23 の場合) または [CHANGE MASTER TO](#) ステートメント (MySQL 8.0.23 の場合) を使用して、ユーザー資格証明を `group_replication_recovery` チャネルの資格証明として設定します。または、MySQL 8.0.21 から、[START GROUP_REPLICATION](#) ステートメントで分散リカバリのユーザー資格証明を指定できます。
 - [CHANGE REPLICATION SOURCE TO](#) | [CHANGE MASTER TO](#) を使用して設定されたユーザー資格証明は、サーバー上のレプリケーションメタデータリポジトリにプレーンテキストで格納されます。これらは、`group_replication_start_on_boot` システム変数が `ON` に設定されている場合は自動起動を含め、グループレプリケーションが開始されるたびに適用されます。
 - [START GROUP_REPLICATION](#) で指定されたユーザー資格証明はメモリーにのみ保存され、[STOP GROUP_REPLICATION](#) ステートメントまたはサーバーの停止によって削除されます。[START GROUP_REPLICATION](#) ステートメントを発行して資格証明を再度指定する必要があるため、これらの資格証明を使用してグループレプリケーションを自動的に開始することはできません。ユーザー資格証明を指定するこの方法は、認可されていないアクセスからグループレプリケーションサーバーを保護するのに役立ちます。

ユーザー資格証明を提供する各方法のセキュリティへの影響の詳細は、[レプリケーションユーザー資格証明のセキュアな提供](#)を参照してください。[CHANGE REPLICATION SOURCE TO](#) | [CHANGE MASTER TO](#) ステートメントを使用してユーザー資格証明を指定する場合は、`rpl_user` および `password` をユーザーの作成時に使用した値に置き換えて、サーバーインスタンスで次のステートメントを発行します:

```
mysql> CHANGE MASTER TO MASTER_USER='rpl_user', MASTER_PASSWORD='password' \\\
```

```
FOR CHANNEL 'group_replication_recovery';

Or from MySQL 8.0.23:
mysql> CHANGE REPLICATION SOURCE TO SOURCE_USER='rpl_user', SOURCE_PASSWORD='password' \\\
FOR CHANNEL 'group_replication_recovery';
```

18.2.1.4 グループレプリケーションの起動

サーバー s1 を構成して起動したら、Group Replication プラグインをインストールします。オプションファイルで `plugin_load_add='group_replication.so'` を使用した場合、Group Replication プラグインがインストールされ、次のステップに進むことができます。プラグインを手動でインストールすることを決定した場合は、サーバーに接続して次を発行します:

```
INSTALL PLUGIN group_replication SONAME 'group_replication.so';
```

重要

Group Replication をロードする前に、`mysql.session` ユーザーが存在している必要があります。`mysql.session` は、MySQL バージョン 8.0.2 で追加されました。データディクショナリが以前のバージョンを使用して初期化された場合は、MySQL のアップグレード手順を実行する必要があります ([セクション 2.11 「MySQL のアップグレード」](#) を参照)。アップグレードが実行されていない場合、Group Replication は起動に失敗し、ユーザーがサーバーにアクセスしようとしたときに `There がエラーメッセージを返します: mysql.session @localhost. ユーザーがサーバーに存在し、mysql_upgrade がサーバー update. の後に実行されたことを確認してください。`

プラグインが正常にインストールされたことを確認するには、`SHOW PLUGINS;` を発行して出力を確認します。次のように表示されます:

```
mysql> SHOW PLUGINS;
+-----+-----+-----+-----+-----+
| Name          | Status | Type          | Library          | License          |
+-----+-----+-----+-----+-----+
| binlog        | ACTIVE | STORAGE ENGINE | NULL             | PROPRIETARY     |
| ...          | ...   | ...           | ...              | ...              |
| group_replication | ACTIVE | GROUP REPLICATION | group_replication.so | PROPRIETARY     |
+-----+-----+-----+-----+-----+
```

18.2.1.5 グループのブートストラップ

グループを初めて起動するプロセスはブートストラップと呼ばれます。グループをブートストラップするには、`group_replication_bootstrap_group` システム変数を使用します。ブートストラップは、単一のサーバー (グループを起動するサーバー) によって一度のみ実行する必要があります。このため、`group_replication_bootstrap_group` オプションの値がインスタンスオプションファイルに格納されませんでした。オプションファイルに保存されている場合、サーバーの再起動時に同じ名前の別のグループが自動的にブートストラップされます。これにより、同じ名前を持つ 2 つの異なるグループが作成されます。このオプションを **ON** に設定してプラグインを停止および再起動する場合も、同じ推論が適用されます。したがって、グループを安全にブートストラップするには、s1 に接続し、次のステートメントを発行します:

```
mysql> SET GLOBAL group_replication_bootstrap_group=ON;
mysql> START GROUP_REPLICATION;
mysql> SET GLOBAL group_replication_bootstrap_group=OFF;
```

または、`START GROUP_REPLICATION` ステートメント (MySQL 8.0.21 から実行可能) で分散リカバリ用のユーザー資格証明を指定する場合は、次のステートメントを発行します:

```
mysql> SET GLOBAL group_replication_bootstrap_group=ON;
mysql> START GROUP_REPLICATION USER='rpl_user', PASSWORD='password';
mysql> SET GLOBAL group_replication_bootstrap_group=OFF;
```

`START GROUP_REPLICATION` ステートメントが戻ると、グループは起動しています。グループが作成され、その中に 1 つのメンバーが存在することを確認できます:

```
mysql> SELECT * FROM performance_schema.replication_group_members;
+-----+-----+-----+-----+-----+
```

CHANNEL_NAME	MEMBER_ID	MEMBER_HOST	MEMBER_PORT	MEMBER_STATE
group_replication_applier	ce9be252-2b71-11e6-b8f4-00212844f856	s1	3306	ONLINE

このテーブルの情報は、一意の識別子 `ce9be252-2b71-11e6-b8f4-00212844f856` を持つメンバーがグループに存在し、それが **ONLINE** であり、`s1` がポート `3306` でクライアント接続をリスニングしていることを確認します。

サーバーが実際にはグループ内にあり、ロードを処理できることを示す目的で、テーブルを作成し、そのテーブルにコンテンツを追加します。

```
mysql> CREATE DATABASE test;
mysql> USE test;
mysql> CREATE TABLE t1 (c1 INT PRIMARY KEY, c2 TEXT NOT NULL);
mysql> INSERT INTO t1 VALUES (1, 'Luis');
```

テーブル `t1` とバイナリログの内容を確認します。

```
mysql> SELECT * FROM t1;
+----+-----+
| c1 | c2 |
+----+-----+
| 1 | Luis |
+----+-----+

mysql> SHOW BINLOG EVENTS;
+-----+-----+-----+-----+-----+-----+
| Log_name | Pos | Event_type | Server_id | End_log_pos | Info |
+-----+-----+-----+-----+-----+-----+
| binlog.000001 | 4 | Format_desc | 1 | 123 | Server ver: 8.0.29-log, Binlog ver: 4 |
| binlog.000001 | 123 | Previous_gtid | 1 | 150 | |
| binlog.000001 | 150 | Gtid | 1 | 211 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaaaaaaaaaa:1' |
| binlog.000001 | 211 | Query | 1 | 270 | BEGIN |
| binlog.000001 | 270 | View_change | 1 | 369 | view_id=14724817264259180:1 |
| binlog.000001 | 369 | Query | 1 | 434 | COMMIT |
| binlog.000001 | 434 | Gtid | 1 | 495 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaaaaaaaaaa:2' |
| binlog.000001 | 495 | Query | 1 | 585 | CREATE DATABASE test |
| binlog.000001 | 585 | Gtid | 1 | 646 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaaaaaaaaaa:3' |
| binlog.000001 | 646 | Query | 1 | 770 | use `test`; CREATE TABLE t1 (c1 INT PRIMARY KEY, c2 TEXT NOT NULL) |
| binlog.000001 | 770 | Gtid | 1 | 831 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaaaaaaaaaa:4' |
| binlog.000001 | 831 | Query | 1 | 899 | BEGIN |
| binlog.000001 | 899 | Table_map | 1 | 942 | table_id: 108 (test.t1) |
| binlog.000001 | 942 | Write_rows | 1 | 984 | table_id: 108 flags: STMT_END_F |
| binlog.000001 | 984 | Xid | 1 | 1011 | COMMIT /* xid=38 */ |
+-----+-----+-----+-----+-----+-----+
```

前述のように、データベースとテーブルオブジェクトが作成され、対応する DDL ステートメントがバイナリログに書き込まれました。また、データはテーブルに挿入されてバイナリログに書き込まれたため、ドナーのバイナリログからの状態転送によって分散回復に使用できます。

18.2.1.6 グループへのインスタンスの追加

この時点で、グループにはサーバー `s1` というメンバーがあり、サーバー `s1` にはデータが含まれています。ここで、以前に構成した他の 2 つのサーバーを追加して、グループを拡張します。

2 番目のインスタンスの追加

別のインスタンスであるサーバー `s2` を追加するには、最初にその構成ファイルを作成します。この構成は、`server_id` などを除き、サーバー `s1` に使用される構成と似ています。これらの異なる行は、次のリストで強調表示されています。

```
[mysqld]
#
# Disable other storage engines
#
disabled_storage_engines="MyISAM,BLACKHOLE,FEDERATED,ARCHIVE,MEMORY"
#
# Replication configuration parameters
#
```



```
server_id=2
gtid_mode=ON
enforce_gtid_consistency=ON
binlog_checksum=NONE      # Not needed from 8.0.21

#
# Group Replication configuration
#
plugin_load_add='group_replication.so'
group_replication_group_name="aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa"
group_replication_start_on_boot=off
group_replication_local_address="s2:33061"
group_replication_group_seeds="s1:33061,s2:33061,s3:33061"
group_replication_bootstrap_group=off
```

サーバー s1 の手順と似ていますが、サーバーを起動するオプションファイルが配置されています。次に、分散リカバリ資格証明を次のように構成します。これらのコマンドは、ユーザーがグループ内で共有されるため、サーバー s1 の設定時に使用されるコマンドと同じです。このメンバーは、[セクション18.2.1.3「分散リカバリのユーザー資格証明」](#) で同じレプリケーションユーザーが構成されている必要があります。分散リカバリを使用してすべてのメンバーでユーザーを構成する場合、s2 がシード s1 に接続すると、レプリケーションユーザーは s1 にレプリケートまたはクローニングされます。s1 でユーザー資格証明を構成したときにバイナリロギングを有効にしておらず、リモートクローニング操作が状態転送に使用されない場合は、s2 でレプリケーションユーザーを作成する必要があります。この場合は、s2 に接続して次のコマンドを発行します：

```
SET SQL_LOG_BIN=0;
CREATE USER rpl_user@%' IDENTIFIED BY 'password';
GRANT REPLICATION SLAVE ON *.* TO rpl_user@%';
GRANT BACKUP_ADMIN ON *.* TO rpl_user@%';
FLUSH PRIVILEGES;
SET SQL_LOG_BIN=1;
```

[CHANGE REPLICATION SOURCE TO](#) | [CHANGE MASTER TO](#) ステートメントを使用してユーザー資格証明を指定する場合は、その後に次のステートメントを発行します：

```
CHANGE MASTER TO MASTER_USER='rpl_user', MASTER_PASSWORD='password' \
FOR CHANNEL 'group_replication_recovery';

Or from MySQL 8.0.23:
CHANGE REPLICATION SOURCE TO SOURCE_USER='rpl_user', SOURCE_PASSWORD='password' \
FOR CHANNEL 'group_replication_recovery';
```

ヒント

MySQL 8 のデフォルトであるキャッシュ SHA-2 認証プラグインを使用している場合は、[キャッシュ SHA-2 認証プラグインを使用するレプリケーションユーザー](#) を参照してください。

必要に応じて、Group Replication プラグインをインストールします。[セクション18.2.1.4「グループレプリケーションの起動」](#) を参照してください。

グループレプリケーションを開始し、s2 がグループに参加するプロセスを開始します。

```
mysql> START GROUP_REPLICATION;
```

または、[START GROUP_REPLICATION](#) ステートメント (MySQL 8.0.21 から) で分散リカバリ用のユーザー資格証明を指定する場合は、次の手順を実行します：

```
mysql> START GROUP_REPLICATION USER='rpl_user', PASSWORD='password';
```

s1 で実行されたステップと同じステップとは異なり、グループはすでに存在するため、グループをブートストラップする必要はありません。つまり、s2 `group_replication_bootstrap_group` では `OFF` に設定されており、グループレプリケーションを開始する前に `SET GLOBAL group_replication_bootstrap_group=ON;` を発行しません。これは、グループがすでにサーバー s1 によって作成およびブートストラップされているためです。この時点で、s2 は既存のグループにのみ追加する必要があります。

ヒント

グループレプリケーションが正常に開始され、サーバーがグループに参加すると、`super_read_only` 変数がチェックされます。メンバー構成ファイルで `super_read_only`

を ON に設定すると、なんらかの理由でグループレプリケーションの起動時に失敗したサーバーがトランザクションを受け入れないようにすることができます。サーバーが読取り/書き込みインスタンスとしてグループに参加する必要がある場合 (たとえば、単一プライマリグループのプライマリまたはマルチプライマリグループのメンバーとして)、`super_read_only` 変数が ON に設定されていると、グループへの参加時に OFF に設定されます。

`performance_schema.replication_group_members` テーブルを再度チェックすると、グループに 2 つの ONLINE サーバーが存在することがわかります。

```
mysql> SELECT * FROM performance_schema.replication_group_members;
+-----+-----+-----+-----+-----+
| CHANNEL_NAME | MEMBER_ID | MEMBER_HOST | MEMBER_PORT | MEMBER_STATE |
+-----+-----+-----+-----+-----+
| group_replication_applier | 395409e1-6dfa-11e6-970b-00212844f856 | s1 | 3306 | ONLINE |
| group_replication_applier | ac39f1e6-6dfa-11e6-a69d-00212844f856 | s2 | 3306 | ONLINE |
+-----+-----+-----+-----+-----+
```

s2 がグループに参加しようとする時、[セクション18.4.3「分散リカバリ」](#) は、s1 が適用したのと同じトランザクションを s2 が適用したことを確認しました。このプロセスが完了すると、s2 はグループをメンバーとして参加でき、この時点で ONLINE としてマークされます。つまり、サーバー s1 を自動的に捕捉しておく必要があります。s2 が ONLINE になると、グループとのトランザクションの処理が開始されます。次のように、s2 がサーバー s1 と実際に同期されていることを確認します。

```
mysql> SHOW DATABASES LIKE 'test';
+-----+
| Database (test) |
+-----+
| test |
+-----+

mysql> SELECT * FROM test.t1;
+----+----+
| c1 | c2 |
+----+----+
| 1 | Luis |
+----+----+

mysql> SHOW BINLOG EVENTS;
+-----+-----+-----+-----+-----+-----+
| Log_name | Pos | Event_type | Server_id | End_log_pos | Info |
+-----+-----+-----+-----+-----+-----+
| binlog.000001 | 4 | Format_desc | 2 | 123 | Server ver: 8.0.29-log, Binlog ver: 4 |
| binlog.000001 | 123 | Previous_gtid | 2 | 150 | |
| binlog.000001 | 150 | Gtid | 1 | 211 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:1' |
| binlog.000001 | 211 | Query | 1 | 270 | BEGIN |
| binlog.000001 | 270 | View_change | 1 | 369 | view_id=14724832985483517:1 |
| binlog.000001 | 369 | Query | 1 | 434 | COMMIT |
| binlog.000001 | 434 | Gtid | 1 | 495 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:2' |
| binlog.000001 | 495 | Query | 1 | 585 | CREATE DATABASE test |
| binlog.000001 | 585 | Gtid | 1 | 646 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:3' |
| binlog.000001 | 646 | Query | 1 | 770 | use `test`; CREATE TABLE t1 (c1 INT PRIMARY KEY, c2 TEXT NOT NULL) |
| binlog.000001 | 770 | Gtid | 1 | 831 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:4' |
| binlog.000001 | 831 | Query | 1 | 890 | BEGIN |
| binlog.000001 | 890 | Table_map | 1 | 933 | table_id: 108 (test.t1) |
| binlog.000001 | 933 | Write_rows | 1 | 975 | table_id: 108 flags: STMT_END_F |
| binlog.000001 | 975 | Xid | 1 | 1002 | COMMIT /* xid=30 */ |
| binlog.000001 | 1002 | Gtid | 1 | 1063 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:5' |
| binlog.000001 | 1063 | Query | 1 | 1122 | BEGIN |
| binlog.000001 | 1122 | View_change | 1 | 1261 | view_id=14724832985483517:2 |
| binlog.000001 | 1261 | Query | 1 | 1326 | COMMIT |
+-----+-----+-----+-----+-----+-----+
```

前述のように、2 つ目のサーバーがグループに追加され、サーバー s1 から変更が自動的にレプリケートされました。つまり、s2 がグループに参加した時点までに s1 で適用されたトランザクションは、s2 にレプリケートされていません。

インスタンスの追加

グループへのインスタンスの追加は、サーバー s2 の場合と同様に構成を変更する必要があることを除き、基本的には 2 つ目のサーバーの追加と同じステップのシーケンスです。必要なコマンドを要約するには:

1. 構成ファイルを作成します。

```
[mysqld]

#
# Disable other storage engines
#
disabled_storage_engines="MyISAM,BLACKHOLE,FEDERATED,ARCHIVE,MEMORY"

#
# Replication configuration parameters
#
server_id=3
gtid_mode=ON
enforce_gtid_consistency=ON
binlog_checksum=NONE      # Not needed from 8.0.21

#
# Group Replication configuration
#
plugin_load_add='group_replication.so'
group_replication_group_name="aaaaaaaa-aaaa-aaaa-aaaaaaaaaaaa"
group_replication_start_on_boot=off
group_replication_local_address= "s3:33061"
group_replication_group_seeds= "s1:33061,s2:33061,s3:33061"
group_replication_bootstrap_group= off
```

2. サーバーを起動して接続します。分散リカバリ用のレプリケーションユーザーを作成します。

```
SET SQL_LOG_BIN=0;
CREATE USER rpl_user@'%' IDENTIFIED BY 'password';
GRANT REPLICATION SLAVE ON *.* TO rpl_user@'%';
GRANT BACKUP_ADMIN ON *.* TO rpl_user@'%';
FLUSH PRIVILEGES;
SET SQL_LOG_BIN=1;
```

[CHANGE REPLICATION SOURCE TO](#) | [CHANGE MASTER TO](#) ステートメントを使用してユーザー資格証明を指定する場合は、その後次のステートメントを発行します:

```
CHANGE MASTER TO MASTER_USER='rpl_user', MASTER_PASSWORD='password' \\\
FOR CHANNEL 'group_replication_recovery';
```

Or from MySQL 8.0.23:
[CHANGE REPLICATION SOURCE TO SOURCE_USER='rpl_user', SOURCE_PASSWORD='password' \\\](#)
[FOR CHANNEL 'group_replication_recovery';](#)

3. 必要に応じて、Group Replication プラグインをインストールします。

```
INSTALL PLUGIN group_replication SONAME 'group_replication.so';
```

4. グループレプリケーションを開始します。

```
mysql> START GROUP_REPLICATION;
```

または、[START GROUP_REPLICATION](#) ステートメント (MySQL 8.0.21 から) で分散リカバリ用のユーザー資格証明を指定する場合は、次の手順を実行します:

```
mysql> START GROUP_REPLICATION USER='rpl_user', PASSWORD='password';
```

この時点で、s3 は起動して実行され、グループに参加して、グループ内の他のサーバーと捕捉されています。[performance_schema.replication_group_members](#) テーブルを再度参照して、これが当てはまることを確認します。

```
mysql> SELECT * FROM performance_schema.replication_group_members;
+-----+-----+-----+-----+-----+
| CHANNEL_NAME | MEMBER_ID | MEMBER_HOST | MEMBER_PORT | MEMBER_STATE |
+-----+-----+-----+-----+-----+
| group_replication_applier | 395409e1-6dfa-11e6-970b-00212844f856 | s1 | 3306 | ONLINE |
| group_replication_applier | 7eb217ff-6df3-11e6-966c-00212844f856 | s3 | 3306 | ONLINE |
| group_replication_applier | ac39f1e6-6dfa-11e6-a69d-00212844f856 | s2 | 3306 | ONLINE |
+-----+-----+-----+-----+-----+
```

サーバー s2 またはサーバー s1 でこの同じクエリーを発行すると、同じ結果になります。また、サーバー s3 が捕捉されたことを確認できます:

```
mysql> SHOW DATABASES LIKE 'test';
+-----+
| Database (test) |
+-----+
| test           |
+-----+

mysql> SELECT * FROM test.t1;
+----+----+
| c1 | c2 |
+----+----+
| 1  | Luis |
+----+----+

mysql> SHOW BINLOG EVENTS;
+-----+-----+-----+-----+-----+-----+
| Log_name | Pos | Event_type | Server_id | End_log_pos | Info |
+-----+-----+-----+-----+-----+-----+
| binlog.000001 | 4 | Format_desc | 3 | 123 | Server ver: 8.0.29-log, Binlog ver: 4 |
| binlog.000001 | 123 | Previous_gtid | 3 | 150 | |
| binlog.000001 | 150 | Gtid | 1 | 211 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaaaaaaaaaa:1' |
| binlog.000001 | 211 | Query | 1 | 270 | BEGIN |
| binlog.000001 | 270 | View_change | 1 | 369 | view_id=14724832985483517:1 |
| binlog.000001 | 369 | Query | 1 | 434 | COMMIT |
| binlog.000001 | 434 | Gtid | 1 | 495 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaaaaaaaaaa:2' |
| binlog.000001 | 495 | Query | 1 | 585 | CREATE DATABASE test |
| binlog.000001 | 585 | Gtid | 1 | 646 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaaaaaaaaaa:3' |
| binlog.000001 | 646 | Query | 1 | 770 | use `test`; CREATE TABLE t1 (c1 INT PRIMARY KEY, c2 TEXT NOT NULL) |
| binlog.000001 | 770 | Gtid | 1 | 831 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaaaaaaaaaa:4' |
| binlog.000001 | 831 | Query | 1 | 890 | BEGIN |
| binlog.000001 | 890 | Table_map | 1 | 933 | table_id: 108 (test.t1) |
| binlog.000001 | 933 | Write_rows | 1 | 975 | table_id: 108 flags: STMT_END_F |
| binlog.000001 | 975 | Xid | 1 | 1002 | COMMIT /* xid=29 */ |
| binlog.000001 | 1002 | Gtid | 1 | 1063 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaaaaaaaaaa:5' |
| binlog.000001 | 1063 | Query | 1 | 1122 | BEGIN |
| binlog.000001 | 1122 | View_change | 1 | 1261 | view_id=14724832985483517:2 |
| binlog.000001 | 1261 | Query | 1 | 1326 | COMMIT |
| binlog.000001 | 1326 | Gtid | 1 | 1387 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaaaaaaaaaa:6' |
| binlog.000001 | 1387 | Query | 1 | 1446 | BEGIN |
| binlog.000001 | 1446 | View_change | 1 | 1585 | view_id=14724832985483517:3 |
| binlog.000001 | 1585 | Query | 1 | 1650 | COMMIT |
+-----+-----+-----+-----+-----+-----+
```

18.2.2 グループレプリケーションのローカルでのデプロイ

グループレプリケーションをデプロイする最も一般的な方法は、複数のサーバーインスタンスを使用して高可用性を提供することです。テスト目的などで、グループレプリケーションをローカルにデプロイすることもできます。このセクションでは、グループレプリケーションをローカルにデプロイする方法について説明します。

重要

グループレプリケーションは、高可用性が保証されるため、通常は複数のホストにデプロイされます。すべての MySQL サーバーインスタンスが同じ単一ホスト上で実行されているため、このセクションの手順は本番デプロイメントには適していません。このホストに障害が発生した場合、グループ全体が失敗します。したがって、この情報はテスト目的で使用する必要があり、本番環境では使用しないでください。

このセクションでは、1つの物理マシン上に3つの MySQL Server インスタンスを持つレプリケーショングループを作成する方法について説明します。これは、3つのデータディレクトリ(サーバーインスタンスごとに1つ)が必要であり、各インスタンスを個別に構成する必要があることを意味します。これは - この手順では、MySQL Server がダウンロードおよび解凍されていることを前提としています - `mysql-8.0` という名前のディレクトリに格納します。各 MySQL サーバーインスタンスには、特定のデータディレクトリが必要です。 `data` という名前のディレクトリを作成し、そのディレクトリに各サーバーインスタンス (s1、s2 および s3 など) のサブディレクトリを作成して、それぞれを初期化します。

```
mysql-8.0/bin/mysqld --initialize-insecure --basedir=$PWD/mysql-8.0 --datadir=$PWD/data/s1
```

```
mysql-8.0/bin/mysqld --initialize-insecure --basedir=$PWD/mysql-8.0 --datadir=$PWD/data/s2  
mysql-8.0/bin/mysqld --initialize-insecure --basedir=$PWD/mysql-8.0 --datadir=$PWD/data/s3
```

data/s1, data/s2, data/s3 内には初期化されたデータディレクトリがあり、mysql システムデータベースや関連テーブルなどが含まれています。初期化手順の詳細は、[セクション2.10.1「データディレクトリの初期化」](#)を参照してください。

警告

本番環境では `-initialize-insecure` を使用しないでください。チュートリアルを簡略化するためにここでのみ使用します。セキュリティ設定の詳細は、[セクション18.5「グループレプリケーションセキュリティ」](#)を参照してください。

ローカルグループレプリケーションメンバーの構成

[セクション18.2.1.2「グループレプリケーション用のインスタンスの構成」](#) をフォローしている場合、前のセクションで追加したデータディレクトリの構成を追加する必要があります。例:

```
[mysqld]  
  
# server configuration  
datadir=<full_path_to_data>/data/s1  
basedir=<full_path_to_bin>/mysql-8.0/  
  
port=24801  
socket=<full_path_to_sock_dir>/s1.sock
```

これらの設定では、以前に作成したデータディレクトリと、サーバーが受信接続のリスニングを開始するポートを使用するように MySQL サーバーを構成します。

注記

このチュートリアルでは、3つのサーバーインスタンスが同じホスト名を使用するため、24801のデフォルト以外のポートが使用されます。3つの異なるマシンがあるセットアップでは、これは必要ありません。

グループレプリケーションでは、メンバー間のネットワーク接続が必要です。つまり、各メンバーは他のすべてのメンバーのネットワークアドレスを解決する必要があります。たとえば、このチュートリアルでは、3つのインスタンスすべてが1つのマシンで実行されるため、メンバーが相互に接続できるように、`report_host=127.0.0.1`などのオプションファイルに行を追加できます。

その後、各メンバーは `group_replication_local_address` 上の他のメンバーに接続できる必要があります。たとえば、メンバー s1 のオプションファイルで、次のように追加します:

```
group_replication_local_address="127.0.0.1:24901"  
group_replication_group_seeds="127.0.0.1:24901,127.0.0.1:24902,127.0.0.1:24903"
```

これにより、シードメンバーとの内部グループ通信にポート 24901 を使用するように s1 が構成されます。グループに追加するサーバーインスタンスごとに、メンバーのオプションファイルでこれらの変更を行います。メンバーごとに一意のアドレスを指定する必要があるため、`group_replication_local_address` のインスタンスごとに一意のポートを使用します。通常、すべてのメンバーを、グループに参加しており、グループで処理されたトランザクションを取得していないメンバーのシードとして機能させる必要があります。この場合は、前述のように、すべてのポートを `group_replication_group_seeds` に追加します。

[セクション18.2.1「単一プライマリモードでのグループレプリケーションのデプロイ」](#)の残りのステップは、この方法でローカルにデプロイしたグループにも同様に適用されます。

18.3 グループレプリケーションの監視

「パフォーマンススキーマ」が有効になっている場合は、「パフォーマンススキーマ」テーブルを使用してグループレプリケーションを監視します。グループレプリケーションにより、次のテーブルが追加されます:

- `performance_schema.replication_group_member_stats`
- `performance_schema.replication_group_members`

これらのパフォーマンススキーマレプリケーションテーブルには、グループレプリケーションに関する情報も表示されます:

- `performance_schema.replication_connection_status` には、グループから受信してアプライヤキュー (リレーログ) にキューに入れられたトランザクションなど、グループレプリケーションに関する情報が表示されます。
- `performance_schema.replication_applier_status` には、Group Replication 関連のチャンネルおよびスレッドの状態が表示されます。トランザクションを適用するワーカースレッドが多数ある場合は、ワーカーテーブルを使用して、各ワーカースレッドの実行内容を監視することもできます。

Group Replication プラグインによって作成されるレプリケーションチャンネルの名前は次のとおりです:

- `group_replication_recovery` - このチャンネルは、分散リカバリフェーズに関連するレプリケーション変更に使用されます。
- `group_replication_applier` - このチャンネルは、グループからの着信変更に使用されます。これは、グループから直接取得したトランザクションを適用するために使用されるチャンネルです。

MySQL 8.0.21 からは、エラー以外の状況のグループレプリケーションライフサイクルイベントはシステムメッセージとして分類され、レプリケーショングループメンバーのサーバーエラーログに常に記録されます。この情報を使用して、レプリケーショングループ内のサーバーメンバーシップの履歴を確認できます。以前のリリースでは、エラー以外の状況のグループレプリケーションライフサイクルイベントは情報メッセージとして分類され、サーバーに `log_error_verbosity` レベル 3 を指定することでエラーログに追加できました。

グループ全体に影響するライフサイクルイベントの中には、グループで **ONLINE** ステータスになった新しいメンバーやプライマリ選択など、すべてのグループメンバーに記録されるものがあります。他のイベントは、メンバーで有効化または無効化されているスーパー読み取り専用モードやグループを離れたメンバーなど、発生したメンバーにのみ記録されます。頻繁に発生した場合に問題を示すことができる多数のライフサイクルイベントは、メンバーが到達不能になり再度到達可能になるなどの警告メッセージとしてログに記録され、メンバーはバイナリログまたはリモートクローニング操作からの状態転送によって分散リカバリを開始します。

次の各セクションでは、グループレプリケーションで使用可能な監視情報を解釈する方法について説明します。

18.3.1 グループレプリケーションサーバーの状態

サーバーインスタンスには様々な状態があります。サーバーが適切に通信している場合、すべてのサーバーで同じ状態が報告されます。ただし、ネットワークパーティションがある場合、またはサーバーがグループから離れる場合は、クエリー対象のサーバーに応じて異なる情報がレポートされる可能性があります。サーバーがグループを離れた場合、他のサーバーの状態に関する更新された情報をレポートすることはできません。クォーラムが失われるようなパーティションがある場合、サーバーはそれ自体を調整できません。その結果、異なるサーバーのステータスを推測できません。したがって、状態を推測するのではなく、一部のサーバーにアクセスできないことが報告されます。

表 18.1 サーバーの状態

Field	説明	グループ同期済
ONLINE	メンバーは、完全に機能するグループメンバーとして機能する準備ができています。つまり、クライアントはトランザクションの接続および実行を開始できます。	はい
RECOVERING	メンバーはグループのアクティブメンバーになる処理中で、現在リカバリプロセスを実行中で、ドナーから状態転送を受信しています。	いいえ
OFFLINE	Group Replication プラグインがロードされましたが、メンバーはどのグループにも属していません。	いいえ
ERROR	メンバーはエラー状態であり、グループメンバーとして	いいえ

Field	説明	グループ同期済
	正しく機能していません。 group_replication_exit_state_action によって設定された終了アクションに応じて、メンバーは読取り専用モード (super_read_only=ON) であり、オフラインモード (offline_mode=ON) にすることもできます。 OFFLINE_MODE 終了アクションに続くオフラインモードのサーバーは、 OFFLINE ではなく ERROR ステータスで表示されることに注意してください。終了アクションが ABORT_SERVER のサーバーが停止し、グループのビューから削除されます。	
UNREACHABLE	ローカル障害検出機能は、たとえば切断されたために特定のサーバーにアクセスできないと疑われる場合は常に、そのサーバーの状態を UNREACHABLE として表示します。	いいえ

18.3.2 replication_group_members テーブル

[performance_schema.replication_group_members](#) テーブルは、グループのメンバーである様々なサーバーインスタンスのステータスの監視に使用されます。新しいメンバーが結合されたときにグループの構成が動的に変更された場合など、ビューが変更されるたびにテーブルの情報が更新されます。その時点で、サーバーはメタデータの一部を交換して同期し、連携を続行します。この情報は、レプリケーショングループのメンバーであるすべてのサーバーインスタンス間で共有されるため、すべてのグループメンバーに関する情報を任意のメンバーからクエリーすることができます。このテーブルを使用して、レプリケーショングループの状態の高レベルビューを取得できます。たとえば、次のように発行します:

```
SELECT * FROM performance_schema.replication_group_members;
+-----+-----+-----+-----+-----+-----+-----+
| CHANNEL_NAME | MEMBER_ID | MEMBER_HOST | MEMBER_PORT | MEMBER_STATE | MEMBER_ROLE | MEMBER_VERSION |
+-----+-----+-----+-----+-----+-----+-----+
| group_replication_applier | 041f26d8-f3f3-11e8-adff-080027337932 | example1 | 3306 | ONLINE | SECONDARY | 8.0.13 |
| group_replication_applier | f60a3e10-f3f2-11e8-8258-080027337932 | example2 | 3306 | ONLINE | PRIMARY | 8.0.13 |
| group_replication_applier | fc890014-f3f2-11e8-a9fd-080027337932 | example3 | 3306 | ONLINE | SECONDARY | 8.0.13 |
+-----+-----+-----+-----+-----+-----+-----+
```

この結果に基づいて、グループが3つのメンバー、クライアントがメンバーへの接続に使用する各メンバーホストとポート番号、およびメンバーの [server_uuid](#) で構成されていることがわかります。 [MEMBER_STATE](#) カラムには [セクション18.3.1「グループレプリケーションサーバーの状態」](#) のいずれかが表示されます。この場合、このグループの3つのメンバーはすべて [ONLINE](#) であり、 [MEMBER_ROLE](#) カラムにはセカンダリが2つあり、プライマリが1つであることが示されています。したがって、このグループはシングルプライマリモードで実行されている必要があります。 [MEMBER_VERSION](#) カラムは、グループをアップグレードし、異なる MySQL バージョンを実行しているメンバーを結合する場合に役立ちます。詳しくは [セクション18.3.1「グループレプリケーションサーバーの状態」](#) をご覧ください。

[Member_host](#) の値と分散リカバリプロセスへの影響の詳細は、 [セクション18.2.1.3「分散リカバリのユーザー資格証明」](#) を参照してください。

18.3.3 replication_group_member_stats テーブル

レプリケーショングループ内の各メンバーは、グループが受信したトランザクションを証明して適用します。証明者および適用者プロセスに関する統計は、適用者キューの増加状況、検出された競合の数、チェックされたトランザクションの数、すべての場所でコミットされたトランザクションなどを理解するために役立ちます。

[performance_schema.replication_group_member_stats](#) テーブルには、証明プロセスに関連するグループレベルの情報と、レプリケーショングループの個々のメンバーが受信および発生したトランザクションの統計が表示されます。この情報は、レプリケーショングループのメンバーであるすべてのサーバーインスタンス間で共有されるため、すべて

のグループメンバーに関する情報を任意のメンバーからクエリーすることができます。リモートメンバーの統計のリフレッシュは、`group_replication_flow_control_period` オプションで指定されたメッセージ期間によって制御されるため、クエリーが行われたメンバーのローカルに収集された統計とは若干異なる場合があります。このテーブルを使用してグループレプリケーションメンバーを監視するには、次のコマンドを発行します：

```
mysql> SELECT * FROM performance_schema.replication_group_member_stats\G
```

これらのフィールドは、グループに接続されているメンバーのパフォーマンスを監視するために重要です。たとえば、グループのいずれかのメンバーが、他のメンバーと比較して常にキュー内の多数のトランザクションをレポートするとします。これは、メンバーが遅延し、グループの他のメンバーと最新の状態を維持できないことを意味します。この情報に基づいて、キューに入れられたトランザクションの数を減らすために、グループからメンバーを削除するか、グループの他のメンバーでトランザクションの処理を遅延するかを決定できます。この情報は、Group Replication プラグインのフロー制御の調整方法の決定にも役立ちます。[セクション18.6.2「フロー制御」](#)を参照してください。

18.4 グループレプリケーション操作

このセクションでは、グループを管理するための一般的な操作について説明します。

18.4.1 オンライングループの構成

グループレプリケーションの実行中に、グループアクションコーディネータに依存する UDF のセットを使用してオンライングループを構成できます。これらの UDF は、バージョン 8.0.13 以上の Group Replication プラグインによってインストールされます。このセクションでは、実行中のグループに対する変更方法と使用可能な UDF について説明します。

重要

コーディネータが実行中のグループに対してグループ全体のアクションを構成できるようにするには、すべてのメンバーが MySQL 8.0.13 以上を実行しており、UDF がインストールされている必要があります。

UDF を使用するには、実行中のグループのメンバーに接続し、`SELECT` ステートメントを使用して UDF を発行します。Group Replication プラグインはアクションとそのパラメータを処理し、UDF を発行したメンバーに表示されるすべてのメンバーにコーディネータが送信します。アクションが受け入れられると、すべてのメンバーがアクションを実行し、完了時に終了メッセージを送信します。すべてのメンバーがアクションを終了として宣言すると、呼出し側メンバーは結果をクライアントに返します。

グループ全体を構成する場合、操作の分散特性は、グループレプリケーションプラグインの多くのプロセスと相互作用するため、次の点に注意する必要があることを意味します：

あらゆる場所で構成操作を発行できます。メンバー A を新しいプライマリにする場合、メンバー A に対する操作を呼び出す必要はありません。すべての操作は、すべてのグループメンバーに対して調整された方法で送信および実行されます。また、この分散操作の実行には異なる影響があります：呼出し側メンバーが停止した場合、すでに実行中の構成プロセスは他のメンバーで引き続き実行されます。呼出し側メンバーが異常終了した場合でも、監視機能を使用して、他のメンバーが正常に操作を完了していることを確認できます。

すべてのメンバーがオンラインである必要があります。移行または選択プロセスを簡略化し、できるだけ高速であることを保証するために、グループには現在分散リカバリプロセスにあるメンバーを含めないでください。そうしないと、構成アクションはステートメントを発行したメンバーによって拒否されます。

構成の変更中にメンバーはグループに参加できません。調整された構成変更中にグループに参加しようとするメンバーは、グループを離れ、その結合プロセスを取り消します。

一度に 1 つの構成のみ。同時構成操作はメンバーの相違につながる可能性があるため、構成変更を実行しているグループは他のグループ構成変更を受け入れることができません。

すべてのメンバーで MySQL 8.0.13 以上が実行されている必要があります。構成アクションは分散されているため、実行するにはすべてのメンバーが認識する必要があります。したがって、MySQL Server バージョン 8.0.12 以下を実行しているサーバーがグループに存在する場合、操作は拒否されます。

18.4.1.1 グループプライマリメンバーの変更

このセクションでは、単一プライマリグループのどのメンバーがプライマリであるかを変更する方法について説明します。グループモードの変更に使用される関数は、任意のメンバーで実行できます。

プライマリのメンバーの変更

`group_replication_set_as_primary()` UDF を使用して、どのメンバーが単一プライマリグループのプライマリであるかを変更します。マルチプライマリグループのメンバーに対して発行された場合、この関数は無効です。プライマリメンバーのみがグループに書き込むことができるため、そのメンバーで非同期チャネルが実行されている場合、非同期チャネルが停止するまで切替えは許可されません。

8.0.17 から MySQL Server バージョンを実行しているメンバーで UDF を発行し、すべてのメンバーが MySQL Server バージョン 8.0.17 以上を実行している場合、パッチバージョンに基づいて、グループ内の最下位の MySQL Server バージョンを実行している新規プライマリメンバーのみを指定できます。この保護策は、グループが新しい機能との互換性を維持するために適用されます。いずれかのメンバーが MySQL 8.0.13 と MySQL 8.0.16 の間で MySQL Server バージョンを実行している場合、この保護策はグループに適用されず、新しいプライマリメンバーを指定できますが、グループ内で最も低い MySQL Server バージョンを実行しているプライマリを選択することをお勧めします。

次のコマンドを発行して、グループの新しいプライマリにするメンバーの `server_uuid` を渡します:

```
SELECT group_replication_set_as_primary(member_uuid);
```

アクションの実行中に、次のコマンドを発行して進行状況を確認できます:

```
SELECT event_name, work_completed, work_estimated FROM performance_schema.events_stages_current WHERE event_name LIKE "%stage/group_rpl%";
```

event_name	work_completed	work_estimated
stage/group_rpl/Primary Election: Waiting for members to turn on super_read_only	3	5

18.4.1.2 グループモードの変更

このセクションでは、グループが実行されているモード (単一プライマリまたはマルチプライマリ) を変更する方法について説明します。グループモードの変更に使用される関数は、任意のメンバーで実行できます。

シングルプライマリモードへの変更

`group_replication_switch_to_single_primary_mode()` UDF を使用して、マルチプライマリモードで実行されているグループをシングルプライマリモードに変更するには、次を発行します:

```
SELECT group_replication_switch_to_single_primary_mode();
```

シングルプライマリモードに変更すると、シングルプライマリモード (`group_replication_enforce_update_everywhere_checks=OFF`) での必要に応じて、すべてのグループメンバーで厳密な整合性チェックも無効になります。

文字列が渡されない場合、結果の単一プライマリグループでの新しいプライマリの選択は、[セクション18.1.3.1「シングルプライマリモード」](#)で説明されている選択ポリシーに従います。選択プロセスを上書きし、マルチプライマリグループの特定のメンバーをプロセスの新しいプライマリとして構成するには、メンバーの `server_uuid` を取得して `group_replication_switch_to_single_primary_mode()` に渡します。たとえば、次のコマンドを発行します:

```
SELECT group_replication_switch_to_single_primary_mode(member_uuid);
```

8.0.17 から MySQL Server バージョンを実行しているメンバーで UDF を発行し、すべてのメンバーが MySQL Server バージョン 8.0.17 以上を実行している場合、パッチバージョンに基づいて、グループ内の最下位の MySQL Server バージョンを実行している新規プライマリメンバーのみを指定できます。この保護策は、グループが新しい機能との互換性を維持するために適用されます。新しいプライマリメンバーを指定しない場合、選択プロセスではグループメンバーのパッチバージョンが考慮されます。

いずれかのメンバーが MySQL 8.0.13 と MySQL 8.0.16 の間で MySQL Server バージョンを実行している場合、この保護策はグループに適用されず、新しいプライマリメンバーを指定できませんが、グループ内で最も低い MySQL Server バージョンを実行しているプライマリを選択することをお勧めします。新しいプライマリメンバーを指定しない場合、選択プロセスではグループメンバーのメジャーバージョンのみが考慮されます。

アクションの実行中に、次のコマンドを発行して進行状況を確認できます：

```
SELECT event_name, work_completed, work_estimated FROM performance_schema.events_stages_current WHERE event_name LIKE "%stage/group_rpl%";
```

event_name	work_completed	work_estimated
stage/group_rpl/Primary Switch: waiting for pending transactions to finish	4	20

マルチプライマリモードへの変更

`group_replication_switch_to_multi_primary_mode()` UDF を使用して、単一プライマリモードで実行されているグループをマルチプライマリモードに変更するには、次のように発行します：

```
SELECT group_replication_switch_to_multi_primary_mode();
```

データの安全性と一貫性を確保するために調整されたグループ操作の後、グループに属するすべてのメンバーがプライマリになります。

シングルプライマリモードで実行されていたグループをマルチプライマリモードで実行するように変更すると、MySQL 8.0.17 以上を実行しているメンバーは、グループに存在する最低バージョンより上位の MySQL サーババージョンを実行している場合、自動的に読取り専用モードになります。MySQL 8.0.16 以下を実行しているメンバーはこのチェックを実行せず、常に読取り/書込みモードになります。

アクションの実行中に、次のコマンドを発行して進行状況を確認できます：

```
SELECT event_name, work_completed, work_estimated FROM performance_schema.events_stages_current WHERE event_name LIKE "%stage/group_rpl%";
```

event_name	work_completed	work_estimated
stage/group_rpl/Multi-primary Switch: applying buffered transactions	0	1

18.4.1.3 グループレプリケーショングループ書込みコンセンサスの使用

このセクションでは、グループのコンセンサスインスタンスの最大数をいつでも検査および構成する方法について説明します。この最大値はグループのイベント範囲と呼ばれ、グループがパラレルに実行できるコンセンサスインスタンスの最大数です。これにより、Group Replication デプロイメントのパフォーマンスを微調整できます。たとえば、LAN 上で実行されているグループにはデフォルト値の 10 が適していますが、WAN などの低速なネットワーク上で動作しているグループには、この数値を増やしてパフォーマンスを向上させます。

グループ書込み同時実行性の検査

`group_replication_get_write_concurrency()` UDF を使用して、実行時に次を発行してグループイベント範囲値を検査します：

```
SELECT group_replication_get_write_concurrency();
```

グループ書込み同時実行性の構成

`group_replication_set_write_concurrency()` UDF を使用して、次を発行することでシステムが並行して実行できるコンセンサスインスタンスの最大数を設定します：

```
SELECT group_replication_set_write_concurrency(instances);
```

ここで、`instances` はコンセンサスインスタンスの新しい最大数です。この UDF を使用するには、`GROUP_REPLICATION_ADMIN` 権限が必要です。

18.4.1.4 グループ通信プロトコルバージョンの設定

MySQL 8.0.16 から、グループレプリケーションにはグループの通信プロトコルの概念があります。Group Replication 通信プロトコルのバージョンは明示的に管理でき、グループでサポートする最も古い MySQL Server バージョンに対応するように設定できます。これにより、下位互換性を確保しながら、異なる MySQL Server バージョンのメンバーからグループを形成できます。MySQL 5.7.14 のバージョンではメッセージを圧縮でき、MySQL 8.0.16 のバージョンではメッセージを断片化することもできます。グループメンバーが異なる MySQL Server リリースを使用できるように、グループのすべてのメンバーは同じ通信プロトコルバージョンを使用する必要がありますが、すべてのグループメンバーが理解できるメッセージのみを送信します。

バージョン X の MySQL サーバーは、グループ通信プロトコルのバージョンが X 以下の場合にのみ、レプリケーショングループの **ONLINE** ステータスに参加して到達できます。新しいメンバーがレプリケーショングループに参加すると、グループの既存のメンバーによって通知される通信プロトコルバージョンがチェックされます。参加メンバーがそのバージョンをサポートしている場合、メンバーが追加の通信機能をサポートしていても、参加メンバーはグループに参加し、グループが発表した通信プロトコルを使用します。参加メンバーが通信プロトコルバージョンをサポートしていない場合は、グループから削除されます。

2 つのメンバーが同じメンバーシップ変更イベントに参加しようとする、両方のメンバーの通信プロトコルバージョンがすでにグループ通信プロトコルバージョンと互換性がある場合のみ参加できます。グループとは異なる通信プロトコルバージョンを持つメンバーは、分離して参加する必要があります。例:

- 一方の MySQL Server 8.0.16 インスタンスは、通信プロトコルバージョン 5.7.24 を使用するグループに正常に参加できます。
- 一方の MySQL Server 5.7.24 インスタンスは、通信プロトコルバージョン 8.0.16 を使用するグループに正常に参加できません。
- 2 つの MySQL Server 8.0.16 インスタンスは、通信プロトコルバージョン 5.7.24 を使用するグループに同時に参加することはできません。
- 2 つの MySQL Server 8.0.16 インスタンスは、通信プロトコルバージョン 8.0.16 を使用するグループに同時に参加できます。

グループがサポートしている最も古い MySQL Server バージョンを返す

`group_replication_get_communication_protocol()` UDF を使用して、グループが使用している通信プロトコルを検査できます。グループの既存のすべてのメンバーは、同じ通信プロトコルバージョンを返します。例:

```
SELECT group_replication_get_communication_protocol();
+-----+
| group_replication_get_communication_protocol() |
+-----+
| 8.0.16 |
+-----+
```

`group_replication_get_communication_protocol()` UDF は、グループがサポートする MySQL の最小バージョンを返します。これは、`group_replication_set_communication_protocol()` UDF に渡されたバージョン番号、および UDF を使用するメンバーにインストールされている MySQL Server バージョンとは異なる場合があります。

以前のリリースのメンバーが参加できるようにグループの通信プロトコルバージョンを変更する必要がある場合は、`group_replication_set_communication_protocol()` UDF を使用して、許可する最も古いメンバーの MySQL Server バージョンを指定します。これにより、可能であれば、グループは互換性のある通信プロトコルバージョンにフォールバックされます。この UDF を使用するには `GROUP_REPLICATION_ADMIN` 権限が必要です。また、ステートメントを発行するときは、大部分を失うことなく、既存のすべてのグループメンバーがオンラインである必要があります。例:

```
SELECT group_replication_set_communication_protocol("5.7.25");
```

レプリケーショングループのすべてのメンバーを新しい MySQL Server リリースにアップグレードしても、グループ通信プロトコルのバージョンは一致するように自動的にアップグレードされません。以前のリリースでメンバーをサポートする必要がなくなった場合は、`group_replication_set_communication_protocol()` UDF を使用して、通信プロトコルバージョンを、メンバーをアップグレードした新しい MySQL Server バージョンに設定できます。例:

```
SELECT group_replication_set_communication_protocol("8.0.16");
```

`group_replication_set_communication_protocol()` UDF はグループアクションとして実装されるため、グループのすべてのメンバーで同時に実行されます。グループアクションはメッセージのバッファリングを開始し、すでに進行中

の送信メッセージの配信が完了するまで待機してから、通信プロトコルのバージョンを変更し、バッファされたメッセージを送信します。通信プロトコルバージョンを変更した後、メンバーがいつでもグループに参加しようとすると、グループメンバーは新しいプロトコルバージョンを通知します。

MySQL InnoDB クラスタは、AdminAPI 操作を使用してクラスタポロジが変更されるたびに、そのメンバーの通信プロトコルバージョンを自動的かつ透過的に管理します。InnoDB クラスタでは、現在クラスタの一部であるか、クラスタに参加しているすべてのインスタンスでサポートされている最新の通信プロトコルバージョンが常に使用されます。詳細は、[InnoDB クラスタ およびグループのレプリケーションプロトコル](#)を参照してください。

18.4.2 トランザクション一貫性保証

グループレプリケーションなどの分散システムの主な影響の1つは、グループとして提供される一貫性保証です。つまり、グループのメンバーに分散されたトランザクションのグローバル同期の一貫性。このセクションでは、グループ内で発生するイベントに応じてグループレプリケーションが一貫性保証を処理する方法と、グループの一貫性保証を最適に構成する方法について説明します。

18.4.2.1 トランザクションの一貫性保証の理解

分散一貫性保証では、通常の修復操作または障害修復操作のいずれかにおいて、グループレプリケーションは常に最終的な一貫性システムでした。つまり、受信トラフィックの速度が低下または停止するとすぐに、すべてのグループメンバーのデータコンテンツが同じになります。システムの整合性に関連するイベントは、手動または障害によって自動的にトリガーされる制御操作と、データフロー操作に分割できます。

グループレプリケーションの場合、一貫性の観点から評価できる制御操作は次のとおりです：

- グループレプリケーション [セクション18.4.3「分散リカバリ」](#) および書き込み保護でカバーされるメンバーの参加または離脱。
- ネットワーク障害。フェンシングモードでカバーされます。
- 単一プライマリグループのプライマリフェイルオーバー (`group_replication_set_as_primary()`) によってトリガーされる操作の場合もあります。

一貫性保証とプライマリフェイルオーバー

単一プライマリグループでは、セカンダリがプライマリに昇格されたときにプライマリフェイルオーバーが発生した場合、レプリケーションバックログの大きさに関係なく、新しいプライマリをアプリケーショントラフィックですぐに使用可能にすることも、バックログが適用されるまでアクセスを制限することもできます。

最初のアプローチでは、新しいプライマリを選択し、古いプライマリから可能性のあるバックログをまだ適用している間にデータアクセスをすぐに許可することで、プライマリの障害後に安定したグループメンバーシップを保護できる最小時間がグループにかかります。書き込み一貫性は保証されますが、読取りでは、新しいプライマリがバックログを適用している間、失効したデータを一時的に取得できます。たとえば、クライアント C1 が障害の直前に古いプライマリに `A=2 WHERE A=1` を書き込んだ場合、クライアント C1 が新しいプライマリに再接続されると、新しいプライマリがバックログを適用してグループを離れる前の古いプライマリの状態でキャッチアップするまで、`A=1` を読み取る可能性があります。

2つ目の代替方法では、プライマリ障害後に安定したグループメンバーシップが保護され、最初の代替方法と同じ方法で新しいプライマリが選択されますが、この場合、グループは新しいプライマリがすべてのバックログを適用するまで待機してから、データアクセスを許可します。これにより、前述のような状況で、クライアント C1 が新しいプライマリに再接続されると、`A=2` が読み取られます。ただし、フェイルオーバーに必要な時間はバックログのサイズに比例するため、適切に構成されたグループでは小さいにする必要があります。

MySQL 8.0.14 より前は、フェイルオーバーポリシーを構成する方法はありませんでした。デフォルトでは、可用性は最初のアプローチで説明されているように最大化されていました。MySQL 8.0.14 以上を実行しているメンバーがいるグループでは、`group_replication_consistency` 変数を使用して、プライマリフェイルオーバー時にメンバーによって提供されるトランザクション一貫性保証のレベルを構成できます。[プライマリ選択に対する整合性の影響](#)を参照してください。

データフロー操作

データフローは、特にこれらの操作がすべてのメンバーに分散されている場合に、グループに対して実行される読取りおよび書き込みによるグループの一貫性保証に関連します。データフロー操作は、グループレプリケーションの両方のモードに適用されます。ただし、この説明を明確にするために、シングルプライマリモードとマルチプライマリモードに制限されています。単一プライマリグループメンバー間で受信読取りまたは書き込みトランザクションを分割する通常の方法は、書き込みをプライマリにルーティングし、読取りをセカンダリに均等に分散することです。グループは単一のエンティティとして動作する必要があるため、プライマリでの書き込みがセカンダリで即時に使用可能であることを期待することが妥当です。Group Replication は Paxos アルゴリズムを実装する Group Communication System (GCS) プロトコルを使用して記述されますが、Group Replication の一部は非同期であり、これはデータがセカンダリに非同期に適用されることを意味します。これは、クライアント C2 がプライマリに `B=2 WHERE B=1` を書き込んで、すぐにセカンダリに接続し、`B=1` を読み取ることができることを意味します。これは、セカンダリがまだバックログを適用しており、プライマリによって適用されたトランザクションを適用していないためです。

トランザクション同期ポイント

グループ全体でトランザクションを同期する時点に基づいて、グループの一貫性保証を構成します。このセクションでは、概念を理解しやすくするために、読取り操作時または書き込み操作時にグループ全体でトランザクションを同期するポイントを簡略化します。読取り時にデータが同期化された場合、現在のクライアントセッションは、特定の時点 (前のすべての更新トランザクションが適用された時点) まで待機してから、実行を開始します。このアプローチでは、このセッションのみが影響を受け、他のすべての同時データ操作は影響を受けません。

書き込み時にデータが同期された場合、書き込みセッションはすべてのセカンダリがデータを書き込むまで待機します。グループレプリケーションでは書き込みの合計順序が使用されるため、これは、セカンダリのキューにあるこの書き込みおよびそれより前のすべての書き込みが適用されるのを待機していることを意味します。したがって、この同期ポイントを使用する場合、書き込みセッションはすべてのセカンダリキューが適用されるまで待機します。

代替方法により、クライアントに説明されている状況では、すぐにセカンダリに接続されていても、C2 は常に `B=2` を読み取るようになります。それぞれの代替方法には、システムワークロードに直接関連する利点とデメリットがあります。次の例では、様々なタイプのワークロードについて説明し、適切な同期ポイントをアドバイスします。

次の状況を考えてみます:

- 失効したデータの読取りを回避するために、読取り元のサーバーに追加の制限をデプロイせずに読取りのロードバランシングを行う場合、グループ書き込みはグループ読取りよりもはるかに一般的ではありません。
- 主に読取り専用データを持つグループがある場合、コミットされたすべての場所に読取り/書き込みトランザクションを適用して、最新の書き込みを含む最新のデータに対して後続の読取りが実行されるようにします。これにより、すべての RO トランザクションに対して同期コストを支払うのではなく、RW トランザクションに対してのみ支払うようになります。

このような場合は、書き込み時に同期化することを選択する必要があります。

次の状況を考えてみます:

- 失効したデータの読取りを回避するために、読取り元のサーバーに追加の制限をデプロイせずに読取りのロードバランシングを行う場合、グループ書き込みはグループ読取りよりもはるかに一般的です。
- ワークロード内の特定のトランザクションで、機密データ (ファイルの資格証明や類似データなど) が更新され、読取りで最新の値が取得されるように強制する場合などに、常にグループから最新のデータを読み取る必要があります。

このような場合は、読取り時に同期化することを選択する必要があります。

18.4.2.2 トランザクション一貫性保証の構成

[トランザクション同期ポイント](#) のセクションでは概念的に説明していますが、選択できる同期ポイントは 2 つあります: 読取り時または書き込み時、これらの用語は簡略化されており、グループレプリケーションで使用される用語は次のとおりです: トランザクション実行の前後。このセクションで示すように、一貫性レベルは、グループによって処理される読取り専用 (RO) トランザクションと読取り/書き込み (RW) トランザクションに異なる影響を与える可能性があります。

- [整合性レベルの選択方法](#)

- 整合性レベルの影響
- プライマリ選択に対する整合性への影響

次のリストは、トランザクションの一貫性保証を高めるために、`group_replication_consistency` 変数を使用してグループレプリケーションで構成できる一貫性レベルを示しています:

- **EVENTUAL**

RO トランザクションと RW トランザクションはどちらも、実行前に先行するトランザクションが適用されるのを待機しません。これは、`group_replication_consistency` 変数が追加される前の Group Replication の動作でした。RW トランザクションは、他のメンバーがトランザクションを適用するのを待機しません。つまり、あるメンバーでトランザクションを外部化してから別のメンバーに外部化できます。つまり、プライマリフェイルオーバーが発生した場合、新しいプライマリは、前のプライマリトランザクションがすべて適用される前に新しい RO および RW トランザクションを受け入れることができます。RO トランザクションは古い値になる可能性があり、RW トランザクションは競合のためロールバックする可能性があります。

- **BEFORE_ON_PRIMARY_FAILOVER**

古いプライマリからバックログを適用している、新しく選択されたプライマリを持つ新しい RO または RW トランザクションは、バックログが適用されるまで保持されます (適用されません)。これにより、プライマリフェイルオーバーが意図的に発生したかどうかにかかわらず、クライアントには常にプライマリの最新の値が表示されます。これにより一貫性が保証されますが、バックログが適用されている場合、クライアントは遅延を処理する必要があります。通常、この遅延は最小限に抑える必要がありますが、バックログのサイズによって異なります。

- **BEFORE**

RW トランザクションは、先行するすべてのトランザクションが完了するまで待機してから適用されます。RO トランザクションは、先行するすべてのトランザクションが完了するまで待機してから実行されます。これにより、トランザクションのレイテンシにのみ影響を与えることで、このトランザクションが最新の値を読み取るようになります。これにより、RO トランザクションでのみ同期が使用されるようになるため、すべての RW トランザクションでの同期のオーバーヘッドが削減されます。この一貫性レベルには、`BEFORE_ON_PRIMARY_FAILOVER` によって提供される一貫性保証も含まれます。

- **AFTER**

RW トランザクションは、その変更が他のすべてのメンバーに適用されるまで待機します。この値は RO トランザクションには影響しません。このモードでは、トランザクションがローカルメンバーでコミットされたときに、後続のトランザクションが書き込み値またはグループメンバーのより新しい値を読み取ることが保証されます。このモードは、主に RO 操作に使用されるグループとともに使用して、適用された RW トランザクションがコミット後のすべての場所に確実に適用されるようにします。これは、後続の読取りで最新の書き込みを含む最新のデータがフェッチされるようにするために、アプリケーションで使用できます。これにより、RW トランザクションでのみ同期が使用されるようになるため、RO トランザクションごとの同期のオーバーヘッドが削減されます。この一貫性レベルには、`BEFORE_ON_PRIMARY_FAILOVER` によって提供される一貫性保証も含まれます。

- **BEFORE_AND_AFTER**

RW トランザクションは、1) 前のすべてのトランザクションが適用されるまで待機し、2) 変更が他のメンバーに適用されるまで待機します。RO トランザクションは、先行するすべてのトランザクションが完了するまで待機してから実行されます。この一貫性レベルには、`BEFORE_ON_PRIMARY_FAILOVER` によって提供される一貫性保証も含まれます。

RO および RW トランザクションでは、`BEFORE` と `BEFORE_AND_AFTER` の両方の一貫性レベルを使用できます。RO トランザクションは変更を生成しないため、`AFTER` の一貫性レベルは RO トランザクションに影響しません。

整合性レベルの選択方法

一貫性レベルが異なると、両方の DBA に柔軟性が提供されます。DBA は、DBA を使用してインフラストラクチャを設定できます。また、アプリケーション要件に最適な整合性レベルを使用できる開発者にも柔軟性があります。次のシナリオでは、グループの使用方法に基づいて一貫性保証レベルを選択する方法を示します:

- シナリオ 1では、失効した読取りを気にすることなく読取りのロードバランシングを行う場合、グループの書き込み操作はグループの読取り操作よりかなり少なくなります。この場合、`AFTER` を選択する必要があります。

- シナリオ 2には、大量の書き込みを適用するデータセットがあり、失効したデータの読取りを気にすることなく、場合によっては読取りを実行する必要があります。この場合、**BEFORE** を選択する必要があります。
- シナリオ 3では、ワークロード内の特定のトランザクションが常にグループから最新のデータを読み取るようにし、機密データ (ファイルの資格証明や類似データなど) が更新されるたびに常に最新の値を読み取るように強制します。この場合、**BEFORE** を選択する必要があります。
- シナリオ 4には、主に読取り専用 (RO) データを持つグループがあり、コミットされたすべての場所に読取り/書き込み (RW) トランザクションを適用して、後続の読取りが最新の書き込みを含む最新データに対して実行され、すべての RO トランザクションで同期を支払わず、RW トランザクションでのみ行われるようにします。この場合、**AFTER** を選択する必要があります。
- シナリオ 5には、主に読取り/書き込み (RW) トランザクションが含まれるグループがあり、読取り/書き込み (RW) トランザクションは常にグループから最新のデータを読み取り、コミットされるとすべての場所に適用されるため、後続の読取りは最新の書き込みを含む最新のデータに対して実行され、読取り専用 (RO) トランザクションごと同期化を支払わずに RW トランザクションに対してのみ行われます。この場合、**BEFORE_AND_AFTER** を選択する必要があります。

一貫性レベルが適用されるスコープを自由に選択できます。整合性レベルをグローバルスコープで設定すると、グループのパフォーマンスに悪影響を与える可能性があるため、これは重要です。したがって、異なるスコープで `group_replication_consistency` システム変数を使用して、グループの一貫性レベルを構成できます。

現在のセッションで整合性レベルを強制するには、セッションスコープを使用します:

```
> SET @@SESSION.group_replication_consistency= 'BEFORE';
```

すべてのセッションで整合性レベルを強制するには、グローバルスコープを使用します:

```
> SET @@GLOBAL.group_replication_consistency= 'BEFORE';
```

特定のセッションで一貫性レベルを設定できるため、次のようなシナリオを利用できます:

- シナリオ 6特定のシステムは、強力な整合性レベルを必要としない命令をいくつか処理しますが、一方の命令には強力な整合性が重要です: ドキュメントへのアクセス権の管理。このシナリオでは、システムによってアクセス権限が変更され、すべてのクライアントに正しい権限が表示されるようにする必要があります。これらの手順では `SET @@SESSION.group_replication_consistency= 'AFTER'` のみが必要で、他の手順はグローバルスコープで設定された `EVENTUAL` で実行するように残しておきます。
- シナリオ 7シナリオ 6 で説明したのと同じシステムでは、命令は毎日分析処理を実行する必要があります。常に最新のデータを読み取る必要があります。これを実現するには、その特定の手順でのみ `SET @@SESSION.group_replication_consistency= 'BEFORE'` を実行する必要があります。

要約すると、特定の一致レベルですべてのトランザクションを実行する必要はありません (特に、一部のトランザクションのみが実際に必要な場合)。

グループレプリケーションではすべての読取り/書き込みトランザクションが完全に順序付けされるため、現在のセッションの一貫性レベルを **AFTER** に設定した場合でも、このトランザクションはその変更がすべてのメンバーに適用されるまで待機します。つまり、セカンダリキューに存在する可能性のあるこのトランザクションおよびそれより前のすべてのトランザクションを待機します。実際には、一貫性レベルの **AFTER** は、このトランザクションまで、およびこのトランザクションを含むすべてを待機します。

整合性レベルの影響

一貫性レベルを分類する別の方法は、グループへの影響、つまり、一貫性レベルが他のメンバーに与える影響です。

トランザクションストリームでの順序付けとは別に、**BEFORE** の整合性レベルはローカルメンバーにのみ影響します。つまり、他のメンバーとの調整は不要で、トランザクションに再フォーカスはありません。つまり、**BEFORE** は、使用されているトランザクションにのみ影響します。

AFTER と **BEFORE_AND_AFTER** の一貫性レベルは、他のメンバーで実行される同時トランザクションに副作用します。これらの一貫性レベルでは、**AFTER** または **BEFORE_AND_AFTER** とのトランザクションの実行中に **EVENTUAL** 一貫性レベルのトランザクションが開始されると、他のメンバートランザクションが待機します。他のメンバーは、他のメンバートランザクションに **EVENTUAL** 一貫性レベルがある場合でも、そのメンバーで **AFTER** ト

ランザクションがコミットされるまで待機します。つまり、**AFTER** および **BEFORE_AND_AFTER** は all **ONLINE** グループメンバーに影響します。

これをさらに説明するために、3つのメンバー、M1、M2 および M3 を持つグループを想定します。メンバー M1 では、クライアントは次を発行します:

```
> SET @@SESSION.group_replication_consistency= AFTER;
> BEGIN;
> INSERT INTO t1 VALUES (1);
> COMMIT;
```

次に、前述のトランザクションが適用されている間に、メンバー M2 でクライアントが発行します:

```
> SET SESSION group_replication_consistency= EVENTUAL;
```

この状況では、2つ目のトランザクションの一貫性レベルは **EVENTUAL** ですが、最初のトランザクションが M2 ですでにコミットフェーズにある間に実行を開始するため、2つ目のトランザクションは最初のトランザクションがコミットを終了するまで待機する必要があり、その後実行できるだけです。

ONLINE メンバーで使用できるのは一貫性レベルの **BEFORE**、**AFTER** および **BEFORE_AND_AFTER** のみで、他の状態のメンバーで使用しようとするとセッションエラーが発生します。

一貫性レベルが **EVENTUAL** でないトランザクションは、`wait_timeout` 値で構成されたタイムアウトに達するまで実行を保持し、デフォルトは 8 時間です。タイムアウトに達すると、**ER_GR_HOLD_WAIT_TIMEOUT** エラーがスローされます。

プライマリ選択に対する整合性の影響

このセクションでは、グループの一貫性レベルが、新しいプライマリを選択した単一プライマリグループに与える影響について説明します。このようなグループは、障害を自動的に検出し、アクティブなメンバー (メンバーシップ構成) のビューを調整します。さらに、グループがシングルプライマリモードでデプロイされている場合、グループメンバーシップが変更されるたびに、グループにプライマリメンバーがまだ存在するかどうかを検出するチェックが実行されます。存在しない場合は、セカンダリメンバーのリストから新しいメンバーが選択されます。通常、これはセカンダリプロモーションと呼ばれます。

システムが障害を自動的に検出して再構成するという事実を考慮すると、ユーザーは昇格が行われると、新しいプライマリが古いプライマリのデータに関する正確な状態になることを期待する場合があります。つまり、レプリケートされたトランザクションの読み取りおよび書き込みが可能になると、そのトランザクションのバックログが新しいプライマリに適用されないことが予想される場合があります。実際には、アプリケーションが新しいプライマリにフェイルオーバーすると、一時的に古いデータを読み取ったり、古いデータレコードに書き込んだりする機会がなくなります。

フロー制御がアクティブ化され、グループで適切にチューニングされた場合、バックログが存在しないか、または小さい必要があるため、昇格の直後に新しく選択されたプライマリから失効したデータを一時的に読み取る機会のごくわずかです。さらに、昇格後にプライマリへのアプリケーションアクセスを制御し、そのレベルで一貫性基準を強制するプロキシレイヤーまたはミドルウェアレイヤーがある場合があります。グループメンバーが MySQL 8.0.14 以上を使用している場合は、`group_replication_consistency` 変数を使用して新しいプライマリの動作を指定できます。この変数は、新しく選択されたプライマリが、バックログが完全に適用されるまで、または MySQL 8.0.13 以前を実行しているメンバーの動作まで読み取りと書き込みの両方をブロックするかどうかを制御します。バックログが適用される新しく選択されたプライマリで `group_replication_consistency` オプションが **BEFORE_ON_PRIMARY_FAILOVER** に設定され、バックログの適用中に新しいプライマリに対してトランザクションが発行された場合、バックログが完全に適用されるまで受信トランザクションはブロックされます。したがって、次の異常は回避されます:

- 読み取り専用および読み取り/書き込みトランザクションに対して失効した読み取りはありません。これにより、失効した読み取りが新しいプライマリによってアプリケーションに外部化されるのを防ぎます。
- 適用を待機しているバックログ内のレプリケートされた読み取り/書き込みトランザクションとの書き込み競合のため、読み取り/書き込みトランザクションの疑似ロールバックはありません。
- 読み取り/書き込みトランザクションでは、次のような読み取りスキューはありません:

```
> BEGIN;
> SELECT x FROM t1; -- x=1 because x=2 is in the backlog;
> INSERT x INTO t2;
```

```
> COMMIT;
```

このクエリーによって競合は発生しませんが、古い値が書き込まれます。

要約すると、`group_replication_consistency` が `BEFORE_ON_PRIMARY_FAILOVER` に設定されている場合は、新しいプライマリが選択されるたびに読取りおよび書き込みが保持されるため、可用性よりも一貫性の優先順位を付けることを選択します。これは、グループを構成する際に考慮する必要があるトレードオフです。また、フロー制御が正常に機能している場合は、バックログを最小限に抑える必要があることにも注意してください。 `BEFORE`、`AFTER` および `BEFORE_AND_AFTER` の一貫性レベルが高いほど、`BEFORE_ON_PRIMARY_FAILOVER` によって提供される一貫性保証も含まれることに注意してください。

プライマリに昇格されるメンバーに関係なく、グループが同じ一貫性レベルを提供するようにするには、グループのすべてのメンバーの `BEFORE_ON_PRIMARY_FAILOVER` (またはそれ以上の整合性レベル) が構成に永続化されている必要があります。たとえば、メンバーごとに次のコマンドを発行します:

```
> SET PERSIST group_replication_consistency='BEFORE_ON_PRIMARY_FAILOVER';
```

これにより、すべてのメンバーが同じように動作し、メンバーの再起動後に構成が保持されます。

`BEFORE_ON_PRIMARY_FAILOVER` 整合性レベルを使用している場合、すべての書き込みは保持されますが、昇格の実行後にバックログを適用している間もサーバーを検査できるように、すべての読取りがブロックされるわけではありません。これは、デバッグ、監視、可観測性およびトラブルシューティングに役立ちます。データを変更しないクエリーには、次のようなものがあります:

- `SHOW` ステートメント
- `SET` ステートメント
- `DO` ステートメント
- `EMPTY` ステートメント
- `USE` ステートメント
- `performance_schema` および `sys` データベースに対する `SELECT` ステートメントの使用
- `infoschema` データベースの `PROCESSLIST` テーブルに対する `SELECT` ステートメントの使用
- テーブルまたはユーザー定義関数を使用しない `SELECT` ステートメント
- `STOP GROUP_REPLICATION` ステートメント
- `SHUTDOWN` ステートメント
- `RESET PERSIST` ステートメント

トランザクションを永久に保留にすることはできず、保留時間が `wait_timeout` を超えると、`ER_GR_HOLD_WAIT_TIMEOUT` エラーが返されます。

18.4.3 分散リカバリ

メンバーがレプリケーショングループに参加したり、レプリケーショングループに再度参加したりする場合は、参加前または退席中にグループメンバーによって適用されたトランザクションをキャッチアップする必要があります。このプロセスは分散リカバリと呼ばれます。

参加メンバーは、グループからすでに受信したがまだ適用されていないトランザクションについて、その `group_replication_applier` チャネルのリレーログを確認することから始まります。結合メンバーが以前にグループに属していた場合は、そのメンバーが残ってから未適用のトランザクションが見つかる可能性があります。その場合は、これらを最初のステップとして適用します。グループの新規メンバーには適用するものではありません。

その後、参加メンバーはオンラインの既存のメンバーに接続して状態の転送を実行します。参加メンバーは、既存のメンバー(ドナーと呼ばれる)によって提供されている、参加前または退席中にグループで発生したすべてのトランザクションを転送します。次に、参加メンバーは、この状態の転送の進行中にグループで行われたトランザクションを

適用します。このプロセスが完了すると、参加メンバーはグループ内の残りのサーバーで捕捉され、グループへの通常の参加を開始します。

グループレプリケーションでは、分散リカバリ中の状態転送に次の方法の組合せを使用します：

- ・ クローンプラグイン機能を使用したりリモートクローニング操作 (MySQL 8.0.17 から使用可能)。この状態転送方法を有効にするには、グループメンバーと参加メンバーにクローンプラグインをインストールする必要があります。Group Replication は、必要なクローンプラグイン設定を自動的に構成し、リモートクローニング操作を管理します。
- ・ ドナーバイナリログからレプリケートし、参加メンバーにトランザクションを適用します。この方法では、ドナーと参加メンバーの間に確立された `group_replication_recovery` という名前の標準非同期レプリケーションチャンネルを使用します。

グループレプリケーションでは、参加メンバーに対して `START GROUP_REPLICATION` を発行した後、これらの方法の最適な組合せが状態転送に自動的に選択されます。これを行うために、グループレプリケーションでは、ドナーとして適切な既存のメンバー、ドナーからの参加メンバーに必要なトランザクションの数、およびグループメンバーのバイナリログファイルに必要なトランザクションが存在しなくなったかどうかをチェックされます。参加メンバーと適切なドナーの間のトランザクションギャップが大きい場合、または一部の必要なトランザクションがドナーのバイナリログファイルにない場合、Group Replication はリモートクローニング操作で分散リカバリを開始します。大きなトランザクションギャップがない場合、またはクローンプラグインがインストールされていない場合、Group Replication はドナーのバイナリログからの状態転送に直接進みます。

- ・ リモートクローニング操作中に、参加メンバーの既存のデータが削除され、ドナーデータのコピーに置き換えられます。リモートクローニング操作が完了し、参加メンバーが再起動すると、ドナーのバイナリログから状態転送が実行され、リモートクローニング操作の進行中にグループが適用したトランザクションが取得されます。
- ・ ドナーのバイナリログからの状態転送中に、結合メンバーはドナーのバイナリログから必要なトランザクションを複製して適用し、受信されたトランザクションを、結合メンバーがグループに参加したことをバイナリログが記録するポイント (ビュー変更イベント) まで適用します。これが進行中の場合、参加メンバーはグループが適用する新しいトランザクションをバッファします。バイナリログからの状態転送が完了すると、結合メンバーはバッファードトランザクションを適用します。

参加メンバーは、すべてのグループトランザクションで最新の状態である場合、オンラインとして宣言され、通常のメンバーとしてグループに参加でき、分散リカバリが完了します。

ヒント

バイナリログからの状態転送は、分散回復のためのグループレプリケーションベースメカニズムであり、レプリケーショングループ内のドナーおよび参加メンバーがクローニングをサポートするように設定されていない場合は、これが唯一使用可能なオプションです。バイナリログからの状態転送はクラシック非同期レプリケーションに基づいているため、グループに参加しているサーバーにグループデータがまったくない場合、または非常に古いバックアップイメージから取得されたデータがある場合は、非常に長い時間がかかることがあります。したがって、この状況では、サーバーをグループに追加する前に、グループ内にすでに存在するサーバーのかなり最近のスナップショットを転送して、グループデータを使用して設定することをお勧めします。これにより、分散リカバリにかかる時間が最小限に抑えられ、保持および転送が必要なバイナリログファイルが少なくなるため、ドナーサーバーへの影響が軽減されます。

18.4.3.1 分散リカバリの接続

参加メンバーが分散リカバリ中に状態転送のためにオンラインの既存メンバーに接続すると、参加メンバーは接続のクライアントとして機能し、既存のメンバーはサーバーとして機能します。ドナーバイナリログからの状態転送がこの接続で (非同期レプリケーションチャンネル `group_replication_recovery` を使用して) 進行中の場合、参加メンバーはレプリカとして機能し、既存のメンバーはソースとして機能します。リモートクローニング操作がこの接続で進行中の場合、参加メンバーは受信者として機能し、既存のメンバーはドナーとして機能します。グループレプリケーションコンテキスト外のロールに適用される構成設定は、グループレプリケーション固有の構成設定または動作によってオーバーライドされないかぎり、グループレプリケーションにも適用できます。

既存のメンバーが分散リカバリのために参加メンバーに提供する接続は、グループのオンラインメンバー間の通信にグループレプリケーションで使用される接続と同じではありません。

- リモート XCom インスタンス間の TCP 通信のグループレプリケーション (XCom、Paxos バリエーション) のためにグループ通信エンジンによって使用される接続は、`group_replication_local_address` システム変数によって指定されます。この接続は、オンラインメンバー間の TCP/IP メッセージに使用されます。ローカルインスタンスとの通信は、共有メモリーを使用して入カチャネルを介して行われます。
- 分散リカバリの場合、MySQL 8.0.20 まで、グループメンバーは、MySQL Server `hostname` および `port` システム変数で指定された、結合メンバーへの標準 SQL クライアント接続を提供します。`report_port` システム変数で代替ポート番号が指定されている場合は、かわりにそのポート番号が使用されます。
- MySQL 8.0.21 から、グループメンバーは分散リカバリエンドポイントの代替リストをメンバー参加専用クライアント接続として通知できるため、メンバーの通常のクライアントユーザーによる接続とは別に分散リカバリトラフィックを制御できます。このリストは `group_replication_advertise_recovery_endpoints` システム変数を使用して指定し、メンバーはグループに参加するときに分散リカバリエンドポイントのリストをグループに送信します。デフォルトでは、メンバーは以前のリリースと同様に標準 SQL クライアント接続を引き続き提供します。

重要

結合メンバーが、MySQL Server `hostname` システム変数で定義されたホスト名を使用して他のメンバーを正しく識別できない場合、分散リカバリは失敗する可能性があります。MySQL を実行しているオペレーティングシステムでは、DNS またはローカル設定を使用して、一意のホスト名を適切に構成することをお勧めします。サーバーが SQL クライアント接続に使用しているホスト名は、「パフォーマンススキーマ」テーブル `replication_group_members` の `Member_host` カラムで確認できます。複数のグループメンバーがオペレーティングシステムによって設定されたデフォルトのホスト名を外部化する場合、参加メンバーが正しいメンバーアドレスに解決せず、分散リカバリのために接続できない可能性があります。この状況では、MySQL Server `report_host` システム変数を使用して、各サーバーによって外部化される一意のホスト名を構成できます。

結合メンバーが分散リカバリ用の接続を確立するステップは、次のとおりです：

1. メンバーがグループに参加すると、`group_replication_group_seeds` システム変数のリストに含まれているシードメンバーのいずれかに接続され、最初はそのリストで指定されている `group_replication_local_address` 接続が使用されます。シードメンバーは、グループのサブセットである場合があります。
2. この接続を介して、シードメンバーはグループレプリケーションメンバーシップサービスを使用して、グループ内のオンラインのすべてのメンバーのリストをビューの形式で結合メンバーに提供します。メンバーシップ情報には、分散リカバリのために各メンバーが提供する分散リカバリエンドポイントまたは標準 SQL クライアント接続の詳細が含まれます。
3. 参加メンバーは、[セクション18.4.3.4「分散リカバリのフォルトトレランス」](#) で説明されている動作に従って、分散リカバリのドナーとなる適切なグループメンバーをこのリストから選択します。
4. その後、参加メンバーは、ドナーによって通知された分散リカバリエンドポイントを使用してドナーへの接続を試行し、それぞれをリストに指定された順序で試行します。ドナーがエンドポイントを提供しない場合、参加メンバーはドナー標準 SQL クライアント接続を使用して接続を試みます。接続の SSL 要件は、[分散リカバリの SSL および認証](#) で説明されている `group_replication_recovery_ssl_*` オプションで指定されているとおりです。
5. 参加メンバーが選択したドナーに接続できない場合、[セクション18.4.3.4「分散リカバリのフォルトトレランス」](#) で説明されている動作に従って、他の適切なドナーで再試行します。参加メンバーが接続を確立せずに通知されたエンドポイントのリストを使い果たした場合、ドナー標準 SQL クライアント接続にフォールバックするのではなく、別のドナーに切り替えることに注意してください。
6. 参加メンバーがドナーとの分散リカバリ接続を確立すると、[セクション18.4.3「分散リカバリ」](#) で説明されているように、その接続が状態転送に使用されます。使用される接続のホストとポートは、参加メンバーログに表示されます。リモートクローニング操作を使用する場合、参加メンバーが操作の最後に再起動すると、バイナリログからの状態転送用に新しいドナーとの接続が確立されます。これは、リモートクローニング操作に使用される元のドナーとは異なるメンバーへの接続であるか、元のドナーへの別の接続である可能性があります。いずれの場合も、分散リカバリプロセスは元のドナーと同じ方法で続行されます。

分散リカバリエンドポイントのアドレスの選択

分散リカバリエンドポイントとして `group_replication_advertise_recovery_endpoints` システム変数によって提供される IP アドレスは、MySQL Server 用に構成する必要はありません (つまり、`admin_address` システム変数または

`bind_address` システム変数のリストで指定する必要はありません)。これらはサーバーに割り当てる必要があります。使用するホスト名は、ローカル IP アドレスに解決される必要があります。IPv4 および IPv6 アドレスを使用できます。

分散リカバリエンドポイントに指定するポートは、MySQL Server 用に構成する必要があるため、`port`、`report_port` または `admin_port` システム変数で指定する必要があります。サーバーは、これらのポートで TCP/IP 接続をリスニングする必要があります。`admin_port` を指定する場合、分散リカバリのレプリケーションユーザーが接続するには `SERVICE_CONNECTION_ADMIN` 権限が必要です。`admin_port` を選択すると、分散リカバリ接続が通常の MySQL クライアント接続とは別に維持されます。

メンバーの結合では、リストで指定された順序で各エンドポイントが順に試行されます。`group_replication_advertise_recovery_endpoints` がエンドポイントのリストではなく `DEFAULT` に設定されている場合、標準 SQL クライアント接続が提供されます。標準 SQL クライアント接続は分散リカバリエンドポイントのリストに自動的に含まれず、エンドポイントのドナーリストが接続なしで使い果たされた場合、フォールバックとして提供されないことに注意してください。標準 SQL クライアント接続を多数の分散リカバリエンドポイントのいずれかとして提供する場合は、`group_replication_advertise_recovery_endpoints` で指定されたリストに明示的に含める必要があります。接続の最後の手段として機能するように、最後の場所に配置できます。

グループメンバー分散リカバリエンドポイント (エンドポイントが指定されていない場合は標準 SQL クライアント接続) は、`group_replication_ip_allowlist` (MySQL 8.0.22) または `group_replication_ip_whitelist` システム変数で指定されたグループレプリケーション許可リストに追加する必要はありません。allowlist は、メンバーごとに `group_replication_local_address` によって指定されたアドレスに対してのみ使用されます。分散リカバリのためにアドレスを取得するには、参加メンバーは allowlist によって許可されたグループへの初期接続を持っている必要があります。

リストした分散リカバリエンドポイントは、システム変数が設定されたとき、および `START GROUP_REPLICATION` ステートメントが発行されたときに検証されます。リストを正しく解析できない場合、またはサーバーがリスニングしていないためにホスト上でいずれかのエンドポイントにアクセスできない場合、Group Replication はエラーをログに記録し、起動しません。

分散リカバリの圧縮

MySQL 8.0.18 から、ドナーバイナリログからの状態転送によって、分散リカバリの圧縮をオプションで構成できます。圧縮は、ネットワーク帯域幅が制限され、ドナーが多数のトランザクションを参加メンバーに転送する必要がある分散リカバリに役立ちます。`group_replication_recovery_compression_algorithm` および `group_replication_recovery_zstd_compression_level` システム変数は、ドナーのバイナリログから状態転送を実行するときに使用される、許可される圧縮アルゴリズムと zstd 圧縮レベルを構成します。詳細は、[セクション4.2.8「接続圧縮制御」](#)を参照してください。

これらの圧縮設定は、リモートクローニング操作には適用されないことに注意してください。リモートクローニング操作を分散リカバリに使用する場合、クローンプラグインの `clone_enable_compression` 設定が適用されます。

分散リカバリのレプリケーションユーザー

分散リカバリでは、グループレプリケーションがメンバー間の直接レプリケーションチャネルを確立できるように、適切な権限を持つレプリケーションユーザーが必要です。レプリケーションユーザーには、リモートクローニング操作のためにドナーでクローンユーザーとして機能する適切な権限も必要です。すべてのグループメンバーで分散リカバリに同じレプリケーションユーザーを使用する必要があります。このレプリケーションユーザーを設定する手順は、[セクション18.2.1.3「分散リカバリのユーザー資格証明」](#)を参照してください。レプリケーションユーザー資格証明を保護する手順は、[セクション18.5.3.1「分散リカバリのためのセキュアなユーザー資格証明」](#)を参照してください。

分散リカバリの SSL および認証

分散リカバリ用の SSL は、サーバーの SSL 設定および `group_replication_ssl_mode` システム変数によって決定される通常のグループ通信用の SSL とは別に構成されます。分散リカバリ接続の場合、専用のグループレプリケーション分散リカバリ SSL システム変数を使用して、分散リカバリ専用の証明書および暗号の使用を構成できます。

デフォルトでは、分散リカバリ接続に SSL は使用されません。これをアクティブ化するには、[セクション18.5.3「分散リカバリ接続の保護」](#)の説明に従って、`group_replication_recovery_use_ssl=ON` を設定し、Group Replication 分

分散リカバリ SSL システム変数を構成します。SSL を使用するように設定されたレプリケーションユーザーが必要です。

分散リカバリが SSL を使用するように構成されている場合、グループレプリケーションは、リモートクローニング操作およびドナーのバイナリログからの状態転送にこの設定を適用します。Group Replication は、クローン SSL オプション (`clone_ssl_ca`、`clone_ssl_cert` および `clone_ssl_key`) の設定を、対応する Group Replication 分散リカバリオプション (`group_replication_recovery_ssl_ca`、`group_replication_recovery_ssl_cert` および `group_replication_recovery_ssl_key`) の設定と一致するように自動的に構成します。

分散リカバリに SSL を使用せず (`group_replication_recovery_use_ssl` が OFF に設定されている)、Group Replication のレプリケーションユーザーアカウントが `caching_sha2_password` プラグイン (MySQL 8.0 のデフォルト) または `sha256_password` プラグインで認証される場合、RSA キーペアがパスワード交換に使用されます。この場合、**キャッシュ SHA-2 認証プラグインを使用するレプリケーションユーザー** で説明されているように、`group_replication_recovery_public_key_path` システム変数を使用して RSA 公開キーファイルを指定するか、`group_replication_recovery_get_public_key` システム変数を使用してソースから公開キーをリクエストします。

18.4.3.2 分散リカバリのためのクローニング

MySQL Server クローンプラグインは、MySQL 8.0.17 から入手できます。グループ内の分散リカバリにリモートクローニング操作を使用する場合は、この機能をサポートするために、事前に既存のメンバーを設定してメンバーを結合する必要があります。この関数をグループで使用しない場合は、設定しないでください。この場合、Group Replication はバイナリログからの状態転送のみを使用します。

クローニングを使用するには、リモートクローニング操作をサポートするために、少なくとも 1 つの既存のグループメンバーと参加メンバーを事前に設定する必要があります。少なくとも、クローンプラグインをドナーにインストールしてメンバーに参加し、分散リカバリのためにレプリケーションユーザーに `BACKUP_ADMIN` 権限を付与し、`group_replication_clone_threshold` システム変数を適切なレベルに設定する必要があります。ドナーの最大可用性を確保するには、リモートクローニング操作をサポートするように、現在および将来のすべてのグループメンバーを設定することをお勧めします。

リモートクローニング操作では、ドナーからデータを転送する前に、結合メンバーからユーザー作成のテーブルスペースおよびデータが削除されることに注意してください。進行中に操作が停止した場合、結合メンバーに部分データが残されるか、データが残されない可能性があります。これを修復するには、Group Replication が自動的に行うリモートクローニング操作を再試行します。

クローニングの前提条件

クローンプラグインを設定および構成する完全な手順については、[セクション 5.6.7 「クローンプラグイン」](#) を参照してください。リモートクローニング操作の詳細な前提条件については、[セクション 5.6.7.3 「リモートデータのクローニング」](#) を参照してください。Group Replication の場合、次の重要なポイントと相違点に注意してください:

- ドナー (既存のグループメンバー) および受信者 (参加メンバー) にクローンプラグインがインストールされ、アクティブである必要があります。これを行う手順は、[セクション 5.6.7.1 「クローンプラグインのインストール」](#) を参照してください。
- ドナーと受信者は、同じオペレーティングシステム上で実行され、同じ MySQL Server バージョン (MySQL 8.0.17 以上でクローンプラグインをサポートする必要があります) である必要があります。したがって、クローニングは、メンバーが異なる MySQL Server バージョンを実行するグループには適していません。
- ドナーと受信者は Group Replication プラグインをインストールしてアクティブにする必要があります。ドナーでアクティブなその他のプラグイン (キーリングプラグインなど) も受信者でアクティブにする必要があります。
- 分散リカバリが SSL (`group_replication_recovery_use_ssl=ON`) を使用するように構成されている場合、グループレプリケーションはリモートクローニング操作にこの設定を適用します。Group Replication は、クローン SSL オプション (`clone_ssl_ca`、`clone_ssl_cert` および `clone_ssl_key`) の設定を、対応する Group Replication 分散リカバリオプション (`group_replication_recovery_ssl_ca`、`group_replication_recovery_ssl_cert` および `group_replication_recovery_ssl_key`) の設定と一致するように自動的に構成します。
- レプリケーショングループに参加するために、`clone_valid_donor_list` システム変数に有効なドナーのリストを設定する必要はありません。グループレプリケーションは、既存のグループメンバーからドナーを選択した後、この設定を自動的に構成します。リモートクローニング操作では、サーバー SQL プロトコルのホスト名とポートが使用されることに注意してください。

- クローンプラグインには、リモートクローニング操作のネットワーク負荷およびパフォーマンスへの影響を管理するための多数のシステム変数があります。グループレプリケーションではこれらの設定は構成されないため、必要に応じて設定したり、デフォルト設定を許可できます。リモートクローニング操作を分散リカバリに使用する場合は、Group Replication 圧縮設定ではなくクローンプラグインの `clone_enable_compression` 設定が操作に適用されることに注意してください。
- 受信者に対してリモートクローニング操作を起動するために、グループレプリケーションでは、`CLONE_ADMIN` 権限がすでにある内部 `mysql.session` ユーザーが使用されるため、これを設定する必要はありません。
- リモートクローニング操作のドナーのクローンユーザーとして、Group Replication では分散リカバリ用に設定したレプリケーションユーザー (セクション18.2.1.3「分散リカバリのユーザー資格証明」で説明) が使用されます。したがって、クローニングをサポートするすべてのグループメンバーに対して、このレプリケーションユーザーに `BACKUP_ADMIN` 権限を付与する必要があります。また、グループレプリケーション用にメンバーを構成する場合は、レプリケーションユーザーにメンバーへの参加権限を付与します。これは、メンバーがグループに参加した後、ドナーとして機能できるためです。すべてのグループメンバーで分散リカバリに同じレプリケーションユーザーが使用されます。既存のメンバーに対してこの権限をレプリケーションユーザーに付与するには、バイナリロギングを無効にして各グループメンバーに対して、またはバイナリロギングを有効にしているグループメンバーに対して、次のステートメントを発行します:

```
GRANT BACKUP_ADMIN ON *.* TO rpl_user@'%';
```

- `START GROUP_REPLICATION` を使用して、以前に `CHANGE REPLICATION SOURCE TO` | `CHANGE MASTER TO` を使用してユーザー資格証明を提供したサーバー上のレプリケーションユーザー資格証明を指定する場合は、リモートクローニング操作を実行する前に、必ずレプリケーションメタデータリポジトリからユーザー資格証明を削除してください。また、結合メンバーに `group_replication_start_on_boot=OFF` が設定されていることを確認します。その手順は、セクション18.5.3「分散リカバリ接続の保護」を参照してください。ユーザー資格証明の設定を解除しない場合、リモートクローニング操作中に参加メンバーに転送されます。その後、元のメンバーまたはそこからクローニングされたメンバーのいずれかで、`group_replication_recovery` チャネルが誤って格納された資格証明で起動される可能性があります。サーバー起動時のグループレプリケーションの自動開始 (リモートクローニング操作後を含む) では、格納されたユーザー資格証明が使用され、オペレータが `START GROUP_REPLICATION` コマンドで分散リカバリ資格証明を指定しなかった場合にも使用されます。

クローニングのしきい値

グループメンバーがクローニングをサポートするように設定されている場合、`group_replication_clone_threshold` システム変数は、分散リカバリでリモートクローニング操作を使用するためのしきい値をトランザクション数で指定します。ドナー上のトランザクションと参加メンバー上のトランザクションの間のギャップがこの数より大きい場合は、技術的に可能であれば、参加メンバーへの状態転送にリモートクローニング操作が使用されます。グループレプリケーションでは、既存のグループメンバーの `gtid_executed` セットに基づいて、しきい値を超えたかどうか計算されます。トランザクションのギャップが大きい場合にリモートクローニング操作を使用すると、事前にグループデータをサーバーに手動で転送せずに新しいメンバーをグループに追加できます。また、非常に古いメンバーをより効率的にキャッチアップできます。

`group_replication_clone_threshold` Group Replication システム変数のデフォルト設定は非常に高いため (GTID 内のトランザクションに許可されている最大シーケンス番号)、バイナリログからの状態転送が可能な場合は、クローニングが事実上非アクティブ化されます。グループレプリケーションで、より適切な状態転送のリモートクローニング操作を選択できるようにするには、システム変数を設定して、クローニングを実行するトランザクションのギャップとしてトランザクション数を指定します。

警告

アクティブグループの `group_replication_clone_threshold` には低い設定を使用しないでください。リモートクローニング操作の進行中にしきい値を超える数のトランザクションがグループで発生した場合、参加メンバーは再起動後にリモートクローニング操作を再度トリガーし、これを無期限に続行できます。この状況を回避するには、リモートクローニング操作にかかる時間内にグループ内で発生すると予想されるトランザクション数よりも大きい数をしきい値に設定してください。

ドナーバイナリログからの状態転送が不可能な場合、たとえば、参加メンバーに必要なトランザクションが既存のグループメンバーのバイナリログで使用できないため、グループレプリケーションはしきい値に関係なくリモートクローニング操作を実行しようとします。グループレプリケーションは、既存のグループメンバーの `gtid_purged` セットに基づいてこれを識別します。必要なトランザクションがどのメンバーバイナリログファイルでも使用できない場

合、`group_replication_clone_threshold` システム変数を使用してクローニングを非アクティブ化することはできません。この状況では、クローニングは、結合するメンバーに手動でデータを転送する唯一の代替手段であるためです。

クローニング操作

グループメンバーおよび参加メンバーがクローニング用に設定されている場合、グループレプリケーションによってリモートクローニング操作が管理されます。データのサイズによっては、リモートクローニング操作の完了に時間がかかる場合があります。プロセスの監視の詳細は、[セクション5.6.7.9「クローニング操作の監視」](#)を参照してください。

注記

状態の転送が完了すると、Group Replication は参加メンバーを再起動してプロセスを完了します。 `START GROUP_REPLICATION` ステートメントでレプリケーションユーザー資格証明を指定するなどの理由で、参加メンバーに `group_replication_start_on_boot=OFF` が設定されている場合は、この再起動後に `START GROUP_REPLICATION` を手動で再発行する必要があります。グループレプリケーションの開始に必要な `group_replication_start_on_boot=ON` およびその他の設定が構成ファイルで設定されている場合、または `SET PERSIST` ステートメントを使用して設定されている場合は、介入する必要はなく、参加メンバーを自動的にオンラインにするプロセスが続行されます。

リモートクローニング手順に時間がかかる場合、MySQL 8.0.22 より前のリリースでは、その期間中にグループに対して蓄積された一連の動作保証情報が大きすぎて、参加メンバーに転送できなくなる可能性があります。その場合、参加メンバーはエラーメッセージをログに記録し、グループには参加しません。MySQL 8.0.22 から、Group Replication は、このシナリオを回避するために、適用されたトランザクションのガベージコレクションプロセスを異なる方法で管理します。以前のリリースでは、このエラーが表示された場合、リモートクローニング操作の完了後、2分間待ってガベージコレクションが行われるようにし、グループ証明情報のサイズを小さくしてください。次に、結合メンバーに対して次のステートメントを発行して、前の一連の証明情報の適用を停止します：

```
RESET SLAVE FOR CHANNEL group_replication_recovery;  
Or from MySQL 8.0.22:  
RESET REPLICA FOR CHANNEL group_replication_recovery;
```

リモートクローニング操作では、ドナーから受信者へのテーブルに保持されている設定およびデータがクローニングされます。Group Replication は、Group Replication チャネルに特に関連する設定を管理します。グループレプリケーションのローカルアドレスなどの構成ファイルに保持されているグループレプリケーションメンバー設定はクローニングされず、参加メンバーでは変更されません。グループレプリケーションでは、SSL の使用に関連するチャネル設定も保持されるため、これらは個々のメンバーに固有です。

`group_replication_recovery` レプリケーションチャネルのドナーが使用するレプリケーションユーザー資格証明が `CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO` ステートメントを使用してレプリケーションメタデータリポジトリに格納されている場合、クローニング後にメンバーに転送されて使用され、そこで有効である必要があります。格納された資格証明を使用すると、リモートクローニング操作によって状態転送を受信したすべてのグループメンバーは、分散リカバリのレプリケーションユーザーおよびパスワードを自動的に受信します。 `START GROUP_REPLICATION` ステートメントでレプリケーションユーザーの資格証明を指定すると、リモートクローニング操作の開始に使用されますが、クローニング後に参加メンバーに転送されず、メンバーによって使用されることはありません。資格証明を新しいジョイナに転送せず、そこで記録する場合は、[セクション18.5.3「分散リカバリ接続の保護」](#)の説明に従ってリモートクローニング操作を実行する前に資格証明の設定を解除し、かわりに `START GROUP_REPLICATION` を使用してそれらを指定してください。

レプリケーションアプライアンスを保護するために `PRIVILEGE_CHECKS_USER` アカウントが使用されている場合 ([セクション17.3.3.2「グループレプリケーションチャネルの権限チェック」](#)を参照)、MySQL 8.0.19 から、ドナーの `PRIVILEGE_CHECKS_USER` アカウントおよび関連設定が結合メンバーにクローニングされます。参加メンバーがブート時にグループレプリケーションを開始するように設定されている場合、適切なレプリケーションチャネルの権限チェックにアカウントが自動的に使用されます。(MySQL 8.0.18 では、多くの制限により、グループレプリケーションチャネルで `PRIVILEGE_CHECKS_USER` アカウントを使用しないことをお勧めします。)

その他の目的でのクローニング

Group Replication は、分散リカバリのクローニング操作を開始および管理します。クローニングをサポートするよう設定されたグループメンバーは、ユーザーが手動で開始するクローニング操作にも参加できます。たとえば、グ

グループメンバーからドナーとしてクローニングして新しいサーバーインスタンスを作成するが、新しいサーバーインスタンスをすぐにグループに参加させない場合やそうでない場合があります。

クローニングをサポートするすべてのリリースで、グループレプリケーションを停止するグループメンバーを含むクローニング操作を手動で開始できます。クローニングではドナーと受信者のアクティブなプラグインが一致する必要があるため、そのサーバーインスタンスをグループに参加させない場合でも、グループレプリケーションプラグインを他のサーバーインスタンスにインストールしてアクティブにする必要があります。プラグインをインストールするには、次のステートメントを発行します:

```
INSTALL PLUGIN group_replication SONAME 'group_replication.so';
```

MySQL 8.0.20 より前のリリースでは、操作にグループレプリケーションが実行されているグループメンバーが含まれる場合、クローニング操作を手動で開始することはできません。クローニング操作で受信者のデータが削除および置換されない場合は、MySQL 8.0.20 からこれを実行できます。したがって、Group Replication が実行されている場合は、クローニング操作を開始するステートメントに `DATA DIRECTORY` 句を含める必要があります。

18.4.3.3 分散リカバリの構成

グループレプリケーション分散リカバリプロセスのいくつかの側面は、システムに合わせて構成できます。

接続試行回数

バイナリログからの状態転送の場合、Group Replication は、ドナーのプールからドナーに接続しようとしたときに参加メンバーが試行する回数を制限します。接続が成功せずに接続再試行の制限に達すると、分散リカバリプロセスはエラーで終了します。この制限は、参加メンバーがドナーへの接続を試行する合計回数を指定することに注意してください。たとえば、2 つのグループメンバーが適切なドナーで、接続再試行制限が 4 に設定されている場合、参加メンバーは制限に達する前に各ドナーへの接続を 2 回試行します。

デフォルトの接続再試行制限は 10 です。この設定は、`group_replication_recovery_retry_count` システム変数を使用して構成できます。次のコマンドは、ドナーへの接続の最大試行回数を 5 に設定します:

```
mysql> SET GLOBAL group_replication_recovery_retry_count= 5;
```

リモートクローニング操作の場合、この制限は適用されません。Group Replication は、バイナリログからの状態転送を開始する前に、クローニングに適した各ドナーへの接続を 1 つだけ試みます。

接続試行のスリープ間隔

バイナリログからの状態転送の場合、`group_replication_recovery_reconnect_interval` システム変数は、ドナーの接続試行の間に分散回復プロセスがスリープする時間を定義します。分散リカバリは、ドナー接続が試行されるたびにスリープしないことに注意してください。参加メンバーが同じサーバーに繰り返し接続するのではなく、異なるサーバーに接続しているため、サーバー A に影響する問題がサーバー B に影響しないと想定できます。したがって、分散リカバリは、可能性のあるすべてのドナーを通過した場合にのみ一時停止します。グループに参加しているサーバーがグループ内の適切な各ドナーに接続しようとする、分散リカバリプロセスは `group_replication_recovery_reconnect_interval` システム変数で構成された秒数スリープします。たとえば、2 つのグループメンバーが適切なドナーで、接続再試行制限が 4 に設定されている場合、参加メンバーは各ドナーへの接続を 1 回試行し、接続再試行間隔でスリープしてから、制限に達する前に各ドナーへの接続をさらに試行します。

デフォルトの接続再試行間隔は 60 秒で、この値は動的に変更できます。次のコマンドは、分散リカバリドナー接続再試行間隔を 120 秒に設定します:

```
mysql> SET GLOBAL group_replication_recovery_reconnect_interval= 120;
```

リモートクローニング操作の場合、この間隔は適用されません。Group Replication は、バイナリログからの状態転送を開始する前に、クローニングに適した各ドナーへの接続を 1 つだけ試みます。

参加メンバーをオンラインとしてマーク

分散リカバリがドナーから参加メンバーへの状態転送を正常に完了したら、参加メンバーをグループ内でオンラインとしてマークし、参加の準備ができます。デフォルトでは、これは参加メンバーが欠落していたすべてのトラ

ンザクションを受信して適用した後に実行されます。必要に応じて、参加メンバーが欠落していたすべてのトランザクションを受信して認証(つまり、競合検出が完了)した後、適用する前に、参加メンバーをオンラインとしてマークできます。これを行う場合は、`group_replication_recovery_complete_at` システム変数を使用して代替設定の `TRANSACTIONS_CERTIFIED` を指定します。

18.4.3.4 分散リカバリのフォルトトレランス

グループレプリケーション分散リカバリプロセスには、プロセス中に問題が発生した場合にフォルトトレランスを確保するためのいくつかの組込みメジャーがあります。

分散リカバリのドナーは、現在のビューの適切なオンライングループメンバーの既存のリストからランダムに選択されます。ランダムドナーを選択することは、複数のメンバーがグループに入るときに同じサーバーが複数回選択されない可能性があることを意味します。MySQL 8.0.17 からは、バイナリログからの状態転送の場合、ジョイナは、それ自体と比較して下位または同等のパッチバージョンの MySQL Server を実行しているドナーのみを選択します。以前のリリースでは、すべてのオンラインメンバーがドナーになることが許可されていました。リモートクローニング操作の場合、ジョイナは、それ自体と同じパッチバージョンを実行しているドナーのみを選択します。結合メンバーは、操作の最後に再起動すると、バイナリログからの状態転送用に新しいドナーとの接続を確認します。これは、リモートクローニング操作に使用される元のドナーとは異なるメンバーである可能性があります。

次の状況では、Group Replication は分散リカバリのエラーを検出し、新しいドナーに自動的にスイッチオーバーして、状態転送を再試行します:

- 接続エラー - 候補者ドナーへの接続に認証の問題があるか、別の問題があります。
- レプリケーションエラー - バイナリログからの状態転送に使用されているレプリケーションスレッド(受信側スレッドまたは適用側スレッド)のいずれかが失敗します。この状態転送方法では既存の MySQL レプリケーションフレームワークが使用されるため、一時的なエラーによってレシーバスレッドまたはアプライヤスレッドでエラーが発生する可能性があります。
- リモートクローニング操作エラー - リモートクローニング操作が失敗するか、完了する前に停止します。
- ドナーがグループから離れる - ドナーはグループを離れるか、状態転送の進行中にドナーで Group Replication が停止します。

「パフォーマンススキーマ」テーブル `replication_applier_status_by_worker` に、最後の再試行の原因となったエラーが表示されます。このような状況では、エラーに続く新しい接続が新しい候補者ドナーで試行されます。エラーが発生した場合に別のドナーを選択することは、新しい候補者ドナーに同じエラーがない可能性があることを意味します。クローンプラグインがインストールされている場合、Group Replication は、適切なオンラインクローンサポートドナーをそれぞれ最初に使用してリモートクローニング操作を試みます。これらの試行がすべて失敗した場合、グループレプリケーションはバイナリログからの状態転送を、可能であればすべての適切なドナーとともに順に試行します。

警告

リモートクローニング操作の場合、リモートクローニング操作でドナーからのデータの転送が開始される前に、受信者(参加メンバー)のユーザー作成のテーブルスペースおよびデータが削除されます。リモートクローニング操作が開始しても完了しない場合、参加メンバーは元のデータファイルの一部またはユーザーデータなしで残される可能性があります。データが完全にクローニングされる前にクローニング操作が停止した場合、ドナーによって転送されたデータは受信者から削除されます。この状況は、Group Replication が自動的に行うクローニング操作を再試行することで修復できます。

次の状況では、分散リカバリプロセスを完了できず、参加メンバーはグループを離れます:

- パージ済トランザクション - 参加メンバーに必要なトランザクションがオンライングループメンバーのバイナリログファイルに存在せず、リモートクローニング操作でデータを取得できません(クローンプラグインがインストールされていないか、クローニングが可能なすべてのドナーで試行されたが失敗したため)。したがって、参加メンバーはグループをキャッチアップできません。
- 競合するトランザクション - 結合メンバーには、グループに存在しないトランザクションがすでに含まれています。リモートクローニング操作が実行された場合、結合メンバーのデータディレクトリが消去されるため、これら

のトランザクションは削除されて失われます。ドナーバイナリログからの状態転送が実行された場合、これらのトランザクションはグループトランザクションと競合する可能性があります。

- 接続再試行制限に達しました - 参加メンバーは、接続再試行制限で許可されているすべての接続試行を行いました。これは、`group_replication_recovery_retry_count` システム変数を使用して構成できます (セクション 18.4.3.3 「分散リカバリの構成」 を参照)。
- ドナーはこれ以上いません - 参加メンバーは、オンラインクローンがサポートする各ドナーとのリモートクローニング操作を順に試行し (クローンプラグインがインストールされている場合)、可能であれば適切な各オンラインドナーを使用してバイナリログからの状態転送を正常に試行しませんでした。
- 参加メンバーはグループから退出 - 参加メンバーはグループを離れるか、状態転送の進行中に参加メンバーで Group Replication が停止します。

参加メンバーが意図せずグループを離れた場合、最後のメンバーを除く前述の状況では、`group_replication_exit_state_action` システム変数で指定されたアクションの実行に進みます。

18.4.3.5 分散リカバリの仕組み

Group Replication 分散回復プロセスがバイナリログから状態転送を実行しているときに、参加メンバーを特定の時点までドナーと同期するために、参加メンバーとドナーは GTID を利用します (セクション 17.1.3 「グローバルトランザクション識別子を使用したレプリケーション」 を参照)。ただし、GTID は、参加メンバーが欠落しているトランザクションを認識する手段のみを提供します。グループに参加するサーバーがキャッチアップする必要がある特定の時点マークしたり、証明書情報を伝達したりするのに役立ちません。これはバイナリログビューマーカーのジョブであり、バイナリログストリーム内のビューの変更をマークし、追加のメタデータ情報も含まれており、結合するメンバーに動作保証関連データが提供されます。

このトピックでは、ビュー変更の役割とビュー変更識別子、およびバイナリログから状態転送を実行する手順について説明します。

変更の表示および表示

view は、現在の構成にアクティブに参加しているメンバーのグループ (特定の時点) に対応します。グループ内で正しくオンラインで機能しています。

変更の表示は、メンバーの参加や離脱など、グループ構成の変更が発生したときに発生します。グループメンバーシップが変更されると、独立したビュー変更が同じ論理ポイントインタイムのすべてのメンバーに伝達されます。

ビュー識別子はビューを一意に識別します。ビューの変更が発生するたびに生成されます。

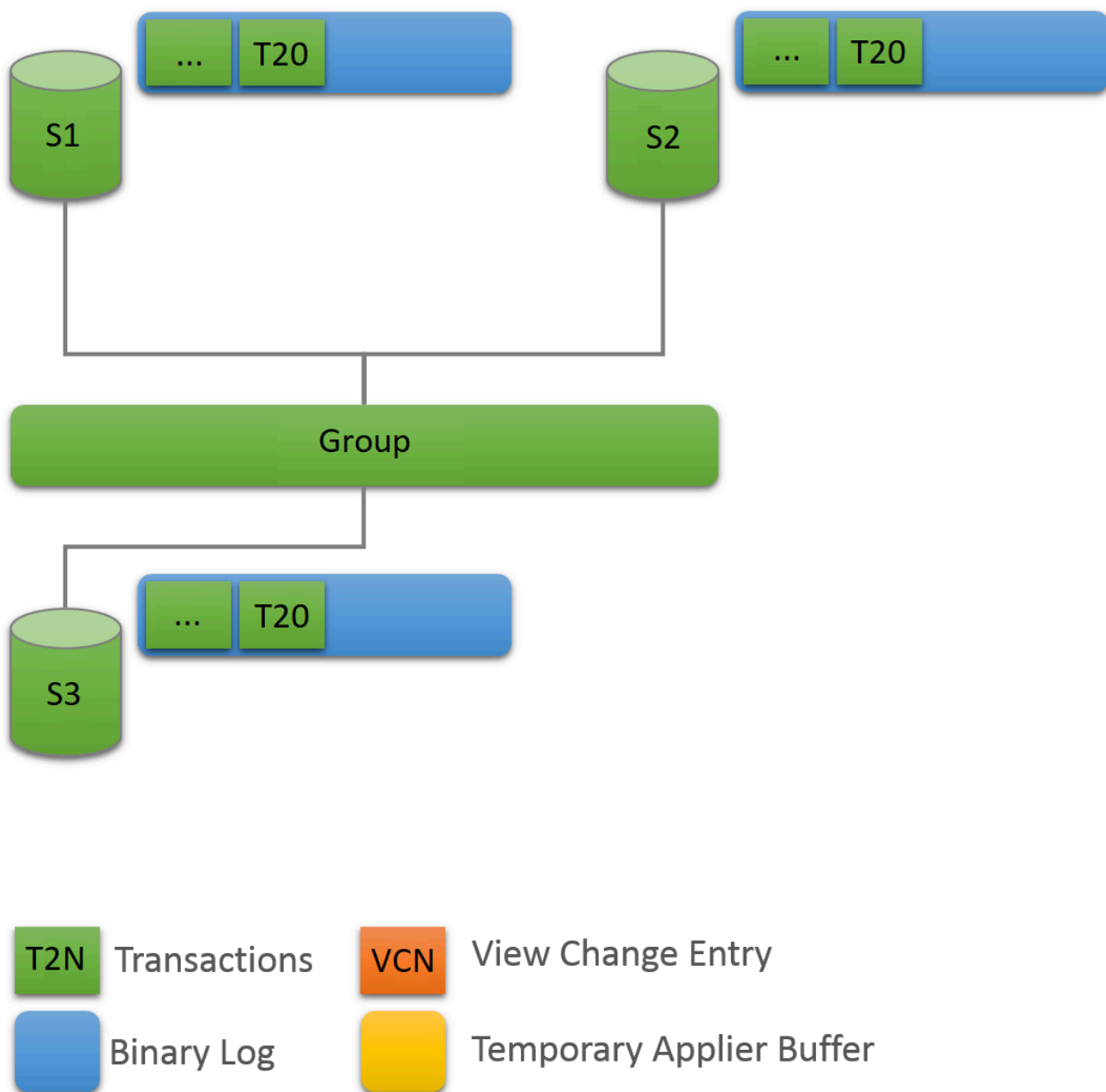
グループ通信レイヤーでは、ビューの変更とそれに関連付けられたビュー識別子により、メンバー結合の前後に交換されたデータ間の境界がマークされます。この概念はバイナリログイベントを介して実装されます: 「変更イベントの表示」。ビュー識別子は、グループメンバーシップでの変更の前後に送信されたトランザクションを境界設定するために記録されます。

ビュー識別子自体は 2 つの部分から作成されます: ランダムに生成された部分と単調に増加する整数。ランダムに生成された部分は、グループの作成時に生成され、グループに少なくとも 1 つのメンバーが存在する間は変更されません。整数は、ビューの変更が発生するたびに増分されます。これら 2 つの異なる部分を使用すると、ビュー識別子で、メンバーの参加または離脱によって発生した増分グループ変更を識別したり、すべてのメンバーがグループ全体の停止でグループを離れる状況を識別したりできます。したがって、グループがどのビューにあったかに関する情報は残りません。グループが最初から起動されたときに識別子の一部をランダムに生成することで、バイナリログ内のデータマーカーが一意のままになり、今後分散リカバリで問題が発生する可能性があるため、完全グループのシャットダウン後も同じ識別子は再利用されません。

Begin: 安定グループ

すべてのサーバーはオンラインであり、グループからの受信トランザクションを処理しています。一部のサーバーは、レプリケートされたトランザクションに関して少し遅れている可能性があります。最終的には収束します。グループは、分散データベースおよびレプリケートされたデータベースとして機能します。

図 18.8 安定グループ

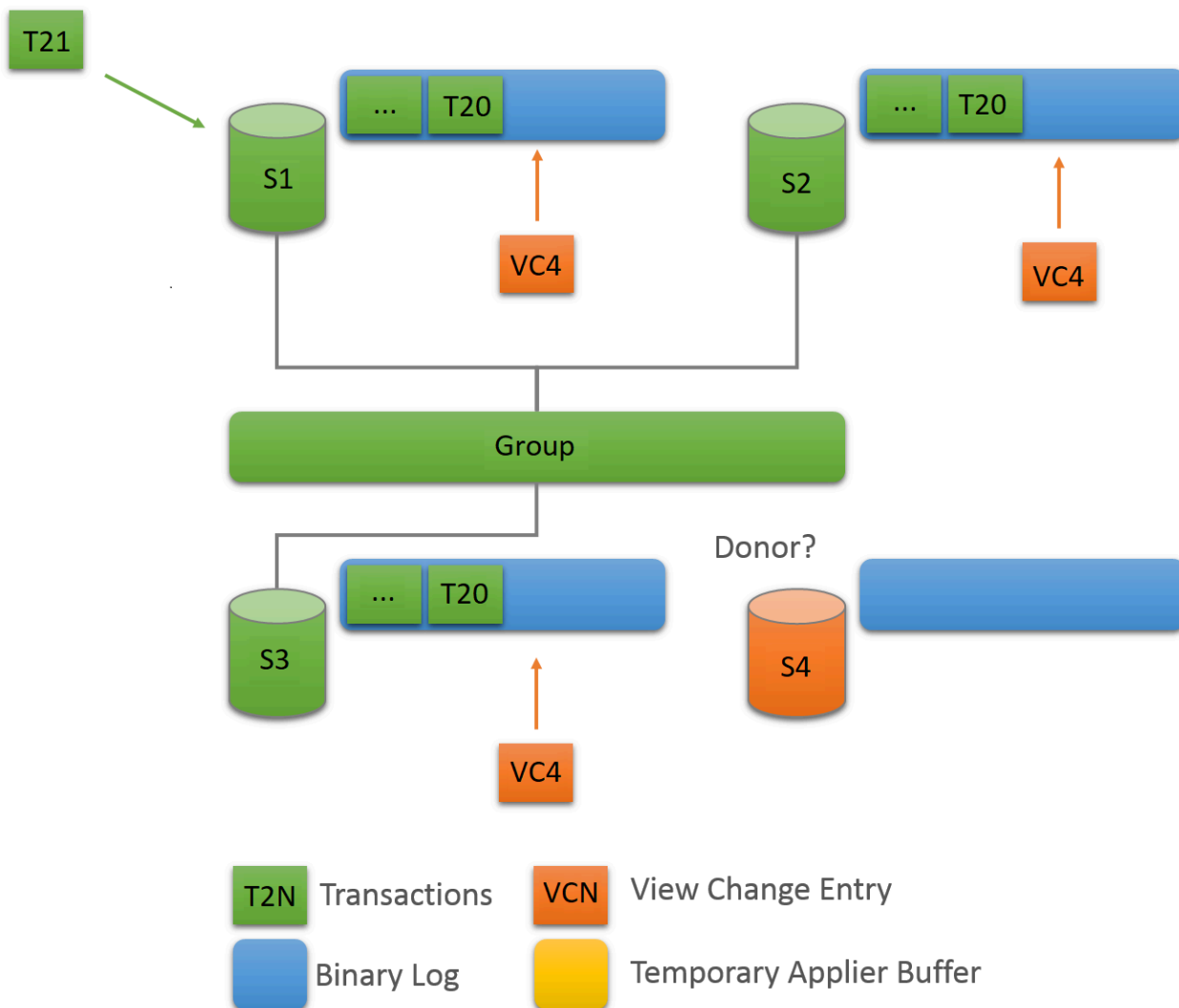


変更の表示: メンバー結合

新しいメンバーがグループに参加してビューの変更が実行されるたびに、すべてのオンラインサーバーは実行のためにビュー変更イベントをキューに入れます。これは、ビューが変更される前に、複数のトランザクションをサーバー上でキューに入れて適用できるため、キューに入れられます。そのため、これらのトランザクションは古いビューに属します。ビュー変更イベントをキューに入れると、その発生時の正しいマーキングが保証されます。

一方、参加メンバーは、ビュー抽象化を介してメンバーシップサービスによって記述されているように、オンラインサーバーのリストから適切なドナーを選択します。メンバーがビュー 4 に参加し、オンラインメンバーがビュー変更イベントをバイナリログに書き込みます。

図 18.9 A メンバー結合

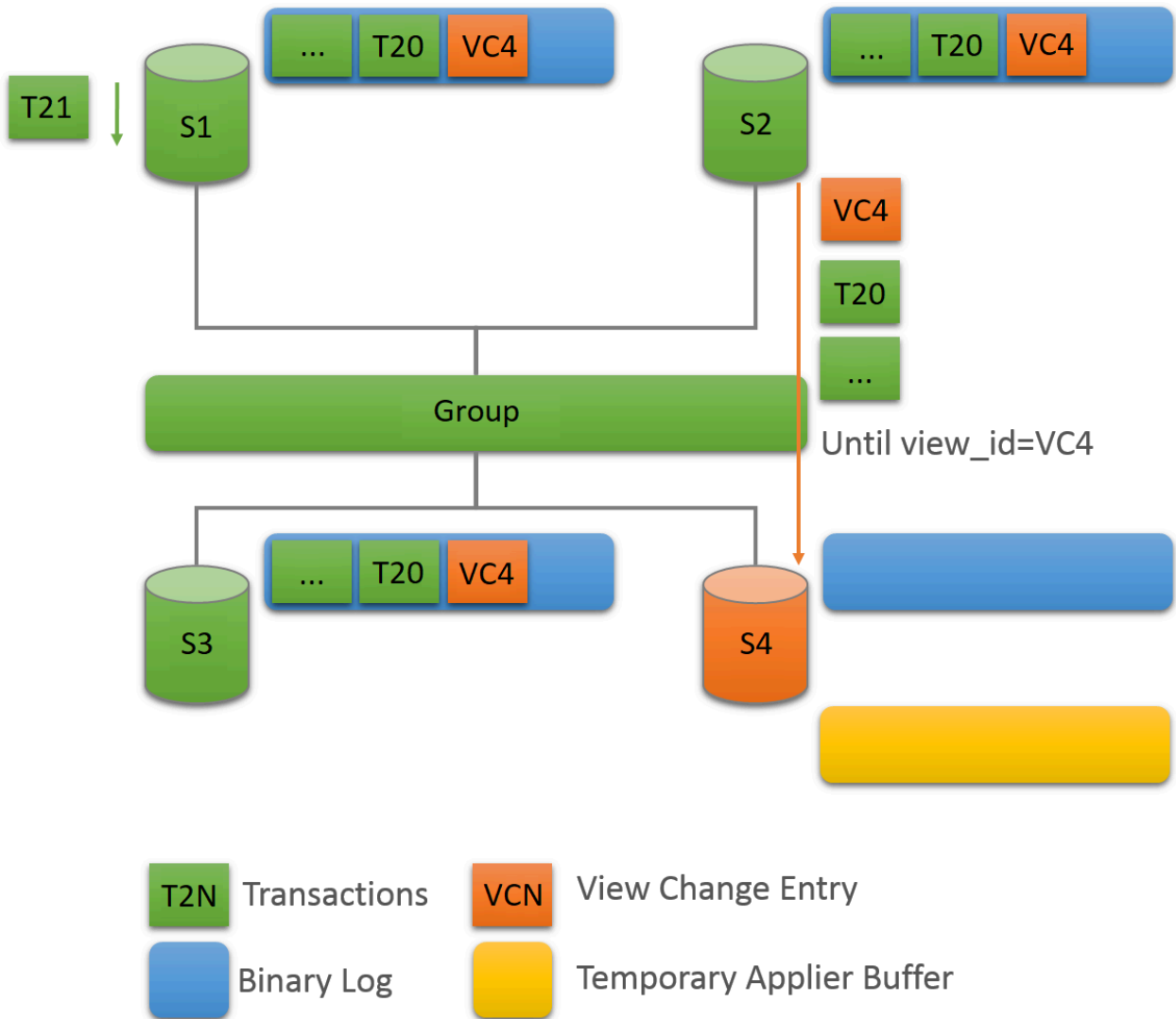


州移管: キャッチアップ

グループメンバーと結合メンバーがクローンプラグインを使用して設定されていて ([セクション18.4.3.2「分散リカバリのためのクローニング」](#) を参照)、結合メンバーとグループの間のトランザクションの差がリモートクローニング操作 (`group_replication_clone_threshold`) に設定されたしきい値を超えた場合、グループレプリケーションはリモートクローニング操作で分散回復を開始します。必要なトランザクションがどのグループメンバーのバイナリログファイルにも存在しない場合は、リモートクローニング操作も実行されます。リモートクローニング操作中に、参加メンバーの既存のデータが削除され、ドナーデータのコピーに置き換えられます。リモートクローニング操作が完了し、参加メンバーが再起動すると、ドナーのバイナリログから状態転送が実行され、リモートクローニング操作の進行中にグループが適用したトランザクションが取得されます。大きなトランザクションギャップがない場合、またはクローンプラグインがインストールされていない場合、Group Replication はドナーのバイナリログからの状態転送に直接進みます。

ドナーバイナリログからのステート転送の場合、参加メンバーとドナーとステート転送の間の接続が確立されます。このドナーとの相互作用は、グループアプライヤスレッドに参加しているサーバーが、グループに参加しているサーバーがグループに入ったときにトリガーされたビュー変更に対応するビュー変更ロギングイベントを処理するまで続きます。つまり、グループに参加するサーバーは、すでに存在するビューマーカーと一致するビュー識別子を持つマーカーに到達するまで、ドナーからレプリケートします。

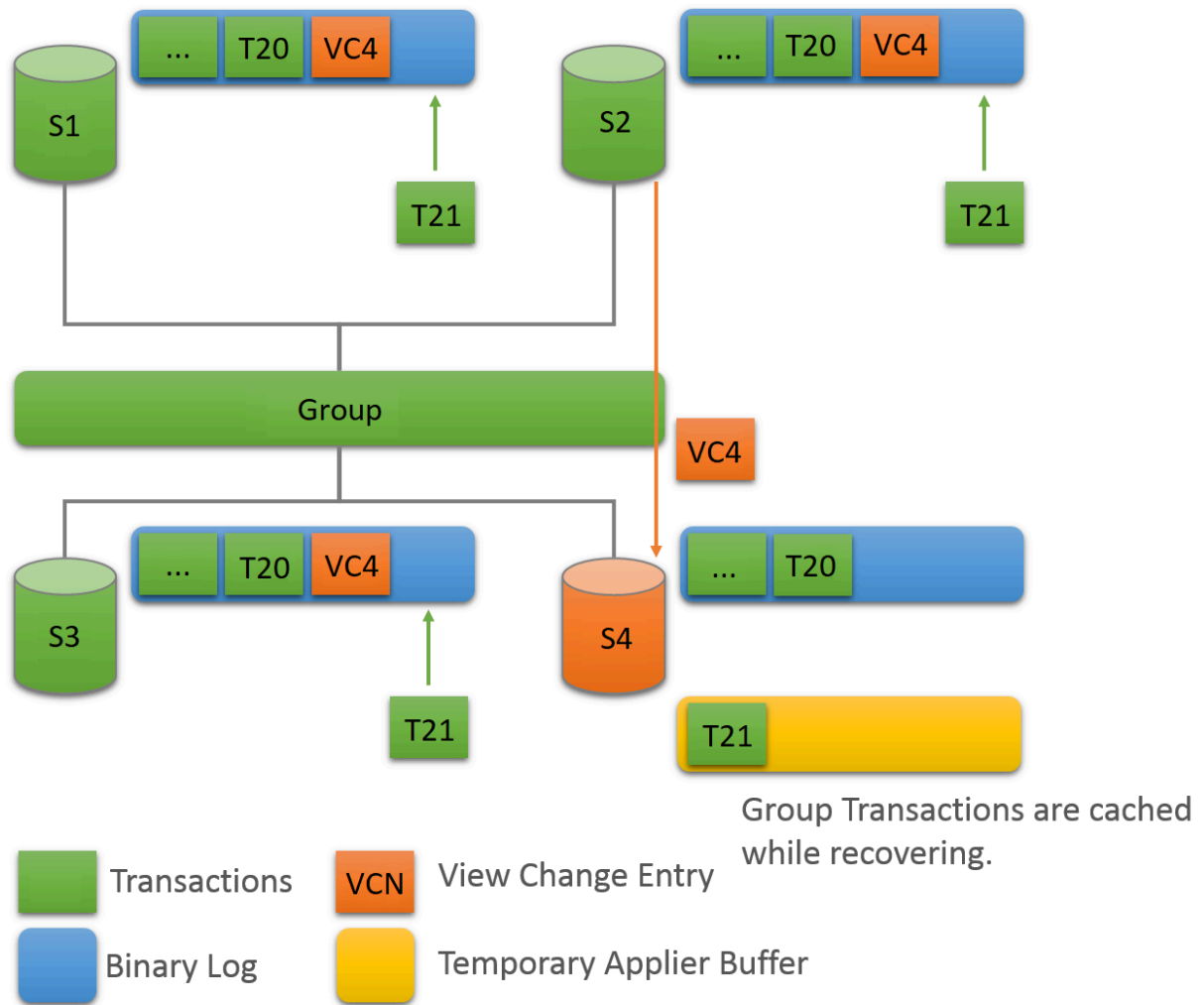
図 18.10 州移管: キャッチアップ



ビュー識別子はグループ内のすべてのメンバーに同時に送信されるため、グループに参加しているサーバーはレプリケートを停止するビュー識別子を認識します。これにより、ビュー識別子によって各グループビューに属するデータが明確にマークされるため、複雑な GTID セット計算が回避されます。

グループに参加するサーバーはドナーからレプリケートしていますが、グループからの受信トランザクションもキャッシュしています。最終的に、ドナーからのレプリケートを停止し、キャッシュされているものを適用するように切り替えます。

図 18.11 キュー済トランザクション



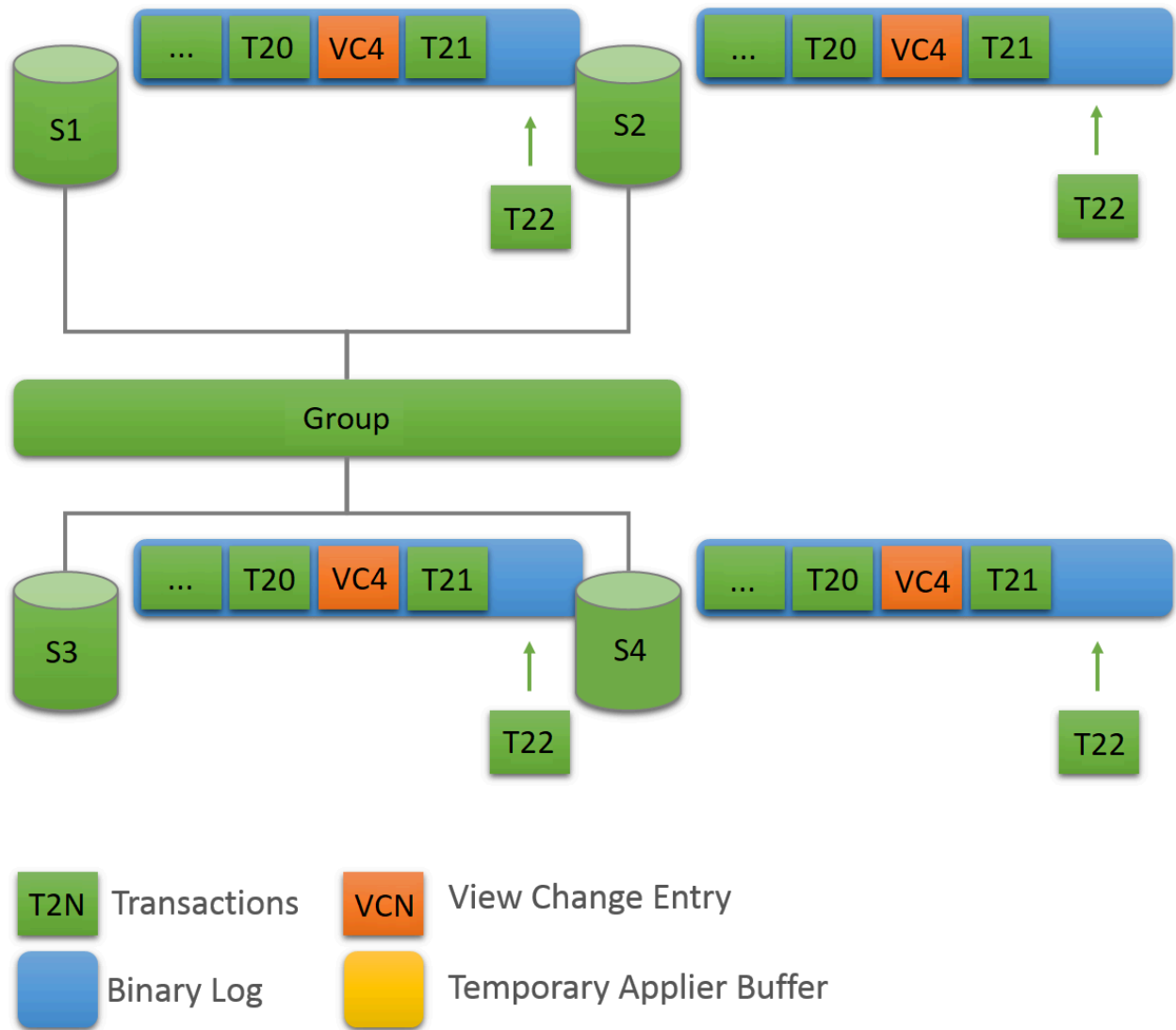
終了: キャッチアップ

グループに参加しているサーバーが予想されるビュー識別子を持つビュー変更ロギイベントを認識すると、ドナーへの接続が終了し、キャッシュされたトランザクションの適用が開始されます。これはバイナリログのマーカースとして機能し、ビューの変更を区切りますが、ビューの変更ロギイベントも別の役割を果たします。これは、グループに参加しているサーバーがグループに入ったときにすべてのサーバーによって認識される動作保証情報、つまり最後のビューの変更を伝達します。そうしないと、グループに参加しているサーバーには、後続のトランザクションを認証(競合の検出)するために必要な情報がありません。

キャッチアップの期間は、ワークロードおよびグループへの受信トランザクションの割合に依存するため、決定的ではありません。このプロセスは完全にオンラインであり、グループに参加しているサーバーはグループ内の他のサーバーをキャッチアップ中にブロックしません。したがって、このステージに移動すると、サーバーがグループに参加しているトランザクションの数が遅れるため、ワークロードに応じて増減できます。

グループに参加しているサーバーがキューに入れられていないトランザクションに到達し、格納されているデータが他のメンバーと等しい場合、パブリック状態はオンラインに変わります。

図 18.12 インスタンスオンライン



18.4.4 ネットワークパーティション化

レプリケートする必要がある変更が発生するたびに、グループはコンセンサスを達成する必要があります。これは通常のトランザクションの場合ですが、グループメンバーシップの変更およびグループの一貫性を維持する一部の内部メッセージングにも必要です。コンセンサスでは、グループメンバーの大部分が特定の決定に同意する必要があります。大部分のグループメンバーが失われると、大部分またはクォーラムを保護できないため、グループは進行できず、ブロックされます。

複数の標準的な障害が発生するとクォーラムが失われ、大部分のサーバーがグループから突然削除される可能性があります。たとえば、5つのサーバーのグループでは、それらの3つが一度にサイレントになった場合、大部分が危険にさらされるため、クォーラムを達成できません。実際、残りの2つは、他の3つのサーバーがクラッシュしたかどうか、またはネットワークパーティションがこれら2つのみを分離しているため、グループを自動的に再構成できません。

一方、サーバーが自発的にグループを終了した場合は、再構成する必要があることをグループに指示します。これは実際には、離れるサーバーが他のサーバーに離れることを通知することを意味します。つまり、他のメンバーはグループを適切に再構成でき、メンバーシップの一貫性が維持され、大部分が再計算されます。たとえば、3つのサーバーが一度に残る5つのサーバーの前述のシナリオでは、3つのサーバーが1つずつ離脱していることをグループに警告した場合、メンバーシップは5から2に調整でき、同時にクォーラムを保護できます。

注記

クォーラムの損失は、それ自体が不正な計画の副作用です。予想される障害の数 (連続しているか、一度にすべて発生しているか、散発的であるかに関係なく) のグループサイズを計画します。

次のセクションでは、グループ内のサーバーによって定足数が自動的に実現されないようにシステムがパーティション化された場合の対処方法について説明します。

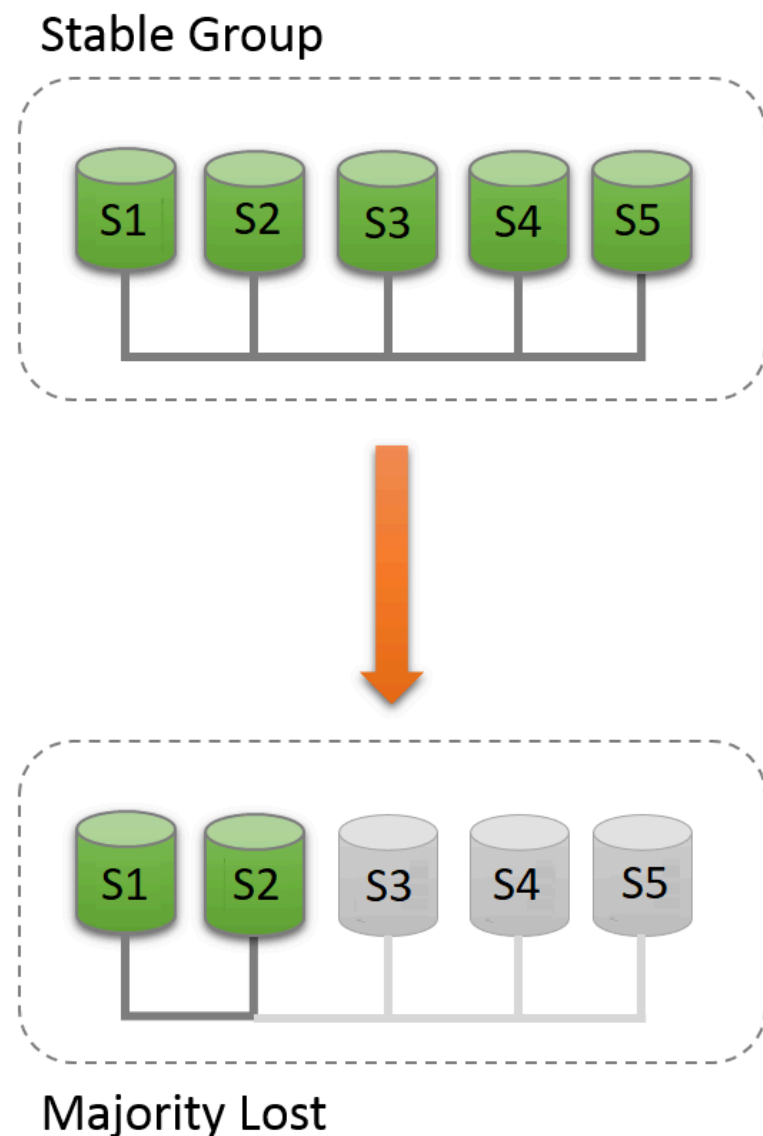
ヒント

過半数の損失後にグループから除外されたプライマリには、新しいグループに含まれていない追加のトランザクションが含まれる場合があります。これが発生した場合、グループから除外されたメンバーを追加しなおそうとすると、「このメンバーには、グループに存在するトランザクションよりも多くの実行済トランザクションがあります」というメッセージが表示されてエラーが発生します。

パーティションの検出

`replication_group_members` パフォーマンススキーマテーブルには、このサーバーの観点から見た現在のビューの各サーバーのステータスが表示されます。システムがパーティション化に実行されない時間の大部分であるため、テーブルにはグループ内のすべてのサーバー間で一貫性のある情報が表示されます。つまり、このテーブルの各サーバーのステータスは、現在のビューのすべてによって一致します。ただし、ネットワークのパーティション化があり、クォーラムが失われた場合は、接続できないサーバーのステータス `UNREACHABLE` がテーブルに表示されます。この情報は、Group Replication に組み込まれているローカル障害検出プログラムによってエクスポートされます。

図 18.13 損失クォーラム



このタイプのネットワークパーティションを理解するために、次のセクションでは、最初に5つのサーバーが正常に連携し、2つのサーバーのみがオンラインになった後にグループに対して行われる変更のシナリオについて説明します。シナリオをこの図に示します。

そのため、これらの5つのサーバーを含むグループが存在するとします:

- メンバー識別子 `199b2df7-4aaf-11e6-bb16-28b2bd168d07` を持つサーバー s1
- メンバー識別子 `199bb88e-4aaf-11e6-babe-28b2bd168d07` を持つサーバー s2
- メンバー識別子 `1999b9fb-4aaf-11e6-bb54-28b2bd168d07` を持つサーバー s3
- メンバー識別子 `19ab72fc-4aaf-11e6-bb51-28b2bd168d07` を持つサーバー s4
- メンバー識別子 `19b33846-4aaf-11e6-ba81-28b2bd168d07` を持つサーバー s5

最初はグループは正常に実行されており、サーバーはお互いに通信しています。これを確認するには、s1 にログインし、その `replication_group_members` パフォーマンススキーマテーブルを参照します。例:

```
mysql> SELECT MEMBER_ID, MEMBER_STATE, MEMBER_ROLE FROM performance_schema.replication_group_members;
+-----+-----+-----+
| MEMBER_ID          | MEMBER_STATE | MEMBER_ROLE |
+-----+-----+-----+
| 1999b9fb-4aaf-11e6-bb54-28b2bd168d07 | ONLINE      | SECONDARY   |
| 199b2df7-4aaf-11e6-bb16-28b2bd168d07 | ONLINE      | PRIMARY     |
| 199bb88e-4aaf-11e6-babe-28b2bd168d07 | ONLINE      | SECONDARY   |
| 19ab72fc-4aaf-11e6-bb51-28b2bd168d07 | ONLINE      | SECONDARY   |
| 19b33846-4aaf-11e6-ba81-28b2bd168d07 | ONLINE      | SECONDARY   |
+-----+-----+-----+
```

ただし、後で致命的な障害が発生し、サーバー s3、s4 および s5 が予期せず停止します。この数秒後、s1 の `replication_group_members` テーブルを再度参照すると、まだオンラインであることが示されますが、他のいくつかのメンバーはオンラインではありません。実際には、次に示すように、`UNREACHABLE` としてマークされます。さらに、大部分が失われたため、システム自体を再構成してメンバーシップを変更できませんでした。

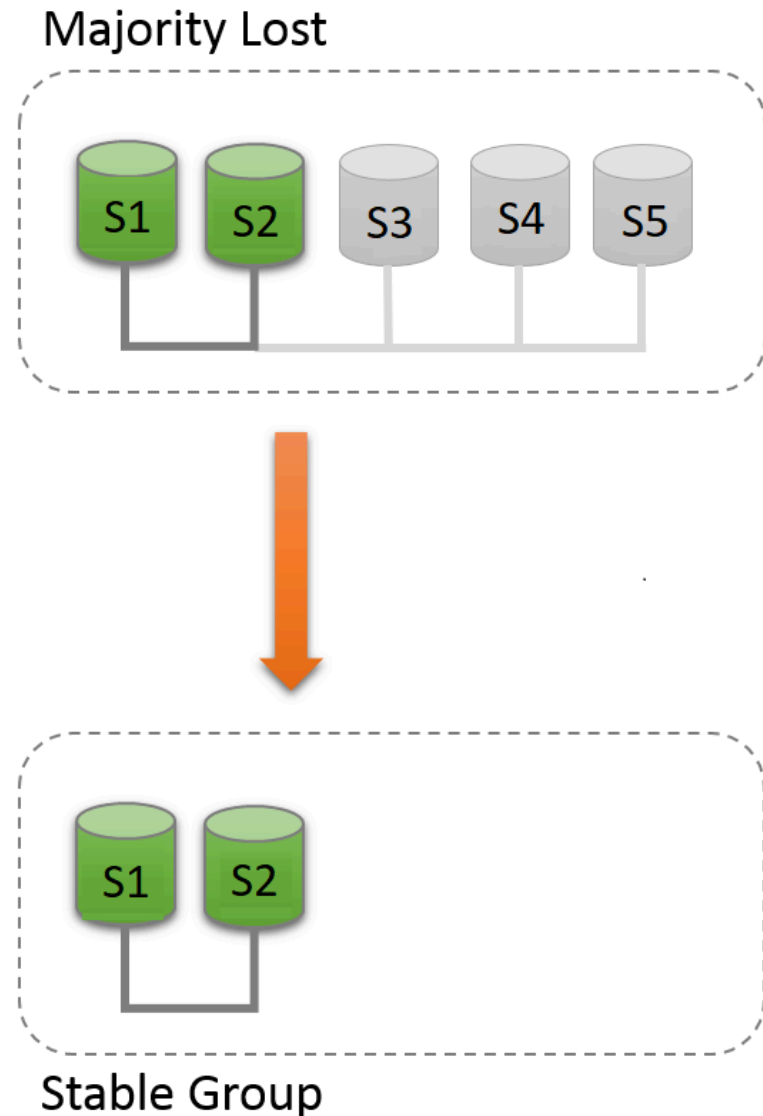
```
mysql> SELECT MEMBER_ID, MEMBER_STATE FROM performance_schema.replication_group_members;
+-----+-----+
| MEMBER_ID          | MEMBER_STATE |
+-----+-----+
| 1999b9fb-4aaf-11e6-bb54-28b2bd168d07 | UNREACHABLE |
| 199b2df7-4aaf-11e6-bb16-28b2bd168d07 | ONLINE      |
| 199bb88e-4aaf-11e6-babe-28b2bd168d07 | ONLINE      |
| 19ab72fc-4aaf-11e6-bb51-28b2bd168d07 | UNREACHABLE |
| 19b33846-4aaf-11e6-ba81-28b2bd168d07 | UNREACHABLE |
+-----+-----+
```

このテーブルは、大部分のサーバーにアクセスできないため、s1 が外部介入なしで進行できないグループになったことを示しています。この場合、このセクションで説明するように、システムを続行できるようにグループメンバーシップリストをリセットする必要があります。または、s1 および s2 でのグループレプリケーションの停止 (または完全に s1 および s2) を選択し、s3、s4 および s5 で発生したことを確認してから、グループレプリケーション (またはサーバー) を再起動することもできます。

パーティションのブロック解除

グループレプリケーションを使用すると、特定の構成を強制することでグループメンバーシップリストをリセットできます。たとえば、s1 および s2 のみがオンラインのサーバーである前述の場合、s1 および s2 のみで構成されるメンバーシップ構成を強制することを選択できます。これには、s1 および s2 に関する一部の情報を確認してから、`group_replication_force_members` 変数を使用する必要があります。

図 18.14 新規メンバーシップの強制



s1 および s2 がグループに残されている唯一のサーバーである状況に戻るとします。サーバー s3、s4 および s5 が予期せずグループから退出しました。サーバー s1 および s2 を続行するには、s1 および s2 のみを含むメンバーシップ構成を強制します。

警告

この手順では `group_replication_force_members` を使用するため、最後のリゾート処置とみなすようにしてください。クォーラムの損失をオーバーライドする場合にのみ、十分に注意して使用する必要があります。誤用された場合は、人工的なスプリットブレインシナリオを作成するか、システム全体を完全にブロックする可能性があります。

システムがブロックされ、現在の構成が次のようになっていることを思い出してください (s1 のローカル障害検出で認識されます):

```
mysql> SELECT MEMBER_ID, MEMBER_STATE FROM performance_schema.replication_group_members;
+-----+-----+
| MEMBER_ID | MEMBER_STATE |
+-----+-----+
```

```

| 1999b9fb-4aaf-11e6-bb54-28b2bd168d07 | UNREACHABLE |
| 199b2df7-4aaf-11e6-bb16-28b2bd168d07 | ONLINE      |
| 199bb88e-4aaf-11e6-babe-28b2bd168d07 | ONLINE      |
| 19ab72fc-4aaf-11e6-bb51-28b2bd168d07 | UNREACHABLE |
| 19b33846-4aaf-11e6-ba81-28b2bd168d07 | UNREACHABLE |
+-----+-----+

```

最初に行うことは、s1 および s2 のローカルアドレス (グループ通信識別子) を確認することです。s1 および s2 にログインし、その情報を次のように取得します。

```
mysql> SELECT @@group_replication_local_address;
```

s1 (127.0.0.1:10000) および s2 (127.0.0.1:10001) のグループ通信アドレスがわかったら、いずれかのサーバーでそれを使用して新しいメンバーシップ構成を注入し、クォーラムを失った既存の構成をオーバーライドできます。s1 でこれを行うには:

```
mysql> SET GLOBAL group_replication_force_members="127.0.0.1:10000,127.0.0.1:10001";
```

これにより、別の構成が強制され、グループのブロックが解除されます。この変更後にグループメンバーシップを確認するには、s1 と s2 の両方で `replication_group_members` をチェックします。s1 では最初。

```
mysql> SELECT MEMBER_ID, MEMBER_STATE FROM performance_schema.replication_group_members;
+-----+-----+
| MEMBER_ID          | MEMBER_STATE |
+-----+-----+
| b5ffe505-4ab6-11e6-b04b-28b2bd168d07 | ONLINE      |
| b60907e7-4ab6-11e6-afb7-28b2bd168d07 | ONLINE      |
+-----+-----+

```

次に、s2 で行います。

```
mysql> SELECT * FROM performance_schema.replication_group_members;
+-----+-----+
| MEMBER_ID          | MEMBER_STATE |
+-----+-----+
| b5ffe505-4ab6-11e6-b04b-28b2bd168d07 | ONLINE      |
| b60907e7-4ab6-11e6-afb7-28b2bd168d07 | ONLINE      |
+-----+-----+

```

新しいメンバーシップ構成を強制する場合は、グループから強制的に除外されるすべてのサーバーが実際に停止されていることを確認します。前述のシナリオでは、s3、s4 および s5 に実際にアクセスできないが、かわりにオンラインの場合、独自の機能パーティションを形成している可能性があります (5 つのうち 3 つであるため、大部分があります)。その場合、s1 および s2 を使用してグループメンバーシップリストを強制すると、人工的なスプリットブレイン状況が発生する可能性があります。したがって、新しいメンバーシップ構成を強制的に実行する前に、除外するサーバーが実際に停止されていることを確認し、停止されていない場合は続行する前に停止することが重要です。

`group_replication_force_members` システム変数を使用して新しいグループメンバーシップを強制し、グループのブロックを解除した後、必ずシステム変数をクリアしてください。`START GROUP_REPLICATION` ステートメントを発行するには、`group_replication_force_members` が空である必要があります。

18.4.5 IPv6 および IPv6 と IPv4 の混合グループのサポート

MySQL 8.0.14 から、グループレプリケーショングループメンバーは、グループ内の通信用の IPv4 アドレスのかわりに IPv6 アドレスを使用できます。IPv6 アドレスを使用するには、サーバーホストと MySQL Server インスタンスの両方のオペレーティングシステムが IPv6 をサポートするように構成されている必要があります。サーバーインスタンスの IPv6 サポートを設定する手順は、[セクション 5.1.13 「IPv6 サポート」](#) を参照してください。

IPv6 アドレス、またはそれらに解決されるホスト名は、メンバーが他のメンバーからの接続用に `group_replication_local_address` オプションで提供するネットワークアドレスとして指定できます。ポート番号とともに指定する場合、IPv6 アドレスは大口で囲んで指定する必要があります。次に例を示します:

```
group_replication_local_address= "[2001:db8:85a3:8d3:1319:8a2e:370:7348]:33061"
```

`group_replication_local_address` で指定されたネットワークアドレスまたはホスト名は、グループレプリケーションによって、レプリケーショングループ内のグループメンバーの一意的識別子として使用されます。サーバーインスタンスのグループレプリケーションのローカルアドレスとして指定されたホスト名が IPv4 アドレスと IPv6 アドレスの両

方に解決される場合、IPv4 アドレスは常にグループレプリケーション接続に使用されます。グループレプリケーションのローカルアドレスとして指定されたアドレスまたはホスト名が MySQL サーバーの SQL プロトコルのホストおよびポートと同じではなく、サーバーインスタンスの `bind_address` システム変数で指定されていません。グループレプリケーションの IP アドレス権限 ([セクション 18.5.1 「グループレプリケーション IP アドレスの権限」](#) を参照) のために、`group_replication_local_address` の各グループメンバーに指定するアドレスを、レプリケーショングループ内の他のサーバー上の `group_replication_ip_allowlist` (MySQL 8.0.22 から) または `group_replication_ip_whitelist` システム変数のリストに追加する必要があります。

レプリケーショングループには、IPv6 アドレスをグループレプリケーションローカルアドレスとして提示するメンバーと、IPv4 アドレスを提示するメンバーの組合せを含めることができます。サーバーがこのような混合グループに参加する場合、シードメンバーが `group_replication_group_seeds` オプションで通知するプロトコル (IPv4 か IPv6 かにかかわらず) を使用して、シードメンバーとの初期接続を確立する必要があります。参加メンバーに IPv4 Group Replication のローカルアドレスがある場合、またはその逆の場合に、グループのシードメンバーのいずれかが IPv6 アドレスとともに `group_replication_group_seeds` オプションにリストされている場合は、必要なプロトコル (またはそのプロトコルのアドレスに解決されるホスト名) の参加メンバーの代替アドレスも設定して許可する必要があります。参加メンバーに適切なプロトコルの許可されたアドレスがない場合、その接続試行は拒否されます。代替アドレスまたはホスト名は、レプリケーショングループ内の他のサーバーの `group_replication_ip_allowlist` (MySQL 8.0.22) または `group_replication_ip_whitelist` システム変数にのみ追加する必要があり、参加メンバーの `group_replication_local_address` 値には追加できません (単一のアドレスのみを含めることができます)。

たとえば、サーバー A はグループのシードメンバーであり、`group_replication_group_seeds` オプションで IPv6 アドレスを通知するために、グループレプリケーションには次の構成設定があります:

```
group_replication_bootstrap_group=on
group_replication_local_address="[2001:db8:85a3:8d3:1319:8a2e:370:7348]:33061"
group_replication_group_seeds="[2001:db8:85a3:8d3:1319:8a2e:370:7348]:33061"
```

サーバー B はグループの参加メンバーであり、グループレプリケーションの次の構成設定があるため、IPv4 グループレプリケーションのローカルアドレスを持ちます:

```
group_replication_bootstrap_group=off
group_replication_local_address="203.0.113.21:33061"
group_replication_group_seeds="[2001:db8:85a3:8d3:1319:8a2e:370:7348]:33061"
```

サーバー B には、代替の IPv6 アドレス `2001:db8:8b0:40:3d9c:cc43:e006:19e8` もあります。サーバー B がグループに正常に参加するには、次の例のように、IPv4 Group Replication のローカルアドレスと代替 IPv6 アドレスの両方をサーバー A の許可リストにリストする必要があります:

```
group_replication_ip_allowlist=
"203.0.113.0/24,2001:db8:85a3:8d3:1319:8a2e:370:7348,
2001:db8:8b0:40:3d9c:cc43:e006:19e8"
```

グループレプリケーションの IP アドレス権限のベストプラクティスとして、サーバー B (および他のすべてのグループメンバー) は、セキュリティ要件で特に要求されないがぎり、サーバー A と同じ許可リストを持つ必要があります。

レプリケーショングループの一部またはすべてのメンバーが、グループレプリケーションでの IPv6 アドレスの使用をサポートしていない古い MySQL Server バージョンを使用している場合、IPv6 アドレス (またはそれに解決されるホスト名) をグループレプリケーションローカルアドレスとして使用してグループに参加することはできません。これは、少なくとも 1 つの既存のメンバーが IPv6 アドレスを使用している場合と、この参加試行をサポートしていない新しいメンバーの両方に適用されます。また、新しいメンバーが IPv6 アドレスを使用して参加しようとしたが、このアドレスをサポートしていないメンバーがグループに少なくとも 1 つ含まれている場合にも適用されます。どの状況でも、新しいメンバーは参加できません。参加メンバーがグループ通信用の IPv4 アドレスを提示するようにするには、`group_replication_local_address` の値を IPv4 アドレスに変更するか、参加メンバーの既存のホスト名を IPv4 アドレスに解決するように DNS を構成します。すべてのグループメンバーを、グループレプリケーション用の IPv6 をサポートする MySQL Server バージョンにアップグレードした後、各メンバーの `group_replication_local_address` 値を IPv6 アドレスに変更するか、IPv6 アドレスを表示するように DNS を構成できます。`group_replication_local_address` の値の変更は、Group Replication を停止して再起動した場合にのみ有効になります。

IPv6 アドレスは、`group_replication_advertise_recovery_endpoints` システム変数を使用して MySQL 8.0.21 から指定できる分散リカバリエンドポイントとしても使用できます。このリストで使用されるアドレスにも同じルールが適用されます。[セクション 18.4.3.1 「分散リカバリの接続」](#) を参照してください。

18.4.6 グループレプリケーションでの MySQL Enterprise Backup の使用

MySQL Enterprise Backup は、MySQL Enterprise Edition で使用可能な MySQL Server の商用ライセンスバックアップユーティリティです。このセクションでは、MySQL Enterprise Backup を使用してグループレプリケーションメンバーをバックアップしてからリストアする方法について説明します。同じ方法を使用して、新しいメンバーをグループにすばやく追加できます。

MySQL Enterprise Backup を使用したグループレプリケーションメンバーのバックアップ

グループレプリケーションメンバーのバックアップは、スタンドアロンの MySQL インスタンスのバックアップと似ています。次の手順は、MySQL Enterprise Backup を使用してバックアップを実行する方法をすでに理解していることを前提としています。そうでない場合は、「MySQL Enterprise Backup 8.0 ユーザーガイド」(特に [Backing Up a Database Server](#))を確認してください。Grant MySQL Privileges to Backup Administrator および [Using MySQL Enterprise Backup with Group Replication](#) で説明されている要件にも注意してください。

同じ名前のホストで実行されている s1、s2 および s3 の 3 つのメンバーを持つ次のグループについて考えてみます:

```
mysql> SELECT member_host, member_port, member_state FROM performance_schema.replication_group_members;
+-----+-----+-----+
| member_host | member_port | member_state |
+-----+-----+-----+
| s1          | 3306        | ONLINE       |
| s2          | 3306        | ONLINE       |
| s3          | 3306        | ONLINE       |
+-----+-----+-----+
```

MySQL Enterprise Backup を使用して、そのホストで次のコマンドなどを発行し、s2 のバックアップを作成します:

```
s2> mysqlbackup --defaults-file=/etc/my.cnf --backup-image=/backups/my.mbi_`date +%d%m_%H%M` \
--backup-dir=/backups/backup_`date +%d%m_%H%M` --user=root -p \
--host=127.0.0.1 backup-to-image
```

メモ

- MySQL Enterprise Backup 8.0.18 以前の場合システム変数 `sql_require_primary_key` がグループに対して ON に設定されている場合、MySQL Enterprise Backup はバックアップの進行状況をサーバーに記録できません。これは、サーバー上の `backup_progress` テーブルが CSV テーブルであり、主キーがサポートされていないためです。その場合、`mysqlbackup` はバックアップ操作中に次の警告を発行します:

```
181011 11:17:06 MAIN WARNING: MySQL query 'CREATE TABLE IF NOT EXISTS
mysql.backup_progress( `backup_id` BIGINT NOT NULL, `tool_name` VARCHAR(4096)
NOT NULL, `error_code` INT NOT NULL, `error_message` VARCHAR(4096) NOT NULL,
`current_time` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP, `current_state` VARCHAR(200) NOT NULL ) ENGINE=CSV
DEFAULT CHARSET=utf8 COLLATE=utf8_bin': 3750, Unable to create a table
without PK, when system variable 'sql_require_primary_key' is set. Add a PK
to the table or unset this variable to avoid this message. Note that tables
without PK can cause performance problems in row-based replication, so please
consult your DBA before changing this setting.
181011 11:17:06 MAIN WARNING: This backup operation's progress info cannot be
logged.
```

ただし、`mysqlbackup` によるバックアップの終了は妨げられません。

- MySQL Enterprise Backup 8.0.20 以前の場合では、セカンダリメンバーをバックアップするときに、MySQL Enterprise Backup が読取り専用サーバーインスタンスにバックアップステータスおよびメタデータを書き込むことができないため、バックアップ操作中に次のような警告が発行される場合があります:

```
181113 21:31:08 MAIN WARNING: This backup operation cannot write to backup
progress. The MySQL server is running with the --super-read-only option.
```

警告を回避するには、`backup` コマンドで `--no-history-logging` オプションを使用します。これは MySQL Enterprise Backup 8.0.21 以上の問題ではありません。詳細は、[Using MySQL Enterprise Backup with Group Replication](#) を参照してください。

失敗したメンバーのリストア

いずれかのメンバー (次の例では `s3`) が破損しているとします。グループメンバー `s2` の最新のバックアップを使用して、`s3` をリストアできます。リストアを実行するステップは次のとおりです:

1. `s2` のバックアップを `s3` のホストにコピーします。バックアップをコピーする正確な方法は、使用可能なオペレーティングシステムおよびツールによって異なります。この例では、ホストが両方とも Linux サーバーであると想定し、SCP を使用してホスト間でファイルをコピーします:

```
s2/backups> scp my.mbi_2206_1429 s3:/backups
```

2. バックアップをリストアします。ターゲットホスト (この場合は `s3` のホスト) に接続し、MySQL Enterprise Backup を使用してバックアップをリストアします。ステップは次のとおりです:

- a. 破損したサーバーがまだ実行されている場合は、停止します。たとえば、`systemd` を使用する Linux デイストリビューションの場合:

```
s3> systemctl stop mysqld
```

- b. 破損したサーバーデータディレクトリ (`auto.cnf` および `mysqld-auto.cnf` が存在する場合) 内の 2 つの構成ファイルを、データディレクトリ外の安全な場所にコピーして保持します。これは、次のステップに必要な `server UUID` および [セクション 5.1.9.3 「永続化されるシステム変数」](#) (使用する場合) を保持するためのものです。

- c. `s3` のデータディレクトリ内のすべてのコンテンツを削除します。例:

```
s3> rm -rf /var/lib/mysql/*
```

システム変数 `innodb_data_home_dir`、`innodb_log_group_home_dir` および `innodb_undo_directory` がデータディレクトリ以外のディレクトリを指している場合は、それらも空にする必要があります。そうしないと、リストア操作は失敗します。

- d. `s2` のバックアップを `s3` のホストにリストアします:

```
s3> mysqlbackup --defaults-file=/etc/my.cnf \  
--datadir=/var/lib/mysql \  
--backup-image=backups/my.mbi_2206_1429 \  
--backup-dir=/tmp/restore_`date +%d%m_%H%M` copy-back-and-apply-log
```

注記

前述のコマンドでは、`s2` と `s3` のバイナリログとリレーログのベース名が同じで、2 つのサーバー上の同じ場所にあることを前提としています。これらの条件が満たされていない場合は、`--log-bin` および `--relay-log` オプションを使用してバイナリログを復元し、`s3` 上の元のファイルパスにログをリレーするようにしてください。たとえば、`s3` でバイナリログベース名が `s3-bin` で、リレーログベース名が `s3-relay-bin` であることがわかっている場合、`restore` コマンドは次のようになります:

```
mysqlbackup --defaults-file=/etc/my.cnf \  
--datadir=/var/lib/mysql \  
--backup-image=backups/my.mbi_2206_1429 \  
--log-bin=s3-bin --relay-log=s3-relay-bin \  
--backup-dir=/tmp/restore_`date +%d%m_%H%M` copy-back-and-apply-log
```

バイナリログとリレーログを正しいファイルパスに復元できるため、復元プロセスが容易になります。何らかの理由で復元できない場合は、[新規メンバーとして再結合するための失敗したメンバーの再構築](#) を参照してください。

3. `s3` の `auto.cnf` ファイルをリストアします。レプリケーショングループに再度参加するには、リストアされたメンバーが以前にグループへの参加に使用したのと同じ `server_uuid` を持っている必要があります。前述のステップ 2 で保持した `auto.cnf` ファイルをリストアされたメンバーのデータディレクトリにコピーして、古いサーバー UUID を指定します。

注記

古い `auto.cnf` ファイルをリストアして、障害が発生したメンバーの元の `server_uuid` をリストアされたメンバーに提供できない場合は、リストアされたメンバーを新しいメンバーとしてグループに参加させる必要があります。その方法は、次の [新規メンバーとして再結合するための失敗したメンバーの再構築](#) の手順を参照してください。

- s3 の `mysqld-auto.cnf` ファイルをリストアします (s3 で永続システム変数が使用されている場合にのみ必要)。失敗したメンバーの構成に使用された [セクション5.1.9.3「永続化されるシステム変数」](#) の設定は、リストアされたメンバーに指定する必要があります。これらの設定は、前述のステップ 2 で保持した障害が発生したサーバーの `mysqld-auto.cnf` ファイルにあります。リストアされたサーバーのデータディレクトリにファイルをリストアします。ファイルのコピーがない場合の対処方法は、[永続化されたシステム変数のリストア](#) を参照してください。
- リストアされたサーバーを起動します。たとえば、`systemd` を使用する Linux ディストリビューションの場合:

```
systemctl start mysqld
```

注記

リストアするサーバーがプライマリメンバーである場合は、[プライマリメンバーのリストア](#) リストアされたサーバーを起動する前で説明されているステップを実行します。

- Group Replication を再起動します。 `mysql` クライアントなどを使用して再起動した s3 に接続し、次のコマンドを発行します:

```
mysql> START GROUP_REPLICATION;
```

リストアされたインスタンスがグループのオンラインメンバーになる前に、バックアップの作成後にグループに発生したトランザクションを適用する必要があります。これは Group Replication [distributed recovery](#) メカニズムを使用して実現され、`START GROUP_REPLICATION` ステートメントの発行後にプロセスが開始されます。リストアされたインスタンスのメンバーステータスを確認するには、次のコマンドを発行します:

```
mysql> SELECT member_host, member_port, member_state FROM performance_schema.replication_group_members;
+-----+-----+-----+
| member_host | member_port | member_state |
+-----+-----+-----+
| s1         | 3306        | ONLINE       |
| s2         | 3306        | ONLINE       |
| s3         | 3306        | RECOVERING   |
+-----+-----+-----+
```

これは、s3 がグループをキャッチアップするためにトランザクションを適用していることを示しています。残りのグループで捕捉されると、その `member_state` は `ONLINE` に変更されます:

```
mysql> SELECT member_host, member_port, member_state FROM performance_schema.replication_group_members;
+-----+-----+-----+
| member_host | member_port | member_state |
+-----+-----+-----+
| s1         | 3306        | ONLINE       |
| s2         | 3306        | ONLINE       |
| s3         | 3306        | ONLINE       |
+-----+-----+-----+
```

注記

リストアするサーバーがプライマリメンバーである場合、グループとの同期を取得して `ONLINE` になったら、[プライマリメンバーのリストア](#) の最後に記載されているステップを実行して、サーバーを起動する前に行った構成の変更を元に戻します。

これで、メンバーはバックアップから完全にリストアされ、グループの通常のメンバーとして機能します。

新規メンバーとして再結合するための失敗したメンバーの再構築

バイナリログまたはリレーログが破損している、あるいは単にバックアップから欠落しているなどの理由で、[失敗したメンバーのリストア](#) で前述の手順を実行できない場合があります。このような場合は、バックアップを使用してメ

メンバーを再構築し、新しいメンバーとしてグループに追加します。次のステップでは、再構築されたメンバーは、障害が発生したメンバーと同様に `s3` という名前であり、`s3` と同じホスト上で実行されていると想定しています:

1. `s2` のバックアップを `s3` のホストにコピーします。バックアップをコピーする正確な方法は、使用可能なオペレーティングシステムおよびツールによって異なります。この例では、ホストが両方とも Linux サーバーであり、SCP を使用してホスト間でファイルをコピーすることを前提としています:

```
s2/backups> scp my.mbi_2206_1429 s3:/backups
```

2. バックアップをリストアします。ターゲットホスト (この場合は `s3` のホスト) に接続し、MySQL Enterprise Backup を使用してバックアップをリストアします。ステップは次のとおりです:

- a. 破損したサーバーがまだ実行されている場合は、停止します。たとえば、`systemd` を使用する Linux ディストリビューションの場合:

```
s3> systemctl stop mysqld
```

- b. 破損したサーバーデータディレクトリに見つかった場合は、データディレクトリ外の安全な場所にコピーして、構成ファイル `mysqld-auto.cnf` を保持します。これは、あとで必要になるサーバー [セクション 5.1.9.3 「永続化されるシステム変数」](#) を保持するためのものです。

- c. `s3` のデータディレクトリ内のすべてのコンテンツを削除します。例:

```
s3> rm -rf /var/lib/mysql/*
```

システム変数 `innodb_data_home_dir`、`innodb_log_group_home_dir` および `innodb_undo_directory` がデータディレクトリ以外のディレクトリを指している場合は、それらも空にする必要があります。そうしないと、リストア操作は失敗します。

- d. `s2` のバックアップを `s3` のホストにリストアします。このアプローチでは、`s3` を新しいメンバーとして再構築するため、古いバイナリログおよびリレーログをバックアップで使用する必要はありません。したがって、これらのログがバックアップに含まれている場合は、`--skip-binlog` および `--skip-relaylog` オプションを使用して除外します:

```
s3> mysqlbackup --defaults-file=/etc/my.cnf \  
--datadir=/var/lib/mysql \  
--backup-image=/backups/my.mbi_2206_1429 \  
--backup-dir=/tmp/restore_`date +%d%m_%H%M` \  
--skip-binlog --skip-relaylog \  
copy-back-and-apply-log
```

注記

問題なくターゲットホストに転送できる正常なバイナリログおよびリレーログがバックアップにある場合は、前述の [失敗したメンバーのリストア](#) で説明した簡単な手順に従うことをお勧めします。

3. `s3` の `mysqld-auto.cnf` ファイルをリストアします (`s3` で永続システム変数が使用されている場合にのみ必要)。障害が発生したメンバーの構成に使用された [セクション 5.1.9.3 「永続化されるシステム変数」](#) の設定は、リストアされたサーバーに指定する必要があります。これらの設定は、前述のステップ 2 で保持した障害が発生したサーバーの `mysqld-auto.cnf` ファイルにあります。リストアされたサーバーのデータディレクトリにファイルをリストアします。ファイルのコピーがない場合の対処方法は、[永続化されたシステム変数のリストア](#) を参照してください。

注記

破損したサーバー `auto.cnf` ファイルを新しいメンバーのデータディレクトリにリストアしないでください。再構築された `s3` が新しいメンバーとしてグループに参加すると、新しいサーバー UUID が割り当てられます。

4. リストアされたサーバーを起動します。たとえば、systemd を使用する Linux ディストリビューションの場合:

```
systemctl start mysqld
```

注記

リストアするサーバーがプライマリメンバーである場合は、[プライマリメンバーのリストア](#) リストアされたサーバーを起動する前で説明されているステップを実行します。

5. グループレプリケーションに参加するようにリストアされたメンバーを再構成します。mysql クライアントを使用してリストアされたサーバーに接続し、次のコマンドを使用してソースおよびレプリカ情報をリセットします:

```
mysql> RESET MASTER;
```

```
mysql> RESET SLAVE ALL;
Or from MySQL 8.0.22:
mysql> RESET REPLICHA ALL;
```

[distributed recovery](#) の Group Replication 組込みメカニズムを使用して、リストアされたサーバーを自動的にリカバリできるようにするには、サーバーの `gtid_executed` 変数を構成します。これを行うには、`s2` のバックアップに含まれる `backup_gtid_executed.sql` ファイルを使用します。これは通常、リストアされたメンバーデータディレクトリの下にリストアされます。バイナリロギングを無効にし、`backup_gtid_executed.sql` ファイルを使用して `gtid_executed` を構成してから、mysql クライアントで次のステートメントを発行してバイナリロギングを再度有効にします:

```
mysql> SET SQL_LOG_BIN=OFF;
mysql> SOURCE datadir/backup_gtid_executed.sql
mysql> SET SQL_LOG_BIN=ON;
```

次に、メンバーで [Group Replication user credentials](#) を構成します:

```
mysql> CHANGE MASTER TO MASTER_USER='rpl_user', MASTER_PASSWORD='password' /
FOR CHANNEL 'group_replication_recovery';

Or from MySQL 8.0.23:
mysql> CHANGE REPLICATION SOURCE TO SOURCE_USER='rpl_user', SOURCE_PASSWORD='password' /
FOR CHANNEL 'group_replication_recovery';
```

6. Group Replication を再起動します。mysql クライアントを使用して、リストアされたサーバーに次のコマンドを発行します:

```
mysql> START GROUP_REPLICATION;
```

リストアされたインスタンスがグループのオンラインメンバーになる前に、バックアップの作成後にグループに発生したトランザクションを適用する必要があります。これは Group Replication [distributed recovery](#) メカニズムを使用して実現され、`START GROUP_REPLICATION` ステートメントの発行後にプロセスが開始されます。リストアされたインスタンスのメンバーステータスを確認するには、次のコマンドを発行します:

```
mysql> SELECT member_host, member_port, member_state FROM performance_schema.replication_group_members;
+-----+-----+-----+
| member_host | member_port | member_state |
+-----+-----+-----+
| s3         | 3306        | RECOVERING   |
| s2         | 3306        | ONLINE       |
| s1         | 3306        | ONLINE       |
+-----+-----+-----+
```

これは、`s3` がグループをキャッチアップするためにトランザクションを適用していることを示しています。残りのグループで捕捉されると、その `member_state` は `ONLINE` に変更されます:

```
mysql> SELECT member_host, member_port, member_state FROM performance_schema.replication_group_members;
+-----+-----+-----+
| member_host | member_port | member_state |
+-----+-----+-----+
| s3         | 3306        | ONLINE       |
| s2         | 3306        | ONLINE       |
| s1         | 3306        | ONLINE       |
+-----+-----+-----+
```


注記

リストアするサーバーがプライマリメンバーである場合、グループとの同期を取得して ONLINE になったら、**プライマリメンバーのリストア**の最後に記載されているステップを実行して、サーバーを起動する前に行った構成の変更を元に戻します。

これで、メンバーは新しいメンバーとしてグループに復元されました。

永続化されたシステム変数のリストア。 `mysqlbackup` では、**セクション5.1.9.3「永続化されるシステム変数」**-the ファイルのバックアップまたは保存をサポートしていません `mysqld-auto.cnf` はバックアップに含まれていません。 リストアされたメンバーを永続変数設定で開始するには、次のいずれかを実行する必要があります：

- 破損したサーバーからの `mysqld-auto.cnf` ファイルのコピーを保持し、リストアしたサーバーデータディレクトリにコピーします。
- `mysqld-auto.cnf` ファイルをグループの別のメンバーからリストアされたサーバーデータディレクトリにコピーします (そのメンバーに破損したメンバーと同じ永続システム変数設定がある場合)。
- リストアされたサーバーの起動後、グループレプリケーションを再起動する前に、`mysql` クライアントを介してすべてのシステム変数を手動で永続化された値に設定します。

プライマリメンバーのリストア。 リストアされたメンバーがグループ内のプライマリである場合は、グループレプリケーション分散リカバリプロセス中にリストアされたデータベースへの書き込みを防ぐように注意する必要があります。 クライアントによるグループへのアクセス方法に応じて、リストアされたメンバーがネットワーク上でアクセス可能になると、メンバーがグループ外でミスしたアクティビティのキャッチアップを終了する前に、そのメンバーに対して DML ステートメントが実行される可能性があります。これを回避するには、リストアされたサーバーを起動する前でサーバーオプションファイルに次のシステム変数を構成します：

```
group_replication_start_on_boot=OFF
super_read_only=ON
event_scheduler=OFF
```

これらの設定により、起動時にメンバーが読み取り専用になり、分散リカバリプロセス中にメンバーがグループで捕捉されている間、イベントスケジューラがオフになります。適切なエラー処理は、リストアされたメンバーでこの期間中に一時的に DML 操作を実行できないように、クライアントでも構成する必要があります。 リストアプロセスが完全に完了し、リストアされたメンバーがグループの残りの部分と同期したら、これらの変更を元に戻し、イベントスケジューラを再起動します：

```
mysql> SET global event_scheduler=ON;
```

メンバーオプションファイルで次のシステム変数を編集して、次回の起動時に正しく構成されるようにします：

```
group_replication_start_on_boot=ON
super_read_only=OFF
event_scheduler=ON
```

18.5 グループレプリケーションセキュリティ

このセクションでは、グループを保護する方法、グループのメンバー間の接続を保護する方法、または IP アドレス許可リストを使用してセキュリティ境界を確立する方法について説明します。

18.5.1 グループレプリケーション IP アドレスの権限

Group Replication プラグインでは、受信グループ通信システム接続を受け入れることができるホストの許可リストを指定できます。サーバー `s1` で `allowlist` を指定した場合、グループ通信を行うためにサーバー `s2` が `s1` への接続を確立すると、`s1` はまず `allowlist` をチェックしてから `s2` からの接続を受け入れます。 `s2` が許可リストにある場合、`s1` は接続を受け入れます。そうでない場合、`s1` は `s2` による接続試行を拒否します。 MySQL 8.0.22 からは、システム変数 `group_replication_ip_allowlist` を使用して許可リストが指定され、MySQL 8.0.22 より前のリリースでは、システム変数 `group_replication_ip_whitelist` が使用されます。新しいシステム変数は古いシステム変数と同じように機能しますが、用語のみが変更されています。

`allowlist` を明示的に指定しない場合、グループ通信エンジン (XCom) はホスト上のアクティブなインタフェースを自動的にスキャンし、プライベートサブネットワーク上のアドレスを持つインタフェースと、各インタフェースに構成

されているサブネットマスクを識別します。これらのアドレス、および IPv4 と (MySQL 8.0.14 の) IPv6 の `localhost` IP アドレスは、自動 Group Replication 許可リストの作成に使用されます。したがって、自動許可リストには、適切なサブネットマスクが適用された後に次の範囲でホストに対して検出された IP アドレスが含まれます:

```
IPv4 (as defined in RFC 1918)
10/8 prefix (10.0.0.0 - 10.255.255.255) - Class A
172.16/12 prefix (172.16.0.0 - 172.31.255.255) - Class B
192.168/16 prefix (192.168.0.0 - 192.168.255.255) - Class C

IPv6 (as defined in RFC 4193 and RFC 5156)
fc00::/7 prefix - unique-local addresses
fe80::/10 prefix - link-local unicast addresses

127.0.0.1 - localhost for IPv4
::1 - localhost for IPv6
```

ホストに対して自動的に許可されたアドレスを示すエントリがエラーログに追加されます。

プライベートアドレスの自動許可リストはプライベートネットワーク外のサーバーからの接続には使用できないため、パブリック IP 上にインタフェースがある場合でも、サーバーはデフォルトで外部ホストからのグループレプリケーション接続を許可しません。異なるマシン上にあるサーバーインスタンス間のグループレプリケーション接続の場合、パブリック IP アドレスを指定し、明示的な許可リストとして指定する必要があります。許可リストにエントリを指定した場合、プライベートアドレスおよび `localhost` アドレスは自動的に追加されないため、これらのいずれかを使用する場合は、明示的に指定する必要があります。

許可リストを手動で指定するには、`group_replication_ip_allowlist` (MySQL 8.0.22) または `group_replication_ip_whitelist` システム変数を使用します。サーバーがレプリケーショングループのアクティブメンバーである間は、そのサーバーの許可リストを変更できません。メンバーがアクティブな場合は、許可リストを変更する前に `STOP GROUP_REPLICATION` を実行し、後で `START GROUP_REPLICATION` を実行する必要があります。

`allowlist` には、各メンバーの `group_replication_local_address` システム変数で指定された IP アドレスまたはホスト名が含まれている必要があります。このアドレスは、MySQL サーバーの SQL プロトコルのホストおよびポートと同じではなく、サーバーインスタンスの `bind_address` システム変数で指定されていません。サーバーインスタンスのグループレプリケーションのローカルアドレスとして使用されるホスト名が IPv4 アドレスと IPv6 アドレスの両方に解決される場合、グループレプリケーション接続には IPv4 アドレスが優先されます。

分散リカバリエンドポイントとして指定された IP アドレス、および分散リカバリ (デフォルト) に使用される場合はメンバー標準 SQL クライアント接続の IP アドレスを許可リストに追加する必要はありません。`allowlist` は、メンバーごとに `group_replication_local_address` によって指定されたアドレスに対してのみ使用されます。分散リカバリのためにアドレスを取得するには、参加メンバーは `allowlist` によって許可されたグループへの初期接続を持っている必要があります。

`allowlist` では、次の任意の組合せを指定できます:

- IPv4 アドレス (`198.51.100.44` など)
- CIDR 表記を使用した IPv4 アドレス (`192.0.2.21/24` など)
- MySQL 8.0.14 からの IPv6 アドレス (`2001:db8:85a3:8d3:1319:8a2e:370:7348` など)
- MySQL 8.0.14 からの CIDR 表記の IPv6 アドレス (`2001:db8:85a3:8d3::/64` など)
- ホスト名 (`example.org` など)
- CIDR 表記法を使用したホスト名 (`www.example.com/24` など)

MySQL 8.0.14 より前では、ホスト名は IPv4 アドレスにのみ解決できました。MySQL 8.0.14 から、ホスト名は IPv4 アドレス、IPv6 アドレス、またはその両方に解決できます。ホスト名が IPv4 アドレスと IPv6 アドレスの両方に解決される場合、IPv4 アドレスは常にグループレプリケーション接続に使用されます。CIDR 表記をホスト名または IP アドレスと組み合わせて使用すると、特定のネットワーク接頭辞を持つ IP アドレスのブロックを許可できますが、指定したサブネット内のすべての IP アドレスが制御下にあることを確認してください。

注記

IP アドレスが許可リストにないために IP アドレスからの接続試行が拒否された場合、拒否メッセージは常に IPv6 形式で IP アドレスを出力します。IPv4 アドレスの前には、この形

式 (IPv4-mapped IPv6 アドレス) の `::ffff:` が付きます。許可リストで IPv4 アドレスを指定するためにこの形式を使用する必要はありません。標準の IPv4 形式を使用します。

許可リストを変更するには、メンバーのグループレプリケーションを停止して再起動する必要があります。allowlist の各エントリはカンマで区切る必要があります。例:

```
mysql> STOP GROUP_REPLICATION;
mysql> SET GLOBAL group_replication_ip_allowlist="192.0.2.21/24,198.51.100.44,203.0.113.0/24,2001:db8:85a3:8d3:1319:8a2e:370:7348,example.org,www.example.org";
mysql> START GROUP_REPLICATION;
```

レプリケーショングループに参加するには、グループへの参加をリクエストするシードメンバーでサーバーが許可されている必要があります。通常、これはレプリケーショングループのブートストラップメンバーですが、グループに参加するサーバーの構成で `group_replication_group_seeds` オプションによってリストされた任意のサーバーを指定できます。結合メンバーに IPv4 `group_replication_local_address` がある場合、またはその逆の場合に、グループのシードメンバーのいずれかが IPv6 アドレスとともに `group_replication_group_seeds` オプションにリストされている場合は、シードメンバーによって提供されるプロトコル (またはそのプロトコルのアドレスに解決されるホスト名) の結合メンバーの代替アドレスも設定して許可する必要があります。これは、サーバーがレプリケーショングループに参加するときに、シードメンバーが `group_replication_group_seeds` オプションで通知するプロトコル (IPv4 か IPv6 にかかわらず) を使用して、シードメンバーとの初期接続を行う必要があるためです。参加メンバーに適切なプロトコルの許可されたアドレスがない場合、その接続試行は拒否されます。IPv4 と IPv6 の混在レプリケーショングループの管理の詳細は、[セクション 18.4.5 「IPv6 および IPv6 と IPv4 の混合グループのサポート」](#) を参照してください。

レプリケーショングループが再構成されると (たとえば、新しいプライマリが選択された場合、メンバーが参加または離脱した場合)、グループメンバーは自身の間の接続を再確立します。再構成後にレプリケーショングループに含まれなくなったサーバーによってのみグループメンバーが許可されている場合、レプリケーショングループ内で許可されていない残りのサーバーには再接続できません。このシナリオを完全に回避するには、レプリケーショングループのメンバーであるすべてのサーバーに同じ allowlist を指定します。

注記

たとえば、異なるサブネットを分離しておくために、セキュリティ要件に応じて異なるグループメンバーに異なる許可リストを構成できます。セキュリティ要件を満たすために異なる許可リストを構成する必要がある場合は、レプリケーショングループ内の許可リスト間に十分な重複があることを確認して、元のシードメンバーが存在しない場合にサーバーが再接続できる可能性を最大化します。

ホスト名の場合、名前解決は、別のサーバーによって接続リクエストが行われた場合にのみ行われます。解決できないホスト名は許可リストの検証で考慮されず、警告メッセージがエラーログに書き込まれます。前方確認の逆引き DNS (FCrDNS) 検証は、解決されたホスト名に対して実行されます。

警告

ホスト名は本質的に許可リストの IP アドレスより安全性が低くなります。FCrDNS の検証は適切なレベルの保護を提供しますが、特定のタイプの攻撃によって危険にさらされる可能性があります。厳密に必要な場合にのみ許可リストにホスト名を指定し、名前解決に使用されるすべてのコンポーネント (DNS サーバーなど) が制御下に保持されていることを確認します。外部コンポーネントを使用しないように、hosts ファイルを使用して名前解決をローカルに実装することもできます。

18.5.2 Secure Socket Layer (SSL) を使用したグループ通信接続の保護

セキュアソケットは、グループのメンバー間のグループ通信接続に使用できます。Group Replication システム変数 `group_replication_ssl_mode` は、グループ通信接続の SSL の使用をアクティブ化し、接続のセキュリティモードを指定するために使用されます。デフォルト設定は、SSL が使用されないことを意味します。このオプションには、次の値を指定できます:

表 18.2 group_replication_ssl_mode 構成値

値	説明
DISABLED	暗号化されていない接続を確立します (デフォルト)。
REQUIRED	サーバーがセキュアな接続をサポートしている場合は、セキュアな接続を確立します。

値	説明
VERIFY_CA	REQUIRED と似ていますが、さらに、構成された認証局 (CA) 証明書に対してサーバー TLS 証明書を検証します。
VERIFY_IDENTITY	VERIFY_CA と似ていますが、さらに、サーバー証明書が接続が試行されるホストと一致することを確認します。

グループレプリケーショングループ通信接続の残りの構成は、サーバー SSL 構成から取得されます。サーバー SSL を構成するオプションの詳細は、[暗号化接続のコマンドオプション](#) を参照してください。グループレプリケーショングループの通信接続に適用されるサーバー SSL オプションは、次のとおりです:

表 18.3 SSL オプション

サーバー構成	説明
ssl_key	PEM 形式の SSL 秘密キーファイルのパス名。クライアント側では、これはクライアント秘密キーです。サーバー側では、これはサーバーの秘密キーです。
ssl_cert	PEM 形式の SSL 公開キー証明書ファイルのパス名。クライアント側では、これはクライアント公開キー証明書です。サーバー側では、これはサーバー公開キー証明書です。
ssl_ca	PEM 形式の認証局 (CA) 証明書ファイルのパス名。
ssl_cacpath	PEM 形式の信頼できる SSL 認証局 (CA) 証明書ファイルを含むディレクトリのパス名。
ssl_crl	PEM 形式の証明書失効リストを含むファイルのパス名。
ssl_crlpath	PEM 形式の証明書失効リストファイルを含むディレクトリのパス名。
ssl_cipher	暗号化された接続に許可される暗号のリスト。
tls_version	サーバーが暗号化された接続に対して許可する TLS プロトコルのリスト。
tls_ciphersuites	サーバーが暗号化された接続に対して許可する TLSv1.3 暗号スイート。

重要

- TLSv1.3 プロトコルのサポートは、MySQL 8.0.16 の MySQL Server で使用できません (MySQL が OpenSSL 1.1.1 以上を使用してコンパイルされている場合)。グループレプリケーションは、MySQL 8.0.18 からの TLSv1.3 をサポートしています。MySQL 8.0.16 および MySQL 8.0.17 では、サーバーが TLSv1.3 をサポートしている場合、プロトコルはグループ通信エンジンではサポートされず、Group Replication では使用できません。
- [tls_version](#) システム変数で指定された TLS プロトコルのリストで、指定されたバージョンが連続していることを確認します (たとえば、[TLSv1,TLSv1.1,TLSv1.2](#))。プロトコルのリストにギャップがある場合 (たとえば、[TLSv1,TLSv1.2](#) を指定し、TLS 1.1 を省略した場合)、Group Replication はグループ通信接続を確立できない可能性があります。
- MySQL 8.0.18 では、TLSv1.3 を分散リカバリ接続のグループレプリケーションで使用できますが、[group_replication_recovery_tls_version](#) および [group_replication_recovery_tls_ciphersuites](#) システム変数は使用できません。したがって、ドナーサーバーでは、[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#) にリストされているように、デフォルトで有効になっている TLSv1.3 暗号スイートを少なくとも 1 つ使用できる必要があります。MySQL 8.0.19 から、オプションを使用して、必要に応じてデフォルト以外の暗号スイートのみを含む任意の暗号スイートのクライアントサポートを構成できます。

レプリケーショングループでは、OpenSSL は、すべてのメンバーでサポートされている最上位 TLS プロトコルの使用をネゴシエーションします。TLSv1.3 ([tls_version=TLSv1.3](#)) のみを使用するように構成された結合メンバーは、既

存のメンバーが TLSv1.3 をサポートしていないレプリケーショングループに参加できません。その場合、グループメンバーは TLS プロトコルバージョンが低いからです。メンバーをグループに参加させるには、参加メンバーを構成して、既存のグループメンバーでサポートされている下位 TLS プロトコルバージョンの使用も許可する必要があります。逆に、参加メンバーが TLSv1.3 をサポートしていないが、既存のグループメンバーが相互の接続にそのバージョンをすべて使用している場合、既存のグループメンバーがすでに適切な下位 TLS プロトコルバージョンの使用を許可しているか、それを許可するように構成すると、メンバーは参加できます。この状況では、OpenSSL は、各メンバーから参加メンバーへの接続に TLS プロトコルの下位バージョンを使用します。他の既存のメンバーへの各メンバー接続では、両方のメンバーがサポートする使用可能な最高のプロトコルが引き続き使用されます。

MySQL 8.0.16 から、実行時に `tls_version` システム変数を変更して、サーバーで許可されている TLS プロトコルバージョンのリストを変更できます。Group Replication の場合、コンテキストを定義するシステム変数の現在の値からサーバー TLS コンテキストを再構成する `ALTER INSTANCE RELOAD TLS` ステートメントは、Group Replication の実行中に Group Replication グループ通信接続の TLS コンテキストを変更しないことに注意してください。これらの接続に再構成を適用するには、`STOP GROUP_REPLICATION` の後に `START GROUP_REPLICATION` を実行して、`tls_version` システム変数を変更したメンバーでグループレプリケーションを再起動する必要があります。同様に、グループのすべてのメンバーが上位または下位の TLS プロトコルバージョンを使用するように変更する場合は、許可された TLS プロトコルバージョンのリストを変更した後に、メンバーでグループレプリケーションのローリング再起動を実行して、ローリング再起動の完了時に OpenSSL が上位 TLS プロトコルバージョンの使用をネゴシエートできるようにする必要があります。実行時に許可される TLS プロトコルバージョンのリストを変更する手順については、[セクション 6.3.2 「暗号化された接続 TLS プロトコルおよび暗号」](#) and [サーバー側のランタイム構成および暗号化された接続の監視](#) を参照してください。

次の例は、サーバーで SSL を構成し、グループレプリケーショングループ通信接続の SSL をアクティブ化する `my.cnf` ファイルのセクションを示しています：

```
[mysqld]
ssl_ca = "cacert.pem"
ssl_capath = "../ca_directory"
ssl_cert = "server-cert.pem"
ssl_cipher = "DHE-RSA-AES256-SHA"
ssl_crl = "crl-server-revoked.crl"
ssl_crlpath = "../crl_directory"
ssl_key = "server-key.pem"
group_replication_ssl_mode= REQUIRED
```

重要

コンテキストを定義するシステム変数の現在の値からサーバー TLS コンテキストを再構成する `ALTER INSTANCE RELOAD TLS` ステートメントは、Group Replication の実行中に Group Replication グループ通信接続の TLS コンテキストを変更しません。これらの接続に再構成を適用するには、`STOP GROUP_REPLICATION` を実行してから `START GROUP_REPLICATION` を実行し、Group Replication を再起動する必要があります。

分散リカバリのために結合メンバーと既存のメンバーの間で行われた接続は、前述のオプションではカバーされません。これらの接続では、[セクション 18.5.3.2 「分散リカバリのための Secure Socket Layer \(SSL\) 接続」](#) で説明されているグループレプリケーション専用分散リカバリ SSL オプションが使用されます。

18.5.3 分散リカバリ接続の保護

メンバーがグループに参加すると、リモートクローニング操作 (使用可能で適切な場合) と非同期レプリケーション接続の組合せを使用して分散リカバリが実行されます。分散リカバリの詳細は、[セクション 18.4.3 「分散リカバリ」](#) を参照してください。

既存のメンバーが分散リカバリのために参加メンバーに提供する接続は、グループのオンラインメンバー間の通信にグループレプリケーションで使用される接続と同じではありません。MySQL 8.0.20 まで、グループメンバーは、MySQL Server `hostname` および `port` システム変数で指定されている分散リカバリのためにメンバーを結合するための標準 SQL クライアント接続を提供します。MySQL 8.0.21 から、グループメンバーは分散リカバリエンドポイントの代替リストをメンバー参加専用のクライアント接続として通知できます。詳細は、[セクション 18.4.3.1 「分散リカバリの接続」](#) を参照してください。

グループ内の分散リカバリ接続を保護するには、レプリケーションユーザーのユーザー資格証明が適切に保護されていることを確認し、可能な場合は分散リカバリ接続に SSL を使用します。

18.5.3.1 分散リカバリのためのセキュアなユーザー資格証明

バイナリログからの状態転送では、グループレプリケーションがメンバー間の直接レプリケーションチャンネルを確立できるように、適切な権限を持つレプリケーションユーザーが必要です。すべてのグループメンバーで分散リカバリに同じレプリケーションユーザーが使用されます。MySQL 8.0.17 から使用可能な分散リカバリの一部としてリモートクローニング操作の使用をサポートするようにグループメンバーが設定されている場合、このレプリケーションユーザーはドナーのクローンユーザーとしても使用され、このロールに対する正しい権限も必要です。このユーザーを設定する手順の詳細は、[セクション18.2.1.3「分散リカバリのユーザー資格証明」](#)を参照してください。

ユーザー資格証明を保護するには、ユーザーアカウントとの接続に SSL を要求し、(MySQL 8.0.21 から) グループレプリケーションの起動時に、レプリカステータステーブルに格納するのではなく、ユーザー資格証明を指定できます。また、キャッシュ SHA-2 認証を使用している場合は、グループメンバーに RSA キーペアを設定する必要があります。

キャッシュ SHA-2 認証プラグインを使用するレプリケーションユーザー

デフォルトでは、MySQL 8 で作成されたユーザーは [セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#) を使用します。分散リカバリ用に構成するレプリケーションユーザーがキャッシュ SHA-2 認証プラグインを使用し、分散リカバリ接続に SSL を使用しない場合、RSA キーペアがパスワード交換に使用されます。RSA キーペアの詳細は、[セクション6.3.3「SSL および RSA 証明書とキーの作成」](#)を参照してください。

この状況では、`rpl_user` の公開キーを結合メンバーにコピーするか、リクエスト時に公開キーを提供するようにドナーを構成できます。よりセキュアなアプローチは、レプリケーションユーザーアカウントの公開キーを参加メンバーにコピーすることです。次に、レプリケーションユーザーアカウントの公開キーへのパスを使用して、結合メンバーで `group_replication_recovery_public_key_path` システム変数を構成する必要があります。

安全性の低いアプローチは、メンバーに参加するためにレプリケーションユーザーアカウントの公開キーを提供するようにドナーに `group_replication_recovery_get_public_key=ON` を設定することです。サーバーのアイデンティティを検証する方法はないため、中間者攻撃などによってサーバーアイデンティティが損なわれるリスクがないことが確実な場合にのみ、`group_replication_recovery_get_public_key=ON` を設定してください。

SSL を使用したレプリケーションユーザー

グループに参加するサーバー (参加メンバー) がドナーに接続する前に、SSL 接続を必要とするレプリケーションユーザーを作成する必要があります。通常、これはサーバーをプロビジョニングしてグループに参加するときに設定されます。SSL 接続を必要とする分散リカバリのレプリケーションユーザーを作成するには、グループに参加するすべてのサーバーで次のステートメントを発行します:

```
mysql> SET SQL_LOG_BIN=0;
mysql> CREATE USER 'rec_ssl_user'@'%' IDENTIFIED BY 'password' REQUIRE SSL;
mysql> GRANT replication slave ON *.* TO 'rec_ssl_user'@'%';
mysql> GRANT BACKUP_ADMIN ON *.* TO 'rec_ssl_user'@'%';
mysql> FLUSH PRIVILEGES;
mysql> SET SQL_LOG_BIN=1;
```

レプリケーションユーザー資格証明のセキュアな提供

レプリケーションユーザーのユーザー資格証明を指定するには、[CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO](#) ステートメントを使用して、`group_replication_recovery` チャンネルの資格証明として永続的に設定します。または、MySQL 8.0.21 から、グループレプリケーションが開始されるたびに `START GROUP_REPLICATION` ステートメントで指定できます。`START GROUP_REPLICATION` で指定されたユーザー資格証明は、[CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO](#) ステートメントを使用して設定されたユーザー資格証明よりも優先されます。

[CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO](#) を使用して設定されたユーザー資格証明は、サーバー上のレプリケーションメタデータリポジトリにプレーンテキストで格納されますが、`START GROUP_REPLICATION` で指定されたユーザー資格証明はメモリーにのみ保存され、`STOP GROUP_REPLICATION` ステートメントまたはサーバーの停止によって削除されます。したがって、`START GROUP_REPLICATION` を使用してユーザー資格証明を指定すると、不正なアクセスからグループレプリケーションサーバーを保護するのに役立ちます。ただし、この方法は、`group_replication_start_on_boot` システム変数で指定された Group Replication の自動起動とは互換性がありません。

[CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO](#) ステートメントを使用してユーザー資格証明を永続的に設定する場合は、グループに参加するメンバーに対して次のステートメントを発行します:

```
mysql> CHANGE MASTER TO MASTER_USER='rec_ssl_user', MASTER_PASSWORD='password'  
FOR CHANNEL 'group_replication_recovery';
```

Or from MySQL 8.0.23:

```
mysql> CHANGE REPLICATION SOURCE TO SOURCE_USER='rec_ssl_user', SOURCE_PASSWORD='password'  
FOR CHANNEL 'group_replication_recovery';
```

`START GROUP_REPLICATION` でユーザー資格証明を指定するには、グループレプリケーションの初回起動時またはサーバーの再起動後に次のステートメントを発行します:

```
mysql> START GROUP_REPLICATION USER='rec_ssl_user', PASSWORD='password';
```

重要

`START GROUP_REPLICATION` を使用して、以前に `CHANGE REPLICATION SOURCE TO` | `CHANGE MASTER TO` を使用して資格証明を提供したサーバー上のユーザー資格証明を指定するには、次のステップを実行して、この変更のセキュリティ上の利点を得る必要があります。

1. `STOP GROUP_REPLICATION` ステートメントを使用して、グループメンバーのグループレプリケーションを停止します。グループレプリケーションの実行中に次の2つのステップを実行できますが、変更を実装するにはグループレプリケーションを再起動する必要があります。
2. `group_replication_start_on_boot` システム変数の値を `OFF` (デフォルトは `ON`) に設定します。
3. 次のステートメントを発行して、レプリカステータステーブルから分散リカバリ資格証明を削除します:

```
mysql> CHANGE MASTER TO MASTER_USER="", MASTER_PASSWORD="  
FOR CHANNEL 'group_replication_recovery';
```

Or from MySQL 8.0.23:

```
mysql> CHANGE REPLICATION SOURCE TO SOURCE_USER="", SOURCE_PASSWORD="  
FOR CHANNEL 'group_replication_recovery';
```

4. 分散リカバリユーザー資格証明を指定する `START GROUP_REPLICATION` ステートメントを使用して、グループメンバーでグループレプリケーションを再起動します。

これらのステップを実行しないと、資格証明はレプリカステータステーブルに格納されたままになり、分散リカバリのリモートクローニング操作中に他のグループメンバーに転送することもできます。その後、元のメンバーまたはそこからクローニングされたメンバーのいずれかで、`group_replication_recovery` チャンネルが誤って格納された資格証明で起動される可能性があります。サーバー起動時のグループレプリケーションの自動開始 (リモートクローニング操作後を含む) では、格納されたユーザー資格証明が使用され、オペレータが `START GROUP_REPLICATION` コマンドで分散リカバリ資格証明を指定しなかった場合にも使用されます。

18.5.3.2 分散リカバリのための Secure Socket Layer (SSL) 接続

標準 SQL クライアント接続と分散リカバリエンドポイントのどちらを使用して分散リカバリ接続を確立する場合でも、セキュアに接続を構成するために Group Replication 専用の分散リカバリ SSL オプションを使用できます。これらのオプションは、グループ通信接続に使用されるサーバー SSL オプションに対応していますが、分散リカバリ接続にのみ適用されます。デフォルトでは、グループ通信接続に対して SSL をアクティブ化した場合でも、分散リカバリ接続では SSL は使用されず、分散リカバリ接続にはサーバー SSL オプションは適用されません。これらの接続は個別に構成する必要があります。

リモートクローニング操作が分散リカバリの一部として使用される場合、Group Replication は、分散リカバリ SSL オプションの設定と一致するようにクローンプラグイン SSL オプションを自動的に構成します。(クローンプラグインが SSL を使用する方法の詳細は、[クローニング用の暗号化された接続の構成](#) を参照してください。)

分散リカバリの SSL オプションは次のとおりです:

- `group_replication_recovery_use_ssl`: グループレプリケーションが分散リカバリ接続に SSL を使用するようにするには、`ON` に設定します。これには、リモートクローニング操作やドナーのバイナリログからの状態転送が含まれます。このオプションを設定するだけで、他の分散リカバリ SSL オプションは設定できません。この場合、サーバーは接続に使用する証明書を自動的に生成し、デフォルトの暗号スイートを使用します。接続用の証明書および暗号スイートを構成する場合は、他の分散リカバリ SSL オプションを使用して構成します。

- `group_replication_recovery_ssl_ca`: 分散リカバリ接続に使用する認証局 (CA) ファイルのパス名。グループレプリケーションでは、クローン SSL オプション `clone_ssl_ca` がこれに一致するように自動的に構成されます。
`group_replication_recovery_ssl_capath`: 信頼できる SSL 認証局 (CA) 証明書ファイルを含むディレクトリのパス名。
- `group_replication_recovery_ssl_cert`: 分散リカバリ接続に使用する SSL 公開キー証明書ファイルのパス名。グループレプリケーションでは、クローン SSL オプション `clone_ssl_cert` がこれに一致するように自動的に構成されます。
- `group_replication_recovery_ssl_key`: 分散リカバリ接続に使用する SSL 秘密キーファイルのパス名。グループレプリケーションでは、クローン SSL オプション `clone_ssl_cert` がこれに一致するように自動的に構成されます。
- `group_replication_recovery_ssl_verify_server_cert`: 分散リカバリ接続で、ドナーが送信した証明書のサーバーの共通名の値をチェックします。このオプションを `ON` に設定することは、グループ通信接続の `group_replication_ssl_mode` オプションに `VERIFY_IDENTITY` を設定する分散リカバリ接続の場合と同じです。
- `group_replication_recovery_ssl_crl`: 証明書失効リストを含むファイルのパス名。
- `group_replication_recovery_ssl_crlpath`: 証明書失効リストを含むディレクトリのパス名。
- `group_replication_recovery_ssl_cipher`: 分散リカバリ接続の接続暗号化に許可される暗号のリスト。コロンで区切られた 1 つ以上の暗号名のリストを指定します。MySQL がサポートする暗号化暗号の詳細は、[セクション 6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#) を参照してください。
- `group_replication_recovery_tls_version`: このサーバーインスタンスが分散リカバリ接続のクライアント (参加メンバー) である場合に、接続暗号化に許可される 1 つ以上の TLS プロトコルのカンマ区切りリスト。指定したバージョンが連続していることを確認します (たとえば、「`TLSv1,TLSv1.1,TLSv1.2`」)。このシステム変数が設定されていない場合、デフォルトの「`TLSv1,TLSv1.1,TLSv1.2,TLSv1.3`」が使用されます。クライアント (参加メンバー) およびサーバー (ドナー) としての各分散リカバリ接続に含まれるグループメンバーは、どちらもサポートするように設定されている最高のプロトコルバージョンをネゴシエートします。このシステム変数は、MySQL 8.0.19 から使用できます。
- `group_replication_recovery_tls_ciphersuites`: 分散リカバリ接続の接続暗号化に TLSv1.3 が使用され、このサーバーインスタンスが分散リカバリ接続のクライアント (参加メンバー) である場合に許可される暗号スイートのコロン区切りリスト。TLSv1.3 の使用時にこのシステム変数が `NULL` に設定されている場合 (システム変数を設定しない場合のデフォルト)、[セクション 6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#) にリストされているように、デフォルトで有効になっている暗号スイートが許可されます。このシステム変数が空の文字列に設定されている場合、暗号スイートは許可されないため、TLSv1.3 は使用されません。このシステム変数は、MySQL 8.0.19 以降で使用できます。

18.6 グループレプリケーションのパフォーマンス

このセクションでは、使用可能な構成オプションを使用してレプリケーショングループから最高のパフォーマンスを得る方法について説明します。

18.6.1 グループ通信スレッドの微調整

Group Replication プラグインがロードされている間、グループ通信スレッド (GCT) はループで実行されます。GCT は、グループおよびプラグインからメッセージを受信し、定足数および障害検出関連タスクを処理し、キープアラートメッセージをいくつか送信し、サーバー/グループとの送受信トランザクションも処理します。GCT は、キュー内の受信メッセージを待機します。メッセージがない場合、GCT は待機します。この待機を (アクティブな待機を行う) 少し長く構成することで、実際にスリープになる前に、場合によっては有益であることが証明されます。これは、オペレーティングシステムがプロセスから GCT を切り替えてコンテキストスイッチを実行するためです。

GCT で強制的にアクティブな待機を実行するには、`group_replication_poll_spin_loops` オプションを使用します。これにより、GCT ループは、次のメッセージのキューを実際にポーリングする前に、構成されたループ数に関係なく何も実行されません。

例:

```
mysql> SET GLOBAL group_replication_poll_spin_loops= 1000;
```

18.6.2 フロー制御

グループレプリケーションでは、グループ内の大部分のメンバーがトランザクションを受信し、同時に送信されたすべてのトランザクション間の相対的な順序で合意した後にのみ、トランザクションがコミットされます。この方法は、グループへの書き込みの合計数がグループ内のどのメンバーの書き込み容量も超えない場合に適しています。書き込みスループットが他のメンバーよりも低い場合 (特にライターメンバーよりも低い場合)、これらのメンバーはライターの遅延を開始できます。

一部のメンバーがグループより遅れていると、問題のある結果が発生します。特に、このようなメンバーに対する読取りによって、非常に古いデータが外部化される可能性があります。メンバーが遅れている理由によっては、低速メンバーからの潜在的なデータ転送リクエストを満たすために、グループ内の他のメンバーがより多くのレプリケーションコンテキストを保存する必要がある場合があります。

ただし、レプリケーションプロトコルには、高速メンバーと低速メンバーの間に適用されるトランザクションに関して距離が多すぎることを回避するメカニズムがあります。これはフロー制御メカニズムと呼ばれます。いくつかの目標に対処しようとしています:

1. メンバー間でバッファリングおよび同期解除を行うのに十分な大きさにメンバーを近づけることは、小さい問題です
2. グループ内の異なるワークロードやより多くのライターなどの条件の変化に迅速に適応できます
3. 各メンバーに使用可能な書き込み容量のフェアシェアを付与
4. リソースの無駄を避けるために必要以上のスループットを削減しません。

グループレプリケーションの設計では、2つの作業キューを考慮してスロットルするかどうかを決定できます: 動作保証キュー、(ii) およびバイナリログ applicer キューで (i) を実行します。これらのキューのいずれかのサイズがユーザー定義のしきい値を超えると、スロットルメカニズムがトリガーされます。構成のみ: (i) では、認証者レベルまたはアプライヤレベル (あるいはその両方) でフロー制御を実行するかどうか、および (ii) では各キューのしきい値は何ですか。

フロー制御は、次の2つの基本メカニズムに依存します:

1. すべてのグループメンバーのスループットおよびキューサイズに関するいくつかの統計を収集するためのメンバーの監視。各メンバーの最大書き込み圧力を測定
2. 各時点で使用可能な容量の公平分配を超えて書き込もうとしているメンバーのスロットル。

18.6.2.1 プローブと統計

監視メカニズムは、各メンバーが一連のプローブをデプロイして、そのワークキューおよびスループットに関する情報を収集するようにすることで機能します。次に、その情報を定期的にグループに伝播して、そのデータを他のメンバーと共有します。

このようなプローブはプラグインスタック全体に分散され、次のようなメトリックを確立できます:

- 証明者のキューサイズ
- レプリケーションアプライアンスのキューサイズ
- 動作保証されたトランザクションの合計数
- メンバーに適用されたリモートトランザクションの合計数
- ローカルトランザクションの総数。

メンバーは、別のメンバーからの統計を含むメッセージを受信すると、最後の監視期間中に動作保証、適用およびローカルに実行されたトランザクションの数に関する追加メトリックを計算します。

監視データは、グループ内の他のユーザーと定期的に共有されます。監視期間は、他のメンバーが現在の書き込みリクエストを決定できるように十分に長くする必要がありますが、グループ帯域幅への影響を最小限に抑えるには十分です。情報は毎秒共有され、この期間で両方の懸案に対処できます。

18.6.2.2 グループレプリケーションスロットル

グループ内のすべてのサーバーにわたって収集されたメトリックに基づいて、スロットルメカニズムが開始され、メンバーが新しいトランザクションを実行/コミットできるレートを制限するかどうかが決まります。

したがって、すべてのメンバーから取得されたメトリックは、各メンバーの容量を計算するための基準となります: メンバーに大規模なキュー (証明またはアプライヤスレッド用) がある場合、新しいトランザクションを実行するための容量は、最後の期間に動作保証または適用された容量に近い必要があります。

グループ内のすべてのメンバーの最小容量によってグループの実際の容量が決まりますが、ローカルトランザクションの数によって書き込み中のメンバーの数が決まり、その結果、使用可能な容量を共有するメンバーの数が決まります。

これは、すべてのメンバーが使用可能な容量に基づいて書き込み割当て制限を確立していることを意味します。つまり、次の期間に安全に発行できるトランザクションの数が多くなります。ライター割当て制限は、証明者またはバイナリログアプライアンスのキューサイズがユーザー定義のしきい値を超えると、スロットルメカニズムによって強制されます。

割当て制限は、前の期間に遅延したトランザクションの数だけ削減され、さらに 10% 削減されて、問題をトリガーしたキューのサイズを縮小できるようになります。キューサイズがしきい値を超えるとスループットのジャンプが大きくなるようにするために、スループットはその後の期間ごとに同じ 10% 増加することのみが許可されます。

現在のスロットルメカニズムでは、割当て制限を下回るトランザクションはペナル化されませんが、監視期間が終了するまで、それを越えるトランザクションの終了は遅延されます。その結果、発行された書き込みリクエストの割当て制限が非常に小さい場合、一部のトランザクションで待機時間が監視期間に近い可能性があります。

18.6.3 メッセージ圧縮

オンライングループメンバー間で送信されるメッセージの場合、グループレプリケーションはデフォルトでメッセージ圧縮を有効にします。特定のメッセージが圧縮されるかどうかは、`group_replication_compression_threshold` システム変数を使用して構成するしきい値によって決まります。ペイロードが指定されたバイト数より大きいメッセージは圧縮されます。

デフォルトの圧縮しきい値は 1000000 バイトです。たとえば、次のステートメントを使用して、圧縮しきい値を 2MB に増やすことができます:

```
STOP GROUP_REPLICATION;  
SET GLOBAL group_replication_compression_threshold = 2097152;  
START GROUP_REPLICATION;
```

`group_replication_compression_threshold` をゼロに設定すると、メッセージ圧縮は無効になります。

グループレプリケーションでは、LZ4 圧縮アルゴリズムを使用して、グループで送信されるメッセージを圧縮します。LZ4 圧縮アルゴリズムでサポートされる最大入力サイズは 2113929216 バイトです。この制限は、XCom で受け入れられる最大メッセージサイズと一致する、`group_replication_compression_threshold` システム変数の最大可能値を下回っています。したがって、LZ4 の最大入力サイズはメッセージ圧縮の実用的な制限であり、メッセージ圧縮が有効な場合、このサイズを超えるトランザクションはコミットできません。LZ4 圧縮アルゴリズムでは、`group_replication_compression_threshold` に 2113929216 バイトを超える値を設定しないでください。

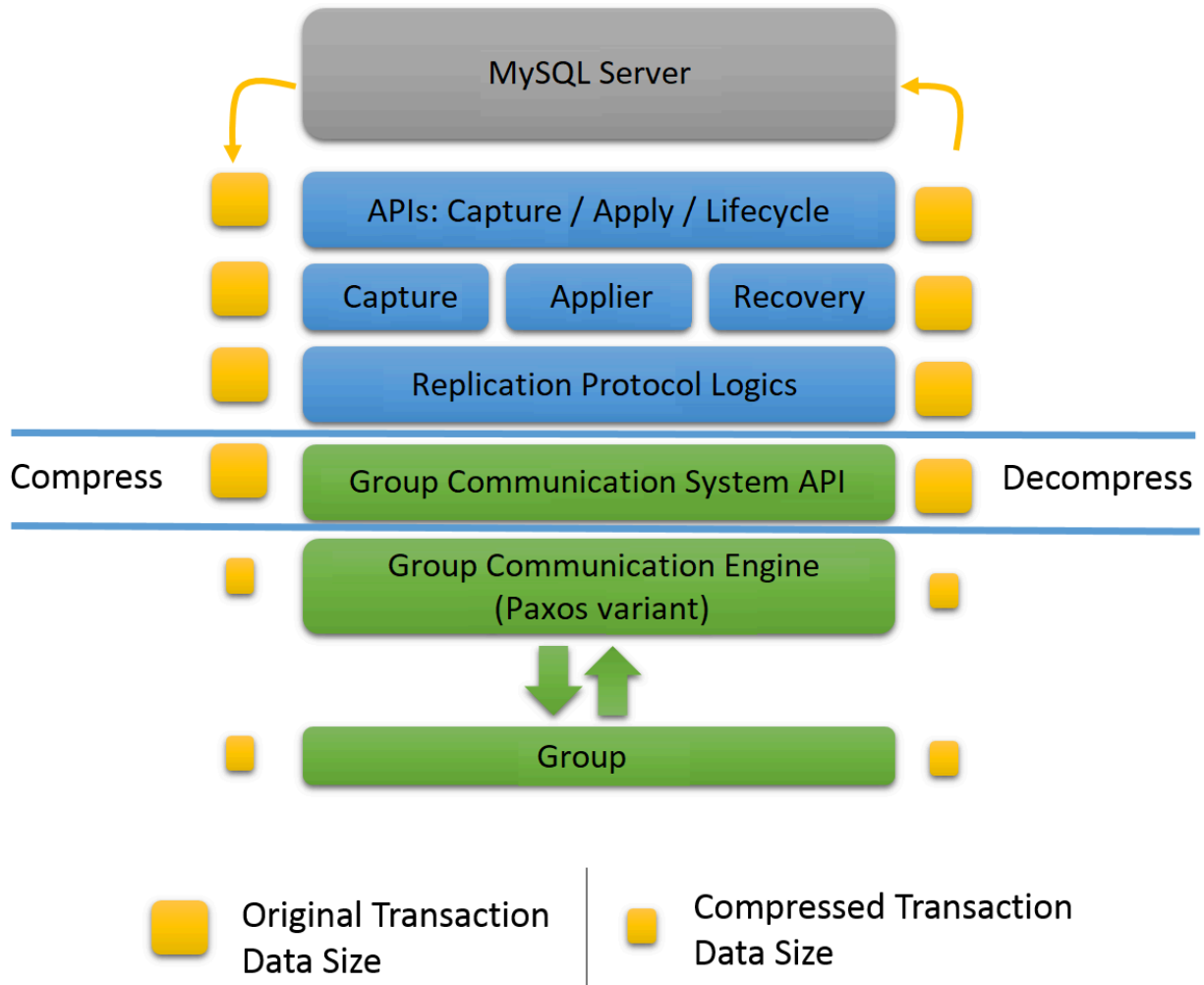
グループレプリケーションでは、`group_replication_compression_threshold` の値はすべてのグループメンバーで同じである必要はありません。ただし、トランザクションの不要なロールバック、メッセージ配信の失敗またはメッセージリカバリの失敗を回避するために、すべてのグループメンバーに同じ値を設定することをお勧めします。

MySQL 8.0.18 から、ドナーバイナリログからの状態転送によって分散リカバリ用に送信されるメッセージの圧縮を構成することもできます。グループ内にすでに存在するドナーから参加メンバーに送信されるこれらのメッセージの圧縮は、`group_replication_recovery_compression_algorithm` および `group_replication_recovery_zstd_compression_level` システム変数を使用して個別に制御されます。詳細は、[セクション4.2.8「接続圧縮制御」](#)を参照してください。

`binlog_transaction_compression` システム変数によってアクティブ化されるバイナリログトランザクション圧縮 (MySQL 8.0.20 で使用可能) を使用して帯域幅を節約することもできます。トランザクションペイロードは、グループメンバー間で転送されても圧縮されません。バイナリログトランザクションの圧縮を Group Replication メッセージの圧縮と組み合わせて使用する場合、メッセージの圧縮ではデータを処理する機会は少なくなりますが、ヘッダーと、圧縮されていないイベントおよびトランザクションペイロードは圧縮できます。バイナリログのトランザクション圧縮の詳細は、[セクション5.4.4.5「バイナリログトランザクション圧縮」](#)を参照してください。

グループ内で送信されるメッセージの圧縮は、データがグループ通信スレッドに渡される前に、グループ通信エンジンレベルで行われるため、mysql ユーザーセッションスレッドのコンテキスト内で行われます。メッセージペイロードサイズが `group_replication_compression_threshold` で設定されたしきい値を超えると、トランザクションペイロードはグループに送信される前に圧縮され、受信時に解凍されます。メッセージを受信すると、メンバーはメッセージエンベロープをチェックして、圧縮されているかどうかを確認します。必要に応じて、メンバーは上位レイヤーに配信する前にトランザクションを解凍します。このプロセスを次の図に示します。

図 18.15 圧縮のサポート



ネットワーク帯域幅がボトルネックの場合、メッセージ圧縮により、グループ通信レベルでスループットが最大 30 から 40% 向上します。これは、負荷の高い大規模なサーバーグループのコンテキスト内で特に重要です。グループ内の N 参加者間の相互接続の TCP ピアツーピアの性質により、送信者は同じ量のデータを N 回送信します。さらに、バイナリログは高い圧縮率を示している可能性があります。これにより、圧縮は、大規模なトランザクションを含むグループレプリケーションワークロードにとって魅力的な機能になります。

18.6.4 メッセージの断片化

グループレプリケーショングループメンバー間で異常に大きいメッセージが送信されると、一部のグループメンバーが失敗としてレポートされ、グループから除外される可能性があります。これは、Group Replication グループ通信エンジン (XCom、Paxos バリエーション) で使用されるシングルスレッドがメッセージの処理に時間がかかりすぎるため、一部のグループメンバーが受信者を失敗として報告する可能性があるためです。デフォルトでは、MySQL 8.0.16 から、大きいメッセージは自動的にフラグメントに分割され、受信者によって個別に送信されて再アセンブルされます。

システム変数 `group_replication_communication_max_message_size` は、グループレプリケーション通信の最大メッセージサイズを指定します。このサイズを超えると、メッセージは断片化されます。デフォルトの最大メッセージサイズは 10485760 バイト (10 MiB) です。最大許容値は、`slave_max_allowed_packet` システム変数の最大値 1073741824 バイト (1 GB) と同じです。アプライヤスレッドは `slave_max_allowed_packet` より大きいメッセージフラグメントを処理できないため、`group_replication_communication_max_message_size` の設定は `slave_max_allowed_packet` の設定より小さくする必要があります。断片化をオフにするには、`group_replication_communication_max_message_size` にゼロ値を指定します。

他のほとんどの Group Replication システム変数と同様に、変更を有効にするには Group Replication プラグインを再起動する必要があります。例:

```
STOP GROUP_REPLICATION;  
SET GLOBAL group_replication_communication_max_message_size= 5242880;  
START GROUP_REPLICATION;
```

断片化されたメッセージのメッセージ配信は、メッセージのすべてのフラグメントが受信され、すべてのグループメンバーによって再アセンブルされると完了とみなされます。断片化されたメッセージには、ヘッダー内の情報が含まれます。この情報を使用すると、メッセージ転送中にメンバーを結合して、参加前に送信された以前のフラグメントをリカバリできます。結合メンバーがフラグメントのリカバリに失敗すると、グループから削除されます。

レプリケーショングループで断片化を使用するには、すべてのグループメンバーが MySQL 8.0.16 以上であり、グループで使用されている Group Replication 通信プロトコルバージョンで断片化が許可されている必要があります。グループがサポートしている最も古い MySQL Server バージョンを返す `group_replication_get_communication_protocol()` UDF を使用して、グループが使用している通信プロトコルを検査できます。MySQL 5.7.14 のバージョンではメッセージを圧縮でき、MySQL 8.0.16 のバージョンではメッセージを断片化することもできます。すべてのグループメンバーが MySQL 8.0.16 以上で、以前のリリースのメンバーの参加を許可する必要がない場合は、`group_replication_set_communication_protocol()` UDF を使用して通信プロトコルバージョンを MySQL 8.0.16 以上に設定し、断片化を許可できます。詳細は、[セクション 18.4.1.4 「グループ通信プロトコルバージョンの設定」](#) を参照してください。

一部のメンバーが断片化をサポートしていないためにレプリケーショングループが断片化を使用できない場合は、システム変数 `group_replication_transaction_size_limit` を使用して、グループが受け入れるトランザクションの最大サイズを制限できます。MySQL 8.0 では、デフォルト設定は約 143 MB です。このサイズを超えるトランザクションはロールバックされます。システム変数 `group_replication_member_expel_timeout` を使用して、失敗した疑いがあるメンバーがグループから削除されるまでの追加時間 (最大 1 時間) を許可することもできます。

18.6.5 XCom キャッシュ管理

グループレプリケーションのグループ通信エンジン (XCom、Paxos バリエーション) には、コンセンサスプロトコルの一部としてグループメンバー間で交換されるメッセージ (およびそのメタデータ) のキャッシュが含まれます。メッセージキャッシュは、他のグループメンバーと通信できなかった期間後にグループに再接続するメンバーによる、欠落したメッセージのリカバリに使用されます。

MySQL 8.0.16 から、`group_replication_message_cache_size` システム変数を使用して XCom メッセージキャッシュのキャッシュサイズ制限を設定できます。キャッシュサイズ制限に達すると、XCom は決定され配信された最も古いエントリを削除します。再接続しようとしている到達不能なメンバーは、欠落したメッセージをリカバリするために他のメンバーをランダムに選択するため、すべてのグループメンバーに同じキャッシュサイズ制限を設定する必要があります。したがって、各メンバーキャッシュで同じメッセージを使用できる必要があります。

MySQL 8.0.16 より前では、キャッシュサイズは 1 GB で、MySQL 8.0.16 のキャッシュサイズのデフォルト設定は同じです。MySQL Server の他のキャッシュおよびオブジェクトプールのサイズを考慮して、選択したキャッシュサイズ制限に十分なメモリーがシステムで使用可能であることを確認します。`group_replication_message_cache_size` を使用して設定された制限はキャッシュに格納されているデータにのみ適用され、キャッシュ構造には追加の 50 MB のメモリーが必要です。

`group_replication_message_cache_size` 設定を選択する場合は、メンバーが削除されるまでの期間内の予想されるメッセージ量を参照してください。この期間の長さは、メンバーが明示されるのではなくグループに戻るための最初の 5 秒間の検出期間に加えて許可される待機期間 (最大時間) を決定する `group_replication_member_expel_timeout` システム変数によって制御されます。MySQL 8.0.21 より前は、`group_replication_member_expel_timeout` システム変数によって設定された追加の明示タイムアウトがデフォルトでゼロに設定されているため、この期間はメンバーが使用できなくなるまでの 5 秒にデフォルト設定されることに注意してください。これは疑わしいが作成されるまでの検出期間です。8.0.21 からは、`expel` タイムアウトはデフォルトで 5 秒に設定されるため、メンバーは少なくとも 10 秒間存在しなくなるまで削除されません。

18.6.5.1 キャッシュサイズの増加

メンバーがグループから削除されるのに十分な長さでない期間存在しない場合、別のメンバーの XCom メッセージ キャッシュから欠落したトランザクションを取得することで、再接続してグループへの参加を再開できます。ただし、メンバーの休暇欠勤中に発生したトランザクションが、最大サイズ制限に達したために他のメンバーの XCom メッセージキャッシュから削除された場合、メンバーはこの方法で再接続できません。

Group Replication Group Communication System (GCS) は、現在アクセスできないメンバーによるリカバリに必要なと思われるメッセージがメッセージキャッシュから削除されたときに、警告メッセージでアラートを生成します。この警告メッセージは、すべてのアクティブなグループメンバーに記録されます (アクセスできないメンバーごとに一度のみ)。グループメンバーは、アクセスできないメンバーに最後に表示されたメッセージを確認できませんが、警告メッセージには、メンバーが削除される前に、選択した待機期間をサポートするのに十分なキャッシュサイズがない可能性があることが示されます。

この状況では、メンバーが正常に戻るために必要なすべての欠落メッセージがキャッシュに含まれるように、`group_replication_member_expel_timeout` システム変数で指定された期間内の予想されるメッセージ量に 5 秒の検出期間を加えて、`group_replication_message_cache_size` の制限を増やすことを検討してください。メンバーが異常な期間使用不可になると予想される場合は、キャッシュサイズ制限を一時的に増やすことも検討できます。

18.6.5.2 キャッシュサイズの削減

XCom メッセージキャッシュサイズの最小設定は、1 GB から MySQL 8.0.20 までです。MySQL 8.0.21 からは、最小設定は 134217728 バイト (128 MB) で、使用可能なメモリー量が制限されたホストへのデプロイメントを可能にします。ホストが不安定なネットワーク上にある場合、`group_replication_message_cache_size` 設定を非常に低くすることはお薦めしません。これは、メッセージキャッシュが小さいほど、接続が一時的に失われた後にグループメンバーが再接続するのが困難になるためです。

再接続するメンバーが XCom メッセージキャッシュから必要なすべてのメッセージを取得できない場合、分散リカバリを使用して別のメンバーバイナリログから欠落しているトランザクションを取得するには、メンバーはグループから移動して再結合する必要があります。MySQL 8.0.21 から、グループを離れたメンバーはデフォルトで 3 回の自動再結合を試行するため、オペレータの介入なしでもグループの再結合プロセスを実行できます。ただし、分散リカバリを使用した再結合は、XCom メッセージキャッシュからメッセージを取得するよりも大幅に長く複雑なプロセスであるため、メンバーが使用可能になり、グループのパフォーマンスが影響を受ける可能性があります。安定したネットワーク (メンバーの接続の一時的な損失の頻度と期間を最小限に抑える) では、この発生の頻度も最小限に抑える必要があるため、グループはパフォーマンスに大きな影響を与えずに、より小さい XCom メッセージキャッシュサイズを許容できる可能性があります。

キャッシュサイズ制限を減らすことを検討している場合は、次のステートメントを使用して「パフォーマンススキーマ」テーブル `memory_summary_global_by_event_name` をクエリーすることができます:

```
SELECT * FROM performance_schema.memory_summary_global_by_event_name
WHERE EVENT_NAME LIKE 'memory/group_rpl/GCS_XCom::xcom_cache';
```

これは、キャッシュされたエントリの現在の数やキャッシュの現在のサイズなど、メッセージキャッシュのメモリー使用量の統計を返します。キャッシュサイズ制限を小さくすると、XCom では、現在のサイズが制限を下回るまで、決定および配信された最も古いエントリが削除されます。この削除プロセスの進行中に、XCom が一時的にキャッシュサイズ制限を超えている可能性があります。

18.6.6 障害検出およびネットワークパーティション化へのレスポンス

グループレプリケーション障害検出メカニズムは、グループと通信しなくなったグループメンバーを識別し、障害が発生した可能性があると思われる場合に明示するように設計されています。障害検出メカニズムを使用すると、グループに正しく機能するメンバーの大部分が含まれ、そのためクライアントからのリクエストが正しく処理される可能性が高くなります。

通常、すべてのグループメンバーは、他のすべてのグループメンバーと定期的にメッセージを交換します。グループメンバーが特定のフェローメンバーからのメッセージを 5 秒間受信しない場合、この検出期間が終了すると、フェローメンバーの疑いが生じます。疑わしいメンバーがタイムアウトすると、疑わしいメンバーは失敗したとみなされ、グループから削除されます。削除されたメンバーは、他のメンバーに表示されるメンバーシップリストから削除されますが、グループから削除されたことは認識されないため、オンラインとして表示され、他のメンバーは使用不可として表示されます。メンバーが実際に失敗しておらず (たとえば、一時的なネットワークの問題のために切断されたばかりのため)、他のメンバーとの通信を再開できる場合は、グループから削除された情報を含むビューを受信します。

これらの状況に対するグループメンバーのレスポンス (失敗したメンバー自体を含む) は、プロセスのいくつかのポイントで構成できます。デフォルトでは、メンバーに障害が発生した疑いがある場合、次の動作が発生します:

1. MySQL 8.0.20 までは、疑いが作成されるとすぐにタイムアウトします。疑わしいメンバーは、失効した疑いがグループによって識別されるとすぐに削除の責任を負います。期限切れの疑いのチェックは定期的に行われるため、メンバーはタイムアウト後数秒間存続する可能性があります。MySQL 8.0.21 からは、疑わしいメンバーが削除の責任を負うまで、5 秒の待機期間が追加されます。
2. 削除されたメンバーが通信を再開し、MySQL 8.0.20 まで削除されたことを認識した場合、グループへの再参加は試行されません。MySQL 8.0.21 からは、グループへの再参加が 3 回自動的に試行され (試行ごとに 5 分)、この自動再結合プロセスが機能しない場合は、グループへの再参加の試行が停止されます。
3. 削除されたメンバーがグループに再度参加しようとしていない場合、スーパー読み取り専用モードに切り替わり、オペレータの注意を待ちます。(ただし、MySQL 8.0.12 から 8.0.15 へのリリースでは例外です。デフォルトでは、メンバー自体がシャットダウンされます。MySQL 8.0.16 から、MySQL 5.7 の動作と一致するように動作が変更されました。)

このセクションで説明する Group Replication 構成オプションを使用すると、システム要件および優先順位に合わせて、これらの動作を永続的または一時的に変更できます。低速なネットワークまたはマシン、予期しない一時的な停止率の高いネットワーク、または計画的なネットワーク停止が原因で不要な削除が発生した場合は、`expel timeout` および `auto-rejoin` の試行を増やすことを検討してください。MySQL 8.0.21 からは、この方向にデフォルト設定が変更され、これらの状況でメンバーを解放するためにオペレータの介入が必要になる頻度が削減されました。メンバーは前述のデフォルトの動作のいずれかを実行していますが、書き込みを受け入れませんが、メンバーがクライアントと通信している場合は、時間の経過とともに失効した読み取りの可能性が高くなり、読み取りを行うことができます。失効した読み取りの回避は、演算子の介入を回避するよりも優先度が高い場合は、`expel timeout` および `auto-rejoin` の試行を減らすか、ゼロに設定することを検討してください。

障害が発生していないメンバーは、ネットワークパーティションのためにレプリケーショングループの一部 (すべてではない) との接続を失う可能性があります。たとえば、5 つのサーバー (S1、S2、S3、S4、S5) のグループでは、(S1、S2) と (S3、S4、S5) の間に切断がある場合、ネットワークパーティションがあります。最初のグループ (S1、S2) は、半分を超えるグループにコンタクトできないため、少数民族になりました。少数民族グループのメンバーによって処理されるトランザクションはブロックされます。これは、グループの大部分にアクセスできないため、グループはクォーラムを達成できないためです。このシナリオの詳細は、[セクション 18.4.4 「ネットワークパーティション化」](#) を参照してください。この場合、デフォルトの動作では、少数民族と大多数のメンバーがグループに残り、(少数民族のメンバーでブロックされていても) トランザクションの受入れを続行し、オペレータの介入を待機します。この動作も構成可能です。

関連する設定をサポートしていない古い MySQL Server リリースのグループメンバー、または異なるデフォルトのリリースのグループメンバーは、前述のデフォルトの動作に従って自身および他のグループメンバーに作用します。たとえば、`group_replication_member_expel_timeout` システム変数をサポートしていないメンバーは、期限切れの疑いが検出されるとすぐに他のメンバーを削除し、システム変数をサポートしており、タイムアウトが長く設定されている場合でも、この削除は他のメンバーによって受け入れられます。

18.6.6.1 Expel タイムアウト

MySQL 8.0.13 から使用可能な `group_replication_member_expel_timeout` システム変数を使用すると、疑わしいメンバーの作成と削除の間に時間を追加できます。疑わしいのは、[セクション 18.1.4.2 「障害検出」](#) で説明されているように、あるサーバーが別のサーバーからメッセージを受信しない場合です。

グループレプリケーショングループメンバーが別のメンバー (またはそれ自体) の疑いを作成する前に、最初の 5 秒の検出期間があります。グループメンバーは、その別のメンバーの疑い (またはそれ自体の疑い) がタイムアウトしたときに削除されます。それよりも短い時間が経過すると、削除メカニズムが削除を検出して実装する前に経過する可能性があります。`group_replication_member_expel_timeout` では、`expel timeout` と呼ばれる、疑わしいメンバーの作成と疑わしいメンバーの削除の間にグループメンバーが待機する時間を秒単位で指定します。サブセクトメンバーは、この待機期間中は `UNREACHABLE` としてリストされますが、グループメンバーシップリストからは削除されません。

- 疑わしいメンバーが待機期間の終了時にタイムアウトする前に再度アクティブになった場合、メンバーは XCom メッセージキャッシュ内の残りのグループメンバーによってバッファされたすべてのメッセージを適用し、オペレータの介入なしで `ONLINE` 状態になります。この状況では、メンバーは同じインカネーションとしてグループによって考慮されます。

- 疑わしいメンバーがアクティブになるのは、疑わしいメンバーがタイムアウトして通信を再開できる場合のみで、削除されたビューを受信し、その時点で削除されたことを認識します。MySQL 8.0.16 から使用可能な [group_replication_autorejoin_tries](#) システム変数を使用して、この時点でメンバーがグループへの再参加を自動的に試行するようにできます。MySQL 8.0.21 からは、この機能はデフォルトでアクティブ化され、メンバーは 3 回の自動再結合を試行します。自動再結合プロシージャが成功しなかった場合、または試行されなかった場合、削除されたメンバーは [group_replication_exit_state_action](#) で指定された終了アクションに従います。

メンバーを削除するまでの待機期間は、以前にグループ内でアクティブになっていたメンバーにのみ適用されます。グループ内でアクティブになっていなかった非メンバーは、この待機期間を取得せず、最初の検出期間の後に削除されます。これは、参加に時間がかかりすぎるためです。

[group_replication_member_expel_timeout](#) が 0 に設定されている場合、待機期間はなく、疑わしいメンバーは 5 秒間の検出期間が終了した直後に削除する責任があります。この設定は、MySQL 8.0.20 までのデフォルトです。これは、[group_replication_member_expel_timeout](#) システム変数をサポートしていない MySQL Server バージョンのグループメンバーの動作でもあります。MySQL 8.0.21 からは、値はデフォルトで 5 に設定されます。つまり、疑わしいメンバーは、5 秒間の検出期間の 5 秒後に強制的に実行されます。グループのすべてのメンバーが [group_replication_member_expel_timeout](#) に対して同じ設定を持つことは必須ではありませんが、予期しない実行を回避するためにお勧めします。すべてのメンバーは、それ自体を含む他のメンバーの疑わしいものを作成できるため、有効な明示タイムアウトは、設定が最も小さいメンバーの疑わしいタイムアウトです。

次のシナリオでは、[group_replication_member_expel_timeout](#) の値をデフォルトから増やすことを検討してください:

- ネットワークの速度が遅く、強制終了までのデフォルトの 5 秒または 10 秒の長さが不足しているため、グループメンバーは常に少なくとも 1 つのメッセージを交換できません。
- ネットワークに一時的な停止があり、この時点で不要な削除およびプライマリメンバーの変更を回避する必要があります。
- ネットワークは直接制御されておらず、オペレータの介入の必要性を最小限に抑える必要があります。
- 一時的なネットワーク停止が予想され、このために一部またはすべてのメンバーを削除しない場合。
- 個々のマシンで速度が低下しており、グループから削除しない場合。

最大 3600 秒 (1 時間) までの指数タイムアウトを指定できます。XCom メッセージキャッシュが、指定した期間内の予想されるメッセージ量に最初の 5 秒の検出期間を加えて十分に大きいことを確認することが重要です。そうしないと、メンバーは再接続できません。[group_replication_message_cache_size](#) システム変数を使用して、キャッシュサイズ制限を調整できます。詳細は、[セクション18.6.5「XCom キャッシュ管理」](#)を参照してください。

グループ内のいずれかのメンバーが現在疑わしい場合、グループメンバーシップを再構成することはできません (メンバーを追加または削除するか、新しいリーダーを選択します)。1 つ以上のメンバーが疑わしいときにグループメンバーシップの変更を実装する必要があり、疑わしいメンバーをグループに残す場合は、可能であれば、メンバーを再度アクティブにするために必要なアクションを実行します。メンバーを再度アクティブにできず、グループから削除する場合は、疑わしいメンバーをただちに強制的にタイムアウトさせることができます。これを行うには、アクティブなメンバーの [group_replication_member_expel_timeout](#) の値を、疑いが作成されてからすでに経過した時間より小さい値に変更します。疑わしいメンバーはすぐに強制的責任を負います。

レプリケーショングループメンバーが予期せず停止し、すぐに再起動された場合 (たとえば、[mysqld_safe](#) で起動されたため)、[group_replication_start_on_boot=on](#) が設定されている場合は、グループへの再参加が自動的に試行されます。この状況では、メンバーの前のインカネーションがグループから削除される前に、再起動および再結合の試行が行われる可能性があります。その場合、メンバーは再結合できません。MySQL 8.0.19 からは、グループレプリケーションはグループ通信システム (GCS) 機能を自動的に使用して、再試行ごとに 5 秒間隔でメンバーの再結合試行を 10 回再試行します。これはほとんどのケースに対応し、前のインカネーションをグループから削除してメンバーを再結合できるようにするのに十分な時間を確保する必要があります。メンバーが削除される前に、より長い待機期間を指定するように [group_replication_member_expel_timeout](#) システム変数が設定されている場合でも、自動再結合の試行は成功しない可能性があります。

[group_replication_member_expel_timeout](#) システム変数が使用できない場合に不要な削除を回避するための代替軽減戦略については、[セクション18.9.2「グループレプリケーションの制限事項」](#)を参照してください。

18.6.6.2 使用できない大多数のタイムアウト

デフォルトでは、ネットワークパーティションのために少数民族で自分自身を検索するメンバーは、グループから自動的に退出しません。システム変数 [group_replication_unreachable_majority_timeout](#) を使用して、大部分のグループ

メンバーとの接続が失われた後にメンバーが待機する秒数を設定し、グループを終了できます。タイムアウトを設定すると、ネットワークパーティション後に少数民族グループ内のサーバーをプロアクティブに監視する必要がなくなり、不適切な介入のためにスプリットブレイン状況(グループメンバーシップの2つのバージョン)を作成する可能性を回避できます。

`group_replication_unreachable_majority_timeout` で指定されたタイムアウトが経過すると、メンバーおよび少数民族グループ内の他のユーザーによって処理された保留中のすべてのトランザクションがロールバックされ、そのグループ内のサーバーは `ERROR` 状態に移行します。MySQL 8.0.16 から使用可能な `group_replication_autorejoin_tries` システム変数を使用して、この時点でメンバーがグループへの再参加を自動的に試行するようにできます。MySQL 8.0.21 からは、この機能はデフォルトでアクティブ化され、メンバーは3回の自動再結合を試行します。自動再結合プロシージャが成功しなかった場合、または試行されなかった場合、少数民族メンバーは `group_replication_exit_state_action` で指定された終了アクションに従います。

到達不能な大多数のタイムアウトを設定するかどうかを決定する場合は、次の点を考慮してください:

- 対称グループ(たとえば、2つまたは4つのサーバーを含むグループ)では、両方のパーティションに同数のサーバーが含まれている場合、両方のグループが自分自身を少数民族とみなし、`ERROR` 状態になります。この場合、グループには機能パーティションがありません。
- 少数民族グループが存在する間、少数民族グループによって処理されたトランザクションは受け入れられますが、少数民族サーバーがクォーラムに到達できないため、これらのサーバーで `STOP GROUP_REPLICATION` が発行されるか、大多数のタイムアウトに達するまでブロックされます。
- 到達不能な大多数のタイムアウトを設定しない場合、少数民族グループのサーバーは `ERROR` 状態に自動的に入らないため、手動で停止する必要があります。
- 過半数の損失が検出された後に少数民族グループのサーバーに設定されている場合、大多数のタイムアウトを設定しても効果はありません。

`group_replication_unreachable_majority_timeout` システム変数を使用しない場合、ネットワークパーティションの発生時にオペレータが発見するプロセスについては、[セクション18.4.4「ネットワークパーティション化」](#)を参照してください。このプロセスでは、どのサーバーが機能しているかを確認し、必要に応じて新しいグループメンバーシップを強制します。

18.6.6.3 Auto-Rejoin

MySQL 8.0.16 から使用可能な `group_replication_autorejoin_tries` システム変数を使用すると、大多数のメンバーが削除されるか、アクセスできないタイムアウトに達した場合に、グループへの再参加が自動的に試行されます。MySQL 8.0.20 までは、システム変数の値はデフォルトで0になるため、自動再結合はデフォルトではアクティブ化されません。MySQL 8.0.21 からは、システム変数の値はデフォルトで3に設定されます。これは、メンバーがグループへの再参加を自動的に3回試行することを意味し、それぞれの間隔は5分です。

自動再結合がアクティブ化されていない場合、メンバーは通信を再開するとすぐに式を受け入れ、`group_replication_exit_state_action` システム変数で指定されたアクションに進みます。この後、メンバーをグループに戻すには手動操作が必要です。自動再結合機能の使用は、失効した読取りの可能性を許容でき、特に一時的なネットワークの問題によってメンバーが説明されることがかなり多い場合に、手動操作の必要性を最小限に抑える場合に適しています。

自動再結合では、メンバー削除または到達不能な大部分のタイムアウトに達すると、(現在のプラグインオプション値を使用して)再結合が試行され、指定された試行回数までさらに自動再結合が試行されます。自動再結合の試行に失敗すると、メンバーは次の試行の5分前に待機します。自動再結合の試行とその間の時間は、自動再結合プロシージャと呼ばれます。指定された試行回数がメンバーの再参加または停止なしで使い果たされた場合、メンバーは `group_replication_exit_state_action` システム変数で指定されたアクションに進みます。

自動再結合の試行の間、メンバーはスーパー読取り専用モードのまま、レプリケーショングループのビューに `ERROR` の状態が表示されます。この間、メンバーは書き込みを受け入れません。ただし、時間の経過とともに失効した読取りの可能性を高くすることで、メンバーに対して読取りを実行できます。自動再結合プロシージャ中にメンバーをオフラインにするために介入する場合は、`STOP GROUP_REPLICATION` ステートメントを使用するか、サーバーを停止することで、いつでも手動でメンバーを停止できます。一定期間失効した読取りの可能性を許容できない場合は、`group_replication_autorejoin_tries` システム変数を0に設定します。

パフォーマンススキーマを使用して自動再結合手順をモニターできます。自動再結合プロシージャが実行されている間、「パフォーマンススキーマ」テーブル `events_stages_current` にはイベント「進行中の自動再結合

「プロシージャ」が表示され、プロシージャのこのインスタンス (`WORK_COMPLETED` フィールド内) でこれまで試行された再試行回数が表示されます。 `events_stages_summary_global_by_event_name` テーブルには、 (`COUNT_STAR` フィールドの) サーバインスタンスが自動再結合プロシージャを開始した回数が表示されます。 `events_stages_history_long` テーブルには、これらの各自動再結合プロシージャが完了した時間 (`TIMER_END` フィールド内) が表示されます。

18.6.6.4 終了処理

MySQL 8.0.12 および MySQL 5.7.24 から使用可能な `group_replication_exit_state_action` システム変数は、エラーまたは問題が原因でメンバーが意図せずグループを離れたときに行われるグループレプリケーションを指定し、自動再結合に失敗するか、試行しません。 削除されたメンバーの場合、メンバーはグループに再接続するまで削除されたことを認識しないため、指定されたアクションが実行されるのは、メンバーが再接続を管理している場合、またはメンバー自体で疑わしい場合のみです。

影響を受ける順序で、exit アクションは次のとおりです:

1. `READ_ONLY` が終了アクションの場合、インスタンスはシステム変数 `super_read_only` を `ON` に設定することで、MySQL をスーパー読み取り専用モードに切り替えます。 メンバーがスーパー読み取り専用モードの場合、クライアントは `CONNECTION_ADMIN` 権限 (または非推奨の `SUPER` 権限) を持っても更新を実行できません。 ただし、クライアントはデータを読み取ることができ、更新が行われなくなったため、時間の経過とともに増加する失効した読み取りの確率があります。 したがって、この設定では、サーバの障害をプロアクティブに監視する必要があります。 この終了アクションは、MySQL 8.0.15 のデフォルトです。 この exit アクションが実行されると、グループのビューにメンバーステータスが `ERROR` として表示されます。
2. `OFFLINE_MODE` が終了アクションの場合、インスタンスはシステム変数 `offline_mode` を `ON` に設定することで、MySQL をオフラインモードに切り替えます。 メンバーがオフラインモードの場合、接続されたクライアントユーザーは次のリクエストで切断され、`CONNECTION_ADMIN` 権限 (または非推奨の `SUPER` 権限) を持つクライアントユーザーを除き、接続は受け入れられなくなります。 Group Replication は、システム変数 `super_read_only` を `ON` に設定することもできるため、クライアントが `CONNECTION_ADMIN` または `SUPER` 権限で接続されていても更新を行うことはできません。 この終了アクションは、更新と失効した読み取りの両方を防止し (指定された権限を持つクライアントユーザーによる読み取りを除く)、MySQL Router などのプロキシツールがサーバが使用不可であることを認識し、クライアント接続をリダイレクトできるようにします。 また、管理者が MySQL を停止せずに問題の解決を試みることができるように、インスタンスは実行されたままになります。 この終了アクションは、MySQL 8.0.18 から使用できます。 この終了アクションが実行されると、 (`OFFLINE` ではなく) グループのビューにメンバーステータスが `ERROR` として表示されます。 つまり、メンバーはグループレプリケーション機能を使用できますが、現在グループに属していません。
3. `ABORT_SERVER` が exit アクションの場合、インスタンスは MySQL を停止します。 メンバーに自身を停止するよう指示すると、失効したすべての読み取りおよびクライアントの更新が防止されますが、そのステップなしで問題が解決された可能性がある場合でも、MySQL Server インスタンスは使用できないため、再起動する必要があります。 この終了アクションは、システム変数が MySQL 8.0.15 に追加されたときの MySQL 8.0.12 のデフォルトでした。 この exit アクションが実行されると、グループのビュー内のサーバのリストからメンバーが削除されます。

自動再結合の試行を使い果たした (または一度も削除されていない)、グループレプリケーションを再起動せずにグループから再度参加することは許可されていないため、設定されている終了アクションにはオペレータの介入が必要であることを注意してください。 exit アクションは、クライアントがグループに再参加できなかったサーバ上のデータを読み取ることができるかどうか、およびサーバが実行中かどうかのみ影響します。

重要

メンバーがグループに正常に参加する前に障害が発生した場合は、`group_replication_exit_state_action` 取得されませんで指定されている終了アクション。 これは、ローカル構成チェック中に障害が発生した場合、または参加メンバーの構成とグループの構成が一致しない場合です。 このような状況では、`super_read_only` システム変数は元の値のまま、サーバは MySQL を停止しません。 したがって、Group Replication が起動しなかったときにサーバが更新を受け入れられないようにするには、起動時に `super_read_only=ON` をサーバ構成ファイルに設定することをお勧めします。 これは、正常に起動された後、プライマリメンバー上の `OFF` にグループレプリケーションが変更されます。 この保護策は、サーバがサーバ起動時に Group Replication (`group_replication_start_on_boot=ON`) を起動するように構成されている場合に特に重要です。

が、`START GROUP_REPLICATION` コマンドを使用して Group Replication を手動で起動する場合にも役立ちます。

メンバーがグループに正常に参加した後に障害が発生した場合は、指定された exit アクションが実行されます。これは、次の状況で発生します:

1. アプライヤエラー - レプリケーションアプライアンスにエラーがあります。この問題はリカバリできません。
2. 分散リカバリはできません - Group Replication 分散リカバリプロセス (バイナリログからのリモートクローニング操作と状態転送を使用) を完了できないことを意味する問題があります。グループレプリケーションは、意味のある場所で分散リカバリを自動的に再試行しますが、プロセスを完了するオプションがなくなると停止します。詳細は、[セクション18.4.3.4「分散リカバリのフォルトトレランス」](#)を参照してください。
3. グループ構成変更エラー - [セクション18.4.1「オンライングループの構成」](#)で説明されているように、UDF を使用して実行されたグループ全体の構成変更中にエラーが発生しました。
4. プライマリ選択エラー - [セクション18.1.3.1「シングルプライマリモード」](#)で説明されているように、シングルプライマリモードでのグループの新規プライマリメンバーの選択中にエラーが発生しました。
5. 大部分のタイムアウトに到達できません - メンバーは大部分のグループメンバーとの連絡を失ったため、少数民族になり、`group_replication_unreachable_majority_timeout` システム変数によって設定されたタイムアウトの期限が切れました。
6. グループから削除されたメンバー - メンバーで疑いが発生し、`group_replication_member_expel_timeout` システム変数によって設定されたタイムアウトが期限切れになり、メンバーがグループとの通信を再開し、削除されたことがわかりました。
7. 自動再結合試行の範囲外 - `group_replication_autorejoin_tries` システム変数は、大部分または明示が失われた後の自動再結合の試行回数を指定するように設定されており、メンバーは成功せずにこの回数の試行を完了しました。

次のテーブルに、失敗のシナリオと各ケースのアクションをまとめます:

表 18.4 Group Replication の失敗状況での終了アクション

失敗状況	グループレプリケーションが <code>START GROUP_REPLICATION</code> で開始されました	グループレプリケーションが <code>group_replication_start_on_boot = ON</code> で開始されました
メンバーがローカル構成チェックに失敗しました	<code>super_read_only</code> および <code>offline_mode</code> は変更されません	<code>super_read_only</code> および <code>offline_mode</code> は変更されません
参加メンバーとグループ構成が一致しません	MySQL は実行を継続 起動時に <code>super_read_only=ON</code> を設定して更新を防止	MySQL は実行を継続 起動時に <code>super_read_only=ON</code> を設定して更新を防止 (重要)
メンバーの適用者エラー	<code>super_read_only</code> を ON に設定	<code>super_read_only</code> を ON に設定
分散リカバリはできません	OR	OR
グループ構成変更エラー	<code>offline_mode</code> および <code>super_read_only</code> が ON に設定されている	<code>offline_mode</code> および <code>super_read_only</code> が ON に設定されている
プライマリ選択エラー	OR	OR
大部分のタイムアウトに到達できません	MySQL が停止	MySQL が停止
グループから削除されたメンバー		
自動再結合試行の範囲外		

18.7 グループレプリケーションのアップグレード

このセクションでは、グループレプリケーション設定をアップグレードする方法について説明します。グループのメンバーをアップグレードする基本プロセスは、スタンドアロンインスタンスのアップグレードと同じです。使用可能

なアップグレードおよびタイプを実行する実際のプロセスは、[セクション2.11「MySQLのアップグレード」](#)を参照してください。インプレースアップグレードと論理アップグレードのどちらを選択するかは、グループに格納されているデータの量によって異なります。通常、インプレースアップグレードは高速であるため、お勧めします。[セクション17.5.3「レプリケーションセットアップをアップグレードする」](#)も参照してください。

オンライングループのアップグレード中に、可用性を最大化するには、異なる MySQL Server バージョンのメンバーを同時に実行する必要がある場合があります。グループレプリケーションには、アップグレード手順中に同じグループ内で異なるバージョンの MySQL を実行しているメンバーを安全に結合できる互換性ポリシーが含まれています。グループによっては、これらのポリシーの影響がグループメンバーのアップグレード順序に影響する場合があります。詳細は、[セクション18.7.1「グループ内の異なるメンバーバージョンの組合せ」](#)を参照してください。

グループを完全にオフラインにできる場合は、[セクション18.7.2「グループレプリケーションのオフラインアップグレード」](#)を参照してください。本番デプロイメントと同様に、グループをオンラインのままにする必要がある場合は、停止時間を最小限に抑えてグループをアップグレードするために使用できる様々なアプローチについて [セクション18.7.3「グループレプリケーションのオンラインアップグレード」](#)を参照してください。

18.7.1 グループ内の異なるメンバーバージョンの組合せ

Group Replication は、Group Replication プラグインがバンドルされている MySQL Server のバージョンに従ってバージョン管理されます。たとえば、メンバーが MySQL 5.7.26 を実行している場合、これは Group Replication プラグインのバージョンです。グループメンバーの MySQL Server のバージョンを確認するには、次のコマンドを発行します:

```
SELECT MEMBER_HOST, MEMBER_PORT, MEMBER_VERSION FROM performance_schema.replication_group_members;
+-----+-----+-----+
| member_host | member_port | member_version |
+-----+-----+-----+
| example.com | 3306        | 8.0.13         |
+-----+-----+-----+
```

MySQL Server のバージョンの理解およびバージョンの選択に関するガイダンスは、[セクション2.1.2「インストールする MySQL のバージョンと配布の選択」](#)を参照してください。

最適な互換性とパフォーマンスを得るには、グループのすべてのメンバーが同じバージョンの MySQL Server を実行する必要があります。グループレプリケーションを実行する必要があります。ただし、オンライングループのアップグレード中は、可用性を最大化するために、異なる MySQL Server バージョンのメンバーを同時に実行する必要がある場合があります。MySQL のバージョン間で行われた変更によっては、この状況で非互換性が発生する可能性があります。たとえば、メジャーバージョン間で機能が非推奨になっている場合、グループ内のバージョンを組み合わせると、非推奨機能に依存するメンバーで障害が発生する可能性があります。逆に、古い MySQL バージョンを実行しているグループに読取り/書き込みメンバーが存在する間に、新しい MySQL バージョンを実行しているメンバーに書き込むと、新しいリリースで導入された関数がないメンバーで問題が発生する可能性があります。

これらの問題を回避するために、グループレプリケーションには、同じグループ内で異なるバージョンの MySQL を実行しているメンバーを安全に結合できる互換性ポリシーが含まれています。メンバーはこれらのポリシーを適用して、グループに通常参加するか、読取り専用モードで参加するか、またはグループに参加しないかを決定します。どちらを選択すると、参加メンバーおよびグループの既存のメンバーの安全な操作が行われます。アップグレードシナリオでは、各サーバーはグループを離れてアップグレードし、新しいサーバーバージョンでグループに再度参加する必要があります。この時点で、メンバーは新しいサーバーバージョンにポリシーを適用します。これは、最初にグループに参加したときに適用したポリシーから変更された可能性があります。

管理者は、サーバーを適切に構成して `START GROUP_REPLICATION` ステートメントを発行することで、任意のグループへの参加を試行するようにサーバーに指示できます。グループに参加するかどうか、または読取り専用モードでグループに参加するかどうかは、グループにメンバーを追加しようとした後に結合メンバー自体によって決定および実装されます。参加メンバーは、現在のグループメンバーの MySQL Server バージョンに関する情報を受け取り、それらのメンバーとの独自の互換性を評価して、互換性があるかどうかを判断するために (既存のメンバーで使用されるポリシーではなく) 独自の MySQL Server バージョンで使用されるポリシーを適用します。

グループに参加しようとしたときに参加メンバーが適用される互換性ポリシーは次のとおりです:

- 既存のグループメンバーが実行されている最下位バージョンより下位の MySQL Server バージョンを実行している場合、メンバーはグループに参加しません。

- メンバーは、既存のグループメンバーが実行されている最下位バージョンと同じ MySQL Server バージョンを実行している場合、通常、グループに参加します。
- メンバーはグループに参加しますが、既存のグループメンバーが実行されている最下位バージョンより上位の MySQL Server バージョンを実行している場合、読取り専用モードのままです。この動作は、グループがマルチプライマリモードで実行されている場合にのみ異なります。これは、シングルプライマリモードで実行されているグループでは、新しく追加されたメンバーはデフォルトで読取り専用になるためです。

MySQL 8.0.17 以上を実行しているメンバーは、互換性をチェックするときにリリースのパッチバージョンを考慮します。MySQL 8.0.16 以下または MySQL 5.7 を実行しているメンバーには、メジャーバージョンのみが考慮されます。たとえば、すべてのメンバーが MySQL バージョン 8.0.13 を実行しているグループがある場合は、次のようになります：

- MySQL バージョン 5.7 を実行しているメンバーは結合されません。
- MySQL 8.0.16 を実行しているメンバーは正常に結合されます (メジャーバージョンを考慮しているため)。
- MySQL 8.0.17 を実行しているメンバーは結合されますが、読取り専用モードのままです (パッチバージョンが考慮されるため)。

MySQL 5.7.27 より前のリリースを実行しているメンバーを結合すると、すべてのグループメンバーがチェックされ、独自の MySQL Server メジャーバージョンが低いかが確認されます。したがって、いずれかのメンバーが MySQL 8.0 リリースを実行しているグループに対してこのチェックが失敗し、MySQL 5.7 を実行している他のメンバーがすでに存在する場合でもグループに参加できません。MySQL 5.7.27 から、メンバーを結合すると、最下位メジャーバージョンを実行しているグループメンバーのみがチェックされるため、他の MySQL 5.7 サーバーが存在する混合バージョングループに参加できます。

異なる MySQL Server バージョンを使用するメンバーを持つマルチプライマリモードグループでは、グループレプリケーションは、MySQL 8.0.17 以上を実行しているメンバーの読取り/書込みおよび読取り専用ステータスを自動的に管理します。メンバーがグループから離れると、現在最も低いバージョンを実行しているメンバーは自動的に読取り/書込みモードに設定されます。シングルプライマリモードで実行されていたグループをマルチプライマリモードで実行するように変更すると、`group_replication_switch_to_multi_primary_mode()` UDF を使用して、グループレプリケーションによって自動的にメンバーが正しいモードに設定されます。メンバーは、グループに存在する最低バージョンより上位の MySQL サーバーバージョンを実行しており、最低バージョンを実行しているメンバーが読取り/書込みモードになっている場合、自動的に読取り専用モードになります。

18.7.1.1 アップグレード中のメンバーバージョン

オンラインアップグレード手順中に、グループがシングルプライマリモードの場合、アップグレード用に現在オフラインになっていないすべてのサーバーは、以前と同様に機能します。グループは、[セクション18.1.3.1「シングルプライマリモード」](#)で説明されている選択ポリシーに従って、必要に応じて新しいプライマリを選択します。プライマリを全体で同じに保つ必要がある場合 (アップグレード時を除く)、最初にすべてのセカンダリをターゲットプライマリメンバーバージョン以上のバージョンにアップグレードしてから、プライマリを最後にアップグレードする必要があります。プライマリは、グループ内で最も低い MySQL Server バージョンを実行していないかぎり、プライマリとして残すことはできません。プライマリがアップグレードされた後、`group_replication_set_as_primary()` UDF を使用してプライマリとして再サポートできます。

グループがマルチプライマリモードの場合、アップグレードされたメンバーはアップグレード後に読取り専用モードで結合されるため、アップグレード手順中に書込みを実行できるオンラインメンバーは少なくなります。MySQL 8.0.17 から、これはパッチバージョン間のアップグレードに適用され、以前のリリースではメジャーバージョン間のアップグレードにのみ適用されます。すべてのメンバーが MySQL 8.0.17 から同じリリースにアップグレードされると、読取り/書込みモードに自動的に戻ります。以前のリリースでは、アップグレード後にプライマリとして機能する各メンバーで、`super_read_only` を `OFF` に手動で設定する必要があります。

アップグレードをロールバックする必要がある場合や緊急時にグループに容量を追加する必要がある場合など、問題の状況に対処するために、他のグループメンバーが使用している最低バージョンより低い MySQL Server バージョンを実行していても、メンバーがオンライングループに参加できるようにすることができます。Group Replication システム変数 `group_replication_allow_local_lower_version_join` は、このような状況で通常の互換性ポリシーをオーバーライドするために使用できます。オプションを `ON` に設定しても、新しいメンバーはグループと互換性がなく、既存のメンバーによる互換性のない動作から保護せずにグループに参加できることに注意してください。したがって、このオプションは特定の状況でのみ慎重に使用する必要があり、通常のグループアクティビティが原因で新しいメンバーが失敗しないように、追加の注意事項を講じる必要があります。これらの注意事項の詳細は、`group_replication_allow_local_lower_version_join` の説明を参照してください。

18.7.1.2 グループレプリケーション通信プロトコルのバージョン

レプリケーショングループは、MySQL Server バージョンのメンバーとは異なるグループレプリケーション通信プロトコルバージョンを使用します。グループ通信プロトコルのバージョンを確認するには、任意のメンバーに対して次のステートメントを発行します：

```
SELECT group_replication_get_communication_protocol();
```

戻り値は、このグループに参加してグループ通信プロトコルを使用できる最も古い MySQL Server バージョンを示します。MySQL 5.7.14 のバージョンではメッセージを圧縮でき、MySQL 8.0.16 のバージョンではメッセージを断片化することもできます。 `group_replication_get_communication_protocol()` UDF は、グループがサポートする MySQL の最小バージョンを返します。これは、 `group_replication_set_communication_protocol()` UDF に渡されたバージョン番号、および UDF を使用するメンバーにインストールされている MySQL Server バージョンとは異なる場合があります。

レプリケーショングループのすべてのメンバーを新しい MySQL Server リリースにアップグレードしても、以前のリリースのメンバーが参加できるようにする必要がある場合に備えて、グループレプリケーション通信プロトコルのバージョンは自動的にアップグレードされません。古いメンバーをサポートする必要がなく、アップグレードされたメンバーが追加された通信機能を使用できるようにする場合は、アップグレード後に `group_replication_set_communication_protocol()` UDF を使用して通信プロトコルをアップグレードし、メンバーのアップグレード先の新しい MySQL Server バージョンを指定します。詳細は、[セクション18.4.1.4「グループ通信プロトコルバージョンの設定」](#)を参照してください。

18.7.2 グループレプリケーションのオフラインアップグレード

グループレプリケーショングループのオフラインアップグレードを実行するには、グループから各メンバーを削除し、メンバーのアップグレードを実行してから、通常どおりにグループを再起動します。マルチプライマリグループでは、任意の順序でメンバーを停止できます。単一プライマリグループでは、最初に各セカンダリを停止してから、最後にプライマリを停止します。グループからメンバーを削除して MySQL を停止する方法は、[セクション18.7.3.2「グループレプリケーションメンバーのアップグレード」](#)を参照してください。

グループがオフラインになったら、すべてのメンバーをアップグレードします。アップグレードの実行方法は、[セクション2.11「MySQL のアップグレード」](#)を参照してください。すべてのメンバーがアップグレードされたら、メンバーを再起動します。

レプリケーショングループがオフラインのときにすべてのメンバーをアップグレードしてからグループを再起動すると、メンバーは新しいリリースのグループレプリケーション通信プロトコルバージョンを使用して参加し、グループ通信プロトコルバージョンになります。以前のリリースのメンバーの参加を許可する要件がある場合は、 `group_replication_set_communication_protocol()` UDF を使用して通信プロトコルバージョンをダウングレードし、最も古いサーバーバージョンがインストールされている見込みグループメンバーの MySQL Server バージョンを指定できます。

18.7.3 グループレプリケーションのオンラインアップグレード

アップグレードするグループを実行しているが、アプリケーションを提供するためにグループをオンラインのままにする必要がある場合は、アップグレードのアプローチを検討する必要があります。このセクションでは、オンラインアップグレードに関連する様々な要素と、グループのアップグレード方法について説明します。

18.7.3.1 オンラインアップグレードに関する考慮事項

オンライングループをアップグレードする場合は、次の点を考慮する必要があります：

- グループのアップグレード方法に関係なく、グループメンバーへの書き込みは、グループに再度参加する準備ができるまで無効にすることが重要です。
- メンバーが停止すると、 `super_read_only` 変数は自動的にオンに設定されますが、この変更は永続化されません。
- MySQL 5.7.22 または MySQL 8.0.11 が MySQL 5.7.21 以下を実行しているグループに参加しようとする場合、MySQL 5.7.21 は `lower_case_table_names` の値を送信しないため、グループへの参加に失敗します。

18.7.3.2 グループレプリケーションメンバーのアップグレード

このセクションでは、グループのメンバーをアップグレードするために必要なステップについて説明します。このプロセスは、[セクション18.7.3.3「グループレプリケーションのオンラインアップグレード方法」](#)で説明されているメソッドの一部です。グループのメンバーをアップグレードするプロセスは、すべての方法に共通であり、最初に説明します。アップグレードしたメンバーを結合する方法は、フォローしている方法、およびグループがシングルプライマリモードで動作しているかマルチプライマリモードで動作しているかなどのその他の要因によって異なります。インプレースまたはプロビジョニングのいずれかのアプローチを使用してサーバーインスタンスをアップグレードする方法は、ここで説明する方法には影響しません。

メンバーをアップグレードするプロセスは、メンバーをアップグレードしてからアップグレードしたメンバーをグループに再度参加させる選択した方法に従って、メンバーをグループから削除することです。単一プライマリグループのメンバーのアップグレードの推奨順序は、すべてのセカンダリをアップグレードしてから、プライマリを最後にアップグレードすることです。プライマリがセカンダリの前にアップグレードされた場合、古いMySQLバージョンを使用する新しいプライマリが選択されますが、このステップは必要ありません。

グループのメンバーをアップグレードするには:

- クライアントをグループメンバーに接続し、`STOP GROUP_REPLICATION` を発行します。続行する前に、`replication_group_members` テーブルを監視して、メンバーステータスが `OFFLINE` であることを確認します。
- グループレプリケーションが自動的に起動しないようにして、アップグレード後に安全にメンバーに接続し、`group_replication_start_on_boot=0` を設定してグループに再参加せずにメンバーを構成できるようにします。

重要

アップグレードされたメンバーに `group_replication_start_on_boot=1` がある場合、MySQL のアップグレード手順を実行する前にグループに再度参加すると、問題が発生する可能性があります。たとえば、アップグレードが失敗し、サーバーが再起動した場合、破損している可能性のあるサーバーがグループに参加しようとする可能性があります。

- たとえば、`mysqladmin shutdown` または `SHUTDOWN` ステートメントを使用してメンバーを停止します。グループ内の他のメンバーは引き続き実行されます。
- インプレースまたはプロビジョニングのアプローチを使用して、メンバーをアップグレードします。詳細は、[セクション2.11「MySQL のアップグレード」](#)を参照してください。アップグレードしたメンバーを再起動する場合、`group_replication_start_on_boot` が 0 に設定されているため、グループレプリケーションはインスタンスで開始されないため、グループに再参加しません。
- メンバーでMySQLのアップグレード手順が実行されたら、`group_replication_start_on_boot` を 1 に設定して、再起動後にグループレプリケーションが正しく開始されるようにする必要があります。メンバーを再起動します。
- アップグレードしたメンバーに接続し、`START GROUP_REPLICATION` を発行します。これにより、メンバーがグループに再結合されます。グループレプリケーションメタデータはアップグレードされたサーバーに配置されているため、通常はグループレプリケーションを再構成する必要はありません。サーバーは、サーバーがオフラインのときにグループによって処理されたトランザクションをキャッチアップする必要があります。グループで捕捉されると、グループのオンラインメンバーになります。

注記

サーバーのアップグレードにかかる時間が長くなるほど、メンバーがオフラインになる時間が長くなるため、サーバーがグループに追加して戻されるまでにかかる時間が長くなります。

アップグレードされたメンバーが、以前のMySQL Serverバージョンを実行しているメンバーを持つグループに参加すると、アップグレードされたメンバーは `super_read_only=on` と結合されます。これにより、すべてのメンバーが新しいバージョンを実行するまで、アップグレードされたメンバーへの書き込みが行われなくなります。マルチプライマリモードグループでは、アップグレードが正常に完了し、グループがトランザクションを処理する準備ができた場合、書き込み可能なプライマリとして意図されているメンバーを讀取り/書き込みモードに設定する必要があります。MySQL 8.0.17 からは、グループのすべてのメンバーが同じリリースにアップグレードされると、讀取り/書き込みモードに自動的に戻ります。以前のリリースでは、各メンバーを手動で讀取り/書き込みモードに設定する必要があります。各メンバーに接続し、次を発行します:

```
SET GLOBAL super_read_only=OFF;
```

18.7.3.3 グループレプリケーションのオンラインアップグレード方法

グループレプリケーショングループをアップグレードするには、次のいずれかの方法を選択します:

ローリングアップグレード

この方法は、新しいバージョンを実行しているサーバーがグループにワークロードを生成せず、古いバージョンのサーバーが存在する場合にサポートされます。つまり、新しいバージョンのサーバーは、セカンダリとしてのみグループに参加できます。この方法では、1つのグループのみが存在し、各サーバーインスタンスがグループから削除され、アップグレードされてグループに再度参加します。

この方法は、単一プライマリグループに適しています。グループがシングルプライマリモードで動作しているときに、プライマリを全体で同じにしておく必要がある場合(アップグレードされる場合を除く)、アップグレードされる最後のメンバーである必要があります。プライマリは、グループ内で最も低いMySQL Serverバージョンを実行していないかぎり、プライマリとして残すことはできません。プライマリがアップグレードされた後、`group_replication_set_as_primary()` UDF を使用してプライマリとして再サポートできます。どのメンバーがプライマリであるかわからない場合は、任意の順序でメンバーをアップグレードできます。グループは、[セクション 18.1.3.1「シングルプライマリモード」](#)で説明されている選択ポリシーに従って、MySQL Serverの最低バージョンを実行しているメンバーの中から必要に応じて新しいプライマリを選択します。

マルチプライマリモードで動作するグループの場合、ローリングアップグレード中にプライマリの数が減り、書き込み可用性が低下します。これは、既存のグループメンバーが実行されている最下位バージョンより上位のMySQL Serverバージョンを実行している場合、メンバーがグループに参加すると、自動的に読取り専用モード(`super_read_only=ON`)のままになるためです。MySQL 8.0.17以上を実行しているメンバーは、これをチェックするときにリリースのパッチバージョンを考慮しますが、MySQL 8.0.16以下またはMySQL 5.7を実行しているメンバーはメジャーバージョンのみを考慮します。すべてのメンバーがMySQL 8.0.17から同じリリースにアップグレードされると、読取り/書き込みモードに自動的に戻ります。以前のリリースでは、アップグレード後にプライマリとして機能する必要がある各メンバーで`super_read_only=OFF`を手動で設定する必要があります。

グループ内のバージョンの互換性、およびこれがアップグレードプロセス中にグループの動作に与える影響の詳細は、[セクション 18.7.1「グループ内の異なるメンバーバージョンの組合せ」](#)を参照してください。

ローリング移行のアップグレード

この方法では、グループからメンバーを削除し、アップグレードしたメンバーを使用して別のグループを作成します。マルチプライマリモードで動作するグループの場合、このプロセス中にプライマリの数が減り、書き込み可用性が低下します。これは、シングルプライマリモードで動作するグループには影響しません。

古いバージョンを実行しているグループは、メンバーのアップグレード中にオンラインであるため、メンバーのアップグレード中に実行されたトランザクションを捕捉するには、新しいバージョンを実行しているグループが必要です。したがって、新しいグループ内のいずれかのサーバーは、古いグループのプライマリのレプリカとして構成されます。これにより、新しいグループが古いグループに追いつくようになります。この方法は、あるグループから別のグループへのデータのレプリケートに使用される非同期レプリケーションチャンネルに依存するため、非同期ソースレプリケーションレプリケーションと同じ前提および要件でサポートされます。[第17章「レプリケーション」](#)を参照してください。シングルプライマリモードで動作するグループの場合、古いグループへの非同期レプリケーション接続では、新しいグループのプライマリにデータを送信する必要があります。マルチプライマリグループの場合、非同期レプリケーションチャンネルは任意のプライマリに接続できます。

プロセスは次のとおりです:

- 古いサーバーバージョンを実行している元のグループからメンバーを削除します。[セクション 18.7.3.2「グループレプリケーションメンバーのアップグレード」](#)を参照してください
- メンバーで実行されているサーバーバージョンをアップグレードします。[セクション 2.11「MySQLのアップグレード」](#)を参照してください。インプレースまたはプロビジョニングのアプローチに従ってアップグレードできます。
- アップグレードしたメンバーで新しいグループを作成します。[第18章「グループレプリケーション」](#)を参照してください。この場合、(古いグループがまだ実行中で古い名前を使用しているため)各メンバーに新しいグループ名を構成し、最初にアップグレードしたメンバーをブートストラップしてから、残りのアップグレード済メンバーを追加する必要があります。
- 古いグループと新しいグループの間に非同期レプリケーションチャンネルを設定します。[セクション 17.1.3.4「GTIDを使用したレプリケーションのセットアップ」](#)を参照してください。古いプライマリが非同期レプリケーション

ソースサーバーとして機能し、新しいグループメンバーが GTID ベースのレプリカとして機能するように構成します。

アプリケーションを新しいグループにリダイレクトする前に、グループがメンバーの障害を処理できるように、新しいグループに適切な数のメンバーがあることを確認する必要があります。 `SELECT * FROM performance_schema.replication_group_members` を発行し、初期グループサイズと新しいグループサイズを比較します。古いグループのすべてのデータが新しいグループに伝播されるまで待機してから、非同期レプリケーション接続を削除し、欠落しているメンバーをアップグレードします。

ローリングアップグレード

この方法では、新しいバージョンを実行しているメンバーで構成される別のグループを作成し、古いグループから欠落しているデータは新しいグループにレプリケートされます。これは、両方のグループを同時に実行するのに十分なサーバーがあることを前提としています。このプロセス中にプライマリの数が減少しないため、マルチプライマリモードで動作するグループの場合、書き込み可用性は低下しません。これにより、ローリングアップグレードはマルチプライマリモードで動作するグループに適しています。これは、シングルプライマリモードで動作するグループには影響しません。

新しいグループのメンバーをプロビジョニングしている間、古いバージョンを実行しているグループはオンラインであるため、メンバーのプロビジョニング中に実行されたトランザクションを捕捉するには、新しいバージョンを実行しているグループが必要です。したがって、新しいグループ内のいずれかのサーバーは、古いグループのプライマリのレプリカとして構成されます。これにより、新しいグループが古いグループに追いつくようになります。この方法は、あるグループから別のグループへのデータのレプリケートに使用される非同期レプリケーションチャンネルに依存するため、非同期ソースレプリケーションレプリケーションと同じ前提および要件でサポートされます。第17章「レプリケーション」を参照してください。シングルプライマリモードで動作するグループの場合、古いグループへの非同期レプリケーション接続では、新しいグループのプライマリにデータを送信する必要があります。マルチプライマリグループの場合、非同期レプリケーションチャンネルは任意のプライマリに接続できます。

プロセスは次のとおりです:

- 新しいバージョンを実行しているグループがメンバーの障害を処理できるように、適切な数のメンバーをデプロイ
- グループのメンバーから既存のデータのバックアップを取得
- 古いメンバーのバックアップを使用して、新しいグループのメンバーをプロビジョニングします。方法については、[セクション18.7.3.4「mysqldumpを使用したグループレプリケーションアップグレード」](#)を参照してください。

注記

バックアップは、バックアップ元と同じバージョンの MySQL にリストアしてから、インプレースアップグレードを実行する必要があります。その手順は、[セクション2.11「MySQLのアップグレード」](#)を参照してください。

- アップグレードしたメンバーで新しいグループを作成します。第18章「グループレプリケーション」を参照してください。この場合、(古いグループがまだ実行中で古い名前を使用しているため)各メンバーに新しいグループ名を構成し、最初にアップグレードしたメンバーをブートストラップしてから、残りのアップグレード済メンバーを追加する必要があります。
- 古いグループと新しいグループの間に非同期レプリケーションチャンネルを設定します。[セクション17.1.3.4「GTIDを使用したレプリケーションのセットアップ」](#)を参照してください。古いプライマリが非同期レプリケーションソースサーバーとして機能し、新しいグループメンバーが GTID ベースのレプリカとして機能するように構成します。

新しいグループから欠落している進行中のデータが、迅速に転送できるほど小さい場合は、書き込み操作を新しいグループにリダイレクトする必要があります。古いグループのすべてのデータが新しいグループに伝播されるまで待機してから、非同期レプリケーション接続を削除します。

18.7.3.4 mysqldump を使用したグループレプリケーションアップグレード

プロビジョニング方法の一環として、MySQL Enterprise Backup を使用して、グループメンバーから新しいメンバーにデータをコピーおよびリストアできます。ただし、この方法を使用して、古いバージョンの MySQL を実行しているメンバーから、新しいバージョンの MySQL を実行しているメンバーにバックアップを直接リストアすることはできません。

きません。解決策は、バックアップ元のメンバーと同じバージョンの MySQL を実行している新しいサーバーインスタンスにバックアップをリストアしてから、インスタンスをアップグレードすることです。このプロセスの内容は次のとおりです:

- `mysqlbackup` を使用して、古いグループのメンバーからバックアップを取得します。 [セクション18.4.6「グループレプリケーションでの MySQL Enterprise Backup の使用」](#) を参照してください。
- 新しいサーバーインスタンスをデプロイします。これは、バックアップが作成された古いメンバーと同じバージョンの MySQL を実行している必要があります。
- `mysqlbackup` を使用して、古いメンバーから新しいインスタンスにバックアップをリストアします。
- 新しいインスタンスで MySQL をアップグレードします。 [セクション2.11「MySQL のアップグレード」](#) を参照してください。

フェイルオーバーを処理できるように、このプロセスを繰り返して適切な数の新規インスタンスを作成します。次に、 [セクション18.7.3.3「グループレプリケーションのオンラインアップグレード方法」](#) に基づいてインスタンスをグループに結合します。`

18.8 グループレプリケーションシステム変数

このセクションでは、Group Replication プラグインに固有のシステム変数を一覧表示します。すべての構成オプションには、接頭辞として `group_replication` が付きます。

Group Replication のほとんどのシステム変数は動的と記述され、その値はサーバーの実行中に変更できます。ただし、ほとんどの場合、変更が有効になるのは、`STOP GROUP_REPLICATION` ステートメントの後に `START GROUP_REPLICATION` ステートメントを使用してグループメンバーでグループレプリケーションを停止して再起動した後のみです。Group Replication を停止および再起動せずに、次のシステム変数への変更が有効になります:

- `group_replication_advertise_recovery_endpoints`
- `group_replication_autorejoin_tries`
- `group_replication_consistency`
- `group_replication_exit_state_action`
- `group_replication_flow_control_applier_threshold`
- `group_replication_flow_control_certifier_threshold`
- `group_replication_flow_control_hold_percent`
- `group_replication_flow_control_max_commit_quota`
- `group_replication_flow_control_member_quota_percent`
- `group_replication_flow_control_min_quota`
- `group_replication_flow_control_min_recovery_quota`
- `group_replication_flow_control_mode`
- `group_replication_flow_control_period`
- `group_replication_flow_control_release_percent`
- `group_replication_force_members`
- `group_replication_member_expel_timeout`
- `group_replication_member_weight`
- `group_replication_transaction_size_limit`
- `group_replication_unreachable_majority_timeout`

グループレプリケーションのほとんどのシステム変数は、グループメンバーごとに異なる値を持つことができます。次のシステム変数では、トランザクションの不要なロールバック、メッセージ配信の失敗またはメッセージリカバリの失敗を回避するために、グループのすべてのメンバーに同じ値を設定することをお勧めします:

- [group_replication_auto_increment_increment](#)
- [group_replication_communication_max_message_size](#)
- [group_replication_compression_threshold](#)
- [group_replication_message_cache_size](#)
- [group_replication_transaction_size_limit](#)

グループレプリケーション固有のシステム変数や一般的なシステム変数など、グループレプリケーショングループメンバーの一部のシステム変数は、グループ全体の構成設定です。これらのシステム変数は、すべてのグループメンバーで同じ値を持つ必要があり、グループレプリケーションの実行中は変更できません。また、値の変更を有効にするには、グループの完全な再起動 ([group_replication_bootstrap_group=ON](#) を使用したサーバーによるブートストラップ) が必要です。これらの条件は、次のシステム変数に適用されます:

- [group_replication_single_primary_mode](#)
- [group_replication_enforce_update_everywhere_checks](#)
- [group_replication_gtid_assignment_block_size](#)
- [default_table_encryption](#)
- [lower_case_table_names](#)
- [transaction_write_set_extraction](#)

MySQL 8.0.16 から、[group_replication_switch_to_single_primary_mode\(\)](#) および [group_replication_switch_to_multi_primary_mode\(\)](#) UDF を使用して、グループの実行中に [group_replication_single_primary_mode](#) および [group_replication_enforce_update_everywhere_checks](#) の値を変更できます。詳細は、[セクション18.4.1.2「グループモードの変更」](#) を参照してください。

重要

- グループレプリケーションの多くのシステム変数は、コマンドライン引数としてサーバーに渡された場合、サーバーの起動時に完全には検証されません。これらのシステム変数には、[group_replication_group_name](#)、[group_replication_single_primary_mode](#)、[group_replication_force_members](#)、SSL 変数およびフロー制御システム変数が含まれます。これらは、サーバーの起動後にのみ完全に検証されます。
- グループメンバーの IP アドレスまたはホスト名を指定するグループレプリケーションのシステム変数は、[START GROUP REPLICATION](#) ステートメントが発行されるまで検証されません。グループレプリケーショングループ通信システム (GCS) は、その時点まで値を検証できません。

Group Replication プラグインに固有のシステム変数は次のとおりです:

- [group_replication_advertise_recovery_endpoints](#)

コマンド行形式	<code>--group-replication-advertise-recovery-endpoints=value</code>
導入	8.0.21
システム変数	group_replication_advertise_recovery_endpoints
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	DEFAULT

このシステム変数の値は、Group Replication の実行中に変更できます。変更はすぐにメンバーに反映されます。ただし、システム変数の以前の値をすでに受け取っている結合メンバーは、その値を引き続き使用します。値の変更後に結合されたメンバーのみが新しい値を受け取ります。

`group_replication_advertise_recovery_endpoints` は、結合メンバーが分散リカバリのために状態転送のために既存のメンバーへの接続を確立する方法を指定します。この接続は、リモートクローニング操作とドナーのバイナリログからの状態転送の両方に使用されます。

デフォルト設定の `DEFAULT` の値は、参加メンバーが MySQL Server `hostname` および `port` システム変数で指定されている既存のメンバー標準 SQL クライアント接続を使用することを意味します。`report_port` システム変数で代替ポート番号が指定されている場合は、かわりにそのポート番号が使用されます。「パフォーマンススキーマ」テーブル `replication_group_members` では、この接続アドレスとポート番号が `MEMBER_HOST` および `MEMBER_PORT` フィールドに表示されます。これは、MySQL 8.0.20 までのリリースでのグループメンバーの動作です。

`DEFAULT` のかわりに、既存のメンバーが使用する参加メンバーに通知する分散リカバリエンドポイントを指定できます。分散リカバリエンドポイントを提供することで、管理者は、グループメンバーへの通常の MySQL クライアント接続とは別に分散リカバリトラフィックを制御できます。メンバーの結合では、リストで指定された順序で各エンドポイントが順に試行されます。

分散リカバリエンドポイントを IP アドレスとポート番号のカンマ区切りリストとして指定します。次に例を示します:

```
group_replication_advertise_recovery_endpoints="127.0.0.1:3306,127.0.0.1:4567,[::1]:3306,localhost:3306"
```

IPv4 アドレスと IPv6 アドレスおよびホスト名は、任意の組合せで使用できます。IPv6 アドレスは大カッコで囲んで指定する必要があります。ホスト名はローカル IP アドレスに解決される必要があります。ワイルドカードアドレス書式は使用できず、空のリストは指定できません。標準 SQL クライアント接続は、分散リカバリエンドポイントのリストに自動的に含まれないことに注意してください。エンドポイントとして使用する場合は、リストに明示的に含める必要があります。

分散リカバリエンドポイントとして IP アドレスとポートを選択する方法、およびメンバーを結合する方法の詳細は、[分散リカバリエンドポイントのアドレスの選択](#) を参照してください。要件のサマリーは次のとおりです:

- IP アドレスは MySQL Server 用に構成する必要はありませんが、サーバーに割り当てる必要があります。
- ポートは、`port`、`report_port` または `admin_port` システム変数を使用して MySQL Server 用に構成する必要があります。
- `admin_port` を使用する場合は、分散リカバリのレプリケーションユーザーに適切な権限が必要です。
- `group_replication_ip_allowlist` または `group_replication_ip_whitelist` システム変数で指定された Group Replication 許可リストに IP アドレスを追加する必要はありません。
- 接続の SSL 要件は、`group_replication_recovery_ssl_*` オプションで指定されたとおりです。
- `group_replication_allow_local_lower_version_join`

コマンド行形式	<code>--group-replication-allow-local-lower-version-join[={OFF ON}]</code>
システム変数	<code>group_replication_allow_local_lower_version_join</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

このシステム変数の値は、Group Replication の実行中に変更できますが、変更はグループメンバーで Group Replication を停止して再起動した後にのみ有効になります。

[group_replication_allow_local_lower_version_join](#) では、現在のサーバーがグループより低いバージョンの MySQL Server を実行している場合でも、グループに参加できます。デフォルト設定の `OFF` では、既存のグループメンバーより低いバージョンを実行しているサーバーはレプリケーショングループに参加できません。この標準ポリシーは、グループのすべてのメンバーがメッセージを交換し、トランザクションを適用できることを確認します。MySQL 8.0.17 以上を実行しているメンバーは、互換性をチェックするときにリリースのパッチバージョンを考慮することに注意してください。MySQL 8.0.16 以下または MySQL 5.7 を実行しているメンバーには、メジャーバージョンのみが考慮されます。

次のシナリオでのみ、[group_replication_allow_local_lower_version_join](#) を `ON` に設定します:

- グループのフォルトトレランスを改善するには、緊急時にサーバーをグループに追加する必要があり、古いバージョンのみを使用できます。
- グループ全体を停止して再度ブートストラップせずに、1 つ以上のレプリケーショングループメンバーのアップグレードをロールバックする場合。

警告

このオプションを `ON` に設定しても、新しいメンバーはグループと互換性がなく、既存のメンバーによる互換性のない動作から保護されずにグループに参加できます。新しいメンバーが正しく動作するようにするには、次の予防措置の `both` を使用します:

1. 下位バージョンを実行しているサーバーがグループに参加する前に、そのサーバーでのすべての書き込みを停止します。
2. 下位バージョンを実行しているサーバーがグループに参加した時点から、グループ内の他のサーバーに対するすべての書き込みを停止します。

これらの予防措置がないと、低いバージョンを実行しているサーバーで問題が発生し、エラーで終了する可能性があります。

- [group_replication_auto_increment_increment](#)

コマンド行形式	<code>--group-replication-auto-increment-increment=#</code>
システム変数	group_replication_auto_increment_increment
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	7
最小値	1
最大値	65535

このシステム変数は、すべてのグループメンバーで同じ値を持つ必要があります。グループレプリケーションの実行中は、このシステム変数の値を変更できません。各グループメンバーでグループレプリケーションを停止し、システム変数の値を変更してから、グループレプリケーションを再起動する必要があります。このプロセス中、システム変数の値はグループメンバー間で異なってもかまいませんが、グループメンバーの一部のトランザクションはロールバックされる可能性があります。

[group_replication_auto_increment_increment](#) は、このサーバーインスタンスで実行されるトランザクションの自動増分カラムの連続する値の間隔を決定します。間隔を追加すると、グループメンバーに対する書き込みに対して重複する自動増分値が選択されないため、トランザクションがロールバックされます。デフォルト値の 7 は、使用可能な値の数とレプリケーショングループの最大許容サイズ (9 メンバー) のバランスを表します。グループのメンバー

数が多いか少ない場合は、グループレプリケーションが開始される前に、グループメンバーの予想数と一致するようにこのシステム変数を設定できます。

グループレプリケーションがサーバーインスタンスで開始されると、サーバーシステム変数 `auto_increment_increment` の値がこの値に変更され、サーバーシステム変数 `auto_increment_offset` の値がサーバー ID に変更されます。グループレプリケーションが停止すると、変更は元に戻されます。これらの変更は、`auto_increment_increment` および `auto_increment_offset` のそれぞれのデフォルト値が 1 の場合にのみ行われ、元に戻されます。これらの値がすでにデフォルトから変更されている場合、Group Replication はそれらを変更しません。MySQL 8.0 からは、グループレプリケーションが単一プライマリモードで、サーバー書き込みが 1 つのみの場合も、システム変数は変更されません。

- [group_replication_autorejoin_tries](#)

コマンド行形式	<code>--group-replication-autorejoin-tries=#</code>
導入	8.0.16
システム変数	group_replication_autorejoin_tries
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値 (≥ 8.0.21)	3
デフォルト値 (≤ 8.0.20)	0
最小値	0
最大値	2016

このシステム変数の値は、Group Replication の実行中に変更でき、変更はただちに有効になります。動作が必要であることを意味する問題が発生すると、システム変数 `current` 値が読み取られます。

[group_replication_autorejoin_tries](#) では、メンバーが明示された場合、または [group_replication_unreachable_majority_timeout](#) 設定に到達する前にグループの大部分にアクセスできない場合に、メンバーが自動的にグループに再参加しようとする試行回数を指定します。メンバー削除または到達不能な大部分のタイムアウトに達すると、(現在のプラグインオプション値を使用して) 再結合を試みてから、指定された試行回数までさらに自動再結合を試みます。自動再結合の試行に失敗すると、メンバーは次の試行の 5 分前に待機します。指定された試行回数がメンバーの再参加または停止なしで使い果たされた場合、メンバーは [group_replication_exit_state_action](#) システム変数で指定されたアクションに進みます。

MySQL 8.0.20 までは、デフォルト設定は 0 です。つまり、メンバーは自動的に再結合を試みません。MySQL 8.0.21 からは、デフォルト設定は 3 です。これは、メンバーがグループへの再参加を自動的に 3 回試行し、それぞれの間に 5 分間試行することを意味します。最大 2016 回の試行を指定できます。

自動再結合の試行中および試行の間、メンバーはスーパー読み取り専用モードのまま書き込みを受け入れませんが、時間の経過とともに失効した読み取りの可能性を高くして、メンバーに対して読み取りを実行できます。一定期間失効した読み取りの可能性を許容できない場合は、[group_replication_autorejoin_tries](#) を 0 に設定します。自動再結合機能、およびこのオプションの値を選択する際の考慮事項の詳細は、[セクション 18.6.6.3 「Auto-Rejoin」](#) を参照してください。

- [group_replication_bootstrap_group](#)

コマンド行形式	<code>--group-replication-bootstrap-group[={OFF ON}]</code>
システム変数	group_replication_bootstrap_group
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean

デフォルト値	OFF
--------	-----

`group_replication_bootstrap_group` は、グループをブートストラップするようにこのサーバーを構成します。このシステム変数は、グループを初めて起動するとき、またはグループ全体を再起動するときに、あるサーバーおよびのみで設定する必要があります。グループがブートストラップされたら、このオプションを `OFF` に設定します。動的にも構成ファイルでも、`OFF` に設定する必要があります。グループの実行中に 2 つのサーバーを起動するか、このオプションを設定して 1 つのサーバーを再起動すると、人工的なスプリットブレイク状況が発生し、同じ名前の独立した 2 つのグループがブートストラップされる場合があります。

- `group_replication_clone_threshold`

コマンド行形式	<code>--group-replication-clone-threshold=#</code>
導入	8.0.17
システム変数	<code>group_replication_clone_threshold</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	<code>9223372036854775807</code>
最小値	<code>1</code>
最大値	<code>9223372036854775807</code>

このシステム変数の値は、Group Replication の実行中に変更できますが、変更はグループメンバーで Group Replication を停止して再起動した後にのみ有効になります。

`group_replication_clone_threshold` では、分散リカバリプロセス中に参加メンバーへの状態転送のためにリモートクローニング操作の使用をトリガーする既存のメンバー (ドナー) と参加メンバー (受信者) の間のトランザクションギャップを多数のトランザクションとして指定します。参加メンバーと適切なドナーの間のトランザクションギャップがしきい値を超えると、グループレプリケーションはリモートクローニング操作で分散リカバリを開始します。トランザクションギャップがしきい値を下回る場合、またはリモートクローニング操作が技術的に可能でない場合、Group Replication はドナーのバイナリログからの状態転送に直接進みます。

警告

アクティブグループの `group_replication_clone_threshold` には低い設定を使用しないでください。リモートクローニング操作の進行中にしきい値を超える数のトランザクションがグループで発生した場合、参加メンバーは再起動後にリモートクローニング操作を再度トリガーし、これを無期限に続行できます。この状況を回避するには、リモートクローニング操作にかかる時間内にグループ内で発生すると予想されるトランザクション数よりも大きい数をしきい値に設定してください。

この関数を使用するには、クローニングをサポートするようにドナーと参加メンバーの両方を事前に設定する必要があります。その手順は、[セクション 18.4.3.2 「分散リカバリのためのクローニング」](#) を参照してください。リモートクローニング操作が実行されると、`group_replication_start_on_boot=ON` が設定されている場合は、グループレプリケーションによって管理され、必要なサーバーの再起動も含まれます。そうでない場合は、サーバーを手動で再起動する必要があります。リモートクローニング操作では、結合メンバーの既存のデータディクショナリが置換されますが、結合メンバーに他のグループメンバーに存在しない追加のトランザクションがある場合、これらのトランザクションはクローニング操作によって消去されるため、グループレプリケーションによってチェックされ、続行されません。

デフォルト設定 (GTID 内のトランザクションに許可される最大シーケンス番号) は、ドナーのバイナリログからの状態転送が、クローニングではなく実質的に常に試行されることを意味します。ただし、結合メンバーに必要なトランザクションが既存のグループメンバーのバイナリログで使用できないためなど、ドナーバイナリログからの状態転送が不可能な場合、グループレプリケーションはしきい値に関係なく常にクローニング操作を実行しようとします。レプリケーショングループでクローニングをまったく使用しない場合は、クローンプラグインをメンバーにインストールしないでください。

- group_replication_communication_debug_options

コマンド行形式	--group-replication-communication-debug-options=value
システム変数	group_replication_communication_debug_options
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	GCS_DEBUG_NONE
有効な値	GCS_DEBUG_NONE GCS_DEBUG_BASIC GCS_DEBUG_TRACE XCOM_DEBUG_BASIC XCOM_DEBUG_TRACE GCS_DEBUG_ALL

このシステム変数の値は、Group Replication の実行中に変更できますが、変更はグループメンバーで Group Replication を停止して再起動した後にのみ有効になります。

group_replication_communication_debug_options は、グループ通信システム (GCS) やグループ通信エンジン (XCom、Paxos バリエーション) などの様々なグループレプリケーションコンポーネントに提供するデバッグメッセージのレベルを構成します。デバッグ情報は、データディレクトリの GCS_DEBUG_TRACE ファイルに格納されません。

文字列として指定された使用可能なオプションのセットを組み合わせることができます。次のオプションを使用できます。

- GCS_DEBUG_NONE では、GCS と XCom の両方のデバッグレベルがすべて無効になります。
- GCS_DEBUG_BASIC を使用すると、GCS で基本的なデバッグ情報を使用できます。
- GCS_DEBUG_TRACE は GCS でトレース情報を有効にします。
- XCOM_DEBUG_BASIC を使用すると、XCom の基本的なデバッグ情報を使用できます。
- XCOM_DEBUG_TRACE を使用すると、XCom でトレース情報を使用できます。
- GCS_DEBUG_ALL では、GCS と XCom の両方ですべてのデバッグレベルが有効になります。

デバッグレベルを GCS_DEBUG_NONE に設定すると、他のオプションを指定せずに指定した場合にのみ有効になります。デバッグレベルを GCS_DEBUG_ALL に設定すると、他のすべてのオプションがオーバーライドされます。

- group_replication_communication_max_message_size

コマンド行形式	--group-replication-communication-max-message-size=#
導入	8.0.16
システム変数	group_replication_communication_max_message_size
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ

型	Integer
デフォルト値	10485760
最小値	0
最大値	1073741824

このシステム変数は、すべてのグループメンバーで同じ値を持つ必要があります。グループレプリケーションの実行中は、このシステム変数の値を変更できません。各グループメンバーでグループレプリケーションを停止し、システム変数の値を変更してから、グループレプリケーションを再起動する必要があります。このプロセス中、システム変数の値はグループメンバー間で異なってもかまいませんが、グループメンバーの一部のトランザクションはロールバックされる可能性があります。

`group_replication_communication_max_message_size` では、グループレプリケーション通信の最大メッセージサイズを指定します。このサイズより大きいメッセージは、個別に送信され、受信者によって再アセンブルされるフラグメントに自動的に分割されます。詳細は、[セクション18.6.4「メッセージの断片化」](#)を参照してください。

10485760 バイト (10 MiB) の最大メッセージサイズがデフォルトで設定されています。つまり、MySQL 8.0.16 からのリリースでは、断片化がデフォルトで使用されます。最大許容値は、`slave_max_allowed_packet` システム変数の最大値 1073741824 バイト (1 GB) と同じです。アプライヤスレッドは `slave_max_allowed_packet` より大きいメッセージフラグメントを処理できないため、`group_replication_communication_max_message_size` の設定は `slave_max_allowed_packet` の設定より小さくする必要があります。断片化をオフにするには、`group_replication_communication_max_message_size` にゼロ値を指定します。

レプリケーショングループのメンバーが断片化を使用するには、グループ通信プロトコルのバージョンが MySQL 8.0.16 以上である必要があります。`group_replication_get_communication_protocol()` UDF を使用して、グループ通信プロトコルのバージョンを表示します。下位バージョンが使用されている場合、グループメンバーはメッセージを断片化しません。すべてのグループメンバーがサポートしている場合は、`group_replication_set_communication_protocol()` UDF を使用して、グループ通信プロトコルを上位バージョンに設定できます。詳細は、[セクション18.4.1.4「グループ通信プロトコルバージョンの設定」](#)を参照してください。

- [group_replication_components_stop_timeout](#)

コマンド行形式	<code>--group-replication-components-stop-timeout=#</code>
システム変数	<code>group_replication_components_stop_timeout</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	31536000
最小値	2
最大値	31536000

このシステム変数の値は、Group Replication の実行中に変更できますが、変更はグループメンバーで Group Replication を停止して再起動した後にのみ有効になります。

`group_replication_components_stop_timeout` では、グループレプリケーションが停止時に各コンポーネントを待機するタイムアウトを秒単位で指定します。

- [group_replication_compression_threshold](#)

コマンド行形式	<code>--group-replication-compression-threshold=#</code>
システム変数	<code>group_replication_compression_threshold</code>
スコープ	グローバル
動的	はい

SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1000000
最小値	0
最大値	4294967295

このシステム変数は、すべてのグループメンバーで同じ値を持つ必要があります。このシステム変数の値は、Group Replication の実行中に変更できます。変更は、メンバーでグループレプリケーションを停止して再起動した後、各グループメンバーで有効になります。このプロセス中、システム変数の値はグループメンバー間で異なってもかまいませんが、メッセージ配信の効率はすべてのメンバーで同じではありません。

[group_replication_compression_threshold](#) では、グループメンバー間で送信されるメッセージに圧縮が適用されるしきい値をバイト単位で指定します。このシステム変数をゼロに設定すると、圧縮は無効になります。

グループレプリケーションでは、LZ4 圧縮アルゴリズムを使用して、グループで送信されるメッセージを圧縮します。LZ4 圧縮アルゴリズムでサポートされる最大入力サイズは 2113929216 バイトです。この制限は、XCom で受け入れられる最大メッセージサイズと一致する、[group_replication_compression_threshold](#) システム変数の最大可能値を下回っています。LZ4 圧縮アルゴリズムでは、メッセージ圧縮が有効な場合、このサイズを超えるトランザクションはコミットできないため、[group_replication_compression_threshold](#) に 2113929216 バイトを超える値を設定しないでください。

詳細は、[セクション18.6.3「メッセージ圧縮」](#)を参照してください。

- [group_replication_consistency](#)

コマンド行形式	--group-replication-consistency=value
導入	8.0.14
システム変数	group_replication_consistency
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	EVENTUAL
有効な値	EVENTUAL BEFORE_ON_PRIMARY_FAILOVER BEFORE AFTER BEFORE_AND_AFTER

このシステム変数の値は、Group Replication の実行中に変更できます。[group_replication_consistency](#) は Group Replication プラグイン固有の変数ではなくサーバーシステム変数であるため、変更を有効にするために Group Replication を再起動する必要はありません。システム変数のセッション値の変更はただちに有効になり、グローバル値の変更は変更後に開始される新しいセッションに対して有効になります。このシステム変数のグローバル設定を変更するには、[GROUP_REPLICATION_ADMIN](#) 権限が必要です。

[group_replication_consistency](#) は、グループが提供するトランザクションの一貫性保証を制御します。一貫性はグローバルに構成することも、トランザクションごとに構成することもできます。[group_replication_consistency](#) は、単一のプライマリグループで新しく選択されたプライマリによって使用されるフェンシングメカニズムも構成します。変数の効果は、読み取り専用 (RO) トランザクションと読み取り/書き込み (RW) トランザクションの両方で考慮する

必要があります。次のリストは、トランザクションの一貫性保証を高めるために、この変数に指定できる値を示しています:

- [EVENTUAL](#)

RO トランザクションと RW トランザクションはどちらも、実行前に先行するトランザクションが適用されるのを待機しません。これは、この変数が追加される前の Group Replication の動作です。RW トランザクションは、他のメンバーがトランザクションを適用するのを待機しません。つまり、あるメンバーでトランザクションを外部化してから別のメンバーに外部化できます。つまり、プライマリフェイルオーバーが発生した場合、新しいプライマリは、前のプライマリトランザクションがすべて適用される前に新しい RO および RW トランザクションを受け入れることができます。RO トランザクションは古い値になる可能性があり、RW トランザクションは競合のためロールバックする可能性があります。

- [BEFORE_ON_PRIMARY_FAILOVER](#)

古いプライマリからバックログを適用している、新しく選択されたプライマリを持つ新しい RO または RW トランザクションは、バックログが適用されるまで保持されます (適用されません)。これにより、プライマリフェイルオーバーが意図的に発生したかどうかにかかわらず、クライアントには常にプライマリの最新の値が表示されます。これにより一貫性が保証されますが、バックログが適用されている場合、クライアントは遅延を処理する必要があります。通常、この遅延は最小限に抑える必要がありますが、バックログのサイズによって異なります。

- [BEFORE](#)

RW トランザクションは、先行するすべてのトランザクションが完了するまで待機してから適用されます。RO トランザクションは、先行するすべてのトランザクションが完了するまで待機してから実行されます。これにより、トランザクションのレイテンシにのみ影響を与えることで、このトランザクションが最新の値を読み取るようになります。これにより、RO トランザクションでのみ同期が使用されるようになるため、すべての RW トランザクションでの同期のオーバーヘッドが削減されます。この一貫性レベルには、[BEFORE_ON_PRIMARY_FAILOVER](#) によって提供される一貫性保証も含まれます。

- [AFTER](#)

RW トランザクションは、その変更が他のすべてのメンバーに適用されるまで待機します。この値は RO トランザクションには影響しません。このモードでは、トランザクションがローカルメンバーでコミットされたときに、後続のトランザクションが書き込み値またはグループメンバーのより新しい値を読み取ることが保証されます。このモードは、主に RO 操作に使用されるグループとともに使用して、適用された RW トランザクションがコミット後のすべての場所に確実に適用されるようにします。これは、後続の読取りで最新の書き込みを含む最新のデータがフェッチされるようにするために、アプリケーションで使用できます。これにより、RW トランザクションでのみ同期が使用されるようになるため、RO トランザクションごとの同期のオーバーヘッドが削減されます。この一貫性レベルには、[BEFORE_ON_PRIMARY_FAILOVER](#) によって提供される一貫性保証も含まれます。

- [BEFORE_AND_AFTER](#)

RW トランザクションは、1) 前のすべてのトランザクションが適用されるまで待機し、2) 変更が他のメンバーに適用されるまで待機します。RO トランザクションは、先行するすべてのトランザクションが完了するまで待機してから実行されます。この一貫性レベルには、[BEFORE_ON_PRIMARY_FAILOVER](#) によって提供される一貫性保証も含まれます。

詳細は、[セクション18.4.2「トランザクション一貫性保証」](#)を参照してください。

- [group_replication_enforce_update_everywhere_checks](#)

コマンド行形式	<code>--group-replication-enforce-update-everywhere-checks[={OFF ON}]</code>
システム変数	<code>group_replication_enforce_update_everywhere_checks</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ

型	Boolean
デフォルト値	OFF

このシステム変数は、グループ全体の構成設定です。値の変更を有効にするには、すべてのグループメンバーで同じ値を持つ必要があり、グループレプリケーションの実行中は変更できず、グループ (`group_replication_bootstrap_group=ON` を使用したサーバーによるブートストラップ) を完全に再起動する必要があります。MySQL 8.0.16 から、`group_replication_switch_to_single_primary_mode()` および `group_replication_switch_to_multi_primary_mode()` UDF を使用して、グループの実行中にこのシステム変数の値を変更できます。詳細は、[セクション18.4.1.2「グループモードの変更」](#)を参照してください。

`group_replication_enforce_update_everywhere_checks` は、マルチプライマリ更新の厳密な整合性チェックをどこでも有効または無効にします。デフォルトでは、チェックは無効になっています。シングルプライマリモードでは、すべてのグループメンバーでこのオプションを無効にする必要があります。マルチプライマリモードでは、このオプションを有効にすると、ステートメントは次のようにチェックされ、マルチプライマリモードと互換性があることが確認されます:

- トランザクションが `SERIALIZABLE` 分離レベルで実行される場合、トランザクション自体をグループと同期するとコミットは失敗します。
- カスケード制約を持つ外部キーを持つテーブルに対してトランザクションが実行される場合、トランザクションはグループとの同期時にコミットに失敗します。
- `group_replication_exit_state_action`

コマンド行形式	<code>--group-replication-exit-state-action=value</code>
導入	8.0.12
システム変数	<code>group_replication_exit_state_action</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値 (≥ 8.0.16)	<code>READ_ONLY</code>
デフォルト値 (≥ 8.0.12, ≤ 8.0.15)	<code>ABORT_SERVER</code>
有効な値 (≥ 8.0.18)	<code>ABORT_SERVER</code> <code>OFFLINE_MODE</code> <code>READ_ONLY</code>
有効な値 (≥ 8.0.12, ≤ 8.0.17)	<code>ABORT_SERVER</code> <code>READ_ONLY</code>

このシステム変数の値は、Group Replication の実行中に変更でき、変更はただちに有効になります。動作が必要であることを意味する問題が発生すると、システム変数 `current` 値が読み取られます。

`group_replication_exit_state_action` では、たとえばアプライアエラーが発生した後、大部分が失われた場合、または疑わしいタイムアウトのためにグループの別のメンバーがグループを明示する場合などに、このサーバーインスタンスが意図せずグループを離れたときのグループレプリケーションの動作を構成します。大部分が失われた場合にメンバーがグループを離れるタイムアウト期間は `group_replication_unreachable_majority_timeout` システム変数によって設定され、疑わしい場合のタイムアウト期間は `group_replication_member_expel_timeout` システム変数によって設定されます。削除されたグループメンバーは、グループに再接続するまで削除されたことを認識しないため、指定されたアクションが実行されるのは、メンバーが再接続を管理している場合、またはメンバーがそれ自体で疑わしいことを引き起こして削除した場合のみです。

疑わしいタイムアウトまたは大部分の損失が原因でグループメンバーが削除された場合、メンバーの `group_replication_autorejoin_tries` システム変数が自動再結合の試行回数を指定するように設定されていると、最初

にスーパー読み専用モードで指定された回数試行し、次に `group_replication_exit_state_action` で指定されたアクションに従います。アプライヤエラーの場合、自動再結合はリカバリできないため試行されません。

`group_replication_exit_state_action` が `READ_ONLY` に設定されている場合、メンバーが誤ってグループを終了するか、自動再結合の試行を使い果たすと、インスタンスは MySQL をスーパー読み専用モードに切り替えます (システム変数 `super_read_only` を `ON` に設定します)。 `READ_ONLY` 終了アクションは、システム変数が導入される前の MySQL 8.0 リリースの動作であり、MySQL 8.0.16 から再度デフォルトになりました。

`group_replication_exit_state_action` が `OFFLINE_MODE` に設定されている場合、メンバーが誤ってグループを終了するか、自動再結合の試行を使い果たすと、インスタンスは MySQL をオフラインモードに切り替えます (システム変数 `offline_mode` を `ON` に設定します)。このモードでは、接続クライアントユーザーは次のリクエストで切断され、`CONNECTION_ADMIN` 権限 (または非推奨の `SUPER` 権限) を持つクライアントユーザーを除き、接続は受け入れられなくなります。Group Replication は、システム変数 `super_read_only` を `ON` に設定することもできるため、クライアントが `CONNECTION_ADMIN` または `SUPER` 権限で接続されていても更新を行うことはできません。 `OFFLINE_MODE` 終了アクションは、MySQL 8.0.18 から使用できます。

`group_replication_exit_state_action` が `ABORT_SERVER` に設定されている場合、メンバーが意図せずグループを終了するか、自動再結合の試行を使い果たすと、インスタンスは MySQL を停止します。この設定は、システム変数が MySQL 8.0.15 に追加されたときの MySQL 8.0.12 のデフォルトでした。

重要

メンバーがグループに正常に参加する前に障害が発生した場合、指定された終了アクション取得されません。これは、ローカル構成チェック中に障害が発生した場合、または参加メンバーの構成とグループの構成が一致しない場合です。このような状況では、`super_read_only` システム変数は元の値のまま、接続は引き続き受け入れられ、サーバーは MySQL を停止しません。したがって、Group Replication が起動しなかったときにサーバーが更新を受け入れられないようにするには、起動時に `super_read_only=ON` をサーバー構成ファイルに設定することをお勧めします。これは、正常に起動された後、プライマリメンバー上の `OFF` にグループレプリケーションが変更されます。この保護策は、サーバーがサーバー起動時に Group Replication (`group_replication_start_on_boot=ON`) を起動するように構成されている場合に特に重要ですが、`START GROUP_REPLICATION` コマンドを使用して Group Replication を手動で起動する場合にも役立ちます。

このオプションの使用方法および exit アクションが実行される状況の完全なリストの詳細は、[セクション 18.6.6.4「終了処理」](#) を参照してください。

- `group_replication_flow_control_applier_threshold`

コマンド行形式	<code>--group-replication-flow-control-applier-threshold=#</code>
システム変数	<code>group_replication_flow_control_applier_threshold</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	25000
最小値	0
最大値	2147483647

このシステム変数の値は、Group Replication の実行中に変更でき、変更はただちに有効になります。

`group_replication_flow_control_applier_threshold` では、フロー制御をトリガーするアプライヤキュー内の待機中トランザクションの数を指定します。

- `group_replication_flow_control_certifier_threshold`

コマンド行形式	<code>--group-replication-flow-control-certifier-threshold=#</code>
---------	---

システム変数	group_replication_flow_control_certifier_threshold
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	25000
最小値	0
最大値	2147483647

このシステム変数の値は、Group Replication の実行中に変更でき、変更はただちに有効になります。

[group_replication_flow_control_certifier_threshold](#) では、フロー制御をトリガーする証明者キュー内の待機中トランザクションの数を指定します。

- [group_replication_flow_control_hold_percent](#)

コマンド行形式	<code>--group-replication-flow-control-hold-percent=#</code>
システム変数	group_replication_flow_control_hold_percent
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	10
最小値	0
最大値	100

このシステム変数の値は、Group Replication の実行中に変更でき、変更はただちに有効になります。

[group_replication_flow_control_hold_percent](#) では、フロー制御下のクラスタがバックログで捕捉できるように、未使用のままになるグループ割当ての割合を定義します。値 0 は、作業バックログを捕捉するために割当て制限の一部が予約されていないことを意味します。

- [group_replication_flow_control_max_commit_quota](#)

コマンド行形式	<code>--group-replication-flow-control-max-commit-quota=#</code>
システム変数	group_replication_flow_control_max_commit_quota
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	2147483647

このシステム変数の値は、Group Replication の実行中に変更でき、変更はただちに有効になります。

[group_replication_flow_control_max_commit_quota](#) では、フロー制御が有効になっている間、グループの最大フロー制御割当て制限または任意の期間の最大使用可能割当て制限が定義されます。値 0 は、最大割当て制限が設定されていないことを意味します。このシステム変数の値は、[group_replication_flow_control_min_quota](#) および [group_replication_flow_control_min_recovery_quota](#) より小さくできません。

- group_replication_flow_control_member_quota_percent

コマンド行形式	<code>--group-replication-flow-control-member-quota-percent=#</code>
システム変数	<code>group_replication_flow_control_member_quota_percent</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	100

このシステム変数の値は、Group Replication の実行中に変更でき、変更はただちに有効になります。

`group_replication_flow_control_member_quota_percent` では、割当て制限の計算時にメンバーが使用可能とみなす必要がある割当て制限の割合が定義されます。値 0 は、割当て制限を最後の期間のライターであったメンバー間で均等に分割する必要があることを意味します。

- group_replication_flow_control_min_quota

コマンド行形式	<code>--group-replication-flow-control-min-quota=#</code>
システム変数	<code>group_replication_flow_control_min_quota</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	2147483647

このシステム変数の値は、Group Replication の実行中に変更でき、変更はただちに有効になります。

`group_replication_flow_control_min_quota` は、最後の期間に実行された計算済の最小目標とは関係なく、メンバーに割り当てることができる最小フロー制御目標を制御します。値 0 は、最小割当て制限がないことを意味します。このシステム変数の値は `group_replication_flow_control_max_commit_quota` より大きくできません。

- group_replication_flow_control_min_recovery_quota

コマンド行形式	<code>--group-replication-flow-control-min-recovery-quota=#</code>
システム変数	<code>group_replication_flow_control_min_recovery_quota</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0

最大値	2147483647
-----	------------

このシステム変数の値は、Group Replication の実行中に変更でき、変更はただちに有効になります。

[group_replication_flow_control_min_recovery_quota](#) は、最後の期間に実行された計算済の最小割当て制限とは関係なく、グループ内の別のリカバリメンバーのためにメンバーに割り当てることができる最小割当て制限を制御します。値 0 は、最小割当て制限がないことを意味します。このシステム変数の値は [group_replication_flow_control_max_commit_quota](#) より大きくできません。

- [group_replication_flow_control_mode](#)

コマンド行形式	<code>--group-replication-flow-control-mode=value</code>
システム変数	group_replication_flow_control_mode
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	QUOTA
有効な値	DISABLED QUOTA

このシステム変数の値は、Group Replication の実行中に変更でき、変更はただちに有効になります。

[group_replication_flow_control_mode](#) では、フロー制御に使用されるモードを指定します。

- [group_replication_flow_control_period](#)

コマンド行形式	<code>--group-replication-flow-control-period=#</code>
システム変数	group_replication_flow_control_period
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1
最小値	1
最大値	60

このシステム変数の値は、Group Replication の実行中に変更でき、変更はただちに有効になります。

[group_replication_flow_control_period](#) では、フロー制御メッセージが送信されてフロー制御管理タスクが実行されるまで待機する秒数を定義します。

- [group_replication_flow_control_release_percent](#)

コマンド行形式	<code>--group-replication-flow-control-release-percent=#</code>
システム変数	group_replication_flow_control_release_percent
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	50

最小値	0
最大値	1000

このシステム変数の値は、Group Replication の実行中に変更でき、変更はただちに有効になります。

`group_replication_flow_control_release_percent` では、フロー制御でライターメンバーを制限する必要がなくなった場合に、グループ割当て制限を解放する方法を定義します。この割合は、フロー制御期間ごとの割当て制限の増加です。値 0 は、フロー制御しきい値が制限内になると、割当て制限が単一のフロー制御反復で解放されることを意味します。範囲を使用すると、割当て制限を現在の割当て制限の最大 10 倍までリリースできます。これにより、主にフロー制御期間が大きく、割当て制限が非常に小さい場合に、より高度な適応が可能になります。

- `group_replication_force_members`

コマンド行形式	<code>--group-replication-force-members=value</code>
システム変数	<code>group_replication_force_members</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列

このシステム変数は、新しいグループメンバーシップを強制するために使用されます。このシステム変数の値は、Group Replication の実行中に変更でき、変更はただちに有効になります。システム変数の値を設定する必要があるのは、グループに残すグループメンバーのいずれかのみです。新しいグループメンバーシップを強制的に実行する必要がある状況と、このシステム変数を使用する際に従う手順の詳細は、[セクション 18.4.4 「ネットワークパーティション化」](#) を参照してください。

`group_replication_force_members` では、ピアアドレスのリストをカンマ区切りリスト (`host1:port1, host2:port2` など) として指定します。リストに含まれていない既存のメンバーは、グループの新しいビューを受信せず、ブロックされます。メンバーとして続行する既存のメンバーごとに、各メンバーの `group_replication_local_address` システム変数で指定されているように、IP アドレスまたはホスト名とポートを含める必要があります。IPv6 アドレスは大カッコで囲んで指定する必要があります。例:

```
"198.51.100.44:33061,[2001:db8:85a3:8d3:1319:8a2e:370:7348]:33061,example.org:33061"
```

Group Replication (XCom) のグループ通信エンジンは、指定された IP アドレスが有効な形式であることをチェックし、現在アクセスできないグループメンバーが含まれていないことをチェックします。それ以外の場合、新しい構成は検証されないため、グループのアクセス可能なメンバーであるオンラインサーバーのみを含めるように注意する必要があります。リスト内の不正な値または無効なホスト名があると、グループが無効な構成でブロックされる可能性があります。

新しいメンバーシップ構成を強制的に実行する前に、除外するサーバーが停止していることを確認することが重要です。そうでない場合は、停止してから続行します。まだオンラインのグループメンバーは新しい構成を自動的に形成でき、これがすでに行われている場合は、さらに新しい構成を強制すると、グループに人工的なスプリットブレイン状況が作成される可能性があります。

`group_replication_force_members` システム変数を使用して新しいグループメンバーシップを強制し、グループのブロックを解除した後、必ずシステム変数をクリアしてください。`START GROUP_REPLICATION` ステートメントを発行するには、`group_replication_force_members` が空である必要があります。

- `group_replication_group_name`

コマンド行形式	<code>--group-replication-group-name=value</code>
システム変数	<code>group_replication_group_name</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ

型	文字列
---	-----

このシステム変数の値は、Group Replication の実行中に変更できますが、変更はグループメンバーで Group Replication を停止して再起動した後にのみ有効になります。

`group_replication_group_name` は、このサーバーインスタンスが属するグループの名前を指定します。これは有効な UUID である必要があります。この UUID は、バイナリログ内のグループレプリケーショントランザクションの GTID を設定するとき内部的に使用されます。

重要

一意の UUID を使用する必要があります。

- `group_replication_group_seeds`

コマンド行形式	<code>--group-replication-group-seeds=value</code>
システム変数	<code>group_replication_group_seeds</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列

このシステム変数の値は、Group Replication の実行中に変更できますが、変更はグループメンバーで Group Replication を停止して再起動した後にのみ有効になります。

`group_replication_group_seeds` は、参加メンバーが現在のすべてのグループメンバーの詳細を取得するために接続できるグループメンバーのリストです。参加メンバーは、これらの詳細を使用してグループメンバーを選択して接続し、グループとの同期に必要なデータを取得します。このリストは、シードメンバーの `group_replication_local_address` システム変数 (MySQL Server `hostname` および `port` システム変数で指定されたシードメンバーの SQL クライアント接続ではなく) で構成されている、含まれる各シードメンバーの単一の内部ネットワークアドレスまたはホスト名で構成されます。シードメンバーのアドレスは、`host1:port1`、`host2:port2` などのカンマ区切りリストとして指定されます。IPv6 アドレスは大カッコで囲んで指定する必要があります。例:

```
group_replication_group_seeds="198.51.100.44:33061,[2001:db8:85a3:8d3:1319:8a2e:370:7348]:33061,example.org:33061"
```

この変数に指定する値は、`START GROUP REPLICATION` ステートメントが発行され、Group Communication System (GCS) が使用可能になるまで検証されないことに注意してください。

通常、このリストはグループのすべてのメンバーで構成されますが、シードするグループメンバーのサブセットを選択できます。リストには、少なくとも 1 つの有効なメンバーアドレスが含まれている必要があります。各アドレスは、Group Replication の開始時に検証されます。リストに有効なメンバーアドレスが含まれていない場合、`START GROUP REPLICATION` の発行は失敗します。

サーバーは、レプリケーショングループに参加するときに、その `group_replication_group_seeds` システム変数にリストされている最初のシードメンバーに接続しようとします。接続が拒否された場合、参加メンバーはリスト内の他の各シードメンバーに順番に接続しようとします。結合メンバーがシードメンバーに接続しても、結果としてレプリケーショングループに追加されない場合 (たとえば、シードメンバーの `allowlist` に結合メンバーアドレスがなく、接続を閉じているため)、結合メンバーはリスト内の残りのシードメンバーを順番に試行し続けます。

参加メンバーは、シードメンバーが `group_replication_group_seeds` オプションで通知するプロトコル (IPv4 または IPv6) と同じプロトコルを使用してシードメンバーと通信する必要があります。グループレプリケーションの IP アドレス権限のために、シードメンバーの許可リストには、シードメンバーが提供するプロトコルの参加メンバーの IP アドレス、またはそのプロトコルのアドレスに解決されるホスト名が含まれている必要があります。このアドレスのプロトコルがシードメンバーに通知されたプロトコルと一致しない場合は、参加メンバー `group_replication_local_address` に加えて、このアドレスまたはホスト名を設定して許可する必要があります。参加メンバーに適切なプロトコルの許可されたアドレスがない場合、その接続試行は拒否されます。詳細は、[セクション 18.5.1 「グループレプリケーション IP アドレスの権限」](#) を参照してください。

- group_replication_gtid_assignment_block_size

コマンド行形式	--group-replication-gtid-assignment-block-size=#
システム変数	group_replication_gtid_assignment_block_size
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1000000
最小値	1
最大値 (64 ビットプラットフォーム)	9223372036854775807
最大値 (32 ビットプラットフォーム)	4294967295

このシステム変数は、グループ全体の構成設定です。値の変更を有効にするには、すべてのグループメンバーで同じ値を持つ必要があり、グループレプリケーションの実行中は変更できず、グループ (group_replication_bootstrap_group=ON を使用したサーバーによるブートストラップ) を完全に再起動する必要があります。

group_replication_gtid_assignment_block_size では、各グループメンバー用に予約されている連続 GTID の数を指定します。各メンバーは独自のブロックを消費し、必要に応じてより多くのブロックを予約します。

- group_replication_ip_allowlist

コマンド行形式	--group-replication-ip-allowlist=value
導入	8.0.22
システム変数	group_replication_ip_allowlist
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	AUTOMATIC

group_replication_ip_allowlist は、group_replication_ip_whitelist のかわりに MySQL 8.0.22 から使用できます。グループレプリケーションの実行中は、このシステム変数の値を変更できません。

group_replication_ip_allowlist は、グループへの接続を許可するホストを指定します。group_replication_local_address の各グループメンバーに指定するアドレスは、レプリケーショングループ内の他のサーバーで許可されている必要があります。この変数に指定する値は、START GROUP_REPLICATION ステートメントが発行され、Group Communication System (GCS) が使用可能になるまで検証されないことに注意してください。

デフォルトでは、このシステム変数は、ホスト上でアクティブなプライベートサブネットワークからの接続を許可する AUTOMATIC に設定されています。Group Replication (XCom) のグループ通信エンジンは、ホスト上のアクティブなインターフェースを自動的にスキャンし、プライベートサブネットワーク上のアドレスを持つインターフェースを識別します。IPv4 および (MySQL 8.0.14 から) IPv6 のこれらのアドレスと localhost IP アドレスを使用して、Group Replication 許可リストを作成します。アドレスが自動的に許可される範囲のリストについては、[セクション 18.5.1 「グループレプリケーション IP アドレスの権限」](#) を参照してください。

プライベートアドレスの自動許可リストは、プライベートネットワーク外のサーバーからの接続には使用できません。異なるマシン上にあるサーバーインスタンス間のグループレプリケーション接続の場合、パブリック IP アドレスを指定し、明示的な許可リストとして指定する必要があります。許可リストにエントリを指定した場合、プラ

イベートアドレスは自動的に追加されないため、これらのいずれかを使用する場合は、明示的に指定する必要があります。localhost IP アドレスは自動的に追加されます。

`group_replication_ip_allowlist` オプションの値として、次の任意の組合せを指定できます:

- IPv4 アドレス (198.51.100.44 など)
- CIDR 表記を使用した IPv4 アドレス (192.0.2.21/24 など)
- MySQL 8.0.14 からの IPv6 アドレス (2001:db8:85a3:8d3:1319:8a2e:370:7348 など)
- MySQL 8.0.14 からの CIDR 表記の IPv6 アドレス (2001:db8:85a3:8d3::/64 など)
- ホスト名 (example.org など)
- CIDR 表記法を使用したホスト名 (www.example.com/24 など)

MySQL 8.0.14 より前では、ホスト名は IPv4 アドレスにのみ解決できました。MySQL 8.0.14 から、ホスト名は IPv4 アドレス、IPv6 アドレス、またはその両方に解決できます。ホスト名が IPv4 アドレスと IPv6 アドレスの両方に解決される場合、IPv4 アドレスは常にグループレプリケーション接続に使用されます。CIDR 表記をホスト名または IP アドレスと組み合わせて使用すると、特定のネットワーク接頭辞を持つ IP アドレスのブロックを許可できますが、指定したサブネット内のすべての IP アドレスが制御下にあることを確認してください。

allowlist の各エントリはカンマで区切る必要があります。例:

```
"192.0.2.21/24,198.51.100.44,203.0.113.0/24,2001:db8:85a3:8d3:1319:8a2e:370:7348,example.org,www.example.com/24"
```

結合メンバーに IPv4 `group_replication_local_address` がある場合、またはその逆の場合に、グループのシードメンバーのいずれかが IPv6 アドレスとともに `group_replication_group_seeds` オプションにリストされている場合は、シードメンバーによって提供されるプロトコル (またはそのプロトコルのアドレスに解決されるホスト名) の結合メンバーの代替アドレスも設定して許可する必要があります。詳細は、[セクション18.5.1「グループレプリケーション IP アドレスの権限」](#)を参照してください。

たとえば、異なるサブネットを分離しておくために、セキュリティ要件に応じて異なるグループメンバーに異なる許可リストを構成できます。ただし、これにより、グループの再構成時に問題が発生する可能性があります。それ以外の場合に特定のセキュリティ要件がない場合は、グループのすべてのメンバーで同じ許可リストを使用します。詳細は、[セクション18.5.1「グループレプリケーション IP アドレスの権限」](#)を参照してください。

ホスト名の場合、名前解決は、別のサーバーによって接続リクエストが行われた場合にのみ行われます。解決できないホスト名は許可リストの検証で考慮されず、警告メッセージがエラーログに書き込まれます。前方確認の逆引き DNS (FCrDNS) 検証は、解決されたホスト名に対して実行されます。

警告

ホスト名は本質的に許可リストの IP アドレスより安全性が低くなります。FCrDNS の検証は適切なレベルの保護を提供しますが、特定のタイプの攻撃によって危険にさらされる可能性があります。厳密に必要な場合にのみ許可リストにホスト名を指定し、名前解決に使用されるすべてのコンポーネント (DNS サーバーなど) が制御下に保持されていることを確認します。外部コンポーネントを使用しないように、hosts ファイルを使用して名前解決をローカルに実装することもできます。

- `group_replication_ip_whitelist`

コマンド行形式	<code>--group-replication-ip-whitelist=value</code>
非推奨	8.0.22
システム変数	<code>group_replication_ip_whitelist</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列

デフォルト値	AUTOMATIC
--------	-----------

MySQL 8.0.22 からは、`group_replication_ip_whitelist` は非推奨であり、`group_replication_ip_allowlist` を使用して置き換えられます。どちらのシステム変数の場合も、デフォルト値は `AUTOMATIC` です。いずれかのシステム変数がユーザー定義値に設定されていて、もう一方が設定されていない場合は、変更された値が使用されます。両方のシステム変数がユーザー定義の値に設定されている場合は、`group_replication_ip_allowlist` の値が使用されます。

新しいシステム変数は古いシステム変数と同じように機能しますが、用語のみが変更されています。`group_replication_ip_allowlist` の動作の説明は、古いシステム変数と新しいシステム変数の両方に適用されます。

- `group_replication_local_address`

コマンド行形式	<code>--group-replication-local-address=value</code>
システム変数	<code>group_replication_local_address</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列

このシステム変数の値は、Group Replication の実行中に変更できますが、変更はグループメンバーで Group Replication を停止して再起動した後にのみ有効になります。

`group_replication_local_address` は、メンバーが他のメンバーからの接続用に提供するネットワークアドレスを、`host:port` フォーマット文字列として指定して設定します。このアドレスは、リモート XCom インスタンス間の TCP 通信用のグループレプリケーション (XCom、Paxos バリエーション) のためにグループ通信エンジンによって使用されるため、グループのすべてのメンバーがアクセスできる必要があります。ローカルインスタンスとの通信は、共有メモリーを使用して入力チャネルを介して行われます。

警告

メンバーとの通信にこのアドレスを使用しないでください。これは、MySQL サーバーの SQL プロトコルのホストおよびポートではありません。

`group_replication_local_address` で指定するアドレスまたはホスト名は、グループレプリケーションによって、レプリケーショングループ内のグループメンバーの一意の識別子として使用されます。ホスト名または IP アドレスがすべて異なるかぎり、レプリケーショングループのすべてのメンバーに同じポートを使用でき、ポートがすべて異なるかぎり、すべてのメンバーに同じホスト名または IP アドレスを使用できます。`group_replication_local_address` の推奨ポートは 33061 です。この変数に指定した値は、`START GROUP_REPLICATION` ステートメントが発行され、Group Communication System (GCS) が使用可能になるまで検証されないことに注意してください。

`group_replication_local_address` によって構成されたネットワークアドレスは、すべてのグループメンバーが解決できる必要があります。たとえば、各サーバーインスタンスが固定ネットワークアドレスを持つ異なるマシン上にある場合、10.0.0.1 などのマシンの IP アドレスを使用できます。ホスト名を使用する場合は、完全修飾名を使用し、DNS、正しく構成された `/etc/hosts` ファイルまたはその他の名前解決プロセスを介して解決できることを確認する必要があります。MySQL 8.0.14 からは、IPv6 アドレス (またはそれらに解決されるホスト名) と IPv4 アドレスを使用できます。IPv6 アドレスは、ポート番号を区別するために大カッコで囲む必要があります。次に例を示します:

```
group_replication_local_address= "[2001:db8:85a3:8d3:1319:8a2e:370:7348]:33061"
```

サーバーインスタンスのグループレプリケーションのローカルアドレスとして指定されたホスト名が IPv4 アドレスと IPv6 アドレスの両方に解決される場合、IPv4 アドレスは常にグループレプリケーション接続に使用されます。IPv6 ネットワークおよび IPv4 を使用するメンバーと IPv6 を使用するメンバーが混在するレプリケーショングループに対するグループレプリケーションサポートの詳細は、[セクション 18.4.5 「IPv6 および IPv6 と IPv4 の混合グループのサポート」](#) を参照してください。

グループレプリケーションの IP アドレス権限のために、`group_replication_local_address` の各グループメンバーに指定するアドレスを、レプリケーショングループ内の他のサーバー上の `group_replication_ip_allowlist` (MySQL

8.0.22 から) または `group_replication_ip_whitelist` システム変数のリストに追加する必要があります。グループのシードメンバーのいずれかが IPv6 アドレスとともに `group_replication_group_seeds` オプションにリストされている場合このメンバーが IPv4 `group_replication_local_address` であれば (またはその逆の場合)、必要なプロトコル (またはそのプロトコルのアドレスに解決されるホスト名) に対してこのメンバーの代替アドレスを設定して許可する必要があります。詳細は、[セクション18.5.1「グループレプリケーション IP アドレスの権限」](#)を参照してください。

- `group_replication_member_expel_timeout`

コマンド行形式	<code>--group-replication-member-expel-timeout=#</code>
導入	8.0.13
システム変数	<code>group_replication_member_expel_timeout</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値 (≥ 8.0.21)	5
デフォルト値 (≤ 8.0.20)	0
最小値	0
最大値 (≥ 8.0.14)	3600
最大値 (≤ 8.0.13)	31536000

このシステム変数の値は、Group Replication の実行中に変更でき、変更はただちに有効になります。Group Replication がタイムアウトをチェックするたびに、システム変数の現在の値が読み取られます。グループのすべてのメンバーが同じ設定を持つことは必須ではありませんが、予期しない実行を回避するためにお勧めします。

`group_replication_member_expel_timeout` では、疑わしいグループを作成した後、メンバーが失敗した疑いがあるグループから追い出されるまでにグループレプリケーショングループメンバーが待機する期間を秒単位で指定します。疑いが作成される前の最初の 5 秒間の検出期間は、この時間の一部としてカウントされません。MySQL 8.0.20 まで、`group_replication_member_expel_timeout` の値はデフォルトで 0 に設定されます。これは、待機期間がなく、疑わしいメンバーが 5 秒間の検出期間の終了直後に実行できることを意味します。MySQL 8.0.21 からは、値はデフォルトで 5 に設定されます。つまり、疑わしいメンバーは、5 秒間の検出期間の 5 秒後に強制的に実行されます。

グループメンバーの `group_replication_member_expel_timeout` の値の変更は、そのグループメンバーの既存および将来の疑いに対してただちに有効になります。したがって、これを方法として使用すると、疑わしいメンバーを強制的にタイムアウトして削除し、グループ構成を変更できます。詳細は、[セクション18.6.6.1「Expel タイムアウト」](#)を参照してください。

`group_replication_member_expel_timeout` の値を大きくすると、低速または安定していないネットワークで不要な削除が発生しないようにしたり、一時的なネットワークの停止やマシンの速度低下が予想される場合に役立ちます。疑わしいメンバーがタイムアウトする前に再度アクティブになると、残りのグループメンバーによってバッファされたすべてのメッセージが適用され、オペレータの介入なしで **ONLINE** 状態になります。最大 3600 秒 (1 時間) までのタイムアウト値を指定できます。XCom メッセージキャッシュが、指定した期間内の予想されるメッセージ量と最初の 5 秒間の検出期間を含むのに十分な大きさであることを確認することが重要です。そうでない場合、メンバーは再接続できません。`group_replication_message_cache_size` システム変数を使用して、キャッシュサイズ制限を調整できます。詳細は、[セクション18.6.5「XCom キャッシュ管理」](#)を参照してください。

タイムアウトを超えた場合、疑わしいメンバーは疑わしいタイムアウトの直後に削除する責任があります。メンバーが通信を再開でき、削除されたビューを受信し、メンバーに自動再結合の試行回数を指定する `group_replication_autorejoin_tries` システム変数が設定されている場合、スーパー読み取り専用モードでは、指定された回数グループへの再参加を試行します。メンバーに自動再結合の試行が指定されていない場合、または指定され

た試行回数を使い果たした場合は、システム変数 `group_replication_exit_state_action` で指定されたアクションに従います。

`group_replication_member_expel_timeout` 設定の使用の詳細は、[セクション18.6.6.1「Expel タイムアウト」](#) を参照してください。このシステム変数が使用できない場合に不要な削除を回避するための代替軽減戦略については、[セクション18.9.2「グループレプリケーションの制限事項」](#) を参照してください。

- `group_replication_member_weight`

コマンド行形式	<code>--group-replication-member-weight=#</code>
システム変数	<code>group_replication_member_weight</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	50
最小値	0
最大値	100

このシステム変数の値は、Group Replication の実行中に変更でき、変更はただちに有効になります。システム変数の現在の値は、フェイルオーバー状況が発生したときに読み取られます。

`group_replication_member_weight` では、既存のプライマリが単一プライマリグループから離れる場合など、フェイルオーバー時にプライマリとして選択されるメンバーの可能性に影響を与えるためにメンバーに割り当てることができる重みの割合を指定します。プライマリのスケジュールされたメンテナンス中やフェイルオーバー時に特定のハードウェアが優先されるように、メンバーに数値の重みを割り当てて特定のメンバーが選択されるようにします。

メンバーが次のように構成されているグループの場合:

- `member-1`: `group_replication_member_weight=30, server_uuid=aaaa`
- `member-2`: `group_replication_member_weight=40, server_uuid=bbbb`
- `member-3`: `group_replication_member_weight=40, server_uuid=cccc`
- `member-4`: `group_replication_member_weight=40, server_uuid=dddd`

新しいプライマリの選択時に、前述のメンバーは `member-2`, `member-3`, `member-4` および `member-1` としてソートされます。これにより、フェイルオーバー時に新しいプライマリとして `member-2` が選択されます。詳細は、[セクション18.1.3.1「シングルプライマリモード」](#) を参照してください。

- `group_replication_message_cache_size`

コマンド行形式	<code>--group-replication-message-cache-size=#</code>
導入	8.0.16
システム変数	<code>group_replication_message_cache_size</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1073741824 (1 GB)
最小値 (64 ビットプラットフォーム, ≥ 8.0.21)	134217728 (128 MB)
最小値 (64 ビットプラットフォーム, ≤ 8.0.20)	1073741824 (1 GB)

最小値 (32 ビットプラットフォーム, ≥ 8.0.21)	134217728 (128 MB)
最小値 (32 ビットプラットフォーム, ≤ 8.0.20)	1073741824 (1 GB)
最大値 (64 ビットプラットフォーム)	18446744073709551615 (16 EiB)
最大値 (32 ビットプラットフォーム)	315360004294967295 (4 GB)

このシステム変数は、すべてのグループメンバーで同じ値を持つ必要があります。このシステム変数の値は、Group Replication の実行中に変更できます。変更は、メンバーでグループレプリケーションを停止して再起動した後、各グループメンバーで有効になります。このプロセス中、システム変数の値はグループメンバー間で異なってもかまいませんが、切断された場合にメンバーが再接続できないことがあります。

[group_replication_message_cache_size](#) は、グループレプリケーション (XCom) のグループ通信エンジンでメッセージキャッシュに使用可能なメモリの最大量を設定します。XCom メッセージキャッシュには、コンセンサスプロトコルの一部としてグループメンバー間で交換されるメッセージ (およびそのメタデータ) が保持されます。メッセージキャッシュは、他のグループメンバーと通信できなかった期間後にグループに再接続するメンバーによる、欠落したメッセージのリカバリに使用されます。

[group_replication_member_expel_timeout](#) システム変数は、メンバーが明示されるのではなくグループに戻るための最初の 5 秒間の検出期間に加えて許可される待機期間 (最大 1 時間) を決定します。XCom メッセージキャッシュのサイズは、メンバーが正常に戻るために必要なすべての欠落メッセージが含まれるように、この期間内の予想されるメッセージ量を参照して設定する必要があります。MySQL 8.0.20 までは、デフォルトは 5 秒の検出期間のみですが、MySQL 8.0.21 からは、合計 10 秒間、5 秒の検出期間後の待機期間は 5 秒です。

MySQL Server の他のキャッシュおよびオブジェクトプールのサイズを考慮して、選択したキャッシュサイズ制限に十分なメモリがシステムで使用可能であることを確認します。デフォルト設定は 1073741824 バイト (1 GB) です。最小設定は、MySQL 8.0.20 まで 1 GB です。MySQL 8.0.21 からは、最小設定は 134217728 バイト (128 MB) で、使用可能なメモリ量が制限されたホストへのデプロイメントが可能になり、グループメンバーの接続の一時的な損失の頻度と期間を最小限に抑えるための適切なネットワーク接続が可能になります。[group_replication_message_cache_size](#) を使用して設定された制限はキャッシュに格納されているデータにのみ適用され、キャッシュ構造には追加の 50 MB のメモリが必要です。

キャッシュサイズ制限は、実行時に動的に増減できます。キャッシュサイズ制限を小さくすると、XCom では、現在のサイズが制限を下回るまで、決定および配信された最も古いエントリが削除されます。Group Replication Group Communication System (GCS) は、現在アクセスできないメンバーによるリカバリに必要と思われるメッセージがメッセージキャッシュから削除されたときに、警告メッセージでアラートを生成します。メッセージキャッシュサイズのチューニングの詳細は、[セクション 18.6.5 「XCom キャッシュ管理」](#) を参照してください。

- [group_replication_poll_spin_loops](#)

コマンド行形式	--group-replication-poll-spin-loops=#
システム変数	group_replication_poll_spin_loops
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値 (64 ビットプラットフォーム)	18446744073709551615
最大値 (32 ビットプラットフォーム)	4294967295

このシステム変数の値は、Group Replication の実行中に変更できますが、変更はグループメンバーで Group Replication を停止して再起動した後にのみ有効になります。

[group_replication_poll_spin_loops](#) では、グループ通信スレッドが通信エンジン mutex の解放を待機する回数を指定します。この回数を超えると、スレッドは追加の着信ネットワークメッセージを待機します。

- [group_replication_recovery_complete_at](#)

コマンド行形式	<code>--group-replication-recovery-complete-at=value</code>
システム変数	<code>group_replication_recovery_complete_at</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	<code>TRANSACTIONS_APPLIED</code>
有効な値	<code>TRANSACTIONS_CERTIFIED</code> <code>TRANSACTIONS_APPLIED</code>

このシステム変数の値は、Group Replication の実行中に変更できますが、変更はグループメンバーで Group Replication を停止して再起動した後にのみ有効になります。

`group_replication_recovery_complete_at` では、既存のメンバーからの状態転送後にキャッシュされたトランザクションを処理する際に、分散リカバリプロセス中に適用されるポリシーを指定します。メンバーがグループ (`TRANSACTIONS_CERTIFIED`) に参加する前に見逃したすべてのトランザクションを受信して認証した後、または受信、認証および適用した後 (`TRANSACTIONS_APPLIED`) にのみ、メンバーをオンラインとしてマークするかどうかを選択できます。

- `group_replication_recovery_compression_algorithm`

コマンド行形式	<code>--group-replication-recovery-compression-algorithm=value</code>
導入	8.0.18
システム変数	<code>group_replication_recovery_compression_algorithm</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Set
デフォルト値	<code>uncompressed</code>
有効な値	<code>zlib</code> <code>zstd</code> <code>uncompressed</code>

このシステム変数の値は、Group Replication の実行中に変更できますが、変更はグループメンバーで Group Replication を停止して再起動した後にのみ有効になります。

`group_replication_recovery_compression_algorithm` では、ドナーバイナリログからの状態転送用のグループレプリケーション分散リカバリ接続に許可される圧縮アルゴリズムを指定します。使用可能なアルゴリズムは、`protocol_compression_algorithms` システム変数の場合と同じです。詳細は、[セクション4.2.8「接続圧縮制御」](#)を参照してください。

この設定は、クローニングをサポートするようにサーバーが設定されており ([セクション18.4.3.2「分散リカバリのためのクローニング」](#)を参照)、分散リカバリ中にリモートクローニング操作が使用されている場合は適用されません。この状態転送方法では、クローンプラグインの `clone_enable_compression` 設定が適用されます。

- `group_replication_recovery_get_public_key`

コマンド行形式	<code>--group-replication-recovery-get-public-key[={OFF ON}]</code>
システム変数	<code>group_replication_recovery_get_public_key</code>

スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

このシステム変数の値は、Group Replication の実行中に変更できますが、変更はグループメンバーで Group Replication を停止して再起動した後にのみ有効になります。

[group_replication_recovery_get_public_key](#) は、RSA キーペアベースのパスワード交換に必要な公開キーをソースからリクエストするかどうかを指定します。 [group_replication_recovery_public_key_path](#) が有効な公開キーファイルに設定されている場合は、[group_replication_recovery_get_public_key](#) よりも優先されます。この変数は、[group_replication_recovery](#) チャンネル ([group_replication_recovery_use_ssl=ON](#)) を介した分散リカバリに SSL を使用せず、グループレプリケーションのレプリケーションユーザーアカウントが [caching_sha2_password](#) プラグイン (MySQL 8.0 のデフォルト) で認証される場合に適用されます。詳細は、[キャッシュ SHA-2 認証プラグインを使用するレプリケーションユーザー](#)を参照してください。

- [group_replication_recovery_public_key_path](#)

コマンド行形式	<code>--group-replication-recovery-public-key-path=file_name</code>
システム変数	group_replication_recovery_public_key_path
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	NULL

このシステム変数の値は、Group Replication の実行中に変更できますが、変更はグループメンバーで Group Replication を停止して再起動した後にのみ有効になります。

[group_replication_recovery_public_key_path](#) では、RSA キーペアベースのパスワード交換のソースで必要な公開キーのレプリカ側コピーを含むファイルへのパス名を指定します。ファイルは PEM 形式である必要があります。 [group_replication_recovery_public_key_path](#) が有効な公開キーファイルに設定されている場合は、[group_replication_recovery_get_public_key](#) よりも優先されます。この変数は、[group_replication_recovery](#) チャンネルを介した分散リカバリに SSL を使用しておらず ([group_replication_recovery_use_ssl](#) が OFF に設定されている場合)、Group Replication のレプリケーションユーザーアカウントが [caching_sha2_password](#) プラグイン (MySQL 8.0 のデフォルト) または [sha256_password](#) プラグインで認証される場合に適用されます。([sha256_password](#) の場合、[group_replication_recovery_public_key_path](#) の設定は、MySQL が OpenSSL を使用して構築された場合にのみ適用されます。) 詳細は、[キャッシュ SHA-2 認証プラグインを使用するレプリケーションユーザー](#)を参照してください。

- [group_replication_recovery_reconnect_interval](#)

コマンド行形式	<code>--group-replication-recovery-reconnect-interval=#</code>
システム変数	group_replication_recovery_reconnect_interval
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	60
最小値	0

最大値	31536000
-----	----------

このシステム変数の値は、Group Replication の実行中に変更できますが、変更はグループメンバーで Group Replication を停止して再起動した後にのみ有効になります。

[group_replication_recovery_reconnect_interval](#) では、分散リカバリに適したドナーがグループに見つからなかった場合の再接続試行の間隔を秒単位で指定します。

- [group_replication_recovery_retry_count](#)

コマンド行形式	<code>--group-replication-recovery-retry-count=#</code>
システム変数	group_replication_recovery_retry_count
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	10
最小値	0
最大値	31536000

このシステム変数の値は、Group Replication の実行中に変更できますが、変更はグループメンバーで Group Replication を停止して再起動した後にのみ有効になります。

[group_replication_recovery_retry_count](#) は、参加しているメンバーが分散リカバリに使用可能なドナーへの接続を試みる回数を指定します。

- [group_replication_recovery_ssl_ca](#)

コマンド行形式	<code>--group-replication-recovery-ssl-ca=value</code>
システム変数	group_replication_recovery_ssl_ca
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列

このシステム変数の値は、Group Replication の実行中に変更できますが、変更はグループメンバーで Group Replication を停止して再起動した後にのみ有効になります。

[group_replication_recovery_ssl_ca](#) は、分散リカバリ接続用の信頼できる SSL 認証局のリストを含むファイルへのパスを指定します。分散リカバリのための SSL の構成の詳細は、[セクション18.5.2「Secure Socket Layer \(SSL\) を使用したグループ通信接続の保護」](#)を参照してください。

このサーバーがクローニングをサポートするように設定されていて ([セクション18.4.3.2「分散リカバリのためのクローニング」](#)を参照)、[group_replication_recovery_use_ssl](#) を ON に設定している場合、Group Replication はクローン SSL オプション `clone_ssl_ca` の設定を [group_replication_recovery_ssl_ca](#) の設定と一致するように自動的に構成します。

- [group_replication_recovery_ssl_capath](#)

コマンド行形式	<code>--group-replication-recovery-ssl-capath=value</code>
システム変数	group_replication_recovery_ssl_capath
スコープ	グローバル
動的	はい

SET_VAR ヒントの適用	いいえ
型	文字列

このシステム変数の値は、Group Replication の実行中に変更できますが、変更はグループメンバーで Group Replication を停止して再起動した後にのみ有効になります。

[group_replication_recovery_ssl_capath](#) は、分散リカバリ接続用の信頼できる SSL 認証局証明書を含むディレクトリへのパスを指定します。分散リカバリのための SSL の構成の詳細は、[セクション18.5.2 「Secure Socket Layer \(SSL\) を使用したグループ通信接続の保護」](#) を参照してください。

- [group_replication_recovery_ssl_cert](#)

コマンド行形式	<code>--group-replication-recovery-ssl-cert=value</code>
システム変数	group_replication_recovery_ssl_cert
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列

このシステム変数の値は、Group Replication の実行中に変更できますが、変更はグループメンバーで Group Replication を停止して再起動した後にのみ有効になります。

[group_replication_recovery_ssl_cert](#) では、分散リカバリ用のセキュアな接続を確立するために使用する SSL 証明書ファイルの名前を指定します。分散リカバリのための SSL の構成の詳細は、[セクション18.5.2 「Secure Socket Layer \(SSL\) を使用したグループ通信接続の保護」](#) を参照してください。

このサーバーがクローニングをサポートするように設定されていて ([セクション18.4.3.2 「分散リカバリのためのクローニング」](#) を参照)、[group_replication_recovery_use_ssl](#) を ON に設定している場合、Group Replication はクローン SSL オプション `clone_ssl_cert` の設定を [group_replication_recovery_ssl_cert](#) の設定と一致するように自動的に構成します。

- [group_replication_recovery_ssl_cipher](#)

コマンド行形式	<code>--group-replication-recovery-ssl-cipher=value</code>
システム変数	group_replication_recovery_ssl_cipher
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列

このシステム変数の値は、Group Replication の実行中に変更できますが、変更はグループメンバーで Group Replication を停止して再起動した後にのみ有効になります。

[group_replication_recovery_ssl_cipher](#) では、SSL 暗号化に許可される暗号のリストを指定します。分散リカバリのための SSL の構成の詳細は、[セクション18.5.2 「Secure Socket Layer \(SSL\) を使用したグループ通信接続の保護」](#) を参照してください。

- [group_replication_recovery_ssl_crl](#)

コマンド行形式	<code>--group-replication-recovery-ssl-crl=value</code>
システム変数	group_replication_recovery_ssl_crl
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ

型	ファイル名
---	-------

このシステム変数の値は、Group Replication の実行中に変更できますが、変更はグループメンバーで Group Replication を停止して再起動した後にのみ有効になります。

[group_replication_recovery_ssl_crl](#) は、証明書失効リストを含むファイルを含むディレクトリへのパスを指定します。分散リカバリのための SSL の構成の詳細は、[セクション18.5.2「Secure Socket Layer \(SSL\) を使用したグループ通信接続の保護」](#) を参照してください。

- [group_replication_recovery_ssl_crlpath](#)

コマンド行形式	<code>--group-replication-recovery-ssl-crlpath=value</code>
システム変数	group_replication_recovery_ssl_crlpath
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	ディレクトリ名

このシステム変数の値は、Group Replication の実行中に変更できますが、変更はグループメンバーで Group Replication を停止して再起動した後にのみ有効になります。

[group_replication_recovery_ssl_crlpath](#) は、証明書失効リストを含むファイルを含むディレクトリへのパスを指定します。分散リカバリのための SSL の構成の詳細は、[セクション18.5.2「Secure Socket Layer \(SSL\) を使用したグループ通信接続の保護」](#) を参照してください。

- [group_replication_recovery_ssl_key](#)

コマンド行形式	<code>--group-replication-recovery-ssl-key=value</code>
システム変数	group_replication_recovery_ssl_key
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列

このシステム変数の値は、Group Replication の実行中に変更できますが、変更はグループメンバーで Group Replication を停止して再起動した後にのみ有効になります。

[group_replication_recovery_ssl_key](#) では、セキュアな接続の確立に使用する SSL キーファイルの名前を指定します。分散リカバリのための SSL の構成の詳細は、[セクション18.5.2「Secure Socket Layer \(SSL\) を使用したグループ通信接続の保護」](#) を参照してください。

このサーバーがクローニングをサポートするように設定されていて ([セクション18.4.3.2「分散リカバリのためのクローニング」](#) を参照)、[group_replication_recovery_use_ssl](#) を ON に設定している場合、Group Replication はクローン SSL オプション `clone_ssl_key` の設定を [group_replication_recovery_ssl_key](#) の設定と一致するように自動的に構成します。

- [group_replication_recovery_ssl_verify_server_cert](#)

コマンド行形式	<code>--group-replication-recovery-ssl-verify-server-cert[={OFF ON}]</code>
システム変数	group_replication_recovery_ssl_verify_server_cert
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ

型	Boolean
デフォルト値	OFF

このシステム変数の値は、Group Replication の実行中に変更できますが、変更はグループメンバーで Group Replication を停止して再起動した後にのみ有効になります。

[group_replication_recovery_ssl_verify_server_cert](#) では、分散リカバリ接続でドナーによって送信された証明書のサーバー共通名の値をチェックするかどうかを指定します。分散リカバリのための SSL の構成の詳細は、[セクション 18.5.2 「Secure Socket Layer \(SSL\) を使用したグループ通信接続の保護」](#) を参照してください。

- [group_replication_recovery_tls_ciphersuites](#)

コマンド行形式	<code>--group-replication-recovery-tls-ciphersuites=value</code>
導入	8.0.19
システム変数	group_replication_recovery_tls_ciphersuites
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	NULL

このシステム変数の値は、Group Replication の実行中に変更できますが、変更はグループメンバーで Group Replication を停止して再起動した後にのみ有効になります。

[group_replication_recovery_tls_ciphersuites](#) では、分散リカバリ接続の接続暗号化に TLSv1.3 が使用され、このサーバーインスタンスが分散リカバリ接続のクライアント (参加メンバー) である場合に、許可されている 1 つ以上の暗号スイートのコロン区切りリストを指定します。TLSv1.3 の使用時にこのシステム変数が NULL に設定されている場合 (システム変数を設定しない場合のデフォルト)、[セクション 6.3.2 「暗号化された接続 TLS プロトコルおよび暗号」](#) にリストされているように、デフォルトで有効になっている暗号スイートが許可されます。このシステム変数が空の文字列に設定されている場合、暗号スイートは許可されないため、TLSv1.3 は使用されません。このシステム変数は、MySQL 8.0.19 以降で使用できます。分散リカバリのための SSL の構成の詳細は、[セクション 18.5.2 「Secure Socket Layer \(SSL\) を使用したグループ通信接続の保護」](#) を参照してください。

- [group_replication_recovery_tls_version](#)

コマンド行形式	<code>--group-replication-recovery-tls-version=value</code>
導入	8.0.19
システム変数	group_replication_recovery_tls_version
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	TLSv1,TLSv1.1,TLSv1.2,TLSv1.3

このシステム変数の値は、Group Replication の実行中に変更できますが、変更はグループメンバーで Group Replication を停止して再起動した後にのみ有効になります。

[group_replication_recovery_tls_version](#) では、このサーバーインスタンスが分散リカバリ接続のクライアント (参加メンバー) である場合に、接続暗号化に許可される TLS プロトコルのカンマ区切りリストを指定します。指定したバージョンが連続していることを確認します (たとえば、「[TLSv1,TLSv1.1,TLSv1.2](#)」)。このシステム変数が設定されていない場合、デフォルトの「[TLSv1,TLSv1.1,TLSv1.2,TLSv1.3](#)」が使用されます。クライアント (参加メンバー) およびサーバー (ドナー) としての各分散リカバリ接続に含まれるグループメンバーは、どちらもサポートするように設定されている最高のプロトコルバージョンをネゴシエートします。このシステム変数は、MySQL 8.0.19

から使用できます。分散リカバリのための SSL の構成の詳細は、[セクション18.5.2「Secure Socket Layer \(SSL\) を使用したグループ通信接続の保護」](#)を参照してください。

- [group_replication_recovery_use_ssl](#)

コマンド行形式	<code>--group-replication-recovery-use-ssl[={OFF ON}]</code>
システム変数	group_replication_recovery_use_ssl
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

このシステム変数の値は、Group Replication の実行中に変更できますが、変更はグループメンバーで Group Replication を停止して再起動した後にのみ有効になります。

[group_replication_recovery_use_ssl](#) では、グループメンバー間のグループレプリケーション分散リカバリ接続で SSL を使用するかどうかを指定します。分散リカバリのための SSL の構成の詳細は、[セクション18.5.2「Secure Socket Layer \(SSL\) を使用したグループ通信接続の保護」](#)を参照してください。

このサーバーがクローニングをサポートするように設定されている場合 ([セクション18.4.3.2「分散リカバリのためのクローニング」](#)を参照)、このオプションを ON に設定すると、グループレプリケーションでは、リモートクローニング操作およびドナーのバイナリログからの状態転送に SSL が使用されます。このオプションを OFF に設定すると、グループレプリケーションはリモートクローニング操作に SSL を使用しません。

- [group_replication_recovery_zstd_compression_level](#)

コマンド行形式	<code>--group-replication-recovery-zstd-compression-level=#</code>
導入	8.0.18
システム変数	group_replication_recovery_zstd_compression_level
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	3
最小値	1
最大値	22

このシステム変数の値は、Group Replication の実行中に変更できますが、変更はグループメンバーで Group Replication を停止して再起動した後にのみ有効になります。

[group_replication_recovery_zstd_compression_level](#) では、zstd 圧縮アルゴリズムを使用するグループレプリケーション分散リカバリ接続に使用する圧縮レベルを指定します。許可されるレベルは 1 から 22 で、大きい値は圧縮レベルの増加を示します。デフォルトの zstd 圧縮レベルは 3 です。zstd 圧縮を使用しない分散リカバリ接続の場合、この変数は無効です。

詳細は、[セクション4.2.8「接続圧縮制御」](#)を参照してください。

- [group_replication_single_primary_mode](#)

コマンド行形式	<code>--group-replication-single-primary-mode[={OFF ON}]</code>
システム変数	group_replication_single_primary_mode
スコープ	グローバル

動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

このシステム変数は、グループ全体の構成設定です。値の変更を有効にするには、すべてのグループメンバーで同じ値を持つ必要があり、グループレプリケーションの実行中は変更できず、グループ (`group_replication_bootstrap_group=ON` を使用したサーバーによるブートストラップ) を完全に再起動する必要があります。MySQL 8.0.16 から、`group_replication_switch_to_single_primary_mode()` および `group_replication_switch_to_multi_primary_mode()` UDF を使用して、グループの実行中にこのシステム変数の値を変更できます。詳細は、[セクション18.4.1.2「グループモードの変更」](#)を参照してください。

`group_replication_single_primary_mode` は、読取り/書き込みワークロードを処理する単一のサーバーを自動的に選択するようにグループに指示します。このサーバーは PRIMARY で、その他はすべて SECONDARIES です。

- `group_replication_ssl_mode`

コマンド行形式	<code>--group-replication-ssl-mode=value</code>
システム変数	<code>group_replication_ssl_mode</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	DISABLED
有効な値	DISABLED REQUIRED VERIFY_CA VERIFY_IDENTITY

このシステム変数の値は、Group Replication の実行中に変更できますが、変更はグループメンバーで Group Replication を停止して再起動した後のみ有効になります。

`group_replication_ssl_mode` は、グループレプリケーションメンバー間のグループ通信接続のセキュリティ状態を設定します。使用可能な値は次のとおりです:

- DISABLED 暗号化されていない接続を確立します (デフォルト)。
- REQUIRED サーバーがセキュアな接続をサポートしている場合は、セキュアな接続を確立します。
- VERIFY_CA `REQUIRED` と似ていますが、さらに、構成された認証局 (CA) 証明書に対してサーバー TLS 証明書を検証します。
- VERIFY_IDENTITY `VERIFY_CA` と似ていますが、さらに、サーバー証明書が接続が試行されるホストと一致することを確認します。

グループ通信用の SSL の構成の詳細は、[セクション18.5.2「Secure Socket Layer \(SSL\) を使用したグループ通信接続の保護」](#)を参照してください。

- `group_replication_start_on_boot`

コマンド行形式	<code>--group-replication-start-on-boot[={OFF ON}]</code>
システム変数	<code>group_replication_start_on_boot</code>
スコープ	グローバル

動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

このシステム変数の値は、Group Replication の実行中に変更できますが、変更はグループメンバーで Group Replication を停止して再起動した後にのみ有効になります。

`group_replication_start_on_boot` では、サーバーの起動時にグループレプリケーションを自動的に起動するか (ON) 起動しないか (OFF) を指定します。このオプションを ON に設定すると、リモートクローニング操作を分散リカバリに使用した後、グループレプリケーションが自動的に再起動されます。

サーバーの起動時に Group Replication を自動的に起動するには、`CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO` ステートメントを使用して、分散リカバリのユーザー資格証明をサーバー上のレプリケーションメタデータリポジトリに格納する必要があります。ユーザー資格証明をメモリーにのみ格納する `START GROUP_REPLICATION` ステートメントでユーザー資格証明を指定する場合は、`group_replication_start_on_boot` が OFF に設定されていることを確認します。

- `group_replication_tls_source`

コマンド行形式	<code>--group-replication-tls-source=value</code>
導入	8.0.21
システム変数	<code>group_replication_tls_source</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	<code>mysql_main</code>
有効な値	<code>mysql_main</code> <code>mysql_admin</code>

このシステム変数の値は、Group Replication の実行中に変更できますが、変更はグループメンバーで Group Replication を停止して再起動した後にのみ有効になります。

`group_replication_tls_source` は、Group Replication の TLS 材料のソースを指定します。

- `group_replication_transaction_size_limit`

コマンド行形式	<code>--group-replication-transaction-size-limit=#</code>
システム変数	<code>group_replication_transaction_size_limit</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	<code>150000000</code>
最小値	<code>0</code>
最大値	<code>2147483647</code>

このシステム変数は、すべてのグループメンバーで同じ値を持つ必要があります。このシステム変数の値は、Group Replication の実行中に変更できます。変更はグループメンバーに対して即時に有効になり、そのメンバーで開始された次のトランザクションから適用されます。このプロセス中、システム変数の値はグループメンバー間で異なってもかまいませんが、一部のトランザクションは拒否される可能性があります。

`group_replication_transaction_size_limit` は、レプリケーショングループが受け入れる最大トランザクションサイズをバイト単位で構成します。このサイズより大きいトランザクションは受信側メンバーによってロールバックされ、グループにブロードキャストされません。大規模なトランザクションでは、システムの色度が低下する原因となる可能性のあるメモリー割当て、または大規模なトランザクションの処理がビジー状態であるためにメンバーに障害が発生した疑いがあるネットワーク帯域幅消費の観点から、レプリケーショングループの問題が発生する可能性があります。

このシステム変数が 0 に設定されている場合、グループが受け入れるトランザクションのサイズに制限はありません。MySQL 8.0 からは、このシステム変数のデフォルト設定は 150000000 バイト (約 143 MB) です。トランザクションの処理にかかる時間はそのサイズに比例することに注意して、グループが許容する必要がある最大メッセージサイズに応じて、このシステム変数の値を調整します。`group_replication_transaction_size_limit` の値は、すべてのグループメンバーで同じである必要があります。大規模トランザクションの軽減戦略の詳細は、[セクション 18.9.2 「グループレプリケーションの制限事項」](#) を参照してください。

- `group_replication_unreachable_majority_timeout`

コマンド行形式	<code>--group-replication-unreachable-majority-timeout=#</code>
システム変数	<code>group_replication_unreachable_majority_timeout</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	31536000

このシステム変数の値は、Group Replication の実行中に変更でき、変更はただちに有効になります。システム変数の現在の値は、動作が必要であることを意味する問題が発生したときに読み取られます。

`group_replication_unreachable_majority_timeout` では、ネットワークパーティションが不足し、大部分のメンバーがグループを離れるまで待機できないメンバーの秒数を指定します。5 つのサーバー (S1、S2、S3、S4、S5) のグループでは、(S1、S2) と (S3、S4、S5) の間に切断がある場合、ネットワークパーティションがあります。最初のグループ (S1、S2) は、半分を超えるグループにコンタクトできないため、少数民族になりました。大多数のグループ (S3、S4、S5) が実行されている間、少数民族グループはネットワークの再接続を指定された時間待機します。このシナリオの詳細は、[セクション 18.4.4 「ネットワークパーティション化」](#) を参照してください。

デフォルトでは、`group_replication_unreachable_majority_timeout` は 0 に設定されています。これは、ネットワークパーティションのために少数民族で自身を検索するメンバーが、グループから離れるまで永久に待機することを意味します。タイムアウトを設定した場合、指定した時間が経過すると、その少数民族によって処理されたすべての保留中のトランザクションがロールバックされ、少数パーティション内のサーバーは `ERROR` 状態に移行します。メンバーの `group_replication_autorejoin_tries` システム変数が自動再結合の試行回数を指定するように設定されている場合、スーパー読み取り専用モードでは、指定された回数だけグループへの再結合が試行されます。メンバーに自動再結合の試行が指定されていない場合、または指定された試行回数を使い果たした場合は、システム変数 `group_replication_exit_state_action` で指定されたアクションに従います。

警告

対称グループがあり、たとえば (S0、S2) のメンバーが 2 つのみで、ネットワークパーティションがあり、大部分がない場合、構成されたタイムアウトの後、すべてのメンバーが `ERROR` 状態になります。

このオプションの使用の詳細は、[セクション 18.6.6.2 「使用できない大多数のタイムアウト」](#) を参照してください。

グループのレプリケーションステータス変数

このセクションでは、グループレプリケーションに関する情報を提供する status 変数について説明します。変数の意味は次のとおりです:

- `group_replication_primary_member`

グループがシングルプライマリモードで動作している場合、プライマリメンバー UUID を表示します。グループがマルチプライマリモードで動作している場合は、空の文字列が表示されます。

警告

`group_replication_primary_member` ステータス変数は非推奨になり、将来のバージョンで削除される予定です。

[プライマリの検索](#)を参照してください。

18.9 要件と制限事項

このセクションでは、グループレプリケーションの要件および制限事項をリストし、説明します。

18.9.1 グループレプリケーションの要件

グループレプリケーションに使用するサーバーインスタンスは、次の要件を満たす必要があります。

インフラストラクチャ

- InnoDB ストレージエンジン。 データは、InnoDB トランザクションストレージエンジンに格納する必要があります。 トランザクションはオプティミスティックに実行され、コミット時に競合がないかチェックされます。競合がある場合、グループ間の一貫性を維持するために、一部のトランザクションがロールバックされます。つまり、トランザクションストレージエンジンが必要です。さらに、InnoDB には、グループレプリケーションと連携して動作する場合の競合の管理および処理を向上させる追加機能がいくつか用意されています。一時的な MEMORY ストレージエンジンを含むほかのストレージエンジンを使用すると、Group Replication でエラーが発生する可能性があります。グループメンバーに `disabled_storage_engines` システム変数を設定することで、ほかのストレージエンジンの使用を防止できます。次に例を示します:

```
disabled_storage_engines="MyISAM,BLACKHOLE,FEDERATED,ARCHIVE,MEMORY"
```

- 主キー。 グループによってレプリケートされるすべてのテーブルには、定義済の主キー、または同等の主キー(同等のものが NULL 以外の一意キー)が必要です。このようなキーは、テーブル内のすべての行の一意的識別子として必要です。これにより、各トランザクションが変更された行を正確に識別することで、競合するトランザクションをシステムで判別できます。Group Replication には、主キーまたは主キーに相当するチェックの独自の組込みセットがあり、`sql_require_primary_key` システム変数によって実行されるチェックは使用されません。Group Replication が実行されているサーバーインスタンスに対して `sql_require_primary_key=ON` を設定でき、Group Replication チャネルに対して `CHANGE REPLICATION SOURCE TO|CHANGE MASTER TO` ステートメントの `REQUIRE_TABLE_PRIMARY_KEY_CHECK` オプションを `ON` に設定できます。ただし、グループレプリケーションの組込みチェックで許可されている一部のトランザクションは、`sql_require_primary_key=ON` または `REQUIRE_TABLE_PRIMARY_KEY_CHECK=ON` の設定時に実行されるチェックでは許可されない場合があることに注意してください。
- ネットワークパフォーマンス。 MySQL Group Replication は、サーバーインスタンスが相互に非常に近いクラスター環境にデプロイされるように設計されています。グループのパフォーマンスと安定性は、ネットワーク待機時間とネットワーク帯域幅の両方の影響を受ける可能性があります。双方向通信は、すべてのグループメンバー間で常に維持する必要があります。インバウンド通信またはアウトバウンド通信のいずれかがサーバーインスタンスでブロックされている場合(ファイアウォールや接続の問題など)、メンバーはグループ内で機能できず、グループメンバー(問題のあるメンバーを含む)は影響を受けるサーバーインスタンスの正しいメンバーステータスをレポートできない可能性があります。

MySQL 8.0.14 からは、リモートグループレプリケーションサーバー間の TCP 通信に IPv4 または IPv6 ネットワークインフラストラクチャを使用することも、これらの組合せを使用することもできます。また、グループレプリケーションが仮想プライベートネットワーク (VPN) 上で動作しなくなることもありません。

また、グループレプリケーションサーバーインスタンスが同じ場所に配置され、ローカルグループ通信エンジン (XCom) インスタンスを共有する MySQL 8.0.14 から、TCP ソケットではなく可能なかぎり、通信にオーバーヘッドの低い専用の入力チャネルが使用されます。グループへの参加など、リモートの XCom インスタンス間の通信を必要とする特定のグループレプリケーションタスクでは、TCP ネットワークが引き続き使用されるため、ネットワークパフォーマンスがグループのパフォーマンスに影響します。

サーバーインスタンス構成

次のオプションは、グループのメンバーであるサーバーインスタンスに表示されるように構成する必要があります。

- 一意のサーバー識別子。レプリケーショントポロジのすべてのサーバーで必要に応じて、`server_id` システム変数を使用して一意のサーバー ID でサーバーを構成します。サーバー ID は、1 から $(2^{32})-1$ までの正の整数である必要があります。レプリケーショントポロジ内の他のサーバーで使用されている他のすべてのサーバー ID とは異なる必要があります。
- バイナリログアクティブ。 `--log-bin[=log_file_name]` を設定します。MySQL Group Replication はバイナリログの内容をレプリケートするため、バイナリログを操作するにはバイナリログがオンになっている必要があります。このオプションはデフォルトで有効となっています。 [セクション5.4.4「バイナリログ」](#) を参照してください。
- 記録されたレプリカ更新。 `--log-slave-updates` を設定します。このオプションはデフォルトで有効となっています。グループメンバーは、参加時にドナーから受信し、レプリケーションアプライアンスを介して適用されるトランザクションをログに記録し、グループから受信して適用するすべてのトランザクションをログに記録する必要があります。これにより、グループレプリケーションは、既存のグループメンバーバイナリログから状態転送による分散リカバリを実行できます。
- バイナリログ形式。 `--binlog-format=row` を設定します。グループレプリケーションは、行ベースのレプリケーション形式に依存して、グループ内のサーバー間で一貫して変更を伝播します。グループ内の異なるサーバーで同時に実行されるトランザクション間の競合を検出するために必要な情報を抽出できるように、行ベースのインフラストラクチャに依存します。MySQL 8.0.19 から、[REQUIRE_ROW_FORMAT](#) 設定がグループレプリケーションチャネルに自動的に追加され、トランザクションの適用時に行ベースのレプリケーションの使用が強制されます。 [セクション17.2.1「レプリケーション形式」](#) および [セクション17.3.3「レプリケーション権限チェック」](#) を参照してください。
- バイナリログチェックサムオフ (MySQL 8.0.20)。MySQL 8.0.20 まで、およびそれを含めて、`--binlog-checksum=NONE` を設定します。これらのリリースでは、Group Replication はチェックサムを使用できず、バイナリログ内でのチェックサムの存在をサポートしていません。MySQL 8.0.21 からは、グループレプリケーションでチェックサムがサポートされるため、グループメンバーはデフォルト設定を使用できます。
- グローバルトランザクション識別子オン。 `gtid_mode=ON` および `enforce_gtid_consistency=ON` を設定します。グループレプリケーションでは、グローバルトランザクション識別子を使用して、すべてのサーバーインスタンスでコミットされたトランザクションを正確に追跡するため、すでにコミットされているトランザクションと競合する可能性のあるトランザクションを実行したサーバーを推測できます。つまり、明示的なトランザクション識別子は、競合する可能性のあるトランザクションを判別できるフレームワークの基本的な部分です。 [セクション17.1.3「グローバルトランザクション識別子を使用したレプリケーション」](#) を参照してください。
- レプリケーション情報リポジトリ。 `master_info_repository=TABLE` および `relay_log_info_repository=TABLE` を設定します。MySQL 8.0 では、この設定はデフォルトであり、`FILE` 設定は非推奨です。MySQL 8.0.23 では、これらのシステム変数の使用は非推奨であるため、システム変数を省略してデフォルトをそのまま使用します。Group Replication プラグインがレプリケーションメタデータの一致したリカバリ可能性とトランザクション管理を持つように、レプリケーションアプライアンスは `mysql.slave_master_info` および `mysql.slave_relay_log_info` システムテーブルにレプリケーションメタデータを書き込む必要があります。 [セクション17.2.4.2「レプリケーションメタデータリポジトリ」](#) を参照してください。
- トランザクション書き込みセット抽出。行を収集してバイナリログに記録するときに、サーバーが書き込みセットも収集するように `--transaction-write-set-extraction=XXHASH64` を設定します。書き込みセットは各行の主キーに基づいており、変更された行を一意に識別するタグの簡略化されたコンパクトなビューです。このタグは、競合の検出に使用されます。このオプションはデフォルトで有効となっています。
- バイナリログの依存性トラッキング。 `binlog_transaction_dependency_tracking=WRITESET_SESSION` を設定すると、グループワークロードに応じて、グループメンバーのパフォーマンスを向上させることができます。Group Replication は、`binlog_transaction_dependency_tracking` に設定された値とは関係な

く、リレーログからトランザクションを適用するときに、証明後に独自のパラレル化を実行します。ただし、`binlog_transaction_dependency_tracking` の値は、グループレプリケーションメンバーのバイナリログへのトランザクションの書き込み方法に影響します。これらのログ内の依存関係情報は、メンバーがグループに参加または再参加するたびに発生する分散リカバリのドナーバイナリログからの状態転送プロセスを支援するために使用されません。

- デフォルトのテーブル暗号化. すべてのグループメンバーで `--default-table-encryption` を同じ値に設定します。デフォルトのスキーマおよびテーブルスペースの暗号化は、設定がすべてのメンバーで同じであるかぎり、有効 (ON) または無効 (OFF、デフォルト) のいずれかにできます。
- 小文字のテーブル名. すべてのグループメンバーで `--lower-case-table-names` を同じ値に設定します。グループレプリケーションに必要な InnoDB ストレージエンジンを使用するには、1 の設定が適切です。この設定は、すべてのプラットフォームでデフォルトであるわけではありません。
- マルチスレッドアプライアンス. グループレプリケーションメンバーはマルチスレッドレプリカとして構成できるため、トランザクションをパラレルに適用できます。 `slave_parallel_workers` にゼロ以外の値を指定すると、メンバーのマルチスレッドアプライアンスが有効になり、最大 1024 のパラレルアプライヤスレッドを指定できます。 `slave_preserve_commit_order=1` を設定すると、パラレルトランザクションの最終コミットは、グループレプリケーションに必要な元のトランザクションと同じ順序で行われます。これは、すべての参加メンバーがコミットされたトランザクションを同じ順序で受信および適用することを保証するために構築された一貫性メカニズムに依存します。最後に、レプリカでパラレルに実行できるトランザクションを決定するために使用されるポリシーを指定する `slave_parallel_type=LOGICAL_CLOCK` の設定が、 `slave_preserve_commit_order=1` で必要です。 `slave_parallel_workers=0` を設定すると、パラレル実行が無効になり、レプリカに単一のアプライヤスレッドが付与され、コーディネータスレッドは付与されません。この設定では、 `slave_parallel_type` および `slave_preserve_commit_order` オプションは効果がなく、無視されます。

18.9.2 グループレプリケーションの制限事項

グループレプリケーションには、次の既知の制限事項があります。マルチプライマリモードグループについて説明されている制限事項と問題は、フェイルオーバーイベント中にシングルプライマリモードクラスタにも適用できますが、新しく選択されたプライマリは古いプライマリからアプライヤキューをフラッシュします。

ヒント

グループレプリケーションは GTID ベースのレプリケーション上に構築されるため、[セクション 17.1.3.7 「GTID ベースレプリケーションの制約」](#) にも注意する必要があります。

- `--upgrade=MINIMAL` オプション. レプリケーション内部が依存するシステムテーブルをアップグレードしない MINIMAL オプション (`--upgrade=MINIMAL`) を使用する MySQL Server のアップグレード後は、グループレプリケーションを開始できません。
- ギャップロック. ギャップロックに関する情報は InnoDB の外部では使用できないため、同時トランザクションのグループレプリケーション動作保証プロセスでは `gap locks` は考慮されません。詳しくは [ギャップロック](#) をご覧ください。

注記

マルチプライマリモードのグループの場合、アプリケーションで `REPEATABLE READ` セマンティクスに依存しないかぎり、グループレプリケーションで `READ COMMITTED` 分離レベルを使用することをお勧めします。InnoDB では、`READ COMMITTED` のギャップロックは使用されません。これにより、InnoDB 内のローカル競合検出が、グループレプリケーションによって実行される分散競合検出に位置合せされます。シングルプライマリモードのグループの場合、プライマリのみが書き込みを受け入れるため、`READ COMMITTED` 分離レベルはグループレプリケーションにとって重要ではありません。

- テーブルロックおよび名前付きロック. 証明プロセスでは、テーブルロック ([セクション 13.3.6 「LOCK TABLES および UNLOCK TABLES ステートメント」](#) を参照) または名前付きロック (`GET_LOCK()` を参照) は考慮されません。
- バイナリログチェックサム. MySQL 8.0.20 まで、グループレプリケーションはチェックサムを使用できず、バイナリログでのチェックサムの存在をサポートしないため、サーバーインスタンスがグループメンバーになるように

構成するときに `binlog_checksum=NONE` を設定する必要があります。MySQL 8.0.21 からは、グループレプリケーションでチェックサムがサポートされるため、グループメンバーはデフォルト設定の `binlog_checksum=CRC32` を使用できます。`binlog_checksum` の設定は、グループのすべてのメンバーで同じである必要はありません。

チェックサムが使用可能な場合、Group Replication はそれらを使用して `group_replication_applier` チャンネル上の着信イベントを検証しません。これは、イベントが複数のソースからそのリレーログに書き込まれ、実際に元のサーバーのバイナリログに書き込まれる前 (チェックサムが生成される時) であるためです。チェックサムは、`group_replication_recovery` チャンネルおよびグループメンバー上の他のレプリケーションチャンネルでイベントの整合性を検証するために使用されます。

- **SERIALIZABLE 分離レベル。** `SERIALIZABLE` 分離レベルは、マルチプライマリグループではデフォルトでサポートされていません。トランザクション分離レベルを `SERIALIZABLE` に設定すると、トランザクションのコミットを拒否するようにグループレプリケーションが構成されます。
- **同時 DDL 操作と DML 操作。** マルチプライマリモードを使用している場合、同じオブジェクトに対して実行されるが異なるサーバーで実行される同時データ定義ステートメントおよびデータ操作ステートメントはサポートされません。オブジェクトに対するデータ定義言語 (DDL) ステートメントの実行中に、同じオブジェクトで異なるサーバーインスタンスで同時データ操作言語 (DML) を実行すると、異なるインスタンスで競合する DDL が検出されないリスクがあります。
- **カスケード制約のある外部キー。** マルチプライマリモードグループ (すべて `group_replication_single_primary_mode=OFF` で構成されたメンバー) は、マルチレベルの外部キー依存性を持つテーブル (特に `CASCADING foreign key constraints` を定義したテーブル) をサポートしていません。これは、マルチプライマリモードグループによってカスケード操作が実行される外部キー制約により、競合が検出されず、グループのメンバー間でデータの一貫性がなくなる可能性があるためです。したがって、検出されない競合を回避するために、マルチプライマリモードグループで使用されるサーバーインスタンスに `group_replication_enforce_update_everywhere_checks=ON` を設定することをお勧めします。

シングルプライマリモードでは、グループの複数のメンバーへの同時書き込みが許可されないため、これは問題ではありません。したがって、検出されない競合のリスクはありません。

- **マルチプライマリモードのデッドロック。** グループがマルチプライマリモードで動作している場合、`SELECT .. FOR UPDATE` ステートメントによってデッドロックが発生する可能性があります。これは、ロックがグループのメンバー間で共有されていないため、このようなステートメントの期待に到達しない可能性があるためです。
- **レプリケーションフィルタ。** グループレプリケーション用に構成された MySQL サーバーインスタンスでは、グローバルレプリケーションフィルタを使用できません。これは、一部のサーバーでトランザクションをフィルタすると、グループが一貫性のある状態で承諾に到達できなくなるためです。グループメンバーがグループ外のソースへのレプリカとしても機能する場合など、グループレプリケーションに直接関与しないレプリケーションチャンネルでチャンネル固有のレプリケーションフィルタを使用できます。`group_replication_applier` または `group_replication_recovery` チャンネルでは使用できません。
- **暗号化接続。** TLSv1.3 プロトコルのサポートは、MySQL 8.0.16 の MySQL Server で使用できます (MySQL が OpenSSL 1.1.1 以上を使用してコンパイルされている場合)。MySQL 8.0.16 および MySQL 8.0.17 では、サーバーが TLSv1.3 をサポートしている場合、プロトコルはグループ通信エンジンではサポートされず、Group Replication では使用できません。グループレプリケーションでは、MySQL 8.0.18 からの TLSv1.3 がサポートされており、グループ通信接続および分散リカバリ接続に使用できます。

MySQL 8.0.18 では、TLSv1.3 を分散リカバリ接続のグループレプリケーションで使用できますが、`group_replication_recovery_tls_version` および `group_replication_recovery_tls_ciphersuites` システム変数は使用できません。したがって、ドナーサーバーでは、[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#) にリストされているように、デフォルトで有効になっている TLSv1.3 暗号スイートを少なくとも 1 つ使用する必要があります。MySQL 8.0.19 から、オプションを使用して、必要に応じてデフォルト以外の暗号スイートのみを含む任意の暗号スイートのクライアントサポートを構成できます。

- **クローニング操作。** グループレプリケーションは分散リカバリのクローニング操作を開始および管理しますが、クローニングをサポートするように設定されているグループメンバーは、ユーザーが手動で開始するクローニング操作にも関与する場合があります。MySQL 8.0.20 より前のリリースでは、操作にグループレプリケーションが実行されているグループメンバーが含まれる場合、クローニング操作を手動で開始することはできません。クローニング操作で受信者のデータが削除および置換されない場合は、MySQL 8.0.20 からこれを実行できます。したがって、Group Replication が実行されている場合は、クローニング操作を開始するステートメントに `DATA DIRECTORY` 句を含める必要があります。[その他の目的でのクローニング](#)を参照してください。

グループサイズの制限

単一のレプリケーショングループのメンバーにできる MySQL サーバーの最大数は 9 です。さらにメンバーがグループに参加しようとする、そのリクエストは拒否されます。この制限は、安定したローカルエリアネットワーク上でグループが確実に実行される安全な境界としてのテストおよびベンチマークから特定されています。

トランザクションサイズの制限

個々のトランザクションの結果、5 秒以内にメッセージをネットワーク上のグループメンバー間でコピーできない十分な大きさのメッセージコンテンツが生成された場合、メンバーはトランザクションの処理にビジー状態であるため、失敗した疑いがあります。また、トランザクションが大きいと、メモリー割当ての問題が原因でシステムが遅くなる可能性があります。これらの問題を回避するには、次の軽減策を使用します:

- 大量のメッセージが原因で不要な削除が発生した場合は、システム変数 `group_replication_member_expel_timeout` を使用して、失敗した疑いがあるメンバーが削除されるまでの追加時間を許可します。疑わしいメンバーがグループから削除されるまで、最初の 5 秒間の検出期間の 1 時間以内に許可できます。MySQL 8.0.21 からは、デフォルトで追加の 5 秒が許可されます。
- 可能な場合は、グループレプリケーションによって処理される前に、トランザクションのサイズを試行して制限してください。たとえば、`LOAD DATA` で使用されるファイルを小さいチャンクに分割します。
- システム変数 `group_replication_transaction_size_limit` を使用して、グループが受け入れる最大トランザクションサイズを指定します。MySQL 8.0 では、このシステム変数はデフォルトで 150000000 バイト (約 143 MB) の最大トランザクションサイズに設定されます。このサイズを超えるトランザクションはロールバックされ、グループへの配布のために Group Replication Group Communication System (GCS) に送信されません。トランザクションの処理にかかる時間はそのサイズに比例することに注意して、グループが許容する必要がある最大メッセージサイズに応じて、この変数の値を調整します。
- システム変数 `group_replication_compression_threshold` を使用して、圧縮が適用されるメッセージサイズを指定します。このシステム変数のデフォルトは 1000000 バイト (1 MB) であるため、大きなメッセージは自動的に圧縮されます。圧縮は、`group_replication_transaction_size_limit` 設定で許可された `group_replication_compression_threshold` 設定を超えたメッセージを受信したときに、Group Replication Group Communication System (GCS) によって実行されます。詳細は、[セクション18.6.3「メッセージ圧縮」](#)を参照してください。
- システム変数 `group_replication_communication_max_message_size` を使用して、メッセージが断片化されるメッセージサイズを指定します。このシステム変数のデフォルトは 10485760 バイト (10 MiB) であるため、大きなメッセージは自動的に断片化されます。GCS は、圧縮されたメッセージが `group_replication_communication_max_message_size` 制限を超えたままである場合、圧縮後に断片化を実行します。レプリケーショングループで断片化を使用するには、すべてのグループメンバーが MySQL 8.0.16 以上であり、グループで使用されている Group Replication 通信プロトコルバージョンで断片化が許可されている必要があります。詳細は、[セクション18.6.4「メッセージの断片化」](#)を参照してください。

関連するシステム変数にゼロ値を指定すると、最大トランザクションサイズ、メッセージ圧縮およびメッセージの断片化をすべて非アクティブ化できます。これらすべての保護を非アクティブ化した場合、レプリケーショングループのメンバーの適用者スレッドで処理できるメッセージの上限サイズは、デフォルト値および最大値 1073741824 バイト (1 GB) のメンバー `slave_max_allowed_packet` システム変数の値です。この制限を超えるメッセージは、受信側メンバーが処理しようすると失敗します。グループメンバーが送信できるメッセージの上限サイズは 4294967295 バイト (約 4 GB) です。これは、GCS が処理した後にメッセージを受信するグループレプリケーション (XCom、Paxos バリエーション) のためにグループ通信エンジンによって受け入れられるパケットサイズの強い制限です。この制限を超えるメッセージは、元のメンバーがブロードキャストしようすると失敗します。

18.10 よくある質問

このセクションでは、よくある質問への回答を示します。

グループ内の MySQL サーバーの最大数はいくつですか。

グループは、最大 9 台のサーバーで構成できます。9 つのメンバーを持つグループに別のサーバーを追加しようすると、リクエストは拒否されます。この制限は、安定したローカルエリアネットワーク上でグループが確実に実行される安全な境界としてのテストおよびベンチマークから特定されています。

グループ内のサーバーはどのように接続されていますか。

グループ内のサーバーは、ピアツーピア TCP 接続を開いてグループ内の他のサーバーに接続します。これらの接続は、グループ内のサーバー間の内部通信およびメッセージの受渡しにのみ使用されます。このアドレスは、`group_replication_local_address` 変数によって構成されます。

`group_replication_bootstrap_group` オプションは何に使用されますか。

ブートストラップフラグは、メンバーにグループを create に指示し、初期シードサーバーとして機能します。グループに参加する 2 つ目のメンバーは、グループに追加するために構成を動的に変更するようにグループをブートストラップしたメンバーに依頼する必要があります。

メンバーは、2 つのシナリオでグループをブートストラップする必要があります。グループが最初に作成されたとき、またはグループ全体を停止して再起動したとき。

分散リカバリプロセスの資格証明を設定するにはどうすればよいですか。

ユーザー資格証明は、`CHANGE REPLICATION SOURCE TO` ステートメント (MySQL 8.0.23 の場合) または `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 の場合) を使用して、`group_replication_recovery` チャンネルの資格証明として永続的に設定できます。または、MySQL 8.0.21 から、グループレプリケーションが開始されるたびに `START GROUP_REPLICATION` ステートメントで指定できます。

`CHANGE REPLICATION SOURCE TO` | `CHANGE MASTER TO` を使用して設定されたユーザー資格証明は、サーバー上のレプリケーションメタデータリポジトリにプレーンテキストで格納されますが、`START GROUP_REPLICATION` で指定されたユーザー資格証明はメモリーにのみ保存され、`STOP GROUP_REPLICATION` ステートメントまたはサーバーの停止によって削除されます。したがって、`START GROUP_REPLICATION` を使用してユーザー資格証明を指定すると、不正なアクセスからグループレプリケーションサーバーを保護するのに役立ちます。ただし、この方法は、`group_replication_start_on_boot` システム変数で指定された Group Replication の自動起動とは互換性がありません。詳細は、[セクション 18.5.3.1 「分散リカバリのためのセキュアなユーザー資格証明」](#) を参照してください。

グループレプリケーションを使用して書き込みロードをスケールアウトできますか。

直接ではありませんが、MySQL Group レプリケーションは、グループ内のすべてのサーバーが同じ量のデータをレプリケートする、何も共有しない完全レプリケーションソリューションです。したがって、トランザクションのコミット操作の結果として、グループ内のあるメンバーが N バイトをストレージに書き込む場合、トランザクションはどこにでもレプリケートされるため、他のメンバーのストレージにも約 N バイトが書き込まれます。

ただし、最初にトランザクションを実行したときに元のメンバーが実行する必要があったのと同じ量の処理を他のメンバーが実行する必要がない場合は、変更がより高速に適用されます。トランザクションは、行変換の適用にのみ使用される形式でレプリケートされ、トランザクションを再実行する必要はありません (行ベースの形式)。

さらに、変更が行ベースの形式で伝播および適用される場合、それらは最適化されたコンパクトな形式で受信され、元のメンバーと比較して必要な IO 操作の数が削減される可能性があります。

要約すると、競合のないトランザクションをグループ内の様々なメンバーに分散することで、処理をスケールアウトできます。また、リモートサーバーは安定したストレージへの読取り/変更/書き込み変更に必要な変更のみを受信するため、IO 操作のごく一部をスケールアウトできます。

グループレプリケーションでは、単純なレプリケーションと比較して、同じワークロードでより多くのネットワーク帯域幅と CPU が必要ですか。

同期化のためにサーバーが相互に常に相互作用する必要があるため、追加のロードが必要になります。これ以上のデータ量を定量化することは困難です。また、グループのサイズによっても異なります (3 台のサーバーは、グループ内の 9 台のサーバーよりも帯域幅要件に応力を軽減します)。

また、サーバー同期部分およびグループメッセージングではより複雑な作業が行われるため、メモリーおよび CPU フットプリントも大きくなります。

広域ネットワーク間でグループレプリケーションをデプロイできますか。

はい。ただし、各メンバー間のネットワーク接続は信頼性が高く、適切なパフォーマンスを備えている必要があります。最適なパフォーマンスを得るには、低レイテンシで高帯域幅のネットワーク接続が必要です。

ネットワーク帯域幅のみに問題がある場合、[セクション18.6.3「メッセージ圧縮」](#)を使用して必要な帯域幅を減らすことができます。ただし、ネットワークがパケットをドロップし、再送信とエンドツーエンドの待機時間が長くなると、スループットと待機時間の両方に悪影響を与えます。

警告

いずれかのグループメンバー間のネットワークラウンドトリップ時間 (RTT) が 5 秒以上の場合、組込みの障害検出メカニズムが誤ってトリガーされる可能性があるため、問題が発生する可能性があります。

一時的な接続の問題が発生した場合、メンバーは自動的にグループに再参加しますか。

これは、接続の問題の理由によって異なります。接続の問題が一時的で、障害検出で認識されないほど迅速に再接続できる場合、サーバーはグループから削除されない可能性があります。接続性の問題が長い場合、障害検出機能は最終的に問題を疑い、サーバーはグループから削除されます。

MySQL 8.0 から、グループに残っているメンバーまたはグループに再参加するメンバーの可能性を高めるために、次の 2 つの設定を使用できます:

- [group_replication_member_expel_timeout](#) では、疑いの作成 (最初の 5 秒間の検出期間の後に発生) とメンバーの削除の間の時間が長くなります。最大 1 時間の待機期間を設定できます。MySQL 8.0.21 からは、待機期間はデフォルトで 5 秒に設定されます。
- [group_replication_autorejoin_tries](#) では、メンバーは強制タイムアウトまたは到達不能な大多数のタイムアウトの後にグループへの再参加を試行します。メンバーは、指定された数の自動再結合試行を 5 分間隔で行います。MySQL 8.0.21 からは、この機能はデフォルトでアクティブ化され、メンバーは 3 回の自動再結合を試行します。

サーバーがグループから削除され、自動再結合の試行が成功しない場合は、再度参加する必要があります。つまり、サーバーがグループから明示的に削除された後、手動で再結合する (またはスクリプトで自動的に再結合する) 必要があります。

メンバーがグループから除外されるのはいつですか。

メンバーがサイレントになると、他のメンバーはグループ構成からメンバーを削除します。実際には、これはメンバーがクラッシュしたか、ネットワークが切断された場合に発生することがあります。

指定されたメンバーの指定されたタイムアウトが経過し、サイレントメンバーが作成されずに新しい構成が作成されると、障害が検出されます。

1 つのノードが大幅に遅れているとどうなりますか。

メンバーをグループから自動的に削除するタイミングのポリシーを定義する方法はありません。メンバーが遅れている理由を確認し、それを修正するか、グループからメンバーを削除する必要があります。そうでない場合、サーバーが非常に低速でフロー制御をトリガーすると、グループ全体も低速になります。フロー制御は、必要に応じて構成できます。

グループ内の問題が疑われる場合、再構成のトリガーを担当する特別なメンバーはいますか。

いいえ。再構成のトリガーを担当する特別なメンバーはグループにありません。

どのメンバーも問題がある疑いがあります。すべてのメンバーは、特定のメンバーが失敗したことを (自動的に) 同意する必要があります。1 人のメンバーが、再構成をトリガーすることでグループから明示的に削除されます。どのメンバーがメンバーの明示を担当しているかは、ユーザーが制御または設定できないものです。

シャーディングにグループレプリケーションを使用できますか。

グループレプリケーションは、可用性の高いレプリカセットを提供するように設計されています。データおよび書き込みは、グループ内の各メンバーで複製されます。単一システムが提供できるものを越えるスケーリングのために、オーケストレーションおよびシャーディングフレームワークが多数のグループレプリケーションセットを中心に構築されている必要があります。各レプリカセットは、データセット全体の特定のシャードまたはパーティションを保持および管理します。このタイプの設定（「シャードクラスタ」とも呼ばれる）では、読み取りおよび書き込みを制限なしで線形的にスケーリングできます。

SELinux でグループレプリケーションを使用するにはどうすればよいですか。

`sestatus -v` を使用して検証できる SELinux が有効になっている場合は、Group Replication 通信ポートの使用を有効にする必要があります。グループレプリケーションの TCP ポートコンテキストの設定を参照してください。

iptables でグループレプリケーションを使用するにはどうすればよいですか。

`iptables` が有効な場合は、マシン間の通信用にグループレプリケーションポートを開く必要があります。各マシンに設定されている現在のルールを確認するには、`iptables -L` を発行します。構成されているポートが 33061 の場合は、`iptables -A INPUT -p tcp --dport 33061 -j ACCEPT` を発行して、必要なポート経由の通信を有効にします。

グループメンバーが使用するレプリケーションチャンネルのリレーログをリカバリするにはどうすればよいですか。

グループレプリケーションで使用されるレプリケーションチャンネルは、レプリカレプリケーションへの非同期ソースで使用されるレプリケーションチャンネルと同じように動作し、リレーログに依存します。`relay_log` 変数が変更された場合、またはオプションが設定されておらず、ホスト名が変更された場合は、エラーが発生する可能性があります。この状況でのリカバリ手順については、セクション 17.2.4.1 「リレーログ」を参照してください。または、Group Replication で特に問題を修正する別の方法は、`STOP GROUP REPLICATION` ステートメントを発行してから、`START GROUP REPLICATION` ステートメントを発行してインスタンスを再起動することです。Group Replication プラグインは、`group_replication_applier` チャンネルを再度作成します。

グループレプリケーションで 2 つのバインドアドレスが使用されるのはなぜですか。

グループレプリケーションでは、クライアントがメンバーと通信するために使用する SQL アドレスと、通信するためにグループメンバーによって内部的に使用される `group_replication_local_address` の間でネットワークトラフィックを分割するために、2 つのバインドアドレスが使用されます。たとえば、ネットワークアドレス `203.0.113.1` および `198.51.100.179` に 2 つのネットワークインタフェースが割り当てられているサーバーがあるとします。このような状況では、`group_replication_local_address=203.0.113.1:33061` を設定して、内部グループのネットワークアドレスに `203.0.113.1:33061` を使用できます。その後、`198.51.100.179 for hostname` および `3306 for the port` を使用できます。その後、クライアント SQL アプリケーションは `198.51.100.179:3306` のメンバーに接続します。これにより、ネットワークごとに異なるルールを構成できます。同様に、内部グループ通信をクライアントアプリケーションに使用されるネットワーク接続と分離して、セキュリティを向上させることができます。

グループレプリケーションでは、ネットワークアドレスとホスト名はどのように使用されますか。

グループレプリケーションではメンバー間のネットワーク接続が使用されるため、その機能はホスト名とポートの構成方法によって直接影響を受けます。たとえば、Group Replication 分散リカバリプロセスでは、サーバーのホスト名とポートを使用して既存のグループメンバーへの接続が作成されます。メンバーがグループに参加すると、`performance_schema.replication_group_members` にリストされているネットワークアドレス情報を使用してグループメンバーシップ情報を受け取ります。そのテーブルにリストされているメンバーのいずれかが、グループから結合メンバーへの欠落データのドナーとして選択されます。

つまり、SQL ネットワークアドレスやグループシードアドレスなど、ホスト名を使用して構成する値は、完全修飾名であり、グループの各メンバーによって解決可能である必要があります。たとえば、DNS、正しく構成された `/etc/`

`hosts` ファイルまたはその他のローカルプロセスを使用して、これを確認できます。サーバーで `MEMBER_HOST` 値を構成する場合は、グループに参加する前に、サーバーで `--report-host` オプションを使用して指定します。

重要

割り当てられた値は直接使用され、`skip_name_resolve` システム変数の影響を受けません。

サーバーで `MEMBER_PORT` を構成するには、`report_port` システム変数を使用して指定します。

サーバーの自動増分設定が変更されたのはなぜですか。

グループレプリケーションがサーバーで開始されると、`auto_increment_increment` の値は `group_replication_auto_increment_increment` の値に変更され (デフォルトは 7)、`auto_increment_offset` の値はサーバー ID に変更されます。グループレプリケーションが停止すると、変更は元に戻されます。これらの設定では、グループメンバーに対する書き込みに対して重複する自動増分値が選択されないため、トランザクションがロールバックされます。グループレプリケーションのデフォルトの自動増分値 7 は、使用可能な値の数とレプリケーショングループの最大許容サイズ (9 メンバー) のバランスを表します。

変更は、`auto_increment_increment` および `auto_increment_offset` のそれぞれのデフォルト値が 1 の場合にのみ行われ、元に戻されます。これらの値がすでにデフォルトから変更されている場合、Group Replication はそれらを変更しません。MySQL 8.0 からは、グループレプリケーションが単一プライマリモードで、サーバー書き込みが 1 つのみの場合も、システム変数は変更されません。

プライマリを見つけるにはどうすればよいですか。

グループがシングルプライマリモードで動作している場合は、どのメンバーがプライマリであるかを確認すると役立ちます。 [プライマリの検索](#) を参照してください

第 19 章 MySQL Shell

MySQL Shell は、MySQL Server の高度なクライアントおよびコードエディタです。mysql と同様に、提供される SQL 機能に加えて、MySQL Shell には JavaScript および Python 用のスクリプト機能が用意されており、MySQL を操作するための API が含まれています。MySQL Shell は、個別にインストールできるコンポーネントです。

次の説明では、MySQL Shell の機能について簡単に説明します。詳細は、<https://dev.mysql.com/doc/mysql-shell/en/> で入手可能な MySQL Shell のマニュアルを参照してください。

MySQL Shell には、MySQL と対話するコードの開発に使用できる、JavaScript および Python に実装された次の API が含まれています。

- X DevAPI を使用すると、MySQL Shell が X プロトコル を使用して MySQL サーバーに接続している場合に、開発者はリレーショナルデータとドキュメントデータの両方を操作できます。これにより、MySQL をドキュメントストア (「NoSQL の使用」とも呼ばれる) として使用できます。詳細は、[第20章「ドキュメントストアとしての MySQL の使用」](#) を参照してください。MySQL Shell に実装されている X DevAPI の概念および使用方法に関するドキュメントは、[X DevAPI User Guide](#) を参照してください。
- AdminAPI を使用すると、データベース管理者は、高度な MySQL の専門知識を必要とせずに、InnoDB ベースの MySQL データベースを使用して高可用性およびスケーラビリティのための統合ソリューションを提供する InnoDB クラスターと連携できます。AdminAPI には、InnoDB クラスターと同様の方法で非同期 GTID ベースレプリケーションを実行する MySQL インスタンスのセットを管理できるようにする InnoDB ReplicaSet のサポートも含まれています。また、AdminAPI を使用すると、InnoDB クラスターと InnoDB ReplicaSet の両方との統合など、MySQL Router の管理が容易になります。[MySQL AdminAPI の使用](#) を参照してください。

MySQL Shell は、Community Edition と Commercial Edition の 2 つのエディションで使用できます。Community Edition は無料で使用できます。Commercial Edition は、追加のエンタープライズ機能を低コストで提供します。

第 20 章 ドキュメントストアとしての MySQL の使用

目次

20.1 MySQL ドキュメントストアへのインターフェース	3374
20.2 ドキュメントストアの概念	3374
20.3 JavaScript クイックスタートガイド: ドキュメントストア用の MySQL Shell	3375
20.3.1 MySQL Shell	3376
20.3.2 world_x データベースのダウンロードおよびインポート	3377
20.3.3 ドキュメントとコレクション	3378
20.3.4 リレーショナルテーブル	3388
20.3.5 テーブル内のドキュメント	3393
20.4 Python クイックスタートガイド: ドキュメントストア用の MySQL Shell	3394
20.4.1 MySQL Shell	3395
20.4.2 world_x データベースのダウンロードおよびインポート	3396
20.4.3 ドキュメントとコレクション	3397
20.4.4 リレーショナルテーブル	3407
20.4.5 テーブル内のドキュメント	3412
20.5 X プラグイン	3413
20.5.1 X プラグイン のインストールの確認	3413
20.5.2 X プラグイン の無効化	3414
20.5.3 X プラグイン での暗号化接続の使用	3414
20.5.4 Caching SHA-2 認証プラグインでの X プラグイン の使用	3415
20.5.5 X プラグイン での接続圧縮	3415
20.5.6 X プラグイン のオプションおよび変数	3418
20.5.7 X プラグイン の監視	3437

この章では、MySQL をドキュメントストア (「NoSQL の使用」とも呼ばれる) として使用する別の方法について説明します。従来の (SQL) 方法で MySQL を使用する場合は、この章はおそらく関連がありません。

従来、MySQL などのリレーショナルデータベースでは、通常、ドキュメントを格納する前にスキーマを定義する必要があります。このセクションで説明する機能を使用すると、MySQL をドキュメントストアとして使用できます。ドキュメントストアはスキーマレスであるため、ドキュメントのスキーマ対応の記憶域システムです。たとえば、製品を説明するドキュメントを作成する場合、ドキュメントを格納して操作する前に、製品の使用可能なすべての属性を把握して定義する必要はありません。これは、製品をデータベースに追加する前にテーブルのすべてのカラムを認識して定義する必要がある場合に、リレーショナルデータベースでの作業およびテーブルへの製品の格納とは異なります。この章で説明する機能を使用すると、MySQL の構成方法、ドキュメントストアモデルのみを使用する方法、またはドキュメントストアモデルの柔軟性とリレーショナルモデルの機能を組み合わせる方法を選択できます。

MySQL をドキュメントストアとして使用するには、次のサーバー機能を使用します:

- X プラグイン を使用すると、MySQL Server は、MySQL をドキュメントストアとして使用するための前提条件である X プロトコル を使用してクライアントと通信できます。X プラグイン は、MySQL 8.0 の時点で MySQL Server でデフォルトで有効になっています。X プラグインのインストールを検証し、X プラグインを構成およびモニターする手順については、[セクション20.5「X プラグイン」](#) を参照してください。
- X プロトコル では、SASL を介した認証である CRUD 操作と SQL 操作の両方がサポートされており、コマンドのストリーミング (パイプライン) が可能であり、プロトコルおよびメッセージレイヤーで拡張可能です。X プロトコルと互換性のあるクライアントには、MySQL Shell および MySQL 8.0 コネクタが含まれます。
- X プロトコル を使用して MySQL Server と通信するクライアントは、X DevAPI を使用してアプリケーションを開発できます。X DevAPI は、確立された業界標準概念をサポートするシンプルで強力な設計を備えた最新のプログラミングインターフェースを提供します。この章では、MySQL Shell で X DevAPI の JavaScript または Python 実装をクライアントとして使用する方法について説明します。X DevAPI の使用に関する詳細なチュートリアルは、[X DevAPI User Guide](#) を参照してください。

20.1 MySQL ドキュメントストアへのインタフェース

MySQL をドキュメントストアとして使用するには、専用コンポーネントと、MySQL サーバーとの通信をサポートするクライアントの選択肢を使用して、ドキュメントベースのアプリケーションを開発します。

- 次の MySQL 製品は X プロトコル をサポートしており、選択した言語で X DevAPI を使用して、ドキュメントストアとして機能する MySQL Server と通信するアプリケーションを開発できます:
 - MySQL Shell (JavaScript および Python での X DevAPI の実装を提供)
 - Connector/C++
 - Connector/J
 - Connector/Node.js
 - Connector/NET
 - Connector/Python
- MySQL Shell は、JavaScript、Python または SQL モードをサポートする MySQL への対話型インタフェースです。MySQL Shell を使用して、アプリケーションのプロトタイプ作成、クエリーの実行およびデータの更新を行うことができます。[MySQL Shell のインストール](#) には、MySQL Shell をダウンロードしてインストールする手順があります。
- この章のクイックスタートガイド (チュートリアル) は、MySQL をドキュメントストアとして MySQL Shell の使用を開始するのに役立ちます。

JavaScript のクイックスタートガイドはここにあります: [セクション20.3 「JavaScript クイックスタートガイド: ドキュメントストア用の MySQL Shell」](#)。

Python のクイックスタートガイドはここにあります: [セクション20.4 「Python クイックスタートガイド: ドキュメントストア用の MySQL Shell」](#)。

- MySQL Shell の構成および使用の詳細は、[MySQL Shell 8.0](#) の「MySQL Shell ユーザーガイド」を参照してください。

20.2 ドキュメントストアの概念

このセクションでは、MySQL をドキュメントストアとして使用する際に導入される概念について説明します。

- [JSON ドキュメント](#)
- [コレクション](#)
- [CRUD 操作](#)

JSON ドキュメント

JSON ドキュメントは、キーと値のペアで構成されるデータ構造であり、MySQL をドキュメントストアとして使用するための基本的な構造です。たとえば、world_x スキーマ (この章の後半でインストール) には次のドキュメントが含まれています:

```
{
  "GNP": .6,
  "IndepYear": 1967,
  "Name": "Sealand",
  "_id": "SEA",
  "demographics": {
    "LifeExpectancy": 79,
    "Population": 27
  },
  "geography": {
    "Continent": "Europe",
```

```
"Region": "British Islands",
"SurfaceArea": 193
},
"government": {
  "GovernmentForm": "Monarchy",
  "HeadOfState": "Michael Bates"
}
}
```

このドキュメントでは、キーの値は整数や文字列などの単純なデータ型にできますが、他のドキュメント、配列およびドキュメントのリストを含めることもできます。たとえば、[geography](#) キー値は複数のキーと値のペアで構成されます。JSON ドキュメントは、[JSON MySQL データ型](#)を介して、MySQL バイナリ JSON オブジェクトを使用して内部的に表されます。

ドキュメントと従来のリレーショナルデータベースから認識されるテーブルの最も重要な違いは、ドキュメントの構造を事前に定義する必要がなく、コレクションに異なる構造を持つ複数のドキュメントを含めることができることです。一方、リレーショナルテーブルでは構造を定義する必要があり、テーブルのすべての行に同じカラムが含まれている必要があります。

コレクション

コレクションは、JSON ドキュメントを MySQL データベースに格納するために使用されるコンテナです。アプリケーションは通常、特定のドキュメントを検索するなど、ドキュメントのコレクションに対して操作を実行します。

CRUD 操作

コレクションに対して発行できる 4 つの基本操作は、作成、読取り、更新および削除 (CRUD) です。MySQL の場合、これは次のことを意味します:

- 新規ドキュメントの作成 (挿入または追加)
- 1 つまたは複数のドキュメントを読む (クエリー)
- 1 つまたは複数のドキュメントの更新
- 1 つまたは複数のドキュメントを削除

20.3 JavaScript クイックスタートガイド: ドキュメントストア用の MySQL Shell

このクイックスタートガイドでは、MySQL Shell と対話形式でドキュメントストアアプリケーションのプロトタイプ作成を開始する手順について説明します。このガイドの内容は次のとおりです:

- MySQL 機能、MySQL Shell および [world_x](#) サンプルスキーマの概要。
- コレクションおよびドキュメントを管理する操作。
- リレーショナルテーブルを管理する操作。
- テーブル内のドキュメントに適用される操作。

このクイックスタートガイドに従うには、X プラグイン がインストールされた MySQL サーバー、8.0 のデフォルト、および MySQL Shell をクライアントとして使用する必要があります。[MySQL Shell 8.0](#) は、MySQL Shell に関する詳細情報を提供します。ドキュメントストアには X DevAPI を使用してアクセスし、MySQL Shell は JavaScript と Python の両方でこの API を提供します。

関連情報

- [MySQL Shell 8.0](#) は、MySQL Shell に関する詳細情報を提供します。
- このクイックスタートガイドで使用されるツールの詳細は、[MySQL Shell のインストール](#) および [セクション 20.5「X プラグイン」](#) を参照してください。

- [X DevAPI User Guide](#) には、X DevAPI を使用してドキュメントストアを使用するアプリケーションを開発するその他の例が用意されています。
- [Python クイックスタートガイド](#) も使用できます。

20.3.1 MySQL Shell

このクイックスタートガイドでは、MySQL Shell について一定レベルの知識があることを前提としています。次のセクションは概要です。詳細は、MySQL Shell のドキュメントを参照してください。MySQL Shell は、MySQL Server への統一されたスクリプトインタフェースです。JavaScript および Python でのスクリプトがサポートされています。JavaScript はデフォルトの処理モードです。

MySQL Shell の起動

MySQL サーバーをインストールして起動したら、MySQL Shell をサーバーインスタンスに接続します。接続する予定の MySQL サーバーインスタンスのアドレスを知っている必要があります。インスタンスをドキュメントストアとして使用できるようにするには、サーバーインスタンスに X プラグイン がインストールされている必要があり、X プロトコル を使用してサーバーに接続する必要があります。たとえば、33060 のデフォルトの X プロトコル ポートでインスタンス `ds1.example.com` に接続するには、ネットワーク文字列 `user@ds1.example.com:33060` を使用します。

ヒント

`mysqlx_port` のかわりに 3306 のデフォルトの `port` を使用するなどして、クラシック MySQL プロトコル を使用してインスタンスに接続する場合、このチュートリアルに示すドキュメントストア機能は使用できません。たとえば、`db` グローバルオブジェクトは移入されません。ドキュメントストアを使用するには、常に X プロトコル を使用して接続します。

MySQL Shell がまだ実行されていない場合は、ターミナルウィンドウを開き、次のコマンドを発行します:

```
mysqlsh user@ds1.example.com:33060/world_x
```

または、MySQL Shell がすでに実行されている場合は、次のコマンドを発行して `\connect` コマンドを使用します:

```
\connect user@ds1.example.com:33060/world_x
```

MySQL Shell を接続する MySQL サーバーインスタンスのアドレスを指定する必要があります。たとえば、前の例では次のようになります:

- `user` は、MySQL アカウントのユーザー名を表します。
- `ds1.example.com` は、MySQL を実行しているサーバーインスタンスのホスト名です。これを、ドキュメントストアとして使用している MySQL サーバーインスタンスのホスト名に置き換えます。
- このセッションのデフォルトスキーマは `world_x` です。 `world_x` スキーマの設定手順は、[セクション 20.3.2 「world_x データベースのダウンロードおよびインポート」](#) を参照してください。

詳細は、[セクション 4.2.5 「URI 類似文字列またはキーと値のペアを使用したサーバーへの接続」](#) を参照してください。

MySQL Shell が開くと、`mysql-js>` プロンプトに、このセッションのアクティブな言語が JavaScript であることが示されます。

```
mysql-js>
```

MySQL Shell は、次のような入力行編集をサポートしています:

- left-arrow および right-arrow キーは、現在の入力行内で水平に移動します。
- up-arrow および down-arrow のキーは、以前に入力した一連の行を上下に移動します。
- 「バックスペース」でカーソルの前の文字を削除でき、新しい文字を入力するとカーソルの位置に挿入されます。
- Enter は、現在の入力行をサーバーに送信します。

MySQL Shell のヘルプの表示

コマンドラインオプションのリストを表示するには、コマンドインタプリタのプロンプトで `mysqlsh --help` と入力します。

```
mysqlsh --help
```

使用可能なコマンドとその説明のリストを表示するには、MySQL Shell プロンプトで `\help` と入力します。

```
mysql-js> \help
```

個々の MySQL Shell コマンドの詳細なヘルプを表示するには、`\help` に続けてコマンド名を入力します。たとえば、`\connect` コマンドのヘルプを表示するには、次のコマンドを発行します：

```
mysql-js> \help \connect
```

MySQL Shell の終了

MySQL Shell を終了するには、次のコマンドを発行します：

```
mysql-js> \quit
```

関連情報

- MySQL Shell での対話型コード実行の動作については、[対話型コードの実行](#) を参照してください。
- セッションおよび接続の代替方法の詳細は、[MySQL Shell スタートガイド](#) を参照してください。

20.3.2 world_x データベースのダウンロードおよびインポート

このクイックスタートガイドの一部として、`world_x` スキーマと呼ばれるサンプルスキーマが用意されています。これらの例の多くは、このスキーマを使用するドキュメントストア機能を示しています。`world_x` スキーマをロードできるように MySQL サーバーを起動し、次のステップに従います：

1. `world_x-db.zip` のダウンロード。
2. `/tmp/` などの一時的な場所にインストールアーカイブを抽出します。アーカイブを解凍すると、`world_x.sql` という名前の単一ファイルが作成されます。
3. `world_x.sql` ファイルをサーバーにインポートします。次のいずれかを実行できます：

- MySQL Shell を SQL モードで起動し、次のコマンドを発行してファイルをインポートします：

```
mysqlsh -u root --sql --file /tmp/world_x-db/world_x.sql  
Enter password: ****
```

- 実行中に MySQL Shell を SQL モードに設定し、次のコマンドを発行してスキーマファイルをソーシングします：

```
\sql  
Switching to SQL mode... Commands end with ;  
\source /tmp/world_x-db/world_x.sql
```

`/tmp/` をシステム上の `world_x.sql` ファイルへのパスに置き換えます。プロンプトが表示されたら、パスワードを入力します。root 以外のアカウントは、そのアカウントが新しいスキーマを作成する権限を持っているかぎり使用できます。

world_x スキーマ

`world_x` サンプルスキーマには、次の JSON コレクションおよびリレーショナルテーブルが含まれています：

- コレクション
 - `countryinfo`: 世界中の国に関する情報。
- テーブル

- **country**: 世界の国に関する最小限の情報。
- **city**: これらの国の一部の都市に関する情報。
- **countrylanguage**: 各国で話されている言語。

関連情報

- [MySQL Shell セッション](#) では、セッションタイプについて説明します。

20.3.3 ドキュメントとコレクション

MySQL をドキュメントストアとして使用している場合、コレクションは、作成、リストおよびドロップできるスキーマ内のコンテナです。コレクションには、追加、検索、更新および削除できる JSON ドキュメントが含まれます。

このセクションの例では、`world_x` スキーマの `countryinfo` コレクションを使用します。`world_x` スキーマの設定手順は、[セクション20.3.2「world_x データベースのダウンロードおよびインポート」](#) を参照してください。

Documents

MySQL では、ドキュメントは JSON オブジェクトとして表されます。内部的には、高速ルックアップおよび更新を可能にする効率的なバイナリ形式で格納されます。

- JavaScript の単純なドキュメント形式:

```
{field1: "value", field2 : 10, "field 3": null}
```

ドキュメントの配列は、カンマで区切られた一連のドキュメントで構成され、`[および]`文字で囲まれます。

- JavaScript のドキュメントの単純な配列:

```
[{"Name": "Aruba", "Code": "ABW"}, {"Name": "Angola", "Code": "AGO"}]
```

MySQL は、JSON ドキュメントで次の JavaScript 値タイプをサポートしています:

- 数値 (整数および浮動小数点)
- 文字列
- boolean (False および True)
- null
- JSON 値の配列
- より多くの JSON 値のネストされた (埋込み) オブジェクト

コレクション

コレクションは、目的を共有し、場合によっては 1 つ以上のインデックスを共有するドキュメントのコンテナです。各コレクションは一意的な名前を持ち、単一のスキーマ内に存在します。

スキーマという用語はデータベースと同等です。つまり、リレーショナルスキーマとは対照的に、データベースオブジェクトのグループであり、データの構造と制約を施行するために使用されます。スキーマは、コレクション内のドキュメントへの準拠を強制しません。

このクイックスタートガイドの内容は次のとおりです:

- 基本オブジェクトには次のものがあります:

オブジェクトフォーム	説明
<code>db</code>	<code>db</code> は、現在アクティブなスキーマに割り当てられたグローバル変数です。コレクションの取得など、スキーマ

オブジェクトフォーム	説明
	に対して操作を実行する場合は、 <code>db</code> 変数に使用可能なメソッドを使用します。
<code>db.getCollections()</code>	<code>db.getCollections()</code> は、スキーマ内のコレクションのリストを返します。このリストを使用して、コレクションオブジェクトへの参照の取得、コレクションオブジェクトの反復などを行います。

- コレクションによってスコープ指定される基本的な操作は次のとおりです:

操作フォーム	説明
<code>db.name.add()</code>	<code>add()</code> メソッドは、ドキュメントまたはドキュメントのリストを名前付きコレクションに挿入します。
<code>db.name.find()</code>	<code>find()</code> メソッドは、名前付きコレクション内の一部またはすべてのドキュメントを返します。
<code>db.name.modify()</code>	<code>modify()</code> メソッドは、名前付きコレクション内のドキュメントを更新します。
<code>db.name.remove()</code>	<code>remove()</code> メソッドは、名前付きコレクションからドキュメントまたはドキュメントのリストを削除します。

関連情報

- 一般的な概要は、[Working with Collections](#) を参照してください。
- [CRUD EBNF Definitions](#) では、操作の完全なリストが提供されます。

20.3.3.1 コレクションの作成、リストおよびドロップ

MySQL Shell では、新しいコレクションを作成し、スキーマ内の既存のコレクションのリストを取得し、スキーマから既存のコレクションを削除できます。コレクション名は大/小文字が区別され、各コレクション名は一意である必要があります。

スキーマの確認

スキーマ変数に割り当てられている値を表示するには、次のコマンドを発行します:

```
mysql-js> db
```

スキーマ値が `Schema:world_x` でない場合は、次のコマンドを発行して `db` 変数を設定します:

```
mysql-js> \use world_x
```

コレクションの作成

既存のスキーマに新しいコレクションを作成するには、`db` オブジェクトの `createCollection()` メソッドを使用します。次の例では、`world_x` スキーマに `flags` というコレクションを作成します。

```
mysql-js> db.createCollection("flags")
```

このメソッドはコレクションオブジェクトを返します。

```
<Collection:flags>
```

コレクションのリスト

`world_x` スキーマ内のすべてのコレクションを表示するには、`db` オブジェクトの `getCollections()` メソッドを使用します。現在接続しているサーバーによって返されるコレクションは、大カッコ内に表示されます。

```
mysql-js> db.getCollections()
[
  <Collection:countryinfo>
```

```
<Collection:flags>
]
```

コレクションのドロップ

スキーマから既存のコレクションを削除するには、`db` オブジェクトの `dropCollection()` メソッドを使用します。たとえば、現行のスキーマから `flags` コレクションを削除するには、次のコマンドを発行します:

```
mysql-js> db.dropCollection("flags")
```

`dropCollection()` メソッドは、スキーマからリレーショナルテーブルを削除するために MySQL Shell でも使用されません。

関連情報

- その他の例については、[Collection Objects](#) を参照してください。

20.3.3.2 コレクションの操作

スキーマ内のコレクションを操作するには、`db` グローバルオブジェクトを使用して現在のスキーマにアクセスします。この例では、以前にインポートした `world_x` スキーマと `countryinfo` コレクションを使用しています。したがって、発行する操作の形式は `db.collection_name.operation` で、`collection_name` は操作の実行対象となるコレクションの名前です。次の例では、操作は `countryinfo` コレクションに対して実行されます。

ドキュメントの追加

`add()` メソッドを使用して、あるドキュメントまたはドキュメントのリストを既存のコレクションに挿入します。次のドキュメントを `countryinfo` コレクションに挿入します。これは複数行コンテンツであるため、Enter を 2 回押してドキュメントを挿入します。

```
mysql-js> db.countryinfo.add(
{
  GNP: .6,
  IndepYear: 1967,
  Name: "Sealand",
  Code: "SEA",
  demographics: {
    LifeExpectancy: 79,
    Population: 27
  },
  geography: {
    Continent: "Europe",
    Region: "British Islands",
    SurfaceArea: 193
  },
  government: {
    GovernmentForm: "Monarchy",
    HeadOfState: "Michael Bates"
  }
}
)
```

このメソッドは、操作のステータスを返します。ドキュメントを検索して操作を確認できます。例:

```
mysql-js> db.countryinfo.find("Name = 'Sealand'")
{
  "GNP": 0.6,
  "_id": "00005e2ff4af0000000000000000f4",
  "Name": "Sealand",
  "Code": "SEA",
  "IndepYear": 1967,
  "geography": {
    "Region": "British Islands",
    "Continent": "Europe",
    "SurfaceArea": 193
  },
  "government": {
    "HeadOfState": "Michael Bates",
    "GovernmentForm": "Monarchy"
  }
}
```

```
},
"demographics": {
  "Population": 27,
  "LifeExpectancy": 79
}
}
```

ドキュメントの追加時に指定されたフィールドに加えて、`_id` というフィールドもあります。各ドキュメントには、`_id` と呼ばれる識別子フィールドが必要です。`_id` フィールドの値は、同じコレクション内のすべてのドキュメント間で一意である必要があります。MySQL 8.0.11 以上では、ドキュメント ID はクライアントではなくサーバーによって生成されるため、MySQL Shell は `_id` 値を自動的に設定しません。8.0.11 以上の MySQL サーバーでは、ドキュメントに `_id` フィールドが含まれていない場合、`_id` 値が設定されます。以前の 8.0 リリースまたは 5.7 の MySQL サーバーは、この状況では `_id` 値を設定しないため、明示的に指定する必要があります。そうでない場合、MySQL Shell はエラー 5115 「文書に必須フィールドがありません」を返します。詳細は、[Understanding Document IDs](#) を参照してください。

関連情報

- 完全な構文の定義は、[CollectionAddFunction](#) を参照してください。
- [Understanding Document IDs](#) を参照してください。

20.3.3.3 ドキュメントの検索

`find()` メソッドを使用して、スキーマ内のコレクションに対してクエリーを実行し、ドキュメントを戻すことができます。MySQL Shell には、返されたドキュメントをフィルタおよびソートするために `find()` メソッドとともに使用する追加のメソッドが用意されています。

MySQL には、検索条件を指定する次の演算子が用意されています: `OR (||)`、`AND (&&)`、`XOR (IS)`、`NOT`、`BETWEEN`、`IN`、`LIKE`、`!=`、`<>`、`>`、`>=`、`<`、`<=`、`&`、`|`、`<<`、`>>`、`+`、`-`、`*`、`/`、`~` および `%`。

コレクション内のすべてのドキュメントの検索

コレクション内のすべてのドキュメントを返すには、検索条件を指定せずに `find()` メソッドを使用します。たとえば、次の操作では、`countryinfo` コレクション内のすべてのドキュメントが返されます。

```
mysql-js> db.countryinfo.find()
[
  {
    "GNP": 828,
    "Code": "ABW",
    "Name": "Aruba",
    "IndepYear": null,
    "geography": {
      "Continent": "North America",
      "Region": "Caribbean",
      "SurfaceArea": 193
    },
    "government": {
      "GovernmentForm": "Nonmetropolitan Territory of The Netherlands",
      "HeadOfState": "Beatrix"
    }
  },
  "demographics": {
    "LifeExpectancy": 78.4000015258789,
    "Population": 103000
  },
  ...
]
240 documents in set (0.00 sec)
```

このメソッドは、コレクション内のすべてのドキュメントに加えて、操作情報を含む結果を生成します。

空のセット（一致するドキュメントがない）は、次の情報を返します:

```
Empty set (0.00 sec)
```

フィルタ検索

`find()` メソッドに検索条件を含めることができます。検索条件を形成する式の構文は、従来の MySQL 第12章「関数と演算子」の構文と同じです。すべての式は引用符で囲む必要があります。簡潔にするために、一部の例では出力が表示されません。

単純検索条件は、`Name` フィールドとドキュメント内にあることがわかっている値で構成できます。次の例では、単一のドキュメントを戻します:

```
mysql-js> db.countryinfo.find("Name = 'Australia'")
[
  {
    "GNP": 351182,
    "Code": "AUS",
    "Name": "Australia",
    "IndepYear": 1901,
    "geography": {
      "Continent": "Oceania",
      "Region": "Australia and New Zealand",
      "SurfaceArea": 7741220
    },
    "government": {
      "GovernmentForm": "Constitutional Monarchy, Federation",
      "HeadOfState": "Elisabeth II"
    },
    "demographics": {
      "LifeExpectancy": 79.80000305175781,
      "Population": 18886000
    },
  },
]
```

次の例では、GNP が 00 億を超えるすべての国を検索します。 `countryinfo` コレクションは GNP を百万単位で測定します。

```
mysql-js> db.countryinfo.find("GNP > 500000")
...[output removed]
10 documents in set (0.00 sec)
```

次のクエリーの「人口」フィールドは、人口統計オブジェクト内に埋め込まれます。埋込みフィールドにアクセスするには、人口統計と人口間のピリオドを使用して関係を識別します。ドキュメント名とフィールド名では大文字と小文字が区別されます。

```
mysql-js> db.countryinfo.find("GNP > 500000 and demographics.Population < 100000000")
...[output removed]
6 documents in set (0.00 sec)
```

次の式の算術演算子は、大文字当たりの GNP が 0000 を超える国をクエリーするために使用されます。検索条件には、算術演算子およびほとんどの MySQL 関数を含めることができます。

注記

`countryinfo` コレクション内の 7 つのドキュメントの母集団値はゼロです。したがって、出力の最後に警告メッセージが表示されます。

```
mysql-js> db.countryinfo.find("GNP*1000000/demographics.Population > 30000")
...[output removed]
9 documents in set, 7 warnings (0.00 sec)
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
```

`bind()` メソッドを使用して、値を検索条件から区切ることができます。たとえば、ハードコードされた国名を条件として指定するかわりに、コロンで構成される名前付きプレースホルダの後に、`country` などの文字で始まる名前を付けます。次に、`bind(placeholder, value)` メソッドを次のように使用します:

```
mysql-js> db.countryinfo.find("Name = :country").bind("country", "Italy")
```



```
{
  "GNP": 1161755,
  "_id": "00005de917d8000000000000006a",
  "Code": "ITA",
  "Name": "Italy",
  "Airports": [],
  "IndepYear": 1861,
  "geography": {
    "Region": "Southern Europe",
    "Continent": "Europe",
    "SurfaceArea": 301316
  },
  "government": {
    "HeadOfState": "Carlo Azeglio Ciampi",
    "GovernmentForm": "Republic"
  },
  "demographics": {
    "Population": 57680000,
    "LifeExpectancy": 79
  }
}
1 document in set (0.01 sec)
```

ヒント

プログラム内でバインディングを使用すると、式にプレースホルダを指定できます。プレースホルダには、実行前に値が入力され、必要に応じて自動エスケープを利用できます。

入力をサニタイズするには、常にバインディングを使用します。文字列連結を使用してクエリーに値を導入しないでください。無効な入力生成され、場合によってはセキュリティの問題が発生することがあります。

プレースホルダおよび `bind()` メソッドを使用して保存済検索を作成し、別の値でコールできます。たとえば、国の保存済検索を作成するには、次のようにします:

```
mysql-js> var myFind = db.countryinfo.find("Name = :country")
mysql-js> myFind.bind('country', 'France')
{
  "GNP": 1424285,
  "_id": "00005de917d80000000000000048",
  "Code": "FRA",
  "Name": "France",
  "IndepYear": 843,
  "geography": {
    "Region": "Western Europe",
    "Continent": "Europe",
    "SurfaceArea": 551500
  },
  "government": {
    "HeadOfState": "Jacques Chirac",
    "GovernmentForm": "Republic"
  },
  "demographics": {
    "Population": 59225700,
    "LifeExpectancy": 78.80000305175781
  }
}
1 document in set (0.0028 sec)

mysql-js> myFind.bind('country', 'Germany')
{
  "GNP": 2133367,
  "_id": "00005de917d80000000000000038",
  "Code": "DEU",
  "Name": "Germany",
  "IndepYear": 1955,
  "geography": {
    "Region": "Western Europe",
    "Continent": "Europe",
    "SurfaceArea": 357022
  },
  "government": {
    "HeadOfState": "Johannes Rau",
```

```
"GovernmentForm": "Federal Republic"
},
"demographics": {
  "Population": 82164700,
  "LifeExpectancy": 77.4000015258789
}
}
}
1 document in set (0.0026 sec)
```

プロジェクト結果

すべてのフィールドを返すかわりに、ドキュメントの特定のフィールドを返すことができます。次の例では、検索条件に一致する `countryinfo` コレクション内のすべてのドキュメントの GNP および Name フィールドを返します。

`fields()` メソッドを使用して、返すフィールドのリストを渡します。

```
mysql-js> db.countryinfo.find("GNP > 5000000").fields(["GNP", "Name"])
[
  {
    "GNP": 8510700,
    "Name": "United States"
  }
]
1 document in set (0.00 sec)
```

また、返されるドキュメントを記述する式を使用して、返されるドキュメント (追加、名前変更、ネストおよび新しいフィールド値の計算) を変更できます。たとえば、次の式を使用してフィールドの名前を変更すると、2 つのドキュメントのみが返されます。

```
mysql-js> db.countryinfo.find().fields(
  mysqlx.expr("{\"Name\": upper(Name), \"GNPPerCapita\": GNP*1000000/demographics.Population}")).limit(2)
[
  {
    "Name": "ARUBA",
    "GNPPerCapita": 8038.834951456311
  },
  {
    "Name": "AFGHANISTAN",
    "GNPPerCapita": 263.0281690140845
  }
]
```

結果の制限、ソートおよびスキップ

`limit()`、`sort()` および `skip()` メソッドを適用して、`find()` メソッドによって返されるドキュメントの数と順序を管理できます。

結果セットに含めるドキュメントの数を指定するには、`limit()` メソッドの値を `find()` メソッドに追加します。次のクエリーは、`countryinfo` コレクション内の最初の 5 つのドキュメントを返します。

```
mysql-js> db.countryinfo.find().limit(5)
... [output removed]
5 documents in set (0.00 sec)
```

結果の順序を指定するには、`sort()` メソッドを `find()` メソッドに追加します。必要に応じて、降順 (`desc`) または昇順 (`asc`) 属性でソートするフィールドのリストを `sort()` メソッドに渡します。昇順がデフォルトのオーダータイプです。

たとえば、次のクエリーでは、すべてのドキュメントが `IndepYear` フィールドでソートされ、最初の 8 つのドキュメントが降順で返されます。

```
mysql-js> db.countryinfo.find().sort(["IndepYear desc"]).limit(8)
... [output removed]
8 documents in set (0.00 sec)
```

デフォルトでは、`limit()` メソッドはコレクション内の最初のドキュメントから始まります。`skip()` メソッドを使用して、開始ドキュメントを変更できます。たとえば、最初のドキュメントを無視し、条件に一致する次の 8 つのドキュメントを返すには、`skip()` メソッドに値 1 を渡します。

```
mysql-js> db.countryinfo.find().sort(["IndepYear desc"]).limit(8).skip(1)
```

```
... [output removed]
8 documents in set (0.00 sec)
```

関連情報

- [MySQL Reference Manual](#) には、関数および演算子に関する詳細なドキュメントが用意されています。
- 完全な構文の定義は、[CollectionFindFunction](#) を参照してください。

20.3.3.4 ドキュメントの変更

`modify()` メソッドを使用して、コレクション内の 1 つ以上のドキュメントを更新できます。X DevAPI には、`modify()` メソッドとともに使用して次の操作を行うための追加メソッドが用意されています:

- ドキュメント内のフィールドを設定および設定解除します。
- 配列を追加、挿入および削除します。
- 変更するドキュメントをバインド、制限およびソートします。

ドキュメントフィールドの設定および設定解除

`modify()` メソッドは、コレクションをフィルタ処理して変更するドキュメントのみを含め、指定した操作をそれらのドキュメントに適用することで機能します。

次の例では、`modify()` メソッドは検索条件を使用して変更するドキュメントを識別し、ネストされたデモグラフィックオブジェクト内の 2 つの値を `set()` メソッドで置き換えます。

```
mysql-js> db.countryinfo.modify("Code = 'SEA'").set(
  "demographics", {"LifeExpectancy": 78, "Population": 28})
```

ドキュメントを変更した後、`find()` メソッドを使用して変更を確認します。

ドキュメントからコンテンツを削除するには、`modify()` および `unset()` メソッドを使用します。たとえば、次のクエリーは、検索条件に一致するドキュメントから GNP を削除します。

```
mysql-js> db.countryinfo.modify("Name = 'Sealand'").unset("GNP")
```

`find()` メソッドを使用して変更を確認します。

```
mysql-js> db.countryinfo.find("Name = 'Sealand'")
{
  "_id": "00005e2ff4af000000000000000f4",
  "Name": "Sealand",
  "Code": "SEA",
  "IndepYear": 1967,
  "geography": {
    "Region": "British Islands",
    "Continent": "Europe",
    "SurfaceArea": 193
  },
  "government": {
    "HeadOfState": "Michael Bates",
    "GovernmentForm": "Monarchy"
  },
  "demographics": {
    "Population": 27,
    "LifeExpectancy": 79
  }
}
```

配列の追加、挿入および削除

配列フィールドに要素を追加したり、配列内の要素を挿入または削除するには、`arrayAppend()`、`arrayInsert()` または `arrayDelete()` メソッドを使用します。次の例では、国際空港のトラッキングを有効にするように `countryinfo` コレクションを変更します。

最初の例では、`modify()` および `set()` メソッドを使用して、すべてのドキュメントに新しい空港フィールドを作成します。

注意

検索条件を指定せずにドキュメントを変更する場合は注意してください。変更すると、コレクション内のすべてのドキュメントが変更されます。

```
mysql-js> db.countryinfo.modify("true").set("Airports", [])
```

空港フィールドが追加された状態で、次の例では、`arrayAppend()` メソッドを使用して新しい空港をいずれかのドキュメントに追加します。次の例の `$.Airports` は、現在のドキュメントの空港フィールドを表しています。

```
mysql-js> db.countryinfo.modify("Name = 'France'").arrayAppend("$.Airports", "ORY")
```

`find()` を使用して変更を確認します。

```
mysql-js> db.countryinfo.find("Name = 'France'")
{
  "GNP": 1424285,
  "_id": "00005de917d800000000000000048",
  "Code": "FRA",
  "Name": "France",
  "Airports": [
    "ORY"
  ],
  "IndepYear": 843,
  "geography": {
    "Region": "Western Europe",
    "Continent": "Europe",
    "SurfaceArea": 551500
  },
  "government": {
    "HeadOfState": "Jacques Chirac",
    "GovernmentForm": "Republic"
  },
  "demographics": {
    "Population": 59225700,
    "LifeExpectancy": 78.80000305175781
  }
}
```

配列内の別の位置に要素を挿入するには、`arrayInsert()` メソッドを使用して、パス式に挿入するインデックスを指定します。この場合、インデックスは 0、または配列の最初の要素です。

```
mysql-js> db.countryinfo.modify("Name = 'France'").arrayInsert("$.Airports[0]", "CDG")
```

配列から要素を削除するには、削除する要素のインデックスを `arrayDelete()` メソッドに渡す必要があります。

```
mysql-js> db.countryinfo.modify("Name = 'France'").arrayDelete("$.Airports[1]")
```

関連情報

- [MySQL Reference Manual](#) には、JSON 値の検索および変更に関係する手順が用意されています。
- 完全な構文の定義は、[CollectionModifyFunction](#) を参照してください。

20.3.3.5 ドキュメントの削除

`remove()` メソッドを使用して、スキーマ内のコレクションから一部またはすべてのドキュメントを削除できます。X DevAPI には、`remove()` メソッドとともに使用して、削除するドキュメントをフィルタおよびソートするための追加メソッドが用意されています。

条件を使用したドキュメントの削除

次の例では、検索条件を `remove()` メソッドに渡します。条件に一致するすべてのドキュメントが `countryinfo` コレクションから削除されます。この例では、1 つのドキュメントが条件に一致します。

```
mysql-js> db.countryinfo.remove("Code = 'SEA'")
```

最初のドキュメントの削除

`countryinfo` コレクションの最初のドキュメントを削除するには、値 1 を指定して `limit()` メソッドを使用します。

```
mysql-js> db.countryinfo.remove("true").limit(1)
```

オーダー内の最後の文書の削除

次の例では、国名で `countryinfo` コレクションの最後のドキュメントを削除します。

```
mysql-js> db.countryinfo.remove("true").sort({"Name desc"}).limit(1)
```

コレクション内のすべてのドキュメントの削除

コレクション内のすべてのドキュメントを削除できます。これを行うには、検索条件を指定せずに `remove("true")` メソッドを使用します。

注意

検索条件を指定せずにドキュメントを削除する場合は注意してください。このアクションは、コレクションからすべてのドキュメントを削除します。

または、`db.drop_collection('countryinfo')` 操作を使用して `countryinfo` コレクションを削除します。

関連情報

- 完全な構文の定義は、[CollectionRemoveFunction](#) を参照してください。
- `world_x` スキーマを再作成する手順は、[セクション20.3.2 「world_x データベースのダウンロードおよびインポート」](#) を参照してください。

20.3.3.6 インデックスの作成および削除

インデックスは、特定のフィールド値を持つドキュメントをすばやく検索するために使用されます。インデックスがない場合、MySQL は最初のドキュメントで始まり、コレクション全体を読み取って関連するフィールドを検索する必要があります。コレクションが大きいほど、このコストは高くなります。コレクションが大きく、特定のフィールドに対するクエリーが一般的な場合は、ドキュメント内の特定のフィールドに対するインデックスの作成を検討してください。

たとえば、移入フィールドのインデックスを使用すると、次のクエリーのパフォーマンスが向上します：

```
mysql-js> db.countryinfo.find("demographics.Population < 100")
...[output removed]
8 documents in set (0.00 sec)
```

`createIndex()` メソッドは、使用するフィールドを指定する JSON ドキュメントで定義できるインデックスを作成します。このセクションでは、インデックス付けの概要について説明します。詳細は、[Indexing Collections](#) を参照してください。

一意でないインデックスの追加

一意でないインデックスを作成するには、インデックス名とインデックス情報を `createIndex()` メソッドに渡します。重複するインデックス名は禁止されています。

次の例では、`demographics` オブジェクトの `Population` フィールドに対して定義され、`Integer` 数値としてインデックス付けされた `popul` という名前のインデックスを指定します。最後のパラメータは、フィールドに `NOT NULL` 制約が必要かどうかを示します。値が `false` の場合、フィールドには `NULL` 値を含めることができます。インデックス情報は、インデックスに含める 1 つ以上のフィールドの詳細を含む JSON ドキュメントです。各フィールド定義には、フィールドへの完全なドキュメントパスを含め、フィールドのタイプを指定する必要があります。

```
mysql-js> db.countryinfo.createIndex("popul", {fields:
```

```
[[{"field": "$.demographics.Population", "type": "INTEGER"}]]
```

ここでは、整数の数値を使用してインデックスが作成されます。GeoJSON データで使用するオプションなど、さらにオプションを使用できます。デフォルトタイプの「index」が適切であるため、ここでは省略されているインデックスのタイプを指定することもできます。

一意インデックスの追加

一意のインデックスを作成するには、インデックス名、インデックス定義およびインデックスタイプ「unique」を `createIndex()` メソッドに渡します。この例は、国名 ("Name") に対して作成された一意のインデックスを示しています。これは、インデックス付けする `countryinfo` コレクションの別の共通フィールドです。インデックスフィールドの説明で、「TEXT(40)」はインデックス付けする文字数を表し、「required': True はフィールドがドキュメントに存在する必要があることを指定します。

```
mysql-js> db.countryinfo.createIndex("name",  
{"fields": [{"field": "$.Name", "type": "TEXT(40)", "required": true}], "unique": true})
```

インデックスの削除

インデックスを削除するには、削除するインデックスの名前を `dropIndex()` メソッドに渡します。たとえば、次のように「popul」インデックスを削除できます:

```
mysql-js> db.countryinfo.dropIndex("popul")
```

関連情報

- 詳しくは [Indexing Collections](#) をご覧ください。
- インデックスを定義する JSON ドキュメントの詳細は、[Defining an Index](#) を参照してください。
- 完全な構文の定義は、[Collection Index Management Functions](#) を参照してください。

20.3.4 リレーショナルテーブル

X DevAPI を使用してリレーショナルテーブルを操作することもできます。MySQL では、各リレーショナルテーブルは特定のストレージエンジンに関連付けられます。このセクションの例では、`world_x` スキーマの `InnoDB` テーブルを使用します。

スキーマの確認

`db` グローバル変数に割り当てられているスキーマを表示するには、`db` を発行します。

```
mysql-js> db  
<Schema:world_x>
```

戻り値が `Schema:world_x` でない場合は、`db` 変数を次のように設定します:

```
mysql-js> \use world_x  
Schema `world_x` accessible through db.
```

すべてのテーブルの表示

`world_x` スキーマ内のすべてのリレーショナルテーブルを表示するには、`db` オブジェクトで `getTables()` メソッドを使用します。

```
mysql-js> db.getTables()  
{  
  "city": <Table:city>,  
  "country": <Table:country>,  
  "countrylanguage": <Table:countrylanguage>  
}
```

基本的なテーブル操作

テーブルによってスコープ指定される基本的な操作は次のとおりです:

操作フォーム	説明
<code>db.name.insert()</code>	<code>insert()</code> メソッドは、名前付きのテーブルに 1 つ以上のレコードを挿入します。
<code>db.name.select()</code>	<code>select()</code> メソッドは、指定されたテーブルの一部またはすべてのレコードを返します。
<code>db.name.update()</code>	<code>update()</code> メソッドは、指定されたテーブルのレコードを更新します。
<code>db.name.delete()</code>	<code>delete()</code> メソッドは、名前付きのテーブルから 1 つ以上のレコードを削除します。

関連情報

- 詳しくは [Working with Relational Tables](#) をご覧ください。
- [CRUD EBNF Definitions](#) では、操作の完全なリストが提供されます。
- `world_x` スキーマサンプルの設定手順は、[セクション 20.3.2 「world_x データベースのダウンロードおよびインポート」](#) を参照してください。

20.3.4.1 テーブルへのレコードの挿入

`insert()` メソッドを `values()` メソッドとともに使用して、既存のリレーショナルテーブルにレコードを挿入できます。`insert()` メソッドは、個々のカラムまたはテーブル内のすべてのカラムを受け入れます。1 つ以上の `values()` メソッドを使用して、挿入する値を指定します。

完全なレコードの挿入

完全なレコードを挿入するには、テーブルのすべてのカラムを `insert()` メソッドに渡します。次に、`values()` メソッドに、テーブルの各カラムの値を渡します。たとえば、`world_x` スキーマの市区町村テーブルに新しいレコードを追加するには、次のレコードを挿入して Enter を 2 回押します。

```
mysql-js> db.city.insert("ID", "Name", "CountryCode", "District", "Info").values(
None, "Olympia", "USA", "Washington", {"Population": 5000})
```

`city` テーブルには 5 つのカラムがあります: ID、名前、CountryCode、地区および情報。各値は、それが表すカラムのデータ型と一致する必要があります。

部分レコードの挿入

次の例では、`city` テーブルの ID、Name および CountryCode カラムに値を挿入します。

```
mysql-js> db.city.insert("ID", "Name", "CountryCode").values(
None, "Little Falls", "USA").values(None, "Happy Valley", "USA")
```

`insert()` メソッドを使用してカラムを指定する場合、値の数はカラムの数と一致する必要があります。前の例では、指定した 3 つのカラムに一致するように 3 つの値を指定する必要があります。

関連情報

- 完全な構文の定義は、[TableInsertFunction](#) を参照してください。

20.3.4.2 テーブルの選択

`select()` メソッドを使用して、データベース内のテーブルに対してクエリーを実行し、レコードを戻すことができます。X DevAPI には、返されたレコードをフィルタおよびソートするために `select()` メソッドとともに使用する追加のメソッドが用意されています。

MySQL には、検索条件を指定する次の演算子が用意されています: `OR (||)`、`AND (&&)`、`XOR (|)`、`IS`、`NOT`、`BETWEEN`、`IN`、`LIKE`、`!=`、`<>`、`>`、`>=`、`<`、`<=`、`&`、`|`、`<<`、`>>`、`+`、`-`、`*`、`/`、`~` および `%`。

すべてのレコードの選択

既存のテーブルからすべてのレコードを返すクエリを発行するには、検索条件を指定せずに `select()` メソッドを使用します。次の例では、`world_x` データベースの `city` テーブルからすべてのレコードを選択します。

注記

空の `select()` メソッドの使用を対話型のステートメントに制限します。アプリケーションコードでは、常に明示的なカラム名の選択を使用します。

```
mysql-js> db.city.select()
+-----+-----+-----+-----+
| ID | Name   | CountryCode | District | Info           |
+-----+-----+-----+-----+
| 1  | Kabul  | AFG         | Kabul   | {"Population": 1780000} |
| 2  | Qandahar | AFG        | Qandahar | {"Population": 237500} |
| 3  | Herat   | AFG         | Herat   | {"Population": 186800} |
...
...
| 4079 | Rafah   | PSE         | Rafah   | {"Population": 92020} |
+-----+-----+-----+-----+
4082 rows in set (0.01 sec)
```

空のセット（一致するレコードなし）は、次の情報を返します：

```
Empty set (0.00 sec)
```

フィルタ検索

一連のテーブルのカラムを返すクエリを発行するには、`select()` メソッドを使用して、角カッコで囲むカラムを指定します。このクエリは、`city` テーブルから `Name` カラムと `CountryCode` カラムを返します。

```
mysql-js> db.city.select(["Name", "CountryCode"])
+-----+-----+
| Name   | CountryCode |
+-----+-----+
| Kabul  | AFG         |
| Qandahar | AFG        |
| Herat   | AFG         |
| Mazar-e-Sharif | AFG       |
| Amsterdam | NLD        |
...
...
| Rafah   | PSE         |
| Olympia | USA         |
| Little Falls | USA       |
| Happy Valley | USA      |
+-----+-----+
4082 rows in set (0.00 sec)
```

特定の検索条件に一致する行を返すクエリを発行するには、`where()` メソッドを使用してそれらの条件を含めます。たとえば、次の例では、文字 Z で始まる都市の名前と国コードを返します。

```
mysql-js> db.city.select(["Name", "CountryCode"]).where("Name like 'Z%")
+-----+-----+
| Name   | CountryCode |
+-----+-----+
| Zaanstad | NLD         |
| Zoetermeer | NLD        |
| Zwolle   | NLD         |
| Zenica   | BIH         |
| Zagazig  | EGY         |
| Zaragoza | ESP         |
| Zamboanga | PHL         |
| Zahedan  | IRN         |
| Zanjian  | IRN         |
| Zabol    | IRN         |
| Zama     | JPN         |
| Zhezqazghan | KAZ        |
| Zhengzhou | CHN         |
...
...
| Zeleznogorsk | RUS        |
```

```
+-----+-----+
59 rows in set (0.00 sec)
```

`bind()` メソッドを使用して、値を検索条件から区切ることができます。たとえば、条件として「Name =」Z%「」を使用するかわりに、コロンで構成される名前付きプレースホルダの後に、`name` などの文字で始まる名前を付けます。次に、次のようにプレースホルダと値を `bind()` メソッドに含めます:

```
mysql-js> db.city.select(["Name", "CountryCode"])
    .where("Name like :name").bind("name", "Z%")
```

ヒント

プログラム内でバインディングを使用すると、式にプレースホルダを指定できます。プレースホルダには、実行前に値が入力され、必要に応じて自動エスケープを利用できます。

入力をサニタイズするには、常にバインディングを使用します。文字列連結を使用してクエリーに値を導入しないでください。無効な入力が発生され、場合によってはセキュリティの問題が発生することがあります。

プロジェクト結果

AND 演算子を使用してクエリーを発行するには、`where()` メソッドの検索条件の間に演算子を追加します。

```
mysql-js> db.city.select(["Name", "CountryCode"]).where(
  "Name like 'Z%' and CountryCode = 'CHN'")
```

```
+-----+-----+
| Name      | CountryCode |
+-----+-----+
| Zhengzhou | CHN         |
| Zibo      | CHN         |
| Zhangjiakou | CHN        |
| Zhuzhou   | CHN         |
| Zhangjiang | CHN         |
| Zigong    | CHN         |
| Zaozhuang | CHN         |
| ...       | ...         |
| Zhangjiagang | CHN        |
+-----+-----+
22 rows in set (0.01 sec)
```

複数の条件演算子を指定するには、検索条件をカッコで囲んで演算子の優先順位を変更します。次の例は、**AND** および **OR** 演算子の配置を示しています。

```
mysql-js> db.city.select(["Name", "CountryCode"])
    .where("Name like 'Z%' and (CountryCode = 'CHN' or CountryCode = 'RUS')")
```

```
+-----+-----+
| Name      | CountryCode |
+-----+-----+
| Zhengzhou | CHN         |
| Zibo      | CHN         |
| Zhangjiakou | CHN        |
| Zhuzhou   | CHN         |
| ...       | ...         |
| Zeleznogorsk | RUS        |
+-----+-----+
29 rows in set (0.01 sec)
```

制限、順序およびオフセットの結果

`limit()`、`orderBy()` および `offset()` メソッドを適用して、`select()` メソッドによって返されるレコードの数と順序を管理できます。

結果セットに含まれるレコード数を指定するには、`limit()` メソッドに `select()` メソッドの値を追加します。たとえば、次のクエリーでは、国テーブルの最初の 5 つのレコードが返されます。

```
mysql-js> db.country.select(["Code", "Name"]).limit(5)
```

```
+-----+-----+
| Code | Name |
+-----+-----+
```

```

| ABW | Aruba |
| AFG | Afghanistan |
| AGO | Angola |
| AIA | Anguilla |
| ALB | Albania |
+----+-----+
5 rows in set (0.00 sec)

```

結果の順序を指定するには、`orderBy()` メソッドを `select()` メソッドに追加します。必要に応じて、降順 (`desc`) または昇順 (`asc`) 属性でソートするカラムのリストを `orderBy()` メソッドに渡します。昇順がデフォルトのオーダータイプです。

たとえば、次のクエリーでは、すべてのレコードが名前カラムでソートされ、最初の 3 つのレコードが降順で返されます。

```

mysql-js> db.country.select(["Code", "Name"]).orderBy(["Name desc"]).limit(3)
+----+-----+
| Code | Name |
+----+-----+
| ZWE | Zimbabwe |
| ZMB | Zambia |
| YUG | Yugoslavia |
+----+-----+
3 rows in set (0.00 sec)

```

デフォルトでは、`limit()` メソッドはテーブルの最初のレコードから開始されます。`offset()` メソッドを使用して、開始レコードを変更できます。たとえば、最初のレコードを無視し、条件に一致する次の 3 つのレコードを返すには、`offset()` メソッドに値 1 を渡します。

```

mysql-js> db.country.select(["Code", "Name"]).orderBy(["Name desc"]).limit(3).offset(1)
+----+-----+
| Code | Name |
+----+-----+
| ZMB | Zambia |
| YUG | Yugoslavia |
| YEM | Yemen |
+----+-----+
3 rows in set (0.00 sec)

```

関連情報

- [MySQL Reference Manual](#) には、関数および演算子に関する詳細なドキュメントが用意されています。
- 完全な構文の定義は、[TableSelectFunction](#) を参照してください。

20.3.4.3 テーブルの更新

`update()` メソッドを使用して、テーブル内のレコードを変更できます。`update()` メソッドは、更新するレコードのみを含めるようにクエリーをフィルタリングし、指定した操作をそれらのレコードに適用することで機能します。

市区町村テーブルの市区町村名を置換するには、`set()` メソッドに新しい市区町村名を渡します。次に、検索および置換する市区町村名を `where()` メソッドに渡します。次の例では、Peking 市を Beijing に置き換えます。

```
mysql-js> db.city.update().set("Name", "Beijing").where("Name = 'Peking'")
```

`select()` メソッドを使用して変更を確認します。

```

mysql-js> db.city.select(["ID", "Name", "CountryCode", "District", "Info"]).where("Name = 'Beijing'")
+----+-----+-----+-----+-----+
| ID | Name | CountryCode | District | Info |
+----+-----+-----+-----+-----+
| 1891 | Beijing | CHN | Peking | {"Population": 7472000} |
+----+-----+-----+-----+-----+
1 row in set (0.00 sec)

```

関連情報

- 完全な構文の定義は、[TableUpdateFunction](#) を参照してください。

20.3.4.4 テーブルの削除

`delete()` メソッドを使用して、データベース内のテーブルから一部またはすべてのレコードを削除できます。X DevAPI には、削除するレコードをフィルタおよび順序付けするために `delete()` メソッドとともに使用する追加のメソッドが用意されています。

条件を使用したレコードの削除

次の例では、検索条件を `delete()` メソッドに渡します。条件に一致するすべてのレコードが市区町村テーブルから削除されます。この例では、1 つのレコードが条件に一致します。

```
mysql-js> db.city.delete().where("Name = 'Olympia'")
```

最初のレコードの削除

市区町村テーブルの最初のレコードを削除するには、値が 1 の `limit()` メソッドを使用します。

```
mysql-js> db.city.delete().limit(1)
```

テーブル内のすべてのレコードの削除

テーブル内のすべてのレコードを削除できます。これを行うには、検索条件を指定せずに `delete()` メソッドを使用します。

注意

検索条件を指定せずにレコードを削除する場合は注意が必要です。削除すると、テーブルからすべてのレコードが削除されます。

テーブルの削除

`dropCollection()` メソッドは、データベースからリレーショナルテーブルを削除するために MySQL Shell でも使用されます。たとえば、`world_x` データベースから `citytest` テーブルを削除するには、次のように発行します:

```
mysql-js> session.dropCollection("world_x", "citytest")
```

関連情報

- 完全な構文の定義は、[TableDeleteFunction](#) を参照してください。
- `world_x` データベースを再作成する手順は、[セクション20.3.2「world_x データベースのダウンロードおよびインポート」](#) を参照してください。

20.3.5 テーブル内のドキュメント

MySQL では、テーブルに従来のリレーショナルデータまたは JSON 値 (あるいはその両方) が含まれる場合があります。ネイティブ JSON データ型を持つカラムにドキュメントを格納することで、従来のデータを JSON ドキュメントと組み合わせることができます。

このセクションの例では、`world_x` スキーマの `city` テーブルを使用します。

市区町村テーブル摘要

`city` テーブルには、5 つのカラム (またはフィールド) があります。

Field	Type	Null	Key	Default	Extra
ID	int(11)	NO	PRI	null	auto_increment
Name	char(35)	NO			
CountryCode	char(3)	NO			
District	char(20)	NO			

```
Info | json | YES | | null | |
+-----+-----+-----+-----+-----+-----+
```

レコードの挿入

テーブルのカラムにドキュメントを挿入するには、正しい順序で整形形式の JSON ドキュメントを `values()` メソッドに渡します。次の例では、情報カラムに挿入される最終値としてドキュメントが渡されます。

```
mysql-js> db.city.insert().values(
None, "San Francisco", "USA", "California", '{"Population":830000}')
```

レコードの選択

式のドキュメント値を評価する検索条件を指定してクエリーを発行できます。

```
mysql-js> db.city.select(["ID", "Name", "CountryCode", "District", "Info"]).where(
"CountryCode = :country and Info->$Population > 1000000").bind(
'country', 'USA')
```

```
+-----+-----+-----+-----+-----+-----+
| ID | Name      | CountryCode | District | Info                                     |
+-----+-----+-----+-----+-----+-----+
| 3793 | New York  | USA         | New York | {"Population": 8008278} |
| 3794 | Los Angeles | USA        | California | {"Population": 3694820} |
| 3795 | Chicago   | USA         | Illinois  | {"Population": 2896016} |
| 3796 | Houston   | USA         | Texas     | {"Population": 1953631} |
| 3797 | Philadelphia | USA        | Pennsylvania | {"Population": 1517550} |
| 3798 | Phoenix   | USA         | Arizona   | {"Population": 1321045} |
| 3799 | San Diego | USA         | California | {"Population": 1223400} |
| 3800 | Dallas    | USA         | Texas     | {"Population": 1188580} |
| 3801 | San Antonio | USA        | Texas     | {"Population": 1144646} |
+-----+-----+-----+-----+-----+-----+
9 rows in set (0.01 sec)
```

関連情報

- 詳しくは [Working with Relational Tables and Documents](#) をご覧ください。
- データ型の詳細は、[セクション11.5「JSON データ型」](#) を参照してください。

20.4 Python クイックスタートガイド: ドキュメントストア用の MySQL Shell

このクイックスタートガイドでは、MySQL Shell と対話形式でドキュメントストアアプリケーションのプロトタイプ作成を開始する手順について説明します。このガイドの内容は次のとおりです:

- MySQL 機能、MySQL Shell および `world_x` サンプルスキーマの概要。
- コレクションおよびドキュメントを管理する操作。
- リレーショナルテーブルを管理する操作。
- テーブル内のドキュメントに適用される操作。

このクイックスタートガイドに従うには、MySQL サーバーに X プラグイン がインストールされている必要があります。8.0 のデフォルトは MySQL Shell で、MySQL Shell は X DevAPI を含み、JavaScript および Python の両方で実装されており、MySQL サーバーインスタンスに X プロトコル を使用して接続し、サーバーをドキュメントストアとして使用できます。

関連情報

- [MySQL Shell 8.0](#) は、MySQL Shell に関する詳細情報を提供します。
- このクイックスタートガイドで使用されるツールの詳細は、[MySQL Shell のインストール](#) および [セクション 20.5「X プラグイン」](#) を参照してください。

- MySQL Shell でサポートされる言語の詳細は、[サポートされる言語](#) を参照してください。
- [X DevAPI User Guide](#) には、X DevAPI を使用して、MySQL をドキュメントストアとして使用するアプリケーションを開発するその他の例があります。
- [JavaScript](#) クイックスタートガイドも使用できます。

20.4.1 MySQL Shell

このクイックスタートガイドでは、MySQL Shell について一定レベルの知識があることを前提としています。次のセクションは概要です。詳細は、MySQL Shell のドキュメントを参照してください。MySQL Shell は、MySQL Server への統一されたスクリプトインタフェースです。JavaScript および Python でのスクリプトがサポートされています。JavaScript はデフォルトの処理モードです。

MySQL Shell の起動

MySQL サーバーをインストールして起動したら、MySQL Shell をサーバーインスタンスに接続します。接続する予定の MySQL サーバーインスタンスのアドレスを知っている必要があります。インスタンスをドキュメントストアとして使用できるようにするには、サーバーインスタンスに X プラグイン がインストールされている必要があり、X プロトコル を使用してサーバーに接続する必要があります。たとえば、33060 のデフォルトの X プロトコル ポートでインスタンス `ds1.example.com` に接続するには、ネットワーク文字列 `user@ds1.example.com:33060` を使します。

ヒント

`mysqlx_port` のかわりに 3306 のデフォルトの `port` を使用するなどして、クラシック MySQL プロトコル を使用してインスタンスに接続する場合、このチュートリアルに示すドキュメントストア機能は使用できません。たとえば、`db` グローバルオブジェクトは移入されません。ドキュメントストアを使用するには、常に X プロトコル を使用して接続します。

MySQL Shell がまだ実行されていない場合は、ターミナルウィンドウを開き、次のコマンドを発行します：

```
mysqlsh user@ds1.example.com:33060/world_x
```

または、MySQL Shell がすでに実行されている場合は、次のコマンドを発行して `\connect` コマンドを使用します：

```
\connect user@ds1.example.com:33060/world_x
```

MySQL Shell を接続する MySQL サーバーインスタンスのアドレスを指定する必要があります。たとえば、前の例では次のようになります：

- `user` は、MySQL アカウントのユーザー名を表します。
- `ds1.example.com` は、MySQL を実行しているサーバーインスタンスのホスト名です。これを、ドキュメントストアとして使用している MySQL サーバーインスタンスのホスト名に置き換えます。
- このセッションのデフォルトスキーマは `world_x` です。 `world_x` スキーマの設定手順は、[セクション 20.4.2 「world_x データベースのダウンロードおよびインポート」](#) を参照してください。

詳細は、[セクション 4.2.5 「URI 類似文字列またはキーと値のペアを使用したサーバーへの接続」](#) を参照してください。

MySQL Shell が開くと、`mysql-js>` プロンプトに、このセッションのアクティブな言語が JavaScript であることが示されます。MySQL Shell を Python モードに切り替えるには、`\py` コマンドを使用します。

```
mysql-js> \py
Switching to Python mode...
mysql-py>
```

MySQL Shell は、次のような入力行編集をサポートしています：

- left-arrow および right-arrow キーは、現在の入力行内で水平に移動します。
- up-arrow および down-arrow のキーは、以前に入力した一連の行を上下に移動します。

- 「バックスペース」でカーソルの前の文字を削除でき、新しい文字を入力するとカーソルの位置に挿入されます。
- Enter は、現在の入力行をサーバーに送信します。

MySQL Shell のヘルプの表示

コマンドラインオプションのリストを表示するには、コマンドインタプリタのプロンプトで `mysqlsh --help` と入力します。

```
mysqlsh --help
```

使用可能なコマンドとその説明のリストを表示するには、MySQL Shell プロンプトで `\help` と入力します。

```
mysql-py> \help
```

個々の MySQL Shell コマンドの詳細なヘルプを表示するには、`\help` に続けてコマンド名を入力します。たとえば、`\connect` コマンドのヘルプを表示するには、次のコマンドを発行します:

```
mysql-py> \help \connect
```

MySQL Shell の終了

MySQL Shell を終了するには、次のコマンドを発行します:

```
mysql-py> \quit
```

関連情報

- MySQL Shell での対話型コード実行の動作については、[対話型コードの実行](#) を参照してください。
- セッションおよび接続の代替方法の詳細は、[MySQL Shell スタートガイド](#) を参照してください。

20.4.2 world_x データベースのダウンロードおよびインポート

このクイックスタートガイドの一部として、`world_x` スキーマと呼ばれるサンプルスキーマが用意されています。これらの例の多くは、このスキーマを使用するドキュメントストア機能を示しています。`world_x` スキーマをロードできるように MySQL サーバーを起動し、次のステップに従います:

1. `world_x-db.zip` のダウンロード。
2. `/tmp` などの一時的な場所にインストールアーカイブを抽出します。アーカイブを解凍すると、`world_x.sql` という名前の単一ファイルが作成されます。
3. `world_x.sql` ファイルをサーバーにインポートします。次のいずれかを実行できます:

- MySQL Shell を SQL モードで起動し、次のコマンドを発行してファイルをインポートします:

```
mysqlsh -u root --sql --file /tmp/world_x-db/world_x.sql  
Enter password: ****
```

- 実行中に MySQL Shell を SQL モードに設定し、次のコマンドを発行してスキーマファイルをソーシングします:

```
\sql  
Switching to SQL mode... Commands end with ;  
\source /tmp/world_x-db/world_x.sql
```

`/tmp` をシステム上の `world_x.sql` ファイルへのパスに置き換えます。プロンプトが表示されたら、パスワードを入力します。root 以外のアカウントは、そのアカウントが新しいスキーマを作成する権限を持っているかぎり使用できません。

world_x スキーマ

`world_x` サンプルスキーマには、次の JSON コレクションおよびリレーショナルテーブルが含まれています:

- コレクション
 - `countryinfo`: 世界中の国に関する情報。
- テーブル
 - `country`: 世界の国に関する最小限の情報。
 - `city`: これらの国の一部の都市に関する情報。
 - `countrylanguage`: 各国で話されている言語。

関連情報

- [MySQL Shell セッション](#) では、セッションタイプについて説明します。

20.4.3 ドキュメントとコレクション

MySQL をドキュメントストアとして使用している場合、コレクションは、作成、リストおよびドロップできるスキーマ内のコンテナです。コレクションには、追加、検索、更新および削除できる JSON ドキュメントが含まれます。

このセクションの例では、`world_x` スキーマの `countryinfo` コレクションを使用します。`world_x` スキーマの設定手順は、[セクション20.4.2「world_x データベースのダウンロードおよびインポート」](#) を参照してください。

Documents

MySQL では、ドキュメントは JSON オブジェクトとして表されます。内部的には、高速ルックアップおよび更新を可能にする効率的なバイナリ形式で格納されます。

- Python の単純なドキュメント形式:

```
{"field1": "value", "field2": 10, "field 3": null}
```

ドキュメントの配列は、カンマで区切られた一連のドキュメントで構成され、`[および]`文字で囲まれます。

- Python のドキュメントの単純な配列:

```
[{"Name": "Aruba", "Code": "ABW"}, {"Name": "Angola", "Code": "AGO"}]
```

MySQL は、JSON ドキュメントで次の Python 値タイプをサポートしています:

- 数値 (整数および浮動小数点)
- 文字列
- boolean (False および True)
- なし
- JSON 値の配列
- より多くの JSON 値のネストされた (埋込み) オブジェクト

コレクション

コレクションは、目的を共有し、場合によっては 1 つ以上のインデックスを共有するドキュメントのコンテナです。各コレクションは一意の名前を持ち、単一のスキーマ内に存在します。

スキーマという用語はデータベースと同等です。つまり、リレーショナルスキーマとは対照的に、データベースオブジェクトのグループであり、データの構造と制約を施行するために使用されます。スキーマは、コレクション内のドキュメントへの準拠を強制しません。

このクイックスタートガイドの内容は次のとおりです:

- 基本オブジェクトには次のものがあります:

オブジェクトフォーム	説明
<code>db</code>	<code>db</code> は、現在アクティブなスキーマに割り当てられたグローバル変数です。コレクションの取得など、スキーマに対して操作を実行する場合は、 <code>db</code> 変数に使用可能なメソッドを使用します。
<code>db.get_collections()</code>	<code>db.get_collections()</code> は、スキーマ内のコレクションのリストを返します。このリストを使用して、コレクションオブジェクトへの参照の取得、コレクションオブジェクトの反復などを行います。

- コレクションによってスコープ指定される基本的な操作は次のとおりです:

操作フォーム	説明
<code>db.name.add()</code>	<code>add()</code> メソッドは、ドキュメントまたはドキュメントのリストを名前付きコレクションに挿入します。
<code>db.name.find()</code>	<code>find()</code> メソッドは、名前付きコレクション内の一部またはすべてのドキュメントを返します。
<code>db.name.modify()</code>	<code>modify()</code> メソッドは、名前付きコレクション内のドキュメントを更新します。
<code>db.name.remove()</code>	<code>remove()</code> メソッドは、名前付きコレクションからドキュメントまたはドキュメントのリストを削除します。

関連情報

- 一般的な概要は、[Working with Collections](#) を参照してください。
- [CRUD EBNF Definitions](#) では、操作の完全なリストが提供されます。

20.4.3.1 コレクションの作成、リストおよびドロップ

MySQL Shell では、新しいコレクションを作成し、スキーマ内の既存のコレクションのリストを取得し、スキーマから既存のコレクションを削除できます。コレクション名は大/小文字が区別され、各コレクション名は一意である必要があります。

スキーマの確認

スキーマ変数に割り当てられている値を表示するには、次のコマンドを発行します:

```
mysql-py> db
```

スキーマ値が `Schema:world_x` でない場合は、次のコマンドを発行して `db` 変数を設定します:

```
mysql-py> \use world_x
```

コレクションの作成

既存のスキーマに新しいコレクションを作成するには、`db` オブジェクトの `createCollection()` メソッドを使用します。次の例では、`world_x` スキーマに `flags` というコレクションを作成します。

```
mysql-py> db.create_collection("flags")
```

このメソッドはコレクションオブジェクトを返します。

```
<Collection:flags>
```

コレクションのリスト

`world_x` スキーマ内のすべてのコレクションを表示するには、`db` オブジェクトの `get_collections()` メソッドを使用します。現在接続しているサーバーによって返されるコレクションは、大カッコ内に表示されます。

```
mysql-py> db.get_collections()
[
  <Collection:countryinfo>,
  <Collection:flags>
]
```

コレクションのドロップ

スキーマから既存のコレクションを削除するには、`db` オブジェクトの `drop_collection()` メソッドを使用します。たとえば、現行のスキーマから `flags` コレクションを削除するには、次のコマンドを発行します:

```
mysql-py> db.drop_collection("flags")
```

`drop_collection()` メソッドは、スキーマからリレーショナルテーブルを削除するために MySQL Shell でも使用されません。

関連情報

- その他の例については、[Collection Objects](#) を参照してください。

20.4.3.2 コレクションの操作

スキーマ内のコレクションを操作するには、`db` グローバルオブジェクトを使用して現在のスキーマにアクセスします。この例では、以前にインポートした `world_x` スキーマと `countryinfo` コレクションを使用しています。したがって、発行する操作の形式は `db.collection_name.operation` で、`collection_name` は操作の実行対象となるコレクションの名前です。次の例では、操作は `countryinfo` コレクションに対して実行されます。

ドキュメントの追加

`add()` メソッドを使用して、あるドキュメントまたはドキュメントのリストを既存のコレクションに挿入します。次のドキュメントを `countryinfo` コレクションに挿入します。これは複数行コンテンツであるため、Enter を 2 回押してドキュメントを挿入します。

```
mysql-py> db.countryinfo.add(
{
  "GNP": .6,
  "IndepYear": 1967,
  "Name": "Sealand",
  "Code": "SEA",
  "demographics": {
    "LifeExpectancy": 79,
    "Population": 27
  },
  "geography": {
    "Continent": "Europe",
    "Region": "British Islands",
    "SurfaceArea": 193
  },
  "government": {
    "GovernmentForm": "Monarchy",
    "HeadOfState": "Michael Bates"
  }
}
)
```

このメソッドは、操作のステータスを返します。ドキュメントを検索して操作を確認できます。例:

```
mysql-py> db.countryinfo.find("Name = 'Sealand'")
{
  "GNP": 0.6,
  "_id": "00005e2ff4af00000000000000f4",
  "Name": "Sealand",
  "Code": "SEA",
  "IndepYear": 1967,
```

```
"geography": {
  "Region": "British Islands",
  "Continent": "Europe",
  "SurfaceArea": 193
},
"government": {
  "HeadOfState": "Michael Bates",
  "GovernmentForm": "Monarchy"
},
"demographics": {
  "Population": 27,
  "LifeExpectancy": 79
}
}
```

ドキュメントの追加時に指定されたフィールドに加えて、`_id` というフィールドもあります。各ドキュメントには、`_id` と呼ばれる識別子フィールドが必要です。`_id` フィールドの値は、同じコレクション内のすべてのドキュメント間で一意である必要があります。MySQL 8.0.11 以上では、ドキュメント ID はクライアントではなくサーバーによって生成されるため、MySQL Shell は `_id` 値を自動的に設定しません。8.0.11 以上の MySQL サーバーでは、ドキュメントに `_id` フィールドが含まれていない場合、`_id` 値が設定されます。以前の 8.0 リリースまたは 5.7 の MySQL サーバーは、この状況では `_id` 値を設定しないため、明示的に指定する必要があります。そうでない場合、MySQL Shell はエラー 5115 「文書に必須フィールドがありません」を返します。詳細は、[Understanding Document IDs](#) を参照してください。

関連情報

- 完全な構文の定義は、[CollectionAddFunction](#) を参照してください。
- [Understanding Document IDs](#) を参照してください。

20.4.3.3 ドキュメントの検索

`find()` メソッドを使用して、スキーマ内のコレクションに対してクエリーを実行し、ドキュメントを戻すことができます。MySQL Shell には、返されたドキュメントをフィルタおよびソートするために `find()` メソッドとともに使用する追加のメソッドが用意されています。

MySQL には、検索条件を指定する次の演算子が用意されています: `OR (||)`、`AND (&&)`、`XOR`、`IS`、`NOT`、`BETWEEN`、`IN`、`LIKE`、`!=`、`<>`、`>`、`>=`、`<`、`<=`、`&`、`|`、`<<`、`>>`、`+`、`-`、`*`、`/`、`~` および `%`。

コレクション内のすべてのドキュメントの検索

コレクション内のすべてのドキュメントを返すには、検索条件を指定せずに `find()` メソッドを使用します。たとえば、次の操作では、`countryinfo` コレクション内のすべてのドキュメントが返されます。

```
mysql-py> db.countryinfo.find()
[
  {
    "GNP": 828,
    "Code": "ABW",
    "Name": "Aruba",
    "IndepYear": null,
    "geography": {
      "Continent": "North America",
      "Region": "Caribbean",
      "SurfaceArea": 193
    },
    "government": {
      "GovernmentForm": "Nonmetropolitan Territory of The Netherlands",
      "HeadOfState": "Beatrix"
    }
  },
  "demographics": {
    "LifeExpectancy": 78.4000015258789,
    "Population": 103000
  },
  ...
}
]
240 documents in set (0.00 sec)
```


このメソッドは、コレクション内のすべてのドキュメントに加えて、操作情報を含む結果を生成します。

空のセット (一致するドキュメントがない) は、次の情報を返します:

```
Empty set (0.00 sec)
```

フィルタ検索

`find()` メソッドに検索条件を含めることができます。検索条件を形成する式の構文は、従来の MySQL 第12章「関数と演算子」の構文と同じです。すべての式は引用符で囲む必要があります。簡潔にするために、一部の例では出力が表示されません。

単純検索条件は、`Name` フィールドとドキュメント内にあることがわかっている値で構成できます。次の例では、単一のドキュメントを返します:

```
mysql-py> db.countryinfo.find("Name = 'Australia'")
[
  {
    "GNP": 351182,
    "Code": "AUS",
    "Name": "Australia",
    "IndepYear": 1901,
    "geography": {
      "Continent": "Oceania",
      "Region": "Australia and New Zealand",
      "SurfaceArea": 7741220
    },
    "government": {
      "GovernmentForm": "Constitutional Monarchy, Federation",
      "HeadOfState": "Elisabeth II"
    },
    "demographics": {
      "LifeExpectancy": 79.80000305175781,
      "Population": 18886000
    }
  }
]
```

次の例では、GNP が 00 億を超えるすべての国を検索します。 `countryinfo` コレクションは GNP を百万単位で測定します。

```
mysql-py> db.countryinfo.find("GNP > 500000")
...[output removed]
10 documents in set (0.00 sec)
```

次のクエリーの「人口」フィールドは、人口統計オブジェクト内に埋め込まれます。埋込みフィールドにアクセスするには、人口統計と人口間のピリオドを使用して関係を識別します。ドキュメント名とフィールド名では大文字と小文字が区別されます。

```
mysql-py> db.countryinfo.find("GNP > 500000 and demographics.Population < 100000000")
...[output removed]
6 documents in set (0.00 sec)
```

次の式の算術演算子は、大文字当たりの GNP が 0000 を超える国をクエリーするために使用されます。検索条件には、算術演算子およびほとんどの MySQL 関数を含めることができます。

注記

`countryinfo` コレクション内の 7 つのドキュメントの母集団値はゼロです。したがって、出力の最後に警告メッセージが表示されます。

```
mysql-py> db.countryinfo.find("GNP*1000000/demographics.Population > 30000")
...[output removed]
9 documents in set, 7 warnings (0.00 sec)
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
```

```
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
```

`bind()` メソッドを使用して、値を検索条件から区切ることができます。たとえば、ハードコードされた国名を条件として指定するかわりに、コロンで構成される名前付きプレースホルダの後に、`country` などの文字で始まる名前を付けます。次に、`bind(placeholder, value)` メソッドを次のように使用します:

```
mysql-py> db.countryinfo.find("Name = :country").bind("country", "Italy")
{
  "GNP": 1161755,
  "_id": "00005de917d8000000000000006a",
  "Code": "ITA",
  "Name": "Italy",
  "Airports": [],
  "IndepYear": 1861,
  "geography": {
    "Region": "Southern Europe",
    "Continent": "Europe",
    "SurfaceArea": 301316
  },
  "government": {
    "HeadOfState": "Carlo Azeglio Ciampi",
    "GovernmentForm": "Republic"
  },
  "demographics": {
    "Population": 57680000,
    "LifeExpectancy": 79
  }
}
1 document in set (0.01 sec)
```

ヒント

プログラム内でバインディングを使用すると、式にプレースホルダを指定できます。プレースホルダには、実行前に値が入力され、必要に応じて自動エスケープを利用できます。

入力をサニタイズするには、常にバインディングを使用します。文字列連結を使用してクエリーに値を導入しないでください。無効な入力が発生され、場合によってはセキュリティの問題が発生することがあります。

プレースホルダおよび `bind()` メソッドを使用して保存済検索を作成し、別の値でコールできます。たとえば、国の保存済検索を作成するには、次のようにします:

```
mysql-py> myFind = db.countryinfo.find("Name = :country")
mysql-py> myFind.bind('country', 'France')
{
  "GNP": 1424285,
  "_id": "00005de917d80000000000000048",
  "Code": "FRA",
  "Name": "France",
  "IndepYear": 843,
  "geography": {
    "Region": "Western Europe",
    "Continent": "Europe",
    "SurfaceArea": 551500
  },
  "government": {
    "HeadOfState": "Jacques Chirac",
    "GovernmentForm": "Republic"
  },
  "demographics": {
    "Population": 59225700,
    "LifeExpectancy": 78.80000305175781
  }
}
1 document in set (0.0028 sec)

mysql-py> myFind.bind('country', 'Germany')
{
  "GNP": 2133367,
  "_id": "00005de917d80000000000000038",
```

```
"Code": "DEU",
>Name": "Germany",
>IndepYear": 1955,
>geography": {
>  "Region": "Western Europe",
>  "Continent": "Europe",
>  "SurfaceArea": 357022
>},
>government": {
>  "HeadOfState": "Johannes Rau",
>  "GovernmentForm": "Federal Republic"
>},
>demographics": {
>  "Population": 82164700,
>  "LifeExpectancy": 77.4000015258789
>}
}
```

```
1 document in set (0.0026 sec)
```

プロジェクト結果

すべてのフィールドを返すかわりに、ドキュメントの特定のフィールドを返すことができます。次の例では、検索条件に一致する `countryinfo` コレクション内のすべてのドキュメントの GNP および Name フィールドを返します。

`fields()` メソッドを使用して、返すフィールドのリストを渡します。

```
mysql-py> db.countryinfo.find("GNP > 5000000").fields(["GNP", "Name"])
```

```
[
  {
    "GNP": 8510700,
    "Name": "United States"
  }
]
```

```
1 document in set (0.00 sec)
```

また、返されるドキュメントを記述する式を使用して、返されるドキュメント (追加、名前変更、ネストおよび新しいフィールド値の計算) を変更できます。たとえば、次の式を使用してフィールドの名前を変更すると、2 つのドキュメントのみが返されます。

```
mysql-py> db.countryinfo.find().fields(
mysql.expr({"Name": upper(Name), "GNPPerCapita": GNP*1000000/demographics.Population})).limit(2)
```

```
{
  "Name": "ARUBA",
  "GNPPerCapita": 8038.834951456311
}
{
  "Name": "AFGHANISTAN",
  "GNPPerCapita": 263.0281690140845
}
```

結果の制限、ソートおよびスキップ

`limit()`、`sort()` および `skip()` メソッドを適用して、`find()` メソッドによって返されるドキュメントの数と順序を管理できます。

結果セットに含めるドキュメントの数を指定するには、`limit()` メソッドの値を `find()` メソッドに追加します。次のクエリーは、`countryinfo` コレクション内の最初の 5 つのドキュメントを返します。

```
mysql-py> db.countryinfo.find().limit(5)
```

```
... [output removed]
```

```
5 documents in set (0.00 sec)
```

結果の順序を指定するには、`sort()` メソッドを `find()` メソッドに追加します。必要に応じて、降順 (`desc`) または昇順 (`asc`) 属性でソートするフィールドのリストを `sort()` メソッドに渡します。昇順がデフォルトのオーダータイプです。

たとえば、次のクエリーでは、すべてのドキュメントが `IndepYear` フィールドでソートされ、最初の 8 つのドキュメントが降順で返されます。

```
mysql-py> db.countryinfo.find().sort(["IndepYear desc"]).limit(8)
... [output removed]
8 documents in set (0.00 sec)
```

デフォルトでは、`limit()` メソッドはコレクション内の最初のドキュメントから始まります。 `skip()` メソッドを使用して、開始ドキュメントを変更できます。たとえば、最初のドキュメントを無視し、条件に一致する次の 8 つのドキュメントを返すには、`skip()` メソッドに値 1 を渡します。

```
mysql-py> db.countryinfo.find().sort(["IndepYear desc"]).limit(8).skip(1)
... [output removed]
8 documents in set (0.00 sec)
```

関連情報

- [MySQL Reference Manual](#) には、関数および演算子に関する詳細なドキュメントが用意されています。
- 完全な構文の定義は、[CollectionFindFunction](#) を参照してください。

20.4.3.4 ドキュメントの変更

`modify()` メソッドを使用して、コレクション内の 1 つ以上のドキュメントを更新できます。X DevAPI には、`modify()` メソッドとともに使用して次の操作を行うための追加メソッドが用意されています:

- ドキュメント内のフィールドを設定および設定解除します。
- 配列を追加、挿入および削除します。
- 変更するドキュメントをバインド、制限およびソートします。

ドキュメントフィールドの設定および設定解除

`modify()` メソッドは、コレクションをフィルタ処理して変更するドキュメントのみを含め、指定した操作をそれらのドキュメントに適用することで機能します。

次の例では、`modify()` メソッドは検索条件を使用して変更するドキュメントを識別し、ネストされたデモグラフィックオブジェクト内の 2 つの値を `set()` メソッドで置き換えます。

```
mysql-py> db.countryinfo.modify("Code = 'SEA'").set(
"demographics", {"LifeExpectancy": 78, "Population": 28})
```

ドキュメントを変更した後、`find()` メソッドを使用して変更を確認します。

ドキュメントからコンテンツを削除するには、`modify()` および `unset()` メソッドを使用します。たとえば、次のクエリは、検索条件に一致するドキュメントから GNP を削除します。

```
mysql-py> db.countryinfo.modify("Name = 'Sealand'").unset("GNP")
```

`find()` メソッドを使用して変更を確認します。

```
mysql-py> db.countryinfo.find("Name = 'Sealand'")
{
  "_id": "00005e2ff4af000000000000000f4",
  "Name": "Sealand",
  "Code": "SEA",
  "IndepYear": 1967,
  "geography": {
    "Region": "British Islands",
    "Continent": "Europe",
    "SurfaceArea": 193
  },
  "government": {
    "HeadOfState": "Michael Bates",
    "GovernmentForm": "Monarchy"
  },
  "demographics": {
    "Population": 27,
    "LifeExpectancy": 79
  }
}
```

```
}  
}
```

配列の追加、挿入および削除

配列フィールドに要素を追加したり、配列内の要素を挿入または削除するには、`array_append()`、`array_insert()` または `array_delete()` メソッドを使用します。次の例では、国際空港のトラッキングを有効にするように `countryinfo` コレクションを変更します。

最初の例では、`modify()` および `set()` メソッドを使用して、すべてのドキュメントに新しい空港フィールドを作成します。

注意

検索条件を指定せずにドキュメントを変更する場合は注意してください。変更すると、コレクション内のすべてのドキュメントが変更されます。

```
mysql-py> db.countryinfo.modify("true").set("Airports", [])
```

空港フィールドが追加された状態で、次の例では、`array_append()` メソッドを使用して新しい空港をいずれかのドキュメントに追加します。次の例の `$.Airports` は、現在のドキュメントの空港フィールドを表しています。

```
mysql-py> db.countryinfo.modify("Name = 'France']").array_append("$.Airports", "ORY")
```

`find()` を使用して変更を確認します。

```
mysql-py> db.countryinfo.find("Name = 'France'")
```

```
{  
  "GNP": 1424285,  
  "_id": "00005de917d80000000000000048",  
  "Code": "FRA",  
  "Name": "France",  
  "Airports": [  
    "ORY"  
  ],  
  "IndepYear": 843,  
  "geography": {  
    "Region": "Western Europe",  
    "Continent": "Europe",  
    "SurfaceArea": 551500  
  },  
  "government": {  
    "HeadOfState": "Jacques Chirac",  
    "GovernmentForm": "Republic"  
  },  
  "demographics": {  
    "Population": 59225700,  
    "LifeExpectancy": 78.80000305175781  
  }  
}
```

配列内の別の位置に要素を挿入するには、`array_insert()` メソッドを使用して、パス式に挿入するインデックスを指定します。この場合、インデックスは 0、または配列の最初の要素です。

```
mysql-py> db.countryinfo.modify("Name = 'France']").array_insert("$.Airports[0]", "CDG")
```

配列から要素を削除するには、削除する要素のインデックスを `array_delete()` メソッドに渡す必要があります。

```
mysql-py> db.countryinfo.modify("Name = 'France']").array_delete("$.Airports[1]")
```

関連情報

- [MySQL Reference Manual](#) には、JSON 値の検索および変更に関係する手順が用意されています。
- 完全な構文の定義は、[CollectionModifyFunction](#) を参照してください。

20.4.3.5 ドキュメントの削除

`remove()` メソッドを使用して、スキーマ内のコレクションから一部またはすべてのドキュメントを削除できます。X DevAPI には、`remove()` メソッドとともに使用して、削除するドキュメントをフィルタおよびソートするための追加メソッドが用意されています。

条件を使用したドキュメントの削除

次の例では、検索条件を `remove()` メソッドに渡します。条件に一致するすべてのドキュメントが `countryinfo` コレクションから削除されます。この例では、1 つのドキュメントが条件に一致します。

```
mysql-py> db.countryinfo.remove("Code = 'SEA'")
```

最初のドキュメントの削除

`countryinfo` コレクションの最初のドキュメントを削除するには、値 1 を指定して `limit()` メソッドを使用します。

```
mysql-py> db.countryinfo.remove("true").limit(1)
```

オーダー内の最後の文書の削除

次の例では、国名で `countryinfo` コレクションの最後のドキュメントを削除します。

```
mysql-py> db.countryinfo.remove("true").sort(["Name desc"]).limit(1)
```

コレクション内のすべてのドキュメントの削除

コレクション内のすべてのドキュメントを削除できます。これを行うには、検索条件を指定せずに `remove("true")` メソッドを使用します。

注意

検索条件を指定せずにドキュメントを削除する場合は注意してください。このアクションは、コレクションからすべてのドキュメントを削除します。

または、`db.drop_collection('countryinfo')` 操作を使用して `countryinfo` コレクションを削除します。

関連情報

- 完全な構文の定義は、[CollectionRemoveFunction](#) を参照してください。
- `world_x` スキーマを再作成する手順は、[セクション20.4.2「world_x データベースのダウンロードおよびインポート」](#) を参照してください。

20.4.3.6 インデックスの作成および削除

インデックスは、特定のフィールド値を持つドキュメントをすばやく検索するために使用されます。インデックスがない場合、MySQL は最初のドキュメントで始まり、コレクション全体を読み取って関連するフィールドを検索する必要があります。コレクションが大きいくほど、このコストは高くなります。コレクションが大きく、特定のフィールドに対するクエリーが一般的な場合は、ドキュメント内の特定のフィールドに対するインデックスの作成を検討してください。

たとえば、移入フィールドのインデックスを使用すると、次のクエリーのパフォーマンスが向上します:

```
mysql-py> db.countryinfo.find("demographics.Population < 100")
...[output removed]
8 documents in set (0.00 sec)
```

`create_index()` メソッドは、使用するフィールドを指定する JSON ドキュメントで定義できるインデックスを作成します。このセクションでは、インデックス付けの概要について説明します。詳細は、[Indexing Collections](#) を参照してください。

一意でないインデックスの追加

一意でないインデックスを作成するには、インデックス名とインデックス情報を `create_index()` メソッドに渡します。重複するインデックス名は禁止されています。

次の例では、`demographics` オブジェクトの `Population` フィールドに対して定義され、`Integer` 数値としてインデックス付けされた `popul` という名前のインデックスを指定します。最後のパラメータは、フィールドに `NOT NULL` 制約が必要かどうかを示します。値が `false` の場合、フィールドには `NULL` 値を含めることができます。インデックス情報は、インデックスに含める 1 つ以上のフィールドの詳細を含む JSON ドキュメントです。各フィールド定義には、フィールドへの完全なドキュメントパスを含め、フィールドのタイプを指定する必要があります。

```
mysql-py> db.countryinfo.createIndex("popul", {fields:
  [{"field": '$.demographics.Population', type: 'INTEGER'}]})
```

ここでは、整数の数値を使用してインデックスが作成されます。GeoJSON データで使用するオプションなど、さらにオプションを使用できます。デフォルトタイプの「`index`」が適切であるため、ここでは省略されているインデックスのタイプを指定することもできます。

一意インデックスの追加

一意のインデックスを作成するには、インデックス名、インデックス定義およびインデックスタイプ「`unique`」を `create_index()` メソッドに渡します。この例は、国名 ("`Name`") に対して作成された一意のインデックスを示しています。これは、インデックス付けする `countryinfo` コレクションの別の共通フィールドです。インデックスフィールドの説明で、"`TEXT(40)`" はインデックス付けする文字数を表し、"`required: True`" はフィールドがドキュメントに存在する必要があることを指定します。

```
mysql-py> db.countryinfo.create_index("name",
  [{"fields": [{"field": "$.Name", "type": "TEXT(40)", "required": True}], "unique": True})
```

インデックスの削除

インデックスを削除するには、削除するインデックスの名前を `drop_index()` メソッドに渡します。たとえば、次のように「`popul`」インデックスを削除できます：

```
mysql-py> db.countryinfo.drop_index("popul")
```

関連情報

- 詳しくは [Indexing Collections](#) をご覧ください。
- インデックスを定義する JSON ドキュメントの詳細は、[Defining an Index](#) を参照してください。
- 完全な構文の定義は、[Collection Index Management Functions](#) を参照してください。

20.4.4 リレーショナルテーブル

X DevAPI を使用してリレーショナルテーブルを操作することもできます。MySQL では、各リレーショナルテーブルは特定のストレージエンジンに関連付けられます。このセクションの例では、`world_x` スキーマの `InnoDB` テーブルを使用します。

スキーマの確認

`db` グローバル変数に割り当てられているスキーマを表示するには、`db` を発行します。

```
mysql-py> db
<Schema:world_x>
```

戻り値が `Schema:world_x` でない場合は、`db` 変数を次のように設定します：

```
mysql-py> \use world_x
Schema `world_x` accessible through db.
```

すべてのテーブルの表示

`world_x` スキーマ内のすべてのリレーショナルテーブルを表示するには、`db` オブジェクトで `get_tables()` メソッドを使用します。

```
mysql-py> db.get_tables()
```

```
[
  <Table:city>,
  <Table:country>,
  <Table:countrylanguage>
]
```

基本的なテーブル操作

テーブルによってスコープ指定される基本的な操作は次のとおりです:

操作フォーム	説明
<code>db.name.insert()</code>	<code>insert()</code> メソッドは、名前付きのテーブルに 1 つ以上のレコードを挿入します。
<code>db.name.select()</code>	<code>select()</code> メソッドは、指定されたテーブルの一部またはすべてのレコードを返します。
<code>db.name.update()</code>	<code>update()</code> メソッドは、指定されたテーブルのレコードを更新します。
<code>db.name.delete()</code>	<code>delete()</code> メソッドは、名前付きのテーブルから 1 つ以上のレコードを削除します。

関連情報

- 詳しくは [Working with Relational Tables](#) をご覧ください。
- [CRUD EBNF Definitions](#) では、操作の完全なリストが提供されます。
- `world_x` スキーマサンプルの設定手順は、[セクション20.4.2「world_x データベースのダウンロードおよびインポート」](#) を参照してください。

20.4.4.1 テーブルへのレコードの挿入

`insert()` メソッドを `values()` メソッドとともに使用して、既存のリレーショナルテーブルにレコードを挿入できます。`insert()` メソッドは、個々のカラムまたはテーブル内のすべてのカラムを受け入れます。1 つ以上の `values()` メソッドを使用して、挿入する値を指定します。

完全なレコードの挿入

完全なレコードを挿入するには、テーブルのすべてのカラムを `insert()` メソッドに渡します。次に、`values()` メソッドに各カラムの値を渡します。たとえば、`world_x` データベースの市区町村テーブルに新しいレコードを追加するには、次のレコードを挿入して Enter を 2 回押します。

```
mysql-py> db.city.insert("ID", "Name", "CountryCode", "District", "Info").values(
None, "Olympia", "USA", "Washington", '{"Population": 5000}')
```

`city` テーブルには 5 つのカラムがあります: ID、名前、CountryCode、地区および情報。各値は、それが表すカラムのデータ型と一致する必要があります。

部分レコードの挿入

次の例では、`city` テーブルの ID、Name および CountryCode カラムに値を挿入します。

```
mysql-py> db.city.insert("ID", "Name", "CountryCode").values(
None, "Little Falls", "USA").values(None, "Happy Valley", "USA")
```

`insert()` メソッドを使用してカラムを指定する場合、値の数はカラムの数と一致する必要があります。前の例では、指定した 3 つのカラムに一致するように 3 つの値を指定する必要があります。

関連情報

- 完全な構文の定義は、[TableInsertFunction](#) を参照してください。

20.4.4.2 テーブルの選択

`select()` メソッドを使用して、データベース内のテーブルに対してクエリーを実行し、レコードを戻すことができます。X DevAPI には、返されたレコードをフィルタおよびソートするために `select()` メソッドとともに使用する追加のメソッドが用意されています。

MySQL には、検索条件を指定する次の演算子が用意されています: `OR (||)`、`AND (&&)`、`XOR`、`IS`、`NOT`、`BETWEEN`、`IN`、`LIKE`、`!=`、`<>`、`>`、`>=`、`<`、`<=`、`&`、`|`、`<<`、`>>`、`+`、`-`、`*`、`/`、`~` および `%`。

すべてのレコードの選択

既存のテーブルからすべてのレコードを返すクエリーを発行するには、検索条件を指定せずに `select()` メソッドを使用します。次の例では、`world_x` データベースの `city` テーブルからすべてのレコードを選択します。

注記

空の `select()` メソッドの使用を対話型のステートメントに制限します。アプリケーションコードでは、常に明示的なカラム名の選択を使用します。

```
mysql-py> db.city.select()
+-----+-----+-----+-----+
| ID | Name | CountryCode | District | Info |
+-----+-----+-----+-----+
| 1 | Kabul | AFG | Kabul | {"Population": 1780000} |
| 2 | Qandahar | AFG | Qandahar | {"Population": 237500} |
| 3 | Herat | AFG | Herat | {"Population": 186800} |
...
| 4079 | Rafah | PSE | Rafah | {"Population": 92020} |
+-----+-----+-----+-----+
4082 rows in set (0.01 sec)
```

空のセット (一致するレコードなし) は、次の情報を返します:

```
Empty set (0.00 sec)
```

フィルタ検索

一連のテーブルのカラムを戻すクエリーを発行するには、`select()` メソッドを使用して、角カッコで囲むカラムを指定します。このクエリーは、`city` テーブルから `Name` カラムと `CountryCode` カラムを返します。

```
mysql-py> db.city.select(["Name", "CountryCode"])
+-----+-----+
| Name | CountryCode |
+-----+-----+
| Kabul | AFG |
| Qandahar | AFG |
| Herat | AFG |
| Mazar-e-Sharif | AFG |
| Amsterdam | NLD |
...
| Rafah | PSE |
| Olympia | USA |
| Little Falls | USA |
| Happy Valley | USA |
+-----+-----+
4082 rows in set (0.00 sec)
```

特定の検索条件に一致する行を返すクエリーを発行するには、`where()` メソッドを使用してそれらの条件を含めます。たとえば、次の例では、文字 Z で始まる都市の名前と国コードを返します。

```
mysql-py> db.city.select(["Name", "CountryCode"]).where("Name like 'Z%")
+-----+-----+
| Name | CountryCode |
+-----+-----+
| Zaanstad | NLD |
| Zoetermeer | NLD |
| Zwolle | NLD |
+-----+-----+
```

```

| Zenica      | BIH      |
| Zagazig    | EGY      |
| Zaragoza   | ESP      |
| Zamboanga  | PHL      |
| Zahedan    | IRN      |
| Zanjian    | IRN      |
| Zabol      | IRN      |
| Zama       | JPN      |
| Zhezqazghan | KAZ     |
| Zhengzhou  | CHN      |
| ...
| Zeleznogorsk | RUS    |
+-----+-----+
59 rows in set (0.00 sec)

```

`bind()` メソッドを使用して、値を検索条件から区切ることができます。たとえば、条件として「Name =」Z%「」を使用するかわりに、コロンで構成される名前付きプレースホルダの後に、name などの文字で始まる名前を付けます。次に、次のようにプレースホルダと値を `bind()` メソッドに含めます:

```

mysql-py> db.city.select(["Name", "CountryCode"]).where(
    "Name like :name").bind("name", "Z%")

```

ヒント

プログラム内でバインディングを使用すると、式にプレースホルダを指定できます。プレースホルダには、実行前に値が入力され、必要に応じて自動エスケープを利用できます。

入力をサニタイズするには、常にバインディングを使用します。文字列連結を使用してクエリーに値を導入しないでください。無効な入力生成され、場合によってはセキュリティの問題が発生することがあります。

プロジェクト結果

AND 演算子を使用してクエリーを発行するには、`where()` メソッドの検索条件の間に演算子を追加します。

```

mysql-py> db.city.select(["Name", "CountryCode"]).where(
    "Name like 'Z%' and CountryCode = 'CHN'")
+-----+-----+
| Name      | CountryCode |
+-----+-----+
| Zhengzhou | CHN          |
| Zibo      | CHN          |
| Zhangjiakou | CHN        |
| Zhuzhou  | CHN          |
| Zhangjiang | CHN          |
| Zigong   | CHN          |
| Zaozhuang | CHN          |
| ...
| Zhangjiagang | CHN        |
+-----+-----+
22 rows in set (0.01 sec)

```

複数の条件演算子を指定するには、検索条件をカッコで囲んで演算子の優先順位を変更します。次の例は、**AND** および **OR** 演算子の配置を示しています。

```

mysql-py> db.city.select(["Name", "CountryCode"]).where(
    "Name like 'Z%' and (CountryCode = 'CHN' or CountryCode = 'RUS')")
+-----+-----+
| Name      | CountryCode |
+-----+-----+
| Zhengzhou | CHN          |
| Zibo      | CHN          |
| Zhangjiakou | CHN        |
| Zhuzhou  | CHN          |
| ...
| Zeleznogorsk | RUS        |
+-----+-----+
29 rows in set (0.01 sec)

```

制限、順序およびオフセットの結果

`limit()`、`order_by()` および `offset()` メソッドを適用して、`select()` メソッドによって返されるレコードの数と順序を管理できます。

結果セットに含まれるレコード数を指定するには、`limit()` メソッドに `select()` メソッドの値を追加します。たとえば、次のクエリーでは、国テーブルの最初の 5 つのレコードが返されます。

```
mysql-py> db.country.select(["Code", "Name"]).limit(5)
+-----+
| Code | Name |
+-----+
| ABW | Aruba |
| AFG | Afghanistan |
| AGO | Angola |
| AIA | Anguilla |
| ALB | Albania |
+-----+
5 rows in set (0.00 sec)
```

結果の順序を指定するには、`order_by()` メソッドを `select()` メソッドに追加します。必要に応じて、降順 (`desc`) または昇順 (`asc`) 属性でソートするカラムのリストを `order_by()` メソッドに渡します。昇順がデフォルトのオーダータイプです。

たとえば、次のクエリーでは、すべてのレコードが名前カラムでソートされ、最初の 3 つのレコードが降順で返されます。

```
mysql-py> db.country.select(["Code", "Name"]).order_by(["Name desc"]).limit(3)
+-----+
| Code | Name |
+-----+
| ZWE | Zimbabwe |
| ZMB | Zambia |
| YUG | Yugoslavia |
+-----+
3 rows in set (0.00 sec)
```

デフォルトでは、`limit()` メソッドはテーブルの最初のレコードから開始されます。`offset()` メソッドを使用して、開始レコードを変更できます。たとえば、最初のレコードを無視し、条件に一致する次の 3 つのレコードを返すには、`offset()` メソッドに値 1 を渡します。

```
mysql-py> db.country.select(["Code", "Name"]).order_by(["Name desc"]).limit(3).offset(1)
+-----+
| Code | Name |
+-----+
| ZMB | Zambia |
| YUG | Yugoslavia |
| YEM | Yemen |
+-----+
3 rows in set (0.00 sec)
```

関連情報

- [MySQL Reference Manual](#) には、関数および演算子に関する詳細なドキュメントが用意されています。
- 完全な構文の定義は、[TableSelectFunction](#) を参照してください。

20.4.4.3 テーブルの更新

`update()` メソッドを使用して、テーブル内のレコードを変更できます。`update()` メソッドは、更新するレコードのみを含めるようにクエリーをフィルタリングし、指定した操作をそれらのレコードに適用することで機能します。

市区町村テーブルの市区町村名を置換するには、`set()` メソッドに新しい市区町村名を渡します。次に、検索および置換する市区町村名を `where()` メソッドに渡します。次の例では、Peking 市を Beijing に置き換えます。

```
mysql-py> db.city.update().set("Name", "Beijing").where("Name = 'Peking'")
```

`select()` メソッドを使用して変更を確認します。

```
mysql-py> db.city.select(["ID", "Name", "CountryCode", "District", "Info"]).where("Name = 'Beijing'")
```

```
+-----+-----+-----+-----+
| ID | Name | CountryCode | District | Info |
+-----+-----+-----+-----+
| 1891 | Beijing | CHN | Peking | {"Population": 7472000} |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

関連情報

- 完全な構文の定義は、[TableUpdateFunction](#) を参照してください。

20.4.4.4 テーブルの削除

`delete()` メソッドを使用して、データベース内のテーブルから一部またはすべてのレコードを削除できます。X DevAPI には、削除するレコードをフィルタおよび順序付けするために `delete()` メソッドとともに使用する追加のメソッドが用意されています。

条件を使用したレコードの削除

次の例では、検索条件を `delete()` メソッドに渡します。条件に一致するすべてのレコードが `city` テーブルから削除されます。この例では、1つのレコードが条件に一致します。

```
mysql-py> db.city.delete().where("Name = 'Olympia'")
```

最初のレコードの削除

市区町村テーブルの最初のレコードを削除するには、値が 1 の `limit()` メソッドを使用します。

```
mysql-py> db.city.delete().limit(1)
```

テーブル内のすべてのレコードの削除

テーブル内のすべてのレコードを削除できます。これを行うには、検索条件を指定せずに `delete()` メソッドを使用します。

注意

検索条件を指定せずにレコードを削除する場合は注意が必要です。削除すると、テーブルからすべてのレコードが削除されます。

テーブルの削除

`drop_collection()` メソッドは、データベースからリレーショナルテーブルを削除するために MySQL Shell でも使用されます。たとえば、`world_x` データベースから `citytest` テーブルを削除するには、次のように発行します:

```
mysql-py> db.drop_collection("citytest")
```

関連情報

- 完全な構文の定義は、[TableDeleteFunction](#) を参照してください。
- `world_x` データベースを再作成する手順は、[セクション20.4.2 「world_x データベースのダウンロードおよびインポート」](#) を参照してください。

20.4.5 テーブル内のドキュメント

MySQL では、テーブルに従来のリレーショナルデータまたは JSON 値 (あるいはその両方) が含まれる場合があります。ネイティブ JSON データ型を持つカラムにドキュメントを格納することで、従来のデータを JSON ドキュメントと組み合わせることができます。

このセクションの例では、`world_x` スキーマの `city` テーブルを使用します。

市区町村テーブル摘要

city テーブルには、5 つのカラム (またはフィールド) があります。

Field	Type	Null	Key	Default	Extra
ID	int(11)	NO	PRI	null	auto_increment
Name	char(35)	NO			
CountryCode	char(3)	NO			
District	char(20)	NO			
Info	json	YES		null	

レコードの挿入

テーブルのカラムにドキュメントを挿入するには、正しい順序で整形形式の JSON ドキュメントを `values()` メソッドに渡します。次の例では、情報カラムに挿入される最終値としてドキュメントが渡されます。

```
mysql-py> db.city.insert().values(
None, "San Francisco", "USA", "California", '{"Population":830000}')
```

レコードの選択

式のドキュメント値を評価する検索条件を指定してクエリーを発行できます。

```
mysql-py> db.city.select(["ID", "Name", "CountryCode", "District", "Info"]).where(
"CountryCode = :country and Info->'$.Population' > 1000000").bind(
'country', 'USA')
```

ID	Name	CountryCode	District	Info
3793	New York	USA	New York	{ "Population": 8008278 }
3794	Los Angeles	USA	California	{ "Population": 3694820 }
3795	Chicago	USA	Illinois	{ "Population": 2896016 }
3796	Houston	USA	Texas	{ "Population": 1953631 }
3797	Philadelphia	USA	Pennsylvania	{ "Population": 1517550 }
3798	Phoenix	USA	Arizona	{ "Population": 1321045 }
3799	San Diego	USA	California	{ "Population": 1223400 }
3800	Dallas	USA	Texas	{ "Population": 1188580 }
3801	San Antonio	USA	Texas	{ "Population": 1144646 }

9 rows in set (0.01 sec)

関連情報

- 詳しくは [Working with Relational Tables and Documents](#) をご覧ください。
- データ型の詳細は、[セクション11.5「JSON データ型」](#) を参照してください。

20.5 X プラグイン

このセクションでは、X プラグイン を使用、構成および監視する方法について説明します。

20.5.1 X プラグイン のインストールの確認

X プラグイン は MySQL 8 ではデフォルトで有効になっているため、MySQL 8 をインストールまたはアップグレードするとプラグインが使用可能になります。X プラグイン が MySQL サーバーのインスタンスにインストールされていることを確認するには、`SHOW plugins` ステートメントを使用してプラグインリストを表示します。

MySQL Shell を使用して X プラグイン がインストールされていることを確認するには、次のコマンドを発行します：

```
shell> mysqlsh -u user --sqlc -P 3306 -e "SHOW plugins"
```

MySQL Client を使用して X プラグイン がインストールされていることを確認するには、次のコマンドを発行します：

```
shell> mysql -u user -p -e "SHOW plugins"
```

X プラグイン がインストールされている場合の結果の例を次に示します:

```
+-----+-----+-----+-----+
| Name      | Status | Type      | Library | License |
+-----+-----+-----+-----+
...
| mysqlx    | ACTIVE | DAEMON    | NULL    | GPL     |
...
+-----+-----+-----+-----+
```

20.5.2 X プラグイン の無効化

起動時に X プラグイン を無効にするには、MySQL 構成ファイルで `mysqlx=0` を設定するか、MySQL サーバーの起動時に `--mysqlx=0` または `--skip-mysqlx` を渡します。

または、`-DWITH_MYSQLX=OFF` CMake オプションを使用して、X プラグイン なしで MySQL Server をコンパイルします。

20.5.3 X プラグイン での暗号化接続の使用

このセクションでは、暗号化された接続を使用するように X プラグイン を構成する方法について説明します。背景情報の詳細は、[セクション6.3「暗号化された接続の使用」](#) を参照してください。

暗号化された接続の構成サポートを有効にするために、X プラグイン には、MySQL Server で使用される `ssl_xxx` システム変数とは異なる値を持つことができる `mysqlx_ssl_xxx` システム変数があります。たとえば、X プラグイン には、MySQL Server で使用されるファイルとは異なる SSL キー、証明書および認証局ファイルを含めることができます。これらの変数については、[セクション20.5.6.2「X プラグイン のオプションとシステム変数」](#) を参照してください。同様に、X プラグイン には、MySQL Server encrypted-connection `Ssl_xxx` ステータス変数に対応する独自の `Mysqlx_ssl_xxx` ステータス変数があります。[セクション20.5.6.3「X プラグイン ステータス変数」](#) を参照してください。

初期化時に、X プラグイン は暗号化された接続の TLS コンテキストを次のように決定します:

- すべての `mysqlx_ssl_xxx` システム変数にデフォルト値がある場合、X プラグイン は、`ssl_xxx` システム変数の値によって決定される MySQL Server メイン接続インタフェースと同じ TLS コンテキストを使用します。
- いずれかの `mysqlx_ssl_xxx` 変数にデフォルト以外の値がある場合、X プラグイン は独自のシステム変数の値で定義された TLS コンテキストを使用します。(これは、`mysqlx_ssl_xxx` システム変数がデフォルトとは異なる値に設定されている場合です。)

つまり、X プラグイン が有効になっているサーバーでは、`ssl_xxx` 変数のみを設定して MySQL プロトコルと X プロトコル 接続で同じ暗号化構成を共有するか、`ssl_xxx` 変数と `mysqlx_ssl_xxx` 変数を個別に構成して MySQL プロトコルと X プロトコル 接続の暗号化構成を個別に共有するかを選択できます。

MySQL プロトコルと X プロトコル 接続で同じ暗号化構成を使用するには、`my.cnf` で `ssl_xxx` システム変数のみを設定します:

```
[mysqld]
ssl_ca=ca.pem
ssl_cert=server-cert.pem
ssl_key=server-key.pem
```

MySQL プロトコルと X プロトコル 接続の暗号化を個別に構成するには、`my.cnf` で `ssl_xxx` と `mysqlx_ssl_xxx` の両方のシステム変数を設定します:

```
[mysqld]
ssl_ca=ca1.pem
ssl_cert=server-cert1.pem
```

```
ssl_key=server-key1.pem  
  
mysqlx_ssl_ca=ca2.pem  
mysqlx_ssl_cert=server-cert2.pem  
mysqlx_ssl_key=server-key2.pem
```

connection-encryption サポートの構成に関する一般情報は、[セクション6.3.1「暗号化接続を使用するための MySQL の構成」](#)を参照してください。この説明は MySQL Server 用に記述されていますが、パラメータ名は X プラグイン用に似ています。(X プラグイン `mysqlx_ssl_xxx` システム変数名は、MySQL Server `ssl_xxx` システム変数名に対応します。)

MySQL プロトコル接続に許可される TLS バージョンを決定する `tls_version` システム変数は、X プロトコル 接続にも適用されます。したがって、両方のタイプの接続で許可される TLS バージョンは同じです。

接続ごとの暗号化はオプションですが、特定のユーザーは、ユーザーを作成する `CREATE USER` ステートメントに適切な `REQUIRE` 句を含めることで、X プロトコル および MySQL プロトコル接続に暗号化を使用する必要があります。詳細は、[セクション13.7.1.3「CREATE USER ステートメント」](#)を参照してください。または、すべてのユーザーに X プロトコル および MySQL プロトコル接続の暗号化の使用を要求するには、`require_secure_transport` システム変数を有効にします。追加情報については [暗号化された接続の必須としての構成](#)を参照してください。

20.5.4 Caching SHA-2 認証プラグインでの X プラグイン の使用

X プラグイン は、`caching_sha2_password` 認証プラグインで作成された MySQL ユーザーアカウントをサポートしています。このプラグインの詳細は、[セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#)を参照してください。X プラグインを使用すると、`SHA256_MEMORY` 認証で非 SSL 接続を使用し、`PLAIN` 認証で SSL 接続を使用し、このようなアカウントに対して認証できます。

`caching_sha2_password` 認証プラグインは認証キャッシュを保持しますが、このキャッシュは X プラグイン と共有されないため、X プラグイン は `SHA256_MEMORY` 認証に独自の認証キャッシュを使用します。X プラグイン 認証キャッシュにはユーザーアカウントパスワードのハッシュが格納され、SQL を使用してアクセスできません。ユーザーアカウントが変更または削除されると、関連するエントリがキャッシュから削除されます。X プラグイン 認証キャッシュは、デフォルトで有効になっている `mysqlx_cache_cleaner` プラグインによって保持され、関連するシステム変数またはステータス変数はありません。

非 SSL X プロトコル 接続を使用して `caching_sha2_password` 認証プラグインを使用するアカウントを認証するには、X プラグイン 認証キャッシュにパスワードを指定するために、そのアカウントが SSL を使用した X プロトコル 接続を介して少なくとも一度認証されている必要があります。SSL 経由のこの初期認証が成功すると、非 SSL X プロトコル 接続を使用できます。

オプション `--mysqlx_cache_cleaner=0` を指定して MySQL サーバーを起動すると、`mysqlx_cache_cleaner` プラグインを無効にできます。これを行うと、X プラグイン 認証キャッシュが無効になるため、`SHA256_MEMORY` 認証を使用した認証時には常に SSL を X プロトコル 接続に使用する必要があります。

20.5.5 X プラグイン での接続圧縮

MySQL 8.0.19 から、X プラグイン は X プロトコル 接続を介して送信されるメッセージの圧縮をサポートします。サーバーとクライアントが相互にサポートされている圧縮アルゴリズムに同意する場合、接続を圧縮できます。圧縮を有効にすると、ネットワーク経由で送信されるバイト数が削減されますが、圧縮および解凍操作のための CPU コストがサーバーおよびクライアントに追加されます。したがって、圧縮の利点は、主にネットワーク帯域幅が低く、ネットワーク転送時間が圧縮および解凍操作のコストを支配し、結果セットが大きい場合に発生します。

注記

MySQL クライアントによって、接続圧縮のサポートが異なる方法で実装されます。詳細は、クライアントのドキュメントを参照してください。たとえば、クラシック MySQL プロトコル 接続の場合は、[セクション4.2.8「接続圧縮制御」](#)を参照してください。

- [X プラグイン の接続圧縮の構成](#)
- [X プラグイン の圧縮接続特性](#)
- [X プラグイン の接続圧縮の監視](#)

X プラグイン の接続圧縮の構成

デフォルトでは、X プラグイン は `zstd`、`LZ4` および `Deflate` 圧縮アルゴリズムをサポートしています。Deflate アルゴリズムを使用した圧縮は `zlib` ソフトウェアライブラリを使用して実行されるため、X プロトコル 接続の `deflate_stream` 圧縮アルゴリズム設定は クラシック MySQL プロトコル 接続の `zlib` 設定と同等です。

サーバー側では、`mysqlx_compression_algorithms` システム変数に許可された圧縮アルゴリズムのみを含めるように設定することで、圧縮アルゴリズムを禁止できます。アルゴリズム名 `zstd_stream`、`lz4_message` および `deflate_stream` は任意の組合せで指定でき、順序と大文字と小文字は重要ではありません。システム変数値が空の文字列の場合、圧縮アルゴリズムは許可されず、接続は圧縮解除されます。

次のテーブルでは、様々な圧縮アルゴリズムの特性を比較し、割り当てられた優先度を示します。デフォルトでは、サーバーとクライアントで共通に許可されているもっとも優先度の高いアルゴリズムがサーバーによって選択されます。クライアントはあとで説明するように優先順位を変更できます。アルゴリズムの短い形式のエイリアスは、指定時にクライアントで使用できます。

表 20.1 X プロトコル 圧縮アルゴリズムの特性

アルゴリズム	エイリアス	圧縮率	スループット	CPU コスト	デフォルト優先度
<code>zsth_stream</code>	<code>zstd</code>	高	高	中	First
<code>lz4_message</code>	<code>lz4</code>	低	高	最低	秒
<code>deflate_stream</code>	<code>deflate</code>	高	低	最高	Third

許可される圧縮アルゴリズムの X プロトコル セット (ユーザー指定かデフォルトかに関係なく) は、`protocol_compression_algorithms` サーバーシステム変数で指定される クラシック MySQL プロトコル 接続用に MySQL Server で許可される圧縮アルゴリズムのセットとは無関係です。`mysqlx_compression_algorithms` システム変数を指定しない場合、X プラグイン は クラシック MySQL プロトコル 接続の圧縮設定の使用にフォールバックしません。かわりに、デフォルトでは、表20.1「X プロトコル 圧縮アルゴリズムの特性」に表示されているすべてのアルゴリズムが許可されます。これは、セクション20.5.3「X プラグイン での暗号化接続の使用」で説明されているように、X プラグイン システム変数が設定されていない場合に MySQL Server 設定が使用される TLS コンテキストの状況とは異なります。クラシック MySQL プロトコル 接続の圧縮の詳細は、セクション4.2.8「接続圧縮制御」を参照してください。

クライアント側では、X プロトコル 接続リクエストで圧縮制御用に複数のパラメータを指定できます:

- 圧縮モード。
- 圧縮レベル (MySQL 8.0.20)。
- 許可されている圧縮アルゴリズムの優先度順のリスト (MySQL 8.0.22)。

注記

一部のクライアントまたはコネクタでは、特定の圧縮制御機能がサポートされていない場合があります。たとえば、X プロトコル 接続の圧縮レベルの指定は MySQL Shell でのみサポートされ、他の MySQL クライアントやコネクタではサポートされません。サポートされる機能およびその使用方法の詳細は、特定の製品のドキュメントを参照してください。

接続モードには、次の値があります:

- `disabled`: 接続が圧縮解除されます。
- `preferred`: サーバーとクライアントは、両方で許可されている圧縮アルゴリズムを見つけるためにネゴシエーションを行います。使用可能な共通アルゴリズムがない場合、接続は圧縮解除されます。これは、明示的に指定されていない場合のデフォルトモードです。
- `required`: 圧縮アルゴリズムのネゴシエーションは `preferred` モードと同様に行われますが、使用可能な共通アルゴリズムがない場合、接続要求はエラーで終了します。

各接続の圧縮アルゴリズムに同意するだけでなく、サーバーとクライアントは、合意されたアルゴリズムに適用される数値範囲の圧縮レベルに同意できます。アルゴリズムの圧縮レベルが高くなるにつれて、データ圧縮率が高くなり、クライアントへのメッセージの送信に必要なネットワーク帯域幅と転送時間が短縮されます。ただし、データ

圧縮に必要な労力も増加し、サーバー上の時間と CPU およびメモリーリソースが消費されます。圧縮作業の増加には、圧縮率の増加と線形関係はありません。

MySQL 8.0.19 では、X プラグイン は常にアルゴリズムごとにライブラリのデフォルトの圧縮レベルを使用し (zstd の場合は 3、LZ4 の場合は 0、Deflate の場合は 6)、クライアントはこれをネゴシエートできません。MySQL 8.0.20 から、クライアントは X プロトコル 接続のためにサーバーとの機能ネゴシエーション中に特定の圧縮レベルをリクエストできます。

MySQL 8.0.20 の X プラグイン で使用されるデフォルトの圧縮レベルは、圧縮時間とネットワーク転送時間の間のトレードオフとしてパフォーマンステストで選択されています。これらのデフォルトは、各アルゴリズムのライブラリのデフォルトと必ずしも同じではありません。これらは、クライアントがアルゴリズムの圧縮レベルをリクエストしない場合に適用されます。デフォルトの圧縮レベルは、zstd の場合は 3、LZ4 の場合は 2、Deflate の場合は 3 に初期設定されます。これらの設定は、`mysqlx_zstd_default_compression_level`、`mysqlx_lz4_default_compression_level` および `mysqlx_deflate_default_compression_level` システム変数を使用して調整できます。

サーバーでの過剰なリソース消費を防ぐために、X プラグイン では、サーバーがアルゴリズムごとに許可する最大圧縮レベルを設定します。クライアントがこの設定を超える圧縮レベルをリクエストした場合、サーバーは最大許容圧縮レベルを使用します (クライアントによる圧縮レベルのリクエストは MySQL Shell でのみサポートされます)。最大圧縮レベルは、zstd の場合は 11、LZ4 の場合は 8、Deflate の場合は 5 に初期設定されます。これらの設定は、`mysqlx_zstd_max_client_compression_level`、`mysqlx_lz4_max_client_compression_level` および `mysqlx_deflate_max_client_compression_level` システム変数を使用して調整できます。

サーバーとクライアントで共通のアルゴリズムが複数許可されている場合、ネゴシエーション中にアルゴリズムを選択するためのデフォルトの優先順位が表 20.1 「X プロトコル 圧縮アルゴリズムの特性」に表示されます。MySQL 8.0.22 から、圧縮アルゴリズムの指定をサポートするクライアントの場合、接続リクエストには、アルゴリズム名またはそのエイリアスを使用して指定された、クライアントで許可されるアルゴリズムのリストを含めることができます。リスト内のこれらのアルゴリズムの順序は、サーバーによる優先順位とみなされます。この場合に使用されるアルゴリズムは、サーバー側でも許可されているクライアントリストの最初のアルゴリズムです。ただし、圧縮アルゴリズムのオプションには圧縮モードが適用されます:

- 圧縮モードが `disabled` の場合、圧縮アルゴリズムオプションは無視されます。
- 圧縮モードが `preferred` であるが、クライアント側で許可されているアルゴリズムがサーバー側で許可されていない場合、接続は圧縮解除されます。
- 圧縮モードが `required` であるが、クライアント側で許可されているアルゴリズムがサーバー側で許可されていない場合は、エラーが発生します。

メッセージ圧縮の影響を監視するには、[X プラグイン の接続圧縮の監視](#) で説明されている X プラグイン ステータス変数を使用します。これらのステータス変数を使用して、現在の設定でのメッセージ圧縮の利点を計算し、その情報を使用して設定をチューニングできます。

X プラグイン の圧縮接続特性

X プロトコル 接続圧縮は、次の動作および境界で動作します:

- アルゴリズム名の `_stream` および `_message` 接尾辞は、2 つの異なる操作モードを参照: ストリームモードでは、単一の接続内のすべての X プロトコル メッセージが連続ストリームに圧縮され、圧縮された順序をスキップせずに圧縮解除する必要があります。メッセージモードでは、各メッセージは個別に圧縮され、圧縮された順序で解凍する必要はありません。また、メッセージモードでは、圧縮されたすべてのメッセージを解凍する必要はありません。
- 圧縮は、認証が成功する前に送信されるメッセージには適用されません。
- 圧縮は、`Mysqlx.Ok`、`Mysqlx.Error`、`Mysqlx.Sql.StmtExecuteOk` メッセージなどの制御フローメッセージには適用されません。
- 他のすべての X プロトコル メッセージは、サーバーとクライアントが機能ネゴシエーション中に相互に許可された圧縮アルゴリズムに同意した場合に圧縮できます。その段階でクライアントが圧縮を要求しない場合、クライアントもサーバーもメッセージに圧縮を適用しません。
- X プロトコル 接続を介して送信されるメッセージが圧縮されても、`mysqlx_max_allowed_packet` システム変数で指定された制限が適用されます。メッセージペイロードが解凍された後、ネットワークパケットはこの制限より小さくする必要があります。制限を超えると、X プラグイン は解凍エラーを返し、接続を閉じます。

- 次の点は、MySQL Shell でのみサポートされているクライアントによる圧縮レベルのリクエストに関連しています:
 - 圧縮レベルは、クライアントで整数として指定する必要があります。他のタイプの値が指定された場合、接続はエラーでクローズします。
 - クライアントがアルゴリズムを指定し、圧縮レベルを指定しない場合、サーバーはアルゴリズムにデフォルトの圧縮レベルを使用します。
 - クライアントがサーバーの最大許容レベルを超えるアルゴリズム圧縮レベルをリクエストした場合、サーバーは最大許容レベルを使用します。
 - クライアントが、サーバーの最小許容レベルより低いアルゴリズム圧縮レベルをリクエストした場合、サーバーは最小許容レベルを使用します。

X プラグイン の接続圧縮の監視

X プラグイン ステータス変数を使用して、メッセージ圧縮の影響を監視できます。メッセージ圧縮が使用されている場合、セッション `Mysqlx_compression_algorithm` ステータス変数には現在の X プロトコル 接続に使用されている圧縮アルゴリズムが表示され、`Mysqlx_compression_level` には選択された圧縮レベルが表示されます。これらのセッションステータス変数は、MySQL 8.0.20 から使用できます。

MySQL 8.0.19 から、X プラグイン ステータス変数を使用して、選択した圧縮アルゴリズムの効率 (データ圧縮率) およびメッセージ圧縮を使用した全体的な影響を計算できます。次の計算でステータス変数のセッション値を使用して、既知の圧縮アルゴリズムを使用した特定のセッションに対するメッセージ圧縮の利点を確認します。または、ステータス変数のグローバル値を使用して、X プロトコル 接続を使用するすべてのセッション (これらのセッションに使用されているすべての圧縮アルゴリズム、およびメッセージ圧縮を使用しなかったすべてのセッションを含む) にわたるサーバーのメッセージ圧縮の全体的な利点を確認します。その後、[X プラグイン の接続圧縮の構成](#) で説明されているように、許可されている圧縮アルゴリズム、最大圧縮レベルおよびデフォルトの圧縮レベルを調整して、メッセージ圧縮をチューニングできます。

メッセージ圧縮が使用されている場合、`Mysqlx_bytes_sent` ステータス変数には、圧縮後に測定された圧縮メッセージペイロード、圧縮されなかった X プロトコル ヘッダーなどの圧縮メッセージ内のアイテム、および圧縮されていないメッセージを含む、サーバーから送信された合計バイト数が表示されます。`Mysqlx_bytes_sent_compressed_payload` ステータス変数は圧縮されたメッセージペイロードとして送信されたバイトの合計数を示し、`Mysqlx_bytes_sent_uncompressed_frame` ステータス変数は圧縮後に測定された同じメッセージペイロードの合計バイト数を示しますが、圧縮前に測定されます。したがって、圧縮アルゴリズムの効率を示す圧縮率は、次の式を使用して計算できます:

```
mmysqlx_bytes_sent_uncompressed_frame / mysqlx_bytes_sent_compressed_payload
```

サーバーから送信される X プロトコル メッセージの圧縮の有効性は、次の式を使用して計算できます:

```
(mysqlx_bytes_sent - mysqlx_bytes_sent_compressed_payload + mysqlx_bytes_sent_uncompressed_frame) / mysqlx_bytes_sent
```

サーバーがクライアントから受信したメッセージの場合、`Mysqlx_bytes_received_compressed_payload` ステータス変数は圧縮メッセージペイロードとして受信したバイトの合計数を示し、圧縮解除前に測定され、`Mysqlx_bytes_received_uncompressed_frame` ステータス変数は同じメッセージペイロードの合計バイト数を示しますが、解凍後に測定されます。`Mysqlx_bytes_received` ステータス変数には、圧縮解除前に測定された圧縮メッセージペイロード、圧縮メッセージ内の圧縮されていないアイテム、および圧縮されていないメッセージが含まれます。

20.5.6 X プラグイン のオプションおよび変数

このセクションでは、X プラグイン を構成するコマンドオプションとシステム変数、および監視目的で使用可能なステータス変数について説明します。起動時に指定された構成値が正しくない場合、X プラグイン は適切に初期化できず、サーバーがロードしません。この場合、サーバーは他の X プラグイン 設定を認識できないため、エラーメッセージを生成することもできます。

20.5.6.1 X プラグイン オプションおよび変数リファレンス

このテーブルは、X プラグイン で提供されるコマンドオプション、システム変数およびステータス変数の概要を示しています。

表 20.2 「X プラグイン オプションおよび変数リファレンス」

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
mysqlx	はい	はい				
mysqlx_aborted_clients				はい	グローバル	いいえ
mysqlx_address				はい	グローバル	いいえ
mysqlx_bind_address	はい	はい	はい		グローバル	いいえ
mysqlx_bytes_received				はい	両方	いいえ
mysqlx_bytes_received_compressed_payload				はい	両方	いいえ
mysqlx_bytes_received_uncompressed_frame				はい	両方	いいえ
mysqlx_bytes_sent				はい	両方	いいえ
mysqlx_bytes_sent_compressed_payload				はい	両方	いいえ
mysqlx_bytes_sent_uncompressed_frame				はい	両方	いいえ
mysqlx_compression_algorithm				はい	セッション	いいえ
mysqlx_compression_algorithms	はい	はい	はい		グローバル	はい
mysqlx_compression_level				はい	セッション	いいえ
mysqlx_connect_timeout	はい	はい	はい		グローバル	はい
mysqlx_connection_accept_errors				はい	両方	いいえ
mysqlx_connection_errors				はい	両方	いいえ
mysqlx_connections_accepted				はい	グローバル	いいえ
mysqlx_connections_closed				はい	グローバル	いいえ
mysqlx_connections_rejected				はい	グローバル	いいえ
mysqlx_crud_create_view				はい	両方	いいえ
mysqlx_crud_delete				はい	両方	いいえ
mysqlx_crud_drop_view				はい	両方	いいえ
mysqlx_crud_find				はい	両方	いいえ
mysqlx_crud_insert				はい	両方	いいえ
mysqlx_crud_modify_view				はい	両方	いいえ
mysqlx_crud_update				はい	両方	いいえ
mysqlx_deflate_default_compression_level	はい	はい	はい		グローバル	はい
mysqlx_deflate_max_client_compression_level	はい	はい	はい		グローバル	はい
mysqlx_document_id_unique_prefix	はい	はい	はい		グローバル	はい
mysqlx_enable_half_notice	はい	はい	はい		グローバル	はい
mysqlx_errors_sent				はい	両方	いいえ
mysqlx_errors_unknown_message_type				はい	両方	いいえ
mysqlx_expect_close				はい	両方	いいえ
mysqlx_expect_open				はい	両方	いいえ
mysqlx_idle_worker_thread_timeout	はい	はい	はい		グローバル	はい
mysqlx_init_error				はい	両方	いいえ
mysqlx_interactive_timeout	はい	はい	はい		グローバル	はい
mysqlx_lz4_default_compression_level	はい	はい	はい		グローバル	はい
mysqlx_lz4_max_client_compression_level	はい	はい	はい		グローバル	はい

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
mysqlx_max_allowed_packet	はい	はい	はい		グローバル	はい
mysqlx_max_connections	はい	はい	はい		グローバル	はい
Mysqlx_messages_sent				はい	両方	いいえ
mysqlx_min_work_threads	はい	はい	はい		グローバル	はい
Mysqlx_notice_global_sent				はい	両方	いいえ
Mysqlx_notice_other_sent				はい	両方	いいえ
Mysqlx_notice_warning_sent				はい	両方	いいえ
Mysqlx_notified_by_group_replication				はい	両方	いいえ
Mysqlx_port				はい	グローバル	いいえ
mysqlx_port	はい	はい	はい		グローバル	いいえ
mysqlx_port_timeout	はい	はい	はい		グローバル	いいえ
mysqlx_read_timeout	はい	はい	はい		セッション	はい
Mysqlx_rows_sent				はい	両方	いいえ
Mysqlx_sessions				はい	グローバル	いいえ
Mysqlx_sessions_accepted				はい	グローバル	いいえ
Mysqlx_sessions_closed				はい	グローバル	いいえ
Mysqlx_sessions_fatal_error				はい	グローバル	いいえ
Mysqlx_sessions_killed				はい	グローバル	いいえ
Mysqlx_sessions_rejected				はい	グローバル	いいえ
Mysqlx_socket				はい	グローバル	いいえ
mysqlx_socket	はい	はい	はい		グローバル	いいえ
Mysqlx_ssl_accept_renegotiates				はい	グローバル	いいえ
Mysqlx_ssl_accepts				はい	グローバル	いいえ
Mysqlx_ssl_active				はい	両方	いいえ
mysqlx_ssl_ca	はい	はい	はい		グローバル	いいえ
mysqlx_ssl_capath	はい	はい	はい		グローバル	いいえ
mysqlx_ssl_cert	はい	はい	はい		グローバル	いいえ
Mysqlx_ssl_cipher				はい	両方	いいえ
mysqlx_ssl_cipher	はい	はい	はい		グローバル	いいえ
Mysqlx_ssl_cipher_list				はい	両方	いいえ
mysqlx_ssl_crl	はい	はい	はい		グローバル	いいえ
mysqlx_ssl_crlpath	はい	はい	はい		グローバル	いいえ
Mysqlx_ssl_ctx_verify_depth				はい	両方	いいえ
Mysqlx_ssl_ctx_verify_mode				はい	両方	いいえ
Mysqlx_ssl_finished_accepts				はい	グローバル	いいえ
mysqlx_ssl_key	はい	はい	はい		グローバル	いいえ
Mysqlx_ssl_server_not_after				はい	グローバル	いいえ
Mysqlx_ssl_server_not_before				はい	グローバル	いいえ
Mysqlx_ssl_verify_depth				はい	グローバル	いいえ
Mysqlx_ssl_verify_mode				はい	グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
Mysqlx_ssl_version				はい	両方	いいえ
Mysqlx_stmt_create_collection				はい	両方	いいえ
Mysqlx_stmt_create_collection_index				はい	両方	いいえ
Mysqlx_stmt_disable_notices				はい	両方	いいえ
Mysqlx_stmt_drop_collection				はい	両方	いいえ
Mysqlx_stmt_drop_collection_index				はい	両方	いいえ
Mysqlx_stmt_enable_notices				はい	両方	いいえ
Mysqlx_stmt_ensure_collection				はい	両方	いいえ
Mysqlx_stmt_execute_mysqlx				はい	両方	いいえ
Mysqlx_stmt_execute_sql				はい	両方	いいえ
Mysqlx_stmt_execute_xplugin				はい	両方	いいえ
Mysqlx_stmt_get_collection_options				はい	両方	いいえ
Mysqlx_stmt_kill_client				はい	両方	いいえ
Mysqlx_stmt_list_clients				はい	両方	いいえ
Mysqlx_stmt_list_notices				はい	両方	いいえ
Mysqlx_stmt_list_objects				はい	両方	いいえ
Mysqlx_stmt_modify_collection_options				はい	両方	いいえ
Mysqlx_stmt_ping				はい	両方	いいえ
mysqlx_wait_timeout	はい	はい	はい		セッション	はい
Mysqlx_worker_threads				はい	グローバル	いいえ
Mysqlx_worker_threads_active				はい	グローバル	いいえ
mysqlx_write_timeout	はい	はい	はい		セッション	はい
mysqlx_zstd_default_compression_level	はい	はい	はい		グローバル	はい
mysqlx_zstd_max_allowed_compression_level	はい	はい	はい		グローバル	はい

20.5.6.2 X プラグイン のオプションとシステム変数

X プラグイン のアクティブ化を制御するには、このオプションを使用します:

- `--mysqlx[=value]`

コマンド行形式	<code>--mysqlx[=value]</code>
型	列挙
デフォルト値	ON
有効な値	ON OFF FORCE FORCE_PLUS_PERMANENT

このオプションは、起動時にサーバーが X プラグイン をロードする方法を制御します。MySQL 8.0 では、X プラグイン はデフォルトで有効になっていますが、このオプションを使用してアクティブ化状態を制御できます。

セクション5.6.1「プラグインのインストールおよびアンインストール」で説明したように、オプションの値は、プラグインのロードオプションに指定可能な値のいずれかである必要があります。

X プラグイン が有効になっている場合、操作を制御できるいくつかのシステム変数が公開されます:

- [mysqlx_bind_address](#)

コマンド行形式	<code>--mysqlx-bind-address=addr</code>
システム変数	mysqlx_bind_address
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	*

X プラグイン が TCP/IP 接続をリスニングするネットワークアドレス。この変数は動的ではなく、起動時のみ構成できます。これは [bind_address](#) システム変数と同等の X プラグイン です。詳細は、その変数の説明を参照してください。

デフォルトでは、X プラグイン は、すべてのサーバーホスト IPv4 インタフェースで TCP/IP 接続を受け入れ、サーバーホストが IPv6 をサポートしている場合は、すべての IPv6 インタフェースで TCP/IP 接続を受け入れます。[mysqlx_bind_address](#) が指定されている場合、その値は次の要件を満たす必要があります:

- MySQL 8.0.21 より前では、[mysqlx_bind_address](#) は、単一の非ワイルドカード IP アドレス (IPv4 または IPv6)、ホスト名、または複数のネットワークインタフェースでのリスニングを許可するワイルドカードアドレス形式 (*、0.0.0.0 または ::) を受け入れます。
- MySQL 8.0.21 では、[mysqlx_bind_address](#) は前述の単一の値またはカンマ区切り値のリストを受け入れます。変数が複数の値のリストを指定する場合、各値は単一のワイルドカード以外の IP アドレス (IPv4 または IPv6) またはホスト名を指定する必要があります。ワイルドカードアドレス書式 (*、0.0.0.0 または ::) は、値リストでは使用できません。
- MySQL 8.0.22 の時点では、値にネットワークネームスペース指定子が含まれる場合があります。

IP アドレスは、IPv4 または IPv6 アドレスとして指定できます。ホスト名である値の場合、X プラグイン は名前を IP アドレスに解決し、そのアドレスにバインドします。ホスト名が複数の IP アドレスに解決される場合、X プラグイン は最初の IPv4 アドレス (存在する場合) または最初の IPv6 アドレスを使用します。

X プラグイン では、様々なタイプのアドレスが次のように扱われます:

- アドレスが * の場合、X プラグイン はすべてのサーバーホスト IPv4 インタフェースで TCP/IP 接続を受け入れ、サーバーホストが IPv6 をサポートしている場合は、すべての IPv6 インタフェースで TCP/IP 接続を受け入れます。このアドレスを使用して、X プラグイン の IPv4 接続と IPv6 接続の両方を許可します。この値がデフォルトです。変数で複数の値のリストが指定されている場合、この値は許可されません。
- アドレスが 0.0.0.0 の場合、X プラグイン はすべてのサーバーホスト IPv4 インタフェースで TCP/IP 接続を受け入れます。変数で複数の値のリストが指定されている場合、この値は許可されません。
- アドレスが :: の場合、X プラグイン はすべてのサーバーホスト IPv4 および IPv6 インタフェースで TCP/IP 接続を受け入れます。変数で複数の値のリストが指定されている場合、この値は許可されません。
- アドレスが IPv4 マップアドレスの場合、X プラグイン はそのアドレスの TCP/IP 接続を IPv4 または IPv6 形式で受け入れます。たとえば、X プラグイン が ::ffff:127.0.0.1 にバインドされている場合、MySQL Shell などのクライアントは `--host=127.0.0.1` または `--host>::ffff:127.0.0.1` を使用して接続できます。
- アドレスが 「regular」 IPv4 アドレスまたは IPv6 アドレス (127.0.0.1、::1 など) の場合、X プラグイン はその IPv4 アドレスまたは IPv6 アドレスに対してのみ TCP/IP 接続を受け入れます。

アドレスのネットワークネームスペースの指定には、次のルールが適用されます:

- ネットワークネームスペースは、IP アドレスまたはホスト名に指定できます。
- ワイルドカード IP アドレスにはネットワークネームスペースを指定できません。

- 指定されたアドレスでは、ネットワーク名前空間はオプションです。指定する場合は、アドレスの直後に `/ns` 接尾辞として指定する必要があります。
- `/ns` 接尾辞のないアドレスは、ホストシステムのグローバル名前空間を使用します。したがって、グローバル名前空間がデフォルトです。
- `/ns` 接尾辞の付いたアドレスは、`ns` という名前の名前空間を使用します。
- ホストシステムはネットワーク名前空間をサポートしている必要があります。各名前付き名前空間は事前に設定されている必要があります。存在しない名前空間に名前を付けると、エラーが発生します。
- 変数値に複数のアドレスが指定されている場合は、グローバル名前空間、名前付き名前空間またはその組合せにアドレスを含めることができます。

ネットワーク名前空間の詳細は、[セクション5.1.14「ネットワーク名前空間のサポート」](#)を参照してください。

重要

X プラグイン は必須プラグインではないため、指定されたアドレスまたはアドレスリストにエラーがある場合、サーバーの起動は妨げられません (MySQL Server が `bind_address` エラーの場合と同様)。X プラグイン では、リストされたアドレスのいずれかを解析できない場合、または X プラグイン がアドレスにバインドできない場合、アドレスはスキップされ、エラーメッセージがログに記録され、X プラグイン は残りの各アドレスへのバインドを試行します。X プラグイン `Mysqlx_address` ステータス変数には、バインドが成功したアドレスのみがリストに表示されます。リストされたアドレスのいずれもバインドに成功しなかった場合、または単一の指定されたアドレスで障害が発生した場合、X プラグイン は X プロトコルを使用できないことを示すエラーメッセージ `ER_XPLUGIN_FAILED_TO_PREPARE_IO_INTERFACES` をログに記録します。`mysqlx_bind_address` は動的ではないため、問題を修正するには、サーバーを停止し、システム変数値を修正してサーバーを再起動する必要があります。

- [mysqlx_compression_algorithms](#)

コマンド行形式	<code>--mysqlx-compression-algorithms=value</code>
導入	8.0.19
システム変数	mysqlx_compression_algorithms
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Set
デフォルト値	deflate_stream , lz4_message , zstd_stream
有効な値	deflate_stream lz4_message zstd_stream

X プロトコル 接続で使用できる圧縮アルゴリズム。デフォルトでは、Deflate、LZ4 および zstd アルゴリズムはすべて許可されています。どのアルゴリズムも許可しないようにするには、許可するアルゴリズムのみを含むように [mysqlx_compression_algorithms](#) を設定します。アルゴリズム名 [deflate_stream](#)、[lz4_message](#) および [zstd_stream](#) は任意の組合せで指定でき、順序と大/小文字は重要ではありません。システム変数を空の文字列に設定した場合、圧縮アルゴリズムは許可されず、圧縮されていない接続のみが使用されます。アルゴリズム固有のシステム変数を使用して、許可される各アルゴリズムのデフォルトおよび最大の圧縮レベルを調整します。X プロトコルの接続圧縮と MySQL Server の同等の設定との関連の詳細は、[セクション20.5.5「X プラグイン での接続圧縮」](#)を参照してください。

- mysqlx_connect_timeout

コマンド行形式	--mysqlx-connect-timeout=#
システム変数	mysqlx_connect_timeout
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	30
最小値	1
最大値	1000000000

新しく接続されたクライアントから最初のパケットが受信されるまで X プラグイン が待機する秒数。これは `connect_timeout` と同等の X プラグイン です。詳細は、その変数の説明を参照してください。

- mysqlx_deflate_default_compression_level

コマンド行形式	--mysqlx_deflate_default_compression_level=#
導入	8.0.20
システム変数	mysqlx_deflate_default_compression_level
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	3
最小値	1
最大値	9

サーバーが X プロトコル 接続の Deflate アルゴリズムに使用するデフォルトの圧縮レベル。レベルは、1 (最小圧縮作業) から 9 (最大作業) の整数で指定します。このレベルは、クライアントが機能ネゴシエーション中に圧縮レベルをリクエストしない場合に使用されます。このシステム変数を指定しない場合、サーバーはレベル 3 をデフォルトとして使用します。詳細は、[セクション20.5.5「X プラグイン での接続圧縮」](#)を参照してください。

- mysqlx_deflate_max_client_compression_level

コマンド行形式	--mysqlx_deflate_max_client_compression_level=#
導入	8.0.20
システム変数	mysqlx_deflate_max_client_compression_level
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	5
最小値	1
最大値	9

サーバーが X プロトコル 接続の Deflate アルゴリズムに許可する最大圧縮レベル。範囲は、このアルゴリズムのデフォルトの圧縮レベルと同じです。クライアントがこれより高い圧縮レベルをリクエストした場合、サーバーはこ

ここで設定したレベルを使用します。このシステム変数を指定しない場合、サーバーは最大圧縮レベルを 5 に設定します。

- mysqlx_document_id_unique_prefix

コマンド行形式	--mysqlx-document-id-unique-prefix=#
システム変数	mysqlx_document_id_unique_prefix
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	65535

ドキュメントがコレクションに追加されたときにサーバーによって生成されるドキュメント ID の最初の 4 バイトを設定します。この変数をインスタンスごとに一意の値に設定することで、ドキュメント ID がインスタンス間で一意であることを確認できます。 [Understanding Document IDs](#) を参照してください。

- mysqlx_enable_hello_notice

コマンド行形式	--mysqlx-enable-hello-notice[={OFF ON}]
システム変数	mysqlx_enable_hello_notice
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

X プロトコル を介した接続を試行する クラシック MySQL プロトコル クライアントに送信されるメッセージを制御します。有効にすると、サーバーの X プロトコル ポートに接続しようとする X プロトコル をサポートしていないクライアントは、間違ったプロトコルを使用していることを説明するエラーを受け取ります。

- mysqlx_idle_worker_thread_timeout

コマンド行形式	--mysqlx-idle-worker-thread-timeout=#
システム変数	mysqlx_idle_worker_thread_timeout
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	60
最小値	0
最大値	3600

アイドル状態のワーカースレッドが終了するまでの秒数。

- mysqlx_interactive_timeout

コマンド行形式	--mysqlx-interactive-timeout=#
システム変数	mysqlx_interactive_timeout

スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	28800
最小値	1
最大値	2147483

対話型クライアントの `mysqlx_wait_timeout` セッション変数のデフォルト値。(対話型クライアントがタイムアウトするまで待機する秒数。)

- `mysqlx_lz4_default_compression_level`

コマンド行形式	<code>--mysqlx_lz4_default_compression_level=#</code>
導入	8.0.20
システム変数	<code>mysqlx_lz4_default_compression_level</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	2
最小値	0
最大値	16

サーバーが X プロトコル 接続の LZ4 アルゴリズムに使用するデフォルトの圧縮レベル。レベルは、0 (最小圧縮作業) から 16 (最大作業) の整数で指定します。このレベルは、クライアントが機能ネゴシエーション中に圧縮レベルをリクエストしない場合に使用されます。このシステム変数を指定しない場合、サーバーはレベル 2 をデフォルトとして使用します。詳細は、[セクション20.5.5「X プラグイン での接続圧縮」](#)を参照してください。

- `mysqlx_lz4_max_client_compression_level`

コマンド行形式	<code>--mysqlx_lz4_max_client_compression_level=#</code>
導入	8.0.20
システム変数	<code>mysqlx_lz4_max_client_compression_level</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	8
最小値	0
最大値	16

サーバーが X プロトコル 接続で LZ4 アルゴリズムに対して許可する最大圧縮レベル。範囲は、このアルゴリズムのデフォルトの圧縮レベルと同じです。クライアントがこれより高い圧縮レベルをリクエストした場合、サーバーはここで設定したレベルを使用します。このシステム変数を指定しない場合、サーバーは最大圧縮レベルを 8 に設定します。

- `mysqlx_max_allowed_packet`

コマンド行形式	<code>--mysqlx-max-allowed-packet=#</code>
---------	--

システム変数	mysqlx_max_allowed_packet
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	67108864
最小値	512
最大値	1073741824

X プラグイン で受信できるネットワークパケットの最大サイズ。この制限は、接続に圧縮が使用される場合にも適用されるため、メッセージが解凍された後は、ネットワークパケットをこのサイズより小さくする必要があります。これは [max_allowed_packet](#) と同等の X プラグイン です。詳細は、その変数の説明を参照してください。

- [mysqlx_max_connections](#)

コマンド行形式	<code>--mysqlx-max-connections=#</code>
システム変数	mysqlx_max_connections
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	100
最小値	1
最大値	65535

X プラグイン が受け入れることができる同時クライアント接続の最大数。これは [max_connections](#) と同等の X プラグイン です。詳細は、その変数の説明を参照してください。

この変数を変更する場合、新しい値が現在の接続数より小さいと、新しい接続に対してのみ新しい制限が考慮されます。

- [mysqlx_min_worker_threads](#)

コマンド行形式	<code>--mysqlx-min-worker-threads=#</code>
システム変数	mysqlx_min_worker_threads
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	2
最小値	1
最大値	100

クライアントリクエストを処理するために X プラグイン で使用されるワーカースレッドの最小数。

- [mysqlx_port](#)

コマンド行形式	<code>--mysqlx-port=port_num</code>
システム変数	mysqlx_port
スコープ	グローバル

動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	33060
最小値	1
最大値	65535

X プラグイン が TCP/IP 接続をリスニングするネットワークポート。これは `port` と同等の X プラグイン です。詳細は、その変数の説明を参照してください。

- [mysqlx_port_open_timeout](#)

コマンド行形式	<code>--mysqlx-port-open-timeout=#</code>
システム変数	<code>mysqlx_port_open_timeout</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	120

X プラグイン が TCP/IP ポートが解放されるまで待機する秒数。

- [mysqlx_read_timeout](#)

コマンド行形式	<code>--mysqlx-read-timeout=#</code>
システム変数	<code>mysqlx_read_timeout</code>
スコープ	セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	28800
最小値	30
最大値	2147483

X プラグイン が読取り操作のブロックの完了を待機する秒数。この時間が経過すると、読取り操作が成功しない場合、X プラグイン は接続をクローズし、エラーコード `ER_IO_READ_ERROR` の警告通知をクライアントアプリケーションに返します。

- [mysqlx_socket](#)

コマンド行形式	<code>--mysqlx-socket=file_name</code>
システム変数	<code>mysqlx_socket</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	<code>/tmp/mysqlx.sock</code>

X プラグイン が接続に使用する Unix ソケットファイルへのパス。この設定は、MySQL Server が Unix オペレーティングシステムで実行している場合のみ使用されます。クライアントはこのソケットを使用して、X プラグインを使用して MySQL Server に接続できます。

デフォルトの `mysqlx_socket` パスおよびファイル名は、MySQL Server のメインソケットファイルのデフォルトパスおよびファイル名に基づいており、ファイル名に `x` が追加されています。メインソケットファイルのデフォルトのパスとファイル名は `/tmp/mysql.sock` であるため、X プラグイン ソケットファイルのデフォルトのパスとファイル名は `/tmp/mysqlx.sock` です。

`socket` システム変数を使用して、サーバーの起動時にメインソケットファイルの代替パスとファイル名を指定した場合、これは X プラグイン ソケットファイルのデフォルトには影響しません。この場合、両方のソケットを単一のパスに格納するには、`mysqlx_socket` システム変数も設定する必要があります。たとえば、構成ファイルでは次のようになります:

```
socket=/home/sockets/mysqlid/mysql.sock
mysqlx_socket=/home/sockets/xplugin/xplugin.sock
```

コンパイル時に `MYSQLX_UNIX_ADDR` コンパイルオプションを使用してメインソケットファイルのデフォルトパスおよびファイル名を変更すると、`MYSQLX_UNIX_ADDR` ファイル名に `x` を追加することによって形成される X プラグイン ソケットファイルのデフォルトに影響します。コンパイル時に X プラグイン ソケットファイルに別のデフォルトを設定する場合は、`MYSQLX_UNIX_ADDR` コンパイルオプションを使用します。

`MYSQLX_UNIX_PORT` 環境変数を使用して、サーバーの起動時に X プラグイン ソケットファイルのデフォルトを設定することもできます (セクション4.9「環境変数」を参照)。この環境変数を設定すると、コンパイルされた `MYSQLX_UNIX_ADDR` 値はオーバーライドされますが、`mysqlx_socket` 値によってオーバーライドされます。

- `mysqlx_ssl_ca`

コマンド行形式	<code>--mysqlx-ssl-ca=file_name</code>
システム変数	<code>mysqlx_ssl_ca</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	NULL

`mysqlx_ssl_ca` システム変数は `ssl_ca` と似ていますが、MySQL Server のメイン接続インターフェースではなく X プラグイン に適用される点が異なります。X プラグイン の暗号化サポートの構成の詳細は、セクション20.5.3「X プラグイン での暗号化接続の使用」を参照してください。

- `mysqlx_ssl_capath`

コマンド行形式	<code>--mysqlx-ssl-capath=dir_name</code>
システム変数	<code>mysqlx_ssl_capath</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ディレクトリ名
デフォルト値	NULL

`mysqlx_ssl_capath` システム変数は `ssl_capath` と似ていますが、MySQL Server のメイン接続インターフェースではなく X プラグイン に適用される点が異なります。X プラグイン の暗号化サポートの構成の詳細は、セクション20.5.3「X プラグイン での暗号化接続の使用」を参照してください。

- mysqlx_ssl_cert

コマンド行形式	--mysqlx-ssl-cert=file_name
システム変数	mysqlx_ssl_cert
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	NULL

mysqlx_ssl_cert システム変数は `ssl_cert` と似ていますが、MySQL Server のメイン接続インタフェースではなく X プラグイン に適用される点が異なります。X プラグイン の暗号化サポートの構成の詳細は、[セクション20.5.3「X プラグイン での暗号化接続の使用」](#) を参照してください。

- mysqlx_ssl_cipher

コマンド行形式	--mysqlx-ssl-cipher=name
システム変数	mysqlx_ssl_cipher
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	NULL

mysqlx_ssl_cipher システム変数は `ssl_cipher` と似ていますが、MySQL Server のメイン接続インタフェースではなく X プラグイン に適用される点が異なります。X プラグイン の暗号化サポートの構成の詳細は、[セクション 20.5.3「X プラグイン での暗号化接続の使用」](#) を参照してください。

- mysqlx_ssl_crl

コマンド行形式	--mysqlx-ssl-crl=file_name
システム変数	mysqlx_ssl_crl
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	NULL

mysqlx_ssl_crl システム変数は `ssl_crl` と似ていますが、MySQL Server のメイン接続インタフェースではなく X プラグイン に適用される点が異なります。X プラグイン の暗号化サポートの構成の詳細は、[セクション20.5.3「X プラグイン での暗号化接続の使用」](#) を参照してください。

- mysqlx_ssl_crlpath

コマンド行形式	--mysqlx-ssl-crlpath=dir_name
システム変数	mysqlx_ssl_crlpath
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ディレクトリ名

デフォルト値	NULL
--------	------

`mysqlx_ssl_crlpath` システム変数は `ssl_crlpath` と似ていますが、MySQL Server のメイン接続インタフェースではなく X プラグイン に適用される点が異なります。X プラグイン の暗号化サポートの構成の詳細は、[セクション 20.5.3 「X プラグイン での暗号化接続の使用」](#) を参照してください。

- `mysqlx_ssl_key`

コマンド行形式	<code>--mysqlx-ssl-key=file_name</code>
システム変数	<code>mysqlx_ssl_key</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	ファイル名
デフォルト値	NULL

`mysqlx_ssl_key` システム変数は `ssl_key` と似ていますが、MySQL Server のメイン接続インタフェースではなく X プラグイン に適用される点が異なります。X プラグイン の暗号化サポートの構成の詳細は、[セクション 20.5.3 「X プラグイン での暗号化接続の使用」](#) を参照してください。

- `mysqlx_wait_timeout`

コマンド行形式	<code>--mysqlx-wait-timeout=#</code>
システム変数	<code>mysqlx_wait_timeout</code>
スコープ	セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	28800
最小値	1
最大値	2147483

X プラグイン が接続のアクティビティを待機する秒数。この時間が経過すると、読取り操作が成功しない場合、X プラグイン は接続をクローズします。クライアントが非対話型の場合、セッション変数の初期値はグローバル `mysqlx_wait_timeout` 変数からコピーされます。対話型クライアントの場合、初期値はセッション `mysqlx_interactive_timeout` からコピーされます。

- `mysqlx_write_timeout`

コマンド行形式	<code>--mysqlx-write-timeout=#</code>
システム変数	<code>mysqlx_write_timeout</code>
スコープ	セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	60
最小値	1
最大値	2147483

X プラグイン が書き込み操作のブロックの完了を待機する秒数。この時間が経過すると、書き込み操作が成功しない場合、X プラグイン は接続をクローズします。

- [mysqlx_zstd_default_compression_level](#)

コマンド行形式	<code>--mysqlx_zstd_default_compression_level=#</code>
導入	8.0.20
システム変数	mysqlx_zstd_default_compression_level
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	3
最小値	-131072
最大値	22

サーバーが X プロトコル 接続で zstd アルゴリズムに使用するデフォルトの圧縮レベル。1.4.0 の zstd ライブラリのバージョンでは、正の値を 1 から 22 (最高の圧縮作業)、またはプログレッシブローエフォートを表す負の値に設定できます。値 0 は値 1 に変換されます。zstd ライブラリの以前のバージョンでは、値 3 のみを指定できます。このレベルは、クライアントが機能ネゴシエーション中に圧縮レベルをリクエストしない場合に使用されます。このシステム変数を指定しない場合、サーバーはレベル 3 をデフォルトとして使用します。詳細は、[セクション 20.5.5 「X プラグイン での接続圧縮」](#)を参照してください。

- [mysqlx_zstd_max_client_compression_level](#)

コマンド行形式	<code>--mysqlx_zstd_max_client_compression_level=#</code>
導入	8.0.20
システム変数	mysqlx_zstd_max_client_compression_level
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	11
最小値	-131072
最大値	22

サーバーが X プロトコル 接続で zstd アルゴリズムに対して許可する最大圧縮レベル。範囲は、このアルゴリズムのデフォルトの圧縮レベルと同じです。クライアントがこれより高い圧縮レベルをリクエストした場合、サーバーはここで設定したレベルを使用します。このシステム変数を指定しない場合、サーバーは最大圧縮レベルを 11 に設定します。

20.5.6.3 X プラグイン ステータス変数

X プラグイン ステータス変数の意味は次のとおりです。

- [Mysqlx_aborted_clients](#)

入出力エラーのために切断されたクライアントの数。

- [Mysqlx_address](#)

X プラグイン が TCP/IP 接続を受け入れるネットワークアドレス。 [mysqlx_bind_address](#) システム変数を使用して複数のアドレスが指定された場合、 [Mysqlx_address](#) にはバインドが成功したアドレスのみが表示されます。 [mysqlx_bind_address](#) によって指定されたすべてのネットワークアドレスでバインドが失敗した場合、または [skip_networking](#) オプションが使用された場合、 [Mysqlx_address](#) の値は `UNDEFINED` になります。 X プラグイン の起動がまだ完了していない場合、 [Mysqlx_address](#) の値は空です。

- [Mysqlx_bytes_received](#)

ネットワークを介して受信された合計バイト数。接続に圧縮が使用されている場合、この図は、解凍前に測定された圧縮メッセージペイロード ([Mysqlx_bytes_received_compressed_payload](#))、圧縮されなかった圧縮メッセージ内のアイテム (X プロトコル ヘッダーなど)、および圧縮されていないメッセージで構成されます。
- [Mysqlx_bytes_received_compressed_payload](#)

圧縮メッセージペイロードとして受信したバイト数で、解凍前に測定されます。
- [Mysqlx_bytes_received_uncompressed_frame](#)

圧縮メッセージペイロードとして受信したバイト数で、解凍後に測定されます。
- [Mysqlx_bytes_sent](#)

ネットワークを介して送信された合計バイト数。接続に圧縮が使用される場合、この図は、圧縮 ([Mysqlx_bytes_sent_compressed_payload](#)) 後に測定された圧縮メッセージペイロード、圧縮されなかった圧縮メッセージ内のアイテム (X プロトコル ヘッダーなど)、および圧縮されていないメッセージで構成されます。
- [Mysqlx_bytes_sent_compressed_payload](#)

圧縮メッセージペイロードとして送信されたバイト数で、圧縮後に測定されます。
- [Mysqlx_bytes_sent_uncompressed_frame](#)

圧縮メッセージペイロードとして送信されたバイト数で、圧縮前に測定されます。
- [Mysqlx_compression_algorithm](#)

(セッションスコープ) このセッションの X プロトコル 接続に使用されている圧縮アルゴリズム。許可される圧縮アルゴリズムは、[mysqlx_compression_algorithms](#) システム変数によって一覧表示されます。
- [Mysqlx_compression_level](#)

(セッションスコープ) このセッションの X プロトコル 接続に使用される圧縮レベル。
- [Mysqlx_connection_accept_errors](#)

受け入れエラーの原因となった接続の数。
- [Mysqlx_connection_errors](#)

エラーの原因となった接続の数。
- [Mysqlx_connections_accepted](#)

受け入れられた接続の数。
- [Mysqlx_connections_closed](#)

クローズされた接続の数。
- [Mysqlx_connections_rejected](#)

拒否された接続の数。
- [Mysqlx_crud_create_view](#)

受信したビュー作成リクエストの数。
- [Mysqlx_crud_delete](#)

受信した削除リクエストの数。
- [Mysqlx_crud_drop_view](#)

受信したドロップビューリクエストの数。

- [Mysqlx_crud_find](#)

受信した検索リクエストの数。

- [Mysqlx_crud_insert](#)

受信した挿入リクエストの数。

- [Mysqlx_crud_modify_view](#)

受信した表示変更リクエストの数。

- [Mysqlx_crud_update](#)

受信した更新リクエストの数。

- [Mysqlx_cursor_close](#)

受信した cursor-close メッセージの数

- [Mysqlx_cursor_fetch](#)

受信したカーソルフェッチメッセージの数

- [Mysqlx_cursor_open](#)

受信したカーソルオープンメッセージの数

- [Mysqlx_errors_sent](#)

クライアントに送信されたエラーの数。

- [Mysqlx_expect_close](#)

クローズされた予想ブロックの数。

- [Mysqlx_expect_open](#)

オープンされた予想ブロックの数。

- [Mysqlx_init_error](#)

初期化中のエラーの数。

- [Mysqlx_messages_sent](#)

クライアントに送信されたすべてのタイプのメッセージの合計数。

- [Mysqlx_notice_global_sent](#)

クライアントに送信されたグローバル通知の数。

- [Mysqlx_notice_other_sent](#)

クライアントに返送されたその他のタイプの通知の数。

- [Mysqlx_notice_warning_sent](#)

クライアントに戻された警告通知の数。

- [Mysqlx_notified_by_group_replication](#)

クライアントに送信された Group Replication 通知の数。

- [Mysqlx_port](#)
X プラグイン がリスニングしている TCP ポート。ネットワークバインドが失敗した場合、または `skip_networking` システム変数が有効になっている場合、値は `UNDEFINED` を示します。
- [Mysqlx_prep_deallocate](#)
受信した prepared-statement-deallocate メッセージの数
- [Mysqlx_prep_execute](#)
受信したプリペアドステートメント実行メッセージの数
- [Mysqlx_prep_prepare](#)
受信したプリペアドステートメントメッセージの数
- [Mysqlx_rows_sent](#)
クライアントに返送された行数。
- [Mysqlx_sessions](#)
オープンされているセッションの数。
- [Mysqlx_sessions_accepted](#)
受け入れられたセッション試行の数。
- [Mysqlx_sessions_closed](#)
クローズされたセッションの数。
- [Mysqlx_sessions_fatal_error](#)
致命的エラーでクローズされたセッションの数。
- [Mysqlx_sessions_killed](#)
中断されたセッションの数。
- [Mysqlx_sessions_rejected](#)
拒否されたセッション試行の数。
- [Mysqlx_socket](#)
X プラグイン がリスニングしている Unix ソケット。
- [Mysqlx_ssl_accept_renegotiates](#)
接続の確立に必要なネゴシエーションの数。
- [Mysqlx_ssl_accepts](#)
受け入れられた SSL 接続の数。
- [Mysqlx_ssl_active](#)
SSL がアクティブな場合。
- [Mysqlx_ssl_cipher](#)
現在の SSL 暗号 (非 SSL 接続の場合は空)。
- [Mysqlx_ssl_cipher_list](#)

使用可能な SSL 暗号のリスト (非 SSL 接続の場合は空)。

- [Mysqlx_ssl_ctx_verify_depth](#)
現在 ctx に設定されている証明書検証の深さ制限。
- [Mysqlx_ssl_ctx_verify_mode](#)
ctx で現在設定されている証明書検証モード。
- [Mysqlx_ssl_finished_accepts](#)
サーバーへの正常な SSL 接続数。
- [Mysqlx_ssl_server_not_after](#)
SSL 証明書が有効な最終日。
- [Mysqlx_ssl_server_not_before](#)
SSL 証明書が有効な最初の日。
- [Mysqlx_ssl_verify_depth](#)
SSL 接続の証明書検証の深さ。
- [Mysqlx_ssl_verify_mode](#)
SSL 接続の証明書検証モード。
- [Mysqlx_ssl_version](#)
SSL 接続に使用されるプロトコルの名前。
- [Mysqlx_stmt_create_collection](#)
受信した create collection ステートメントの数。
- [Mysqlx_stmt_create_collection_index](#)
受信した CREATE COLLECTION INDEX ステートメントの数。
- [Mysqlx_stmt_disable_notices](#)
受信した disable notice ステートメントの数。
- [Mysqlx_stmt_drop_collection](#)
受信した drop collection ステートメントの数。
- [Mysqlx_stmt_drop_collection_index](#)
受信した drop collection index ステートメントの数。
- [Mysqlx_stmt_enable_notices](#)
受信した enable notice ステートメントの数。
- [Mysqlx_stmt_ensure_collection](#)
受信した ensure collection ステートメントの数。
- [Mysqlx_stmt_execute_mysqlx](#)
ネームスペースを `mysqlx` に設定して受信した StmtExecute メッセージの数。

- [Mysqlx_stmt_execute_sql](#)
SQL ネームスペースに対して受信された StmtExecute リクエストの数。
- [Mysqlx_stmt_execute_xplugin](#)
[xplugin](#) ネームスペースに対して受信された StmtExecute リクエストの数。MySQL 8.0.19 から、[xplugin](#) ネームスペースが削除されたため、このステータス変数は使用されなくなりました。
- [Mysqlx_stmt_get_collection_options](#)
受信した get collection object ステートメントの数。
- [Mysqlx_stmt_kill_client](#)
受信した kill クライアントステートメントの数。
- [Mysqlx_stmt_list_clients](#)
受信したリストクライアントステートメントの数。
- [Mysqlx_stmt_list_notices](#)
受信したリスト通知ステートメントの数。
- [Mysqlx_stmt_list_objects](#)
受信したリストオブジェクトステートメントの数。
- [Mysqlx_stmt_modify_collection_options](#)
受信した modify collection options ステートメントの数。
- [Mysqlx_stmt_ping](#)
受信した ping ステートメントの数。
- [Mysqlx_worker_threads](#)
使用可能なワーカースレッドの数。
- [Mysqlx_worker_threads_active](#)
現在使用中のワーカースレッドの数。

20.5.7 X プラグイン の監視

X プラグイン の一般的な監視には、公開するステータス変数を使用します。 [セクション20.5.6.3「X プラグイン ステータス変数」](#) を参照してください。 メッセージ圧縮の影響の監視の詳細は、 [X プラグイン の接続圧縮の監視](#) を参照してください。

X プラグイン によって生成された SQL の監視

このセクションでは、X DevAPI 操作の実行時に X プラグイン によって生成される SQL ステートメントを監視する方法について説明します。 CRUD ステートメントを実行すると、SQL に変換され、サーバーに対して実行されます。生成された SQL を監視できるようにするには、「パフォーマンススキーマ」テーブルを有効にする必要があります。SQL は、[performance_schema.events_statements_current](#)、[performance_schema.events_statements_history](#) および [performance_schema.events_statements_history_long](#) の各テーブルに登録されます。次の例では、このセクションのクイックスタートチュートリアルの一部としてインポートされた [world_x](#) スキーマを使用します。Python モードで MySQL Shell を使用し、SQL モードに変更せずに SQL ステートメントを発行できる `\sql` コマンドを使用します。かわりに SQL モードに切り替えると、X DevAPI 操作ではなくこの操作の結果がプロシージャに表示されるため、これは重要です。JavaScript モードで MySQL Shell を使用している場合、`\sql` コマンドは同じ方法で使用されます。

1. [events_statements_history](#) コンシューマが有効かどうかを確認します。次のコマンドを発行します:

```
mysql-py> \sql SELECT enabled FROM performance_schema.setup_consumers WHERE NAME = 'events_statements_history'
+-----+
| enabled |
+-----+
| YES    |
+-----+
```

2. すべてのインストゥルメントがコンシューマにデータをレポートするかどうかを確認します。次のコマンドを発行します:

```
mysql-py> \sql SELECT NAME, ENABLED, TIMED FROM performance_schema.setup_instruments WHERE NAME LIKE 'statement/%' AND NOT (ENABLED AND TIMED)
```

このステートメントが少なくとも 1 つの行を報告する場合は、インストゥルメントを有効にする必要があります。セクション27.4「パフォーマンススキーマ実行時構成」を参照してください。

3. 現在の接続のスレッド ID を取得します。次のコマンドを発行します:

```
mysql-py> \sql SELECT thread_id INTO @id FROM performance_schema.threads WHERE processlist_id=connection_id()
```

4. 生成された SQL を表示する X DevAPI CRUD 操作を実行します。たとえば、次のように発行します:

```
mysql-py> db.CountryInfo.find("Name = :country").bind("country", "Italy")
```

正しい結果を表示するために、次のステップでこれ以上の操作を発行しないでください。

5. このスレッド ID によって行われた最後の SQL クエリーを表示します。次のコマンドを発行します:

```
mysql-py> \sql SELECT THREAD_ID, MYSQL_ERRNO,SQL_TEXT FROM performance_schema.events_statements_history WHERE THREAD_ID=@id ORDER BY THREAD_ID, MYSQL_ERRNO,SQL_TEXT
+-----+-----+-----+-----+
| THREAD_ID | MYSQL_ERRNO | SQL_TEXT |
+-----+-----+-----+-----+
| 29 | 0 | SELECT doc FROM `world_x`.`CountryInfo` WHERE (JSON_EXTRACT(doc,'$.Name') = 'Italy') |
+-----+-----+-----+-----+
```

結果には、最新のステートメント (この場合は前のステップの X DevAPI CRUD 操作) に基づいて X プラグイン によって生成された SQL が表示されます。

第 21 章 InnoDB クラスタ

この章では、MySQL テクノロジを組み合わせて、MySQL の完全な統合高可用性ソリューションをデプロイおよび管理できる MySQL InnoDB クラスタについて説明します。このコンテンツは、InnoDB クラスタ の概要です。完全なドキュメントは、[MySQL InnoDB クラスタ](#) を参照してください。

重要

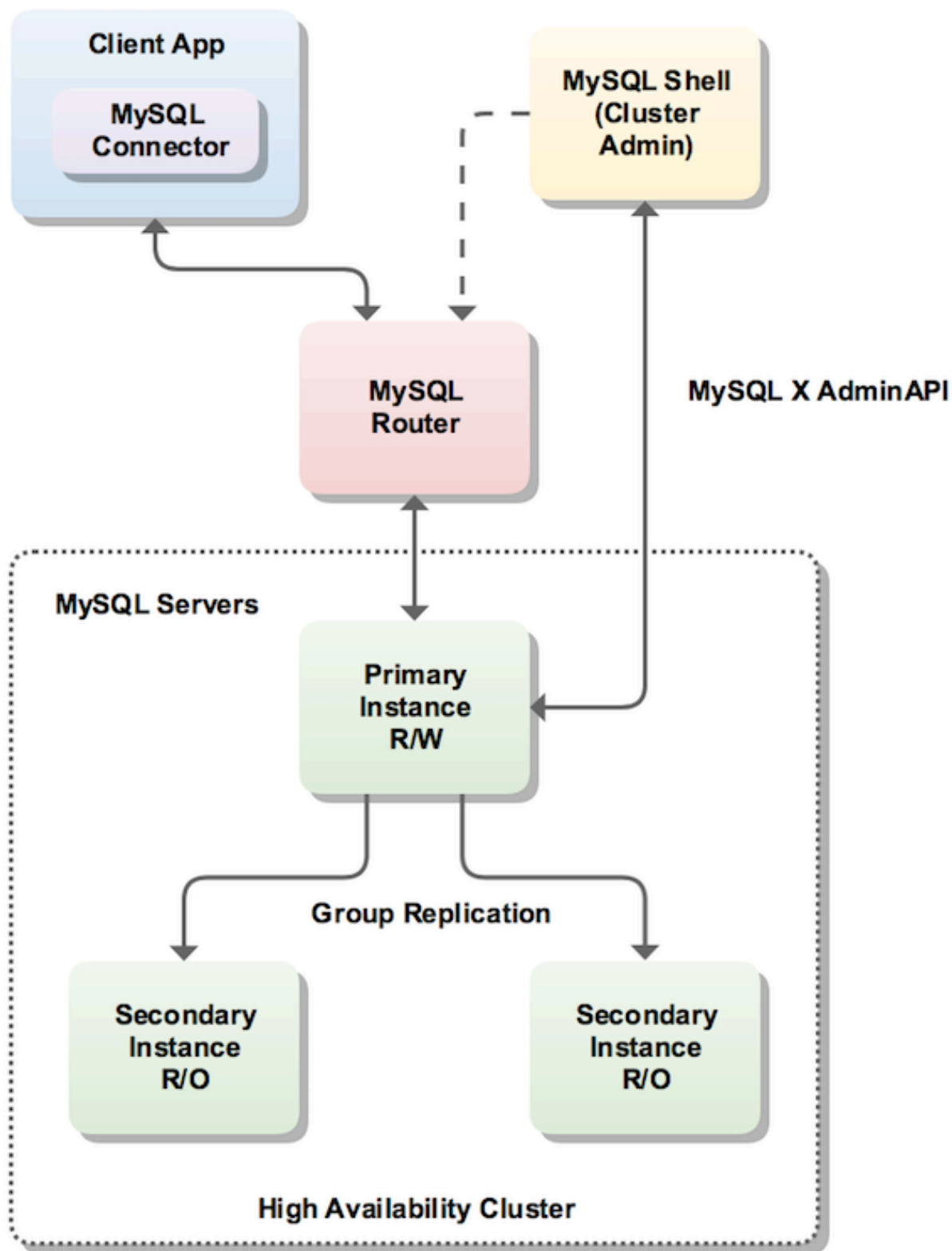
InnoDB クラスタ では、MySQL NDB Cluster はサポートされていません。MySQL NDB Cluster の詳細は、[第23章「MySQL NDB Cluster 8.0」](#) および [セクション23.1.6「MySQL Server NDB Cluster と比較した InnoDB の使用」](#) を参照してください。

InnoDB クラスタ は、少なくとも 3 つの MySQL Server インスタンスで構成され、高可用性およびスケーリング機能を提供します。InnoDB クラスタ では、次の MySQL テクノロジが使用されます:

- [MySQL Shell](#) は、MySQL の高度なクライアントおよびコードエディタです。
- MySQL Server、および [Group Replication](#) は、MySQL インスタンスのセットが高可用性を提供できるようにします。InnoDB クラスタ には、グループレプリケーションを操作するための、プログラマ的で使いやすい代替方法が用意されています。
- [MySQL Router](#):アプリケーションと InnoDB クラスタ 間の透過的なルーティングを提供する軽量ミドルウェアです。

次の図は、これらのテクノロジの連携の概要を示しています:

図 21.1 InnoDB クラスターの概要



MySQL [Group Replication](#) 上に構築され、自動メンバーシップ管理、フォルトトレランス、自動フェイルオーバーなどの機能を提供します。InnoDB クラスターは通常、単一プライマリモードで実行され、単一のプライマリインスタンス

ス (読取り/書込み) と複数のセカンダリインスタンス (読取り専用) があります。上級ユーザーは、すべてのインスタンスがプライマリである [multi-primary](#) モードを利用することもできます。InnoDB クラスタがオンラインのときにクラスタのトポロジを変更して、可用性を最大限に高めることもできます。

MySQL Shell の一部として提供されている [AdminAPI](#) を使用して、InnoDB クラスタ を操作します。AdminAPI は JavaScript および Python で使用可能で、MySQL のスクリプト作成および自動化に適しており、高可用性およびスケラビリティを実現します。MySQL ShellAdminAPI を使用すると、多数のインスタンスを手動で構成する必要がなくなります。かわりに、AdminAPI は MySQL インスタンスのセットに有効な最新のインタフェースを提供し、単一の中央ツールからデプロイメントをプロビジョニング、管理および監視できます。

InnoDB クラスタ を開始するには、MySQL Shell の[ダウンロード](#)および[インストール](#)が必要です。一部のホストには、MySQL Server インスタンスが[インストール](#)されていることが必要です。また、[install MySQL Router](#) も必要です。

InnoDB クラスタ では [MySQL Clone](#) がサポートされているため、インスタンスを簡単にプロビジョニングできます。以前は、MySQL インスタンスのセットを結合する前に新しいインスタンスをプロビジョニングするには、トランザクションを結合インスタンスに手動で転送する必要がありました。これには、ファイルコピーの作成、手動でのコピーなどが含まれる場合があります。InnoDB クラスタ を使用すると、単にクラスタに[インスタンスを追加](#)し、自動的にプロビジョニングされます。

同様に、InnoDB クラスタ は [MySQL Router](#) と緊密に統合されており、AdminAPI を使用してそれらと一緒に[作業](#)できます。MySQL Router は、[bootstrapping](#) と呼ばれるプロセスで InnoDB クラスタ に基づいて自身を自動的に構成できるため、ルーティングを手動で構成する必要はありません。MySQL Router はクライアントアプリケーションを InnoDB クラスタ に透過的に接続し、クライアント接続のルーティングおよびロードバランシングを提供します。この統合により、AdminAPI を使用して、InnoDB クラスタ に対してブートストラップされた MySQL Router の一部の側面を管理することもできます。InnoDB クラスタ のステータス情報には、クラスタに対してブートストラップされた MySQL Routers の詳細が含まれます。操作を使用すると、クラスタレベルで [create MySQL Router users](#) を実行したり、クラスタに対してブートストラップされた MySQL Routers を使用したりできます。

これらのテクノロジーの詳細は、説明にリンクされているユーザードキュメントを参照してください。このユーザードキュメントに加えて、MySQL Shell JavaScript API リファレンスまたは MySQL Shell Python API リファレンスのすべての AdminAPI メソッドの開発者ドキュメントもあり、「[コネクタおよび API](#)」から入手できます。

第 22 章 InnoDB ReplicaSet

この章では、MySQL テクノロジーを組み合わせて [第17章「レプリケーション」](#) のデプロイおよび管理を可能にする MySQL InnoDB ReplicaSet について説明します。このコンテンツは、InnoDB ReplicaSet の概要です。完全なドキュメントは、[MySQL InnoDB ReplicaSet](#) を参照してください。

InnoDB ReplicaSet は、少なくとも 2 つの MySQL Server インスタンスで構成され、読取りスケールアウトやデータセキュリティなど、よく理解しているすべての MySQL レプリケーション機能を提供します。InnoDB ReplicaSet では、次の MySQL テクノロジーが使用されます：

- [MySQL Shell](#) は、MySQL の高度なクライアントおよびコードエディタです。
- MySQL Server および [第17章「レプリケーション」](#)。これにより、MySQL インスタンスのセットで可用性および非同期読取りスケールアウトを提供できます。InnoDB ReplicaSet には、レプリケーションを操作するためのプログラマティックな代替で使いやすい方法が用意されています。
- [MySQL Router](#): アプリケーションと InnoDB クラスター間の透過的なルーティングを提供する軽量ミドルウェアです。

InnoDB ReplicaSet へのインターフェースは [MySQL InnoDB クラスター](#) に似ており、MySQL Shell を使用して MySQL Server インスタンスを ReplicaSet として操作し、MySQL Router も InnoDB クラスターと同様に緊密に統合されています。

MySQL レプリケーションに基づいている場合、InnoDB ReplicaSet には単一のプライマリがあり、複数のセカンダリインスタンスにレプリケートされます。InnoDB ReplicaSet では、自動フェイルオーバーやマルチプライマリモードなど、InnoDB クラスターが提供するすべての機能が提供されるわけではありません。ただし、インスタンスの構成、追加、削除などの機能も同様にサポートされています。たとえば、障害が発生した場合は、セカンダリインスタンスに手動でスイッチオーバーまたはフェイルオーバーできます。既存のレプリケーションデプロイメントを採用し、InnoDB ReplicaSet として管理することもできます。

MySQL Shell の一部として提供されている [AdminAPI](#) を使用して、InnoDB ReplicaSet を操作します。AdminAPI は JavaScript および Python で使用可能で、MySQL のスクリプト作成および自動化に適しており、高可用性およびスケーラビリティを実現します。MySQL Shell AdminAPI を使用すると、多数のインスタンスを手動で構成する必要がなくなります。かわりに、AdminAPI は MySQL インスタンスのセットに有効な最新のインターフェースを提供し、単一の中央ツールからデプロイメントをプロビジョニング、管理および監視できます。

InnoDB ReplicaSet を開始するには、[download](#) および [install](#) MySQL Shell が必要です。一部のホストには、MySQL Server インスタンスが [インストール](#) されていることが必要です。また、[install](#) MySQL Router も必要です。

InnoDB クラスターでは [MySQL Clone](#) がサポートされているため、インスタンスを簡単にプロビジョニングできます。以前は、MySQL Replication デプロイメントに参加する前に新しいインスタンスをプロビジョニングするには、トランザクションを結合中のインスタンスに手動で転送する必要がありました。これには、ファイルコピーの作成、手動でのコピーなどが含まれる場合があります。レプリカセットに対して [add an instance](#) を実行するだけで、自動的にプロビジョニングされます。

同様に、InnoDB ReplicaSet は [MySQL Router](#) と緊密に統合されており、AdminAPI を使用してそれらと一緒に [作業](#) できます。MySQL Router は、[bootstrapping](#) と呼ばれるプロセスで InnoDB ReplicaSet に基づいて自身を自動的に構成できるため、ルーティングを手動で構成する必要はありません。MySQL Router はクライアントアプリケーションを InnoDB クラスターに透過的に接続し、クライアント接続のルーティングおよびロードバランシングを提供します。この統合により、AdminAPI を使用して、InnoDB ReplicaSet に対してブートストラップされた MySQL Router の一部の側面を管理することもできます。InnoDB ReplicaSet のステータス情報には、クラスターに対してブートストラップされた MySQL Routers の詳細が含まれます。操作を使用すると、クラスターレベルで [create MySQL Router users](#) を実行したり、クラスターに対してブートストラップされた MySQL Routers を使用したりできます。

これらのテクノロジーの詳細は、説明にリンクされているユーザードキュメントを参照してください。このユーザードキュメントに加えて、MySQL Shell JavaScript API リファレンスまたは MySQL Shell Python API リファレンスのすべての AdminAPI メソッドの開発者ドキュメントもあり、「[コネクタおよび API](#)」から入手できます。

第 23 章 MySQL NDB Cluster 8.0

目次

23.1 NDB Cluster の概要	3448
23.1.1 NDB Cluster のコア概念	3450
23.1.2 NDB Cluster ノード、ノードグループ、フラグメントレプリカ、およびパーティション	3452
23.1.3 NDB Cluster のハードウェア、ソフトウェア、およびネットワーク要件	3455
23.1.4 NDB Cluster の新機能	3456
23.1.5 NDB 8.0 で追加、非推奨または削除されたオプション、変数、およびパラメータ	3474
23.1.6 MySQL Server NDB Cluster と比較した InnoDB の使用	3479
23.1.7 NDB Cluster の既知の制限事項	3481
23.2 NDB Cluster のインストール	3492
23.2.1 Linux での NDB Cluster のインストール	3494
23.2.2 Windows への NDB Cluster のインストール	3502
23.2.3 NDB Cluster の初期構成	3509
23.2.4 NDB Cluster の初期起動	3511
23.2.5 テーブルとデータを含む NDB Cluster の例	3512
23.2.6 NDB Cluster の安全なシャットダウンと再起動	3515
23.2.7 NDB Cluster のアップグレードおよびダウングレード	3516
23.2.8 NDB Cluster Auto-Installer (サポートされなくなりました)	3518
23.3 NDB Cluster の構成	3540
23.3.1 NDB Cluster のクイックテスト設定	3541
23.3.2 NDB Cluster 構成パラメータ、オプション、および変数の概要	3543
23.3.3 NDB Cluster 構成ファイル	3560
23.3.4 NDB Cluster での高速インターコネクトの使用	3724
23.4 NDB Cluster プログラム	3724
23.4.1 ndbd — NDB Cluster データノードデーモン	3724
23.4.2 ndbinfo_select_all — ndbinfo テーブルからの選択	3731
23.4.3 ndbmtm — NDB Cluster データノードデーモン (マルチスレッド)	3733
23.4.4 ndb_mgmd — NDB Cluster 管理サーバーデーモン	3734
23.4.5 ndb_mgm — NDB Cluster 管理クライアント	3741
23.4.6 ndb_blob_tool — NDB Cluster テーブルの BLOB および TEXT カラムのチェックおよび修復	3743
23.4.7 ndb_config — NDB Cluster 構成情報の抽出	3745
23.4.8 ndb_delete_all — NDB テーブルからのすべての行の削除	3753
23.4.9 ndb_desc — NDB テーブルの表示	3754
23.4.10 ndb_drop_index — NDB テーブルからのインデックスの削除	3760
23.4.11 ndb_drop_table — NDB テーブルの削除	3761
23.4.12 ndb_error_reporter — NDB エラーレポートユーティリティ	3761
23.4.13 ndb_import — NDB への CSV データのインポート	3763
23.4.14 ndb_index_stat — NDB インデックス統計ユーティリティ	3774
23.4.15 ndb_move_data — NDB データコピーユーティリティ	3779
23.4.16 ndb_perror — NDB エラーメッセージ情報の取得	3782
23.4.17 ndb_print_backup_file — NDB バックアップファイルの内容の出力	3783
23.4.18 ndb_print_file — NDB データファイル内容の出力	3784
23.4.19 ndb_print_frag_file — NDB フラグメントリストファイルの内容の出力	3784
23.4.20 ndb_print_schema_file — NDB スキーマファイル内容の出力	3785
23.4.21 ndb_print_sys_file — NDB システムファイル内容の出力	3785
23.4.22 ndb_redo_log_reader — クラスタ redo ログの内容の確認および印刷	3786
23.4.23 ndb_restore — NDB Cluster バックアップの復元	3788
23.4.24 ndb_select_all — NDB テーブルの行の出力	3814
23.4.25 ndb_select_count — NDB テーブルの行数の出力	3817
23.4.26 ndb_setup.py — NDB Cluster 用ブラウザベースの Auto-Installer の起動 (非推奨)	3818
23.4.27 ndb_show_tables — NDB テーブルのリストの表示	3821
23.4.28 ndb_size.pl — NDBCLUSTER サイズ要件エスティメータ	3822
23.4.29 ndb_top — NDB スレッドの CPU 使用率情報の表示	3825
23.4.30 ndb_waiter — NDB Cluster が特定のステータスに達するまで待機	3830

23.4.31 ndbxfm — NDB Cluster によって作成されたファイルの圧縮、圧縮解除、暗号化、および復号化	3832
23.4.32 NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション	3834
23.5 NDB Cluster の管理	3838
23.5.1 NDB Cluster 管理クライアントのコマンド	3839
23.5.2 NDB Cluster ログメッセージ	3843
23.5.3 NDB Cluster で生成されるイベントレポート	3858
23.5.4 NDB Cluster 起動フェーズのサマリー	3869
23.5.5 NDB Cluster のローリング再起動の実行	3871
23.5.6 NDB Cluster のシングルユーザーモード	3873
23.5.7 NDB Cluster データノードのオンラインでの追加	3874
23.5.8 NDB Cluster のオンラインバックアップ	3884
23.5.9 NDB Cluster での MySQL Server の使用	3889
23.5.10 NDB Cluster ディスクデータテーブル	3890
23.5.11 NDB Cluster での ALTER TABLE を使用したオンライン操作	3896
23.5.12 NDB_STORED_USER での分散 MySQL 権限	3899
23.5.13 NDB API 統計のカウントと変数	3900
23.5.14 ndbinfo: NDB Cluster 情報データベース	3910
23.5.15 NDB Cluster の INFORMATION_SCHEMA テーブル	3976
23.5.16 クイックリファレンス: NDB Cluster SQL ステートメント	3976
23.5.17 NDB Cluster のセキュリティーの問題	3982
23.6 NDB Cluster レプリケーション	3989
23.6.1 NDB Cluster レプリケーション: 省略形と記号	3991
23.6.2 NDB Cluster レプリケーションの一般的な要件	3991
23.6.3 NDB Cluster レプリケーションの既知の問題	3992
23.6.4 NDB Cluster レプリケーションスキーマおよびテーブル	3999
23.6.5 NDB Cluster のレプリケーションの準備	4001
23.6.6 NDB Cluster レプリケーションの開始 (シングルレプリケーションチャンネル)	4003
23.6.7 NDB Cluster レプリケーションでの 2 つのレプリケーションチャンネルの使用	4005
23.6.8 NDB Cluster レプリケーションによるフェイルオーバーの実装	4005
23.6.9 NDB Cluster レプリケーションによる NDB Cluster バックアップ	4007
23.6.10 NDB Cluster レプリケーション: 双方向および循環レプリケーション	4013
23.6.11 NDB Cluster レプリケーションの競合解決	4016
23.7 NDB Cluster リリースノート	4028

MySQL NDB Cluster は、分散コンピューティング環境に適した高可用性高冗長性バージョンの MySQL です。最新の NDB Cluster リリースシリーズでは、バージョン 8 の **NDB** ストレージエンジン (**NDBCLUSTER** と呼ばれる) を使用して、クラスタ内の MySQL サーバーやその他のソフトウェアで複数のコンピュータを実行できるようにします。NDB Cluster 8.0 は、バージョン 8.0.19 以降で General Availability (GA) リリースとして使用できるようになり、**NDB** ストレージエンジンのバージョン 8.0 が組み込まれています。NDB Cluster 7.6 および NDB Cluster 7.5 は GA リリースとして引き続き使用でき、それぞれバージョン 7.6 および 7.5 の **NDB** を使用します。以前の GA リリースは、本番、NDB Cluster 7.4 および NDB Cluster 7.3 で引き続き使用でき、それぞれ **NDB** バージョン 7.4 および 7.3 が組み込まれています。NDB 7.2 以前のリリースシリーズはサポートまたは保守されなくなりました。

NDB ストレージエンジンのサポートは、オラクルによってビルドされた標準の MySQL Server 8.0 バイナリには含まれていません。代わりに、Oracle の NDB Cluster バイナリのユーザーは、サポートされているプラットフォームの NDB Cluster の最新のバイナリリリースにアップグレードするようにしてください。これには、ほとんどの Linux ディストリビューションで動作する RPM が含まれます。NDB Cluster ソースから構築する 8.0 ユーザーは、MySQL 8.0 用に提供されているソースを使用し、NDB サポートを提供するために必要なオプションで構築するようにしてください。(ソースを入手できる場所については、このセクションで後述します。)

重要

MySQL NDB Cluster は InnoDB クラスタをサポートしていません。InnoDB クラスタは、**InnoDB** ストレージエンジンとともに MySQL Server 8.0 を使用して配備する必要があります。NDB Cluster 配布に含まれていない追加のアプリケーションを使用して配備する必要があります。MySQL Server 8.0 バイナリは、MySQL NDB Cluster では使用できません。InnoDB クラスタのデプロイおよび使用の詳細は、[MySQL AdminAPI の使用](#) を参照してください。[セクション 23.1.6 「MySQL Server NDB Cluster と比較した InnoDB の使用」](#) では、**NDB** ストレージエンジンと **InnoDB** ストレージエンジンの違いについて説明します。

この章では、8.0.23 を介した NDB Cluster 8.0 リリースについて説明します。NDB Cluster 8.0 が一般提供リリースとして使用可能になり (NDB 8.0.19 以降)、新しい配備に推奨されるようになりました。使用可能な最新リリースは NDB 8.0.22 です。NDB Cluster 7.6 および 7.5 は、本番で引き続きサポートされている以前の GA リリースです。NDB Cluster 7.6 については、[What is New in NDB Cluster 7.6](#) を参照してください。NDB Cluster 7.5 に関する同様の情報については、[What is New in NDB Cluster 7.5](#) を参照してください。NDB Cluster 7.4 および 7.3 は以前の GA リリースで本番で引き続きサポートされていますが、本番用の新しい配備では NDB Cluster 8.0 を使用することをお勧めします。[MySQL NDB Cluster 7.3 and NDB Cluster 7.4](#) を参照してください。

サポートされるプラットフォーム。NDB Cluster は現在使用可能であり、多数のプラットフォームでサポートされています。オペレーティングシステムバージョン、オペレーティングシステム配布、およびハードウェアプラットフォームの特定の組み合わせで利用可能な正確なサポートレベルについては、<https://www.mysql.com/support/supportedplatforms/cluster.html> を参照してください。

可用性。NDB Cluster バイナリおよびソースパッケージは、<https://dev.mysql.com/downloads/cluster/> からサポートされているプラットフォームで使用できます。

NDB Cluster のリリース番号。NDB 8.0 は、MySQL 8.0.13 および MySQL NDB Cluster 8.0.13 以降、MySQL Server 8.0 シリーズのリリースと同じリリースパターンに従います。この「手動」およびその他の MySQL ドキュメントでは、「NDB」で始まるバージョン番号を使用して、これら以降の NDB Cluster リリースを識別します。このバージョン番号は NDB 8.0 リリースで使用される **NDBCLUSTER** ストレージエンジンのバージョン番号であり、NDB Cluster 8.0 リリースのベースとなる MySQL 8.0 サーバーのバージョンと同じです。

NDB Cluster ソフトウェアで使用されるバージョン文字列。MySQL NDB Cluster デイストリビューションで提供される `mysql` クライアントによって表示されるバージョン文字列は、次の形式を使用します:

```
mysql-mysql_server_version-cluster
```

`mysql_server_version` は NDB Cluster リリースのベースとなる MySQL Server のバージョンを表します。NDB Cluster 8.0 のすべてのリリースでは、これは `8.0.n` です。ここで、`n` はリリース番号です。`-DWITH_NDBCLUSTER` または同等のものを使用してソースからビルドすると、`-cluster` 接尾辞がバージョン文字列に追加されます。(セクション 23.2.1.4 「Linux でのソースからの NDB Cluster の構築」およびセクション 23.2.2.2 「Windows でのソースからの NDB Cluster のコンパイルとインストール」を参照してください。) 次に示すように、`mysql` クライアントでこの書式が使用されていることがわかります。

```
shell> mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 8.0.23-cluster Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT VERSION()\G
***** 1. row *****
VERSION(): 8.0.23-cluster
1 row in set (0.00 sec)
```

MySQL 8.0 を使用した NDB Cluster の最初の General Availability リリースは、MySQL 8.0.19 を使用した NDB 8.0.19 です。

MySQL 8.0 配布に通常含まれていないほかの NDB Cluster プログラムによって表示されるバージョン文字列は、次の形式を使用します:

```
mysql-mysql_server_version ndb-ndb_engine_version
```

`mysql_server_version` は NDB Cluster リリースのベースとなる MySQL Server のバージョンを表します。NDB Cluster 8.0 のすべてのリリースでは、これは `8.0.n` です。ここで、`n` はリリース番号です。`ndb_engine_version` は、NDB Cluster ソフトウェアのこのリリースで使用される **NDB** ストレージエンジンのバージョンです。すべての NDB 8.0 リリースで、この番号は MySQL Server バージョンと同じです。この形式は、次のように `ndb_mgm` クライアントの `SHOW` コマンドの出力で使用されます:

```
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
-----
[ndbd(NDB)] 2 node(s)
id=1 @10.0.10.6 (mysql-8.0.29 ndb-8.0.23, Nodegroup: 0, *)
```



```
id=2 @10.0.10.8 (mysql-8.0.29 ndb-8.0.23, Nodegroup: 0)
```

```
[ndb_mgmd(MGM)] 1 node(s)
```

```
id=3 @10.0.10.2 (mysql-8.0.29 ndb-8.0.23)
```

```
[mysqld(API)] 2 node(s)
```

```
id=4 @10.0.10.10 (mysql-8.0.29 ndb-8.0.23)
```

```
id=5 (not connected, accepting connect from any host)
```

標準の MySQL 8.0 リリースとの互換性。多くの標準 MySQL スキーマおよびアプリケーションは NDB Cluster を使用して動作できますが、NDB Cluster を使用して実行すると、変更されていないアプリケーションおよびデータベーススキーマにわずかな互換性がないか、パフォーマンスが最適でない可能性があります ([セクション23.1.7「NDB Cluster の既知の制限事項」](#) を参照)。これらの問題のほとんどは克服できますが、これは、スキーマ、クエリー、およびアプリケーションに変更を加える可能性を考慮せずに、既存の (たとえば、[MyISAM](#) や [InnoDB](#) を使用する) アプリケーションデータストアを NDB ストレージエンジンを使用するものに切り替えることがほとんど不可能であることも意味します。NDB サポートなしでコンパイルされた (つまり、`-DWITH_NDBCLUSTER_STORAGE_ENGINE` またはそのエイリアス `-DWITH_NDBCLUSTER` なしで構築された) `mysqld` は、それを使用して構築された `mysqld` のドロップイン置換として機能できません。

NDB Cluster 開発ソースツリー。NDB Cluster 開発ツリーには、<https://github.com/mysql/mysql-server> からアクセスできます。

<https://github.com/mysql/mysql-server> で保守される NDB Cluster 開発ソースは GPL のもとでライセンス供与されます。Git を使用して MySQL ソースを取得し、自分で構築する方法の詳細は、[セクション2.9.5「開発ソースツリーを使用して MySQL をインストールする」](#) を参照してください。

注記

MySQL Server 8.0 と同様に、NDB Cluster 8.0 リリースは `CMake` を使用して構築されます。

NDB Cluster 8.0 は NDB 8.0.19 以降で一般提供リリースとして使用可能であり、新しい配備に推奨されます。NDB Cluster 7.6 および 7.5 は、本番で引き続きサポートされている以前の GA リリースです。NDB Cluster 7.6 については、[What is New in NDB Cluster 7.6](#) を参照してください。NDB Cluster 7.5 に関する同様の情報については、[What is New in NDB Cluster 7.5](#) を参照してください。NDB Cluster 7.4 および 7.3 は以前の GA リリースで本番で引き続きサポートされていますが、本番用の新しい配備では NDB Cluster 8.0 を使用することをお勧めします。[MySQL NDB Cluster 7.3 and NDB Cluster 7.4](#) を参照してください。

NDB Cluster が進化し続けるため、この章の内容は改訂の対象となります。NDB Cluster に関する追加情報は、<http://www.mysql.com/products/cluster/> の MySQL web サイトにあります。

追加のリソース。NDB Cluster の詳細は、次の場所を参照してください:

- NDB Cluster に関してよく寄せられる質問への回答については、[セクションA.10「MySQL 8.0 FAQ: NDB Cluster」](#) を参照してください。
- NDB Cluster フォーラム: <https://forums.mysql.com/list.php?25>。
- NDB Cluster に関する多くの NDB Cluster ユーザーおよび開発者のブログで NDB Cluster に関する経験が得られ、[PlanetMySQL](#) を介してこれらのフィードを利用できるようになります。

23.1 NDB Cluster の概要

NDB Cluster は、共有なしシステムでインメモリーデータベースのクラスタリングを可能にするテクノロジーです。シェアードナッシングアーキテクチャーでは、非常に安価なハードウェアでシステムが動作し、ハードウェアやソフトウェアの要件が最小限に抑えられます。

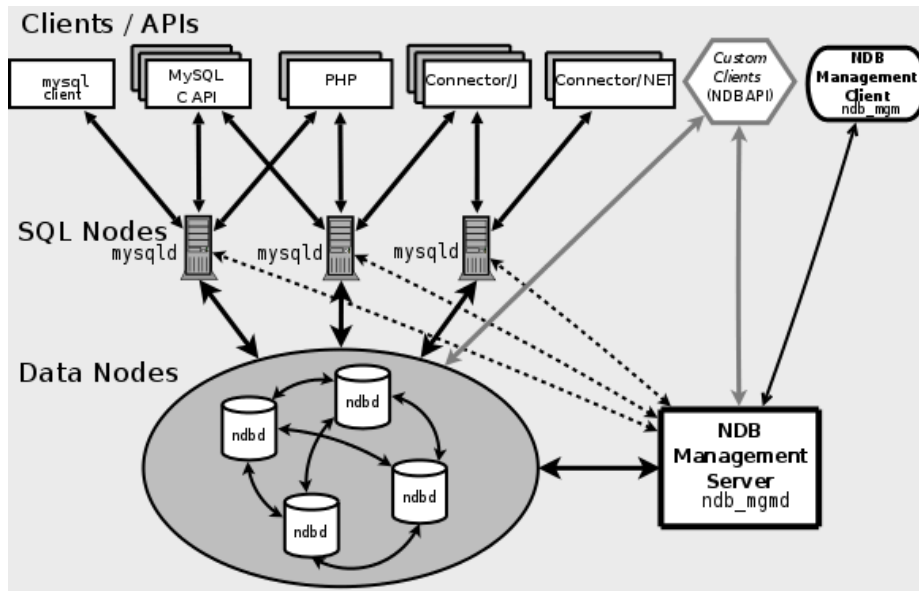
NDB Cluster は、単一点障害を持たないように設計されています。シェアードナッシングシステムでは、各コンポーネントに固有のメモリーとディスクが用意され、ネットワーク共有、ネットワークファイルシステム、SAN などの共有ストレージメカニズムの使用は推奨またはサポートされません。

NDB Cluster は、標準の MySQL サーバーを NDB (「N etwork D ata B ase」を表す) と呼ばれるメモリー内クラスタ化ストレージエンジンと統合します。このドキュメントでは、NDB という用語はストレージエンジンに固有の設定部

分を指しますが、「MySQL NDB Cluster」とは、1つ以上の MySQL サーバーと NDB ストレージエンジンの組み合わせを指します。

NDB Cluster は、ホストと呼ばれる一連のコンピュータで構成され、それぞれが 1つ以上のプロセスを実行しています。ノードと呼ばれるこれらのプロセスには、MySQL Server (NDB データへのアクセス用)、データノード (データのストレージ用)、1つ以上の管理サーバー、および場合によってはその他の特殊なデータアクセスプログラムが含まれます。NDB Cluster 内のこれらのコンポーネントの関係を次に示します:

図 23.1 NDB Cluster コンポーネント



これらのプログラムはすべて連携して NDB Cluster を形成します (セクション23.4「NDB Cluster プログラム」を参照)。NDB ストレージエンジンによってデータが格納されると、データノードにテーブル (およびテーブルデータ) が格納されます。このようなテーブルには、クラスタ内のほかのすべての MySQL Server (SQL ノード) から直接アクセスできます。したがって、クラスタにデータを格納する給料計算アプリケーションでは、あるアプリケーションが社員の給料を更新すると、このデータをクエリーするほかのすべての MySQL サーバーがこの変更をただちに認識できます。

NDB Cluster SQL ノードは `mysqld` サーバーデーモンを使用しますが、MySQL 8.0 ディストリビューションで提供される `mysqld` バイナリと多くの重要な点で異なり、2つのバージョンの `mysqld` は交換できません。

また、NDB Cluster に接続されていない MySQL サーバーは、NDB ストレージエンジンを使用できず、NDB Cluster データにアクセスできません。

NDB Cluster のデータノードに格納されているデータはミラー化できます。クラスタは、トランザクションの状態が失われたために少数のトランザクションが中止されること以外に影響を与えることなく、個々のデータノードの障害を処理できます。トランザクションの失敗はトランザクションアプリケーションが処理すると想定されるため、これが問題の原因になることはないはずです。

個々のノードは、停止して再起動したあと、システム (クラスタ) に再度参加できます。構成の変更やソフトウェアのアップグレードを行うときは、ローリング再起動 (すべてのノードが順に再起動される) が使用されます (セクション23.5.5「NDB Cluster のローリング再起動の実行」を参照してください)。ローリング再起動は、新しいデータノードをオンラインで追加するプロセスの一部としても使用されます (セクション23.5.7「NDB Cluster データノードのオンラインでの追加」を参照してください)。データノード、NDB Cluster 内での編成方法、および NDB Cluster データの処理方法と格納方法の詳細は、セクション23.1.2「NDB Cluster ノード、ノードグループ、フラグメントレプリカ、およびパーティション」を参照してください。

NDB Cluster データベースのバックアップと復元は、NDB Cluster 管理クライアントにある NDB ネイティブ機能と NDB Cluster 配布に含まれる `ndb_restore` プログラムを使用して実行できます。詳細は、セクション23.5.8「NDB Cluster のオンラインバックアップ」およびセクション23.4.23「`ndb_restore` — NDB Cluster バックアップの復元」を

参照してください。 [mysqldump](#) および MySQL Server でこの目的のために用意されている標準の MySQL 機能を使用することもできます。詳細は、[セクション4.5.4「mysqldump — データベースバックアッププログラム」](#)を参照してください。

NDB Cluster ノードは、ノード間通信にさまざまなトランスポートメカニズムを採用できます。ほとんどの実際の配備では、標準 100 Mbps 以上の TCP/IP または高速 Ethernet ハードウェアが使用されます。

23.1.1 NDB Cluster のコア概念

NDBCLUSTER (NDB と呼ばれる) は、高可用性とデータ永続性の機能を備えたインメモリーストレージエンジンです。

NDBCLUSTER ストレージエンジンはさまざまなフェイルオーバーとロードバランシングのオプションを使って構成できますが、クラスタレベルのストレージエンジンから始めるのがもっとも簡単です。NDB Cluster **NDB** ストレージエンジンには、クラスタ自体内のほかのデータにのみ依存する完全なデータセットが含まれています。

NDB Cluster の「クラスタ」部分は、MySQL サーバーから独立して構成されます。NDB Cluster では、クラスタの各部分はノードとみなされます。

注記

多くのコンテキストでは、「ノード」という用語はコンピュータを示すために使用されますが、NDB Cluster について説明するときはプロセスを意味します。1 台のコンピュータで複数のノードを実行できます。1 つ以上のクラスタノードを実行しているコンピュータに対しては、クラスタホストという用語を使用します。

クラスタノードには 3 つのタイプがあり、NDB Cluster の最小構成では、少なくとも 3 つのノードがあり、これらの各タイプのいずれかです:

- 管理ノード: このタイプのノードの役割は、NDB Cluster 内のほかのノードを管理し、構成データの提供、ノードの起動と停止、バックアップの実行などの機能を実行することです。このノードタイプはほかのノードの構成を管理するため、このタイプのノードは最初に (ほかのノードより先に) 起動するようにしてください。管理ノードは、コマンド `ndb_mgmd` を使用して起動されます。

- データノード: このタイプのノードにはクラスタのデータが格納されます。フラグメントレプリカの数にフラグメントの数を掛けた数と同じ数のデータノードがあります ([セクション23.1.2「NDB Cluster ノード、ノードグループ、フラグメントレプリカ、およびパーティション」](#)を参照)。たとえば、2 つのフラグメントレプリカがあり、それぞれに 2 つのフラグメントがある場合、4 つのデータノードが必要です。1 つのフラグメントレプリカではデータストレージに十分ですが、冗長性は提供されません。したがって、冗長性と高可用性を実現するために、2 つ以上のフラグメントレプリカを使用することをお勧めします。データノードは、`ndbd` ([セクション23.4.1「ndbd — NDB Cluster データノードデーモン」](#)を参照してください) または `ndbmt` ([セクション23.4.3「ndbmt — NDB Cluster データノードデーモン \(マルチスレッド\)」](#)を参照してください) コマンドで起動します。

「NDB Cluster」テーブルは通常、ディスクではなくメモリーに完全に格納されます (このため、「NDB Cluster」を in-memory データベースと呼びます)。ただし、一部の NDB Cluster データはディスクに格納できます。詳細は、[セクション23.5.10「NDB Cluster ディスクデータテーブル」](#)を参照してください。

- SQL ノード: これは、クラスタデータにアクセスするノードです。NDB Cluster の場合、SQL ノードは **NDBCLUSTER** ストレージエンジンを使用する従来の MySQL サーバーです。SQL ノードは、`--ndbcluster` および `--ndb-connectstring` オプション (これらについては、この章の別の場所で説明します) を指定して起動される `mysqld` プロセスです。場合によっては、追加の MySQL Server オプションも指定できます。

SQL ノードは、NDB Cluster データにアクセスするすべてのアプリケーションを指定する特殊なタイプの API ノードです。API ノードのもう 1 つの例は、クラスタのバックアップをリストアするために使われる `ndb_restore` コーティリティーです。NDB API を使用してこのようなアプリケーションを作成できます。NDB API の基本情報については、[Getting Started with the NDB API](#)を参照してください。

重要

本番環境で 3 ノードセットアップの採用を期待するのは現実的ではありません。このような構成では冗長性は提供されません。NDB Cluster の高可用性機能を利用するには、複数

のデータおよび SQL ノードを使用する必要があります。複数の管理ノードを使用することも、強くお勧めします。

NDB Cluster 内のノード、ノードグループ、フラグメントレプリカ、およびパーティション間の関係の簡単な概要については、[セクション23.1.2「NDB Cluster ノード、ノードグループ、フラグメントレプリカ、およびパーティション」](#)を参照してください。

クラスタの構成には、クラスタ内の各ノードの構成と、ノード間の個々の通信リンクの設定が含まれます。NDB Cluster は現在、データノードがプロセッサの電力、メモリー領域、および帯域幅に関して同種であることを意図して設計されています。さらに、一元管理の構成を提供するため、クラスタのすべての構成データが全体として1つの構成ファイルに格納されます。

管理サーバーは、クラスタ構成ファイルとクラスタログを管理します。クラスタ内の各ノードは、管理サーバーから構成データを取得するため、管理サーバーがどこにあるかを特定する手段を必要とします。データノードで注目するイベントが発生すると、そのノードはこれらのイベントに関する情報を管理サーバーに転送し、管理サーバーはその情報をクラスタログに書き込みます。

さらに、任意の数のクラスタクライアントプロセスまたはアプリケーションが存在する可能性があります。これらには、標準の MySQL クライアント、NDB 専用の API プログラム、管理クライアントなどが含まれます。次のいくつかの段落で、これらについて説明します。

標準の MySQL クライアント。 NDB Cluster は、PHP、Perl、C、C++、Java、Python、Ruby などで記述された既存の MySQL アプリケーションで使用できます。このようなクライアントアプリケーションは、NDB Cluster SQL ノードとして機能する MySQL サーバーとの間で、スタンドアロン MySQL サーバーとの対話とほぼ同じ方法で SQL ステートメントを送受信します。

NDB Cluster をデータソースとして使用する MySQL クライアントは、複数の MySQL サーバーに接続して負荷分散とフェイルオーバーを実現する機能を利用するように変更できます。たとえば、Connector/J 5.0.6 以降を使用する Java クライアントは、`jdbc:mysql:loadbalance://` URL (Connector/J 5.1.7 で改善された) を使用して、負荷分散を透過的に実現できます。NDB Cluster で Connector/J を使用方法の詳細は、[Using Connector/J with NDB Cluster](#) を参照してください。

NDB クライアントプログラム。 高レベルの C++ API である NDB API を使用して、クラスタに接続されている MySQL Servers をバイパスして、`NDBCLUSTER` ストレージエンジンから NDB Cluster データに直接アクセスするクライアントプログラムを作成できます。このようなアプリケーションは、データへの SQL インタフェースを必要としない特殊な目的に役立ちます。詳細は、[The NDB API](#)を参照してください。

NDB 固有の Java アプリケーションは、NDB Cluster Connector for Java を使用して NDB Cluster 用に記述することもできます。この NDB Cluster コネクタには、`NDBCLUSTER` に直接接続する Hibernate や JPA などのオブジェクトリレーショナルマッピング永続性フレームワークに似た高レベルのデータベース API である ClusterJ が含まれているため、MySQL Server にアクセスする必要はありません。詳細は、[Java and NDB Cluster](#)および[The ClusterJ API and Data Object Model](#)を参照してください。

NDB Cluster は、Node.js を使用して JavaScript で記述されたアプリケーションもサポートします。JavaScript 用の MySQL Connector には、MySQL Server だけでなく、`NDB` ストレージエンジンに直接アクセスするためのアダプタも含まれています。この Connector を使用するアプリケーションは、通常イベント駆動型であり、ClusterJ に採用されているものと多くの点で似ているドメインオブジェクトモデルを使用します。詳細は、[MySQL NoSQL Connector for JavaScript](#)を参照してください。

管理クライアント。 これらのクライアントは、管理サーバーに接続して、ノードの正常な起動と停止、メッセージトレースの開始と停止 (デバッグバージョンのみ)、ノードのバージョンとステータスの表示、バックアップの開始と停止などのコマンドを提供します。このタイプのプログラムの例としては、NDB Cluster に付属する `ndb_mgm` 管理クライアントがあります ([セクション23.4.5「ndb_mgm — NDB Cluster 管理クライアント」](#)を参照)。このようなアプリケーションは、NDB Cluster 管理サーバーと直接通信する C 言語 API である MGM API を使用して記述できます。詳細は、[The MGM API](#)を参照してください。

Oracle は、多数のノードで NDB Cluster を再起動するなど、多くの複雑な NDB Cluster 管理タスクを簡略化する高度なコマンド行インタフェースを提供する MySQL Cluster Manager も使用可能にします。MySQL Cluster Manager クライアントは、ほとんどのノード構成パラメータの値、NDB Cluster に関連する `mysqld` サーバーオプションおよび変数を取得および設定するためのコマンドもサポートしています。MySQL Cluster Manager 1.4.8 は NDB 8.0 の実験的なサポートを提供します。詳しくは[MySQL Cluster Manager 1.4.8 User Manual](#),をご覧ください。

イベントログ。 NDB Cluster は、イベントをカテゴリ (起動、シャットダウン、エラー、チェックポイントなど)、優先順位、および重大度別にログに記録します。すべてのレポート可能イベントの完全なリストについては、[セクション 23.5.3 「NDB Cluster で生成されるイベントレポート」](#) を参照してください。 イベントログには、ここに示す 2 つのタイプがあります。

- クラスタログ: クラスタに関する必要なすべてのレポート可能イベントが全体として保持されます。
- ノードログ: 個々のノードごとに保持される個別のログです。

注記

通常の場合では、クラスタログのみを保持して調べるだけで必要十分です。 ノードログを調べる必要があるのは、アプリケーション開発やデバッグの場合だけです。

Checkpoint。 一般的には、データをディスクに保存したときにチェックポイントに達したといいます。 NDB Cluster に固有のチェックポイントは、コミットされたすべてのトランザクションがディスクに格納される時点です。 NDB ストレージエンジンには、クラスタデータの一貫したビューが維持されるように連携する 2 つのタイプのチェックポイントがあります。 次のリストにそれらを示します。

- ローカルチェックポイント (LCP): これは、1 つのノードに固有のチェックポイントです。 ただし、LCP はクラスタ内のすべてのノードである程度同時に発生します。 LCP は通常、数分ごとに発生します。 正確な間隔は、ノードに格納されるデータ量、クラスタアクティビティのレベルおよびその他の要因によって異なります。

NDB 8.0 は部分 LCP をサポートしており、状況によってはパフォーマンスを大幅に向上させることができます。 部分 LCP を有効にし、それらが使用する記憶域の量を制御する [EnablePartialLcp](#) および [RecoveryWork](#) の構成パラメータの説明を参照してください。

- グローバルチェックポイント (GCP): GCP は、すべてのノードのトランザクションが同期し、Redo ログがディスクにフラッシュされたときに、数秒間隔で発生します。

ローカルチェックポイントとグローバルチェックポイントによって作成されるファイルおよびディレクトリの詳細は、[NDB Cluster Data Node File System Directory](#) を参照してください。

23.1.2 NDB Cluster ノード、ノードグループ、フラグメントレプリカ、およびパーティション

このセクションでは、NDB Cluster がストレージのデータを分割および複製する方法について説明します。

次の各段落では、このトピックを理解する上で中心となるいくつかの概念について説明します。

データノード。 `ndbd` または `ndbmtid` プロセス。 ノードがメンバーであるノードグループに割り当てられたフラグメントレプリカ、つまりパーティションのコピーを格納します。

各データノードは、別個のコンピュータに配置するようにしてください。 単一のコンピュータ上で複数のデータノードプロセスをホストすることもできますが、このような構成は通常はお勧めしません。

「ノード」および「データノード」という用語は、`ndbd` または `ndbmtid` プロセスを参照するときに同じ意味で使用されます。 ここで言及しているように、管理ノード (`ndb_mgmd` プロセス) および SQL ノード (`mysqld` プロセス) はこの説明で指定されています。

ノードグループ。 ノードグループは、1 つ以上のノードで構成され、パーティションまたはフラグメントレプリカのセットを格納します (次の項目を参照)。

NDB Cluster 内のノードグループの数は直接構成できません。 次に示すように、データノードの数とフラグメントレプリカの数 (`NoOfReplicas` 構成パラメータ) の関数です:

```
[# of node groups] = [# of data nodes] / NoOfReplicas
```

したがって、4 つのデータノードを持つ NDB Cluster には、`config.ini` ファイルで `NoOfReplicas` が 1 に設定されている場合は 4 つのノードグループ、`NoOfReplicas` が 2 に設定されている場合は 2 つのノードグループ、`NoOfReplicas` が 4 に設定されている場合は 1 つのノードグループがあります。 フラグメントレプリカについては、このセクション

の後半で説明します。NoOfReplicas の詳細は、[セクション23.3.3.6「NDB Cluster データノードの定義」](#)を参照してください。

注記

NDB Cluster 内のすべてのノードグループは、同じ数のデータノードを持つ必要があります。

実行中の NDB Cluster に新しいノードグループ (およびその結果として新しいデータノード) をオンラインで追加できます。詳細は、[セクション23.5.7「NDB Cluster データノードのオンラインでの追加」](#)を参照してください。

パーティション。これは、クラスタに格納されるデータの一部分です。各ノードは、そのノードに割り当てられているパーティション (少なくとも 1 つのフラグメントレプリカ) の少なくとも 1 つのコピーをクラスタで使用できるようにします。

NDB Cluster によってデフォルトで使用されるパーティションの数は、次に示すように、データノードの数およびデータノードで使用されている LDM スレッドの数によって異なります:

```
[# of partitions] = [# of data nodes] * [# of LDM threads]
```

ndbmttd を実行しているデータノードを使用する場合、LDM スレッドの数は MaxNoOfExecutionThreads の設定によって制御されます。ndbd を使用する場合、単一の LDM スレッドが存在します。つまり、クラスタに参加しているノードと同じ数のクラスタパーティションが存在します。これは、MaxNoOfExecutionThreads を 3 以下に設定して ndbmttd を使用する場合にも当てはまります。(LDM スレッドの数はこのパラメータの値とともに増加しますが、厳密に線形な方法では増加せず、設定に追加の制約があることに注意してください。詳細は、MaxNoOfExecutionThreads の説明を参照してください。)

NDB とユーザー定義のパーティション化。NDB Cluster は通常、NDBCLUSTER テーブルを自動的にパーティション化します。ただし、NDBCLUSTER テーブルにユーザー定義のパーティション化を適用することもできます。これには、次の制限があります。

1. NDB テーブルを使用した本番では、KEY および LINEAR KEY パーティション化スキームのみがサポートされません。
2. 任意の NDB テーブルに対して明示的に定義できるパーティションの最大数は、このセクションで前述したように決定される NDB Cluster 内のノードグループの数である $8 * [\text{number of LDM threads}] * [\text{number of node groups}]$ です。データノードプロセスに対して ndbd を実行している場合、LDM スレッドの数を設定しても効果はありません (ThreadConfig は ndbmttd にのみ適用されるため)。このような場合、この計算を実行するために、この値は 1 と同等であるかのように処理できます。

詳細は、[セクション23.4.3「ndbmttd — NDB Cluster データノードデーモン \(マルチスレッド\)」](#)を参照してください。

NDB Cluster およびユーザー定義パーティショニングに関する詳細は、[セクション23.1.7「NDB Cluster の既知の制限事項」](#)、および [セクション24.6.2「ストレージエンジンに関連するパーティショニング制限」](#)を参照してください。

フラグメントのレプリカ。これは、クラスタパーティションのコピーです。ノードグループ内の各ノードには、フラグメントレプリカが格納されます。パーティションレプリカと呼ばれることもあります。フラグメントレプリカの数は、ノードグループ当たりのノード数と同じです。

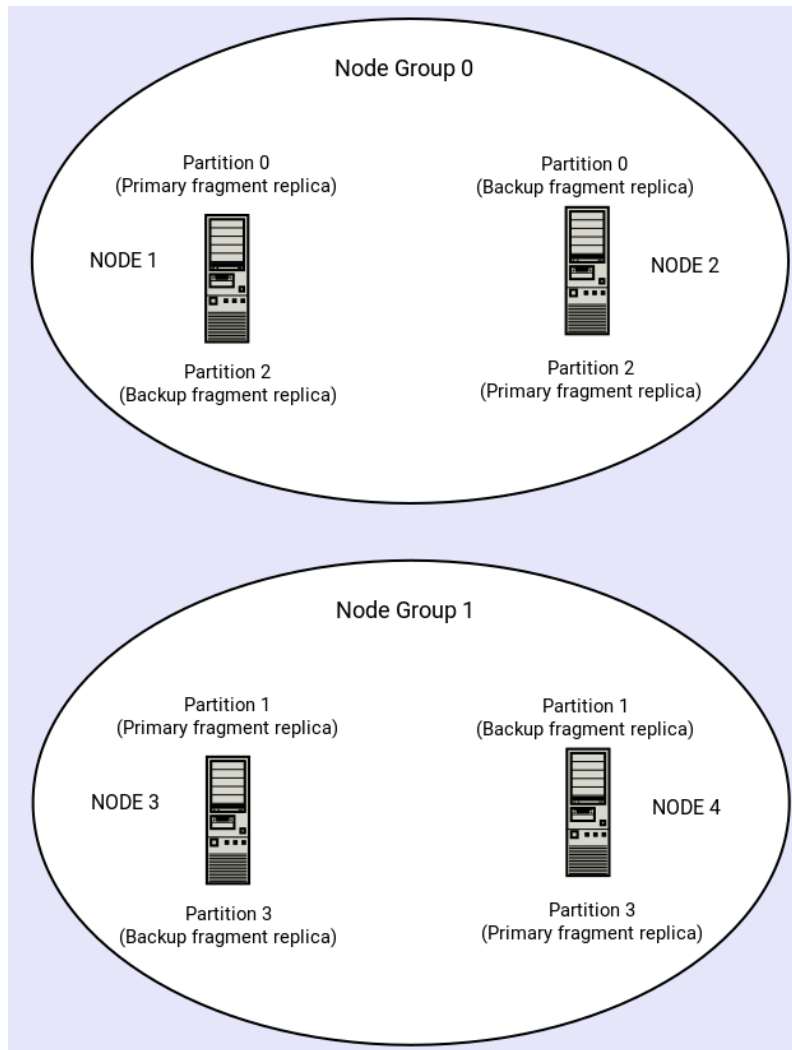
フラグメントレプリカは、完全に単一のノードに属します。ノードには、複数のフラグメントレプリカを格納できません (通常は格納できます)。

次の図は、ndbd を実行している 4 つのデータノードがあり、それぞれ 2 つのノードの 2 つのノードグループに配置されている NDB Cluster を示しています。ノード 1 と 2 はノードグループ 0 に属し、ノード 3 と 4 はノードグループ 1 に属しています。

注記

ここにはデータノードのみが表示されます。動作中の NDB Cluster では、クラスタ管理のために ndb_mgmd プロセスが必要ですが、クラスタによって格納されているデータにアクセスするために少なくとも 1 つの SQL ノードが必要ですが、わかりやすくするため、これは図から省略されています。

図 23.2 2 つのノードグループを持つ NDB Cluster

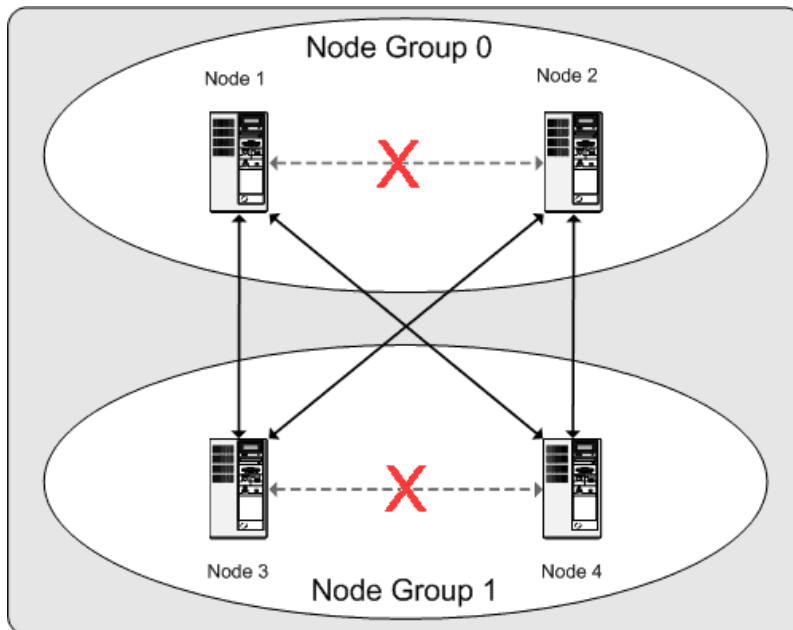


クラスタに格納されたデータは、0、1、2、3 の番号が付けられた 4 つのパーティションに分割されています。各パーティションは同じノードグループに (複数のコピーで) 格納されます。パーティションは、次のように各ノードグループに交互に格納されます。

- パーティション 0 はノードグループ 0 に格納され、プライマリフラグメントレプリカ (プライマリコピー) はノード 1 に格納され、バックアップフラグメントのレプリカ (パーティションのバックアップコピー) はノード 2 に格納されます。
- パーティション 1 は他のノードグループ (ノードグループ 1) に格納され、このパーティションのプライマリフラグメントレプリカはノード 3 にあり、そのバックアップフラグメントレプリカはノード 4 にあります。
- パーティション 2 はノードグループ 0 に格納されます。ただし、その 2 つのフラグメントレプリカの配置は、パーティション 0 の配置から逆になります。パーティション 2 の場合、プライマリフラグメントレプリカはノード 2 に格納され、バックアップはノード 1 に格納されます。
- パーティション 3 はノードグループ 1 に格納され、その 2 つのフラグメントレプリカの配置はパーティション 1 の配置から逆になります。つまり、プライマリフラグメントレプリカはノード 4 にあり、バックアップはノード 3 にあります。

NDB Cluster の継続的な操作に関する意味は次のとおりです: クラスタに参加している各ノードグループのノードが少なくとも 1 つ動作しているかぎり、クラスタにはすべてのデータの完全なコピーがあり、実行可能なままです。これを次の図に示します。

図 23.3 2x2 NDB Cluster に必要なノード



この例では、クラスタはそれぞれ 2 つのデータノードで構成される 2 つのノードグループで構成されています。各データノードは、`ndbd` のインスタンスを実行しています。クラスタ「アライブ」を保持するには、ノードグループ 0 の少なくとも 1 つのノードとノードグループ 1 の少なくとも 1 つのノードの組み合わせで十分です。ただし、単一ノードグループの両方のノードで障害が発生した場合、他のノードグループの残りの 2 つのノードで構成される組み合わせでは不十分です。この状況では、クラスタはパーティション全体を失ったため、すべての NDB Cluster データの完全なセットにアクセスできなくなります。

単一の NDB Cluster インスタンスでサポートされるノードグループの最大数は 48 です。

23.1.3 NDB Cluster のハードウェア、ソフトウェア、およびネットワーク要件

NDB Cluster の強みの 1 つは、すべてのライブデータストレージがメモリー内で実行されるため、コモディティハードウェア上で実行でき、大量の RAM 以外はこの点で異常な要件がないことです。(ディスクデータテーブルを使用してこの要件を削減することもできます。詳細は、[セクション 23.5.10 「NDB Cluster ディスクデータテーブル」](#) を参照してください。) 当然ながら、複数の高速な CPU を使用すればパフォーマンスは向上します。ほかの NDB Cluster プロセスのメモリー要件は比較的小さくなります。

NDB Cluster のソフトウェア要件も最新です。ホストオペレーティングシステムで NDB Cluster をサポートするために、異常なモジュール、サービス、アプリケーション、または構成は必要ありません。サポートされるオペレーティングシステムについては、標準のインストールで十分のはずです。MySQL のソフトウェア要件は単純です: 必要なのは NDB Cluster の本番リリースだけです。NDB Cluster を使用できるようにするためにのみ、MySQL を自分でコンパイルする必要はありません。使用しているプラットフォームに適したバイナリを使用していることを前提としています。このバイナリは、<https://dev.mysql.com/downloads/cluster/> の NDB Cluster ソフトウェアのダウンロードページから入手できます。

ノード間の通信の場合、NDB Cluster は任意の標準トポロジで TCP/IP ネットワークをサポートし、各ホストに必要な最小値は、標準の 100 Mbps Ethernet カードと、クラスタ全体にネットワーク接続を提供するためのスイッチ、ハブ、またはルーターです。NDB Cluster は、次の理由により、クラスタの一部を形成していないマシンと共有されていない独自のサブネット上で実行することを強くお勧めします:

- **セキュリティ.** NDB Cluster ノード間の通信は暗号化またはシールドされません。NDB Cluster 内の転送を保護する唯一の方法は、保護されたネットワーク上で NDB Cluster を実行することです。NDB Cluster を Web アプリケーションに使用する場合、クラスタはネットワークの非武装地帯 (DMZ) またはほかの場所ではなく、ファイアウォールの内側に確実に配置するようにしてください。

詳細は、[セクション 23.5.17.1 「NDB Cluster のセキュリティおよびネットワークの問題」](#) を参照してください。

- 効率. プライベートネットワークまたは保護されたネットワークで NDB Cluster を設定すると、クラスタはクラスタホスト間の帯域幅を排他的に使用できます。NDB Cluster に別のスイッチを使用すると、NDB Cluster データへの不正アクセスから保護できるだけでなく、NDB Cluster ノードがネットワーク上のほかのコンピュータ間の転送によって発生する干渉から遮断されることも保証されます。信頼性を高めるため、デュアルスイッチとデュアルカードを使用して、単一障害点になったネットワークを除去できます。多くのデバイスドライバがこのような通信リンクのフェイルオーバーをサポートしています。

ネットワーク通信と待機時間. NDB Cluster でクエリーおよび更新を実行するには、データノードと API ノード (SQL ノードを含む) の間、およびデータノードとほかのデータノードの間の通信が必要です。これらのプロセス間通信の待機時間は、ユーザークエリーの観測されるパフォーマンスと待機時間に直接影響を与える可能性があります。さらに、ノードのサイレント障害にもかかわらず一貫性とサービスを維持するために、NDB Cluster は、ノードからの通信の拡張損失をノード障害として処理するハートビートおよびタイムアウトメカニズムを使用します。これは、冗長性の低下につながる可能性があります。データの整合性を維持するために、NDB Cluster はノードグループ内の最後のノードで障害が発生したときにシャットダウンすることを思い出してください。したがって、強制的なシャットダウンのリスクを増やさないようにするため、ノード間通信の中断はできるかぎり回避すべきです。

データノードまたは API ノードに障害が発生すると、障害が発生したノードに関係するコミットされていないすべてのトランザクションが中止されます。データノードをリカバリするには、データノードのサービスを再開する前に、障害が発生したノードのデータを残存するデータノードのデータと同期し、ディスクベースの redo ログとチェックポイントログを再構築する必要があります。このリカバリには時間がかかる場合があり、その間、クラスタは冗長性が低下した状態で動作します。

ハートビートは、すべてのノードでハートビート信号が遅延なく生成されるかどうか依存しています。ノードに過大な負荷がかかったり、マシンの CPU がほかのプログラムと共有されるために不十分だったり、スワッピングによる遅延が発生したりすると、これは不可能になります。ハートビート生成の遅延が十分に大きくなると、ほかのノードは応答が遅いノードを障害発生ノードとして扱います。

このように遅いノードを障害発生ノードとして扱うことは、ノードの遅い動作がクラスタの残りの部分にどの程度影響するかによって、望ましい場合とそうでない場合があります。NDB Cluster 用の `HeartbeatIntervalDbDb` や `HeartbeatIntervalDbApi` などのタイムアウト値を設定する場合は、迅速な検出、フェイルオーバー、およびサービスへの復帰を実現しながら、費用のかかる誤検出を回避するよう注意する必要があります。

データノード間の通信待機時間が LAN 環境での予想値 (およそ 100 マイクロ秒) より大きくなると予想される場合は、タイムアウトのパラメータを増やして、許容する待機時間が構成したタイムアウトの範囲に十分に収まるようにする必要があります。このようにタイムアウトを増やすと、最大の障害検出時間および (その結果としての) サービスリカバリ時間にも類似の効果があります。

LAN 環境は、通常、安定した小さい待機時間で構成できるため、高速フェイルオーバーによる冗長性を提供できます。個々のリンク障害は、最小および制御された待機時間を TCP レベルで表示して回復できます (NDB Cluster は通常動作します)。WAN 環境では、待機時間に幅があり、冗長性についてもフェイルオーバーに時間がかかる可能性があります。個々のリンク障害では、エンドツーエンドの接続をリストアする前に、ルート変更を伝播する必要があります。これは、TCP レベルでは個々のチャネルの大きな待機時間として現れます。これらのシナリオで最悪の場合に観測される TCP 待機時間は、IP 層で障害を回避するために行われる再ルーティングの最大時間に関連しています。

23.1.4 NDB Cluster の新機能

次のセクションでは、以前のリリースシリーズと比較して、8.0.23 を介した MySQL NDB Cluster 8.0 での NDB Cluster の実装の変更点について説明します。NDB Cluster 8.0 は、NDB 8.0.19 以降の General Availability (GA) リリースとして使用できます。NDB Cluster 7.6 および 7.5 は、本番で引き続きサポートされている以前の GA リリースです。NDB Cluster 7.6 については、[What is New in NDB Cluster 7.6](#) を参照してください。NDB Cluster 7.5 に関する同様の情報については、[What is New in NDB Cluster 7.5](#) を参照してください。NDB Cluster 7.4 および 7.3 は以前の GA リリースで本番で引き続きサポートされていますが、本番用の新しい配備では NDB Cluster 8.0 を使用することをお勧めします。[MySQL NDB Cluster 7.3 and NDB Cluster 7.4](#) を参照してください。

NDB Cluster 8.0 の新機能

関心のある NDB Cluster 8.0 の主な変更点と新機能を次のリストに示します:

- 互換性の強化. 次の変更により、他の MySQL ストレージエンジンと比較して、NDB の動作における重要でない長期的な違いが軽減されます:

- MySQL サーバーと並行した開発。 このリリース以降、MySQL NDB Cluster は、次の機能を備えた新しい統合リリースモデルで、標準の MySQL 8.0 サーバーと並行して開発されています:
- NDB 8.0 は、MySQL 8.0 ソースコードツリーで開発、構築、およびリリースされます。
- NDB Cluster 8.0 リリースの番号スキームは、バージョン 8.0.13 以降の MySQL 8.0 のスキームに従います。
- NDB サポートを使用してソースを構築すると、次に示すように、`mysql -V` によって返されるバージョン文字列に `-cluster` が追加されます:

```
shell>> mysql -V
mysql Ver 8.0.23-cluster for Linux on x86_64 (Source distribution)
```

NDB バイナリには、次のように MySQL Server バージョンと NDB エンジンバージョンの両方が引き続き表示されます:

```
shell> ndb_mgm -V
MySQL distrib mysql-8.0.23 ndb-8.0.23, for Linux (x86_64)
```

MySQL Cluster NDB 8.0 では、これらの 2 つのバージョン番号は常に同じです。

NDB Cluster をサポートする MySQL 8.0.13 (またはそれ以降) ソースを構築するには、CMake オプション `-DWITH_NDBCLUSTER` を使用します。

- プラットフォームサポートノート。 NDB 8.0 は、プラットフォームサポートで次の変更を行います:
 - `NDBCLUSTER` では、32-bit プラットフォームはサポートされなくなりました。 NDB 8.0.21 以降、NDB 構築プロセスはシステムアーキテクチャーをチェックし、64 ビットプラットフォームでない場合は中止します。
 - NDB 8.0.18 以降では、64 ビット ARM CPU のソースから NDB を構築できます。現在、このサポートはソースのみであり、このプラットフォームにプリコンパイルされたバイナリは提供されていません。
 - データベース名とテーブル名。 NDB 8.0.18 の時点では、データベースおよびテーブルの識別子に対する 63 バイトの制限が削除されます。これらの識別子は、ほかの MySQL ストレージエンジンを使用するオブジェクトの場合と同様に、最大 64 バイトを使用できるようになりました。 [セクション 23.1.7.11 「前 NDB Cluster 8.0 で解決される NDB Cluster の問題」](#) を参照してください。
 - 外部キーに対して生成された名前。 NDB (バージョン 8.0.18 以降) では、内部的に生成された外部キーの命名にパターン `tbl_name_fk_N` が使用されるようになりました。これは、InnoDB で使用されるパターンに似ています。
 - スキーマとメタデータの分散および同期。 NDB 8.0 では、MySQL データディクショナリを使用して、クラスタに参加している SQL ノードにスキーマ情報を分散し、既存の SQL ノード間で新しいスキーマ変更を同期します。次のリストでは、この統合作業に関連する個々の拡張機能について説明します:
 - スキーマ分散の拡張機能。 スキーマ操作を処理し、その進行状況を追跡する NDB スキーマ配布コーディネータは NDB 8.0.17 で拡張され、スキーマ操作中に使用されるリソースがその最後に解放されるようになりました。以前は、この作業の一部はスキーマ分散クライアントによって実行されていました。これは、クライアントに必ずしもすべての必要な状態情報がないために変更されており、クライアントが完了前にコーディネータに通知せずにスキーマ操作を破棄することを決定した場合、リソースリークが発生する可能性があります。
- この問題を修正するために、スキーマ操作のタイムアウト検出がスキーマ配布クライアントからコーディネータに移動され、スキーマ操作中に使用されたリソースをクリーンアップする機会がコーディネータに提供されました。コーディネータは、進行中のスキーマ操作のタイムアウトを定期的にチェックし、タイムアウトの検出時に、指定されたスキーマ操作をまだ完了していない参加者を失敗としてマークします。また、スキーマ操作のタイムアウトが発生するたびに適切な警告が表示されます。(このようなタイムアウトが検出されると、スキーマ操作自体が続行されることに注意してください。) 追加のレポートは、アクティブなスキーマ操作のリストを、これらの操作の進行中に定期的に出力することによって行われます。

この作業の追加部分として、新しい `mysqld` オプション `--ndb-schema-dist-timeout` を使用すると、スキーマ操作がタイムアウトとマークされるまで待機する時間の長さを設定できます。

- ディスクデータファイルの分散. NDB Cluster 8.0.14 以降、NDB は MySQL データディクショナリを使用して、ディスクデータファイルおよび関連する構成 (テーブルスペースやログファイルグループなど) が、接続されているすべての SQL ノード間で正しく分散されていることを確認します。
- テーブルスペースオブジェクトのスキーマ同期. MySQL Server は、SQL ノードとして NDB クラスタに接続するときに、そのデータディクショナリを NDB デイクショナリにある情報と照合してチェックします。

以前は、新しい SQL ノードの接続時に同期された NDB オブジェクトはデータベースとテーブルだけでした。MySQL NDB Cluster 8.0.14 以降では、テーブルスペースやログファイルグループを含むディスクデータオブジェクトのスキーマ同期も実装されていました。これにより、テーブルスペースおよびログファイルグループが MySQL Server データディクショナリではなく NDB デイクショナリにリストアされたネイティブのバックアップおよびリストア後に、MySQL データディクショナリと NDB デイクショナリの間で不一致が発生する可能性がなくなります。

存在しないテーブルスペースを参照する `CREATE TABLE` ステートメントを発行することもできなくなりました。このようなステートメントはエラーで失敗します。

- データベース DDL 同期の拡張機能. NDB 8.0.17 で行われた作業により、新しく結合 (または再結合) された SQL ノードと既存の SQL ノード上のデータベースの同期がデータディクショナリを適切に使用できるようになり、この SQL ノードで見逃された可能性のあるデータベースレベルの操作 (`CREATE DATABASE`、`ALTER DATABASE` または `DROP DATABASE`) がクラスタに接続 (または再接続) されたときに正しく複製されるようになりました。

起動時に実行されるスキーマ同期化プロセスの一部として、SQL ノードはクラスタデータノード上のすべてのデータベースを独自のデータディクショナリ内のデータベースと比較するようになり、これらのいずれかが SQL ノードデータディクショナリから欠落していることが判明した場合、SQL ノードは `CREATE DATABASE` ステートメントを実行してローカルにインストールします。このように作成されたデータベースでは、ステートメントの実行時にこの SQL ノードで有効なデフォルトの MySQL Server データベースプロパティ (`character_set_database` や `collation_database` によって決定されるものなど) が使用されます。

- NDB メタデータ変更の検出および同期. NDB 8.0.16 は、MySQL データディクショナリを使用して、テーブル、テーブルスペース、ログファイルグループなどのデータオブジェクトのメタデータの更新を検出するための新しいメカニズムを実装しています。これは、バックグラウンドで実行され、NDB デイクショナリと MySQL データディクショナリの間での不整合を定期的にチェックする NDB メタデータ変更モニタースレッドであるスレッドを使用して行われます。

モニターは、デフォルトで 60 秒ごとにメタデータチェックを実行します。ポーリング間隔を調整するには、`ndb_metadata_check_interval` システム変数の値を設定します。ポーリングを完全に無効にするには、`ndb_metadata_check` システム変数を `OFF` に設定します。ステータス変数 (こちらも NDB 8.0.16 で追加) `Ndb_metadata_detected_count` は、`mysqld` が最後に起動されてから不一致が検出された回数を示します。

バージョン 8.0.18 以降、NDB では、起動後の操作中にメタデータチェンジモニタースレッドによって発行された NDB テーブル、ログファイルグループおよびテーブルスペースオブジェクトの不一致が自動的にチェックされ、NDB binlog スレッドによって同期化されます。

NDB 8.0.18 は、自動同期に関連する 2 つのステータス変数も追加: `Ndb_metadata_synced_count` には、自動的に同期されたオブジェクトの数が表示されます。`Ndb_metadata_excluded_count` は、同期が失敗したオブジェクトの数を示します (NDB 8.0.22 より前では、この変数の名前は `Ndb_metadata_blacklist_size` でした)。また、どのオブジェクトが同期化されているかを確認するには、クラスタログを調べます。

NDB 8.0.19 は、変更が検出および同期されるオブジェクトにデータベースを追加することによって、この機能をさらに強化します。NDB テーブルで実際に使用されるデータベースのみが処理され、MySQL データディクショナリに存在する可能性のある他のデータベースは無視されます。これにより、このデータベースを手動で作成するために、NDB にテーブルが存在していても、それが属するテーブルおよびデータベースが SQL ノードに存在

しない場合に、以前の要件がなくなります。このような場合は、データベースおよびそれに属するすべての NDB テーブルを SQL ノードに自動的に作成する必要があります。

NDB 8.0.19 では、`ndb_metadata_sync` システム変数も導入されています。この変数を `true` に設定すると、`ndb_metadata_check_interval` および `ndb_metadata_check` に対して行われたすべての設定がオーバーライドされ、変更モニタースレッドが連続したメタデータ変更検出を開始します。

NDB 8.0.22 以降では、`ndb_metadata_sync` を `true` に設定すると、以前に同期が失敗したオブジェクトのリストがクリアされるため、個々のテーブルを検出したり、SQL ノードをクラスタに再接続して同期を再トリガーしたりする必要がなくなります。また、この変数を `false` に設定すると、再試行を待機しているオブジェクトのリストがクリアされます。

NDB 8.0.21 以降、ログメッセージまたはステータス変数から取得できる自動同期の現在の状態に関する詳細情報は、MySQL パフォーマンススキーマに追加された 2 つの新しいテーブルによって提供されます。テーブルは次のとおりです：

- `ndb_sync_pending_objects`: NDB デイクシヨナリと MySQL データデイクシヨナリの間で不一致が検出された (および自動同期から除外されていない) データベースオブジェクトに関する情報が含まれます。
- `ndb_sync_excluded_objects`: 除外された NDB データベースオブジェクトに関する情報が含まれます。これらのオブジェクトは、NDB デイクシヨナリと MySQL データデイクシヨナリの間で同期できないため、手動操作が必要になります。

これらのテーブルのいずれかの行には、データベースオブジェクトの親スキーマ、名前およびタイプが表示されます。オブジェクトのタイプには、スキーマ、テーブルスペース、ログファイルグループおよびテーブルがあります。(オブジェクトがログファイルグループまたはテーブルスペースの場合、親スキーマは `NULL` です。) また、`ndb_sync_excluded_objects` テーブルには、オブジェクトが除外された理由が表示されます。

これらのテーブルは、`NDBCLUSTER` ストレージエンジンのサポートが有効になっている場合にのみ存在します。これらのテーブルの詳細は、[セクション 27.12.12 「パフォーマンススキーマ NDB Cluster テーブル」](#) を参照してください。

- NDB テーブルの追加メタデータの変更。 NDB 8.0.14 以降では、NDB テーブルの追加のメタデータプロパティは、以前のバージョンと同様にテーブルのバイナリ表現を格納するのではなく、MySQL データデイクシヨナリから直列化されたメタデータを格納するために使用されます。(これは `.frm` ファイルであり、MySQL Server-see [第 14 章 「MySQL データデイクシヨナリ」](#) では使用されなくなりました。) この変更をサポートする作業の一環として、テーブルの追加メタデータの使用可能なサイズが増加しました。つまり、NDB Cluster 8.0.14 以降で作成された NDB テーブルは、以前の NDB Cluster リリースと互換性がありません。以前のリリースで作成されたテーブルは NDB 8.0.14 以降で使用できますが、それより前のバージョンで開くことはできません。

このメタデータには、NDB 8.0.13 で実装された NDB API メソッド `getExtraMetadata()` および `setExtraMetadata()` を使用してアクセスできます。

詳細は、[セクション 23.2.7 「NDB Cluster のアップグレードおよびダウングレード」](#) を参照してください。

- `.frm` ファイルを使用したテーブルのオンザフライアップグレード。 NDB 7.6 以前で作成されたテーブルには、MySQL 8.0 ではサポートされなくなった圧縮 `.frm` ファイルの形式でメタデータが含まれています。NDB 8.0 へのオンラインアップグレードを容易にするために、NDB はこのメタデータのオンザフライ変換を実行し、MySQL Server データデイクシヨナリに書き込みます。これにより、NDB Cluster 8.0 内の `mysqld` は、以前のバージョンの NDB ソフトウェアによるテーブルの以降の使用を妨げずにテーブルを操作できます。

重要

NDB 8.0 でテーブル構造が変更されると、そのメタデータはデータデイクシヨナリを使用して格納され、NDB 7.6 以前からはアクセスできなくなります。

この拡張機能により、以前のバージョンを使用して作成された NDB バックアップを NDB 8.0 以降を実行しているクラスタに復元することもできます。

- メタデータ整合性チェックのエラーロギング。 NDB 8.0 で以前に実行された作業の一部として、NDB デイクシヨナリ内の NDB テーブルの表現と MySQL データデイクシヨナリ内の対応する表現との間の自動同期の一部

として実行されるメタデータチェックには、テーブル名、ストレージエンジン、および内部 ID が含まれます。NDB 8.0.23 以降、チェックされるプロパティの範囲は、次のデータオブジェクトのプロパティを含むように拡張されます:

- Columns
- インデックス
- 外部キー

また、メタデータプロパティの不一致の詳細が MySQL サーバーのエラーログに書き込まれるようになります。エラーログメッセージに使用される書式は、差異がテーブルレベルで検出されるか、コラム、インデックスまたは外部キーのレベルで検出されるかによって若干異なります。テーブルレベルのプロパティの不一致によるログエラーの形式を次に示します。ここで、`property` はプロパティ名、`ndb_value` は NDB デイクシヨナリに格納されているプロパティ値、`mysqld_value` は MySQL データデイクシヨナリに格納されているプロパティの値です:

```
Diff in 'property' detected, 'ndb_value' != 'mysqld_value'
```

コラム、インデックスおよび外部キーのプロパティの不一致の場合、形式は次のようになります。ここで、`obj_type` は `column`、`index` または `foreign key` のいずれかで、`obj_name` はオブジェクトの名前です:

```
Diff in obj_type 'obj_name.property' detected, 'ndb_value' != 'mysqld_value'
```

NDB Cluster 内の SQL ノードとして機能する任意の `mysqld` のデータデイクシヨナリにインストールされると、NDB テーブルの自動同期中にメタデータチェックが実行されます。`mysqld` がデバッグコンパイルされている場合は、`CREATE TABLE` ステートメントが実行されるたび、および NDB テーブルが開かれるたびにチェックが行われます。

- NDB_STORED_USER とのユーザー権限の同期。 NDB 8.0.18 以降では、`NDB_STORED_USER` 権限を使用して、SQL ノード間でユーザー、役割、および特権を共有および同期するための新しいメカニズムを使用できます。NDB 7.6 以前で実装された分散特権 ([Distributed Privileges Using Shared Grant Tables](#) を参照) はサポートされなくなりました。

SQL ノードでユーザーアカウントが作成されると、ユーザーとその権限を NDB に格納できるため、次のような `GRANT` ステートメントを発行して、クラスタ内のすべての SQL ノード間で共有できます:

```
GRANT NDB_STORED_USER ON *.* TO 'jon'@'localhost';
```

`NDB_STORED_USER` は常にグローバルスコープを持ち、`ON *.*` を使用して付与する必要があります。`mysql.session@localhost` や `mysql.infoschema@localhost` などのシステム予約アカウントには、この権限を割り当ててはできません。

適切な `GRANT NDB_STORED_USER` ステートメントを発行して、ロールを SQL ノード間で共有することもできます。このようなロールをユーザーに割り当てても、ユーザーは共有されません。`NDB_STORED_USER` 権限は、各ユーザーに明示的に付与する必要があります。

`NDB_STORED_USER` を持つユーザーまたは役割は、その特権とともに、特定の NDB Cluster に参加するとすぐにすべての SQL ノードと共有されます。ユーザーまたはロールの権限に対する変更は、接続されているすべての SQL ノードとただちに同期されます。このような変更は接続されている任意の SQL ノードから行うことができますが、異なる SQL ノードからの権限に影響するステートメントの実行順序はすべての SQL ノードで同じであることが保証されないため、指定された SQL ノードからのみ行うことをお勧めします。

アップグレードの意味。 MySQL サーバー特権システム ([セクション6.2.3「付与テーブル」](#) を参照) が変更されたため、NDB ストレージエンジンを使用する特権テーブルは NDB 8.0 で正しく機能しません。NDB 7.6 以前で作成されたこのような特権テーブルを保持する必要はありませんが、アクセス制御には使用されなくなりました。NDB 8.0.16 以降、SQL ノードとして機能し、NDB でこのようなテーブルを検出する `mysqld` は、MySQL サーバーログに警告を書き込み、`InnoDB` シャドウテーブルをそれぞれ自体に対してローカルに作成します。このようなシャドウテーブルは、クラスタに接続されている各 MySQL サーバー上に作成されます。NDB 7.6 以前からアップグレードを実行する場合、SQL ノードとして機能するすべての MySQL サーバーがアップグレードされると、NDB を使用してい

の特権テーブルを `ndb_drop_table` を使用して安全に削除できます (セクション23.2.7「NDB Cluster のアップグレードおよびダウングレード」を参照)。

`ndb_restore` ユーティリティの `--restore-privilege-tables` オプションは非推奨ですが、NDB 8.0 では引き続き適用され、以前のリリースの NDB Cluster から取得されたバックアップに存在する分散特権テーブルを NDB 8.0 を実行しているクラスタに復元するために引き続き使用できます。これらのテーブルは、前の段落で説明されているように処理されます。

共有ユーザーおよび権限は `ndb_sql_metadata` テーブルに格納されますが、NDB 8.0.19 以降の `ndb_restore` ではデフォルトで復元されません。`--include-stored-grants` オプションを指定してこれを行うことができます。

- **INFORMATION_SCHEMA の変更点.** `INFORMATION_SCHEMA.FILES` テーブルのディスクデータファイルに関する情報の表示では、次の変更が行われます:
 - テーブルスペースおよびログファイルグループは、`FILES` テーブルに表示されなくなりました。(これらの構成は実際にはファイルではありません。)
 - 各データファイルは、`FILES` テーブルの単一行でテーブルされるようになりました。各 undo ログファイルも、このテーブルでは 1 行のみでテーブルされるようになりました。(以前は、各データノード上のこれらの各ファイルのコピーごとに行が表示されていました。)

また、`INFORMATION_SCHEMA` テーブルには、MySQL クラスタテーブルのテーブルスペース統計が移入されるようになりました。(Bug #27167728)

- **ndb_perror のエラー情報.** `pererror` の非推奨の `--ndb` オプションは削除されました。かわりに、`ndb_perror` を使用して、`NDB` エラーコードからエラーメッセージ情報を取得します。(Bug #81704、Bug #81705、Bug #23523926、Bug #23523957)
- **条件プッシュダウンの拡張機能.** 以前は、条件プッシュダウンは、条件がプッシュされたのと同じテーブルのカラム値を参照する述語条件に制限されていました。NDB 8.0.16 では、この制限は、クエリープラン内の以前のテーブルのカラム値をプッシュされた条件から参照することもできるように削除されます。NDB 8.0.18 では、同じテーブル内のカラム間の比較と同様に、カラム式を比較する結合がサポートされています。比較するカラムおよびカラム式は、完全に同じタイプである必要があります。つまり、これらの属性が適用される場合は常に、同じ符号性、長さ、文字セット、精度およびスケールである必要があります。

条件の大きな部分をプッシュダウンすると、データノードによってより多くの行をフィルタで除外できるため、結合処理中に `mysqld` が処理する必要がある行数が削減されます。これらの拡張機能の別の利点は、SQL ノード上の単一の `mysqld` プロセスではなく、LDM スレッドで並列でフィルタリングを実行できることです。これにより、クエリーのパフォーマンスが大幅に向上する可能性があります。

比較対象のカラム値間の型の互換性に関する既存のルールが引き続き適用されます (セクション8.2.1.5「エンジンコンディショニングプッシュダウンの最適化」を参照)。

NDB 8.0.21 では、次の追加の改善が行われています:

- **NOT EXISTS および NOT IN クエリー** (セクション8.2.2.1「準結合変換による IN および EXISTS サブクエリー述語の最適化」を参照) の変換によって MySQL オプティマイザによって生成されたアンチ結合は、`NDB` によってデータノードにプッシュダウンできます。

これは、テーブルにプッシュされていない条件がなく、外部結合をプッシュダウンするために満たす必要がある他の条件をクエリーが満たしている場合に実行できます。
- **NDB は、連結先のテーブルから行を取得する前に、非依存スカラーサブクエリーを識別および評価しようとしています。** その場合、取得された値は、値を提供したサブクエリーを使用するかわりに、プッシュされた条件の一部として使用されます。
- **最大行サイズの増加.** NDB 8.0.18 は、`NDBCLUSTER` テーブルに格納できる最大バイト数を 14000 から 30000 バイトに増やします。

`BLOB` または `TEXT` カラムは、以前と同様に、この合計の 264 バイトを引き続き使用します。

`NDB` テーブルの固定幅カラムの最大オフセットは 8188 バイトです。これは、8.0.18 より前のリリースからも変更されていません。

詳細は、[セクション23.1.7.5「NDB Cluster 内のデータベースオブジェクトに関連付けられる制限」](#)を参照してください。

- `ndb_mgm SHOW` コマンドおよびシングルユーザーモード。 NDB 8.0.17 以降では、シングルユーザーモードのクラスタの場合、管理クライアントの `SHOW` コマンドの出力に、このモードが有効なときに排他的アクセス権を持つ API または SQL ノードが示されます。
- オンラインカラム名の変更。 NDB 8.0.18 以降では、`ALGORITHM=INPLACE` を使用して NDB テーブルのカラムの名前をオンラインで変更できます。詳しくは[セクション23.5.11「NDB Cluster での ALTER TABLE を使用したオンライン操作」](#)をご覧ください。
- `ndb_mgmd` の起動時間の改善。 NDB 8.0.18 以降では、管理ノードデーモンの起動時間が次のように大幅に改善されました:
 - 構成データのノードプロパティをハッシュテーブルで処理するために以前に `ndb_mgmd` で使用されていたリストデータ構造を置き換えるため、管理サーバーの全体的な起動時間が 6 以上減少しました。
 - また、管理サーバーの `hosts` ファイルに存在しないデータおよび SQL ノードのホスト名がクラスタ構成ファイルで使用されている場合は、`ndb_mgmd` の起動時間を以前のケースの 20 倍短くすることができます。
- NDB API の拡張機能。 NDB 8.0.18 以降では、`NdbScanFilter::cmp()` と `NdbInterpretedCode` のいくつかの比較方法を使用して、テーブルカラム値を相互に比較できます。影響を受ける `NdbInterpretedCode` メソッドを次に示します:
 - `branch_col_eq()`
 - `branch_col_ge()`
 - `branch_col_gt()`
 - `branch_col_le()`
 - `branch_col_lt()`
 - `branch_col_ne()`

前述のすべての方法で、比較されるテーブルのカラム値は、長さ、精度、符号性、スケール、文字セットおよび照合順序に関して、正確に一致する型である必要があります。

詳細は、個々の API メソッドの説明を参照してください。

- オフラインマルチスレッドインデックスの構築。 ファイル I/O、圧縮、解凍などの通常の I/O 職務とは対照的に、順序付けられたインデックスのオフラインマルチスレッドビルドを実行する I/O スレッドに使用するコアのセットを指定できるようになりました。このコンテキストの「オフライン」とは、親テーブルが書き込まれていないときに実行される順序付けされたインデックスの構築を指します。このような構築は、NDB クラスタがノードまたはシステムの再起動を実行したとき、または `ndb_restore --rebuild-indexes` を使用したバックアップからのクラスタのリストアの一環として行われます。

また、オフラインインデックス作成作業のデフォルトの動作は、I/O スレッド用に予約されたコアに制限されるのではなく、`ndbmt` で使用可能なすべてのコアを使用するように変更されます。これにより、再起動とリストアの時間、パフォーマンス、可用性およびユーザーエクスペリエンスを向上させることができます。

この拡張機能は、次のように実装されます:

1. `BuildIndexThreads` のデフォルト値は 0 から 128 に変更されています。つまり、オフライン順序付けされたインデックス構築はデフォルトでマルチスレッド化されるようになりました。
2. `TwoPassInitialNodeRestartCopy` のデフォルト値は、`false` から `true` に変更されました。つまり、最初のノードの再起動では、最初にすべてのデータが「ライブ」ノードから、インデックス構築の順序付きインデックスをオフラインで作成せずに開始しているノードにコピーされてから、そのデータがライブノードと再度同期化されます。つまり、2 回同期化され、2 つの同期化の間でオフラインでインデックスが構築されます。これにより、ノードの初期再起動はノードの通常の再起動と同様に動作し、インデックスの作成に必要な時間が短縮されます。

3. `ThreadConfig` 構成パラメータに新しいスレッドタイプ (`idxbld`) が定義され、オフラインインデックス構築スレッドを特定の CPU にロックできるようになりました。

また、NDB では、次の 2 つの基準によって `ThreadConfig` にアクセス可能なスレッドタイプが区別されるようになりました:

1. スレッドが実行スレッドかどうか。 `main`, `ldm`, `recv`, `rep`, `tc` および `send` タイプのスレッドは実行スレッドであり、`io`、`watchdog` および `idxbld` タイプは実行スレッドではありません。
2. 指定されたタスクへのスレッドの割当てが永続的か一時的か。現在、`idxbld` 以外のすべてのスレッドタイプは永続的です。

追加情報については、マニュアルで示されているパラメータの説明を参照してください。(Bug #25835748、Bug #26928111)

- `logbuffers` テーブルバックアッププロセス情報。 NDB バックアップを実行すると、`ndbinfo.logbuffers` テーブルに、各データノード上のバックアッププロセスによるバッファの使用状況に関する情報が表示されるようになりました。これは、`REDO` および `DD-UNDO` に加えて、2 つの新しいログタイプを反映する行として実装されます。これらのいずれかの行のログタイプは `BACKUP-DATA` で、バックアップ中にフラグメントをバックアップファイルにコピーするために使用されるデータバッファの量が表示されます。もう一方の行のログタイプは `BACKUP-LOG` で、バックアップの開始後に行われた変更を記録するためにバックアップ中に使用されたログバッファの量が表示されます。これらの `log_type` 行のそれぞれが、クラスタ内の各データノードの `logbuffers` テーブルに表示されます。これら 2 つのログタイプを持つ行は、NDB バックアップが現在進行中の場合にのみテーブルに存在します。(Bug #25822988)
- Windows 上の `ndbinfo.processes` テーブル。 `mysqld` を生成および再起動するために `RESTART` によって Windows プラットフォームで使用されるモニタープロセスのプロセス ID が、`angel_pid` として `processes` テーブルに表示されるようになりました。
- 文字列ハッシュの改善。 NDB 8.0 より前では、すべての文字列ハッシュは、最初に文字列を正規化された形式に変換してから、結果のバイナリイメージを MD5 ハッシュ化することに基づいていました。これにより、次の理由でパフォーマンスの問題が発生する可能性があります:
 - 正規化された文字列は、常に空白がその全長に埋め込まれます。 `VARCHAR` の場合、多くの場合、これには元の文字列の文字よりも多くのスペースの追加が含まれます。
 - このスペースパディング用に文字列ライブラリが最適化されなかったため、ユースケースによってはかなりのオーバーヘッドが発生しました。
 - パディングセマンティクスは文字セット間で異なり、その一部は全長までパディングされませんでした。
 - 変換された文字列は、空白埋めなしでも非常に大きくなる可能性があります。Unicode 9.0 照合順序によっては、単一のコードポイントを 100 バイト以上の文字データに変換できる場合があります。
 - 後続の MD5 ハッシングは主にスペースで埋めることで構成されており、特に効率的ではなかったため、L1 キャッシュの重要な部分をフラッシュすることでパフォーマンスのペナルティが増大する可能性があります。

照合には独自のハッシュ関数が用意されており、最初に正規化された文字列を作成せずに文字列を直接ハッシュします。また、Unicode 9.0 照合の場合、ハッシュはパディングなしで計算されます。Unicode 9.0 照合順序を使用して識別された文字列をハッシュするたびに、NDB でこの組込み関数が利用されるようになりました。

他の照合の場合、変換された文字列でハッシュパーティション化されている既存のデータベースが存在するため、NDB では、これらを使用する文字列をハッシュ化するための以前の方法を引き続き使用して互換性を維持します。(Bug #89590、Bug #89604、Bug #89609、Bug #27515000、Bug #27523758、Bug #27522732)

- **RESET MASTER の変更.** MySQL Server はグローバル読み取りロックを使用して **RESET MASTER** を実行するようになったため、NDB Cluster で使用された場合のこのステートメントの動作は、次の 2 つの点で変更されました:
 - シノクロであることは保証されなくなりました。つまり、バイナリログがローテーションされるまで、**RESET MASTER** が発行される直前に行われる読み取りがログに記録されない可能性があります。
 - バイナリログを書き込む同じ SQL ノード上でステートメントが発行されたか、同じクラスタ内の別の SQL ノード上で発行されたかにかかわらず、まったく同じように動作するようになりました。

注記

SHOW BINLOG EVENTS、**FLUSH LOGS** およびほとんどのデータ定義ステートメントは、以前の **NDB** バージョンと同様に、同期方式で動作します。

- **ndb_restore オプションの使用法.** NDB 8.0.16 以降では、**ndb_restore** を呼び出すときに **--nodeid** と **--backupid** の両方のオプションが必要です。
- **ndb_log_bin のデフォルト.** NDB 8.0.16 以降、**ndb_log_bin** システム変数のデフォルト値が **TRUE** から **FALSE** に変更されました。
- **動的トランザクションリソース割当て.** トランザクションコルディネータ (**The DBTC Block** を参照) でのリソースの割当ては、動的メモリープールを使用して実行されるようになりました。これは、**MaxDMLOperationsPerTransaction**、**MaxNoOfConcurrentIndexOperations**、**MaxNoOfConcurrentOperations**、**MaxNoOfConcurrentScans**、**MaxNoOfConcurrentTransactions**、**MaxNoOfFiredTriggers**、**MaxNoOfLocalScans** などのデータノード構成パラメータによって決定されるリソース割り当てと **TransactionBufferMemory** が、これらの各パラメータによって表される負荷がそのようなすべてのリソースのターゲット負荷内にある場合、使用可能なリソースの合計を超えないように制限できることを意味します。

この作業の一環として、次に示すように、**DBTC** のトランザクションリソースを制御するいくつかの新しいデータノードパラメータが追加されました:

- **ReservedConcurrentIndexOperations**
- **ReservedConcurrentOperations**
- **ReservedConcurrentScans**
- **ReservedConcurrentTransactions**
- **ReservedFiredTriggers**
- **ReservedLocalScans**
- **ReservedTransactionBufferMemory**.

詳細は、前述のパラメータの説明を参照してください。

- **データノードごとに複数の LDM を使用したバックアップ.** 複数のローカルデータマネージャ (LDM) を使用して、個々のデータノードで **NDB** バックアップを平行に実行できるようになりました。(以前は、バックアップはデータノード間で並列で実行されていましたが、常にデータノードプロセス内でシリアル化されていました。) **ndb_mgm** クライアントの **START BACKUP** コマンドでこの機能を有効にするために特別な構文は必要ありませんが、すべてのデータノードが複数の LDM を使用している必要があります。つまり、データノードは **ndbmtid** を実行している必要があります (**ndbd** はシングルスレッドであるため、常に LDM が 1 つしかありません)、バックアップを取得する前に複数の LDM を使用するように構成する必要があります。これを行うには、マルチスレッドデータノード構成パラメータ **MaxNoOfExecutionThreads** または **ThreadConfig** のいずれかに適切な設定を選択します。

複数の LDM を使用したバックアップでは、サブディレクトリが LDM ごとに **BACKUP/BACKUP-backup_id/**ディレクトリの下に作成されます。**ndb_restore** では、これらのサブディレクトリが自動的に検出されるようになり、存在する場合は、バックアップの平行リストアが試行されます。詳細は、**セクション 23.4.23.3 「平行で作成されたバックアップからのリストア」** を参照してください。(シングルスレッドバックアップは、以前のバージョンの **NDB** と同様にリストアされます。) 通常の復元手順を変更することによって、以前のバージョンの **NDB Cluster**

から `ndb_restore` バイナリを使用して並列で作成されたバックアップを復元することもできます。これを行う方法については、[パラレルバックアップのシリアルリスト](#) を参照してください。

- バイナリ構成ファイルの拡張。 NDB 8.0.18 以降では、管理サーバーのバイナリ構成ファイルに新しい形式が使用されます。以前は、クラスタ構成ファイルに最大 16381 個のセクションが表示されていましたが、セクションの最大数は 4G でした。これは、この変更前に可能だったよりも多くのノードをクラスタでサポートすることを目的としています。

新しい形式へのアップグレードは比較的シームレスであり、管理サーバーは問題なく古い形式を読み取ることができ、手動操作が必要になることはほとんどありません。NDB 8.0.18 (またはそれ以降) から古いバージョンの NDB Cluster ソフトウェアへのダウングレードでは、バイナリ構成ファイルを手動で削除するか、`--initial` オプションを使用して古い管理サーバーバイナリを起動する必要があります。

詳細は、[セクション 23.2.7 「NDB Cluster のアップグレードおよびダウングレード」](#) を参照してください。

- データノードの数の増加。 NDB 8.0.18 は、クラスタごとにサポートされるデータノードの最大数を 144 に増やします (以前は 48 でした)。データノードは、1 から 144 の範囲のノード ID を使用できるようになりました。

以前は、管理ノードの推奨ノード ID は 49 および 50 でした。これらは引き続き管理ノードでサポートされますが、これを使用するとデータノードの最大数が 142 に制限されるため、ノード ID 145 および 146 を管理ノードに使用することをお勧めします。

この作業の一環として、データノード `sysfile` に使用される形式がバージョン 2 に更新されました。このファイルには、各ノードの最後のグローバルチェックポイントインデックス、再起動ステータス、ノードグループメンバーシップなどの情報が記録されます ([NDB Cluster Data Node File System Directory](#) を参照)。

- RedoOverCommitCounter および RedoOverCommitLimit の変更。 これらを 0 に設定するセマンティクスがあいまいなため、NDB 8.0.19 以降、データノード構成パラメータ `RedoOverCommitCounter` および `RedoOverCommitLimit` のそれぞれの最小値が 1 に増加しました。
- `ndb_autoincrement_prefetch_sz` の変更点。 NDB 8.0.19 では、`ndb_autoincrement_prefetch_sz` サーバースystem 変数のデフォルト値が 512 に増加しています。
- パラメータ `maxmimums` および `default` の変更。 NDB 8.0.19 は、構成パラメータの最大値とデフォルト値を次のように変更します:
 - `DataMemory` の最大値は 16 テラバイトに増加します。
 - `DiskPageBufferMemory` の最大値も 16 テラバイトに増加します。
 - `StringMemory` のデフォルト値は 25% に増加します。
 - `LcpScanProgressTimeout` のデフォルトは 180 秒に増加します。
- ディスクデータチェックポイントの改善。 NDB Cluster 8.0.19 には、ソリッドステートドライブなどの不揮発性メモリーデバイスおよびそのようなデバイスの NVMe 仕様を使用する場合に、「ディスクデータ」テーブルおよびテーブルスペースのチェックポイントの待機時間を短縮するのに役立つ多くの新しい拡張機能が用意されています。これらの改善点には、次のリストの改善点が含まれます:
 - チェックポイントディスク書き込みのバーストの回避
 - redo ログまたは undo ログがいっぱいになったときのディスクデータテーブルスペースのチェックポイントの高速化
 - 必要に応じて、チェックポイントをディスクチェックポイントとインメモリーチェックポイントのバランスを調整
 - 過負荷からディスクデバイスを保護し、高負荷での低レイテンシを保証

この作業の一環として、NDB 8.0.19 には 2 つの新しいデータノード構成パラメータが導入されています。`MaxDiskDataLatency` では、ディスクアクセスに許可されるレイテンシの程度に上限が設定され、この時間よりもトランザクションの中断に時間がかかります。`DiskDataUsingSameDisk` では、このようなテーブルスペースのチェックポイントを実行できる速度を上げることで、別々のディスク上の「ディスクデータ」テーブルスペースを利用できます。

さらに、ここに一覧表示されている `ndbinfo` データベース内の 3 つの新しいテーブルは、ディスクデータパフォーマンスに関する情報を提供します (こちらも NDB 8.0.19 で追加):

- `diskstat` テーブルは、過去 1 秒間の「ディスクデータ」テーブルスペースへの書き込みについてレポート
- `diskstats_1sec` テーブルは、過去 20 秒間の「ディスクデータ」テーブルスペースへの書き込みについてレポート
- `pgman_time_track_stats` テーブルには、「ディスクデータ」テーブルスペースに関連するディスク操作のレイテンシがレポートされます
- メモリー割当てと TransactionMemory. NDB 8.0.19 では、トランザクションおよびローカルデータマネージャー (LDM) メモリーをプールするために実行される作業の一環として、トランザクションのデータノードメモリーの割り当てを簡略化する新しい `TransactionMemory` パラメータが導入されました。このパラメータは、非推奨になったいくつかの古いトランザクションメモリーパラメータを置き換えることを目的としています。

トランザクションメモリーは、次の 3 つの方法のいずれかで設定できるようになりました:

- いくつかの構成パラメータは `TransactionMemory` と互換性がありません。これらのいずれかが設定されている場合、`TransactionMemory` は設定できず (`TransactionMemory` と互換性のないパラメータ を参照)、データノードトランザクションメモリーは NDB 8.0.19 より前の状態で決定されます。

注記

`config.ini` ファイルで `TransactionMemory` とこれらのパラメータを同時に設定しようとすると、管理サーバーが起動しなくなります。

- `TransactionMemory` が設定されている場合、この値はトランザクションメモリーの決定に使用されます。前の項目で説明した互換性のないパラメータも設定されている場合、`TransactionMemory` は設定できません。
- 互換性のないパラメータが設定されておらず、`TransactionMemory` も設定されていない場合、トランザクションメモリーは NDB によって設定されます。

詳細は、`TransactionMemory` および [セクション 23.3.3.13 「データノードのメモリー管理」](#) の説明を参照してください。

- 追加のフラグメントレプリカのサポート. NDB 8.0.19 は、本番でサポートされるフラグメントレプリカの最大数を 2 から 4 に増やします。(以前は、`NoOfReplicas` を 3 または 4 に設定できましたが、テストで正式にサポートまたは検証されていませんでした。)
- スライスによる復元. NDB 8.0.20 以降では、`ndb_restore` に実装されている 2 つの新しいオプションを使用して、バックアップをほぼ同じ部分 (スライス) に分割し、これらのスライスを並列で復元できます:
 - `--num-slices` は、バックアップを分割するスライスの数を決定します。
 - `--slice-id` は、`ndb_restore` の現在のインスタンスによって復元されるスライスの ID を提供します。

これにより、`ndb_restore` の複数のインスタンスを使用してバックアップのサブセットをパラレルにリストアできるため、リストア操作の実行に必要な時間が短縮される可能性があります。

詳細は、`ndb_restore --num-slices` オプションの説明を参照してください。

- 有効なフラグメントレプリカからの読取り. NDB 8.0.19 以降では、すべての NDB テーブルでフラグメントレプリカからの読取りがデフォルトで有効になっています。これは、`ndb_read_backup` システム変数のデフォルト値が ON になり、新しい NDB テーブルの作成時に `NDB_TABLE` コメントオプション `READ_BACKUP` の値が 1 になることを意味します。フラグメントレプリカからの読取りを有効にすると、書き込みへの影響を最小限に抑えながら、NDB テーブルからの読取りのパフォーマンスが大幅に向上します。

詳細は、`ndb_read_backup` システム変数および [セクション 13.1.20.11 「NDB_TABLE オプションの設定」](#) の説明を参照してください。

- `ndb_blob_tool` の拡張機能. NDB 8.0.20 以降、`ndb_blob_tool` ユーティリティーはインラインパートが存在する欠落している BLOB パートを検出し、それらを正しい長さのプレースホルダー BLOB パート (スペース文字で構成される) に置き換えることができます。BLOB 部分が欠落しているかどうかを確認するには、このプログラムで

--check-missing オプションを使用します。欠落している BLOB 部分をプレースホルダに置き換えるには、--add-missing オプションを使用します。

詳細は、[セクション23.4.6「ndb_blob_tool — NDB Cluster テーブルの BLOB および TEXT カラムのチェックおよび修復」](#)を参照してください。

- ndbinfo バージョニング。 NDB 8.0.20 以降では、ndbinfo テーブルのバージョニングがサポートされ、そのテーブルの現在の定義が内部的に保持されます。起動時に、NDB はサポートされている ndbinfo バージョンをデータディクショナリに格納されているバージョンと比較します。バージョンが異なる場合、NDB は古い ndbinfo テーブルを削除し、現在の定義を使用して再作成します。
- Fedora Linux のサポート。 NDB 8.0.20 以降、Fedora Linux は NDB Cluster コミュニティリリースでサポートされているプラットフォームであり、Oracle がこの目的で提供する RPM を使用してインストールできます。これらは「[NDB Cluster のダウンロードページ](#)」から取得できます。
- NDB プログラム -NDBT 依存関係の削除。 多数の NDB ユーティリティプログラムの NDBT ライブラリへの依存性が削除されました。このライブラリは開発用に内部的に使用され、通常の使用には必要ありません。これらのプログラムに含めると、テスト時に不要な問題が発生する可能性があります。

影響を受けるプログラムを、依存関係が削除された NDB バージョンとともに次に示します：

- ndb_restore、NDB 8.0.17
- ndb_delete_all、NDB 8.0.18
- ndb_show_tables、NDB 8.0.20
- ndb_waiter、NDB 8.0.20

ユーザーに対するこの変更の主な影響は、これらのプログラムが実行の完了後に `NDBT_ProgramExit - status` を印刷しなくなることです。このような動作に依存するアプリケーションは、示されたバージョンにアップグレードするときに変更を反映するように更新する必要があります。

- 外部結合および準結合のプッシュダウン。 NDB 8.0.20 で実行される作業では、主キーまたは一意キー検索を使用するものだけでなく、多くの外部結合および準結合をデータノードにプッシュダウンできます ([セクション 8.2.1.5「エンジンコンディションプッシュダウンの最適化」](#)を参照)。

プッシュできるようになったスキャンを使用した外部結合には、次の条件を満たす外部結合が含まれます：

- テーブルにプッシュされていない条件はありません
- 同じ結合ネスト内または依存する上位結合ネスト内の他のテーブルに未プッシュ条件がありません
- 同じ結合ネスト内または依存する上位結合ネスト内の他のすべてのテーブルもプッシュされます

インデックススキャンを使用する準結合は、プッシュされた外部結合で示された条件を満たし、`firstMatch` 戦略を使用する場合にプッシュできるようになりました ([セクション8.2.2.1「準結合変換による IN および EXISTS サブクエリ述語の最適化」](#)を参照)。

結合をプッシュできない場合、`EXPLAIN` は理由を指定する必要があります。

- 外部キーと大文字小文字の区別。 NDB では、定義された大/小文字を使用して外部キーの名前が格納されます。以前は、`lower_case_table_names` システム変数の値が 0 に設定されていた場合、`SELECT` で使用されている外部キー名と格納されている名前を持つ他の SQL ステートメントとの大/小文字を区別した比較が実行されていました。NDB 8.0.20 以降、このような比較は、`lower_case_table_names` の値に関係なく、常に大文字と小文字を区別しない方法で実行されるようになりました。
- 複数のトランスポータ。 NDB 8.0.20 では、データノードのペア間のノード間通信を処理するための複数のトランスポータのサポートが導入されています。これにより、クラスタ内の各ノードグループに対する更新操作の速度が向上し、単一ソケットを使用したノード間通信に対するシステムまたはその他の制限による制約を回避できます。

デフォルトでは、NDB は、ローカルデータ管理 (LDM) スレッドの数またはトランザクションコーディネータ (TC) スレッドの数 (いずれか大きい方) に基づいて、多数のトランスポータを使用するようになりました。デフォルトでは、トランスポータの数はこの数の半分です。ほとんどのワークロードではデフォルトのパフォーマンスが適切で

すが、`NodeGroupTransporters` データノード構成パラメータ (NDB 8.0.20 でも導入されています) を LDM スレッドの最大数または TC スレッドの数まで設定することで、各ノードグループで使用されるトランスポータの数を調整できます。0 に設定すると、トランスポータの数は LDM スレッドの数と同じになります。

- `ndb_restore`: 主キースキーマの変更。 NDB 8.0.21 (以降) は、`--allow-pk-changes` オプションを指定して実行されたときに `ndb_restore` を使用して NDB ネイティブバックアップを復元するときに、ソーステーブルとターゲットテーブルに対して異なる主キー定義をサポートします。元の主キーを構成するカラム数の増減がサポートされています。

主キーが追加のカラムで拡張されている場合、追加されたカラムは `NOT NULL` として定義する必要があり、そのようなカラムの値はバックアップの取得中に変更できません。一部のアプリケーションでは、更新時に行のすべてのカラム値が設定されるため、すべての値が実際に変更されているかどうかにかかわらず、主キーに追加されるカラムの値が変更されていなくてもリストア操作が失敗する可能性があります。 `--ignore-extended-pk-updates` オプションを使用して、この動作をオーバーライドできます (こちらも NDB 8.0.21 で追加)。この場合、このような値が変更されていないことを確認する必要があります。

このカラムがテーブルの一部であるかどうかにかかわらず、テーブルの主キーからカラムを削除できます。

詳細は、`ndb_restore` の `--allow-pk-changes` オプションの説明を参照してください。

- `ndb_restore` とのバックアップのマージ。 場合によっては、NDB Cluster の異なるインスタンス (すべて同じスキーマを使用) に格納されていたデータを単一のターゲット NDB Cluster に統合することが望ましいことがあります。これは、NDB 8.0.21 で `--restore-data` とともに追加された `--remap-column` オプションを使用して、`ndb_mgm` クライアント (セクション 23.5.8.2 「NDB Cluster 管理クライアントを使用したバックアップの作成」を参照) で作成されたバックアップを使用し、`ndb_restore` でそれらを復元するときにサポートされるようになりました (さらに、必要に応じて互換性のある追加オプションもあります)。 `--remap-column` を使用すると、主キー値と一意キー値がソースクラスタ間で重複しているケースを処理できます。また、外部キーなどのテーブル間の他の関係を持すするために、ターゲットクラスタで重複しないようにする必要があります。

`--remap-column` は、その引数として `db.tbl.col:fn:args` という形式の文字列を取ります。ここで、`db`、`tbl` および `col` はそれぞれ、データベース、テーブルおよびカラムの名前、`fn` は再マッピング関数の名前、`args` は `fn` への 1 つ以上の引数です。デフォルト値はありません。関数名として `offset` のみがサポートされ、バックアップからターゲットテーブルに挿入するときにカラムの値に適用される整数オフセットとして `args` が使用されます。このカラムは、`INT` または `BIGINT` のいずれかである必要があります。オフセット値の許容範囲は、そのタイプの符号付きバージョンと同じです (これにより、必要に応じてオフセットを負にできます)。

`ndb_restore` の同じ起動で新しいオプションを複数回使用できるため、同じテーブルまたは異なるテーブル (あるいはその両方) の新しい値を複数のカラムに再マップできます。オフセット値は、オプションのすべてのインスタンスで同じである必要はありません。

さらに、NDB 8.0.21 から、`ndb_desc` 用に 2 つの新しいオプションが提供されています:

- `--auto-inc` (ショートフォーム `-a`): テーブルに `AUTO_INCREMENT` カラムがある場合は、次の自動インクリメント値が出力に含まれます。
- `--context` (ショートフォーム `-x`): スキーマ、データベース名、テーブル名、内部 ID など、テーブルに関する追加情報を提供します。

詳細および例については、`--remap-column` オプションの説明を参照してください。

- スレッド改善の送信。 NDB 8.0.20 の時点では、各送信スレッドがトランスポータのサブセットへの送信を処理するようになり、各ブロックスレッドは 1 つの送信スレッドのみを支援するようになったため、より多くの送信スレッドが生成されるため、パフォーマンスとデータノードのスケラビリティが向上します。
- `SpinMethod` を使用した適応スピン制御。 NDB 8.0.20 では、`SpinMethod` データノードパラメータを使用して、それをサポートするプラットフォームで適応 CPU スピンを設定するための単純なインタフェースが導入されています。このパラメータには 4 つの設定があり、それぞれ静的スピン用、コストベースの適応スピン用、待機時間最適化された適応スピン用、および各スレッドが独自の CPU を持つデータベースマシン用に最適化された適応スピン用です。これらの各設定により、データノードは 1 つまたは複数のスピンパラメータに事前定義された値のセットを

使用し、適応スピンを有効にし、スピントイミングを設定し、スピンオーバーヘッドを特定のシナリオに応じて設定するため、一般的なユースケースでこれらを直接設定する必要がなくなります。

スピン動作を微調整するには、既存の `SchedulerSpinTimer` データノード構成パラメータおよび `ndb_mgm` クライアントの次の `DUMP` コマンドを使用して、これらのスピンパラメータと追加のスピンパラメータを直接設定することもできます:

- `DUMP 104000 (SetSchedulerSpinTimerAll)`: すべてのスレッドのスピン時間を設定
- `DUMP 104001 (SetSchedulerSpinTimerThread)`: 指定されたスレッドのスピン時間を設定
- `DUMP 104002 (SetAllowedSpinOverhead)`: 1 単位の待機時間を取得できる CPU 時間の単位数としてスピンオーバーヘッドを設定
- `DUMP 104003 (SetSpintimePerCall)`: スピンするコールの時間を設定
- `DUMP 104004 (EnableAdaptiveSpinning)`: 患者のスピンを有効または無効にします

NDB 8.0.20 は、指定された TCP 接続に対してスピンする時間を設定する新しい TCP 構成パラメータ `TcpSpinTime` も追加します。

`ndb_top` ツールは、スレッドごとにスピン時間情報を提供するように拡張されています。

詳細は、`SpinMethod` パラメータ、リストされている `DUMP` コマンドおよび [セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」](#) の説明を参照してください。

- ディスクデータとクラスタの再起動。 NDB 8.0.21 以降では、クラスタを最初に再起動すると、テーブルスペースやログファイルグループなどのすべてのディスクデータオブジェクト (これらのオブジェクトに関連付けられたデータファイルや undo ログファイルを含む) が強制的に削除されます。

詳細は、[セクション23.5.10 「NDB Cluster ディスクデータテーブル」](#) を参照してください。

- ディスクデータエクステント割当て。 NDB 8.0.20 以降では、データファイル内のエクステントの割り当ては、特定のテーブルスペースで使用されるすべてのデータファイル間でラウンドロビン方式で行われます。これは、ディスクデータストレージに複数のストレージデバイスが使用されている場合に、データの分散を改善することが期待されます。

詳細は、[セクション23.5.10.1 「NDB Cluster ディスクデータオブジェクト」](#) を参照してください。

- `--ndb-log-fail-terminate` オプション。 NDB 8.0.21 以降では、すべての行イベントを完全にログに記録できない場合は常に SQL ノードを終了させることができます。これを行うには、`--ndb-log-fail-terminate` オプションを指定して `mysqld` を起動します。
- `AllowUnresolvedHostNames` パラメータ。 デフォルトでは、管理ノードは、グローバル構成ファイルに存在するホスト名を解決できない場合に起動を拒否します (Kubernetes などの一部の環境で問題が発生する可能性があります)。 NDB 8.0.22 以降では、クラスタグローバル会議ファイル (`config.ini` ファイル) の `[tcp default]` セクションで `AllowUnresolvedHostNames` を `true` に設定することによって、この動作をオーバーライドできます。これにより、そのようなエラーはかわりに警告として処理され、`ndb_mgmd` の起動を続行できます
- BLOB 書込みパフォーマンスの向上。 NDB 8.0.22 は、同じ行で複数の BLOB カラムを変更するとき、または同じステートメントで BLOB カラムを含む複数の行を変更するとき、これらの変更を適用するときに SQL またはほかの API ノードとデータノードの間で必要なラウンドトリップ数を減らすことによって、より効率的なバッチ処理を可能にする多数の改善を実装しています。したがって、多くの `INSERT`、`UPDATE` および `DELETE` ステートメントのパフォーマンスを向上させることができます。このようなステートメントの例を次に示します。 `table` は、1 つ以上の BLOB カラムを含む NDB テーブルです:
 - `INSERT INTO table VALUES ROW(1, blob_value1, blob_value2, ...)`、つまり、BLOB カラムを含む行の挿入
 - `INSERT INTO table VALUES ROW(1, blob_value1), ROW(2, blob_value2), ROW(3, blob_value3), ...`、つまり、1 つ以上の BLOB カラムを含む複数行の挿入
 - `UPDATE table SET blob_column1 = blob_value1, blob_column2 = blob_value2, ...`

- `UPDATE table SET blob_column = blob_value WHERE primary_key_column IN (value_list)`:主キーカラムが BLOB 型ではない場合
- `DELETE FROM table WHERE primary_key_column = value`:主キーカラムが BLOB 型ではない場合
- `DELETE FROM table WHERE primary_key_column IN (value_list)`:主キーカラムが BLOB 型ではない場合

他の SQL ステートメントでも、これらの改善のメリットが得られる場合があります。これには、`LOAD DATA INFILE` および `CREATE TABLE ... SELECT ...` が含まれます。また、`table` がステートメントの実行前に NDB 以外のストレージエンジンを使用する `ALTER TABLE table ENGINE = NDB` も、より効率的に実行できます。

この拡張機能は、MySQL タイプ `BLOB`, `MEDIUMBLOB`, `LOB`, `TEXT`, `MEDIUMTEXT` および `LONGTEXT` のカラムに影響するステートメントに適用されます。`TINYBLOB` カラムまたは `TINYTEXT` カラム (あるいはその両方) のみを更新するステートメントは、この作業の影響を受けないため、パフォーマンスは変更されません。

バッチ処理を分割するテーブル Blob カラムのスキャンが必要なため、一部の SQL ステートメントのパフォーマンスはこの拡張によって大幅に向上しません。このようなステートメントには、次に示すタイプのステートメントが含まれます:

- `SELECT FROM table [WHERE key_column IN (blob_value_list)]`: Blob 型を使用する主キーまたは一意キーカラムを照合することで行が選択されます
- `UPDATE table SET blob_column = blob_value WHERE condition`:一意の値に依存しない `condition` を使用
- 一意の値に依存しない `condition` を使用して、BLOB カラムを含む行を削除する `DELETE FROM table WHERE condition`
- ステートメントの実行前に NDB ストレージエンジンをすでに使用しており、ステートメントの実行前または実行後 (あるいはその両方) に行に BLOB カラムが含まれているテーブルに対するコピー `ALTER TABLE` ステートメント

この改善を最大限に活用するために、`mysqld` の `--ndb-batch-size` および `--ndb-blob-write-batch-bytes` オプションに使用される値を増やして、BLOB の変更に必要なラウンドトリップの回数を最小限に抑えることができます。レプリケーションの場合は、`slave_allow_batching` システム変数を有効にすることもお薦めします。これにより、エポクトランザクションを適用するためにレプリカクラスターで必要なラウンドトリップの回数が最小限に抑えられます。

- Node.js の更新。 NDB 8.0.22 以降、NDB Adapter for Node.js はバージョン 12.18.3 を使用して構築され、そのバージョン (またはそれ以降のバージョンの Node.js) のみがサポートされるようになりました。
- 暗号化バックアップ。 NDB 8.0.22 では、AES-256-CBC を使用して暗号化されたバックアップファイルのサポートが追加されています。これは、未承認のパーティによってアクセスされたバックアップからのデータの回復から保護することを目的としています。暗号化すると、バックアップデータはユーザー指定のパスワードによって保護されます。パスワードには、`!`, `'`, `"`, `$`, `%`, `\` および `^` 以外の印刷可能な ASCII 文字の範囲から 256 文字までの任意の文字列を指定できます。指定された NDB Cluster バックアップの暗号化に使用されるパスワードの保持は、ユーザーまたはアプリケーションが実行する必要があります。NDB はパスワードを保存しません。パスワードは空にできますが、これはお薦めしません。

NDB Cluster のバックアップを作成するときは、管理クライアントの `START BACKUP` コマンドで `ENCRYPT PASSWORD=password` を使用して暗号化できます。MGM API のユーザーは、`ndb_mgm_start_backup4()` をコールして暗号化バックアップを開始することもできます。

暗号化バックアップからリストアするには、`--decrypt` および `--backup-password` オプションを指定して `ndb_restore` を使用します。両方のオプションと、暗号化されていない場合に同じバックアップをリストアするために必要なその他のオプションが必要です。`ndb_print_backup_file` および `ndbxfrm` は、それぞれ `-P password` および `--decrypt-password= password` を使用して、暗号化されたファイルを読み取ることもできます。

暗号化または復号化のためにパスワードを指定する場合は、すべて引用符で囲む必要があります。パスワードを区切るには、一重引用符または二重引用符を使用できます。

8.0.22 リリースで NDB Cluster ディストリビューションに追加された `ndbxfrm` ユーティリティを使用して、既存のバックアップファイルを暗号化できます。このプログラムは、暗号化されたバックアップファイルの復号化にも

使用できます。さらに、`CompressedBackup` 構成パラメータが 1 に設定されている場合、`ndbxfm` は NDB Cluster がバックアップの作成に使用するのと同じ方法を使用して、バックアップファイルを圧縮し、圧縮されたバックアップファイルを解凍できます。

この機能は、クラスタグローバル構成ファイルの`[ndbd default]`セクションで `RequireEncryptedBackup=1` を設定することで、暗号化バックアップを強制する機能も実装します。これが行われると、`ndb_mgm` クライアントは暗号化されていないバックアップの実行を拒否します。

- IPv6 のサポート。 NDB 8.0.22 以降、IPv6 アドレス指定は、管理ノードおよびデータノードへの接続でサポートされます。これには、管理ノードと SQL ノードを持つデータノード間の接続が含まれます。クラスタを構成する場合、数値の IPv6 アドレス、IPv6 アドレスに解決されるホスト名、またはその両方を使用できます。

IPv6 アドレス指定が機能するには、クラスタがデプロイされているオペレーティングプラットフォームおよびネットワークで IPv6 がサポートされている必要があります。IPv4 アドレス指定を使用する場合と同様に、IPv6 アドレスへのホスト名解決はオペレーティングプラットフォームによって提供される必要があります。

IPv4 アドレス指定は、引き続き NDB でサポートされます。IPv4 アドレスと IPv6 アドレスを同時に使用することはお薦めしませんが、次の場合に機能するようにできます:

- 管理ノードが IPv6 で構成され、データノードが `config.ini` ファイル内の IPv4 アドレスで構成されている場合: これは、`--bind-address` が `mgmd` で使用されておらず、`--ndb-connectstring` が管理ノードの IPv4 アドレスに設定された状態でデータノードが起動された場合に機能します。
- 管理ノードが IPv4 で構成され、データノードが `config.ini` の IPv6 アドレスで構成されている場合: 他の場合と同様に、これは、`--bind-address` が `mgmd` に渡されず、`--ndb-connectstring` が管理ノードの IPv6 アドレスに設定された状態でデータノードが起動された場合に機能します。

これらのケースは、`ndb_mgmd` がデフォルトではどの IP アドレスにもバインドしないため機能します。

IPv6 アドレス指定をサポートしていないバージョンの NDB からそのバージョンへのアップグレードを実行するには、ネットワークで IPv4 および IPv6 がサポートされている場合、まずソフトウェアのアップグレードを実行します。これが完了したら、`config.ini` ファイルで使用されている IPv4 アドレスを IPv6 アドレスで更新できます。その後、構成の変更を有効にし、IPv6 アドレスを使用してクラスタを起動するには、クラスタのシステム再起動を実行する必要があります。

- 自動インストーラの非推奨および削除。 MySQL NDB Cluster Auto-Installer web ベースのインストールツール (`ndb_setup.py`) は NDB 8.0.22 では非推奨であり、NDB 8.0.23 以降では削除されます。サポートされなくなりました。
- `ndbmemcache` の非推奨および削除。 `ndbmemcache` はサポートされなくなりました。`ndbmemcache` は NDB 8.0.22 で非推奨になり、NDB 8.0.23 で削除されました。
- `ndbinfo backup_id` テーブル。 NDB 8.0.24 は、`backup_id` テーブルを `ndbinfo` 情報データベースに追加します。これは、`ndb_select_all` を使用して内部 `SYSTAB_0` 入力可能オブジェクトの内容をダンプすることで、この情報を取得するための代替手段として機能します。これはエラーが発生しやすく、実行に非常に時間がかかります。

このテーブルには、`START BACKUP` 管理クライアントコマンドを使用して取得されたクラスタの最新のバックアップの ID を含む単一の列および行があります。このクラスタのバックアップが見つからない場合、テーブルには列の値が 0 の単一行が含まれます。

- テーブルのパーティション化の拡張機能。 NDB 8.0.23 には、テーブルのパーティションおよびフラグメントを処理するための新しい方法が導入されており、これにより、REDO ログ部分の数に関係なく、特定のデータノードのローカルデータマネージャー (LDM) の数を決定できます。これは、LDM の数が大きく変動する可能性があることを意味します。NDB は、NDB 8.0.23 にも実装されている `ClassicFragmentation` データノード構成パラメータが `false` に設定されている場合にこの方法を使用できます。この場合、LDM の数はデータノードごとのテーブルに作成するパーティションの数を決定するために使用されなくなり、`PartitionsPerNode` パラメータの値 (NDB 8.0.23 にも導入されています) によってこの数が代わりに決定され、テーブルに使用されるフラグメントの数の計算にも使用されます。

`ClassicFragmentation` のデフォルト値が `true` の場合、LDM の数を使用する従来の方法を使用して、テーブルに必要なフラグメントの数が決定されます。

詳細は、[マルチスレッドの構成パラメータ \(ndbmtid\)](#) で前述の新しいパラメータの説明を参照してください。

- 用語の更新. NDB 8.0.23 は、MySQL 8.0.21 および NDB 8.0.21 で開始された作業に合わせて、次に示す用語の多くの変更を実装します:
 - システム変数 `ndb_slave_conflict_role` は非推奨になりました。 `ndb_conflict_role` に置き換えられています。
 - 多くの NDB ステータス変数は非推奨です。 これらの変数とその置換を次のテーブルに示します:

表 23.1 非推奨の NDB ステータス変数とその代替

非推奨の変数	置換
<code>Ndb_api_adaptive_send_deferred_count_slave</code>	<code>Ndb_api_adaptive_send_deferred_count_replica</code>
<code>Ndb_api_adaptive_send_forced_count_slave</code>	<code>Ndb_api_adaptive_send_forced_count_replica</code>
<code>Ndb_api_adaptive_send_unforced_count_slave</code>	<code>Ndb_api_adaptive_send_unforced_count_replica</code>
<code>Ndb_api_bytes_received_count_slave</code>	<code>Ndb_api_bytes_received_count_replica</code>
<code>Ndb_api_bytes_sent_count_slave</code>	<code>Ndb_api_bytes_sent_count_replica</code>
<code>Ndb_api_pk_op_count_slave</code>	<code>Ndb_api_pk_op_count_replica</code>
<code>Ndb_api_pruned_scan_count_slave</code>	<code>Ndb_api_pruned_scan_count_replica</code>
<code>Ndb_api_range_scan_count_slave</code>	<code>Ndb_api_range_scan_count_replica</code>
<code>Ndb_api_read_row_count_slave</code>	<code>Ndb_api_read_row_count_replica</code>
<code>Ndb_api_scan_batch_count_slave</code>	<code>Ndb_api_scan_batch_count_replica</code>
<code>Ndb_api_table_scan_count_slave</code>	<code>Ndb_api_table_scan_count_replica</code>
<code>Ndb_api_trans_abort_count_slave</code>	<code>Ndb_api_trans_abort_count_replica</code>
<code>Ndb_api_trans_close_count_slave</code>	<code>Ndb_api_trans_close_count_replica</code>
<code>Ndb_api_trans_commit_count_slave</code>	<code>Ndb_api_trans_commit_count_replica</code>
<code>Ndb_api_trans_local_read_row_count_slave</code>	<code>Ndb_api_trans_local_read_row_count_replica</code>
<code>Ndb_api_trans_start_count_slave</code>	<code>Ndb_api_trans_start_count_replica</code>
<code>Ndb_api_uk_op_count_slave</code>	<code>Ndb_api_uk_op_count_replica</code>
<code>Ndb_api_wait_exec_complete_count_slave</code>	<code>Ndb_api_wait_exec_complete_count_replica</code>
<code>Ndb_api_wait_meta_request_count_slave</code>	<code>Ndb_api_wait_meta_request_count_replica</code>
<code>Ndb_api_wait_nanos_count_slave</code>	<code>Ndb_api_wait_nanos_count_replica</code>
<code>Ndb_api_wait_scan_result_count_slave</code>	<code>Ndb_api_wait_scan_result_count_replica</code>
<code>Ndb_slave_max_replicated_epoch</code>	<code>Ndb_replica_max_replicated_epoch</code>

非推奨のステータス変数は引き続き `SHOW STATUS` の出力に表示されますが、将来のリリースシリーズでの可用性は保証されないため、アプリケーションは依存しないようにできるかぎり早く更新する必要があります。

- `ndbinfo` `ndbinfo.table_distribution_status` テーブルの `tab_copy_status` カラムに以前表示されていた `ADD_TABLE_MASTER` および `ADD_TABLE_SLAVE` の値は非推奨になりました。これらはそれぞれ、`ADD_TABLE_COORDINATOR` および `ADD_TABLE_PARTICIPANT` の値に置き換えられます。
- `ndb_restore` などの一部の NDB クライアントプログラムおよびユーティリティプログラムの `--help` 出力が変更されました。
- ThreadConfig の拡張機能. NDB 8.0.23 の時点で、`ThreadConfig` パラメータの構成可能性は、次に示す 2 つの新しいスレッドタイプで拡張されています:
 - `query`: クエリースレッドは、ローカルデータマネージャ (LDM) 内で動作し、フラグメントをクエリーします。クエリースレッドは、リカバリスレッドとしても機能します。クエリースレッドの数は、LDM スレッドの数の 0、1、2、または 3 倍にする必要があります。0 (`ThreadConfig` を使用しない場合、または `AutomaticThreadConfig` が有効になっている場合、デフォルト) を指定すると、LDM は NDB 8.0.23 の前と同じように動作します。

- **recover**: リカバリスレッドは、ローカルチェックポイントからデータを取得します。 そのように指定されたリカバリスレッドは、クエリースレッドとして機能しません。

次のいずれかの方法で、既存の **main** スレッドと **rep** スレッドを組み合わせることもできます:

- これらの引数のいずれかを 0 に設定して、単一スレッドにします。 これを実行すると、結果として生成される結合スレッドが **ndbinfo.threads** テーブルに **main_rep** という名前が表示されます。
- **ldm** と **tc** の両方を 0 に設定し、**recv** を 1 に設定することで、**recv** スレッドと連携します。 この場合、結合されたスレッドの名前は **main_rep_recv** です。

また、既存のスレッドタイプの最大数が増加しました。 クエリースレッドおよびリカバリスレッドの最大値を含む新しい最大値を次に示します:

- LDM: 332
- Query: 332
- Recovery: 332
- TC: 128
- Receive: 64
- Send: 64
- メイン: 2

その他のスレッドタイプの最大値は変更されません。

詳細は、**ThreadConfig** パラメータおよび **ndbinfo.threads** テーブルの説明を参照してください。

また、このタスクに関連して実行された作業の結果、**NDB** では、32 を超えるブロックスレッドを使用する場合に **mutex** を使用してジョブバッファを保護するようになりました。 これにより、パフォーマンスがわずかに低下する可能性があります (ほとんどの場合、1 から 2 パーセント)、非常に大規模な構成に必要なメモリー量も大幅に減少します。 たとえば、**NDB 8.0.23** より前の 2 G バイトのジョブバッファメモリーを使用する 64 スレッドの設定では、**NDB 8.0.23** 以降では約 1 G バイトのみが必要です。 このテストでは、非常に複雑なクエリーの実行で全体的に 5 パーセントの順序で改善されました。

- **ndbmtid** スレッドの自動構成。 **NDB 8.0.23** 以降では、**ndbmtid** 構成パラメータ **AutomaticThreadConfig** を使用して、マルチスレッドデータノードのスレッドの自動構成を使用できます。 このパラメータが 1 に設定されている場合、**NDB** は、アプリケーションで使用可能なプロセッサの数に基づいて、前の項目で説明した新しい **query** および **recover** スレッドタイプを含む、すべてのスレッドでサポートされるスレッドタイプのスレッド割当てを自動的に設定します。 システムがプロセッサの数を制限しない場合は、必要に応じて **NumCPUs** を設定することで制限できます (こちらも **NDB 8.0.23** で追加)。 それ以外の場合、自動スレッド構成は最大 1024 個の CPU に対応します。

自動スレッド構成は、**config.ini** で **ThreadConfig** または **MaxNoOfExecutionThreads** に設定された値に関係なく行われます。 つまり、これらのパラメータのいずれも設定する必要はありません。

さらに、**NDB 8.0.23** は、ハードウェアと CPU の可用性に関する情報を提供する多くの新しい **ndbinfo** 情報データベーステーブル、および **NDB** データノードによる CPU 使用率を実装しています。 これらのテーブルは次のとおりです:

- **cpudata**
- **cpudata_1sec**
- **cpudata_20sec**
- **cpudata_50ms**
- **cpuinfo**

- [hwinfo](#)

NDB Cluster でサポートされているすべてのプラットフォームでこれらのテーブルの一部を使用できるわけではありません。詳細は、それらの個々の説明を参照してください。

- NDB データベースオブジェクトの階層ビュー。 NDB 8.0.24 の [ndbinfo](#) 情報データベースに追加された [dict_obj_tree](#) テーブルは、次のような多くの NDB データベースオブジェクトの階層およびツリー形式のビューを提供できます:

- テーブルおよび関連するインデックス
- テーブルスペースおよび関連するデータファイル
- ログファイルグループおよび関連する UNDO ログファイル

詳細および例については、[セクション23.5.14.22「ndbinfo dict_obj_tree テーブル」](#)を参照してください。

MySQL Cluster Manager 1.4.8 は NDB Cluster 8.0 の実験的なサポートも提供します。MySQL Cluster Manager には、多くの複雑な NDB Cluster 管理タスクを簡略化できる高度なコマンド行インタフェースがあります。詳しくは [MySQL Cluster Manager 1.4.8 User Manual](#), をご覧ください。

23.1.5 NDB 8.0 で追加、非推奨または削除されたオプション、変数、およびパラメータ

- [導入されたパラメータ: NDB 8.0](#)
- [非推奨となったパラメータ: NDB 8.0](#)
- [削除されたパラメータ: NDB 8.0](#)
- [導入されたオプションおよび変数: NDB 8.0](#)
- [非推奨となったオプションおよび変数: NDB 8.0](#)
- [削除されたオプションおよび変数: NDB 8.0](#)

次のいくつかのセクションでは、NDB 8.0 に追加、非推奨になった、または NDB 8.0 から削除された [NDB ノード構成パラメータ](#) および NDB 固有の [mysqld オプション](#) と変数について説明します。

導入されたパラメータ: NDB 8.0

次のノード構成パラメータが追加されました: NDB 8.0.

- [AllowUnresolvedHostNames](#): false (デフォルト) の場合、管理ノードによるホスト名の解決に失敗すると致命的なエラーが発生します。true の場合、未解決のホスト名は警告としてのみ報告されます。追加されたバージョン: NDB 8.0.22.
- [AutomaticThreadConfig](#): 自動スレッド構成を使用し、ThreadConfig および MaxNoOfExecutionThreads の設定をオーバーライド。追加されたバージョン: NDB 8.0.23.
- [ClassicFragmentation](#): true の場合、従来のテーブル断片化を使用します。false に設定すると、LDM 間でのテーブルフラグメントの柔軟な分散が有効になります。追加されたバージョン: NDB 8.0.23.
- [DiskDataUsingSameDisk](#): 「ディスクデータ」テーブルスペースが別の物理ディスクにある場合は false に設定。追加されたバージョン: NDB 8.0.19.
- [MaxDiskDataLatency](#): トランザクションの中断を開始するまでのディスクアクセスの最大許容平均レイテンシ (ミリ秒)。追加されたバージョン: NDB 8.0.19.
- [NodeGroupTransporters](#): 同じノードグループ内のノード間で使用するトランスポータの数。追加されたバージョン: NDB 8.0.20.
- [NumCPUs](#): AutomaticThreadConfig で使用する CPU の数の指定。追加されたバージョン: NDB 8.0.23.

- **PartitionsPerNode**: 各データノードで作成されるテーブルパーティションの数を決定します。ClassicFragmentation が有効な場合は使用されません。追加されたバージョン: NDB 8.0.23.
- **RequireEncryptedBackup**: バックアップを暗号化する必要があるかどうか (1 = 暗号化が必要、それ以外の場合は 0)。追加されたバージョン: NDB 8.0.22.
- **ReservedConcurrentIndexOperations**: 1 つのデータノード上に専用のリソースを持つ同時インデックス操作の数。追加されたバージョン: NDB 8.0.16.
- **ReservedConcurrentOperations**: 1 つのデータノード上のトランザクションコーディネータに専用のリソースを持つ同時操作の数。追加されたバージョン: NDB 8.0.16.
- **ReservedConcurrentScans**: 1 つのデータノード上で専用のリソースを持つ同時スキャンの数。追加されたバージョン: NDB 8.0.16.
- **ReservedConcurrentTransactions**: 1 つのデータノード上に専用のリソースを持つ同時トランザクションの数。追加されたバージョン: NDB 8.0.16.
- **ReservedFiredTriggers**: 1 つのデータノード上に専用のリソースを持つトリガーの数。追加されたバージョン: NDB 8.0.16.
- **ReservedLocalScans**: 1 つのデータノード上に専用のリソースを持つ同時フラグメントスキャンの数。追加されたバージョン: NDB 8.0.16.
- **ReservedTransactionBufferMemory**: 各データノードに割り当てられたキーおよび属性データの動的バッファ領域 (バイト)。追加されたバージョン: NDB 8.0.16.
- **SpinMethod**: データノードで使用されるスピン方法を決定します。詳細は、ドキュメントを参照してください。追加されたバージョン: NDB 8.0.20.
- **TcpSpinTime**: 受信時にスリープするまでのスピン時間。追加されたバージョン: NDB 8.0.20.
- **TransactionMemory**: 各データノード上のトランザクションに割り当てられたメモリー。追加されたバージョン: NDB 8.0.19.

非推奨となったパラメータ: NDB 8.0

次のノード構成パラメータが非推奨になりました: NDB 8.0.

- **BatchSizePerLocalScan**: 保留ロックがあるスキャンのロックレコード数の計算に使用されます。非推奨になったバージョン: NDB 8.0.19.
- **MaxNoOfConcurrentIndexOperations**: 1 つのデータノードで同時に実行できるインデックス操作の合計数。非推奨になったバージョン: NDB 8.0.19.
- **MaxNoOfConcurrentTransactions**: このデータノードで同時に実行されるトランザクションの最大数。同時に実行できるトランザクションの合計数は、この値にクラスター内のデータノードの数を掛けた数です。非推奨になったバージョン: NDB 8.0.19.
- **MaxNoOfFiredTriggers**: 1 つのデータノード上で同時に起動できるトリガーの総数。非推奨になったバージョン: NDB 8.0.19.
- **MaxNoOfLocalOperations**: このデータノードに定義されている操作レコードの最大数。非推奨になったバージョン: NDB 8.0.19.
- **MaxNoOfLocalScans**: このデータノードで並列に実行されるフラグメントスキャンの最大数。非推奨になったバージョン: NDB 8.0.19.
- **ReservedTransactionBufferMemory**: 各データノードに割り当てられたキーおよび属性データの動的バッファ領域 (バイト)。非推奨になったバージョン: NDB 8.0.19.

削除されたパラメータ: NDB 8.0

ノード構成パラメータは削除されていません: NDB 8.0.

導入されたオプションおよび変数: NDB 8.0

次のシステム変数、ステータス変数、およびサーバーオプションが追加されました: NDB 8.0.

- [Ndb_api_adaptive_send_deferred_count_replica](#): このレプリカによって実際に送信されていない適応送信コールの数. 追加されたバージョン: NDB 8.0.23.
- [Ndb_api_adaptive_send_forced_count_replica](#): このレプリカによって送信された強制送信セットを含む適応送信の数. 追加されたバージョン: NDB 8.0.23.
- [Ndb_api_adaptive_send_unforced_count_replica](#): このレプリカによって強制送信されていない適応送信の数. 追加されたバージョン: NDB 8.0.23.
- [Ndb_api_bytes_received_count_replica](#): このレプリカによってデータノードから受信されたデータの量 (バイト). 追加されたバージョン: NDB 8.0.23.
- [Ndb_api_bytes_sent_count_replica](#): このレプリカによってデータノードに送信されたデータの数量 (バイト). 追加されたバージョン: NDB 8.0.23.
- [Ndb_api_pk_op_count_replica](#): このレプリカによる主キーに基づく、または主キーを使用する操作の数. 追加されたバージョン: NDB 8.0.23.
- [Ndb_api_pruned_scan_count_replica](#): このレプリカによって 1 つのパーティションにプルーニングされたスキャンの数. 追加されたバージョン: NDB 8.0.23.
- [Ndb_api_range_scan_count_replica](#): このレプリカによって開始されたレンジスキャンの数. 追加されたバージョン: NDB 8.0.23.
- [Ndb_api_read_row_count_replica](#): このレプリカによって読み取られた行の合計数. 追加されたバージョン: NDB 8.0.23.
- [Ndb_api_scan_batch_count_replica](#): このレプリカによって受信された行のバッチ数. 追加されたバージョン: NDB 8.0.23.
- [Ndb_api_table_scan_count_replica](#): このレプリカによって開始されたテーブルスキャンの数 (内部テーブルのスキャンを含む). 追加されたバージョン: NDB 8.0.23.
- [Ndb_api_trans_abort_count_replica](#): このレプリカによって中断されたトランザクションの数. 追加されたバージョン: NDB 8.0.23.
- [Ndb_api_trans_close_count_replica](#): このレプリカによって中断されたトランザクションの数 (TransCommitCount と TransAbortCount の合計より大きい可能性があります). 追加されたバージョン: NDB 8.0.23.
- [Ndb_api_trans_commit_count_replica](#): このレプリカによってコミットされたトランザクションの数. 追加されたバージョン: NDB 8.0.23.
- [Ndb_api_trans_local_read_row_count_replica](#): このレプリカによって読み取られた行の合計数. 追加されたバージョン: NDB 8.0.23.
- [Ndb_api_trans_start_count_replica](#): このレプリカによって開始されたトランザクションの数. 追加されたバージョン: NDB 8.0.23.
- [Ndb_api_uk_op_count_replica](#): このレプリカによる一意キーに基づく、または一意キーを使用する操作の数. 追加されたバージョン: NDB 8.0.23.
- [Ndb_api_wait_exec_complete_count_replica](#): このレプリカによる操作実行の完了を待機中にスレッドがブロックされた回数. 追加されたバージョン: NDB 8.0.23.
- [Ndb_api_wait_meta_request_count_replica](#): このレプリカによるメタデータベースのシグナルを待機してスレッドがブロックされた回数. 追加されたバージョン: NDB 8.0.23.
- [Ndb_api_wait_nanos_count_replica](#): このレプリカによるデータノードからの何らかのタイプのシグナルの待機に費やされた合計時間 (ナノ秒). 追加されたバージョン: NDB 8.0.23.
- [Ndb_api_wait_scan_result_count_replica](#): このレプリカによるスキャンベースのシグナルの待機中にスレッドがブロックされた回数. 追加されたバージョン: NDB 8.0.23.

- **Ndb_config_generation**: クラスターの現在の構成の生成番号. 追加されたバージョン: NDB 8.0.24.
- **Ndb_metadata_blacklist_size**: NDB binlog スレッドが同期に失敗した NDB メタデータオブジェクトの数. NDB 8.0.22 では `Ndb_metadata_excluded_count` として名前が変更されました. 追加されたバージョン: NDB 8.0.18.
- **Ndb_metadata_detected_count**: NDB メタデータ変更モニタースレッドが変更を検出した回数. 追加されたバージョン: NDB 8.0.16.
- **Ndb_metadata_excluded_count**: NDB binlog スレッドが同期に失敗した NDB メタデータオブジェクトの数. 追加されたバージョン: NDB 8.0.22.
- **Ndb_metadata_synced_count**: 同期された NDB メタデータオブジェクトの数. 追加されたバージョン: NDB 8.0.18.
- **Ndb_trans_hint_count_session**: このセッションで開始されたヒントを使用するトランザクションの数. 追加されたバージョン: NDB 8.0.17.
- **ndb-log-fail-terminate**: 見つかったすべての行イベントの完全なロギングが不可能な場合は `mysqld` プロセスを終了. 追加されたバージョン: NDB 8.0.21.
- **ndb-schema-dist-timeout**: スキーマ分散中にタイムアウトを検出するまでの待機時間. 追加されたバージョン: NDB 8.0.17.
- **ndb_conflict_role**: レプリカが競合検出および解決で果たす役割. 値は PRIMARY、SECONDARY、PASS、NONE (デフォルト) のいずれかです。レプリケーション SQL スレッドが停止している場合にのみ変更できます。詳細は、ドキュメントを参照してください. 追加されたバージョン: NDB 8.0.23.
- **ndb_dbg_check_shares**: 言語共有のチェック (デバッグビルドのみ). 追加されたバージョン: NDB 8.0.13.
- **ndb_metadata_check**: MySQL データディクショナリに関する NDB メタデータ変更の自動検出を有効にします。デフォルトで有効になっています. 追加されたバージョン: NDB 8.0.16.
- **ndb_metadata_check_interval**: MySQL データディクショナリに関する NDB メタデータ変更のチェックを実行する間隔 (秒). 追加されたバージョン: NDB 8.0.16.
- **ndb_metadata_sync**: NDB ディクショナリと MySQL データディクショナリ間のすべての変更の即時同期をトリガーします。これにより、`ndb_metadata_check` 値と `ndb_metadata_check_interval` 値は無視されます。同期が完了すると `false` にリセットされます. 追加されたバージョン: NDB 8.0.19.
- **ndb_schema_dist_lock_wait_timeout**: エラーが返される前にロックを待機するスキーマ配布中の時間. 追加されたバージョン: NDB 8.0.18.
- **ndb_schema_dist_timeout**: スキーマ分散中にタイムアウトを検出するまでの待機時間. 追加されたバージョン: NDB 8.0.16.
- **ndb_schema_dist_upgrade_allowed**: NDB への接続時にスキーマ配布テーブルのアップグレードを許可. 追加されたバージョン: NDB 8.0.17.
- **ndbinfo**: `ndbinfo` プラグインの有効化 (サポートされている場合). 追加されたバージョン: NDB 8.0.13.

非推奨となったオプションおよび変数: NDB 8.0

次のシステム変数、ステータス変数、およびオプションが非推奨になりました: NDB 8.0.

- **Ndb_api_adaptive_send_deferred_count_slave**: このレプリカによって実際に送信されていない適応送信コールの数. 非推奨になったバージョン: NDB 8.0.23.
- **Ndb_api_adaptive_send_forced_count_slave**: このレプリカによって送信された強制送信セットを含む適応送信の数. 非推奨になったバージョン: NDB 8.0.23.
- **Ndb_api_adaptive_send_unforced_count_slave**: このレプリカによって強制送信されていない適応送信の数. 非推奨になったバージョン: NDB 8.0.23.
- **Ndb_api_bytes_received_count_slave**: このレプリカによってデータノードから受信されたデータの量 (バイト). 非推奨になったバージョン: NDB 8.0.23.
- **Ndb_api_bytes_sent_count_slave**: このレプリカによってデータノードに送信されたデータの数量 (バイト). 非推奨になったバージョン: NDB 8.0.23.

- `Ndb_api_pk_op_count_slave`: このレプリカによる主キーに基づく、または主キーを使用する操作の数。非推奨になったバージョン: NDB 8.0.23.
- `Ndb_api_pruned_scan_count_slave`: このレプリカによって 1 つのパーティションにプルーニングされたスキャンの数。非推奨になったバージョン: NDB 8.0.23.
- `Ndb_api_range_scan_count_slave`: このレプリカによって開始されたレンジスキャンの数。非推奨になったバージョン: NDB 8.0.23.
- `Ndb_api_read_row_count_slave`: このレプリカによって読み取られた行の合計数。非推奨になったバージョン: NDB 8.0.23.
- `Ndb_api_scan_batch_count_slave`: このレプリカによって受信された行のバッチ数。非推奨になったバージョン: NDB 8.0.23.
- `Ndb_api_table_scan_count_slave`: このレプリカによって開始されたテーブルスキャンの数 (内部テーブルのスキャンを含む)。非推奨になったバージョン: NDB 8.0.23.
- `Ndb_api_trans_abort_count_slave`: このレプリカによって中断されたトランザクションの数。非推奨になったバージョン: NDB 8.0.23.
- `Ndb_api_trans_close_count_slave`: このレプリカによって中断されたトランザクションの数 (TransCommitCount と TransAbortCount の合計より大きい可能性があります)。非推奨になったバージョン: NDB 8.0.23.
- `Ndb_api_trans_commit_count_slave`: このレプリカによってコミットされたトランザクションの数。非推奨になったバージョン: NDB 8.0.23.
- `Ndb_api_trans_local_read_row_count_slave`: このレプリカによって読み取られた行の合計数。非推奨になったバージョン: NDB 8.0.23.
- `Ndb_api_trans_start_count_slave`: このレプリカによって開始されたトランザクションの数。非推奨になったバージョン: NDB 8.0.23.
- `Ndb_api_uk_op_count_slave`: このレプリカによる一意キーに基づく、または一意キーを使用する操作の数。非推奨になったバージョン: NDB 8.0.23.
- `Ndb_api_wait_exec_complete_count_slave`: このレプリカによる操作実行の完了を待機中にスレッドがブロックされた回数。非推奨になったバージョン: NDB 8.0.23.
- `Ndb_api_wait_meta_request_count_slave`: このレプリカによるメタデータベースのシグナルを待機してスレッドがブロックされた回数。非推奨になったバージョン: NDB 8.0.23.
- `Ndb_api_wait_nanos_count_slave`: このレプリカによるデータノードからの何らかのタイプのシグナルの待機に費やされた合計時間 (ナノ秒)。非推奨になったバージョン: NDB 8.0.23.
- `Ndb_api_wait_scan_result_count_slave`: このレプリカによるスキャンベースのシグナルの待機中にスレッドがブロックされた回数。非推奨になったバージョン: NDB 8.0.23.
- `Ndb_metadata_blacklist_size`: NDB binlog スレッドが同期に失敗した NDB メタデータオブジェクトの数。NDB 8.0.22 では `Ndb_metadata_excluded_count` として名前が変更されました。非推奨になったバージョン: NDB 8.0.21.
- `Ndb_replica_max_replicated_epoch`: このレプリカで最近コミットされた NDB エポック。この値が `Ndb_conflict_last_conflict_epoch` 以上の場合、競合はまだ検出されていません。非推奨になったバージョン: NDB 8.0.23.
- `ndb_slave_conflict_role`: レプリカが競合検出および解決で果たす役割。値は PRIMARY、SECONDARY、PASS、NONE (デフォルト) のいずれかです。レプリケーション SQL スレッドが停止している場合にのみ変更できます。詳細は、ドキュメントを参照してください。非推奨になったバージョン: NDB 8.0.23.

削除されたオプションおよび変数: NDB 8.0

次のシステム変数、ステータス変数、およびオプションが削除されました: NDB 8.0.

- `Ndb_metadata_blacklist_size`: NDB binlog スレッドが同期に失敗した NDB メタデータオブジェクトの数。NDB 8.0.22 では `Ndb_metadata_excluded_count` として名前が変更されました。削除されたバージョン: NDB 8.0.22.

23.1.6 MySQL Server NDB Cluster と比較した InnoDB の使用

MySQL Server では、ストレージエンジンに数多くの選択肢があります。NDB と InnoDB の両方がトランザクション MySQL ストレージエンジンとして機能できるため、MySQL Server のユーザーが NDB Cluster に関心を持つことがあります。NDB は、MySQL 8.0 のデフォルトの InnoDB ストレージエンジンの代替候補またはアップグレード候補とみなされています。NDB と InnoDB は共通の特性を共有しますが、アーキテクチャーと実装に違いがあるため、一部の既存の MySQL Server アプリケーションと使用シナリオは NDB Cluster には適していませんが、すべてに適しているわけではありません。

このセクションでは、NDB 8.0 で使用される NDB ストレージエンジンのいくつかの特性と、MySQL 8.0 で使用される InnoDB について説明し、比較します。次のいくつかのセクションでは、技術的な比較を示します。多くの場合、NDB Cluster を使用するタイミングと場所に関する決定は、すべての要因を考慮してケース単位で行う必要があります。このドキュメントでは、すべての有意義な使用シナリオに固有の情報を提供することは範囲外ですが、InnoDB バックエンドではなく、NDB の一部の共通タイプのアプリケーションの相対的な適合性に関する一般的なガイダンスも提供しようとしています。

NDB Cluster 8.0 は、InnoDB 1.1 のサポートを含む、MySQL 8.0 に基づく `mysqld` を使用します。NDB Cluster で InnoDB テーブルを使用することは可能ですが、このようなテーブルはクラスタ化されません。NDB Cluster 8.0 配布のプログラムまたはライブラリを MySQL Server 8.0 とともに使用することも、その逆もできません。

NDB Cluster または MySQL Server (InnoDB ストレージエンジンを使用している可能性が高い) のいずれかで実行できる共通ビジネスアプリケーションの一部のタイプにも当てはまりますが、いくつかの重要なアーキテクチャーと実装の違いがあります。セクション 23.1.6.1 「NDB および InnoDB ストレージエンジンの違い」には、これらの違いのサマリーが表示されます。これらの違いのため、一部の使用シナリオは明らかにどちらか一方のエンジンに適しています。セクション 23.1.6.2 「NDB と InnoDB のワークロード」を参照してください。これは、NDB または InnoDB とともに使用するのが適切なアプリケーションのタイプにも影響を与えます。それぞれが一般的なタイプのデータベースアプリケーションでの使用にどの程度適合するかの比較については、セクション 23.1.6.3 「NDB および InnoDB の機能使用のサマリー」を参照してください。

NDB および MEMORY ストレージエンジンの相対的な特徴については、MEMORY または NDB Cluster を使用する場合を参照してください。

MySQL ストレージエンジンに関する追加情報は、第 16 章「代替ストレージエンジン」を参照してください。

23.1.6.1 NDB および InnoDB ストレージエンジンの違い

NDB ストレージエンジンは、分散型のシェアードナッシングアーキテクチャーを使用して実装されます。これにより、多くの点で InnoDB と動作が異なります。NDB の操作に慣れていないユーザーにとっては、NDB の持つ分散型の性質によって、トランザクション、外部キー、テーブルの制限、およびその他の特性に関して予期しない動作が発生する可能性があります。次の表にそれらを示します。

表 23.2 InnoDB ストレージエンジンと NDB ストレージエンジンの違い

機能	InnoDB (MySQL 8.0)	NDB 8.0
MySQL Server のバージョン	8.0	8.0
InnoDB のバージョン	InnoDB 8.0.29	InnoDB 8.0.29
NDB Cluster バージョン	N/A	NDB 8.0.23/8.0.23
ストレージの制限	64TB	128TB
外部キー	はい	はい
トランザクション	すべての標準タイプ	READ COMMITTED
MVCC	はい	いいえ
データ圧縮	はい	いいえ (NDB チェックポイントおよびバックアップファイルは圧縮できません)
大規模な行のサポート (> 14K)	VARBINARY、VARCHAR、BLOB、および TEXT カラムでサポートされます	BLOB および TEXT カラムでのみサポートされます (これらのタイプを使用して非常に大量のデータを格納する)

機能	InnoDB (MySQL 8.0)	NDB 8.0
		と NDB のパフォーマンスが低下する可能性があります)
レプリケーションのサポート	MySQL レプリケーションを使用した非同期および準同期レプリケーション (MySQL Group Replication)	NDB Cluster 内の自動同期レプリケーション。MySQL レプリケーションを使用した NDB Cluster 間の非同期レプリケーション (準同期レプリケーションはサポートされていません)
読み取り操作のスケールアウト	はい (MySQL レプリケーション)	はい (NDB Cluster での自動パーティション分割、NDB Cluster レプリケーション)
書き込み操作のスケールアウト	アプリケーションレベルのパーティション化 (シャーディング) が必要です	はい (NDB Cluster の自動パーティション分割はアプリケーションに対して透過的です)
高可用性 (HA)	組み込み (InnoDB クラスタから)	はい (99.999% の稼働時間を実現するように設計されています)
ノードの障害リカバリとフェイルオーバー	MySQL Group Replication から	Automatic (NDB アーキテクチャーの鍵要素)
ノードの障害リカバリにかかる時間	30 秒以上	通常は 1 秒未満
リアルタイムのパフォーマンス	いいえ	はい
インメモリーテーブル	いいえ	はい (一部のデータはオプションでディスクに格納できます。インメモリーとディスクの両方のデータストレージは永続的です)
ストレージエンジンへの NoSQL アクセス	はい	はい (Memcached、Node.js/JavaScript、Java、JPA、C++、HTTP/REST を含む複数の API)
同時および並列書き込み	はい	同時書き込みのために最適化された最大 48 個のライター
競合の検出および解決 (複数のソース)	はい (MySQL Group Replication)	はい
ハッシュインデックス	いいえ	はい
オンラインのノード追加	MySQL Group Replication を使用した読取り/書き込みレプリカ	はい (すべてのノードタイプ)
オンラインのアップグレード	はい (レプリケーションを使用)	はい
オンラインのスキーマ変更	はい (MySQL 8.0 の一部として)	はい

23.1.6.2 NDB と InnoDB のワークロード

NDB Cluster には、高可用性、高速フェイルオーバー、高スループット、および低待機時間を必要とするアプリケーションの提供に最適な一意の属性の範囲があります。NDB Cluster には、分散アーキテクチャーとマルチノード実装のために、一部のワークロードが適切に実行されないようにする可能性がある特定の制約もあります。データベース駆動型アプリケーションの一般的なワークロードのタイプに関する [NDB](#) ストレージエンジンと [InnoDB](#) ストレージエンジンの動作の主な違いを、次の表にいくつか示します。

表 23.3 InnoDB ストレージエンジンと NDB ストレージエンジンの違いは、一般的なタイプのデータ駆動型アプリケーションワークロードです。

ワークロード	InnoDB	NDB Cluster (NDB)
大容量 OLTP アプリケーション	はい	はい
DSS アプリケーション (データマート、分析)	はい	制限付き (サイズが 3T バイトを超える OLTP データセット間の結合操作は不可)

ワークロード	InnoDB	NDB Cluster (NDB)
カスタムアプリケーション	はい	はい
パッケージ化されたアプリケーション	はい	制限付き (主キーアクセスであるべきです)。NDB Cluster 8.0 は外部キーをサポートしています
ネットワーク内電気通信アプリケーション (HLR、HSS、SDP)	いいえ	はい
セッション管理およびキャッシュ	はい	はい
電子商取引アプリケーション	はい	はい
ユーザープロファイル管理、AAA プロトコル	はい	はい

23.1.6.3 NDB および InnoDB の機能使用のサマリー

InnoDB と NDB の機能に対するアプリケーション機能要件を比較すると、その一部は明らかにどちらか一方のストレージエンジンと高い互換性があります。

次の表に、サポートされるアプリケーション機能を、一般に各機能がより適合するストレージエンジンに従って示します。

表 23.4 各機能が通常適しているストレージエンジンに応じてサポートされているアプリケーション機能

InnoDB がより適合するアプリケーション要件	NDB がより適合するアプリケーション要件
<ul style="list-style-type: none"> 外部キー <p style="text-align: center;">注記</p> <p style="text-align: center;">NDB Cluster 8.0 は外部キーをサポート</p> <ul style="list-style-type: none"> 完全なテーブルスキャン 非常に大規模なデータベース、行、またはトランザクション READ COMMITTED 以外のトランザクション 	<ul style="list-style-type: none"> 書き込みのスケールング 99.999% の稼働時間 オンラインのノード追加とオンラインのスキーマ操作 複数の SQL および NoSQL API (NDB Cluster APIs: Overview and Conceptsを参照してください) リアルタイムのパフォーマンス BLOB カラムの限定的な使用 外部キーがサポートされますが、外部キーの使用は高スループットでのパフォーマンスに影響する可能性があります。

23.1.7 NDB Cluster の既知の制限事項

以降のセクションでは、NDB Cluster の現在のリリースにおける既知の制限について、MyISAM および InnoDB ストレージエンジンを使用するときに使用可能な機能と比較して説明します。 <http://bugs.mysql.com> の MySQL バグデータベースで「クラスタ」カテゴリを確認すると、 <http://bugs.mysql.com> の MySQL バグデータベースにある「MySQL Server:」の下の次のカテゴリで既知のバグが見つかります。これは、NDB Cluster の今後のリリースで修正する予定です:

- NDB Cluster
- Cluster Direct API (NDBAPI)
- Cluster Disk Data
- Cluster Replication
- ClusterJ

この情報は、規定された条件に関して完結するように意図されています。不具合が発生した場合は、[セクション 1.6 「質問またはバグをレポートする方法」](#) に示す手順を使用して MySQL バグデータベースにそれらを報告できます。NDB Cluster 8.0 で修正を計画していない問題はすべて、リストに追加されます。

NDB Cluster 8.0 で解決された以前のリリースの問題のリストについては、[セクション23.1.7.11「前 NDB Cluster 8.0 で解決される NDB Cluster の問題」](#) を参照してください。

注記

NDB Cluster レプリケーションに固有の制限事項およびその他の問題については、[セクション23.6.3「NDB Cluster レプリケーションの既知の問題」](#) を参照してください。

23.1.7.1 NDB Cluster の SQL 構文に準拠していません

次のリストに示すように、特定の MySQL 機能に関連する SQL ステートメントの中には、NDB テーブルで使ったときにエラーを生成するものがあります。

- 一時テーブル. 一時テーブルはサポートされません。NDB ストレージエンジンを使用する一時テーブルを作成しようとしたり、NDB を使用するように既存の一時テーブルを変更しようとしたりすると、失敗して次のエラーが発生します:`Table storage engine 'ndbcluster' does not support the create option 'TEMPORARY'`.
- NDB テーブルのインデックスとキー. 「NDB Cluster」テーブルのキーおよびインデックスには、次の制限事項があります:
 - カラム幅. 3072 バイトを超える幅の NDB テーブルカラムに対するインデックスを作成しようとすると、成功しますが、インデックスに実際に使用されるのは最初の 3072 バイトだけです。このような場合は、警告 `Specified key was too long; max key length is 3072 bytes` が発行され、`SHOW CREATE TABLE` ステートメントではインデックスの長さが 3072 と表示されます。
 - TEXT および BLOB カラム. TEXT または BLOB データ型のどちらかを使用する NDB テーブルカラムに対するインデックスは作成できません。
 - FULLTEXT インデックス. NDB ストレージエンジンは、MyISAM および InnoDB テーブルでのみ可能な FULLTEXT インデックスをサポートしていません。
ただし、NDB テーブルの VARCHAR カラムに対するインデックスは作成できます。
 - USING HASH キーと NULL. 一意キーおよび主キーで NULL 可能カラムを使用すると、これらのカラムを使用するクエリーは完全なテーブルスキャンとして処理されます。この問題を回避するには、カラムを NOT NULL にするか、USING HASH オプションを指定せずにインデックスを再作成します。
 - プリフィクス. プリフィクスインデックスはありません。インデックスはカラム全体にのみ付けることができます。(NDB のカラムインデックスのサイズは、このセクションで前述したように、カラムの幅 (バイト単位) と常に同じ (最大 3072 バイトまで) です。追加情報については、[セクション23.1.7.6「NDB Cluster でサポートされない機能または欠落している機能」](#) を参照してください。)
 - BIT カラム. BIT カラムを主キー、一意キー、またはインデックスにすることはできません。また、複合の主キー、一意キー、またはインデックスの一部にすることもできません。
 - AUTO_INCREMENT カラム. ほかの MySQL ストレージエンジンと同様に、NDB ストレージエンジンはテーブルあたり最大 1 つの AUTO_INCREMENT カラムを処理できます。ただし、明示的な主キーを持たない NDB テーブルの場合は、AUTO_INCREMENT カラムが自動的に定義され、「非表示」主キーとして使用されます。このため、明示的な AUTO_INCREMENT カラムを持つテーブルは、PRIMARY KEY オプションを同時に使用してカラムを宣言しないかぎり定義できません。テーブルの主キーでない AUTO_INCREMENT カラムを持つテーブルを作成しようとして、NDB ストレージエンジンを使用すると、失敗してエラーが発生します。
- 外部キーに関する制限. NDB 8.0 での外部キー制約のサポートは、InnoDB で提供されるものと同等ですが、次の制限事項があります:
 - 外部キーとして参照されるすべてのカラムは、それがテーブルの主キーでない場合、明示的な一意キーを必要とします。
 - 参照先が親テーブルの主キーである場合、ON UPDATE CASCADE はサポートされません。

これは、主キーの更新が古い行 (古い主キーを含む) の削除と新しい行 (新しい主キーを持つ) の挿入として実装されているためです。これは NDB カーネルには表示されないため、これらの 2 つの行は同じであると表示されるため、この更新をカスケードする必要があることを知る方法はありません。

- NDB 8.0.16 の時点: **TEXT** 型または **BLOB** 型のカラムが子テーブルに含まれている場合、**ON DELETE CASCADE** はサポートされません。(Bug #89511、Bug #27484882)
- **SET DEFAULT** はサポートされません。(InnoDB でもサポートされません。)
- **NO ACTION** キーワードは受け入れられますが、**RESTRICT** として扱われます。MySQL 8.0 では、標準の SQL キーワードである **NO ACTION** がデフォルトです。(InnoDB でも同じです。)
- NDB Cluster の以前のバージョンでは、別のテーブル内のインデックスを参照する外部キーを持つテーブルを作成するとき、適切なエラーが常に内部的に返されなかったために、インデックス内のカラムの順序が一致しなかった場合でも、外部キーを作成できるように見えることがありました。この問題を部分的に修正すると、ほとんどの場合に内部的に使用されるエラーが改善されましたが、親インデックスが一意インデックスである場合でも、この状況が発生する可能性があります。(Bug #18094360)

詳細は、[セクション13.1.20.5「FOREIGN KEY の制約」](#) および [セクション1.7.3.2「FOREIGN KEY の制約」](#) を参照してください。

- NDB Cluster およびジオメトリデータタイプ。
NDB テーブルでは、ジオメトリデータ型 (**WKT** および **WKB**) がサポートされます。ただし、空間インデックスはサポートされません。
- 文字セットとバイナリログファイル。現在、**ndb_apply_status** および **ndb_binlog_index** テーブルは **latin1** (ASCII) 文字セットを使用して作成されています。バイナリログの名前はこのテーブルに記録されるため、ラテン語以外の文字を使用する名前が付けられたバイナリログファイルはこれらのテーブルで正しく参照されません。これは既知の問題であり、現在修正中です。(Bug #50226)

この問題を回避するには、バイナリログファイルの名前を付けるときや、**--basedir**、**--log-bin**、または **--log-bin-index** オプションを設定するときに、Latin-1 文字のみを使用してください。

- ユーザー定義のパーティション化を使用した「NDB の作成」テーブル。NDB Cluster でのユーザー定義パーティショニングに対する のサポートは、**[LINEAR] KEY** パーティショニングに制限されています。**CREATE TABLE** ステートメントで **ENGINE=NDB** または **ENGINE=NDBCLUSTER** とともにほかのパーティショニングタイプを使用すると、エラーが発生します。

この制限はオーバーライドできますが、本番設定での使用はサポートされていません。詳細は、[ユーザー定義のパーティション分割と NDB ストレージエンジン \(NDB Cluster\)](#) を参照してください。

デフォルトのパーティション化スキーム。「すべての NDB Cluster」テーブルは、デフォルトでは、テーブル主キーをパーティション化キーとして使用して **KEY** によってパーティション化されます。テーブルに明示的に設定された主キーがない場合は、代わりに **NDB ストレージエンジン** によって自動的に作成された「隠し」主キーが使用されます。これらおよび関連する問題の追加情報については、[セクション24.2.5「KEY パーティショニング」](#) を参照してください。

ユーザーパーティション化された **NDBCLUSTER** テーブルが次の 2 つの要件のどちらかまたは両方を満たさない原因となる **CREATE TABLE** および **ALTER TABLE** ステートメントは許可されず、失敗してエラーが発生します。

1. テーブルに明示的な主キーが存在する必要があります。
2. テーブルのパーティショニング式に指定されたすべてのカラムが主キーの一部である必要があります。

例外。空のカラムリストを使用して (つまり、**PARTITION BY [LINEAR] KEY()** を使用して) ユーザーパーティション化された **NDBCLUSTER** テーブルを作成する場合は、明示的な主キーは必要ありません。

NDBCLUSTER テーブルの最大パーティション数。ユーザー定義のパーティション化を使用したときに **NDBCLUSTER** テーブルに定義できるパーティションの最大数は、ノードグループあたり 8 個です。(NDB Cluster ノードグループの詳細は、[セクション23.1.2「NDB Cluster ノード、ノードグループ、フラグメントレプリカ、およびパーティション」](#) を参照してください。)

DROP PARTITION の未サポート。**ALTER TABLE ... DROP PARTITION** を使用して NDB テーブルからパーティションを削除することはできません。**ALTER TABLE** に対するその他のパーティション化拡張機能 **-ADD PARTITION**、**REORGANIZE PARTITION**、および **COALESCE PARTITION** は NDB テーブルでサポートされま

ですが、コピーを使用するため、最適化されません。 [セクション24.3.1「RANGE および LIST パーティションの管理」](#) および [セクション13.1.9「ALTER TABLE ステートメント」](#) を参照してください。

- 行ベースのレプリケーション。
NDB Cluster で行ベースレプリケーションを使用する場合、バイナリロギングを無効にすることはできません。つまり、NDB ストレージエンジンは `sql_log_bin` の値を無視します。
- JSON データ型。 MySQL JSON データ型は、NDB 8.0 で提供される `mysqld` 内の NDB テーブルでサポートされます。

NDB テーブルには、最大 3 つの JSON カラムを含めることができます。

NDB API には、単に BLOB データとして表示される JSON データを操作するための特別なプロビジョニングはありません。 JSON としてのデータの処理は、アプリケーションで実行する必要があります。

23.1.7.2 NDB Cluster と標準の MySQL 制限の制限と相違点

このセクションでは、NDB Cluster で検出された制限のうち、標準 MySQL で検出された制限とは異なる制限または見つからない制限を一覧表示します。

メモリーの使用量とリカバリ。ほかのストレージエンジンと同様、NDB テーブルにデータを挿入したときに消費されたメモリーは、削除したときに自動的にリカバリされません。代わりに、次のルールが適用されます。

- NDB テーブルに対して `DELETE` ステートメントを実行すると、削除された行で以前使用されていたメモリーが同じテーブルでの挿入にかぎって再利用可能になります。ただし、`OPTIMIZE TABLE` を実行すると、このメモリーの一般的な再利用が可能になります。

クラスタのローリング再起動が行われた場合も、削除された行で使用されていたメモリーが解放されます。 [セクション23.5.5「NDB Cluster のローリング再起動の実行」](#) を参照してください。

- NDB テーブルに対して `DROP TABLE` または `TRUNCATE TABLE` 操作を実行すると、このテーブルで使用されていたメモリーが解放され、任意の NDB テーブルで (同じテーブルでも別の NDB テーブルでも) 再利用可能になります。

注記

`TRUNCATE TABLE` によってテーブルが削除され、再作成されることを思い出してください。 [セクション13.1.37「TRUNCATE TABLE ステートメント」](#) を参照してください。

- クラスタの構成によって課される制限。
多くの構成可能なハード制限がありますが、クラスタ内の利用可能なメインメモリーによって制限が設定されます。 [セクション23.3.3「NDB Cluster 構成ファイル」](#) の構成パラメータの完全なリストを参照してください。ほとんどの構成パラメータはオンラインでアップグレードできます。これらのハード制限には次のようなものがあります。

- データベースのメモリーサイズとインデックスのメモリーサイズ (それぞれ `DataMemory` と `IndexMemory`)。

`DataMemory` は 32K バイトのページとして割り当てられます。各 `DataMemory` ページは、使用されたときに特定のテーブルに割り当てられます。割り当てられたあとは、テーブルを削除した場合を除いてこのメモリーを解放できません。

詳細は、[セクション23.3.3.6「NDB Cluster データノードの定義」](#) を参照してください。

- 1 つのトランザクションで実行できる操作の最大数は、構成パラメータ `MaxNoOfConcurrentOperations` および `MaxNoOfLocalOperations` を使用して設定されます。

注記

一括ロード、`TRUNCATE TABLE`、および `ALTER TABLE` は、複数のトランザクションを実行することによって特別なケースとして扱われるため、この制限が適用されません。

- テーブルとインデックスに関するさまざまな制限。たとえば、クラスタ内の順序付けされたインデックスの最大数は `MaxNoOfOrderedIndexes` によって決定され、テーブルあたりの順序付けされたインデックスの最大数は 16 です。
- ノードとデータオブジェクトの最大数。クラスタノードとメタデータオブジェクトの数には、次の制限が適用されます。
 - NDB 8.0.18 の時点では、データノードの最大数は 145 です。(以前は 48 でした。)データノードは 1 から 144 までの (これらを含む) 範囲のノード ID を持つ必要があります。(NDB 8.0.17 以前のリリースでは、これは 1 から 48 でした。)
- 管理ノードおよび API ノードでは、1 から 255 の範囲のノード ID を使用できます。
- NDB Cluster 内のノードの合計最大数は 255 です。この数には、すべての SQL ノード (MySQL Server)、API ノード (MySQL Server 以外のクラスタにアクセスするアプリケーション)、データノード、および管理サーバーが含まれます。
- NDB Cluster の現在のバージョンのメタデータオブジェクトの最大数は 20320 です。この制限はハードコーディングされています。

詳細は、[セクション 23.1.7.11 「前 NDB Cluster 8.0 で解決される NDB Cluster の問題」](#) を参照してください。

23.1.7.3 NDB Cluster でのトランザクション処理に関する制限

NDB Cluster には、トランザクションの処理に関していくつかの制限があります。これらには、次のものが含まれません。

- トランザクション分離レベル。 `NDBCLUSTER` ストレージエンジンは `READ COMMITTED` トランザクション分離レベルのみをサポートします。(たとえば、`InnoDB` は `READ COMMITTED`、`READ UNCOMMITTED`、`REPEATABLE READ`、および `SERIALIZABLE` をサポートします。) `NDB` は `READ COMMITTED` を行単位で実装する必要があることに注意してください。読取りリクエストが行を格納しているデータノードに到着すると、その時点で最後にコミットされたバージョンの行が返されます。

コミットされていないデータが返されることはありませんが、複数の行を変更するトランザクションが同じ行を読み取るトランザクションと同時にコミットする場合、特定の行読取りリクエストを他のトランザクションのコミットの前または後に処理できるため、読取りを実行するトランザクションは、これらの間で異なる行に対して値の前、値の後、またはその両方を監視できます。

特定のトランザクションが変更前または変更後の値のみを読み取るようにするには、`SELECT ... LOCK IN SHARE MODE` を使用して行ロックを設定します。このような場合、ロックは、所有しているトランザクションがコミットされるまで保持されます。行ロックを使用すると、次の問題が発生することもあります：

- ロック待機タイムアウトエラーの頻度の向上と同時実行性の低下
- コミットフェーズを必要とする読取りによるトランザクション処理オーバーヘッドの増加
- 使用可能な同時ロック数を使い果たす可能性があり、`MaxNoOfConcurrentOperations` によって制限されています

`NDB` では、`LOCK IN SHARE MODE` や `FOR UPDATE` などの修飾子を使用しないが、すべての読取りに `READ COMMITTED` が使用されます。`LOCK IN SHARE MODE` では共有行ロックが使用され、`FOR UPDATE` では排他行ロックが使用されます。一意キー読取りでは、自己一貫性読取りを保証するために `NDB` によってロックが自動的にアップグレードされます。`BLOB` 読取りでは、一貫性のために追加ロックも使用されます。

`NDB` のトランザクション分離レベルのクラスタ実装が `NDB` データベースのバックアップおよび復元に与える影響については、[セクション 23.5.8.4 「NDB Cluster バックアップのトラブルシューティング」](#) を参照してください。

- トランザクションと `BLOB` または `TEXT` カラム。 `NDBCLUSTER` は、MySQL から認識できるテーブルに MySQL の `BLOB` または `TEXT` データ型のいずれかを使用するカラム値の一部のみを格納します。`BLOB` または `TEXT` の残りの部分は、MySQL からアクセスできない別の内部テーブルに格納されます。これに関連して、これらの型のカラムを含むテーブルに対して `SELECT` ステートメントを実行するときに常に注意すべき問題が 2 つ発生します。

1. 「NDB Cluster」テーブルからの **SELECT** の場合: **SELECT** に **BLOB** または **TEXT** カラムが含まれている場合、**READ COMMITTED** トランザクション分離レベルは読取りロック付きの読取りに変換されます。これは一貫性を保証するために行われます。
2. 一意キーのルックアップを使用して **BLOB** または **TEXT** データ型のいずれかを使用するカラムを取得し、1 つのトランザクション内で実行される **SELECT** の場合は、共有読み取りロックがトランザクションの期間中 (つまり、トランザクションがコミットまたは中止されるまで) そのテーブルに保持されます。

インデックスまたはテーブルスキャンを使用するクエリーでは、**BLOB** または **TEXT** カラムを含む **NDB** テーブルが対象であっても、この問題は発生しません。

たとえば、次の **CREATE TABLE** ステートメントによって定義されたテーブル **t** について考えます。

```
CREATE TABLE t (  
  a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,  
  b INT NOT NULL,  
  c INT NOT NULL,  
  d TEXT,  
  INDEX i(b),  
  UNIQUE KEY u(c)  
) ENGINE = NDB;
```

t で次のクエリーを実行すると、一意キー検索が使用されるため、共有読み取りロックが発生します:

```
SELECT * FROM t WHERE c = 1;
```

しかし、ここに示す 4 つのクエリーでは、いずれも共有読み取りロックは発生しません。

```
SELECT * FROM t WHERE b = 1;
```

```
SELECT * FROM t WHERE d = '1';
```

```
SELECT * FROM t;
```

```
SELECT b,c WHERE a = 1;
```

その理由は、これら 4 つのクエリーのうち、1 つ目はインデックススキャンを使用し、2 つ目と 3 つ目はテーブルスキャンを使用し、4 つ目は (主キールックアップを使用していますが) **BLOB** または **TEXT** カラムの値を取得していないためです。

BLOB または **TEXT** カラムを取得する一意キールックアップを使用するクエリーを回避するか、このようなクエリーを回避できない場合はトランザクションのコミットをできるだけあとで行うようにすると、共有読み取りロックによる問題を最小限に抑えることができます。

- 一意キー参照とトランザクション分離。一意インデックスは、内部的にメンテナンスされる非表示インデックステーブルを使用して **NDB** に実装されます。一意インデックスを使用してユーザーが作成した **NDB** テーブルにアクセスすると、非表示のインデックステーブルが最初に読み取られて主キーが検索され、次にユーザーが作成したテーブルの読取りに使用されます。この二重読取り操作中にインデックスが変更されないように、非表示のインデックステーブルで検出された行はロックされます。ユーザー作成の **NDB** テーブルの一意インデックスによって参照される行が更新されると、非表示のインデックステーブルには、更新が実行されるトランザクションによる排他ロックが適用されます。つまり、同じ (ユーザーが作成した) **NDB** テーブルに対する読取り操作は、更新が完了するまで待機する必要があります。これは、読取り操作のトランザクションレベルが **READ COMMITTED** の場合でも当てはまります。

潜在的なブロッキング読取りをバイパスするために使用できる回避策の 1 つは、読取りの実行時に SQL ノードが一意のインデックスを無視するように強制することです。これを行うには、テーブルを読み取る **SELECT** ステートメントの一部として **IGNORE INDEX** インデックスヒントを使用します (セクション 8.9.4 「インデックスヒント」を参照)。MySQL サーバーでは、**NDB** で作成された一意のインデックスごとにシャドウ順序付きインデックスが作成されるため、かわりに順序付きインデックスを読み取ることができ、一意のインデックスアクセスロックが回避

されます。結果の読取りは、主キーによってコミットされた読取りと同じ一貫性があり、行の読取り時に最後にコミットされた値を返します。

順序付けされたインデックスを介して読み取ると、クラスタ内のリソースの使用効率が低下し、待機時間が長くなる可能性があります。

一意の値ではなく範囲をクエリーすることで、アクセスに一意のインデックスを使用しないようにすることもできます。

- **ロールバック。** 部分的なトランザクションおよびトランザクションの部分的なロールバックはありません。重複キーまたは同様のエラーが発生すると、トランザクション全体がロールバックされます。

この動作は、個々のステートメントをロールバックできる **InnoDB** などのほかのトランザクション対応のストレージエンジンと異なります。

- **トランザクションとメモリー使用量。**
この章のほかの場所で説明されているように、NDB Cluster は大きなトランザクションを適切に処理しません。多数の操作を含む単一の大きなトランザクションを試みるよりも、少数の操作で多数の小さなトランザクションを実行することをお勧めします。特に考慮すべき点は、大規模なトランザクションが非常に大量のメモリーを必要とすることです。このため、次のリストで説明するように、多数の MySQL ステートメントのトランザクション動作が影響を受けます：
 - **TRUNCATE TABLE** は、NDB テーブルに対して使用した場合、トランザクションになりません。 **TRUNCATE TABLE** でテーブルを空にできない場合は、成功するまでこれを再実行する必要があります。
 - **DELETE FROM (WHERE 句が内場合も含む)** は、トランザクションになります。テーブルに非常に多くの行が含まれる場合は、複数の **DELETE FROM ... LIMIT ...** ステートメントを使用して削除操作を「ひとまとめに」すると、パフォーマンスが向上することがあります。テーブルを空にすることが目的である場合は、代わりに **TRUNCATE TABLE** を使用することをお勧めします。
 - **LOAD DATA** ステートメント。 NDB テーブルで使用する場合、**LOAD DATA** はトランザクション対応ではありません。

重要

LOAD DATA ステートメントを実行すると、NDB エンジン是不規則な間隔でコミットを実行し、通信ネットワークの使用率を向上させます。このようなコミットが発生するタイミングを事前に知ることはできません。

- **ALTER TABLE とトランザクション。** **ALTER TABLE** の一部として NDB テーブルをコピーする場合、コピーの作成はトランザクションになりません。(いずれにしても、コピーが削除されたときにこの操作はロールバックされます。)
- **トランザクションと COUNT() 関数。** NDB Cluster レプリケーションを使用する場合、レプリカ上の **COUNT()** 関数のトランザクションの一貫性を保証することはできません。つまり、ソースで単一トランザクション内のテーブルの行数を変更する一連のステートメント (**INSERT**、**DELETE**、またはその両方) を実行する場合、レプリカで **SELECT COUNT(*) FROM table** クエリーを実行すると中間結果が生成されることがあります。これは、**SELECT COUNT(...)** がダーティ読み取りを行うために発生し、NDB ストレージエンジンのバグではありません。(詳細は、Bug #31321 を参照してください。)

23.1.7.4 NDB Cluster のエラー処理

ノードの起動、停止、および再起動によって、トランザクションが失敗する原因となる一時的なエラーが発生する場合があります。これらには、次のケースが含まれます。

- **一時エラー。** ノードを最初に起動するときは、エラー 1204 **Temporary failure, distribution changed** および同様の一時エラーが発生する可能性があります。
- **ノード障害によるエラー。** データノードの停止または障害によって、いくつかの異なるノード障害エラーが発生する場合があります。(ただし、クラスタの計画的なシャットダウンを実行したときに中止されるトランザクションは存在しないはずで)

これらのいずれの場合も、生成されたエラーはアプリケーションの内部で処理する必要があります。これは、トランザクションの再試行によって行うべきです。

[セクション23.1.7.2「NDB Cluster と標準の MySQL 制限の制限と相違点」](#)も参照してください。

23.1.7.5 NDB Cluster 内のデータベースオブジェクトに関連付けられる制限

NDBCLUSTER ストレージエンジンを使用するときは、テーブルやインデックスなどの一部のデータベースオブジェクトに関してさまざまな制限があります。

- データベースオブジェクトの数。 単一の NDB Cluster (データベース、テーブル、およびインデックスを含む) 内の all **NDB** データベースオブジェクトの最大数は 20320 に制限されます。
- テーブルあたりの属性数。 特定のテーブルに属することができる属性 (つまり、カラムとインデックス) の最大数は 512 です。
- キーあたりの属性数。 キーあたりの属性の最大数は 32 です。
- 行サイズ。 NDB 8.0.18 以降、いずれかの行の最大許容サイズは 30000 バイトで、NDB 8.0.17 以前のリリースでは、この制限は 14000 バイトです。

BLOB または **TEXT** の各カラムは、この合計に 256 バイトを超える 8 = 264 バイトを占めます。これには **JSON** カラムが含まれます。これらのタイプの詳細は、[文字列型の格納要件](#) および [JSON 記憶域の要件](#) を参照してください。

また、**NDB** テーブルの固定幅カラムの最大オフセットは 8188 バイトです。この制限に違反するテーブルを作成しようとする、NDB エラー 851 「[固定サイズカラムの最大オフセットを超えました](#)」で失敗します。メモリーベースのカラムの場合、**VARCHAR** などの可変幅のカラムタイプを使用するか、カラムを **COLUMN_FORMAT=DYNAMIC** として定義することで、この制限を回避できます。これは、ディスクに格納されているカラムでは機能しません。ディスクベースのカラムの場合、テーブルの作成に使用された **CREATE TABLE** ステートメントで最後に定義されたディスクベースのカラム以外のすべてのカラムの幅の組合せが 8188 バイトを超えず、**CHAR** や **VARCHAR** などの一部のデータ型に対して実行される可能性のある丸めを減らすように、テーブルディスクベースのカラムの順序を変更できます。そうでない場合は、問題のあるカラムの 1 つ以上にメモリーベースの記憶域を使用する必要があります。

- テーブルごとの BIT カラムの記憶域。 特定の **NDB** テーブルで使用されるすべての **BIT** カラムの最大合計幅は 4096 です。
- **FIXED** カラムの格納。 NDB Cluster 8.0 は、**FIXED** カラム内のデータのフラグメントあたり最大 128 TB をサポートします。

23.1.7.6 NDB Cluster でサポートされない機能または欠落している機能

NDB テーブルでは、ほかのストレージエンジンでサポートされるいくつかの機能がサポートされません。NDB Cluster でこれらの機能のいずれかを使用しようとしても、それ自体のエラーは発生しません。ただし、機能がサポートまたは適用されることが予想されるアプリケーションでは、エラーが発生する可能性があります。このような機能を参照するステートメントは、**NDB** によって事実上無視される場合でも、構文上有効である必要があります。

- インデックスのプリフィクス。 インデックスの接頭辞は、**NDB** テーブルではサポートされません。接頭辞が **CREATE TABLE**、**ALTER TABLE**、**CREATE INDEX** などのステートメントのインデックス指定の一部として使用されている場合、その接頭辞は **NDB** によって作成されません。

インデックス接頭辞を含むステートメント、および **NDB** テーブルの作成または変更は、構文的に有効である必要があります。たとえば、次のステートメントは、ストレージエンジンに関係なく、常にエラー 1089 「[接頭辞キーが正しくありません。使用されているキー部分が文字列でないか、使用されている長さがキー部分より長い、ストレージエンジンが一意的な接頭辞キーをサポートしていません](#)」で失敗します:

```
CREATE TABLE t1 (  
  c1 INT NOT NULL,  
  c2 VARCHAR(100),  
  INDEX i1 (c2(500))  
);
```

これは、インデックスにそれ自体より大きい接頭辞がない可能性があるという SQL 構文ルールを考慮して発生します。

- セーブポイントとロールバック。 セーブポイントおよびセーブポイントへのロールバックは、[MyISAM](#) と同様に無視されます。
- コミットの持続性。 ディスクに対する永続的なコミットはありません。 コミットはレプリケートされますが、コミット時にログがディスクにフラッシュされる保証はありません。
- レプリケーション。 ステートメントベースのレプリケーションはサポートされません。 クラスタのレプリケーションを設定するときは、`--binlog-format=ROW` (または `--binlog-format=MIXED`) を使用してください。 詳細は、[セクション23.6「NDB Cluster レプリケーション」](#)を参照してください。

グローバルトランザクション識別子 (GTID) を使用したレプリケーションは NDB Cluster と互換性がなく、NDB Cluster 8.0 ではサポートされません。 NDB Cluster レプリケーションの失敗までの問題を引き起こす可能性が非常に高いため、[NDB ストレージエンジン](#)を使用するときに GTID を有効にしないでください。

準同期レプリケーションは NDB Cluster ではサポートされていません。

- 生成されるカラム。 [NDB ストレージエンジン](#)は、仮想生成カラムのインデックスをサポートしていません。

他のストレージエンジンと同様に、格納された生成カラムにインデックスを作成できますが、[NDB](#) では、生成されたカラムの格納に [DataMemory](#) が使用され、インデックスに [IndexMemory](#) が使用されることに注意する必要があります。 例については、[NDB Cluster での JSON カラムと間接インデックス](#)を参照してください。

NDB Cluster は、格納された生成カラム内の変更をバイナリログに書き込みますが、仮想カラムに対して行われた変更はログに記録しません。 これは、NDB Cluster レプリケーションまたは [NDB](#) とほかの MySQL ストレージエンジン間のレプリケーションには影響しません。

注記

[NDB](#) でのトランザクション処理に関する制限の詳細は、[セクション23.1.7.3「NDB Cluster でのトランザクション処理に関する制限」](#)を参照してください。

23.1.7.7 NDB Cluster のパフォーマンスに関する制限事項

次のパフォーマンスの問題は、NDB Cluster に固有または特に発音されます：

- Range scans。 [NDB ストレージエンジン](#)への順次アクセスによって発生するクエリーのパフォーマンスの問題があります。 多数の範囲スキャンの実行も、[MyISAM](#) や [InnoDB](#) に比べて負荷が高くなります。
- Records in range の信頼性。 [Records in range](#) 統計は使用可能ですが、完全なテストや正式なサポートは行われていません。 このため、場合によっては最適でないクエリー計画が生成されることがあります。 必要な場合は、[USE INDEX](#) または [FORCE INDEX](#) を使用して実行プランを変更できます。 これを行う方法の詳細は、[セクション8.9.4「インデックスヒント」](#)を参照してください。
- 一意のハッシュインデックス。 [USING HASH](#) で作成された一意のハッシュインデックスは、[NULL](#) がキーの一部として指定された場合はテーブルへのアクセスに使用できません。

23.1.7.8 NDB Cluster 専用の問題

[NDB ストレージエンジン](#)に固有の制限は次のとおりです：

- マシンアーキテクチャー。 クラスタ内で使用されるすべてのマシンが同じアーキテクチャーである必要があります。 つまり、ノードをホストするすべてのマシンがビッグエンディアンまたはリトルエンディアンのどちらかである必要があります、両者を組み合わせて使用することはできません。 たとえば、PowerPC で実行されている管理ノードから x86 マシンで実行されているデータノードに指示することはできません。 この制限は、単に [mysql](#) を実行しているマシンや、クラスタ SQL ノードにアクセスしている可能性のあるその他のクライアントには適用されません。
- バイナリロギング。
NDB Cluster には、バイナリロギングに関して次の制限事項があります：
 - `sql_log_bin` は、データ操作では効果がありませんが、スキーマ操作ではサポートされます。
 - NDB Cluster は、[BLOB](#) カラムを持つが主キーを持たないテーブルのバイナリログを生成できません。

- ステートメントを実行する `mysqld` に配置されていないクラスタバイナリログには、次のスキーマ操作だけが記録されます。
 - `CREATE TABLE`
 - `ALTER TABLE`
 - `DROP TABLE`
 - `CREATE DATABASE / CREATE SCHEMA`
 - `DROP DATABASE / DROP SCHEMA`
 - `CREATE TABLESPACE`
 - `ALTER TABLESPACE`
 - `DROP TABLESPACE`
 - `CREATE LOGFILE GROUP`
 - `ALTER LOGFILE GROUP`
 - `DROP LOGFILE GROUP`
- スキーマ操作. スキーマ操作 (DDL ステートメント) は、データノードの再起動中に拒否されます。オンラインアップグレードまたはダウングレードの実行中は、スキーマ操作もサポートされません。
- フラグメントレプリカの数. `NoOfReplicas` データノード構成パラメータによって決定されるフラグメントレプリカ数は、NDB Cluster によって格納されるすべてのデータのコピーの数です。このパラメータを 1 に設定すると、単一のコピーのみが存在することを意味します。この場合、冗長性は提供されず、データノードの損失にはデータの損失が伴います。冗長性を保証し、データノードに障害が発生した場合でもデータを保持するには、このパラメータを 2 に設定します。これは、本番でのデフォルト値および推奨値です。

`NoOfReplicas` を 2 より大きい値 (最大 4) に設定することはサポートされていますが、データ損失から保護する必要はありません。

[セクション23.1.7.10「複数の NDB Cluster ノードに関する制限事項」](#) も参照してください。

23.1.7.9 NDB Cluster ディスクデータストレージに関する制限事項

ディスクデータオブジェクトの最大数と最小数. ディスクデータオブジェクトには、次の最大数および最小数が適用されます。

- テーブルスペースの最大数: 2^{32} (4294967296)
- テーブルスペースあたりのデータファイルの最大数: 2^{16} (65536)
- テーブルスペースのデータファイルのエクステントが取り得る最小および最大サイズは、それぞれ 32K および 2G です。詳細は、[セクション13.1.21「CREATE TABLESPACE ステートメント」](#) を参照してください。

また、「NDB ディスクデータ」テーブルを使用する場合は、データファイルおよびエクステントに関する次の問題に注意する必要があります:

- データファイルは `DataMemory` を使用します。使用方法は、メモリー内データの場合と同じです。
- データファイルはファイル記述子を使用します。データファイルは常に開いていることに注意してください。つまり、ファイル記述子は常に使用され、他のシステムタスクに再利用することはできません。
- エクステントには十分な `DiskPageBufferMemory` が必要です。すべてのエクステントで使用されるすべてのメモリー (エクステント数×エクステントのサイズ) を考慮するために、このパラメータを十分に予約する必要があります。

ディスクデータテーブルとディスクレスモード. ディスクレスモードでクラスタを実行している場合、「ディスクデータ」テーブルの使用はサポートされません。

23.1.7.10 複数の NDB Cluster ノードに関する制限事項

複数の SQL ノード.

NDB Cluster SQL ノードとしての複数の MySQL サーバーの使用に関連する問題は次のとおりで、`NDBCLUSTER` ストレージエンジンに固有です:

- ストアドプログラムが配布されていません。 ストアドプロシージャ、ストアドファンクション、トリガーおよびスケジュール済イベントはすべて、`NDB` ストレージエンジンを使用するテーブルでサポートされていますが、これらはクラスタ SQL ノードとして機能する MySQL Servers 間で自動的に伝播されないため、各 SQL ノードで個別に再作成する必要があります。 [NDB Cluster 内のストアドルーチンとトリガー](#)を参照してください。
- 分散型のテーブルロックがない。 `LOCK TABLES` は、ロックが発行された SQL ノードに対してのみ機能します。クラスタ内のほかの SQL ノードでは、このロックは「認識」されません。これは、操作の一部としてテーブルをロックするステートメントによって発行されたロックにも当てはまりません。(例については、次の項目を参照してください。)
- `ALTER TABLE` 操作。 複数の MySQL サーバー (SQL ノード) が実行されているときは、`ALTER TABLE` によるロックは完全ではありません。(前の項目で説明したように、NDB Cluster は分散テーブルロックをサポートしていません。)

複数の管理ノード.

複数の管理サーバーを使用する場合:

- いずれかの管理サーバーが同じホスト上で実行されている場合は、ノード ID の自動割当てが同じホスト上の複数の管理サーバー間で機能しないため、接続文字列でノードに明示的な ID を指定する必要があります。すべての管理サーバーが異なるホストに存在する場合、これは必要ありません。
- 管理サーバーは起動時に、まず同じ NDB Cluster 内のほかの管理サーバーをチェックし、ほかの管理サーバーへの接続が成功すると、その構成データを使用します。つまり、管理サーバーの `--reload` および `--initial` 起動オプションは、それが実行中の唯一の管理サーバーでないかぎり、無視されます。また、複数の管理ノードを持つ NDB Cluster のローリング再起動を実行する場合、管理サーバーは、それがこの NDB Cluster で実行されている唯一の管理サーバーである場合のみ、独自の構成ファイルを読み取ります。詳細は、[セクション23.5.5「NDB Cluster のローリング再起動の実行」](#)を参照してください。

複数のネットワークアドレス。 1つのデータノードに対する複数のネットワークアドレスはサポートされません。これらを使用すると、問題が発生しやすくなります。データノードに障害が発生すると、SQL ノードはデータノードが停止したことの確認を待機しますが、そのデータノードへの別のルートが開いたままであるため、この確認を受信できません。これにより、クラスタは実質的に操作不能になります。

注記

1つのデータノードで複数のネットワークハードウェアインタフェース (Ethernet カードなど) を使用できますが、これらは同じアドレスにバインドする必要があります。これは、`config.ini` ファイルで接続ごとに複数の `[tcp]` セクションを使用できないことも意味します。詳細は、[セクション23.3.3.10「NDB Cluster TCP/IP 接続」](#)を参照してください。

23.1.7.11 前 NDB Cluster 8.0 で解決される NDB Cluster の問題

NDB Cluster の以前のバージョンに存在していた多くの制限事項および関連する問題が NDB 8.0 で解決されました。これらについて、次のリストで簡単に説明します。

- データベース名とテーブル名。 NDB 8.0.18 より前では、`NDB` ストレージエンジンを使用する場合、データベース名とテーブル名の両方に許可される最大長は 63 文字であり、この制限より長いデータベース名またはテーブル名を使用するステートメントは適切なエラーで失敗しました。NDB 8.0.18 の時点では、この制限はなくなり、`NDB` データベースおよびテーブルの識別子は、ほかの MySQL データベースおよびテーブル名と同様に最大 64 バイトを使用できるようになりました。
- IPv6 のサポート。 NDB 8.0.22 より前では、NDB Cluster 内のノード間の接続に使用されるすべてのネットワークアドレスが、IPv4 アドレスを使用するか、または解決できるようにする必要がありました。NDB 8.0.22 以降、`NDB` は、すべてのタイプのクラスタノード、および NDB API または MGM API を使用するアプリケーションに対して IPv6 アドレスをサポートします。

23.2 NDB Cluster のインストール

このセクションでは、NDB Cluster の計画、インストール、構成、および実行の基本について説明します。 [セクション23.3「NDB Cluster の構成」](#) の例ではさまざまなクラスタ化オプションおよび構成に関する詳細な情報が提供されていますが、ここで説明するガイドラインおよび手順に従った結果は、データの可用性および保護のための最小要件を満たす使用可能な NDB Cluster であるべきです。

NDB Cluster をリリースバージョン間でアップグレードまたはダウングレードする方法については、 [セクション23.2.7「NDB Cluster のアップグレードおよびダウングレード」](#) を参照してください。

このセクションでは、ハードウェアとソフトウェアの要件、ネットワークの問題、NDB Cluster のインストール、基本的な構成の問題、クラスタの起動、停止、および再起動、サンプルデータベースのロード、およびクエリーの実行について説明します。

NDB Cluster は、NDB Cluster ディストリビューションの一部として、web ベースのグラフィカルインストーラである NDB Cluster Auto-Installer (現在は非推奨) も提供します。 Auto-Installer を使用すると、1 台以上のホストコンピュータで NDB Cluster の基本的なインストールとセットアップを実行できます。 詳しくは [セクション23.2.8「NDB Cluster Auto-Installer \(サポートされなくなりました\)」](#) をご覧ください。

仮定. 以降のセクションでは、クラスタの物理的構成とネットワーク構成に関していくつかの仮定を立てています。 これらの仮定については、次のいくつかの段落で説明します。

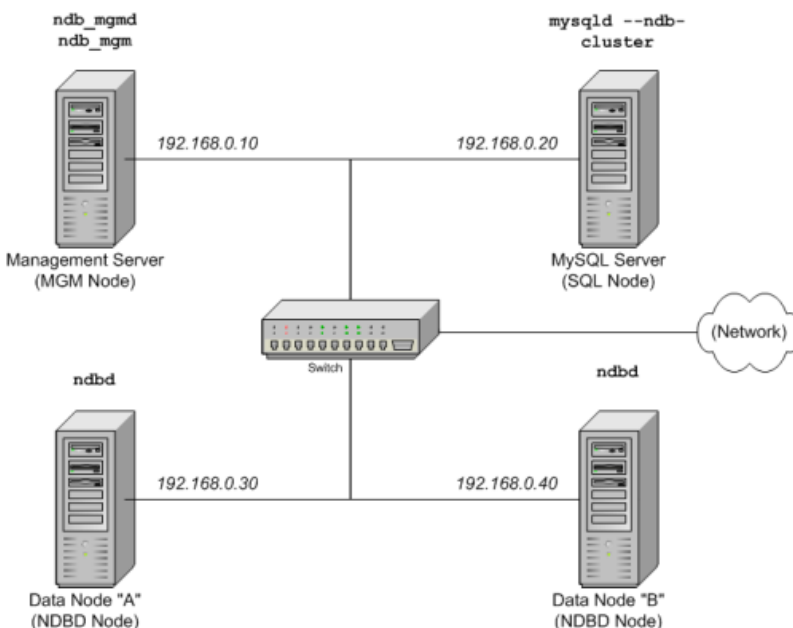
クラスタノードとホストコンピュータ. このクラスタは、ここに示す 4 つのノードで構成されます。 各ノードは別のホストコンピュータ上に配置され、一般的な Ethernet ネットワーク上に固定のネットワークアドレスを持っています。

表 23.5 クラスタ例のノードのネットワークアドレス

ノード	IP アドレス
管理ノード (mgmd)	198.51.100.10
SQL ノード (mysqld)	198.51.100.20
データノード "A" (ndbd)	198.51.100.30
データノード "B" (ndbd)	198.51.100.40

この設定は、次の図にも示されています:

図 23.4 NDB Cluster マルチコンピュータの設定



ネットワークアドレス設定。この How-To では、簡略化 (および信頼性) のために数値 IP アドレスのみが使用されます。ただし、ネットワーク上で DNS 解決が利用可能な場合は、クラスタ構成時に IP アドレスの代わりにホスト名を使用できます。また、`hosts` ファイル (通常、Linux およびその他の Unix 系オペレーティングシステムでは `/etc/hosts`、Windows では `C:\WINDOWS\system32\drivers\etc\hosts`、または使用しているオペレーティングシステムの同等のファイル) が使用可能な場合は、ホスト検索を行う手段として使用することもできます。

NDB 8.0.22 より前では、データおよび管理ノードとの接続に使用されるすべてのネットワークアドレスは、IPv4 を使用して使用または解決できる必要があります。これには、SQL ノードが他のノードに接続するために使用するアドレスが含まれます。NDB 8.0.22 以降では、NDB Cluster は任意のクラスタノードとすべてのクラスタノード間の接続のために IPv6 をサポートしています。

hosts ファイルの潜在的な問題。クラスタノードにホスト名を使用しようとしたときによくある問題は、一部のオペレーティングシステム (一部の Linux 配布を含む) がインストール中にシステム独自のホスト名を `/etc/hosts` に設定する方法が原因で発生します。 `ndb1` および `ndb2` というホスト名を持つ 2 台のマシンがどちらも `cluster` ネットワークドメインに含まれる場合について考えます。Red Hat Linux (CentOS や Fedora などの一部の派生バージョンを含む) では、これらのマシンの `/etc/hosts` ファイルに次のエントリが設定されます。

```
# ndb1 /etc/hosts:
127.0.0.1 ndb1.cluster ndb1 localhost.localdomain localhost
```

```
# ndb2 /etc/hosts:
127.0.0.1 ndb2.cluster ndb2 localhost.localdomain localhost
```

SUSE Linux (OpenSUSE を含む) では、マシンの `/etc/hosts` ファイルにこれらのエントリが設定されます。

```
# ndb1 /etc/hosts:
127.0.0.1 localhost
127.0.0.2 ndb1.cluster ndb1
```

```
# ndb2 /etc/hosts:
127.0.0.1 localhost
127.0.0.2 ndb2.cluster ndb2
```

どちらの場合も、`ndb1` は `ndb1.cluster` をループバック IP アドレスにルーティングしますが、DNS から `ndb2.cluster` のパブリック IP アドレスを取得します。一方、`ndb2` は `ndb2.cluster` をループバックアドレスにルーティングし、`ndb1.cluster` のパブリックアドレスを取得します。その結果、各データノードは管理サーバーに接続しますが、ほかのデータノードが接続したことを検出できないため、データノードが起動中にハングアップしたように見えます。

注意

`config.ini` では `localhost` とほかのホスト名または IP アドレスを混在できません。これらの理由により、このようなケースの (`config.ini` のすべての `HostName` エントリで IP アドレスを使用する以外の) 解決策は、すべてのクラスタホストの `/etc/hosts` から完全修飾ホスト名を削除し、`config.ini` で使用することです。

ホストコンピュータのタイプ。このインストールシナリオに含まれる各ホストコンピュータは、標準的な構成でディスクにインストールされたサポート対象のオペレーティングシステムを実行し、不要なサービスを実行していない Intel ベースのデスクトップ PC です。標準の TCP/IP ネットワーク機能を含む中核的なオペレーティングシステムで十分です。また、簡略化のため、すべてのホストのファイルシステムが完全に同じように設定されていると仮定します。そうでない場合は、状況に応じてこれらの手順を適用してください。

ネットワークハードウェア。各マシンには標準の 100M ビット/秒または 1 ギガビット Ethernet カードが (カードに対応するドライバとともに) 取り付けられ、4 台のホストすべてがスイッチなどの標準仕様の Ethernet ネットワークアプライアンスを介して接続されています。(すべてのマシンで、同じスループットのネットワークカードを使用する必要があります。つまり、クラスタ内の 4 台のマシンで 100M ビット/秒カードを使用するか、または 4 台のマシンで 1 ギガビットカードを使用してください。) NDB Cluster は 100 Mbps ネットワークで動作しますが、ギガビットイーサネットを使用するとパフォーマンスが向上します。

重要

NDB Cluster は、スループットが 100 Mbps 未満のネットワーク、または待機時間が非常に長いネットワークでの使用を目的としていません。このため (特に)、インターネットなどの広域ネットワーク上で NDB Cluster を実行しようとしても成功しない可能性が高く、本番ではサポートされません。

サンプルデータ。MySQL の web サイトからダウンロードできる `world` データベースを使用します (<https://dev.mysql.com/doc/index-other.html> を参照)。各マシンには、オペレーティングシステム、必要な NDB Cluster プロセス、および (データノード上の) データベースを格納するための十分なメモリーがあることを前提としています。

MySQL のインストールに関する一般的な情報は、第2章「MySQL のインストールとアップグレード」を参照してください。Linux およびその他の Unix に似たオペレーティングシステムへの NDB Cluster のインストールについては、[セクション23.2.1「Linux での NDB Cluster のインストール」](#)を参照してください。Windows オペレーティングシステムへの NDB Cluster のインストールについては、[セクション23.2.2「Windows への NDB Cluster のインストール」](#)を参照してください。

NDB Cluster のハードウェア、ソフトウェア、およびネットワーク要件に関する一般的な情報については、[セクション23.1.3「NDB Cluster のハードウェア、ソフトウェア、およびネットワーク要件」](#)を参照してください。

23.2.1 Linux での NDB Cluster のインストール

このセクションでは、Linux およびその他の Unix に似たオペレーティングシステムでの NDB Cluster のインストール方法について説明します。次のセクションでは Linux オペレーティングシステムについて言及しますが、記載されている指示と手順は、サポートされるほかの Unix 系プラットフォームにも簡単に適用できます。Windows システムに固有の手動のインストールおよびセットアップ手順については、[セクション23.2.2「Windows への NDB Cluster のインストール」](#)を参照してください。

各 NDB Cluster ホストコンピュータには、正しい実行可能プログラムがインストールされている必要があります。SQL ノードを実行するホストには、MySQL Server バイナリ (`mysqld`) がインストールされている必要があります。管理ノードには、管理サーバードーモン (`ndb_mgmd`) が必要です。データノードには、データノードデーモン (`ndbd` または `ndbmtbd`) が必要です。管理ノードホストおよびデータノードホストに MySQL Server バイナリをインストールする必要はありません。管理サーバードーモンには管理クライアント (`ndb_mgm`) もインストールすることをお勧めします。

Linux への NDB Cluster のインストールは、Oracle (.tar.gz アーカイブとしてダウンロード)、RPM パッケージ (Oracle から入手可能)、またはソースコードからプリコンパイルされたバイナリを使用して実行できます。次のセクションでは、これら 3 つのすべてのインストール方法について説明します。

使用する方法に関係なく、NDB Cluster バイナリをインストールしたあとも、クラスタを起動する前に、すべてのクラスタノードの構成ファイルを作成する必要があります。[セクション23.2.3「NDB Cluster の初期構成」](#)を参照してください。

23.2.1.1 Linux への NDB Cluster バイナリリリースのインストール

このセクションでは、Oracle が提供する事前コンパイル済みバイナリからクラスタノードの各タイプに対応する適切な実行可能ファイルをインストールするために必要なステップについて説明します。

プリコンパイルされたバイナリを使用してクラスタを設定する場合、各クラスタホストのインストールプロセスの最初のステップは、「[NDB Cluster のダウンロードページ](#)」からバイナリアーカイブをダウンロードすることです。(最新の 64 ビット NDB 8.0 リリースの場合、これは `mysql-cluster-gpl-8.0.22-linux-glibc2.12-x86_64.tar.gz` です。)ここでは、このファイルが各マシンの `/var/tmp` ディレクトリに配置されていると仮定します。

カスタムバイナリが必要な場合は、[セクション2.9.5「開発ソースツリーを使用して MySQL をインストールする」](#)を参照してください。

注記

インストールが完了しても、バイナリはまだ起動しないでください。ノードの構成に続いてその実行方法を示します ([セクション23.2.3「NDB Cluster の初期構成」](#)を参照してください)。

SQL ノード。SQL ノードのホストとして指定された各マシンで、システムの `root` ユーザーとして次のステップを実行します。

1. `/etc/passwd` および `/etc/group` ファイルを調べて (または、オペレーティングシステムが提供する何らかのユーザーおよびグループ管理ツールを使用して)、システムに `mysql` グループと `mysql` ユーザーがすでに存在しているかどうかを確認します。一部の OS 配布では、オペレーティングシステムのインストールプロセスの一部としてこれらが作成されます。まだ存在しない場合は、`mysql` ユーザーグループを新規作成し、このグループに `mysql` ユーザーを追加します。

```
shell> groupadd mysql
shell> useradd -g mysql -s /bin/false mysql
```

`useradd` および `groupadd` の構文は、Unix のバージョンによって多少異なる場合があります。また `adduser` および `addgroup` などの別な名前を使用している場合もあります。

- ダウンロードしたファイルを含むディレクトリに移動し、アーカイブを解凍して、`mysql` という名前で `mysql` ディレクトリへのシンボリックリンクを作成します。

注記

実際のファイル名とディレクトリ名は NDB Cluster のバージョン番号によって異なります。

```
shell> cd /var/tmp
shell> tar -C /usr/local -zxvf mysql-cluster-gpl-8.0.22-linux-glibc2.12-x86_64.tar.gz
shell> ln -s /usr/local/mysql-cluster-gpl-8.0.22-linux-glibc2.12-x86_64 /usr/local/mysql
```

- 次に示すように、場所を `mysql` ディレクトリに変更し、`mysqld --initialize` を使用してシステムデータベースを設定します:

```
shell> cd mysql
shell> mysqld --initialize
```

これにより、MySQL `root` アカウントのランダムパスワードが生成されます。ランダムパスワードを生成しない場合は、`--initialize` の `--initialize-insecure` オプションに置き換えることができます。いずれの場合も、このステップを実行する前に、[セクション2.10.1「データディレクトリの初期化」](#)で追加情報を確認する必要があります。[セクション4.4.2「mysql_secure_installation — MySQL インストールのセキュリティ改善」](#)も参照してください。

- MySQL サーバーおよびデータディレクトリに必要なアクセス許可を設定します。

```
shell> chown -R root .
shell> chown -R mysql data
shell> chgrp -R mysql .
```

- MySQL 起動スクリプトを適切なディレクトリにコピーし、実行可能にして、オペレーティングシステムがブートしたときに起動するように設定します。

```
shell> cp support-files/mysql.server /etc/rc.d/init.d/
shell> chmod +x /etc/rc.d/init.d/mysql.server
shell> chkconfig --add mysql.server
```

(起動スクリプトのディレクトリは、オペレーティングシステムとバージョンによって異なります。たとえば、一部の Linux 配布では `/etc/init.d` です。)

ここでは、起動スクリプトへのリンクを作成するために Red Hat の `chkconfig` を使用します。使用しているプラットフォームのこの目的に適した何らかの手段 (Debian の `update-rc.d` など) を使用してください。

前述の各ステップは、SQL ノードを配置するマシンごとに繰り返す必要があります。

データノード. データノードのインストールには、`mysqld` バイナリは必要ありません。NDB Cluster データノード実行可能ファイル `ndbd` (シングルスレッド) または `ndbmtbd` (マルチスレッド) のみが必要です。これらのバイナリも、`.tar.gz` アーカイブに含まれています。ここでも、このアーカイブが `/var/tmp` に配置されていると仮定します。

システムの `root` として (つまり、`sudo`、`su root`、または使用しているシステムでシステム管理者アカウントの特権を一時的に持つための同等のコマンドを使用したあとで)、次のステップを実行してデータノードホストにデータノードバイナリをインストールします。

- `/var/tmp` ディレクトリに移動し、アーカイブに含まれる `ndbd` および `ndbmtbd` バイナリを `/usr/local/bin` などの適切なディレクトリに抽出します。

```
shell> cd /var/tmp
shell> tar -zxvf mysql-cluster-gpl-8.0.22-linux-glibc2.12-x86_64.tar.gz
shell> cd mysql-cluster-gpl-8.0.22-linux-glibc2.12-x86_64
shell> cp bin/ndbd /usr/local/bin/ndbd
shell> cp bin/ndbmtbd /usr/local/bin/ndbmtbd
```


(`ndb_mgm` および `ndb_mgmd` を実行可能ファイルのディレクトリにコピーしたあとは、ダウンロードしたアーカイブを解凍したときに作成されたディレクトリ (およびディレクトリ内のファイル) を `/var/tmp` から安全に削除できます。)

2. ファイルをコピーしたディレクトリに移動して、両方のファイルを実行可能にします。

```
shell> cd /usr/local/bin
shell> chmod +x ndb*
```

前述のステップは、データノードホストごとに繰り返してください。

NDB Cluster データノードを実行するために必要なデータノード実行可能ファイルは 1 つだけですが、前の手順で `ndbd` と `ndbmtbd` の両方をインストールする方法を示しました。NDB Cluster をインストールまたはアップグレードするときは、NDB Cluster のいずれか一方のみを使用する予定の場合でも、これを行うことをお勧めします。これにより、あとでいずれかから他方への変更を決定した場合に時間と問題が節約されるためです。

注記

データノードをホストする各マシン上のデータディレクトリは `/usr/local/mysql/data` です。この情報は、管理ノードを構成するときに重要になります。(セクション23.2.3「NDB Cluster の初期構成」を参照してください。)

管理ノード。管理ノードのインストールには、`mysqld` バイナリは必要ありません。NDB Cluster 管理サーバー (`ndb_mgmd`) のみが必要です。多くの場合、管理クライアント (`ndb_mgm`) もインストールします。これらのバイナリは、どちらも `.tar.gz` アーカイブに含まれています。ここでも、このアーカイブが `/var/tmp` に配置されていると仮定します。

システムの `root` として、次のステップを実行して管理ノードホストに `ndb_mgmd` および `ndb_mgm` をインストールします。

1. `/var/tmp` ディレクトリに移動し、アーカイブに含まれる `ndb_mgm` および `ndb_mgmd` を `/usr/local/bin` などの適切なディレクトリに抽出します。

```
shell> cd /var/tmp
shell> tar -zxvf mysql-cluster-gpl-8.0.22-linux-glibc2.12-x86_64.tar.gz
shell> cd mysql-cluster-gpl-8.0.22-linux-glibc2.12-x86_64
shell> cp bin/ndb_mgm* /usr/local/bin
```

(`ndb_mgm` および `ndb_mgmd` を実行可能ファイルのディレクトリにコピーしたあとは、ダウンロードしたアーカイブを解凍したときに作成されたディレクトリ (およびディレクトリ内のファイル) を `/var/tmp` から安全に削除できます。)

2. ファイルをコピーしたディレクトリに移動して、両方のファイルを実行可能にします。

```
shell> cd /usr/local/bin
shell> chmod +x ndb_mgm*
```

セクション23.2.3「NDB Cluster の初期構成」では、NDB Cluster の例のすべてのノードの構成ファイルを作成します。

23.2.1.2 RPM から NDB Cluster をインストール

このセクションでは、Oracle が提供する RPM パッケージを使用して、NDB Cluster 8.0 ノードのタイプごとに正しい実行可能ファイルをインストールするために必要な手順について説明します。

このセクションで説明する方法の代わりに、Oracle は、多くの一般的な Linux ディストリビューションと互換性のある NDB Cluster 用の MySQL リポジトリを提供します。RPM ベースの配布では、次の 2 つのリポジトリを使用できます:

- `yum` または `dnf` を使用する配布の場合は、NDB Cluster 用の MySQL Yum リポジトリを使用できます。手順および追加情報は、「[Yum リポジトリを使用した MySQL NDB Cluster のインストール](#)」を参照してください。
- SLES の場合、NDB Cluster 用の MySQL SLES リポジトリを使用できます。手順および追加情報は、「[SLES リポジトリを使用した MySQL NDB Cluster のインストール](#)」を参照してください。

RPM は、32 ビットと 64 ビットの両方の Linux プラットフォームで使用できます。これらの RPM のファイル名には、次のパターンが使用されています。

```
mysql-cluster-community-data-node-8.0.22-1.el7.x86_64.rpm  
mysql-cluster-license-component-ver-rev.distro.arch.rpm  
  
license:= {commercial | community}  
  
component: {management-server | data-node | server | client | other—see text}  
  
ver: major.minor.release  
  
rev: major[.minor]  
  
distro: {el6 | el7 | sles12}  
  
arch: {i686 | x86_64}
```

`license` は、RPM が NDB Cluster のコマーシャルリリースまたはコミュニティリリースのどちらの一部であることを示します。このセクションの残りの部分では、Community リリースをインストールする例を想定しています。

次のテーブルに、`component` で使用可能な値とその説明を示します:

表 23.6 NDB Cluster RPM ディストリビューションのコンポーネント

コンポーネント	説明
<code>auto-installer</code> (DEPRECATED)	NDB Cluster Auto Installer プログラム。使用方法については、 セクション23.2.8「NDB Cluster Auto-Installer (サポートされなくなりました)」 を参照してください
クライアント	MySQL および NDB クライアントプログラム。mysql クライアント、ndb_mgm クライアントおよびその他のクライアントツールが含まれます
<code>common</code>	MySQL サーバーに必要な文字セットおよびエラーメッセージ情報
<code>data-node</code>	ndbd および ndbmtbd データノードバイナリ
<code>devel</code>	MySQL クライアント開発に必要なヘッダーおよびライブラリファイル
<code>embedded</code>	埋込み MySQL サーバー
<code>embedded-compat</code>	下位互換埋込み MySQL サーバー
<code>embedded-devel</code>	埋込み MySQL 用のアプリケーションを開発するためのヘッダーおよびライブラリファイル
<code>java</code>	ClusterJ アプリケーションのサポートに必要な JAR ファイル
<code>libs</code>	MySQL クライアントライブラリ
<code>libs-compat</code>	下位互換性のある MySQL クライアントライブラリ
<code>management-server</code>	NDB Cluster 管理サーバー (ndb_mgmd)
<code>memcached</code>	ndbmemcache のサポートに必要なファイル
<code>minimal-debuginfo</code>	パッケージ server-minimal のデバッグ情報。このパッケージを使用するアプリケーションを開発する場合、またはこのパッケージをデバッグする場合に役立ちます
<code>ndbclient</code>	NDB API および MGM API アプリケーションを実行するための NDB クライアントライブラリ (libndbclient)
<code>ndbclient-devel</code>	NDB API および MGM API アプリケーションの開発に必要なヘッダーおよびその他のファイル
<code>nodejs</code>	NDB Cluster の Node.JS サポートの設定に必要なファイル

コンポーネント	説明
server	NDB ストレージエンジンサポートを含む MySQL サーバー (mysqld) と、関連する MySQL サーバープログラム
server-minimal	NDB および関連ツール用の MySQL サーバーの最小インストール
test	mysqltest 、その他の MySQL テストプログラムおよびサポートファイル

特定のプラットフォームおよびアーキテクチャーのすべての NDB Cluster RPM の単一バンドル ([.tar](#) ファイル) も使用できます。このファイルの名前は、次に示すパターンに従います:

```
mysql-cluster-license-ver-rev.distro.arch.rpm-bundle.tar
```

[tar](#) またはアーカイブの抽出に適したツールを使用して、このファイルから個々の RPM ファイルを抽出できます。

NDB Cluster ノードの 3 つの主要なタイプをインストールするために必要なコンポーネントを次のリストに示します:

- 管理ノード: [management-server](#)
- データノード: [data-node](#)
- SQL ノード: [server](#) および [common](#)

また、[client](#) RPM をインストールして、少なくとも 1 つの管理ノードに [ndb_mgm](#) 管理クライアントを提供する必要があります。SQL ノードにインストールして、[mysql](#) およびその他の MySQL クライアントプログラムを使用可能にすることもできます。ノードのインストールについては、このセクションの後半でタイプ別に説明します。

[ver](#) は、例で [8.0.22](#) として表示される 8.0. x 形式の 3 部分の NDB ストレージエンジンのバージョン番号を表します。[rev](#) は [major.minor](#) 形式の RPM リビジョン番号を提供します。このセクションの例では、この値に [1.1](#) を使用します。

[distro](#) (Linux ディストリビューション) は、[rhel5](#) (Oracle Linux 5, Red Hat Enterprise Linux 4 and 5)、[el6](#) (Oracle Linux 6, Red Hat Enterprise Linux 6)、[el7](#) (Oracle Linux 7, Red Hat Enterprise Linux 7) または [sles12](#) (SUSE Enterprise Linux 12) のいずれかです。このセクションの例では、ホストで Oracle Linux 7、Red Hat Enterprise Linux 7 または同等 ([el7](#)) が実行されていることを前提としています。

[arch](#) は、32-bit RPM 用の [i686](#) および 64-bit バージョン用の [x86_64](#) です。ここに示す例では、64 ビットプラットフォームを想定しています。

RPM ファイル名 (ここでは [8.0.22](#) として示されています) の NDB Cluster のバージョン番号は、実際に使用しているバージョンによって異なる可能性があります。インストールするすべてのクラスタ RPM のバージョン番号が同じになっていることが非常に重要です。アーキテクチャは RPM がインストールされるマシンにも適している必要があります。特に、64 ビット RPM ([x86_64](#)) は 32 ビットオペレーティングシステムでは使用できないことに注意してください (後者には [i686](#) を使用してください)。

データノード. NDB Cluster データノードをホストするコンピュータでは、[data-node](#) RPM のみをインストールする必要があります。これを行うには、この RPM をデータノードホストにコピーし、システムの root ユーザーとして次のコマンドを実行し、必要に応じて RPM に表示される名前を MySQL web サイトからダウンロードした RPM の名前と一致するように置き換えます:

```
shell> rpm -Uvh mysql-cluster-community-data-node-8.0.22-1.el7.x86_64.rpm
```

これにより、[ndbd](#) および [ndbmttd](#) データノードのバイナリが [/usr/sbin](#) にインストールされます。これらのいずれかを使用して、このホストでデータノードプロセスを実行できます。

SQL ノード. NDB Cluster SQL ノードのホストに使用する各マシンに [server](#) および [common](#) RPM をコピーします ([server](#) には [common](#) が必要です)。システムルートユーザーとして次のコマンドを実行し、MySQL web サイトからダウンロードした RPM の名前と一致するように、必要に応じて RPM に表示される名前を置き換えて、[server](#) RPM をインストールします:

```
shell> rpm -Uvh mysql-cluster-community-server-8.0.22-1.el7.x86_64.rpm
```

これにより、NDB ストレージエンジンがサポートされた MySQL サーババイナリ (`mysqld`) が `/usr/sbin` ディレクトリにインストールされます。また、必要なすべての MySQL Server サポートファイルと、`mysql.server` および `mysqld_safe` 起動スクリプトを含む有用な MySQL サーバプログラムもインストールされます (`/usr/share/mysql` および `/usr/bin` 内)。RPM インストーラでは、一般的な構成の問題 (必要に応じて `mysql` ユーザーおよびグループを作成するなど) に自動的に対応します。

重要

NDB Cluster 用にリリースされたこれらの RPM のバージョンを使用する必要があります。標準の MySQL サーバ用にリリースされた RPM は、NDB ストレージエンジンのサポートを提供しません。

SQL ノード (MySQL サーバ) を管理するには、ここに示すように `client` RPM もインストールするようにしてください。

```
shell> rpm -Uvh mysql-cluster-community-client-8.0.22-1.el7.x86_64.rpm
```

これにより、`mysql` クライアントおよびその他の MySQL クライアントプログラム (`mysqladmin` や `mysqldump` など) が `/usr/bin` にインストールされます。

管理ノード. NDB Cluster 管理サーバをインストールするには、`management-server` RPM のみを使用する必要があります。この RPM を管理ノードをホストするコンピュータにコピーし、システムルートユーザーとして次のコマンドを実行してインストールします (MySQL web サイトからダウンロードした `management-server` RPM の名前と一致するように、RPM に示されている名前を必要に応じて置き換えます):

```
shell> rpm -Uvh mysql-cluster-community-management-server-8.0.22-1.el7.x86_64.rpm
```

この RPM は、管理サーバのバイナリ `ndb_mgmd` を `/usr/sbin` ディレクトリにインストールします。これは管理ノードを実行するために実際に必要な唯一のプログラムですが、`ndb_mgm` NDB Cluster 管理クライアントも使用可能にすることをお勧めします。前述のように `client` RPM をインストールすると、このプログラムおよび `ndb_desc` や `ndb_config` などの他の NDB クライアントプログラムを入手できます。

オラクルが提供する RPM を使った MySQL のインストールに関する一般的な情報は、[セクション2.5.4「Oracle の RPM パッケージを使用した Linux への MySQL のインストール」](#)を参照してください。

RPM からインストールした後も、クラスタを構成する必要があります。関連情報は、[セクション23.2.3「NDB Cluster の初期構成」](#)を参照してください。

インストールするすべてのクラスタ RPM のバージョン番号が同じになっていることが非常に重要です。[architecture](#) の指定は、RPM をインストールするマシンにも適している必要があります。特に、64 ビット RPM は 32 ビットオペレーティングシステムでは使用できないことに注意してください。

データノード. クラスタのデータノードをホストするコンピュータには、`server` RPM のみをインストールする必要があります。これを行うには、この RPM をデータノードホストにコピーし、システムの `root` ユーザーとして次のコマンドを実行し、必要に応じて RPM に表示される名前を MySQL web サイトからダウンロードした RPM の名前と一致するように置き換えます:

```
shell> rpm -Uvh MySQL-Cluster-server-gpl-8.0.22-1.sles11.i386.rpm
```

これにより、すべての NDB Cluster バイナリがインストールされますが、NDB Cluster データノードを実行するには、プログラム `ndbd` または `ndbmtd` (両方とも `/usr/sbin` 内) のみが実際に必要です。

SQL ノード. クラスタ SQL ノードのホストに使用する各マシンで、システムルートユーザーとして次のコマンドを実行し、MySQL web サイトからダウンロードした RPM の名前と一致するように RPM に表示される名前を置き換えて、`server` RPM をインストールします:

```
shell> rpm -Uvh MySQL-Cluster-server-gpl-8.0.22-1.sles11.i386.rpm
```

これにより、NDB ストレージエンジンのサポートを含む MySQL サーババイナリ (`mysqld`) と必要なすべての MySQL Server サポートファイルが `/usr/sbin` ディレクトリにインストールされます。また、`mysql.server` および `mysqld_safe` 起動スクリプトも (それぞれ `/usr/share/mysql` および `/usr/bin` に) インストールされます。RPM インストーラでは、一般的な構成の問題 (必要に応じて `mysql` ユーザーおよびグループを作成するなど) に自動的に対応します。

SQL ノード (MySQL サーバー) を管理するには、ここに示すように `client` RPM もインストールするようにしてください。

```
shell> rpm -Uvh MySQL-Cluster-client-gpl-8.0.22-1.sles11.i386.rpm
```

これにより、`mysql` クライアントプログラムがインストールされます。

管理ノード、NDB Cluster 管理サーバーをインストールするには、`server` RPM のみを使用する必要があります。この RPM を管理ノードをホストするコンピュータにコピーし、システムルートユーザーとして次のコマンドを実行してインストールします (MySQL web サイトからダウンロードした `server` RPM の名前と一致するように、RPM に示されている名前を必要に応じて置き換えます)：

```
shell> rpm -Uvh MySQL-Cluster-server-gpl-8.0.22-1.sles11.i386.rpm
```

この RPM によってほかの多くのファイルがインストールされますが、管理ノードを実行するために実際に必要なのは管理サーバーバイナリ `ndb_mgmd` (`/usr/sbin` ディレクトリにあります) だけです。`server` RPM によって、NDB の管理クライアントである `ndb_mgm` もインストールされます。

オラクルが提供する RPM を使った MySQL のインストールに関する一般的な情報は、[セクション2.5.4「Oracle の RPM パッケージを使用した Linux への MySQL のインストール」](#)を参照してください。必要なインストール後の構成の詳細は、[セクション23.2.3「NDB Cluster の初期構成」](#)を参照してください。

23.2.1.3 .deb ファイルを使用した NDB Cluster のインストール

このセクションでは、この目的のために Oracle によって提供される `.deb` ファイルを使用して、NDB Cluster を Debian および関連する Linux ディストリビューション (Ubuntu) にインストールする方法について説明します。

Oracle は、Debian およびその他の配布用の NDB Cluster APT リポジトリも提供します。手順および追加情報は、[「APT リポジトリを使用した MySQL NDB Cluster のインストール」](#)を参照してください。

Oracle には、32 ビットおよび 64 ビットプラットフォーム用の NDB Cluster 用の `.deb` インストーラファイルが用意されています。Debian ベースのシステムでは、単一のインストーラファイルのみが必要です。このファイルには、適用可能な NDB Cluster バージョン、Debian バージョン、およびアーキテクチャーに従って、次に示すパターンを使用して名前が付けられます：

```
mysql-cluster-gpl-ndbver-debiandebianver-arch.deb
```

ここで、`ndbver` は 3 部分の NDB エンジンのバージョン番号、`debianver` は Debian (8 または 9) のメジャーバージョン、`arch` は `i686` または `x86_64` のいずれかです。次の例では、NDB 8.0.22 を 64 ビット Debian 9 システムにインストールすることを前提としています。この場合、インストーラファイルの名前は `mysql-cluster-gpl-8.0.22-debian9-x86_64.deb-bundle.tar` です。

適切な `.deb` ファイルをダウンロードしたら、解凍し、次のように `dpkg` を使用してコマンドラインからインストールできます：

```
shell> dpkg -i mysql-cluster-gpl-8.0.22-debian9-i686.deb
```

次に示すように、`dpkg` を使用して削除することもできます：

```
shell> dpkg -r mysql
```

インストーラファイルは、Gnome デスクトップ用の `GDebi` など、`.deb` ファイルを操作するほとんどのグラフィカルパッケージマネージャとも互換性がある必要があります。

`.deb` ファイルは NDB Cluster を `/opt/mysql/server-version/` の下にインストールします。ここで、`version` は含まれている MySQL サーバーの 2 部分リリースシリーズバージョンです。NDB 8.0 の場合、これは常に `5.7` です。ディレクトリレイアウトは、一般的な Linux バイナリ配布 ([表2.3「一般的な Unix/Linux バイナリパッケージの MySQL インストールのレイアウト」](#)を参照) の場合と同じですが、起動スクリプトおよび構成ファイルが `share` ではなく `support-files` にある点が異なります。`ndb_mgm`、`ndbd`、`ndb_mgmd` などの NDB Cluster 実行可能ファイルはすべて、`bin` ディレクトリに配置されます。

23.2.1.4 Linux でのソースからの NDB Cluster の構築

このセクションでは、Linux およびその他の Unix に似たプラットフォームで NDB Cluster をコンパイルする方法について説明します。ソースから NDB Cluster を構築することは、ここで説明するいくつかの重要な点で異なりますが、標準の MySQL Server を構築することに似ています。ソースからの MySQL のビルドに関する一般的な情報は、[セクション2.9「ソースから MySQL をインストールする」](#)を参照してください。Windows プラットフォームで NDB Cluster をコンパイルする方法については、[セクション23.2.2「Windows でのソースからの NDB Cluster のコンパイルとインストール」](#)を参照してください。

MySQL NDB Cluster 8.0 を構築するには、MySQL Server 8.0 ソースを使用する必要があります。これらは、<https://dev.mysql.com/downloads/> の MySQL ダウンロードページから入手できます。アーカイブされたソースファイルには、[mysql-8.0.22.tar.gz](https://dev.mysql.com/downloads/mysql/8.0.22/#os-linux) と同様の名前を付ける必要があります。<https://github.com/mysql/mysql-server> の GitHub からソースを取得することもできます。

注記

以前のバージョンでは、標準 MySQL Server ソースからの NDB Cluster の構築はサポートされていませんでした。MySQL 8.0 および NDB Cluster 8.0 では、これは両方の製品が同じソースから構築されましたではなくりました。

CMake の `WITH_NDBCLUSTER` オプションを使用すると、管理ノード、データノード、およびその他の NDB Cluster プログラムのバイナリが構築されます。また、`mysqld` は NDB ストレージエンジンをサポートしてコンパイルされます。NDB Cluster を構築するときは、このオプション (またはそのエイリアス `WITH_NDBCLUSTER_STORAGE_ENGINE` と `WITH_PLUGIN_NDBCLUSTER` のいずれか) が必要です。

重要

`WITH_NDB_JAVA` オプションはデフォルトで有効になっています。つまり、デフォルトでは CMake でシステム上の Java の場所が見つからなかった場合に構成プロセスが失敗します。Java および ClusterJ のサポートを有効にしない場合は、`-DWITH_NDB_JAVA=OFF` を使用してビルドを構成することで、これを明示的に示す必要があります。必要な場合は、`WITH_CLASSPATH` を使用して Java クラスパスを指定します。

NDB Cluster の構築に固有の CMake オプションの詳細は、[Options for Compiling NDB Cluster](#) を参照してください。

`make && make install` (または使用しているシステムの同等のコマンド) を実行すると、同じ場所に事前コンパイル済みバイナリを解凍した場合と同じ結果が得られます。

管理ノード. ソースからビルドしてデフォルトの `make install` を実行すると、管理サーバーと管理クライアントのバイナリ (`ndb_mgmd` と `ndb_mgm`) が `/usr/local/mysql/bin` に見つかります。管理ノードホストに配置する必要があるのは `ndb_mgmd` だけですが、同じホストマシンに `ndb_mgm` も配置することをお勧めします。これらの実行可能ファイルは、どちらもホストマシンのファイルシステム上の特定の場所に配置する必要はありません。

データノード. データノードホストに配置する必要がある実行可能ファイルは、データノードバイナリ `ndbd` または `ndbmtbd` だけです。(たとえば、`mysqld` をホストマシンに配置する必要はありません。) ソースからビルドすると、デフォルトではこのファイルは `/usr/local/mysql/bin` ディレクトリに配置されます。複数のデータノードホストにインストールする場合、ほかのマシンにコピーする必要があるのは `ndbd` または `ndbmtbd` だけです。(これは、すべてのデータノードホストで同じアーキテクチャーとオペレーティングシステムが使用されていることが前提です。そうでない場合は、異なるプラットフォームごとに別個にコンパイルする必要がある可能性があります。) データノードバイナリは、その場所が既知であるかぎり、ホストのファイルシステム上の特定の場所に配置する必要はありません。

NDB Cluster をソースからコンパイルする場合、マルチスレッドデータノードバイナリを構築するための特別なオプションは必要ありません。NDB ストレージエンジンのサポート付きでビルドを構成すると、自動的に `ndbmtbd` がビルドされます。`make install` を実行すると、`ndbmtbd` バイナリは `mysqld`、`ndbd`、および `ndb_mgm` とともにインストールの `bin` ディレクトリに配置されます。

SQL ノード. クラスタリングのサポート付きで MySQL をコンパイルし、デフォルトのインストールを実行 (システムの `root` ユーザーとして `make install` を使用) すると、`mysqld` は `/usr/local/mysql/bin` に配置されます。[セクション2.9「ソースから MySQL をインストールする」](#) に示したステップに従って、`mysqld` を使用できるようにします。複数の SQL ノードを実行する場合は、同じ `mysqld` 実行可能ファイルと関連するサポートファイルのコピーを複数のマシンで使用できます。これを実行するもっとも簡単な方法は、`/usr/local/mysql` ディレクトリ全体およびその内部に含まれているすべてのディレクトリとファイルをほかの SQL ノードホストにコピーし、各マシンで [セクション2.9「ソースから MySQL をインストールする」](#) のステップを繰り返すことです。デフォルトではない `PREFIX` オプションを指定してビルドを構成する場合は、それに合わせてディレクトリを調整する必要があります。

セクション23.2.3「NDB Cluster の初期構成」では、NDB Cluster の例のすべてのノードの構成ファイルを作成します。

23.2.2 Windows への NDB Cluster のインストール

このセクションでは、Windows ホストでの NDB Cluster のインストール手順について説明します。Windows 用の NDB Cluster 8.0 バイナリは、<https://dev.mysql.com/downloads/cluster/> から取得できます。Oracle が提供するバイナリリリースから Windows に NDB Cluster をインストールする方法については、セクション23.2.2.1「バイナリリリースから Windows への NDB Cluster のインストール」を参照してください。

Microsoft Visual Studio を使用して、Windows 上のソースから NDB Cluster をコンパイルおよびインストールすることもできます。詳細は、セクション23.2.2.2「Windows でのソースからの NDB Cluster のコンパイルとインストール」を参照してください。

23.2.2.1 バイナリリリースから Windows への NDB Cluster のインストール

このセクションでは、次のテーブルに示すように、このセクション (セクション23.2「NDB Cluster のインストール」を参照) の最初に説明したものと同一 4 ノード設定を使用して、Oracle で提供されるバイナリ「no-install」NDB Cluster リリースを使用した Windows への NDB Cluster の基本的なインストールについて説明します:

表 23.7 クラスタ例のノードのネットワークアドレス

ノード	IP アドレス
管理ノード (mgmd)	198.51.100.10
SQL ノード (mysqld)	198.51.100.20
データノード "A" (ndbd)	198.51.100.30
データノード "B" (ndbd)	198.51.100.40

ほかのプラットフォームと同様に、SQL ノードを実行している NDB Cluster ホストコンピュータには、MySQL Server バイナリ (mysqld.exe) がインストールされている必要があります。このホストには MySQL クライアント (mysql.exe) も配置するようにしてください。管理ノードおよびデータノードに MySQL Server バイナリをインストールする必要はありません。各管理サーバーには、管理サーバーデーモン (ndb_mgmd.exe) が必要です。各データノードには、データノードデーモン (ndbd.exe または ndbmtbd.exe) が必要です。この例では、ndbd.exe をデータノード実行可能ファイルと呼びますが、このプログラムのマルチスレッドバージョンである ndbmtbd.exe をまったく同じ方法でインストールできます。管理サーバーホストには、管理クライアント (ndb_mgm.exe) もインストールするようにしてください。このセクションでは、NDB Cluster ノードのタイプごとに正しい Windows バイナリをインストールするために必要な手順について説明します。

注記

ほかの Windows プログラムと同様に、NDB Cluster 実行可能ファイルには .exe ファイル拡張子が付けられます。ただし、コマンド行からこれらのプログラムを起動するときに .exe 拡張子を含める必要はありません。そのため、このドキュメントでは多くの場合、これらのプログラムを mysqld、mysql、ndb_mgmd などと呼びます。ここでは、(たとえば) mysqld と mysqld.exe のどちらの呼び方であっても、どちらの名前も同じもの (MySQL Server プログラム) を意味しています。

Oracle no-install バイナリを使用して NDB Cluster を設定する場合、インストールプロセスの最初のステップは、<https://dev.mysql.com/downloads/cluster/> から最新の NDB Cluster Windows ZIP バイナリアーカイブをダウンロードすることです。このアーカイブには mysql-cluster-gpl-ver-winarch.zip のファイル名があり、ver は NDB ストレージエンジンのバージョン (8.0.22 など)、arch はアーキテクチャー (32 ビットバイナリの場合は 32、64 ビットバイナリの場合は 64) です。たとえば、64 ビット Windows システム用 NDB Cluster 8.0.22 アーカイブの名前は mysql-cluster-gpl-8.0.22-win64.zip です。

32 ビット NDB Cluster バイナリは、32 ビットバージョンと 64 ビットバージョンの両方の Windows で実行できますが、64 ビット NDB Cluster バイナリは 64 ビットバージョンの Windows でのみ使用できます。64 ビット CPU を搭載したコンピュータで 32 ビットバージョンの Windows を使用している場合は、32 ビット NDB Cluster バイナリを使用する必要があります。

インターネットからダウンロードしたりマシン間でコピーしたりする必要があるファイルの数を最小限に抑えるため、ここでは SQL ノードを実行するコンピュータから始めます。

SQL ノード. 198.51.100.20 という IP アドレスを持つコンピュータ上のディレクトリ `C:\Documents and Settings\username\My Documents\Downloads` にアーカイブのコピーが配置されていることを前提としています。ここで、`username` は現在のユーザーの名前です。(この名前は、コマンド行で `ECHO %USERNAME%` を使用すると表示されます。) NDB Cluster 実行可能ファイルを Windows サービスとしてインストールして実行するには、このユーザーが `Administrators` グループのメンバーである必要があります。

アーカイブからすべてのファイルを抽出します。このタスクには、Windows Explorer に組み込まれた抽出ウィザードが適しています。(別のアーカイブプログラムを使用する場合は、アーカイブからすべてのファイルとディレクトリが抽出されたこと、アーカイブのディレクトリ構造が維持されていることを確認してください。) 宛先ディレクトリを要求されたら、`C:\` と入力します。これにより、抽出ウィザードによってアーカイブが `C:\mysql-cluster-gpl-ver-winarch` ディレクトリに抽出されます。このディレクトリの名前を `C:\mysql` に変更します。

NDB Cluster バイナリは `C:\mysql\bin` 以外のディレクトリにインストールできますが、インストールする場合は、この手順に示すパスを適宜変更する必要があります。特に、MySQL Server (SQL ノード) バイナリを `C:\mysql` または `C:\Program Files\MySQL\MySQL Server 8.0` 以外の場所にインストールした場合や、SQL ノードのデータディレクトリが `C:\mysql\data` または `C:\Program Files\MySQL\MySQL Server 8.0\data` 以外の場所にある場合は、SQL ノードの起動時に、追加の構成オプションをコマンド行で使用するか、`my.ini` または `my.cnf` ファイルに追加する必要があります。標準以外の場所で実行するように MySQL Server を構成する方法の詳細は、[セクション 2.3.4 「noinstall ZIP アーカイブを使用した Microsoft Windows への MySQL のインストール」](#) を参照してください。

NDB Cluster をサポートする MySQL Server を NDB Cluster の一部として実行するには、`--ndbcluster` および `--ndb-connectstring` オプションを指定して起動する必要があります。これらのオプションは、コマンド行で指定することもできますが、通常はオプションファイルに設定する方が便利です。そのためには、メモ帳などのテキストエディタで新しいテキストファイルを作成します。このファイルに次の構成情報を入力します。

```
[mysqld]
# Options for mysqld process:
ndbcluster                # run NDB storage engine
ndb-connectstring=198.51.100.10 # location of management server
```

この MySQL Server が使用するほかのオプション ([セクション 2.3.4.2 「オプションファイルの作成」](#) を参照してください) を必要に応じて追加できますが、このファイルには少なくともここに示したオプションを含める必要があります。このファイルを `C:\mysql\my.ini` として保存します。これで、SQL ノードのインストールとセットアップが完了します。

データノード. Windows ホスト上の NDB Cluster データノードには、`ndbd.exe` または `ndbmtid.exe` のいずれかの単一の実行可能ファイルのみが必要です。この例では、`ndbd.exe` を使用すると仮定しますが、`ndbmtid.exe` を使用するときも同じ手順が適用されます。データノードを実行する各コンピュータ (IP アドレスが 198.51.100.30 および 198.51.100.40 のコンピュータ) で、`C:\mysql`、`C:\mysql\bin`、および `C:\mysql\cluster-data` の各ディレクトリを作成します。次に、`no-install` アーカイブをダウンロードして抽出したコンピュータで、`C:\mysql\bin` ディレクトリ内の `ndbd.exe` を見つけます。このファイルを 2 台のデータノードホストの `C:\mysql\bin` ディレクトリにそれぞれコピーします。

NDB Cluster の一部として機能するには、各データノードに管理サーバーのアドレスまたはホスト名を指定する必要があります。この情報を指定するには、各データノードプロセスの起動時にコマンド行で `--ndb-connectstring` または `-c` オプションを使用します。ただし、通常はオプションファイルにこの情報を指定することをお勧めします。そのためには、メモ帳などのテキストエディタで新しいテキストファイルを作成して、次のテキストを入力します。

```
[mysql_cluster]
# Options for data node process:
ndb-connectstring=198.51.100.10 # location of management server
```

このファイルをデータノードホストに `C:\mysql\my.ini` として保存します。もう一方のデータノードホストで同じ内容を含むテキストファイルをもう 1 つ作成し、それを `C:\mysql\my.ini` として保存するか、`my.ini` ファイルを 1 つ目のデータノードホストから 2 つ目のデータノードホストにコピーし、そのコピーを 2 つ目のデータノードの `C:\mysql` ディレクトリに確実に配置します。両方のデータノードホストを NDB Cluster で使用する準備ができました。これにより、インストールおよび構成する管理ノードのみが残されます。

管理ノード. NDB Cluster 管理ノードのホストに使用されるコンピュータで必要な実行可能プログラムは、管理サーバープログラム `ndb_mgmd.exe` だけです。ただし、NDB Cluster を起動したあとで管理するには、NDB Cluster 管理クライアントプログラム `ndb_mgm.exe` も管理サーバーと同じマシンにインストールするようにしてください。 `no-`

`install` アーカイブをダウンロードして抽出したマシンで、これら 2 つのプログラムを見つけます。これは、SQL ノードホストの `C:\mysql\bin` ディレクトリになります。IP アドレスが 198.51.100.10 であるコンピュータに `C:\mysql\bin` ディレクトリを作成し、両方のプログラムをこのディレクトリにコピーします。

ここで、`ndb_mgmd.exe` が使用する 2 つの構成ファイルを作成してください。

1. 管理ノード自体に固有の構成データを提供するローカル構成ファイル。通常、このファイルは NDB Cluster グローバル構成ファイルの場所を指定するだけで済みます (項目 2 を参照)。

このファイルを作成するには、メモ帳などのテキストエディタで新しいテキストファイルを作成し、次の情報を入力します。

```
[mysql_cluster]
# Options for management node process
config-file=C:/mysql/bin/config.ini
```

このファイルをテキストファイル `C:\mysql\bin\my.ini` として保存します。

2. NDB Cluster 全体を管理する構成情報を管理ノードが取得できるグローバル構成ファイル。少なくとも、このファイルには NDB Cluster 内の各ノードのセクションと、管理ノードおよびすべてのデータノードの IP アドレスまたはホスト名 (`HostName` 構成パラメータ) が含まれている必要があります。また、次の追加情報も含めることをお勧めします。

- SQL ノードの IP アドレスまたはホスト名
- 各データノードに割り当てられたデータメモリーおよびインデックスメモリー (`DataMemory` および `IndexMemory` 構成パラメータ)
- `NoOfReplicas` 構成パラメータを使用したフラグメントレプリカの数 (セクション 23.1.2 「NDB Cluster ノード、ノードグループ、フラグメントレプリカ、およびパーティション」 を参照)
- 各データノードがデータおよびログファイルを格納するディレクトリと、管理ノードがログファイルを保持するディレクトリ (どちらの場合も、`DataDir` 構成パラメータ)

メモ帳などのテキストエディタを使用して新しいテキストファイルを作成し、次の情報を入力します。

```
[ndbd default]
# Options affecting ndbd processes on all data nodes:
NoOfReplicas=2          # Number of fragment replicas
DataDir=C:/mysql/cluster-data # Directory for each data node's data files
                          # Forward slashes used in directory path,
                          # rather than backslashes. This is correct;
                          # see Important note in text
DataMemory=80M         # Memory allocated to data storage
IndexMemory=18M        # Memory allocated to index storage
                          # For DataMemory and IndexMemory, we have used the
                          # default values. Since the "world" database takes up
                          # only about 500KB, this should be more than enough for
                          # this example Cluster setup.

[ndb_mgmd]
# Management process options:
HostName=198.51.100.10 # Hostname or IP address of management node
DataDir=C:/mysql/bin/cluster-logs # Directory for management node log files

[ndbd]
# Options for data node "A":
# (one [ndbd] section per data node)
HostName=198.51.100.30 # Hostname or IP address

[ndbd]
# Options for data node "B":
HostName=198.51.100.40 # Hostname or IP address

[mysqld]
# SQL node options:
HostName=198.51.100.20 # Hostname or IP address
```

このファイルをテキストファイル `C:\mysql\bin\config.ini` として保存します。

重要

Windows 上の NDB Cluster で使用されるプログラムオプションまたは構成ファイルでディレクトリパスを指定する場合、単一のバックスラッシュ文字 (\) は使用できません。代わりに、個々のバックスラッシュ文字を 2 つ目のバックスラッシュ (\\) でエスケープするか、バックスラッシュをスラッシュ文字 (/) に置き換えてください。たとえば、NDB Cluster `config.ini` ファイルの `[ndb_mgmd]` セクションからの次の行は機能しません:

```
DataDir=C:\mysql\bin\cluster-logs
```

代わりに、次のいずれかを使用できます。

```
DataDir=C:\\mysql\\bin\\cluster-logs # Escaped backslashes
```

```
DataDir=C:/mysql/bin/cluster-logs # Forward slashes
```

簡潔さと読みやすさの理由から、NDB Cluster プログラムのオプションおよび Windows の構成ファイルで使用されるディレクトリパスにはスラッシュを使用することをお勧めします。

23.2.2.2 Windows でのソースからの NDB Cluster のコンパイルとインストール

Oracle には、ほとんどのユーザーに適したプリコンパイル済み NDB Cluster バイナリが Windows 用に用意されています。ただし、必要に応じて、NDB Cluster for Windows をソースコードからコンパイルすることもできます。これを実行する手順は、標準の Windows 用 MySQL Server バイナリをコンパイルする場合の手順とほぼ同じであり、同じツールを使用します。ただし、2 つの大きな違いがあります。

- MySQL NDB Cluster 8.0 を構築するには、MySQL Server 8.0 ソースを使用する必要があります。これらは、<https://dev.mysql.com/downloads/> の MySQL ダウンロードページから入手できます。アーカイブされたソースファイルには、[mysql-8.0.22.tar.gz](https://github.com/mysql/mysql-server) と同様の名前を付ける必要があります。<https://github.com/mysql/mysql-server> の GitHub からソースを取得することもできます。
- CMake で使用するその他のビルドオプションに加えて、`WITH_NDBCLUSTER` オプションを使用してビルドを構成する必要があります。`WITH_NDBCLUSTER_STORAGE_ENGINE` および `WITH_PLUGIN_NDBCLUSTER` は、`WITH_NDBCLUSTER` のエイリアスとしてサポートされており、まったく同じ方法で動作します。

重要

`WITH_NDB_JAVA` オプションはデフォルトで有効になっています。つまり、デフォルトでは CMake でシステム上の Java の場所が見つからなかった場合に構成プロセスが失敗します。Java および ClusterJ のサポートを有効にしない場合は、`-DWITH_NDB_JAVA=OFF` を使用してビルドを構成することで、これを明示的に示す必要があります。(Bug #12379735) 必要な場合は、`WITH_CLASSPATH` を使用して Java クラスパスを指定します。

NDB Cluster の構築に固有の CMake オプションの詳細は、[Options for Compiling NDB Cluster](#) を参照してください。

ビルドプロセスが完了したら、コンパイルされたバイナリを含む Zip アーカイブを作成できます。Windows システムでこのタスクを実行するのに必要なコマンドについては、[セクション 2.9.4 「標準ソース配布を使用して MySQL をインストールする」](#) を参照してください。NDB Cluster バイナリは、生成されるアーカイブの `bin` ディレクトリにあり、これは `no-install` アーカイブと同等であり、同じ方法でインストールおよび構成できます。詳細は、[セクション 23.2.2.1 「バイナリリリースから Windows への NDB Cluster のインストール」](#) を参照してください。

23.2.2.3 Windows での NDB Cluster の初期起動

NDB Cluster の実行可能ファイルと必要な構成ファイルが配置されたら、クラスタ内のすべてのノードで NDB Cluster の実行可能ファイルを起動するだけで、クラスタの初期起動を実行できます。各クラスタノードプロセスは、それが配置されているホストコンピュータ上で別個に起動する必要があります。最初に管理ノード、次にデータノード、最後に SQL ノードを起動するようにしてください。

- 管理ノードホストでは、コマンド行から次のコマンドを発行して管理ノードプロセスを起動します。ここに示すような出力が表示されます。

```
C:\mysql\bin> ndb_mgmd
```

```
2010-06-23 07:53:34 [MgmtSrvr] INFO -- NDB Cluster Management Server. mysql-8.0.23-ndb-8.0.23
2010-06-23 07:53:34 [MgmtSrvr] INFO -- Reading cluster configuration from 'config.ini'
```

管理ノードプロセスは、ロギング出力をコンソールに出力し続けます。管理ノードは Windows サービスとして実行されていないため、これは正常な動作です。(Linux などの Unix に似たプラットフォームで NDB Cluster を使用している場合、Windows でのこの点での管理ノードのデフォルト動作は、Unix システムでのその動作とは逆であり、デフォルトでは Unix デーモンプロセスとして実行されます。この動作は、Windows で実行されている NDB Cluster データノードプロセスにも当てはまります。)このため、`ndb_mgmd.exe` が実行されているウィンドウを閉じないでください。閉じると、管理ノードプロセスが強制終了されます。(NDB Cluster プロセスを Windows サービスとしてインストールおよび実行する方法については、[セクション23.2.2.4「NDB Cluster プロセスを Windows サービスとしてインストール」](#)を参照してください。)

必須の `-f` オプションで、管理ノードにグローバル構成ファイル (`config.ini`) がある場所を指示します。このオプションの長形式は `--config-file` です。

重要

NDB Cluster 管理ノードは、`config.ini` から読み取る構成データをキャッシュします。構成キャッシュを作成すると、ほかの方法を強制しないかぎり、後続の起動時に `config.ini` ファイルを無視します。つまり、このファイル内のエラーが原因で管理ノードが起動に失敗した場合は、エラーを修正してから、管理ノードに `config.ini` を再度読み取らせる必要があります。これを行うには、コマンド行で `--reload` または `--initial` オプションを指定して `ndb_mgmd.exe` を起動します。これらのオプションは、どちらも構成キャッシュをリフレッシュする機能を持っています。

管理ノードの `my.ini` ファイルでこれらのオプションのいずれかを使用することは、必要ないか、または推奨されません。

`ndb_mgmd` で使用できるオプションに関する追加情報は、[セクション23.4.4「ndb_mgmd — NDB Cluster 管理サーバーデーモン」](#)および[セクション23.4.32「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#)を参照してください。

2. 各データノードホストで、ここに示すコマンドを実行してデータノードプロセスを起動します。

```
C:\mysqlbin> ndbd
2010-06-23 07:53:46 [ndbd] INFO -- Configuration fetched from 'localhost:1186', generation: 1
```

いずれの場合も、データノードプロセスによって生成される出力の最初の行は前の例で示したものと似ていますが、そのあとにロギング出力の行が追加されます。管理ノードと同様に、データノードは Windows サービスとして実行されていないため、これは正常な動作です。このため、データノードプロセスが実行されているコンソールウィンドウを閉じないでください。閉じると、`ndbd.exe` が強制終了されます。(詳細は[セクション23.2.2.4「NDB Cluster プロセスを Windows サービスとしてインストール」](#)を参照してください。)

3. SQL ノードをまだ起動しないでください。データノードの起動(しばらく時間がかかることがあります)が完了するまでは、SQL ノードをクラスターに接続できません。代わりに、管理ノードホスト上の新しいコンソールウィンドウで、NDB Cluster 管理クライアント `ndb_mgm.exe` を起動します。これは、管理ノードホスト上の `C:\mysql\bin` 内にあるはずですが、(CTRL+C を入力して `ndb_mgmd.exe` が実行されているコンソールウィンドウを再利用しないでください。これを行うと管理ノードが強制終了されます。)生成される出力は次のようになります。

```
C:\mysqlbin> ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm>
```

`ndb_mgm>` というプロンプトが表示されたら、これは管理クライアントが NDB Cluster 管理コマンドを受信する準備ができていることを示します。管理クライアントのプロンプトで `ALL STATUS` と入力すると、データノードの起動時のステータスを確認できます。このコマンドによって、データノードの起動シーケンスのレポートが実行され、次のように表示されます。

```
ndb_mgm> ALL STATUS
Connected to Management Server at: localhost:1186
Node 2: starting (Last completed phase 3) (mysql-8.0.23-ndb-8.0.23)
Node 3: starting (Last completed phase 3) (mysql-8.0.23-ndb-8.0.23)

Node 2: starting (Last completed phase 4) (mysql-8.0.23-ndb-8.0.23)
Node 3: starting (Last completed phase 4) (mysql-8.0.23-ndb-8.0.23)
```



```
Node 2: Started (version 8.0.23)
Node 3: Started (version 8.0.23)

ndb_mgm>
```

注記

管理クライアントで発行されるコマンドでは大文字と小文字が区別されません。ここでは、コマンドの標準形式として大文字を使用しますが、`ndb_mgm` クライアントに入力するときに、この表記法に従う必要はありません。詳細は、[セクション23.5.1「NDB Cluster 管理クライアントのコマンド」](#)を参照してください。

`ALL STATUS` によって生成される出力は、データノードの起動速度、使用している NDB Cluster ソフトウェアのリリースバージョン番号、およびその他の要因に応じて、ここに表示される内容とは異なる可能性があります。重要なのは、両方のデータノードの起動を確認したときに、SQL ノードの起動準備が整うことです。

`ndb_mgm.exe` を実行したままにできます。NDB Cluster のパフォーマンスに悪影響はなく、次の手順でこれを使用して、SQL ノードが起動後にクラスタに接続されていることを確認します。

4. SQL ノードホストとして指定されたコンピュータで、コンソールウィンドウを開き、NDB Cluster バイナリを展開したディレクトリに移動します (この例に従っている場合は、`C:\mysql\bin` です)。

SQL ノードを起動するには、ここに示すように、コマンド行で `mysqld.exe` を起動します。

```
C:\mysql\bin> mysqld --console
```

`--console` オプションによって、コンソールにロギング情報が書き込まれます。これは問題が発生したときに役立つことがあります。(SQL ノードが問題なく実行されていることを確認できたら、SQL ノードを停止してから `--console` オプションを指定せずに起動すると、ロギングが通常どおり実行されるようになります。)

管理ノードホストの管理クライアント (`ndb_mgm.exe`) が実行されているコンソールウィンドウで、`SHOW` コマンドを入力します。ここに示すような出力が生成されます。

```
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
-----
[ndbd(NDB)] 2 node(s)
id=2 @198.51.100.30 (Version: 8.0.23-ndb-8.0.23, Nodegroup: 0, *)
id=3 @198.51.100.40 (Version: 8.0.23-ndb-8.0.23, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=1 @198.51.100.10 (Version: 8.0.23-ndb-8.0.23)

[mysqld(API)] 1 node(s)
id=4 @198.51.100.20 (Version: 8.0.23-ndb-8.0.23)
```

`SHOW ENGINE NDB STATUS` ステートメントを使用して、SQL ノードが `mysql` クライアント (`mysql.exe`) の NDB Cluster に接続されていることを確認することもできます。

NDB Cluster の `NDBCLUSTER` ストレージエンジンを使用してデータベースオブジェクトおよびデータを操作する準備ができました。詳細と例については、[セクション23.2.5「テーブルとデータを含む NDB Cluster の例」](#)を参照してください。

`ndb_mgmd.exe`、`ndbd.exe`、および `ndbmted.exe` を Windows サービスとしてインストールすることもできます。これを行う方法については、[セクション23.2.2.4「NDB Cluster プロセスを Windows サービスとしてインストール」](#)を参照してください。

23.2.2.4 NDB Cluster プロセスを Windows サービスとしてインストール

NDB Cluster が必要に応じて実行されていることを確認したら、管理ノードとデータノードを Windows サービスとしてインストールして、Windows が起動または停止されるたびにこれらのプロセスが自動的に起動および停止されるようにできます。これにより、適切な `SC START` および `SC STOP` コマンドを使用してコマンドラインから、または Windows グラフィカル `Services` ユーティリティを使用して、これらのプロセスを制御することもできます。`NET START` および `NET STOP` コマンドも使用できます。

プログラムを Windows サービスとしてインストールする場合は、通常、システム上の Administrator 権利を持つアカウントを使用する必要があります。

管理ノードを Windows 上のサービスとしてインストールするには、ここに示すように、管理ノードをホストするマシンのコマンド行で `--install` オプションを使用して `ndb_mgmd.exe` を起動します。

```
C:\> C:\mysql\bin\ndb_mgmd.exe --install
Installing service 'NDB Cluster Management Server'
as "C:\mysql\bin\ndbd.exe" "--service=ndb_mgmd"
Service successfully installed.
```

重要

NDB Cluster プログラムを Windows サービスとしてインストールする場合は、常に完全なパスを指定するようにしてください。そうしないと、サービスのインストールがエラー「**指定されたファイルが見つかりません**」で失敗する可能性があります。

`--install` オプションは、`ndb_mgmd.exe` に指定できるほかのオプションより先に使用する必要があります。ただし、このようなオプションはオプションファイルに指定することをお勧めします。オプションファイルが `ndb_mgmd.exe` `--help` の出力に示されるデフォルトの場所のいずれにも存在しない場合は、`--config-file` オプションを使用するとその場所を指定できます。

これで、このように管理サーバーを起動および停止できるようになります。

```
C:\> SC START ndb_mgmd
C:\> SC STOP ndb_mgmd
```

注記

NET コマンドを使用する場合は、次に示すように、わかりやすい名前を使用して Windows サービスとして管理サーバーを起動または停止することもできます:

```
C:\> NET START 'NDB Cluster Management Server'
The NDB Cluster Management Server service is starting.
The NDB Cluster Management Server service was started successfully.

C:\> NET STOP 'NDB Cluster Management Server'
The NDB Cluster Management Server service is stopping..
The NDB Cluster Management Server service was stopped successfully.
```

通常は、短いサービス名を指定するか、サービスのインストール時にデフォルトのサービス名を使用できるようにしてから、サービスの起動または停止時にその名前を参照する方が簡単です。 `ndb_mgmd` 以外のサービス名を指定するには、この例に示すように `--install` オプションを追加します。

```
C:\> C:\mysql\bin\ndb_mgmd.exe --install=mgmd1
Installing service 'NDB Cluster Management Server'
as "C:\mysql\bin\ndb_mgmd.exe" "--service=mgmd1"
Service successfully installed.
```

これで、このように指定した名前を使用してサービスを起動または停止できるようになります。

```
C:\> SC START mgmd1
C:\> SC STOP mgmd1
```

管理ノードサービスを削除するには、`SC DELETE service_name` を使用します:

```
C:\> SC DELETE mgmd1
```

または、次に示すように、`--remove` オプションを指定して `ndb_mgmd.exe` を起動します:

```
C:\> C:\mysql\bin\ndb_mgmd.exe --remove
Removing service 'NDB Cluster Management Server'
Service successfully removed.
```

デフォルト以外のサービス名を使用してサービスをインストールした場合は、次のように、`ndb_mgmd.exe --remove` オプションの値としてサービス名を渡します:

```
C:\> C:\mysql\bin\ndb_mgmd.exe --remove=mgmd1
```



```
Removing service 'mgmd1'  
Service successfully removed.
```

NDB Cluster データノードプロセスの Windows サービスとしてのインストールは、次に示すように、`ndbd.exe` (または `ndbmted.exe`) の `--install` オプションを使用して同様の方法で実行できます:

```
C:\> C:\mysql\bin\ndbd.exe --install  
Installing service 'NDB Cluster Data Node Daemon' as ""C:\mysql\bin\ndbd.exe" "--service=ndbd"  
Service successfully installed.
```

これで、次の例に示すように、データノードを起動または停止できます:

```
C:\> SC START ndbd
```

```
C:\> SC STOP ndbd
```

データノードサービスを削除するには、`SC DELETE service_name` を使用します:

```
C:\> SC DELETE ndbd
```

または、次に示すように、`--remove` オプションを指定して `ndbd.exe` を起動します:

```
C:\> C:\mysql\bin\ndbd.exe --remove  
Removing service 'NDB Cluster Data Node Daemon'  
Service successfully removed.
```

`ndb_mgmd.exe` (および `mysqld.exe`) と同様に、`ndbd.exe` を Windows サービスとしてインストールするときは、このようにサービスの名前を `--install` の値として指定し、サービスの起動または停止時にそれを使用することもできます。

```
C:\> C:\mysql\bin\ndbd.exe --install=dnode1  
Installing service 'dnode1' as ""C:\mysql\bin\ndbd.exe" "--service=dnode1"  
Service successfully installed.
```

```
C:\> SC START dnode1
```

```
C:\> SC STOP dnode1
```

データノードサービスのインストール時にサービス名を指定した場合は、次に示すように、この名前を削除するときにも使用できます:

```
C:\> SC DELETE dnode1
```

または、次に示すように、`ndbd.exe --remove` オプションの値としてサービス名を渡すこともできます:

```
C:\> C:\mysql\bin\ndbd.exe --remove=dnode1  
Removing service 'dnode1'  
Service successfully removed.
```

SQL ノードの Windows サービスとしてのインストール、サービスの起動、サービスの停止およびサービスの削除は、`mysqld --install`、`SC START`、`SC STOP`、`SC DELETE`(または `mysqld --remove`) を使用して同様の方法で実行されます。NET コマンドを使用して、サービスを起動または停止することもできます。追加情報については、[セクション2.3.4.8「Windows のサービスとして MySQL を起動する」](#)を参照してください。

23.2.3 NDB Cluster の初期構成

このセクションでは、構成ファイルを作成および編集して、インストールされた NDB Cluster の手動構成について説明します。

NDB Cluster には、別のアプリケーションでテキストファイルを編集しなくても構成を実行するために使用できる GUI インストーラも用意されています。詳細は、[セクション23.2.8「NDB Cluster Auto-Installer \(サポートされなくなりました\)」](#)を参照してください。

4 ノードの 4 ホスト NDB Cluster ([クラスタノードとホストコンピュータ](#) を参照) の場合、4 つの構成ファイルをノードホストごとに 1 つずつ書き込む必要があります。

- 個々のデータノードまたは SQL ノードには `my.cnf` ファイルが必要です。このファイルは、管理ノードが配置されたノードを指定する接続文字列と、このホスト (データノードをホストしているマシン) の MySQL サーバーに対して `NDBCLUSTER` ストレージエンジンを有効にするように指示する行の 2 つの情報を提供します。

接続文字列の詳細は、[セクション23.3.3.3「NDB Cluster 接続文字列」](#)を参照してください。

- 管理ノードには、保持するフラグメントレプリカの数、各データノードのデータおよびインデックスに割り当てるメモリー量、データノードの場所、各データノードのディスクへのデータの保存場所および SQL ノードの検索場所を示す `config.ini` ファイルが必要です。

データノードと SQL ノードの構成。データノードに必要な `my.cnf` ファイルはかなり単純です。この構成ファイルは、`/etc` ディレクトリに配置され、任意のテキストエディタを使用して編集できます。(このファイルが存在しない場合は作成してください。) 例:

```
shell> vi /etc/my.cnf
```

注記

ここでは、`vi` を使用してこのファイルを作成しますが、どのテキストエディタでも同じように機能します。

このセットアップ例の各データノードおよび SQL ノードでは、`my.cnf` はこのようになります。

```
[mysqld]
# Options for mysqld process:
ndbcluster          # run NDB storage engine

[mysql_cluster]
# Options for NDB Cluster processes:
ndb-connectstring=198.51.100.10 # location of management server
```

上記の情報を入力したら、このファイルを保存してテキストエディタを終了します。これを、データノード「A」、データノード「B」、および SQL ノードをホストしているマシンで実行します。

重要

上記の `my.cnf` ファイルの `[mysqld]` および `[mysql_cluster]` セクションの `ndbcluster` および `ndb-connectstring` パラメータを使用して `mysqld` プロセスを起動したあとは、クラスタが実際に起動しないと、`CREATE TABLE` または `ALTER TABLE` ステートメントを実行できません。そうしないと、これらのステートメントはエラーで失敗します。これは意図的なものです。

管理ノードの構成。管理ノードの構成では、最初のステップとして構成ファイルを配置するディレクトリを作成し、その後構成ファイル自体を作成します。例 (`root` として実行します):

```
shell> mkdir /var/lib/mysql-cluster
shell> cd /var/lib/mysql-cluster
shell> vi config.ini
```

ここで使用する典型的なセットアップでは、`config.ini` ファイルは次のようになります。

```
[ndbd default]
# Options affecting ndbd processes on all data nodes:
NoOfReplicas=2 # Number of fragment replicas
DataMemory=98M # How much memory to allocate for data storage

[ndb_mgmd]
# Management process options:
HostName=198.51.100.10 # Hostname or IP address of management node
DataDir=/var/lib/mysql-cluster # Directory for management node log files

[ndbd]
# Options for data node "A":
# (one [ndbd] section per data node)
HostName=198.51.100.30 # Hostname or IP address
NodeId=2 # Node ID for this data node
DataDir=/usr/local/mysql/data # Directory for this data node's data files

[ndbd]
# Options for data node "B":
HostName=198.51.100.40 # Hostname or IP address
NodeId=3 # Node ID for this data node
DataDir=/usr/local/mysql/data # Directory for this data node's data files
```

```
[mysqld]
# SQL node options:
HostName=198.51.100.20      # Hostname or IP address
                             # (additional mysqld connections can be
                             # specified for this node for various
                             # purposes such as running ndb_restore)
```

注記

`world` データベースは、<https://dev.mysql.com/doc/index-other.html> からダウンロードできます。

すべての構成ファイルを作成し、最小限のオプションを指定したら、クラスタの起動とすべてのプロセスの実行確認に進む準備が整います。これを行う方法については、[セクション23.2.4「NDB Cluster の初期起動」](#)で説明します。

使用可能な NDB Cluster 構成パラメータとその使用方法の詳細は、[セクション23.3.3「NDB Cluster 構成ファイル」](#) および [セクション23.3「NDB Cluster の構成」](#) を参照してください。バックアップの作成に関連する NDB Cluster の構成については、[セクション23.5.8.3「NDB Cluster バックアップの構成」](#) を参照してください。

注記

クラスタ管理ノードのデフォルトポートは 1186 です。データノードのデフォルトポートは 2202 です。ただし、クラスタではすでに開放されているポートからデータノードのポートを自動的に割り当てることができます。

23.2.4 NDB Cluster の初期起動

構成後のクラスタを起動することは、それほど難しいことはありません。各クラスタノードプロセスは、配置されているホストで別個に起動する必要があります。最初に管理ノード、次にデータノード、最後に SQL ノードを起動してください。

1. 管理ホストで、システムシェルから次のコマンドを発行して管理ノードプロセスを起動します。

```
shell> ndb_mgmd -f /var/lib/mysql-cluster/config.ini
```

`ndb_mgmd` をはじめて起動するときは、`-f` または `--config-file` オプションを使用してその構成ファイルの場所を指定する必要があります。(詳細は、[セクション23.4.4「ndb_mgmd — NDB Cluster 管理サーバーデーモン」](#)を参照してください。)

`ndb_mgmd` で使用できる追加のオプションについては、[セクション23.4.32「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#)を参照してください。

2. 個々のデータノードホストで、このコマンドを実行して `ndbd` プロセスを起動します。

```
shell> ndbd
```

3. SQL ノードを配置するクラスタホストで RPM ファイルを使用して MySQL をインストールした場合は、付属の起動スクリプトを使用して SQL ノードの MySQL サーバードキュメントを起動できます (起動するようにしてください)。

すべてが順調に進み、クラスタが正しくセットアップされると、クラスタは使用できる状態になります。これをテストするには、`ndb_mgm` 管理ノードクライアントを起動します。ここに示すような出力が表示されますが、使用している MySQL の特定のバージョンによって出力内容がやや異なる場合があります。

```
shell> ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
-----
[ndbd(NDB)] 2 node(s)
id=2 @198.51.100.30 (Version: 8.0.23-ndb-8.0.23, Nodegroup: 0, *)
id=3 @198.51.100.40 (Version: 8.0.23-ndb-8.0.23, Nodegroup: 0)
[ndb_mgmd(MGM)] 1 node(s)
```

```
id=1 @198.51.100.10 (Version: 8.0.23-ndb-8.0.23)
```

```
[mysqld(API)] 1 node(s)
```

```
id=4 @198.51.100.20 (Version: 8.0.23-ndb-8.0.23)
```

ここでは、SQL ノードは[mysqld(API)]として参照されます。これは、mysqld プロセスが NDB Cluster API ノードとして機能しているという事実を反映しています。

注記

SHOW の出力で特定の NDB Cluster SQL またはその他の API ノードに表示される IP アドレスは、どの管理ノードにも接続せずに、SQL または API ノードがクラスタデータノードに接続するために使用するアドレスです。

NDB Cluster 内のデータベース、テーブル、およびデータを操作する準備ができました。簡単な説明については、[セクション23.2.5「テーブルとデータを含む NDB Cluster の例」](#)を参照してください。

23.2.5 テーブルとデータを含む NDB Cluster の例

注記

このセクションの情報は、Unix プラットフォームと Windows プラットフォームの両方で実行されている NDB Cluster に適用されます。

NDB Cluster でのデータベーステーブルおよびデータの操作は、標準の MySQL での作業とほとんど異なりません。留意すべき重要なポイントが 2 つあります。

- クラスタ内でレプリケートされるテーブルでは、NDBCLUSTER ストレージエンジンを使用する必要があります。これを指定するには、テーブルの作成時に ENGINE=NDBCLUSTER または ENGINE=NDB オプションを使用します。

```
CREATE TABLE tbl_name (col_name column_definitions) ENGINE=NDBCLUSTER;
```

または、異なるストレージエンジンを使用する既存のテーブルに対して ALTER TABLE を使用して、NDBCLUSTER を使用するようにテーブルを変更します。

```
ALTER TABLE tbl_name ENGINE=NDBCLUSTER;
```

- すべての NDBCLUSTER テーブルには主キーがあります。テーブルの作成時にユーザーが主キーを定義しなかった場合は、NDBCLUSTER ストレージエンジンが隠し主キーを自動的に生成します。このようなキーは、ほかのテーブルインデックスと同じサイズの領域を占有します。(これらの自動的に作成されるインデックスを格納する十分なメモリがないために問題が発生することは、珍しくありません。)

mysqldump の出力を使用して既存のデータベースからテーブルをインポートする場合は、SQL スクリプトをテキストエディタで開いて、テーブル作成ステートメントに ENGINE オプションを追加するか、既存の ENGINE オプションを置き換えることができます。NDB Cluster をサポートしていない別の MySQL サーバーに world サンプルデータベースがあり、City テーブルをエクスポートするとします:

```
shell> mysqldump --add-drop-table world City > city_table.sql
```

結果の city_table.sql ファイルには、次のテーブル作成ステートメント (およびテーブルデータのインポートに必要な INSERT ステートメント) が含まれます:

```
DROP TABLE IF EXISTS `City`;  
CREATE TABLE `City` (  
  `ID` int(11) NOT NULL auto_increment,  
  `Name` char(35) NOT NULL default "",  
  `CountryCode` char(3) NOT NULL default "",  
  `District` char(20) NOT NULL default "",  
  `Population` int(11) NOT NULL default '0',  
  PRIMARY KEY (`ID`)  
) ENGINE=MyISAM DEFAULT CHARSET=latin1;  
  
INSERT INTO `City` VALUES (1,'Kabul','AFG','Kabol',1780000);  
INSERT INTO `City` VALUES (2,'Qandahar','AFG','Qandahar',237500);  
INSERT INTO `City` VALUES (3,'Herat','AFG','Herat',186800);  
(remaining INSERT statements omitted)
```

MySQL がこのテーブルに対して **NDBCLUSTER** ストレージエンジンを使用するか確認する必要があります。これを行うには 2 つの方法があります。1 つは、テーブル定義をクラスタデータベースにインポートする前に変更することです。例として **City** テーブルを使用し、定義の **ENGINE** オプションを次のように変更します。

```
DROP TABLE IF EXISTS `City`;
CREATE TABLE `City` (
  `ID` int(11) NOT NULL auto_increment,
  `Name` char(35) NOT NULL default "",
  `CountryCode` char(3) NOT NULL default "",
  `District` char(20) NOT NULL default "",
  `Population` int(11) NOT NULL default '0',
  PRIMARY KEY (`ID`)
) ENGINE=NDBCLUSTER DEFAULT CHARSET=latin1;

INSERT INTO `City` VALUES (1,'Kabul','AFG','Kabol',1780000);
INSERT INTO `City` VALUES (2,'Qandahar','AFG','Qandahar',237500);
INSERT INTO `City` VALUES (3,'Herat','AFG','Herat',186800);
(remaining INSERT statements omitted)
```

クラスタ化されたデータベースの一部になる個々のテーブルの定義で、これを行う必要があります。これを行うもっとも簡単な方法は、定義を含むファイルに対して検索置換を実行し、**TYPE=engine_name** または **ENGINE=engine_name** のすべてのインスタンスを **ENGINE=NDBCLUSTER** に置換することです。ファイルを変更したくない場合は、変更していないファイルを使用してテーブルを作成してから、**ALTER TABLE** を使用してストレージエンジンを変更します。詳細はこのセクションで後述します。

クラスタの SQL ノードで **world** という名前のデータベースがすでに作成されていることを前提に、**mysql** コマンド行クライアントを使用して **city_table.sql** を読み取り、対応するテーブルを通常の方法で作成して移入します。

```
shell> mysql world < city_table.sql
```

非常に重要な留意点として、前述のコマンドは SQL ノードを実行しているホスト (この場合は、IP アドレスが **198.51.100.20** のマシン) で実行する必要があります。

SQL ノード上に **world** データベース全体のコピーを作成するには、非クラスタサーバー上の **mysqldump** を使用して、**world.sql** という名前のファイル (**/tmp** ディレクトリなど) にデータベースをエクスポートします。次に、前述のとおりテーブル定義を変更し、このようにファイルをクラスタの SQL ノードにインポートします。

```
shell> mysql world < /tmp/world.sql
```

ファイルを別の場所に保存する場合は、それに合わせて前述の手順を調整してください。

SQL ノードでの **SELECT** クエリーの実行は、MySQL サーバーのほかのインスタンスでの実行と違いはありません。コマンド行からクエリーを実行するには、最初に通常の方法で (**Enter password:** プロンプトで **root** パスワードを指定して) MySQL Monitor にログインする必要があります。

```
shell> mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 8.0.23-ndb-8.0.23

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

ここでは、MySQL サーバーの **root** アカウントを使用し、MySQL サーバーのインストールに関する標準的なセキュリティ対策 (強力な **root** パスワードの設定を含む) に従っていると仮定します。詳細は、[セクション2.10.4「初期MySQLアカウントの保護」](#)を参照してください。

NDB Cluster ノードは、相互にアクセスするときに MySQL 特権システムを使用しないことを考慮する価値があります。MySQL ユーザーアカウント (**root** アカウントを含む) の設定または変更は、SQL ノードにアクセスするアプリケーションにのみ影響し、ノード間のやり取りには影響しません。詳細は、[セクション23.5.17.2「NDB Cluster および MySQL の権限」](#)を参照してください。

SQL スクリプトをインポートする前にテーブル定義の **ENGINE** 句を変更しなかった場合は、この時点で次のステートメントを実行してください。

```
mysql> USE world;
mysql> ALTER TABLE City ENGINE=NDBCLUSTER;
mysql> ALTER TABLE Country ENGINE=NDBCLUSTER;
```

```
mysql> ALTER TABLE CountryLanguage ENGINE=NDBCLUSTER;
```

データベースの選択とそのデータベース内のテーブルに対する **SELECT** クエリーの実行も、MySQL Monitor の終了と同様に通常の方法で行います。

```
mysql> USE world;
mysql> SELECT Name, Population FROM City ORDER BY Population DESC LIMIT 5;
+-----+-----+
| Name   | Population |
+-----+-----+
| Bombay | 10500000  |
| Seoul  | 9981619   |
| São Paulo | 9968485  |
| Shanghai | 9696300  |
| Jakarta | 9604900  |
+-----+-----+
5 rows in set (0.34 sec)

mysql> \q
Bye

shell>
```

MySQL を使用するアプリケーションは、標準の API を使用して **NDB** テーブルにアクセスできます。アプリケーションは、管理ノードやデータノードではなく、SQL ノードにアクセスする必要があります。この簡単な例では、ネットワーク上の別の場所にある Web サーバーで実行されている PHP 5.X の **mysqli** 拡張を使用して、ここに示す **SELECT** ステートメントを実行する方法を示しています。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type"
        content="text/html; charset=iso-8859-1">
  <title>SIMPLE mysqli SELECT</title>
</head>
<body>
<?php
# connect to SQL node:
$link = new mysqli("198.51.100.20", 'root', 'root_password', 'world');
# parameters for mysqli constructor are:
# host, user, password, database

if( mysqli_connect_errno() )
  die("Connect failed: " . mysqli_connect_error());

$query = "SELECT Name, Population
          FROM City
          ORDER BY Population DESC
          LIMIT 5";

# if no errors...
if( $result = $link->query($query) )
{
  ?>
<table border="1" width="40%" cellpadding="4" cellspacing="1">
  <tbody>
  <tr>
    <th width="10%">City</th>
    <th>Population</th>
  </tr>
  <?
  # then display the results...
  while($row = $result->fetch_object())
    printf("<tr>\n  <td align='center'>%s</td><td>%d</td>\n</tr>\n",
          $row->Name, $row->Population);
  ?>
  </tbody>
</table>
<?
# ...and verify the number of rows that were retrieved
printf("<p>Affected rows: %d</p>\n", $link->affected_rows);
}
```



```
else
# otherwise, tell us what went wrong
echo mysqli_error());

# free the result set and the mysqli connection object
$result->close();
$link->close();
?>
</body>
</html>
```

ここでは、Web サーバーで実行されているプロセスが SQL ノードの IP アドレスに到達できると仮定します。

同様の方法で、MySQL C API、Perl-DBI、Python-mysql、または MySQL Connector を使用して、MySQL で通常実行する同じデータの定義や操作のタスクを実行できます。

23.2.6 NDB Cluster の安全なシャットダウンと再起動

クラスタをシャットダウンするには、管理ノードをホストしているマシン上のシェルで次のコマンドを入力します。

```
shell> ndb_mgm -e shutdown
```

この `-e` オプションは、シェルから `ndb_mgm` クライアントにコマンドを渡すために使用されます。(このオプションの詳細は、[セクション23.4.32「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#)を参照してください。) このコマンドによって、`ndb_mgm`、`ndb_mgmd`、およびすべての `ndbd` または `ndbmt` プロセスが適切な方法で終了します。SQL ノードは `mysqladmin shutdown` およびその他の方法で終了できます。Windows プラットフォームでは、SQL ノードが Windows サービスとしてインストールされている場合、`SC STOP service_name` または `NET STOP service_name` を使用できます。

Unix プラットフォームでクラスタを再起動するには、次のコマンドを実行します。

- 管理ホスト (このセットアップ例では `198.51.100.10`):

```
shell> ndb_mgmd -f /var/lib/mysql-cluster/config.ini
```

- 個々のデータノードホスト (`198.51.100.30` および `198.51.100.40`):

```
shell> ndbd
```

- `ndb_mgm` クライアントを使用して、両方のデータノードが正常に起動したか確認します。
- SQL ホスト (`198.51.100.20`):

```
shell> mysql_safe &
```

Windows プラットフォームでは、デフォルトのサービス名 ([セクション23.2.2.4「NDB Cluster プロセスを Windows サービスとしてインストール」](#)を参照) を使用してすべての NDB Cluster プロセスを Windows サービスとしてインストールしたと仮定すると、次のようにクラスタを再起動できます:

- 管理ホスト (このセットアップ例では `198.51.100.10`) で、次のコマンドを実行します。

```
C:\> SC START ndb_mgmd
```

- 個々のデータノードホスト (`198.51.100.30` および `198.51.100.40`) で、次のコマンドを実行します。

```
C:\> SC START ndbd
```

- 管理ノードホストで、`ndb_mgm` クライアントを使用して管理ノードと両方のデータノードが正常に起動したか確認します ([セクション23.2.2.3「Windows での NDB Cluster の初期起動」](#)を参照してください)。
- SQL ノードホスト (`198.51.100.20`) で、次のコマンドを実行します。

```
C:\> SC START mysql
```

本番設定では、通常、クラスタを完全にシャットダウンすることは望ましくありません。多くの場合、構成変更やクラスタのハードウェアまたはソフトウェア (あるいはその両方) のアップグレードを実行して個々のホストマシンをシャットダウンする必要が生じた場合でも、クラスタのローリング再起動を実行することで、クラスタ全体をシャットダウンせずに処理を完了できます。この実行方法の詳細は、[セクション23.5.5「NDB Cluster のローリング再起動の実行」](#)を参照してください。

23.2.7 NDB Cluster のアップグレードおよびダウングレード

このセクションでは、アップグレードとダウングレードの実行に関する NDB Cluster 8.0 のさまざまなリリース間の NDB Cluster ソフトウェアとテーブルファイルの互換性、および互換性マトリクスとノートについて説明します。アップグレードまたはダウングレードを試みる前に、NDB Cluster のインストールおよび構成に精通している必要があります。セクション23.3「NDB Cluster の構成」を参照してください。

SQL DDL ステートメントを含むスキーマ操作は、データノードの再起動中、およびクラスターのオンラインアップグレードまたはダウングレード中には実行できません。オンラインアップグレードの実行に使用されるローリング再起動手順に関するその他の情報は、セクション23.5.5「NDB Cluster のローリング再起動の実行」を参照してください。

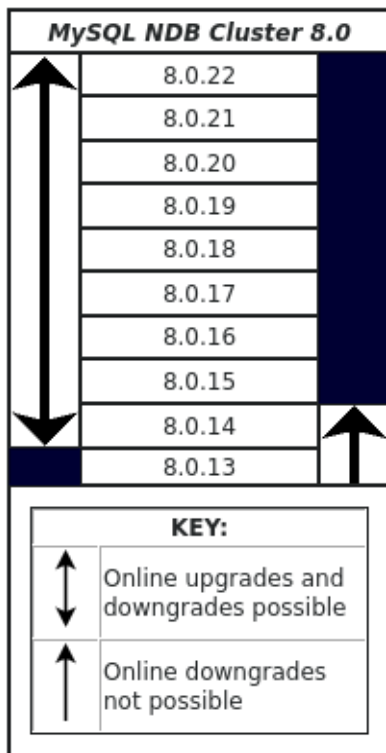
重要

このセクションでは、MySQL バージョン間の `NDBCLUSTER` に関する互換性のみを考慮しますが、考慮すべき問題はほかにもある場合があります。NDB Cluster ソフトウェアのアップグレードまたはダウングレードを試みる前に、ほかの MySQL ソフトウェアのアップグレードまたはダウングレードと同様に、移行元および移行先の MySQL バージョンについて、MySQL マニュアルの関連する部分を確認することを強くお勧めします。セクション 2.11「MySQL のアップグレード」を参照してください。

ここに示すテーブルは、NDB 8.0 の異なるリリース間での NDB Cluster のアップグレードおよびダウングレードの互換性に関する情報を示しています。NDB Cluster 8.0 リリースシリーズへのアップグレード、NDB Cluster 8.0 リリースシリーズからのダウングレード、または NDB Cluster 8.0 リリースシリーズ内でのダウングレードに関する追加のノートは、次のテーブルのあとにあります。

アップグレードとダウングレード、NDB Cluster 8.0

図 23.5 NDB Cluster のアップグレードおよびダウングレードの互換性、MySQL NDB Cluster 8.0



バージョンのサポート。 NDB Cluster 8.0 (8.0.19 以降) の GA リリースへのアップグレードでは、次のバージョンの NDB Cluster がサポートされています:

- NDB Cluster 7.6: NDB 7.6.4 以降
- NDB Cluster 7.5: NDB 7.5.4 以降
- NDB Cluster 7.4: NDB 7.4.6 以降

NDB 7.4 より前のリリースシリーズからアップグレードするには、最初にリストされているいずれかのバージョンにアップグレードしてから、そのバージョンから最新の NDB 8.0 リリースにアップグレードする必要があります。このような場合は、最初の手順として最新の NDB 7.6 リリースにアップグレードすることをお勧めします。

既知の問題。 NDB 8.0 リリース間でアップグレードする場合、次の問題が発生することがわかっています:

- NDB 8.0.14 から以前のリリースへのオンラインダウングレードはサポートされていません。NDB 8.0.14 で作成されたテーブルは、以前のリリースとの下位互換性がありません。これは、MySQL データディクショナリを完全にサポートするために、NDB テーブルによって実装される追加のメタデータプロパティの使用方法が変更されたためです。

詳細は、[NDB テーブルの追加メタデータの変更](#)を参照してください。第14章「MySQL データディクショナリ」も参照してください。

- 以前のリリースシリーズ ([Distributed Privileges Using Shared Grant Tables](#) を参照) で実装された MySQL サーバ間で共有される分散特権は、NDB Cluster 8.0 ではサポートされていません。NDB 8.0.16 で提供される `mysql` は、起動時に、あとで NDB ストレージエンジンを使用する付与テーブルの存在をチェックします。見つかった場合は、`InnoDB` を使用してこれらのローカルコピー (「シャドウテーブル」) を作成します。これは NDB Cluster に接続されている各 MySQL サーバに当てはまります。NDB Cluster SQL ノードとして機能するすべての MySQL サーバでこれが実行されたあと、NDB Cluster 配布に付属の `ndb_drop_table` ユーティリティを使用して、NDB 付与テーブルを安全に削除できます。次に例を示します:

```
ndb_drop_table -d mysql user db columns_priv tables_priv proxies_priv procs_priv
```

NDB 付与テーブルは安全に保持できますが、アクセス制御には使用されず、事実上無視されます。

NDB 8.0 で使用される MySQL 特権システムの詳細は、[セクション23.5.12「NDB_STORED_USER での分散 MySQL 権限」](#) および [セクション6.2.3「付与テーブル」](#) を参照してください。

- NDB 8.0.18 では、バイナリ構成ファイル形式が拡張され、以前のバージョンよりも多くのノードをサポートするようになりました。新しい形式は、8.0.17 および古いノードからはアクセスできませんが、新しい管理サーバは古いノードを検出し、適切な形式を使用してそれらと通信できます。

8.0.17 以前から NDB 8.0.18 以降へのアップグレードは、この点で問題になるべきではありません。NDB 8.0.18 以降から 8.0.17 以前へのダウングレードの場合、古い管理サーバは新しいバイナリ構成ファイル形式を読み取ることができないため、手動の介入が必要になります。このようなダウングレードを実行する場合は、古い NDB ソフトウェアバージョンを使用して管理を開始する前に、キャッシュされたバイナリ構成ファイルを削除し、管理サーバでプレーンテキスト構成ファイルを読み取ることができるようにする必要があります。または、`--initial` オプションを使用して古い管理サーバを起動することもできます (再度、`config.ini` を使用可能にする必要があります)。クラスタで複数の管理サーバを使用する場合は、管理サーババイナリごとに次の 2 つのいずれかを実行する必要があります。

また、ノード数の増加のサポートに関連して、NDB 8.0.18 でデータノード LCP `Sysfile` に実装された互換性のない変更のため、NDB 8.0.18 (以降) から以前のリリースへのオンラインダウングレードを実行して、`--initial` オプションを使用してすべてのデータノードを再起動する必要があります。

NDB 7.6.4 より前のリリースを NDB 8.0 リリースにアップグレードする場合も、`--initial` を使用してデータノードを再起動する必要があります。

- 48 を超えるデータノードを実行しているクラスタ、または 48 を超えるノード ID を使用しているデータノードを NDB 8.0.17 以前の NDB 8.0.18 に直接ダウングレードすることはサポートされていません。このような場合は、データノードの数を減らし、古い最大値を超えないように、48 以下のノード ID を使用するようすべてのデータノードの構成を変更する必要があります。
- NDB 8.0 から NDB 7.5 または NDB 7.4 にダウングレードする場合、まだ存在しないときは、クラスタ構成ファイルで `IndexMemory` の明示的な値を設定する必要があります。これは、NDB 8.0 が (NDB 7.6 で削除された) このパラメータを使用せず、デフォルトで 0 に設定するのに対し、NDB 7.5 および NDB 7.4 では必須であ

り、`IndexMemory` がゼロ以外の値に設定されていない場合、どちらのクラスタも「管理サーバーから無効な構成を受信しました ...」での起動を拒否するためです。

NDB 8.0 から NDB 7.6 へのダウングレードには、`IndexMemory` の設定は必要ありません。

- NDB 8.0.22 では、`config.ini` ファイル内の管理ノードおよびデータノードに対する IPv6 アドレス指定のサポートが追加されました。アップグレードの一環として IPv6 アドレスの使用を開始するには、次のステップを実行します:
 1. 通常の方法で、8.0.22 以降のバージョンの NDB Cluster ソフトウェアへのクラスタのアップグレードを実行します。
 2. `config.ini` ファイルで使用されているアドレスを IPv6 アドレスに変更します。
 3. クラスタのシステム再起動を実行します。

23.2.8 NDB Cluster Auto-Installer (サポートされなくなりました)

注記

この機能は NDB Cluster から削除され、サポートされなくなりました。詳しくは [セクション 23.1.4 「NDB Cluster の新機能」](#) をご覧ください。

このセクションでは、NDB Cluster 配布の一部として含まれる web ベースのグラフィカル構成インストーラについて説明します。ここでは、インストーラとその部品、ソフトウェア、およびインストーラの実行、GUI のナビゲート、およびインストーラを使用した 1 台以上のホストコンピュータでの NDB Cluster の設定と起動、または停止に関するその他の要件の概要について説明します。

NDB Cluster Auto-Installer は 2 つのコンポーネントで構成されています。フロントエンドは、Firefox や Microsoft Internet Explorer などの標準の Web ブラウザでロードおよび実行される Web ページとして実装される GUI クライアントです。バックエンドは、ローカルマシンまたはユーザーがアクセスできる別のホストで実行されるサーバープロセス (`ndb_setup.py`) です。

これら 2 つのコンポーネント (クライアントとサーバー) は、標準の HTTP 要求および応答を使用して相互に通信します。バックエンドは、バックエンドユーザーがアクセス権を付与した任意のホストで NDB Cluster ソフトウェアプログラムを管理できます。NDB Cluster ソフトウェアが別のホスト上にある場合、バックエンドは SSH を使用してアクセスします。

23.2.8.1 NDB Cluster 自動インストーラの要件

このセクションでは、NDB Cluster Auto-Installer を実行するためにサポートされているオペレーティングシステムとソフトウェア、必要なソフトウェア、およびその他の前提条件について説明します。

サポートされるプラットフォーム. NDB Cluster Auto-Installer は、最新バージョンの Linux、Windows、Solaris、および macOS 用の NDB 8.0 デистриビューションで使用できます。NDB Cluster および NDB Cluster 自動インストーラのプラットフォームサポートの詳細は、<https://www.mysql.com/support/supportedplatforms/cluster.html> を参照してください。

サポートされる web ブラウザ. web ベースのインストーラは、最新バージョンの Firefox および Microsoft Internet Explorer でサポートされます。また、Opera、Safari、および Chrome の最近のバージョンでも動作しますが、これらのブラウザとの互換性に関する詳細なテストは行われていません。

必要なソフトウェア (セットアップホスト). Auto-Installer が実行されているホストに次のソフトウェアをインストールする必要があります:

- Python 2.6 以上. Auto-Installer には、Python インタプリタと標準ライブラリが必要です。これらがシステムにまだインストールされていない場合は、システムのパッケージマネージャーを使用して追加できる可能性があります。それ以外の場合は、<http://python.org/download/> からダウンロードできます。
- Paramiko 2 以上. これがシステムパッケージマネージャから入手できない場合は、<http://www.lag.net/paramiko/> からダウンロードできます。
- Pycrypto バージョン 1.9 以上. この暗号化モジュールは Paramiko に必要であり、`pip install cryptography` を使用してインストール解除できます。pip がインストールされておらず、システムパッケージ管理を使用してモジュールを使用できない場合は、<https://www.dlitz.net/software/pycrypto/> からダウンロードできます。

構成ツールの Windows バージョンには、前のリストに示したすべてのソフトウェアが含まれているため、別個にインストールする必要はありません。

必要なソフトウェア (リモートホスト). NDB Cluster ノードを配備するリモートホストに必要なソフトウェアは SSH サーバーだけです。SSH サーバーは通常、Linux および Solaris システムにデフォルトでインストールされます。Windows では、いくつかの代替りとなるものが使用可能です。概要については、http://en.wikipedia.org/wiki/Comparison_of_SSH_servers を参照してください。

複数のホストを使用する場合の追加要件は、次のいくつかの段落で説明するように、SSH および適切なキーまたはユーザー資格証明を使用して任意のリモートホストに対して認証できることです:

認証とセキュリティー. リモートアクセスのための 3 つの基本的なセキュリティーまたは認証メカニズムを Auto-Installer で使用できます。ここでは、これらについて説明します:

- SSH. セキュアシェル接続を使用すると、バックエンドからリモートホストでのアクションを実行できるようになります。そのためには、リモートホストで SSH サーバーが実行されている必要があります。また、インストーラを実行しているオペレーティングシステムユーザーは、ユーザー名とパスワード、または公開キーと秘密キーを使用してリモートサーバーにアクセスできる必要があります。

重要

システムの `root` アカウントは安全性がきわめて低いため、リモートアクセスには決して使用しないでください。また、`mysqld` はシステムの `root` によって正常に起動できません。これらの理由やその他の理由から、システムの `root` ではく、ターゲットシステム上の通常のユーザーアカウントの SSH 資格証明を提供してください。この問題の詳細は、[セクション 6.1.5 「MySQL を通常ユーザーとして実行する方法」](#) を参照してください。

- HTTPS. デフォルトでは、Web ブラウザのフロントエンドとバックエンドの間のリモート通信は暗号化されません。つまり、ユーザー SSH パスワードなどの情報はクリアテキストとして送信され、すべてのユーザーが読み取ることができます。リモートクライアントからの通信を暗号化するには、バックエンドに証明書を用意し、フロントエンドが HTTP ではなく HTTPS を使用してバックエンドと通信する必要があります。HTTPS を有効にするには、自己署名証明書を発行するのもっとも簡単です。証明書を発行したあとは、それが使用されていることを確認する必要があります。これを行うには、`--use-https (-S)` および `--cert-file (-c)` オプションを指定してコマンドラインから `ndb_setup.py` を起動します。

サンプルの証明書ファイル `cfg.pem` が含まれており、デフォルトで使用されます。このファイルは、インストール共有ディレクトリの下に `mcc` ディレクトリにあります。Linux では、ファイルへのフルパスは通常 `/usr/share/mysql/mcc/cfg.pem` です。Windows システムでは、これは通常 `C:\Program Files\MySQL\MySQL Server 8.0\share\mcc\cfg.pem` です。デフォルトを使用できるということは、テスト目的で、`-S` オプションを使用してインストーラを起動し、ブラウザとバックエンド間の HTTPS 接続を使用できることを意味します。

Auto-Installer は、`ndb_setup.py` 実行可能ファイルを呼び出すユーザーのホームディレクトリに、特定のクラスター `mycluster01` の構成ファイルを `mycluster01.mcc` として保存します。このファイルは、(Fernet を使用して) ユーザーが指定したパスフレーズで暗号化されます。HTTP はパスフレーズをクリアなリモートホスト上の Auto-Installer にアクセスするには、常に HTTPS 接続を使用することを強くお勧めしますで転送するためです。

- 証明書ベースの認証. バックエンドの `ndb_setup.py` プロセスは、ローカルホストだけでなくリモートホストに対してもコマンドを実行できます。これは、バックエンドに接続した任意のユーザーがコマンドの実行方法を管理できることを意味します。バックエンドへの不要な接続を拒否するには、クライアントを認証するための証明書が必要です。この場合は、証明書がユーザーによって発行され、ブラウザにインストールされ、バックエンドで認証に使用できるようになっている必要があります。 `--ca-certs-file (-a)` オプションを指定して `ndb_setup.py` を起動することで、この要件を (パスワードまたはキー認証とともに、またはそのかわりに) 実施できます。

クライアントブラウザが Auto-Installer バックエンドと同じホスト上で実行されている場合、セキュアな認証は必要ありません。

さらに一般的な MySQL セキュリティー情報については、NDB Cluster を配備するときに考慮すべきセキュリティー上の考慮事項を考慮した [セクション 23.5.17 「NDB Cluster のセキュリティーの問題」](#) および [第 6 章 「セキュリティー」](#) も参照してください。

23.2.8.2 NDB Cluster Auto-Installer の使用

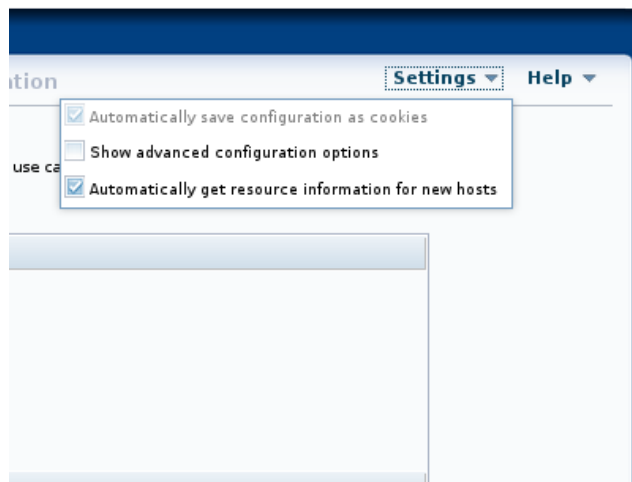
NDB Cluster の Auto-Installer インタフェースは複数のページで構成され、それぞれが NDB Cluster の構成および配備に使用されるプロセスのステップに対応しています。これらのページは、次の順序でリストされます:

- **Welcome:** 新しい NDB Cluster を構成するか、既存の NDB Cluster の構成を続行するかを選択して、Auto-Installer の使用を開始します。
- **「Define Cluster」:** クラスタ全体の基本的な情報 (名前、ホスト、負荷タイプなど) を設定します。ここでは、必要に応じてリモートホストへのアクセスに使用する SSH 認証のタイプを設定することもできます。
- **Define Hosts:** NDB Cluster プロセスを実行する予定のホストを特定します。
- **「Define Processes」:** 各クラスタホストに特定のタイプのプロセスを 1 つ以上割り当てます。
- **Define Parameters:** プロセスまたはプロセスのタイプの構成属性を設定します。
- **Deploy Configuration:** 以前に設定した構成でクラスタをデプロイし、デプロイしたクラスタを起動および停止します。

NDB Cluster インストーラの Settings および Help メニュー

これらのメニューは、ようこそ画面を除くすべての画面に表示されます。インストーラの設定および情報にアクセスできます。「設定」メニューの詳細を次に示します。

図 23.6 NDB Cluster の Auto-Installer Settings メニュー



「設定」メニューには、次のエントリがあります:

- **「Automatically save configuration as cookies」:** 構成情報 (ホスト名、プロセスデータ、パラメータ値など) をブラウザの Cookie として保存します。このオプションを選択すると、SSH パスワードを除くすべての情報が保存されます。つまり、ブラウザを終了して再起動し、前のセッションの終了時と同じ構成で作業を続行できます。このオプションはデフォルトで有効となっています。

SSH パスワードは保存されません。SSH パスワードを使用する場合は、新しいセッションの開始時にパスワードを指定する必要があります。

- **拡張構成オプションの表示:** デフォルトで、拡張構成パラメータが使用可能な場合に表示されます。

設定した拡張パラメータは、明示的に変更またはリセットされるまで構成ファイルで引き続き使用されます。これは、高度なパラメータが現在インストーラに表示されているかどうかは関係ありません。メニュー項目を無効にしても、パラメータの値はリセットされません。

「パラメータの定義 screen」 の個々のプロセスの拡張パラメータの表示を切り替えることもできます。

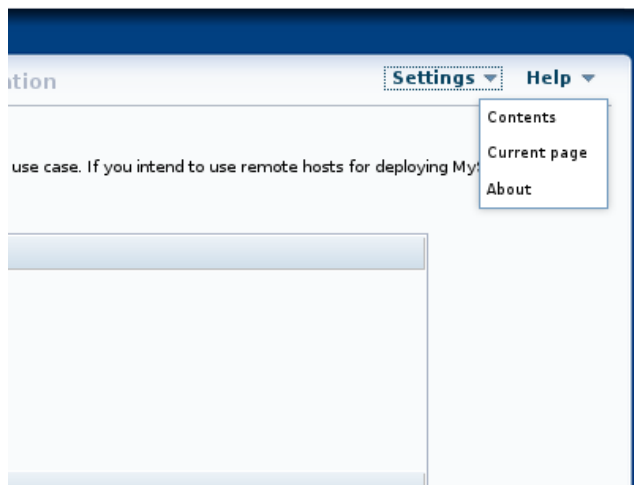
このオプションはデフォルトで無効になっています。

- 「Automatically get resource information for new hosts」: 新しいホストのハードウェアリソース情報を自動的にクエリーして、構成オプションおよび値を事前に移入します。この場合、提案された値は必須ではありませんが、インストーラの対応する編集オプションを使用して明示的に変更しないかぎり、その値が使用されます。

このオプションはデフォルトで有効となっています。

インストーラの Help メニューを次に示します:

図 23.7 NDB Cluster Auto-Installer の Help メニュー



「ヘルプ」メニューには、次のリストで説明する複数のオプションがあります。

- 目次: 組み込みユーザーガイドを表示します。これは、別のブラウザウィンドウで開くため、ワークフローを中断せずにインストーラと同時に使用できます。
- 「現在のページ」: 組み込みのユーザーガイドの、インストーラに現在表示されているページについて説明するセクションを開きます。
- バージョン情報: インストーラ名と、それが提供された NDB Cluster ディストリビューションのバージョン番号を表示するダイアログを開きます。

Auto-Installer では、ほとんどの入力ウィジェットでツールチップ形式のコンテキスト依存ヘルプも提供されます。

また、ほとんどの NDB 構成パラメータの名前は、オンラインドキュメントの説明にリンクされています。ドキュメントは別のブラウザウィンドウに表示されます。

次のセクションでは、Auto-Installer の起動について説明します。次の各セクションでは、これらの各ページの目的と機能について、前述の順序で詳しく説明します。

NDB Cluster Auto-Installer の起動

Auto-Installer は NDB Cluster ソフトウェアとともに提供されます。Auto-Installer のみを含む RPM パッケージと .deb パッケージは、多くの Linux ディストリビューションでも個別に使用できます。(セクション23.2「NDB Cluster のインストール」を参照してください。)

このセクションでは、インストーラの起動方法について説明します。これを行うには、`ndb_setup.py` 実行可能ファイルを起動します。

ユーザーおよび権限

`ndb_setup.py` は通常のユーザーとして実行する必要があります。そのために特別な権限は必要ありません。このプログラムは、`mysql` ユーザーとして、またはシステムの `root` または

管理者アカウントを使用して実行しないでください。実行すると、インストールが失敗する可能性があります。

`ndb_setup.py` は NDB Cluster インストールディレクトリ内の `bin` にあります。一般的な場所は、Linux システム上の `/usr/local/mysql/bin` または Windows システム上の `C:\Program Files\MySQL\MySQL Server 8.0\bin` です。これは、NDB Cluster ソフトウェアがインストールされているシステム上の場所とインストール方法によって異なる場合があります。

Windows では、NDB Cluster のインストールディレクトリで `setup.bat` を実行してインストーラを起動することもできます。コマンドラインから起動した場合、このバッチファイルは `ndb_setup.py` と同じオプションを受け入れません。

`ndb_setup.py` は、その動作に影響を与える複数のオプションを指定して起動できますが、通常はデフォルト設定を使用するだけで十分です。その場合は、次の 2 つの方法で `ndb_setup.py` を起動できます。

1. 端末で NDB Cluster `bin` ディレクトリに移動し、次のような追加の引数やオプションを指定せずにコマンド行から呼び出します:

```
shell> ndb_setup.py
Running out of install dir: /usr/local/mysql/bin
Starting web server on port 8081
URL is https://localhost:8081/welcome.html
deathkey=627876
Press CTRL+C to stop web server.
The application should now be running in your browser.
(Alternatively you can navigate to https://localhost:8081/welcome.html to start it)
```

これは、オペレーティングシステムに関係なく有効です。

2. ファイルブラウザで NDB Cluster `bin` ディレクトリ (Windows の場合は Windows エクスプローラ、Linux の場合は Konqueror、Dolphin、Nautilus など) に移動し、`ndb_setup.py` ファイルアイコンをアクティブにします (通常はダブルクリックします)。これは Windows で有効ですが、ほとんどの一般的な Linux デスクトップでも有効です。

Windows では、NDB Cluster のインストールディレクトリに移動して、`setup.bat` ファイルアイコンをアクティブにすることもできます。

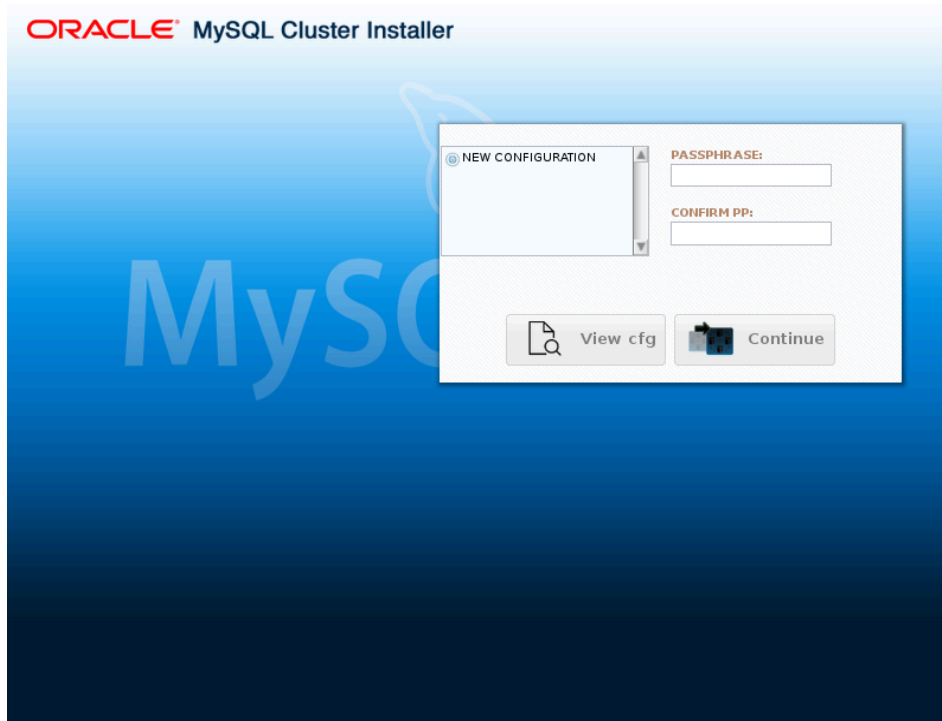
いずれの場合も、`ndb_setup.py` が呼び出されると、Auto-Installer 「ようこそ screen」がシステムのデフォルトの web ブラウザで開かれます。そうでない場合は、ページを開くことができます [http://localhost:8081/welcome.html](https://localhost:8081/welcome.html) または <https://localhost:8081/welcome.html> ブラウザで手動で。

場合によっては、接続に HTTPS を指定したり、Auto-Installer が実行する web サーバーを含む別のポートを指定するなど、インストーラにデフォルト以外の設定を使用することがあります。その場合は、必要なデフォルト値をオーバーライドして、1 つ以上の起動オプションで `ndb_setup.py` を起動する必要があります。NDB Cluster ソフトウェアディストリビューション内のこのようなプラットフォーム用に提供されている `setup.bat` ファイルを使用して、Windows システムで同じ起動オプションを使用できます。これはコマンドラインを使用して実行できますが、これらのオプションを使用してデスクトップまたはファイルブラウザからインストーラを起動する必要がある場合は、適切な起動を含むスクリプトまたはバッチファイルを作成し、ファイルブラウザでそのファイルアイコンをダブルクリックしてインストーラを起動することもできます。(Linux システムでは、最初にスクリプトファイルを実行可能にする必要があります。) リモートホストから Auto-Installer を使用する場合は、`-S` オプションの使用を開始するようにしてください。NDB Cluster Auto-Installer のこのオプションおよびその他の高度な起動オプションについては、[セクション 23.4.26 「ndb_setup.py — NDB Cluster 用ブラウザベースの Auto-Installer の起動 \(非推奨\)」](#) を参照してください。

NDB Cluster 自動インストーラのようこそ画面

ようこそ画面は、`ndb_setup.py` の起動時にデフォルトブラウザにロードされます。Auto-Installer がはじめて実行される時 (または何らかの理由で既存の構成が存在しない場合)、この画面は次のように表示されます:

図 23.8 「NDB Cluster 自動インストーラのようこそ」画面、最初の実行

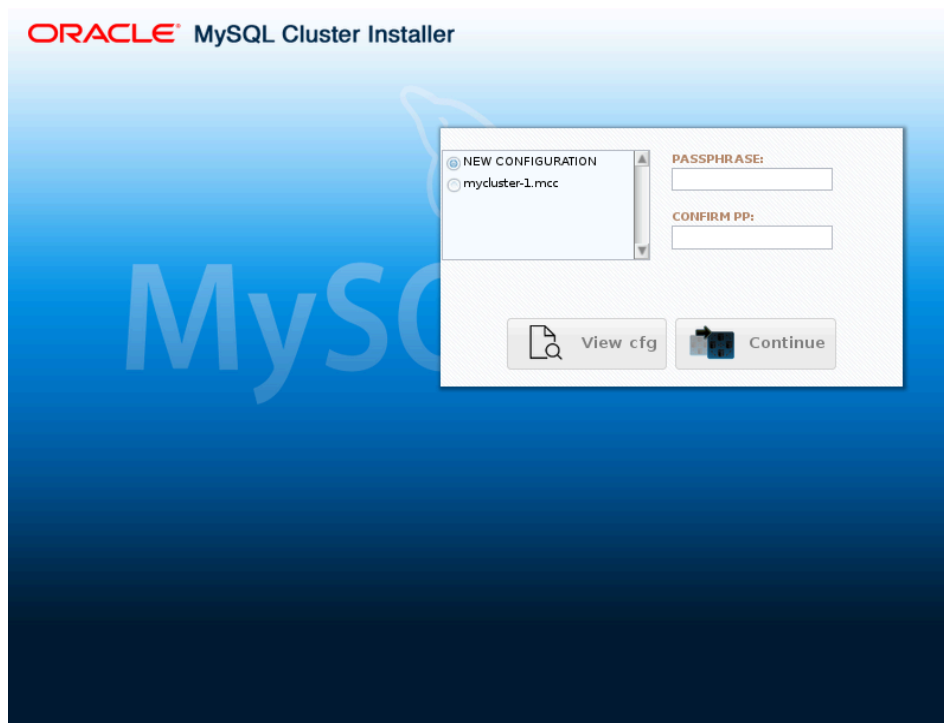


この場合、リストされるクラスタの選択肢は新しいクラスタの構成のみで、構成の表示ボタンと続行ボタンの両方が非アクティブになります。

新しい構成を作成するには、表示されたテキストボックスにパスフレーズを入力して確認します。これが完了したら、続行をクリックしてクラスタの定義画面に進み、新しいクラスタに名前を割り当てることができます。

Auto-Installer を使用してすでに 1 つ以上のクラスタを作成している場合は、それらが名前で一覧表示されます。この例は、[mycluster-1](#) という名前の既存のクラスタを示しています:

図 23.9 「NDB Cluster 自動インストーラのようこそ」画面 (以前に作成したクラスタ mycluster-1 を含む)



特定のクラスタの構成を表示して操作するには、リストでその名前の横にあるラジオボタンを選択し、作成に使用したパスフレーズを入力して確認します。これが正しく完了したら、構成の表示をクリックして、このクラスタ構成を表示および編集できます。

NDB Cluster 自動インストーラの Define Cluster 画面

クラスタの定義画面は、「ようこそ screen」の後に表示され、クラスタの一般プロパティの設定に使用されます。「Define Cluster」画面のレイアウトを次に示します。

図 23.10 「NDB Cluster 自動インストーラクラスタの定義」画面

この画面および後続の画面には、このセクションの後半で説明する「設定」および Help のメニューも含まれています。NDB Cluster インストーラの [Settings](#) および [Help](#) メニューを参照してください。

クラスタの定義画面では、クラスタの 3 種類のプロパティを設定できます: クラスタプロパティ、SSH プロパティおよびインストールプロパティ。

この画面で設定できるクラスタプロパティは、次のとおりです:

- クラスタ名: クラスタを識別する名前。この例では、`mycluster-1` です。名前は前の画面で設定され、ここでは変更できません。
- 「Host list」: クラスタプロセスを実行する 1 台以上のホストのカンマ区切りリスト。デフォルトでは、これは `127.0.0.1` です。リストにリモートホストを追加する場合は、SSH プロパティとして指定された資格証明を使用して接続する必要があります。
- 「Application type」: 次のいずれかを選択します。
 1. 「Simple testing」: 小規模なテストに対応する最小限のリソース使用。これはデフォルトです。本番環境を目的にしたものではありません。
 2. 「Web」: 特定のハードウェアに合わせてパフォーマンスを最大化します。
 3. 「Real-time」: 障害の発生したクラスタプロセスの検出にかかる時間を最小限に抑えるため、タイムアウトへの感度を最大限にしながらパフォーマンスを最大化します。
- 「Write load」: クラスタ全体で予測される書き込み数のレベルを選択します。次のいずれかのレベルを選択できます。
 1. 「Low」: 予想される負荷に、1 秒あたり 100 件未満の書き込みトランザクションが含まれます。
 2. 中: 予想される負荷には、100 ~ 1000 個の書き込みトランザクション/秒が含まれます。これがデフォルトです。
 3. 「High」: 予想される負荷に、1 秒あたり 1000 件を超える書き込みトランザクションが含まれます。

SSH プロパティについては、次のリストで説明します:

- キーベースの SSH: リモートホストへのキー対称ログインを使用する場合は、このボックスを選択します。選択した場合、キーユーザーとパスフレーズも指定する必要があります。それ以外の場合は、リモートログインアカウントのユーザーとパスワードが必要です。
- User: リモートログインアクセス権を持つユーザーの名前。
- パスワード: リモートユーザーのパスワード。
- キーユーザー: キーが有効なユーザーの名前 (オペレーティングシステムユーザーと同じでない場合)。
- キーパスフレーズ: 必要に応じて、キーのパスフレーズ。
- キーファイル: キーファイルへのパス。デフォルトは `~/.ssh/id_rsa` です。

このページで設定した SSH プロパティは、クラスタ内のすべてのホストに適用されます。これらは、ホストの定義画面でホストプロパティを編集することで、特定のホストに対してオーバーライドできます。

この画面では、次の 2 つのインストールプロパティも設定できます:

- MySQL クラスタのインストール: この設定は、Auto-Installer が NDB Cluster ソフトウェア (存在する場合) をクラスタホストにインストールするソースを決定します。使用可能な値とその効果を次に示します:
 1. **DOCKER:** から MySQL Cluster Docker イメージをインストールしてみます <https://hub.docker.com/r/mysql/mysql-cluster/> 各ホスト上
 2. **REPO:** 各ホストの [MySQL リポジトリ](#) から NDB Cluster ソフトウェアをインストールしてみます
 3. **BOTH:** Docker イメージまたはソフトウェアのいずれかを各ホストのリポジトリからインストールし、リポジトリを優先
 4. **NONE:** NDB Cluster ソフトウェアをホストにインストールしないでください。これがデフォルトです
- FW ポートを開く: インストーラがすべてのホスト上の NDB Cluster プロセスに必要なポートを開こうとするようにするには、このチェックボックスをオンにします。

次の図に、すべてのノードが `localhost` で実行されている小規模なテストクラスタの設定が表示されたクラスタの定義ページを示します:

図 23.11 テストクラスタの設定が表示された「NDB Cluster 自動インストーラクラスタの定義」画面

ORACLE MySQL Cluster Installer

Define cluster > Define hosts > Define processes > Define parameters > Deploy configuration

Settings ▾ Help ▾

Cluster Type and SSH Credentials
MySQL Cluster is able to operate in various configurations. Please specify the settings below to define the right cluster type that fits your use case. If you intend to use remote hosts for deploying MySQL Cluster, SSH must be enabled. Unless key based SSH is possible, you must submit your user name and password below.

Cluster property	Value
Cluster name [?]	mycluster-1
Host list [?]	localhost
Application area [?]	simple testing ▾
Write load [?]	low ▾

SSH property (Cluster-wide)	Value
Key based SSH [?]	<input type="checkbox"/>
User name [?]	<input type="text"/>
Password [?]	<input type="text"/>
Key user: [?]	<input type="text"/>
Key passphrase: [?]	<input type="text"/>
Key file: [?]	<input type="text"/>

Install properties (Cluster-wide)	Value
Install MySQL Cluster [?]	NONE ▾
Open FW ports [?]	<input type="checkbox"/>

◀ Previous Save&Next Finish ▶

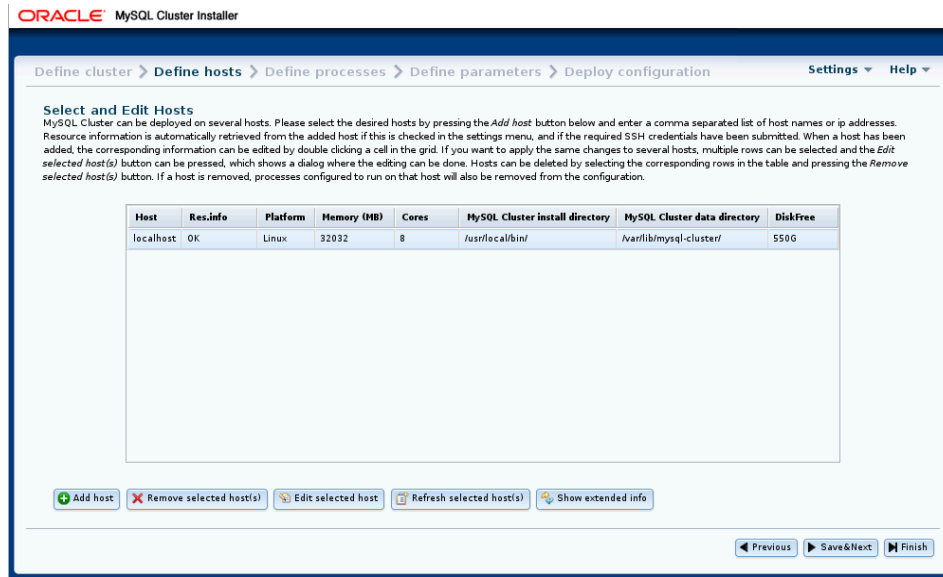
必要な設定を行ったら、構成ファイルに保存し、& を次に保存ボタンをクリックしてホストの定義画面に進むことができます。

保存せずにインストーラを終了した場合、構成ファイルは変更されません。

NDB Cluster 自動インストーラの Define Hosts 画面

ここに示す「Define Hosts」画面では、各クラスタホストのいくつかの主要なプロパティを表示および指定できます。

図 23.12 「NDB Cluster ホストの定義」画面, start



表示されるプロパティは次のとおりです:

- ホスト: このホストの名前または IP アドレス
- Res.info: インストーラがこのホストからリクエストされたリソース情報を取得できた場合に **OK** を表示
- Platform: オペレーティングシステムまたはプラットフォーム
- メモリー (MB): このホストの RAM の量
- コア: このホストで使用可能な CPU コアの数
- MySQL クラスターのインストールディレクトリ: NDB Cluster ソフトウェアがこのホストにインストールされているディレクトリへのパス。デフォルトは `/usr/local/bin/` です
- MySQL クラスターデータディレクトリ: このホスト上の NDB Cluster プロセスによってデータに使用されるディレクトリへのパス。デフォルトは `/var/lib/mysql-cluster` です。
- DiskFree: 空きディスク領域 (バイト)

複数のディスクがあるホストの場合、データディレクトリに使用されるディスク上の使用可能な領域のみが表示されます。

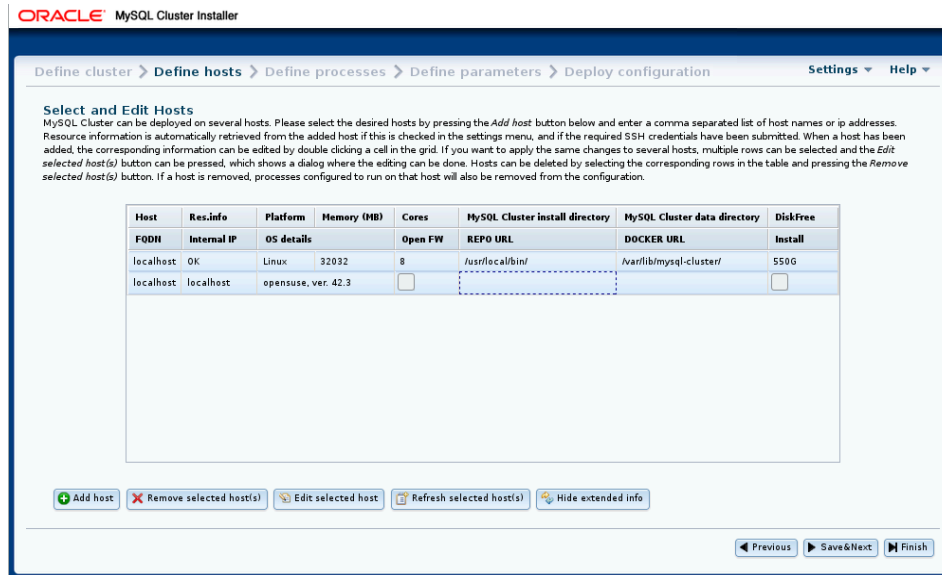
この画面には、次のプロパティを含む各ホストの拡張ビューも表示されます:

- FDQN: このホストの完全修飾ドメイン名。インストーラが接続、構成情報の配布、クラスタプロセスの起動と停止に使用します。
- 内部 IP: 別の場所で実行されているプロセスによってこのホストで実行されているクラスタプロセスとの通信に使用される IP アドレス。
- OS 詳細: オペレーティングシステム名およびバージョン情報の詳細。
- FW のオープン: このチェックボックスが有効になっている場合、インストーラは、クラスタプロセスに必要なホストファイアウォールのポートを開こうとします。
- REPO URL: MySQL NDB Cluster リポジトリの URL
- DOCKER URL: MySQL NDB Cluster Docker イメージの URL。NDB 8.0 の場合、これは `mysql/mysql-cluster:8.0` です。

- Install: このチェックボックスが有効な場合、Auto-Installer は NDB Cluster ソフトウェアをこのホストにインストールしようとします

拡張ビューを次に示します:

図 23.13 「NDB Cluster ホストの定義」画面、拡張ホスト情報ビュー



表示内のすべてのセルは編集可能ですが、「ホスト」、Res.info および FQDN カラムのセルは例外です。

リモートホストから情報を取得するには時間がかかる場合があることに注意してください。値を取得できなかったフィールドは省略記号 (...) で示されます。リストでホストを選択して「選択したホストのリフレッシュ」ボタンをクリックすると、複数のホストからのリソース情報のフェッチを再試行できます。

ホストの追加および削除

次に示すように、ホストの追加ボタンをクリックし、「新規ホストの追加」ダイアログに示されている必要なプロパティを入力することで、複数のホストを追加できます:

図 23.14 「NDB Cluster のホストの追加」ダイアログ

The image shows a dialog box titled "Add new host" with a close button (X) in the top right corner. The dialog contains the following fields and controls:

- Host name: [?]**: A text input field with a red border and a red warning icon on the right.
- Host internal IP (VPN): [?]**: A text input field.
- Key-based auth: [?]**: A checkbox that is currently unchecked.
- User [?]**: A text input field.
- Passphrase [?]**: A text input field.
- Key file [?]**: A text input field.
- Ordinary login:**
 - User [?]**: A text input field.
 - Password [?]**: A text input field.
- Open FW ports [?]**: A checkbox that is currently unchecked.
- Configure installation [?]**: A checkbox that is currently unchecked.
- Cancel**: A button.
- Add**: A button.

このダイアログには、次のフィールドがあります:

- ホスト名: 1 つ以上のホスト名、IP アドレス、またはその両方のカンマ区切りリスト。これらは、Auto-Installer が実行されているホストからアクセスできる必要があります。
- ホスト内部 IP (VPN): VPN または他の内部ネットワークで実行するようにクラスタを設定する場合は、他のホストのクラスタノードによる接続に使用する IP アドレスを入力します。
- キーベースの認証: 選択すると、キーベースの認証が有効になります。User、Passphrase および「キーファイル」の各フィールドに必要な追加情報を入力できます。
- 通常のログイン: パスワードベースのログインを使用してこのホストにアクセスする場合は、User およびパスワードフィールドに適切な情報を入力します。
- FW ポートを開く: このチェックボックスを選択すると、インストーラは、このホストファイアウォールでクラスタプロセスに必要なポートを開こうとします。
- インストールの構成: これをチェックすると、Auto-Install はこのホストで NDB Cluster ソフトウェアの設定を試みることができます。

新しいホストとそのプロパティを保存するには、「追加」をクリックします。変更を保存せずに取り消す場合は、代わりに「取消」をクリックします。

同様に、「Remove selected host(s)」というラベルのボタンを使用して 1 台以上のホストを削除できます。ホストを削除すると、そのホスト用に構成されたプロセスも削除されます。

警告

選択したホストの削除はすぐに動作します。確認ダイアログはありません。エラーのあるホストを削除する場合は、ホストの追加を使用して名前とプロパティを手動で再入力する必要があります。

「[クラスタの定義 screen](#)」の SSH ユーザー資格証明が変更されると、Auto-Installer は、情報が欠落しているホストからリソース情報をリフレッシュしようとしています。

ホストプラットフォーム名、ハードウェアリソース情報、インストールディレクトリおよびデータディレクトリを編集するには、グリッド内の対応するセルをクリックし、ホストを選択して「選択したホストの編集」というラベルのボタンをクリックします。これにより、ここに示すように、これらのフィールドを編集できるダイアログボックスが表示されます。

図 23.15 「NDB Cluster 自動インストーラホストの編集」ダイアログ

Edit selected host(s)

Please edit the fields you want to change. The changes will be applied to all selected hosts. Fields that are not edited in the form below will be left unchanged.

Platform [?]	Memory (MB) [?]	CPU cores [?]	MySQL Cluster install directory [?]	MySQL Cluster data directory [?]	DiskFree [?]
Linux	32,032	8			550G

Host external IP: [?]
localhost

Host internal IP (VPN): [?]
localhost

Key-based auth: [?]

User [?] Passphrase [?]
[] []

Key file [?]
[]

Ordinary login:
User [?] Password [?]
[] []

Open FW ports [?] Configure installation [?]

Cancel Save

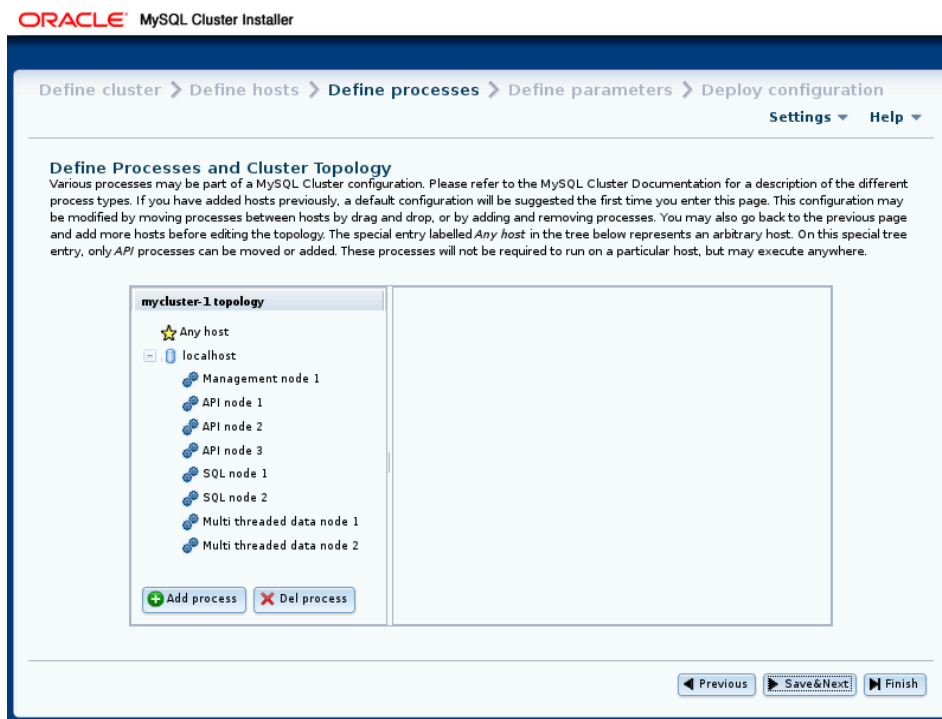
複数のホストを選択すると、編集した値が選択したすべてのホストに適用されます。

必要なすべてのホスト情報を入力したら、& を次に保存ボタンを使用して情報をクラスタ構成ファイルに保存し、「プロセスの定義 screen」に進み、そこで NDB Cluster プロセスを 1 つ以上のホストに設定できます。

NDB Cluster 自動インストーラの Define Processes 画面

ここに示すプロセスの定義画面では、NDB Cluster プロセス (ノード) をクラスタホストに割り当てる方法が提供されます:

図 23.16 「NDB Cluster 自動インストーラプロセスの定義」ダイアログ



この画面には、クラスタホストと各ホストで実行するように設定されたプロセスを示すプロセスツリー、およびツリーで現在選択されている項目に関する情報を表示するパネルが含まれています。

特定のクラスタでこの画面に最初にアクセスすると、ホストの数に基づいてデフォルトのプロセスセットが自動的に定義されます。後で「[ホストの定義 screen](#)」に戻ってすべてのホストを削除し、新しいホストを追加すると、新しいデフォルトのプロセスセットも定義されます。

NDB Cluster プロセスは、このリストで説明されているタイプです:

- 管理ノード。 個々のデータノードの停止、ノードやクラスタのステータスのクエリー、バックアップの作成などの管理タスクを実行します。実行可能ファイル: `ndb_mgmd`。
- シングルスレッドのデータノード。 データを格納し、クエリーを実行します。実行可能ファイル: `ndbd`。
- マルチスレッドのデータノード。 並列で実行される複数のワーカースレッドによってデータを格納し、クエリーを実行します。実行可能ファイル: `ndbmd`。
- SQL ノード。 NDB に対して SQL クエリーを実行するための MySQL サーバー。実行可能ファイル: `mysqld`。
- API ノード。 SQL を使用せずに、NDB API またはその他の低レベルのクライアント API を使用して NDB 内のデータにアクセスするクライアント。詳細は、[MySQL NDB Cluster API Developer Guide](#)を参照してください。

プロセス (ノード) タイプの詳細は、[セクション23.1.1「NDB Cluster のコア概念」](#)を参照してください。

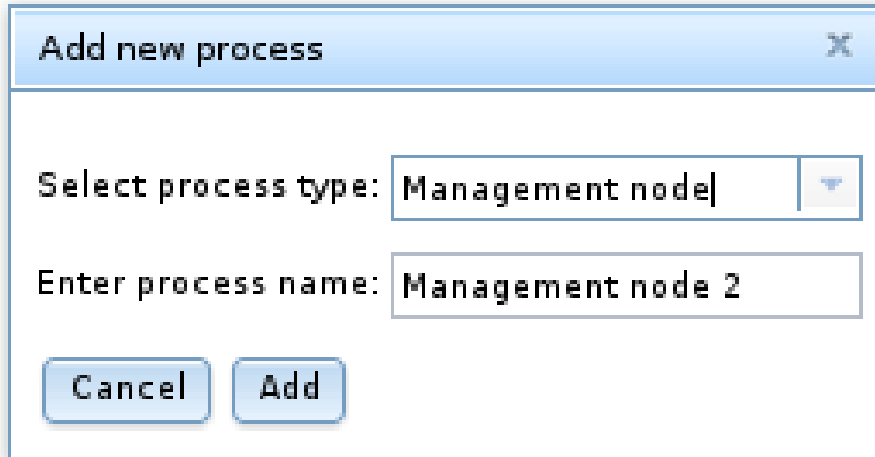
ツリーに表示されるプロセスには、簡単に識別できるように (たとえば、`SQL node 1`、`SQL node 2` のように) ホストごとにタイプ別の連番が付けられています。

個々の管理ノード、データノード、または SQL プロセスは、特定のホストに割り当てる必要があり、ほかのホストでは実行できません。API ノードは 1 台のホストに割り当てることもできますが、これは必須ではありません。代わりに、ほかのホストとは別にツリー内に表示される「Any host」エントリに別途割り当てることができます。このエントリは、任意のホストで実行できるプロセスのプレースホルダとして機能します。この「Any host」エントリを使用できるのは API プロセスだけです。

プロセスの追加。 特定のホストに新しいプロセスを追加するには、ツリーでそのホストエントリを右クリックして「プロセスの追加」ポップアップが表示されたらそれを選択するか、プロセスツリーでホストを選択して、プロセ

ツリーの下のプロセスの追加ボタンを押します。これらのアクションのどちらを実行した場合も、ここに示すようなプロセス追加ダイアログが開きます。

図 23.17 NDB Cluster 自動インストーラのプロセスの追加ダイアログ



ここでは、このセクションで前述した利用可能なプロセスタイプを選択できます。必要に応じて、提案された値の代わりに任意のプロセス名を入力することもできます。

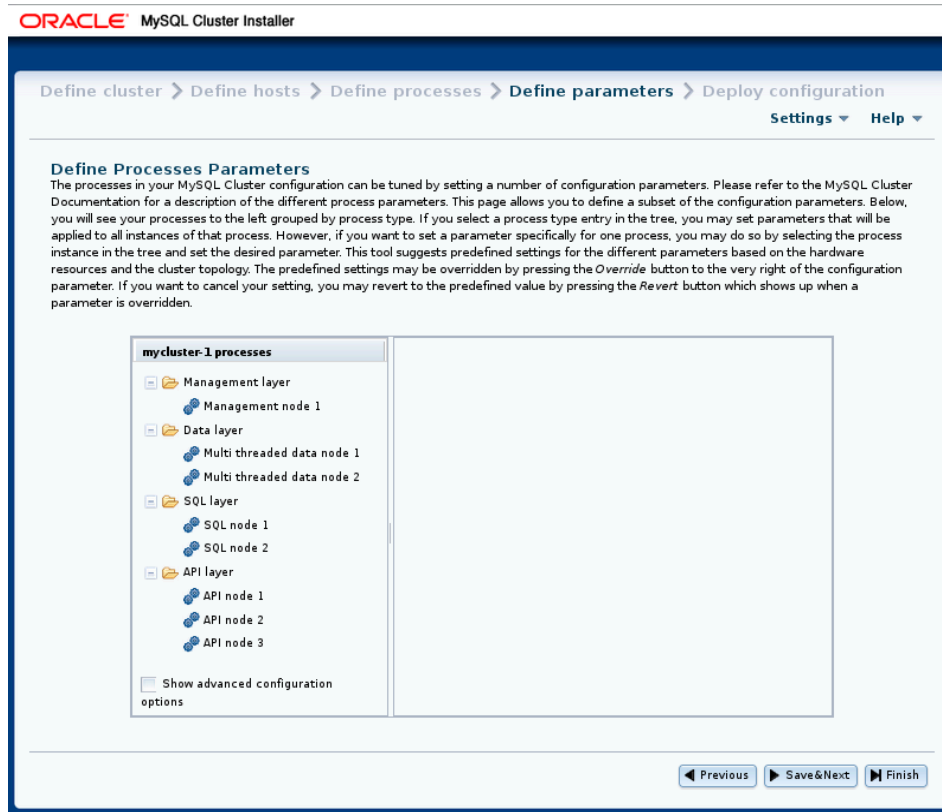
プロセスの削除 プロセスを削除するには、ツリーでそのプロセスを選択し、Del process ボタンを使用します。

プロセスツリーでプロセスを選択すると、そのプロセスに関する情報が情報パネルに表示され、ここでプロセス名とそのタイプを変更できます。マルチスレッドデータノード (`ndbmtid`) をシングルスレッドデータノード (`ndbd`) に変更することも、その逆のみを変更することもできます。ほかのプロセスタイプの変更は許可されません。他のプロセスタイプ間で変更を行う場合は、最初に元のプロセスを削除してから、目的のタイプの新しいプロセスを追加する必要があります。

NDB Cluster 自動インストーラのパラメータの定義画面

「プロセスの定義 screen」と同様に、この画面にはプロセスツリーが含まれています。パラメータの定義プロセスツリーは、「管理レイヤー」、「データレイヤー」、「SQL レイヤー」および「API レイヤー」というラベルの付いたグループにプロセスまたはノードタイプ別に編成されています。情報パネルには、現在選択されているアイテムに関する情報が表示されます。「Define Attributes」画面をここに示します。

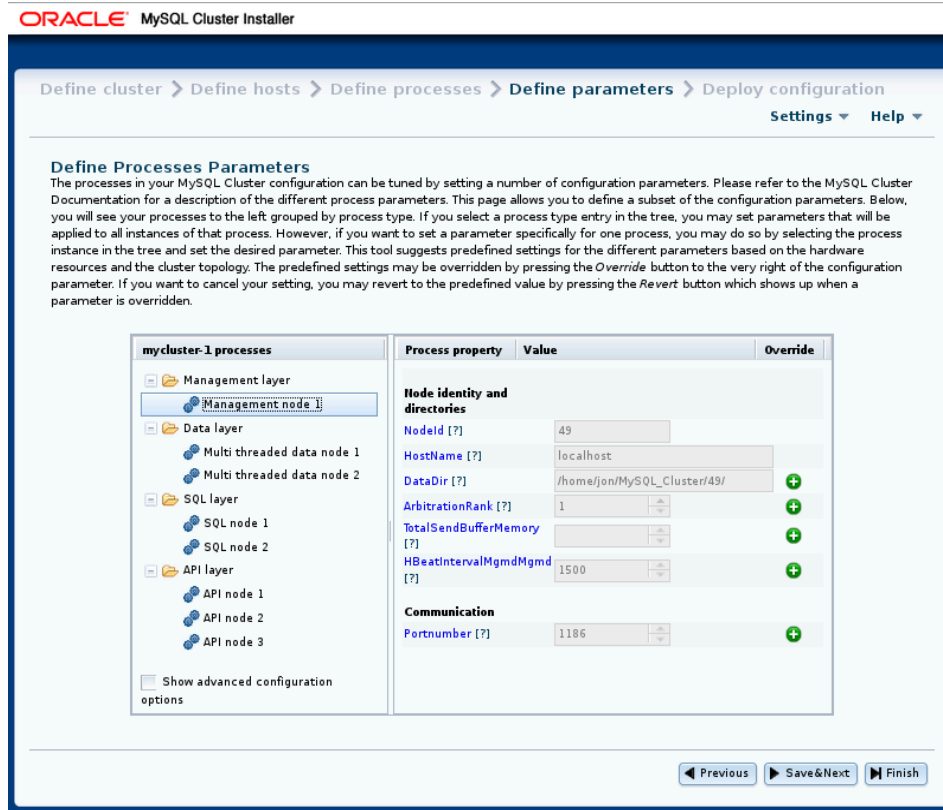
図 23.18 「NDB Cluster Auto-Installer パラメータの定義」画面



「拡張構成の表示」のラベルが付いたチェックボックスを選択すると、データノードおよび SQL ノードプロセスの拡張オプションが情報ペインに表示されます。これらのオプションは、表示されているかどうかに関係なく設定および使用されます。この動作は、「設定」の下の「拡張構成オプションの表示」をチェックしてグローバルに有効にすることもできます (NDB Cluster インストーラの [Settings](#) および [Help](#) メニュー を参照)。

1つのプロセスの属性を編集するには、ツリーからそのプロセスを選択します。クラスタに含まれる同じタイプのプロセスすべての属性を編集するには、いずれかの「Layer」フォルダを選択します。プロセス単位で設定された特定の属性の値は、該当するプロセスに以前に適用されていた属性のグループ単位の設定をオーバーライドします。このような情報パネルの例 (SQL プロセスの場合) をここに示します。

図 23.19 パラメータの定義 - プロセス属性



値をオーバーライドできる属性は、プラス記号が付いたボタンとともに情報パネルに表示されます。この + ボタンをクリックすると、属性の入カウィジェットがアクティブ化され、属性の値を変更できるようになります。値がオーバーライドされると、このボタンは X を表示するボタンに変わります。X ボタンを使用すると、特定の属性に加えられた変更が元に戻され、すぐに事前定義済の値に戻ります。

すべての構成属性には、インストーラがホスト名、ノード ID、ノードタイプなどの要因に基づいて計算した事前定義値があります。ほとんどの場合、これらの値はそのままにしておくことができます。これらの操作をまだ習熟していない場合は、属性値を変更する前に該当するドキュメントを読むことを強くお勧めします。この情報を見つけやすくするために、情報パネルに表示される各属性名は、NDB Cluster のオンラインドキュメントの説明にリンクされています。

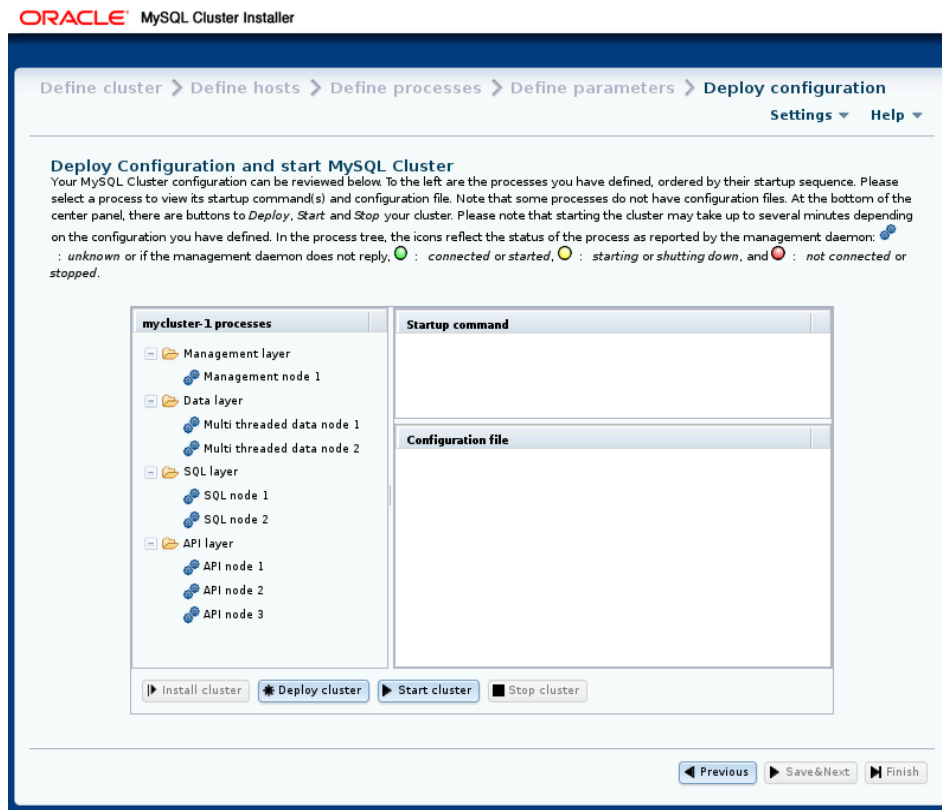
NDB Cluster 自動インストーラの Deploy Configuration 画面

この画面では、次のタスクを実行できます。

- 適用されるプロセス起動コマンドと構成ファイルを確認します
- すべてのクラスタホストで必要なファイルおよびディレクトリを作成して、構成ファイルを配布します (つまり、現在の構成に従ってクラスタを配備します)
- クラスタを起動および停止します

構成のデプロイ画面を次に示します:

図 23.20 「NDB Cluster 自動インストーラの配備構成」画面



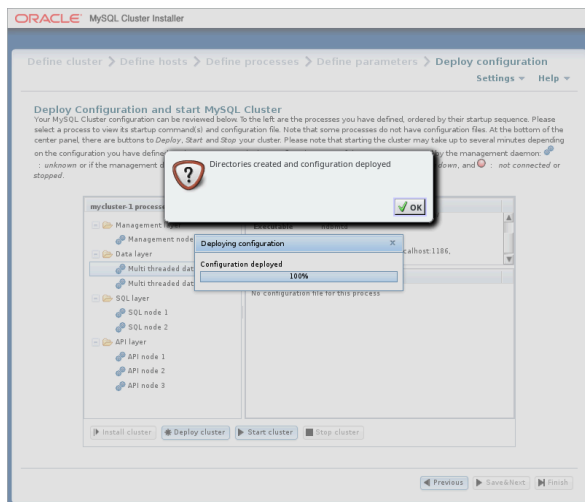
「パラメータの定義 screen」と同様に、この画面にはプロセスタイプ別に編成されたプロセスツリーが表示されます。ツリー内の各プロセスの横には、プロセスの現在のステータスを示すステータスアイコンがあります: 接続 (CONNECTED)、起動 (STARTING)、実行 (STARTED)、停止 (STOPPING) または切断 (NO_CONTACT)。このアイコンは、プロセスが接続されているか実行中の場合は緑色、起動中または停止中の場合は黄色、プロセスが停止しているか管理サーバーから接続できない場合は赤色で表示されます。

この画面には、起動コマンドまたは選択したプロセスの開始に必要なコマンドを示す 2 つの情報パネルも含まれています。(プロセスによっては、初期化が必要な場合など、複数のコマンドが必要な場合があります。) もう一方のパネルには、指定されたプロセスの構成ファイルがある場合、その内容が表示されます。

この画面には、次のリストで説明する機能としてラベル付けされ、実行される 4 つのボタンもあります:

- クラスタのインストール: このリリースでは機能しません。将来のリリースを対象とした実装です。
- 「Deploy cluster」: 構成が有効かどうか検証します。クラスタホスト上に必要なディレクトリを作成し、各ホストに構成ファイルを配布します。進捗バーには、次に示すようにデプロイメントの進行状況が表示され、次に示すように、デプロイメントが完了するとダイアログが表示されます:

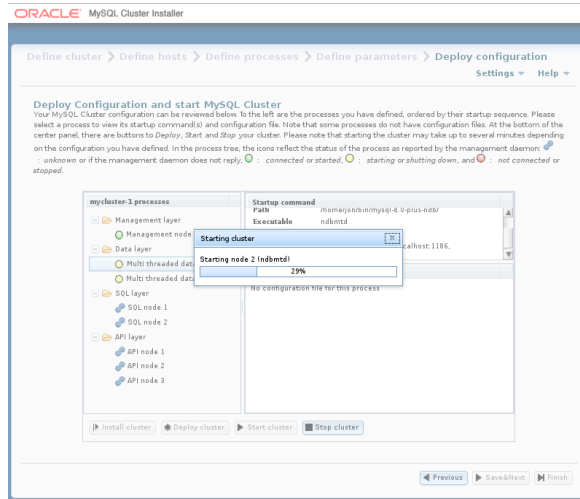
図 23.21 クラスタデプロイメントプロセス



- 「Start cluster」: 「Deploy cluster」と同様にクラスタを配備し、その後、すべてのクラスタプロセスを正しい順序で起動します。

これらのプロセスの起動には、ある程度時間がかかることがあります。完了までの推定時間が長すぎる場合は、起動プロセスを取り消しまたは続行できます。ここに示すように、進行状況バーに起動手順の現在のステータスがされます。

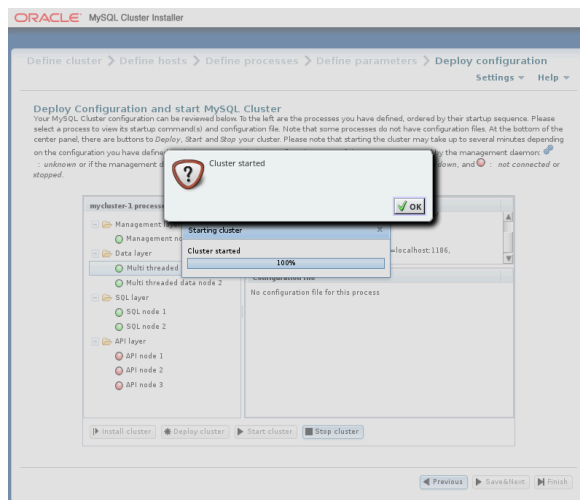
図 23.22 進行状況バーのあるクラスタ起動プロセス



プロセスツリーに表示される項目の横にあるプロセスステータスアイコンも、各プロセスのステータスで更新されます。

次に示すように、起動プロセスが完了すると、確認ダイアログが表示されます:

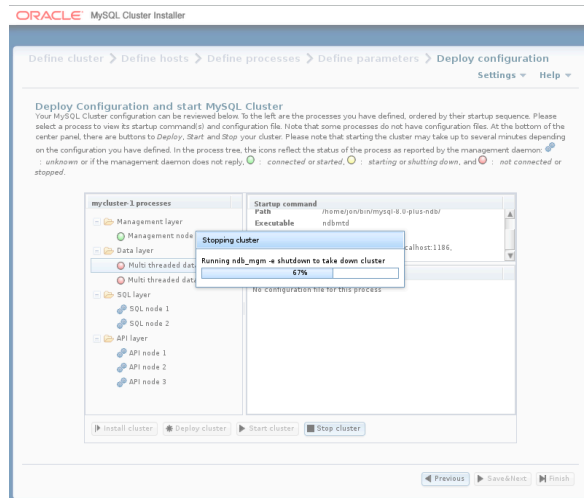
図 23.23 クラスタの起動、プロセス完了ダイアログ



- クラスタの停止: クラスタを起動したら、これを使用して停止できます。クラスタの起動と同様に、クラスタは瞬時にシャットダウンされず、ある程度時間がかかることがあります。クラスタの起動時に表示されるのと同様の進

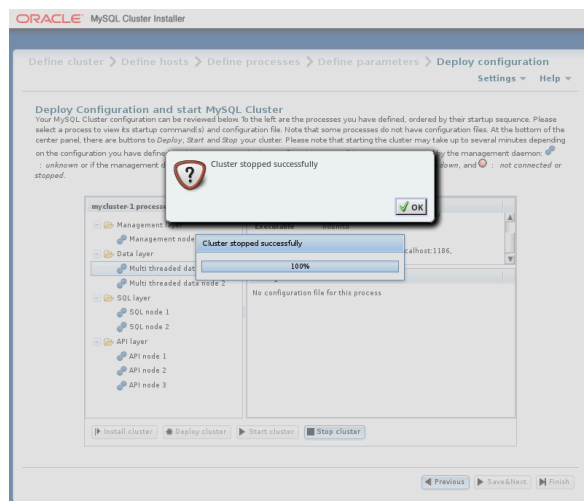
行状況バーが表示され、プロセスツリーの横にあるプロセスステータスアイコンと同じように、クラスタのシャットダウン手順の現在の大きなステータスが示されます。進捗バーは次のとおりです：

図 23.24 クラスタ停止プロセス (進行状況バーあり)



停止プロセスが完了すると、確認ダイアログが表示されます：

図 23.25 Cluster Shutdown, Process Completed ダイアログ



Auto-Installer は、管理ノードごとに NDB ノードパラメータを含む `config.ini` ファイルと、クラスタ内の `mysqld` プロセスごとに適切なオプションを含む `my.cnf` ファイルを生成します。データノードまたは API ノードの構成ファイルは作成されません。

23.3 NDB Cluster の構成

NDB Cluster の一部である MySQL サーバーは、NDB ストレージエンジンを採用しているという点で、通常の (クラスタ化されていない) MySQL サーバーとは主に異なります。このエンジンは `NDBCLUSTER` と呼ばれることもありますが、`NDB` が推奨されます。

不要なリソースの割り当てを避けるため、デフォルトでは `NDB` ストレージエンジンを無効にするようにサーバーが構成されます。`NDB` を有効にするには、サーバーの `my.cnf` 構成ファイルを変更するか、`--ndbcluster` オプションを指定してサーバーを起動する必要があります。

この MySQL サーバーはクラスタの一部であるため、管理ノードにアクセスしてクラスタ構成データを取得する方法も知っている必要があります。デフォルトの動作では、`localhost` 上の管理ノードを検索します。ただし、それがほかの場所であることを指定する必要がある場合は、`my.cnf` で、または `mysql` クライアントを使用してこれを行います。NDB ストレージエンジンを使用する前に、少なくとも 1 つの管理ノードと必要なデータノードが使用可能である必要があります。

--`ndbcluster` および NDB Cluster に固有のその他の `mysqld` オプションの詳細は、[NDB Cluster の MySQL Server オプション](#) を参照してください。

NDB Cluster のインストールに関する一般情報については、[セクション23.2「NDB Cluster のインストール」](#) を参照してください。

23.3.1 NDB Cluster のクイックテスト設定

基本を理解するために、機能する NDB Cluster のもっとも単純な構成について説明します。その後、この章の関連するほかのセクションに示した情報から、必要なセットアップを設計できるようになります。

最初に、システムの `root` ユーザーとして次のコマンドを実行して、`/var/lib/mysql-cluster` などの構成ディレクトリを作成する必要があります。

```
shell> mkdir /var/lib/mysql-cluster
```

このディレクトリで、次の情報を含む `config.ini` という名前のファイルを作成します。必要に応じて、`HostName` および `DataDir` をシステムの適切な値に置き換えます。

```
# file "config.ini" - showing minimal setup consisting of 1 data node,
# 1 management server, and 3 MySQL servers.
# The empty default sections are not required, and are shown only for
# the sake of completeness.
# Data nodes must provide a hostname but MySQL Servers are not required
# to do so.
# If you don't know the hostname for your machine, use localhost.
# The DataDir parameter also has a default value, but it is recommended to
# set it explicitly.
# Note: [db], [api], and [mgm] are aliases for [ndbd], [mysqld], and [ndb_mgmd],
# respectively. [db] is deprecated and should not be used in new installations.

[ndbd default]
NoOfReplicas= 1

[mysqld default]
[ndb_mgmd default]
[tcp default]

[ndb_mgmd]
HostName= myhost.example.com

[ndbd]
HostName= myhost.example.com
DataDir= /var/lib/mysql-cluster

[mysqld]
[mysqld]
[mysqld]
```

これで、`ndb_mgmd` 管理サーバーを起動できるようになりました。デフォルトでは現在の作業ディレクトリ内にある `config.ini` ファイルの読み取りが試行されるため、このファイルが配置されているディレクトリに場所を移動してから `ndb_mgmd` を起動します。

```
shell> cd /var/lib/mysql-cluster
shell> ndb_mgmd
```

次に、`ndbd` を実行して 1 つのデータノードを起動します。

```
shell> ndbd
```

`ndbd` の起動時に使用できるコマンド行オプションについては、[セクション23.4.32「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#) を参照してください。

デフォルトでは、`ndbd` は `localhost` のポート 1186 で管理サーバーを検索します。

注記

バイナリ tarball から MySQL をインストールした場合は、`ndb_mgmd` および `ndbd` サーバーのパスを明示的に指定する必要があります。(通常、これらは `/usr/local/mysql/bin` にあります。)

最後に、MySQL データディレクトリ (通常は `/var/lib/mysql` または `/usr/local/mysql/data`) に場所を変更して、NDB ストレージエンジンを有効にするのに必要なオプションが `my.cnf` ファイルに含まれていることを確認します。

```
[mysql@d]
ndbcluster
```

これで、MySQL サーバーを通常どおり起動できるようになりました。

```
shell> mysql_safe --user=mysql &
```

しばらく待ってから、MySQL サーバーが適切に実行されていることを確認します。「`mysql ended`」という通知が表示された場合は、サーバーの `.err` ファイルをチェックして、どのような不具合があったかを調べます。

ここまで問題なく進んだ場合は、クラスタを使用し始めることができます。サーバーに接続して、`NDBCLUSTER` ストレージエンジンが有効になっていることを確認します。

```
shell> mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 8.0.29

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SHOW ENGINES\G
...
***** 12. row *****
Engine: NDBCLUSTER
Support: YES
Comment: Clustered, fault-tolerant, memory-based tables
***** 13. row *****
Engine: NDB
Support: YES
Comment: Alias for NDBCLUSTER
...
```

前の出力例に表示されている行番号は、サーバーの構成方法によっては、使用しているシステムで表示されるものと異なる可能性があります。

`NDBCLUSTER` テーブルを作成してみます。

```
shell> mysql
mysql> USE test;
Database changed

mysql> CREATE TABLE ctest (i INT) ENGINE=NDBCLUSTER;
Query OK, 0 rows affected (0.09 sec)

mysql> SHOW CREATE TABLE ctest \G
***** 1. row *****
      Table: ctest
      Create Table: CREATE TABLE `ctest` (
        `i` int(11) default NULL
      ) ENGINE=ndbcluster DEFAULT CHARSET=latin1
      1 row in set (0.00 sec)
```

ノードが適切にセットアップされたことをチェックするには、管理クライアントを起動します。

```
shell> ndb_mgm
```

クラスタのステータスに関するレポートを取得するには、管理クライアント内で `SHOW` コマンドを使用します。

```
ndb_mgm> SHOW
Cluster Configuration
-----
[ndbd(NDB)] 1 node(s)
id=2 @127.0.0.1 (Version: 8.0.23-ndb-8.0.23, Nodegroup: 0, *)
```

```
[ndb_mgmd(MGM)] 1 node(s)
id=1 @127.0.0.1 (Version: 8.0.23-ndb-8.0.23)

[mysqld(API)] 3 node(s)
id=3 @127.0.0.1 (Version: 8.0.23-ndb-8.0.23)
id=4 (not connected, accepting connect from any host)
id=5 (not connected, accepting connect from any host)
```

これで、動作中の NDB Cluster が正常に設定されました。これで、`ENGINE=NDBCLUSTER` またはそのエイリアスである `ENGINE=NDB` を指定して作成したテーブルを使用して、クラスタにデータを格納できます。

23.3.2 NDB Cluster 構成パラメータ、オプション、および変数の概要

次のいくつかのセクションでは、NDB Cluster プロセスとして実行するとき `mysqld` によって `my.cnf` ファイルまたはコマンド行から読み取られるオプションおよび変数のほかに、ノードの動作のさまざまな側面を制御するために `config.ini` ファイルで使用される NDB Cluster ノード構成パラメータのサマリーテーブルを提供します。各ノードパラメータテーブルには、特定のタイプ (`ndbd`, `ndb_mgmd`, `mysqld`, `computer`, `tcp` または `shm`) のパラメータがリストされます。すべてのテーブルには、パラメータ、オプションまたは変数のデータ型に加えて、必要に応じてデフォルト値、最小値および最大値が含まれます。

ノードの再起動時の考慮事項。 ノードパラメータの場合、これらのテーブルには、必要な再起動のタイプ (ノードの再起動またはシステムの再起動) と、特定の構成パラメータの値を変更するために `--initial` で再起動する必要があるかどうかも示されます。ノードの再起動またはノードの初期再起動を実行する場合は、すべてのクラスタデータノードを順番に再起動する必要があります (ローリング再起動とも呼ばれます)。 `node` としてマークされたクラスタ構成パラメータは、オンラインで (つまり、この方法でクラスタをシャットダウンすることなく) 更新できます。ノードの初期再起動では、`--initial` オプションを指定して各 `ndbd` プロセスを再起動する必要があります。

システムの再起動では、クラスタ全体を完全にシャットダウンして再起動する必要があります。システムの初期再起動では、クラスタのバックアップを取り、シャットダウン後にクラスタファイルシステムを消去して、再起動後にバックアップからリストアする必要があります。

どのクラスタ再起動でも、クラスタのすべての管理サーバーを再起動して、更新された構成パラメータ値を読み取ることができるようにする必要があります。

重要

数値型のクラスタパラメータの値は通常問題なく増やすことができますが、そのような調整は比較的小さいインクリメントで徐々に行うことをお勧めします。多くのパラメータは、ローリング再起動を使用してオンラインで増やすことができます。

ただし、このようなパラメータ値の削減は、ノードの再起動、ノードの初期再起動、またはクラスタの完全なシステムの再起動のどれを使用するかに関係なく、安易に行うべきではありません。その場合は、事前に入念な計画とテストを行うことをお勧めします。これは、メモリー使用やディスクスペースに関連するパラメータ (`MaxNoOfTables`, `MaxNoOfOrderedIndexes`, `MaxNoOfUniqueHashIndexes` など) に関して特に当てはまります。さらに、一般的には、メモリーおよびディスク使用に関連する構成パラメータは単純なノードの再起動により値を増やすことができますが、値を減らすにはノードの初期再起動が必要になります。

これらのパラメータの一部は、複数のタイプのクラスタノードの構成に使用できるため、複数の表に表示されている場合があります。

注記

これらの表では、多くの場合、最大値として `4294967039` が表示されています。この値は、`NDBCLUSTER` のソースで `MAX_INT_RNIL` として定義されており、`0xFFFFFFFF`、または $2^{32} - 2^8 - 1$ と等価です。

23.3.2.1 NDB Cluster データノード構成パラメータ

このセクションのリストは、NDB Cluster データノードを構成するために `config.ini` ファイルの `[ndbd]` または `[ndbd default]` セクションで使用されるパラメータに関する情報を提供します。これらの各パラメータに関する詳しい説明およびその他の追加情報については、[セクション 23.3.3.6 「NDB Cluster データノードの定義」](#) を参照してください。

これらのパラメータは、マルチスレッドバージョンの `ndbd` である `ndbmtbd` にも適用されます。詳細は、[セクション 23.4.3 「ndbmtbd — NDB Cluster データノードデーモン \(マルチスレッド\)」](#) を参照してください。

- **Arbitration**: ノード障害時のスプリットブレインの問題を回避するためのアービトレーションの実行方法。
- **ArbitrationTimeout**: データベースパーティションがアービトレーション信号を待機する最大時間 (ミリ秒)。
- **BackupDataBufferSize**: バックアップのデータバッファのデフォルトサイズ (バイト)。
- **BackupDataDir**: バックアップを格納する場所へのパス。* 有効な * デフォルトが `FileSystemPath/BACKUP` になるように、文字列 `/BACKUP` が常にこの設定に追加されることに注意してください。
- **BackupDiskWriteSpeedPct**: バックアップの開始時に LCP 用に予約するデータノードの割り当て済み最大書き込み速度 (`MaxDiskWriteSpeed`) の割合を設定。
- **BackupLogBufferSize**: バックアップのログバッファのデフォルトサイズ (バイト)。
- **BackupMaxWriteSize**: バックアップによるファイルシステム書き込みの最大サイズ (バイト)。
- **BackupMemory**: ノード当たりのバックアップに割り当てられた合計メモリー (バイト)。
- **BackupReportFrequency**: バックアップ中のバックアップステータスレポートの頻度 (秒)。
- **BackupWriteSize**: バックアップによるファイルシステム書き込みのデフォルトサイズ (バイト)。
- **BatchSizePerLocalScan**: 保留ロックがあるスキンのロックレコード数の計算に使用されます。
- **BuildIndexThreads**: システムまたはノードの再起動時に順序付けされた索引の構築に使用するスレッドの数。`ndb_restore --rebuild-indexes` の実行時にも適用されます。このパラメータを 0 に設定すると、順序付き索引のマルチスレッド構築が無効になります。
- **CompressedBackup**: `zlib` を使用した書き込み時のバックアップの圧縮。
- **CompressedLCP**: `zlib` を使用した圧縮 LCP の書き込み。
- **ConnectCheckIntervalDelay**: データノード接続性チェックステージ間の時間。データノードは、1 間隔後は疑わしいと見なされ、2 間隔後は応答なしで停止。
- **CrashOnCorruptedTuple**: 有効にすると、破損したタプルが検出されるたびにノードが強制的に停止されます。
- **DataDir**: このノードのデータディレクトリ。
- **DataMemory**: データを格納するために割り当てられた各データノード上のバイト数。使用可能なシステム RAM および `IndexMemory` のサイズに従います。
- **DefaultHashMapSize**: テーブルハッシュマップに使用するサイズ (バケット単位) を設定します。3 つの値がサポートされています: 0、240 および 3840。
- **DictTrace**: NDB 開発のための `DBDICT` デバッグの有効化。
- **DiskDataUsingSameDisk**: 「ディスクデータ」テーブルスペースが別の物理ディスクにある場合は `false` に設定。
- **DiskIOThreadPool**: ファイルアクセス用のバインドされていないスレッドの数。ディスクデータにのみ適用されません。
- **Diskless**: ディスクを使用せずに実行。
- **DiskPageBufferEntries**: `DiskPageBufferMemory` で割り当てるメモリー。非常に大きなディスクトランザクションでは、この値を増やす必要がある場合があります。
- **DiskPageBufferMemory**: ディスクページバッファキャッシュに割り当てられた各データノードのバイト数。
- **DiskSyncSize**: 同期が強制される前にファイルに書き込まれるデータ量。
- **EnablePartialLcp**: 部分 LCP を有効にします (`true`)。これが無効 (`false`) の場合、すべての LCP が完全なチェックポイントを書き込みます。
- **EnableRedoControl**: `REDO` ログの使用を制御するための適応チェックポイント処理速度の有効化。

- **EventLogBufferSize**: データノード内の NDB ログイベントの循環バッファのサイズ。
- **ExecuteOnComputer**: 以前に定義された COMPUTER を参照する文字列。
- **ExtraSendBufferMemory**: TotalSendBufferMemory または SendBufferMemory によって割り当てられたものに加えて、送信バッファに使用するメモリー。デフォルト (0) では最大 16MB まで許可されます。
- **FileSystemPath**: データノードがデータを格納するディレクトリへのパス (ディレクトリが存在する必要があります)。
- **FileSystemPathDataFiles**: データノードがディスクデータファイルを格納するディレクトリへのパス。デフォルト値は、設定されている場合は FileSystemPathDD です。設定されている場合は、FileSystemPath が使用されます。それ以外の場合は、DataDir の値が使用されます。
- **FileSystemPathDD**: データノードがディスクデータと Undo ファイルを格納するディレクトリへのパス。デフォルト値は、設定されている場合は FileSystemPath です。それ以外の場合は、DataDir の値が使用されます。
- **FileSystemPathUndoFiles**: データノードがディスクデータ用の Undo ファイルを格納するディレクトリへのパス。デフォルト値は、設定されている場合は FileSystemPathDD です。設定されている場合は、FileSystemPath が使用されます。それ以外の場合は、DataDir の値が使用されます。
- **FragmentLogFileSize**: 各 REDO ログファイルのサイズ。
- **HeartbeatIntervalDbApi**: API ノードデータノードのハートビート間の時間。(3 つのハートビートの欠落後に API 接続がクローズされました)。
- **HeartbeatIntervalDbDb**: データノードからデータノードへのハートビート間の時間。データノードは、ハートビートが 3 回失われたあとで停止したと見なされます。
- **HeartbeatOrder**: 指定されたノードがまだアクティブでクラスタに接続されているかどうかを判断するために、データノードが相互にハートビートをチェックする順序を設定します。すべてのデータノードでゼロであるか、すべてのデータノードで個別のゼロ以外の値である必要があります。詳細は、ドキュメントを参照してください。
- **HostName**: このデータノードのホスト名または IP アドレス。
- **IndexMemory**: インデックスを格納するために割り当てられた各データノード上のバイト数。使用可能なシステム RAM および DataMemory のサイズに従います。
- **IndexStatAutoCreate**: 索引作成時の自動統計収集の有効化/無効化。
- **IndexStatAutoUpdate**: 索引の変更の監視および自動統計更新のトリガー。
- **IndexStatSaveScale**: 格納された索引統計のサイズを決定するために使用されるスケール変更係数。
- **IndexStatSaveSize**: 索引ごとの保存済統計の最大サイズ (バイト)。
- **IndexStatTriggerPct**: 索引統計の更新に対する DML 操作のしきい値の変更率。値は IndexStatTriggerScale によってスケールダウンされます。
- **IndexStatTriggerScale**: 大きい索引の場合、IndexStatTriggerPct をこの量でスケールダウンし、2 を底とする索引サイズの対数で乗算します。スケーリングを無効にするには 0 に設定。
- **IndexStatUpdateDelay**: 指定された索引の自動索引統計更新間の最小遅延。0 は遅延なしを意味。
- **InitFragmentLogFiles**: フラグメントログファイルの初期化 (スパース/フル)。
- **InitialLogFileGroup**: 最初の起動時に作成されるログファイルグループについて説明します。形式についてはドキュメントを参照してください。
- **InitialNoOfOpenFiles**: データノードごとの開いているファイルの初期数。(ファイルごとに 1 つのスレッドが作成されます)。
- **InitialTablespace**: 最初の起動時に作成されるテーブルスペースについて説明します。形式についてはドキュメントを参照してください。
- **InsertRecoveryWork**: 挿入された行に使用される RecoveryWork の割合。部分的なローカルチェックポイントが使用されていないかぎり効果はありません。

- **LateAlloc**: 管理サーバーへの接続の確立後にメモリーを割り当てます。
- **LcpScanProgressTimeout**: システム全体の LCP の進行を保証するために、ノードが停止されるまでにローカルチェックポイントフラグメントスキャンを停止できる最大時間。無効にするには 0 を使用。
- **LockExecuteThreadToCPU**: CPU ID のカンマ区切りリスト。
- **LockMaintThreadsToCPU**: メンテナンススレッドを実行する CPU を示す CPU ID。
- **LockPagesInMainMemory**: 0 = ロックを無効化、1 = メモリー割当て後にロック、2 = メモリー割当て前にロック。
- **LogLevelCheckpoint**: stdout に出力されるローカルおよびグローバルチェックポイント情報のログレベル。
- **LogLevelCongestion**: 標準出力に出力される輻輳情報のレベル。
- **LogLevelConnection**: stdout に出力されるノード接続/切断情報のレベル。
- **LogLevelError**: トランスポータ、標準出力に出力されるハートビートエラー。
- **LogLevelInfo**: ハートビートおよびログ情報が stdout に出力される。
- **LogLevelNodeRestart**: ノード再起動のレベルおよびノード障害情報が stdout に出力されます。
- **LogLevelShutdown**: stdout に出力されるノード停止情報のレベル。
- **LogLevelStartup**: stdout に出力されるノード起動情報のレベル。
- **LogLevelStatistic**: stdout に出力されるトランザクション、操作およびトランスポータ情報のレベル。
- **LongMessageBuffer**: 内部ロングメッセージ用に各データノードに割り当てられたバイト数。
- **MaxAllocate**: テーブルのメモリー割当て時に使用する割当ての最大サイズ。
- **MaxBufferedEpochs**: サブスクライブしているノードが遅れることのできるエポックの許容数 (未処理エポック)。超過すると、遅れているサブスクライバが切断されます。
- **MaxBufferedEpochBytes**: エポックをバッファリングするために割り当てられた合計バイト数。
- **MaxDiskDataLatency**: トランザクションの中断を開始するまでのディスクアクセスの最大許容平均レイテンシ (ミリ秒)。
- **MaxDiskWriteSpeed**: 再起動が進行中でない場合に LCP およびバックアップで書き込むことができる最大バイト数/秒。
- **MaxDiskWriteSpeedOtherNodeRestart**: 別のノードの再起動時に LCP およびバックアップで書き込むことができる最大バイト数/秒。
- **MaxDiskWriteSpeedOwnRestart**: このノードの再起動時に LCP およびバックアップで書き込むことができる最大バイト数/秒。
- **MaxFKBuildBatchSize**: 外部キーの作成に使用するスキャンバッチの最大サイズ。この値を大きくすると、外部キーの構築が高速化されますが、進行中のトラフィックにも影響。
- **MaxDMLOperationsPerTransaction**: トランザクションのサイズを制限します。この数を超える DML 操作が必要な場合は、トランザクションを中断します。無効にするには 0 に設定。
- **MaxLCPStartDelay**: LCP が、テーブルデータの並列リカバリのために自身をロックキューに入れるまでに、(他のデータノードがメタデータ同期を完了できるようにするために) チェックポイント mutex をポーリングする時間 (秒)。
- **MaxNoOfAttributes**: データベースに格納されている属性の合計数を提案します (すべてのテーブルの合計)。
- **MaxNoOfConcurrentIndexOperations**: 1 つのデータノードで同時に実行できるインデックス操作の合計数。
- **MaxNoOfConcurrentOperations**: トランザクションコーディネータ内の操作レコードの最大数。
- **MaxNoOfConcurrentScans**: データノードで同時に実行されるスキャンの最大数。

- **MaxNoOfConcurrentSubOperations**: 同時サブスクリバ操作の最大数.
- **MaxNoOfConcurrentTransactions**: このデータノードで同時に実行されるトランザクションの最大数。同時に実行できるトランザクションの合計数は、この値にクラスタ内のデータノードの数を掛けた数です.
- **MaxNoOfFiredTriggers**: 1 つのデータノード上で同時に起動できるトリガーの総数.
- **MaxNoOfLocalOperations**: このデータノードに定義されている操作レコードの最大数.
- **MaxNoOfLocalScans**: このデータノードで並列に実行されるフラグメントスキャンの最大数.
- **MaxNoOfOpenFiles**: データ node.(One thread is created per file) 当たりの最大オープンファイル数.
- **MaxNoOfOrderedIndexes**: システムで定義できる順序付き索引の合計数.
- **MaxNoOfSavedMessages**: エラーログに書き込むエラーメッセージの最大数および保持するトレースファイルの最大数.
- **MaxNoOfSubscribers**: サブスクリバの最大数.
- **MaxNoOfSubscriptions**: サブスクリプションの最大数 (デフォルトは 0 = MaxNoOfTables).
- **MaxNoOfTables**: データベースに格納されている NDB テーブルの合計数を提案.
- **MaxNoOfTriggers**: システムで定義できるトリガーの合計数.
- **MaxNoOfUniqueHashIndexes**: システムで定義できる一意のハッシュ索引の合計数.
- **MaxParallelCopyInstances**: ノードの再起動時のパラレルコピーの数。デフォルトは 0 で、両方のノードの LDM の数を最大 16 まで使用.
- **MaxParallelScansPerFragment**: フラグメント当たりのパラレルスキャンの最大数。この制限に達すると、スキャンはシリアル化されます.
- **MaxReorgBuildBatchSize**: テーブルパーティションの再編成に使用するスキャンバッチの最大サイズ。この値を大きくすると、テーブルパーティションの再編成が高速化されますが、進行中のトラフィックにも影響.
- **MaxStartFailRetries**: データノードが起動時に失敗した場合の最大再試行回数。StopOnError = 0 が必要です。0 に設定すると、開始試行は無期限に続行されます.
- **MaxUIBuildBatchSize**: 一意キーの作成に使用するスキャンバッチの最大サイズ。この値を大きくすると、一意キーの構築が高速化されますが、進行中のトラフィックにも影響.
- **MemReportFrequency**: メモリーレポートの頻度 (秒)。0 = パーセンテージ制限を超えた場合にのみレポート.
- **MinDiskWriteSpeed**: LCP およびバックアップで書き込むことができる最小バイト数/秒.
- **MinFreePct**: 再起動のために予約しておくメモリーリソースの割合.
- **NodeGroup**: データノードが属するノードグループ。クラスタの初期起動時にのみ使用されます.
- **NodeGroupTransporters**: 同じノードグループ内のノード間で使用するトランスポータの数.
- **NodeId**: クラスタ内のすべてのノード間でデータノードを一意に識別する番号.
- **NoOfFragmentLogFiles**: データノードに属する 4 つの各ファイルセット内の 16 M バイト REDO ログファイルの数.
- **NoOfReplicas**: データベース内のすべてのデータのコピー数.
- **Numa**: (Linux のみ。libnuma が必要) NUMA サポートを制御します。0 に設定すると、システムはデータノードプロセスによるインターリーブの使用を決定できます。1 はデータノードによるインターリーブの使用を意味.
- **ODirect**: 可能な場合は O_DIRECT ファイルの読取りおよび書込みを使用.
- **ODirectSyncFlag**: O_DIRECT 書込みは同期書込みとして処理されます。ODirect が有効になっていない場合、InitFragmentLogFiles が SPARSE に設定されている場合、またはその両方の場合は無視されます.
- **RealtimeScheduler**: true の場合、データノードスレッドはリアルタイムスレッドとしてスケジューラされます。デフォルトは false です.

- **RecoveryWork**: LCP ファイルの記憶域オーバーヘッドの割合: 値が大きいほど、通常の操作での作業が少なくなり、リカバリ中の作業が増えます。
- **RedoBuffer**: REDO ログの書き込み用に割り当てられた各データノードのバイト数。
- **RedoOverCommitCounter**: RedoOverCommitLimit がこの回数を超えると、トランザクションは中断され、操作は DefaultOperationRedoProblemAction で指定されたとおりに処理されます。
- **RedoOverCommitLimit**: 現行の REDO バッファのフラッシュにかかる時間がこの秒数を超えるたびに、これが発生した回数が RedoOverCommitCounter と比較されます。
- **ReservedConcurrentIndexOperations**: 1 つのデータノード上に専用のリソースを持つ同時インデックス操作の数。
- **ReservedConcurrentOperations**: 1 つのデータノード上のトランザクションコーディネータに専用のリソースを持つ同時操作の数。
- **ReservedConcurrentScans**: 1 つのデータノード上で専用のリソースを持つ同時スキャンの数。
- **ReservedConcurrentTransactions**: 1 つのデータノード上に専用のリソースを持つ同時トランザクションの数。
- **ReservedFiredTriggers**: 1 つのデータノード上に専用のリソースを持つトリガーの数。
- **ReservedLocalScans**: 1 つのデータノード上に専用のリソースを持つ同時フラグメントスキャンの数。
- **ReservedTransactionBufferMemory**: 各データノードに割り当てられたキーおよび属性データの動的バッファ領域 (バイト)。
- **RestartOnErrorInsert**: 挿入エラーが原因の再起動の制御タイプ (StopOnError が有効な場合)。
- **SchedulerExecutionTimer**: 送信前にスケジューラで実行されるマイクロ秒数。
- **SchedulerResponsiveness**: NDB スケジューラの応答の最適化を 0-10 に設定します。値が大きいほど応答時間は短縮されますが、スループットは低下。
- **SchedulerSpinTimer**: スリープ前にスケジューラで実行されるマイクロ秒数。
- **ServerPort**: API ノードからの着信接続用にトランスポータを設定するために使用されるポート。
- **SharedGlobalMemory**: 使用するために割り当てられた各データノード上の合計バイト数。
- **SpinMethod**: データノードで使用されるスピン方法を決定します。詳細は、ドキュメントを参照してください。
- **StartFailRetryDelay**: 再試行前の起動失敗後の遅延 (秒)。StopOnError = 0 が必要です。
- **StartFailureTimeout**: 終了するまでの待機時間 (ミリ秒)。 (0 = 永久に待機)。
- **StartNoNodeGroupTimeout**: 起動を試行する前にノードグループのないノードを待機する時間 (0 = 永久)。
- **StartPartialTimeout**: すべてのノードなしで起動を試行するまでの待機時間 (ミリ秒)。 (0 = 永久に待機)。
- **StartPartitionedTimeout**: パーティション化を開始する前に待機するミリ秒数。 (0 = 永久に待機)。
- **StartupStatusReportFrequency**: 起動時のステータスレポートの頻度。
- **StopOnError**: 0 に設定すると、データノードは自動的に再起動し、次のノード障害を回復。
- **StringMemory**: 文字列メモリのデフォルトサイズ (0 から 100 = 最大の %、101+= 実際のバイト)。
- **TcpBind_INADDR_ANY**: 任意の場所から接続を確立できるように IP_ADDR_ANY をバインドします (自動生成された接続の場合)。
- **TimeBetweenEpochs**: エポック間の時間 (レプリケーションに使用される同期)。
- **TimeBetweenEpochsTimeout**: エポック間の時間のタイムアウト。 ノード停止の原因を超えています。
- **TimeBetweenGlobalCheckpoints**: ディスクへのトランザクションのグループコミット間隔。
- **TimeBetweenGlobalCheckpointsTimeout**: ディスクへのトランザクションのグループコミットの最小タイムアウト。
- **TimeBetweenInactiveTransactionAbortCheck**: 非アクティブなトランザクションのチェック間隔。

- **TimeBetweenLocalCheckpoints**: データベースのスナップショットを取得する間隔 (base-2 のバイトの対数で表現).
- **TimeBetweenWatchDogCheck**: データノード内での実行チェック間の時間.
- **TimeBetweenWatchDogCheckInitial**: データノード内部での実行チェック間の時間 (メモリー割当て時の初期開始フェーズ).
- **TotalSendBufferMemory**: すべてのトランスポータ送信バッファに使用する合計メモリー. .
- **TransactionBufferMemory**: 各データノードに割り当てられたキーおよび属性データの動的バッファ領域 (バイト).
- **TransactionDeadlockDetectionTimeout**: トランザクションがデータノード内での実行に費やすことができる時間。これは、トランザクションコーディネータがトランザクションに参加している各データノードがリクエストを実行するのを待機する時間です。データノードにこの時間より長い時間がかかった場合、トランザクションは中止されます。
- **TransactionInactiveTimeout**: トランザクションの別の部分を実行する前にアプリケーションが待機する時間 (ミリ秒)。これは、アプリケーションがトランザクションの別の部分 (クエリー、ステートメント) を実行または送信するのをトランザクションコーディネータが待機する時間です。アプリケーションに時間がかかりすぎると、トランザクションは中断されます。Timeout = 0 は、アプリケーションがタイムアウトしないことを意味。
- **TransactionMemory**: 各データノード上のトランザクションに割り当てられたメモリー.
- **TwoPassInitialNodeRestartCopy**: ノードの初期再起動時に 2 つのパスにデータをコピーします。これにより、このような再起動のための順序付き索引のマルチスレッド構築が可能になります。
- **UndoDataBuffer**: データ UNDO ログの書き込み用に割り当てられた各データノードのバイト数.
- **UndoIndexBuffer**: インデックス Undo ログの書き込み用に割り当てられた各データノード上のバイト数.
- **UseShm**: このデータノードと、このホストで実行されている API ノードの間の共有メモリー接続を使用.

次のパラメータは、`ndbmysqld` に固有です:

- **AutomaticThreadConfig**: 自動スレッド構成を使用し、ThreadConfig および MaxNoOfExecutionThreads の設定をオーバーライド.
- **ClassicFragmentation**: true の場合、従来のテーブル断片化を使用します。false に設定すると、LDM 間でのテーブルフラグメントの柔軟な分散が有効になります。
- **MaxNoOfExecutionThreads**: ndbmysqld の場合のみ、実行スレッドの最大数を指定.
- **NoOfFragmentLogParts**: このデータノードに属する REDO ログファイルグループの数.
- **NumCPUs**: AutomaticThreadConfig で使用する CPU の数の指定.
- **PartitionsPerNode**: 各データノードで作成されるテーブルパーティションの数を決定します。ClassicFragmentation が有効な場合は使用されません.
- **ThreadConfig**: マルチスレッドデータノード (ndbmysqld) の構成に使用されます。デフォルトは空の文字列です。構文およびその他の情報は、ドキュメントを参照してください。

23.3.2.2 NDB Cluster 管理ノードの構成パラメータ

このセクションのリストでは、NDB Cluster 管理ノードを構成するために `config.ini` ファイルの `[ndb_mgmd]` または `[mgm]` セクションで使用されるパラメータについて説明します。各パラメータ関する詳しい説明およびその他の追加情報については、[セクション 23.3.3.5 「NDB Cluster 管理サーバーの定義」](#) を参照してください。

- **ArbitrationDelay**: アービトレートするよう求められた場合、アービトレータはこの時間待機してから投票します (ミリ秒).
- **ArbitrationRank**: 0 の場合、管理ノードはアービトレータではありません。カーネルがアービトレータを選択する順序 1, 2.
- **DataDir**: このノードのデータディレクトリ.
- **ExecuteOnComputer**: 以前に定義された COMPUTER を参照する文字列.

- **ExtraSendBufferMemory**: TotalSendBufferMemory または SendBufferMemory によって割り当てられたものに加えて、送信バッファに使用するメモリー。デフォルト (0) では最大 16MB まで許可されます。
- **HeartbeatIntervalMgmdMgmd**: 管理ノードから管理ノードへのハートビート間の時間。管理ノード間の接続は、ハートビートが 3 回失われた後に失われたとみなされます。
- **HeartbeatThreadPriority**: 管理ノードのハートビートスレッドポリシーおよび優先度を設定します。使用可能な値については、手動を参照してください。
- **HostName**: この管理ノードのホスト名または IP アドレス。
- **Id**: 管理ノードを識別する番号。現在は非推奨です。かわりに NodeId を使用してください。
- **LogDestination**: ログメッセージの送信先: コンソール、システムログ、または指定されたログファイル。
- **NodeId**: クラスタ内のすべてのノード間で管理ノードを一意に識別する番号。
- **PortNumber**: 管理サーバーにコマンドを送信し、管理サーバーから構成をフェッチするためのポート番号。
- **PortNumberStats**: 管理サーバーから統計情報を取得するために使用されるポート番号。
- **TotalSendBufferMemory**: すべてのトランスポータ送信バッファに使用する合計メモリー。
- **wan**: WAN の TCP 設定をデフォルトとして使用します。

注記

管理ノードの構成に変更を加えたあとは、新しい構成を有効にするために、クラスタのローリング再起動を実行する必要があります。詳細は、[セクション23.3.3.5「NDB Cluster 管理サーバーの定義」](#)を参照してください。

実行中の NDB Cluster に新しい管理サーバーを追加するには、既存の `config.ini` ファイルを変更したあとに、すべてのクラスタノードのローリング再起動も実行する必要があります。複数の管理ノードを使用するときに発生する問題の詳細は、[セクション23.1.7.10「複数の NDB Cluster ノードに関する制限事項」](#)を参照してください。

23.3.2.3 NDB Cluster SQL ノードおよび API ノードの構成パラメータ

このセクションのリストでは、NDB Cluster SQL ノードおよび API ノードを構成するために `config.ini` ファイルの `[mysqld]` および `[api]` セクションで使用されるパラメータについて説明します。各パラメータに関する詳しい説明およびその他の追加情報については、[セクション23.3.3.7「NDB Cluster での SQL およびその他の API ノードの定義」](#)を参照してください。

- **ApiVerbose**: NDB API のデバッグを有効にします (NDB 開発用)。
- **ArbitrationDelay**: アービトレートするよう求められた場合、アービトレータは投票前にこのミリ秒数待機。
- **ArbitrationRank**: 0 の場合、API ノードはアービトレータではありません。カーネルがアービトレータを選択する順序 1, 2。
- **AutoReconnect**: クラスタから切断されたときに API ノードを完全に再接続するかどうかを指定。
- **BatchByteSize**: デフォルトのバッチサイズ (バイト)。
- **BatchSize**: レコード数のデフォルトバッチサイズ。
- **ConnectBackoffMaxTime**: この API ノードによる特定のデータノードへの接続試行間の最長時間をミリ秒単位で指定します (最大 100 ミリ秒の解決)。接続試行の進行中に経過した時間を除外します。最悪の場合は数秒かかることがあります。0 に設定すると無効になります。この API ノードに現在接続されているデータノードがない場合は、代わりに StartConnectBackoffMaxTime が使用されます。
- **ConnectionMap**: 接続するデータノードを指定します。
- **DefaultHashMapSize**: テーブルハッシュマップに使用するサイズ (バケット単位) を設定します。3 つの値がサポートされています: 0、240 および 3840。
- **DefaultOperationRedoProblemAction**: RedoOverCommitCounter を超えた場合の操作の処理方法。

- **ExecuteOnComputer**: 以前に定義された COMPUTER を参照する文字列。
- **ExtraSendBufferMemory**: TotalSendBufferMemory または SendBufferMemory によって割り当てられたものに加えて、送信バッファに使用するメモリー。デフォルト (0) では最大 16MB まで許可されます。
- **HeartbeatThreadPriority**: ハートビートスレッドポリシーおよび API ノードの優先度を設定します。使用可能な値についてはマニュアルを参照してください。
- **HostName**: この SQL または API ノードのホスト名または IP アドレス。
- **Id**: MySQL サーバーまたは API ノード (Id) を識別する番号。現在は非推奨です。かわりに NodeId を使用してください。
- **MaxScanBatchSize**: 1 つのスキャンの最大集合バッチサイズ。
- **NodeId**: クラスタ内のすべてのノード間で SQL ノードまたは API ノードを一意に識別する番号。
- **StartConnectBackoffMaxTime**: ConnectBackoffMaxTime と同じですが、この API ノードにデータノードが接続されていない場合にこのパラメータがかわりに使用される点が異なります。
- **TotalSendBufferMemory**: すべてのトランスポート送信バッファに使用する合計メモリー。
- **wan**: WAN の TCP 設定をデフォルトとして使用します。

NDB Cluster の MySQL サーバーオプションについては、[NDB Cluster の MySQL Server オプション](#) を参照してください。NDB Cluster に関連する MySQL サーバースystem変数については、[NDB Cluster システム変数](#) を参照してください。

注記

実行中の NDB Cluster の構成に新しい SQL または API ノードを追加するには、新しい `[mysqld]` または `[api]` セクションを `config.ini` ファイル (または、複数の管理サーバーを使用している場合はファイル) に追加したあとで、すべてのクラスタノードのローリング再起動を実行する必要があります。これは、新しい SQL または API ノードをクラスタに接続する前に実行する必要があります。

新しい SQL または API ノードがクラスタ構成内の以前に使用されていない API スロットを使用してクラスタに接続する場合、クラスタの再起動を実行する必要はありません。

23.3.2.4 その他の NDB Cluster 構成パラメータ

このセクションのリストでは、NDB Cluster を構成するために `config.ini` ファイルの `[computer]`、`[tcp]`、および `[shm]` セクションで使用されるパラメータについて説明します。個々のパラメータの詳細および追加情報は、必要に応じて [セクション 23.3.3.10 「NDB Cluster TCP/IP 接続」](#) または [セクション 23.3.3.12 「NDB Cluster の共有メモリー接続」](#) を参照してください。

`config.ini` ファイルの `[computer]` セクションには、次のパラメータが適用されます:

- **HostName**: このコンピュータのホスト名または IP アドレス。
- **Id**: このコンピュータの一意的識別子。

`config.ini` ファイルの `[tcp]` セクションには、次のパラメータが適用されます:

- **AllowUnresolvedHostNames**: `false` (デフォルト) の場合、管理ノードによるホスト名の解決に失敗すると致命的なエラーが発生します。`true` の場合、未解決のホスト名は警告としてのみ報告されます。
- **Checksum**: チェックサムが有効になっている場合、ノード間のすべてのシグナルでエラーがチェックされます。
- **Group**: グループ近接性に使用されます。値が小さいほど、近接していると解釈されます。
- **NodeId1**: 接続の片側のノード (データノード、API ノード、または管理ノード) の ID。
- **NodeId2**: 接続の片側のノード (データノード、API ノード、または管理ノード) の ID。
- **NodeIdServer**: TCP 接続のサーバー側の設定。

- **OverloadLimit**: この数を超える未送信バイトが送信バッファにある場合、接続は過負荷とみなされます。
- **PreSendChecksum**: このパラメータとチェックサムの間が有効になっている場合は、送信前チェックサムチェックを実行し、ノード間のすべての TCP シグナルにエラーがないかどうかを確認。
- **Proxy**: .
- **ReceiveBufferMemory**: このノードが受信したシグナルのバッファのバイト数。
- **SendBufferMemory**: このノードから送信されたシグナルの TCP バッファのバイト数。
- **SendSignalId**: 各シグナルで ID を送信します。トレースファイルで使用されます。デバッグビルドでは、デフォルトは true です。
- **TCP_MAXSEG_SIZE**: TCP_MAXSEG に使用される値。
- **TCP_RCV_BUF_SIZE**: SO_RCVBUF に使用される値。
- **TCP_SND_BUF_SIZE**: SO_SNDBUF に使用される値。
- **TcpBind_INADDR_ANY**: 接続のサーバー部分のホスト名のかわりに InAddrAny をバインド。

config.ini ファイルの[shm]セクションには、次のパラメータが適用されます:

- **Checksum**: チェックサムが有効になっている場合、ノード間のすべてのシグナルでエラーがチェックされます。
- **Group**: グループ近接性に使用されます。値が小さいほど、近接していると解釈されます。
- **NodeId1**: 接続の片側のノード (データノード、API ノード、または管理ノード) の ID。
- **NodeId2**: 接続の片側のノード (データノード、API ノード、または管理ノード) の ID。
- **NodeIdServer**: SHM 接続のサーバー側の設定。
- **OverloadLimit**: この数を超える未送信バイトが送信バッファにある場合、接続は過負荷とみなされます。
- **PreSendChecksum**: このパラメータとチェックサムの間が有効になっている場合は、送信前チェックサムチェックを実行し、ノード間のすべての SHM シグナルにエラーがないかどうかを確認。
- **SendBufferMemory**: このノードから送信されたシグナルの共有メモリーバッファ内のバイト数。
- **SendSignalId**: 各シグナルで ID を送信します。トレースファイルで使用。
- **ShmKey**: 共有メモリーキー。1 に設定すると、NDB によって計算されます。
- **ShmSpinTime**: 受信時に、スリープするまでにスピンするミリ秒数。
- **ShmSize**: 共有メモリーセグメントのサイズ。
- **Signum**: シグナリングに使用されるシグナル番号。

23.3.2.5 NDB Cluster mysqld オプションおよび変数のリファレンス

次のテーブルに、NDB Cluster で SQL ノードとして実行されている場合に `mysqld` 内で適用可能なコマンド行オプション、サーバー、およびステータス変数のリストを示します。 `mysqld` で使用できるすべてのコマンド行オプションとサーバーおよびステータス変数を示す表については、[セクション5.1.4「サーバーオプション、システム変数およびステータス変数リファレンス」](#)を参照してください。

- **Com_show_ndb_status**: SHOW NDB STATUS ステートメントの数。
- **Handler_discover**: テーブルが検出された回数。
- **ndb-batch-size**: NDB トランザクションバッチで使用されるサイズ (バイト)。
- **ndb-blob-read-batch-bytes**: 大規模な BLOB 読み取りがバッチ化されるサイズをバイトで指定します。0 = 制限なし。
- **ndb-blob-write-batch-bytes**: 大規模な BLOB 書き込みがバッチ化されるサイズをバイトで指定します。0 = 制限なし。
- **ndb-cluster-connection-pool**: MySQL で使用されるクラスタへの接続数。

- **ndb-cluster-connection-pool-nodeids**: MySQL で使用されるクラスタへの接続用のノード ID のカンマ区切りリスト。リスト内のノード数は `--ndb-cluster-connection-pool` に設定された値と一致する必要があります。
- **ndb-connectstring**: このクラスタの構成情報を配布する NDB 管理サーバーのアドレス。
- **ndb-default-column-format**: テーブルのカラムを作成または追加する場合は、`COLUMN_FORMAT` および `ROW_FORMAT` オプションにデフォルトでこの値 (`FIXED` または `DYNAMIC`) を使用。
- **ndb-deferred-constraints**: 一意のインデックスに関する制約チェック (サポートされている場合) を、コミット時まで遅延させることを指定します。通常は必要なく、使用されません。テストの目的にのみ使用されます。
- **ndb-distribution**: NDBCLUSTER の新しいテーブルのデフォルトの配布 (`KEYHASH` または `LINHASH`、デフォルトは `KEYHASH`)。
- **ndb-log-apply-status**: レプリカとして機能する MySQL サーバーが、即時ソースから受信した `mysql.ndb_apply_status` 更新を、独自のサーバー ID を使用して独自のバイナリログに記録します。 `--ndbcluster` オプションを指定してサーバーを起動した場合にのみ有効です。
- **ndb-log-empty-epochs**: 有効にすると、`--log-slave-updates` が有効になっている場合でも、変更がなかったエポックが `ndb_apply_status` および `ndb_binlog_index` テーブルに書き込まれます。
- **ndb-log-empty-update**: 有効にすると、`--log-slave-updates` が有効な場合でも、変更を生成しなかった更新が `ndb_apply_status` および `ndb_binlog_index` テーブルに書き込まれます。
- **ndb-log-exclusive-reads**: 排他ロックを使用する主キーの読み取りをログに記録します。読み取りの競合に基づく競合解決を可能にします。
- **ndb-log-fail-terminate**: 見つかったすべての行イベントの完全なロギングが不可能な場合は `mysqld` プロセスを終了。
- **ndb-log-orig**: 発信サーバー ID とエポックを `mysql.ndb_binlog_index` テーブルに記録します。
- **ndb-log-transaction-id**: NDB トランザクション ID をバイナリログに書き込みます。 `--log-bin-v1-events=OFF` が必要です。
- **ndb-log-update-as-write**: ソースでの更新のロギングを更新 (`OFF`) と書き込み (`ON`) の間で切り替えます。
- **ndb-mgmd-host**: 管理サーバーに接続するためのホスト (および必要に応じてポート) の設定。
- **ndb-nodeid**: この MySQL サーバーの NDB Cluster ノード ID。
- **ndb-transid-mysql-connection-map**: `ndb_transid_mysql_connection_map` プラグインを有効または無効にします。つまり、その名前の `INFORMATION_SCHEMA` テーブルを有効または無効にします。
- **ndb-wait-connected**: MySQL サーバーが MySQL クライアント接続を受け入れる前にクラスタ管理およびデータノードへの接続を待機する時間 (秒)。
- **ndb-wait-setup**: NDB エンジンの設定が完了するまで MySQL サーバーが待機する時間 (秒)。
- **ndb-allow-copying-alter-table**: `ALTER TABLE` が NDB テーブルでコピー操作を使用しないようにするには、`OFF` に設定。
- **Ndb_api_adaptive_send_deferred_count**: この MySQL Server (SQL ノード) によって実際に送信されなかった適応送信コールの数。
- **Ndb_api_adaptive_send_deferred_count_slave**: このクライアントセッションで実際に送信されなかった適応送信コールの数。
- **Ndb_api_adaptive_send_deferred_count_replica**: このレプリカによって実際に送信されていない適応送信コールの数。
- **Ndb_api_adaptive_send_deferred_count_slave**: このレプリカによって実際に送信されていない適応送信コールの数。
- **Ndb_api_adaptive_send_forced_count**: この MySQL Server (SQL ノード) によって送信された強制送信セットを含む適応送信の数。
- **Ndb_api_adaptive_send_forced_count_session**: このクライアントセッションで強制送信が設定されている適応送信の数。

- `Ndb_api_adaptive_send_forced_count_replica`: このレプリカによって送信された強制送信セットを含む適応送信の数.
- `Ndb_api_adaptive_send_forced_count_slave`: このレプリカによって送信された強制送信セットを含む適応送信の数.
- `Ndb_api_adaptive_send_unforced_count`: この MySQL Server (SQL ノード) によって強制送信されなかった適応送信の数.
- `Ndb_api_adaptive_send_unforced_count_session`: このクライアントセッションでの強制送信なしの適応送信の数.
- `Ndb_api_adaptive_send_unforced_count_replica`: このレプリカによって強制送信されていない適応送信の数.
- `Ndb_api_adaptive_send_unforced_count_slave`: このレプリカによって強制送信されていない適応送信の数.
- `Ndb_api_bytes_received_count`: この MySQL Server (SQL ノード) によってデータノードから受信されたデータの量 (バイト).
- `Ndb_api_bytes_received_count_session`: このクライアントセッションでデータノードから受信されたデータの量 (バイト単位).
- `Ndb_api_bytes_received_count_replica`: このレプリカによってデータノードから受信されたデータの量 (バイト).
- `Ndb_api_bytes_received_count_slave`: このレプリカによってデータノードから受信されたデータの量 (バイト).
- `Ndb_api_bytes_sent_count`: この MySQL Server (SQL ノード) によってデータノードに送信されたデータの量 (バイト単位).
- `Ndb_api_bytes_sent_count_session`: このクライアントセッションでデータノードに送信されたデータの量 (バイト単位).
- `Ndb_api_bytes_sent_count_replica`: このレプリカによってデータノードに送信されたデータの数量 (バイト).
- `Ndb_api_bytes_sent_count_slave`: このレプリカによってデータノードに送信されたデータの数量 (バイト).
- `Ndb_api_event_bytes_count`: この MySQL Server (SQL ノード) によって受信されたイベントのバイト数.
- `Ndb_api_event_bytes_count_injector`: NDB バイナリロギングジェクタスレッドが受信したイベントデータのバイト数.
- `Ndb_api_event_data_count`: この MySQL Server (SQL ノード) によって受信された行変更イベントの数.
- `Ndb_api_event_data_count_injector`: NDB バイナリロギングジェクタスレッドによって受信された行変更イベントの数.
- `Ndb_api_event_nondata_count`: この MySQL Server (SQL ノード) によって受信された、行変更イベント以外のイベントの数.
- `Ndb_api_event_nondata_count_injector`: NDB バイナリロギングジェクタスレッドによって受信された行変更イベント以外のイベントの数.
- `Ndb_api_pk_op_count`: この MySQL Server (SQL ノード) による主キーに基づく、または主キーを使用する操作の数.
- `Ndb_api_pk_op_count_session`: このクライアントセッションで、主キーに基づいた、または主キーを使用した操作の数.
- `Ndb_api_pk_op_count_replica`: このレプリカによる主キーに基づく、または主キーを使用する操作の数.
- `Ndb_api_pk_op_count_slave`: このレプリカによる主キーに基づく、または主キーを使用する操作の数.
- `Ndb_api_pruned_scan_count`: この MySQL Server (SQL ノード) によってパーティションにプルーニングされたスキャンの数.
- `Ndb_api_pruned_scan_count_session`: このクライアントセッションで 1 つのパーティションにプルーニングされたスキャンの数.
- `Ndb_api_pruned_scan_count_replica`: このレプリカによって 1 つのパーティションにプルーニングされたスキャンの数.

- [Ndb_api_pruned_scan_count_slave](#): このレプリカによって 1 つのパーティションにプルーニングされたスキャンの数.
- [Ndb_api_range_scan_count](#): この MySQL Server (SQL ノード) によって開始されたレンジスキャンの数.
- [Ndb_api_range_scan_count_session](#): このクライアントセッションで開始された範囲スキャンの回数.
- [Ndb_api_range_scan_count_replica](#): このレプリカによって開始されたレンジスキャンの数.
- [Ndb_api_range_scan_count_slave](#): このレプリカによって開始されたレンジスキャンの数.
- [Ndb_api_read_row_count](#): この MySQL Server (SQL ノード) によって読み取られた行の合計数.
- [Ndb_api_read_row_count_session](#): このクライアントセッションで読み取られた行の合計数.
- [Ndb_api_read_row_count_replica](#): このレプリカによって読み取られた行の合計数.
- [Ndb_api_read_row_count_slave](#): このレプリカによって読み取られた行の合計数.
- [Ndb_api_scan_batch_count](#): この MySQL Server (SQL ノード) によって受信された行のバッチ数.
- [Ndb_api_scan_batch_count_session](#): このクライアントセッションで受信されたバッチの行数.
- [Ndb_api_scan_batch_count_replica](#): このレプリカによって受信された行のバッチ数.
- [Ndb_api_scan_batch_count_slave](#): このレプリカによって受信された行のバッチ数.
- [Ndb_api_table_scan_count](#): この MySQL Server (SQL ノード) によって開始されたテーブルスキャンの数 (内部テーブルのスキャンを含む).
- [Ndb_api_table_scan_count_session](#): このクライアントセッションで開始された、内部テーブルのスキャンを含むテーブルスキャンの回数.
- [Ndb_api_table_scan_count_replica](#): このレプリカによって開始されたテーブルスキャンの数 (内部テーブルのスキャンを含む).
- [Ndb_api_table_scan_count_slave](#): このレプリカによって開始されたテーブルスキャンの数 (内部テーブルのスキャンを含む).
- [Ndb_api_trans_abort_count](#): この MySQL Server (SQL ノード) によって中断されたトランザクションの数.
- [Ndb_api_trans_abort_count_session](#): このクライアントセッションで中止されたトランザクションの数.
- [Ndb_api_trans_abort_count_replica](#): このレプリカによって中断されたトランザクションの数.
- [Ndb_api_trans_abort_count_slave](#): このレプリカによって中断されたトランザクションの数.
- [Ndb_api_trans_close_count](#): この MySQL Server (SQL ノード) によって中断されたトランザクションの数 (TransCommitCount および TransAbortCount の合計より大きい可能性があります).
- [Ndb_api_trans_close_count_session](#): このクライアントセッションで中断されたトランザクションの数 (TransCommitCount と TransAbortCount の合計より大きい可能性があります).
- [Ndb_api_trans_close_count_replica](#): このレプリカによって中断されたトランザクションの数 (TransCommitCount と TransAbortCount の合計より大きい可能性があります).
- [Ndb_api_trans_close_count_slave](#): このレプリカによって中断されたトランザクションの数 (TransCommitCount と TransAbortCount の合計より大きい可能性があります).
- [Ndb_api_trans_commit_count](#): この MySQL Server (SQL ノード) によってコミットされたトランザクションの数.
- [Ndb_api_trans_commit_count_session](#): このクライアントセッションでコミットされたトランザクションの数.
- [Ndb_api_trans_commit_count_replica](#): このレプリカによってコミットされたトランザクションの数.
- [Ndb_api_trans_commit_count_slave](#): このレプリカによってコミットされたトランザクションの数.
- [Ndb_api_trans_local_read_row_count](#): この MySQL Server (SQL ノード) によって読み取られた行の合計数.

- `Ndb_api_trans_local_read_row_count_session`: このクライアントセッションで読み取られた行の合計数。
- `Ndb_api_trans_local_read_row_count_replica`: このレプリカによって読み取られた行の合計数。
- `Ndb_api_trans_local_read_row_count_slave`: このレプリカによって読み取られた行の合計数。
- `Ndb_api_trans_start_count`: この MySQL Server (SQL ノード) によって開始されたトランザクションの数。
- `Ndb_api_trans_start_count_session`: このクライアントセッションで開始されたトランザクションの数。
- `Ndb_api_trans_start_count_replica`: このレプリカによって開始されたトランザクションの数。
- `Ndb_api_trans_start_count_slave`: このレプリカによって開始されたトランザクションの数。
- `Ndb_api_uk_op_count`: この MySQL Server (SQL ノード) による一意キーに基づく、または一意キーを使用する操作の数。
- `Ndb_api_uk_op_count_session`: このクライアントセッションで、一意のキーに基づいた、または一意のキーを使用した操作の数。
- `Ndb_api_uk_op_count_replica`: このレプリカによる一意キーに基づく、または一意キーを使用する操作の数。
- `Ndb_api_uk_op_count_slave`: このレプリカによる一意キーに基づく、または一意キーを使用する操作の数。
- `Ndb_api_wait_exec_complete_count`: この MySQL Server (SQL ノード) による操作実行の完了を待機中にスレッドがブロックされた回数。
- `Ndb_api_wait_exec_complete_count_session`: このクライアントセッションで操作実行の完了を待機中にスレッドがブロックされた回数。
- `Ndb_api_wait_exec_complete_count_replica`: このレプリカによる操作実行の完了を待機中にスレッドがブロックされた回数。
- `Ndb_api_wait_exec_complete_count_slave`: このレプリカによる操作実行の完了を待機中にスレッドがブロックされた回数。
- `Ndb_api_wait_meta_request_count`: この MySQL Server (SQL ノード) によるメタデータベースのシグナルを待機してスレッドがブロックされた回数。
- `Ndb_api_wait_meta_request_count_session`: このクライアントセッションでメタデータベースのシグナルを待機してスレッドがブロックされた回数。
- `Ndb_api_wait_meta_request_count_replica`: このレプリカによるメタデータベースのシグナルを待機してスレッドがブロックされた回数。
- `Ndb_api_wait_meta_request_count_slave`: このレプリカによるメタデータベースのシグナルを待機してスレッドがブロックされた回数。
- `Ndb_api_wait_nanos_count`: この MySQL Server (SQL ノード) によってデータノードからの何らかのタイプのシグナルの待機に費やされた合計時間 (ナノ秒)。
- `Ndb_api_wait_nanos_count_session`: このクライアントセッションでデータノードからの何らかのタイプのシグナルの待機に費やされた合計時間 (ナノ秒)。
- `Ndb_api_wait_nanos_count_replica`: このレプリカによるデータノードからの何らかのタイプのシグナルの待機に費やされた合計時間 (ナノ秒)。
- `Ndb_api_wait_nanos_count_slave`: このレプリカによるデータノードからの何らかのタイプのシグナルの待機に費やされた合計時間 (ナノ秒)。
- `Ndb_api_wait_scan_result_count`: この MySQL Server (SQL ノード) によるスキャンベースのシグナルの待機中にスレッドがブロックされた回数。
- `Ndb_api_wait_scan_result_count_session`: このクライアントセッションでスキャンベースのシグナルを待機している間にスレッドがブロックされた回数。
- `Ndb_api_wait_scan_result_count_replica`: このレプリカによるスキャンベースのシグナルの待機中にスレッドがブロックされた回数。

- `Ndb_api_wait_scan_result_count_slave`: このレプリカによるスキャンベースのシグナルの待機中にスレッドがブロックされた回数。
- `ndb_autoincrement_prefetch_sz`: NDB 自動インクリメント値のプリフェッチサイズ。
- `ndb_cache_check_time`: MySQL クエリーキャッシュによって作成されたクラスタ SQL ノードのチェック間のミリ秒数。
- `ndb_clear_apply_status`: RESET SLAVE/RESET REPLICA で `ndb_apply_status` テーブルからすべての行をクリアします。デフォルトは ON です。
- `Ndb_cluster_node_id`: NDB Cluster SQL ノードとして動作している場合のこのサーバーのノード ID。
- `Ndb_config_from_host`: NDB Cluster 管理サーバーのホスト名または IP アドレス。
- `Ndb_config_from_port`: NDB Cluster 管理サーバーに接続するためのポート。
- `Ndb_config_generation`: クラスタの現在の構成の生成番号。
- `Ndb_conflict_fn_epoch`: `NDB$EPOCH()` 競合検出関数によって競合が検出された行数。
- `Ndb_conflict_fn_epoch2`: `NDB$EPOCH2()` 競合検出関数によって競合が検出された行数。
- `Ndb_conflict_fn_epoch2_trans`: `NDB$EPOCH2_TRANS()` 競合検出関数によって競合が検出された行数。
- `Ndb_conflict_fn_epoch_trans`: `NDB$EPOCH_TRANS()` 競合検出関数によって競合が検出された行数。
- `Ndb_conflict_fn_max`: サーバーがクラスタレプリケーションに関与する NDB Cluster の一部である場合に、「より大きなタイムスタンプが優先される」に基づく競合解決が適用された回数。
- `Ndb_conflict_fn_old`: このサーバーがクラスタレプリケーションに関与する NDB Cluster の一部である場合に、「同じタイムスタンプが優先」競合解決が適用された回数。
- `Ndb_conflict_last_conflict_epoch`: 競合が検出された、このレプリカ上の最新の NDB エポック。
- `Ndb_conflict_last_stable_epoch`: トランザクション競合関数によって競合していることが検出された行数。
- `Ndb_conflict_reflected_op_discard_count`: 実行中にエラーが発生したために適用されなかった反映操作の数。
- `Ndb_conflict_reflected_op_prepare_count`: 実行の準備が完了した、受信したリフレクト操作の数。
- `Ndb_conflict_refresh_op_count`: 準備されたリフレッシュ操作の数。
- `ndb_conflict_role`: レプリカが競合検出および解決で果たす役割。値は PRIMARY、SECONDARY、PASS、NONE (デフォルト) のいずれかです。レプリケーション SQL スレッドが停止している場合にのみ変更できます。詳細は、ドキュメントを参照してください。
- `Ndb_conflict_trans_conflict_commit_count`: トランザクション競合処理を必要としたあとでコミットされたエポックトランザクションの数。
- `Ndb_conflict_trans_detect_iter_count`: エポックトランザクションのコミットに必要な内部反復数。
`Ndb_conflict_trans_conflict_commit_count` より (わずかに) 大きいか、または等しい値にしてください。
- `Ndb_conflict_trans_reject_count`: トランザクション競合関数によって競合が検出された後に拒否されたトランザクションの数。
- `Ndb_conflict_trans_row_conflict_count`: トランザクション競合関数によって競合が検出された行数。競合するトランザクションに含まれる行または競合するトランザクションに依存する行が含まれます。
- `Ndb_conflict_trans_row_reject_count`: トランザクション競合関数によって競合が検出された後に再編成された行の合計数。`Ndb_conflict_trans_row_conflict_count`、および競合するトランザクションに含まれているか、または依存しているすべての行が含まれます。
- `ndb_data_node_neighbour`: トランザクションヒントおよび完全にレプリケートされたテーブルに対して、この MySQL Server に最も近いクラスタデータノードを指定。
- `ndb_default_column_format`: 新しい NDB テーブルに使用されるデフォルトの行形式およびカラム形式 (FIXED または DYNAMIC) を設定。

- `ndb_deferred_constraints`: 制約チェック (サポートされている場合) を遅延させることを指定します。通常は必要なく、使用されません。テストの目的にのみ使用されます。
- `ndb_dbg_check_shares`: 言語共有のチェック (デバッグビルドのみ)。
- `ndb-schema-dist-timeout`: スキーマ分散中にタイムアウトを検出するまでの待機時間。
- `ndb_distribution`: NDBCLUSTER の新しいテーブルのデフォルトの配布 (KEYHASH または LINHASH、デフォルトは KEYHASH)。
- `Ndb_epoch_delete_delete_count`: 削除の競合が検出された数 (削除操作は適用されましたが、行が存在しません)。
- `ndb_eventbuffer_free_percent`: `ndb_eventbuffer_max_alloc` によって設定された制限に達した後、バッファリングを再開する前にイベントバッファで使用可能にする必要がある空きメモリーの割合。
- `ndb_eventbuffer_max_alloc`: NDB API によってイベントをバッファリングするために割り当てることができる最大メモリー。デフォルトは 0 (無制限) です。
- `Ndb_execute_count`: 操作によって行われた NDB カーネルへのラウンドトリップ数。
- `ndb_extra_logging`: NDB Cluster スキーマ、接続、およびデータ配布イベントの MySQL エラーログへのロギングを制御。
- `ndb_force_send`: NDB に対し、ほかのスレッドを待機せずただちにバッファを送るよう命令。
- `ndb_fully_replicated`: 新しい NDB テーブルが完全にレプリケートされるかどうか。
- `ndb_index_stat_enable`: クエリーの最適化で NDB インデックス統計を使用。
- `ndb_index_stat_option`: NDB インデックス統計の調整可能なオプションのコンマ区切りリスト。list にはスペースを含めないでください。
- `ndb_join_pushdown`: データノードへの結合のプッシュダウンを有効化。
- `Ndb_last_commit_epoch_server`: NDB によって直近でコミットされたエポック。
- `Ndb_last_commit_epoch_session`: この NDB クライアントによって直近でコミットされたエポック。
- `ndb_log_apply_status`: レプリカログとして機能する MySQL サーバーが、自身のサーバー ID を使用して、その即時ソースから受信した `mysql.ndb_apply_status` 更新を独自のバイナリログに記録するかどうか。
- `ndb_log_bin`: NDB テーブルへの更新をバイナリログに書き込みます。--log-bin でバイナリロギングが有効になっている場合にのみ有効です。
- `ndb_log_binlog_index`: エポックとバイナリログの位置の間のマッピングを `ndb_binlog_index` テーブルに挿入します。デフォルトでは ON になります。バイナリロギングが有効な場合にのみ有効です。
- `ndb_log_empty_epochs`: 有効にすると、`log_slave_updates` が有効になっていても、変更がなかったエポックは `ndb_apply_status` テーブルおよび `ndb_binlog_index` テーブルに書き込まれます。
- `ndb_log_empty_update`: 有効にすると、`log_slave_updates` が有効になっている場合でも、変更を生成しない更新は `ndb_apply_status` テーブルおよび `ndb_binlog_index` テーブルに書き込まれます。
- `ndb_log_exclusive_reads`: 排他ロックを使用する主キーの読み取りをログに記録します。読み取りの競合に基づく競合解決を可能にします。
- `ndb_log_orig`: 発信元サーバーの id と epoch が `mysql.ndb_binlog_index` テーブルに記録されるかどうか。mysqld の起動時に --ndb-log-orig オプションを使用して設定。
- `ndb_log_transaction_id`: NDB トランザクション ID がバイナリログに書き込まれるかどうか (読み取り専用)。
- `ndb-log-update-minimal`: 最小限の形式で更新を記録。
- `ndb-log-updated-only`: 完全な行 (ON) または更新のみ (OFF) をログに記録。
- `ndb_metadata_check`: MySQL データディクショナリに関する NDB メタデータ変更の自動検出を有効にします。デフォルトで有効になっています。

- [Ndb_metadata_blacklist_size](#): NDB binlog スレッドが同期に失敗した NDB メタデータオブジェクトの数。NDB 8.0.22 では [Ndb_metadata_excluded_count](#) として名前が変更されました。
- [ndb_metadata_check_interval](#): MySQL データディクショナリに関する NDB メタデータ変更のチェックを実行する間隔 (秒)。
- [Ndb_metadata_detected_count](#): NDB メタデータ変更モニタースレッドが変更を検出した回数。
- [Ndb_metadata_excluded_count](#): NDB binlog スレッドが同期に失敗した NDB メタデータオブジェクトの数。
- [ndb_metadata_sync](#): NDB ディクショナリと MySQL データディクショナリ間のすべての変更の即時同期をトリガーします。これにより、[ndb_metadata_check](#) 値と [ndb_metadata_check_interval](#) 値は無視されます。同期が完了すると false にリセットされます。
- [Ndb_metadata_synced_count](#): 同期された NDB メタデータオブジェクトの数。
- [Ndb_number_of_data_nodes](#): この NDB クラスタ内のデータノードの数。サーバーがクラスタに参加する場合にのみ設定されます。
- [ndb-optimization-delay](#): NDB テーブルで OPTIMIZE TABLE によって一連の行を処理する間に待機するミリ秒数。
- [ndb_optimized_node_selection](#): SQL ノードがトランザクションコーディネータとして使用するクラスタデータノードを選択する方法を決定。
- [Ndb_pruned_scan_count](#): パーティションプルーニングを使用できるクラスタが最後に起動されてから NDB によって実行されたスキャンの数。
- [Ndb_pushed_queries_defined](#): API ノードがデータノードにプッシュダウンしようとした結合の数。
- [Ndb_pushed_queries_dropped](#): API ノードがデータノードにプッシュダウンしようとしたが失敗した結合の数。
- [Ndb_pushed_queries_executed](#): データノードで正常にプッシュダウンおよび実行された結合の数。
- [Ndb_pushed_reads](#): プッシュダウン結合によってデータノードで実行された読み取りの数。
- [ndb_read_backup](#): すべての NDB テーブルの任意のレプリカからの読み取りを有効にします。個々の NDB テーブルを有効または無効にするには、CREATE TABLE または ALTER TABLE とともに NDB_TABLE=READ_BACKUP={0|1} を使用。
- [ndb_recv_thread_activation_threshold](#): 受信スレッドがクラスタ接続のポーリングを引き継ぐときのアクティブ化しきい値 (同時にアクティブなスレッドで測定)。
- [ndb_recv_thread_cpu_mask](#): 受信スレッドを特定の CPU にロックするための CPU マスク。16 進数で指定します。詳細は、ドキュメントを参照してください。
- [Ndb_replica_max_replicated_epoch](#): このレプリカで最近コミットされた NDB エポック。この値が [Ndb_conflict_last_conflict_epoch](#) 以上の場合、競合はまだ検出されていません。
- [ndb_report_thresh_binlog_epoch_slip](#): NDB 7.5.4 以降: NDB 7.5.4 の前に BUFFERED_EPOCHS_OVER_THRESHOLD イベントバッファーステータスメッセージを生成する、完全にバッファリングされたがまだ消費されていないエポックの数のしきい値 (超過した場合): バイナリログステータスをレポートする前に遅れるエポック数のしきい値。
- [ndb_report_thresh_binlog_mem_usage](#): バイナリログステータスを報告する前に残っている空きメモリの割合のしきい値。
- [ndb_row_checksum](#): 有効にすると、行チェックサムが設定され、デフォルトで有効になります。
- [Ndb_scan_count](#): クラスタが最後に起動されてから NDB によって実行されたスキャンの合計数。
- [ndb_schema_dist_lock_wait_timeout](#): エラーが返される前にロックを待機するスキーマ配布中の時間。
- [ndb_schema_dist_timeout](#): スキーマ分散中にタイムアウトを検出するまでの待機時間。
- [ndb_schema_dist_upgrade_allowed](#): NDB への接続時にスキーマ配布テーブルのアップグレードを許可。
- [ndb_show_foreign_key_mock_tables](#): [foreign_key_checks=0](#) のサポートに使用されるモックテーブルを表示。

- `ndb_slave_conflict_role`: レプリカが競合検出および解決で果たす役割。値は PRIMARY、SECONDARY、PASS、NONE (デフォルト) のいずれかです。レプリケーション SQL スレッドが停止している場合にのみ変更できます。詳細は、ドキュメントを参照してください。
- `Ndb_slave_max_replicated_epoch`: このレプリカで最近コミットされた NDB エポック。この値が `Ndb_conflict_last_conflict_epoch` 以上の場合、競合はまだ検出されていません。
- `Ndb_system_name`: 構成されたクラスタシステム名。サーバーが NDB に接続されていない場合は空。
- `ndb_table_no_logging`: この設定が有効であるときに作成された NDB テーブルはディスクにチェックポイントされません (テーブルスキーマファイルは作成されます)。NDBCLUSTER を使用してテーブルが作成または変更されたときに有効になる設定は、テーブルの存続期間中保持されます。
- `ndb_table_temporary`: NDB テーブルはディスクで永続的ではありません: スキーマファイルは作成されず、テーブルは記録されません。
- `Ndb_trans_hint_count_session`: このセッションで開始されたヒントを使用するトランザクションの数。
- `ndb_use_copying_alter_table`: NDB Cluster で ALTER TABLE 操作のコピーを使用。
- `ndb_use_exact_count`: クエリーを計画する際に正確な行数を使用。
- `ndb_use_transactions`: SELECT COUNT(*) クエリーの高速化を計画するときに、レコードカウントを使用するよう NDB に指示。
- `ndb_version`: ビルドおよび NDB エンジンのバージョンを整数で表示します。
- `ndb_version_string`: NDB エンジンのバージョンを含むビルド情報を `ndb-x.y.z` の形式で表示します。
- `ndbcluster`: NDB Cluster を有効化 (MySQL のこのバージョンがサポートする場合)。無効化 `--skip-ndbcluster`。
- `ndbinfo`: `ndbinfo` プラグインの有効化 (サポートされている場合)。
- `ndbinfo_database`: NDB 情報データベースに使用される名前。読み取り専用。
- `ndbinfo_max_bytes`: デバッグにのみ使用されます。
- `ndbinfo_max_rows`: デバッグにのみ使用されます。
- `ndbinfo_offline`: `ndbinfo` データベースをオフラインモードにする (テーブルまたはビューから行が返されない)。
- `ndbinfo_show_hidden`: mysql クライアントで `ndbinfo` 内部ベーステーブルを表示するかどうか。デフォルトは OFF です。
- `ndbinfo_table_prefix`: `ndbinfo` 内部ベーステーブルの命名に使用する接頭辞。読み取り専用。
- `ndbinfo_version`: `ndbinfo` エンジンバージョン;読み取り専用。
- `server_id_bits`: NDB API アプリケーションがアプリケーションデータをもっとも重要なビットに格納できるように、サーバーの識別に実際に使用される `server_id` 内の最下位ビットの数。この値を累乗するには、`server_id` が 2 未満である必要があります。
- `skip-ndbcluster`: NDB Cluster ストレージエンジンを無効にします。
- `slave_allow_batching`: レプリカのバッチ更新のオンとオフを切り替えます。
- `transaction_allow_batching`: 1 つのトランザクション内でのステートメントのバッチ処理を許可します。使用するには AUTOCOMMIT を無効にします。

23.3.3 NDB Cluster 構成ファイル

NDB Cluster を構成するには、次の 2 つのファイルを操作する必要があります:

- `my.cnf`: すべての NDB Cluster 実行可能ファイルのオプションを指定します。このファイルは、(これまでの MySQL の使用経験からご存じのとおり) クラスタ内で実行されている個々の実行可能ファイルからアクセスできる必要があります。

- **config.ini**: このファイル (グローバル構成ファイルとも呼ばれる) は NDB Cluster 管理サーバーによって読み取り専用になされ、NDB Cluster 管理サーバーはこれに含まれる情報をクラスタに参加しているすべてのプロセスに配布します。config.ini には、クラスタに含まれる各ノードの説明が含まれています。これには、データノードに関する構成パラメータと、クラスタ内のすべてのノード間の接続に関する構成パラメータが含まれます。このファイルに含まれる可能性があるセクションと各セクションに配置される構成パラメータの種類を簡単に確認するには、「[config.ini ファイルのセクション](#)」を参照してください。

構成データのキャッシュ。 NDB では、ステートフル構成を使用します。管理サーバーは、再起動されるたびにグローバル構成ファイルを読み取るのではなく、最初の起動時に構成をキャッシュし、その後は次の条件のいずれかが true である場合にのみグローバル構成ファイルを読み取ります。

- **--initial** オプションを使用すると管理サーバーが起動します。 **--initial** を使用すると、グローバル構成ファイルが再度読み取られ、既存のキャッシュファイルが削除され、管理サーバーによって新しい構成キャッシュが作成されません。
- **--reload** オプションを使用すると管理サーバーが起動します。 **--reload** オプションを使用すると、管理サーバーはそのキャッシュをグローバル構成ファイルと比較します。これらが異なる場合は、管理サーバーによって新しい構成キャッシュが作成されます。既存の構成キャッシュは保持されますが、使用されません。管理サーバーキャッシュとグローバル構成ファイルに同じ構成データが含まれている場合、既存のキャッシュが使用され、新しいキャッシュは作成されません。
- 管理サーバーは **--config-cache=FALSE** を使用して起動されます。 これにより、**--config-cache**(デフォルトで有効) が無効になり、管理サーバーが構成キャッシュを完全にバイパスするように強制するために使用できます。この場合、管理サーバーは存在する可能性がある構成ファイルを無視し、常に **config.ini** ファイルからその構成データを読み取ります。
- 構成キャッシュが見つかりません。 この場合、管理サーバーはグローバル構成ファイルを読み取り、そのファイルと同じ構成データを含むキャッシュを作成します。

構成キャッシュファイル。 管理サーバーは、デフォルトで MySQL インストールディレクトリ内の **mysql-cluster** という名前のディレクトリに構成キャッシュファイルを作成します。(Unix システム上のソースから NDB Cluster を構築する場合、デフォルトの場所は **/usr/local/mysql-cluster** です。) これは、**--configdir** オプションを指定して管理サーバーを起動すると、実行時にオーバーライドできます。構成キャッシュファイルは、**ndb_node_id_config.bin.seq_id** というパターンで命名されるバイナリファイルです。**node_id** は管理サーバーのクラスタ内のノード ID で、**seq_id** はキャッシュの識別子です。キャッシュファイルには、作成された順に **seq_id** を使用して連番が付けられます。管理サーバーは、**seq_id** で特定される最新のキャッシュファイルを使用します。

注記

あとの構成キャッシュファイルを削除するか、**seq_id** が大きくなるように前のキャッシュファイルの名前を変更すると、前の構成にロールバックできます。ただし、構成キャッシュファイルはバイナリ形式で書き込まれるため、その内容を手動で編集しないでください。

NDB Cluster 管理サーバーの **--configdir**、**--config-cache**、**--initial** および **--reload** オプションの詳細は、[セクション 23.4.4 「ndb_mgmd — NDB Cluster 管理サーバーデーモン」](#) を参照してください。

NDB Cluster 構成を継続的に改善し、このプロセスを簡略化しようとしています。下位互換性の維持に努めていますが、場合によっては互換性のない変更が行われる可能性があります。このような場合は、変更の下位互換性がない場合に NDB Cluster ユーザーに事前に知らせようとしています。このような変更を発見し、それに関する情報が提供されていない場合は、[セクション 1.6 「質問またはバグをレポートする方法」](#) に示した手順を使用して MySQL バグデータベースに報告してください。

23.3.3.1 NDB Cluster 構成: 基本例

NDB Cluster をサポートするには、次の例に示すように **my.cnf** を更新するようにしてください。これらのパラメータを実行可能ファイルの起動時にコマンド行に指定することもできます。

注記

ここに示すオプションを、**config.ini** グローバル構成ファイルで使用されるものと混同しないようにしてください。グローバル構成オプションについては、このセクションで後述します。


```
# my.cnf
# example additions to my.cnf for NDB Cluster
# (valid in MySQL 8.0)

# enable ndbcluster storage engine, and provide connection string for
# management server host (default port is 1186)
[mysqld]
ndbcluster
ndb-connectstring=ndb_mgmd.mysql.com

# provide connection string for management server host (default port: 1186)
[ndbd]
connect-string=ndb_mgmd.mysql.com

# provide connection string for management server host (default port: 1186)
[ndb_mgm]
connect-string=ndb_mgmd.mysql.com

# provide location of cluster configuration file
[ndb_mgmd]
config-file=/etc/config.ini
```

(接続文字列の詳細は、[セクション23.3.3.3「NDB Cluster 接続文字列」](#)を参照してください。)

```
# my.cnf
# example additions to my.cnf for NDB Cluster
# (works on all versions)

# enable ndbcluster storage engine, and provide connection string for management
# server host to the default port 1186
[mysqld]
ndbcluster
ndb-connectstring=ndb_mgmd.mysql.com:1186
```

重要

前に示したように、`my.cnf` ファイルの `[mysqld]` に `NDBCLUSTER` および `ndb-connectstring` パラメータを指定して `mysqld` プロセスを起動したあとは、クラスタを実際に起動しないと、`CREATE TABLE` または `ALTER TABLE` ステートメントを実行できません。そうしないと、これらのステートメントはエラーで失敗します。これは意図的なものです。

クラスタの `my.cnf` ファイルでは、すべての実行可能ファイルによって読み取られ、使用される設定で別個の `[mysql_cluster]` セクションを使用することもできます。

```
# cluster-specific settings
[mysql_cluster]
ndb-connectstring=ndb_mgmd.mysql.com:1186
```

`my.cnf` ファイルに設定できる追加の `NDB` 変数については、[NDB Cluster システム変数](#)を参照してください。

NDB Cluster のグローバル構成ファイルは、`config.ini` という命名規則に従っています(ただし、これは必須ではありません)。これは、必要に応じて `ndb_mgmd` が起動時に読み取るため、読み取りが可能な任意の場所に配置できます。構成の場所と名前は、コマンド行の `ndb_mgmd` で `--config-file=path_name` を使用して指定します。このオプションにデフォルト値はなく、`ndb_mgmd` で構成キャッシュを使用する場合、このオプションは無視されます。

NDB Cluster のグローバル構成ファイルは INI 形式を使用します。INI 形式は、セクション見出し(角括弧で囲まれた部分)のあとに適切なパラメータ名と値が続くセクションで構成されます。標準 INI 形式との相違点の 1 つは、パラメータ名と値をコロン(:)と等号(=)で区切ることができることですが、等号をお勧めします。もう 1 つの違いは、セクションがセクション名で一意的に識別されないことです。代わりに、一意のセクション(同じタイプの 2 つの異なるノードなど)はセクション内のパラメータとして指定された一意の ID で識別されます。

デフォルト値は、ほとんどのパラメータに定義されており、`config.ini` で指定することもできます。デフォルト値のセクションを作成するには、セクション名に `default` という単語を追加します。たとえば、`[ndbd]` セクションには特定のデータノードに適用されるパラメータが含まれていますが、`[ndbd default]` セクションにはすべてのデータノードに適用されるパラメータが含まれています。すべてのデータノードが同じデータメモリーサイズを使用するとします。これらすべてを構成するには、データメモリーサイズを指定する `DataMemory` 行を含む `[ndbd default]` セクションを作成します。

使用する場合は、構成ファイルの[ndbd]セクションの前に[ndbd default]セクションを配置する必要があります。これは、他のタイプの default セクションにも当てはまります。

注記

NDB Cluster の一部の旧リリースでは、NoOfReplicas のデフォルト値はありませんでした。これは常に[ndbd default]セクションで明示的に指定する必要がありました。現在、このパラメータのデフォルト値はほとんどの一般的な使用シナリオでの推奨設定である 2 になっていますが、今後もこのパラメータを明示的に設定することが推奨されます。

グローバル構成ファイルでは、クラスタに關与するコンピュータとノード、およびノードをどのコンピュータに配置するかを定義する必要があります。1つの管理サーバー、2つのデータノード、および2つのMySQLサーバーで構成されるクラスタ用の簡単な構成ファイルの例をここに示します。

```
# file "config.ini" - 2 data nodes and 2 SQL nodes
# This file is placed in the startup directory of ndb_mgmd (the
# management server)
# The first MySQL Server can be started from any host. The second
# can be started only on the host mysqld_5.mysql.com

[ndbd default]
NoOfReplicas= 2
DataDir= /var/lib/mysql-cluster

[ndb_mgmd]
Hostname= ndb_mgmd.mysql.com
DataDir= /var/lib/mysql-cluster

[ndbd]
HostName= ndbd_2.mysql.com

[ndbd]
HostName= ndbd_3.mysql.com

[mysqld]
[mysqld]
HostName= mysqld_5.mysql.com
```

注記

前述の例は NDB Cluster に慣れるための最小限の開始構成として意図されており、本番設定では十分ではありません。完全な初期構成例については、[セクション23.3.3.2「NDB Cluster の推奨開始構成」](#)を参照してください。

config.ini ファイルには、各ノードに固有のセクションがあります。たとえば、このクラスタには2つのデータノードがあるため、前に示した構成ファイルにはこれらのノードを定義する2つの [ndbd] セクションがあります。

注記

config.ini ファイルでは、セクション見出しと同じ行にコメントを配置しないでください。これを行うと、管理サーバーはこのような構成ファイルを解析できないため、起動しなくなります。

config.ini ファイルのセクション

次のリストで説明するように、config.ini 構成ファイルでは6つの異なるセクションを使用できます。

- **[computer]:** クラスタホストを定義します。これは、実行可能な NDB Cluster を構成するために必要ではありませんが、大きなクラスタを設定するときに便利に使用できます。詳細は、[セクション23.3.3.4「NDB Cluster でのコンピュータの定義」](#)を参照してください。
- **[ndbd]:** クラスタデータノード (ndbd プロセス) を定義します。詳細は、[セクション23.3.3.6「NDB Cluster データノードの定義」](#)を参照してください。
- **[mysqld]:** クラスタのMySQLサーバーノード (SQL ノードや API ノードとも呼ばれる) を定義します。SQL ノードの構成については、[セクション23.3.3.7「NDB Cluster での SQL およびその他の API ノードの定義」](#)を参照してください。

- `[mgm]` または `[ndb_mgmd]`: クラスタの管理サーバー (MGM) ノードを定義します。管理ノードの構成については、[セクション23.3.3.5「NDB Cluster 管理サーバーの定義」](#)を参照してください。
- `[tcp]`: TCP/IP がデフォルトのトランスポートプロトコルであるクラスタノード間の TCP/IP 接続を定義します。通常、NDB Cluster はこれを自動的に処理するため、`[tcp]` または `[tcp default]` セクションを設定する必要はありませんが、状況によっては、クラスタによって提供されるデフォルトをオーバーライドする必要があります。使用可能な TCP/IP 構成パラメータと使用方法については、[セクション23.3.3.10「NDB Cluster TCP/IP 接続」](#)を参照してください。(場合によっては、[セクション23.3.3.11「直接接続を使用した NDB Cluster TCP/IP 接続」](#)も参考になる可能性があります。)
- `[shm]`: ノード間の共有メモリー接続を定義します。これは、MySQL 8.0 ではデフォルトで有効になっていますが、まだ実験的なものだと考えるようにしてください。SHM 相互接続については、[セクション23.3.3.12「NDB Cluster の共有メモリー接続」](#)を参照してください。
- `[sci]`: クラスタデータノード間のスケラブルコヒーラントインタフェース接続を定義します。NDB 8.0 ではサポートされていません。

セクションごとに `default` 値を定義できます。使用する場合、`default` セクションはそのタイプの他のセクションの前に配置する必要があります。たとえば、`[ndbd default]` セクションは、構成ファイルの `[ndbd]` セクションの前に表示される必要があります。

NDB Cluster パラメータ名は、MySQL Server `my.cnf` または `my.ini` ファイルで指定されていないかぎり、大文字と小文字が区別されません。

23.3.3.2 NDB Cluster の推奨開始構成

NDB Cluster から最高のパフォーマンスを得るには、次のような多くの要因があります:

- NDB Cluster ソフトウェアバージョン
- データノードおよび SQL ノードの数
- ハードウェア
- オペレーティングシステム
- 格納されるデータの量
- クラスタ稼働時の負荷のサイズとタイプ

したがって、最適な構成の取得は反復プロセスである可能性が高く、その結果は NDB Cluster デプロイメントごとに大きく異なる可能性があります。構成の変更は、クラスタが実行されているプラットフォーム、または NDB Cluster のデータを使用するアプリケーションで変更が行われたときにも示される可能性があります。これらの理由により、すべての使用シナリオに適した単一の構成を提示することは不可能です。ただし、このセクションでは推奨されるベース構成を示します。

初期の `config.ini` ファイル。 NDB Cluster 8.0 を実行しているクラスタを構成するには、次の `config.ini` ファイルを開始することをお勧めします:

```
# TCP PARAMETERS

[tcp default]
SendBufferMemory=2M
ReceiveBufferMemory=2M

# Increasing the sizes of these 2 buffers beyond the default values
# helps prevent bottlenecks due to slow disk I/O.

# MANAGEMENT NODE PARAMETERS

[ndb_mgmd default]
DataDir=path/to/management/server/data/directory

# It is possible to use a different data directory for each management
# server, but for ease of administration it is preferable to be
# consistent.

[ndb_mgmd]
HostName=management-server-A-hostname
```

```
# Nodeld=management-server-A-nodeid

[ndb_mgmd]
HostName=management-server-B-hostname
# Nodeld=management-server-B-nodeid

# Using 2 management servers helps guarantee that there is always an
# arbitrator in the event of network partitioning, and so is
# recommended for high availability. Each management server must be
# identified by a HostName. You may for the sake of convenience specify
# a Nodeld for any management server, although one is allocated
# for it automatically; if you do so, it must be in the range 1-255
# inclusive and must be unique among all IDs specified for cluster
# nodes.

# DATA NODE PARAMETERS

[ndbd default]
NoOfReplicas=2

# Using two fragment replicas is recommended to guarantee availability of data;
# using only one fragment replica does not provide any redundancy, which means
# that the failure of a single data node causes the entire cluster to
# shut down. As of NDB 8.0.19, it is also possible (but not required) to
# use more than two fragment replicas, although two fragment replicas are
# sufficient to provide high availability.

LockPagesInMainMemory=1

# On Linux and Solaris systems, setting this parameter locks data node
# processes into memory. Doing so prevents them from swapping to disk,
# which can severely degrade cluster performance.

DataMemory=3456M

# The value provided for DataMemory assumes 4 GB RAM
# per data node. However, for best results, you should first calculate
# the memory that would be used based on the data you actually plan to
# store (you may find the ndb\_size.pl utility helpful in estimating
# this), then allow an extra 20% over the calculated values. Naturally,
# you should ensure that each data node host has at least as much
# physical memory as the sum of these two values.

# ODirect=1

# Enabling this parameter causes NDBCLUSTER to try using O_DIRECT
# writes for local checkpoints and redo logs; this can reduce load on
# CPUs. We recommend doing so when using NDB Cluster on systems running
# Linux kernel 2.6 or later.

NoOfFragmentLogFiles=300
DataDir=path/to/data/node/data/directory
MaxNoOfConcurrentOperations=100000

SchedulerSpinTimer=400
SchedulerExecutionTimer=100
RealTimeScheduler=1
# Setting these parameters allows you to take advantage of real-time scheduling
# of NDB threads to achieve increased throughput when using ndbd. They
# are not needed when using ndbmtid; in particular, you should not set
# RealTimeScheduler for ndbmtid data nodes.

TimeBetweenGlobalCheckpoints=1000
TimeBetweenEpochs=200
RedoBuffer=32M

# CompressedLCP=1
# CompressedBackup=1
# Enabling CompressedLCP and CompressedBackup causes, respectively, local
# checkpoint files and backup files to be compressed, which can result in a space
# savings of up to 50% over noncompressed LCPs and backups.

# MaxNoOfLocalScans=64
MaxNoOfTables=1024
MaxNoOfOrderedIndexes=256
```

```
[ndbd]
HostName=data-node-A-hostname
# Nodeld=data-node-A-nodeid

LockExecuteThreadToCPU=1
LockMaintThreadsToCPU=0
# On systems with multiple CPUs, these parameters can be used to lock NDBCLUSTER
# threads to specific CPUs

[ndbd]
HostName=data-node-B-hostname
# Nodeld=data-node-B-nodeid

LockExecuteThreadToCPU=1
LockMaintThreadsToCPU=0

# You must have an [ndbd] section for every data node in the cluster;
# each of these sections must include a HostName. Each section may
# optionally include a Nodeld for convenience, but in most cases, it is
# sufficient to allow the cluster to allocate node IDs dynamically. If
# you do specify the node ID for a data node, it must be in the range 1
# to 144 inclusive and must be unique among all IDs specified for
# cluster nodes. (Previous to NDB 8.0.18, this range was 1 to 48 inclusive.)

# SQL NODE / API NODE PARAMETERS

[mysqld]
# HostName=sql-node-A-hostname
# Nodeld=sql-node-A-nodeid

[mysqld]

[mysqld]

# Each API or SQL node that connects to the cluster requires a [mysqld]
# or [api] section of its own. Each such section defines a connection
# 「slot」; you should have at least as many of these sections in the
# config.ini file as the total number of API nodes and SQL nodes that
# you wish to have connected to the cluster at any given time. There is
# no performance or other penalty for having extra slots available in
# case you find later that you want or need more API or SQL nodes to
# connect to the cluster at the same time.
# If no HostName is specified for a given [mysqld] or [api] section,
# then any API or SQL node may use that slot to connect to the
# cluster. You may wish to use an explicit HostName for one connection slot
# to guarantee that an API or SQL node from that host can always
# connect to the cluster. If you wish to prevent API or SQL nodes from
# connecting from other than a desired host or hosts, then use a
# HostName for every [mysqld] or [api] section in the config.ini file.
# You can if you wish define a node ID (Nodeld parameter) for any API or
# SQL node, but this is not necessary; if you do so, it must be in the
# range 1 to 255 inclusive and must be unique among all IDs specified
# for cluster nodes.
```

推奨される SQL ノードの my.cnf オプション。 NDB Cluster SQL ノードとして機能する MySQL Servers は、常に `--ndbcluster` および `--ndb-connectstring` オプションを使用して、コマンド行または `my.cnf` で起動する必要があります。さらに、セットアップでほかのオプションを必要とする場合を除き、クラスタ内のすべての `mysqld` プロセスに次のオプションを設定してください。

- `--ndb-use-exact-count=0`
- `--ndb-index-stat-enable=0`
- `--ndb-force-send=1`
- `--optimizer-switch=engine_condition_pushdown=on`

23.3.3.3 NDB Cluster 接続文字列

NDB Cluster 管理サーバー (`ndb_mgmd`) を除き、NDB Cluster の一部である各ノードには、管理サーバーの場所を指す接続文字列が必要です。この接続文字列は、管理サーバーとの接続を確立するときに使用されるほか、クラスタ

内でのノードの役割によっては、ほかのタスクを実行するときにも使用されます。接続文字列の構文は次のとおりです。

```
[nodeid=node_id, ]host-definition[, host-definition[, ...]]
```

```
host-definition:
  host_name[:port_number]
```

`node_id` は、`config.ini` 内のノードを識別する 1 以上の整数です。`host_name` は、有効なインターネット名または IP アドレスを表す文字列です。`port_number` は、TCP/IP ポート番号を参照する整数です。

```
example 1 (long): "nodeid=2,myhost1:1100,myhost2:1100,198.51.100.3:1200"
```

```
example 2 (short): "myhost1"
```

何も指定されなかった場合は、デフォルトの接続文字列値として `localhost:1186` が使用されます。接続文字列から `port_num` が省略された場合、デフォルトポートは 1186 です。このポートは、この目的のために IANA によって割り当てられたものであるため、ネットワーク上で常に使用可能です (詳細は、<http://www.iana.org/assignments/port-numbers> を参照してください)。

複数のホスト定義を列挙すると、複数の冗長管理サーバーを指定できます。NDB Cluster データまたは API ノードは、接続が正常に確立されるまで、指定された順序で各ホスト上の連続した管理サーバーへの接続を試みます。

接続文字列には、管理サーバーに接続する複数のネットワークインタフェースを持つノードで使用される 1 つ以上のバインドアドレスを指定することもできます。バインドアドレスは、ホスト名またはネットワークアドレスと、オプションのポート番号で構成されます。この拡張された接続文字列の構文をここに示します。

```
[nodeid=node_id, ]
[bind-address=host-definition, ]
host-definition[: bind-address=host-definition]
host-definition[: bind-address=host-definition]
[ ...]
```

```
host-definition:
  host_name[:port_number]
```

接続文字列で管理ホストを指定する前に 1 つのバインドアドレスを使用すると、そのアドレスは管理ホストのいずれかに接続するためのデフォルトとして使用されます (特定の管理サーバーにオーバーライドされた場合を除きます。例については、このセクションで後述します)。たとえば、次の接続文字列によって、このノードは接続先の管理サーバーに関係なく `198.51.100.242` を使用します。

```
bind-address=198.51.100.242, poseidon:1186, perch:1186
```

管理ホストの定義のあとにバインドアドレスを指定すると、そのアドレスはその管理ノードへの接続でのみ使用されます。次の接続文字列について考えます。

```
poseidon:1186;bind-address=localhost, perch:1186;bind-address=198.51.100.242
```

この場合、ノードは `poseidon` という名前のホストで実行されている管理サーバーに接続するために `localhost` を使用し、`perch` という名前のホストで実行されている管理サーバーに接続するために `198.51.100.242` を使用します。

デフォルトのバインドアドレスを指定してから、1 台以上の管理ホストにこのデフォルトをオーバーライドできます。次の例では、ホスト `poseidon` で実行されている管理サーバーに接続するために `localhost` が使用されます。`198.51.100.242` は、最初に (どの管理サーバーの定義よりも前に) 指定されているため、デフォルトのバインドアドレスであり、ホスト `perch` および `orca` 上の管理サーバーに接続するために使用されます。

```
bind-address=198.51.100.242,poseidon:1186;bind-address=localhost,perch:1186,orca:2200
```

接続文字列を指定するには、いくつかの異なる方法があります。

- 各実行可能ファイルには、起動時に管理サーバーを指定できる固有のコマンド行オプションがあります。(各実行可能ファイルのドキュメントを参照してください。)
- 管理サーバーの `my.cnf` ファイルの `[mysql_cluster]` セクションに接続文字列を配置することで、クラスタ内のすべてのノードの接続文字列を同時に設定することもできます。
- 下位互換性のため、同じ構文を使用するオプションがほかに 2 つ用意されています。
 - `NDB_CONNECTSTRING` 環境変数を設定して、接続文字列を含めます。

2. 各実行可能ファイル用の接続文字列を `Ndb.cfg` という名前のテキストファイルに記述し、このファイルを実行可能ファイルの起動ディレクトリに配置します。

ただし、これらは現在非推奨であり、新しいインストールに使用しないでください。

推奨される接続文字列の指定方法は、各実行可能ファイルのコマンド行または `my.cnf` ファイルでの設定です。

23.3.3.4 NDB Cluster でのコンピュータの定義

`[computer]` セクションは、システム内の各ノードのホスト名を定義しないで済む方法を提供する以外は、基本的には重要ではありません。ここに示すすべてのパラメータは必須です。

- `Id`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	string
デフォルト	[...]
範囲	...
再起動タイプ	IS (NDB 8.0.13)

これは、構成ファイルに含まれるホストコンピュータを参照するために使用する一意の識別子です。

重要

コンピュータ ID は、管理、API、またはデータノードに使用するノード ID とは異なります。ノード ID の場合と異なり、`config.ini` ファイルの `[computer]` セクションでは、`Id` の代わりに `Nodeld` を使用できません。

- `HostName`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	name or IP address
デフォルト	[...]
範囲	...
再起動タイプ	N (NDB 8.0.13)

これは、コンピュータのホスト名または IP アドレスです。

再起動のタイプ。このセクションのパラメータの説明で使用される再起動タイプに関する情報を次のテーブルに示します:

表 23.8 NDB Cluster の再起動タイプ

シンボル	再起動タイプ	説明
N	ノード	パラメータはローリング再起動を使用して更新できます (セクション 23.5.5 「NDB Cluster のローリング再起動の実行」 を参照)
S	システム	このパラメータの変更を有効にするには、すべてのクラスタノードを完全に停止してから再起動する必要があります
I	Initial	<code>--initial</code> オプションを使用してデータノードを再起動する必要があります

23.3.3.5 NDB Cluster 管理サーバーの定義

[[ndb_mgmd](#)] セクションは、管理サーバーの動作を構成するために使用されます。複数の管理サーバーが採用されている場合は、それらのすべてに共通するパラメータを [[ndb_mgmd default](#)] セクションに指定できます。[[mgm](#)] と [[mgm default](#)] はこれらの古いエイリアスで、下位互換性に対応しています。

次のリストに示すパラメータはすべてオプションであり、省略した場合はデフォルト値が使用されます。

注記

[ExecuteOnComputer](#) も [HostName](#) パラメータも存在しない場合は、両方にデフォルト値の [localhost](#) が使用されます。

• [Id](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	unsigned
デフォルト	[...]
範囲	1 - 255
再起動タイプ	IS (NDB 8.0.13)

クラスタ内の各ノードは一意の ID を持っています。管理ノードの場合、これは 1-255 の (これらを含む) 範囲の整数値で表されます。この ID は、ノードをアドレス指定するためにすべての内部クラスタメッセージで使用されるため、ノードのタイプに関係なく NDB Cluster ノードごとに一意である必要があります。

注記

データノードの ID は、145 未満にする必要があります。多数のデータノードを配備する予定の場合は、管理ノード (および API ノード) のノード ID を 144 より大きい値に制限することをお勧めします。(NDB 8.0.17 以前では、データノード ID の最大値は 48 でした。)

[Id](#) パラメータを使った管理ノードの識別は、[NodeId](#) を優先するため非推奨になりました。[Id](#) は下位互換性のために引き続きサポートされていますが、警告が生成されるようになり、NDB Cluster の将来のバージョンで削除される可能性があります。

• [NodeId](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	unsigned
デフォルト	[...]
範囲	1 - 255
再起動タイプ	IS (NDB 8.0.13)

クラスタ内の各ノードは一意の ID を持っています。管理ノードでは、これは 1-255 の (これらを含む) 範囲の整数値で表されます。この ID は、ノードをアドレス指定するためにすべての内部クラスタメッセージで使用されるため、ノードのタイプに関係なく NDB Cluster ノードごとに一意である必要があります。

注記

NDB 8.0.18 の時点では、データノード ID は 145 未満である必要があります。(以前は 49 未満でした。) 多数のデータノードを配備する予定の場合は、管理ノード (および API ノード) のノード ID を 144 より大きい値に制限することをお勧めします。

[NodeId](#) は、管理ノードを識別時の使用が推奨されるパラメータ名です。古い [Id](#) は下位互換性のために引き続きサポートされますが、非推奨になり、使用時に警告が生成されるようになりました。将来の NDB Cluster リリースでも削除される可能性があります。

• [ExecuteOnComputer](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	name
デフォルト	[...]
範囲	...
非推奨	Yes (in NDB 7.5)
再起動タイプ	S (NDB 8.0.13)

これは、`config.ini` ファイルの `[computer]` セクションに定義されたいずれかのコンピュータに設定されている `Id` を参照します。

重要

このパラメータは非推奨であり、将来のリリースで削除される予定です。かわりに `HostName` パラメータを使用してください。

• PortNumber

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	unsigned
デフォルト	1186
範囲	0 - 64K
再起動タイプ	S (NDB 8.0.13)

これは、管理サーバーが構成要求と管理コマンドを待機するポートの番号です。

•

このノードのノード ID は、明示的にリクエストする接続にのみ指定できます。ノード ID をリクエストする管理サーバーは、このノード ID を使用できません。このパラメータは、同じホストで複数の管理サーバーを実行していて、`HostName` でプロセスを区別するのに十分でない場合に使用できます。テストで使用するためのものです。

• HostName

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	name or IP address
デフォルト	[...]
範囲	...
再起動タイプ	N (NDB 8.0.13)

このパラメータを指定すると、管理ノードを配置するコンピュータのホスト名が定義されます。`localhost` 以外のホスト名を指定するには、このパラメータまたは `ExecuteOnComputer` のどちらかが必要です。

• LocationDomainId

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	0
範囲	0 - 16
再起動タイプ	

クラウド内の特定の「可用性ドメイン」(可用性ゾーンとも呼ばれる)に管理ノードを割り当てます。どのノードがどの可用性ドメインにあるかを `NDB` に通知することで、次の方法でクラウド環境のパフォーマンスを向上させることができます:

- リクエストされたデータが同じノードで見つからない場合、読取りは同じ可用性ドメイン内の別のノードに転送できます。
- 異なる可用性ドメイン内のノード間の通信では、それ以上の手動操作なしで **NDB** トランスポータ WAN サポートを使用することが保証されています。
- トランスポータグループ番号は、SQL および他の API ノードが可能な場合は常に同じ可用性ドメイン内のローカルデータノードと通信するように、使用される可用性ドメインに基づくことができます。
- アービトラータは、データノードが存在しない可用性ドメインから選択することも、そのような可用性ドメインが見つからない場合は 3 番目の可用性ドメインから選択することもできます。

LocationDomainId は 0 以上 16 以下の整数値を取り、0 がデフォルトです。0 を使用することは、パラメータを未設定のままにすることと同じです。

- **LogDestination**

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	{CONSOLE SYSLOG FILE}
デフォルト	FILE: filename=ndb_nodeid_cluster.log, maxsize=1000000, maxfiles=6
範囲	...

再起動タイプ

N (NDB 8.0.13)

このパラメータは、クラスタのロギング情報の送信先を指定します。これには `CONSOLE`、`SYSLOG`、`FILE` の 3 つのオプションがあり、`FILE` がデフォルトです。

- `CONSOLE` では、`stdout` にログが出力されます。

CONSOLE

- `SYSLOG` では、`syslog` 機能にログが送信されます。指定できる値は、`auth`、`authpriv`、`cron`、`daemon`、`ftp`、`kern`、`lpr`、`mail`、`news`、`syslog`、`user`、`uucp`、`local0`、`local1`、`local2`、`local3`、`local4` のいずれかです。

注記

すべてのオペレーティングシステムで必ずしもすべての機能がサポートされるとはかぎりません。

SYSLOG:facility=syslog

- `FILE` では、クラスタのログ出力が同じマシン上の通常のファイルにパイプされます。次の値を指定できます。

- `filename`: ログファイルの名前。

このような場合に使用されるデフォルトのログファイル名は `ndb_nodeid_cluster.log` です。

- `maxsize`: ロギングが新しいファイルにロールオーバーされる前のファイルの最大サイズ (バイト単位)。これが発生すると、古いログファイルの名前が変更され、ファイル名の末尾に `.N` が追加されます (`N` はこの名前に対してまだ使用されていない次の番号です)。
- `maxfiles`: ログファイルの最大数。

FILE:filename=cluster.log,maxsize=1000000,maxfiles=6

`FILE` パラメータのデフォルト値は、`FILE:filename=ndb_node_id_cluster.log,maxsize=1000000,maxfiles=6` です (`node_id` はノードの ID です)。

ここに示すように、複数のログの保存先をセミコロンで区切って指定できます。

CONSOLE;SYSLOG:facility=local0;FILE:filename=/var/log/mgmd

- `ArbitrationRank`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	0-2
デフォルト	1
範囲	0 - 2
再起動タイプ	N (NDB 8.0.13)

このパラメータは、アービトレータとして機能するノードを定義するために使用されます。アービトレータにできるのは、管理ノードと SQL ノードだけです。 `ArbitrationRank` には次のいずれかの値を指定できます。

- `0`: ノードがアービトレータとして使用されることはありません。
- `1`: ノードは高い優先度を持ちます。つまり、優先度の低いノードよりアービトレータとして優先されます。

- 2: 優先度の高いノードがその目的に使用できない場合にのみアービトラータとして使用される低優先度ノードを示します。

通常は、管理サーバーの `ArbitrationRank` を 1 (管理ノードのデフォルト) に設定し、すべての SQL ノードを 0 (SQL ノードのデフォルト) に設定することにより、管理サーバーをアービトラータとして構成してください。

すべての管理および SQL ノードで `ArbitrationRank` を 0 に設定するか、`config.ini` グローバル構成ファイルの `[ndbd default]` セクションに `Arbitration` パラメータを設定することで、アービトレーションを完全に無効化できます。`Arbitration` を設定すると、`ArbitrationRank` の設定はすべて無視されます。

- `ArbitrationDelay`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	milliseconds
デフォルト	0
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

アービトレーション要求に対する管理サーバーの応答をミリ秒数だけ遅らせるための整数値。デフォルトでは、この値は 0 です。通常、これを変更する必要はありません。

- `DataDir`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	path
デフォルト	.
範囲	...
再起動タイプ	N (NDB 8.0.13)

これは、管理サーバーからの出力ファイルが配置されるディレクトリを指定します。これらのファイルには、クラスタログファイル、プロセス出力ファイル、およびデーモンプロセス ID (PID) ファイルが含まれます。(ログファイルでは、`LogDestination`、の `FILE` パラメータをこのセクションで前述したように設定することで、この場所をオーバーライドできます。)

このパラメータのデフォルト値は、`ndb_mgmd` が配置されているディレクトリです。

- `PortNumberStats`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	unsigned
デフォルト	[...]
範囲	0 - 64K
再起動タイプ	N (NDB 8.0.13)

このパラメータは、NDB Cluster 管理サーバーから統計情報を取得するために使用されるポート番号を指定します。デフォルト値はありません。

- `Wan`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	boolean
デフォルト	false
範囲	true, false

再起動タイプ	N (NDB 8.0.13)
--------	----------------

WAN の TCP 設定をデフォルトとして使用します。

- [HeartbeatThreadPriority](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	string
デフォルト	[...]
範囲	...
再起動タイプ	

管理および API ノードのハートビートスレッドのスケジューリングポリシーと優先度を設定します。

このパラメータを設定するための構文をここに示します。

```
HeartbeatThreadPriority = policy[, priority]
```

```
policy:
{FIFO | RR}
```

このパラメータを設定するときは、ポリシーを指定する必要があります。これは、[FIFO](#) (先入れ先出し) または [RR](#) (ラウンドロビン) のいずれかです。オプションで、ポリシー値のあとに優先度 (整数) を指定できます。

- [ExtraSendBufferMemory](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	bytes
デフォルト	0
範囲	0 - 32G
再起動タイプ	N (NDB 8.0.13)

このパラメータは、[TotalSendBufferMemory](#)、[SendBufferMemory](#)、またはその両方を使用して設定されたメモリーに加えて割り当てられるトランスポート送信バッファメモリーの量を指定します。

- [TotalSendBufferMemory](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	bytes
デフォルト	0
範囲	256K - 4294967039 (0xFFFFFEFF)
再起動タイプ	N (NDB 8.0.13)

このパラメータは、すべての構成済みトランスポート間で共有される送信バッファメモリーの、このノードに割り当てられるメモリー合計量を決定するために使用されます。

このパラメータが設定されている場合、許可される最小値は 256KB です。0 はパラメータが設定されていないことを示します。詳細は、[セクション 23.3.3.14 「NDB Cluster 送信バッファパラメータの構成」](#) を参照してください。

- [HeartbeatIntervalMgmdMgmd](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	milliseconds
デフォルト	1500

範囲	100 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

別の管理ノードがこの管理ノードに接続しているかどうかを判定するために使用されるハートビートメッセージの間隔を指定します。管理ノードは、この間隔を 3 回待ったあとで接続の切断を宣言します。このため、デフォルト設定の 1500 ミリ秒では、管理ノードはおよそ 1600 ミリ秒待ってからタイムアウトします。

注記

管理ノードの構成を変更したあとは、新しい構成を有効にするためにクラスタのローリング再起動を実行する必要があります。

実行中の NDB Cluster に新しい管理サーバーを追加するには、既存の `config.ini` ファイルを変更したあとに、すべてのクラスタノードのローリング再起動も実行する必要があります。複数の管理ノードを使用するときに発生する問題の詳細は、[セクション 23.1.7.10「複数の NDB Cluster ノードに関する制限事項」](#) を参照してください。

再起動のタイプ。このセクションのパラメータの説明で 사용되는再起動タイプに関する情報を次のテーブルに示します:

表 23.9 NDB Cluster の再起動タイプ

シンボル	再起動タイプ	説明
N	ノード	パラメータはローリング再起動を使用して更新できます (セクション 23.5.5「NDB Cluster のローリング再起動の実行」 を参照)
S	システム	このパラメータの変更を有効にするには、すべてのクラスタノードを完全に停止してから再起動する必要があります
I	Initial	<code>--initial</code> オプションを使用してデータノードを再起動する必要があります

23.3.3.6 NDB Cluster データノードの定義

クラスタのデータノードの動作を構成するには、`[ndbd]` および `[ndbd default]` セクションを使用します。

データノードのプロセスとして `ndbd` バイナリと `ndbmt` バイナリのどちらを使用する場合でも、セクション名としては常に `[ndbd]` と `[ndbd default]` が使用されます。

バッファサイズ、プールサイズ、タイムアウトなどを制御する数多くのパラメータがあります。唯一の必須パラメータは、`ExecuteOnComputer` または `HostName` のいずれかです。これは、ローカルの `[ndbd]` セクションで定義する必要があります。

`NoOfReplicas` パラメータは、すべてのクラスタデータノードに共通であるため、`[ndbd default]` セクションに定義してください。`NoOfReplicas` を設定する必要は必ずしもありませんが、これを明示的に設定することをお勧めします。

ほとんどのデータノードパラメータは `[ndbd default]` セクションに設定します。`[ndbd]` セクションでの変更が許可されるのは、ローカル値の設定を明示的に規定されているパラメータだけです。`HostName`、`NodeId`、および `ExecuteOnComputer` (存在する場合) は、`config.ini` のほかのセクションではなく、ローカルの `[ndbd]` で定義する必要があります。つまり、これらのパラメータの設定は 1 つのデータノードに固有のものであります。

メモリー使用状況やバッファサイズに影響を与えるパラメータでは、1024、1024 * 1024、または 1024 * 1024 * 1024 の単位を示すサフィクスとして `K`、`M`、または `G` を使用できます。(たとえば、`100K` は `100 * 1024 = 102400` を意味します。)

MySQL Server `my.cnf` または `my.ini` ファイルで使用されている場合を除き、パラメータ名および値では大/小文字が区別されません。

「NDB Cluster ディスクデータ」テーブルに固有の構成パラメータの詳細は、このセクション ([ディスクデータの構成パラメータ](#) を参照) で後述します。

これらのパラメータはすべて、`ndbmt` (`ndbd` のマルチスレッドバージョン) にも適用されます。追加の 3 つのデータノード構成パラメータ (`MaxNoOfExecutionThreads`、`ThreadConfig`、および `NoOfFragmentLogParts`) は、`ndbmt` にも適用されます。これらを `ndbd` に使用しても、無効になります。詳細は、[マルチスレッドの構成パラメータ \(ndbmt\)](#) を参照してください。 [セクション 23.4.3 「ndbmt — NDB Cluster データノードデーモン \(マルチスレッド\)」](#) も参照してください。

データノードの識別。 `Nodeld` または `Id` の値 (つまり、データノードの識別子) は、ノード起動時のコマンド行、または構成ファイルで割り当てることができます。

- `Nodeld`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	unsigned
デフォルト	[...]
範囲	1 - 48
バージョン (またはそれ以降)	NDB 8.0.18
タイプまたは単位	unsigned
デフォルト	[...]
範囲	1 - 144
再起動タイプ	IS (NDB 8.0.13)

一意のノード ID は、クラスタのすべての内部メッセージでノードのアドレスとして使用されます。データノードでは、これは 1-144 の (これらを含む) 範囲の整数です。 (NDB 8.0.17 以前では、これは 1 から 48 でした。) クラスタ内の各ノードは、一意の識別子を持っている必要があります。

`Nodeld` は、データノードを識別するときに使用できる唯一のパラメータ名です。

- `ExecuteOnComputer`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	name
デフォルト	[...]
範囲	...
非推奨	Yes (in NDB 7.5)
再起動タイプ	S (NDB 8.0.13)

これは、[\[computer\]](#) セクションに定義されたいずれかのコンピュータに設定されている `Id` を参照します。

重要

このパラメータは非推奨であり、将来のリリースで削除される予定です。かわりに [HostName](#) パラメータを使用してください。

このノードのノード ID は、明示的にリクエストする接続にのみ指定できます。ノード ID をリクエストする管理サーバーは、このノード ID を使用できません。このパラメータは、同じホストで複数の管理サーバーを実行していて、[HostName](#) でプロセスを区別するのに十分でない場合に使用できます。テストで使用するためのものです。

- `HostName`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	name or IP address

デフォルト	localhost
範囲	...
再起動タイプ	N (NDB 8.0.13)

このパラメータを指定すると、データノードを配置するコンピュータのホスト名が定義されます。localhost 以外のホスト名を指定するには、このパラメータまたは [ExecuteOnComputer](#) のどちらかが必要です。

- [ServerPort](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	unsigned
デフォルト	[...]
範囲	1 - 64K
再起動タイプ	S (NDB 8.0.13)

クラスタ内の各ノードは、ほかのノードに接続するためにポートを使用します。デフォルトでは、このポートは同じホストコンピュータ上の 2 つのノードで同じポート番号を受け取らないよう動的に割り当てられるため、通常はこのパラメータの値を指定する必要はありません。

ただし、データノードと API ノード (SQL ノードを含む) 間の通信を許可するため、ファイアウォールで特定のポートを開く必要がある場合は、`config.ini` ファイルの `[ndbd]` セクションまたは (複数のデータノードに対してこれを行う必要がある場合は) `[ndbd default]` セクションでこのパラメータを必要なポートの番号に設定すると、SQL ノード、API ノード、またはその両方からの着信接続のためにその番号のポートを開くことができます。

注記

データノードから管理ノードへの接続は、`ndb_mgmd` 管理ポート (管理サーバー `PortNumber`) を使用して行われるため、任意のデータノードからそのポートへの送信接続は常に許可されます。

- [TcpBind_INADDR_ANY](#)

このパラメータを `TRUE` または `1` に設定すると、`IP_ADDR_ANY` がバインドされ、任意の場所から接続できるようになります (自動生成接続の場合)。デフォルトは `FALSE (0)` です。

- [NodeGroup](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	unsigned
デフォルト	[...]
範囲	0 - 65536
再起動タイプ	IS (NDB 8.0.13)

このパラメータを使用して、データノードを特定のノードグループに割り当てることができます。これはクラスタを最初に起動したときは読み取り専用であり、オンラインでデータノードを別のノードグループに再割り当てすることはできません。`config.ini` ファイルの `[ndbd default]` セクションでこのパラメータを使用することは、通常望ましくありません。また、ノードをノードグループに割り当てるときは、ノードグループに無効な数のノードが割り当てられないように注意する必要があります。

`NodeGroup` パラメータは主に、ローリング再起動を実行せずに、実行中の NDB Cluster に新しいノードグループを追加するために使用されます。そのためには、これを 65536 (最大値) に設定します。`NodeGroup` の値は、すべてのクラスタデータノードではなく、あとで起動され、新しいノードグループとしてクラスタに追加されるノードのみ設定する必要があります。詳細は、[セクション 23.5.7.3 「NDB Cluster データノードのオンラインでの追加: 詳細な例」](#) を参照してください。

- LocationDomainId

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	0
範囲	0 - 16
再起動タイプ	

クラウド内の特定の「**可用性ドメイン**」(可用性ゾーンとも呼ばれる)にデータノードを割り当てます。どのノードがどの可用性ドメインにあるかを NDB に通知することで、次の方法でクラウド環境のパフォーマンスを向上させることができます:

- リクエストされたデータが同じノードで見つからない場合、読み取りは同じ可用性ドメイン内の別のノードに転送できます。
- 異なる可用性ドメイン内のノード間の通信では、それ以上の手動操作なしで NDB トランスポータ WAN サポートを使用することが保証されています。
- トランスポータグループ番号は、SQL および他の API ノードが可能な場合は常に同じ可用性ドメイン内のローカルデータノードと通信するように、使用される可用性ドメインに基づくことができます。
- アービトラータは、データノードが存在しない可用性ドメインから選択することも、そのような可用性ドメインが見つからない場合は 3 番目の可用性ドメインから選択することもできます。

LocationDomainId は 0 以上 16 以下の整数値を取り、0 がデフォルトです。0 を使用することは、パラメータを未設定のままにすることと同じです。

- NoOfReplicas

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	2
範囲	1 - 2
バージョン (またはそれ以降)	NDB 8.0.19
タイプまたは単位	integer
デフォルト	2
範囲	1 - 4
再起動タイプ	IS (NDB 8.0.13)

このグローバルパラメータは、[[ndbd default](#)]セクションでのみ設定でき、クラスタに格納されている各テーブルのフラグメントレプリカ数を定義します。このパラメータは、ノードグループのサイズも指定します。ノードグループは、すべてが同じ情報を格納するノードのセットです。

ノードグループは暗黙的に形成されます。最初のノードグループはノード ID がもっとも小さいデータノードのセットで形成され、次のノードグループはノード ID が次に小さいデータノードのセットで形成されます (以下同様)。例として、4 つのデータノードがあり、NoOfReplicas が 2 に設定されているとします。4 つのデータノードのノード ID はそれぞれ 2、3、4、および 5 です。この場合、1 つ目のノードグループはノード 2 および 3 で形成され、2 つ目のノードグループはノード 4 および 5 で形成されます。1 つのハードウェアの障害によってクラスタ

全体が機能停止するため、同じノードグループ内のノードが同じコンピュータに配置されないようにクラスタを構成することが重要です。

ノード ID が指定されていない場合、データノードの順序はノードグループの決定要素になります。明示的な割当てが行われたかどうかにかかわらず、管理クライアントの `SHOW` コマンドの出力に表示できます。

`NoOfReplicas` のデフォルト値は 2 です。これはほとんどの本番環境で推奨される値であり、NDB 8.0.18 より前はサポートされていた最大値でした。NDB 8.0.19 以降では、このパラメータ値を 3 または 4 に設定すると、本番で完全にテストされ、サポートされます。

警告

`NoOfReplicas` を 1 に設定することは、すべてのクラスタデータのコピーが 1 つだけ存在することを意味します。この場合は、そのデータに格納されているデータの追加のコピーが存在しないため、1 つのデータノードの喪失によってクラスタが機能停止します。

クラスタ内のデータノードの数は、このパラメータの値で均等に割り切れる必要があります。たとえば、データノードが 2 つある場合、2/3 と 2/4 の両方で小数値が生成されるため、`NoOfReplicas` は 1 または 2 のいずれかである必要があります。4 つのデータノードがある場合、`NoOfReplicas` は 1、2 または 4 である必要があります。

• `DataDir`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	path
デフォルト	.
範囲	...
再起動タイプ	IN (NDB 8.0.13)

このパラメータは、トレースファイル、ログファイル、PID ファイル、およびエラーログが配置されるディレクトリを指定します。

デフォルトはデータノードプロセスの作業ディレクトリです。

• `FileSystemPath`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	path
デフォルト	<code>DataDir</code>
範囲	...
再起動タイプ	IN (NDB 8.0.13)

このパラメータは、メタデータ用に作成されるすべてのファイル、Redo ログ、Undo ログ (ディスクデータテーブル用)、およびデータファイルが配置されるディレクトリを指定します。デフォルトは、`DataDir` で指定されたディレクトリです。

注記

このディレクトリは、`ndbd` プロセスが開始される前に存在している必要があります。

NDB Cluster の推奨ディレクトリ階層には、ノードファイルシステムのディレクトリが作成される `/var/lib/mysql-cluster` が含まれています。このサブディレクトリの名前にはノード ID が含まれます。たとえば、ノード ID が 2 の場合、このサブディレクトリの名前は `ndb_2_fs` です。

• `BackupDataDir`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	path

デフォルト	FileSystemPath
範囲	...
再起動タイプ	IN (NDB 8.0.13)

このパラメータは、バックアップが配置されるディレクトリを指定します。

重要

この値の末尾には常に '/BACKUP' という文字列が追加されます。たとえば、BackupDataDir の値を /var/lib/cluster-data に設定すると、すべてのバックアップが /var/lib/cluster-data/BACKUP の下に格納されます。これは、事実上のデフォルトのバックアップ場所が FileSystemPath パラメータで指定された場所の下にある BACKUP という名前のディレクトリであることも意味しています。

データメモリー、インデックスメモリー、および文字列メモリー

DataMemory と IndexMemory は、実際のレコードとインデックスの格納に使用されるメモリーセグメントのサイズを指定する [ndbd] パラメータです。これらの値を設定する場合、DataMemory の使用方法を理解することが重要です。これは通常、クラスタによる実際の使用状況を反映するために更新する必要があるためです。

注記

IndexMemory は非推奨であり、NDB Cluster の将来のバージョンで削除される可能性があります。詳細は、次の説明を参照してください。

• DataMemory

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	bytes
デフォルト	98M
範囲	1M - 1T
バージョン (またはそれ以降)	NDB 8.0.19
タイプまたは単位	bytes
デフォルト	98M
範囲	1M - 16T
再起動タイプ	N (NDB 8.0.13)

このパラメータは、データベースレコードの格納に使用できるスペースの量 (バイト単位) を定義します。この値によって指定される量の全体がメモリー内に割り当てられるため、それに対応できるだけの十分な物理メモリーがマシンに搭載されていることが非常に重要です。

DataMemory によって割り当てられたメモリーは、実際のレコードとインデックスの両方を格納するために使用されます。各レコードには 16 バイトのオーバーヘッドがあります。各レコードは 128 バイトのページオーバーヘッドを含む 32K バイトのページに格納されるため、レコードごとに追加の量が発生します (次を参照してください)。また、各レコードが 1 つのページのみにも格納されるため、ページごとに少量の無駄が発生します。

可変サイズのテーブル属性については、DataMemory から割り当てられた別個のデータページにデータが格納されます。可変長レコードは、可変サイズ部分を参照するための 4 バイトの追加オーバーヘッドを含む固定サイズ部分を使用します。可変サイズ部分には、2 バイトのオーバーヘッドと属性あたり 2 バイトが含まれます。

NDB 8.0.18 の時点では、最大レコードサイズは 30000 バイトです。以前は 14000 バイトでした。

DataMemory に割り当てられたリソースは、すべてのデータおよびインデックスの格納に使用されます。(IndexMemory として構成されたメモリーは、DataMemory が共通リソースプールを形成するために使用するメモリーに自動的に追加されます。)

現在、NDB Cluster はパーティションあたりのハッシュインデックスに最大 512 MB を使用できます。つまり、`ndb_mgm -e "ALL REPORT MEMORYUSAGE"` がかなりの空き `DataMemory` を示していても、MySQL クライアントアプリケーションで「テーブルがいっぱいです」エラーが発生する可能性があります。このため、データの負荷が高いノードでデータノードを再起動するときに問題が発生する場合があります。

特定のテーブルのローカルデータマネージャごとのパーティション数を制御するには、テーブルの作成時に、LDM ごとにそれぞれ 1、2、3 または 4 つのパーティションに対して、`NDB_TABLE` オプション `PARTITION_BALANCE` を `FOR_RA_BY_LDM`、`FOR_RA_BY_LDM_X_2`、`FOR_RA_BY_LDM_X_3` または `FOR_RA_BY_LDM_X_4` のいずれかの値に設定します (セクション 13.1.20.11 「`NDB_TABLE` オプションの設定」 を参照)。

注記

以前のバージョンの「NDB Cluster」では、「NDB Cluster」テーブルに追加のパーティションを作成できるため、`CREATE TABLE` の `MAX_ROWS` オプションを使用してハッシュインデックスに使用可能なメモリーを増やすことができました。下位互換性のために引き続きサポートされていますが、この目的で `MAX_ROWS` を使用することは非推奨です。かわりに `PARTITION_BALANCE` を使用する必要があります。

`MinFreePct` 構成パラメータを使用して、ノード再起動の問題を回避することもできます。

`DataMemory` によって割り当てられるメモリースペースは 32K バイトのページで構成され、テーブルのフラグメントに割り当てられます。各テーブルは、通常、クラスタ内のデータノードと同じ数のフラグメントにパーティション化されます。このため、各ノードには `NoOfReplicas` での設定と同じ数のフラグメントがあります。

ページの割り当て後、テーブルを削除する以外の方法で空きページのプールに戻すことは、現在不可能です。(これは、特定のページに割り当てられた `DataMemory` ページをほかのテーブルで使用できないことも意味します。) データノードのリカバリを実行すると、すべてのレコードがほかの稼働しているノードから空のパーティションに挿入されるため、パーティションも圧縮されます。

`DataMemory` メモリー領域には UNDO 情報も含まれています。更新ごとに、変更されていないレコードのコピーが `DataMemory` に割り当てられます。順序付けされたテーブルインデックスにも各コピーへの参照が含まれています。一意のハッシュインデックスは、一意のインデックスカラムが更新されたときだけ更新されます。その場合は、コミット時にインデックステーブル内の新しいエントリが挿入され、古いエントリが削除されます。このため、クラスタを使用するアプリケーションによって実行される大規模なトランザクションを処理できるだけの十分なメモリーを割り当てる必要もあります。いずれにしても、次の理由から、少数の大規模なトランザクションを実行するよりも、多数の小規模なトランザクションを使用する方がメリットがあります。

- 大規模なトランザクションは小規模なトランザクションより高速に実行できません
- 大規模なトランザクションでは、トランザクション障害の発生時に消失して繰り返す必要がある操作の数が多くなります
- 大規模なトランザクションはより多くのメモリーを使用します

NDB 8.0 での `DataMemory` のデフォルト値は 98MB です。最小値は 1MB です。最大サイズはありませんが、実際には、制限に達したときにプロセスがスワップを開始しないように最大サイズを決定する必要があります。この制限は、マシン上の使用可能な物理 RAM 量と、オペレーティングシステムが 1 つのプロセスに振り向けることができるメモリー量によって決まります。32 ビットオペレーティングシステムでは通常プロセスあたり 2-4G バイトに制限されますが、64 ビットオペレーティングシステムではより多くのメモリーを使用できます。このため、大規模なデータベースでは 64 ビットオペレーティングシステムを使用することをお勧めします。

• `IndexMemory`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	bytes
デフォルト	0
範囲	1M - 1T
非推奨	Yes (in NDB 7.6)

再起動タイプ

N (NDB 8.0.13)

IndexMemory パラメータは非推奨になりました (将来の削除の対象となります)。IndexMemory に割り当てられたメモリーは、かわりに **DataMemory** と同じプールに割り当てられます。これは、データおよびインデックスをメモリーに格納するために必要なすべてのリソースのみを担当します。NDB 8.0 では、クラスタ構成ファイルで **IndexMemory** を使用すると、管理サーバーから警告がトリガーされます。

ハッシュインデックスのサイズは、この式を使用して見積もることができます。

```
size = ( (fragments * 32K) + (rows * 18) )
      * fragment_replicas
```

fragments はフラグメントの数、**fragment_replicas** はフラグメントレプリカの数 (通常は 2)、**rows** は行数です。テーブルに 100 万行、8 フラグメント、および 2 フラグメントレプリカがある場合、予想されるインデックスメモリー使用量は次のように計算されます:

```
((8 * 32K) + (1000000 * 18)) * 2 = ((8 * 32768) + (1000000 * 18)) * 2
= (262144 + 18000000) * 2
= 18262144 * 2 = 36524288 bytes = ~35MB
```

順序付けされたインデックスのインデックス統計 (有効な場合) は、**mysql.ndb_index_stat_sample** テーブルに格納されます。このテーブルにはハッシュインデックスがあるため、これによってインデックスのメモリー使用量が追加されます。特定の順序付けされたインデックスの行数の上限は、次のように計算できます。

```
sample_size= key_size + ((key_attributes + 1) * 4)

sample_rows = IndexStatSaveSize
              * ((0.01 * IndexStatSaveScale * log2(rows * sample_size)) + 1)
              / sample_size
```

前の式で、**key_size** は順序付けされたインデックスキーのサイズ (バイト単位)、**key_attributes** は順序付けされたインデックスキー内の属性の数、**rows** はベーステーブルの行数です。

テーブル **t1** に、100 万個の行と、2 つの 4 バイト整数に対する **ix1** という名前の順序付けされたインデックスがあるとします。また、**IndexStatSaveSize** と **IndexStatSaveScale** がこのデフォルト値 (それぞれ 32K および 100) に設定されているとします。前の 2 つの式を使用して、次のように計算できます。

```
sample_size = 8 + ((1 + 2) * 4) = 20 bytes

sample_rows = 32K
              * ((0.01 * 100 * log2(1000000*20)) + 1)
              / 20
              = 32768 * ((1 * ~16.811) + 1) / 20
              = 32768 * ~17.811 / 20
              = ~29182 rows
```

予想されるインデックスのメモリー使用量は $2 * 18 * 29182 =$ 約 1050550 バイトです。

NDB 8.0 では、このパラメータの最小値とデフォルト値は 0 (ゼロ) です。

- StringMemory

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	% or bytes
デフォルト	25
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	S (NDB 8.0.13)

このパラメータは、テーブル名などの文字列用に割り当てられるメモリー量を決定するもので、**config.ini** ファイルの **[ndbd]** または **[ndbd default]** セクションに指定されます。0 から 100 までの (これらを含む) 値は、デフォルトの最大値のパーセントとして解釈されます。デフォルトの最大値は、テーブルの数、テーブル名のサイズ、**.FRM**

ファイルの最大サイズ、[MaxNoOfTriggers](#)、カラム名の最大サイズ、およびデフォルトの最大カラム値を含むいくつかの係数に基づいて計算されます。

100 を超える値は、バイト数として解釈されます。

デフォルト値は 25 (つまり、デフォルトの最大の 25%) です。

ほとんどの場合、デフォルト値で十分ですが、多数の NDB テーブル (1000 以上) がある場合は、エラー 773 「文字列メモリー不足です。StringMemory 構成パラメータを変更してください: 永続エラー: スキーマエラー」が発生する可能性があります、その場合はこの値を増やす必要があります。25 (25%) は過剰ではないため、このエラーが最も極端な状況を除くすべての状況で繰り返されるのを防ぐ必要があります。

次の例は、1 つのテーブルでメモリーがどのように使用されるかを示しています。このテーブル定義について考えます。

```
CREATE TABLE example (
  a INT NOT NULL,
  b INT NOT NULL,
  c INT NOT NULL,
  PRIMARY KEY(a),
  UNIQUE(b)
) ENGINE=NDBCLUSTER;
```

各レコードに、12 バイトのデータと 12 バイトのオーバーヘッドがあります。NULL 可能カラムがない場合は、4 バイトのオーバーヘッドを節約できます。さらに、カラム [a](#) および [b](#) に対して 2 つの順序付けされたインデックスがあり、レコードごとにおよそ 10 バイトが消費されます。ベーステーブルには主キーのハッシュインデックスがあり、レコードあたりおよそ 29 バイトが使用されます。[b](#) を主キーとし、[a](#) をカラムとする別のテーブルによって一意の制約が実装されています。この別のテーブルでは、8 バイトのレコードデータと 12 バイトのオーバーヘッドに加えて、[example](#) テーブル内のレコードあたり 29 バイトのインデックスメモリーが追加で消費されます。

したがって、100 万個のレコードに対して、主キーと一意の制約のハッシュインデックスを処理するには 58M バイトのインデックスメモリーが必要です。さらに、ベーステーブルのレコード、一意のインデックステーブル、および 2 つの順序付けされたインデックステーブルに 64M バイトが必要です。

ハッシュインデックスがかなりのメモリー量を占有しますが、データへのアクセスは非常に高速になります。また、NDB Cluster で一意性制約を処理するためにも使用されます。

現在、唯一のパーティション化アルゴリズムはハッシュ化であり、順序付けされたインデックスは各ノードでローカルに使用されます。したがって、一般的なケースで順序付けされたインデックスを使用して一意制約を処理することはできません。

[IndexMemory](#) と [DataMemory](#) の両方で重要な点は、各ノードグループのすべてのデータメモリーとインデックスメモリーの合計がデータベースの合計サイズになることです。各ノードグループはレプリケートされた情報の格納に使用されるため、2 つのフラグメントレプリカを持つ 4 つのノードがある場合は、2 つのノードグループがあります。したがって、使用可能なデータメモリーの合計はデータノードごとに 2 * [DataMemory](#) です。

すべてのノードで [DataMemory](#) と [IndexMemory](#) を同じ値に設定することを強くお勧めします。データの分布はクラスタ内のすべてのノードで均等であるため、各ノードで使用できるスペースの最大量はクラスタ内でもっとも小さいノードの最大量と同じになります。

[DataMemory](#) は変更できますが、それを減らすとリスクが高くなる可能性があります。そうすると、メモリー領域が不足しているためにノードまたは NDB Cluster 全体が再起動できなくなる可能性があります。これらの値を増やすことは問題ありませんが、このようなアップグレードはソフトウェアのアップグレードと同じ方法で行うことをお勧めします。つまり、最初に構成ファイルを更新し、次に管理サーバーを起動して、各データノードを順に再起動します。

[MinFreePct](#). [DataMemory](#) を含むデータノードリソースの比率 (デフォルトでは 5%) は、再起動の実行時にデータノードがメモリーを使い果たさないように予約されています。これは、[MinFreePct](#) データノード構成パラメータを使用して調整できます (デフォルトは 5)。

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	unsigned
デフォルト	5
範囲	0 - 100

再起動タイプ

N (NDB 8.0.13)

更新によって、使用されるインデックスメモリーの量は増えません。挿入はただちに有効になりますが、行の削除はトランザクションがコミットされるまで実際には行われません。

トランザクションパラメータ。次に説明する `[ndbd]` のいくつかのパラメータは、システムで処理できる並列トランザクションの数とトランザクションのサイズに影響を与えるという点で重要です。 `MaxNoOfConcurrentTransactions` は、ノード内で実行できる並列トランザクションの数を設定します。 `MaxNoOfConcurrentOperations` は、同時に更新フェーズに入ったり、ロックしたりできるレコードの数を設定します。

これらのパラメータはどちらも (特に `MaxNoOfConcurrentOperations` は)、デフォルト値を使用せずに特定の値を設定するユーザーのターゲットになる可能性があります。デフォルト値は、小規模なトランザクションを使用するシステムで過大なメモリーが使用されないように設定されています。

`MaxDMLOperationsPerTransaction` は、特定のトランザクションで実行できる DML 操作の最大数を設定します。

- `MaxNoOfConcurrentTransactions`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	4096
範囲	32 - 4294967039 (0xFFFFFFFF)
非推奨	NDB 8.0.19
再起動タイプ	N (NDB 8.0.13)

各クラスタデータノードには、クラスタ内の個々のアクティブなトランザクションに対するトランザクションレコードが必要です。トランザクションを調整するタスクは、すべてのデータノードに分配されます。クラスタ内のトランザクションレコードの合計数は、特定のノード内のトランザクションの数にクラスタ内のノードの数を掛けた値です。

トランザクションレコードは、個々の MySQL サーバーに割り当てられます。MySQL サーバーへの個々の接続には、少なくとも 1 つのトランザクションレコードと、その接続でアクセスされるテーブルごとに追加のトランザクションオブジェクトが必要です。つまり、クラスタ内のトランザクション合計数の妥当な最小値を次のように表すことができます。

```
TotalNoOfConcurrentTransactions =
(maximum number of tables accessed in any single transaction + 1)
* number of SQL nodes
```

クラスタを使用する 10 個の SQL ノードがあるとして、10 個のテーブルが関与する 1 回の結合には、11 個のトランザクションレコードが必要です。トランザクション内にこのような結合が 10 回ある場合、このトランザクションには MySQL サーバーあたり $10 * 11 = 110$ 個のトランザクションレコード (または合計 $110 * 10 = 1100$ 個のトランザクションレコード) が必要です。各データノードは、`TotalNoOfConcurrentTransactions` / データノードの数を処理することが期待されます。4 つのデータノードを持つ NDB Cluster の場合、これは各データノードの `MaxNoOfConcurrentTransactions` を $1100 / 4 = 275$ に設定することを意味します。また、単一ノードグループがすべての同時トランザクションに対応できるようにすることで、障害リカバリを提供する必要があります。つまり、各データノード `MaxNoOfConcurrentTransactions` では、`TotalNoOfConcurrentTransactions` / ノードグループの数と等しいトランザクションの数を十分に処理できます。このクラスタにノードグループが 1 つだけある場合は、`MaxNoOfConcurrentTransactions` を 1100 (クラスタ全体の並列トランザクションの数と同じ) に設定するようにしてください。

また、各トランザクションに少なくとも 1 つの操作が関与します。このため、`MaxNoOfConcurrentTransactions` に設定する値は常に `MaxNoOfConcurrentOperations` の値以下にします。

このパラメータは、すべてのクラスタデータノードで同じ値に設定する必要があります。これは、データノードの機能停止時に、残存するもっとも古いノードが機能停止したノードで実行されていたトランザクションのトランザクション状態を再作成するためです。

ローリング再起動を使用してこの値を変更できますが、実行中のクラスタ上のトラフィック量は、発生するトランザクションが新旧どちらか低い方のレベルを超えない程度にする必要があります。

デフォルト値は 4096 です。

- [MaxNoOfConcurrentOperations](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	32K
範囲	32 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

このパラメータの値は、トランザクションのサイズと数に合わせて調整することをお勧めします。少数の操作とレコードしか関与しないトランザクションを実行する場合は、通常、このパラメータのデフォルト値で十分です。多数のレコードが関与する大規模なトランザクションを実行する場合は、通常、その値を増やす必要があります。

クラスタデータを更新する各トランザクションでは、トランザクションコーディネータと実際の更新が実行されるノードの両方でレコードが保持されます。これらのレコードには、ロールバック、ロックキュー、およびその他の目的に使用する Undo レコードを見つけるために必要な状態情報が含まれています。

このパラメータは、最低でも、トランザクションで同時に更新されるレコードの数をクラスタデータノードの数で割った値に設定するようにしてください。たとえば、4 つのデータノードがあるクラスタで、トランザクションを使用して 100 万件の並列更新を処理することが予想される場合は、この値を $1000000/4 = 250000$ に設定します。障害からの回復力を提供できるように、このパラメータを、個々のデータノードでノードグループの負荷を処理できる十分な大きさの値に設定することをお勧めします。つまり、[並列操作の合計数/ノードグループの数](#)と同じ値に設定するようにしてください。(ノードグループが 1 つだけの場合、これはクラスタ全体の並列操作の合計数と同じです。)

各トランザクションには常に少なくとも 1 つの操作が関与しているため、[MaxNoOfConcurrentOperations](#) の値は常に [MaxNoOfConcurrentTransactions](#) の値以上にします。

ロックを設定する読み取りクエリーでも、操作レコードが作成されます。ノード間の分布が完全でない場合に対応できるように、個々のノード内にある程度の追加スペースが割り当てられます。

クエリーで一意的ハッシュインデックスが使用されると、実際にはトランザクション内のレコードあたり 2 つの操作レコードが使用されます。1 つ目のレコードはインデックステーブルの読み取りを表し、2 つ目はバーステーブルに対する操作を扱います。

デフォルト値は 32768 です。

このパラメータは、実際には別個に構成できる 2 つの値を扱います。これらのうち 1 つ目は、トランザクションコーディネータによって配置される操作レコードの数を指定します。2 つ目の部分は、データベースにローカルに格納される操作レコードの数を指定します。

8 ノードのクラスタで実行される非常に大規模なトランザクションでは、そのトランザクションに関与する読み取り、更新、および削除と同じ数の操作レコードがトランザクションコーディネータ内に必要です。ただし、これらの操作レコードは 8 個のノードすべてに分散しています。このため、1 つの非常に大規模なトランザクション用にシステムを構成する必要がある場合は、この 2 つの部分を実個に構成することをお勧めします。[MaxNoOfConcurrentOperations](#) は常に、ノードのトランザクションコーディネータ部分の操作レコードの数を計算するために使用されます。

操作レコードのメモリ要件を考慮することも重要です。これらは、レコードあたり約 1K バイトを消費します。

- [MaxNoOfLocalOperations](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	UNDEFINED
範囲	32 - 4294967039 (0xFFFFFFFF)

非推奨	NDB 8.0.19
再起動タイプ	N (NDB 8.0.13)

デフォルトでは、このパラメータは $1.1 * \text{MaxNoOfConcurrentOperations}$ として計算されます。これは、多くの同時トランザクションを含み、そのいずれもが大規模ではないシステムに適合します。一度に 1 つの非常に大規模なトランザクションが実行され、多数のノードがある場合は、このパラメータを明示的に指定してデフォルトをオーバーライドすることをお勧めします。

このパラメータは NDB 8.0.19 の時点では非推奨であり、将来の NDB Cluster リリースで削除される予定です。また、このパラメータは `TransactionMemory` パラメータと互換性がありません。クラスタ構成ファイル (`config.ini`) で両方のパラメータの値を設定しようとすると、管理サーバーは起動を拒否します。

- `MaxDMLOperationsPerTransaction`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	operations (DML)
デフォルト	4294967295
範囲	32 - 4294967295
再起動タイプ	N (NDB 8.0.13)

このパラメータは、トランザクションのサイズを制限します。これを超える数の DML 操作が必要になった場合、トランザクションは中止されます。トランザクションあたりの最小操作数は 32 ですが、`MaxDMLOperationsPerTransaction` を 0 に設定すると、トランザクションあたりの DML 操作の数に対する制限を無効にできます。最大 (およびデフォルト) は 4294967295 です。

このパラメータの値は、`MaxNoOfConcurrentOperations` に設定されている値を超えることはできません。

トランザクションの一時ストレージ. 次の一連の `[ndbd]` パラメータは、クラスタトランザクションの一部であるステートメントの実行時に一時ストレージを決定するために使用されます。このステートメントが完了し、クラスタがコミットまたはロールバックを待機しているときに、すべてのレコードが解放されます。

これらのパラメータのデフォルト値は、ほとんどの状況に適しています。ただし、多数の行または操作が関与するトランザクションをサポートする必要がある場合は値を増やして、システム内の並列性を高める必要があることがあります。一方、比較的小規模なトランザクションを必要とするアプリケーションでは、これらの値を減らしてメモリーを節約できます。

- `MaxNoOfConcurrentIndexOperations`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	8K
範囲	0 - 4294967039 (0xFFFFFFFF)
非推奨	NDB 8.0.19
再起動タイプ	N (NDB 8.0.13)

一意のハッシュインデックスを使用するクエリーでは、クエリーの実行フェーズで別の一時的な操作レコードのセットが使用されます。このパラメータは、そのレコードプールのサイズを設定します。このため、このレコードはクエリーの一部を実行している間だけ割り当てられます。この部分の実行が完了した直後にレコードは解放されます。中止とコミットを処理するために必要な状態では通常の操作レコードによって処理され、プールサイズは `MaxNoOfConcurrentOperations` パラメータで設定されます。

このパラメータのデフォルト値は 8192 です。一意のハッシュインデックスを使用して非常に高い並列性を実現させるまれなケースでのみ、この値を増やす必要があります。クラスタに高いレベルの並列性が不必要なことを DBA が確信している場合は、小さい値を使用してメモリーを節約できます。

このパラメータは NDB 8.0.19 の時点では非推奨であり、将来の NDB Cluster リリースで削除される予定です。また、このパラメータは [TransactionMemory](#) パラメータと互換性がありません。クラスタ構成ファイル (`config.ini`) で両方のパラメータの値を設定しようとすると、管理サーバーは起動を拒否します。

- [MaxNoOfFiredTriggers](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	4000
範囲	0 - 4294967039 (0xFFFFFFFF)
非推奨	NDB 8.0.19
再起動タイプ	N (NDB 8.0.13)

[MaxNoOfFiredTriggers](#) のデフォルト値は 4000 です。ほとんどの状況では、これで十分です。場合によっては、クラスタに並列性はそれほど必要ないと DBA が確信している場合は、値を減らすこともできます。

一意のハッシュインデックスに影響を与える操作が実行されたときに、レコードが作成されます。一意のハッシュインデックスを使用してテーブル内のレコードを挿入または削除したり、一意のハッシュインデックスの一部であるカラムを更新したりすると、インデックステーブルで挿入または削除が発生します。生成されたレコードは、このインデックステーブル操作を発生させた元の操作の完了を待機している間、インデックステーブル操作を表すために使用されます。この操作は短期間ですが、一意のハッシュインデックスのセットを含むベーステーブルに対して多数の並列書き込み操作が行われる状況では、レコードプール内に多数のレコードを必要とする可能性があります。

このパラメータは NDB 8.0.19 の時点では非推奨であり、将来の NDB Cluster リリースで削除される予定です。また、このパラメータは [TransactionMemory](#) パラメータと互換性がありません。クラスタ構成ファイル (`config.ini`) で両方のパラメータの値を設定しようとすると、管理サーバーは起動を拒否します。

- [TransactionBufferMemory](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	bytes
デフォルト	1M
範囲	1K - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

このパラメータの影響を受けるメモリーは、インデックステーブルの更新と一意のインデックスの読み取り時に発生した操作を追跡するために使用されます。このメモリーは、これらの操作のキーおよびカラムの情報を格納するために使用されます。このパラメータの値をデフォルトから変更する必要があるのは、非常にまれなケースだけです。

[TransactionBufferMemory](#) のデフォルト値は 1M バイトです。

通常読み取りおよび書き込み操作でも同じようなバッファが使用されますが、その使用期間はさらに短くなります。コンパイル時のパラメータ `ZATTRBUF_FILESIZE` (`ndb/src/kernel/blocks/Dbtc/Dbtc.hpp` にあります) は 4000 * 128 バイト (500K バイト) に設定されています。キー情報用の同様のバッファ `ZDATABUF_FILESIZE` (これも `Dbtc.hpp` にあります) には、4000 * 16 = 62.5K バイトのバッファースペースが含まれています。`Dbtc` は、トランザクションのコーディネーションを扱うモジュールです。

トランザクションリソース割当てパラメータ。次のリストのパラメータは、トランザクションコーディネータ (DBTC) でトランザクションリソースを割り当てるために使用されます。これらのいずれかをデフォルト (0) に設定したままにすると、対応するリソースの推定合計データノード使用量の 25% のトランザクションメモリー専用になります。これらのパラメータに指定できる実際の最大値は、通常、データノードで使用可能なメモリー量によって制限されます。これらの値を設定しても、データノードに割り当てられるメモリーの合計量には影響しません。また、[MaxDMLOperationsPerTransaction](#)、[MaxNoOfConcurrentIndexOperations](#)、[MaxNoOfConcurrentOperations](#)、[MaxNoOfConcurrentScans](#)、[MaxNoOfConcurrentTransactions](#)、[MaxNoOfFiredTriggers](#)、[MaxNoOfLocalScans](#) または

`TransactionBufferMemory` の設定に関係なく、データノード用に予約された内部レコードの数を制御することに注意する必要があります ([トランザクションパラメータ](#) および [トランザクションの一時ストレージ](#) を参照)。

- `ReservedConcurrentIndexOperations`

バージョン (またはそれ以降)	NDB 8.0.16
タイプまたは単位	numeric
デフォルト	0
範囲	0 - 4294967039 (0xFFFFFFFF)
追加	NDB 8.0.16
再起動タイプ	N (NDB 8.0.13)

1 つのデータノード上に専用リソースを持つ同時インデックス操作の数。

- `ReservedConcurrentOperations`

バージョン (またはそれ以降)	NDB 8.0.16
タイプまたは単位	numeric
デフォルト	0
範囲	0 - 4294967039 (0xFFFFFFFF)
追加	NDB 8.0.16
再起動タイプ	N (NDB 8.0.13)

単一のデータノード上のトランザクションコーディネータに専用のリソースを持つ同時操作の数。

- `ReservedConcurrentScans`

バージョン (またはそれ以降)	NDB 8.0.16
タイプまたは単位	numeric
デフォルト	0
範囲	0 - 4294967039 (0xFFFFFFFF)
追加	NDB 8.0.16
再起動タイプ	N (NDB 8.0.13)

1 つのデータノード上に専用のリソースを持つ同時スキャンの数。

- `ReservedConcurrentTransactions`

バージョン (またはそれ以降)	NDB 8.0.16
タイプまたは単位	numeric
デフォルト	0
範囲	0 - 4294967039 (0xFFFFFFFF)
追加	NDB 8.0.16
再起動タイプ	N (NDB 8.0.13)

単一のデータノード上に専用リソースを持つ同時トランザクションの数。

- `ReservedFiredTriggers`

バージョン (またはそれ以降)	NDB 8.0.16
タイプまたは単位	numeric

デフォルト	0
範囲	0 - 4294967039 (0xFFFFFFFF)
追加	NDB 8.0.16
再起動タイプ	N (NDB 8.0.13)

単一の ndbd(DB) ノードに専用のリソースを持つトリガーの数。

- [ReservedLocalScans](#)

バージョン (またはそれ以降)	NDB 8.0.16
タイプまたは単位	numeric
デフォルト	0
範囲	0 - 4294967039 (0xFFFFFFFF)
追加	NDB 8.0.16
再起動タイプ	N (NDB 8.0.13)

1 つのデータノード上に専用のリソースを持つ同時フラグメントスキャンの数。

- [ReservedTransactionBufferMemory](#)

バージョン (またはそれ以降)	NDB 8.0.16
タイプまたは単位	numeric
デフォルト	0
範囲	0 - 4294967039 (0xFFFFFFFF)
追加	NDB 8.0.16
非推奨	NDB 8.0.19
再起動タイプ	N (NDB 8.0.13)

各データノードに割り当てられたキーおよび属性データの動的バッファ領域 (バイト単位)。

- [TransactionMemory](#)

バージョン (またはそれ以降)	NDB 8.0.19
タイプまたは単位	bytes
デフォルト	0
範囲	0 - 16384G
追加	NDB 8.0.19
再起動タイプ	N (NDB 8.0.13)

このパラメータは、各データノードのトランザクションに割り当てられるメモリー (バイト単位) を決定します。トランザクションメモリーの設定は、次の 3 つの方法のいずれかで処理できます:

- 多くの構成パラメータは、[TransactionMemory](#) と互換性がありません。これらのいずれかが設定されている場合、トランザクションメモリーは NDB 8.0.19 より前の状態で計算されます。これらのパラメータのいずれも [TransactionMemory](#) と同時に設定できないことに注意してください。設定しようとする、管理サーバーを起動できません ([TransactionMemory](#) と互換性のないパラメータ を参照)。
- [TransactionMemory](#) が設定されている場合、この値はトランザクションメモリーの決定に使用されます。

- 互換性のないパラメータも `TransactionMemory` も設定されていない場合、トランザクションメモリーは NDB によって `DataMemory` 構成パラメータの値の 10% に設定されます。

`TransactionMemory` と互換性のないパラメータ。 次のパラメータは `TransactionMemory` と同時に使用できず、NDB 8.0.19 の時点で非推奨になりました:

- `MaxNoOfConcurrentIndexOperations`
- `MaxNoOfFiredTriggers`
- `MaxNoOfLocalOperations`
- `MaxNoOfLocalScans`

`TransactionMemory` がクラスタ構成ファイル (`config.ini`) にも設定されている場合、リストされているパラメータのいずれかを明示的に設定すると、管理ノードは起動しません。

NDB Cluster データノードでのリソース割り当ての詳細は、[セクション23.3.3.13「データノードのメモリー管理」](#)を参照してください。

スキャンとバッファリング。 (`ndb/src/kernel/blocks/Dblqh/Dblqh.hpp` 内の) `Dblqh` モジュールには、読み取りと更新に影響を与える追加の `[ndbd]` パラメータがあります。これらには、デフォルトで 10000 * 128 バイト (1250K バイト) に設定される `ZATTRINBUF_FILESIZE` と、デフォルトで 10000 * 16 バイト (およそ 156K バイト) のバッファースペースに設定される `ZDATABUF_FILE_SIZE` が含まれます。これまでに、これらのコンパイル時制限のいずれかを増やすことを示すユーザーからの報告または弊社の大規模なテストの結果はありません。

- `BatchSizePerLocalScan`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	256
範囲	1 - 992
非推奨	NDB 8.0.19
再起動タイプ	N (NDB 8.0.13)

このパラメータは、並列スキャン操作を処理するために使用されるロックレコードの数を計算するために使用されます。

`BatchSizePerLocalScan` は、SQL ノードで定義される `BatchSize` と深い関係があります。

NDB 8.0.19 の時点では非推奨です。

- `LongMessageBuffer`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	bytes
デフォルト	64M
範囲	512K - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

これは、個々のノード内およびノード間でメッセージを渡すために使用される内部バッファです。デフォルトは 64M バイトです。

ほとんどの場合、このパラメータをデフォルトから変更する必要はありません。

- `MaxFKBuildBatchSize`

バージョン (またはそれ以降)	NDB 8.0.13
-----------------	------------

タイプまたは単位	integer
デフォルト	64
範囲	16 - 512
再起動タイプ	

外部キーの作成に使用される最大スキャンバッチサイズ。このパラメータに設定された値を増やすと、進行中のトランザクションへの影響が大きくなるため、外部キービルドの構築が高速化される場合があります。

- [MaxNoOfConcurrentScans](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	256
範囲	2 - 500
再起動タイプ	N (NDB 8.0.13)

このパラメータは、クラスタ内で実行できる並列スキャンの数を制御するために使用されます。各トランザクションコーディネータは、このパラメータで定義された数の並列スキャンを処理できます。各スキャンクエリーは、すべてのパーティションを並列でスキャンして実行されます。各パーティションスキャンでは、パーティションが配置されているノード内のスキャンレコードが使用されます。レコードの数は、このパラメータの値にノードの数を掛けた値です。クラスタは、クラスタ内のすべてのノードで同時に発生した [MaxNoOfConcurrentScans](#) 個のスキャンを維持できます。

スキャンは、実際には2つのケースで実行されます。これらのうち1つ目のケースは、クエリーを処理するためのハッシュまたは順序付けされたインデックスが存在しないときに発生します。この場合は、フルテーブルスキャンを実行するとクエリーが実行されます。2つ目のケースは、クエリーをサポートするハッシュインデックスが存在せず、順序付けされたインデックスが存在する場合に発生します。順序付けされたインデックスの使用は、並列の範囲スキャンの実行を意味します。この順序はローカルのパーティションでのみ維持されるため、すべてのパーティションでインデックススキャンを実行する必要があります。

[MaxNoOfConcurrentScans](#) のデフォルト値は 256 です。最大値は 500 です。

- [MaxNoOfLocalScans](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	$4 * \text{MaxNoOfConcurrentScans} * [\# \text{ of data nodes}] + 2$
範囲	32 - 4294967039 (0xFFFFFFFF)
非推奨	NDB 8.0.19
再起動タイプ	N (NDB 8.0.13)

多くのスキャンが完全に並列化されていない場合の、ローカルスキャンレコードの数を指定します。ローカルスキャンレコードの数が指定されていない場合は、ここに示すように計算されます。

$4 * \text{MaxNoOfConcurrentScans} * [\# \text{ data nodes}] + 2$

このパラメータは NDB 8.0.19 の時点では非推奨であり、将来の NDB Cluster リリースで削除される予定です。また、このパラメータは [TransactionMemory](#) パラメータと互換性がありません。クラスタ構成ファイル ([config.ini](#)) で両方のパラメータの値を設定しようとすると、管理サーバーは起動を拒否します。

- [MaxParallelCopyInstances](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	0

範囲	0 - 64
再起動タイプ	

このパラメータは、ノードの再起動またはシステムの再起動のコピーフェーズで使用されるパラレル化を設定します。現在起動中のノードが、最新のノードから変更されたレコードをコピーすることによって、現在のデータを持つノードと同期化されている場合です。このような場合、完全な並列性によって過負荷状態が発生する可能性があります。そのため、`MaxParallelCopyInstances` ではそれを減らす手段が提供されます。このパラメータのデフォルト値は 0 です。この値は、有効な並列性が、ノードを更新するノードだけでなく、ノード内の LDM インスタンスの数と同じであることを意味します。

- `MaxParallelScansPerFragment`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	bytes
デフォルト	256
範囲	1 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

順次処理のキューイングが開始される前に許可される並列スキャン (TUP スキャンおよび TUX スキャン) の最大数を構成できます。これを増やすことにより、多数のスキャンを並列で実行するときに未使用の CPU を利用して、パフォーマンスを向上させることができます。

- `MaxReorgBuildBatchSize`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	64
範囲	16 - 512
再起動タイプ	

テーブルパーティションの再編成に使用される最大スキャンバッチサイズ。このパラメータに設定された値を増やすと、進行中のトラフィックへの影響が大きくなるため、再編成が高速化される場合があります。

- `MaxUIBuildBatchSize`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	64
範囲	16 - 512
再起動タイプ	

一意キーの作成に使用される最大スキャンバッチサイズ。このパラメータに設定された値を増やすと、進行中のトラフィックへの影響が大きくなるため、このようなビルドが高速化される可能性があります。

メモリー割り当て

MaxAllocate

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	unsigned
デフォルト	32M
範囲	1M - 1G
再起動タイプ	N (NDB 8.0.13)

これは、テーブル用のメモリーを割り当てるときに使用するメモリーユニットの最大サイズです。NDB で「メモリー不足」エラーが発生したが、使用可能なすべてのメモリーがまだ使用されていないことをクラスタログまたは DUMP 1000 の出力を調べることで明らかになる場合は、このパラメータの値 (MaxNoOfTables またはその両方) を増やして、NDB で十分なメモリーを使用できるようにすることができます。

複数のトランスポータ

バージョン 8.0.20 以降、NDB はデータノードのペア間の通信に複数のトランスポータを割り当てます。このように割り当てられたトランスポータの数は、そのリリースで導入された `NodeGroupTransporters` パラメータに適切な値を設定することによって影響を受ける可能性があります。

NodeGroupTransporters

バージョン (またはそれ以降)	NDB 8.0.20
タイプまたは単位	integer
デフォルト	0
範囲	0 - 32
追加	NDB 8.0.20
再起動タイプ	N (NDB 8.0.13)

このパラメータは、同じノードグループ内のノード間で使用されるトランスポータの数を決定します。デフォルト値 (0) は、使用されるトランスポータの数がノード内の LDM の数と同じであることを意味します。ほとんどのユースケースではこれで十分です。したがって、この値をデフォルトから変更する必要はほとんどありません。

`NodeGroupTransporters` を LDM スレッドの数または TC スレッドの数 (いずれか大きい方) より大きい値に設定すると、NDB はこれらの 2 つのスレッドの最大数を使用します。つまり、これより大きい値は事実上無視されます。

ハッシュマップサイズ

DefaultHashMapSize

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	LDM threads
デフォルト	240
範囲	0 - 3840
再起動タイプ	N (NDB 8.0.13)

このパラメータの元の使用目的は、アップグレードを容易にし、特にデフォルトのハッシュマップサイズが異なる非常に古いリリースとの間のダウングレードを容易にすることでした。NDB Cluster 7.3 以降から新しいバージョンにアップグレードする場合、これは問題ではありません。

`DefaultHashMapSize` を 3840 に設定してテーブルを作成または変更したあと、オンラインでこのパラメータを減らす方法は、現在サポートされていません。

ロギングとチェックポイント。 次の `[ndbd]` パラメータは、ログとチェックポイントの動作を制御します。

- `FragmentLogFileSize`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	bytes
デフォルト	16M
範囲	4M - 1G
再起動タイプ	IN (NDB 8.0.13)

このパラメータを設定すると、Redo ログファイルのサイズを直接制御できます。これは、NDB Cluster が高負荷で動作しており、新しいフラグメントログファイルを開こうとする前にフラグメントログファイルを十分に迅速に閉じることができない (一度に開くことができるフラグメントログファイルは 2 つだけです)、フラグメントログ

ファイルのサイズを増やすと、新しいフラグメントログファイルを開く前にクラスタの時間が長くなる可能性があります。このパラメータのデフォルト値は 16M です。

フラグメントログファイルの詳細は、[NoOfFragmentLogFiles](#) の説明を参照してください。

- [InitialNoOfOpenFiles](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	files
デフォルト	27
範囲	20 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

このパラメータは、開いたファイルに割り当てる初期の内部スレッド数を設定します。

デフォルト値は 27 です。

- [InitFragmentLogFiles](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	[see values]
デフォルト	SPARSE
範囲	SPARSE, FULL
再起動タイプ	IN (NDB 8.0.13)

デフォルトでは、データノードの初期起動の実行時はフラグメントログファイルがまばらに作成されます。つまり、使用するオペレーティングシステムとファイルシステムによって、必ずしもすべてのバイトがディスクに書き込まれるわけではありません。ただし、このパラメータを使用すると、使用されているプラットフォームやファイルシステムのタイプに関係なく、この動作をオーバーライドしてすべてのバイトが強制的に書き込まれるように設定できます。[InitFragmentLogFiles](#) は 2 つの値のいずれかを取ります。

- **SPARSE** フラグメントログファイルがまばらに作成されます。これはデフォルト値です。
- **FULL**。フラグメントログファイルのすべてのバイトが強制的にディスクに書き込まれます。

オペレーティングシステムとファイルシステムによっては、[InitFragmentLogFiles=FULL](#) を設定することで、Redo ログへの書き込み時の I/O エラーを解消できる場合があります。

- [EnablePartialLcp](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	boolean
デフォルト	true
範囲	...
再起動タイプ	N (NDB 8.0.13)

true の場合、部分的なローカルチェックポイントを有効にします: これは、各 LCP がデータベース全体の一部のみを記録し、最後の LCP 以降に変更された行を含むレコードのみを記録することを意味します。変更された行がない場合、LCP は LCP 制御ファイルのみを更新し、データファイルは更新しません。

[EnablePartialLcp](#) が無効 (**false**) の場合、各 LCP は単一のファイルのみを使用し、完全チェックポイントを書き込みます。これには LCP に必要なディスク容量が最小限ですが、LCP ごとに書き込み負荷が増加します。デフォルト

値は enabled (`true`) です。部分 LCPS で使用される領域の割合は、[RecoveryWork](#) 構成パラメータの設定によって変更できます。

LCP 全体および一部に使用されるファイルおよびディレクトリの詳細は、[NDB Cluster Data Node File System Directory](#) を参照してください。

このパラメータを `false` に設定すると、適応 LCP 制御メカニズムで使用されるディスク書き込み速度の計算も無効になります。

- [LcpScanProgressTimeout](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	second
デフォルト	60
範囲	0 - 4294967039 (0xFFFFFFFF)
バージョン (またはそれ以降)	NDB 8.0.19
タイプまたは単位	second
デフォルト	180
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

ローカルチェックポイントのフラグメントスキャンウォッチドッグは、ローカルチェックポイントの一部として実行されている各フラグメントスキャンが進行していないかどうか定期的にチェックし、特定の時間が経過しても進行しない場合はそのノードをシャットダウンします。この間隔は、LCP フラグメントスキャンウォッチドッグがノードを停止するまでにローカルチェックポイントを停止できる最大時間を設定する [LcpScanProgressTimeout](#) データノード構成パラメータを使用して設定できます。

デフォルト値は 60 秒です (以前のリリースと互換性があります)。このパラメータを 0 に設定すると、LCP のフラグメントスキャンウォッチドッグが完全に無効化されます。

- [MaxNoOfOpenFiles](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	unsigned
デフォルト	0
範囲	20 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

このパラメータは、開いたファイルに割り当てる内部スレッド数の上限を設定します。このパラメータの変更が必要な状況が発生した場合は、それをバグとして報告するようにしてください。

デフォルト値は 0 です。ただし、このパラメータに設定できる最小値は 20 です。

- [MaxNoOfSavedMessages](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	25
範囲	0 - 4294967039 (0xFFFFFFFF)

再起動タイプ	N (NDB 8.0.13)
--------	----------------

このパラメータは、エラーログに書き込まれるエラーの最大数と、既存のトレースファイルを上書きする前に保持されるトレースファイルの最大数を設定します。トレースファイルは、何らかの理由でノードがクラッシュしたときに生成されます。

デフォルトは 25 で、これらの最大値は 25 個のエラーメッセージおよび 25 個のトレースファイルに設定されません。

- [MaxLCPStartDelay](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	seconds
デフォルト	0
範囲	0 - 600
再起動タイプ	N (NDB 8.0.13)

データノードの並列リカバリでは、実際にはテーブルデータのみが並列でコピーされ、同期されます。ディクショナリ情報やチェックポイント情報などのメタデータの同期は順次に行われます。また、ディクショナリおよびチェックポイント情報のリカバリは、ローカルチェックポイントの実行と並列で実行できません。つまり、多くのデータノードを同時に起動したり再起動したりすると、データノードがローカルチェックポイントの実行中に待たされて、ノードのリカバリ時間が長くなる可能性があります。

より多くの (場合によってはすべての) データノードでメタデータの同期を完了できるように、ローカルチェックポイントの遅延を強制できます。各データノードでメタデータの同期が完了したあとは、ローカルチェックポイントの実行中でも、すべてのデータノードで並列にテーブルデータをリカバリできます。このような遅延を強制するには、[MaxLCPStartDelay](#) を設定します。これは、データノードでメタデータの同期が継続されている間にクラスタがローカルチェックポイントの開始を待機する秒数を決定します。このパラメータは、すべてのデータノードで同じになるように `config.ini` ファイルの `[ndbd default]` セクションに設定するようにしてください。最大値は 600、デフォルトは 0 です。

- [NoOffFragmentLogFiles](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	16
範囲	3 - 4294967039 (0xFFFFFFFF)
再起動タイプ	IN (NDB 8.0.13)

このパラメータは、ノードの Redo ログファイルの数 (つまり、Redo ログギングに割り当てられるスペースの量) を設定します。Redo ログファイルはリング状に編成されるため、セット内の最初と最後のログファイル (それぞれ「head」および「tail」ログファイルとも呼ばれる) が交わらないことが非常に重要です。これらが互いに寄りすぎると、新しいログレコードを追加する余裕がないため、ノードは更新を含むすべてのトランザクションを中止し始めます。

REDO ログレコードは、そのログレコードが挿入されてから、両方の必要なローカルチェックポイントが完了するまで削除されません。チェックポイントの頻度は、この章で別途説明する固有の構成パラメータセットによって決定されます。

デフォルトのパラメータ値は 16 です。これは、デフォルトでは 16M バイトのファイル 4 個が 16 セット (合計 1024M バイト) あることを意味します。個々のログファイルのサイズは、[FragmentLogFileSize](#) パラメータを使用して構成できます。非常に多くの更新が必要なシナリオでは、Redo ログに対して十分なスペースを提供するため、[NoOffFragmentLogFiles](#) の値を 300 以上に設定する必要がある場合があります。

チェックポイントが遅く、データベースへの書き込みが多いためにログファイルがいっぱいになって、リカバリに悪影響を与えずにログの末尾をカットできない場合は、すべての更新トランザクションが内部エラーコード

410 「[Out of log file space temporarily](#)」によって中止されます。この状態は、チェックポイントが完了してログの末尾を前に移動できるようになるまで続きます。

重要

このパラメータは「実行中に」変更できません。`--initial` を使用してノードを再起動する必要があります。実行中のクラスターのすべてのデータノードでこの値を変更する必要がある場合は、(各データノードの起動時に `--initial` を使用して) ノードのローリング再起動を使用します。

• [RecoveryWork](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	60
範囲	25 - 100
再起動タイプ	N (NDB 8.0.13)

LCP ファイルの記憶域オーバーヘッドの割合。このパラメータは、`EnablePartialLcp` が true の場合、つまり部分的なローカルチェックポイントが有効な場合にのみ有効です。値が大きいほど、次のことを意味します:

- LCP ごとに書き込まれるレコードが少なくなり、LCP はより多くの領域を使用
- 再起動中にさらに作業が必要

`RecoveryWork` の値が小さいほど、次のことを意味します:

- LCP ごとに書き込まれるレコードは増えてますが、LCP で必要なディスク上の領域は少なくなります。
- 再起動中の作業が少なくなり、通常の操作中の作業が増えてますが、再起動が速くなります

たとえば、`RecoveryWork` を 60 に設定すると、LCP の合計サイズは、チェックポイントするデータのサイズの約 1 倍 0.6 = 1.6 倍になります。つまり、再起動のリストアフェーズでは、フルチェックポイントを使用する再起動時に実行される作業と比較して 60% 多くの作業が必要になります。(これは、部分 LCP を使用している場合でも完全 LCP を使用している場合よりも再起動全体が高速になるように、再起動の他のフェーズで補正されるよりも多くあります。) redo ログをいっばいにしないためには、チェックポイント中のデータ変更率の $1 + (1 / \text{RecoveryWork})$ 回書き込む必要があります。したがって、`RecoveryWork` = 60 では、変更率の約 $1 + (1 / 0.6) = 2.67$ 回書き込む必要があります。つまり、変更が毎秒 10 MByte で書き込まれる場合、チェックポイントはおよそ 26.7 MByte/秒で書き込む必要があります。

`RecoveryWork` = 40 を設定すると、LCP の合計サイズの 1.4 倍のみが必要になります (したがって、リストアフェーズにかかる時間は 10 から 15% 短縮されます)。この場合、チェックポイント書き込み率は変更率の 3.5 倍です。

NDB ソース配布には、LCP をシミュレートするためのテストプログラムが含まれています。[lcp_simulator.cc](#) は、`storage/ndb/src/kernel/blocks/backup/` にあります。Unix プラットフォームでコンパイルおよび実行するには、次に示すコマンドを実行します:

```
shell> gcc lcp_simulator.cc
shell> ./a.out
```

このプログラムには `stdio.h` 以外の依存関係はなく、NDB クラスターまたは MySQL サーバーへの接続は必要ありません。デフォルトでは、300 LCP (それぞれ挿入、更新、削除で構成される 100 LCP の 3 つのセット) がシミュレートされ、各 LCP の後に LCP のサイズがレポートされます。シミュレーションを変更するには、ソースで `recovery_work`、`insert_work` および `delete_work` の値を変更し、再コンパイルします。詳細は、プログラムのソースを参照してください。

• [InsertRecoveryWork](#)

バージョン (またはそれ以降)	NDB 8.0.13
-----------------	------------

タイプまたは単位	integer
デフォルト	40
範囲	0 - 70
再起動タイプ	N (NDB 8.0.13)

挿入された行に使用される [RecoveryWork](#) の割合。値を大きくすると、ローカルチェックポイント中の書き込み数が増加し、LCP の合計サイズが減少します。値を小さくすると LCP 中の書き込み数が減りますが、LCP に使用される領域が増えるため、リカバリに時間がかかります。このパラメータは、[EnablePartialLcp](#) が true の場合、つまり部分的なローカルチェックポイントが有効な場合にのみ有効です。

- [EnableRedoControl](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	boolean
デフォルト	false
範囲	...
再起動タイプ	N (NDB 8.0.13)

redo ログの使用を制御するための適応チェックポイント処理速度を有効にします。無効にするには、[false](#) に設定します (デフォルト)。[EnablePartialLcp](#) を [false](#) に設定すると、適応計算も無効になります。

[EnableRedoControl](#) を有効にすると、LCP をディスクに書き込む速度に関して、データノードの柔軟性が向上します。具体的には、このパラメータを有効にすると、LCP が完了し、redo ログがより迅速にトリミングされるように、書き込み率が高くなり、リカバリ時間とディスク領域要件が削減されます。この機能により、データノードは、Non-Volatile Memory Express (NVMe) を使用して、ソリッドステートドライブ (SSD) などの最新のソリッドステートストレージデバイスおよびプロトコルから使用可能なより高い速度の I/O およびより高い帯域幅をより効果的に使用できます。

従来のハードディスク (HDD) を使用するシステムなど、ソリッドステートテクノロジーを採用するシステムに対して I/O または帯域幅が制約されているシステムに [NDB](#) がまだ広くデプロイされているため、このパラメータは現在 [false](#) (無効) にデフォルト設定されています。このような設定では、[EnableRedoControl](#) メカニズムによって I/O サブシステムが飽和状態になり、データノードの入出力の待機時間が長くなる可能性があります。特に、これにより、制約付き IO サブシステムをデータノード LCP および redo ログファイルと共有しているテーブルスペースまたはログファイルグループを持つ「[NDB データノード](#)」テーブルで問題が発生する可能性があります。このような問題には、GCP 停止エラーによるノードまたはクラスタの障害が含まれる可能性があります。

メタデータオブジェクト。次の一連の [\[ndbd\]](#) パラメータは、インデックス、イベント、およびクラスタ間のレプリケーションで使用される属性、テーブル、インデックス、およびトリガーオブジェクトの最大数を定義するために使用されるメタデータオブジェクトのプールサイズを定義します。

注記

これらはクラスタに対する「提案」としてのみ機能し、指定されていないものは表示されているデフォルト値に戻ります。

- [MaxNoOfAttributes](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	1000
範囲	32 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

このパラメータは、クラスタに定義できる属性の推奨最大数を設定します。[MaxNoOfTables](#) と同様、厳密な上限の役割を果たすためのものではありません。

(古い NDB Cluster リリースでは、このパラメータは特定の操作の強い制限として扱われることがありました。これにより、NDB Cluster レプリケーションでは、レプリケート可能な数より多くのテーブルを作成できる場合に問題が発生し、状況によっては可能なときに混乱することがあります (または不可能な場合によっては、`MaxNoOfAttributes` 属性を複数作成することもあります)。

デフォルト値は 1000 です。指定可能な最小値は 32 です。最大値は 4294967039 です。すべてのメタデータがサーバー上で完全にレプリケートされるため、各属性はノードあたり 200 バイト前後のストレージを消費します。

`MaxNoOfAttributes` を設定するときは、今後実行する可能性がある `ALTER TABLE` ステートメントを事前に用意することが重要です。これは、クラスタテーブルで `ALTER TABLE` の実行中に、元のテーブルに含まれる 3 倍の数の属性が使用されるためであり、この数の 2 倍を許可することをお勧めします。たとえば、属性 (`greatest_number_of_attributes`) の数が最大の「NDB Cluster」テーブルに 100 個の属性がある場合、`MaxNoOfAttributes` の値の適切な開始点は $6 * \text{greatest_number_of_attributes} = 600$ になります。

また、テーブルあたりの平均属性数を見積もり、それに `MaxNoOfTables` を掛けることもお勧めします。この値が前の段落で得られた値より大きい場合は、大きい値を使用するようにしてください。

必要なすべてのテーブルが問題なく作成できると仮定して、パラメータの構成後に実際に `ALTER TABLE` を試行し、この数が十分であることを確認するようにしてください。これが成功しなかった場合は、`MaxNoOfAttributes` をさらに `MaxNoOfTables` の数だけ増やして、再度テストしてください。

- `MaxNoOfTables`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	128
範囲	8 - 20320
再起動タイプ	N (NDB 8.0.13)

テーブルオブジェクトは、クラスタ内のテーブルごと、および一意のハッシュインデックスごとに割り当てられます。このパラメータは、クラスタ全体のテーブルオブジェクトの推奨最大数を設定します。`MaxNoOfAttributes` と同様、厳密な上限の役割を果たすものではありません。

(古い NDB Cluster リリースでは、このパラメータは特定の操作の強い制限として扱われることがありました。これにより、NDB Cluster レプリケーションで、レプリケート可能な数より多くのテーブルを作成できた場合に問題が発生し、状況によっては、`MaxNoOfTables` テーブルを複数作成するために可能な場合 (または不可能な場合) に混乱することがありました。)

`BLOB` データ型を持つ各属性では、`BLOB` データの大部分を格納するために追加のテーブルが使用されます。テーブルの合計数を定義するときは、これらのテーブルも考慮に入れる必要があります。

このパラメータのデフォルト値は 128 です。最小値は 8、最大値は 20320 です。各テーブルオブジェクトは、ノードあたりおよそ 20K バイトを消費します。

注記

`MaxNoOfTables`、`MaxNoOfOrderedIndexes`、および `MaxNoOfUniqueHashIndexes` の合計が $2^{32} - 2$ (4294967294) を超えないようにする必要があります。

- `MaxNoOfOrderedIndexes`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	128
範囲	0 - 4294967039 (0xFFFFFFFF)

再起動タイプ

N (NDB 8.0.13)

クラスタ内の順序付けされたインデックスごとに、インデックスされた内容とそのストレージセグメントを記述するオブジェクトが割り当てられます。デフォルトでは、そのように定義された各インデックスによって、順序付けされたインデックスも定義されます。個々の一意のインデックスおよび主キーには、順序付けされたインデックスとハッシュインデックスの両方が含まれています。 [MaxNoOfOrderedIndexes](#) は、システム内で一度に使用できる順序付けされたインデックスの合計数を設定します。

このパラメータのデフォルト値は 128 です。各インデックスオブジェクトは、ノードあたりおよそ 10K バイトのデータを消費します。

注記

[MaxNoOfTables](#)、[MaxNoOfOrderedIndexes](#)、および [MaxNoOfUniqueHashIndexes](#) の合計が $2^{32} - 2$ (4294967294) を超えないようにする必要があります。

• [MaxNoOfUniqueHashIndexes](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	64
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

主キー以外の個々の一意のインデックスには、一意キーをインデックスされたテーブルの主キーにマップする専用のテーブルが割り当てられます。デフォルトでは、個々の一意のインデックスに対して順序付けされたインデックスも定義されます。これを禁止するには、一意のインデックスを定義するときに **USING HASH** オプションを指定する必要があります。

デフォルト値は 64 です。各インデックスは、ノードあたりおよそ 15K バイトを消費します。

注記

[MaxNoOfTables](#)、[MaxNoOfOrderedIndexes](#)、および [MaxNoOfUniqueHashIndexes](#) の合計が $2^{32} - 2$ (4294967294) を超えないようにする必要があります。

• [MaxNoOfTriggers](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	768
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

個々の一意のハッシュインデックスには、内部更新、挿入、および削除のトリガーが割り当てられます。(これは、一意のハッシュインデックスごとに 3 つのトリガーが作成されることを意味します。) ただし、順序付けされたインデックスに必要なトリガーオブジェクトは 1 つだけです。バックアップでも、クラスタ内の通常のテーブルごとに 3 つのトリガーオブジェクトが使用されます。

クラスタ間のレプリケーションでも、内部トリガーが使用されます。

このパラメータは、クラスタ内のトリガーオブジェクトの最大数を設定します。

デフォルト値は 768 です。

• [MaxNoOfSubscriptions](#)

バージョン (またはそれ以降)	NDB 8.0.13
-----------------	------------

タイプまたは単位	unsigned
デフォルト	0
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

NDB Cluster 内の各 NDB テーブルには、NDB カーネルでのサブスクリプションが必要です。一部の NDB API アプリケーションでは、このパラメータの変更が必要または望ましい場合があります。ただし、SQL ノードとして機能する MySQL サーバーを通常どおりに使用する場合は、これを行う必要はありません。

`MaxNoOfSubscriptions` のデフォルト値は 0 です。これは、`MaxNoOfTables` と等価として扱われます。各サブスクリプションは 108 バイトを消費します。

- `MaxNoOfSubscribers`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	unsigned
デフォルト	0
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

このパラメータは NDB Cluster レプリケーションを使用している場合にのみ重要です。デフォルト値は 0 です。NDB 8.0.25 より前は、これは $2 * \text{MaxNoOfTables}$ として扱われていました。NDB 8.0.25 以降は、 $2 * \text{MaxNoOfTables} + 2 * [\text{number of API nodes}]$ として扱われます。各 MySQL サーバー (レプリケーションソースとして機能するサーバーとレプリカとして機能するサーバー) には、NDB テーブルごとに 1 つのサブスクリプションがあります。各サブスクライバは 16 バイトのメモリーを使用します。

循環レプリケーション、マルチソースレプリケーションおよび 2 台を超える MySQL サーバーを含むその他のレプリケーション設定を使用する場合は、このパラメータをレプリケーションに含まれる `mysqld` プロセスの数に増やす必要があります (多くの場合、これは常にクラスタの数と同じではありません)。たとえば、3 つの NDB Cluster を使用して循環レプリケーション設定があり、各クラスタに 1 つの `mysqld` が接続されており、これらの各 `mysqld` プロセスがソースおよびレプリカとして機能する場合は、`MaxNoOfSubscribers` を $3 * \text{MaxNoOfTables}$ と同等に設定するようにしてください。

詳細は、[セクション 23.6 「NDB Cluster レプリケーション」](#) を参照してください。

- `MaxNoOfConcurrentSubOperations`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	unsigned
デフォルト	256
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

このパラメータは、クラスタ内のすべての API ノードが一度に実行できる操作数の上限を設定します。通常の操作ではデフォルト値 (256) で十分ですが、API ノードが多数あり、それぞれが大量の操作を同時に実行しているシナリオでのみ、調整が必要になる可能性があります。

ブールパラメータ。データノードの動作は、ブール値を取る一連の `[ndbnd]` パラメータにも影響されます。これらの各パラメータは、`TRUE` として指定する場合は 1 または Y に設定し、`FALSE` として指定する場合は 0 または N に設定します。

- `CompressedLCP`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	boolean

デフォルト	false
範囲	true, false
再起動タイプ	N (NDB 8.0.13)

このパラメータを 1 に設定すると、ローカルチェックポイントが圧縮されます。使用される圧縮は `gzip --fast` と同等であり、データノードで非圧縮チェックポイントファイルの格納に必要なスペースの 50% 以上を節約できます。圧縮 LCP は、個々のデータノードまたは (`config.ini` ファイルの `[ndbd default]` セクションにこのパラメータを設定することで) すべてのデータノードで有効化できます。

重要

圧縮ローカルチェックポイントを、この機能をサポートしない MySQL バージョンを実行するクラスタにリストアすることはできません。

デフォルト値は 0 (無効) です。

- [CrashOnCorruptedTuple](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	boolean
デフォルト	true
範囲	true, false
再起動タイプ	

このパラメータを有効にすると (デフォルト)、破損したタプルが検出されるたびにデータノードが強制的にシャットダウンされます。

- [Diskless](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	true false (1 0)
デフォルト	false
範囲	true, false
再起動タイプ	IS (NDB 8.0.13)

「NDB Cluster」テーブルをディスクレスとして指定できます。つまり、テーブルはディスクにチェックポイントされず、ロギングも行われません。このようなテーブルはメインメモリーにのみ存在します。ディスクレステーブルを使用すると、クラッシュ発生時にテーブルもテーブル内のレコードも失われます。ただし、ディスクレスモードで動作するときは、ディスクレスコンピュータで `ndbd` を実行できます。

重要

この機能を使用すると、クラスタ全体がディスクレスモードで動作します。

この機能を有効にすると、クラスタのオンラインバックアップが無効になります。また、クラスタの部分的起動ができなくなります。

`Diskless` はデフォルトで無効になっています。

- [LateAlloc](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	numeric
デフォルト	1
範囲	0 - 1

再起動タイプ	N (NDB 8.0.13)
--------	----------------

管理サーバーへの接続が確立されたあとで、このデータノードのメモリーを割り当てます。デフォルトで有効。

- [LockPagesInMainMemory](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	numeric
デフォルト	0
範囲	0 - 2
再起動タイプ	N (NDB 8.0.13)

Solaris と Linux を含むいくつかのオペレーティングシステムでは、プロセスをメモリーにロックすると、ディスクへのスワップを回避できます。これを使用すると、クラスタのリアルタイム特性が保証されやすくなります。

このパラメータは、整数値 **0**、**1**、または **2** のいずれかを取ります。これらの機能を次のリストに示します。

- **0**: ロックを無効にします。これはデフォルト値です。
- **1**: プロセスのメモリーを割り当てたあとでロックを実行します。
- **2**: プロセスのメモリーを割り当てる前にロックを実行します。

権限のないユーザーがページをロックできるようにオペレーティングシステムが構成されていない場合は、このパラメータを利用するデータノードプロセスをシステムの root として実行する必要がある可能性があります。(LockPagesInMainMemory では、`mlockall` 関数が使用されます。Linux kernel 2.6.9 以降では、権限のないユーザーが `max locked memory` の制限に従ってメモリーをロックできます。詳細は、[ulimit -l](#) および <http://linux.die.net/man/2/mlock> を参照してください)。

注記

古い NDB Cluster リリースでは、このパラメータはブールでした。**0** または **false** がデフォルト設定で、ロックが無効になっています。**1** または **true** は、メモリーの割当て後にプロセスのロックを有効にしました。NDB Cluster 8.0 は、このパラメータの値として **true** または **false** をエラーとして扱います。

重要

`glibc` 2.10 以降では、`glibc` が共有プールでのロック競合を減らすためにスレッド単位のアリーナを使用し、これによって実メモリーが消費されます。通常、データノードプロセスは起動後にメモリー割り当てを実行しないため、スレッド単位のアリーナを必要としません。(このアロケータの違いがパフォーマンスに重大な影響を与えることはありません。)

この `glibc` の動作は `MALLOC_ARENA_MAX` 環境変数を使用して構成できるようになっていますが、`glibc` 2.16 より前では、このメカニズムのバグが原因でこの変数を 8 未満に設定できなかつたため、無駄になったメモリーを再利用できませんでした。(Bug #15907219。この問題の詳細は、http://sourceware.org/bugzilla/show_bug.cgi?id=13137 も参照してください。)

この問題の考えられる回避策は、`LD_PRELOAD` 環境変数を使用して `jemalloc` メモリー割り当てライブラリをプリロードし、`glibc` に付属するライブラリの代わりに使用することです。

- [ODirect](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	boolean
デフォルト	false

範囲	true, false
再起動タイプ	N (NDB 8.0.13)

このパラメータを有効にすると、NDB が LCP、バックアップ、および Redo ログで `O_DIRECT` 書き込みを試行し、多くの場合 `kswapd` と CPU の使用率が低下します。Linux で NDB Cluster を使用しているときに、2.6 以降のカーネルを使用している場合は `ODirect` を有効にします。

`ODirect` はデフォルトで無効になっています。

- `ODirectSyncFlag`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	boolean
デフォルト	false
範囲	true, false
再起動タイプ	N (NDB 8.0.13)

このパラメータを有効にすると、完了した各ファイルシステム書き込みが `fsync` へのコールとして処理されるように redo ログ書き込みが実行されます。次のいずれかの条件が満たされている場合、このパラメータの設定は無視されません:

- `ODirect` が有効になっていません。
- `InitFragmentLogFiles` は `SPARSE` に設定されています。

デフォルトで無効になっています。

- `RestartOnErrorInsert`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	error code
デフォルト	2
範囲	0 - 4
再起動タイプ	N (NDB 8.0.13)

この機能は、コードの個々のブロックをテストの一部として実行する際に、エラーの挿入が可能なデバッグバージョンをビルドする場合にのみ利用できます。

この機能はデフォルトで無効になっています。

- `StopOnError`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	boolean
デフォルト	1
範囲	0, 1

再起動タイプ	N (NDB 8.0.13)
--------	----------------

このパラメータは、データノードプロセスでエラー状態が発生したときに、プロセスを終了するか、自動的に再起動するか指定します。

このパラメータのデフォルト値は 1 です。これは、デフォルトでは、エラーによってデータノードのプロセスが停止することを意味します。

エラーが発生し、`StopOnError` が 0 の場合は、データノードプロセスが再起動されます。

MySQL Cluster Manager のユーザーは、`StopOnError` が 1 の場合、これにより、MySQL Cluster Manager エージェントが独自の再起動およびリカバリを実行した後にデータノードを再起動できないことに注意する必要があります。詳しくは [Starting and Stopping the Agent on Linux](#) をご覧ください。

- [UseShm](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	boolean
デフォルト	false
範囲	true, false
再起動タイプ	

このデータノードと、このホストでも実行されている API ノードとの間の共有メモリー接続を有効にします。有効にするには 1 に設定します。

タイムアウト、間隔、およびディスクページングの制御

クラスタデータノード内のさまざまなアクション間のタイムアウトと間隔を指定する、いくつかの `[ndbd]` パラメータがあります。ほとんどのタイムアウト値はミリ秒単位で指定します。この例外については、該当する箇所で説明します。

- [TimeBetweenWatchDogCheck](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	milliseconds
デフォルト	6000
範囲	70 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

メインスレッドがある時点で無限ループに陥ることを防ぐため、「ウォッチドッグ」スレッドがメインスレッドをチェックします。このパラメータは、チェック間のミリ秒数を指定します。プロセスが 3 回のチェック後も同じ状態の場合、ウォッチドッグスレッドはプロセスを強制終了します。

このパラメータは、実験のため、またはローカルの状態に合わせて簡単に変更できます。これをノード単位で指定することもできますが、指定することはほとんどありません。

デフォルトのタイムアウトは 6000 ミリ秒 (6 秒) です。

- [TimeBetweenWatchDogCheckInitial](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	milliseconds
デフォルト	6000
範囲	70 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

これは `TimeBetweenWatchDogCheck` パラメータと似ていますが、メモリーが割り当てられる最早開始フェーズで、`TimeBetweenWatchDogCheckInitial` がストレージノード内の実行チェック間の経過時間を制御する点が異なります。

デフォルトのタイムアウトは 6000 ミリ秒 (6 秒) です。

- `StartPartialTimeout`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	milliseconds
デフォルト	30000
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

このパラメータは、クラスタの初期化ルーチンが呼び出されるまでにクラスタがデータノードの起動を待つ時間を指定します。このタイムアウトは、可能なかぎり、クラスタの部分的起動を回避するために使用されます。

このパラメータは、クラスタの初期起動または初期再起動の実行時にオーバーライドされます。

デフォルト値は 30000 ミリ秒 (30 秒) です。0 にするとタイムアウトが無効になり、すべてのノードが使用可能な場合のみクラスタを起動できるようになります。

- `StartPartitionedTimeout`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	milliseconds
デフォルト	0
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

`StartPartialTimeout` ミリ秒の待機後にクラスタが起動できる状態になっても、まだパーティション化された状態の可能性がある場合、クラスタはこのタイムアウトが経過するまで待機します。`StartPartitionedTimeout` が 0 に設定されている場合、クラスタは無期限に待機します ($2^{32}-1$ ミリ秒または約 49.71 日)。

このパラメータは、クラスタの初期起動または初期再起動の実行時にオーバーライドされます。

- `StartFailureTimeout`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	milliseconds
デフォルト	0
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

データノードでこのパラメータで指定された時間内に起動シーケンスが完了されなかった場合、ノードの起動は失敗します。このパラメータを 0 (デフォルト値) に設定すると、データノードのタイムアウトは適用されません。

このパラメータでは、0 以外の値はミリ秒単位で測定されます。非常に多くのデータを含むデータノードでは、このパラメータを増やすようにしてください。たとえば、数ギガバイトのデータを含むデータノードの場合、ノードを再起動するのに 10-15 分 (つまり、600000-1000000 ミリ秒) の時間が必要です。

- `StartNoNodeGroupTimeout`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	milliseconds

デフォルト	15000
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

`Nodegroup = 65536` でデータノードを構成すると、そのノードはどのノードグループにも割り当てられないものとみなされます。その場合、クラスタは `StartNoNodegroupTimeout` ミリ秒待機してから、そのようなノードを `--nowait-nodes` オプションに渡されたリストに追加されたものとして扱い、起動します。デフォルト値は `15000` です (つまり、管理サーバーは 15 秒待機します)。このパラメータを `0` に設定すると、クラスタは無期限に待機します。

`StartNoNodegroupTimeout` はクラスタ内のすべてのデータノードで同じにする必要があります。そのため、これは個々のデータノードではなく、常に `config.ini` ファイルの `[ndbd default]` セクションに設定するようにしてください。

詳細は、[セクション23.5.7「NDB Cluster データノードのオンラインでの追加」](#)を参照してください。

- `HeartbeatIntervalDbDb`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	milliseconds
デフォルト	5000
範囲	10 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

障害の発生したノードを検出する主な方法の 1 つは、ハートビートを使用することです。このパラメータは、ハートビート信号の送信回数と予想される受信回数を指定します。ハートビートは無効にできません。

行に 4 つのハートビート間隔がない場合、ノードは `dead` と宣言されます。したがって、ハートビートメカニズムを使用して障害を検出する最大時間は、ハートビート間隔の 5 倍です。

デフォルトのハートビート間隔は 5000 ミリ秒 (5 秒) です。このパラメータを大幅に変更しないでください。また、ノード間の違いが大きくなるようにしてください。あるノードが 5000 ミリ秒を使用し、それを監視しているノードが 1000 ミリ秒を使用している場合、明らかにノードは非常に迅速に `dead` と宣言されます。このパラメータはオンラインのソフトウェアアップグレード中に変更できますが、少しずつ増やしてください。

[ネットワーク通信と待機時間](#) および `ConnectCheckIntervalDelay` 構成パラメータの説明も参照してください。

- `HeartbeatIntervalDbApi`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	milliseconds
デフォルト	1500
範囲	100 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

各データノードは各 MySQL サーバー (SQL ノード) にハートビート信号を送信して、接続されたままであるか確認します。MySQL サーバーが時間内にハートビートを送信できなかった場合、「デッド」と宣言されます。その場合、進行中のすべてのトランザクションが完了し、リソースが解放されます。以前の MySQL インスタンスによって開始されたすべてのアクティビティが完了するまで、SQL ノードは再接続できません。この判定に使用される 3 ハートビートの条件は、`HeartbeatIntervalDbDb` で説明したものと同じです。

デフォルトの間隔は 1500 ミリ秒 (1.5 秒) です。個々のデータノードはほかのすべてのデータノードと関係なく接続中の MySQL サーバーを監視するため、この間隔は各データノードで異なることがあります。

詳細は、[ネットワーク通信と待機時間](#)を参照してください。

- HeartbeatOrder

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	numeric
デフォルト	0
範囲	0 - 65535
再起動タイプ	S (NDB 8.0.13)

データノードは、各データノードが直前のデータノードをモニターする循環的な方法で、相互にハートビートを送信します。ハートビートが特定のデータノードによって検出されなかった場合、このノードは循環の直前のデータノードを「デッド」（つまり、クラスタからアクセスできなくなった）と宣言します。データノードがデッドであるという判定はグローバルに行われます。つまり、データノードがデッドと宣言されると、そのノードはクラスタ内のすべてのノードからそのようにみなされます。

異なるホストに配置されたデータノード間のハートビートが（たとえば、非常に長いハートビート間隔や一時的な接続の問題によって）ほかのノードペア間のハートビートに比べて遅すぎるために、データノードがまだクラスタの一部として機能しているにもかかわらず、デッドと宣言される可能性があります。

このような状況では、データノード間のハートビートの転送順序が、特定のデータノードがデッドと宣言されるかどうかに影響している可能性があります。この宣言が不必要に発生すると、それがノードグループの損失を招き、さらにクラスタの障害につながる可能性があります。

次の表に示すように、2台のホストコンピュータ `host1` および `host2` で実行されている4つのデータノード A、B、C、および D があり、これらのデータノードが2つのノードグループを構成しているセットアップについて考えます。

表 23.10 2台のホストコンピュータ `host1`、`host2` で実行されている4つのデータノード A、B、C、D。各データノードは、2つのノードグループのいずれかに属します。

ノードグループ	<code>host1</code> で実行されているノード	<code>host2</code> で実行されているノード
ノードグループ 0:	ノード A	ノード B
ノードグループ 1:	ノード C	ノード D

ハートビートが A->B->C->D->A の順に転送されるとします。この場合、ホスト間のハートビートが失われると、ノード B がノード A をデッドと宣言し、ノード C が B をデッドと宣言します。この結果、ノードグループ 0 が失われるため、クラスタが機能停止します。一方、転送の順序が A->B->D->C->A である（およびほかのすべての条件が前述のままである）場合は、ハートビートが失われると、ノード A および D がデッドと宣言されます。この場合、各ノードグループに1つずつノードが残っているため、クラスタは存続します。

`HeartbeatOrder` 構成パラメータは、ユーザーがハートビートの転送順序を構成できるようにします。`HeartbeatOrder` のデフォルト値は 0 です。すべてのデータノードでデフォルト値を使用できるようにすると、ハートビートの転送順序は NDB によって決定されます。このパラメータを使用する場合は、クラスタ内のすべてのデータノードで 0 以外の値（最大 65535）に設定する必要があります。また、この値は各データノードで一意である必要があります。これにより、ハートビートが `HeartbeatOrder` 値の小さいノードから大きいノードに順に転送される（その後、`HeartbeatOrder` がもっとも大きいノードからもっとも小さいノードに転送され、循環が完結する）ようになります。値は連続している必要はありません。たとえば、前述のシナリオでハートビート転送順序 A->B->D->C->A を強制するには、次に示すように `HeartbeatOrder` 値を設定します:

表 23.11 A->B->D->C->A のハートビート遷移順序を強制する `HeartbeatOrder` 値。

ノード	<code>HeartbeatOrder</code> 値
A	10
B	20
C	30

ノード	HeartbeatOrder 値
D	25

このパラメータを使用して実行中の NDB Cluster 内のハートビート転送順序を変更するには、まずグローバル構成 (config.ini) ファイル内のクラスタ内のデータノードごとに `HeartbeatOrder` を設定する必要があります。変更を有効にするには、次のいずれかを実行する必要があります。

- クラスタ全体の完全なシャットダウンと再起動。
- 2 回連続のクラスタのローリング再起動。両方のローリング再起動で、すべてのノードが同じ順序で再起動される必要があります。

`DUMP 908` を使用すると、データノードログでこのパラメータの効果を観察できます。

• `ConnectCheckIntervalDelay`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	milliseconds
デフォルト	0
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

このパラメータは、データノードのいずれかがハートビートチェックに失敗したあと、最大 `HeartbeatIntervalDbDb` ミリ秒の 5 つの間隔でデータノード間の接続チェックを有効にします。

`ConnectCheckIntervalDelay` ミリ秒の間隔内でさらに応答に失敗したこのようなデータノードは疑わしいとみなされ、このような間隔が 2 回連続くとデッドとみなされます。これは、待機時間の既知の問題がある設定で役立ちます。

このパラメータのデフォルト値は 0 (無効) です。

• `TimeBetweenLocalCheckpoints`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	number of 4-byte words, as base-2 logarithm
デフォルト	20
範囲	0 - 31
再起動タイプ	N (NDB 8.0.13)

このパラメータは、新しいローカルポイントを開始するまでの待機時間を指定しないという点で例外です。どちらかといえば、比較的少数の更新が行われるクラスタ内でローカルチェックポイントが実行されないようにするために使用されます。更新回数が多いほとんどのクラスタでは、直前のローカルチェックポイントが完了した直後に新しいローカルチェックポイントが開始される可能性があります。

前のローカルチェックポイントの開始以降に実行されたすべての書き込み操作のサイズが追加されます。このパラメータは、4 バイトワードの数の底 2 の対数として指定される点でも例外的です。したがって、デフォルト値の 20 は 4M バイト ($4 * 2^{20}$) の書き込み操作を意味し、21 は 8M バイトを意味し、最大値の 31 は 8G バイトの書き込み操作と同等です。

クラスタ内のすべての書き込み操作がまとめて追加されます。`TimeBetweenLocalCheckpoints` を 6 以下に設定すると、クラスタのワークロードに関係なく、ローカルチェックポイントが一時停止せずに継続的に実行されます。

• `TimeBetweenGlobalCheckpoints`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	milliseconds

デフォルト	2000
範囲	20 - 32000
再起動タイプ	N (NDB 8.0.13)

トランザクションがコミットされると、データがミラー化されているノードのメインメモリーでコミットされません。ただし、トランザクションログレコードはコミットの一部としてディスクにフラッシュされません。トランザクションを少なくとも 2 台の自立的なホストマシン上で安全にコミットすることで、持続性に関する妥当な基準が満たされるはずだということがこの動作の根拠です。

また、最悪のケース (クラスタの完全なクラッシュ) も適切に処理されるようにすることが重要です。この実行を保証するために、特定の間隔以内に行われるすべてのトランザクションはグローバルチェックポイントに取り込まれます。これは、ディスクにフラッシュされたコミット済みのトランザクションとみなすことができます。つまり、トランザクションはコミットプロセスの一部としてグローバルチェックポイントグループに組み込まれます。その後、このグループのログレコードがディスクにフラッシュされると、トランザクションのグループ全体がクラスタ内のコンピュータ上のディスクに安全にコミットされます。

NDB 8.0.19 以降では、この値を減らす「ディスクデータ」テーブルでソリッドステートディスク (特に NVMe を採用しているディスク) を使用する場合に推奨されます。このような場合は、[MaxDiskDataLatency](#) が適切なレベルに設定されていることも確認する必要があります。

このパラメータは、グローバルチェックポイント間の間隔を定義します。デフォルトは 2000 ミリ秒です。

- [TimeBetweenGlobalCheckpointsTimeout](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	milliseconds
デフォルト	120000
範囲	10 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

このパラメータは、グローバルチェックポイント間の最小タイムアウトを定義します。デフォルトは 120000 ミリ秒です。

- [TimeBetweenEpochs](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	milliseconds
デフォルト	100
範囲	0 - 32000
再起動タイプ	N (NDB 8.0.13)

このパラメータは、NDB Cluster レプリケーションの同期エポック間の間隔を定義します。デフォルト値は 100 ミリ秒です。

[TimeBetweenEpochs](#) は、NDB Cluster レプリケーションのパフォーマンスを向上させるために使用できる「micro-GCPs」の実装の一部です。

- [TimeBetweenEpochsTimeout](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	milliseconds
デフォルト	0
範囲	0 - 256000

再起動タイプ	N (NDB 8.0.13)
--------	----------------

このパラメータは、NDB Cluster レプリケーションの同期エポックのタイムアウトを定義します。このパラメータで決定された時間内にノードがグローバルチェックポイントに参加できなかった場合、ノードはシャットダウンされます。デフォルト値は 0 です。つまり、タイムアウトは無効になります。

`TimeBetweenEpochsTimeout` は、NDB Cluster レプリケーションのパフォーマンスを向上させるために使用できる「micro-GCPs」の実装の一部です。

GCP 保存に 1 分より長い時間がかかるか、GCP コミットに 10 秒より長い時間がかかると、このパラメータの現在の値および警告がクラスタログに書き込まれます。

このパラメータを 0 に設定すると、保存のタイムアウト、コミットのタイムアウト、またはその両方によって発生する GCP の停止を無効にします。このパラメータに指定できる最大値は 256000 ミリ秒です。

- `MaxBufferedEpochs`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	epochs
デフォルト	100
範囲	0 - 100000
再起動タイプ	N (NDB 8.0.13)

サブスクリブするノードが遅延できる未処理のエポック数。この数を超えると、遅延するサブスクリバが切断されます。

ほとんどの通常の操作では、デフォルト値の 100 で十分です。サブスクリブするノードの遅延によって切断が発生する場合、その原因は通常、プロセスまたはスレッドに関するネットワークまたはスケジューリングの問題です。(まれな状況ですが、NDB クライアントのバグによってこの問題が起きることもあります。) エポックが大きいつきは、この値をデフォルトより小さく設定するのが望ましい場合もあります。

切断することで、クライアントの問題によってデータノードサービスが影響を受け、データをバッファリングするためのメモリーが不足し、最終的にシャットダウンすることはなくなります。代わりに、切断の結果として(たとえば、バイナリログのギャップイベントによって)クライアントが影響を受け、クライアントは強制的にプロセスを再接続または再起動します。

- `MaxBufferedEpochBytes`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	bytes
デフォルト	26214400
範囲	26214400 (0x01900000) - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

このノードがエポックをバッファリングするために割り当てるバイトの合計数。

- `TimeBetweenInactiveTransactionAbortCheck`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	milliseconds
デフォルト	1000
範囲	1000 - 4294967039 (0xFFFFFFFF)

再起動タイプ	N (NDB 8.0.13)
--------	----------------

タイムアウトの処理は、各トランザクションのタイマーをこのパラメータで指定された間隔ごとに 1 回チェックして実行されます。したがって、このパラメータが 1000 ミリ秒に設定されている場合、すべてのトランザクションのタイムアウトが毎秒 1 回チェックされます。

デフォルト値は 1000 ミリ秒 (1 秒) です。

- [TransactionInactiveTimeout](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	milliseconds
デフォルト	4294967039 (0xFFFFFFFF)
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

このパラメータは、トランザクションが中止されるまでに、同じトランザクション内の操作間で経過が許容される最大時間を指定します。

このパラメータのデフォルトは 4G です (最大も同じです)。トランザクションがロックを保持する期間が長すぎないようにする必要のあるリアルタイムデータベースでは、このパラメータを比較的小さい値に設定するようにしてください。0 に設定すると、アプリケーションはタイムアウトしません。単位はミリ秒です。

- [TransactionDeadlockDetectionTimeout](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	milliseconds
デフォルト	1200
範囲	50 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

ノードは、トランザクションを含むクエリーの実行時に、クラスタ内のほかのノードが応答するのを待機してから処理を続行します。このパラメータは、トランザクションがデータノード内での実行に費やすことができる時間、つまり、トランザクションに参加している各データノードがリクエストを実行するまでトランザクションコーディネータが待機する時間を設定します。

応答の失敗は、次のいずれかの理由で発生します。

- ノードが「デッド」です
- 操作がロックキューに入りました
- アクションの実行を要求したノードに非常に高い負荷がかかっている可能性があります。

このタイムアウトパラメータは、トランザクションが中止されるまでにトランザクションコーディネータが別のノードによるクエリーの実行を待機する時間を指定するもので、ノード障害の処理とデッドロックの検出において重要です。

デフォルトのタイムアウト値は 1200 ミリ秒 (1.2 秒) です。

このパラメータの最小は 50 ミリ秒です。

- [DiskSyncSize](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	bytes
デフォルト	4M

範囲	32K - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

これは、ローカルチェックポイントファイルにデータをフラッシュするまでに格納される最大バイト数です。これは、パフォーマンスを大幅に低下させる可能性がある書き込みバッファリングを禁止するために行われます。このパラメータは、[TimeBetweenLocalCheckpoints](#) の代わりに使用するものではありません。

注記

[ODirect](#) が有効になっている場合は、[DiskSyncSize](#) を設定する必要はありません。実際、そのような場合は、その値が無視されます。

デフォルト値は 4M (4 メガバイト) です。

- [MaxDiskWriteSpeed](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	numeric
デフォルト	20M
範囲	1M - 1024G
再起動タイプ	S (NDB 8.0.13)

この NDB Cluster で (このデータノードまたはその他のデータノードによって) 再起動が行われない場合の、ローカルチェックポイントおよびバックアップ操作によるディスクへの書き込みの最大速度 (バイト/秒) を設定します。

このデータノードの再起動中に許可されるディスク書き込みの最大速度を設定するには、[MaxDiskWriteSpeedOwnRestart](#) を使用します。ほかのデータノードの再起動中に許可されるディスク書き込みの最大速度を設定するには、[MaxDiskWriteSpeedOtherNodeRestart](#) を使用します。すべての LCP およびバックアップ操作によるディスク書き込みの最小速度を調整するには、[MinDiskWriteSpeed](#) を設定します。

- [MaxDiskWriteSpeedOtherNodeRestart](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	numeric
デフォルト	50M
範囲	1M - 1024G
再起動タイプ	S (NDB 8.0.13)

この NDB Cluster 内の 1 つ以上のデータノードがこのノード以外で再起動されているときの、ローカルチェックポイントおよびバックアップ操作によるディスクへの書き込みの最大速度をバイト/秒単位で設定します。

このデータノードの再起動中に許可されるディスク書き込みの最大速度を設定するには、[MaxDiskWriteSpeedOwnRestart](#) を使用します。クラスタ内ではデータノードが再起動されていない場合に許可されるディスク書き込みの最大速度を設定するには、[MaxDiskWriteSpeed](#) を使用します。すべての LCP およびバックアップ操作によるディスク書き込みの最小速度を調整するには、[MinDiskWriteSpeed](#) を設定します。

- [MaxDiskWriteSpeedOwnRestart](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	numeric
デフォルト	200M
範囲	1M - 1024G

再起動タイプ	S (NDB 8.0.13)
--------	----------------

このデータノードの再起動時の、ローカルチェックポイントおよびバックアップ操作によるディスク書き込みの最大速度 (バイト/秒) を設定します。

ほかのデータノードの再起動中に許可されるディスク書き込みの最大速度を設定するには、[MaxDiskWriteSpeedOtherNodeRestart](#) を使用します。クラスタ内ではデータノードが再起動されていない場合に許可されるディスク書き込みの最大速度を設定するには、[MaxDiskWriteSpeed](#) を使用します。すべての LCP およびバックアップ操作によるディスク書き込みの最小速度を調整するには、[MinDiskWriteSpeed](#) を設定します。

- [MinDiskWriteSpeed](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	numeric
デフォルト	10M
範囲	1M - 1024G
再起動タイプ	S (NDB 8.0.13)

ローカルチェックポイントおよびバックアップ操作によるディスク書き込みの最小速度 (バイト/秒) を設定します。

さまざまな条件下で LCP およびバックアップに許可されるディスク書き込みの最大速度は、パラメータ [MaxDiskWriteSpeed](#)、[MaxDiskWriteSpeedOwnRestart](#)、および [MaxDiskWriteSpeedOtherNodeRestart](#) を使用して調整できます。詳細は、これらのパラメータの説明を参照してください。

- [ArbitrationTimeout](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	milliseconds
デフォルト	7500
範囲	10 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

このパラメータは、データノードがアービトレーションメッセージに対するアービレータからの応答を待機する時間を設定します。これを超えた場合は、ネットワークが切断されたとみなされます。

デフォルト値は 7500 ミリ秒 (7.5 秒) です。

- [Arbitration](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	enumeration
デフォルト	Default
範囲	Default, Disabled, WaitExternal
再起動タイプ	N (NDB 8.0.13)

[Arbitration](#) パラメータを使用して、このパラメータに指定できる 3 つの値のいずれかに対応するアービトレーションスキームを選択できます。

- デフォルト. この場合は、管理および API ノードの [ArbitrationRank](#) 設定で指定されるとおりに、アービトレーションが正常に進行します。これはデフォルト値です。
- Disabled. `config.ini` ファイルの `[ndbd default]` セクションに `Arbitration = Disabled` を設定すると、すべての管理および API ノードで [ArbitrationRank](#) の 0 を設定した場合と同じ結果が得られます。 [Arbitration](#) をこのように設定すると、[ArbitrationRank](#) の設定はすべて無視されます。

- **WaitExternal.** [Arbitration](#) パラメータを使用して、クラスタが内部でアービトレーションを処理する代わりに [ArbitrationTimeout](#) で指定された時間が経過するまで外部のクラスタマネージャアプリケーションによるアービトレーションの実行を待機する方法で、アービトレーションを構成することもできます。これを行うには、[config.ini](#) ファイルの [\[ndbd default\]](#) セクションに [Arbitration = WaitExternal](#) を設定します。[WaitExternal](#) の設定で最良の結果を得るためには、[ArbitrationTimeout](#) を外部クラスタマネージャがアービトレーションを実行するのに必要な間隔の 2 倍に設定することをお勧めします。

重要

このパラメータは、クラスタ構成ファイルの [\[ndbd default\]](#) セクションでのみ使用するようになっています。[Arbitration](#) を個々のデータノードで異なる値に設定すると、クラスタの動作は不特定になります。

- [RestartSubscriberConnectTimeout](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	ms
デフォルト	12000
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	

このパラメータは、サブスクライブしている API ノードの接続をデータノードが待機する時間を決定します。このタイムアウトに達すると、「欠落」API ノードはクラスタから切断されます。このタイムアウトを無効にするには、[RestartSubscriberConnectTimeout](#) を 0 に設定します。

このパラメータはミリ秒単位で指定されますが、タイムアウト自体は次に作成される秒単位に解決されます。

バッファリングとロギング. 上級ユーザーは、いくつかの [\[ndbd\]](#) 構成パラメータを使用することで、ノードプロセスが使用するリソースをより細かく制御したり、必要に応じてさまざまなバッファサイズを調整したりできます。

これらのバッファは、ログレコードをディスクに書き込む際に、ファイルシステムに対するフロントエンドとして使用されます。ノードがディスクレスモードで実行されている場合は、NDB ストレージエンジンのファイルシステム抽象化レイヤーによってディスク書き込みが「偽装」されるため、ペナルティーなしでこれらのパラメータを最小値に設定できます。

- [UndoIndexBuffer](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	unsigned
デフォルト	2M
範囲	1M - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

このパラメータでサイズが設定される Undo インデックスバッファは、ローカルチェックポイント中に使用されます。NDB ストレージエンジンは、使用可能な Redo ログとともに、チェックポイントの一貫性に基づくリカバリスキームを使用します。システム全体の書き込みをブロックせずに一貫性のあるチェックポイントを作成するため、ローカルチェックポイントの実行中に Undo ロギングが行われます。Undo ロギングは、一度に 1 つのテーブルフラグメントに対してアクティブ化されます。この最適化が可能なのは、テーブルが完全にメインメモリーに格納されているためです。

Undo インデックスバッファは、主キーのハッシュインデックスの更新で使用されます。挿入と削除では、ハッシュインデックスが再編成されます。NDB ストレージエンジンは、すべての物理的な変更をシステムの再起動時に取り消すことができるように、変更を 1 つのインデックスページにマップする Undo ログレコードを書き込みます。また、ローカルチェックポイントの開始時に、各フラグメントのアクティブな挿入操作も記録します。

読み取りと更新では、ロックビットが設定され、ハッシュインデックスエントリのヘッダーが更新されます。これらの変更はページ書き込みアルゴリズムによって処理されるため、これらの操作に Undo ロギングは必要ありません。

このバッファはデフォルトで 2M バイトです。最小値は 1M バイトです。ほとんどのアプリケーションではこれで十分です。大規模なトランザクションや大規模な主キーとともに非常に大規模または多数の挿入および削除を実行するアプリケーションでは、このバッファのサイズを増やす必要がある場合があります。このバッファが小さすぎる場合、NDB ストレージエンジンは内部エラーコード 677 「[Index UNDO buffers overloaded](#)」を発行します。

重要

ローリング再起動中にこのパラメータの値を減らすのは安全ではありません。

• UndoDataBuffer

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	unsigned
デフォルト	16M
範囲	1M - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

このパラメータは、Undo データバッファのサイズを設定します。Undo データバッファは、Undo インデックスバッファとほぼ同じ機能を実行しますが、インデックスメモリーではなくデータメモリーに関して使用されます。このバッファは、フラグメントのローカルチェックポイントフェーズで挿入、削除、および更新のために使用されます。

Undo ログエントリは記録される操作が増えるにつれて大きくなる傾向があるため、このバッファも対応するインデックスメモリーより大きくなります (デフォルト値は 16M バイトです)。

一部のアプリケーションでは、このメモリー量が不必要に大きい場合があります。そのような場合は、このサイズを最小の 1M バイトに減らすことができます。

ほとんどの場合、このバッファのサイズを増やす必要はありません。そのような必要がある場合は、データベースの更新アクティビティーによって発生する負荷をディスクが実際に処理できているかどうかをチェックすることをお勧めします。このバッファのサイズを増やしても、ディスクスペースの不足を解消することはできません。

このバッファが小さすぎて過密状態になった場合、NDB ストレージエンジンは次の内部エラーコードを発行します: 891 ([Data UNDO buffers overloaded](#))。

重要

ローリング再起動中にこのパラメータの値を減らすのは安全ではありません。

• RedoBuffer

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	bytes
デフォルト	32M
範囲	1M - 4294967039 (0xFFFFFFFF)

再起動タイプ	N (NDB 8.0.13)
--------	----------------

更新アクティビティーも、すべて記録する必要があります。Redo ログにより、システムが再起動されるたびにこれらの更新を再現できるようになります。NDB のリカバリアルゴリズムは、Undo ログとともにデータの「フアジー」チェックポイントを使用し、Redo ログを適用してリストアポイントまでのすべての変更を再現します。

`RedoBuffer` は、Redo ログが書き込まれるバッファのサイズを設定します。デフォルト値は 32M バイトです。最小値は 1M バイトです。

このバッファが小さすぎる場合、NDBストレージエンジンは内部エラーコード 1221 (`REDO log buffers overloaded`)を発行します。このため、クラスタ構成のオンライン変更の一部として `RedoBuffer` の値を減らそうとする場合は注意してください。

`ndbmt` は、LDM スレッドごとに別個のバッファを割り当てます (`ThreadConfig` を参照してください)。たとえば、4 つの LDM スレッドがある場合、`ndbmt` データノードには実際には 4 つのバッファがあり、それぞれに `RedoBuffer` バイト (合計で $4 * \text{RedoBuffer}$ バイト) が割り当てられます。

- `EventLogBufferSize`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	bytes
デフォルト	8192
範囲	0 - 64K
再起動タイプ	S (NDB 8.0.13)

データノード内部の NDB ログイベントに使用される循環バッファのサイズを制御します。

ログメッセージの制御。クラスタの管理では、`stdout` に送信されるさまざまなイベントタイプのログメッセージ数を制御することが非常に重要です。イベントカテゴリごとに、16 個の設定可能なイベントレベル (番号 0-15) があります。特定のイベントカテゴリのイベントレポートをレベル 15 に設定すると、そのカテゴリのすべてのイベントレポートが `stdout` に送信されます。0 に設定すると、そのカテゴリのイベントレポートは作成されません。

デフォルトでは、起動メッセージのみが `stdout` に送信され、残りのイベントレポートのレベルはデフォルトの 0 に設定されます。これは、これらのメッセージが管理サーバーのクラスタログにも送信されるためです。

管理クライアントで同じようなレベルのセットを設定すると、クラスタログに記録するイベントのレベルを指定できます。

- `LogLevelStartup`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	1
範囲	0 - 15
再起動タイプ	N (NDB 8.0.13)

プロセスの起動中に生成されるイベントのレポートレベル。

デフォルトのレベルは 1 です。

- `LogLevelShutdown`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	0
範囲	0 - 15
再起動タイプ	N (NDB 8.0.13)

ノードの正常なシャットダウンの一部として生成されるイベントのレポートレベル。

デフォルトのレベルは 0 です。

- [LogLevelStatistic](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	0
範囲	0 - 15
再起動タイプ	N (NDB 8.0.13)

主キー読み取りの数、更新の数、挿入の数、バッファーの使用状況に関する情報などの統計イベントのレポートレベル。

デフォルトのレベルは 0 です。

- [LogLevelCheckpoint](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	log level
デフォルト	0
範囲	0 - 15
再起動タイプ	N (NDB 8.0.13)

ローカルおよびグローバルチェックポイントによって生成されるイベントのレポートレベル。

デフォルトのレベルは 0 です。

- [LogLevelNodeRestart](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	0
範囲	0 - 15
再起動タイプ	N (NDB 8.0.13)

ノード再起動中に生成されるイベントのレポートレベル。

デフォルトのレベルは 0 です。

- [LogLevelConnection](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	0
範囲	0 - 15
再起動タイプ	N (NDB 8.0.13)

クラスタノード間の接続によって生成されるイベントのレポートレベル。

デフォルトのレベルは 0 です。

- [LogLevelError](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	0
範囲	0 - 15
再起動タイプ	N (NDB 8.0.13)

クラスタ全体のエラーおよび警告によって生成されるイベントのレポートレベル。これらのエラーは、ノードの障害の原因にはなりませんが、レポートする価値はあると考えられます。

デフォルトのレベルは 0 です。

- [LogLevelCongestion](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	level
デフォルト	0
範囲	0 - 15
再起動タイプ	N (NDB 8.0.13)

輻輳によって生成されるイベントのレポートレベル。これらのエラーは、ノードの障害の原因にはなりませんが、レポートする価値はあると考えられます。

デフォルトのレベルは 0 です。

- [LogLevelInfo](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	0
範囲	0 - 15
再起動タイプ	N (NDB 8.0.13)

クラスタの一般的な状態に関する情報として生成されるイベントのレポートレベル。

デフォルトのレベルは 0 です。

- [MemReportFrequency](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	unsigned
デフォルト	0
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

このパラメータは、データノードのメモリ使用状況レポートをクラスタログに記録する頻度を制御します。これは、レポート間の秒数を表す整数値です。

各データノードのデータメモリとインデックスメモリの使用量は、[config.ini](#) ファイルに設定されている [DataMemory](#) のパーセンテージと 32 KB ページの両方として記録されます。たとえば、[DataMemory](#) が 100M バイ

トであり、特定のデータノードがデータメモリのストレージとして 50M バイトを使用している場合、クラスタログの対応する行はこのようになります。

```
2006-12-24 01:18:16 [MgmSrvr] INFO -- Node 2: Data usage is 50%(1280 32K pages of total 2560)
```

`MemReportFrequency` は必須のパラメータではありません。使用する場合は、`config.ini` の `[ndbd default]` セクションですべてのクラスタデータノード用に設定することも、構成ファイルの対応する `[ndbd]` セクションで個々のデータノード用に設定またはオーバーライドすることもできます。最小値 (デフォルト値も同じ) は 0 です。この場合は、[セクション23.5.3.2「NDB Cluster ログイベント」](#)の統計イベントの説明で指摘したように、メモリー使用量が特定の割合 (80%、90%、および 100%) に達したときにのみ、メモリーのレポートが記録されます。

- [StartupStatusReportFrequency](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	seconds
デフォルト	0
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

`--initial` を指定してデータノードを起動すると、起動フェーズ 4 ([セクション23.5.4「NDB Cluster 起動フェーズのサマリー」](#)を参照してください) で Redo ログファイルが初期化されます。

`NoOfFragmentLogFiles`、`FragmentLogFileSize`、またはその両方に非常に大きな値を設定すると、この初期化に長い時間がかかることがあります。`StartupStatusReportFrequency` 構成パラメータを使用して、このプロセスの進行に関するレポートを定期的に記録するよう強制できます。この場合、ここに示すように、初期化されたファイルの数とスペースの量に関する進行状況がクラスタログにレポートされます。

```
2009-06-20 16:39:23 [MgmSrvr] INFO -- Node 1: Local redo log file initialization status:
#Total files: 80, Completed: 60
#Total MBytes: 20480, Completed: 15557
2009-06-20 16:39:23 [MgmSrvr] INFO -- Node 2: Local redo log file initialization status:
#Total files: 80, Completed: 60
#Total MBytes: 20480, Completed: 15570
```

これらのレポートは、起動フェーズ 4 で `StartupStatusReportFrequency` 秒ごとに記録されます。`StartupStatusReportFrequency` が 0 (デフォルト) の場合は、Redo ログファイルの初期化プロセスの開始時と完了時にのみ、レポートがクラスタログに書き込まれます。

データノードのデバッグパラメータ

次のパラメータは、データノードのテストまたはデバッグ中に使用することを目的としており、本番では使用しません。

- [DictTrace](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	bytes
デフォルト	undefined
範囲	0 - 100
再起動タイプ	N (NDB 8.0.13)

`DictTrace` を使用してテーブルを作成および削除することで、生成されたイベントのトレースをロギングできます。このパラメータは、NDB のカーネルコードをデバッグする場合にのみ役立ちます。`DictTrace` は整数値を取ります。デフォルトは 0 で、ロギングは実行されません。1 はトレースロギングを有効にし、2 は追加の `DBDICT` デバッグ出力のロギングを有効にします。

- [WatchdogImmediateKill](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	boolean

デフォルト	false
範囲	true, false
再起動タイプ	

WatchdogImmediateKill データノード構成パラメータを有効にすると、ウォッチドッグの問題が発生するたびにスレッドがただちに強制終了されるようになります。このパラメータは、デバッグまたはトラブルシューティング時のみ使用し、実行が中止された瞬間の発生を正確にレポートするトレースファイルを取得します。

バックアップパラメータ。このセクションで説明する `[ndbd]` パラメータは、オンラインバックアップの実行用に確保されるメモリーバッファを定義します。

- **BackupDataBufferSize**

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	bytes
デフォルト	16M
範囲	512K - 4294967039 (0xFFFFFFFF)
非推奨	Yes (in NDB 7.6)
再起動タイプ	N (NDB 8.0.13)

バックアップの作成では、データをディスクに転送するために使用される 2 つのバッファがあります。バックアップデータバッファは、ノードのテーブルをスキャンして記録されるデータを書き込むために使用されます。このバッファへの書き込みが `BackupWriteSize` で指定されたレベルに達すると、ページがディスクに転送されます。バックアッププロセスでは、データをディスクにフラッシュしながら、スペースがなくなるまでこのバッファに書き込み続けることができます。これが発生すると、バックアッププロセスでスキャンを一時停止し、ディスク書き込みがある程度完了してメモリーが解放され、スキャンを続行できるようになるまで待機します。

このパラメータのデフォルト値は 16M バイトです。最小値は 512K です。

- **BackupDiskWriteSpeedPct**

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	percent
デフォルト	50
範囲	0 - 90
再起動タイプ	N (NDB 8.0.13)

通常の操作中、データノードはローカルチェックポイントおよびバックアップに使用されるディスク書き込み速度を最大化しようとしますが、`MinDiskWriteSpeed` および `MaxDiskWriteSpeed` によって設定された範囲内に残ります。ディスク書き込みスロットルにより、各 LDM スレッドに合計予算の均等なシェアが与えられます。これにより、ディスク I/O 予算を超えずにパラレル LCP を実行できます。バックアップは 1 つの LDM スレッドによってのみ実行されるため、事実上、これによって予算の停止が発生し、バックアップ完了時間が長くなります。また、バックアップログバッファの充填率が達成可能な書き込み率よりも高い場合に、変更率が十分に高くなってバックアップを完了できない場合です。

この問題に対処するには、LCP の LDM スレッド間で残りの予算を共有する前に予約されているノード最大書き込みレート予算の割合として解釈される、0-90 (含む) の範囲の値をとる `BackupDiskWriteSpeedPct` 構成パラメータを使用します。バックアップを実行している LDM スレッドは、バックアップの書き込みレート予算全体と、ローカルチェックポイントの書き込みレート予算の (削減された) シェアを受け取ります。

このパラメータのデフォルト値は 50 (50% として解釈) です。

- **BackupLogBufferSize**

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	bytes

デフォルト	16M
範囲	2M - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

バックアップログバッファはバックアップデータバッファと同じような役割を果たしますが、バックアップの実行中に行われたテーブル書き込みのログを生成するために使用されます。これらのページの書き込みにはバックアップデータバッファと同じ原則が適用されますが、バックアップログバッファのスペースがなくなると、バックアップは失敗します。そのため、バックアップログバッファのサイズは、バックアップ実行中の書き込みアクティビティによって発生する負荷を処理するのに十分な大きさである必要があります。[セクション 23.5.8.3 「NDB Cluster バックアップの構成」](#)を参照してください。

ほとんどのアプリケーションでは、このパラメータのデフォルト値で十分です。実際、バックアップログバッファがいっぱいになった場合より、ディスク書き込みの速度が不十分な場合の方が、バックアップが失敗する可能性は高くなります。ディスクサブシステムがアプリケーションから発生する書き込み負荷に合わせて構成されていない場合は、クラスタが必要な操作を実行できない可能性が高くなります。

ディスクやネットワーク接続よりもプロセッサがボトルネックになるような方法でクラスタノードを構成することをお勧めします。

このパラメータのデフォルト値は 16M バイトです。

- [BackupMemory](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	bytes
デフォルト	32M
範囲	0 - 4294967039 (0xFFFFFFFF)
非推奨	Yes (in NDB 7.4)
再起動タイプ	N (NDB 8.0.13)

このパラメータは非推奨であり、NDB Cluster の将来のバージョンで削除される可能性があります。設定はすべて無視されます。

- [BackupReportFrequency](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	seconds
デフォルト	0
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

このパラメータは、バックアップ中に管理クライアントでバックアップステータスレポートを発行する頻度と、そのようなレポートをクラスタログに書き込む頻度を制御します (クラスタイベントロギングで許可するように構成されている場合。[ロギングとチェックポイント](#)を参照してください)。[BackupReportFrequency](#) は、バックアップステータスレポート間の時間 (秒単位) を表します。

デフォルト値は 0 です。

- [BackupWriteSize](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	bytes
デフォルト	256K
範囲	32K - 4294967039 (0xFFFFFFFF)

非推奨	Yes (in NDB 7.6)
再起動タイプ	N (NDB 8.0.13)

このパラメータは、バックアップログおよびバックアップデータバッファによってディスクに書き込まれるメッセージのデフォルトサイズを指定します。

このパラメータのデフォルト値は 256K バイトです。

- [BackupMaxWriteSize](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	bytes
デフォルト	1M
範囲	256K - 4294967039 (0xFFFFFFFF)
非推奨	Yes (in NDB 7.6)
再起動タイプ	N (NDB 8.0.13)

このパラメータは、バックアップログおよびバックアップデータバッファによってディスクに書き込まれるメッセージの最大サイズを指定します。

このパラメータのデフォルト値は 1M バイトです。

- [CompressedBackup](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	boolean
デフォルト	false
範囲	true, false
再起動タイプ	N (NDB 8.0.13)

このパラメータを有効にすると、バックアップファイルが圧縮されます。使用される圧縮は `gzip --fast` と同等であり、データノードで非圧縮バックアップファイルの格納に必要なスペースの 50% 以上を節約できます。圧縮バックアップは、個々のデータノードまたは (`config.ini` ファイルの `[ndbd default]` セクションにこのパラメータを設定することで) すべてのデータノードで有効化できます。

重要

圧縮バックアップを、この機能をサポートしない MySQL バージョンを実行するクラスタにリストアすることはできません。

デフォルト値は 0 (無効) です。

- [RequireEncryptedBackup](#)

バージョン (またはそれ以降)	NDB 8.0.22
タイプまたは単位	integer
デフォルト	0
範囲	0 - 1
追加	NDB 8.0.22
再起動タイプ	N (NDB 8.0.13)

1 に設定した場合、バックアップを暗号化する必要があります。このパラメータはデータノードごとに個別に設定できますが、`config.ini` グローバル構成ファイルの `[ndbd default]` セクションで設定することをお勧めします。暗号

化バックアップの実行の詳細は、[セクション23.5.8.2「NDB Cluster 管理クライアントを使用したバックアップの作成」](#)を参照してください。

NDB 8.0.22 に追加されました。

注記

バックアップファイルの場所は、`BackupDataDir` データノード構成パラメータによって決まります。

追加要件. これらのパラメータを指定するときは、次の関係が有効である必要があります。 そうしないと、データノードを起動できません。

- `BackupDataBufferSize >= BackupWriteSize + 188K` バイト
- `BackupLogBufferSize >= BackupWriteSize + 16K` バイト
- `BackupMaxWriteSize >= BackupWriteSize`

NDB Cluster のリアルタイムパフォーマンスパラメータ

このセクションで説明する `[ndbd]` パラメータは、マルチプロセッサのデータノードホスト上の特定の CPU に対するスレッドのスケジューリングとロックで使用されます。

注記

これらのパラメータを使用するには、システムの root としてデータノードプロセスを実行する必要があります。

- [BuildIndexThreads](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	numeric
デフォルト	128
範囲	0 - 128
再起動タイプ	

このパラメータは、システムまたはノードの起動中に順序付けされたインデックスの再構築時、および `ndb_restore --rebuild-indexes` の実行時に作成するスレッドの数を指定します。これは、データノードごとにテーブルに複数のフラグメントがある場合 (たとえば、`COMMENT="NDB_TABLE=PARTITION_BALANCE=FOR_RA_BY_LDM_X_2"` が `CREATE TABLE` とともに使用されている場合) にのみサポートされます。

このパラメータを 0 (デフォルト) に設定すると、順序付けられたインデックスのマルチスレッド作成が無効になります。

このパラメータは、`ndbd` または `ndbmtid` を使用するときをサポートされます。

`TwoPassInitialNodeRestartCopy` データノード構成パラメータを `TRUE` に設定することで、データノードの初期再起動時にマルチスレッドビルドを有効にできます。

- [LockExecuteThreadToCPU](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	set of CPU IDs
デフォルト	0
範囲	...
再起動タイプ	N (NDB 8.0.13)

`ndbd` で使用する場合、このパラメータ (現在は文字列) は `NDBCLUSTER` の実行スレッドを処理するために割り当てられる CPU の ID を指定します。 `ndbmtid` で使用する場合、このパラメータの値は実行スレッドを処理するために割り当てられる CPU ID のカンマ区切りリストです。 リスト内の各 CPU ID は、0-65535 の (これらを含む) 範囲の整数にします。

指定する ID の数は、 `MaxNoOfExecutionThreads` によって指定される実行スレッドの数と一致するようにしてください。 ただし、このパラメータを使用したときに、特定の順序でスレッドが CPU に割り当てられる保証はありません。 `ThreadConfig` を使用すると、このようなより細かい制御が可能になります。

`LockExecuteThreadToCPU` にデフォルト値はありません。

- `LockMaintThreadsToCPU`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	CPU ID
デフォルト	0
範囲	0 - 64K
再起動タイプ	N (NDB 8.0.13)

このパラメータは、 `NDBCLUSTER` の管理スレッドを処理するために割り当てられる CPU の ID を指定します。

このパラメータの値は、0-65535 の (これらを含む) 範囲の整数です。 デフォルト値はありません。

- `Numa`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	numeric
デフォルト	1
範囲	...
再起動タイプ	N (NDB 8.0.13)

このパラメータは、Non-Uniform Memory Access (NUMA) がオペレーティングシステムまたはデータノードプロセスのどちらによって制御されるかを決定します (データノードで `ndbd` と `ndbmtid` のどちらを使用するか)。 デフォルトでは、 `NDB` は、ホストオペレーティングシステムが NUMA サポートを提供する任意のデータノードでインターリーブ NUMA メモリー割当てポリシーを使用しようとします。

`Numa = 0` を設定することは、 `datanode` プロセス自体がメモリー割当てのポリシーを設定しようとせず、この動作をオペレーティングシステムによって決定できることを意味します。 これは、別の `numactl` ツールによってさらにガイドされる場合があります。 つまり、 `Numa = 0` ではシステムのデフォルト動作が生成されますが、これは `numactl` でカスタマイズできます。 多くの Linux システムでは、システムのデフォルトの動作は、ソケットローカルメモリーを割当て時に任意のプロセスに割り当てることです。 これは、 `ndbmtid` の使用時に問題になる可能性があります。 これは、特にメインメモリー内のページをロックするときに、 `ndbmtid` が起動時にすべてのメモリーを割り当てるため、不均衡になり、ソケットごとに異なるアクセス速度が与えられるためです。

`Numa = 1` を設定することは、データノードプロセスが `libnuma` を使用してインターリーブされたメモリー割当てを要求することを意味します。 (これは、 `numactl` を使用して、オペレーティングシステムレベルで手動で実行することもできます。) インターリーブ割当てを有効に使用すると、データノードプロセスは不均一なメモリーアクセスを無視するように指示されますが、高速なローカルメモリーを利用しようとすることはありません。 代わりに、データノードプロセスは、低速なリモートメモリーによる不均衡を回避しようとします。 インターリーブ割当てが望ましくない場合は、オペレーティングシステムレベルで目的の動作を決定できるように、 `Numa` を 0 に設定します。

`Numa` 構成パラメータは、 `libnuma.so` が使用可能な Linux システムでのみサポートされます。

- `RealtimeScheduler`

バージョン (またはそれ以降)	NDB 8.0.13
-----------------	------------

タイプまたは単位	boolean
デフォルト	false
範囲	true, false
再起動タイプ	N (NDB 8.0.13)

このパラメータを 1 に設定すると、データノードスレッドのリアルタイムスケジューリングが有効になります。

デフォルトは 0 (スケジューリングが無効) です。

- SchedulerExecutionTimer

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	µs
デフォルト	50
範囲	0 - 11000
再起動タイプ	N (NDB 8.0.13)

このパラメータは、スレッドがスケジューラで実行されてから送信されるまでの時間 (ミリ秒) を指定します。これを 0 に設定すると、応答時間が最小化されます。スループットを向上させるため、この値を増やすことができますが、その代償として応答時間は長くなります。

デフォルトは 50 マイクロ秒です。弊社のテストでは、これによって高負荷のケースで実質的に要求を遅延させずにスループットが若干向上することがわかっています。

- SchedulerResponsiveness

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	5
範囲	0 - 10
再起動タイプ	

速度とスループットのバランスを NDB スケジューラで設定します。このパラメータは、値が 0-10 の範囲の整数を取ります。デフォルトは 5 です。値を大きくすると、スループットと比較してレスポンス時間が向上します。値を小さくするとスループットが向上しますが、レスポンス時間は長くなります。

- SchedulerSpinTimer

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	µs
デフォルト	0
範囲	0 - 500
再起動タイプ	N (NDB 8.0.13)

このパラメータは、スレッドがスケジューラで実行されてからスリープ状態になるまでの時間 (ミリ秒) を指定します。

NDB 8.0.20 以降では、SpinMethod が設定されている場合、このパラメータの設定は無視されます。

- SpinMethod

バージョン (またはそれ以降)	NDB 8.0.20
タイプまたは単位	enumeration
デフォルト	StaticSpinning

範囲	CostBasedSpinning, LatencyOptimisedSpinning, DatabaseMachineSpinning, StaticSpinning
追加	NDB 8.0.20
再起動タイプ	N (NDB 8.0.13)

このパラメータは、次のリストに示すように、スピンパラメータ値に事前設定された 4 つの値を使用して、データノード上の適応スピンを制御する単純なインタフェースを提供します:

1. **StaticSpinning** (default): **EnableAdaptiveSpinning** を **false** に、**SchedulerSpinTimer** を 0 に設定します。(この場合、**SetAllowedSpinOverhead** は関係ありません。)
2. **CostBasedSpinning**: **EnableAdaptiveSpinning** を **true**、**SchedulerSpinTimer** を 100、**SetAllowedSpinOverhead** を 200 に設定します。
3. **LatencyOptimisedSpinning**: **EnableAdaptiveSpinning** を **true**、**SchedulerSpinTimer** を 200、**SetAllowedSpinOverhead** を 1000 に設定します。
4. **DatabaseMachineSpinning**: **EnableAdaptiveSpinning** を **true**、**SchedulerSpinTimer** を 500、**SetAllowedSpinOverhead** を 10000 に設定します。これは、スレッドが独自の CPU を所有する場合に使用することを目的としています。

次のリストに、**SpinMethod** によって変更されるスピンパラメータを示します:

- **SchedulerSpinTimer**: これは、その名前のデータノード構成パラメータと同じです。**SpinMethod** によってこのパラメータに適用された設定は、**config.ini** ファイルに設定された値をオーバーライドします。
- **EnableAdaptiveSpinning**: 適応スピンを有効または無効にします。これを無効にすると、CPU リソースのチェックを行わずにスピニングが実行されます。このパラメータはクラスタ構成ファイルで直接設定することはできず、ほとんどの状況では有効にする必要はありませんが、**DUMP 104004 1** を使用して直接有効にするか、**ndb_mgm** 管理クライアントで **DUMP 104004 0** を使用して無効にできます。
- **SetAllowedSpinOverhead**: 待機時間を取得できる CPU 時間を設定します。このパラメータは、**config.ini** ファイルで直接設定できません。ほとんどの場合、**SpinMethod** によって適用される設定は十分ですが、直接変更する必要がある場合は、**DUMP 104002 overhead** を使用して変更できます。ここで、**overhead** は 0 から 10000 の範囲の値です。詳細は、示された **DUMP** コマンドの説明を参照してください。

PowerPC や一部の SPARC プラットフォームなど、使用可能なスピン命令がないプラットフォームでは、スピニング時間はすべての状況で 0 に設定され、**StaticSpinning** 以外の **SpinMethod** の値は無視されます。

- **TwoPassInitialNodeRestartCopy**

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	boolean
デフォルト	true
範囲	true, false
再起動タイプ	N (NDB 8.0.13)

順序付けされたインデックスのマルチスレッド構築は、この構成パラメータを **true** (デフォルト値) に設定することで、データノードの初期再起動のために有効にできます。これにより、ノードの初期再起動時にデータの 2 パスコピーが可能になります。

同時に **BuildIndexThreads** を 0 以外の値に設定する必要があります。

マルチスレッドの構成パラメータ (**ndbmt**)。 **ndbmt** は、デフォルトではシングルスレッドプロセスとして動作するため、2 つの方法のいずれかを使用して、複数のスレッドを使用するように構成する必要があります。どちらの場合も、**config.ini** ファイルに構成パラメータを設定する必要があります。1 つ目の方法では、**MaxNoOfExecutionThreads** 構成パラメータに適切な値を設定します。別の方法として、**ThreadConfig** を使用して **ndbmt** マルチスレッドのより複雑なルールを設定できます。次のいくつかの段落では、これらのパラメータおよびマルチスレッドデータノードでのそれらの使用に関する情報を提供します。

注記

データノードで並列性を使用したバックアップでは、バックアップを取得する前に、クラスタ内のすべてのデータノードで複数の LDM が使用されている必要があります。詳細は、[セクション 23.5.8.5 「並列データノードを使用した NDB バックアップの作成」](#) および [セクション 23.4.23.3 「パラレルで作成されたバックアップからのリストア」](#) を参照してください。

- [AutomaticThreadConfig](#)

バージョン (またはそれ以降)	NDB 8.0.23
タイプまたは単位	boolean
デフォルト	false
範囲	true, false
追加	NDB 8.0.23
再起動タイプ	IS (NDB 8.0.13)

1 に設定すると、[taskset](#)、[numactl](#)、仮想マシン、Docker など、特定のアプリケーションで使用可能な CPU の制限を考慮して、データノードで使用可能な CPU の数を自動的に使用可能にします (Windows プラットフォームでは、自動スレッド構成はオンラインのすべての CPU を使用します)。または、[NumCPUs](#) を必要な CPU の数 (最大 1024、自動スレッド構成で処理可能な CPU の最大数) に設定できます。[ThreadConfig](#) および [MaxNoOfExecutionThreads](#) の設定は無視されます。

- [ClassicFragmentation](#)

バージョン (またはそれ以降)	NDB 8.0.23
タイプまたは単位	boolean
デフォルト	true
範囲	true, false
追加	NDB 8.0.23
再起動タイプ	N (NDB 8.0.13)

有効 ([true](#) に設定) にすると、NDB は LDM 間でのテーブルの断片の柔軟な分散を使用します。つまり、ノードごとのデフォルトのパーティション数は、データノードごとの LDM スレッドの最小数ではなく [PartitionsPerNode](#) と等しくなり、各テーブルのパーティション数はデータノードの数の 4 倍 ([PartitionsPerNode](#) のデフォルト) になります。

NDB 8.0.22 以前へのダウングレードが発生することが予想されない新しいクラスタの場合は、クラスタを最初に設定するときに [ClassicFragmentation](#) を [false](#) に設定することをお勧めします。これにより、すべてのパーティションがすべての LDM 間で均等に分散されます。

- [MaxNoOfExecutionThreads](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	2
範囲	2 - 72

再起動タイプ

IS (NDB 8.0.13)

このパラメータは、`ndbmysd` で使用される実行スレッドの数を最大 72 まで直接制御します。このパラメータは、`config.ini` ファイルの `[ndbd]` または `[ndbd default]` セクションに設定しますが、`ndbmysd` 専用であり、`ndbd` には適用されません。

`MaxNoOfExecutionThreads` を設定すると、ファイル `storage/ndb/src/kernel/vm/mt_thr_config.cpp` のマトリックスによって決定されるタイプごとのスレッド数が設定されます。このテーブルは、`MaxNoOfExecutionThreads` の可能な値に対するこれらのスレッド数を示しています。

表 23.12 スレッドタイプ別の `MaxNoOfExecutionThreads` 値および対応するスレッド数 (LQH、TC、送信、受信)。

MaxNoOfExecutionThreads の値	LQH スレッド	TC スレッド	送信スレッド	受信スレッド
0..3	1	0	0	1
4..6	2	0	0	1
7..8	4	0	0	1
9	4	2	0	1
10	4	2	1	1
11	4	3	1	1
12	6	2	1	1
13	6	3	1	1
14	6	3	1	2
15	6	3	2	2
16	8	3	1	2
17	8	4	1	2
18	8	4	2	2
19	8	5	2	2
20	10	4	2	2
21	10	5	2	2
22	10	5	2	3
23	10	6	2	3
24	12	5	2	3
25	12	6	2	3
26	12	6	3	3
27	12	7	3	3
28	12	7	3	4
29	12	8	3	4
30	12	8	4	4
31	12	9	4	4
32	16	8	3	3
33	16	8	3	4
34	16	8	4	4
35	16	9	4	4
36	16	10	4	4

MaxNoOfExecutionThreads の値	DM スレッド	TC スレッド	送信スレッド	受信スレッド
37	16	10	4	5
38	16	11	4	5
39	16	11	5	5
40	20	10	4	4
41	20	10	4	5
42	20	11	4	5
43	20	11	5	5
44	20	12	5	5
45	20	12	5	6
46	20	13	5	6
47	20	13	6	6
48	24	12	5	5
49	24	12	5	6
50	24	13	5	6
51	24	13	6	6
52	24	14	6	6
53	24	14	6	7
54	24	15	6	7
55	24	15	7	7
56	24	16	7	7
57	24	16	7	8
58	24	17	7	8
59	24	17	8	8
60	24	18	8	8
61	24	18	8	9
62	24	19	8	9
63	24	19	9	9
64	32	16	7	7
65	32	16	7	8
66	32	17	7	8
67	32	17	8	8
68	32	18	8	8
69	32	18	8	9
70	32	19	8	9
71	32	20	8	9

MaxNoOfExecutionThreads の値	LDM スレッド	TC スレッド	送信スレッド	受信スレッド
72	32	20	8	10

SUMA (レプリケーション) スレッドは常に 1 つです。

`NoOfFragmentLogParts` は、このパラメータの設定によって決定される、`ndbmtid` で使用される LDM スレッドの数と同じに設定する必要があります。この比率は 4:1 以下にする必要があります。このような場合の構成は特に禁止されています。

LDM スレッドの数によって、明示的にパーティション化されていない NDB テーブルで使用されるパーティションの数も決まります。これは、LDM スレッドの数にクラスタ内のデータノードの数を掛けた数です。(`ndbmtid` ではなくデータノードで `ndbd` が使用されている場合、常に単一の LDM スレッドが存在します。この場合、自動的に作成されるパーティションの数はデータノードの数と同じになります。詳しくは [セクション 23.1.2 「NDB Cluster ノード、ノードグループ、フラグメントレプリカ、およびパーティション」](#) をご覧ください。

デフォルト数を超える LDM スレッドを使用している場合に「ディスクデータ」テーブルに大きなテーブルスペースを追加すると、ディスクページバッファが十分に大きくないと、リソースおよび CPU 使用率に問題が発生する可能性があります。詳細は、[DiskPageBufferMemory](#) 構成パラメータの説明を参照してください。

スレッドタイプについては、このセクションで後述します ([ThreadConfig](#) を参照してください)。

このパラメータを許容範囲外の値に設定すると、管理サーバーの起動が中止され、次のエラーが発生します: [Error line number: Illegal value value for parameter MaxNoOfExecutionThreads](#)。

`MaxNoOfExecutionThreads` では、値 0 または 1 は NDB の内部で 2 に切り上げられるため、2 がこのパラメータのデフォルト値および最小値とみなされます。

`MaxNoOfExecutionThreads` は、通常、使用可能な CPU スレッド数と同じ値に設定し、各タイプのスレッド数を一般的なワークロードに合わせて割り当てることを目的としています。指定された CPU に特定のスレッドを割り当てるものではありません。提供された設定を変更することが望ましい場合や、スレッドを CPU にバインドする場合は、代わりに必要なタイプ、CPU、またはその両方に直接スレッドを割り当てることができる [ThreadConfig](#) を使用するようしてください。

マルチスレッドデータノードプロセスは常に、少なくとも次のスレッドを生成します:

- 1 個のローカルクエリーハンドラ (LDM) スレッド
- 1 個の受信スレッド
- 1 個のサブスクリプションマネージャー (SUMA またはレプリケーション) スレッド

`MaxNoOfExecutionThreads` 値が 8 以下の場合、TC スレッドは作成されず、かわりに TC 処理がメインスレッドによって実行されます。

LDM スレッドの数を変更するには、通常、このパラメータを使用して変更するか [ThreadConfig](#) を使用して変更するかに関係なく、システムの再起動が必要ですが、次の 2 つの条件が満たされている場合は、ノードの初期再起動 (NI) を使用して変更を有効にできます:

- 各 LDM スレッドは最大 8 つのフラグメントを処理
- テーブルフラグメントの合計数は、LDM スレッドの数の整数倍です。

NDB 8.0 では、NDB Cluster の一部の古いバージョンであったため、このパラメータの変更を有効にするために初期再起動は必要ありません。

- `NoOfFragmentLogParts`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	numeric
デフォルト	4

範囲	4, 6, 8, 10, 12, 16, 20, 24, 32
再起動タイプ	IN (NDB 8.0.13)

この `ndbmtid` に属する Redo ログのログファイルグループの数を設定します。このパラメータの値は、`MaxNoOfExecutionThreads` の設定によって決定される、`ndbmtid` で使用される LDM スレッドの数と同じに設定する必要があります。LDM ごとに 4 つ以上の redo ログ部分を使用する構成は許可されていません。

詳細は、`MaxNoOfExecutionThreads` の説明を参照してください。

- `NumCPUs`

バージョン (またはそれ以降)	NDB 8.0.23
タイプまたは単位	integer
デフォルト	0
範囲	0 - 1024
追加	NDB 8.0.23
再起動タイプ	IS (NDB 8.0.13)

自動スレッド構成では、この数の CPU のみが使用されます。`AutomaticThreadConfig` が有効になっていない場合は無効です。

- `PartitionsPerNode`

バージョン (またはそれ以降)	NDB 8.0.23
タイプまたは単位	integer
デフォルト	2
範囲	1 - 32
追加	NDB 8.0.23
再起動タイプ	N (NDB 8.0.13)

新しい NDB テーブルの作成時に各ノードで使用されるパーティションの数を設定します。これにより、ローカルデータマネージャ (LDM) の数が増加した場合に、テーブルを過剰な数のパーティションに分割することを回避できます。

このパラメータは異なるデータノード上で異なる値に設定でき、そのための既知の問題はありませんが、グローバル `config.ini` ファイルの `[ndbd default]` セクションですべてのデータノードに対して一度だけ設定することをお勧めします。

- `ThreadConfig`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	string
デフォルト	"
範囲	...
再起動タイプ	IS (NDB 8.0.13)

このパラメータは、`ndbmtid` で異なるタイプのスレッドを別の CPU に割り当てるために使用されます。この値は、次の構文を持つ形式の文字列です。

```
ThreadConfig := entry[,entry[,...]]
```

```
entry := type={param[,param[,...]]}
```

```
type (NDB 8.0.22 and earlier) := Idm | main | rcv | send | rep | io | tc | watchdog | idxbld
```

```
type (NDB 8.0.23 and later) := Idm | query | recover | main | rcv | send | rep | io | tc | watchdog | idxbld
```

```
param := count=number
| cpubind=cpu_list
| cpuset=cpu_list
| spintime=number
| realtime={0|1}
| nosend={0|1}
| thread_prio={0..10}
| cpubind_exclusive=cpu_list
| cpuset_exclusive=cpu_list
```

リストにパラメータが 1 つしかない場合でも、パラメータのリストを囲む中カッコ ({...}) は必須です。

`param` (パラメータ) は、次の情報のいずれかまたはすべてを指定します:

- 指定されたタイプ (`count`) のスレッド数。
- 特定のタイプのスレッドが非排他的にバインドされる CPU のセット。これは、`cpubind` または `cpuset` のいずれかによって決定されます。`cpubind` では、各スレッドがセット内の CPU に (非排他的に) バインドされます。`cpuset` では、各スレッドが指定された CPU のセットに (非排他的に) バインドされます。

Solaris では、かわりに、特定のタイプのスレッドが排他的にバインドされる CPU のセットを指定できます。`cpubind_exclusive` では、各スレッドがセット内の CPU に排他的にバインドされます。`cpuset_exclusive` では、各スレッドが指定された CPU のセットに排他的にバインドされます。

単一の構成で指定できるのは、`cpubind`、`cpuset`、`cpubind_exclusive` または `cpuset_exclusive` のいずれかのみです。

- `spintime` は、スリープする前にスレッドがスピンする待機時間をマイクロ秒単位で決定します。

`spintime` のデフォルト値は、`SchedulerSpinTimer` データノード構成パラメータの値です。

`spintime` は、I/O スレッド、ウォッチドッグまたはオフラインインデックス構築スレッドには適用されないため、これらのスレッドタイプには設定できません。

- `realtime` は 0 または 1 に設定できます。1 に設定すると、スレッドはリアルタイム優先度で実行されます。これは、`thread_prio` を設定できないことも意味します。

`realtime` パラメータは、デフォルトで `RealtimeScheduler` データノード構成パラメータの値に設定されます。

オフラインインデックス構築スレッドには `realtime` を設定できません。

- `nosend` を 1 に設定すると、`main`、`ldm`、`rep` または `tc` スレッドが送信スレッドを支援できなくなります。このパラメータはデフォルトでは 0 で、他のタイプのスレッドでは使用できません。
- `thread_prio` は、0 から 10 の範囲で設定できるスレッド優先度レベルで、10 は最大の優先度を表します。デフォルトは 5 です。このパラメータの正確な効果はプラットフォーム固有であり、このセクションの後半で説明します。

スレッド優先度レベルは、オフラインインデックス構築スレッドには設定できません。

プラットフォーム別の `thread_prio` の設定および効果。 `thread_prio` の実装は、Linux/FreeBSD、Solaris および Windows で異なります。次のリストでは、これらの各プラットフォームへの影響について順に説明します:

- Linux および FreeBSD: `thread_prio` を `nice` システムコールに提供される値にマップします。プロセスの有効性の値が低いほどプロセスの優先度が高くなるため、`thread_prio` を増やすと `nice` 値が小さくなります。

表 23.13 Linux および FreeBSD での `nice` 値への `thread_prio` のマッピング

<code>thread_prio</code> 値	<code>nice</code> 値
0	19
1	16
2	12

thread_prio 値	nice 値
3	8
4	4
5	0
6	-4
7	-8
8	-12
9	-16
10	-20

一部のオペレーティングシステムでは、最大プロセスの有効性レベルが 20 になる場合がありますが、これはすべてのターゲットバージョンでサポートされているわけではありません。このため、設定可能な最大 nice 値として 19 を選択します。

- Solaris : Solaris で `thread_prio` を設定すると、次のテーブルに示すマッピングを使用して Solaris FX 優先度が設定されます:

表 23.14 Solaris での thread_prio から FX への優先度のマッピング

thread_prio 値	Solaris FX の優先度
0	15
1	20
2	25
3	30
4	35
5	40
6	45
7	50
8	55
9	59
10	60

`thread_prio` 設定 9 は、Solaris で特別な FX 優先度値 59 にマップされます。これは、オペレーティングシステムが独自の CPU コアでスレッドを強制的に単独で実行しようとすることを意味します。

- Windows: `thread_prio` を、Windows API `SetThreadPriority()` 関数に渡される Windows スレッド優先度値にマップします。このマッピングを次のテーブルに示します:

表 23.15 thread_prio から Windows スレッド優先度へのマッピング

thread_prio 値	Windows スレッドの優先順位
0 - 1	THREAD_PRIORITY_LOWEST
2 - 3	THREAD_PRIORITY_BELOW_NORMAL
4 - 5	THREAD_PRIORITY_NORMAL
6 - 7	THREAD_PRIORITY_ABOVE_NORMAL

thread_prio 値	Windows スレッドの優先順位
8 - 10	THREAD_PRIORITY_HIGHEST

`type` 属性は、NDB のスレッドタイプを表します。次のリストに、サポートされるスレッドタイプと、それぞれに許可される `count` 値の範囲を示します:

- `ldm`: データを処理するローカルクエリーハンドラ (`DBLQH` カーネルブロック)。使用される LDM スレッドの数が多いほど、データがより高度にパーティション化されます。各 LDM スレッドには、固有のデータおよびインデックスパーティションのセットと固有の Redo ログが保持されます。NDB 8.0.23 より前では、`ldm` の値セットは 1、2、4、6、8、12、16、24、または 32 のいずれかである必要があります。In NDB 8.0.23 and later, it is possible to set `ldm` to any value in the range 1 to 332 inclusive; it also becomes possible to set it to 0, provided that `main`, `rep`, and `tc` are also 0, and that `recv` is set to 1; doing this causes `ndbmtid` to emulate `ndbd`.

LDM スレッドの数を変更するには、通常、クラスタ操作を有効かつ安全に行うためにシステムの再起動が必要です。この要件は、このセクションの後半で説明するように、特定の状況で緩和されます。これは、`MaxNoOfExecutionThreads` を使用して実行する場合にも当てはまります。

デフォルト数を超える LDM を使用している場合に「ディスクデータ」テーブルに大きなテーブルスペース (数 GB 以上) を追加すると、`DiskPageBufferMemory` が十分に大きくないと、リソースおよび CPU の使用率に問題が発生する可能性があります。

- `query` (NDB 8.0.23 で追加): クエリースレッドは LDM に関連付けられ、LDM グループを形成します。`READ COMMITTED` クエリーに対してのみ機能します。クエリースレッドの数は、LDM スレッドの数の 0、1、2 または 3 倍に設定する必要があります。これが `query` をゼロ以外の値に設定するか、`AutomaticThreadConfig` パラメータを有効にすることによってオーバーライドされないかぎり、クエリースレッドは使用されません。この場合、LDM は NDB 8.0.23 の前と同じように動作します。

クエリースレッドもリカバリスレッドとして機能しますが (次の項目を参照)、その逆は当てはまりません。

クエリースレッドの数を変更するには、ノードの再起動が必要です。

- `recover` (NDB 8.0.23 で追加): 回復スレッドは、LCP の一部としてフラグメントからデータを復元します。

リカバリスレッドの数を変更するには、ノードの再起動が必要です。

- `tc`: 進行中のトランザクションの状態を含むトランザクションコーディネータスレッド (`DBTC` カーネルブロック)。NDB 8.0.23 以降では、TC スレッドの最大数は 128 で、以前は 32 でした。

新しいトランザクションをそれぞれ 1 つの新しい TC スレッドに適切に割り当てることができます。ほとんどの場合、この実行が保証されるには、2 つの LDM スレッドに対して 1 つの TC スレッドで十分です。読み取りの数に比べて書き込みの数が少ないケースでは、4 つの LQH スレッドあたり 1 つの TC スレッドがあればトランザクション状態を維持できる可能性があります。逆に、非常に多くの更新を実行するアプリケーションでは、LDM スレッドに対する TC スレッドの比率を 1 に近づける (たとえば、4 つの LDM スレッドに対して TC スレッドを 3 つにする) 必要がある場合があります。

`tc` を 0 に設定すると、TC 処理がメインスレッドによって実行されます。ほとんどの場合、これは事実上 1 に設定することと同じです。

Range: 0-64(NDB 8.0.22 以前: 0 - 32)

- **main**: スキーマ管理を提供するデータディクショナリおよびトランザクションコーディネータ (DBDIH および DBTC カーネルブロック)。NDB 8.0.23 より前は、これは NDB 8.0.23 から始まる単一の専用スレッドによって常に処理されていましたが、ゼロまたは 2 つのメインスレッドを指定することもできました。

範囲:

- NDB 8.0.22 以前: 1 only.

NDB 8.0.23 以降: 0-2。

main を 0 に、**rep** を 1 に設定すると、**main** ブロックが **rep** スレッドに配置されます。結合されたスレッドは、`ndbinfo.threads` テーブルに `main_rep` として表示されます。これは事実上、**rep** を 1 に、**main** を 0 に設定することと同じです。

main と **rep** の両方を 0(ゼロ) に設定することもできます。この場合、両方のスレッドが最初の `recv` スレッドに配置され、結果として生成される結合スレッドの名前は `threads` テーブルで `main_rep_recv` になります。

- **recv**: 受信スレッド (CMVMI カーネルブロック)。各受信スレッドは、NDB Cluster 内のほかのノードと通信するための 1 つ以上のソケットを、ノードごとに 1 つのソケットで処理します。NDB Cluster は複数の受信スレッドをサポートします。このようなスレッドの最大数は 16 です。

範囲:

- NDB 8.0.22 以前: 1 - 16

- NDB 8.0.23 以降: 1 - 64

- **send**: 送信スレッド (CMVMI カーネルブロック)。スループットを向上させるため、1 つ以上 (最大 8 個) の独立した専用スレッドから送信を実行できます。

NDB 8.0.20 以降では、マルチスレッド実装が変更されたため、多くの送信スレッドを使用すると、スケーラビリティに悪影響を与える可能性があります。

以前は、すべてのスレッドが独自の送信を直接処理していました。これは、送信スレッドの数を 0 に設定することで実行できます (`MaxNoOfExecutionThreads` が 10 未満に設定されている場合にも発生します)。これを行うと、スループットに悪影響を及ぼす可能性があります。場合によっては待機時間が減る可能性もあります。

範囲:

- NDB 8.0.22 以前: 0 - 16

- NDB 8.0.23 以降: 0 - 64

- **rep**: レプリケーションスレッド (SUMA カーネルブロック)。NDB 8.0.23 より前では、非同期レプリケーション操作は常に単一の専用スレッドによって処理されます。NDB 8.0.23 以降では、このスレッドをメインスレッドと組み合わせることができます (範囲情報を参照)。

範囲:

- NDB 8.0.22 以前: 1 only.

- NDB 8.0.23 以降: 0-1。

rep を 0 に、**main** を 1 に設定すると、**rep** ブロックが **main** スレッドに配置されます。結合されたスレッドは、`ndbinfo.threads` テーブルに `main_rep` として表示されます。これは事実上、**main** を 1 に、**rep** を 0 に設定することと同じです。

main と **rep** の両方を 0(ゼロ) に設定することもできます。この場合、両方のスレッドが最初の `recv` スレッドに配置され、結果として生成される結合スレッドの名前は `threads` テーブルで `main_rep_recv` になります。

- `io`: ファイルシステムおよびその他の操作。これらは、要求の厳しいタスクではなく、常に 1 つの I/O 専用スレッドでまとめて処理されます。

範囲: 1 のみ。

- `watchdog`: このタイプに関連付けられたパラメータ設定は、実際には複数のスレッドに適用され、それぞれに特定の用途があります。これらのスレッドには、他のノードから接続設定を受信する `SocketServer` スレッド、他のノードへの接続を設定しようとする `SocketClient` スレッド、およびスレッドが進行中であることを確認するスレッドウォッチドッグスレッドが含まれます。

範囲: 1 のみ。

- `idxbld`: オフラインインデックス構築スレッド。前述の永続的な他のスレッドタイプとは異なり、これらは、ノードまたはシステムの再起動時、または `ndb_restore --rebuild-indexes` の実行時にのみ作成および使用される一時スレッドです。永続スレッドタイプにバインドされた CPU セットと重複する CPU セットにバインドできます。

`thread_prio`、`realtime` および `spintime` の値は、オフラインインデックス構築スレッドには設定できません。また、このタイプのスレッドでは、`count` は無視されます。

`idxbld` が指定されていない場合、デフォルトの動作は次のとおりです:

- オフラインインデックス構築スレッドは、I/O スレッドもバインドされておらず、これらのスレッドが使用可能なコアを使用している場合はバインドされません。
- I/O スレッドがバインドされている場合、オフラインインデックス構築スレッドはバインドされたスレッドのセット全体にバインドされます。これは、これらのスレッドが実行する他のタスクがないためです。

Range: 0 - 1。

`ThreadConfig` を変更するには通常、システムの初期再起動が必要ですが、この要件は特定の状況下で緩和できます:

- 変更後も LDM スレッドの数が以前と同じである場合は、変更を実装するために単純なノードの再起動 (ローリング再起動または N) 以外はありません。
- それ以外の場合 (つまり、LDM スレッドの数が変更された場合)、次の 2 つの条件が満たされていれば、ノードの初期再起動 (NI) を使用して変更を有効にできます:
 - a. 各 LDM スレッドは最大 8 つのフラグメントを処理
 - b. テーブルフラグメントの合計数は、LDM スレッドの数の整数倍です。

それ以外の場合は、このパラメータを変更するためにシステムの初期再起動が必要です。

NDB では、次の両方の基準によってスレッドタイプを区別できます:

- スレッドが実行スレッドかどうか。 `main`、`ldm`、`query` タイプ (NDB 8.0.23 以降)、`recv`、`rep`、`tc`、および `send` は実行スレッドです。`io`、`recover` (NDB 8.0.23 以降)、`watchdog`、および `idxbld` スレッドは実行スレッドとみなされません。
- 特定のタスクへのスレッドの割当てが永続的か一時的か。現在、`idxbld` 以外のすべてのスレッドタイプは永続とみなされます。`idxbld` スレッドは一時スレッドとみなされます。

簡単な例:

```
# Example 1.
ThreadConfig=ldm={count=2,cpubind=1,2},main={cpubind=12},rep={cpubind=11}

# Example 2.
```

```
ThreadConfig=main={cpubind=0},ldm={count=4,cpubind=1,2,5,6},io={cpubind=3}
```

通常、スレッドの使用法を構成するときは、データノードホストの 1 つ以上の CPU をオペレーティングシステムおよびその他のタスク用に確保します。したがって、24 個の CPU を搭載したホストマシンでは、20 個の CPU スレッド (8 個の LDM スレッド、4 個の TC スレッド (LDM スレッドの半分)、3 個の送信スレッド、3 個の受信スレッド、およびスキーマ管理、非同期レプリケーション、I/O 操作のそれぞれに 1 個のスレッド) を使用できます (ほかの用途のために 4 個を残します)。 (これは、[MaxNoOfExecutionThreads](#) を 20 に設定したときに使用されるスレッドの分布とほぼ同じです。) 次の [ThreadConfig](#) 設定では、これらの割り当てを行い、さらにこれらのスレッドを特定の CPU にバインドしています。

```
ThreadConfig=ldm={count=8,cpubind=1,2,3,4,5,6,7,8},main={cpubind=9},io={cpubind=9},\
rep={cpubind=10},tc={count=4,cpubind=11,12,13,14},recv={count=3,cpubind=15,16,17},\
send={count=3,cpubind=18,19,20}
```

ほとんどの場合、ここに示す例で行なったように、main (スキーマ管理) スレッドと I/O スレッドは同じ CPU にバインドできます。

次の例では、[cpuset](#) と [cpubind](#) の両方を使用して定義された CPU のグループ、およびスレッド優先順位付けの使用を組み込みます。

```
ThreadConfig=ldm={count=4,cpuset=0-3,thread_prio=8,spintime=200},\
ldm={count=4,cpubind=4-7,thread_prio=8,spintime=200},\
tc={count=4,cpuset=8-9,thread_prio=6},send={count=2,thread_prio=10,cpubind=10-11},\
main={count=1,cpubind=10},rep={count=1,cpubind=11}
```

この場合、2 つの LDM グループを作成します。最初の LDM グループは [cpubind](#) を使用し、2 つ目の LDM グループは [cpuset](#) を使用します。[thread_prio](#) と [spintime](#) は、グループごとに同じ値に設定されます。これは、LDM スレッドの合計が 8 つあることを意味します。 ([NoOfFragmentLogParts](#) も 8 に設定されていることを確認してください。) 4 つの TC スレッドは 2 つの CPU のみを使用します。[cpuset](#) を使用して、グループ内のスレッドより少ない CPU を指定することは可能です。 (これは、[cpubind](#) には当てはまりません。) 送信スレッドは、[cpubind](#) を使用してこれらのスレッドを CPU 10 および 11 にバインドする 2 つのスレッドを使用します。メインスレッドおよびレポートスレッドは、これらの CPU を再利用できます。

この例では、オペレーティングシステムの機能および割り込みで CPU 10、11、22 および 23 を使用可能なままにして、ハイパースレッドを使用する 24 CPU ホストに [ThreadConfig](#) および [NoOfFragmentLogParts](#) を設定する方法を示します:

```
NoOfFragmentLogParts=10
ThreadConfig=ldm={count=10,cpubind=0-4,12-16,thread_prio=9,spintime=200},\
tc={count=4,cpuset=6-7,18-19,thread_prio=8},send={count=1,cpuset=8},\
recv={count=1,cpuset=20},main={count=1,cpuset=9,21},rep={count=1,cpuset=9,21},\
io={count=1,cpuset=9,21,thread_prio=8},watchdog={count=1,cpuset=9,21,thread_prio=9}
```

次のいくつかの例には、[idxbld](#) の設定が含まれます。これらのうち最初の 2 つは、[idxbld](#) に定義された CPU セットが他の (永続的な) スレッドタイプに指定された CPU セットとオーバーラップする方法を示しています。最初に [cpuset](#) を使用し、次に [cpubind](#) を使用します:

```
ThreadConfig=main,ldm={count=4,cpuset=1-4},tc={count=4,cpuset=5,6,7},\
io={cpubind=8},idxbld={cpuset=1-8}
```

```
ThreadConfig=main,ldm={count=1,cpubind=1},idxbld={count=1,cpubind=1}
```

次の例では、I/O スレッドの CPU を指定しますが、インデックス構築スレッドの CPU は指定しません:

```
ThreadConfig=main,ldm={count=4,cpuset=1-4},tc={count=4,cpuset=5,6,7},\
io={cpubind=8}
```

[ThreadConfig](#) 設定では、1 から 8 の番号が付けられた 8 つのコアにスレッドがロックされるため、ここに示す設定と同等です:

```
ThreadConfig=main,ldm={count=4,cpuset=1-4},tc={count=4,cpuset=5,6,7},\
io={cpubind=8},idxbld={cpuset=1,2,3,4,5,6,7,8}
```

[ThreadConfig](#) を使用することで得られる高い安定性を生かすには、CPU が隔離されていて、割り込みを受けたり、オペレーティングシステムによってほかのタスクをスケジュールされたりしない必要があります。多くの Linux システムでは、[/etc/sysconfig/irqbalance](#) の [IRQBALANCE_BANNED_CPUS](#) を `0xFFFFF0` に設定

し、`grub.conf` の `isolcpus` ブートオプションを使用することで、これを実現できます。具体的な情報については、オペレーティングシステムまたはプラットフォームのドキュメントを参照してください。

ディスクデータの構成パラメータ。ディスクデータの動作に影響を与える構成パラメータには、次が含まれます。

- [DiskPageBufferEntries](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	32K pages
デフォルト	10
範囲	1 - 1000
バージョン (またはそれ以降)	NDB 8.0.19
タイプまたは単位	bytes
デフォルト	64MB
範囲	4MB - 16TB
再起動タイプ	N (NDB 8.0.13)

これは、割り当てるページエントリ (ページ参照) の数です。これは、[DiskPageBufferMemory](#) で 32K ページの数として指定されます。ほとんどの場合、デフォルトで十分ですが、「ディスクデータ」テーブルで非常に大きなトランザクションに問題が発生した場合は、このパラメータの値を増やす必要があります。各ページエントリには約 100 バイトが必要です。

- [DiskPageBufferMemory](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	bytes
デフォルト	64M
範囲	4M - 1T
バージョン (またはそれ以降)	NDB 8.0.19
タイプまたは単位	bytes
デフォルト	64M
範囲	4M - 16T
再起動タイプ	N (NDB 8.0.13)

これは、ディスク上のページをキャッシュするために使用されるスペースの量を指定するもので、`config.ini` ファイルの `[ndbd]` または `[ndbd default]` セクションで設定されます。

注記

以前は、このパラメータは 32 KB のページ数として指定されていました。NDB 8.0.19 以降では、バイト数として指定されます。

`ThreadConfig` (`{lwm=6...}` など) でデフォルト数を超える LDM スレッドを使用する場合に [DiskPageBufferMemory](#) の値が小さすぎると、大きな (500G などの) データファイルをディスクベースの NDB テーブルに追加しようとしたときに問題が発生することがあり、CPU コアのいずれかを占有している間、プロセスが無限に長くなる可能性があります。

これは、テーブルスペースへのデータファイルの追加の一環としてエクステントページが PGMAN ワーカー スレッドでメモリーにロックされ、メタデータにすばやくアクセスできるためです。大きいファイルを追加する場合、このワーカーにはすべてのデータファイルメタデータに対して十分なメモリーがありません。このような場合は、[DiskPageBufferMemory](#) を増やすか、小さいテーブルスペースファイルを追加する必要があります。[DiskPageBufferEntries](#) の調整が必要になる場合もあります。

`ndbinfo.diskpagebuffer` テーブルをクエリーすると、不要なディスクシークを最小限に抑えるためにこのパラメータの値を増やすべきかどうか判定しやすくなります。詳細は、[セクション23.5.14.27「ndbinfo diskpagebuffer テーブル」](#)を参照してください。

- `SharedGlobalMemory`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	bytes
デフォルト	128M
範囲	0 - 64T
再起動タイプ	N (NDB 8.0.13)

このパラメータは、テーブルスペース、ログファイルグループ、Undo ファイル、およびデータファイル用のログバッファ、ディスク操作 (ページ要求や待機キューなど)、およびメタデータに使用されるメモリーの量を指定します。共有グローバルメモリープールは、`CREATE LOGFILE GROUP` および `ALTER LOGFILE GROUP` ステートメントで使用される `UNDO_BUFFER_SIZE` オプションのメモリー要件を満たすために使用されるメモリーも提供します。これには、`InitialLogFileGroup` データノード構成パラメータの設定によってこのオプションに暗黙的に指定されるデフォルト値も含まれます。`SharedGlobalMemory` は、`config.ini` 構成ファイルの `[ndbd]` または `[ndbd default]` セクションに設定でき、バイト単位で測定されます。

デフォルト値は `128M` です。

- `DiskIOThreadPool`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	threads
デフォルト	2
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

このパラメータは、ディスクデータファイルへのアクセスに使用される未バインドスレッドの数を指定します。`DiskIOThreadPool` が導入される前は、ディスクデータファイルごとに1つのスレッドしか生成されなかったため、特に非常に大きいデータファイルを使用する場合に、パフォーマンスの問題が発生する可能性がありました。`DiskIOThreadPool` を使用すると、たとえば、並列で動作する複数のスレッドを使用して1つの大きなデータファイルにアクセスできます。

このパラメータは、ディスクデータ I/O スレッドにのみ適用されます。

このパラメータの最適な値は、使用しているハードウェアと構成によって異なり、次の要因を含みます。

- ディスクデータファイルの物理的な分布。データファイル、Undo ログファイル、およびデータノードファイルシステムを別々の物理ディスクに配置することで、パフォーマンスを向上させることができます。これらのファイルセットの一部またはすべてでこれを行う場合は、`DiskIOThreadPool` をより高く設定して、個別のスレッドで各ディスク上のファイルを処理できるようにすることができます。

NDB 8.0.19 以降では、「ディスクデータ」ファイル用に別のディスクを使用する場合も `DiskDataUsingSameDisk` を無効にするようにしてください。これにより、「ディスクデータ」テーブルスペースのチェックポイントを実行できる速度が向上します。

- ディスクのパフォーマンスとタイプ。ディスクデータファイルの処理用に提供できるスレッドの数は、ディスクの速度とスループットにも依存します。ディスクの速度が速く、スループットが高いほど、より多くの I/O スレッドを使用できます。弊社のテスト結果では、従来のディスクよりソリッドステートディスクドライブの方が

(つまり `DiskIOThreadPool` の値が大きいほど) はるかに多くのディスク I/O スレッドを処理できることを示しています。

ソリッドステートディスクドライブ (特に NVMe を使用するディスクドライブ) を使用する場合は、`TimeBetweenGlobalCheckpoints` を減らすこともお勧めします。 [ディスクデータ待機時間パラメータ](#) も参照してください。

このパラメータのデフォルト値は 2 です。

- ディスクデータのファイルシステムパラメータ。 次のリストのパラメータを使用すると、シンボリックリンクを使用しなくても NDB Cluster ディスクデータファイルを特定のディレクトリに配置できます。
- [FileSystemPathDD](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	filename
デフォルト	FileSystemPath
範囲	...
再起動タイプ	IN (NDB 8.0.13)

このパラメータが指定されている場合、NDB Cluster ディスクデータデータファイルと undo ログファイルは指定されたディレクトリに配置されます。 データファイル、Undo ログファイル、またはその両方をオーバーライドするには、`FileSystemPathDataFiles`、`FileSystemPathUndoFiles`、またはその両方の値をこれらのパラメータの説明に従って指定します。 データファイルでは `CREATE TABLESPACE` または `ALTER TABLESPACE` ステートメントの `ADD DATAFILE` 句にパスを指定し、Undo ログファイルでは `CREATE LOGFILE GROUP` または `ALTER LOGFILE GROUP` ステートメントの `ADD UNDOFILE` 句でパスを指定してオーバーライドすることもできます。 `FileSystemPathDD` が指定されていない場合は、`FileSystemPath` が使用されます。

特定のデータノードで `FileSystemPathDD` ディレクトリが指定された場合 (`config.ini` ファイルの `[ndbd default]` セクションでこのパラメータが指定された場合を含む)、`--initial` を指定してデータノードを起動すると、ディレクトリ内のすべてのファイルが削除されます。

- [FileSystemPathDataFiles](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	filename
デフォルト	FileSystemPathDD
範囲	...
再起動タイプ	IN (NDB 8.0.13)

このパラメータが指定されている場合、NDB Cluster ディスクデータファイルは指定されたディレクトリに配置されます。 これは、`FileSystemPathDD` に設定された値をオーバーライドします。 特定のデータファイルのパラメータをオーバーライドするには、そのデータファイルを作成するために使用される `CREATE TABLESPACE` または `ALTER TABLESPACE` ステートメントの `ADD DATAFILE` 句でパスを指定します。 `FileSystemPathDataFiles` が指定されていない場合は、`FileSystemPathDD` が使用されます (`FileSystemPathDD` も設定されていない場合は、`FileSystemPath` が使用されます)。

特定のデータノードで `FileSystemPathDataFiles` ディレクトリが指定された場合 (`config.ini` ファイルの `[ndbd default]` セクションにこのパラメータが指定された場合を含む)、`--initial` を指定してそのデータノードを起動すると、ディレクトリ内のすべてのファイルが削除されます。

- [FileSystemPathUndoFiles](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	filename
デフォルト	FileSystemPathDD

範囲	...
再起動タイプ	IN (NDB 8.0.13)

このパラメータが指定されている場合、NDB Cluster Disk Data undo ログファイルは指定されたディレクトリに配置されます。これは、`FileSystemPathDD` に設定された値をオーバーライドします。特定のデータファイルの作成に使用する `CREATE LOGFILE GROUP` ステートメントまたは `ALTER LOGFILE GROUP` ステートメントの `ADD UNDO` 句にパスを指定することで、そのデータファイルのこのパラメータをオーバーライドできます。`FileSystemPathUndoFiles` が指定されていない場合は、`FileSystemPathDD` が使用されます (`FileSystemPathDD` も設定されていない場合は、`FileSystemPath` が使用されます)。

特定のデータノードで `FileSystemPathUndoFiles` ディレクトリが指定された場合 (`config.ini` ファイルの `[nbd default]` セクションにこのパラメータが指定された場合を含む)、`--initial` を指定してそのデータノードを起動すると、ディレクトリ内のすべてのファイルが削除されます。

詳細は、[セクション23.5.10.1「NDB Cluster ディスクデータオブジェクト」](#)を参照してください。

- ディスクデータのオブジェクト作成パラメータ。 次の2つのパラメータを使用すると、クラスタを最初に起動したときに、SQL ステートメントを使用せずにディスクデータのログファイルグループ、テーブルスペース、またはその両方を作成できます。
- `InitialLogFileGroup`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	string
デフォルト	[see documentation]
範囲	...
再起動タイプ	S (NDB 8.0.13)

このパラメータを使用して、クラスタの初期起動の実行時に作成されるログファイルグループを指定できます。`InitialLogFileGroup` は、ここに示すように指定されます。

```
InitialLogFileGroup = [name=name:] [undo_buffer_size=size:] file-specification-list
```

```
file-specification-list:
  file-specification[: file-specification[: ...]]
```

```
file-specification:
  filename:size
```

ログファイルグループの `name` はオプションであり、デフォルト値は `DEFAULT-LG` です。`undo_buffer_size` もオプションです。省略した場合のデフォルト値は `64M` です。`file-specification` は、それぞれ1つの Undo ログファイルに対応し、`file-specification-list` に少なくとも1つ指定する必要があります。Undo ログファイルは、`FileSystemPath`、`FileSystemPathDD`、および `FileSystemPathUndoFiles` に設定された値に従って、`CREATE LOGFILE GROUP` または `ALTER LOGFILE GROUP` ステートメントの結果として作成された場合と同じように配置されます。

次について考えます。

```
InitialLogFileGroup = name=LG1; undo_buffer_size=128M; undo1.log:250M; undo2.log:150M
```

これは、次の SQL ステートメントと同等です。

```
CREATE LOGFILE GROUP LG1
  ADD UNDOFILE 'undo1.log'
  INITIAL_SIZE 250M
  UNDO_BUFFER_SIZE 128M
  ENGINE NDBCLUSTER;

ALTER LOGFILE GROUP LG1
  ADD UNDOFILE 'undo2.log'
  INITIAL_SIZE 150M
```

```
ENGINE NDBCLUSTER;
```

このログファイルグループは、`--initial` を指定してデータノードを起動したときに作成されます。

初期ログファイルグループのリソースは、`SharedGlobalMemory` の値で示されるリソースとともにグローバルメモリープールに追加されます。

このパラメータを使用する場合は、常に `config.ini` ファイルの `[nbd default]` セクションに設定するようにしてください。異なるデータノードに異なる値が設定されている場合の NDB Cluster の動作は定義されません。

- `InitialTablespace`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	string
デフォルト	[see documentation]
範囲	...
再起動タイプ	S (NDB 8.0.13)

このパラメータを使用して、クラスタの初期起動時に作成される「NDB Cluster ディスクデータ」テーブルスペースを指定できます。`InitialTablespace` は、ここに示すように指定されます。

```
InitialTablespace = [name=name:] [extent_size=size:] file-specification-list
```

テーブルスペースの `name` はオプションであり、デフォルト値は `DEFAULT-TS` です。`extent_size` もオプションです。デフォルト値は `1M` です。`file-specification-list` には、`InitialLogfileGroup` パラメータに示された同じ構文が使用されます。唯一の違いは、`InitialTablespace` で使用される `file-specification` がそれぞれ 1 つのデータファイルに対応することです。`file-specification-list` には、少なくとも 1 つを指定する必要があります。データファイルは、`FileSystemPath`、`FileSystemPathDD`、および `FileSystemPathDataFiles` に設定された値に従って、`CREATE TABLESPACE` または `ALTER TABLESPACE` ステートメントの結果として作成された場合と同じように配置されます。

たとえば、`config.ini` ファイルの `[nbd default]` セクションで `InitialTablespace` を指定する次の行を考えてみます (`InitialLogfileGroup` の場合と同様に、異なるデータノードに異なる値が設定されている場合の NDB Cluster の動作として、このパラメータは常に `[nbd default]` セクションで設定するようにしてください)：

```
InitialTablespace = name=TS1; extent_size=8M; data1.dat:2G; data2.dat:4G
```

これは、次の SQL ステートメントと同等です。

```
CREATE TABLESPACE TS1
  ADD DATAFILE 'data1.dat'
  EXTENT_SIZE 8M
  INITIAL_SIZE 2G
  ENGINE NDBCLUSTER;

ALTER TABLESPACE TS1
  ADD DATAFILE 'data2.dat'
  INITIAL_SIZE 4G
  ENGINE NDBCLUSTER;
```

このテーブルスペースは、データノードが `--initial` で起動されたときに作成され、その後に「NDB Cluster ディスクデータ」テーブルを作成するたびに使用できます。

- ディスクデータ待機時間パラメータ。ここにリストされている 2 つのパラメータを使用して、「NDB Cluster ディスクデータ」テーブルの待機時間の問題の処理を改善できます。

- `MaxDiskDataLatency`

バージョン (またはそれ以降)	NDB 8.0.19
タイプまたは単位	ms
デフォルト	0

範囲	0 - 8000
追加	NDB 8.0.19
再起動タイプ	N (NDB 8.0.13)

このパラメータは、ディスクアクセスの最大許容平均レイテンシ (最大 8000 ミリ秒) を制御します。この制限に達すると、ディスクデータ I/O サブシステムの圧力を下げするために、NDB はトランザクションの中断を開始します。0 を使用して、待機時間チェックを無効にします。

- [DiskDataUsingSameDisk](#)

バージョン (またはそれ以降)	NDB 8.0.19
タイプまたは単位	boolean
デフォルト	true
範囲	...
追加	NDB 8.0.19
再起動タイプ	N (NDB 8.0.13)

「ディスクデータ」テーブルスペースで個別のディスクが使用されている場合は、このパラメータを `false` に設定します。これにより、ディスクの共有時に通常使用されるよりも高い速度でテーブルスペースへのチェックポイントを実行できます。

`DiskDataUsingSameDisk` が `true` の場合、NDB では、インメモリーチェックポイントの進行中は常にディスクデータチェックポイントの速度が低下するため、ディスク負荷を一定に保つことができます。

ディスクデータと GCP 停止エラー。 「GCP 停止が検出されたため、ノード `nodeid` はこのノードを強制終了しました」 (エラー 2303) などの「ディスクデータ」テーブルの使用時に発生する エラーは、多くの場合「GCP 停止エラー」と呼ばれます。このようなエラーは、Redo ログが十分な速さでディスクにフラッシュされていない場合に発生します。これは通常、ディスクの速度が低く、ディスクのスループットが十分でないことが原因です。

高速なディスクを使用し、ディスクデータファイルをデータノードファイルシステムとは別のディスクに配置することで、これらのエラーを回避しやすくなります。`TimeBetweenGlobalCheckpoints` の値を減らすと、グローバルチェックポイントごとに書き込まれるデータの量が減りやすくなるため、グローバルチェックポイントを書き込もうとしたときに Redo ログバッファのオーバーフローをある程度回避できる可能性があります。ただし、この値を減らすと GCP を書き込むことができる時間も減るため、注意が必要です。

前述の `DiskPageBufferMemory` について示した考慮事項に加えて、`DiskIOThreadPool` 構成パラメータを正しく設定することも非常に重要です。`DiskIOThreadPool` の設定値が大きすぎると、GCP 停止エラーが発生する可能性が非常に高くなります (Bug #37227)。

GCP の停止は、保存またはコミットのタイムアウトによって発生することがあります。`TimeBetweenEpochsTimeout` データノード構成パラメータは、コミットのタイムアウトを指定します。ただし、このパラメータを 0 に設定することで、両方のタイプのタイムアウトを無効にできます。

送信バッファメモリーの割り当てを構成するためのパラメータ。送信バッファメモリーは、すべてのトランスポータ間で共有されるメモリープールから動的に割り当てられます。これは、送信バッファのサイズを必要に応じて調整できることを意味します。(以前は、クラスター内のすべてのノードで NDB カーネルが固定サイズの送信バッファを使用していました。これは、ノードの起動時に割り当てられ、ノードの実行中は変更できませんでした。) `TotalSendBufferMemory` および `OverLoadLimit` データノード構成パラメータを使用して、このメモリー割り当ての制限を設定できます。これらのパラメータ (および `SendBufferMemory`) の使用方法の詳細は、[セクション 23.3.3.14 「NDB Cluster 送信バッファパラメータの構成」](#) を参照してください。

- [ExtraSendBufferMemory](#)

このパラメータは、`TotalSendBufferMemory`、`SendBufferMemory`、またはその両方を使用して設定されたメモリーに加えて割り当てられるトランスポータ送信バッファメモリーの量を指定します。

- [TotalSendBufferMemory](#)

このパラメータは、すべての構成済みトランスポータ間で共有される送信バッファメモリの、このノードに割り当てられるメモリ合計量を決定するために使用されます。

このパラメータが設定されている場合、許可される最小値は 256KB です。0 はパラメータが設定されていないことを示します。詳細は、[セクション23.3.3.14「NDB Cluster 送信バッファパラメータの構成」](#)を参照してください。

[セクション23.5.7「NDB Cluster データノードのオンラインでの追加」](#)も参照してください。

Redo ログのオーバーコミット処理。 Redo ログをディスクにフラッシュする時間が長すぎる場合の、データノードによる操作の処理を制御できます。これは、特定の Redo ログのフラッシュが `RedoOverCommitLimit` 秒より長い時間をかけて `RedoOverCommitCounter` 回を超える回数分行われ、保留中のトランザクションが中止されたときに発生します。これが発生すると、トランザクションを送信した API ノードは、コミットされるはずだった操作を (`DefaultOperationRedoProblemAction` の指定に従って) キューに配置して再試行するか、中止することで処理できません。API ノードでこのアクションが実行される前に超過が許されるタイムアウトと回数を設定するためのデータノード構成パラメータについて、次のリストで説明します。

- [RedoOverCommitCounter](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	numeric
デフォルト	3
範囲	1 - 4294967039 (0xFFFFFFFF)
バージョン (またはそれ以降)	NDB 8.0.19
タイプまたは単位	numeric
デフォルト	3
範囲	1 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

特定の Redo ログをディスクに書き込もうとしたときに `RedoOverCommitLimit` を超過した回数がこの回数を超えると、結果としてコミットされなかったトランザクションはすべて中止され、トランザクションの発生元である API ノードは、これらのトランザクションを構成する操作を `DefaultOperationRedoProblemAction` の値に従って (操作をキューに入れて再試行するか、中止して) 処理します。

- [RedoOverCommitLimit](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	seconds
デフォルト	20
範囲	1 - 4294967039 (0xFFFFFFFF)
バージョン (またはそれ以降)	NDB 8.0.19
タイプまたは単位	seconds
デフォルト	20
範囲	1 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

このパラメータは、特定の Redo ログをディスクに書き込もうとしたときにタイムアウトするまでの上限 (秒単位) を設定します。データノードがこの Redo ログをフラッシュしようとして `RedoOverCommitLimit` よりも長い時間がかかった回数が保存され、`RedoOverCommitCounter` と比較されます。フラッシュにかかる時間が長すぎた回数がこのパラメータの値より多くなったときは、フラッシュタイムアウトの結果としてコミットされなかったトランザクションがすべて中止されます。これが発生すると、トランザクションの発生元である API ノードは、これらのトランザクションを構成する操作を `DefaultOperationRedoProblemAction` の設定に従って (操作をキューに入れて再試行するか、中止することで) 処理します。

再起動試行の制御。 [MaxStartFailRetries](#) および [StartFailRetryDelay](#) データノード構成パラメータを使用すると、データノードによる起動失敗時の再起動試行を細かく制御できます。

[MaxStartFailRetries](#) は、データノードの起動を中止するまでに行われる再試行の合計数を制限します。[StartFailRetryDelay](#) は、再試行間の秒数を設定します。これらのパラメータをここに示します。

- [StartFailRetryDelay](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	unsigned
デフォルト	0
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

このパラメータを使用して、データノードによる起動失敗時の再試行間の秒数を設定します。デフォルトは 0 (遅延なし) です。

[StopOnError](#) が 0 でない場合は、このパラメータと [MaxStartFailRetries](#) の両方が無視されます。

- [MaxStartFailRetries](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	unsigned
デフォルト	3
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

このパラメータを使用して、データノードによる起動失敗時の再試行回数を制限します。デフォルトは 3 回です。

[StopOnError](#) が 0 でない場合は、このパラメータと [StartFailRetryDelay](#) の両方が無視されます。

NDB インデックス統計のパラメータ。 次のリストのパラメータは NDB インデックス統計の生成に関連しています。

- [IndexStatAutoCreate](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	0
範囲	0, 1
再起動タイプ	

インデックスの作成時に自動統計収集を有効 (1 に設定) または無効 (0 に設定) にします。デフォルトで無効になっています。

- [IndexStatAutoUpdate](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	0
範囲	0, 1
再起動タイプ	

インデックスの変更の監視を有効化 (1 に設定) または無効化 (0 に設定) し、これらの自動統計更新をトリガーします。更新をトリガーするために必要な変更の量およびレベルは、[IndexStatTriggerPct](#) および [IndexStatTriggerScale](#) オプションの設定によって決定されます。

- [IndexStatSaveSize](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	bytes
デフォルト	32768
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	IN (NDB 8.0.13)

NDB システムテーブルおよび `mysqld` メモリーキャッシュに保存される特定のインデックスの統計に対して許容される最大スペース (バイト単位)。

サイズ制限に関係なく、常に少なくとも 1 つのサンプルが生成されます。このサイズは、[IndexStatSaveScale](#) によってスケールされます。

大規模なインデックスでは、[IndexStatSaveSize](#) に指定されたサイズが [IndexStatTriggerPct](#) に 0.01 を掛けた値によって調整されます。これにはさらに、インデックスサイズの 2 を底とする対数が乗算されます。[IndexStatTriggerPct](#) を 0 に設定すると、この調整が無効になります。

- [IndexStatSaveScale](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	percentage
デフォルト	100
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	IN (NDB 8.0.13)

大規模なインデックスでは、[IndexStatSaveSize](#) に指定されたサイズが [IndexStatTriggerPct](#) に 0.01 を掛けた値によって調整されます。これにはさらに、インデックスサイズの 2 を底とする対数が乗算されます。[IndexStatTriggerPct](#) を 0 に設定すると、この調整が無効になります。

- [IndexStatTriggerPct](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	percentage
デフォルト	100
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	IN (NDB 8.0.13)

インデックス統計更新をトリガーする更新内の変更割合。この値は、[IndexStatTriggerScale](#) によって調整されます。このトリガーを完全に無効にするには、[IndexStatTriggerPct](#) を 0 に設定します。

- [IndexStatTriggerScale](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	percentage
デフォルト	100
範囲	0 - 4294967039 (0xFFFFFFFF)

再起動タイプ	IN (NDB 8.0.13)
--------	-----------------

大規模なインデックスでは、この量に 0.01 を掛けた値で `IndexStatTriggerPct` を調整します。値 0 で調整が無効になります。

- `IndexStatUpdateDelay`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	seconds
デフォルト	60
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	IN (NDB 8.0.13)

特定のインデックスの自動インデックス統計更新間の最小遅延 (秒数)。この変数を 0 に設定すると、遅延が無効になります。デフォルトは 60 秒です。

再起動のタイプ。このセクションのパラメータの説明で使用される再起動タイプに関する情報を次のテーブルに示します:

表 23.16 NDB Cluster の再起動タイプ

シンボル	再起動タイプ	説明
N	ノード	パラメータはローリング再起動を使用して更新できます (セクション 23.5.5 「NDB Cluster のローリング再起動の実行」 を参照)
S	システム	このパラメータの変更を有効にするには、すべてのクラスタノードを完全に停止してから再起動する必要があります
I	Initial	<code>--initial</code> オプションを使用してデータノードを再起動する必要があります

23.3.3.7 NDB Cluster での SQL およびその他の API ノードの定義

`config.ini` ファイルの `[mysqld]` および `[api]` セクションでは、クラスタデータへのアクセスに使用される MySQL サーバー (SQL ノード) およびその他のアプリケーション (API ノード) が定義されます。示されているどのパラメータも必須ではありません。コンピュータ名またはホスト名が指定されていない場合は、任意のホストでこの SQL または API ノードを使用できます。

一般的には、`[mysqld]` セクションはクラスタへの SQL インタフェースを提供する MySQL サーバーを示すために使用され、`[api]` セクションはクラスタデータにアクセスする `mysqld` プロセス以外のアプリケーションのために使用されますが、この 2 つの指定は実際には同義です。たとえば、SQL ノードとして機能する MySQL サーバーのパラメータを `[api]` セクションに指定できます。

注記

NDB Cluster の MySQL サーバーオプションについては、[NDB Cluster の MySQL Server オプション](#) を参照してください。NDB Cluster に関連する MySQL サーバースystem変数については、[NDB Cluster システム変数](#) を参照してください。

- `ld`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	unsigned
デフォルト	[...]
範囲	1 - 255

再起動タイプ	IS (NDB 8.0.13)
--------	-----------------

`Id` は、すべてのクラスタ内部メッセージ内でノードを識別するために使用される整数値です。許容される値の範囲は、1-255 (これらを含む) です。この値は、ノードのタイプに関係なく、クラスタ内の各ノードで一意である必要があります。

注記

NDB 8.0.18 以降では、データノード ID は 145 未満である必要があります。多数のデータノードを配備する予定の場合は、API ノード (および管理ノード) のノード ID を 144 より大きい値に制限することをお勧めします。(NDB 8.0.18 より前では、データノード ID でサポートされる最大値は 48 でした。)

`NodeId` は、API ノードを識別するときに使用することが推奨されるパラメータ名です。(`Id` は、下位互換性に引き続き対応しますが、現在は非推奨であり、使用時に警告を生成します。また、これは今後削除される予定です。)

• ConnectionMap

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	string
デフォルト	[...]
範囲	...
再起動タイプ	N (NDB 8.0.13)

接続するデータノードを指定します。

• NodeId

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	unsigned
デフォルト	[...]
範囲	1 - 255
再起動タイプ	IS (NDB 8.0.13)

`NodeId` は、すべてのクラスタ内部メッセージ内でノードを識別するために使用される整数値です。許容される値の範囲は、1-255 (これらを含む) です。この値は、ノードのタイプに関係なく、クラスタ内の各ノードで一意である必要があります。

注記

NDB 8.0.18 以降では、データノード ID は 145 未満である必要があります。多数のデータノードを配備する予定の場合は、API ノード (および管理ノード) のノード ID を 144 より大きい値に制限することをお勧めします。(NDB 8.0.18 より前では、データノード ID でサポートされる最大値は 48 でした。)

`NodeId` は、管理ノードを識別するときに使用することが推奨されるパラメータ名です。エイリアス `Id` は、NDB Cluster の非常に古いバージョンでこの目的に使用され、下位互換性のために引き続きサポートされています。現在は非推奨になっており、使用時に警告が生成され、NDB Cluster の将来のリリースで削除される可能性があります。

• ExecuteOnComputer

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	name
デフォルト	[...]
範囲	...

非推奨	Yes (in NDB 7.5)
再起動タイプ	S (NDB 8.0.13)

これは、構成ファイルの `[computer]` セクションに定義されたいずれかのコンピュータ (ホスト) に設定されている `id` を参照します。

重要

このパラメータは非推奨であり、将来のリリースで削除される予定です。かわりに `HostName` パラメータを使用してください。

このノードのノード ID は、明示的にリクエストする接続にのみ指定できます。ノード ID をリクエストする管理サーバーは、このノード ID を使用できません。このパラメータは、同じホストで複数の管理サーバーを実行して、`HostName` でプロセスを区別するのに十分でない場合に使用できます。テストで使用するためのものです。

• `HostName`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	name or IP address
デフォルト	[...]
範囲	...
再起動タイプ	N (NDB 8.0.13)

このパラメータを指定すると、SQL ノード (API ノード) が配置されるコンピュータのホスト名が定義されます。ホスト名を指定するには、このパラメータまたは `ExecuteOnComputer` のいずれかが必要です。

`config.ini` ファイルの特定の `[mysql]` または `[api]` セクションに `HostName` または `ExecuteOnComputer` が指定されていない場合、SQL または API ノードはネットワーク接続を確立できる任意のホストから対応する「スロット」を使用して管理サーバーホストマシンに接続できます。これは、ほかに指定されていない場合 `localhost` が `HostName` として使用されるデータノードのデフォルトの動作とは異なります。

• `LocationDomainId`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	0
範囲	0 - 16
再起動タイプ	

クラウド内の特定の「可用性ドメイン」(可用性ゾーンとも呼ばれる) に SQL またはその他の API ノードを割り当てます。どのノードがどの可用性ドメインにあるかを `NDB` に通知することで、次の方法でクラウド環境のパフォーマンスを向上させることができます:

- リクエストされたデータが同じノードで見つからない場合、読取りは同じ可用性ドメイン内の別のノードに転送できます。
- 異なる可用性ドメイン内のノード間の通信では、それ以上の手動操作なしで `NDB` トランスポータ WAN サポートを使用することが保証されています。
- トランスポータグループ番号は、SQL および他の API ノードが可能な場合は常に同じ可用性ドメイン内のローカルデータノードと通信するように、使用される可用性ドメインに基づくことができます。

- アービトレータは、データノードが存在しない可用性ドメインから選択することも、そのような可用性ドメインが見つからない場合は 3 番目の可用性ドメインから選択することもできます。

`LocationDomainId` は 0 以上 16 以下の整数値を取り、0 がデフォルトです。0 を使用することは、パラメータを未設定のままにすることと同じです。

- `ArbitrationRank`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	0-2
デフォルト	0
範囲	0 - 2
再起動タイプ	N (NDB 8.0.13)

このパラメータは、アービトレータとして機能できるノードを定義します。管理ノードと SQL ノードの両方がアービトレータになることができます。値 0 は、指定されたノードがアービトレータとして使用されないことを意味します。値 1 は、ノードにアービトレータとしての高い優先度を与えます。値 2 は低い優先度を与えます。通常の構成では、管理サーバーの `ArbitrationRank` が 1 (管理ノードのデフォルト) に設定され、各 SQL ノードでは 0 (SQL ノードのデフォルト) に設定されるため、管理サーバーがアービトレータとして使用されます。

すべての管理および SQL ノードで `ArbitrationRank` を 0 に設定すると、アービトレーションを完全に無効化できます。このパラメータをオーバーライドしてアービトレーションを制御することもできます。そのためには、`config.ini` グローバル構成ファイルの `[ndbd default]` セクションに `Arbitration` パラメータを設定します。

- `ArbitrationDelay`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	milliseconds
デフォルト	0
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

このパラメータを 0 (デフォルト) 以外の値に設定すると、アービトレータによるアービトレーションリクエストへのレスポンスは、指定されたミリ秒数遅延されます。通常、この値を変更する必要はありません。

- `BatchByteSize`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	bytes
デフォルト	16K
範囲	1K - 1M
再起動タイプ	N (NDB 8.0.13)

フルテーブルスキャンまたはインデックスの範囲スキャンに変換されるクエリーでは、最良のパフォーマンスを得るため、適切にサイズ調整されたバッチでレコードをフェッチすることが重要です。レコード数 (`BatchSize`) とバイト数 (`BatchByteSize`) の両方で、適切なサイズを設定できます。実際のバッチサイズは両方のパラメータによって制限されます。

このパラメータの設定方法によっては、クエリーの実行速度の変化率が 40% を超える可能性があります。

このパラメータはバイト単位で測定されます。デフォルト値は 16K です。

- `BatchSize`

バージョン (またはそれ以降)	NDB 8.0.13
-----------------	------------

タイプまたは単位	records
デフォルト	256
範囲	1 - 992
再起動タイプ	N (NDB 8.0.13)

このパラメータはレコード数で測定され、デフォルトで 256 に設定されます。最大サイズは 992 です。

- [ExtraSendBufferMemory](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	bytes
デフォルト	0
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

このパラメータは、[TotalSendBufferMemory](#)、[SendBufferMemory](#)、またはその両方を使用して設定されたメモリーに加えて割り当てられるトランスポート送信バッファメモリーの量を指定します。

- [HeartbeatThreadPriority](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	string
デフォルト	[...]
範囲	...
再起動タイプ	

このパラメータを使用して、管理および API ノードのハートビートスレッドのスケジューリングポリシーと優先度を設定します。このパラメータを設定するための構文をここに示します。

```
HeartbeatThreadPriority = policy[, priority]
```

```
policy:
{FIFO | RR}
```

このパラメータを設定するときは、ポリシーを指定する必要があります。これは、[FIFO](#) (先入れ先出し) または [RR](#) (ラウンドロビン) のいずれかです。このあとに、オプションで優先度 (整数) を指定できます。

- [MaxScanBatchSize](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	bytes
デフォルト	256K
範囲	32K - 16M
再起動タイプ	N (NDB 8.0.13)

バッチサイズは、各データノードから送信される各バッチのサイズです。多数のノードから並列で受信される過大なデータ量から MySQL サーバーを保護するため、ほとんどのスキャンは並列で実行されます。このパラメータは、すべてのノードの合計バッチサイズに対する制限を設定します。

このパラメータのデフォルトの値は 256K バイトです。最大サイズは 16M バイトです。

- [TotalSendBufferMemory](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	bytes

デフォルト	0
範囲	256K - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

このパラメータは、すべての構成済みトランスポータ間で共有される送信バッファメモリの、このノードに割り当てられるメモリ合計量を決定するために使用されます。

このパラメータが設定されている場合、許可される最小値は 256KB です。0 はパラメータが設定されていないことを示します。詳細は、[セクション 23.3.3.14 「NDB Cluster 送信バッファパラメータの構成」](#) を参照してください。

- [AutoReconnect](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	boolean
デフォルト	false
範囲	true, false
再起動タイプ	N (NDB 8.0.13)

このパラメータは、デフォルトで `false` です。これによって、切断された API ノード (SQL ノードとして機能する MySQL サーバーを含む) が既存の接続を再使用しようとせず、クラスタへの新しい接続を強制的に使用するようになります (接続を再使用すると、動的に割り当てられたノード ID を使用するとき問題が発生することがあるため)。(Bug #45921)

注記

このパラメータは、NDB API を使用してオーバーライドできません。詳細は、[Ndb_cluster_connection::set_auto_reconnect\(\)](#) および [Ndb_cluster_connection::get_auto_reconnect\(\)](#) を参照してください。

- [DefaultOperationRedoProblemAction](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	enumeration
デフォルト	QUEUE
範囲	ABORT, QUEUE
再起動タイプ	

このパラメータは、([RedoOverCommitLimit](#) および [RedoOverCommitCounter](#) とともに) Redo ログをディスクにフラッシュする時間が長すぎる場合にデータノードによる操作の処理を制御します。これは、特定の Redo ログのフラッシュが [RedoOverCommitLimit](#) 秒より長い時間をかけて [RedoOverCommitCounter](#) 回を超える回数分行われ、保留中のトランザクションが中止されたときに発生します。

この発生時は、ノードはここに示す [DefaultOperationRedoProblemAction](#) の値に応じた 2 つの方法のいずれかで応答できます。

- **ABORT**: 中止されたトランザクションに含まれる保留中の操作も中止されます。
- **QUEUE**: 中止されたトランザクションに含まれる保留中の操作がキューに入れられ、再試行されます。これはデフォルトです。redo ログの領域が不足した場合、つまり [P_TAIL_PROBLEM](#) エラーが発生した場合、保留中の操作は中断されます。

- [DefaultHashMapSize](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	buckets 3653

デフォルト	3840
範囲	0 - 3840
再起動タイプ	N (NDB 8.0.13)

NDB で使用されるテーブルハッシュマップのサイズは、このパラメータを使用して構成できます。[DefaultHashMapSize](#) には、3 つの値 (0、240、3840) のいずれかを指定できます。これらの値とその効果について、次の表で説明します。

表 23.17 DefaultHashMapSize パラメータ値

値	説明/効果
0	クラスタ内のすべてのデータおよび API ノードの中で、このパラメータに設定されたもっとも小さい値を (あれば) 使用します。どのデータまたは API ノードにも設定されていない場合は、デフォルト値を使用します。
240	古いデフォルトハッシュマップサイズ
3840	NDB 8.0 でデフォルトで使用されるハッシュマップサイズ

このパラメータの元の使用目的は、ハッシュマップのサイズが異なる古い NDB Cluster バージョンとの間のアップグレードおよびダウングレードを容易にすることでした。これは、この変更の後方互換性がないためです。NDB Cluster 8.0 へのアップグレードまたは NDB Cluster 8.0 からのダウングレードでは、これは問題になりません。

- [Wan](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	boolean
デフォルト	false
範囲	true, false
再起動タイプ	N (NDB 8.0.13)

WAN の TCP 設定をデフォルトとして使用します。

- [ConnectBackoffMaxTime](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	0
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

多数の未起動のデータノードがある NDB Cluster では、このパラメータの値を上げて、クラスタ内でまだ機能していないデータノードへの接続試行を回避したり、管理ノードへの高トラフィックを中程度にしたりできます。API ノードが新しいデータノードに接続されていない場合は、[StartConnectBackoffMaxTime](#) パラメータの値が適用されます。それ以外の場合は、[ConnectBackoffMaxTime](#) を使用して接続試行間の待機時間の長さ (ミリ秒) が決定されません。

このパラメータの経過時間を計算する際に、ノードの接続試行中に経過する時間は考慮されません。タイムアウトは、100 ミリ秒の遅延から始まり、約 100 ミリ秒の分解能で適用されます。後続の試行のたびに、[ConnectBackoffMaxTime](#) ミリ秒 (最大 100000 ミリ秒 (100 秒)) に達するまでこの期間の長さが倍加されます。

API ノードがデータノードに接続され、そのノードが (ハートビートメッセージで) ほかのデータノードに接続したことを報告すると、データノードに対する接続試行はこのパラメータの影響を受けなくなり、その後接続されるまで 100 ミリ秒間隔で行われます。データノードが起動すると、これが発生したことを API ノードに通知するために [HeartbeatIntervalDbApi](#) を使用できます。

- [StartConnectBackoffMaxTime](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	0
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

多数の未起動のデータノードがある NDB Cluster では、このパラメータの値を上げて、クラスタ内でまだ機能していないデータノードへの接続試行を回避したり、管理ノードへの高トラフィックを中程度にしたりできます。API ノードが新しいデータノードに接続されていない場合は、[StartConnectBackoffMaxTime](#) パラメータの値が適用されます。それ以外の場合は、[ConnectBackoffMaxTime](#) を使用して接続試行間の待機時間の長さ (ミリ秒) が決定されます。

このパラメータの経過時間を計算する際に、ノードの接続試行中に経過する時間は考慮されません。タイムアウトは、100 ミリ秒の遅延から始まり、約 100 ミリ秒の分解能で適用されます。後続の試行のために、[StartConnectBackoffMaxTime](#) ミリ秒 (最大 100000 ミリ秒 (100 秒)) に達するまでこの期間の長さが倍加されます。

API ノードがデータノードに接続され、そのノードが (ハートビートメッセージで) ほかのデータノードに接続したことを報告すると、データノードに対する接続試行はこのパラメータの影響を受けなくなり、その後接続されるまで 100 ミリ秒間隔で行われます。データノードが起動すると、これが発生したことを API ノードに通知するために [HeartbeatIntervalDbApi](#) を使用できます。

API ノードのデバッグパラメータ。 [ApiVerbose](#) 構成パラメータを使用して、特定の API ノードからのデバッグ出力を有効にできます。このパラメータは整数値を取ります。デフォルトは 0 で、このようなデバッグは無効になります。1 ではクラスタログへのデバッグ出力が有効になり、2 では [DBDICT](#) デバッグ出力も追加されます。(Bug #20638450) [DUMP 1229](#) も参照してください。

次に示すように、`mysql` クライアントで [SHOW STATUS](#) を使用して NDB Cluster SQL ノードとして実行されている MySQL サーバーから情報を取得することもできます:

```
mysql> SHOW STATUS LIKE 'ndb%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Ndb_cluster_node_id | 5 |
| Ndb_config_from_host | 198.51.100.112 |
| Ndb_config_from_port | 1186 |
| Ndb_number_of_storage_nodes | 4 |
+-----+-----+
4 rows in set (0.02 sec)
```

このステートメントの出力に表示されるステータス変数については、[NDB Cluster ステータス変数](#) を参照してください。

注記

実行中の NDB Cluster の構成に新しい SQL または API ノードを追加するには、新しい `[mysqld]` または `[api]` セクションを `config.ini` ファイル (または、複数の管理サーバーを使用している場合はファイル) に追加したあとで、すべてのクラスタノードのローリング再起動を実行する必要があります。これは、新しい SQL または API ノードをクラスタに接続する前に実行する必要があります。

新しい SQL または API ノードがクラスタ構成内の以前に使用されていない API スロットを使用してクラスタに接続する場合、クラスタの再起動を実行する必要はありません。

再起動のタイプ。 このセクションのパラメータの説明で使用される再起動タイプに関する情報を次のテーブルに示します:

表 23.18 NDB Cluster の再起動タイプ

シンボル	再起動タイプ	説明
N	ノード	パラメータはローリング再起動を使用して更新できます (セクション 23.5.5 「NDB Cluster のローリング再起動の実行」 を参照)
S	システム	このパラメータの変更を有効にするには、すべてのクラスタノードを完全に停止してから再起動する必要があります
I	Initial	--initial オプションを使用してデータノードを再起動する必要があります

23.3.3.8 システムの定義

[system]セクションは、クラスタ全体に適用されるパラメータに使用されます。Name システムパラメータは MySQL Enterprise Monitor で使用され、ConfigGenerationNumber および PrimaryMGMNode は本番環境では使用されません。NDB Cluster を MySQL Enterprise Monitor とともに使用する場合を除き、config.ini ファイルに[system]セクションを含める必要はありません。

これらのパラメータの詳細は、次のリストを参照してください:

- [ConfigGenerationNumber](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	unsigned
デフォルト	0
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

構成生成番号。このパラメータは現在使用されていません。

- [Name](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	string
デフォルト	[...]
範囲	...
再起動タイプ	N (NDB 8.0.13)

クラスタの名前を設定します。このパラメータは、MySQL Enterprise Monitor を使用したデプロイメントに必要です。それ以外の場合は未使用です。

このパラメータの値を取得するには、Ndb_system_name ステータス変数を確認します。NDB API アプリケーションでは、get_system_name() を使用して取得することもできます。

- [PrimaryMGMNode](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	unsigned
デフォルト	0
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

プライマリ管理ノードのノード ID。このパラメータは現在使用されていません。

再起動のタイプ。このセクションのパラメータの説明で使用される再起動タイプに関する情報を次のテーブルに示します:

表 23.19 NDB Cluster の再起動タイプ

シンボル	再起動タイプ	説明
N	ノード	パラメータはローリング再起動を使用して更新できません (セクション 23.5.5 「NDB Cluster のローリング再起動の実行」を参照)
S	システム	このパラメータの変更を有効にするには、すべてのクラスタノードを完全に停止してから再起動する必要があります
I	Initial	--initial オプションを使用してデータノードを再起動する必要があります

23.3.3.9 NDB Cluster の MySQL Server オプションおよび変数

このセクションでは、NDB Cluster に固有の MySQL サーバーオプション、サーバーおよびステータス変数について説明します。これらの使用に関する一般的な情報、および NDB Cluster に固有でないその他のオプションと変数については、[セクション 5.1 「MySQL Server」](#) を参照してください。

クラスタ構成ファイル (通常は `config.ini` という名前) で使用される NDB Cluster 構成パラメータについては、[セクション 23.3 「NDB Cluster の構成」](#) を参照してください。

NDB Cluster の MySQL Server オプション

このセクションでは、NDB Cluster に関連する `mysqld` サーバーオプションについて説明します。NDB Cluster に固有ではない `mysqld` オプションについて、および `mysqld` でのオプションの使用に関する一般的な情報については、[セクション 5.1.7 「サーバーコマンドオプション」](#) を参照してください。

ほかの NDB Cluster プロセス (`ndbd`、`ndb_mgmd`、および `ndb_mgm`) で使用されるコマンド行オプションについては、[セクション 23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#) を参照してください。NDB のユーティリティープログラム (`ndb_desc`、`ndb_size.pl`、`ndb_show_tables` など) で使用されるコマンド行オプションについては、[セクション 23.4 「NDB Cluster プログラム」](#) を参照してください。

- `--ndbcluster`

コマンド行形式	<code>--ndbcluster[=value]</code>
無効化	<code>skip-ndbcluster</code>
型	列挙
デフォルト値	ON
有効な値	OFF FORCE

NDB Cluster を使用するには、`NDBCLUSTER` ストレージエンジンが必要です。 `mysqld` バイナリに `NDBCLUSTER` ストレージエンジンのサポートが含まれている場合、このエンジンはデフォルトで無効になっています。これを有効にするには、`--ndbcluster` オプションを使用します。エンジンを明示的に無効にするには、`--skip-ndbcluster` を使用します。

`--initialize` も使用されている場合、`--ndbcluster` オプションは無視されます (および `NDB` ストレージエンジンが有効になっていません)。 (このオプションを `--initialize` とともに使用する必要はありません。)

- `--ndb-allow-copying-alter-table=[ON|OFF]`

コマンド行形式	--ndb-allow-copying-alter-table[={OFF ON}]
システム変数	ndb_allow_copying_alter_table
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

ALTER TABLE および他の DDL ステートメントで NDB テーブルのコピー操作を使用できるようにします。これが発生しないようにするには、**OFF** に設定します。そうすると、クリティカルなアプリケーションのパフォーマンスが向上する可能性があります。

- --ndb-batch-size=#

コマンド行形式	--ndb-batch-size
システム変数	ndb_batch_size
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	32768
最小値	0
最大値	31536000

これは、NDB のトランザクションバッチで使用されるサイズ (バイト単位) を設定します。

- --ndb-cluster-connection-pool=#

コマンド行形式	--ndb-cluster-connection-pool
システム変数	ndb_cluster_connection_pool
システム変数	ndb_cluster_connection_pool
スコープ	グローバル
スコープ	グローバル
動的	いいえ
動的	いいえ
SET_VAR ヒントの適用	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1
最小値	1
最大値	63

このオプションを 1 (デフォルト) より大きい値に設定すると、**mysqld** プロセスはクラスタに対して複数の接続を使用し、実質的に複数の SQL ノードを模倣します。各接続には、クラスタ構成 (**config.ini**) ファイル内に固有の **[api]**

または `[mysqld]` セクションを必要とし、クラスタでサポートされる API 接続の最大数を参照してカウントされま

す。
2 台のクラスタホストコンピュータがあり、実行している各 SQL ノードの `mysqld` プロセスが `--ndb-cluster-connection-pool=4` を指定して起動されたとします。これは、これらの接続に使用できる API スロットがクラスタに (2 個ではなく) 8 個存在する必要があることを意味します。これらすべての接続は、SQL ノードがクラスタに接続したときに設定され、ラウンドロビン方式でスレッドに割り当てられます。

このオプションは、複数の CPU、複数のコア、またはその両方を搭載したホストマシンで `mysqld` を実行する場合にのみ有効です。最良の結果を得るには、この値をホストマシンで使用できるコアの合計数より小さくします。これより大きい値に設定すると、パフォーマンスが大幅に低下する可能性があります。

重要

接続プールを使用する各 SQL ノードは複数の API ノードスロットを占有する (各スロットにはクラスタ内で固有のノード ID がある) ため、接続プールを採用する `mysqld` プロセスを起動するときは、クラスタ接続文字列の一部としてノード ID を使用しないでください。

`--ndb-cluster-connection-pool` を使用する際に接続文字列にノード ID を設定すると、SQL ノードのクラスタへの接続試行時に、ノード ID の割り当てエラーが発生します。

- `--ndb-cluster-connection-pool-nodeids=list`

コマンド行形式	<code>--ndb-cluster-connection-pool-nodeids</code>
システム変数	<code>ndb_cluster_connection_pool_nodeids</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Set
デフォルト値	

SQL ノードで使用されるクラスタに接続するためのノード ID のカンマ区切りリストを指定します。このリストのノード数は、`--ndb-cluster-connection-pool` オプションに設定された値と同じである必要があります。

- `--ndb-blob-read-batch-bytes=bytes`

コマンド行形式	<code>--ndb-blob-read-batch-bytes</code>
システム変数	<code>ndb_blob_read_batch_bytes</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	65536
最小値	0

最大値	4294967295
-----	------------

このオプションを使用すると、NDB Cluster アプリケーションで **BLOB** データ読み取りをバッチ処理するためのサイズ (バイト単位) を設定できます。現在のトランザクション内で読み取られる **BLOB** データの量がこのバッチサイズを超えると、保留中のすべての **BLOB** 読み取り操作がただちに実行されます。

このオプションの最大値は 4294967295 です。デフォルトは 65536 です。0 に設定すると、**BLOB** 読み取りのバッチ化を無効にします。

注記

NDB API アプリケーションでは、`setMaxPendingBlobReadBytes()` および `getMaxPendingBlobReadBytes()` メソッドを使用すると **BLOB** 書き込みのバッチ化を制御できます。

- `--ndb-blob-write-batch-bytes=bytes`

コマンド行形式	<code>--ndb-blob-write-batch-bytes</code>
システム変数	<code>ndb_blob_write_batch_bytes</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	65536
最小値	0
最大値	4294967295

このオプションを使用すると、NDB Cluster アプリケーションで **BLOB** データ書き込みをバッチ処理するためのサイズ (バイト単位) を設定できます。現在のトランザクション内で書き込まれる **BLOB** データの量がこのバッチサイズを超えると、保留中のすべての **BLOB** 書き込み操作がただちに実行されます。

このオプションの最大値は 4294967295 です。デフォルトは 65536 です。0 に設定すると、**BLOB** 書き込みのバッチ化を無効にします。

注記

NDB API アプリケーションでは、`setMaxPendingBlobWriteBytes()` および `getMaxPendingBlobWriteBytes()` メソッドを使用すると **BLOB** 書き込みのバッチ化を制御できます。

- `--ndb-connectstring=connection_string`

コマンド行形式	<code>--ndb-connectstring</code>
型	文字列

このオプションは、**NDBCLUSTER** ストレージエンジンの使用時に、クラスタ構成データを配信する管理サーバーを指定します。構文については、[セクション23.3.3.3「NDB Cluster 接続文字列」](#)を参照してください。

- `--ndb-default-column-format={FIXED|DYNAMIC}`

コマンド行形式	<code>--ndb-default-column-format={FIXED DYNAMIC}</code>
システム変数	<code>ndb_default_column_format</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ

型	列挙
デフォルト値	FIXED
有効な値	FIXED DYNAMIC

新しいテーブルのデフォルトの `COLUMN_FORMAT` および `ROW_FORMAT` を設定します (セクション 13.1.20 「CREATE TABLE ステートメント」 を参照)。デフォルトは `FIXED` です。

- `--ndb-deferred-constraints=[0|1]`

コマンド行形式	<code>--ndb-deferred-constraints</code>
システム変数	<code>ndb_deferred_constraints</code>
スコープ	グローバル、セッション
動的	はい
<code>SET_VAR</code> ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	1

一意のインデックスに対する制約チェックがサポートされている場合に、チェックをコミット時まで遅延するかどうかを制御します。0 がデフォルトです。

このオプションは通常、NDB Cluster または NDB Cluster レプリケーションの操作には必要なく、主にテストで使用することを目的としています。

- `--ndb-schema-dist-timeout=#`

コマンド行形式	<code>--ndb-schema-dist-timeout=#</code>
導入	8.0.17-ndb-8.0.17
システム変数	<code>ndb_schema_dist_timeout</code>
スコープ	グローバル
動的	いいえ
<code>SET_VAR</code> ヒントの適用	いいえ
型	Integer
デフォルト値	120
最小値	5
最大値	1200
単位	seconds

この `mysqld` がスキーマ操作の完了を待機する最大時間 (秒) を指定します。この時間が経過すると、スキーマ操作はタイムアウトとしてマークされます。

- `--ndb-distribution=[KEYHASH|LINHASH]`

コマンド行形式	<code>--ndb-distribution={KEYHASH LINHASH}</code>
システム変数	<code>ndb_distribution</code>
スコープ	グローバル
動的	はい

SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	KEYHASH
有効な値	LINHASH KEYHASH

NDB テーブルのデフォルトの分配方法を制御します。KEYHASH (キーハッシュ) または LINHASH (線形ハッシュ) のいずれかに設定できます。KEYHASH がデフォルトです。

- `--ndb-log-apply-status`

コマンド行形式	<code>--ndb-log-apply-status[={OFF ON}]</code>
システム変数	<code>ndb_log_apply_status</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

レプリカ `mysqld` は、即時ソースから受信した更新を、ソースのサーバー ID ではなく独自のサーバー ID を使用して、独自のバイナリログ内の `mysql.ndb_apply_status` テーブルに記録します。循環レプリケーションまたはチェーンレプリケーション設定では、このような更新を、現在の `mysqld` のレプリカとして構成されている MySQL サーバーの `mysql.ndb_apply_status` テーブルに伝播できます。

チェーンレプリケーション設定でこのオプションを使用すると、ダウンストリーム (レプリカ) クラスタは、すべてのアップストリームコンプリタ (ソース) に対して相対的な位置を認識できます。

循環レプリケーション設定では、このオプションを使用すると、`ndb_apply_status` テーブルへの変更によって回路全体が完了し、最終的に元の NDB Cluster に伝播されます。これにより、レプリケーションソースとして機能するクラスタは、その変更 (エポック) が円の他のクラスタに適用された時期を確認することもできます。

このオプションは、MySQL サーバーが `--ndbcluster` オプションを使用して起動されていない場合は無効になります。

- `--ndb-log-empty-epochs=[ON|OFF]`

コマンド行形式	<code>--ndb-log-empty-epochs[={OFF ON}]</code>
システム変数	<code>ndb_log_empty_epochs</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

`log_slave_updates` が有効な場合でも、`ndb_apply_status` および `ndb_binlog_index` テーブルに変更が書き込まれなかったエポックが発生します。

このオプションはデフォルトで無効になっています。`--ndb-log-empty-epochs` を無効にすると、変更がなかったエポックトランザクションがバイナリログに書き込まれなくなりますが、`ndb_binlog_index` には空のエポックの行も書き込まれます。

`--ndb-log-empty-epochs=1` では、バイナリログのサイズとは無関係に `ndb_binlog_index` テーブルのサイズが増加するため、ユーザーは、クラスタがアイドル状態であると予想される場合でも、このテーブルの増加を管理する準備をするようにしてください。

- `--ndb-log-empty-update=[ON|OFF]`

コマンド行形式	<code>--ndb-log-empty-update[={OFF ON}]</code>
システム変数	<code>ndb_log_empty_update</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

`log_slave_updates` が有効な場合、変更を生成しなかった更新が `ndb_apply_status` および `ndb_binlog_index` テーブルに書き込まれます。

デフォルトでは、このオプションは無効 (OFF) です。 `--ndb-log-empty-update` を無効にすると、変更のない更新はバイナリログに書き込まれません。

- `--ndb-log-exclusive-reads=[0|1]`

コマンド行形式	<code>--ndb-log-exclusive-reads[={OFF ON}]</code>
システム変数	<code>ndb_log_exclusive_reads</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	0

このオプションを使用してサーバーを起動すると、主キーの読み取りが排他ロックで記録されるため、NDB Cluster レプリケーションの競合検出および読み取りの競合に基づく解決が可能になります。 `ndb_log_exclusive_reads` システム変数の値を 1 または 0 に設定すると、このロックを実行時に有効または無効にすることもできます。0 (ロックを無効化) がデフォルトです。

詳細は、[読み取り競合の検出と解決](#)を参照してください。

- `--ndb-log-fail-terminate`

コマンド行形式	<code>--ndb-log-fail-terminate</code>
導入	8.0.21-ndb-8.0.21
システム変数	<code>ndb_log_fail_terminate</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	FALSE

このオプションが指定され、見つかったすべての行イベントの完全なロギングが不可能な場合、`mysqld` プロセスは終了します。

- `--ndb-log-orig`

コマンド行形式	<code>--ndb-log-orig[={OFF ON}]</code>
システム変数	<code>ndb_log_orig</code>
スコープ	グローバル

動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

発信元サーバーの ID とエポックを `ndb_binlog_index` テーブルに記録します。

注記

これにより、特定のエポックが、元のエポックごとに 1 つずつ、`ndb_binlog_index` に複数の行を持つことができます。

詳細は、[セクション23.6.4「NDB Cluster レプリケーションスキーマおよびテーブル」](#)を参照してください。

• `--ndb-log-transaction-id`

コマンド行形式	<code>--ndb-log-transaction-id[={OFF ON}]</code>
システム変数	<code>ndb_log_transaction_id</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

レプリカ `mysqld` が NDB トランザクション ID をバイナリログの各行に書き込みます。デフォルト値は `FALSE` です。

このオプションはメインラインの MySQL Server 8.0 ではサポートされません。 `NDB$EPOCH_TRANS()` 関数を使用して NDB Cluster レプリケーションの競合検出および解決を有効にする必要があります (`NDB$EPOCH_TRANS()` を参照)。詳細は、[セクション23.6.11「NDB Cluster レプリケーションの競合解決」](#)を参照してください。

`--ndb-log-transaction-id` を使用する場合は、非推奨の `log_bin_use_v1_row_events` システム変数 (デフォルトは `OFF`) を `ON` に設定しないでください。

• `--ndb-log-update-minimal`

コマンド行形式	<code>--ndb-log-update-minimal[={OFF ON}]</code>
システム変数	<code>ndb_log_update_minimal</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

ビフォアイメージに主キー値のみを書き込み、アフターイメージに変更されたカラムのみを書き込むことで、最小限の方法で更新を記録します。これにより、`NDB` 以外のストレージエンジンにレプリケートする場合に互換性の問題が発生する可能性があります。

• `--ndb-mgmd-host=host[:port]`

コマンド行形式	<code>--ndb-mgmd-host=host_name[:port_num]</code>
型	文字列

デフォルト値	localhost:1186
--------	----------------

プログラムに接続される単一の管理サーバーのホストとポート番号の設定に使用できます。プログラムの接続情報として複数の管理サーバーのノード ID または参照 (あるいはその両方) が必要な場合は、代わりに `--ndb-connectstring` オプションを使用します。

- `--ndb-nodeid=#`

コマンド行形式	<code>--ndb-nodeid=#</code>
ステータス変数	<code>Ndb_cluster_node_id</code>
スコープ	グローバル
動的	いいえ
型	Integer
最小値	1
最大値	255
最大値	63

NDB Cluster でこの MySQL サーバーノード ID を設定します。

`--ndb-nodeid` オプションは、`--ndb-connectstring` で設定されたノード ID を (2 つのオプションの使用順序に関係なく) オーバーライドします。

また、`--ndb-nodeid` を使用する場合は、`config.ini` の `[mysqld]` または `[api]` セクションに一致するノード ID が存在するか、このファイルに「オープンな」`[mysqld]` または `[api]` セクション (つまり、`Nodetid` または `Id` パラメータが指定されていないセクション) が存在する必要があります。これは、ノード ID が接続文字列の一部として指定されている場合にも当てはまります。

ノード ID は、その指定方法に関係なく、`SHOW STATUS` の出力にグローバルステータス変数 `Ndb_cluster_node_id` の値として表示され、`SHOW ENGINE NDBCLUSTER STATUS` の出力の `connection` 行に `cluster_node_id` として表示されます。

NDB Cluster SQL ノードのノード ID の詳細は、[セクション23.3.3.7「NDB Cluster での SQL およびその他の API ノードの定義」](#) を参照してください。

- `--ndbinfo={ON|OFF|FORCE}`

コマンド行形式	<code>--ndbinfo=[value]</code> (≥ 8.0.13-ndb-8.0.13)
導入	8.0.13-ndb-8.0.13
型	列挙
デフォルト値	ON
有効な値	ON OFF FORCE

`ndbinfo` 情報データベースのプラグインを有効にします。デフォルトでは、`NDBCLUSTER` が有効になっている場合は常に ON です。

- `--ndb-optimization-delay=milliseconds`

コマンド行形式	<code>--ndb-optimization-delay=#</code>
システム変数	<code>ndb_optimization_delay</code>
スコープ	グローバル
動的	はい

SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	10
最小値	0
最大値	100000

NDB テーブルに対する `OPTIMIZE TABLE` ステートメントによる行セット間で待機するミリ秒数を設定します。デフォルトは 10 です。

- `--ndb-transid-mysql-connection-map=state`

コマンド行形式	<code>--ndb-transid-mysql-connection-map[=state]</code>
型	列挙
デフォルト値	ON
有効な値	ON OFF FORCE

`INFORMATION_SCHEMA` データベースの `ndb_transid_mysql_connection_map` テーブルを処理するプラグインを有効または無効にします。ON、OFF、FORCE のいずれかの値を取ります。ON (デフォルト) では、プラグインが有効になります。OFF では、プラグインが無効になり、`ndb_transid_mysql_connection_map` にアクセスできなくなります。FORCE では、プラグインのロードと起動に失敗した場合に MySQL Server が起動しなくなります。

`ndb_transid_mysql_connection_map` テーブルプラグインが実行されているかどうかを確認するには、`SHOW PLUGINS` の出力をチェックします。

- `--ndb-wait-connected=seconds`

コマンド行形式	<code>--ndb-wait-connected=#</code>
システム変数	<code>ndb_wait_connected</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	30
最小値	0
最大値	31536000

このオプションは、MySQL クライアント接続を受け入れる前に、NDB Cluster 管理およびデータノードへの接続が確立されるのを MySQL サーバーが待機する期間を設定します。この時間は秒単位で指定します。デフォルト値は 30 です。

- `--ndb-wait-setup=seconds`

コマンド行形式	<code>--ndb-wait-setup=#</code>
システム変数	<code>ndb_wait_setup</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer

デフォルト値	30
デフォルト値	30
デフォルト値	15
デフォルト値	15
最小値	0
最大値	31536000

この変数は、MySQL サーバーが **NDB** ストレージエンジンのセットアップが完了されるまで待機しタイムアウトして、**NDB** を使用不可として扱うまでの時間を示します。この時間は秒単位で指定します。デフォルト値は **30** です。

- `--skip-ndbcluster`

コマンド行形式	<code>--skip-ndbcluster</code>
---------	--------------------------------

NDBCLUSTER ストレージエンジンを無効にします。これは、**NDBCLUSTER** ストレージエンジンのサポート付きでビルドされたバイナリのデフォルトです。サーバーは、`--ndbcluster` オプションが明示的に指定された場合のみ、このストレージエンジン用にメモリーおよびその他のリソースを割り当てます。例については、[セクション 23.3.1 「NDB Cluster のクイックテスト設定」](#)を参照してください。

NDB Cluster システム変数

このセクションでは、NDB Cluster および **NDB** ストレージエンジンに固有の MySQL サーバースystem変数について詳しく説明します。NDB Cluster に固有でないシステム変数については、[セクション 5.1.8 「サーバースystem変数」](#)を参照してください。システム変数の使用に関する一般情報は、[セクション 5.1.9 「システム変数の使用」](#)を参照してください。

- `ndb_autoincrement_prefetch_sz`

コマンド行形式	<code>--ndb-autoincrement-prefetch-sz=#</code>
システム変数	<code>ndb_autoincrement_prefetch_sz</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値 (≥ 8.0.19-ndb-8.0.19)	512
デフォルト値 (≤ 8.0.18-ndb-8.0.18)	1
最小値	1
最大値	65536

自動インクリメントカラムのギャップの可能性を指定します。これを最小化するには、**1** に設定します。最適化のために高い値に設定すると、挿入は高速になりますが、挿入のバッチで連続する自動インクリメント番号が使用される可能性は低くなります。

この変数は、ステートメント間でフェッチされる **AUTO_INCREMENT** ID の数にのみ影響します。特定のステートメント内では、一度に少なくとも **32** 個の ID が取得されます。

重要

この変数は、**INSERT ... SELECT** を使用して実行される挿入には影響しません。

- `ndb_cache_check_time`

コマンド行形式	<code>--ndb-cache-check-time=#</code>	3667
---------	---------------------------------------	------

非推奨	はい
システム変数	ndb_cache_check_time
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0

MySQL クエリーキャッシュによる NDB Cluster SQL ノードのチェック間に経過するミリ秒数。これを 0 (デフォルトおよび最小値) に設定すると、クエリーキャッシュですべてのクエリーの妥当性がチェックされます。

この変数の推奨最大値は 1000 です。これは、チェックが毎秒 1 回行われることを意味します。より大きな値にすると、チェックが実行され、場合によっては異なる SQL ノードでの更新のために無効になる頻度が少なくなります。これを 2000 より大きな値に設定することは、通常、望ましくありません。

注記

クエリーキャッシュ [ndb_cache_check_time](#) は MySQL 5.7 で非推奨になりました。クエリーキャッシュは MySQL 8.0 で削除されました。

- [ndb_clear_apply_status](#)

コマンド行形式	<code>--ndb-clear-apply-status[={OFF ON}]</code>
システム変数	ndb_clear_apply_status
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

デフォルトでは、[RESET SLAVE](#) を実行すると NDB Cluster レプリカはその [ndb_apply_status](#) テーブルからすべての行をパーズします。これを無効にするには、`ndb_clear_apply_status=OFF` を設定します。

- [ndb_conflict_role](#)

コマンド行形式	<code>--ndb-conflict-role=value</code>
導入	8.0.23-ndb-8.0.23
システム変数	ndb_conflict_role
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	NONE
有効な値	NONE PRIMARY SECONDARY

PASS

循環 (「active-active」) レプリケーション設定でのこの SQL ノード (および NDB Cluster) の役割を決定します。 `ndb_slave_conflict_role` には、 `PRIMARY`、 `SECONDARY`、 `PASS` または `NULL` (デフォルト) のいずれかの値を指定できます。 `ndb_slave_conflict_role` を変更する前に、レプリカ SQL スレッドを停止する必要があります。また、これを `PASS` と `PRIMARY` または `SECONDARY` のいずれかの間で直接変更することはできません。変更する場合は、SQL スレッドが停止していることを確認してから、先に `SET @@GLOBAL.ndb_slave_conflict_role = 'NONE'` を実行する必要があります。

NDB 8.0.23 の時点で非推奨になった `ndb_slave_conflict_role` は、この変数に置き換えられます。

詳細は、 [セクション23.6.11「NDB Cluster レプリケーションの競合解決」](#) を参照してください。

- `ndb_data_node_neighbour`

コマンド行形式	<code>--ndb-data-node-neighbour=#</code>
システム変数	<code>ndb_data_node_neighbour</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	255

「nearest」 データノードの ID を設定します。つまり、SQL または API ノードと同じホストで実行されているノードではなく、優先される非ローカルデータノードがトランザクションの実行に選択されます。これは、完全にレプリケートされたテーブルにアクセスするときに、テーブルのローカルコピーが可能なかぎり常に使用されるように、このデータノード上でアクセスするために使用されます。これは、トランザクションのヒントを提供するためにも使用できます。

これにより、物理的に同じホスト上の他のノードよりも近く、ネットワークスループットが高いノードの場合に、データアクセス時間を改善できます。

詳細は、 [セクション13.1.20.11「NDB_TABLE オプションの設定」](#) を参照してください。

注記

NDB API アプリケーションで使用するための同等の方法 `set_data_node_neighbour()` が提供されています。

- `ndb_dbg_check_shares`

コマンド行形式	<code>--ndb-dbg-check-shares=#</code>
導入	8.0.13-ndb-8.0.13
システム変数	<code>ndb_dbg_check_shares</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	1

1に設定した場合は、リンガリングしているシェアがないことを確認します。デバッグビルドでのみ使用できません。

NDB 8.0.13 に追加されました。

- [ndb_default_column_format](#)

コマンド行形式	--ndb-default-column-format={FIXED DYNAMIC}
システム変数	ndb_default_column_format
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	FIXED
有効な値	FIXED DYNAMIC

新しいテーブルのデフォルトの `COLUMN_FORMAT` および `ROW_FORMAT` を設定します (セクション 13.1.20 「CREATE TABLE ステートメント」を参照)。デフォルトは `FIXED` です。

- [ndb_deferred_constraints](#)

コマンド行形式	--ndb-deferred-constraints=#
システム変数	ndb_deferred_constraints
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	1

制約チェックがサポートされる場合に、それを遅延するかどうかを制御します。0がデフォルトです。

この変数は通常、NDB Cluster または NDB Cluster レプリケーションの操作には必要なく、主にテストで使用することを目的としています。

- [ndb_distribution](#)

コマンド行形式	--ndb-distribution={KEYHASH LINHASH}
システム変数	ndb_distribution
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	KEYHASH
有効な値	LINHASH

KEYHASH

NDB テーブルのデフォルトの分配方法を制御します。KEYHASH (キーハッシュ) または LINHASH (線形ハッシュ) のいずれかに設定できます。KEYHASH がデフォルトです。

- [ndb_eventbuffer_free_percent](#)

コマンド行形式	<code>--ndb-eventbuffer-free-percent=#</code>
システム変数	ndb_eventbuffer_free_percent
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	20
最小値	1
最大値	99

イベントバッファ (`ndb_eventbuffer_max_alloc`) に割り当てられる最大メモリの割合を設定します。この値は、最大値に達した後、再度バッファを開始する前にイベントバッファで使用可能である必要があります。

- [ndb_eventbuffer_max_alloc](#)

コマンド行形式	<code>--ndb-eventbuffer-max-alloc=#</code>
システム変数	ndb_eventbuffer_max_alloc
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0
最小値	0
最大値	4294967295

NDB API によるイベントのバッファリングに割り当てる可能な最大メモリー量 (バイト単位) を設定します。0 はデフォルトで、制限が適用されません。

- [ndb_extra_logging](#)

コマンド行形式	<code>ndb_extra_logging=#</code>
システム変数	ndb_extra_logging
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1

この変数は、NDB ストレージエンジンに固有の情報を MySQL エラーログに記録できるようにします。

この変数を 0 に設定すると、MySQL エラーログに書き込まれる NDB 固有の情報がトランザクション処理に関するものだけになります。0 より大きく 10 より小さい値に設定すると、NDB のテーブルスキーマイベントと接続イベント、競合解決の使用中の有無、および NDB のその他のエラーと情報も記録されます。値を 10 以上に設定する

と、NDB の内部に関する情報 (クラスタノード間のデータ分配の進行状況など) も MySQL エラーログに書き込まれます。デフォルトは 1 です。

- [ndb_force_send](#)

コマンド行形式	<code>--ndb-force-send[={OFF ON}]</code>
システム変数	ndb_force_send
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

ほかのスレッドを待機せずに、バッファを NDB にただちに送信します。デフォルトは ON です。

- [ndb_fully_replicated](#)

コマンド行形式	<code>--ndb-fully-replicated[={OFF ON}]</code>
システム変数	ndb_fully_replicated
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

新しい NDB テーブルを完全にレプリケートするかどうかを決定します。この設定は、CREATE TABLE ステートメントまたは ALTER TABLE ステートメントで `COMMENT="NDB_TABLE=FULLY_REPLICATED=..."` を使用して、個々のテーブルに対してオーバーライドできます。構文およびその他の情報は、[セクション 13.1.20.11 「NDB_TABLE オプションの設定」](#) を参照してください。

- [ndb_index_stat_enable](#)

コマンド行形式	<code>--ndb-index-stat-enable[={OFF ON}]</code>
システム変数	ndb_index_stat_enable
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

クエリーの最適化で NDB インデックス統計を使用します。デフォルトは ON です。

- [ndb_index_stat_option](#)

コマンド行形式	<code>--ndb-index-stat-option=value</code>
システム変数	ndb_index_stat_option
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	文字列

デフォルト値	<code>loop_checkon=1000ms,loop_idle=1000ms,loop_busy=100ms,update_batch=1,read_batch=4,idle_batch=32,check_batch=32,check_delay=1m,delete_batch=8,clean_delay=0,error_batch=4,error_delay=1m,evict_batch=8,evict_delay=1m,cache_limit=32M,cache_lowpct=90</code>
--------	--

この変数は、NDB インデックス統計の生成に関する調整オプションを提供するために使用されます。リストはオプション名と値のカンマ区切りの名前と値のペアで構成され、このリストに空白文字を含めることはできません。

`ndb_index_stat_option` の設定時に使用していなかったオプションは、デフォルト値から変更できません。たとえば、`ndb_index_stat_option = 'loop_idle=1000ms,cache_limit=32M'` のように設定できます。

時間値の末尾には、オプションで **h** (時間)、**m** (分) または **s** (秒) を付けることができます。ミリ秒の値は、オプションで **ms** を使用して指定できます。ミリ秒の値は、**h**、**m** または **s** を使用して指定できません。) 整数値のサフィクスとして **K**、**M**、または **G** を使用できます。

この変数を使用して設定できるオプションの名前を次の表に示します。この表には、オプションの簡単な説明、デフォルト値、および (適用可能な場合は) 最小値と最大値も示します。

表 23.20 `ndb_index_stat_option` のオプションと値

名前	説明	デフォルト/単位	最小/最大
<code>loop_enable</code>		1000 ms	0/4G
<code>loop_idle</code>	アイドル時のスリープ時間	1000 ms	0/4G
<code>loop_busy</code>	追加作業が待機しているときのスリープ時間	100 ms	0/4G
<code>update_batch</code>		1	0/4G
<code>read_batch</code>		4	1/4G
<code>idle_batch</code>		32	1/4G
<code>check_batch</code>		8	1/4G
<code>check_delay</code>	新しい統計をチェックする頻度	10 m	1/4G
<code>delete_batch</code>		8	0/4G
<code>clean_delay</code>		1 m	0/4G
<code>error_batch</code>		4	1/4G
<code>error_delay</code>		1 m	1/4G
<code>evict_batch</code>		8	1/4G
<code>evict_delay</code>	LRU キャッシュを削除します (読み取り時から)	1 m	0/4G
<code>cache_limit</code>	この <code>mysqld</code> がキャッシュされたインデックス統計のために使用するメモリの最大量 (バイト単位)。これを超えたときにキャッシュをクリーンアップします。	32 M	0/4G
<code>cache_lowpct</code>		90	0/100
<code>zero_total</code>	これを 1 に設定すると、 <code>ndb_index_stat_status</code> のすべての累積カウンタが 0 にリセットされます。これを実行すると、このオプションの値も 0 にリセットされます。	0	0/1

- [ndb_join_pushdown](#)

システム変数	ndb_join_pushdown
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

この変数は、NDB テーブル上の結合が NDB カーネル (データノード) にプッシュダウンされるかどうかを制御します。以前は、SQL ノードによる NDB の複数アクセスにより結合が処理されていましたが、[ndb_join_pushdown](#) を有効にすると、プッシュ可能な結合のすべてがデータノードに送信されます。この場合、結合はデータノード間で分配され、データの複数のコピーに対して並列で実行され、単一のマージされた結果が `mysqld` に返されます。これにより、このような結合を処理するために必要な SQL ノードとデータノード間のラウンドトリップの回数を大幅に減らすことができます。

デフォルトでは、[ndb_join_pushdown](#) は有効になっています。

NDB プッシュダウン結合の条件。結合をプッシュ可能にするには、次の条件を満たす必要があります。

1. 比較できるのはカラムのみであり、結合されるすべてのカラムが完全に同じデータ型を使用している必要があります。つまり、`INT` カラムと `BIGINT` カラムの結合もプッシュダウンできません。

以前は、`t1.a = t2.a + constant` などの式をプッシュダウンできませんでした。この制限は NDB 8.0.18 以降で解除されています。比較するカラムに対する操作の結果は、カラム自体と同じ型になる必要があります。

NDB 8.0.18 以降では、同じテーブルのカラムを比較する式をプッシュダウンできます。カラム (またはこれらのカラムに対する操作の結果) は、同じ符号、長さ、文字セットと照合順序、精度およびスケール (該当する場合) を含む、まったく同じタイプである必要があります。

2. `BLOB` または `TEXT` カラムを参照するクエリーはサポートされません。
3. 明示的なロックはサポートされません。ただし、NDB ストレージエンジンの特徴である暗黙的な行ベースのロックは適用されます。

これは、`FOR UPDATE` を使用する結合をプッシュダウンできないことを意味します。

4. 結合をプッシュダウンするには、`ref`、`eq_ref`、または `const` アクセスメソッドのいずれか、またはこれらのメソッドの組み合わせを使用して結合の子テーブルにアクセスする必要があります。

外部結合された子テーブルは、`eq_ref` のみを使用してプッシュできます。

プッシュされた結合のルートが `eq_ref` または `const` である場合は、`eq_ref` によって結合された子テーブルのみを追加できます。(`ref` によって結合されたテーブルは、プッシュされた結合の別のルートになる可能性があります。)

クエリーオプティマイザで候補となる子テーブルに `Using join cache` を指定した場合は、そのテーブルは子としてプッシュできません。ただし、プッシュされたテーブルの別のセットのルートになることはできます。

5. `[LINEAR] HASH`、`LIST`、または `RANGE` によって明示的にパーティション化されたテーブルを参照する結合は、現在のところプッシュダウンできません。

特定の結合をプッシュダウンできるかどうかを確認するには、それを `EXPLAIN` でチェックします。結合をプッシュダウンできる場合は、この例に示すように、出力の `Extra` カラムに `pushed join` への参照が表示されます。

```
mysql> EXPLAIN
-> SELECT e.first_name, e.last_name, t.title, d.dept_name
-> FROM employees e
-> JOIN dept_emp de ON e.emp_no=de.emp_no
-> JOIN departments d ON d.dept_no=de.dept_no
-> JOIN titles t ON e.emp_no=t.emp_no\G
```

```

***** 1. row *****
  id: 1
  select_type: SIMPLE
  table: d
  type: ALL
  possible_keys: PRIMARY
  key: NULL
  key_len: NULL
  ref: NULL
  rows: 9
  Extra: Parent of 4 pushed join@1
***** 2. row *****
  id: 1
  select_type: SIMPLE
  table: de
  type: ref
  possible_keys: PRIMARY,emp_no,dept_no
  key: dept_no
  key_len: 4
  ref: employees.d.dept_no
  rows: 5305
  Extra: Child of 'd' in pushed join@1
***** 3. row *****
  id: 1
  select_type: SIMPLE
  table: e
  type: eq_ref
  possible_keys: PRIMARY
  key: PRIMARY
  key_len: 4
  ref: employees.de.emp_no
  rows: 1
  Extra: Child of 'de' in pushed join@1
***** 4. row *****
  id: 1
  select_type: SIMPLE
  table: t
  type: ref
  possible_keys: PRIMARY,emp_no
  key: emp_no
  key_len: 4
  ref: employees.de.emp_no
  rows: 19
  Extra: Child of 'e' in pushed join@1
4 rows in set (0.00 sec)

```

注記

内部結合されたテーブルが `ref` によって結合され、かつその結果がソートされたインデックスによって順序付けまたはグループ化された場合、このインデックスはソートされた行を提供できず、ソートされた一時ファイルへの書き込みが強制されます。

プッシュされた結合の動作については、追加の情報源が 2 つあります。

1. ステータス変数

`Ndb_pushed_queries_defined`、`Ndb_pushed_queries_dropped`、`Ndb_pushed_queries_executed`、および `Ndb_pushed_reads`。

2. DBSPJ カーネルブロックに属する `ndbinfo.counters` テーブル内のカウンタ。

- `ndb_log_apply_status`

コマンド行形式	<code>--ndb-log-apply-status[={OFF ON}]</code>
システム変数	<code>ndb_log_apply_status</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ

型	Boolean
デフォルト値	OFF

--ndb-log-apply-status オプションを使用してサーバーが起動されたかどうかを示す読み取り専用の変数。

- [ndb_log_bin](#)

コマンド行形式	--ndb-log-bin[={OFF ON}]
システム変数	ndb_log_bin
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値 (≥ 8.0.16-ndb-8.0.16)	OFF
デフォルト値 (≤ 8.0.15-ndb-8.0.15)	ON

NDB テーブルの更新がバイナリログに書き込まれるようになります。log_bin を使用しているサーバーでバイナリロギングがまだ有効になっていない場合、この変数の設定は無効です。NDB 8.0.16 以降では、ndb_log_bin のデフォルトは 0 (FALSE) です。

- [ndb_log_binlog_index](#)

コマンド行形式	--ndb-log-binlog-index[={OFF ON}]
システム変数	ndb_log_binlog_index
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

エポックとバイナリログ内の位置とのマッピングが ndb_binlog_index テーブルに挿入されるようになります。サーバーのバイナリロギングがまだ log_bin を使用して有効になっていない場合は、この変数を設定しても無効になります。(また、ndb_log_bin も無効にしないでください。) ndb_log_binlog_index のデフォルトは 1 (ON) です。通常、本番環境でこの値を変更する必要はありません。

- [ndb_log_empty_epochs](#)

コマンド行形式	--ndb-log-empty-epochs[={OFF ON}]
システム変数	ndb_log_empty_epochs
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

この変数を 0 に設定すると、変更がなかったエポックトランザクションがバイナリログに書き込まれなくなりますが、ndb_binlog_index には空のエポックの行も書き込まれます。

- [ndb_log_empty_update](#)

3676	コマンド行形式	--ndb-log-empty-update[={OFF ON}]
------	---------	-----------------------------------

システム変数	ndb_log_empty_update
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

この変数が ON (1) に設定されている場合、[log_slave_updates](#) が有効になっていても、変更のない更新トランザクションはバイナリログに書き込まれます。

- [ndb_log_exclusive_reads](#)

コマンド行形式	<code>--ndb-log-exclusive-reads[={OFF ON}]</code>
システム変数	ndb_log_exclusive_reads
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	0

この変数は、主キーの読み取りを排他ロックで記録するかどうかを決定します。これにより、NDB Cluster レプリケーションの競合検出および読み取りの競合に基づく解決が可能になります。このロックを有効にするには、[ndb_log_exclusive_reads](#) の値を 1 に設定します。0 (このようなロックを無効にする) がデフォルトです。

詳細は、[読み取り競合の検出と解決](#)を参照してください。

- [ndb_log_orig](#)

コマンド行形式	<code>--ndb-log-orig[={OFF ON}]</code>
システム変数	ndb_log_orig
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

発信元サーバーの ID とエポックが [ndb_binlog_index](#) テーブルに記録されるかどうかを示します。 `--ndb-log-orig` サーバーオプションを使用して設定します。

- [ndb_log_transaction_id](#)

システム変数	ndb_log_transaction_id
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

この読み取り専用のブールシステム変数は、レプリカ `mysqld` が NDB トランザクション ID をバイナリログに書き込むかどうかを示します (`NDB$EPOCH_TRANS()` 競合検出で「active-active」 NDB Cluster レプリケーションを使用するために必要です)。この設定を変更するには、`--ndb-log-transaction-id` オプションを使用します。

メインラインの MySQL Server 8.0 では `ndb_log_transaction_id` はサポートされません。

詳細は、[セクション23.6.11「NDB Cluster レプリケーションの競合解決」](#)を参照してください。

- [ndb_metadata_check](#)

コマンド行形式	<code>--ndb-metadata-check[={OFF ON}]</code>
導入	8.0.16-ndb-8.0.16
システム変数	ndb_metadata_check
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

NDB 8.0.16 以降、NDB はバックグラウンドスレッドを使用して、MySQL データディクショナリと比較して `ndb_metadata_check_interval` 秒ごとにメタデータの変更をチェックします。このメタデータ変更検出スレッドは、`ndb_metadata_check` を OFF に設定することで無効にできます。スレッドはデフォルトで有効になっています。

- [ndb_metadata_check_interval](#)

コマンド行形式	<code>--ndb-metadata-check-interval=#</code>
導入	8.0.16-ndb-8.0.16
システム変数	ndb_metadata_check_interval
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	60
最小値	0
最大値	31536000
単位	seconds

NDB 8.0.16 以降では、NDB はメタデータ変更検出スレッドをバックグラウンドで実行して、NDB デクショナリが MySQL データディクショナリに関していつ変更されたかを判断します。デフォルトでは、このようなチェックの間隔は 60 秒です。これは、`ndb_metadata_check_interval` の値を設定することで調整できます。スレッドを有効または無効にするには、`ndb_metadata_check` を使用します。

- [ndb_metadata_sync](#)

導入	8.0.19-ndb-8.0.19
システム変数	ndb_metadata_sync
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean

デフォルト値	false
--------	-------

この変数を設定すると、変更モニタースレッドは `ndb_metadata_check` または `ndb_metadata_check_interval` に設定された値をオーバーライドし、連続した変更検出の期間を入力します。スレッドは、検出する変更がなくなったことを確認すると、バイナリロギングスレッドが検出されたすべてのオブジェクトの同期を終了するまで停止します。その後、`ndb_metadata_sync` は `false` に設定され、変更モニタースレッドは `ndb_metadata_check` および `ndb_metadata_check_interval` の設定によって決定された動作に戻ります。

NDB 8.0.22 以降では、この変数を `true` に設定すると、除外されたオブジェクトのリストがクリアされ、`false` に設定すると、再試行されるオブジェクトのリストがクリアされます。

- `ndb_optimized_node_selection`

コマンド行形式	<code>--ndb-optimized-node-selection=#</code>
システム変数	<code>ndb_optimized_node_selection</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	3
最小値	0
最大値	3

最適化されたノードの選択には、ここの示す 2 つの形式があります。

1. SQL ノードは、近接性を使用してトランザクションコーディネータを指定します。つまり、その SQL ノードから「もっとも近い」データノードがトランザクションコーディネータとして選択されます。この目的のために、SQL ノードとの共有メモリー接続を持つデータノードは、SQL ノードへの「closest」とみなされます。次に近い (近接度が低い順に): `localhost` への TCP 接続に続いて、`localhost` 以外のホストからの TCP 接続。
2. SQL スレッドは、分布の認識を使用してデータノードを選択します。つまり、特定のトランザクションの最初のステートメントによってアクセスされるクラスタパーティションを収容するデータノードが、トランザクション全体のトランザクションコーディネータとして使用されます。(これは、トランザクションの最初のステートメントが 1 つのクラスタパーティションにしかアクセスしない場合にのみ有効です。)

このオプションは、0、1、2、または 3 のいずれかの整数値を取ります。3 がデフォルトです。これらの値は、ノード選択に次のように影響します。

- 0: ノード選択は最適化されません。SQL スレッドが次のデータノードに進む前に、各データノードがトランザクションコーディネータとして 8 回ずつ採用されます。
- 1: SQL ノードへの近接性を使用してトランザクションコーディネータが指定されます。
- 2: 分布の認識を使用してトランザクションコーディネータが選択されます。ただし、トランザクションの最初のステートメントが複数のクラスタパーティションにアクセスする場合は、SQL ノードはこのオプションを 0 に設定したときに見られるラウンドロビンの動作に戻ります。
- 3: トランザクションコーディネータを指定するために分布の認識を使用できる場合は、それが使用されます。そうでない場合は、近接性を使用してトランザクションコーディネータが選択されます。(これはデフォルトの動作です。)

近接度は次のように決定されます:

1. `Group` パラメータに設定された値で開始します (デフォルトは 55)。
2. 同じホストを他の API ノードと共有する API ノードの場合は、値を 1 減らします。`Group` のデフォルト値を想定すると、API ノードと同じホスト上のデータノードの実効値は 54 で、リモートデータノード 55 の実効値です。

3. `ndb_data_node_neighbour` を設定すると、有効な `Group` 値がさらに 50 減少し、このノードは最も近いノードとみなされます。これは、すべてのデータノードが API ノードをホストするホスト以外のホスト上にあり、それらのいずれかを API ノード専用にすることが望ましい場合にのみ必要です。通常は、前述のデフォルトの調整で十分です。

これによりクラスタ接続の状態が変更されるため、安定化されるまで各スレッドからの新しいトランザクションの選択アルゴリズムが中断される可能性があるため、`ndb_data_node_neighbour` で頻繁に変更することはお薦めしません。

- `ndb_read_backup`

コマンド行形式	<code>--ndb-read-backup[={OFF ON}]</code>
システム変数	<code>ndb_read_backup</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値 (≥ 8.0.19-ndb-8.0.19)	ON
デフォルト値 (≤ 8.0.18-ndb-8.0.18)	OFF

その後作成される NDB テーブルのフラグメントレプリカからの読取りを有効にします。これにより、テーブルの読取りパフォーマンスが比較的低い書き込みコストで大幅に向上します。

個々のテーブルのフラグメントレプリカからの読取りを有効または無効にするには、`CREATE TABLE` または `ALTER TABLE` ステートメントで、それに応じてテーブルの `NDB_TABLE` オプション `READ_BACKUP` を設定します。詳細は、[セクション 13.1.20.11 「NDB_TABLE オプションの設定」](#) を参照してください。

- `ndb_recv_thread_activation_threshold`

コマンド行形式	<code>--ndb-recv-thread-activation-threshold=#</code>
システム変数	<code>ndb_recv_thread_activation_threshold</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	8
最小値	0 (MIN_ACTIVATION_THRESHOLD)
最大値	16 (MAX_ACTIVATION_THRESHOLD)

現在アクティブなスレッドがこの数に達すると、受信スレッドがクラスタ接続のポーリングを引き継ぎます。

この変数のスコープはグローバルです。起動時に設定することもできます。

- `ndb_recv_thread_cpu_mask`

コマンド行形式	<code>--ndb-recv-thread-cpu-mask=mask</code>
システム変数	<code>ndb_recv_thread_cpu_mask</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	ビットマップ

デフォルト値	[empty]
--------	---------

受信者スレッドを特定の CPU にロックするための CPU マスク。これは 16 進数ビットマスクとして指定されます。たとえば、0x33 は、受信側スレッドごとに 1 つの CPU が使用されることを意味します。空の文字列がデフォルトです。ndb_recv_thread_cpu_mask をこの値に設定すると、以前に設定された受信者スレッドのロックがすべて削除されます。

この変数のスコープはグローバルです。起動時に設定することもできます。

- [ndb_report_thresh_binlog_epoch_slip](#)

コマンド行形式	--ndb-report-thresh-binlog-epoch-slip=#
システム変数	ndb_report_thresh_binlog_epoch_slip
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	10
最小値	0
最大値	256

これは、イベントバッファに完全にバッファされたが、binlog インジェクタスレッドによってまだ消費されていないエポックの数のしきい値を表します。このスリッピング(ラグ)の程度を超えると、理由として `BUFFERED_EPOCHS_OVER_THRESHOLD` が提供されたイベントバッファステータスメッセージがレポートされます(セクション23.5.2.3「クラスタログでのイベントバッファのレポート」を参照)。データノードからエポックが受信され、イベントバッファに完全にバッファリングされると、スリップが増加します。エポックが binlog インジェクタスレッドによって消費されると、スリップは減少します。空のエポックはバッファリングされてキューに入れられるため、NDB API の `Ndb::setEventBufferQueueEmptyEpoch()` メソッドを使用してこれが有効になっている場合にのみ、この計算に含まれます。

- [ndb_report_thresh_binlog_mem_usage](#)

コマンド行形式	--ndb-report-thresh-binlog-mem-usage=#
システム変数	ndb_report_thresh_binlog_mem_usage
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	10
最小値	0
最大値	10

これは、バイナリログのステータスを報告するまでに残存する空きメモリの割合に対するしきい値です。たとえば、10 (デフォルト) の値は、データノードからバイナリログデータを受信するために使用可能なメモリの量が 10% を下回ると、ステータスメッセージがクラスタログに送信されることを意味します。

- [ndb_row_checksum](#)

システム変数	ndb_row_checksum
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ

型	Integer
デフォルト値	1
最小値	0
最大値	1

従来、NDB では、パフォーマンスを犠牲にしてハードウェアの問題をチェックする行チェックサムを含むテーブルが作成されていました。 `ndb_row_checksum` を 0 に設定すると、行チェックサムは新規または変更されたテーブルには使用されず、すべてのタイプのクエリーのパフォーマンスに大きく影響します。この変数はデフォルトで 1 に設定され、下位互換性のある動作を提供します。

- [ndb_schema_dist_lock_wait_timeout](#)

コマンド行形式	<code>--ndb-schema-dist-lock-wait-timeout=value</code>
導入	8.0.18-ndb-8.0.18
システム変数	ndb_schema_dist_lock_wait_timeout
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	30
最小値	0
最大値	1200
単位	seconds

DDL ステートメントの変更を反映するようにローカルデータディクショナリを変更するために各 SQL ノードで取得されたメタデータロックのスキーマ分散中に待機する秒数。この時間が経過すると、特定の SQL ノードデータディクショナリが変更で更新されなかったことを示す警告が返されます。これにより、バイナリロギングスレッドがスキーマ操作の処理中に過剰な時間待機することを回避できます。

- [ndb_schema_dist_timeout](#)

コマンド行形式	<code>--ndb-schema-dist-timeout=value</code>
導入	8.0.16-ndb-8.0.16
システム変数	ndb_schema_dist_timeout
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	120
最小値	5
最大値	1200
単位	seconds

スキーマ分散中にタイムアウトを検出するまで待機する秒数。これは、他の SQL ノードで過剰なアクティビティが発生しているか、この時点で必要なリソースを取得できない可能性があることを示します。

- [ndb_schema_dist_upgrade_allowed](#)

コマンド行形式	<code>--ndb-schema-dist-upgrade-allowed=value</code>
導入	8.0.17-ndb-8.0.17

システム変数	ndb_schema_dist_upgrade_allowed
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	true

NDB への接続時にスキーマ分散テーブルのアップグレードを許可します。true (デフォルト) の場合、この変更はすべての SQL ノードが同じバージョンの NDB Cluster ソフトウェアにアップグレードされるまで延期されます。

注記

スキーマ分散のパフォーマンスは、アップグレードが実行されるまで多少低下する可能性があります。

- [ndb_show_foreign_key_mock_tables](#)

コマンド行形式	--ndb-show-foreign-key-mock-tables[={OFF ON}]
システム変数	ndb_show_foreign_key_mock_tables
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

NDB が `foreign_key_checks=0` のサポートに使用するモックテーブルを示します。これを有効にすると、テーブルを作成および削除するときに追加の警告が表示されます。このテーブルの実際の (内部の) 名前は、[SHOW CREATE TABLE](#) の出力に表示されます。

- [ndb_slave_conflict_role](#)

コマンド行形式	--ndb-slave-conflict-role=value
非推奨	8.0.23-ndb-8.0.23
システム変数	ndb_slave_conflict_role
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	列挙
デフォルト値	NONE
有効な値	NONE PRIMARY SECONDARY PASS

NDB 8.0.23 では非推奨であり、将来のリリースで削除される可能性があります。かわりに [ndb_conflict_role](#) を使用してください。

- [ndb_table_no_logging](#)

システム変数	ndb_table_no_logging
--------	--------------------------------------

スコープ	セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

この変数を ON または 1 に設定すると、NDB テーブルのチェックポイントがディスクに対して実行されなくなります。より具体的には、この設定は `ndb_table_no_logging` が有効になっているときに `ENGINE NDB` を使用して作成または変更されたテーブルに適用され、あとで `ndb_table_no_logging` が変更された場合でも、テーブルの存続期間にわたって適用され続けます。ここに示すように、テーブル A、B、C、および D を作成 (および変更) して、`ndb_table_no_logging` の設定も変更したとします。

```
SET @@ndb_table_no_logging = 1;

CREATE TABLE A ... ENGINE NDB;

CREATE TABLE B ... ENGINE MYISAM;
CREATE TABLE C ... ENGINE MYISAM;

ALTER TABLE B ENGINE NDB;

SET @@ndb_table_no_logging = 0;

CREATE TABLE D ... ENGINE NDB;
ALTER TABLE C ENGINE NDB;

SET @@ndb_table_no_logging = 1;
```

前の一連のイベントのあと、テーブル A と B のチェックポイントは実行されません (A は `ENGINE NDB` を使用して作成され、B は `NDB` を使用するように変更されましたが、どちらも `ndb_table_no_logging` の有効時に行われたため)。ただし、テーブル C と D は記録されます (C は `NDB` を使用するように変更され、D は `ENGINE NDB` を使用して作成されましたが、どちらも `ndb_table_no_logging` の無効時に行われたため)。`ndb_table_no_logging` を 1 または ON に再度設定しても、テーブル C または D のチェックポイントは実行されません。

注記

`ndb_table_no_logging` は、NDB テーブルスキーマファイルの作成では無効になります。これをサポートするには、代わりに `ndb_table_temporary` を使用してください。

- [ndb_table_temporary](#)

システム変数	ndb_table_temporary
スコープ	セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

この変数を ON または 1 に設定すると、NDB テーブルがディスクに書き込まれなくなります。これは、テーブルスキーマファイルが作成されず、テーブルが記録されないことを意味します。

注記

現在、この変数を設定しても効果はありません。これは既知の問題です。Bug #34036 を参照してください。

- [ndb_use_copying_alter_table](#)

システム変数	ndb_use_copying_alter_table
--------	---

スコープ	グローバル、セッション
動的	いいえ
SET_VAR ヒントの適用	いいえ

オンラインの **ALTER TABLE** 操作で問題が発生したときに、NDB によりテーブルのコピーが強制的に使用されま
す。デフォルト値は **OFF** です。

- [ndb_use_exact_count](#)

システム変数	ndb_use_exact_count
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

SELECT COUNT(*) クエリーでこのタイプのクエリーの高速化を計画するときに、NDB によりレコードのカウン
トが強制的に使用されます。デフォルト値は **OFF** で、クエリー全体が高速化されます。

- [ndb_use_transactions](#)

コマンド行形式	<code>--ndb-use-transactions[={OFF ON}]</code>
システム変数	ndb_use_transactions
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

NDB のトランザクションサポートを無効にするには、この変数の値を **OFF** に設定します (推奨されません)。デ
フォルトは **ON** です。

- [ndb_version](#)

システム変数	ndb_version
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	

NDB エンジンのバージョン (合成整数として)。

- [ndb_version_string](#)

システム変数	ndb_version_string
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	

NDB エンジンのバージョン (ndb-x.y.z 形式)。

- `server_id_bits`

コマンド行形式	<code>--server-id-bits=#</code>
システム変数	<code>server_id_bits</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	32
最小値	7
最大値	32

この変数は、実際にサーバーを識別する 32-bit `server_id` 内の最下位ビットの数を示します。サーバーが実際には 32 ビット未満で識別されることを示すと、NDB API Event API を使用して `OperationOptions` 構造の `AnyValue` 内でアプリケーションによって生成されたユーザーデータを格納するなど、残りのビットの一部をほかの目的に使用できます (NDB Cluster は `AnyValue` を使用してサーバー ID を格納します)。

サーバーは、レプリケーションループの検出などで `server_id` から有効なサーバー ID を抽出するときに、残りのビットを無視します。`server_id_bits` 変数は、サーバー ID に基づいてイベントを無視するかどうかを決定する際に、I/O および SQL スレッド内の `server_id` の無関係なビットをマスクアウトするために使用されます。

このデータは、独自の `server_id_bits` 変数を 32 (デフォルト) に設定して実行すると、`mysqlbinlog` によってバイナリログから読み取ることができます。

`server_id` の値が `server_id_bits` の累乗に対して 2 以上の場合、`mysqld` は起動を拒否します。

このシステム変数は NDB Cluster でのみサポートされます。標準の MySQL 8.0 Server ではサポートされません。

- `slave_allow_batching`

コマンド行形式	<code>--slave-allow-batching[={OFF ON}]</code>
システム変数	<code>slave_allow_batching</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

NDB Cluster レプリカでバッチ更新が有効になっているかどうか。

この変数の設定は、NDB ストレージエンジンでレプリケーションを使用する場合にのみ有効です。MySQL Server 8.0 では存在しますが、何も行いません。詳細は、[セクション23.6.6「NDB Cluster レプリケーションの開始 \(シングルレプリケーションチャンネル\)」](#)を参照してください。

- `transaction_allow_batching`

システム変数	<code>transaction_allow_batching</code>
スコープ	セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean

デフォルト値	OFF
--------	-----

この変数を 1 または ON に設定すると、同じトランザクション内のステートメントのバッチ化が有効になります。この変数を使用するには、先に `autocommit` を 0 または OFF に設定して無効にする必要があります。これを実行しない場合、`transaction_allow_batching` を設定しても無効になります。

この変数は、有効にすると「以前の」イメージから読み取りが行われる可能性があるため、書き込みのみを実行するトランザクションで使用するのが安全です。SELECT を発行する前に、(必要に応じて明示的な COMMIT を使用して) 保留中のトランザクションをすべて確実にコミットするようにしてください。

重要

特定のステートメントの効果が同じトランザクション内の前のステートメントの結果に依存する可能性がある場合は、`transaction_allow_batching` を使用しないでください。

この変数は現在 NDB Cluster でのみサポートされています。

次のリストのシステム変数は、すべて `ndbinfo` 情報データベースに関連するものです。

• `ndbinfo_database`

システム変数	<code>ndbinfo_database</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	<code>ndbinfo</code>

NDB 情報データベースに使用される名前を示します。デフォルトは `ndbinfo` です。これは、コンパイル時に値が決定される読み取り専用変数です。

• `ndbinfo_max_bytes`

コマンド行形式	<code>--ndbinfo-max-bytes=#</code>
システム変数	<code>ndbinfo_max_bytes</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	0

テストとデバッグでのみ使用されます。

• `ndbinfo_max_rows`

コマンド行形式	<code>--ndbinfo-max-rows=#</code>
システム変数	<code>ndbinfo_max_rows</code>
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	10

テストとデバッグでのみ使用されます。

- [ndbinfo_offline](#)

システム変数	ndbinfo_offline
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

[ndbinfo](#) データベースをオフラインモードにします。オフラインモードでは、テーブルやビューが実際には存在しない場合や、存在するが NDB での定義が異なる場合でも、テーブルやビューを開くことができます。このようなテーブル (またはビュー) からは、行は返されません。

- [ndbinfo_show_hidden](#)

コマンド行形式	<code>--ndbinfo-show-hidden[={OFF ON}]</code>
システム変数	ndbinfo_show_hidden
スコープ	グローバル、セッション
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF
有効な値	ON OFF

[ndbinfo](#) データベースのベースとなる内部テーブルが `mysql` クライアントに表示されるかどうか。デフォルトは OFF です。

- [ndbinfo_table_prefix](#)

システム変数	ndbinfo_table_prefix
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列
デフォルト値	<code>ndb\$</code>

[ndbinfo](#) データベースのベーステーブル ([ndbinfo_show_hidden](#) を設定して公開されないかぎり、通常は非表示です) の名前に使用されるプリフィクス。これは読取り専用変数で、デフォルト値は `ndb$` です。接頭辞自体はコンパイル時に決定されます。

- [ndbinfo_version](#)

システム変数	ndbinfo_version
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	文字列

デフォルト値	
--------	--

使用中の `ndbinfo` エンジンのバージョンを示します (読み取り専用)。

NDB Cluster ステータス変数

このセクションでは、NDB Cluster および NDB ストレージエンジンに関連する MySQL サーバーのステータス変数について詳しく説明します。NDB Cluster に固有でないステータス変数、およびステータス変数の使用に関する一般的な情報については、[セクション5.1.10「サーバーステータス変数」](#)を参照してください。

- [Handler_discover](#)

MySQL サーバーは、`NDBCLUSTER` ストレージエンジンに対して特定の名前を持つテーブルの情報を問い合わせることができます。これは検出と呼ばれます。`Handler_discover` は、このメカニズムを使用してテーブルが検出された回数を示します。

- [Ndb_api_adaptive_send_deferred_count](#)

実際に送信されなかった適応送信コールの数。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_adaptive_send_deferred_count_session](#)

実際に送信されなかった適応送信コールの数。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_adaptive_send_deferred_count_replica](#)

このレプリカによって実際に送信されなかった適応送信コールの数。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_adaptive_send_deferred_count_slave](#)

注記

NDB 8.0.23 では非推奨です。代わりに `Ndb_api_adaptive_send_deferred_count_replica` を使用してください。

このレプリカによって実際に送信されなかった適応送信コールの数。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_adaptive_send_forced_count](#)

この MySQL Server (SQL ノード) によって送信された強制送信を使用した適応送信コールの数。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_adaptive_send_forced_count_session](#)

このクライアントセッションで送信された強制送信を使用した適応送信コールの数。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_adaptive_send_forced_count_replica](#)

このレプリカによって送信された強制送信を使用する適応送信コールの数。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_adaptive_send_forced_count_slave](#)

注記

NDB 8.0.23 では非推奨です。代わりに `Ndb_api_adaptive_send_forced_count_replica` を使用してください。

このレプリカによって送信された強制送信を使用する適応送信コールの数。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- `Ndb_api_adaptive_send_unforced_count`

この MySQL サーバー (SQL ノード) によって強制送信されていない適応送信コールの数。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- `Ndb_api_adaptive_send_unforced_count_session`

このクライアントセッションで強制送信が送信されていない適応送信コールの数。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- `Ndb_api_adaptive_send_unforced_count_replica`

このレプリカによって強制送信されていない適応送信コールの数。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- `Ndb_api_adaptive_send_unforced_count_slave`

注記

NDB 8.0.23 では非推奨です。代わりに `Ndb_api_adaptive_send_unforced_count_replica` を使用してください。

このレプリカによって強制送信されていない適応送信コールの数。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- `Ndb_api_bytes_sent_count_session`

このクライアントセッションでデータノードに送信されたデータ量 (バイト単位)。

この変数は、`SHOW GLOBAL STATUS` または `SHOW SESSION STATUS` を使用して読み取ることができますが、現在のセッションにのみ関連しており、この `mysqld` のほかのクライアントには影響しません。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- `Ndb_api_bytes_sent_count_replica`

このレプリカによってデータノードに送信されたデータの量 (バイト単位)。

この変数は、`SHOW GLOBAL STATUS` または `SHOW SESSION STATUS` を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_bytes_sent_count_slave](#)

注記

NDB 8.0.23 では非推奨です。代わりに [Ndb_api_bytes_sent_count_replica](#) を使用してください。

このレプリカによってデータノードに送信されたデータの量 (バイト単位)。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウントと変数」](#)を参照してください。

- [Ndb_api_bytes_sent_count](#)

この MySQL サーバー (SQL ノード) によってデータノードに送信されたデータ量 (バイト単位)。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。

詳細は、[セクション23.5.13「NDB API 統計のカウントと変数」](#)を参照してください。

- [Ndb_api_bytes_received_count_session](#)

このクライアントセッションでデータノードから受信されたデータ量 (バイト単位)。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、現在のセッションにのみ関連しており、この `mysqld` のほかのクライアントには影響しません。

詳細は、[セクション23.5.13「NDB API 統計のカウントと変数」](#)を参照してください。

- [Ndb_api_bytes_received_count_replica](#)

このレプリカによってデータノードから受信されたデータの量 (バイト単位)。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウントと変数」](#)を参照してください。

- [Ndb_api_bytes_received_count_slave](#)

注記

NDB 8.0.23 では非推奨です。代わりに [Ndb_api_bytes_received_count_replica](#) を使用してください。

このレプリカによってデータノードから受信されたデータの量 (バイト単位)。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウントと変数」](#)を参照してください。

- [Ndb_api_bytes_received_count](#)

この MySQL サーバー (SQL ノード) によってデータノードから受信されたデータ量 (バイト単位)。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_event_data_count_injector](#)

NDB binlog インジェクタスレッドによって受信された行変更イベントの数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_event_data_count](#)

この MySQL サーバー (SQL ノード) によって受信された行変更イベントの数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_event_nondata_count_injector](#)

NDB バイナリログインジェクタスレッドによって受信された、行変更イベント以外のイベントの数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_event_nondata_count](#)

この MySQL サーバー (SQL ノード) によって受信された、行変更イベント以外のイベントの数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_event_bytes_count_injector](#)

NDB binlog インジェクタスレッドによって受信されたイベントのバイト数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_event_bytes_count](#)

この MySQL サーバー (SQL ノード) によって受信されたイベントのバイト数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_pk_op_count_session](#)

このクライアントセッションで、主キーに基づいた、または主キーを使用した操作の数。これには、BLOB テーブルに対する操作、暗黙的なロック解除操作、自動インクリメント操作、およびユーザーの管理下にある主キー操作が含まれます。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、現在のセッションにのみ関連しており、この `mysqld` のほかのクライアントには影響しません。

詳細は、[セクション23.5.13「NDB API 統計のカウントと変数」](#)を参照してください。

- [Ndb_api_pk_op_count_replica](#)

主キーに基づいた、または主キーを使用した、このレプリカによる操作の数。これには、BLOB テーブルに対する操作、暗黙的なロック解除操作、自動インクリメント操作、およびユーザーの管理下にある主キー操作が含まれます。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウントと変数」](#)を参照してください。

- [Ndb_api_pk_op_count_slave](#)

注記

NDB 8.0.23 では非推奨です。代わりに [Ndb_api_pk_op_count_replica](#) を使用してください。

主キーに基づいた、または主キーを使用した、このレプリカによる操作の数。これには、BLOB テーブルに対する操作、暗黙的なロック解除操作、自動インクリメント操作、およびユーザーの管理下にある主キー操作が含まれます。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウントと変数」](#)を参照してください。

- [Ndb_api_pk_op_count](#)

この MySQL サーバー (SQL ノード) による、主キーに基づいた、または主キーを使用した操作の数。これには、BLOB テーブルに対する操作、暗黙的なロック解除操作、自動インクリメント操作、およびユーザーの管理下にある主キー操作が含まれます。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。

詳細は、[セクション23.5.13「NDB API 統計のカウントと変数」](#)を参照してください。

- [Ndb_api_pruned_scan_count_session](#)

このクライアントセッションで単一のパーティションにプルーニングされたスキャンの回数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、現在のセッションにのみ関連しており、この `mysqld` のほかのクライアントには影響しません。

詳細は、[セクション23.5.13「NDB API 統計のカウントと変数」](#)を参照してください。

- [Ndb_api_pruned_scan_count_replica](#)

単一のパーティションにプルーニングされた、このレプリカによるスキャンの数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#) を参照してください。

- [Ndb_api_pruned_scan_count_slave](#)

注記

NDB 8.0.23 では非推奨です。代わりに [Ndb_api_pruned_scan_count_replica](#) を使用してください。

単一のパーティションにプルーニングされた、このレプリカによるスキャンの数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#) を参照してください。

- [Ndb_api_pruned_scan_count](#)

この MySQL サーバー (SQL ノード) によって単一のパーティションにプルーニングされたスキャンの回数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#) を参照してください。

- [Ndb_api_range_scan_count_session](#)

このクライアントセッションで開始された範囲スキャンの回数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、現在のセッションにのみ関連しており、この `mysqlid` のほかのクライアントには影響しません。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#) を参照してください。

- [Ndb_api_range_scan_count_replica](#)

このレプリカによって開始されたレンジスキャンの数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#) を参照してください。

- [Ndb_api_range_scan_count_slave](#)

注記

NDB 8.0.23 では非推奨です。代わりに [Ndb_api_range_scan_count_replica](#) を使用してください。

このレプリカによって開始されたレンジスキャンの数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_range_scan_count](#)

この MySQL サーバー (SQL ノード) によって開始された範囲スキャンの回数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_read_row_count_session](#)

このクライアントセッションで読み取られた行の合計数。これには、このクライアントセッションで作成された主キー、一意キー、またはスキャン操作によって読み取られたすべての行が含まれます。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、現在のセッションにのみ関連しており、この `mysqld` のほかのクライアントには影響しません。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_read_row_count_replica](#)

このレプリカによって読み取られた行の合計数。これには、このレプリカによって作成された主キー、一意キーまたはスキャン操作によって読み取られたすべての行が含まれます。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_read_row_count_slave](#)

注記

NDB 8.0.23 では非推奨です。代わりに [Ndb_api_read_row_count_replica](#) を使用してください。

このレプリカによって読み取られた行の合計数。これには、このレプリカによって作成された主キー、一意キーまたはスキャン操作によって読み取られたすべての行が含まれます。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_read_row_count](#)

この MySQL サーバー (SQL ノード) によって読み取られた行の合計数。これには、この MySQL Server (SQL ノード) によって作成された主キー、一意キー、またはスキャン操作によって読み取られたすべての行が含まれます。

`SELECT COUNT(*)` クエリーによって読み取られる行に関しては、この値が完全に正確でない場合があることに注意してください。この場合、MySQL サーバーは実際に `[table fragment ID]:[number of rows in fragment]` 形式で疑似行を読み取り、テーブル内のすべてのフラグメントのフラグメント当たりの行数を合計して、すべての行の推定数を導出します。`Ndb_api_read_row_count` では、テーブルの実際の行数ではなく、この見積りが使用されます。

この変数は、`SHOW GLOBAL STATUS` または `SHOW SESSION STATUS` を使用して読み取ることができますが、スコープは実質的にグローバルです。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_scan_batch_count_session](#)

このクライアントセッションで受信された行のバッチ数。1 バッチは、単一のフラグメントから得られた 1 セットのスキャン結果として定義されます。

この変数は、`SHOW GLOBAL STATUS` または `SHOW SESSION STATUS` を使用して読み取ることができますが、現在のセッションにのみ関連しており、この `mysqld` のほかのクライアントには影響しません。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_scan_batch_count_replica](#)

このレプリカによって受信された行のバッチ数。1 バッチは、単一フラグメントからのスキャン結果の 1 セットとして定義されます。

この変数は、`SHOW GLOBAL STATUS` または `SHOW SESSION STATUS` を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_scan_batch_count_slave](#)

注記

NDB 8.0.23 では非推奨です。代わりに `Ndb_api_scan_batch_count_replica` を使用してください。

このレプリカによって受信された行のバッチ数。1 バッチは、単一フラグメントからのスキャン結果の 1 セットとして定義されます。

この変数は、`SHOW GLOBAL STATUS` または `SHOW SESSION STATUS` を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_scan_batch_count](#)

この MySQL Server (SQL ノード) によって受信された行のバッチ数。1 バッチは、単一のフラグメントから得られた 1 セットのスキャン結果として定義されます。

この変数は、`SHOW GLOBAL STATUS` または `SHOW SESSION STATUS` を使用して読み取ることができますが、スコープは実質的にグローバルです。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_table_scan_count_session](#)

このクライアントセッションで開始された、内部テーブルのスキャンを含むテーブルスキャンの回数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、現在のセッションにのみ関連しており、この `mysqld` のほかのクライアントには影響しません。

詳細は、[セクション23.5.13「NDB API 統計のカウントと変数」](#)を参照してください。

- [Ndb_api_table_scan_count_replica](#)

このレプリカによって開始されたテーブルスキャンの数 (内部テーブルのスキャンを含む)。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウントと変数」](#)を参照してください。

- [Ndb_api_table_scan_count_slave](#)

注記

NDB 8.0.23 では非推奨です。代わりに [Ndb_api_table_scan_count_replica](#) を使用してください。

このレプリカによって開始されたテーブルスキャンの数 (内部テーブルのスキャンを含む)。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウントと変数」](#)を参照してください。

- [Ndb_api_table_scan_count](#)

この MySQL サーバー (SQL ノード) によって開始された、内部テーブルのスキャンを含むテーブルスキャンの回数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。

詳細は、[セクション23.5.13「NDB API 統計のカウントと変数」](#)を参照してください。

- [Ndb_api_trans_abort_count_session](#)

このクライアントセッションで中止されたトランザクションの数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、現在のセッションにのみ関連しており、この `mysqld` のほかのクライアントには影響しません。

詳細は、[セクション23.5.13「NDB API 統計のカウントと変数」](#)を参照してください。

- [Ndb_api_trans_abort_count_replica](#)

このレプリカによって中断されたトランザクションの数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウントと変数」](#)を参照してください。

- [Ndb_api_trans_abort_count_slave](#)

注記

NDB 8.0.23 では非推奨です。代わりに [Ndb_api_trans_abort_count_replica](#) を使用してください。

このレプリカによって中断されたトランザクションの数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_trans_abort_count](#)

この MySQL サーバー (SQL ノード) により中止されたトランザクションの数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_trans_close_count_session](#)

このクライアントセッションで閉じられたトランザクションの数。一部のトランザクションがロールバックされている場合があるため、この値は [Ndb_api_trans_commit_count_session](#) と [Ndb_api_trans_abort_count_session](#) の合計より大きくなる場合があります。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、現在のセッションにのみ関連しており、この `mysqld` のほかのクライアントには影響しません。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_trans_close_count_replica](#)

このレプリカによってクローズされたトランザクションの数。一部のトランザクションはロールバックされている可能性があるため、この値は [Ndb_api_trans_commit_count_replica](#) と [Ndb_api_trans_abort_count_replica](#) の合計より大きい場合があります。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_trans_close_count_slave](#)

注記

NDB 8.0.23 では非推奨です。代わりに [Ndb_api_trans_close_count_replica](#) を使用してください。

このレプリカによってクローズされたトランザクションの数。一部のトランザクションはロールバックされている可能性があるため、この値は [Ndb_api_trans_commit_count_replica](#) と [Ndb_api_trans_abort_count_replica](#) の合計より大きい場合があります。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_trans_close_count](#)

この MySQL サーバー (SQL ノード) により閉じられたトランザクションの数。一部のトランザクションがロールバックされている場合があるため、この値は [Ndb_api_trans_commit_count](#) と [Ndb_api_trans_abort_count](#) の合計より大きくなる場合があります。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。

詳細は、[セクション23.5.13「NDB API 統計のカウントと変数」](#)を参照してください。

- [Ndb_api_trans_commit_count_session](#)

このクライアントセッションでコミットされたトランザクションの数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、現在のセッションにのみ関連しており、この `mysqld` のほかのクライアントには影響しません。

詳細は、[セクション23.5.13「NDB API 統計のカウントと変数」](#)を参照してください。

- [Ndb_api_trans_commit_count_replica](#)

このレプリカによってコミットされたトランザクションの数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウントと変数」](#)を参照してください。

- [Ndb_api_trans_commit_count_slave](#)

注記

NDB 8.0.23 では非推奨です。代わりに [Ndb_api_trans_commit_count_replica](#) を使用してください。

このレプリカによってコミットされたトランザクションの数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウントと変数」](#)を参照してください。

- [Ndb_api_trans_commit_count](#)

この MySQL サーバー (SQL ノード) によりコミットされたトランザクションの数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。

詳細は、[セクション23.5.13「NDB API 統計のカウントと変数」](#)を参照してください。

- [Ndb_api_trans_local_read_row_count_session](#)

このクライアントセッションで読み取られた行の合計数。これには、このクライアントセッションで作成された主キー、一意キー、またはスキャン操作によって読み取られたすべての行が含まれます。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、現在のセッションにのみ関連しており、この `mysqld` のほかのクライアントには影響しません。

詳細は、[セクション23.5.13「NDB API 統計のカウントと変数」](#)を参照してください。

- [Ndb_api_trans_local_read_row_count_replica](#)

このレプリカによって読み取られた行の合計数。これには、このレプリカによって作成された主キー、一意キーまたはスキャン操作によって読み取られたすべての行が含まれます。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_trans_local_read_row_count_slave](#)

注記

NDB 8.0.23 では非推奨です。代わりに [Ndb_api_trans_local_read_row_count_replica](#) を使用してください。

このレプリカによって読み取られた行の合計数。これには、このレプリカによって作成された主キー、一意キーまたはスキャン操作によって読み取られたすべての行が含まれます。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_trans_local_read_row_count](#)

この MySQL サーバー (SQL ノード) によって読み取られた行の合計数。これには、この MySQL Server (SQL ノード) によって作成された主キー、一意キー、またはスキャン操作によって読み取られたすべての行が含まれます。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_trans_start_count_session](#)

このクライアントセッションで開始されたトランザクションの数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、現在のセッションにのみ関連しており、この `mysqld` のほかのクライアントには影響しません。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_trans_start_count_replica](#)

このレプリカによって開始されたトランザクションの数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_trans_start_count_slave](#)

注記

NDB 8.0.23 では非推奨です。代わりに [Ndb_api_trans_start_count_replica](#) を使用してください。

このレプリカによって開始されたトランザクションの数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_trans_start_count](#)

この MySQL サーバー (SQL ノード) により開始されたトランザクションの数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_uk_op_count_session](#)

このクライアントセッションで、一意キーに基づいた、または一意キーを使用した操作の数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、現在のセッションにのみ関連しており、この `mysqld` のほかのクライアントには影響しません。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_uk_op_count_replica](#)

一意キーに基づいた、または一意キーを使用した、このレプリカによる操作の数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_uk_op_count_slave](#)

注記

NDB 8.0.23 では非推奨です。代わりに [Ndb_api_uk_op_count_replica](#) を使用してください。

一意キーに基づいた、または一意キーを使用した、このレプリカによる操作の数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_uk_op_count](#)

この MySQL サーバー (SQL ノード) による、一意キーに基づいた、または一意キーを使用した操作の数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_wait_exec_complete_count_session](#)

このクライアントセッションで、操作の実行が完了するのを待機する間にスレッドがブロックされた回数。これには、すべての `execute()` コールと、クライアントに表示されない BLOB および自動増分操作に対する暗黙的な実行が含まれます。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、現在のセッションにのみ関連しており、この `mysqld` のほかのクライアントには影響しません。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_wait_exec_complete_count_replica](#)

操作の実行が完了するのを待機している間に、このレプリカによってスレッドがブロックされた回数。これには、すべての `execute()` コールと、クライアントに表示されない BLOB および自動増分操作に対する暗黙的な実行が含まれます。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_wait_exec_complete_count_slave](#)

注記

NDB 8.0.23 では非推奨です。代わりに [Ndb_api_wait_exec_complete_count_replica](#) を使用してください。

操作の実行が完了するのを待機している間に、このレプリカによってスレッドがブロックされた回数。これには、すべての `execute()` コールと、クライアントに表示されない BLOB および自動増分操作に対する暗黙的な実行が含まれます。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_wait_exec_complete_count](#)

操作の実行が完了するまで待機する間に、この MySQL Server (SQL ノード) によってスレッドがブロックされた回数。これには、すべての `execute()` コールと、クライアントに表示されない BLOB および自動増分操作に対する暗黙的な実行が含まれます。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_wait_meta_request_count_session](#)

このクライアントセッションで、メタデータベースの信号 (DDL 要求、新しいエポック、トランザクションレコードの占有など) を待機する間にスレッドがブロックされた回数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、現在のセッションにのみ関連しており、この `mysqld` のほかのクライアントには影響しません。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_wait_meta_request_count_replica](#)

DDL リクエスト、新しいエポック、トランザクションレコードの清算など、メタデータベースのシグナルを待機しているこのレプリカによってスレッドがブロックされた回数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_wait_meta_request_count_slave](#)

注記

NDB 8.0.23 では非推奨です。代わりに [Ndb_api_wait_meta_request_count_replica](#) を使用してください。

DDL リクエスト、新しいエポック、トランザクションレコードの清算など、メタデータベースのシグナルを待機しているこのレプリカによってスレッドがブロックされた回数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_wait_meta_request_count](#)

メタデータベースの信号 (DDL 要求、新しいエポック、トランザクションレコードの占有など) を待機する間に、この MySQL Server (SQL ノード) によってスレッドがブロックされた回数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_wait_nanos_count_session](#)

このクライアントセッションで、データノードから送信される任意のタイプの信号の待機に費やされた合計時間 (ナノ秒)。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、現在のセッションにのみ関連しており、この `mysqld` のほかのクライアントには影響しません。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_wait_nanos_count_replica](#)

このレプリカがデータノードからの任意のタイプのシグナルを待機するために費やした合計時間 (ナノ秒)。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_wait_nanos_count_slave](#)

注記

NDB 8.0.23 では非推奨です。代わりに [Ndb_api_wait_nanos_count_replica](#) を使用してください。

このレプリカがデータノードからの任意のタイプのシグナルを待機するために費やした合計時間 (ナノ秒)。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_wait_nanos_count](#)

この MySQL Server (SQL ノード) によって、データノードから送信される任意のタイプの信号の待機に費やされた合計時間 (ナノ秒)。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_wait_scan_result_count_session](#)

このクライアントセッションで、スキャンベースの信号を待機する間 (追加のスキャン結果を待機するときや、スキャンが閉じるまで待機するときなど) にスレッドがブロックされた回数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、現在のセッションにのみ関連しており、この `mysqld` のほかのクライアントには影響しません。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_wait_scan_result_count_replica](#)

スキャンベースのシグナルを待機している間、またはスキャンの終了を待機しているときなどに、このレプリカによってスレッドがブロックされた回数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_wait_scan_result_count_slave](#)

注記

NDB 8.0.23 では非推奨です。代わりに [Ndb_api_wait_scan_result_count_replica](#) を使用してください。

スキャンベースのシグナルを待機している間、またはスキャンの終了を待機しているときなどに、このレプリカによってスレッドがブロックされた回数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。この MySQL サーバーがレプリカとして機能しない場合、または NDB テーブルを使用しない場合、この値は常に 0 です。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_api_wait_scan_result_count](#)

スキャンベースの信号を待機する間 (追加のスキャン結果を待機するときや、スキャンが閉じるまで待機するときなど) に、この MySQL Server (SQL ノード) によってスレッドがブロックされた回数。

この変数は、[SHOW GLOBAL STATUS](#) または [SHOW SESSION STATUS](#) を使用して読み取ることができますが、スコープは実質的にグローバルです。

詳細は、[セクション23.5.13「NDB API 統計のカウンタと変数」](#)を参照してください。

- [Ndb_cluster_node_id](#)

サーバーが NDB Cluster ノードとして機能している場合、この変数の値はクラスタ内のそのノード ID です。

サーバーが NDB Cluster の一部でない場合、この変数の値は 0 です。

- [Ndb_config_from_host](#)

サーバーが NDB Cluster の一部である場合、この変数の値は、その構成データを取得するクラスタ管理サーバーのホスト名または IP アドレスです。

サーバーが NDB Cluster の一部でない場合、この変数の値は空の文字列になります。

- [Ndb_config_from_port](#)

サーバーが NDB Cluster の一部である場合、この変数の値は、サーバーが構成データを取得するクラスタ管理サーバーに接続されているポートの番号です。

サーバーが NDB Cluster の一部でない場合、この変数の値は 0 です。

- [Ndb_config_generation](#)

クラスタの現在の構成の世代番号を表示します。これは、この SQL ノードが最後にクラスタに接続してからクラスタの構成が変更されたかどうかを判断するインジケータとして使用できます。

- [Ndb_conflict_fn_max_del_win](#)

この `mysqld` が最後に起動されてから、[NDB\\$MAX_DELETE_WIN\(\)](#) を使用した NDB Cluster レプリケーション競合解決のために現在の SQL ノードで行が拒否された回数を示します。

詳細は、[セクション23.6.11「NDB Cluster レプリケーションの競合解決」](#)を参照してください。

- [Ndb_conflict_fn_max](#)

NDB Cluster レプリケーション競合解決で使用されるこの変数は、この `mysqld` が最後に起動されてから、「最大のタイムスタンプ優先」競合解決のために現在の SQL ノードに行が適用されなかった回数を示します。

詳細は、[セクション23.6.11「NDB Cluster レプリケーションの競合解決」](#)を参照してください。

- [Ndb_conflict_fn_old](#)

NDB Cluster レプリケーション競合解決で使用されるこの変数は、前回の再起動以降に、特定の `mysqld` で「同じタイムスタンプを優先」競合解決の結果として行が適用されなかった回数を示します。

詳細は、[セクション23.6.11「NDB Cluster レプリケーションの競合解決」](#)を参照してください。

- [Ndb_conflict_fn_epoch](#)

NDB Cluster レプリケーション競合解決で使用されるこの変数は、前回の再起動以降に、特定の `mysqld` で [NDB\\$EPOCH\(\)](#) 競合解決を使用して競合が検出された行数を示します。

詳細は、[セクション23.6.11「NDB Cluster レプリケーションの競合解決」](#)を参照してください。

- [Ndb_conflict_fn_epoch2](#)

`NDB$EPOCH2()` を使用しているときに、前回の再起動以降にプライマリとして指定されたソースで NDB Cluster レプリケーション競合解決で競合が検出された行数を表示します。

詳細は、[NDB\\$EPOCH2\(\)](#) を参照してください。

- [Ndb_conflict_fn_epoch_trans](#)

NDB Cluster レプリケーション競合解決で使用されるこの変数は、前回の再起動以降に、特定の `mysqld` で `NDB$EPOCH_TRANS()` 競合解決を使用して競合が検出された行数を示します。

詳細は、[セクション23.6.11「NDB Cluster レプリケーションの競合解決」](#) を参照してください。

- [Ndb_conflict_fn_epoch2_trans](#)

NDB Cluster レプリケーション競合解決で使用されるこの変数は、前回の再起動以降に、特定の `mysqld` で `NDB$EPOCH_TRANS2()` 競合解決を使用して競合が検出された行数を示します。

詳細は、[NDB\\$EPOCH2_TRANS\(\)](#) を参照してください。

- [Ndb_conflict_last_conflict_epoch](#)

このレプリカで競合が検出された最新のエポック。この値は `Ndb_replica_max_replicated_epoch` と比較できません。`Ndb_replica_max_replicated_epoch` が `Ndb_conflict_last_conflict_epoch` より大きい場合、競合はまだ検出されていません。

詳細は、[セクション23.6.11「NDB Cluster レプリケーションの競合解決」](#) を参照してください。

- [Ndb_conflict_reflected_op_discard_count](#)

NDB Cluster レプリケーション競合解決を使用している場合、これは、実行中にエラーが発生したためにセカンダリに適用されなかった反映された操作の数です。

詳細は、[セクション23.6.11「NDB Cluster レプリケーションの競合解決」](#) を参照してください。

- [Ndb_conflict_reflected_op_prepare_count](#)

NDB Cluster レプリケーションで競合解決を使用する場合、このステータス変数には、定義されている (つまり、セカンダリで実行するために準備されている) 反映された操作の数が含まれます。

[セクション23.6.11「NDB Cluster レプリケーションの競合解決」](#) を参照してください。

- [Ndb_conflict_refresh_op_count](#)

NDB Cluster レプリケーションで競合解決を使用する場合、これにより、セカンダリで実行するために準備されているリフレッシュ操作の数わかります。

詳細は、[セクション23.6.11「NDB Cluster レプリケーションの競合解決」](#) を参照してください。

- [Ndb_conflict_last_stable_epoch](#)

トランザクション競合関数によって競合状態であることが検出された行数

詳細は、[セクション23.6.11「NDB Cluster レプリケーションの競合解決」](#) を参照してください。

- [Ndb_conflict_trans_row_conflict_count](#)

NDB Cluster レプリケーション競合解決で使用されるこのステータス変数は、前回の再起動以降に、特定の `mysqld` 上のトランザクション競合関数によって直接競合していることが検出された行の数を示します。

現在、NDB Cluster でサポートされているトランザクション競合検出関数は `NDB$EPOCH_TRANS()` のみであるため、このステータス変数は事実上 `Ndb_conflict_fn_epoch_trans` と同じです。

詳細は、[セクション23.6.11「NDB Cluster レプリケーションの競合解決」](#) を参照してください。

- [Ndb_conflict_trans_row_reject_count](#)

NDB Cluster レプリケーション競合解決で使用されるこのステータス変数は、トランザクション競合検出回数によって競合していると判断されたために再割り当てされた行の合計数を示します。これには、[Ndb_conflict_trans_row_conflict_count](#) だけでなく、競合するトランザクションに含まれる行や依存する行も含まれます。

詳細は、[セクション23.6.11「NDB Cluster レプリケーションの競合解決」](#)を参照してください。

- [Ndb_conflict_trans_reject_count](#)

NDB Cluster レプリケーション競合解決で使用されるこのステータス変数は、トランザクション競合検出回数によって競合していることが検出されたトランザクションの数を示します。

詳細は、[セクション23.6.11「NDB Cluster レプリケーションの競合解決」](#)を参照してください。

- [Ndb_conflict_trans_detect_iter_count](#)

NDB Cluster レプリケーション競合解決で使用され、エポックトランザクションのコミットに必要な内部反復の数を示します。[Ndb_conflict_trans_conflict_commit_count](#) より (わずかに) 大きいか、等しい値にしてください。

詳細は、[セクション23.6.11「NDB Cluster レプリケーションの競合解決」](#)を参照してください。

- [Ndb_conflict_trans_conflict_commit_count](#)

NDB Cluster レプリケーションの競合解決で使用され、トランザクションの競合処理が必要になったあとにコミットされたエポックトランザクションの数を示します。

詳細は、[セクション23.6.11「NDB Cluster レプリケーションの競合解決」](#)を参照してください。

- [Ndb_epoch_delete_delete_count](#)

delete-delete 競合検出を使用する場合、これは、削除操作が適用されるが、示された行が存在しない、検出された delete-delete 競合の数です。

- [Ndb_execute_count](#)

操作によって行われた NDB カーネルへのラウンドトリップの回数を示します。

- [Ndb_last_commit_epoch_server](#)

NDB によって最後にコミットされたエポック。

- [Ndb_last_commit_epoch_session](#)

この NDB クライアントによって最後にコミットされたエポック。

- [Ndb_metadata_detected_count](#)

このサーバーが最後に起動されてから NDB メタデータ変更検出スレッドが MySQL データディクショナリに関する変更を検出した回数。

NDB 8.0.16 に追加されました。

- [Ndb_metadata_excluded_count](#)

NDB binlog スレッドが最後に再起動されてから、この SQL ノードで同期できなかったメタデータオブジェクトの数。

オブジェクトを除外すると、ユーザーが不一致を手動で修正するまで、自動同期化の対象とはみなされません。これを行うには、[SHOW CREATE TABLE table](#)、[SELECT * FROM table](#) などのステートメント、またはテーブル検出をトリガーするその他のステートメントでテーブルを使用します。

NDB 8.0.18 に追加されました。NDB 8.0.22 より前では、この変数は [Ndb_metadata_blacklist_size](#) という名前でした。

- [Ndb_metadata_synced_count](#)

この SQL ノードが最後に再起動されてから同期された NDB メタデータオブジェクトの数。

NDB 8.0.18 に追加されました。

- [Ndb_number_of_data_nodes](#)

サーバーが NDB Cluster の一部である場合、この変数の値はクラスタ内のデータノードの数です。

サーバーが NDB Cluster の一部でない場合、この変数の値は 0 です。

- [Ndb_pushed_queries_defined](#)

データノードでの分散処理のために NDB カーネルにプッシュダウンされた結合の合計数。

注記

プッシュダウン可能な [EXPLAIN](#) を使用してテストされた結合は、この数値に寄与しません。

- [Ndb_pushed_queries_dropped](#)

NDB カーネルにプッシュダウンされ、処理できなかった結合の数。

- [Ndb_pushed_queries_executed](#)

NDB に正常にプッシュダウンされ、実行された結合の数。

- [Ndb_pushed_reads](#)

プッシュダウンされた結合によって NDB カーネルから `mysqld` に返された行の数。

注記

NDB にプッシュダウンできる結合で [EXPLAIN](#) を実行しても、この数値には追加されません。

- [Ndb_pruned_scan_count](#)

この変数は、NDB Cluster が最後に起動されてから、[NDBCLUSTER](#) がパーティションプルーニングを使用できるようになった [NDBCLUSTER](#) によって実行されたスキャンの数を保持します。

この変数を [Ndb_scan_count](#) とともに使用すると、単一のテーブルパーティションにスキャンをプルーニングするサーバーの機能を最大化して、単一のデータノードのみを含むスキーマ設計で役立ちます。

- [Ndb_replica_max_replicated_epoch](#)

このレプリカで最後にコミットされたエポック。この値は [Ndb_conflict_last_conflict_epoch](#) と比較できません。 [Ndb_replica_max_replicated_epoch](#) がこの値より大きい場合、競合はまだ検出されていません。

詳細は、[セクション23.6.11「NDB Cluster レプリケーションの競合解決」](#)を参照してください。

- [Ndb_scan_count](#)

この変数は、NDB Cluster が最後に起動されてから [NDBCLUSTER](#) によって実行されたスキャンの合計数を保持します。

- [Ndb_slave_max_replicated_epoch](#)

注記

NDB 8.0.23 では非推奨です。代わりに [Ndb_slave_max_replicated_epoch](#) を使用してください。

このレプリカで最後にコミットされたエポック。この値は [Ndb_conflict_last_conflict_epoch](#) と比較できます。 [Ndb_slave_max_replicated_epoch](#) がこの値より大きい場合、競合はまだ検出されていません。

詳細は、[セクション23.6.11「NDB Cluster レプリケーションの競合解決」](#)を参照してください。

- [Ndb_system_name](#)

この MySQL Server が NDB クラスタに接続されている場合、この読み取り専用変数はクラスタシステム名を示します。それ以外の場合、値は空の文字列です。

- [Ndb_trans_hint_count_session](#)

現在のセッションで開始されたヒントを使用したトランザクションの数。 [Ndb_api_trans_start_count_session](#) と比較して、ヒントを使用できるすべての NDB トランザクションの比率を取得します。NDB 8.0.17 に追加されました。

23.3.3.10 NDB Cluster TCP/IP 接続

TCP/IP は NDB Cluster 内のノード間のすべての接続のデフォルトのトランスポートメカニズムです。通常、TCP/IP 接続を定義する必要はありません。NDB Cluster は、すべてのデータノード、管理ノード、および SQL または API ノードに対してこのような接続を自動的に設定します。

注記

このルールの例外については、[セクション23.3.3.11「直接接続を使用した NDB Cluster TCP/IP 接続」](#)を参照してください。

デフォルトの接続パラメータをオーバーライドするには、[config.ini](#) ファイルで 1 つ以上の `[tcp]` セクションを使用して接続を定義する必要があります。各 `[tcp]` セクションでは、2 つの NDB Cluster ノード間の TCP/IP 接続を明示的に定義し、少なくともパラメータ `Nodeld1` と `Nodeld2`、およびオーバーライドする接続パラメータを含める必要があります。

これらのパラメータのデフォルト値を `[tcp default]` セクションに設定して変更することもできます。

重要

[config.ini](#) ファイル内の `[tcp]` セクションは、最後に (ファイル内のほかのすべてのセクションのあとで) 指定するようにしてください。ただし、`[tcp default]` セクションについては、これは必須ではありません。この要件は、NDB Cluster 管理サーバーが [config.ini](#) ファイルを読み取る方法に関する既知の問題です。

[config.ini](#) ファイルの `[tcp]` および `[tcp default]` セクションに設定できる接続パラメータをここに示します。

- [AllowUnresolvedHostNames](#)

バージョン (またはそれ以降)	NDB 8.0.22
タイプまたは単位	boolean
デフォルト	false
範囲	true, false
追加	NDB 8.0.22
再起動タイプ	N (NDB 8.0.13)

デフォルトでは、接続の試行中に管理ノードがホスト名の解決に失敗すると、致命的エラーが発生します。この動作は、グローバル構成ファイル (通常は [config.ini](#) という名前) の `[tcp default]` セクションで

`AllowUnresolvedHostNames` を `true` に設定することでオーバーライドできます。この場合、ホスト名の解決の失敗は警告として扱われ、`ndb_mgmd` の起動は中断されずに続行されます。

- Checksum

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	boolean
デフォルト	false
範囲	true, false
再起動タイプ	N (NDB 8.0.13)

このパラメータはブールパラメータです (`Y` または `1` に設定すると有効になり、`N` または `0` に設定すると無効になります)。デフォルトでは無効になっています。有効にすると、送信バッファに配置される前にすべてのメッセージのチェックサムが計算されます。この機能によって、メッセージが送信バッファでの待機中に (またはトランスポートメカニズムによって) 破損していないことが確認されます。

- グループ

`ndb_optimized_node_selection` が有効になっている場合、接続先のノードを選択するためにノードの近接性が使用されることがあります。このパラメータは、「closer」として解釈される下限値に設定することで、近接性に影響を与えるために使用できます。詳細は、システム変数の説明を参照してください。

- HostName1

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	name or IP address
デフォルト	[...]
範囲	...
再起動タイプ	N (NDB 8.0.13)

`HostName1` および `HostName2` パラメータを使用すると、2つのノード間の特定の TCP 接続で使用する特定のネットワークインタフェースを指定できます。これらのパラメータに使用する値は、ホスト名または IP アドレスです。

- HostName2

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	name or IP address
デフォルト	[...]
範囲	...
再起動タイプ	N (NDB 8.0.13)

`HostName1` および `HostName2` パラメータを使用すると、2つのノード間の特定の TCP 接続で使用する特定のネットワークインタフェースを指定できます。これらのパラメータに使用する値は、ホスト名または IP アドレスです。

- NodeId1

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	numeric
デフォルト	[none]
範囲	1 - 255

再起動タイプ	N (NDB 8.0.13)
--------	----------------

2つのノード間の接続を識別するには、構成ファイルの `[tcp]` セクションに `Nodeld1` および `Nodeld2` の値としてノード ID を指定する必要があります。これらは、各ノードに対する一意の `Id` 値であり、[セクション 23.3.3.7「NDB Cluster での SQL およびその他の API ノードの定義」](#) で説明しているものと同じです。

- `Nodeld2`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	numeric
デフォルト	[none]
範囲	1 - 255
再起動タイプ	N (NDB 8.0.13)

2つのノード間の接続を識別するには、構成ファイルの `[tcp]` セクションに `Nodeld1` および `Nodeld2` の値としてノード ID を指定する必要があります。これらは、各ノードに対する一意の `Id` 値であり、[セクション 23.3.3.7「NDB Cluster での SQL およびその他の API ノードの定義」](#) で説明しているものと同じです。

- `OverloadLimit`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	bytes
デフォルト	0
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

送信バッファにこれより多くの未送信バイトがあるときは、接続が過負荷状態であるとみなされます。

このパラメータを使用すると、接続を過負荷状態であるとみなす前に送信バッファに存在する未送信データ量を決定できます。詳細は、[セクション 23.3.3.14「NDB Cluster 送信バッファパラメータの構成」](#) を参照してください。

- `PreSendChecksum`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	boolean
デフォルト	false
範囲	true, false
再起動タイプ	

このパラメータと `Checksum` の両方が有効になっている場合は、送信前チェックサムチェックを実行し、ノード間のすべての TCP シグナルでエラーをチェックします。`Checksum` も有効になっていない場合は、何の効果もありません。

- `ReceiveBufferMemory`

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	bytes
デフォルト	2M
範囲	16K - 4294967039 (0xFFFFFFFF)

再起動タイプ	N (NDB 8.0.13)
--------	----------------

TCP/IP ソケットからデータを受信するときに使用するバッファのサイズを指定します。

このパラメータのデフォルト値は 2M バイトです。指定可能な最小値は 16K バイトです。理論的な最大は 4G バイトです。

- [SendBufferMemory](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	unsigned
デフォルト	2M
範囲	256K - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

TCP トランスポータは、オペレーティングシステムに対する送信呼び出しを実行する前に、バッファを使用してすべてのメッセージを格納します。このバッファが 64K バイトに達すると、その内容が送信されます。これは、一連のメッセージが実行されたときにも送信されます。一時的な過負荷状態に対応するため、より大きな送信バッファを定義することもできます。

このパラメータが明示的に設定されている場合は、メモリーが各トランスポータ専用でなくなります。代わりに、使用された値によって、(使用可能なメモリーの合計、つまり [TotalSendBufferMemory](#) のうち) 単一のトランスポータが使用できるメモリー量に関する厳密な制限が示されます。NDB Cluster での動的トランスポータ送信バッファメモリー割り当ての構成の詳細は、[セクション23.3.3.14「NDB Cluster 送信バッファパラメータの構成」](#)を参照してください。

送信バッファのデフォルトサイズは 2M バイトです。これは、ほとんどの状況で推奨されるサイズです。最小サイズは 64K バイトです。理論的な最大は 4G バイトです。

- [SendSignalId](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	boolean
デフォルト	false (debug builds: true)
範囲	true, false
再起動タイプ	N (NDB 8.0.13)

配信されたメッセージデータグラムを再トレースできるようにするには、各メッセージを識別する必要があります。このパラメータを **Y** に設定すると、メッセージ ID がネットワーク経由で転送されます。この機能は、製品ビルドではデフォルトで無効になっており、**-debug** ビルドで有効になります。

- [TcpBind_INADDR_ANY](#)

このパラメータを **TRUE** または **1** に設定すると、**IP_ADDR_ANY** がバインドされ、任意の場所から接続できるようになります (自動生成接続の場合)。デフォルトは **FALSE (0)** です。

- [TcpSpinTime](#)

バージョン (またはそれ以降)	NDB 8.0.20
タイプまたは単位	µsec
デフォルト	0
範囲	0 - 2000
追加	NDB 8.0.20

再起動タイプ	N (NDB 8.0.13)
--------	----------------

TCP トランスポータのスピンを制御します。有効にしないで、ゼロ以外の値に設定します。これは、接続のデータノード側と管理側または SQL ノード側の両方で機能します。

- TCP_MAXSEG_SIZE

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	unsigned
デフォルト	0
範囲	0 - 2G
再起動タイプ	N (NDB 8.0.13)

TCP トランスポータの初期化時に設定されるメモリのサイズを決定します。ほとんどの一般的な使用ケースでは、デフォルトが推奨されます。

- TCP_RCV_BUF_SIZE

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	unsigned
デフォルト	0
範囲	0 - 2G
再起動タイプ	N (NDB 8.0.13)

TCP トランスポータの初期化時に設定される受信バッファのサイズを指定します。デフォルト値および最小値は 0 で、オペレーティングシステムまたはプラットフォームでこの値を設定できます。ほとんどの一般的な使用ケースでは、デフォルトが推奨されます。

- TCP_SND_BUF_SIZE

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	unsigned
デフォルト	0
範囲	0 - 2G
再起動タイプ	N (NDB 8.0.13)

TCP トランスポータの初期化時に設定される送信バッファのサイズを指定します。デフォルト値および最小値は 0 で、オペレーティングシステムまたはプラットフォームでこの値を設定できます。ほとんどの一般的な使用ケースでは、デフォルトが推奨されます。

再起動のタイプ。 このセクションのパラメータの説明で 사용되는再起動タイプに関する情報を次のテーブルに示します:

表 23.21 NDB Cluster の再起動タイプ

シンボル	再起動タイプ	説明
N	ノード	パラメータはローリング再起動を使用して更新できます (セクション 23.5.5 「NDB Cluster のローリング再起動の実行」 を参照)
S	システム	このパラメータの変更を有効にするには、すべてのクラスタノードを完全に停止してから再起動する必要があります

シンボル	再起動タイプ	説明
I	Initial	--initial オプションを使用してデータノードを再起動する必要があります

23.3.3.11 直接接続を使用した NDB Cluster TCP/IP 接続

データノード間の直接接続を使用するクラスタをセットアップするには、クラスタの `config.ini` ファイルの `[tcp]` セクションに、そのように接続されるデータノードのクロスオーバー IP アドレスを明示的に指定する必要があります。

次の例では、少なくとも 4 台 (管理サーバー、SQL ノード、および 2 つのデータノード用に 1 台ずつ) のホストを含むクラスタについて考えます。クラスタ全体が LAN の `172.23.72.*` サブネットに配置されています。次に示すように、通常のネットワーク接続に加えて、2 つのデータノードが標準のクロスオーバーケーブルにより直接接続されており、`1.1.0.*` アドレス範囲内の IP アドレスを使用して相互に直接通信します。

```
# Management Server
[ndb_mgmd]
id=1
HostName=172.23.72.20

# SQL Node
[mysqld]
id=2
HostName=172.23.72.21

# Data Nodes
[ndbd]
id=3
HostName=172.23.72.22

[ndbd]
id=4
HostName=172.23.72.23

# TCP/IP Connections
[tcp]
Nodeld1=3
Nodeld2=4
HostName1=1.1.0.1
HostName2=1.1.0.2
```

`HostName1` および `HostName2` パラメータは、直接接続を指定するときのみ使用されます。

データノード間の直接 TCP 接続を使用すると、データノードがスイッチ、ハブ、ルーターなどの Ethernet デバイスをバイパスできるようになり、クラスタの待機時間が減るため、クラスタ全体の効率が向上する可能性があります。

注記

2 つ以上のデータノードでこの方法で直接接続を最大限に活用するには、各データノードと同じノードグループ内の他のすべてのデータノードとの間に直接接続が必要です。

23.3.3.12 NDB Cluster の共有メモリー接続

NDB クラスタノード間の通信は通常、TCP/IP を使用して処理されます。共有メモリー (SHM) トランスポータは、ソケット上ではなくメモリー内に書き込むことによってシグナルが送信されるという事実によって区別されます。共有メモリートランスポータ (SHM) は、同じホスト上で API ノード (通常は SQL ノード) とデータノードを同時に実行するときに TCP 接続に必要なオーバーヘッドの最大 20% を軽減することで、パフォーマンスを向上させることができます。共有メモリー接続は、次の 2 つの方法のいずれかで有効にできます:

- `UseShm` データノード構成パラメータを `1` に設定し、データノードには `HostName`、API ノードには `HostName` を同じ値に設定します。
- クラスタ構成ファイルの `[shm]` セクションを使用することで、それぞれに `Nodeld1` および `Nodeld2` の設定が含まれます。このメソッドの詳細は、このセクションの後半で説明します。

クラスタが、ノード ID 1 を持つデータノードと、`10.0.0.1` の同じホストコンピュータ上にノード ID 51 を持つ SQL ノードを実行しているとします。これら 2 つのノード間の SHM 接続を有効にするには、次のエントリがクラスタ構成ファイルに含まれていることを確認する必要があります:

```
[ndbd]
Nodeid=1
HostName=10.0.0.1
UseShm=1

[mysqld]
Nodeid=51
HostName=10.0.0.1
```

重要

ここで示した 2 つのエントリは、クラスタで必要な他のエントリおよびパラメータ設定に追加されています。より完全な例は、このセクションの後半で示します。

SHM 接続を使用するデータノードを起動する前に、そのようなデータノードをホストする各コンピュータ上のオペレーティングシステムに、共有メモリーセグメントに十分なメモリーが割り当てられていることも確認する必要があります。詳細は、ご使用のオペレーティングプラットフォームのドキュメントを参照してください。複数のホストがそれぞれデータノードと API ノードを実行している設定では、構成ファイルの[ndbd default]セクションで `UseShm` を設定することによって、このようなすべてのホストで共有メモリーを有効にできます。これをこのセクションの後半の例に示します。

厳密には必要ありませんが、クラスタ構成 (`config.ini`) ファイルの[shm default]セクションで次のパラメータのいずれかまたは複数を設定することで、クラスタ内のすべての SHM 接続のチューニングを実行できます:

- `ShmSize`: 共有メモリーサイズ
- `ShmSpinTime`: スリープ前にスピンする時間 (秒)
- `SendBufferMemory`: このノードから送信されたシグナルのバッファサイズ (バイト単位)。
- `SendSignalId`: トランスポータを介して送信される各シグナルにシグナル ID が含まれていることを示します。
- `Checksum`: トランスポータを介して送信される各シグナルにチェックサムが含まれることを示します。
- `PreSendChecksum`: チェックサムのチェックはシグナルの送信前に行われます。これが機能するにはチェックサムも有効にする必要があります

この例は、NDB Cluster 内の複数のホストに SHM 接続が定義されている単純な設定を示しています。ここにリストされている 3 台のコンピュータをホスト名別で使用し、示されているノードタイプをホストします:

1. `10.0.0.0`: 管理サーバー
2. `10.0.0.1`: データノードと SQL ノード
3. `10.0.0.2`: データノードと SQL ノード

このシナリオでは、各データノードは TCP トランスポータを使用して管理サーバーとほかのデータノードの両方と通信します。各 SQL ノードは共有メモリートランスポータを使用してローカルのデータノードと通信し、TCP トランスポータを使用してリモートデータノードと通信します。この設定を反映する基本構成は、次に示す内容の `config.ini` ファイルによって有効になります:

```
[ndbd default]
DataDir=/path/to/datadir
UseShm=1

[shm default]
ShmSize=8M
ShmSpintime=200
SendBufferMemory=4M

[tcp default]
SendBufferMemory=8M

[ndb_mgmd]
Nodeid=49
Hostname=10.0.0.0
DataDir=/path/to/datadir

[ndbd]
Nodeid=1
```

```

Hostname=10.0.0.1
DataDir=/path/to/datadir

[ndbd]
Nodeid=2
Hostname=10.0.0.2
DataDir=/path/to/datadir

[mysqld]
Nodeid=51
Hostname=10.0.0.1

[mysqld]
Nodeid=52
Hostname=10.0.0.2

[api]
[api]

```

すべての共有メモリートランスポータに影響するパラメータは、`[shm default]`セクションで設定されます。これらは、1つ以上の`[shm]`セクションで接続ごとにオーバーライドできます。このような各セクションは、`Nodeid1` および `Nodeid2` を使用して特定の SHM 接続に関連付ける必要があります。これらのパラメータに必要な値は、トランスポータによって接続された2つのノードのノード ID です。`HostName1` および `HostName2` を使用してホスト名でノードを識別することもできますが、これらのパラメータは必須ではありません。

ホスト名が設定されていない API ノードは、TCP トランスポータを使用して、起動されたホストから独立したデータノードと通信します。構成ファイルの`[tcp default]`セクションで設定されたパラメータと値は、クラスタ内のすべての TCP トランスポータに適用されます。

最適なパフォーマンスを得るために、SHM トランスポータ (`ShmSpinTime` パラメータ) のスピン時間を定義できます。これは、NDB のデータノード受信側スレッドとポーリング所有者 (受信スレッドまたはユーザースレッド) の両方に影響します。

- Checksum

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	boolean
デフォルト	true
範囲	true, false
再起動タイプ	N (NDB 8.0.13)

このパラメータはブール (Y/N) パラメータであり、デフォルトで無効になっています。有効にすると、送信バッファに配置される前にすべてのメッセージのチェックサムが計算されます。

この機能によって、送信バッファで待機中のメッセージの破損が回避されます。これは、トランスポート時のデータ破損に対するチェックとしても機能します。

- グループ

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	unsigned
デフォルト	35
範囲	0 - 200
再起動タイプ	N (NDB 8.0.13)

グループの近接度を決定します。小さい値は近いと解釈されます。ほとんどの場合、デフォルト値で十分です。

- HostName1

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	name or IP address

デフォルト	[...]
範囲	...
再起動タイプ	N (NDB 8.0.13)

[HostName1](#) および [HostName2](#) パラメータを使用すると、2つのノード間の特定の SHM 接続で使用する特定のネットワークインタフェースを指定できます。これらのパラメータに使用する値は、ホスト名または IP アドレスです。

- [HostName2](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	name or IP address
デフォルト	[...]
範囲	...
再起動タイプ	N (NDB 8.0.13)

[HostName1](#) および [HostName2](#) パラメータを使用すると、2つのノード間の特定の SHM 接続で使用する特定のネットワークインタフェースを指定できます。これらのパラメータに使用する値は、ホスト名または IP アドレスです。

- [Nodeld1](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	numeric
デフォルト	[none]
範囲	1 - 255
再起動タイプ	N (NDB 8.0.13)

2つのノード間の接続を識別するには、各ノードのノード識別子を [Nodeld1](#) および [Nodeld2](#) として指定する必要があります。

- [Nodeld2](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	numeric
デフォルト	[none]
範囲	1 - 255
再起動タイプ	N (NDB 8.0.13)

2つのノード間の接続を識別するには、各ノードのノード識別子を [Nodeld1](#) および [Nodeld2](#) として指定する必要があります。

- [NodeldServer](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	numeric
デフォルト	[none]
範囲	1 - 63
再起動タイプ	N (NDB 8.0.13)

共有メモリー接続のサーバー側を識別します。デフォルトでは、これはデータノードのノード ID です。

- [OverloadLimit](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	bytes
デフォルト	0
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

送信バッファにこれより多くの未送信バイトがあるときは、接続が過負荷状態であるとみなされます。詳細は、[セクション23.3.3.14「NDB Cluster 送信バッファパラメータの構成」](#)および[セクション23.5.14.55「ndbinfo transporters テーブル」](#)を参照してください。

- [PreSendChecksum](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	boolean
デフォルト	false
範囲	true, false
再起動タイプ	

このパラメータと [Checksum](#) の両方が有効になっている場合は、送信前チェックサムチェックを実行し、ノード間のすべての SHM シグナルにエラーがないかどうかを確認します。Checksum も有効になっていない場合は、何の効果もありません。

- [SendBufferMemory](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	2M
範囲	256K - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

共有メモリー接続を使用してこのノードから送信されたシグナルの共有メモリーバッファのサイズ (バイト)。

- [SendSignalId](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	boolean
デフォルト	false
範囲	true, false
再起動タイプ	N (NDB 8.0.13)

配信されたメッセージの経路をたどるには、各メッセージに一意の識別子を設定する必要があります。このパラメータを Y に設定すると、メッセージ ID もネットワーク経由で転送されるようになります。この機能は、製品ビルドではデフォルトで無効になっており、`-debug` ビルドで有効になります。

- [ShmKey](#)

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	unsigned
デフォルト	0
範囲	0 - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

共有メモリーセグメントを設定するときは、整数で表されるノード ID を使用して、通信に使用する共有メモリーセグメントを一意に識別します。デフォルト値はありません。UseShm が有効な場合、共有メモリーキーは NDB によって自動的に計算されます。

- ShmSize

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	bytes
デフォルト	4M
範囲	64K - 4294967039 (0xFFFFFFFF)
再起動タイプ	N (NDB 8.0.13)

各 SHM 接続には、送信側と読み取り側でノード間のメッセージを配置する共有メモリーセグメントがあります。このセグメントのサイズは ShmSize で定義されます。デフォルトの値は 4M バイトです。

- ShmSpinTime

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	integer
デフォルト	0
範囲	0 - 2000
再起動タイプ	

受信時にスリープ前に待機する時間 (マイクロ秒)。

- SigNum

バージョン (またはそれ以降)	NDB 8.0.13
タイプまたは単位	unsigned
デフォルト	[...]
範囲	0 - 4294967039 (0xFFFFFFFF)
非推奨	Yes (in NDB 7.6)
再起動タイプ	N (NDB 8.0.13)

このパラメータは以前はオペレーティングシステムのシグナル番号をオーバーライドするために使用されていました。NDB 8.0 では使用されなくなり、その設定は無視されます。

再起動のタイプ。このセクションのパラメータの説明で使用する再起動タイプに関する情報を次のテーブルに示します:

表 23.22 NDB Cluster の再起動タイプ

シンボル	再起動タイプ	説明
N	ノード	パラメータはローリング再起動を使用して更新できます (セクション 23.5.5 「NDB Cluster のローリング再起動の実行」 を参照)
S	システム	このパラメータの変更を有効にするには、すべてのクラスタノードを完全に停止してから再起動する必要があります
I	Initial	--initial オプションを使用してデータノードを再起動する必要があります

23.3.3.13 データノードのメモリー管理

データノードのすべてのメモリー割り当ては、ノードの起動時に実行されます。これにより、スワップメモリーを使用せずにデータノードを安定した方法で実行できるため、NDB を待機時間に依存する (リアルタイム) アプリケーションに使用できます。データノードの起動時には、次のタイプのメモリーが割り当てられます:

- データメモリー
- 共有グローバルメモリー
- redo ログバッファ
- ジョブバッファ
- バッファの送信
- ディスクデータレコードのページキャッシュ
- スキーマトランザクションメモリー
- トランザクションメモリー
- undo ログバッファ
- クエリーメモリー
- ブロックオブジェクト
- スキーマメモリー
- ブロックデータ構造
- ロング信号メモリー
- 共有メモリー通信バッファ

ほとんどのデータノードのメモリーを規制する NDB メモリーマネージャーは、次のメモリーリソースを処理します:

- データメモリー ([DataMemory](#))
- redo ログバッファ ([RedoBuffer](#))
- ジョブバッファ
- バッファの送信 ([SendBufferMemory](#), [TotalSendBufferMemory](#), [ExtraSendBufferMemory](#), [ReservedSendBufferMemory](#))
- ディスクデータレコードページキャッシュ ([DiskPageBufferMemory](#)、[DiskPageBufferEntries](#))
- トランザクションメモリー ([TransactionMemory](#))
- クエリーメモリー
- ディスクアクセスレコード
- ファイルバッファ

これらの各リソースは、予約済メモリー領域と最大メモリー領域を使用して設定されます。予約済メモリー領域は、予約されているリソースでのみ使用でき、他のリソースと共有することはできません。特定のリソースは、そのリソースに許可されている最大メモリーを超えることはできません。最大メモリーがないリソースは、メモリーマネージャーのすべての共有メモリーを使用するように拡張できます。

これらのリソースのグローバル共有メモリーのサイズは、[SharedGlobalMemory](#) 構成パラメータによって制御されます (デフォルト): 128 MB).

データメモリーは常に予約され、共有メモリーからメモリーを取得することはありません。これは、最大 16384 GB の [DataMemory](#) 構成パラメータを使用して制御されます。[DataMemory](#) では、ハッシュインデックス (行当たり約 15 バイト)、順序付けされたインデックス (インデックス当たり 10-12 バイト) および行ヘッダー (行当たり 16-32 バイト) を含むレコードが格納されます。

redo ログバッファも予約済メモリーのみを使用します。これは、LDM スレッドごとの redo ログバッファのサイズを設定する [RedoBuffer](#) 構成パラメータによって制御されます。つまり、実際に使用されるメモリー量は、このパラメータの値にデータノード内の LDM スレッドの数を乗算した値になります。

ジョブバッファは予約済メモリーのみを使用します。このメモリーのサイズは、様々なタイプのスレッド数に基づいて [NDB](#) によって計算されます。

送信バッファには予約された部分がありますが、共有グローバルメモリーの 25% を追加で割り当てることもできます。送信バッファの予約サイズは、次の 2 つの手順で計算されます:

1. [TotalSendBufferMemory](#) 構成パラメータの値 (デフォルト値なし) またはデータノードへのすべての個々の接続で使用される個々の送信バッファの合計を使用します。データノードは、ほかのすべてのデータノード、すべての API ノード、およびすべての管理ノードに接続されます。つまり、2 つのデータノード、2 つの管理ノード、および 10 個の API ノードを持つクラスタでは、各データノードに 13 個のノード接続があります。データノード接続の [SendBufferMemory](#) のデフォルト値は 2 MByte であるため、これは合計 26 MB になります。
2. 送信バッファの予約済サイズの合計を取得するには、[ExtraSendBufferMemory](#) 構成パラメータの値 (ある場合) (デフォルト値は 0)。が前のステップで取得した値に追加されます。

つまり、[TotalSendBufferMemory](#) が設定されている場合、送信バッファサイズは [TotalSendBufferMemory](#) + [ExtraSendBufferMemory](#) です。それ以外の場合、送信バッファのサイズは $([\text{number of node connections}] * \text{SendBufferMemory}) + \text{ExtraSendBufferMemory}$ と等しくなります。

ディスクデータレコードのページキャッシュは予約済リソースのみを使用します。このリソースのサイズは、[DiskPageBufferMemory](#) 構成パラメータ (デフォルトは 64 MB) によって制御されます。32 KB のディスクページエントリのメモリーも割り当てられます。これらの数は、[DiskPageBufferEntries](#) 構成パラメータ (デフォルトは 10) によって決まります。

トランザクションメモリーには、[NDB](#) によって計算される予約部分があるか、[NDB 8.0.19](#) で導入された [TransactionMemory](#) 構成パラメータを使用して明示的に設定されています (以前のリリースでは、この値は常に [NDB](#) によって計算されていました)。トランザクションメモリーでは、無制限の量の共有グローバルメモリーを使用することもできます。トランザクションメモリーは、トランザクション、スキャン、ロック、スキャンバッファおよびトリガー操作を処理するすべての操作リソースに使用されます。また、次のコミットによってデータメモリーに書き込まれる前に、テーブルの行も更新時に保持されます。

[NDB 8.0.16](#) 以前では、操作レコードは、サイズが多数の構成パラメータによって制御されていた専用リソースを使用していました。[NDB 8.0.17](#) 以降、これらはすべて共通トランザクションメモリーリソースから割り当てられ、グローバル共有メモリーのリソースを使用することもできます。[NDB 8.0.19](#) 以降では、このリソースのサイズは単一の [TransactionMemory](#) 構成パラメータを使用して制御できます。

undo ログバッファの予約済メモリーは、[InitialLogFileGroup](#) 構成パラメータを使用して設定できます。undo ログバッファが [CREATE LOGFILE GROUP](#) SQL ステートメントの一部として作成された場合、メモリーはトランザクションメモリーから取得されます。

ディスクデータリソースのメタデータに関連する多くのリソースには予約された部分もなく、共有グローバルメモリーのみが使用されます。したがって、共有グローバル共有メモリーは、送信バッファ、トランザクションメモリー間で共有されます。ディスクデータのメタデータ。

[TransactionMemory](#) が設定されていない場合、次のパラメータに基づいて計算されます:

- [MaxNoOfConcurrentOperations](#)
- [MaxNoOfConcurrentTransactions](#)
- [MaxNoOfFiredTriggers](#)
- [MaxNoOfLocalOperations](#)

- [MaxNoOfConcurrentIndexOperations](#)
- [MaxNoOfConcurrentScans](#)
- [MaxNoOfLocalScans](#)
- [BatchSizePerLocalScan](#)
- [TransactionBufferMemory](#)

[TransactionMemory](#) が明示的に設定されている場合、リストされている構成パラメータはメモリーサイズの計算に使用されません。また、パラメータ [MaxNoOfConcurrentIndexOperations](#), [MaxNoOfFiredTriggers](#), [MaxNoOfLocalOperations](#) および [MaxNoOfLocalScans](#) は [TransactionMemory](#) と互換性がなく、同時に設定することはできません。[TransactionMemory](#) が設定され、これら 4 つのパラメータのいずれかが `config.ini` 構成ファイルにも設定されている場合、管理サーバーは起動できません。これらの 4 つのパラメータは NDB 8.0.19 で非推奨になりました。MySQL NDB Cluster の将来のリリースから削除される予定です。

トランザクションメモリーリソースに多数のメモリープールが含まれています。各メモリープールはオブジェクトタイプを表し、オブジェクトのセットを含みます。各プールには、起動時にプールに割り当てられた予約部分が含まれます。この予約済メモリーは共有グローバルメモリーに戻されません。予約済レコードは、高速取得のための単一レベルのみを持つデータ構造を使用して検出されます。つまり、各プール内の多数のレコードを予約する必要があります。各プール内の予約済レコードの数は、パフォーマンスおよび予約済メモリーの割当てにある程度影響しますが、通常、予約済サイズを明示的に設定するには、非常に高度なユースケースでのみ必要です。

プールの予約部分のサイズは、次の構成パラメータを設定することで制御できます:

- [ReservedConcurrentIndexOperations](#)
- [ReservedFiredTriggers](#)
- [ReservedConcurrentOperations](#)
- [ReservedLocalScans](#)
- [ReservedConcurrentTransactions](#)
- [ReservedConcurrentScans](#)
- [ReservedTransactionBufferMemory](#)

前述のパラメータが設定されていない場合、予約済の設定はトランザクションメモリーの 25% です。予約レコードの数はデータノードごとです。これらのレコードは、各ノード上のそれら进行处理するスレッド (LDM および TC スレッド) 間で分割されます。ほとんどの場合、[TransactionMemory](#) のみを設定し、プール内のレコード数をその値で制御できるようにするだけで十分です。

[MaxNoOfConcurrentScans](#) は、各 TC スレッドでアクティブにできる同時スキャンの数を制限します。これは、クラスタの過負荷から保護するために重要です。

[MaxNoOfConcurrentOperations](#) では、トランザクションの更新時に一度にアクティブにできる操作の数が制限されます。(単純読取りは、このパラメータの影響を受けません。) この数を制限する必要があるのは、ノード障害処理用にメモリーを事前に割り当てる必要があり、ノード障害が発生した場合に 1 つの TC スレッドでアクティブな操作の最大数を処理するためにリソースを使用できる必要があるためです。すべてのノードで [MaxNoOfConcurrentOperations](#) を同じ数に設定する必要があります (これは、`config.ini` グローバル構成ファイルの `[ndbd default]` セクションで値を一度設定することで最も簡単に実行できます)。ローリング再起動を使用して値を増やすことはできますが ([セクション 23.5.5「NDB Cluster のローリング再起動の実行」](#) を参照)、ローリング再起動中にノード障害が発生する可能性があるため、この方法で値を減らすことは安全とはみなされません。

NDB Cluster 内の単一トランザクションのサイズは、[MaxDMLOperationsPerTransaction](#) パラメータを使用して制限できます。これが設定されていない場合、TC スレッド当たりの同時操作の合計数が制限されるため、あるトランザクションのサイズは [MaxNoOfConcurrentOperations](#) によって制限されます。

スキーマメモリーサイズは、次の一連の構成パラメータによって制御されます:

- [MaxNoOfSubscriptions](#)

- [MaxNoOfSubscribers](#)
- [MaxNoOfConcurrentSubOperations](#)
- [MaxNoOfAttributes](#)
- [MaxNoOfTables](#)
- [MaxNoOfOrderedIndexes](#)
- [MaxNoOfUniqueHashIndexes](#)
- [MaxNoOfTriggers](#)

各テーブルおよび各パーティション (およびそのフラグメントレプリカ) のパーティション数はスキーマメモリーで表す必要があるため、ノードの数と LDM スレッドの数もスキーマメモリーのサイズに大きく影響します。

また、他の多数のレコードが起動時に割り当てられます。これらは比較的小さいです。各スレッドの各ブロックには、メモリーを使用するブロックオブジェクトが含まれます。このメモリーサイズは、通常、ほかのデータノードメモリー構造に比べて非常に小さくなります。

23.3.3.14 NDB Cluster 送信バッファパラメータの構成

NDB カーネルは、すべてのトランスポートが共有するプールからメモリーが動的に割り当てられる統合送信バッファを採用します。これにより、送信バッファのサイズを必要に応じて調整できます。統合された送信バッファの構成は、次のパラメータを設定すると実現できます。

- **TotalSendBufferMemory.** このパラメータは、すべてのタイプの NDB Cluster ノードに対して設定できます。つまり、`config.ini` ファイルの `[ndbd]`、`[mgm]`、および `[api]` (または `[mysql]`) セクションで設定できます。これは、各ノードによって割り当てられ、すべての構成済みトランスポート間での使用に設定されるメモリーの合計量 (バイト単位) を表します。設定する場合、その最小は 256K バイト、最大は 4294967039 です。

既存の構成との下位互換性を確保するため、このパラメータのデフォルト値は、すべての構成済みトランスポートの最大送信バッファサイズの合計にトランスポートあたり 32K バイト (1 ページ) を加えた値になっています。最大は、次の表に示すように、トランスポートのタイプによって異なります。

表 23.23 最大送信バッファサイズのトランスポートタイプ

トランスポート	最大送信バッファサイズ (バイト)
TCP	<code>SendBufferMemory</code> (デフォルト = 2M)
SHM	20K

これにより、NDB Cluster 6.3 以前の場合とほぼ同じ方法で、同じ量のメモリーを使用して既存の構成を機能させ、各トランスポートで使用可能なバッファースペースを送信できます。ただし、特定のトランスポートで使用されていないメモリーは、ほかのトランスポートでも使用できません。

- **OverloadLimit.** このパラメータは、`config.ini` ファイルの `[tcp]` セクションで使用され、接続が過負荷状態であるとみなされる前に送信バッファに存在する未送信データの量 (バイト単位) を表します。このような過負荷状態が発生すると、過負荷の接続に影響を与えるトランザクションが失敗し、過負荷のステータスが終了するまで次のエラーが発生します: NDB API エラー 1218 (`Send Buffers overloaded in NDB kernel`)。デフォルト値は 0 です。その場合、特定の接続に対する実質的な過負荷の制限は `SendBufferMemory * 0.8` として計算されます。このパラメータの最大値は 4G です。
- **SendBufferMemory.** この値は、単一のトランスポートが `TotalSendBufferMemory` で指定されたプール全体から使用できるメモリー量に対する厳密な制限を表します。ただし、すべての構成済みトランスポートの `SendBufferMemory` の合計は、特定のノードに対して設定された `TotalSendBufferMemory` より大きくなることはありません。多数のノードが使用されているときは、すべてのトランスポートがメモリーの最大量を同時に必要としないかぎり、このような方法でメモリーが節約されます。

`ndbinfo.transporters` テーブルを使用して、送信バッファのメモリー使用量を監視し、パフォーマンスに悪影響を与える可能性のある低速および過負荷状態を検出できます。

23.3.4 NDB Cluster での高速インターコネクトの使用

1996年に **NDBCLUSTER** の設計の開始前に、並列データベースの構築で発生する主な問題の1つがネットワーク内のノード間の通信であることは認識されていました。このため、**NDBCLUSTER** は当初から複数の異なるデータトランスポートメカニズムを使用できるように設計されました。このマニュアルでは、このメカニズムに対しトランスポートという用語を使用します。

NDB Cluster コードベースには、4つの異なるトランスポートが用意されています:

- [セクション23.3.3.10「NDB Cluster TCP/IP 接続」](#)で説明している 100M ビット/秒またはギガビット Ethernet を使用した TCP/IP。
- ダイレクト (マシン間) TCP/IP。このトランスポートは前の項目で示したものと同一 TCP/IP プロトコルを使用しますが、ハードウェアを異なる方法でセットアップする必要があり、構成方法も異なります。このため、NDB Cluster 用の個別のトランスポートメカニズムとみなされます。詳細は、[セクション23.3.3.11「直接接続を使用した NDB Cluster TCP/IP 接続」](#)を参照してください。
- 共有メモリー (SHM)。SHM の詳細は、[セクション23.3.3.12「NDB Cluster の共有メモリー接続」](#)を参照してください。
- スケーラブルコヒーレントインタフェース (SCI)。

注記

NDB Cluster で SCI トランスポートを使用するには、NDB 8.0 で使用できない特殊なハードウェア、ソフトウェア、および MySQL バイナリが必要です。

ほとんどのユーザーは現在、広く普及していることから、TCP/IP over Ethernet を採用しています。TCP/IP は、NDB Cluster で使用するための最適なトランスポートでもあります。

使用されるトランスポートに関係なく、**NDB** は、すべてのタイプのデータ転送に利点があるため、できるだけ大きなチャンクを使用してデータノードプロセスとの通信を確実に実行しようとします。

23.4 NDB Cluster プログラム

NDB Cluster を使用および管理するには、この章で説明するいくつかの特殊なプログラムが必要です。NDB Cluster 内のこれらのプログラムの目的、プログラムの使用方法、および各プログラムで使用可能な起動オプションについて説明します。

これらのプログラムには、NDB Cluster データ、管理、および SQL ノードプロセス (`ndbd`, `ndbmtd`, `ndb_mgmd` および `mysqld`) と管理クライアント (`ndb_mgm`) が含まれます。

NDB Cluster Auto-Installer の起動に使用されるプログラム `ndb_setup.py` (現在は非推奨) に関する情報もこのセクションに含まれています。[セクション23.4.26「ndb_setup.py — NDB Cluster 用ブラウザベースの Auto-Installer の起動 \(非推奨\)」](#)にはコマンドラインクライアントに関する情報のみが含まれていることに注意してください。このプログラムによって生成された GUI インストーラを使用して NDB Cluster を構成および配備する方法については、[The NDB Cluster Auto-Installer \(NDB 7.5\) \(No longer supported\)](#) を参照してください。

NDB Cluster プロセスとしての `mysqld` の使用については、[セクション23.5.9「NDB Cluster での MySQL Server の使用」](#)を参照してください。

NDB Cluster ディストリビューションには、その他の **NDB** ユーティリティ、診断、およびサンプルプログラムが含まれています。これらには、`ndb_restore`、`ndb_show_tables`、および `ndb_config` が含まれます。これらのプログラムについてもこのセクションで説明します。

このセクションの最後の部分には、さまざまな NDB Cluster プログラムに共通のオプションのテーブルが含まれています。

23.4.1 ndbd — NDB Cluster データノードデーモン

nbd は、NDB Cluster ストレージエンジンを使用してテーブル内のすべてのデータを処理するために使用されるプロセスです。これは、分散型トランザクションの処理、ノードのリカバリ、ディスクでのチェックポイントの実行、オンラインバックアップ、および関連するタスクをデータノードが行うことができるようにするプロセスです。

NDB Cluster では、一連の nbd プロセスが連携してデータを処理します。これらのプロセスは、同じコンピュータ (ホスト) 上または別個のコンピュータ上で実行できます。データノードと Cluster ホストの通信は詳細に構成できます。

次のテーブルに、NDB Cluster データノードプログラム nbd に固有のコマンドオプションを示します。追加説明が表のあとにあります。ほとんどの NDB Cluster プログラム (nbd を含む) に共通のオプションについては、[セクション 23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#) を参照してください。

表 23.24 プログラムで使用されるコマンドライン・オプション nbd

形式	説明	追加、非推奨、または削除された
<code>--bind-address=name</code>	ローカルバインドアドレス	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--connect-delay=#</code>	管理サーバーへの接続を試行する間隔 (秒)。0 は試行の間待機しないことを意味	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--connect-retries=#</code>	中止する前に接続を再試行する回数を設定します。0 は 1 回の試行のみを意味し、再試行は行われません	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--connect-retry-delay=#</code>	管理サーバーへの接続を試行する間隔 (秒)。0 は試行の間待機しないことを意味	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--daemon,</code> <code>-d</code>	nbd をデーモンとして開始します (デフォルト)。オーバーライドするには <code>--nodaemon</code> を指定します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--foreground</code>	nbd をフォアグラウンドで実行します。デバッグのために提供されています (<code>--nodaemon</code> の意味を含みます)	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--initial</code>	nbd の初期起動 (ファイルシステムのクリーンアップを含む) を実行します。このオプションを使用する前にドキュメントを参照してください	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--initial-start</code>	部分的な初期起動を実行します (<code>--nowait-nodes</code> が必要です)	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--install[=name]</code>	データノードプロセスを Windows サービスとしてインストールするために使用します。他のプラットフォームには適用されません	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--logbuffer-size=#</code>	ログバッファのサイズを制御します。多くのログメッセージを生成してデバッグする場合に使用します。通常の操作にはデフォルトで十分です	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--nodaemon</code>	nbd をデーモンとして開始しません。テストのために提供されています	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--nostart,</code> <code>-n</code>	nbd をすぐに起動しないでください。nbd はコマンドが <code>ndb_mgm</code> から起動するまで待機	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--nowait-nodes=list</code>	これらのデータノードが起動するまで待機しないでください (ノード ID の	(MySQLに基づくすべてのNDBリリースでサポート 8.0)

形式	説明	追加、非推奨、または削除された
<code>--remove[=name]</code>	コマンド区切りリストを取ります)。 -- ndb-nodeid が必要です 以前に Windows サービスとしてインストールされたデータノードプロセスを削除するために使用します。他のプラットフォームには適用されません	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--verbose,</code> <code>-v</code>	追加のデバッグ情報をノードログに書き込みます	(MySQLに基づくすべてのNDBリリースでサポート 8.0)

注記

これらのオプションはすべて、このプログラム (ndbmysqld) のマルチスレッドバージョンにも適用され、このセクションで後者が発生した場合は常に、「ndbd」のかわりに「ndbmysqld」を使用できます。

- `--bind-address`

コマンド行形式	<code>--bind-address=name</code>
型	文字列
デフォルト値	

ndbd が特定のネットワークインタフェース (ホスト名または IP アドレス) にバインドされます。このオプションにはデフォルト値はありません。

- `--connect-delay=#`

コマンド行形式	<code>--connect-delay=#</code>
非推奨	はい
型	数値
デフォルト値	5
最小値	0
最大値	3600

起動時に管理サーバーへの接続を試行する間隔を決定します (試行回数は `--connect-retries` オプションによって制御されます)。デフォルトは 5 秒です。

このオプションは非推奨であり、NDB Cluster の将来のリリースで削除される可能性があります。かわりに `--connect-retry-delay` を使用してください。

- `--connect-retries=#`

コマンド行形式	<code>--connect-retries=#</code>
型	数値
デフォルト値	12
最小値	0
最大値	65535

中止する前に接続を再試行する回数を設定します。0 は 1 回の試行のみを意味し、再試行は行われません。デフォルトは 12 回の試行です。試行間の待機時間は、`--connect-retry-delay` オプションによって制御されます。

- `--connect-retry-delay=#`

コマンド行形式	<code>--connect-retry-delay=#</code>
---------	--------------------------------------

型	数値
デフォルト値	5
最小値	0
最大値	4294967295

起動時に管理サーバーへの各接続試行の間に待機する時間を決定します (試行の間の時間は `--connect-retries` オプションによって制御されます)。デフォルトは 5 秒です。

このオプションは `--connect-delay` オプションの代わりに使用され、現在は非推奨であり、NDB Cluster の将来のリリースで削除される可能性があります。

- `--daemon`、`-d`

コマンド行形式	<code>--daemon</code>
型	Boolean
デフォルト値	TRUE

`nbd` または `nbdmtd` にデーモンプロセスとして実行するように指示します。これはデフォルトの動作です。`--nodaemon` を使用すると、プロセスがデーモンとして実行されなくなります。

Windows プラットフォームで `nbd` または `nbdmtd` を実行している場合、このオプションは効果がありません。

- `--foreground`

コマンド行形式	<code>--foreground</code>
型	Boolean
デフォルト値	FALSE

主にデバッグのために、`nbd` または `nbdmtd` がフォアグラウンドプロセスとして実行されます。このオプションは `--nodaemon` オプションの意味を含みます。

Windows プラットフォームで `nbd` または `nbdmtd` を実行している場合、このオプションは効果がありません。

- `--initial`

コマンド行形式	<code>--initial</code>
型	Boolean
デフォルト値	FALSE

`nbd` に初期起動を実行するように指示します。初期起動によって、以前の `nbd` のインスタンスでリカバリのために作成されたファイルが消去されます。また、リカバリログファイルが再作成されます。一部のオペレーティングシステムでは、このプロセスにかなりの時間がかかる場合があります。

`--initial` の起動は、非常に特殊な状況下で `nbd` プロセスを起動するときのみを使用します。これは、このオプションによって NDB Cluster ファイルシステムからすべてのファイルが削除され、すべての redo ログファイルが再作成されるためです。それらの状況を次に示します。

- ファイルの内容が変更されたソフトウェアアップグレードを実行する場合。
- 新しいバージョンの `nbd` でノードを再起動する場合。
- 何かの理由でノードまたはシステムの再起動が繰り返し失敗する場合の最後の手段として。この場合、データファイルが破壊されるため、このノードはデータのリストアに使用できなくなります。

警告

最終的なデータ損失の可能性を回避するために、`--initial` オプションを `StopOnError = 0` とともに使用しないことをお勧めします。かわりに、クラスタの起動後にのみ `config.ini`

で `StopOnError` を 0 に設定し、データノードを通常どおり (`--initial` オプションなしで) 再起動します。この問題の詳細は、`StopOnError` パラメータの説明を参照してください。(Bug #24945638)

このオプションを使用すると、`StartPartialTimeout` および `StartPartitionedTimeout` 構成パラメータの効果がなくなります。

重要

このオプションは、影響を受けるノードによってすでに作成されているバックアップファイルには影響しません。

NDB 8.0.21 より前では、`--initial` オプションもディスクデータファイルに影響を与えてきました。NDB 8.0.21 以降では、このオプションを使用してクラスタの初期再起動を実行すると、「ディスクデータ」テーブルスペースに関連付けられているすべてのデータファイルと、このデータノードに以前存在していたログファイルシステムに関連付けられているすべての Undo ログファイルが削除されます(セクション23.5.10「NDB Cluster ディスクデータテーブル」を参照)。

このオプションは、すでに実行されているデータノードから起動(または再起動)しているデータノードによるデータの回復にも影響しません(初期再起動の一環として `--initial` でも起動された場合を除く)。このデータの回復は自動的に行われ、正常に実行されている NDB Cluster でのユーザーの介入は必要ありません。

クラスタを最初に起動するとき(つまり、データノードファイルが作成される前)にこのオプションを使用することはできますが、そうする必要はありません。

- `--initial-start`

コマンド行形式	<code>--initial-start</code>
型	Boolean
デフォルト値	FALSE

このオプションは、クラスタを部分的に初期起動するときに使用します。各ノードは、このオプションおよび `--nowait-nodes` を指定して起動してください。

データノードの ID が 2、3、4、および 5 である 4 ノードクラスタがあり、ノード 2、4、および 5 のみを使用して(つまり、ノード 3 を除外して)部分的に初期起動を実行するとします。

```
shell> nbd --ndb-nodeid=2 --nowait-nodes=3 --initial-start
shell> nbd --ndb-nodeid=4 --nowait-nodes=3 --initial-start
shell> nbd --ndb-nodeid=5 --nowait-nodes=3 --initial-start
```

このオプションを使用する場合は、`--ndb-nodeid` オプションを指定して起動するデータノードのノード ID も指定する必要があります。

重要

複数の管理サーバーで構成されたクラスタをすべての管理サーバーがオンラインでなくても起動できるようにするために使用できる `ndb_mgmd` の `--nowait-nodes` オプションとこのオプションを混同しないでください。

- `--install[=name]`

コマンド行形式	<code>--install[=name]</code>
プラットフォーム固有	Windows
型	文字列

デフォルト値	ndbd
--------	------

ndbd が Windows サービスとしてインストールされます。必要に応じて、サービス名を指定できます。設定しない場合、サービス名は ndbd にデフォルト設定されます。ほかの ndbd プログラムオプションは my.ini または my.cnf 構成ファイルに指定することが推奨されますが、--install と一緒に使用できます。ただし、そのような場合、Windows サービスのインストールが成功するには、--install オプションを最初に指定してから、ほかのオプションを指定する必要があります。

一般的に、このオプションを --initial オプションと一緒に使用することはお勧めしません。それにより、サービスが停止および開始されるたびにデータノードファイルシステムが消去および再作成されるためです。データノードの起動に影響するほかの ndbd オプション (--initial-start、--nstart、および --nowait-nodes を含む) を --install と一緒に使用する場合は、それを行うことを十分に理解し、起こり得る結果に対して十分に準備していることをしっかりと確認してください。

--install オプションは、Windows 以外のプラットフォームでは効果がありません。

- --logbuffer-size=#

コマンド行形式	--logbuffer-size=#
型	Integer
デフォルト値	32768
最小値	2048
最大値	4294967295

データノードのログバッファのサイズを設定します。大量の追加ロギングを使用してデバッグする場合、ログメッセージが多すぎると、ログバッファの領域が不足する可能性があります。この場合、一部のログメッセージが失われる可能性があります。これは、通常の操作中には発生しません。

- --nodaemon

コマンド行形式	--nodaemon
型	Boolean
デフォルト値	FALSE

ndbd または ndbmtbd がデーモンプロセスとして実行されなくなります。このオプションは --daemon オプションをオーバーライドします。これは、バイナリをデバッグするときに、出力を画面にリダイレクトする場合に便利です。

Windows での ndbd および ndbmtbd のデフォルト動作はフォアグラウンドでの実行であり、Windows プラットフォームではこのオプションは効果がなく、必要ありません。

- --nstart, -n

コマンド行形式	--nstart
型	Boolean
デフォルト値	FALSE

ndbd に自動的に起動しないように指示します。このオプションを使用すると、ndbd は管理サーバーに接続してそこから構成データを取得し、通信オブジェクトを初期化します。ただし、管理サーバーによってそうするように明示的に要求されるまで、実行エンジンを実際に起動しません。これは、管理クライアントで適切な START コマンドを発行することによって実現されます (セクション 23.5.1 「NDB Cluster 管理クライアントのコマンド」を参照してください)。

- --nowait-nodes=node_id_1[, node_id_2[, ...]]

コマンド行形式	--nowait-nodes=list
型	文字列

デフォルト値

このオプションは、起動前にクラスタが待機しないデータノードのリストを取ります。

これは、パーティション化された状態のクラスタを起動するために使用できます。たとえば、4 ノードクラスタで実行されているデータノード (ノード 2、3、4、および 5) の半分のデータノードのみでクラスタを起動するには、`--nowait-nodes=3,5` を指定して各 `ndbd` プロセスを開始します。この場合、クラスタはノード 2 およびノード 4 が接続するとすぐに起動し、ノード 3 およびノード 5 が接続するのを (接続していない場合) `StartPartitionedTimeout` ミリ秒間待機しません。

前の例と同じクラスタを 1 つの `ndbd` を除外して起動する場合 (たとえば、ノード 3 のホストマシンでハードウェアの障害が発生している場合) は、`--nowait-nodes=3` を指定してノード 2、4、および 5 を起動します。その後、ノード 2、4 および 5 が接続されるとすぐにクラスタが起動し、ノード 3 が起動するのを待機しません。

- `--remove[=name]`

コマンド行形式	<code>--remove[=name]</code>
プラットフォーム固有	Windows
型	文字列
デフォルト値	<code>ndbd</code>

以前に Windows サービスとしてインストールされた `ndbd` プロセスを削除します。必要に応じて、アンインストールするサービスの名前を指定できます。設定しない場合、サービス名は `ndbd` にデフォルト設定されます。

`--remove` オプションは、Windows 以外のプラットフォームでは効果がありません。

- `--verbose, -v`

追加のデバッグ出力がノードログに書き込まれます。

`NODELOG DEBUG ON` および `NODELOG DEBUG OFF` を使用して、データノードの実行中にこの追加ロギングを有効または無効にすることもできます。

`ndbd` は、`config.ini` 構成ファイルの `DataDir` で指定されているディレクトリに配置される一連のログファイルを生成します。

これらのログファイルを次に示します。`node_id` は、ノードの一意の識別子です。たとえば、`ndb_2_error.log` はノード ID が 2 のデータノードによって生成されたエラーログです。

- `ndb_node_id_error.log` は、参照先の `ndbd` プロセスで発生したすべてのクラッシュのレコードを含むファイルです。このファイルの各レコードには、簡単なエラー文字列、およびこのクラッシュのトレースファイルへの参照が含まれています。このファイルの一般的なエントリを次に示します。

```
Date/Time: Saturday 30 July 2004 - 00:20:01
Type of error: error
Message: Internal program error (failed ndbrequire)
Fault ID: 2341
Problem data: DbtupFixAlloc.cpp
Object of reference: DBTUP (Line: 173)
ProgramName: NDB Kernel
ProcessID: 14909
TraceFile: ndb_2_trace.log.2
***EOM***
```

データノードプロセスが予期せずに停止されたときに生成される可能性がある `ndbd` の終了コードおよびメッセージのリストは、[Data Node Error Messages](#)にあります。

重要

このエラーログファイルの最後のエントリが最新のエントリであるとはかぎりません (その可能性もあまりありません)。このエラーログのエントリは、時系列順で一覧されず、`ndb_node_id_trace.log.next` ファイル (次を参照してください) で決定されるトレース

ファイルの順序に対応しています。エラーログのエントリは、このように循環的に上書きされ、順次的ではありません。

- `ndb_node_id_trace.log.trace_id` は、エラーが発生する直前に発生した内容を正確に記述したトレースファイルです。この情報は、NDB Cluster 開発チームによる分析に役立ちます。

古いファイルが上書きされる前に作成されるこれらのトレースファイルの数を構成できます。`trace_id` は、連続するトレースファイルごとに増分される数値です。

- `ndb_node_id_trace.log.next` は、割り当てられる次のトレースファイル番号を追跡するファイルです。
- `ndb_node_id_out.log` は、`ndbd` プロセスによるデータ出力が含まれているファイルです。このファイルは、`ndbd` がデーモンとして開始された (デフォルトの動作) 場合にのみ作成されます。
- `ndb_node_id.pid` には、`ndbd` プロセスがデーモンとして開始された場合のプロセス ID が含まれています。これは、同じ識別子でノードが起動されるのを防ぐためのロックファイルとしても機能します。
- `ndb_node_id_signal.log` は、`ndbd` のデバッグバージョンでのみ使用されるファイルであり、`ndbd` プロセスのそれらのデータのすべての受信、送信、および内部メッセージをトレースできます。

NFS を使用してマウントしたディレクトリは使用しないことをお勧めします。一部の環境では、プロセスが終了しても `.pid` ファイルに対するロックが有効なままとなる問題が発生することがあるためです。

`ndbd` を開始する場合、管理サーバーのホスト名およびそれが待機するポートも指定する必要があることがあります。必要に応じて、プロセスが使用するノード ID を指定することもできます。

```
shell> ndbd --connect-string="nodeid=2;host=ndb_mgmd.mysql.com:1186"
```

これについての詳細は、[セクション23.3.3.3「NDB Cluster 接続文字列」](#)を参照してください。[セクション23.4.32「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#)では、`ndbd` で使用できるその他のコマンド行オプションについて説明しています。データノード構成パラメータについては、[セクション23.3.3.6「NDB Cluster データノードの定義」](#)を参照してください。

`ndbd` が開始されると、実際には 2 つのプロセスが開始されます。最初のプロセスは「エンジェルプロセス」と呼ばれ、その唯一の役割は実行プロセスが完了したときにそれを検出し、`ndbd` プロセスを再起動することです (そのように構成されている場合)。このため、UNIX の `kill` コマンドを使用して `ndbd` を強制終了しようとする場合は、両方のプロセスを強制終了する必要があります (エンジェルプロセスが最初)。`ndbd` プロセスを終了させるための推奨される方法は、管理クライアントを使用してそこからプロセスを停止することです。

実行プロセスは、データの読み取り、書き込み、スキャン、およびほかのすべてのアクティビティーに 1 つのスレッドを使用します。数千の同時アクションを容易に処理できるように、このスレッドは非同期に実装されます。また、監視スレッドは実行スレッドが無限ループでハングアップしないように管理します。スレッドのプールによってファイル I/O が処理され、各スレッドは 1 つのオープンファイルを処理できます。スレッドは `ndbd` プロセスのトランスポーターによるトランスポーター接続に使用することもできます。多数の操作 (更新を含む) を実行するマルチプロセッサシステムの場合、`ndbd` プロセスは最大 2 つの CPU を使用できます (許可されている場合)。

多数の CPU を持つマシンの場合は、異なるノードグループに属する複数の `ndbd` プロセスを使用できます。ただし、そのような構成はまだ実験的と見なされ、MySQL 8.0 の本番設定ではサポートされません。[セクション23.1.7「NDB Cluster の既知の制限事項」](#)を参照してください。

23.4.2 ndbinfo_select_all — ndbinfo テーブルからの選択

`ndbinfo_select_all` は、`ndbinfo` データベースの 1 つ以上のテーブルからすべての行およびカラムを選択するクライアントプログラムです

`mysql` クライアントで使用可能なすべての `ndbinfo` テーブルをこのプログラムで読み取ることができるわけではありません。また、`ndbinfo_select_all` では、`tables` および `columns` メタデータテーブルなど、SQL を使用してアクセスできない `ndbinfo` 内部の一部のテーブルに関する情報を表示できます。

`ndbinfo_select_all` を使用して 1 つ以上の `ndbinfo` テーブルから選択するには、プログラムを呼び出すときに次のようにテーブル名を指定する必要があります。

```
shell> ndbinfo_select_all table_name1 [table_name2] [...]
```

例:

```

shell> ndbinfo_select_all logbuffers logspaces
== logbuffers ==
node_id log_type  log_id log_part  total used  high
5  0  0  0  33554432  262144  0
6  0  0  0  33554432  262144  0
7  0  0  0  33554432  262144  0
8  0  0  0  33554432  262144  0
== logspaces ==
node_id log_type  log_id log_part  total used  high
5  0  0  0  268435456  0  0
5  0  0  1  268435456  0  0
5  0  0  2  268435456  0  0
5  0  0  3  268435456  0  0
6  0  0  0  268435456  0  0
6  0  0  1  268435456  0  0
6  0  0  2  268435456  0  0
6  0  0  3  268435456  0  0
7  0  0  0  268435456  0  0
7  0  0  1  268435456  0  0
7  0  0  2  268435456  0  0
7  0  0  3  268435456  0  0
8  0  0  0  268435456  0  0
8  0  0  1  268435456  0  0
8  0  0  2  268435456  0  0
8  0  0  3  268435456  0  0
shell>

```

次の表には、`ndbinfo_select_all` に固有のオプションが含まれています。追加説明が表のあとにあります。ほとんどの NDB Cluster プログラム (`ndbinfo_select_all` を含む) に共通のオプションについては、[セクション23.4.32「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#) を参照してください。

表 23.25 プログラムで使用されるコマンドライン・オプション `ndbinfo_select_all`

形式	説明	追加、非推奨、または削除された
<code>--delay=#</code>	ループ間の遅延を秒単位で設定	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--loops=#,</code> <code>-l</code>	選択を実行する回数の設定	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--database=db_name,</code> <code>-d</code>	テーブルが存在するデータベースの名前	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--parallelism=#,</code> <code>-p</code>	並列度の設定	(MySQLに基づくすべてのNDBリリースでサポート 8.0)

- `--delay=seconds`

コマンド行形式	<code>--delay=#</code>
型	数値
デフォルト値	5
最小値	0
最大値	MAX_INT

このオプションは、ループの各実行の間に待機する秒数を設定します。`--loops` に 0 または 1 が設定されている場合は効果がありません。

- `--loops=number, -l number`

コマンド行形式	<code>--loops=#</code>
---------	------------------------

型	数値
デフォルト値	1
最小値	0
最大値	MAX_INT

このオプションは選択を実行する回数を設定します。各ループの時間間隔を設定するには、`--delay` を使用します。

23.4.3 ndbmtd — NDB Cluster データノードデーモン (マルチスレッド)

`ndbmtd` はマルチスレッドバージョンの `ndbd` であり、`NDBCLUSTER` ストレージエンジンを使用してテーブル内のすべてのデータを処理するために使用されるプロセスです。`ndbmtd` は、複数の CPU コアを持つホストコンピュータでの使用を目的としています。特に明記されていないかぎり、`ndbmtd` は `ndbd` と同じ方法で機能するため、このセクションでは、`ndbmtd` が `ndbd` と異なる方法に集中しており、シングルスレッドバージョンとマルチスレッドバージョンの両方のデータノードプロセスに適用される NDB Cluster データノードの実行に関する追加情報については、[セクション23.4.1「ndbd — NDB Cluster データノードデーモン」](#) に問い合わせてください。

`ndbd` で使用されるコマンド行オプションおよび構成パラメータは、`ndbmtd` にも当てはまります。これらのオプションおよびパラメータについては、それぞれ[セクション23.4.1「ndbd — NDB Cluster データノードデーモン」](#) および[セクション23.3.3.6「NDB Cluster データノードの定義」](#)を参照してください。

また、`ndbmtd` は `ndbd` とファイルシステム互換です。言い換えると、`ndbd` を実行しているデータノードを停止し、バイナリを `ndbmtd` に置き換えて、データ損失なしに再起動できます (ただし、`ndbmtd` をマルチスレッド方式で実行する場合は、ノードを再起動する前に、`MaxNoOfExecutionThreads` が適切な値に設定されていることを確認する必要があります。) 同様に、ノードを停止し、マルチスレッドバイナリのかわりに `ndbd` を起動するだけで、`ndbmtd` バイナリを `ndbd` に置き換えることができます。これらの2つを切り替えるときに、`--initial` を使用してデータノードバイナリを開始する必要はありません。

`ndbmtd` を使用した場合は、`ndbd` を使用した場合と2つの重要な点で異なります。

1. `ndbmtd` はデフォルトではシングルスレッドモードで実行されるため (つまり、`ndbd` のように動作します)、マルチスレッドを使用するように構成する必要があります。これを行うには、`config.ini` ファイルの `MaxNoOfExecutionThreads` 構成パラメータまたは `ThreadConfig` 構成パラメータに適切な値を設定します。`MaxNoOfExecutionThreads` は簡単に使用できますが、`ThreadConfig` にはより柔軟性があります。これらの構成パラメータおよびその使用方法については、[マルチスレッドの構成パラメータ \(ndbmtd\)](#)を参照してください。
2. トレースファイルは `ndbmtd` プロセスで重大なエラーが発生したときに生成されますが、`ndbd` で障害が発生したときにこれらが生成される仕組みとは若干異なります。これらの相違については、次のいくつかの段落で説明します。

`ndbd` と同様に、`ndbmtd` は `config.ini` 構成ファイルの `DataDir` で指定されたディレクトリに配置される一連のログファイルを生成します。トレースファイルを除き、これらは `ndbd` で生成されるものと同様に生成され、同じ名前が付けられます。

重大なエラーが発生した場合、`ndbmtd` はエラーが発生する直前に発生した事象を示すトレースファイルを生成します。これらのファイルはデータノード `DataDir` にあり、NDB Cluster 開発およびサポートチームによる問題の分析に役立ちます。各 `ndbmtd` スレッドに対して1つのトレースファイルが生成されます。これらのファイルの名前には次のパターンがあります。

```
ndb_node_id_trace.log.trace_id_tthread_id,
```

このパターンで、`node_id` はクラスタ内のデータノード一意のノード ID を表し、`trace_id` はトレースシーケンス番号であり、`thread_id` はスレッド ID です。たとえば、NDB Cluster データノードとして実行されている `ndbmtd` プロセスがノード ID 3 を持ち、`MaxNoOfExecutionThreads` が 4 である場合、4 つのトレースファイルがデータノードデータディレクトリに生成されます。このノードで障害が発生したのがはじめてであった場合、これらのファイルには `ndb_3_trace.log.1_t1`、`ndb_3_trace.log.1_t2`、`ndb_3_trace.log.1_t3`、および `ndb_3_trace.log.1_t4` と名前が付けられます。内部的には、これらのトレースファイルは `ndbd` トレースファイルと同じ形式です。

データノードプロセスが予期せずに停止されたときに生成される `ndbd` の終了コードおよびメッセージは、`ndbmtd` でも使用されます。これらのリストについては、[Data Node Error Messages](#)を参照してください。

注記

`ndbd` と `ndbmtid` は、同じ NDB Cluster 内の異なるデータノードで同時に使用できます。ただし、そのような構成は十分にテストされていないため、現時点で本番設定でこれを行うことはお勧めできません。

23.4.4 ndb_mgmd — NDB Cluster 管理サーバーデーモン

管理サーバーは、クラスタ構成ファイルを読み取り、この情報を要求したクラスタ内のすべてのノードにそれを配布するプロセスです。また、これはクラスタのアクティビティーに関するログを管理します。管理クライアントは、管理サーバーに接続してクラスタのステータスをチェックできます。

次のテーブルには、NDB Cluster 管理サーバープログラム `ndb_mgmd` に固有のオプションが含まれています。追加説明が表のあとにあります。ほとんどの NDB Cluster プログラム (`ndb_mgmd` を含む) に共通のオプションについては、[セクション23.4.32「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#)を参照してください。

表 23.26 プログラムで使用されるコマンドライン・オプション `ndb_mgmd`

形式	説明	追加、非推奨、または削除された
<code>--bind-address=host</code>	ローカルバインドアドレス	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--cluster-config-suffix=name</code>	my.cnf ファイルの <code>cluster_config</code> セクションの読み取り時にデフォルトのグループサフィクスをオーバーライドします。テストで使用されます	追加: NDB 8.0.24
<code>--config-cache[=TRUE FALSE]</code>	管理サーバー構成キャッシュを有効にします。デフォルトは true です	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--config-file=file,</code> <code>-f</code>	クラスタ構成ファイルを指定します。構成キャッシュが存在する場合は、 <code>--reload</code> または <code>--initial</code> も指定してオーバーライド	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--configdir=directory,</code> <code>--config-dir=directory</code>	クラスタ管理サーバー構成キャッシュディレクトリの指定	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--daemon,</code> <code>-d</code>	<code>ndb_mgmd</code> をデーモンモードで実行します (デフォルト)	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--initial</code>	構成キャッシュをバイパスして、管理サーバーが構成ファイルから構成データをリロード	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--install[=name]</code>	管理サーバープロセスを Windows サービスとしてインストールするために使用されます。他のプラットフォームには適用されません	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--interactive</code>	<code>ndb_mgmd</code> をインタラクティブモードで実行します (本番では正式にサポートされていません。テストのためのみです)	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--log-name=name</code>	このノードに適用されるクラスタログメッセージの書き込み時に使用する名前	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--mycnf</code>	my.cnf ファイルからクラスタ構成データを読み取ります	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--no-nodeid-checks</code>	ノード ID チェックを指定しないでください	(MySQLに基づくすべてのNDBリリースでサポート 8.0)

形式	説明	追加、非推奨、または削除された
<code>--nodaemon</code>	ndb_mgmd をデーモンとして実行しません	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--nowait-nodes=list</code>	この管理サーバーの起動時に指定された管理ノードを待機しません。--ndb-nodeid オプションが必要です	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--print-full-config,</code> <code>-P</code>	すべての構成を出力して終了します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--reload</code>	管理サーバーが構成ファイルを構成キャッシュと比較	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--remove[=name]</code>	以前に Windows サービスとしてインストールされた管理サーバープロセスを削除するために使用します。オプションで、削除するサービスの名前を指定します。他のプラットフォームには適用されません	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--verbose,</code> <code>-v</code>	ログへの追加情報の書込み	(MySQLに基づくすべてのNDBリリースでサポート 8.0)

- `--bind-address=host`

コマンド行形式	<code>--bind-address=host</code>
型	文字列
デフォルト値	<code>[none]</code>

管理サーバーを特定のネットワークインタフェース (ホスト名または IP アドレス) にバインドします。このオプションにはデフォルト値はありません。

- `cluster-config-suffix`

コマンド行形式	<code>--cluster-config-suffix=name</code>
導入	8.0.24-ndb-8.0.24
型	文字列
デフォルト値	<code>[none]</code>

`my.cnf` でクラスタ構成セクションを読み取るときに、デフォルトのグループサフィクスをオーバーライドします。テストで使用されます。

- `--config-cache`

コマンド行形式	<code>--config-cache[=TRUE FALSE]</code>
型	Boolean
デフォルト値	<code>TRUE</code>

このオプション (デフォルト値は 1 (または TRUE、ON)) は、管理サーバーが起動するたびに `config.ini` から構成が読み取られるように (セクション23.3.3「NDB Cluster 構成ファイル」を参照してください) その構成キャッシュを無効にするために使用できます。これを行うには、次のいずれかのオプションを指定して `ndb_mgmd` プロセスを開始します。

- `--config-cache=0`
- `--config-cache=FALSE`
- `--config-cache=OFF`

- `--skip-config-cache`

一覧したいいずれかのオプションの使用が有効となるのは、管理サーバーが開始されたときに格納されている構成がない場合のみです。管理サーバーが構成キャッシュファイルを見つけると、`--config-cache` オプションまたは `--skip-config-cache` オプションは無視されます。このため、構成キャッシュを無効にするには、管理サーバーを最初に起動するときにこのオプションを使用してください。それ以外の場合、つまり、構成キャッシュがすでに作成されている管理サーバーの構成キャッシュを無効にする場合は、管理サーバーを停止して、既存の構成キャッシュファイルを手動で削除してから、`--skip-config-cache` を指定して (または `--config-cache` に 0、OFF、または FALSE を設定して) 管理サーバーを再起動する必要があります。

構成キャッシュファイルは通常、インストールディレクトリの下 `mysql-cluster` という名前のディレクトリに作成されます (`--configdir` オプションを使用してこの場所がオーバーライドされていない場合)。管理サーバーが構成データを更新するたびに、新しいキャッシュファイルが生成されます。このファイルは、次の形式を使用して作成順に連続した名前が付けられます。

```
ndb_node-id_config.bin.seq-number
```

`node-id` は管理サーバーのノード ID であり、`seq-number` は 1 から始まるシーケンス番号です。たとえば、管理サーバーのノード ID が 5 の場合、最初の 3 つの構成キャッシュファイルが作成されるときに `ndb_5_config.bin.1`、`ndb_5_config.bin.2`、および `ndb_5_config.bin.3` という名前が付けられます。

実際にキャッシュを無効にせずに、構成キャッシュをパージまたはリロードする場合は、`--skip-config-cache` オプションの代わりに `--reload` オプションまたは `--initial` オプションのいずれかを指定して `ndb_mgmd` を開始してください。

構成キャッシュを再度有効にするには、構成キャッシュを無効にするために前に使用した `--config-cache` オプションまたは `--skip-config-cache` オプションを指定せずに管理サーバーを再起動します。

`--skip-config-cache` が使用された場合、`ndb_mgmd` は構成ディレクトリ (`--configdir`) をチェックせずに作成しようとします。(Bug #13428853)

- `--config-file=filename, -f filename`

コマンド行形式	<code>--config-file=file</code>
型	ファイル名
デフォルト値	[none]

構成ファイルとして使用すべきファイルを管理サーバーに指示します。デフォルトでは、管理サーバーは `ndb_mgmd` 実行可能ファイルと同じディレクトリで `config.ini` という名前のファイルを探します。それ以外の場合は、ファイル名および場所を明示的に指定する必要があります。

このオプションにはデフォルト値はなく、`--reload` または `--initial` オプションを指定して `ndb_mgmd` が起動されたか、管理サーバーが構成キャッシュを見つけることができなかつたために、管理サーバーが構成ファイルを読み取るように強制された場合を除き、無視されます。このオプションは、`--config-cache=OFF` を指定して `ndb_mgmd` が開始された場合にも読み取られます。詳細は、[セクション23.3.3「NDB Cluster 構成ファイル」](#)を参照してください。

- `--configdir=dir_name`

コマンド行形式	<code>--configdir=directory</code> <code>--config-dir=directory</code>
型	ファイル名
デフォルト値	<code>\$INSTALLDIR/mysql-cluster</code>

クラスタ管理サーバーの構成キャッシュディレクトリを指定します。`--config-dir` はこのオプションのエイリアスです。

- `--daemon`、`-d`

コマンド行形式	<code>--daemon</code>
型	Boolean
デフォルト値	TRUE

`ndb_mgmd` にデーモンプロセスとして開始するように指示します。これはデフォルトの動作です。

このオプションは `ndb_mgmd` を Windows プラットフォームで実行している場合は効果がありません。

- `--initial`

コマンド行形式	<code>--initial</code>
型	Boolean
デフォルト値	FALSE

構成データは、管理サーバーが開始されるたびにクラスタグローバル構成ファイルから読み取られるのではなく、内部的にキャッシュされます ([セクション23.3.3「NDB Cluster 構成ファイル」](#)を参照してください)。`--initial` オプションを使用するとこの動作がオーバーライドされ、管理サーバーが既存のキャッシュファイルを削除し、クラスタ構成ファイルから構成データを再度読み取り、新しいキャッシュを作成するように強制されます。

これは、2つの点で `--reload` オプションと異なります。まず、`--reload` を指定すると、サーバーが構成ファイルをキャッシュと照合し、ファイルの内容がキャッシュと異なる場合にのみデータをリロードすることが強制されます。2番目に、`--reload` は既存のキャッシュファイルを削除しません。

`--initial` を指定して `ndb_mgmd` が呼び出されたけれどもグローバル構成ファイルが見つからない場合、管理サーバーは起動できません。

管理サーバーは起動時に、同じ NDB Cluster 内の別の管理サーバーをチェックし、ほかの管理サーバー構成データを使用しようとします。この動作は、複数の管理ノードを持つ NDB Cluster のローリング再起動を実行するときに影響を及ぼします。詳細は、[セクション23.5.5「NDB Cluster のローリング再起動の実行」](#)を参照してください。

`--config-file` オプションとともに使用すると、キャッシュは構成ファイルが実際に見つかった場合にのみクリアされます。

- `--install[=name]`

コマンド行形式	<code>--install[=name]</code>
プラットフォーム固有	Windows
型	文字列
デフォルト値	<code>ndb_mgmd</code>

`ndb_mgmd` が Windows サービスとしてインストールされます。必要に応じて、サービスの名前を指定できます。設定しない場合、サービス名は `ndb_mgmd` にデフォルト設定されます。その他の `ndb_mgmd` プログラムオプションは `my.ini` または `my.cnf` 構成ファイルに指定することが推奨されますが、`--install` と一緒に使用できます。ただし、そのような場合、Windows サービスのインストールが成功するには、`--install` オプションを最初に指定してから、ほかのオプションを指定する必要があります。

このオプションを `--initial` オプションと一緒に使用することは一般的にお勧めしません。サービスが停止および開始されるたびに構成キャッシュが消去されて再作成されるためです。管理サーバーの起動に影響するほかの `ndb_mgmd` オプションを使用する場合にも注意すべきであり、それを行うことを十分に理解し、起こり得る結果に対して十分に準備していることをしっかりと確認してください。

`--install` オプションは、Windows 以外のプラットフォームでは効果がありません。

- `--interactive`

コマンド行形式	<code>--interactive</code>
---------	----------------------------

型	Boolean
デフォルト値	FALSE

ndb_mgmd をインタラクティブモードで開始します。つまり、管理サーバーが実行されるとすぐに ndb_mgm クライアントセッションが開始されます。このオプションは、ほかの NDB Cluster ノードを起動しません。

- `--log-name=name`

コマンド行形式	<code>--log-name=name</code>
型	文字列
デフォルト値	MgmtSrvr

クラスタログでこのノードに使用される名前を指定します。

- `--mycnf`

コマンド行形式	<code>--mycnf</code>
型	Boolean
デフォルト値	FALSE

my.cnf ファイルから構成データを読み取ります。

- `--no-nodeid-checks`

コマンド行形式	<code>--no-nodeid-checks</code>
型	Boolean
デフォルト値	FALSE

ノード ID のチェックを実行しません。

- `--nodaemon`

コマンド行形式	<code>--nodaemon</code>
型	Boolean
デフォルト値	FALSE

ndb_mgmd にデーモンプロセスとして開始しないように指示します。

Windows での ndb_mgmd のデフォルト動作はフォアグラウンドでの実行であるため、Windows プラットフォームではこのオプションは必要ありません。

- `--nowait-nodes`

コマンド行形式	<code>--nowait-nodes=list</code>
型	数値
デフォルト値	[none]
最小値	1
最大値	255

NDB Cluster を 2 つの管理ノードで起動する場合、各管理サーバーは通常、ほかの ndb_mgmd も動作しているかどうか、およびほかの管理サーバー構成がそれ自体と同じかどうかを確認します。ただし、クラスタを 1 つの管理ノードのみで起動すること (およびおそらく別の ndb_mgmd をあとで起動することを許可すること) が望ましい場合があります。このオプションを指定すると、このオプションに渡されたノード ID を持つ別の管理ノードを管理ノードがチェックせず、起動された管理ノードのみを使用するように構成されているかのようにクラスタが起動することを許可します。

説明のために、`config.ini` ファイルに次の部分があるものとします (ここでは、この例に関連しないほとんどの構成パラメータを省略しています)。

```
[ndbd]
Nodeid = 1
HostName = 198.51.100.101

[ndbd]
Nodeid = 2
HostName = 198.51.100.102

[ndbd]
Nodeid = 3
HostName = 198.51.100.103

[ndbd]
Nodeid = 4
HostName = 198.51.100.104

[ndb_mgmd]
Nodeid = 10
HostName = 198.51.100.150

[ndb_mgmd]
Nodeid = 11
HostName = 198.51.100.151

[api]
Nodeid = 20
HostName = 198.51.100.200

[api]
Nodeid = 21
HostName = 198.51.100.201
```

ノード ID が 10 で、IP アドレスが 198.51.100.150 のホストで動作する管理サーバーのみを使用して、このクラスタを起動するとします。(たとえば、別の管理サーバーを実行する予定のホストコンピュータがハードウェア障害のために一時的に使用できず、それが修復されるのを待っていると想定してください)。このようにクラスタを起動するには、198.51.100.150 のマシンのコマンド行を使用して、次のコマンドを入力します。

```
shell> ndb_mgmd --ndb-nodeid=10 --nowait-nodes=11
```

前の例に示されているように、`--nowait-nodes` を使用する場合は、`--ndb-nodeid` オプションも使用して、この `ndb_mgmd` プロセスのノード ID を指定する必要があります。

その後、クラスタの各データノードを通常の方法で起動できます。最初の管理サーバーに加えて、あとでデータノードを再起動せずに 2 番目の管理サーバーを起動して使用する場合は、両方の管理サーバーを参照する接続文字列を次のように指定して、各データノードを起動する必要があります。

```
shell> ndbd -c 198.51.100.150,198.51.100.151
```

このクラスタに接続された NDB Cluster SQL ノードとして起動する `mysqld` プロセスで使用される接続文字列についても同様です。詳細は、[セクション 23.3.3.3 「NDB Cluster 接続文字列」](#) を参照してください。

`ndb_mgmd` とともに使用すると、このオプションは別の管理ノードに関してのみ管理ノードの動作に影響します。`ndbd` または `ndbmtid` (全量より少ない数のデータノードでクラスタを起動できる) に使用される `--nowait-nodes` オプションと混同しないでください。このオプションをデータノードに使用した場合は、ほかのデータノードに関係する動作にのみ影響します。

複数の管理ノード ID は、カンマ区切りリストとしてこのオプションに渡すことができます。各ノード ID は、1 から 255 までである必要があります。実際には、同じ NDB Cluster に複数の管理サーバーを使用する (または必要な

場合)ことはほとんどありません。ほとんどの場合、このオプションに渡す必要があるのは、クラスタの起動時に使用しない単一の管理サーバーの単一ノード ID のみです。

注記

「欠けている」管理サーバーをあとで起動する場合は、クラスタですでに使用されている管理サーバーと構成が一致している必要があります。そうしないと、既存の管理サーバーによって実行される構成チェックで失敗して起動されません。

- `--print-full-config, -P`

コマンド行形式	<code>--print-full-config</code>
型	Boolean
デフォルト値	FALSE

クラスタの構成に関する詳細情報を表示します。このオプションをコマンド行に指定すると、`ndb_mgmd` プロセスはクラスタ設定に関する情報(クラスタ構成セクションの詳細なリスト、およびパラメータとその値を含む)を出力します。通常、`--config-file (-f)` オプションと一緒に使用します。

- `--reload`

コマンド行形式	<code>--reload</code>
型	Boolean
デフォルト値	FALSE

NDB Cluster 構成データは、管理サーバーが起動されるたびにクラスタグローバル構成ファイルから読み取られるのではなく、内部的に格納されます(セクション23.3.3「NDB Cluster 構成ファイル」を参照)。このオプションを使用すると、管理サーバーが内部データストアをクラスタ構成ファイルと照合し、構成ファイルがキャッシュと一致しないことが判明した場合は、構成をリロードすることを強制します。既存の構成キャッシュファイルは維持されますが使用されません。

これは、2つの点で `--initial` オプションと異なります。最初に、`--initial` ではすべてのキャッシュファイルが削除されます。2番目に、`--initial` は、グローバル構成ファイルを再度読み取って新しいキャッシュを作成することを管理サーバーに強制します。

管理サーバーがグローバル構成ファイルを見つけられない場合、`--reload` オプションは無視されます。

`--reload` を使用する場合、管理サーバーはグローバル構成ファイルを読み取ろうとする前に、クラスタ内のデータノードおよびその他の管理サーバーと通信できる必要があります。通信できない場合、管理サーバーは起動に失敗します。これは、ノードの新しい IP アドレスやファイアウォール構成の変更など、ネットワーク環境の変更が原因で発生する可能性があります。このような場合は、かわりに `--initial` を使用して、既存のキャッシュ構成を強制的に破棄し、ファイルからリロードする必要があります。詳細は、セクション23.5.5「NDB Cluster のローリング再起動の実行」を参照してください。

- `--remove[=name]`

コマンド行形式	<code>--remove[=name]</code>
プラットフォーム固有	Windows
型	文字列
デフォルト値	<code>ndb_mgmd</code>

Windows サービスとしてインストールされている管理サーバープロセスを削除します。オプションで、削除するサービスの名前を指定します。Windows プラットフォームにのみ適用されます。

- `--verbose, -v`

コマンド行形式	<code>--verbose</code>
型	Boolean

デフォルト値

FALSE

Windows サービスとしてインストールされている管理サーバープロセスを削除します。オプションで、削除するサービスの名前を指定します。Windows プラットフォームにのみ適用されます。

管理サーバーを起動するときに、接続文字列を必ず指定する必要があるわけではありません。ただし、複数の管理サーバーを使用している場合は、接続文字列を指定して、クラスタ内の各ノードにノード ID を明示的に指定してください。

接続文字列の使用方法については、[セクション23.3.3.3「NDB Cluster 接続文字列」](#)を参照してください。[セクション23.4.4「ndb_mgmd — NDB Cluster 管理サーバーデーモン」](#)では、`ndb_mgmd`のその他のオプションについて説明しています。

次のファイルは、起動ディレクトリにある `ndb_mgmd` によって作成または使用され、`config.ini` 構成ファイルに指定されている `DataDir` に配置されます。次のリストでは、`node_id` は一意のノード識別子です。

- `config.ini` は、クラスタ全体の構成ファイルです。このファイルはユーザーが作成し、管理サーバーによって読み取られます。[セクション23.3「NDB Cluster の構成」](#)では、このファイルをセットアップする方法について説明しています。

- `ndb_node_id_cluster.log` はクラスタイベントログファイルです。そのようなイベントの例としては、チェックポイントの開始と完了、ノードの起動イベント、ノードの障害、およびメモリー使用率のレベルが含まれます。クラスタイベントの完全なリストおよびその説明は、[セクション23.5「NDB Cluster の管理」](#)にあります。

デフォルトでは、クラスタログのサイズが 100 万バイトに達すると、ファイルの名前が `ndb_node_id_cluster.log.seq_id` に変更されます。ここで、`seq_id` はクラスタログファイルの順序番号です。(たとえば、シーケンス番号 1、2、3 を持つファイルがすでに存在する場合、次のログファイルは番号 4 を使用して名前が付けられます)。`LogDestination` 構成パラメータを使用して、ファイルのサイズと数、およびクラスタログのその他の特性を変更できます。

- `ndb_node_id_out.log` は、管理サーバーがデーモンとして実行されているときに、`stdout` および `stderr` のために使用されるファイルです。

- `ndb_node_id.pid` は、管理サーバーをデーモンとして実行しているときに使用されるプロセス ID ファイルです。

23.4.5 ndb_mgm — NDB Cluster 管理クライアント

`ndb_mgm` 管理クライアントプロセスは、クラスタを実行するために実際には必要ありません。その価値は、クラスタのステータスをチェックしたり、バックアップを開始したり、その他の管理機能を実行したりするための一連のコマンドを提供することにあります。管理クライアントは C API を使用して管理サーバーにアクセスします。上級ユーザーは、この API を使用して専用の管理プロセスをプログラムし、`ndb_mgm` によって実行されるタスクと同様のことを実行することもできます。

管理クライアントを開始するには、管理サーバーのホスト名およびポート番号を指定する必要があります。

```
shell> ndb_mgm [host_name [port_num]]
```

例:

```
shell> ndb_mgm ndb_mgmd.mysql.com 1186
```

デフォルトのホスト名およびポート番号は、それぞれ `localhost` および 1186 です。

次のテーブルに、NDB Cluster 管理クライアントプログラム `ndb_mgm` に固有のオプションを示します。追加説明が表のあとにあります。ほとんどの NDB Cluster プログラム (`ndb_mgm` を含む) に共通のオプションについては、[セクション23.4.32「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#)を参照してください。

表 23.27 プログラムで使用されるコマンドライン・オプション `ndb_mgm`

形式	説明	追加、非推奨、または削除された
<code>--connect-retries=#</code>	中止する前に接続を再試行する回数を設定します。0 は 1 回の試行のみを意味し、再試行は行われません	(MySQLに基づくすべてのNDBリリースでサポート 8.0)

形式	説明	追加、非推奨、または削除された
<code>--try-reconnect=#,</code> <code>-t</code> <code>--execute=name,</code> <code>-e</code>	切断する前に接続を再試行する回数を設定します。 <code>--connect-retries</code> のシノニムです コマンドを実行して終了します	(MySQLに基づくすべてのNDBリリースでサポート 8.0) (MySQLに基づくすべてのNDBリリースでサポート 8.0)

- `--connect-retries=#`

コマンド行形式	<code>--connect-retries=#</code>
型	数値
デフォルト値	3
最小値	0
最大値	4294967295

このオプションは、最初に接続を再試行してから中止するまでの回数を指定します (クライアントは常に接続を 1 回以上試行します)。試行ごとの待機時間は、`--connect-retry-delay` を使用して設定されます。

このオプションは、非推奨になった `--try-reconnect` オプションと同義です。

このオプションのデフォルトは、他の NDB プログラムで使用する場合のデフォルトとは異なります。詳しくは [セクション 23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#)、をご覧ください。

- `--execute=command, -e command`

コマンド行形式	<code>--execute=name</code>
---------	-----------------------------

このオプションを使用すると、NDB Cluster 管理クライアントにシステムシェルからコマンドを送信できます。たとえば、次のコマンドはいずれも管理クライアントで `SHOW` を実行することと同等です。

```
shell> ndb_mgm -e "SHOW"
```

```
shell> ndb_mgm --execute="SHOW"
```

これは `--execute` または `-e` オプションが `mysql` コマンド行クライアントで動作する仕組みに似ています。 [セクション 4.2.2.1 「コマンド行でのオプションの使用」](#) を参照してください。

注記

このオプションを使用して渡される管理クライアントコマンドにスペース文字が含まれている場合は、コマンドを引用符で囲む必要があります。一重引用符または二重引用符を使用できます。管理クライアントコマンドにスペース文字が含まれていない場合は、引用符を使用しなくてもかまいません。

- `--try-reconnect=number`

コマンド行形式	<code>--try-reconnect=#</code>
非推奨	はい
型	数値
型	Integer
デフォルト値	12
デフォルト値	3
最小値	0
最大値	4294967295

管理サーバーへの接続が切断された場合、ノードは 5 秒ごとに再接続を試みます (成功するまで)。このオプションを使用することで、接続を中止して代わりにエラーを報告する前に接続を試行する回数を `number` 回に制限できます。

このオプションは非推奨であり、将来のリリースで削除される可能性があります。かわりに `--connect-retries` を使用してください。

`ndb_mgm` の使用方法に関する追加情報については、[セクション 23.5.1 「NDB Cluster 管理クライアントのコマンド」](#) を参照してください。

23.4.6 ndb_blob_tool — NDB Cluster テーブルの BLOB および TEXT カラムのチェックおよび修復

このツールは、孤立した BLOB カラムパーツをチェックして NDB テーブルから削除するため、および孤立したパーツを一覧したファイルを生成するために使用できます。これは、BLOB または TEXT カラムが含まれている破損または損傷した NDB テーブルを診断および修復する場合に役に立つことがあります。

`ndb_blob_tool` の基本的な構文を次に示します。

```
ndb_blob_tool [options] table [column, ...]
```

`--help` オプションを使用する場合を除き、1 つ以上のオプション (`--check-orphans`、`--delete-orphans`、または `--dump-file`) を含めることによって、実行するアクションを指定する必要があります。これらのオプションを指定すると、`ndb_blob_tool` がそれぞれ孤立した BLOB パーツをチェックしたり、孤立した BLOB パーツを削除したり、孤立した BLOB パーツを一覧するダンプファイルを生成したりします。これについての詳細は、このセクションで後述します。

`ndb_blob_tool` を呼び出すときは、テーブルの名前も指定する必要があります。また、必要に応じて、テーブル名の後ろに、(カンマで区切った) そのテーブルの 1 つ以上の BLOB または TEXT カラムの名前を続けることができます。カラムをリストしない場合、このツールはテーブルのすべての BLOB および TEXT カラムを処理します。データベースを指定する必要がある場合は、`--database (-d)` オプションを使用します。

`--verbose` オプションは、ツールの進行状況に関する追加の情報を出力します。

次の表には、`ndb_blob_tool` に固有のオプションが含まれています。追加説明が表のあとにあります。ほとんどの NDB Cluster プログラム (`ndb_blob_tool` を含む) に共通のオプションについては、[セクション 23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#) を参照してください。

表 23.28 プログラムで使用されるコマンドライン・オプション `ndb_blob_tool`

形式	説明	追加、非推奨、または削除された
<code>--add-missing</code>	欠落しているものを実行するダミー BLOB パーツを書き込みます	追加: NDB 8.0.20
<code>--check-missing</code>	インラインパーツを持つがパーツテーブルに 1 つまたは複数のパーツがない BLOB をチェック	追加: NDB 8.0.20
<code>--check-orphans</code>	対応するインラインパーツがない BLOB パーツをチェック	(MySQLに基づくすべての NDB リリースでサポート 8.0)
<code>--database=db_name,</code> <code>-d</code>	テーブルを探すデータベース	(MySQLに基づくすべての NDB リリースでサポート 8.0)
<code>--delete-orphans</code>	対応するインラインパーツがない BLOB パーツの削除	(MySQLに基づくすべての NDB リリースでサポート 8.0)
<code>--dump-file=file</code>	指定したファイルに孤立したキーを書き込みます	(MySQLに基づくすべての NDB リリースでサポート 8.0)
<code>--verbose,</code> <code>-v</code>	冗長出力	(MySQLに基づくすべての NDB リリースでサポート 8.0)

- `--add-missing`

コマンド行形式	<code>--add-missing</code>
導入	8.0.20-ndb-8.0.20
型	Boolean
デフォルト値	FALSE

対応する BLOB 部分がない「NDB Cluster」テーブルのインライン部分ごとに、空白で構成される必要な長さのダミー BLOB 部分を記述します。

- `--check-missing`

コマンド行形式	<code>--check-missing</code>
導入	8.0.20-ndb-8.0.20
型	Boolean
デフォルト値	FALSE

対応する BLOB 部分がない「NDB Cluster」テーブルのインライン部分を確認します。

- `--check-orphans`

コマンド行形式	<code>--check-orphans</code>
型	Boolean
デフォルト値	FALSE

対応するインライン部分がない「NDB Cluster」テーブルの BLOB 部分を確認します。

- `--database=db_name, -d`

コマンド行形式	<code>--database=db_name</code>
型	文字列
デフォルト値	[none]

テーブルを探すデータベースを指定します。

- `--delete-orphans`

コマンド行形式	<code>--delete-orphans</code>
型	Boolean
デフォルト値	FALSE

対応するインライン部分がない BLOB 部分を「NDB Cluster」テーブルから削除します。

- `--dump-file=file`

コマンド行形式	<code>--dump-file=file</code>
型	ファイル名
デフォルト値	[none]

孤立した BLOB カラムパーツのリストを `file` に書き込みます。ファイルに書き込まれる情報には、各孤立した BLOB パーツのテーブルキーおよび BLOB パーツ番号が含まれます。

- `--verbose`

コマンド行形式	<code>--verbose</code>
---------	------------------------

型	Boolean
デフォルト値	FALSE

進行状況に関する追加情報をツールの出力に書き込みます。

例

最初に、次に示す `CREATE TABLE` ステートメントを使用して、`test` データベースに `NDB` テーブルを作成します。

```
USE test;

CREATE TABLE btest (
  c0 BIGINT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  c1 TEXT,
  c2 BLOB
) ENGINE=NDB;
```

その後、これに似た一連のステートメントを使用して、このテーブルにいくつかの行を挿入します。

```
INSERT INTO btest VALUES (NULL, 'x', REPEAT('x', 1000));
```

このテーブルに対して `--check-orphans` を指定して `ndb_blob_tool` を実行すると、次の出力が生成されます。

```
shell> ndb_blob_tool --check-orphans --verbose -d test btest
connected
processing 2 blobs
processing blob #0 c1 NDB$BLOB_19_1
NDB$BLOB_19_1: nextResult: res=1
total parts: 0
orphan parts: 0
processing blob #1 c2 NDB$BLOB_19_2
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=1
total parts: 10
orphan parts: 0
disconnected

NDBT_ProgramExit: 0 - OK
```

`c1` は `TEXT` カラムですが、カラム `c1` に関連付けられている `NDB BLOB` カラムパーツはないことをこのツールは報告しています。これは、`NDB` テーブルでは、`BLOB` または `TEXT` カラム値の最初の 256 バイトのみがインラインで格納され、それを超過する部分 (ある場合) のみが別個に格納されるためです。したがって、これらのタイプのいずれかのカラムに 256 バイトを超える値がない場合、このカラムの `BLOB` カラムパーツは `NDB` によって作成されません。詳細は、[セクション11.7「データ型のストレージ要件」](#)を参照してください。

23.4.7 ndb_config — NDB Cluster 構成情報の抽出

このツールは、複数のソースのいずれかからデータノード、SQL ノード、および API ノードの現在の構成情報を抽出: `NDB Cluster` 管理ノード、またはその `config.ini` または `my.cnf` ファイル。デフォルトでは、管理ノードが構成データのソースです。デフォルトをオーバーライドするには、`--config-file` オプションまたは `--mycnf` オプションを指定して `ndb_config` を実行します。 `--config_from_node=node_id` にノード ID を指定することによって、データノードをソースとして使用することもできます。

`ndb_config` は、使用できるすべての構成パラメータ、およびそれらのデフォルト値、最大値、最小値、その他の情報のオフラインダンプを生成することもできます。このダンプはテキスト形式または XML 形式で生成できます。詳細は、このセクションで後述する `--configinfo` および `--xml` オプションの説明を参照してください。

--nodes、--system、または--connections のいずれかのオプションを使用すると、結果をセクション (DB、SYSTEM、または CONNECTIONS) によってフィルタリングできます。

次の表には、ndb_config に固有のオプションが含まれています。追加説明が表のあとにあります。ほとんどの NDB Cluster プログラム (ndb_config を含む) に共通のオプションについては、[セクション23.4.32「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#)を参照してください。

表 23.29 プログラムで使用されるコマンドライン・オプション ndb_config

形式	説明	追加、非推奨、または削除された
--cluster-config-suffix=name	my.cnf ファイルの cluster_config セクションの読取り時にデフォルトのグループサフィクスをオーバーライドします。テストで使用されます	追加: NDB 8.0.24
--config-file=file_name	config.ini ファイルへのパスを設定します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
--config-from-node=#	この ID を持つノードから構成データを取得します (データノードである必要があります)	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
--configinfo	すべての NDB 構成パラメータに関する情報をデフォルト値、最大値、および最小値とともにテキスト形式でダンプします。XML 出力を取得するには、--xml とともに使用します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
--connections	接続情報 ([tcp]、[tcp default]、[sci]、[sci default]、[shm]、またはクラスタ構成ファイルの[shm default]セクション) のみを出力します。--system または --nodes とともに使用できません	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
--diff-default	デフォルト以外の値を持つ構成パラメータのみ出力	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
--fields=string,	フィールド区切り文字	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
-f		
--host=name	ホストを指定します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
--mycnf	my.cnf ファイルから構成データを読み取ります	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
--nodeid	この ID のノードの構成を取得します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
--nodes	ノード情報 ([ndbd]またはクラスタ構成ファイルの[ndbd default]セクション) のみを出力します。--system または --connections とともに使用できません	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
-c	--ndb-connectstring の短縮形	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
--query=string,	1 つ以上のクエリーオプション (属性)	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
-q		
--query-all,	すべてのパラメータと値を単一のカンマ区切り文字列にダンプ	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
-a		
--rows=string,	行区切り文字	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
-r		

形式	説明	追加、非推奨、または削除された
<code>--system</code>	「SYSTEM の印刷」セクション情報のみ (ndb_config --configinfo の出力を参照)。 --nodes または --connections とともに使用できません	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--type=name</code>	ノードタイプを指定します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--configinfo --xml</code>	すべての NDB 構成パラメータのダンプをデフォルト値、最大値、および最小値とともに XML 形式で取得するには、--configinfo に -xml を使用します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)

• `cluster-config-suffix`

コマンド行形式	<code>--cluster-config-suffix=name</code>
導入	8.0.24-ndb-8.0.24
型	文字列
デフォルト値	[none]

`my.cnf` でクラスタ構成セクションを読み取る際に、デフォルトのグループサフィクスをオーバーライドします。テストで使用されます。

• `--configinfo`

`--configinfo` オプションを指定すると、`ndb_config` は、`ndb_config` が属する NDB Cluster 配布でサポートされている各 NDB Cluster 構成パラメータのリストをダンプします。次の情報が含まれます:

- 各パラメータの目的、効果、および使用方法の簡単な説明
- パラメータを使用できる `config.ini` ファイルのセクション
- パラメータのデータ型または単位
- 該当する場合、パラメータのデフォルト値、最小値、および最大値
- NDB Cluster のリリースバージョンとビルド情報

デフォルトでは、この出力はテキスト形式です。この出力の一部を次に示します。

```
shell> ndb_config --configinfo
***** SYSTEM *****

Name (String)
Name of system (NDB Cluster)
MANDATORY

PrimaryMGMNode (Non-negative Integer)
Node id of Primary ndb_mgmd(MGM) node
Default: 0 (Min: 0, Max: 4294967039)

ConfigGenerationNumber (Non-negative Integer)
Configuration generation number
Default: 0 (Min: 0, Max: 4294967039)

***** DB *****

MaxNoOfSubscriptions (Non-negative Integer)
Max no of subscriptions (default 0 == MaxNoOfTables)
Default: 0 (Min: 0, Max: 4294967039)

MaxNoOfSubscribers (Non-negative Integer)
Max no of subscribers (default 0 == 2 * MaxNoOfTables)
Default: 0 (Min: 0, Max: 4294967039)
```

...

このオプションを `--xml` オプションとともに使用して、XML 形式で出力を取得します。

- `--config-file=path-to-file`

コマンド行形式	<code>--config-file=file_name</code>
型	ファイル名
デフォルト値	

管理サーバーの構成ファイル (`config.ini`) へのパスを指定します。これには相対パスまたは絶対パスを指定できます。`ndb_config` が呼び出されるホストと別のホストに管理ノードがある場合は、絶対パスを使用する必要があります。

- `--config_from_node=#`

コマンド行形式	<code>--config-from-node=#</code>
型	数値
デフォルト値	<code>none</code>
最小値	<code>1</code>
最大値	<code>48</code>

この ID を持つデータノードからクラスタの構成データを取得します。

この ID を持つノードがデータノードではない場合、`ndb_config` はエラーで失敗します (代わりに、管理ノードから構成データを取得するには、単純にこのオプションを省略します)。

- `--connections`

コマンド行形式	<code>--connections</code>
型	Boolean
デフォルト値	<code>FALSE</code>

`CONNECTIONS` 情報のみ (クラスタ構成ファイルの `[tcp]`, `[tcp default]`, `[shm]` または `[shm default]` セクションにあるパラメータに関する情報) を出力するように `ndb_config` に指示します (詳細は、[セクション23.3.3.10「NDB Cluster TCP/IP 接続」](#) および [セクション23.3.3.12「NDB Cluster の共有メモリー接続」](#) を参照)。

このオプションは、`--nodes` および `--system` と相互に排他的です。これらの3つのオプションのいずれかのみを使用できます。

- `--diff-default`

コマンド行形式	<code>--diff-default</code>
型	Boolean
デフォルト値	<code>FALSE</code>

デフォルト以外の値を持つ構成パラメータのみを出力します。

- `--fields=delimiter, -f delimiter`

コマンド行形式	<code>--fields=string</code>
型	文字列

デフォルト値	
--------	--

結果のフィールドを区切るために使用される `delimiter` 文字列を指定します。デフォルトは、`,` (カンマ) です。

注記

`delimiter` にスペースまたはエスケープ (改行文字の `\n` など) が含まれている場合は、引用符で囲む必要があります。

• `--host=hostname`

コマンド行形式	<code>--host=name</code>
型	文字列
デフォルト値	

構成情報を取得するノードのホスト名を指定します。

注記

ホスト名 `localhost` は通常、IP アドレス `127.0.0.1` に解決されますが、これはすべてのオペレーティングプラットフォームおよび構成でそうなるとはかぎりません。これは、`config.ini` に `localhost` が使用されて、`localhost` が別のアドレスに解決される (たとえば、一部のバージョンの SUSE Linux では、これは `127.0.0.2` です) 別のホストで `ndb_config` が実行されている場合、`ndb_config --host=localhost` が失敗することがあることを意味します。一般に、最良の結果を得るには、ホストに関連するすべての NDB Cluster 構成値に数値の IP アドレスを使用するか、すべての NDB Cluster ホストが同じ方法で `localhost` を処理することを確認するようにしてください。

• `--mycnf`

コマンド行形式	<code>--mycnf</code>
型	Boolean
デフォルト値	<code>FALSE</code>

`my.cnf` ファイルから構成データを読み取ります。

• `--ndb-connectstring=connection_string, -c connection_string`

コマンド行形式	<code>--ndb-connectstring=connectstring</code> <code>--connect-string=connectstring</code>
型	文字列
デフォルト値	<code>localhost:1186</code>

管理サーバーに接続するために使用する接続文字列を指定します。この接続文字列の形式は、[セクション 23.3.3.3「NDB Cluster 接続文字列」](#)で説明したものと同一であり、デフォルトは `localhost:1186` です。

• `--nodeid=node_id`

コマンド行形式	<code>--ndb-nodeid=#</code>
型	数値
デフォルト値	<code>0</code>

構成情報を取得するノードのノード ID を指定します。

• `--nodes`

コマンド行形式	<code>--nodes</code>
---------	----------------------

型	Boolean
デフォルト値	FALSE

クラスタ構成ファイルの[ndbd]または[ndbd default]セクションで定義されたパラメータにのみ関連する情報を出力するように `ndb_config` に指示します (セクション23.3.3.6「NDB Cluster データノードの定義」を参照)。

このオプションは、`--connections` および `--system` と相互に排他的です。これらの3つのオプションのいずれかのみを使用できます。

- `--query=query-options, -q query-options`

コマンド行形式	<code>--query=string</code>
型	文字列
デフォルト値	

これは、クエリーオプションのカンマ区切りのリストです。つまり、1つ以上の返されるノード属性のリストです。これには、`nodeid` (ノード ID)、`タイプ` (ノードタイプ、`ndbd`、`mysqlid` または `ndb_mgmd`) および値を取得する構成パラメータが含まれます。

たとえば、`--query=nodeid,type,datamemory,datadir` は各ノードのノード ID、ノードタイプ、`DataMemory` および `DataDir` を返します。

注記

指定したパラメータが特定のタイプのノードに当てはまらない場合、対応する値には空の文字列が返されます。詳細はこのセクションで後述する例を参照してください。

- `--query-all, -a`

コマンド行形式	<code>--query-all</code>
型	文字列
デフォルト値	

すべてのクエリーオプション (ノード属性) のカンマ区切りリストを返します。このリストは単一の文字列であることに注意してください。

- `--rows=separator, -r separator`

コマンド行形式	<code>--rows=string</code>
型	文字列
デフォルト値	

結果の行を区切るために使用される `separator` 文字列を指定します。デフォルトはスペース文字です。

注記

`separator` にスペースまたはエスケープ (改行文字の `\n` など) が含まれている場合は、引用符で囲む必要があります。

- `--system`

コマンド行形式	<code>--system</code>
型	Boolean

デフォルト値	FALSE
--------	-------

SYSTEM の情報のみを出力するように `ndb_config` に通知します。これは、実行時に変更できないシステム変数で構成されているため、クラスタ構成ファイルに対応するセクションはありません。これらは、`ndb_config --configinfo` の出力に表示されます (***** SYSTEM ***** という接頭辞が付いています)。

このオプションは、`--nodes` および `--connections` と相互に排他的です。これらの3つのオプションのいずれかのみを使用できます。

- `--type=node_type`

コマンド行形式	<code>--type=name</code>
型	列挙
デフォルト値	[none]
有効な値	ndbd mysqld ndb_mgmd

指定した `node_type` のノード (ndbd、mysqld、または ndb_mgmd) に該当する構成値のみが返されるように、結果をフィルタリングします。

- `--usage`、`--help` または `-?`

コマンド行形式	<code>--help</code> <code>--usage</code>
---------	---

`ndb_config` が使用可能なオプションのリストを出力してから終了します。

- `--version`、`-V`

コマンド行形式	<code>--version</code>
---------	------------------------

`ndb_config` がバージョン情報文字列を出力してから終了します。

- `--configinfo --xml`

コマンド行形式	<code>--configinfo --xml</code>
型	Boolean
デフォルト値	false

このオプションを追加して、`ndb_config --configinfo` が XML として出力を提供するようにします。このような出力の一部を次の例に示します:

```
shell> ndb_config --configinfo --xml
<configvariables protocolversion="1" ndbversionstring="5.7.33-ndb-7.5.21"
  ndbversion="460032" ndbversionmajor="7" ndbversionminor="5"
  ndbversionbuild="0">
<section name="SYSTEM">
  <param name="Name" comment="Name of system (NDB Cluster)" type="string"
    mandatory="true"/>
  <param name="PrimaryMGMNode" comment="Node id of Primary ndb_mgmd(MGM) node"
    type="unsigned" default="0" min="0" max="4294967039"/>
  <param name="ConfigGenerationNumber" comment="Configuration generation number"
    type="unsigned" default="0" min="0" max="4294967039"/>
</section>
<section name="MYSQLD" primarykeys="Nodeld">
  <param name="wan" comment="Use WAN TCP setting as default" type="bool"
    default="false"/>
```

```
<param name="HostName" comment="Name of computer for this node"
  type="string" default=""/>
<param name="Id" comment="NodeId" type="unsigned" mandatory="true"
  min="1" max="255" deprecated="true"/>
<param name="NodeId" comment="Number identifying application node (mysqld(API))"
  type="unsigned" mandatory="true" min="1" max="255"/>
<param name="ExecuteOnComputer" comment="HostName" type="string"
  deprecated="true"/>

...

</section>

...

</configvariables>
```

注記

通常、`ndb_config --configinfo --xml` によって生成される XML 出力では 1 行に 1 要素の形式が使用されますが、上記の例および次の例では読みやすくするために空白を追加しています。ほとんどの XML プロセッサでは通常の処理として不要な空白が無視されるか、無視するように指示できるため、これによってこの出力を使用するアプリケーションに違いは生じないはずで

この XML 出力は、特定のパラメータが変更されたときに、`--initial` オプションを使用してデータノードを再起動する必要があることも示しています。これは、対応する `<param>` 要素に `initial="true"` 属性が表示されることによって示されます。また、再起動タイプ (`system` または `node`) も示されます。これは、特定のパラメータでシステムの再起動が要求される場合に、対応する `<param>` 要素に `restart="system"` 属性が表示されることによって示されます。たとえば、`Diskless` パラメータの値セットを変更すると、次に示されているようにシステムの初期再起動が必要となります (`restart` 属性および `initial` 属性が見やすいように強調表示されています)。

```
<param name="Diskless" comment="Run wo/ disk" type="bool" default="false"
  restart="system" initial="true"/>
```

現在、初期再起動が必要ないパラメータに対応する `<param>` 要素の XML 出力には、`initial` 属性は表示されません。言い換えると、`initial="false"` がデフォルトであり、属性が表示されていない場合は値 `false` を想定してください。同様に、デフォルトの再起動タイプは `node` (つまり、クラスタのオンラインまたは「ローリング」再起動) ですが、`restart` 属性は再起動タイプが `system` (すべてのクラスタノードを同時にシャットダウンしてから再起動する必要があることを意味します) である場合にのみ表示されます。

非推奨のパラメータは、次に示すように、`deprecated` 属性によって XML 出力に示されます:

```
<param name="NoOfDiskPagesToDiskAfterRestartACC" comment="DiskCheckpointSpeed"
  type="unsigned" default="20" min="1" max="4294967039" deprecated="true"/>
```

このような場合、`comment` は、非推奨のパラメータよりも優先される 1 つ以上のパラメータを参照します。`initial` と同様に、`deprecated` 属性は、パラメータが `deprecated="true"` で非推奨の場合にのみ示され、非推奨ではないパラメータにはまったく表示されません。(Bug #21127135)

NDB 7.5.0 以降では、次に示すように、必要なパラメータは `mandatory="true"` で示されます:

```
<param name="NodeId"
  comment="Number identifying application node (mysqld(API))"
  type="unsigned" mandatory="true" min="1" max="255"/>
```

`initial` または `deprecated` 属性は、初期再起動が必要なパラメータまたは非推奨のパラメータに対してのみ表示されるのと同じ方法で、`mandatory` 属性は、指定されたパラメータが実際に必要な場合にのみ含まれます。

重要

`--xml` オプションは `--configinfo` オプションを指定した場合にのみ使用できます。 `--configinfo` を指定せずに `--xml` を使用すると、エラーで失敗します。

このプログラムで現在の構成データを取得するために使用されるオプションとは異なり、`--configinfo` および `--xml` は、`ndb_config` のコンパイル時に NDB Cluster ソースから取得された情報を使用します。このため、これらのオ

プシオンでは、実行中の NDB Cluster への接続や、`config.ini` または `my.cnf` ファイルへのアクセスは必要ありません。

他の `ndb_config` オプション (`--query`、`--type` など) と `--configinfo` (`--xml` オプションの有無にかかわらず) の組合せはサポートされていません。現在、そうしようとする通常、`--configinfo` または `--xml` 以外のすべてのオプションが単に無視されます。ただし、この動作は保証されず、今後変更されることがあります。また、`ndb_config` を `--configinfo` オプションとともに使用した場合、NDB Cluster にアクセスしたり、ファイルを読み取ったりしないため、`--ndb-connectstring` や `--config-file with --configinfo` などの追加オプションを指定しようとしても目的はありません。

例

1. クラスタのノード ID および各ノードのタイプを取得するには、次のようにします。

```
shell> ./ndb_config --query=nodeid,type --fields=':' --rows='\n'
1:ndbd
2:ndbd
3:ndbd
4:ndbd
5:ndb_mgmd
6:mysqlq
7:mysqlq
8:mysqlq
9:mysqlq
```

この例では、`--fields` オプションを使用して ID および各ノードのタイプをコロン (:) で区切り、`--rows` オプションを使用して各ノードの値を新しい行に出力しています。

2. データノード、SQL ノード、および API ノードが管理サーバーに接続するために使用できる接続文字列を生成するには、次のようにします。

```
shell> ./ndb_config --config-file=usr/local/mysql/cluster-data/config.ini \
--query=hostname,portnumber --fields=: --rows=, --type=ndb_mgmd
198.51.100.179:1186
```

3. このように `ndb_config` を起動すると、(`--type` オプションを使用して) データノードのみがチェックされ、各ノード ID とホスト名の値、およびその `DataMemory` と `DataDir` パラメータに設定された値が表示されます:

```
shell> ./ndb_config --type=ndbd --query=nodeid,host,datamemory,datadir -f ':' -r '\n'
1 : 198.51.100.193 : 83886080 : /usr/local/mysql/cluster-data
2 : 198.51.100.112 : 83886080 : /usr/local/mysql/cluster-data
3 : 198.51.100.176 : 83886080 : /usr/local/mysql/cluster-data
4 : 198.51.100.119 : 83886080 : /usr/local/mysql/cluster-data
```

この例では、短いオプションの `-f` および `-r` を使用してフィールドデリミタと行セパレータをそれぞれ設定し、短いオプションの `-q` を使用して取得するパラメータのリストを渡しています。

4. 特定の 1 つのホスト以外のホストを結果から除外するには、`--host` オプションを使用します。

```
shell> ./ndb_config --host=198.51.100.176 -f ':' -r '\n' -q id,type
3:ndbd
5:ndb_mgmd
```

また、この例では、短い形式 `-q` を使用して、問い合わせる属性を指定しています。

同様に、`--nodeid` オプションを使用して、特定の ID を持つノードに結果を制限できます。

23.4.8 ndb_delete_all — NDB テーブルからのすべての行の削除

`ndb_delete_all` は、指定された NDB テーブルからすべての行を削除します。これは、`DELETE` または `TRUNCATE TABLE` よりも速いことがあります。

使用法

```
ndb_delete_all -c connection_string tbl_name -d db_name
```

これは、`db_name` という名前のデータベースの `tbl_name` という名前のテーブルからすべての行を削除しています。MySQL で `TRUNCATE db_name.tbl_name` を実行することとまったく同じです。

次の表には、`ndb_delete_all` に固有のオプションが含まれています。追加説明が表のあとにあります。ほとんどの NDB Cluster プログラム (`ndb_delete_all` を含む) に共通のオプションについては、[セクション 23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#) を参照してください。

表 23.30 プログラムで使用されるコマンドライン・オプション `ndb_delete_all`

形式	説明	追加、非推奨、または削除された
<code>--database=dbname,</code> <code>-d</code>	テーブルを探すデータベースの名前	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--transactional,</code> <code>-t</code>	単一トランザクションで削除を実行します (操作の数が足りなくなることがあります)	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--tupscan</code>	TUP スキャンを実行します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--diskscan</code>	ディスクスキャンを実行します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)

- `--transactional, -t`

このオプションを使用すると、削除操作が単一のトランザクションとして実行されます。

警告

非常に大きいテーブルの場合は、このオプションを使用すると、クラスタで使用できる操作の数を越えることがあります。

NDB 8.0.18 より前では、このプログラムは、NDBT テストライブラリへの不要な依存関係のため、実行の完了時に `NDBT_ProgramExit - status` を出力しました。この依存関係は削除され、余分な出力がなくなりました。

23.4.9 ndb_desc — NDB テーブルの表示

`ndb_desc` は、1 つ以上の NDB テーブルの詳細な説明を出力します。

使用法

```
ndb_desc -c connection_string tbl_name -d db_name [options]
```

```
ndb_desc -c connection_string index_name -d db_name -t tbl_name
```

`ndb_desc` で使用できるその他のオプションは、このセクションの後半で説明します。

出力例

MySQL テーブル作成およびレコード挿入ステートメント:

```
USE test;

CREATE TABLE fish (
  id INT NOT NULL AUTO_INCREMENT,
  name VARCHAR(20) NOT NULL,
  length_mm INT NOT NULL,
  weight_gm INT NOT NULL,

  PRIMARY KEY pk (id),
  UNIQUE KEY uk (name)
) ENGINE=NDB;

INSERT INTO fish VALUES
  (NULL, 'guppy', 35, 2), (NULL, 'tuna', 2500, 150000),
  (NULL, 'shark', 3000, 110000), (NULL, 'manta ray', 1500, 50000),
  (NULL, 'grouper', 900, 125000), (NULL, 'puffer', 250, 2500);
```


ndb_desc の出力:

```

shell> ./ndb_desc -c localhost fish -d test -p
-- fish --
Version: 2
Fragment type: HashMapPartition
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: no
Number of attributes: 4
Number of primary keys: 1
Length of frm data: 337
Max Rows: 0
Row Checksum: 1
Row GCI: 1
SingleUserMode: 0
ForceVarPart: 1
PartitionCount: 2
FragmentCount: 2
PartitionBalance: FOR_RP_BY_LDM
ExtraRowGciBits: 0
ExtraRowAuthorBits: 0
TableStatus: Retrieved
Table options:
HashMap: DEFAULT-HASHMAP-3840-2
-- Attributes --
id Int PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY AUTO_INCR
name Varchar(20;latin1_swedish_ci) NOT NULL AT=SHORT_VAR ST=MEMORY DYNAMIC
length_mm Int NOT NULL AT=FIXED ST=MEMORY DYNAMIC
weight_gm Int NOT NULL AT=FIXED ST=MEMORY DYNAMIC
-- Indexes --
PRIMARY KEY(id) - UniqueHashIndex
PRIMARY(id) - OrderedIndex
uk(name) - OrderedIndex
uk$unique(name) - UniqueHashIndex
-- Per partition info --

```

Partition	Row count	Commit count	Frag fixed memory	Frag var sized memory	Extent_space	Free extent_space
0	2	2	32768	32768	0	0
1	4	4	32768	32768	0	0

```

NDBT_ProgramExit: 0 - OK

```

複数のテーブルに関する情報は、テーブル名をスペースで区切って `ndb_desc` を 1 回呼び出すことによって取得できます。すべてのテーブルは同じデータベースにある必要があります。

`--table` (短い形式) を使用して、特定のインデックスに関する追加情報を取得できます: `-t` オプションで、次に示すように、インデックスの名前を `ndb_desc` の最初の引数として指定します:

```

shell> ./ndb_desc uk -d test -t fish
-- uk --
Version: 2
Base table: fish
Number of attributes: 1
Logging: 0
Index type: OrderedIndex
Index status: Retrieved
-- Attributes --
name Varchar(20;latin1_swedish_ci) NOT NULL AT=SHORT_VAR ST=MEMORY
-- IndexTable 10/uk --
Version: 2
Fragment type: FragUndefined
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: yes
Number of attributes: 2
Number of primary keys: 1
Length of frm data: 0
Max Rows: 0
Row Checksum: 1
Row GCI: 1
SingleUserMode: 2

```

```
ForceVarPart: 0
PartitionCount: 2
FragmentCount: 2
FragmentCountType: ONE_PER_LDM_PER_NODE
ExtraRowGciBits: 0
ExtraRowAuthorBits: 0
TableStatus: Retrieved
Table options:
-- Attributes --
name Varchar(20;latin1_swedish_ci) NOT NULL AT=SHORT_VAR ST=MEMORY
NDB$TNODE Unsigned [64] PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY
-- Indexes --
PRIMARY KEY(NDB$TNODE) - UniqueHashIndex

NDBT_ProgramExit: 0 - OK
```

このようにインデックスを指定すると、`--extra-partition-info` および `--extra-node-info` オプションは効果がありません。

出力の `Version` カラムには、テーブルのスキーマオブジェクトバージョンが表示されます。この値の解釈については、[NDB Schema Object Versions](#) を参照してください。

`CREATE TABLE` および `ALTER TABLE` ステートメントに埋め込まれた `NDB_TABLE` コメントを使用して設定できるテーブルプロパティの 3 つは、`ndb_desc` 出力にも表示されます。`FRAGMENT_COUNT_TYPE` テーブルは、常に `FragmentCountType` カラムに表示されます。`READ_ONLY` および `FULLY_REPLICATED` は、1 に設定されている場合、`Table options` カラムに表示されます。これは、`mysql` クライアントで次の `ALTER TABLE` ステートメントを実行した後に確認できます:

```
mysql> ALTER TABLE fish COMMENT='NDB_TABLE=READ_ONLY=1,FULLY_REPLICATED=1';
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
+-----+-----+-----+-----+-----+-----+
| Level | Code | Message                                                                                               |
+-----+-----+-----+-----+-----+-----+
| Warning | 1296 | Got error 4503 'Table property is FRAGMENT_COUNT_TYPE=ONE_PER_LDM_PER_NODE but not in comment' from NDB |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

`READ_ONLY=1` では、テーブルフラグメント数タイプが `ONE_PER_LDM_PER_NODE_GROUP` である (またはこれに設定されている) 必要があるため、警告が発行されます。このような場合、`NDB` ではこれが自動的に設定されます。`SHOW CREATE TABLE` を使用して、`ALTER TABLE` ステートメントが目的の効果を持つことを確認できます:

```
mysql> SHOW CREATE TABLE fish\G
***** 1. row *****
      Table: fish
Create Table: CREATE TABLE `fish` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(20) NOT NULL,
  `length_mm` int(11) NOT NULL,
  `weight_gm` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `uk` (`name`)
) ENGINE=ndbcluster DEFAULT CHARSET=latin1
COMMENT='NDB_TABLE=READ_BACKUP=1,FULLY_REPLICATED=1'
1 row in set (0.01 sec)
```

`FRAGMENT_COUNT_TYPE` は明示的に設定されなかったため、その値は `SHOW CREATE TABLE` によって出力されるコメントテキストには表示されません。ただし、`ndb_desc` では、この属性の更新された値が表示されます。`Table options` カラムには、有効になったばかりのバイナリプロパティが表示されます。これは、次に示す出力に表示されます (強調表示されたテキスト):

```
shell> ./ndb_desc -c localhost fish -d test -p
-- fish --
Version: 4
Fragment type: HashMapPartition
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: no
```

```

Number of attributes: 4
Number of primary keys: 1
Length of frm data: 380
Max Rows: 0
Row Checksum: 1
Row GCI: 1
SingleUserMode: 0
ForceVarPart: 1
PartitionCount: 1
FragmentCount: 1
FragmentCountType: ONE_PER_LDM_PER_NODE_GROUP
ExtraRowGciBits: 0
ExtraRowAuthorBits: 0
TableStatus: Retrieved
Table options: readbackup, fullyreplicated
HashMap: DEFAULT-HASHMAP-3840-1
-- Attributes --
id Int PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY AUTO_INCR
name Varchar(20;latin1_swedish_ci) NOT NULL AT=SHORT_VAR ST=MEMORY DYNAMIC
length_mm Int NOT NULL AT=FIXED ST=MEMORY DYNAMIC
weight_gm Int NOT NULL AT=FIXED ST=MEMORY DYNAMIC
-- Indexes --
PRIMARY KEY(id) - UniqueHashIndex
PRIMARY(id) - OrderedIndex
uk(name) - OrderedIndex
uk$unique(name) - UniqueHashIndex
-- Per partition info --
Partition      Row count      Commit count      Frag fixed memory      Frag varsized memory      Extent_space      Free extent_space

NDBT_ProgramExit: 0 - OK

```

これらのテーブルプロパティの詳細は、[セクション13.1.20.11「NDB_TABLE オプションの設定」](#)を参照してください。

[Extent_space](#) および [Free extent_space](#) カラムは、ディスクにカラムがある NDB テーブルにのみ関係しています。インメモリーカラムのみを持つテーブルの場合、これらのカラムには常に値 0 が表示されます。

これらの使用方法を説明するために、前の例を変更します。最初に、必要なディスクデータオブジェクトを次のように作成する必要があります。

```

CREATE LOGFILE GROUP lg_1
  ADD UNDOFILE 'undo_1.log'
  INITIAL_SIZE 16M
  UNDO_BUFFER_SIZE 2M
  ENGINE NDB;

ALTER LOGFILE GROUP lg_1
  ADD UNDOFILE 'undo_2.log'
  INITIAL_SIZE 12M
  ENGINE NDB;

CREATE TABLESPACE ts_1
  ADD DATAFILE 'data_1.dat'
  USE LOGFILE GROUP lg_1
  INITIAL_SIZE 32M
  ENGINE NDB;

ALTER TABLESPACE ts_1
  ADD DATAFILE 'data_2.dat'
  INITIAL_SIZE 48M
  ENGINE NDB;

```

(上記のステートメントおよびそれらによって作成されるオブジェクトについては、[セクション23.5.10.1「NDB Cluster ディスクデータオブジェクト」](#)、[セクション13.1.16「CREATE LOGFILE GROUP ステートメント」](#)、および[セクション13.1.21「CREATE TABLESPACE ステートメント」](#)を参照してください。)

カラムの 2 をディスクに格納するバージョンの fish テーブルを作成して移入できるようになりました (前のバージョンのテーブルが存在する場合はそれを最初に削除します)。

```
DROP TABLE IF EXISTS fish;
```

```
CREATE TABLE fish (
  id INT NOT NULL AUTO_INCREMENT,
  name VARCHAR(20) NOT NULL,
  length_mm INT NOT NULL,
  weight_gm INT NOT NULL,

  PRIMARY KEY pk (id),
  UNIQUE KEY uk (name)
) TABLESPACE ts_1 STORAGE DISK
ENGINE=NDB;

INSERT INTO fish VALUES
(NULL, 'guppy', 35, 2), (NULL, 'tuna', 2500, 150000),
(NULL, 'shark', 3000, 110000), (NULL, 'manta ray', 1500, 50000),
(NULL, 'grouper', 900, 125000), (NULL, 'puffer', 250, 2500);
```

`ndb_desc` をこのバージョンのテーブルに対して実行すると、次の出力が表示されます。

```
shell> ./ndb_desc -c localhost fish -d test -p
-- fish --
Version: 1
Fragment type: HashMapPartition
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: no
Number of attributes: 4
Number of primary keys: 1
Length of frm data: 1001
Max Rows: 0
Row Checksum: 1
Row GCI: 1
SingleUserMode: 0
ForceVarPart: 1
PartitionCount: 2
FragmentCount: 2
PartitionBalance: FOR_RP_BY_LDM
ExtraRowGciBits: 0
ExtraRowAuthorBits: 0
TableStatus: Retrieved
Table options: readbackup
HashMap: DEFAULT-HASHMAP-3840-2
Tablespace id: 16
Tablespace: ts_1
-- Attributes --
id Int PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY AUTO_INCR
name Varchar(80;utf8mb4_0900_ai_ci) NOT NULL AT=SHORT_VAR ST=MEMORY
length_mm Int NOT NULL AT=FIXED ST=DISK
weight_gm Int NOT NULL AT=FIXED ST=DISK
-- Indexes --
PRIMARY KEY(id) - UniqueHashIndex
PRIMARY(id) - OrderedIndex
uk(name) - OrderedIndex
uk$unique(name) - UniqueHashIndex
-- Per partition info --
Partition  Row count  Commit count  Frag fixed memory  Frag var sized memory  Extent_space  Free extent_space
0          2          2          32768             32768                 1048576       1044440
1          4          4          32768             32768                 1048576       1044400

NDBT_ProgramExit: 0 - OK
```

これは、テーブルスペースからこのテーブルの各パーティションに 1048576 バイトが割り当てられ、そのうちの 1044440 バイトが追加の格納のための空き領域であることを意味します。言い換えると、このテーブルのディスクベースのカラムのデータを格納するために、現在パーティションごとに $1048576 - 1044440 = 4136$ バイトが使用されています。Free extent space として表示されるバイト数は、fish テーブルのディスク上カラムデータを格納するためだけにのみ使用できます。このため、INFORMATION_SCHEMA.FILES テーブルから選択したときに表示されません。

NDB 8.0.21 以降の「ディスクデータ」テーブルでは、Tablespace id および Tablespace が表示されます。

完全にレプリケートされたテーブルの場合、ndb_desc ではプライマリパーティションフラグメントのレプリカを保持しているノードのみが表示され、コピーフラグメントレプリカ(のみ)を持つノードは無視されます。このような情

報は、mysql クライアントを使用して、ndbinfo データベースの table_distribution_status, table_fragments, table_info テーブルおよび table_replicas テーブルから取得できます。

次の表には、ndb_desc に固有のオプションが含まれています。追加説明が表のあとにあります。ほとんどの NDB Cluster プログラム (ndb_desc を含む) に共通のオプションについては、[セクション23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#) を参照してください。

表 23.31 プログラムで使用されるコマンドライン・オプション ndb_desc

形式	説明	追加、非推奨、または削除された
<code>--auto-inc,</code> <code>-a</code>	テーブルに次の値がある場合、AUTO_INCREMENT column の次の値を表示	追加: NDB 8.0.21
<code>--blob-info,</code> <code>-b</code>	BLOB テーブルのパーティション情報を出力に含めます。-p オプションも使用する必要があります	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--context,</code> <code>-x</code>	データベース、スキーマ、名前、内部 ID などのテーブルの追加情報を表示	追加: NDB 8.0.21
<code>--database=dbname,</code> <code>-d</code>	テーブルが含まれているデータベースの名前	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--extra-node-info,</code> <code>-n</code>	パーティションとデータノードのマッピングを出力に含めます。-p オプションも使用する必要があります	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--extra-partition-info,</code> <code>-p</code>	パーティションに関する情報を表示します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--retries=#,</code> <code>-r</code>	接続を再試行する回数 (1 秒おき)	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--table=tbl_name,</code> <code>-t</code>	インデックスを探すテーブルを指定します。このオプションを使用すると、-p および -n は効果がなくなり、無視されます	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--unqualified,</code> <code>-u</code>	修飾されていないテーブル名を使用します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)

- `--auto-inc, -a`

テーブルの AUTO_INCREMENT カラムに次の値がある場合は表示します。

- `--blob-info, -b`

従属する BLOB および TEXT カラムに関する情報を含めます。

このオプションを使用する場合は、`--extra-partition-info (-p)` オプションも使用する必要があります。

- `--context, -x`

スキーマ、データベース名、テーブル名、テーブルの内部 ID など、テーブルの追加のコンテキスト情報を表示します。

- `--database=db_name, -d`

テーブルが見つかるはずのデータベースを指定します。

- `--extra-node-info, -n`

テーブルパーティションおよびそれらが存在するデータノードのマッピングに関する情報を含めます。この情報は、配布認識メカニズムを検証し、NDB Cluster に格納されているデータへのより効率的なアプリケーションアクセスをサポートするのに役立ちます。

このオプションを使用する場合は、`--extra-partition-info (-p)` オプションも使用する必要があります。

- `--extra-partition-info, -p`

テーブルのパーティションに関する追加情報を出力します。

- `--retries=#, -r`

接続を停止する前に、この回数だけ接続を試みます。接続の試行は 1 秒おきに実行されます。

- `--table=tbl_name, -t`

インデックスを探すテーブルを指定します。

- `--unqualified, -u`

修飾されていないテーブル名を使用します。

出力にリストされているテーブルインデックスは、ID 順に並べられています。

23.4.10 ndb_drop_index — NDB テーブルからのインデックスの削除

`ndb_drop_index` は、指定されたインデックスを NDB テーブルから削除します。このユーティリティーは、NDB API アプリケーションを記述するための例としてのみ使用することをお勧めします。詳細は、このセクションで後述する「警告」を参照してください。

使用法

```
ndb_drop_index -c connection_string table_name index -d db_name
```

上記のステートメントは、`database` の `table` から `index` という名前のインデックスを削除します。

次の表には、`ndb_drop_index` に固有のオプションが含まれています。追加説明が表のあとにあります。ほとんどの NDB Cluster プログラム (`ndb_drop_index` を含む) に共通のオプションについては、[セクション23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#) を参照してください。

表 23.32 プログラムで使用されるコマンドライン・オプション `ndb_drop_index`

形式	説明	追加、非推奨、または削除された
<code>--database=dbname,</code> <code>-d</code>	テーブルが見つかったデータベースの名前	(MySQLに基づくすべてのNDBリリースでサポート 8.0)

警告

NDB API を使用して Cluster テーブルインデックスに操作を実行しても、MySQL は認識できず、MySQL サーバーはそのテーブルを使用できません。このプログラムを使用してインデックスを削除し、SQL ノードからテーブルにアクセスしようとすると、次のようなエラーが発生します。

```
shell> ./ndb_drop_index -c localhost dogs ix -d ctest1
Dropping index dogs/idx...OK

NDBT_ProgramExit: 0 - OK

shell> ./mysql -u jon -p ctest1
Enter password: *****
```



```
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
```

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7 to server version: 5.7.33-ndb-7.5.21
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

```
mysql> SHOW TABLES;
+-----+
| Tables_in_ctest1 |
+-----+
| a                |
| bt1              |
| bt2              |
| dogs             |
| employees        |
| fish             |
+-----+
6 rows in set (0.00 sec)

mysql> SELECT * FROM dogs;
ERROR 1296 (HY000): Got error 4243 'Index not found' from NDBCLUSTER
```

そのような場合、MySQL でテーブルをふたたび使用できるようにする唯一の方法は、テーブルを削除して再作成することです。テーブルを削除するには、SQL ステートメント `DROP TABLE` または `ndb_drop_table` ユーティリティ (セクション23.4.11「`ndb_drop_table` — NDB テーブルの削除」を参照してください) を使用できます。

23.4.11 ndb_drop_table — NDB テーブルの削除

`ndb_drop_table` は指定された NDB テーブルを削除します (NDB 以外のストレージエンジンで作成されたテーブルに対してこれを使用しようとすると、次のエラーで失敗します: 723: No such table exists。) この操作は非常に速く、NDB テーブルに対して MySQL の `DROP TABLE` ステートメントを使用するよりも格段に速いことがあります。

使用法

```
ndb_drop_table -c connection_string tbl_name -d db_name
```

次の表には、`ndb_drop_table` に固有のオプションが含まれています。追加説明が表のあとにあります。ほとんどの NDB Cluster プログラム (`ndb_drop_table` を含む) に共通のオプションについては、セクション23.4.32「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」を参照してください。

表 23.33 プログラムで使用されるコマンドライン・オプション `ndb_drop_table`

形式	説明	追加、非推奨、または削除された
<code>--database=dbname,</code> <code>-d</code>	テーブルが見つかったデータベースの名前	(MySQLに基づくすべてのNDBリリースでサポート 8.0)

NDB 8.0.17 より前は、このユーティリティを使用して削除された NDB テーブルは MySQL データディクショナリに保持されていましたが、`mysql` クライアントで `DROP TABLE` を使用して削除できませんでした。NDB 8.0.17 以降では、このような「孤立」テーブルは `DROP TABLE` を使用して削除できます。(Bug #29125206、Bug #93672)

23.4.12 ndb_error_reporter — NDB エラーレポートユーティリティ

`ndb_error_reporter` は、クラスタのバグまたはその他の問題の診断に使用できるデータノードおよび管理ノードログファイルからアーカイブを作成します。NDB Cluster のバグのレポートを提出するときは、このユーティリティを使用することを強くお勧めします。

次のテーブルに、NDB Cluster プログラム `ndb_error_reporter` に固有のコマンドオプションを示します。追加説明が表のあとにあります。ほとんどの NDB Cluster プログラム (`ndb_error_reporter` を含む) に共通のオプションについては、セクション23.4.32「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」を参照してください。

表 23.34 プログラムで使用されるコマンドライン・オプション ndb_error_reporter

形式	説明	追加、非推奨、または削除された
<code>--connection-timeout=timeout</code>	ノードへの接続時にタイムアウトするまで待機する秒数	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--dry-scp</code>	リモートホストで scp を無効にします。テストのためのみに使用します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--fs</code>	エラーレポートにファイルシステムデータを含めます。大量のディスク領域が使用されることがあります	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--skip-nodegroup=nodegroup_id</code>	この ID のノードグループ内のすべてのノードをスキップします	(MySQLに基づくすべてのNDBリリースでサポート 8.0)

使用法

```
ndb_error_reporter path/to/config-file [username] [options]
```

このユーティリティは管理ノードホストで使用することが意図されており、管理ホスト構成ファイル (通常、`config.ini` という名前です) へのパスが必要となります。必要に応じて、データノードログファイルをコピーするために、SSH を使用してクラスタのデータノードにアクセスできるユーザーの名前を指定できます。`ndb_error_reporter` は、実行されたディレクトリと同じディレクトリに作成されるアーカイブにそれらのすべてのファイルを含めます。アーカイブの名前は `ndb_error_report_YYYYMMDDhhmmss.tar.bz2` で、`YYYYMMDDhhmmss` は日時文字列です。

`ndb_error_reporter` は次に一覧されているオプションも受け入れます。

- `--connection-timeout=timeout`

コマンド行形式	<code>--connection-timeout=timeout</code>
型	Integer
デフォルト値	0

ノードに接続を試みるときに、タイムアウトする前にこの秒数だけ待機します。

- `--dry-scp`

コマンド行形式	<code>--dry-scp</code>
型	Boolean
デフォルト値	TRUE

リモートホストからの scp を使用せずに `ndb_error_reporter` を実行します。テストにのみ使用されます。

- `--fs`

コマンド行形式	<code>--fs</code>
型	Boolean
デフォルト値	FALSE

データノードファイルシステムを管理ホストにコピーし、それらをアーカイブに含めます。

データノードファイルシステムは圧縮されていても非常に大きいことがあるため、特にそうするように要請された場合を除き、このオプションを使用して作成したアーカイブをオラクルに送信しないでください。

- `--skip-nodegroup=nodegroup_id`

コマンド行形式	<code>--connection-timeout=timeout</code>
型	Integer

デフォルト値

0

指定したノードグループ ID を持つノードグループに属するすべてのノードをスキップします。

23.4.13 ndb_import — NDB への CSV データのインポート

`ndb_import` は NDB API を使用して、`mysqldump --tab` によって生成された CSV 形式のデータなどを NDB に直接インポートします。`ndb_import` が機能するには NDB 管理サーバー (`ndb_mgmd`) への接続が必要です。MySQL Server への接続は必要ありません。

使用法

```
ndb_import db_name file_name options
```

`ndb_import` には、2 つの引数が必要です。`db_name` は、データのインポート先のテーブルが見つかったデータベースの名前です。`file_name` は、データの読取り元の CSV ファイルの名前です。現在のディレクトリにない場合は、このファイルへのパスを含める必要があります。ファイルの名前はテーブルの名前と一致する必要があります。ファイル拡張子がある場合は考慮されません。`ndb_import` でサポートされているオプションには、フィールドセパレータ、エスケープおよび行終了記号を指定するオプションが含まれます。これらのオプションについては、このセクションの後半で説明します。`ndb_import` は NDB Cluster 管理サーバーに接続できる必要があります。このため、クラスタ `config.ini` ファイルに未使用の `[api]` スロットが存在する必要があります。

InnoDB などの別のストレージエンジンを使用する既存のテーブルを NDB テーブルとして複製するには、`mysql` クライアントを使用して `SELECT INTO OUTFILE` ステートメントを実行し、既存のテーブルを CSV ファイルにエクスポートしてから `CREATE TABLE LIKE` ステートメントを実行して、既存のテーブルと同じ構造を持つ新しいテーブルを作成し、新しいテーブルで `ALTER TABLE ... ENGINE=NDB` を実行します。その後、システムシェルから `ndb_import` を呼び出して、新しい NDB テーブルにデータをロードします。たとえば、`myinnodb` というデータベース内の `myinnodb_table` という名前の既存の InnoDB テーブルを、次に示すように `myndb` という名前のデータベース内の `myndb_table` という名前の NDB テーブルにエクスポートできます (適切な権限を持つ MySQL ユーザーとしてすでにログインしていることを前提としています)：

1. `mysql` クライアントで、次のようにします：

```
mysql> USE myinnodb;

mysql> SELECT * INTO OUTFILE '/tmp/myndb_table.csv'
> FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' ESCAPED BY '\\'
> LINES TERMINATED BY '\n'
> FROM myinnodbtable;

mysql> CREATE DATABASE myndb;

mysql> USE myndb;

mysql> CREATE TABLE myndb_table LIKE myinnodb.myinnodb_table;

mysql> ALTER TABLE myndb_table ENGINE=NDB;

mysql> EXIT;
Bye
shell>
```

ターゲットデータベースおよびテーブルが作成されると、実行中の `mysqld` は不要になります。必要に応じて、続ける前に `mysqldadmin shutdown` または別の方法を使用して停止できます。

2. システムシェルで、次の手順を実行します：

```
# if you are not already in the MySQL bin directory:
shell> cd path-to-mysql-bin-dir

shell> ndb_import myndb /tmp/myndb_table.csv --fields-optionally-enclosed-by="'" \
--fields-terminated-by="," --fields-escaped-by="\\'
```

出力は次のようになります：

```
job-1 import myndb.myndb_table from /tmp/myndb_table.csv
```

```

job-1 [running] import myndb.myndb_table from /tmp/myndb_table.csv
job-1 [success] import myndb.myndb_table from /tmp/myndb_table.csv
job-1 imported 19984 rows in 0h0m9s at 2277 rows/s
jobs summary: defined: 1 run: 1 with success: 1 with failure: 0
shell>

```

次のテーブルに、`ndb_import` に固有のオプションを示します。追加説明が表のあとにあります。ほとんどの NDB Cluster プログラム (`ndb_import` を含む) に共通のオプションについては、[セクション 23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#) を参照してください。

表 23.35 プログラムで使用されるコマンドライン・オプション `ndb_import`

形式	説明	追加、非推奨、または削除された
<code>--abort-on-error</code>	致命的なエラー時にコアをダンプします。デバッグに使用されます	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--ai-increment=#</code>	PK が非表示のテーブルの場合は、自動増分を指定します。mysqlqld を参照	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--ai-offset=#</code>	PK が非表示のテーブルの場合は、自動増分オフセットを指定します。mysqlqld を参照	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--ai-prefetch-sz=#</code>	PK が非表示のテーブルの場合は、プリフェッチされる自動インクリメント値の数を指定します。mysqlqld を参照	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--connections=#</code>	作成するクラスタ接続の数	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--continue</code>	ジョブが失敗した場合は、次のジョブに進みます	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--db-workers=#</code>	データベース操作を実行しているスレッド数 (データノード当たり)	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--errins-type=name</code>	テスト用の挿入タイプでエラーが発生しました。使用可能なすべての値を取得するには "list" を使用	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--errins-delay=#</code>	エラー挿入遅延 (ミリ秒)。ランダムな変動が追加されます	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--fields-enclosed-by=char</code>	LOAD DATA ステートメントの「FIELDS ENCLOSED BY」オプションと同じです。CSV 入力の場合、これは <code>--fields-optionally-enclosed-by</code> の使用と同じです	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--fields-escaped-by=name</code>	LOAD DATA ステートメントの「FIELDS ESCAPED BY」オプションと同じ	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--fields-optionally-enclosed-by=char</code>	LOAD DATA ステートメントの「FIELDS OPTIONALLY ENCLOSED BY」オプションと同じ	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--fields-terminated-by=char</code>	LOAD DATA ステートメントの「FIELDS TERMINATED BY」オプションと同じ	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--idlesleep=#</code>	これ以上の処理を待機するスリープ時間 (ミリ秒)	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--idlespin=#</code>	アイドル状態になる前に再試行する回数	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--ignore-lines=#</code>	入力ファイルの最初の # 行を無視します。データ以外のヘッダーをスキップするために使用	(MySQLに基づくすべてのNDBリリースでサポート 8.0)

形式	説明	追加、非推奨、または削除された
<code>--input-type=name</code>	入力タイプ: random または csv	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--input-workers=#</code>	入力を処理するスレッドの数。 --input-type が csv の場合は 2 以上である必要があります	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--keep-state</code>	状態ファイル (空でない *.rej ファイルを除く) は、通常、ジョブの完了時に削除されます。このオプションを使用すると、かわりにすべての状態ファイルが保持されます	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--lines-terminated-by=name</code>	LOAD DATA ステートメントの「LINES TERMINATED BY」オプションと同じ	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--max-rows=#</code>	この数の入力データ行のみをインポートします。デフォルトは 0 で、すべての行がインポートされます	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--monitor=#</code>	何かに変更された場合 (ステータス、拒否された行、一時エラー)、実行中のジョブのステータスを定期的に印刷します。値 0 は無効になります。値 1 は、表示された変更をすべて出力します。値を大きくすると、事前定義された制限までステータス出力が指数関数的に減少	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--no-asynch</code>	単一トランザクションでのバッチとしてのデータベース操作の実行	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--no-hint</code>	分散キーヒントを使用してデータノード (TC) を選択しない	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--opbatch=#</code>	db 実行バッチは、NDB カーネルに送信される一連のトランザクションおよび操作です。このオプションは、db 実行バッチ内の NDB 操作 (blob 操作を含む) を制限します。したがって、非同期トランザクションの数も制限されます。値 0 は無効です	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--opbytes=#</code>	実行バッチのバイト数の制限 (デフォルトは 0 = 制限なし)	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--output-type=name</code>	出力タイプ: ndb はデフォルトで、null がテストに使用されます	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--output-workers=#</code>	出力またはリレーデータベース操作を処理するスレッドの数	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--pagesize=#</code>	I/O バッファを指定されたサイズに整理	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--pagecnt=#</code>	ページサイズの倍数としての I/O バッファのサイズ。CSV 入力ワーカーがダブルサイズのバッファを割り当てます	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--polltimeout=#</code>	完了した非同期トランザクションのポーリング当たりのタイムアウト。ポーリングは、すべてのポーリングが完了するか、エラーが発生するまで続行されます	(MySQLに基づくすべてのNDBリリースでサポート 8.0)

形式	説明	追加、非推奨、または削除された
<code>--rejects=#</code>	データロードで拒否される行 (永続エラーのある行) の数を制限します。デフォルトは 0 で、拒否された行によって致命的エラーが発生することを意味します。制限を超える行も *.rej に追加されます	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--resume</code>	ジョブが中断された場合 (一時エラー、ユーザー割り込み)、まだ処理されていない行で再開	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--rowbatch=#</code>	行キューの行を制限します (デフォルトは 0 = 無制限)。--input-type がランダムの場合は 1 以上にする必要があります	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--rowbytes=#</code>	行キュー内のバイト数の制限 (0 = 制限なし)	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--state-dir=name</code>	状態ファイルを書き込む場所。デフォルトは current ディレクトリです	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--stats</code>	パフォーマンス関連のオプションおよび内部統計を *.sto および *.stt ファイルに保存します。これらのファイルは、--keep-state が使用されていない場合でも正常終了時に保持されます	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--tempdelay=#</code>	一時エラー間でスリープするミリ秒数	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--temperrors=#</code>	実行バッチごとに、一時エラーが原因でトランザクションが失敗できる回数。0 は、一時エラーが致命的であることを意味します。このようなエラーによって .rej ファイルに行が書き込まれることはありません	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--verbose,</code> <code>-v</code>	詳細出力の有効化	(MySQLに基づくすべてのNDBリリースでサポート 8.0)

- `--abort-on-error`

コマンド行形式	<code>--abort-on-error</code>
型	Boolean
デフォルト値	FALSE

致命的エラー時にコアをダンプします。デバッグにのみ使用されます。

- `--ai-increment=#`

コマンド行形式	<code>--ai-increment=#</code>
型	Integer
デフォルト値	1
最小値	1
最大値	4294967295

非表示の主キーを持つテーブルの場合は、`auto_increment_increment` システム変数が MySQL Server で行うように、自動増分を指定します。

- `--ai-offset=#`

コマンド行形式	<code>--ai-offset=#</code>
型	Integer
デフォルト値	1
最小値	1
最大値	4294967295

非表示の主キーを持つテーブルの場合は、自動増分オフセットを指定します。 `auto_increment_offset` システム変数に似ています。

- `--ai-prefetch-sz=#`

コマンド行形式	<code>--ai-prefetch-sz=#</code>
型	Integer
デフォルト値	1024
最小値	1
最大値	4294967295

非表示の主キーを持つテーブルの場合は、プリフェッチされる自動増分値の数を指定します。 MySQL Server での `ndb_autoincrement_prefetch_sz` システム変数と同様に動作します。

- `--connections=#`

コマンド行形式	<code>--connections=#</code>
型	Integer
デフォルト値	1
最小値	1
最大値	4294967295

作成するクラスタ接続の数。

- `--continue`

コマンド行形式	<code>--continue</code>
型	Boolean
デフォルト値	FALSE

ジョブが失敗した場合は、次のジョブに進みます。

- `--db-workers=#`

コマンド行形式	<code>--db-workers=#</code>
型	Integer
デフォルト値	4
最小値	1
最大値	4294967295

データベース操作を実行しているスレッド数 (データノードあたり)。

- `--errins-type=name`

コマンド行形式	<code>--errins-type=name</code>
---------	---------------------------------

型	列挙
デフォルト値	[none]
有効な値	stopjob stopall sighup sigint list

挿入タイプでエラーが発生しました。使用可能なすべての値を取得するには、`name` 値として `list` を使用します。このオプションは、テスト目的でのみ使用されます。

- `--errins-delay=#`

コマンド行形式	<code>--errins-delay=#</code>
型	Integer
デフォルト値	1000
最小値	0
最大値	4294967295
単位	ms

エラー挿入遅延 (ミリ秒)。ランダムなバリエーションが追加されます。このオプションは、テスト目的でのみ使用されます。

- `--fields-enclosed-by=char`

コマンド行形式	<code>--fields-enclosed-by=char</code>
型	文字列
デフォルト値	[none]

これは、`LOAD DATA` ステートメントの `FIELDS ENCLOSED BY` オプションと同様に機能し、フィールド値を引用符で囲む文字を指定します。CSV 入力の場合、これは `--fields-optionally-enclosed-by` と同じです。

- `--fields-escaped-by=name`

コマンド行形式	<code>--fields-escaped-by=name</code>
型	文字列
デフォルト値	\

SQL `LOAD DATA` ステートメントの `FIELDS ESCAPED BY` オプションと同じ方法でエスケープ文字を指定します。

- `--fields-optionally-enclosed-by=char`

コマンド行形式	<code>--fields-optionally-enclosed-by=char</code>
型	文字列
デフォルト値	[none]

これは、`LOAD DATA` ステートメントの `FIELDS OPTIONALLY ENCLOSED BY` オプションと同様に機能し、オプションでフィールド値を引用符で囲んだ文字を指定します。CSV 入力の場合、これは `--fields-enclosed-by` と同じです。

- `--fields-terminated-by=char`

コマンド行形式	<code>--fields-terminated-by=char</code>
型	文字列
デフォルト値	<code>\t</code>

これは、`LOAD DATA` ステートメントの `FIELDS TERMINATED BY` オプションと同様に機能し、インターペットする文字をフィールドセパレータとして指定します。

- `--idlesleep=#`

コマンド行形式	<code>--idlesleep=#</code>
型	Integer
デフォルト値	1
最小値	1
最大値	4294967295
単位	ms

これ以上の作業の実行を待機するスリープ時間 (ミリ秒)。

- `--idlespin=#`

コマンド行形式	<code>--idlespin=#</code>
型	Integer
デフォルト値	0
最小値	0
最大値	4294967295

スリープする前に再試行する回数。

- `--ignore-lines=#`

コマンド行形式	<code>--ignore-lines=#</code>
型	Integer
デフォルト値	0
最小値	0
最大値	4294967295

`ndb_import` が入力ファイルの最初の `#` 行を無視するようにします。これは、データを含まないファイルヘッダーをスキップするために使用できます。

- `--input-type=name`

コマンド行形式	<code>--input-type=name</code>
型	列挙
デフォルト値	<code>csv</code>
有効な値	<code>random</code> <code>csv</code>

入力タイプのタイプを設定します。デフォルトは `csv` です。`random` はテストのみを目的としています。

- `--input-workers=#`

コマンド行形式	<code>--input-workers=#</code>
型	Integer
デフォルト値	4
最小値	1
最大値	4294967295

入力を処理するスレッドの数を設定します。

- `--keep-state`

コマンド行形式	<code>--keep-state</code>
型	Boolean
デフォルト値	false

デフォルトでは、`ndb_import` はジョブの完了時にすべての状態ファイル (空でない `*.rej` ファイルを除く) を削除します。プログラムですべての状態ファイルを強制的に保持するには、このオプション (または引数は必須) を指定します。

- `--lines-terminated-by=name`

コマンド行形式	<code>--lines-terminated-by=name</code>
型	文字列
デフォルト値	<code>\n</code>

これは、`LOAD DATA` ステートメントの `LINES TERMINATED BY` オプションと同様に機能し、インターペットする文字を行末として指定します。

- `--log-level=#`

コマンド行形式	<code>--log-level=#</code>
型	Integer
デフォルト値	0
最小値	0
最大値	2

指定されたレベルで内部ロギングを実行します。このオプションは、主に内部および開発での使用を目的としています。

NDB のデバッグビルドでのみ、このオプションを使用してロギングレベルを最大 4 に設定できます。

- `--max-rows=#`

コマンド行形式	<code>--max-rows=#</code>
型	Integer
デフォルト値	0
最小値	0
最大値	4294967295
単位	bytes

この数の入力データ行のみをインポートします。デフォルトは 0 で、すべての行がインポートされます。

- `--monitor=#`

コマンド行形式	<code>--monitor=#</code>
型	Integer
デフォルト値	2
最小値	0
最大値	4294967295
単位	bytes

なんらかの変更 (ステータス、拒否された行、一時エラー) があった場合は、実行中のジョブのステータスを定期的に出力します。このレポートを無効にするには、0 に設定します。1 に設定すると、表示される変更が印刷されません。値を大きくすると、このステータスレポートの頻度が低くなります。

- `--no-asynch`

コマンド行形式	<code>--no-asynch</code>
型	Boolean
デフォルト値	FALSE

単一のトランザクションでバッチとしてデータベース操作を実行します。

- `--no-hint`

コマンド行形式	<code>--no-hint</code>
型	Boolean
デフォルト値	FALSE

分散キーのヒントを使用してデータノードを選択しないでください。

- `--opbatch=#`

コマンド行形式	<code>--opbatch=#</code>
型	Integer
デフォルト値	256
最小値	1
最大値	4294967295
単位	bytes

実行バッチごとの操作数 (blob 操作を含む) および非同期トランザクション数の制限を設定します。

- `--opbytes=#`

コマンド行形式	<code>--opbytes=#</code>
型	Integer
デフォルト値	0
最小値	0
最大値	4294967295
単位	bytes

実行バッチ当たりのバイト数の制限を設定します。無制限の場合は 0 を使用します。

- `--output-type=name`

コマンド行形式	<code>--output-type=name</code>
---------	---------------------------------

型	列挙
デフォルト値	ndb
有効な値	null

出力タイプを設定します。ndb がデフォルトです。null はテストにのみ使用されます。

- `--output-workers=#`

コマンド行形式	<code>--output-workers=#</code>
型	Integer
デフォルト値	2
最小値	1
最大値	4294967295

出力またはリレーデータベース操作を処理するスレッドの数を設定します。

- `--pagesize=#`

コマンド行形式	<code>--pagesize=#</code>
型	Integer
デフォルト値	4096
最小値	1
最大値	4294967295
単位	bytes

I/O バッファを指定されたサイズに位置合せします。

- `--pagecnt=#`

コマンド行形式	<code>--pagecnt=#</code>
型	Integer
デフォルト値	64
最小値	1
最大値	4294967295

I/O バッファのサイズをページサイズの倍数として設定します。CSV 入力ワーカーは、サイズが増えたバッファを割り当てます。

- `--polltimeout=#`

コマンド行形式	<code>--polltimeout=#</code>
型	Integer
デフォルト値	1000
最小値	1
最大値	4294967295
単位	ms

完了した非同期トランザクションのポーリングごとにタイムアウトを設定します。ポーリングは、すべてのポーリングが完了するか、エラーが発生するまで続行されます。

- `--rejects=#`

コマンド行形式	<code>--rejects=#</code>
型	Integer
デフォルト値	0
最小値	0
最大値	4294967295

データロードで拒否される行 (永続エラーのある行) の数を制限します。デフォルトは 0 で、拒否された行によって致命的エラーが発生することを意味します。制限を超える原因となった行は、`.rej` ファイルに追加されます。

このオプションによって課される制限は、現在の実行中に有効です。`--resume` を使用して再起動された実行は、この目的では「new」の実行とみなされます。

- `--resume`

コマンド行形式	<code>--resume</code>
型	Boolean
デフォルト値	FALSE

ジョブが中断された場合 (一時 DB エラーまたはユーザーによる割込みが原因)、まだ処理されていない行で再開します。

- `--rowbatch=#`

コマンド行形式	<code>--rowbatch=#</code>
型	Integer
デフォルト値	0
最小値	0
最大値	4294967295
単位	rows

行キューごとの行数の制限を設定します。無制限の場合は 0 を使用します。

- `--rowbytes=#`

コマンド行形式	<code>--rowbytes=#</code>
型	Integer
デフォルト値	262144
最小値	0
最大値	4294967295
単位	bytes

行キュー当たりのバイト数の制限を設定します。無制限の場合は 0 を使用します。

- `--stats`

コマンド行形式	<code>--stats</code>
型	Boolean

デフォルト値	false
--------	-------

パフォーマンスおよびその他の内部統計に関連するオプションに関する情報を、*.sto および *.stt という名前のファイルに保存します。これらのファイルは、(--keep-state も指定されていない場合でも) 正常終了時に常に保持されます。

- --state-dir=name

コマンド行形式	--state-dir=name
型	文字列
デフォルト値	.

プログラムの実行によって生成された状態ファイル (tbl_name.map、tbl_name.rej、tbl_name.res および tbl_name.stt) を書き込む場所。デフォルトは現在のディレクトリです。

- --tempdelay=#

コマンド行形式	--tempdelay=#
型	Integer
デフォルト値	10
最小値	0
最大値	4294967295
単位	ms

一時エラー間でスリープするミリ秒数。

- --temperrors=#

コマンド行形式	--temperrors=#
型	Integer
デフォルト値	0
最小値	0
最大値	4294967295

実行バッチごとに、一時エラーが原因でトランザクションが失敗する可能性がある回数。デフォルトは 0 で、これは一時エラーが致命的であることを意味します。一時エラーにより、.rej ファイルに行が追加されることはありません。

- --verbose, -v

コマンド行形式	--verbose
型	Boolean
デフォルト値	false

冗長出力を有効にします。

LOAD DATA と同様に、フィールドおよび行の書式設定のオプションは、CSV ファイルの作成に使用されたオプションと、SELECT INTO ... OUTFILE を使用して実行されたか他の方法で実行されたかにかかわらず、ほとんど一致します。LOAD DATA ステートメントの STARTING WITH オプションと同等のものはありません。

ndb_import は NDB 7.6.2 で追加されました。

`ndb_index_stat` は、NDB テーブルのインデックスに関するフラグメントごとの統計情報を表示します。これには、キャッシュバージョンと経過期間、パーティションごとのインデックスエントリの数、およびインデックスによるメモリー使用量が含まれます。

使用法

指定した NDB テーブルの基本的なインデックス統計を取得するには、最初の引数としてテーブル名を指定し、`--database (-d)` オプションを使用してこのテーブルが含まれているデータベース名をその直後に指定して、`ndb_index_stat` を次のように呼び出します。

```
ndb_index_stat table -d database
```

この例では、`ndb_index_stat` を使用して、`test` データベースの `mytable` という名前の NDB テーブルに関するそのような情報を取得しています。

```
shell> ndb_index_stat -d test mytable
table:City index:PRIMARY fragCount:2
sampleVersion:3 loadTime:1399585986 sampleCount:1994 keyBytes:7976
query cache: valid:1 sampleCount:1994 totalBytes:27916
times in ms: save: 7.133 sort: 1.974 sort per sample: 0.000

NDBT_ProgramExit: 0 - OK
```

`sampleVersion` は、統計データが取得されたキャッシュのバージョン番号です。`--update` オプションを指定して `ndb_index_stat` を実行すると、`sampleVersion` が増分されます。

`loadTime` はキャッシュが最後に更新された時間を示しています。これは UNIX エポックからの秒数として表されません。

`sampleCount` はパーティションごとに見つかったインデックスエントリの数です。エントリの合計数を見積もるには、これをフラグメントの数 (`fragCount` として表示されます) で乗算します。

`sampleCount` は `SHOW INDEX` または `INFORMATION_SCHEMA.STATISTICS` のカーディナリティーと似ています。ただし、後者の 2 つはテーブル全体の統計を示し、`ndb_index_stat` はフラグメントごとの平均を示します。

`keyBytes` はインデックスによって使用されるバイト数です。この例では、主キーは整数であり、各インデックスに 4 バイトが必要となるため、`keyBytes` はこの場合次のように計算できます。

```
keyBytes = sampleCount * (4 bytes per index) = 1994 * 4 = 7976
```

この情報は、`INFORMATION_SCHEMA.COLUMNS` の対応するカラム定義を使用して取得することもできます (これには、MySQL Server および MySQL クライアントアプリケーションが必要となります)。

`totalBytes` はテーブルのすべてのインデックスで使用される合計メモリーです (バイト単位)。

前述の例に示されている時間は、`ndb_index_stat` の各呼び出しに固有のものです。

`--verbose` オプションを指定すると、次のように追加出力が表示されます。

```
shell> ndb_index_stat -d test mytable --verbose
random seed 1337010518
connected
loop 1 of 1
table:mytable index:PRIMARY fragCount:4
sampleVersion:2 loadTime:1336751773 sampleCount:0 keyBytes:0
read stats
query cache created
query cache: valid:1 sampleCount:0 totalBytes:0
times in ms: save: 20.766 sort: 0.001
disconnected

NDBT_ProgramExit: 0 - OK

shell>
```

プログラムからの出力が `NDBT_ProgramExit: 0 - OK` のみの場合は、統計がまだ存在しないことを示している可能性があります。これらを強制的に作成 (またはすでに存在する場合は更新) するには、`--update` オプションを指定して `ndb_index_stat` を起動するか、`mysql` クライアントのテーブルで `ANALYZE TABLE` を実行します。

オプション

次のテーブルに、NDB Cluster `ndb_index_stat` ユーティリティに固有のオプションを示します。詳しい説明は表のあとに一覧されています。ほとんどの NDB Cluster プログラム (`ndb_index_stat` を含む) に共通のオプションについては、[セクション23.4.32「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#)を参照してください。

表 23.36 プログラムで使用されるコマンドライン・オプション `ndb_index_stat`

形式	説明	追加、非推奨、または削除された
<code>--database=name,</code> <code>-d</code>	テーブルが含まれているデータベースの名前	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--delete</code>	テーブルのインデックス統計を削除し、以前に構成された自動更新を停止	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--update</code>	テーブルのインデックス統計を更新し、以前に構成された自動更新を再起動	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--dump</code>	クエリーキャッシュの印刷	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--query=#</code>	最初のキー属性に対してランダム範囲クエリーを実行します (int unsigned である必要があります)	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--sys-drop</code>	NDB カーネルの統計テーブルおよびイベントを削除します (すべての統計が失われます)	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--sys-create</code>	NDB カーネルにすべての統計テーブルおよびイベントを作成します (それらがまったく存在していなかった場合)	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--sys-create-if-not-exist</code>	NDB カーネルにあらかじめ存在しない統計テーブルおよびイベントを作成します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--sys-create-if-not-valid</code>	NDB カーネルにまだ存在しない統計テーブルまたはイベントを、無効なものを削除したあとに作成	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--sys-check</code>	NDB システムのインデックス統計およびイベントテーブルが存在することを確認します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--sys-skip-tables</code>	sys-* オプションをテーブルに適用しません	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--sys-skip-events</code>	sys-* オプションをイベントに適用しません	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--verbose,</code> <code>-v</code>	冗長出力を有効にします	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--loops=#</code>	指定したコマンドを実行する回数を設定します。デフォルトは 0 です	(MySQLに基づくすべてのNDBリリースでサポート 8.0)

`ndb_index_stat` の統計オプション。 次のオプションはインデックス統計を生成するために使用します。これらは指定されたテーブルおよびデータベースを処理します。これらはシステムオプション (`ndb_index_stat` のシステムオプションを参照してください) と混在させることはできません。

- `--database=name, -d name`

コマンド行形式	<code>--database=name</code>
型	文字列
デフォルト値	<code>[none]</code>
最小値	
最大値	

問い合わせるテーブルが含まれているデータベースの名前。

- `--delete`

コマンド行形式	<code>--delete</code>
型	Boolean
デフォルト値	<code>false</code>
最小値	
最大値	

指定したテーブルのインデックス統計を削除し、以前構成された自動更新を停止します。

- `--update`

コマンド行形式	<code>--update</code>
型	Boolean
デフォルト値	<code>false</code>
最小値	
最大値	

指定したテーブルのインデックス統計を更新し、以前構成された自動更新を再開します。

- `--dump`

コマンド行形式	<code>--dump</code>
型	Boolean
デフォルト値	<code>false</code>
最小値	
最大値	

クエリーキャッシュの内容をダンプします。

- `--query=#`

コマンド行形式	<code>--query=#</code>
型	数値
デフォルト値	<code>0</code>
最小値	<code>0</code>
最大値	<code>MAX_INT</code>

最初のキー属性 (符号なしの int である必要があります) に対してランダム範囲クエリーを実行します。

ndb_index_stat のシステムオプション。 次のオプションは、NDB カーネルの統計テーブルを生成および更新するために使用します。これらのオプションは、統計オプションと混在させることはできません (ndb_index_stat の統計オプションを参照してください)。

- `--sys-drop`

コマンド行形式	<code>--sys-drop</code>
型	Boolean
デフォルト値	<code>false</code>
最小値	
最大値	

NDB カーネルのすべての統計テーブルおよびイベントを削除します。これを実行すると、すべての統計が失われます。

- `--sys-create`

コマンド行形式	<code>--sys-create</code>
型	Boolean
デフォルト値	<code>false</code>
最小値	
最大値	

NDB カーネルにすべての統計テーブルおよびイベントを作成します。これはそれらがあらかじめ存在していなかった場合にのみ動作します。

- `sys-create-if-not-exist`

コマンド行形式	<code>--sys-create-if-not-exist</code>
型	Boolean
デフォルト値	<code>false</code>
最小値	
最大値	

このプログラムが呼び出されたときにあらかじめ存在していなかった NDB システム統計テーブルまたはイベント (あるいはその両方) を作成します。

- `--sys-create-if-not-valid`

コマンド行形式	<code>--sys-create-if-not-valid</code>
型	Boolean
デフォルト値	<code>false</code>
最小値	
最大値	

無効なものを削除したあとにあらかじめ存在していなかった NDB システムの統計テーブルまたはイベントを作成します。

- `--sys-check`

コマンド行形式	<code>--sys-check</code>
型	Boolean
デフォルト値	<code>false</code>
最小値	

最大値	
-----	--

必要なすべてのシステム統計テーブルおよびイベントが NDB カーネルに存在することを検証します。

- `--sys-skip-tables`

コマンド行形式	<code>--sys-skip-tables</code>
型	Boolean
デフォルト値	false
最小値	
最大値	

`--sys-*` オプションを統計テーブルに適用しません。

- `--sys-skip-events`

コマンド行形式	<code>--sys-skip-events</code>
型	Boolean
デフォルト値	false
最小値	
最大値	

`--sys-*` オプションをイベントに適用しません。

- `--verbose`

コマンド行形式	<code>--verbose</code>
型	Boolean
デフォルト値	false
最小値	
最大値	

冗長出力を有効にします。

- `--loops=#`

コマンド行形式	<code>--loops=#</code>
型	数値
デフォルト値	0
最小値	0
最大値	MAX_INT

コマンドをこの回数繰り返します (テストで使用するため)。

23.4.15 ndb_move_data — NDB データコピーユーティリティー

`ndb_move_data` は、NDB テーブル間でデータをコピーします。

使用法

プログラムは、ソーステーブルとターゲットテーブルの名前を使用して起動されます。これらのいずれかまたは両方を、オプションでデータベース名で修飾できます。どちらのテーブルも NDB ストレージエンジンを使用する必要があります。

ndb_move_data options source target

次のテーブルに、`ndb_move_data` に固有のオプションを示します。追加説明が表のあとにあります。ほとんどの NDB Cluster プログラム (`ndb_move_data` を含む) に共通のオプションについては、[セクション 23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#) を参照してください。

表 23.37 プログラムで使用されるコマンドライン・オプション `ndb_move_data`

形式	説明	追加、非推奨、または削除された
<code>--abort-on-error</code>	永続エラー時にコアをダンプ (デバッグオプション)	(MySQL に基づくすべての NDB リリースでサポート 8.0)
<code>--character-sets-dir=name</code>	文字セットが存在するディレクトリ	(MySQL に基づくすべての NDB リリースでサポート 8.0)
<code>--database=dbname,</code> <code>-d</code>	テーブルが見つかったデータベースの名前	(MySQL に基づくすべての NDB リリースでサポート 8.0)
<code>--drop-source</code>	すべての行の移動後にソーステーブルを削除	(MySQL に基づくすべての NDB リリースでサポート 8.0)
<code>--error-insert</code>	ランダムな一時エラーの挿入 (テストオプション)	(MySQL に基づくすべての NDB リリースでサポート 8.0)
<code>--exclude-missing-columns</code>	ソーステーブルまたはターゲットテーブルの余分なカラムを無視	(MySQL に基づくすべての NDB リリースでサポート 8.0)
<code>--lossy-conversions,</code> <code>-l</code>	小さい型への変換時に属性データの切捨てを許可	(MySQL に基づくすべての NDB リリースでサポート 8.0)
<code>--promote-attributes,</code> <code>-A</code>	属性データの大きい型への変換を許可	(MySQL に基づくすべての NDB リリースでサポート 8.0)
<code>--staging-tries=x[,y[,z]]</code>	一時エラー時の試行回数を指定します。形式は <code>x[,y[,z]]</code> です。ここで <code>x</code> は最大試行回数 (0 = 制限なし)、 <code>y</code> は最小遅延 (ミリ秒)、 <code>z</code> = 最大遅延 (ミリ秒) です	(MySQL に基づくすべての NDB リリースでサポート 8.0)
<code>--verbose</code>	詳細メッセージの有効化	(MySQL に基づくすべての NDB リリースでサポート 8.0)

- `--abort-on-error`

コマンド行形式	<code>--abort-on-error</code>
型	Boolean
デフォルト値	FALSE

永続エラー時にコアをダンプします (デバッグオプション)。

- `--character-sets-dir=name`

コマンド行形式	<code>--character-sets-dir=name</code>
型	文字列
デフォルト値	[none]

文字セットが存在するディレクトリ。

- `--database=dbname, -d`

コマンド行形式	<code>--database=dbname</code>
型	文字列

デフォルト値	TEST_DB
--------	---------

テーブルが見つかるデータベースの名前。

- `--drop-source`

コマンド行形式	<code>--drop-source</code>
型	Boolean
デフォルト値	FALSE

すべての行が移動されたら、ソーステーブルを削除します。

- `--error-insert`

コマンド行形式	<code>--error-insert</code>
型	Boolean
デフォルト値	FALSE

ランダムな一時エラーを挿入します (テストオプション)。

- `--exclude-missing-columns`

コマンド行形式	<code>--exclude-missing-columns</code>
型	Boolean
デフォルト値	FALSE

ソーステーブルまたはターゲットテーブルの余分なカラムを無視します。

- `--lossy-conversions, -l`

コマンド行形式	<code>--lossy-conversions</code>
型	Boolean
デフォルト値	FALSE

属性データを小さい型に変換するときに切り捨てられるようにします。

- `--promote-attributes, -A`

コマンド行形式	<code>--promote-attributes</code>
型	Boolean
デフォルト値	FALSE

属性データを大きい型に変換できるようにします。

- `--staging-tries=x[,y[,z]]`

コマンド行形式	<code>--staging-tries=x[,y[,z]]</code>
型	文字列
デフォルト値	0,1000,60000

一時エラー時の試行回数を指定します。形式は `x[,y[,z]]` で、`x` は最大試行回数 (0 = 制限なし)、`y` は最小遅延 (ミリ秒)、`z` は最大遅延 (ミリ秒) です。

- `--verbose`

コマンド行形式	<code>--verbose</code>
---------	------------------------

型	Boolean
デフォルト値	FALSE

詳細メッセージを有効にします。

23.4.16 ndb_perror — NDB エラーメッセージ情報の取得

`ndb_perror` は、エラーコードが指定された NDB エラーに関する情報を表示します。これには、エラーメッセージ、エラーのタイプ、およびエラーが永続的か一時的かが含まれます。NDB 7.6.4 の MySQL NDB Cluster ディストリビューションに追加されました。これは、`peror --ndb` のドロップインの代替として使用されます。

使用法

```
ndb_perror [options] error_code
```

`ndb_perror` は、実行中の NDB Cluster またはノード (SQL ノードを含む) にアクセスする必要はありません。指定された NDB エラーに関する情報を表示するには、次のようにエラーコードを引数として使用してプログラムを呼び出します:

```
shell> ndb_perror 323
NDB error code 323: Invalid nodegroup id, nodegroup already existing: Permanent error: Application error
```

エラーメッセージのみを表示するには、次に示すように、`--silent` オプション (`-s` の短縮形) を指定して `ndb_perror` を起動します:

```
shell> ndb_perror -s 323
Invalid nodegroup id, nodegroup already existing: Permanent error: Application error
```

`peror` と同様に、`ndb_perror` は複数のエラーコードを受け入れます:

```
shell> ndb_perror 321 1001
NDB error code 321: Invalid nodegroup id: Permanent error: Application error
NDB error code 1001: Illegal connect string
```

`ndb_perror` の追加のプログラムオプションについては、このセクションの後半で説明します。

`ndb_perror` は、NDB 7.6.4 の時点で非推奨であり、MySQL NDB Cluster の将来のリリースで削除される予定の `peror --ndb` に置き換わります。NDB エラー情報を取得するために `peror` に依存する可能性のあるスクリプトやその他のアプリケーションで置換を容易にするために、`ndb_perror` は独自の「「ダミー」」 `--ndb` オプションをサポートしていますが、これは何も行いません。

次のテーブルに、NDB Cluster プログラム `ndb_perror` に固有のすべてのオプションを示します。追加説明が表のあとにあります。

表 23.38 プログラムで使用されるコマンドライン・オプション `ndb_perror`

形式	説明	追加、非推奨、または削除された
<code>--help</code> , <code>-?</code>	ヘルプテキストの表示	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--ndb</code>	古いバージョンの <code>peror</code> に依存するアプリケーションとの互換性のために、何も行いません	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--silent</code> , <code>-s</code>	エラーメッセージのみ表示	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--version</code> , <code>-V</code>	プログラムのバージョン情報を出力して終了	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--verbose</code> , <code>-v</code>	冗長出力。 <code>--silent</code> で無効にします	(MySQLに基づくすべてのNDBリリースでサポート 8.0)

その他のオプション

- `--help, -?`

コマンド行形式	<code>--help</code>
型	Boolean
デフォルト値	TRUE

プログラムのヘルプテキストを表示して終了します。

- `--ndb`

コマンド行形式	<code>--ndb</code>
型	Boolean
デフォルト値	TRUE

そのプログラムの `--ndb` オプションを使用する古いバージョンの `peror` に依存するアプリケーションとの互換性のため、`ndb_peror` で使用する場合、このオプションは何もせず、無視されます。

- `--silent, -s`

コマンド行形式	<code>--silent</code>
型	Boolean
デフォルト値	TRUE

エラーメッセージのみ表示します。

- `--version, -V`

コマンド行形式	<code>--version</code>
型	Boolean
デフォルト値	TRUE

プログラムのバージョン情報を出力して終了します。

- `--verbose, -v`

コマンド行形式	<code>--verbose</code>
型	Boolean
デフォルト値	TRUE

冗長出力。`--silent` で無効にします。

23.4.17 ndb_print_backup_file — NDB バックアップファイルの内容の出力

`ndb_print_backup_file` は、クラスタバックアップファイルから診断情報を取得します。

使用法

```
ndb_print_backup_file [-P password] file_name
```

`file_name` はクラスタバックアップファイル名です。これには、クラスタバックアップディレクトリにある任意のファイル (`.Data`、`.ctl`、または `.log` ファイル) を指定できます。これらのファイルは、データノードのバックアップディレクトリのサブディレクトリ `BACKUP-#` にあります。ここで、`#` はバックアップのシーケンス番号です。クラスタバックアップファイルおよびその内容については、[セクション23.5.8.1「NDB Cluster バックアップの概念」](#)を参照してください。

`ndb_print_schema_file` および `ndb_print_sys_file` と同様に (および管理サーバーホストで実行するか、管理サーバーに接続することが意図されているほかのほとんどの NDB ユーティリティと異なり)、`ndb_print_backup_file` は、データノードファイルシステムに直接アクセスするため、クラスタデータノードで実行する必要があります。このユーティリティは管理サーバーを使用しないため、管理サーバーが実行されていない場合、およびクラスタが完全にシャットダウンされている場合にも使用できます。

NDB 8.0.17 以降では、このプログラムを使用して undo ログファイルを読み取ることもできます。

NDB 8.0.22 以降、このプログラムは暗号化されたバックアップを読み取ることができます。これを行うには、バックアップの暗号化に使用するパスワードとともに `-P` オプションを指定します。

その他のオプション

なし

23.4.18 ndb_print_file — NDB データファイル内容の出力

`ndb_print_file` は NDB Cluster データファイルから情報を取得します。

使用方法

```
ndb_print_file [-v] [-q] file_name+
```

`file_name` は NDB Cluster データファイルの名前です。複数のファイル名はスペースで区切ることによって受け入れられます。

`ndb_print_schema_file` や `ndb_print_sys_file` と同様に (および管理サーバーホスト上で実行することや管理サーバーに接続することを意図しているほかのほとんどの NDB ユーティリティとは異なり)、`ndb_print_file` はデータノードファイルシステムに直接アクセスするため、NDB Cluster データノード上で実行する必要があります。このユーティリティは管理サーバーを使用しないため、管理サーバーが実行されていない場合、およびクラスタが完全にシャットダウンされている場合にも使用できます。

その他のオプション

`ndb_print_file` では、次のオプションがサポートされます。

- `-v`: 出力を冗長にします。
- `-q`: 出力を抑制します (抑制モード)。
- `--help, -h, -?`: ヘルプメッセージを出力します。

詳細は、[セクション 23.5.10 「NDB Cluster データテーブル」](#) を参照してください。

23.4.19 ndb_print_frag_file — NDB フラグメントリストファイルの内容の出力

`ndb_print_frag_file` は、クラスタフラグメントリストファイルから情報を取得します。これは、データノードの再起動に関する問題の診断に役立つことを目的としています。

使用方法

```
ndb_print_frag_file file_name
```

`file_name` はクラスタフラグメントリストファイルの名前で、`SX.FragList` というパターンに一致します。`X` は 2-9 の範囲内の数字で、ノード ID `nodeid` を持つデータノードのデータノードファイルシステムにあり、`ndb_nodeid_fs/DN/DBDIH/` という名前のディレクトリにあります。`N` は 1 または 2 です。各フラグメントファイルには、各 NDB テーブルに属するフラグメントのレコードが含まれます。クラスタフラグメントファイルの詳細は、[NDB Cluster Data Node File System Directory](#) を参照してください。

`ndb_print_backup_file`、`ndb_print_sys_file` および `ndb_print_schema_file` と同様に (および管理サーバーホスト上で実行することや管理サーバーに接続することを意図している他の NDB ユーティリティとは異なり)、`ndb_print_frag_file` はデータノードファイルシステムに直接アクセスするため、クラスタデータノード上で実行する必要があります。このユーティリティは管理サーバーを使用しないため、管理サーバーが実行されていない場合、およびクラスタが完全にシャットダウンされている場合にも使用できます。

その他のオプション

なし

出力例

```
shell> ndb_print_frag_file /usr/local/mysqld/data/ndb_3_fs/D1/DBDIH/S2.FragList
Filename: /usr/local/mysqld/data/ndb_3_fs/D1/DBDIH/S2.FragList with size 8192
noOfPages = 1 noOfWords = 182
Table Data
-----
Num Frags: 2 NoOfReplicas: 2 hashpointer: 4294967040
kvalue: 6 mask: 0x00000000 method: HashMap
Storage is on Logged and checkpointed, survives SR
----- Fragment with FragId: 0 -----
Preferred Primary: 2 numStoredReplicas: 2 numOldStoredReplicas: 0 distKey: 0 LogPartId: 0
-----Stored Replica-----
Replica node is: 2 initialGci: 2 numCrashedReplicas = 0 nextLcpNo = 1
LcpNo[0]: maxGciCompleted: 1 maxGciStarted: 2 lcpId: 1 lcpStatus: valid
LcpNo[1]: maxGciCompleted: 0 maxGciStarted: 0 lcpId: 0 lcpStatus: invalid
-----Stored Replica-----
Replica node is: 3 initialGci: 2 numCrashedReplicas = 0 nextLcpNo = 1
LcpNo[0]: maxGciCompleted: 1 maxGciStarted: 2 lcpId: 1 lcpStatus: valid
LcpNo[1]: maxGciCompleted: 0 maxGciStarted: 0 lcpId: 0 lcpStatus: invalid
----- Fragment with FragId: 1 -----
Preferred Primary: 3 numStoredReplicas: 2 numOldStoredReplicas: 0 distKey: 0 LogPartId: 1
-----Stored Replica-----
Replica node is: 3 initialGci: 2 numCrashedReplicas = 0 nextLcpNo = 1
LcpNo[0]: maxGciCompleted: 1 maxGciStarted: 2 lcpId: 1 lcpStatus: valid
LcpNo[1]: maxGciCompleted: 0 maxGciStarted: 0 lcpId: 0 lcpStatus: invalid
-----Stored Replica-----
Replica node is: 2 initialGci: 2 numCrashedReplicas = 0 nextLcpNo = 1
LcpNo[0]: maxGciCompleted: 1 maxGciStarted: 2 lcpId: 1 lcpStatus: valid
LcpNo[1]: maxGciCompleted: 0 maxGciStarted: 0 lcpId: 0 lcpStatus: invalid
```

23.4.20 ndb_print_schema_file — NDB スキーマファイル内容の出力

`ndb_print_schema_file` は、クラスタスキーマファイルから診断情報を取得します。

使用法

```
ndb_print_schema_file file_name
```

`file_name` はクラスタスキーマファイル名です。クラスタスキーマファイルについては、[NDB Cluster Data Node File System Directory](#)を参照してください。

`ndb_print_backup_file` や `ndb_print_sys_file` と同様に (管理サーバーホスト上で実行することや管理サーバーに接続することを意図している他の `NDB` ユーティリティとは異なり)、`ndb_print_schema_file` はデータノードファイルシステムに直接アクセスするため、クラスタデータノード上で実行する必要があります。このユーティリティは管理サーバーを使用しないため、管理サーバーが実行されていない場合、およびクラスタが完全にシャットダウンされている場合にも使用できます。

その他のオプション

なし

23.4.21 ndb_print_sys_file — NDB システムファイル内容の出力

`ndb_print_sys_file` は `NDB Cluster` システムファイルから診断情報を取得します。

使用法

```
ndb_print_sys_file file_name
```

`file_name` はクラスタシステムファイル (sysfile) 名です。クラスタシステムファイルはデータノードのデータディレクトリ (`DataDir`) にあり、このディレクトリからシステムファイルへのパスは `ndb_#_fs/D#/DBDIH/P#.sysfile` というパターンに一致します。それぞれの箇所で、`#` は数字を表しています (同じ数字であるとはかぎりません)。詳細は、[NDB Cluster Data Node File System Directory](#)を参照してください。

`ndb_print_backup_file` および `ndb_print_schema_file` と同様に (および管理サーバーのホストで実行すること、または管理サーバーに接続することが意図されているほかのほとんどの NDB ユーティリティーと異なり)、`ndb_print_backup_file` は、データノードファイルシステムに直接アクセスするため、クラスタデータノードで実行する必要があります。このユーティリティーは管理サーバーを使用しないため、管理サーバーが実行されていない場合、およびクラスタが完全にシャットダウンされている場合にも使用できます。

その他のオプション

なし

23.4.22 ndb_redo_log_reader — クラスタ redo ログの内容の確認および印刷

redo ログファイルの読み取り、エラーのチェック、判読可能な形式での内容の出力、またはその両方を行います。`ndb_redo_log_reader` は、主に NDB Cluster 開発者およびサポート担当者が問題のデバッグおよび診断に使用することを目的としています。

このユーティリティーはまだ開発中であり、その構文と動作は NDB Cluster の将来のリリースで変更される可能性があります。

`ndb_redo_log_reader` の C++ ソースファイルは、`/storage/ndb/src/kernel/blocks/dblqh/redoLogReader` ディレクトリにあります。

次のテーブルに、NDB Cluster プログラム `ndb_redo_log_reader` に固有のオプションを示します。追加説明が表のあとにあります。ほとんどの NDB Cluster プログラム (`ndb_redo_log_reader` を含む) に共通のオプションについては、[セクション23.4.32「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#)を参照してください。

表 23.39 プログラムで使用されるコマンドライン・オプション `ndb_redo_log_reader`

形式	説明	追加、非推奨、または削除された
<code>-dump</code>	ダンプ情報の出力	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>-filedescriptors</code>	ファイル記述子のみを出力	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--help</code>	使用方法の情報を出力します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>-lap</code>	最大 GCI が開始および完了したラップ情報を指定	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>-mbyte #</code>	開始メガバイト	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>-mbyteheaders</code>	ファイル内の各メガバイトの最初のページヘッダーのみ表示	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>-nocheck</code>	レコードのエラーをチェックしません	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>-noprint</code>	レコードを出力しません	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>-page #</code>	このページから開始	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>-pageheaders</code>	ページヘッダーのみ表示	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>-pageindex #</code>	このページインデックスで開始	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>-twiddle</code>	ビットシフトダンプ	(MySQLに基づくすべてのNDBリリースでサポート 8.0)

使用方法

```
ndb_redo_log_reader file_name [options]
```

`file_name` は、クラスタ redo ログファイルの名前です。redo ログファイルは、データノードデータディレクトリ (`DataDir`) の下の番号付きディレクトリにあります。このディレクトリの下の redo ログファイルへのパスは、`ndb_nodeid_fs/D#/DBLQH/S#.FragLog` のパターンと一致します。`nodeid` はデータノードのノード ID です。`#` の 2 つのインスタンスは、それぞれ数値 (必ずしも同じ数値ではない) を表します。`D` に続く数値は 8-39 の範囲です。`S` に続く数値の範囲は、`NoOfFragmentLogFiles` 構成パラメータの値 (デフォルト値は 16) によって異なります。したがって、ファイル名の数値のデフォルト範囲は 0-15 です。詳細は、[NDB Cluster Data Node File System Directory](#) を参照してください。

読み取られるファイルの名前の後ろには、次に一覧する 1 つ以上のオプションが続くことがあります。

- `-dump`

コマンド行形式	<code>-dump</code>
型	Boolean
デフォルト値	FALSE

ダンプ情報を出力します。

- | | |
|---------|-------------------------------|
| コマンド行形式 | <code>-filedescriptors</code> |
| 型 | Boolean |
| デフォルト値 | FALSE |

`-filedescriptors`: ファイル記述子のみを出力します。

- | | |
|---------|---------------------|
| コマンド行形式 | <code>--help</code> |
|---------|---------------------|

`--help`: 使用方法の情報を出力します。

- `-lap`

コマンド行形式	<code>-lap</code>
型	Boolean
デフォルト値	FALSE

最大 GCI が開始および完了したラップ情報を指定してください。

- | | |
|---------|-----------------------|
| コマンド行形式 | <code>-mbyte #</code> |
| 型 | 数値 |
| デフォルト値 | 0 |
| 最小値 | 0 |
| 最大値 | 15 |

`-mbyte #`: 開始メガバイト。

`#` は、0 から 15 の範囲の整数です。

- | | |
|---------|----------------------------|
| コマンド行形式 | <code>-mbyteheaders</code> |
| 型 | Boolean |
| デフォルト値 | FALSE |

`-mbyteheaders`: ファイル内のすべての MB の最初のページヘッダーのみを表示します。

- | | |
|---------|-----------------------|
| コマンド行形式 | <code>-noprint</code> |
| 型 | Boolean |
| デフォルト値 | FALSE |

-noprnt: ログファイルの内容を出力しません。

• コマンド行形式	-nocheck
型	Boolean
デフォルト値	FALSE

-nocheck: ログファイルのエラーをチェックしません。

• コマンド行形式	-page #
型	Integer
デフォルト値	0
最小値	0
最大値	31

-page #: このページから開始します。

#は、0 から 31 の範囲の整数です。

• コマンド行形式	-pageheaders
型	Boolean
デフォルト値	FALSE

-pageheaders: ページヘッダーのみ表示します。

• コマンド行形式	-pageindex #
型	Integer
デフォルト値	12
最小値	12
最大値	8191

-pageindex #: このページインデックスから開始します。

#は、12 以上 8191 以下の整数です。

- **-twiddle**

コマンド行形式	-twiddle
型	Boolean
デフォルト値	FALSE

Bit-shifted dump.

ndb_print_backup_file や **ndb_print_schema_file** と同様に (管理サーバーホスト上で実行することや管理サーバーに接続することを意図している **NDB** ユーティリティとは異なり)、**ndb_redo_log_reader** はデータノードファイルシステムに直接アクセスするため、クラスタデータノード上で実行する必要があります。このユーティリティは管理サーバーを使用しないため、管理サーバーが実行されていない場合、およびクラスタが完全にシャットダウンされている場合にも使用できます。

23.4.23 ndb_restore — NDB Cluster バックアップの復元

NDB Cluster リストアプログラムは、通常は MySQL **bin** ディレクトリにある個別のコマンド行ユーティリティ **ndb_restore** として実装されます。このプログラムは、バックアップの結果として作成されたファイルを読み取り、格納されている情報をデータベースに挿入します。

注記

NDB 8.0.17 以降、このプログラムは実行の終了時に `NDBT_ProgramExit: ...` を出力しなくなりました。NDB 8.0.16 以前から NDB 8.0 以降のリリースにアップグレードする場合は、この動作に依存するアプリケーションを適宜変更するようにしてください。

`ndb_restore` は、バックアップを作成するために使用される `START BACKUP` コマンド ([セクション23.5.8.2「NDB Cluster 管理クライアントを使用したバックアップの作成」](#)を参照してください) によって作成されたバックアップファイルごとに、一度実行する必要があります。これは、バックアップが作成された時点の、クラスタ内のデータノード数と同じです。

注記

複数のデータノードを並列でリストアしている場合を除き、`ndb_restore` を使用する前に、クラスタをシングルユーザーモードで実行することをお勧めします。詳細は、[セクション23.5.6「NDB Cluster のシングルユーザーモード」](#)を参照してください。

次のテーブルに、NDB Cluster ネイティブバックアップリストアッププログラム `ndb_restore` に固有のオプションを示します。追加説明が表のあとにあります。ほとんどの NDB Cluster プログラム (`ndb_restore` を含む) に共通のオプションについては、[セクション23.4.32「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#)を参照してください。

表 23.40 プログラムで使用されるコマンドライン・オプション `ndb_restore`

形式	説明	追加、非推奨、または削除された
<code>--allow-pk-changes[=0 1]</code>	テーブルの主キーを構成するカラムセットの変更を許可	追加: NDB 8.0.21
<code>--append</code>	タブ区切りファイルへのデータの追加	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--backup-password=string</code>	<code>--decrypt</code> を使用して暗号化バックアップを復号化するためのパスワードを指定します。許可される値については、ドキュメントを参照してください	追加: NDB 8.0.22
<code>--backup-path=dir_name</code>	バックアップファイルディレクトリへのパス	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--backupid=#,</code> <code>-b</code>	この ID を持つバックアップからのリストア	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--connect,</code>	<code>--connectstring</code> のエイリアス	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>-c</code> <code>--decrypt</code>	暗号化バックアップを復号化します。 <code>--backup-password</code> が必要です	追加: NDB 8.0.22
<code>--disable-indexes</code>	バックアップからのインデックスが無視されます。データのリストアに必要な時間が短縮される可能性があります	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--dont-ignore-systab-0,</code> <code>-f</code>	リストア中はシステムテーブルを無視しないでください。実験のみで、本番での使用には使用しません	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--exclude-databases=db-list</code>	1 つ以上の除外するデータベースのリスト (指定されていないものを含めません)	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--exclude-intermediate-sql-tables[=TRUE FALSE]</code>	TRUE (デフォルト) の場合、ALTER TABLE 操作のコピーから残された中間テーブル (接頭辞 <code>#sql-'</code> が付いた名前) をリストアしません	(MySQLに基づくすべてのNDBリリースでサポート 8.0)

形式	説明	追加、非推奨、または削除された
<code>--exclude-missing-columns</code>	データベースのテーブルのバージョンから欠落しているテーブルのバックアップバージョンのカラムは無視されます	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--exclude-missing-tables</code>	データベースから欠落しているバックアップからのテーブルは無視されます	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--exclude-tables=table-list</code>	除外する 1 つ以上のテーブルのリスト (名前のない同じデータベース内のテーブルを含む)。各テーブル参照にはデータベース名が含まれている必要があります	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--fields-enclosed-by=char</code>	フィールドはこの文字で囲まれています	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--fields-optionally-enclosed-by</code>	フィールドはオプションでこの文字で囲まれます	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--fields-terminated-by=char</code>	フィールドはこの文字で終了	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--hex</code>	バイナリタイプを 16 進形式で出力します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--ignore-extended-pk-updates[=0 1]</code>	現在拡張主キーに含まれているカラムの更新を含むログエントリを無視	追加: NDB 8.0.21
<code>--include-databases=db-list</code>	1 つ以上のリストアするデータベースのリスト (指定されていないものを除外します)	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--include-stored-grants</code>	共有ユーザーおよび権限を <code>ndb_sql_metadata</code> テーブルにリストア	追加: NDB 8.0.19
<code>--include-tables=table-list</code>	リストアする 1 つ以上のテーブルのリスト (名前のない同じデータベース内のテーブルを除く)。各テーブル参照にはデータベース名を含める必要があります	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--lines-terminated-by=char</code>	行はこの文字で終了	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--lossy-conversions,</code> <code>-L</code>	バックアップからデータをリストアするときに、カラム値の不可逆変換 (型の昇格または符号の変更) を許可します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--no-binlog</code>	<code>mysqld</code> が接続され、バイナリロギングを使用している場合は、復元されたデータをログに記録しません	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--no-restore-disk-objects,</code> <code>-d</code>	ディスクデータに関連するオブジェクトをリストアしません	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--no-upgrade,</code> <code>-u</code>	VAR データがまだサイズ変更されていない可変サイズ属性の配列タイプをアップグレードせず、カラム属性を変更しません	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--ndb-nodegroup-map=map,</code> <code>-z</code>	NDBCLUSTER ストレージエンジンのノードグループマップ; 構文: リスト (source_nodegroup、destination_nodegroup)	(MySQLに基づくすべてのNDBリリースでサポート 8.0)

形式	説明	追加、非推奨、または削除された
<code>--nodeid=#,</code> <code>-n</code>	バックアップが作成されたノードの ID	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--num-slices=#</code>	スライスごとの復元時に適用するスライスの数	追加: NDB 8.0.20
<code>--parallelism=#,</code> <code>-p</code>	データのリストア中に使用する並列トランザクションの数	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--preserve-trailing-spaces,</code> <code>-P</code>	固定幅文字列型を可変幅型に昇格するときに、末尾の空白 (パディングを含む) を維持することを許可します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--print</code>	メタデータ、データおよびログを stdout に出力します (<code>--print-meta --print-data --print-log</code> と同等)	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--print-data</code>	データを stdout に出力します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--print-log</code>	stdout へのログの出力	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--print-meta</code>	メタデータを stdout に出力します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--print-sql-log</code>	SQL ログを stdout に書き込みます。デフォルトは FALSE です	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--progress-frequency=#</code>	指定された秒数ごとに復元のステータスを出力	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--promote-attributes,</code> <code>-A</code>	バックアップからデータをリストアするときに、属性の昇格を許可します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--rebuild-indexes</code>	バックアップで見つかった順序付けられたインデックスをマルチスレッドで再構築します。使用されるスレッドの数は、BuildIndexThreads の設定によって決定されます	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--remap-column=[db].[tbl].[col]:[fn]:[args]</code>	指定された関数および引数を使用して、指定されたカラムの値にオフセットを適用	追加: NDB 8.0.21
<code>--restore-data,</code> <code>-r</code>	NDB API を使用してテーブルデータを復元し、NDB Cluster にログイン	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--restore-epoch,</code> <code>-e</code>	エポック情報をステータステーブルにリストアします。レプリカクラスタでレプリケーションを開始する場合に役立ちます。ID 0 の <code>mysql.ndb_apply_status</code> の行を更新または挿入	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--restore-meta,</code> <code>-m</code>	NDB API を使用した NDB Cluster へのメタデータの復元	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--restore-privilege-tables</code>	以前 NDB に移動された MySQL 権限テーブルをリストアします	非推奨: NDB 8.0.16
<code>--rewrite-database=olddb,newdb</code>	別の名前のデータベースへのリストア	(MySQLに基づくすべてのNDBリリースでサポート 8.0)

形式	説明	追加、非推奨、または削除された
<code>--skip-broken-objects</code>	バックアップファイルで欠落している BLOB テーブルを無視	(MySQLに基づくすべての NDB リリースでサポート 8.0)
<code>--skip-table-check,</code> <code>-s</code>	リストア中にテーブル構造チェックをスキップ	(MySQLに基づくすべての NDB リリースでサポート 8.0)
<code>--skip-unknown-objects</code>	新しい NDB バージョンから古いバージョンにバックアップを復元するときに、 <code>ndb_restore</code> で認識されないスキーマオブジェクトを無視	(MySQLに基づくすべての NDB リリースでサポート 8.0)
<code>--slice-id=#</code>	スライス ID(スライス別に復元する場合)	追加: NDB 8.0.20
<code>--tab=dir_name,</code> <code>-T dir_name</code>	指定されたパスのテーブルごとにタブ区切りの .txt ファイルを作成	(MySQLに基づくすべての NDB リリースでサポート 8.0)
<code>--verbose=#</code>	出力の冗長性レベル	(MySQLに基づくすべての NDB リリースでサポート 8.0)

このユーティリティーの一般的なオプションを次に示します。

```
ndb_restore [-c connection_string] -n node_id -b backup_id \
  [-m] -r --backup-path=/path/to/backup/files
```

通常、NDB Cluster バックアップから復元する場合、`ndb_restore` には少なくとも `--nodeid` (短い形式: `-n`)、`--backupid` (短い形式: `-b`)、および `--backup-path` のオプション。

NDB 8.0.19 より前では、`ndb_restore` を使用して一意のインデックスを含むテーブルを復元した場合、`--disable-indexes` または `--rebuild-indexes` を含める必要がありました。NDB 8.0.19 以降では、自動メタデータ同期が有効になっている場合、これは必要なくなりました。

`-c` オプションは、クラスタ管理サーバーの場所を `ndb_restore` に指示する接続文字列を指定するために使用します (セクション 23.3.3.3 「NDB Cluster 接続文字列」を参照)。このオプションを使用しない場合、`ndb_restore` は `localhost:1186` の管理サーバーに接続を試みます。このユーティリティーはクラスタ API ノードとして動作するため、クラスタ管理サーバーに接続するための空き接続「スロット」が必要となります。これは、クラスタ `config.ini` ファイル内にそれが使用できる `[api]` セクションまたは `[mysqld]` セクションが少なくとも 1 つ存在する必要があることを意味します。このため、MySQL サーバーまたはほかのアプリケーションに使用されていない空の `[api]` セクションまたは `[mysqld]` セクションを、`config.ini` 内に少なくとも 1 つ用意することをお勧めします (セクション 23.3.3.7 「NDB Cluster での SQL およびその他の API ノードの定義」を参照してください)。

NDB 8.0.22 以降では、`ndb_restore` は `--decrypt` および `--backup-password` を使用して暗号化バックアップを復号化できます。復号化を実行するには、両方のオプションを指定する必要があります。暗号化バックアップの作成の詳細は、`START BACKUP` 管理クライアントコマンドのドキュメントを参照してください。

`ndb_restore` がクラスタに接続されていることを確認するには、`ndb_mgm` 管理クライアントで `SHOW` コマンドを使用します。システムシェルで次のようにすることでこれを実現することもできます。

```
shell> ndb_mgm -e "SHOW"
```

`ndb_restore` で使用されるすべてのオプションの詳細は、次のリストを参照してください:

- `--allow-pk-changes`

コマンド行形式	<code>--allow-pk-changes[=0 1]</code>
導入	8.0.21-ndb-8.0.21
型	Integer
デフォルト値	0

最小値	0
最大値	1

このオプションが 1 に設定されている場合、`ndb_restore` では、テーブル定義の主キーをバックアップ内の同じテーブルの主キーと異なるものにすることができます。これは、複数のテーブルで主キーが変更されている異なるスキーマバージョン間でバックアップおよびリストアを行う場合に望ましいことがあり、テーブルスキーマおよびデータのリストア後に多くの `ALTER TABLE` ステートメントを発行するよりも、`ndb_restore` を使用してリストア操作を実行する方が単純または効率的であるようです。

`--allow-pk-changes` では、主キー定義の次の変更がサポートされています:

- 主キーの拡張: バックアップ内のテーブルスキーマに存在する NULL 以外のカラムは、データベース内のテーブル主キーの一部になります。

重要

テーブルの主キーを拡張する場合、バックアップの実行中に主キーの一部となるカラムは更新しないでください。`ndb_restore` によって検出された更新によって、値が変更されなくてもリストア操作は失敗します。場合によっては、`--ignore-extended-pk-updates` オプションを使用してこの動作をオーバーライドできます。詳細は、このオプションの説明を参照してください。

- 主キーの契約 (1): すでにバックアップスキーマのテーブル主キーの一部であるカラムは、主キーの一部ではありませんが、テーブルには残ります。
- 主キーの契約 (2): すでにバックアップスキーマのテーブル主キーの一部であるカラムは、テーブルから完全に削除されます。

これらの違いは、ステージングテーブルの使用を必要とする BLOB カラムやテキストカラムの変更など、`ndb_restore` でサポートされている他のスキーマの違いと組み合わせることができます。

主キースキーマの変更を使用する一般的なシナリオの基本ステップを次に示します:

- `ndb_restore --restore-meta` を使用したテーブルスキーマのリストア
- 必要なスキーマに変更するか、スキーマを作成
- 目的のスキーマのバックアップ
- 前のステップのバックアップを使用して `ndb_restore --disable-indexes` を実行し、インデックスおよび制約を削除
- `ndb_restore --allow-pk-changes` を (`--ignore-extended-pk-updates`、`--disable-indexes` および必要に応じてその他のオプションとともに) 実行して、すべてのデータをリストア
- 目的のスキーマで作成されたバックアップを使用して `ndb_restore --rebuild-indexes` を実行し、インデックスおよび制約を再構築

主キーを拡張する場合、リストア操作中に `ndb_restore` で一時セカンダリー意インデックスを使用して、古い主キーから新しい主キーにマップする必要がある場合があります。このようなインデックスは、バックアップログから拡張主キーを持つテーブルにイベントを適用する必要がある場合にのみ作成されます。このインデックスは `NDB $RESTORE_PK_MAPPING` という名前で、それを必要とする各テーブルに作成されます。必要に応じて、パラレルで実行されている `ndb_restore` インスタンスの複数のインスタンスで共有できます。(リストアプロセスの最後に `ndb_restore --rebuild-indexes` を実行すると、このインデックスは削除されます。)

- `--append`

コマンド行形式	<code>--append</code>
---------	-----------------------

`--tab` および `--print-data` オプションとともに使用すると、同じ名前の既存のファイルにデータが追加されます。

- `--backup-path=dir_name`

コマンド行形式	<code>--backup-path=dir_name</code>
型	ディレクトリ名
デフォルト値	<code>./</code>

バックアップディレクトリへのパスは必須です。これは、`--backup-path` オプションを使用して `ndb_restore` に提供され、リストアするバックアップの ID バックアップに対応するサブディレクトリを含める必要があります。たとえば、データノードの `DataDir` が `/var/lib/mysql-cluster` である場合、バックアップディレクトリは `/var/lib/mysql-cluster/BACKUP` であり、ID 3 のバックアップのバックアップファイルは `/var/lib/mysql-cluster/BACKUP/BACKUP-3` にあります。パスは、絶対パスまたは `ndb_restore` 実行可能ファイルが配置されているディレクトリへの相対パスで、必要に応じて `backup-path=` を接頭辞として付けることができます。

作成されたデータベースとは異なる構成のデータベースにバックアップをリストアできます。たとえば、2 および 3 というノード ID を持つ 2 つのストレージノードを持つクラスタに作成された、バックアップ ID が 12 のバックアップを、4 つのノードを持つクラスタにリストアするとします。その後、バックアップが作成されたクラスタ内のストレージノードごとに、`ndb_restore` を 2 回実行する必要があります。ただし、`ndb_restore` はあるバージョンの MySQL で実行されているクラスタから作成されたバックアップを、別のバージョンの MySQL で実行されているクラスタに常に復旧できるとはかぎりません。

重要

古いバージョンの `ndb_restore` を使用して新しいバージョンの NDB Cluster から作成されたバックアップを復元することはできません。新しいバージョンの MySQL から作成されたバックアップを古いクラスタに復元できますが、そのためには新しい NDB Cluster バージョンの `ndb_restore` のコピーを使用する必要があります。

たとえば、NDB Cluster 7.5.21 を実行しているクラスタから取得したクラスタバックアップを NDB Cluster 7.4.31 を実行しているクラスタに復元するには、NDB Cluster 7.5.21 ディストリビューションに付属の `ndb_restore` を使用する必要があります。

リストアをより高速にするために、十分な数のクラスタ接続が使用できる場合は、データを並列でリストアできます。より正確に言うと、複数のノードを並列でリストアする場合は、各並列 `ndb_restore` プロセスに使用できる `[api]` または `[mysqld]` セクションがクラスタ `config.ini` ファイルに存在する必要があります。ただし、データファイルは常にログの前に適用する必要があります。

- `--backup-password=password`

コマンド行形式	<code>--backup-password=string</code>
導入	8.0.22-ndb-8.0.22
型	文字列
デフォルト値	<code>[none]</code>

このオプションは、`--decrypt` オプションを使用して暗号化バックアップを復号化するとき使用するパスワードを指定します。これは、バックアップの暗号化に使用されたパスワードと同じである必要があります。

パスワードは 256 文字以内で、一重引用符または二重引用符で囲む必要があります。文字コード 32、35、38、40-91、93、95 および 97-126 を持つ ASCII 文字を含めることができます。つまり、`!,',",$,%,\` および `^` 以外の印刷可能な ASCII 文字を使用できます。

- `--backupid=#, -b`

コマンド行形式	<code>--backupid=#</code>
型	数値

デフォルト値	none
--------	------

このオプションは、バックアップの ID または順序番号を指定するために使用され、バックアップの完了時に表示される `Backup backup_id completed` メッセージに管理クライアントによって表示される番号と同じです。(セクション 23.5.8.2 「NDB Cluster 管理クライアントを使用したバックアップの作成」を参照してください。)

重要

クラスタバックアップをリストアするときは、同じバックアップ ID を持つバックアップからすべてのデータノードをリストアする必要があります。異なるバックアップのファイルを使用すると、クラスタが一貫性のない状態にリストアされ、完全に失敗する可能性があります。

NDB 8.0.15 以降では、このオプションは必須です。

- `--connect, -c`

コマンド行形式	<code>--connect</code>
型	文字列
デフォルト値	<code>localhost:1186</code>

`--ndb-connectstring` のエイリアス。

- `--decrypt`

コマンド行形式	<code>--decrypt</code>
導入	8.0.22-ndb-8.0.22

`--backup-password` オプションで指定されたパスワードを使用して、暗号化バックアップを復号化します。

- `--disable-indexes`

コマンド行形式	<code>--disable-indexes</code>
---------	--------------------------------

ネイティブ NDB バックアップからのデータのリストア中にインデックスのリストアを無効にします。その後、`--rebuild-indexes` を使用してマルチスレッドでインデックスを作成することで、すべてのテーブルのインデックスを一度にリストアできます。これは、非常に大きなテーブルのインデックスを同時に再構築するより高速である必要があります。

- `--dont-ignore-systab-0, -f`

コマンド行形式	<code>--dont-ignore-systab-0</code>
---------	-------------------------------------

通常、テーブルデータおよびメタデータをリストアする場合、`ndb_restore` は、バックアップに存在する NDB システムテーブルのコピーを無視します。`--dont-ignore-systab-0` により、システムテーブルがリストアされます。このオプションは、実験および開発での使用のみが意図されており、本番環境には推奨されていません。

- `--exclude-databases=db-list`

コマンド行形式	<code>--exclude-databases=db-list</code>
型	文字列
デフォルト値	

リストアしない 1 つ以上のデータベースのカンマ区切りリスト。

このオプションは、多くの場合、`--exclude-tables` と組み合わせて使用されます。詳細および例は、そのオプションの説明を参照してください。

- `--exclude-intermediate-sql-tables[=TRUE|FALSE]`

コマンド行形式	<code>--exclude-intermediate-sql-tables[=TRUE FALSE]</code>
型	Boolean
デフォルト値	TRUE

`ALTER TABLE` のコピー操作を実行するときに、`mysqld` は中間テーブルを作成します (名前の前に `#sql-` が付けられます)。TRUE の場合、`--exclude-intermediate-sql-tables` オプションは、`ndb_restore` がこれらの操作から残された可能性のあるテーブルをリストアしないようにします。このオプションはデフォルトでは TRUE です。

- `--exclude-missing-columns`

コマンド行形式	<code>--exclude-missing-columns</code>
---------	--

このオプションを使用して、選択したテーブルのカラムのみをリストアできます。これにより、`ndb_restore` は、バックアップで見つかったテーブルのバージョンと比較して、リストアされるテーブルから欠落しているカラムを無視します。このオプションはリストアされるすべてのテーブルに適用されます。このオプションを選択したテーブルまたはデータベースにのみ適用する場合は、このセクションの他の場所で説明されている `--include-*` または `--exclude-*` のオプションと組み合わせて使用し、これらのオプションの補完セットを使用して残りのテーブルにデータをリストアできます。

- `--exclude-missing-tables`

コマンド行形式	<code>--exclude-missing-tables</code>
---------	---------------------------------------

このオプションを使用して、選択したテーブルのみをリストアできます。これにより、`ndb_restore` は、ターゲットデータベースで見つからないバックアップのテーブルを無視します。

- `--exclude-tables=table-list`

コマンド行形式	<code>--exclude-tables=table-list</code>
型	文字列
デフォルト値	

除外する 1 つ以上のテーブルのリスト。各テーブル参照にはデータベース名を含める必要があります。多くの場合、`--exclude-databases` とともに使用されます。

`--exclude-databases` または `--exclude-tables` を使用した場合は、これらのオプションによって指定されたデータベースまたはテーブルのみが除外されます。ほかのすべてのデータベースおよびテーブルは `ndb_restore` によってリストアされます。

このテーブルは、`--exclude-*` オプションを使用した `ndb_restore` のいくつかの呼び出し (わかりやすくするためにほかのオプションが省略されている可能性があります) と、これらのオプションが NDB Cluster バックアップからの復元に与える影響を示しています:

表 23.41 `--exclude-*` オプションを使用した `ndb_restore` のいくつかの呼び出し、およびこれらのオプションが NDB Cluster バックアップからの復元に与える影響。

オプション	結果
<code>--exclude-databases=db1</code>	<code>db1</code> を除くすべてのデータベース内のすべてのテーブルがリストアされます。 <code>db1</code> 内のテーブルはリストアされません
<code>--exclude-databases=db1,db2</code> (または <code>--exclude-databases=db1 --exclude-databases=db2</code>)	<code>db1</code> および <code>db2</code> を除くすべてのデータベース内のすべてのテーブルがリストアされます。 <code>db1</code> または <code>db2</code> 内のテーブルはリストアされません
<code>--exclude-tables=db1.t1</code>	データベース <code>db1</code> 内の <code>t1</code> を除くすべてのテーブルがリストアされます。 <code>db1</code> 内のほかのすべてのテーブルはリストアされます。ほかのすべてのデータベース内のすべてのテーブルはリストアされます

オプション	結果
<code>--exclude-tables=db1.t2,db2.t1</code> (または <code>--exclude-tables=db1.t2 --exclude-tables=db2.t1</code>)	データベース <code>db1</code> 内の <code>t2</code> を除くすべてのテーブル、およびデータベース <code>db2</code> 内の <code>t1</code> を除くすべてのテーブルがリストアされます。 <code>db1</code> または <code>db2</code> 内のほかのテーブルはリストアされません。ほかのすべてのデータベース内のすべてのテーブルはリストアされます

これらの2つのオプションは一緒に使用できます。たとえば、次のようにすると、すべてのデータベースのすべてのテーブルが except for データベース `db1` および `db2` にリストアされ、データベース `db3` のテーブル `t1` および `t2` がリストアされます:

```
shell> ndb_restore [...] --exclude-databases=db1,db2 --exclude-tables=db3.t1,db3.t2
```

(この例でも、わかりやすく簡潔にするために、必要となる可能性があるほかのオプションを省略しています)。

`--include-*` オプションおよび `--exclude-*` オプションは、次のルールに従って一緒に使用できます。

- すべての `--include-*` および `--exclude-*` オプションのアクションは累積されます。
- すべての `--include-*` および `--exclude-*` オプションは、`ndb_restore` に渡された順序で右から左に評価されます。
- 競合するオプションがある場合は、最初(もっとも右側)のオプションが優先されます。言い換えると、対象となるデータベースまたはテーブルと一致する最初のオプション(右から左に適用して)が「適用されます」。

たとえば、次の一連のオプションを指定すると、`ndb_restore` がデータベース `db1` の `db1.t1` を除くすべてのテーブルをリストアし、ほかのデータベースのほかのテーブルはリストアしません。

```
--include-databases=db1 --exclude-tables=db1.t1
```

ただし、前述のオプションの順序を逆にすると、データベース `db1` のすべてのテーブルがリストアされます(`db1.t1` は含まれますが、ほかのデータベースのテーブルは含まれません)。これは、もっとも右にある `--include-databases` オプションがデータベース `db1` に対して最初に一致し、`db1` または `db1` 内のテーブルと一致するその他のオプションより優先されるためです。

```
--exclude-tables=db1.t1 --include-databases=db1
```

- `--fields-enclosed-by=char`

コマンド行形式	<code>--fields-enclosed-by=char</code>
型	文字列
デフォルト値	

各カラム値は、このオプションに渡される文字列で囲われます(データ型に関係なく、`--fields-optionally-enclosed-by` の説明を参照)。

- `--fields-optionally-enclosed-by`

コマンド行形式	<code>--fields-optionally-enclosed-by</code>
型	文字列
デフォルト値	

このオプションに渡された文字列は、文字データが含まれているカラム値(`CHAR`、`VARCHAR`、`BINARY`、`TEXT`、`ENUM` など)を囲むために使用されます。

- `--fields-terminated-by=char`

コマンド行形式	<code>--fields-terminated-by=char</code>
型	文字列

デフォルト値	\t (tab)
--------	----------

このオプションに渡された文字列は、カラム値を区切るために使用されます。デフォルト値はタブ文字 (\t) です。

- `--hex`

コマンド行形式	<code>--hex</code>
---------	--------------------

このオプションを使用すると、すべてのバイナリ値は 16 進形式で出力されます。

- `--ignore-extended-pk-updates`

コマンド行形式	<code>--ignore-extended-pk-updates[=0 1]</code>
導入	8.0.21-ndb-8.0.21
型	Integer
デフォルト値	0
最小値	0
最大値	1

`--allow-pk-changes` を使用する場合、バックアップの実行中にテーブルの主キーの一部になるカラムは更新しないでください。そのようなカラムは、値を含む行が削除されるまで、時間値から同じ値が挿入されたままにしておく必要があります。`ndb_restore` がバックアップのリストア時にこれらのカラムの更新を検出すると、リストアは失敗します。一部のアプリケーションでは、行の更新時にすべてのカラムの値が設定される場合があるため、一部のカラム値が変更されていない場合でも、実際には変更されていないカラムの更新に表示されるログイベントがバックアップに含まれることがあります。このような場合は、`--ignore-extended-pk-updates` を 1 に設定して、`ndb_restore` でこのような更新を強制的に無視できます。

重要

これらの更新を無視する場合、ユーザーは主キーの一部になるカラムの値に対する更新がないことを確認する必要があります。

詳細は、`--allow-pk-changes` の説明を参照してください。

- `--include-databases=db-list`

コマンド行形式	<code>--include-databases=db-list</code>
型	文字列
デフォルト値	

リストアする 1 つ以上のデータベースのカンマ区切りリスト。多くの場合、`--include-tables` とともに使用されます。詳細および例は、そのオプションの説明を参照してください。

- `--include-stored-grants`

コマンド行形式	<code>--include-stored-grants</code>
導入	8.0.19-ndb-8.0.19
型	Boolean
デフォルト値	FALSE

NDB 8.0.19 以降では、`ndb_restore` はデフォルトで、`ndb_sql_metadata` テーブルへの共有ユーザーおよび権限付与 (セクション 23.5.12 「NDB_STORED_USER での分散 MySQL 権限」 を参照) を復元しません。このオプションを指定すると、これが実行されます。

- `--include-tables=table-list`

コマンド行形式	<code>--include-tables=table-list</code>
---------	--

型	文字列
デフォルト値	

リストアするテーブルのカンマ区切りリスト。各テーブル参照にはデータベース名を含める必要があります。

`--include-databases` または `--include-tables` を使用すると、それらのオプションによって指定されたデータベースまたはテーブルのみがリストアされます。ほかのすべてのデータベースおよびテーブルは `ndb_restore` によって除外され、リストアされません。

次のテーブルに、`--include-*` オプションを使用した `ndb_restore` のいくつかの呼び出し (わかりやすくするためにほかのオプションが省略されている可能性があります) と、NDB Cluster バックアップからの復元に対するそれらの影響を示します:

表 23.42 `--include-*` オプションを使用した `ndb_restore` のいくつかの呼び出しと、NDB Cluster バックアップからの復元へのそれらの影響。

オプション	結果
<code>--include-databases=db1</code>	データベース <code>db1</code> 内のテーブルのみがリストアされます。ほかのすべてのデータベース内のすべてのテーブルは無視されます
<code>--include-databases=db1,db2</code> (または <code>--include-databases=db1 --include-databases=db2</code>)	データベース <code>db1</code> および <code>db2</code> 内のテーブルのみがリストアされます。ほかのすべてのデータベース内のすべてのテーブルは無視されます
<code>--include-tables=db1.t1</code>	データベース <code>db1</code> 内のテーブル <code>t1</code> のみがリストアされます。 <code>db1</code> またはその他のデータベース内のほかのテーブルはリストアされません
<code>--include-tables=db1.t2,db2.t1</code> (または <code>--include-tables=db1.t2 --include-tables=db2.t1</code>)	データベース <code>db1</code> 内のテーブル <code>t2</code> およびデータベース <code>db2</code> 内のテーブル <code>t1</code> のみがリストアされます。 <code>db1</code> 、 <code>db2</code> 、またはその他のデータベース内のほかのテーブルはリストアされません

これらの 2 つのオプションは一緒に使用することもできます。たとえば、次の場合は、データベース `db1` および `db2` 内のすべてのテーブルに加えて、データベース `db3` 内のテーブル `t1` および `t2` がリストアされます (ほかのデータベースまたはテーブルはリストアされません)。

```
shell> ndb_restore [...] --include-databases=db1,db2 --include-tables=db3.t1,db3.t2
```

(この例でも、必要となる可能性があるほかのオプションを省略しています。)

次に示す構文を使用して、`--include-*` (または `--exclude-*`) オプションなしで、選択したデータベースまたは選択したテーブルのみを単一のデータベースからリストアすることもできます:

```
ndb_restore other_options db_name,[db_name[...]|tbl_name[tbl_name][...]]
```

言い換えると、次のいずれかをリストアすることを指定できます。

- 1 つ以上のデータベースのすべてのテーブル
- 単一データベースの 1 つ以上のテーブル
- `--lines-terminated-by=char`

コマンド行形式	<code>--lines-terminated-by=char</code>
型	文字列
デフォルト値	<code>\n</code> (linebreak)

出力の各行を終了するために使用する文字列を指定します。デフォルトは改行文字 (`\n`) です。

- `--lossy-conversions, -L`

コマンド行形式	<code>--lossy-conversions</code>
型	Boolean
デフォルト値	FALSE (オプションを使用しない場合)

このオプションは、`--promote-attributes` オプションを補完することを意図しています。 `--lossy-conversions` を使用するときは、バックアップからデータをリストアするときにカラム値の不可逆変換 (型降格または符号の変更) が許可されます。いくつかの例外はありますが、降格を制御するルールは MySQL レプリケーションの場合と同じです。属性降格によって現在サポートされる型変換については、[データ型が異なるカラムのレプリケーション](#)を参照してください。

`ndb_restore` は、不可逆変換中に実行されるデータの切り捨てを属性およびカラムごとに報告します。

- `--no-binlog`

コマンド行形式	<code>--no-binlog</code>
---------	--------------------------

このオプションは、接続されている SQL ノードが `ndb_restore` によってリストアされるデータをバイナリログに書き込むことを阻止します。

- `--no-restore-disk-objects, -d`

コマンド行形式	<code>--no-restore-disk-objects</code>
型	Boolean
デフォルト値	FALSE

このオプションは、`ndb_restore` による NDB Cluster ディスクデータオブジェクト (テーブルスペースやログファイルグループなど) の復元を停止します。これらの詳細は、[セクション23.5.10「NDB Cluster ディスクデータテーブル」](#)を参照してください。

- `--no-upgrade, -u`

コマンド行形式	<code>--no-upgrade</code>
---------	---------------------------

`ndb_restore` を使用してバックアップをリストアする場合、古い固定長形式を使用して作成された `VARCHAR` カラムは、現在採用されている可変幅形式を使用してサイズ変更および再作成されます。この動作は、`--no-upgrade` を指定することでオーバーライドできます。

- `--ndb-nodegroup-map=map, -z`

コマンド行形式	<code>--ndb-nodegroup-map=map</code>
---------	--------------------------------------

このオプションは、あるノードグループから取得されたバックアップを別のノードグループにリストアするために使用できます。引数は、`source_node_group, target_node_group` という形式のリストです。

- `--nodeid=#, -n`

コマンド行形式	<code>--nodeid=#</code>
型	数値
デフォルト値	none

バックアップが作成されたデータノードのノード ID を指定します。

複数のファイルを単一のデータノードに復元する必要があります。) 追加情報および例については、[セクション 23.4.23.2 「異なる数のデータノードへの復元」](#)を参照してください。

NDB 8.0.15 以降では、このオプションは必須です。

- `--num-slices=#`

コマンド行形式	<code>--num-slices=#</code>
導入	8.0.20-ndb-8.0.20
型	Integer
デフォルト値	1
最小値	1
最大値	1024

スライスごとにバックアップを復元する場合、このオプションはバックアップを分割するスライスの数を設定します。これにより、`ndb_restore` の複数のインスタンスで非結合サブセットをパラレルにリストアできるため、リストア操作の実行に必要な時間が短縮される可能性があります。

スライスは、特定のバックアップ内のデータのサブセットです。つまり、`--slice-id` オプションを使用して指定された同じスライス ID を持つフラグメントのセットです。2 つのオプションは常に一緒に使用する必要があり、`--slice-id` によって設定される値は常にスライスの数より小さくする必要があります。

`ndb_restore` はフラグメントを検出し、各フラグメントにフラグメントカウンタを割り当てます。スライスごとに復元する場合、スライス ID は各フラグメントに割り当てられます。このスライス ID は、スライスの数より 0 から 1 の範囲内にあります。BLOB テーブルではないテーブルの場合、特定のフラグメントが属するスライスは、次に示す式を使用して決定されます:

```
[slice_ID] = [fragment_counter] % [number_of_slices]
```

BLOB テーブルの場合、フラグメントカウンタは使用されず、かわりにフラグメント番号が BLOB テーブルのメインテーブルの ID とともに使用されます (NDB では BLOB 値が内部的に別のテーブルに格納されることに注意してください)。この場合、特定のフラグメントのスライス ID は次のように計算されます:

```
[slice_ID] =
([main_table_ID] + [fragment_ID]) % [number_of_slices]
```

したがって、N スライスによるリストアとは、`ndb_restore` の N インスタンスを実行し、すべて `--num-slices= N` (およびその他の必要なオプション) を使用し、それぞれに `--slice-id=1`, `--slice-id=2`, `--slice-id=3` を使用して `slice-id=N-1` を介して実行することです。

例. 各データノードのノードファイルシステム上のデフォルトディレクトリ `/var/lib/mysql-cluster/BACKUP/BACKUP-3` にある `BACKUP-1` という名前のバックアップを、ノード ID 1、2、3、および 4 を持つ 4 つのデータノードを持つクラスタに復元するとします。5 つのスライスを使用してこの操作を実行するには、次のリストに示す一連のコマンドを実行します:

1. 次に示すように、`ndb_restore` を使用してクラスタメタデータをリストアします:

```
shell> ndb_restore -b 1 -n 1 -m --disable-indexes --backup-path=/home/ndbuser/backups
```

2. 次に示すように、`ndb_restore` を起動するデータノードにクラスタデータを復元します:

```
shell> ndb_restore -b 1 -n 1 -r --num-slices=5 --slice-id=0 --backup-path=/var/lib/mysql-cluster/BACKUP/BACKUP-1
shell> ndb_restore -b 1 -n 1 -r --num-slices=5 --slice-id=1 --backup-path=/var/lib/mysql-cluster/BACKUP/BACKUP-1
shell> ndb_restore -b 1 -n 1 -r --num-slices=5 --slice-id=2 --backup-path=/var/lib/mysql-cluster/BACKUP/BACKUP-1
shell> ndb_restore -b 1 -n 1 -r --num-slices=5 --slice-id=3 --backup-path=/var/lib/mysql-cluster/BACKUP/BACKUP-1
shell> ndb_restore -b 1 -n 1 -r --num-slices=5 --slice-id=4 --backup-path=/var/lib/mysql-cluster/BACKUP/BACKUP-1

shell> ndb_restore -b 1 -n 2 -r --num-slices=5 --slice-id=0 --backup-path=/var/lib/mysql-cluster/BACKUP/BACKUP-1
shell> ndb_restore -b 1 -n 2 -r --num-slices=5 --slice-id=1 --backup-path=/var/lib/mysql-cluster/BACKUP/BACKUP-1
shell> ndb_restore -b 1 -n 2 -r --num-slices=5 --slice-id=2 --backup-path=/var/lib/mysql-cluster/BACKUP/BACKUP-1
shell> ndb_restore -b 1 -n 2 -r --num-slices=5 --slice-id=3 --backup-path=/var/lib/mysql-cluster/BACKUP/BACKUP-1
shell> ndb_restore -b 1 -n 2 -r --num-slices=5 --slice-id=4 --backup-path=/var/lib/mysql-cluster/BACKUP/BACKUP-1
```

```

shell> ndb_restore -b 1 -n 3 -r --num-slices=5 --slice-id=0 --backup-path=/var/lib/mysql-cluster/BACKUP/BACKUP-1
shell> ndb_restore -b 1 -n 3 -r --num-slices=5 --slice-id=1 --backup-path=/var/lib/mysql-cluster/BACKUP/BACKUP-1
shell> ndb_restore -b 1 -n 3 -r --num-slices=5 --slice-id=2 --backup-path=/var/lib/mysql-cluster/BACKUP/BACKUP-1
shell> ndb_restore -b 1 -n 3 -r --num-slices=5 --slice-id=3 --backup-path=/var/lib/mysql-cluster/BACKUP/BACKUP-1
shell> ndb_restore -b 1 -n 3 -r --num-slices=5 --slice-id=4 --backup-path=/var/lib/mysql-cluster/BACKUP/BACKUP-1

shell> ndb_restore -b 1 -n 4 -r --num-slices=5 --slice-id=0 --backup-path=/var/lib/mysql-cluster/BACKUP/BACKUP-1
shell> ndb_restore -b 1 -n 4 -r --num-slices=5 --slice-id=1 --backup-path=/var/lib/mysql-cluster/BACKUP/BACKUP-1
shell> ndb_restore -b 1 -n 4 -r --num-slices=5 --slice-id=2 --backup-path=/var/lib/mysql-cluster/BACKUP/BACKUP-1
shell> ndb_restore -b 1 -n 4 -r --num-slices=5 --slice-id=3 --backup-path=/var/lib/mysql-cluster/BACKUP/BACKUP-1
shell> ndb_restore -b 1 -n 4 -r --num-slices=5 --slice-id=4 --backup-path=/var/lib/mysql-cluster/BACKUP/BACKUP-1

```

クラスタへの接続に十分なスロットがある場合は、このステップで示したすべてのコマンドをパラレルで実行できます (--backup-path オプションの説明を参照)。

- 次に示すように、通常どおりにインデックスをリストアします:

```
shell> ndb_restore -b 1 -n 1 --rebuild-indexes --backup-path=/var/lib/mysql-cluster/BACKUP/BACKUP-1
```

- 最後に、次に示すコマンドを使用してエポックをリストアします:

```
shell> ndb_restore -b 1 -n 1 --restore-epoch --backup-path=/var/lib/mysql-cluster/BACKUP/BACKUP-1
```

スライス、クラスタデータのリストアにのみ使用する必要があります。メタデータ、インデックスまたはエポック情報のリストア時に --num-slices または --slice-id を使用する必要はありません。これらの復元を制御する ndb_restore オプションとともにこれらのオプションの一方または両方が使用されている場合、プログラムはそれらを無視します。

--parallelism オプションを使用したリストア速度への影響は、ndb_restore の複数のインスタンスを使用したスライスまたはパラレルリストアによって生成されるものとは無関係です (--parallelism は single ndb_restore スレッドによって実行されるパラレルトランザクションの数を指定します)。ただし、これらのいずれかまたは両方ともを使用できます。--parallelism を増やすと、ndb_restore のクラスタに大きな負荷がかかることに注意してください。システムでこれを処理できる場合は、リストアをさらに迅速に完了する必要があります。

--num-slices の値は、CPU または CPU コアの数、RAM の量などのハードウェアに関連する値に直接依存することではなく、LDM の数にも依存しません。

同じリストアの一部として、異なるデータノードでこのオプションに異なる値を使用することもできます。そうしないと、それ自体で悪影響が生じることはありません。

- parallelism=#, -p

コマンド行形式	--parallelism=#
型	数値
デフォルト値	128
最小値	1
最大値	1024

ndb_restore では、単一行トランザクションを使用して多数の行を同時に適用します。このパラメータは、ndb_restore のインスタンスが使用しようとするパラレルトランザクション (同時行) の数を決定します。デフォルトでは、これは 128 であり、最小値は 1、最大値は 1024 です。

挿入を実行する作業は、関係するデータノード内のスレッド間で並列化されます。このメカニズムは、.Data ファイル (データのファジースナップショット) からバルクデータをリストアするために使用されます。インデックスの作成または再構築には使用されません。変更ログはシリアルに適用されます。インデックスの削除および構築は DDL 操作であり、個別に処理されます。リストアのクライアント側にスレッドレベルの並列性はありません。

- `--preserve-trailing-spaces, -P`

コマンド行形式	<code>--preserve-trailing-spaces</code>
---------	---

固定幅文字データ型を同等の可変幅に昇格させるとき、つまり `CHAR` カラム値を `VARCHAR` に昇格させるとき、または `BINARY` カラム値を `VARBINARY` に昇格させるときに、末尾の空白が保持されるようにします。それ以外の場合は、新しいカラムに挿入されるときに、末尾の空白はそのようなカラム値から削除されます。

注記

`CHAR` カラムを `VARCHAR` に、および `BINARY` カラムを `VARBINARY` に昇格することはできますが、`VARCHAR` カラムを `CHAR` に、または `VARBINARY` カラムを `BINARY` に昇格することはできません。

- `--print`

コマンド行形式	<code>--print</code>
型	Boolean
デフォルト値	<code>FALSE</code>

`ndb_restore` がすべてのデータ、メタデータ、およびログを `stdout` に出力します。`--print-data`、`--print-meta` および `--print-log` オプションと一緒に使用することと同等です。

注記

`--print` または `--print_*` オプションのいずれかを使用した場合は、仮実行をするときに有効になります。これらのオプションのいずれかまたは複数を含めると、すべての出力が `stdout` にリダイレクトされます。このような場合、`ndb_restore` は NDB Cluster にデータまたはメタデータを復元しようとしません。

- `--print-data`

コマンド行形式	<code>--print-data</code>
型	Boolean
デフォルト値	<code>FALSE</code>

`ndb_restore` がその出力を `stdout` に送信するようにします。多くの場合、1 つ以上の `--tab`、`--fields-enclosed-by`、`--fields-optionally-enclosed-by`、`--fields-terminated-by`、`--hex` および `--append` とともに使用されます。

`TEXT` および `BLOB` カラム値は常に切り捨てられます。このような値は、出力の最初の 256 バイトに切り捨てられます。現在、`--print-data` を使用している場合はオーバーライドできません。

- `--print-log`

コマンド行形式	<code>--print-log</code>
型	Boolean
デフォルト値	<code>FALSE</code>

`ndb_restore` がログを `stdout` に出力するようにします。

- `--print-meta`

コマンド行形式	<code>--print-meta</code>
型	Boolean
デフォルト値	<code>FALSE</code>

すべてのメタデータを `stdout` に出力します。

- `print-sql-log`

コマンド行形式	<code>--print-sql-log</code>
型	Boolean
デフォルト値	FALSE

SQL ステートメントを `stdout` に記録します。オプションを使用して有効にします。通常、この動作は無効になっています。このオプションは、リストアされるすべてのテーブルに主キーが明示的に定義されているかどうかをログに記録する前にチェックします。NDB によって実装された非表示の主キーのみを持つテーブルに対するクエリは、有効な SQL に変換できません。

このオプションは、BLOB カラムを含むテーブルでは機能しません。

- `--progress-frequency=N`

コマンド行形式	<code>--progress-frequency=#</code>
型	数値
デフォルト値	0
最小値	0
最大値	65535

バックアップの進行中に、ステータスレポートを N 秒ごとに出力します。0 (デフォルト) を指定すると、ステータスレポートは出力されません。最大値は 65535 です。

- `--promote-attributes, -A`

コマンド行形式	<code>--promote-attributes</code>
---------	-----------------------------------

`ndb_restore` では、MySQL レプリケーションでサポートされているのと同様方法で限定された属性プロモーションがサポートされます。つまり、特定のタイプのカラムからバックアップされたデータは、通常、「大規模、類似」タイプを使用してカラムにリストアできます。たとえば、`CHAR(20)` カラムのデータは `VARCHAR(20)`、`VARCHAR(30)`、または `CHAR(30)` として宣言されているカラムにリストアでき、`MEDIUMINT` カラムのデータは、`INT` または `BIGINT` 型のカラムにリストアできます。属性昇格によって現在サポートされる型変換の表については、[データ型が異なるカラムのレプリケーション](#)を参照してください。

`ndb_restore` による属性昇格は、次のように明示的に有効にする必要があります。

1. バックアップをリストアするテーブルを準備します。`ndb_restore` を使用してオリジナルと異なる定義でテーブルを再作成することはできません。これは、テーブルを手動で作成するか、またはテーブルメタデータをリストアしたあとかつデータをリストアする前に昇格するカラムを `ALTER TABLE` を使用して変更する必要があることを意味します。
2. テーブルデータをリストアするときに、`--promote-attributes` オプション (短縮形: `-A`) を指定して `ndb_restore` を呼び出します。このオプションを使用しない場合、属性昇格は行われず、リストア操作がエラーで失敗します。

文字データ型と `TEXT` または `BLOB` の間で変換する場合は、文字型 (`CHAR` および `VARCHAR`) とバイナリ型 (`BINARY` および `VARBINARY`) の間の変換のみを同時に実行できます。たとえば、同じ `ndb_restore` 呼び出しで、`VARCHAR` カラムを `TEXT` に昇格するときに、`INT` カラムを `BIGINT` に昇格できません。

異なる文字セットを使用した `TEXT` カラム間の変換はサポートされておらず、明示的には許可されません。

`ndb_restore` を使用して文字型またはバイナリ型から `TEXT` または `BLOB` への変換を実行するときに、`table_name$STnode_id` という名前の 1 つ以上のステージングテーブルが作成および使用されることに気付くことがあります。これらのテーブルはその後必要ではなくなるため、通常はリストアが成功したあとに `ndb_restore` によって削除されます。

- `--rebuild-indexes`

コマンド行形式	<code>--rebuild-indexes</code>
---------	--------------------------------

ネイティブ NDB バックアップのリストア中に、順序付けられたインデックスのマルチスレッド再構築を有効にします。このオプションを使用して `ndb_restore` によって順序付けられたインデックスを構築するために使用されるスレッドの数は、`BuildIndexThreads` データノード構成パラメータと LDM の数によって制御されます。

このオプションを使用する必要があるのは、`ndb_restore` の最初の実行の場合のみです。これにより、以降のノードをリストアするときに `--rebuild-indexes` をふたたび使用しなくても、すべての順序付けられたインデックスが再構築されます。このオプションはデータベースに新しい行を挿入する前に使用してください。そうしないと、インデックスを再構築しようとするときに、挿入される行があとで一意制約違反になることがあります。

順序付けされたインデックスの構築は、デフォルトで LDM の数で並列化されます。ノードおよびシステムの再起動中に実行されるオフラインインデックス構築は、`BuildIndexThreads` データノード構成パラメータを使用して高速化できます。このパラメータは、オンラインで実行される `ndb_restore` によるインデックスの削除および再構築には影響しません。

一意インデックスの再構築では、Redo ロギングおよびローカルチェックポイント処理のディスク書き込み帯域幅が使用されます。この帯域幅が十分ではない場合、Redo バッファオーバーロードエラーまたはログオーバーロードエラーになることがあります。そのような場合は、`ndb_restore --rebuild-indexes` を再度実行できます。この処理はエラーが発生した箇所から再開されます。これは、一時エラーが発生した場合にも実行できます。`ndb_restore --rebuild-indexes` の実行は無期限に繰り返すことができます。このようなエラーは、`--parallelism` の値を減らすことで停止できる場合があります。領域不足の場合は、redo ログのサイズ (`FragmentLogFileSize` ノード構成パラメータ) を増やすか、LCP の実行速度 (`MaxDiskWriteSpeed` および関連パラメータ) を上げて領域をより迅速に解放できます。

- `--remap-column=db.tbl.col:fn:args`

コマンド行形式	<code>--remap-column=[db].[tbl].[col]:[fn]:[args]</code>
導入	8.0.21-ndb-8.0.21
型	文字列
デフォルト値	<code>[none]</code>

このオプションを `--restore-data` とともに使用すると、示されたカラムの値に関数が適用されます。引数文字列の値を次に示します:

- `db`: `--rewrite-database` によって実行された名前の変更後のデータベース名。
- `tbl`: テーブル名。
- `col`: 更新するカラムの名前。このカラムのタイプは、`INT` または `BIGINT` である必要があります。カラムは `UNSIGNED` でもかまいませんが、必須ではありません。
- `fn`: 関数名。現在サポートされている名前は `offset` のみです。
- `args`: 関数に渡される引数。現在、`offset` 関数によって追加されるオフセットのサイズである単一の引数のみがサポートされています。負の値がサポートされています。引数のサイズは、カラムタイプの符号付きバリエーションのサイズを超えることはできません。たとえば、`col` が `INT` カラムの場合、`offset` 関数に渡される引数の許容範囲は `-2147483648` から `2147483647` (セクション 11.1.2 「整数型 (真数値) - `INTEGER`、`INT`、`SMALLINT`、`TINYINT`、`MEDIUMINT`、`BIGINT`」を参照) です。

オフセット値をカラムに適用すると、オーバーフローまたはアンダーフローが発生する場合、リストア操作は失敗します。これは、たとえば、カラムが `BIGINT` で、 $4294967291 + 8 = 4294967299 > 4294967295$ 以降、カラム値が `4294967291` である行にオフセット値 `8` を適用しようとした場合に発生する可能性があります。

このオプションは、NDB ネイティブバックアップ (セクション 23.5.8.2 「NDB Cluster 管理クライアントを使用したバックアップの作成」を参照) と `ndb_restore` を使用してデータをマージし、主キーと一意キーの値がソースクラスト間で重複しており、これらの値を重複しない範囲に再マップするプロセスの一環として、NDB Cluster の複数のソースインスタンスに格納されているデータを単一の宛先 NDB Cluster にマージする場合に役立ちます。

テーブル間の他の関係を保持する必要がある場合もあります。このような要件を満たすために、次に示すように、`ndb_restore` の同じ起動でオプションを複数回使用して、異なるテーブルのカラムを再マップできます：

```
shell> ndb_restore --restore-data --remap-column=hr.employee.id:offset:1000 \
--remap-column=hr.manager.id:offset:1000 --remap-column=hr.firstaiders.id:offset:1000
```

(ここに示されていない他のオプションも使用できます。)

`--remap-column` を使用して、同じテーブルの複数のカラムを更新することもできます。複数のテーブルとカラムの組合せが可能です。次のように、同じテーブルの異なるカラムに異なるオフセット値を使用することもできます：

```
shell> ndb_restore --restore-data --remap-column=hr.employee.salary:offset:10000 \
--remap-column=hr.employee.hours:offset:-10
```

マージしない重複テーブルがソースバックアップに含まれている場合は、`--exclude-tables`、`--exclude-databases` またはアプリケーション内の他の方法を使用してこれを処理できます。

マージされるテーブルの構造およびその他の特性に関する情報は、`SHOW CREATE TABLE`、`ndb_desc` ツール、`MAX()`、`MIN()`、`LAST_INSERT_ID()` およびその他の MySQL 関数を使用して取得できます。

NDB Cluster の個別のインスタンスで、マージされたテーブルからマージされていないテーブル、またはマージされていないテーブルからマージされたテーブルへの変更のレプリケーションはサポートされていません。

- `--restore-data, -r`

コマンド行形式	<code>--restore-data</code>
型	Boolean
デフォルト値	FALSE

NDB テーブルのデータおよびログを出力します。

- `--restore-epoch, -e`

コマンド行形式	<code>--restore-epoch</code>
---------	------------------------------

エポック情報をクラスタレプリケーションステータステーブルに追加 (またはリストア) します。これは NDB Cluster レプリカでレプリケーションを開始する場合に役立ちます。このオプションを使用すると、`id` カラムが 0 である `mysql.ndb_apply_status` 内の行が更新されます (存在する場合)。存在しない場合はそのような行が挿入されます (セクション 23.6.9 「NDB Cluster レプリケーションによる NDB Cluster バックアップ」を参照してください。)

- `--restore-meta, -m`

コマンド行形式	<code>--restore-meta</code>
型	Boolean
デフォルト値	FALSE

このオプションを指定すると、`ndb_restore` が NDB テーブルメタデータを出力します。

`ndb_restore` リストアプログラムを最初に実行するときは、メタデータもリストアする必要があります。つまり、データベーステーブルを再作成する必要があります。これは、`--restore-meta (-m)` オプションを指定して実行することで実行できます。メタデータのリストアは、単一のデータノードでのみ実行する必要があります。これは、クラスタ全体にリストアするのに十分です。

古いバージョンの NDB Cluster では、このオプションを使用してスキーマが復元されたテーブルは、新しいクラスタとは異なる数のデータノードがあった場合でも、元のクラスタと同じ数のパーティションを使用していました。NDB 8.0 では、メタデータを復元するときにはこれは問題ではなくなりました。ローカルデータマネージャスレッド

ドの数も元のクラスタ内のデータノードの数から変更されないかぎり、`ndb_restore` はターゲットクラスタのデフォルトのパーティション数を使用するようになりました。

NDB 8.0.16 以降でこのオプションを使用する場合は、`ndb_restore` がメタデータの復元を完了するまで `ndb_metadata_check=OFF` を設定して自動同期を無効にすることをお勧めします。その後、NDB ディクショナリに新しく作成されたオブジェクトを同期するために再度有効にすることができます。

注記

バックアップのリストアを開始する場合、クラスタには空のデータベースが必要です。(つまり、リストアを実行する前に、`--initial` を使用してデータノードを起動する必要があります。)

- `--restore-privilege-tables`

コマンド行形式	<code>--restore-privilege-tables</code>
非推奨	8.0.16-ndb-8.0.16
型	Boolean
デフォルト値	<code>FALSE</code> (オプションを使用しない場合)

`ndb_restore` は、NDB 7.6 以前で実装されている分散特権をサポートしていない、バージョン 8.0 より前の NDB Cluster のリリースで作成された分散 MySQL 特権テーブルをデフォルトで復元しません。このオプションを選択すると、`ndb_restore` によってリストアされます。

NDB 8.0.16 以降では、このようなテーブルはアクセス制御に使用されません。MySQL サーバーのアップグレードプロセスの一環として、サーバーはこれらのテーブルの InnoDB コピーをそれ自体に対してローカルに作成します。詳細は、[セクション 23.2.7 「NDB Cluster のアップグレードおよびダウングレード」](#) および [セクション 6.2.3 「付与テーブル」](#) を参照してください。

- `--rewrite-database=olddb,newdb`

コマンド行形式	<code>--rewrite-database=olddb,newdb</code>
型	文字列
デフォルト値	<code>none</code>

このオプションを指定すると、バックアップに使用された名前と異なる名前を持つデータベースにリストアできます。たとえば、バックアップが `products` という名前のデータベースから作成された場合は、このオプションを次のように使用して (必要となる可能性があるほかのオプションを省略しています)、含まれているデータを `inventory` という名前のデータベースにリストアできます。

```
shell> ndb_restore --rewrite-database=product,inventory
```

このオプションは、`ndb_restore` の単一呼び出しで複数回指定できます。このため、`--rewrite-database=db1,db2 --rewrite-database=db3,db4` を使用して、`db1` という名前のデータベースから `db2` という名前のデータベースに、および `db3` という名前のデータベースから `db4` という名前のデータベースに同時にリストアできます。複数の `--rewrite-database` 発生箇所の間にはほかの `ndb_restore` オプションを指定できます。

複数の `--rewrite-database` オプションで競合が発生した場合は、左から右に読んで最後に使用されている `--rewrite-database` オプションが有効となります。たとえば、`--rewrite-database=db1,db2 --rewrite-database=db1,db3` が使用された場合、`--rewrite-database=db1,db3` のみが有効となり、`--rewrite-database=db1,db2` は無視されます。複数のデータベースから単一データベースにリストアすることもできるため、`--rewrite-database=db1,db3 --rewrite-database=db2,db3` を指定すると、データベース `db1` および `db2` のすべてのテーブルおよびデータがデータベース `db3` にリストアされます。

重要

`--rewrite-database` を使用して複数のバックアップデータベースから単一ターゲットデータベースにリストアする場合、テーブル名またはその他のオブジェクト名間の競合は

チェックされず、行がリストアされる順序は保証されません。これは、そのような場合に行が上書きされて更新が失われることがあることを意味します。

- `--skip-broken-objects`

コマンド行形式	<code>--skip-broken-objects</code>
---------	------------------------------------

このオプションを指定すると、`ndb_restore` がネイティブ NDB バックアップを読み取るときに破損しているテーブルを無視して、残りのテーブル (破損していない) のリストアを続行します。現在のところ、`--skip-broken-objects` オプションは BLOB パーツテーブルが欠けている場合のみ機能します。

- `--skip-table-check, -s`

コマンド行形式	<code>--skip-table-check</code>
---------	---------------------------------

テーブルメタデータをリストアせずにデータをリストアできます。デフォルトでは、テーブルデータとテーブルスキーマの間に不一致が見つかった場合、`ndb_restore` はエラーで失敗します。このオプションはその動作をオーバーライドします。

`ndb_restore` を使用してデータをリストアするときのカラム定義内の不一致に関する制限の一部が緩和されています。それらのタイプの不一致のいずれかが発生しても、`ndb_restore` は以前のようにエラーで停止しなくなり、代わりにデータを受け入れてターゲットテーブルに挿入し、これが行われているという警告をユーザーに発行します。この動作は、`--skip-table-check` オプションまたは `--promote-attributes` オプションが使用されているかどうかにかかわらず、実行されます。カラム定義でのこれらの違いには次のタイプがあります。

- 異なる `COLUMN_FORMAT` 設定 (`FIXED`、`DYNAMIC`、`DEFAULT`)
- 異なる `STORAGE` 設定 (`MEMORY`、`DISK`)
- 異なるデフォルト値
- 異なる配布キー設定

- `--skip-unknown-objects`

コマンド行形式	<code>--skip-unknown-objects</code>
---------	-------------------------------------

このオプションを指定すると、`ndb_restore` がネイティブ NDB バックアップを読み取るときに、認識されないスキーマオブジェクトを無視します。これは、NDB 7.6 を実行しているクラスタから NDB Cluster 7.5 を実行しているクラスタに作成されたバックアップを復元するために使用できます。

- `--slice-id=#`

コマンド行形式	<code>--slice-id=#</code>
導入	8.0.20-ndb-8.0.20
型	Integer
デフォルト値	0
最小値	0
最大値	1023

スライスで復元する場合、これは復元するスライスの ID です。このオプションは常に `--num-slices` とともに使用され、その値は常に `--num-slices` の値より小さくする必要があります。

詳細は、このセクションの他の場所にある `--num-slices` の説明を参照してください。

- `--tab=dir_name, -T dir_name`

コマンド行形式	<code>--tab=dir_name</code>
型	ディレクトリ名

`--print-data` がダンプファイルをテーブルごとに作成し、それぞれに `tbl_name.txt` という名前を付けます。ファイルを保存すべきディレクトリへのパスを引数が必要です。現在のディレクトリの場合は `.` を使用します。

- `--verbose=#`

コマンド行形式	<code>--verbose=#</code>
型	数値
デフォルト値	1
最小値	0
最大値	255

出力の冗長性のレベルを設定します。最小値は 0 であり、最大値は 255 です。デフォルト値は 1 です。

エラー報告

`ndb_restore` は一時的および永続的エラーの両方を報告します。一時エラーの場合は、それらからリカバリできる場合があります、そのようなときには「[Restore successful, but encountered temporary error, please look at configuration](#)」と報告されます。

重要

`ndb_restore` を使用して循環レプリケーションで使用する NDB Cluster を初期化したあと、レプリカとして機能する SQL ノード上のバイナリログは自動的に作成されないため、手動で作成する必要があります。バイナリログを作成させるには、[START SLAVE](#) を実行する前に、その SQL ノードで [SHOW TABLES](#) ステートメントを発行します。これは NDB Cluster の既知の問題です。

23.4.23.1 異なるバージョンの NDB Cluster への NDB バックアップの復元

次の 2 つのセクションでは、ネイティブ NDB バックアップを、バックアップが作成されたバージョンとは異なるバージョンの NDB Cluster に復元する方法について説明します。

また、NDB ソフトウェアの別のバージョンを実行しているクラスタに NDB バックアップを復元しようとしたときに発生する可能性のあるその他の問題については、[セクション 23.2.7 「NDB Cluster のアップグレードおよびダウングレード」](#) に相談してください。

NDB 8.0 と以前のバージョンの NDB Cluster の間で、特定の状況に関連する可能性のあるその他の変更については、[NDB Cluster 8.0 の新機能](#) および [セクション 2.11.4 「MySQL 8.0 での変更」](#) を確認することもお勧めします。

NDB バックアップを以前のバージョンの NDB Cluster に復元

以前のバージョンには存在しない機能を使用しているため、新しいバージョンの NDB Cluster から以前のバージョンにバックアップを復元するときに問題が発生することがあります。これらの問題の一部を次に示します：

- NDB 8.0 で作成されたテーブルは、NDB 7.6 以前では使用できない `utf8mb4_ai_ci` 文字セットをデフォルトで使用するため、これらの以前のバージョンのいずれかから `ndb_restore` バイナリによって読み取ることはできません。このような場合は、バックアップを実行する前に古いバージョンでサポートされている文字セットを使用するように、`utf8mb4_ai_ci` を使用してテーブルを変更する必要があります。
- MySQL Server および NDB がテーブルメタデータを処理する方法が変更されたため、含まれている MySQL サーバーバイナリを使用して NDB 8.0.14 以降から作成または変更されたテーブルは、`ndb_restore` を使用して以前のバージョンの NDB Cluster に復元できません。このようなテーブルでは、古いバージョンの `mysqld` で認識されない `.sdi` ファイルが使用されます。

NDB 8.0.13 以前で作成され、NDB 8.0.14 以降へのアップグレード以降に変更されていない NDB 8.0.14 以降のテーブルで作成されたバックアップは、古いバージョンの NDB Cluster に復元可能である必要があります。

メタデータとテーブルデータを別々にリストアできるため、このような場合は、`mysqldump` を使用して作成したダンプからテーブルスキーマをリストアするか、必要な `CREATE TABLE` ステートメントを手動で実行してから、`--restore-data` オプションを指定した `ndb_restore` を使用してテーブルデータのみをインポートできます。

- NDB 8.0.22 以降で作成された暗号化バックアップは、NDB 8.0.21 以前の `ndb_restore` を使用して復元することはできません。
- NDB 8.0.18 より前では、`NDB_STORED_USER` 権限はサポートされていません。
- NDB Cluster 8.0.18 以降では最大 144 個のデータノードがサポートされますが、それ以前のバージョンでは最大 48 個のデータノードのみがサポートされます。この非互換性によって問題が発生する状況の詳細は、[元のノードより少ないノードへのリストア](#) を参照してください。

NDB Cluster の新しいバージョンへの NDB バックアップの復元

一般に、古いバージョンの NDB で `ndb_mgm` クライアントの `START BACKUP` コマンドを使用して作成されたバックアップは、新しいバージョンに付属の `ndb_restore` バイナリを使用している場合は、新しいバージョンに復元できます。(古いバージョンの `ndb_restore` を使用することもできますが、お薦めしません。) その他の潜在的な問題を次に示します:

- バックアップ (`--restore-meta` オプション) からメタデータをリストアする場合、`ndb_restore` は通常、取得したテーブルスキーマをバックアップの作成時とまったく同じように再現しようとします。

8.0.14 より前のバージョンの NDB で作成されたテーブルは、そのメタデータに `.frm` ファイルを使用します。これらのファイルは NDB 8.0.14 以降の `mysqld` で読み取ることができ、それに含まれる情報を使用して、MySQL データディクショナリで使用される `.sdi` ファイルを新しいバージョンで作成できます。

- 古いバックアップを NDB の新しいバージョンに復元する場合、ハッシュマップパーティション分割、より多くのハッシュマップバケット、読み取りバックアップ、さまざまなパーティションレイアウトなどの新しい機能を利用できないことがあります。このため、NDB が新しいスキーマ機能を利用できるように、`mysqldump` および `mysql` クライアントを使用して古いスキーマを復元することをお勧めします。
- 小数秒をサポートしていない古い時間型 (MySQL 5.6.4 および NDB 7.3.31 より前に使用されていた) を使用しているテーブルは、`ndb_restore` を使用して NDB 8.0 に復元できません。`CHECK TABLE` を使用してこのようなテーブルをチェックし、必要に応じて `mysql` クライアントで `REPAIR TABLE` を使用して、新しい時間カラム形式にアップグレードできます。これは、バックアップを取得する前に行う必要があります。詳しくは [セクション 2.11.5 「アップグレード用のインストールの準備」](#) をご覧ください。

また、`mysqldump` で作成されたダンプを使用して、このようなテーブルをリストアします。

- NDB 7.6 以前で作成された分散付与テーブルは NDB 8.0 ではサポートされていません。このようなテーブルは NDB 8.0 クラスタに復元できますが、アクセス制御には影響しません。

23.4.23.2 異なる数のデータノードへの復元

NDB バックアップから、バックアップが作成された元のものとは異なる数のデータノードを持つクラスタに復元できます。次の 2 つのセクションでは、ターゲットクラスタのデータノード数がバックアップのソースより少ないか多い場合についてそれぞれ説明します。

元のノードより少ないノードへのリストア

多数のノードが小さい数の偶数倍である場合は、元のノードより少ないデータノードを持つクラスタに復元できます。次の例では、4 つのデータノードを持つクラスタで作成したバックアップを、2 つのデータノードを持つクラスタに使用します。

1. 元のクラスタの管理サーバーはホスト `host10` 上にあります。元のクラスタには 4 つのデータノードがあり、ノード ID とホスト名は次のように管理サーバーの `config.ini` ファイルから抽出されたものです:

```
[ndbd]
NodeId=2
HostName=host2

[ndbd]
NodeId=4
HostName=host4

[ndbd]
NodeId=6
```

```
HostName=host6  
  
[ndbd]  
NodId=8  
HostName=host8
```

各データノードは、最初は `ndbmtid --ndb-connectstring=host10` または同等のものを使用して起動されたものとしてします。

2. 通常の方法でバックアップを実行します。これを行う方法については、[セクション23.5.8.2「NDB Cluster 管理クライアントを使用したバックアップの作成」](#)を参照してください。
3. 各データノードでバックアップによって作成されたファイルがここに一覧表示されます。ここで、**N** はノード ID、**B** はバックアップ ID です。

- [BACKUP-B-0.N.Data](#)
- [BACKUP-B.N.ctl](#)
- [BACKUP-B.N.log](#)

これらのファイルは、各データノードの `BackupDataDir /BACKUP/BACKUP-B` の下にあります。この例の残りの部分では、バックアップ ID は 1 であると想定しています。

これらのすべてのファイルを後で新しいデータノードにコピーできるようにします (ここでは、`ndb_restore` によってデータノードのローカルファイルシステム上でアクセスできます)。これらはすべて単一の場所にコピーするのが最も簡単です。これが完了したことを前提としています。

4. ターゲットクラスタの管理サーバーはホスト `host20` 上にあり、ターゲットには、`host20` 上の管理サーバー `config.ini` ファイルからのノード ID とホスト名が表示された 2 つのデータノードがあります:

```
[ndbd]  
NodId=3  
hostname=host3  
  
[ndbd]  
NodId=5  
hostname=host5
```

新しい (ターゲット) クラスタがクリーンデータノードファイルシステムで起動するように、`host3` および `host5` の各データノードプロセスは、`ndbmtid -c host20 --initial` または同等のものを使用して起動する必要があります。

5. 2 つの異なる 2 つのバックアップファイルのセットを各ターゲットデータノードにコピーします。この例では、バックアップファイルを元のクラスタのノード 2 および 4 からターゲットクラスタのノード 3 にコピーします。これらのファイルは次のとおりです:

- [BACKUP-1-0.2.Data](#)
- [BACKUP-1.2.ctl](#)
- [BACKUP-1.2.log](#)
- [BACKUP-1-0.6.Data](#)
- [BACKUP-1.6.ctl](#)
- [BACKUP-1.6.log](#)

次に、バックアップファイルをノード 6 および 8 からノード 5 にコピーします。これらのファイルを次のリストに示します:

- [BACKUP-1-0.4.Data](#)
- [BACKUP-1.4.ctl](#)
- [BACKUP-1.4.log](#)

- [BACKUP-1-0.8.Data](#)
- [BACKUP-1.8.ctl](#)
- [BACKUP-1.8.log](#)

この例の残りの部分では、それぞれのバックアップファイルが各ノード 3 および 5 のディレクトリ/[BACKUP-1](#) に保存されていることを前提としています。

- 2 つのターゲットデータノードのそれぞれで、両方のバックアップセットから復元する必要があります。まず、次に示すように `host3` で `ndb_restore` を起動して、ノード 2 および 4 からノード 3 にバックアップをリストアします:

```
shell> ndb_restore -c host20 --nodeid=2 --backupid=1 --restore-data --backup-path=/BACKUP-1
shell> ndb_restore -c host20 --nodeid=4 --backupid=1 --restore-data --backup-path=/BACKUP-1
```

次に、次のように `host5` で `ndb_restore` を起動して、ノード 6 および 8 からノード 5 にバックアップをリストアします:

```
shell> ndb_restore -c host20 --nodeid=6 --backupid=1 --restore-data --backup-path=/BACKUP-1
shell> ndb_restore -c host20 --nodeid=8 --backupid=1 --restore-data --backup-path=/BACKUP-1
```

元のノードより多くのノードへのリストア

特定の `ndb_restore` コマンドに指定されたノード ID は、元のバックアップ内のノードのノード ID であり、リストア先のデータノードのノード ID ではありません。このセクションで説明する方法を使用してバックアップを実行する場合、`ndb_restore` は管理サーバーに接続し、バックアップのリストア先となるクラスタ内のデータノードのリストを取得します。リストアされたデータは適宜分散されるため、バックアップの実行時にターゲットクラスタ内のノード数を把握または計算する必要はありません。

注記

ノードグループ当たりの LCP スレッドまたは LQH スレッドの合計数を変更する場合は、`mysqldump` を使用して作成されたバックアップからスキーマを再作成する必要があります。

- データのバックアップの作成。これを行うには、次のように、システムシェルから `ndb_mgm` クライアントの `START BACKUP` コマンドを呼び出します:

```
shell> ndb_mgm -e "START BACKUP 1"
```

これは、目的のバックアップ ID が 1 であることを前提としています。

- スキーマのバックアップを作成します。この手順は、ノードグループごとの LCP スレッドまたは LQH スレッドの合計数に変更された場合にのみ必要です。

```
shell> mysqldump --no-data --routines --events --triggers --databases > myschema.sql
```

重要

`ndb_mgm` を使用して NDB ネイティブバックアップを作成した後は、スキーマのバックアップを作成する前にスキーマを変更しないでください (作成した場合)。

- バックアップディレクトリを新しいクラスタにコピーします。たとえば、リストアするバックアップの ID が 1 で、`BackupDataDir = /backups/node_nodeid` の場合、このノードのバックアップへのパスは `/backups/node_1/BACKUP/BACKUP-1` です。このディレクトリ内には、次の 3 つのファイルがあります:

- [BACKUP-1-0.1.Data](#)
- [BACKUP-1.1.ctl](#)
- [BACKUP-1.1.log](#)

ディレクトリ全体を新しいノードにコピーする必要があります。

スキーマファイルを作成する必要がある場合は、これを `mysqld` で読み取ることができる SQL ノード上の場所にコピーします。

特定のノードからバックアップをリストアする必要はありません。

作成したばかりのバックアップからリストアするには、次のステップを実行します:

1. スキーマのリストア。

- `mysqldump` を使用して別のスキーマバックアップファイルを作成した場合は、次に示すように、`mysql` クライアントを使用してこのファイルをインポートします:

```
shell> mysql < myschema.sql
```

スキーマファイルをインポートする場合、`mysql` クライアントが MySQL サーバーに接続できるように、表示されている内容に加えて、`--user` および `--password` のオプション (場合によってはその他) を指定する必要があります。

- スキーマファイルを作成する必要がなかった場合は、次に示すように、`ndb_restore --restore-meta` (短縮形 `-m`) を使用してスキーマを再作成できます:

```
shell> ndb_restore --nodeid=1 --backupid=1 --restore-meta --backup-path=/backups/node_1/BACKUP/BACKUP-1
```

`ndb_restore` は管理サーバーに接続できる必要があります。これを可能にするには、必要に応じて `--ndb-connectstring` オプションを追加します。

- #### 2. データのリストア。
- これは、元のクラスター内のデータノードごとに、そのデータノード ID を使用するたびに 1 回実行する必要があります。最初に 4 つのデータノードがあったと仮定すると、必要な一連のコマンドは次のようになります:

```
ndb_restore --nodeid=1 --backupid=1 --restore-data --backup-path=/backups/node_1/BACKUP/BACKUP-1 --disable-indexes
ndb_restore --nodeid=2 --backupid=1 --restore-data --backup-path=/backups/node_2/BACKUP/BACKUP-1 --disable-indexes
ndb_restore --nodeid=3 --backupid=1 --restore-data --backup-path=/backups/node_3/BACKUP/BACKUP-1 --disable-indexes
ndb_restore --nodeid=4 --backupid=1 --restore-data --backup-path=/backups/node_4/BACKUP/BACKUP-1 --disable-indexes
```

これらはパラレルで実行できます。

必要に応じて、`--ndb-connectstring` オプションを追加してください。

- #### 3. インデックスの再構築。
- これらは、前述のコマンドで使用された `--disable-indexes` オプションによって無効化されました。インデックスを再作成すると、リストアがすべての時点で一貫していないことによるエラーを回避できます。インデックスを再構築すると、場合によってはパフォーマンスが向上することもあります。インデックスを再構築するには、単一ノードで次のコマンドを 1 回実行します:

```
shell> ndb_restore --nodeid=1 --backupid=1 --backup-path=/backups/node_1/BACKUP/BACKUP-1 --rebuild-indexes
```

前述のように、`ndb_restore` が管理サーバーに接続できるように、`--ndb-connectstring` オプションを追加する必要がある場合があります。

23.4.23.3 パラレルで作成されたバックアップからのリストア

NDB Cluster 8.0.16 以降では、複数の LDM を持つ `ndbmtl` を使用して、各データノードで並列バックアップを取ることができます (セクション 23.5.8.5 「並列データノードを使用した NDB バックアップの作成」を参照)。次の 2 つのセクションでは、この方法で作成されたバックアップをリストアする方法について説明します。

パラレルバックアップのリストア

並列バックアップの復元には、NDB Cluster 配布バージョン 8.0.16 以降からの `ndb_restore` バイナリが必要です。このプロセスは、`ndb_restore` プログラムの説明の下の一般的な使用方法のセクションで概説されているプロセスとは大きく異なりません。ここで示すように、`ndb_restore` の 2 回の実行で構成されます:

```
shell> ndb_restore -n 1 -b 1 -m --backup-path=path/to/backup_dir/BACKUP/BACKUP-backup_id
```

```
shell> ndb_restore -n 1 -b 1 -r --backup-path=path/to/backup_dir/BACKUP/BACKUP-backup_id
```

`backup_id` は、リストアするバックアップの ID です。一般に、追加の特別な引数は必要ありません。`ndb_restore` は常に、`--backup-path` オプションで指定されたディレクトリの下に平行サブディレクトリが存在するかどうかをチェックし、メタデータを (シリアルに) リストアしてから、テーブルデータを (平行に) リストアします。

平行バックアップのシリアルリストア

並列性を使用してデータノード上で作成されたバックアップをシリアル方式で復元できます。これを行うには、メインバックアップディレクトリの下各 LDM によって作成されたサブディレクトリを指す `--backup-path` を使用して `ndb_restore` を起動し、メタデータをリストアするサブディレクトリのいずれかを一度起動します (各サブディレクトリにメタデータの完全なコピーが含まれているため、どのサブディレクトリにも関係ありません)。その後、各サブディレクトリを順に起動してデータをリストアします。4 つの LDM で作成されたバックアップ ID 100 を持つバックアップをリストアし、`BackupDataDir` が `/opt` であるとします。この場合、メタデータをリストアするには、次のように `ndb_restore` を起動します:

```
shell> ndb_restore -n 1 -b 1 -m --backup-path=opt/BACKUP/BACKUP-100/BACKUP-100-PART-1-OF-4
```

テーブルデータをリストアするには、次に示すように、いずれかのサブディレクトリを順に使用して、`ndb_restore` を 4 回実行します:

```
shell> ndb_restore -n 1 -b 1 -r --backup-path=opt/BACKUP/BACKUP-100/BACKUP-100-PART-1-OF-4
shell> ndb_restore -n 1 -b 1 -r --backup-path=opt/BACKUP/BACKUP-100/BACKUP-100-PART-2-OF-4
shell> ndb_restore -n 1 -b 1 -r --backup-path=opt/BACKUP/BACKUP-100/BACKUP-100-PART-3-OF-4
shell> ndb_restore -n 1 -b 1 -r --backup-path=opt/BACKUP/BACKUP-100/BACKUP-100-PART-4-OF-4
```

以前のバージョンの NDB Cluster ソフトウェアで提供されていた `ndb_restore` バイナリを使用して、並列バックアップをサポートしていない古いバージョンの NDB Cluster (NDB 8.0.16 より前) に並列バックアップを復元する場合と同じ方法を使用できます。

23.4.24 ndb_select_all — NDB テーブルの行の出力

`ndb_select_all` は、NDB テーブルのすべての行を `stdout` に出力します。

使用法

```
ndb_select_all -c connection_string tbl_name -d db_name [> file_name]
```

次のテーブルに、NDB Cluster ネイティブバックアップリストアプログラム `ndb_select_all` に固有のオプションを示します。追加説明が表のあとにあります。ほとんどの NDB Cluster プログラム (`ndb_select_all` を含む) に共通のオプションについては、[セクション 23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#) を参照してください。

表 23.43 プログラムで使用されるコマンドライン・オプション `ndb_select_all`

形式	説明	追加、非推奨、または削除された
<code>--database=dbname,</code> <code>-d</code>	テーブルが見つかったデータベースの名前	(MySQL に基づくすべての NDB リリースでサポート 8.0)
<code>--parallelism=#,</code> <code>-p</code>	並列性の度合い	(MySQL に基づくすべての NDB リリースでサポート 8.0)
<code>--lock=#,</code> <code>-l</code>	ロックタイプ	(MySQL に基づくすべての NDB リリースでサポート 8.0)
<code>--order=index,</code> <code>-o</code>	この名前を持つインデックスに従って結果セットをソート	(MySQL に基づくすべての NDB リリースでサポート 8.0)
<code>--descending,</code> <code>-z</code>	結果セットを降順でソート (<code>--order</code> が必要)	(MySQL に基づくすべての NDB リリースでサポート 8.0)

形式	説明	追加、非推奨、または削除された
<code>--header,</code> <code>-h</code>	ヘッダーを出力します (出力でヘッダーを無効にするには 0 FALSE を設定します)	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--useHexFormat,</code> <code>-x</code>	数値を 16 進形式で出力します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--delimiter=char,</code> <code>-D</code>	カラムデリミタの設定	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--disk</code>	ディスク参照を出力します (インデックスが設定されていないカラムを持つディスクデータテーブルでのみ役に立ちます)	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--rowid</code>	印刷行 ID	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--gci</code>	出力に GCI を含めます	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--gci64</code>	出力に GCI および行エポックを含めます	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--tupscan,</code> <code>-t</code>	TUP 順序でスキャンします	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--nodata</code>	テーブルカラムデータを出力しません	(MySQLに基づくすべてのNDBリリースでサポート 8.0)

- `--database=dbname, -d dbname`

テーブルが見つかるデータベースの名前。デフォルト値は `TEST_DB` です。

- `parallelism=#, -p #`

並列の度合いを指定します。

- `--lock=lock_type, -l lock_type`

テーブルを読み取るときにロックを適用します。 `lock_type` に指定できる値を次に示します。

- 0: 読み取りロック
- 1: ホールド付きの読み取りロック
- 2: 排他的読み取りロック

このオプションにはデフォルト値はありません。

- `--order=index_name, -o index_name`

`index_name` という名前のインデックスの順序で出力します。

注記

これは、カラムではなくインデックスの名前です。インデックスは、作成時に明示的に名前が付けられている必要があります。

- `--descending, -z`

出力を降順でソートします。このオプションは、`-o (--order)` オプションを指定した場合にのみ使用できます。

- `--header=FALSE`

出力からカラムヘッダーを除外します。

- `--useHexFormat -x`

すべての数値を 16 進数形式で表示します。これは文字列または日時値に含まれている数値の出力には影響しません。

- `--delimiter=character, -D character`

`character` をカラム区切り文字として使用します。この区切り文字で区切られるのはテーブルデータカラムのみです。

デフォルトの区切り文字はタブ文字です。

- `--disk`

出力にディスクリファレンスカラムを追加します。このカラムが空でないのは、インデックスが設定されていないカラムを持つディスクデータテーブルの場合だけです。

- `--rowid`

行が保存されるフラグメントに関する情報を示す `ROWID` カラムを追加します。

- `--gci`

各行が最後に更新されたグローバルチェックポイントを示す `GCI` カラムを出力に追加します。チェックポイントに関する詳細は、[セクション23.1「NDB Cluster の概要」](#)および[セクション23.5.3.2「NDB Cluster ログイベント」](#)を参照してください。

- `--gci64`

各行が最後に更新されたグローバルチェックポイントおよびこの更新が発生したエポックの数を示す `ROW$GCI64` カラムを出力に追加します。

- `--tupscan, -t`

タプルの順序でテーブルをスキャンします。

- `--nodata`

テーブルデータを省きます。

出力例

MySQL `SELECT` ステートメントからの出力:

```
mysql> SELECT * FROM ctest1.fish;
+-----+
| id | name |
+-----+
| 3 | shark |
| 6 | puffer |
| 2 | tuna |
| 4 | manta ray |
| 5 | grouper |
| 1 | guppy |
+-----+
6 rows in set (0.04 sec)
```

同等の `ndb_select_all` 呼び出しからの出力:

```
shell> ./ndb_select_all -c localhost fish -d ctest1
id name
3 [shark]
6 [puffer]
```

```

2 [tuna]
4 [manta ray]
5 [grouper]
1 [guppy]
6 rows returned

```

```
NDBT_ProgramExit: 0 - OK
```

`ndb_select_all` の出力では、すべての文字列値が大カッコ ([...]) で囲まれています。別の例として、次に示すように作成および移入されたテーブルを考えてみます:

```

CREATE TABLE dogs (
  id INT(11) NOT NULL AUTO_INCREMENT,
  name VARCHAR(25) NOT NULL,
  breed VARCHAR(50) NOT NULL,
  PRIMARY KEY pk (id),
  KEY ix (name)
)
TABLESPACE ts STORAGE DISK
ENGINE=NDBCLUSTER;

INSERT INTO dogs VALUES
('Lassie', 'collie'),
('Scooby-Doo', 'Great Dane'),
('Rin-Tin-Tin', 'Alsatian'),
('Rosscoe', 'Mutt');

```

ほかのいくつかの `ndb_select_all` オプションの使用方法を次に示します。

```

shell> ./ndb_select_all -d ctest1 dogs -o ix -z --gci --disk
GCI id name breed DISK_REF
834461 2 [Scooby-Doo] [Great Dane] [ m_file_no: 0 m_page: 98 m_page_idx: 0 ]
834878 4 [Rosscoe] [Mutt] [ m_file_no: 0 m_page: 98 m_page_idx: 16 ]
834463 3 [Rin-Tin-Tin] [Alsatian] [ m_file_no: 0 m_page: 34 m_page_idx: 0 ]
835657 1 [Lassie] [Collie] [ m_file_no: 0 m_page: 66 m_page_idx: 0 ]
4 rows returned

NDBT_ProgramExit: 0 - OK

```

23.4.25 ndb_select_count — NDB テーブルの行数の出力

`ndb_select_count` は、1 つ以上の NDB テーブルの行数を出力します。単一テーブルでは、結果は MySQL ステートメント `SELECT COUNT(*) FROM tbl_name` を使用して取得される結果と同じになります。

使用法

```
ndb_select_count [-c connection_string] -ddb_name tbl_name[, tbl_name2[, ...]]
```

次のテーブルに、NDB Cluster ネイティブバックアップリストアッププログラム `ndb_select_count` に固有のオプションを示します。追加説明が表のあとにあります。ほとんどの NDB Cluster プログラム (`ndb_select_count` を含む) に共通のオプションについては、[セクション 23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#) を参照してください。

表 23.44 プログラムで使用されるコマンドライン・オプション `ndb_select_count`

形式	説明	追加、非推奨、または削除された
<code>--database=dbname,</code> <code>-d</code>	テーブルが見つかったデータベースの名前	(MySQL に基づくすべての NDB リリースでサポート 8.0)
<code>--parallelism=#,</code> <code>-p</code>	並列性の度合い	(MySQL に基づくすべての NDB リリースでサポート 8.0)
<code>--lock=#,</code>	ロックタイプ	(MySQL に基づくすべての NDB リリースでサポート 8.0)

形式	説明	追加、非推奨、または削除された
-l		

同じデータベース内の複数のテーブルの行数を取得するには、「出力例」に示されているように、このコマンドを呼び出すときにテーブル名をスペースで区切ってリストします。

出力例

```
shell> ./ndb_select_count -c localhost -d ctest1 fish dogs
6 records in table fish
4 records in table dogs

NDBT_ProgramExit: 0 - OK
```

23.4.26 ndb_setup.py — NDB Cluster 用ブラウザベースの Auto-Installer の起動 (非推奨)

注記

この機能は非推奨になっており、使用しないようにしてください。NDB Cluster の将来のバージョンで削除される可能性があります。

`ndb_setup.py` は NDB Cluster Auto-Installer を起動し、デフォルトの Web ブラウザでインストーラの Start ページを開きます。

重要

このプログラムは、`mysql`、システム `root` またはその他の管理アカウントではなく、通常のユーザーとして起動することを目的としています。

このセクションでは、コマンド行ツールのみ使用方法およびプログラムオプションについて説明します。`ndb_setup.py` を呼び出すと開始される Auto-Installer GUI の使用方法については、[The NDB Cluster Auto-Installer \(NDB 7.5\) \(No longer supported\)](#) を参照してください。

使用法

すべてのプラットフォーム:

```
ndb_setup.py [options]
```

または、Windows プラットフォームのみ:

```
setup.bat [options]
```

次のテーブルには、NDB Cluster のインストールおよび構成プログラム `ndb_setup.py` でサポートされるすべてのオプションが含まれています。追加説明が表のあとにあります。

表 23.45 プログラムで使用されるコマンドライン・オプション `ndb_setup.py`

形式	説明	追加、非推奨、または削除された
<code>--browser-start-page=filename,</code> <code>-s</code>	起動時に web ブラウザが開くページ	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--ca-certs-file=filename,</code> <code>-a</code>	サーバーへの接続が許可されているクライアント証明書を含むファイル	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--cert-file=filename,</code>	サーバーを識別する X509 証明書を含むファイル	(MySQLに基づくすべてのNDBリリースでサポート 8.0)

形式	説明	追加、非推奨、または削除された
-c		
--debug-level=level,		
-d	Python ロギングモジュールのデバッグレベル。DEBUG、INFO、WARNING (デフォルト)、ERROR または CRITICAL のいずれか	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
--help,	ヘルプメッセージを出力します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
-h		
--key-file=file,	プライベートキーが含まれているファイルを指定します (--cert-file に含まれていない場合)	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
-k		
--no-browser,	ブラウザで開始ページを開かず、単にツールを起動	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
-n		
--port=#,	web サーバーで使用されるポートを指定	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
-p		
--server-log-file=file,	リクエストをこのファイルに記録します。かわりに、' 'を使用して stderr に強制的にロギング	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
-o		
--server-name=name,	接続先のサーバーの名前	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
-N		
--use-http,	非暗号化 (HTTP) クライアント/サーバー接続の使用	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
-H		
--use-https,	暗号化された (HTTPS) クライアント/サーバー接続の使用	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
-S		

- --browser-start-page=file, -s

コマンド行形式	--browser-start-page=filename
型	文字列
デフォルト値	index.html

ブラウザでインストールおよび構成の「Start」ページとして開くファイルを指定します。デフォルトは [index.html](#) です。

- --ca-certs-file=file, -a

コマンド行形式	--ca-certs-file=filename
型	ファイル名
デフォルト値	[none]

サーバーに接続できるクライアント証明書の一覧が含まれているファイルを指定します。デフォルトは空の文字列であり、クライアント認証が使用されないことを意味します。

- --cert-file=file, -c

コマンド行形式	--cert-file=filename
型	ファイル名
デフォルト値	/usr/share/mysql/mcc/cfg.pem

サーバーを識別する X509 証明書が含まれているファイルを指定します。証明書は自己署名できます。デフォルトは `cfg.pem` です。

- `--debug-level=level, -d`

コマンド行形式	<code>--debug-level=level</code>
型	列挙
デフォルト値	WARNING
有効な値	WARNING DEBUG INFO ERROR CRITICAL

Python ロギングモジュールデバッグレベルを設定します。これは、`DEBUG`、`INFO`、`WARNING`、`ERROR`、または `CRITICAL` のいずれかです。 `WARNING` がデフォルトです。

- `--help, -h`

コマンド行形式	<code>--help</code>
---------	---------------------

ヘルプメッセージを出力します。

- `--key-file=file, -d`

コマンド行形式	<code>--key-file=file</code>
型	ファイル名
デフォルト値	[none]

X509 証明書ファイル (`--cert-file`) にプライベートキーが含まれていない場合、これが含まれているファイルを指定します。デフォルトは空の文字列であり、そのようなファイルが使用されないことを意味します。

- `--no-browser, -n`

コマンド行形式	<code>--no-browser</code>
---------	---------------------------

インストールおよび構成ツールを開始しますが、ブラウザで「Start」ページを開きません。

- `--port=#, -p`

コマンド行形式	<code>--port=#</code>
型	数値
デフォルト値	8081
最小値	1
最大値	65535

Web サーバーによって使用されるポートを設定します。デフォルトは 8081 です。

- `--server-log-file=file, -o`

コマンド行形式	<code>--server-log-file=file</code>
型	ファイル名

デフォルト値	ndb_setup.log
有効な値	ndb_setup.log - (stderr にログを記録します)

要求のログをこのファイルに記録します。デフォルトは `ndb_setup.log` です。ファイルではなく `stderr` にロギングするように指定するには、ファイル名に `-` (ダッシュ文字) を使用します。

- `--server-name=host, -N`

コマンド行形式	<code>--server-name=name</code>
型	文字列
デフォルト値	localhost

ブラウザが接続するときに使用するホスト名または IP アドレスを指定します。デフォルトは `localhost` です。

- `--use-http, -H`

コマンド行形式	<code>--use-http</code>
---------	-------------------------

ブラウザで HTTP を使用してサーバーに接続します。つまり、接続は暗号化されず、どのような方法でも保護されません。

- `--use-https, -S`

コマンド行形式	<code>--use-https</code>
---------	--------------------------

ブラウザがサーバーに対してセキュア (HTTPS) 接続を使用します。

23.4.27 ndb_show_tables — NDB テーブルのリストの表示

`ndb_show_tables` は、クラスタ内のすべての NDB データベースオブジェクトのリストを表示します。デフォルトでは、これにはユーザーが作成したテーブルと NDB システムテーブルの両方だけでなく、NDB 固有のインデックス、内部トリガー、NDB Cluster ディスクデータオブジェクトも含まれます。

次のテーブルに、NDB Cluster ネイティブバックアップリストアッププログラム `ndb_show_tables` に固有のオプションを示します。追加説明が表のあとにあります。ほとんどの NDB Cluster プログラム (`ndb_show_tables` を含む) に共通のオプションについては、[セクション23.4.32「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#)を参照してください。

表 23.46 プログラムで使用されるコマンドライン・オプション `ndb_show_tables`

形式	説明	追加、非推奨、または削除された
<code>--database=string,</code> <code>-d</code>	テーブルが見つかったデータベースを指定します。データベース名の後にはテーブル名が続く必要があります	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--loops=#,</code> <code>-l</code>	出力を繰り返す回数	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--parsable,</code> <code>-p</code>	MySQL LOAD DATA ステートメントに適した出力を返します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--show-temp-status</code>	テーブル一時フラグを表示します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--type=#,</code> <code>-t</code>	このタイプのオブジェクトに出力を制限します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)

形式	説明	追加、非推奨、または削除された
<code>--unqualified,</code> <code>-u</code>	テーブル名を修飾しません	(MySQLに基づくすべてのNDBリリースでサポート 8.0)

使用法

```
ndb_show_tables [-c connection_string]
```

- `--database, -d`

目的のテーブルが見つかるデータベースの名前を指定します。このオプションを指定する場合、テーブルの名前はデータベース名の後に指定する必要があります。

このオプションが指定されておらず、`TEST_DB` データベースにテーブルが見つからない場合、`ndb_show_tables` は警告を発行します。

- `--loops, -l`

ユーティリティを実行すべき回数を指定します。このオプションを指定しない場合はこれは 1 ですが、このオプションを使用する場合はそれに整数の引数を指定する必要があります。

- `--parsable, -p`

このオプションを使用すると、出力は `LOAD DATA` での使用に適した形式になります。

- `--show-temp-status`

指定した場合は、一時テーブルが表示されます。

- `--type, -t`

次に示す整数型コードで指定された 1 種類のオブジェクトに出力を制限するために使用できます。

- 1: システムテーブル
- 2: ユーザーが作成したテーブル
- 3: 一意のハッシュインデックス

その他の値を指定すると、すべての `NDB` データベースオブジェクトが一覧されます (デフォルト)。

- `--unqualified, -u`

これを指定すると、修飾されていないオブジェクト名が表示されます。

注記

MySQL からアクセスできるのは、ユーザーが作成した「NDB Cluster」テーブルのみです。`SYSTAB_0` などのシステムテーブルは、`mysql` には表示されません。ただし、`ndb_select_all` などの `NDB` API アプリケーションを使用してシステムテーブルの内容を調べることができます ([セクション 23.4.24 「ndb_select_all — NDB テーブルの行の出力」](#)を参照してください)。

`NDB 8.0.20` より前では、このプログラムは、`NDBT` テストライブラリへの不要な依存関係のため、実行の完了時に `NDBT_ProgramExit - status` を出力しました。この依存関係は削除され、余分な出力がなくなりました。

23.4.28 ndb_size.pl — NDBCLUSTER サイズ要件エスティメータ

これは、`NDBCLUSTER` ストレージエンジンを使用するように変更された場合に、MySQL データベースによって要求される空き領域の大きさを見積もるために使用できる Perl スクリプトです。このセクションで説明するほかのユーティリティとは異なり、`NDB Cluster` にアクセスする必要はありません (実際には、これを行う理由はありません)。ただし、テストするデータベースがある MySQL サーバーにアクセスする必要があります。

要件

- 実行されている MySQL サーバー。サーバーインスタンスは NDB Cluster をサポートする必要はありません。
- Perl の有効なインストール。
- DBI モジュール (まだ Perl インストールの一部でない場合は、CPAN から取得できます) (多くの Linux およびその他のオペレーティングシステム配布では、このライブラリの独自のパッケージが提供されています)。
- 必要な権限を持つ MySQL ユーザーアカウント。既存のアカウントを使用したくない場合は、[GRANT USAGE ON db_name.*](#) を使用して作成するだけでこの用途には十分です。ここで、`db_name` は検査するデータベースの名前です。

`ndb_size.pl` は、`storage/ndb/tools` 内の MySQL ソースでも見つかります。

次のテーブルに、NDB Cluster プログラム `ndb_size.pl` に固有のオプションを示します。追加説明が表のあとにあります。ほとんどの NDB Cluster プログラム (`ndb_size.pl` を含む) に共通のオプションについては、[セクション 23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#) を参照してください。

表 23.47 プログラムで使用されるコマンドライン・オプション `ndb_size.pl`

形式	説明	追加、非推奨、または削除された
<code>--database=dbname</code>	検査するデータベース。カンマ区切りのリスト。デフォルトは ALL (サーバー上のすべてのデータベースを使用) です	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--hostname[:port]</code>	ホストおよびオプションポートを <code>host[:port]</code> として指定します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--socket=file_name</code>	接続先のソケットの指定	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--user=string</code>	MySQL ユーザー名の指定	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--password=string</code>	MySQL ユーザーパスワードの指定	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--format=string</code>	出力形式 (テキストまたは HTML) を設定します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--excludetables=tbl_list</code>	カンマ区切りリストのテーブルをスキップ	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--excludedbs=db_list</code>	カンマ区切りリストのデータベースをスキップ	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--savequeries=file</code>	指定したファイルにデータベースのすべてのクエリーを保存	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--loadqueries=file</code>	指定されたファイルからすべてのクエリーをロードします。データベースには接続しません	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--real_table_name=table</code>	一意のインデックスサイズ計算を処理するテーブルを指定	(MySQLに基づくすべてのNDBリリースでサポート 8.0)

使用方法

```
perl ndb_size.pl [--database={db_name|ALL}] [--hostname=host[:port]] [--socket=socket] \
  [--user=user] [--password=password] \
  [--help|-h] [--format={html|text}] \
  [--loadqueries=file_name] [--savequeries=file_name]
```

デフォルトでは、このユーティリティーはサーバー上のすべてのデータベースを分析しようとします。 `--database` オプションを使用すると、単一データベースを指定できます。デフォルト動作を明示的に指定するには、データベース

の名前に **ALL** を使用します。1 つ以上のデータベースを除外するには、**--excludedbs** オプションを使用して、スキップするデータベース名のカンマ区切りのリストを指定します。同様に、特定のテーブルをスキップするには、オプションの **--excludetables** オプションに続けて、それらの名前をカンマで区切ってリストします。ホスト名を指定するには、**--hostname** を使用します。デフォルトは **localhost** です。ホストに加えてポートを指定するには、**--hostname** の値に **host:port** 形式を使用します。デフォルトのポート番号は 3306 です。必要に応じて、ソケットも指定できます。デフォルトは **/var/lib/mysql.sock** です。MySQL のユーザー名およびパスワードは、表示されている対応するオプションで指定できます。出力の形式を制御するには、**--format** オプションを使用します。これには、値 **html** または **text** を指定でき、**text** がデフォルトです。テキスト出力の例を次に示します。

```
shell> ndb_size.pl --database=test --socket=/tmp/mysql.sock
ndb_size.pl report for database: 'test' (1 tables)
-----
Connected to: DBI:mysql:host=localhost:mysql_socket=/tmp/mysql.sock

Including information for versions: 4.1, 5.0, 5.1

test.t1
-----

DataMemory for Columns (* means var sized DataMemory):
  Column Name      Type  VarSized  Key  4.1  5.0  5.1
  -----
  HIDDEN_NDB_PKEY  bigint  PRI      8   8   8
  c2  varchar(50)   Y      52  52  4*
  c1  int(11)         4      4   4
  -----
Fixed Size Columns DM/Row          64  64  12
VarSize Columns DM/Row            0   0   4

DataMemory for Indexes:
  Index Name      Type    4.1   5.0   5.1
  -----
  PRIMARY        BTREE   16    16    16
  -----
Total Index DM/Row          16   16   16

IndexMemory for Indexes:
  Index Name      4.1   5.0   5.1
  -----
  PRIMARY        33    16    16
  -----
Indexes IM/Row            33   16   16

Summary (for THIS table):
  4.1   5.0   5.1
Fixed Overhead DM/Row    12    12    16
NULL Bytes/Row          4      4      4
DataMemory/Row         96    96    48
(Includes overhead, bitmap and indexes)

VarSize Overhead DM/Row    0      0      8
VarSize NULL Bytes/Row    0      0      4
Avg VarSize DM/Row        0      0     16

No. Rows      0      0      0

Rows/32kb DM Page    340    340    680
Fixedsize DataMemory (KB)    0      0      0

Rows/32kb VarSize DM Page    0      0    2040
VarSize DataMemory (KB)    0      0      0

Rows/8kb IM Page     248    512    512
IndexMemory (KB)      0      0      0

Parameter Minimum Requirements
-----
* indicates greater than default

  Parameter  Default  4.1   5.0   5.1
  -----
  DataMemory (KB)  81920    0     0     0
  NoOfOrderedIndexes  128     1     1     1
  NoOfTables      128     1     1     1
  IndexMemory (KB) 18432    0     0     0
  NoOfUniqueHashIndexes  64     0     0     0
  NoOfAttributes  1000    3     3     3
```

```
NoOfTriggers      768      5      5      5
```

デバッグ目的で、このスクリプトによって実行されるクエリーを含む Perl 配列は、を使用して指定されたファイルから `--savequeries` を使用してファイルに保存できます。スクリプトの実行中に読み取られるこのような配列を含むファイルは、`--loadqueries` を使用して指定できます。これらのオプションにはデフォルト値はありません。

出力を HTML 形式で生成するには、次に示すように `--format` オプションを使用して出力をファイルにリダイレクトします。

```
shell> ndb_size.pl --database=test --socket=/tmp/mysql.sock --format=html > ndb_size.html
```

(リダイレクトしない場合、出力は `stdout` に送られます)。

このスクリプトの出力には次の情報が含まれています。

- 分析されるテーブルに対応するために必要な `DataMemory`、`IndexMemory`、`MaxNoOfTables`、`MaxNoOfAttributes`、`MaxNoOfOrderedIndexes` および `MaxNoOfTriggers` 構成パラメータの最小値。
- データベースに定義されているすべてのテーブル、属性、順序付けされたインデックス、および一意のハッシュインデックスのメモリー要件。
- テーブルおよびテーブル行ごとに必要な `IndexMemory` および `DataMemory`。

23.4.29 ndb_top — NDB スレッドの CPU 使用率情報の表示

`ndb_top` は、NDB Cluster データノード上の NDB スレッドによる CPU 使用率に関する実行中の情報を端末に表示します。出力では、各スレッドは 2 行で表され、最初にシステム統計が表示され、2 行目にスレッドの測定統計が表示されます。

`ndb_top` は、MySQL NDB Cluster 7.6.3 以降で使用できます。

使用法

```
ndb_top [-h hostname] [-t port] [-u user] [-p pass] [-n node_id]
```

`ndb_top` は、クラスタの SQL ノードとして実行中の MySQL Server に接続します。デフォルトでは、`localhost` およびポート 3306 で実行されている `mysqld` に、パスワードを指定せずに MySQL `root` ユーザーとして接続しようとします。デフォルトのホストおよびポートは、それぞれ `--host (-h)` および `--port (-t)` を使用してオーバーライドできます。MySQL のユーザーおよびパスワードを指定するには、`--user (-u)` および `--passwd (-p)` オプションを使用します。このユーザーは、`ndbinfo` データベース内のテーブルを読み取ることができる必要があります (`ndb_top` では、`ndbinfo.cputat` および関連するテーブルの情報が使用されます)。

MySQL のユーザーアカウントおよびパスワードの詳細は、[セクション6.2「アクセス制御とアカウント管理」](#) を参照してください。

出力はプレーンテキストまたは ASCII グラフとして使用できます。これは、それぞれ `--text (-x)` および `--graph (-g)` オプションを使用して指定できます。これら 2 つの表示モードは同じ情報を提供し、同時に使用できます。少なくとも 1 つの表示モードを使用する必要があります。

グラフの色表示は、デフォルトでサポートされ、有効になっています (`--color` または `-c` オプション)。カラーサポートが有効な場合、グラフ表示には OS ユーザー時間が青、OS システム時間が緑、アイドル時間が空白で表示されます。測定負荷の場合、実行時間には青、送信時間には黄色、送信バッファのフル待機に費やされた時間には赤、アイドル時間には空白が使用されます。グラフ表示に表示されるパーセンテージは、アイドル状態でないすべてのスレッドのパーセンテージの合計です。現在、色は構成できません。かわりに、`--skip-color` を使用してグレースケールを使用できます。

ソートされたビュー (`--sort, -r`) は、測定された負荷の最大および OS によって報告された負荷に基づきます。これらの表示は、`--measured-load (-m)` および `--os-load (-o)` オプションを使用して有効または無効にできます。少なくとも 1 つの荷重の表示を有効にする必要があります。

プログラムは、`--node-id (-n)` オプションで指定されたノード ID を持つデータノードから統計情報を取得しようとします。指定されていない場合、これは 1 です。`ndb_top` は、他のタイプのノードに関する情報を提供できません。

ビュー自体が端末ウィンドウの高さと幅に合わせて調整されます。サポートされる最小幅は 76 文字です。

一度起動すると、`ndb_top` は強制的に終了するまで継続して実行されます。Ctrl-C を使用してプログラムを終了できます。表示は毎秒 1 回更新されます。異なる遅延間隔を設定するには、`--sleep-time (-s)` を使用します。

注記

`ndb_top` は、macOS、Linux および Solaris で使用できます。現在、Windows プラットフォームではサポートされていません。

次のテーブルに、NDB Cluster プログラム `ndb_top` に固有のすべてのオプションを示します。追加説明が表のあとにあります。

表 23.48 プログラムで使用されるコマンドライン・オプション `ndb_top`

形式	説明	追加、非推奨、または削除された
<code>--color,</code> <code>-c</code>	ASCII グラフを色で表示します。無効にするには <code>--skip-colors</code> を使用	(MySQL に基づくすべての NDB リリースでサポート 8.0)
<code>--graph,</code> <code>-g</code>	グラフを使用してデータを表示します。 <code>--skip-graphs</code> を使用して無効にします	(MySQL に基づくすべての NDB リリースでサポート 8.0)
<code>--help,</code> <code>-?</code>	プログラム使用状況情報の表示	(MySQL に基づくすべての NDB リリースでサポート 8.0)
<code>--host[=name],</code> <code>-h</code>	接続先の MySQL Server のホスト名または IP アドレス	(MySQL に基づくすべての NDB リリースでサポート 8.0)
<code>--measured-load,</code> <code>-m</code>	測定された負荷をスレッド別に表示	(MySQL に基づくすべての NDB リリースでサポート 8.0)
<code>--node-id[=#],</code> <code>-n</code>	このノード ID を持つウォッチノード	(MySQL に基づくすべての NDB リリースでサポート 8.0)
<code>--os-load,</code> <code>-o</code>	オペレーティングシステムによって測定された負荷の表示	(MySQL に基づくすべての NDB リリースでサポート 8.0)
<code>--password[=password],</code> <code>-p</code>	このパスワードを使用して接続	(MySQL に基づくすべての NDB リリースでサポート 8.0)
<code>--port[=#],</code> <code>-P (>=7.6.6)</code>	MySQL Server への接続時に使用するポート番号	(MySQL に基づくすべての NDB リリースでサポート 8.0)
<code>--sleep-time[=seconds],</code> <code>-s</code>	表示リフレッシュ間の待機時間 (秒)	(MySQL に基づくすべての NDB リリースでサポート 8.0)
<code>--socket,</code> <code>-S</code>	接続で使用するソケットファイル	(MySQL に基づくすべての NDB リリースでサポート 8.0)
<code>--sort,</code> <code>-r</code>	スレッドを使用状況でソートします。 <code>--skip-sort</code> を使用して無効にします	(MySQL に基づくすべての NDB リリースでサポート 8.0)
<code>--text,</code> <code>-t (>=7.6.6)</code>	テキストを使用したデータの表示	(MySQL に基づくすべての NDB リリースでサポート 8.0)
<code>--user[=name],</code>	この MySQL ユーザーとして接続	(MySQL に基づくすべての NDB リリースでサポート 8.0)

形式	説明	追加、非推奨、または削除された
-u		

NDB 7.6.6 以降では、`ndb_top` は共通の NDB プログラムオプション `--defaults-file`, `--defaults-extra-file`, `--print-defaults`, `--no-defaults` および `--defaults-group-suffix` もサポートしています。(Bug #86614、Bug #26236298)

その他のオプション

- `--color, -c`

コマンド行形式	<code>--color</code>
型	Boolean
デフォルト値	TRUE

ASCII グラフを色で表示します。無効にするには `--skip-colors` を使用します。

- `--graph, -g`

コマンド行形式	<code>--graph</code>
型	Boolean
デフォルト値	TRUE

グラフを使用してデータを表示します。無効にするには `--skip-graphs` を使用します。このオプションまたは `--text` は true である必要があります。両方のオプションが true の場合があります。

- `--help, -?`

コマンド行形式	<code>--help</code>
型	Boolean
デフォルト値	TRUE

プログラム使用状況情報を表示します。

- `--host[=name], -h`

コマンド行形式	<code>--host[=name]</code>
型	文字列
デフォルト値	localhost

接続先の MySQL Server のホスト名または IP アドレス。

- `--measured-load, -m`

コマンド行形式	<code>--measured-load</code>
型	Boolean
デフォルト値	FALSE

測定された負荷をスレッド別に表示します。このオプションまたは `--os-load` は true である必要があります。両方のオプションが true の場合があります。

- `--node-id[=#], -n`

コマンド行形式	<code>--node-id[=#]</code>
型	Integer
デフォルト値	1

このノード ID を持つデータノードを監視します。

- `--os-load, -o`

コマンド行形式	<code>--os-load</code>
型	Boolean
デフォルト値	TRUE

オペレーティングシステムによって測定された負荷を表示します。このオプションまたは `--measured-load` は true である必要があります。両方のオプションが true の場合があります。

- `--passwd[=password], -p`

コマンド行形式	<code>--passwd[=password]</code>
型	Boolean
デフォルト値	NULL

このパスワードを使用して接続します。

このオプションは NDB 7.6.4 では非推奨です。NDB 7.6.6 では削除され、`--password` オプションに置き換えられます。(Bug #26907833)

- `--password[=password], -p`

コマンド行形式	<code>--password[=password]</code>
型	Boolean
デフォルト値	NULL

このパスワードを使用して接続します。

このオプションは、NDB 7.6.6 で、以前に使用された `--passwd` オプションの代替として追加されました。(Bug #26907833)

- `--port[=#], -t` (NDB 7.6.6 以降: `-P`)

コマンド行形式	<code>--port[=#]</code>
型	Integer
デフォルト値	3306

MySQL Server への接続時に使用するポート番号。

NDB 7.6.6 以降では、このオプションの短い形式は `-P` で、`-t` は `--text` オプションの短い形式として再利用されます。(Bug #26907833)

- `--sleep-time[=seconds], -s`

コマンド行形式	<code>--sleep-time[=seconds]</code>
型	Integer
デフォルト値	1

表示がリフレッシュされるまでの待機時間 (秒)。

- `--socket=path/to/file, -S`

コマンド行形式	<code>--socket</code>
型	パス名

デフォルト値	[none]
--------	--------

指定されたソケットファイルを接続に使用します。

NDB 7.6.6 に追加されました。(Bug #86614、Bug #26236298)

- `--sort, -r`

コマンド行形式	<code>--sort</code>
型	Boolean
デフォルト値	TRUE

スレッドを使用状況でソートします。無効にするには `--skip-sort` を使用します。

- `--text, -t`

コマンド行形式	<code>--text</code>
型	Boolean
デフォルト値	FALSE

テキストを使用したデータの表示。このオプションまたは `--graph` は true である必要があります。両方のオプションが true の場合があります。

このオプションの短い形式は以前のバージョンの NDB Cluster では `-x` でしたが、これはサポートされなくなりました。

- `--user[=name], -u`

コマンド行形式	<code>--user[=name]</code>
型	文字列
デフォルト値	root

この MySQL ユーザーとして接続します。

出力例. 次の図は、負荷が中程度の `ndbmtid` データノードを備えた Linux システムの端末ウィンドウで実行されている `ndb_top` を示しています。ここでは、テキストとグラフの両方の出力を提供するために、`ndb_top -n8 -x` を使用してプログラムが起動されています:

図 23.26 ターミナルで実行されている `ndb_top`

```

jens@vathaj:~$ ndb_top -n8 -x
ndb_top: 7.6.6
ldm_thr_no 2 OS view [user: 24%, system: 13%, idle: 63%] 37%
ldm_thr_no 2 OS view [user: 5%, system: 11%, idle: 84%] 16%
main_thr_no 0 OS view [user: 5%, system: 11%, idle: 84%] 16%
main_thr_no 0 OS view [user: 3%, system: 10%, idle: 87%] 13%
recv_thr_no 3 OS view [user: 3%, system: 10%, idle: 87%] 13%
rep_thr_no 1 OS view [user: 0%, system: 0%, idle: 100%] 0%
rep_thr_no 1 OS view [user: 0%, system: 0%, idle: 100%] 0%

```

NDB 8.0.20 以降、`ndb_top` はスレッドのスピン時間も緑色で表示します。

23.4.30 ndb_waiter — NDB Cluster が特定のステータスに達するまで待機

`ndb_waiter` は、クラスタが指定したステータスになるか、`--timeout` 制限を超えるまで、すべてのクラスタデータノードのステータスを繰り返し出力してから (100 ミリ秒ごと)、終了します。デフォルトでは、すべてのノードが起動されてクラスタに接続された状態である `STARTED` ステータスにクラスタがなるまで待機します。これは、`--no-contact` および `--not-started` オプションを使用してオーバーライドできます。

このユーティリティによって報告されるノード状態を次に示します。

- `NO_CONTACT`: ノードに接続できません。
- `UNKNOWN`: ノードに接続できますが、ステータスがまだ不明です。通常、これはノードが管理サーバーから `START` または `RESTART` コマンドを受け取ったけれども、まだそこで実行されていないことを意味します。
- `NOT_STARTED`: ノードは停止されたけれども、クラスタにまだ接続されています。これは、管理クライアントの `RESTART` コマンドを使用してノードを再起動したときに表示されます。
- `STARTING`: ノードの `ndbd` プロセスが開始されたけれども、ノードはまだクラスタに接続されていません。
- `STARTED`: ノードは動作しており、クラスタに接続されています。
- `SHUTTING_DOWN`: ノードがシャットダウン中です。
- `SINGLE USER MODE`: これは、クラスタがシングルユーザーモードの場合に、すべてのクラスタデータノードに表示されます。

次のテーブルに、NDB Cluster ネイティブバックアップリストアッププログラム `ndb_waiter` に固有のオプションを示します。追加説明が表のあとにあります。ほとんどの NDB Cluster プログラム (`ndb_waiter` を含む) に共通のオプションについては、[セクション23.4.32「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#)を参照してください。

表 23.49 プログラムで使用されるコマンドライン・オプション `ndb_waiter`

形式	説明	追加、非推奨、または削除された
<code>--no-contact</code> , <code>-n</code>	クラスタが NO CONTACT 状態になるのを待機します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--not-started</code>	クラスタが NOT STARTED 状態になるのを待機します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--single-user</code>	クラスタがシングルユーザーモードになるのを待機します	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--timeout=#</code> , <code>-t</code>	この秒数待機してから、クラスタが目的の状態に達したかどうかを終了	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--nowait-nodes=list</code>	待機しないノードのリスト	(MySQLに基づくすべてのNDBリリースでサポート 8.0)
<code>--wait-nodes=list</code> , <code>-w</code>	待機するノードのリスト	(MySQLに基づくすべてのNDBリリースでサポート 8.0)

使用法

```
ndb_waiter [-c connection_string]
```

その他のオプション

- `--no-contact`, `-n`

STARTED 状態を待機する代わりに、`ndb_waiter` はクラスタが **NO_CONTACT** ステータスになるまで実行を継続してから終了します。

- `--not-started`

STARTED 状態を待機する代わりに、`ndb_waiter` はクラスタが **NOT_STARTED** ステータスになるまで実行を継続してから終了します。

- `--timeout=seconds, -t seconds`

待機する時間。この秒数内に目的の状態に達しなかった場合、プログラムは終了します。デフォルトは 120 秒 (1200 レポートサイクル) です。

- `--single-user`

プログラムはクラスタがシングルユーザーモードになるのを待機します。

- `--nowait-nodes=list`

このオプションを使用すると、`ndb_waiter` は ID がリストされているノードを待機しません。次に示すように、このリストはカンマ区切りであり、範囲はダッシュで示すことができます。

```
shell> ndb_waiter --nowait-nodes=1,3,7-9
```

重要

このオプションは `--wait-nodes` オプションと一緒に使用しないでください。

- `--wait-nodes=list, -w list`

このオプションを使用した場合、`ndb_waiter` は ID がリストされているノードのみを待機します。次に示すように、このリストはカンマ区切りであり、範囲はダッシュで示すことができます。

```
shell> ndb_waiter --wait-nodes=2,4-6,10
```

重要

このオプションは `--nowait-nodes` オプションと一緒に使用しないでください。

出力例. 4 ノードのクラスタで 2 つのノードをシャットダウンしてから手動で再度起動したときの `ndb_waiter` の出力を次に示します。重複するレポート (... で示されています) は省略しています。

```
shell> ./ndb_waiter -c localhost
```

```
Connecting to mgmsrv at (localhost)
State node 1 STARTED
State node 2 NO_CONTACT
State node 3 STARTED
State node 4 NO_CONTACT
Waiting for cluster enter state STARTED
```

...

```
State node 1 STARTED
State node 2 UNKNOWN
State node 3 STARTED
State node 4 NO_CONTACT
Waiting for cluster enter state STARTED
```

...

```
State node 1 STARTED
State node 2 STARTING
State node 3 STARTED
State node 4 NO_CONTACT
Waiting for cluster enter state STARTED
```

...

```

State node 1 STARTED
State node 2 STARTING
State node 3 STARTED
State node 4 UNKNOWN
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTING
State node 3 STARTED
State node 4 STARTING
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTED
State node 3 STARTED
State node 4 STARTING
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTED
State node 3 STARTED
State node 4 STARTED
Waiting for cluster enter state STARTED
    
```

注記

接続文字列を指定しない場合、`ndb_waiter` は `localhost` の管理サーバーへの接続を試み、「Connecting to mgmsrv at (null)」を報告します。

NDB 8.0.20 より前では、このプログラムは、`NDBT` テストライブラリへの不要な依存関係のため、実行の完了時に `NDBT_ProgramExit - status` を出力しました。この依存関係は削除され、余分な出力がなくなりました。

23.4.31 ndbxfrm — NDB Cluster によって作成されたファイルの圧縮、圧縮解除、暗号化、および復号化

NDB 8.0.22 で導入された `ndbxfrm` ユーティリティを使用すると、NDB Cluster によって作成された圧縮、暗号化、またはその両方のファイルに関する情報を解凍、復号化、および出力できます。ファイルの圧縮または暗号化にも使用できます。

表 23.50 プログラムで使用されるコマンドライン・オプション ndbxfrm

形式	説明	追加、非推奨、または削除された
<code>--compress,</code> <code>-c</code>	Compress file	追加: NDB 8.0.22
<code>--decrypt-password=string</code>	ファイルの復号化にこのパスワードを使用	追加: NDB 8.0.22
<code>--encrypt-kdf-iter-count=#,</code> <code>-k</code>	キー定義で使用される反復数	追加: NDB 8.0.22
<code>--encrypt-password=string</code>	このパスワードを使用してファイルを暗号化	追加: NDB 8.0.22
<code>--help,</code> <code>-?</code>	使用方法の情報を出力します	追加: NDB 8.0.22
<code>--info,</code>	ファイル情報の印刷	追加: NDB 8.0.22

形式	説明	追加、非推奨、または削除された
<code>-i</code> <code>--usage,</code> <code>-?</code> <code>--version,</code> <code>-V</code>	使用方法の情報を出力します。--help のシノニムです 出力バージョン情報	追加: NDB 8.0.22 追加: NDB 8.0.22

使用法

```
ndbxfm --info file[ file ...]
ndbxfm --compress input_file output_file
ndbxfm --decrypt-password=password input_file output_file
ndbxfm [--encrypt-ldf-iter-count=#] --encrypt-password=password input_file output_file
```

`input_file` と `output_file` を同じファイルにすることはできません。

オプション

- `--compress, -c`

NDB Cluster バックアップの圧縮に使用されるものと同じ圧縮方法を使用して入力ファイルを圧縮し、出力ファイルに書き込みます。暗号化されていない圧縮 NDB バックアップファイルを解凍するには、圧縮ファイルおよび出力ファイルの名前を使用して `ndbxfm` を起動する必要があります (オプションは必要ありません)。

- `--decrypt-password=password`

指定されたパスワードを使用して、NDB によって暗号化されたファイルを復号化します。

- `--encrypt-kdf-iter-count=#, -k #`

ファイルを暗号化するときに、暗号化キーに使用する反復回数を指定します。--encrypt-password オプションが必要です。

- `--encrypt-password=password`

オプションで指定されたパスワードを使用してバックアップファイルを暗号化します。パスワードは、次の要件を満たしている必要があります:

- `!,',",$,%,\` および `^` 以外の印刷可能な ASCII 文字を使用
- 256 文字以下
- 一重引用符または二重引用符で囲みます

空のパスワード ("または"") を使用することは可能ですが、お薦めしません。

- `--help, -?`

プログラムの使用状況情報を出力します。

- `--info, -i`

1 つまたは複数の入力ファイルに関する次の情報を出力します:

- ファイルの名前
- ファイルが圧縮されているかどうか (`compression=yes` または `compression=no`)
- ファイルが暗号化されているかどうか (`encryption=yes` または `encryption=no`)

例:

```
shell> ndbxfrm -i BACKUP-10-0.5.Data BACKUP-10.5.ctl BACKUP-10.5.log
File=BACKUP-10-0.5.Data, compression=no, encryption=yes
File=BACKUP-10.5.ctl, compression=no, encryption=yes
File=BACKUP-10.5.log, compression=no, encryption=yes
```

- `--usage, -?`
`--help` と同義です。

- `--version, -V`

バージョン情報を出力します。

`ndbxfrm` では、NDB Cluster の任意のバージョンで作成されたバックアップを暗号化できます。バックアップを構成する `.Data`、`.ctl` および `.log` ファイルは個別に暗号化する必要があり、これらのファイルはデータノードごとに個別に暗号化する必要があります。暗号化すると、そのようなバックアップは NDB Cluster 8.0.22 以降の `ndbxfrm`、`ndb_restore`、または `ndb_print_backup` によってのみ復号化できます。

暗号化されたファイルは、次のように、`--encrypt-password` オプションと `--decrypt-password` オプションを一緒に使用して新しいパスワードで再暗号化できます:

```
ndbxfrm --decrypt-password=old --encrypt-password=new input_file output_file
```

前述の例では、`old` と `new` はそれぞれ古いパスワードと新しいパスワードです。これらの両方を引用符で囲む必要があります。入力ファイルは復号化され、出力ファイルとして暗号化されます。入力ファイル自体は変更されません。古いパスワードを使用してアクセスできないようにするには、入力ファイルを手動で削除する必要があります。

`ndbxfrm` では、このセクションに明示的にリストされているオプション (`--help`、`--usage` および `--version`) を除き、[セクション23.4.32「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#) に示されているオプションは使用されません。

23.4.32 NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション

NDB Cluster プログラムはすべて、このセクションで説明するオプションを受け入れますが、次の例外があります:

- `mysqld`
- `ndb_print_backup_file`
- `ndb_print_schema_file`
- `ndb_print_sys_file`

注記

NDB Cluster の以前のバージョンのユーザーは、これらのオプションの一部を変更して、互いに一貫性を持たせ、`mysqld` と整合性を持たせるようにしてください。`ndb_print_backup_file`、`ndb_print_schema_file`、および `ndb_print_sys_file` 以外の NDB Cluster プログラムで `--help` オプションを使用すると、プログラムがサポートするオプションのリストを表示できます。

次のテーブルのオプションは、すべての NDB Cluster 実行可能ファイル (このセクションで前述したオプションを除く) に共通です。

個々の NDB Cluster プログラムに固有のオプションについては、[セクション23.4「NDB Cluster プログラム」](#) を参照してください。

NDB Cluster に関連する `mysqld` オプションについては、[NDB Cluster の MySQL Server オプション](#) を参照してください。

• `--character-sets-dir=name`

コマンド行形式	<code>--character-sets-dir=dir_name</code>
型	ディレクトリ名
デフォルト値	

文字セット情報を探す場所をプログラムに通知します。

• `--connect-retries=#`

コマンド行形式	<code>--connect-retries=#</code>
型	数値
デフォルト値	12
最小値	0
最大値	4294967295

このオプションは、最初に接続を再試行してから中止するまでの回数を指定します (クライアントは常に接続を 1 回以上試行します)。試行ごとの待機時間は、`--connect-retry-delay` を使用して設定されます。

注記

`ndb_mgm` とともに使用する場合、このオプションのデフォルトは 3 です。詳しくは [セクション 23.4.5 「ndb_mgm — NDB Cluster 管理クライアント」](#) をご覧ください。

• `--connect-retry-delay=#`

コマンド行形式	<code>--connect-retry-delay=#</code>
型	数値
デフォルト値	5
最小値	1
最小値	0
最大値	4294967295

このオプションは、接続が切断されるまでに試行ごとに待機する時間の長さを指定します。接続を試行する回数は、`--connect-retries` によって設定されます。

• `--core-file`

コマンド行形式	<code>--core-file</code>
型	Boolean
デフォルト値	FALSE

プログラムが停止した場合、コアファイルが出力されます。コアファイルの名前および場所はシステムによって異なります (Linux で実行されている NDB Cluster プログラムノードの場合、デフォルトの場所は、データノードの場合はノード `DataDir` です。) 一部のシステムでは、制限または制限がある場合があります。たとえば、サーバーを起動する前に `ulimit -c unlimited` を実行することが必要な場合もあります。詳細は、システムのドキュメントを参照してください。

NDB Cluster が `configure` の `--debug` オプションを使用して構築された場合、`--core-file` はデフォルトで有効になっています。通常のビルドの場合、デフォルトでは `--core-file` は無効にされています。

• `--debug=[options]`

コマンド行形式	<code>--debug=options</code>
型	文字列

デフォルト値	d:t:O,/tmp/ndb_restore.trace
--------	------------------------------

このオプションは、デバッグを有効にしてコンパイルしたバージョンにのみ使用できます。mysqld プロセスの場合と同様に、デバッグ呼び出しからの出力を有効にするために使用します。

- `--defaults-extra-file=filename`

コマンド行形式	<code>--defaults-extra-file=filename</code>
型	文字列
デフォルト値	[none]

グローバルオプションファイルが読み取られた後、このファイルを読み取ります。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--defaults-file=filename`

コマンド行形式	<code>--defaults-file=filename</code>
型	文字列
デフォルト値	[none]

このファイルからデフォルトのオプションを読み取ります。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--defaults-group-suffix`

コマンド行形式	<code>--defaults-group-suffix</code>
型	文字列
デフォルト値	[none]

また、名前がこの接尾辞で終わるグループも読み取ります。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--help, --usage, -?`

コマンド行形式	<code>--help</code> <code>--usage</code>
---------	---

使用可能なコマンドオプションの説明の短いリストを出力します。

- `--login-path=path`

コマンド行形式	<code>--login-path=path</code>
型	文字列
デフォルト値	[none]

ログインファイルからこのパスを読み取ります。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--ndb-connectstring=connection_string, --connect-string=connection_string, -c connection_string`

コマンド行形式	<code>--ndb-connectstring=connectstring</code> <code>--connect-string=connectstring</code>
型	文字列
デフォルト値	<code>localhost:1186</code>

このオプションは、次に示すように、アプリケーションが接続する管理サーバーを指定する NDB Cluster 接続文字列を取ります:

```
shell> ndbd --ndb-connectstring="nodeid=2;host=ndb_mgmd.mysql.com:1186"
```

詳細は、[セクション23.3.3.3「NDB Cluster 接続文字列」](#)を参照してください。

- `--ndb-mgmd-host=host[:port]`

コマンド行形式	<code>--ndb-mgmd-host=host[:port]</code>
型	文字列
デフォルト値	<code>localhost:1186</code>

プログラムに接続される単一の管理サーバーのホストとポート番号の設定に使用できます。プログラムが接続情報に複数の管理サーバーのノード ID または参照 (あるいはその両方) を必要とする場合は、`--ndb-connectstring` オプションを代わりに使用します。

- `--ndb-nodeid=#`

コマンド行形式	<code>--ndb-nodeid=#</code>
型	数値
デフォルト値	<code>0</code>

このノード NDB Cluster ノード ID を設定します。許可される値の範囲は、ノードタイプ (データ、管理、または API) と NDB Cluster ソフトウェアバージョンによって異なります。詳細は、[セクション23.1.7.2「NDB Cluster と標準の MySQL 制限の制限と相違点」](#)を参照してください。

- `--no-defaults`

コマンド行形式	<code>--no-defaults</code>
型	Boolean
デフォルト値	<code>TRUE</code>

ログインファイル以外のオプションファイルからデフォルトのオプションを読み取らないでください。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--ndb-optimized-node-selection`

コマンド行形式	<code>--ndb-optimized-node-selection</code>
型	Boolean
デフォルト値	<code>TRUE</code>

トランザクションのためのノードの選択を最適化します。デフォルトで有効。

- `--print-defaults`

コマンド行形式	<code>--print-defaults</code>
型	Boolean

デフォルト値	TRUE
--------	------

プログラム引数リストを出力して終了します。

このオプションおよびその他のオプションファイルオプションの詳細は、[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)を参照してください。

- `--version, -V`

コマンド行形式	<code>--version</code>
---------	------------------------

実行可能ファイルの NDB Cluster バージョン番号を出力します。すべてのバージョンを一緒に使用できるわけではないため、バージョン番号が関連しており、NDB Cluster の起動プロセスは、使用されているバイナリのバージョンが同じクラスタ内に共存できることを検証します。これは、NDB Cluster のオンライン（ローリング）ソフトウェアアップグレードまたはダウングレードを実行する場合にも重要です。

詳細は、[セクション23.5.5「NDB Cluster のローリング再起動の実行」](#)を参照してください。

23.5 NDB Cluster の管理

NDB Cluster の管理には多数のタスクが含まれ、最初に NDB Cluster を構成および起動します。これについては、[セクション23.3「NDB Cluster の構成」](#)および[セクション23.4「NDB Cluster プログラム」](#)で説明されています。

次のいくつかのセクションでは、実行中の NDB Cluster の管理について説明します。

NDB Cluster の管理および配備に関するセキュリティの問題については、[セクション23.5.17「NDB Cluster のセキュリティの問題」](#)を参照してください。

基本的に、実行中の NDB Cluster をアクティブに管理する方法は 2 つあります。それらのうちの 1 つ目は、管理クライアントに入力されたコマンドを使用するものです。これにより、クラスタのステータスをチェックしたり、ログレベルを変更したり、バックアップを開始および停止したり、ノードを停止および起動したりできます。2 つ目の方法には、クラスタログ `ndb_node_id_cluster.log` の内容の調査が伴います。通常、これは管理サーバーの `DataDir` ディレクトリで見つかりますが、この場所は `LogDestination` オプションを使用すればオーバーライドできます。(`node_id` は、アクティビティのログが記録されるノードの一意の識別子を表すことを思い出してください。) クラスタログには、`ndbd` で生成されたイベントレポートが含まれます。クラスタログのエントリを Unix システムログに送信することもできます。

クラスタ操作の一部の側面は、`SHOW ENGINE NDB STATUS` ステートメントを使用して SQL ノードから監視することもできます。

NDB Cluster 操作の詳細は、`ndbinfo` データベースを使用した SQL インタフェースを介してリアルタイムで入手できます。詳細は、[セクション23.5.14「ndbinfo: NDB Cluster 情報データベース」](#)を参照してください。

NDB 統計カウンタでは、`mysql` クライアントを使用したモニタリングが改善されています。NDB カーネルに実装されているこれらのカウンタは、トランザクションの開始、クローズ、中断、主キーと一意キーの操作、テーブル、範囲、ブルーニングされたスキャン、さまざまな操作の完了を待機しているブロックされたスレッド、NDB Cluster によって送受信されたデータとイベントなど、`Ndb` オブジェクトによって実行または影響を受ける操作に関連しています。カウンタは、NDB API が呼び出されるたびに、またはデータノードでデータが送受信されるたびに NDB カーネルによってインクリメントされます。

`mysqld` では、NDB API 統計カウンタがシステムステータス変数として表示されます。これらは、すべての名前に共通するプリフィクス (`Ndb_api_`) から識別できます。これらの変数の値は、`SHOW STATUS` ステートメントの出力から、またはパフォーマンススキーマ `session_status` または `global_status` テーブルのいずれかをクエリーすることによって、`mysql` クライアントで読み取ることができます。NDB テーブルに作用する SQL ステートメントの実行前後のステータス変数の値を比較することで、NDB Cluster のモニタリングおよびパフォーマンスチューニングに役立つこのステートメントに対応する NDB API レベルで実行されるアクションを監視できます。

MySQL Cluster Manager は、多数のノードを持つ NDB Cluster の起動、停止、再起動など、それ以外の多くの複雑な NDB Cluster 管理タスクを簡略化する高度なコマンド行インタフェースを提供します。MySQL Cluster Manager クライアントは、ほとんどのノード構成パラメータの値、NDB Cluster に関連する `mysqld` サーバーオプションおよび変数

を取得および設定するためのコマンドもサポートしています。MySQL Cluster Manager バージョン 1.4.8 は NDB 8.0 の実験的なサポートを提供します。詳しくは [MySQL Cluster Manager 1.4.8 User Manual](#), をご覧ください。

23.5.1 NDB Cluster 管理クライアントのコマンド

主要な構成ファイルに加えて、管理クライアント `ndb_mgm` から使用可能なコマンド行インタフェースによって、クラスタを制御することもできます。これは、実行中のクラスタへのプライマリ管理インタフェースです。

イベントログのコマンドは、[セクション23.5.3「NDB Cluster で生成されるイベントレポート」](#)に示しています。バックアップを作成するためのコマンドと、バックアップからリストアするためのコマンドは、[セクション23.5.8「NDB Cluster のオンラインバックアップ」](#)で説明しています。

MySQL Cluster Manager での `ndb_mgm` の使用。MySQL Cluster Manager 1.4.8 は NDB 8.0 を完全にサポートします。MySQL Cluster Manager はプロセスの起動と停止を内部的に処理し、その状態を追跡するため、MySQL Cluster Manager 制御下にある NDB Cluster のこれらのタスクに `ndb_mgm` を使用する必要はありません。NDB Cluster 配布に付属の `ndb_mgm` コマンドラインクライアントを使用して、ノードの起動または停止を伴う操作を実行しないことをお勧めします。これらには、`START`、`STOP`、`RESTART`、および `SHUTDOWN` が含まれますが、これだけに限定されません。詳細は、[MySQL Cluster Manager Process Commands](#)を参照してください。

管理クライアントには、次のような基本的なコマンドが用意されています。次のリストで、`node_id` はデータノード ID またはキーワード `ALL` のいずれかを示します。これは、コマンドをすべてのクラスタデータノードに適用する必要があることを示します。

- `HELP`

使用可能なすべてのコマンドに関する情報を表示します。

- `CONNECT connection-string`

接続文字列で示される管理サーバーに接続します。クライアントがすでにこのサーバーに接続されている場合、クライアントは再接続します。

- `SHOW`

クラスタのステータスに関する情報を表示します。可能性のあるノードステータスの値には、`UNKNOWN`、`NO_CONTACT`、`NOT_STARTED`、`STARTING`、`STARTED`、`SHUTTING_DOWN`、および `RESTARTING` が含まれます。

このコマンドからの出力には、クラスタがシングルユーザーモード (ステータス `SINGLE USER MODE`) になるタイミングも示されます。NDB 8.0.17 以降では、このモードが有効な場合、どの API または SQL ノードが排他的アクセスを持つかも示します。これは、クラスタに接続されているすべてのデータノードおよび管理ノードが 8.0.17 以降のバージョンである場合にのみ機能します。

- `node_id START`

`node_id` によって識別されるデータノード (またはすべてのデータノード) をオンラインにします。

`ALL START` はすべてのデータノードでのみ動作し、管理ノードには影響を与えません。

重要

このコマンドを使用してデータノードをオンラインにするには、`--nostart` または `-n` を使用してデータノードが起動されている必要があります。

- `node_id STOP [-a] [-f]`

`node_id` によって識別されるデータノードまたは管理ノードを停止します。

注記

`ALL STOP` は、すべてのデータノードのみを停止するように機能し、管理ノードには影響しません。

このコマンドの影響を受けるノードはクラスタから切断され、それに関連付けられた `ndbd` または `ndb_mgmd` プロセスは終了します。

`-a` オプションを使用すると、保留中のトランザクションが完了するまで待機せずに、すぐにノードが停止されます。

通常、結果として不完全なクラスタが生成される場合は、`STOP` に失敗します。`-f` オプションを使用すると、これをチェックせずに、強制的にノードがシャットダウンされます。このオプションが使用され、結果が不完全なクラスタである場合は、すぐにクラスタがシャットダウンします。

警告

また、`-a` オプションを使用すると、ノードが停止されても不完全なクラスタが生成されないように、`STOP` の呼び出し時に本来実行される安全性チェックも無効になります。つまり、`STOP` コマンドで `-a` オプションを使用すると、クラスタは `NDB` に格納されたすべてのデータの完全なコピーを保持しなくなるため、強制的にシャットダウンされる可能性があることにより、このオプションを使用する際は、細心の注意を払うようにしてください。

• `node_id RESTART [-n] [-i] [-a] [-f]`

`node_id` によって識別されるデータノード (またはすべてのデータノード) を再起動します。

`RESTART` で `-i` オプションを使用すると、データノードは初期再起動を実行します。つまり、ノードファイルシステムが削除され、再作成されます。この効果は、データノードのプロセスを停止してから、システムシェルから `ndbd --initial` を使用して再起動した場合と同じです。

注記

このオプションを使用した場合、バックアップファイルおよびディスクデータファイルは削除されません。

`-n` オプションを使用するとデータノードプロセスが再起動されますが、実際には、適切な `START` コマンドが発行されるまでデータノードはオンラインになりません。このオプションの効果は、データノードを停止してから、システムシェルから `ndbd --nostart` または `ndbd -n` を使用して再起動した場合と同じです。

`-a` を使用すると、このノードに依存している現在のトランザクションがすべて中止されます。ノードがクラスタに再度参加するときは、GCP チェックが実行されません。

通常、ノードをオフラインにしたことで不完全なクラスタが生成される場合は、`RESTART` が失敗します。`-f` オプションを使用すると、これをチェックせずに、強制的にノードが再起動されます。このオプションが使用され、結果が不完全なクラスタである場合は、クラスタ全体が再起動されます。

• `node_id STATUS`

`node_id` によって識別されるデータノード (またはすべてのデータノード) のステータス情報を表示します。

このコマンドからの出力には、クラスタがシングルユーザーモードになるタイミングも示されます。

• `node_id REPORT report-type`

`node_id` によって識別されるデータノード、または `ALL` を使用するすべてのデータノードに対する `report-type` タイプのレポートを表示します。

現在、`report-type` には次の 3 つの許容値があります:

- `BackupStatus` では、進行中のクラスタバックアップに関するステータスレポートが提供されます。
- `MemoryUsage` では、次の例で示すように、各データノードで使用されているデータメモリーおよびインデックスメモリーの量が表示されます。

```
ndb_mgm> ALL REPORT MEMORY
```

```
Node 1: Data usage is 5%(177 32K pages of total 3200)
Node 1: Index usage is 0%(108 8K pages of total 12832)
Node 2: Data usage is 5%(177 32K pages of total 3200)
Node 2: Index usage is 0%(108 8K pages of total 12832)
```

この情報は、`ndbinfo.memoryusage` テーブルから取得することもできます。

- `EventLog` は、1 つ以上のデータノードのイベントログバッファからイベントを報告します。

`report-type` は大文字と小文字が区別されず、「あいまい」です。`MemoryUsage` では、`MEMORY` (前述の例で示すとおり)、`memory`、または単に `MEM` (または `mem`) を使用できます。`BackupStatus` も同様に短縮できます。

- `ENTER SINGLE USER MODE node_id`

シングルユーザーモードを開始します。これにより、ノード ID `node_id` によって識別される MySQL サーバーにのみデータベースへのアクセスが許可されます。

NDB 8.0.17 以降、`ndb_mgm` クライアントは、次に示すように、このコマンドが発行されて有効になったことを明確に確認します：

```
ndb_mgm> ENTER SINGLE USER MODE 100
Single user mode entered
Access is granted for API node 100 only.
```

また、NDB 8.0.17 以降では、シングルユーザーモードのときに排他的アクセスを持つ API または SQL ノードは、次のように `SHOW` コマンドの出力に示されます：

```
ndb_mgm> SHOW
Cluster Configuration
-----
[ndbd(NDB)] 2 node(s)
id=5 @127.0.0.1 (mysql-8.0.17 ndb-8.0.17, single user mode, Nodegroup: 0, *)
id=6 @127.0.0.1 (mysql-8.0.17 ndb-8.0.17, single user mode, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=50 @127.0.0.1 (mysql-8.0.17 ndb-8.0.17)

[mysqld(API)] 2 node(s)
id=100 @127.0.0.1 (mysql-8.0.17 ndb-8.0.17, allowed single user)
id=101 (not connected, accepting connect from any host)
```

注記

この機能を有効にするには、すべてのデータおよび管理ノードで NDB Cluster ソフトウェアのバージョン 8.0.17 が実行されている必要があります。

- `EXIT SINGLE USER MODE`

シングルユーザーモードを終了します。これにより、すべての SQL ノード (つまり、実行中のすべての `mysqld` プロセス) がデータベースにアクセスできます。

注記

シングルユーザーモードでないときでも、`EXIT SINGLE USER MODE` を使用できますが、この場合、コマンドの効果はありません。

- `QUIT, EXIT`

管理クライアントを終了します。

このコマンドは、クラスタに接続されているノードに影響を与えません。

- SHUTDOWN

すべてのクラスタデータノードおよび管理ノードをシャットダウンします。これが実行されたあとに管理クライアントを終了するには、[EXIT](#) または [QUIT](#) を使用します。

このコマンドは、クラスタに接続しているどの SQL ノードまたは API ノードもシャットダウンしません。

- CREATE NODEGROUP nodeid[, nodeid, ...]

新しい NDB Cluster ノードグループを作成し、データノードを結合させます。

このコマンドは、NDB Cluster に新しいデータノードをオンラインで追加したあとに使用され、新しいノードグループに参加してクラスタへの参加を完全に開始します。このコマンドは唯一のパラメータとして、ノード ID をカンマで区切ったリストを取りますが、これらは、先ほど追加した、新しいノードグループに参加することになるノードの ID です。ノードの数は、すでにクラスタの一部である各ノードグループ内のノードの数と同じである必要があります (NDB Cluster ノードグループごとに同じ数のノードを持つ必要があります)。つまり、NDB Cluster にそれぞれ 2 つのデータノードの 2 つのノードグループがある場合、新しいノードグループにも 2 つのデータノードが必要です。

このコマンドで作成された新しいノードグループのノードグループ ID は自動的に決定され、常に、クラスタ内で次に最大の未使用のノードグループ ID となり、手動で設定することはできません。

詳細は、[セクション23.5.7「NDB Cluster データノードのオンラインでの追加」](#)を参照してください。

- DROP NODEGROUP nodegroup_id

指定された `nodegroup_id` の NDB Cluster ノードグループを削除します。

このコマンドを使用すると、NDB Cluster からノードグループを削除できます。[DROP NODEGROUP](#) は唯一の引数として、削除するノードグループのノードグループ ID を取ります。

[DROP NODEGROUP](#) は、影響を受けるノードグループ内のデータグループをそのノードグループから削除するためにのみ機能します。データノードは停止されず、別のノードグループにも割り当てられず、クラスタの構成からも削除されません。ノードグループに属さないデータノードは、次のように (太字のテキストを使用して示すように)、ノードグループ ID の代わりに、`no nodegroup` を付けた管理クライアントの [SHOW](#) コマンドの出力に表示されます。

```
id=3 @10.100.2.67 (8.0.23-ndb-8.0.23, no nodegroup)
```

[DROP NODEGROUP](#) は、削除されるノードグループ内のすべてのデータノードに、テーブルデータやテーブル定義がまったく存在しない場合にのみ機能します。現在は、`ndb_mgm` または `mysql` クライアントを使用して、特定のデータノードやノードグループからすべてのデータを削除する方法がないため、これは次のような 2 つの場合にのみコマンドが成功することを意味します。

1. `ndb_mgm` クライアントで [CREATE NODEGROUP](#) を発行した後、`mysql` クライアントで [ALTER TABLE ... REORGANIZE PARTITION](#) ステートメントを発行する前。
2. [DROP TABLE](#) を使用してすべての `NDBCLUSTER` テーブルを削除したあと。

[TRUNCATE TABLE](#) はテーブルデータのみを削除するため、この目的では機能しません。テーブルのメタデータが削除される [DROP TABLE](#) ステートメントが発行されるまで、データノードには引き続き `NDBCLUSTER` テーブルの定義が格納されます。

[DROP NODEGROUP](#) の詳細は、[セクション23.5.7「NDB Cluster データノードのオンラインでの追加」](#)を参照してください。

- [PROMPT \[prompt\]](#)

`ndb_mgm` によって表示されるプロンプトを文字列リテラル `prompt` に変更します。

`prompt` は引用符で囲まないでください (プロンプトに引用符を含める場合を除く)。 `mysql` クライアントの場合とは異なり、特殊文字シーケンスおよびエスケープは認識されません。引数を指定せずにコールすると、プロンプトはデフォルト値 (`ndb_mgm>`) にリセットされます。

次に例をいくつか示します:

```
ndb_mgm> PROMPT mgm#1:
mgm#1: SHOW
Cluster Configuration
...
mgm#1: PROMPT mymgm >
mymgm > PROMPT 'mymgm:'
'mymgm:' PROMPT mymgm:
mymgm: PROMPT
ndb_mgm> EXIT
jon@valhaj:~/bin>
```

`prompt` 文字列内の先頭の空白および空白は切り捨てられないことに注意してください。末尾のスペースは削除されます。

- [node_id NODELOG DEBUG {ON|OFF}](#)

影響を受けるデータノードが `--verbose` オプションで起動されたかのように、ノードログ内のデバッグロギングを切り替えます。 `NODELOG DEBUG ON` はデバッグロギングを開始します。 `NODELOG DEBUG OFF` はデバッグロギングをオフに切り替えます。

その他のコマンド。 次のリストに示すように、 `ndb_mgm` クライアントで使用可能なその他の多くのコマンドについては、別の場所で説明します:

- `START BACKUP` は、 `ndb_mgm` クライアントでオンラインバックアップを実行するために使用されます。 `ABORT BACKUP` コマンドは、すでに進行中のバックアップを取り消すために使用されます。詳細は、 [セクション 23.5.8 「NDB Cluster のオンラインバックアップ」](#) を参照してください。
- `CLUSTERLOG` コマンドは、様々なロギング機能を実行するために使用します。詳細および例については、 [セクション 23.5.3 「NDB Cluster で生成されるイベントレポート」](#) を参照してください。このセクションで前述したように、 `NODELOG DEBUG` はノードログのデバッグ出力をアクティブ化または非アクティブ化します。
- テストおよび診断作業のために、クライアントは、クラスタで内部コマンドを実行するために使用できる `DUMP` コマンドをサポートしています。MySQL サポートから指示されないかぎり、本番設定では使用しないでください。詳細は、 [MySQL NDB Cluster Internals Manual](#) を参照してください。

23.5.2 NDB Cluster ログメッセージ

このセクションでは、さまざまなクラスタログイベントに対応してクラスタに書き込まれるメッセージに関する情報を提供します。 `NDB` トランスポータのエラーについて、より具体的な追加情報も提供します。

23.5.2.1 NDB Cluster: クラスタログ内のメッセージ

次の表には、もっとも一般的な `NDB` クラスタログメッセージを一覧表示します。クラスタログ、ログイベント、およびイベントタイプについては、 [セクション 23.5.3 「NDB Cluster で生成されるイベントレポート」](#) を参照してください。これらのログメッセージは、MGM API のログイベントタイプにも対応します。Cluster API の開発者が関心を持つような関連情報については、 [The Ndb_logevent_type Type](#) を参照してください。

表 23.51 NDB クラスタの一般的なログメッセージ

ログメッセージ	説明	イベント名	イベントタイプ	優先度	重大度
<code>Node mgm_node_id: Node data_node_id Connected</code>	ノード ID <code>node_id</code> を持つデータノードが管理サーバー (ノード	<code>Connected</code>	<code>Connection</code>	8	<code>INFO</code>

ログメッセージ	説明	イベント名	イベントタイプ	優先度	重大度
	mgm_node_id) に接続しました。				
Node mgm_node_id: Node data_node_id Disconnected	ノード ID data_node_id を持つデータノードが管理サーバー (ノード mgm_node_id) から切断しました。	Disconnected	Connection	8	ALERT
Node data_node_id: Communication to Node api_node_id closed	ノード ID api_node_id を持つ API ノードまたは SQL ノードがデータノード data_node_id と通信しなくなりました。	CommunicationClosed	Connection	8	INFO
Node data_node_id: Communication to Node api_node_id opened	ノード ID api_node_id を持つ API ノードまたは SQL ノードがデータノード data_node_id と通信しています。	CommunicationOpened	Connection	8	INFO
Node mgm_node_id: Node api_node_id: API version	ノード ID api_node_id を持つ API ノードが NDB API バージョン version (一般に、MySQL バージョン番号と同じ) を使用して、管理ノード mgm_node_id に接続しました。	ConnectedApiVersion	Connection	8	INFO
Node node_id: Global checkpoint gci started	ID gci を持つグローバルチェックポイントが開始されました。ノード node_id は、このグローバルチェックポイントを担当するマスターです。	GlobalCheckpointStarted	Checkpoint	9	INFO
Node node_id: Global checkpoint gci completed	ID gci を持つグローバルチェックポイントが完了しました。ノード node_id は、このグローバルチェックポイントを担当したマスターでした。	GlobalCheckpointCompleted	Checkpoint	10	INFO
Node node_id: Local checkpoint	シーケンス ID lcp を持つロー	LocalCheckpointStarted	Checkpoint	7	INFO

ログメッセージ	説明	イベント名	イベントタイプ	優先度	重大度
lcp started. Keep GCI = current_gci oldest restorable GCI = old_gci	カルチェックポイントがノード <code>node_id</code> で開始されました。使用可能な最新の GCI はインデックス <code>current_gci</code> を持ち、クラスタをリストアできるもっとも古い GCI はインデックス <code>old_gci</code> を持っています。				
Node node_id: Local checkpoint lcp completed	ノード <code>node_id</code> 上でシーケンス ID <code>lcp</code> を持つローカルチェックポイントが完了しました。	LocalCheckpointCompleted	Checkpoint	8	INFO
Node node_id: Local Checkpoint stopped in CALCULATED_KEEPING	このノードは、使用可能な最新の GCI を特定できませんでした。	LCPStoppedInCalculating	Checkpoint	0	ALERT
Node node_id: Table ID = table_id, fragment ID = fragment_id has completed LCP on Node node_id maxGciStarted: started_gci maxGciCompleted: completed_gci	ノード <code>node_id</code> 上のディスクに対して、テーブルフラグメントのチェックポイントが実行されました。進行中の GCI はインデックス <code>started_gci</code> を持ち、最近完了した GCI はインデックス <code>completed_gci</code> を持っています。	LCPFragmentCompleted	Checkpoint	11	INFO
Node node_id: ACC Blocked num_1 and TUP Blocked num_2 times last second	ログバッファがオーバーフローし、そのため、Undo ロギングがブロックされています。	UndoLogBlocked	Checkpoint	7	INFO
Node node_id: Start initiated version	NDB バージョン <code>version</code> を実行しているデータノード <code>node_id</code> が起動プロセスを開始しています。	NDBStartStarted	StartUp	1	INFO
Node node_id: Started version	NDB バージョン <code>version</code> を実行しているデータノード <code>node_id</code> が正常に起動されました。	NDBStartCompleted	StartUp	1	INFO

ログメッセージ	説明	イベント名	イベントタイプ	優先度	重大度
Node node_id: STTORY received after restart finished	ノードがクラスタの再起動が完了したことを示すシグナルを受信しました。	STTORYReieved	StartUp	15	INFO
Node node_id: Start phase phase completed (type)	ノードが type 起動の起動フェーズ phase を完了しました。起動フェーズのリストについては、セクション23.5.4「NDB Cluster 起動フェーズのサマリー」を参照してください (type は initial、system、node、initial node、または <Unknown> のいずれかです)。	StartPhaseComple	StartUp	4	INFO
Node node_id: CM_REGCONF president = president_id, own Node = own_id, our dynamic id = dynamic_id	ノード president_id が「プレジデント」として選択されました。own_id および dynamic_id は、常にレポートノードの ID (node_id) と同じであるべきです。	CM_REGCONF	StartUp	3	INFO
Node node_id: CM_REGREF from Node president_id to our Node node_id. Cause = cause	レポートノード (ID node_id) がプレジデントとしてノード president_id を受け入れることができませんでした。問題の cause は、Busy、Election with wait = false、Not president、Election without selecting new candidate、または No such cause のいずれかとして指定されます。	CM_REGREF	StartUp	8	INFO
Node node_id: We are Node own_id with dynamic ID dynamic_id, our left neighbor is Node id_1, our right is Node id_2	ノードがクラスタ内の隣接するノード (ノード id_1 とノード id_2) を検出しました。node_id、own_id、および	FIND_NEIGHBOUR	StartUp	8	INFO

ログメッセージ	説明	イベント名	イベントタイプ	優先度	重大度
	<code>dynamic_id</code> は、常に同じであるべきです。そうでない場合は、クラスタノード構成の重大な誤りを示しています。				
Node node_id: type shutdown initiated	ノードがシャットダウンシグナルを受信しました。シャットダウンの <code>type</code> は <code>Cluster</code> または <code>Node</code> です。	NDBStopStarted	StartUp	1	INFO
Node node_id: Node shutdown completed [, action] [Initiated by signal.]	ノードがシャットダウンされました。このレポートには、 <code>action</code> が含まれる場合があります。これが存在する場合は、 <code>restarting</code> 、 <code>no start</code> 、または <code>initial</code> のいずれかです。また、レポートには <code>NDB</code> プロトコル <code>signal</code> への参照が含まれる場合があります。可能性のあるシグナルについては、 Operations and Signals を参照してください。	NDBStopCompleted	StartUp	1	INFO
Node node_id: Forced node shutdown completed [, action]. [Occurred during startphase start_phase.] [Initiated by signal.] [Caused by error error_code: 'error_message(error_message)'. error_status'. [(extra info extra_code)]]	ノードが強制的にシャットダウンされました。あとで実行された <code>action</code> (<code>restarting</code> 、 <code>no start</code> 、または <code>initial</code> のいずれか) があれば、それもレポートされます。ノードの起動時にシグナルが送信された場合、レポートに、ノードの障害が発生した <code>start_phase</code> が含まれます。これが、ノードに <code>signal</code> が送信された結果であった場合は、その情報も提供されます (詳細は、 Operations and Signals を参照	NDBStopForced	StartUp	1	ALERT

ログメッセージ	説明	イベント名	イベントタイプ	優先度	重大度
	してください)。障害の原因が既知のエラーである場合は、それも含まれます。NDBのエラーメッセージおよび分類についての詳細は、 NDB Cluster API Errors を参照してください。				
Node node_id: Node shutdown aborted	ノードのシャットダウンプロセスがユーザーによって中止されました。	NDBStopAborted	StartUp	1	INFO
Node node_id: StartLog: [GCI Keep: keep_pos LastCompleted: last_pos NewestRestorable: restore_pos]	これは、ノード起動時に参照されるグローバルチェックポイントをレポートします。keep_posよりも前の Redo ログは削除されます。last_posは、データノードが最後に関与したグローバルチェックポイントです。restore_posは、すべてのデータノードをリストアするために実際に使用されるグローバルチェックポイントです。	StartREDOLog	StartUp	4	INFO
startup_message [Listed separately; see below.]	さまざまな環境下でログに記録される可能性のある多くの起動メッセージがあります。これらは個別にリストされます。 セクション23.5.2.2「NDB Cluster のログ起動メッセージ」 を参照してください。	StartReport	StartUp	4	INFO
Node node_id: Node restart completed copy of dictionary information	再起動されたノードへのデータディクショナリ情報のコピーが完了しました。	NR_CopyDict	NodeRestart	8	INFO
Node node_id: Node restart completed copy	再起動されたノードへのデータ配布	NR_CopyDistr	NodeRestart	8	INFO

ログメッセージ	説明	イベント名	イベントタイプ	優先度	重大度
of distribution information	情報のコピーが完了しました。				
Node node_id: Node restart starting to copy the fragments to Node node_id	起動データノード node_id へのフラグメントのコピーが開始されました。	NR_CopyFragStart	NodeRestart	8	INFO
Node node_id: Table ID = table_id, fragment ID = fragment_id have been copied to Node node_id	テーブル table_id からのフラグメント fragment_id がデータノード node_id にコピーされました。	NR_CopyFragDone	NodeRestart	10	INFO
Node node_id: Node restart completed copying the fragments to Node node_id	再起動データノード node_id へのすべてのテーブルフラグメントのコピーが完了しました。	NR_CopyFragCompleted	NodeRestart	8	INFO
Node node_id: Node node1_id completed failure of Node node2_id	データノード node1_id がデータノード node2_id の障害を検出しました。	NodeFailCompleted	NodeRestart	8	ALERT
All nodes completed failure of Node node_id	すべての(残りの)データノードがデータノード node_id の障害を検出しました。	NodeFailCompleted	NodeRestart	8	ALERT
Node failure of node_idblock completed	データノード node_id の障害が blockNDB カーネルブロックで検出されました。ここで、ブロックは DBTC、DBDICT、DBDIH、または DBLQH のいずれかです。詳細は、 NDB Kernel Blocks を参照してください。	NodeFailCompleted	NodeRestart	8	ALERT
Node mgm_node_id: Node data_node_id has failed. The Node state at failure was state_code	データノードで障害が発生しました。障害発生時の状態は、アービトレーション状態コード state_code で記述されます。可能性のある状態コードの値は、 include/kernel/signaldata/ArbitSignalData.hpp ファイルで見つかります。	NODE_FAILREP	NodeRestart	8	ALERT

ログメッセージ	説明	イベント名	イベントタイプ	優先度	重大度
President restarts arbitration thread [state=state_code] または Prepare arbitrator node node_id [ticket=ticket_id] または Receive arbitrator node node_id [ticket=ticket_id] または Started arbitrator node node_id [ticket=ticket_id] または Lost arbitrator node node_id - process failure [state=state_code] または Lost arbitrator node node_id - process exit [state=state_code] または Lost arbitrator node node_id - error_message [state=state_code]	これは、クラスタ内のアービトレーションの現在の状態および進行状況に関するレポートです。node_id は、アービレータとして選択された管理ノードまたは SQL ノードのノード ID です。state_code は、include/kernel/signaldata/ArbitSignalData.hpp で見つかるアービトレーション状態コードです。エラーが発生すると、ArbitSignalData.hpp でも定義されている error_message が表示されます。ticket_id は、アービレータが選択された時にそれによって、その選択に参加したすべてのノードに渡された一意の識別子です。これは、アービトレーションをリクエストする各ノードが選択プロセスに参加したノードのいずれかであることを確認するために使用されます。	ArbitState	NodeRestart	6	INFO
Arbitration check lost - less than 1/2 nodes left または Arbitration check won - all node groups and more than 1/2 nodes left または Arbitration check won - node group majority または Arbitration check lost - missing node group または Network partitioning - arbitration required または	このメッセージは、アービトレーションの結果についてレポートします。アービトレーションに失敗した場合、error_message およびアービトレーション state_code が表示されます。これらの両方の定義は、include/kernel/signaldata/ArbitSignalData.hpp で見つかります。	ArbitResult	NodeRestart	2	ALERT

ログメッセージ	説明	イベント名	イベントタイプ	優先度	重大度
Arbitration won - positive reply from node node_id または Arbitration lost - negative reply from node node_id または Network partitioning - no arbitrator available または Network partitioning - no arbitrator configured または Arbitration failure - error_message [state=state_code]					
Node node_id: GCP Take over started	このノードは、次のグローバルチェックポイントに対する責任を負おうとしています (つまり、マスターノードになります)。	GCP_TakeoverStarted	NodeRestart	7	INFO
Node node_id: GCP Take over completed	このノードはマスターになり、次のグローバルチェックポイントに対する責任を負いました。	GCP_TakeoverCompleted	NodeRestart	7	INFO
Node node_id: LCP Take over started	このノードは、次のグローバルチェックポイントセットに対する責任を負おうとしています (つまり、マスターノードになります)。	LCP_TakeoverStarted	NodeRestart	7	INFO
Node node_id: LCP Take over completed	このノードはマスターになり、次のグローバルチェックポイントセットに対する責任を負いました。	LCP_TakeoverCompleted	NodeRestart	7	INFO
Node node_id: Trans. Count = transactions, Commit Count = commits, Read Count = reads, Simple Read Count = simple_reads, Write Count = writes,	このトランザクションアクティビティのレポートは、約 10 秒に 1 回表示されます	TransReportCount	Statistic	8	INFO

ログメッセージ	説明	イベント名	イベントタイプ	優先度	重大度
AttrInfo Count = AttrInfo_objects, Concurrent Operations = concurrent_operations, Abort Count = aborts, Scans = scans, Range scans = range_scans					
Node node_id: Operations=operations	このノードで実行された操作数です (約 10 秒に 1 回表示されます)	OperationReportCount	Statistic	8	INFO
Node node_id: Table with ID = table_id created	示されたテーブル ID を持つテーブルが作成されました	TableCreated	Statistic	7	INFO
Node node_id: Mean loop Counter in doJob last 8192 times = count		JobStatistic	Statistic	9	INFO
Mean send size to Node = node_id last 4096 sends = bytes bytes	このノードはノード node_id への送信あたり、平均で bytes バイト送信しています	SendBytesStatistic	Statistic	9	INFO
Mean receive size to Node = node_id last 4096 sends = bytes bytes	このノードはノード node_id からデータを受信するたびに、平均で bytes バイトのデータを受信しています	ReceiveBytesStatistic	Statistic	9	INFO
Node node_id: Data usage is data_memory_percentage% (data_pages_used 32K pages of total data_pages_total) / Node node_id: Index usage is index_memory_percentage% (index_pages_used 8K pages of total index_pages_total)	このレポートは、クラスタ管理クライアントで DUMP 1000 コマンドが発行されると生成されます	MemoryUsage	Statistic	5	INFO
Node node1_id: Transporter to node node2_id reported error error_code: error_message	ノード node2_id との通信中に、トランスポータのエラーが発生しました。トランスポータのエラーコードおよびメッセージのリストについては、MySQL NDB	TransporterError	Error	2	ERROR

ログメッセージ	説明	イベント名	イベントタイプ	優先度	重大度
	Cluster Internals Manual のNDB Transporter Errorsを参照してください。				
Node node1_id: Transporter to node node2_id reported error error_code: error_message	ノード node2_id との通信中の潜在的なトランスポータ問題の警告。トランスポータのエラーコードおよびメッセージのリストについて詳しくは、 NDB Transporter Errors を参照してください。	TransporterWarningError		8	WARNING
Node node1_id: Node node2_id missed heartbeat heartbeat_id	このノードはノード node2_id からのハートビートを受信できませんでした	MissedHeartbeat	Error	8	WARNING
Node node1_id: Node node2_id declared dead due to missed heartbeat	このノードは、ノード node2_id からの少なくとも3つのハートビートを受信できなかったため、そのノードの「停止」を宣言しました	DeadDueToHeartbeatError	Error	8	ALERT
Node node1_id: Node Sent Heartbeat to node = node2_id	このノードはノード node2_id にハートビートを送信しました	SentHeartbeat	Info	12	INFO
Node node_id: Event buffer status (object_id): used=bytes_used (percent_used% of alloc) alloc=bytes_allocated max=bytes_available latest_consumed_epoch latest_buffered_epoch report_reason=report_reason	このレポートは、比較的短い期間に多数の更新が適用されている場合など、イベントバッファの使用中に表示されます。このレポートには、使用されているイベントバッファのバイト数と割合、割り当てられているバイト数と使用可能な割合、および最新のバッファ済みブロックと消費済みブロックが表示されます。詳細は、 セクション23.5.2.3「クラ	EventBufferStatus2	Info	7	INFO

ログメッセージ	説明	イベント名	イベントタイプ	優先度	重大度
	スタログでのイベントバッファのレポート」を参照してください				
Node node_id: Entering single user mode, Node node_id: Entered single user mode Node API_node_id has exclusive access, Node node_id: Entering single user mode	これらのレポートは、シングルユーザーモードの開始時および終了時にクラスタログに書き込まれます。API_node_id は、クラスタへの排他アクセス権を持っている API または SQL のノード ID です (詳細は、 セクション 23.5.6 「NDB Cluster のシングルユーザーモード」 を参照してください)。「Unknown single user report API_node_id」というメッセージは、エラーが発生したことを示し、通常の操作では表示されません。	SingleUser	Info	7	INFO
Node node_id: Backup backup_id started from node mgm_node_id	mgm_node_id を持つ管理ノードを使用して、バックアップが開始されました。このメッセージは、START BACKUP コマンドの発行時にクラスタ管理クライアントにも表示されます。詳細は、 セクション 23.5.8.2 「NDB Cluster 管理クライアントを使用したバックアップの作成」 を参照してください。	BackupStarted	Backup	7	INFO
Node node_id: Backup backup_id started from node mgm_node_id completed. StartGCP: start_gcp StopGCP: stop_gcp	ID backup_id を持つバックアップが完了しました。詳細は、 セクション 23.5.8.2 「NDB Cluster 管理クライアントを使用したバックアップの作成」 を参照してください。	BackupCompleted	Backup	7	INFO

ログメッセージ	説明	イベント名	イベントタイプ	優先度	重大度
#Records: records #LogRecords: log_records Data: data_bytes bytes Log: log_bytes bytes	作成」を参照してください				
Node node_id: Backup request from mgm_node_id failed to start. Error: error_code	バックアップの開始に失敗しました。エラーコードについては、MGM API Errorsを参照してください	BackupFailedToStart	Backup	7	ALERT
Node node_id: Backup backup_id started from mgm_node_id has been aborted. Error: error_code	バックアップが開始後に終了しました。ユーザーの介入に原因がある可能性があります	BackupAborted	Backup	7	ALERT

23.5.2.2 NDB Cluster のログ起動メッセージ

可能な起動メッセージとその説明を次のリストに示します:

- Initial start, waiting for %s to connect, nodes [all: %s connected: %s no-wait: %s]
- Waiting until nodes: %s connects, nodes [all: %s connected: %s no-wait: %s]
- Waiting %u sec for nodes %s to connect, nodes [all: %s connected: %s no-wait: %s]
- Waiting for non partitioned start, nodes [all: %s connected: %s missing: %s no-wait: %s]
- Waiting %u sec for non partitioned start, nodes [all: %s connected: %s missing: %s no-wait: %s]
- Initial start with nodes %s [missing: %s no-wait: %s]
- Start with all nodes %s
- Start with nodes %s [missing: %s no-wait: %s]
- Start potentially partitioned with nodes %s [missing: %s no-wait: %s]
- Unknown startreport: 0x%x [%s %s %s %s]

23.5.2.3 クラスタログでのイベントバッファのレポート

NDB は、データノードから受信したイベントに 1 つ以上のメモリーバッファを使用します。このようなバッファは、テーブルイベントをサブスクライブする Ndb オブジェクトごとに存在します。つまり、通常、バイナリロギングを実行する mysqld ごとにバッファが 2 つあります (スキーマイベント用にバッファが 1 つ、データイベント用にバッファが 1 つ)。各バッファには、イベントで構成されるエポックが含まれます。これらのイベントは、操作タイプ (挿入、更新、削除) および行データ (ビフォアイメージとアフターイメージおよびメタデータ) で構成されます。

NDB は、これらのバッファの状態を記述するメッセージをクラスタログに生成します。これらのレポートはクラスタログに表示されますが、API ノード上のバッファを参照します (データノードによって生成される他のほとんどのクラスタログメッセージとは異なります)。

クラスタログのイベントバッファロギングレポートでは、次に示す形式を使用します:

```
Node node_id: Event buffer status (object_id):
used=bytes_used (percent_used% of alloc)
alloc=bytes_allocated (percent_alloc% of max) max=bytes_available
latest_consumed_epoch=latest_consumed_epoch
```

```
latest_buffered_epoch=latest_buffered_epoch
report_reason=report_reason
```

このレポートを構成するフィールドとその説明を次に示します:

- **node_id**: レポートが生成されたノードの ID。
- **object_id**: レポートが生成された **Ndb** オブジェクトの ID。
- **bytes_used**: バッファで使用されるバイト数。
- **percent_used**: 使用された割当て済バイトの割合。
- **bytes_allocated**: このバッファに割り当てられたバイト数。
- **percent_alloc**: 使用可能なバイト数の割合。 **ndb_eventbuffer_max_alloc** が 0 (無制限) の場合は出力されません。
- **bytes_available**: 使用可能なバイト数。 **ndb_eventbuffer_max_alloc** が 0 (無制限) の場合は 0 です。
- **latest_consumed_epoch**: 完了までに最後に消費されたエポック。(NDB API アプリケーションでは、これは **nextEvent()** を呼び出すことによって行われます。)
- **latest_buffered_epoch**: イベントバッファで最後にバッファされた (完全にバッファされた) エポック。
- **report_reason**: レポートを作成する理由。考えられる原因をこのセクションの後半に示します。

次のリストに、レポートの考えられる理由を示します:

- **ENOUGH_FREE_EVENTBUFFER**: イベントバッファに十分な領域があります。

LOW_FREE_EVENTBUFFER: イベントバッファの空き領域が少なくなっています。

これらのレポートをトリガーするしきい値の空き率レベルは、 **ndb_report_thresh_binlog_mem_usage** サーバー変数を設定することで調整できます。

- **BUFFERED_EPOCHS_OVER_THRESHOLD**: バッファされたエポックの数が構成済のしきい値を超えたかどうか。この数は、完全に受信された最新のエポックと最近消費されたエポックの違いです (NDB API アプリケーションでは、これは **nextEvent()** または **nextEvent2()** を呼び出すことによって行われます)。レポートは、バッファされたエポックの数がしきい値を下回るまで毎秒生成され、 **ndb_report_thresh_binlog_epoch_slip** サーバー変数を設定して調整できます。NDB API アプリケーションのしきい値は、 **setEventBufferQueueEmptyEpoch()** を呼び出すことによって調整することもできます。
- **PARTIALLY_DISCARDING**: イベントバッファメモリーを使い果たしました。つまり、 **ndb_eventbuffer_max_alloc** の 100% が使用されています。部分的にバッファされたエポックは、使用率が 100% を超えても完了までバッファされますが、受信した新しいエポックは破棄されます。つまり、イベントストリームでギャップが発生しています。
- **COMPLETELY_DISCARDING**: エポックはバッファリングされません。
- **PARTIALLY_BUFFERING**: ギャップに続くバッファの空き率がしきい値まで上昇しました。 **ndb_eventbuffer_free_percent** サーバーシステム変数を使用するか、NDB API アプリケーションで **set_eventbuffer_free_percent()** を呼び出すことによって **mysql** クライアントで設定できます。新しいエポックがバッファされます。ギャップのために完了できなかった Epoch は破棄されます。
- **COMPLETELY_BUFFERING**: 受信したすべてのエポックがバッファされています。これは、十分なイベントバッファメモリーがあることを意味します。イベントストリーム内のギャップがクローズされました。

23.5.2.4 NDB Cluster: NDB トランスポータエラー

このセクションでは、トランスポータエラーの発生時にクラスタログに書き込まれるエラーのコード、名前、およびメッセージを一覧表示します。

```
0x00    TE_NO_ERROR
```

```
        No error
```


0x01	TE_ERROR_CLOSING_SOCKET	Error found during closing of socket
0x02	TE_ERROR_IN_SELECT_BEFORE_ACCEPT	Error found before accept. The transporter will retry
0x03	TE_INVALID_MESSAGE_LENGTH	Error found in message (invalid message length)
0x04	TE_INVALID_CHECKSUM	Error found in message (checksum)
0x05	TE_COULD_NOT_CREATE_SOCKET	Error found while creating socket(can't create socket)
0x06	TE_COULD_NOT_BIND_SOCKET	Error found while binding server socket
0x07	TE_LISTEN_FAILED	Error found while listening to server socket
0x08	TE_ACCEPT_RETURN_ERROR	Error found during accept(accept return error)
0x0b	TE_SHM_DISCONNECT	The remote node has disconnected
0x0c	TE_SHM_IPC_STAT	Unable to check shm segment
0x0d	TE_SHM_UNABLE_TO_CREATE_SEGMENT	Unable to create shm segment
0x0e	TE_SHM_UNABLE_TO_ATTACH_SEGMENT	Unable to attach shm segment
0x0f	TE_SHM_UNABLE_TO_REMOVE_SEGMENT	Unable to remove shm segment
0x10	TE_TOO_SMALL_SIGID	Sig ID too small
0x11	TE_TOO_LARGE_SIGID	Sig ID too large
0x12	TE_WAIT_STACK_FULL	Wait stack was full
0x13	TE_RECEIVE_BUFFER_FULL	Receive buffer was full

0x14	TE_SIGNAL_LOST_SEND_BUFFER_FULL	Send buffer was full,and trying to force send fails
0x15	TE_SIGNAL_LOST	Send failed for unknown reason(signal lost)
0x16	TE_SEND_BUFFER_FULL	The send buffer was full, but sleeping for a while solved
0x21	TE_SHM_IPC_PERMANENT	Shm ipc Permanent error

注記

0x20 および 0x22 を介したトランスポータエラーコード 0x17 は SCI 接続用に予約されており、NDB Cluster のこのバージョンではサポートされていないため、ここには含まれていません。

23.5.3 NDB Cluster で生成されるイベントレポート

このセクションでは、NDB Cluster によって提供されるイベントログのタイプと、ログに記録されるイベントのタイプについて説明します。

NDB Cluster には、次の 2 種類のイベントログが用意されています:

- すべてのクラスタノードで生成されたイベントが含まれるクラスタログ。クラスタログは、単一の場所でクラスタ全体に関するロギング情報を提供するため、ほとんどの使用目的で推奨されるログです。

デフォルトでは、クラスタログは `ndb_node_id_cluster.log` という名前のファイル (`node_id` は管理サーバーのノード ID) に管理サーバー `DataDir` に保存されます。

クラスタのロギング情報は、`DataDir` および `LogDestination` 構成パラメータに設定された値によって指定されたファイルに保存することに加えて、またはその代わりに、`stdout` または `syslog` 機能に送信することもできます。これらのパラメータについての詳細は、[セクション23.3.3.5「NDB Cluster 管理サーバーの定義」](#)を参照してください。

- ノードログは、各ノードにローカルです。

ノードイベントロギングによって生成された出力は、ノード `DataDir` の `ndb_node_id_out.log` (`node_id` はノードノード ID) ファイルに書き込まれます。ノードイベントログは、管理ノードとデータノードの両方で生成されます。

ノードログは、アプリケーションの開発時またはアプリケーションコードのデバッグに使用されることのみを目的としています。

両方のタイプのイベントログを設定すると、イベントのさまざまなサブセットのログを記録できます。

レポート可能な各イベントは、次の 3 つの基準に従って区別できます。

- カテゴリ: これは、`STARTUP`、`SHUTDOWN`、`STATISTICS`、`CHECKPOINT`、`NODERESTART`、`CONNECTION`、`ERROR`、または `INFO` 値のいずれかを指定できます。
- 優先度: これは、0 から 15 までの数値のいずれかで表されます。0 は「最も重要」および 15 「重要度が最も低い」を示します。
- 重大度レベル: これは、`ALERT`、`CRITICAL`、`ERROR`、`WARNING`、`INFO`、または `DEBUG` 値のいずれかを指定できます。

クラスタログとノードログのどちらもこれらのプロパティでフィルタ処理できます。

クラスタログで使用される形式は、次に示すとおりです。

```
2007-01-26 19:35:55 [MgmSrvr] INFO -- Node 1: Data usage is 2%(60 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO -- Node 1: Index usage is 1%(24 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO -- Node 1: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:35:55 [MgmSrvr] INFO -- Node 2: Data usage is 2%(76 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO -- Node 2: Index usage is 1%(24 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO -- Node 2: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:35:55 [MgmSrvr] INFO -- Node 3: Data usage is 2%(58 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO -- Node 3: Index usage is 1%(25 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO -- Node 3: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:35:55 [MgmSrvr] INFO -- Node 4: Data usage is 2%(74 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO -- Node 4: Index usage is 1%(25 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO -- Node 4: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:39:42 [MgmSrvr] INFO -- Node 4: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO -- Node 1: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO -- Node 1: Node 9: API 8.0.23-ndb-8.0.23
2007-01-26 19:39:42 [MgmSrvr] INFO -- Node 2: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO -- Node 2: Node 9: API 8.0.23-ndb-8.0.23
2007-01-26 19:39:42 [MgmSrvr] INFO -- Node 3: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO -- Node 3: Node 9: API 8.0.23-ndb-8.0.23
2007-01-26 19:39:42 [MgmSrvr] INFO -- Node 4: Node 9: API 8.0.23-ndb-8.0.23
2007-01-26 19:59:22 [MgmSrvr] ALERT -- Node 2: Node 7 Disconnected
2007-01-26 19:59:22 [MgmSrvr] ALERT -- Node 2: Node 7 Disconnected
```

クラスタログの各行には、次の情報が含まれます。

- YYYY-MM-DD HH:MM:SS 形式のタイムスタンプ。
- ロギングを実行しているノードのタイプ。クラスタログでは、これは常に [MgmSrvr] です。
- イベントの重大度。
- イベントをレポートするノードの ID。
- イベントの説明。ログに表示されるもっとも一般的なイベントタイプは、クラスタ内のさまざまなノード間およびチェックポイントの発生時の接続と切断です。場合によっては、説明にステータス情報が含まれることがあります。

23.5.3.1 NDB Cluster ロギング管理コマンド

`ndb_mgm` は、クラスタログおよびノードログに関連する多数の管理コマンドをサポートしています。次のリストで、`node_id` はストレージノード ID またはキーワード `ALL` のいずれかを示します。これは、コマンドをすべてのクラスタデータノードに適用する必要があることを示します。

- `CLUSTERLOG ON`
クラスタログをオンにします。
- `CLUSTERLOG OFF`
クラスタログをオフにします。
- `CLUSTERLOG INFO`
クラスタログの設定に関する情報を提供します。
- `node_id CLUSTERLOG category=threshold`
`threshold` 以下の優先度を持つ `category` のイベントをクラスタログに記録します。
- `CLUSTERLOG FILTER severity_level`
指定された `severity_level` のイベントのクラスタロギングを切り換えます。

次の表では、クラスタログカテゴリのしきい値の (すべてのデータノードの) デフォルト設定について説明します。イベントの優先度が優先度のしきい値以下である場合は、そのイベントがクラスタログにレポートされます。

注記

イベントはデータノードごとに報告され、しきい値はノードごとに異なる値に設定できません。

表 23.52 クラスタログカテゴリ (デフォルトのしきい値設定)

カテゴリ	デフォルトのしきい値 (すべてのデータノード)
STARTUP	7
SHUTDOWN	7
STATISTICS	7
CHECKPOINT	7
NODERESTART	7
CONNECTION	7
ERROR	15
INFO	7

STATISTICS カテゴリは、役に立つデータを大量に提供できます。詳細は、[セクション23.5.3.3「NDB Cluster 管理クライアントでの CLUSTERLOG STATISTICS の使用」](#)を参照してください。

しきい値は、各カテゴリ内のイベントをフィルタ処理するために使用されます。たとえば、優先度が3の STARTUP イベントのログは、STARTUP のしきい値が3以上に設定されていなければ記録されません。しきい値が3の場合は、優先度が3以下のイベントのみが送信されます。

次の表は、イベントの重大度レベルを示しています。

注記

これらは、使用されることもマップされることもない LOG_EMERG と LOG_NOTICE を除く、Unix の syslog レベルに対応します。

表 23.53 イベントの重大度

重大度レベル値	重大度	説明
1	ALERT	すぐに修正すべき状況 (システムデータベースの破損など)
2	CRITICAL	クリティカルな状況 (デバイスエラーやリソース不足など)
3	ERROR	修正すべき状況 (構成エラーなど)
4	WARNING	エラーではないが、特別な処理が必要な可能性がある状況
5	INFO	情報メッセージ
6	DEBUG	NDBCLUSTER の開発で 사용되는デバッグメッセージ

イベント重大度レベルは、[CLUSTERLOG FILTER](#) を使用することでオンとオフを切り替えることができます (上記を参照してください)。重大度レベルをオンにすると、優先度がカテゴリしきい値以下であるすべてのイベントがログに記録されます。重大度レベルをオフにすると、その重大度レベルに属するイベントはログに記録されません。

重要

クラスタログレベルは、`ndb_mgmd` ごとに、スクライバ単位で設定されます。つまり、複数の管理サーバーを持つ NDB Cluster では、ある管理サーバーに接続されている `ndb_mgmd` のインスタンスで `CLUSTERLOG` コマンドを使用すると、その管理サーバーによって生成されたログにのみ影響し、ほかの管理サーバーによって生成されたログには影響しません。

また、管理サーバーの 1 台が再起動されると、その管理サーバーで生成されたログのみが、再起動によって発生したログレベルのリセットによる影響を受けます。

23.5.3.2 NDB Cluster ログイベント

イベントログでレポートされるイベントレポートの形式は、次のとおりです。

```
datetime [string] severity -- message
```

例:

```
09:19:30 2005-07-24 [NDB] INFO -- Node 4 Start phase 4 completed
```

このセクションでは、レポート可能なすべてのイベントについて、カテゴリおよび各カテゴリ内の重大度レベルの順に説明します。

イベントの説明では、GCP と LCP はそれぞれ「グローバルチェックポイント」および「ローカルチェックポイント」を表します。

CONNECTION イベント

これらのイベントは、クラスタノード間の接続に関連するものです。

表 23.54 クラスタノード間の接続に関連付けられたイベント

イベント	優先度	重大度レベル	説明
Connected	8	INFO	データノードが接続されました
Disconnected	8	ALERT	データノードが切断されました
CommunicationClosed	8	INFO	SQL ノードまたはデータノードの接続が閉じられました
CommunicationOpened	8	INFO	SQL ノードまたはデータノードの接続が開かれました
ConnectedApiVersion	8	INFO	API バージョンを使用した接続

CHECKPOINT イベント

次に示すロギングメッセージは、チェックポイントに関連するものです。

表 23.55 チェックポイントに関連付けられたイベント

イベント	優先度	重大度レベル	説明
GlobalCheckpointStarted	9	INFO	GCP の開始: Redo ログがディスクに書き込まれます
GlobalCheckpointCompleted	10	INFO	GCP が終了しました
LocalCheckpointStarted	7	INFO	LCP の開始: データがディスクに書き込まれます
LocalCheckpointCompleted	7	INFO	LCP が正常に完了しました
LCPStoppedInCalcKeepGci	0	ALERT	LCP が停止しました
LCPFragmentCompleted	11	INFO	フラグメント上の LCP が完了しました
UndoLogBlocked	7	INFO	Undo ロギングがブロックされました。バッファアの

イベント	優先度	重大度レベル	説明
			オーバーフローに近付いています
RedoStatus	7	INFO	Redo ステータス

STARTUP イベント

ノードまたはクラスタの起動や、その成功または失敗に対応して、次のイベントが生成されます。それらは、起動プロセスの進行状況に関する情報(ロギングアクティビティーに関する情報を含む)も提供します。

表 23.56 ノードまたはクラスタの起動に関連するイベント

イベント	優先度	重大度レベル	説明
NDBStartStarted	1	INFO	データノードの起動フェーズが開始されました(すべてのノードが起動します)
NDBStartCompleted	1	INFO	起動フェーズが完了しました、すべてのデータノード
STTORRYRecieved	15	INFO	再起動の完了後にブロックを受信しました
StartPhaseCompleted	4	INFO	データノードの起動フェーズ X が完了しました
CM_REGCONF	3	INFO	ノードが正常にクラスタに追加されました。ノード、管理ノード、および動的 ID が表示されます
CM_REGREF	8	INFO	ノードはクラスタへの追加を拒否されました。構成が間違っている、通信を確立できないなどの問題が原因で、クラスタに追加できません
FIND_NEIGHBOURS	8	INFO	隣接したデータノードを表示します
NDBStopStarted	1	INFO	データノードのシャットダウンが開始されました
NDBStopCompleted	1	INFO	データノードのシャットダウンが完了しました
NDBStopForced	1	ALERT	データノードが強制的にシャットダウンされました
NDBStopAborted	1	INFO	データノードを正常にシャットダウンできませんでした
StartREDOLog	4	INFO	新しい Redo ログが開始されました。GCI は X を保持し、最新のリストア可能な GCI は Y です
StartLog	10	INFO	新しいログが開始されました。ログ部分は X、開始 MB は Y、終了 MB は Z です
UNDORecordsExecuted	15	INFO	Undo レコードが実行されました

イベント	優先度	重大度レベル	説明
StartReport	4	INFO	レポートが開始されました
LogFileInitStatus	7	INFO	ログファイルの初期化ステータス
LogFileInitCompStatus	7	INFO	ログファイルの完了ステータス
StartReadLCP	10	INFO	ローカルチェックポイントの読み取りを開始します
ReadLCPComplete	10	INFO	ローカルチェックポイントの読み取りが完了しました
RunRedo	8	INFO	Redo ログが実行されています
RebuildIndex	10	INFO	インデックスを再構築しています

NODERESTART イベント

次のイベントは、ノードを再起動するときに生成され、ノードの再起動プロセスの成功または失敗に関連するものです。

表 23.57 ノードの再起動に関連するイベント

イベント	優先度	重大度レベル	説明
NR_CopyDict	7	INFO	ディレクトリ情報のコピーが完了しました
NR_CopyDistr	7	INFO	ディストリビューション情報のコピーが完了しました
NR_CopyFragStarted	7	INFO	フラグメントのコピーを開始しています
NR_CopyFragDone	10	INFO	フラグメントのコピーが完了しました
NR_CopyFragCompleted	7	INFO	すべてのフラグメントのコピーが完了しました
NodeFailCompleted	8	ALERT	ノードの失敗フェーズが完了しました
NODE_FAILREP	8	ALERT	ノードが失敗したことをレポートします
ArbitState	6	INFO	<p>アービトラータが見つかったかどうかをレポートします。アービトラータの検索時の可能性のある結果は、次に示す 7 つです。</p> <ul style="list-style-type: none"> 管理サーバーがアービトラーションスレッドを再起動します [状態 =X] アービトラータノード X を準備します [チケット =Y] アービトラータノード X を受信します [チケット =Y]

イベント	優先度	重大度レベル	説明
			<ul style="list-style-type: none"> • アービトレータノード X を起動しました [チケット=Y] • アービトレータノード X が失われました - プロセスの失敗 [状態=Y] • アービトレータノード X が失われました - プロセスの終了 [状態=Y] • アービトレータノード X が失われました <エラーメッセージ> [状態=Y]
ArbitResult	2	ALERT	<p>アービトレータの結果をレポートします。アービトレータの試行の可能性のある結果は、次に示した 8 つです。</p> <ul style="list-style-type: none"> • アービトレーションチェックに失敗しました: 1/2 未満のノードが残っています • アービトレーションチェックに成功しました: ノードグループの大部分 • アービトレーションチェックに失敗しました: ノードグループが見つかりません • ネットワークのパーティション化: アービトレーションが必要です • アービトレーションに成功しました: ノード X からの肯定的応答 • アービトレーションに失敗しました: ノード X からの否定的応答 • ネットワークのパーティション化: 使用可能なアービトレータがありません • ネットワークのパーティション化: アービトレータが構成されていません
GCP_TakeoverStarted	7	INFO	GCP テイクオーバーが開始されました
GCP_TakeoverCompleted	7	INFO	GCP テイクオーバーが完了しました

イベント	優先度	重大度レベル	説明
LCP_TakeoverStarted	7	INFO	LCP テイクオーバーが開始されました
LCP_TakeoverCompleted	7	INFO	LCP テイクオーバーが完了しました (状態 = X)
ConnectCheckStarted	6	INFO	接続チェックが開始されました
ConnectCheckCompleted	6	INFO	接続チェックが完了しました
NodeFailRejected	6	ALERT	ノードの失敗フェーズに失敗しました

STATISTICS イベント

次のイベントは、統計的な特性を持っています。それらは、トランザクションやその他の操作の回数、各ノードによって送受信されたデータの量、メモリーの使用状況などの情報を提供します。

表 23.58 統計的性質のイベント

イベント	優先度	重大度レベル	説明
TransReportCounters	8	INFO	トランザクション、コミット、読み取り、単純な読み取り、書き込み、同時操作、属性情報、中止の数などのトランザクション統計をレポートします
OperationReportCounters	8	INFO	操作数
TableCreated	7	INFO	作成されたテーブルの数をレポートします
JobStatistic	9	INFO	内部ジョブスケジューリングの平均の統計
ThreadConfigLoop	9	INFO	スレッド構成ループの数
SendBytesStatistic	9	INFO	ノード X に送信された平均バイト数
ReceiveBytesStatistic	9	INFO	ノード X から受信された平均バイト数
MemoryUsage	5	INFO	データおよびインデックスメモリー使用状況 (80%、90%、および 100%)
MTSignalStatistics	9	INFO	マルチスレッドシグナル

SCHEMA イベント

これらのイベントは NDB Cluster スキーマ操作に関連しています。

表 23.59 NDB Cluster スキーマ操作に関連するイベント

イベント	優先度	重大度レベル	説明
CreateSchemaObject	8	INFO	スキーマオブジェクトが作成されました
AlterSchemaObject	8	INFO	スキーマオブジェクトが更新されました
DropSchemaObject	8	INFO	スキーマオブジェクトが削除されました

ERROR イベント

これらのイベントは、クラスタのエラーおよび警告に関連するものです。一般に、これらが1つ以上存在することは、重大な誤動作や障害が発生したことを示しています。

表 23.60 クラスタのエラーおよび警告に関連するイベント

イベント	優先度	重大度レベル	説明
TransporterError	2	ERROR	トランスポータエラー
TransporterWarning	8	WARNING	トランスポータ警告
MissedHeartbeat	8	WARNING	ノード X でハートビート番号 Y が失われました
DeadDueToHeartbeat	8	ALERT	ハートビートが失われたため、ノード X が「停止」を宣言しました
WarningEvent	2	WARNING	一般警告イベント
SubscriptionStatus	4	WARNING	サブスクリプションのステータスの変更

INFO イベント

これらのイベントは、クラスタの状態およびロギングやハートビート伝送などのクラスタのメンテナンスに関連するアクティビティーに関する一般情報を提供します。

表 23.61 情報イベント

イベント	優先度	重大度レベル	説明
SentHeartbeat	12	INFO	ハートビートを送信しました
CreateLogBytes	11	INFO	ログの作成: ログ部分、ログファイル、MB 単位のサイズ
InfoEvent	2	INFO	一般情報イベント
EventBufferStatus	7	INFO	イベントバッファーステータス
EventBufferStatus2	7	INFO	イベントバッファーステータス情報の改善

注記

`SentHeartbeat` イベントは、NDB Cluster が `VM_TRACE` を有効にしてコンパイルされた場合にのみ使用できます。

SINGLEUSER イベント

これらのイベントは、シングルユーザーモードの開始と終了に関連するものです。

表 23.62 シングルユーザーモードに関連するイベント

イベント	優先度	重大度レベル	説明
SingleUser	7	INFO	シングルユーザーモードを開始または終了しています

BACKUP イベント

これらのイベントは、作成またはリストアされるバックアップに関する情報を提供します。

表 23.63 バックアップイベント

イベント	優先度	重大度レベル	説明
BackupStarted	7	INFO	バックアップが開始されました
BackupStatus	7	INFO	バックアップのステータス
BackupCompleted	7	INFO	バックアップが完了しました
BackupFailedToStart	7	ALERT	バックアップの開始に失敗しました
BackupAborted	7	ALERT	バックアップがユーザーによって中止されました
RestoreStarted	7	INFO	バックアップからのリストアが開始されました
RestoreMetaData	7	INFO	メタデータをリストアしています
RestoreData	7	INFO	データをリストアしています
RestoreLog	7	INFO	ログファイルをリストアしています
RestoreCompleted	7	INFO	バックアップからのリストアが完了しました
SavedEvent	7	INFO	イベントが保存されました

23.5.3.3 NDB Cluster 管理クライアントでの CLUSTERLOG STATISTICS の使用

NDB 管理クライアントの `CLUSTERLOG STATISTICS` コマンドは、その出力に多くの有用な統計を提供できます。クラスタの状態に関する情報を提供するカウンタは、トランザクションコーディネータ (TC) およびローカルクエリハンドラ (LQH) によって 5 秒のレポート間隔で更新され、クラスタログに書き込まれます。

トランザクションコーディネータの統計。各トランザクションには、1 つのトランザクションコーディネータがあり、それは次の方法のいずれかによって選択されます。

- ラウンドロビン方式で
- 通信の近さによって
- トランザクションの開始時にデータ配置ヒントを指定

注記

`ndb_optimized_node_selection` システム変数を使用すると、特定の SQL ノードから起動されたトランザクションに使用される TC の選択方法を判断できます。

同じトランザクション内の操作ではすべて、同じトランザクションコーディネータが使用され、次のような統計がレポートされます。

- Trans count. これは、この TC をトランザクションコーディネータとして使用して、最後の期間で開始されたトランザクション数です。これらのトランザクションのいずれかは、レポート期間の最後でコミットされた、中止された、または未コミットのままの状態になっている可能性があります。

注記

トランザクションは TC 間で移行しません。

- Commit count. これは、最後のレポート期間でコミットされた、この TC をトランザクションコーディネータとして使用したトランザクション数です。このレポート期間でコミットされた一部のトランザクションは、前のレポート期間で開始された可能性があるため、Commit count が Trans count よりも大きくなる可能性もあります。

- **Read count.** これは、最後のレポート期間で開始された、この TC をトランザクションコーディネータとして使用した主キーの読み取り操作 (単純な読み取りを含む) 数です。このカウントには、一意のインデックス操作の一部として実行される読み取りも含まれます。一意のインデックス読み取り操作では、2 つの主キー読み取り操作 (非表示の一意のインデックステーブルに対して 1 つ、および読み取りが行われるテーブルに対して 1 つ) が発生しません。
- **Simple read count.** これは、最後のレポート期間で開始された、この TC をトランザクションコーディネータとして使用した単純な読み取り操作数です。
- **Write count.** これは、最後のレポート期間で開始された、この TC をトランザクションコーディネータとして使用した主キーの書き込み操作数です。これには、一意のインデックス操作の一部として実行される書き込みに加えて、すべての挿入、更新、書き込み、および削除も含まれます。

注記

一意のインデックスの更新操作では、インデックステーブルおよびベーステーブルで、複数の PK 読み取りおよび書き込み操作が発生する可能性があります。

- **AttrInfoCount.** これは、この TC をトランザクションコーディネータとして使用した主キー操作の最後のレポート期間で受信された 32 ビットのデータ語の数です。読み取りの場合、これはリクエストされたカラムの数に比例します。挿入および更新の場合、これは書き込まれたカラムの数、およびそれらのデータのサイズに比例します。削除操作の場合、これは通常ゼロです。

一意のインデックス操作では、複数の PK 操作が発生するため、このカウントが増加します。ただし、ここでは、PK 操作自体を記述するために送信されたデータ語、および送信されたキー情報はカウントされません。スキャン用に読み取られるカラムを記述するため、または ScanFilters を記述するために送信された属性情報も、AttrInfoCount ではカウントされません。

- **Concurrent Operations.** これは、最後のレポート期間中に開始されたが完了しなかった、この TC をトランザクションコーディネータとして使用した主キーまたはスキャン操作数です。このカウンタは、操作が開始されるとインクリメントされ、操作が完了するとデクリメントされます。これは、トランザクションのコミット後に行われます。このカウンタは失敗した操作だけでなく、ダーティー読み取りおよび書き込みでもデクリメントされます。

Concurrent Operations に指定可能な最大値は、TC ブロックでサポートできる操作の最大数です。現在、これは $(2 * \text{MaxNoOfConcurrentOperations}) + 16 + \text{MaxNoOfConcurrentTransactions}$ です。(これらの構成パラメータについての詳細は、[セクション 23.3.3.6 「NDB Cluster データノードの定義」](#)の「トランザクションパラメータ」セクションを参照してください。)

- **Abort count.** これは、最後のレポート期間中に中止された、この TC をトランザクションコーディネータとして使用したトランザクション数です。最後のレポート期間で中止された一部のトランザクションは、前のレポート期間で開始された可能性があるため、Abort count が Trans count よりも大きくなる可能性もあります。
- **Scans.** これは、最後のレポート期間中に開始された、この TC をトランザクションコーディネータとして使用したテーブルスキャン数です。これには、範囲スキャン (つまり、順序付きインデックススキャン) は含まれません。
- **Range scans.** これは、最後のレポート期間で開始された、この TC をトランザクションコーディネータとして使用した順序付きインデックススキャン数です。
- **ローカル読み取り.** これは、レコードのプライマリフラグメントレプリカも保持するノードでトランザクションコーディネータを使用して実行される主キー読み取り操作の数です。この数は、ndbinfo.counters テーブルの LOCAL_READS カウンタからも取得できます。
- **ローカル書き込み.** これには、レコードのプライマリフラグメントレプリカも保持するノードでトランザクションコーディネータを使用して実行された主キー読み取り操作の数が含まれます。この数は、ndbinfo.counters テーブルの LOCAL_WRITES カウンタからも取得できます。

ローカルクエリーハンドラの統計 (操作). ローカルクエリーハンドラブロックごとに、1 つのクラスイベント (つまり、データノードプロセスごとに 1 つずつ) があります。操作は、それらが操作しているデータが存在する LQH に記録されます。

注記

単一のトランザクションが複数の LQH ブロックに格納されたデータを操作する場合もあります。

Operations 統計には、最後のレポート期間で、この LQH ブロックによって実行されたローカル操作数が表示され、すべてのタイプの読み取りおよび書き込み操作 (挿入、更新、書き込み、および削除の操作) が含まれます。これには、書き込みをレプリケートするために使用される操作も含まれます。たとえば、2 つのフラグメントレプリカを持つクラスタでは、プライマリフラグメントレプリカへの書き込みはプライマリ LQH に記録され、バックアップへの書き込みはバックアップ LQH に記録されます。一意のキー操作では、複数のローカル操作が発生する可能性があります。ただし、これには、テーブルスキャンまたは順序付きインデックススキャンの結果として発生するローカル操作は含まれず、カウントもされません。

プロセススケジューラの統計。各 `ndbd` プロセスには、トランザクションコーディネータおよびローカルクエリーハンドラによって報告される統計に加えて、NDB Cluster のパフォーマンスに関連する有用なメトリックも提供するスケジューラがあります。このスケジューラは、無限ループ時に実行されます。各ループ中に、スケジューラは次のタスクを実行します。

1. ソケットからジョブバッファに受信メッセージを読み込みます。
2. 時間指定のメッセージが実行されたかどうかをチェックします。実行された場合は、これらのメッセージもジョブバッファに配置します。
3. ジョブバッファ内のメッセージを (ループ内で) 実行します。
4. ジョブバッファ内のメッセージを実行することで生成された配信されるメッセージを送信します。
5. 新しい受信メッセージを待機します。

プロセススケジューラの統計には、次の情報が含まれています。

- Mean Loop Counter. これは、上記のリストの 3 番目のステップで実行されたループの数です。TCP/IP バッファの使用率が改善されると、この統計のサイズが増加します。これを使用すると、新しいデータノードプロセスを追加したときにパフォーマンスの変化をモニターできます。
- Mean send size と Mean receive size. これらの統計を使用すると、ノード間の書き込みと読み取りのそれぞれの効率性を測定できます。値はバイト単位で指定されます。値が大きいくほど、送受信される 1 バイト当たりのコストが小さくなることを意味します。最大値は 64K です。

NDB 管理クライアントで次のコマンドを使用すると、クラスタログの統計をすべて記録できます。

```
ndb_mgm> ALL CLUSTERLOG STATISTICS=15
```

注記

`STATISTICS` のしきい値を 15 に設定すると、クラスタログが非常に冗長になり、NDB Cluster 内のクラスタノードの数とアクティビティーの量に直接比例してサイズが大幅に大きくなります。

ロギングおよびレポートに関連する NDB Cluster 管理クライアントコマンドの詳細は、[セクション 23.5.3.1 「NDB Cluster ロギング管理コマンド」](#) を参照してください。

23.5.4 NDB Cluster 起動フェーズのサマリー

このセクションでは、NDB Cluster データノードの起動時に必要な手順の簡単な概要を示します。『[NDB Internals Guide](#)』の [NDB Cluster Start Phases](#) では、さらに完全な情報を見つけることができます。

これらのフェーズは、管理クライアントで `node_id STATUS` コマンドからの出力にレポートされるものと同じです ([セクション 23.5.1 「NDB Cluster 管理クライアントのコマンド」](#) を参照してください)。これらの起動フェーズは、`ndbinfo.nodes` テーブルの `start_phase` カラムにもレポートされます。

起動のタイプ。次のリストに示すように、さまざまな起動のタイプおよびモードがあります。

- 初期起動。すべてのデータノード上のクリーンなファイルシステムでクラスタが起動します。これは、まったくはじめてクラスタが起動される時、または `--initial` オプションを使用してすべてのデータノードが再起動される時に発生します。

注記

`--initial` を使用してノードを再起動すると、ディスクデータファイルが削除されません。

- システムの再起動. クラスタが起動し、データノードに格納されたデータを読み取ります。これは、クラスタが使用されたあとにシャットダウンされたとき、およびクラスタが操作を中止した時点から再開することが望ましいときに発生します。
- ノードの再起動. これは、クラスタ自体が実行しているときのクラスタノードのオンライン再起動です。
- ノードの初期再起動. これは、クリーンなファイルシステムでノードが再初期化および起動される点を除いて、ノードの再起動と同じです。

設定と初期化 (フェーズ -1). 起動する前に、各データノード (`ndbd` プロセス) が初期化されている必要があります。初期化は次のステップで構成されます。

1. ノード ID を取得します
2. 構成データをフェッチします
3. ノード間通信で使用されるポートを割り当てます
4. 構成ファイルから取得された設定に従って、メモリーを割り当てます

データノードまたは SQL ノードがはじめて管理ノードに接続すると、クラスタノード ID を予約します。ほかのノードによって同じノード ID が割り当てられないように、この ID は、ノードからクラスタへの接続が完了し、このノードが接続されたことが少なくとも 1 つの `ndbd` でレポートされるまで保持されます。このようなノード ID の保持は、該当するノードと `ndb_mgmd` 間の接続で保護されます。

各データノードが初期化されたら、クラスタの起動プロセスに進むことができます。このプロセス中にクラスタが進む段階を次に示します。

- フェーズ 0. `NDBFS` および `NDBCNTR` ブロックが起動します。 `-initial` オプションを付けて起動されたデータノード上で、データノードファイルシステムがクリアされます。
- フェーズ 1. この段階では、残りのすべての `NDB` カーネルブロックが起動されます。 `NDB Cluster` 接続が設定され、ブロック間通信が確立され、ハートビートが開始されます。ノードの再起動の場合は、API ノードの接続もチェックされます。

注記

フェーズ 1 で 1 つ以上のノードがハングアップし、フェーズ 2 で残りの 1 つまたは複数のノードがハングアップするときは、多くの場合にネットワークの問題を示しています。このような問題が発生する原因の 1 つとして、複数のネットワークインタフェースを持つ 1 つ以上のクラスタホストが考えられます。このような状況が発生する問題のもう 1 つの一般的な原因は、クラスタノード間の通信に必要な TCP/IP ポートのブロックです。後者では、これは多くの場合に、ファイアウォールが誤って構成されているためです。

- フェーズ 2. `NDBCNTR` カーネルブロックは、既存のすべてのノードの状態をチェックします。マスターノードが選択され、クラスタスキーマファイルが初期化されます。
- フェーズ 3. `DBLQH` および `DBTC` カーネルブロックは、それらの間の通信を設定します。起動タイプが特定されます。これが再起動の場合、`DBDIH` ブロックは再起動を実行するための権限を取得します。
- フェーズ 4. 初期起動またはノードの初期再起動の場合、Redo ログファイルが作成されます。これらのファイルの数は、`NoOfFragmentLogFiles` に等しいです。

システムの再起動の場合:

- 1 つまたは複数のスキーマを読み取ります。
- ローカルチェックポイントからデータを読み取ります。
- 最新のリストア可能なグローバルチェックポイントに達するまで、すべての Redo 情報を適用します。

ノードの再起動の場合、Redo ログの末尾を検索します。

- フェーズ 5. このフェーズ中に、データノード起動のデータベース関連部分のほとんどが実行されます。初期起動またはシステムの再起動の場合、ローカルチェックポイントに続いて、グローバルチェックポイントが実行され

ます。このフェーズ中に、メモリー使用率の定期的なチェックが開始され、必要なノードのテイクオーバーが実行されます。

- フェーズ 6. このフェーズでは、ノードグループが定義および設定されます。
- フェーズ 7. アービトラータノードが選択され、動作が開始されます。次のバックアップ ID、およびバックアップディスクの書き込み速度が設定されます。この起動フェーズに到達したノードは、**Started** とマークされます。この時点で、API ノード (SQL ノードを含む) はクラスタに接続できます。
- フェーズ 8. これがシステムの再起動の場合、すべてのインデックスが (DBDIH によって) 再構築されます。
- フェーズ 9. ノード内部の起動変数がリセットされます。
- フェーズ 100 (廃止). 以前は、ノードの再起動またはノードの初期再起動のこの時点で、API ノードはノードに接続し、イベントの受信を開始できました。現在は、このフェーズが空になっています。
- フェーズ 101. ノードの再起動またはノードの初期再起動のこの時点で、クラスタに参加するノードにイベント配信が渡されます。新たに参加したノードは、そのプライマリデータをサブスクリバに配信する責任を引き受けます。このフェーズは、**SUMA** ハンドオーバーフェーズとも呼ばれます。

初期起動またはシステムの再起動の場合、このプロセスが完了すると、トランザクションの処理が有効になります。ノードの再起動またはノードの初期再起動の場合、起動プロセスが完了したことは、現在ノードがトランザクションコーディネータとして動作している可能性があることを意味します。

23.5.5 NDB Cluster のローリング再起動の実行

このセクションでは、NDB Cluster インストールのローリング再起動を実行する方法について説明します。これは、クラスタ自体が動作したままになるように、各ノードを順番に停止して起動 (または再起動) するためです。これは多くの場合に、ローリングアップグレードまたはローリングダウングレードの一部として実行されます。ここでは、クラスタの高可用性が必須であり、クラスタ全体の停止時間は許可されません。一般に、ここで提供しているアップグレードについての情報は、ダウングレードにも適用されます。

ローリング再起動が望ましい理由は、いくつかあります。次のいくつかの段落で、これらについて説明します。

構成の変更.

クラスタへの SQL ノードの追加や、構成パラメータの新しい値への設定などのクラスタ構成の変更を行うため。

NDB Cluster ソフトウェアアップグレードまたはダウングレード. NDB Cluster ソフトウェアの新しいバージョンにクラスタをアップグレードする (または古いバージョンにダウングレードする)。これは通常、古いバージョンの NDB Cluster に戻す場合、「**ローリングアップグレード**」 (または「**ローリングダウングレード**」) と呼ばれます。

ノードホスト上での変更. 1 つまたは複数の NDB Cluster ノードプロセスが実行されているハードウェアまたはオペレーティングシステムに変更を加えるため。

システムのリセット (クラスタのリセット).

望ましくない状態に達したために、クラスタをリセットするため。このような場合は、多くの場合に 1 つ以上のデータノードのデータおよびメタデータを再ロードすることが望ましいと考えられます。これは、次の 3 つの方法のいずれかで実行できます。

- **--initial** オプションを使用して各データノードプロセス (**ndbd** または場合によっては **ndbmtbd**) を起動します。これにより、データノードはそのファイルシステムを強制的にクリアし、ほかのデータノードからすべての NDB Cluster データおよびメタデータをリロードします。

NDB 8.0.21 以降では、これらのオブジェクトに関連付けられたすべてのディスクデータオブジェクトおよびファイルも強制的に削除されます。

- 再起動を実行する前に、**ndb_mgm** クライアントの **START BACKUP** コマンドを使用してバックアップを作成します。アップグレード後に、**ndb_restore** を使用して、1 つまたは複数のノードをリストアします。

詳細は、[セクション 23.5.8 「NDB Cluster のオンラインバックアップ」](#) および [セクション 23.4.23 「ndb_restore — NDB Cluster バックアップの復元」](#) を参照してください。

- **mysqldump** を使用して、アップグレード前にバックアップを作成します。その後、**LOAD DATA** を使用してダンプをリストアします。

リソースのリカバリ。

他の「NDB Cluster」テーブルで再利用するために、INSERT および DELETE の連続した操作によって以前にテーブルに割り当てられたメモリーを解放します。

ローリング再起動を実行するプロセスは、次のように一般化できます。

1. すべてのクラスタ管理ノード (`ndb_mgmd` プロセス) を停止し、再構成し、再起動します。(複数の管理サーバーでのローリング再起動を参照してください。)
2. 各クラスタデータノード (`ndbd` プロセス) を順番に停止し、再構成し、再起動します。

一部のノード構成パラメータは、前のステップに従って `ndb_mgm` クライアントの各データノードに対して `RESTART` を発行することで更新できます。その他のパラメータでは、管理クライアントの `STOP` コマンドを使用してデータノードを完全に停止してから、必要に応じて `ndbd` または `ndbmt` 実行可能ファイルを呼び出してシステムシェルから再度起動する必要があります。(ほとんどの Unix システムでは、`kill` などのシェルコマンドを使用してデータノードのプロセスを停止することもできますが、`STOP` コマンドが推奨され、通常はより簡単です。)

注記

Windows では、`SC STOP` および `SC START` コマンド、`NET STOP` および `NET START` コマンド、または Windows サービスマネージャーを使用して、Windows サービスとしてインストールされているノードを停止および起動することもできます(セクション 23.2.2.4 「NDB Cluster プロセスを Windows サービスとしてインストール」を参照)。

必要な再起動のタイプは、各ノード構成パラメータのドキュメントに示されています。セクション 23.3.3 「NDB Cluster 構成ファイル」を参照してください。

3. 各クラスタ SQL ノード (`mysqld` プロセス) を順番に停止し、再構成し、再起動します。

NDB Cluster は、ノードをアップグレードするためある程度柔軟な順序をサポートしています。NDB Cluster をアップグレードするときは、管理ノード、データノード、またはその両方をアップグレードする前に、API ノード (SQL ノードを含む) をアップグレードできます。言い換えると、API ノードおよび SQL ノードを任意の順序でアップグレードすることが許可されています。これには、次のような条件が課せられます。

- この機能は、オンラインアップグレードの一部として使用することのみを目的としています。異なる NDB Cluster リリースからのノードバイナリの混在は、本番設定での継続的な長期的な使用を目的としておらず、サポートされていません。
- データノードをアップグレードする前に、すべての管理ノードをアップグレードする必要があります。このことは、クラスタの API ノードおよび SQL ノードをアップグレードする順序に関係なく当てはまります。
- すべての管理ノードおよびデータノードがアップグレードされるまで、「新しい」バージョンに固有の機能は使用しないでください。

このことは、NDB エンジンバージョンの変更に加えて、適用される可能性のある任意の MySQL サーバーバージョンの変更にも適用されます。したがって、アップグレードを計画する際には、これを考慮に入れることを忘れないでください。(これは、NDB Cluster のオンラインアップグレード全般に当てはまります。)

どの API ノードでも、ノードの再起動時にスキーマ操作 (データ定義ステートメントなど) を実行できません。この制限の一部であるため、オンラインでのアップグレードまたはダウングレード中にスキーマ操作もサポートされません。

複数の管理サーバーでのローリング再起動。複数の管理ノードを持つ NDB Cluster のローリング再起動を実行する場合、`ndb_mgmd` はほかの管理ノードが実行されているかどうかを確認し、実行されている場合はそのノード構成データを使用しようとすることに注意してください。これが発生することを回避し、`ndb_mgmd` にその構成ファイルを強制的に再読み取りさせるには、次のステップを実行します。

1. すべての NDB Cluster `ndb_mgmd` プロセスを停止します。
2. すべての `config.ini` ファイルを更新します。
3. 必要に応じて、`--reload`、`--initial`、またはその両方のオプションを付けて単一の `ndb_mgmd` を起動します。

4. 最初の `ndb_mgmd` を `--initial` オプションを付けて起動した場合は、残りの `ndb_mgmd` プロセスもすべて `--initial` を使用して起動する必要があります。

最初の `ndb_mgmd` を起動するときに使用されたその他のオプションに関係なく、最初の `ndb_mgmd` プロセスのあとに、`--reload` を使用して残りのプロセスを再起動しないようにしてください。

5. 通常どおりに、データノードおよび API ノードのローリング再起動を完了します。

ローリング再起動を実行してクラスタの構成を更新する際に、`ndbinfo.nodes` テーブルの `config_generation` カラムを使用して、新しい構成で正常に再起動されたデータノードを追跡できます。 [セクション23.5.14.38「ndbinfo nodes テーブル」](#) を参照してください。

23.5.6 NDB Cluster のシングルユーザーモード

シングルユーザーモードにより、データベース管理者はデータベースシステムへのアクセスを、MySQL サーバー (SQL ノード) や `ndb_restore` のインスタンスなど単一の API ノードに制限できます。シングルユーザーモードに入ると、その他のすべての API ノードへの接続が正常に閉じられ、実行中のトランザクションがすべて中止されます。新しいトランザクションの開始は許可されません。

クラスタがシングルユーザーモードに入ると、指定された API ノードにのみデータベースへのアクセス権が付与されます。

`ndb_mgm` クライアントで `ALL STATUS` コマンドを使用すると、クラスタがシングルユーザーモードに入ったタイミングを確認できます。また、`ndbinfo.nodes` テーブルの `status` カラムをチェックすることもできます (詳細は、[セクション23.5.14.38「ndbinfo nodes テーブル」](#) を参照してください)。

例:

```
ndb_mgm> ENTER SINGLE USER MODE 5
```

このコマンドが実行され、クラスタがシングルユーザーモードに入ると、ノード ID が 5 の API ノードがクラスタのみに許可されたユーザーになります。

前述のコマンドで指定したノードは API ノードである必要があります。他のタイプのノードを指定しようとすると拒否されます。

注記

前述のコマンドが呼び出されると、指定されたノードで実行されているすべてのトランザクションが中止され、接続が閉じられるため、サーバーを再起動する必要があります。

コマンド `EXIT SINGLE USER MODE` は、クラスタデータノードの状態をシングルユーザーモードから通常モードに変更します。接続を待機している (つまり、クラスタの準備ができ、使用可能になるまで待機している) MySQL サーバーなどの API ノードは、接続が再度許可されます。単一ユーザーノードとして指定された API ノードは、状態の変更中および変更後も引き続き実行されます (まだ接続されている場合)。

例:

```
ndb_mgm> EXIT SINGLE USER MODE
```

シングルユーザーモードでの実行時にノードの障害を処理する際には、次の 2 つの方法が推奨されています。

- 方法 1:
 1. シングルユーザーモードのトランザクションをすべて終了します
 2. `EXIT SINGLE USER MODE` コマンドを発行します
 3. クラスタのデータノードを再起動します
- 方法 2:

シングルユーザーモードに入る前に、ストレージノードを再起動します。

23.5.7 NDB Cluster データノードのオンラインでの追加

このセクションでは、NDB Cluster データノード「「オンライン」」を追加する方法について説明します。つまり、クラスタを完全にシャットダウンし、プロセスの一環として再起動する必要はありません。

重要

現在、新しいノードグループの一部として NDB Cluster に新しいデータノードを追加する必要があります。また、フラグメントレプリカの数 (またはノードグループごとのノード数) をオンラインに変更することはできません。

23.5.7.1 NDB Cluster データノードのオンラインでの追加: 一般的な問題

このセクションでは、NDB Cluster ノードをオンラインで追加する際の動作および現在の制限に関する一般的な情報を提供します。

データの再配布. 新しいノードをオンラインで追加する機能には、[ALTER TABLE ... REORGANIZE PARTITION](#) ステートメントを使用して、新しいノードを含むすべてのデータノードに分散されるように [NDBCLUSTER](#) テーブルのデータおよびインデックスを再編成する手段が含まれます。インメモリとディスクデータの両方のテーブルの再編成がサポートされています。現在、この再配布には一意のインデックスが含まれていません (順序付きインデックスのみが再配布されます)。

新しいデータノードが追加される前にすでに存在していた [NDBCLUSTER](#) テーブルの再配布は、自動的に実行されませんが、[mysql](#) または別の MySQL クライアントアプリケーションで単純な SQL ステートメントを使用して実現できます。ただし、新しいノードグループが追加されたあとに作成されたテーブルに追加されたすべてのデータおよびインデックスは、すべてのクラスタデータノード (新しいノードグループの一部として追加されたものを含む) 間で自動的に配布されます。

部分的な起動. 新しいデータノードがすべて起動されていなくても、新しいノードグループを追加できます。また、新しいノードグループを機能低下状態のクラスタ (つまり、部分的にしか起動されていないクラスタや、1 つ以上のデータノードが実行されていないクラスタ) に追加することもできます。後者の場合、新しいノードグループを追加する前に、クラスタで十分な数のノードが実行可能になっている必要があります。

進行中の操作への影響. NDB Cluster データを使用する通常の DML 操作は、新しいノードグループの作成や追加、またはテーブルの再編成によって妨げられることはありません。ただし、テーブルの再編成と同時に DDL を実行することはできません。つまり、[ALTER TABLE ... REORGANIZE PARTITION](#) ステートメントの実行中は、ほかの DDL ステートメントを発行できません。さらに、[ALTER TABLE ... REORGANIZE PARTITION](#) の実行 (またはその他の DDL ステートメントの実行) 中は、クラスタデータノードを再起動できません。

エラー処理. ノードグループの作成中およびテーブルの再編成中のデータノードの障害は、次のテーブルに示すように処理されます:

表 23.64 ノードグループの作成およびテーブルの再編成中のデータノード障害処理

失敗期間	「旧」データノードで障害が発生しました	「新規」データノードで障害が発生しました	システム障害
ノードグループの作成	<ul style="list-style-type: none"> マスター以外のノードでエラーが発生した場合: ノードグループの作成は常にロールフォワードされます。 マスターでエラーが発生した場合: <ul style="list-style-type: none"> 内部コミットポイントに達した場合: ノードグループの作成はロールフォワードされます。 内部コミットポイントにまだ達していない場 	<ul style="list-style-type: none"> マスター以外のノードでエラーが発生した場合: ノードグループの作成は常にロールフォワードされます。 マスターでエラーが発生した場合: <ul style="list-style-type: none"> 内部コミットポイントに達した場合: ノードグループの作成はロールフォワードされます。 内部コミットポイントにまだ達していない場 	<ul style="list-style-type: none"> CREATE NODEGROUP の実行が内部コミットポイントに達した場合: 再起動時に、クラスタは新しいノードグループを含みます。それ以外の場合は、含みません。 CREATE NODEGROUP の実行が内部コミットポイントにまだ達していない場合: 再起動時に、クラスタは新しいノードグループを含みません。

失敗期間	「旧」データノードで障害が発生しました	「新規」データノードで障害が発生しました	システム障害
	合: ノードグループの作成はロールバックされます。	合: ノードグループの作成はロールバックされます。	
テーブルの再編成	<ul style="list-style-type: none"> マスター以外のノードでエラーが発生した場合: テーブルの再編成は常にロールフォワードされます。 マスターでエラーが発生した場合: <ul style="list-style-type: none"> 内部コミットポイントに達した場合: テーブルの再編成はロールフォワードされます。 内部コミットポイントにまだ達していない場合: テーブルの再編成はロールバックされます。 	<ul style="list-style-type: none"> マスター以外のノードでエラーが発生した場合: テーブルの再編成は常にロールフォワードされます。 マスターでエラーが発生した場合: <ul style="list-style-type: none"> 内部コミットポイントに達した場合: テーブルの再編成はロールフォワードされます。 内部コミットポイントにまだ達していない場合: テーブルの再編成はロールバックされます。 	<ul style="list-style-type: none"> ALTER TABLE ... REORGANIZE PARTITION ステートメントの実行が内部コミットポイントに達した場合: クラスタの再起動時に、table に属するデータおよびインデックスが「新しい」データノードを使用して配布されます。 ALTER TABLE ... REORGANIZE PARTITION ステートメントの実行がまだ内部コミットポイントに達していない場合: クラスタの再起動時に、table に属するデータおよびインデックスが「古い」データノードのみを使用して配布されます。

ノードグループの削除。 `ndb_mgm` クライアントは、`DROP NODEGROUP` コマンドをサポートしていますが、ノードグループ内のデータノードにどのデータも含まれていない場合にのみ削除できます。現在、特定のデータノードまたはノードグループを「空にする」方法がないため、このコマンドは次の2つのケースでしか機能しません。

- `ndb_mgm` クライアントで `CREATE NODEGROUP` を発行した後、`mysql` クライアントで `ALTER TABLE ... REORGANIZE PARTITION` ステートメントを発行する前。
- `DROP TABLE` を使用してすべての `NDBCLUSTER` テーブルを削除したあと。

データノードには引き続きテーブルの定義が格納されるため、この目的では `TRUNCATE TABLE` は機能しません。

23.5.7.2 NDB Cluster データノードのオンラインでの追加: 基本手順

このセクションでは、NDB Cluster に新しいデータノードを追加するために必要な基本的な手順を示します。この手順は、データノードのプロセスに `ndbd` バイナリと `ndbmt` バイナリのどちらを使用する場合でも適用されます。詳細な例については、[セクション23.5.7.3「NDB Cluster データノードのオンラインでの追加: 詳細な例」](#)を参照してください。

NDB Cluster がすでに実行されていると仮定すると、データノードをオンラインで追加するには、次の手順が必要です:

- 追加対象のノードに対応する新しい `[ndbd]` セクションを追加して、クラスタ構成 `config.ini` ファイルを編集します。クラスタで複数の管理サーバーが使用されている場合は、管理サーバーで使用されるすべての `config.ini` ファイルに、これらの変更を加える必要があります。

`config.ini` ファイルに追加される新しいデータノードのノード ID が、既存のノードで使用されるノード ID と重複しないように注意する必要があります。動的に割り当てられたノード ID を使用している API ノードがあり、それらの ID が新しいノードに使用するノード ID と一致する場合は、この手順の後半で説明するように、このような API ノードを強制的に「移行」できます。

- すべての NDB Cluster 管理サーバーのローリング再起動を実行します。

重要

新しい構成を強制的に読み取るには、すべての管理サーバーを `--reload` または `--initial` オプションを付けて再起動する必要があります。

3. すべての既存の NDB Cluster データノードのローリング再起動を実行します。既存のデータノードを再起動するときは、`--initial` を使用する必要がありません (通常は望ましくもありません)。

新しいデータノードに割り当てられているいずれかのノード ID と一致する動的に割り当てられた ID を持つ API ノードを使用している場合は、このステップでデータノードプロセスのいずれかを再起動する前に、すべての API ノード (SQL ノードを含む) を再起動する必要があります。これにより、事前に明示的に割り当てられなかったノード ID を持つ API ノードは、このようなノード ID を放棄し、新しい ID を取得します。

4. NDB Cluster に接続されている SQL または API ノードのローリング再起動を実行します。
5. 新しいデータノードを起動します。

新しいデータノードは、任意の順序で起動できます。またそれらは、既存のすべてのデータノードのローリング再起動が完了してから、次のステップに進むまでの間に起動されるかぎり、同時に起動することもできます。

6. NDB Cluster 管理クライアントで 1 つまたは複数の `CREATE NODEGROUP` コマンドを実行して、新しいデータノードが属する新しいノードグループまたはノードグループを作成します。
7. 新しいデータノードを含め、すべてのデータノード間でクラスタデータを再配布します。通常、これを行うには、`NDBCLUSTER` テーブルごとに `mysql` クライアントで `ALTER TABLE ... ALGORITHM=INPLACE, REORGANIZE PARTITION` ステートメントを発行します。

例外: `MAX_ROWS` オプションを使用して作成されたテーブルの場合、このステートメントは機能しません。かわりに、`ALTER TABLE ... ALGORITHM=INPLACE MAX_ROWS=...` を使用してこのようなテーブルを再編成します。また、`MAX_ROWS` を使用してパーティション数をこの方法で設定することは非推奨であり、かわりに `PARTITION_BALANCE` を使用する必要があります。詳細は、[セクション 13.1.20.11 「NDB_TABLE オプションの設定」](#) を参照してください。

注記

これは、新しいノードグループを追加する時点ですでに存在しているテーブルに対してのみ実行する必要があります。新しいノードグループが追加された後に作成されたテーブルのデータは自動的に分散されますが、新しいノードが追加される前に存在していた特定のテーブル `tbl` に追加されたデータは、そのテーブルが再編成されるまで新しいノードを使用して分散されません。

8. `ALTER TABLE ... REORGANIZE PARTITION ALGORITHM=INPLACE` はパーティションを再編成しますが、「old」ノードで解放された領域は再利用しません。これを行うには、`mysql` クライアントで `NDBCLUSTER` テーブルごとに `OPTIMIZE TABLE` ステートメントを発行します。

これは、インメモリー NDB テーブルの可変幅カラムで使用される領域に対して機能します。 `OPTIMIZE TABLE` は、インメモリーテーブルの固定幅カラムではサポートされていません。「ディスクデータ」テーブルでもサポートされていません。

目的のノードをすべて追加してから、複数の `CREATE NODEGROUP` コマンドを連続して発行し、新しいノードグループをクラスタに追加できます。

23.5.7.3 NDB Cluster データノードのオンラインでの追加: 詳細な例

このセクションでは、単一ノードグループ内に 2 つのデータノードを持つ NDB Cluster から開始し、2 つのノードグループ内に 4 つのデータノードを持つクラスタと結論付けて、新しい NDB Cluster データノードをオンラインで追加する方法を示す詳細な例を示します。

構成の開始。 説明のために、最小限の構成とし、次の情報のみを含む `config.ini` ファイルをクラスタで使用すると仮定します。

```
[ndbd default]
```

```
DataMemory = 100M
IndexMemory = 100M
NoOfReplicas = 2
DataDir = /usr/local/mysql/var/mysql-cluster
```

```
[ndbd]
id = 1
HostName = 198.51.100.1
```

```
[ndbd]
id = 2
HostName = 198.51.100.2
```

```
[mgm]
HostName = 198.51.100.10
id = 10
```

```
[api]
id=20
HostName = 198.51.100.20
```

```
[api]
id=21
HostName = 198.51.100.21
```

注記

データノード ID とその他のノード間のシーケンスにギャップを残しています。これにより、あとで新たに追加されるデータノードに、まだ使用されていないノード ID を簡単に割り当てることができます。

また、適切なコマンド行または `my.cnf` オプションを使用して、クラスタをすでに起動しており、管理クライアントで `SHOW` を実行すると、次に示すものと同様の出力が生成されると仮定します。

```
-- NDB Cluster -- Management Client --
ndb_mgm> SHOW
Connected to Management Server at: 198.51.100.10:1186
Cluster Configuration
-----
[ndbd(NDB)] 2 node(s)
id=1 @198.51.100.1 (8.0.23-ndb-8.0.23, Nodegroup: 0, *)
id=2 @198.51.100.2 (8.0.23-ndb-8.0.23, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=10 @198.51.100.10 (8.0.23-ndb-8.0.23)

[mysqld(API)] 2 node(s)
id=20 @198.51.100.20 (8.0.23-ndb-8.0.23)
id=21 @198.51.100.21 (8.0.23-ndb-8.0.23)
```

最後に、次に示すように作成された単一の `NDBCLUSTER` テーブルがクラスタに含まれていると仮定します。

```
USE n;

CREATE TABLE ips (
  id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  country_code CHAR(2) NOT NULL,
  type CHAR(4) NOT NULL,
  ip_address VARCHAR(15) NOT NULL,
  addresses BIGINT UNSIGNED DEFAULT NULL,
  date BIGINT UNSIGNED DEFAULT NULL
) ENGINE NDBCLUSTER;
```

このセクションの後半で示すメモリの使用率および関連情報は、このテーブルに約 50000 行挿入したあとに生成されました。

注記

この例では、データノードプロセスに、シングルスレッド `ndbd` を使用することを示します。この例は、マルチスレッド `ndbmtid` を使用している場合にも適用できます。そのためには、次のステップに示されている箇所で `ndbmtid` を `ndbd` に置き換えます。

ステップ 1: 構成ファイルの更新. テキストエディタでクラスタのグローバル構成ファイルを開き、2つの新しいデータノードに対応する `[ndbd]` セクションを追加します。(これらのデータノードに ID 3 と 4 を付与し、それらがそれぞれ 198.51.100.3 と 198.51.100.4 のアドレスにあるホストマシンで実行されると仮定します。) 新しいセクションを追加したら、`config.ini` ファイルの内容が次に示すように見えます。ここで、ファイルに追加した部分は太字で示しています。

```
[ndbd default]
DataMemory = 100M
IndexMemory = 100M
NoOfReplicas = 2
DataDir = /usr/local/mysql/var/mysql-cluster

[ndbd]
id = 1
HostName = 198.51.100.1

[ndbd]
id = 2
HostName = 198.51.100.2

[ndbd]
id = 3
HostName = 198.51.100.3

[ndbd]
id = 4
HostName = 198.51.100.4

[mgm]
HostName = 198.51.100.10
id = 10

[api]
id=20
HostName = 198.51.100.20

[api]
id=21
HostName = 198.51.100.21
```

必要な変更が完了したら、ファイルを保存します。

ステップ 2: 管理サーバーの再起動. クラスタ管理サーバーを再起動するには、次のように別々のコマンドを発行して、管理サーバーを停止してから再起動する必要があります。

1. 次に示すように管理クライアントの `STOP` コマンドを使用して、管理サーバーを停止します。

```
ndb_mgm> 10 STOP
Node 10 has shut down.
Disconnecting to allow Management Server to shutdown

shell>
```

2. 管理サーバーをシャットダウンすると管理クライアントが終了するため、システムシェルから管理サーバーを起動する必要があります。単純にするために、`config.ini` は管理サーバーバイナリと同じディレクトリにあると仮定していますが、実際には、構成ファイルの正確なパスを指定する必要があります。また、管理サーバーがその構成キャッシュからではなく、ファイルから新しい構成を読み取るように、`--reload` または `--initial` オプションも指定する必要があります。シェルの現在のディレクトリも管理サーバーバイナリが配置されているディレクトリと同じである場合は、次に示すように管理サーバーを呼び出すことができます。

```
shell> ndb_mgmd -f config.ini --reload
2008-12-08 17:29:23 [MgmSrvr] INFO -- NDB Cluster Management Server. 8.0.23-ndb-8.0.23
2008-12-08 17:29:23 [MgmSrvr] INFO -- Reading cluster configuration from 'config.ini'
```

`ndb_mgm` プロセスが再起動されたあとに、管理クライアントで `SHOW` の出力をチェックすると、次に示すように表示されます。

```
-- NDB Cluster -- Management Client --
ndb_mgm> SHOW
Connected to Management Server at: 198.51.100.10:1186
Cluster Configuration
```

```

-----
[ndbd(NDB)] 2 node(s)
id=1 @198.51.100.1 (8.0.23-ndb-8.0.23, Nodegroup: 0, *)
id=2 @198.51.100.2 (8.0.23-ndb-8.0.23, Nodegroup: 0)
id=3 (not connected, accepting connect from 198.51.100.3)
id=4 (not connected, accepting connect from 198.51.100.4)

[ndb_mgmd(MGM)] 1 node(s)
id=10 @198.51.100.10 (8.0.23-ndb-8.0.23)

[mysqld(API)] 2 node(s)
id=20 @198.51.100.20 (8.0.23-ndb-8.0.23)
id=21 @198.51.100.21 (8.0.23-ndb-8.0.23)

```

ステップ 3: 既存のデータノードのローリング再起動の実行。次に示すように **RESTART** コマンドを使用すると、このステップを完全にクラスタ管理クライアント内で実現できます。

```

ndb_mgm> 1 RESTART
Node 1: Node shutdown initiated
Node 1: Node shutdown completed, restarting, no start.
Node 1 is being restarted

ndb_mgm> Node 1: Start initiated (version 8.0.23)
Node 1: Started (version 8.0.23)

ndb_mgm> 2 RESTART
Node 2: Node shutdown initiated
Node 2: Node shutdown completed, restarting, no start.
Node 2 is being restarted

ndb_mgm> Node 2: Start initiated (version 8.0.23)

ndb_mgm> Node 2: Started (version 8.0.23)

```

重要

各 **X RESTART** コマンドを発行したら、管理クライアントによって「**Node X: Started (version ...)**」というレポートが出力されるまで待機してから、先に進みます。

mysql クライアントで **ndbinfo.nodes** テーブルをチェックすると、既存のデータノードがすべて更新済みの構成を使用して再起動されたことを確認できます。

ステップ 4: すべてのクラスタ API ノードのローリング再起動の実行。 **mysqladmin shutdown** に続いて、**mysqld_safe** (または別の起動スクリプト) を使用して、クラスタ内の SQL ノードとして機能している各 MySQL サーバーをシャットダウンし、再起動します。これは、次に示すものと同様になります。ここで、**password** は特定の MySQL サーバーインスタンスの MySQL **root** パスワードです。

```

shell> mysqladmin -uroot -ppassword shutdown
081208 20:19:56 mysqld_safe mysqld from pid file
/usr/local/mysql/var/tonfisk.pid ended
shell> mysqld_safe --ndbcluster --ndb-connectstring=198.51.100.10 &
081208 20:20:06 mysqld_safe Logging to '/usr/local/mysql/var/tonfisk.err'.
081208 20:20:06 mysqld_safe Starting mysqld daemon with databases
from /usr/local/mysql/var

```

当然、正確な入力および出力は、MySQL がシステムにインストールされている方法や場所、および起動する際に選択したオプション (およびこれらのオプションの一部またはすべてを **my.cnf** ファイルに指定しているかどうか) によって異なります。

ステップ 5: 新しいデータノードの初期起動の実行。次に示すように、新しいデータノードの各ホスト上のシステムシェルから **--initial** オプションを使用して、データノードを起動します。

```
shell> ndbd -c 198.51.100.10 --initial
```

注記

既存のデータノードを再起動する場合とは異なり、新しいデータノードは同時に起動できます。あるデータノードの起動が完了するまで待機してから、別のデータノードを起動する必要はありません。

新しいデータノードの両方が起動されるまで待機してから、次のステップに進みます。新しいデータノードが起動されたら、管理クライアントの **SHOW** コマンドの出力で、(次に太字で示されているように) それらがどのノードグループにも属していないことを確認できます。

```
ndb_mgm> SHOW
Connected to Management Server at: 198.51.100.10:1186
Cluster Configuration
-----
[ndbd(NDB)] 2 node(s)
id=1 @198.51.100.1 (8.0.23-ndb-8.0.23, Nodegroup: 0, *)
id=2 @198.51.100.2 (8.0.23-ndb-8.0.23, Nodegroup: 0)
id=3 @198.51.100.3 (8.0.23-ndb-8.0.23, no nodegroup)
id=4 @198.51.100.4 (8.0.23-ndb-8.0.23, no nodegroup)

[ndb_mgmd(MGM)] 1 node(s)
id=10 @198.51.100.10 (8.0.23-ndb-8.0.23)

[mysqld(API)] 2 node(s)
id=20 @198.51.100.20 (8.0.23-ndb-8.0.23)
id=21 @198.51.100.21 (8.0.23-ndb-8.0.23)
```

ステップ 6: 新しいノードグループの作成。これは、クラスタ管理クライアントで **CREATE NODEGROUP** コマンドを発行することで実行できます。次に示すように、このコマンドは、引数として、新しいノードグループに含むデータノードのノード ID をカンマで区切ったリストを取ります。

```
ndb_mgm> CREATE NODEGROUP 3,4
Nodegroup 1 created
```

再度 **SHOW** を発行すると、(同様に太字に示されているように) データノード 3 と 4 が新しいノードグループに参加したことを確認できます。

```
ndb_mgm> SHOW
Connected to Management Server at: 198.51.100.10:1186
Cluster Configuration
-----
[ndbd(NDB)] 2 node(s)
id=1 @198.51.100.1 (8.0.23-ndb-8.0.23, Nodegroup: 0, *)
id=2 @198.51.100.2 (8.0.23-ndb-8.0.23, Nodegroup: 0)
id=3 @198.51.100.3 (8.0.23-ndb-8.0.23, Nodegroup: 1)
id=4 @198.51.100.4 (8.0.23-ndb-8.0.23, Nodegroup: 1)

[ndb_mgmd(MGM)] 1 node(s)
id=10 @198.51.100.10 (8.0.23-ndb-8.0.23)

[mysqld(API)] 2 node(s)
id=20 @198.51.100.20 (8.0.23-ndb-8.0.23)
id=21 @198.51.100.21 (8.0.23-ndb-8.0.23)
```

ステップ 7: クラスタデータの再配布。ノードグループが作成されても、管理クライアントで適切な **REPORT** コマンドを発行することで確認できるように、既存のデータおよびインデックスは新しいノードグループのデータノードに自動的に配布されません:

```
ndb_mgm> ALL REPORT MEMORY

Node 1: Data usage is 5%(177 32K pages of total 3200)
Node 1: Index usage is 0%(108 8K pages of total 12832)
Node 2: Data usage is 5%(177 32K pages of total 3200)
Node 2: Index usage is 0%(108 8K pages of total 12832)
Node 3: Data usage is 0%(0 32K pages of total 3200)
Node 3: Index usage is 0%(0 8K pages of total 12832)
Node 4: Data usage is 0%(0 32K pages of total 3200)
Node 4: Index usage is 0%(0 8K pages of total 12832)
```

-p オプションを付けて **ndb_desc** を使用すると、出力にパーティション化の情報が含まれるため、テーブルではまだ 2 つのパーティションしか使用されていないことを (ここに太字のテキストで示されている、出力の「**Per partition info**」セクションで) 確認できます。

```
shell> ndb_desc -c 198.51.100.10 -d n ips -p
-- ips --
Version: 1
Fragment type: 9
```



```

K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: no
Number of attributes: 6
Number of primary keys: 1
Length of frm data: 340
Row Checksum: 1
Row GCI: 1
SingleUserMode: 0
ForceVarPart: 1
FragmentCount: 2
TableStatus: Retrieved
-- Attributes --
id Bigint PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY AUTO_INCR
country_code Char(2;latin1_swedish_ci) NOT NULL AT=FIXED ST=MEMORY
type Char(4;latin1_swedish_ci) NOT NULL AT=FIXED ST=MEMORY
ip_address Varchar(15;latin1_swedish_ci) NOT NULL AT=SHORT_VAR ST=MEMORY
addresses Bigunsigned NULL AT=FIXED ST=MEMORY
date Bigunsigned NULL AT=FIXED ST=MEMORY

-- Indexes --
PRIMARY KEY(id) - UniqueHashIndex
PRIMARY(id) - OrderedIndex

-- Per partition info --
Partition Row count Commit count Frag fixed memory Frag var sized memory
0 26086 26086 1572864 557056
1 26329 26329 1605632 557056

NDBT_ProgramExit: 0 - OK

```

mysql クライアントで NDB テーブルごとに `ALTER TABLE ... ALGORITHM=INPLACE, REORGANIZE PARTITION` ステートメントを実行することで、すべてのデータノード間でデータを再分散できます。

重要

`ALTER TABLE ... ALGORITHM=INPLACE, REORGANIZE PARTITION` は、`MAX_ROWS` オプションで作成されたテーブルでは機能しません。かわりに、`ALTER TABLE ... ALGORITHM=INPLACE, MAX_ROWS=...` を使用してこのようなテーブルを再編成します。

`MAX_ROWS` を使用してテーブル当たりのパーティション数を設定することは非推奨であり、かわりに `PARTITION_BALANCE` を使用する必要があります。詳細は、[セクション 13.1.20.11 「NDB_TABLE オプションの設定」](#) を参照してください。

`ALTER TABLE ips ALGORITHM=INPLACE, REORGANIZE PARTITION` ステートメントを発行した後、`ndb_desc` を使用して、このテーブルのデータが 4 つのパーティションを使用して格納されるようになったことを確認できます (出力の関連部分は太字で示されています):

```

shell> ndb_desc -c 198.51.100.10 -d n ips -p
-- ips --
Version: 16777217
Fragment type: 9
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: no
Number of attributes: 6
Number of primary keys: 1
Length of frm data: 341
Row Checksum: 1
Row GCI: 1
SingleUserMode: 0
ForceVarPart: 1
FragmentCount: 4
TableStatus: Retrieved
-- Attributes --
id Bigint PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY AUTO_INCR
country_code Char(2;latin1_swedish_ci) NOT NULL AT=FIXED ST=MEMORY
type Char(4;latin1_swedish_ci) NOT NULL AT=FIXED ST=MEMORY
ip_address Varchar(15;latin1_swedish_ci) NOT NULL AT=SHORT_VAR ST=MEMORY
addresses Bigunsigned NULL AT=FIXED ST=MEMORY

```

```

date Bigunsigned NULL AT=FIXED ST=MEMORY

-- Indexes --
PRIMARY KEY(id) - UniqueHashIndex
PRIMARY(id) - OrderedIndex

-- Per partition info --
Partition  Row count  Commit count  Frag fixed memory  Frag varsized memory
0          12981    52296        1572864           557056
1          13236    52515        1605632           557056
2          13105    13105        819200            294912
3          13093    13093        819200            294912

NDBT_ProgramExit: 0 - OK

```

注記

通常、`ALTER TABLE table_name [ALGORITHM=INPLACE,] REORGANIZE PARTITION` は、パーティション識別子のリストおよびパーティション定義のセットとともに使用され、すでに明示的にパーティション化されているテーブルの新しいパーティション化スキームを作成します。ここでは、データを新しい NDB Cluster ノードグループに再配布するために使用することは例外です。この方法で使用する場合、`REORGANIZE PARTITION` に続くほかのキーワードや識別子はありません。

詳細は、[セクション 13.1.9 「ALTER TABLE ステートメント」](#) を参照してください。

また、無駄な領域を再利用するには、テーブルごとに `ALTER TABLE` ステートメントの後に `OPTIMIZE TABLE` を続ける必要があります。`INFORMATION_SCHEMA.TABLES` テーブルに対して次のクエリーを使用すると、すべての `NDBCLUSTER` テーブルのリストを取得できます。

```

SELECT TABLE_SCHEMA, TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE ENGINE = 'NDBCLUSTER';

```

注記

「NDB Cluster」テーブルの `INFORMATION_SCHEMA.TABLES.ENGINE` 値は、テーブル（または、既存のテーブルを別のストレージエンジンから変換するために使用される `ALTER TABLE` ステートメント）の作成に使用された `CREATE TABLE` ステートメントの `ENGINE` オプションで `NDB` と `NDBCLUSTER` のどちらが使用されたかに関係なく、常に `NDBCLUSTER` です。

これらのステートメントを実行したあとは、次に示すように `ALL REPORT MEMORY` の出力で、データおよびインデックスがすべてのデータノード間で再配布されたことを確認できます。

```

ndb_mgm> ALL REPORT MEMORY

Node 1: Data usage is 5%(176 32K pages of total 3200)
Node 1: Index usage is 0%(76 8K pages of total 12832)
Node 2: Data usage is 5%(176 32K pages of total 3200)
Node 2: Index usage is 0%(76 8K pages of total 12832)
Node 3: Data usage is 2%(80 32K pages of total 3200)
Node 3: Index usage is 0%(51 8K pages of total 12832)
Node 4: Data usage is 2%(80 32K pages of total 3200)
Node 4: Index usage is 0%(50 8K pages of total 12832)

```

注記

`NDBCLUSTER` テーブルに対して一度に実行できる DDL 操作は 1 つのみであるため、次の DDL 操作を発行する前に、各 `ALTER TABLE ... REORGANIZE PARTITION` ステートメントが終了するまで待機する必要があります。

新しいデータノードが追加されたあとに作成された `NDBCLUSTER` テーブルに対して `ALTER TABLE ... REORGANIZE PARTITION` ステートメントを発行する必要はありません。このようなテーブルに追加されたデータは、すべてのデータノードに自動的に分散されます。ただし、次より前に存在する `NDBCLUSTER` テーブルでは、`ALTER TABLE ... REORGANIZE PARTITION` を使用してこれらのテーブルが再編成されるまで、既存のデータも新しいデータも新しいノードを使用して分散されません。

ローリング再起動を使用しない代替の手順。 はじめてクラスタを起動するときに、余分なデータノードを構成するが、それらを起動しないことによって、ローリング再起動の必要性を回避できます。 前のように、まず 1 つのノードグループ内の 2 つのデータノード (ノード 1 と 2) から開始し、あとでノード 3 と 4 で構成される 2 つめのノードグループを追加することで、クラスタを 4 つのデータノードまで拡張すると仮定します。

```
[ndbd default]
DataMemory = 100M
IndexMemory = 100M
NoOfReplicas = 2
DataDir = /usr/local/mysql/var/mysql-cluster

[ndbd]
Id = 1
HostName = 198.51.100.1

[ndbd]
Id = 2
HostName = 198.51.100.2

[ndbd]
Id = 3
HostName = 198.51.100.3
Nodegroup = 65536

[ndbd]
Id = 4
HostName = 198.51.100.4
Nodegroup = 65536

[mgm]
HostName = 198.51.100.10
Id = 10

[api]
Id=20
HostName = 198.51.100.20

[api]
Id=21
HostName = 198.51.100.21
```

あとでオンラインになるデータノード (ノード 3 と 4) を `NodeGroup = 65536` で構成できます。この場合、ノード 1 と 2 はそれぞれ次のように起動できます。

```
shell> ndbd -c 198.51.100.10 --initial
```

`NodeGroup = 65536` で構成されたデータノードは、`StartNoNodeGroupTimeout` データノード構成パラメータの設定によって指定された期間待機してから、`--nowait-nodes=3,4` を使用してノード 1 と 2 を起動したかのように、管理サーバーによって扱われます。デフォルトでは、これは 15 秒 (15000 ミリ秒) です。

注記

`StartNoNodegroupTimeout` はクラスタ内のすべてのデータノードで同じである必要があります。そのため、これは個々のデータノードではなく、常に `config.ini` ファイルの `[ndbd default]` セクションに設定するようにしてください。

2 つめのノードグループを追加する準備ができたら、次の追加ステップを実行する必要があるだけです。

1. データノード 3 と 4 を起動し、新しいノードごとにデータノードのプロセスを 1 回ずつ呼び出します。

```
shell> ndbd -c 198.51.100.10 --initial
```

2. 管理クライアントで適切な `CREATE NODEGROUP` コマンドを発行します。

```
ndb_mgm> CREATE NODEGROUP 3,4
```

3. `mysql` クライアントで、既存の `NDBCLUSTER` テーブルごとに `ALTER TABLE ... REORGANIZE PARTITION` および `OPTIMIZE TABLE` ステートメントを発行します。(このセクションの他の場所で説明されているように、既存の「NDB Cluster」テーブルでは、これが完了するまでデータ分散に新しいノードを使用できません。)

23.5.8 NDB Cluster のオンラインバックアップ

次のいくつかのセクションでは、`ndb_mgm` 管理クライアントにあるこの目的のための機能を使用して NDB Cluster バックアップを準備してから作成する方法について説明します。このタイプのバックアップを `mysqldump` を使用して作成されたバックアップと区別するために、「native」NDB Cluster バックアップと呼ばれることもあります。(`mysqldump` を使用したバックアップの作成については、[セクション4.5.4「mysqldump — データベースバックアッププログラム」](#)を参照してください。) NDB Cluster バックアップの復元は、NDB Cluster 配布で提供される `ndb_restore` ユーティリティを使用して実行されます。`ndb_restore` と NDB Cluster バックアップの復元でのその使用については、[セクション23.4.23「ndb_restore — NDB Cluster バックアップの復元」](#)を参照してください。

NDB 8.0.16 以降では、データノードで並列性を実現するために、複数の LDM を使用してバックアップを作成できます。[セクション23.5.8.5「並列データノードを使用した NDB バックアップの作成」](#)を参照してください。

23.5.8.1 NDB Cluster バックアップの概念

バックアップとは、特定の時点でのデータベースのスナップショットです。バックアップは次の3つの主要な部分で構成されます。

- **メタデータ。** すべてのデータベーステーブルの名前と定義
- **テーブルレコード。** バックアップが作成された時点でデータベーステーブルに実際に格納されたデータ
- **トランザクションログ。** データがデータベースに格納された方法と日時を示す順次レコード

これらの各部分は、バックアップに参加しているすべてのノード上に保存されます。バックアップ中に、各ノードはこれらの3つの部分をディスク上の3つのファイルに保存します。

- [BACKUP-backup_id.node_id.cti](#)

制御情報およびメタデータを含む制御ファイルです。各ノードは、このファイルの各ノード独自のバージョンに、(クラスタ内のすべてのテーブルの) 同じテーブル定義を保存します。

- [BACKUP-backup_id-0.node_id.data](#)

フラグメントごとに保存され、テーブルレコードを含むデータファイルです。つまり、バックアップ中にノードによって異なるフラグメントを保存します。各ノードによって保存されたファイルは、レコードが属するテーブルを示すヘッダーで始まります。レコードのリストのあとに、すべてのレコードのチェックサムを含むフッターが続きます。

- [BACKUP-backup_id.node_id.log](#)

コミットされたトランザクションのレコードを含むログファイルです。このログには、バックアップに格納されたテーブル上のトランザクションのみが格納されます。ノードによって異なるデータベースのフラグメントをホストするため、バックアップに関与するノードはさまざまなレコードを保存します。

前述のリストで、`backup_id` はバックアップ識別子を表し、`node_id` はファイルを作成するノードの一意的識別子です。

バックアップファイルの場所は、`BackupDataDir` パラメータによって決まります。

23.5.8.2 NDB Cluster 管理クライアントを使用したバックアップの作成

バックアップを開始する前に、バックアップを実行するためにクラスタが適切に構成されていることを確認します。([セクション23.5.8.3「NDB Cluster バックアップの構成」](#)を参照してください。))

`START BACKUP` コマンドは、バックアップを作成するために使用されます。

```
START BACKUP [backup_id]  
  [encryption_option]  
  [wait_option]  
  [snapshot_option]  
  
encryption_option:  
ENCRYPT PASSWORD=password  
  
password:
```

```
{'password_string' | "password_string"}  
  
wait_option:  
WAIT {STARTED | COMPLETED} | NOWAIT  
  
snapshot_option:  
SNAPSHOTSTART | SNAPSHOTEND
```

連続したバックアップは自動的に順次識別されるため、`backup_id` (1 以上の整数) はオプションです。これを省略すると、次に使用可能な値が使用されます。既存の `backup_id` 値が使用されると、`Backup failed: file already exists` というエラーでバックアップに失敗します。これを使用する場合は、その他のオプションを使用する前に、`START BACKUP` の直後に `backup_id` を指定する必要があります。

NDB 8.0.22 以降では、`ENCRYPT PASSWORD=password` を使用した暗号化バックアップの作成がサポートされています。`password` は、次の要件をすべて満たす必要があります:

- `!, ', ", $, %, \` および `^` 以外の印刷可能な ASCII 文字を使用
- 256 文字以下
- 一重引用符または二重引用符で囲みます

空のパスワード ("または") を使用することは可能ですが、お薦めしません。

暗号化バックアップは、次のいずれかのコマンドを使用して復号化できます:

- `ndb_restore --decrypt --backup-password=password`
- `ndbxfrm --decrypt-password=password input_file output_file`
- `ndb_print_backup_file -P password file_name`

必要な追加オプションなどの詳細は、これらのプログラムの説明を参照してください。

`wait_option` を使用すると、次のリストに示すように、`START BACKUP` コマンドを発行したあとに、管理クライアントに制御が返されるタイミングを判断できます。

- `NOWAIT` を指定すると、すぐに次に示すようなプロンプトが管理クライアントに表示されます。

```
ndb_mgm> START BACKUP NOWAIT  
ndb_mgm>
```

この場合、バックアッププロセスから進行状況の情報が出力されている間でも、管理クライアントを使用できます。

- `WAIT STARTED` を指定すると、次に示すように、管理クライアントはバックアップが開始されるまで待機してから、ユーザーに制御を返します。

```
ndb_mgm> START BACKUP WAIT STARTED  
Waiting for started, this may take several minutes  
Node 2: Backup 3 started from node 1  
ndb_mgm>
```

- `WAIT COMPLETED` を指定すると、管理クライアントはバックアッププロセスが完了するまで待機してから、ユーザーに制御を返します。

`WAIT COMPLETED` はデフォルトです。

`snapshot_option` を使用すると、`START BACKUP` が発行されたとき、またはそれが完了したときのクラスタの状態とバックアップが一致するかどうかを判断できます。`SNAPSHOTSTART` を使用すると、バックアップがバックアップを開始したときのクラスタの状態と一致します。`SNAPSHOTEND` を使用すると、バックアップはバックアップが終了したときのクラスタの状態を反映します。`SNAPSHOTEND` がデフォルトであり、以前の NDB Cluster リリースで検出された動作と一致します。

注記

`START BACKUP` を付けて `SNAPSHOTSTART` を使用するとき、`CompressedBackup` パラメータが有効になっている場合、データファイルおよび制御ファイルのみが圧縮され、ログファイルは圧縮されません。

`wait_option` と `snapshot_option` の両方を使用する場合、それらはいずれの順序でも指定できます。たとえば、ID として 4 を持つ既存のバックアップが存在しないと仮定すると、次のコマンドはすべて有効です。

```
START BACKUP WAIT STARTED SNAPSHOTSTART
START BACKUP SNAPSHOTSTART WAIT STARTED
START BACKUP 4 WAIT COMPLETED SNAPSHOTSTART
START BACKUP SNAPSHOTEND WAIT COMPLETED
START BACKUP 4 NOWAIT SNAPSHOTSTART
```

バックアップを作成する手順は、次のステップで構成されます。

1. 管理クライアント (`ndb_mgm`) がまだ実行されていない場合は、起動します。
2. `START BACKUP` コマンドを実行します。これにより、次に示すように、バックアップの進行状況を示す複数行の出力が生成されます。

```
ndb_mgm> START BACKUP
Waiting for completed, this may take several minutes
Node 2: Backup 1 started from node 1
Node 2: Backup 1 started from node 1 completed
StartGCP: 177 StopGCP: 180
#Records: 7362 #LogRecords: 0
Data: 453648 bytes Log: 0 bytes
ndb_mgm>
```

3. バックアップが開始されると、次のメッセージが管理クライアントに表示されます。

```
Backup backup_id started from node node_id
```

`backup_id` は、特定のバックアップを表す一意の識別子です。ほかに構成されていない場合、この識別子はクラスタログに保存されます。`node_id` は、データノードとバックアップを調整する管理サーバーの識別子です。バックアッププロセスのこの時点で、クラスタはバックアップリクエストを受信して処理しています。バックアップが終了したことを意味するわけではありません。次に、このステートメントの例を示します。

```
Node 2: Backup 1 started from node 1
```

4. 管理クライアントは、次のようなメッセージでバックアップが開始されたことを示します。

```
Backup backup_id started from node node_id completed
```

バックアップが開始されたことを示す通知の場合も同様に、`backup_id` は、この特定のバックアップを表す一意の識別子であり、`node_id` は、データノードとバックアップを調整する管理サーバーのノード ID です。この出力には、次に示すように、関連するグローバルチェックポイント、バックアップされたレコードの数、およびデータのサイズを含む追加情報が記載されます。

```
Node 2: Backup 1 started from node 1 completed
StartGCP: 177 StopGCP: 180
#Records: 7362 #LogRecords: 0
Data: 453648 bytes Log: 0 bytes
```

また、次の例に示すように、`-e` または `--execute` オプションを付けて `ndb_mgm` を呼び出して、システムシェルからバックアップを実行することもできます。

```
shell> ndb_mgm -e "START BACKUP 6 WAIT COMPLETED SNAPSHOTSTART"
```

この方法で `START BACKUP` を使用する際は、バックアップ ID を指定する必要があります。

クラスタのバックアップは、デフォルトで各データノード上の `DataDir` の `BACKUP` サブディレクトリに作成されます。これは、1 つ以上のデータノードごとに個別にオーバーライドすることも、`BackupDataDir` 構成パラメータを使用することで、`config.ini` ファイル内のすべてのクラスタデータノードに対してオーバーライドすることもできます。特定の `backup_id` を持つバックアップ用に作成されたバックアップファイルは、バックアップディレクトリ内の `BACKUP-backup_id` という名前のサブディレクトリに格納されます。

バックアップの取消し。すでに進行中のバックアップを取り消すか中断するには、次のステップを実行します:

1. 管理クライアントを起動します。
2. 次のコマンドを実行します。


```
ndb_mgm> ABORT BACKUP backup_id
```

数値 `backup_id` は、バックアップが開始されたときに管理クライアントの応答 (「Backup `backup_id` started from `node management_node_id`」メッセージ内) に含まれていたバックアップの識別子です。

- 管理クライアントは、`Abort of backup backup_id ordered` で中断リクエストを確認します。

注記

この時点で、管理クライアントはまだクラスタデータノードからこのリクエストへの応答を受信していないため、実際にはバックアップはまだ中止されていません。

- バックアップが中断されると、管理クライアントは次のような方法でこの事実を報告します:

```
Node 1: Backup 3 started from 5 has been aborted.  
Error: 1321 - Backup aborted by user request: Permanent error: User defined error  
Node 3: Backup 3 started from 5 has been aborted.  
Error: 1323 - 1323: Permanent error: Internal error  
Node 2: Backup 3 started from 5 has been aborted.  
Error: 1323 - 1323: Permanent error: Internal error  
Node 4: Backup 3 started from 5 has been aborted.  
Error: 1323 - 1323: Permanent error: Internal error
```

この例では、4つのデータノードを持つクラスタのサンプル出力を示します。ここで、中止されるバックアップのシーケンス番号は `3` で、クラスタ管理クライアントが接続される管理ノードのノード ID は `5` です。バックアップの中止時にその役割を最初に完了したノードは、中止の理由がユーザーからのリクエストのためであったことをレポートします。(残りのノードは、不明な内部エラーが原因でバックアップが中止されたことをレポートします。)

注記

クラスタノードが特定の順序で `ABORT BACKUP` コマンドに応答する保証はありません。

「Backup `backup_id` started from `node management_node_id` has been aborted」というメッセージは、バックアップが終了し、このバックアップに関連するすべてのファイルがクラスタファイルシステムから削除されたことを示しています。

このコマンドを使用すると、システムシェルから進行中のバックアップを中止することもできます。

```
shell> ndb_mgm -e "ABORT BACKUP backup_id"
```

注記

`ABORT BACKUP` の発行時に、ID `backup_id` を持つバックアップが実行されていない場合は、管理クライアントが応答することも、クラスタログに無効な中止コマンドが送信されたことが表示されることもありません。

23.5.8.3 NDB Cluster バックアップの構成

バックアップには、次の5つの構成パラメータが不可欠です。

- `BackupDataBufferSize`

ディスクに書き込む前のデータをバッファーに入れる際に使用されるメモリの量。

- `BackupLogBufferSize`

ディスクに書き込む前のログレコードをバッファーに入れる際に使用されるメモリの量。

- `BackupMemory`

バックアップ用にデータノードに割り当てられた合計メモリー。これは、バックアップデータバッファーとバックアップログバッファー用に割り当てられたメモリーの合計になります。

- `BackupWriteSize`

ディスクに書き込まれたブロックのデフォルトサイズ。これは、バックアップデータバッファとバックアップログバッファの両方に適用されます。

- [BackupMaxWriteSize](#)

ディスクに書き込まれたブロックの最大サイズ。これは、バックアップデータバッファとバックアップログバッファの両方に適用されます。

また、[CompressedBackup](#) では、バックアップファイルの作成時およびバックアップファイルへの書き込み時に [NDB](#) で圧縮が使用されます。

これらのパラメータに関する詳細は、「[バックアップパラメータ](#)」で見つかります。

[BackupDataDir](#) 構成パラメータを使用して、バックアップファイルの場所を設定することもできます。デフォルトは `FileSystemPath /BACKUP/BACKUP-backup_id` です。

NDB 8.0.22 以降では、[RequireEncryptedBackup](#) を有効にすることによってバックアップファイルの暗号化を強制できます。このパラメータを 1 に設定すると、`START BACKUP` コマンドの一部として `ENCRYPT PASSWORD=password` を指定せずにバックアップを作成することはできません。

23.5.8.4 NDB Cluster バックアップのトラブルシューティング

バックアップリクエストを発行した際にエラーコードが返される場合は、不十分なメモリまたはディスク容量が原因である可能性がもっとも高いです。バックアップ用に十分なメモリが割り当てられていることを確認するようにしてください。

重要

[BackupDataBufferSize](#) および [BackupLogBufferSize](#) を設定していて、それらの合計が 4M バイトよりも大きい場合は、[BackupMemory](#) も設定する必要があります。

バックアップ対象のハードドライブパーティション上に十分な容量があることも確認するようにしてください。

[NDB](#) では反復可能読み取りがサポートされていないため、リストアプロセスで問題が発生する可能性があります。バックアッププロセスは「ホット」ですが、[NDB Cluster](#) をバックアップから復元することは 100% の「ホット」プロセスではありません。その理由は、リストアプロセス中にトランザクションを実行すると、リストアされたデータから反復不可能読み取りが発生するためです。つまり、リストアの進行中は、データの状態に一貫がありません。

23.5.8.5 並列データノードを使用した NDB バックアップの作成

NDB 8.0.16 以降では、データノード上で並列に動作する複数のローカルデータマネージャー (LDM) を使用してバックアップを作成できます。これが機能するには、クラスタ内のすべてのデータノードが複数の LDM を使用し、各データノードが同じ数の LDM を使用する必要があります。つまり、すべてのデータノードで `ndbmt` を実行する必要があります (`ndbd` はシングルスレッドであるため、常に LDM が 1 つしかありません)、バックアップを取得する前に複数の LDM を使用するように構成する必要があります。`ndbmt` はデフォルトでシングルスレッドモードで実行されます。マルチスレッドデータノード構成パラメータ [MaxNoOfExecutionThreads](#) または [ThreadConfig](#) のいずれかに適切な設定を選択することで、複数の LDM を使用させることができます。これらのパラメータを変更すると、クラスタの再起動が必要になることに注意してください。これはローリング再起動である場合があります。

LDM の数やその他の要因によっては、[NoOfFragmentLogParts](#) を増やすことも必要になる場合があります。大規模な「ディスクデータ」テーブルを使用している場合は、[DiskPageBufferMemory](#) も増やす必要がある場合があります。シングルスレッドバックアップと同様に、[BackupDataBufferSize](#)、[BackupMemory](#) およびバックアップに関連するその他の構成パラメータの設定も必要または調整する必要があります ([バックアップパラメータ](#) を参照)。

すべてのデータノードが複数の LDM を使用している場合は、NDB 管理クライアントで `START BACKUP` コマンドを使用して並列バックアップを取ることができます。これは、データノードが `ndbd` (またはシングルスレッドモードで `ndbmt` を実行している場合と同様です。追加または特殊な構文は必要なく、必要に応じて任意の組み合わせでバックアップ ID、待機オプション、またはスナップショットオプションを指定できます。

複数の LDM を使用したバックアップでは、サブディレクトリが作成されます。サブディレクトリは LDM ごとに、各データノードのディレクトリ `BACKUP/BACKUP-backup_id/` (`BackupDataDir` の下にあり) の下に作成されます。これらのサブディレクトリの名前は `BACKUP-backup_id-PART-1-OF-N/`、`BACKUP-backup_id-PART-2-OF-N/` な

どで、`BACKUP-backup_id-PART-N-OF-N/`までです。`backup_id` は、このバックアップに使用されるバックアップ ID で、`N` はデータノードごとの LDM の数です。これらの各サブディレクトリには、通常のバックアップファイル `BACKUP-backup_id-0.node_id.Data`、`BACKUP-backup_id.node_id.ctl` および `BACKUP-backup_id.node_id.log` が含まれます。ここで、`node_id` はこのデータノードのノード ID です。

NDB 8.0.16 以降では、`ndb_restore` は、先ほど説明したサブディレクトリの存在を自動的にチェックします。見つかった場合は、バックアップの並列復元を試みます。複数の LDM で作成されたバックアップのリストアの詳細は、[セクション23.4.23.3「パラレルで作成されたバックアップからのリストア」](#)を参照してください。

23.5.9 NDB Cluster での MySQL Server の使用

`mysqld` は従来の MySQL サーバーのプロセスです。NDB Cluster で使用するには、<https://dev.mysql.com/downloads/> から使用可能なプリコンパイル済みバイナリ内にあるため、NDB ストレージエンジンをサポートするように `mysqld` を構築する必要があります。ソースから MySQL を構築する場合は、`-DWITH_NDBCLUSTER=1` オプションを付けて `CMake` を呼び出して、NDB のサポートを含める必要があります。

ソースから NDB Cluster をコンパイルする方法の詳細は、[セクション23.2.1.4「Linux でのソースからの NDB Cluster の構築」](#)、および [セクション23.2.2.2「Windows でのソースからの NDB Cluster のコンパイルとインストール」](#)を参照してください。

(NDB Cluster に関連するこのセクションで説明されているオプションと変数に加えて、`mysqld` のオプションと変数については、[セクション23.3.3.9「NDB Cluster の MySQL Server オプションおよび変数」](#)を参照してください。)

Cluster サポートを使用して `mysqld` バイナリを構築した場合、NDBCLUSTER ストレージエンジンはまだデフォルトで無効になっています。次の 2 つの指定可能なオプションのいずれかを使用すると、このエンジンを有効にできます。

- `mysqld` を起動する際に、コマンド行で起動オプションとして `--ndbcluster` を使用します。
- `my.cnf` ファイルの `[mysqld]` セクションに `ndbcluster` を含む行を挿入します。

NDBCLUSTER ストレージエンジンを有効にしてサーバーが実行されていることを確認する簡単な方法は、MySQL Monitor (`mysql`) で `SHOW ENGINES` ステートメントを発行することです。NDBCLUSTER の行に、Support 値として値 `YES` が表示されます。この行に `NO` が表示される場合や、このような行が出力に表示されない場合は、NDB 対応の MySQL バージョンが動作していません。この行に `DISABLED` が表示される場合は、先ほど説明した 2 つの方法のいずれかを使用して有効にする必要があります。

クラスタ構成データを読み取るには、MySQL サーバーに少なくとも次の 3 つの情報が必要です。

- MySQL サーバー自身のクラスタノード ID
- 管理サーバーのホスト名または IP アドレス
- 管理サーバーに接続できる TCP/IP ポートの数

ノード ID は動的に割り当てられるため、明示的に指定する必要は厳密にはありません。

接続文字列を指定するには、`mysqld` の起動時にコマンド行で、または `my.cnf` 内に、`mysqld` パラメータ `ndb-connectstring` を使用します。接続文字列には、管理サーバーを検出できるホスト名または IP アドレス、および使用される TCP/IP ポートが含まれます。

次の例では、`ndb_mgmd.mysql.com` は管理サーバーが存在するホストであり、管理サーバーはポート 1186 でクラスタメッセージを待機します。

```
shell> mysqld --ndbcluster --ndb-connectstring=ndb_mgmd.mysql.com:1186
```

接続文字列の詳細は、[セクション23.3.3.3「NDB Cluster 接続文字列」](#)を参照してください。

この情報を指定すると、MySQL サーバーはクラスタの完全な参加者として機能できます。(この方法で実行している `mysqld` プロセスは、多くの場合 SQL ノードと呼んでいます。)すべてのクラスタデータノードとそのステータスを完全に認識し、すべてのデータノードへの接続を確立します。この場合、任意のデータノードをトランザクションコーディネータとして使用し、ノードデータを読み取って更新できます。

`mysql` クライアントで `SHOW PROCESSLIST` を使用すると、MySQL サーバーがクラスタに接続されているかどうかを確認できます。MySQL サーバーがクラスタに接続されていて、`PROCESS` 権限を持っている場合は、出力の最初の行が次のように表示されます。

```
mysql> SHOW PROCESSLIST \G
***** 1. row *****
  Id: 1
  User: system user
  Host:
  db:
  Command: Daemon
  Time: 1
  State: Waiting for event from ndbcluster
  Info: NULL
```

重要

NDB Cluster に参加するには、オプション `--ndbcluster` および `--ndb-connectstring` (または `my.cnf` での同等のもの) を both で `mysqld` プロセスを起動する必要があります。 `--ndbcluster` オプションのみを付けて `mysqld` が起動された場合や、それがクラスタに接続できない場合は、ストレージエンジンに関係なく、NDB テーブルを操作することも、新しいテーブルを作成することもできません。後者の制約は、SQL ノードがクラスタに接続されていない間に、NDB テーブルと同じ名前を持つテーブルが作成されることを防ぐ目的の安全対策です。 `mysqld` プロセスが NDB Cluster に参加していないときに別のストレージエンジンを使用してテーブルを作成する場合は、サーバーなしを `--ndbcluster` オプションで再起動する必要があります。

23.5.10 NDB Cluster ディスクデータテーブル

NDB Cluster は、RAM ではなく、NDB テーブルのインデックスなしカラムのディスクへの格納をサポートしています。次のセクションで説明するように、カラムデータおよびロギングメタデータは、テーブルスペースおよびログファイルグループとして概念化されたデータファイルおよび undo ログファイルに保持されます。 [セクション 23.5.10.1 「NDB Cluster ディスクデータオブジェクト」](#) を参照してください。

NDB Cluster ディスクデータのパフォーマンスは、多数の構成パラメータの影響を受ける可能性があります。これらのパラメータとその効果の詳細は、 [ディスクデータの構成パラメータ](#) および [ディスクデータと GCP 停止エラー](#) を参照してください。

NDB 8.0.19 以降では、ディスクデータファイルに個別のディスクを使用する場合は、 `DiskDataUsingSameDisk` データノード構成パラメータも `false` に設定するようにしてください。

[ディスクデータのファイルシステムパラメータ](#) も参照してください。

NDB 8.0.19 以降では、ソリッドステートドライブ、特に NVMe を使用するドライブで「ディスクデータ」テーブルを使用する場合のサポートが向上しています。詳細は、次のドキュメントを参照してください：

- [ディスクデータ待機時間パラメータ](#)
- [セクション 23.5.14.28 「ndbinfo diskstat テーブル」](#)
- [セクション 23.5.14.29 「ndbinfo diskstats_1sec テーブル」](#)
- [セクション 23.5.14.40 「ndbinfo pgman_time_track_stats テーブル」](#)

23.5.10.1 NDB Cluster ディスクデータオブジェクト

NDB Cluster ディスクデータストレージは、次のオブジェクトを使用して実装されます：

- テーブルスペース: ほかのディスクデータオブジェクトのコンテナとして機能します。テーブルスペースには、1 つ以上のデータファイルと 1 つ以上の undo ログファイルグループが含まれます。
- データファイル: カラムデータを格納します。データファイルは、テーブルスペースに直接割り当てられます。
- undo ログファイル: トランザクションのロールバックに必要な undo 情報が含まれます。undo ログファイルグループに割り当てられます。
- log file group: 1 つ以上の undo ログファイルが含まれています。テーブルスペースに割り当てられます。

undo ログファイルとデータファイルは、各データノードのファイルシステム内の実際のファイルです。デフォルトでは、NDB Cluster `config.ini` ファイルで指定された `DataDir` 内の `ndb_node_id_fs` に配置され、`node_id` はデータノード

のノード ID です。Undo ログファイルまたはデータファイルの作成時に、ファイル名の一部として絶対パスまたは相対パスを指定すると、これらを別の場所に配置できます。これらのファイルを作成するステートメントは、このセクションの後半で示します。

undo ログファイルは「ディスクデータ」テーブルでのみ使用され、メモリーにのみ格納されている NDB テーブルでは不要または使用されません。

「NDB Cluster」テーブルスペースおよびログファイルグループは、ファイルとして実装されません。

すべてのディスクデータオブジェクトがファイルとして実装されるとはかぎりませんが、すべてが同じ名前空間を共有します。つまり、各ディスクデータオブジェクトは (単に、特定の型の各ディスクデータオブジェクトというだけでなく)、一意の名前が付けられている必要があります。たとえば、テーブルスペースとログファイルグループの両方に `dd1` という名前を付けることはできません。

すべてのノード (管理ノードおよび SQL ノードを含む) で「NDB Cluster」をすでに設定している場合、ディスクに「NDB Cluster」テーブルを作成する基本的なステップは次のとおりです:

1. ログファイルグループを作成し、それに 1 つ以上の Undo ログファイルを割り当てます (Undo ログファイルは、`undofile` と呼ばれることもあります)。
2. テーブルスペースを作成し、そのテーブルスペースにログファイルグループおよび 1 つ以上のデータファイルを割り当てます。
3. データストレージ用に、このテーブルスペースを使用するディスクデータテーブルを作成します。

これらの各タスクは、次の例に示すように、`mysql` クライアントまたはその他の MySQL クライアントアプリケーションで SQL ステートメントを使用することで実現できます。

1. `CREATE LOGFILE GROUP` を使用して、`lg_1` という名前のログファイルグループを作成します。このログファイルグループは、`undo_1.log` および `undo_2.log` という名前の 2 つの Undo ログファイルで構成されます。初期サイズは、それぞれ 16M バイトと 12M バイトです。(Undo ログファイルのデフォルト初期サイズは 128M バイトです。) 必要に応じて、ログファイルグループの undo バッファのサイズを指定したり、デフォルト値の 8 MB を想定することもできます。この例では、UNDO バッファサイズを 2 MB に設定します。ログファイルグループは、Undo ログファイルとともに作成する必要があります。そのため、次の `CREATE LOGFILE GROUP` ステートメントで `undo_1.log` を `lg_1` に追加しています。

```
CREATE LOGFILE GROUP lg_1
  ADD UNDOFILE 'undo_1.log'
  INITIAL_SIZE 16M
  UNDO_BUFFER_SIZE 2M
  ENGINE NDBCLUSTER;
```

ログファイルグループに `undo_2.log` を追加するには、次の `ALTER LOGFILE GROUP` ステートメントを使用します。

```
ALTER LOGFILE GROUP lg_1
  ADD UNDOFILE 'undo_2.log'
  INITIAL_SIZE 12M
  ENGINE NDBCLUSTER;
```

いくつかの注意事項:

- ここで使用される `.log` ファイル拡張子は必須ではありません。ログファイルを簡単に認識できるようにするだけで済みます。
- すべての `CREATE LOGFILE GROUP` および `ALTER LOGFILE GROUP` ステートメントに `ENGINE` オプションが含まれている必要があります。このオプションに指定できる値は、`NDBCLUSTER` および `NDB` のみです。

重要

常に同じ NDB Cluster 内に最大 1 つのログファイルグループが存在できます。

- `ADD UNDOFILE 'filename'` を使用して、ログファイルグループに Undo ログファイルを追加すると、クラスタ内の各データノードの `DataDir` 内にある `ndb_node_id_fs` ディレクトリに、`filename` という名前のファイルが作成されます。ここで、`node_id` はデータノードのノード ID です。各 Undo ログファイルのサイズは、SQL ステートメントで指定されます。たとえば、NDB Cluster に 4 つのデータノードがある場合、示されている

`ALTER LOGFILE GROUP` ステートメントは 4 つの undo ログファイルを作成し、それぞれ 4 つのデータノードのデータディレクトリに 1 つずつ作成します。これらの各ファイルの名前は `undo_2.log` で、各ファイルのサイズは 12 MB です。

- `UNDO_BUFFER_SIZE` は、使用可能なシステムメモリーの量によって制限されます。
 - これらのステートメントの詳細は、[セクション 13.1.16 「CREATE LOGFILE GROUP ステートメント」](#) および [セクション 13.1.6 「ALTER LOGFILE GROUP ステートメント」](#) を参照してください。
2. では、「ディスクデータ」テーブルがデータを格納するために使用するファイルの抽象コンテナであるテーブルスペースを作成できるようになりました。テーブルスペースは特定のログファイルグループに関連付けられます。新しいテーブルスペースを作成する場合は、undo ログに使用するログファイルグループを指定する必要があります。少なくとも 1 つのデータファイルを指定する必要があります。テーブルスペースの作成後に、テーブルスペースにデータファイルを追加できます。テーブルスペースからデータファイルを削除することもできます (このセクションの後半の例を参照)。

ログファイルグループとして `lg_1` を使用する `ts_1` という名前のテーブルスペースを作成すると仮定します。テーブルスペースには、初期サイズがそれぞれ 32 MB および 48 MB の `data_1.dat` および `data_2.dat` という名前の 2 つのデータファイルが含まれるようにします。(`INITIAL_SIZE` のデフォルトの値は 128M バイトです。) これは、次に示すように、2 つの SQL ステートメントを使用することで実行できます。

```
CREATE TABLESPACE ts_1
  ADD DATAFILE 'data_1.dat'
  USE LOGFILE GROUP lg_1
  INITIAL_SIZE 32M
  ENGINE NDBCLUSTER;

ALTER TABLESPACE ts_1
  ADD DATAFILE 'data_2.dat'
  INITIAL_SIZE 48M;
```

`CREATE TABLESPACE` ステートメントは、データファイル `data_1.dat` を含むテーブルスペース `ts_1` を作成し、`ts_1` をログファイルグループ `lg_1` に関連付けます。 `ALTER TABLESPACE` は、2 つめのデータファイル (`data_2.dat`) を追加します。

いくつかの注意事項:

- undo ログファイルにこの例で使用されている `.log` ファイル拡張子と同様に、`.dat` ファイル拡張子には特別な意味はなく、認識しやすいようにのみ使用されます。
- `ADD DATAFILE 'filename'` を使用して、テーブルスペースにデータファイルを追加すると、クラスター内の各データノードの `DataDir` 内にある `ndb_node_id_fs` ディレクトリに、`filename` という名前のファイルが作成されます。ここで、`node_id` はデータノードのノード ID です。各データファイルのサイズは、SQL ステートメントで指定します。たとえば、NDB Cluster に 4 つのデータノードがある場合、示されている `ALTER TABLESPACE` ステートメントは、4 つのデータファイルを作成し、それぞれ 4 つの各データノードのデータディレクトリに 1 つずつ作成します。これらの各ファイルの名前は `data_2.dat` で、各ファイルのサイズは 48 MB です。
- NDB は、データノードの再起動時に使用するために各テーブルスペースの 4% を予約します。この領域はデータの格納には使用できません。
- `CREATE TABLESPACE` ステートメントには `ENGINE` 句を含める必要があります。テーブルスペースに作成できるのは、テーブルスペースと同じストレージエンジンを使用するテーブルのみです。 `ALTER TABLESPACE` では、`ENGINE` 句は受け入れられますが、非推奨であり、将来のリリースで削除される可能性があります。NDB テーブルスペースの場合、このオプションに使用できる値は `NDBCLUSTER` および `NDB` のみです。
- NDB 8.0.20 以降では、エクステントの割り当ては、特定のテーブルスペースで使用されるすべてのデータファイル間でラウンドロビン方式で実行されます。
- `CREATE TABLESPACE` および `ALTER TABLESPACE` ステートメントについての詳細は、[セクション 13.1.21 「CREATE TABLESPACE ステートメント」](#) および [セクション 13.1.10 「ALTER TABLESPACE ステートメント」](#) を参照してください。

3. テーブルスペース `ts_1` のファイルを使用して、インデックス付けされていないカラムがディスクに格納されているテーブルを作成できるようになりました:

```
CREATE TABLE dt_1 (  
  member_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,  
  last_name VARCHAR(50) NOT NULL,  
  first_name VARCHAR(50) NOT NULL,  
  dob DATE NOT NULL,  
  joined DATE NOT NULL,  
  INDEX(last_name, first_name)  
)  
TABLESPACE ts_1 STORAGE DISK  
ENGINE NDBCLUSTER;
```

`TABLESPACE ts_1 STORAGE DISK` は、ディスク上のデータ記憶域にテーブルスペース `ts_1` を使用するように NDB ストレージエンジンに指示します。

次のようにテーブル `ts_1` が作成されたら、その他の MySQL テーブルの場合と同様に、`INSERT`、`SELECT`、`UPDATE`、および `DELETE` ステートメントを実行できます。

`CREATE TABLE` ステートメントまたは `ALTER TABLE` ステートメントのカラム定義の一部として `STORAGE` 句を使用して、個々のカラムをディスクに格納するかメモリーに格納するかを指定することもできます。`STORAGE DISK` を指定するとカラムはディスク上に格納され、`STORAGE MEMORY` を指定するとインメモリーストレージが使用されます。詳細は、[セクション13.1.20「CREATE TABLE ステートメント」](#)を参照してください。

次に示すように、`INFORMATION_SCHEMA` データベースの `FILES` テーブルをクエリーすることで、作成した NDB ディスクデータファイルおよび undo ログファイルに関する情報を取得できます:

```
mysql> SELECT  
  FILE_NAME AS File, FILE_TYPE AS Type,  
  TABLESPACE_NAME AS Tablespace, TABLE_NAME AS Name,  
  LOGFILE_GROUP_NAME AS 'File group',  
  FREE_EXTENTS AS Free, TOTAL_EXTENTS AS Total  
FROM INFORMATION_SCHEMA.FILES  
WHERE ENGINE='ndbcluster';  
+-----+-----+-----+-----+-----+-----+  
| File      | Type      | Tablespace | Name | File group | Free | Total |  
+-----+-----+-----+-----+-----+-----+  
| ./undo_1.log | UNDO LOG | lg_1      | NULL | lg_1      | 0 | 4194304 |  
| ./undo_2.log | UNDO LOG | lg_1      | NULL | lg_1      | 0 | 3145728 |  
| ./data_1.dat | DATAFILE | ts_1      | NULL | lg_1      | 32 | 32 |  
| ./data_2.dat | DATAFILE | ts_1      | NULL | lg_1      | 48 | 48 |  
+-----+-----+-----+-----+-----+-----+  
4 rows in set (0.00 sec)
```

詳細および例については、[セクション26.15「INFORMATION_SCHEMA FILES テーブル」](#)を参照してください。

ディスクに暗黙的に格納されたカラムのインデックス付け。前述の例で定義されているテーブル `dt_1` の場合、`dob` および `joined` カラムのみがディスクに格納されます。この理由は、`id`、`last_name`、および `first_name` カラムにインデックスがあるため、これらのカラムに属するデータが RAM 内に格納されるためです。インデックス付けされていないカラムのみをディスクに保持できます。インデックスおよびインデックス付けされたカラムデータは引き続きメモリーに格納されます。インデックスの使用と RAM の保存のこのようなトレードオフは、ディスクデータテーブルを設計する際に留意する必要があります。

最初に記憶域タイプを `MEMORY` に変更しないと、明示的に宣言された `STORAGE DISK` のカラムにインデックスを追加できません。追加しようとする、エラーで失敗します。暗黙的がディスク記憶域を使用するカラムにはインデックスを付けることができます。その場合、カラム記憶域タイプは自動的に `MEMORY` に変更されます。「暗黙的」では、記憶域タイプが宣言されていないが、親テーブルから継承されるカラムを意味します。次の `CREATE TABLE` ステートメント (以前に定義したテーブルスペース `ts_1` を使用) では、カラム `c2` および `c3` は暗黙的にディスク記憶域を使用します:

```
mysql> CREATE TABLE ti (  
->  c1 INT PRIMARY KEY,  
->  c2 INT,  
->  c3 INT,  
->  c4 INT
```

```
-> )
-> STORAGE DISK
-> TABLESPACE ts_1
-> ENGINE NDBCLUSTER;
Query OK, 0 rows affected (1.31 sec)
```

c2、c3 および c4 自体は **STORAGE DISK** で宣言されていないため、インデックス付けできます。ここでは、**CREATE INDEX** および **ALTER TABLE** をそれぞれ使用して、c2 および c3 にインデックスを追加します:

```
mysql> CREATE INDEX i1 ON ti(c2);
Query OK, 0 rows affected (2.72 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> ALTER TABLE ti ADD INDEX i2(c3);
Query OK, 0 rows affected (0.92 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

SHOW CREATE TABLE により、インデックスが追加されたことが確認されます。

```
mysql> SHOW CREATE TABLE ti\G
***** 1. row *****
      Table: ti
Create Table: CREATE TABLE `ti` (
  `c1` int(11) NOT NULL,
  `c2` int(11) DEFAULT NULL,
  `c3` int(11) DEFAULT NULL,
  `c4` int(11) DEFAULT NULL,
  PRIMARY KEY (`c1`),
  KEY `i1` (`c2`),
  KEY `i2` (`c3`)
) /*!50100 TABLESPACE `ts_1` STORAGE DISK */ ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

インデックス付けされたカラム (強調されたテキスト) がディスク上の記憶域ではなくインメモリーを使用するようになったことを、**ndb_desc** を使用して確認できます:

```
shell> ./ndb_desc -d test t1
-- t1 --
Version: 33554433
Fragment type: HashMapPartition
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: no
Number of attributes: 4
Number of primary keys: 1
Length of frm data: 317
Max Rows: 0
Row Checksum: 1
Row GCI: 1
SingleUserMode: 0
ForceVarPart: 1
PartitionCount: 4
FragmentCount: 4
PartitionBalance: FOR_RP_BY_LDM
ExtraRowGciBits: 0
ExtraRowAuthorBits: 0
TableStatus: Retrieved
Table options:
HashMap: DEFAULT-HASHMAP-3840-4
-- Attributes --
c1 Int PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY
c2 Int NULL AT=FIXED ST=MEMORY
c3 Int NULL AT=FIXED ST=MEMORY
c4 Int NULL AT=FIXED ST=DISK
-- Indexes --
PRIMARY KEY(c1) - UniqueHashIndex
i2(c3) - OrderedIndex
PRIMARY(c1) - OrderedIndex
i1(c2) - OrderedIndex

NDBT_ProgramExit: 0 - OK
```

パフォーマンスに関する注意。データノードファイルシステムから分離された物理ディスク上にディスクデータファイルが保持されている場合、ディスクデータストレージを使用しているクラスタのパフォーマンスが大幅に改善されます。顕著な利点を引き出すには、クラスタ内のデータノードごとに、これを実行する必要があります。

ADD UNDOFILE および **ADD DATAFILE** では、絶対および相対ファイルシステムパスを使用できます。相対パスは、データノードデータディレクトリに関して計算されます。

これらを使用しているログファイルグループ、テーブルスペース、およびディスクデータテーブルは、特定の順序で作成する必要があります。これは、これらのオブジェクト (次の制約の対象) を削除する場合にも当てはまります:

- ログファイルグループは、テーブルスペースで使用されているかぎり削除できません。
- テーブルスペースは、データファイルを格納しているかぎり、削除できません。
- テーブルスペースを使用しているテーブルが残っているかぎり、テーブルスペースからどのデータファイルも削除できません。
- ファイルが作成されたテーブルスペース以外の別のテーブルスペースと関連して作成されたファイルは削除できません。

たとえば、このセクションでこれまでに作成したすべてのオブジェクトを削除するには、次のステートメントを使用できます:

```
mysql> DROP TABLE dt_1;

mysql> ALTER TABLESPACE ts_1
-> DROP DATAFILE 'data_2.dat'
-> ENGINE NDBCLUSTER;

mysql> ALTER TABLESPACE ts_1
-> DROP DATAFILE 'data_1.dat'
-> ENGINE NDBCLUSTER;

mysql> DROP TABLESPACE ts_1
-> ENGINE NDBCLUSTER;

mysql> DROP LOGFILE GROUP lg_1
-> ENGINE NDBCLUSTER;
```

これらのステートメントは、ここで示した順序で実行する必要があります。ただし、2つの **ALTER TABLESPACE ... DROP DATAFILE** ステートメントは、どちらの順序でも実行できます。

23.5.10.2 NDB Cluster ディスクデータストレージの要件

ディスクデータストレージの要件には、次の項目が適用されます:

- ディスクデータテーブルの可変長カラムは、固定の容量を占有します。これは、行ごとに、そのカラムで可能性のある最大値を格納するために必要な容量に等しくなります。

このような値の計算に関する一般的な情報については、[セクション11.7「データ型のストレージ要件」](#)を参照してください。

INFORMATION_SCHEMA.FILES テーブルをクエリーして、データファイルおよび Undo ログファイルで使用可能な推定容量を取得できます。詳細および例については、[セクション26.15「INFORMATION_SCHEMA FILES テーブル」](#)を参照してください。

注記

OPTIMIZE TABLE ステートメントを実行しても、ディスクデータテーブルは影響を受けません。

- ディスクデータテーブルでは、**TEXT** または **BLOB** カラムの最初の 256 バイトがメモリー内に格納され、その残りがディスク上に格納されます。
- ディスクデータテーブル内の各行では、ディスク上に格納されたデータを指すためにメモリー内の 8 バイトが使用されます。つまり、場合によっては、インメモリーカラムをディスクベースの形式に変換すれば、実際にメモリー

の使用率を改善できます。たとえば、`CHAR(4)` カラムをメモリーベースの形式からディスクベースの形式に変換すると、1 行当たりで使用される `DataMemory` の量が 4 バイトから 8 バイトに増加します。

重要

`--initial` オプションを付けてクラスタを起動しても、ディスクデータファイルは削除されません。クラスタの初期再起動を実行する前に、これらを手動で削除する必要があります。

`DiskPageBufferMemory` のサイズが十分であることを確認して、ディスクシークの回数を最小限に抑えると、ディスクデータテーブルのパフォーマンスを改善できます。`diskpagebuffer` テーブルをクエリーすると、このパラメータの値を大きくする必要があるかどうかを判断する際に役立ちます。

23.5.11 NDB Cluster での ALTER TABLE を使用したオンライン操作

MySQL NDB Cluster 8.0 は、MySQL Server (`ALGORITHM=DEFAULT|INPLACE|COPY`) で採用され、ほかで説明されている標準の `ALTER TABLE` 構文を使用して、オンラインテーブルスキーマの変更をサポートします。

注記

NDB Cluster の一部の旧リリースでは、オンライン `ALTER TABLE` 操作に NDB 固有の構文が使用されていました。その構文は削除されました。

NDB テーブルの可変幅カラム上のインデックスを追加および削除する操作はオンラインで実行されます。オンライン操作はコピーなしです。つまり、インデックスを再作成する必要はありません。NDB Cluster 内のほかの API ノードによるアクセスから変更されるテーブルはロックされません (ただし、このセクションの後半の [NDB オンライン操作の制限](#) を参照してください)。このような操作では、複数の API ノードを含む NDB クラスタで行われた NDB テーブルの変更にシングルユーザーモードは必要ありません。トランザクションは、オンライン DDL 操作中も中断なく続行できます。

`ALGORITHM=INPLACE` を使用して、NDB テーブルに対してオンラインの `ADD COLUMN`、`ADD INDEX (CREATE INDEX ステートメントを含む)` および `DROP INDEX` 操作を実行できます。また、NDB テーブルのオンラインでの名前変更もサポートされます。

以前は、NDB テーブルのカラムの名前をオンラインで変更できませんでした。この制限は NDB 8.0.18 では削除されています。

現在、ディスクベースのカラムを NDB テーブルにオンラインで追加することはできません。つまり、テーブルレベルの `STORAGE DISK` オプションを使用する NDB テーブルにインメモリーカラムを追加する場合は、新しいカラムをメモリーベースのストレージの使用として明示的に宣言する必要があります。たとえば、すでにテーブルスペース `ts1` を作成していると仮定して、テーブル `t1` を次のように作成するとします。

```
mysql> CREATE TABLE t1 (
>   c1 INT NOT NULL PRIMARY KEY,
>   c2 VARCHAR(30)
> )
> TABLESPACE ts1 STORAGE DISK
> ENGINE NDB;
```

```
Query OK, 0 rows affected (1.73 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

次に示すように、新しいインメモリーカラムをこのテーブルにオンラインで追加できます。

```
mysql> ALTER TABLE t1
>   ADD COLUMN c3 INT COLUMN_FORMAT DYNAMIC STORAGE MEMORY,
>   ALGORITHM=INPLACE;
```

```
Query OK, 0 rows affected (1.25 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

`STORAGE MEMORY` オプションが省略されている場合、このステートメントは失敗します。

```
mysql> ALTER TABLE t1
>   ADD COLUMN c4 INT COLUMN_FORMAT DYNAMIC,
>   ALGORITHM=INPLACE;
ERROR 1846 (0A000): ALGORITHM=INPLACE is not supported. Reason:
Adding column(s) or add/reorganize partition not supported online. Try
ALGORITHM=COPY.
```

COLUMN_FORMAT DYNAMIC オプションを省略した場合は、動的なカラムフォーマットが自動的に使用されますが、次に示すような警告が発行されます。

```
mysql> ALTER ONLINE TABLE t1 ADD COLUMN c4 INT STORAGE MEMORY;
Query OK, 0 rows affected, 1 warning (1.17 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Warning
Code: 1478
Message: DYNAMIC column c4 with STORAGE DISK is not supported, column will
become FIXED

mysql> SHOW CREATE TABLE t1\G
***** 1. row *****
Table: t1
Create Table: CREATE TABLE `t1` (
  `c1` int(11) NOT NULL,
  `c2` varchar(30) DEFAULT NULL,
  `c3` int(11) /*!50606 STORAGE MEMORY */ /*!50606 COLUMN_FORMAT DYNAMIC */ DEFAULT NULL,
  `c4` int(11) /*!50606 STORAGE MEMORY */ DEFAULT NULL,
  PRIMARY KEY (`c1`)
) /*!50606 TABLESPACE ts_1 STORAGE DISK */ ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.03 sec)
```

注記

STORAGE および **COLUMN_FORMAT** キーワードは NDB Cluster でのみサポートされます。ほかのバージョンの MySQL では、**CREATE TABLE** または **ALTER TABLE** ステートメントでこれらのキーワードのいずれかを使用しようとするとエラーになります。

NDB テーブルでは、**partition_names INTO (partition_definitions)** オプションを指定せずに **ALTER TABLE ... REORGANIZE PARTITION, ALGORITHM=INPLACE** ステートメントを使用することもできます。これは、クラスタにオンラインで追加された新しいデータノード間で NDB Cluster データを再配布するために使用できます。これによりデフラグは実行されず、**OPTIMIZE TABLE** または null の **ALTER TABLE** ステートメントが必要になります。詳細は、[セクション23.5.7「NDB Cluster データノードのオンラインでの追加」](#)を参照してください。

NDB オンライン操作の制限

オンライン **DROP COLUMN** 操作はサポートされていません。

カラムを追加するか、あるいはインデックスを追加または削除するオンライン **ALTER TABLE**、**CREATE INDEX**、または **DROP INDEX** ステートメントは、次の制限に従います。

- 特定のオンライン **ALTER TABLE** では、**ADD COLUMN**、**ADD INDEX**、**DROP INDEX** のいずれか 1 つのみを使用できます。1 つのステートメントで、1 つ以上のカラムをオンラインで追加できます。1 つのステートメントで、1 つのインデックスのみをオンラインで作成または削除できます。
- 変更されるテーブルは、オンライン **ALTER TABLE ADD COLUMN**、**ADD INDEX**、または **DROP INDEX** 操作 (あるいは **CREATE INDEX** または **DROP INDEX** ステートメント) が実行されている API ノード以外の API ノードに対してロックされません。ただし、オンライン操作が実行されている間、このテーブルは同じ API ノードから発信されているほかのすべての操作に対してロックされます。
- 変更されるテーブルには明示的な主キーが存在する必要があります。NDB ストレージエンジンによって作成された非表示の主キーは、この目的には不十分です。
- テーブルで使用されるストレージエンジンをオンラインで変更することはできません。
- テーブルで使用されるテーブルスペースはオンラインで変更できません。NDB 8.0.21 以降では、**ALTER TABLE ndb_table ... ALGORITHM=INPLACE, TABLESPACE=new_tablespace** などのステートメントは特に許可されません。(Bug #99269、Bug #31180526)
- 「NDB Cluster ディスクデータ」テーブルで使用する場合、カラムの記憶域タイプ (**DISK** または **MEMORY**) をオンラインで変更することはできません。つまり、操作がオンラインで実行されるような方法でインデックスを追

加または削除し、カラムの記憶域タイプを変更する場合は、インデックスを追加または削除するステートメントで `ALGORITHM=COPY` を使用する必要があります。

オンラインで追加されるカラムは `BLOB` または `TEXT` 型を使用できず、次の条件を満たす必要があります。

- このカラムは動的である必要があります。つまり、`COLUMN_FORMAT DYNAMIC` を使用して作成できる必要があります。`COLUMN_FORMAT DYNAMIC` オプションを省略した場合は、動的なカラムフォーマットが自動的に使用されます。
- このカラムは `NULL` 値を許可する必要があり、`NULL` 以外の明示的なデフォルト値があってはなりません。オンラインで追加されるカラムは、次に示すように、`DEFAULT NULL` として自動的に作成されます。

```
mysql> CREATE TABLE t2 (
  > c1 INT NOT NULL AUTO_INCREMENT PRIMARY KEY
  > ) ENGINE=NDB;
Query OK, 0 rows affected (1.44 sec)
```

```
mysql> ALTER TABLE t2
  > ADD COLUMN c2 INT,
  > ADD COLUMN c3 INT,
  > ALGORITHM=INPLACE;
Query OK, 0 rows affected, 2 warnings (0.93 sec)
```

```
mysql> SHOW CREATE TABLE t1\G
***** 1. row *****
Table: t1
Create Table: CREATE TABLE `t2` (
  `c1` int(11) NOT NULL AUTO_INCREMENT,
  `c2` int(11) DEFAULT NULL,
  `c3` int(11) DEFAULT NULL,
  PRIMARY KEY (`c1`)
) ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

- このカラムは、既存のすべてのカラムのあとに追加する必要があります。既存のいずれかのカラムの前に、または `FIRST` キーワードを使用してオンラインでカラムを追加しようとすると、このステートメントはエラーで失敗します。
- 既存のテーブルカラムをオンラインで並べ替えることはできません。

NDB テーブルでのオンライン `ALTER TABLE` 操作の場合、固定形式のカラムは、オンラインで追加されたとき、またはオンラインでインデックスが作成または削除されたときに動的に変換されます (わかりやすくするために、`CREATE TABLE` および `ALTER TABLE` ステートメントを繰り返します):

```
mysql> CREATE TABLE t2 (
  > c1 INT NOT NULL AUTO_INCREMENT PRIMARY KEY
  > ) ENGINE=NDB;
Query OK, 0 rows affected (1.44 sec)

mysql> ALTER TABLE t2
  > ADD COLUMN c2 INT,
  > ADD COLUMN c3 INT,
  > ALGORITHM=INPLACE;
Query OK, 0 rows affected, 2 warnings (0.93 sec)

mysql> SHOW WARNINGS;
***** 1. row *****
Level: Warning
Code: 1478
Message: Converted FIXED field 'c2' to DYNAMIC to enable online ADD COLUMN
***** 2. row *****
Level: Warning
Code: 1478
Message: Converted FIXED field 'c3' to DYNAMIC to enable online ADD COLUMN
2 rows in set (0.00 sec)
```

オンラインで追加するカラムのみが動的である必要があります。既存のカラムは必須ではありません。これには、次に示すように、`FIXED` でもある可能性があるテーブル主キーが含まれます:

```
mysql> CREATE TABLE t3 (
```



```
> c1 INT NOT NULL AUTO_INCREMENT PRIMARY KEY COLUMN_FORMAT FIXED
> )ENGINE=NDB;
Query OK, 0 rows affected (2.10 sec)

mysql> ALTER TABLE t3 ADD COLUMN c2 INT, ALGORITHM=INPLACE;
Query OK, 0 rows affected, 1 warning (0.78 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> SHOW WARNINGS;
***** 1. row *****
Level: Warning
Code: 1478
Message: Converted FIXED field 'c2' to DYNAMIC to enable online ADD COLUMN
1 row in set (0.00 sec)
```

名前の変更操作によって、カラムが `FIXED` から `DYNAMIC` のカラムフォーマットに変換されることはありません。`COLUMN_FORMAT` の詳細は、[セクション13.1.20「CREATE TABLE ステートメント」](#)を参照してください。

`KEY`、`CONSTRAINT` および `IGNORE` キーワードは、`ALGORITHM=INPLACE` を使用する `ALTER TABLE` ステートメントでサポートされています。

オンラインの `ALTER TABLE` ステートメントを使用して `MAX_ROWS` を 0 に設定することはできません。この操作を実行するには、`ALTER TABLE` のコピーを使用する必要があります。(Bug #21960004)

23.5.12 NDB_STORED_USER での分散 MySQL 権限

NDB 8.0.18 では、NDB Cluster に接続された SQL ノード間でユーザー、役割、および特権を共有および同期するための新しいメカニズムが導入されました。これを有効にするには、`NDB_STORED_USER` 権限を付与します。使用方法の詳細は、権限の説明を参照してください。

`NDB_STORED_USER` は、他の権限と同様に `SHOW GRANTS` の出力に出力されます。特権が共有されていることを確認するには、NDB Cluster ディストリビューションに付属の `ndb_select_all` ユーティリティを使用します (フォーマットを保持するためにラップされた出力の一部):

```
shell> ndb_select_all -d mysql ndb_sql_metadata
type name seq note sql_ddl_text
11 "jon'@'localhost'" 0 4 "CREATE USER 'jon'@'localhost'
IDENTIFIED WITH 'caching_sha2_password' AS
'$A$005${B};3!7t!'.EFyZA5K5DQHRvBvuRNYTIMeO0YeBIPpZotFRPjVTYzTA5b0' REQUIRE
NONE PASSWORD EXPIRE DEFAULT ACCOUNT UNLOCK PASSWORD HISTORY DEFAULT PASSWORD
REUSE INTERVAL DEFAULT PASSWORD REQUIRE CURRENT DEFAULT"
12 "jon'@'localhost'" 0 [NULL] "GRANT USAGE ON *.* TO 'jon'@'localhost'"
12 "jon'@'localhost'" 3 [NULL] "GRANT ALL PRIVILEGES ON `test`.* TO 'jon'@'localhost'"
12 "jon'@'localhost'" 2 [NULL] "GRANT ALL PRIVILEGES ON `mydb`.* TO 'jon'@'localhost'"
12 "jon'@'localhost'" 1 [NULL] "GRANT NDB_STORED_USER ON *.* TO 'jon'@'localhost'"
5 rows returned
```

`ndb_sql_metadata` は、`mysql` または他の MySQL クライアントを使用して表示できない特別な `NDB` テーブルです。

`GRANT NDB_STORED_USER ON *.* TO 'cluster_app_user'@'localhost'` などの `NDB_STORED_USER` 権限を付与するステートメントは、`SHOW CREATE USER cluster_app_user@localhost` および `SHOW GRANTS FOR cluster_app_user@localhost` のクエリーを使用してスナップショットを作成し、その結果を `ndb_sql_metadata` に格納するように `NDB` に指示することで機能します。その後、他の SQL ノードはスナップショットの読み取りおよび適用をリクエストされます。MySQL サーバーは、クラスタを起動して SQL ノードとして結合するたびに、これらのストア `CREATE USER` および `GRANT` ステートメントをクラスタスキーマの同期プロセスの一環として実行します。

元の SQL ノード以外の SQL ノードで SQL ステートメントが実行されると、そのステートメントは `NDBCLUSTER` ストレージエンジンのユーティリティスレッドで実行されます。これは、MySQL レプリケーションレプリカアプリケーションスレッドと同等のセキュリティ環境内で実行されます。

`NDB_STORED_USER` を使用したユーザーへの変更は、非同期に分散されることに注意してください。分散スキーマの変更操作は同期的に実行されるため、分散ユーザーまたはユーザーに対する変更後の次の分散スキーマの変更は同期ポイントとして機能します。保留中のユーザー変更は、スキーマ変更分散が開始される前に完了まで実行されません。この後、スキーマ変更自体が同期的に実行されます。たとえば、`DROP DATABASE` ステートメントが分散ユーザーの `DROP USER` に続く場合、ユーザーの削除がすべての SQL ノードで完了するまで、データベースの削除は実行されません。

複数の **GRANT**、**REVOKE** または複数の SQL ノードからの他のユーザー管理ステートメントによって、特定のユーザーの権限が異なる SQL ノードで相違する場合は、権限が正しいことがわかっている SQL ノードでこのユーザーに対して **GRANT NDB_STORED_USER** を発行することで、この問題を修正できます。これにより、権限の新しいスナップショットが取得され、他の SQL ノードと同期されます。

NDB Cluster 8.0 は、NDB 7.6 以前のリリースで実装されているように、**NDB ストレージエンジン**を使用するように MySQL 特権テーブルを変換することによって、NDB Cluster 内の SQL ノード間での MySQL ユーザーおよび特権の配布をサポートしていません (**Distributed Privileges Using Shared Grant Tables** を参照)。以前のリリースから NDB 8.0 へのアップグレードに対するこの変更の影響については、**セクション23.2.7「NDB Cluster のアップグレードおよびダウングレード」** を参照してください。

23.5.13 NDB API 統計のカウンタと変数

Ndb オブジェクトによって実行されるアクションや、これらのオブジェクトに影響を与えるアクションに関する多くのタイプの統計カウンタを使用できます。このようなアクションには、トランザクションの開始と終了 (または中止)、主キーおよび一意のキーの操作、テーブルスキャン、範囲スキャン、およびブルーニングスキャン、さまざまな操作が完了するまで待機している間にブロックされたスレッド、**NDBCLUSTER** によって送受信されたデータおよびイベントが含まれます。NDB API 呼び出しが行われるが、データノードがデータを送受信するたびに、NDB カーネル内でカウンタが増分されます。**mysqld** は、これらのカウンタをシステムステータス変数として公開します。それらの値は、**SHOW STATUS** の出力で読み取るか、パフォーマンススキーマ **session_status** または **global_status** テーブルをクエリーすることによって読み取ることができます。**NDB** テーブルを操作するステートメントの前後で値を比較すると、API レベルで実行された対応するアクション、およびステートメントを実行するコストを確認できます。

次の **SHOW STATUS** ステートメントを使用すると、これらのステータス変数をすべて一覧表示できます。

```
mysql> SHOW STATUS LIKE 'ndb_api%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Ndb_api_wait_exec_complete_count_session | 0 |
| Ndb_api_wait_scan_result_count_session | 0 |
| Ndb_api_wait_meta_request_count_session | 0 |
| Ndb_api_wait_nanos_count_session | 0 |
| Ndb_api_bytes_sent_count_session | 0 |
| Ndb_api_bytes_received_count_session | 0 |
| Ndb_api_trans_start_count_session | 0 |
| Ndb_api_trans_commit_count_session | 0 |
| Ndb_api_trans_abort_count_session | 0 |
| Ndb_api_trans_close_count_session | 0 |
| Ndb_api_pk_op_count_session | 0 |
| Ndb_api_uk_op_count_session | 0 |
| Ndb_api_table_scan_count_session | 0 |
| Ndb_api_range_scan_count_session | 0 |
| Ndb_api_pruned_scan_count_session | 0 |
| Ndb_api_scan_batch_count_session | 0 |
| Ndb_api_read_row_count_session | 0 |
| Ndb_api_trans_local_read_row_count_session | 0 |
| Ndb_api_event_data_count_injector | 0 |
| Ndb_api_event_nondata_count_injector | 0 |
| Ndb_api_event_bytes_count_injector | 0 |
| Ndb_api_wait_exec_complete_count_slave | 0 |
| Ndb_api_wait_scan_result_count_slave | 0 |
| Ndb_api_wait_meta_request_count_slave | 0 |
| Ndb_api_wait_nanos_count_slave | 0 |
| Ndb_api_bytes_sent_count_slave | 0 |
| Ndb_api_bytes_received_count_slave | 0 |
| Ndb_api_trans_start_count_slave | 0 |
| Ndb_api_trans_commit_count_slave | 0 |
| Ndb_api_trans_abort_count_slave | 0 |
| Ndb_api_trans_close_count_slave | 0 |
| Ndb_api_pk_op_count_slave | 0 |
| Ndb_api_uk_op_count_slave | 0 |
| Ndb_api_table_scan_count_slave | 0 |
| Ndb_api_range_scan_count_slave | 0 |
| Ndb_api_pruned_scan_count_slave | 0 |
| Ndb_api_scan_batch_count_slave | 0 |
| Ndb_api_read_row_count_slave | 0 |
| Ndb_api_trans_local_read_row_count_slave | 0 |
| Ndb_api_wait_exec_complete_count | 2 |
```

```

Ndb_api_wait_scan_result_count      | 3      |
Ndb_api_wait_meta_request_count     | 27     |
Ndb_api_wait_nanos_count            | 45612023 |
Ndb_api_bytes_sent_count            | 992    |
Ndb_api_bytes_received_count        | 9640   |
Ndb_api_trans_start_count           | 2      |
Ndb_api_trans_commit_count          | 1      |
Ndb_api_trans_abort_count           | 0      |
Ndb_api_trans_close_count           | 2      |
Ndb_api_pk_op_count                 | 1      |
Ndb_api_uk_op_count                 | 0      |
Ndb_api_table_scan_count            | 1      |
Ndb_api_range_scan_count            | 0      |
Ndb_api_pruned_scan_count           | 0      |
Ndb_api_scan_batch_count            | 0      |
Ndb_api_read_row_count              | 1      |
Ndb_api_trans_local_read_row_count  | 1      |
Ndb_api_event_data_count            | 0      |
Ndb_api_event_nondata_count         | 0      |
Ndb_api_event_bytes_count           | 0      |
+-----+-----+

```

60 rows in set (0.02 sec)

これらのステータス変数は、次に示すように、パフォーマンススキーマ `session_status` および `global_status` テーブルからも使用できます:

```

mysql> SELECT * FROM performance_schema.session_status
-> WHERE VARIABLE_NAME LIKE 'ndb_api%';
+-----+-----+
| VARIABLE_NAME                | VARIABLE_VALUE |
+-----+-----+
| NDB_API_WAIT_EXEC_COMPLETE_COUNT_SESSION | 2              |
| NDB_API_WAIT_SCAN_RESULT_COUNT_SESSION  | 0              |
| NDB_API_WAIT_META_REQUEST_COUNT_SESSION | 1              |
| NDB_API_WAIT_NANOS_COUNT_SESSION        | 8144375       |
| NDB_API_BYTES_SENT_COUNT_SESSION        | 68             |
| NDB_API_BYTES_RECEIVED_COUNT_SESSION    | 84             |
| NDB_API_TRANS_START_COUNT_SESSION       | 1              |
| NDB_API_TRANS_COMMIT_COUNT_SESSION      | 1              |
| NDB_API_TRANS_ABORT_COUNT_SESSION       | 0              |
| NDB_API_TRANS_CLOSE_COUNT_SESSION       | 1              |
| NDB_API_PK_OP_COUNT_SESSION             | 1              |
| NDB_API_UK_OP_COUNT_SESSION             | 0              |
| NDB_API_TABLE_SCAN_COUNT_SESSION        | 0              |
| NDB_API_RANGE_SCAN_COUNT_SESSION        | 0              |
| NDB_API_PRUNED_SCAN_COUNT_SESSION       | 0              |
| NDB_API_SCAN_BATCH_COUNT_SESSION        | 0              |
| NDB_API_READ_ROW_COUNT_SESSION          | 1              |
| NDB_API_TRANS_LOCAL_READ_ROW_COUNT_SESSION | 1              |
| NDB_API_EVENT_DATA_COUNT_INJECTOR       | 0              |
| NDB_API_EVENT_NONDATA_COUNT_INJECTOR    | 0              |
| NDB_API_EVENT_BYTES_COUNT_INJECTOR      | 0              |
| NDB_API_WAIT_EXEC_COMPLETE_COUNT_SLAVE  | 0              |
| NDB_API_WAIT_SCAN_RESULT_COUNT_SLAVE    | 0              |
| NDB_API_WAIT_META_REQUEST_COUNT_SLAVE   | 0              |
| NDB_API_WAIT_NANOS_COUNT_SLAVE          | 0              |
| NDB_API_BYTES_SENT_COUNT_SLAVE          | 0              |
| NDB_API_BYTES_RECEIVED_COUNT_SLAVE       | 0              |
| NDB_API_TRANS_START_COUNT_SLAVE         | 0              |
| NDB_API_TRANS_COMMIT_COUNT_SLAVE        | 0              |
| NDB_API_TRANS_ABORT_COUNT_SLAVE         | 0              |
| NDB_API_TRANS_CLOSE_COUNT_SLAVE         | 0              |
| NDB_API_PK_OP_COUNT_SLAVE               | 0              |
| NDB_API_UK_OP_COUNT_SLAVE               | 0              |
| NDB_API_TABLE_SCAN_COUNT_SLAVE          | 0              |
| NDB_API_RANGE_SCAN_COUNT_SLAVE          | 0              |
| NDB_API_PRUNED_SCAN_COUNT_SLAVE         | 0              |
| NDB_API_SCAN_BATCH_COUNT_SLAVE          | 0              |
| NDB_API_READ_ROW_COUNT_SLAVE            | 0              |
| NDB_API_TRANS_LOCAL_READ_ROW_COUNT_SLAVE | 0              |
| NDB_API_WAIT_EXEC_COMPLETE_COUNT        | 4              |
| NDB_API_WAIT_SCAN_RESULT_COUNT         | 3              |
| NDB_API_WAIT_META_REQUEST_COUNT        | 28             |
| NDB_API_WAIT_NANOS_COUNT                | 53756398      |

```

```

| NDB_API_BYTES_SENT_COUNT          | 1060 |
| NDB_API_BYTES_RECEIVED_COUNT     | 9724 |
| NDB_API_TRANS_START_COUNT        | 3    |
| NDB_API_TRANS_COMMIT_COUNT       | 2    |
| NDB_API_TRANS_ABORT_COUNT        | 0    |
| NDB_API_TRANS_CLOSE_COUNT        | 3    |
| NDB_API_PK_OP_COUNT              | 2    |
| NDB_API_UK_OP_COUNT              | 0    |
| NDB_API_TABLE_SCAN_COUNT         | 1    |
| NDB_API_RANGE_SCAN_COUNT         | 0    |
| NDB_API_PRUNED_SCAN_COUNT        | 0    |
| NDB_API_SCAN_BATCH_COUNT         | 0    |
| NDB_API_READ_ROW_COUNT           | 2    |
| NDB_API_TRANS_LOCAL_READ_ROW_COUNT | 2    |
| NDB_API_EVENT_DATA_COUNT         | 0    |
| NDB_API_EVENT_NONDATA_COUNT      | 0    |
| NDB_API_EVENT_BYTES_COUNT        | 0    |
+-----+-----+
60 rows in set (0.00 sec)

mysql> SELECT * FROM performance_schema.global_status
-> WHERE VARIABLE_NAME LIKE 'ndb_api%';
+-----+-----+
| VARIABLE_NAME                | VARIABLE_VALUE |
+-----+-----+
| NDB_API_WAIT_EXEC_COMPLETE_COUNT_SESSION | 2              |
| NDB_API_WAIT_SCAN_RESULT_COUNT_SESSION | 0              |
| NDB_API_WAIT_META_REQUEST_COUNT_SESSION | 1              |
| NDB_API_WAIT_NANOS_COUNT_SESSION      | 8144375        |
| NDB_API_BYTES_SENT_COUNT_SESSION      | 68              |
| NDB_API_BYTES_RECEIVED_COUNT_SESSION  | 84              |
| NDB_API_TRANS_START_COUNT_SESSION     | 1              |
| NDB_API_TRANS_COMMIT_COUNT_SESSION    | 1              |
| NDB_API_TRANS_ABORT_COUNT_SESSION     | 0              |
| NDB_API_TRANS_CLOSE_COUNT_SESSION     | 1              |
| NDB_API_PK_OP_COUNT_SESSION           | 1              |
| NDB_API_UK_OP_COUNT_SESSION           | 0              |
| NDB_API_TABLE_SCAN_COUNT_SESSION      | 0              |
| NDB_API_RANGE_SCAN_COUNT_SESSION      | 0              |
| NDB_API_PRUNED_SCAN_COUNT_SESSION     | 0              |
| NDB_API_SCAN_BATCH_COUNT_SESSION      | 0              |
| NDB_API_READ_ROW_COUNT_SESSION        | 1              |
| NDB_API_TRANS_LOCAL_READ_ROW_COUNT_SESSION | 1              |
| NDB_API_EVENT_DATA_COUNT_INJECTOR     | 0              |
| NDB_API_EVENT_NONDATA_COUNT_INJECTOR  | 0              |
| NDB_API_EVENT_BYTES_COUNT_INJECTOR    | 0              |
| NDB_API_WAIT_EXEC_COMPLETE_COUNT_SLAVE | 0              |
| NDB_API_WAIT_SCAN_RESULT_COUNT_SLAVE  | 0              |
| NDB_API_WAIT_META_REQUEST_COUNT_SLAVE | 0              |
| NDB_API_WAIT_NANOS_COUNT_SLAVE        | 0              |
| NDB_API_BYTES_SENT_COUNT_SLAVE        | 0              |
| NDB_API_BYTES_RECEIVED_COUNT_SLAVE    | 0              |
| NDB_API_TRANS_START_COUNT_SLAVE       | 0              |
| NDB_API_TRANS_COMMIT_COUNT_SLAVE      | 0              |
| NDB_API_TRANS_ABORT_COUNT_SLAVE       | 0              |
| NDB_API_TRANS_CLOSE_COUNT_SLAVE       | 0              |
| NDB_API_PK_OP_COUNT_SLAVE             | 0              |
| NDB_API_UK_OP_COUNT_SLAVE             | 0              |
| NDB_API_TABLE_SCAN_COUNT_SLAVE        | 0              |
| NDB_API_RANGE_SCAN_COUNT_SLAVE        | 0              |
| NDB_API_PRUNED_SCAN_COUNT_SLAVE       | 0              |
| NDB_API_SCAN_BATCH_COUNT_SLAVE        | 0              |
| NDB_API_READ_ROW_COUNT_SLAVE          | 0              |
| NDB_API_TRANS_LOCAL_READ_ROW_COUNT_SLAVE | 0              |
| NDB_API_WAIT_EXEC_COMPLETE_COUNT      | 4              |
| NDB_API_WAIT_SCAN_RESULT_COUNT        | 3              |
| NDB_API_WAIT_META_REQUEST_COUNT       | 28             |
| NDB_API_WAIT_NANOS_COUNT              | 53756398       |
| NDB_API_BYTES_SENT_COUNT              | 1060           |
| NDB_API_BYTES_RECEIVED_COUNT          | 9724           |
| NDB_API_TRANS_START_COUNT            | 3              |
| NDB_API_TRANS_COMMIT_COUNT           | 2              |
| NDB_API_TRANS_ABORT_COUNT            | 0              |
| NDB_API_TRANS_CLOSE_COUNT            | 3              |
| NDB_API_PK_OP_COUNT                  | 2              |

```

NDB_API_UK_OP_COUNT	0	
NDB_API_TABLE_SCAN_COUNT	1	
NDB_API_RANGE_SCAN_COUNT	0	
NDB_API_PRUNED_SCAN_COUNT	0	
NDB_API_SCAN_BATCH_COUNT	0	
NDB_API_READ_ROW_COUNT	2	
NDB_API_TRANS_LOCAL_READ_ROW_COUNT	2	
NDB_API_EVENT_DATA_COUNT	0	
NDB_API_EVENT_NONDATA_COUNT	0	
NDB_API_EVENT_BYTES_COUNT	0	
+-----+		

60 rows in set (0.00 sec)

各 **Ndb** オブジェクトは、それぞれに独自のカウンタを持っています。NDB API アプリケーションは、最適化やモニタリングで使用するためにカウンタの値を読み取ることができます。複数の **Ndb** オブジェクトを同時に使用するマルチスレッドクライアントの場合、特定の **Ndb_cluster_connection** に属するすべての **Ndb** オブジェクトからカウンタの合計ビューを取得することもできます。

4 セットのカウンタが表示されます。1 セットは、現在のセッションにのみ適用されます。その他の 3 セットはグローバルです。これは、**mysql** クライアントでは、これらの値をセッションとグローバルのどちらのステータス変数としても取得できることに関係ありません。つまり、**SHOW STATUS** で **SESSION** または **GLOBAL** キーワードを指定しても NDB API 統計ステータス変数について報告される値には影響せず、これらの各変数の値は、**session_status** または **global_status** テーブルの同等のカラムから取得されるかどうかにかかわらず同じです。

- セッションカウンタ (セッションに固有)

セッションカウンタは、現在のセッションで (のみ) 使用される **Ndb** オブジェクトに関連します。このようなオブジェクトをほかの MySQL クライアントが使用しても、これらのカウンタは影響を受けません。

標準の MySQL セッション変数との混同を最小限にするために、これらの NDB API セッションに対応する変数は、先頭に下線を付けた「**_session** 変数」と呼んでいます。

- レプリカカウンタ (グローバル)

このカウンタセットは、レプリカ SQL スレッドで使用される **Ndb** オブジェクト (存在する場合) に関連します。この **mysqld** がレプリカとして機能しない場合、または **NDB** テーブルを使用しない場合、これらのカウントはすべて 0 になります。

関連するステータス変数は、(先頭に下線を付けた)「**_slave** 変数」と呼んでいます。

- インジェクタカウンタ (グローバル)

インジェクタカウンタは、バイナリログインジェクタスレッドがクラスタイベントを待機するために使用する **Ndb** オブジェクトに関連します。バイナリログを書き込まない場合でも、NDB Cluster に接続された **mysqld** プロセスは、スキーマの変更などの一部のイベントを引き続き待機します。

NDB API インジェクタカウンタに対応するステータス変数は、(先頭に下線を付けた)「**_injector** 変数」と呼んでいます。

- サーバー (グローバル) カウンタ (グローバル)

このカウンタセットは、この **mysqld** によって現在使用されているすべての **Ndb** オブジェクトに関連します。これには、すべての MySQL クライアントアプリケーション、レプリカ SQL スレッド (存在する場合)、バイナリログインジェクタおよび **NDB** ユーティリティスレッドが含まれます。

これらのカウンタに対応するステータス変数は、「グローバル変数」または「**mysqld** レベルの変数」と呼んでいます。

(一般的なプリフィクス **Ndb_api** とともに) 変数名の部分文字列 **session**、**slave**、または **injector** で追加でフィルタ処理すると、特定の变数セットの値を取得できます。**_session** 変数の場合は、これを次に示すように実行できます。

```
mysql> SHOW STATUS LIKE 'ndb_api%session';
+-----+
| Variable_name | Value |
+-----+
| Ndb_api_wait_exec_complete_count_session | 2 |
```

```

| Ndb_api_wait_scan_result_count_session | 0 |
| Ndb_api_wait_meta_request_count_session | 1 |
| Ndb_api_wait_nanos_count_session | 8144375 |
| Ndb_api_bytes_sent_count_session | 68 |
| Ndb_api_bytes_received_count_session | 84 |
| Ndb_api_trans_start_count_session | 1 |
| Ndb_api_trans_commit_count_session | 1 |
| Ndb_api_trans_abort_count_session | 0 |
| Ndb_api_trans_close_count_session | 1 |
| Ndb_api_pk_op_count_session | 1 |
| Ndb_api_uk_op_count_session | 0 |
| Ndb_api_table_scan_count_session | 0 |
| Ndb_api_range_scan_count_session | 0 |
| Ndb_api_pruned_scan_count_session | 0 |
| Ndb_api_scan_batch_count_session | 0 |
| Ndb_api_read_row_count_session | 1 |
| Ndb_api_trans_local_read_row_count_session | 1 |
+-----+
18 rows in set (0.50 sec)

```

NDB API `mysqld` レベルのステータス変数のリストを取得するには、次のように、`ndb_api` で始まり、`_count` で終わる変数名でフィルタ処理します。

```

mysql> SELECT * FROM performance_schema.session_status
-> WHERE VARIABLE_NAME LIKE 'ndb_api%count';
+-----+
| VARIABLE_NAME | VARIABLE_VALUE |
+-----+
| NDB_API_WAIT_EXEC_COMPLETE_COUNT | 4 |
| NDB_API_WAIT_SCAN_RESULT_COUNT | 3 |
| NDB_API_WAIT_META_REQUEST_COUNT | 28 |
| NDB_API_WAIT_NANOS_COUNT | 53756398 |
| NDB_API_BYTES_SENT_COUNT | 1060 |
| NDB_API_BYTES_RECEIVED_COUNT | 9724 |
| NDB_API_TRANS_START_COUNT | 3 |
| NDB_API_TRANS_COMMIT_COUNT | 2 |
| NDB_API_TRANS_ABORT_COUNT | 0 |
| NDB_API_TRANS_CLOSE_COUNT | 3 |
| NDB_API_PK_OP_COUNT | 2 |
| NDB_API_UK_OP_COUNT | 0 |
| NDB_API_TABLE_SCAN_COUNT | 1 |
| NDB_API_RANGE_SCAN_COUNT | 0 |
| NDB_API_PRUNED_SCAN_COUNT | 0 |
| NDB_API_SCAN_BATCH_COUNT | 0 |
| NDB_API_READ_ROW_COUNT | 2 |
| NDB_API_TRANS_LOCAL_READ_ROW_COUNT | 2 |
| NDB_API_EVENT_DATA_COUNT | 0 |
| NDB_API_EVENT_NONDATA_COUNT | 0 |
| NDB_API_EVENT_BYTES_COUNT | 0 |
+-----+
21 rows in set (0.09 sec)

```

4 セットすべてのステータス変数に、すべてのカウンタが反映されるとはかぎりません。イベントカウンタ `DataEventsRecvdCount`、`NondataEventsRecvdCount`、および `EventBytesRecvdCount` の場合は、`_injector` と `mysqld` レベルの NDB API ステータス変数のみを使用できます。

```

mysql> SHOW STATUS LIKE 'ndb_api%event%';
+-----+
| Variable_name | Value |
+-----+
| Ndb_api_event_data_count_injector | 0 |
| Ndb_api_event_nondata_count_injector | 0 |
| Ndb_api_event_bytes_count_injector | 0 |
| Ndb_api_event_data_count | 0 |
| Ndb_api_event_nondata_count | 0 |
| Ndb_api_event_bytes_count | 0 |
+-----+
6 rows in set (0.00 sec)

```

次に示すように、その他の NDB API カウンタには、`_injector` ステータス変数が実装されていません。

```

mysql> SHOW STATUS LIKE 'ndb_api%injector%';
+-----+
| Variable_name | Value |
+-----+

```



```

+-----+-----+
| Ndb_api_event_data_count_injector | 0 |
| Ndb_api_event_nondata_count_injector | 0 |
| Ndb_api_event_bytes_count_injector | 0 |
+-----+-----+
3 rows in set (0.00 sec)

```

ステータス変数の名前は、対応するカウンタの名前に簡単に関連付けることができます。次の表に、各 NDB API 統計カウンタを、説明およびこのカウンタに対応する MySQL サーバーステータス変数の名前とともに一覧表示しています。

表 23.65 NDB API 統計カウンタ

カウンタ名	説明	ステータス変数 (統計タイプ別):
		<ul style="list-style-type: none"> セッション スレーブ (レプリカ) インジェクタ サーバー
WaitExecCompleteCount	操作の実行が完了するまで待機する間にスレッドがブロックされた回数。すべての <code>execute()</code> 呼び出しと、クライアントから見えない BLOB および自動インクリメント操作の暗黙的な実行が含まれます。	<ul style="list-style-type: none"> Ndb_api_wait_exec_complete_count_session Ndb_api_wait_exec_complete_count_slave [none] Ndb_api_wait_exec_complete_count
WaitScanResultCount	追加の結果やスキャンが閉じるまで待機するなど、スキャンベースの信号を待機する間にスレッドがブロックされた回数。	<ul style="list-style-type: none"> Ndb_api_wait_scan_result_count_session Ndb_api_wait_scan_result_count_slave [none] Ndb_api_wait_scan_result_count
WaitMetaRequestCount	メタベースの信号を待機する間にスレッドがブロックされた回数。これは、DDL 操作またはエポックが開始される (または終了する) まで待機しているときに発生する可能性があります。	<ul style="list-style-type: none"> Ndb_api_wait_meta_request_count_session Ndb_api_wait_meta_request_count_slave [none] Ndb_api_wait_meta_request_count
WaitNanosCount	データノードからの何らかのタイプの信号の待機にかかった合計時間 (ナノ秒)。	<ul style="list-style-type: none"> Ndb_api_wait_nanos_count_session Ndb_api_wait_nanos_count_slave [none] Ndb_api_wait_nanos_count
BytesSentCount	データノードに送信されたデータ量 (バイト単位)。	<ul style="list-style-type: none"> Ndb_api_bytes_sent_count_session Ndb_api_bytes_sent_count_slave [none] Ndb_api_bytes_sent_count
BytesRecvdCount	データノードから受信されたデータ量 (バイト単位)。	<ul style="list-style-type: none"> Ndb_api_bytes_received_count_session Ndb_api_bytes_received_count_slave [none]

カウンタ名	説明	ステータス変数 (統計タイプ別):
		<ul style="list-style-type: none"> セッション スレーブ (レプリカ) インジェクタ サーバー
		<ul style="list-style-type: none"> Ndb_api_bytes_received_count
TransStartCount	開始されたトランザクションの数。	<ul style="list-style-type: none"> Ndb_api_trans_start_count_session Ndb_api_trans_start_count_slave [none] Ndb_api_trans_start_count
TransCommitCount	コミットされたトランザクションの数。	<ul style="list-style-type: none"> Ndb_api_trans_commit_count_session Ndb_api_trans_commit_count_slave [none] Ndb_api_trans_commit_count
TransAbortCount	中止されたトランザクションの数。	<ul style="list-style-type: none"> Ndb_api_trans_abort_count_session Ndb_api_trans_abort_count_slave [none] Ndb_api_trans_abort_count
TransCloseCount	中止されたトランザクションの数。 (この値は、 TransCommitCount と TransAbortCount との合計より大きい場合があります。)	<ul style="list-style-type: none"> Ndb_api_trans_close_count_session Ndb_api_trans_close_count_slave [none] Ndb_api_trans_close_count
PkOpCount	主キーに基づいた操作または主キーを使用した操作の数。このカウントには、BLOB 部分テーブル操作、暗黙的なロック解除操作、自動インクリメント操作、および通常 MySQL クライアントから見える主キー操作が含まれません。	<ul style="list-style-type: none"> Ndb_api_pk_op_count_session Ndb_api_pk_op_count_slave [none] Ndb_api_pk_op_count
UkOpCount	一意のキーに基づいた操作または一意のキーを使用した操作の数。	<ul style="list-style-type: none"> Ndb_api_uk_op_count_session Ndb_api_uk_op_count_slave [none] Ndb_api_uk_op_count
TableScanCount	開始されたテーブルスキャンの数。これには、内部テーブルのスキャンが含まれます。	<ul style="list-style-type: none"> Ndb_api_table_scan_count_session Ndb_api_table_scan_count_slave [none] Ndb_api_table_scan_count

カウンタ名	説明	ステータス変数 (統計タイプ別):
		<ul style="list-style-type: none"> セッション スレーブ (レプリカ) インジェクタ サーバー
RangeScanCount	開始された範囲スキャンの数。	<ul style="list-style-type: none"> Ndb_api_range_scan_count_session Ndb_api_range_scan_count_slave [none] Ndb_api_range_scan_count
PrunedScanCount	単一パーティションにブルーニングされたスキャンの数。	<ul style="list-style-type: none"> Ndb_api_pruned_scan_count_session Ndb_api_pruned_scan_count_slave [none] Ndb_api_pruned_scan_count
ScanBatchCount	受信された行のバッチの数。(このコンテキストで、バッチとは単一フラグメントからのスキャン結果のセットです。)	<ul style="list-style-type: none"> Ndb_api_scan_batch_count_session Ndb_api_scan_batch_count_slave [none] Ndb_api_scan_batch_count
ReadRowCount	読み取られた行の合計数。主キー、一意のキー、またはスキャン操作を使用して読み取られた行が含まれます。	<ul style="list-style-type: none"> Ndb_api_read_row_count_session Ndb_api_read_row_count_slave [none] Ndb_api_read_row_count
TransLocalReadRowCount	トランザクションが実行された同じデータノードから読み取られた行数。	<ul style="list-style-type: none"> Ndb_api_trans_local_read_row_count_session Ndb_api_trans_local_read_row_count_slave [none] Ndb_api_trans_local_read_row_count
DataEventsRecvdCount	受信された行変更イベントの数。	<ul style="list-style-type: none"> [none] [none] Ndb_api_event_data_count_injector Ndb_api_event_data_count
NondataEventsRecvdCount	受信された行変更イベント以外のイベントの数。	<ul style="list-style-type: none"> [none] [none] Ndb_api_event_nondata_count_injector Ndb_api_event_nondata_count
EventBytesRecvdCount	受信されたイベントのバイト数。	<ul style="list-style-type: none"> [none]

カウンタ名	説明	ステータス変数 (統計タイプ別):
		<ul style="list-style-type: none"> セッション スレーブ (レプリカ) インジェクタ サーバー
		<ul style="list-style-type: none"> [none] Ndb_api_event_bytes_count_injector Ndb_api_event_bytes_count

コミットされたトランザクションのすべてのカウント、つまり [TransCommitCount](#) カウンタのステータス変数を確認するには、次のように、[SHOW STATUS](#) の結果を部分文字列 [trans_commit_count](#) でフィルタ処理できます。

```
mysql> SHOW STATUS LIKE '%trans_commit_count%';
+-----+-----+
| Variable_name          | Value |
+-----+-----+
| Ndb_api_trans_commit_count_session | 1 |
| Ndb_api_trans_commit_count_slave | 0 |
| Ndb_api_trans_commit_count      | 2 |
+-----+-----+
3 rows in set (0.00 sec)
```

ここから、現在の [mysql](#) クライアントセッションで 1 つのトランザクションがコミットされ、この [mysqld](#) で、それが最後に再起動されてから、2 つのトランザクションがコミットされたことを判断できます。

ステートメントを実行した直前と直後で、対応する [_session](#) ステータス変数の値を比較すると、特定の SQL ステートメントによって、さまざまな NDB API カウンタがどのように増分されているかを確認できます。この例では、[SHOW STATUS](#) から初期値を取得したあとに、[test](#) データベースに、単一のカラムを持つ [t](#) という名前の [NDB](#) テーブルを作成します。

```
mysql> SHOW STATUS LIKE 'ndb_api%session%';
+-----+-----+
| Variable_name          | Value |
+-----+-----+
| Ndb_api_wait_exec_complete_count_session | 2 |
| Ndb_api_wait_scan_result_count_session | 0 |
| Ndb_api_wait_meta_request_count_session | 3 |
| Ndb_api_wait_nanos_count_session | 820705 |
| Ndb_api_bytes_sent_count_session | 132 |
| Ndb_api_bytes_received_count_session | 372 |
| Ndb_api_trans_start_count_session | 1 |
| Ndb_api_trans_commit_count_session | 1 |
| Ndb_api_trans_abort_count_session | 0 |
| Ndb_api_trans_close_count_session | 1 |
| Ndb_api_pk_op_count_session | 1 |
| Ndb_api_uk_op_count_session | 0 |
| Ndb_api_table_scan_count_session | 0 |
| Ndb_api_range_scan_count_session | 0 |
| Ndb_api_pruned_scan_count_session | 0 |
| Ndb_api_scan_batch_count_session | 0 |
| Ndb_api_read_row_count_session | 1 |
| Ndb_api_trans_local_read_row_count_session | 1 |
+-----+-----+
18 rows in set (0.00 sec)

mysql> USE test;
Database changed
mysql> CREATE TABLE t (c INT) ENGINE NDBCLUSTER;
Query OK, 0 rows affected (0.85 sec)
```

この時点で、次に示す (出力で変更された行を強調表示しています) ように、新しい [SHOW STATUS](#) ステートメントを実行し、変更を確認できます。

```
mysql> SHOW STATUS LIKE 'ndb_api%session%';
+-----+-----+
| Variable_name          | Value |
+-----+-----+
| Ndb_api_wait_exec_complete_count_session | 8     |
| Ndb_api_wait_scan_result_count_session   | 0     |
| Ndb_api_wait_meta_request_count_session  | 17    |
| Ndb_api_wait_nanos_count_session        | 706871709 |
| Ndb_api_bytes_sent_count_session        | 2376  |
| Ndb_api_bytes_received_count_session    | 3844  |
| Ndb_api_trans_start_count_session       | 4     |
| Ndb_api_trans_commit_count_session      | 4     |
| Ndb_api_trans_abort_count_session       | 0     |
| Ndb_api_trans_close_count_session       | 4     |
| Ndb_api_pk_op_count_session             | 6     |
| Ndb_api_uk_op_count_session             | 0     |
| Ndb_api_table_scan_count_session        | 0     |
| Ndb_api_range_scan_count_session        | 0     |
| Ndb_api_pruned_scan_count_session       | 0     |
| Ndb_api_scan_batch_count_session        | 0     |
| Ndb_api_read_row_count_session          | 2     |
| Ndb_api_trans_local_read_row_count_session | 1     |
+-----+-----+
18 rows in set (0.00 sec)
```

同様に、`t` に行を挿入することで発生した NDB API 統計カウンタの変更も確認できます。次に示すように、行を挿入してから、前の例で使用したのと同じ `SHOW STATUS` ステートメントを実行します。

```
mysql> INSERT INTO t VALUES (100);
Query OK, 1 row affected (0.00 sec)

mysql> SHOW STATUS LIKE 'ndb_api%session%';
+-----+-----+
| Variable_name          | Value |
+-----+-----+
| Ndb_api_wait_exec_complete_count_session | 11    |
| Ndb_api_wait_scan_result_count_session   | 6     |
| Ndb_api_wait_meta_request_count_session  | 20    |
| Ndb_api_wait_nanos_count_session        | 707370418 |
| Ndb_api_bytes_sent_count_session        | 2724  |
| Ndb_api_bytes_received_count_session    | 4116  |
| Ndb_api_trans_start_count_session       | 7     |
| Ndb_api_trans_commit_count_session      | 6     |
| Ndb_api_trans_abort_count_session       | 0     |
| Ndb_api_trans_close_count_session       | 7     |
| Ndb_api_pk_op_count_session             | 8     |
| Ndb_api_uk_op_count_session             | 0     |
| Ndb_api_table_scan_count_session        | 1     |
| Ndb_api_range_scan_count_session        | 0     |
| Ndb_api_pruned_scan_count_session       | 0     |
| Ndb_api_scan_batch_count_session        | 0     |
| Ndb_api_read_row_count_session          | 3     |
| Ndb_api_trans_local_read_row_count_session | 2     |
+-----+-----+
18 rows in set (0.00 sec)
```

これらの結果から、いくつかのことを観察できます。

- 明示的な主キーなしで `t` を作成しましたが、その際に 5 つの主キー操作が実行されました (`Ndb_api_pk_op_count_session` の「前」と「後」の値の差、つまり 6 から 1 を引く)。これは、NDB ストレージエンジンを使用しているすべてのテーブルの機能である非表示の主キーの作成を反映しています。
- `Ndb_api_wait_nanos_count_session` の連続した値を比較すると、`CREATE TABLE` ステートメントを実装している NDB API 操作が、`INSERT` によって実行された操作 ($707370418 - 706871709 = 498709$ ナノ秒、つまり約 0.0005 秒) よりも大幅に長い時間 ($706871709 - 820705 = 706051004$ ナノ秒、つまり約 0.7 秒)、データノードからの応答を待機したことを確認できます。mysql クライアントでこれらのステートメントについてレポートされた実行時間は、これらの数値に大まかに関連しています。

十分な (ナノ秒) 時間解決がないプラットフォームでは、非常に迅速に実行される SQL ステートメントが `Ndb_api_wait_nanos_count_session`、`Ndb_api_wait_nanos_count_slave`、または `Ndb_api_wait_nanos_count` の値に表示されないことがあるため、`WaitNanosCount` NDB API カウンタの値の小さな変更がある場合があります。

- `Ndb_api_read_row_count_session` と `Ndb_api_trans_local_read_row_count_session` の値が増加したことを反映して、`INSERT` ステートメントによって `ReadRowCount` と `TransLocalReadRowCount` の両方の NDB API 統計カウンタが増分されています。

23.5.14 ndbinfo: NDB Cluster 情報データベース

`ndbinfo` は NDB Cluster に固有の情報を含むデータベースです。

このデータベースには多数のテーブルが含まれており、それぞれ NDB Cluster ノードのステータス、リソース使用率、および操作に関する異なる種類のデータを提供します。次のいくつかのセクションで、これらの各テーブルに関する詳細な情報を見つけることができます。

`ndbinfo` は NDB Cluster のサポートとともに MySQL Server に含まれており、特別なコンパイルや構成の手順は必要ありません。テーブルは、クラスタへの接続時に MySQL Server によって作成されます。`SHOW PLUGINS` を使用すると、特定の MySQL サーバーインスタンスで `ndbinfo` サポートがアクティブになっていることを確認できます。`ndbinfo` サポートが有効になっている場合は、次に (強調表示したテキストで) 示すように、`Name` カラムに `ndbinfo` が含まれ、`Status` カラムに `ACTIVE` が含まれる行が表示されます。

```
mysql> SHOW PLUGINS;
+-----+-----+-----+-----+-----+
| Name          | Status | Type          | Library | License |
+-----+-----+-----+-----+-----+
| binlog        | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| mysql_native_password | ACTIVE | AUTHENTICATION | NULL    | GPL     |
| sha256_password | ACTIVE | AUTHENTICATION | NULL    | GPL     |
| MRG_MYISAM    | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| MEMORY        | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| CSV           | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| MyISAM        | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| InnoDB        | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| INNODB_TRX    | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_LOCKS  | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_LOCK_WAITS | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMP    | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMP_RESET | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMPMEM | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMPMEM_RESET | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMP_PER_INDEX | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMP_PER_INDEX_RESET | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_BUFFER_PAGE | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_BUFFER_PAGE_LRU | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_BUFFER_POOL_STATS | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_TEMP_TABLE_INFO | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_METRICS | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_FT_DEFAULT_STOPWORD | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_FT_DELETED | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_FT_BEING_DELETED | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_FT_CONFIG | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_FT_INDEX_CACHE | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_FT_INDEX_TABLE | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_SYS_TABLES | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_SYS_TABLESTATS | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_SYS_INDEXES | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_SYS_COLUMNS | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_SYS_FIELDS | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_SYS_FOREIGN | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_SYS_FOREIGN_COLS | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_SYS_TABLESPACES | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_SYS_DATAFILES | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_SYS_VIRTUAL | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| PERFORMANCE_SCHEMA | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| ndbCluster    | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| ndbinfo       | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| ndb_transid_mysql_connection_map | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| BLACKHOLE     | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| ARCHIVE       | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| partition     | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| ngram         | ACTIVE | FTPARSER       | NULL    | GPL     |
+-----+-----+-----+-----+-----+
46 rows in set (0.00 sec)
```


また、次に (強調表示されたテキストで) 示すように、Engine カラムに ndbinfo が含まれ、Support カラムに YES が含まれる行があるかどうかについて、SHOW ENGINES の出力をチェックすることで、これを実行することもできます。

```
mysql> SHOW ENGINES;
+-----+-----+-----+-----+-----+-----+
| Engine | Support | Comment | Transactions | XA | Savepoints |
+-----+-----+-----+-----+-----+-----+
1. row
Engine: ndbcluster
Support: YES
Comment: Clustered, fault-tolerant tables
Transactions: YES
XA: NO
Savepoints: NO
2. row
Engine: CSV
Support: YES
Comment: CSV storage engine
Transactions: NO
XA: NO
Savepoints: NO
3. row
Engine: InnoDB
Support: DEFAULT
Comment: Supports transactions, row-level locking, and foreign keys
Transactions: YES
XA: YES
Savepoints: YES
4. row
Engine: BLACKHOLE
Support: YES
Comment: /dev/null storage engine (anything you write to it disappears)
Transactions: NO
XA: NO
Savepoints: NO
5. row
Engine: MyISAM
Support: YES
Comment: MyISAM storage engine
Transactions: NO
XA: NO
Savepoints: NO
6. row
Engine: MRG_MYISAM
Support: YES
Comment: Collection of identical MyISAM tables
Transactions: NO
XA: NO
Savepoints: NO
7. row
Engine: ARCHIVE
Support: YES
Comment: Archive storage engine
Transactions: NO
XA: NO
Savepoints: NO
8. row
Engine: ndbinfo
Support: YES
Comment: NDB Cluster system information storage engine
Transactions: NO
XA: NO
Savepoints: NO
9. row
Engine: PERFORMANCE_SCHEMA
Support: YES
Comment: Performance Schema
Transactions: NO
XA: NO
Savepoints: NO
10. row
Engine: MEMORY
Support: YES
Comment: Hash based, stored in memory, useful for temporary tables
Transactions: NO
XA: NO
```

```
Savepoints: NO  
10 rows in set (0.00 sec)
```

ndbinfo サポートが有効になっている場合は、mysql や別の MySQL クライアントで SQL ステートメントを使用して、ndbinfo にアクセスできます。たとえば、次に示すように、SHOW DATABASES の出力に ndbinfo がリストされます (強調表示されたテキスト):

```
mysql> SHOW DATABASES;  
+-----+  
| Database |  
+-----+  
| information_schema |  
| mysql |  
| ndbinfo |  
| performance_schema |  
| sys |  
+-----+  
5 rows in set (0.04 sec)
```

--ndbcluster オプションを付けて mysqld プロセスが起動されなかった場合は、ndbinfo を使用できず、SHOW DATABASES によって表示されません。mysqld が以前 NDB Cluster に接続されていたものの、クラスタが使用できなくなった場合 (クラスタのシャットダウン、ネットワーク接続の損失などのイベントのため)、ndbinfo とそのテーブルは表示されたままですが、(blocks や config_params 以外の) テーブルにアクセスしようとすると「NDBINFO からエラー 157NDB への接続に失敗しましたが発生しました」で失敗します。

blocks および config_params テーブルを除いて、ndbinfo 「テーブル」と呼ばれるものは、実際には、MySQL サーバーから通常見えない内部 NDB テーブルから生成されるビューです。

ndbinfo テーブルはすべて読み取り専用であり、クエリーの実行時に要求に応じて生成されます。これらの多くはデータノードによって並列して生成されますが、その他は特定の SQL ノードに固有であるため、一貫性のあるスナップショットを提供する保証はありません。

さらに、ndbinfo テーブルでは、結合のプッシュダウンがサポートされていません。そのため、クエリーで WHERE 句を使用している場合でも、大きな ndbinfo テーブルの結合には、リクエスト元の API ノードに大量のデータを転送する必要があることがあります。

クエリーキャッシュに、ndbinfo テーブルが含まれていません。(Bug #59831)

その他のデータベースの場合と同様に、USE ステートメントで ndbinfo データベースを選択してから、SHOW TABLES ステートメントを発行すると、次のようなテーブルのリストを取得できます。

```
mysql> USE ndbinfo;  
Database changed  
  
mysql> SHOW TABLES;  
+-----+  
| Tables_in_ndbinfo |  
+-----+  
| arbitrator_validity_detail |  
| arbitrator_validity_summary |  
| blocks |  
| cluster_locks |  
| cluster_operations |  
| cluster_transactions |  
| config_nodes |  
| config_params |  
| config_values |  
| counters |  
| cpustat |  
| cpustat_1sec |  
| cpustat_20sec |  
| cpustat_50ms |  
| dict_obj_info |  
| dict_obj_types |  
| disk_write_speed_aggregate |  
| disk_write_speed_aggregate_node |  
| disk_write_speed_base |  
| diskpagebuffer |  
| error_messages |  
| locks_per_fragment |
```

```
logbuffers |
logspaces |
membership |
memory_per_fragment |
memoryusage |
nodes |
operations_per_fragment |
processes |
resources |
restart_info |
server_locks |
server_operations |
server_transactions |
table_distribution_status |
table_fragments |
table_info |
table_replicas |
tc_time_track_stats |
threadblocks |
threads |
threadstat |
transporters |
+-----+
44 rows in set (0.00 sec)
```

NDB 8.0 では、すべての `ndbinfo` テーブルが NDB ストレージエンジンを使用しますが、前に説明したように、`SHOW ENGINES` および `SHOW PLUGINS` の出力には引き続き `ndbinfo` エントリが表示されます。

通常の想定どおりに、これらのテーブルに対して `SELECT` ステートメントを実行できます。

```
mysql> SELECT * FROM memoryusage;
+-----+-----+-----+-----+-----+
| node_id | memory_type | used | used_pages | total | total_pages |
+-----+-----+-----+-----+-----+
| 5 | Data memory | 753664 | 23 | 1073741824 | 32768 |
| 5 | Index memory | 163840 | 20 | 1074003968 | 131104 |
| 5 | Long message buffer | 2304 | 9 | 67108864 | 262144 |
| 6 | Data memory | 753664 | 23 | 1073741824 | 32768 |
| 6 | Index memory | 163840 | 20 | 1074003968 | 131104 |
| 6 | Long message buffer | 2304 | 9 | 67108864 | 262144 |
+-----+-----+-----+-----+-----+
6 rows in set (0.02 sec)
```

`memoryusage` テーブルを使用する次の 2 つの `SELECT` ステートメントのような、さらに複雑なクエリーも実行できます。

```
mysql> SELECT SUM(used) as 'Data Memory Used, All Nodes'
> FROM memoryusage
> WHERE memory_type = 'Data memory';
+-----+
| Data Memory Used, All Nodes |
+-----+
| 6460 |
+-----+
1 row in set (0.37 sec)

mysql> SELECT SUM(max) as 'Total IndexMemory Available'
> FROM memoryusage
> WHERE memory_type = 'Index memory';
+-----+
| Total IndexMemory Available |
+-----+
| 25664 |
+-----+
1 row in set (0.33 sec)
```

`ndbinfo` のテーブル名およびカラム名では、大/小文字が区別されます (`ndbinfo` データベース自体の名前と同様)。これらの識別子は小文字です。大文字と小文字を誤って使用すると、次の例で示すようなエラーが発生します。

```
mysql> SELECT * FROM nodes;
+-----+-----+-----+
| node_id | uptime | status | start_phase |
+-----+-----+-----+

```

```
| 1 | 13602 | STARTED | 0 |  
| 2 | 16 | STARTED | 0 |  
+-----+-----+-----+  
2 rows in set (0.04 sec)
```

```
mysql> SELECT * FROM Nodes;  
ERROR 1146 (42S02): Table 'ndbinfo.Nodes' doesn't exist
```

mysqldump は、ndbinfo データベースを完全に無視し、どの出力からも除外します。このことは、`--databases` または `--all-databases` オプションを使用する場合でも当てはまります。

NDB Cluster は、NDB Cluster ディスクデータストレージに使用されるファイルに関する情報を含む `FILES` テーブルや、トランザクション、トランザクションコーディネータ、NDB Cluster API ノード間の関係を示す `ndb_transid_mysql_connection_map` テーブルなど、`INFORMATION_SCHEMA` 情報データベース内のテーブルも保持します。詳細は、テーブルまたは [セクション23.5.15「NDB Cluster の INFORMATION_SCHEMA テーブル」](#) の説明を参照してください。

23.5.14.1 ndbinfo arbitrator_validity_detail テーブル

`arbitrator_validity_detail` テーブルには、クラスタ内の各データノードがアービトレータを持っているビューが表示されます。これは、`membership` テーブルのサブセットです。

`arbitrator_validity_detail` テーブルには、次のカラムがあります:

- `node_id`
このノードのノード ID
- `arbitrator`
アービトレータのノード ID
- `arb_ticket`
アービトレーションを追跡するために使用される内部識別子
- `arb_connected`
このノードがアービトレータに接続されているかどうか (`Yes` または `No` のいずれか)
- `arb_state`
アービトレーションの状態

メモ

このノード ID は、`ndb_mgm -e "SHOW"` でレポートされるものと同じです。

すべてのノードで `arb_state` の値が同じであるだけでなく、`arbitrator` と `arb_ticket` の値も同じです。可能性のある `arb_state` の値は `ARBIT_NULL`、`ARBIT_INIT`、`ARBIT_FIND`、`ARBIT_PREP1`、`ARBIT_PREP2`、`ARBIT_START`、`ARBIT_RUN`、`ARBIT_CHOOSE`、および `UNKNOWN` です。

`arb_connected` は、現在のノードが `arbitrator` に接続されているかどうかを示します。

23.5.14.2 ndbinfo arbitrator_validity_summary テーブル

`arbitrator_validity_summary` テーブルは、クラスタのデータノードに関するアービトレータの複合ビューを示します。

`arbitrator_validity_summary` テーブルには、次のカラムがあります:

- `arbitrator`
アービトレータのノード ID

- [arb_ticket](#)

アービトレーションを追跡するために使用される内部識別子

- [arb_connected](#)

このアービトレータがクラスタに接続されているかどうか

- [consensus_count](#)

このノードがアービトレータとして表示されるデータノードの数 (Yes または No のいずれか)

メモ

通常の操作では、このテーブルには、相当に長い期間でも 1 行しか含まれません。やや長い期間で複数行が含まれる場合は、一部のノードがアービトレータに接続されていないか、またはすべてのノードが接続されているが、同じアービトレータを承認していません。

[arbitorator](#) カラムは、アービトレータのノード ID を示します。

[arb_ticket](#) は、このアービトレータで使用される内部識別子です。

[arb_connected](#) は、このノードがアービトレータとしてクラスタに接続されているかどうかを示します。

23.5.14.3 ndbinfo backup_id テーブル

このテーブルでは、このクラスタに対して最近開始されたバックアップの ID を確認できます。

[backup_id](#) テーブルには、[ndb_mgm](#) クライアントの [START BACKUP](#) コマンドを使用して作成されたバックアップ ID に対応する単一カラムの [id](#) が含まれます。このテーブルには単一行が含まれます。

例: NDB 管理クライアントで次の一連の [START BACKUP](#) コマンドが発行され、クラスタが最初に起動されてからほかのバックアップが作成されていないとします:

```
ndb_mgm> START BACKUP
Waiting for completed, this may take several minutes
Node 5: Backup 1 started from node 50
Node 5: Backup 1 started from node 50 completed
StartGCP: 27894 StopGCP: 27897
#Records: 2057 #LogRecords: 0
Data: 51580 bytes Log: 0 bytes
ndb_mgm> START BACKUP 5
Waiting for completed, this may take several minutes
Node 5: Backup 5 started from node 50
Node 5: Backup 5 started from node 50 completed
StartGCP: 27905 StopGCP: 27908
#Records: 2057 #LogRecords: 0
Data: 51580 bytes Log: 0 bytes
ndb_mgm> START BACKUP
Waiting for completed, this may take several minutes
Node 5: Backup 6 started from node 50
Node 5: Backup 6 started from node 50 completed
StartGCP: 27912 StopGCP: 27915
#Records: 2057 #LogRecords: 0
Data: 51580 bytes Log: 0 bytes
ndb_mgm> START BACKUP 3
Connected to Management Server at: localhost:1186
Waiting for completed, this may take several minutes
Node 5: Backup 3 started from node 50
Node 5: Backup 3 started from node 50 completed
StartGCP: 28149 StopGCP: 28152
#Records: 2057 #LogRecords: 0
Data: 51580 bytes Log: 0 bytes
ndb_mgm>
```

この後、[backup_id](#) テーブルには、[mysql](#) クライアントを使用して、次に示す単一行が含まれます:

```
mysql> USE ndbinfo;
```

```
Database changed
mysql> SELECT * FROM backup_id;
+-----+
| id |
+-----+
| 3 |
+-----+
1 row in set (0.00 sec)
```

バックアップが見つからない場合、テーブルには 0 を id 値として含む単一行が含まれます。

NDB 8.0.24 に `backup_id` テーブルが追加されました。

23.5.14.4 ndbinfo blocks テーブル

`blocks` テーブルは、すべての NDB カーネルブロックの名前と内部 ID を単に格納する静的テーブルです ([NDB Kernel Blocks](#) を参照してください)。これは、人間が読める出力を生成するためにブロック番号をブロック名にマップする際に、その他の `ndbinfo` テーブル (大部分が実際にはビューです) で使用されます。

`blocks` テーブルには、次のカラムがあります:

- `block_number`
ブロック番号
- `block_name`
ブロック名

メモ

すべてのブロック名のリストを取得するには、`SELECT block_name FROM ndbinfo.blocks` を実行するだけです。これは静的テーブルですが、NDB Cluster のリリースによって内容が異なる場合があります。

23.5.14.5 ndbinfo cluster_locks テーブル

`cluster_locks` テーブルは NDB Cluster 内の `NDB` テーブルのロックを保持および待機している現在のロックリクエストに関する情報を提供し、`cluster_operations` へのコンパニオンテーブルとして使用されます。`cluster_locks` テーブルから取得した情報は、ストールおよびデッドロックの調査に役立つ場合があります。

`cluster_locks` テーブルには、次のカラムがあります:

- `node_id`
レポートノードの ID
- `block_instance`
レポート LDM インスタンスの ID
- `tableid`
この行を含むテーブルの ID
- `fragmentid`
ロックされた行を含むフラグメントの ID
- `rowid`
ロックされた行の ID
- `transid`
トランザクション ID

- `mode`
ロックリクエストモード
- `state`
ロック状態
- `detail`
これが行ロックキューで最初にロックを保持しているかどうか
- `op`
操作タイプ
- `duration_millis`
ロックの待機または保持にかかったミリ秒
- `lock_num`
ロックオブジェクトの ID
- `waiting_for`
この ID のロックを待機中

メモ

テーブル ID (`tableid` カラム) は内部的に割り当てられ、他の `ndbinfo` テーブルで使用されているものと同じです。また、`ndb_show_tables` の出力にも表示されます。

トランザクション ID (`transid` カラム) は、NDB API によって生成された、現在のロックを要求または保持しているトランザクションの識別子です。

`mode` カラムにはロックモードが表示されます。これは常に、`S` (共有ロックを示す) または `X` (排他ロック) のいずれかです。トランザクションが特定の行の排他ロックを保持している場合、その行の他のすべてのロックは同じトランザクション ID を持ちます。

`state` カラムにはロック状態が表示されます。その値は、常に `H` (保持) または `W` (待機) のいずれかです。待機中のロックリクエストは、別のトランザクションによって保持されているロックを待機します。

`detail` カラムに `*` (アスタリスク文字) が含まれている場合、これは、このロックが影響を受ける行ロックキューの最初の保持ロックであることを意味します。それ以外の場合、このカラムは空です。この情報は、ロックリクエストのリスト内の一意のエントリを識別するのに役立ちます。

`op` カラムには、ロックをリクエストしている操作のタイプが表示されます。これは常に、`READ`, `INSERT`, `UPDATE`, `DELETE`, `SCAN` または `REFRESH` のいずれかの値です。

`duration_millis` カラムには、このロックリクエストがロックを待機または保持しているミリ秒数が表示されます。待機中のリクエストに対してロックが付与されると、これは 0 にリセットされます。

ロック ID (`lockid` カラム) は、このノードおよびブロックインスタンスに対して一意です。

ロック状態は `lock_state` カラムに表示されます。これが `W` の場合、ロックは付与を待機しており、`waiting_for` カラムには、このリクエストが待機しているロックオブジェクトのロック ID が表示されます。それ以外の場合、`waiting_for` カラムは空です。`waiting_for` は、`node_id`, `block_instance`, `tableid`, `fragmentid` および `rowid` で識別される同じ行のロックのみを参照できます。

23.5.14.6 ndbinfo cluster_operations テーブル

`cluster_operations` テーブルは、NDB Cluster 内のすべてのアクティビティーの操作ごとの (ステートフル主キー `op`) ビューを、ローカルデータ管理 (LQH) ブロックの観点から提供します (`The DBLQH Block` を参照)。

`cluster_operations` テーブルには、次のカラムがあります:

- `node_id`
レポート LQH ブロックのノード ID
- `block_instance`
LQH ブロックインスタンス
- `transid`
トランザクション ID
- `operation_type`
操作のタイプ (可能性のある値についてはテキストを参照)
- `state`
操作の状態 (可能性のある値についてはテキストを参照)
- `tableid`
テーブル ID
- `fragmentid`
フラグメント ID
- `client_node_id`
クライアントノード ID
- `client_block_ref`
クライアントのブロック参照
- `tc_node_id`
トランザクションコーディネータノード ID
- `tc_block_no`
トランザクションコーディネータブロック番号
- `tc_block_instance`
トランザクションコーディネータブロックインスタンス

メモ

トランザクション ID は、NDB API の `getTransactionId()` メソッドを使用して取得できる一意の 64 ビット数値です。(現在、MySQL サーバーは進行中のトランザクションの NDB API トランザクション ID を公開しません。)

`operation_type` カラムは、`READ`、`READ-SH`、`READ-EX`、`INSERT`、`UPDATE`、`DELETE`、`WRITE`、`UNLOCK`、`REFRESH`、`SCAN`、`SCAN-SH`、`SCAN-EX`、または `<unknown>` のいずれかの値を取ることができます。

`state` カラム

は、`ABORT_QUEUED`、`ABORT_STOPPED`、`COMMITTED`、`COMMIT_QUEUED`、`COMMIT_STOPPED`、`COPY_CLOSE_STOPPED` または `WAIT_TUP_TO_ABORT` のいずれかの値を取ることができます。(`ndbinfo_show_hidden` を有効にして MySQL サーバーが実行されている場合は、通常は非表示になっている `ndb$dblqh_tcconnect_state` テーブルから選択することで、この状態のリストを表示できます。)

`ndb_show_tables` の出力をチェックして、テーブル ID から NDB テーブルの名前を取得できます。

`fragid` は、`ndb_desc --extra-partition-info` (短縮形式 `-p`) の出力で見られるパーティション番号と同じです。

`client_node_id` および `client_block_ref` では、`client` は NDB Cluster API または SQL ノード (つまり、NDB API クライアントまたはクラスタに接続された MySQL Server) を指します。

`block_instance` および `tc_block_instance` カラムには、それぞれ `DBLQH` および `DBTC` ブロックインスタンス番号が表示されます。これらをブロック名とともに使用して、`threadblocks` テーブルから特定のスレッドに関する情報を取得できます。

23.5.14.7 ndbinfo cluster_transactions テーブル

`cluster_transactions` テーブルには、NDB Cluster 内で進行中のすべてのトランザクションに関する情報が表示されません。

`cluster_transactions` テーブルには、次のカラムがあります:

- `node_id`
トランザクションコーディネータのノード ID
- `block_instance`
TC ブロックインスタンス
- `transid`
トランザクション ID
- `state`
操作の状態 (可能性のある値についてはテキストを参照)
- `count_operations`
トランザクション内のステートフルな主キー操作 (DML 操作に加えて、ロックを伴う読み取りを含む) の数
- `outstanding_operations`
ローカルデータ管理ブロックでまだ実行されている操作
- `inactive_seconds`
API の待機に要した時間
- `client_node_id`
クライアントノード ID
- `client_block_ref`
クライアントのブロック参照

メモ

トランザクション ID は、NDB API の `getTransactionId()` メソッドを使用して取得できる一意の 64 ビット数値です。(現在、MySQL サーバーは進行中のトランザクションの NDB API トランザクション ID を公開しません。)

`block_instance` は、カーネルブロックのインスタンスを指します。この番号は、ブロック名とともに使用して、`threadblocks` テーブル内の特定のインスタンスを検索できます。

`state` カラム

は、`CS_ABORTING`、`CS_COMMITTING`、`CS_COMMIT_SENT`、`CS_COMPLETE_SENT`、`CS_COMPLETING`、`CS_CONNECTED` のいずれかの値を持つ可能性があります。(`ndbinfo_show_hidden` を有効にして MySQL サーバーが実行されている場合は、通常は非表示になっている `ndb$dbtc_apiconnect_state` テーブルから選択することで、この状態のリストを表示できます。)

`client_node_id` および `client_block_ref` では、`client` は NDB Cluster API または SQL ノード (つまり、NDB API クライアントまたはクラスタに接続された MySQL Server) を指します。

`tc_block_instance` カラムには、`DBTC` ブロックインスタンス番号が表示されます。これをブロック名とともに使用して、`threadblocks` テーブルから特定のスレッドに関する情報を取得できます。

23.5.14.8 ndbinfo config_nodes テーブル

`config_nodes` テーブルには、NDB Cluster `config.ini` ファイルで構成されたノードが表示されます。ノードごとに、ノード ID、ノードのタイプ (管理ノード、データノード、または API ノード)、およびノードが実行されるように構成されているホストの名前または IP アドレスを含む行がテーブルに表示されます。

このテーブルには、特定のノードが実際に実行されているかどうか、または現在クラスタに接続されているかどうかは表示されません。NDB Cluster に接続されているノードに関する情報は、`nodes` および `processes` テーブルから取得できます。

`config_nodes` テーブルには、次のカラムがあります:

- `node_id`
ノード ID
- `node_type`
ノードのタイプ
- `node_hostname`
ノードが存在するホストの名前または IP アドレス

メモ

`node_id` カラムには、このノードの `config.ini` ファイルで使用されるノード ID が表示されます。何も指定されていない場合は、このノードに自動的に割り当てられるノード ID が表示されます。

`node_type` カラムには、次のいずれかの値が表示されます:

- `MGM`: 管理ノード。
- `NDB`: データノード。
- `API`: API ノード。これには SQL ノードが含まれます。

`node_hostname` カラムには、`config.ini` ファイルで指定されているノードホストが表示されます。クラスタ構成ファイルで `HostName` が設定されていない場合、API ノードでは空にできます。構成ファイルのデータノードに `HostName` が設定されていない場合は、ここで `localhost` が使用されます。`localhost` は、`HostName` が管理ノードに指定されていない場合にも使用されます。

23.5.14.9 ndbinfo config_params テーブル

`config_params` テーブルは、NDB Cluster 構成パラメータの名前と内部 ID 番号、およびその他の情報を提供する静的テーブルです。このテーブルを `config_values` テーブルとともに使用して、ノード構成パラメータに関するリアルタイム情報を取得することもできます。

`config_params` テーブルには、次のカラムがあります:

- `param_number`
パラメータの内部 ID 番号
- `param_name`
パラメータの名前

- [param_description](#)
パラメータの簡単な説明
- [param_type](#)
パラメータのデータ型
- [param_default](#)
パラメータのデフォルト値 (存在する場合)
- [param_min](#)
パラメータの最大値 (存在する場合)
- [param_max](#)
パラメータの最小値 (存在する場合)
- [param_mandatory](#)
パラメータが必要な場合は 1、それ以外の場合は 0
- [param_status](#)
現在未使用

メモ

このテーブルは読取り専用です。

これは静的テーブルですが、サポートされているパラメータはソフトウェアリリース、クラスタハードウェア構成、およびその他の要因によって異なる可能性があるため、NDB Cluster インストールによって内容が異なる場合があります。

23.5.14.10 ndbinfo config_values テーブル

[config_values](#) テーブルには、ノード構成パラメータ値の現在の状態に関する情報が表示されます。テーブルの各行は、特定のノードのパラメータの現在の値に対応します。

[config_values](#) テーブルには、次のカラムがあります:

- [node_id](#)
クラスタ内のノードの ID
- [config_param](#)
パラメータの内部 ID 番号
- [config_value](#)
パラメータの現在の値

メモ

このテーブルの [config_param](#) カラムと [config_params](#) テーブルの [param_number](#) カラムでは、同じパラメータ識別子が使用されます。これらのカラムで 2 つのテーブルを結合することで、必要なノード構成パラメータに関する詳細情報を取得できます。ここに示すクエリーでは、クラスタ内の各データノード上のすべてのパラメータの現在の値が、ノード ID とパラメータ名順に表示されます:

```
SELECT v.node_id AS 'Node Id',  
       p.param_name AS 'Parameter',
```

```
v.config_value AS 'Value'
FROM config_values v
JOIN config_params p
ON v.config_param=p.param_number
WHERE p.param_name NOT LIKE '\_\_%'
ORDER BY v.node_id, p.param_name;
```

単純なテストに使用される小規模なサンプルクラスタで実行した場合の、前のクエリーからの出力の一部:

```
+-----+-----+-----+
| Node Id | Parameter                | Value |
+-----+-----+-----+
| 2 | Arbitration                | 1     |
| 2 | ArbitrationTimeout         | 7500  |
| 2 | BackupDataBufferSize       | 16777216 |
| 2 | BackupDataDir              | /home/jon/data |
| 2 | BackupDiskWriteSpeedPct    | 50    |
| 2 | BackupLogBufferSize        | 16777216 |
...
| 3 | TotalSendBufferMemory      | 0     |
| 3 | TransactionBufferMemory    | 1048576 |
| 3 | TransactionDeadlockDetectionTimeout | 1200 |
| 3 | TransactionInactiveTimeout | 4294967039 |
| 3 | TwoPassInitialNodeRestartCopy | 0 |
| 3 | UndoDataBuffer             | 16777216 |
| 3 | UndoIndexBuffer            | 2097152 |
+-----+-----+-----+
248 rows in set (0.02 sec)
```

WHERE 句は、名前が二重アンダースコア (__) で始まるパラメータをフィルタで除外します。これらのパラメータは NDB 開発者によるテストおよびその他の内部使用のために予約されており、本番 NDB Cluster での使用を目的としていません。

適切なクエリーを発行することで、より具体的で詳細な出力、またはその両方を取得できます。この例では、クラスタ内のすべてのデータノードに現在設定されている、**NodeId**、**NoOfReplicas**、**HostName**、**DataMemory**、**IndexMemory** および **TotalSendBufferMemory** パラメータに関する使用可能なすべてのタイプの情報を提供します:

```
SELECT p.param_name AS Name,
       v.node_id AS Node,
       p.param_type AS Type,
       p.param_default AS 'Default',
       p.param_min AS Minimum,
       p.param_max AS Maximum,
       CASE p.param_mandatory WHEN 1 THEN 'Y' ELSE 'N' END AS 'Required',
       v.config_value AS Current
FROM config_params p
JOIN config_values v
ON p.param_number = v.config_param
WHERE p.param_name
IN ('NodeId', 'NoOfReplicas', 'HostName',
    'DataMemory', 'IndexMemory', 'TotalSendBufferMemory')\G
```

単純なテストに使用される 2 つのデータノードを持つ小さい NDB Cluster で実行された場合のこのクエリーからの出力を次に示します (NDB 8.0.18 以降):

```
***** 1. row *****
Name: NodeId
Node: 2
Type: unsigned
Default:
Minimum: 1
Maximum: 144
Required: Y
Current: 2
***** 2. row *****
Name: HostName
Node: 2
Type: string
Default: localhost
Minimum:
```



```
Maximum:
Required: N
Current: 127.0.0.1
***** 3. row *****
  Name: TotalSendBufferMemory
  Node: 2
  Type: unsigned
  Default: 0
  Minimum: 262144
  Maximum: 4294967039
  Required: N
  Current: 0
***** 4. row *****
  Name: NoOfReplicas
  Node: 2
  Type: unsigned
  Default: 2
  Minimum: 1
  Maximum: 4
  Required: N
  Current: 2
***** 5. row *****
  Name: DataMemory
  Node: 2
  Type: unsigned
  Default: 102760448
  Minimum: 1048576
  Maximum: 109951162776
  Required: N
  Current: 524288000
***** 6. row *****
  Name: NodeId
  Node: 3
  Type: unsigned
  Default:
  Minimum: 1
  Maximum: 144
  Required: Y
  Current: 3
***** 7. row *****
  Name: HostName
  Node: 3
  Type: string
  Default: localhost
  Minimum:
  Maximum:
  Required: N
  Current: 127.0.0.1
***** 8. row *****
  Name: TotalSendBufferMemory
  Node: 3
  Type: unsigned
  Default: 0
  Minimum: 262144
  Maximum: 4294967039
  Required: N
  Current: 0
***** 9. row *****
  Name: NoOfReplicas
  Node: 3
  Type: unsigned
  Default: 2
  Minimum: 1
  Maximum: 4
  Required: N
  Current: 2
***** 10. row *****
  Name: DataMemory
  Node: 3
  Type: unsigned
  Default: 102760448
  Minimum: 1048576
  Maximum: 109951162776
  Required: N
  Current: 524288000
```

10 rows in set (0.01 sec)

23.5.14.11 ndbinfo counters テーブル

`counters` テーブルは、特定のカーネルブロックおよびデータノードに対する読み取りや書き込みなどのイベントの現在までの合計数を示します。最近のノードの起動または再起動はカウントされません。ノードを起動または再起動すると、そのノード上のすべてのカウンタがリセットされます。すべてのカーネルブロックですべてのタイプのカウンタを使用しているとはかぎりません。

`counters` テーブルには、次のカラムがあります:

- `node_id`
データノード ID
- `block_name`
関連付けられた NDB カーネルブロックの名前 ([NDB Kernel Blocks](#)を参照してください)。
- `block_instance`
ブロックインスタンス
- `counter_id`
カウンタの内部 ID 番号。通常は 1 から 10 までの整数。
- `counter_name`
カウンタの名前。各カウンタが関連付けられている個々のカウンタおよび NDB カーネルブロックの名前については、テキストを参照してください。
- `val`
カウンタの値

メモ

各カウンタは、特定の NDB カーネルブロックに関連付けられています。

`OPERATIONS` カウンタは、`DBLQH` (ローカルクエリーハンドラ) カーネルブロックに関連付けられます。主キーの読み取りは、主キーの更新と同様に、1 操作としてカウントされます。読み取りの場合、`DBTC` での操作ごとに `DBLQH` での操作が 1 回発生します。書き込みの場合、フラグメントレプリカごとに 1 つの操作がカウントされます。

`ATTRINFO`, `TRANSACTIONS`, `COMMITTS`, `READS`, `LOCAL_READS`, `SIMPLE_READS`, `WRITES`, `LOCAL_WRITES`, `ABORTS`, `TABLE_SCANS` および `RANGE_SCANS` カウンタは、`DBTC` (トランザクション座標) カーネルブロックに関連付けられています。

`LOCAL_WRITES` および `LOCAL_READS` は、レコードのプライマリフラグメントレプリカも保持するノードでトランザクションコーディネータを使用する主キー操作です。

`READS` カウンタには、すべての読み取りが含まれます。`LOCAL_READS` には、このトランザクションコーディネータと同じノード上のプライマリフラグメントレプリカの読み取りのみが含まれます。`SIMPLE_READS` には、読み取り操作が特定のトランザクションの開始および終了操作である読み取りのみが含まれます。単純読み取りではロックは保持されませんが、ロックはトランザクションの一部であり、ロックを含むトランザクションによって行われたコミットされていない変更は監視されますが、コミットされていない他のトランザクションは監視されません。このような読み取りは TC ブロックの観点からの「シンプル」です。ロックは保持されないため、永続的ではなく、`DBTC` が関連する LQH ブロックにルーティングすると、それらの状態は保持されません。

`ATTRINFO` には、解釈済みプログラムがデータノードに送信される回数のカウンタが保持されます。`NDB` カーネルの `ATTRINFO` メッセージについての詳細は、[NDB Protocol Messages](#)を参照してください。

`LOCAL_TABLE_SCANS_SENT`, `READS_RECEIVED`, `PRUNED_RANGE_SCANS_RECEIVED`, `RANGE_SCANS_RECEIVED`, `LOCAL_READS_SENT`, `CONST_PRUNED_RANGE_SCANS_RECEIVED`,

LOCAL_RANGE_SCANS_SENT, REMOTE_READS_SENT, REMOTE_RANGE_SCANS_SENT, READS_NOT_FOUND, SCAN_BATCHES_RETURNED, TABLE_SCANS_RECEIVED および SCAN_ROWS_RETURNED カウンタは、DBSPJ (選択プッシュダウン結合) カーネルブロックに関連付けられています。

block_name および block_instance カラムはそれぞれ、適用可能な NDB カーネルブロック名とインスタンス番号を提供します。これらを使用して、threadblocks テーブルから特定のスレッドに関する情報を取得できます。

このような問題のトラブルシューティングを行う際に、いくつかのカウンタによってトランスポータの過負荷および送信バッファのサイズに関する情報が提供されます。LQH インスタンスごとに、次のリストに示す各カウンタのインスタンスが 1 つ存在します。

- **LQHKEY_OVERLOAD**: トランスポータの過負荷が原因で、LQH ブロックインスタンスで拒否された主キーリクエストの数
- **LQHKEY_OVERLOAD_TC**: TC ノードのトランスポータが過負荷状態になった **LQHKEY_OVERLOAD** のインスタンス数
- **LQHKEY_OVERLOAD_READER**: API リーダー (読み取り専用) ノードが過負荷状態になった **LQHKEY_OVERLOAD** のインスタンス数。
- **LQHKEY_OVERLOAD_NODE_PEER**: 次のバックアップデータノード (書き込み専用) が過負荷状態になった **LQHKEY_OVERLOAD** のインスタンス数
- **LQHKEY_OVERLOAD_SUBSCRIBER**: イベントサブスクライバ (書き込み専用) が過負荷状態になった **LQHKEY_OVERLOAD** のインスタンス数。
- **LQHSCAN_SLOWDOWNS**: スキャン中の API トランスポータの過負荷が原因で、フラグメントスキャンのバッチサイズが減少したインスタンスの数。

23.5.14.12 ndbinfo cpudata テーブル

cpudata テーブルには、最後の秒の CPU 使用率に関するデータが表示されます。

cpustat テーブルには、次のカラムがあります:

- **node_id**
ノード ID
- **cpu_no**
CPU ID
- **cpu_online**
CPU が現在オンラインの場合は 1、それ以外の場合は 0
- **cpu_userspace_time**
ユーザー領域で費やされた CPU 時間
- **cpu_idle_time**
アイドルに費やされた CPU 時間
- **cpu_system_time**
システム時間に要した CPU 時間
- **cpu_interrupt_time**
割り込み (ハードウェアおよびソフトウェア) の処理に費やされた CPU 時間
- **cpu_exec_vm_time**

仮想マシンの実行に要した CPU 時間

メモ

`cpudata` テーブルは、Linux および Solaris オペレーティングシステムでのみ使用できます。

このテーブルは NDB 8.0.23 で追加されました。

23.5.14.13 ndbinfo cpudata_1sec テーブル

`cpudata_1sec` テーブルには、過去 20 秒間の CPU 使用率/秒に関するデータが表示されます。

`cpustat` テーブルには、次のカラムがあります:

- `node_id`
ノード ID
- `measurement_id`
測定シーケンス ID;後の測定には低い ID があります
- `cpu_no`
CPU ID
- `cpu_online`
CPU が現在オンラインの場合は 1、それ以外の場合は 0
- `cpu_userspace_time`
ユーザー領域で費やされた CPU 時間
- `cpu_idle_time`
アイドルに費やされた CPU 時間
- `cpu_system_time`
システム時間に要した CPU 時間
- `cpu_interrupt_time`
割り込み (ハードウェアおよびソフトウェア) の処理に費やされた CPU 時間
- `cpu_exec_vm_time`
仮想マシンの実行に要した CPU 時間
- `elapsed_time`
この測定に使用される時間 (マイクロ秒)

メモ

`cpudata_1sec` テーブルは、Linux および Solaris オペレーティングシステムでのみ使用できます。

このテーブルは NDB 8.0.23 で追加されました。

23.5.14.14 ndbinfo cpudata_20sec テーブル

`cpudata_20sec` テーブルには、過去 400 秒間の 20 秒間隔ごとの CPU 使用率に関するデータが表示されます。

`cpustat` テーブルには、次のカラムがあります:

- `node_id`
ノード ID
- `measurement_id`
測定シーケンス ID;後の測定には低い ID があります
- `cpu_no`
CPU ID
- `cpu_online`
CPU が現在オンラインの場合は 1、それ以外の場合は 0
- `cpu_userspace_time`
ユーザー領域で費やされた CPU 時間
- `cpu_idle_time`
アイドルに費やされた CPU 時間
- `cpu_system_time`
システム時間に要した CPU 時間
- `cpu_interrupt_time`
割り込み (ハードウェアおよびソフトウェア) の処理に費やされた CPU 時間
- `cpu_exec_vm_time`
仮想マシンの実行に要した CPU 時間
- `elapsed_time`
この測定に使用される時間 (マイクロ秒)

メモ

`cpudata_20sec` テーブルは、Linux および Solaris オペレーティングシステムでのみ使用できます。

このテーブルは NDB 8.0.23 で追加されました。

23.5.14.15 ndbinfo cpudata_50ms テーブル

`cpudata_50ms` テーブルには、過去 1 秒間の 50 ミリ秒間隔当たりの CPU 使用率に関するデータが表示されます。

`cpustat` テーブルには、次のカラムがあります:

- `node_id`
ノード ID
- `measurement_id`
測定シーケンス ID;後の測定には低い ID があります
- `cpu_no`
CPU ID

- [cpu_online](#)
CPU が現在オンラインの場合は 1、それ以外の場合は 0
- [cpu_userspace_time](#)
ユーザー領域で費やされた CPU 時間
- [cpu_idle_time](#)
アイドルに費やされた CPU 時間
- [cpu_system_time](#)
システム時間に要した CPU 時間
- [cpu_interrupt_time](#)
割り込み (ハードウェアおよびソフトウェア) の処理に費やされた CPU 時間
- [cpu_exec_vm_time](#)
仮想マシンの実行に要した CPU 時間
- [elapsed_time](#)
この測定に使用される時間 (マイクロ秒)

メモ

[cpudata_50ms](#) テーブルは、Linux および Solaris オペレーティングシステムでのみ使用できます。
このテーブルは NDB 8.0.23 で追加されました。

23.5.14.16 ndbinfo cpuinfo テーブル

[cpuinfo](#) テーブルには、特定のデータノードが実行される CPU に関する情報が表示されます。

[cpuinfo](#) テーブルには、次のカラムがあります:

- [node_id](#)
ノード ID
- [cpu_no](#)
CPU ID
- [cpu_online](#)
CPU がオンラインの場合は 1、それ以外の場合は 0
- [core_id](#)
CPU コア ID
- [socket_id](#)
CPU ソケット ID

メモ

[cpuinfo](#) テーブルは、MacOS および FreeBSD を除き、NDB でサポートされているすべてのオペレーティングシステムで使用できます。

このテーブルは NDB 8.0.23 で追加されました。

23.5.14.17 ndbinfo cpustat テーブル

`cpustat` テーブルには、NDB カーネルで実行されているスレッドごとに、毎秒収集されたスレッドごとの CPU 統計が示されます。

`cpustat` テーブルには、次のカラムがあります:

- `node_id`
スレッドが実行中のノードの ID
- `thr_no`
スレッド ID (このノードに固有)
- `OS_user`
OS ユーザー時間
- `OS_system`
OS システム時間
- `OS_idle`
OS アイドル時間
- `thread_exec`
スレッド実行時間
- `thread_sleeping`
スレッドスリープ時間
- `thread_spinning`
スレッドスピン時間
- `thread_send`
スレッド送信時間
- `thread_buffer_full`
スレッドバッファ一杯時間
- `elapsed_time`
経過時間

23.5.14.18 ndbinfo cpustat_50ms テーブル

`cpustat_50ms` テーブルには、NDB カーネルで実行されているスレッドごとに、50 ミリ秒ごとに取得されたスレッドごとの RAW CPU データが表示されます。

`cpustat_1sec` および `cpustat_20sec` と同様に、このテーブルにはスレッドごとに 20 個の測定セットが表示され、それぞれが指定された期間の期間を参照します。したがって、`cpustat_50ms` では 1 秒分の履歴が提供されます。

`cpustat_50ms` テーブルには、次のカラムがあります:

- `node_id`
スレッドが実行中のノードの ID
- `thr_no`

スレッド ID (このノードに固有)

- [OS_user_time](#)
OS ユーザー時間
- [OS_system_time](#)
OS システム時間
- [OS_idle_time](#)
OS アイドル時間
- [exec_time](#)
スレッド実行時間
- [sleep_time](#)
スレッドスリープ時間
- [spin_time](#)
スレッドスピン時間
- [send_time](#)
スレッド送信時間
- [buffer_full_time](#)
スレッドバッファ一杯時間
- [elapsed_time](#)
経過時間

23.5.14.19 ndbinfo cpustat_1sec テーブル

[cpustat_1sec](#) テーブルには、NDB カーネルで実行されているスレッドごとに取得されるスレッドごとの RAW CPU データが表示されます。

[cpustat_50ms](#) および [cpustat_20sec](#) と同様に、このテーブルにはスレッドごとに 20 個の測定セットが表示され、それぞれが指定された期間の期間を参照します。したがって、[cpustat_1sec](#) には 20 秒の履歴が用意されています。

[cpustat_1sec](#) テーブルには、次のカラムがあります:

- [node_id](#)
スレッドが実行中のノードの ID
- [thr_no](#)
スレッド ID (このノードに固有)
- [OS_user_time](#)
OS ユーザー時間
- [OS_system_time](#)
OS システム時間
- [OS_idle_time](#)

OS アイドル時間

- [exec_time](#)

スレッド実行時間

- [sleep_time](#)

スレッドスリープ時間

- [spin_time](#)

スレッドスピン時間

- [send_time](#)

スレッド送信時間

- [buffer_full_time](#)

スレッドバッファ一杯時間

- [elapsed_time](#)

経過時間

23.5.14.20 ndbinfo cpustat_20sec テーブル

[cpustat_20sec](#) テーブルは、NDB カーネルで実行されているスレッドごとに、20 秒ごとに取得されるスレッドごとの RAW CPU データを提供します。

[cpustat_50ms](#) および [cpustat_1sec](#) と同様に、このテーブルにはスレッドごとに 20 個の測定セットが表示され、それぞれが指定された期間の期間を参照します。したがって、[cpsustat_20sec](#) では 400 秒の履歴が提供されます。

[cpustat_20sec](#) テーブルには、次のカラムがあります:

- [node_id](#)

スレッドが実行中のノードの ID

- [thr_no](#)

スレッド ID (このノードに固有)

- [OS_user_time](#)

OS ユーザー時間

- [OS_system_time](#)

OS システム時間

- [OS_idle_time](#)

OS アイドル時間

- [exec_time](#)

スレッド実行時間

- [sleep_time](#)

スレッドスリープ時間

- [spin_time](#)

スレッドスピン時間

- [send_time](#)

スレッド送信時間

- [buffer_full_time](#)

スレッドバッファ一杯時間

- [elapsed_time](#)

経過時間

23.5.14.21 ndbinfo dict_obj_info テーブル

[dict_obj_info](#) テーブルは、テーブルやインデックスなどの NDB データディクショナリ (DICT) オブジェクトに関する情報を提供します。([dict_obj_types](#) テーブルでは、すべてのタイプのリストをクエリーすることができます。) この情報には、オブジェクトタイプ、状態、親オブジェクト (存在する場合) および完全修飾名が含まれます。

[dict_obj_info](#) テーブルには、次のカラムがあります:

- [type](#)

DICT オブジェクトのタイプ。 [dict_obj_types](#) で結合して名前を取得

- [id](#)

オブジェクト識別子。ディスクデータ undo ログファイルおよびデータファイルの場合、これは [INFORMATION_SCHEMA.FILES](#) テーブルの [LOGFILE_GROUP_NUMBER](#) カラムに表示される値と同じです。undo ログファイルの場合、 [ndbinfo logbuffers](#) および [logspaces](#) テーブルの [log_id](#) カラムに表示される値と同じです

- [version](#)

オブジェクトバージョン

- [state](#)

オブジェクトの状態

- [parent_obj_type](#)

親オブジェクトタイプ ([dict_obj_types](#) タイプ ID)。0 はオブジェクトに親がないことを示します

- [parent_obj_id](#)

親オブジェクト ID (実テーブルなど)。0 は、オブジェクトに親がないことを示します

- [fq_name](#)

完全修飾オブジェクト名。テーブルの場合は [database_name/def/table_name](#) 形式、主キーの場合は [sys/def/table_id/PRIMARY](#) 形式、一意キーの場合は [sys/def/table_id/uk_name\\$unique](#) 形式です

23.5.14.22 ndbinfo dict_obj_tree テーブル

[dict_obj_tree](#) テーブルは、 [dict_obj_info](#) テーブルからのテーブル情報のツリーベースのビューを提供します。これは主にテストでの使用を目的としていますが、NDB データベースオブジェクトの階層を視覚化する場合に役立ちます。

[dict_obj_tree](#) テーブルには、次のカラムがあります:

- [type](#)

DICT オブジェクトのタイプ。オブジェクトタイプの名前を取得するには、 [dict_obj_types](#) で結合

- `id`
オブジェクト識別子 (`dict_obj_info` の `id` カラムと同じ)
ディスクデータの Undo ログファイルおよびデータファイルの場合、これは `INFORMATION_SCHEMA.FILES` テーブルの `LOGFILE_GROUP_NUMBER` カラムに表示される値と同じです。Undo ログファイルの場合、`ndbinfo logbuffers` および `logspaces` テーブルの `log_id` カラムに表示される値とも同じです
- `name`
オブジェクトの完全修飾名 (`dict_obj_info` の `fq_name` カラムと同じ)
テーブルの場合、これは `database_name/def/table_name` (`parent_name` と同じ) です。すべてのタイプのインデックスの場合、これは `NDB$INDEX_index_id_CUSTOM` という形式になります
- `parent_type`
このオブジェクト親オブジェクトの `DICT` オブジェクト型。オブジェクト型の名前を取得するには、`dict_obj_types` で結合
- `parent_id`
このオブジェクトの親オブジェクトの識別子 (`dict_obj_info` テーブルの `id` カラムと同じ)
- `parent_name`
このオブジェクト親オブジェクトの完全修飾名。 `dict_obj_info` テーブルの `fq_name` カラムと同じです
テーブルの場合、これは `database_name/def/table_name` 形式になります。インデックスの場合、名前は `sys/def/table_id/index_name` です。主キーの場合は `sys/def/table_id/PRIMARY`、一意キーの場合は `sys/def/table_id/uk_name$unique` です
- `root_type`
ルートオブジェクトの `DICT` オブジェクト型。オブジェクト型の名前を取得するには、`dict_obj_types` で結合
- `root_id`
ルートオブジェクトの識別子 (`dict_obj_info` テーブルの `id` カラムと同じ)
- `root_name`
ルートオブジェクトの完全修飾名 (`dict_obj_info` テーブルの `fq_name` カラムと同じ)
- `level`
階層内のオブジェクトのレベル
- `パス`
`NDB` オブジェクト階層内のオブジェクトへの完全なパス。オブジェクトは、左のルートオブジェクトから右矢印 (`->`) で区切られます
- `indented_name`
階層内のオブジェクトの深さに対応する、(`->` として表される) 右矢印の前に複数のスペースがある `name`
`path` カラムは、特定の `NDB` データベースオブジェクトへの完全なパスを単一行で取得する場合に便利ですが、`indented_name` カラムを使用すると、目的のオブジェクトの完全な階層情報のツリー形式のレイアウトを取得できます。
例: `test` データベースが存在し、このデータベースに `t1` という名前の既存のテーブルがないと仮定して、次の SQL ステートメントを実行します:

```
CREATE TABLE test.t1 (
```

```
a INT PRIMARY KEY,  
b INT,  
UNIQUE KEY(b)  
) ENGINE = NDB;
```

ここに示すクエリーを使用して、作成したテーブルへのパスを取得できます:

```
mysql> SELECT path FROM ndbinfo.dict_obj_tree  
-> WHERE name LIKE 'test%t1';  
+-----+  
| path |  
+-----+  
| test/def/t1 |  
+-----+  
1 row in set (0.14 sec)
```

次のようなクエリーで、テーブルへのパスをルート名として使用して、このテーブルのすべての依存オブジェクトへのパスを表示できます:

```
mysql> SELECT path FROM ndbinfo.dict_obj_tree  
-> WHERE root_name = 'test/def/t1';  
+-----+  
| path |  
+-----+  
| test/def/t1 |  
| test/def/t1 -> sys/def/13/b |  
| test/def/t1 -> sys/def/13/b -> NDB$INDEX_15_CUSTOM |  
| test/def/t1 -> sys/def/13/b$unique |  
| test/def/t1 -> sys/def/13/b$unique -> NDB$INDEX_16_UI |  
| test/def/t1 -> sys/def/13/PRIMARY |  
| test/def/t1 -> sys/def/13/PRIMARY -> NDB$INDEX_14_CUSTOM |  
+-----+  
7 rows in set (0.16 sec)
```

すべての依存オブジェクトを含む `t1` テーブルの階層ビューを取得するには、次のようなクエリーを実行します。このクエリーでは、ルートオブジェクトの名前として `test/def/t1` を持つ各オブジェクトのインデント名が選択されます:

```
mysql> SELECT indented_name FROM ndbinfo.dict_obj_tree  
-> WHERE root_name = 'test/def/t1';  
+-----+  
| indented_name |  
+-----+  
| test/def/t1 |  
| -> sys/def/13/b |  
| -> NDB$INDEX_15_CUSTOM |  
| -> sys/def/13/b$unique |  
| -> NDB$INDEX_16_UI |  
| -> sys/def/13/PRIMARY |  
| -> NDB$INDEX_14_CUSTOM |  
+-----+  
7 rows in set (0.15 sec)
```

「ディスクデータ」テーブルを操作する場合、このコンテキストではテーブルスペースまたはログファイルグループがルートオブジェクトとみなされることに注意してください。つまり、特定のテーブルに関連付けられているテーブルスペースまたはログファイルグループの名前を知っているか、[SHOW CREATE TABLE](#) からこの情報を取得してから [INFORMATION_SCHEMA.FILES](#) をクエリーするか、次に示すような方法が必要です:

```
mysql> SHOW CREATE TABLE test.dt_1\G  
***** 1. row *****  
Table: dt_1  
Create Table: CREATE TABLE `dt_1` (  
  `member_id` int unsigned NOT NULL AUTO_INCREMENT,  
  `last_name` varchar(50) NOT NULL,  
  `first_name` varchar(50) NOT NULL,  
  `dob` date NOT NULL,  
  `joined` date NOT NULL,  
  PRIMARY KEY (`member_id`),  
  KEY `last_name` (`last_name`,`first_name`)  
) /*!50100 TABLESPACE `ts_1` STORAGE DISK */ ENGINE=ndbcluster DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci  
1 row in set (0.00 sec)  
  
mysql> SELECT DISTINCT TABLESPACE_NAME, LOGFILE_GROUP_NAME
```



```
-> FROM INFORMATION_SCHEMA.FILES WHERE TABLESPACE_NAME='ts_1';
+-----+-----+
| TABLESPACE_NAME | LOGFILE_GROUP_NAME |
+-----+-----+
| ts_1             | lg_1                |
+-----+-----+
1 row in set (0.00 sec)
```

これで、次のようにテーブル、テーブルスペースおよびログファイルグループの階層情報を取得できます:

```
mysql> SELECT indented_name FROM ndbinfo.dict_obj_tree
-> WHERE root_name = 'test/def/dt_1';
+-----+
| indented_name |
+-----+
| test/def/dt_1 |
| -> sys/def/23/last_name |
| -> NDB$INDEX_25_CUSTOM |
| -> sys/def/23/PRIMARY |
| -> NDB$INDEX_24_CUSTOM |
+-----+
5 rows in set (0.15 sec)

mysql> SELECT indented_name FROM ndbinfo.dict_obj_tree
-> WHERE root_name = 'ts_1';
+-----+
| indented_name |
+-----+
| ts_1          |
| -> data_1.dat |
| -> data_2.dat |
+-----+
3 rows in set (0.17 sec)

mysql> SELECT indented_name FROM ndbinfo.dict_obj_tree
-> WHERE root_name LIKE 'lg_1';
+-----+
| indented_name |
+-----+
| lg_1          |
| -> undo_1.log |
| -> undo_2.log |
+-----+
3 rows in set (0.16 sec)
```

NDB 8.0.24 に [dict_obj_tree](#) テーブルが追加されました。

23.5.14.23 ndbinfo dict_obj_types テーブル

[dict_obj_types](#) テーブルは、NDB カーネルで使用される可能性のあるディクショナリオブジェクトのタイプを一覧表示した静的なテーブルです。これらは、NDB API の [Object::Type](#) で定義されるタイプと同じです。

[dict_obj_types](#) テーブルには、次のカラムがあります:

- [type_id](#)
このタイプのタイプ ID
- [type_name](#)
このタイプの名前

23.5.14.24 ndbinfo disk_write_speed_base テーブル

[disk_write_speed_base](#) テーブルは、LCP、バックアップ、およびリストア操作時のディスク書き込みの速度に関する基本情報を示します。

[disk_write_speed_base](#) テーブルには、次のカラムがあります:

- [node_id](#)

このノードのノード ID

- [thr_no](#)

この LDM スレッドのスレッド ID

- [millis_ago](#)

このレポート期間が終了してからのミリ秒数

- [millis_passed](#)

このレポート期間内に経過したミリ秒数

- [backup_lcp_bytes_written](#)

この期間中にローカルチェックポイントおよびバックアッププロセスによってディスクに書き込まれたバイト数

- [redo_bytes_written](#)

この期間中に Redo ログに書き込まれたバイト数

- [target_disk_write_speed](#)

LDM スレッド (基本データ) ごとの実際のディスク書き込み速度

23.5.14.25 ndbinfo disk_write_speed_aggregate テーブル

[disk_write_speed_aggregate](#) テーブルは、LCP、バックアップ、およびリストア操作時のディスク書き込みの速度に関して集計された情報を示します。

[disk_write_speed_aggregate](#) テーブルには、次のカラムがあります:

- [node_id](#)

このノードのノード ID

- [thr_no](#)

この LDM スレッドのスレッド ID

- [backup_lcp_speed_last_sec](#)

過去 1 秒間にバックアップおよび LCP プロセスによってディスクに書き込まれたバイト数

- [redo_speed_last_sec](#)

過去 1 秒間に Redo ログに書き込まれたバイト数

- [backup_lcp_speed_last_10sec](#)

バックアップおよび LCP プロセスによってディスクに書き込まれた 1 秒当たりのバイト数 (過去 10 秒間の平均値)

- [redo_speed_last_10sec](#)

Redo ログにディスクに書き込まれた 1 秒当たりのバイト数 (過去 10 秒間の平均値)

- [std_dev_backup_lcp_speed_last_10sec](#)

バックアップおよび LCP プロセスによってディスクに書き込まれた 1 秒当たりのバイト数の標準偏差 (過去 10 秒間の平均値)

- [std_dev_redo_speed_last_10sec](#)

Redo ログにディスクに書き込まれた 1 秒当たりのバイト数の標準偏差 (過去 10 秒間の平均値)

- [backup_lcp_speed_last_60sec](#)
バックアップおよび LCP プロセスによってディスクに書き込まれた 1 秒当たりのバイト数 (過去 60 秒間の平均値)
- [redo_speed_last_60sec](#)
Redo ログにディスクに書き込まれた 1 秒当たりのバイト数 (過去 10 秒間の平均値)
- [std_dev_backup_lcp_speed_last_60sec](#)
バックアップおよび LCP プロセスによってディスクに書き込まれた 1 秒当たりのバイト数の標準偏差 (過去 60 秒間の平均値)
- [std_dev_redo_speed_last_60sec](#)
Redo ログにディスクに書き込まれた 1 秒当たりのバイト数の標準偏差 (過去 60 秒間の平均値)
- [slowdowns_due_to_io_lag](#)
REDO ログの I/O ラグが原因でディスク書き込みが遅くなった最後のノード起動以降の秒数
- [slowdowns_due_to_high_cpu](#)
最後のノード起動以降、CPU 使用率が高いためにディスク書き込みが遅くなった秒数
- [disk_write_speed_set_to_min](#)
最後のノードが起動してからディスク書き込み速度が最小に設定された秒数
- [current_target_disk_write_speed](#)
LDM スレッドごとの実際のディスク書き込み速度 (集計値)

23.5.14.26 ndbinfo disk_write_speed_aggregate_node テーブル

[disk_write_speed_aggregate_node](#) テーブルは、LCP、バックアップ、およびリストア操作時のディスク書き込みの速度に関して集計された情報を示します。

[disk_write_speed_aggregate_node](#) テーブルには、次のカラムがあります:

- [node_id](#)
このノードのノード ID
- [backup_lcp_speed_last_sec](#)
過去 1 秒間にバックアップおよび LCP プロセスによってディスクに書き込まれたバイト数
- [redo_speed_last_sec](#)
過去 1 秒間に Redo ログに書き込まれたバイト数
- [backup_lcp_speed_last_10sec](#)
バックアップおよび LCP プロセスによってディスクに書き込まれた 1 秒当たりのバイト数 (過去 10 秒間の平均値)
- [redo_speed_last_10sec](#)
Redo ログにディスクに書き込まれた 1 秒当たりのバイト数 (過去 10 秒間の平均値)
- [backup_lcp_speed_last_60sec](#)
バックアップおよび LCP プロセスによってディスクに書き込まれた 1 秒当たりのバイト数 (過去 60 秒間の平均値)
- [redo_speed_last_60sec](#)

バックアップおよび LCP プロセスによってディスクに書き込まれた 1 秒当たりのバイト数 (過去 60 秒間の平均値)

23.5.14.27 ndbinfo diskpagebuffer テーブル

`diskpagebuffer` テーブルには、「NDB Cluster ディスクデータ」テーブルによるディスクページバッファの使用状況に関する統計が表示されます。

`diskpagebuffer` テーブルには、次のカラムがあります:

- `node_id`
データノード ID
- `block_instance`
ブロックインスタンス
- `pages_written`
ディスクに書き込まれたページ数。
- `pages_written_lcp`
ローカルチェックポイントによって書き込まれたページ数。
- `pages_read`
ディスクから読み取られたページ数
- `log_waits`
ログがディスクに書き込まれるまで待機しているページ書き込みの数
- `page_requests_direct_return`
バッファ内で使用可能だったページに対するリクエストの数
- `page_requests_wait_queue`
ページがバッファ内で使用可能になるまで待機する必要があったリクエストの数
- `page_requests_wait_io`
ディスク上のページから読み取る必要があった (バッファ内のページを使用できなかった) リクエストの数

メモ

このテーブルを「NDB Cluster ディスクデータ」テーブルとともに使用すると、`DiskPageBufferMemory` がディスクからではなくバッファからデータを読み取るのに十分な大きさであるかどうかを判断できます。ディスクシークを最小限に抑えると、このようなテーブルのパフォーマンスを向上させることができます。

このようなクエリーを使用すると、読み取りの合計回数に対する `DiskPageBufferMemory` からの読み取りの比率を特定できます。この比率は、パーセンテージとして取得されます。

```
SELECT
  node_id,
  100 * page_requests_direct_return /
  (page_requests_direct_return + page_requests_wait_io)
  AS hit_ratio
FROM ndbinfo.diskpagebuffer;
```

このクエリーによる結果は、クラスタ内のデータノード (この例では、クラスタには 4 つのデータノードがあります) ごとに 1 行で、次に示すものと同様になります。

```
+-----+-----+
| node_id | hit_ratio |
```

```
+-----+-----+
| 5 | 97.6744 |
| 6 | 97.6879 |
| 7 | 98.1776 |
| 8 | 98.1343 |
+-----+-----+
4 rows in set (0.00 sec)
```

`hit_ratio` の値が 100% に近づいていることは、バッファからではなく、ディスクから行われている読み取りの数が非常にわずかしかかないことを示します。つまり、ディスクデータの読み取りパフォーマンスが最適なレベルに近づいています。これらの値のいずれかが 95% 未満である場合は、`config.ini` ファイルで `DiskPageBufferMemory` の設定を大きくする必要があることを強く示しています。

注記

`DiskPageBufferMemory` の変更が有効になる前に、クラスタのすべてのデータノードのローリング再起動が必要です。

`block_instance` は、カーネルブロックのインスタンスを指します。この番号は、ブロック名とともに使用して、`threadblocks` テーブル内の特定のインスタンスを検索できます。この情報を使用すると、個々のスレッドに関連するディスクページバッファメトリックに関する情報を取得できます。`LIMIT 1` を使用して出力を単一スレッドに制限するクエリーの例を次に示します:

```
mysql> SELECT
>   node_id, thr_no, block_name, thread_name, pages_written,
>   pages_written_lcp, pages_read, log_waits,
>   page_requests_direct_return, page_requests_wait_queue,
>   page_requests_wait_io
> FROM ndbinfo.diskpagebuffer
> INNER JOIN ndbinfo.threadblocks USING (node_id, block_instance)
> INNER JOIN ndbinfo.threads USING (node_id, thr_no)
> WHERE block_name = 'PGMAN' LIMIT 1G
***** 1. row *****
      node_id: 1
      thr_no: 1
      block_name: PGMAN
      thread_name: rep
      pages_written: 0
      pages_written_lcp: 0
      pages_read: 1
      log_waits: 0
page_requests_direct_return: 4
page_requests_wait_queue: 0
page_requests_wait_io: 1
1 row in set (0.01 sec)
```

23.5.14.28 ndbinfo diskstat テーブル

`diskstat` テーブルには、過去 1 秒間の「ディスクデータ」テーブルスペースへの書き込みに関する情報が表示されます。

`diskstat` テーブルには、次のカラムがあります:

- `node_id`
このノードのノード ID
- `block_instance`
PGMAN のレポートインスタンスの ID
- `pages_made_dirty`
過去 1 秒間にダーティになったページ数
- `reads_issued`
過去 1 秒間に発行された読み取り

- `reads_completed`
過去 1 秒間に完了した読取り
- `writes_issued`
過去 1 秒間に発行された書き込み
- `writes_completed`
過去 1 秒間に完了した書き込み
- `log_writes_issued`
過去 1 秒間にページ書き込みにログ書き込みが必要だった回数
- `log_writes_completed`
最後の秒の間に完了したログ書き込み数
- `get_page_calls_issued`
過去 1 秒間に発行された `get_page()` コールの数
- `get_page_reqs_issued`
`get_page()` コールで I/O の待機または過去 1 秒間にすでに開始された I/O の完了が発生した回数
- `get_page_reqs_completed`
過去 1 秒間に完了した I/O または I/O の完了を待機している `get_page()` コールの数

メモ

このテーブルの各行は、`PGMAN` のインスタンスに対応しています。LDM スレッドごとに 1 つのインスタンスに加えて、データノードごとに追加のインスタンスがあります。

NDB 8.0.19 に `diskstat` テーブルが追加されました。

23.5.14.29 ndbinfo diskstats_1sec テーブル

`diskstats_1sec` テーブルには、過去 20 秒間の「ディスクデータ」テーブルスペースへの書き込みに関する情報が表示されます。

`diskstat` テーブルには、次のカラムがあります:

- `node_id`
このノードのノード ID
- `block_instance`
`PGMAN` のレポートインスタンスの ID
- `pages_made_dirty`
指定された 1 秒間隔でダーティになったページ
- `reads_issued`
指定された 1 秒間隔中に発行された読取り
- `reads_completed`
指定された 1 秒間隔で読取りが完了しました

- [writes_issued](#)
指定された 1 秒間隔中に発行された書込み
- [writes_completed](#)
指定された 1 秒間隔で書込みが完了しました
- [log_writes_issued](#)
指定された 1 秒間隔の間にページ書込みでログ書込みが必要になった回数
- [log_writes_completed](#)
指定された 1 秒間隔中に完了したログ書込み数
- [get_page_calls_issued](#)
指定された 1 秒間隔中に発行された [get_page\(\)](#) コールの数
- [get_page_reqs_issued](#)
指定された 1 秒間隔中に、[get_page\(\)](#) コールが I/O の待機または I/O の完了をすでに開始した回数
- [get_page_reqs_completed](#)
指定された 1 秒間隔中に完了した I/O または I/O の完了を待機している [get_page\(\)](#) コールの数
- [seconds_ago](#)
この行が適用される間隔の過去の 1 秒間隔の数

メモ

このテーブルの各行は、0 から 19 秒前に発生する 1 秒間隔の間の [PGMAN](#) のインスタンスに対応します。LDM スレッドごとにそのようなインスタンスが 1 つあり、データノードごとに追加のインスタンスがあります。

NDB 8.0.19 に [diskstats_1sec](#) テーブルが追加されました。

23.5.14.30 ndbinfo error_messages テーブル

[error_messages](#) テーブルには、次の情報が表示されます

[error_messages](#) テーブルには、次のカラムがあります:

- [error_code](#)
数値エラーコード
- [error_description](#)
エラーの説明
- [error_status](#)
Error status code
- [error_classification](#)
エラー分類コード

メモ

[error_code](#) は数値の NDB エラーコードです。これは、[ndb_perror](#) (NDB 8.0.13 より前の場合は [pererror --ndb](#)) に提供できるエラーコードと同じです。

`error_description` では、エラーの原因となった条件の基本的な説明が提供されます。

`error_status` カラムには、エラーに関連するステータス情報が表示されます。このカラムに使用可能な値は次のとおりです:

- No error
- Illegal connect string
- Illegal server handle
- Illegal reply from server
- Illegal number of nodes
- Illegal node status
- メモリー不足
- Management server not connected
- Could not connect to socket
- Start failed
- Stop failed
- Restart failed
- Could not start backup
- Could not abort backup
- Could not enter single user mode
- Could not exit single user mode
- Failed to complete configuration change
- Failed to get configuration
- Usage error
- 成功
- Permanent error
- Temporary error
- Unknown result
- Temporary error, restart node
- Permanent error, external action needed
- Ndbd file system error, restart node initial
- Unknown

`error_classification` カラムには、エラーの分類が表示されます。分類コードとその意味の詳細は、[NDB Error Classifications](#) を参照してください。

23.5.14.31 ndbinfo hwinfo テーブル

`hwinfo` テーブルには、特定のデータノードが実行されるハードウェアに関する情報が表示されます。

hwinfo テーブルには、次のカラムがあります:

- [node_id](#)
ノード ID
- [cpu_cnt_max](#)
このホスト上のプロセッサの数
- [cpu_cnt](#)
このノードで使用可能なプロセッサの数
- [num_cpu_cores](#)
このホスト上の CPU コアの数
- [num_cpu_sockets](#)
このホスト上の CPU ソケットの数
- [HW_memory_size](#)
このホストで使用可能なメモリー量
- [model_name](#)
CPU モデル名

メモ

hwinfo テーブルは、NDB でサポートされているすべてのオペレーティングシステムで使用できます。

このテーブルは NDB 8.0.23 で追加されました。

23.5.14.32 ndbinfo locks_per_fragment テーブル

locks_per_fragment テーブルには、operations_per_fragment および memory_per_fragment へのコンパニオンテーブルとして機能する、ロック要求リクエストの数およびこれらのリクエストの結果に関する情報がフラグメント単位で表示されます。このテーブルには、フラグメントまたはテーブルの作成以降、または最後の再起動以降にロックの待機に成功した時間と失敗した時間の合計も表示されます。

locks_per_fragment テーブルには、次のカラムがあります:

- [fq_name](#)
完全修飾テーブル名
- [parent_fq_name](#)
親オブジェクトの完全修飾名
- [type](#)
テーブルタイプ。使用可能な値についてはテキストを参照してください
- [table_id](#)
テーブル ID
- [node_id](#)
レポートノード ID

- [block_instance](#)
LDM インスタンス ID
- [fragment_num](#)
フラグメント識別子
- [ex_req](#)
排他ロックリクエストが開始されました
- [ex_imm_ok](#)
排他ロック要求がすぐに付与されました
- [ex_wait_ok](#)
待機後に付与された排他ロックリクエスト
- [ex_wait_fail](#)
排他ロックリクエストが付与されていません
- [sh_req](#)
共有ロック要求が開始されました
- [sh_imm_ok](#)
共有ロックリクエストがすぐに付与されました
- [sh_wait_ok](#)
次の待機後に許可される共有ロックリクエスト
- [sh_wait_fail](#)
共有ロックリクエストが付与されていません
- [wait_ok_millis](#)
付与されたロックリクエストの待機にかかった時間 (ミリ秒)
- [wait_fail_millis](#)
失敗したロックリクエストの待機時間 (ミリ秒)

メモ

[block_instance](#) は、カーネルブロックのインスタンスを指します。この番号は、ブロック名とともに使用して、[threadblocks](#) テーブル内の特定のインスタンスを検索できます。

[fq_name](#) は、[test/def/t1](#)、[sys/def/10/b\\$unique](#) など、[database / schema / name](#) 形式の完全修飾データベースオブジェクト名です。

[parent_fq_name](#) は、このオブジェクト親オブジェクト (テーブル) の完全修飾名です。

[table_id](#) は、NDB によって生成されるテーブルの内部 ID です。これは、他の [ndbinfo](#) テーブルに表示されるのと同じ内部テーブル ID で、[ndb_show_tables](#) の出力にも表示されます。

[type](#) カラムには、テーブルのタイプが表示されます。これは常に [System table](#)、[User table](#)、[Unique hash index](#)、[Hash index](#)、[Unique ordered index](#)、[Ordered index](#)、[Hash index trigger](#)、[Subscription trigger](#)、[Read only constraint](#)、[Index trigger](#)、[Reorganize trigger](#)、[Tablespace](#)、[Log file group](#)、[Data file](#)、[Undo file](#)、[Hash map](#)、[Foreign key definition](#)、[Foreign key parent trigger](#)、[Foreign key child trigger](#) または [Schema transaction](#) のいずれかです。

`ex_req`, `ex_req_imm_ok`, `ex_wait_ok`, `ex_wait_fail`, `sh_req`, `sh_req_imm_ok`, `sh_wait_ok` および `sh_wait_fail` のすべてのカラムに表示される値は、テーブルまたはフラグメントが作成されてから、またはこのノードが最後に再起動されてからのいずれか後で発生したリクエストの累積数をテーブルします。これは、`wait_ok_millis` カラムおよび `wait_fail_millis` カラムに表示される時間値にも当てはまります。

すべてのロックリクエストは、進行中であるか、なんらかの方法 (成功または失敗) で完了したとみなされます。これは、次の関係が真であることを意味します:

```
ex_req >= (ex_req_imm_ok + ex_wait_ok + ex_wait_fail)
sh_req >= (sh_req_imm_ok + sh_wait_ok + sh_wait_fail)
```

現在進行中のリクエストの数は、次に示すように、不完全なリクエストの現在の数です:

```
[exclusive lock requests in progress] =
  ex_req - (ex_req_imm_ok + ex_wait_ok + ex_wait_fail)

[shared lock requests in progress] =
  sh_req - (sh_req_imm_ok + sh_wait_ok + sh_wait_fail)
```

失敗した待機は中断されたトランザクションを示しますが、中断はロック待機タイムアウトによって発生する場合と発生しない場合があります。次に示すように、ロック待機中の中断の合計数を取得できます:

```
[aborts while waiting for locks] = ex_wait_fail + sh_wait_fail
```

23.5.14.33 ndbinfo logbuffers テーブル

`logbuffer` テーブルは NDB Cluster ログバッファの使用状況に関する情報を提供します。

`logbuffers` テーブルには、次のカラムがあります:

- `node_id`
このデータノードの ID。
- `log_type`
ログのタイプ。次のいずれか: `REDO`, `DD-UNDO`, `BACKUP-DATA` または `BACKUP-LOG`。
- `log_id`
ログ ID。ディスクデータ undo ログファイルの場合、これは `INFORMATION_SCHEMA.FILES` テーブルの `LOGFILE_GROUP_NUMBER` カラムに表示される値および `ndbinfo logspaces` テーブルの `log_id` カラムに表示される値と同じです
- `log_part`
ログパート番号
- `total`
このログに使用可能な合計容量
- `used`
このログによって使用された容量

メモ

NDB バックアップの実行時には、2 つの追加のログタイプを反映する `logbuffers` テーブル行を使用できます。これらのいずれかの行のログタイプは `BACKUP-DATA` で、バックアップ中にフラグメントをバックアップファイルにコピーするために使用されるデータバッファの量が表示されます。もう一方の行のログタイプは `BACKUP-LOG` で、バックアップの開始後に行われた変更を記録するためにバックアップ中に使用されたログバッファの量が表示されます。これらの `log_type` 行のそれぞれが、クラスタ内の各データノードの `logbuffers` テーブルに表示されます。NDB バックアップが現在実行されていないかぎり、これらの行は存在しません。

23.5.14.34 ndbinfo logspaces テーブル

このテーブルは、NDB Cluster のログ領域使用量に関する情報を提供します。

`logspaces` テーブルには、次のカラムがあります:

- `node_id`
このデータノードの ID。
- `log_type`
ログのタイプ。REDO または DD-UNDO のいずれか。
- `log_id`
ログ ID。ディスクデータ undo ログファイルの場合、これは `INFORMATION_SCHEMA.FILES` テーブルの `LOGFILE_GROUP_NUMBER` カラムに表示される値および `ndbinfo logbuffers` テーブルの `log_id` カラムに表示される値と同じです
- `log_part`
ログパート番号。
- `total`
このログに使用可能な合計容量。
- `used`
このログによって使用された容量。

23.5.14.35 ndbinfo membership テーブル

`membership` テーブルは、ノードグループのメンバーシップ、プレジデントノード、アービトレータ、アービトレータの後継ノード、アービトレータの接続状態、およびその他の情報を含む、各データノードがクラスタ内のほかのすべてのノードについて保持するビューを示します。

`membership` テーブルには、次のカラムがあります:

- `node_id`
このノードのノード ID
- `group_id`
このノードが属するノードグループ
- `left node`
前のノードのノード ID
- `right_node`
次のノードのノード ID
- `president`
プレジデントのノード ID
- `successor`
プレジデントの後継ノードのノード ID
- `succession_order`

このノードがプレジデントの任務を継承する順序

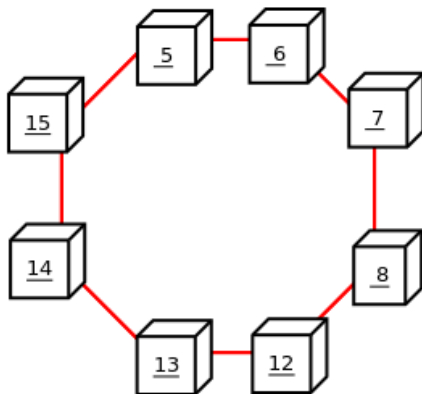
- `Conf_HB_order`
-
- `arbiterator`
アービトレータのノード ID
- `arb_ticket`
アービトレーションを追跡するために使用される内部識別子
- `arb_state`
アービトレーションの状態
- `arb_connected`
このノードがアービトレータに接続されているかどうか (Yes または No のいずれか)
- `connected_rank1_arbs`
接続されたランク 1 のアービトレータ
- `connected_rank2_arbs`
接続されたランク 1 のアービトレータ

メモ

ノード ID およびノードグループ ID は、`ndb_mgm -e "SHOW"` でレポートされるものと同じです。

`left_node` および `right_node` は、次に示すように、時計文字板上の数字の順序と同様に、すべてのデータノードを、それらのノード ID の順序で円形に接続するモデルに関して定義されます。

図 23.27 NDB Cluster ノードの循環配置



この例では、円形に右回りの順序で、5、6、7、8、12、13、14、15の番号が付けられた8つのデータノードがあります。「左」と「右」は、円の内側から判断します。ノード5の左側にあるノードはノード15であり、ノード5の右側にあるノードはノード6です。次のクエリーを実行し、その出力を調査することで、これらの関係をすべて確認できます。

```
mysql> SELECT node_id,left_node,right_node
-> FROM ndbinfo.membership;
+-----+-----+-----+
| node_id | left_node | right_node |
```

```
+-----+-----+-----+
| 5 | 15 | 6 |
| 6 | 5 | 7 |
| 7 | 6 | 8 |
| 8 | 7 | 12 |
| 12 | 8 | 13 |
| 13 | 12 | 14 |
| 14 | 13 | 15 |
| 15 | 14 | 5 |
+-----+-----+-----+
8 rows in set (0.00 sec)
```

「左」と「右」の指定は、イベントログでも同様に使用されます。

`president` ノードは、現在のノードから、アービトレータを設定する責任を持つものとして見られるノードです ([NDB Cluster Start Phases](#) を参照してください)。プレジデントで障害が発生したり、切断されたりすると、現在のノードは、`successor` カラムに ID が示されたノードが新しいプレジデントになることを期待します。`succession_order` カラムは、現在のノードが自分自身で持っているのみならず後継キュー内の場所を示します。

通常の NDB Cluster では、すべてのデータノードに `president` と同じノードが表示され、その `successor` と同じノード (社長以外) が表示されます。さらに、現在のプレジデントは、自分自身を後継順序の 1 とみなし、`successor` ノードは、自分自身を 2 とみなす、というようにする必要があります。

すべてのノードで同じ `arb_state` の値、および同じ `arb_ticket` の値を示すべきです。可能性のある `arb_state` の値は `ARBIT_NULL`、`ARBIT_INIT`、`ARBIT_FIND`、`ARBIT_PREP1`、`ARBIT_PREP2`、`ARBIT_START`、`ARBIT_RUN`、`ARBIT_CHOOSE`、および `UNKNOWN` です。

`arb_connected` は、このノードの `arbitrator` として示されているノードに、このノードが接続されているかどうかを示します。

`connected_rank1_arbs` および `connected_rank2_arbs` の各カラムには、`ArbitrationRank` がそれぞれ 1 または 2 と等しい 0 個以上のアービトレータのリストが表示されます。

注記

管理ノードと API ノードの両方に、アービトレータになる資格があります。

23.5.14.36 ndbinfo memoryusage テーブル

このテーブルをクエリーすると、`ndb_mgm` クライアントで `ALL REPORT MemoryUsage` コマンドによって提供されるものや、`ALL DUMP 1000` によってログに記録されるものと同様の情報が提供されます。

`memoryusage` テーブルには、次のカラムがあります:

- `node_id`
このデータノードのノード ID。
- `memory_type`
`Data memory`、`Index memory` または `Long message buffer` のいずれか。
- `used`
このデータノードでデータメモリーまたはインデックスメモリーに現在使用されているバイト数。
- `used_pages`
このデータノードでデータメモリーまたはインデックスメモリーに現在使用されているページ数 (テキストを参照してください)。
- `total`
このデータノードに使用可能なデータメモリーまたはインデックスメモリーの合計バイト数 (テキストを参照してください)。

- [total_pages](#)

このデータノード上のデータメモリーまたはインデックスメモリーに使用可能なメモリーページの合計数 (テキストを参照してください)。

メモ

total カラムは、特定のデータノード上の特定のリソース (データメモリーまたはインデックスメモリー) に使用可能なメモリーの合計量をバイト単位で表します。この数値は、`config.ini` ファイル内の対応する構成パラメータの設定にほぼ等しいです。

クラスタにノード ID 5 と 6 を持つ 2 つのデータノードがあり、`config.ini` ファイルに次が含まれると仮定します。

```
[ndbd default]
DataMemory = 1G
IndexMemory = 1G
```

また、`LongMessageBuffer` 構成パラメータの値がデフォルト (64 MB) であるとします。

次のクエリーでは、ほぼ同じ値が表示されます。

```
mysql> SELECT node_id, memory_type, total
> FROM ndbinfo.memoryusage;
+-----+-----+-----+
| node_id | memory_type      | total      |
+-----+-----+-----+
| 5 | Data memory      | 1073741824 |
| 5 | Index memory     | 1074003968 |
| 5 | Long message buffer | 67108864  |
| 6 | Data memory      | 1073741824 |
| 6 | Index memory     | 1074003968 |
| 6 | Long message buffer | 67108864  |
+-----+-----+-----+
6 rows in set (0.00 sec)
```

この場合、インデックスメモリーの **total** カラムの値は、内部の丸め処理によって `IndexMemory` の値セットよりもわずかに大きくなります。

`used_pages` および `total_pages` カラムでは、リソースがページ単位で計測されます。これらのサイズは、`DataMemory` では 32K、`IndexMemory` では 8K です。長いメッセージバッファメモリーの場合、ページサイズは 256 バイトです。

23.5.14.37 ndbinfo memory_per_fragment テーブル

`memory_per_fragment` テーブルには、個々のフラグメントによるメモリーの使用状況に関する情報が表示されます。

`memory_per_fragment` テーブルには、次のカラムがあります:

- [fq_name](#)

このフラグメントの名前

- [parent_fq_name](#)

このフラグメントの親の名前

- [type](#)

オブジェクトのタイプ (可能性のある値についてはテキストを参照してください)

- [table_id](#)

このテーブルのテーブル ID

- [node_id](#)

このノードのノード ID

- [block_instance](#)

カーネルブロックインスタンス ID

- [fragment_num](#)

フラグメント ID (番号)

- [fixed_elem_alloc_bytes](#)

固定サイズの要素に割り当てられたバイト数

- [fixed_elem_free_bytes](#)

固定サイズの要素に割り当てられたページ単位の残りの空きバイト数

- [fixed_elem_size_bytes](#)

固定サイズの各要素のバイト単位の長さ

- [fixed_elem_count](#)

固定サイズの要素の数

- [fixed_elem_free_count](#)

固定サイズの要素用の空き行数

- [var_elem_alloc_bytes](#)

可変サイズの要素に割り当てられたバイト数

- [var_elem_free_bytes](#)

可変サイズの要素に割り当てられたページ単位の残りの空きバイト数

- [var_elem_count](#)

可変サイズの要素の数

- [hash_index_alloc_bytes](#)

ハッシュインデックスに割り当てられたバイト数

メモ

このテーブルの [type](#) カラムは、このフラグメントで使用されるディクショナリオブジェクトのタイプ (NDB API では [Object::Type](#)) を示し、次のリストに示す値のいずれかを取る可能性があります。

- System table
- User table
- Unique hash index
- Hash index
- Unique ordered index
- Ordered index
- Hash index trigger

- Subscription trigger
- Read only constraint
- Index trigger
- Reorganize trigger
- テーブルスペース
- Log file group
- データファイル
- Undo file
- Hash map
- Foreign key definition
- Foreign key parent trigger
- Foreign key child trigger
- Schema transaction

このリストは、`mysql` クライアントで `SELECT * FROM ndbinfo.dict_obj_types` を実行して取得することもできます。

`block_instance` カラムは NDB カーネルブロックのインスタンス番号を提供します。これを使用して、`threadblocks` テーブルから特定のスレッドに関する情報を取得できます。

23.5.14.38 ndbinfo nodes テーブル

このテーブルには、データノードのステータスに関する情報が格納されます。このテーブルの対応する行には、クラスタ内で実行されているデータノードごとに、ノードのノード ID、ステータス、および稼働時間が示されます。起動中のノードごとに、現在の起動フェーズも示されます。

`nodes` テーブルには、次のカラムがあります:

- `node_id`
クラスタ内のデータノードの一意のノード ID。
- `uptime`
ノードが最後に起動されてからの秒単位の時間。
- `status`
データノードの現在のステータス。可能性のある値についてはテキストを参照してください。
- `start_phase`
データノードが起動中の場合、現在の起動フェーズ。
- `config_generation`
このデータノードで使用中のクラスタ構成ファイルのバージョン。

メモ

`uptime` カラムは、このノードが最後に起動または再起動されてから実行している時間を秒単位で示します。これは `BIGINT` 値です。この図には、ノードの起動に実際に必要な時間が含まれています。つまり、このカウンタは、`ndbd` または `ndbmtid` が最初に起動された時点の実行を開始します。したがって、まだ起動が終了していないノードの場合でも、`uptime` にゼロ以外の値が表示されることがあります。

`status` カラムは、ノードの現在のステータスを示します。これは、`NOTHING`、`CMVMI`、`STARTING`、`STARTED`、`SINGLEUSER`、`STOPPING_1`、`STOPPING_2`、`STOPPING_3`、または `STOPPING_4` のいずれかです。ステータスが `STARTING` の場合は、`start_phase` カラムで現在の起動フェーズを確認できます (このセクションの後半を参照してください)。クラスタがシングルユーザーモードになっているときは、すべてのデータノードの `status` カラムに `SINGLEUSER` が表示されます (セクション23.5.6「NDB Cluster のシングルユーザーモード」を参照してください)。いずれかの `STOPPING` 状態が表示されていることは必ずしもノードが停止していることを意味するわけではありませんが、新しい状態になることを意味する場合があります。たとえば、クラスタをシングルユーザーモードにすると、ステータスが `SINGLEUSER` に変更される前に、データノードがその状態を `STOPPING_2` として簡単にレポートする場合があります。

`start_phase` カラムでは、`ndb_mgm` クライアントの `node_id STATUS` コマンドの出力で使用するものと同じ範囲の値が使用されます (セクション23.5.1「NDB Cluster 管理クライアントのコマンド」を参照してください)。ノードが現在起動中でない場合、このカラムは `0` を示します。NDB Cluster の起動フェーズと説明の一覧については、セクション23.5.4「NDB Cluster 起動フェーズのサマリー」を参照してください。

`config_generation` カラムは、各データノードで有効になっているクラスタ構成のバージョンを示します。これは、構成パラメータの変更を行うために、クラスタのローリング再起動を実行する際に役立つことがあります。たとえば、次の `SELECT` ステートメントの出力から、ノード 1、2、4 では最新バージョンのクラスタ構成 (6) が使用されているが、ノード 3 ではまだ使用されていないことを確認できます。

```
mysql> USE ndbinfo;
Database changed
mysql> SELECT * FROM nodes;
+-----+-----+-----+-----+-----+
| node_id | uptime | status | start_phase | config_generation |
+-----+-----+-----+-----+-----+
| 1 | 10462 | STARTED | 0 | 6 |
| 2 | 10460 | STARTED | 0 | 6 |
| 3 | 10457 | STARTED | 0 | 5 |
| 4 | 10455 | STARTED | 0 | 6 |
+-----+-----+-----+-----+-----+
2 rows in set (0.04 sec)
```

したがって、上記で示したケースでは、クラスタのローリング再起動を完了するために、ノード 3 を再起動するようにしてください。

このテーブルでは、停止されたノードは考慮されません。NDB Cluster に 4 つのデータノード (ノード ID 1、2、3、および 4) があり、すべてのノードが正常に実行されているとします。このテーブルには、データノードごとに 1 つずつ 4 つの行が含まれます:

```
mysql> USE ndbinfo;
Database changed
mysql> SELECT * FROM nodes;
+-----+-----+-----+-----+-----+
| node_id | uptime | status | start_phase | config_generation |
+-----+-----+-----+-----+-----+
| 1 | 11776 | STARTED | 0 | 6 |
| 2 | 11774 | STARTED | 0 | 6 |
| 3 | 11771 | STARTED | 0 | 6 |
| 4 | 11769 | STARTED | 0 | 6 |
+-----+-----+-----+-----+-----+
4 rows in set (0.04 sec)
```

ノードのいずれかをシャットダウンすると、この `SELECT` ステートメントの出力には、次に示すように、まだ実行中のノードのみが表示されます。

```
ndb_mgm> 2 STOP
Node 2: Node shutdown initiated
Node 2: Node shutdown completed.
Node 2 has shutdown.
```

```
mysql> SELECT * FROM nodes;
+-----+-----+-----+-----+-----+
| node_id | uptime | status | start_phase | config_generation |
+-----+-----+-----+-----+-----+
| 1 | 11807 | STARTED | 0 | 6 |
| 3 | 11802 | STARTED | 0 | 6 |
| 4 | 11800 | STARTED | 0 | 6 |
+-----+-----+-----+-----+-----+
```


3 rows in set (0.02 sec)

23.5.14.39 ndbinfo operations_per_fragment テーブル

`operations_per_fragment` テーブルには、個々のフラグメントおよびフラグメントレプリカに対して実行された操作と、これらの操作の結果の一部に関する情報が表示されます。

`operations_per_fragment` テーブルには、次のカラムがあります:

- `fq_name`
このフラグメントの名前
- `parent_fq_name`
このフラグメントの親の名前
- `type`
オブジェクトのタイプ (可能性のある値についてはテキストを参照してください)
- `table_id`
このテーブルのテーブル ID
- `node_id`
このノードのノード ID
- `block_instance`
カーネルブロックインスタンス ID
- `fragment_num`
フラグメント ID (番号)
- `tot_key_reads`
このフラグメントレプリカのキー読取りの合計数
- `tot_key_inserts`
このフラグメントレプリカのキー挿入の合計数
- `tot_key_updates`
このフラグメントレプリカのキー更新の合計数
- `tot_key_writes`
このフラグメントレプリカに対するキー書込みの合計数
- `tot_key_deletes`
このフラグメントレプリカのキー削除の合計数
- `tot_key_refs`
拒否されたキー操作の数
- `tot_key_attrinfo_bytes`
すべての `attrinfo` 属性の合計サイズ
- `tot_key_keyinfo_bytes`

- すべての `keyinfo` 属性の合計サイズ
- `tot_key_prog_bytes`
`attrinfo` 属性によって実行されるすべての解釈済プログラムの合計サイズ
- `tot_key_inst_exec`
キー操作のために解釈済プログラムによって実行された命令の合計数
- `tot_key_bytes_returned`
キー読み取り操作から返されたすべてのデータおよびメタデータの合計サイズ
- `tot_frag_scans`
このフラグメントレプリカで実行されたスキャンの合計数
- `tot_scan_rows_examined`
スキャンによって検査された行の合計数
- `tot_scan_rows_returned`
クライアントに返された行の合計数
- `tot_scan_bytes_returned`
クライアントに返されるデータおよびメタデータの合計サイズ
- `tot_scan_prog_bytes`
スキャン操作の解釈されたプログラムの合計サイズ
- `tot_scan_bound_bytes`
順序付けされたインデックススキャンで使用されるすべての境界の合計サイズ
- `tot_scan_inst_exec`
スキャンで実行された命令の合計数
- `tot_qd_frag_scans`
このフラグメントレプリカのスキャンがキューに入れられた回数
- `conc_frag_scans`
このフラグメントレプリカで現在アクティブなスキャンの数 (キューに入れられたスキャンを除く)
- `conc_qd_frag_scans`
このフラグメントレプリカに対して現在キューに入っているスキャンの数
- `tot_commits`
このフラグメントレプリカにコミットされた行変更の合計数

メモ

`fq_name` には、このフラグメントレプリカが属するスキーマオブジェクトの完全修飾名が含まれます。現在の形式は次のとおりです:

- ベーステーブル: `DbName/def/TblName`

- BLOB テーブル: `DbName/def/NDB$BLOB_BaseTblId_ColNo`
- 順序付きインデックス: `sys/def/BaseTblId/IndexName`
- 一意インデックス: `sys/def/BaseTblId/IndexName$unique`

一意のインデックスに表示される `$unique` 接尾辞は `mysqlid` によって追加されます。別の NDB API クライアントアプリケーションによって作成されたインデックスの場合は、これが異なるか、存在しない可能性があります。

完全修飾オブジェクト名に示されている構文は、将来のリリースで変更される可能性がある内部インターフェースです。

次の SQL ステートメントによって作成および変更されたテーブル `t1` について考えてみます:

```
CREATE DATABASE mydb;

USE mydb;

CREATE TABLE t1 (
  a INT NOT NULL,
  b INT NOT NULL,
  t TEXT NOT NULL,
  PRIMARY KEY (b)
) ENGINE=ndbcluster;

CREATE UNIQUE INDEX ix1 ON t1(b) USING HASH;
```

`t1` にテーブル ID 11 が割り当てられている場合、次に示す `fq_name` 値が生成されます:

- ベーステーブル: `mydb/def/t1`
- BLOB テーブル: `mydb/def/NDB$BLOB_11_2`
- 順序付きインデックス (主キー): `sys/def/11/PRIMARY`
- 一意インデックス: `sys/def/11/ix1$unique`

インデックスまたは BLOB テーブルの場合、`parent_fq_name` カラムには対応する実テーブルの `fq_name` が含まれません。実テーブルの場合、このカラムは常に `NULL` です。

`type` カラムには、このフラグメントに使用されるスキーマオブジェクトタイプが表示され、`System table`、`User table`、`Unique hash index` または `Ordered index` のいずれかの値を取ることができます。BLOB テーブルは、`User table` として表示されます。

`table_id` カラムの値はいつでも一意ですが、対応するオブジェクトが削除されている場合は再利用できます。同じ ID は、`ndb_show_tables` ユーティリティを使用して表示できます。

`block_instance` カラムには、このフラグメントレプリカが属する LDM インスタンスが表示されます。これを使用して、`threadblocks` テーブルから特定のスレッドに関する情報を取得できます。このような最初のインスタンスには、常に 0 という番号が付けられます。

通常、2つのフラグメントレプリカがあり、そのような場合は、同じノードグループの2つの異なるデータノードで、各 `fragment_num` 値がテーブルに2回表示される必要があります。

NDB は順序付けされたインデックスに単一キーアクセスを使用しないため、`tot_key_reads`、`tot_key_inserts`、`tot_key_updates`、`tot_key_writes` および `tot_key_deletes` の数は順序付けられたインデックス操作によって増分されません。

注記

`tot_key_writes` を使用する場合、このコンテキストの書き込み操作では、キーが存在する場合は行が更新され、それ以外の場合は新しい行が挿入されることに注意してください。(これは、`REPLACE` SQL ステートメントの NDB 実装で使用します。)

`tot_key_refs` カラムには、LDM によって拒否されたキー操作の数が表示されます。通常、このような拒否は、キー(挿入)、「キーが見つかりません」エラー(更新、削除および読取り)が原因であるか、キーに一致する行の述語として使用される解釈済プログラムによって操作が拒否されたことが原因です。

`tot_key_attrinfo_bytes` および `tot_key_keyinfo_bytes` カラムによってカウントされる `attrinfo` および `keyinfo` 属性は、LDM による鍵操作の開始に使用される `LQHKEYREQ` シグナルの属性です ([The NDB Communication Protocol](#) を参照)。`attrinfo` には、通常、タプルフィールド値 (挿入および更新) または投影仕様 (読取り用) が含まれます。`keyinfo` には、このスキーマオブジェクト内で特定のタプルを検索するために必要な主キーまたは一意キーが含まれます。

`tot_frag_scans` で表示される値には、全体スキャン (各行を調べる) とサブセットのスキャンの両方が含まれます。一意インデックスおよび `BLOB` テーブルはスキャンされないため、これらのフラグメントレプリカの場合、この値は他のスキャン関連数と同様に 0 になります。

順序付けされたインデックススキャンは境界によって制限される可能性があるため、`tot_scan_rows_examined` では、指定されたフラグメントレプリカの行の合計数より少ない数が表示される場合があります。また、クライアントは、一致する可能性のあるすべての行が調査される前にスキャンを終了することもできます。これは、たとえば、`LIMIT` または `EXISTS` 句を含む SQL ステートメントを使用する場合に発生します。`tot_scan_rows_returned` は、常に `tot_scan_rows_examined` 以下です。

`tot_scan_bytes_returned` には、プッシュされた結合の場合、NDB カーネルの `DBSPJ` ブロックに返される投影が含まれます。

`tot_qd_frag_scans` は、単一フラグメントレプリカで同時に実行できるスキャンの数を制限する `MaxParallelScansPerFragment` データノード構成パラメータの設定によって影響を受ける可能性があります。

23.5.14.40 ndbinfo pgman_time_track_stats テーブル

このテーブルは、「NDB Cluster ディスクデータ」テーブルスペースのディスク操作のレイテンシに関する情報を示します。

`pgman_time_track_stats` テーブルには、次のカラムがあります:

- `node_id`
クラスタ内のこのノードの一意のノード ID
- `block_number`
ブロック番号 (`blocks` テーブルから)
- `block_instance`
ブロックインスタンス番号
- `upper_bound`
上限
- `page_reads`
ページ読取りレイテンシ (ミリ秒)
- `page_writes`
ページ書込みレイテンシ (ミリ秒)
- `log_waits`
ログ待機時間 (ミリ秒)
- `get_page`
`get_page()` コールのレイテンシ (ミリ秒)

メモ

読取りレイテンシ (`page_reads` カラム) は、読取りリクエストがファイルシステムスレッドに送信されてから読取りが完了して実行スレッドにレポートされるまでの時間を測定します。書込みレイテンシ (`page_writes`) は、同様の方

法で計算されます。「ディスクデータ」テーブルスペースに対して読取りまたは書き込みを行うページのサイズは、常に 32 KB です。

ログ待機待機時間 (`log_waits` カラム) は、undo ログがフラッシュされるまでページ書き込みが待機する必要がある時間の長さで、各ページ書き込みの前に実行する必要があります。

NDB 8.0.19 に `pgman_time_track_stats` テーブルが追加されました。

23.5.14.41 ndbinfo processes テーブル

このテーブルには NDB Cluster ノードプロセスに関する情報が含まれています。各ノードはテーブル内の行で表されます。このテーブルには、クラスタに接続されているノードのみが表示されます。構成されているが、クラスタに接続されていないノードに関する情報は、`nodes` テーブルおよび `config_nodes` テーブルから取得できます。

`processes` テーブルには、次のカラムがあります:

- `node_id`
クラスタ内の一意のノード ID
- `node_type`
ノードのタイプ (管理、データまたは API ノード。テキストを参照)
- `node_version`
このノードで実行されている NDB ソフトウェアプログラムのバージョン。
- `process_id`
このノードプロセス ID
- `angel_process_id`
このノード angel プロセスのプロセス ID
- `process_name`
実行可能ファイルの名前
- `service_URI`
このノードのサービス URI (テキストを参照)

メモ

`node_id` は、クラスタ内のこのノードに割り当てられた ID です。

`node_type` カラムには、次のいずれかの値が表示されます:

- **MGM**: 管理ノード。
- **NDB**: データノード。
- **API**: API または SQL ノード。

NDB Cluster ディストリビューションに同梱されている実行可能ファイルの場合、`node_version` は `8.0.23-ndb-8.0.23` などのソフトウェアクラスタのバージョン文字列を表示します。

`process_id` は、Linux では `top`、Windows プラットフォームではタスクマネージャなどのプロセス表示アプリケーションを使用して、ホストオペレーティングシステムによって表示されるノード実行可能プロセス ID です。

`angel_process_id` は、ノードアンジェルプロセスのシステムプロセス ID です。これにより、障害発生時にデータノードまたは SQL が自動的に再起動されます。管理ノードおよび SQL ノード以外の API ノードの場合、このカラムの値は `NULL` です。

`process_name` カラムには、実行中の実行可能ファイルの名前が表示されます。管理ノードの場合、これは `ndb_mgmd` です。データノードの場合、これは `ndbd` (シングルスレッド) または `ndbmt` (マルチスレッド) です。SQL ノードの場合、これは `mysqld` です。その他のタイプの API ノードの場合は、クラスタに接続されている実行可能プログラムの名前です。NDB API アプリケーションは、`Ndb_cluster_connection::set_name()` を使用してこのカスタム値を設定できます。

`service_URI` には、サービスネットワークアドレスが表示されます。管理ノードおよびデータノードの場合、使用されるスキームは `ndb://` です。SQL ノードの場合、これは `mysql://` です。デフォルトでは、SQL ノード以外の API ノードはスキームに `ndb://` を使用します。NDB API アプリケーションは、`Ndb_cluster_connection::set_service_uri()` を使用してこれをカスタム値に設定できます。このスキームのあとには、ノードタイプに関係なく、該当するノードの NDB トランスポータで使用される IP アドレスが続きます。管理ノードおよび SQL ノードの場合、このアドレスにはポート番号が含まれます (通常、管理ノードの場合は 1186、SQL ノードの場合は 3306)。SQL ノードが `bind_address` システム変数セットで起動された場合、バインドアドレスが `*`、`0.0.0.0` または `::` に設定されていないかぎり、このアドレスがトランスポータアドレスのかわりに使用されます。

様々な構成オプションを反映する SQL ノードの `service_URI` 値に、追加のパス情報を含めることができます。たとえば、`mysql://198.51.100.3/tmp/mysql.sock` は SQL ノードが `skip_networking` システム変数を有効にして起動されたことを示し、`mysql://198.51.100.3:3306/?server-id=1` はこの SQL ノードに対してレプリケーションが有効になっていることを示します。

23.5.14.42 ndbinfo resources テーブル

このテーブルは、データノードリソースの可用性および使用状況に関する情報を示します。

これらのリソースは、スーパープールと呼ばれることがあります。

`resources` テーブルには、次のカラムがあります:

- `node_id`
このデータノードの一意のノード ID。
- `resource_name`
リソースの名前。テキストを参照してください。
- `reserved`
このリソース用に予約された量。
- `used`
このリソースで実際に使用された量。
- `max`
ノードが最後に起動されてから使用されたこのリソースの最大量。

メモ

`resource_name` には、次のテーブルに示すいずれかの名前を指定できます:

- `RESERVED`: システムによって予約されています。上書きできません。
- `DISK_OPERATIONS`: ログファイルグループが割り当てられている場合は、undo ログバッファのサイズを使用してこのリソースのサイズが設定されます。このリソースは、undo ログファイルグループに undo ログバッファを割り当てるためのみ使用されます。そのようなグループは 1 つのみです。超過割当ては、`CREATE LOGFILE GROUP` で必要に応じて発生します。
- `DISK_RECORDS`: ディスクデータ操作に割り当てられたレコード。
- `DATA_MEMORY`: メインメモリーテーブル、インデックス、およびハッシュインデックスに使用されます。DataMemory と IndexMemory の合計に、それぞれ 32 KB の 8 ページを加えたもの (IndexMemory が設定されている場合)。超過割当てできません。

- **JOBBUFFER**: NDB スケジューラによるジョブバッファの割り当てに使用されます。過剰に割り当てることはできません。これは、スレッド当たり約 2 MB と、通信可能なすべてのスレッドの双方向の 1 MB バッファです。大規模な構成の場合、これは数 GB を消費します。
- **FILE_BUFFERS**: **DBLQH** カーネルブロックの redo ログハンドラによって使用されます。過剰割り当てはできません。サイズは `NoOfFragmentLogParts * RedoBuffer` にログファイル部分ごとに 1 MB を加えたものです。
- **TRANSPORTER_BUFFERS**: `ndbmt` によるバッファの送信に使用され、`TotalSendBufferMemory` と `ExtraSendBufferMemory` の合計です。最大 25% 超過割当可能なこのリソース。`TotalSendBufferMemory` は、ノード当たりの送信バッファメモリー (デフォルト値は 2 MB) を合計して計算されます。したがって、4 つのデータノードと 8 つの API ノードを持つシステムでは、データノードに $12 * 2$ MB の送信バッファメモリーがあります。`ExtraSendBufferMemory` は、`ndbmt` で使用され、スレッド当たり最大 2 MB の追加メモリーを使用します。したがって、4 つの LDM スレッド、2 つの TC スレッド、1 つのメインスレッド、1 つのレプリケーションスレッド、および 2 つの受信スレッドでは、`ExtraSendBufferMemory` は $10 * 2$ MB です。このリソースの超過割り当ては、`SharedGlobalMemory` データノード構成パラメータを設定することによって実行できます。
- **DISK_PAGE_BUFFER**: ディスクページバッファに使用され、`DiskPageBufferMemory` 構成パラメータによって決定されます。超過割当できません。
- **QUERY_MEMORY**: **DBSPJ** カーネルブロックによって使用されます。
- **SCHEMA_TRANS_MEMORY**: 最小値は 2 MB で、残りの使用可能なメモリーを使用するように過剰に割り当てることができます。

23.5.14.43 ndbinfo restart_info テーブル

`restart_info` テーブルには、ノードの再起動操作に関する情報が含まれます。テーブルの各エントリは、指定されたノード ID を持つデータノードからのリアルタイムのノード再起動ステータスレポートに対応します。特定のノードの最新のレポートのみが表示されます。

`restart_info` テーブルには、次のカラムがあります:

- `node_id`
クラスタ内のノード ID
- `node_restart_status`
ノードステータス。値についてはテキストを参照してください。これらはそれぞれ、`node_restart_status_int` の可能な値に対応しています。
- `node_restart_status_int`
ノードステータスコード。値についてはテキストを参照してください。
- `secs_to_complete_node_failure`
ノード障害処理が完了するまでの時間 (秒)
- `secs_to_allocate_node_id`
ノード障害の完了からノード ID の割当てまでの時間 (秒)
- `secs_to_include_in_heartbeat_protocol`
ノード ID の割当てからハートビートプロトコルに含めるまでの時間 (秒)
- `secs_until_wait_for_ndbcntr_master`
ハートビートプロトコルに含まれてから `NDBCNTR` マスターが開始されるまでの時間 (秒)
- `secs_wait_for_ndbcntr_master`
`NDBCNTR` マスターによる起動の受入れの待機に費やされた時間 (秒)

- [secs_to_get_start_permitted](#)
マスターからの起動権限の受信から、すべてのノードがこのノードの起動を受け入れられるまでの経過時間 (秒)
- [secs_to_wait_for_lcp_for_copy_meta_data](#)
メタデータをコピーするまでの LCP 完了の待機時間 (秒)
- [secs_to_copy_meta_data](#)
マスターから新規起動ノードへのメタデータのコピーに必要な時間 (秒)
- [secs_to_include_node](#)
GCP を待機し、すべてのノードをプロトコルに含める時間 (秒)
- [secs_starting_node_to_request_local_recovery](#)
ノードの起動時にローカルリカバリのリクエストの待機に費やされた時間 (秒)
- [secs_for_local_recovery](#)
ノードによるローカルリカバリの開始に必要な時間 (秒)
- [secs_restore_fragments](#)
LCP ファイルからフラグメントをリストアするために必要な時間 (秒)
- [secs_undo_disk_data](#)
レコードのディスクデータ部分で undo ログを実行するために必要な時間 (秒)
- [secs_exec_redo_log](#)
リストアされたすべてのフラグメントで redo ログを実行するために必要な時間 (秒)
- [secs_index_rebuild](#)
リストアされたフラグメントでインデックスを再構築するために必要な時間 (秒)
- [secs_to_synchronize_starting_node](#)
ライブノードからの開始ノードの同期に必要な時間 (秒)
- [secs_wait_lcp_for_restart](#)
LCP の起動および再起動が完了するまでに必要な時間 (秒)
- [secs_wait_subscription_handover](#)
レプリケーションサブスクリプションのハンドオーバーの待機に費やされた時間 (秒)
- [total_restart_secs](#)
ノード障害からノードが再起動されるまでの合計秒数

メモ

次のリストには、[node_restart_status_int](#) カラムに定義された値とその内部ステータス名 (カッコ内) および [node_restart_status](#) カラムに表示される対応するメッセージが含まれています:

- 0 (ALLOCATED_NODE_ID)
Allocated node id
- 1 (INCLUDED_IN_HB_PROTOCOL)

- Included in heartbeat protocol
- 2 (NDBCNTR_START_WAIT)
Wait for NDBCNTR master to permit us to start
- 3 (NDBCNTR_STARTED)
NDBCNTR master permitted us to start
- 4 (START_PERMITTED)
All nodes permitted us to start
- 5 (WAIT_LCP_TO_COPY_DICT)
Wait for LCP completion to start copying metadata
- 6 (COPY_DICT_TO_STARTING_NODE)
Copying metadata to starting node
- 7 (INCLUDE_NODE_IN_LCP_AND_GCP)
Include node in LCP and GCP protocols
- 8 (LOCAL_RECOVERY_STARTED)
Restore fragments ongoing
- 9 (COPY_FRAGMENTS_STARTED)
Synchronizing starting node with live nodes
- 10 (WAIT_LCP_FOR_RESTART)
Wait for LCP to ensure durability
- 11 (WAIT_SUMA_HANOVER)
Wait for handover of subscriptions
- 12 (RESTART_COMPLETED)
Restart completed
- 13 (NODE_FAILED)
Node failed, failure handling in progress
- 14 (NODE_FAILURE_COMPLETED)
Node failure handling completed
- 15 (NODE_GETTING_PERMIT)
All nodes permitted us to start
- 16 (NODE_GETTING_INCLUDED)
Include node in LCP and GCP protocols
- 17 (NODE_GETTING_SYNCED)
Synchronizing starting node with live nodes

- 18 (NODE_GETTING_LCP_WAITED)

[none]

- 19 (NODE_ACTIVE)

Restart completed

- 20 (NOT_DEFINED_IN_CLUSTER)

[none]

- 21 (NODE_NOT_RESTARTED_YET)

Initial state

ステータス番号 0 から 12 はマスターノードにのみ適用されます。テーブルに示されている残りの番号は、再起動するすべてのデータノードに適用されます。ステータス番号 13 および 14 は、ノード障害状態を定義します。20 および 21 は、特定のノードの再起動に関する情報がない場合に発生します。

[セクション23.5.4「NDB Cluster 起動フェーズのサマリー」](#)も参照してください。

23.5.14.44 ndbinfo server_locks テーブル

[server_locks](#) テーブルの構造は [cluster_locks](#) テーブルと似ており、後者のテーブルにある情報のサブセットを提供しますが、これはそれが存在する SQL ノード (MySQL サーバー) に固有のもので、([cluster_locks](#) テーブルには、クラスタ内のすべてのロックに関する情報が表示されます。) より正確には、[server_locks](#) には、現在の [mysqld](#) インスタンスに属するスレッドによってリクエストされたロックに関する情報が含まれ、[server_operations](#) へのコンパニオンテーブルとして機能します。これは、ロックパターンを特定の MySQL ユーザーセッション、クエリーまたはユーザーケースと関連させる場合に役立ちます。

[server_locks](#) テーブルには、次のカラムがあります:

- [mysql_connection_id](#)

MySQL 接続 ID

- [node_id](#)

レポートノードの ID

- [block_instance](#)

レポート LDM インスタンスの ID

- [tableid](#)

この行を含むテーブルの ID

- [fragmentid](#)

ロックされた行を含むフラグメントの ID

- [rowid](#)

ロックされた行の ID

- [transid](#)

トランザクション ID

- [mode](#)

ロックリクエストモード

- [state](#)
ロック状態
- [detail](#)
これが行ロックキューで最初にロックを保持しているかどうか
- [op](#)
操作タイプ
- [duration_millis](#)
ロックの待機または保持にかかったミリ秒
- [lock_num](#)
ロックオブジェクトの ID
- [waiting_for](#)
この ID のロックを待機中

メモ

[mysql_connection_id](#) カラムには、[SHOW PROCESSLIST](#) に示されている MySQL 接続またはスレッド ID が表示されます。

[block_instance](#) は、カーネルブロックのインスタンスを指します。この番号は、ブロック名とともに使用して、[threadblocks](#) テーブル内の特定のインスタンスを検索できます。

[tableid](#) は、NDB によってテーブルに割り当てられます。[ndb_show_tables](#) の出力と同様に、他の [ndbinfo](#) テーブルのこのテーブルにも同じ ID が使用されます。

[transid](#) カラムに表示されるトランザクション ID は、NDB API によって生成された、現在のロックを要求または保持しているトランザクションの識別子です。

[mode](#) カラムには、常に **S** (共有ロック) または **X** (排他ロック) のいずれかのロックモードが表示されます。トランザクションが特定の行に対して排他ロックを持っている場合、その行に対する他のすべてのロックは同じトランザクション ID を持ちます。

[state](#) カラムにはロック状態が表示されます。その値は、常に **H** (保持) または **W** (待機) のいずれかです。待機中のロックリクエストは、別のトランザクションによって保持されているロックを待機します。

[detail](#) カラムは、このロックが、影響を受ける行ロックキューで最初に保持しているロックかどうかを示します。この場合、このカラムには * (アスタリスク文字) が含まれます。それ以外の場合、このカラムは空です。この情報は、ロックリクエストのリスト内の一意のエントリを識別するのに役立ちます。

[op](#) カラムには、ロックをリクエストしている操作のタイプが表示されます。これは常に、**READ**, **INSERT**, **UPDATE**, **DELETE**, **SCAN** または **REFRESH** のいずれかの値です。

[duration_millis](#) カラムには、このロックリクエストがロックを待機または保持しているミリ秒数が表示されます。待機中のリクエストに対してロックが付与されると、これは 0 にリセットされます。

ロック ID ([lockid](#) カラム) は、このノードおよびブロックインスタンスに対して一意です。

[lock_state](#) カラムの値が **W** の場合、このロックは付与を待機しており、[waiting_for](#) カラムには、このリクエストが待機しているロックオブジェクトのロック ID が表示されます。それ以外の場合、[waiting_for](#) は空です。[waiting_for](#) は、([node_id](#), [block_instance](#), [tableid](#), [fragmentid](#) および [rowid](#) で識別される) 同じ行のロックのみを参照できます。

23.5.14.45 ndbinfo server_operations テーブル

`server_operations` テーブルは、現在の SQL ノード (MySQL サーバー) が現在関与している進行中のすべての NDB 操作を表すエントリを格納します。それは事実上、その他の SQL ノードおよび API ノードに対する操作が表示されない `cluster_operations` テーブルのサブセットです。

`server_operations` テーブルには、次のカラムがあります:

- `mysql_connection_id`
MySQL Server 接続 ID
- `node_id`
ノード ID
- `block_instance`
ブロックインスタンス
- `transid`
トランザクション ID
- `operation_type`
操作のタイプ (可能性のある値についてはテキストを参照)
- `state`
操作の状態 (可能性のある値についてはテキストを参照)
- `tableid`
テーブル ID
- `fragmentid`
フラグメント ID
- `client_node_id`
クライアントノード ID
- `client_block_ref`
クライアントのブロック参照
- `tc_node_id`
トランザクションコーディネータノード ID
- `tc_block_no`
トランザクションコーディネータブロック番号
- `tc_block_instance`
トランザクションコーディネータブロックインスタンス

メモ

`mysql_connection_id` は、`SHOW PROCESSLIST` の出力に示された接続またはセッション ID と同じです。それは `INFORMATION_SCHEMA` のテーブル `NDB_TRANSID_MYSQL_CONNECTION_MAP` から取得されます。

`block_instance` は、カーネルブロックのインスタンスを指します。この番号は、ブロック名とともに使用して、`threadblocks` テーブル内の特定のインスタンスを検索できます。

トランザクション ID (`transid`) は一意の 64 ビット番号で、NDB API `getTransactionId()` メソッドを使用して取得できます。(現在、MySQL サーバーは進行中のトランザクションの NDB API トランザクション ID を公開しません。)

`operation_type` カラムは、`READ`、`READ-SH`、`READ-EX`、`INSERT`、`UPDATE`、`DELETE`、`WRITE`、`UNLOCK`、`REFRESH`、`SCAN`、`SCAN-SH`、`SCAN-EX`、または `<unknown>` のいずれかの値を取ることができます。

`state` カラム

は、`ABORT_QUEUED`、`ABORT_STOPPED`、`COMMITTED`、`COMMIT_QUEUED`、`COMMIT_STOPPED`、`COPY_CLOSE_ST` または `WAIT_TUP_TO_ABORT` のいずれかの値を取ることができます。(`ndbinfo_show_hidden` を有効にして MySQL サーバーが実行されている場合は、通常は非表示になっている `ndb$tblqh_tcconnect_state` テーブルから選択することで、この状態のリストを表示できます。)

`ndb_show_tables` の出力をチェックして、テーブル ID から NDB テーブルの名前を取得できます。

`fragid` は、`ndb_desc --extra-partition-info` (短縮形式 `-p`) の出力で見られるパーティション番号と同じです。

`client_node_id` および `client_block_ref` では、`client` は NDB Cluster API または SQL ノード (つまり、NDB API クライアントまたはクラスタに接続された MySQL Server) を指します。

`block_instance` および `tc_block_instance` カラムは NDB カーネルブロックのインスタンス番号を提供します。これらを使用して、`threadblocks` テーブルから特定のスレッドに関する情報を取得できます。

23.5.14.46 ndbinfo server_transactions テーブル

`server_transactions` テーブルは、`cluster_transactions` テーブルのサブセットですが、このテーブルに関連する接続 ID が含まれている場合は、現在の SQL ノード (MySQL サーバー) が参加しているトランザクションのみが含まれます。

`server_transactions` テーブルには、次のカラムがあります:

- `mysql_connection_id`
MySQL Server 接続 ID
- `node_id`
トランザクションコーディネータノード ID
- `block_instance`
トランザクションコーディネータブロックインスタンス
- `transid`
トランザクション ID
- `state`
操作の状態 (可能性のある値についてはテキストを参照)
- `count_operations`
トランザクションでのステータフル操作の数
- `outstanding_operations`
ローカルデータ管理レイヤー (LQH ブロック) でまだ実行されている操作
- `inactive_seconds`
API の待機に要した時間
- `client_node_id`
クライアントノード ID

- [client_block_ref](#)

クライアントのブロック参照

メモ

[mysql_connection_id](#) は、`SHOW PROCESSLIST` の出力に示された接続またはセッション ID と同じです。それは `INFORMATION_SCHEMA` のテーブル `NDB_TRANSID_MYSQL_CONNECTION_MAP` から取得されます。

[block_instance](#) は、カーネルブロックのインスタンスを指します。この番号は、ブロック名とともに使用して、`threadblocks` テーブル内の特定のインスタンスを検索できます。

トランザクション ID (`transid`) は一意の 64 ビット番号で、NDB API `getTransactionId()` メソッドを使用して取得できます。(現在、MySQL サーバーは進行中のトランザクションの NDB API トランザクション ID を公開しません。)

`state` カラム

は、`CS_ABORTING`、`CS_COMMITTING`、`CS_COMMIT_SENT`、`CS_COMPLETE_SENT`、`CS_COMPLETING`、`CS_CONNECTED` のいずれかの値を持つ可能性があります。(`ndbinfo_show_hidden` を有効にして MySQL サーバーが実行されている場合は、通常は非表示になっている `ndb$dbtc_apiconnect_state` テーブルから選択することで、この状態のリストを表示できます。)

`client_node_id` および `client_block_ref` では、`client` は NDB Cluster API または SQL ノード (つまり、NDB API クライアントまたはクラスタに接続された MySQL Server) を指します。

`block_instance` カラムには、`DBTC` カーネルブロックインスタンス番号が表示されます。これを使用して、`threadblocks` テーブルから特定のスレッドに関する情報を取得できます。

23.5.14.47 ndbinfo table_distribution_status テーブル

`table_distribution_status` テーブルには、NDB テーブルのテーブル分散の進行状況に関する情報が表示されます。

`table_distribution_status` テーブルには、次のカラムがあります:

- [node_id](#)

ノード ID

- [table_id](#)

テーブル ID

- [tab_copy_status](#)

ディスクへのテーブル配布データのコピーのステータス (`IDLE`、`SR_PHASE1_READ_PAGES`、`SR_PHASE2_READ_TABLE`、`SR_PHASE3_COPY_TABLE`、`REMOVE_NODE`、`LCP_READ_TABLE`、`COPY_TAB_REQ`、`COPY_NODE_STATE`、`ADD_TABLE_COORDINATOR` (NDB 8.0.23 より前) のいずれか): `ADD_TABLE_MASTER`、`ADD_TABLE_PARTICIPANT` (NDB 8.0.23 より前): `ADD_TABLE_SLAVE`、`INVALIDATE_NODE_LCP`、`ALTER_TABLE`、`COPY_TO_SAVE` または `GET_TABINFO`

- [tab_update_status](#)

テーブル分散データの更新のステータス (`IDLE`、`LOCAL_CHECKPOINT`、`LOCAL_CHECKPOINT_QUEUED`、`REMOVE_NODE`、`COPY_TAB_REQ`、`ADD_TABLE_MASTER`、`ADD_TABLE_SLAVE`、`INVALIDATE_NODE_LCP` または `CALLBACK` のいずれか)

- [tab_lcp_status](#)

テーブル LCP のステータス。 `ACTIVE` (ローカルチェックポイントの実行を待機中)、`WRITING_TO_FILE` (チェックポイントは実行されましたが、まだディスクに書き込まれていません) または `COMPLETED` (チェックポイントは実行され、ディスクに永続化されます) のいずれか

- [tab_status](#)

テーブルの内部ステータス。 `ACTIVE` (テーブルが存在)、`CREATING` (テーブルを作成中) または `DROPPING` (テーブルを削除中) のいずれかです

- [tab_storage](#)

テーブルのリカバリ可能性。 [NORMAL](#) (redo ログおよびチェックポイントを使用して完全にリカバリ可能)、 [NOLOGGING](#) (ノードクラッシュからリカバリ可能、次の空のクラスタクラッシュから空)、または [TEMPORARY](#) (リカバリ不可) のいずれか

- [tab_partitions](#)

テーブル内のパーティション数

- [tab_fragments](#)

[tab_partitions](#) * [number of node groups]と等しい完全にレプリケートされたテーブルの場合、テーブル内のフラグメントの数 (通常は [tab_partitions](#) と同じ)

- [current_scan_count](#)

アクティブなスキャンの現在の数

- [scan_count_wait](#)

[ALTER TABLE](#) が完了するまでに実行を待機しているスキャンの現在の数。

- [is_reorg_ongoing](#)

テーブルが現在再編成されているかどうか (true の場合は 1)

23.5.14.48 ndbinfo table_fragments テーブル

[table_fragments](#) テーブルは、NDB テーブルの断片化、パーティション化、分散および (内部) レプリケーションに関する情報を提供します。

[table_fragments](#) テーブルには、次のカラムがあります:

- [node_id](#)

ノード ID (DIH マスター)

- [table_id](#)

テーブル ID

- [partition_id](#)

パーティション ID

- [fragment_id](#)

フラグメント ID (テーブルが完全にレプリケートされないかぎり、パーティション ID と同じ)

- [partition_order](#)

パーティション内のフラグメントの順序

- [log_part_id](#)

フラグメントのログ部分 ID

- [no_of_replicas](#)

フラグメントレプリカの数

- [current_primary](#)

現在のプライマリノード ID

- [preferred_primary](#)
優先プライマリノード ID
- [current_first_backup](#)
現在の最初のバックアップノード ID
- [current_second_backup](#)
現在の 2 番目のバックアップノード ID
- [current_third_backup](#)
現在の 3 番目のバックアップノード ID
- [num_alive_replicas](#)
ライブフラグメントレプリカの現在の数
- [num_dead_replicas](#)
デッドフラグメントレプリカの現在の数
- [num_lcp_replicas](#)
チェックポイントを実行するフラグメントレプリカの数

23.5.14.49 ndbinfo table_info テーブル

[table_info](#) テーブルには、個々の **NDB** テーブルに対して有効なロギング、チェックポイント、分散および記憶域オプションに関する情報が表示されます。

[table_info](#) テーブルには、次のカラムがあります:

- [table_id](#)
テーブル ID
- [logged_table](#)
テーブルがログに記録されるか (1) または否か (0)
- [row_contains_gci](#)
テーブルの行に GCI が含まれているかどうか (1 true、0 false)
- [row_contains_checksum](#)
テーブルの行にチェックサムが含まれるかどうか (1 true、0 false)
- [read_backup](#)
バックアップフラグメントの複製が読み取られた場合は 1、それ以外の場合は 0
- [fully_replicated](#)
テーブルが完全にレプリケートされている場合は 1、それ以外の場合は 0
- [storage_type](#)
テーブル記憶域タイプ (**MEMORY** または **DISK** のいずれか)
- [hashmap_id](#)
ハッシュマップ ID

- [partition_balance](#)

テーブルに使用されるパーティションバランス (フラグメント数タイプ)。FOR_RP_BY_NODE, FOR_RA_BY_NODE, FOR_RP_BY_LDM または FOR_RA_BY_LDM のいずれか

- [create_gci](#)

テーブルが作成された GCI

23.5.14.50 ndbinfo table_replicas テーブル

[table_replicas](#) テーブルは、NDB テーブルフラグメントおよびフラグメントレプリカのコピー、配布およびチェックポイントに関する情報を提供します。

[table_replicas](#) テーブルには、次のカラムがあります:

- [node_id](#)

データのフェッチ元ノードの ID (DIH マスター)

- [table_id](#)

テーブル ID

- [fragment_id](#)

フラグメント ID

- [initial_gci](#)

テーブルの初期 GCI

- [replica_node_id](#)

フラグメントレプリカが格納されるノードの ID

- [is_lcp_ongoing](#)

LCP がこのフラグメントで進行中の場合は 1、それ以外の場合は 0

- [num_crashed_replicas](#)

クラッシュしたフラグメントレプリカインスタンスの数

- [last_max_gci_started](#)

最新 LCP で最も高い GCI が開始されました

- [last_max_gci_completed](#)

最新の LCP で完了した最高 GCI

- [last_lcp_id](#)

最新 LCP の ID

- [prev_lcp_id](#)

前の LCP の ID

- [prev_max_gci_started](#)

前の LCP で開始された最高 GCI

- [prev_max_gci_completed](#)

前の LCP で完了した最高 GCI

- [last_create_gci](#)

最後にクラッシュしたフラグメントレプリカインスタンスの GCI の最終作成

- [last_replica_gci](#)

最後にクラッシュしたフラグメントレプリカインスタンスの最後の GCI

- [is_replica_alive](#)

このフラグメントレプリカが有効な場合は 1、そうでない場合は 0

23.5.14.51 ndbinfo tc_time_track_stats テーブル

[tc_time_track_stats](#) テーブルは、API ノードが [NDB](#) にアクセスすることで、データノード内の [DBTC](#) ブロック (TC) インスタンスから取得されたタイムトラッキング情報を提供します。各 TC インスタンスは、API ノードまたは他のデータノードのかわりに実行される一連のアクティビティのレイテンシを追跡します。これらのアクティビティには、トランザクション、トランザクションエラー、キー読み取り、キー書き込み、一意インデックス操作、任意のタイプの失敗したキー操作、スキャン、失敗したスキャン、フラグメントスキャンおよび失敗したフラグメントスキャンが含まれます。

カウンタのセットはアクティビティごとに保持され、各カウンタは上限以下の待機時間の範囲をカバーします。各アクティビティの終了時に、その待機時間が決定され、適切なカウンタが増分されます。[tc_time_track_stats](#) では、この情報が行として表示され、次の各インスタンスの行が表示されます:

- ID を使用したデータノード
- TC ブロックインスタンス
- ID を使用したその他の通信データノードまたは API ノード
- 上限値

各行には、アクティビティタイプごとの値が含まれます。これは、行で指定された範囲内 (つまり、待機時間が上限を超えない範囲) でこのアクティビティが発生した回数です。

[tc_time_track_stats](#) テーブルには、次のカラムがあります:

- [node_id](#)

要求ノード ID

- [block_number](#)

TC ブロック番号

- [block_instance](#)

TC ブロックインスタンス番号

- [comm_node_id](#)

通信 API またはデータノードのノード ID

- [upper_bound](#)

間隔の上限 (マイクロ秒)

- [scans](#)

スキャンのオープンからクローズまでの成功期間に基づいて、スキャンをリクエストしている API またはデータノードに対して追跡されます。

- [scan_errors](#)

失敗したスキャンのオープンからクローズまでの期間に基づいて、スキャンをリクエストしている API またはデータノードに対して追跡されます。

- [scan_fragments](#)

オープンからクローズまでの正常なフラグメントスキャンの期間に基づいて、それらを実行しているデータノードに対して追跡されます

- [scan_fragment_errors](#)

失敗したフラグメントスキャンのオープンからクローズまでの期間に基づいて、それらを実行しているデータノードに対して追跡されます

- [transactions](#)

成功したトランザクションの開始からコミット [ACK](#) の送信までの期間に基づいて、トランザクションをリクエストしている API またはデータノードに対して追跡されます。ステートレストランザクションは含まれません。

- [transaction_errors](#)

失敗したトランザクションの開始から失敗までの期間に基づいて、トランザクションをリクエストしている API またはデータノードに対して追跡されます。

- [read_key_ops](#)

ロック付きの正常な主キー読取りの期間に基づきます。リクエストしている API またはデータノードと、それらを実行しているデータノードの両方に対して追跡されます。

- [write_key_ops](#)

主キーの書込みが成功した期間に基づいて、それらを実行している API またはデータノードと、それらを実行しているデータノードの両方に対して追跡されます。

- [index_key_ops](#)

一意インデックスキー操作が成功した期間に基づいて、それらを実行している API またはデータノードと、実テーブルの読取りを実行しているデータノードの両方に対して追跡されます。

- [key_op_errors](#)

失敗したすべてのキー読取りまたは書込み操作の期間に基づいて、それらを実行している API またはデータノードと、それらを実行しているデータノードの両方に対して追跡されます。

メモ

[block_instance](#) カラムには、[DBTC](#) カーネルブロックインスタンス番号が表示されます。これをブロック名とともに使用して、[threadblocks](#) テーブルから特定のスレッドに関する情報を取得できます。

23.5.14.52 ndbinfo threadblocks テーブル

[threadblocks](#) テーブルは、[NDB](#) カーネルブロックのデータノード、スレッド、およびインスタンスを関連付けます。

[threadblocks](#) テーブルには、次のカラムがあります:

- [node_id](#)

ノード ID

- [thr_no](#)

スレッド ID

- [block_name](#)

ブロック名

- [block_instance](#)

ブロックインスタンス番号

メモ

このテーブルの [block_name](#) の値は、[ndbinfo.blocks](#) テーブルから選択したときに [block_name](#) カラムに表示される値のいずれかです。可能な値のリストは、特定の NDB Cluster リリースで静的ですが、リストはリリースによって異なる場合があります。

[block_instance](#) カラムには、カーネルブロックインスタンス番号が表示されます。

23.5.14.53 ndbinfo threads テーブル

[threads](#) テーブルは、NDB カーネルで実行されているスレッドに関する情報を提供します。

[threads](#) テーブルには、次のカラムがあります:

- [node_id](#)

スレッドが実行中のノードの ID

- [thr_no](#)

スレッド ID (このノードに固有)

- [thread_name](#)

スレッド名 (スレッドのタイプ)

- [thread_description](#)

スレッド (タイプ) の説明

メモ

スレッドの説明を含む、2 ノードのサンプルクラスタからの出力例を次に示します:

```
mysql> SELECT * FROM threads;
+-----+-----+-----+-----+
| node_id | thr_no | thread_name | thread_description |
+-----+-----+-----+-----+
| 5 | 0 | main | main thread, schema and distribution handling |
| 5 | 1 | rep | rep thread, asynch replication and proxy block handling |
| 5 | 2 | ldm | ldm thread, handling a set of data partitions |
| 5 | 3 | recv | receive thread, performing receive and polling for new receives |
| 6 | 0 | main | main thread, schema and distribution handling |
| 6 | 1 | rep | rep thread, asynch replication and proxy block handling |
| 6 | 2 | ldm | ldm thread, handling a set of data partitions |
| 6 | 3 | recv | receive thread, performing receive and polling for new receives |
+-----+-----+-----+-----+
8 rows in set (0.01 sec)
```

NDB 8.0.23 では、[ThreadConfig](#) 引数 [main](#) または [rep](#) のいずれかを 0 に設定し、もう一方を 1 のままにする可能性が導入されています。この場合、スレッド名は [main_rep](#) で、その説明は [main and rep thread, schema, distribution, proxy block and asynch replication handling](#) です。NDB 8.0.23 以降では、[main](#) と [rep](#) の両方を 0 に設定することもできます。この場合、生成されるスレッドの名前は [main_rep_recv](#) としてこのテーブルに表示され、その説明は [main, rep and recv thread, schema, distribution, proxy block and asynch replication handling and handling receive and polling for new receives](#) です。

23.5.14.54 ndbinfo threadstat テーブル

[threadstat](#) テーブルは、NDB カーネルで実行されているスレッドの統計の概略のスナップショットを示します。

threadstat テーブルには、次のカラムがあります:

- `node_id`
ノード ID
- `thr_no`
スレッド ID
- `thr_nm`
スレッド名
- `c_loop`
メインループ内のループ数
- `c_exec`
実行されたシグナルの数
- `c_wait`
追加入力を待機している回数
- `c_l_sent_prioa`
独自のノードに送信された優先度 A のシグナルの数
- `c_l_sent_priob`
独自のノードに送信された優先度 B のシグナルの数
- `c_r_sent_prioa`
リモートノードに送信された優先度 A のシグナルの数
- `c_r_sent_priob`
リモートノードに送信された優先度 B のシグナルの数
- `os_tid`
OS スレッド ID
- `os_now`
OS 時間 (ms)
- `os_ru_utime`
OS ユーザーの CPU 時間 (μ s)
- `os_ru_stime`
OS システムの CPU 時間 (μ s)
- `os_ru_minflt`
OS ページ再利用 (ソフトページフォルト)
- `os_ru_majflt`
OS ページフォルト (ハードページフォルト)
- `os_ru_nvcsw`

OS の自発的コンテキストスイッチ

- [os_ru_nivcsw](#)

OS の非自発的コンテキストスイッチ

メモ

[os_time](#) では、[gettimeofday\(\)](#) システムコールが使用されます。

[os_ru_utime](#)、[os_ru_stime](#)、[os_ru_minflt](#)、[os_ru_majflt](#)、[os_ru_nivcsw](#)、および [os_ru_nivcsw](#) カラムの値は、[getrusage\(\)](#) システムコールまたは同等のものを使用することで取得されます。

このテーブルには特定の時点で取得されたカウントが格納されるため、最適な結果を得るには、このテーブルでクエリーを定期的に行い、その結果を 1 つまたは複数の中間テーブルに格納する必要があります。MySQL サーバーのイベントスケジューラを採用して、このようなモニタリングを自動化できます。詳細は、[セクション25.4「イベントスケジューラの使用」](#)を参照してください。

23.5.14.55 ndbinfo transporters テーブル

このテーブルには、NDB トランスポータに関する情報が格納されます。

[transporters](#) テーブルには、次のカラムがあります:

- [node_id](#)
クラスタ内のこのデータノードの一意のノード ID
- [remote_node_id](#)
リモートデータノードのノード ID
- [status](#)
接続のステータス
- [remote_address](#)
リモートホストの名前または IP アドレス
- [bytes_sent](#)
この接続を使用して送信されたバイト数
- [bytes_received](#)
この接続を使用して受信されたバイト数
- [connect_count](#)
このトランスポータ上で確立された接続の回数
- [overloaded](#)
このトランスポータが現在過負荷状態である場合は 1、それ以外の場合は 0
- [overload_count](#)
このトランスポータが接続してから過負荷状態になった回数
- [slowdown](#)
このトランスポータが低速状態の場合は 1、それ以外の場合は 0
- [slowdown_count](#)

接続以降、このトランスポータが低速状態になった回数

メモ

`transporters` テーブルには、クラスタ内で実行中のデータノードごとに、そのノードとクラスタ内のすべてのノード (自分自身を含む) との各接続のステータスを示す行が表示されます。この情報は、テーブルの `status` カラムに示されます。このカラムは、`CONNECTING`、`CONNECTED`、`DISCONNECTING`、または `DISCONNECTED` のいずれかの値になる可能性があります。

構成されているが、現在はクラスタに接続されていない API および管理ノードへの接続は、`DISCONNECTED` のステータスで示されます。`node_id` が、現在接続されていないデータノードの行である場合、このテーブルには表示されません。(これは、`ndbinfo.nodes` テーブルの切断されたノードの省略に似ています。

`remote_address` は、`remote_node_id` カラムに ID が表示されないノードのホスト名またはアドレスです。このノードからの `bytes_sent` およびこのノードによる `bytes_received` はそれぞれ、接続が確立されてから、この接続を使用してノードによって送信されたバイト数と受信されたバイト数です。ステータスが `CONNECTING` または `DISCONNECTED` であるノードの場合、これらのカラムには常に `0` が表示されます。

`ndb_mgm` クライアントの `SHOW` コマンドの出力に示されるように、2 つのデータノード、2 つの SQL ノード、および 1 つの管理ノードで構成される 5 ノードクラスタがあると仮定します。

```
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
-----
[ndbd(NDB)] 2 node(s)
id=1 @10.100.10.1 (8.0.23-ndb-8.0.23, Nodegroup: 0, *)
id=2 @10.100.10.2 (8.0.23-ndb-8.0.23, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=10 @10.100.10.10 (8.0.23-ndb-8.0.23)

[mysqld(API)] 2 node(s)
id=20 @10.100.10.20 (8.0.23-ndb-8.0.23)
id=21 @10.100.10.21 (8.0.23-ndb-8.0.23)
```

すべてのデータノードが実行されていると仮定すると、次に示すように、`transporters` テーブルには 10 行 (1 番目のデータノード用に 5 行、2 番目のデータノード用に 5 行) が表示されます。

```
mysql> SELECT node_id, remote_node_id, status
-> FROM ndbinfo.transporters;
+-----+-----+-----+
| node_id | remote_node_id | status      |
+-----+-----+-----+
| 1 | 1 | DISCONNECTED |
| 1 | 2 | CONNECTED    |
| 1 | 10 | CONNECTED   |
| 1 | 20 | CONNECTED   |
| 1 | 21 | CONNECTED   |
| 2 | 1 | CONNECTED    |
| 2 | 2 | DISCONNECTED |
| 2 | 10 | CONNECTED   |
| 2 | 20 | CONNECTED   |
| 2 | 21 | CONNECTED   |
+-----+-----+-----+
10 rows in set (0.04 sec)
```

`ndb_mgm` クライアントで `2 STOP` コマンドを使用して、このクラスタ内のデータノードのいずれかをシャットダウンしてから、(再度 `mysql` クライアントを使用して) 前のクエリを繰り返すと、次に示すように、このテーブルには、5 行 (残りの管理ノードから別のノードへの接続 (そのデータノード自体と現在オフラインになっているデータノードの両方を含む) ごとに 1 行) のみが表示され、現在オフラインになっているデータノードへの残りの各接続のステータスを表す `CONNECTING` が表示されます。

```
mysql> SELECT node_id, remote_node_id, status
-> FROM ndbinfo.transporters;
+-----+-----+-----+
| node_id | remote_node_id | status      |
+-----+-----+-----+
| 1 | 1 | DISCONNECTED |
```

```

| 1 | 2 | CONNECTING |
| 1 | 10 | CONNECTED |
| 1 | 20 | CONNECTED |
| 1 | 21 | CONNECTED |
+-----+-----+-----+
5 rows in set (0.02 sec)

```

`connect_count`、`overloaded`、`overload_count`、`slowdown` および `slowdown_count` カウンタは接続時にリセットされ、リモートノードが切断された後も値が保持されます。`bytes_sent` および `bytes_received` カウンタも接続時にリセットされるため、切断後も値は保持されます (次の接続がリセットされるまで)。

`overloaded` カラムおよび `overload_count` カラムで参照される過負荷の状態は、このトランスポートの送信バッファに `OVERloadLimit` バイトを超える値が含まれている場合に発生します (デフォルトは `SendBufferMemory` の 80%、つまり $0.8 * 2097152 = 1677721$ バイト)。特定のトランスポートが過負荷状態にある場合、このトランスポートを使用しようとする新しいトランザクションはエラー 1218 (「NDB カーネルでオーバーロードされたバッファの送信」) で失敗します。これは、スキャンと主キー操作の両方に影響します。

このテーブルの `slowdown` カラムおよび `slowdown_count` カラムによって参照される低速の状態は、トランスポートの送信バッファにオーバーロード制限の 60% を超える値が含まれている場合に発生します (デフォルトでは $0.6 * 2097152 = 1258291$ バイトと等しくなります)。この状態では、このトランスポートを使用する新しいスキャンのバッチサイズが小さくなり、トランスポートの負荷が最小限に抑えられます。

送信バッファの速度低下またはオーバーロードの一般的な原因は、次のとおりです:

- データサイズ。特に、`TEXT` カラムまたは `BLOB` カラム (あるいはその両方のタイプのカラム) に格納されるデータの量
- バイナリロギングに参与する SQL ノードと同じホスト上にデータノード (ndbd または ndbmtd) がある
- トランザクションまたはトランザクションバッチ当たりの多数の行
- `SendBufferMemory` の不足などの構成の問題
- RAM 不足やネットワーク接続の低下などのハードウェアの問題

[セクション23.3.3.14「NDB Cluster 送信バッファパラメータの構成」](#) も参照してください。

23.5.15 NDB Cluster の INFORMATION_SCHEMA テーブル

2 つの `INFORMATION_SCHEMA` テーブルは、NDB Cluster を管理するときに特に使用される情報を提供します。`FILES` テーブルは NDB Cluster ディスクデータファイルに関する情報を提供します ([セクション23.5.10.1「NDB Cluster ディスクデータオブジェクト」](#) を参照)。`ndb_transid_mysql_connection_map` テーブルは、トランザクション、トランザクションコーディネータおよび API ノード間のマッピングを提供します。

NDB Cluster のトランザクション、操作、スレッド、ブロック、およびパフォーマンスのその他の側面に関する追加の統計およびその他のデータは、`ndbinfo` データベース内のテーブルから取得できます。これらのテーブルの詳細は、[セクション23.5.14「ndbinfo: NDB Cluster 情報データベース」](#) を参照してください。

23.5.16 クイックリファレンス: NDB Cluster SQL ステートメント

このセクションでは、NDB Cluster に接続されている MySQL サーバーの管理およびモニタリングに役立つことがあるいくつかの SQL ステートメントについて説明し、場合によってはクラスタ自体に関する情報を提供します。

- `SHOW ENGINE NDB STATUS`、`SHOW ENGINE NDBCLUSTER STATUS`

このステートメントの出力には、クラスタへのサーバー接続、NDB Cluster オブジェクトの作成と使用、NDB Cluster レプリケーションのバイナリロギングに関する情報が含まれています。

使用例および詳細な情報については、[セクション13.7.7.15「SHOW ENGINE ステートメント」](#) を参照してください。

- `SHOW ENGINES`

このステートメントを使用すると、MySQL サーバーでクラスタ化のサポートが有効になっているかどうか、およびその場合はアクティブであるかどうかを確認できます。

詳細は、[セクション13.7.7.16「SHOW ENGINES ステートメント」](#)を参照してください。

注記

このステートメントでは、[LIKE](#) 句がサポートされていません。ただし、次の項目で説明するように、[LIKE](#) を使用すれば、[INFORMATION_SCHEMA.ENGINES](#) テーブルに対するクエリーをフィルタ処理できます。

- [SELECT * FROM INFORMATION_SCHEMA.ENGINES \[WHERE ENGINE LIKE 'NDB%'\]](#)

これは、[SHOW ENGINES](#) と同等ですが、[INFORMATION_SCHEMA](#) の [ENGINES](#) テーブルを使用します。[SHOW ENGINES](#) ステートメントを使用する場合は異なり、[LIKE](#) 句を使用して結果をフィルタ処理したり、特定の列を選択してスクリプトで役立つ可能性のある情報を取得したりできます。たとえば、次のクエリーは、[NDB](#) サポートを使用してサーバーが構築されたかどうか、およびその場合は有効になっているかどうかを表示します。

```
mysql> SELECT ENGINE, SUPPORT FROM INFORMATION_SCHEMA.ENGINES
-> WHERE ENGINE LIKE 'NDB%';
+-----+-----+
| ENGINE | SUPPORT |
+-----+-----+
| ndbcluster | YES |
| ndbinfo | YES |
+-----+-----+
```

[NDB](#) サポートが有効になっていない場合、前述のクエリーは空のセットを返します。詳しくは[セクション26.13「INFORMATION_SCHEMA ENGINES テーブル」](#),をご覧ください。

- [SHOW VARIABLES LIKE 'NDB%'](#)

このステートメントは、次に示すように、[NDB](#) ストレージエンジンに関するほとんどのサーバーシステム変数、およびそれらの値のリストを表示します。

```
mysql> SHOW VARIABLES LIKE 'NDB%';
+-----+-----+-----+
| Variable_name | Value | |
+-----+-----+-----+
| ndb_allow_copying_alter_table | ON | |
| ndb_autoincrement_prefetch_sz | 512 | |
| ndb_batch_size | 32768 | |
| ndb_blob_read_batch_bytes | 65536 | |
| ndb_blob_write_batch_bytes | 65536 | |
| ndb_clear_apply_status | ON | |
| ndb_cluster_connection_pool | 1 | |
| ndb_cluster_connection_pool_nodeids | | |
| ndb_connectstring | 127.0.0.1 | |
| ndb_data_node_neighbour | 0 | |
| ndb_default_column_format | FIXED | |
| ndb_deferred_constraints | 0 | |
| ndb_distribution | KEYHASH | |
| ndb_eventbuffer_free_percent | 20 | |
| ndb_eventbuffer_max_alloc | 0 | |
| ndb_extra_logging | 1 | |
| ndb_force_send | ON | |
| ndb_fully_replicated | OFF | |
| ndb_index_stat_enable | ON | |
| ndb_index_stat_option | loop_enable=1000ms,loop_idle=1000ms,
loop_busy=100ms,update_batch=1,read_batch=4,idle_batch=32,check_batch=8,
check_delay=10m,delete_batch=8,clean_delay=1m,error_batch=4,error_delay=1m,
evict_batch=8,evict_delay=1m,cache_limit=32M,cache_lowpct=90,zero_total=0 | |
| ndb_join_pushdown | ON | |
| ndb_log_apply_status | OFF | |
| ndb_log_bin | OFF | |
| ndb_log_binlog_index | ON | |
| ndb_log_empty_epochs | OFF | |
| ndb_log_empty_update | OFF | |
| ndb_log_exclusive_reads | OFF | |
| ndb_log_orig | OFF | |
| ndb_log_transaction_id | OFF | |
| ndb_log_update_as_write | ON | |
| ndb_log_update_minimal | OFF | |
```

ndb_log_updated_only	ON	
ndb_metadata_check	ON	
ndb_metadata_check_interval	60	
ndb_metadata_sync	OFF	
ndb_mgmd_host	127.0.0.1	
ndb_nodeid	0	
ndb_optimization_delay	10	
ndb_optimized_node_selection	3	
ndb_read_backup	ON	
ndb_rcv_thread_activation_threshold	8	
ndb_rcv_thread_cpu_mask		
ndb_report_thresh_binlog_epoch_slip	10	
ndb_report_thresh_binlog_mem_usage	10	
ndb_row_checksum	1	
ndb_schema_dist_lock_wait_timeout	30	
ndb_schema_dist_timeout	120	
ndb_schema_dist_upgrade_allowed	ON	
ndb_show_foreign_key_mock_tables	OFF	
ndb_slave_conflict_role	NONE	
ndb_table_no_logging	OFF	
ndb_table_temporary	OFF	
ndb_use_copying_alter_table	OFF	
ndb_use_exact_count	OFF	
ndb_use_transactions	ON	
ndb_version	524308	
ndb_version_string	ndb-8.0.23	
ndb_wait_connected	30	
ndb_wait_setup	30	
ndbinfo_database	ndbinfo	
ndbinfo_max_bytes	0	
ndbinfo_max_rows	10	
ndbinfo_offline	OFF	
ndbinfo_show_hidden	OFF	
ndbinfo_table_prefix	ndb\$	
ndbinfo_version	524308	

詳細は、[セクション5.1.8「サーブシステム変数」](#)を参照してください。

- [SELECT * FROM performance_schema.global_variables WHERE VARIABLE_NAME LIKE 'NDB%'](#)

このステートメントは、前の項目で説明した [SHOW VARIABLES](#) ステートメントと同等で、次に示すようにほぼ同一の出力を提供します:

```
mysql> SELECT * FROM performance_schema.global_variables
-> WHERE VARIABLE_NAME LIKE 'NDB%';
+-----+-----+
| VARIABLE_NAME | VARIABLE_VALUE |
+-----+-----+
| ndb_allow_copying_alter_table | ON |
| ndb_autoincrement_prefetch_sz | 512 |
| ndb_batch_size | 32768 |
| ndb_blob_read_batch_bytes | 65536 |
| ndb_blob_write_batch_bytes | 65536 |
| ndb_clear_apply_status | ON |
| ndb_cluster_connection_pool | 1 |
| ndb_cluster_connection_pool_nodeids | |
| ndb_connectstring | 127.0.0.1 |
| ndb_data_node_neighbour | 0 |
| ndb_default_column_format | FIXED |
| ndb_deferred_constraints | 0 |
| ndb_distribution | KEYHASH |
| ndb_eventbuffer_free_percent | 20 |
| ndb_eventbuffer_max_alloc | 0 |
| ndb_extra_logging | 1 |
| ndb_force_send | ON |
| ndb_fully_replicated | OFF |
| ndb_index_stat_enable | ON |
| ndb_index_stat_option | loop_enable=1000ms,loop_idle=1000ms,
loop_busy=100ms,update_batch=1,read_batch=4,idle_batch=32,check_batch=8,
check_delay=10m,delete_batch=8,clean_delay=1m,error_batch=4,error_delay=1m,
evict_batch=8,evict_delay=1m,cache_limit=32M,cache_lowpct=90,zero_total=0 |
| ndb_join_pushdown | ON |
| ndb_log_apply_status | OFF |
```

```
|ndb_log_bin          |OFF          |
|ndb_log_binlog_index|ON           |
|ndb_log_empty_epochs|OFF         |
|ndb_log_empty_update|OFF         |
|ndb_log_exclusive_reads|OFF        |
|ndb_log_orig        |OFF         |
|ndb_log_transaction_id|OFF        |
|ndb_log_update_as_write|ON         |
|ndb_log_update_minimal|OFF        |
|ndb_log_updated_only |ON          |
|ndb_metadata_check   |ON          |
|ndb_metadata_check_interval|60       |
|ndb_metadata_sync    |OFF         |
|ndb_mgmd_host        |127.0.0.1  |
|ndb_nodeid           |0           |
|ndb_optimization_delay|10         |
|ndb_optimized_node_selection|3        |
|ndb_read_backup      |ON          |
|ndb_recv_thread_activation_threshold|8        |
|ndb_recv_thread_cpu_mask|          |
|ndb_report_thresh_binlog_epoch_slip|10       |
|ndb_report_thresh_binlog_mem_usage|10       |
|ndb_row_checksum     |1           |
|ndb_schema_dist_lock_wait_timeout|30       |
|ndb_schema_dist_timeout|120        |
|ndb_schema_dist_upgrade_allowed|ON        |
|ndb_show_foreign_key_mock_tables|OFF       |
|ndb_slave_conflict_role|NONE        |
|ndb_table_no_logging |OFF         |
|ndb_table_temporary  |OFF         |
|ndb_use_copying_alter_table|OFF        |
|ndb_use_exact_count  |OFF         |
|ndb_use_transactions |ON          |
|ndb_version          |524308     |
|ndb_version_string   |ndb-8.0.23|
|ndb_wait_connected  |30         |
|ndb_wait_setup       |30         |
|ndbinfo_database    |ndbinfo    |
|ndbinfo_max_bytes   |0           |
|ndbinfo_max_rows    |10         |
|ndbinfo_offline     |OFF         |
|ndbinfo_show_hidden |OFF         |
|ndbinfo_table_prefix|ndb$        |
|ndbinfo_version     |524308     |
+-----+-----+
```

SHOW VARIABLES ステートメントの場合とは異なり、個々のカラムを選択できます。例:

```
mysql> SELECT VARIABLE_VALUE
-> FROM performance_schema.global_variables
-> WHERE VARIABLE_NAME = 'ndb_force_send';
+-----+
|VARIABLE_VALUE|
+-----+
|ON             |
+-----+
```

より有用なクエリーを次に示します:

```
mysql> SELECT VARIABLE_NAME AS Name, VARIABLE_VALUE AS Value
> FROM performance_schema.global_variables
> WHERE VARIABLE_NAME
> IN ('version', 'ndb_version',
>     'ndb_version_string', 'ndbinfo_version');
+-----+-----+
|Name      |Value      |
+-----+-----+
|ndb_version      |524308     |
|ndb_version_string|ndb-8.0.20 |
|ndbinfo_version  |524308     |
|version         |8.0.20-cluster|
+-----+-----+
```

4 rows in set (0.00 sec)

詳細は、[セクション27.12.15「パフォーマンススキーマのステータス変数のテーブル」](#) および [セクション5.1.8「サーバーシステム変数」](#) を参照してください。

- `SHOW STATUS LIKE 'NDB%'`

このステートメントは、MySQL サーバーがクラスタ SQL ノードとして動作しているかどうかをひと目でわかるように表示し、その場合は次に示すように、MySQL サーバーのクラスタノード ID、接続先のクラスタ管理サーバーのホスト名とポート、およびクラスタ内のデータノードの数を表示します。

```
mysql> SHOW STATUS LIKE 'NDB%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Ndb_metadata_detected_count | 0 |
| Ndb_cluster_node_id | 100 |
| Ndb_config_from_host | 127.0.0.1 |
| Ndb_config_from_port | 1186 |
| Ndb_number_of_data_nodes | 2 |
| Ndb_number_of_ready_data_nodes | 2 |
| Ndb_connect_count | 0 |
| Ndb_execute_count | 0 |
| Ndb_scan_count | 0 |
| Ndb_pruned_scan_count | 0 |
| Ndb_schema_locks_count | 0 |
| Ndb_api_wait_exec_complete_count_session | 0 |
| Ndb_api_wait_scan_result_count_session | 0 |
| Ndb_api_wait_meta_request_count_session | 1 |
| Ndb_api_wait_nanos_count_session | 163446 |
| Ndb_api_bytes_sent_count_session | 60 |
| Ndb_api_bytes_received_count_session | 28 |
| Ndb_api_trans_start_count_session | 0 |
| Ndb_api_trans_commit_count_session | 0 |
| Ndb_api_trans_abort_count_session | 0 |
| Ndb_api_trans_close_count_session | 0 |
| Ndb_api_pk_op_count_session | 0 |
| Ndb_api_uk_op_count_session | 0 |
| Ndb_api_table_scan_count_session | 0 |
| Ndb_api_range_scan_count_session | 0 |
| Ndb_api_pruned_scan_count_session | 0 |
| Ndb_api_scan_batch_count_session | 0 |
| Ndb_api_read_row_count_session | 0 |
| Ndb_api_trans_local_read_row_count_session | 0 |
| Ndb_api_adaptive_send_forced_count_session | 0 |
| Ndb_api_adaptive_send_unforced_count_session | 0 |
| Ndb_api_adaptive_send_deferred_count_session | 0 |
| Ndb_trans_hint_count_session | 0 |
| Ndb_sorted_scan_count | 0 |
| Ndb_pushed_queries_defined | 0 |
| Ndb_pushed_queries_dropped | 0 |
| Ndb_pushed_queries_executed | 0 |
| Ndb_pushed_reads | 0 |
| Ndb_last_commit_epoch_server | 37632503447571 |
| Ndb_last_commit_epoch_session | 0 |
| Ndb_system_name | MC_20191126162038 |
| Ndb_api_event_data_count_injector | 0 |
| Ndb_api_event_nondata_count_injector | 0 |
| Ndb_api_event_bytes_count_injector | 0 |
| Ndb_api_wait_exec_complete_count_slave | 0 |
| Ndb_api_wait_scan_result_count_slave | 0 |
| Ndb_api_wait_meta_request_count_slave | 0 |
| Ndb_api_wait_nanos_count_slave | 0 |
| Ndb_api_bytes_sent_count_slave | 0 |
| Ndb_api_bytes_received_count_slave | 0 |
| Ndb_api_trans_start_count_slave | 0 |
| Ndb_api_trans_commit_count_slave | 0 |
| Ndb_api_trans_abort_count_slave | 0 |
| Ndb_api_trans_close_count_slave | 0 |
| Ndb_api_pk_op_count_slave | 0 |
| Ndb_api_uk_op_count_slave | 0 |
| Ndb_api_table_scan_count_slave | 0 |
| Ndb_api_range_scan_count_slave | 0 |
```

```

| Ndb_api_pruned_scan_count_slave | 0 |
| Ndb_api_scan_batch_count_slave | 0 |
| Ndb_api_read_row_count_slave | 0 |
| Ndb_api_trans_local_read_row_count_slave | 0 |
| Ndb_api_adaptive_send_forced_count_slave | 0 |
| Ndb_api_adaptive_send_unforced_count_slave | 0 |
| Ndb_api_adaptive_send_deferred_count_slave | 0 |
| Ndb_slave_max_replicated_epoch | 0 |
| Ndb_api_wait_exec_complete_count | 4 |
| Ndb_api_wait_scan_result_count | 7 |
| Ndb_api_wait_meta_request_count | 172 |
| Ndb_api_wait_nanos_count | 1083548094028 |
| Ndb_api_bytes_sent_count | 4640 |
| Ndb_api_bytes_received_count | 109356 |
| Ndb_api_trans_start_count | 4 |
| Ndb_api_trans_commit_count | 1 |
| Ndb_api_trans_abort_count | 1 |
| Ndb_api_trans_close_count | 4 |
| Ndb_api_pk_op_count | 2 |
| Ndb_api_uk_op_count | 0 |
| Ndb_api_table_scan_count | 1 |
| Ndb_api_range_scan_count | 1 |
| Ndb_api_pruned_scan_count | 0 |
| Ndb_api_scan_batch_count | 1 |
| Ndb_api_read_row_count | 3 |
| Ndb_api_trans_local_read_row_count | 2 |
| Ndb_api_adaptive_send_forced_count | 1 |
| Ndb_api_adaptive_send_unforced_count | 5 |
| Ndb_api_adaptive_send_deferred_count | 0 |
| Ndb_api_event_data_count | 0 |
| Ndb_api_event_nondata_count | 0 |
| Ndb_api_event_bytes_count | 0 |
| Ndb_metadata_excluded_count | 0 |
| Ndb_metadata_synced_count | 0 |
| Ndb_conflict_fn_max | 0 |
| Ndb_conflict_fn_old | 0 |
| Ndb_conflict_fn_max_del_win | 0 |
| Ndb_conflict_fn_epoch | 0 |
| Ndb_conflict_fn_epoch_trans | 0 |
| Ndb_conflict_fn_epoch2 | 0 |
| Ndb_conflict_fn_epoch2_trans | 0 |
| Ndb_conflict_trans_row_conflict_count | 0 |
| Ndb_conflict_trans_row_reject_count | 0 |
| Ndb_conflict_trans_reject_count | 0 |
| Ndb_conflict_trans_detect_iter_count | 0 |
| Ndb_conflict_trans_conflict_commit_count | 0 |
| Ndb_conflict_epoch_delete_delete_count | 0 |
| Ndb_conflict_reflected_op_prepare_count | 0 |
| Ndb_conflict_reflected_op_discard_count | 0 |
| Ndb_conflict_refresh_op_count | 0 |
| Ndb_conflict_last_conflict_epoch | 0 |
| Ndb_conflict_last_stable_epoch | 0 |
| Ndb_index_stat_status | allow:1,enable:1,busy:0,
loop:1000,list:(new:0,update:0,read:0,idle:0,check:0,delete:0,error:0,total:0),
analyze:(queue:0,wait:0),stats:(nostats:0,wait:0),total:(analyze:(all:0,error:0),
query:(all:0,nostats:0,error:0),event:(act:0,skip:0,miss:0),cache:(refresh:0,
clean:0,pinned:0,drop:0,evict:0)),cache:(query:0,clean:0,drop:0,evict:0,
usedpct:0.00,highpct:0.00)
| Ndb_index_stat_cache_query | 0 |
| Ndb_index_stat_cache_clean | 0 |
+-----+

```

MySQL サーバーが **NDB** サポートを使用して構築されているが、現在クラスタに接続されていない場合、このステートメントの出力の各行には、**Value** カラムのゼロまたは空の文字列が含まれます。

[セクション13.7.7.37 「SHOW STATUS ステートメント」](#) も参照してください。

- `SELECT * FROM performance_schema.global_status WHERE VARIABLE_NAME LIKE 'NDB%'`

このステートメントは、前の項目で説明した `SHOW STATUS` ステートメントと同様の出力を提供します。 `SHOW STATUS` の場合とは異なり、`SELECT` ステートメントを使用して SQL で値を抽出し、監視および自動化のためにスクリプトで使用できます。

詳細は、[セクション27.12.15「パフォーマンススキーマのステータス変数のテーブル」](#)を参照してください。

- `SELECT * FROM INFORMATION_SCHEMA.PLUGINS WHERE PLUGIN_NAME LIKE 'NDB%'`

このステートメントは、次に示すように、NDB Cluster に関連付けられたプラグイン (バージョン、作成者、ライセンスなど) に関する `INFORMATION_SCHEMA.PLUGINS` テーブルからの情報を表示します:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.PLUGINS
> WHERE PLUGIN_NAME LIKE 'NDB%\G
***** 1. row *****
  PLUGIN_NAME: ndbcluster
  PLUGIN_VERSION: 1.0
  PLUGIN_STATUS: ACTIVE
  PLUGIN_TYPE: STORAGE ENGINE
  PLUGIN_TYPE_VERSION: 80029.0
  PLUGIN_LIBRARY: NULL
  PLUGIN_LIBRARY_VERSION: NULL
  PLUGIN_AUTHOR: Oracle Corporation
  PLUGIN_DESCRIPTION: Clustered, fault-tolerant tables
  PLUGIN_LICENSE: GPL
  LOAD_OPTION: ON
***** 2. row *****
  PLUGIN_NAME: ndbinfo
  PLUGIN_VERSION: 0.1
  PLUGIN_STATUS: ACTIVE
  PLUGIN_TYPE: STORAGE ENGINE
  PLUGIN_TYPE_VERSION: 80029.0
  PLUGIN_LIBRARY: NULL
  PLUGIN_LIBRARY_VERSION: NULL
  PLUGIN_AUTHOR: Oracle Corporation
  PLUGIN_DESCRIPTION: MySQL Cluster system information storage engine
  PLUGIN_LICENSE: GPL
  LOAD_OPTION: ON
***** 3. row *****
  PLUGIN_NAME: ndb_transid_mysql_connection_map
  PLUGIN_VERSION: 0.1
  PLUGIN_STATUS: ACTIVE
  PLUGIN_TYPE: INFORMATION SCHEMA
  PLUGIN_TYPE_VERSION: 80029.0
  PLUGIN_LIBRARY: NULL
  PLUGIN_LIBRARY_VERSION: NULL
  PLUGIN_AUTHOR: Oracle Corporation
  PLUGIN_DESCRIPTION: Map between MySQL connection ID and NDB transaction ID
  PLUGIN_LICENSE: GPL
  LOAD_OPTION: ON
```

`SHOW PLUGINS` ステートメントを使用してこの情報を表示することもできますが、そのステートメントからの出力は簡単にフィルタできません。 `PLUGINS` テーブルの情報を取得する場所と方法について説明している [The MySQL Plugin API](#) も参照してください。

`ndbinfo` 情報データベース内のテーブルをクエリーして、多数の NDB Cluster 操作に関するリアルタイムデータを取得することもできます。 [セクション23.5.14「ndbinfo: NDB Cluster 情報データベース」](#)を参照してください。

23.5.17 NDB Cluster のセキュリティーの問題

このセクションでは、NDB Cluster を設定および実行するときに考慮すべきセキュリティー上の考慮事項について説明します。

このセクションで説明するトピックは、次のとおりです。

- NDB Cluster およびネットワークのセキュリティーの問題
- NDB Cluster のセキュアな実行に関連する構成の問題

- NDB Cluster および MySQL 特権システム
- NDB Cluster に適用可能な MySQL 標準セキュリティ手順

23.5.17.1 NDB Cluster のセキュリティおよびネットワークの問題

このセクションでは、NDB Cluster に関連する基本的なネットワークセキュリティの問題について説明します。NDB Cluster 「即時利用可能」はセキュリティ保護されていないことを覚えておくことが非常に重要です。ユーザーまたはネットワーク管理者は、クラスタがネットワーク上で危険にさらされないように適切な手順を実行する必要があります。

クラスタの通信プロトコルは本質的にセキュアではなく、クラスタ内のノード間の通信では暗号化や同様のセキュリティ対策が使用されません。ネットワーク速度と待機時間はクラスタの効率に直接影響するため、ノード間のネットワーク接続に SSL またはその他の暗号化を使用することもお薦めしません。このようなスキームを使用すると、通信が遅くなります。

NDB Cluster への API ノードアクセスの制御に認証が使用されないことも当てはまります。暗号化と同様に、認証の要件を課すオーバーヘッドによって、クラスタのパフォーマンスが影響を受けることがあります。

さらに、クラスタへのアクセス時に、次のいずれに対するソース IP アドレスのチェックも実行されません。

- `config.ini` ファイル内の空の `[mysqld]` または `[api]` セクションによって作成される「空きスロット」を使用している SQL ノードまたは API ノード

つまり、`config.ini` ファイルに空の `[mysqld]` または `[api]` セクションがある場合、管理サーバーのホスト名 (または IP アドレス) とポートを認識する API ノード (SQL ノードを含む) は、制限なしでクラスタに接続し、そのデータにアクセスできます。(このことおよび関連する問題についての詳細は、[セクション 23.5.17.2 「NDB Cluster および MySQL の権限」](#) を参照してください。)

注記

`config.ini` ファイルのすべての `[mysqld]` および `[api]` セクションに `HostName` パラメータを指定することで、クラスタへの SQL および API ノードアクセスをある程度制御できます。ただし、これは、これまで使用されていないホストからクラスタに API ノードを接続する場合、そのホスト名を含む `[api]` セクションを `config.ini` ファイルに追加する必要があります。

詳細は、`HostName` パラメータについてのこの章のほかの場所で説明しています。API ノードで `HostName` を使用する構成例について、[セクション 23.3.1 「NDB Cluster のクイックテスト設定」](#) も参照してください。

- 任意の `ndb_mgm` クライアント

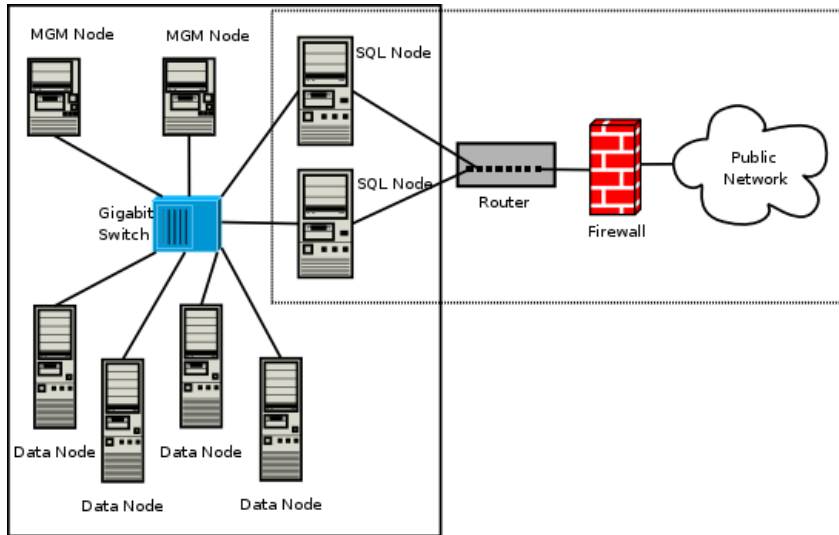
つまり、管理サーバーのホスト名 (または IP アドレス) とポート (標準ポートでない場合) が指定されたクラスタ管理クライアントは、クラスタに接続して任意の管理クライアントコマンドを実行できます。これには、`ALL STOP` や `SHUTDOWN` などのコマンドが含まれます。

これらの理由から、ネットワークレベルでクラスタを保護する必要があります。もっとも安全なクラスタのネットワーク構成は、クラスタノード間の接続をその他のネットワーク通信から分離する構成です。これは、次の方法のいずれかによって実現できます。

1. どのパブリックネットワークからも物理的に分離されたネットワーク上にクラスタノードを保持します。このオプションは、信頼性が高いために、実装するコストも高くなります。

このような物理的に分離されたネットワークを使用した NDB Cluster 設定の例を次に示します:

図 23.28 ハードウェアファイアウォールを備えた NDB Cluster



この設定には、クラスタ管理サーバーおよびデータノードの1つのプライベート (実線の四角形) と、SQL ノードが存在する1つのパブリック (点線の四角形) の2つのネットワークがあります。(ギガビットスイッチは最高のパフォーマンスが実現されるため、これを使用して接続された管理ノードとデータノードを示しています。) どちらのネットワークも、ハードウェアファイアウォール (ネットワークベースのファイアウォールと呼ばれることもあります) によって外部から保護されています。

SQL ノードでパケットの転送が許可されていないと、パケットは SQL ノードを通過せずに、ネットワーク外部からクラスタの管理ノードまたはデータノードに到達できない (どのクラスタの内部通信も外部に到達できない) ため、このネットワーク設定がもっとも安全です。これは、当然、すべての SQL ノードをハッキングの試みに対して、セキュリティー保護する必要があることを意味します。

重要

潜在的なセキュリティーの脆弱性に関しては、SQL ノードとその他の MySQL サーバーとの間に相違はありません。MySQL サーバーをセキュリティー保護するために使用できる技法については、[セクション6.1.3「攻撃者に対する MySQL のセキュアな状態の維持」](#)を参照してください。

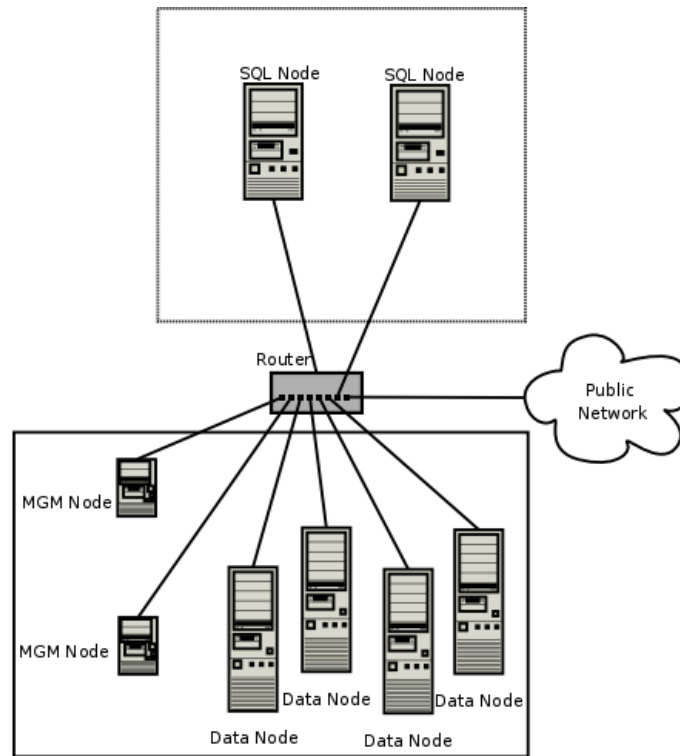
2. 1つ以上のソフトウェアファイアウォール (ホストベースのファイアウォールとも呼ばれる) を使用して、ネットワークのアクセスを必要としない部分からクラスタに通過するパケットを制御します。このタイプの設定では、

それがなければローカルネットワークの外部からアクセスできてしまう可能性のあるクラスタ内のすべてのホストに、ソフトウェアファイアウォールをインストールする必要があります。

ホストベースのオプションは、実装するコストがもっとも低いですが、保護を提供するソフトウェアに完全に依存しているため、セキュアな状態を維持することがもっとも困難です。

NDB Cluster のこのタイプのネットワーク設定を次に示します:

図 23.29 ソフトウェアファイアウォールを備えた NDB Cluster



このタイプのネットワーク設定を使用することは、NDB Cluster ホストに 2 つのゾーンがあることを意味します。各クラスタホストは、クラスタ内の他のすべてのマシンと通信する必要がありますが、SQL ノードをホストしているマシン (点線の四角形) のみ、外部との接続を許可でき、データノードおよび管理ノードを含むゾーン内のマシン (実線の四角形) は、クラスタに含まれないすべてのマシンから分離する必要があります。クラスタを使用するアプリケーションとそれらのアプリケーションのユーザーには、管理ノードおよびデータノードホストへの直接アクセスを許可しないでください。

これを実現するには、各クラスタホストコンピュータ上で実行されているノードのタイプに応じて、次の表に示す 1 つまたは複数のタイプにトラフィックを制限するソフトウェアファイアウォールを設定する必要があります。

表 23.66 ホストベースのファイアウォールクラスタ構成のノードタイプ

ノードタイプ	許可されるトラフィック
SQL ノードまたは API ノード	<ul style="list-style-type: none">それは、管理ノードまたはデータノードの IP アドレスから (任意の TCP または UDP ポートを使用して) 発信されています。それは、クラスタが存在し、アプリケーションで使用されているポート上にあるネットワーク内から発信されています。
データノードまたは管理ノード	<ul style="list-style-type: none">それは、管理ノードまたはデータノードの IP アドレスから (任意の TCP または UDP ポートを使用して) 発信されています。

ノードタイプ	許可されるトラフィック
	• それは、SQL ノードまたは API ノードの IP アドレスから発信されています。

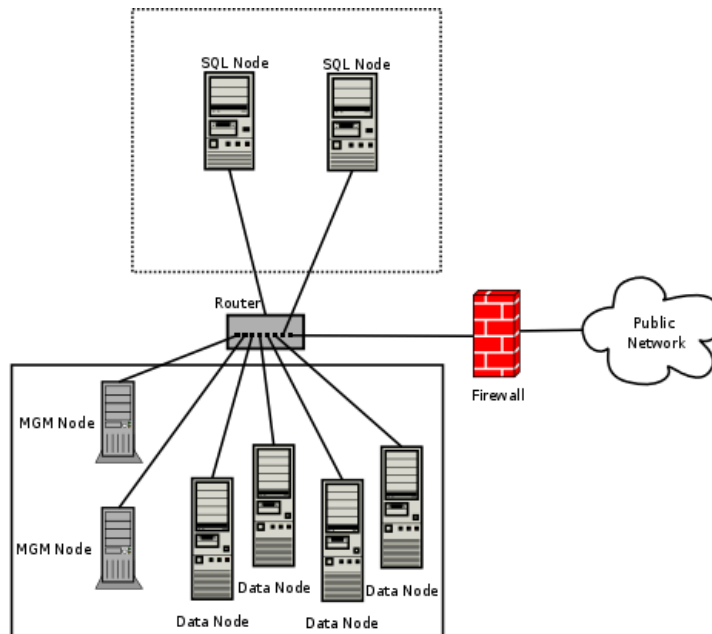
特定のノードタイプの、表に示した以外のトラフィックはすべて拒否されるべきです。

ファイアウォール構成の仕様は、ファイアウォールアプリケーションごとに異なるため、このマニュアルのスコプ外です。iptables は非常に一般的で、信頼性の高いファイアウォールアプリケーションであるため、多くの場合に構成をより簡単にするためにフロントエンドとして APF とともに使用されます。採用するソフトウェアファイアウォール、このタイプの NDB Cluster ネットワーク設定の実装を選択した場合、または次の項目で説明する「mixed」タイプのドキュメントを参照してください。

3. ハードウェアとソフトウェアの両方を使用してクラスタを保護する、つまり、ネットワークベースとホストベースの両方のファイアウォールを使用して、最初の 2 つの方法を組み合わせて使用することもできます。これは、セキュリティーレベルとコストの両方の点で、最初の 2 つのスキームの中間に位置するものです。このタイプのネットワーク設定では、クラスタがハードウェアファイアウォールの背後に配置されますが、受信パケットは SQL ノードに到達するために、すべてのクラスタホストに接続しているルータを通過することが許可されます。

ハードウェアファイアウォールとソフトウェアファイアウォールを組み合わせで使用した NDB Cluster の可能なネットワーク配備の 1 つを次に示します:

図 23.30 ハードウェアファイアウォールとソフトウェアファイアウォールの組み合わせを使用した NDB Cluster



この場合、SQL ノードおよび API ノードへのトラフィックを除くすべての外部トラフィックを拒否し、アプリケーションで必要とするポート上でのみそれらのノードへのトラフィックを許可するように、ハードウェアファイアウォールにルールを設定できます。

どのネットワーク構成を使用する場合でも、クラスタのセキュリティーを維持するという観点からの目標は同じであることは忘れないでください。つまり、クラスタ内のノード間でもっとも効率的な通信を確保しながら、不要なトラフィックがクラスタに到達することを妨げます。

NDB Cluster では、ノード間の通信用に多数のポートを開く必要があるため、分離されたネットワークを使用することをお勧めします。これは、不要なトラフィックがクラスタに到達することを防ぐためのもっとも簡単な方法です。

注記

NDB Cluster をリモートで (つまり、ローカルネットワーク外から) 管理する場合は、`ssh` または別のセキュアログインシェルを使用して SQL ノードホストにアクセスすることをお勧めします。このホストから、管理クライアントを実行して、クラスタ独自のローカルネットワーク内から安全に管理サーバーにアクセスできます。

`ndb_mgm` を使用して、クラスタが実行されているローカルネットワーク外部から直接クラスタを管理することは理論上可能ですが、推奨されていません。管理クライアントと管理サーバー間では認証も暗号化も行われなため、これはクラスタを管理するきわめてセキュアでない方法であり、遅かれ早かれ、ほぼ確実に危険にさらされます。

23.5.17.2 NDB Cluster および MySQL の権限

このセクションでは、NDB Cluster に関連して MySQL 特権システムがどのように機能するか、および NDB Cluster をセキュアに保つためのこれの影響について説明します。

標準 MySQL 権限は、「NDB Cluster」テーブルに適用されます。これには、データベース、テーブル、およびカラムのレベルで付与されるすべての MySQL 権限タイプ (`SELECT` 権限、`UPDATE` 権限、`DELETE` 権限など) が含まれます。その他の MySQL サーバーの場合と同様に、ユーザーおよび権限の情報は `mysql` システムデータベースに格納されます。NDB テーブル、このようなテーブルを含むデータベース、およびこのようなテーブル内のカラムに対する権限を付与および取り消すために使用される SQL ステートメントは、(その他の) MySQL ストレージエンジンを含むデータベースオブジェクトとの接続に使用される `GRANT` および `REVOKE` ステートメントとすべての点で同じです。`CREATE USER` および `DROP USER` ステートメントについても、同じことが当てはまります。

デフォルトでは、MySQL 付与テーブルは `MyISAM` ストレージエンジンを使用することに注意してください。このため、これらのテーブルは通常、NDB Cluster 内の SQL ノードとして機能する MySQL サーバー間で複製または共有されません。言い換えると、デフォルトでは、ユーザーおよびそれらの権限を変更しても自動的に SQL ノード間に反映されません。必要に応じて、NDB Cluster SQL ノード間での MySQL ユーザーおよび特権の自動配布を有効にできます。詳細は、[セクション 23.5.12 「NDB_STORED_USER での分散 MySQL 権限」](#) を参照してください。

逆に、MySQL には権限を拒否する方法がないため (権限は最初の場所で取り消すことも付与しないこともできますが、拒否することもできません)、ある SQL ノード上の NDB テーブルに対する特別な保護はありません (これは、ユーザー権限の自動配布を使用していない場合でも当てはまります)。これを示す明確な例として、任意のデータベースオブジェクトに対して任意のアクションを実行できる MySQL の `root` アカウントが挙げられます。このアカウントを `config.ini` ファイルの空の `[mysqld]` または `[api]` セクションと組み合わせると、著しく危険になる可能性があります。その理由を理解するために、次のようなシナリオを検討します。

- `config.ini` ファイルには、少なくとも 1 つの空の `[mysqld]` または `[api]` セクションが含まれています。つまり、NDB Cluster 管理サーバーは、MySQL Server (またはほかの API ノード) が NDB Cluster にアクセスするホストのチェックを実行しません。
- ファイアウォールが存在しないか、ファイアウォールが NDB Cluster へのアクセスをネットワーク外部のホストから保護できません。
- NDB Cluster 管理サーバーのホスト名または IP アドレスは既知であるか、ネットワークの外部から決定できます。

これらの条件が `true` の場合、だれでも `--ndbcluster --ndb-connectstring=management_host` を使用して MySQL Server を起動し、この NDB Cluster にアクセスできます。MySQL `root` アカウントを使用すると、このユーザーは次のアクションを実行できます。

- `SHOW DATABASES` ステートメント (サーバー上のすべての NDB データベースのリストを取得する) または `SHOW TABLES FROM some_ndb_database` ステートメントなどのメタデータステートメントを実行して、特定のデータベース内のすべての NDB テーブルのリストを取得します。
- 検出された任意のテーブルに対して、次のような有効な MySQL ステートメントを実行します:
 - 任意のテーブルからすべてのデータを読み取る `SELECT * FROM some_table`
 - テーブルからすべてのデータを削除する `DELETE FROM some_table`
 - テーブルスキーマを確認する `DESCRIBE some_table` または `SHOW CREATE TABLE some_table`

- テーブルカラムに「不正」データを入力する `UPDATE some_table SET column1 = some_value`。これは実際に、単にすべてのデータを削除するよりも、はるかに大きな損害が発生する可能性があります

より狡猾なバリエーションには、次のようなステートメントが含まれることがあります。

```
UPDATE some_table SET an_int_column = an_int_column + 1
```

または

```
UPDATE some_table SET a_varchar_column = REVERSE(a_varchar_column)
```

このような悪意のあるステートメントは、攻撃者の想像力でしか制限されません。

このような種類の攻撃を受けるおそれのない唯一のテーブルは、**NDB** 以外のストレージエンジンを使用して作成され、そのために「不正な」SQL ノードから見えないテーブルです。

`root` としてログインできるユーザーは、**INFORMATION_SCHEMA** データベースとそのテーブルにもアクセスできるため、データベース、テーブル、ストアドルーチン、スケジュール済イベント、およびメタデータが **INFORMATION_SCHEMA** に格納されているその他のデータベースオブジェクトに関する情報を取得できます。

また、分散特権を使用していないかぎり、異なる NDB Cluster SQL ノード上の `root` アカウントに異なるパスワードを使用することをお勧めします。

つまり、ローカルネットワークの外部から直接アクセスできる場合は、安全な NDB Cluster を持つことはできません。

重要

MySQL の `root` アカウントのパスワードは空のままにしないでください。これは、NDB Cluster SQL ノードとして MySQL を実行している場合と同様に、スタンドアロン (非クラスタ) MySQL Server として実行している場合にも当てはまり、NDB Cluster で MySQL Server を SQL ノードとして構成する前に、MySQL インストールプロセスの一部として実行するようしてください。

NDB Cluster 分散特権機能を採用する場合は、**NDB** ストレージエンジンを手動で使用するよう `mysql` データベース内のシステムテーブルを変換するだけではありません。代わりに、この目的のために提供されているストアドプロシージャを使用してください。セクション 23.5.12 「**NDB_STORED_USER** での分散 MySQL 権限」を参照してください。

それ以外の場合、SQL ノード間で `mysql` システムテーブルを同期する必要がある場合は、標準の MySQL レプリケーションを使用してこれを実行することも、スクリプトを使用して MySQL サーバー間でテーブルエントリをコピーすることもできます。

サマリー. NDB Cluster に関する MySQL 特権システムについて覚えておくべきもっとも重要な点を次に示します:

1. ある SQL ノード上で確立されたユーザーおよび権限は、クラスタ内のその他の SQL ノードで自動的に存在することも、有効になることもありません。逆に、クラスタ内のある SQL ノード上のユーザーまたは権限を削除しても、その他の SQL ノードからユーザーまたは権限は削除されません。
2. NDB Cluster 配布でこの目的のために提供されている SQL スクリプトとそれに含まれているストアドプロシージャを使用して、MySQL ユーザーおよび権限を SQL ノード間で配布できます。
3. NDB Cluster 内のある SQL ノードから **NDB** テーブルに対する権限が MySQL ユーザーに付与されると、そのユーザーは、権限配布を使用していない場合でも、データの元となった SQL ノードに関係なく、そのテーブル内の任意のデータを「参照」できます。

23.5.17.3 NDB Cluster および MySQL セキュリティー手順

このセクションでは、NDB Cluster の実行に適用される MySQL 標準セキュリティ手順について説明します。

一般に、MySQL をセキュアに実行するための標準の手順は、NDB Cluster の一部として MySQL Server を実行する場合にも適用されます。まず、MySQL Server は常に `mysql` オペレーティングシステムユーザーとして実行する必要が

あります。これは、標準 (非クラスタ) 環境での MySQL の実行とは異なりません。mysql システムアカウントは、一意かつ明確に定義されるべきです。幸運にも、これは新しい MySQL インストールのデフォルトの動作です。次に示すようなシステムコマンドを使用して、mysqld プロセスが mysql オペレーティングシステムユーザーとして実行されていることを確認できます:

```
shell> ps aux | grep mysql
root  10467 0.0 0.1 3616 1380 pts/3  S  11:53  0:00 \
/bin/sh ./mysqld_safe --ndbcluster --ndb-connectstring=localhost:1186
mysql  10512 0.2 2.5 58528 26636 pts/3  Sl  11:53  0:00 \
/usr/local/mysql/libexec/mysqld --basedir=/usr/local/mysql \
--datadir=/usr/local/mysql/var --user=mysql --ndbcluster \
--ndb-connectstring=localhost:1186 --pid-file=/usr/local/mysql/var/mothra.pid \
--log-error=/usr/local/mysql/var/mothra.err
jon   10579 0.0 0.0 2736 688 pts/0   S+  11:54  0:00 grep mysql
```

mysqld プロセスが mysql 以外のユーザーとして実行されている場合は、それをすぐにシャットダウンしてから、mysql ユーザーとして再起動するようにしてください。このユーザーがシステムに存在しない場合は、mysql ユーザーアカウントを作成し、このユーザーを mysql ユーザーグループの一部にする必要があります。この場合は、このシステム上の MySQL データディレクトリ (mysqld の --datadir オプションを使用して設定) が mysql ユーザーによって所有されていること、および SQL ノードの my.cnf ファイルの [mysqld] セクションに user=mysql が含まれていることも確認する必要があります。また、コマンド行で --user=mysql を付けて MySQL サーバプロセスを起動できますが、コマンド行オプションを使用することを忘れたために、意図せずに mysqld が別のユーザーとして実行されている可能性もあるため、my.cnf オプションを使用することをお勧めします。mysqld_safe 起動スクリプトを使用すると、MySQL が強制的に mysql ユーザーとして実行されます。

重要

絶対に mysqld をシステムの root ユーザーとして実行しないでください。これを行うと、システム上の任意のファイルが MySQL によって読み取られる可能性があるため、攻撃者によって MySQL が危険にさらされます。

前のセクション (セクション 23.5.17.2 「NDB Cluster および MySQL の権限」を参照) で説明したように、MySQL Server を実行したらすぐに root パスワードを設定する必要があります。また、デフォルトでインストールされている匿名ユーザーアカウントを削除するようにしてください。次のステートメントを使用すると、これらのタスクを実現できます。

```
shell> mysql -u root

mysql> UPDATE mysql.user
-> SET Password=PASSWORD('secure_password')
-> WHERE User='root';

mysql> DELETE FROM mysql.user
-> WHERE User=";

mysql> FLUSH PRIVILEGES;
```

DELETE ステートメントを実行する際は、WHERE 句を省略しないように注意してください。そうしないと、すべての MySQL ユーザーが削除される危険性があります。mysql.user テーブルを変更したら、その変更が即座に有効になるように、すぐに FLUSH PRIVILEGES ステートメントを実行してください。FLUSH PRIVILEGES を実行しなければ、次回サーバーを再起動するまで、変更が有効になりません。

注記

ndb_show_tables、ndb_desc、ndb_select_all などの NDB Cluster ユーティリティの多くも認証なしで動作し、テーブル名、スキーマ、およびデータを表示できます。デフォルトで、これらは Unix スタイルのシステムにアクセス権 wxr-xr-x (755) でインストールされます。これは、mysql/bin ディレクトリにアクセスできる任意のユーザーが実行できることを意味します。

これらのユーティリティについての詳細は、セクション 23.4 「NDB Cluster プログラム」を参照してください。

23.6 NDB Cluster レプリケーション

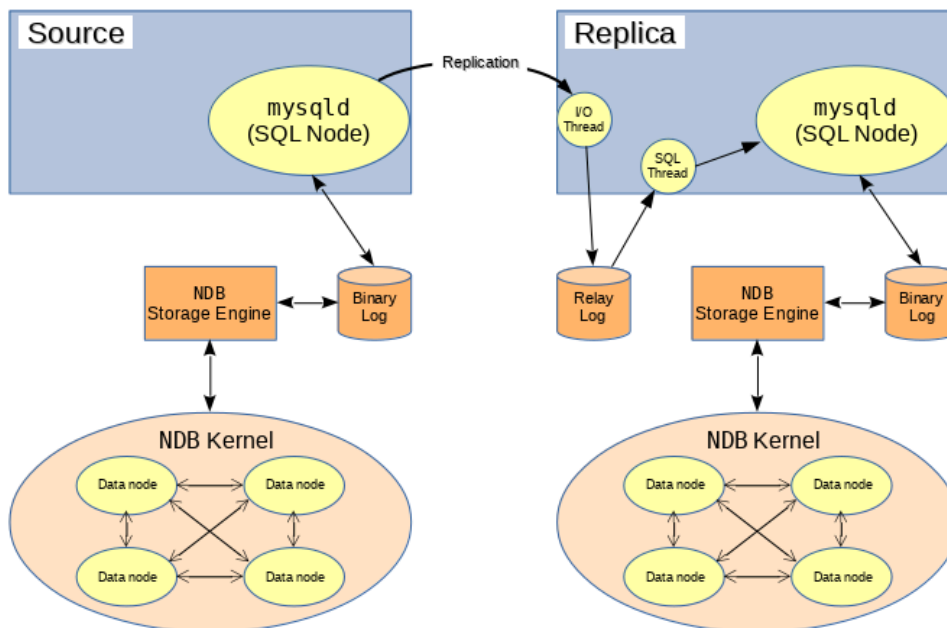
NDB Cluster は非同期レプリケーションをサポートしており、通常は単に「レプリケーション」と呼ばれます。このセクションでは、NDB Cluster として動作しているコンピュータの 1 つのグループが 2 つ目のコンピュータまたはコンピュータのグループにレプリケートする構成を設定および管理する方法について説明します。標準の MySQL レプリケーションに関してはこのマニュアルで別途説明しています。(第17章「レプリケーション」を参照してください。)

注記

NDB Cluster は GTID を使用したレプリケーションをサポートしていません。準同期レプリケーションおよびグループレプリケーションも NDB ストレージエンジンでサポートされていません。

通常の (クラスタ化されていない) レプリケーションには、ソースサーバーとレプリカサーバー、レプリケートされる操作とデータの発生源であるために名前が付けられているソース、およびレプリカがこれらの受信者になります。NDB Cluster では、レプリケーションは概念的に非常に似ていますが、2 つの完全なクラスタ間のレプリケーションを含む多数の異なる構成に対応するように拡張できるため、実際には複雑になることがあります。NDB Cluster 自体はクラスタリング機能のために NDB ストレージエンジンに依存しますが、レプリケートされたテーブルのレプリカコピーのストレージエンジンとして NDB を使用する必要はありません (NDB から別のストレージエンジンへのレプリケーションを参照)。ただし、可用性を最大限に高めるために、NDB Cluster 間でレプリケートすることが可能であり、次の図に示すように、このシナリオについて説明します:

図 23.31 NDB Cluster 間レプリケーションのレイアウト



このシナリオでは、レプリケーションプロセスは、ソースクラスタの連続した状態がログに記録され、レプリカクラスタに保存されるプロセスです。このプロセスは NDB バイナリログインジェクタスレッドとして知られる特別なスレッドによって実行されます。このスレッドは各 MySQL サーバーで動作し、バイナリログ (binlog) を作成します。このスレッドは、バイナリログを作成するクラスタのすべての変更 (MySQL Server を介して影響を受けた変更だけではありません) を、シリアライゼーションの正しい順序でバイナリログに挿入します。MySQL ソースサーバーとレプリカサーバーをレプリケーションサーバーまたはレプリケーションノードと呼び、それらの間のデータフローまたは通信回線をレプリケーションチャンネルと呼びます。

NDB Cluster および NDB Cluster レプリケーションを使用したポイントインタイムリカバリの実行については、セクション 23.6.9.2 「NDB Cluster レプリケーションを使用したポイントインタイムリカバリ」を参照してください。

NDB API レプリカステータス変数. NDB API カウンタは、レプリカクラスタで拡張モニタリング機能を提供できます。これらのカウンタは、[SHOW STATUS](#) の出力に表示される NDB 統計 `_slave` ステータス変数として、または NDB Cluster レプリケーションでレプリカとして機能している MySQL Server に接続されている `mysql` クライアントセッションのパフォーマンススキーマ `session_status` または `global_status` テーブルに対するクエリーの結果として実装されます。レプリケートされた NDB テーブルに影響を与えるステートメントの実行前後にこれらのステータス変数の値を比較することで、NDB Cluster のレプリケーションをモニタリングまたはトラブルシューティングするときに役立つ可能性のある、NDB API レベルで実行された対応するアクションをレプリカで監視できます。[セクション 23.5.13 「NDB API 統計のカウンタと変数」](#) では、追加情報が提供されます。

NDB テーブルから非 NDB テーブルへのレプリケーション. レプリケーションソースとして機能する NDB Cluster から、複製 `mysqld` 上のほかの MySQL ストレージエンジン (`InnoDB` や `MyISAM` など) を使用してテーブルに NDB テーブルを複製できます。これには多くの条件が適用されます。詳細については、[NDB から別のストレージエンジンへのレプリケーション](#) および [NDB から非トランザクションストレージエンジンへのレプリケーション](#) を参照してください。

23.6.1 NDB Cluster レプリケーション: 省略形と記号

このセクションでは、ソースクラスタとレプリカクラスタ、およびクラスタまたはクラスタノードで実行されるプロセスとコマンドを参照するために、次の省略形または記号を使用します:

表 23.67 このセクションでは、ソースクラスタとレプリカクラスタ、およびクラスタノードで実行されるプロセスとコマンドの略称を示します

記号または略語	説明 (... を参照)
S	(プライマリ) レプリケーションソースとして機能するクラスタ
R	(プライマリ) レプリカとして機能するクラスタ
shellS>	ソースクラスタで発行されるシェルコマンド
mysqlS>	ソースクラスタで SQL ノードとして実行されている単一の MySQL サーバーで発行された MySQL クライアントコマンド
mysqlS*>	レプリケーションソースクラスタに参加しているすべての SQL ノードで発行される MySQL クライアントコマンド
shellR>	レプリカクラスタで発行されるシェルコマンド
mysqlR>	レプリカクラスタで SQL ノードとして実行されている単一の MySQL サーバーで発行された MySQL クライアントコマンド
mysqlR*>	レプリカクラスタに参加しているすべての SQL ノードで発行される MySQL クライアントコマンド
C	プライマリレプリケーションチャンネル
C'	セカンダリレプリケーションチャンネル
S'	セカンダリレプリケーションソース
R'	セカンダリレプリカ

23.6.2 NDB Cluster レプリケーションの一般的な要件

レプリケーションチャンネルには、レプリケーションサーバーとして機能する 2 つの MySQL サーバーが必要です (それぞれがソースおよびレプリカ用です)。たとえば、2 つのレプリケーションチャンネルを持つレプリケーション設定の場合 (冗長性のための追加チャンネルを提供するため)、合計 4 つのレプリケーションノードがクラスタごとに 2 つあります。

このセクションで説明されている NDB Cluster のレプリケーションと、次のレプリケーションは行ベースレプリケーションに依存します。これは、[セクション 23.6.6 「NDB Cluster レプリケーションの開始 \(シングルレプリケーション](#)

チャンネル)」で説明されているように、レプリケーションソースの MySQL サーバーが `--binlog-format=ROW` または `--binlog-format=MIXED` で実行されている必要があることを意味します。行ベースのレプリケーションの一般的な情報は、[セクション17.2.1「レプリケーション形式」](#)を参照してください。

重要

NDB Cluster レプリケーションを `--binlog-format=STATEMENT` で使用しようとする、ソースクラスタ上の `ndb_binlog_index` テーブルおよびレプリカクラスタ上の `ndb_apply_status` テーブルの `epoch` カラムが更新されないため、レプリケーションは正しく機能しません ([セクション23.6.4「NDB Cluster レプリケーションスキーマおよびテーブル」](#)を参照)。かわりに、レプリケーションソースとして機能する MySQL サーバー上の更新のみがレプリカに伝播され、ソースクラスタ内の他の SQL ノードからの更新はレプリケートされません。

`--binlog-format` オプションのデフォルト値は `MIXED` です。

いずれかのクラスタでレプリケーションに使用される各 MySQL サーバーは、いずれかのクラスタに参加しているすべての MySQL レプリケーションサーバー間で一意に識別される必要があります (ソースクラスタとレプリカクラスタの両方で同じ ID を共有するレプリケーションサーバーを持つことはできません)。これは、`--server-id=id` オプションを使用して各 SQL ノードを起動して、実行できます。ここで、`id` は一意の整数です。厳密には必要ありませんが、この説明では、すべての NDB Cluster バイナリが同じリリースバージョンであることを前提としています。

MySQL レプリケーションでは通常、使用するレプリケーションプロトコルおよびサーバーがサポートする SQL 機能セットの両方のバージョンに両方の MySQL サーバー (`mysqld` プロセス) に互換性がある必要があります ([セクション17.5.2「MySQL バージョン間のレプリケーション互換性」](#)を参照してください)。NDB Cluster と MySQL Server 8.0 デイストリビューションのバイナリの違いにより、NDB Cluster Replication には、両方の `mysqld` バイナリが NDB Cluster デイストリビューションからのものであるという追加要件があります。`mysqld` サーバーに互換性があることを保証するもっとも簡単に簡単な方法は、すべてのソースおよびレプリカ `mysqld` バイナリに同じ NDB Cluster 配布を使用することです。

レプリカサーバーまたはクラスタがソースクラスタのレプリケーション専用であり、他のデータが格納されていないことを前提としています。

レプリケートされるすべての NDB テーブルは、MySQL サーバーおよびクライアントを使用して作成する必要があります。NDB API を使用して (たとえば、`Dictionary::createTable()` を使用して) 作成されたテーブルおよびその他のデータベースオブジェクトは、MySQL サーバーに表示されないため、レプリケートされません。NDB API アプリケーションによる、MySQL サーバーを使用して作成された既存のテーブルへの更新はレプリケートできます。

注記

ステートメントベースレプリケーションを使用して NDB Cluster をレプリケートできます。ただし、この場合、次の制限が適用されます。

- ソースとして機能するクラスタ上のデータ行に対するすべての更新は、単一の MySQL サーバーに転送する必要があります。
- 複数の同時 MySQL レプリケーションプロセスを使用してクラスタを複製することはできません。
- SQL レベルで行われた変更のみが複製されます。

これらは、行ベースのレプリケーションと対照的に、ステートメントベースのレプリケーションのその他の制限事項に追加されます。2つのレプリケーション形式の間の違いに関する詳細な情報については、[セクション17.2.1.1「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」](#)を参照してください。

23.6.3 NDB Cluster レプリケーションの既知の問題

このセクションでは、NDB Cluster でレプリケーションを使用するときの既知の問題について説明します。

ソースとレプリカ間の接続が失われました。ソースクラスタの SQL ノードとレプリカクラスタの SQL ノード間、またはソース SQL ノードとソースクラスタのデータノード間で接続が失われる可能性があります。後者の場合にこ

の発生は、物理的な接続が喪失した結果（たとえば、ネットワークケーブルの破損など）だけではなく、データノードイベントのバッファのオーバーフローによる可能性もあります。SQL ノードは、応答が遅すぎると、クラスタでドロップされる可能性があります（これは、[MaxBufferedEpochs](#) および [TimeBetweenEpochs](#) 構成パラメータを調整することで、ある程度まで制御が可能です）。これが発生した場合は、ソース SQL ノードのバイナリログに記録されずに、ソースクラスタに新しいデータを挿入することも完全に可能です。このため、高可用性を保証するには、バックアップレプリケーションチャネルを維持し、プライマリチャネルを監視し、必要に応じてセカンダリレプリケーションチャネルにフェイルオーバーしてレプリカクラスタとソースの同期を保つことが非常に重要です。NDB Cluster は、そのようなモニタリングを単独で実行するようには設計されていません。このためには、外部アプリケーションが必要です。

ソース SQL ノードは、ソースクラスタへの接続または再接続時に「gap」イベントを発行します。（ギャップイベントは一種の「インシデントイベント」であり、データベースの内容に影響を与えるが、一連の変更として容易に表現できないインシデントが発生したことを示します。インシデントの例としては、サーバー障害、データベースの再同期化、一部のソフトウェア更新および一部のハードウェア変更があります。）レプリカは、レプリケーションログにギャップが発生すると、エラーメッセージとともに停止します。このメッセージは [SHOW REPLICA | SLAVE STATUS](#) の出力で使用可能で、レプリケーションストリームに登録されたインシデントのために SQL スレッドが停止し、手動操作が必要であることを示します。このような状況での対処の詳細については、[セクション23.6.8「NDB Cluster レプリケーションによるフェイルオーバーの実装」](#)を参照してください。

重要

NDB Cluster はレプリケーションステータスをモニターしたりフェイルオーバーを提供したりするように独自に設計されていないため、レプリケーションサーバーまたはクラスタの高可用性が要件である場合は、複数のレプリケーションラインを設定し、プライマリレプリケーションラインでソース [mysqld](#) をモニターして、必要に応じてセカンダリ行にフェイルオーバーするように準備する必要があります。これは、手動や、場合によってはサードパーティーのアプリケーションによって行う必要があります。この種のセットアップの実装に関する情報については、[セクション23.6.7「NDB Cluster レプリケーションでの2つのレプリケーションチャネルの使用」](#)および [セクション23.6.8「NDB Cluster レプリケーションによるフェイルオーバーの実装」](#)を参照してください。

スタンドアロン MySQL サーバーから NDB Cluster にレプリケートする場合、通常は1つのチャネルで十分です。

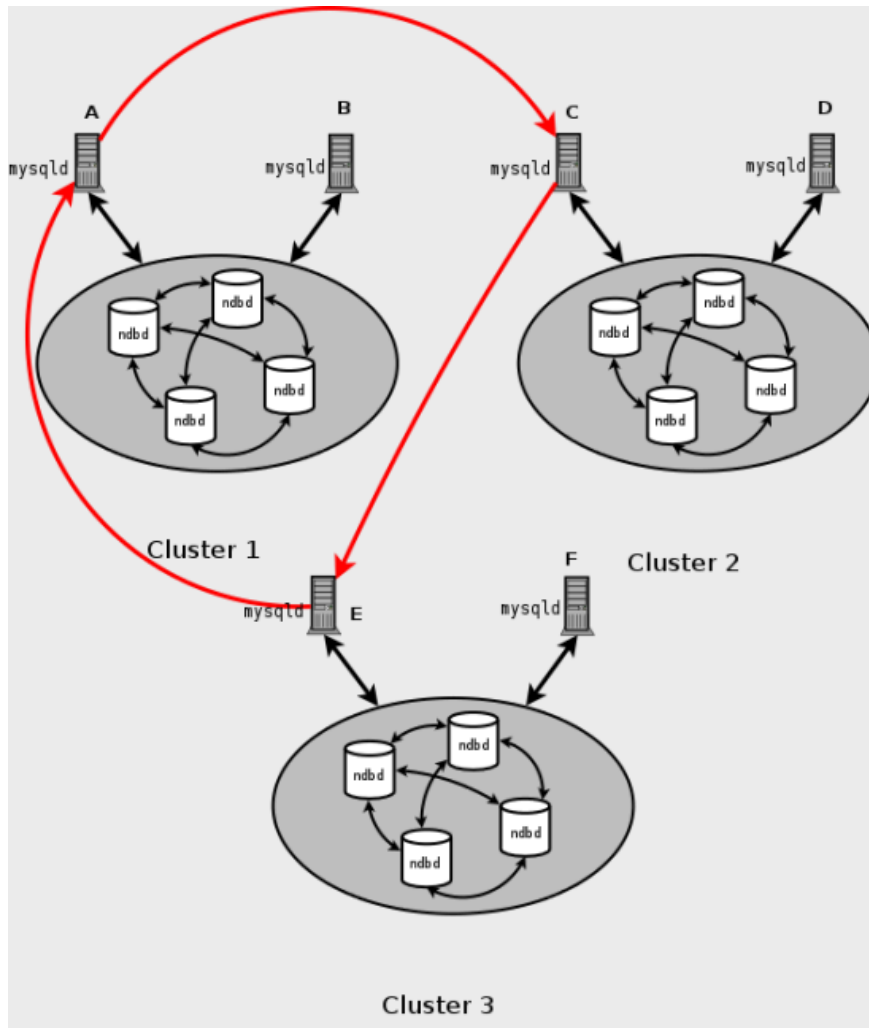
循環レプリケーション。 次の例に示すように、NDB Cluster Replication は循環レプリケーションをサポートします。レプリケーションの設定には、1、2、および3という番号が付けられた3つのNDB Clusterが含まれます。クラスタ1はクラスタ2のレプリケーションソースとして機能し、クラスタ2はクラスタ3のソースとして機能し、クラスタ3はクラスタ1のソースとして機能するため、円が完成します。各NDB Clusterには2つのSQLノードがあり、SQLノードAとBはクラスタ1に属し、SQLノードCとDはクラスタ2に属し、SQLノードEとFはクラスタ3に属します。

これらのクラスタを使用する循環レプリケーションは、次の条件を満たすかぎり、サポートされます。

- すべてのソースクラスタとレプリカクラスタのSQLノードが同じです。
- ソースおよびレプリカとして機能するすべてのSQLノードは、[log_slave_updates](#) システム変数を有効にして起動されます。

このタイプの循環レプリケーションのセットアップは、次の図に示すとおりです。

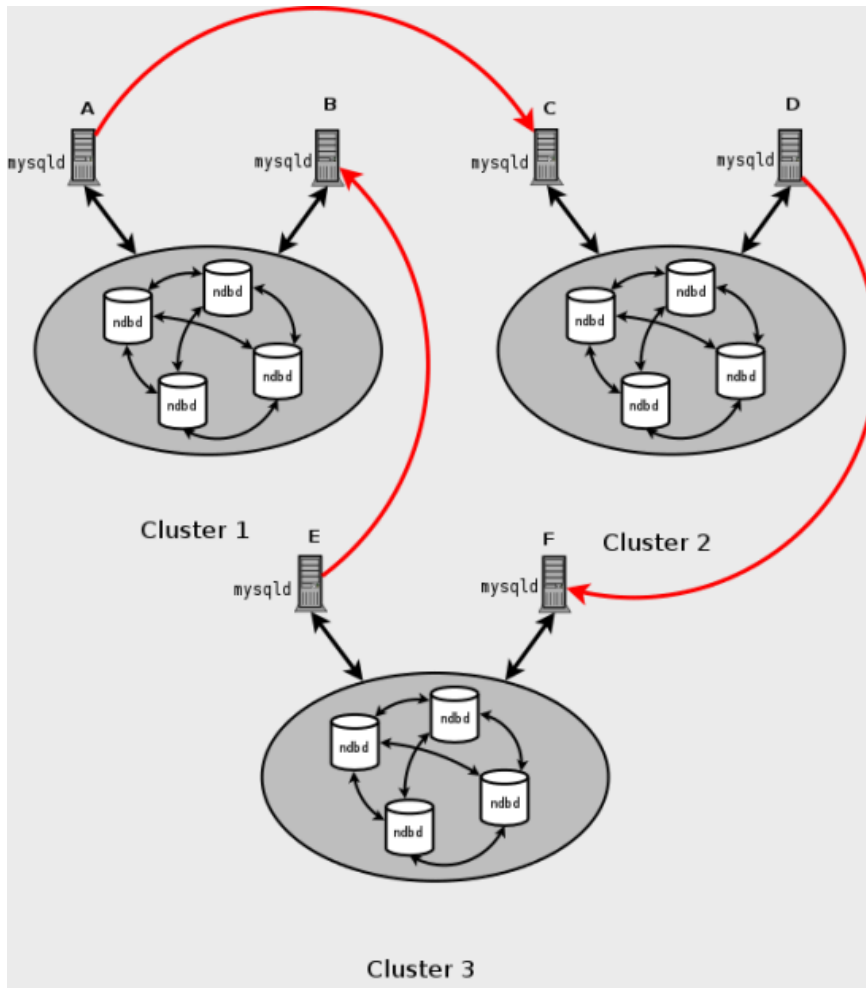
図 23.32 すべてのソースをレプリカとして使用する NDB Cluster 循環レプリケーション



このシナリオでは、クラスタ 1 の SQL ノード A はクラスタ 2 の SQL ノード C に複製し、SQL ノード C はクラスタ 3 の SQL ノード E に複製し、SQL ノード E は SQL ノード A に複製します。つまり、レプリケーション線 (ダイアグラムの曲線矢印で示される) は、ソースおよびレプリカとして使用されるすべての SQL ノードを直接接続します。

次に示すように、すべてのソース SQL ノードがレプリカであるわけではない循環レプリケーションを設定することもできます:

図 23.33 すべてのソースがレプリカではない NDB Cluster 循環レプリケーション



この場合、各クラスタ内の異なる SQL ノードがソースおよびレプリカとして使用されます。ただし、`log_slave_updates` システム変数を有効にして SQL ノードを起動しないでください。NDB Cluster のこのタイプの循環レプリケーションスキームでは、レプリケーションの行 (図の曲線矢印で再度示されています) が不連続である可能性があります。また徹底的にテストされていないため、実験的のみならず必要があることに注意してください。

注記

NDB ストレージエンジンは、NDB Cluster の循環レプリケーションを中断する重複キーおよびその他のエラーを抑制する多重呼出し不変実行モードを使用します。これはグローバル `slave_exec_mode` システム変数を `IDEMPOTENT` に設定することと同等ですが、NDB Cluster レプリケーションでは必要ありません。NDB Cluster はこの変数を自動的に設定し、明示的に設定しようとする試みを無視するためです。

NDB Cluster のレプリケーションと主キー。 ノードに障害が発生した場合でも、このような場合に重複行が挿入される可能性があるため、主キーのない NDB テーブルのレプリケーションでエラーが発生する可能性があります。このため、レプリケートされるすべての NDB テーブルに明示的な主キーを設定することを強くお勧めします。

NDB Cluster レプリケーションと一意キー。 古いバージョンの NDB Cluster では、NDB テーブルの一意キーカラムの値を更新する操作を複製すると、重複キーエラーが発生する可能性がありました。NDB テーブル間のレプリケーションに関するこの問題は、すべてのテーブル行の更新が実行されるまで一意キーのチェックを保留することで解決されます。

この方法で制約を保留することは、現在 NDB でのみサポートされています。したがって、NDB から InnoDB や MyISAM などの別のストレージエンジンにレプリケートする場合、一意の鍵の更新はまだサポートされていません。

一意キー更新の遅延チェックなしでレプリケートする場合に発生する問題は、次に示すように、`t`などの NDB テーブルを使用してソースで作成および移入され、遅延一意キー更新をサポートしないレプリカに転送されます:

```
CREATE TABLE t (  
  p INT PRIMARY KEY,  
  c INT,  
  UNIQUE KEY u (c)  
) ENGINE NDB;  
  
INSERT INTO t  
VALUES (1,1), (2,2), (3,3), (4,4), (5,5);
```

影響を受ける行は `ORDER BY` オプションで決定された順序で処理されるため、`t`での次の `UPDATE` ステートメントはソースで、テーブル全体での実行で、成功します:

```
UPDATE t SET c = c - 1 ORDER BY p;
```

テーブル全体ではなく一度に 1 つのパーティションに対して行の更新の順序が実行されるため、同じステートメントが複製キーエラーまたは他の制約違反で失敗します。

注記

各 NDB テーブルは、作成時に、キーで無条件にパーティション化されます。詳細については、[セクション24.2.5「KEY パーティショニング」](#)を参照してください。

GTID は未サポート。 グローバルトランザクション ID を使用するレプリケーションは、NDB ストレージエンジンと互換性がなく、サポートされていません。GTID を有効にすると、NDB Cluster レプリケーションが失敗する可能性があります。

マルチスレッド複製はサポートされていません。 NDB Cluster はマルチスレッドレプリカをサポートせず、`slave_parallel_workers`、`slave_checkpoint_group`、`slave_checkpoint_group` (または同等の `mysqld` 起動オプション) などの関連するシステム変数を設定しても効果はありません。

これは、レプリカが同じエポック内に書き込まれた場合、あるデータベースで発生しているトランザクションを別のデータベースで発生しているトランザクションから分離できない可能性があるためです。また、NDB ストレージエンジンによって処理されるすべてのトランザクションには、少なくとも 2 つのデータベース (ターゲットデータベースと `mysql` システムデータベース) が含まれ、`mysql.ndb_apply_status` テーブルを更新するための要件が満たされます ([セクション23.6.4「NDB Cluster レプリケーションスキーマおよびテーブル」](#)を参照)。これにより、トランザクションが特定のデータベースに固有であるというマルチスレッドの要件が解消されます。

`--initial` での再起動。 `--initial` オプションを使用してクラスタを再起動すると、GCI 番号とエポック番号のシーケンスが 0 からやり直されます。(これは通常 NDB Cluster に当てはまり、クラスタに関連するレプリケーションシナリオに限定されません。) この場合、レプリケーションに関与する MySQL サーバーは再起動されます。その後、`RESET MASTER` ステートメントと `RESET REPLICAS | SLAVE` ステートメントを使用して、無効な `ndb_binlog_index` テーブルと `ndb_apply_status` テーブルをそれぞれクリアする必要があります。

NDB から別のストレージエンジンへのレプリケーション。 ここに示す制限を考慮して、レプリカ上の別のストレージエンジンを使用して、ソース上の NDB テーブルをテーブルにレプリケートできます:

- マルチソースレプリケーションおよび循環レプリケーションはサポートされていません (これが機能するには、ソースとレプリカの両方のテーブルで NDB ストレージエンジンを使用する必要があります)。
- レプリカ上のテーブルのバイナリロギングを実行しないストレージエンジンを使用するには、特別な処理が必要です。
- レプリカ上のテーブルに非トランザクションストレージエンジンを使用するには、特別な処理も必要です。
- ソース `mysqld` は、`--ndb-log-update-as-write=0` または `--ndb-log-update-as-write=OFF` で起動する必要があります。

次のいくつかのパラグラフでは、ここで説明したそれぞれの問題の追加情報を示します。

NDB をほかのストレージエンジンにレプリケートする場合、複数のソースはサポートされません。 NDB から別のストレージエンジンへのレプリケーションの場合、2 つのデータベース間の関係は 1 対 1 である必要があります。つま

り、NDB Cluster とほかのストレージエンジンの間では双方向または循環レプリケーションはサポートされていません。

また、NDB および別のストレージエンジン間で複製する場合、複数のレプリケーションチャンネルを構成できません。(NDB Cluster データベースは、複数の NDB Cluster データベースに同時に複製できます。) ソースで NDB テーブルが使用されている場合でも、すべての変更のバイナリログを複数の MySQL Server で保持できますが、レプリカでソースを変更(フェイルオーバー)するには、レプリカで新しいソースとレプリケーションの関係を明示的に定義する必要があります。

バイナリロギングを実行しないストレージエンジンへの「NDB の複製」テーブル。 NDB Cluster から、独自のバイナリロギングを処理しないストレージエンジンを使用するレプリカに複製しようとすると、レプリケーションプロセスはエラー「バイナリロギングができません ... 複数のエンジンが関与し、少なくとも 1 つのエンジンがセルフロギングであるため、ステートメントをアトミックに書き込めません」(Error 1595) で中止されます。次の方法のいずれかで、この問題を回避できます。

- レプリカのバイナリロギングをオフにします。 これを実現するには、`sql_log_bin = 0` を設定します。
- `mysql.ndb_apply_status` テーブルに使用されるストレージエンジンを変更します。 このテーブルが独自のバイナリロギングを処理しないエンジンを使用することになるため、競合も解消できます。 これを行うには、レプリカで `ALTER TABLE mysql.ndb_apply_status ENGINE=MyISAM` などのステートメントを発行します。レプリカで NDB 以外のストレージエンジンを使用する場合は、複数のレプリカの同期を維持する必要がないため、これを安全に行うことができます。
- レプリカの `mysql.ndb_apply_status` テーブルに対する変更を除外。 これを行うには、`--replicate-ignore-table=mysql.ndb_apply_status` を使用してレプリカを起動します。レプリケーションでほかのテーブルを無視する必要がある場合、代わりに適切な `--replicate-wild-ignore-table` オプションを使用することをお勧めします。

重要

NDB Cluster 間でレプリケートする場合は、`mysql.ndb_apply_status` のレプリケーションまたはバイナリロギングを無効にしたり、このテーブルに使用されるストレージエンジンを変更したりしないでください。詳細は、[NDB Cluster 間のレプリケーションを使用したレプリケーションおよびバイナリログフィルタリング規則](#)を参照してください。

NDB から非トランザクションストレージエンジンへのレプリケーション。 `MyISAM` などの非トランザクションストレージエンジンに NDB から複製する場合、`INSERT ... ON DUPLICATE KEY UPDATE` ステートメントを複製するときに、不要な重複キーエラーが発生することがあります。これらを抑制するには、`--ndb-log-update-as-write=0` を使用します。これにより、更新は更新としてではなく書き込みとして強制的に記録されます。

NDB Cluster 間のレプリケーションを使用したレプリケーションおよびバイナリログフィルタリング規則。 レプリケートされるデータベースまたはテーブルのフィルタリングにオプション `--replicate-do-*`、`--replicate-ignore-*`、`--binlog-do-db` または `--binlog-ignore-db` を使用している場合は、NDB Cluster 間のレプリケーションが正しく動作するために必要な `mysql.ndb_apply_status` のレプリケーションまたはバイナリロギングをブロックしないように注意する必要があります。特に、次の点に留意しておく必要があります。

- `--replicate-do-db=db_name` を使用すると (およびほかの `--replicate-do-*` または `--replicate-ignore-*` オプションを使用しない)、データベース `db_name` にあるテーブルだけが複製されます。この場合は、`--replicate-do-db=mysql`、`--binlog-do-db=mysql` または `--replicate-do-table=mysql.ndb_apply_status` を使用して、`mysql.ndb_apply_status` がレプリカに移入されていることも確認する必要があります。
`--binlog-do-db=db_name` を使用すると (およびほかの `--binlog-do-db` オプションを使用しない)、データベース `db_name` にあるテーブルへの変更のみがバイナリログに書き込まれます。この場合は、`--replicate-do-db=mysql`、`--binlog-do-db=mysql` または `--replicate-do-table=mysql.ndb_apply_status` を使用して、`mysql.ndb_apply_status` がレプリカに移入されていることも確認する必要があります。
- `--replicate-ignore-db=mysql` を使用すると、`mysql` データベースにあるテーブルは複製されません。この場合、`--replicate-do-table=mysql.ndb_apply_status` も使用して、確実に `mysql.ndb_apply_status` を複製する必要があります。
`--binlog-ignore-db=mysql` を使用すると、`mysql` データベースにあるテーブルへの変更はバイナリログに書き込まれません。この場合、`--replicate-do-table=mysql.ndb_apply_status` も使用して、確実に `mysql.ndb_apply_status` を複製する必要があります。

各レプリケーションルールには次のことも必要になります。

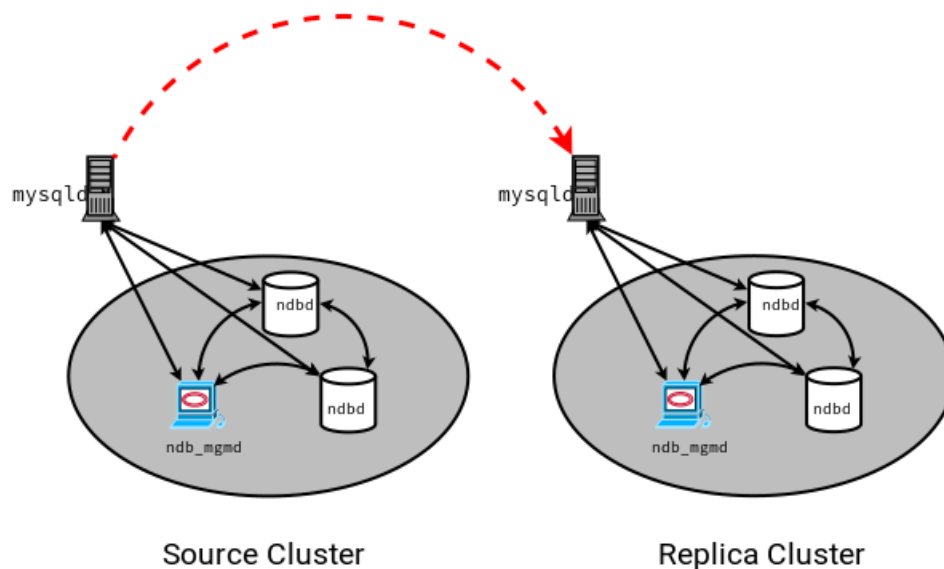
1. 独自の `--replicate-do-*` または `--replicate-ignore-*` オプション。複数のルールはレプリケーションの1つのフィルタリングオプションで表現できません。これらのルールについての情報は、[セクション17.1.6「レプリケーションおよびバイナリロギングのオプションと変数」](#)を参照してください。
2. 独自の `--binlog-do-db` または `--binlog-ignore-db` オプション。複数のルールはバイナリログの1つのフィルタリングオプションで表現できません。これらのルールについての情報は、[セクション5.4.4「バイナリログ」](#)を参照してください。

NDB Cluster を、NDB 以外のストレージエンジンを使用するレプリカにレプリケートする場合は、このセクションのほかの場所で説明されているように、以前に指定した考慮事項が適用されないことがあります。

NDB Cluster レプリケーションと IPv6。 NDB 8.0.22 以降では、すべてのタイプの NDB Cluster ノードが IPv6 をサポートします。これには、管理ノード、データノード、および API または SQL ノードが含まれます。

NDB 8.0.22 より前では、NDB API および MGM API (およびデータノードと管理ノード) は IPv6 をサポートしていませんが、NDB Cluster 内で SQL ノードとして機能する MySQL Servers-including は IPv6 を使用してほかの MySQL Servers に接続できます。8.0.22 より前のバージョンの NDB Cluster では、次の図の点線矢印で示されているように、IPv6 を使用してソースおよびレプリカとして機能する SQL ノードを接続し、クラスター間で複製できます：

図 23.34 IPv6 を使用して接続された SQL ノード間のレプリケーション



NDB 8.0.22 より前は、範囲内を起点とするすべての接続で、前の図で実線矢印で示されている NDB Cluster が IPv4 を使用する必要があります。つまり、NDB Cluster データノード、管理サーバー、および管理クライアントはすべて、IPv4 を使用して相互にアクセスできる必要があります。また、SQL ノードは IPv4 を使用してクラスターと通信する必要があります。NDB 8.0.22 以降では、これらの制限は適用されなくなりました。また、NDB および MGM API を使用して記述されたアプリケーションは、IPv6 のみの環境を想定して記述および配備できます。

属性の昇格と降格。 NDB Cluster レプリケーションには、属性の昇格と降格のサポートが含まれます。後者の実装では、不可逆型変換と非不可逆型変換が区別され、レプリカでのそれらの使用は、`slave_type_conversions` グローバルサーバーシステム変数を設定することで制御できます。

NDB Cluster での属性の昇格および降格の詳細は、[行ベースレプリケーション: 属性の昇格と降格](#)を参照してください。

NDB は、InnoDB や MyISAM とは異なり、仮想カラムへの変更をバイナリログに書き込みませんが、NDB Cluster レプリケーションまたは NDB とほかのストレージエンジン間のレプリケーションに悪影響はありません。格納された生成カラムに対する変更がログに記録されます。

23.6.4 NDB Cluster レプリケーションスキーマおよびテーブル

NDB Cluster 内のレプリケーションでは、レプリケートされるクラスタとレプリカの両方で SQL ノードとして機能する各 MySQL Server インスタンス上の `mysql` データベース内の多数の専用テーブルが使用されます。これは、レプリカが単一サーバーであるかクラスタであるかに関係なく当てはまります。これらのテーブルは、MySQL のインストールプロセス中に作成され、バイナリログのインデックス作成データを格納するためのテーブルを含みます。`ndb_binlog_index` テーブルは各 MySQL サーバーに対してローカルであり、クラスタ化には関与しないため、InnoDB ストレージエンジンを使用します。つまり、ソースクラスタに参加している各 `mysqld` に個別に作成する必要があります。(バイナリログ自体には、レプリケートされるクラスタ内のすべての MySQL サーバーからの更新が含まれません。) このテーブルは次のように定義されます。

```
CREATE TABLE `ndb_binlog_index` (
  `Position` BIGINT(20) UNSIGNED NOT NULL,
  `File` VARCHAR(255) NOT NULL,
  `epoch` BIGINT(20) UNSIGNED NOT NULL,
  `inserts` INT(10) UNSIGNED NOT NULL,
  `updates` INT(10) UNSIGNED NOT NULL,
  `deletes` INT(10) UNSIGNED NOT NULL,
  `schemaops` INT(10) UNSIGNED NOT NULL,
  `orig_server_id` INT(10) UNSIGNED NOT NULL,
  `orig_epoch` BIGINT(20) UNSIGNED NOT NULL,
  `gci` INT(10) UNSIGNED NOT NULL,
  `next_position` bigint(20) unsigned NOT NULL,
  `next_file` varchar(255) NOT NULL,
  PRIMARY KEY (`epoch`,`orig_server_id`,`orig_epoch`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

注記

古いリリース (NDB 7.5.2 より前) からアップグレードする場合は、MySQL のアップグレード手順を実行して、システムテーブルがアップグレードされていることを確認します。(MySQL 8.0.16 の時点で、`--upgrade=FORCE` オプションを使用してサーバーを起動します。MySQL 8.0.16 より前は、サーバーの起動後に `--force` および `--upgrade-system-tables` オプションを指定して `mysql_upgrade` を起動してください。) システムテーブルのアップグレードにより、このテーブルに対して `ALTER TABLE ... ENGINE=INNODB` ステートメントが実行されます。このテーブルに対する MyISAM ストレージエンジンの使用は、下位互換性のために引き続きサポートされています。

`ndb_binlog_index` では、InnoDB への変換後に追加のディスク領域が必要になる場合があります。これが問題になった場合は、このテーブルの InnoDB テーブルスペースを使用するか、`ROW_FORMAT` を `COMPRESSED` に変更するか、またはその両方を行うことで、領域を節約できます。詳細は、[セクション13.1.21「CREATE TABLESPACE ステートメント」](#)、[セクション13.1.20「CREATE TABLE ステートメント」](#) および [セクション15.6.3「テーブルスペース」](#) を参照してください。

`ndb_binlog_index` テーブルのサイズは、バイナリログファイルごとのエポックの数とバイナリログファイルの数によって異なります。一般的に、バイナリログファイル当たりのエポック数は、エポックごとに作成されるバイナリログの量とバイナリログファイルのサイズに依存し、エポックが小さくなるとファイル当たりのエポックが増えます。`--ndb-log-empty-epochs` オプションが `OFF` の場合でも、空のエポックによって `ndb_binlog_index` テーブルへの挿入が生成されることに注意してください。つまり、ファイル当たりのエントリ数は、ファイルが使用されている時間の長さによって異なります。この関係は、次の式で表すことができます:

$$[\text{number of epochs per file}] = [\text{time spent per file}] / \text{TimeBetweenEpochs}$$

ビジー状態の NDB Cluster はバイナリログに定期的書き込み、おそらく静かなバイナリログファイルよりも速くバイナリログファイルをローテーションします。つまり、`--ndb-log-empty-epochs=ON` を使用した「quiet」NDB Cluster は、実際には、多数のアクティビティを持つものよりも、ファイルあたりの `ndb_binlog_index` 行数が多くなる可能性があります。

`--ndb-log-orig` オプションを指定して `mysqld` を起動すると、`orig_server_id` および `orig_epoch` カラムにはそれぞれ、イベントが発生したサーバーの ID と、元のサーバーでイベントが発生したエポックが格納されます。これは、複数のソースを使用する NDB Cluster レプリケーション設定で役立ちます。マルチソース設定 ([セクション23.6.10「NDB Cluster レプリケーション: 双方向および循環レプリケーション」](#) を参照) でレプリカに適用されたエポックに最も近いバイナリログ位置を見つけるために使用される `SELECT` ステートメントでは、インデックス付けされていないこれ

らの2つのカラムが使用されます。特にソースが `--ndb-log-empty-epochs=ON` で実行されている場合、クエリーでテーブルスキャンを実行する必要があるため、フェイルオーバーを試行するとパフォーマンスの問題が発生する可能性があります。マルチソースのフェイルオーバー時間を改善するには、次に示すように、これらのカラムにインデックスを追加します:

```
ALTER TABLE mysql.ndb_binlog_index
  ADD INDEX orig_lookup USING BTREE (orig_server_id, orig_epoch);
```

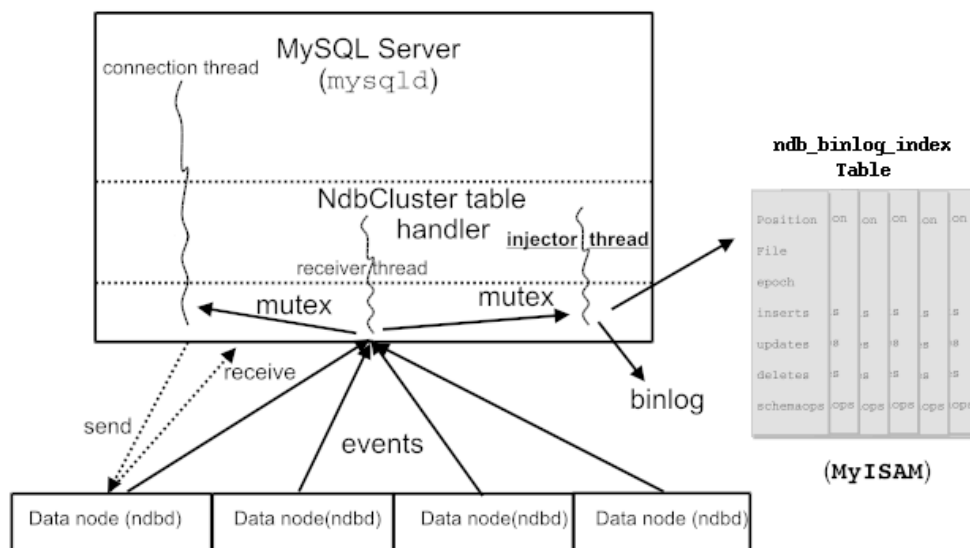
このような場合、バイナリログの位置を取得するために使用されるクエリーは `orig_server_id` または `orig_epoch` を使用しないため、単一のソースから単一のレプリカにレプリケートするときにこのインデックスを追加しても利点はありませぬ。

`next_position` および `next_file` カラムの使用についての詳細は、[セクション23.6.8「NDB Cluster レプリケーションによるフェイルオーバーの実装」](#)を参照してください。

次の図は、NDB Cluster レプリケーションソースサーバー、バイナリログインジェクタスレッド、および `mysql.ndb_binlog_index` テーブルの関係を示しています。

図 23.35 レプリケーションソースクラスタ

MySQL Replication Between Clusters, Injecting into Binlog



`ndb_apply_status` という名前の追加のテーブルを使用して、ソースからレプリカにレプリケートされた操作のレコードを保持します。 `ndb_binlog_index` の場合とは異なり、このテーブルのデータは (レプリカ) クラスタ内のいずれの SQL ノードにも固有ではないため、 `ndb_apply_status` では次に示すように `NDBCLUSTER` ストレージエンジンを使用できます:

```
CREATE TABLE `ndb_apply_status` (
  `server_id` INT(10) UNSIGNED NOT NULL,
  `epoch` BIGINT(20) UNSIGNED NOT NULL,
  `log_name` VARCHAR(255) CHARACTER SET latin1 COLLATE latin1_bin NOT NULL,
  `start_pos` BIGINT(20) UNSIGNED NOT NULL,
  `end_pos` BIGINT(20) UNSIGNED NOT NULL,
  PRIMARY KEY (`server_id`) USING HASH
) ENGINE=NDBCLUSTER DEFAULT CHARSET=latin1;
```


`ndb_apply_status` テーブルはレプリカにのみ移入されるため、ソースではこのテーブルに行が含まれることはありません。したがって、そこで `DataMemory` を `ndb_apply_status` に割り当てる必要はありません。

このテーブルはソース上で生成されたデータから移入されるため、レプリケートを許可するようにしてください。誤ってレプリカが `ndb_apply_status` を更新できないようにしたり、ソースがバイナリログに書き込まれないようにしたりするレプリケーションフィルタリングまたはバイナリログフィルタリングの規則によって、クラスタ間のレプリケーションが正しく動作しなくなる可能性があります。このようなフィルタリングルールに起因する潜在的な問題の詳細は、[NDB Cluster 間のレプリケーションを使用したレプリケーションおよびバイナリログフィルタリング規則](#)を参照してください。

`ndb_binlog_index` と `ndb_apply_status` のテーブルは、ユーザーによって明示的に複製されることはないため、`mysql` データベースに作成されます。これらのテーブルはどちらも `NDB` バイナリログ (binlog) インジェクタスレッドによって保守されるため、通常、どちらのテーブルの作成または保守にもユーザーの介入は必要ありません。これにより、ソース `mysqld` プロセスが `NDB` ストレージエンジンによって実行された変更を更新されたままになります。`NDB` binlog インジェクタスレッドは `NDB` ストレージエンジンから直接イベントを受け取ります。`NDB` インジェクタは、クラスタ内のすべてのデータイベントを取得する役割を担い、データを変更、挿入、または削除するすべてのイベントが `ndb_binlog_index` テーブルに記録されたかを確認します。レプリカ I/O スレッドは、ソースバイナリログからレプリカリレーログにイベントを転送します。

`ndb_binlog_index` および `ndb_apply_status` は自動的に作成および保守されますが、`NDB Cluster` をレプリケーション用に準備する最初の手順として、これらのテーブルの存在と整合性をチェックすることをお勧めします。バイナリログに記録されたイベントデータを表示するには、ソースで `mysql.ndb_binlog_index` テーブルを直接クエリーします。これは、ソースまたはレプリカ `SQL` ノードのいずれかで `SHOW BINLOG EVENTS` ステートメントを使用して実行することもできます。([セクション13.7.7.2「SHOW BINLOG EVENTS ステートメント」](#)を参照してください。)

`SHOW ENGINE NDB STATUS` の出力から有効な情報を取得することもできます。

注記

`NDB` テーブルでスキーマの変更を行う場合、アプリケーションは `ALTER TABLE` ステートメントを発行した `MySQL` クライアント接続でこのステートメントが戻るまで待ってから、更新されたテーブル定義の使用を試みます。

`ndb_apply_status` テーブルがレプリカに存在しない場合は、`ndb_restore` によって再作成されます。

`NDB Cluster` レプリケーションの競合解決には、追加の `mysql.ndb_replication` テーブルが存在する必要があります。現在、このテーブルは手動で作成する必要があります。これを行う方法については、[セクション23.6.11「NDB Cluster レプリケーションの競合解決」](#)を参照してください。

23.6.5 NDB Cluster のレプリケーションの準備

`NDB Cluster` のレプリケーションの準備には、次の手順が含まれます:

1. すべての `MySQL` サーバーに対してバージョンの互換性を確認します ([セクション23.6.2「NDB Cluster レプリケーションの一般的な要件」](#)を参照してください)。
2. 次の 2 つの `SQL` ステートメントを使用して、適切な権限でソースクラスタにレプリケーションアカウントを作成します:

```
mysqlS> CREATE USER 'replica_user'@'replica_host'  
-> IDENTIFIED BY 'replica_password';  
  
mysqlS> GRANT REPLICATION SLAVE ON *.*  
-> TO 'replica_user'@'replica_host';
```

前述のステートメントで、`replica_user` はレプリケーションアカウントユーザー名、`replica_host` はレプリカのホスト名または IP アドレス、`replica_password` はこのアカウントに割り当てるパスワードです。

たとえば、`myreplica` という名前のレプリカユーザーアカウントを作成し、`replica-host` という名前のホストからログインし、パスワード `53cr37` を使用するには、次の `CREATE USER` および `GRANT` ステートメントを使用します:

```
mysqlS> CREATE USER 'myreplica'@'replica-host'  
-> IDENTIFIED BY '53cr37';
```

```
mysqlS> GRANT REPLICATION SLAVE ON *.*  
-> TO 'myreplica'@'replica-host';
```

セキュリティ上の理由から、レプリケーションアカウントには、他の目的には使用されない一意のユーザーアカウントを使用することをお勧めします。

3. ソースを使用するようにレプリカを設定します。これは、[mysql](#) クライアントを使用して、[CHANGE REPLICATION SOURCE TO](#) ステートメント (MySQL 8.0.23 の場合) または [CHANGE MASTER TO](#) ステートメント (MySQL 8.0.23 の場合) で実行できます:

```
mysqlR> CHANGE MASTER TO  
-> MASTER_HOST='source_host',  
-> MASTER_PORT=source_port,  
-> MASTER_USER='replica_user',  
-> MASTER_PASSWORD='replica_password';
```

Or from MySQL 8.0.23:

```
mysqlR> CHANGE REPLICATION SOURCE TO  
-> SOURCE_HOST='source_host',  
-> SOURCE_PORT=source_port,  
-> SOURCE_USER='replica_user',  
-> SOURCE_PASSWORD='replica_password';
```

前述のステートメントで、[source_host](#) はレプリケーションソースのホスト名または IP アドレス、[source_port](#) はソースへの接続時に使用するレプリカのポート、[replica_user](#) はソースのレプリカに設定されたユーザー名、[replica_password](#) は前のステップでそのユーザーアカウントに設定されたパスワードです。

たとえば、前のステップで作成したレプリケーションアカウントでホスト名が [rep-source](#) の MySQL サーバーを使用するようレプリカに指示するには、次のステートメントを使用します:

```
mysqlR> CHANGE MASTER TO  
-> MASTER_HOST='rep-source',  
-> MASTER_PORT=3306,  
-> MASTER_USER='myreplica',  
-> MASTER_PASSWORD='53cr37';
```

Or from MySQL 8.0.23:

```
mysqlR> CHANGE REPLICATION SOURCE TO  
-> SOURCE_HOST='rep-source',  
-> SOURCE_PORT=3306,  
-> SOURCE_USER='myreplica',  
-> SOURCE_PASSWORD='53cr37';
```

このステートメントで使用できるオプションの完全なリストについては、[セクション13.4.2.1「CHANGE MASTER TO ステートメント」](#)を参照してください。

レプリケーションバックアップ機能を提供するには、レプリケーションプロセスを開始する前に、レプリカ [my.cnf](#) ファイルに [--ndb-connectstring](#) オプションを追加する必要があります。詳細は、[セクション23.6.9「NDB Cluster レプリケーションによる NDB Cluster バックアップ」](#)を参照してください。

レプリカ用に [my.cnf](#) で設定できるその他のオプションについては、[セクション17.1.6「レプリケーションおよびバイナリロギングのオプションと変数」](#)を参照してください。

4. ソースクラスタがすでに使用されている場合は、ソースのバックアップを作成してレプリカにロードし、レプリカがソースと同期するのに必要な時間を短縮できます。レプリカで NDB Cluster も実行されている場合は、[セクション23.6.9「NDB Cluster レプリケーションによる NDB Cluster バックアップ」](#)で説明されているバックアップおよび復元手順を使用してこれを実行できます。

```
ndb-connectstring=management_host[:port]
```

レプリカで NDB Cluster を使用していない場合は、ソースで次のコマンドを使用してバックアップを作成できます:

```
shellS> mysqldump --master-data=1
```

次に、ダンプファイルをレプリカにコピーして、結果のデータダンプをレプリカにインポートします。この後、次に示すように、[mysql](#) クライアントを使用してダンプファイルからレプリカデータベースにデータをインポート

できます。ここで、`dump_file` はソースで `mysqldump` を使用して生成されたファイルの名前、`db_name` はレプリケートされるデータベースの名前です:

```
shellR> mysql -u root -p db_name < dump_file
```

`mysqldump` で使用するオプションの完全なリストは、[セクション4.5.4 「mysqldump — データベースバックアッププログラム」](#) を参照してください。

注記

この方法でレプリカにデータをコピーする場合は、コマンドラインで `--skip-slave-start` オプションを使用してレプリカが起動されていることを確認するか、レプリカ `my.cnf` ファイルに `skip-slave-start` を含めて、すべてのデータがロードされる前にソースへの接続を試行してレプリケートを開始しないようにする必要があります。データのロードが完了したら、次の2つのセクションで説明する追加ステップに従います。

- レプリケーションソースとして機能する各 MySQL サーバーに一意のサーバー ID が割り当てられ、行ベースの形式を使用してバイナリロギングが有効になっていることを確認します。([セクション17.2.1 「レプリケーション形式」](#) を参照してください。) また、`slave_allow_batching` システム変数を有効にし、場合によっては `--ndb-batch-size` および `--ndb-blob-write-batch-bytes` オプションで使用される値も増やすことをお勧めします。これらのオプションはすべて、ソースサーバーの `my.cnf` ファイルで設定することも、ソース `mysqld` プロセスの起動時にコマンドラインで設定することもできます。詳しくは [セクション23.6.6 「NDB Cluster レプリケーションの開始 \(シングルレプリケーションチャンネル\)」](#) をご覧ください。

23.6.6 NDB Cluster レプリケーションの開始 (シングルレプリケーションチャンネル)

このセクションでは、単一のレプリケーションチャンネルを使用して NDB Cluster レプリケーションを開始する手順の概要を示します。

- 次のコマンドを発行して、MySQL レプリケーションソースサーバーを起動します:

```
shellS> mysql --ndbcluster --server-id=id \
--log-bin --ndb-log-bin &
```

前述のステートメントで、`id` はこのサーバーの一意の ID です ([セクション23.6.2 「NDB Cluster レプリケーションの一般的な要件」](#) を参照)。これにより、適切なロギング形式を使用してバイナリロギングを有効にして、サーバー `mysqld` プロセスが開始されます。NDB 8.0 では、`--ndb-log-bin` オプションを使用して NDB テーブルへの更新のロギングを明示的に有効にすることも必要です。これは NDB Cluster の以前のバージョンからの変更であり、このオプションはデフォルトで有効になっていました。

注記

`--binlog-format=MIXED` を使用してソースを起動することもできます。この場合、クラスター間のレプリケート時に行ベースのレプリケーションが自動的に使用されます。ステートメントベースのバイナリロギングは NDB Cluster レプリケーションではサポートされていません ([セクション23.6.2 「NDB Cluster レプリケーションの一般的な要件」](#) を参照)。

- 次に示すように、MySQL レプリカサーバーを起動します:

```
shellR> mysql --ndbcluster --server-id=id &
```

前述のコマンドで、`id` はレプリカサーバーの一意の ID です。レプリカでロギングを有効にする必要はありません。

注記

このコマンドで `--skip-slave-start` オプションを使用するか、レプリケーションをすぐに開始しないかぎり、`skip-slave-start` をレプリカサーバーの `my.cnf` ファイルに含める必要があります。このオプションを使用すると、次のステップ4で説明するように、適切な `START REPLICA | SLAVE` ステートメントが発行されるまでレプリケーションの開始が遅延されます。

3. レプリカサーバーをソースサーバーのレプリケーションバイナリログと同期化する必要があります。バイナリロギングがソースで以前に実行されていない場合は、レプリカで次のステートメントを実行します:

```
mysqlR> CHANGE MASTER TO
-> MASTER_LOG_FILE=",
-> MASTER_LOG_POS=4;

Or from MySQL 8.0.23:
mysqlR> CHANGE REPLICATION SOURCE TO
-> SOURCE_LOG_FILE=",
-> SOURCE_LOG_POS=4;
```

これにより、レプリカはログの開始点からソースサーバーのバイナリログの読取りを開始するように指示されます。その他:バックアップを使用してソースからデータをロードする場合。このような場合に `MASTER_LOG_FILE` および `MASTER_LOG_POS` に使用する正しい値を取得する方法の詳細は、[セクション 23.6.8 「NDB Cluster レプリケーションによるフェイルオーバーの実装」](#) を参照してください。

4. 最後に、レプリカ上の `mysql` クライアントから次のコマンドを発行して、レプリケーションの適用を開始するようにレプリカに指示します:

```
mysqlR> START SLAVE;
Or from MySQL 8.0.22:
mysqlR> START REPLICA;
```

これにより、ソースからレプリカへのデータおよび変更の転送も開始されます。

次のセクションで説明する手順に類似した方法で、2つのレプリケーションチャンネルを使用することも可能です。この方法と1つのレプリケーションチャンネルを使用する方法との違いは[セクション23.6.7「NDB Cluster レプリケーションでの2つのレプリケーションチャンネルの使用」](#)で説明しています。

バッチ更新を有効にして、クラスタのレプリケーションのパフォーマンスを向上することも可能です。これを行うには、レプリカ `mysqld` プロセスで `slave_allow_batching` システム変数を設定します。通常、更新は受け取るとすぐに適用されます。ただし、バッチ処理を使用すると、それぞれ 32 KB のバッチで更新が適用されます。特に個々の更新が比較的小さい場合、スループットが高くなり、CPU 使用率が低下する可能性があります。

注記

バッチ処理はエポック単位で機能します。複数のトランザクションに属する更新は、同じバッチの一部として送信できます。

すべての未処理の更新は、その更新の合計が 32K バイト未満であっても、エポックの最後に達したときに適用されます。

バッチ処理は、実行時にオンとオフを切り替えることができます。実行時にこれを有効にするため、次の2つのステートメントのどちらかを使用できます。

```
SET GLOBAL slave_allow_batching = 1;
SET GLOBAL slave_allow_batching = ON;
```

特定のバッチによって問題が発生した場合 (効果が正しくレプリケートされていないように見えるステートメントなど)、次のいずれかのステートメントを使用してバッチ処理を非アクティブ化できます:

```
SET GLOBAL slave_allow_batching = 0;
SET GLOBAL slave_allow_batching = OFF;
```

次のように、適切な `SHOW VARIABLES` ステートメントを使用して、バッチ処理が現在使用されているかどうかを確認できます:

```
mysql> SHOW VARIABLES LIKE 'slave%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| slave_allow_batching | ON |
| slave_compressed_protocol | OFF |
| slave_load_tmpdir | /tmp |
| slave_net_timeout | 3600 |
| slave_skip_errors | OFF |
```

```
| slave_transaction_retries | 10 |
+-----+-----+
6 rows in set (0.00 sec)
```

23.6.7 NDB Cluster レプリケーションでの 2 つのレプリケーションチャンネルの使用

さらに完成された例のシナリオでは、2 つのレプリケーションチャンネルを使用することで冗長性を提供し、これによって、1 つのレプリケーションチャンネルで発生する可能性のある障害を防ぐことを想定しています。これには、合計 4 台のレプリケーションサーバー、ソースクラスタに 2 台のソースサーバー、レプリカクラスタに 2 台のレプリカサーバーが必要です。以後の説明では、一意の識別子が次のように割り当てられているものとします。

表 23.68 テキストで説明されている NDB Cluster レプリケーションサーバー

サーバー ID	説明
1	ソース - プライマリレプリケーションチャンネル (S)
2	ソース - セカンダリレプリケーションチャンネル (S')
3	レプリカ - プライマリレプリケーションチャンネル (R)
4	レプリカ - セカンダリレプリケーションチャンネル (R')

2 つのチャンネルを使用するレプリケーションの設定は、1 つのレプリケーションチャンネルの設定と根本的に異なるわけではありません。まず、プライマリおよびセカンダリのレプリケーションソースサーバーの `mysqld` プロセスを起動してから、プライマリおよびセカンダリの複製のプロセスを起動する必要があります。レプリケーションプロセスは、各レプリカに対して `START REPLICIA | SLAVE` ステートメントを発行することで開始できます。コマンドと、そのコマンドの発行に必要な順序を次に示します。

1. プライマリレプリケーションソースを起動します:

```
shellS> mysqld --ndbcluster --server-id=1 \
--log-bin &
```

2. セカンダリレプリケーションソースを起動します:

```
shellS> mysqld --ndbcluster --server-id=2 \
--log-bin &
```

3. プライマリレプリカサーバーを起動します:

```
shellR> mysqld --ndbcluster --server-id=3 \
--skip-slave-start &
```

4. セカンダリレプリカサーバーを起動します:

```
shellR> mysqld --ndbcluster --server-id=4 \
--skip-slave-start &
```

5. 最後に、次に示すように、プライマリレプリカで `START REPLICIA | SLAVE` ステートメントを実行して、プライマリチャンネルでレプリケーションを開始します:

```
mysqlR> START SLAVE;
Or from MySQL 8.0.22:
mysqlR> START REPLICIA;
```

警告

この時点ではプライマリチャンネルのみを起動する必要があります。セカンダリレプリケーションチャンネルは、[セクション23.6.8「NDB Cluster レプリケーションによるフェイルオーバーの実装」](#)で説明されているように、プライマリレプリケーションチャンネルに障害が発生した場合にのみ起動する必要があります。複数のレプリケーションチャンネルを同時に実行すると、不要な重複レコードがレプリカに作成される可能性があります。

前述のように、レプリカでバイナリロギングを有効にする必要はありません。

23.6.8 NDB Cluster レプリケーションによるフェイルオーバーの実装

プライマリクラスタのレプリケーションプロセスが失敗した場合、セカンダリレプリケーションチャンネルに切り替えることができます。次に、この実現に必要なステップについて説明します。

1. 最新のグローバルチェックポイント (GCP) の時刻を取得します。つまり、次のクエリーを使用して、レプリカクラスタの `ndb_apply_status` テーブルから最新のエポックを判別する必要があります:

```
mysqlR> SELECT @latest:=MAX(epoch)
-> FROM mysql.ndb_apply_status;
```

ソースとレプリカが各ホストで実行されている循環レプリケーショントポロジでは、`ndb_log_apply_status=1` を使用している場合、NDB Cluster エポックは複製バイナリログに書き込まれます。つまり、`ndb_apply_status` テーブルには、このホスト上のレプリカおよびこのホストで実行されているレプリケーションソースサーバーのレプリカとして機能する他のホストのレプリカに関する情報が含まれます。

この場合、このレプリカの最新のエポックを決定して、このレプリカの設定に使用された `CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO` ステートメントの `IGNORE_SERVER_IDS` オプションにリストされていないエポックをこのレプリカバイナリログ内の他のレプリカから除外する必要があります。このようなエポックを除外する理由は、このレプリカソースの準備に使用された `CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO` ステートメントの `IGNORE_SERVER_IDS` リストに一致するサーバー ID を持つ `mysql.ndb_apply_status` テーブルの行も、レプリカ独自のサーバー ID を持つ行に加えて、ローカルサーバーからの行とみなされるためです。このリストは、`SHOW REPLICA | SLAVE STATUS` の出力から `Replicate_Ignore_Server_Ids` として取得できます。このリストを取得し、前のバージョンのクエリーと同様に、`@latest` という名前の変数に最大のエポックを選択する、ここに示すクエリーで `ignore_server_ids` に置換することを前提としています:

```
mysqlR> SELECT @latest:=MAX(epoch)
-> FROM mysql.ndb_apply_status
-> WHERE server_id NOT IN (ignore_server_ids);
```

場合によっては、前述のクエリーの `WHERE` 条件に含めるサーバー ID のリストと `server_id IN server_id_list` を使用する方が、より単純または効率的 (あるいはその両方) な場合があります。

2. ステップ 1 で示したクエリーから取得した情報を使用して、ソースクラスタの `ndb_binlog_index` テーブルから対応するレコードを取得します。

次のクエリーを使用して、ソースの `ndb_binlog_index` テーブルから必要なレコードを取得できます:

```
mysqlS> SELECT
-> @file:=SUBSTRING_INDEX(next_file, '/', -1),
-> @pos:=next_position
-> FROM mysql.ndb_binlog_index
-> WHERE epoch >= @latest
-> ORDER BY epoch ASC LIMIT 1;
```

これらは、プライマリレプリケーションチャンネルの失敗以降にソースに保存されたレコードです。ここでは、ユーザー変数 `@latest` を使用して、ステップ 1 で取得した値を表します。もちろん、ある `mysqld` インスタンスが、ほかのサーバーインスタンスに設定されたユーザー変数には直接アクセスできません。これらの値は、手動またはアプリケーションによる 2 番目のクエリーに対する「プラグイン済」である必要があります。

重要

`START REPLICATION | SLAVE` を実行する前に、レプリカ `mysqld` が `--slave-skip-errors=ddl_exist_errors` で起動されていることを確認する必要があります。そうしないと、レプリケーションが重複 DDL エラーで停止する可能性があります。

3. セカンダリレプリカサーバーで次のクエリーを実行して、セカンダリチャンネルを同期化できるようになりました:

```
mysqlR> CHANGE MASTER TO
-> MASTER_LOG_FILE=@file,
-> MASTER_LOG_POS=@pos;

Or from MySQL 8.0.23:
mysqlR> CHANGE REPLICATION SOURCE TO
-> SOURCE_LOG_FILE=@file,
-> SOURCE_LOG_POS=@pos;
```


ここでも、ステップ 2 で取得してステップ 3 で適用した値を表すためにユーザー変数 (この場合は `@file` と `@pos`) を使用しています。実際には、これらの値を手動で挿入するか、関係する両方のサーバーにアクセスできるアプリケーションを使用して挿入する必要があります。

注記

`@file` は `'/var/log/mysql/replication-source-bin.00001'` などの文字列値であるため、SQL またはアプリケーションコードで使用する場合は引用符で囲む必要があります。ただし、`@pos` で表される値は引用符で囲む必要はありません。通常、MySQL は文字列を数字に変換しようとしますが、この場合は例外です。

4. セカンダリレプリカ `mysqld` で適切なコマンドを発行して、セカンダリチャンネルでレプリケーションを開始できるようになりました:

```
mysqlR> START SLAVE;
Or from MySQL 8.0.22:
mysqlR> START REPLICA;
```

セカンダリレプリケーションチャンネルがアクティブになったら、プライマリの不具合を調べて、修復できます。これを行うために必要な正確なアクションは、プライマリチャンネルが失敗した理由によって異なります。

警告

セカンダリレプリケーションチャンネルは、プライマリレプリケーションチャンネルの停止時や停止した場合にのみ、起動されます。複数のレプリケーションチャンネルを同時に実行すると、不要な重複レコードがレプリカに作成される可能性があります。

障害が単一のサーバーに制限されている場合は、理論上、`S` から `R` または `S` から `R` にレプリケートできる必要があります。

23.6.9 NDB Cluster レプリケーションによる NDB Cluster バックアップ

このセクションでは、NDB Cluster レプリケーションを使用してバックアップを作成し、そこから復元する方法について説明します。レプリケーションサーバーは、前に説明したとおりすでに構成されたものとし (セクション 23.6.5 「NDB Cluster のレプリケーションの準備」 およびその直後のセクションを参照してください)。これがすでに行われている場合、バックアップを作成してそのバックアップからリストアする手順は次のとおりです。

1. バックアップを開始するには、2 つの異なる方法があります。
 - 方法 A. この方法では、レプリケーションプロセスを開始する前に、ソースサーバーでクラスタバックアッププロセスが有効になっている必要があります。これを行うには、`my.cnf file` の `[mysql_cluster]` セクションに次の行を含めます。ここで、`management_host` はソースクラスタの NDB 管理サーバーの IP アドレスまたはホスト名、`port` は管理サーバーのポート番号です:

```
ndb-connectstring=management_host[:port]
```

注記

ポート番号は、デフォルトのポート (1186) が使用されていない場合にのみ、指定する必要があります。NDB Cluster でのポートおよびポート割り当ての詳細は、セクション 23.2.3 「NDB Cluster の初期構成」 を参照してください。

この場合、レプリケーションソースで次のステートメントを実行してバックアップを開始できます:

```
shellS> ndb_mgm -e "START BACKUP"
```

- 方法 B. `my.cnf` ファイルで、管理ホストを検出する場所が指定されていない場合、`START BACKUP` コマンドの一部としてこの情報を NDB 管理クライアントに渡すことで、バックアッププロセスを起動できます。ここで示すように、これを実行できます。ここで、`management_host` と `port` は管理サーバーのホスト名とポート番号です。

```
shellS> ndb_mgm management_host:port -e "START BACKUP"
```

前に述べたようなシナリオの場合 (セクション23.6.5 「NDB Cluster のレプリケーションの準備」を参照してください)、これは次のように実行されます。

```
shellS> ndb_mgm rep-source:1186 -e "START BACKUP"
```

- オンラインにするレプリカにクラスタバックアップファイルをコピーします。ソースクラスタの `ndbd` プロセスを実行している各システムにはクラスタバックアップファイルが配置されており、これらのファイルの `all` をレプリカにコピーして、リストアを正常に実行する必要があります。バックアップファイルは、レプリカ管理ホストが存在するコンピュータ上の任意のディレクトリにコピーできます (MySQL および NDB バイナリにそのディレクトリでの読み取りアクセス権がある場合)。この場合、これらのファイルはディレクトリ `/var/BACKUPS/BACKUP-1` にコピーされているものとします。

レプリカクラスタにソースと同じ数の `ndbd` プロセス (データノード) がある必要はありませんが、この数は同じにすることを強くお勧めします。レプリケーションプロセスの早期起動を防ぐには、`--skip-slave-start` オプションを使用してレプリカを起動する必要があります。

- ソースクラスタに存在し、レプリケートするレプリカクラスタにデータベースを作成します。

重要

レプリケートする各データベースに対応する `CREATE DATABASE` (または `CREATE SCHEMA`) ステートメントは、レプリカクラスタ内の各 SQL ノードで実行する必要があります。

- `mysql` クライアントで次のステートメントを使用して、レプリカクラスタをリセットします:

```
mysqlR> RESET SLAVE;
Or from MySQL 8.0.22:
mysqlR> RESET REPLICA;
```

- 各バックアップファイルに対して `ndb_restore` コマンドを順番に使用して、レプリカでクラスタリストアプロセスを開始できるようになりました。最初に、次に示すように、クラスタメタデータをリストアするための `-m` オプションを含める必要があります:

```
shellR> ndb_restore -c replica_host:port -n node-id \
-b backup-id -m -r dir
```

`dir` は、バックアップファイルがレプリカに配置されているディレクトリへのパスです。残りのバックアップファイルに対応する `ndb_restore` コマンドに、`-m` オプションを使用しないでください。

バックアップファイルがディレクトリ `/var/BACKUPS/BACKUP-1` にコピーされている 4 つのデータノード (セクション23.6 「NDB Cluster レプリケーション」の図を参照) を含むソースクラスタからリストアする場合、レプリカで実行されるコマンドの適切な順序は次のようになります:

```
shellR> ndb_restore -c replica-host:1186 -n 2 -b 1 -m \
-r ./var/BACKUPS/BACKUP-1
shellR> ndb_restore -c replica-host:1186 -n 3 -b 1 \
-r ./var/BACKUPS/BACKUP-1
shellR> ndb_restore -c replica-host:1186 -n 4 -b 1 \
-r ./var/BACKUPS/BACKUP-1
shellR> ndb_restore -c replica-host:1186 -n 5 -b 1 -e \
-r ./var/BACKUPS/BACKUP-1
```

重要

エポックがレプリカ `mysql.ndb_apply_status` テーブルに書き込まれるようにするには、この例で `ndb_restore` を最後に起動するときの `-e` (または `--restore-epoch`) オプションが必要です。この情報がないと、レプリカはソースと適切に同期できません。(セクション23.4.23 「`ndb_restore` — NDB Cluster バックアップの復元」を参照してください。)

- ここで、レプリカの `ndb_apply_status` テーブルから最新のエポックを取得する必要があります (セクション23.6.8 「NDB Cluster レプリケーションによるフェイルオーバーの実装」を参照):

```
mysqlR> SELECT @latest:=MAX(epoch)
FROM mysql.ndb_apply_status;
```

7. 前のステップで取得したエポック値として `@latest` を使用すると、次に示すクエリーを使用して、正しいバイナリログファイル `@file` 内の正しい開始位置 `@pos` をソースの `mysql.ndb_binlog_index` テーブルから取得できます:

```
mysqlS> SELECT
-> @file:=SUBSTRING_INDEX(File, '/', -1),
-> @pos:=Position
-> FROM mysql.ndb_binlog_index
-> WHERE epoch >= @latest
-> ORDER BY epoch ASC LIMIT 1;
```

レプリケーショントラフィックが現在存在しない場合は、ソースで `SHOW MASTER STATUS` を実行し、`File` カラムに表示されるすべてのファイルの接尾辞が最大値のファイルの出力の `Position` カラムに表示される値を使用して、この情報を取得できます。この場合、これがどのファイルであるかを判断し、次のステップで手動で、またはスクリプトを使用して出力を解析して名前を指定する必要があります。

8. 前のステップで取得した値を使用して、レプリカ `mysql` クライアントで適切な `CHANGE REPLICATION SOURCE TO` ステートメント (MySQL 8.0.23) または `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 より前) を発行できるようになりました:

```
mysqlR> CHANGE MASTER TO
-> MASTER_LOG_FILE=@file,
-> MASTER_LOG_POS=@pos;

Or from MySQL 8.0.23:
mysqlR> CHANGE REPLICATION SOURCE TO
-> SOURCE_LOG_FILE=@file,
-> SOURCE_LOG_POS=@pos;
```

9. ソースからのデータの読み取りを開始するバイナリログファイルのどの時点からレプリカが認識されるようになったので、次のステートメントを使用してレプリカのレプリケーションを開始できます:

```
mysqlR> START SLAVE;
Or from MySQL 8.0.22:
mysqlR> START REPLICA;
```

2 つ目のレプリケーションチャンネルでバックアップおよびリストアを実行するには、必要に応じて、セカンダリソースおよびレプリカのホスト名と ID をプライマリソースおよびレプリカサーバーのホスト名と ID に置き換え、それらに対して前述のステートメントを実行して、これらのステップを繰り返す必要があります。

クラスタのバックアップの実行とバックアップからのクラスタのリストアに関する詳細は、[セクション 23.5.8 「NDB Cluster のオンラインバックアップ」](#) を参照してください。

23.6.9.1 NDB Cluster レプリケーション: レプリカとソースバイナリログの同期の自動化

前のセクションで説明した多くのプロセスを自動化できます ([セクション 23.6.9 「NDB Cluster レプリケーションによる NDB Cluster バックアップ」](#) 参照してください)。次の Perl スクリプト `reset-replica.pl` は、これを実行する方法の例として機能します。

```
#!/user/bin/perl -w

# file: reset-replica.pl

# Copyright (c) 2005, 2020, Oracle and/or its affiliates. All rights reserved.

# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.

# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to:
# Free Software Foundation, Inc.
# 59 Temple Place, Suite 330
# Boston, MA 02111-1307 USA
#
```

```
# Version 1.1

##### Includes #####

use DBI;

##### Globals #####

my $m_host=";
my $m_port=";
my $m_user=";
my $m_pass=";
my $s_host=";
my $s_port=";
my $s_user=";
my $s_pass=";
my $dbhM=";
my $dbhS=";

##### Sub Prototypes #####

sub CollectCommandPromptInfo;
sub ConnectToDatabases;
sub DisconnectFromDatabases;
sub GetReplicaEpoch;
sub GetSourceInfo;
sub UpdateReplica;

##### Program Main #####

CollectCommandPromptInfo;
ConnectToDatabases;
GetReplicaEpoch;
GetSourceInfo;
UpdateReplica;
DisconnectFromDatabases;

##### Collect Command Prompt Info #####

sub CollectCommandPromptInfo
{
  ### Check that user has supplied correct number of command line args
  die "Usage:\n
    reset-replica >source MySQL host< >source MySQL port< \n
      >source user< >source pass< >replica MySQL host< \n
      >replica MySQL port< >replica user< >replica pass< \n
    All 8 arguments must be passed. Use BLANK for NULL passwords\n"
    unless @ARGV == 8;

  $m_host = $ARGV[0];
  $m_port = $ARGV[1];
  $m_user = $ARGV[2];
  $m_pass = $ARGV[3];
  $s_host = $ARGV[4];
  $s_port = $ARGV[5];
  $s_user = $ARGV[6];
  $s_pass = $ARGV[7];

  if ($m_pass eq "BLANK") { $m_pass = ";}
  if ($s_pass eq "BLANK") { $s_pass = ";}
}

##### Make connections to both databases #####

sub ConnectToDatabases
{
  ### Connect to both source and replica cluster databases

  ### Connect to source
  $dbhM
  = DBI->connect(
    "dbi:mysql:database=mysql:host=$m_host;port=$m_port",
    "$m_user", "$m_pass")
  or die "Can't connect to source cluster MySQL process!
```

```
Error: $DBI::errstr\n";

### Connect to replica
$dbhS
= DBI->connect(
    "dbi:mysql:database=mysql:host=$s_host",
    "$s_user", "$s_pass")
or die "Can't connect to replica cluster MySQL process!
    Error: $DBI::errstr\n";
}

##### Disconnect from both databases #####

sub DisconnectFromDatabases
{
    ### Disconnect from source

    $dbhM->disconnect
    or warn " Disconnection failed: $DBI::errstr\n";

    ### Disconnect from replica

    $dbhS->disconnect
    or warn " Disconnection failed: $DBI::errstr\n";
}

##### Find the last good GCI #####

sub GetReplicaEpoch
{
    $sth = $dbhS->prepare("SELECT MAX(epoch)
        FROM mysql.ndb_apply_status;")
    or die "Error while preparing to select epoch from replica: ",
        $dbhS->errstr;

    $sth->execute
    or die "Selecting epoch from replica error: ", $sth->errstr;

    $sth->bind_col (1, \$epoch);
    $sth->fetch;
    print "\tReplica epoch = $epoch\n";
    $sth->finish;
}

##### Find the position of the last GCI in the binary log #####

sub GetSourceInfo
{
    $sth = $dbhM->prepare("SELECT
        SUBSTRING_INDEX(File, '/', -1), Position
        FROM mysql.ndb_binlog_index
        WHERE epoch > $epoch
        ORDER BY epoch ASC LIMIT 1;")
    or die "Prepare to select from source error: ", $dbhM->errstr;

    $sth->execute
    or die "Selecting from source error: ", $sth->errstr;

    $sth->bind_col (1, \$binlog);
    $sth->bind_col (2, \$binpos);
    $sth->fetch;
    print "\tSource binary log file = $binlog\n";
    print "\tSource binary log position = $binpos\n";
    $sth->finish;
}

##### Set the replica to process from that location #####

sub UpdateReplica
{
    $sth = $dbhS->prepare("CHANGE MASTER TO
        MASTER_LOG_FILE=$binlog,
        MASTER_LOG_POS=$binpos;")
    or die "Prepare to CHANGE MASTER error: ", $dbhS->errstr;
```

```
$sth->execute
  or die "CHANGE MASTER on replica error: ", $sth->errstr;
$sth->finish;
print "\tReplica has been updated. You may now start the replica.\n";
}

# end reset-replica.pl
```

23.6.9.2 NDB Cluster レプリケーションを使用したポイントインタイムリカバリ

ポイントインタイムリカバリ (つまり、特定の時点よりあとに行われたデータ変更のリカバリ) は、サーバーをバックアップが行われた時点の状態に戻す完全バックアップのリストア後に実行されます。「NDB Cluster」および「NDB Cluster」レプリケーションを使用して「NDB Cluster」テーブルの point-in-time リカバリを実行するには、ネイティブの NDB データバックアップ (`ndb_mgm` クライアントで `CREATE BACKUP` を発行して取得) を使用し、`ndb_binlog_index` テーブルを (`mysqldump` を使用して作成したダンプからリストアします)。

NDB Cluster のポイントインタイムリカバリを実行するには、次に示す手順に従う必要があります:

1. `ndb_mgm` クライアントで `START BACKUP` コマンドを使用して、クラスタ内のすべての NDB データベースをバックアップします (セクション23.5.8「NDB Cluster のオンラインバックアップ」を参照してください)。
2. その後の時点でクラスタをリストアする前に、`mysql.ndb_binlog_index` テーブルのバックアップを作成します。このタスクに `mysqldump` を使用することが、おそらくもっとも簡単です。このときに、バイナリログファイルもバックアップします。

このバックアップは、必要に応じて定期的 (等間隔で 1 時間ごとなど) に更新してください。

3. (重大な障害またはエラーが発生します。)
4. 最新の既知の良好なバックアップを探します。
5. データノードのファイルシステムをクリアします (`ndbd --initial` または `ndbmtnd --initial` を使用します)。

注記

NDB 8.0.21 以降、「ディスクデータ」テーブルスペースおよびログファイルは `--initial` によって削除されます。以前は、これらを手動で削除する必要がありました。

6. `mysql.ndb_binlog_index` テーブルで `DROP TABLE` または `TRUNCATE TABLE` を使用します。
7. `ndb_restore` を実行して、すべてのデータをリストアします。`ndb_apply_status` テーブルが正しく移入されるように、`ndb_restore` の実行時に `--restore-epoch` オプションを含める必要があります。(詳細については、セクション23.4.23「`ndb_restore` — NDB Cluster バックアップの復元」を参照してください。)
8. 必要に応じて、`mysqldump` の出力から `ndb_binlog_index` テーブルをリストアしたり、バックアップからバイナリログファイルをリストアしたりします。
9. 最後に適用されたエポック (すなわち、`ndb_apply_status` テーブルの `epoch` カラムの最大値) をユーザー変数 `@LATEST_EPOCH` として検索します (重要)。

```
SELECT @LATEST_EPOCH:=MAX(epoch)
FROM mysql.ndb_apply_status;
```

10. `ndb_binlog_index` テーブルの `@LATEST_EPOCH` に対応する、最新のバイナリログファイル (`@FIRST_FILE`) と位置 (`Position` カラム値) をこのファイル内で検索します。

```
SELECT Position, @FIRST_FILE:=File
FROM mysql.ndb_binlog_index
WHERE epoch > @LATEST_EPOCH ORDER BY epoch ASC LIMIT 1;
```

11. `mysqlbinlog` を使用して、障害が発生した時点まで、特定のファイルと位置からバイナリログイベントを再現します。(セクション4.6.8「`mysqlbinlog` — バイナリログファイルを処理するためのユーティリティー」を参照してください。)

バイナリログ、レプリケーション、および増分リカバリに関する詳細は、セクション7.5「Point-in-Time (増分) リカバリ」も参照してください。

23.6.10 NDB Cluster レプリケーション: 双方向および循環レプリケーション

NDB Cluster は、2 つのクラスタ間の双方向レプリケーションや、任意の数のクラスタ間の循環レプリケーションに使用できます。

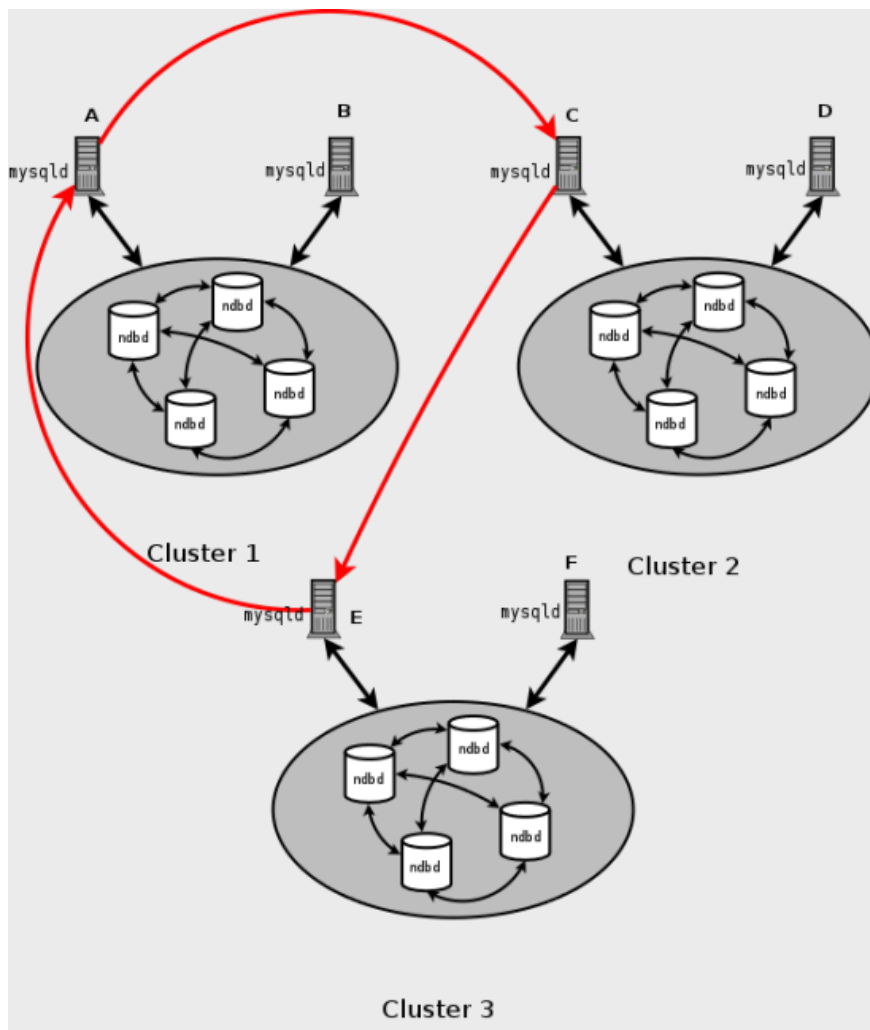
循環レプリケーションの例。 次のいくつかの段落では、クラスタ 1 がクラスタ 2 のレプリケーションソースとして機能し、クラスタ 3 がクラスタ 1 のソースとして機能する 3 つの NDB Cluster 番号 1、2、および 3 を含むレプリケーション設定の例を検討します。 各クラスタは 2 つの SQL ノードを持ち、SQL ノード A と B はクラスタ 1 に属し、SQL ノード C と D はクラスタ 2 に属し、SQL ノード E と F はクラスタ 3 に属しています。

これらのクラスタを使用する循環レプリケーションは、次の条件を満たすかぎり、サポートされます。

- すべてのソースおよびレプリカの SQL ノードが同じです。
- ソースおよびレプリカとして機能するすべての SQL ノードは、`log_slave_updates` システム変数を有効にして起動されます。

このタイプの循環レプリケーションのセットアップは、次の図に示すとおりです。

図 23.36 すべてのソースがレプリカである NDB Cluster 循環レプリケーション

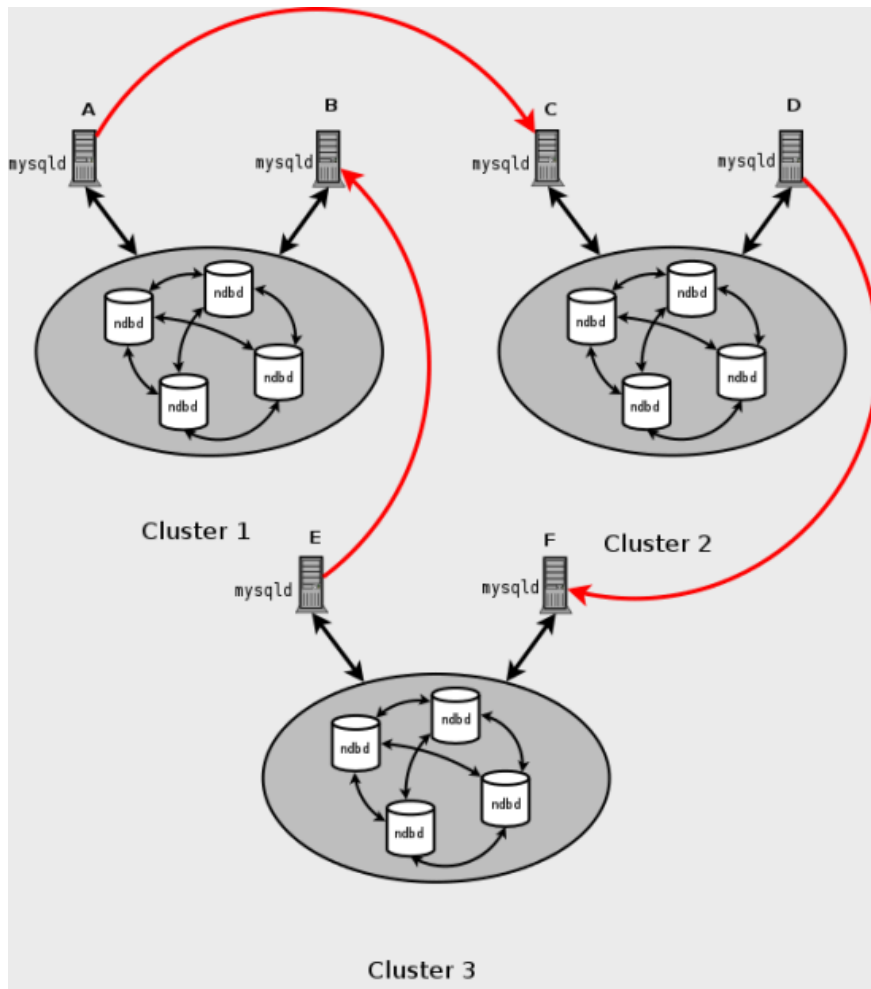


このシナリオでは、クラスタ 1 の SQL ノード A はクラスタ 2 の SQL ノード C に複製し、SQL ノード C はクラスタ 3 の SQL ノード E に複製し、SQL ノード E は SQL ノード A に複製します。つまり、レプリケーションソースお

よびレプリカとして使用されるすべての SQL ノードは、レプリケーション線 (図の曲線の矢印で示されています) によって直接接続されます。

次に示すように、すべてのソース SQL ノードがレプリカであるわけではないような方法で循環レプリケーションを設定することもできます:

図 23.37 すべてのソースがレプリカではない NDB Cluster 循環レプリケーション

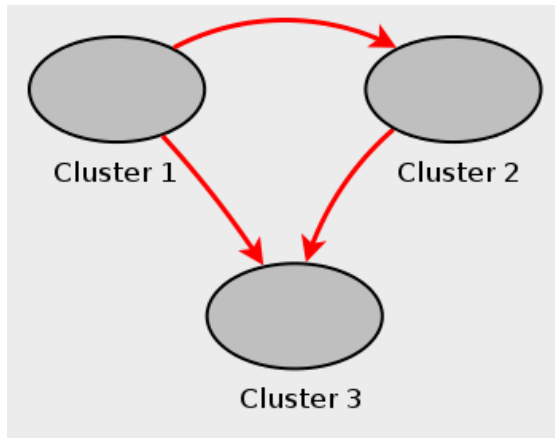


この場合、各クラスタ内の異なる SQL ノードがレプリケーションソースおよびレプリカとして使用されます。[log_slave_updates](#) システム変数を有効にして SQL ノードを起動しないでください。NDB Cluster のこのタイプの循環レプリケーションスキームでは、レプリケーションの行 (図の曲線矢印で再度示されています) が不連続である可能性があります。まだ徹底的にテストされていないため、実験的のみならず必要があることに注意してください。

NDB ネイティブのバックアップおよびリストアを使用したレプリカクラスタの初期化。循環レプリケーションを設定する場合、ある NDB Cluster で管理クライアントの [START BACKUP](#) コマンドを使用してバックアップを作成し、[ndb_restore](#) を使用してこのバックアップを別の NDB Cluster に適用することによって、レプリカクラスタを初期化できます。これにより、レプリカとして機能する 2 つ目の NDB Cluster SQL ノードにバイナリログが自動的に作成されることはありません。バイナリログを作成するには、その SQL ノードで [SHOW TABLES](#) ステートメントを発行する必要があります。これは、[START REPLICA | SLAVE](#) を実行する前に行う必要があります。これは既知の問題です。

マルチソースフェイルオーバーの例。このセクションでは、サーバー ID 1、2、および 3 を持つ 3 つの NDB Cluster を持つマルチソース NDB Cluster レプリケーション設定でのフェイルオーバーについて説明します。このシナリオでは、クラスタ 1 はクラスタ 2 および 3 に複製し、クラスタ 2 もクラスタ 3 に複製します。この関係はここで示しております。

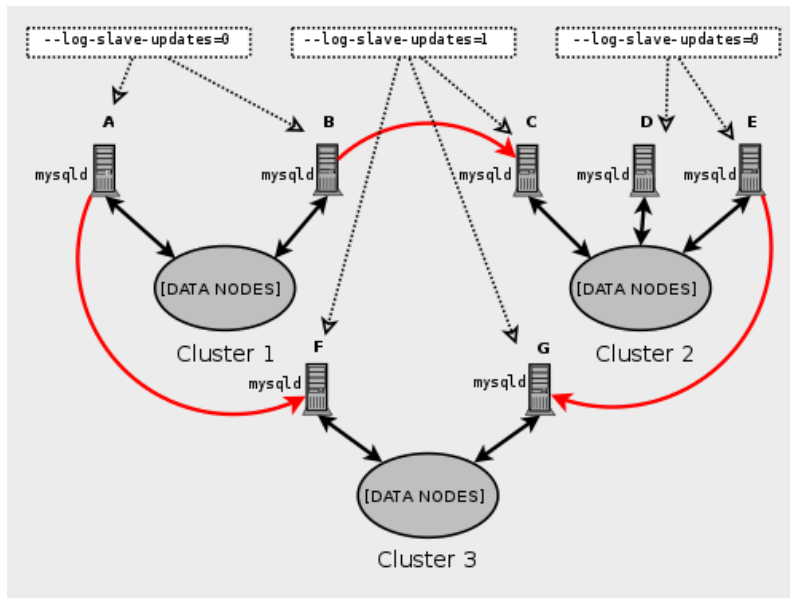
図 23.38 3 つのソースを持つ NDB Cluster マルチマスターレプリケーション



つまり、データはクラスタ 1 からクラスタ 3 へ異なる 2 つのルートを通じて (直接、およびクラスタ 2 経由で) 複製されます。

マルチソースレプリケーションに関与するすべての MySQL サーバーがソースとレプリカの両方として動作するわけではなく、特定の NDB Cluster がレプリケーションチャンネルごとに異なる SQL ノードを使用する可能性があります。このようなケースを次に示します。

図 23.39 NDB Cluster マルチソースレプリケーション (MySQL Servers を使用)



レプリカとして機能する MySQL サーバーは、`log_slave_updates` システム変数を有効にして実行する必要があります。また、どの `mysqld` プロセスでこのオプションが必要か、前の図に示されています。

注記

`log_slave_updates` システム変数を使用しても、レプリカとして実行されていないサーバーには影響しません。

複製しているクラスタの 1 つが停止した場合、フェイルオーバーの必要性が生じます。この例では、クラスタ 1 のサービスが失われ、そのためにクラスタ 3 がクラスタ 1 の更新の 2 つのソースを失うケースを検討します。NDB Cluster 間のレプリケーションは非同期であるため、クラスタ 1 から直接発生したクラスタ 3 の更新が、クラスタ 2 を介して受信した更新よりも新しいことは保証されません。クラスタ 1 からの更新に関して、クラスタ 3 が確実にク

ラスト 2 に追いつくことで、これに対処できます。すなわち、MySQL サーバーに関して、未処理の更新を MySQL サーバー C からサーバー F に複製する必要があります。

サーバー C で、次のクエリーを実行します。

```
mysqlC> SELECT @latest:=MAX(epoch)
-> FROM mysql.ndb_apply_status
-> WHERE server_id=1;

mysqlC> SELECT
-> @file:=SUBSTRING_INDEX(File, '/', -1),
-> @pos:=Position
-> FROM mysql.ndb_binlog_index
-> WHERE orig_epoch >= @latest
-> AND orig_server_id = 1
-> ORDER BY epoch ASC LIMIT 1;
```

注記

適切なインデックスを `ndb_binlog_index` テーブルに追加することで、このクエリーのパフォーマンスを向上できるため、フェイルオーバー時間が大幅に短縮される可能性があります。詳細については、[セクション23.6.4「NDB Cluster レプリケーションスキーマおよびテーブル」](#)を参照してください。

`@file` および `@pos` の値を手動でサーバー C からサーバー F にコピーをします (またはアプリケーションで同様に実行させます)。次に、サーバー F で、次の `CHANGE REPLICATION SOURCE TO` ステートメント (MySQL 8.0.23 の場合) または `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 の場合) を実行します:

```
mysqlF> CHANGE MASTER TO
-> MASTER_HOST = 'serverC'
-> MASTER_LOG_FILE=@file;
-> MASTER_LOG_POS=@pos;

Or from MySQL 8.0.23:
mysqlF> CHANGE REPLICATION SOURCE TO
-> SOURCE_HOST = 'serverC'
-> SOURCE_LOG_FILE=@file;
-> SOURCE_LOG_POS=@pos;
```

これが完了したら、MySQL サーバー F で `START REPLICAS | SLAVE` ステートメントを発行できます。これにより、サーバー B から発生した欠落している更新がサーバー F にレプリケートされます。

`CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO` ステートメントは、サーバー ID のコンマ区切りリストを使用し、対応するサーバーから発生したイベントを無視する `IGNORE_SERVER_IDS` オプションもサポートしています。詳細については、[セクション13.4.2.1「CHANGE MASTER TO ステートメント」](#) および [セクション13.7.7.36「SHOW SLAVE | REPLICAS STATUS ステートメント」](#) を参照してください。このオプションと `ndb_log_apply_status` 変数との相互作用の詳細は、[セクション23.6.8「NDB Cluster レプリケーションによるフェイルオーバーの実装」](#) を参照してください。

23.6.11 NDB Cluster レプリケーションの競合解決

複数のソース (循環レプリケーションを含む) を含むレプリケーション設定を使用する場合、異なるソースがレプリカ上の同じ行を異なるデータで更新しようとする可能性があります。NDB Cluster レプリケーションでの競合解決では、特定のソースの更新をレプリカに適用するかどうかを決定するためにユーザー定義の解決カラムを使用できるようにすることによって、このような競合を解決する手段が提供されます。

NDB Cluster (`NDB$OLD()`, `NDB$MAX()`, `NDB$MAX_DELETE_WIN()`) でサポートされている一部のタイプの競合解決は、このユーザー定義カラムを「timestamp」カラムとして実装します (ただし、このセクションで後述するように、そのタイプを `TIMESTAMP` にすることはできません)。このタイプの競合解決は、常に、トランザクションごとではなく、行ごとに適用されます。エポックベースの競合解決関数 `NDB$EPOCH()` および `NDB$EPOCH_TRANS()` は、エポックが複製された順番を比較します (このため、これらの関数はトランザクション型です)。このセクションの後半で説明するように、競合が発生したときにレプリカの解決カラムの値を比較するには、様々な方法を使用できます。使用する方法はテーブルごとに設定できます。

更新を適用するかどうかを判断するときに解決関数で適切な選択を行えるように、解決カラムに適切な値で正しく移入されることを確認することは、アプリケーションの責任になります。

要件. 競合解消の準備は、ソースとレプリカの両方で行う必要があります。これらのタスクは次のリストで説明します。

- バイナリログを書き込むソースで、送信するカラム (すべてのカラム、または更新されたカラムのみ) を決定する必要があります。MySQL Server でこれを行うには、全体的には `mysqld` 起動オプション `--ndb-log-updated-only` (このセクションの後半で説明します) を適用し、テーブル単位には `mysql.ndb_replication` テーブルのエントリによって実行します (`ndb_replication` システムテーブルを参照してください)。

注記

非常に大きなカラム (`TEXT` や `BLOB` カラムなど) を持つテーブルをレプリケートする場合、`--ndb-log-updated-only` は、バイナリログのサイズを小さくし、`max_allowed_packet` の超過によるレプリケーションの失敗を回避するためにも役立つことがあります。

この問題に関する詳細は、[セクション17.5.1.20「レプリケーションとmax_allowed_packet」](#)を参照してください。

- レプリカで、適用する競合解消のタイプ (「最新のタイムスタンプ優先」、「同じタイムスタンプを優先」、「プライマリ優先」、「プライマリ受注、トランザクションの完了」またはなし) を決定する必要があります。これは、`mysql.ndb_replication` システムテーブルを使用して、テーブルごとに行われます (`ndb_replication` システムテーブルを参照してください)。
- NDB Cluster は、読み取りの競合検出もサポートしています。つまり、あるクラスタ内の特定の行の読み取りと別のクラスタ内の同じ行の更新または削除の間の競合を検出します。これには、レプリカで `ndb_log_exclusive_reads` を 1 に設定して取得した排他的読み取りロックが必要です。競合している読み取りによって読み取られたすべての行のログが、例外テーブルに記録されます。詳細は、[読み取り競合の検出と解決](#)を参照してください。

タイムスタンプベースの競合解決に関数 `NDB$OLD()`、`NDB$MAX()`、および `NDB$MAX_DELETE_WIN()` を使用する場合、更新を「タイムスタンプ」カラムとして指定するために使用されるカラムを参照するのが一般的です。ただし、このカラムのデータ型は `TIMESTAMP` にしないでください。このデータ型は `INT (INTEGER)` または `BIGINT` にしてください。また、「timestamp」カラムは `UNSIGNED` および `NOT NULL` にしてください。

このセクションで後述する `NDB$EPOCH()` および `NDB$EPOCH_TRANS()` 関数は、プライマリ NDB Cluster とセカンダリ NDB Cluster に適用されるレプリケーションエポックの相対的な順序を比較することで機能し、タイムスタンプは使用しません。

ソースカラムコントロール. 「前」のイメージおよび「後」のイメージ (すなわち、更新が適用される前と後のテーブルの状態) に関して、更新の操作を確認できます。通常、主キーを使用してテーブルを更新する場合、ビフォアイメージは重要ではありません。ただし、更新された値をレプリカで使用するかどうかを更新ごとに決定する必要がある場合は、両方のイメージがソースバイナリログに書き込まれていることを確認する必要があります。このセクションの後半で説明するように、これは `mysqld` の `--ndb-log-update-as-write` オプションで実行されます。

重要

行全体のロギングを行うか、更新されたカラムだけのロギングを行うかは、MySQL サーバーが起動されたときに決まり、オンラインでは変更できません。異なるロギングオプションを使用して、`mysqld` を再起動するか、新しい `mysqld` インスタンスを起動する必要があります。

すべてまたは一部の行のロギング (`--ndb-log-updated-only` オプション)

コマンド行形式	<code>--ndb-log-updated-only[={OFF ON}]</code>
システム変数	<code>ndb_log_updated_only</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

競合解決のために、行のログを取る基本的な方法は 2 つあり、その方法は `mysqld` の `--ndb-log-updated-only` オプションを設定すると指定されます。

- 行全体のログを取得します
- 更新されたカラムデータ (すなわち、値が設定されたカラムデータ) だけのログを取ります。この値が実際に変更されたかどうかには関係しません。これはデフォルトの動作です。

通常、更新されたカラムのみのログを取るだけで十分 (しかも効率的) です。ただし、すべての行のログを取る必要がある場合、`--ndb-log-updated-only` を `0` または `OFF` に設定すると、これを実行できます。

--ndb-log-update-as-write オプション: 変更されたデータを更新としてログを取得

コマンド行形式	<code>--ndb-log-update-as-write[={OFF ON}]</code>
システム変数	<code>ndb_log_update_as_write</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

MySQL Server の `--ndb-log-update-as-write` オプションの設定では、ロギングが「前」のイメージがある状態で実行されるか、ない状態で実行されるかを指定します。競合解消は MySQL Server 更新ハンドラで行われるため、更新が書き込みではなく更新であるようにレプリケーションソースによって実行されるロギングを制御する必要があります。つまり、既存の行を置き換える場合でも、更新は既存の行の書き込みではなく既存の行の変更として処理されます。このオプションはデフォルトではオンです。つまり、更新は書き込みとして処理されます。つまり、更新はデフォルトで、`update_row` イベントとしてではなくバイナリログに `write_row` イベントとして書き込まれます。

このオプションを無効にするには、`--ndb-log-update-as-write=0` または `--ndb-log-update-as-write=OFF` を使用してソース `mysqld` を起動します。NDB テーブルから異なるストレージエンジンを使用するテーブルに複製する場合は、これを行う必要があります。詳細は、[NDB から別のストレージエンジンへのレプリケーション](#)および[NDB から非トランザクションストレージエンジンへのレプリケーション](#)を参照してください。

競合解決の制御。 通常、競合解決は競合が発生する可能性のあるサーバーで有効になっています。ロギング方法の選択と同様に、`mysql.ndb_replication` テーブル内のエントリによって有効化されます。

ndb_replication システムテーブル。 競合解消を有効にするには、使用する競合解消タイプおよび方法に応じて、ソースまたはレプリカ (あるいはその両方) の `mysql` システムデータベースに `ndb_replication` テーブルを作成する必要があります。このテーブルは、ロギングと競合解決関数をテーブル単位に制御するために使用され、レプリケーションに関与する 1 つの行単位テーブルを持ちます。`ndb_replication` が作成され、競合を解決すべきサーバー上の制御情報が格納されます。レプリカ上でデータをローカルに変更できる単純なソース-レプリケーション設定では、これは通常レプリカです。双方向レプリケーションなどのより複雑なレプリケーションスキームでは、これは通常、関係するすべてのソースです。`mysql.ndb_replication` の各行は複製されるテーブルに対応し、対象テーブルに対するログの取得方法と競合の解決方法 (すなわち、もし競合が発生した場合に、どの競合解決関数を使用するか) を指定します。`mysql.ndb_replication` テーブルの定義は次のとおりです。

```
CREATE TABLE mysql.ndb_replication (  
  db VARBINARY(63),  
  table_name VARBINARY(63),  
  server_id INT UNSIGNED,  
  binlog_type INT UNSIGNED,  
  conflict_fn VARBINARY(128),  
  PRIMARY KEY USING HASH (db, table_name, server_id)  
) ENGINE=NDB  
PARTITION BY KEY(db,table_name);
```

このテーブルのカラムは、次のいくつかの段落で説明します。

db. 複製されるテーブルを含むデータベースの名前です。ワイルドカード `_` と `%` のどちらか、またはその両方を、データベース名の一部として使用できます。マッチングは、`LIKE` 演算子に対して実装されたマッチングに似ています。

table_name. 複製されるテーブルの名前です。テーブル名に、ワイルドカード `_` と `%` のどちらか、またはその両方を含めることができます。マッチングは、`LIKE` 演算子に対して実装されたマッチングに似ています。

server_id. テーブルが存在する MySQL インスタンス (SQL ノード) の一意のサーバー ID です。

binlog_type. 使用されるバイナリロギングのタイプです。これは、次の表に示すように指定されます。

表 23.69 binlog_type 値 (内部値と説明を含む)

値	内部値	説明
0	<code>NBT_DEFAULT</code>	サーバーのデフォルトを使用します
1	<code>NBT_NO_LOGGING</code>	バイナリログにこのテーブルの記録を行いません
2	<code>NBT_UPDATED_ONLY</code>	更新された属性のみが記録されます
3	<code>NBT_FULL</code>	更新されない場合でも、行全体を記録します (MySQL サーバーのデフォルトの動作)
4	<code>NBT_USE_UPDATE</code>	(<code>NBT_UPDATED_ONLY_USE_UPDATE</code> および <code>NBT_FULL_USE_UPDATE</code> の値の生成のみを行い、単独では使用しません)
5	[Not used]	---
6	<code>NBT_UPDATED_ONLY_USE_UPDATE</code> <code>NBT_UPDATED_ONLY</code> <code>NBT_USE_UPDATE</code> に相当)	値が変更されていない場合でも、更新された属性を使用します
7	<code>NBT_FULL_USE_UPDATE</code> <code>NBT_FULL</code> <code>NBT_USE_UPDATE</code> に相当)	値が変更されていない場合でも、行全体を使用します

conflict_fn. 適用される競合解決関数です。この関数は、次のリストに示されたいずれかの関数として指定する必要があります。

- `NDB$OLD(column_name)`
- `NDB$MAX(column_name)`
- `NDB$MAX_DELETE_WIN()`
- `NDB$EPOCH()` および `NDB$EPOCH_TRANS()`
- `NDB$EPOCH_TRANS()`
- `NDB$EPOCH2()`
- `NDB$EPOCH2_TRANS()`
- `NULL`: 競合解消が対応するテーブルに使用されないことを示します。

これらの関数は、次のいくつかのパラグラフで説明します。

`NDB$OLD(column_name)`. `column_name` の値がソースとレプリカの両方で同じである場合、更新が適用されます。それ以外の場合、更新はレプリカに適用されず、例外がログに書き込まれます。これは、次の擬似コードで説明します。

```
if (source_old_column_value == replica_current_column_value)
    apply_update();
else
    log_exception();
```

この関数は「同じ値が優先」競合解決に使用されます。このタイプの競合解決では、誤ったソースからのレプリカに更新が適用されないようにします。

重要

この関数では、ソースのビフォアイメージのカラム値が使用されます。

`NDB$MAX(column_name)`。ソースからの特定の行の「timestamp」カラム値がレプリカの値より大きい場合は適用され、それ以外の場合はレプリカに適用されません。これは、次の擬似コードで説明します。

```
if (source_new_column_value > replica_current_column_value)
    apply_update();
```

この関数は「もっとも大きいタイムスタンプが優先」競合解決に使用できます。このタイプの競合解決では、競合が発生した場合、最後に更新された行のバージョンが存続するバージョンになります。

重要

この関数では、アフターイメージのソースのカラム値が使用されます。

`NDB$MAX_DELETE_WIN()`。これは `NDB$MAX()` のバリエーションです。削除操作に使用できるタイムスタンプがないため、`NDB$MAX()` を使用した削除は実際には `NDB$OLD` として処理されますが、一部のユースケースでは最適ではありません。`NDB$MAX_DELETE_WIN()` では、ソースから取得された既存の行を追加または更新する特定の行の「timestamp」カラム値がレプリカの値より大きい場合、その値が適用されます。ただし、削除操作は常に値が高いものとして処理されます。これは、次の擬似コードで説明します。

```
if ((source_new_column_value > replica_current_column_value)
    ||
    operation_type == "delete")
    apply_update();
```

この関数は「もっとも大きいタイムスタンプ、削除が優先」競合解決に使用できます。このタイプの競合解決では、競合が発生した場合、削除された行のバージョン、または(そうでなければ)最後に更新された行のバージョンが存続するバージョンになります。

注記

`NDB$MAX()` と同様に、ソースのアフターイメージのカラム値は、この関数で使用される値です。

`NDB$EPOCH()` および `NDB$EPOCH_TRANS()`。 `NDB$EPOCH()` 関数は、レプリカクラスタでレプリケートされたエポックが適用される順序を、レプリカで発生した変更と比較して追跡します。この相対的な順序付けを使用して、レプリカで発生した変更がローカルで発生した変更と同時にわれ、競合が発生する可能性があるかどうかを判断します。

`NDB$EPOCH()` の説明で従う内容のほとんどは、`NDB$EPOCH_TRANS()` にも適用されます。例外は本文に記載されています。

`NDB$EPOCH()` は非対称であり、双方向レプリケーション構成(「active-active」レプリケーションとも呼ばれる)内のいずれかの NDB Cluster で動作します。ここで、プライマリとして動作するクラスタと、セカンダリとして動作するもう一方のクラスタについて言及します。プライマリ上のレプリカは競合の検出および処理を担当しますが、セカンダリ上のレプリカは競合の検出または処理には関与しません。

プライマリ上のレプリカは、競合を検出すると、これらを補正するためにイベントを独自のバイナリログに注入します。これにより、セカンダリ NDB Cluster が最終的にプライマリと再配置されるため、プライマリとセカンダリが相違しなくなります。この補正および再編成メカニズムでは、プライマリ NDB Cluster がセカンダリとの競合を常に優先する必要があります。つまり、競合が発生した場合、プライマリ変更はセカンダリからの変更ではなく常に使用されます。この「プライマリが常に勝つ」ルールには次の意味が込められています。

- プライマリでコミットされたデータを変更する操作は完全に永続的であり、競合検出および解決によって元に戻されたりロールバックされることはありません。
- プライマリから読み取られたデータには一貫性があります。プライマリ(ローカルまたはレプリカから)でコミットされた変更は、後で元に戻されません。
- セカンダリでデータを変更する操作は、競合が発生しているとプライマリが判断した場合、あとで取り消される可能性があります。

- セカンダリで読み取られた各行は、常に自己矛盾がなく、各行は、セカンダリでコミットされた状態、またはプライマリでコミットされた状態のいずれかを常に反映しています。
- セカンダリで読み取られた一連の行は、任意の時点で必ずしも一貫していません。 `NDB$EPOCH_TRANS()` の場合、これは一時的な状態であり、`NDB$EPOCH()` の場合は、永続的な状態となることがあります。
- 競合のない十分な長さの期間を想定すると、セカンダリ NDB Cluster 上のすべてのデータ (最終的に) はプライマリデータと整合性がとれます。

`NDB$EPOCH()` および `NDB$EPOCH_TRANS()` では、競合を検出するためにユーザースキーマを修正したり、アプリケーションを変更したりする必要はありません。ただし、システム全体が指定された制限内で動作することを検証するために、使用するスキーマ、および使用するアクセスパターンを考慮する必要があります。

`NDB$EPOCH()` および `NDB$EPOCH_TRANS()` の各関数は、オプションのパラメータを取ることができます。これはエポックの下位 32 ビットを表すために使用するビット数で、次に示すように計算された値以下に設定する必要があります:

```
CEIL( LOG2( TimeBetweenGlobalCheckpoints / TimeBetweenEpochs ), 1)
```

これらの構成パラメータがデフォルト値 (それぞれ、2000 および 100 ミリ秒) である場合、値は 5 バイトになり、デフォルト値 (6) で十分です。ただし、ほかの値が `TimeBetweenGlobalCheckpoints`、`TimeBetweenEpochs`、またはその両方に使用される場合を除きます。値が小さすぎると誤判定となる可能性があります。一方、値が大きすぎると、データベース内に無駄なスペースが増える可能性があります。

競合する行のエントリは、このセクションの他の場所で説明されているのと同じ例外テーブルスキーマルールに従って定義されている場合、`NDB$EPOCH()` と `NDB$EPOCH_TRANS()` の両方によって関連する例外テーブルに挿入されます (`NDB$OLD(column_name)` を参照)。使用するデータテーブルを作成する前に、例外テーブルを作成する必要があります。

`NDB$EPOCH()` および `NDB$EPOCH_TRANS()` は、このセクションで説明したほかの競合検出関数と同様に、`mysql.ndb_replication` テーブルに関連するエントリを含めることで起動されます (`ndb_replication` システムテーブルを参照してください)。このシナリオでのプライマリ NDB Cluster とセカンダリ NDB Cluster の役割は、`mysql.ndb_replication` テーブルエントリによって完全に決定されます。

`NDB$EPOCH()` および `NDB$EPOCH_TRANS()` で使用される競合検出アルゴリズムは非対称であるため、プライマリレプリカとセカンダリレプリカの `server_id` エントリには異なる値を使用する必要があります。

`NDB$EPOCH()` または `NDB$EPOCH_TRANS()` を使用して競合をトリガーするには、`DELETE` 操作間の競合のみでは不十分であり、エポック内の相対的な配置は関係ありません。(Bug #18459944)

競合検出ステータス変数。競合検出の監視には、いくつかのステータス変数を使用できます。このレプリカが `Ndb_conflict_fn_epoch` システムステータス変数の現在の値から最後に再起動されてから、`NDB$EPOCH()` によって競合が検出された行数を確認できます。

`Ndb_conflict_fn_epoch_trans` には、`NDB$EPOCH_TRANS()` によって直接競合が検出された行数が用意されています。`Ndb_conflict_fn_epoch2` および `Ndb_conflict_fn_epoch2_trans` には、`NDB$EPOCH2()` および `NDB$EPOCH2_TRANS()` によって競合が検出された行数がそれぞれ表示されます。実際に再編成された行数 (他の競合する行と同じトランザクションへのメンバーシップまたは同じトランザクションへの依存関係の影響を含む) は、`Ndb_conflict_trans_row_reject_count` によって指定されます。

詳細については、[NDB Cluster ステータス変数](#)を参照してください。

`NDB$EPOCH()` の制約。 `NDB$EPOCH()` を使用して競合検出を実行する場合、現在次の制限が適用されます:

- NDB Cluster のエポック境界を使用して競合が検出され、`TimeBetweenEpochs` に比例した粒度 (デフォルト: 100 milliseconds)。最小競合ウィンドウは、両方のクラスタの同じデータへの並列更新が常に競合を報告する最小時間です。これは、常にゼロでない時間であり、 $2 * (\text{latency} + \text{queueing} + \text{TimeBetweenEpochs})$ にほぼ比例します。これは、`TimeBetweenEpochs` にデフォルトを仮定し、またクラスタ間の待機時間 (およびキューイング遅延) を無視して、最小競合ウィンドウが約 200 ミリ秒であることを意味します。この最小ウィンドウは、予期されたアプリケーション「競争」パターンを調べるときに考慮してください。
- `NDB$EPOCH()` および `NDB$EPOCH_TRANS()` 関数を使用するテーブルには、追加ストレージが必要です。関数に渡される値によって、1 行当たり 1 ビットから 32 ビットのスペースが必要です。

- 削除操作の競合は、プライマリとセカンダリの間で相違につながる可能性があります。行が両方のクラスタ上で同時に削除される場合、競合は検出できますが、行が削除されるために競合は記録されません。つまり、後続の再編成操作の伝播中にさらに競合が検出されないため、相違が発生する可能性があります。

削除は外部シリアライズするか、1つのクラスタだけにルーティングしてください。また、行の削除の間の競合を追跡できるように、このような削除および削除に続く挿入でトランザクションごとに個々の行を更新してください。これには、アプリケーションの変更が必要となる場合があります。

- 競合検出に `NDB$EPOCH()` または `NDB$EPOCH_TRANS()` を使用している場合、現在、発生「active-active」構成内の2つのNDB Clusterのみがサポートされます。
- `BLOB` または `TEXT` カラムを持つテーブルは、現在 `NDB$EPOCH()` または `NDB$EPOCH_TRANS()` ではサポートされていません。

`NDB$EPOCH_TRANS()`。 `NDB$EPOCH_TRANS()` は、`NDB$EPOCH()` 関数を拡張します。「プライマリがすべてに優先」ルール (`NDB$EPOCH()` および `NDB$EPOCH_TRANS()` を参照してください) を使用する同じ方法で競合が検出され、処理されますが、競合が発生した同じトランザクションで更新されたその他の行も競合していると見なされ、という条件が追加されます。つまり、`NDB$EPOCH()` がセカンダリの競合している各行を再編成するのに対して、`NDB$EPOCH_TRANS()` は競合しているトランザクションを再編成します。

また、競合しているトランザクションへの依存が検出されたトランザクションも競合していると見なされ、これらの依存関係はセカンダリクラスタのバイナリログの内容によって判断されます。バイナリログにはデータ変更操作だけ (挿入、更新、削除) が含まれるため、重複するデータの変更だけがトランザクション間の依存関係の判断に使用されます。

`NDB$EPOCH_TRANS()` には、`NDB$EPOCH()` と同じ条件および制限が適用され、さらに、すべてのトランザクションIDがセカンダリバイナリログ (`--ndb-log-transaction-id` オプション) に記録され、可変オーバーヘッド (行あたり最大13バイト) が追加される必要があります。非推奨の `log_bin_use_v1_row_events` システム変数 (デフォルトは `OFF`) は、`NDB$EPOCH_TRANS()` を使用した `ON` に設定しないでください。

`NDB$EPOCH()` および `NDB$EPOCH_TRANS()` を参照してください。

ステータス情報。 サーバーステータス変数 `Ndb_conflict_fn_max` は、`mysqld` が最後に起動されてから、「もっとも大きいタイムスタンプが優先」競合解決によって現在のSQLノードに行が適用されなかった回数のカウンタを示します。

指定された `mysqld` が最後に再起動されたあとに「同じタイムスタンプが優先」競合解決の結果として行が挿入されなかった回数は、グローバルステータス変数 `Ndb_conflict_fn_old` によって取得されます。`Ndb_conflict_fn_old` をインクリメントすること以外に、このセクションの後半の説明のように、使用されなかった行の主キーは例外テーブルに挿入されます。

`NDB$EPOCH2()`。 `NDB$EPOCH2()` 関数は `NDB$EPOCH()` と似ていますが、`NDB$EPOCH2()` には双方向レプリケーショントポロジでの削除処理が用意されています。このシナリオでは、`ndb_slave_conflict_role` システム変数を各ソース (通常は各 `PRIMARY`、`SECONDARY` のいずれか) で適切な値に設定することで、プライマリロールとセカンダリロールが2つのソースに割り当てられます。これが行われると、セカンダリによる変更はプライマリによってセカンダリに反映され、セカンダリは条件付きで適用されます。

`NDB$EPOCH2_TRANS()`。 `NDB$EPOCH2_TRANS()` は、`NDB$EPOCH2()` 関数を拡張します。競合は同じ方法で検出および処理され、プライマリロールとセカンダリロールがレプリケートクラスタに割り当てられますが、競合が発生した同じトランザクションで更新された他の行も競合とみなされる追加の条件があります。つまり、`NDB$EPOCH2()` ではセカンダリで競合する個々の行が再編成され、`NDB$EPOCH_TRANS()` では競合するトランザクションが再編成されます。

`NDB$EPOCH()` および `NDB$EPOCH_TRANS()` では、最後に変更されたエポックごとに行ごとに指定されたメタデータを使用して、セカンダリからの受信レプリケートされた行変更がローカルでコミットされた変更と並行しているかどうかをプライマリで判断します。同時変更は競合しているとみなされ、セカンダリの例外テーブルの更新および再編成が行われます。プライマリで行が削除されると問題が発生するため、レプリケートされた操作が競合するかどうかを判断するために使用可能な最終変更エポックはなくなります。つまり、競合する削除操作は検出されません。これにより相違が生じる可能性があります。たとえば、一方のクラスタでの削除は、他方での削除および挿入と並行しているため、`NDB$EPOCH()` および `NDB$EPOCH_TRANS()` の使用時に削除操作を一方のクラスタにのみルーティングできます。

`NDB$EPOCH2()` では、削除 - 削除の競合を無視し、潜在的な結果の相違を回避することで、削除された行に関する情報を PRIMARY に格納するという問題は回避されます。これは、セカンダリに正常に適用され、セカンダリからセカンダリにレプリケートされた操作を反映することで実現されます。セカンダリに戻ると、プライマリから発生した操作によって削除されたセカンダリに操作を再適用するために使用できます。

`NDB$EPOCH2()` を使用する場合、セカンダリはプライマリから削除を適用し、反映された操作によってリストアされるまで新しい行を削除することに注意する必要があります。理論上、セカンダリでの後続の挿入または更新はプライマリからの削除と競合しますが、この場合、クラスタ間の相違を防ぐために、これを無視してセカンダリを「win」に許可することを選択します。つまり、削除後、プライマリは競合を検出せず、すぐにセカンダリの次の変更を採用します。このため、セカンダリ状態は最終 (安定) 状態に進むと、複数の前のコミット済状態に再アクセスでき、これらの一部が表示される場合があります。

また、セカンダリからプライマリにすべての操作を反映すると、プライマリログバイナリログのサイズが増加し、帯域幅、CPU 使用率およびディスク I/O が必要になることにも注意してください。

セカンダリでの反映された操作の適用は、セカンダリのターゲット行の状態によって異なります。セカンダリに反映された変更が適用されるかどうかは、`Ndb_conflict_reflected_op_prepare_count` および `Ndb_conflict_reflected_op_discard_count` のステータス変数をチェックすることで追跡できます。適用される変更の数は、これらの値の違いにすぎません (`Ndb_conflict_reflected_op_prepare_count` は常に `Ndb_conflict_reflected_op_discard_count` 以上であることに注意してください)。

イベントは、次の両方の条件に該当する場合にのみ適用されます:

- 行 (イベントのタイプに従って存在するかどうか) の存在。削除および更新操作では、行がすでに存在している必要があります。挿入操作の場合、行は存在していない必要があります。
- 行はプライマリによって最後に変更されました。反映された操作の実行によって変更が行われた可能性があります。

両方の条件が満たされない場合、反映された操作はセカンダリによって破棄されます。

競合解決の例外テーブル。 `NDB$OLD()` の競合解消機能を使用するには、このタイプの競合解消を使用する各 `NDB` テーブルに対応する例外テーブルも作成する必要があります。これは、`NDB$EPOCH()` または `NDB$EPOCH_TRANS()` を使用する場合にも当てはまります。このテーブルの名前は、競合解決が適用されるテーブルの名前に文字列 `$EX` を付加した名前です。(たとえば、元のテーブルの名前が `mytable` である場合、対応する例外テーブルの名前は `mytable$EX` になります。) 例外テーブルを作成する構文は、次のとおりです:

```
CREATE TABLE original_table$EX (  
  [NDB$]server_id INT UNSIGNED,  
  [NDB$]source_server_id INT UNSIGNED,  
  [NDB$]source_epoch BIGINT UNSIGNED,  
  [NDB$]count INT UNSIGNED,  
  
  [NDB$OP_TYPE ENUM('WRITE_ROW','UPDATE_ROW', 'DELETE_ROW',  
    'REFRESH_ROW', 'READ_ROW') NOT NULL,]  
  [NDB$CFT_CAUSE ENUM('ROW_DOES_NOT_EXIST', 'ROW_ALREADY_EXISTS',  
    'DATA_IN_CONFLICT', 'TRANS_IN_CONFLICT') NOT NULL,]  
  [NDB$ORIG_TRANSID BIGINT UNSIGNED NOT NULL,]  
  
  original_table_pk_columns,  
  
  [orig_table_column|orig_table_column$OLD|orig_table_column$NEW,]  
  
  [additional_columns,]  
  
  PRIMARY KEY([NDB$]server_id, [NDB$]source_server_id, [NDB$]source_epoch, [NDB$]count)  
) ENGINE=NDB;
```

最初の 4 つのカラムが必須です。最初の 4 つのカラムの名前と元のテーブルの主キーカラムと一致するカラムは重要ではありません。ただし、わかりやすく一貫性のある理由から、ここに示す名前を `server_id`, `source_server_id`, `source_epoch` および `count` カラムに使用し、元のテーブルの主キーと一致するカラムに元のテーブルと同じ名前を使用することをお勧めします。

例外テーブルで、このセクションの後半で説明するオプションのカラム `NDB$OP_TYPE`、`NDB$CFT_CAUSE` または `NDB$ORIG_TRANSID` を使用する場合は、必要な各カラムに接頭辞 `NDB$` を使用して名前を付ける必要があります。

必要に応じて、オプションカラムを定義しない場合でも `NDB$` プリフィクスを使用して必須カラムに名前を付けることができます。ただしこの場合、4 つすべての必須カラムにプリフィクスを使用して名前を付ける必要があります。

このカラムに続き、元のテーブルの主キーを構成するカラムは、元のテーブルの主キーの定義に使用される順番でコピーをしてください。元のテーブルの主キーカラムを複製するカラムのデータ型は、元のカラムと同じ (または大きい) データ型にしてください。主キーカラムのサブセットを使用できます。

例外テーブルでは、`NDB` ストレージエンジンを使用する必要があります。(例外テーブルで `NDB$OLD()` を使用する例は、このセクションの後半で示します。)

オプションで、コピーされる主キーカラムのあとに追加カラムを定義できますが、その前に追加カラムを定義することはできません。このような追加カラムは `NOT NULL` にはできません。NDB Cluster は、次のいくつかの段落で説明する、3 つの事前定義された追加のオプションカラム `NDB$OP_TYPE`、`NDB$CFT_CAUSE`、および `NDB$ORIG_TRANSID` をサポートしています。

`NDB$OP_TYPE`: このカラムを使用して、競合の原因となっている操作のタイプを取得できます。このカラムを使用する場合、ここで示すように定義します。

```
NDB$OP_TYPE ENUM('WRITE_ROW', 'UPDATE_ROW', 'DELETE_ROW',  
'REFRESH_ROW', 'READ_ROW') NOT NULL
```

`WRITE_ROW`、`UPDATE_ROW`、および `DELETE_ROW` 操作タイプは、ユーザー起動の操作を表します。`REFRESH_ROW` 操作は、競合を検出したクラスタから元のクラスタに戻されたトランザクションを相殺するときに、競合解決によって生成される操作です。`READ_ROW` 操作は、排他的行ロックを使用して定義される、ユーザー起動の読み取り追跡操作です。

`NDB$CFT_CAUSE`: 登録された競合の原因を示すオプションのカラム `NDB$CFT_CAUSE` を定義できます。このカラムは、使用する場合、ここで示すように定義されます。

```
NDB$CFT_CAUSE ENUM('ROW_DOES_NOT_EXIST', 'ROW_ALREADY_EXISTS',  
'DATA_IN_CONFLICT', 'TRANS_IN_CONFLICT') NOT NULL
```

`ROW_DOES_NOT_EXIST` は `UPDATE_ROW` および `WRITE_ROW` 操作の原因として報告できます。`ROW_ALREADY_EXISTS` は `WRITE_ROW` イベントに対して報告できます。`DATA_IN_CONFLICT` は、行ベースの競合関数が競合を検出した場合に報告されます。`TRANS_IN_CONFLICT` は、トランザクション競合関数がトランザクション全体に属するすべての操作を拒否する場合に報告されます。

`NDB$ORIG_TRANSID`: `NDB$ORIG_TRANSID` カラム (使用する場合) には、元のトランザクションの ID が含まれます。このカラムは次のように定義されます。

```
NDB$ORIG_TRANSID BIGINT UNSIGNED NOT NULL
```

`NDB$ORIG_TRANSID` は `NDB` によって生成される 64 ビットの値です。この値は、同じまたは異なる例外テーブルから同じ競合トランザクションに属する複数の例外テーブルのエントリを相互に関連付けるために使用されます。

元のテーブルの主キーの一部ではない追加の参照カラムには、`colname$OLD` または `colname$NEW` という名前を付けることができます。`colname$OLD` は、更新および削除操作 (`DELETE_ROW` イベントを含む操作) で古い値を参照します。`colname$NEW` を使用すると、挿入および更新操作、つまり `WRITE_ROW` イベント、`UPDATE_ROW` イベントまたは両方のタイプのイベントを使用した操作で新しい値を参照できます。競合する操作で主キーではない特定の参照カラムの値が指定されていない場合、例外テーブルの行には、`NULL` またはそのカラムに定義されているデフォルト値が含まれます。

重要

`mysql.ndb_replication` テーブルは、データテーブルがレプリケーション用にセットアップされたときに読み取られるため、複製されるテーブルに対応する行は、複製されるテーブルが作成される前に `mysql.ndb_replication` に挿入する必要があります。

例

次の例では、[セクション23.6.5「NDB Cluster のレプリケーションの準備」](#) および [セクション23.6.6「NDB Cluster レプリケーションの開始 \(シングルレプリケーションチャネル\)」](#) で説明されているように、NDB Cluster レプリケーションがすでに機能していることを前提としています。

NDB\$MAX() の例。 「タイムスタンプ」としてカラム `mycol` を使用して、テーブル `test.t1` で「もっとも大きいタイムスタンプが優先」競合解決を有効にするものとします。これは、次のステップで実行できます。

1. ソース `mysqlid` と `--ndb-log-update-as-write=OFF` が起動されていることを確認します。
2. ソースで、次の `INSERT` ステートメントを実行します:

```
INSERT INTO mysql.ndb_replication
VALUES ('test', 't1', 0, NULL, 'NDB$MAX(mycol));
```

`server_id` に 0 を挿入すると、このテーブルにアクセスするすべての SQL ノードが競合解決を使用します。特定の `mysqlid` だけで競合解決を使用する場合は、実際のサーバー ID を使用します。

`binlog_type` カラムに `NULL` を挿入すると、0 (`NBT_DEFAULT`) の挿入と同じ効果があり、サーバーのデフォルトが使用されます。

3. `test.t1` テーブルを作成します。

```
CREATE TABLE test.t1 (
  columns
  mycol INT UNSIGNED,
  columns
) ENGINE=NDB;
```

これで、このテーブルで更新が実行されると、競合解決が適用され、`mycol` の最大値を持つ行のバージョンがレプリカに書き込まれます。

注記

コマンドラインオプションを使用するのではなく、`ndb_replication` テーブルを使用してソースのロギングを制御するには、`NBT_UPDATED_ONLY_USE_UPDATE` としてのその他の `binlog_type` オプションを使用する必要があります。

NDB\$OLD() の例。 `NDB` テーブル (ここで定義されたテーブルなど) が複製中であり、このテーブルへの更新に「同じタイムスタンプが優先」競合解決を有効にするものとします。

```
CREATE TABLE test.t2 (
  a INT UNSIGNED NOT NULL,
  b CHAR(25) NOT NULL,
  columns,
  mycol INT UNSIGNED NOT NULL,
  columns,
  PRIMARY KEY pk (a, b)
) ENGINE=NDB;
```

ここで示す順番で、次のステップが必要です。

1. まず (`test.t2` を作成する前に)、ここで示すように `mysql.ndb_replication` テーブルに行を挿入する必要があります。

```
INSERT INTO mysql.ndb_replication
VALUES ('test', 't2', 0, NULL, 'NDB$OLD(mycol));
```

`binlog_type` カラムに可能な値は、このセクションの前半に示しています。値 `'NDB$OLD(mycol)'` を `conflict_fn` カラムに挿入してください。

2. `test.t2` に対応する例外テーブルを作成します。ここに示すテーブル作成ステートメントには、必要なすべてのカラムが含まれます。これらのカラムの後、およびテーブル主キーの定義の前に、追加のカラムを宣言する必要があります。

```
CREATE TABLE test.t2$EX (
  server_id INT UNSIGNED,
  source_server_id INT UNSIGNED,
  source_epoch BIGINT UNSIGNED,
  count INT UNSIGNED,
  a INT UNSIGNED NOT NULL,
  b CHAR(25) NOT NULL,
```

```
[additional_columns,]
PRIMARY KEY(server_id, source_server_id, source_epoch, count)
) ENGINE=NDB;
```

特定の競合のタイプ、原因および元のトランザクション ID に関する情報を示す追加の列を含めることができます。元のテーブルのすべての主キー列に一致する列を提供する必要はありません。つまり、次のような例外テーブルを作成できます:

```
CREATE TABLE test.t2$EX (
  NDB$server_id INT UNSIGNED,
  NDB$source_server_id INT UNSIGNED,
  NDB$source_epoch BIGINT UNSIGNED,
  NDB$count INT UNSIGNED,
  a INT UNSIGNED NOT NULL,

  NDB$OP_TYPE ENUM('WRITE_ROW','UPDATE_ROW','DELETE_ROW',
  'REFRESH_ROW','READ_ROW') NOT NULL,
  NDB$CFT_CAUSE ENUM('ROW_DOES_NOT_EXIST','ROW_ALREADY_EXISTS',
  'DATA_IN_CONFLICT','TRANS_IN_CONFLICT') NOT NULL,
  NDB$ORIG_TRANSID BIGINT UNSIGNED NOT NULL,

  [additional_columns,]

  PRIMARY KEY(NDB$server_id, NDB$source_server_id, NDB$source_epoch, NDB$count)
) ENGINE=NDB;
```

注記

テーブル定義に `NDB$OP_TYPE`、`NDB$CFT_CAUSE` または `NDB$ORIG_TRANSID` のいずれかの列が含まれているため、4 つの必須列には `NDB$` 接頭辞が必要です。

3. 前に示したように、テーブル `test.t2` を作成します。

`NDB$OLD()` を使用して競合解決を実行するテーブルごとに、これらのステップに従う必要があります。このようなテーブルごとに、[mysql.ndb_replication](#) に対応する行があり、複製されているテーブルと同じデータベースに例外テーブルがある必要があります。

読み取り競合の検出と解決。NDB Cluster は読み取り操作の追跡もサポートしているため、循環レプリケーション設定で、あるクラスタ内の特定の行の読み取りと別のクラスタ内の同じ行の更新または削除との間の競合を管理できません。この例では、レプリカクラスタ (B) がインターリーブされたトランザクションで従業員の前部門の従業員数を更新している間に、`employee` および `department` テーブルを使用して、従業員がソースクラスタ上のある部門から別の部門に異動されるシナリオをモデル化します (これ以降、クラスタ A と呼びます)。

データテーブルは次の SQL ステートメントで作成されました。

```
# Employee table
CREATE TABLE employee (
  id INT PRIMARY KEY,
  name VARCHAR(2000),
  dept INT NOT NULL
) ENGINE=NDB;

# Department table
CREATE TABLE department (
  id INT PRIMARY KEY,
  name VARCHAR(2000),
  members INT
) ENGINE=NDB;
```

2 つのテーブルの内容は、次の `SELECT` ステートメントの出力 (一部) に示される行を含みます。

```
mysql> SELECT id, name, dept FROM employee;
+-----+-----+
| id | name | dept |
+-----+-----+
...
| 998 | Mike | 3 |
| 999 | Joe | 3 |
```

```
| 1000 | Mary | 3 |
...
+-----+-----+-----+
mysql> SELECT id, name, members FROM department;
+-----+-----+-----+
| id | name      | members |
+-----+-----+-----+
...
| 3  | Old project | 24      |
...
+-----+-----+-----+
```

4 つの必須カラム (これらはこのテーブルの主キーに使用されます)、操作タイプと原因用のオプションカラム、および元のテーブルの主キーカラムを含んだ、ここで示した SQL ステートメントで作成された例外テーブルをすでに使用しているものとします。

```
CREATE TABLE employee$EX (
  NDB$server_id INT UNSIGNED,
  NDB$source_server_id INT UNSIGNED,
  NDB$source_epoch BIGINT UNSIGNED,
  NDB$count INT UNSIGNED,

  NDB$OP_TYPE ENUM( 'WRITE_ROW','UPDATE_ROW', 'DELETE_ROW',
    'REFRESH_ROW','READ_ROW') NOT NULL,
  NDB$CFT_CAUSE ENUM( 'ROW_DOES_NOT_EXIST',
    'ROW_ALREADY_EXISTS',
    'DATA_IN_CONFLICT',
    'TRANS_IN_CONFLICT') NOT NULL,

  id INT NOT NULL,

  PRIMARY KEY(NDB$server_id, NDB$source_server_id, NDB$source_epoch, NDB$count)
) ENGINE=NDB;
```

2 つのクラスタ上で同時に 2 つのトランザクションが発生するものとします。クラスタ A では、新しい部門を作成してから、従業員番号 999 をその部門に移動します。次の SQL ステートメントを使用します。

```
BEGIN;
INSERT INTO department VALUES (4, "New project", 1);
UPDATE employee SET dept = 4 WHERE id = 999;
COMMIT;
```

同時にクラスタ B では、次に示すように、別のトランザクションが `employee` から読み取ります。

```
BEGIN;
SELECT name FROM employee WHERE id = 999;
UPDATE department SET members = members - 1 WHERE id = 3;
commit;
```

競合しているトランザクションは、通常は競合解決メカニズムで検出されません。これは、競合が読み取り (`SELECT`) と更新操作の間で発生しているためです。この問題を回避するには、レプリカクラスタで `SET ndb_log_exclusive_reads = 1` を実行します。この方法で排他読み取りロックを取得すると、ソースで読み取られた行にレプリカクラスタで競合解決が必要であるというフラグが付けられます。これらのトランザクションのロギングの前にこの方法で排他読み取りを有効にすると、クラスタ B での読み取りが追跡され、解決のためにクラスタ A に送信されます。その後、従業員行での競合が検出され、クラスタ B でのトランザクションが中断されます。

競合は、ここで示すように (クラスタ A にある) 例外テーブルに `READ_ROW` 操作として登録されます (操作タイプの説明については、[競合解決の例外テーブル](#)を参照してください)。

```
mysql> SELECT id, NDB$OP_TYPE, NDB$CFT_CAUSE FROM employee$EX;
+-----+-----+-----+
| id | NDB$OP_TYPE | NDB$CFT_CAUSE |
+-----+-----+-----+
...
| 999 | READ_ROW   | TRANS_IN_CONFLICT |
+-----+-----+-----+
```

読み取り操作で検出された、存在するどの行にもフラグが付けられます。すなわち、ここで示すように、クラスタ A での更新と、同時に発生したトランザクションで同じテーブルからのクラスタ B での複数行の読み取りとの間で競合

の影響を調べることで、例外テーブルに同じ競合に起因する複数の行のログが取られる場合があります。ここで、クラスタ A で実行されるトランザクションを示します。

```
BEGIN;
INSERT INTO department VALUES (4, "New project", 0);
UPDATE employee SET dept = 4 WHERE dept = 3;
SELECT COUNT(*) INTO @count FROM employee WHERE dept = 4;
UPDATE department SET members = @count WHERE id = 4;
COMMIT;
```

同時に、ここで示すステートメントを含むトランザクションがクラスタ B で実行されます。

```
SET ndb_log_exclusive_reads = 1; # Must be set if not already enabled
...
BEGIN;
SELECT COUNT(*) INTO @count FROM employee WHERE dept = 3 FOR UPDATE;
UPDATE department SET members = @count WHERE id = 3;
COMMIT;
```

この場合、ここで示すように、2 番目のトランザクションの `SELECT` の中の `WHERE` 条件に一致する 3 つすべての行が読み取られ、例外テーブルでフラグが付けられます。

```
mysql> SELECT id, NDB$OP_TYPE, NDB$CFT_CAUSE FROM employee$EX;
+-----+-----+-----+
| id | NDB$OP_TYPE | NDB$CFT_CAUSE |
+-----+-----+-----+
...
| 998 | READ_ROW | TRANS_IN_CONFLICT |
| 999 | READ_ROW | TRANS_IN_CONFLICT |
| 1000 | READ_ROW | TRANS_IN_CONFLICT |
...
+-----+-----+-----+
```

読み取りの追跡は、存在する行だけに基づいて行われます。特定条件の追跡に基づく読み取りは、検出された行だけと競合し、インターリーブされたトランザクションで挿入された行とは競合しません。これは、NDB Cluster の単一インスタンスで排他的行ロックが実行される方法に似ています。

23.7 NDB Cluster リリースノート

NDB Cluster リリースでの変更は、このリファレンスマニュアルとは別にドキュメント化されています。NDB Cluster 8.0 の各リリースの変更に関するリリースノートは、「[NDB 8.0 リリースノート](#)」にあります。

NDB Cluster の古いバージョンのリリースノートは、「[NDB Cluster リリースノート](#)」から入手できます。

第 24 章 パーティション化

目次

24.1 MySQL のパーティショニングの概要	4030
24.2 パーティショニングタイプ	4032
24.2.1 RANGE パーティショニング	4034
24.2.2 LIST パーティショニング	4038
24.2.3 COLUMNS パーティショニング	4040
24.2.4 HASH パーティショニング	4047
24.2.5 KEY パーティショニング	4049
24.2.6 サブパーティショニング	4051
24.2.7 MySQL パーティショニングによる NULL の扱い	4053
24.3 パーティション管理	4057
24.3.1 RANGE および LIST パーティションの管理	4057
24.3.2 HASH および KEY パーティションの管理	4063
24.3.3 パーティションとサブパーティションをテーブルと交換する	4064
24.3.4 パーティションの保守	4071
24.3.5 パーティションに関する情報を取得する	4072
24.4 パーティションプルーニング	4074
24.5 パーティション選択	4077
24.6 パーティショニングの制約と制限	4082
24.6.1 パーティショニングキー、主キー、および一意キー	4088
24.6.2 ストレージエンジンに関連するパーティショニング制限	4091
24.6.3 関数に関連するパーティショニング制限	4092

この章では、ユーザー定義のパーティション化について説明します。

注記

テーブルのパーティション化は、ウィンドウ関数で使用されるパーティション化とは異なります。ウィンドウ関数の詳細は、[セクション12.21「ウィンドウ関数」](#)を参照してください。

MySQL 8.0 では、パーティション分割のサポートは [InnoDB](#) および [NDB](#) ストレージエンジンによって提供されません。

MySQL 8.0 は現在、[MyISAM](#) などの [InnoDB](#) または [NDB](#) 以外のストレージエンジンを使用したテーブルのパーティション化をサポートしていません。ネイティブパーティション化サポートを提供しないストレージエンジンを使用してパーティションテーブルを作成しようとすると、[ER_CHECK_NOT_IMPLEMENTED](#) で失敗します。

Oracle によって提供される MySQL 8.0 Community バイナリには、[InnoDB](#) および [NDB](#) ストレージエンジンによって提供されるパーティショニングサポートが含まれます。MySQL Enterprise Edition バイナリで提供されるパーティション分割サポートの詳細は、[第30章「MySQL Enterprise Edition」](#)を参照してください。

ソースから MySQL 8.0 をコンパイルする場合は、[InnoDB](#) サポートを使用してビルドを構成すれば、[InnoDB](#) テーブルのパーティションサポートを使用してバイナリを作成できます。詳細は、[セクション2.9「ソースから MySQL をインストールする」](#)を参照してください。

[InnoDB](#) によるパーティション化サポートを有効にするために、これ以上行う必要はありません (たとえば、[my.cnf](#) ファイルに特別なエントリは必要ありません)。

[InnoDB](#) ストレージエンジンによるパーティション分割サポートを無効にすることはできません。

パーティショニングの概要およびパーティショニングの概念については、[セクション24.1「MySQL のパーティショニングの概要」](#)を参照してください。

サブパーティション化に加えて、いくつかのタイプのパーティション化がサポートされています。[セクション24.2「パーティショニングタイプ」](#) および [セクション24.2.6「サブパーティショニング」](#)を参照してください。

[セクション24.3「パーティション管理」](#)では、既存のパーティション化されたテーブルでパーティションを追加、削除および変更する方法について説明しています。

[セクション24.3.4「パーティションの保守」](#)では、パーティション化されたテーブルで使用するテーブル保守コマンドについて説明しています。

`INFORMATION_SCHEMA` データベースの `PARTITIONS` テーブルには、パーティションおよびパーティション化されたテーブルに関する情報があります。詳細は、[セクション26.21「INFORMATION_SCHEMA PARTITIONS テーブル」](#)を参照してください。このテーブルに対するクエリーのいくつかの例については、[セクション24.2.7「MySQL パーティショニングによる NULL の扱い」](#)を参照してください。

MySQL 8.0 のパーティショニングの既知の問題については、[セクション24.6「パーティショニングの制約と制限」](#)を参照してください。

また、パーティション化されたテーブルを使用して作業を行うときに、次のリソースが役に立つことがあります。

追加のリソース。MySQL のユーザー定義パーティショニングに関するその他の情報ソースには、次のものがあります。

- [MySQL パーティショニングフォーラム](#)

これは、MySQL パーティショニングテクノロジーに興味があり実験している人のための公式ディスカッションフォーラムです。MySQL 開発者などからの発表や更新が掲載されています。パーティショニングの開発チームおよびドキュメントチームのメンバーによってモニターされています。

- [Mikael Ronström のブログ](#)

MySQL Partitioning Architect および Lead Developer Mikael Ronström は、MySQL Partitioning および NDB Cluster での作業に関する記事を頻繁に投稿しています。

- [PlanetMySQL](#)

MySQL を使用している人が関心を持つと思われる MySQL 関連ブログをまとめた MySQL ニュースサイト。MySQL パーティショニングを使用している人が更新しているブログへのリンクをここでチェックしたり、自分のブログを追加したりすることをお勧めします。

24.1 MySQL のパーティショニングの概要

このセクションでは、MySQL 8.0 のパーティショニングの概念について説明します。

パーティショニングの制約および機能制限については、[セクション24.6「パーティショニングの制約と制限」](#)を参照してください。

SQL 標準では、データ保存の物理的な仕様に関するガイダンスはあまり提供されていません。SQL 言語自体が、それが動作するスキーマ、テーブル、行、またはカラムの基盤となるデータ構造やメディアと独立して動作するように意図されています。それにもかかわらず、ほとんどの高度なデータベース管理システムでは、ファイルシステム、ハードウェア、またはその両方について、特定のデータを格納するために使用される物理的な場所を判別する方法が開発されてきました。MySQL では、`InnoDB` ストレージエンジンが長い間テーブルスペースの概念をサポートしており([セクション15.6.3「テーブルスペース」](#)を参照)、パーティション分割を導入する前でも、異なるデータベースを格納するために異なる物理ディレクトリを使用するように MySQL Server を構成できます(この方法については [セクション8.12.2「シンボリックリンクの使用」](#)を参照)。

パーティショニングはこの認識をさらに一歩進めて、必要に応じて多くの部分を設定できるルールに従って、個々のテーブルの部分をファイルシステムに配分できるようにしています。それにより、テーブルの異なる部分が別個のテーブルとして別個の場所に格納されます。データを分割するためにユーザーが選択するルールはパーティショニング関数と呼ばれ、MySQL では法、範囲セットまたは値リストに対する単純な照合、内部ハッシュ関数、または線形ハッシュ関数が使用されます。関数は、ユーザーが指定したパーティショニングタイプに従って選択され、ユーザーが指定した式の値をパラメータとして取ります。この式には、使用されるパーティショニングのタイプに応じて、カラム値、1つ以上のカラム値を操作する関数、または1つ以上のカラム値のセットを指定できます。

`RANGE`、`LIST`、および `[LINEAR] HASH` パーティショニングの場合、パーティショニングカラムの値はパーティショニング関数に渡され、特定のレコードを格納すべきパーティションの番号を表す整数値が返されます。この関数は非定数および非ランダムである必要があります。クエリーを含めることはできませんが、式が `NULL` または次のような整数 `intval` を返すかぎり、MySQL で有効な SQL 式を使用できます。


```
-MAXVALUE <= intval <= MAXVALUE
```

(MAXVALUE は対象となる整数型の上限を表すために使用されます。-MAXVALUE は下限を表します。)

[LINEAR] KEY、RANGE COLUMNS、および LIST COLUMNS パーティショニングの場合、パーティショニング式は 1 つ以上のカラムのリストから構成されます。

[LINEAR] KEY パーティショニングの場合、パーティショニング関数は MySQL によって提供されます。

許可されるパーティショニングカラムタイプおよびパーティショニング関数については、[セクション24.2「パーティショニングタイプ」](#)、およびパーティショニング構文の説明および追加例を示している[セクション13.1.20「CREATE TABLE ステートメント」](#)を参照してください。パーティショニング関数の制約については、[セクション24.6.3「関数に関連するパーティショニング制限」](#)を参照してください。

これは「水平パーティショニング」と呼ばれます。つまり、テーブル内の異なる行を異なる物理パーティションに割り当てることができます。MySQL 8.0 では、テーブルの異なるカラムが異なる物理パーティションに割り当てられる垂直パーティション化はサポートされていません。現時点では、MySQL に垂直パーティション化を導入する計画はありません。

パーティション化されたテーブルを作成するには、それらをサポートするストレージエンジンを使用する必要があります。MySQL 8.0 では、同じパーティション化されたテーブルのすべてのパーティションが同じストレージエンジンを使用する必要があります。ただし、同じ MySQL サーバーまたは同じデータベース上の異なるパーティション化されたテーブルに、異なるストレージエンジンを使用することはできません。

MySQL 8.0 では、パーティション分割をサポートするストレージエンジンは InnoDB と NDB だけです。パーティション分割は、それをサポートしていないストレージエンジンでは使用できません。これには、MyISAM、MERGE、CSV および FEDERATED ストレージエンジンが含まれます。

KEY または LINEAR KEY によるパーティショニングは NDB で使用できますが、ほかのタイプのユーザー定義パーティショニングはこのストレージエンジンを使用するテーブルでサポートされません。また、ユーザー定義パーティショニングを使用する NDB テーブルには明示的な主キーが必要であり、テーブルのパーティショニング式で参照されるカラムは主キーの一部である必要があります。ただし、ユーザーパーティション化された NDB テーブルを作成または変更するために使用される CREATE TABLE または ALTER TABLE ステートメントの PARTITION BY KEY 句または PARTITION BY LINEAR KEY 句にカラムが 1 つもリストされていない場合は、テーブルに明示的な主キーは必要ありません。詳細は、[セクション23.1.7.1「NDB Cluster の SQL 構文に準拠していません」](#)を参照してください。

パーティション化されたテーブルを作成する場合、デフォルトのストレージエンジンはほかのテーブルを作成する場合と同様に使用されます。この動作をオーバーライドするには、パーティション化されていないテーブルの場合と同様に [STORAGE] ENGINE オプションを使用するだけで済みます。ターゲットストレージエンジンは、ネイティブパーティショニングサポートを提供する必要があります。そうしないと、ステートメントは失敗します。CREATE TABLE ステートメントでパーティション化オプションを使用する前に、[STORAGE] ENGINE (およびその他のテーブルオプション) をリストする必要があることに注意してください。次の例では、ハッシュによって 6 つのパーティションにパーティション化され、(default_storage_engine の値に関係なく) InnoDB ストレージエンジンを使用するテーブルを作成する方法を示します:

```
CREATE TABLE ti (id INT, amount DECIMAL(7,2), tr_date DATE)
ENGINE=INNODB
PARTITION BY HASH( MONTH(tr_date) )
PARTITIONS 6;
```

各 PARTITION 句に [STORAGE] ENGINE オプションを含めることはできますが、MySQL 8.0 ではこれは効果がありません。

特に指定がないかぎり、この説明の残りの例では、default_storage_engine が InnoDB であると想定しています。

重要

パーティショニングはテーブルのすべてのデータおよびインデックスに適用されます。データだけにパーティション化しインデックスは行わないことはできず (その逆も不可)、テーブルの一部のみをパーティション化することもできません。

各パーティションのデータおよびインデックスを特定のディレクトリに割り当てするには、パーティション化されたテーブルを作成するために使用する CREATE TABLE ステートメントの PARTITION 句に DATA DIRECTORY および INDEX DIRECTORY オプションを使用します。

InnoDB テーブルの個々のパーティションおよびサブパーティションでは、[DATA DIRECTORY](#) オプションのみがサポートされます。MySQL 8.0.21 では、[DATA DIRECTORY](#) 句で指定されたディレクトリは InnoDB で認識されている必要があります。詳細は、[DATA DIRECTORY 句の使用](#)を参照してください。

テーブルのパーティション化式で使用されるすべてのカラムは、主キーを含め、テーブルに含まれる可能性のあるすべての一意キーの一部である必要があります。つまり、次の SQL ステートメントで作成されたこのようなテーブルはパーティション化できません:

```
CREATE TABLE tnp (  
  id INT NOT NULL AUTO_INCREMENT,  
  ref BIGINT NOT NULL,  
  name VARCHAR(255),  
  PRIMARY KEY pk (id),  
  UNIQUE KEY uk (name)  
);
```

キー `pk` と `uk` には共通のカラムがないため、パーティション化式で使用できるカラムはありません。この状況で考えられる回避策には、`name` カラムのテーブル主キーへの追加、`id` カラムの `uk` への追加、または一意キーの完全な削除が含まれます。詳しくは[セクション24.6.1「パーティショニングキー、主キー、および一意キー」](#)をご覧ください。

また、`MAX_ROWS` および `MIN_ROWS` は、各パーティションに格納できる行のそれぞれ最大数および最小数を決定するために使用できます。これらのオプションの詳細は、[セクション24.3「パーティション管理」](#)を参照してください。

`MAX_ROWS` オプションは、余分なパーティションを含む「NDB Cluster」テーブルを作成する場合にも役立つため、ハッシュインデックスの記憶域を増やすことができます。詳細は、[DataMemory データノード構成パラメータのドキュメント](#)、および [セクション23.1.2「NDB Cluster ノード、ノードグループ、フラグメントレプリカ、およびパーティション」](#)を参照してください。

パーティショニングのいくつかの利点を次に示します。

- パーティショニングを使用すると、単一ディスクまたはファイルシステムパーティションに保持できるデータより多くのデータを1つのテーブルに格納できます。
- 有効性を失っているデータは、多くの場合、そのデータのみが含まれているパーティションを削除することによって、パーティション化されたテーブルから簡単に削除できます。反対に、新しいデータを追加する処理は、そのデータだけを格納するための1つ以上の新しいパーティションを追加することによって、非常に便利になることがあります。
- 指定された `WHERE` 句を満たすデータを1つ以上のパーティションのみに格納できることによって、検索からほかのパーティションが自動的に除外され、一部のクエリーが大幅に最適化されることがあります。パーティションはパーティション化されたテーブルが作成されたあとに変更できるため、使用頻度の高いクエリー (パーティショニングスキームが最初に設定されたときはあまり使用されていなかった) を改善するためにデータを再編成できます。一致しないパーティション (およびそれらに含まれている行) を除外するこの機能は、よくパーティションプルニングと呼ばれます。詳細は、[セクション24.4「パーティションプルニング」](#)を参照してください。

また、MySQL では、クエリーの明示的なパーティション選択もサポートされています。たとえば、`SELECT * FROM t PARTITION (p0,p1) WHERE c < 5` は、パーティション `p0` および `p1` の `WHERE` 条件に一致する行のみを選択します。この場合、MySQL はテーブル `t` のほかのパーティションをチェックしません。これにより、検査するパーティションが事前にわかっているときにクエリー速度が大幅に向上することがあります。パーティション選択は、データ変更ステートメント (`DELETE`、`INSERT`、`REPLACE`、`UPDATE`、`LOAD DATA`、および `LOAD XML`) でもサポートされます。詳細および例については、これらのステートメントの説明を参照してください。

24.2 パーティショニングタイプ

このセクションでは、MySQL 8.0 で使用できるパーティショニングのタイプについて説明します。これらには、次に一覧したタイプが含まれます。

- RANGE パーティショニング**。このタイプのパーティショニングは、指定された範囲に含まれるカラム値に基づいて、行をパーティションに割り当てます。[セクション24.2.1「RANGE パーティショニング」](#)を参照してください。このタイプを拡張した `RANGE COLUMNS` については、[セクション24.2.3.1「RANGE COLUMNS パーティショニング」](#)を参照してください。

- LIST パーティショニング. [RANGE](#) によるパーティショニングに似ていますが、別個の値のセットのいずれかに一致するカラムに基づいて、パーティションが選択されます。 [セクション24.2.2「LIST パーティショニング」](#)を参照してください。このタイプを拡張した [LIST COLUMNS](#) については、 [セクション24.2.3.2「LIST COLUMNS パーティショニング」](#)を参照してください。
- HASH パーティショニング. このタイプのパーティショニングでは、テーブルに挿入される行内のカラム値を操作するユーザー定義式によって返される値に基づいて、パーティションが選択されます。関数は、負ではない整数値を返す MySQL の有効な式で構成できます。このタイプを拡張した [LINEAR HASH](#) も使用できます。 [セクション24.2.4「HASH パーティショニング」](#)を参照してください。
- KEY パーティショニング. このタイプのパーティショニングは、[HASH](#) によるパーティショニングに似ていますが、評価される 1 つ以上のカラムのみを指定し、MySQL サーバーが独自のハッシュ関数を提供します。MySQL によって提供されるハッシュ関数ではカラムデータ型に関係なく整数結果が保証されるため、これらのカラムに整数以外の値が含まれていてもかまいません。このタイプを拡張した [LINEAR KEY](#) も使用できます。 [セクション24.2.5「KEY パーティショニング」](#)を参照してください。

データベースパーティショニングの非常に一般的な使用法は、日付によってデータを分けることです。一部のデータベースシステムは、MySQL 8.0 では実装されていない、明示的な日付パーティショニングをサポートしています。ただし、MySQL で、[DATE](#)、[TIME](#)、または [DATETIME](#) カラムに基づいて、またはそのようなカラムを使用する式に基づいて、パーティショニングスキームを作成することは難しくありません。

[KEY](#) または [LINEAR KEY](#) でパーティショニングする場合は、[DATE](#)、[TIME](#)、または [DATETIME](#) カラムを、カラム値の変更を実行しないパーティショニングカラムとして使用できます。たとえば、次のテーブル作成ステートメントは MySQL で完全に有効です。

```
CREATE TABLE members (  
  firstname VARCHAR(25) NOT NULL,  
  lastname VARCHAR(25) NOT NULL,  
  username VARCHAR(16) NOT NULL,  
  email VARCHAR(35),  
  joined DATE NOT NULL  
)  
PARTITION BY KEY(joined)  
PARTITIONS 6;
```

MySQL 8.0 では、[RANGE COLUMNS](#) および [LIST COLUMNS](#) パーティショニングを使用して、[DATE](#) または [DATETIME](#) カラムをパーティショニングカラムとして使用することもできます。

他のパーティション化タイプでは、整数値または [NULL](#) を生成するパーティション化式が必要です。[RANGE](#)、[LIST](#)、[HASH](#)、または [LINEAR HASH](#) による日付ベースパーティショニングを使用する場合は、次のように単純に [DATE](#)、[TIME](#)、または [DATETIME](#) カラムを操作してそのような値を返す関数を使用できます。

```
CREATE TABLE members (  
  firstname VARCHAR(25) NOT NULL,  
  lastname VARCHAR(25) NOT NULL,  
  username VARCHAR(16) NOT NULL,  
  email VARCHAR(35),  
  joined DATE NOT NULL  
)  
PARTITION BY RANGE( YEAR(joined) ) (  
  PARTITION p0 VALUES LESS THAN (1960),  
  PARTITION p1 VALUES LESS THAN (1970),  
  PARTITION p2 VALUES LESS THAN (1980),  
  PARTITION p3 VALUES LESS THAN (1990),  
  PARTITION p4 VALUES LESS THAN MAXVALUE  
);
```

日付を使用したパーティショニングの追加例は、この章の次のセクションにあります。

- [セクション24.2.1「RANGE パーティショニング」](#)
- [セクション24.2.4「HASH パーティショニング」](#)
- [セクション24.2.4.1「LINEAR HASH パーティショニング」](#)

日付ベースパーティショニングのより複雑な例については、次のセクションを参照してください。

- [セクション24.4「パーティションプルーニング」](#)

• セクション24.2.6「サブパーティショニング」

MySQL パーティショニングは、`TO_DAYS()`、`YEAR()`、および `TO_SECONDS()` 関数で使用するために最適化されています。ただし、整数または `NULL` を返すほかの日時関数 (`WEEKDAY()`、`DAYOFYEAR()`、`MONTH()` など) を使用できません。そのような関数の詳細については、[セクション12.7「日付および時間関数」](#)を参照してください。

使用するパーティショニングのタイプにかかわらず、パーティションには作成時に常に 0 から始まる番号が順番に自動的に付けられることを覚えておくことが重要です。新しい行がパーティション化されたテーブルに挿入される場合は、これらのパーティション番号が正しいパーティションを識別するために使用されます。たとえば、テーブルで 4 つのパーティションが使用される場合、これらのパーティションには 0、1、2、および 3 という番号が付けられます。RANGE および LIST パーティショニングタイプの場合は、各パーティション番号のパーティションが定義されている必要があります。HASH パーティション化の場合、ユーザー指定の式は 0 より大きい整数値に評価される必要があります。KEY パーティショニングの場合は、MySQL サーバーが内部で使用しているハッシュ関数によって、この問題が自動的に対処されます。

パーティションの名前は通常、ほかの MySQL 識別子 (テーブル名、データベース名など) を制御するルールに従っています。ただし、パーティション名では大/小文字が区別されないことに注意してください。たとえば、次の `CREATE TABLE` ステートメントは、示されているように失敗します。

```
mysql> CREATE TABLE t2 (val INT)
-> PARTITION BY LIST(val)(
-> PARTITION mypart VALUES IN (1,3,5),
-> PARTITION MyPart VALUES IN (2,4,6)
-> );
ERROR 1488 (HY000): Duplicate partition name mypart
```

失敗は、MySQL がパーティション名 `mypart` と `MyPart` の違いを認識できないために発生します。

テーブルのパーティション番号を指定するときは、先行ゼロなしのゼロ以外の正の整数リテラルとして表現する必要があり、`0.8E+01` や `6-2` などの式であってははいけません (これが整数値に評価されるとしても)。小数は許可されません。

次の各セクションでは、各パーティションタイプの作成に使用できる構文に使用可能なすべての形式を必ずしも指定するわけではありません。この詳細は、[セクション13.1.20「CREATE TABLE ステートメント」](#)を参照してください。

24.2.1 RANGE パーティショニング

範囲によってパーティション化されるテーブルは、各パーティションに含まれる行のパーティショニング式値が指定された範囲に収まるようにパーティション化されます。範囲は、連続しているけれども重複しないものであるべきで、`VALUES LESS THAN` 演算子を使用して定義されます。次のいくつかの例では、20 のビデオ店で構成されるチェーン (1 から 20 までの番号が付けられている) の従業員レコードを保持する、次のようなテーブルを作成していると想定してください。

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT NOT NULL,
  store_id INT NOT NULL
);
```

注記

ここで使用する `employees` テーブルには主キーまたは一意キーがありません。これらの例はここでの説明のためのもので、実際のテーブルは主キー、一意キー、またはその両方を備えている可能性がきわめて高く、パーティショニングカラムに利用できる選択肢はこれらのキー (ある場合) に使用されるカラムに依存します。これらの事項については、[セクション24.6.1「パーティショニングキー、主キー、および一意キー」](#)を参照してください。

このテーブルは、必要に応じていくつかの方法で、範囲によるパーティション化を実行できます。1 つの方法は、`store_id` カラムを使用することです。たとえば、次のように `PARTITION BY RANGE` 句を追加することで、テーブルを 4 つのパーティションに分割することを決定できます。

```
CREATE TABLE employees (  
  id INT NOT NULL,  
  fname VARCHAR(30),  
  lname VARCHAR(30),  
  hired DATE NOT NULL DEFAULT '1970-01-01',  
  separated DATE NOT NULL DEFAULT '9999-12-31',  
  job_code INT NOT NULL,  
  store_id INT NOT NULL  
)  
PARTITION BY RANGE (store_id) (  
  PARTITION p0 VALUES LESS THAN (6),  
  PARTITION p1 VALUES LESS THAN (11),  
  PARTITION p2 VALUES LESS THAN (16),  
  PARTITION p3 VALUES LESS THAN (21)  
);
```

このパーティショニングスキームでは、店舗 1 から店舗 5 で働いている従業員に対応するすべての行がパーティション `p0` に格納され、店舗 6 から店舗 10 の従業員がパーティション `p1` に格納されます (以下同様)。各パーティションは、最低から最高の順に定義されます。これは `PARTITION BY RANGE` 構文の要件です。これは、C または Java の一連の `if ... elseif ...` ステートメントに似ていると考えることができます。

(72, 'Mitchell', 'Wilson', '1998-06-25', NULL, 13) データを含む新しい行がパーティション `p2` に挿入されたことを簡単に判別できますが、チェーンで 21 の `st` ストアが追加されるとどうなりますか。このスキームでは、`store_id` が 20 よりも大きい行に対応するルールがなく、サーバーはどこに置くべきかわからないため、エラーになります。これが発生しないようにするには、「すべての状況に対応する」`VALUES LESS THAN` 句を `CREATE TABLE` ステートメントで使用して、明示的に指定されている最大値を超えるすべての値に備えます。

```
CREATE TABLE employees (  
  id INT NOT NULL,  
  fname VARCHAR(30),  
  lname VARCHAR(30),  
  hired DATE NOT NULL DEFAULT '1970-01-01',  
  separated DATE NOT NULL DEFAULT '9999-12-31',  
  job_code INT NOT NULL,  
  store_id INT NOT NULL  
)  
PARTITION BY RANGE (store_id) (  
  PARTITION p0 VALUES LESS THAN (6),  
  PARTITION p1 VALUES LESS THAN (11),  
  PARTITION p2 VALUES LESS THAN (16),  
  PARTITION p3 VALUES LESS THAN MAXVALUE  
);
```

(この章のその他の例と同様に、デフォルトのストレージエンジンは `InnoDB` であると想定しています。)

注記

一致する値が見つからないときにエラーを回避するための別の方法は、`INSERT` ステートメントの一部として `IGNORE` キーワードを使用することです。例については、[セクション 24.2.2 「LIST パーティショニング」](#) を参照してください。また、`IGNORE` の一般的な情報については、[セクション 13.2.6 「INSERT ステートメント」](#) を参照してください。

`MAXVALUE` は、可能な最大整数値よりも確実に大きな整数値を表します (数学用語では、上限です)。これによって、`store_id` カラム値が 16 (定義されている最大値) 以上である行は、パーティション `p3` に格納されます。将来のある時点で、店舗の数が 25、30、またはそれ以上に増加したときは、`ALTER TABLE` ステートメントを使用して、店舗 21-25、26-30 などのための新しいパーティションを追加できます (これを行う方法の詳細については、[セクション 24.3 「パーティション管理」](#) を参照してください)。

同様に、従業員ジョブコードに基づいて (つまり、`job_code` カラム値の範囲に基づいて) テーブルをパーティション化できます。たとえば、正規 (インストア) 従業員に 2 桁のジョブコード、オフィスおよびサポート従業員に 3 桁のコード、および管理職に 4 桁のコードが使用されると想定すると、次のステートメントを使用してパーティション化されたテーブルを作成できます。

```
CREATE TABLE employees (  
  id INT NOT NULL,  
  fname VARCHAR(30),  
  lname VARCHAR(30),  
  hired DATE NOT NULL DEFAULT '1970-01-01',
```



```

separated DATE NOT NULL DEFAULT '9999-12-31',
job_code INT NOT NULL,
store_id INT NOT NULL
)
PARTITION BY RANGE (job_code) (
  PARTITION p0 VALUES LESS THAN (100),
  PARTITION p1 VALUES LESS THAN (1000),
  PARTITION p2 VALUES LESS THAN (10000)
);

```

この例では、インストア従業員に関連するすべての行はパーティション **p0**、オフィスおよびサポートスタッフに関連するものは **p1**、および管理職に関連するものはパーティション **p2** に格納されます。

VALUES LESS THAN 句に式を使用することもできます。ただし、MySQL は、**LESS THAN (<)** 比較の一環として式の戻り値を評価できる必要があります。

店舗番号に従ってテーブルデータを分割するのではなく、代わりに 2 つの **DATE** カラムのうちの 1 つに基づく式を使用できます。たとえば、各従業員が会社を退職した年 (つまり、**YEAR(separated)** の値) に基づいてパーティション化するとします。そのようなパーティショニングスキームを実装する **CREATE TABLE** ステートメントの例を次に示します。

```

CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT,
  store_id INT
)
PARTITION BY RANGE ( YEAR(separated) ) (
  PARTITION p0 VALUES LESS THAN (1991),
  PARTITION p1 VALUES LESS THAN (1996),
  PARTITION p2 VALUES LESS THAN (2001),
  PARTITION p3 VALUES LESS THAN MAXVALUE
);

```

このスキームでは、1991 年より前に退職したすべての従業員の場合、行はパーティション **p0** に格納されます。1991 年から 1995 年までに退職した人は **p1**、1996 年から 2000 年までに退職した人は **p2**、および 2000 年よりあとに退職した従業員は **p3** に格納されます。

次の例に示すように、**UNIX_TIMESTAMP()** 関数を使用して、**TIMESTAMP** カラムの値に基づいて、**RANGE** によってテーブルをパーティション化することもできます。

```

CREATE TABLE quarterly_report_status (
  report_id INT NOT NULL,
  report_status VARCHAR(20) NOT NULL,
  report_updated TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)
PARTITION BY RANGE ( UNIX_TIMESTAMP(report_updated) ) (
  PARTITION p0 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-01-01 00:00:00') ),
  PARTITION p1 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-04-01 00:00:00') ),
  PARTITION p2 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-07-01 00:00:00') ),
  PARTITION p3 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-10-01 00:00:00') ),
  PARTITION p4 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-01-01 00:00:00') ),
  PARTITION p5 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-04-01 00:00:00') ),
  PARTITION p6 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-07-01 00:00:00') ),
  PARTITION p7 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-10-01 00:00:00') ),
  PARTITION p8 VALUES LESS THAN ( UNIX_TIMESTAMP('2010-01-01 00:00:00') ),
  PARTITION p9 VALUES LESS THAN (MAXVALUE)
);

```

TIMESTAMP 値を含むほかの式は許可されません (Bug #42849 を参照してください)。

次の条件の 1 つ以上が true のときは、**RANGE** パーティショニングが特に役立ちます。

- 「古い」データを削除したい、またはする必要がある。前述の **employees** テーブルのパーティション化スキームを使用している場合は、**ALTER TABLE employees DROP PARTITION p0;**を使用して、1991 年より前に企業の勤務を停止した従業員に関連するすべての行を削除できます。(詳細は、[セクション13.1.9「ALTER TABLE ステートメント](#)

ント」およびセクション24.3「パーティション管理」を参照してください)。テーブルに多数の行がある場合、これは `DELETE FROM employees WHERE YEAR(separated) <= 1990;` などの `DELETE` クエリーを実行するよりもはるかに効率的な場合があります。

- 日付または時間値、または何らかのほかの一連値から生じる値が含まれるカラムを使用したい。
- テーブルのパーティショニングに使用されるカラムに直接依存するクエリーを頻繁に実行する。たとえば、`EXPLAIN SELECT COUNT(*) FROM employees WHERE separated BETWEEN '2000-01-01' AND '2000-12-31' GROUP BY store_id;`などのクエリーを実行する場合、MySQL では、`WHERE` 句を満たすレコードを残りのパーティションに含めることができないため、パーティション `p2` のみをスキャンする必要があると迅速に判断できます。これがどのように実現されるかについての詳細は、セクション24.4「パーティションプルーニング」を参照してください。

このタイプのパーティショニングのバリエーションが `RANGE COLUMNS` パーティショニングです。 `RANGE COLUMNS` によるパーティショニングでは、複数のカラムを使用してパーティショニング範囲を定義できます (パーティション内の行の配置、およびパーティションプルーニングを実行するときに特定のパーティションの包含または除外を判断する際に適用されます)。詳細は、セクション24.2.3.1「`RANGE COLUMNS` パーティショニング」を参照してください。

時間間隔に基づくパーティショニングスキーム。MySQL 8.0 で時間の範囲または間隔に基づいてパーティショニングスキームを実装する場合は、2つの方法があります。

1. 次のように、`RANGE` によってテーブルをパーティション化し、パーティショニング式に `DATE`、`TIME`、または `DATETIME` カラムを操作して整数値を返す関数を使用します。

```
CREATE TABLE members (
  firstname VARCHAR(25) NOT NULL,
  lastname VARCHAR(25) NOT NULL,
  username VARCHAR(16) NOT NULL,
  email VARCHAR(35),
  joined DATE NOT NULL
)
PARTITION BY RANGE( YEAR(joined) ) (
  PARTITION p0 VALUES LESS THAN (1960),
  PARTITION p1 VALUES LESS THAN (1970),
  PARTITION p2 VALUES LESS THAN (1980),
  PARTITION p3 VALUES LESS THAN (1990),
  PARTITION p4 VALUES LESS THAN MAXVALUE
);
```

MySQL 8.0 では、次の例に示すように `UNIX_TIMESTAMP()` 関数を使用して、`TIMESTAMP` カラムの値に基づいて `RANGE` によってテーブルをパーティション化することもできます。

```
CREATE TABLE quarterly_report_status (
  report_id INT NOT NULL,
  report_status VARCHAR(20) NOT NULL,
  report_updated TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)
PARTITION BY RANGE ( UNIX_TIMESTAMP(report_updated) ) (
  PARTITION p0 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-01-01 00:00:00') ),
  PARTITION p1 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-04-01 00:00:00') ),
  PARTITION p2 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-07-01 00:00:00') ),
  PARTITION p3 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-10-01 00:00:00') ),
  PARTITION p4 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-01-01 00:00:00') ),
  PARTITION p5 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-04-01 00:00:00') ),
  PARTITION p6 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-07-01 00:00:00') ),
  PARTITION p7 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-10-01 00:00:00') ),
  PARTITION p8 VALUES LESS THAN ( UNIX_TIMESTAMP('2010-01-01 00:00:00') ),
  PARTITION p9 VALUES LESS THAN (MAXVALUE)
);
```

MySQL 8.0 では、`TIMESTAMP` 値を含むほかの式は許可されません。(Bug #42849 を参照してください)。

注記

MySQL 8.0 では、`LIST` によってパーティション化されるテーブルのパーティショニング式として `UNIX_TIMESTAMP(timestamp_column)` を使用することもできます。ただし、このようにするのは通常は実用的ではありません。

2. **DATE** または **DATETIME** カラムをパーティショニングカラムとして使用して、**RANGE COLUMNS** によってテーブルをパーティション化します。たとえば、次のように **joined** カラムを直接使用して **members** テーブルを定義できます。

```
CREATE TABLE members (
  firstname VARCHAR(25) NOT NULL,
  lastname VARCHAR(25) NOT NULL,
  username VARCHAR(16) NOT NULL,
  email VARCHAR(35),
  joined DATE NOT NULL
)
PARTITION BY RANGE COLUMNS(joined) (
  PARTITION p0 VALUES LESS THAN ('1960-01-01'),
  PARTITION p1 VALUES LESS THAN ('1970-01-01'),
  PARTITION p2 VALUES LESS THAN ('1980-01-01'),
  PARTITION p3 VALUES LESS THAN ('1990-01-01'),
  PARTITION p4 VALUES LESS THAN MAXVALUE
);
```

注記

DATE または **DATETIME** 以外の日付または時間型を使用してパーティショニングカラムを使用することは、**RANGE COLUMNS** ではサポートされません。

24.2.2 LIST パーティショニング

MySQL の LIST パーティショニングは、多くの点で RANGE パーティショニングに似ています。RANGE によるパーティショニングと同様に、各パーティションを明示的に定義する必要があります。2つのタイプのパーティショニングの主な違いは、LIST パーティショニングでは、各パーティションが、連続する値の範囲のセットのいずれかではなく、値リストのセットのいずれかに含まれるカラム値のメンバーシップに基づいて定義および選択されることです。これを行うには、**PARTITION BY LIST (expr)** を使用します。ここで、**expr** はカラム値またはカラム値に基づく式で、整数値を返し、**VALUES IN (value_list)** で各パーティションを定義します。ここで、**value_list** はカンマで区切られた整数のリストです。

注記

MySQL 8.0 では、LIST によってパーティション化するとき、整数 (および NULL も可。セクション24.2.7「MySQL パーティショニングによる NULL の扱い」を参照してください) のリストに対してのみ照合できます。

ただし、LIST COLUMN パーティショニングを使用するときは、ほかのカラムタイプを値リストで使用できます (これについては、このセクションで後述します)。

範囲で定義されるパーティションの場合と異なり、リストパーティションは特定の順序で宣言する必要はありません。構文についての詳細は、セクション13.1.20「CREATE TABLE ステートメント」を参照してください。

以降の例では、パーティション化するテーブルの基本定義が、次に示す CREATE TABLE ステートメントによって提供されることを想定します。

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT,
  store_id INT
);
```

(これは、セクション24.2.1「RANGE パーティショニング」の例の基礎として使用されるテーブルと同じです。他のパーティション化の例と同様に、**default_storage_engine** は InnoDB であると想定しています。)

次の表に示すように、20 のビデオ店があり、それらが 4 つのフランチャイズに分類されていると想定します。

地域	店舗 ID 番号
北	3、5、6、9、17

地域	店舗 ID 番号
東	1、2、10、11、19、20
西	4、12、13、14、18
中央	7、8、15、16

同じ地域に属する店舗の行が同じパーティションに格納されるようにこのテーブルをパーティション化するには、次のような `CREATE TABLE` ステートメントを使用できます。

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT,
  store_id INT
)
PARTITION BY LIST(store_id) (
  PARTITION pNorth VALUES IN (3,5,6,9,17),
  PARTITION pEast VALUES IN (1,2,10,11,19,20),
  PARTITION pWest VALUES IN (4,12,13,14,18),
  PARTITION pCentral VALUES IN (7,8,15,16)
);
```

これにより、特定の地域に関連する従業員レコードをテーブルで簡単に追加または削除できるようになります。たとえば、西地域の全店舗が別の会社に売却されたとします。MySQL 8.0 では、その地域の店舗で働いていた従業員に関連するすべての行をクエリー `ALTER TABLE employees TRUNCATE PARTITION pWest` を使用して削除できます。これは、同等の `DELETE` ステートメント `DELETE FROM employees WHERE store_id IN (4,12,13,14,18)`; よりもはるかに効率的に実行できます (`ALTER TABLE employees DROP PARTITION pWest` を使用してもこれらのすべての行が削除されますが、テーブルの定義からパーティション `pWest` も削除されるため、`ALTER TABLE ... ADD PARTITION` ステートメントを使用してテーブルの元のパーティショニングスキームをリストアする必要があります)。

RANGE パーティショニングと同様に、**LIST** パーティショニングとハッシュまたはキーによるパーティショニングを組み合わせることによって、複合パーティショニング (サブパーティショニング) を生成できます。 [セクション 24.2.6 「サブパーティショニング」](#) を参照してください。

RANGE パーティショニングの場合と異なり、**MAXVALUE** などの「すべての状況に対応する」ものはありません。パーティショニング式で予期されるすべての値を `PARTITION ... VALUES IN (...)` 句で指定してください。一致しないパーティショニングカラム値が含まれている `INSERT` ステートメントは、次の例に示すように、エラーで失敗します。

```
mysql> CREATE TABLE h2 (
-> c1 INT,
-> c2 INT
-> )
-> PARTITION BY LIST(c1) (
-> PARTITION p0 VALUES IN (1, 4, 7),
-> PARTITION p1 VALUES IN (2, 5, 8)
-> );
Query OK, 0 rows affected (0.11 sec)

mysql> INSERT INTO h2 VALUES (3, 5);
ERROR 1525 (HY000): Table has no partition for value 3
```

単一の `INSERT` ステートメントを使用して複数の行を単一の **InnoDB** テーブルに挿入する場合、**InnoDB** はそのステートメントを単一のトランザクションとみなし、一致しない値があるとステートメントが完全に失敗するため、行は挿入されません。

このタイプのエラーは、**IGNORE** キーワードを使用することで無視させることができます。そうした場合、一致しないパーティショニングカラム値が含まれる行は挿入されませんが、一致する値を持つ行は挿入されてエラーが報告されません。

```
mysql> TRUNCATE h2;
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SELECT * FROM h2;
Empty set (0.00 sec)

mysql> INSERT IGNORE INTO h2 VALUES (2, 5), (6, 10), (7, 5), (3, 1), (1, 9);
Query OK, 3 rows affected (0.00 sec)
Records: 5 Duplicates: 2 Warnings: 0

mysql> SELECT * FROM h2;
+-----+-----+
| c1 | c2 |
+-----+-----+
| 7 | 5 |
| 1 | 9 |
| 2 | 5 |
+-----+-----+
3 rows in set (0.00 sec)
```

MySQL 8.0 では、[LIST COLUMNS](#) パーティション化もサポートされています。これは、カラムのパーティション化に整数型以外の型のカラムを使用したり、複数のカラムをパーティション化キーとして使用できる [LIST](#) パーティション化のバリエーションです。詳細は、[セクション24.2.3.2「LIST COLUMNS パーティショニング」](#)を参照してください。

24.2.3 COLUMNS パーティショニング

次の2つのセクションでは、[RANGE](#) および [LIST](#) パーティショニングのバリエーションである [COLUMNS](#) パーティショニングについて説明します。[COLUMNS](#) パーティショニングでは、パーティショニングキーに複数のカラムを使用できます。これらのすべてのカラムが、パーティションに行を配置するため、およびパーティションプルーフでのパーティションで一致する行をチェックするかを判断するという両方の目的のために考慮されます。

また、[RANGE COLUMNS](#) パーティショニングおよび [LIST COLUMNS](#) パーティショニングの両方が、値範囲またはリストメンバーの定義のために整数以外のカラムの使用をサポートします。許可されるデータ型を次のリストに示します。

- すべての整数型: [TINYINT](#)、[SMALLINT](#)、[MEDIUMINT](#)、[INT \(INTEGER\)](#)、および [BIGINT](#) (これは、[RANGE](#) および [LIST](#) によるパーティショニングと同じです)。

ほかの数値データ型 ([DECIMAL](#)、[FLOAT](#) など) はパーティショニングカラムとしてサポートされません。

- [DATE](#) および [DATETIME](#)。

日付または時間に関連するほかのデータ型を使用するカラムは、パーティショニングカラムとしてサポートされません。

- 次の文字列型: [CHAR](#)、[VARCHAR](#)、[BINARY](#)、および [VARBINARY](#)。

[TEXT](#) カラムおよび [BLOB](#) カラムはパーティショニングカラムとしてサポートされません。

次の2つのセクションでの [RANGE COLUMNS](#) および [LIST COLUMNS](#) パーティショニングの説明では、MySQL 5.1以降でサポートされる範囲およびリストに基づくパーティショニングをすでに理解していることを想定しています。これらについての詳細は、[セクション24.2.1「RANGE パーティショニング」](#) および [セクション24.2.2「LIST パーティショニング」](#) をそれぞれ参照してください。

24.2.3.1 RANGE COLUMNS パーティショニング

[RANGE COLUMNS](#) パーティショニングは [RANGE](#) パーティショニングに似ていますが、複数のカラム値に基づく範囲を使用してパーティションを定義できます。また、整数型以外の型のカラムを使用して範囲を定義できます。

[RANGE COLUMNS](#) パーティショニングは、次の点で [RANGE](#) パーティショニングと大きく異なります。

- [RANGE COLUMNS](#) は式を受け入れません (カラムの名前のみ)。
- [RANGE COLUMNS](#) は 1 つ以上のカラムのリストを受け入れます。

[RANGE COLUMNS](#) パーティションは、スカラー値の比較ではなく、タプル (カラム値のリスト) の比較に基づきます。[RANGE COLUMNS](#) パーティションでの行の配置も、タプルの比較に基づきます。これについては、このセクションで後述します。

- **RANGE COLUMNS** パーティショニングカラムは整数カラムに制限されません。文字列、**DATE**、および **DATETIME** カラムもパーティショニングカラムとして使用できます。(詳細は、[セクション24.2.3「COLUMNS パーティショニング」](#)を参照してください。)

RANGE COLUMNS によってパーティション化されたテーブルを作成するための基本構文を次に示します。

```
CREATE TABLE table_name
PARTITIONED BY RANGE COLUMNS(column_list) (
  PARTITION partition_name VALUES LESS THAN (value_list),
  PARTITION partition_name VALUES LESS THAN (value_list)],
...)
)

column_list:
column_name[, column_name][, ...]

value_list:
value[, value][, ...]
```

注記

パーティション化されたテーブルを作成するときに使用できる **CREATE TABLE** オプションをすべて示しているわけではありません。詳細は、[セクション13.1.20「CREATE TABLE ステートメント」](#)を参照してください。

前述の構文で、**column_list** は 1 つ以上のカラムのリスト (パーティショニングカラムリストと呼ばれることもあります)、**value_list** は値のリスト (つまり、パーティション定義値リストです) です。**value_list** は各パーティション定義に指定する必要があり、各 **value_list** には **column_list** のカラムと同じ数の値が必要です。一般的に、**COLUMNS** 句に **N** 個のカラムを使用する場合は、各 **VALUES LESS THAN** 句にも **N** 個の値のリストを指定する必要があります。

パーティショニングカラムリストおよび各パーティションを定義する値リスト内の要素は、同じ順序で指定する必要があります。また、値リスト内の各要素は、カラムリスト内の対応する要素と同じデータ型である必要があります。ただし、パーティショニングカラムリストおよび値リスト内のカラム名の順序は、**CREATE TABLE** ステートメントの主要部内のテーブルカラム定義の順序と同じである必要はありません。**RANGE** によってパーティション化されるテーブルと同様に、**MAXVALUE** を使用して、指定されたカラムに挿入される正当な値より確実に大きな値を表すことができます。これらの点をすべて説明するために役立つ **CREATE TABLE** ステートメントの例を次に示します。

```
mysql> CREATE TABLE rcx (
-> a INT,
-> b INT,
-> c CHAR(3),
-> d INT
-> )
-> PARTITION BY RANGE COLUMNS(a,d,c) (
-> PARTITION p0 VALUES LESS THAN (5,10,'ggg'),
-> PARTITION p1 VALUES LESS THAN (10,20,'mmm'),
-> PARTITION p2 VALUES LESS THAN (15,30,'sss'),
-> PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE,MAXVALUE)
-> );
Query OK, 0 rows affected (0.15 sec)
```

テーブル **rcx** にはカラム **a**、**b**、**c**、および **d** が含まれています。**COLUMNS** 句に指定されたパーティショニングカラムリストには、これらのカラムのうちの 3 つが **a**、**d**、および **c** の順に使用されています。パーティションを定義するために使用される各値リストには、同じ順序で 3 つの値が含まれます。つまり、各値リストタプルの形式は、カラム **a**、**d**、および **c** が (この順序で) 使用するデータ型に対応する、(**INT**, **INT**, **CHAR(3)**) です。

パーティションに行がどのように配置されるかは、挿入される行のタプル (**COLUMNS** 句内のカラムリストに一致) と **VALUES LESS THAN** 句に使用されるタプル (テーブルのパーティションを定義) を比較することによって判断されます。スカラー値ではなくタプル (つまり、値のリストまたはセット) を比較するため、**RANGE COLUMNS** パーティションで使用される **VALUES LESS THAN** のセマンティクスは、単純な **RANGE** パーティションの場合とは若干異なります。**RANGE** パーティショニングでは、**VALUES LESS THAN** 内の制限値と等しい式値を生成する行は、対応するパーティションに配置されません。ただし、**RANGE COLUMNS** パーティショニングを使用するときは、パーティショニングカラムリストの最初の要素が **VALUES LESS THAN** 値リスト内の最初の要素と値が等しい行が、対応するパーティションに配置されることがあります。

次のステートメントによって作成されるパーティション化された **RANGE** テーブルを検討します。

```
CREATE TABLE r1 (
  a INT,
  b INT
)
PARTITION BY RANGE (a) (
  PARTITION p0 VALUES LESS THAN (5),
  PARTITION p1 VALUES LESS THAN (MAXVALUE)
);
```

各行の `a` のカラム値が `5` である 3 つの行をこのテーブルに挿入する場合、各行の `a` カラム値が `5` 以上であるため、3 行がすべてパーティション `p1` に格納されます。これは、`INFORMATION_SCHEMA.PARTITIONS` テーブルに対して適切なクエリーを実行することによって確認できます。

```
mysql> INSERT INTO r1 VALUES (5,10), (5,11), (5,12);
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0
```

```
mysql> SELECT PARTITION_NAME, TABLE_ROWS
-> FROM INFORMATION_SCHEMA.PARTITIONS
-> WHERE TABLE_NAME = 'r1';
+-----+-----+
| PARTITION_NAME | TABLE_ROWS |
+-----+-----+
| p0             | 0           |
| p1             | 3           |
+-----+-----+
2 rows in set (0.00 sec)
```

ここで、次のように作成される、カラム `a` および `b` の両方が `COLUMNS` 句で参照される、`RANGE COLUMNS` パーティショニングを使用する同様のテーブル `rc1` を検討します。

```
CREATE TABLE rc1 (
  a INT,
  b INT
)
PARTITION BY RANGE COLUMNS(a, b) (
  PARTITION p0 VALUES LESS THAN (5, 12),
  PARTITION p3 VALUES LESS THAN (MAXVALUE, MAXVALUE)
);
```

`r1` に挿入したのとまったく同じ行を `rc1` に挿入した場合、行の配分はかなり異なります。

```
mysql> INSERT INTO rc1 VALUES (5,10), (5,11), (5,12);
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0
```

```
mysql> SELECT PARTITION_NAME, TABLE_ROWS
-> FROM INFORMATION_SCHEMA.PARTITIONS
-> WHERE TABLE_NAME = 'rc1';
+-----+-----+-----+
| TABLE_SCHEMA | PARTITION_NAME | TABLE_ROWS |
+-----+-----+-----+
| p             | p0             | 2           |
| p             | p1             | 1           |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

これは、スカラー値ではなく行を比較しているためです。挿入された行値と、テーブル `rc1` のパーティション `p0` を定義するために使用された `VALUES THAN LESS THAN` 句の行制限値とを次のように比較できます。

```
mysql> SELECT (5,10) < (5,12), (5,11) < (5,12), (5,12) < (5,12);
+-----+-----+-----+
| (5,10) < (5,12) | (5,11) < (5,12) | (5,12) < (5,12) |
+-----+-----+-----+
| 1 | 1 | 0 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

2 つのタプル `(5,10)` および `(5,11)` は `(5,12)` より小さいと評価されるため、パーティション `p0` に格納されています。5 は 5 以上、12 は 12 以上であるため、`(5,12)` は `(5,12)` 以上と見なされ、パーティション `p1` に格納されています。

前の例の `SELECT` ステートメントは、次のように明示的な行コンストラクタを使用して記述することもできました。


```
SELECT ROW(5,10) < ROW(5,12), ROW(5,11) < ROW(5,12), ROW(5,12) < ROW(5,12);
```

MySQL で行コンストラクタを使用する方法についての詳細は、[セクション13.2.11.5「行サブクエリー」](#)を参照してください。

単一パーティショニングカラムのみを使用して **RANGE COLUMNS** によってパーティション化されたテーブルの場合、パーティションへの行の格納は **RANGE** によってパーティション化された同等のテーブルの場合と同じです。次の **CREATE TABLE** ステートメントでは、1つのパーティショニングカラムを使用して **RANGE COLUMNS** によってパーティション化されるテーブルが作成されます。

```
CREATE TABLE rx (
  a INT,
  b INT
)
PARTITION BY RANGE COLUMNS (a) (
  PARTITION p0 VALUES LESS THAN (5),
  PARTITION p1 VALUES LESS THAN (MAXVALUE)
);
```

このテーブルに行 (5,10)、(5,11)、および (5,12) を挿入する場合、それらの配置は、前に作成して移入したテーブル r の場合と同じです。

```
mysql> INSERT INTO rx VALUES (5,10), (5,11), (5,12);
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0
```

```
mysql> SELECT PARTITION_NAME, TABLE_ROWS
-> FROM INFORMATION_SCHEMA.PARTITIONS
-> WHERE TABLE_NAME = 'rx';
+-----+-----+-----+
| TABLE_SCHEMA | PARTITION_NAME | TABLE_ROWS |
+-----+-----+-----+
| p             | p0              | 0           |
| p             | p1              | 3           |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

1つ以上のカラムの制限値が連続するパーティション定義で繰り返される、**RANGE COLUMNS** によってパーティション化されるテーブルを作成することもできます。これを行うには、パーティションを定義するために使用されるカラム値のタプルが厳密に増加する必要があります。たとえば、次の各 **CREATE TABLE** ステートメントは有効です。

```
CREATE TABLE rc2 (
  a INT,
  b INT
)
PARTITION BY RANGE COLUMNS(a,b) (
  PARTITION p0 VALUES LESS THAN (0,10),
  PARTITION p1 VALUES LESS THAN (10,20),
  PARTITION p2 VALUES LESS THAN (10,30),
  PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE)
);

CREATE TABLE rc3 (
  a INT,
  b INT
)
PARTITION BY RANGE COLUMNS(a,b) (
  PARTITION p0 VALUES LESS THAN (0,10),
  PARTITION p1 VALUES LESS THAN (10,20),
  PARTITION p2 VALUES LESS THAN (10,30),
  PARTITION p3 VALUES LESS THAN (10,35),
  PARTITION p4 VALUES LESS THAN (20,40),
  PARTITION p5 VALUES LESS THAN (MAXVALUE,MAXVALUE)
);
```

次のステートメントも成功しますが、カラム **b** の制限値がパーティション **p0** で 25 およびパーティション **p1** で 20、カラム **c** の制限値がパーティション **p1** で 100 およびパーティション **p2** で 50 であるため、一見したところではそうでないように見えるかもしれません。

```
CREATE TABLE rc4 (
```

```

a INT,
b INT,
c INT
)
PARTITION BY RANGE COLUMNS(a,b,c) (
  PARTITION p0 VALUES LESS THAN (0,25,50),
  PARTITION p1 VALUES LESS THAN (10,20,100),
  PARTITION p2 VALUES LESS THAN (10,30,50)
  PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE,MAXVALUE)
);

```

RANGE COLUMNS によってパーティション化されるテーブルを設計するときは、次のように **mysql** クライアントを使用して目的のタプルを比較することで、いつでも連続するパーティション定義をテストできます。

```

mysql> SELECT (0,25,50) < (10,20,100), (10,20,100) < (10,30,50);
+-----+-----+
| (0,25,50) < (10,20,100) | (10,20,100) < (10,30,50) |
+-----+-----+
| 1 | 1 |
+-----+-----+
1 row in set (0.00 sec)

```

CREATE TABLE ステートメントに含まれるパーティション定義が、厳密にしないで増加しない順序である場合は、次の例に示すようにエラーで失敗します。

```

mysql> CREATE TABLE rcf (
->  a INT,
->  b INT,
->  c INT
-> )
-> PARTITION BY RANGE COLUMNS(a,b,c) (
->  PARTITION p0 VALUES LESS THAN (0,25,50),
->  PARTITION p1 VALUES LESS THAN (20,20,100),
->  PARTITION p2 VALUES LESS THAN (10,30,50),
->  PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE,MAXVALUE)
-> );
ERROR 1493 (HY000): VALUES LESS THAN value must be strictly increasing for each partition

```

そのようなエラーが発生するときは、それらのカラムリストの「小なり」比較を作成することで、無効なパーティション定義を推定できます。この場合、問題はパーティション **p2** の定義にあります。次に示すように、それを定義するために使用されるタプルが、パーティション **p3** を定義するために使用されるタプル以上であるためです。

```

mysql> SELECT (0,25,50) < (20,20,100), (20,20,100) < (10,30,50);
+-----+-----+
| (0,25,50) < (20,20,100) | (20,20,100) < (10,30,50) |
+-----+-----+
| 1 | 0 |
+-----+-----+
1 row in set (0.00 sec)

```

RANGE COLUMNS を使用するときには、複数の **VALUES LESS THAN** 句内の同じカラムに **MAXVALUE** を指定することもできます。ただし、それ以外の場合は、連続するパーティション定義内の個々のカラムの制限値ははだいに増加するべきであり、**MAXVALUE** をすべてのカラム値の上限として使用するパーティションは 1 つだけ定義するべきであり、このパーティション定義は **PARTITION ... VALUES LESS THAN** 句のリストの最後に指定するべきです。また、複数のパーティション定義の最初のカラムの制限値として **MAXVALUE** を使用することはできません。

前述したように、**RANGE COLUMNS** パーティショニングでは、整数以外のカラムをパーティショニングカラムとして使用することもできます (これらの完全な一覧については、[セクション 24.2.3 「COLUMNS パーティショニング」](#)を参照してください)。次のステートメントを使用して作成された **employees** という名前のテーブル (パーティション化されていません) を検討します。

```

CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30),
  hired DATE NOT NULL DEFAULT '1970-01-01',
  separated DATE NOT NULL DEFAULT '9999-12-31',
  job_code INT NOT NULL,
  store_id INT NOT NULL
);

```

RANGE COLUMNS パーティショニングを使用して、次のように従業員の姓に基づいて各行を 4 つのパーティションのいずれかに格納する、このテーブルのバージョンを作成できます。

```
CREATE TABLE employees_by_lname (  
  id INT NOT NULL,  
  fname VARCHAR(30),  
  lname VARCHAR(30),  
  hired DATE NOT NULL DEFAULT '1970-01-01',  
  separated DATE NOT NULL DEFAULT '9999-12-31',  
  job_code INT NOT NULL,  
  store_id INT NOT NULL  
)  
PARTITION BY RANGE COLUMNS (lname) (  
  PARTITION p0 VALUES LESS THAN ('g'),  
  PARTITION p1 VALUES LESS THAN ('m'),  
  PARTITION p2 VALUES LESS THAN ('t'),  
  PARTITION p3 VALUES LESS THAN (MAXVALUE)  
);
```

または、次の **ALTER TABLE** ステートメントを実行することで、前に作成した **employees** テーブルをこのスキームを使用してパーティション化できます。

```
ALTER TABLE employees PARTITION BY RANGE COLUMNS (lname) (  
  PARTITION p0 VALUES LESS THAN ('g'),  
  PARTITION p1 VALUES LESS THAN ('m'),  
  PARTITION p2 VALUES LESS THAN ('t'),  
  PARTITION p3 VALUES LESS THAN (MAXVALUE)  
);
```

注記

文字セットおよび照合順序が異なるとソート順序が異なるため、文字列カラムをパーティショニングカラムとして使用するとき、使用している文字セットおよび照合順序が、指定された行が **RANGE COLUMNS** によってパーティション化されるテーブルのどのパーティションに格納されるかに影響することがあります。また、そのようなテーブルが作成されたあとに指定されたデータベース、テーブル、またはカラムの文字セットまたは照合順序を変更すると、行がどのように配分されるかが変わることがあります。たとえば、大/小文字を区別する照合を使用する場合、**'and'**は**'Andersen'**の前にソートしますが、大/小文字を区別しない照合を使用する場合、その逆が当てはまります。

MySQL が文字セットおよび照合順序をどのように扱うかについては、[第10章「文字セット、照合順序、Unicode」](#)を参照してください。

同様に、**employees** テーブルを、各行がいくつかの 10 年間(その間に対応する従業員が雇用された)ベースのパーティションのいずれかに格納されるように、次のような **ALTER TABLE** ステートメントを使用してパーティション化できます。

```
ALTER TABLE employees PARTITION BY RANGE COLUMNS (hired) (  
  PARTITION p0 VALUES LESS THAN ('1970-01-01'),  
  PARTITION p1 VALUES LESS THAN ('1980-01-01'),  
  PARTITION p2 VALUES LESS THAN ('1990-01-01'),  
  PARTITION p3 VALUES LESS THAN ('2000-01-01'),  
  PARTITION p4 VALUES LESS THAN ('2010-01-01'),  
  PARTITION p5 VALUES LESS THAN (MAXVALUE)  
);
```

PARTITION BY RANGE COLUMNS 構文の詳細については、[セクション13.1.20「CREATE TABLE ステートメント」](#)を参照してください。

24.2.3.2 LIST COLUMNS パーティショニング

MySQL 8.0 は **LIST COLUMNS** パーティショニングのサポートを提供します。これは **LIST** パーティショニングのバリエーションで、複数のカラムをパーティションキーとして使用でき、整数型以外のデータ型のカラムをパーティショニングカラムとして使用できます。文字列型、**DATE**、および **DATETIME** カラムを使用できます (**COLUMNS** パーティショニングカラムに許可されるデータ型の詳細については、[セクション24.2.3「COLUMNS パーティショニング」](#)を参照してください)。

ある会社の顧客が 12 の都市に存在し、販売およびマーケティングのために、それらを次の表に示すように 3 つの都市で構成される 4 つの地域に分類すると想定します。

地域	都市
1	Oskarshamn、Högsby、Mönsterås
2	Vimmerby、Hultsfred、Västervik
3	Nässjö、Eksjö、Vetlanda
4	Uppvidinge、Alvesta、Växjö

LIST COLUMNS パーティショニングでは、ここで示すように、顧客が所在する都市の名前に基づいてこれらの地域に対応する 4 つのパーティションのいずれかに行を割り当てる、顧客データのテーブルを作成できます。

```
CREATE TABLE customers_1 (
  first_name VARCHAR(25),
  last_name VARCHAR(25),
  street_1 VARCHAR(30),
  street_2 VARCHAR(30),
  city VARCHAR(15),
  renewal DATE
)
PARTITION BY LIST COLUMNS(city) (
  PARTITION pRegion_1 VALUES IN('Oskarshamn', 'Högsby', 'Mönsterås'),
  PARTITION pRegion_2 VALUES IN('Vimmerby', 'Hultsfred', 'Västervik'),
  PARTITION pRegion_3 VALUES IN('Nässjö', 'Eksjö', 'Vetlanda'),
  PARTITION pRegion_4 VALUES IN('Uppvidinge', 'Alvesta', 'Växjö')
);
```

RANGE COLUMNS によるパーティショニングのように、**COLUMNS()** 句で式を使用してカラム値を整数に変換する必要はありません (実際、カラム名ではなく式を使用することは **COLUMNS()** では許可されません)。

DATE および **DATETIME** カラムを使用することもでき、次の例では、前に示した **customers_1** テーブルと同じ名前およびカラムを使用していますが、**renewal** カラムに基づく **LIST COLUMNS** パーティショニングを使用して、顧客のアカウントの更新がスケジュールされている 2010 年 2 月の週に応じて、4 つのパーティションのいずれかに行が格納されることを示しています。

```
CREATE TABLE customers_2 (
  first_name VARCHAR(25),
  last_name VARCHAR(25),
  street_1 VARCHAR(30),
  street_2 VARCHAR(30),
  city VARCHAR(15),
  renewal DATE
)
PARTITION BY LIST COLUMNS(renewal) (
  PARTITION pWeek_1 VALUES IN('2010-02-01', '2010-02-02', '2010-02-03',
    '2010-02-04', '2010-02-05', '2010-02-06', '2010-02-07'),
  PARTITION pWeek_2 VALUES IN('2010-02-08', '2010-02-09', '2010-02-10',
    '2010-02-11', '2010-02-12', '2010-02-13', '2010-02-14'),
  PARTITION pWeek_3 VALUES IN('2010-02-15', '2010-02-16', '2010-02-17',
    '2010-02-18', '2010-02-19', '2010-02-20', '2010-02-21'),
  PARTITION pWeek_4 VALUES IN('2010-02-22', '2010-02-23', '2010-02-24',
    '2010-02-25', '2010-02-26', '2010-02-27', '2010-02-28')
);
```

これは機能しますが、関係する日付の数が非常に多くなってきた場合に、定義および保守が面倒になります。そのような場合は通常、**RANGE** または **RANGE COLUMNS** パーティショニングを代わりに使用するほうが現実的です。この場合、パーティショニングキーとして使用するカラムは **DATE** カラムであるため、次に示すように **RANGE COLUMNS** パーティショニングを使用します。

```
CREATE TABLE customers_3 (
  first_name VARCHAR(25),
  last_name VARCHAR(25),
  street_1 VARCHAR(30),
  street_2 VARCHAR(30),
  city VARCHAR(15),
  renewal DATE
)
```

```
PARTITION BY RANGE COLUMNS(renewal) (  
  PARTITION pWeek_1 VALUES LESS THAN('2010-02-09'),  
  PARTITION pWeek_2 VALUES LESS THAN('2010-02-15'),  
  PARTITION pWeek_3 VALUES LESS THAN('2010-02-22'),  
  PARTITION pWeek_4 VALUES LESS THAN('2010-03-01')  
);
```

詳細は、[セクション24.2.3.1「RANGE COLUMNS パーティショニング」](#)を参照してください。

また ([RANGE COLUMNS](#) パーティショニングと同様に)、[COLUMNS\(\)](#) 句で複数のカラムを使用できます。

[PARTITION BY LIST COLUMNS\(\)](#) 構文についての詳細は、[セクション13.1.20「CREATE TABLE ステートメント」](#)を参照してください。

24.2.4 HASH パーティショニング

[HASH](#) によるパーティショニングは、事前に決められた数のパーティションにデータを均等に配分するために主に使用されます。レンジパーティション化またはリストパーティション化では、特定のカラム値またはカラム値のセットを格納するパーティションを明示的に指定する必要があります。ハッシュパーティション化では、この決定が行われ、ハッシュされるカラム値およびパーティションテーブルが分割されるパーティションの数に基づいてカラム値または式のみを指定する必要があります。

[HASH](#) パーティショニングを使用してテーブルをパーティション化する場合は、[CREATE TABLE](#) ステートメントに [PARTITION BY HASH \(expr\)](#) 句を付加する必要があります。ここで、[expr](#) は整数を返す式です。これは、型が MySQL 整数型のいずれかであるカラムの名前にすることができます。また、多くの場合、この後に [PARTITIONS num](#) を使用します。ここで、[num](#) は、テーブルが分割されるパーティションの数を表す正の整数です。

注記

わかりやすくするため、次の例のテーブルではキーを使用していません。テーブルに一意キーがある場合、このテーブルのパーティション化式で使用されるすべてのカラムは、主キーを含むすべての一意キーの一部である必要があります。詳しくは[セクション24.6.1「パーティショニングキー、主キー、および一意キー」](#)をご覧ください。

次のステートメントは、[store_id](#) カラムにハッシュを使用し、4 つのパーティションに分割されるテーブルを作成します：

```
CREATE TABLE employees (  
  id INT NOT NULL,  
  fname VARCHAR(30),  
  lname VARCHAR(30),  
  hired DATE NOT NULL DEFAULT '1970-01-01',  
  separated DATE NOT NULL DEFAULT '9999-12-31',  
  job_code INT,  
  store_id INT  
)  
PARTITION BY HASH(store_id)  
PARTITIONS 4;
```

[PARTITIONS](#) 句を含めない場合、パーティションの数はデフォルトで 1 になります。[PARTITIONS](#) キーワードを数字なしで使用すると、構文エラーが発生します。

整数を返す SQL 式を [expr](#) に使用することもできます。たとえば、従業員が雇用された年度に基づいてパーティション化するとします。これは、次のように行うことができます。

```
CREATE TABLE employees (  
  id INT NOT NULL,  
  fname VARCHAR(30),  
  lname VARCHAR(30),  
  hired DATE NOT NULL DEFAULT '1970-01-01',  
  separated DATE NOT NULL DEFAULT '9999-12-31',  
  job_code INT,  
  store_id INT  
)  
PARTITION BY HASH( YEAR(hired) )  
PARTITIONS 4;
```

`expr` は、定数以外のランダムではない整数値 (つまり、変化するけれども決定論的であるべき) を返す必要があり、[セクション24.6「パーティショニングの制約と制限」](#)で説明されている禁止された構造体を含んではいけません。また、この式は行が挿入または更新 (または場合によっては削除) されるたびに評価されるべきです。これは、非常に複雑な式がパフォーマンスの問題を起こすことがあることを意味します (特に、一度に多くの行に影響する操作 (バッチ挿入など) を実行するとき)。

もっとも効率的なハッシュ関数は、単一テーブルカラムに実行され、その値がカラム値に対して比例的に増加または減少するもので、これによってパーティションの範囲を「プルーニング」できます。つまり、式がそのベースのカラムの値に対してより密接に変化するほど、MySQL は式を HASH パーティショニングにより効率的に使用できます。

たとえば、`date_col` が `DATE` 型のカラムである場合、式 `TO_DAYS(date_col)` は `date_col` の値に正比例すると表現されます。`date_col` の値が変わるたびに、式の値が一定の方法で変化するためです。`date_col` に対する式 `YEAR(date_col)` の変化は、`TO_DAYS(date_col)` ほど比例的ではありません。`date_col` のあらゆる変化に対して `YEAR(date_col)` が同等に変化するとはかぎらないためです。それでも、`YEAR(date_col)` はハッシュ関数の良い候補の 1 つです。`date_col` の一部と正比例し、`date_col` の変化によって `YEAR(date_col)` で比例的でない変化が発生することがないためです。

比較のために、型が `INT` である `int_col` という名前のカラムがあるとします。式 `POW(5-int_col,3) + 6` を検討してみてください。これは、`int_col` の値が変化したときに、式の値に比例的に変化することが保証されないため、ハッシュ関数の良い候補ではありません。`int_col` の値を特定の量で変更すると、式の値に大きく異なる変更が生じる可能性があります。たとえば、`int_col` が 5 から 6 に変化すると、式の値が -1 に変化しますが、`int_col` の値が 6 から 7 に変化すると、式の値が -7 に変化します。

つまり、カラム値と式の値のグラフが、等式 $y=cx$ (ここで、 c はゼロでない何らかの定数) によって描かれるような直線に近くなるほど、その式はハッシュにより適切になります。これは、式が非直線的であるほど、パーティションに対するデータの配分が不均衡になる傾向があることに関係しています。

理論上は、複数のカラム値を使用する式をプルーニングすることもできますが、そのような式のどれが適しているかを判断するのがかなり難しく、時間がかかることがあります。このため、複数のカラムを含むハッシュ式を使用することはあまり推奨されていません。

`PARTITION BY HASH` を使用する場合、ストレージエンジンは式の結果のモジュラスに基づいて、使用する `num` パーティションのパーティションを決定します。つまり、特定の式 `expr` の場合、レコードが格納されるパーティションはパーティション番号 `N` で、 $N = \text{MOD}(\text{expr}, \text{num})$ です。テーブル `t1` が次のように 4 つのパーティションを持つように定義されているとします。

```
CREATE TABLE t1 (col1 INT, col2 CHAR(5), col3 DATE)
PARTITION BY HASH( YEAR(col3) )
PARTITIONS 4;
```

`t1` に `col3` 値が '2005-09-15' であるレコードを挿入した場合、それが格納されるパーティションは次のように判断されます。

```
MOD(YEAR('2005-09-01'),4)
= MOD(2005,4)
= 1
```

MySQL 8.0 では、線形ハッシングと呼ばれる `HASH` パーティション化のバリエーションもサポートされており、パーティションテーブルに挿入される新しい行の配置を決定するためのより複雑なアルゴリズムが採用されています。このアルゴリズムについては、[セクション24.2.4.1「LINEAR HASH パーティショニング」](#)を参照してください。

ユーザー指定の式は、レコードが挿入または更新されるたびに評価されます。状況によっては、レコードが削除されるときにも評価されることがあります。

24.2.4.1 LINEAR HASH パーティショニング

MySQL は線形ハッシュもサポートしています。通常のハッシュと異なるところは、線形ハッシュは線形二乗アルゴリズムを使用し、通常のハッシュはハッシュ関数の値の法を使用することです。

構文的には、リニアハッシュパーティショニングと通常のハッシュの唯一の違いは、次に示すように、`PARTITION BY` 句に `LINEAR` キーワードが追加されていることです。

```
CREATE TABLE employees (
```



```

id INT NOT NULL,
fname VARCHAR(30),
lname VARCHAR(30),
hired DATE NOT NULL DEFAULT '1970-01-01',
separated DATE NOT NULL DEFAULT '9999-12-31',
job_code INT,
store_id INT
)
PARTITION BY LINEAR HASH( YEAR(hired) )
PARTITIONS 4;

```

式 `expr` の場合、線形ハッシュが使用されるときにレコードが格納されるパーティションは、`num` パーティションのうちのパーティション番号 `N` です。ここで、`N` は次のアルゴリズムに従って導出されます。

1. `num` よりも大きい次の 2 の累乗を見つけます。この値を `V` と呼ぶことにします。これは次のように計算できません。

```
V = POWER(2, CEILING(LOG(2, num)))
```

(`num` が 13 であるとしします。その場合、`LOG(2,13)` は 3.7004397181411 です。`CEILING(3.7004397181411)` は 4、`V = POWER(2,4)` は 16 です。)

2. `N = F(column_list) & (V - 1)` を設定します。

3. `N >= num` の間:

- `V` を `V / 2` に設定
- `N = N & (V - 1)` を設定します

線形ハッシュパーティショニングを使用し、6 個のパーティションを持つテーブル `t1` を次のステートメントを使用して作成するとします。

```

CREATE TABLE t1 (col1 INT, col2 CHAR(5), col3 DATE)
PARTITION BY LINEAR HASH( YEAR(col3) )
PARTITIONS 6;

```

`col3` カラムの値が '2003-04-14' および '1998-10-19' である 2 つのレコードを `t1` に挿入するとします。これらの 1 番目のパーティション番号は次のように決定されます。

```

V = POWER(2, CEILING( LOG(2,6) )) = 8
N = YEAR('2003-04-14') & (8 - 1)
  = 2003 & 7
  = 3

```

(3 >= 6 is FALSE: record stored in partition #3)

2 番目のレコードが格納されるパーティションの番号は、次のように計算されます。

```

V = 8
N = YEAR('1998-10-19') & (8 - 1)
  = 1998 & 7
  = 6

```

(6 >= 6 is TRUE: additional step required)

```

N = 6 & ((8 / 2) - 1)
  = 6 & 3
  = 2

```

(2 >= 6 is FALSE: record stored in partition #2)

線形ハッシュによるパーティショニングの利点は、パーティションの追加、削除、マージ、および分割の速度が向上することです。これは、非常に大量 (テラバイト) のデータが含まれるテーブルを扱うときに利点になることがあります。欠点は、通常のハッシュパーティショニングを使用して獲得される配分と比べて、データがパーティションに均等に配分される可能性が低いことです。

24.2.5 KEY パーティショニング

キーによるパーティショニングはハッシュによるパーティショニングと似ていますが、ハッシュパーティショニングはユーザー定義の式を使用し、キーパーティショニング用のハッシュ関数は MySQL サーバーによって提供されます。NDB Cluster はこの目的のために `MD5()` を使用します。ほかのストレージエンジンを使用するテーブルの場合、サーバーは独自の内部ハッシュ関数を使用します。

`CREATE TABLE ... PARTITION BY KEY` の構文規則は、ハッシュによってパーティション化されたテーブルを作成する場合のものに似ています。主な違いを次に示します。

- `HASH` ではなく `KEY` が使用されます。
- `KEY` は、0 個以上のカラム名のリストのみを取ります。パーティショニングキーとして使用されるカラムは、テーブルの主キーの一部またはすべてを構成している必要があります (テーブルにそれがあある場合)。パーティショニングキーとしてカラム名を指定しない場合は、テーブルの主キーが使用されます (ある場合)。たとえば、次の `CREATE TABLE` ステートメントは MySQL 8.0 で有効です。

```
CREATE TABLE k1 (  
  id INT NOT NULL PRIMARY KEY,  
  name VARCHAR(20)  
)  
PARTITION BY KEY()  
PARTITIONS 2;
```

主キーはないけれども一意キーはある場合は、パーティショニングキーに一意キーが使用されます。

```
CREATE TABLE k1 (  
  id INT NOT NULL,  
  name VARCHAR(20),  
  UNIQUE KEY (id)  
)  
PARTITION BY KEY()  
PARTITIONS 2;
```

ただし、一意キーカラムが `NOT NULL` として定義されていない場合、前のステートメントは失敗します。

どちらの場合も、パーティショニングキーは `id` カラムです。ただし、`SHOW CREATE TABLE` の出力や `INFORMATION_SCHEMA.PARTITIONS` テーブルの `PARTITION_EXPRESSION` カラムには表示されません。

ほかのパーティショニングタイプの場合と異なり、`KEY` によるパーティショニングに使用されるカラムは、整数または `NULL` 値に制限されません。たとえば、次の `CREATE TABLE` ステートメントは有効です。

```
CREATE TABLE tm1 (  
  s1 CHAR(32) PRIMARY KEY  
)  
PARTITION BY KEY(s1)  
PARTITIONS 10;
```

ほかのパーティショニングタイプが指定された場合、前のステートメントは有効でなくなります (この場合、`s1` はテーブルの主キーであるため、単純に `PARTITION BY KEY()` を使用することも有効であり、`PARTITION BY KEY(s1)` と同じ効果があります)。

この問題の詳細については、[セクション24.6「パーティショニングの制約と制限」](#)を参照してください。

インデックス接頭辞を持つカラムは、パーティション化キーではサポートされていません。つまり、`CHAR`、`VARCHAR`、`BINARY` カラムおよび `VARBINARY` カラムは、接頭辞を採用していないかぎり、パーティション化キーで使用できます。インデックス定義で `BLOB` カラムおよび `TEXT` カラムに接頭辞を指定する必要があるため、これら 2 つのタイプのカラムをパーティション化キーで使用することはできません。MySQL 8.0.21 より前では、パーティション化されたテーブルの作成、変更、またはアップグレード時に、それらがテーブルのパーティショニングキーに含まれていなくても、接頭辞を使用するカラムは許可されていました。MySQL 8.0.21 以降では、この許容される動作は非推奨であり、そのようなカラムが使用されると、サーバーは適切な警告またはエラーを表示します。詳細および例については、[カラムインデックス接頭辞はキーパーティション化ではサポートされていません](#)を参照してください。

注記

NDB ストレージエンジンを使用するテーブルは、ほかの MySQL ストレージエンジンと同様に、テーブルの主キーをパーティショニングキーとして使用して、`KEY` によって暗黙

的にパーティション化されます。「NDB Cluster」テーブルに明示的な主キーがない場合は、NDB ストレージエンジンによって生成された各「NDB Cluster」テーブルの「非表示」主キーがパーティション化キーとして使用されます。

NDB テーブルに明示的なパーティショニングスキームを定義する場合は、テーブルに明示的な主キーが必要であり、パーティショニング式に使用されるカラムがこのキーの一部である必要があります。ただし、テーブルが「空」のパーティショニング式を使用する（つまり、カラム参照なしの `PARTITION BY KEY()`）場合、明示的な主キーは必要ありません。

このパーティショニングは、`ndb_desc` ユーティリティ (-p オプション付き) を使用して確認できます。

重要

キーによってパーティション化されたテーブルの場合は、`ALTER TABLE DROP PRIMARY KEY` を実行できません。それを実行すると次のエラーが生成されます: `ERROR 1466 (HY000): Field in list of fields for partition function not found in table`。これは、`KEY` によってパーティション化された「NDB Cluster」テーブルでは問題ではありません。このような場合、テーブルは「非表示」主キーをテーブルの新しいパーティション化キーとして使用して再編成されます。第23章「MySQL NDB Cluster 8.0」を参照してください。

リニアキーによってテーブルをパーティション化することもできます。次に、単純な例を示します。

```
CREATE TABLE tk (
  col1 INT NOT NULL,
  col2 CHAR(5),
  col3 DATE
)
PARTITION BY LINEAR KEY (col1)
PARTITIONS 3;
```

`LINEAR` キーワードは、`KEY` パーティション化と `HASH` パーティション化で同じ効果を持ち、パーティション番号はモジュロ演算ではなく 2 乗アルゴリズムを使用して導出されます。このアルゴリズムの説明およびその影響については、セクション 24.2.4.1「`LINEAR HASH` パーティショニング」を参照してください。

24.2.6 サブパーティショニング

サブパーティショニング (複合パーティショニングとも呼ばれます) は、パーティション化されたテーブルの各パーティションをさらに分割することです。次の `CREATE TABLE` ステートメントを検討します。

```
CREATE TABLE ts (id INT, purchased DATE)
PARTITION BY RANGE( YEAR(purchased) )
SUBPARTITION BY HASH( TO_DAYS(purchased) )
SUBPARTITIONS 2 (
  PARTITION p0 VALUES LESS THAN (1990),
  PARTITION p1 VALUES LESS THAN (2000),
  PARTITION p2 VALUES LESS THAN MAXVALUE
);
```

テーブル `ts` には 3 つの `RANGE` パーティションがあります。これらの各パーティション (`p0`、`p1`、および `p2`) は、さらに 2 つのサブパーティションに分割されます。実際には、テーブル全体が $3 * 2 = 6$ パーティションに分割されます。ただし、`PARTITION BY RANGE` 句のアクションによって、これらの最初の 2 つには `purchased` カラムで値が 1990 より小さいレコードのみが格納されます。

`RANGE` または `LIST` でパーティション化されたテーブルをサブパーティション化できます。サブパーティショニングには、`HASH` または `KEY` パーティショニングを使用できます。これは、複合パーティショニングとも呼ばれます。

注記

`SUBPARTITION BY HASH` および `SUBPARTITION BY KEY` は通常それぞれ、`PARTITION BY HASH` および `PARTITION BY KEY` と同じ構文規則に従います。これの例外

は、**SUBPARTITION BY KEY** は現在 (**PARTITION BY KEY** と異なり) デフォルトカラムをサポートしないことで、この目的に使用されるカラムを指定する必要があります (テーブルに明示的な主キーがある場合でも)。これは既知の問題であり、対処中です。詳細および例については [サブパーティションに関する問題](#) を参照してください。

SUBPARTITION 句を使用して個々のサブパーティションのオプションを指定することによって、サブパーティションを明示的に定義することもできます。たとえば、前の例と同じテーブル **ts** をより冗長な形式で作成するには、次のようになります。

```
CREATE TABLE ts (id INT, purchased DATE)
PARTITION BY RANGE( YEAR(purchased) )
SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
  PARTITION p0 VALUES LESS THAN (1990) (
    SUBPARTITION s0,
    SUBPARTITION s1
  ),
  PARTITION p1 VALUES LESS THAN (2000) (
    SUBPARTITION s2,
    SUBPARTITION s3
  ),
  PARTITION p2 VALUES LESS THAN MAXVALUE (
    SUBPARTITION s4,
    SUBPARTITION s5
  )
);
```

構文に関するいくつかの注意事項を次に示します。

- 各パーティションには、同じ数のサブパーティションが必要です。
- パーティション化されたテーブルのパーティションに **SUBPARTITION** を使用してサブパーティションを明示的に定義する場合は、それらのすべてを定義する必要があります。つまり、次のステートメントは失敗します:

```
CREATE TABLE ts (id INT, purchased DATE)
PARTITION BY RANGE( YEAR(purchased) )
SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
  PARTITION p0 VALUES LESS THAN (1990) (
    SUBPARTITION s0,
    SUBPARTITION s1
  ),
  PARTITION p1 VALUES LESS THAN (2000),
  PARTITION p2 VALUES LESS THAN MAXVALUE (
    SUBPARTITION s2,
    SUBPARTITION s3
  )
);
```

このステートメントは、**SUBPARTITIONS 2** を使用していても失敗します。

- 各 **SUBPARTITION** 句には、(少なくとも) サブパーティションの名前が含まれている必要があります。それ以外は、サブパーティションに適切なオプションを設定するか、またはそのオプションのデフォルト設定を想定します。
- サブパーティション名はテーブル全体で一意である必要があります。たとえば、次の **CREATE TABLE** ステートメントは有効です。

```
CREATE TABLE ts (id INT, purchased DATE)
PARTITION BY RANGE( YEAR(purchased) )
SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
  PARTITION p0 VALUES LESS THAN (1990) (
    SUBPARTITION s0,
    SUBPARTITION s1
  ),
  PARTITION p1 VALUES LESS THAN (2000) (
    SUBPARTITION s2,
    SUBPARTITION s3
  ),
  PARTITION p2 VALUES LESS THAN MAXVALUE (
    SUBPARTITION s4,
    SUBPARTITION s5
  )
);
```

```
)  
);
```

24.2.7 MySQL パーティショニングによる NULL の扱い

MySQL のパーティショニングには、パーティショニング式の値 (カラム値またはユーザー定義式の値にかかわらず) として **NULL** を拒否する手段はありません。式の値として **NULL** を使用することは許可されていますが (そうでない場合は整数を返す必要がある)、**NULL** は数値でないことを認識することは重要です。MySQL のパーティショニング実装は、**ORDER BY** のように、**NULL** でない値より小さい値として **NULL** を扱います。

これは、**NULL** の扱いは各タイプのパーティショニングで異なり、これに準備していない場合は予期しない動作になる可能性があることを意味します。このような状況があるので、このセクションでは、各 MySQL パーティショニングタイプが、行をどのパーティションに格納するべきかを判断するときに **NULL** 値をどのように扱うかを説明し、それぞれの例を示します。

RANGE パーティショニングでの **NULL** の扱い。パーティションを判断するために使用されるカラム値が **NULL** である行を、**RANGE** によってパーティション化されたテーブルに挿入した場合、行はもっとも低いパーティションに挿入されます。p という名前のデータベースに、次のように作成された 2 つのテーブルがあるとします。

```
mysql> CREATE TABLE t1 (  
-> c1 INT,  
-> c2 VARCHAR(20)  
-> )  
-> PARTITION BY RANGE(c1) (  
-> PARTITION p0 VALUES LESS THAN (0),  
-> PARTITION p1 VALUES LESS THAN (10),  
-> PARTITION p2 VALUES LESS THAN MAXVALUE  
-> );
```

Query OK, 0 rows affected (0.09 sec)

```
mysql> CREATE TABLE t2 (  
-> c1 INT,  
-> c2 VARCHAR(20)  
-> )  
-> PARTITION BY RANGE(c1) (  
-> PARTITION p0 VALUES LESS THAN (-5),  
-> PARTITION p1 VALUES LESS THAN (0),  
-> PARTITION p2 VALUES LESS THAN (10),  
-> PARTITION p3 VALUES LESS THAN MAXVALUE  
-> );
```

Query OK, 0 rows affected (0.09 sec)

これらの 2 つの **CREATE TABLE** ステートメントによって作成されたパーティションについては、次のクエリーを **INFORMATION_SCHEMA** データベース内の **PARTITIONS** テーブルに対して使用することで確認できます。

```
mysql> SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH, DATA_LENGTH  
> FROM INFORMATION_SCHEMA.PARTITIONS  
> WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME LIKE 't';
```

TABLE_NAME	PARTITION_NAME	TABLE_ROWS	AVG_ROW_LENGTH	DATA_LENGTH
t1	p0	0	0	0
t1	p1	0	0	0
t1	p2	0	0	0
t2	p0	0	0	0
t2	p1	0	0	0
t2	p2	0	0	0
t2	p3	0	0	0

7 rows in set (0.00 sec)

(このテーブルについての詳細は、[セクション26.21「INFORMATION_SCHEMA PARTITIONS テーブル」](#)を参照してください。)ここで、これらの各テーブルのパーティショニングキーとして使用されるカラムに **NULL** が含まれる単一行を移入し、2 つの **SELECT** ステートメントを使用してこれらの行が挿入されたことを確認します。

```
mysql> INSERT INTO t1 VALUES (NULL, 'mothra');  
Query OK, 1 row affected (0.00 sec)
```

```
mysql> INSERT INTO t2 VALUES (NULL, 'mothra');  
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SELECT * FROM t1;
+----+-----+
| id | name |
+----+-----+
| NULL | mothra |
+----+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT * FROM t2;
+----+-----+
| id | name |
+----+-----+
| NULL | mothra |
+----+-----+
1 row in set (0.00 sec)
```

挿入された行を格納するためにどのパーティションが使用されたかについては、前のクエリーを `INFORMATION_SCHEMA.PARTITIONS` に対して再実行して出力を検査することで確認できます。

```
mysql> SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH, DATA_LENGTH
> FROM INFORMATION_SCHEMA.PARTITIONS
> WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME LIKE 't_*';
+-----+-----+-----+-----+-----+
| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+-----+-----+-----+-----+-----+
| t1 | p0 | 1 | 20 | 20 |
| t1 | p1 | 0 | 0 | 0 |
| t1 | p2 | 0 | 0 | 0 |
| t2 | p0 | 1 | 20 | 20 |
| t2 | p1 | 0 | 0 | 0 |
| t2 | p2 | 0 | 0 | 0 |
| t2 | p3 | 0 | 0 | 0 |
+-----+-----+-----+-----+-----+
7 rows in set (0.01 sec)
```

また、これらのパーティションを削除してから `SELECT` ステートメントを再実行することで、これらの行が各テーブルの最小番号のパーティションに格納されたことを示すこともできます：

```
mysql> ALTER TABLE t1 DROP PARTITION p0;
Query OK, 0 rows affected (0.16 sec)

mysql> ALTER TABLE t2 DROP PARTITION p0;
Query OK, 0 rows affected (0.16 sec)

mysql> SELECT * FROM t1;
Empty set (0.00 sec)

mysql> SELECT * FROM t2;
Empty set (0.00 sec)
```

(`ALTER TABLE ... DROP PARTITION` の詳細については、[セクション13.1.9「ALTER TABLE ステートメント」](#)を参照してください。)

SQL 関数を使用するパーティショニング式の場合も、`NULL` はこのように扱われます。次のような `CREATE TABLE` ステートメントを使用してテーブルを定義するとします。

```
CREATE TABLE tndate (
  id INT,
  dt DATE
)
PARTITION BY RANGE( YEAR(dt) ) (
  PARTITION p0 VALUES LESS THAN (1990),
  PARTITION p1 VALUES LESS THAN (2000),
  PARTITION p2 VALUES LESS THAN MAXVALUE
);
```

ほかの MySQL 関数と同様に、`YEAR(NULL)` は `NULL` を返します。 `dt` カラム値が `NULL` である行は、パーティショニング式がほかの値より小さい値に評価されたかのように扱われ、パーティション `p0` に挿入されます。

`LIST` パーティショニングでの `NULL` の扱い。 `LIST` によってパーティション化されたテーブルで `NULL` 値が許可されるのは、`NULL` が含まれている値リストを使用していずれかのパーティションが定義されている場合のみです。こ

れとは逆に、**LIST** によってパーティション化されたテーブルが、値リストで **NULL** を明示的に使用していない場合は、次の例のようにパーティショニング式で **NULL** 値に評価される行を拒否します。

```
mysql> CREATE TABLE ts1 (  
-> c1 INT,  
-> c2 VARCHAR(20)  
-> )  
-> PARTITION BY LIST(c1) (  
-> PARTITION p0 VALUES IN (0, 3, 6),  
-> PARTITION p1 VALUES IN (1, 4, 7),  
-> PARTITION p2 VALUES IN (2, 5, 8)  
-> );  
Query OK, 0 rows affected (0.01 sec)  
  
mysql> INSERT INTO ts1 VALUES (9, 'mothra');  
ERROR 1504 (HY000): Table has no partition for value 9  
  
mysql> INSERT INTO ts1 VALUES (NULL, 'mothra');  
ERROR 1504 (HY000): Table has no partition for value NULL
```

ts1 に挿入できるのは、**c1** 値が **0** 以上 **8** 以下の行のみです。 **NULL** は、数値 **9** と同様にこの範囲を外れます。 **NULL** が含まれる値リストを持つテーブル **ts2** および **ts3** は次のように作成できます。

```
mysql> CREATE TABLE ts2 (  
-> c1 INT,  
-> c2 VARCHAR(20)  
-> )  
-> PARTITION BY LIST(c1) (  
-> PARTITION p0 VALUES IN (0, 3, 6),  
-> PARTITION p1 VALUES IN (1, 4, 7),  
-> PARTITION p2 VALUES IN (2, 5, 8),  
-> PARTITION p3 VALUES IN (NULL)  
-> );  
Query OK, 0 rows affected (0.01 sec)  
  
mysql> CREATE TABLE ts3 (  
-> c1 INT,  
-> c2 VARCHAR(20)  
-> )  
-> PARTITION BY LIST(c1) (  
-> PARTITION p0 VALUES IN (0, 3, 6),  
-> PARTITION p1 VALUES IN (1, 4, 7, NULL),  
-> PARTITION p2 VALUES IN (2, 5, 8)  
-> );  
Query OK, 0 rows affected (0.01 sec)
```

パーティショニングの値リストを定義するときに、**NULL** をほかの値と同様に扱うことができます (そうすべきです)。たとえば、**VALUES IN (NULL)** および **VALUES IN (1, 4, 7, NULL)** は両方とも有効であり、**VALUES IN (1, NULL, 4, 7)**、**VALUES IN (NULL, 1, 4, 7)** などと同様です。カラム **c1** が **NULL** である行をテーブル **ts2** および **ts3** にそれぞれ挿入できます。

```
mysql> INSERT INTO ts2 VALUES (NULL, 'mothra');  
Query OK, 1 row affected (0.00 sec)  
  
mysql> INSERT INTO ts3 VALUES (NULL, 'mothra');  
Query OK, 1 row affected (0.00 sec)
```

INFORMATION_SCHEMA.PARTITIONS に対して適切なクエリを発行することによって、先ほど挿入した行を格納するためにどのパーティションが使用されたかを確認できます (前の例と同様に、パーティション化されたテーブルが **p** データベースに作成されたことを想定しています)。

```
mysql> SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH, DATA_LENGTH  
-> FROM INFORMATION_SCHEMA.PARTITIONS  
-> WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME LIKE 'ts_*';  
+-----+-----+-----+-----+-----+  
| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |  
+-----+-----+-----+-----+-----+  
| ts2 | p0 | 0 | 0 | 0 |  
| ts2 | p1 | 0 | 0 | 0 |  
| ts2 | p2 | 0 | 0 | 0 |  
| ts2 | p3 | 1 | 20 | 20 |  
| ts3 | p0 | 0 | 0 | 0 |
```

```

| ts3 | p1 | 1 | 20 | 20 |
| ts3 | p2 | 0 | 0 | 0 |
+-----+-----+
7 rows in set (0.01 sec)

```

このセクションですでに示したように、行を格納するためにどのパーティションが使用されたかについては、それらのパーティションを削除してから **SELECT** を実行することで確認できます。

HASH および **KEY** パーティショニングでの **NULL** の扱い。 **HASH** または **KEY** によってパーティション化されたテーブルの場合、**NULL** の扱いは少し異なります。これらの場合、**NULL** 値を返すパーティショニング式は、戻り値がゼロであったかのように扱われます。この動作については、**HASH** によってパーティション化されたテーブルを作成して該当する値が含まれるレコードを挿入することで、ファイルシステムにどのような影響があるかをチェックすることで確認できます。次のステートメントを使用して作成されたテーブル **th** (これも **p** データベース内) があるとします。

```

mysql> CREATE TABLE th (
-> c1 INT,
-> c2 VARCHAR(20)
-> )
-> PARTITION BY HASH(c1)
-> PARTITIONS 2;
Query OK, 0 rows affected (0.00 sec)

```

このテーブルに属するパーティションは、次のクエリーを使用して表示できます。

```

mysql> SELECT TABLE_NAME,PARTITION_NAME,TABLE_ROWS,AVG_ROW_LENGTH,DATA_LENGTH
> FROM INFORMATION_SCHEMA.PARTITIONS
> WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME = 'th';
+-----+-----+-----+-----+-----+
| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+-----+-----+-----+-----+-----+
| th | p0 | 0 | 0 | 0 |
| th | p1 | 0 | 0 | 0 |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)

```

各パーティションの **TABLE_ROWS** は 0 です。ここで次に示すように、**c1** カラム値が **NULL** および 0 である 2 つの行を **th** に挿入し、それらの行が挿入されたことを確認します。

```

mysql> INSERT INTO th VALUES (NULL, 'mothra'), (0, 'gigan');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM th;
+-----+-----+
| c1 | c2 |
+-----+-----+
| NULL | mothra |
+-----+-----+
| 0 | gigan |
+-----+-----+
2 rows in set (0.01 sec)

```

任意の整数 **N** について、**NULL MOD N** の値は常に **NULL** であることを思い出してください。 **HASH** または **KEY** によってパーティション化されたテーブルの場合、この結果は正しいパーティションを判別するために 0 として扱われます。 **INFORMATION_SCHEMA.PARTITIONS** テーブルを再度確認すると、両方の行がパーティション **p0** に挿入されたことがわかります。

```

mysql> SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH, DATA_LENGTH
> FROM INFORMATION_SCHEMA.PARTITIONS
> WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME = 'th';
+-----+-----+-----+-----+-----+
| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+-----+-----+-----+-----+-----+
| th | p0 | 2 | 20 | 20 |
| th | p1 | 0 | 0 | 0 |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)

```

テーブルの定義で **PARTITION BY HASH** のかわりに **PARTITION BY KEY** を使用して最後の例を繰り返すことで、**NULL** がこのタイプのパーティション化でも 0 のように処理されることを確認できます。

24.3 パーティション管理

SQL ステートメントを使用してパーティションテーブルを変更する方法は多数あります。`ALTER TABLE` ステートメントのパーティション化拡張を使用して、既存のパーティションを追加、削除、再定義、マージまたは分割できます。パーティション化されたテーブルおよびパーティションに関する情報を取得する方法もあります。以降のセクションでは次のトピックについて説明します。

- `RANGE` または `LIST` によってパーティション化されたテーブルのパーティション管理については、[セクション 24.3.1 「RANGE および LIST パーティションの管理」](#) を参照してください。
- `HASH` および `KEY` パーティションの管理については、[セクション 24.3.2 「HASH および KEY パーティションの管理」](#) を参照してください。
- パーティション化されたテーブルおよびパーティションに関する情報を取得するために MySQL 8.0 で提供されるメカニズムについては、[セクション 24.3.5 「パーティションに関する情報を取得する」](#) を参照してください。
- パーティションの保守操作の実行については、[セクション 24.3.4 「パーティションの保守」](#) を参照してください。

注記

パーティションテーブルのすべてのパーティションは、同じ数のサブパーティションを持つ必要があります。テーブルの作成後にサブパーティション化を変更することはできません。

テーブルパーティション化スキームを変更するには、パーティションテーブルを作成するために `CREATE TABLE` で使用される構文と同じ構文を持つ `partition_options` オプションとともに `ALTER TABLE` ステートメントを使用する必要があります。このオプション（および）は、常に `PARTITION BY` キーワードで始まります。次の `CREATE TABLE` ステートメントを使用して、レンジでパーティション化されたテーブルを作成したとします：

```
CREATE TABLE trb3 (id INT, name VARCHAR(50), purchased DATE)
PARTITION BY RANGE( YEAR(purchased) ) (
  PARTITION p0 VALUES LESS THAN (1990),
  PARTITION p1 VALUES LESS THAN (1995),
  PARTITION p2 VALUES LESS THAN (2000),
  PARTITION p3 VALUES LESS THAN (2005)
);
```

このテーブルをキーによるパーティション化でパーティション化し直して、キーをベースとする `id` カラム値を使用する 2 つのパーティションに分割するために、次のステートメントを使用できます。

```
ALTER TABLE trb3 PARTITION BY KEY(id) PARTITIONS 2;
```

これは、テーブルを削除してから `CREATE TABLE trb3 PARTITION BY KEY(id) PARTITIONS 2;` を使用して再作成する場合と同じ効果を、テーブルの構造に対して持ちます。

`ALTER TABLE ... ENGINE = ...` は、テーブルによって使用されるストレージエンジンのみを変更し、テーブルのパーティショニングスキームはそのままにします。このステートメントは、ターゲットストレージエンジンがパーティショニングサポートを提供している場合にのみ成功します。`ALTER TABLE ... REMOVE PARTITIONING` を使用して、テーブルのパーティション化を削除できます。[セクション 13.1.9 「ALTER TABLE ステートメント」](#) を参照してください。

重要

`ALTER TABLE` ステートメントに使用できるのは、単一の `PARTITION BY`、`ADD PARTITION`、`DROP PARTITION`、`REORGANIZE PARTITION`、または `COALESCE PARTITION` 句のみです。たとえば、パーティションを削除してテーブルの残りのパーティションを再編成する場合は、(`DROP PARTITION` を使用してから `REORGANIZE PARTITION` を使用して別の `ALTER TABLE` ステートメントを使用して) これを行う必要があります。

`ALTER TABLE ... TRUNCATE PARTITION` を使用して、選択した 1 つ以上のパーティションからすべての行を削除できます。

24.3.1 RANGE および LIST パーティションの管理

レンジパーティションとリストパーティションの追加および削除は同様の方法で処理されるため、このセクションでは両方のパーティション化の管理について説明します。ハッシュまたはキーによってパーティション化されたテーブルの管理については、[セクション24.3.2「HASH および KEY パーティションの管理」](#)を参照してください。

RANGE または **LIST** によってパーティション化されたテーブルからパーティションを削除するには、**DROP PARTITION** オプションを指定した **ALTER TABLE** ステートメントを使用します。レンジでパーティション化され、次の **CREATE TABLE** および **INSERT** ステートメントを使用して 10 個のレコードが移入されるテーブルを作成したとします:

```
mysql> CREATE TABLE tr (id INT, name VARCHAR(50), purchased DATE)
-> PARTITION BY RANGE( YEAR(purchased) ) (
-> PARTITION p0 VALUES LESS THAN (1990),
-> PARTITION p1 VALUES LESS THAN (1995),
-> PARTITION p2 VALUES LESS THAN (2000),
-> PARTITION p3 VALUES LESS THAN (2005),
-> PARTITION p4 VALUES LESS THAN (2010),
-> PARTITION p5 VALUES LESS THAN (2015)
-> );
Query OK, 0 rows affected (0.28 sec)

mysql> INSERT INTO tr VALUES
-> (1, 'desk organiser', '2003-10-15'),
-> (2, 'alarm clock', '1997-11-05'),
-> (3, 'chair', '2009-03-10'),
-> (4, 'bookcase', '1989-01-10'),
-> (5, 'exercise bike', '2014-05-09'),
-> (6, 'sofa', '1987-06-05'),
-> (7, 'espresso maker', '2011-11-22'),
-> (8, 'aquarium', '1992-08-04'),
-> (9, 'study desk', '2006-09-16'),
-> (10, 'lava lamp', '1998-12-25');
Query OK, 10 rows affected (0.05 sec)
Records: 10 Duplicates: 0 Warnings: 0
```

パーティション **p2** に挿入されているはずの項目を以下のように確認できます。

```
mysql> SELECT * FROM tr
-> WHERE purchased BETWEEN '1995-01-01' AND '1999-12-31';
+----+-----+-----+
| id | name   | purchased |
+----+-----+-----+
|  2 | alarm clock | 1997-11-05 |
| 10 | lava lamp  | 1998-12-25 |
+----+-----+-----+
2 rows in set (0.00 sec)
```

次に示すように、パーティション選択を使用してこの情報を取得することもできます:

```
mysql> SELECT * FROM tr PARTITION (p2);
+----+-----+-----+
| id | name   | purchased |
+----+-----+-----+
|  2 | alarm clock | 1997-11-05 |
| 10 | lava lamp  | 1998-12-25 |
+----+-----+-----+
2 rows in set (0.00 sec)
```

詳しくは[セクション24.5「パーティション選択」](#)をご覧ください。

p2 という名前のパーティションを削除するには、次のコマンドを実行します。

```
mysql> ALTER TABLE tr DROP PARTITION p2;
Query OK, 0 rows affected (0.03 sec)
```

注記

NDBCLUSTER ストレージエンジンは **ALTER TABLE ... DROP PARTITION** をサポートしていません。ただし、この章で説明されている **ALTER TABLE** へのほかのパーティショニング関連拡張はサポートしています。

パーティションを削除すると、そのパーティションに格納されていたすべてのデータも削除されることを覚えておくことは非常に重要です。前の `SELECT` クエリーを再実行することで、これが本当であることを確認できます。

```
mysql> SELECT * FROM tr WHERE purchased
-> BETWEEN '1995-01-01' AND '1999-12-31';
Empty set (0.00 sec)
```

注記

`DROP PARTITION` は、ネイティブパーティション化インプレース API でサポートされており、`ALGORITHM={COPY|INPLACE}` で使用できます。`ALGORITHM=INPLACE` を使用した `DROP PARTITION` は、パーティションに格納されているデータを削除し、パーティションを削除します。ただし、`ALGORITHM=COPY` または `old_alter_table=ON` を使用した `DROP PARTITION` では、パーティションテーブルが再構築され、削除されたパーティションから互換性のある `PARTITION ... VALUES` 定義を持つ別のパーティションへのデータの移動が試行されます。別のパーティションに移動できないデータは削除されます。

このため、テーブルに対して `ALTER TABLE ... DROP PARTITION` を実行するには、そのテーブルの `DROP` 権限が必要です。

テーブル定義およびそのパーティショニングスキームを保持したまま、すべてのパーティションからすべてのデータを削除する場合は、`TRUNCATE TABLE` ステートメントを使用してください。(セクション13.1.37「`TRUNCATE TABLE` ステートメント」を参照してください。)

データを失うことなくテーブルのパーティショニングを変更する場合は、代わりに `ALTER TABLE ... REORGANIZE PARTITION` を使用してください。`REORGANIZE PARTITION` については、後続の説明またはセクション13.1.9「`ALTER TABLE` ステートメント」を参照してください。

ここで `SHOW CREATE TABLE` ステートメントを実行すると、テーブルのパーティショニング構成がどのように変更されたかを確認できます。

```
mysql> SHOW CREATE TABLE tr\G
***** 1. row *****
Table: tr
Create Table: CREATE TABLE `tr` (
  `id` int(11) DEFAULT NULL,
  `name` varchar(50) DEFAULT NULL,
  `purchased` date DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
/*!50100 PARTITION BY RANGE ( YEAR(purchased))
(PARTITION p0 VALUES LESS THAN (1990) ENGINE = InnoDB,
PARTITION p1 VALUES LESS THAN (1995) ENGINE = InnoDB,
PARTITION p3 VALUES LESS THAN (2005) ENGINE = InnoDB,
PARTITION p4 VALUES LESS THAN (2010) ENGINE = InnoDB,
PARTITION p5 VALUES LESS THAN (2015) ENGINE = InnoDB) */
1 row in set (0.00 sec)
```

'1995-01-01'と'2004-12-31'の間の `purchased` カラム値を使用して、変更されたテーブルに新しい行を挿入すると、これらの行はパーティション `p3` に格納されます。このことを次のようにして確認できます。

```
mysql> INSERT INTO tr VALUES (11, 'pencil holder', '1995-07-12');
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SELECT * FROM tr WHERE purchased
-> BETWEEN '1995-01-01' AND '2004-12-31';
+----+-----+-----+
| id | name      | purchased |
+----+-----+-----+
|  1 | desk organiser | 2003-10-15 |
| 11 | pencil holder | 1995-07-12 |
+----+-----+-----+
2 rows in set (0.00 sec)
```

```
mysql> ALTER TABLE tr DROP PARTITION p3;
Query OK, 0 rows affected (0.03 sec)
```

```
mysql> SELECT * FROM tr WHERE purchased
-> BETWEEN '1995-01-01' AND '2004-12-31';
Empty set (0.00 sec)
```

`ALTER TABLE ... DROP PARTITION` の結果としてテーブルから削除された行数は、同等の `DELETE` クエリーとは異なり、サーバーから報告されません。

`LIST` パーティションを削除する場合は、`RANGE` パーティションの削除に使用するものとまったく同じ `ALTER TABLE ... DROP PARTITION` 構文を使用します。ただし、この操作が持つ影響について、このテーブルをあとで使用する際に重要な違いが 1 つあります。このテーブルには、削除したパーティションを定義する値リストに含まれていた値を持つ行を挿入できなくなります。(例については、[セクション24.2.2「LIST パーティショニング」](#)を参照してください)。

すでにパーティション化されたテーブルに新しい範囲またはリストパーティションを追加するには、`ALTER TABLE ... ADD PARTITION` ステートメントを使用します。`RANGE` によってパーティション化されたテーブルの場合は、これを使用して、既存のパーティションのリストの最後に新しい範囲を追加できます。次のように定義された、組織のメンバーシップデータが含まれるパーティション化されたテーブルがあるとします。

```
CREATE TABLE members (  
  id INT,  
  fname VARCHAR(25),  
  lname VARCHAR(25),  
  dob DATE  
)  
PARTITION BY RANGE( YEAR(dob) ) (  
  PARTITION p0 VALUES LESS THAN (1980),  
  PARTITION p1 VALUES LESS THAN (1990),  
  PARTITION p2 VALUES LESS THAN (2000)  
);
```

さらに、メンバーの最少年齢は 16 歳であるとします。カレンダーが 2015 年末に近づくにつれて、2000 年 (以降) に生まれたメンバーを入学させる準備が間もなくできていくことがわかりました。次のように `members` テーブルを変更することで、2000 年から 2010 年までに生まれた新しいメンバーを受け入れることができます。

```
ALTER TABLE members ADD PARTITION (PARTITION p3 VALUES LESS THAN (2010));
```

範囲によってパーティション化されたテーブルで `ADD PARTITION` を使用するときは、パーティションリストの上端にのみ新しいパーティションを追加できます。この方法で新しいパーティションを既存のパーティションの間または前に追加しようとすると、次のようにエラーになります。

```
mysql> ALTER TABLE members  
> ADD PARTITION (  
> PARTITION n VALUES LESS THAN (1970));  
ERROR 1463 (HY000): VALUES LESS THAN value must be strictly »  
increasing for each partition
```

この問題は、次のように最初のパーティションを 2 つに再編成し、それらの間の範囲を分割することで回避できます。

```
ALTER TABLE members  
  REORGANIZE PARTITION p0 INTO (  
    PARTITION n0 VALUES LESS THAN (1970),  
    PARTITION n1 VALUES LESS THAN (1980)  
  );
```

`SHOW CREATE TABLE` を使用することで、`ALTER TABLE` ステートメントによって意図した効果が得られたことを確認できます。

```
mysql> SHOW CREATE TABLE members\G  
***** 1. row *****  
Table: members  
Create Table: CREATE TABLE `members` (  
  `id` int(11) DEFAULT NULL,  
  `fname` varchar(25) DEFAULT NULL,  
  `lname` varchar(25) DEFAULT NULL,  
  `dob` date DEFAULT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=latin1  
/*!50100 PARTITION BY RANGE ( YEAR(dob) )  
(PARTITION n0 VALUES LESS THAN (1970) ENGINE = InnoDB,  
PARTITION n1 VALUES LESS THAN (1980) ENGINE = InnoDB,  
PARTITION p1 VALUES LESS THAN (1990) ENGINE = InnoDB,  
PARTITION p2 VALUES LESS THAN (2000) ENGINE = InnoDB,  
PARTITION p3 VALUES LESS THAN (2010) ENGINE = InnoDB) */
```



```
1 row in set (0.00 sec)
```

セクション13.1.9.1「ALTER TABLE パーティション操作」も参照してください。

ALTER TABLE ... ADD PARTITION を使用して、LIST によってパーティション化されたテーブルに新しいパーティションを追加することもできます。次の CREATE TABLE ステートメントを使用してテーブル `tt` が定義されています。

```
CREATE TABLE tt (
  id INT,
  data INT
)
PARTITION BY LIST(data) (
  PARTITION p0 VALUES IN (5, 10, 15),
  PARTITION p1 VALUES IN (6, 12, 18)
);
```

`data` カラム値が 7、14、および 21 である行を格納する新しいパーティションを次のように追加できます。

```
ALTER TABLE tt ADD PARTITION (PARTITION p2 VALUES IN (7, 14, 21));
```

既存のパーティションの値リストにすでに含まれている値を含む新しい LIST パーティションは追加できないことに注意してください。これを試行すると、次のエラーが発生します：

```
mysql> ALTER TABLE tt ADD PARTITION
> (PARTITION np VALUES IN (4, 8, 12));
ERROR 1465 (HY000): Multiple definition of same constant »
in list partitioning
```

`data` カラム値が 12 である行がパーティション `p1` にすでに割り当てられているため、値リストに 12 が含まれる新しいパーティションをテーブル `tt` に作成することはできません。これを実現するために、`p1` を削除し、`np` を追加してから、定義を変更した新しい `p1` を追加できます。ただし、すでに説明したように、これによって `p1` に格納されていたすべてのデータが失われるので、これが実際にやりたいことでないことが多いです。別の解決策になる可能性があるのが、CREATE TABLE ... SELECT ... を使用して、新しいパーティショニング付きでテーブルのコピーを作成し、データをそこにコピーしてから、古いテーブルを削除して新しいテーブルを名前変更することですが、これは大量のデータを扱うときに非常に時間がかかる可能性があります。高可用性が要求される状況では実行できない可能性もあります。

次のように単一 ALTER TABLE ... ADD PARTITION ステートメントで複数のパーティションを追加できます。

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(50) NOT NULL,
  lname VARCHAR(50) NOT NULL,
  hired DATE NOT NULL
)
PARTITION BY RANGE( YEAR(hired) ) (
  PARTITION p1 VALUES LESS THAN (1991),
  PARTITION p2 VALUES LESS THAN (1996),
  PARTITION p3 VALUES LESS THAN (2001),
  PARTITION p4 VALUES LESS THAN (2005)
);

ALTER TABLE employees ADD PARTITION (
  PARTITION p5 VALUES LESS THAN (2010),
  PARTITION p6 VALUES LESS THAN MAXVALUE
);
```

ありがたいことに、MySQL のパーティショニング実装は、データを失うことなくパーティショニングを再定義する方法を提供しています。まず、RANGE パーティショニングを使用するいくつかの簡単な例を見てみましょう。次のように定義された `members` テーブルを思い出してください。

```
mysql> SHOW CREATE TABLE members\G
***** 1. row *****
Table: members
Create Table: CREATE TABLE `members` (
  `id` int(11) DEFAULT NULL,
  `fname` varchar(25) DEFAULT NULL,
  `lname` varchar(25) DEFAULT NULL,
```

```
`dob` date DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
/*150100 PARTITION BY RANGE ( YEAR(dob))
(PARTITION n0 VALUES LESS THAN (1970) ENGINE = InnoDB,
PARTITION n1 VALUES LESS THAN (1980) ENGINE = InnoDB,
PARTITION p1 VALUES LESS THAN (1990) ENGINE = InnoDB,
PARTITION p2 VALUES LESS THAN (2000) ENGINE = InnoDB,
PARTITION p3 VALUES LESS THAN (2010) ENGINE = InnoDB) */
1 row in set (0.00 sec)
```

1960 年より前に生まれたメンバーを表すすべての行を別のパーティションに移動するとします。すでに説明したように、これは `ALTER TABLE ... ADD PARTITION` を使用して行うことはできません。ただし、`ALTER TABLE` への別のパーティション関連拡張を使用して、これを行うことができます。

```
ALTER TABLE members REORGANIZE PARTITION n0 INTO (
PARTITION s0 VALUES LESS THAN (1960),
PARTITION s1 VALUES LESS THAN (1970)
);
```

実際には、このコマンドはパーティション `p0` を 2 つの新しいパーティション `s0` および `s1` に分割します。さらに、`p0` に格納されていたデータを 2 つの `PARTITION ... VALUES ...` 句に示されているルールに従って新しいパーティションに移動する結果、`s0` には `YEAR(dob)` が 1960 より小さいレコードのみが含まれ、`s1` には `YEAR(dob)` が 1960 以上で 1970 より小さい行が含まれます。

`REORGANIZE PARTITION` 句を使用して、隣接するパーティションをマージすることもできます。次に示すように、前のステートメントの `members` テーブルへの影響を元に戻すことができます:

```
ALTER TABLE members REORGANIZE PARTITION s0,s1 INTO (
PARTITION p0 VALUES LESS THAN (1970)
);
```

`REORGANIZE PARTITION` を使用してパーティションを分割またはマージしてもデータは失われません。上記のステートメントを実行すると、MySQL はパーティション `s0` および `s1` に格納されていたすべてのレコードをパーティション `p0` に移動します。

`REORGANIZE PARTITION` の一般的な構文を次に示します。

```
ALTER TABLE tbl_name
REORGANIZE PARTITION partition_list
INTO (partition_definitions);
```

ここで、`tbl_name` はパーティションテーブルの名前で、`partition_list` は変更する既存のパーティションの名前のカンマ区切りリストです。`partition_definitions` は、新しいパーティション定義のカンマ区切りリストで、`CREATE TABLE` で使用される `partition_definitions` リストと同じルールに従います。`REORGANIZE PARTITION` を使用している場合、複数のパーティションを 1 つにマージしたり、1 つのパーティションを複数のパーティションに分割したりすることは制限されません。たとえば、次のように、`members` テーブルの 4 つのパーティションをすべて 2 つに再編成できます:

```
ALTER TABLE members REORGANIZE PARTITION p0,p1,p2,p3 INTO (
PARTITION m0 VALUES LESS THAN (1980),
PARTITION m1 VALUES LESS THAN (2000)
);
```

`LIST` によってパーティション化されたテーブルで `REORGANIZE PARTITION` を使用することもできます。リストによってパーティション化された `tt` テーブルに新しいパーティションを追加する操作が、既存のパーティションのいずれかの値リストにすでに存在する値が新しいパーティションに含まれていることが原因で失敗する問題に戻ります。これは、競合しない値のみが含まれるパーティションを追加してから、新しいパーティションと既存のものを再編成して既存のものに格納されていた値が新しいものに移動するようにすることで、対処できます。

```
ALTER TABLE tt ADD PARTITION (PARTITION np VALUES IN (4, 8));
ALTER TABLE tt REORGANIZE PARTITION p1,np INTO (
PARTITION p1 VALUES IN (6, 18),
PARTITION np VALUES in (4, 8, 12)
);
```

`RANGE` または `LIST` によってパーティション化されたテーブルをパーティション化し直すために `ALTER TABLE ... REORGANIZE PARTITION` を使用するときには注意すべき、いくつかの重要点を次に示します。

- 新しいパーティション化スキームの決定に使用される **PARTITION** オプションは、**CREATE TABLE** ステートメントで使用されるものと同じルールに従います。

新しい **RANGE** パーティション化スキームには、重複する範囲を含めることはできません。新しい **LIST** パーティション化スキームには、重複する値セットを含めることはできません。

- **partition_definitions** リストのパーティションの組み合わせは、**partition_list** に指定されたパーティションの組み合わせの範囲または値セット全体と同じであるべきです。

たとえば、パーティション **p1** と **p2** は、このセクションの例として使用される **members** テーブルの 1980 年から 1999 年までをカバーします。これら 2 つのパーティションの再編成では、全体的に同じ年の範囲をカバーする必要があります。

- **RANGE** によってパーティション化されたテーブルの場合、再編成できるのは隣接するパーティションのみで、レンジパーティションはスキップできません。

たとえば、1970 年より前の年は **p0** でカバーされ、1990 年から 1999 年までの年は **p2** でカバーされるため、**ALTER TABLE members REORGANIZE PARTITION p0,p2 INTO ...** で始まるステートメントを使用して **members** テーブルの例を再編成することはできませんでした。そのため、これらは隣接するパーティションではありません。(この場合、パーティション **p1** はスキップできません。)

- **REORGANIZE PARTITION** を使用して、テーブルで使用されるパーティション化のタイプを変更することはできません(たとえば、**RANGE** パーティションを **HASH** パーティションに変更したり、その逆を行うことはできません)。このステートメントを使用してパーティション化式またはカラムを変更することもできません。テーブルを削除および再作成せずにこれらのタスクのいずれかを実行するには、次に示すように **ALTER TABLE ... PARTITION BY ...** を使用できます:

```
ALTER TABLE members
PARTITION BY HASH( YEAR(dob) )
PARTITIONS 8;
```

24.3.2 HASH および KEY パーティションの管理

ハッシュまたはキーによってパーティション化されたテーブルは、パーティショニングセットアップで変更に関して互いによく似ていますが、範囲またはリストによってパーティション化されたテーブルとはいくつかの点で異なります。このため、このセクションではハッシュまたはキーによってパーティション化されたテーブルの変更についてのみ取り上げます。範囲またはリストによってパーティション化されたテーブルのパーティションを追加および削除することについては、[セクション24.3.1「RANGE および LIST パーティションの管理」](#)を参照してください。

HASH または **KEY** によってパーティション化されたテーブルから、**RANGE** または **LIST** によってパーティション化されたテーブルと同じ方法でパーティションを削除することはできません。ただし、**ALTER TABLE ... COALESCE PARTITION** を使用して **HASH** または **KEY** パーティションをマージできます。クライアントに関するデータを含む **clients** テーブルが、次に示すように作成された 12 個のパーティションに分割されているとします:

```
CREATE TABLE clients (
  id INT,
  fname VARCHAR(30),
  lname VARCHAR(30),
  signed DATE
)
PARTITION BY HASH( MONTH(signed) )
PARTITIONS 12;
```

パーティション数を 12 から 8 に減らすには、次の **ALTER TABLE** ステートメントを実行します:

```
mysql> ALTER TABLE clients COALESCE PARTITION 4;
Query OK, 0 rows affected (0.02 sec)
```

COALESCE は、**HASH**、**KEY**、**LINEAR HASH**、または **LINEAR KEY** によってパーティション化されたテーブルで同等に適切に動作します。次の例は前の例と似ていますが、テーブルが **LINEAR KEY** によってパーティション化されている点のみが異なります。

```
mysql> CREATE TABLE clients_lk (
->   id INT,
->   fname VARCHAR(30),
```

```
-> Inname VARCHAR(30),
-> signed DATE
-> )
-> PARTITION BY LINEAR KEY(signed)
-> PARTITIONS 12;
Query OK, 0 rows affected (0.03 sec)
```

```
mysql> ALTER TABLE clients_1k COALESCE PARTITION 4;
Query OK, 0 rows affected (0.06 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

`COALESCE PARTITION` のあとの数値は、残りにマージするパーティションの数です。つまり、テーブルから削除するパーティションの数です。

テーブルにあるよりも多くのパーティションを削除しようとすると、次のようなエラーが発生します:

```
mysql> ALTER TABLE clients COALESCE PARTITION 18;
ERROR 1478 (HY000): Cannot remove all partitions, use DROP TABLE instead
```

`clients` テーブルのパーティション数を 12 から 18 に増やすには、次に示すように `ALTER TABLE ... ADD PARTITION` を使用します:

```
ALTER TABLE clients ADD PARTITION PARTITIONS 6;
```

24.3.3 パーティションとサブパーティションをテーブルと交換する

MySQL 8.0 では、`ALTER TABLE pt EXCHANGE PARTITION p WITH TABLE nt` を使用して、テーブルパーティションまたはサブパーティションをテーブルと交換できます。ここで、`pt` はパーティション化されたテーブル、`p` はパーティション化されていないテーブル、`nt` と交換する `pt` のパーティションまたはサブパーティションです (次の記述が `true` である場合)。

1. テーブル `nt` 自体はパーティション化されていない。
2. テーブル `nt` は一時テーブルではない。
3. テーブル `pt` および `nt` の構造はそれ以外の点で同じである。
4. テーブル `nt` は外部キー参照を含まず、ほかのどのテーブルも `nt` を参照する外部キーを持たない。
5. `nt` 内に `p` のパーティション定義の境界の外に存在する行がない。この条件は、`WITHOUT VALIDATION` が使用されている場合は適用されません。
6. InnoDB テーブルの場合、両方のテーブルで同じ行形式が使用されます。InnoDB テーブルの行形式を確認するには、`INFORMATION_SCHEMA.INNODB_TABLES` をクエリーします。
7. `nt` には、`DATA DIRECTORY` オプションを使用するパーティションはありません。この制限は、MySQL 8.0.14 以降の InnoDB テーブルではなくなりました。

`ALTER TABLE` ステートメントに通常必要な `ALTER`、`INSERT`、および `CREATE` 権限に加えて、`ALTER TABLE ... EXCHANGE PARTITION` を実行するための `DROP` 権限が必要です。

`ALTER TABLE ... EXCHANGE PARTITION` の次の影響も考慮してください。

- `ALTER TABLE ... EXCHANGE PARTITION` を実行しても、パーティション化されたテーブルまたは交換されるテーブルに対するトリガーは呼び出されません。
- 交換されるテーブル内の `AUTO_INCREMENT` カラムがリセットされます。
- `IGNORE` キーワードは、`ALTER TABLE ... EXCHANGE PARTITION` と一緒に使用された場合、効果を持つません。

`ALTER TABLE ... EXCHANGE PARTITION` の構文を次に示します。ここで、`pt` はパーティションテーブル、`p` は交換されるパーティション (またはサブパーティション)、`nt` は `p` と交換される非パーティションテーブルです:

```
ALTER TABLE pt
EXCHANGE PARTITION p
```

```
WITH TABLE nt;
```

オプションで、**WITH VALIDATION** または **WITHOUT VALIDATION** を追加できます。**WITHOUT VALIDATION** が指定されている場合、**ALTER TABLE ... EXCHANGE PARTITION** 操作では、パーティションを非パーティションテーブルと交換するときに行ごとの検証は実行されず、データベース管理者は行がパーティション定義の境界内にあることを確認する責任を負うことができます。**WITH VALIDATION** がデフォルトです。

単一 **ALTER TABLE EXCHANGE PARTITION** ステートメントでは、1つのパーティションまたはサブパーティションのみを1つのパーティション化されていないテーブルのみと交換できます。複数のパーティションまたはサブパーティションを交換するには、複数の **ALTER TABLE EXCHANGE PARTITION** ステートメントを使用してください。**EXCHANGE PARTITION** は、ほかの **ALTER TABLE** オプションと組み合わせることはできません。パーティション化されたテーブルによって使用されるパーティショニングおよび (該当する場合) サブパーティショニングには、MySQL 8.0 でサポートされる任意のタイプを選択できます。

パーティションをパーティション化されていないテーブルと交換する

次の SQL ステートメントを使用して、パーティション化されたテーブル **e** が作成および移入されているとします。

```
CREATE TABLE e (  
  id INT NOT NULL,  
  fname VARCHAR(30),  
  lname VARCHAR(30)  
)  
PARTITION BY RANGE (id) (  
  PARTITION p0 VALUES LESS THAN (50),  
  PARTITION p1 VALUES LESS THAN (100),  
  PARTITION p2 VALUES LESS THAN (150),  
  PARTITION p3 VALUES LESS THAN (MAXVALUE)  
);  
  
INSERT INTO e VALUES  
  (1669, "Jim", "Smith"),  
  (337, "Mary", "Jones"),  
  (16, "Frank", "White"),  
  (2005, "Linda", "Black");
```

ここで、**e2** という名前の、**e** のパーティション化されていないコピーを作成します。これは、**mysql** クライアントを使用して次のように行うことができます。

```
mysql> CREATE TABLE e2 LIKE e;  
Query OK, 0 rows affected (0.04 sec)  
  
mysql> ALTER TABLE e2 REMOVE PARTITIONING;  
Query OK, 0 rows affected (0.07 sec)  
Records: 0 Duplicates: 0 Warnings: 0
```

テーブル **e** のどのパーティションに行が含まれるかは、次のように **INFORMATION_SCHEMA.PARTITIONS** テーブルを照会することで確認できます。

```
mysql> SELECT PARTITION_NAME, TABLE_ROWS  
FROM INFORMATION_SCHEMA.PARTITIONS  
WHERE TABLE_NAME = 'e';  
+-----+-----+  
| PARTITION_NAME | TABLE_ROWS |  
+-----+-----+  
| p0              | 1           |  
| p1              | 0           |  
| p2              | 0           |  
| p3              | 3           |  
+-----+-----+  
2 rows in set (0.00 sec)
```

注記

パーティション化された InnoDB テーブルの場合、**INFORMATION_SCHEMA.PARTITIONS** テーブルの **TABLE_ROWS** カラムに示される行数は、SQL 最適化で使用される見積もり値であり、常に正確とはかぎりません。

テーブル **e** のパーティション **p0** をテーブル **e2** と交換するには、次に示すように **ALTER TABLE** を使用できます：

```
mysql> ALTER TABLE e EXCHANGE PARTITION p0 WITH TABLE e2;  
Query OK, 0 rows affected (0.04 sec)
```

より正確に言うと、ここで発行したステートメントによって、パーティションで見つかる行がテーブルで見つかるものと交換されます。これがどのように行われたかは、前のように `INFORMATION_SCHEMA.PARTITIONS` テーブルを照会することで観察できます。パーティション `p0` で以前は見つかったテーブル行が存在しなくなっています。

```
mysql> SELECT PARTITION_NAME, TABLE_ROWS  
FROM INFORMATION_SCHEMA.PARTITIONS  
WHERE TABLE_NAME = 'e';
```

```
+-----+-----+  
| PARTITION_NAME | TABLE_ROWS |  
+-----+-----+  
| p0             | 0           |  
| p1             | 0           |  
| p2             | 0           |  
| p3             | 3           |  
+-----+-----+  
4 rows in set (0.00 sec)
```

テーブル `e2` を照会すると、「見つからない」行がそこで見つかります。

```
mysql> SELECT * FROM e2;
```

```
+-----+-----+  
| id | fname | lname |  
+-----+-----+  
| 16 | Frank | White |  
+-----+-----+  
1 row in set (0.00 sec)
```

パーティションと交換されるテーブルは、必ずしも空である必要はありません。これを示すために、まずテーブル `e` に新しい行を挿入し、50 未満の `id` カラム値を選択してこの行がパーティション `p0` に格納されていることを確認し、`PARTITIONS` テーブルをクエリーしてこれを後で検証します：

```
mysql> INSERT INTO e VALUES (41, "Michael", "Green");  
Query OK, 1 row affected (0.05 sec)
```

```
mysql> SELECT PARTITION_NAME, TABLE_ROWS  
FROM INFORMATION_SCHEMA.PARTITIONS  
WHERE TABLE_NAME = 'e';
```

```
+-----+-----+  
| PARTITION_NAME | TABLE_ROWS |  
+-----+-----+  
| p0             | 1           |  
| p1             | 0           |  
| p2             | 0           |  
| p3             | 3           |  
+-----+-----+  
4 rows in set (0.00 sec)
```

ここで、前と同じ `ALTER TABLE` ステートメントを使用して、ふたたびパーティション `p0` をテーブル `e2` と交換します。

```
mysql> ALTER TABLE e EXCHANGE PARTITION p0 WITH TABLE e2;  
Query OK, 0 rows affected (0.28 sec)
```

次のクエリーの出力は、`ALTER TABLE` ステートメントを発行する前に、パーティション `p0` に格納されていたテーブル行およびテーブル `e2` に格納されていたテーブル行の配置が切り替わったことを示しています。

```
mysql> SELECT * FROM e;
```

```
+-----+-----+  
| id | fname | lname |  
+-----+-----+  
| 16 | Frank | White |  
| 1669 | Jim | Smith |  
| 337 | Mary | Jones |  
| 2005 | Linda | Black |  
+-----+-----+  
4 rows in set (0.00 sec)
```

```
mysql> SELECT PARTITION_NAME, TABLE_ROWS  
FROM INFORMATION_SCHEMA.PARTITIONS
```



```
WHERE TABLE_NAME = 'e';
+-----+-----+
| PARTITION_NAME | TABLE_ROWS |
+-----+-----+
| p0              | 1           |
| p1              | 0           |
| p2              | 0           |
| p3              | 3           |
+-----+-----+
4 rows in set (0.00 sec)

mysql> SELECT * FROM e2;
+----+-----+-----+
| id | fname | lname |
+----+-----+-----+
| 41 | Michael | Green |
+----+-----+-----+
1 row in set (0.00 sec)
```

一致しない行

`ALTER TABLE ... EXCHANGE PARTITION` ステートメントを発行する前にパーティション化されていないテーブルで見つかる行は、それらがターゲットパーティションに格納されるために必要な条件を満たしている必要があり、そうでない場合はステートメントが失敗することを覚えておいてください。これがどのように発生するかを確認するために、まずテーブル `e` のパーティション `p0` のパーティション定義の境界外の行を、`e2` に挿入します。たとえば、`id` カラム値が大きすぎる行を挿入してから、テーブルをパーティションとふたたび交換してみてください。

```
mysql> INSERT INTO e2 VALUES (51, "Ellen", "McDonald");
Query OK, 1 row affected (0.08 sec)

mysql> ALTER TABLE e EXCHANGE PARTITION p0 WITH TABLE e2;
ERROR 1707 (HY000): Found row that does not match the partition
```

`WITHOUT VALIDATION` オプションのみがこの操作の成功を許可します:

```
mysql> ALTER TABLE e EXCHANGE PARTITION p0 WITH TABLE e2 WITHOUT VALIDATION;
Query OK, 0 rows affected (0.02 sec)
```

パーティション定義と一致しない行を含むテーブルとパーティションを交換する場合、データベース管理者は、`REPAIR TABLE` または `ALTER TABLE ... REPAIR PARTITION` を使用して実行できる不一致の行を修正する必要があります。

行ごとの検証なしでのパーティションの交換

多数の行を含むテーブルとパーティションを交換するときに時間のかかる検証を回避するために、`ALTER TABLE ... EXCHANGE PARTITION` ステートメントに `WITHOUT VALIDATION` を追加することで、行ごとの検証ステップをスキップできます。

次の例では、パーティションを非パーティションテーブルと交換する際の実行時間の違いを、検証ありおよび検証なしで比較します。パーティションテーブル (テーブル `e`) には、それぞれ 100 万行の 2 つのパーティションが含まれます。テーブル `e` の `p0` の行は削除され、`p0` は 100 万行のパーティション化されていないテーブルと交換されます。`WITH VALIDATION` 操作には 0.74 秒かかります。比較すると、`WITHOUT VALIDATION` 操作には 0.01 秒かかります。

```
# Create a partitioned table with 1 million rows in each partition

CREATE TABLE e (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30)
)
PARTITION BY RANGE (id) (
  PARTITION p0 VALUES LESS THAN (1000001),
  PARTITION p1 VALUES LESS THAN (2000001),
);

mysql> SELECT COUNT(*) FROM e;
| COUNT(*) |
+-----+
```

```
| 2000000 |
+-----+
1 row in set (0.27 sec)

# View the rows in each partition

SELECT PARTITION_NAME, TABLE_ROWS FROM INFORMATION_SCHEMA.PARTITIONS WHERE TABLE_NAME = 'e';
+-----+-----+
| PARTITION_NAME | TABLE_ROWS |
+-----+-----+
| p0             | 1000000     |
| p1             | 1000000     |
+-----+-----+
2 rows in set (0.00 sec)

# Create a nonpartitioned table of the same structure and populate it with 1 million rows

CREATE TABLE e2 (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30)
);

mysql> SELECT COUNT(*) FROM e2;
+-----+
| COUNT(*) |
+-----+
| 1000000  |
+-----+
1 row in set (0.24 sec)

# Create another nonpartitioned table of the same structure and populate it with 1 million rows

CREATE TABLE e3 (
  id INT NOT NULL,
  fname VARCHAR(30),
  lname VARCHAR(30)
);

mysql> SELECT COUNT(*) FROM e3;
+-----+
| COUNT(*) |
+-----+
| 1000000  |
+-----+
1 row in set (0.25 sec)

# Drop the rows from p0 of table e

mysql> DELETE FROM e WHERE id < 1000001;
Query OK, 1000000 rows affected (5.55 sec)

# Confirm that there are no rows in partition p0

mysql> SELECT PARTITION_NAME, TABLE_ROWS FROM INFORMATION_SCHEMA.PARTITIONS WHERE TABLE_NAME = 'e';
+-----+-----+
| PARTITION_NAME | TABLE_ROWS |
+-----+-----+
| p0             | 0           |
| p1             | 1000000     |
+-----+-----+
2 rows in set (0.00 sec)

# Exchange partition p0 of table e with the table e2 'WITH VALIDATION'

mysql> ALTER TABLE e EXCHANGE PARTITION p0 WITH TABLE e2 WITH VALIDATION;
Query OK, 0 rows affected (0.74 sec)

# Confirm that the partition was exchanged with table e2

mysql> SELECT PARTITION_NAME, TABLE_ROWS FROM INFORMATION_SCHEMA.PARTITIONS WHERE TABLE_NAME = 'e';
+-----+-----+
| PARTITION_NAME | TABLE_ROWS |
+-----+-----+
| p0             | 1000000     |
```

```
| p1      | 1000000 |
+-----+-----+
2 rows in set (0.00 sec)

# Once again, drop the rows from p0 of table e

mysql> DELETE FROM e WHERE id < 1000001;
Query OK, 1000000 rows affected (5.55 sec)

# Confirm that there are no rows in partition p0

mysql> SELECT PARTITION_NAME, TABLE_ROWS FROM INFORMATION_SCHEMA.PARTITIONS WHERE TABLE_NAME = 'e';
+-----+-----+
| PARTITION_NAME | TABLE_ROWS |
+-----+-----+
| p0              | 0           |
| p1              | 1000000    |
+-----+-----+
2 rows in set (0.00 sec)

# Exchange partition p0 of table e with the table e3 'WITHOUT VALIDATION'

mysql> ALTER TABLE e EXCHANGE PARTITION p0 WITH TABLE e3 WITHOUT VALIDATION;
Query OK, 0 rows affected (0.01 sec)

# Confirm that the partition was exchanged with table e3

mysql> SELECT PARTITION_NAME, TABLE_ROWS FROM INFORMATION_SCHEMA.PARTITIONS WHERE TABLE_NAME = 'e';
+-----+-----+
| PARTITION_NAME | TABLE_ROWS |
+-----+-----+
| p0              | 1000000    |
| p1              | 1000000    |
+-----+-----+
2 rows in set (0.00 sec)
```

パーティション定義と一致しない行を含むテーブルとパーティションを交換する場合、データベース管理者は、[REPAIR TABLE](#) または [ALTER TABLE ... REPAIR PARTITION](#) を使用して実行できる不一致の行を修正する必要があります。

サブパーティションをパーティション化されていないテーブルと交換する

[ALTER TABLE ... EXCHANGE PARTITION](#) ステートメントを使用して、サブパーティション化されたテーブルのサブパーティション ([セクション24.2.6「サブパーティショニング」](#)を参照してください) をパーティション化されていないテーブルと交換することもできます。次の例では、まず [RANGE](#) によってパーティション化され、[KEY](#) によってサブパーティション化されたテーブル [es](#) を作成し、テーブル [e](#) と同様にこのテーブルに移入してから、このテーブルの空のパーティション化されていないコピー [es2](#) を作成します。

```
mysql> CREATE TABLE es (
-> id INT NOT NULL,
-> fname VARCHAR(30),
-> lname VARCHAR(30)
-> )
-> PARTITION BY RANGE (id)
-> SUBPARTITION BY KEY (lname)
-> SUBPARTITIONS 2 (
-> PARTITION p0 VALUES LESS THAN (50),
-> PARTITION p1 VALUES LESS THAN (100),
-> PARTITION p2 VALUES LESS THAN (150),
-> PARTITION p3 VALUES LESS THAN (MAXVALUE)
-> );
Query OK, 0 rows affected (2.76 sec)

mysql> INSERT INTO es VALUES
-> (1669, "Jim", "Smith"),
-> (337, "Mary", "Jones"),
-> (16, "Frank", "White"),
-> (2005, "Linda", "Black");
Query OK, 4 rows affected (0.04 sec)
Records: 4 Duplicates: 0 Warnings: 0
```

```
mysql> CREATE TABLE es2 LIKE es;
Query OK, 0 rows affected (1.27 sec)
```

```
mysql> ALTER TABLE es2 REMOVE PARTITIONING;
Query OK, 0 rows affected (0.70 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

テーブル `es` の作成時にサブパーティションに明示的に名前を付けませんでした。次に示すように、`INFORMATION_SCHEMA` から `PARTITIONS` テーブルの `SUBPARTITION_NAME` カラムを含めて、生成された名前を取得できます:

```
mysql> SELECT PARTITION_NAME, SUBPARTITION_NAME, TABLE_ROWS
-> FROM INFORMATION_SCHEMA.PARTITIONS
-> WHERE TABLE_NAME = 'es';
+-----+-----+-----+
| PARTITION_NAME | SUBPARTITION_NAME | TABLE_ROWS |
+-----+-----+-----+
| p0             | p0sp0             | 1           |
| p0             | p0sp1             | 0           |
| p1             | p1sp0             | 0           |
| p1             | p1sp1             | 0           |
| p2             | p2sp0             | 0           |
| p2             | p2sp1             | 0           |
| p3             | p3sp0             | 3           |
| p3             | p3sp1             | 0           |
+-----+-----+-----+
8 rows in set (0.00 sec)
```

次の `ALTER TABLE` ステートメントは、テーブル `es` のサブパーティション `p3sp0` をパーティション化されていないテーブル `es2` と交換します:

```
mysql> ALTER TABLE es EXCHANGE PARTITION p3sp0 WITH TABLE es2;
Query OK, 0 rows affected (0.29 sec)
```

次のクエリを発行することで、それらの行が交換されたことを確認できます。

```
mysql> SELECT PARTITION_NAME, SUBPARTITION_NAME, TABLE_ROWS
-> FROM INFORMATION_SCHEMA.PARTITIONS
-> WHERE TABLE_NAME = 'es';
+-----+-----+-----+
| PARTITION_NAME | SUBPARTITION_NAME | TABLE_ROWS |
+-----+-----+-----+
| p0             | p0sp0             | 1           |
| p0             | p0sp1             | 0           |
| p1             | p1sp0             | 0           |
| p1             | p1sp1             | 0           |
| p2             | p2sp0             | 0           |
| p2             | p2sp1             | 0           |
| p3             | p3sp0             | 0           |
| p3             | p3sp1             | 0           |
+-----+-----+-----+
8 rows in set (0.00 sec)

mysql> SELECT * FROM es2;
+-----+-----+-----+
| id | fname | lname |
+-----+-----+-----+
| 1669 | Jim | Smith |
| 337 | Mary | Jones |
| 2005 | Linda | Black |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

テーブルがサブパーティション化されている場合、次に示すように、パーティション化されていないテーブルと交換できるのは、テーブルのパーティション全体ではなくサブパーティションのみです。

```
mysql> ALTER TABLE es EXCHANGE PARTITION p3 WITH TABLE es2;
ERROR 1704 (HY000): Subpartitioned table, use subpartition instead of partition
```

テーブル構造は厳密に比較されます。パーティションテーブルと非パーティションテーブルのカラムおよびインデックスの数、順序、名前およびタイプは正確に一致する必要があります。また、両方のテーブルが同じストレージエンジンを使用している必要があります。

```
mysql> CREATE TABLE es3 LIKE e;  
Query OK, 0 rows affected (1.31 sec)  
  
mysql> ALTER TABLE es3 REMOVE PARTITIONING;  
Query OK, 0 rows affected (0.53 sec)  
Records: 0 Duplicates: 0 Warnings: 0  
  
mysql> SHOW CREATE TABLE es3G  
***** 1. row *****  
      Table: es3  
Create Table: CREATE TABLE `es3` (  
  `id` int(11) NOT NULL,  
  `fname` varchar(30) DEFAULT NULL,  
  `lname` varchar(30) DEFAULT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4  
1 row in set (0.00 sec)  
  
mysql> ALTER TABLE es3 ENGINE = MyISAM;  
Query OK, 0 rows affected (0.15 sec)  
Records: 0 Duplicates: 0 Warnings: 0  
  
mysql> ALTER TABLE es EXCHANGE PARTITION p3sp0 WITH TABLE es3;  
ERROR 1497 (HY000): The mix of handlers in the partitions is not allowed in this version of MySQL
```

24.3.4 パーティションの保守

このような目的の SQL ステートメントを使用して、パーティションテーブルに対して多数のテーブルおよびパーティションメンテナンスタスクを実行できます。

パーティション化されたテーブルのテーブル保守は、パーティション化されたテーブルでサポートされる [CHECK TABLE](#)、[OPTIMIZE TABLE](#)、[ANALYZE TABLE](#)、および [REPAIR TABLE](#) ステートメントを使用して実現できます。

次のリストで説明しているように、[ALTER TABLE](#) へのいくつかの拡張を使用して、1 つ以上のパーティションに対してこのタイプの操作を直接実行できます。

- **パーティションの再構築。** パーティションを再構築します。これは、パーティションに格納されているすべてのレコードを削除してからそれらを再度挿入することと同じ効果があります。これはデフラグに役立つことがあります。

例:

```
ALTER TABLE t1 REBUILD PARTITION p0, p1;
```

- **パーティションの最適化。** パーティションから多数の行を削除した場合、または可変長行を持つ (つまり、[VARCHAR](#)、[BLOB](#)、または [TEXT](#) カラムを持つ) パーティション化されたテーブルに多くの変更を行なった場合は、[ALTER TABLE ... OPTIMIZE PARTITION](#) を使用して、未使用領域を解放したりパーティションデータファイルをデフラグしたりできます。

例:

```
ALTER TABLE t1 OPTIMIZE PARTITION p0, p1;
```

指定されたパーティションに [OPTIMIZE PARTITION](#) を使用することは、そのパーティションに [CHECK PARTITION](#)、[ANALYZE PARTITION](#)、および [REPAIR PARTITION](#) を実行することと同等です。

[InnoDB](#) を含む一部の MySQL ストレージエンジンは、パーティションごとの最適化をサポートしていません。この場合、[ALTER TABLE ... OPTIMIZE PARTITION](#) はテーブル全体を分析して再構築し、適切な警告が発行されます。(Bug #11751825、Bug #42822)。この問題を回避するには、代わりに [ALTER TABLE ... REBUILD PARTITION](#) および [ALTER TABLE ... ANALYZE PARTITION](#) を使用してください。

- **パーティションの分析。** これは、パーティションのキー分布を読み取って格納します。

例:

```
ALTER TABLE t1 ANALYZE PARTITION p3;
```

- **パーティションの修復。** これは、破損したパーティションを修復します。

例:

```
ALTER TABLE t1 REPAIR PARTITION p0,p1;
```

通常、パーティションに重複キーエラーが含まれている場合、**REPAIR PARTITION** は失敗します。このオプションを指定して **ALTER IGNORE TABLE** を使用できます。この場合、重複キーが存在するために移動できないすべての行がパーティションから削除されます (Bug #16900947)。

- パーティションのチェック。パーティション化されていないテーブルで **CHECK TABLE** を使用できるのと同様に、パーティションのエラーをチェックできます。

例:

```
ALTER TABLE trb3 CHECK PARTITION p1;
```

このステートメントは、テーブル **t1** のパーティション **p1** のデータまたはインデックスが破損しているかどうかを示します。その場合は、**ALTER TABLE ... REPAIR PARTITION** を使用してパーティションを修復してください。

通常、パーティションに重複キーエラーが含まれている場合、**CHECK PARTITION** は失敗します。このオプションを指定して **ALTER IGNORE TABLE** を使用すると、重複キー違反が見つかったパーティション内の各行の内容がステートメントによって返されます。テーブルのパーティション化式のカラムの値のみがレポートされます。(Bug #16900947)

上記のリストの各ステートメントでは、パーティション名のリストの代わりにキーワード **ALL** もサポートされます。**ALL** を使用すると、テーブル内のすべてのパーティションにステートメントが作用します。

ALTER TABLE ... TRUNCATE PARTITION を使用してパーティションを切り捨てることもできます。このステートメントは、**TRUNCATE TABLE** がテーブルからすべての行を削除するのと同様に、1つ以上のパーティションからすべての行を削除するために使用できます。

ALTER TABLE ... TRUNCATE PARTITION ALL はテーブル内のすべてのパーティションを切り捨てます。

24.3.5 パーティションに関する情報を取得する

このセクションでは、既存のパーティションに関する情報を取得する方法 (いくつかの方法が可能) について説明します。そのような情報を取得する方法には次のものが含まれます。

- **SHOW CREATE TABLE** ステートメントを使用して、パーティション化されたテーブルの作成に使用されたパーティショニング句を表示する。
- **SHOW TABLE STATUS** ステートメントを使用して、テーブルがパーティション化されているかどうかを判別する。
- **INFORMATION_SCHEMA.PARTITIONS** テーブルを照会する。
- ステートメント **EXPLAIN SELECT** を使用して、特定の **SELECT** で使用されているパーティションを確認します。

MySQL 8.0.16 から、パーティション化されたテーブルに対して挿入、削除、または更新が行われると、バイナリログには、パーティションに関する情報と、行イベントが発生したサブパーティション (存在する場合) に関する情報が記録されます。関連するテーブルが同じであっても、別のパーティションまたはサブパーティションで行われる変更に対して新しい行イベントが作成されます。したがって、トランザクションに3つのパーティションまたはサブパーティションが含まれる場合は、3つの行イベントが生成されます。更新イベントの場合、ビフォアイメージとアフターイメージの両方のパーティション情報が記録されます。**mysqlbinlog** を使用してバイナリログを表示するときに **-v** または **--verbose** オプションを指定すると、パーティション情報が表示されます。パーティション情報は、行ベースのロギングが使用中 (**binlog_format=ROW**) の場合にのみ記録されます。

この章のほかの場所でも説明しているように、**SHOW CREATE TABLE** の出力にはパーティション化されたテーブルの作成に使用された **PARTITION BY** 句が含まれます。例:

```
mysql> SHOW CREATE TABLE trb3G
***** 1. row *****
Table: trb3
Create Table: CREATE TABLE `trb3` (
  `id` int(11) DEFAULT NULL,
```



```
`name` varchar(50) DEFAULT NULL,  
`purchased` date DEFAULT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4  
/*150100 PARTITION BY RANGE (YEAR(purchased))  
(PARTITION p0 VALUES LESS THAN (1990) ENGINE = InnoDB,  
PARTITION p1 VALUES LESS THAN (1995) ENGINE = InnoDB,  
PARTITION p2 VALUES LESS THAN (2000) ENGINE = InnoDB,  
PARTITION p3 VALUES LESS THAN (2005) ENGINE = InnoDB) */  
0 row in set (0.00 sec)
```

パーティション化されたテーブルに対する `SHOW TABLE STATUS` の出力は、`Create_options` カラムに文字列 `partitioned` が含まれることを除いて、パーティション化されていないテーブルの場合と同じです。 `Engine` カラムには、テーブルのすべてのパーティションによって使用されるストレージエンジンの名前が含まれます。(このステートメントについての詳細は、[セクション13.7.7.38 「SHOW TABLE STATUS ステートメント」](#)を参照してください)。

パーティションに関する情報は、`PARTITIONS` テーブルを含む `INFORMATION_SCHEMA` から取得できます。 [セクション26.21 「INFORMATION_SCHEMA PARTITIONS テーブル」](#)を参照してください。

`EXPLAIN` を使用して、特定の `SELECT` クエリーに含まれるパーティションテーブルのパーティションを判別できます。 `EXPLAIN` 出力の `partitions` カラムには、クエリーによってレコードが照合されるパーティションがリストされます。

テーブル `trb1` が作成され、次のように移入されるとします:

```
CREATE TABLE trb1 (id INT, name VARCHAR(50), purchased DATE)  
PARTITION BY RANGE(id)  
(  
PARTITION p0 VALUES LESS THAN (3),  
PARTITION p1 VALUES LESS THAN (7),  
PARTITION p2 VALUES LESS THAN (9),  
PARTITION p3 VALUES LESS THAN (11)  
);  
  
INSERT INTO trb1 VALUES  
(1, 'desk organiser', '2003-10-15'),  
(2, 'CD player', '1993-11-05'),  
(3, 'TV set', '1996-03-10'),  
(4, 'bookcase', '1982-01-10'),  
(5, 'exercise bike', '2004-05-09'),  
(6, 'sofa', '1987-06-05'),  
(7, 'popcorn maker', '2001-11-22'),  
(8, 'aquarium', '1992-08-04'),  
(9, 'study desk', '1984-09-16'),  
(10, 'lava lamp', '1998-12-25');
```

`SELECT * FROM trb1;` などのクエリーでどのパーティションが使用されるかを次のように確認できます。

```
mysql> EXPLAIN SELECT * FROM trb1\G  
***** 1. row *****  
id: 1  
select_type: SIMPLE  
table: trb1  
partitions: p0,p1,p2,p3  
type: ALL  
possible_keys: NULL  
key: NULL  
key_len: NULL  
ref: NULL  
rows: 10  
Extra: Using filesort
```

この場合、4つのパーティションがすべて検索されます。ただし、次のようにパーティショニングキーを使用する制限条件をクエリーに追加すると、一致する値が含まれているパーティションのみがスキャンされることがわかります。

```
mysql> EXPLAIN SELECT * FROM trb1 WHERE id < 5\G  
***** 1. row *****  
id: 1  
select_type: SIMPLE  
table: trb1  
partitions: p0,p1
```

```

type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 10
Extra: Using where

```

EXPLAIN では、使用されるキーおよび使用可能なキーに関する情報も提供されます:

```

mysql> ALTER TABLE trb1 ADD PRIMARY KEY (id);
Query OK, 10 rows affected (0.03 sec)
Records: 10 Duplicates: 0 Warnings: 0

mysql> EXPLAIN SELECT * FROM trb1 WHERE id < 5\G
***** 1. row *****
   id: 1
select_type: SIMPLE
  table: trb1
 partitions: p0,p1
   type: range
possible_keys: PRIMARY
  key: PRIMARY
 key_len: 4
   ref: NULL
  rows: 7
Extra: Using where

```

EXPLAIN を使用して非パーティションテーブルに対するクエリーを調べる場合、エラーは生成されませんが、**partitions** カラムの値は常に **NULL** です。

EXPLAIN 出力の **rows** カラムには、テーブル内の行の合計数が表示されます。

[セクション13.8.2「EXPLAIN ステートメント」](#)も参照してください。

24.4 パーティションプルーニング

パーティションプルーニングと呼ばれる最適化は、「一致する値がない可能性があるパーティションをスキャンしません」と記述できる比較的単純な概念に基づいています。次のステートメントによってパーティションテーブル **t1** が作成されるとします:

```

CREATE TABLE t1 (
  fname VARCHAR(50) NOT NULL,
  lname VARCHAR(50) NOT NULL,
  region_code TINYINT UNSIGNED NOT NULL,
  dob DATE NOT NULL
)
PARTITION BY RANGE( region_code ) (
  PARTITION p0 VALUES LESS THAN (64),
  PARTITION p1 VALUES LESS THAN (128),
  PARTITION p2 VALUES LESS THAN (192),
  PARTITION p3 VALUES LESS THAN MAXVALUE
);

```

次のような **SELECT** ステートメントから結果を取得するとします:

```

SELECT fname, lname, region_code, dob
FROM t1
WHERE region_code > 125 AND region_code < 130;

```

返される必要がある行がパーティション **p0** または **p3** のいずれにもないことを簡単に確認できます。つまり、一致する行を検索するには、パーティション **p1** および **p2** でのみ検索が必要です。検索を制限することで、テーブル内のすべてのパーティションをスキャンするよりも、一致する行の検索にかかる時間と労力を大幅に削減できます。必要のないパーティションをこのように「省く」ことをプルーニングといいます。オプティマイザがこのクエリーの実行でパーティションプルーニングを使用できるときは、同じカラム定義およびデータが含まれているパーティション化されていないテーブルで同じクエリーを実行するよりも、速度が大幅に向上することがあります。

WHERE 条件を次の2つのケースのいずれかにまとめられるとき、オプティマイザはプルーニングを実行できます。

- **partition_column = constant**

- `partition_column IN (constant1, constant2, ..., constantN)`

最初のケースで、オプティマイザは指定された値についてパーティショニング式を単純に評価し、どのパーティションに値が含まれるかを判別して、このパーティションのみをスキャンします。多くの場合、この等号を別の算術比較 (<, >, <=, >=, および <> を含む) に置き換えることができます。WHERE 句で BETWEEN を使用するクエリも、パーティションプルーニングの利点を活用できます。このセクションの後続の例を参照してください。

2 番目のケースで、オプティマイザはリスト内の各値についてパーティショニング式を評価して、一致するパーティションのリストを作成してから、このパーティションリストのパーティションのみをスキャンします。

SELECT、DELETE および UPDATE ステートメントでは、パーティションプルーニングがサポートされます。INSERT ステートメントは、挿入された行ごとに 1 つのパーティションにのみアクセスします。これは、現在 EXPLAIN の出力には表示されていませんが、HASH または KEY によってパーティション化されたテーブルにも当てはまります。

短い範囲にもプルーニングを適用できます (オプティマイザが同等の値リストに変換できるもの)。たとえば、前の例では、WHERE 句を WHERE region_code IN (126, 127, 128, 129) に変換できます。次に、オプティマイザは、リスト内の最初の 2 つの値がパーティション p1 にあること、パーティション p2 内の残りの 2 つの値があること、および他のパーティションに関連する値が含まれていないことを判断できるため、一致する行を検索する必要はありません。

オプティマイザは、RANGE COLUMNS または LIST COLUMNS パーティション化を使用するテーブルの複数のカラムに対する前述の型の比較を含む WHERE 条件のプルーニングも実行できます。

このタイプの最適化は、パーティショニング式が同一性や範囲で構成されていてそれを同一性のセットにまとめられるとき、またはパーティショニング式が増減する関係を表すときに適用できます。パーティショニング式が YEAR() または TO_DAYS() 関数を使用するとき、DATE カラムまたは DATETIME カラムでパーティション化されるテーブルにプルーニングを適用することもできます。パーティション化式で TO_SECONDS() 関数を使用する場合は、このようなテーブルにプルーニングを適用することもできます。

DATE カラムでパーティション化されたテーブル t2 が、次に示すステートメントを使用して作成されるとします:

```
CREATE TABLE t2 (  
  fname VARCHAR(50) NOT NULL,  
  lname VARCHAR(50) NOT NULL,  
  region_code TINYINT UNSIGNED NOT NULL,  
  dob DATE NOT NULL  
)  
PARTITION BY RANGE( YEAR(dob) ) (  
  PARTITION d0 VALUES LESS THAN (1970),  
  PARTITION d1 VALUES LESS THAN (1975),  
  PARTITION d2 VALUES LESS THAN (1980),  
  PARTITION d3 VALUES LESS THAN (1985),  
  PARTITION d4 VALUES LESS THAN (1990),  
  PARTITION d5 VALUES LESS THAN (2000),  
  PARTITION d6 VALUES LESS THAN (2005),  
  PARTITION d7 VALUES LESS THAN MAXVALUE  
);
```

t2 を使用する次のステートメントでは、パーティションプルーニングを使用できます。

```
SELECT * FROM t2 WHERE dob = '1982-06-23';  
  
UPDATE t2 SET region_code = 8 WHERE dob BETWEEN '1991-02-15' AND '1997-04-25';  
  
DELETE FROM t2 WHERE dob >= '1984-06-21' AND dob <= '1999-06-21'
```

最後のステートメントの場合、オプティマイザは次のようにも動作できます。

1. 範囲の下限が含まれるパーティションを見つけます。

YEAR('1984-06-21') は値 1984 を返し、それはパーティション d3 に見つかります。

2. 範囲の上限が含まれるパーティションを見つけます。

YEAR('1999-06-21') は 1999 と評価され、それはパーティション d5 に見つかります。

3. これらの2つのパーティションおよびそれらの間にある可能性のあるパーティションのみをスキャンします。

この場合、これはパーティション `d3`、`d4`、および `d5` のみがスキャンされることを意味します。残りのパーティションは安全に無視できます (そして無視されます)。

重要

パーティション化されたテーブルに対するステートメントの `WHERE` 条件で参照される無効な `DATE` および `DATETIME` 値は、`NULL` として扱われます。これは、`SELECT * FROM partitioned_table WHERE date_column < '2008-12-00'` などのクエリーは値を返さないことを意味します (Bug #40972 を参照してください)。

ここまで、`RANGE` パーティショニングを使用する例のみを見てきましたが、プルーニングはほかのパーティショニングタイプにも適用できます。

次に示すテーブル `t3` のように、パーティショニング式が増加または減少している `LIST` によってパーティション化されたテーブルがあるとします。(この例では、簡単にするために `region_code` カラムが値 1 から 10 まで (両端を含む) に制限されると想定します)。

```
CREATE TABLE t3 (  
  fname VARCHAR(50) NOT NULL,  
  lname VARCHAR(50) NOT NULL,  
  region_code TINYINT UNSIGNED NOT NULL,  
  dob DATE NOT NULL  
)  
PARTITION BY LIST(region_code) (  
  PARTITION r0 VALUES IN (1, 3),  
  PARTITION r1 VALUES IN (2, 5, 8),  
  PARTITION r2 VALUES IN (4, 9),  
  PARTITION r3 VALUES IN (6, 7, 10)  
);
```

`SELECT * FROM t3 WHERE region_code BETWEEN 1 AND 3` などのステートメントの場合、オプティマイザはどのパーティションで値 1、2、および 3 が見つかるかを判別して (`r0` および `r1`)、残りのもの (`r2` および `r3`) をスキップします。

`HASH` または `[LINEAR] KEY` によってパーティション化されたテーブルの場合も、パーティショニング式で使用するカラムに対して `WHERE` 句が単純な `=` 関係を使用しているときは、パーティションプルーニングを適用できます。次のように作成されたテーブルがあるとします。

```
CREATE TABLE t4 (  
  fname VARCHAR(50) NOT NULL,  
  lname VARCHAR(50) NOT NULL,  
  region_code TINYINT UNSIGNED NOT NULL,  
  dob DATE NOT NULL  
)  
PARTITION BY KEY(region_code)  
PARTITIONS 8;
```

カラム値を定数と比較するステートメントはプルーニングできます。

```
UPDATE t4 WHERE region_code = 7;
```

プルーニングは短い範囲にも適用できます。オプティマイザがそのような条件を `IN` 関係に変換できるためです。たとえば、前に定義したものと同一テーブル `t4` を使用して、次のようなクエリーをプルーニングできます。

```
SELECT * FROM t4 WHERE region_code > 2 AND region_code < 6;
```

```
SELECT * FROM t4 WHERE region_code BETWEEN 3 AND 5;
```

どちらの場合も、`WHERE` 句はオプティマイザによって `WHERE region_code IN (3, 4, 5)` に変換されます。

重要

この最適化は、範囲サイズがパーティションの数より小さい場合にのみ使用されます。次のステートメントがあるとします。

```
DELETE FROM t4 WHERE region_code BETWEEN 4 AND 12;
```

WHERE 句の範囲は 9 個の値 (4、5、6、7、8、9、10、11、12) ですが、**t4** のパーティションは 8 個だけです。これはこの **DELETE** をブルーニングできないことを意味します。

HASH または **[LINEAR] KEY** によってパーティション化されたテーブルの場合、ブルーニングを使用できるのは整数カラムに対してのみです。たとえば、**dob** は **DATE** カラムであるため、次のステートメントにはブルーニングを使用できません。

```
SELECT * FROM t4 WHERE dob >= '2001-04-14' AND dob <= '2005-10-15';
```

ただし、このテーブルが年値を **INT** カラムに格納する場合は、**WHERE year_col >= 2001 AND year_col <= 2005** を持つクエリーをブルーニングできます。

MySQL Cluster で使用される **NDB** ストレージエンジンなど、自動パーティション分割を提供するストレージエンジンを使用するテーブルは、明示的にパーティション化されている場合にブルーニングできます。

24.5 パーティション選択

特定の **WHERE** 条件に一致する行のパーティションおよびサブパーティションの明示的な選択がサポートされています。パーティション選択は、特定のパーティションのみで一致がチェックされる点でパーティションブルーニングと似ていますが、2 つの重要な点で異なります。

1. チェックされるパーティションは、パーティションブルーニングと異なり (自動)、ステートメントの発行者が指定します。
2. パーティションブルーニングはクエリーのみ適用されますが、明示的なパーティション選択はクエリーおよびいくつかの DML ステートメントの両方でサポートされます。

明示的なパーティション選択をサポートする SQL ステートメントを次に一覧します。

- **SELECT**
- **DELETE**
- **INSERT**
- **REPLACE**
- **UPDATE**
- **LOAD DATA**。
- **LOAD XML**。

このセクションの残りの部分では、上記に一覧したステートメントに一般的に適用される明示的なパーティション選択について説明し、いくつかの例を示します。

明示的なパーティション選択は、**PARTITION** オプションを使用して実装されます。サポートされるすべてのステートメントについて、このオプションは次のような構文を使用します。

```
PARTITION (partition_names)
```

```
partition_names:  
partition_name, ...
```

このオプションは常に、パーティションが属するテーブルの名前の後ろに続けます。**partition_names** は、使用されるパーティションまたはサブパーティションのカンマ区切りのリストです。このリスト内の各名前は、指定されたテーブルの既存のパーティションまたはサブパーティションの名前である必要があります。パーティションまたはサブパーティションが見つからない場合、ステートメントは次のエラーで失敗します (**partition 'partition_name' doesn't exist**)。 **partition_names** に指定するパーティションまたはサブパーティションは、任意の順序でリストでき、重複していてもかまいません。

PARTITION オプションを使用すると、リストされたパーティションおよびサブパーティションのみで一致する行がチェックされます。このオプションを **SELECT** ステートメントで使用すると、指定したパーティションに属する行

を判別できます。次のようなステートメントを使用して作成および移入された、`employees` という名前のパーティション化されたテーブルがあるとします。

```
SET @@SQL_MODE = "";

CREATE TABLE employees (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  fname VARCHAR(25) NOT NULL,
  lname VARCHAR(25) NOT NULL,
  store_id INT NOT NULL,
  department_id INT NOT NULL
)
PARTITION BY RANGE(id) (
  PARTITION p0 VALUES LESS THAN (5),
  PARTITION p1 VALUES LESS THAN (10),
  PARTITION p2 VALUES LESS THAN (15),
  PARTITION p3 VALUES LESS THAN MAXVALUE
);

INSERT INTO employees VALUES
('Bob', 'Taylor', 3, 2), ('Frank', 'Williams', 1, 2),
('Ellen', 'Johnson', 3, 4), ('Jim', 'Smith', 2, 4),
('Mary', 'Jones', 1, 1), ('Linda', 'Black', 2, 3),
('Ed', 'Jones', 2, 1), ('June', 'Wilson', 3, 1),
('Andy', 'Smith', 1, 3), ('Lou', 'Waters', 2, 4),
('Jill', 'Stone', 1, 4), ('Roger', 'White', 3, 2),
('Howard', 'Andrews', 1, 2), ('Fred', 'Goldberg', 3, 3),
('Barbara', 'Brown', 2, 3), ('Alice', 'Rogers', 2, 2),
('Mark', 'Morgan', 3, 3), ('Karen', 'Cole', 3, 2);
```

どの行がパーティション `p1` に格納されているかは次のように確認できます。

```
mysql> SELECT * FROM employees PARTITION (p1);
+----+-----+-----+-----+-----+
| id | fname | lname | store_id | department_id |
+----+-----+-----+-----+-----+
| 5 | Mary | Jones | 1 | 1 |
| 6 | Linda | Black | 2 | 3 |
| 7 | Ed | Jones | 2 | 1 |
| 8 | June | Wilson | 3 | 1 |
| 9 | Andy | Smith | 1 | 3 |
+----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

この結果は、クエリー `SELECT * FROM employees WHERE id BETWEEN 5 AND 9` によって取得されるものと同じです。

複数のパーティションからの行を取得するには、それらの名前をカンマ区切りのリストとして指定します。たとえば、`SELECT * FROM employees PARTITION (p1, p2)` はパーティション `p1` および `p2` からのすべての行を返し、残りのパーティションからの行を除外します。

パーティション化されたテーブルに対する有効なクエリーは、`PARTITION` オプションを使用して、結果を1つ以上の目的のパーティションに制限するように書き直すことができます。`WHERE` 条件、`ORDER BY` オプション、`LIMIT` オプションなどを使用できます。集約関数を `HAVING` および `GROUP BY` オプション付きで使用することもできます。次の各クエリーは、前に定義した `employees` テーブルで実行するときに、有効な結果を生成します。

```
mysql> SELECT * FROM employees PARTITION (p0, p2)
-> WHERE lname LIKE 'S%';
+----+-----+-----+-----+-----+
| id | fname | lname | store_id | department_id |
+----+-----+-----+-----+-----+
| 4 | Jim | Smith | 2 | 4 |
| 11 | Jill | Stone | 1 | 4 |
+----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> SELECT id, CONCAT(fname, ' ', lname) AS name
-> FROM employees PARTITION (p0) ORDER BY lname;
+----+-----+
| id | name |
+----+-----+
| 3 | Ellen Johnson |
```



```

| 4 | Jim Smith |
| 1 | Bob Taylor |
| 2 | Frank Williams |
+-----+
4 rows in set (0.06 sec)

mysql> SELECT store_id, COUNT(department_id) AS c
-> FROM employees PARTITION (p1,p2,p3)
-> GROUP BY store_id HAVING c > 4;
+-----+
| c | store_id |
+-----+
| 5 | 2 |
| 5 | 3 |
+-----+
2 rows in set (0.00 sec)

```

パーティション選択を使用するステートメントは、サポートされている任意のパーティション化タイプを使用してテーブルで使用できます。テーブルが **[LINEAR] HASH** または **[LINEAR] KEY** パーティショニングを使用して作成されているけれども、パーティションの名前が指定されていない場合は、MySQL はパーティションに **p0**、**p1**、**p2**、...、**pN-1** という名前を自動的に付けます。ここで、**N** はパーティションの数です。明示的に名前が付けられていないサブパーティションの場合、MySQL は各パーティション **pX** 内のサブパーティションに **pXsp0**、**pXsp1**、**pXsp2**、...、**pXspM-1** という名前を自動的に割り当てます。ここで、**M** はサブパーティションの数です。このテーブルで **SELECT** (または明示的パーティション選択が許可されるほかの SQL ステートメント) を実行するときは、次のようにこれらの生成された名前を **PARTITION** オプションで使用できます。

```

mysql> CREATE TABLE employees_sub (
-> id INT NOT NULL AUTO_INCREMENT,
-> fname VARCHAR(25) NOT NULL,
-> lname VARCHAR(25) NOT NULL,
-> store_id INT NOT NULL,
-> department_id INT NOT NULL,
-> PRIMARY KEY pk (id, lname)
-> )
-> PARTITION BY RANGE(id)
-> SUBPARTITION BY KEY (lname)
-> SUBPARTITIONS 2 (
-> PARTITION p0 VALUES LESS THAN (5),
-> PARTITION p1 VALUES LESS THAN (10),
-> PARTITION p2 VALUES LESS THAN (15),
-> PARTITION p3 VALUES LESS THAN MAXVALUE
-> );
Query OK, 0 rows affected (1.14 sec)

mysql> INSERT INTO employees_sub # reuse data in employees table
-> SELECT * FROM employees;
Query OK, 18 rows affected (0.09 sec)
Records: 18 Duplicates: 0 Warnings: 0

mysql> SELECT id, CONCAT(fname, ' ', lname) AS name
-> FROM employees_sub PARTITION (p2sp1);
+-----+
| id | name |
+-----+
| 10 | Lou Waters |
| 14 | Fred Goldberg |
+-----+
2 rows in set (0.00 sec)

```

次のように **PARTITION** オプションを **INSERT ... SELECT** ステートメントの **SELECT** 部分に使用することもできます。

```

mysql> CREATE TABLE employees_copy LIKE employees;
Query OK, 0 rows affected (0.28 sec)

mysql> INSERT INTO employees_copy
-> SELECT * FROM employees PARTITION (p2);
Query OK, 5 rows affected (0.04 sec)
Records: 5 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM employees_copy;
+-----+

```

```
| id | fname | lname | store_id | department_id |
+----+-----+-----+-----+-----+
| 10 | Lou   | Waters | 2       | 4             |
| 11 | Jill  | Stone  | 1       | 4             |
| 12 | Roger | White  | 3       | 2             |
| 13 | Howard| Andrews| 1       | 2             |
| 14 | Fred  | Goldberg| 3       | 3             |
+----+-----+-----+-----+
5 rows in set (0.00 sec)
```

パーティション選択は結合と一緒に使用することもできます。次のステートメントを使用して2つのテーブルを作成して移入するとします。

```
CREATE TABLE stores (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  city VARCHAR(30) NOT NULL
)
PARTITION BY HASH(id)
PARTITIONS 2;

INSERT INTO stores VALUES
('Nambucca'), ('Uranga'),
('Bellingen'), ('Grafton');

CREATE TABLE departments (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(30) NOT NULL
)
PARTITION BY KEY(id)
PARTITIONS 2;

INSERT INTO departments VALUES
('Sales'), ('Customer Service'),
('Delivery'), ('Accounting');
```

任意またはすべてのテーブルからのパーティション (またはサブパーティション、あるいはその両方) を結合で明示的に選択できます (特定のテーブルからパーティションを選択するために使用される **PARTITION** オプションは、テーブルのエイリアスを含む他のすべてのオプションの直前のテーブルの名前に続きます。) たとえば、次のクエリーは都市 Nambucca および Bellingen (**stores** テーブルのパーティション **p0**) のいずれかの店舗の販売部門または配送部門 (**departments** テーブルのパーティション **p1**) で働いているすべての従業員の、名前、従業員 ID、部門、および都市を取得します。

```
mysql> SELECT
->   e.id AS 'Employee ID', CONCAT(e.fname, ' ', e.lname) AS Name,
->   s.city AS City, d.name AS department
-> FROM employees AS e
-> JOIN stores PARTITION (p1) AS s ON e.store_id=s.id
-> JOIN departments PARTITION (p0) AS d ON e.department_id=d.id
-> ORDER BY e.lname;
+----+-----+-----+-----+
| Employee ID | Name           | City       | department |
+----+-----+-----+-----+
| 14 | Fred Goldberg | Bellingen | Delivery   |
| 5  | Mary Jones   | Nambucca  | Sales      |
| 17 | Mark Morgan  | Bellingen | Delivery   |
| 9  | Andy Smith   | Nambucca  | Delivery   |
| 8  | June Wilson  | Bellingen | Sales      |
+----+-----+-----+-----+
5 rows in set (0.00 sec)
```

MySQL での結合の一般情報については、[セクション13.2.10.2「JOIN 句」](#)を参照してください。

DELETE ステートメントで **PARTITION** オプションを使用すると、オプションにリストされているパーティション (およびサブパーティション (ある場合)) でのみ削除される行がチェックされます。次のようにほかのパーティションは無視されます。

```
mysql> SELECT * FROM employees WHERE fname LIKE 'j%';
+----+-----+-----+-----+
| id | fname | lname | store_id | department_id |
+----+-----+-----+-----+
| 4  | Jim   | Smith | 2       | 4             |
| 8  | June  | Wilson| 3       | 1             |
```

```
| 11 | Jill | Stone | 1 | 4 |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> DELETE FROM employees PARTITION (p0, p1)
-> WHERE fname LIKE 'j%';
Query OK, 2 rows affected (0.09 sec)

mysql> SELECT * FROM employees WHERE fname LIKE 'j%';
+-----+-----+-----+-----+
| id | fname | lname | store_id | department_id |
+-----+-----+-----+-----+
| 11 | Jill | Stone | 1 | 4 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

WHERE 条件と一致するパーティション **p0** および **p1** 内の 2 つの行のみが削除されました。 **SELECT** を 2 回目に実行したときの結果から確認できるように、 **WHERE** 条件に一致する 1 行がテーブルに残っていますが、別のパーティション (**p2**) にあります。

明示的パーティション選択を使用する **UPDATE** ステートメントも同様に動作します。次のステートメントを実行することによって確認できるように、 **PARTITION** オプションによって参照されるパーティション内の行のみが、更新される行を判別するときに考慮されます。

```
mysql> UPDATE employees PARTITION (p0)
-> SET store_id = 2 WHERE fname = 'Jill';
Query OK, 0 rows affected (0.00 sec)
Rows matched: 0 Changed: 0 Warnings: 0

mysql> SELECT * FROM employees WHERE fname = 'Jill';
+-----+-----+-----+-----+
| id | fname | lname | store_id | department_id |
+-----+-----+-----+-----+
| 11 | Jill | Stone | 1 | 4 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> UPDATE employees PARTITION (p2)
-> SET store_id = 2 WHERE fname = 'Jill';
Query OK, 1 row affected (0.09 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT * FROM employees WHERE fname = 'Jill';
+-----+-----+-----+-----+
| id | fname | lname | store_id | department_id |
+-----+-----+-----+-----+
| 11 | Jill | Stone | 2 | 4 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

同様に、 **DELETE** 付きで **PARTITION** を使用すると、パーティションリストに指定されたパーティション内の行のみが削除をチェックされます。

行を挿入するステートメントの動作は、適切なパーティションが見つからないとステートメントが失敗する点が異なります。これは、次のように **INSERT** および **REPLACE** ステートメントの両方に当てはまります。

```
mysql> INSERT INTO employees PARTITION (p2) VALUES (20, 'Jan', 'Jones', 1, 3);
ERROR 1729 (HY000): Found a row not matching the given partition set
mysql> INSERT INTO employees PARTITION (p3) VALUES (20, 'Jan', 'Jones', 1, 3);
Query OK, 1 row affected (0.07 sec)

mysql> REPLACE INTO employees PARTITION (p0) VALUES (20, 'Jan', 'Jones', 3, 2);
ERROR 1729 (HY000): Found a row not matching the given partition set

mysql> REPLACE INTO employees PARTITION (p3) VALUES (20, 'Jan', 'Jones', 3, 2);
Query OK, 2 rows affected (0.09 sec)
```

InnoDB ストレージエンジンを使用するパーティション化されたテーブルに複数の行を書き込むステートメントの場合: **VALUES** に続くリスト内のいずれかの行を、 **partition_names** リストで指定されたいずれかのパーティションに書き込めない場合、ステートメント全体が失敗し、行は書き込まれません。これについては、次の例 (**employees** テーブルを再使用) の **INSERT** ステートメントで示されています。

```

mysql> ALTER TABLE employees
-> REORGANIZE PARTITION p3 INTO (
-> PARTITION p3 VALUES LESS THAN (20),
-> PARTITION p4 VALUES LESS THAN (25),
-> PARTITION p5 VALUES LESS THAN MAXVALUE
-> );
Query OK, 6 rows affected (2.09 sec)
Records: 6 Duplicates: 0 Warnings: 0

mysql> SHOW CREATE TABLE employees\G
***** 1. row *****
Table: employees
Create Table: CREATE TABLE `employees` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `fname` varchar(25) NOT NULL,
  `lname` varchar(25) NOT NULL,
  `store_id` int(11) NOT NULL,
  `department_id` int(11) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=27 DEFAULT CHARSET=utf8mb4
/*150100 PARTITION BY RANGE (id)
(PARTITION p0 VALUES LESS THAN (5) ENGINE = InnoDB,
PARTITION p1 VALUES LESS THAN (10) ENGINE = InnoDB,
PARTITION p2 VALUES LESS THAN (15) ENGINE = InnoDB,
PARTITION p3 VALUES LESS THAN (20) ENGINE = InnoDB,
PARTITION p4 VALUES LESS THAN (25) ENGINE = InnoDB,
PARTITION p5 VALUES LESS THAN MAXVALUE ENGINE = InnoDB) */
1 row in set (0.00 sec)

mysql> INSERT INTO employees PARTITION (p3, p4) VALUES
-> (24, 'Tim', 'Greene', 3, 1), (26, 'Linda', 'Mills', 2, 1);
ERROR 1729 (HY000): Found a row not matching the given partition set

mysql> INSERT INTO employees PARTITION (p3, p4, p5) VALUES
-> (24, 'Tim', 'Greene', 3, 1), (26, 'Linda', 'Mills', 2, 1);
Query OK, 2 rows affected (0.06 sec)
Records: 2 Duplicates: 0 Warnings: 0

```

前述のことは、複数の行を書き込む **INSERT** および **REPLACE** ステートメントの両方に当てはまります。

パーティションの選択は、**NDB** などの自動パーティション分割を提供するストレージエンジンを使用するテーブルでは無効になっています。

24.6 パーティショニングの制約と制限

このセクションでは、MySQL パーティショニングサポートでの現在の制約と制限について説明します。

禁止されている構造体。 次の構造体はパーティショニング式で許可されません。

- ストアドプロシージャ、ストアドファンクション、UDF、またはプラグイン。
- 宣言された変数またはユーザー変数。

パーティショニング式で許可される SQL 関数のリストについては、[セクション24.6.3「関数に関連するパーティショニング制限」](#)を参照してください。

算術および論理演算子。 算術演算子 **+**、**-**、および ***** の使用は、パーティショニング式で許可されます。ただし、結果は整数値または **NULL** である必要があります (この章のほかの場所で説明しているように、[\[LINEAR\] KEY](#) パーティショニングの場合を除きます。詳細は、[セクション24.2「パーティショニングタイプ」](#)を参照してください)。

DIV 演算子もサポートされています。/演算子は使用できません。

ビット演算子 **|**、**&**、**^**、**<<**、**>>**、および **~** はパーティショニング式では許可されません。

サーバー SQL モード。 ユーザー定義パーティショニングを使用するテーブルは、それらが作成された時点で有効だった SQL モードを保持しません。このマニュアルの他の場所で説明されているように ([セクション5.1.11「サーバー SQL モード」](#)を参照)、多くの MySQL 関数および演算子の結果は、サーバーの SQL モードに応じて変わる可能性があります。このため、パーティション化されたテーブルの作成後の任意の時点で SQL モードを変更すると、そ

のようなテーブルの動作が大きく変わることがあり、データの破損または損失が発生しやすくなることがあります。これらの理由により、パーティション化されたテーブルを作成したあとにサーバー SQL モードを決して変更しないことが強く推奨されています。

パーティション化されたテーブルを使用不可にするサーバー SQL モードでこのような変更を行う場合は、`NO_UNSIGNED_SUBTRACTION` モードが有効な場合にのみ正常に実行できる次の `CREATE TABLE` ステートメントを検討してください:

```
mysql> SELECT @@sql_mode;
+-----+
| @@sql_mode |
+-----+
|           |
+-----+
1 row in set (0.00 sec)

mysql> CREATE TABLE tu (c1 BIGINT UNSIGNED)
-> PARTITION BY RANGE(c1 - 10) (
-> PARTITION p0 VALUES LESS THAN (-5),
-> PARTITION p1 VALUES LESS THAN (0),
-> PARTITION p2 VALUES LESS THAN (5),
-> PARTITION p3 VALUES LESS THAN (10),
-> PARTITION p4 VALUES LESS THAN (MAXVALUE)
-> );
ERROR 1563 (HY000): Partition constant is out of partition function domain

mysql> SET sql_mode='NO_UNSIGNED_SUBTRACTION';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @@sql_mode;
+-----+
| @@sql_mode          |
+-----+
| NO_UNSIGNED_SUBTRACTION |
+-----+
1 row in set (0.00 sec)

mysql> CREATE TABLE tu (c1 BIGINT UNSIGNED)
-> PARTITION BY RANGE(c1 - 10) (
-> PARTITION p0 VALUES LESS THAN (-5),
-> PARTITION p1 VALUES LESS THAN (0),
-> PARTITION p2 VALUES LESS THAN (5),
-> PARTITION p3 VALUES LESS THAN (10),
-> PARTITION p4 VALUES LESS THAN (MAXVALUE)
-> );
Query OK, 0 rows affected (0.05 sec)
```

`tu` を作成したあとに `NO_UNSIGNED_SUBTRACTION` サーバー SQL モードを削除すると、このテーブルにアクセスできなくなる可能性があります。

```
mysql> SET sql_mode="";
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM tu;
ERROR 1563 (HY000): Partition constant is out of partition function domain
mysql> INSERT INTO tu VALUES (20);
ERROR 1563 (HY000): Partition constant is out of partition function domain
```

[セクション5.1.11「サーバー SQL モード」](#) も参照してください。

サーバー SQL モードは、パーティション化されたテーブルのレプリケーションにも影響します。ソースとレプリカで SQL モードを変えると、パーティション化式が異なる方法で評価される可能性があります。これにより、パーティション間でのデータの分散が特定のテーブルのソースとレプリカコピーで異なる可能性があり、ソースで成功したパーティションテーブルへの挿入がレプリカで失敗する可能性もあります。最良の結果を得るには、ソースとレプリカで常に同じサーバー SQL モードを使用する必要があります。

パフォーマンス考慮事項. パーティション化操作がパフォーマンスに与える影響の一部を次に示します:

- **ファイルシステム操作.** パーティション化および再パーティション化操作 (`PARTITION BY ...`、`REORGANIZE PARTITION`、`REMOVE PARTITIONING` を使用した `ALTER TABLE` など) は、実装のためのファイルシステム操作

に依存します。これは、これらの操作の速度が、ファイルシステムのタイプと特性、ディスク速度、スワップ領域、オペレーティングシステムによるファイル処理効率、ファイル処理に関連する MySQL サーバーのオプションと変数などの要因に影響されることを意味します。特に、`large_files_support` が有効になっていて、`open_files_limit` が適切に設定されていることを確認してください。`innodb_file_per_table` を有効にすると、InnoDB テーブルを含むパーティション化および再パーティション化操作がより効率的になります。

[パーティションの最大数](#)も参照してください。

- テーブルロック。通常、テーブルに対してパーティション化操作を実行するプロセスは、テーブルに対する書き込みロックを取得します。そのようなテーブルからの読み取りは比較的影響を受けません。保留中の `INSERT` および `UPDATE` 操作は、パーティショニング操作が完了するとすぐに実行されます。この制限に対する InnoDB 固有の例外については、[パーティション化操作](#) を参照してください。
- インデックス、パーティションプルーニング。パーティション化されていないテーブルと同様に、インデックスを適切に使用することで、パーティション化されたテーブルに対する照会速度が大幅に向上することがあります。また、パーティション化されたテーブルおよびこれらのテーブルに対するクエリをパーティションプルーニングの利点を活用するように設計することで、パフォーマンスが劇的に向上することがあります。詳細は、[セクション 24.4「パーティションプルーニング」](#) を参照してください。

インデックス条件プッシュダウンは、パーティションテーブルでサポートされています。[セクション 8.2.1.6「インデックスコンディションプッシュダウンの最適化」](#) を参照してください。
- `LOAD DATA` のパフォーマンス。MySQL 8.0 では、`LOAD DATA` はパフォーマンスを向上させるためにバッファリングを使用します。これを実現するために、バッファがパーティションごとに 130K バイトメモリーを使用することを認識してください。

パーティションの最大数。

NDB ストレージエンジンを使用しない特定のテーブルで可能なパーティションの最大数は 8192 です。この数にはサブパーティションが含まれます。

NDB ストレージエンジンを使用するテーブルのユーザー定義パーティションの最大数は、使用されている NDB Cluster ソフトウェアのバージョン、データノードの数、およびその他の要因に応じて決定されます。詳細は、[NDB とユーザー定義のパーティション化](#) を参照してください。

多数のパーティション (ただし、最大数より少ない) を持つテーブルを作成するときに、`Got error ... from storage engine: Out of resources when opening file` などのエラーメッセージが表示される場合は、`open_files_limit` システム変数の値を増やすことによってこの問題に対処できることがあります。ただし、これはオペレーティングシステムによって異なるため、すべてのプラットフォームで可能または推奨されるとはかぎりません。詳細は、[セクション B.3.2.16「ファイルが見つからず同様のエラーが発生しました」](#) を参照してください。場合によっては、多数の (数百の) パーティションを使用することがほかの問題のために推奨されないこともあり、より多くのパーティションを使用することが自動的に良い結果となるとはかぎりません。

[ファイルシステム操作](#)も参照してください。

パーティション化された InnoDB テーブルで外部キーがサポートされない。

InnoDB ストレージエンジンを使用するパーティション化されたテーブルでは、外部キーはサポートされません。これは具体的には、次の 2 つの記述が `true` であることを意味します。

1. ユーザー定義パーティショニングを使用する InnoDB テーブルの定義には、外部キー参照を含めることはできません。定義に外部キー参照が含まれる InnoDB テーブルはパーティション化できません。
2. InnoDB テーブル定義に、ユーザーパーティション化されたテーブルへの外部キー参照を含めることはできません。ユーザー定義パーティショニングを持つ InnoDB テーブルに、外部キーによって参照されるカラムを含めることはできません。

上記の制約の範囲には、InnoDB ストレージエンジンを使用するすべてのテーブルが含まれます。結果のテーブルがこれらの制約に違反する `CREATE TABLE` および `ALTER TABLE` ステートメントは許可されません。

`ALTER TABLE ... ORDER BY`。パーティション化されたテーブルに `ALTER TABLE ... ORDER BY column` ステートメントを実行すると、各パーティション内でのみ行が並べ替えられます。

主キーを変更することによる `REPLACE` ステートメントへの影響。テーブルの主キーを変更することが望ましい場合があります ([セクション 24.6.1「パーティショニングキー、主キー、および一意キー」](#) を参照してください)。

REPLACE ステートメントを使用するアプリケーションでこれを行うと、これらのステートメントの結果が大きく変わることがあることを認識してください。詳細および例については、[セクション13.2.9「REPLACE ステートメント」](#)を参照してください。

FULLTEXT インデックス。

パーティションテーブルでは、**FULLTEXT** インデックスまたは検索はサポートされていません。

空間カラム。 **POINT**、**GEOMETRY** などの空間データ型を持つカラムは、パーティション化されたテーブルで使用できません。

一時テーブル。

一時テーブルはパーティション化できません

ログテーブル。 ログテーブルをパーティション化することはできません。そのようなテーブルに **ALTER TABLE ... PARTITION BY ...** ステートメントを実行すると、エラーで失敗します。

パーティショニングキーのデータ型。

パーティショニングキーは、整数カラム、または整数に解決される式である必要があります。 **ENUM** カラムを使用する式は使用できません。カラムまたは式の値は **NULL** でもかまいません。 [セクション24.2.7「MySQL パーティショニングによる NULL の扱い」](#) を参照してください。

この制約には 2 つの例外があります。

1. **[LINEAR] KEY** でパーティション化する場合、**TEXT** または **BLOB** 以外の有効な MySQL データ型のカラムをパーティション化キーとして使用できます。これは、内部キーハッシュ関数がこれらの型から正しいデータ型を生成するためです。たとえば、次の 2 つの **CREATE TABLE** ステートメントは有効です。

```
CREATE TABLE tkc (c1 CHAR)
PARTITION BY KEY(c1)
PARTITIONS 4;

CREATE TABLE tke
(c1 ENUM('red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet'))
PARTITION BY LINEAR KEY(c1)
PARTITIONS 6;
```

2. **RANGE COLUMNS** または **LIST COLUMNS** によってパーティショニングするときは、文字列、**DATE**、および **DATETIME** カラムを使用できます。たとえば、次の各 **CREATE TABLE** ステートメントは有効です。

```
CREATE TABLE rc (c1 INT, c2 DATE)
PARTITION BY RANGE COLUMNS(c2) (
PARTITION p0 VALUES LESS THAN('1990-01-01'),
PARTITION p1 VALUES LESS THAN('1995-01-01'),
PARTITION p2 VALUES LESS THAN('2000-01-01'),
PARTITION p3 VALUES LESS THAN('2005-01-01'),
PARTITION p4 VALUES LESS THAN(MAXVALUE)
);

CREATE TABLE lc (c1 INT, c2 CHAR(1))
PARTITION BY LIST COLUMNS(c2) (
PARTITION p0 VALUES IN('a', 'd', 'g', 'j', 'm', 'p', 's', 'v', 'y'),
PARTITION p1 VALUES IN('b', 'e', 'h', 'k', 'n', 'q', 't', 'w', 'z'),
PARTITION p2 VALUES IN('c', 'f', 'i', 'l', 'o', 'r', 'u', 'x', NULL)
);
```

上記の例外は、**BLOB** または **TEXT** カラム型には該当しません。

サブクエリー。

パーティショニングキーはサブクエリーにできません (そのサブクエリーが整数値または **NULL** に解決される場合でも)。

カラムインデックス接頭辞はキーパーティション化ではサポートされていません。キーでパーティション化されたテーブルを作成する場合、カラム接頭辞を使用するパーティション化キー内のカラムは、テーブルパーティション化関数では使用されません。3 つの **VARCHAR** カラムがあり、主キーが 3 つのカラムすべてを使用し、それらのうちの 2 つに接頭辞を指定する次の **CREATE TABLE** ステートメントについて考えてみます:

```
CREATE TABLE t1 (
```

```
a VARCHAR(10000),
b VARCHAR(25),
c VARCHAR(10),
PRIMARY KEY (a(10), b, c(2))
) PARTITION BY KEY() PARTITIONS 2;
```

このステートメントは受け入れられますが、結果のテーブルは、パーティション化キーの接頭辞 (カラム b) を含まない主キーカラムのみを使用して、次のステートメントを発行したかのように実際に作成されます:

```
CREATE TABLE t1 (
  a VARCHAR(10000),
  b VARCHAR(25),
  c VARCHAR(10),
  PRIMARY KEY (a(10), b, c(2))
) PARTITION BY KEY(b) PARTITIONS 2;
```

MySQL 8.0.21 より前は、次に示すように、パーティション化キーに指定されたすべてのカラムで接頭辞が使用された場合を除き、警告は発行されず、それ以外の場合は、ステートメントは失敗しましたが、誤解を招くエラーメッセージが表示されます:

```
mysql> CREATE TABLE t2 (
->  a VARCHAR(10000),
->  b VARCHAR(25),
->  c VARCHAR(10),
->  PRIMARY KEY (a(10), b(5), c(2))
-> ) PARTITION BY KEY() PARTITIONS 2;
ERROR 1503 (HY000): A PRIMARY KEY must include all columns in the
table's partitioning function
```

これは、ALTER TABLE の実行時またはこのようなテーブルのアップグレード時にも発生します。

この許容動作は、MySQL 8.0.21 では非推奨です (将来のバージョンの MySQL では削除される可能性があります)。MySQL 8.0.21 以降では、パーティション化キーに接頭辞を持つカラムを使用すると、次に示すように、そのようなカラムごとに警告が表示されます:

```
mysql> CREATE TABLE t1 (
->  a VARCHAR(10000),
->  b VARCHAR(25),
->  c VARCHAR(10),
->  PRIMARY KEY (a(10), b, c(2))
-> ) PARTITION BY KEY() PARTITIONS 2;
Query OK, 0 rows affected, 2 warnings (1.25 sec)

mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Warning
Code: 1681
Message: Column 'test.t1.a' having prefix key part 'a(10)' is ignored by the
partitioning function. Use of prefixed columns in the PARTITION BY KEY() clause
is deprecated and will be removed in a future release.
***** 2. row *****
Level: Warning
Code: 1681
Message: Column 'test.t1.c' having prefix key part 'c(2)' is ignored by the
partitioning function. Use of prefixed columns in the PARTITION BY KEY() clause
is deprecated and will be removed in a future release.
2 rows in set (0.00 sec)
```

これには、空の PARTITION BY KEY() 句を使用して、パーティション化関数で使われるカラムがテーブルの主キーのカラムとして暗黙的に定義される場合が含まれます。

MySQL 8.0.21 以降では、パーティション化キーに指定されたすべてのカラムに接頭辞が使用されている場合、使用される CREATE TABLE ステートメントは失敗し、問題を正しく識別するエラーメッセージが表示されます:

```
mysql> CREATE TABLE t1 (
->  a VARCHAR(10000),
->  b VARCHAR(25),
->  c VARCHAR(10),
->  PRIMARY KEY (a(10), b(5), c(2))
-> ) PARTITION BY KEY() PARTITIONS 2;
ERROR 1503 (HY000): A PRIMARY KEY must include all columns in the table's
```

partitioning function (prefixed columns are not considered).

キーによるテーブルのパーティション化の一般情報は、[セクション24.2.5「KEY パーティショニング」](#)を参照してください。

サブパーティションに関する問題.

サブパーティションは **HASH** または **KEY** パーティショニングを使用する必要があります。サブパーティション化できるのは **RANGE** および **LIST** パーティションのみです。**HASH** および **KEY** パーティションはサブパーティション化できません。

SUBPARTITION BY KEY では、**PARTITION BY KEY** の場合とは異なり、サブパーティション化カラムを明示的に指定する必要があります。ここでは省略できます (この場合、テーブルの主キーカラムがデフォルトで使用されます)。次のステートメントによって作成されたテーブルがあるとします。

```
CREATE TABLE ts (  
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,  
  name VARCHAR(30)  
);
```

次のようなステートメントを使用することで、**KEY** によってパーティション化された、同じカラムを持つテーブルを作成できます。

```
CREATE TABLE ts (  
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,  
  name VARCHAR(30)  
)  
PARTITION BY KEY()  
PARTITIONS 4;
```

前のステートメントは、次のように記述されているかのように扱われます (テーブルの主キーカラムがパーティショニングカラムとして使用されます)。

```
CREATE TABLE ts (  
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,  
  name VARCHAR(30)  
)  
PARTITION BY KEY(id)  
PARTITIONS 4;
```

ただし、次のステートメントは、デフォルトカラムをサブパーティショニングカラムとして使用するサブパーティション化されたテーブルを作成しようとするため失敗します。このステートメントが成功するには次のようにカラムを指定する必要があります。

```
mysql> CREATE TABLE ts (  
-> id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,  
-> name VARCHAR(30)  
-> )  
-> PARTITION BY RANGE(id)  
-> SUBPARTITION BY KEY()  
-> SUBPARTITIONS 4  
-> (  
-> PARTITION p0 VALUES LESS THAN (100),  
-> PARTITION p1 VALUES LESS THAN (MAXVALUE)  
-> );  
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that  
corresponds to your MySQL server version for the right syntax to use near ')'  
  
mysql> CREATE TABLE ts (  
-> id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,  
-> name VARCHAR(30)  
-> )  
-> PARTITION BY RANGE(id)  
-> SUBPARTITION BY KEY(id)  
-> SUBPARTITIONS 4  
-> (  
-> PARTITION p0 VALUES LESS THAN (100),  
-> PARTITION p1 VALUES LESS THAN (MAXVALUE)  
-> );  
Query OK, 0 rows affected (0.07 sec)
```

これは既知の問題です (Bug #51470 を参照してください)。

DATA DIRECTORY および INDEX DIRECTORY オプション。 テーブルレベル [DATA DIRECTORY](#) および [INDEX DIRECTORY](#) オプションは無視されます (Bug #32091 を参照してください)。これらのオプションは、InnoDB テーブルの個々のパーティションまたはサブパーティションに使用できます。MySQL 8.0.21 では、[DATA DIRECTORY](#) 句で指定されたディレクトリは InnoDB で認識されている必要があります。詳細は、[DATA DIRECTORY 句の使用](#) を参照してください。

パーティション化されたテーブルを修復および再構築する。 ステートメント [CHECK TABLE](#)、[OPTIMIZE TABLE](#)、[ANALYZE TABLE](#)、および [REPAIR TABLE](#) がパーティション化されたテーブルでサポートされます。

また、[ALTER TABLE ... REBUILD PARTITION](#) を使用することで、パーティション化されたテーブルの 1 つ以上のパーティションを再構築できます。[ALTER TABLE ... REORGANIZE PARTITION](#) でもパーティションが再構築されます。これら 2 つのステートメントの詳細については、[セクション 13.1.9 「ALTER TABLE ステートメント」](#) を参照してください。

サブパーティションでは、[ANALYZE](#)、[CHECK](#)、[OPTIMIZE](#)、[REPAIR](#) および [TRUNCATE](#) 操作がサポートされています。[セクション 13.1.9.1 「ALTER TABLE パーティション操作」](#) を参照してください。

パーティションおよびサブパーティションのファイル名デリミタ。 テーブルパーティションおよびサブパーティションファイルの名前には、[#P#](#)や[#SP#](#)などの生成されたデリミタが含まれます。このようなデリミタの大文字と小文字は異なる場合があるため、依存しないでください。

24.6.1 パーティショニングキー、主キー、および一意キー

このセクションでは、パーティショニングキーと主キーおよび一意キーとの関係について説明します。この関係を制御するルールは次のように表現できます。パーティション化されたテーブルのパーティショニング式で使用されるすべてのカラムは、テーブルが持つことができるすべての一意キーの一部である必要があります。

つまり、テーブルのすべての一意キーは、テーブルパーティション化式のすべてのカラムを使用する必要があります。(これには、定義上一意のキーであるため、テーブルの主キーも含まれます。この点については、このセクションで後述します)。たとえば、次の各テーブル作成ステートメントは無効です。

```
CREATE TABLE t1 (  
  col1 INT NOT NULL,  
  col2 DATE NOT NULL,  
  col3 INT NOT NULL,  
  col4 INT NOT NULL,  
  UNIQUE KEY (col1, col2)  
)  
PARTITION BY HASH(col3)  
PARTITIONS 4;  
  
CREATE TABLE t2 (  
  col1 INT NOT NULL,  
  col2 DATE NOT NULL,  
  col3 INT NOT NULL,  
  col4 INT NOT NULL,  
  UNIQUE KEY (col1),  
  UNIQUE KEY (col3)  
)  
PARTITION BY HASH(col1 + col3)  
PARTITIONS 4;
```

どちらの場合も、記述されたテーブルには、パーティショニング式に使用されているすべてのカラムを含んでいない一意キーが少なくとも 1 つあります。

次の各ステートメントは有効で、対応する無効なテーブル作成ステートメントを機能させる 1 つの方法を示しています。

```
CREATE TABLE t1 (  
  col1 INT NOT NULL,  
  col2 DATE NOT NULL,  
  col3 INT NOT NULL,  
  col4 INT NOT NULL,  
  UNIQUE KEY (col1, col2, col3)
```

```
)  
PARTITION BY HASH(col3)  
PARTITIONS 4;  
  
CREATE TABLE t2 (  
  col1 INT NOT NULL,  
  col2 DATE NOT NULL,  
  col3 INT NOT NULL,  
  col4 INT NOT NULL,  
  UNIQUE KEY (col1, col3)  
)  
PARTITION BY HASH(col1 + col3)  
PARTITIONS 4;
```

次の例は、そのような場合に生成されるエラーを示しています。

```
mysql> CREATE TABLE t3 (  
-> col1 INT NOT NULL,  
-> col2 DATE NOT NULL,  
-> col3 INT NOT NULL,  
-> col4 INT NOT NULL,  
-> UNIQUE KEY (col1, col2),  
-> UNIQUE KEY (col3)  
-> )  
-> PARTITION BY HASH(col1 + col3)  
-> PARTITIONS 4;  
ERROR 1491 (HY000): A PRIMARY KEY must include all columns in the table's partitioning function
```

この CREATE TABLE ステートメントは、指定されたパーティショニングキーに col1 および col3 が含まれているけれども、これらのカラムのいずれもテーブルの両方の一意キーの一部でないために、失敗します。無効なテーブル定義の考えられる解決策の 1 つを次に示します。

```
mysql> CREATE TABLE t3 (  
-> col1 INT NOT NULL,  
-> col2 DATE NOT NULL,  
-> col3 INT NOT NULL,  
-> col4 INT NOT NULL,  
-> UNIQUE KEY (col1, col2, col3),  
-> UNIQUE KEY (col3)  
-> )  
-> PARTITION BY HASH(col3)  
-> PARTITIONS 4;  
Query OK, 0 rows affected (0.05 sec)
```

この場合、指定されたパーティショニングキー col3 は両方の一意キーの一部であるため、このテーブル作成ステートメントは成功します。

次のテーブルは、両方の一意キーに属するカラムをパーティショニングキーに含めることができないため、パーティション化できません。

```
CREATE TABLE t4 (  
  col1 INT NOT NULL,  
  col2 INT NOT NULL,  
  col3 INT NOT NULL,  
  col4 INT NOT NULL,  
  UNIQUE KEY (col1, col3),  
  UNIQUE KEY (col2, col4)  
);
```

すべての主キーは自明で一意キーであるため、この制約にはテーブルの主キーも含まれます (ある場合)。たとえば、次の 2 つのステートメントは無効です。

```
CREATE TABLE t5 (  
  col1 INT NOT NULL,  
  col2 DATE NOT NULL,  
  col3 INT NOT NULL,  
  col4 INT NOT NULL,  
  PRIMARY KEY (col1, col2)  
)  
PARTITION BY HASH(col3)  
PARTITIONS 4;
```

```
CREATE TABLE t6 (  
  col1 INT NOT NULL,  
  col2 DATE NOT NULL,  
  col3 INT NOT NULL,  
  col4 INT NOT NULL,  
  PRIMARY KEY(col1, col3),  
  UNIQUE KEY(col2)  
)  
PARTITION BY HASH( YEAR(col2) )  
PARTITIONS 4;
```

どちらの場合も、パーティショニング式で参照されるすべてのカラムが主キーに含まれていません。ただし、次の2つのステートメントは両方とも有効です。

```
CREATE TABLE t7 (  
  col1 INT NOT NULL,  
  col2 DATE NOT NULL,  
  col3 INT NOT NULL,  
  col4 INT NOT NULL,  
  PRIMARY KEY(col1, col2)  
)  
PARTITION BY HASH(col1 + YEAR(col2))  
PARTITIONS 4;
```

```
CREATE TABLE t8 (  
  col1 INT NOT NULL,  
  col2 DATE NOT NULL,  
  col3 INT NOT NULL,  
  col4 INT NOT NULL,  
  PRIMARY KEY(col1, col2, col4),  
  UNIQUE KEY(col2, col1)  
)  
PARTITION BY HASH(col1 + YEAR(col2))  
PARTITIONS 4;
```

テーブルに一意キーがない場合(主キーがない場合を含む)はこの制約は適用されず、カラム型がパーティショニングタイプと互換性があるかぎり、パーティショニング式に任意のカラムを使用できます。

同じ理由で、テーブルのパーティショニング式で使用されるすべてのカラムが一意キーに含まれている場合を除き、パーティション化されたテーブルにあとから一意キーを追加することはできません。次のように作成されたパーティション化されたテーブルがあるとします。

```
mysql> CREATE TABLE t_no_pk (c1 INT, c2 INT)  
-> PARTITION BY RANGE(c1) (  
-> PARTITION p0 VALUES LESS THAN (10),  
-> PARTITION p1 VALUES LESS THAN (20),  
-> PARTITION p2 VALUES LESS THAN (30),  
-> PARTITION p3 VALUES LESS THAN (40)  
-> );  
Query OK, 0 rows affected (0.12 sec)
```

次のいずれかの `ALTER TABLE` ステートメントを使用することで、`t_no_pk` に主キーを追加できます。

```
# possible PK  
mysql> ALTER TABLE t_no_pk ADD PRIMARY KEY(c1);  
Query OK, 0 rows affected (0.13 sec)  
Records: 0 Duplicates: 0 Warnings: 0  
  
# drop this PK  
mysql> ALTER TABLE t_no_pk DROP PRIMARY KEY;  
Query OK, 0 rows affected (0.10 sec)  
Records: 0 Duplicates: 0 Warnings: 0  
  
# use another possible PK  
mysql> ALTER TABLE t_no_pk ADD PRIMARY KEY(c1, c2);  
Query OK, 0 rows affected (0.12 sec)  
Records: 0 Duplicates: 0 Warnings: 0  
  
# drop this PK  
mysql> ALTER TABLE t_no_pk DROP PRIMARY KEY;  
Query OK, 0 rows affected (0.09 sec)  
Records: 0 Duplicates: 0 Warnings: 0
```


ただし、次のステートメントは失敗します。c1 は、パーティショニングキーの一部ですが、指定された主キーの一部ではないためです。

```
# fails with error 1503
mysql> ALTER TABLE t_no_pk ADD PRIMARY KEY(c2);
ERROR 1503 (HY000): A PRIMARY KEY must include all columns in the table's partitioning function
```

t_no_pk のパーティショニング式は c1 のみであるため、c2 のみに一意キーを追加しようとすると失敗します。ただし、c1 および c2 の両方を使用する一意キーは追加できます。

これらのルールは、既存のパーティション化されていないテーブルを ALTER TABLE ... PARTITION BY を使用してパーティション化するときにも適用されます。次のように作成されたテーブル np_pk があるとします。

```
mysql> CREATE TABLE np_pk (
-> id INT NOT NULL AUTO_INCREMENT,
-> name VARCHAR(50),
-> added DATE,
-> PRIMARY KEY (id)
-> );
Query OK, 0 rows affected (0.08 sec)
```

次の ALTER TABLE ステートメントは、added カラムがテーブルの一意キーの一部でないため、エラーで失敗します。

```
mysql> ALTER TABLE np_pk
-> PARTITION BY HASH( TO_DAYS(added) )
-> PARTITIONS 4;
ERROR 1503 (HY000): A PRIMARY KEY must include all columns in the table's partitioning function
```

ただし、次に示すように、パーティショニングカラムに id カラムを使用する次のステートメントは有効です。

```
mysql> ALTER TABLE np_pk
-> PARTITION BY HASH(id)
-> PARTITIONS 4;
Query OK, 0 rows affected (0.11 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

np_pk の場合、パーティショニング式の一部として使用できる唯一のカラムは id です。このテーブルをパーティショニング式でほかのカラムを使用してパーティション化する場合は、目的のカラムを主キーに追加するか、主キーをすべて削除することによって、まずテーブルを変更する必要があります。

24.6.2 ストレージエンジンに関連するパーティショニング制限

MySQL 8.0 では、パーティショニングサポートは実際には MySQL Server によって提供されるのではなく、テーブルストレージエンジンの所有するパーティショニングハンドラまたはネイティブパーティショニングハンドラによって提供されます。MySQL 8.0 では、InnoDB ストレージエンジンのみがネイティブのパーティショニングハンドラを提供します。つまり、パーティション化されたテーブルは、ほかのストレージエンジンを使用して作成できません。

注記

MySQL Cluster NDB ストレージエンジンは、ネイティブパーティショニングサポートも提供しますが、MySQL 8.0 では現在サポートされていません。

ALTER TABLE ... OPTIMIZE PARTITION は、InnoDB を使用するパーティションテーブルでは正しく機能しません。そのようなテーブルの場合は、代わりに ALTER TABLE ... REBUILD PARTITION および ALTER TABLE ... ANALYZE PARTITION を使用してください。詳細は、[セクション13.1.9.1「ALTER TABLE パーティション操作」](#)を参照してください。

ユーザー定義のパーティション分割と NDB ストレージエンジン (NDB Cluster)。KEY (LINEAR KEY を含む) によるパーティショニングは、NDB ストレージエンジンでサポートされる唯一のパーティショニングタイプです。「NDB Cluster」の通常の状態では、[LINEAR] KEY 以外のパーティション化タイプを使用して「NDB Cluster」テーブルを作成することはできず、作成しようとするとエラーで失敗します。

例外 (本番用ではない): NDB Cluster SQL ノード上の new システム変数を ON に設定することで、この制限をオーバーライドできます。これを行う場合、[LINEAR] KEY 以外のパーティション化タイプを使用するテーブルは本番で

サポートされないことに注意してください。「このような場合は、[KEY](#) または [LINEAR KEY](#) 以外のパーティショニングタイプのテーブルを作成して使用できますが、これは完全にリスクがあります」。

[NDB](#) テーブルに定義できるパーティションの最大数は、クラスタ内のデータノードとノードグループの数、使用中の [NDB Cluster](#) ソフトウェアのバージョン、およびその他の要因によって異なります。詳細は、[NDB とユーザー定義のパーティショニング](#)を参照してください。

[NDB](#) テーブルのパーティショニングごとに格納できる固定サイズデータの最大量は 128 TB です。以前は 16 GB でした。

ユーザーパーティショニングされた [NDB](#) テーブルが次の 2 つの要件のいずれかまたは両方を満たさなくなる [CREATE TABLE](#) ステートメントおよび [ALTER TABLE](#) ステートメントは許可されず、エラーで失敗します。

1. テーブルに明示的な主キーが存在する必要があります。
2. テーブルのパーティショニング式に指定されたすべてのカラムが主キーの一部である必要があります。

例外. ユーザーパーティショニングされた [NDB](#) テーブルが空のカラムリストを使用して (つまり、[PARTITION BY KEY\(\)](#) または [PARTITION BY LINEAR KEY\(\)](#) を使用して) 作成された場合、明示的な主キーは必要ありません。

パーティショニングされたテーブルをアップグレードする。アップグレードを実行する場合は、[KEY](#) によってパーティショニングされたテーブルをダンプしてリロードする必要があります。[InnoDB](#) 以外のストレージエンジンを使用するパーティショニングされたテーブルは、MySQL 5.7 以前から MySQL 8.0 以降にアップグレードできません。[ALTER TABLE ... REMOVE PARTITIONING](#) を使用してこのようなテーブルからパーティショニングを削除するか、アップグレード前に [ALTER TABLE ... ENGINE=INNODB](#) を使用して [InnoDB](#) に変換する必要があります。

[MyISAM](#) テーブルの [InnoDB](#) への変換の詳細は、[セクション15.6.1.5「MyISAM から InnoDB へのテーブルの変換」](#)を参照してください。

24.6.3 関数に関連するパーティショニング制限

このセクションでは特に、パーティショニング式で使用される関数に関連する、MySQL パーティショニングの制限について説明します。

パーティショニング式で使用できるのは、次のリストに示す MySQL 関数のみです:

- [ABS\(\)](#)
- [CEILING\(\)](#) ([CEILING\(\)](#) および [FLOOR\(\)](#)を参照してください)
- [DATEDIFF\(\)](#)
- [DAY\(\)](#)
- [DAYOFMONTH\(\)](#)
- [DAYOFWEEK\(\)](#)
- [DAYOFYEAR\(\)](#)
- [EXTRACT\(\)](#) ([WEEK](#) 指定子付きの [EXTRACT\(\)](#) 関数を参照してください)
- [FLOOR\(\)](#) ([CEILING\(\)](#) および [FLOOR\(\)](#)を参照してください)
- [HOUR\(\)](#)
- [MICROSECOND\(\)](#)
- [MINUTE\(\)](#)
- [MOD\(\)](#)
- [MONTH\(\)](#)
- [QUARTER\(\)](#)

- [SECOND\(\)](#)
- [TIME_TO_SEC\(\)](#)
- [TO_DAYS\(\)](#)
- [TO_SECONDS\(\)](#)
- [UNIX_TIMESTAMP\(\)](#) (TIMESTAMP カラムを含む)
- [WEEKDAY\(\)](#)
- [YEAR\(\)](#)
- [YEARWEEK\(\)](#)

MySQL 8.0 では、[TO_DAYS\(\)](#)、[TO_SECONDS\(\)](#)、[YEAR\(\)](#) および [UNIX_TIMESTAMP\(\)](#) 関数でパーティショニングがサポートされています。詳細は、[セクション24.4「パーティショニング」](#)を参照してください。

[CEILING\(\)](#) および [FLOOR\(\)](#)。これらの各関数は、正確な数値型 (INT 型または DECIMAL 型のいずれかなど) の引数を渡された場合にのみ整数を返します。これはたとえば、次の [CREATE TABLE](#) ステートメントがここで示すようにエラーで失敗することを意味します。

```
mysql> CREATE TABLE t (c FLOAT) PARTITION BY LIST( FLOOR(c) )(
-> PARTITION p0 VALUES IN (1,3,5),
-> PARTITION p1 VALUES IN (2,4,6)
-> );
ERROR 1490 (HY000): The PARTITION function returns the wrong type
```

WEEK 指定子付きの [EXTRACT\(\)](#) 関数。 [EXTRACT\(\)](#) 関数によって返される値は、[EXTRACT\(WEEK FROM col\)](#) として使用されるときに、[default_week_format](#) システム変数の値に依存します。このため、ユニットを WEEK として指定した場合、[EXTRACT\(\)](#) はパーティショニング関数として許可されません。(Bug #54483)

これらの関数の戻り型についての詳細は、[セクション12.6.2「数学関数」](#) および [セクション11.1「数値データ型」](#)を参照してください。

第 25 章 ストアドオブジェクト

目次

25.1	ストアドプログラムの定義	4096
25.2	ストアドルーチンの使用	4097
25.2.1	ストアドルーチンの構文	4097
25.2.2	ストアドルーチンと MySQL 権限	4098
25.2.3	ストアドルーチンのメタデータ	4099
25.2.4	ストアドプロシージャ、関数、トリガー、および LAST_INSERT_ID()	4099
25.3	トリガーの使用	4099
25.3.1	トリガーの構文と例	4100
25.3.2	トリガーのメタデータ	4103
25.4	イベントスケジューラの使用	4104
25.4.1	イベントスケジューラの概要	4104
25.4.2	イベントスケジューラの構成	4105
25.4.3	イベント構文	4107
25.4.4	イベントメタデータ	4107
25.4.5	イベントスケジューラのステータス	4108
25.4.6	イベントスケジューラと MySQL 権限	4108
25.5	ビューの使用	4111
25.5.1	ビューの構文	4111
25.5.2	ビュー処理アルゴリズム	4111
25.5.3	更新可能および挿入可能なビュー	4112
25.5.4	WITH CHECK OPTION 句の表示	4115
25.5.5	ビューのメタデータ	4116
25.6	ストアドオブジェクトのアクセス制御	4116
25.7	ストアドプログラムバイナリロギング	4119
25.8	ストアドプログラムの制約	4125
25.9	ビューの制約	4128

この章では、後で実行するためにサーバーに格納される SQL コードで定義されるストアドデータベースオブジェクトについて説明します。

ストアドオブジェクトには、次のオブジェクトタイプが含まれます:

- ストアドプロシージャ: `CREATE PROCEDURE` で作成され、`CALL` ステートメントを使用して起動されるオブジェクト。プロシージャは、戻り値がありませんが、呼び出し元があとから検査できるようにそのパラメータを変更できます。また、クライアントプログラムに戻される結果セットも生成できます。
- ストアドファンクション: `CREATE FUNCTION` で作成され、組み込み関数と同様に使用されるオブジェクト。式で呼び出し、式の評価中に値を返します。
- トリガー: テーブルに関連付けられた `CREATE TRIGGER` で作成されたオブジェクト。トリガーは、テーブルに対して挿入や更新などの特定のイベントが発生したときにアクティブ化されます。
- Event: `CREATE EVENT` で作成され、スケジュールに従ってサーバーによって起動されるオブジェクト。
- 表示: 参照時に結果セットを生成する `CREATE VIEW` で作成されるオブジェクト。ビューは仮想テーブルとして機能します。

このドキュメントで使用されている用語は、格納されているオブジェクト階層を反映しています:

- ストアドルーチンには、ストアドプロシージャとストアドファンクションが含まれます。
- ストアドプログラムには、ストアドルーチン、トリガー、およびイベントが含まれます。
- ストアドオブジェクトには、ストアドプログラムおよびビューが含まれます。

この章では、ストアドオブジェクトの使用方法について説明します。次の各セクションでは、これらのオブジェクトに関連するステートメントの SQL 構文およびオブジェクト処理に関する追加情報を示します：

- オブジェクト型ごとに、どのオブジェクトが存在し、どのように定義されているかを制御する `CREATE`、`ALTER`、および `DROP` ステートメントがあります。 [セクション13.1「データ定義ステートメント」](#) を参照してください。
- `CALL` ステートメントは、ストアドプロシージャの呼び出しに使用されます。 [セクション13.2.1「CALL ステートメント」](#) を参照してください。
- スタドプログラム定義には、複合ステートメント、ループ、条件文、および宣言された変数を使用できる本体が含まれます。 [セクション13.6「複合ステートメントの構文」](#) を参照してください。
- スタドプログラムによって参照されるオブジェクトに対するメタデータの変更が検出され、プログラムが次に実行されるたびに、影響を受けるステートメントが自動的に再解析されます。詳細については、 [セクション 8.10.3「プリパードステートメントおよびストアドプログラムのキャッシュ」](#) を参照してください。

25.1 スタドプログラムの定義

各ストアドプログラムには、SQL ステートメントから構成される本体が含まれます。このステートメントは、セミコロン (;) 文字で区切られた複数のステートメントから構成される複合ステートメントの場合があります。たとえば、次のストアドプロシージャには、`SET` ステートメントと `REPEAT` ループ (ループ自体に別の `SET` ステートメントが含まれます) を含む `BEGIN ... END` ブロックから構成される本体があります。

```
CREATE PROCEDURE dorepeat(p1 INT)
BEGIN
  SET @x = 0;
  REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
END;
```

`mysql` クライアントプログラムを使用してセミコロン文字を含むストアドプログラムを定義すると、問題が発生します。デフォルトでは、`mysql` 自体はセミコロンをステートメント区切り文字と認識します。したがって、`mysql` がストアドプログラム定義全体をサーバーに渡すように、区切り文字を一時的に再定義する必要があります。

`mysql` の区切り文字を再定義するには、`delimiter` コマンドを使用します。次の例は、上記の `dorepeat()` プロシージャについてこれを行う方法を示しています。区切り文字は `//` に変更され、定義全体を単一のステートメントとしてサーバーに渡して、プロシージャの呼び出し前に ; にリストアできます。これにより、プロシージャ本体で使用される ; 区切り文字を、`mysql` 自体が解釈するのではなく、サーバーに渡すようにすることができます。

```
mysql> delimiter //
mysql> CREATE PROCEDURE dorepeat(p1 INT)
-> BEGIN
-> SET @x = 0;
-> REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
-> END
-> //
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> CALL dorepeat(1000);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x;
+-----+
| @x |
+-----+
| 1001 |
+-----+
1 row in set (0.00 sec)
```

区切り文字を `//` 以外の文字列に再定義でき、区切り文字は単一の文字から構成することも、複数の文字から構成することもできます。バックスラッシュ (\) 文字は、MySQL のエスケープ文字なので使用しないでください。

次に、パラメータを受け取り、SQL 関数を使用して操作を実行したあと、結果を返す関数例を示します。この場合は、関数定義に内部の ; ステートメント区切り文字が含まれていないため、`delimiter` を使用する必要はありません。


```
mysql> CREATE FUNCTION hello(s CHAR(20))
mysql> RETURNS CHAR(50) DETERMINISTIC
-> RETURN CONCAT('Hello, ',s, '!');
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT hello('world');
+-----+
| hello('world') |
+-----+
| Hello, world! |
+-----+
1 row in set (0.00 sec)
```

25.2 ストアドルーチンの使用

MySQL は、ストアドルーチン (プロシージャとファンクション) をサポートしています。ストアドルーチンとは、サーバーに格納できる一連の SQL ステートメントです。これが行われていると、クライアントは個々のステートメントを繰り返し発行し続ける必要はなく、代わりにストアドルーチンを参照できます。

ストアドルーチンは特に、次のような特定の状況で役立ちます。

- クライアントアプリケーションが異なる言語で作成されているか、異なるプラットフォームで動作しているが、同じデータベース操作を実行する必要がある場合。
- セキュリティーが最重要である場合。たとえば、銀行では、すべての一般的な操作に対してストアドプロシージャおよびストアドファンクションを使用します。これにより一貫したセキュアな環境が得られ、ルーチンによってそれぞれの操作が正しく記録されるようになります。このようなセットアップでは、アプリケーションおよびユーザーはデータベーステーブルに直接アクセスできませんが、特定のストアドルーチンだけを実行できます。

ストアドルーチンは、サーバーとクライアント間で送信する必要のある情報が少なくなるので、パフォーマンスを改善できます。そのトレードオフでは、これによりサーバー側で行われる作業が増え、クライアント (アプリケーション) 側で行われる作業が少なくなるので、データベースサーバーでのロードが増大します。1 台または少数のデータベースサーバーだけで多数のクライアントマシン (Web サーバーなど) にサービスを提供している場合にはこれを検討してください。

ストアドルーチンを使用すれば、データベースサーバーで関数のライブラリを保持することもできます。これは、内部的に (たとえばクラスを使用して) このような設計を可能にする、現代のアプリケーション言語で共有されている機能です。これらのクライアントアプリケーションの言語機能を使用すると、データベース使用のスコープ外でもプログラムにとって利点があります。

MySQL はストアドルーチンについて SQL:2003 構文に従っており、これは IBM の DB2 でも使用されています。ここで説明するすべての構文はサポートされており、すべての制限と拡張が適宜ドキュメント化されています。

追加のリソース

- ストアドプロシージャおよびストアドファンクションを扱うときに、[ストアドプロシージャのユーザーフォーラム](#)が役立ちます。
- MySQL のストアドルーチンに関するよくある質問とその回答については、[セクションA.4「MySQL 8.0 FAQ: ストアドプロシージャおよびストアドファンクション」](#)を参照してください。
- ストアドルーチンの使用にはいくつかの制限があります。[セクション25.8「ストアドプログラムの制約」](#)を参照してください。
- ストアドルーチンのバイナリロギングは、[セクション25.7「ストアドプログラムバイナリロギング」](#)で説明しているように行われます。

25.2.1 ストアドルーチンの構文

ストアドルーチンはプロシージャまたは関数のどちらかです。ストアドルーチンは、[CREATE PROCEDURE](#) および [CREATE FUNCTION](#) ステートメントで作成されます ([セクション13.1.17「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」](#)を参照してください)。プロシージャは [CALL](#) ステートメントを使用して呼び出され ([セクション13.2.1「CALL ステートメント」](#)を参照してください)、出力変数の使用でのみ値を戻

することができます。関数は、ほかの関数とまったく同様に (つまり、関数の名前を呼び出すことによって) ステートメント内部から呼び出すことができ、スカラー値を戻すことができます。ストアドルーチンの本体では、複合ステートメントを使用できます ([セクション13.6「複合ステートメントの構文」](#)を参照してください)。

ストアドルーチンは、[DROP PROCEDURE](#) および [DROP FUNCTION](#) ステートメントで削除でき ([セクション13.1.29「DROP PROCEDURE および DROP FUNCTION ステートメント」](#)を参照してください)、[ALTER PROCEDURE](#) および [ALTER FUNCTION](#) ステートメントで変更できます ([セクション13.1.7「ALTER PROCEDURE ステートメント」](#)を参照してください)。

ストアドプロシージャまたはストアドファンクションは、特定のデータベースに関連付けられています。これにはいくつかの問題があります。

- ルーチンが呼び出されると、暗黙の `USE db_name` が実行されます (その後、ルーチンが終了すると元に戻ります)。ストアドルーチン内での `USE` ステートメントは許可されていません。
- データベース名でルーチン名を修飾できます。これは現在のデータベースに存在しないルーチンを参照する場合に使用できます。たとえば、`test` データベースに関連するストアドプロシージャ `p` またはストアドファンクション `f` を呼び出すには、`CALL test.p()` または `test.f()` と指定します。
- データベースを削除すると、そのデータベースに関連付けられたすべてのストアドルーチンも削除されます。

ストアドファンクションは再帰関数にはできません。

ストアドプロシージャでの再帰は許可されていますが、デフォルトでは無効になっています。再帰を有効にするには、`max_sp_recursion_depth` サーバースystem変数を正の値に設定します。ストアドプロシージャの再帰により、スレッドスタック領域の要求が増加します。`max_sp_recursion_depth` の値を増やした場合、サーバ起動時に `thread_stack` の値を増やすことによってスレッドスタックサイズを増やすことが必要な場合もあります。詳細は、[セクション5.1.8「サーバースystem変数」](#)を参照してください。

MySQL では、通常の `SELECT` ステートメントをストアドプロシージャ内で (つまり、カーソルまたはローカル変数を使用せずに) 使用できるようにする非常に役立つ拡張をサポートしています。このようなクエリーの結果セットは単にクライアントに直接送信されます。複数の `SELECT` ステートメントは複数の結果セットを生成するので、クライアントは複数の結果セットをサポートしている MySQL クライアントライブラリを使用する必要があります。これは、クライアントが、4.1 以降の MySQL のバージョンからクライアントライブラリを使用する必要があることを意味します。クライアントは、接続するときに、`CLIENT_MULTI_RESULTS` オプションも指定する必要があります。C プログラムの場合、これは、`mysql_real_connect()` C API 関数で実行できます。[mysql_real_connect\(\)](#) および [Multiple Statement Execution Support](#) を参照してください。

MySQL 8.0.22 以降では、ストアドプロシージャのステートメントによって参照されるユーザー変数のタイプは、プロシージャの初回起動時に決定され、その後プロシージャが起動されるたびにこのタイプが保持されます。

25.2.2 ストアドルーチンと MySQL 権限

MySQL 許可システムはストアドルーチンを次のように考慮します。

- ストアドルーチンを生成するには、[CREATE ROUTINE](#) 権限が必要です。
- ストアドルーチンを変更または削除するには、[ALTER ROUTINE](#) 権限が必要です。この権限は、必要に応じて、ルーチンの作成者に自動的に与えられ、ルーチンが削除されると作成者から削除されます。
- ストアドルーチンを実行するには、[EXECUTE](#) 権限が必要です。ただし、この権限は、必要に応じて、ルーチンの作成者に自動的に与えられます (ルーチンが削除されると作成者から削除されます)。また、ルーチンのデフォルトの `SQL SECURITY` 特性は `DEFINER` であり、これにより、ルーチンが関連付けられているデータベースにアクセス可能なユーザーがルーチンを実行できるようになります。
- `automatic_sp_privileges` システム変数が 0 である場合、[EXECUTE](#) および [ALTER ROUTINE](#) 権限は作成者に対して自動的に付与および削除されません。
- ルーチンの作成者は、ルーチンの `CREATE` ステートメントを実行するために使用されるアカウントです。これは、ルーチン定義で `DEFINER` として名前が指定されているアカウントと同じでないことがあります。
- ルーチン `DEFINER` として指定されたアカウントは、その定義を含むすべてのルーチンプロパティを表示できます。したがって、アカウントは、次によって生成されるルーチン出力への完全なアクセス権を持ちます:

- `INFORMATION_SCHEMA.ROUTINES` テーブルの内容。
- `SHOW CREATE FUNCTION` および `SHOW CREATE PROCEDURE` ステートメント。
- `SHOW FUNCTION CODE` および `SHOW PROCEDURE CODE` ステートメント。
- `SHOW FUNCTION STATUS` および `SHOW PROCEDURE STATUS` ステートメント。
- ルーチン `DEFINER` として指定されたアカウント以外のアカウントのルーチンプロパティへのアクセスは、アカウントに付与された権限によって異なります:
 - `SHOW ROUTINE` 権限またはグローバル `SELECT` 権限を持つアカウントは、その定義を含むすべてのルーチンプロパティを表示できます。
 - `CREATE ROUTINE`、`ALTER ROUTINE` または `EXECUTE` 権限がルーチンを含むスコープで付与されている場合、アカウントはその定義を除くすべてのルーチンプロパティを表示できます。

25.2.3 ストアドルーチンのメタデータ

ストアドルーチンに関するメタデータを取得するには:

- `INFORMATION_SCHEMA` データベースの `ROUTINES` テーブルをクエリーします。 [セクション 26.30 「INFORMATION_SCHEMA ROUTINES テーブル」](#) を参照してください。
- `SHOW CREATE PROCEDURE` および `SHOW CREATE FUNCTION` ステートメントを使用して、ルーチン定義を表示します。 [セクション 13.7.7.9 「SHOW CREATE PROCEDURE ステートメント」](#) を参照してください。
- `SHOW PROCEDURE STATUS` および `SHOW FUNCTION STATUS` ステートメントを使用して、ルーチン特性を表示します。 [セクション 13.7.7.28 「SHOW PROCEDURE STATUS ステートメント」](#) を参照してください。
- `SHOW PROCEDURE CODE` および `SHOW FUNCTION CODE` ステートメントを使用して、ルーチンの内部実装の表現を確認します。 [セクション 13.7.7.27 「SHOW PROCEDURE CODE ステートメント」](#) を参照してください。

25.2.4 ストアドプロシージャー、関数、トリガー、および `LAST_INSERT_ID()`

ストアドルーチン (プロシージャーまたは関数) またはトリガーの本体内では、`LAST_INSERT_ID()` の値は、このような種類のオブジェクトの本体外で実行されたステートメントと同様に変更されます ([セクション 12.16 「情報関数」](#) を参照してください)。あとに続くステートメントで参照される `LAST_INSERT_ID()` の値でのストアドルーチンまたはトリガーの効果は、ルーチンの種類によって異なります。

- ストアドプロシージャーで `LAST_INSERT_ID()` の値を変更するステートメントが実行される場合は、プロシージャー呼び出しが続くステートメントで変更された値が参照されます。
- 値を変更するストアドファンクションやトリガーでは、値は関数やトリガーが終了したときにリストアされるので、後続のステートメントは変更された値を表示しません。

25.3 トリガーの使用

トリガーとは、テーブルに関連付けられ、そのテーブルに対して特定のイベントが発生するとアクティブ化される名前付きデータベースオブジェクトのことです。トリガーを使用する場合には、テーブルに挿入する値のチェックを実行したり、更新にかかわる値の計算を実行したりする場合があります。

トリガーは、関連付けられたテーブルでステートメントが行の挿入、更新、または削除を行なったときにアクティブ化するように定義されます。これらの行操作がトリガーイベントになります。たとえば、行は、`INSERT` または `LOAD DATA` ステートメントで挿入でき、挿入トリガーは挿入された行ごとにアクティブ化します。トリガーは、トリガーイベントの前または後のどちらかでアクティブ化するように設定できます。たとえば、テーブルに挿入される各行の前、または更新される各行のあとでトリガーをアクティブ化させることができます。

重要

MySQL のトリガーは、SQL ステートメントがテーブルに対して行なった変更の場合にのみアクティブ化します。これには、更新可能なビューの基礎となる実テーブルに対する変

更が含まれます。トリガーでは、SQL ステートメントを MySQL Server に送信しない API によって行われたテーブルに対する変更はアクティブ化されません。これは、トリガーが NDB API を使用して行われた更新によってアクティブ化されないことを意味します。

トリガーは、`INFORMATION_SCHEMA` テーブルまたは `performance_schema` テーブルの変更によってアクティブ化されません。これらのテーブルは実際にはビューであり、トリガーはビューでは許可されません。

次のセクションでは、トリガーを作成および削除するための構文について説明し、使用方法の例をいくつか挙げ、トリガーメタデータを取得する方法を示します。

追加のリソース

- トリガーを扱うときには、[トリガーユーザーフォーラム](#)が役立ちます。
- MySQL でのトリガーに関するよくある質問とその回答については、[セクションA.5「MySQL 8.0 FAQ: トリガー」](#)を参照してください。
- トリガーの使用にはいくつかの制限があります。[セクション25.8「ストアプログラムの制約」](#)を参照してください。
- トリガーのバイナリロギングは、[セクション25.7「ストアプログラムバイナリロギング」](#)で説明しているように行います。

25.3.1 トリガーの構文と例

トリガーを作成したり、トリガーを削除したりするには、[セクション13.1.22「CREATE TRIGGER ステートメント」](#)および[セクション13.1.34「DROP TRIGGER ステートメント」](#)で説明しているように、`CREATE TRIGGER` または `DROP TRIGGER` ステートメントを使用します。

次に、`INSERT` 操作に対してアクティブ化するトリガーをテーブルに関連付ける簡単な例を示します。このトリガーは加算器として機能し、テーブルのいずれかのカラムに挿入された値を合計します。

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
Query OK, 0 rows affected (0.03 sec)

mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account
FOR EACH ROW SET @sum = @sum + NEW.amount;
Query OK, 0 rows affected (0.01 sec)
```

`CREATE TRIGGER` ステートメントは、`account` テーブルに関連付けられている `ins_sum` という名前のトリガーを作成します。トリガーアクションタイム、トリガーイベント、およびトリガーがアクティブ化したときに行う動作を指定する句も含まれます。

- キーワード `BEFORE` は、トリガーアクションタイムを示します。この場合、トリガーは、テーブルに挿入された各行の前にアクティブ化します。ここで許可されている別のキーワードは `AFTER` です。
- キーワード `INSERT` は、トリガーイベント、つまりトリガーをアクティブ化する操作の種類を示します。例では、`INSERT` 操作がトリガーのアクティブ化を引き起こします。`DELETE` および `UPDATE` 操作に対するトリガーも作成できます。
- `FOR EACH ROW` に続くステートメントは、トリガー本体を定義します。これは、トリガーがアクティブ化するたびに実行するステートメントであり、トリガーイベントによって影響される行ごとに一度行われます。この例では、トリガー本体は、`amount` カラムに挿入された値をユーザー変数に累積する単純な `SET` です。このステートメントは、「新しい行に挿入される `amount` カラムの値」を意味する `NEW.amount` としてカラムを参照します。

トリガーを使用するには、加算器変数をゼロにセットし、`INSERT` ステートメントを実行して、その後変数がどの値になっているかを確認します。

```
mysql> SET @sum = 0;
mysql> INSERT INTO account VALUES(137,14.98),(141,1937.50),(97,-100.00);
mysql> SELECT @sum AS 'Total amount inserted';
+-----+
| Total amount inserted |
```

```
+-----+
| 1852.48 |
+-----+
```

この場合、`INSERT` ステートメントの実行後の `@sum` の値は `14.98 + 1937.50 - 100` または `1852.48` です。

トリガーを破棄するには、`DROP TRIGGER` ステートメントを使用します。トリガーがデフォルトスキーマにない場合、スキーマ名を指定する必要があります。

```
mysql> DROP TRIGGER test.ins_sum;
```

テーブルを削除すると、そのテーブルのトリガーもすべて削除されます。

トリガー名はスキーマの名前空間内に存在します。つまり、すべてのトリガーがスキーマ内で一意の名前を持つ必要があります。異なるスキーマ内のトリガーは同じ名前を持つことができます。

同じトリガーイベントおよびアクション時間を持つ特定のテーブルに対して複数のトリガーを定義できます。たとえば、1つのテーブルに対して2つの `BEFORE UPDATE` トリガーを定義できます。デフォルトでは、同じトリガーイベントおよびアクション時間を持つトリガーは、作成された順序で実行されます。トリガーの順序を指定するには、`FOR EACH ROW` のあとに `FOLLOWS` または `PRECEDES` を示す句、および同じトリガーイベントとアクション時間を持つ既存のトリガーの名前を指定します。`FOLLOWS` を指定すると、新しいトリガーは既存のトリガーのあとに実行されます。`PRECEDES` を指定すると、新しいトリガーは既存のトリガーの前に実行されます。

たとえば、次のトリガー定義では、`account` テーブルに対して別の `BEFORE INSERT` トリガーを定義します：

```
mysql> CREATE TRIGGER ins_transaction BEFORE INSERT ON account
FOR EACH ROW PRECEDES ins_sum
SET
@deposits = @deposits + IF(NEW.amount>0,NEW.amount,0),
@withdrawals = @withdrawals + IF(NEW.amount<0,-NEW.amount,0);
Query OK, 0 rows affected (0.01 sec)
```

このトリガーである `ins_transaction` は、`ins_sum` と似ていますが、デポジットと引出しを別々に累計します。これには、`ins_sum` の前にアクティブ化する `PRECEDES` 句があります。この句がない場合、`ins_sum` の後に作成されるため、`ins_sum` の後にアクティブ化されます。

トリガー本体内で、`OLD` および `NEW` キーワードを使用すると、トリガーの影響を受ける行のカラムにアクセスできます。`OLD` および `NEW` は、トリガーに対する MySQL の拡張機能であり、大/小文字は区別されません。

`INSERT` トリガー内では、`NEW.col_name` だけを使用できます。古い行はありません。`DELETE` トリガーでは、`OLD.col_name` だけを使用できます。新しい行はありません。`UPDATE` トリガーでは、`OLD.col_name` を使用して、更新される前の行のカラムを参照でき、`NEW.col_name` を使用して、更新されたあとの行のカラムを参照できます。

`OLD` で指名されたカラムは読み取り専用です。(それに対する `SELECT` 権限がある場合)参照はできますが、変更はできません。`NEW` で指名されたカラムは、それに対する `SELECT` 権限がある場合に参照できます。`BEFORE` トリガーでは、それに対する `UPDATE` 権限がある場合、`SET NEW.col_name = value` でその値を変更することもできます。これは、トリガーを使用して、新しい行に挿入する値または行の更新に使用される値を変更できることを意味します。(このような `SET` ステートメントは、行の変更がすでに発生しているため、`AFTER` トリガーには影響しません。)

`BEFORE` トリガーでは、`AUTO_INCREMENT` カラムの `NEW` 値は 0 であり、新しい行が実際に挿入されるときに自動的に生成されるシーケンス番号ではありません。

`BEGIN ... END` 構造構文を使用することにより、複数のステートメントを実行するトリガーを定義できます。`BEGIN` ブロック内では、条件文やループなど、ストアルーチン内で許可されたほかの構文を使用することもできます。ただし、ストアルーチンの場合と同様に、`mysql` プログラムを使用して、複数のステートメントを実行するトリガーを定義する場合、トリガー定義内で ; ステートメント区切り文字を使用できるように、`mysql` ステートメント区切り文字を再定義する必要があります。次の例はこれらの要点を示しています。ここでは、各行の更新に使用する新しい値をチェックし、0 から 100 の範囲に収まるように値を変更する `UPDATE` トリガーを定義しています。行の更新に使用される前に値をチェックする必要がありますので、これは `BEFORE` トリガーにする必要があります。

```
mysql> delimiter //
mysql> CREATE TRIGGER upd_check BEFORE UPDATE ON account
```



```

FOR EACH ROW
BEGIN
  IF NEW.amount < 0 THEN
    SET NEW.amount = 0;
  ELSEIF NEW.amount > 100 THEN
    SET NEW.amount = 100;
  END IF;
END;
//
mysql> delimiter ;

```

ストアードプロシージャを個別に定義してから、単純な **CALL** ステートメントを使用してトリガーから呼び出したほうが簡単になる場合があります。これは、複数のトリガー内から同じコードを実行する場合にも便利です。

アクティブ化したときにトリガーが実行するステートメントに表示できる対象には制限があります。

- トリガーは、**CALL** ステートメントを使用して、データをクライアントに戻すストアードプロシージャや、ダイナミック SQL を使用するストアードプロシージャの呼び出しはできません。(ストアードプロシージャは、**OUT** または **INOUT** パラメータを通じてトリガーにデータを返すことが許可されています。)
- トリガーは、**START TRANSACTION**、**COMMIT**、**ROLLBACK** など、トランザクションを明示的または暗黙的に開始したり終了したりするステートメントを使用できません。(**ROLLBACK to SAVEPOINT** はトランザクションを終了しないため、許可されます。)

[セクション25.8「ストアードプログラムの制約」](#) も参照してください。

MySQL は次のようにトリガー実行中にエラーを処理します。

- **BEFORE** トリガーが失敗した場合、対応する行に対する操作は実行されません。
- **BEFORE** トリガーは、行を挿入または変更しようとする試行によってアクティブ化され、その試行がその後成功するかどうかには関係ありません。
- **AFTER** トリガーは、すべての **BEFORE** トリガーと行操作の実行が成功した場合にのみ実行されます。
- **BEFORE** または **AFTER** トリガーのどちらかの実行中にエラーが発生すると、トリガーの呼び出しを起こしたステートメント全体が失敗します。
- トランザクションテーブルの場合、ステートメントの失敗により、ステートメントが実行したすべての変更がロールバックされます。トリガーの失敗はステートメントの失敗を招くので、トリガーの失敗はロールバックも引き起こします。非トランザクションテーブルの場合、このようなロールバックは行えないので、ステートメントが失敗しても、エラーの時点以前に実行されたすべて変更は有効なままです。

次の例に示す **testref** という名前のトリガーなど、トリガーには、名前によるテーブルへの直接の参照を含めることができます。

```

CREATE TABLE test1(a1 INT);
CREATE TABLE test2(a2 INT);
CREATE TABLE test3(a3 INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
CREATE TABLE test4(
  a4 INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  b4 INT DEFAULT 0
);

delimiter |

CREATE TRIGGER testref BEFORE INSERT ON test1
FOR EACH ROW
BEGIN
  INSERT INTO test2 SET a2 = NEW.a1;
  DELETE FROM test3 WHERE a3 = NEW.a1;
  UPDATE test4 SET b4 = b4 + 1 WHERE a4 = NEW.a1;
END;
|

delimiter ;

INSERT INTO test3 (a3) VALUES
(NULL), (NULL), (NULL), (NULL), (NULL),
(NULL), (NULL), (NULL), (NULL), (NULL);

```



```
INSERT INTO test4 (a4) VALUES  
(0), (0), (0), (0), (0), (0), (0), (0), (0), (0), (0);
```

次に示すように、テーブル `test1` に次の値を挿入するとします。

```
mysql> INSERT INTO test1 VALUES  
(1), (3), (1), (7), (1), (8), (4), (4);  
Query OK, 8 rows affected (0.01 sec)  
Records: 8 Duplicates: 0 Warnings: 0
```

この結果、4 つのテーブルに次のデータが含まれます。

```
mysql> SELECT * FROM test1;  
+-----+  
| a1 |  
+-----+  
| 1 |  
| 3 |  
| 1 |  
| 7 |  
| 1 |  
| 8 |  
| 4 |  
| 4 |  
+-----+  
8 rows in set (0.00 sec)  
  
mysql> SELECT * FROM test2;  
+-----+  
| a2 |  
+-----+  
| 1 |  
| 3 |  
| 1 |  
| 7 |  
| 1 |  
| 8 |  
| 4 |  
| 4 |  
+-----+  
8 rows in set (0.00 sec)  
  
mysql> SELECT * FROM test3;  
+-----+  
| a3 |  
+-----+  
| 2 |  
| 5 |  
| 6 |  
| 9 |  
| 10 |  
+-----+  
5 rows in set (0.00 sec)  
  
mysql> SELECT * FROM test4;  
+-----+  
| a4 | b4 |  
+-----+  
| 1 | 3 |  
| 2 | 0 |  
| 3 | 1 |  
| 4 | 2 |  
| 5 | 0 |  
| 6 | 0 |  
| 7 | 1 |  
| 8 | 1 |  
| 9 | 0 |  
| 10 | 0 |  
+-----+  
10 rows in set (0.00 sec)
```

25.3.2 トリガーのメタデータ

トリガーに関するメタデータを取得するには:

- [INFORMATION_SCHEMA](#) データベースの [TRIGGERS](#) テーブルをクエリーします。 [セクション 26.45「INFORMATION_SCHEMA TRIGGERS テーブル」](#) を参照してください。
- [SHOW CREATE TRIGGER](#) ステートメントを使用します。 [セクション 13.7.7.11「SHOW CREATE TRIGGER ステートメント」](#) を参照してください。
- [SHOW TRIGGERS](#) ステートメントを使用します。 [セクション 13.7.7.40「SHOW TRIGGERS ステートメント」](#) を参照してください。

25.4 イベントスケジューラの使用

MySQL イベントスケジューラは、イベント、つまりスケジュールに従って実行するタスクのスケジュール設定および実行を管理します。次の説明では、イベントスケジューラを取り上げ、次のセクションに分かれています。

- [セクション 25.4.1「イベントスケジューラの概要」](#) では、MySQL イベントの概論と概念的概要を示します。
- [セクション 25.4.3「イベント構文」](#) では、MySQL イベントを作成、変更、および削除するための SQL ステートメントについて説明します。
- [セクション 25.4.4「イベントメタデータ」](#) では、イベントに関する情報の取得方法と、MySQL Server でのこの情報の格納方法を示します。
- [セクション 25.4.6「イベントスケジューラと MySQL 権限」](#) では、イベントを処理するために必要な権限と、実行時に権限に関してイベントが持つ派生問題について説明します。

ストアルーチンには、[mysql](#) システムデータベース内の [events](#) データディクショナリテーブルが必要です。このテーブルは、MySQL 8.0 インストール手順中に作成されます。以前のバージョンから MySQL 8.0 にアップグレードする場合は、アップグレード手順を実行して、システムデータベースが最新であることを確認してください。 [セクション 2.11「MySQL のアップグレード」](#) を参照してください。

追加のリソース

- スケジュール設定済みイベントを扱うときには、「[MySQL Event Scheduler User Forum](#)」が役立ちます。
- イベントの使用にはいくつかの制限があります。 [セクション 25.8「ストアプログラムの制約」](#) を参照してください。
- イベントのバイナリロギングは、 [セクション 25.7「ストアプログラムバイナリロギング」](#) で説明しているように行われます。

25.4.1 イベントスケジューラの概要

MySQL イベントはスケジュールに従って実行するタスクです。したがって、これらをスケジュール設定済みイベントと呼ぶことがあります。イベントの作成時には、特定の日に開始して終了し、1つ以上の定期的な間隔で実行される1つ以上の SQL ステートメントを含んだ、名前付きデータベースオブジェクトを作成します。概念的には、このことは Unix の [crontab](#) (「cron ジョブ」とも呼ばれます) や、Windows のタスクスケジューラの考え方に似ています。

この種のスケジュール設定済みのタスクは、「時間トリガー」と呼ばれる場合もあり、これらが時間の経過によってトリガーされるオブジェクトであることを示しています。これは基本的には正しいのですが、 [セクション 25.3「トリガーの使用」](#) で説明している種類のトリガーと混同しないように、イベントの用語を使用します。さらに厳密に言えば、イベントは「時間トリガー」と混同しないようにする必要があります。トリガーは、指定したテーブルで行われるイベントの特定の種類に応じて実行されるステートメントを持つデータベースオブジェクトですが、(スケジュール設定済み) イベントは、指定された時間間隔の経過に応じて実行されるステートメントを持つオブジェクトです。

SQL 標準にはイベントのスケジュール設定への対応はありませんが、ほかのデータベースシステムには先例があり、これらの実装と MySQL Server で見られる実装との間には一定の類似性が認められます。

MySQL イベントには次の主要機能およびプロパティがあります。

- MySQL では、イベントはその名前およびイベントが割り当てられているスキーマによって一意に識別されます。
- イベントは、スケジュールに従って特定のアクションを実行します。このアクションは、SQL ステートメントから構成され、必要に応じて `BEGIN ... END` ブロック内の複合ステートメントにできます (セクション13.6「複合ステートメントの構文」を参照してください)。イベントのタイミングは一度だけまたは繰り返しのどちらかです。一度だけのイベントは一度しか実行しません。繰り返しのイベントは、一定の間隔でアクションを繰り返し、イベントを繰り返すためのスケジュールに、特定の開始日時と終了日時の両方または一方を割り当てるか、どちらも割り当てないことができます。(デフォルトで、繰り返しイベントのスケジュールは作成されるとすぐに開始し、無効または削除されるまで続きます。)

繰り返しイベントがスケジュール間隔内に終了しない場合は、イベントの複数のインスタンスが同時に実行される可能性があります。これが好ましくない場合は、同時インスタンスを回避するためのメカニズムを設けてください。たとえば、`GET_LOCK()` 関数や、行またはテーブルのロックを使用できます。

- ユーザーは、これらの目的用の SQL ステートメントを使用してスケジュール設定済みイベントを作成、変更、および削除できます。構文が無効なイベント作成および変更ステートメントは失敗し、対応するエラーメッセージが表示されます。ユーザーは、実際には自身が保有していない権限を必要とするステートメントを、イベントのアクションに含めることがあります。イベントの作成または変更ステートメントは成功しますが、イベントのアクションは失敗します。詳細は、セクション25.4.6「イベントスケジューラと MySQL 権限」を参照してください。
- イベントのプロパティーの多くは、SQL ステートメントを使用して設定または変更できます。これらのプロパティーには、イベントの名前、タイミング、持続性(つまり、そのスケジュールの有効期限が切れたあとも保持されるかどうか)、ステータス(有効または無効)、実行するアクション、および割り当て先のスキーマが含まれます。セクション13.1.3「ALTER EVENT ステートメント」を参照してください。

イベントのデフォルトの定義者は、イベントが変更されていない場合は、イベントを作成したユーザーであり、変更されている場合は、定義者はそのイベントに影響する `ALTER EVENT` ステートメントを最後に発行したユーザーです。イベントが定義されているデータベースに対する `EVENT` 権限を保有するすべてのユーザーは、そのイベントを変更できます。セクション25.4.6「イベントスケジューラと MySQL 権限」を参照してください。

- イベントのアクションステートメントには、ストアルーチン内で許可されているほとんどの SQL ステートメントを含めることができます。制限については、セクション25.8「ストアプログラムの制約」を参照してください。

25.4.2 イベントスケジューラの構成

イベントは、特別なイベントスケジューラスレッドによって実行されます。イベントスケジューラと呼ぶ場合、実際にはこのスレッドを指しています。実行中、イベントスケジューラスレッドとその現在の状態は、次の説明で示すように、`PROCESS` 権限を保有するユーザーが `SHOW PROCESSLIST` の出力で確認できます。

`event_scheduler` グローバルシステム変数によって、イベントスケジューラがサーバー上で有効であり実行しているかどうかが決まります。ここで説明するように、これらの3つの値のいずれかがイベントのスケジューリングに影響します。デフォルトは `ON` です。

- `ON`: イベントスケジューラが開始され、イベントスケジューラスレッドがすべてのスケジュール設定済みイベントを実行しています。

イベントスケジューラが `ON` の場合、イベントスケジューラスレッドは、デーモンプロセスとして `SHOW PROCESSLIST` の出力に一覧表示され、その状態は次に示すように表示されます。

```
mysql> SHOW PROCESSLIST\G
***** 1. row *****
  Id: 1
  User: root
  Host: localhost
  db: NULL
  Command: Query
  Time: 0
  State: NULL
  Info: show processlist
***** 2. row *****
  Id: 2
  User: event_scheduler
  Host: localhost
  db: NULL
  Command: Daemon
```

```
Time: 3
State: Waiting for next activation
Info: NULL
2 rows in set (0.00 sec)
```

イベントスケジューラ設定は、`event_scheduler` の値を `OFF` に設定することで停止できます。

- **OFF**: イベントスケジューラは停止しています。 イベントスケジューラスレッドは実行されておらず、`SHOW PROCESSLIST` の出力に表示されておらず、スケジュール設定済みイベントが実行されていません。

イベントスケジューラが停止している場合 (`event_scheduler` が `OFF` です)、`event_scheduler` の値を `ON` に設定することで開始できます。(次の項目を参照してください。)

- **DISABLED**: この値はイベントスケジューラを動作しないようにします。 イベントスケジューラが `DISABLED` の場合、イベントスケジューラスレッドは実行していません(また、`SHOW PROCESSLIST` の出力にも表示されません)。また、イベントスケジューラの状態は実行時に変更できません。

イベントスケジューラのスレータスが `DISABLED` に設定されていない場合、(`SET` を使用して) `event_scheduler` の `ON` と `OFF` を切り替えることができます。この変数を設定するときに、`OFF` に `0` を、`ON` に `1` を使用することも可能です。したがって、`mysql` クライアントで次の 4 つのどのステートメントを使用しても、イベントスケジューラをオンにできます。

```
SET GLOBAL event_scheduler = ON;
SET @@GLOBAL.event_scheduler = ON;
SET GLOBAL event_scheduler = 1;
SET @@GLOBAL.event_scheduler = 1;
```

同様に、次の 4 つのどのステートメントを使用してもイベントスケジューラをオフにできます。

```
SET GLOBAL event_scheduler = OFF;
SET @@GLOBAL.event_scheduler = OFF;
SET GLOBAL event_scheduler = 0;
SET @@GLOBAL.event_scheduler = 0;
```

`ON` と `OFF` には対応する数値がありますが、`SELECT` または `SHOW VARIABLES` によって `event_scheduler` に対して表示される値は、常に `OFF`、`ON`、または `DISABLED` のいずれかになります。`DISABLED` に対応する数値はありません。このため、この変数を設定するときに、`ON` と `OFF` は通常 `1` と `0` よりも優先されます。

グローバル変数として指定しないで `event_scheduler` を設定しようとすると、エラーが発生します。

```
mysql> SET @@event_scheduler = OFF;
ERROR 1229 (HY000): Variable 'event_scheduler' is a GLOBAL
variable and should be set with SET GLOBAL
```

重要

イベントスケジューラを `DISABLED` に設定できるのは、サーバーの起動時だけです。`event_scheduler` が `ON` または `OFF` の場合、実行時にこれを `DISABLED` には設定できません。また、イベントスケジューラが起動時に `DISABLED` に設定されている場合、実行時に `event_scheduler` の値に変更できません。

イベントスケジューラを無効にするには、次の 2 つの方法のいずれかを使用します。

- サーバーの起動時のコマンド行オプションとして

```
--event-scheduler=DISABLED
```

- サーバー構成ファイル (`my.cnf` または Windows システム上の `my.ini`) に、サーバーが読み取ることができる行を含めます (`[mysqld]` セクションなど):

```
event_scheduler=DISABLED
```

イベントスケジューラを有効にするには、必要に応じて、`--event-scheduler=DISABLED` コマンド行オプションを使用しないでサーバーを再起動するか、サーバー構成ファイルの `event_scheduler=DISABLED` を含む行を削除するかコメントアウトしたあとでサーバーを再起動します。または、サーバーの起動時に `DISABLED` 値の代わりに `ON` (または `1`) が `OFF` (または `0`) を使用できます。

注記

`event_scheduler` が `DISABLED` に設定されている場合、イベント操作ステートメントを発行できません。このような場合には警告もエラーも生成されません (ステートメント自体が有効であるとして)。ただし、この変数を `ON` (または `1`) に設定するまで、スケジュール設定済みイベントは実行できません。これが行われると、イベントスケジューラスレッドは、スケジュール設定条件が満たされているすべてのイベントを実行します。

`--skip-grant-tables` オプションを使用して MySQL Server を起動すると、`event_scheduler` が `DISABLED` に設定され、コマンド行や `my.cnf` または `my.ini` ファイルで設定されたほかのすべての値をオーバーライドします (Bug #26807)。

イベントの作成、変更、または削除に使用される SQL ステートメントについては、[セクション25.4.3「イベント構文」](#)を参照してください。

MySQL は、`INFORMATION_SCHEMA` データベースに `EVENTS` テーブルを提供します。このテーブルは、サーバー上で定義されているスケジュール設定済みイベントに関する情報を取得するためにクエリーできます。詳細は、[セクション25.4.4「イベントメタデータ」](#) および [セクション26.14「INFORMATION_SCHEMA EVENTS テーブル」](#)を参照してください。

イベントスケジュール設定と MySQL 権限システムに関する情報については、[セクション25.4.6「イベントスケジューラと MySQL 権限」](#)を参照してください。

25.4.3 イベント構文

MySQL には、スケジュールされたイベントを操作するための複数の SQL ステートメントが用意されています:

- 新しいイベントは `CREATE EVENT` ステートメントを使用して定義されます。 [セクション13.1.13「CREATE EVENT ステートメント」](#)を参照してください。
- 既存のイベントの定義は、`ALTER EVENT` ステートメントによって変更できます。 [セクション13.1.3「ALTER EVENT ステートメント」](#)を参照してください。
- スケジュール設定済みイベントが不要になった場合は、その定義者が、`DROP EVENT` ステートメントを使用してサーバーから削除できます。 [セクション13.1.25「DROP EVENT ステートメント」](#)を参照してください。 イベントがそのスケジュールの終了を過ぎても持続されるかどうかは、`ON COMPLETION` 句がある場合、この句によって決まります。 [セクション13.1.13「CREATE EVENT ステートメント」](#)を参照してください。

イベントが定義されているデータベースに対する `EVENT` 権限を保有するすべてのユーザーは、そのイベントをドロップできます。 [セクション25.4.6「イベントスケジューラと MySQL 権限」](#)を参照してください。

25.4.4 イベントメタデータ

イベントに関するメタデータを取得するには:

- `INFORMATION_SCHEMA` データベースの `EVENTS` テーブルをクエリーします。 [セクション26.14「INFORMATION_SCHEMA EVENTS テーブル」](#)を参照してください。
- `SHOW CREATE EVENT` ステートメントを使用します。 [セクション13.7.7.7「SHOW CREATE EVENT ステートメント」](#)を参照してください。
- `SHOW EVENTS` ステートメントを使用します。 [セクション13.7.7.18「SHOW EVENTS ステートメント」](#)を参照してください。

イベントスケジューラの時間表現

MySQL の各セッションには、セッションタイムゾーン (STZ) があります。これは、セッションの開始時にサーバーの `time_zone` グローバル値から初期化される `time_zone` セッション値ですが、セッション中に変更される可能性があります。

`CREATE EVENT` または `ALTER EVENT` ステートメントが実行するときに使用されているセッションタイムゾーンが、イベント定義で指定されている時間の解釈に使用されます。これがイベントタイムゾーン (ETZ) になります。つまり、イベントのスケジュール設定に使用され、イベントが実行するときにそのイベント内で有効になるタイムゾーンになります。

データディクショナリでのイベント情報の表現では、`execute_at`、`starts` および `ends` の時間は UTC に変換され、イベントタイムゾーンとともに格納されます。これにより、サーバータイムゾーンまたはサマータイムの影響に対し生じた変更とは無関係に、定義されたとおりにイベントの実行を処理できます。`last_executed` 時間も UTC で格納されます。

イベント時間は、`INFORMATION_SCHEMA.EVENTS` テーブルまたは `SHOW EVENTS` から選択することで取得できますが、ETZ または STZ 値としてレポートされます。次の表は、イベント時間の表現をまとめています。

値	INFORMATION_SCHEMA.EVENTS	SHOW EVENTS
Execute at	ETZ	ETZ
Starts	ETZ	ETZ
Ends	ETZ	ETZ
Last executed	ETZ	該当なし
Created	STZ	該当なし
Last altered	STZ	該当なし

25.4.5 イベントスケジューラのステータス

イベントスケジューラは、エラーまたは警告で終了したイベント実行に関する情報を、MySQL Server のエラーログに書き込みます。例については [セクション25.4.6「イベントスケジューラと MySQL 権限」](#) を参照してください。

デバッグおよびトラブルシューティングのためにイベントスケジューラの状態に関する状態を取得するには、`mysqladmin debug` を実行します ([セクション4.5.2「mysqladmin — A MySQL Server 管理プログラム」](#) を参照してください)。このコマンドの実行後に、ここに示すようなイベントスケジューラに関連した出力がサーバーのエラーログに含まれます。

```
Events status:
LLA = Last Locked At  LUA = Last Unlocked At
WOC = Waiting On Condition  DL = Data Locked

Event scheduler status:
State      : INITIALIZED
Thread id  : 0
LLA       : n/a:0
LUA       : n/a:0
WOC       : NO
Workers   : 0
Executed   : 0
Data locked: NO

Event queue status:
Element count : 0
Data locked   : NO
Attempting lock : NO
LLA          : init_queue:95
LUA          : init_queue:103
WOC          : NO
Next activation : never
```

イベントスケジューラが実行するイベントの一部として生じるステートメント内で、診断メッセージ (エラーだけでなく警告も) がエラーログと、Windows ではアプリケーションイベントログに書き込まれます。頻繁に実行するイベントの場合、これにより、多数のメッセージが記録される結果になることがあります。たとえば、`SELECT ... INTO var_list` ステートメントの場合、クエリーが行を返さなければ、エラーコード 1329 で警告が発生し (`No data`)、変数値は変更されないままになります。クエリーが複数の行を返す場合は、エラー 1172 が発生します (`結果が 2 行以上です`)。どちらの条件についても、条件ハンドラを宣言すると、警告を記録させないようにできます。[セクション13.6.7.2「DECLARE ... HANDLER ステートメント」](#) を参照してください。複数の行を取得できるステートメントの場合、`LIMIT 1` を使用して結果セットを単一行に制限するという別の方法があります。

25.4.6 イベントスケジューラと MySQL 権限

スケジュール設定済みイベントの実行を有効または無効にするには、`event_scheduler` グローバルシステム変数の値を設定する必要があります。これには、グローバルシステム変数を設定するのに十分な権限が必要です。[セクション5.1.9.1「システム変数権限」](#) を参照してください。

EVENT 権限は、イベントの作成、変更、および削除を制御します。この権限は、**GRANT** を使用して与えることができます。たとえば、次の **GRANT** ステートメントは、**myschema** という名前のスキーマに対する **EVENT** 権限を、ユーザー **jon@ghidora** に与えます。

```
GRANT EVENT ON myschema.* TO jon@ghidora;
```

(このユーザーアカウントがすでに存在していることと、その他の点では変更されないままであると想定しています。)

この同じユーザーにすべてのスキーマに対する **EVENT** 権限を認めるには、次のステートメントを使用します。

```
GRANT EVENT ON *.* TO jon@ghidora;
```

EVENT 権限にはグローバルまたはスキーマレベルのスコープがあります。このため、単一のテーブルに対してこれを与えようとすると、次のようなエラーが生じます。

```
mysql> GRANT EVENT ON myschema.mytable TO jon@ghidora;
ERROR 1144 (42000): Illegal GRANT/REVOKE command; please
consult the manual to see which privileges can be used
```

イベントはその定義者の権限で実行され、定義者が必須の権限を保有していないアクションは実行できません。たとえば、**jon@ghidora** が **myschema** に対する **EVENT** 権限を保有しているとします。また、このユーザーは **myschema** に対する **SELECT** 権限は保有しているが、このスキーマに対するほかの権限は保有していないとします。**jon@ghidora** は、次のような新しいイベントを作成できます。

```
CREATE EVENT e_store_ts
ON SCHEDULE
EVERY 10 SECOND
DO
INSERT INTO myschema.mytable VALUES (UNIX_TIMESTAMP());
```

このユーザーは 1 分ほど待機したあと、テーブルに複数の新しい行が表示されることを予想して **SELECT * FROM mytable**; クエリーを実行します。実際は、テーブルは空です。ユーザーは該当するテーブルに対する **INSERT** 権限がないので、イベントの効果はありませんでした。

MySQL エラーログ (**hostname.err**) を調べると、イベントが実行中であることがわかりますが、実行しようとしているアクションは失敗します:

```
2013-09-24T12:41:31.261992Z 25 [ERROR] Event Scheduler:
[jon@ghidora][cookbook.e_store_ts] INSERT command denied to user
'jon'@'ghidora' for table 'mytable'
2013-09-24T12:41:31.262022Z 25 [Note] Event Scheduler:
[jon@ghidora].[myschema.e_store_ts] event execution failed.
2013-09-24T12:41:41.271796Z 26 [ERROR] Event Scheduler:
[jon@ghidora][cookbook.e_store_ts] INSERT command denied to user
'jon'@'ghidora' for table 'mytable'
2013-09-24T12:41:41.272761Z 26 [Note] Event Scheduler:
[jon@ghidora].[myschema.e_store_ts] event execution failed.
```

このユーザーは、エラーログにアクセスできない可能性が非常に高いので、直接それを実行することによって、イベントのアクションステートメントが有効であるかどうか検証できます。

```
mysql> INSERT INTO myschema.mytable VALUES (UNIX_TIMESTAMP());
ERROR 1142 (42000): INSERT command denied to user
'jon'@'ghidora' for table 'mytable'
```

INFORMATION_SCHEMA.EVENTS テーブルを調べることによって、**e_store_ts** が存在し有効になっているが、その **LAST_EXECUTED** カラムが **NULL** になっていることがわかります。

```
mysql> SELECT * FROM INFORMATION_SCHEMA.EVENTS
> WHERE EVENT_NAME='e_store_ts'
> AND EVENT_SCHEMA='myschema'G
***** 1. row *****
EVENT_CATALOG: NULL
EVENT_SCHEMA: myschema
EVENT_NAME: e_store_ts
DEFINER: jon@ghidora
EVENT_BODY: SQL
EVENT_DEFINITION: INSERT INTO myschema.mytable VALUES (UNIX_TIMESTAMP())
EVENT_TYPE: RECURRING
```

```
EXECUTE_AT: NULL
INTERVAL_VALUE: 5
INTERVAL_FIELD: SECOND
SQL_MODE: NULL
STARTS: 0000-00-00 00:00:00
ENDS: 0000-00-00 00:00:00
STATUS: ENABLED
ON_COMPLETION: NOT PRESERVE
CREATED: 2006-02-09 22:36:06
LAST_ALTERED: 2006-02-09 22:36:06
LAST_EXECUTED: NULL
EVENT_COMMENT:
1 row in set (0.00 sec)
```

EVENT 権限を取り消すには、**REVOKE** ステートメントを使用します。この例では、スキーマ **myschema** に対する **EVENT** 権限が **jon@ghidora** ユーザーアカウントから削除されます。

```
REVOKE EVENT ON myschema.* FROM jon@ghidora;
```

重要

ユーザーから **EVENT** 権限を取り消しても、そのユーザーが作成したイベントが削除されたり無効にされたりすることはありません。

作成したユーザーの名前を変更したり削除したりしても、イベントが移行または削除されることはありません。

ユーザー **jon@ghidora** に、**myschema** スキーマに対する **EVENT** および **INSERT** 権限が与えられているとします。続いてこのユーザーが次のイベントを作成します。

```
CREATE EVENT e_insert
ON SCHEDULE
EVERY 7 SECOND
DO
INSERT INTO myschema.mytable;
```

このイベントの作成後、**root** は **jon@ghidora** の **EVENT** 権限を取り消します。ただし、**e_insert** は実行し続け、7秒ごとに新しい行が **mytable** に挿入されます。**root** が次のどちらかのステートメントを発行した場合も、同じことが当てはまります。

- **DROP USER jon@ghidora;**
- **RENAME USER jon@ghidora TO someotherguy@ghidora;**

これが正しいことを確認するには、**DROP USER** または **RENAME USER** ステートメントの発行前後に **INFORMATION_SCHEMA.EVENTS** テーブル (セクション26.14「**INFORMATION_SCHEMA EVENTS テーブル**」を参照) を調べます。

イベント定義はデータディクショナリに格納されます。別のユーザーアカウントによって作成されたイベントを削除するには、MySQL **root** ユーザーまたは必要な権限を持つ別のユーザーである必要があります。

ユーザーの **EVENT** 権限は、**mysql.user** および **mysql.db** テーブルの **Event_priv** カラムに格納されています。どちらの場合でも、このカラムには、「**Y**」または「**N**」のどちらかの値が含まれています。「**N**」がデフォルトです。指定されたユーザーがグローバルな **EVENT** 権限を保有している場合 (つまり、**GRANT EVENT ON *.*** を使用して権限が与えられた場合) にのみ、そのユーザーの **mysql.user.Event_priv** は「**Y**」に設定されます。スキーマレベルの **EVENT** 権限の場合、**GRANT** は、**mysql.db** に行を作成し、その行の **Db** カラムをスキーマの名前に、**User** カラムをユーザーの名前に、**Event_priv** カラムを「**Y**」に設定します。**GRANT EVENT** および **REVOKE EVENT** ステートメントがこれらのテーブルでの必要な操作を実行するので、これらのテーブルを直接操作する必要はありません。

5つのステータス変数が、イベント関連操作のカウンタを提供します (ただし、イベントが実行するステートメントのカウンタは提供しません。セクション25.8「**ストアドプログラムの制約**」を参照してください)。これらを次に示します。

- **Com_create_event:** サーバーが最後に再起動してから実行された **CREATE EVENT** ステートメントの数。
- **Com_alter_event:** サーバーが最後に再起動してから実行された **ALTER EVENT** ステートメントの数。

- `Com_drop_event`: サーバーが最後に再起動してから実行された `DROP EVENT` ステートメントの数。
- `Com_show_create_event`: サーバーが最後に再起動してから実行された `SHOW CREATE EVENT` ステートメントの数。
- `Com_show_events`: サーバーが最後に再起動してから実行された `SHOW EVENTS` ステートメントの数。

ステートメント `SHOW STATUS LIKE '%event%'`; を実行すると、これらのすべての現在値を一度に表示できます。

25.5 ビューの使用

MySQL では、更新可能なビューを含むビューがサポートされています。ビューは、呼び出されたときに結果セットを生成するストアドクエリーです。ビューは仮想テーブルとして機能します。

次の説明では、ビューを作成し削除するための構文について記述し、それらの使用法の例をいくつか示します。

追加のリソース

- ビューを扱うときには、「[Views User Forum](#)」が役立ちます。
- MySQL のビューに関するよくある質問とその回答については、[セクション A.6 「MySQL 8.0 FAQ: ビュー」](#) を参照してください。
- ビューの使用にはいくつかの制限があります。[セクション 25.9 「ビューの制約」](#) を参照してください。

25.5.1 ビューの構文

`CREATE VIEW` ステートメントは新しいビューを作成します ([セクション 13.1.23 「CREATE VIEW ステートメント」](#) を参照してください)。ビューの定義を変更したり、ビューを削除したりするには、`ALTER VIEW` ([セクション 13.1.11 「ALTER VIEW ステートメント」](#) を参照してください) または `DROP VIEW` ([セクション 13.1.35 「DROP VIEW ステートメント」](#) を参照してください) を使用します。

ビューは、多くの種類の `SELECT` ステートメントから作成できます。ベーステーブルまたはほかのビューを参照できます。結合、`UNION`、およびサブクエリーを使用できます。`SELECT` がテーブルをまったく参照しなくてもかまいません。次の例では、別のテーブルからの 2 つのカラムに加え、それらのカラムから計算される式を選択するビューを定義しています。

```
mysql> CREATE TABLE t (qty INT, price INT);
mysql> INSERT INTO t VALUES(3, 50), (5, 60);
mysql> CREATE VIEW v AS SELECT qty, price, qty*price AS value FROM t;
mysql> SELECT * FROM v;
+----+-----+-----+
| qty | price | value |
+----+-----+-----+
| 3 | 50 | 150 |
| 5 | 60 | 300 |
+----+-----+-----+
mysql> SELECT * FROM v WHERE qty = 5;
+----+-----+-----+
| qty | price | value |
+----+-----+-----+
| 5 | 60 | 300 |
+----+-----+-----+
```

25.5.2 ビュー処理アルゴリズム

`CREATE VIEW` または `ALTER VIEW` のオプションの `ALGORITHM` 句は、標準 SQL に対する MySQL 拡張です。これは、MySQL によるビューの処理方法に影響を与えます。`ALGORITHM` は、`MERGE`、`TEMPTABLE`、または `UNDEFINED` の 3 つの値を受け取ります。

- `MERGE` の場合、ビューを参照するステートメントのテキストとビュー定義がマージされ、ビュー定義の部分が対応するステートメントの部分と置き換えられます。
- `TEMPTABLE` の場合、ビューの結果が一時テーブル内に取得され、その後、ステートメントを実行するために使用されます。

- **UNDEFINED** の場合、MySQL は使用するアルゴリズムを選択します。できるかぎり **TEMPTABLE** より **MERGE** が優先されます。これは通常、**MERGE** のほうが効率性が高く、一時テーブルを使用するとビューを更新できなくなるためです。
- **ALGORITHM** 句が存在しない場合、デフォルトのアルゴリズムは `optimizer_switch` システム変数の `derived_merge` フラグの値によって決定されます。詳細は、[セクション8.2.2.4「マージまたは実体化を使用した導出テーブル、ビュー参照および共通テーブル式の最適化」](#)を参照してください。

TEMPTABLE を明示的に指定する理由は、一時テーブルの作成後、ステートメントの処理を終了するために使用する前に、基礎となるテーブルでロックを解放できるためです。その結果、**MERGE** アルゴリズムよりもすみやかにロックが解除され、ビューを使用するほかのクライアントが長時間ブロックされることがなくなります。

次の3つの理由によって、ビューアルゴリズムを**UNDEFINED**にできます。

- **CREATE VIEW** ステートメントの中に **ALGORITHM** 句が存在しない。
- **CREATE VIEW** ステートメントに明示的な **ALGORITHM = UNDEFINED** 句が含まれている。
- 一時テーブルだけでしか処理できないビューに対して、**ALGORITHM = MERGE** が指定されている。この場合、MySQL は警告を発生し、アルゴリズムを **UNDEFINED** に設定します。

前述のように、**MERGE** は、ビュー定義の対応する部分を、ビューを参照するステートメントにマージして処理されます。次の例で、**MERGE** アルゴリズムの動作について簡単に説明します。例では、次の定義を含むビュー `v_merge` が存在していると想定します。

```
CREATE ALGORITHM = MERGE VIEW v_merge (vc1, vc2) AS
SELECT c1, c2 FROM t WHERE c3 > 100;
```

例 1: 次のステートメントを発行するとします。

```
SELECT * FROM v_merge;
```

MySQL は次のようにステートメントを処理します。

- `v_merge` は `t` になる
- `*` は `vc1, vc2` となり、`c1, c2` と一致する
- ビュー **WHERE** 句が追加される

結果が実行されるステートメントは次のようになります。

```
SELECT c1, c2 FROM t WHERE c3 > 100;
```

例 2: 次のステートメントを発行するとします。

```
SELECT * FROM v_merge WHERE vc1 < 100;
```

このステートメントは、前述のステートメントと同様に処理されますが、`vc1 < 100` が `c1 < 100` になり、**AND** 連結詞を使用してビュー **WHERE** 句がステートメント **WHERE** 句に追加される点が異なります (また、句の一部が確実に正しい優先順位で実行されるように、かっこが追加されます)。結果が実行されるステートメントは次のようになります。

```
SELECT c1, c2 FROM t WHERE (c3 > 100) AND (c1 < 100);
```

事実上、実行されるステートメントには、次の形式の **WHERE** 句が含まれます。

```
WHERE (select WHERE) AND (view WHERE)
```

MERGE アルゴリズムを使用できない場合、一時テーブルを代わりに使用する必要があります。マージを防止する構成要素は、導出テーブルおよび共通テーブル式でのマージを防止する構成要素と同じです。例として、サブクエリー内の **SELECT DISTINCT** または **LIMIT** があります。詳細は、[セクション8.2.2.4「マージまたは実体化を使用した導出テーブル、ビュー参照および共通テーブル式の最適化」](#)を参照してください。

25.5.3 更新可能および挿入可能なビュー

一部のビューは更新可能であり、それらへの参照を使用して、データ変更ステートメントで更新するテーブルを指定できます。つまり、これらのビューを `UPDATE`、`DELETE`、`INSERT` などのステートメントで使用して、ベースとなるテーブルの内容を更新できます。導出テーブルおよび共通テーブル式は、複数テーブルの `UPDATE` ステートメントおよび `DELETE` ステートメントでも指定できますが、更新または削除する行を指定するためのデータの読取りにのみ使用できます。通常、ビュー参照は更新可能である必要があります。つまり、マージされて実体化されない場合があります。コンポジットビューには、より複雑なルールがあります。

ビューが更新可能であるためには、そのビュー内の行とベースとなるテーブル内の行の間に 1 対 1 の関係が存在する必要があります。また、ビューを更新不可能にするその他の特定の構造構文も存在します。より具体的には、次のいずれかを含む場合、ビューは更新可能ではありません。

- 集計関数またはウィンドウ関数 (`SUM()`、`MIN()`、`MAX()`、`COUNT()` など)

- `DISTINCT`

- `GROUP BY`

- `HAVING`

- `UNION` または `UNION ALL`

- 選択リスト内のサブクエリー

選択リスト内の非依存サブクエリーは、`INSERT` では失敗しますが、`UPDATE`、`DELETE` では問題ありません。選択リスト内の依存サブクエリーの場合、データ変更ステートメントは許可されません。

- 特定の結合 (このセクションで後述する結合に関する追加説明を参照してください)

- `FROM` 句内の更新不可ビューへの参照

- `FROM` 句のテーブルを参照する `WHERE` 句のサブクエリー

- リテラル値だけの参照 (この場合、更新するベースとなるテーブルがありません)

- `ALGORITHM = TEMPTABLE` (一時テーブルを使用すると、常にビューが更新不可になります)

- 実テーブルの任意のカラムへの複数の参照 (`INSERT` では失敗、`UPDATE`、`DELETE` では OK)

ビュー内の生成されたカラムは、割り当て可能であるため、更新可能とみなされます。ただし、このようなカラムが明示的に更新される場合、許可される値は `DEFAULT` のみです。生成されるカラムの詳細は、[セクション 13.1.20.8「CREATE TABLE および生成されるカラム」](#) を参照してください。

`MERGE` アルゴリズムで処理できるとすれば、複数テーブルビューが更新できる可能性があります。これを実現するには、ビューで (外部結合または `UNION` ではなく) 内部結合を使用する必要があります。また、ビュー定義内の単一のテーブルだけを更新できるので、`SET` 句は、ビュー内のいずれかのテーブルのカラムだけを指名する必要があります。理論的には更新可能であっても、`UNION ALL` を使用するビューは許可されません。

挿入可能性 (`INSERT` ステートメントで更新可能であること) については、更新可能なビューがビューカラムに対する次の追加要件も満たしている場合に挿入可能になります。

- 重複したビューカラム名が存在しないようにする必要があります。

- ビューには、デフォルト値を持たない、ベーステーブル内のすべてのカラムが含まれている必要があります。

- ビューカラムは単純なカラム参照である必要があります。次のような式は使用できません:

```
3.14159
col1 + 3
UPPER(col2)
col3 / col4
(subquery)
```

MySQL は、`CREATE VIEW` 時に、ビューの更新可能性フラグというフラグを設定します。`UPDATE` および `DELETE` (および同様の操作) がビューで有効な場合、フラグは `YES` (true) に設定されます。それ以外の場合、フラグは `NO` (false) に設定されます。`INFORMATION_SCHEMA.VIEWS` テーブルの `IS_UPDATABLE` カラムは、このフラグのス

ステータスを表示します。これは、ビューが更新可能であるかどうかをサーバーが常に把握していることを意味します。

ビューが更新可能でない場合、**UPDATE**、**DELETE** および **INSERT** などのステートメントは無効であり、拒否されます。(ビューが更新可能であっても、このセクションの別の場所で説明されているように、ビューに挿入できない場合があります。)

ビューを更新できるかどうかは、`updatable_views_with_limit` システム変数の値に影響されます。 [セクション 5.1.8「サーバーシステム変数」](#) を参照してください。

次の説明では、これらのテーブルおよびビューが存在するとします:

```
CREATE TABLE t1 (x INTEGER);
CREATE TABLE t2 (c INTEGER);
CREATE VIEW vmat AS SELECT SUM(x) AS s FROM t1;
CREATE VIEW vup AS SELECT * FROM t2;
CREATE VIEW vjoin AS SELECT * FROM vmat JOIN vup ON vmat.s=vup.c;
```

INSERT、**UPDATE** および **DELETE** ステートメントは、次のように許可されます:

- **INSERT:** **INSERT** ステートメントの挿入テーブルは、マージされるビュー参照である場合があります。ビューが結合ビューの場合、ビューのすべてのコンポーネントは更新可能である必要があります(実体化されていません)。更新可能な複数テーブルビューでは、**INSERT** は、単一のテーブルに挿入する場合に機能します。

結合ビューのいずれかのコンポーネントが更新不可であるため、このステートメントは無効です:

```
INSERT INTO vjoin (c) VALUES (1);
```

このステートメントは有効です。ビューには実体化コンポーネントが含まれていません:

```
INSERT INTO vup (c) VALUES (1);
```

- **UPDATE:** **UPDATE** ステートメントで更新されるテーブルは、マージされるビュー参照である場合があります。ビューが結合ビューの場合、ビューの少なくとも 1 つのコンポーネントが更新可能である必要があります (**INSERT** とは異なります)。

複数テーブルの **UPDATE** ステートメントでは、ステートメントの更新されたテーブル参照は実テーブルまたは更新可能なビュー参照である必要があります。更新されていないテーブル参照は、実体化ビューまたは導出テーブルです。

次のステートメントは有効です。カラム `c` は結合ビューの更新可能な部分のものです:

```
UPDATE vjoin SET c=c+1;
```

このステートメントは無効です。カラム `x` は更新不可能な部分のものです:

```
UPDATE vjoin SET x=x+1;
```

次のステートメントは有効です。複数テーブルの **UPDATE** の更新されたテーブル参照は更新可能なビュー (`vup`) です:

```
UPDATE vup JOIN (SELECT SUM(x) AS s FROM t1) AS dt ON ...
SET c=c+1;
```

このステートメントは無効です。実体化導出テーブルを更新しようとしています:

```
UPDATE vup JOIN (SELECT SUM(x) AS s FROM t1) AS dt ON ...
SET s=s+1;
```

- **DELETE:** **DELETE** ステートメントで削除するテーブルは、マージビューである必要があります。結合ビューは使用できません (**INSERT** および **UPDATE** とは異なります)。

ビューが結合ビューであるため、このステートメントは無効です:

```
DELETE vjoin WHERE ...;
```

このステートメントは、ビューがマージ (更新可能) ビューであるため有効です:


```
DELETE vup WHERE ...;
```

このステートメントは、マージされた (更新可能な) ビューから削除されるため有効です:

```
DELETE vup FROM vup JOIN (SELECT SUM(x) AS s FROM t1) AS dt ON ...;
```

その他の説明と例を次に示します。

このセクションの以前の説明では、すべてのカラムが単純なカラム参照ではない場合 (たとえば、式または複合式であるカラムが含まれている場合)、ビューを挿入できないことを指摘していました。このようなビューは挿入できませんが、式ではないカラムのみを更新すると更新できます。次のビューを考えてみてください。

```
CREATE VIEW v AS SELECT col1, 1 AS col2 FROM t;
```

`col2` が式であるため、このビューは挿入できません。ただし、更新で `col2` を更新しようとしていない場合は、更新可能になります。次の更新は許可されます。

```
UPDATE v SET col1 = 0;
```

式カラムを更新しようとしているため、この更新は許可されません:

```
UPDATE v SET col2 = 0;
```

テーブルに `AUTO_INCREMENT` カラムが含まれている場合、`AUTO_INCREMENT` カラムが含まれていないテーブル上の挿入可能なビューに挿入すると、`LAST_INSERT_ID()` の値を変更しません。これは、ビューの一部ではないカラムにデフォルト値を挿入した副作用が現れないようにするためです。

25.5.4 WITH CHECK OPTION 句の表示

更新可能なビューに `WITH CHECK OPTION` 句を指定して、`select_statement` の `WHERE` 句が `true` でない行への挿入を防止できます。また、`WHERE` 句が `true` であるが、更新によって `true` にならない行の更新も防止されます (つまり、表示可能な行が非表示の行に更新されないようにします)。

更新可能なビューに対する `WITH CHECK OPTION` 句では、そのビューが別のビューとの関連で定義されている場合、`LOCAL` および `CASCADED` キーワードによってチェックテストの範囲が決定されます。どちらのキーワードも指定されていない場合、デフォルトは `CASCADED` になります。

`WITH CHECK OPTION` テストは標準に準拠しています:

- `LOCAL` では、ビューの `WHERE` 句がチェックされ、基礎となるビューに対してチェックが繰り返され、同じルールが適用されます。
- `CASCADED` では、ビューの `WHERE` 句がチェックされ、基礎となるビューに対してチェックが繰り返され、`WITH CASCADED CHECK OPTION` が追加されて (チェックの目的で、定義は変更されません)、同じルールが適用されます。
- チェックオプションを指定しない場合、ビューの `WHERE` 句はチェックされず、基礎となるビューに対してチェックが繰り返され、同じルールが適用されます。

次のテーブルと一連のビューの定義を考えてみてください。

```
CREATE TABLE t1 (a INT);
CREATE VIEW v1 AS SELECT * FROM t1 WHERE a < 2
WITH CHECK OPTION;
CREATE VIEW v2 AS SELECT * FROM v1 WHERE a > 0
WITH LOCAL CHECK OPTION;
CREATE VIEW v3 AS SELECT * FROM v1 WHERE a > 0
WITH CASCADED CHECK OPTION;
```

ここで、`v2` ビューと `v3` ビューは、別のビュー `v1` で定義されています。

`v2` の挿入が `LOCAL` チェックオプションに対してチェックされ、チェックが `v1` に対して繰り返され、ルールが再度適用されます。 `v1` のルールによってチェックが失敗します。 `v3` のチェックも失敗します:

```
mysql> INSERT INTO v2 VALUES (2);
ERROR 1369 (HY000): CHECK OPTION failed 'test.v2'
mysql> INSERT INTO v3 VALUES (2);
```

```
ERROR 1369 (HY000): CHECK OPTION failed 'test.v3'
```

25.5.5 ビューのメタデータ

ビューに関するメタデータを取得するには:

- `INFORMATION_SCHEMA` データベースの `VIEWS` テーブルをクエリーします。 [セクション 26.48 「INFORMATION_SCHEMA VIEWS テーブル」](#) を参照してください。
- `SHOW CREATE VIEW` ステートメントを使用します。 [セクション 13.7.7.13 「SHOW CREATE VIEW ステートメント」](#) を参照してください。

25.6 ストアドオブジェクトのアクセス制御

ストアドプログラム (プロシージャ、ファンクション、トリガーおよびイベント) およびビューは、使用前に定義され、参照されると、権限を決定するセキュリティコンテキスト内で実行されます。ストアドオブジェクトの実行に適用可能な権限は、その `DEFINER` 属性および `SQL SECURITY` 特性によって制御されます。

- [DEFINER 属性](#)
- [SQL SECURITY 特性](#)
- [例](#)
- [孤立したストアドオブジェクト](#)
- [リスク最小化のガイドライン](#)

DEFINER 属性

格納されたオブジェクト定義には、MySQL アカウントを指定する `DEFINER` 属性を含めることができます。定義で `DEFINER` 属性が省略されている場合、デフォルトのオブジェクト定義者はそれを作成したユーザーです。

次のルールによって、格納されたオブジェクトの `DEFINER` 属性として指定できるアカウントが決定されます:

- `SET_USER_ID` 権限 (または非推奨の `SUPER` 権限) がある場合は、任意のアカウントを `DEFINER` 属性として指定できます。アカウントが存在しない場合は、警告が生成されます。また、ストアドオブジェクトの `DEFINER` 属性を `SYSTEM_USER` 権限を持つアカウントに設定するには、`SYSTEM_USER` 権限が必要です。
- それ以外の場合、許可されるアカウントは、文字どおり、または `CURRENT_USER` または `CURRENT_USER()` として指定された独自のアカウントのみです。定義者は他のアカウントに設定できません。

存在しない `DEFINER` アカウントでストアドオブジェクトを作成すると、孤立したオブジェクトが作成され、負の結果になる可能性があります。 [孤立したストアドオブジェクト](#) を参照してください。

SQL SECURITY 特性

ストアドルーチン (プロシージャおよびファンクション) およびビューの場合、オブジェクト定義に `DEFINER` または `INVOKER` の値を持つ `SQL SECURITY` 特性を含めて、オブジェクトが定義者コンテキストで実行されるか起動側コンテキストで実行されるかを指定できます。定義で `SQL SECURITY` 特性が省略されている場合、デフォルトは定義者コンテキストです。

トリガーとイベントには、`SQL SECURITY` 特性がなく、常に定義側のコンテキストで実行します。サーバーが必要に応じて自動的にこれらのオブジェクトを呼び出すので、呼び出し元ユーザーは存在しません。

定義側と呼び出し元のセキュリティのコンテキストは次のように異なります。

- 定義者セキュリティコンテキストで実行されるストアドオブジェクトは、その `DEFINER` 属性で指定されたアカウントの権限で実行されます。これらの権限は、呼び出し元ユーザーの権限とは完全に異なる場合があります。実行者は、オブジェクトを参照する適切な権限 (たとえば、ストアドプロシージャをコールするための `EXECUTE` またはビューから選択する `SELECT` など) を持っている必要がありますが、オブジェクトの実行中、実行者権限は無視され、`DEFINER` アカウント権限のみが関係します。 `DEFINER` アカウントに少数の権限がある場合、オブジェクトは実行可能な操作に対応して制限されます。 `DEFINER` アカウントが高い権限を持つ場合 (管理アカウントなど)、オブジェクトは強力な操作誰が呼び出すかに関係なくを実行できます。

- 呼び出し元のセキュリティコンテキストで実行するストアドルーチンまたはビューは、呼び出し元が権限を持つ操作だけを実行できます。DEFINER 属性はオブジェクトの実行には影響しません。

例

定義者セキュリティコンテキストで実行するために SQL SECURITY DEFINER で宣言されている次のストアドプロシージャについて考えてみます:

```
CREATE DEFINER = 'admin'@'localhost' PROCEDURE p1()
SQL SECURITY DEFINER
BEGIN
  UPDATE t1 SET counter = counter + 1;
END;
```

p1 に対する EXECUTE 権限を持つどのユーザーでも、CALL ステートメントを使用してこれを呼び出すことができます。ただし、p1 を実行すると、定義者セキュリティコンテキストで実行されるため、DEFINER 属性として指定されたアカウントである 'admin'@'localhost' の権限で実行されます。このアカウントには、p1 に対する EXECUTE 権限と、オブジェクト本体内で参照される t1 テーブルに対する UPDATE 権限が必要です。それ以外の場合、プロシージャは失敗します。

続いて次のストアドプロシージャを検討してください。これは p1 と同じですが、その SQL SECURITY 特性が INVOKER である点が異なります。

```
CREATE DEFINER = 'admin'@'localhost' PROCEDURE p2()
SQL SECURITY INVOKER
BEGIN
  UPDATE t1 SET counter = counter + 1;
END;
```

p1 とは異なり、p2 は起動側セキュリティコンテキストで実行されるため、DEFINER 属性値に関係なく、起動側ユーザーの権限を使用します。実行者に p2 に対する EXECUTE 権限または t1 テーブルに対する UPDATE 権限がない場合、p2 は失敗します。

孤立したストアドオブジェクト

孤立したストアドオブジェクトは、その DEFINER 属性が存在しないアカウントを指定するオブジェクトです:

- 孤立したストアドオブジェクトは、オブジェクトの作成時に存在しない DEFINER アカウントを指定することで作成できます。
- 既存のストアドオブジェクトは、オブジェクトの DEFINER アカウントを削除する DROP USER ステートメント、またはオブジェクトの DEFINER アカウントの名前を変更する RENAME USER ステートメントの実行によって孤立する可能性があります。

孤立したストアドオブジェクトには、次のような問題がある可能性があります:

- DEFINER アカウントが存在しないため、定義者セキュリティコンテキストで実行される場合、オブジェクトは予想どおりに動作しない可能性があります:
 - ストアドルーチンの場合、SQL SECURITY 値が DEFINER で定義者アカウントが存在しないと、ルーチンの実行時にエラーが発生します。
 - トリガーの場合、アカウントが実際に存在するまでトリガーのアクティブ化を行うことはお薦めしません。それ以外の権限確認に関する動作は定義されていません。
 - イベントの場合、アカウントが存在しないと、イベントの実行時にエラーが発生します。
 - ビューでは、SQL SECURITY 値が DEFINER で定義者アカウントが存在しない場合にビューが参照されると、エラーが発生します。
- その後、オブジェクトに関連しない目的で存在しない DEFINER アカウントが再作成されると、オブジェクトにセキュリティリスクが生じる可能性があります。この場合、オブジェクトと適切な権限を持つアカウント「採用」は、意図していない場合でもオブジェクトを実行できます。

MySQL 8.0.22 の時点では、サーバーは、(おそらく誤って) 格納されたオブジェクトが孤立したり、現在孤立している格納されたオブジェクトを採用する原因となる操作を防ぐために設計された追加のアカウント管理セキュリティチェックを強制します:

- 削除するアカウントが格納されたオブジェクトの **DEFINER** 属性として指定されている場合、**DROP USER** はエラーで失敗します。(つまり、アカウントを削除すると、格納されたオブジェクトが孤立する場合、ステートメントは失敗します。)
- 名前を変更するアカウントが格納されているオブジェクトの **DEFINER** 属性として指定されている場合、**RENAME USER** はエラーで失敗します。(つまり、アカウントの名前を変更すると、格納されているオブジェクトが孤立する場合、ステートメントは失敗します。)
- 作成するアカウントの名前が格納されたオブジェクトの **DEFINER** 属性として指定されている場合、**CREATE USER** はエラーで失敗します。(つまり、アカウントを作成すると、アカウントが現在孤立しているストアドオブジェクトを採用する場合、ステートメントは失敗します。)

特定の状況では、これらのアカウント管理ステートメントは、それ以外の場合でも意図せずに実行する必要がある場合があります。これを可能にするために、ユーザーが **SET_USER_ID** 権限を持っている場合、その権限は孤立したオブジェクトセキュリティチェックをオーバーライドし、ステートメントはエラーで失敗するのではなく警告で成功します。

MySQL インストールでストアドオブジェクト定義者として使用されるアカウントに関する情報を取得するには、**INFORMATION_SCHEMA** にクエリーします。

このクエリーは、**DEFINER** 属性を持つオブジェクトを記述する **INFORMATION_SCHEMA** テーブルを識別します:

```
mysql> SELECT TABLE_SCHEMA, TABLE_NAME FROM INFORMATION_SCHEMA.COLUMNS
WHERE COLUMN_NAME = 'DEFINER';
+-----+-----+
| TABLE_SCHEMA | TABLE_NAME |
+-----+-----+
| information_schema | EVENTS |
| information_schema | ROUTINES |
| information_schema | TRIGGERS |
| information_schema | VIEWS |
+-----+-----+
```

結果には、どのテーブルをクエリーして、どのストアドオブジェクト **DEFINER** 値が存在するか、およびどのオブジェクトが特定の **DEFINER** 値を持つかが示されます:

- 各テーブルに存在する **DEFINER** 値を識別するには、次のクエリーを使用します:

```
SELECT DISTINCT DEFINER FROM INFORMATION_SCHEMA.EVENTS;
SELECT DISTINCT DEFINER FROM INFORMATION_SCHEMA.ROUTINES;
SELECT DISTINCT DEFINER FROM INFORMATION_SCHEMA.TRIGGERS;
SELECT DISTINCT DEFINER FROM INFORMATION_SCHEMA.VIEWS;
```

クエリー結果は、次のように表示されるすべてのアカウントにとって重要です:

- アカウントが存在する場合、アカウントを削除または名前変更すると、格納されているオブジェクトが孤立します。アカウントを削除または名前変更する場合は、まず、関連付けられているストアドオブジェクトを削除するか、別の定義者を持つように再定義することを検討してください。
- アカウントが存在しない場合は作成すると、現在孤立している格納済オブジェクトが採用されます。アカウントを作成する場合は、孤立したオブジェクトを関連付けるかどうかを検討してください。そうでない場合は、別の定義者を持つように再定義します。

別の定義者でオブジェクトを再定義するには、**ALTER EVENT** または **ALTER VIEW** を使用して、イベントおよびビューの **DEFINER** アカウントを直接変更できます。ストアドプロシージャとストアドファンクションおよびトリガーの場合は、オブジェクトを削除して再作成し、別の **DEFINER** アカウントを割り当てる必要があります

- 特定の **DEFINER** アカウントを持つオブジェクトを識別するには、次のクエリーを使用して、対象のアカウントを **user_name@host_name** に置き換えます:

```
SELECT EVENT_SCHEMA, EVENT_NAME FROM INFORMATION_SCHEMA.EVENTS
WHERE DEFINER = 'user_name@host_name';
```

```
SELECT ROUTINE_SCHEMA, ROUTINE_NAME, ROUTINE_TYPE
FROM INFORMATION_SCHEMA.ROUTINES
WHERE DEFINER = 'user_name@host_name';
SELECT TRIGGER_SCHEMA, TRIGGER_NAME FROM INFORMATION_SCHEMA.TRIGGERS
WHERE DEFINER = 'user_name@host_name';
SELECT TABLE_SCHEMA, TABLE_NAME FROM INFORMATION_SCHEMA.VIEWS
WHERE DEFINER = 'user_name@host_name';
```

ROUTINES テーブルの場合、クエリーには **ROUTINE_TYPE** カラムが含まれるため、出力行は **DEFINER** がストアプロシージャ用かストアアドファンクション用かを区別します。

検索するアカウントが存在しない場合、それらのクエリーによって表示されるオブジェクトは孤立したオブジェクトです。

リスク最小化のガイドライン

ストアオブジェクトの作成および使用のリスクを最小限に抑えるには、次のガイドラインに従います：

- 孤立したストアオブジェクト (**DEFINER** 属性が存在しないアカウントを指定するオブジェクト) は作成しないでください。既存のオブジェクトの **DEFINER** 属性で指定されたアカウントを削除または名前変更して、格納されたオブジェクトを孤立させないでください。
- 可能な場合は、ストアルーチンまたはビューに対して、オブジェクト定義の **SQL SECURITY INVOKER** を使用して、オブジェクトが実行する操作に適したアクセス許可を持つユーザーだけが使用できるようにします。
- SET_USER_ID** 権限 (または非推奨の **SUPER** 権限) を持つアカウントの使用中に定義者コンテキストストアオブジェクトを作成する場合は、オブジェクトで実行される操作に必要な権限のみを持つアカウントを指定する明示的な **DEFINER** 属性を指定します。高い権限を持つ **DEFINER** アカウントは、絶対に必要な場合にのみ指定してください。
- 管理者は、**SET_USER_ID** 権限 (または非推奨の **SUPER** 権限) を付与しないことで、高い権限を持つ **DEFINER** アカウントを指定するストアオブジェクトをユーザーが作成できないようにできます。
- 定義側のコンテキストのオブジェクトを作成するときには、呼び出し元ユーザーに権限のないデータに定義側がアクセスできる場合があります。場合によっては、権限のないユーザーに特定の権限を付与しないことで、これらのオブジェクトへの参照を防止できます：
 - ストアルーチンに対する **EXECUTE** 権限を持たないユーザーは、ストアルーチンを参照できません。
 - ビューに対する適切な権限 (ビューから選択するための **SELECT**、ビューに挿入するための **INSERT** など) を持っていないユーザーは、ビューを参照できません。

ただし、トリガーおよびイベントは常に定義者コンテキストで実行されるため、これらのコントロールは存在しません。サーバーは必要に応じてこれらのオブジェクトを自動的に呼び出し、ユーザーはこれらを直接参照しません：

- トリガーは、トリガーが関連付けられているテーブルへのアクセスによってアクティブ化されます。特別な権限を持たないユーザーによる通常のテーブルアクセスでもアクティブ化されます。
- イベントは、スケジュールに基づいてサーバーによって実行されます。

どちらの場合も、**DEFINER** アカウントが高い権限を持つ場合、オブジェクトは機密操作または危険な操作を実行できる可能性があります。これは、オブジェクトの作成に必要な権限が、そのオブジェクトを作成したユーザーのアカウントから取り消された場合も当てはまります。管理者は、特にユーザーへのオブジェクト作成権限の付与に注意する必要があります。

25.7 ストアドプログラムバイナリロギング

バイナリログには、データベースの内容を変更する SQL ステートメントに関する情報が含まれます。この情報は、変更について記述した「イベント」の形式で格納されます。(バイナリログイベントは、スケジュールされたイベントストアオブジェクトとは異なります。) バイナリログには 2 つの重要な目的があります。

- レプリケーションの場合、バイナリログは、レプリカサーバーに送信されるステートメントのレコードとしてソースレプリケーションサーバーで使用されます。ソースは、バイナリログに含まれているイベントをそのレプリカ

に送信します。このレプリカは、それらのイベントを実行して、ソースで行われたものと同じデータ変更を行います。[セクション17.2「レプリケーションの実装」](#)を参照してください。

- ある特定のデータリカバリ操作には、バイナリログの使用が必要です。バックアップファイルがリストアされたあと、バックアップの作成後に記録されたバイナリログ内のイベントが、再度実行されます。これらのイベントは、データベースをバックアップのポイントから最新の状態で持って行きます。[セクション7.3.2「リカバリへのバックアップの使用」](#)を参照してください。

ただし、ステートメントレベルでロギングが発生した場合は、ストアドプログラム (ストアドプロシージャーとストアドファンクション、トリガー、およびイベント) に関する特定のバイナリロギングの問題があります:

- 場合によっては、ステートメントがソースとレプリカの異なる行セットに影響することがあります。
- レプリカで実行されるレプリケートされたステートメントは、完全な権限を持つレプリカ SQL スレッドによって処理されます。プロシージャは、ソースサーバーとレプリカサーバーで異なる実行パスに従うことができるため、ユーザーは、完全な権限を持つスレッドによって処理されるレプリカでのみ実行される危険なステートメントを含むルーチンを記述できます。
- データを変更するストアドプログラムが非決定的である場合、再現可能ではありません。これにより、ソースとレプリカでデータが異なる場合や、リストアされたデータが元のデータと異なる場合があります。

このセクションでは、MySQL がストアドプログラムのバイナリロギングを処理する方法について説明します。これは、実装がストアドプログラムの使用に適用する現在の状態と、ロギングの問題を回避するために実行できることを示します。また、これらの条件の理由に関する追加情報も示します。

一般に、ここで説明する問題は、バイナリロギングが SQL ステートメントレベル (ステートメントベースのバイナリロギング) で発生した場合に発生します。行ベースのバイナリロギングを使用する場合、ログには、SQL ステートメントを実行した結果として個々の行に行われた変更が含まれます。ルーチンまたはトリガーが実行されると、行の変更が記録されますが、変更を行なったステートメントは記録されません。ストアドプロシージャーの場合、これは `CALL` ステートメントが記録されないことを意味します。ストアドファンクションの場合、関数内で行われた行の変更が記録され、関数呼び出しは記録されません。トリガーの場合、トリガーによって行われた行の変更が記録されます。レプリカ側では、行の変更のみが表示され、ストアドプログラムの起動は表示されません。

混合形式のバイナリロギング (`binlog_format=MIXED`) では、ステートメントベースのバイナリロギングが使用されますが、行ベースのバイナリロギングのみが正しい結果になることが保証されている場合を除きます。混合形式では、ストアドファンクション、ストアドプロシージャー、トリガー、イベント、またはプリアドステートメントにステートメントベースのバイナリロギングに安全でないものが含まれている場合、ステートメント全体が安全でないとしてマークされ、行形式で記録されます。プロシージャ、関数、トリガーおよびイベントの作成および削除に使用されるステートメントは、常に安全であり、ステートメントの形式で記録されます。行ベース、混合およびステートメントベースのロギング、および安全なステートメントと安全でないステートメントの決定方法の詳細は、[セクション17.2.1「レプリケーション形式」](#)を参照してください。

特に明記されていないかぎり、ここでの備考は、バイナリロギングがサーバーで有効になっていることを前提としています ([セクション5.4.4「バイナリログ」](#)を参照。) バイナリログが有効でない場合、レプリケーションは可能でなく、バイナリログをデータリカバリに利用することもできません。

MySQL でストアドファンクションを使用する場合の条件は、次のとおりです。これらの条件は、ストアドプロシージャーまたはイベントスケジューラのイベントには適用されず、バイナリロギングが有効でないかぎり適用されません。

- スタドファンクションを作成または変更するには、通常必要な `CREATE ROUTINE` または `ALTER ROUTINE` 権限に加えて、`SET_USER_ID` 権限 (または非推奨の `SUPER` 権限) が必要です。(関数定義の `DEFINER` 値によっては、バイナリロギングが有効になっているかどうかに関係なく、`SET_USER_ID` または `SUPER` が必要になる場合があります。[セクション13.1.17「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」](#)を参照してください。)
- スタドファンクションを作成するとき、その関数が決定的であるということ、またはデータを変更しないということを宣言する必要があります。そのようにしないと、データリカバリまたはレプリケーションにとって安全でなくなる可能性があります。

デフォルトでは、`CREATE FUNCTION` ステートメントを受け入れるには、`DETERMINISTIC`、`NO SQL`、または `READS SQL DATA` の少なくとも 1 つを明示的に指定する必要があります。そうでない場合はエラーが発生します。


```
ERROR 1418 (HY000): This function has none of DETERMINISTIC, NO SQL,
or READS SQL DATA in its declaration and binary logging is enabled
(you *might* want to use the less safe log_bin_trust_function_creators
variable)
```

次の関数は決定的なため (また、データを変更しません)、安全です。

```
CREATE FUNCTION f1(i INT)
RETURNS INT
DETERMINISTIC
READS SQL DATA
BEGIN
  RETURN i;
END;
```

次の関数は `UUID()` を使用しますが、これは決定的でないため、関数も決定的でなく、安全ではありません。

```
CREATE FUNCTION f2()
RETURNS CHAR(36) CHARACTER SET utf8
BEGIN
  RETURN UUID();
END;
```

次の関数はデータを変更するので、安全ではない可能性があります。

```
CREATE FUNCTION f3(p_id INT)
RETURNS INT
BEGIN
  UPDATE t SET modtime = NOW() WHERE id = p_id;
  RETURN ROW_COUNT();
END;
```

関数の性質の評価は、作成者の「正直」に基づきます。MySQL では、`DETERMINISTIC` と宣言された関数が、非決定的な結果を生成するステートメントを持たないかどうかはチェックされません。

- ストアドファンクションを実行しようとするときに、`binlog_format=STATEMENT` が設定されている場合は、関数定義に `DETERMINISTIC` キーワードを指定する必要があります。そうでない場合は、このチェックをオーバーライドするように `log_bin_trust_function_creators=1` が指定されていないかぎり、エラーが生成され、関数は実行されません (次を参照)。再帰的関数コールの場合、`DETERMINISTIC` キーワードは最も外側のコールでのみ必要です。行ベースまたは混合バイナリロギングが使用されている場合、関数が `DETERMINISTIC` キーワードなしで定義されていても、ステートメントは受け入れられ、レプリケートされます。
- MySQL では、関数が作成時に実際に決定的かどうかはチェックされないため、`DETERMINISTIC` キーワードを使用してストアドファンクションを起動すると、ステートメントベースのロギングに対して安全でないアクションが実行されたり、安全でないステートメントを含むファンクションまたはプロシージャが起動される可能性があります。これが `binlog_format=STATEMENT` の設定時に発生した場合は、警告メッセージが発行されます。行ベースまたは混合バイナリロギングが使用されている場合、警告は発行されず、ステートメントは行ベースの形式でレプリケートされます。
- 関数作成に関する前述の条件 (`SUPER` 権限を持つ必要があることと、関数が決定的であるか、データを変更しないと宣言する必要があること) を緩和するには、`log_bin_trust_function_creators` グローバルシステム変数を 1 に設定します。デフォルトでこの変数には 0 の値が設定されていますが、次のように変更できます。

```
mysql> SET GLOBAL log_bin_trust_function_creators = 1;
```

この変数は、サーバー起動時に設定することもできます。

バイナリロギングが有効でない場合、`log_bin_trust_function_creators` は適用されません。前述のように、関数定義の `DEFINER` 値が必要としないかぎり、関数の作成に `SUPER` は必要ありません。

- レプリケーションで安全ではない可能性のある (そのため、これらを使用するストアドファンクションも安全でなくなります) 組み込み関数の詳細は、[セクション17.5.1「レプリケーションの機能と問題」](#)を参照してください。

トリガーは、ストアドファンクションと似ているので、関数に関する前述の説明がトリガーにも当てはまりますが、`CREATE TRIGGER` にはオプションの `DETERMINISTIC` 特性がないため、トリガーは常に決定的であると想定されるという点が異なります。ただし、場合によっては、この仮定が無効になることがあります。たとえば、`UUID()` 関数は非決定的です (また、複製しません)。トリガーでのこのような関数の使用には注意してください。

トリガーはテーブルを更新できるので、必要な権限がない場合には、`CREATE TRIGGER` で、ストアドファンクションの場合と同様のエラーメッセージが表示されます。レプリカ側では、レプリカはトリガー `DEFINER` 属性を使用して、トリガーの作成者とみなされるユーザーを決定します。

このセクションの残りの部分では、ロギングの実装とその意味に関する追加詳細について説明します。ストアドルーチンの使用に関する現在のロギング関連の条件の理論的根拠についての背景に関心がある場合には、こちらをお読みください。この説明はステートメントベースのロギングにのみ該当し、行ベースのロギングには該当しませんが、`CREATE` および `DROP` ステートメントは、ロギングモードとは無関係にステートメントとして記録されるという最初の項目は除きます。

- サーバーは、`CREATE EVENT`、`CREATE PROCEDURE`、`CREATE FUNCTION`、`ALTER EVENT`、`ALTER PROCEDURE`、`ALTER FUNCTION`、`DROP EVENT`、`DROP PROCEDURE`、および `DROP FUNCTION` ステートメントをバイナリログに書き込みます。
- ストアドファンクションの呼び出しは、この関数がデータを変更し、それ以外では記録されないようなステートメント内で行われた場合に、`SELECT` ステートメントとして記録されます。これにより、記録されないステートメントでストアドファンクションを使用した結果生じたデータの変更をレプリケーションできなくなるという事態が防止されます。たとえば、`SELECT` ステートメントはバイナリログに書き込まれませんが、`SELECT` は、変更を行うストアドファンクションを呼び出す場合があります。これを扱うため、`SELECT func_name()` ステートメントは、指定した関数が変更を行うときにバイナリログに書き込まれます。次のステートメントがソースサーバーで実行されるとします：

```
CREATE FUNCTION f1(a INT) RETURNS INT
BEGIN
  IF (a < 3) THEN
    INSERT INTO t2 VALUES (a);
  END IF;
  RETURN 0;
END;

CREATE TABLE t1 (a INT);
INSERT INTO t1 VALUES (1),(2),(3);

SELECT f1(a) FROM t1;
```

`SELECT` ステートメントが実行されると、関数 `f1()` は 3 回呼び出されます。このうち 2 回の呼び出しで行を挿入し、MySQL は各行に対し `SELECT` ステートメントを記録します。つまり、MySQL は次のステートメントをバイナリログに書き込みます。

```
SELECT f1(1);
SELECT f1(2);
```

サーバーは、エラーを発生させるストアドプロシージャーをストアドファンクションが呼び出すときに、そのストアドファンクションの呼び出しに対する `SELECT` ステートメントも記録します。この場合、サーバーは、予想されるエラーコードとともに、`SELECT` ステートメントをログに書き込みます。レプリカで同じエラーが発生した場合は、予想される結果となり、レプリケーションが続行されます。それ以外の場合は、レプリケーションは停止します。

- 関数によって実行されるステートメントではなく、ストアドファンクションの呼び出しのロギングは、レプリケーションでは、次の 2 つの要因から生じるセキュリティ上の意味があります。
 - 関数は、ソースサーバーとレプリカサーバーで異なる実行パスに従うことができます。
 - レプリカで実行されるステートメントは、完全な権限を持つレプリカ SQL スレッドによって処理されます。

つまり、ユーザーは関数を作成するために `CREATE ROUTINE` 権限を持っている必要がありますが、完全な権限を持つスレッドによって処理されるレプリカに対してのみ実行される危険なステートメントを含む関数を記述できます。たとえば、ソースサーバーとレプリカサーバーのサーバー ID 値がそれぞれ 1 と 2 の場合、ソースサーバー上のユーザーは、次のように安全でない関数 `unsafe_func()` を作成して起動できます：

```
mysql> delimiter //
mysql> CREATE FUNCTION unsafe_func () RETURNS INT
-> BEGIN
-> IF @@server_id=2 THEN dangerous_statement; END IF;
-> RETURN 1;
-> END;
```

```
-> //  
mysql> delimiter ;  
mysql> INSERT INTO t VALUES(unsafe_func());
```

CREATE FUNCTION および **INSERT** ステートメントはバイナリログに書き込まれるため、レプリカはそれらを実行します。レプリカ SQL スレッドには完全な権限があるため、危険なステートメントが実行されます。したがって、関数の呼出しはソースとレプリカに異なる影響を与えるため、レプリケーションに対して安全ではありません。

バイナリロギングを有効にしているサーバーに対するこの危険から保護するために、ストアドファンクションの作成者は、必要な通常の **CREATE ROUTINE** 権限に加え、**SUPER** 権限を持つ必要があります。同様に、**ALTER FUNCTION** を使用するには、ユーザーは **ALTER ROUTINE** 権限に加え、**SUPER** 権限を持つ必要があります。**SUPER** 権限がない場合は、次のエラーが発生します:

```
ERROR 1419 (HY000): You do not have the SUPER privilege and  
binary logging is enabled (you *might* want to use the less safe  
log_bin_trust_function_creators variable)
```

関数作成者が **SUPER** 権限を持つよう要求しない場合 (たとえば、システム上の **CREATE ROUTINE** 権限を持つすべてのユーザーが経験豊かなアプリケーション開発者である場合)、**log_bin_trust_function_creators** グローバルシステム変数を 1 に設定します。この変数は、サーバー起動時に設定することもできます。バイナリロギングが有効でない場合、**log_bin_trust_function_creators** は適用されません。前述のように、関数定義の **DEFINER** 値が必要としないかぎり、関数の作成に **SUPER** は必要ありません。

- 更新を実行する関数が非決定的である場合、再現可能ではありません。これは次の 2 つの望ましくない影響を及ぼす可能性があります。
 - これにより、レプリカがソースと異なります。
 - リストアされたデータが元のデータと一致しません。

これらの問題に対処するために、MySQL では次の要件が適用されます: ソースサーバーでは、関数を決定的に宣言するか、データを変更しない場合を除き、関数の作成および変更は拒否されます。ここでは次の 2 つの関数特性セットが適用されます。

- **DETERMINISTIC** 特性と **NOT DETERMINISTIC** 特性は、関数が一定の入力に対して常に同じ結果を生成するかどうかを示します。どちらの特性も指定されていない場合は、デフォルトは **NOT DETERMINISTIC** です。関数が決定的であることを宣言するには、明示的に **DETERMINISTIC** を指定する必要があります。
- **CONTAINS SQL**、**NO SQL**、**READS SQL DATA**、および **MODIFIES SQL DATA** 特性は、関数がデータを読み取るか書き込むかに関する情報を示します。**NO SQL** または **READS SQL DATA** は、関数がデータを変更しないことを示しますが、特性が指定されていない場合にデフォルトは **CONTAINS SQL** になるので、これらのどちらかを明示的に指定する必要があります。

デフォルトでは、**CREATE FUNCTION** ステートメントを受け入れるには、**DETERMINISTIC**、**NO SQL**、または **READS SQL DATA** の少なくとも 1 つを明示的に指定する必要があります。そうでない場合はエラーが発生します。

```
ERROR 1418 (HY000): This function has none of DETERMINISTIC, NO SQL,  
or READS SQL DATA in its declaration and binary logging is enabled  
(you *might* want to use the less safe log_bin_trust_function_creators  
variable)
```

log_bin_trust_function_creators を 1 に設定した場合、関数が決定的であるか、データを変更しないという要件は破棄されます。

- ストアドプロシージャの呼び出しは **CALL** レベルでなく、ステートメントレベルで記録されます。つまり、サーバーは、**CALL** ステートメントを記録せず、実際に実行するプロシージャ内のステートメントを記録します。その結果、ソースサーバーで発生するのと同じ変更がレプリカでも発生します。これにより、別々のマシン上で異なる実行パスを持つプロシージャから生じる問題が防止されます。

一般に、ストアドプロシージャ内で実行されるステートメントは、スタンドアロンでステートメントを実行した場合に適用されるものと同じルールを使用して、バイナリログに書き込まれます。プロシージャ内でのステート

メントの実行は、非プロシージャのコンテキストとまったく同じにはならないので、プロシージャステートメントのロギング時には、十分に注意してください。

- 記録されるステートメントには、ローカルプロシージャ変数への参照が含まれる場合があります。これらの変数は、ストアドプロシージャのコンテキスト外に存在しないので、このような変数を参照するステートメントは、文字どおりには記録できません。代わりに、ローカル変数のそれぞれの参照は、ロギングのために次の構造構文に置き換えられます。

```
NAME_CONST(var_name, var_value)
```

`var_name` はローカル変数名であり、`var_value` は、ステートメントの記録時に変数に含まれていた値を示す定数です。 `NAME_CONST()` には `var_value` の値と `var_name` の「名前」が含まれます。したがって、この関数を直接呼び出した場合、次のような結果が得られます。

```
mysql> SELECT NAME_CONST('myname', 14);
+-----+
| myname |
+-----+
|   14   |
+-----+
```

`NAME_CONST()` では、ログに記録されたスタンドアロンステートメントを、ストアドプロシージャ内のソースで実行された元のステートメントと同じ効果でレプリカで実行できます。

`NAME_CONST()` を使用した結果、ソースカラム式がローカル変数を参照するときに、`CREATE TABLE ... SELECT` ステートメントの問題が生じる場合があります。これらの参照を `NAME_CONST()` 式に変換すると、ソースサーバーとレプリカサーバーで異なるカラム名、または名前が長すぎて有効なカラム識別子にならない場合があります。回避策では、ローカル変数を参照するカラムのエイリアスを指定します。 `myvar` の値が 1 の場合は次のステートメントを検討してください。

```
CREATE TABLE t1 SELECT myvar;
```

これは次のように書き換えられます：

```
CREATE TABLE t1 SELECT NAME_CONST(myvar, 1);
```

ソーステーブルとレプリカテーブルのカラム名が同じであることを確認するには、次のようなステートメントを記述します：

```
CREATE TABLE t1 SELECT myvar AS myvar;
```

書き換えられたステートメントは次のようになります。

```
CREATE TABLE t1 SELECT NAME_CONST(myvar, 1) AS myvar;
```

- 記録されるステートメントには、ユーザー定義変数への参照が含まれる場合があります。これを処理するために、MySQL はバイナリログに `SET` ステートメントを書き込み、ソースと同じ値を持つ変数がレプリカに存在することを確認します。たとえば、ステートメントが変数 `@my_var` を参照している場合、バイナリログ内でそのステートメントの前に次のステートメントが続きます。ここで、`value` はソース上の `@my_var` の値です：

```
SET @my_var = value;
```

- プロシージャの呼び出しは、コミットまたはロールバックしたトランザクション内で行えます。プロシージャ実行のトランザクションの側面が正しく複製されるように、トランザクションのコンテキストが説明されます。つまり、サーバーは、実際にデータを実行し修正するプロシージャ内のステートメントを記録し、`BEGIN`、`COMMIT`、および `ROLLBACK` ステートメントも必要に応じて記録します。たとえば、プロシージャがトランザクションテーブルだけを更新し、ロールバックされるトランザクション内で実行される場合、これらの更新は記録されません。プロシージャがコミットされたトランザクション内で行われた場合、`BEGIN` および `COMMIT` ステートメントが更新とともに記録されます。ロールバックしたトランザクション内で実行するプロシージャの場合、そのステートメントは、ステートメントがスタンドアロンで実行された場合に適用されるものと同じルールを使用して記録されます。
- トランザクションテーブルに対する更新は記録されません。
- 非トランザクションテーブルに対する更新は、ロールバックで取り消されないため、記録されます。

- トランザクションテーブルと非トランザクションテーブルの混在に対する更新は、レプリカがソースと同じ変更およびロールバックを行うように、**BEGIN** と **ROLLBACK** で囲まれて記録されます。
- ストアドプロシージャの呼び出しは、ストアドファンクション内から呼び出される場合、ステートメントレベルのバイナリログに書き込まれません。この場合、記録される唯一の対象は、関数を呼び出すステートメント (記録されたステートメント内で行われた場合) または **DO** ステートメント (記録されないステートメント内で行われた場合) です。このため、それ以外の場合にプロシージャ自体が安全であっても、プロシージャを呼び出すストアドファンクションを使用するときには注意を払う必要があります。

25.8 ストアドプログラムの制約

- [ストアルーチンでは許可されていない SQL ステートメント](#)
- [ストアドファンクションの制約](#)
- [トリガーの制約](#)
- [ストアルーチン内の名前競合](#)
- [レプリケーションに関する考慮事項](#)
- [デバッグに関する考慮事項](#)
- [SQL:2003 標準のサポート外の構文](#)
- [ストアルーチンの同時実行性に関する考慮事項](#)
- [イベントスケジューラの制約](#)
- [NDB Cluster 内のストアルーチンとトリガー](#)

これらの制約は、[第25章「ストアドオブジェクト」](#)で説明している機能に適用されます。

ここに記載されている制約の中には、すべてのストアルーチン、つまりストアドプロシージャとストアドファンクションの両方に適用されるものがあります。また、ストアドプロシージャには適用されず、[ストアドファンクションに固有の制約](#)もいくつか存在します。

ストアドファンクションの制約は、トリガーにも適用されます。[トリガーに固有の制約](#)もいくつかあります。

ストアドプロシージャの制約は、イベントスケジューラのイベント定義の **DO** 句にも適用されます。[イベントに固有の制約](#)もいくつかあります。

ストアルーチンでは許可されていない SQL ステートメント

ストアルーチンには自由に SQL ステートメントを含めることはできません。次のステートメントは許可されていません。

- [LOCK TABLES](#) および [UNLOCK TABLES](#) のロックステートメント。
- [ALTER VIEW](#)
- [LOAD DATA](#)。
- SQL 準備済みステートメント ([PREPARE](#)、[EXECUTE](#)、[DEALLOCATE PREPARE](#)) は、ストアドプロシージャで使用できますが、ストアドファンクションやトリガーでは使用できません。そのため、ストアドファンクションとトリガーは動的 SQL (この場合はステートメントを文字列として構築してから実行します) を使用できません。
- 通常、SQL 準備済みステートメントで許可されていないステートメントは、ストアドプログラムでも許可されません。準備済みステートメントとしてサポートされているステートメントのリストについては、[セクション13.5「リペアドステートメント」](#)を参照してください。例外は [SIGNAL](#)、[RESIGNAL](#)、および [GET DIAGNOSTICS](#) であり、これらは準備済みステートメントとして許可されていませんが、ストアドプログラムで許可されます。

- ローカル変数はストアドプログラムの実行中にのみスコープ内にあるので、これらの参照は、ストアドプログラム内で作成された準備済みステートメントでは許可されていません。準備済みステートメントのスコープは現在のセッションであり、ストアドプログラムではないので、ステートメントはプログラムの終了後に実行でき、この時点で変数はスコープ内に存在しなくなります。たとえば、`SELECT ... INTO local_var` は準備済みステートメントとして使用できません。この制約は、ストアドプロシージャおよびストアドファンクションのパラメータにも適用されます。 [セクション13.5.1「PREPARE ステートメント」](#) を参照してください。
- すべてのストアドプログラム (ストアドプロシージャとストアドファンクション、トリガー、およびイベント) 内で、パーサーは、`BEGIN [WORK]` を `BEGIN ... END` ブロックの開始として扱います。このコンテキストでトランザクションを開始するには、代わりに `START TRANSACTION` を使用します。

ストアドファンクションの制約

次の追加ステートメントまたは操作は、ストアドファンクション内で許可されていません。これらはストアドプロシージャで許可されていますが、ストアドファンクションまたはトリガー内から呼び出されるストアドプロシージャを除きます。たとえば、ストアドプロシージャで `FLUSH` を使用する場合、ストアドファンクションまたはトリガーからそのストアドプロシージャを呼び出すことはできません。

- 明示的または暗黙的なコミットまたはロールバックを実行するステートメント。これらのステートメントのサポートは、SQL 標準では必要ありません。SQL 標準では、各 DBMS ベンダーがこれらのステートメントを許可するかどうかを決められると定めています。
- 結果セットを返すステートメント。これには、`INTO var_list` 句を含まない `SELECT` ステートメントや、`SHOW`、`EXPLAIN`、および `CHECK TABLE` などのほかのステートメントも含まれます。関数は、`SELECT ... INTO var_list` を使用するか、カーソルと `FETCH` ステートメントを使用すると、結果セットを処理できます。 [セクション13.2.10.1「SELECT ... INTO ステートメント」](#) および [セクション13.6.6「カーソル」](#) を参照してください。
- `FLUSH` ステートメント。
- ストアドファンクションは再帰的に使用できません。
- ストアドファンクションまたはトリガーは、そのストアドファンクションまたはトリガーを呼び出したステートメントによって (読み取りまたは書き込みに) すでに使用されているテーブルを変更できません。
- ストアドファンクションで、一時テーブルを異なるエイリアスで複数回参照する場合、ストアドファンクション内の別々のステートメントで参照を行う場合でも、「表を再オープンできません: 'tbl_name'」というエラーが発生します。
- ストアドファンクションを呼び出す `HANDLER ... READ` ステートメントは、レプリケーションエラーを引き起こす可能性があり、許可されません。

トリガーの制約

トリガーの場合、さらに次の制約が適用されます。

- トリガーは外部キーアクションでアクティブ化されません。
- 行ベースのレプリケーションを使用している場合、レプリカのトリガーは、ソースで発生したステートメントによってアクティブ化されません。レプリカのトリガーは、ステートメントベースレプリケーションを使用している場合にアクティブになります。詳細は、 [セクション17.5.1.36「レプリケーションとトリガー」](#) を参照してください。
- `RETURN` ステートメントはトリガーでは許可されていません。トリガーは値を返すことができません。すぐにトリガーを終了するには、`LEAVE` ステートメントを使用します。
- トリガーは、`mysql` データベース内のテーブルでは許可されていません。 `INFORMATION_SCHEMA` または `performance_schema` テーブルでも許可されていません。これらのテーブルは実際にはビューであり、トリガーはビューでは許可されません。
- トリガーキャッシュは、ベースとなるオブジェクトのメタデータが変更された場合は検出しません。トリガーがテーブルを使用し、トリガーがキャッシュにロードされたあとにそのテーブルに変更があった場合、トリガーは古いメタデータを使用して動作します。

ストアドルーチン内の名前競合

ルーチンパラメータ、ローカル変数、およびテーブルカラムに同じ識別子が使用される場合があります。また、同じローカル変数名を、ネスト化されたブロックで使用することもできます。例:

```
CREATE PROCEDURE p (i INT)
BEGIN
  DECLARE i INT DEFAULT 0;
  SELECT i FROM t;
  BEGIN
    DECLARE i INT DEFAULT 1;
    SELECT i FROM t;
  END;
END;
```

このような場合、識別子はあいまいになり、次の優先順位ルールが適用されます。

- ローカル変数では、ルーチンパラメータやテーブルカラムが優先されます。
- ルーチンパラメータでは、テーブルカラムが優先されます。
- 内部ブロック内のローカル変数では、外部ブロック内のローカル変数が優先されます。

変数でテーブルカラムが優先される動作は、非標準です。

レプリケーションに関する考慮事項

ストアドルーチンを使用すると、レプリケーションの問題が生じることがあります。この問題については、[セクション25.7「ストアプログラムバイナリロギング」](#)で詳しく述べられています。

`--replicate-wild-do-table=db_name.tbl_name` オプションはテーブル、ビュー、およびトリガーに適用されます。ストアアドプロシージャと関数、またはイベントには適用されません。後者のオブジェクトで作用するステートメントをフィルタするには、1つまたは複数の `--replicate-*-db` オプションを使用します。

デバッグに関する考慮事項

ストアドルーチンのデバッグ機能は存在しません。

SQL:2003 標準のサポート外の構文

MySQL ストアドドルーチン構文は SQL:2003 標準に基づきます。この標準の次の項目は現在サポートされていません。

- [UNDO](#) ハンドラ
- [FOR](#) ループ

ストアドルーチンの同時実行性に関する考慮事項

セッション間のやり取りの問題を防止するために、クライアントのステートメント発行時、サーバーではステートメントの実行に利用できるルーチンとトリガーのスナップショットが使用されます。つまり、サーバーは、ステートメントの実行中に使用される可能性のあるプロシージャ、関数、およびトリガーのリストを算出してロードし、ステートメントの実行に進みます。ステートメントの実行時は、ほかのセッションが実行するルーチンへの変更は認識されません。

並列性を最大にするために、ストアドファンクションでは、その副作用を最小限に抑える必要があります。特に、ストアドファンクション内のテーブルを更新することにより、そのテーブルでの並列操作が減少することがあります。ストアドファンクションは、実行前にテーブルロックを取得して、ステートメントが実行する順序とログに表示されるときに順序の不一致によるバイナリログの不整合を回避します。ステートメントベースのバイナリロギングが使用される場合、関数内で実行されるステートメントではなく、関数を呼び出すステートメントが記録されます。その結果、ベースとなる同じテーブルを更新するストアドファンクションは、並列で実行しません。対照的に、ストアアドプロシージャはテーブルレベルのロックを取得しません。ストアアドプロシージャ内で実行されたすべてのステートメント

トメントは、ステートメントベースのバイナリロギングの場合でも、バイナリログに書き込まれます。 [セクション 25.7「ストアプログラムバイナリロギング」](#)を参照してください。

イベントスケジューラの制約

次の制限は、イベントスケジューラに固有のもので、

- イベント名は大文字と小文字を区別せずに処理されます。たとえば、`anEvent` と `AnEvent` という名前の 2 つのイベントを同じデータベース内に含めることはできません。
- イベント名が変数によって指定されている場合、ストアプログラム内からイベントを作成、変更、または削除することはできません。イベントは、ストアルーチンやトリガーを作成、変更、削除することもできません。
- `LOCK TABLES` ステートメントの有効時は、イベントでの DDL ステートメントは禁止されています。
- `YEAR`、`QUARTER`、`MONTH`、および `YEAR_MONTH` の間隔を使用したイベントのタイミングは、月で解決されます。ほかの間隔を使用したタイミングは秒で解決されます。同時に行われるようにスケジュール設定されたイベントは、指定の順序で実行できません。さらに、丸め、スレッドアプリケーションの特性、およびイベントを作成しその実行を信号で伝えるためにゼロ以外の時間長が必要になるため、イベントが 1、2 秒ほど遅れる場合があります。ただし、`INFORMATION_SCHEMA.EVENTS` テーブルの `LAST_EXECUTED` カラムに表示される時間は、常に実際のイベント実行時間の 1 秒以内に正確です。(Bug #16522 も参照してください。)
- イベントの本体に含まれるステートメントの各実行は、新しい接続で行われます。したがって、これらのステートメントは、`SHOW STATUS` によって表示される `Com_select` や `Com_insert` などのサーバーステートメント数に対する特定のユーザーセッションには影響しません。ただし、このような数はグローバルスコープで更新されます。(Bug #16422)
- イベントは、Unix エポックの最後の時間以降をサポートしません。この時間は 2038 年の年頭あたりになります。このような日付はイベントスケジューラで特に許可されません。(Bug #16396)
- `CREATE EVENT` および `ALTER EVENT` ステートメントの `ON SCHEDULE` 句でのストアドファンクション、ユーザー定義関数、およびテーブルの参照はサポートされていません。このような種類の参照は許可されていません。(詳細は Bug #22830 を参照してください。)

NDB Cluster 内のストアルーチンとトリガー

ストアプロシージャ、ストアドファンクション、トリガーおよびスケジュール済イベントはすべて、NDB ストレージエンジンを使用するテーブルでサポートされますが、これらはクラスタ SQL ノードとして機能する MySQL Servers 間で自動的に伝播されないことに注意する必要があります。これは、ストアルーチンおよびトリガー定義が、クラスタノード間でコピーされない InnoDB テーブルを使用して `mysql` システムデータベース内のテーブルに格納されるためです。

MySQL Cluster テーブルと対話するストアルーチンまたはトリガーは、ストアルーチンまたはトリガーを使用するクラスタに参加する各 MySQL Server で適切な `CREATE PROCEDURE`、`CREATE FUNCTION`、または `CREATE TRIGGER` ステートメントを実行して再作成する必要があります。同様に、既存のストアルーチンまたはトリガーに対する変更は、クラスタにアクセスする各 MySQL Server で適切な `ALTER` ステートメントまたは `DROP` ステートメントを使用して、すべてのクラスタ SQL ノードで明示的に実行する必要があります。

警告

NDB ストレージエンジンを使用するように `mysql` データベーステーブルを変換して、ここで説明した問題を回避しないでください。「`mysql` データベースでのシステムテーブルの変更はサポートされていません」では、望ましくない結果が生成される可能性があります。

25.9 ビューの制約

ビューの定義で参照できるテーブルの最大数は 61 です。

ビューの処理は最適化されていません。

- ビューにはインデックスを作成できません。

- マージアルゴリズムを使用して処理されたビューに、インデックスを使用することは可能です。ただし、TEMPTABLE アルゴリズムで処理されたビューは、そのベースとなるテーブルのインデックスを利用できません (ただし、一時テーブルの作成中にはインデックスを使用できます)。

一般的な原則では、テーブルを変更することも、サブクエリーの同じテーブルから選択することもできません。 [セクション13.2.11.12「サブクエリーの制約」](#)を参照してください。

テーブルから選択するビューを選ぶ場合、ビューがサブクエリーのテーブルから選択される場合や、ビューがマージアルゴリズムを使用して評価される場合にも、同じ原則が適用されます。例:

```
CREATE VIEW v1 AS
SELECT * FROM t2 WHERE EXISTS (SELECT 1 FROM t1 WHERE t1.a = t2.a);

UPDATE t1, v2 SET t1.a = 1 WHERE t1.b = v2.b;
```

一時テーブルを使用してビューが評価される場合、ビューサブクエリーのテーブルから選択し、さらに外部クエリーでそのテーブルを変更することが可能です。この場合、ビューは一時テーブルに格納されるため、サブクエリーのテーブルから選択して同時に変更することはありません。(これは、ビュー定義で `ALGORITHM = TEMPTABLE` を指定することによって、MySQL で TEMPTABLE アルゴリズムを強制的に使用させる 1 つの理由です。)

`DROP TABLE` または `ALTER TABLE` を使用すると、ビュー定義で使用されているテーブルを削除または変更できます。ビューを無効化する場合でも、`DROP` または `ALTER` 操作によって警告が発せられることはありません。代わりに、あとからビューを使用するときにエラーが発生します。`CHECK TABLE` は、`DROP` または `ALTER` 操作で無効化されたビューのチェックに使用できます。

ビューの更新可能性に関しては、どのビューでも理論的に更新可能であれば、実際に更新可能である必要があるというのがビューの全体的な目的です。MySQL はできるだけ迅速に実行できます。理論的には多くの更新可能なビューを更新できるようになりましたが、まだ制限があります。詳細は、[セクション25.5.3「更新可能および挿入可能なビュー」](#)を参照してください。

ビューの現在の実装には欠点があります。ビューの作成に必要な基本権限 (`CREATE VIEW` および `SELECT` 権限) がユーザーに付与されている場合、そのユーザーに `SHOW VIEW` 権限も付与されていないかぎり、そのユーザーはそのオブジェクトに対して `SHOW CREATE VIEW` をコールできません。

権限の不足のために `mysqldump` が失敗する可能性があるという欠点によって、このコマンドを使用したデータベースのバックアップで問題が発生する場合があります。この問題については [Bug #22062](#) で説明しています。

問題の回避策として、ビューが作成されたときに MySQL が暗黙的に `SHOW VIEW` 権限を与えないため、`CREATE VIEW` を認められているユーザーに、管理者が手動でこの権限を付与します。

ビューにはインデックスがないので、インデックスのヒントは適用されません。ビューからの選択時のインデックスヒントの使用は許可されていません。

`SHOW CREATE VIEW` は、カラムごとに `AS alias_name` 句を使用してビュー定義を表示します。式からカラムを作成する場合、デフォルトのエイリアスは式テキストになり、かなり長くなることがあります。`CREATE VIEW` ステートメント内のカラム名に対するエイリアスは、(256 文字の最大のエイリアス長ではなく) 64 文字の最大のカラム長に対してチェックされます。その結果、いずれかのカラムエイリアスが 64 文字を超えた場合、`SHOW CREATE VIEW` の出力から作成されたビューは失敗します。これによって、長すぎるエイリアスを持つビューに対し、次の環境で問題が起きる可能性があります。

- ビュー定義は、カラム長制限を強制する新しいレプリカへのレプリケートに失敗します。
- `mysqldump` で作成されたダンプファイルは、カラム長の制約を強制するサーバーにロードできません。

これらの問題を回避するには、短いカラム名を提供するエイリアスを使用するように、問題のある各ビュー定義を変更します。その後、ビューは適切にレプリケートされ、エラーを発生させずにダンプおよびリロードできます。定義を変更するには、`DROP VIEW` および `CREATE VIEW` でビューを削除してから再度作成するか、`CREATE OR REPLACE VIEW` を使用して定義を置き換えます。

ダンプファイルのビュー定義リロード時に問題が発生する場合は、`CREATE VIEW` ステートメントを変更するようにダンプファイルを編集するとこの問題を回避できます。ただし、これは元のビュー定義を変更しないので、その後のダンプ操作の問題を引き起こす可能性があります。

第 26 章 INFORMATION_SCHEMA テーブル

目次

26.1 はじめに	4132
26.2 INFORMATION_SCHEMA ADMINISTRABLE_ROLE_AUTHORIZATIONS テーブル	4135
26.3 INFORMATION_SCHEMA APPLICABLE_ROLES テーブル	4136
26.4 INFORMATION_SCHEMA CHARACTER_SETS テーブル	4136
26.5 INFORMATION_SCHEMA CHECK_CONSTRAINTS テーブル	4137
26.6 INFORMATION_SCHEMA COLLATIONS テーブル	4137
26.7 INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY テーブル	4138
26.8 INFORMATION_SCHEMA COLUMNS テーブル	4138
26.9 INFORMATION_SCHEMA COLUMNS_EXTENSIONS テーブル	4141
26.10 INFORMATION_SCHEMA COLUMN_PRIVILEGES テーブル	4141
26.11 INFORMATION_SCHEMA COLUMN_STATISTICS テーブル	4142
26.12 INFORMATION_SCHEMA ENABLED_ROLES テーブル	4143
26.13 INFORMATION_SCHEMA ENGINES テーブル	4143
26.14 INFORMATION_SCHEMA EVENTS テーブル	4144
26.15 INFORMATION_SCHEMA FILES テーブル	4147
26.16 INFORMATION_SCHEMA KEY_COLUMN_USAGE テーブル	4155
26.17 INFORMATION_SCHEMA ndb_transid_mysql_connection_map テーブル	4156
26.18 INFORMATION_SCHEMA KEYWORDS テーブル	4157
26.19 INFORMATION_SCHEMA OPTIMIZER_TRACE テーブル	4157
26.20 INFORMATION_SCHEMA PARAMETERS テーブル	4158
26.21 INFORMATION_SCHEMA PARTITIONS テーブル	4159
26.22 INFORMATION_SCHEMA PLUGINS テーブル	4162
26.23 INFORMATION_SCHEMA PROCESSLIST テーブル	4163
26.24 INFORMATION_SCHEMA PROFILING テーブル	4165
26.25 INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS テーブル	4166
26.26 INFORMATION_SCHEMA RESOURCE_GROUPS テーブル	4167
26.27 INFORMATION_SCHEMA ROLE_COLUMN_GRANTS テーブル	4167
26.28 INFORMATION_SCHEMA ROLE_ROUTINE_GRANTS テーブル	4168
26.29 INFORMATION_SCHEMA ROLE_TABLE_GRANTS テーブル	4169
26.30 INFORMATION_SCHEMA ROUTINES テーブル	4169
26.31 INFORMATION_SCHEMA SCHEMATA テーブル	4172
26.32 INFORMATION_SCHEMA SCHEMATA_EXTENSIONS テーブル	4173
26.33 INFORMATION_SCHEMA SCHEMA_PRIVILEGES テーブル	4173
26.34 INFORMATION_SCHEMA STATISTICS テーブル	4174
26.35 INFORMATION_SCHEMA ST_GEOMETRY_COLUMNS テーブル	4176
26.36 INFORMATION_SCHEMA ST_SPATIAL_REFERENCE_SYSTEMS テーブル	4177
26.37 INFORMATION_SCHEMA ST_UNITS_OF_MEASURE テーブル	4178
26.38 INFORMATION_SCHEMA TABLES テーブル	4179
26.39 INFORMATION_SCHEMA TABLES_EXTENSIONS テーブル	4182
26.40 INFORMATION_SCHEMA TABLESPACES テーブル	4183
26.41 INFORMATION_SCHEMA TABLESPACES_EXTENSIONS テーブル	4183
26.42 INFORMATION_SCHEMA TABLE_CONSTRAINTS テーブル	4183
26.43 INFORMATION_SCHEMA TABLE_CONSTRAINTS_EXTENSIONS テーブル	4184
26.44 INFORMATION_SCHEMA TABLE_PRIVILEGES テーブル	4184
26.45 INFORMATION_SCHEMA TRIGGERS テーブル	4185
26.46 INFORMATION_SCHEMA USER_ATTRIBUTES テーブル	4187
26.47 INFORMATION_SCHEMA USER_PRIVILEGES テーブル	4188
26.48 INFORMATION_SCHEMA VIEWS テーブル	4189
26.49 INFORMATION_SCHEMA VIEW_ROUTINE_USAGE テーブル	4190
26.50 INFORMATION_SCHEMA VIEW_TABLE_USAGE テーブル	4191
26.51 INFORMATION_SCHEMA InnoDB テーブル	4191
26.51.1 INFORMATION_SCHEMA INNODB_BUFFER_PAGE テーブル	4191
26.51.2 INFORMATION_SCHEMA INNODB_BUFFER_PAGE_LRU テーブル	4195

26.51.3	INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS テーブル	4198
26.51.4	INFORMATION_SCHEMA INNODB_CACHED_INDEXES テーブル	4201
26.51.5	INFORMATION_SCHEMA INNODB_CMP および INNODB_CMP_RESET テーブル	4201
26.51.6	INFORMATION_SCHEMA INNODB_CMPMEM および INNODB_CMPMEM_RESET テーブル	4203
26.51.7	INFORMATION_SCHEMA INNODB_CMP_PER_INDEX および INNODB_CMP_PER_INDEX_RESET テーブル	4204
26.51.8	INFORMATION_SCHEMA INNODB_COLUMNS テーブル	4205
26.51.9	INFORMATION_SCHEMA INNODB_DATAFILES テーブル	4207
26.51.10	INFORMATION_SCHEMA INNODB_FIELDS テーブル	4207
26.51.11	INFORMATION_SCHEMA INNODB_FOREIGN テーブル	4208
26.51.12	INFORMATION_SCHEMA INNODB_FOREIGN_COLS テーブル	4209
26.51.13	INFORMATION_SCHEMA INNODB_FT_BEING_DELETED テーブル	4209
26.51.14	INFORMATION_SCHEMA INNODB_FT_CONFIG テーブル	4210
26.51.15	INFORMATION_SCHEMA INNODB_FT_DEFAULT_STOPWORD テーブル	4211
26.51.16	INFORMATION_SCHEMA INNODB_FT_DELETED テーブル	4212
26.51.17	INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE テーブル	4213
26.51.18	INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE テーブル	4214
26.51.19	INFORMATION_SCHEMA INNODB_INDEXES テーブル	4215
26.51.20	INFORMATION_SCHEMA INNODB_LOCKS テーブル	4217
26.51.21	INFORMATION_SCHEMA INNODB_LOCK_WAITS テーブル	4218
26.51.22	INFORMATION_SCHEMA INNODB_METRICS テーブル	4218
26.51.23	INFORMATION_SCHEMA INNODB_SESSION_TEMP_TABLESPACES テーブル	4220
26.51.24	INFORMATION_SCHEMA INNODB_TABLES テーブル	4221
26.51.25	INFORMATION_SCHEMA INNODB_TABLESPACES テーブル	4222
26.51.26	INFORMATION_SCHEMA INNODB_TABLESPACES_BRIEF テーブル	4224
26.51.27	INFORMATION_SCHEMA INNODB_TABLESTATS ビュー	4225
26.51.28	INFORMATION_SCHEMA INNODB_TEMP_TABLE_INFO テーブル	4226
26.51.29	INFORMATION_SCHEMA INNODB_TRX テーブル	4227
26.51.30	INFORMATION_SCHEMA INNODB_VIRTUAL テーブル	4230
26.52	INFORMATION_SCHEMA スレッドプールテーブル	4231
26.52.1	INFORMATION_SCHEMA TP_THREAD_GROUP_STATE テーブル	4231
26.52.2	INFORMATION_SCHEMA TP_THREAD_GROUP_STATS テーブル	4232
26.52.3	INFORMATION_SCHEMA TP_THREAD_STATE テーブル	4232
26.53	INFORMATION_SCHEMA の接続制御テーブル	4233
26.53.1	INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS テーブル	4233
26.54	INFORMATION_SCHEMA MySQL Enterprise Firewall テーブル	4233
26.54.1	INFORMATION_SCHEMA MYSQL_FIREWALL_USERS テーブル	4233
26.54.2	INFORMATION_SCHEMA MYSQL_FIREWALL_WHITELIST テーブル	4234
26.55	SHOW ステートメントの拡張	4234

INFORMATION_SCHEMA では、データベースメタデータへのアクセスを実現し、データベースまたはテーブルの名前、カラムのデータ型、アクセス権限などの MySQL Server に関する情報を提供します。この情報に使用されることがある別の用語が、データディクショナリとシステムカタログです。

26.1 はじめに

INFORMATION_SCHEMA では、データベースメタデータへのアクセスを実現し、データベースまたはテーブルの名前、カラムのデータ型、アクセス権限などの MySQL Server に関する情報を提供します。この情報に使用されることがある別の用語が、データディクショナリとシステムカタログです。

- [INFORMATION_SCHEMA の使用上のノート](#)
- [文字セットの考慮事項](#)
- [SHOW ステートメントの代替方法としての INFORMATION_SCHEMA](#)
- [INFORMATION_SCHEMA および権限](#)
- [パフォーマンスに関する考慮事項](#)

- [標準に関する考慮事項](#)
- [INFORMATION_SCHEMA 参照セクションでの規則](#)
- [関連情報](#)

INFORMATION_SCHEMA の使用上のノート

INFORMATION_SCHEMA は、各 MySQL インスタンス内のデータベースであり、MySQL Server が保持するほかのすべてのデータベースに関する情報を格納する場所です。 **INFORMATION_SCHEMA** データベースには複数の読み取り専用テーブルが含まれます。これらには実際にはビューがあるので、関連付けられたファイルはなく、トリガーは設定できません。また、その名前を持つデータベースディレクトリもありません。

USE ステートメントを使用してデフォルトデータベースとして **INFORMATION_SCHEMA** を選択できますが、実行できる操作はテーブル内容の読み取りだけで、テーブルに対する **INSERT**、**UPDATE**、**DELETE** 操作は実行できません。

次に **INFORMATION_SCHEMA** から情報を取り出すステートメントの例を示します。

```
mysql> SELECT table_name, table_type, engine
        FROM information_schema.tables
        WHERE table_schema = 'db5'
        ORDER BY table_name;
+-----+-----+-----+
| table_name | table_type | engine |
+-----+-----+-----+
| fk        | BASE TABLE | InnoDB |
| fk2       | BASE TABLE | InnoDB |
| goto      | BASE TABLE | MyISAM |
| into      | BASE TABLE | MyISAM |
| k         | BASE TABLE | MyISAM |
| kurs      | BASE TABLE | MyISAM |
| loop      | BASE TABLE | MyISAM |
| pk        | BASE TABLE | InnoDB |
| t         | BASE TABLE | MyISAM |
| t2        | BASE TABLE | MyISAM |
| t3        | BASE TABLE | MyISAM |
| t7        | BASE TABLE | MyISAM |
| tables    | BASE TABLE | MyISAM |
| v         | VIEW        | NULL   |
| v2        | VIEW        | NULL   |
| v3        | VIEW        | NULL   |
| v56       | VIEW        | NULL   |
+-----+-----+-----+
17 rows in set (0.01 sec)
```

説明: このステートメントは、データベース **db5** 内のすべてのテーブルのリストを要求し、テーブルの名前、種類、ストレージエンジンという 3 つの情報だけを示します。

文字セットの考慮事項

文字カラム (**TABLES.TABLE_NAME** など) の定義は、通常、**VARCHAR(N) CHARACTER SET utf8** であり、ここで **N** は少なくとも 64 です。MySQL は、このようなカラムでのすべての検索、ソート、比較、およびほかの文字列操作に、この文字セット (**utf8_general_ci**) のデフォルトの照合順序を使用します。

一部の MySQL オブジェクトはファイルとして表されるので、**INFORMATION_SCHEMA** 文字列カラムでの検索は、ファイルシステムでの大文字と小文字の区別によって影響を受けることがあります。詳細は、[セクション 10.8.7 「INFORMATION_SCHEMA 検索での照合の使用」](#) を参照してください。

SHOW ステートメントの代替方法としての INFORMATION_SCHEMA

SELECT ... FROM INFORMATION_SCHEMA ステートメントは、MySQL がサポートするさまざまな **SHOW** ステートメント (**SHOW DATABASES**、**SHOW TABLES** など) により提供された情報にアクセスするための一貫性の高い方法として用意されています。 **SELECT** は、**SHOW** に比べて次の利点があります。

- すべてのアクセスがテーブル上で行われるので、Codd のルールに準拠しています。

- **SELECT** ステートメントの使い慣れた構文を使用でき、学習が必要なものは一部のテーブルおよびカラムの名前だけです。
- 実装者はキーワードの追加を心配する必要がありません。
- **INFORMATION_SCHEMA** クエリーからの結果をフィルタリング、ソート、連結でき、構文解析するデータ構造やテキスト表現など、アプリケーションに必要なあらゆる形式に変換できます。
- この手法は、ほかのデータベースシステムとの相互運用可能性に優れています。たとえば、Oracle Database ユーザーは、Oracle データディクショナリのテーブルのクエリーに慣れています。

SHOW はよく知られ、広く使用されているので、**SHOW** ステートメントは引き続き代替方法として有効です。実際、[セクション26.55「SHOW ステートメントの拡張」](#)で説明しているように、**INFORMATION_SCHEMA** の実装に伴い、**SHOW** に拡張が行われています。

INFORMATION_SCHEMA および権限

ほとんどの **INFORMATION_SCHEMA** テーブルでは、各 MySQL ユーザーはそれらにアクセスする権限を持ちますが、ユーザーが適切なアクセス権限を持つオブジェクトに対応するテーブルの行のみを表示できます。場合によっては (たとえば、**INFORMATION_SCHEMA ROUTINES** テーブルの **ROUTINE_DEFINITION** カラム)、権限が不十分なユーザーに **NULL** が表示されます。一部のテーブルには異なる権限要件があります。これらの要件については、該当するテーブルの説明に記載されています。たとえば、InnoDB テーブル (**INNODB_**で始まる名前を持つテーブル) には、**PROCESS** 権限が必要です。

同じ権限が、**INFORMATION_SCHEMA** からの情報の選択と、**SHOW** ステートメントを通じた同じ情報の表示に適用されます。どちらの場合でも、オブジェクトに関する情報を表示するには、そのオブジェクトに対する何らかの権限が必要です。

パフォーマンスに関する考慮事項

複数のデータベースの情報を検索する **INFORMATION_SCHEMA** クエリーは、長時間かかり、パフォーマンスに影響を及ぼす可能性があります。クエリーの効率を確認するには、**EXPLAIN** を使用できます。**EXPLAIN** 出力を使用した **INFORMATION_SCHEMA** クエリーの調整に関する詳細は、[セクション8.2.3「INFORMATION_SCHEMA クエリーの最適化」](#)を参照してください。

標準に関する考慮事項

MySQL での **INFORMATION_SCHEMA** テーブル構造の実装は ANSI/ISO SQL:2003 標準パート 11 の Schemata に準拠しています。SQL:2003 の中核機能の F021 基本情報スキーマにほぼ準拠することを意図しています。

SQL Server 2000 (これも標準に準拠しています) のユーザーであれば、非常に類似していることがわかるでしょう。ただし、MySQL では実装に関連しない多くのカラムを省略し、MySQL 固有のカラムを追加しています。そのように追加されたカラムの 1 つは、**INFORMATION_SCHEMA TABLES** テーブルの **ENGINE** カラムです。

ほかの DBMS では **syscat** や **system** などのさまざまな名前を使用していますが、標準の名前は **INFORMATION_SCHEMA** です。

標準または DB2、SQL Server、Oracle で予約されている名前を使用しないように、「MySQL 拡張」のマークを付けて一部のカラムの名前を変更しています。(たとえば、**TABLES** テーブルでの **COLLATION** を **TABLE_COLLATION** に変更しました。) <https://web.archive.org/web/20070428032454/http://www.dbazine.com/db2/db2-disarticles/gulutzan5> の記事の末尾近くにある予約語のリストを参照してください。

INFORMATION_SCHEMA 参照セクションでの規則

次のセクションでは、**INFORMATION_SCHEMA** でのテーブルおよびカラムのそれぞれについて説明します。カラムごとに次の 3 つの情報が 있습니다。

- 「**INFORMATION_SCHEMA** 名」には、**INFORMATION_SCHEMA** テーブルのカラムの名前が示されます。これは、「備考」フィールドで「MySQL 拡張」と記されていないかぎり、標準の SQL 名に一致します。

- 「SHOW 名」には、もっとも近い SHOW ステートメントに同等のフィールド名がある場合は、この名前が表示されます。
- 「備考」には、必要に応じて追加情報が記されます。このフィールドが NULL の場合、カラムの値が常に NULL であることを意味します。このフィールドに「MySQL 拡張」とある場合、そのカラムは標準 SQL に対する MySQL 拡張です。

多くのセクションは、どの SHOW ステートメントが、`INFORMATION_SCHEMA` から情報を取得する SELECT と同等であるかを示します。デフォルトデータベースの情報を表示する SHOW ステートメントでは、`FROM db_name` 句を省略した場合、`INFORMATION_SCHEMA` テーブルから情報を取得するクエリーの WHERE 句に `AND TABLE_SCHEMA = SCHEMA()` 条件を追加すると、デフォルトデータベースの情報を選択できます。

関連情報

次の各セクションでは、`INFORMATION_SCHEMA` 関連のその他のトピックについて説明します:

- InnoDB ストレージエンジンに固有の `INFORMATION_SCHEMA` テーブルに関する情報: [セクション 26.51 「INFORMATION_SCHEMA InnoDB テーブル」](#)
- スレッドプールプラグインに固有の `INFORMATION_SCHEMA` テーブルに関する情報: [セクション 26.52 「INFORMATION_SCHEMA スレッドプールテーブル」](#)
- `CONNECTION_CONTROL` プラグインに固有の `INFORMATION_SCHEMA` テーブルに関する情報: [セクション 26.53 「INFORMATION_SCHEMA の接続制御テーブル」](#)
- `INFORMATION_SCHEMA` データベースに関するよくある質問への回答: [セクション A.7 「MySQL 8.0 FAQ: INFORMATION_SCHEMA」](#)
- `INFORMATION_SCHEMA` クエリーおよびオプティマイザ: [セクション 8.2.3 「INFORMATION_SCHEMA クエリーの最適化」](#)
- `INFORMATION_SCHEMA` 比較への照合の影響: [セクション 10.8.7 「INFORMATION_SCHEMA 検索での照合の使用」](#)

26.2 INFORMATION_SCHEMA ADMINISTRABLE_ROLE_AUTHORIZATIONS テーブル

`ADMINISTRABLE_ROLE_AUTHORIZATIONS` テーブル (MySQL 8.0.19 で使用可能) には、他のユーザーまたはロールに付与できる現在のユーザーまたはロールに適用可能なロールに関する情報が表示されます。

`ADMINISTRABLE_ROLE_AUTHORIZATIONS` テーブルには、次のカラムがあります:

- `USER`
現在のユーザーアカウントのユーザー名部分。
- `HOST`
現在のユーザーアカウントのホスト名部分。
- `GRANTEE`
ロールが付与されるアカウントのユーザー名部分。
- `GRANTEE_HOST`
ロールが付与されるアカウントのホスト名部分。
- `ROLE_NAME`
付与されたロールのユーザー名部分。
- `ROLE_HOST`

付与されたロールのホスト名部分。

- [IS_GRANTABLE](#)

ロールが他のアカウントに付与可能かどうかに応じて、[YES](#) または [NO](#)。

- [IS_DEFAULT](#)

ロールがデフォルトロールであるかどうかに応じて、[YES](#) または [NO](#)。

- [IS_MANDATORY](#)

ロールが必須かどうかに応じて、[YES](#) または [NO](#)。

26.3 INFORMATION_SCHEMA APPLICABLE_ROLES テーブル

[APPLICABLE_ROLES](#) テーブル (MySQL 8.0.19 で使用可能) には、現在のユーザーに適用可能なロールに関する情報が表示されます。

[APPLICABLE_ROLES](#) テーブルには、次のカラムがあります:

- [USER](#)

現在のユーザーアカウントのユーザー名部分。

- [HOST](#)

現在のユーザーアカウントのホスト名部分。

- [GRANTEE](#)

ロールが付与されるアカウントのユーザー名部分。

- [GRANTEE_HOST](#)

ロールが付与されるアカウントのホスト名部分。

- [ROLE_NAME](#)

付与されたロールのユーザー名部分。

- [ROLE_HOST](#)

付与されたロールのホスト名部分。

- [IS_GRANTABLE](#)

ロールが他のアカウントに付与可能かどうかに応じて、[YES](#) または [NO](#)。

- [IS_DEFAULT](#)

ロールがデフォルトロールであるかどうかに応じて、[YES](#) または [NO](#)。

- [IS_MANDATORY](#)

ロールが必須かどうかに応じて、[YES](#) または [NO](#)。

26.4 INFORMATION_SCHEMA CHARACTER_SETS テーブル

[CHARACTER_SETS](#) テーブルは、利用できる文字セットに関する情報を提供します。

[CHARACTER_SETS](#) テーブルには、次のカラムがあります:

- [CHARACTER_SET_NAME](#)
文字セット名。
- [DEFAULT_COLLATE_NAME](#)
文字セットのデフォルトの照合順序。
- [DESCRIPTION](#)
文字セットの説明。
- [MAXLEN](#)
1文字の格納に必要な最大バイト数。

メモ

文字セット情報は、[SHOW CHARACTER SET](#) ステートメントからも入手できます。[セクション13.7.7.3「SHOW CHARACTER SET ステートメント」](#)を参照してください。次のステートメントは同等です。

```
SELECT * FROM INFORMATION_SCHEMA.CHARACTER_SETS
[WHERE CHARACTER_SET_NAME LIKE 'wild']

SHOW CHARACTER SET
[LIKE 'wild']
```

26.5 INFORMATION_SCHEMA CHECK_CONSTRAINTS テーブル

[CHECK_CONSTRAINTS](#) テーブル (MySQL 8.0.16 で使用可能) は、テーブルに定義されている [CHECK](#) 制約に関する情報を提供します。

[CHECK_CONSTRAINTS](#) テーブルには、次のカラムがあります:

- [CONSTRAINT_CATALOG](#)
制約が属するカタログの名前。この値は常に `def` です。
- [CONSTRAINT_SCHEMA](#)
制約が属するスキーマ (データベース) の名前。
- [CONSTRAINT_NAME](#)
制約の名前。
- [CHECK_CLAUSE](#)
制約条件を指定する式。

26.6 INFORMATION_SCHEMA COLLATIONS テーブル

[COLLATIONS](#) テーブルは、各文字セットの照合順序に関する情報を提供します。

[COLLATIONS](#) テーブルには、次のカラムがあります:

- [COLLATION_NAME](#)
照合名。
- [CHARACTER_SET_NAME](#)
照合順序が関連付けられる文字セットの名前。

- ID

照合 ID。

- IS_DEFAULT

照合順序がその文字セットのデフォルトであるかどうか。

- IS_COMPILED

文字セットがサーバーにコンパイルされるかどうか。

- SORTLEN

これは、文字セットで表される文字列のソートに必要なメモリー量に関連します。

- PAD_ATTRIBUTE

照合パッド属性 (NO PAD または PAD SPACE)。この属性は、文字列比較で末尾の空白が重要かどうかに影響します。比較での後続領域の処理を参照してください。

メモ

照合順序情報は、SHOW COLLATION ステートメントから取得することもできます。セクション13.7.7.4「SHOW COLLATION ステートメント」を参照してください。次のステートメントは同等です。

```
SELECT COLLATION_NAME FROM INFORMATION_SCHEMA.COLLATIONS
[WHERE COLLATION_NAME LIKE 'wild']

SHOW COLLATION
[LIKE 'wild']
```

26.7 INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY テーブル

COLLATION_CHARACTER_SET_APPLICABILITY テーブルは、どの文字セットがどの照合順序に適用されるかを示します。

COLLATION_CHARACTER_SET_APPLICABILITY テーブルには、次のカラムがあります:

- COLLATION_NAME

照合名。

- CHARACTER_SET_NAME

照合順序が関連付けられる文字セットの名前。

メモ

COLLATION_CHARACTER_SET_APPLICABILITY のカラムは、SHOW COLLATION ステートメントで表示される最初の 2 つのカラムと同等です。

26.8 INFORMATION_SCHEMA COLUMNS テーブル

COLUMNS テーブルは、テーブル内のカラムに関する情報を提供します。関連する ST_GEOMETRY_COLUMNS テーブルは、空間データを格納するテーブルのカラムに関する情報を提供します。セクション 26.35「INFORMATION_SCHEMA ST_GEOMETRY_COLUMNS テーブル」を参照してください。

COLUMNS テーブルには、次のカラムがあります:

- **TABLE_CATALOG**
カラムを含むテーブルが属するカタログの名前。この値は常に **def** です。
- **TABLE_SCHEMA**
カラムを含むテーブルが属するスキーマ (データベース) の名前。
- **TABLE_NAME**
カラムを含むテーブルの名前。
- **COLUMN_NAME**
カラムの名前。
- **ORDINAL_POSITION**
テーブル内のカラムの位置。 **ORDINAL_POSITION** は **ORDER BY ORDINAL_POSITION** と記す場合があるため必要です。 **SHOW COLUMNS** とは異なり、 **COLUMNS** テーブルの **SELECT** には自動順序付けはありません。
- **COLUMN_DEFAULT**
カラムのデフォルト値。これは、カラムのデフォルトが明示的に **NULL** に設定されている場合、またはカラム定義に **DEFAULT** 句が含まれていない場合の **NULL** です。
- **IS_NULLABLE**
カラムの **NULL** 値可能性。この値は、 **NULL** 値をカラムに格納できる場合は **YES** で、格納できない場合は **NO** です。
- **DATA_TYPE**
カラムのデータ型。
DATA_TYPE 値はタイプ名のみで、他の情報はありません。 **COLUMN_TYPE** 値には、型名と、精度や長さなどのその他の情報が含まれます。
- **CHARACTER_MAXIMUM_LENGTH**
文字列カラムの場合、最大長 (文字数)。
- **CHARACTER_OCTET_LENGTH**
文字列カラムの場合、最大長 (バイト単位)。
- **NUMERIC_PRECISION**
数値カラムの場合、数値精度。
- **NUMERIC_SCALE**
数値カラムの場合、数値スケール。
- **DATETIME_PRECISION**
時間カラムの場合、小数秒精度。
- **CHARACTER_SET_NAME**
文字列カラムの場合、文字セット名。
- **COLLATION_NAME**
文字列カラムの場合、照合順序名。

- **COLUMN_TYPE**

カラムのデータ型。

DATA_TYPE 値はタイプ名のみで、他の情報はありません。**COLUMN_TYPE** 値には、型名と、精度や長さなどのその他の情報が含まれます。

- **COLUMN_KEY**

カラムがインデックス付けされているかどうか:

- **COLUMN_KEY** が空の場合、カラムはインデックス付けされていないが、複数カラムの非一意インデックスのセカンダリカラムとしてのみインデックス付けされます。
- **COLUMN_KEY** が **PRI** の場合、カラムは **PRIMARY KEY** であるが、複数カラム **PRIMARY KEY** のカラムのいずれかです。
- **COLUMN_KEY** が **UNI** の場合、カラムは **UNIQUE** インデックスの最初のカラムです。(**UNIQUE** インデックスでは複数の **NULL** 値が許可されますが、**Null** カラムをチェックすることで、カラムで **NULL** が許可されるかどうかを確認できます。)
- **COLUMN_KEY** が **MUL** の場合、カラムは一意でないインデックスの最初のカラムであり、特定の値の複数の出現がカラム内で許可されます。

複数の **COLUMN_KEY** 値がテーブルの特定のカラムに適用される場合、**COLUMN_KEY** では優先度が最も高い値が **PRI**, **UNI**, **MUL** の順序で表示されます。

UNIQUE インデックスは、**NULL** 値を含むことができず、かつテーブル内に **PRIMARY KEY** が存在しない場合は **PRI** として表示される可能性があります。**UNIQUE** インデックスは、複数のカラムが複合 **UNIQUE** インデックスを形成している場合は **MUL** として表示される可能性があります。このカラムの組み合わせは一意であるにもかかわらず、各カラムには引き続き、特定の値が複数回現れることがあります。

- **EXTRA**

特定のカラムについて使用可能な追加情報。次の場合、値は空ではありません:

- **AUTO_INCREMENT** 属性を持つカラムの **auto_increment**。
- **ON UPDATE CURRENT_TIMESTAMP** 属性を持つ **on update CURRENT_TIMESTAMP for TIMESTAMP** または **DATETIME** のカラム。
- 生成されたカラムの **STORED GENERATED** または **VIRTUAL GENERATED**。
- 式のデフォルト値を持つカラムの **DEFAULT_GENERATED**。

- **PRIVILEGES**

カラムに対して持っている権限。

- **COLUMN_COMMENT**

カラム定義に含まれるコメント。

- **GENERATION_EXPRESSION**

生成されたカラムの場合、カラム値の計算に使用される式を表示します。生成されないカラムの場合は空です。生成されるカラムの詳細は、[セクション13.1.20.8「CREATE TABLE および生成されるカラム」](#) を参照してください。

- **SRS_ID**

この値は空間カラムに適用されます。これには、カラムに格納されている値の空間参照システムを示すカラム **SRID** 値が含まれます。[セクション11.4.1「空間データ型」](#) および [セクション11.4.5「空間参照システムのサポート」](#) を参照してください。非空間カラムおよび **SRID** 属性のない空間カラムの場合、値は **NULL** です。

メモ

- `SHOW COLUMNS` では、`Type` 表示に複数の異なる `COLUMNS` カラムの値が含まれます。
- `CHARACTER_OCTET_LENGTH` は、マルチバイト文字セットを除き、`CHARACTER_MAXIMUM_LENGTH` と同じである必要があります。
- `CHARACTER_SET_NAME` は、`COLLATION_NAME` から導出できます。たとえば、`SHOW FULL COLUMNS FROM t` と言い、`COLLATION_NAME` カラムに `utf8_swedish_ci` の値が表示されている場合、文字セットは最初のアンダースコアの前にあります: `utf8`。

カラム情報は、`SHOW COLUMNS` ステートメントからも入手できます。 [セクション13.7.7.5「SHOW COLUMNS ステートメント」](#) を参照してください。 次のステートメントはほぼ同等です。

```
SELECT COLUMN_NAME, DATA_TYPE, IS_NULLABLE, COLUMN_DEFAULT
FROM INFORMATION_SCHEMA.COLUMNS
WHERE table_name = 'tbl_name'
[AND table_schema = 'db_name']
[AND column_name LIKE 'wild']

SHOW COLUMNS
FROM tbl_name
[FROM db_name]
[LIKE 'wild']
```

26.9 INFORMATION_SCHEMA COLUMNS_EXTENSIONS テーブル

`COLUMNS_EXTENSIONS` テーブル (MySQL 8.0.21 の時点で使用可能) は、プライマリストレージエンジンおよびセカンダリストレージエンジンに定義されているカラム属性に関する情報を提供します。

注記

`COLUMNS_EXTENSIONS` テーブルは、将来の使用のために予約されています。

`COLUMNS_EXTENSIONS` テーブルには、次のカラムがあります:

- `TABLE_CATALOG`
テーブルが属するカタログの名前。この値は常に `def` です。
- `TABLE_SCHEMA`
テーブルが属するスキーマ (データベース) の名前。
- `TABLE_NAME`
テーブルの名前。
- `COLUMN_NAME`
カラムの名前。
- `ENGINE_ATTRIBUTE`
プライマリストレージエンジンに定義されたカラム属性。将来使用するために予約されています。
- `SECONDARY_ENGINE_ATTRIBUTE`
セカンダリストレージエンジンに定義されたカラム属性。将来使用するために予約されています。

26.10 INFORMATION_SCHEMA COLUMN_PRIVILEGES テーブル

`COLUMN_PRIVILEGES` テーブルは、カラムの権限に関する情報を提供します。 `mysql.columns_priv` システムテーブルから値を取得します。

`COLUMN_PRIVILEGES` テーブルには、次のカラムがあります:

- `GRANTEE`

権限が付与されるアカウントの名前 ('user_name'@'host_name'形式)。

- `TABLE_CATALOG`

カラムを含むテーブルが属するカタログの名前。この値は常に `def` です。

- `TABLE_SCHEMA`

カラムを含むテーブルが属するスキーマ (データベース) の名前。

- `TABLE_NAME`

カラムを含むテーブルの名前。

- `COLUMN_NAME`

カラムの名前。

- `PRIVILEGE_TYPE`

付与された権限。値は、カラムレベルで付与できる任意の権限です。[セクション13.7.1.6「GRANT ステートメント」](#)を参照してください。各行には単一の権限がリストされるため、権限受領者が保持するカラム権限ごとに1つの行があります。

`SHOW FULL COLUMNS` からの出力では、権限はすべて1つのカラムにあり、`select,insert,update,references` などの小文字で表示されます。`COLUMN_PRIVILEGES` では、行ごとに1つの権限があり、大文字で記されます。

- `IS_GRANTABLE`

ユーザーが `GRANT OPTION` 権限を持っている場合は `YES`、それ以外の場合は `NO`。この出力では、`GRANT OPTION` は `PRIVILEGE_TYPE='GRANT OPTION'` とは別の行としてリストされません。

メモ

- `COLUMN_PRIVILEGES` は非標準の `INFORMATION_SCHEMA` テーブルです。

次のステートメントは同等ではありません。

```
SELECT ... FROM INFORMATION_SCHEMA.COLUMN_PRIVILEGES
SHOW GRANTS ...
```

26.11 INFORMATION_SCHEMA COLUMN_STATISTICS テーブル

`COLUMN_STATISTICS` テーブルでは、カラム値のヒストグラム統計にアクセスできます。

ヒストグラム統計の詳細は、[セクション8.9.6「オプティマイザ統計」](#) および [セクション13.7.3.1「ANALYZE TABLE ステートメント」](#)を参照してください。

一部の権限を持つカラムの情報のみを表示できます。

`COLUMN_STATISTICS` テーブルには、次のカラムがあります:

- `SCHEMA_NAME`

統計が適用されるスキーマの名前。

- `TABLE_NAME`

統計が適用されるカラムの名前。

- **COLUMN_NAME**

統計が適用されるカラムの名前。

- **HISTOGRAM**

ヒストグラムとして格納される、カラム統計を記述する **JSON** オブジェクト。

26.12 INFORMATION_SCHEMA ENABLED_ROLES テーブル

ENABLED_ROLES テーブル (MySQL 8.0.19 で使用可能) には、現在のセッション内で有効になっているロールに関する情報が表示されます。

ENABLED_ROLES テーブルには、次のカラムがあります:

- **ROLE_NAME**

付与されたロールのユーザー名部分。

- **ROLE_HOST**

付与されたロールのホスト名部分。

- **IS_DEFAULT**

ロールがデフォルトロールであるかどうかに応じて、**YES** または **NO**。

- **IS_MANDATORY**

ロールが必須かどうかに応じて、**YES** または **NO**。

26.13 INFORMATION_SCHEMA ENGINES テーブル

ENGINES テーブルは、ストレージエンジンに関する情報を提供します。これは、ストレージエンジンがサポートされているかどうかをチェックしたり、デフォルトのエンジンが何であるかを確認したりするために特に役立ちます。

ENGINES テーブルには、次のカラムがあります:

- **ENGINE**

ストレージエンジンの名前。

- **SUPPORT**

次のテーブルに示すストレージエンジンのサーバーレベルのサポート。

値	意味
行う	このエンジンはサポートされており、アクティブです
DEFAULT	YES と同様であることに加え、これがデフォルトのエンジンです
NO	このエンジンはサポートされていません
DISABLED	このエンジンはサポートされていますが、無効になっています

NO の値は、サーバーがこのエンジンに対するサポートなしでコンパイルされたことを示すため、実行時にこのエンジンを有効にすることはできません。

DISABLED の値は、サーバーがこのエンジンを無効にするオプションを使用して起動されたか、またはこのエンジンを有効にするために必要な一部のオプションが指定されなかったために発生します。後者の場合、エラーログにはオプションが無効になっている理由が含まれている必要があります。 [セクション5.4.2「エラーログ」](#) を参照してください。

ストレージエンジンに対する `DISABLED` はまた、サーバーがそれをサポートするようにコンパイルされたが、`--skip-engine_name` オプションで起動された場合にも表示される可能性があります。NDB ストレージエンジンの場合、`DISABLED` はサーバーが NDB Cluster をサポートしてコンパイルされたが、`--ndbcluster` オプションで起動されなかったことを意味します。

すべての MySQL サーバーで `MyISAM` テーブルがサポートされます。 `MyISAM` を無効にすることはできません。

- `COMMENT`

ストレージエンジンの簡単な説明。

- `TRANSACTIONS`

ストレージエンジンがトランザクションをサポートするかどうか。

- `XA`

ストレージエンジンが XA トランザクションをサポートするかどうか。

- `SAVEPOINTS`

ストレージエンジンがセーブポイントをサポートするかどうか。

メモ

- `ENGINES` は非標準の `INFORMATION_SCHEMA` テーブルです。

ストレージエンジンの情報は、`SHOW ENGINES` ステートメントからも入手できます。 [セクション 13.7.7.16 「SHOW ENGINES ステートメント」](#) を参照してください。 次のステートメントは同等です。

```
SELECT * FROM INFORMATION_SCHEMA.ENGINES
SHOW ENGINES
```

26.14 INFORMATION_SCHEMA EVENTS テーブル

`EVENTS` テーブルには、 [セクション 25.4 「イベントスケジューラの使用」](#) で説明されているイベントマネージャイベントに関する情報が表示されます。

`EVENTS` テーブルには、次のカラムがあります：

- `EVENT_CATALOG`

イベントが属するカタログの名前。 この値は常に `def` です。

- `EVENT_SCHEMA`

イベントが属するスキーマ (データベース) の名前。

- `EVENT_NAME`

イベントの名前。

- `DEFINER`

'`user_name`'@'`host_name`'形式の、`DEFINER` 句で指定されたアカウント (多くの場合、イベントを作成したユーザー)。

- `TIME_ZONE`

イベントのタイムゾーン。 イベントのスケジュールに使用され、イベントの実行時にイベント内で有効なタイムゾーンです。 デフォルト値は `SYSTEM` です。

- `EVENT_BODY`

イベント `DO` 句のステートメントに使用される言語。値は常に `SQL` です。

- `EVENT_DEFINITION`

イベント `DO` 句を構成する SQL ステートメントのテキスト。つまり、このイベントによって実行されるステートメント。

- `EVENT_TYPE`

イベントの繰り返しタイプ。 `ONE TIME` (一時) または `RECURRING` (繰り返し)。

- `EXECUTE_AT`

ワンタイムイベントの場合、これは、イベントの作成に使用される `CREATE EVENT` ステートメントの `AT` 句、またはイベントを変更した最後の `ALTER EVENT` ステートメントで指定された `DATETIME` 値です。このカラムに表示された値は、イベントの `AT` 句に含まれた、`INTERVAL` 値の加算または減算に影響します。たとえば、イベントが `ON SCHEDULE AT CURRENT_TIMESTAMP + '1:6' DAY_HOUR` を使用して作成され、イベントが 2018-02-09 の 14:05:30 に作成された場合、カラムに表示される値は '2018-02-10 20:05:30' になります。イベントのタイミングが `AT` 句ではなく `EVERY` 句で決定される場合 (つまり、イベントが繰り返しである場合)、このカラムの値は `NULL` になります。

- `INTERVAL_VALUE`

繰り返しイベントの場合、イベント実行間で待機する間隔の数。一時イベントの場合、値は常に `NULL` です。

- `INTERVAL_FIELD`

繰り返しイベントが繰り返される前に待機する間隔に使用される時間単位。一時イベントの場合、値は常に `NULL` です。

- `SQL_MODE`

イベントが作成または変更され、イベントが実行されるときに有効な SQL モード。指定可能な値については、[セクション5.1.11「サーバー SQL モード」](#)を参照してください。

- `STARTS`

繰り返しイベントの開始日時。これは `DATETIME` 値として表示され、このイベントの開始日付と開始時間が定義されていない場合は `NULL` です。一時的なイベントの場合、このカラムは常に `NULL` です。定義に `STARTS` 句が含まれる繰り返しイベントの場合、このカラムには対応する `DATETIME` 値が含まれます。 `EXECUTE_AT` カラムの場合と同様に、この値は使用されている式を解きます。イベントのタイミングに影響する `STARTS` 句がない場合、このカラムは `NULL` です。

- `ENDS`

定義に `ENDS` 句が含まれる繰り返しイベントの場合、このカラムには対応する `DATETIME` 値が含まれます。 `EXECUTE_AT` カラムの場合と同様に、この値は使用されている式を解きます。イベントのタイミングに影響する `ENDS` 句がない場合、このカラムは `NULL` です。

- `STATUS`

イベントステータス。 `ENABLED`、`DISABLED`、`SLAVESIDE_DISABLED` のいずれか。 `SLAVESIDE_DISABLED` は、イベントの作成が、レプリケーションソースとして機能する別の MySQL サーバーで発生し、レプリカとして機能している現在の MySQL サーバーにレプリケートされたが、そのイベントがレプリカで現在実行されていないことを示します。詳細は、[セクション17.5.1.16「呼び出される機能のレプリケーション」](#)の情報を参照してください。

- `ON_COMPLETION`

`PRESERVE` または `NOT PRESERVE` のいずれかの値。

- `CREATED`

イベントが作成された日時。これは `TIMESTAMP` 値です。

- **LAST_ALTERED**

イベントが最後に変更された日時。これは **TIMESTAMP** 値です。イベントが作成されてから変更されていない場合、この値は **CREATED** の値と同じです。

- **LAST_EXECUTED**

イベントが最後に実行された日時。これは **DATETIME** 値です。イベントが一度も実行されていない場合、このカラムは **NULL** です。

LAST_EXECUTED はイベントが開始した時点を示します。このため、**ENDS** カラムが **LAST_EXECUTED** より小さくなることは決してありません。

- **EVENT_COMMENT**

コメントのテキスト (イベントにコメントがある場合)。そうでない場合、この値は空です。

- **ORIGINATOR**

イベントが作成された MySQL サーバーのサーバー ID。レプリケーションで使用されます。この値は、レプリケーションソースで実行された場合、**ALTER EVENT** によって、そのステートメントが発生したサーバーのサーバー ID に更新されることがあります。デフォルト値は 0 です。

- **CHARACTER_SET_CLIENT**

イベント作成時の **character_set_client** システム変数のセッション値。

- **COLLATION_CONNECTION**

イベント作成時の **collation_connection** システム変数のセッション値。

- **DATABASE_COLLATION**

イベントが関連付けられているデータベースの照合。

メモ

- **EVENTS** は非標準の **INFORMATION_SCHEMA** テーブルです。
- **EVENTS** テーブルの時間は、[セクション25.4.4「イベントメタデータ」](#) で説明されているように、イベントタイムゾーン、現行セッションタイムゾーンまたは UTC を使用して表示されます。
- **SLAVESIDE_DISABLED** および **ORIGINATOR** カラムの詳細は、[セクション17.5.1.16「呼び出される機能のレプリケーション」](#) を参照してください。

例

次に示すように、ユーザー'**jon'@'ghidora'**が **e_daily** という名前のイベントを作成し、**ALTER EVENT** ステートメントを使用して数分後に変更するとします:

```
DELIMITER |
CREATE EVENT e_daily
ON SCHEDULE
  EVERY 1 DAY
COMMENT 'Saves total number of sessions then clears the table each day'
DO
BEGIN
  INSERT INTO site_activity.totals (time, total)
  SELECT CURRENT_TIMESTAMP, COUNT(*)
  FROM site_activity.sessions;
  DELETE FROM site_activity.sessions;
END |
DELIMITER ;
```

```
ALTER EVENT e_daily  
ENABLE;
```

(コメントは複数の行にわたって記述できます。)

このユーザーは続いて次の **SELECT** ステートメントを実行し、次の出力が表示されます。

```
mysql> SELECT * FROM INFORMATION_SCHEMA.EVENTS  
WHERE EVENT_NAME = 'e_daily'  
AND EVENT_SCHEMA = 'myschema'\G  
***** 1. row *****  
EVENT_CATALOG: def  
EVENT_SCHEMA: myschema  
EVENT_NAME: e_daily  
DEFINER: jon@ghidora  
TIME_ZONE: SYSTEM  
EVENT_BODY: SQL  
EVENT_DEFINITION: BEGIN  
INSERT INTO site_activity.totals (time, total)  
SELECT CURRENT_TIMESTAMP, COUNT(*)  
FROM site_activity.sessions;  
DELETE FROM site_activity.sessions;  
END  
EVENT_TYPE: RECURRING  
EXECUTE_AT: NULL  
INTERVAL_VALUE: 1  
INTERVAL_FIELD: DAY  
SQL_MODE: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,  
NO_ZERO_IN_DATE,NO_ZERO_DATE,  
ERROR_FOR_DIVISION_BY_ZERO,  
NO_ENGINE_SUBSTITUTION  
STARTS: 2018-08-08 11:06:34  
ENDS: NULL  
STATUS: ENABLED  
ON_COMPLETION: NOT PRESERVE  
CREATED: 2018-08-08 11:06:34  
LAST_ALTERED: 2018-08-08 11:06:34  
LAST_EXECUTED: 2018-08-08 16:06:34  
EVENT_COMMENT: Saves total number of sessions then clears the  
table each day  
ORIGINATOR: 1  
CHARACTER_SET_CLIENT: utf8mb4  
COLLATION_CONNECTION: utf8mb4_0900_ai_ci  
DATABASE_COLLATION: utf8mb4_0900_ai_ci
```

イベント情報は、**SHOW EVENTS** ステートメントからも入手できます。 [セクション13.7.7.18「SHOW EVENTS ステートメント」](#)を参照してください。 次のステートメントは同等です。

```
SELECT  
EVENT_SCHEMA, EVENT_NAME, DEFINER, TIME_ZONE, EVENT_TYPE, EXECUTE_AT,  
INTERVAL_VALUE, INTERVAL_FIELD, STARTS, ENDS, STATUS, ORIGINATOR,  
CHARACTER_SET_CLIENT, COLLATION_CONNECTION, DATABASE_COLLATION  
FROM INFORMATION_SCHEMA.EVENTS  
WHERE table_schema = 'db_name'  
[AND column_name LIKE 'wild']  
  
SHOW EVENTS  
[FROM db_name]  
[LIKE 'wild']
```

26.15 INFORMATION_SCHEMA FILES テーブル

FILES テーブルには、MySQL テーブルスペースデータが格納されているファイルに関する情報が表示されます。

FILES テーブルには、**InnoDB** データファイルに関する情報が表示されます。 NDB Cluster では、このテーブルは「NDB Cluster データ」テーブルが格納されているファイルに関する情報も提供します。 **InnoDB** に固有の追加情報については、このセクションの後半の **InnoDB の注意** を参照してください。 NDB Cluster に固有の追加情報については、**NDB に関する注意事項** を参照してください。

FILES テーブルには、次のカラムがあります:

- **FILE_ID**

InnoDB の場合: `space_id` または `fil_space_t::id` と呼ばれるテーブルスペース ID。

NDB の場合: ファイル識別子。 `FILE_ID` カラムの値は自動生成されます。

- **FILE_NAME**

InnoDB の場合: データファイルの名前。 File-per-table および general テーブルスペースには、`.ibd` ファイル名拡張子が付いています。 undo テーブルスペースには、接頭辞として `undo` が付きます。 システムテーブルスペースには、接頭辞として `ibdata` が付きます。 グローバル一時テーブルスペースには、接頭辞として `ibtmp` が付きます。 ファイル名にはファイルパスが含まれており、これは MySQL データディレクトリ (`datadir` システム変数の値) に対して相対である可能性があります。

NDB の場合: `CREATE LOGFILE GROUP` または `ALTER LOGFILE GROUP` によって作成された undo ログファイル、あるいは `CREATE TABLESPACE` または `ALTER TABLESPACE` によって作成されたデータファイルの名前。 NDB 8.0 では、ファイル名は相対パスで表示されます。 undo ログファイルの場合、このパスはディレクトリ `DataDir/ndb_Nodeid_fs/LG` を基準とした相対パスで、データファイルの場合はディレクトリ `DataDir/ndb_Nodeid_fs/TS` を基準とした相対パスです。 これは、たとえば、`ALTER TABLESPACE ts ADD DATAFILE 'data_2.dat' INITIAL SIZE 256M` で作成されたデータファイルの名前が `./data_2.dat` として表示されることを意味します。

- **FILE_TYPE**

InnoDB の場合: テーブルスペースのファイルタイプ。 InnoDB ファイルには、3つのファイルタイプがあります。 `TABLESPACE` は、テーブル、インデックスまたはその他の形式のユーザーデータを保持するシステムテーブルスペースファイル、一般テーブルスペースファイルまたはファイルごとのテーブルスペースファイルのファイルタイプです。 `TEMPORARY` は、一時テーブルスペースのファイルタイプです。 `UNDO LOG` は、undo レコードを保持する undo テーブルスペースのファイルタイプです。

NDB の場合: `UNDO LOG` または `DATAFILE` のいずれかの値。 NDB 8.0.13 より前では、`TABLESPACE` も可能な値でした。

- **TABLESPACE_NAME**

ファイルが関連付けられているテーブルスペースの名前。

InnoDB の場合: 一般的なテーブルスペース名は、作成時に指定したとおりです。 テーブルごとのファイルテーブルスペース名は、次の形式で表示されます: `schema_name/table_name`。 InnoDB システムのテーブルスペース名は `innodb_system` です。 グローバル一時テーブルスペース名は `innodb_temporary` です。 デフォルトの UNDO テーブルスペース名は、`innodb_undo_001` および `innodb_undo_002` です。 ユーザーが作成した UNDO テーブルスペースの名前は、作成時に指定したとおりです。

- **TABLE_CATALOG**

この値は常に空です。

- **TABLE_SCHEMA**

これは常に `NULL` です。

- **TABLE_NAME**

これは常に `NULL` です。

- **LOGFILE_GROUP_NAME**

InnoDB の場合: これは常に `NULL` です。

NDB の場合: ログファイルまたはデータファイルが属するログファイルグループの名前。

- **LOGFILE_GROUP_NUMBER**

InnoDB の場合: これは常に `NULL` です。

NDB の場合: ディスクデータ undo ログファイルの場合、ログファイルが属するログファイルグループの自動生成された ID 番号。これは、この undo ログファイルの `ndbinfo.dict_obj_info` テーブルの `id` カラムおよび `ndbinfo.logspaces` テーブルおよび `ndbinfo.logspaces` テーブルの `log_id` カラムに表示される値と同じです。

- `ENGINE`

InnoDB の場合: この値は常に `InnoDB` です。

NDB の場合: この値は常に `ndbcluster` です。

- `FULLTEXT_KEYS`

これは常に `NULL` です。

- `DELETED_ROWS`

これは常に `NULL` です。

- `UPDATE_COUNT`

これは常に `NULL` です。

- `FREE_EXTENTS`

InnoDB の場合: 現在のデータファイル内の完全に使用可能なエクステントの数。

NDB の場合: ファイルでまだ使用されていないエクステントの数。

- `TOTAL_EXTENTS`

InnoDB の場合: 現在のデータファイルで使用されている全エクステントの数。ファイルの最後にある部分エクステントはカウントされません。

NDB の場合: ファイルに割り当てられたエクステントの合計数。

- `EXTENT_SIZE`

InnoDB の場合: ページサイズが 4KB、8KB または 16KB のファイルのエクステントサイズは 1048576 (1MB) です。エクステントサイズは、32KB のページサイズのファイルの場合 2097152 バイト (2MB)、64KB のページサイズのファイルの場合 4194304 (4MB) です。FILES では、InnoDB のページサイズはレポートされません。ページサイズは、`innodb_page_size` システム変数によって定義されます。エクステントサイズ情報は、`FILES.FILE_ID = INNODB_TABLESPACES.SPACE` が存在する `INNODB_TABLESPACES` テーブルから取得することもできます。

NDB の場合: ファイルのエクステントのサイズ (バイト単位)。

- `INITIAL_SIZE`

InnoDB の場合: ファイルの初期サイズ (バイト単位)。

NDB の場合: ファイルのサイズ (バイト単位)。これは、ファイルの作成に使用された `CREATE LOGFILE GROUP`、`ALTER LOGFILE GROUP`、`CREATE TABLESPACE`、または `ALTER TABLESPACE` ステートメントの `INITIAL_SIZE` 句に使用された値と同じです。

- `MAXIMUM_SIZE`

InnoDB の場合: ファイルで許可される最大バイト数。事前定義済のシステムテーブルスペースデータファイルを除くすべてのデータファイルの値は `NULL` です。システムテーブルスペースの最大ファイルサイズは、`innodb_data_file_path` によって定義されます。グローバル一時テーブルスペースの最大ファイルサイズは、`innodb_temp_data_file_path` によって定義されます。事前定義済のシステムテーブルスペースデータファイルの `NULL` 値は、ファイルサイズ制限が明示的に定義されていないことを示します。

NDB の場合: この値は、常に `INITIAL_SIZE` の値と同じです。

- `AUTOEXTEND_SIZE`

テーブルスペースの自動拡張サイズ。NDB の場合、AUTOEXTEND_SIZE は常に NULL です。

- CREATION_TIME

これは常に NULL です。

- LAST_UPDATE_TIME

これは常に NULL です。

- LAST_ACCESS_TIME

これは常に NULL です。

- RECOVER_TIME

これは常に NULL です。

- TRANSACTION_COUNTER

これは常に NULL です。

- VERSION

InnoDB の場合: これは常に NULL です。

NDB の場合: ファイルのバージョン番号。

- ROW_FORMAT

InnoDB の場合: これは常に NULL です。

NDB の場合: FIXED または DYNAMIC のいずれか。

- TABLE_ROWS

これは常に NULL です。

- AVG_ROW_LENGTH

これは常に NULL です。

- DATA_LENGTH

これは常に NULL です。

- MAX_DATA_LENGTH

これは常に NULL です。

- INDEX_LENGTH

これは常に NULL です。

- DATA_FREE

InnoDB の場合: テーブルスペース全体の空き領域の合計量 (バイト)。システムテーブルスペースおよび一時テーブルのテーブルスペースを含む事前定義済のシステムテーブルスペースには、1 つ以上のデータファイルを含めることができます。

NDB の場合: これは常に NULL です。

- CREATE_TIME

これは常に NULL です。

- UPDATE_TIME

これは常に NULL です。

- CHECK_TIME

これは常に NULL です。

- CHECKSUM

これは常に NULL です。

- STATUS

InnoDB の場合: デフォルトでは、この値は **NORMAL** です。InnoDB file-per-table テーブルスペースは、テーブルスペースがまだ使用できないことを示す **IMPORTING** を報告する場合があります。

NDB の場合: NDB Cluster データファイルの場合、この値は常に **NORMAL** です。

- EXTRA

InnoDB の場合: これは常に NULL です。

NDB の場合: (NDB 8.0.15 以降) undo ログファイルの場合、このカラムには undo ログバッファサイズが表示され、データファイルの場合は常に NULL になります。詳細は、次のいくつかの段落で説明します。

NDBCLUSTER では、各データファイルと各 undo ログファイルのコピーがクラスタ内の各データノードに格納されます。NDB 8.0.13 以降では、**FILES** テーブルにはそのようなファイルごとに 1 つの行のみが含まれます。4 つのデータノードを持つ NDB Cluster で次の 2 つのステートメントを実行するとします:

```
CREATE LOGFILE GROUP mygroup
  ADD UNDOFILE 'new_undo.dat'
  INITIAL_SIZE 2G
  ENGINE NDBCLUSTER;

CREATE TABLESPACE myts
  ADD DATAFILE 'data_1.dat'
  USE LOGFILE GROUP mygroup
  INITIAL_SIZE 256M
  ENGINE NDBCLUSTER;
```

これら 2 つのステートメントを正常に実行すると、**FILES** テーブルに対する次のクエリーのような結果が表示されます:

```
mysql> SELECT LOGFILE_GROUP_NAME, FILE_TYPE, EXTRA
-> FROM INFORMATION_SCHEMA.FILES
-> WHERE ENGINE = 'ndbcluster';
```

LOGFILE_GROUP_NAME	FILE_TYPE	EXTRA
mygroup	UNDO LOG	UNDO_BUFFER_SIZE=8388608
mygroup	DATAFILE	NULL

undo ログバッファサイズ情報が NDB 8.0.13 で誤って削除されましたが、NDB 8.0.15 で復元されました。(Bug #92796、Bug #28800252)

NDB 8.0.13 より前では、**FILES** テーブルには、ファイルが属する各データノード上のこれらの各ファイルの行と、その Undo バッファのサイズが含まれていました。これらのバージョンでは、次に示すように、同じクエリーの結果にデータノードごとに 1 つの行が含まれます:

LOGFILE_GROUP_NAME	FILE_TYPE	EXTRA
mygroup	UNDO LOG	CLUSTER_NODE=5;UNDO_BUFFER_SIZE=8388608
mygroup	UNDO LOG	CLUSTER_NODE=6;UNDO_BUFFER_SIZE=8388608
mygroup	UNDO LOG	CLUSTER_NODE=7;UNDO_BUFFER_SIZE=8388608
mygroup	UNDO LOG	CLUSTER_NODE=8;UNDO_BUFFER_SIZE=8388608

```
| mygroup | DATAFILE | CLUSTER_NODE=5 |
| mygroup | DATAFILE | CLUSTER_NODE=6 |
| mygroup | DATAFILE | CLUSTER_NODE=7 |
| mygroup | DATAFILE | CLUSTER_NODE=8 |
+-----+-----+-----+
```

メモ

- **FILES** は非標準の **INFORMATION_SCHEMA** テーブルです。
- MySQL 8.0.21 では、このテーブルをクエリーするには **PROCESS** 権限が必要です。

InnoDB の注意

InnoDB データファイルには、次のノートが適用されます。

- **FILES** によってレポートされるデータは、オープンファイルの **InnoDB** インメモリキャッシュからレポートされます。比較すると、**INNODB_DATAFILES** は **InnoDB SYS_DATAFILES** 内部データディクショナリテーブルのデータをレポートします。
- **FILES** によってレポートされるデータには、グローバル一時テーブルスペースデータが含まれます。このデータは、**InnoDB SYS_DATAFILES** 内部データディクショナリテーブルでは使用できないため、**INNODB_DATAFILES** ではレポートされません。
- undo テーブルスペースデータは、個別の undo テーブルスペースが存在する場合に **FILES** によってレポートされます。undo テーブルスペースデータは、デフォルトでは MySQL 8.0 にあります
- 次のクエリーは、**InnoDB** テーブルスペースに関連するすべてのデータを返します。

```
SELECT
FILE_ID, FILE_NAME, FILE_TYPE, TABLESPACE_NAME, FREE_EXTENTS,
TOTAL_EXTENTS, EXTENT_SIZE, INITIAL_SIZE, MAXIMUM_SIZE,
AUTOEXTEND_SIZE, DATA_FREE, STATUS
FROM INFORMATION_SCHEMA.FILES WHERE ENGINE='InnoDB'\G
```

NDB に関する注意事項

- **FILES** テーブルには、ディスクデータ files に関する情報のみが表示されます。個々の **NDB** テーブルのディスク領域割当てまたは可用性の決定には使用できません。ただし、**ndb_desc** を使用して、データがディスク上に格納された **NDB** テーブルごとにどれだけの領域が割り当てられているかとともに、そのテーブルに対しディスク上のデータのストレージに利用できる領域がどれだけ残っているかも表示できます。
- **CREATION_TIME**、**LAST_UPDATE_TIME** および **LAST_ACCESSED** の値は、オペレーティングシステムによってレポートされ、**NDB** ストレージエンジンによって提供されません。オペレーティングシステムによって値が指定されていない場合、これらのカラムには **NULL** が表示されます。
- **TOTAL_EXTENTS** カラムと **FREE_EXTENTS** カラムの違いは、ファイルで現在使用されているエクステントの数です:

```
SELECT TOTAL_EXTENTS - FREE_EXTENTS AS extents_used
FROM INFORMATION_SCHEMA.FILES
WHERE FILE_NAME = './myfile.dat';
```

ファイルで使用されているディスク領域の量を概算するには、その差に **EXTENT_SIZE** カラムの値を乗算します。これにより、ファイルのエクステントのサイズがバイト単位で指定されます:

```
SELECT (TOTAL_EXTENTS - FREE_EXTENTS) * EXTENT_SIZE AS bytes_used
FROM INFORMATION_SCHEMA.FILES
WHERE FILE_NAME = './myfile.dat';
```

同様に、**FREE_EXTENTS** と **EXTENT_SIZE** とを乗算すると、特定のファイルに利用できる残りの領域の容量を見積もることができます。

```
SELECT FREE_EXTENTS * EXTENT_SIZE AS bytes_free
FROM INFORMATION_SCHEMA.FILES
```

```
WHERE FILE_NAME = './myfile.dat';
```

重要

前述のクエリーで生成されたバイト値は概算に過ぎず、その精度は `EXTENT_SIZE` の値に反比例します。つまり、`EXTENT_SIZE` が大きくなれば、概算の精度は低くなります。

エクステントがいったん使用されると、エクステントが含まれているデータファイルを破棄せずに再度解放することはできません。これにより、ディスクデータのテーブルからの削除はディスクスペースを解放しないこととなります。

エクステントサイズは `CREATE TABLESPACE` ステートメントで設定できます。詳細は、[セクション 13.1.21 「CREATE TABLESPACE ステートメント」](#) を参照してください。

- NDB 8.0.13 より前では、ログファイルグループの作成後に、`FILE_NAME` カラムに `NULL` がある `FILES` テーブルに追加の行が存在していました。NDB 8.0.13 以降では、どのファイルにも対応していないこの行は表示されなくなり、`ndbinfo.logspaces` テーブルをクエリーして undo ログファイルの使用状況情報を取得する必要があります。詳細は、このテーブルおよび [セクション 23.5.10.1 「NDB Cluster ディスクデータオブジェクト」](#) の説明を参照してください。

この項目の残りの説明は NDB 8.0.12 以前にのみ適用されます。`FILE_NAME` カラムに `NULL` がある行では、`FILE_ID` カラムの値は常に 0 で、`FILE_TYPE` カラムの値は常に `UNDO LOG` で、`STATUS` カラムの値は常に `NORMAL` です。`ENGINE` カラムの値は常に `ndbcluster` です。

この行の `FREE_EXTENTS` カラムには、名前と番号がそれぞれ `LOGFILE_GROUP_NAME` カラムと `LOGFILE_GROUP_NUMBER` カラムに表示される特定のログファイルグループに属す、すべての Undo ファイルで利用可能な空きエクステントの合計数が表示されます。

NDB Cluster に既存のログファイルグループがなく、次のステートメントを使用して作成するとします：

```
mysql> CREATE LOGFILE GROUP lg1
      ADD UNDOFILE 'undofile.dat'
      INITIAL_SIZE = 16M
      UNDO_BUFFER_SIZE = 1M
      ENGINE = NDB;
```

`FILES` テーブルにクエリーすると、`NULL` 行が表示されます。

```
mysql> SELECT DISTINCT
      FILE_NAME AS File,
      FREE_EXTENTS AS Free,
      TOTAL_EXTENTS AS Total,
      EXTENT_SIZE AS Size,
      INITIAL_SIZE AS Initial
      FROM INFORMATION_SCHEMA.FILES;
+-----+-----+-----+-----+-----+
| File   | Free  | Total | Size | Initial |
+-----+-----+-----+-----+-----+
| undofile.dat | NULL | 4194304 | 4 | 16777216 |
| NULL    | 4184068 | NULL | 4 | NULL |
+-----+-----+-----+-----+-----+
```

Undo ロギングに利用できる空きエクステントの総数は常に、Undo ファイルの維持に必要なオーバーヘッドのために、ログファイルグループ内のすべての Undo ファイルの `TOTAL_EXTENTS` カラムの値の合計よりもいくぶん少なくなります。これは、ログファイルグループに 2 番目の Undo ファイルを追加してから、`FILES` テーブルに対して前述のクエリーを繰り返すと表示できます。

```
mysql> ALTER LOGFILE GROUP lg1
      ADD UNDOFILE 'undofile02.dat'
      INITIAL_SIZE = 4M
      ENGINE = NDB;

mysql> SELECT DISTINCT
      FILE_NAME AS File,
      FREE_EXTENTS AS Free,
      TOTAL_EXTENTS AS Total,
      EXTENT_SIZE AS Size,
      INITIAL_SIZE AS Initial
```

```

FROM INFORMATION_SCHEMA.FILES;
+-----+-----+-----+-----+
| File      | Free  | Total | Size | Initial |
+-----+-----+-----+-----+
| undofile.dat | NULL | 4194304 | 4 | 16777216 |
| undofile02.dat | NULL | 1048576 | 4 | 4194304 |
| NULL      | 5223944 | NULL | 4 | NULL |
+-----+-----+-----+-----+

```

このログファイルグループを使用したディスクデータテーブルが Undo ロギングに利用できる空き領域の容量 (バイト単位) は、空きエクステンツの数と初期サイズとを乗算すると概算できます。

```

mysql> SELECT
FREE_EXTENTS AS 'Free Extents',
FREE_EXTENTS * EXTENT_SIZE AS 'Free Bytes'
FROM INFORMATION_SCHEMA.FILES
WHERE LOGFILE_GROUP_NAME = 'lg1'
AND FILE_NAME IS NULL;
+-----+-----+
| Free Extents | Free Bytes |
+-----+-----+
| 5223944 | 20895776 |
+-----+-----+

```

「NDB Cluster ディスクデータ」テーブルを作成し、そのテーブルにいくつかの行を挿入すると、次の例のように、後で undo ロギング用に残っている領域の量を確認できます：

```

mysql> CREATE TABLESPACE ts1
ADD DATAFILE 'data1.dat'
USE LOGFILE GROUP lg1
INITIAL_SIZE 512M
ENGINE = NDB;

mysql> CREATE TABLE dd (
c1 INT NOT NULL PRIMARY KEY,
c2 INT,
c3 DATE
)
TABLESPACE ts1 STORAGE DISK
ENGINE = NDB;

mysql> INSERT INTO dd VALUES
(NULL, 1234567890, '2007-02-02'),
(NULL, 1126789005, '2007-02-03'),
(NULL, 1357924680, '2007-02-04'),
(NULL, 1642097531, '2007-02-05');

mysql> SELECT
FREE_EXTENTS AS 'Free Extents',
FREE_EXTENTS * EXTENT_SIZE AS 'Free Bytes'
FROM INFORMATION_SCHEMA.FILES
WHERE LOGFILE_GROUP_NAME = 'lg1'
AND FILE_NAME IS NULL;
+-----+-----+
| Free Extents | Free Bytes |
+-----+-----+
| 5207565 | 20830260 |
+-----+-----+

```

- NDB 8.0.13 より前は、「NDB Cluster ディスクデータ」テーブルスペースごとに追加の行が `FILES` テーブルに存在していました。実際のファイルに対応していなかったため、NDB 8.0.13 で削除されました。この行には `FILE_NAME` カラムの値に対する `NULL` があり、`FILE_ID` カラムの値は常に `0` で、`FILE_TYPE` カラムの値は常に `TABLESPACE` で、`STATUS` カラムの値は `NORMAL` であり、`ENGINE` カラムの値は常に `NDBCLUSTER` です。

NDB 8.0.13 以降では、`ndb_desc` ユーティリティを使用して「ディスクデータ」テーブルスペースに関する情報を取得できます。詳細は、[セクション23.5.10.1「NDB Cluster ディスクデータオブジェクト」](#) および `ndb_desc` の説明を参照してください。

- NDB Cluster ディスクデータオブジェクトに関する追加情報および情報の作成、削除、および取得の例については、[セクション23.5.10「NDB Cluster ディスクデータテーブル」](#) を参照してください。

26.16 INFORMATION_SCHEMA KEY_COLUMN_USAGE テーブル

[KEY_COLUMN_USAGE](#) テーブルは、どのキーカラムに制約があるかを説明します。このテーブルは式であり、カラムに関する情報のみを提供するため、関数キー部分に関する情報は提供しません。

[KEY_COLUMN_USAGE](#) テーブルには、次のカラムがあります:

- [CONSTRAINT_CATALOG](#)
制約が属するカタログの名前。この値は常に `def` です。
- [CONSTRAINT_SCHEMA](#)
制約が属するスキーマ (データベース) の名前。
- [CONSTRAINT_NAME](#)
制約の名前。
- [TABLE_CATALOG](#)
テーブルが属するカタログの名前。この値は常に `def` です。
- [TABLE_SCHEMA](#)
テーブルが属するスキーマ (データベース) の名前。
- [TABLE_NAME](#)
制約があるテーブルの名前。
- [COLUMN_NAME](#)
制約があるカラムの名前。
制約が外部キーの場合、これは外部キーのカラムで、外部キーが参照するカラムではありません。
- [ORDINAL_POSITION](#)
制約内のカラムの位置。テーブル内のカラムの位置ではありません。カラムの位置には 1 から始まる番号が付けられています。
- [POSITION_IN_UNIQUE_CONSTRAINT](#)
一意制約および主キー制約用の `NULL`。外部キー制約の場合、このカラムは参照されるテーブルのキー内の順序位置です。
- [REFERENCED_TABLE_SCHEMA](#)
制約によって参照されるスキーマの名前。
- [REFERENCED_TABLE_NAME](#)
制約によって参照されるテーブルの名前。
- [REFERENCED_COLUMN_NAME](#)
制約によって参照されるカラムの名前。

次の定義を持つ `t1` および `t3` という 2 つのテーブルがあるとします。

```
CREATE TABLE t1
(
  s1 INT,
  s2 INT,
  s3 INT,
  PRIMARY KEY(s3)
```

```

) ENGINE=InnoDB;

CREATE TABLE t3
(
  s1 INT,
  s2 INT,
  s3 INT,
  KEY(s1),
  CONSTRAINT CO FOREIGN KEY (s2) REFERENCES t1(s3)
) ENGINE=InnoDB;

```

これらの 2 つのテーブルに対し、[KEY_COLUMN_USAGE](#) テーブルには次の 2 つの行があります。

- [CONSTRAINT_NAME](#) = 'PRIMARY'、[TABLE_NAME](#) = 't1'、[COLUMN_NAME](#) = 's3'、[ORDINAL_POSITION](#) = 1、[POSITION_IN_UNIQUE_CONSTRAINT](#) = NULL を含む 1 つの行。

[NDB](#) の場合: この値は常に NULL です。

- [CONSTRAINT_NAME](#) = 'CO'、[TABLE_NAME](#) = 't3'、[COLUMN_NAME](#) = 's2'、[ORDINAL_POSITION](#) = 1、[POSITION_IN_UNIQUE_CONSTRAINT](#) = 1 を含む 1 つの行。

26.17 INFORMATION_SCHEMA ndb_transid_mysql_connection_map テーブル

[ndb_transid_mysql_connection_map](#) テーブルは、NDB Cluster に API ノードとして接続された [NDB](#) トランザクション、[NDB](#) トランザクションコーディネータ、および MySQL Servers 間のマッピングを提供します。この情報は、[ndbinfo](#) NDB Cluster 情報データベースの [server_operations](#) および [server_transactions](#) テーブルを移入するときに使用されます。

INFORMATION_SCHEMA 名	SHOW 名	備考
mysql_connection_id		MySQL Server 接続 ID
node_id		トランザクションコーディネータノード ID
ndb_transid		NDB トランザクション ID

[mysql_connection_id](#) は、[SHOW PROCESSLIST](#) の出力に示された接続またはセッション ID と同じです。

このテーブルに関連付けられた [SHOW](#) ステートメントはありません。

これは [NDB Cluster](#) に固有の非標準テーブルです。これは、[INFORMATION_SCHEMA](#) プラグインとして実装されます。[SHOW PLUGINS](#) の出力を確認して、このプラグインがサポートされていることを検証できます。[ndb_transid_mysql_connection_map](#) サポートが有効である場合、次に (強調テキストを使用して) 示すように、このステートメントの出力には、この名前を持ち、タイプが [INFORMATION_SCHEMA](#) で、ステータスが [ACTIVE](#) であるプラグインが含まれます。

```

mysql> SHOW PLUGINS;
+-----+-----+-----+-----+-----+
| Name          | Status | Type          | Library | License |
+-----+-----+-----+-----+-----+
| binlog        | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| mysql_native_password | ACTIVE | AUTHENTICATION | NULL    | GPL     |
| CSV           | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| MEMORY        | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| MRG_MYISAM    | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| MyISAM        | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| PERFORMANCE_SCHEMA | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| BLACKHOLE     | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| ARCHIVE       | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| ndbcluster    | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| ndbinfo       | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| ndb_transid_mysql_connection_map | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| InnoDB        | ACTIVE | STORAGE ENGINE | NULL    | GPL     |
| INNODB_TRX    | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_LOCKS  | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_LOCK_WAITS | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |

```



```

| INNODB_CMP                | ACTIVE | INFORMATION SCHEMA | NULL | GPL |
| INNODB_CMP_RESET         | ACTIVE | INFORMATION SCHEMA | NULL | GPL |
| INNODB_CMPMEM            | ACTIVE | INFORMATION SCHEMA | NULL | GPL |
| INNODB_CMPMEM_RESET     | ACTIVE | INFORMATION SCHEMA | NULL | GPL |
| partition                 | ACTIVE | STORAGE ENGINE    | NULL | GPL |
+-----+-----+-----+-----+
22 rows in set (0.00 sec)

```

プラグインはデフォルトで有効になっています。--ndb-transid-mysql-connection-map オプションを付けてサーバーを起動すると、これを無効(または、プラグインが開始していないがぎりサーバーが実行できないように強制する)にできます。プラグインが無効の場合、SHOW PLUGINS でステータスが DISABLED と表示されます。実行時にプラグインは有効または無効にできません。

このテーブルとカラムの名前は小文字を使用して表示されますが、SQL ステートメントで参照するときには大文字も小文字も使用できます。

このテーブルを作成するには、MySQL Server が NDB Cluster 配布で提供されるバイナリか、NDB ストレージエンジンのサポートが有効になっている NDB Cluster ソースから構築されたバイナリである必要があります。標準の MySQL 8.0 Server では使用できません。

26.18 INFORMATION_SCHEMA KEYWORDS テーブル

KEYWORDS テーブルには、MySQL で考慮されるキーワードがリストされ、各キーワードについて予約されているかどうかを示されます。予約済キーワードは、識別子として使用される場合の特別な引用符など、一部のコンテキストで特別な処理が必要になることがあります(セクション9.3「キーワードと予約語」を参照)。このテーブルは、アプリケーションが MySQL キーワード情報のランタイムソースを提供します。

MySQL 8.0.13 より前は、デフォルトのデータベースが選択されていない KEYWORDS テーブルから選択すると、エラーが発生しました。(Bug #90160、Bug #27729859)

KEYWORDS テーブルには、次のカラムがあります:

- WORD

キーワード。

- RESERVED

キーワードが予約されている (1) か予約されていない (0) かを示す整数。

次のクエリーでは、すべてのキーワード、すべての予約済キーワードおよびすべての非予約済キーワードがそれぞれリストされます:

```

SELECT * FROM INFORMATION_SCHEMA.KEYWORDS;
SELECT WORD FROM INFORMATION_SCHEMA.KEYWORDS WHERE RESERVED = 1;
SELECT WORD FROM INFORMATION_SCHEMA.KEYWORDS WHERE RESERVED = 0;

```

後者の2つのクエリーは、次と同等です:

```

SELECT WORD FROM INFORMATION_SCHEMA.KEYWORDS WHERE RESERVED;
SELECT WORD FROM INFORMATION_SCHEMA.KEYWORDS WHERE NOT RESERVED;

```

ソースから MySQL をビルドする場合、ビルドプロセスによって、キーワードの配列とその予約済ステータスを含む keyword_list.h ヘッダーファイルが生成されます。このファイルは、ビルドディレクトリの下に sql ディレクトリにあります。このファイルは、キーワードリストの静的ソースを必要とするアプリケーションに役立つ場合があります。

26.19 INFORMATION_SCHEMA OPTIMIZER_TRACE テーブル

OPTIMIZER_TRACE テーブルは、トレースされたステートメントのオプティマイザトレース機能によって生成される情報を提供します。追跡を有効にするには、optimizer_trace システム変数を使用します。詳細については、「MySQL Internals: Tracing the Optimizer」を参照してください。

OPTIMIZER_TRACE テーブルには、次のカラムがあります:

- QUERY

トレースされたステートメントのテキスト。

- **TRACE**

JSON 形式のトレース。

- **MISSING_BYTES_BEYOND_MAX_MEM_SIZE**

記憶されている各トレースは、最適化の進行に応じて拡張され、データが追加される文字列です。**optimizer_trace_max_mem_size** 変数は、現在記憶されているすべてのトレースで使用されるメモリーの合計量に制限を設定します。この制限に達すると、現在のトレースは拡張されず (したがって不完全)、**MISSING_BYTES_BEYOND_MAX_MEM_SIZE** カラムにトレースから欠落しているバイト数が表示されます。

- **INSUFFICIENT_PRIVILEGES**

トレースされたクエリーが、**DEFINER** の値を持つ **SQL SECURITY** を持つビューまたはストアドルーチンを使用している場合、定義者以外のユーザーがクエリーのトレースを表示できない可能性があります。その場合、トレースは空として表示され、**INSUFFICIENT_PRIVILEGES** の値は 1 です。それ以外の場合、値は 0 です。

26.20 INFORMATION_SCHEMA PARAMETERS テーブル

PARAMETERS テーブルは、ストアドルーチン (ストアドプロシージャとストアドファンクション) のパラメータ、およびストアドファンクションの戻り値に関する情報を提供します。**PARAMETERS** テーブルには、組込み SQL 関数またはユーザー定義関数 (UDF) は含まれていません。

PARAMETERS テーブルには、次のカラムがあります:

- **SPECIFIC_CATALOG**

パラメータを含むルーチンが属するカタログの名前。この値は常に **def** です。

- **SPECIFIC_SCHEMA**

パラメータを含むルーチンが属するスキーマ (データベース) の名前。

- **SPECIFIC_NAME**

パラメータを含むルーチンの名前。

- **ORDINAL_POSITION**

ストアドプロシージャとストアドファンクションの連続したパラメータについては、**ORDINAL_POSITION** 値は 1、2、3 などです。ストアドファンクションの場合は、(**RETURNS** 句で説明されているように) 関数の戻り値に適用される行もあります。戻り値は真のパラメータではないので、これについて説明している行には次の一意の特性があります。

- **ORDINAL_POSITION** 値は 0 です。

- 戻り値には名前がなく、モードが適用されないため、**PARAMETER_NAME** 値と **PARAMETER_MODE** 値は **NULL** です。

- **PARAMETER_MODE**

パラメータのモード。この値は、**IN**、**OUT** または **INOUT** のいずれかです。ストアドファンクションの戻り値の場合、この値は **NULL** です。

- **PARAMETER_NAME**

パラメータの名前。ストアドファンクションの戻り値の場合、この値は **NULL** です。

- **DATA_TYPE**

パラメータのデータ型。

`DATA_TYPE` 値はタイプ名のみで、他の情報はありません。 `DTD_IDENTIFIER` 値には、型名と、精度や長さなどのその他の情報が含まれます。

- `CHARACTER_MAXIMUM_LENGTH`
文字列パラメータの場合、最大長 (文字数)。
- `CHARACTER_OCTET_LENGTH`
文字列パラメータの場合、最大長 (バイト単位)。
- `NUMERIC_PRECISION`
数値パラメータの場合は、数値精度。
- `NUMERIC_SCALE`
数値パラメータの場合は、数値スケール。
- `DATETIME_PRECISION`
時間パラメータの場合、小数秒精度。
- `CHARACTER_SET_NAME`
文字列パラメータの場合は、文字セット名。
- `COLLATION_NAME`
文字列パラメータの場合、照合順序名。
- `DTD_IDENTIFIER`
パラメータのデータ型。

`DATA_TYPE` 値はタイプ名のみで、他の情報はありません。 `DTD_IDENTIFIER` 値には、型名と、精度や長さなどのその他の情報が含まれます。

- `ROUTINE_TYPE`
ストアドプロシージャ用の `PROCEDURE`、ストアドファンクション用の `FUNCTION`。

26.21 INFORMATION_SCHEMA PARTITIONS テーブル

`PARTITIONS` テーブルは、テーブルパーティションに関する情報を提供します。このテーブルの各行は、パーティションテーブルの個々のパーティションまたはサブパーティションに対応します。テーブルのパーティション化の詳細は、[第24章「パーティション化」](#)を参照してください。

`PARTITIONS` テーブルには、次のカラムがあります:

- `TABLE_CATALOG`
テーブルが属するカタログの名前。この値は常に `def` です。
- `TABLE_SCHEMA`
テーブルが属するスキーマ (データベース) の名前。
- `TABLE_NAME`
パーティションを含むテーブルの名前。
- `PARTITION_NAME`
パーティションの名前。

- **SUBPARTITION_NAME**

PARTITIONS テーブルの行がサブパーティションを表す場合はサブパーティションの名前、それ以外の場合は **NULL**。

NDB の場合: この値は常に **NULL** です。

- **PARTITION_ORDINAL_POSITION**

すべてのパーティションは、定義されている順序でインデックス付けされ、**1** は最初のパーティションに割り当てられた番号です。インデックス設定は、パーティションが追加、削除、再編成されると変わることがあります。このカラムに表示された番号は、インデックス設定の変更を考慮した現在の順序を表します。

- **SUBPARTITION_ORDINAL_POSITION**

特定のパーティション内のサブパーティションも、パーティションがテーブル内でインデックス付けされるのと同じ方法でインデックス付けおよび再インデックス付けされます。

- **PARTITION_METHOD**

RANGE, **LIST**, **HASH**, **LINEAR HASH**, **KEY** または **LINEAR KEY** のいずれかの値、つまり [セクション24.2「パーティショニングタイプ」](#) で説明されている使用可能なパーティション化タイプ。

- **SUBPARTITION_METHOD**

値 **HASH**, **LINEAR HASH**, **KEY** または **LINEAR KEY**。つまり、[セクション24.2.6「サブパーティショニング」](#) で説明されている使用可能なサブパーティション化タイプのいずれかです。

- **PARTITION_EXPRESSION**

テーブルの現在のパーティション化スキームを作成した **CREATE TABLE** ステートメントまたは **ALTER TABLE** ステートメントで使用されるパーティション化関数の式。

たとえば、次のステートメントを使用して **test** データベースに作成されたパーティション化されたテーブルを考えてみます。

```
CREATE TABLE tp (  
  c1 INT,  
  c2 INT,  
  c3 VARCHAR(25)  
)  
PARTITION BY HASH(c1 + c2)  
PARTITIONS 4;
```

このテーブルのパーティションの **PARTITIONS** テーブルの行の **PARTITION_EXPRESSION** カラムには、次に示すように **c1 + c2** が表示されます:

```
mysql> SELECT DISTINCT PARTITION_EXPRESSION  
  FROM INFORMATION_SCHEMA.PARTITIONS  
  WHERE TABLE_NAME='tp' AND TABLE_SCHEMA='test';  
+-----+  
| PARTITION_EXPRESSION |  
+-----+  
| c1 + c2              |  
+-----+
```

- **SUBPARTITION_EXPRESSION**

これは、**PARTITION_EXPRESSION** がテーブルパーティション化の定義に使用するパーティション化式の場合と同様に、テーブルのサブパーティション化を定義するサブパーティション化式でも機能します。

テーブルにサブパーティションがない場合、このカラムは **NULL** です。

- **PARTITION_DESCRIPTION**

このカラムは **RANGE** および **LIST** パーティションに使用されます。**RANGE** パーティションの場合、パーティションの **VALUES LESS THAN** 句で設定された値が含まれます。これには整数か **MAXVALUE** のどちらかを指定で

きます。LISTパーティションの場合、このカラムには、カンマ区切りの整数値のリストであるパーティションのVALUES IN句で定義された値が含まれます。

PARTITION_METHODがRANGEまたはLIST以外であるパーティションの場合、このカラムは常にNULLになります。

- **TABLE_ROWS**

パーティション内のテーブルの行数。

パーティション化されたInnoDBテーブルの場合、TABLE_ROWSカラムで指定された行数は、SQL最適化で使われる推定値に過ぎず、必ずしも正確であるとはかぎりません。

NDBテーブルの場合は、ndb_descユーティリティを使用してこの情報を取得することもできます。

- **AVG_ROW_LENGTH**

このパーティションまたはサブパーティションに格納されている行の平均長(バイト)。これはTABLE_ROWSで分割されたDATA_LENGTHと同じです。

NDBテーブルの場合は、ndb_descユーティリティを使用してこの情報を取得することもできます。

- **DATA_LENGTH**

このパーティションまたはサブパーティションに格納されているすべての行の合計長(バイト)。つまり、パーティションまたはサブパーティションに格納されている合計バイト数です。

NDBテーブルの場合は、ndb_descユーティリティを使用してこの情報を取得することもできます。

- **MAX_DATA_LENGTH**

このパーティションまたはサブパーティションに格納できる最大バイト数。

NDBテーブルの場合は、ndb_descユーティリティを使用してこの情報を取得することもできます。

- **INDEX_LENGTH**

このパーティションまたはサブパーティションのインデックスファイルの長さ(バイト単位)。

NDBテーブルのパーティションの場合、テーブルで暗黙的または明示的なパーティション化を使用するかどうかにかかわらず、INDEX_LENGTHカラムの値は常に0です。ただし、ndb_descユーティリティを使用して同等の情報を取得できます。

- **DATA_FREE**

パーティションまたはサブパーティションに割り当てられているが使用されていないバイト数。

NDBテーブルの場合は、ndb_descユーティリティを使用してこの情報を取得することもできます。

- **CREATE_TIME**

パーティションまたはサブパーティションが作成された時刻。

- **UPDATE_TIME**

パーティションまたはサブパーティションが最後に変更された時刻。

- **CHECK_TIME**

このパーティションまたはサブパーティションが属するテーブルが最後にチェックされた時刻。

パーティション化されたInnoDBテーブルの場合、値は常にNULLです。

- **CHECKSUM**

チェックサム値(存在する場合)。それ以外の場合はNULL。

- **PARTITION_COMMENT**

コメントのテキスト (パーティションにコメントがある場合)。そうでない場合、この値は空です。

パーティションコメントの最大長は 1024 文字として定義され、**PARTITION_COMMENT** カラムの表示幅も 1024 文字で、この制限に一致します。

- **NODEGROUP**

これは、パーティションが属するノードグループです。これは「NDB Cluster」テーブルにのみ関連します。それ以外の場合、値は常に 0 です。

- **TABLESPACE_NAME**

パーティションが属するテーブルスペースの名前。テーブルで **NDB** ストレージエンジンが使用されていないがぎり、値は常に **DEFAULT** です (このセクションの最後にある「ノート」を参照)。

メモ

- **PARTITIONS** は非標準の **INFORMATION_SCHEMA** テーブルです。
- **NDB** 以外のストレージエンジンを使用し、パーティション化されていないテーブルには、**PARTITIONS** テーブル内に 1 つの行があります。ただし、**PARTITION_NAME**, **SUBPARTITION_NAME**, **PARTITION_ORDINAL_POSITION**, **SUBPARTITION_ORDINAL_POSITION**, **PARTITION_METHOD**, **SUBPARTITION_METHOD**, **PARTITION_EXPRESSION**, **SUBPARTITION_EXPRESSION** カラムおよび **PARTITION_DESCRIPTION** カラムの値はすべて **NULL** です。また、この場合、**PARTITION_COMMENT** カラムは空白です。
- 明示的にパーティション化されていない **NDB** テーブルには、**NDB** クラスタ内の各データノードの **PARTITIONS** テーブル内に 1 つの行があります。そのような行ごとに、次の手順を実行します:
 - **SUBPARTITION_NAME**, **SUBPARTITION_ORDINAL_POSITION**, **SUBPARTITION_METHOD**, **PARTITION_EXPRESSION**, **SUBPARTITION_EXPRESSION**, **CREATE_TIME**, **UPDATE_TIME**, **CHECK_TIME**, **CHECKSUM** および **TABLESPACE_NAME** のカラムはすべて **NULL** です。
 - **PARTITION_METHOD** は常に **AUTO** です。
 - **NODEGROUP** カラムは **default** です。
 - **PARTITION_EXPRESSION** カラムと **PARTITION_COMMENT** カラムが空です。

26.22 INFORMATION_SCHEMA PLUGINS テーブル

PLUGINS テーブルは、サーバーのプラグインに関する情報を提供します。

PLUGINS テーブルには、次のカラムがあります:

- **PLUGIN_NAME**
INSTALL PLUGIN や **UNINSTALL PLUGIN** などのステートメントでプラグインを参照するために使用される名前。
- **PLUGIN_VERSION**
プラグイン一般型ディスクリプタのバージョン。
- **PLUGIN_STATUS**
プラグインステータス (**ACTIVE**, **INACTIVE**, **DISABLED**, **DELETING** のいずれか、または **DELETED**)。
- **PLUGIN_TYPE**
プラグインのタイプ (**STORAGE ENGINE**, **INFORMATION_SCHEMA**, **AUTHENTICATION** など)。
- **PLUGIN_TYPE_VERSION**

プラグインタイプ固有のディスクリプタのバージョン。

- **PLUGIN_LIBRARY**

プラグイン共有ライブラリファイルの名前。これは、**INSTALL PLUGIN** や **UNINSTALL PLUGIN** などのステートメントでプラグインファイルを参照するために使用される名前です。このファイルは、`plugin_dir` システム変数によって指名されたディレクトリに置かれます。ライブラリ名が **NULL** である場合、プラグインはコンパイルされませんが、**UNINSTALL PLUGIN** でアンインストールできません。

- **PLUGIN_LIBRARY_VERSION**

プラグイン API インタフェースのバージョン。

- **PLUGIN_AUTHOR**

プラグインの作成者。

- **PLUGIN_DESCRIPTION**

プラグインの簡単な説明。

- **PLUGIN_LICENSE**

プラグインのライセンス方法 (GPL など)。

- **LOAD_OPTION**

プラグインのロード方法。値は、**OFF**、**ON**、**FORCE**、または **FORCE_PLUS_PERMANENT** です。 [セクション 5.6.1「プラグインのインストールおよびアンインストール」](#) を参照してください。

メモ

- **PLUGINS** は非標準の **INFORMATION_SCHEMA** テーブルです。

- **INSTALL PLUGIN** でインストールされたプラグインの場合、**PLUGIN_NAME** および **PLUGIN_LIBRARY** 値は、`mysql.plugin` テーブルにも登録されます。

- **PLUGINS** テーブル内の情報のベースを形成するプラグインデータ構造の詳細は、[The MySQL Plugin API](#) を参照してください。

プラグイン情報は、**SHOW PLUGINS** ステートメントからも入手できます。 [セクション 13.7.7.25「SHOW PLUGINS ステートメント」](#) を参照してください。次のステートメントは同等です。

```
SELECT
  PLUGIN_NAME, PLUGIN_STATUS, PLUGIN_TYPE,
  PLUGIN_LIBRARY, PLUGIN_LICENSE
FROM INFORMATION_SCHEMA.PLUGINS;

SHOW PLUGINS;
```

26.23 INFORMATION_SCHEMA PROCESSLIST テーブル

MySQL プロセスリストには、サーバー内で実行されているスレッドのセットによって現在実行されている操作が示されます。 **PROCESSLIST** テーブルは、プロセス情報のソースです。このテーブルと他のソースの比較は、 [プロセス情報のソース](#) を参照してください。

PROCESSLIST テーブルには、次のカラムがあります:

- **ID**

接続識別子。これは、**SHOW PROCESSLIST** ステートメントの `Id` カラムに表示される値と同じで、パフォーマンススキーマ `threads` テーブルの `PROCESSLIST_ID` カラムに表示され、スレッド内の `CONNECTION_ID()` 関数によって返されます。

- **USER**

このステートメントを発行した MySQL ユーザー。 `system user` の値は、遅延行ハンドラスレッド、レプリカホストで使用される I/O または SQL スレッドなど、タスクを内部的に処理するためにサーバーによって起動される非クライアントスレッドを指します。 `system user` の場合、 `Host` カラムにホストが指定されていません。 `unauthenticated user` は、クライアント接続に関連付けられたが、クライアントユーザーの認証がまだ行われていないスレッドを参照します。 `event_scheduler` は、スケジュールされたイベントをモニターするスレッドを指します (セクション25.4「イベントスケジューラの使用」を参照)。

注記

`system user` の `USER` 値は、 `SYSTEM_USER` 権限とは異なります。前者は内部スレッドを指定します。後者は、システムユーザーと通常のユーザーアカウントカテゴリを区別します (セクション6.2.11「アカウントカテゴリ」を参照)。

- **HOST**

ステートメントを発行するクライアントのホスト名 (ホストがない `system user` を除く)。TCP/IP 接続のホスト名は、 `host_name:client_port` 形式でレポートされるため、どのクライアントが何を実行しているかを簡単に判別できます。

- **DB**

スレッドのデフォルトデータベース。選択されていない場合は `NULL`。

- **COMMAND**

スレッドがクライアントのかわりに実行しているコマンドのタイプ。セッションがアイドル状態の場合は `Sleep`。スレッドコマンドの説明については、セクション8.14「サーバスレッド (プロセス) 情報の確認」を参照してください。このカラムの値は、クライアント/サーバープロトコルの `COM_xxx` コマンドと `Com_xxx` ステータス変数に対応します。セクション5.1.10「サーバスステータス変数」を参照してください。

- **TIME**

スレッドが現在の状態になってからの秒数。レプリカ SQL スレッドの場合、この値は、最後にレプリケートされたイベントのタイムスタンプとレプリカホストのリアルタイムの間の秒数です。セクション17.2.3「レプリケーションスレッド」を参照してください。

- **STATE**

スレッドが行なっていることを示すアクション、イベント、または状態。 `STATE` の値の詳細は、セクション8.14「サーバスレッド (プロセス) 情報の確認」を参照してください。

ほとんどの状態がきわめてすばやい操作に対応します。スレッドの状態が何秒間も特定の状態にとどまっている場合は、調査が必要な問題が発生している可能性があります。

- **INFO**

スレッドが実行しているステートメント。ステートメントを実行していない場合は `NULL`。このステートメントは、サーバーに送信されるステートメント、またはこのステートメントがほかのステートメントを実行する場合は、もっとも内側のステートメントである可能性があります。たとえば、 `CALL` ステートメントが `SELECT` ステートメントを実行しているストアードプロシージャを実行する場合、 `INFO` 値には `SELECT` ステートメントが表示されます。

メモ

- `PROCESSLIST` は非標準の `INFORMATION_SCHEMA` テーブルです。

- `SHOW PROCESSLIST` ステートメントからの出力と同様に、 `PROCESSLIST` テーブルには、 `PROCESS` 権限がある場合は他のユーザーに属するスレッドも含め、すべてのスレッドに関する情報が表示されます。それ以外の場合 (`PROCESS` 権限なし)、非匿名ユーザーは自分のスレッドに関する情報にはアクセスできますが、他のユーザーのスレッドにはアクセスできず、匿名ユーザーはスレッド情報にアクセスできません。

- SQL ステートメントが `PROCESSLIST` テーブルを参照する場合、MySQL はステートメントの実行が開始されるとテーブル全体を一度移入するため、ステートメントの実行中に読取り一貫性があります。複数ステートメントトランザクションの読取り一貫性はありません。

次のステートメントは同等です。

```
SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST
SHOW FULL PROCESSLIST
```

26.24 INFORMATION_SCHEMA PROFILING テーブル

`PROFILING` テーブルは、ステートメントプロファイリング情報を提供します。その内容は、`SHOW PROFILE` および `SHOW PROFILES` ステートメントによって生成される情報に対応しています ([セクション13.7.7.30「SHOW PROFILE ステートメント」](#) を参照)。`profiling` セッション変数は 1 に設定されないかぎり、テーブルは空です。

注記

このテーブルは非推奨です。将来の MySQL リリースで削除される予定です。かわりに `Performance Schema` を使用します。[セクション27.19.1「パフォーマンススキーマを使用したクエリープロファイリング」](#) を参照してください。

`PROFILING` テーブルには、次のカラムがあります:

- `QUERY_ID`
数値のステートメント識別子。
- `SEQ`
同じ `QUERY_ID` 値を持つ行の表示順序を示す順序番号。
- `STATE`
行の測定が適用されるプロファイリング状態。
- `DURATION`
ステートメントの実行が指定された状態のままだった時間 (秒)。
- `CPU_USER`, `CPU_SYSTEM`
ユーザーおよびシステムの CPU 使用率 (秒)。
- `CONTEXT_VOLUNTARY`, `CONTEXT_INVOLUNTARY`
自主的および標準的なコンテキストスイッチが発生した数。
- `BLOCK_OPS_IN`, `BLOCK_OPS_OUT`
ブロックの入出力操作の数。
- `MESSAGES_SENT`, `MESSAGES_RECEIVED`
送受信された通信メッセージの数。
- `PAGE_FAULTS_MAJOR`, `PAGE_FAULTS_MINOR`
メジャーおよびマイナーページフォルトの数。
- `SWAPS`
スワップがいくつ発生したか。
- `SOURCE_FUNCTION`, `SOURCE_FILE` および `SOURCE_LINE`

プロファイリングされた状態がソースコードのどこで実行されるかを示す情報。

メモ

- `PROFILING` は非標準の `INFORMATION_SCHEMA` テーブルです。

プロファイリング情報は、`SHOW PROFILE` および `SHOW PROFILES` ステートメントからも入手できます。セクション13.7.7.30「`SHOW PROFILE` ステートメント」を参照してください。たとえば、次のクエリーは同等です:

```
SHOW PROFILE FOR QUERY 2;  
  
SELECT STATE, FORMAT(DURATION, 6) AS DURATION  
FROM INFORMATION_SCHEMA.PROFILING  
WHERE QUERY_ID = 2 ORDER BY SEQ;
```

26.25 INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS テーブル

`REFERENTIAL_CONSTRAINTS` テーブルは、外部キーに関する情報を提供します。

`REFERENTIAL_CONSTRAINTS` テーブルには、次のカラムがあります:

- `CONSTRAINT_CATALOG`
制約が属するカタログの名前。この値は常に `def` です。
- `CONSTRAINT_SCHEMA`
制約が属するスキーマ (データベース) の名前。
- `CONSTRAINT_NAME`
制約の名前。
- `UNIQUE_CONSTRAINT_CATALOG`
制約が参照する一意制約を含むカタログの名前。この値は常に `def` です。
- `UNIQUE_CONSTRAINT_SCHEMA`
制約が参照する一意制約を含むスキーマの名前。
- `UNIQUE_CONSTRAINT_NAME`
制約が参照する一意制約の名前。
- `MATCH_OPTION`
制約 `MATCH` 属性の値。現時点で有効な値は `NONE` のみです。
- `UPDATE_RULE`
制約 `ON UPDATE` 属性の値。使用可能な値は `CASCADE`, `SET NULL`, `SET DEFAULT`, `RESTRICT`, `NO ACTION` です。
- `DELETE_RULE`
制約 `ON DELETE` 属性の値。使用可能な値は `CASCADE`, `SET NULL`, `SET DEFAULT`, `RESTRICT`, `NO ACTION` です。
- `TABLE_NAME`
テーブルの名前。この値は、`TABLE_CONSTRAINTS` テーブルの値と同じです。
- `REFERENCED_TABLE_NAME`

制約によって参照されるテーブルの名前。

26.26 INFORMATION_SCHEMA RESOURCE_GROUPS テーブル

`RESOURCE_GROUPS` テーブルからは、リソースグループに関する情報にアクセスできます。リソースグループ機能の一般的な説明については、[セクション5.1.16「リソースグループ」](#)を参照してください。

一部の権限を持つカラムの情報のみを表示できます。

`RESOURCE_GROUPS` テーブルには、次のカラムがあります:

- `RESOURCE_GROUP_NAME`

リソースグループの名前。

- `RESOURCE_GROUP_TYPE`

リソースグループタイプ (`SYSTEM` または `USER`)。

- `RESOURCE_GROUP_ENABLED`

リソースグループが有効 (1) か無効 (0) か

- `VCPU_IDS`

CPU アフィニティー。つまり、リソースグループが使用できる仮想 CPU のセット。値は、カンマ区切りの CPU 番号または範囲のリストです。

- `THREAD_PRIORITY`

リソースグループに割り当てられたスレッドの優先順位。プライオリティの範囲は -20 (最高プライオリティ) から 19 (最低プライオリティ) です。システムリソースグループの優先順位の範囲は -20 から 0 です。ユーザーリソースグループの優先度の範囲は 0 から 19 です。

26.27 INFORMATION_SCHEMA ROLE_COLUMN_GRANTS テーブル

`ROLE_COLUMN_GRANTS` テーブル (MySQL 8.0.19 で使用可能) には、現在有効なロールで使用可能または付与されているロールのカラム権限に関する情報が表示されます。

`ROLE_COLUMN_GRANTS` テーブルには、次のカラムがあります:

- `GRANTOR`

ロールを付与したアカウントのユーザー名部分。

- `GRANTOR_HOST`

ロールを付与したアカウントのホスト名部分。

- `GRANTEE`

ロールが付与されるアカウントのユーザー名部分。

- `GRANTEE_HOST`

ロールが付与されるアカウントのホスト名部分。

- `TABLE_CATALOG`

ロールが適用されるカタログの名前。この値は常に `def` です。

- `TABLE_SCHEMA`

ロールが適用されるスキーマ (データベース) の名前。

- **TABLE_NAME**
ロールが適用されるテーブルの名前。
- **COLUMN_NAME**
ロールが適用されるカラムの名前。
- **PRIVILEGE_TYPE**
付与された権限。値は、カラムレベルで付与できる任意の権限です。[セクション13.7.1.6「GRANT ステートメント」](#)を参照してください。各行には単一の権限がリストされるため、権限受領者が保持するカラム権限ごとに1つの行があります。
- **IS_GRANTABLE**
ロールが他のアカウントに付与可能かどうかに応じて、**YES** または **NO**。

26.28 INFORMATION_SCHEMA_ROLE_ROUTINE_GRANTS テーブル

ROLE_ROUTINE_GRANTS テーブル (MySQL 8.0.19 で使用可能) には、現在有効なロールで使用可能または付与されているロールのルーチン権限に関する情報が表示されます。

ROLE_ROUTINE_GRANTS テーブルには、次のカラムがあります:

- **GRANTOR**
ロールを付与したアカウントのユーザー名部分。
- **GRANTOR_HOST**
ロールを付与したアカウントのホスト名部分。
- **GRANTEE**
ロールが付与されるアカウントのユーザー名部分。
- **GRANTEE_HOST**
ロールが付与されるアカウントのホスト名部分。
- **SPECIFIC_CATALOG**
ルーチンが属するカタログの名前。この値は常に **def** です。
- **SPECIFIC_SCHEMA**
ルーチンが属するスキーマ (データベース) の名前。
- **SPECIFIC_NAME**
ルーチンの名前。
- **ROUTINE_CATALOG**
ルーチンが属するカタログの名前。この値は常に **def** です。
- **ROUTINE_SCHEMA**
ルーチンが属するスキーマ (データベース) の名前。
- **ROUTINE_NAME**
ルーチンの名前。
- **PRIVILEGE_TYPE**

付与された権限。この値は、ルーチンレベルで付与できる任意の権限です。[セクション13.7.1.6「GRANT ステートメント」](#)を参照してください。各行には単一の権限がリストされるため、権限受領者が保持するカラム権限ごとに1つの行があります。

- [IS_GRANTABLE](#)

ロールが他のアカウントに付与可能かどうかに応じて、[YES](#) または [NO](#)。

26.29 INFORMATION_SCHEMA_ROLE_TABLE_GRANTS テーブル

[ROLE_TABLE_GRANTS](#) テーブル (MySQL 8.0.19 で使用可能) には、現在有効なロールで使用可能または付与されているロールのテーブル権限に関する情報が表示されます。

[ROLE_TABLE_GRANTS](#) テーブルには、次のカラムがあります:

- [GRANTOR](#)

ロールを付与したアカウントのユーザー名部分。

- [GRANTOR_HOST](#)

ロールを付与したアカウントのホスト名部分。

- [GRANTEE](#)

ロールが付与されるアカウントのユーザー名部分。

- [GRANTEE_HOST](#)

ロールが付与されるアカウントのホスト名部分。

- [TABLE_CATALOG](#)

ロールが適用されるカタログの名前。この値は常に [def](#) です。

- [TABLE_SCHEMA](#)

ロールが適用されるスキーマ (データベース) の名前。

- [TABLE_NAME](#)

ロールが適用されるテーブルの名前。

- [PRIVILEGE_TYPE](#)

付与された権限。この値は、テーブルレベルで付与できる任意の権限です。[セクション13.7.1.6「GRANT ステートメント」](#)を参照してください。各行には単一の権限がリストされるため、権限受領者が保持するカラム権限ごとに1つの行があります。

- [IS_GRANTABLE](#)

ロールが他のアカウントに付与可能かどうかに応じて、[YES](#) または [NO](#)。

26.30 INFORMATION_SCHEMA_ROUTINES テーブル

[ROUTINES](#) テーブルは、ストアードルーチン (ストアードプロシージャとストアードファンクション) に関する情報を提供します。[ROUTINES](#) テーブルには、組込み SQL 関数またはユーザー定義関数 (UDF) は含まれていません。

[ROUTINES](#) テーブルには、次のカラムがあります:

- [SPECIFIC_NAME](#)

ルーチンの名前。

- **ROUTINE_CATALOG**
ルーチンが属するカタログの名前。この値は常に **def** です。
- **ROUTINE_SCHEMA**
ルーチンが属するスキーマ (データベース) の名前。
- **ROUTINE_NAME**
ルーチンの名前。
- **ROUTINE_TYPE**
ストアドプロシージャ用の **PROCEDURE**、ストアドファンクション用の **FUNCTION**。
- **DATA_TYPE**
ルーチンがストアドファンクションの場合、戻り値のデータ型。ルーチンがストアドプロシージャの場合、この値は空です。
DATA_TYPE 値はタイプ名のみで、他の情報はありません。 **DTD_IDENTIFIER** 値には、型名と、精度や長さなどのその他の情報が含まれます。
- **CHARACTER_MAXIMUM_LENGTH**
ストアドファンクション文字列の戻り値の場合、最大長 (文字数)。ルーチンがストアドプロシージャの場合、この値は **NULL** です。
- **CHARACTER_OCTET_LENGTH**
ストアドファンクション文字列の戻り値の最大長 (バイト単位)。ルーチンがストアドプロシージャの場合、この値は **NULL** です。
- **NUMERIC_PRECISION**
ストアドファンクションの数値の戻り値の場合、数値精度。ルーチンがストアドプロシージャの場合、この値は **NULL** です。
- **NUMERIC_SCALE**
ストアドファンクションの数値の戻り値の場合、数値スケール。ルーチンがストアドプロシージャの場合、この値は **NULL** です。
- **DATETIME_PRECISION**
ストアドファンクションの時間的戻り値の場合、小数秒精度。ルーチンがストアドプロシージャの場合、この値は **NULL** です。
- **CHARACTER_SET_NAME**
ストアドファンクションの文字列戻り値の場合、文字セット名。ルーチンがストアドプロシージャの場合、この値は **NULL** です。
- **COLLATION_NAME**
ストアドファンクションの文字列の戻り値の場合、照合名。ルーチンがストアドプロシージャの場合、この値は **NULL** です。
- **DTD_IDENTIFIER**
ルーチンがストアドファンクションの場合、戻り値のデータ型。ルーチンがストアドプロシージャの場合、この値は空です。
DATA_TYPE 値はタイプ名のみで、他の情報はありません。 **DTD_IDENTIFIER** 値には、型名と、精度や長さなどのその他の情報が含まれます。

- **ROUTINE_BODY**
ルーチン定義に使用される言語。この値は常に **SQL** です。
- **ROUTINE_DEFINITION**
ルーチンによって実行される SQL ステートメントのテキスト。
- **EXTERNAL_NAME**
この値は常に **NULL** です。
- **EXTERNAL_LANGUAGE**
ストアルーチンの言語。この値は、**mysql.routines** データディクショナリテーブルの **external_language** カラムから読み取られます。
- **PARAMETER_STYLE**
この値は常に **SQL** です。
- **IS_DETERMINISTIC**
ルーチンが **DETERMINISTIC** 特性で定義されているかどうかに応じて、**YES** または **NO**。
- **SQL_DATA_ACCESS**
ルーチンのデータアクセス特性。値は、**CONTAINS SQL**, **NO SQL**, **READS SQL DATA** または **MODIFIES SQL DATA** のいずれかです。
- **SQL_PATH**
この値は常に **NULL** です。
- **SECURITY_TYPE**
ルーチンの **SQL SECURITY** 特性。値は、**DEFINER** または **INVOKER** のいずれかです。
- **CREATED**
ルーチンが作成された日時。これは **TIMESTAMP** 値です。
- **LAST_ALTERED**
ルーチンが最後に変更された日時。これは **TIMESTAMP** 値です。ルーチンが作成されてから変更されていない場合、この値は **CREATED** 値と同じです。
- **SQL_MODE**
ルーチンが作成または変更され、そのルーチンが実行されたときに有効な SQL モード。指定可能な値については、[セクション5.1.11「サーバー SQL モード」](#)を参照してください。
- **ROUTINE_COMMENT**
コメントのテキスト (ルーチンにコメントがある場合)。そうでない場合、この値は空です。
- **DEFINER**
'user_name'@'host_name'形式の **DEFINER** 句で指定されたアカウント (多くの場合、ルーチンを作成したユーザー)。
- **CHARACTER_SET_CLIENT**
ルーチン作成時の **character_set_client** システム変数のセッション値。
- **COLLATION_CONNECTION**

ルーチン作成時の `collation_connection` システム変数のセッション値。

- `DATABASE_COLLATION`

ルーチンが関連付けられているデータベースの照合。

メモ

- ルーチンに関する情報を表示するには、ルーチン `DEFINER` として指定されたユーザー、`SHOW_ROUTINE` 権限、グローバルレベルでの `SELECT` 権限、またはルーチンを含むスコープで付与された `CREATE ROUTINE`、`ALTER ROUTINE` または `EXECUTE` 権限を持っている必要があります。 `CREATE ROUTINE`、`ALTER ROUTINE` または `EXECUTE` のみがある場合、`ROUTINE_DEFINITION` カラムは `NULL` です。
- ストアドファンクションの戻り値に関する情報は、`PARAMETERS` テーブルにもあります。ストアドファンクションの戻り値の行は、`ORDINAL_POSITION` 値が 0 の行として識別できます。

26.31 INFORMATION_SCHEMA SCHEMATA テーブル

スキーマはデータベースなので、`SCHEMATA` テーブルはデータベースに関する情報を提供します。

`SCHEMATA` テーブルには、次のカラムがあります：

- `CATALOG_NAME`

スキーマが属するカタログの名前。この値は常に `def` です。

- `SCHEMA_NAME`

スキーマの名前。

- `DEFAULT_CHARACTER_SET_NAME`

スキーマのデフォルトの文字セット。

- `DEFAULT_COLLATION_NAME`

スキーマのデフォルトの照合。

- `SQL_PATH`

この値は常に `NULL` です。

- `DEFAULT_ENCRYPTION`

スキーマのデフォルトの暗号化。このカラムは、MySQL 8.0.16 で追加されました。

スキーマ名は、`SHOW DATABASES` ステートメントからも使用できます。 [セクション13.7.7.14「SHOW DATABASES ステートメント」](#) を参照してください。次のステートメントは同等です。

```
SELECT SCHEMA_NAME AS `Database`  
FROM INFORMATION_SCHEMA.SCHEMATA  
[WHERE SCHEMA_NAME LIKE 'wild']
```

```
SHOW DATABASES  
[LIKE 'wild']
```

グローバルな `SHOW DATABASES` 権限を持っていないかぎり、何らかの種類の権限を持っているデータベースしか表示できません。

注意

静的グローバル権限はすべてのデータベースに対する権限とみなされるため、静的グローバル権限を使用すると、ユーザーは、部分的な取消しによってデータベースレベルで制限されているデータベースを除き、`SHOW DATABASES` を使用する

か、`INFORMATION_SCHEMA` の `SCHEMATA` テーブルを調べることで、すべてのデータベース名を表示できます。

メモ

- `SCHEMATA_EXTENSIONS` テーブルは、スキーマオプションに関する情報で `SCHEMATA` テーブルを拡張します。

26.32 INFORMATION_SCHEMA SCHEMATA_EXTENSIONS テーブル

`SCHEMATA_EXTENSIONS` テーブル (MySQL 8.0.22 で使用可能) は、`SCHEMATA` テーブルをスキーマオプションに関する情報で拡張します。

`SCHEMATA_EXTENSIONS` テーブルには、次のカラムがあります:

- `CATALOG_NAME`
スキーマが属するカタログの名前。この値は常に `def` です。
- `SCHEMA_NAME`
スキーマの名前。
- `OPTIONS`
スキーマのオプション。スキーマが読み取り専用の場合、値には `READ ONLY=1` が含まれます。スキーマが読み取り専用でない場合、`READ ONLY` オプションは表示されません。

例

```
mysql> ALTER SCHEMA mydb READ ONLY = 1;
mysql> SELECT * FROM INFORMATION_SCHEMA.SCHEMATA_EXTENSIONS
  WHERE SCHEMA_NAME = 'mydb';
+-----+-----+
| CATALOG_NAME | SCHEMA_NAME | OPTIONS |
+-----+-----+
| def          | mydb        | READ ONLY=1 |
+-----+-----+

mysql> ALTER SCHEMA mydb READ ONLY = 0;
mysql> SELECT * FROM INFORMATION_SCHEMA.SCHEMATA_EXTENSIONS
  WHERE SCHEMA_NAME = 'mydb';
+-----+-----+
| CATALOG_NAME | SCHEMA_NAME | OPTIONS |
+-----+-----+
| def          | mydb        |         |
+-----+-----+
```

メモ

- `SCHEMATA_EXTENSIONS` は非標準の `INFORMATION_SCHEMA` テーブルです。

26.33 INFORMATION_SCHEMA SCHEMA_PRIVILEGES テーブル

`SCHEMA_PRIVILEGES` テーブルは、スキーマ (データベース) 権限に関する情報を提供します。 `mysql.db` システムテーブルから値を取得します。

`SCHEMA_PRIVILEGES` テーブルには、次のカラムがあります:

- `GRANTEE`
権限が付与されるアカウントの名前 ('user_name'@'host_name'形式)。
- `TABLE_CATALOG`

スキーマが属するカタログの名前。この値は常に `def` です。

- `TABLE_SCHEMA`

スキーマの名前。

- `PRIVILEGE_TYPE`

付与された権限。この値は、スキーマレベルで付与できる任意の権限にできます。[セクション13.7.1.6「GRANT ステートメント」](#)を参照してください。各行には単一の権限がリストされるため、権限受領者が保持するスキーマ権限ごとに1つの行があります。

- `IS_GRANTABLE`

ユーザーが `GRANT OPTION` 権限を持っている場合は `YES`、それ以外の場合は `NO`。この出力では、`GRANT OPTION` は `PRIVILEGE_TYPE='GRANT OPTION'` とは別の行としてリストされません。

メモ

- `SCHEMA_PRIVILEGES` は非標準の `INFORMATION_SCHEMA` テーブルです。

次のステートメントは同等ではありません。

```
SELECT ... FROM INFORMATION_SCHEMA.SCHEMA_PRIVILEGES
SHOW GRANTS ...
```

26.34 INFORMATION_SCHEMA STATISTICS テーブル

`STATISTICS` テーブルは、テーブルインデックスの情報を提供します。

テーブル統計を表す `STATISTICS` のカラムには、キャッシュされた値が保持されます。

`information_schema_stats_expiry` システム変数は、キャッシュされたテーブルの統計が期限切れになるまでの期間を定義します。デフォルトは 86400 秒 (24 時間) です。キャッシュされた統計がない場合、または統計が期限切れになっている場合は、テーブル統計カラムのクエリー時にストレージエンジンから統計が取得されます。特定のテーブルのキャッシュされた値をいつでも更新するには、`ANALYZE TABLE` を使用します。常に最新の統計をストレージエンジンから直接取得するには、`information_schema_stats_expiry=0` を設定します。詳細は、[セクション 8.2.3「INFORMATION_SCHEMA クエリーの最適化」](#)を参照してください。

注記

`innodb_read_only` システム変数が有効になっている場合、`InnoDB` を使用するデータディクショナリの統計テーブルを更新できないため、`ANALYZE TABLE` が失敗することがあります。キー分散を更新する `ANALYZE TABLE` 操作では、操作によってテーブル自体が更新された場合でも (`MyISAM` テーブルの場合など)、障害が発生する可能性があります。更新された分散統計を取得するには、`information_schema_stats_expiry=0` を設定します。

`STATISTICS` テーブルには、次のカラムがあります:

- `TABLE_CATALOG`

インデックスを含むテーブルが属するカタログの名前。この値は常に `def` です。

- `TABLE_SCHEMA`

インデックスを含むテーブルが属するスキーマ (データベース) の名前。

- `TABLE_NAME`

インデックスを含むテーブルの名前。

- `NON_UNIQUE`

このインデックスが重複を含むことができない場合は 0、できる場合は 1。

- **INDEX_SCHEMA**

インデックスが属するスキーマ (データベース) の名前。

- **INDEX_NAME**

インデックスの名前。このインデックスが主キーである場合、その名前は常に **PRIMARY** です。

- **SEQ_IN_INDEX**

インデックス内のコラムシーケンス番号であり、1 から始まります。

- **COLUMN_NAME**

コラム名。 **EXPRESSION** コラムの説明も参照してください。

- **COLLATION**

インデックス内でのコラムのソート方法。これには、**A** (昇順)、**D** (降順) または **NULL** (ソートなし) の値を指定できます。

- **CARDINALITY**

このインデックス内の一意の値の数の推定値。この数を更新するには、**ANALYZE TABLE** または (MyISAM テーブルの場合は) **myisamchk -a** を実行します。

CARDINALITY は整数として格納された統計に基づいてカウントされるため、小さいテーブルの場合でも値が正確であるとはかぎりません。カーディナリティーが高いほど、MySQL が結合を実行するときこのインデックスを使用する可能性は高くなります。

- **SUB_PART**

インデックス接頭辞。つまり、コラムが部分的にのみインデックス付けされている場合はインデックス付けされた文字の数で、コラム全体がインデックス付けされている場合は **NULL** です。

注記

接頭辞 **limits** はバイト単位で測定されます。ただし、**CREATE TABLE**、**ALTER TABLE** および **CREATE INDEX** ステートメントのインデックス指定の接頭辞 **lengths** は、非バイナリ文字列型 (**CHAR**、**VARCHAR**、**TEXT**) の場合は文字数として解釈され、バイナリ文字列型 (**BINARY**、**VARBINARY**、**BLOB**) の場合はバイト数として解釈されます。マルチバイト文字セットを使用する非バイナリ文字列コラムに接頭辞の長さを指定する場合は、これを考慮してください。

インデックス接頭辞の詳細は、[セクション8.3.5「コラムインデックス」](#) および [セクション13.1.15「CREATE INDEX ステートメント」](#) を参照してください。

- **PACKED**

キーがパックされる方法を示します。パックされない場合は **NULL**。

- **NULLABLE**

このコラムに **NULL** 値を含めることができる場合は **YES** が、できない場合は **"** が含まれます。

- **INDEX_TYPE**

使用されるインデックス方法 (**BTREE**、**FULLTEXT**、**HASH**、**RTREE**)。

- **COMMENT**

各コラムで説明されていないこのインデックスに関する情報 (このインデックスが無効になっている場合の **disabled** など)。

- **INDEX_COMMENT**

このインデックスが作成されたときに `COMMENT` 属性でインデックスに対して提供された任意のコメント。

- `IS_VISIBLE`

オプティマイザがインデックスを参照できるかどうか。 [セクション8.3.12「不可視のインデックス」](#) を参照してください。

- `EXPRESSION`

MySQL 8.0.13 以上では、`COLUMN_NAME` カラムと `EXPRESSION` カラムの両方に影響する機能キー部分 ([機能キー部品](#) を参照) がサポートされています:

- 機能しないキー部分の場合、`COLUMN_NAME` はキー部分でインデックス付けされたカラムを示し、`EXPRESSION` は `NULL` です。
- 関数キー部分の場合、`COLUMN_NAME` カラムは `NULL` で、`EXPRESSION` はキー部分の式を示します。

メモ

- インデックス用の標準 `INFORMATION_SCHEMA` テーブルはありません。MySQL のカラムリストは、`QUALIFIER` および `OWNER` がそれぞれ `CATALOG` および `SCHEMA` に置き換えられる点を除き、SQL Server 2000 が `sp_statistics` に対して返すものと似ています。

テーブルインデックスに関する情報は、`SHOW INDEX` ステートメントからも入手できます。 [セクション13.7.7.22「SHOW INDEX ステートメント」](#) を参照してください。 次のステートメントは同等です。

```
SELECT * FROM INFORMATION_SCHEMA.STATISTICS
WHERE table_name = 'tbl_name'
AND table_schema = 'db_name'

SHOW INDEX
FROM tbl_name
FROM db_name
```

26.35 INFORMATION_SCHEMA ST_GEOMETRY_COLUMNS テーブル

`ST_GEOMETRY_COLUMNS` テーブルは、空間データを格納するテーブルのカラムに関する情報を提供します。このテーブルは、SQL/MM (ISO/IEC 13249-3) 標準に基づいており、記載されている拡張機能があります。MySQL は、`ST_GEOMETRY_COLUMNS` を `INFORMATION_SCHEMA COLUMNS` テーブルのビューとして実装します。

`ST_GEOMETRY_COLUMNS` テーブルには、次のカラムがあります:

- `TABLE_CATALOG`

カラムを含むテーブルが属するカタログの名前。この値は常に `def` です。

- `TABLE_SCHEMA`

カラムを含むテーブルが属するスキーマ (データベース) の名前。

- `TABLE_NAME`

カラムを含むテーブルの名前。

- `COLUMN_NAME`

カラムの名前。

- `SRS_NAME`

空間参照システム (SRS) 名。

- `SRS_ID`

空間参照システム ID (SRID)。

- [GEOMETRY_TYPE_NAME](#)

カラムのデータ型。許可される値は次のとおりです: [geometry](#), [point](#), [linestring](#), [polygon](#), [multipoint](#), [multilinestring](#), [multipolygon](#), [geometrycollection](#)。このカラムは、標準に対する MySQL の拡張機能です。

26.36 INFORMATION_SCHEMA ST_SPATIAL_REFERENCE_SYSTEMS テーブル

[ST_SPATIAL_REFERENCE_SYSTEMS](#) テーブルには、空間データに使用可能な空間参照システム (SRS) に関する情報が表示されます。このテーブルは、SQL/MM (ISO/IEC 13249-3) 標準に基づいています。

[ST_SPATIAL_REFERENCE_SYSTEMS](#) テーブルのエントリは「[欧州石油調査グループ](#)」(EPSG) データセットに基づいていますが、SRID 0 は例外です。SRID 0 は、軸に単位が割り当てられていない無限平坦なデカルト面を表す MySQL で使用される特別な SRS に相当します。SRS の詳細は、[セクション11.4.5「空間参照システムのサポート」](#)を参照してください。

[ST_SPATIAL_REFERENCE_SYSTEMS](#) テーブルには、次のカラムがあります:

- [SRS_NAME](#)

空間参照システム名。この値は一意です。

- [SRS_ID](#)

空間参照システムの数値 ID。この値は一意です。

[SRS_ID](#) 値は、ジオメトリ値の SRID と同じ種類の値を表すが、SRID 引数として空間関数に渡されます。SRID 0 (単位なしデカルト平面) は特殊です。常に有効な空間参照システム ID であり、SRID 値に依存する空間データの計算に使用できます。

- [ORGANIZATION](#)

空間参照系の基になる座標系を定義した組織の名前。

- [ORGANIZATION_COORDSYS_ID](#)

空間参照システムを定義した組織によって空間参照システムに指定された数値 ID。

- [DEFINITION](#)

空間参照システム定義。 [DEFINITION](#) 値は WKT 値で、[Open Geospatial Consortium](#) ドキュメント [OGC 12-063r5](#) で指定されているとおりに表されます。

SRS 定義解析は、GIS 関数で定義が必要な場合にオンデマンドで実行されます。解析された定義は、再利用を可能にし、SRS 情報を必要とするすべてのステートメントの解析オーバーヘッドが発生しないように、データディクショナリキャッシュに格納されます。

- [DESCRIPTION](#)

空間参照システムの説明。

メモ

- [SRS_NAME](#), [ORGANIZATION](#), [ORGANIZATION_COORDSYS_ID](#) および [DESCRIPTION](#) のカラムには、ユーザーにとって関心のある情報が含まれていますが、MySQL では使用されません。

例

```
mysql> SELECT * FROM ST_SPATIAL_REFERENCE_SYSTEMS
```

```

WHERE SRS_ID = 4326\G
***** 1. row *****
      SRS_NAME: WGS 84
      SRS_ID: 4326
      ORGANIZATION: EPSG
      ORGANIZATION_COORDSYS_ID: 4326
      DEFINITION: GEOGCS["WGS 84",DATUM["World Geodetic System 1984",
      SPHEROID["WGS 84",6378137,298.257223563,
      AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],
      PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],
      UNIT["degree",0.017453292519943278,
      AUTHORITY["EPSG","9122"]],
      AXIS["Lat",NORTH],AXIS["Long",EAST],
      AUTHORITY["EPSG","4326"]]
      DESCRIPTION:

```

このエントリでは、GPS システムに使用される SRS について説明します。名前 (SRS_NAME) は WGS 84 で、ID (SRS_ID) は 4326 で、これは「[欧州石油調査グループ](#)」(EPSG) で使用される ID です。

投影 SRS および地理 SRS の DEFINITION 値は、それぞれ PROJCS および GEOGCS で始まります。SRID 0 の定義は特殊で、空の DEFINITION 値を持ちます。次のクエリーでは、DEFINITION 値に基づいて、ST_SPATIAL_REFERENCE_SYSTEMS テーブル内のエントリのうち、投影 SR、地理的 SR およびその他の SRS に対応するエントリ数が決定されます:

```

mysql> SELECT
COUNT(*),
CASE LEFT(DEFINITION, 6)
  WHEN 'PROJCS' THEN 'Projected'
  WHEN 'GEOGCS' THEN 'Geographic'
  ELSE 'Other'
END AS SRS_TYPE
FROM INFORMATION_SCHEMA.ST_SPATIAL_REFERENCE_SYSTEMS
GROUP BY SRS_TYPE;
+-----+-----+
| COUNT(*) | SRS_TYPE |
+-----+-----+
| 1 | Other |
| 4668 | Projected |
| 483 | Geographic |
+-----+-----+

```

データディクショナリに格納されている SRS エントリの操作を可能にするために、MySQL には次の SQL ステートメントが用意されています:

- [CREATE SPATIAL REFERENCE SYSTEM](#): [セクション13.1.19「CREATE SPATIAL REFERENCE SYSTEM ステートメント」](#) を参照してください。このステートメントの説明には、SRS 構成部品に関する追加情報が含まれます。
- [DROP SPATIAL REFERENCE SYSTEM](#): [セクション13.1.31「DROP SPATIAL REFERENCE SYSTEM ステートメント」](#) を参照してください。

26.37 INFORMATION_SCHEMA ST_UNITS_OF_MEASURE テーブル

ST_UNITS_OF_MEASURE テーブル (MySQL 8.0.14 の時点で使用可能) には、ST_Distance() 関数で使用可能なユニットに関する情報が表示されます。

ST_UNITS_OF_MEASURE テーブルには、次のカラムがあります:

- **UNIT_NAME**
単位の名前。
- **UNIT_TYPE**
単位タイプ (たとえば、[LINEAR](#))。
- **CONVERSION_FACTOR**

内部計算に使用される換算係数。

- [DESCRIPTION](#)

単位の説明。

26.38 INFORMATION_SCHEMA TABLES テーブル

[TABLES](#) テーブルは、データベース内のテーブルに関する情報を提供します。

テーブル統計を表す [TABLES](#) のカラムには、キャッシュされた値が保持されます。 [information_schema_stats_expiry](#) システム変数は、キャッシュされたテーブルの統計が期限切れになるまでの期間を定義します。 デフォルトは 86400 秒 (24 時間) です。 キャッシュされた統計がない場合、または統計が期限切れになっている場合は、テーブル統計カラムのクエリー時にストレージエンジンから統計が取得されます。 特定のテーブルのキャッシュされた値をいつでも更新するには、[ANALYZE TABLE](#) を使用します。 常に最新の統計をストレージエンジンから直接取得するには、[information_schema_stats_expiry](#) を 0 に設定します。 詳細は、[セクション8.2.3「INFORMATION_SCHEMA クエリーの最適化」](#)を参照してください。

注記

[innodb_read_only](#) システム変数が有効になっている場合、[InnoDB](#) を使用するデータディクショナリの統計テーブルを更新できないため、[ANALYZE TABLE](#) が失敗することがあります。 キー分散を更新する [ANALYZE TABLE](#) 操作では、操作によってテーブル自体が更新された場合でも ([MyISAM](#) テーブルの場合など)、障害が発生する可能性があります。 更新された分散統計を取得するには、[information_schema_stats_expiry=0](#) を設定します。

[TABLES](#) テーブルには、次のカラムがあります:

- [TABLE_CATALOG](#)

テーブルが属するカタログの名前。 この値は常に `def` です。

- [TABLE_SCHEMA](#)

テーブルが属するスキーマ (データベース) の名前。

- [TABLE_NAME](#)

テーブルの名前。

- [TABLE_TYPE](#)

テーブルの場合は [BASE TABLE](#)、ビューの場合は [VIEW](#)、[INFORMATION_SCHEMA](#) テーブルの場合は [SYSTEM VIEW](#) です。

[TABLES](#) テーブルには、[TEMPORARY](#) テーブルはリストされません。

- [ENGINE](#)

このテーブルのストレージエンジン。 [第15章「InnoDB ストレージエンジン」](#) および [第16章「代替ストレージエンジン」](#)を参照してください。

パーティション化されたテーブルの場合、[ENGINE](#) には、すべてのパーティションで使用されるストレージエンジンの名前が表示されます。

- [VERSION](#)

このカラムは未使用です。 MySQL 8.0 で `.frm` ファイルを削除すると、このカラムには、MySQL 5.7 で最後に使用された `.frm` ファイルバージョンである 10 のハードコードされた値がレポートされるようになりました。

- [ROW_FORMAT](#)

行ストレージフォーマット ([Fixed](#)、[Dynamic](#)、[Compressed](#)、[Redundant](#)、[Compact](#))。 [MyISAM](#) テーブルの場合、[Dynamic](#) は、`myisamchk -dvv` が [Packed](#) としてレポートする内容に対応します。

- **TABLE_ROWS**

行数。MyISAM などの一部のストレージエンジンは、正確な数を格納します。InnoDB などのほかのストレージエンジンの場合、この値は概算であり、実際の値と 40% から 50% まで異なる可能性があります。このような場合、正確な数を取得するには `SELECT COUNT(*)` を使用します。

TABLE_ROWS は、INFORMATION_SCHEMA テーブル用の NULL です。

InnoDB テーブルの場合、行カウントは SQL 最適化で使用される単なる概算です。(InnoDB テーブルがパーティション化されている場合も、これは当てはまります。)

- **AVG_ROW_LENGTH**

平均行長。

- **DATA_LENGTH**

MyISAM の場合、DATA_LENGTH はデータファイルの長さ (バイト単位) です。

InnoDB の場合、DATA_LENGTH は、クラスタ化されたインデックスに割り当てられるおおよその容量 (バイト単位) です。具体的には、クラスタ化されたインデックスサイズ (ページ単位) に InnoDB ページサイズを乗算したものです。

その他のストレージエンジンについては、このセクションの最後にある注を参照してください。

- **MAX_DATA_LENGTH**

MyISAM の場合、MAX_DATA_LENGTH はデータファイルの最大長です。これは、このテーブル内に格納できるデータの合計バイト数です (使用されるデータポイントサイズが指定された場合)。

InnoDB では未使用です。

その他のストレージエンジンについては、このセクションの最後にある注を参照してください。

- **INDEX_LENGTH**

MyISAM の場合、INDEX_LENGTH はインデックスファイルの長さ (バイト単位) です。

InnoDB の場合、INDEX_LENGTH は、クラスタ化されていないインデックスに割り当てられる領域の概算量 (バイト単位) です。具体的には、クラスタ化されていないインデックスサイズの合計 (ページ数) に InnoDB ページサイズを乗算した値です。

その他のストレージエンジンについては、このセクションの最後にある注を参照してください。

- **DATA_FREE**

割り当てられているが、使用されていないバイト数。

InnoDB テーブルは、このテーブルが属するテーブルスペースの空き領域をレポートします。共有テーブルスペース内に存在するテーブルの場合、これはその共有テーブルスペースの空き領域です。複数のテーブルスペースを使用していて、このテーブルに独自のテーブルスペースがある場合は、そのテーブルのみの空き領域になります。空き領域とは、完全な空きエクステントから安全上のマージンを引いたバイト数を示します。空き領域が 0 として表示されている場合でも、新しいエクステントを割り当てる必要がないかぎり、行を挿入できる可能性があります。

NDB Cluster の場合、DATA_FREE は、ディスク上の「ディスクデータ」テーブルまたはフラグメント用にディスクに割り当てられたが使用されていない領域を表示します。(メモリー内データリソース使用率は、DATA_LENGTH カラムによってレポートされます。)

パーティション化されたテーブルの場合、この値は推定値にすぎず、絶対的に正しいとはかぎりません。このような場合にこの情報を取得するより正確な方法は、次の例に示すように、INFORMATION_SCHEMA PARTITIONS テーブルをクエリーすることです:

```
SELECT SUM(DATA_FREE)
FROM INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_SCHEMA = 'mydb'
```



```
AND TABLE_NAME = 'mytable';
```

詳細は、[セクション26.21「INFORMATION_SCHEMA PARTITIONS テーブル」](#)を参照してください。

- [AUTO_INCREMENT](#)

次の [AUTO_INCREMENT](#) 値。

- [CREATE_TIME](#)

いつテーブルが作成されたか。

- [UPDATE_TIME](#)

いつデータファイルが最後に更新されたか。一部のストレージエンジンでは、この値は `NULL` です。たとえば、[InnoDB](#) はその [システムテーブルスペース](#) 内に複数のテーブルを格納するため、データファイルのタイムスタンプは適用されません。各 [InnoDB](#) テーブルが個別の `.ibd` ファイル内に存在する [file-per-table](#) モードの場合でも、[変更バッキング](#) によってデータファイルへの書き込みが遅延される可能性があるため、ファイルの変更時間は最後の挿入、更新、または削除の時間とは異なります。[MyISAM](#) の場合、データファイルのタイムスタンプが使用されますが、Windows ではタイムスタンプは更新によって更新されないため、値は正確ではありません。

[UPDATE_TIME](#) には、パーティション化されていない [InnoDB](#) テーブルに対して最後に実行された [UPDATE](#)、[INSERT](#) または [DELETE](#) のタイムスタンプ値が表示されます。MVCC の場合、タイムスタンプ値は最終更新時間とみなされる [COMMIT](#) 時間を反映します。タイムスタンプは、サーバーの再起動時、またはテーブルが [InnoDB](#) データディクショナリキャッシュから削除されたときに永続化されません。

- [CHECK_TIME](#)

いつテーブルが最後にチェックされたか。すべてのストレージエンジンがこの時間を更新するわけではありません。この場合、値は常に `NULL` です。

パーティション化された [InnoDB](#) テーブルの場合、[CHECK_TIME](#) は常に `NULL` です。

- [TABLE_COLLATION](#)

テーブルのデフォルトの照合。出力にはテーブルのデフォルトの文字セットは明示的にリストされませんが、照合順序名は文字セット名で始まります。

- [CHECKSUM](#)

ライブチェックサム値 (存在する場合)。

- [CREATE_OPTIONS](#)

[CREATE TABLE](#) で使用される追加のオプション。

[CREATE_OPTIONS](#) には、パーティションテーブルの [partitioned](#) が表示されます。

MySQL 8.0.16 より前の [CREATE_OPTIONS](#) では、[file-per-table](#) テーブルスペースに作成されたテーブルに指定された [ENCRYPTION](#) 句が表示されます。MySQL 8.0.16 では、テーブルが暗号化されている場合、または指定された暗号化がスキーマ暗号化と異なる場合、[file-per-table](#) テーブルスペースの暗号化句が表示されます。暗号化句は、一般テーブルスペースに作成されたテーブルには表示されません。暗号化された [file-per-table](#) および一般的なテーブルスペースを識別するには、[INNODB_TABLESPACES ENCRYPTION](#) カラムをクエリーします。

[strict mode](#) を無効にしてテーブルを作成する場合、指定した行フォーマットがサポートされていないと、ストレージエンジンのデフォルトの行フォーマットが使用されます。テーブルの実際の行形式は、[ROW_FORMAT](#) カラムにレポートされます。[CREATE_OPTIONS](#) には、[CREATE TABLE](#) ステートメントで指定された行形式が表示されます。

テーブルのストレージエンジンを変更する場合、新しいストレージエンジンに適用できないテーブルオプションはテーブル定義に保持され、必要に応じて、以前に定義されたオプションを持つテーブルを元のストレージエンジンに戻すことができます。[CREATE_OPTIONS](#) カラムに保持オプションが表示される場合があります。

- [TABLE_COMMENT](#)

このテーブルを作成するときに使用されたコメント (または、MySQL がテーブル情報にアクセスできなかった理由に関する情報)。

メモ

- NDB テーブルの場合、このステートメントの出力には `AVG_ROW_LENGTH` カラムと `DATA_LENGTH` カラムの適切な値が表示されますが、`BLOB` カラムは考慮されません。
- NDB テーブルの場合、`DATA_LENGTH` にはメインメモリーに格納されているデータのみが含まれます。`MAX_DATA_LENGTH` および `DATA_FREE` カラムはディスクデータに適用されます。
- 「NDB Cluster ディスクデータの場合」テーブル、`MAX_DATA_LENGTH` には、「ディスクデータ」テーブルまたはフラグメントのディスク部分に割り当てられた領域が表示されます。(メモリー内データリソース使用率は、`DATA_LENGTH` カラムによってレポートされます。)
- MEMORY テーブルの場合、`DATA_LENGTH`、`MAX_DATA_LENGTH` および `INDEX_LENGTH` の値は、割り当てられたメモリーの実際の量を概算します。割り当てアルゴリズムは、割り当て操作の数を減らすために、大量のメモリーを確保します。
- ビューの場合、`TABLE_NAME` がビュー名を示し、`CREATE_TIME` が作成時間を示し、`TABLE_COMMENT` が `VIEW` を示すことを除き、ほとんどの `TABLES` カラムは 0 または `NULL` です。

テーブル情報は、`SHOW TABLE STATUS` ステートメントおよび `SHOW TABLES` ステートメントからも入手できます。[セクション13.7.7.38「SHOW TABLE STATUS ステートメント」](#) および [セクション13.7.7.39「SHOW TABLES ステートメント」](#) を参照してください。次のステートメントは同等です。

```
SELECT
  TABLE_NAME, ENGINE, VERSION, ROW_FORMAT, TABLE_ROWS, AVG_ROW_LENGTH,
  DATA_LENGTH, MAX_DATA_LENGTH, INDEX_LENGTH, DATA_FREE, AUTO_INCREMENT,
  CREATE_TIME, UPDATE_TIME, CHECK_TIME, TABLE_COLLATION, CHECKSUM,
  CREATE_OPTIONS, TABLE_COMMENT
FROM INFORMATION_SCHEMA.TABLES
WHERE table_schema = 'db_name'
[AND table_name LIKE 'wild']

SHOW TABLE STATUS
FROM db_name
[LIKE 'wild']
```

次のステートメントは同等です。

```
SELECT
  TABLE_NAME, TABLE_TYPE
FROM INFORMATION_SCHEMA.TABLES
WHERE table_schema = 'db_name'
[AND table_name LIKE 'wild']

SHOW FULL TABLES
FROM db_name
[LIKE 'wild']
```

26.39 INFORMATION_SCHEMA TABLES_EXTENSIONS テーブル

`TABLES_EXTENSIONS` テーブル (MySQL 8.0.21 の時点で使用可能) は、プライマリストレージエンジンおよびセカンダリストレージエンジン用に定義されたテーブル属性に関する情報を提供します。

注記

`TABLES_EXTENSIONS` テーブルは、将来の使用のために予約されています。

`TABLES_EXTENSIONS` テーブルには、次のカラムがあります:

- `TABLE_CATALOG`

テーブルが属するカタログの名前。この値は常に `def` です。

- [TABLE_SCHEMA](#)
テーブルが属するスキーマ (データベース) の名前。
- [TABLE_NAME](#)
テーブルの名前。
- [ENGINE_ATTRIBUTE](#)
プライマリストレージエンジンに定義されているテーブル属性。将来使用するために予約されています。
- [SECONDARY_ENGINE_ATTRIBUTE](#)
セカンダリストレージエンジンに定義されているテーブル属性。将来使用するために予約されています。

26.40 INFORMATION_SCHEMA TABLESPACES テーブル

このテーブルは未使用です。非推奨になりました。将来の MySQL リリースで削除される予定です。その他の [INFORMATION_SCHEMA](#) テーブルでは、関連情報が提供される場合があります:

- [NDB](#) の場合、[INFORMATION_SCHEMA FILES](#) テーブルにはテーブルスペース関連の情報が表示されます。
- [InnoDB](#) の場合、[INFORMATION_SCHEMA INNODB_TABLESPACES](#) および [INNODB_DATAFILES](#) テーブルはテーブルスペースメタデータを提供します。

26.41 INFORMATION_SCHEMA TABLESPACES_EXTENSIONS テーブル

[TABLESPACES_EXTENSIONS](#) テーブル (MySQL 8.0.21 の時点で使用可能) は、プライマリストレージエンジンに定義されているテーブルスペース属性に関する情報を提供します。

注記

[TABLESPACES_EXTENSIONS](#) テーブルは、将来の使用のために予約されています。

[TABLESPACES_EXTENSIONS](#) テーブルには、次のカラムがあります:

- [TABLESPACE_NAME](#)
テーブルスペースの名前。
- [ENGINE_ATTRIBUTE](#)
プライマリストレージエンジンに定義されているテーブルスペース属性。将来使用するために予約されています。

26.42 INFORMATION_SCHEMA TABLE_CONSTRAINTS テーブル

[TABLE_CONSTRAINTS](#) テーブルは、どのテーブルに制約があるかを説明します。

[TABLE_CONSTRAINTS](#) テーブルには、次のカラムがあります:

- [CONSTRAINT_CATALOG](#)
制約が属するカタログの名前。この値は常に `def` です。
- [CONSTRAINT_SCHEMA](#)
制約が属するスキーマ (データベース) の名前。
- [TABLE_SCHEMA](#)
テーブルが属するスキーマ (データベース) の名前。

- **TABLE_NAME**
テーブルの名前。
- The **CONSTRAINT_TYPE**
制約のタイプ。値は、**UNIQUE**, **PRIMARY KEY**, **FOREIGN KEY** または (MySQL 8.0.16) **CHECK** です。これは **CHAR** (非 **ENUM**) カラムです。
UNIQUE および **PRIMARY KEY** の情報は、**Non_unique** カラムが 0 の場合、**SHOW INDEX** からの出力の **Key_name** カラムから取得した情報とほぼ同じです。
- **ENFORCED**
CHECK 制約の場合、制約が施行されるかどうかを示す値は **YES** または **NO** です。その他の制約の場合、値は常に **YES** です。
このカラムは、MySQL 8.0.16 で追加されました。

26.43 INFORMATION_SCHEMA TABLE_CONSTRAINTS_EXTENSIONS テーブル

TABLE_CONSTRAINTS_EXTENSIONS テーブル (MySQL 8.0.21 の時点で使用可能) は、プライマリストレージエンジンおよびセカンダリストレージエンジン用に定義されたテーブル制約属性に関する情報を提供します。

注記

TABLE_CONSTRAINTS_EXTENSIONS テーブルは、将来の使用のために予約されています。

TABLE_CONSTRAINTS_EXTENSIONS テーブルには、次のカラムがあります:

- **CONSTRAINT_CATALOG**
テーブルが属するカタログの名前。
- **CONSTRAINT_SCHEMA**
テーブルが属するスキーマ (データベース) の名前。
- **CONSTRAINT_NAME**
制約の名前。
- **TABLE_NAME**
テーブルの名前。
- **ENGINE_ATTRIBUTE**
プライマリストレージエンジンに定義されている制約属性。将来使用するために予約されています。
- **SECONDARY_ENGINE_ATTRIBUTE**
セカンダリストレージエンジンに定義された制約属性。将来使用するために予約されています。

26.44 INFORMATION_SCHEMA TABLE_PRIVILEGES テーブル

TABLE_PRIVILEGES テーブルは、テーブル権限に関する情報を提供します。 `mysql.tables_priv` システムテーブルから値を取得します。

TABLE_PRIVILEGES テーブルには、次のカラムがあります:

- **GRANTEE**

権限が付与されるアカウントの名前 ('user_name'@'host_name'形式)。

- [TABLE_CATALOG](#)

テーブルが属するカタログの名前。この値は常に `def` です。

- [TABLE_SCHEMA](#)

テーブルが属するスキーマ (データベース) の名前。

- [TABLE_NAME](#)

テーブルの名前。

- [PRIVILEGE_TYPE](#)

付与された権限。この値は、テーブルレベルで付与できる任意の権限です。[セクション13.7.1.6「GRANT ステートメント」](#)を参照してください。各行には単一の権限がリストされるため、権限受領者が保持するテーブル権限ごとに1つの行があります。

- [IS_GRANTABLE](#)

ユーザーが `GRANT OPTION` 権限を持っている場合は `YES`、それ以外の場合は `NO`。この出力では、`GRANT OPTION` は `PRIVILEGE_TYPE='GRANT OPTION'`とは別の行としてリストされません。

メモ

- [TABLE_PRIVILEGES](#) は非標準の `INFORMATION_SCHEMA` テーブルです。

次のステートメントは同等ではありません。

```
SELECT ... FROM INFORMATION_SCHEMA.TABLE_PRIVILEGES  
  
SHOW GRANTS ...
```

26.45 INFORMATION_SCHEMA TRIGGERS テーブル

`TRIGGERS` テーブルはトリガーに関する情報を提供します。テーブルトリガーに関する情報を表示するには、そのテーブルに対する `TRIGGER` 権限が必要です。

`TRIGGERS` テーブルには、次のカラムがあります:

- [TRIGGER_CATALOG](#)

トリガーが属するカタログの名前。この値は常に `def` です。

- [TRIGGER_SCHEMA](#)

トリガーが属するスキーマ (データベース) の名前。

- [TRIGGER_NAME](#)

トリガーの名前。

- [EVENT_MANIPULATION](#)

トリガーイベント。これは、トリガーが有効になる、関連付けられたテーブルに対する操作の種類です。値は、`INSERT` (行が挿入された場合)、`DELETE` (行が削除された場合) または `UPDATE` (行が変更された場合) です。

- [EVENT_OBJECT_CATALOG](#)、[EVENT_OBJECT_SCHEMA](#) および [EVENT_OBJECT_TABLE](#)

[セクション25.3「トリガーの使用」](#)で説明されているように、すべてのトリガーは1つのテーブルにのみ関連付けられます。これらのカラムは、このテーブルが発生するカタログおよびスキーマ (データベース) と、それぞれテーブル名を示します。`EVENT_OBJECT_CATALOG` 値は常に `def` です。

- **ACTION_ORDER**
同じ **EVENT_MANIPULATION** 値と **ACTION_TIMING** 値を持つ同じテーブルのトリガーリスト内のトリガーアクションの順序位置。
- **ACTION_CONDITION**
この値は常に **NULL** です。
- **ACTION_STATEMENT**
トリガー本体 (トリガーがアクティブ化されたときに実行されるステートメント)。このテキストは UTF-8 エンコーディングを使用します。
- **ACTION_ORIENTATION**
この値は常に **ROW** です。
- **ACTION_TIMING**
トリガーを起動するイベントの前または後にトリガーをアクティブ化するかどうか。値は **BEFORE** または **AFTER** です。
- **ACTION_REFERENCE_OLD_TABLE**
この値は常に **NULL** です。
- **ACTION_REFERENCE_NEW_TABLE**
この値は常に **NULL** です。
- **ACTION_REFERENCE_OLD_ROW** および **ACTION_REFERENCE_NEW_ROW**
古いカラム識別子と新しいカラム識別子。 **ACTION_REFERENCE_OLD_ROW** の値は常に **OLD** で、 **ACTION_REFERENCE_NEW_ROW** の値は常に **NEW** です。
- **CREATED**
トリガーが作成された日時。これは、トリガーの **TIMESTAMP(2)** 値 (小数部は数百秒) です。
- **SQL_MODE**
トリガーの作成時およびトリガーの実行時に有効な SQL モード。指定可能な値については、[セクション 5.1.11 「サーバー SQL モード」](#) を参照してください。
- **DEFINER**
'user_name'@'host_name'形式の **DEFINER** 句で指定されたアカウント (多くの場合、トリガーを作成したユーザー)。
- **CHARACTER_SET_CLIENT**
トリガー作成時の **character_set_client** システム変数のセッション値。
- **COLLATION_CONNECTION**
トリガー作成時の **collation_connection** システム変数のセッション値。
- **DATABASE_COLLATION**
トリガーが関連付けられているデータベースの照合。

例

次の例では、[セクション 25.3 「トリガーの使用」](#) で定義されている **ins_sum** トリガーを使用します:


```
mysql> SELECT * FROM INFORMATION_SCHEMA.TRIGGERS
WHERE TRIGGER_SCHEMA='test' AND TRIGGER_NAME='ins_sum'\G
***** 1. row *****
TRIGGER_CATALOG: def
TRIGGER_SCHEMA: test
TRIGGER_NAME: ins_sum
EVENT_MANIPULATION: INSERT
EVENT_OBJECT_CATALOG: def
EVENT_OBJECT_SCHEMA: test
EVENT_OBJECT_TABLE: account
ACTION_ORDER: 1
ACTION_CONDITION: NULL
ACTION_STATEMENT: SET @sum = @sum + NEW.amount
ACTION_ORIENTATION: ROW
ACTION_TIMING: BEFORE
ACTION_REFERENCE_OLD_TABLE: NULL
ACTION_REFERENCE_NEW_TABLE: NULL
ACTION_REFERENCE_OLD_ROW: OLD
ACTION_REFERENCE_NEW_ROW: NEW
CREATED: 2018-08-08 10:10:12.61
SQL_MODE: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,
NO_ZERO_IN_DATE,NO_ZERO_DATE,
ERROR_FOR_DIVISION_BY_ZERO,
NO_ENGINE_SUBSTITUTION
DEFINER: me@localhost
CHARACTER_SET_CLIENT: utf8mb4
COLLATION_CONNECTION: utf8mb4_0900_ai_ci
DATABASE_COLLATION: utf8mb4_0900_ai_ci
```

トリガー情報は、[SHOW TRIGGERS](#) ステートメントからも入手できます。 [セクション13.7.7.40「SHOW TRIGGERS ステートメント」](#)を参照してください。

26.46 INFORMATION_SCHEMA USER_ATTRIBUTES テーブル

[USER_ATTRIBUTES](#) テーブル (MySQL 8.0.21 で使用可能) には、ユーザーコメントおよびユーザー属性に関する情報が表示されます。 [mysql.user](#) システムテーブルから値を取得します。

[USER_ATTRIBUTES](#) テーブルには、次のカラムがあります:

- [USER](#)

[ATTRIBUTE](#) カラムの値が適用されるアカウントのユーザー名部分。

- [HOST](#)

[ATTRIBUTE](#) カラムの値が適用されるアカウントのホスト名部分。

- [ATTRIBUTE](#)

[USER](#) カラムおよび [HOST](#) カラムで指定されたアカウントに属するユーザーコメントまたはユーザー属性 (あるいはその両方)。値は JSON オブジェクト表記です。属性は、[CREATE USER ... ATTRIBUTE ...](#) ステートメントまたは [ALTER USER ... ATTRIBUTE ...](#) ステートメントを使用して設定されたとおりに表示されます。ユーザーコメントは、キーとして [comment](#) を持つキーと値のペアとして表示されます。

たとえば、[CREATE USER 'bill'@'localhost' COMMENT 'A comment' ATTRIBUTE '{"foo": "bar", "bazz": "fazz"}'](#) というステートメントでは、[USER_ATTRIBUTES](#) テーブルに次の行が追加されます:

```
+-----+-----+-----+
| USER | HOST | ATTRIBUTE |
+-----+-----+-----+
| bill | localhost | {"foo": "bar", "bazz": "fazz", "comment": "A comment"} |
+-----+-----+-----+
```

メモ

- [USER_ATTRIBUTES](#) は非標準の [INFORMATION_SCHEMA](#) テーブルです。
- 引用符で囲まれていない文字列として特定のユーザーのユーザーコメントのみを取得するには、次のようなクエリを使用できます:

```
mysql> SELECT ATTRIBUTE->"$.comment" AS Comment
-> FROM INFORMATION_SCHEMA.USER_ATTRIBUTES
-> WHERE USER='bill' AND HOST='localhost';
+-----+
| Comment |
+-----+
| A comment |
+-----+
```

同様に、キーを使用して、特定のユーザー属性の引用符で囲まれていない値を取得できます。

- MySQL 8.0.22 より前は、`USER_ATTRIBUTES` のコンテンツには誰でもアクセスできます。MySQL 8.0.22 では、`USER_ATTRIBUTES` の内容に次のようにアクセスできます:
 - 次の場合、すべての行にアクセスできます:
 - 現在のスレッドはレプリカスレッドです。
 - アクセス制御システムが初期化されていません (たとえば、`--skip-grant-tables` オプションを使用してサーバーを起動した場合)。
 - 現在認証されているアカウントには、`mysql.user` システムテーブルに対する `UPDATE` または `SELECT` 権限があります。
 - 現在認証されているアカウントには、`CREATE USER` および `SYSTEM_USER` 権限があります。
 - それ以外の場合、現在認証されているアカウントはそのアカウントの行を表示できます。また、アカウントに `CREATE USER` 権限はあるが `SYSTEM_USER` 権限はない場合、`SYSTEM_USER` 権限を持たない他のすべてのアカウントの行を表示できます。

アカウントコメントおよび属性の指定の詳細は、[セクション13.7.1.3「CREATE USER ステートメント」](#) を参照してください。

26.47 INFORMATION_SCHEMA USER_PRIVILEGES テーブル

`USER_PRIVILEGES` テーブルは、グローバル権限に関する情報を提供します。`mysql.user` システムテーブルから値を取得します。

`USER_PRIVILEGES` テーブルには、次のカラムがあります:

- `GRANTEE`
権限が付与されるアカウントの名前 ('user_name'@'host_name'形式)。
- `TABLE_CATALOG`
カタログの名前。この値は常に `def` です。
- `PRIVILEGE_TYPE`
付与された権限。この値は、グローバルレベルで付与できる任意の権限です。[セクション13.7.1.6「GRANT ステートメント」](#) を参照してください。各行には単一の権限がリストされるため、権限受領者が保持するグローバル権限ごとに 1 つの行があります。
- `IS_GRANTABLE`
ユーザーが `GRANT OPTION` 権限を持っている場合は `YES`、それ以外の場合は `NO`。この出力では、`GRANT OPTION` は `PRIVILEGE_TYPE='GRANT OPTION'` とは別の行としてリストされません。

メモ

- `USER_PRIVILEGES` は非標準の `INFORMATION_SCHEMA` テーブルです。

次のステートメントは同等ではありません。

```
SELECT ... FROM INFORMATION_SCHEMA.USER_PRIVILEGES  
SHOW GRANTS ...
```

26.48 INFORMATION_SCHEMA VIEWS テーブル

VIEWS テーブルは、データベース内のビューに関する情報を提供します。このテーブルにアクセスするには **SHOW VIEW** 権限が必要です。

VIEWS テーブルには、次のカラムがあります:

- **TABLE_CATALOG**

ビューが属するカタログの名前。この値は常に **def** です。

- **TABLE_SCHEMA**

ビューが属するスキーマ (データベース) の名前。

- **TABLE_NAME**

ビューの名前。

- **VIEW_DEFINITION**

ビューの定義を提供する **SELECT** ステートメント。このカラムには、**SHOW CREATE VIEW** によって生成される **Create Table** カラムに表示されるもののほとんどが含まれます。**SELECT** より前の語をスキップし、**WITH CHECK OPTION** の語をスキップします。元のステートメントが次のとおりだったとします。

```
CREATE VIEW v AS  
SELECT s2,s1 FROM t  
WHERE s1 > 5  
ORDER BY s1  
WITH CHECK OPTION;
```

この場合、ビュー定義は次のようになります。

```
SELECT s2,s1 FROM t WHERE s1 > 5 ORDER BY s1
```

- **CHECK_OPTION**

CHECK_OPTION 属性の値。値は、**NONE**、**CASCADE** または **LOCAL** のいずれかです。

- **IS_UPDATABLE**

MySQL は、**CREATE VIEW** 時に、ビューの更新可能性フラグというフラグを設定します。**UPDATE** および **DELETE** (および同様の操作) がビューで有効な場合、フラグは **YES** (true) に設定されます。それ以外の場合、フラグは **NO** (false) に設定されます。**VIEWS** テーブルの **IS_UPDATABLE** カラムは、このフラグのステータスを表示します。これは、ビューが更新可能であるかどうかをサーバーが常に把握していることを意味します。

ビューが更新可能でない場合、**UPDATE**、**DELETE** および **INSERT** などのステートメントは無効であり、拒否されます。(ビューが更新可能な場合でも、ビューに挿入できない場合があります。詳細は、[セクション25.5.3「更新可能および挿入可能なビュー」](#) を参照してください。)

- **DEFINER**

ビューを作成したユーザーのアカウント ('user_name'@'host_name'形式)。

- **SECURITY_TYPE**

ビューの **SQL SECURITY** 特性。値は、**DEFINER** または **INVOKER** のいずれかです。

- **CHARACTER_SET_CLIENT**

ビュー作成時の **character_set_client** システム変数のセッション値。

- `COLLATION_CONNECTION`

ビュー作成時の `collation_connection` システム変数のセッション値。

メモ

MySQL では、様々な `sql_mode` 設定を使用して、サポートする SQL 構文のタイプをサーバーに通知できます。たとえば、ANSI SQL モードを使用すると、クエリーで、MySQL で標準 SQL 連結演算子の二重バー (`||`) が正しく解釈されます。その後、項目を連結するビューを作成した場合、`sql_mode` 設定を ANSI とは別の値に変更すると、そのビューが無効になるという懸念があります。ただし、そのようなことはありません。MySQL は、記述方法には関係なく、常にビュー定義を正規の形式で同じ方法で格納します。サーバーが二重バーの連結演算子を `CONCAT()` 関数にどのように変更するかを示す例を次に示します。

```
mysql> SET sql_mode = 'ANSI';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE VIEW test.v AS SELECT 'a' || 'b' as col1;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT VIEW_DEFINITION FROM INFORMATION_SCHEMA.VIEWS
       WHERE TABLE_SCHEMA = 'test' AND TABLE_NAME = 'v';
+-----+
| VIEW_DEFINITION |
+-----+
| select concat('a','b') AS `col1` |
+-----+
1 row in set (0.00 sec)
```

ビュー定義を正規の形式で格納する利点は、後で `sql_mode` の値を変更してもビューの結果に影響しないことです。ただし、追加の結果として、`SELECT` より前のコメントはサーバーによって定義から削除されます。

26.49 INFORMATION_SCHEMA VIEW_ROUTINE_USAGE テーブル

`VIEW_ROUTINE_USAGE` テーブル (MySQL 8.0.13 で使用可能) を使用すると、ビュー定義で使用されるストアードファンクションに関する情報にアクセスできます。このテーブルは、定義で使用される組み込み SQL 関数またはユーザー定義関数 (UDF) に関する情報を示していません。

情報を表示できるのは、権限を持つビューおよび権限を持つ関数のみです。

`VIEW_ROUTINE_USAGE` テーブルには、次のカラムがあります:

- `TABLE_CATALOG`

ビューが属するカタログの名前。この値は常に `def` です。

- `TABLE_SCHEMA`

ビューが属するスキーマ (データベース) の名前。

- `TABLE_NAME`

ビューの名前。

- `SPECIFIC_CATALOG`

ビュー定義で使用される関数が属するカタログの名前。この値は常に `def` です。

- `SPECIFIC_SCHEMA`

ビュー定義で使用される関数が属するスキーマ (データベース) の名前。

- `SPECIFIC_NAME`

ビュー定義で使用される関数の名前。

26.50 INFORMATION_SCHEMA VIEW_TABLE_USAGE テーブル

[VIEW_TABLE_USAGE](#) テーブル (MySQL 8.0.13 で使用可能) を使用すると、ビュー定義で使用されるテーブルおよびビューに関する情報にアクセスできます。

情報を表示できるのは、一部の権限を持つビューおよび一部の権限を持つテーブルの場合のみです。

[VIEW_TABLE_USAGE](#) テーブルには、次のカラムがあります:

- [VIEW_CATALOG](#)

ビューが属するカタログの名前。この値は常に `def` です。

- [VIEW_SCHEMA](#)

ビューが属するスキーマ (データベース) の名前。

- [VIEW_NAME](#)

ビューの名前。

- [TABLE_CATALOG](#)

ビュー定義で使用されるテーブルまたはビューが属するカタログの名前。この値は常に `def` です。

- [TABLE_SCHEMA](#)

ビュー定義で使用されるテーブルまたはビューが属するスキーマ (データベース) の名前。

- [TABLE_NAME](#)

ビュー定義で使用されるテーブルまたはビューの名前。

26.51 INFORMATION_SCHEMA InnoDB テーブル

このセクションでは、[InnoDB INFORMATION_SCHEMA](#) テーブルのテーブル定義について説明します。関連する情報と例については、[セクション15.15 「InnoDB INFORMATION_SCHEMA テーブル」](#) を参照してください。

[InnoDB INFORMATION_SCHEMA](#) テーブルは、進行中の [InnoDB](#) アクティビティのモニタリング、問題になる前の非効率性の検出、またはパフォーマンスおよび容量の問題のトラブルシューティングに使用できます。データベースが増大しさらにビジーになり、ハードウェアの容量の限度に達したときに、データベースがスムーズに実行し続けられるようにこれらの点をモニターし調整します。

26.51.1 INFORMATION_SCHEMA INNODB_BUFFER_PAGE テーブル

[INNODB_BUFFER_PAGE](#) テーブルには、[InnoDB buffer pool](#) の各 [page](#) に関する情報が表示されます。

関連する使用法と使用例については、[セクション15.15.5 「InnoDB INFORMATION_SCHEMA バッファプールテーブル」](#) を参照してください。

警告

[INNODB_BUFFER_PAGE](#) テーブルをクエリーすると、パフォーマンスに影響する可能性があります。パフォーマンスへの影響を認識し、許容できると判断した場合を除き、本番システムではこのテーブルをクエリーしないでください。本番システムのパフォーマンスへの影響を回避するには、調査する問題を再現し、テストインスタンスのバッファプール統計をクエリーします。

[INNODB_BUFFER_PAGE](#) テーブルには、次のカラムがあります:

- [POOL_ID](#)

バッファプール ID。これは、複数のバッファプールインスタンスを区別する識別子です。

- [BLOCK_ID](#)

バッファプールブロック ID。

- [SPACE](#)

テーブルスペース ID。 [INNODB_TABLES.SPACE](#) と同じ値です。

- [PAGE_NUMBER](#)

ページ番号。

- [PAGE_TYPE](#)

ページタイプ。次の表は、許可される値を示しています。

表 26.1 INNODB_BUFFER_PAGE.PAGE_TYPE の値

ページタイプ	説明
ALLOCATED	新規割当済ページ
BLOB	「非圧縮 BLOB」ページ
COMPRESSED_BLOB2	後続のコンポーネント BLOB ページ
COMPRESSED_BLOB	最初の圧縮 BLOB ページ
ENCRYPTED_RTREE	暗号化 R ツリー
EXTENT_DESCRIPTOR	エクステンツ記述子ページ
FILE_SPACE_HEADER	ファイルスペースヘッダー
FIL_PAGE_TYPE_UNUSED	未使用
IBUF_BITMAP	バッファビットマップの挿入
IBUF_FREE_LIST	バッファ空きリストの挿入
IBUF_INDEX	バッファインデックスの挿入
INDEX	B ツリーノード
INODE	インデックスノード
LOB_DATA	非圧縮 LOB データ
LOB_FIRST	非圧縮 LOB の最初のページ
LOB_INDEX	非圧縮 LOB インデックス
PAGE_IO_COMPRESSED	圧縮ページ
PAGE_IO_COMPRESSED_ENCRYPTED	圧縮および暗号化されたページ
PAGE_IO_ENCRYPTED	暗号化ページ
RSEG_ARRAY	ロールバックセグメント配列
RTREE_INDEX	R ツリーインデックス
SDI_BLOB	非圧縮 SDI BLOB
SDI_COMPRESSED_BLOB	圧縮 SDI BLOB
SDI_INDEX	SDI インデックス
SYSTEM	システムページ
TRX_SYSTEM	トランザクションシステムデータ
UNDO_LOG	undo ログページ
UNKNOWN	不明

ページタイプ	説明
ZLOB_DATA	圧縮 LOB データ
ZLOB_FIRST	圧縮 LOB の最初のページ
ZLOB_FRAG	圧縮 LOB フラグメント
ZLOB_FRAG_ENTRY	圧縮 LOB フラグメントインデックス
ZLOB_INDEX	圧縮 LOB インデックス

- **FLUSH_TYPE**
フラッシュタイプ。
- **FIX_COUNT**
バッファプール内でこのブロックを使用するスレッドの数。ゼロのとき、ブロックは削除対象です。
- **IS_HASHED**
ハッシュインデックスがこのページに作成されているかどうか。
- **NEWEST_MODIFICATION**
最も若い変更のログ順序番号。
- **OLDEST_MODIFICATION**
最も古い変更のログ順序番号。
- **ACCESS_TIME**
ページの最初のアクセス時間の判断に使用される無名数。
- **TABLE_NAME**
ページが属するテーブルの名前。このカラムは、**PAGE_TYPE** 値が **INDEX** のページにのみ適用されます。
- **INDEX_NAME**
ページが属するインデックスの名前。これには、クラスタ化されたインデックスまたはセカンダリインデックスの名前を指定できます。このカラムは、**PAGE_TYPE** 値が **INDEX** のページにのみ適用されます。
- **NUMBER_RECORDS**
ページ内のレコード数。
- **DATA_SIZE**
レコードのサイズの合計。このカラムは、**PAGE_TYPE** 値が **INDEX** のページにのみ適用されます。
- **COMPRESSED_SIZE**
圧縮されたページサイズ。圧縮されていないページ用の **NULL**。
- **PAGE_STATE**
ページの状態。次の表は、許可される値を示しています。

表 26.2 INNODB_BUFFER_PAGE.PAGE_STATE の値

ページ状態	説明
FILE_PAGE	バッファファイルページ
MEMORY	メインメモリーオブジェクトを含みます
NOT_USED	空きリスト内

ページ状態	説明
NULL	圧縮されたページ、フラッシュリスト内の圧縮されたページ、バッファプール監視センチネルとして使用されるページを消去
READY_FOR_USE	フリーページ
REMOVE_HASH	空きリストに入れる前にハッシュインデックスを削除する必要があります

- [IO_FIX](#)

このページの I/O が保留中かどうか: [IO_NONE](#) = 保留中の I/O, なし、[IO_READ](#) = 読取り保留中、[IO_WRITE](#) = 書き込み保留中。

- [IS_OLD](#)

LRU リスト内の古いブロックのサブリストにブロックがあるかどうか。

- [FREE_PAGE_CLOCK](#)

ブロックが最後に LRU リストの先頭に置かれたときの [freed_page_clock](#) カウンタの値。 [freed_page_clock](#) カウンタは、LRU リストの末尾から削除されたブロックの数を追跡します。

- [IS_STALE](#)

ページが失効しているかどうか。 MySQL 8.0.24 で追加されました。

例

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE LIMIT 1\G
***** 1. row *****
      POOL_ID: 0
      BLOCK_ID: 0
      SPACE: 97
      PAGE_NUMBER: 2473
      PAGE_TYPE: INDEX
      FLUSH_TYPE: 1
      FIX_COUNT: 0
      IS_HASHED: YES
NEWEST_MODIFICATION: 733855581
OLDEST_MODIFICATION: 0
      ACCESS_TIME: 3378385672
      TABLE_NAME: `employees`.`salaries`
      INDEX_NAME: PRIMARY
      NUMBER_RECORDS: 468
      DATA_SIZE: 14976
      COMPRESSED_SIZE: 0
      PAGE_STATE: FILE_PAGE
      IO_FIX: IO_NONE
      IS_OLD: YES
      FREE_PAGE_CLOCK: 66
      IS_STALE: NO
```

メモ

- このテーブルは、主にエキスパートレベルのパフォーマンス監視、または MySQL のパフォーマンス関連の拡張機能を開発する場合に役立ちます。
- このテーブルをクエリーするには [PROCESS](#) 権限が必要です。
- [INFORMATION_SCHEMA COLUMNS](#) テーブルまたは [SHOW COLUMNS](#) ステートメントを使用して、データ型やデフォルト値など、このテーブルのカラムに関する追加情報を表示します。
- テーブル、テーブルの行、パーティションまたはインデックスが削除されると、他のデータに領域が必要になるまで、関連付けられたページはバッファプールに残ります。 [INNODB_BUFFER_PAGE](#) テーブルでは、バッファプールから削除されるまで、これらのページに関する情報がレポートされます。 [InnoDB](#) がバッファプールデータを管理する方法の詳細は、[セクション15.5.1「バッファプール」](#) を参照してください。

26.51.2 INFORMATION_SCHEMA INNODB_BUFFER_PAGE_LRU テーブル

INNODB_BUFFER_PAGE_LRU テーブルには、InnoDB buffer pool のページに関する情報、特に LRU リストでのページの順序付け方法が示されます。LRU リストでは、ページがいっぱいになったときにバッファプールから `evict` に表示されるページが決定されます。

INNODB_BUFFER_PAGE_LRU テーブルには INNODB_BUFFER_PAGE テーブルと同じカラムがありますが、いくつかの例外があります。これには、`BLOCK_ID` および `PAGE_STATE` カラムのかわりに `LRU_POSITION` および `COMPRESSED` カラムが含まれ、`IS_STALE` カラムは含まれません。

関連する使用法と使用例については、[セクション15.15.5「InnoDB INFORMATION_SCHEMA バッファプールテーブル」](#)を参照してください。

警告

INNODB_BUFFER_PAGE_LRU テーブルをクエリーすると、パフォーマンスに影響する可能性があります。パフォーマンスへの影響を認識し、許容できると判断した場合を除き、本番システムではこのテーブルをクエリーしないでください。本番システムのパフォーマンスへの影響を回避するには、調査する問題を再現し、テストインスタンスのバッファプール統計をクエリーします。

INNODB_BUFFER_PAGE_LRU テーブルには、次のカラムがあります:

- `POOL_ID`

バッファプール ID。これは、複数のバッファプールインスタンスを区別する識別子です。

- `LRU_POSITION`

LRU リストでのページの位置。

- `SPACE`

テーブルスペース ID。INNODB_TABLES.SPACE と同じ値です。

- `PAGE_NUMBER`

ページ番号。

- `PAGE_TYPE`

ページタイプ。次の表は、許可される値を示しています。

表 26.3 INNODB_BUFFER_PAGE_LRU.PAGE_TYPE の値

ページタイプ	説明
<code>ALLOCATED</code>	新規割当済ページ
<code>BLOB</code>	「非圧縮 BLOB」ページ
<code>COMPRESSED_BLOB2</code>	後続のコンポーネント BLOB ページ
<code>COMPRESSED_BLOB</code>	最初の圧縮 BLOB ページ
<code>ENCRYPTED_RTREE</code>	暗号化 R ツリー
<code>EXTENT_DESCRIPTOR</code>	エクステンツ記述子ページ
<code>FILE_SPACE_HEADER</code>	ファイルスペースヘッダー
<code>FIL_PAGE_TYPE_UNUSED</code>	未使用
<code>IBUF_BITMAP</code>	バッファビットマップの挿入
<code>IBUF_FREE_LIST</code>	バッファ空きリストの挿入
<code>IBUF_INDEX</code>	バッファインデックスの挿入

ページタイプ	説明
INDEX	B ツリーノード
INODE	インデックスノード
LOB_DATA	非圧縮 LOB データ
LOB_FIRST	非圧縮 LOB の最初のページ
LOB_INDEX	非圧縮 LOB インデックス
PAGE_IO_COMPRESSED	圧縮ページ
PAGE_IO_COMPRESSED_ENCRYPTED	圧縮および暗号化されたページ
PAGE_IO_ENCRYPTED	暗号化ページ
RSEG_ARRAY	ロールバックセグメント配列
RTREE_INDEX	R ツリーインデックス
SDI_BLOB	非圧縮 SDI BLOB
SDI_COMPRESSED_BLOB	圧縮 SDI BLOB
SDI_INDEX	SDI インデックス
SYSTEM	システムページ
TRX_SYSTEM	トランザクションシステムデータ
UNDO_LOG	undo ログページ
UNKNOWN	不明
ZLOB_DATA	圧縮 LOB データ
ZLOB_FIRST	圧縮 LOB の最初のページ
ZLOB_FRAG	圧縮 LOB フラグメント
ZLOB_FRAG_ENTRY	圧縮 LOB フラグメントインデックス
ZLOB_INDEX	圧縮 LOB インデックス

- **FLUSH_TYPE**

フラッシュタイプ。

- **FIX_COUNT**

バッファプール内でこのブロックを使用するスレッドの数。ゼロのとき、ブロックは削除対象です。

- **IS_HASHED**

ハッシュインデックスがこのページに作成されているかどうか。

- **NEWEST_MODIFICATION**

最も若い変更のログ順序番号。

- **OLDEST_MODIFICATION**

最も古い変更のログ順序番号。

- **ACCESS_TIME**

ページの最初のアクセス時間の判断に使用される無名数。

- **TABLE_NAME**

ページが属するテーブルの名前。このカラムは、**PAGE_TYPE** 値が **INDEX** のページにのみ適用されます。

- **INDEX_NAME**

ページが属するインデックスの名前。これには、クラスタ化されたインデックスまたはセカンダリインデックスの名前を指定できます。このカラムは、`PAGE_TYPE` 値が `INDEX` のページにのみ適用されます。

- `NUMBER_RECORDS`

ページ内のレコード数。

- `DATA_SIZE`

レコードのサイズの合計。このカラムは、`PAGE_TYPE` 値が `INDEX` のページにのみ適用されます。

- `COMPRESSED_SIZE`

圧縮されたページサイズ。圧縮されていないページ用の `NULL`。

- `COMPRESSED`

ページを圧縮するかどうか。

- `IO_FIX`

このページの I/O が保留中かどうか: `IO_NONE` = 保留中の I/O、なし、`IO_READ` = 読取り保留中、`IO_WRITE` = 書き込み保留中。

- `IS_OLD`

LRU リスト内の古いブロックのサブリストにブロックがあるかどうか。

- `FREE_PAGE_CLOCK`

ブロックが最後に LRU リストの先頭に置かれたときの `freed_page_clock` カウンタの値。 `freed_page_clock` カウンタは、LRU リストの末尾から削除されたブロックの数を追跡します。

例

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE_LRU LIMIT 1\G
***** 1. row *****
  POOL_ID: 0
  LRU_POSITION: 0
    SPACE: 97
  PAGE_NUMBER: 1984
  PAGE_TYPE: INDEX
  FLUSH_TYPE: 1
  FIX_COUNT: 0
  IS_HASHED: YES
NEWEST_MODIFICATION: 719490396
OLDEST_MODIFICATION: 0
  ACCESS_TIME: 3378383796
  TABLE_NAME: `employees`.`salaries`
  INDEX_NAME: PRIMARY
NUMBER_RECORDS: 468
  DATA_SIZE: 14976
COMPRESSED_SIZE: 0
  COMPRESSED: NO
    IO_FIX: IO_NONE
    IS_OLD: YES
  FREE_PAGE_CLOCK: 0
```

メモ

- このテーブルは、主にエキスパートレベルのパフォーマンス監視、または MySQL のパフォーマンス関連の拡張機能を開発する場合に役立ちます。
- このテーブルをクエリーするには `PROCESS` 権限が必要です。
- `INFORMATION_SCHEMA COLUMNS` テーブルまたは `SHOW COLUMNS` ステートメントを使用して、データ型やデフォルト値など、このテーブルのカラムに関する追加情報を表示します。

- このテーブルをクエリーするには、MySQL で、バッファプール内のアクティブページ数の 64 バイト倍を超える連続メモリの大きなブロックを割り当てる必要があります。この割り当ては、特に数 G バイトのバッファプールを持つシステムで、メモリ不足エラーを引き起こす可能性があります。
- このテーブルをクエリーした場合、MySQLで、LRU リストのトラバース中、バッファプールを表すデータ構造をロックする必要があります。これにより、特に数 G バイトのバッファプールを持つシステムで並列性を軽減できます。
- テーブル、テーブルの行、パーティションまたはインデックスが削除されると、他のデータに領域が必要になるまで、関連付けられたページはバッファプールに残ります。INNODB_BUFFER_PAGE_LRU テーブルでは、バッファプールから削除されるまで、これらのページに関する情報がレポートされます。InnoDB がバッファプールデータを管理する方法の詳細は、[セクション15.5.1「バッファプール」](#)を参照してください。

26.51.3 INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS テーブル

INNODB_BUFFER_POOL_STATS テーブルは、[SHOW ENGINE INNODB STATUS](#) 出力で提供されるものと同じバッファプール情報の多くを提供します。同じ情報の多くは、InnoDB バッファプールの[サーバーステータス変数](#)を使用して取得することもできます。

バッファプール内のページに「新しい」または「新しくない」というマークを付けるという考えは、バッファプールデータ構造の先頭と末尾にある[サブリスト](#)間にそれらを転送することを意味します。「新しい」とされたページは、バッファプールから削除されるまでに長い時間がかかりますが、「新しくない」とされたページは、[エビクション](#)の時点に近づけられます。

関連する使用法と使用例については、[セクション15.15.5「InnoDB INFORMATION_SCHEMA バッファプールテーブル」](#)を参照してください。

INNODB_BUFFER_POOL_STATS テーブルには、次のカラムがあります：

- **POOL_ID**
バッファプール ID。これは、複数のバッファプールインスタンスを区別する識別子です。
- **POOL_SIZE**
InnoDB バッファプールのサイズ (ページ単位)。
- **FREE_BUFFERS**
InnoDB バッファプール内の空きページの数
- **DATABASE_PAGES**
データを含む InnoDB バッファプール内のページの数。この数にはダーティページとクリーンページの両方が含まれます。
- **OLD_DATABASE_PAGES**
old バッファプールサブリスト内のページの数。
- **MODIFIED_DATABASE_PAGES**
変更された (ダーティ) データベースページの数
- **PENDING_DECOMPRESS**
圧縮解除保留中のページの数
- **PENDING_READS**
読み取り保留中の数
- **PENDING_FLUSH_LRU**

LRU 内のフラッシュ保留中のページの数

- [PENDING_FLUSH_LIST](#)
フラッシュリスト内のフラッシュ保留中のページの数
- [PAGES_MADE_YOUNG](#)
新しいとされたページの数
- [PAGES_NOT_MADE_YOUNG](#)
新しいとされていないページの数
- [PAGES_MADE_YOUNG_RATE](#)
秒あたりに新しいとされたページの数 (最後の出力/経過時間以降に新しいとされたページ)
- [PAGES_MADE_NOT_YOUNG_RATE](#)
秒あたりに新しいとされなかったページの数 (最後の出力/経過時間以降に新しいとされなかったページ)
- [NUMBER_PAGES_READ](#)
読み取られたページの数。
- [NUMBER_PAGES_CREATED](#)
作成されたページの数。
- [NUMBER_PAGES_WRITTEN](#)
書き込まれたページの数
- [PAGES_READ_RATE](#)
秒あたりに読み取られたページの数 (最後の出力/経過時間以降に読み取られたページ)
- [PAGES_CREATE_RATE](#)
秒あたりに作成されたページの数 (最後の出力/経過時間以降に作成されたページ)
- [PAGES_WRITTEN_RATE](#)
秒あたりに書き込まれたページの数 (最後の出力/経過時間以降に書き込まれたページ)
- [NUMBER_PAGES_GET](#)
論理読み取りリクエスト数。
- [HIT_RATE](#)
バッファープールのヒット率
- [YOUNG_MAKE_PER_THOUSAND_GETS](#)
1000 の取得あたりに新しいとされたページの数
- [NOT_YOUNG_MAKE_PER_THOUSAND_GETS](#)
1000 の取得あたりに新しいとされていないページの数
- [NUMBER_PAGES_READ_AHEAD](#)
先読みされたページの数

- **NUMBER_READ_AHEAD_EVICTED**

先読みバックグラウンドスレッドによって InnoDB バッファプールに読み取られ、クエリーでアクセスされることなくその後に削除されたページの数。

- **READ_AHEAD_RATE**

先読み速度 / 秒 (最後の印刷以降に先読みされたページ / 経過時間)。

- **READ_AHEAD_EVICTED_RATE**

アクセスなしで削除された先読みページの数 / 秒 (最後の印刷 / 経過時間以降アクセスされなかった先読みページ)。

- **LRU_IO_TOTAL**

LRU I/O の合計。

- **LRU_IO_CURRENT**

現在の間隔の LRU I/O。

- **UNCOMPRESS_TOTAL**

解凍されたページの合計数。

- **UNCOMPRESS_CURRENT**

現在の間隔で圧縮解除されたページの数

例

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_BUFFER_POOL_STATS
***** 1. row *****
      POOL_ID: 0
      POOL_SIZE: 8192
      FREE_BUFFERS: 1
      DATABASE_PAGES: 8085
      OLD_DATABASE_PAGES: 2964
      MODIFIED_DATABASE_PAGES: 0
      PENDING_DECOMPRESS: 0
      PENDING_READS: 0
      PENDING_FLUSH_LRU: 0
      PENDING_FLUSH_LIST: 0
      PAGES_MADE_YOUNG: 22821
      PAGES_NOT_MADE_YOUNG: 3544303
      PAGES_MADE_YOUNG_RATE: 357.62602199870594
      PAGES_MADE_NOT_YOUNG_RATE: 0
      NUMBER_PAGES_READ: 2389
      NUMBER_PAGES_CREATED: 12385
      NUMBER_PAGES_WRITTEN: 13111
      PAGES_READ_RATE: 0
      PAGES_CREATE_RATE: 0
      PAGES_WRITTEN_RATE: 0
      NUMBER_PAGES_GET: 33322210
      HIT_RATE: 1000
      YOUNG_MAKE_PER_THOUSAND_GETS: 18
      NOT_YOUNG_MAKE_PER_THOUSAND_GETS: 0
      NUMBER_PAGES_READ_AHEAD: 2024
      NUMBER_READ_AHEAD_EVICTED: 0
      READ_AHEAD_RATE: 0
      READ_AHEAD_EVICTED_RATE: 0
      LRU_IO_TOTAL: 0
      LRU_IO_CURRENT: 0
      UNCOMPRESS_TOTAL: 0
      UNCOMPRESS_CURRENT: 0
```

メモ

- このテーブルは、主にエキスパートレベルのパフォーマンス監視、または MySQL のパフォーマンス関連の拡張機能を開発する場合に役立ちます。

- このテーブルをクエリーするには **PROCESS** 権限が必要です。
- **INFORMATION_SCHEMA COLUMNS** テーブルまたは **SHOW COLUMNS** ステートメントを使用して、データ型やデフォルト値など、このテーブルのカラムに関する追加情報を表示します。

26.51.4 INFORMATION_SCHEMA INNODB_CACHED_INDEXES テーブル

INNODB_CACHED_INDEXES テーブルには、インデックスごとに **InnoDB** バッファプールにキャッシュされたインデックスページの数レポートされます。

関連する使用法と使用例については、[セクション15.15.5「InnoDB INFORMATION_SCHEMA バッファプールテーブル」](#)を参照してください。

INNODB_CACHED_INDEXES テーブルには、次のカラムがあります:

- **SPACE_ID**
テーブルスペース ID。
- **INDEX_ID**
インデックスの識別子。インデックス識別子は、インスタンス内のすべてのデータベースで一意です。
- **N_CACHED_PAGES**
InnoDB バッファプールにキャッシュされたインデックスページの数。

例

このクエリーは、特定のインデックスに対して **InnoDB** バッファプールにキャッシュされたインデックスページの数返します:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_CACHED_INDEXES WHERE INDEX_ID=65\G
***** 1. row *****
SPACE_ID: 4294967294
INDEX_ID: 65
N_CACHED_PAGES: 45
```

このクエリーは、**INNODB_INDEXES** テーブルおよび **INNODB_TABLES** テーブルを使用して各 **INDEX_ID** 値のテーブル名およびインデックス名を解決し、各インデックスの **InnoDB** バッファプールにキャッシュされたインデックスページの数返します。

```
SELECT
  tables.NAME AS table_name,
  indexes.NAME AS index_name,
  cached.N_CACHED_PAGES AS n_cached_pages
FROM
  INFORMATION_SCHEMA.INNODB_CACHED_INDEXES AS cached,
  INFORMATION_SCHEMA.INNODB_INDEXES AS indexes,
  INFORMATION_SCHEMA.INNODB_TABLES AS tables
WHERE
  cached.INDEX_ID = indexes.INDEX_ID
  AND indexes.TABLE_ID = tables.TABLE_ID;
```

メモ

- このテーブルをクエリーするには **PROCESS** 権限が必要です。
- **INFORMATION_SCHEMA COLUMNS** テーブルまたは **SHOW COLUMNS** ステートメントを使用して、データ型やデフォルト値など、このテーブルのカラムに関する追加情報を表示します。

26.51.5 INFORMATION_SCHEMA INNODB_CMP および INNODB_CMP_RESET テーブル

INNODB_CMP テーブルおよび INFORMATION_SCHEMA.INNODB_CMP_RESET テーブルは、compressed InnoDB テーブルに関連する操作のステータス情報を提供します。

INNODB_CMP テーブルと INFORMATION_SCHEMA.INNODB_CMP_RESET テーブルには、次のカラムがあります:

- **PAGE_SIZE**

圧縮されたページサイズ (バイト)。

- **COMPRESS_OPS**

サイズ **PAGE_SIZE** の B ツリーページが圧縮された回数。空のページが作成されたり、非圧縮変更ログ用の領域が不足したりするたびに、ページが圧縮されます。

- **COMPRESS_OPS_OK**

サイズ **PAGE_SIZE** の B ツリーページが正常に圧縮された回数。このカウントが **COMPRESS_OPS** を超えることは決してありません。

- **COMPRESS_TIME**

サイズが **PAGE_SIZE** の B ツリーページの圧縮試行に使用される合計時間 (秒)。

- **UNCOMPRESS_OPS**

サイズ **PAGE_SIZE** の B ツリーページが圧縮解除された回数。圧縮が失敗したときや、圧縮解除されたページがバッファプールに存在しない場合に最初にアクセスするときに、B ツリーページは圧縮解除されます。

- **UNCOMPRESS_TIME**

PAGE_SIZE サイズの B ツリーページの圧縮解除に使用される合計時間 (秒)。

例

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_CMP\G
***** 1. row *****
  page_size: 1024
 compress_ops: 0
compress_ops_ok: 0
  compress_time: 0
uncompress_ops: 0
uncompress_time: 0
***** 2. row *****
  page_size: 2048
 compress_ops: 0
compress_ops_ok: 0
  compress_time: 0
uncompress_ops: 0
uncompress_time: 0
***** 3. row *****
  page_size: 4096
 compress_ops: 0
compress_ops_ok: 0
  compress_time: 0
uncompress_ops: 0
uncompress_time: 0
***** 4. row *****
  page_size: 8192
 compress_ops: 86955
compress_ops_ok: 81182
  compress_time: 27
uncompress_ops: 26828
uncompress_time: 5
***** 5. row *****
  page_size: 16384
 compress_ops: 0
compress_ops_ok: 0
  compress_time: 0
```

```
uncompress_ops: 0
uncompress_time: 0
```

メモ

- これらのテーブルを使用して、データベース内の InnoDB テーブルの圧縮の有効性を測定します。
- このテーブルをクエリーするには PROCESS 権限が必要です。
- INFORMATION_SCHEMA COLUMNS テーブルまたは SHOW COLUMNS ステートメントを使用して、データ型やデフォルト値など、このテーブルのカラムに関する追加情報を表示します。
- 用法については、セクション 15.9.1.4 「実行時の InnoDB テーブル圧縮の監視」およびセクション 15.15.1.3 「圧縮情報スキーマテーブルの使用」を参照してください。InnoDB テーブルの圧縮に関する一般情報については、セクション 15.9 「InnoDB のテーブルおよびページの圧縮」を参照してください。

26.51.6 INFORMATION_SCHEMA INNODB_CMPMEM および INNODB_CMPMEM_RESET テーブル

INNODB_CMPMEM テーブルおよび INNODB_CMPMEM_RESET テーブルは、InnoDB buffer pool 内の圧縮された pages のステータス情報を提供します。

INNODB_CMPMEM テーブルと INNODB_CMPMEM_RESET テーブルには、次のカラムがあります:

- PAGE_SIZE

バイト単位のブロックサイズ。このテーブルの各レコードは、このサイズのブロックを記述します。

- BUFFER_POOL_INSTANCE

バッファプールインスタンスの一意の識別子。

- PAGES_USED

現在使用中のサイズ PAGE_SIZE のブロック数。

- PAGES_FREE

現在割当て可能なサイズ PAGE_SIZE のブロック数。このカラムは、メモリープールの外部断片化を示します。これらの数値は最大 1 にしてください。

- RELOCATION_OPS

サイズが PAGE_SIZE のブロックが再配置された回数。バディシステムは、より大きな解放されたブロックを生成しようとするときに、解放されたブロックの割り当て済みの「バディー」を再配置できます。INNODB_CMPMEM_RESET テーブルから読み取ると、このカウントはリセットされます。

- RELOCATION_TIME

サイズが PAGE_SIZE のブロックの再配置に使用される合計時間 (マイクロ秒)。テーブル INNODB_CMPMEM_RESET から読み取ると、このカウントはリセットされます。

例

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_CMPMEM\G
***** 1. row *****
    page_size: 1024
buffer_pool_instance: 0
    pages_used: 0
    pages_free: 0
    relocation_ops: 0
    relocation_time: 0
***** 2. row *****
    page_size: 2048
buffer_pool_instance: 0
```

```

pages_used: 0
pages_free: 0
relocation_ops: 0
relocation_time: 0
***** 3. row *****
page_size: 4096
buffer_pool_instance: 0
pages_used: 0
pages_free: 0
relocation_ops: 0
relocation_time: 0
***** 4. row *****
page_size: 8192
buffer_pool_instance: 0
pages_used: 7673
pages_free: 15
relocation_ops: 4638
relocation_time: 0
***** 5. row *****
page_size: 16384
buffer_pool_instance: 0
pages_used: 0
pages_free: 0
relocation_ops: 0
relocation_time: 0

```

メモ

- これらのテーブルを使用して、データベース内の InnoDB テーブルの **圧縮** の有効性を測定します。
- このテーブルをクエリーするには **PROCESS** 権限が必要です。
- **INFORMATION_SCHEMA COLUMNS** テーブルまたは **SHOW COLUMNS** ステートメントを使用して、データ型やデフォルト値など、このテーブルのカラムに関する追加情報を表示します。
- 使用法については、[セクション15.9.1.4「実行時の InnoDB テーブル圧縮の監視」](#) および [セクション15.15.1.3「圧縮情報スキーマテーブルの使用」](#) を参照してください。InnoDB テーブルの圧縮に関する一般情報については、[セクション15.9「InnoDB のテーブルおよびページの圧縮」](#) を参照してください。

26.51.7 INFORMATION_SCHEMA INNODB_CMP_PER_INDEX および INNODB_CMP_PER_INDEX_RESET テーブル

INNODB_CMP_PER_INDEX テーブルおよび **INNODB_CMP_PER_INDEX_RESET** テーブルでは、**compressed** InnoDB のテーブルおよびインデックスに関連する操作のステータス情報が提供され、データベース、テーブルおよびインデックスの組合せごとに個別の統計が提供されるため、特定のテーブルの圧縮のパフォーマンスおよび有用性を評価できます。

圧縮 InnoDB テーブルについては、テーブルデータとすべての**セカンダリインデックス**の両方が圧縮されます。このコンテキストでは、テーブルデータは、単なる別のインデックス、たまたますべてのカラムが含まれているインデックス (**クラスタ化されたインデックス**) として扱われます。

INNODB_CMP_PER_INDEX テーブルと **INNODB_CMP_PER_INDEX_RESET** テーブルには、次のカラムがあります:

- **DATABASE_NAME**
適用可能なテーブルを含むスキーマ (データベース)。
- **TABLE_NAME**
圧縮統計を監視するテーブル。
- **INDEX_NAME**
圧縮統計を監視するインデックス。
- **COMPRESS_OPS**

試行された圧縮操作の数。空のページが作成されたり、非圧縮変更ログ用の領域が不足したりするたびに、ページが圧縮されます。

- [COMPRESS_OPS_OK](#)

成功した圧縮操作の数。 [COMPRESS_OPS](#) 値から引き算すると、[圧縮の失敗](#)の回数が求められます。 [COMPRESS_OPS](#) 値で割り算すると、圧縮の失敗の割合が求められます。

- [COMPRESS_TIME](#)

このインデックスのデータの圧縮に使用される合計時間 (秒)。

- [UNCOMPRESS_OPS](#)

実行された圧縮解除操作の数。圧縮 InnoDB ページは、圧縮が失敗した場合はいつでも圧縮解除されます。または、[バッファプール](#)においてはじめて圧縮ページへのアクセスがあり、圧縮解除されたページが存在しないときに圧縮解除されます。

- [UNCOMPRESS_TIME](#)

このインデックスのデータを圧縮解除するために使用される合計時間 (秒)。

例

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_CMP_PER_INDEX
***** 1. row *****
database_name: employees
table_name: salaries
index_name: PRIMARY
compress_ops: 0
compress_ops_ok: 0
compress_time: 0
uncompress_ops: 23451
uncompress_time: 4
***** 2. row *****
database_name: employees
table_name: salaries
index_name: emp_no
compress_ops: 0
compress_ops_ok: 0
compress_time: 0
uncompress_ops: 1597
uncompress_time: 0
```

メモ

- これらのテーブルを使用して、特定のテーブルまたはインデックス、あるいはその両方について InnoDB テーブル圧縮の有効性を測定します。
- これらのテーブルをクエリーするには [PROCESS](#) 権限が必要です。
- [INFORMATION_SCHEMA COLUMNS](#) テーブルまたは [SHOW COLUMNS](#) ステートメントを使用して、データ型やデフォルト値など、これらのテーブルのカラムに関する追加情報を表示します。
- すべてのインデックスで個別に測定値を収集すると、大幅なパフォーマンスオーバーヘッドが発生するので、デフォルトでは [INNODB_CMP_PER_INDEX](#) および [INNODB_CMP_PER_INDEX_RESET](#) 統計は収集されません。監視する圧縮テーブルに対して操作を実行する前に、[innodb_cmp_per_index_enabled](#) システム変数を有効にする必要があります。
- 用法については、[セクション15.9.1.4「実行時の InnoDB テーブル圧縮の監視」](#) および [セクション15.15.1.3「圧縮情報スキーマテーブルの使用」](#) を参照してください。InnoDB テーブルの圧縮に関する一般情報については、[セクション15.9「InnoDB のテーブルおよびページの圧縮」](#) を参照してください。

26.51.8 INFORMATION_SCHEMA INNODB_COLUMNS テーブル

[INNODB_COLUMNS](#) テーブルは、InnoDB テーブルのカラムに関するメタデータを提供します。

関連する使用法と使用例については、[セクション15.15.3「InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル」](#)を参照してください。

INNODB_COLUMNS テーブルには、次のカラムがあります:

- **TABLE_ID**

カラムに関連付けられたテーブルを表す識別子で、`INNODB_TABLES.TABLE_ID` と同じ値です。

- **NAME**

カラムの名前。これらの名前の大文字/小文字は、`lower_case_table_names` 設定に応じて異なります。カラムの特別なシステム予約名はありません。

- **POS**

0 から始まり連続的に増加する、テーブル内のカラムの順序位置。あるカラムを削除すると、順序に欠落ができないように残りのカラムの順序が変更されます。仮想生成カラムの `POS` 値は、カラムの順序番号および順序位置をエンコードします。詳細は、[セクション26.51.30「INFORMATION_SCHEMA.INNODB_VIRTUAL テーブル」](#)の `POS` カラムの説明を参照してください。

- **MTYPE**

「メインの型」を表します。カラムタイプの数値識別子。1 = `VARCHAR`、2 = `CHAR`、3 = `FIXBINARY`、4 = `BINARY`、5 = `BLOB`、6 = `INT`、7 = `SYS_CHILD`、8 = `SYS`、9 = `FLOAT`、10 = `DOUBLE`、11 = `DECIMAL`、12 = `VARMYSQL`、13 = `MYSQL`、14 = `GEOMETRY`。

- **PRTYPE**

InnoDB の「正確な型」。MySQL データ型、文字セット、および NULL 可能性を表すビットを含むバイナリ値です。

- **LEN**

カラム長。たとえば `INT` には 4、`BIGINT` には 8 です。マルチバイト文字セットの文字カラムの場合、この長さ値は、`VARCHAR(N)` などの定義を表すために必要なバイト単位の最大長です。つまり、文字エンコーディングに応じて、`2*N`、`3*N` などになります。

- **HAS_DEFAULT**

`ALGORITHM=INSTANT` で `ALTER TABLE ... ADD COLUMN` を使用して即時に追加されたカラムにデフォルト値があるかどうかを示すブール値。即時に追加されたすべてのカラムにはデフォルト値があり、このカラムが即時に追加されたかどうかを示すインジケータになります。

- **DEFAULT_VALUE**

`ALGORITHM=INSTANT` で `ALTER TABLE ... ADD COLUMN` を使用して即時に追加されたカラムの初期デフォルト値。デフォルト値が `NULL` であるか、指定されていない場合、このカラムは `NULL` をレポートします。明示的に指定された `NULL` 以外のデフォルト値は、内部バイナリ形式で表示されます。以降にカラムのデフォルト値を変更しても、このカラムでレポートされる値は変更されません。

例

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_COLUMNS where TABLE_ID = 71\G
***** 1. row *****
TABLE_ID: 71
NAME: col1
POS: 0
MTYPE: 6
PRTYPE: 1027
LEN: 4
HAS_DEFAULT: 0
DEFAULT_VALUE: NULL
***** 2. row *****
TABLE_ID: 71
NAME: col2
```

```

POS: 1
MTYPE: 2
PRTYPE: 524542
LEN: 10
HAS_DEFAULT: 0
DEFAULT_VALUE: NULL
***** 3. row *****
TABLE_ID: 71
NAME: col3
POS: 2
MTYPE: 1
PRTYPE: 524303
LEN: 10
HAS_DEFAULT: 0
DEFAULT_VALUE: NULL

```

メモ

- このテーブルをクエリーするには [PROCESS](#) 権限が必要です。
- [INFORMATION_SCHEMA COLUMNS](#) テーブルまたは [SHOW COLUMNS](#) ステートメントを使用して、データ型やデフォルト値など、このテーブルのカラムに関する追加情報を表示します。

26.51.9 INFORMATION_SCHEMA INNODB_DATAFILES テーブル

[INNODB_DATAFILES](#) テーブルには、[InnoDB file-per-table](#) および一般テーブルスペースのデータファイルパス情報が表示されます。

関連する使用法と使用例については、[セクション15.15.3「InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル」](#)を参照してください。

注記

[INFORMATION_SCHEMA FILES](#) テーブルには、[file-per-table](#) テーブルスペース、一般テーブルスペース、システムテーブルスペース、グローバル一時テーブルスペースおよび undo テーブルスペースを含む [InnoDB](#) テーブルスペースタイプのメタデータがレポートされません。

[INNODB_DATAFILES](#) テーブルには、次のカラムがあります:

- [SPACE](#)

テーブルスペース ID。

- [PATH](#)

テーブルスペースデータファイルのパス。[file-per-table](#) テーブルスペースが MySQL データディレクトリ外の場所に作成される場合、パス値は完全修飾ディレクトリパスです。それ以外の場合、パスはデータディレクトリに対する相対パスになります。

例

```

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_DATAFILES WHERE SPACE = 57G
***** 1. row *****
SPACE: 57
PATH: ./test/t1.ibd

```

メモ

- このテーブルをクエリーするには [PROCESS](#) 権限が必要です。
- [INFORMATION_SCHEMA COLUMNS](#) テーブルまたは [SHOW COLUMNS](#) ステートメントを使用して、データ型やデフォルト値など、このテーブルのカラムに関する追加情報を表示します。

26.51.10 INFORMATION_SCHEMA INNODB_FIELDS テーブル

INNODB_FIELDS テーブルは、InnoDB インデックスのキーカラム (フィールド) に関するメタデータを提供します。

関連する使用法と使用例については、[セクション15.15.3「InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル」](#)を参照してください。

INNODB_FIELDS テーブルには、次のカラムがあります:

- INDEX_ID

このキーフィールドに関連付けられたインデックスの識別子 (INNODB_INDEXES.INDEX_ID と同じ値)。

- NAME

テーブルの元のカラムの名前。INNODB_COLUMNS.NAME と同じ値です。

- POS

0 から始まり連続的に増加する、インデックス内のキーフィールドの順序位置。あるカラムを削除すると、順序に欠落ができないように残りのカラムの順序が変更されます。

例

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FIELDS WHERE INDEX_ID = 117\G
***** 1. row *****
INDEX_ID: 117
NAME: col1
POS: 0
```

メモ

- このテーブルをクエリーするには [PROCESS](#) 権限が必要です。
- [INFORMATION_SCHEMA COLUMNS](#) テーブルまたは [SHOW COLUMNS](#) ステートメントを使用して、データ型やデフォルト値など、このテーブルのカラムに関する追加情報を表示します。

26.51.11 INFORMATION_SCHEMA INNODB_FOREIGN テーブル

INNODB_FOREIGN テーブルは、[InnoDB foreign keys](#) に関するメタデータを提供します。

関連する使用法と使用例については、[セクション15.15.3「InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル」](#)を参照してください。

INNODB_FOREIGN テーブルには、次のカラムがあります:

- ID

外部キーインデックスの名前 (数値ではない)。先頭にスキーマ (データベース) 名 (`test/products_fk` など)。

- FOR_NAME

この外部キー関係の[子テーブル](#)の名前。

- REF_NAME

この外部キー関係の[親テーブル](#)の名前。

- N_COLS

外部キーインデックスのカラム数。

- TYPE

外部キーカラムに関する情報を含むビットフラグの集合。0 = [ON DELETE/UPDATE RESTRICT](#)、1 = [ON DELETE CASCADE](#)、2 = [ON DELETE SET NULL](#)、4 = [ON UPDATE CASCADE](#)、8 = [ON UPDATE SET NULL](#)、16 = [ON DELETE NO ACTION](#)、32 = [ON UPDATE NO ACTION](#)。

例

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FOREIGNIG
***** 1. row *****
ID: test/fk1
FOR_NAME: test/child
REF_NAME: test/parent
N_COLS: 1
TYPE: 1
```

メモ

- このテーブルをクエリーするには `PROCESS` 権限が必要です。
- `INFORMATION_SCHEMA COLUMNS` テーブルまたは `SHOW COLUMNS` ステートメントを使用して、データ型やデフォルト値など、このテーブルのカラムに関する追加情報を表示します。

26.51.12 INFORMATION_SCHEMA INNODB_FOREIGN_COLS テーブル

`INNODB_FOREIGN_COLS` テーブルには、`InnoDB` 外部キーカラムに関するステータス情報が表示されます。

関連する使用法と使用例については、[セクション15.15.3「InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル」](#)を参照してください。

`INNODB_FOREIGN_COLS` テーブルには、次のカラムがあります:

- `ID`
このインデックスキーフィールドに関連付けられた外部キーインデックス (`INNODB_FOREIGN.ID` と同じ値)。
- `FOR_COL_NAME`
子テーブルに関連付けられたカラムの名前。
- `REF_COL_NAME`
親テーブルに関連付けられたカラムの名前。
- `POS`
0 から始まる、外部キーインデックス内のこのキーフィールドの順序位置。

例

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FOREIGN_COLS WHERE ID = 'test/fk1'G
***** 1. row *****
ID: test/fk1
FOR_COL_NAME: parent_id
REF_COL_NAME: id
POS: 0
```

メモ

- このテーブルをクエリーするには `PROCESS` 権限が必要です。
- `INFORMATION_SCHEMA COLUMNS` テーブルまたは `SHOW COLUMNS` ステートメントを使用して、データ型やデフォルト値など、このテーブルのカラムに関する追加情報を表示します。

26.51.13 INFORMATION_SCHEMA INNODB_FT_BEING_DELETED テーブル

`INNODB_FT_BEING_DELETED` テーブルは、`INNODB_FT_DELETED` テーブルのスナップショットを提供します。これは、`OPTIMIZE TABLE` のメンテナンス操作中にのみ使用されます。`OPTIMIZE TABLE` を実行すると、`INNODB_FT_BEING_DELETED` テーブルが空になり、`INNODB_FT_DELETED` テーブルから `DOC_ID` 値が削除

されます。 `INNODB_FT_BEING_DELETED` の内容は一般に有効期間が短いため、モニタリングやデバッグでのこのテーブルの有用性は限られます。 `FULLTEXT` インデックスを持つテーブルでの `OPTIMIZE TABLE` の実行の詳細は、[セクション 12.10.6 「MySQL の全文検索の微調整」](#) を参照してください。

このテーブルは最初は空です。クエリーする前に、`innodb_ft_aux_table` システム変数の値を、`FULLTEXT` インデックスを含むテーブルの名前 (`test/articles` など) に設定します。出力は、`INNODB_FT_DELETED` テーブルに示されている例のようになります。

関連する使用方法と使用例については、[セクション 15.15.4 「InnoDB INFORMATION_SCHEMA FULLTEXT インデックステーブル」](#) を参照してください。

`INNODB_FT_BEING_DELETED` テーブルには、次のカラムがあります:

- `DOC_ID`

削除されている過程にある行のドキュメント ID。この値は、基礎となるテーブルに定義した ID カラムの値を反映しているか、テーブルに適切なカラムが含まれていない場合に InnoDB によって生成される順序値である可能性があります。この値は、テキスト検索の実行時に、削除された行のデータが `OPTIMIZE TABLE` ステートメントによって `FULLTEXT` インデックスから物理的に削除される前に、`INNODB_FT_INDEX_TABLE` テーブルの行をスキップするために使用されます。詳細は、[InnoDB 全文インデックスの最適化](#) を参照してください。

メモ

- `INFORMATION_SCHEMA COLUMNS` テーブルまたは `SHOW COLUMNS` ステートメントを使用して、データ型やデフォルト値など、このテーブルのカラムに関する追加情報を表示します。
- このテーブルをクエリーするには `PROCESS` 権限が必要です。
- InnoDB `FULLTEXT` 検索の詳細は、[セクション 15.6.2.4 「InnoDB FULLTEXT インデックス」](#) および [セクション 12.10 「全文検索関数」](#) を参照してください。

26.51.14 INFORMATION_SCHEMA INNODB_FT_CONFIG テーブル

`INNODB_FT_CONFIG` テーブルは、InnoDB テーブルの `FULLTEXT` インデックスおよび関連する処理に関するメタデータを提供します。

このテーブルは最初は空です。クエリーする前に、`innodb_ft_aux_table` システム変数の値を、`FULLTEXT` インデックスを含むテーブルの名前 (`test/articles` など) に設定します。

関連する使用方法と使用例については、[セクション 15.15.4 「InnoDB INFORMATION_SCHEMA FULLTEXT インデックステーブル」](#) を参照してください。

`INNODB_FT_CONFIG` テーブルには、次のカラムがあります:

- `KEY`

`FULLTEXT` インデックスを含んだ InnoDB テーブルのメタデータの項目を指定する名前。

このカラムの値は、InnoDB 全文処理のパフォーマンスチューニングおよびデバッグのニーズに応じて変わる可能性があります。キー名とその意味は次のとおりです:

- `optimize_checkpoint_limit`: `OPTIMIZE TABLE` の実行が停止するまでの秒数。
- `synced_doc_id`: 次に発行される `DOC_ID` です。
- `stopword_table_name`: ユーザー定義のストップワードテーブルの `database/table` 名。ユーザー定義のストップワードテーブルがない場合、`VALUE` カラムは空です。
- `use_stopword`: `FULLTEXT` インデックスの作成時に定義されるストップワードテーブルを使用するかどうかを示します。

- `VALUE`

InnoDB テーブルの FULLTEXT インデックスの側面に対する一部の制限または現在値を反映した、KEY カラムに関連付けられた値。

例

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_CONFIG;
+-----+-----+
| KEY          | VALUE          |
+-----+-----+
| optimize_checkpoint_limit | 180          |
| synced_doc_id          | 0            |
| stopword_table_name    | test/my_stopwords |
| use_stopword          | 1            |
+-----+-----+
```

メモ

- このテーブルは、内部構成のみを対象としています。統計情報の使用は想定されていません。
- このテーブルをクエリーするには `PROCESS` 権限が必要です。
- `INFORMATION_SCHEMA.COLUMNS` テーブルまたは `SHOW COLUMNS` ステートメントを使用して、データ型やデフォルト値など、このテーブルのカラムに関する追加情報を表示します。
- InnoDB FULLTEXT 検索の詳細は、[セクション15.6.2.4「InnoDB FULLTEXT インデックス」](#) および [セクション12.10「全文検索関数」](#) を参照してください。

26.51.15 INFORMATION_SCHEMA.INNODB_FT_DEFAULT_STOPWORD テーブル

`INNODB_FT_DEFAULT_STOPWORD` テーブルには、InnoDB テーブルに FULLTEXT インデックスを作成するときにデフォルトで使用される `stopwords` のリストが格納されます。デフォルトの InnoDB ストップワードリストに関する情報と、自身のストップワードリストの定義方法については、[セクション12.10.4「全文ストップワード」](#) を参照してください。

関連する使用法と使用例については、[セクション15.15.4「InnoDB INFORMATION_SCHEMA FULLTEXT インデックステーブル」](#) を参照してください。

`INNODB_FT_DEFAULT_STOPWORD` テーブルには、次のカラムがあります：

- `value`

InnoDB テーブルで FULLTEXT インデックスのストップワードとしてデフォルトで使用される単語。これは、デフォルトのストップワード処理を `innodb_ft_server_stopword_table` または `innodb_ft_user_stopword_table` システム変数でオーバーライドする場合には使用されません。

例

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_DEFAULT_STOPWORD;
+-----+
| value |
+-----+
| a     |
| about |
| an    |
| are   |
| as    |
| at    |
| be    |
| by    |
| com   |
| de    |
| en    |
| for   |
| from  |
| how   |
```

```

i |
in |
is |
it |
la |
of |
on |
or |
that |
the |
this |
to |
was |
what |
when |
where |
who |
will |
with |
und |
the |
www |
+-----+
36 rows in set (0.00 sec)

```

メモ

- このテーブルをクエリーするには [PROCESS](#) 権限が必要です。
- [INFORMATION_SCHEMA COLUMNS](#) テーブルまたは [SHOW COLUMNS](#) ステートメントを使用して、データ型やデフォルト値など、このテーブルのカラムに関する追加情報を表示します。
- InnoDB FULLTEXT 検索の詳細は、[セクション15.6.2.4「InnoDB FULLTEXT インデックス」](#) および [セクション12.10「全文検索関数」](#) を参照してください。

26.51.16 INFORMATION_SCHEMA INNODB_FT_DELETED テーブル

INNODB_FT_DELETED テーブルには、InnoDB テーブルの FULLTEXT インデックスから削除された行が格納されます。InnoDB FULLTEXT インデックスに対する DML 操作中の高コストのインデックス再編成を回避するために、新しく削除された単語に関する情報は個別に格納され、テキスト検索の実行時に検索結果から除外され、InnoDB テーブルに対して [OPTIMIZE TABLE](#) ステートメントを発行した場合にのみメイン検索インデックスから削除されます。詳細は、[InnoDB 全文インデックスの最適化](#) を参照してください。

このテーブルは最初は空です。クエリーする前に、`innodb_ft_aux_table` システム変数の値を、FULLTEXT インデックスを含むテーブルの名前 (`test/articles` など) に設定します。

関連する使用方法と使用例については、[セクション15.15.4「InnoDB INFORMATION_SCHEMA FULLTEXT インデックステーブル」](#) を参照してください。

INNODB_FT_DELETED テーブルには、次のカラムがあります：

- [DOC_ID](#)

新たに削除された行のドキュメント ID。この値は、基礎となるテーブルに定義した ID カラムの値を反映しているか、テーブルに適切なカラムが含まれていない場合に InnoDB によって生成される順序値である可能性があります。この値は、テキスト検索の実行時に、削除された行のデータが [OPTIMIZE TABLE](#) ステートメントによって FULLTEXT インデックスから物理的に削除される前に、[INNODB_FT_INDEX_TABLE](#) テーブルの行をスキップするために使用されます。詳細は、[InnoDB 全文インデックスの最適化](#) を参照してください。

例

```

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_DELETED;
+-----+
| DOC_ID |
+-----+
| 6 |
| 7 |
| 8 |

```

+-----+

メモ

- このテーブルをクエリーするには `PROCESS` 権限が必要です。
- `INFORMATION_SCHEMA COLUMNS` テーブルまたは `SHOW COLUMNS` ステートメントを使用して、データ型やデフォルト値など、このテーブルのカラムに関する追加情報を表示します。
- InnoDB FULLTEXT 検索の詳細は、[セクション15.6.2.4「InnoDB FULLTEXT インデックス」](#) および [セクション12.10「全文検索関数」](#) を参照してください。

26.51.17 INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE テーブル

`INNODB_FT_INDEX_CACHE` テーブルには、`FULLTEXT` インデックスに新しく挿入された行に関するトークン情報が表示されます。DML 操作中の高コストのインデックス再編成を回避するために、新しくインデックス付けされたワードに関する情報は個別に格納され、`OPTIMIZE TABLE` の実行時、サーバーの停止時、またはキャッシュサイズが `innodb_ft_cache_size` または `innodb_ft_total_cache_size` システム変数で定義された制限を超えた場合にのみメイン検索インデックスと結合されます。

このテーブルは最初は空です。クエリーする前に、`innodb_ft_aux_table` システム変数の値を、`FULLTEXT` インデックスを含むテーブルの名前 (`test/articles` など) に設定します。

関連する使用法と使用例については、[セクション15.15.4「InnoDB INFORMATION_SCHEMA FULLTEXT インデックステーブル」](#) を参照してください。

`INNODB_FT_INDEX_CACHE` テーブルには、次のカラムがあります:

- `WORD`
新しく挿入された行のテキストから抽出された単語。
- `FIRST_DOC_ID`
この単語が `FULLTEXT` インデックスに出現する最初のドキュメント ID。
- `LAST_DOC_ID`
この単語が `FULLTEXT` インデックスに出現する最後のドキュメント ID。
- `DOC_COUNT`
この単語が `FULLTEXT` インデックスに出現する行数。同じ単語は、`DOC_ID` 値と `POSITION` 値の組み合わせごとに一度ずつ、キャッシュテーブル内で複数回現れる可能性があります。
- `DOC_ID`
新しく挿入された行のドキュメント ID。この値は、基礎となるテーブルに定義した ID カラムの値を反映しているか、テーブルに適切なカラムが含まれていない場合に InnoDB によって生成される順序値である可能性があります。
- `POSITION`
`DOC_ID` 値で識別された関連ドキュメント内のこの単語の特定のインスタンス位置。値は絶対位置を表しません。その単語の 1 つ前のインスタンスの `POSITION` に追加されたオフセットです。

メモ

- このテーブルは最初は空です。クエリーする前に、`innodb_ft_aux_table` システム変数の値を、`FULLTEXT` インデックスを含むテーブルの名前 (`test/articles` など) に設定します。次の例は、`innodb_ft_aux_table` システム変数を使用して、指定したテーブルの `FULLTEXT` インデックスに関する情報を表示する方法を示しています。

```
mysql> USE test;
```

```
mysql> CREATE TABLE articles (
  id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
  title VARCHAR(200),
  body TEXT,
  FULLTEXT (title,body)
) ENGINE=InnoDB;

mysql> INSERT INTO articles (title,body) VALUES
('MySQL Tutorial','DBMS stands for DataBase ...'),
('How To Use MySQL Well','After you went through a ...'),
('Optimizing MySQL','In this tutorial we show ...'),
('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
('MySQL vs. YourSQL','In the following database comparison ...'),
('MySQL Security','When configured properly, MySQL ...');

mysql> SET GLOBAL innodb_ft_aux_table = 'test/articles';

mysql> SELECT WORD, DOC_COUNT, DOC_ID, POSITION
FROM INFORMATION_SCHEMA.INNODB_FT_INDEX_CACHE LIMIT 5;
+-----+-----+-----+-----+
| WORD      | DOC_COUNT | DOC_ID | POSITION |
+-----+-----+-----+-----+
| 1001      | 1         | 4      | 0       |
| after     | 1         | 2      | 22      |
| comparison | 1         | 5      | 44      |
| configured | 1         | 6      | 20      |
| database  | 2         | 1      | 31      |
+-----+-----+-----+-----+
```

- このテーブルをクエリーするには [PROCESS](#) 権限が必要です。
- [INFORMATION_SCHEMA COLUMNS](#) テーブルまたは [SHOW COLUMNS](#) ステートメントを使用して、データ型やデフォルト値など、このテーブルのカラムに関する追加情報を表示します。
- InnoDB FULLTEXT 検索の詳細は、[セクション15.6.2.4「InnoDB FULLTEXT インデックス」](#) および [セクション12.10「全文検索関数」](#) を参照してください。

26.51.18 INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE テーブル

[INNODB_FT_INDEX_TABLE](#) テーブルは、InnoDB テーブルの [FULLTEXT](#) インデックスに対するテキスト検索の処理に使用される逆インデックスに関する情報を提供します。

このテーブルは最初は空です。クエリーする前に、[innodb_ft_aux_table](#) システム変数の値を、[FULLTEXT](#) インデックスを含むテーブルの名前 ([test/articles](#) など) に設定します。

関連する使用法と使用例については、[セクション15.15.4「InnoDB INFORMATION_SCHEMA FULLTEXT インデックステーブル」](#) を参照してください。

[INNODB_FT_INDEX_TABLE](#) テーブルには、次のカラムがあります:

- [WORD](#)
FULLTEXT の一部であるカラムのテキストから抽出された単語。
- [FIRST_DOC_ID](#)
この単語が [FULLTEXT](#) インデックスに出現する最初のドキュメント ID。
- [LAST_DOC_ID](#)
この単語が [FULLTEXT](#) インデックスに出現する最後のドキュメント ID。
- [DOC_COUNT](#)
この単語が [FULLTEXT](#) インデックスに出現する行数。同じ単語は、[DOC_ID](#) 値と [POSITION](#) 値の組み合わせごとに一度ずつ、キャッシュテーブル内で複数回現れる可能性があります。
- [DOC_ID](#)

単語を含む行のドキュメント ID。この値は、基礎となるテーブルに定義した ID カラムの値を反映しているか、テーブルに適切なカラムが含まれていない場合に InnoDB によって生成される順序値である可能性があります。

- POSITION

DOC_ID 値で識別された関連ドキュメント内のこの単語の特定のインスタンス位置。

メモ

- このテーブルは最初は空です。クエリーする前に、`innodb_ft_aux_table` システム変数の値を、FULLTEXT インデックスを含むテーブルの名前 (`test/articles` など) に設定します。次の例は、`innodb_ft_aux_table` システム変数を使用して、指定したテーブルの FULLTEXT インデックスに関する情報を表示する方法を示しています。新しく挿入された行の情報が `INNODB_FT_INDEX_TABLE` に表示される前に、FULLTEXT インデックスキャッシュは、ディスクにフラッシュされる必要があります。これを行うには、`innodb_optimize_fulltext_only` システム変数を有効にして、インデックス付けされたテーブルに対して `OPTIMIZE TABLE` 操作を実行します。(この例では、変数は一時的にのみ有効にすることを意図しているため、最後に再度無効にします。)

```
mysql> USE test;

mysql> CREATE TABLE articles (
  id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
  title VARCHAR(200),
  body TEXT,
  FULLTEXT (title,body)
) ENGINE=InnoDB;

mysql> INSERT INTO articles (title,body) VALUES
('MySQL Tutorial','DBMS stands for DataBase ...'),
('How To Use MySQL Well','After you went through a ...'),
('Optimizing MySQL','In this tutorial we show ...'),
('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
('MySQL vs. YourSQL','In the following database comparison ...'),
('MySQL Security','When configured properly, MySQL ...');

mysql> SET GLOBAL innodb_optimize_fulltext_only=ON;

mysql> OPTIMIZE TABLE articles;
+-----+-----+-----+-----+
| Table   | Op   | Msg_type | Msg_text |
+-----+-----+-----+-----+
| test.articles | optimize | status | OK      |
+-----+-----+-----+-----+

mysql> SET GLOBAL innodb_ft_aux_table = 'test/articles';

mysql> SELECT WORD, DOC_COUNT, DOC_ID, POSITION
FROM INFORMATION_SCHEMA.INNODB_FT_INDEX_TABLE LIMIT 5;
+-----+-----+-----+-----+
| WORD      | DOC_COUNT | DOC_ID | POSITION |
+-----+-----+-----+-----+
| 1001      | 1         | 4      | 0       |
| after     | 1         | 2      | 22      |
| comparison | 1         | 5      | 44      |
| configured | 1         | 6      | 20      |
| database  | 2         | 1      | 31      |
+-----+-----+-----+-----+

mysql> SET GLOBAL innodb_optimize_fulltext_only=OFF;
```

- このテーブルをクエリーするには `PROCESS` 権限が必要です。
- `INFORMATION_SCHEMA COLUMNS` テーブルまたは `SHOW COLUMNS` ステートメントを使用して、データ型やデフォルト値など、このテーブルのカラムに関する追加情報を表示します。
- InnoDB FULLTEXT 検索の詳細は、[セクション15.6.2.4「InnoDB FULLTEXT インデックス」](#) および [セクション12.10「全文検索関数」](#) を参照してください。

26.51.19 INFORMATION_SCHEMA INNODB_INDEXES テーブル

INNODB_INDEXES テーブルは、InnoDB インデックスに関するメタデータを提供します。

関連する使用法と使用例については、[セクション15.15.3「InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル」](#)を参照してください。

INNODB_INDEXES テーブルには、次のカラムがあります:

- INDEX_ID

インデックスの識別子。インデックス識別子は、インスタンス内のすべてのデータベースで一意です。

- NAME

インデックスの名前。InnoDB によって暗黙的に作成されたほとんどのインデックスには一意の名前が付けられていますが、インデックス名は必ずしも一意ではありません。例: 主キーインデックスの場合は PRIMARY、主キーを表すインデックスが指定されていない場合は GEN_CLUST_INDEX、外部キー制約の場合は ID_IND、FOR_IND および REF_IND。

- TABLE_ID

インデックスに関連付けられたテーブルを表す識別子 (INNODB_TABLES.TABLE_ID と同じ値)。

- TYPE

インデックスタイプを識別するビットレベルの情報から導出される数値。0 = 一意でないセカンダリインデックス、1 = 自動的に生成されるクラスタインデックス (GEN_CLUST_INDEX)、2 = 一意の非クラスタインデックス、3 = クラスタインデックス、32 = フルテキストインデックス、64 = 空間インデックス、128 = virtual generated column 上のセカンダリインデックス。

- N_FIELDS

インデックスキーのカラムの数。GEN_CLUST_INDEX インデックスの場合、インデックスは実際のテーブルのカラムではなく人為的な値を使用して作成されるため、この値は 0 です。

- PAGE_NO

インデックス B ツリーのルートページ番号。全文インデックスの場合、全文インデックスは複数の B ツリー (補助テーブル) に配置されるため、PAGE_NO カラムは使用されず、-1 (FIL_NULL) に設定されます。

- SPACE

インデックスが存在するテーブルスペースの識別子。0 は InnoDB システムテーブルスペースを示します。その他の数値は、file-per-table モードで別の .ibd ファイルを使用して作成されたテーブルをテーブルします。この識別子は、TRUNCATE TABLE ステートメントのあとでも同じままです。テーブルのすべてのインデックスが、テーブルと同じテーブルスペースに存在するので、この値は必ずしも一意にはなりません。

- MERGE_THRESHOLD

インデックスページのマージしきい値。行が削除されたとき、または更新操作によって行が短縮されたときに、インデックスページのデータ量が MERGE_THRESHOLD 値を下回った場合、InnoDB はインデックスページを隣接するインデックスページとマージしようとします。デフォルトのしきい値は 50% です。詳細は、[セクション 15.8.11「インデックスページのマージしきい値の構成」](#)を参照してください。

例

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_INDEXES WHERE TABLE_ID = 34\G
***** 1. row *****
INDEX_ID: 39
NAME: GEN_CLUST_INDEX
TABLE_ID: 34
TYPE: 1
N_FIELDS: 0
PAGE_NO: 3
SPACE: 23
MERGE_THRESHOLD: 50
```



```

***** 2. row *****
INDEX_ID: 40
NAME: i1
TABLE_ID: 34
TYPE: 0
N_FIELDS: 1
PAGE_NO: 4
SPACE: 23
MERGE_THRESHOLD: 50

```

メモ

- このテーブルをクエリーするには [PROCESS](#) 権限が必要です。
- [INFORMATION_SCHEMA COLUMNS](#) テーブルまたは [SHOW COLUMNS](#) ステートメントを使用して、データ型やデフォルト値など、このテーブルのカラムに関する追加情報を表示します。

26.51.20 INFORMATION_SCHEMA INNODB_LOCKS テーブル

[INNODB_LOCKS](#) テーブルには、[InnoDB](#) トランザクションがリクエストしたがまだ取得していない各ロックと、別のトランザクションをブロックしているトランザクションが保持している各ロックに関する情報が表示されます。

注記

このテーブルは非推奨で、MySQL 8.0.1 の時点で削除されています。代わりに、パフォーマンススキーマ [data_locks](#) テーブルを使用してください。 [セクション 27.12.13.1 「data_locks テーブル」](#) を参照してください。

[INNODB_LOCKS](#) と [data_locks](#) の相違点:

- トランザクションがロックを保持している場合、[INNODB_LOCKS](#) は、別のトランザクションがロックを待機している場合にのみロックを表示します。[data_locks](#) では、トランザクションが待機しているかどうかに関係なく、ロックが表示されます。
- [data_locks](#) テーブルには、[LOCK_SPACE](#)、[LOCK_PAGE](#) または [LOCK_REC](#) に対応するカラムはありません。
- [INNODB_LOCKS](#) テーブルには、グローバル [PROCESS](#) 権限が必要です。[data_locks](#) テーブルには、選択元のテーブルに対する [SELECT](#) の通常のパフォーマンススキーマ権限が必要です。

次のテーブルに、[INNODB_LOCKS](#) カラムから [data_locks](#) カラムへのマッピングを示します。この情報を使用して、あるテーブルから別のテーブルにアプリケーションを移行します。

表 26.4 [INNODB_LOCKS](#) から [data_locks](#) カラムへのマッピング

INNODB_LOCKS カラム	data_locks カラム
LOCK_ID	ENGINE_LOCK_ID
LOCK_TRX_ID	ENGINE_TRANSACTION_ID
LOCK_MODE	LOCK_MODE
LOCK_TYPE	LOCK_TYPE
LOCK_TABLE (スキーマ名とテーブル名の組合せ)	OBJECT_SCHEMA (スキーマ名)、 OBJECT_NAME (テーブル名)
LOCK_INDEX	INDEX_NAME
LOCK_SPACE	なし
LOCK_PAGE	なし
LOCK_REC	なし
LOCK_DATA	LOCK_DATA

26.51.21 INFORMATION_SCHEMA INNODB_LOCK_WAITS テーブル

INNODB_LOCK_WAITS テーブルには、ブロックされている InnoDB トランザクションごとに 1 つ以上の行が含まれ、トランザクションで要求したロックと、その要求をブロックしているすべてのロックを示します。

注記

このテーブルは非推奨で、MySQL 8.0.1 の時点で削除されています。代わりに、パフォーマンススキーマ `data_lock_waits` テーブルを使用してください。 [セクション 27.12.13.2 「data_lock_waits テーブル」](#) を参照してください。

テーブルは必要な権限が異なります: INNODB_LOCK_WAITS テーブルには、グローバル PROCESS 権限が必要です。 data_lock_waits テーブルには、選択元のテーブルに対する SELECT の通常のパフォーマンススキーマ権限が必要です。

次のテーブルに、INNODB_LOCK_WAITS カラムから data_lock_waits カラムへのマッピングを示します。この情報を使用して、あるテーブルから別のテーブルにアプリケーションを移行します。

表 26.5 INNODB_LOCK_WAITS から data_lock_waits カラムへのマッピング

INNODB_LOCK_WAITS カラム	data_lock_waits カラム
REQUESTING_TRX_ID	REQUESTING_ENGINE_TRANSACTION_ID
REQUESTED_LOCK_ID	REQUESTING_ENGINE_LOCK_ID
BLOCKING_TRX_ID	BLOCKING_ENGINE_TRANSACTION_ID
BLOCKING_LOCK_ID	BLOCKING_ENGINE_LOCK_ID

26.51.22 INFORMATION_SCHEMA INNODB_METRICS テーブル

INNODB_METRICS テーブルは、様々な InnoDB パフォーマンス情報を提供し、InnoDB の「パフォーマンススキーマ」テーブルの特定のフォーカス領域を補完します。単純なクエリーで、システムのヘルス全体を確認できます。より詳細なクエリーを使用すると、パフォーマンスのボトルネック、リソース不足、アプリケーション問題などの問題を診断できます。

各モニターは、カウンタ情報を収集するようにインストールメントされる InnoDB ソースコード内でのポイントを表します。各カウンタは、開始、停止、およびリセットできます。共通のモジュール名を使用して、カウンタのグループに対して、これらのアクションを実行することもできます。

デフォルトでは、比較的少量のデータが収集されます。カウンタを起動、停止およびリセットするには、カウンタの名前、モジュールの名前、「%」文字を使用したそのような名前のワイルドカード一致、または特殊なキーワード `all` を使用して、システム変数 `innodb_monitor_enable`, `innodb_monitor_disable`, `innodb_monitor_reset` または `innodb_monitor_reset_all` のいずれかを設定します。

用法については、 [セクション 15.15.6 「InnoDB INFORMATION_SCHEMA メトリックテーブル」](#) を参照してください。

INNODB_METRICS テーブルには、次のカラムがあります:

- **NAME**
カウンタの一意の名前。
- **SUBSYSTEM**
メトリックが適用される InnoDB の側面。
- **COUNT**
カウンタが有効化されてからの値。
- **MAX_COUNT**

カウンタが有効になってからの最大値。

- **MIN_COUNT**

カウンタが有効化されてからの最小値。

- **AVG_COUNT**

カウンタが有効化されてからの平均値。

- **COUNT_RESET**

最後にリセットされてからのカウンタ値。 (**_RESET** カラムはストップウォッチのラップカウンタのように機能: 一定の時間間隔中にアクティビティを測定できますが、累積値は **COUNT**、**MAX_COUNT** など引き続き使用できません。)

- **MAX_COUNT_RESET**

最後のリセット以降の最大カウンタ値。

- **MIN_COUNT_RESET**

最後にリセットされてからの最小カウンタ値。

- **AVG_COUNT_RESET**

最後にリセットされてからの平均カウンタ値。

- **TIME_ENABLED**

最後の起動のタイムスタンプ。

- **TIME_DISABLED**

最後の停止のタイムスタンプ。

- **TIME_ELAPSED**

カウンタが起動してからの経過時間 (秒)。

- **TIME_RESET**

最後のリセットのタイムスタンプ。

- **STATUS**

カウンタがまだ実行中か (**enabled**) 停止しているか (**disabled**)。

- **TYPE**

項目が累積カウンタであるのか、または一部のリソースの現在値を測定しているのか。

- **COMMENT**

カウンタの説明。

例

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='dml_inserts'\G
***** 1. row *****
  NAME: dml_inserts
SUBSYSTEM: dml
  COUNT: 3
MAX_COUNT: 3
MIN_COUNT: NULL
AVG_COUNT: 0.046153846153846156
```

```

COUNT_RESET: 3
MAX_COUNT_RESET: 3
MIN_COUNT_RESET: NULL
AVG_COUNT_RESET: NULL
TIME_ENABLED: 2014-12-04 14:18:28
TIME_DISABLED: NULL
TIME_ELAPSED: 65
TIME_RESET: NULL
STATUS: enabled
TYPE: status_counter
COMMENT: Number of rows inserted

```

メモ

- このテーブルをクエリーするには [PROCESS](#) 権限が必要です。
- [INFORMATION_SCHEMA COLUMNS](#) テーブルまたは [SHOW COLUMNS](#) ステートメントを使用して、データ型やデフォルト値など、このテーブルのカラムに関する追加情報を表示します。
- トランザクションカウンタの [COUNT](#) 値は、パフォーマンススキーマ [EVENTS_TRANSACTIONS_SUMMARY](#) テーブルで報告されるトランザクションイベントの数とは異なる場合があります。InnoDB は、実行したトランザクションのみをカウントしますが、パフォーマンススキーマは、サーバーによって開始された中断されていないすべてのトランザクション (空のトランザクションを含む) のイベントを収集します。

26.51.23 INFORMATION_SCHEMA INNODB_SESSION_TEMP_TABLESPACES テーブル

[INNODB_SESSION_TEMP_TABLESPACES](#) テーブルは、内部およびユーザー作成の一時テーブルに使用されるセッション一時テーブルスペースに関するメタデータを提供します。このテーブルは、MySQL 8.0.13 で追加されました。

[INNODB_SESSION_TEMP_TABLESPACES](#) テーブルには、次のカラムがあります:

- **ID**
プロセスまたはセッション ID。
- **SPACE**
テーブルスペース ID。400 万の領域 ID の範囲は、セッション一時テーブルスペース用に予約されています。セッション一時テーブルスペースは、サーバーが起動するたびに再作成されます。スペース ID は、サーバーの停止時に永続化されず、再利用できます。
- **PATH**
テーブルスペースデータファイルのパス。セッション一時テーブルスペースには、[ibt](#) ファイル拡張子が付きます。
- **SIZE**
テーブルスペースのサイズ (バイト単位)。
- **STATE**
テーブルスペースの状態。[ACTIVE](#) は、テーブルスペースがセッションで現在使用されていることを示します。[INACTIVE](#) は、テーブルスペースが使用可能なセッション一時テーブルスペースのプール内にあることを示します。
- **PURPOSE**
テーブルスペースの目的。[INTRINSIC](#) は、テーブルスペースがオプティマイザによって最適化された内部一時テーブルの使用に使用されることを示します。[SLAVE](#) は、ユーザーが作成した一時テーブルをレプリケーションスレーブに格納するためにテーブルスペースが割り当てられていることを示します。[USER](#) は、テーブルスペースがユーザー作成一時テーブルに使用されることを示します。[NONE](#) は、テーブルスペースが使用されていないことを示します。

例

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_SESSION_TEMP_TABLESPACES;
+----+-----+-----+-----+-----+-----+
| ID | SPACE | PATH | SIZE | STATE | PURPOSE |
+----+-----+-----+-----+-----+-----+
| 8 | 4294566162 | ./#innodb_temp/temp_10.ibt | 81920 | ACTIVE | INTRINSIC |
| 8 | 4294566161 | ./#innodb_temp/temp_9.ibt | 98304 | ACTIVE | USER |
| 0 | 4294566153 | ./#innodb_temp/temp_1.ibt | 81920 | INACTIVE | NONE |
| 0 | 4294566154 | ./#innodb_temp/temp_2.ibt | 81920 | INACTIVE | NONE |
| 0 | 4294566155 | ./#innodb_temp/temp_3.ibt | 81920 | INACTIVE | NONE |
| 0 | 4294566156 | ./#innodb_temp/temp_4.ibt | 81920 | INACTIVE | NONE |
| 0 | 4294566157 | ./#innodb_temp/temp_5.ibt | 81920 | INACTIVE | NONE |
| 0 | 4294566158 | ./#innodb_temp/temp_6.ibt | 81920 | INACTIVE | NONE |
| 0 | 4294566159 | ./#innodb_temp/temp_7.ibt | 81920 | INACTIVE | NONE |
| 0 | 4294566160 | ./#innodb_temp/temp_8.ibt | 81920 | INACTIVE | NONE |
+----+-----+-----+-----+-----+-----+
```

メモ

- このテーブルをクエリーするには `PROCESS` 権限が必要です。
- `INFORMATION_SCHEMA COLUMNS` テーブルまたは `SHOW COLUMNS` ステートメントを使用して、データ型やデフォルト値など、このテーブルのカラムに関する追加情報を表示します。

26.51.24 INFORMATION_SCHEMA INNODB_TABLES テーブル

`INNODB_TABLES` テーブルは、`InnoDB` テーブルに関するメタデータを提供します。

関連する使用法と使用例については、[セクション15.15.3「InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル」](#)を参照してください。

`INNODB_TABLES` テーブルには、次のカラムがあります:

• `TABLE_ID`

`InnoDB` テーブルの識別子。この値は、インスタンス内のすべてのデータベースで一貫しています。

• `NAME`

テーブルの名前。必要に応じてスキーマ (データベース) 名が前に付きます (例: `test/t1`)。データベースおよびユーザーテーブルの名前の大文字/小文字は、もともと定義されていたものと同じで、`lower_case_table_names` 設定による影響を受ける可能性があります。

• `FLAG`

テーブル形式および記憶特性に関するビットレベルの情報を表す数値。

• `N_COLS`

テーブル内のカラムの数。レポートされる数値には、`InnoDB` によって作成される 3 つの非表示カラム (`DB_ROW_ID`、`DB_TRX_ID`、`DB_ROLL_PTR`) が含まれます。レポートされる数には、`virtual generated columns` も含まれます (存在する場合)。

• `SPACE`

テーブルが存在するテーブルスペースの識別子。0 は `InnoDB システムテーブルスペース` を示します。その他の数値は、`file-per-table` テーブルスペースまたは一般テーブルスペースのいずれかを表します。この識別子は、`TRUNCATE TABLE` ステートメントのあとでも同じままです。file-per-table テーブルスペースの場合、この識別子はインスタンス内のすべてのデータベースのテーブルに対して一貫しています。

• `ROW_FORMAT`

テーブルの行形式 (`Compact`、`Redundant`、`Dynamic` または `Compressed`)。

• `ZIP_PAGE_SIZE`

Zip ページサイズ。行フォーマットが **Compressed** のテーブルにのみ適用されます。

- **SPACE_TYPE**

テーブルが属するテーブルスペースのタイプ。使用可能な値は、システムテーブルスペースの場合は **System**、一般テーブルスペースの場合は **General**、file-per-table テーブルスペースの場合は **Single** です。**CREATE TABLE** または **ALTER TABLE TABLESPACE=innodb_system** を使用してシステムテーブルスペースに割り当てられたテーブルの **SPACE_TYPE** は **General** です。詳細は、**CREATE TABLESPACE** を参照してください。

- **INSTANT_COLS**

ALTER TABLE ... ADD COLUMN と **ALGORITHM=INSTANT** を使用して最初のインスタントカラムを追加する前のテーブルのカラム数。

例

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TABLES WHERE TABLE_ID = 214\G
***** 1. row *****
TABLE_ID: 214
NAME: test/t1
FLAG: 129
N_COLS: 4
SPACE: 233
ROW_FORMAT: Compact
ZIP_PAGE_SIZE: 0
SPACE_TYPE: General
INSTANT_COLS: 0
```

メモ

- このテーブルをクエリーするには **PROCESS** 権限が必要です。
- **INFORMATION_SCHEMA COLUMNS** テーブルまたは **SHOW COLUMNS** ステートメントを使用して、データ型やデフォルト値など、このテーブルのカラムに関する追加情報を表示します。

26.51.25 INFORMATION_SCHEMA INNODB_TABLESPACES テーブル

INNODB_TABLESPACES テーブルは、**InnoDB file-per-table**、**general** および **undo** テーブルスペースに関するメタデータを提供します。

関連する使用法と使用例については、[セクション15.15.3「InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル」](#)を参照してください。

注記

INFORMATION_SCHEMA FILES テーブルには、file-per-table テーブルスペース、一般テーブルスペース、システムテーブルスペース、グローバル一時テーブルスペースおよび undo テーブルスペースを含む **InnoDB** テーブルスペースタイプのメタデータがレポートされます。

INNODB_TABLESPACES テーブルには、次のカラムがあります:

- **SPACE**

テーブルスペース ID。

- **NAME**

スキーマ (データベース) およびテーブル名。

- **FLAG**

テーブルスペースの形式および記憶特性に関するビットレベルの情報を表す数値。

- **ROW_FORMAT**

テーブルスペースの行形式 ([Compact or Redundant](#)、[Dynamic](#) または [Compressed](#) または [Undo](#))。このカラムのデータは、データファイルに存在するテーブルスペースフラグ情報から解釈されます。

テーブルスペースの行形式が [Redundant](#) または [Compact](#) の場合、このフラグ情報から判断する方法はありません。そのため、使用可能な [ROW_FORMAT](#) 値のいずれかが [Compact or Redundant](#) になります。

- **PAGE_SIZE**

テーブルスペースのページサイズ。このカラムのデータは、[.ibd file](#) に存在するテーブルスペースフラグ情報から解釈されます。

- **ZIP_PAGE_SIZE**

テーブルスペースの Zip ページサイズ。このカラムのデータは、[.ibd file](#) に存在するテーブルスペースフラグ情報から解釈されます。

- **SPACE_TYPE**

テーブルスペースのタイプ。使用可能な値は、一般的なテーブルスペースの場合は [General](#)、file-per-table テーブルスペースの場合は [Single](#)、システムテーブルスペースの場合は [System](#)、undo テーブルスペースの場合は [Undo](#) です。

- **FS_BLOCK_SIZE**

ホールパンチングに使用される単位サイズであるファイルシステムのブロックサイズ。このカラムは、[InnoDB transparent page compression](#) 機能に関連します。

- **FILE_SIZE**

圧縮解除されたファイルの最大サイズを表す、ファイルの見かけ上のサイズ。このカラムは、[InnoDB transparent page compression](#) 機能に関連します。

- **ALLOCATED_SIZE**

ファイルの実際のサイズ。これは、ディスクに割り当てられた領域の量です。このカラムは、[InnoDB transparent page compression](#) 機能に関連します。

- **AUTOEXTEND_SIZE**

テーブルスペースの自動拡張サイズ。このカラムは、MySQL 8.0.23 で追加されました。

- **SERVER_VERSION**

テーブルスペースを作成した MySQL バージョン、テーブルスペースのインポート先の MySQL バージョン、または最後のメジャー MySQL バージョンアップグレードのバージョン。この値は、MySQL 8.0 からのアップグレードなど、リリースシリーズアップグレードによって変更されません。x から 8.0 へ。y。値は、テーブルスペースの「作成」マーカーまたは「認定」マーカーとみなすことができます。

- **SPACE_VERSION**

テーブルスペース形式の変更を追跡するために使用されるテーブルスペースのバージョン。

- **ENCRYPTION**

テーブルスペースが暗号化されているかどうか。このカラムは、MySQL 8.0.13 で追加されました。

- **STATE**

テーブルスペースの状態。このカラムは、MySQL 8.0.14 で追加されました。

file-per-table および general テーブルスペースの場合、状態は次のとおりです:

- **normal**: テーブルスペースは正常でアクティブです。

- **discarded**: テーブルスペースが `ALTER TABLE ... DISCARD TABLESPACE` ステートメントによって破棄されました。
- **corrupted**: テーブルスペースは、InnoDB によって破損として識別されます。

undo テーブルスペースの場合、状態は次のとおりです:

- **active**: undo テーブルスペースのロールバックセグメントは、新しいトランザクションに割り当てることができません。
- **inactive**: undo テーブルスペースのロールバックセグメントは、新しいトランザクションでは使用されなくなりました。切捨てプロセスが進行中です。undo テーブルスペースがパージスレッドによって暗黙的に選択されたが、`ALTER UNDO TABLESPACE ... SET INACTIVE` ステートメントによって非アクティブにされました。
- **empty**: undo テーブルスペースは切り捨てられ、アクティブではなくなりました。これで、`ALTER UNDO TABLESPACE ... SET INACTIVE` ステートメントによって削除または再度アクティブ化する準備ができました。

例

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TABLESPACES WHERE SPACE = 26\G
***** 1. row *****
SPACE: 26
NAME: test/t1
FLAG: 0
ROW_FORMAT: Compact or Redundant
PAGE_SIZE: 16384
ZIP_PAGE_SIZE: 0
SPACE_TYPE: Single
FS_BLOCK_SIZE: 4096
FILE_SIZE: 98304
ALLOCATED_SIZE: 65536
AUTOEXTEND_SIZE: 0
SERVER_VERSION: 8.0.23
SPACE_VERSION: 1
ENCRYPTION: N
STATE: normal
```

メモ

- このテーブルをクエリーするには `PROCESS` 権限が必要です。
- `INFORMATION_SCHEMA COLUMNS` テーブルまたは `SHOW COLUMNS` ステートメントを使用して、データ型やデフォルト値など、このテーブルのカラムに関する追加情報を表示します。

26.51.26 INFORMATION_SCHEMA INNODB_TABLESPACES_BRIEF テーブル

`INNODB_TABLESPACES_BRIEF` テーブルには、file-per-table、general、undo および system テーブルスペースの領域 ID、名前、パス、フラグおよび領域タイプのメタデータが表示されます。

`INNODB_TABLESPACES` は同じメタデータを提供しますが、`FS_BLOCK_SIZE`、`FILE_SIZE`、`ALLOCATED_SIZE` など、テーブルによって提供される他のメタデータを動的にロードする必要があるため、ロード速度が遅くなります。

スペースおよびパスのメタデータは、`INNODB_DATAFILES` テーブルによっても提供されます。

`INNODB_TABLESPACES_BRIEF` テーブルには、次のカラムがあります:

- **SPACE**
テーブルスペース ID。
- **NAME**
テーブルスペース名。file-per-table テーブルスペースの場合、名前は `schema/table_name` の形式になります。
- **PATH**

テーブルスペースデータファイルのパス。 [file-per-table](#) テーブルスペースが MySQL データディレクトリ外の場所に作成される場合、パス値は完全修飾ディレクトリパスです。それ以外の場合、パスはデータディレクトリに対する相対パスになります。

- **FLAG**

テーブルスペースの形式および記憶特性に関するビットレベルの情報を表す数値。

- **SPACE_TYPE**

テーブルスペースのタイプ。使用可能な値には、 [General](#) for InnoDB general tablespaces、 [Single](#) for InnoDB file-per-table tablespaces、 [System](#) for the InnoDB system tablespace などがあります。

例

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TABLESPACES_BRIEF WHERE SPACE = 7;
+-----+-----+-----+-----+-----+
|SPACE|NAME  |PATH      |FLAG |SPACE_TYPE|
+-----+-----+-----+-----+-----+
| 7   |test/t1|.test/t1.ibd|16417|Single   |
+-----+-----+-----+-----+-----+
```

メモ

- このテーブルをクエリーするには [PROCESS](#) 権限が必要です。
- [INFORMATION_SCHEMA COLUMNS](#) テーブルまたは [SHOW COLUMNS](#) ステートメントを使用して、データ型やデフォルト値など、このテーブルのカラムに関する追加情報を表示します。

26.51.27 INFORMATION_SCHEMA INNODB_TABLESTATS ビュー

[INNODB_TABLESTATS](#) テーブルには、InnoDB テーブルに関する低レベルのステータス情報のビューが表示されます。このデータは、InnoDB テーブルのクエリー時に使用するインデックスを計算するために MySQL オプティマイザによって使用されます。この情報は、ディスクに格納されているデータではなく、インメモリーデータ構造から導出されます。対応する内部 InnoDB システムテーブルはありません。

InnoDB テーブルは、前回のサーバー再起動以降にオープンされ、テーブルキャッシュからエージアウトされていない場合に、このビューに表示されます。永続的統計を利用できるテーブルは、このビューに常に表示されます。

テーブル統計は、インデックス付けされたカラムを変更する [DELETE](#) または [UPDATE](#) 操作に対してのみ更新されます。インデックス付けされていないカラムのみを変更する操作では、統計は更新されません。

[ANALYZE TABLE](#) によってテーブル統計がクリアされ、[STATS_INITIALIZED](#) カラムが [Uninitialized](#) に設定されます。統計は、次回テーブルにアクセスしたときに再度収集されます。

関連する使用法と使用例については、[セクション15.15.3「InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル」](#)を参照してください。

[INNODB_TABLESTATS](#) テーブルには、次のカラムがあります:

- **TABLE_ID**

統計が使用可能なテーブルを表す識別子 ([INNODB_TABLES.TABLE_ID](#) と同じ値)。

- **NAME**

テーブルの名前。 [INNODB_TABLES.NAME](#) と同じ値です。

- **STATS_INITIALIZED**

値は、統計がすでに収集されている場合は [Initialized](#) で、収集されていない場合は [Uninitialized](#) です。

- **NUM_ROWS**

現在の推定されるテーブル内の行数。それぞれの DML 操作後に更新されます。コミットされていないトランザクションがテーブルに対して挿入または削除されている場合、値が正しくない可能性があります。

- [CLUST_INDEX_SIZE](#)

InnoDB テーブルデータを主キー順に保持する、クラスタ化されたインデックスを格納するディスク上のページ数。この値は、テーブルの統計がまだ収集されていない場合は NULL になることがあります。

- [OTHER_INDEX_SIZE](#)

テーブルのすべてのセカンダリインデックスを格納するディスク上のページ数。この値は、テーブルの統計がまだ収集されていない場合は NULL になることがあります。

- [MODIFIED_COUNTER](#)

INSERT、UPDATE、DELETE などの DML 操作と外部キーのカスケード操作によって変更された行数。このカラムは、テーブル統計が再計算されるごとにリセットされます。

- [AUTOINC](#)

すべての自動インクリメントベースの操作で発行される次の番号。AUTOINC 値の変更ペースは、自動インクリメント番号が要求された回数と、要求ごとに認められる番号の数に応じて異なります。

- [REF_COUNT](#)

このカウンタがゼロになると、テーブルメタデータをテーブルキャッシュから削除できます。

例

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TABLESTATS where TABLE_ID = 71\G
***** 1. row *****
TABLE_ID: 71
NAME: test/t1
STATS_INITIALIZED: Initialized
NUM_ROWS: 1
CLUST_INDEX_SIZE: 1
OTHER_INDEX_SIZE: 0
MODIFIED_COUNTER: 1
AUTOINC: 0
REF_COUNT: 1
```

メモ

- このテーブルは、主にエキスパートレベルのパフォーマンス監視、または MySQL のパフォーマンス関連の拡張機能を開発する場合に役立ちます。
- このテーブルをクエリーするには [PROCESS](#) 権限が必要です。
- [INFORMATION_SCHEMA COLUMNS](#) テーブルまたは [SHOW COLUMNS](#) ステートメントを使用して、データ型やデフォルト値など、このテーブルのカラムに関する追加情報を表示します。

26.51.28 INFORMATION_SCHEMA INNODB_TEMP_TABLE_INFO テーブル

[INNODB_TEMP_TABLE_INFO](#) テーブルは、InnoDB インスタンスでアクティブなユーザー作成の InnoDB 一時テーブルに関する情報を提供します。オプティマイザで使用される内部 InnoDB 一時テーブルに関する情報は提供されません。[INNODB_TEMP_TABLE_INFO](#) テーブルは、最初のクエリー時に作成され、メモリー内のみ存在し、ディスクに永続化されません。

使用方法および例については、[セクション15.15.7「InnoDB INFORMATION_SCHEMA 一時テーブル情報テーブル」](#)を参照してください。

[INNODB_TEMP_TABLE_INFO](#) テーブルには、次のカラムがあります:

- [TABLE_ID](#)

一時テーブルのテーブル ID。

- **NAME**

一時テーブルの名前。

- **N_COLS**

一時テーブルのカラム数。この数値には、**InnoDB (DB_ROW_ID、DB_TRX_ID および DB_ROLL_PTR)** によって作成された 3 つの非表示カラムが含まれます。

- **SPACE**

一時テーブルが存在する一時テーブルスペースの ID。

例

```
mysql> CREATE TEMPORARY TABLE t1 (c1 INT PRIMARY KEY) ENGINE=INNODB;
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TEMP_TABLE_INFO
***** 1. row *****
TABLE_ID: 97
NAME: #sql8c88_43_0
N_COLS: 4
SPACE: 76
```

メモ

- このテーブルは、主にエキスパートレベルの監視に役立ちます。
- このテーブルをクエリーするには **PROCESS** 権限が必要です。
- **INFORMATION_SCHEMA COLUMNS** テーブルまたは **SHOW COLUMNS** ステートメントを使用して、データ型やデフォルト値など、このテーブルのカラムに関する追加情報を表示します。

26.51.29 INFORMATION_SCHEMA INNODB_TRX テーブル

INNODB_TRX テーブルには、**InnoDB** 内で現在実行されているすべてのトランザクションに関する情報が表示されます。これには、トランザクションがロックを待機しているかどうか、トランザクションの開始時、トランザクションが実行されている SQL ステートメント (ある場合) などが含まれます。

使用法については、[セクション15.15.2.1「InnoDB トランザクションの使用および情報のロック」](#)を参照してください。

INNODB_TRX テーブルには、次のカラムがあります：

- **TRX_ID**

InnoDB 内部の一意のトランザクション ID 番号。これらの ID は、読み取り専用および非ロックのトランザクションに対しては作成されません。詳細は、[セクション8.5.3「InnoDB の読み取り専用トランザクションの最適化」](#)を参照してください。

- **TRX_WEIGHT**

トランザクションの重み。これは、トランザクションが変更した行の数とロックした行の数を反映したものです (ただし必ずしも正確な数ではありません)。デッドロックを解決するために、**InnoDB** はロールバックする「被害者」として最小の重みを持つトランザクションを選択します。非トランザクションテーブルを変更したトランザクションは、変更およびロックされた行の数に関係なく、他のトランザクションよりも重いとみなされます。

- **TRX_STATE**

トランザクションの実行状態。許可される値は、**RUNNING**、**LOCK WAIT**、**ROLLING BACK** および **COMMITTING** です。

- [TRX_STARTED](#)

トランザクション開始時間。

- [TRX_REQUESTED_LOCK_ID](#)

[TRX_STATE](#) が [LOCK WAIT](#) の場合は、トランザクションが現在待機しているロックの ID。それ以外の場合は [NULL](#)。ロックの詳細を取得するには、このカラムをパフォーマンススキーマ [data_locks](#) テーブルの [ENGINE_LOCK_ID](#) カラムと結合します。

- [TRX_WAIT_STARTED](#)

[TRX_STATE](#) が [LOCK WAIT](#) の場合、トランザクションがロックの待機を開始した時刻。それ以外の場合は [NULL](#)。

- [TRX_MYSQL_THREAD_ID](#)

MySQL スレッド ID。スレッドの詳細を取得するには、このカラムを [INFORMATION_SCHEMA PROCESSLIST](#) テーブルの [ID](#) カラムと結合しますが、[セクション 15.15.2.3 「InnoDB トランザクションおよびロック情報の永続性と一貫性」](#) を参照してください。

- [TRX_QUERY](#)

トランザクションによって実行されている SQL ステートメント。

- [TRX_OPERATION_STATE](#)

トランザクションの現在の操作 (存在する場合)。それ以外の場合は [NULL](#)。

- [TRX_TABLES_IN_USE](#)

このトランザクションの現在の SQL ステートメントを処理しているときに使用される [InnoDB](#) テーブルの数。

- [TRX_TABLES_LOCKED](#)

現在の SQL ステートメントで行ロックが有効になっている [InnoDB](#) テーブルの数。(これらはテーブルロックではなく行ロックなので、一部の行がロックされているかどうかにかかわらず、通常、複数のトランザクションによるテーブルからの読み取りおよびテーブルへの書き込みを実行できます。)

- [TRX_LOCK_STRUCTS](#)

トランザクションで予約されたロックの数。

- [TRX_LOCK_MEMORY_BYTES](#)

メモリー内のこのトランザクションのロック構造にかかった合計サイズ。

- [TRX_ROWS_LOCKED](#)

このトランザクションによってロックされたおおよその数または行数。この値には、物理的には存在するがトランザクションから認識できない削除マークが付けられた行が含まれる場合があります。

- [TRX_ROWS_MODIFIED](#)

このトランザクションで変更および挿入された行の数。

- [TRX_CONCURRENCY_TICKETS](#)

[innodb_concurrency_tickets](#) システム変数で指定された、スワップアウトされる前に現在のトランザクションが実行できる作業量を示す値。

- [TRX_ISOLATION_LEVEL](#)

現在のトランザクションの分離レベル。

- [TRX_UNIQUE_CHECKS](#)

現在のトランザクションで一意チェックがオンになっているか、オフになっているか。たとえば、バルクデータロード中にオフになる場合があります。

- [TRX_FOREIGN_KEY_CHECKS](#)

現在のトランザクションで外部キーチェックがオンになっているか、オフになっているか。たとえば、バルクデータロード中にオフになる場合があります。

- [TRX_LAST_FOREIGN_KEY_ERROR](#)

最後の外部キーエラーの詳細なエラーメッセージ (存在する場合)。それ以外の場合は `NULL`。

- [TRX_ADAPTIVE_HASH_LATCHED](#)

適応ハッシュインデックスが現在のトランザクションによってロックされているかどうか。適応型ハッシュインデックス検索システムがパーティション化されている場合、単一のトランザクションは適応型ハッシュインデックス全体をロックしません。適応型ハッシュインデックスパーティション化は、デフォルトで 8 に設定されている [innodb_adaptive_hash_index_parts](#) によって制御されます。

- [TRX_ADAPTIVE_HASH_TIMEOUT](#)

アダプティブハッシュインデックスの検索ラッチをすぐに破棄するか、MySQL からの呼び出し全体で保持するか。適応型ハッシュインデックスの競合がない場合、この値はゼロのまま、ステートメントは終了するまでラッチを予約します。競合の間、ゼロまでカウントダウンし、ステートメントは各行ルックアップの直後にラッチを解放します。適応ハッシュインデックス検索システムがパーティション化されている場合 ([innodb_adaptive_hash_index_parts](#) によって制御されます)、値は 0 のままです。

- [TRX_IS_READ_ONLY](#)

値 1 は、トランザクションが読取り専用であることを示します。

- [TRX_AUTOCOMMIT_NON_LOCKING](#)

値 1 は、トランザクションが `FOR UPDATE` 句または `LOCK IN SHARED MODE` 句を使用しない `SELECT` ステートメントであり、トランザクションにこのステートメントのみが含まれるように `autocommit` を有効にして実行されていることを示します。このカラムと [TRX_IS_READ_ONLY](#) がどちらも 1 である場合、InnoDB は、テーブルデータを変更するトランザクションと関連付けられたオーバーヘッドを軽減するように、トランザクションを最適化します。

- [TRX_SCHEDULE_WEIGHT](#)

ロックを待機しているトランザクションに競合対応トランザクションスケジューリング (CATS) アルゴリズムによって割り当てられるトランザクションスケジューリングの重み。この値は、他のトランザクションの値に対して相対的です。値が大きいほど重みが大きくなります。値は、[TRX_STATE](#) カラムでレポートされる `LOCK WAIT` 状態のトランザクションに対してのみ計算されます。ロックを待機していないトランザクションについては、`NULL` 値が報告されます。[TRX_SCHEDULE_WEIGHT](#) の値は、[TRX_WEIGHT](#) の値とは異なります。この値は、目的に応じて異なるアルゴリズムによって計算されます。

例

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TRX
***** 1. row *****
      trx_id: 1510
      trx_state: RUNNING
      trx_started: 2014-11-19 13:24:40
      trx_requested_lock_id: NULL
      trx_wait_started: NULL
      trx_weight: 586739
      trx_mysql_thread_id: 2
      trx_query: DELETE FROM employees.salaries WHERE salary > 65000
      trx_operation_state: updating or deleting
      trx_tables_in_use: 1
      trx_tables_locked: 1
      trx_lock_structs: 3003
      trx_lock_memory_bytes: 450768
```

```

trx_rows_locked: 1407513
trx_rows_modified: 583736
trx_concurrency_tickets: 0
trx_isolation_level: REPEATABLE READ
trx_unique_checks: 1
trx_foreign_key_checks: 1
trx_last_foreign_key_error: NULL
trx_adaptive_hash_latched: 0
trx_adaptive_hash_timeout: 10000
trx_is_read_only: 0
trx_autocommit_non_locking: 0
trx_schedule_weight: NULL

```

メモ

- このテーブルを使用すると、負荷の大きな同時ロードの時間中に生じるパフォーマンスの問題の診断に役立ちます。その内容は、[セクション15.15.2.3「InnoDB トランザクションおよびロック情報の永続性と一貫性」](#)で説明しているように更新されます。
- このテーブルをクエリーするには `PROCESS` 権限が必要です。
- `INFORMATION_SCHEMA COLUMNS` テーブルまたは `SHOW COLUMNS` ステートメントを使用して、データ型やデフォルト値など、このテーブルのカラムに関する追加情報を表示します。

26.51.30 INFORMATION_SCHEMA INNODB_VIRTUAL テーブル

`INNODB_VIRTUAL` テーブルは、[InnoDB virtual generated columns](#) および仮想生成カラムのベースとなるカラムに関するメタデータを提供します。

仮想生成カラムの基になるカラムごとに、`INNODB_VIRTUAL` テーブルに行が表示されます。

`INNODB_VIRTUAL` テーブルには、次のカラムがあります:

- `TABLE_ID`

仮想カラムに関連付けられたテーブルを表す識別子 (`INNODB_TABLES.TABLE_ID` と同じ値)。

- `POS`

`virtual generated column` の位置値。カラムの順序番号と順序位置がエンコードされるため、値は大きくなります。値の計算に使用される式では、ビット単位の演算が使用されます:

```

((nth virtual generated column for the InnoDB instance + 1) << 16)
+ the ordinal position of the virtual generated column

```

たとえば、`InnoDB` インスタンスの最初の仮想生成カラムがテーブルの3番目のカラムである場合、式は $(0 + 1) \ll 16) + 2$ です。`InnoDB` インスタンスの最初の仮想生成カラムは常に数値0です。テーブルの3番目のカラムとして、仮想生成カラムの順序位置は2です。序数の位置は0からカウントされます。

- `BASE_POS`

仮想生成カラムのベースとなるカラムの順序位置。

例

```

mysql> CREATE TABLE `t1` (
  `a` int(11) DEFAULT NULL,
  `b` int(11) DEFAULT NULL,
  `c` int(11) GENERATED ALWAYS AS (a+b) VIRTUAL,
  `h` varchar(10) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_VIRTUAL
WHERE TABLE_ID IN
(SELECT TABLE_ID FROM INFORMATION_SCHEMA.INNODB_TABLES
WHERE NAME LIKE "test/t1");
+-----+-----+-----+

```

TABLE_ID	POS	BASE_POS
98	65538	0
98	65538	1

メモ

- 次のテーブルに示すように、定数値が **virtual generated column** に割り当てられている場合、そのカラムのエントリは **INNODB_VIRTUAL** テーブルに表示されません。エントリを表示するには、仮想生成カラムにベースカラムが必要です。

```
CREATE TABLE `t1` (
  `a` int(11) DEFAULT NULL,
  `b` int(11) DEFAULT NULL,
  `c` int(11) GENERATED ALWAYS AS (5) VIRTUAL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

ただし、このようなカラムのメタデータは **INNODB_COLUMNS** テーブルに表示されます。

- このテーブルをクエリーするには **PROCESS** 権限が必要です。
- INFORMATION_SCHEMA COLUMNS** テーブルまたは **SHOW COLUMNS** ステートメントを使用して、データ型やデフォルト値など、このテーブルのカラムに関する追加情報を表示します。

26.52 INFORMATION_SCHEMA スレッドプールテーブル

注記

MySQL 8.0.14 の時点では、スレッドプールの **INFORMATION_SCHEMA** テーブルは「パフォーマンススキーマ」テーブルとしても使用できます。(セクション 27.12.16 「パフォーマンススキーマスレッドプールテーブル」を参照してください。) **INFORMATION_SCHEMA** テーブルは非推奨です。将来のバージョンの MySQL で削除される予定です。アプリケーションは、古いテーブルから新しいテーブルに移行する必要があります。たとえば、アプリケーションで次のクエリーを使用するとします:

```
SELECT * FROM INFORMATION_SCHEMA.TP_THREAD_STATE;
```

アプリケーションでは、かわりに次のクエリーを使用する必要があります:

```
SELECT * FROM performance_schema.tp_thread_state;
```

次のセクションでは、スレッドプールプラグインに関連付けられた **INFORMATION_SCHEMA** テーブルについて説明します(セクション 5.6.3 「MySQL Enterprise Thread Pool」を参照)。スレッドプール操作に関する情報を示します。

- TP_THREAD_GROUP_STATE**: スレッドプールのスレッドグループ状態に関する情報
- TP_THREAD_GROUP_STATS**: スレッドグループ統計
- TP_THREAD_STATE**: スレッドプールのスレッド状態に関する情報

これらのテーブル内の行は、特定時点のスナップショットを表します。 **TP_THREAD_STATE** の場合、スレッドグループのすべての行で特定時点のスナップショットを構成します。そのため、MySQL Server では、スナップショットの生成中にスレッドグループの相互排他ロックを保持します。ただし、 **TP_THREAD_STATE** に対するステートメントが MySQL Server 全体をブロックできないようにするため、すべてのスレッドグループで同時に相互排他ロックを保持することはありません。

INFORMATION_SCHEMA スレッドプールテーブルは、個々のプラグインによって実装され、ロードするかどうかは他のプラグインから独立して決定できます(セクション 5.6.3.2 「スレッドプールのインストール」を参照)。ただし、すべてのテーブルの内容は、有効なスレッドプールプラグインに応じて異なります。テーブルプラグインが有効になってもスレッドプールプラグインが有効になっていない場合、そのテーブルは表示可能になり、アクセスできませんが空です。

26.52.1 INFORMATION_SCHEMA TP_THREAD_GROUP_STATE テーブル

注記

MySQL 8.0.14 の時点では、スレッドプールの `INFORMATION_SCHEMA` テーブルは「パフォーマンススキーマ」テーブルとしても使用できます。(セクション 27.12.16 「パフォーマンススキーマスレッドプールテーブル」を参照してください。) `INFORMATION_SCHEMA` テーブルは非推奨になりました。将来のバージョンの MySQL で削除される予定です。アプリケーションは、古いテーブルから新しいテーブルに移行する必要があります。たとえば、アプリケーションで次のクエリーを使用するとします:

```
SELECT * FROM INFORMATION_SCHEMA.TP_THREAD_GROUP_STATE;
```

アプリケーションでは、かわりに次のクエリーを使用する必要があります:

```
SELECT * FROM performance_schema.tp_thread_group_state;
```

`TP_THREAD_GROUP_STATE` テーブルには、スレッドプール内のスレッドグループごとに 1 つの行があります。それぞれの行には、グループの現在の状態に関する情報が表示されます。

`INFORMATION_SCHEMA TP_THREAD_GROUP_STATE` テーブルのカラムの説明は、セクション 27.12.16.1 「`tp_thread_group_state` テーブル」を参照してください。パフォーマンススキーマ `tp_thread_group_state` テーブルには同等のカラムがあります。

26.52.2 INFORMATION_SCHEMA TP_THREAD_GROUP_STATS テーブル

注記

MySQL 8.0.14 の時点では、スレッドプールの `INFORMATION_SCHEMA` テーブルは「パフォーマンススキーマ」テーブルとしても使用できます。(セクション 27.12.16 「パフォーマンススキーマスレッドプールテーブル」を参照してください。) `INFORMATION_SCHEMA` テーブルは非推奨になりました。将来のバージョンの MySQL で削除される予定です。アプリケーションは、古いテーブルから新しいテーブルに移行する必要があります。たとえば、アプリケーションで次のクエリーを使用するとします:

```
SELECT * FROM INFORMATION_SCHEMA.TP_THREAD_GROUP_STATS;
```

アプリケーションでは、かわりに次のクエリーを使用する必要があります:

```
SELECT * FROM performance_schema.tp_thread_group_stats;
```

`TP_THREAD_GROUP_STATS` テーブルには、スレッドグループごとの統計がレポートされます。グループごとに 1 行があります。

`INFORMATION_SCHEMA TP_THREAD_GROUP_STATS` テーブルのカラムの説明は、セクション 27.12.16.2 「`tp_thread_group_stats` テーブル」を参照してください。パフォーマンススキーマ `tp_thread_group_stats` テーブルには同等のカラムがあります。

26.52.3 INFORMATION_SCHEMA TP_THREAD_STATE テーブル

注記

MySQL 8.0.14 の時点では、スレッドプールの `INFORMATION_SCHEMA` テーブルは「パフォーマンススキーマ」テーブルとしても使用できます。(セクション 27.12.16 「パフォーマンススキーマスレッドプールテーブル」を参照してください。) `INFORMATION_SCHEMA` テーブルは非推奨になりました。将来のバージョンの MySQL で削除される予定です。アプリケーションは、古いテーブルから新しいテーブルに移行する必要があります。たとえば、アプリケーションで次のクエリーを使用するとします:

```
SELECT * FROM INFORMATION_SCHEMA.TP_THREAD_STATE;
```

アプリケーションでは、かわりに次のクエリーを使用する必要があります:

```
SELECT * FROM performance_schema.tp_thread_state;
```

TP_THREAD_STATE テーブルには、接続を処理するためにスレッドプールによって作成されたスレッドごとに 1 つの行があります。

INFORMATION_SCHEMA TP_THREAD_STATE テーブルの列の説明は、[セクション 27.12.16.3 「tp_thread_state テーブル」](#) を参照してください。パフォーマンススキーマ tp_thread_state テーブルには同等の列があります。

26.53 INFORMATION_SCHEMA の接続制御テーブル

次の各セクションでは、[CONNECTION_CONTROL](#) プラグインに関連付けられた INFORMATION_SCHEMA テーブルについて説明します。

26.53.1 INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS テーブル

このテーブルには、アカウント (ユーザー/ホストの組合せ) ごとの接続試行の連続失敗の現在の回数に関する情報が表示されます。

[CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS](#) には、次の列があります:

- [USERHOST](#)

接続試行に失敗したアカウントを示す 'user_name'@'host_name' 形式のユーザー/ホストの組合せ。

- [FAILED_ATTEMPTS](#)

[USERHOST](#) 値に対する接続試行の現在の連続失敗回数。これにより、遅延したかどうかに関係なく、失敗したすべての試行がカウントされます。サーバーがレスポンスに遅延を追加した試行回数は、[FAILED_ATTEMPTS](#) 値と [connection_control_failed_connections_threshold](#) システム変数値の差です。

メモ

- このテーブルを使用できるようにするには、[CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS](#) プラグインをアクティブ化する必要があります。また、[CONNECTION_CONTROL](#) プラグインをアクティブ化するか、テーブルの内容が常に空である必要があります。[セクション 6.4.2 「Connection-Control プラグイン」](#) を参照してください。
- このテーブルには、後続の正常な試行が行われずに 1 つ以上の接続試行が連続して失敗したアカウントの行のみが含まれます。アカウントが正常に接続されると、その失敗した接続数はゼロにリセットされ、サーバーはアカウントに対応する行を削除します。
- 実行時に [connection_control_failed_connections_threshold](#) システム変数に値を割り当てると、累積されたすべての失敗した接続カウンタがゼロにリセットされ、テーブルが空になります。

26.54 INFORMATION_SCHEMA MySQL Enterprise Firewall テーブル

次の各セクションでは、MySQL Enterprise Firewall に関連付けられた INFORMATION_SCHEMA テーブルについて説明します ([セクション 6.4.7 「MySQL Enterprise Firewall」](#) を参照)。ファイアウォールのインメモリーデータキャッシュへのビューを提供します。これらのテーブルは、適切なファイアウォールプラグインが有効になっている場合のみ使用できます。

26.54.1 INFORMATION_SCHEMA MYSQL_FIREWALL_USERS テーブル

[MYSQL_FIREWALL_USERS](#) テーブルは、MySQL Enterprise Firewall のインメモリーデータキャッシュのビューを提供します。登録済ファイアウォールアカウントプロファイルの名前と操作モードがリストされます。ファイアウォールデータの永続的な記憶域を提供する [mysql.firewall_users](#) システムテーブルとともに使用されます。[MySQL Enterprise Firewall テーブル](#) を参照してください。

[MYSQL_FIREWALL_USERS](#) テーブルには、次の列があります:

- [USERHOST](#)

アカウントプロファイル名。各アカウント名の形式は `user_name@host_name` です。

- **MODE**

プロファイルの現在の操作モード。許可されるモード値は、`OFF`、`DETECTING`、`PROTECTING`、`RECORDING` および `RESET` です。意味の詳細は、[ファイアウォール操作の概念](#) を参照してください。

26.54.2 INFORMATION_SCHEMA MYSQL_FIREWALL_WHITELIST テーブル

`MYSQL_FIREWALL_WHITELIST` テーブルは、MySQL Enterprise Firewall のインメモリーデータキャッシュのビューを提供します。登録済ファイアウォールアカウントプロファイルの許可リストルールがリストされます。ファイアウォールデータの永続的な記憶域を提供する `mysql.firewall_whitelist` システムテーブルとともに使用されます。[MySQL Enterprise Firewall テーブル](#) を参照してください。

`MYSQL_FIREWALL_WHITELIST` テーブルには、次のカラムがあります:

- **USERHOST**

アカウントプロファイル名。各アカウント名の形式は `user_name@host_name` です。

- **RULE**

プロファイルの許容可能なステートメントパターンを示す正規化されたステートメント。プロファイル許可リストは、そのルールの和集合です。

26.55 SHOW ステートメントの拡張

`SHOW` ステートメントの一部の拡張は、`INFORMATION_SCHEMA` の実装を伴います。

- `SHOW` を使用すると、`INFORMATION_SCHEMA` 自体の構造に関する情報を取得できます。
- いくつかの `SHOW` ステートメントでは、表示する行をより柔軟に指定できる `WHERE` 句を使用できます。

`INFORMATION_SCHEMA` は情報データベースなので、その名前は `SHOW DATABASES` の出力に含まれます。同様に、`SHOW TABLES` を `INFORMATION_SCHEMA` と一緒に使用すると、テーブルのリストを取得できます。

```
mysql> SHOW TABLES FROM INFORMATION_SCHEMA;
```

```
+-----+
| Tables_in_INFORMATION_SCHEMA |
+-----+
| CHARACTER_SETS                |
| COLLATIONS                    |
| COLLATION_CHARACTER_SET_APPLICABILITY |
| COLUMNS                      |
| COLUMN_PRIVILEGES             |
| ENGINES                      |
| EVENTS                        |
| FILES                         |
| KEY_COLUMN_USAGE              |
| PARTITIONS                   |
| PLUGINS                       |
| PROCESSLIST                  |
| REFERENTIAL_CONSTRAINTS      |
| ROUTINES                     |
| SCHEMATA                     |
| SCHEMA_PRIVILEGES            |
| STATISTICS                   |
| TABLES                      |
| TABLE_CONSTRAINTS           |
| TABLE_PRIVILEGES            |
| TRIGGERS                     |
| USER_PRIVILEGES              |
| VIEWS                        |
+-----+
```

`SHOW COLUMNS` および `DESCRIBE` は個々の `INFORMATION_SCHEMA` テーブルのカラムに関する情報を表示できます。

表示される行を制限する **LIKE** 句を受け入れる **SHOW** ステートメントは、選択した行が満たす必要のあるより一般的な条件を指定する **WHERE** 句も許可します。

```
SHOW CHARACTER SET
SHOW COLLATION
SHOW COLUMNS
SHOW DATABASES
SHOW FUNCTION STATUS
SHOW INDEX
SHOW OPEN TABLES
SHOW PROCEDURE STATUS
SHOW STATUS
SHOW TABLE STATUS
SHOW TABLES
SHOW TRIGGERS
SHOW VARIABLES
```

WHERE 句がある場合、これは **SHOW** ステートメントで表示されるカラム名に対して評価されます。たとえば、**SHOW CHARACTER SET** ステートメントはこれらの出力カラムを生成します。

```
mysql> SHOW CHARACTER SET;
+-----+-----+-----+-----+
| Charset | Description          | Default collation | Maxlen |
+-----+-----+-----+-----+
| big5    | Big5 Traditional Chinese | big5_chinese_ci   | 2      |
| dec8    | DEC West European      | dec8_swedish_ci   | 1      |
| cp850   | DOS West European      | cp850_general_ci  | 1      |
| hp8     | HP West European       | hp8_english_ci    | 1      |
| koi8r   | KOI8-R Relcom Russian  | koi8r_general_ci  | 1      |
| latin1  | cp1252 West European   | latin1_swedish_ci | 1      |
| latin2  | ISO 8859-2 Central European | latin2_general_ci | 1      |
| ...
```

WHERE 句を **SHOW CHARACTER SET** と一緒に使用するには、これらのカラム名を参照します。たとえば、次のステートメントは、デフォルトの照合順序が文字列 **'japanese'** を含む文字セットに関する情報を表示します。

```
mysql> SHOW CHARACTER SET WHERE `Default collation` LIKE '%japanese%';
+-----+-----+-----+-----+
| Charset | Description          | Default collation | Maxlen |
+-----+-----+-----+-----+
| ujis    | EUC-JP Japanese      | ujis_japanese_ci  | 3      |
| sjis    | Shift-JIS Japanese   | sjis_japanese_ci  | 2      |
| cp932   | SJIS for Windows Japanese | cp932_japanese_ci | 2      |
| eucjpms | UJIS for Windows Japanese | eucjpms_japanese_ci | 3      |
+-----+-----+-----+-----+
```

このステートメントはマルチバイト文字セットを表示します。

```
mysql> SHOW CHARACTER SET WHERE Maxlen > 1;
+-----+-----+-----+-----+
| Charset | Description          | Default collation | Maxlen |
+-----+-----+-----+-----+
| big5    | Big5 Traditional Chinese | big5_chinese_ci   | 2      |
| ujis    | EUC-JP Japanese      | ujis_japanese_ci  | 3      |
| sjis    | Shift-JIS Japanese   | sjis_japanese_ci  | 2      |
| euckr   | EUC-KR Korean        | euckr_korean_ci   | 2      |
| gb2312  | GB2312 Simplified Chinese | gb2312_chinese_ci | 2      |
| gbk     | GBK Simplified Chinese | gbk_chinese_ci    | 2      |
| utf8    | UTF-8 Unicode         | utf8_general_ci   | 3      |
| ucs2    | UCS-2 Unicode         | ucs2_general_ci   | 2      |
| cp932   | SJIS for Windows Japanese | cp932_japanese_ci | 2      |
| eucjpms | UJIS for Windows Japanese | eucjpms_japanese_ci | 3      |
+-----+-----+-----+-----+
```


第 27 章 MySQL パフォーマンススキーマ

目次

27.1 パフォーマンススキーマクイックスタート	4239
27.2 パフォーマンススキーマビルド構成	4244
27.3 パフォーマンススキーマ起動構成	4245
27.4 パフォーマンススキーマ実行時構成	4247
27.4.1 パフォーマンススキーマイベントタイミング	4247
27.4.2 パフォーマンススキーマイベントフィルタリング	4250
27.4.3 イベントの事前フィルタリング	4251
27.4.4 インストゥルメントによる事前フィルタリング	4252
27.4.5 オブジェクトによる事前フィルタリング	4253
27.4.6 スレッドによる事前フィルタリング	4255
27.4.7 コンシューマによる事前フィルタリング	4257
27.4.8 コンシューマ構成の例	4259
27.4.9 フィルタリング操作のインストゥルメントまたはコンシューマの指定	4264
27.4.10 インストゥルメントされているものの特定	4264
27.5 パフォーマンススキーマクエリー	4265
27.6 パフォーマンススキーマインストゥルメント命名規則	4265
27.7 パフォーマンススキーマステータスモニタリング	4269
27.8 パフォーマンススキーマの原子的および分子的イベント	4272
27.9 現在および過去のイベントのパフォーマンススキーマテーブル	4272
27.10 パフォーマンススキーマのステートメントダイジェストとサンプリング	4273
27.11 パフォーマンススキーマの一般的なテーブル特性	4277
27.12 パフォーマンススキーマテーブルの説明	4278
27.12.1 パフォーマンススキーマテーブルインデックス	4279
27.12.2 パフォーマンススキーマセットアップテーブル	4282
27.12.3 パフォーマンススキーマインスタンステーブル	4289
27.12.4 パフォーマンススキーマ待機イベントテーブル	4294
27.12.5 パフォーマンススキーマステージイベントテーブル	4299
27.12.6 パフォーマンススキーマステートメントイベントテーブル	4305
27.12.7 パフォーマンススキーマのトランザクションテーブル	4315
27.12.8 パフォーマンススキーマ接続テーブル	4322
27.12.9 パフォーマンススキーマ接続属性テーブル	4325
27.12.10 パフォーマンススキーマのユーザー定義変数テーブル	4329
27.12.11 パフォーマンススキーマレプリケーションテーブル	4329
27.12.12 パフォーマンススキーマ NDB Cluster テーブル	4348
27.12.13 パフォーマンススキーマロックテーブル	4350
27.12.14 パフォーマンススキーマシステム変数テーブル	4358
27.12.15 パフォーマンススキーマのステータス変数のテーブル	4363
27.12.16 パフォーマンススキーマスレッドプールテーブル	4364
27.12.17 パフォーマンススキーマクローンテーブル	4369
27.12.18 パフォーマンススキーマサマリーテーブル	4371
27.12.19 パフォーマンススキーマのその他のテーブル	4397
27.13 パフォーマンススキーマオプションおよび変数リファレンス	4415
27.14 パフォーマンススキーマコマンドオプション	4418
27.15 パフォーマンススキーマシステム変数	4419
27.16 パフォーマンススキーマステータス変数	4435
27.17 パフォーマンススキーマのメモリー割り当てモデル	4438
27.18 パフォーマンススキーマとプラグイン	4439
27.19 問題を診断するためのパフォーマンススキーマの使用	4439
27.19.1 パフォーマンススキーマを使用したクエリープロファイリング	4440
27.19.2 親イベント情報の取得	4442
27.20 パフォーマンススキーマの制約	4443

MySQL パフォーマンススキーマは低レベルで MySQL サーバーの実行をモニタリングするための機能です。パフォーマンススキーマには、これらの特性があります。

- パフォーマンススキーマは、実行時にサーバーの内部実行を検査する方法を提供します。それは、[PERFORMANCE_SCHEMA](#) ストレージエンジンおよび [performance_schema](#) データベースを使用して実装されます。パフォーマンススキーマは、主にパフォーマンスデータに焦点を合わせています。これは、メタデータの検査に使用される [INFORMATION_SCHEMA](#) と異なります。
- パフォーマンススキーマはサーバーイベントをモニターします。「イベント」は、時間がかかり、タイミング情報を収集できるようにインストールされた、サーバーが実行するすべてのことです。一般に、イベントは、関数呼び出し、オペレーティングシステムの待機、解析やソートなどの SQL ステートメント実行のステージ、またはステートメント全体やステートメントのグループなどになります。イベント収集を使用すると、サーバーやいくつかのストレージエンジンの同期呼び出し (相互排他ロックなど) ファイルやテーブルの I/O、テーブルロックに関する情報にアクセスできます。
- パフォーマンススキーマイベントは、サーバーのバイナリログに書き込まれるイベント (これはデータの変更を説明する) やイベントスケジューラのイベント (これはストアードプログラムの一種である) とは区別されます。
- パフォーマンススキーマイベントは MySQL サーバーの特定のインスタンスに固有です。「パフォーマンススキーマ」テーブルはサーバーに対してローカルであるとみなされ、それらに対する変更はバイナリログにレプリケートまたは書き込まれません。
- 現在のイベントに加えて、イベントの履歴とサマリーを取得できます。これにより、インストールされたアクティビティが何回実行され、それらにどのくらいの時間がかかったかを判断できます。イベント情報を取得して、特定のスレッドのアクティビティ、または相互排他ロックやファイルなどの特定のオブジェクトに関連付けられているアクティビティを表示できます。
- [PERFORMANCE_SCHEMA](#) ストレージエンジンは、サーバーソースコードで「インストールメンテーションポイント」を使用して、イベントデータを収集します。
- 収集されたイベントは、[performance_schema](#) データベース内のテーブルに格納されます。これらのテーブルは、ほかのテーブルと同様に [SELECT](#) ステートメントを使用してクエリーできます。
- パフォーマンススキーマの構成は、SQL ステートメントから [performance_schema](#) データベース内のテーブルを更新することによって、動的に変更できます。構成の変更はデータコレクションにすぐに影響しません。
- パフォーマンススキーマ内のテーブルは、永続的なディスク上のストレージを使用しないインメモリーテーブルです。コンテンツは、サーバーの起動時から再移入され、サーバーの停止時に破棄されます。
- モニタリングは MySQL でサポートされているすべてのプラットフォームで使用できます。
いくつかの制限が適用されることがあります。タイマーの種類はプラットフォームごとに異なることがあります。ストレージエンジンに適用されるインストールは、すべてのストレージエンジンに実装されていないことがあります。各サードパーティーエンジンのインストールメンテーションはエンジン管理者の責任です。[セクション 27.20「パフォーマンススキーマの制約」](#) も参照してください。
- データコレクションは、サーバーソースコードを変更し、インストールメンテーションを追加することによって実装されます。レプリケーションやイベントスケジューラなどのほかの機能と異なり、パフォーマンススキーマに関連付けられた個別のスレッドはありません。

パフォーマンススキーマは、サーバーパフォーマンスに与える影響を最小にしながら、サーバー実行に関する有益な情報へのアクセスを提供することを目的としています。実装はこれらの設計目標に従います。

- パフォーマンススキーマのアクティブ化によって、サーバーの動作は変更されません。たとえば、それによってスレッドスケジューリングが変更されず、クエリー実行計画 ([EXPLAIN](#) によって表示される) が変更されません。
- サーバーのモニタリングはごくわずかなオーバーヘッドで、継続的かつ目立たずに行われます。パフォーマンススキーマのアクティブ化によって、サーバーが使用不能になりません。
- パーサーは変更されません。新しいキーワードやステートメントはありません。
- パフォーマンススキーマが内部で失敗しても、サーバーコードの実行は正常に続行されます。

- 初期のイベントの収集時に処理を実行するか、あとのイベント取得時に処理を実行するかのどちらかを選択する場合、収集が速くなるほうが優先されます。これは、取得がオンデマンドで、まったく行われなくてもあるのに対し、収集は進行中であるためです。
- 「最もパフォーマンスの高いスキーマ」テーブルには、全テーブルスキャン以外の実行計画へのアクセス権をオプティマイザに付与するインデックスがあります。詳細は、[セクション8.2.4「パフォーマンススキーマクエリーの最適化」](#)を参照してください。
- 新しいインストゥルメンテーションポイントを追加することは簡単です。
- インストゥルメンテーションはバージョン管理されます。インストゥルメンテーションの実装が変更されても、以前にインストゥルメントされたコードは引き続き機能します。これは、最新のパフォーマンススキーマの変更と同期させておくために、各プラグインをアップグレードする必要がないため、サードパーティープラグインの開発者にメリットがあります。

注記

MySQL `sys` スキーマは、パフォーマンススキーマによって収集されたデータへの便利なアクセスを提供する一連のオブジェクトです。`sys` スキーマはデフォルトでインストールされます。使用手順については、[第28章「MySQL sys スキーマ」](#)を参照してください。

27.1 パフォーマンススキーマクイックスタート

このセクションでは、その使用方法を示す例によって、パフォーマンススキーマについて簡単に紹介します。追加の例については、[セクション27.19「問題を診断するためのパフォーマンススキーマの使用」](#)を参照してください。

パフォーマンススキーマはデフォルトで有効になっています。それを明示的に有効または無効にするには、`performance_schema` 変数を適切な値に設定して、サーバーを起動します。たとえば、サーバー `my.cnf` ファイルで次の行を使用します:

```
[mysqld]
performance_schema=ON
```

サーバーは起動すると、`performance_schema` を確認し、パフォーマンススキーマの初期化を試みます。初期化の成功を確認するには、このステートメントを使用します。

```
mysql> SHOW VARIABLES LIKE 'performance_schema';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| performance_schema | ON   |
+-----+-----+
```

`ON` の値はパフォーマンススキーマが正常に初期化され、使用する準備ができていることを意味します。`OFF` の値は何かのエラーが発生していることを意味します。何に異常が発生したかに関する情報については、サーバーエラーログをチェックしてください。

パフォーマンススキーマはストレージエンジンとして実装されるため、`INFORMATION_SCHEMA.ENGINES` テーブルまたは `SHOW ENGINES` ステートメントからの出力に一覧表示されます:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.ENGINES
      WHERE ENGINE='PERFORMANCE_SCHEMA'G
***** 1. row *****
ENGINE: PERFORMANCE_SCHEMA
SUPPORT: YES
COMMENT: Performance Schema
TRANSACTIONS: NO
XA: NO
SAVEPOINTS: NO

mysql> SHOW ENGINESG
...
Engine: PERFORMANCE_SCHEMA
Support: YES
Comment: Performance Schema
```

```
Transactions: NO
XA: NO
Savepoints: NO
...
```

PERFORMANCE_SCHEMA ストレージエンジンは、**performance_schema** データベース内のテーブルを操作します。そのテーブルへの参照をデータベース名で修飾する必要がないように、**performance_schema** をデフォルトのデータベースにすることができます。

```
mysql> USE performance_schema;
```

パフォーマンススキーマテーブルは **performance_schema** データベースに格納されます。このデータベースとそのテーブルの構造に関する情報を取得するには、ほかのすべてのデータベースのように、**INFORMATION_SCHEMA** データベースから選択するか、**SHOW** ステートメントを使用します。たとえば、どのパフォーマンススキーマテーブルが存在するか確認するには、これらのいずれかのステートメントを使用します。

```
mysql> SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'performance_schema';
```

```
+-----+
| TABLE_NAME |
+-----+
| accounts |
| cond_instances |
| ... |
| events_stages_current |
| events_stages_history |
| events_stages_history_long |
| events_stages_summary_by_account_by_event_name |
| events_stages_summary_by_host_by_event_name |
| events_stages_summary_by_thread_by_event_name |
| events_stages_summary_by_user_by_event_name |
| events_stages_summary_global_by_event_name |
| events_statements_current |
| events_statements_history |
| events_statements_history_long |
| ... |
| file_instances |
| file_summary_by_event_name |
| file_summary_by_instance |
| host_cache |
| hosts |
| memory_summary_by_account_by_event_name |
| memory_summary_by_host_by_event_name |
| memory_summary_by_thread_by_event_name |
| memory_summary_by_user_by_event_name |
| memory_summary_global_by_event_name |
| metadata_locks |
| mutex_instances |
| objects_summary_global_by_type |
| performance_timers |
| replication_connection_configuration |
| replication_connection_status |
| replication_applier_configuration |
| replication_applier_status |
| replication_applier_status_by_coordinator |
| replication_applier_status_by_worker |
| rlock_instances |
| session_account_connect_attrs |
| session_connect_attrs |
| setup_actors |
| setup_consumers |
| setup_instruments |
| setup_objects |
| socket_instances |
| socket_summary_by_event_name |
| socket_summary_by_instance |
| table_handles |
| table_io_waits_summary_by_index_usage |
| table_io_waits_summary_by_table |
| table_lock_waits_summary_by_table |
| threads |
| users |
+-----+
```



```
mysql> SHOW TABLES FROM performance_schema;
+-----+
| Tables_in_performance_schema |
+-----+
| accounts                      |
| cond_instances                |
| events_stages_current         |
| events_stages_history         |
| events_stages_history_long    |
| ...                           |
```

「パフォーマンススキーマ」テーブルの数は、追加のインストゥルメンテーションの実装が進行するにつれて時間の経過とともに増加します。

`performance_schema` データベースの名前は小文字で、その中のテーブルの名前も同様です。クエリーでは名前を小文字で指定してください。

個々のテーブルの構造を表示するには、`SHOW CREATE TABLE` を使用します。

```
mysql> SHOW CREATE TABLE performance_schema.setup_consumers\G
***** 1. row *****
      Table: setup_consumers
Create Table: CREATE TABLE `setup_consumers` (
  `NAME` varchar(64) NOT NULL,
  `ENABLED` enum('YES','NO') NOT NULL,
  PRIMARY KEY (`NAME`)
) ENGINE=PERFORMANCE_SCHEMA DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

テーブル構造は、`INFORMATION_SCHEMA.COLUMNS` などのテーブルから選択するか、`SHOW COLUMNS` などのステートメントを使用して取得することもできます。

`performance_schema` データベース内のテーブルはそれらの中の情報の種類 (現在のイベント、イベント履歴およびサマリー、オブジェクトインスタンス、およびセットアップ (構成) 情報) に従ってグループ化できます。次の例に、これらのテーブルのいくつかの使用法を示します。各グループのテーブルに関する詳細については、[セクション 27.12 「パフォーマンススキーマテーブルの説明」](#) を参照してください。

最初に、すべてのインストゥルメントとコンシューマが有効にされていないため、パフォーマンススキーマはすべてのイベントを収集しません。これらのすべてをオンにし、イベントタイミングを有効にするには、2つのステートメントを実行します (行のカウントはMySQLバージョンによって異なることがあります)。

```
mysql> UPDATE performance_schema.setup_instruments
      SET ENABLED = 'YES', TIMED = 'YES';
Query OK, 560 rows affected (0.04 sec)
mysql> UPDATE performance_schema.setup_consumers
      SET ENABLED = 'YES';
Query OK, 10 rows affected (0.00 sec)
```

現在サーバーが何を行なっているかを確認するには、`events_waits_current` テーブルを調査します。それには、スレッドごとに、各スレッドの最新のモニターされたイベントを示す1行が含まれます。

```
mysql> SELECT *
      FROM performance_schema.events_waits_current\G
***** 1. row *****
      THREAD_ID: 0
      EVENT_ID: 5523
      END_EVENT_ID: 5523
      EVENT_NAME: wait/synch/mutex/mysys/THR_LOCK::mutex
      SOURCE: thr_lock.c:525
      TIMER_START: 201660494489586
      TIMER_END: 201660494576112
      TIMER_WAIT: 86526
      SPINS: NULL
      OBJECT_SCHEMA: NULL
      OBJECT_NAME: NULL
      INDEX_NAME: NULL
      OBJECT_TYPE: NULL
      OBJECT_INSTANCE_BEGIN: 142270668
      NESTING_EVENT_ID: NULL
      NESTING_EVENT_TYPE: NULL
```

```
OPERATION: lock
NUMBER_OF_BYTES: NULL
FLAGS: 0
...
```

このイベントは、スレッド 0 が `THR_LOCK::mutex` のロック、`mysys` サブシステム内の相互排他ロックを獲得するために、86,526 ピコ秒待機していたことを示しています。最初のいくつかのカラムは次の情報を提供します。

- ID カラムはイベントの発生元のスレッドとイベント番号を示します。
- `EVENT_NAME` はインストゥルメントされたものを示し、`SOURCE` は、インストゥルメントされたコードを含むソースファイルを示します。
- タイマーカラムはイベントが開始および停止したタイミングとそれにかかった時間を示します。イベントがまだ進行中の場合は、`TIMER_END` と `TIMER_WAIT` の値が `NULL` になります。タイマー値は概算で、ピコ秒で表されます。タイマーおよびイベント時間コレクションについては、[セクション 27.4.1 「パフォーマンススキーマイベント タイミング」](#) を参照してください。

履歴テーブルには、現在のイベントテーブルと同じ種類の行が含まれますが、ほかの行もあり、サーバーが「現在」ではなく、「最近」何を実行していたかが示されます。`events_waits_history` および `events_waits_history_long` テーブルにはスレッドごとに最新の 10 イベントと最新の 10,000 イベントがそれぞれ含まれます。たとえば、スレッド 13 によって生成された最新イベントの情報を表示するには、次を実行します。

```
mysql> SELECT EVENT_ID, EVENT_NAME, TIMER_WAIT
FROM performance_schema.events_waits_history
WHERE THREAD_ID = 13
ORDER BY EVENT_ID;
+-----+-----+-----+
| EVENT_ID | EVENT_NAME                               | TIMER_WAIT |
+-----+-----+-----+
| 86 | wait/synch/mutex/mysys/THR_LOCK::mutex | 686322 |
| 87 | wait/synch/mutex/mysys/THR_LOCK_malloc | 320535 |
| 88 | wait/synch/mutex/mysys/THR_LOCK_malloc | 339390 |
| 89 | wait/synch/mutex/mysys/THR_LOCK_malloc | 377100 |
| 90 | wait/synch/mutex/sql/LOCK_plugin        | 614673 |
| 91 | wait/synch/mutex/sql/LOCK_open         | 659925 |
| 92 | wait/synch/mutex/sql/THD::LOCK_thd_data | 494001 |
| 93 | wait/synch/mutex/mysys/THR_LOCK_malloc | 222489 |
| 94 | wait/synch/mutex/mysys/THR_LOCK_malloc | 214947 |
| 95 | wait/synch/mutex/mysys/LOCK_alarm      | 312993 |
+-----+-----+-----+
```

履歴テーブルに新しいイベントが追加されると、テーブルがいっぱいである場合、古いイベントが破棄されます。

サマリーテーブルは、時間をかけてすべてのイベントについて集計された情報を提供します。このグループのテーブルには、さまざまな方法で、イベントデータが要約されます。もっとも多くの回数実行されたか、またはもっとも待機時間がかったインストゥルメントを確認するには、`COUNT_STAR` または `SUM_TIMER_WAIT` カラムで `events_waits_summary_global_by_event_name` テーブルをソートします。これらのカラムはすべてのイベント全体で計算された、`COUNT(*)` または `SUM(TIMER_WAIT)` 値にそれぞれ対応します。

```
mysql> SELECT EVENT_NAME, COUNT_STAR
FROM performance_schema.events_waits_summary_global_by_event_name
ORDER BY COUNT_STAR DESC LIMIT 10;
+-----+-----+
| EVENT_NAME                               | COUNT_STAR |
+-----+-----+
| wait/synch/mutex/mysys/THR_LOCK_malloc   | 6419 |
| wait/io/file/sql/FRM                     | 452 |
| wait/synch/mutex/sql/LOCK_plugin         | 337 |
| wait/synch/mutex/mysys/THR_LOCK_open     | 187 |
| wait/synch/mutex/mysys/LOCK_alarm        | 147 |
| wait/synch/mutex/sql/THD::LOCK_thd_data  | 115 |
| wait/io/file/myisam/kfile                | 102 |
| wait/synch/mutex/sql/LOCK_global_system_variables | 89 |
| wait/synch/mutex/mysys/THR_LOCK::mutex   | 89 |
| wait/synch/mutex/sql/LOCK_open          | 88 |
+-----+-----+

mysql> SELECT EVENT_NAME, SUM_TIMER_WAIT
```

```
FROM performance_schema.events_waits_summary_global_by_event_name
ORDER BY SUM_TIMER_WAIT DESC LIMIT 10;
```

EVENT_NAME	SUM_TIMER_WAIT
wait/io/file/sql/MYSQL_LOG	1599816582
wait/synch/mutex/mysys/THR_LOCK_malloc	1530083250
wait/io/file/sql/binlog_index	1385291934
wait/io/file/sql/FRM	1292823243
wait/io/file/myisam/kfile	411193611
wait/io/file/myisam/dfile	322401645
wait/synch/mutex/mysys/LOCK_alarm	145126935
wait/io/file/sql/casetest	104324715
wait/synch/mutex/sql/LOCK_plugin	86027823
wait/io/file/sql/pid	72591750

これらの結果には、[THR_LOCK_malloc](#) 相互排他ロックが、その使用される頻度とスレッドがそれを獲得しようとして待機する時間の量の両方に関して、「ホット」であることが示されます。

注記

[THR_LOCK_malloc](#) 相互排他ロックはデバッグビルドでのみ使用されます。本番ビルドでは、それが存在しないため、ホットではありません。

インスタンステーブルは、インストールされたオブジェクトの種類を記述します。インストールされたオブジェクトは、サーバーによって使われると、イベントを生成します。これらのテーブルは、イベント名と説明のメモまたはステータス情報を提供します。たとえば、[file_instances](#) テーブルは、ファイル I/O 操作のインストールのインスタンスとそれらに関連付けられたファイルを一覧表示します。

```
mysql> SELECT *
FROM performance_schema.file_instances\G
***** 1. row *****
FILE_NAME: /opt/mysql-log/60500/binlog.000007
EVENT_NAME: wait/io/file/sql/binlog
OPEN_COUNT: 0
***** 2. row *****
FILE_NAME: /opt/mysql/60500/data/mysql/tables_priv.MYI
EVENT_NAME: wait/io/file/myisam/kfile
OPEN_COUNT: 1
***** 3. row *****
FILE_NAME: /opt/mysql/60500/data/mysql/columns_priv.MYI
EVENT_NAME: wait/io/file/myisam/kfile
OPEN_COUNT: 1
...
```

セットアップテーブルは、モニタリング特性の構成と表示に使われます。たとえば、[setup_instruments](#) では、イベントを収集できるインストールのセットがリストされ、有効になっているインストールが表示されます：

```
mysql> SELECT NAME, ENABLED, TIMED
FROM performance_schema.setup_instruments;
+-----+-----+-----+
| NAME                                | ENABLED | TIMED |
+-----+-----+-----+
...
| stage/sql/end                       | NO      | NO    |
| stage/sql/executing                 | NO      | NO    |
| stage/sql/init                      | NO      | NO    |
| stage/sql/insert                    | NO      | NO    |
...
| statement/sql/load                  | YES     | YES   |
| statement/sql/grant                 | YES     | YES   |
| statement/sql/check                 | YES     | YES   |
| statement/sql/flush                 | YES     | YES   |
...
| wait/synch/mutex/sql/LOCK_global_read_lock | YES | YES |
| wait/synch/mutex/sql/LOCK_global_system_variables | YES | YES |
| wait/synch/mutex/sql/LOCK_lock_db | YES | YES |
| wait/synch/mutex/sql/LOCK_manager | YES | YES |
...
| wait/synch/rwlock/sql/LOCK_grant | YES | YES |
```

```

|wait/synch/rwlock/sql/LOGGER::LOCK_logger      |YES |YES |
|wait/synch/rwlock/sql/LOCK_sys_init_connect    |YES |YES |
|wait/synch/rwlock/sql/LOCK_sys_init_slave      |YES |YES |
...
|wait/io/file/sql/binlog                        |YES |YES |
|wait/io/file/sql/binlog_index                  |YES |YES |
|wait/io/file/sql/casetest                      |YES |YES |
|wait/io/file/sql/dbopt                        |YES |YES |
...

```

インストゥルメント名の解釈方法を理解するには、[セクション27.6「パフォーマンススキーマインストゥルメント命名規則」](#)を参照してください。

インストゥルメントのイベントを収集するかどうかを制御するには、その **ENABLED** 値を **YES** または **NO** に設定します。例:

```

mysql> UPDATE performance_schema.setup_instruments
      SET ENABLED = 'NO'
      WHERE NAME = 'wait/synch/mutex/sql/LOCK_mysql_create_db';

```

パフォーマンススキーマは収集されたイベントを使用して、イベント情報の「コンシューマ」として機能する `performance_schema` データベース内のテーブルを更新します。 `setup_consumers` テーブルは、使用可能なコンシューマとどれが有効にされているかを示します。

```
mysql> SELECT * FROM performance_schema.setup_consumers;
```

NAME	ENABLED
events_stages_current	NO
events_stages_history	NO
events_stages_history_long	NO
events_statements_current	YES
events_statements_history	YES
events_statements_history_long	NO
events_transactions_current	YES
events_transactions_history	YES
events_transactions_history_long	NO
events_waits_current	NO
events_waits_history	NO
events_waits_history_long	NO
global_instrumentation	YES
thread_instrumentation	YES
statements_digest	YES

パフォーマンススキーマがコンシューマをイベント情報の宛先として保守するかどうかを制御するには、その **ENABLED** 値を設定します。

セットアップテーブルについてとそれらを使用して、イベント収集を制御する詳細については、[セクション27.4.2「パフォーマンススキーマイベントフィルタリング」](#)を参照してください。

先述のグループのいずれにも分類されないその他のテーブルがいくつかあります。たとえば、`performance_timers` は、使用可能なイベントタイマーとそれらの特性を一覧表示します。タイマーの詳細については、[セクション27.4.1「パフォーマンススキーマイベントタイミング」](#)を参照してください。

27.2 パフォーマンススキーマビルド構成

パフォーマンススキーマは必須であり、常にコンパイルされます。パフォーマンススキーマインストゥルメンテーションの特定の部分を除外できます。たとえば、ステージインストゥルメンテーションとステートメントインストゥルメンテーションを除外するには、次のようにします:

```

shell> cmake . \
      -DDISABLE_PSI_STAGE=1 \
      -DDISABLE_PSI_STATEMENT=1

```

詳細は、[セクション2.9.7「MySQL ソース構成オプション」](#)の `DISABLE_PSI_XXX` CMake オプションの説明を参照してください。

パフォーマンススキーマを使用せずに構成された以前のインストール (またはテーブルが欠落しているか古い古いバージョンのパフォーマンススキーマを使用して) に MySQL をインストールする場合。この問題は、エラーログに次のようなメッセージが存在することを示します:

```
[ERROR] Native table 'performance_schema'.events_waits_history'
has the wrong structure
[ERROR] Native table 'performance_schema'.events_waits_history_long'
has the wrong structure
...
```

この問題を修正するには、MySQL のアップグレード手順を実行します。 [セクション2.11「MySQL のアップグレード」](#) を参照してください。

パフォーマンススキーマは構築時にサーバーに構成されるため、[SHOW ENGINES](#) からの出力に [PERFORMANCE_SCHEMA](#) の行が表示されます。これは、パフォーマンススキーマが有効ではなく、使用可能であることを意味します。それを有効にするには、次のセクションで説明するように、サーバーの起動時にそうする必要があります。

27.3 パフォーマンススキーマ起動構成

MySQL パフォーマンススキーマを使用するには、イベント収集を有効にするために、サーバーの起動時に有効にする必要があります。

パフォーマンススキーマはデフォルトで有効になっています。それを明示的に有効または無効にするには、[performance_schema](#) 変数を適切な値に設定して、サーバーを起動します。たとえば、サーバー `my.cnf` ファイルで次の行を使用します:

```
[mysqld]
performance_schema=ON
```

パフォーマンススキーマの初期化時に、サーバーが内部バッファを割り当てることができない場合、パフォーマンススキーマは自動的に無効になり、[performance_schema](#) を `OFF` に設定して、サーバーがインストールメンターションなしで実行します。

パフォーマンススキーマでは、サーバーの起動時にインストールメントおよびコンシューマ構成も許可されます。

サーバー起動時のインストールメントを制御するには、この形式のオプションを使用します。

```
--performance-schema-instrument='instrument_name=value'
```

ここで `instrument_name` は `wait/synch/mutex/sql/LOCK_open` などのインストールメント名で、`value` はこれらのいずれかの値です。

- `OFF`、`FALSE` または `0`: インストールメントの無効化
- `ON`、`TRUE` または `1`: インストールメントの有効化および時間
- `COUNTED`: インストールメントを有効にして (時間ではなく) カウント

各 `--performance-schema-instrument` オプションではインストールメント名を1つしか指定できませんが、オプションの複数のインスタンスを指定して、複数のインストールメントを構成できます。さらに、インストールメント名にパターンを使用でき、パターンに一致するインストールメントを構成します。すべての条件同期インストールメントを有効で、カウント対象として構成するには、次のオプションを使用します。

```
--performance-schema-instrument='wait/synch/cond/%=COUNTED'
```

すべてのインストールメントを無効にするには、次のオプションを使用します。

```
--performance-schema-instrument='%=OFF'
```

例外: `memory/performance_schema/%` インストールメントは組み込まれており、起動時に無効にすることはできません。

長いインストゥルメント名文字列は、順序に関係なく、短いパターン名より優先されます。インストゥルメントを選択するためのパターンの指定については、[セクション27.4.9「フィルタリング操作のインストゥルメントまたはコンシューマの指定」](#)を参照してください。

認識されないインストゥルメント名は無視されます。あとでインストールされたプラグインによってインストゥルメントを作成することは可能で、そのときに名前が認識され、構成されます。

サーバー起動時のコンシューマを制御するには、この形式のオプションを使用します。

```
--performance-schema-consumer-consumer_name=value
```

ここで、`consumer_name` は `events_waits_history` などのコンシューマ名で、`value` はこれらのいずれかです。

- `OFF`、`FALSE` または `0`: コンシューマのイベントを収集しない
- `ON`、`TRUE` または `1`: コンシューマのイベントの収集

たとえば、`events_waits_history` コンシューマを有効にするには、次のオプションを使用します。

```
--performance-schema-consumer-events-waits-history=ON
```

許可されるコンシューマ名は、`setup_consumers` テーブルを調べるとわかります。パターンは許可されません。`setup_consumers` テーブル内のコンシューマ名は下線が使われますが、起動時に設定されたコンシューマでは、名前の中のダッシュと下線は同等です。

パフォーマンススキーマには、構成情報を提供するいくつかのシステム変数が含まれます。

```
mysql> SHOW VARIABLES LIKE 'perf%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| performance_schema | ON |
| performance_schema_accounts_size | 100 |
| performance_schema_digests_size | 200 |
| performance_schema_events_stages_history_long_size | 10000 |
| performance_schema_events_stages_history_size | 10 |
| performance_schema_events_statements_history_long_size | 10000 |
| performance_schema_events_statements_history_size | 10 |
| performance_schema_events_waits_history_long_size | 10000 |
| performance_schema_events_waits_history_size | 10 |
| performance_schema_hosts_size | 100 |
| performance_schema_max_cond_classes | 80 |
| performance_schema_max_cond_instances | 1000 |
| ...
```

`performance_schema` 変数は `ON` または `OFF` で、パフォーマンススキーマが有効か無効かを示します。ほかの変数はテーブルサイズ (行数) やメモリー割り当て値を示します。

注記

パフォーマンススキーマが有効にされている場合、パフォーマンススキーマインスタンスの数は、おそらく大きくサーバーメモリーフットプリントに影響します。パフォーマンススキーマは、必要なだけメモリーを使用するように多くのパラメータを自動スケールリングします。[セクション27.17「パフォーマンススキーマのメモリー割り当てモデル」](#)を参照してください。

パフォーマンススキーマシステム変数の値を変更するには、それらをサーバー起動時に設定します。たとえば、待機イベントの履歴テーブルのサイズを変更するには、`my.cnf` ファイルに次の行を挿入します:

```
[mysqld]
performance_schema
performance_schema_events_waits_history_size=20
performance_schema_events_waits_history_long_size=15000
```

パフォーマンススキーマは、明示的に設定されていない場合、サーバーの起動時にいくつかのパラメータの値のサイズを自動的に設定します。たとえば、それはイベント待機テーブルのサイズを制御するパラメータをこのようにサイズ設定します。パフォーマンススキーマは、サーバーの起動時に必要なすべてのメモリーを割り当てるのではな

く、メモリー使用量を実際のサーバー負荷に合わせて増分的に割り当てます。したがって、多くのサイズ設定パラメータを設定する必要はありません。自動サイズ設定または自動スケール設定されているパラメータを確認するには、`mysqld --verbose --help` を使用してオプションの説明を調べるか、[セクション27.15「パフォーマンススキーマシステム変数」](#) を参照してください。

サーバーの起動時に設定されない自動サイズ設定されたパラメータごとに、パフォーマンススキーマは次のシステム値の値に基づいてその値を設定する方法を決定します。これらの値は、MySQL サーバーの構成方法について「[ヒント](#)」とみなされます：

```
max_connections
open_files_limit
table_definition_cache
table_open_cache
```

特定のパラメータの自動サイズ設定または自動スケーリングをオーバーライドするには、起動時に-1以外の値に設定します。この場合、パフォーマンススキーマはそれに指定された値を割り当てます。

実行時に、`SHOW VARIABLES` では、自動サイズ設定されたパラメータに設定されていた実際の値が表示されます。自動スケールされたパラメータは-1の値で表示されます。

パフォーマンススキーマが無効になっている場合、その自動サイズ変更されたパラメータと自動スケール変更されたパラメータは-1のまま、`SHOW VARIABLES` には-1と表示されます。

27.4 パフォーマンススキーマ実行時構成

実行時に特定のパフォーマンススキーマ機能を有効にして、発生するイベント収集のタイプを制御できます。

パフォーマンススキーマセットアップテーブルには、モニタリング構成に関する情報が含まれます。

```
mysql> SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
        WHERE TABLE_SCHEMA = 'performance_schema'
        AND TABLE_NAME LIKE 'setup%';
+-----+
| TABLE_NAME |
+-----+
| setup_actors |
| setup_consumers |
| setup_instruments |
| setup_objects |
| setup_threads |
+-----+
```

これらのテーブルの内容を調査して、パフォーマンススキーマのモニタリング特性に関する情報を取得できます。`UPDATE` 権限を持っている場合は、セットアップテーブルを変更して、モニタリングが行われる方法に影響するパフォーマンススキーマ操作を変更できます。これらのテーブルの追加の詳細については、[セクション27.12.2「パフォーマンススキーマセットアップテーブル」](#) を参照してください。

`setup_instruments` および `setup_consumers` テーブルは、イベントを収集できるインストゥルメントと、イベント情報が実際に収集されるコンシューマの種類をそれぞれ一覧表示します。その他のセットアップテーブルにより、モニタリング構成をさらに変更できます。[セクション27.4.2「パフォーマンススキーマイベントフィルタリング」](#) では、イベントコレクションに影響するように、これらのテーブルをどのように変更できるかについて説明しています。

実行時に SQL ステートメントを使用して行う必要があるパフォーマンススキーマ構成の変更があり、サーバーが起動するたびにこれらの変更を有効にする場合は、ステートメントをファイルに配置し、ファイルに名前を付けるように設定された `init_file` システム変数を使用してサーバーを起動します。この戦略は、簡易サーバーヘルスモニタリング、インシデント調査、アプリケーション動作のトラブルシューティングなど、さまざまな種類のモニタリングを生成するようにそれぞれカスタマイズされている複数のモニタリング構成がある場合にも役に立つ可能性があります。各監視構成のステートメントを独自のファイルに配置し、サーバーの起動時に `init_file` 値として適切なファイルを指定します。

27.4.1 パフォーマンススキーマイベントタイミング

イベントは、サーバーソースコードに追加されたインストゥルメンテーションを使用して収集されます。インストゥルメントはイベントの時間を測定しますが、これはパフォーマンススキーマがイベントにどれくらいの時間がかかる

かを知らせる方法です。タイミング情報を収集しないようにインストゥルメントを構成することもできます。このセクションでは、使用可能なタイマーとそれらの特性、およびイベント内のタイミング値を表す方法について説明します。

パフォーマンススキーマタイマー

パフォーマンススキーマタイマーは、精度とオーバーヘッドの量が異なります。使用可能なタイマーとそれらの特性を確認するには、`performance_timers` テーブルをチェックしてください。

```
mysql> SELECT * FROM performance_schema.performance_timers;
+-----+-----+-----+-----+
| TIMER_NAME | TIMER_FREQUENCY | TIMER_RESOLUTION | TIMER_OVERHEAD |
+-----+-----+-----+-----+
| CYCLE      | 2389029850      | 1                | 72              |
| NANOSECOND | 1000000000      | 1                | 112             |
| MICROSECOND | 1000000         | 1                | 136             |
| MILLISECOND | 1036            | 1                | 168             |
+-----+-----+-----+-----+
```

特定のタイマーに関連付けられた値が `NULL` の場合、そのタイマーはプラットフォームでサポートされていません。

カラムの意味は次のとおりです:

- `TIMER_NAME` カラムには使用可能なタイマーの名前が表示されます。 `CYCLE` は CPU (プロセッサ) サイクルカウンタに基づいたタイマーを表します。
- `TIMER_FREQUENCY` は、1 秒あたりのタイマー単位数を示します。 サイクルタイマーの場合、頻度は一般に CPU 速度に関連します。 示された値は、2.4GHz プロセッサを搭載するシステムで取得されました。 ほかのタイマーは固定の数秒に基づきます。
- `TIMER_RESOLUTION` は、タイマー値が一度に増加するタイマー単位数を示します。 タイマーの分解能が 10 の場合、その値は毎回 10 ずつ増加します。
- `TIMER_OVERHEAD` は特定のタイマーで 1 つのタイミングを取得するためのオーバーヘッドの最小サイクル数です。 タイマーはイベントの開始と終了で呼び出されるため、イベントあたりのオーバーヘッドは表示される値の 2 倍になります。

パフォーマンススキーマは、次のようにタイマーを割り当てます:

- 待機タイマーは `CYCLE` を使用します。
- アイドルタイマー、ステージタイマー、ステートメントタイマーおよびトランザクションタイマーは、`NANOSECOND` タイマーが使用可能なプラットフォームでは `NANOSECOND` を使用し、それ以外の場合は `MICROSECOND` を使用します。

サーバーの起動時に、パフォーマンススキーマは構築時にタイマーの割り当てに関する仮定が正しいことを検証し、タイマーが使用できない場合は警告を表示します。

待機イベントを時間するには、タイマーの精度を犠牲にしてオーバーヘッドを削減することが最も重要な基準であるため、`CYCLE` タイマーを使用することをお勧めします。

ステートメント (またはステージ) の実行にかかる時間は、一般に単一の待機の実行にかかる時間より桁違いに大きくなります。ステートメントの時間を測定するために、もっとも重要な基準は、プロセッサの周波数の変更に影響を受けない正確な測定基準を設定することであるため、サイクルに基づいていないタイマーを使用することがもっとも適切です。ステートメントのデフォルトのタイマーは `NANOSECOND` です。 `CYCLE` タイマーと比較すると、余分な「オーバーヘッド」は重要ではありません。これは、タイマーのコールによって発生するオーバーヘッド (ステートメントの開始時に一度、終了時に一度) が、ステートメント自体の実行に使用される CPU 時間と比較して大幅に低いからです。 `CYCLE` タイマーを使用することにはメリットがなく、欠点だけです。

サイクルカウンタによって提供される精度はプロセッサ速度によって異なります。プロセッサが 1 GHz (10 億サイクル/秒) 以上で実行する場合、サイクルカウンタはナノ秒未満の精度を実現します。サイクルカウンタを使用することは、実際の時間を取得するより、はるかに負荷が小さくなります。たとえば、標準 `gettimeofday()` 関数は数百サイクルかかる可能性があり、これは 1 秒あたり数千または数百万回発生する可能性のあるデータ収集では、許容できないオーバーヘッドです。

サイクルカウンタには欠点もあります。

- エンドユーザーは、何分の 1 秒などの時計の単位でタイミングを知ることを期待します。サイクルから秒に変換することは大きな負荷がかかる可能性があります。このため、変換はすばやいかなり概算的な乗算演算です。
- ラップトップが節電モードになったときや熱の生成を抑えるために、CPU の速度が低下したときなどに、プロセッサのサイクルレートが変わることがあります。プロセッサのサイクルレートが変動した場合、サイクルから実時間単位への変換でエラーが発生することがあります。
- サイクルカウンタは、プロセッサやオペレーティングシステムによって、信頼できない場合や使用できない場合があります。たとえば、Pentium では、命令は `RDTSC` (C 命令ではなくアセンブリ言語) であり、理論上オペレーティングシステムはユーザーモードプログラムのその使用を妨げることができます。
- 異常実行またはマルチプロセッサ同期に関する一部のプロセッサの詳細により、カウンタが最大 1000 サイクルごとに高速になったり、低速になったりするように見えることがあります。

MySQL は、x386 (Windows、macOS、Linux、Solaris、その他の UNIX フレーバ)、PowerPC、IA-64 のサイクルカウンタで動作します。

イベントでのパフォーマンススキーマのタイマー表現

現在のイベントおよび履歴イベントを格納する「パフォーマンススキーマ」テーブルの行には、タイミング情報をテーブルの 3 つのカラムがあります: `TIMER_START` および `TIMER_END` はイベントがいつ開始および終了したかを示し、`TIMER_WAIT` はイベント期間を示します。

`setup_instruments` テーブルにはイベントを収集するインストゥルメントを示す `ENABLED` カラムがあります。このテーブルには、時間が測定されるインストゥルメントを示す `TIMED` カラムもあります。インストゥルメントが有効にされていない場合、イベントを生成しません。有効にされているインストゥルメントの時間が測定されない場合、インストゥルメントによって生成されたイベントの `TIMER_START`、`TIMER_END`、および `TIMER_WAIT` タイマー値が `NULL` になります。これにより、サマリーテーブルの集計時間値 (合計、最小、最大および平均) の計算時にこれらの値が無視されます。

内部的には、イベント内の時間は、イベントタイミングの開始時に有効なタイマーで指定された単位で格納されます。イベントが「パフォーマンススキーマ」テーブルから取得されたときに表示されるように、時間はピコ秒 (1 秒に 1 兆) で表示され、選択されているタイマーに関係なく標準単位に正規化されます。

サーバー起動時のパフォーマンススキーマの初期化で、タイマーベースライン (「時間ゼロ」) が発生します。イベント内の `TIMER_START` および `TIMER_END` 値はベースライン以降のピコ秒を表します。`TIMER_WAIT` 値はピコ秒での期間です。

イベント内のピコ秒値は概算です。それらの精度は、ある単位から別の単位への変換に伴う通常の誤差の形式に左右されます。`CYCLE` タイマーが使われ、プロセッサのレートがさまざまに異なる場合、ドリフトが発生することがあります。このため、サーバーの起動から経過した時間の正確な測定基準として、イベントの `TIMER_START` 値を見ることは妥当ではありません。一方、開始時間や期間によって、イベントを順序付けるために、`ORDER BY` 句に `TIMER_START` 値または `TIMER_WAIT` 値を使用することは適切です。

イベントでマイクロ秒などの値ではなく、ピコ秒を選択したことには、パフォーマンス上の根拠があります。1 つの実装目標は、タイマーに関係なく、統一された時間単位で結果を表示することでした。理想の世界では、この時間単位は時計の単位のように見え、適度に正確である、つまりマイクロ秒です。ただし、サイクルまたはナノ秒をマイクロ秒に変換するには、すべてのインストゥルメンテーションで除算を実行する必要がある場合があります。除算は多くのプラットフォームで高い負荷がかかります。乗算は負荷が高くないため、それを使用しています。そのため、時間単位は、大きな精度の損失がないように十分に大きな乗数を使用した、可能な限り最大の `TIMER_FREQUENCY` 値の整数の倍数です。その結果、時間単位が「ピコ秒」になります。この精度は疑似ですが、この決定によりオーバーヘッドを最小にすることができます。

`wait`、`stage`、`statement`、または `transaction` イベントの実行中、各 `current-event` テーブルには現在のイベントのタイミング情報が表示されます:

```
events_waits_current
events_stages_current
events_statements_current
events_transactions_current
```

完了していないイベントが実行されている期間を判別できるように、タイマーカラムは次のように設定されます:

- `TIMER_START` が移入されます。
- `TIMER_END` には、現在のタイマー値が移入されます。
- `TIMER_WAIT` には、これまでの経過時間 (`TIMER_END - TIMER_START`) が移入されます。

まだ完了していないイベントの `END_EVENT_ID` 値は `NULL` です。イベントについてこれまでに経過した時間を評価するには、`TIMER_WAIT` カラムを使用します。したがって、まだ完了しておらず、これまでに `N` ピコ秒より長くかかったイベントを識別するために、モニタリングアプリケーションはクエリーで次の式を使用できます:

```
WHERE END_EVENT_ID IS NULL AND TIMER_WAIT > N
```

前述のイベント識別では、対応するインストゥルメントの `ENABLED` および `TIMED` が `YES` に設定されており、関連するコンシューマが有効になっていることを前提としています。

27.4.2 パフォーマンススキーマイベントフィルタリング

イベントはプロデューサ/コンシューマ方式で処理されます。

- インストゥルメントされたコードは、イベントのソースで、収集されるイベントを生成します。 `setup_instruments` テーブルは、イベントを収集できるインストゥルメント、それらが有効にされているかどうか、および (有効にされているインストゥルメントの場合) タイミング情報を収集するかどうかを一覧表示します。

```
mysql> SELECT NAME, ENABLED, TIMED
       FROM performance_schema.setup_instruments;
+-----+-----+-----+
| NAME                                | ENABLED | TIMED |
+-----+-----+-----+
...
| wait/synch/mutex/sql/LOCK_global_read_lock | YES | YES |
| wait/synch/mutex/sql/LOCK_global_system_variables | YES | YES |
| wait/synch/mutex/sql/LOCK_lock_db | YES | YES |
| wait/synch/mutex/sql/LOCK_manager | YES | YES |
...
```

`setup_instruments` テーブルはイベント生成のもっとも基本的な制御の形式を提供します。モニターされるオブジェクトやスレッドの種類に基づいて、イベント生成をさらに絞り込むには、[セクション27.4.3「イベントの事前フィルタリング」](#)に説明するように、ほかのテーブルを使用できます。

- パフォーマンススキーマテーブルは、イベントの宛先で、イベントを消費します。 `setup_consumers` テーブルは、イベント情報を送信できるコンシューマの種類とそれらが有効にされているかどうかを一覧表示します。

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+-----+-----+
| NAME                                | ENABLED |
+-----+-----+
| events_stages_current | NO |
| events_stages_history | NO |
| events_stages_history_long | NO |
| events_statements_current | YES |
| events_statements_history | YES |
| events_statements_history_long | NO |
| events_transactions_current | YES |
| events_transactions_history | YES |
| events_transactions_history_long | NO |
| events_waits_current | NO |
| events_waits_history | NO |
| events_waits_history_long | NO |
| global_instrumentation | YES |
| thread_instrumentation | YES |
| statements_digest | YES |
+-----+-----+
```

フィルタリングはパフォーマンスモニタリングのさまざまなステージで実行できます。

- **事前フィルタリング。** これは、プロデューサから特定の種類のイベントのみが収集され、収集されたイベントが特定のコンシューマのみを更新するように、パフォーマンススキーマ構成を変更することによって行われます。こ

れを実行するには、インストゥルメントまたはコンシューマを有効または無効にします。事前フィルタリングは、パフォーマンススキーマによって行われ、すべてのユーザーに適用されるグローバルな効果を持ちます。

事前フィルタリングを使用する理由:

- オーバーヘッドを削減するため。パフォーマンススキーマのオーバーヘッドは、すべてのインストゥルメントを有効にしても最小であるはずですが、さらに縮小したいと考える可能性があります。または、タイミングイベントに関心がなく、タイミングオーバーヘッドを解消するために、タイミングコードを無効にしたいと考えます。
- 関心のないイベントの現在のイベントまたは履歴テーブルへの入力を妨げるため。事前フィルタリングにより、これらのテーブル内に、有効なインストゥルメントの種類別のインスタンス用に多くの「空き」を残します。事前フィルタリングでファイルインストゥルメントのみを有効にしている場合、非ファイルインストゥルメントの行は収集されません。事後フィルタリングで、非ファイルイベントが収集され、ファイルイベントの少ない行が残されます。
- 特定の種類のイベントテーブルの保守を回避するため。コンシューマを無効にすると、サーバーはそのコンシューマの宛先の保守に時間を費やさなくなります。たとえば、イベント履歴に関心がいない場合、履歴テーブルコンシューマを無効にして、パフォーマンスを向上できます。
- 事後フィルタリング。これには、クエリー内で、パフォーマンススキーマテーブルから情報を選択する `WHERE` 句を使用して、使用可能なイベントのうち表示したいものを指定することが含まれます。事後フィルタリングは、各ユーザーが使用可能なイベントのうち関心のあるものを選択するため、ユーザー単位で実行されます。

事後フィルタリングを使用する理由:

- 関心のあるイベント情報に関して、個々のユーザーの決断を避けるため。
- 事前フィルタリングの使用に課せられる制限が前もってわからない場合に、パフォーマンススキーマを使用して、パフォーマンスの問題を調査するため。

次のセクションでは、事前フィルタリングの詳細を説明し、フィルタリング操作でインストゥルメントやコンシューマを指定するためのガイドラインを提供します。情報を取得するためのクエリーの書き方(事後フィルタリング)については、[セクション27.5「パフォーマンススキーマクエリー」](#)を参照してください。

27.4.3 イベントの事前フィルタリング

事前フィルタリングは、パフォーマンススキーマによって行われ、すべてのユーザーに適用されるグローバルな効果を持ちます。事前フィルタリングは、イベント処理のプロデューサまたはコンシューマステージに適用できます。

- プロデューサステージで事前フィルタリングを構成するには、いくつかのテーブルを使用できます。
 - `setup_instruments` は使用可能なインストゥルメントを示します。このテーブルで無効にされているインストゥルメントは、ほかの生成関連セットアップテーブルの内容に関係なく、イベントを生成しません。このテーブルで有効にされているインストゥルメントは、イベントの生成が許可され、ほかのテーブルの内容に依存します。
 - `setup_objects` は、パフォーマンススキーマが特定のテーブルおよびストアプログラムオブジェクトをモニターするかどうかを制御します。
 - `threads` スレッドは各サーバースレッドでモニタリングが有効にされているかどうかを示します。
 - `setup_actors` は、新しいフォアグラウンドスレッドの初期モニタリング状態を決定します。
- コンシューマステージで事前フィルタリングを構成するには、`setup_consumers` テーブルを変更します。これによって、イベントの送信先が決まります。`setup_consumers` はイベント生成にも暗黙的に影響します。指定されたイベントがどの宛先にも送信されない(つまり、消費されない)場合、パフォーマンススキーマはそれを生成しません。

これらのテーブルのいずれかを変更すると、すぐに監視に影響しますが、`setup_actors` テーブルを変更すると、変更後に作成されたフォアグラウンドスレッドにのみ影響し、既存のスレッドには影響しません。

モニタリング構成を変更すると、パフォーマンススキーマは履歴テーブルをフラッシュしません。すでに収集されたイベントは、新しいイベントによって置き換えられるまで、現在のイベントと履歴テーブルに残ります。インストゥ

ルメントを無効にする場合、それらのイベントが関心のある新しいイベントによって置き換えられるまで、しばらく待つ必要がある場合があります。または、[TRUNCATE TABLE](#) を使用して、履歴テーブルを空にします。

インストゥルメンテーションの変更後、サマリーテーブルを切り捨てる必要がある場合があります。一般に、サマリーカラムは行を削除するのではなく、0 または `NULL` にリセットされます。これにより、収集された値をクリアし、アグリゲーションを再開できます。それは、実行時構成の変更を行なったあとなどに便利な場合があります。この切捨て動作の例外は、個々のサマリーテーブルのセクションに記載されています。

次のセクションでは、特定のテーブルを使用して、パフォーマンススキーマの事前フィルタリングを制御する方法について説明します。

27.4.4 インストゥルメントによる事前フィルタリング

`setup_instruments` テーブルは使用可能なインストゥルメントを一覧表示します。

```
mysql> SELECT NAME, ENABLED, TIMED
        FROM performance_schema.setup_instruments;
+-----+-----+-----+
| NAME                                | ENABLED | TIMED |
+-----+-----+-----+
...
| stage/sql/end                       | NO      | NO    |
| stage/sql/executing                 | NO      | NO    |
| stage/sql/init                      | NO      | NO    |
| stage/sql/insert                    | NO      | NO    |
...
| statement/sql/load                  | YES     | YES   |
| statement/sql/grant                 | YES     | YES   |
| statement/sql/check                 | YES     | YES   |
| statement/sql/flush                 | YES     | YES   |
...
| wait/synch/mutex/sql/LOCK_global_read_lock | YES | YES |
| wait/synch/mutex/sql/LOCK_global_system_variables | YES | YES |
| wait/synch/mutex/sql/LOCK_lock_db | YES | YES |
| wait/synch/mutex/sql/LOCK_manager | YES | YES |
...
| wait/synch/rwlock/sql/LOCK_grant | YES | YES |
| wait/synch/rwlock/sql/LOGGER::LOCK_logger | YES | YES |
| wait/synch/rwlock/sql/LOCK_sys_init_connect | YES | YES |
| wait/synch/rwlock/sql/LOCK_sys_init_slave | YES | YES |
...
| wait/io/file/sql/binlog | YES | YES |
| wait/io/file/sql/binlog_index | YES | YES |
| wait/io/file/sql/casetest | YES | YES |
| wait/io/file/sql/dbopt | YES | YES |
...
```

インストゥルメントを有効にするかどうかを制御するには、その `ENABLED` カラムを `YES` または `NO` に設定します。有効にされたインストゥルメントのタイミング情報を収集するかどうかを構成するには、その `TIMED` 値を `YES` または `NO` に設定します。 `TIMED` カラムを設定すると、[セクション27.4.1「パフォーマンススキーマイベントタイミング」](#) に説明するように、パフォーマンススキーマテーブルの内容に影響します。

ほとんどの `setup_instruments` 行を変更すると、すぐに監視に影響します。一部のインストゥルメントでは、変更はサーバーの起動時にのみ有効です。実行時に変更しても効果はありません。これは主にサーバー内の `mutex`、条件、および `rwlocks` に影響しますが、これが当てはまるほかのインストゥルメントが存在する可能性があります。

`setup_instruments` テーブルはイベント生成のもっとも基本的な制御の形式を提供します。モニターされるオブジェクトやスレッドの種類に基づいて、イベント生成をさらに絞り込むには、[セクション27.4.3「イベントの事前フィルタリング」](#) に説明するように、ほかのテーブルを使用できます。

次の例に、`setup_instruments` テーブルへの可能な操作を示します。ほかの事前フィルタリング操作と同様に、これらの変更はすべてのユーザーに影響します。これらの一部のクエリーでは、`LIKE` 演算子とパターンマッチインストゥルメント名を使用しています。インストゥルメントを選択するためのパターンの指定に関する追加情報については、[セクション27.4.9「フィルタリング操作のインストゥルメントまたはコンシューマの指定」](#) を参照してください。

- すべてのインストゥルメントを無効にします。

```
UPDATE performance_schema.setup_instruments
```



```
SET ENABLED = 'NO';
```

これで、イベントは収集されません。

- すべてのインストゥルメントを無効にし、それらを現在の無効にされているインストゥルメントのセットに追加します。

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'NO'
WHERE NAME LIKE 'wait/io/file/%';
```

- ファイルインストゥルメントのみを無効にし、ほかのすべてのインストゥルメントを有効にします。

```
UPDATE performance_schema.setup_instruments
SET ENABLED = IF(NAME LIKE 'wait/io/file/%', 'NO', 'YES');
```

- [mysys](#) ライブラリ内のインストゥルメントを除くすべてのインストゥルメントを有効にします。

```
UPDATE performance_schema.setup_instruments
SET ENABLED = CASE WHEN NAME LIKE '%/mysys/%' THEN 'YES' ELSE 'NO' END;
```

- 特定のインストゥルメントを無効にします。

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'NO'
WHERE NAME = 'wait/synch/mutex/mysys/TMPDIR_mutex';
```

- インストゥルメントの状態を切り替えるには、その **ENABLED** 値を「反転」します。

```
UPDATE performance_schema.setup_instruments
SET ENABLED = IF(ENABLED = 'YES', 'NO', 'YES')
WHERE NAME = 'wait/synch/mutex/mysys/TMPDIR_mutex';
```

- すべてのイベントのタイミングを無効にします。

```
UPDATE performance_schema.setup_instruments
SET TIMED = 'NO';
```

27.4.5 オブジェクトによる事前フィルタリング

[setup_objects](#) テーブルは、パフォーマンススキーマが特定のテーブルおよびストアプログラムオブジェクトをモニターするかどうかを制御します。初期 [setup_objects](#) の内容は次のように見えます。

```
mysql> SELECT * FROM performance_schema.setup_objects;
+-----+-----+-----+-----+
| OBJECT_TYPE | OBJECT_SCHEMA | OBJECT_NAME | ENABLED | TIMED |
+-----+-----+-----+-----+
| EVENT      | mysql         | %           | NO      | NO    |
| EVENT      | performance_schema | %         | NO      | NO    |
| EVENT      | information_schema | %         | NO      | NO    |
| EVENT      | %             | %           | YES     | YES   |
| FUNCTION   | mysql         | %           | NO      | NO    |
| FUNCTION   | performance_schema | %         | NO      | NO    |
| FUNCTION   | information_schema | %         | NO      | NO    |
| FUNCTION   | %             | %           | YES     | YES   |
| PROCEDURE  | mysql         | %           | NO      | NO    |
| PROCEDURE  | performance_schema | %         | NO      | NO    |
| PROCEDURE  | information_schema | %         | NO      | NO    |
| PROCEDURE  | %             | %           | YES     | YES   |
| TABLE     | mysql         | %           | NO      | NO    |
| TABLE     | performance_schema | %         | NO      | NO    |
| TABLE     | information_schema | %         | NO      | NO    |
| TABLE     | %             | %           | YES     | YES   |
| TRIGGER    | mysql         | %           | NO      | NO    |
| TRIGGER    | performance_schema | %         | NO      | NO    |
| TRIGGER    | information_schema | %         | NO      | NO    |
| TRIGGER    | %             | %           | YES     | YES   |
+-----+-----+-----+-----+
```

[setup_objects](#) テーブルへの変更はただちにオブジェクトモニタリングに影響します。

OBJECT_TYPE カラムは行が適用されるオブジェクトの種類を示します。 **TABLE** フィルタリングはテーブル I/O イベント (`wait/io/table/sql/handler` インストゥルメント) およびテーブルロックイベント (`wait/lock/table/sql/handler` インストゥルメント) に影響します。

OBJECT_SCHEMA および **OBJECT_NAME** カラムには、リテラルスキーマまたはオブジェクト名、あるいは任意の名前に一致する '%' が含まれている必要があります。

ENABLED カラムは一致するオブジェクトがモニターされているかどうかを示し、**TIMED** はタイミング情報を収集するかどうかを示します。 **TIMED** カラムを設定すると、[セクション27.4.1「パフォーマンススキーマイベントタイミング」](#)に説明するように、パフォーマンススキーマテーブルの内容に影響します。

デフォルトのオブジェクト構成の効果は、`mysql`、`INFORMATION_SCHEMA` および `performance_schema` データベースのオブジェクトを除くすべてのオブジェクトをインストゥルメント処理することです。(`INFORMATION_SCHEMA` データベース内のテーブルは、`setup_objects` の内容に関係なくインストゥルメントされず、`information_schema.%` の行は単にこのデフォルトを明示します。)

パフォーマンススキーマは、`setup_objects` の一致をチェックする場合、まずより詳細な一致を見つけようとします。特定の **OBJECT_TYPE** に一致する行の場合、パフォーマンススキーマは次の順序で行をチェックします：

- **OBJECT_SCHEMA='literal'** および **OBJECT_NAME='literal'** を含む行。
- **OBJECT_SCHEMA='literal'** および **OBJECT_NAME='%'** を含む行。
- **OBJECT_SCHEMA='%'** および **OBJECT_NAME='%'** を含む行。

たとえば、テーブル `db1.t1` の場合、パフォーマンススキーマは **TABLE** の行で `'db1'` と `'t1'` の一致を検索し、次に `'db1'` と `'%'` の一致を検索し、次に `'%'` と `'%'` の一致を検索します。さまざまな一致する `setup_objects` 行はさまざまな **ENABLED** 値と **TIMED** 値を持つ可能性があるため、一致が発生する順序が重要です。

テーブル関連イベントの場合、パフォーマンススキーマは `setup_objects` の内容と `setup_instruments` を組み合わせて、インストゥルメントを有効にするかどうか、および有効にされているインストゥルメントの時間を測定するかどうかを判断します。

- `setup_objects` 内の行に一致するテーブルでは、テーブルインストゥルメントは、`setup_instruments` と `setup_objects` の両方で、**ENABLED** が **YES** である場合にのみイベントを生成します。
- 両方の値が **YES** の場合にのみ、タイミング情報が収集されるように、2つのテーブル内の **TIMED** 値が組み合わせられます。

ストアプログラムオブジェクトの場合、パフォーマンススキーマは **ENABLED** および **TIMED** カラムを `setup_objects` 行から直接取得します。 `setup_instruments` との値の組合せはありません。

`setup_objects` に、`db1`、`db2` および `db3` に適用される次の **TABLE** 行が含まれているとします：

OBJECT_TYPE	OBJECT_SCHEMA	OBJECT_NAME	ENABLED	TIMED
TABLE	db1	t1	YES	YES
TABLE	db1	t2	NO	NO
TABLE	db2	%	YES	YES
TABLE	db3	%	NO	NO
TABLE	%	%	YES	YES

`setup_instruments` のオブジェクト関連インストゥルメントの **ENABLED** 値が **NO** の場合、オブジェクトのイベントは監視されません。 **ENABLED** 値が **YES** の場合、イベントモニタリングは、関連 `setup_objects` 行内の **ENABLED** 値に従って行われます。

- `db1.t1` イベントはモニターされます
- `db1.t2` イベントはモニターされません
- `db2.t3` イベントはモニターされます
- `db3.t4` イベントはモニターされません

- `db4.t5` イベントはモニターされます

`setup_instruments` および `setup_objects` テーブルの `TIMED` カラムを組み合わせ、イベントタイミング情報を収集するかどうかを判断する場合も同様のロジックが当てはまります。

永続的テーブルと一時テーブルが同じ名前を持つ場合、`setup_objects` 行に対する照合が両方に対して同様に行われます。一方のテーブルのモニタリングを有効にして、他方を有効にしないことはできません。ただし、各テーブルは個別にインストゥルメントされます。

27.4.6 スレッドによる事前フィルタリング

`threads` テーブルは各サーバスレッドの行を格納します。各行は、スレッドに関する情報を格納し、それに対するモニタリングが有効にされているかどうかを示します。スレッドをモニターするパフォーマンススキーマの場合、これらのことが当てはまる必要があります。

- `setup_consumers` テーブル内の `thread_instrumentation` コンシューマは `YES` である必要があります。
- `threads.INSTRUMENTED` カラムは `YES` である必要があります。
- `setup_instruments` テーブル内で有効にされているインストゥルメントから生成されたスレッドイベントに対してのみ、モニタリングが行われます。

`threads` テーブルには、履歴イベントロギングを実行するかどうかともサーバスレッドごとに示されます。これには待機イベント、ステージイベント、ステートメントイベントおよびトランザクションイベントが含まれ、次のテーブルへのロギングに影響します:

```
events_waits_history
events_waits_history_long
events_stages_history
events_stages_history_long
events_statements_history
events_statements_history_long
events_transactions_history
events_transactions_history_long
```

履歴イベントロギングを実行するには、次のことが当てはまる必要があります:

- `setup_consumers` テーブルの適切な履歴関連コンシューマを有効にする必要があります。たとえば、`events_waits_history` および `events_waits_history_long` テーブルの待機イベントロギングでは、対応する `events_waits_history` および `events_waits_history_long` コンシューマが `YES` である必要があります。
- `threads.HISTORY` カラムは `YES` である必要があります。
- ロギングは、`setup_instruments` テーブルで有効になっているインストゥルメントから生成されたスレッドイベントに対してのみ行われます。

フォアグラウンドスレッド (クライアント接続の結果) の場合、`threads` テーブルの行の `INSTRUMENTED` および `HISTORY` カラムの初期値は、スレッドに関連付けられたユーザーアカウントが `setup_actors` テーブルのいずれかの行と一致するかどうかによって決まります。値は、一致する `setup_actors` テーブルの行の `ENABLED` カラムおよび `HISTORY` カラムから取得されます。

バックグラウンドスレッドの場合、関連付けられたユーザーはありません。 `INSTRUMENTED` および `HISTORY` はデフォルトで `YES` であり、`setup_actors` は参照されません。

初期 `setup_actors` の内容は次のように見えます。

```
mysql> SELECT * FROM performance_schema.setup_actors;
+-----+-----+-----+-----+-----+
| HOST | USER | ROLE | ENABLED | HISTORY |
+-----+-----+-----+-----+
| %   | %   | %   | YES   | YES   |
+-----+-----+-----+-----+
```

`HOST` および `USER` カラムにはリテラルのホストまたはユーザー名、または任意の名前に一致する `'%'` が格納されているべきです。

ENABLED および **HISTORY** のカラムは、前述の他の条件に従って、一致するスレッドのインストゥルメンテーションおよび履歴イベントロギングを有効にするかどうかを示します。

パフォーマンススキーマは、**setup_actors** の新しいフォアグラウンドスレッドごとに一致をチェックするとき、**USER** および **HOST** カラムを使用して、より具体的な一致を最初に見つけようとしています (**ROLE** は使用されません):

- **USER='literal'** および **HOST='literal'** を含む行。
- **USER='literal'** および **HOST='%'** を含む行。
- **USER='%'** および **HOST='literal'** を含む行。
- **USER='%'** および **HOST='%'** を含む行。

一致する **setup_actors** 行が異なると **USER** 値と **HOST** 値が異なる可能性があるため、一致が発生する順序は重要です。これにより、**ENABLED** および **HISTORY** のカラム値に基づいて、インストゥルメント処理および履歴イベントロギングをホスト、ユーザーまたはアカウント (ユーザーとホストの組合せ) ごとに選択的に適用できます:

- 最適な一致が **ENABLED=YES** の行である場合、スレッドの **INSTRUMENTED** 値は **YES** になります。最適な一致が **HISTORY=YES** の行である場合、スレッドの **HISTORY** 値は **YES** になります。
- 最適な一致が **ENABLED=NO** の行である場合、スレッドの **INSTRUMENTED** 値は **NO** になります。最適な一致が **HISTORY=NO** の行である場合、スレッドの **HISTORY** 値は **NO** になります。
- 一致するものが見つからない場合、スレッドの **INSTRUMENTED** および **HISTORY** の値は **NO** になります。

setup_actors 行の **ENABLED** カラムと **HISTORY** カラムは、互いに独立して **YES** または **NO** に設定できます。つまり、履歴イベントを収集するかどうかとは別にインストゥルメンテーションを有効にできます。

デフォルトでは、監視および履歴イベント収集はすべての新しいフォアグラウンドスレッドに対して有効になっています。これは、**setup_actors** テーブルには、**HOST** と **USER** の両方の **'%'** を含む行が最初に含まれているためです。一部のフォアグラウンドスレッドに対してのみ監視を有効にするなど、より限定的な照合を実行するには、この行が任意の接続に一致するため、この行を変更し、より具体的な **HOST/USER** の組合せに対して行を追加する必要があります。

次のように **setup_actors** を変更するとします。

```
UPDATE performance_schema.setup_actors
SET ENABLED = 'NO', HISTORY = 'NO'
WHERE HOST = '%' AND USER = '%';
INSERT INTO performance_schema.setup_actors
(HOST,USER,ROLE,ENABLED,HISTORY)
VALUES('localhost','joe','%','YES','YES');
INSERT INTO performance_schema.setup_actors
(HOST,USER,ROLE,ENABLED,HISTORY)
VALUES('hosta.example.com','joe','%','YES','NO');
INSERT INTO performance_schema.setup_actors
(HOST,USER,ROLE,ENABLED,HISTORY)
VALUES('%','sam','%','NO','YES');
```

UPDATE ステートメントは、インストゥルメンテーションおよび履歴イベント収集を無効にするようにデフォルトの一致を変更します。**INSERT** ステートメントは、より具体的な一致のために行を追加します。

パフォーマンススキーマは、新しい接続スレッドの **INSTRUMENTED** および **HISTORY** の値を次のように設定する方法を決定します:

- **joe** がローカルホストから接続する場合、接続は最初に挿入された行に一致します。スレッドの **INSTRUMENTED** および **HISTORY** の値は **YES** になります。
- **joe** が **hosta.example.com** から接続する場合、接続は挿入された 2 番目の行と一致します。スレッドの **INSTRUMENTED** 値は **YES** になり、**HISTORY** 値は **NO** になります。
- **joe** がほかのすべてのホストから接続する場合、一致はありません。スレッドの **INSTRUMENTED** および **HISTORY** の値は **NO** になります。

- `sam` がいずれかのホストから接続する場合、接続は 3 番目に挿入された行と一致します。スレッドの `INSTRUMENTED` 値は `NO` になり、`HISTORY` 値は `YES` になります。
- その他の接続では、`HOST` および `USER` が '%' に設定されている行が一致します。この行の `ENABLED` および `HISTORY` は `NO` に設定されているため、スレッドの `INSTRUMENTED` および `HISTORY` の値は `NO` になります。

`setup_actors` テーブルの変更は、変更後に作成されたフォアグラウンドスレッドにのみ影響し、既存のスレッドには影響しません。既存のスレッドに影響を与えるには、`threads` テーブルの行の `INSTRUMENTED` および `HISTORY` カラムを変更します。

27.4.7 コンシューマによる事前フィルタリング

`setup_consumers` テーブルは使用可能なコンシューマの種類とどれが有効にされているかを一覧表示します。

```
mysql> SELECT * FROM performance_schema.setup_consumers;
```

NAME	ENABLED
events_stages_current	NO
events_stages_history	NO
events_stages_history_long	NO
events_statements_current	YES
events_statements_history	YES
events_statements_history_long	NO
events_transactions_current	YES
events_transactions_history	YES
events_transactions_history_long	NO
events_waits_current	NO
events_waits_history	NO
events_waits_history_long	NO
global_instrumentation	YES
thread_instrumentation	YES
statements_digest	YES

コンシューマステージで、事前フィルタリングに影響するように、`setup_consumers` テーブルを変更し、イベントの送信先を決定します。コンシューマを有効または無効にするには、その `ENABLED` 値を `YES` または `NO` に設定します。

`setup_consumers` テーブルへの変更はただちにモニタリングに影響します。

コンシューマを無効にすると、サーバーはそのコンシューマの宛先の保守に時間を費やさなくなります。たとえば、履歴イベント情報に関心がない場合、履歴コンシューマを無効にします。

```
UPDATE performance_schema.setup_consumers
SET ENABLED = 'NO'
WHERE NAME LIKE '%history%';
```

`setup_consumers` テーブル内のコンシューマ設定は、高いレベルから低いレベルまでの階層を形成します。次の原則が当てはまります。

- パフォーマンススキーマがコンシューマをチェックし、コンシューマが有効にされていないかぎり、コンシューマに関連付けられている宛先はイベントを受信しません。
- コンシューマは、それが依存するすべてのコンシューマ (ある場合) が有効にされている場合にのみチェックされません。
- コンシューマがチェックされていない場合、またはチェックされているが無効にされている場合、それに依存するほかのコンシューマはチェックされません。
- 依存コンシューマには独自の依存コンシューマがあることがあります。
- イベントがどの宛先にも送信されない場合、パフォーマンススキーマはそれを生成しません。

次のリストに、使用可能なコンシューマ値を説明します。いくつかの代表的なコンシューマ構成とそれらのインストールメンテーションへの効果については、[セクション27.4.8「コンシューマ構成の例」](#)を参照してください。

- [グローバルおよびスレッドコンシューマ](#)

- [待機イベントコンシューマ](#)
- [ステージイベントコンシューマ](#)
- [ステートメントイベントコンシューマ](#)
- [トランザクションイベントコンシューマ](#)
- [ステートメントダイジェストコンシューマ](#)

グローバルおよびスレッドコンシューマ

- `global_instrumentation` は最高レベルのコンシューマです。 `global_instrumentation` が `NO` の場合、それはグローバルインストゥルメンテーションを無効にします。ほかのすべての設定は低いレベルで、チェックされず、何が設定されているかは重要ではありません。グローバルまたはスレッドごとに情報が保守されず、個々のイベントが現在のイベントまたはイベント履歴テーブルに収集されません。 `global_instrumentation` が `YES` の場合、パフォーマンススキーマはグローバル状態の情報を保守し、 `thread_instrumentation` コンシューマもチェックします。
- `thread_instrumentation` は `global_instrumentation` が `YES` の場合のみチェックされます。そうでなければ、 `thread_instrumentation` が `NO` の場合、スレッド固有のインストゥルメンテーションが無効になり、すべての低レベル設定が無視されます。スレッドごとに情報が保守されず、個々のイベントが現在のイベントまたはイベント履歴テーブルに収集されません。 `thread_instrumentation` が `YES` の場合、パフォーマンススキーマはスレッド固有の情報を保守し、 `events_xxx_current` コンシューマもチェックします。

待機イベントコンシューマ

これらのコンシューマには `global_instrumentation` と `thread_instrumentation` の両方が `YES` である必要があり、そうでないと、それらはチェックされません。チェックされた場合、それらは次のように動作します。

- `events_waits_current` は `NO` の場合、 `events_waits_current` テーブル内の個々の待機イベントの収集を無効にします。 `YES` の場合、待機イベント収集を有効にし、パフォーマンススキーマは `events_waits_history` および `events_waits_history_long` コンシューマをチェックします。
- `events_waits_history` は `event_waits_current` が `NO` の場合にチェックされません。そうでない場合、 `NO` または `YES` の `events_waits_history` 値は、 `events_waits_history` テーブルへの待機イベントの収集を無効または有効にします。
- `events_waits_history_long` は `event_waits_current` が `NO` の場合にチェックされません。そうでない場合、 `NO` または `YES` の `events_waits_history_long` 値は、 `events_waits_history_long` テーブルへの待機イベントの収集を無効または有効にします。

ステージイベントコンシューマ

これらのコンシューマには `global_instrumentation` と `thread_instrumentation` の両方が `YES` である必要があり、そうでないと、それらはチェックされません。チェックされた場合、それらは次のように動作します。

- `events_stages_current` は、 `NO` の場合に、 `events_stages_current` テーブルへの個々のステージイベントの収集を無効にします。 `YES` の場合、ステージイベント収集を有効にし、パフォーマンススキーマは `events_stages_history` および `events_stages_history_long` コンシューマをチェックします。
- `events_stages_history` は `event_stages_current` が `NO` の場合にチェックされません。そうでない場合、 `NO` または `YES` の `events_stages_history` 値は、 `events_stages_history` テーブルへのステージイベントの収集を無効または有効にします。
- `events_stages_history_long` は `event_stages_current` が `NO` の場合にチェックされません。そうでない場合、 `NO` または `YES` の `events_stages_history_long` 値は、 `events_stages_history_long` テーブルへのステージイベントの収集を無効または有効にします。

ステートメントイベントコンシューマ

これらのコンシューマには `global_instrumentation` と `thread_instrumentation` の両方が `YES` である必要があり、そうでないと、それらはチェックされません。チェックされた場合、それらは次のように動作します。

- `events_statements_current`は `NO` の場合、`events_statements_current` テーブルへの個々のステートメントイベントの収集を無効にします。 `YES` の場合、ステートメントイベント収集を有効にし、パフォーマンススキーマは `events_statements_history` および `events_statements_history_long` コンシューマをチェックします。
- `events_statements_history` は `events_statements_current` が `NO` の場合にチェックされません。 そうでない場合、`NO` または `YES` の `events_statements_history` 値は、`events_statements_history` テーブルへのステートメントイベントの収集を無効または有効にします。
- `events_statements_history_long` は `events_statements_current` が `NO` の場合にチェックされません。 そうでない場合、`NO` または `YES` の `events_statements_history_long` 値は、`events_statements_history_long` テーブルへのステートメントイベントの収集を無効または有効にします。

トランザクションイベントコンシューマ

これらのコンシューマには `global_instrumentation` と `thread_instrumentation` の両方が `YES` である必要があり、そうでないと、それらはチェックされません。 チェックされた場合、それらは次のように動作します。

- `events_transactions_current` は、`NO` の場合、`events_transactions_current` テーブル内の個々のトランザクションイベントの収集を無効にします。 `YES` の場合は、トランザクションイベント収集が有効になり、パフォーマンススキーマによって `events_transactions_history` および `events_transactions_history_long` コンシューマがチェックされます。
- `events_transactions_current` が `NO` の場合、`events_transactions_history` はチェックされません。 それ以外の場合、`NO` または `YES` の `events_transactions_history` 値は、`events_transactions_history` テーブルでのトランザクションイベントの収集を無効または有効にします。
- `events_transactions_current` が `NO` の場合、`events_transactions_history_long` はチェックされません。 それ以外の場合、`NO` または `YES` の `events_transactions_history_long` 値は、`events_transactions_history_long` テーブルでのトランザクションイベントの収集を無効または有効にします。

ステートメントダイジェストコンシューマ

`statements_digest` コンシューマでは、`global_instrumentation` が `YES` である必要があります。 そうでない場合は、チェックされません。 ステートメントイベントコンシューマへの依存関係がないため、`events_statements_current` に統計を収集する必要なく、ダイジェストごとに統計を取得することができ、これはオーバーヘッドの点で有利です。 逆に、ダイジェストなしで `events_statements_current` の詳細なステートメントを取得できます (この場合、`DIGEST` および `DIGEST_TEXT` のカラムは `NULL` です)。

ステートメントダイジェストの詳細については、[セクション27.10「パフォーマンススキーマのステートメントダイジェストとサンプリング」](#)を参照してください。

27.4.8 コンシューマ構成の例

`setup_consumers` テーブル内のコンシューマ設定は、高いレベルから低いレベルまでの階層を形成します。 次の説明では、コンシューマのしくみを説明し、コンシューマ設定が高から低に段階に有効にされたときの特定の構成とそれらの効果を示します。 示されているコンシューマ値は代表的なものです。 ここで説明する一般原則は、使用可能なほかのコンシューマ値に当てはまります。

構成の説明は、機能とオーバーヘッドが増加する順番で示しています。 下位レベルの設定を有効にすることで提供される情報が必要ない場合は、パフォーマンススキーマが実行するコードが少なくなり、サイフトスルーする情報が少なくなるように無効にします。

`setup_consumers` テーブルには次の値の階層が格納されます。

```
global_instrumentation
thread_instrumentation
events_waits_current
  events_waits_history
  events_waits_history_long
events_stages_current
  events_stages_history
  events_stages_history_long
events_statements_current
  events_statements_history
```

```
events_statements_history_long
events_transactions_current
events_transactions_history
events_transactions_history_long
statements_digest
```

注記

コンシューマ階層では、待機、ステージ、ステートメントおよびトランザクションのコンシューマはすべて同じレベルにあります。これは、待機イベントがステージイベント内にネストするイベントネスト階層とは異なります。ステージイベントはステートメントイベント内にネストされ、ステートメントイベントはトランザクションイベント内にネストされます。

特定のコンシューマ設定が **NO** の場合、パフォーマンススキーマはコンシューマに関連付けられたインストゥルメンテーションを無効にし、すべての低レベルの設定を無視します。特定の設定が **YES** の場合、パフォーマンススキーマはそれに関連付けられたインストゥルメンテーションを有効にし、次の最低レベルの設定をチェックします。各コンシューマのルールの説明については、[セクション27.4.7「コンシューマによる事前フィルタリング」](#)を参照してください。

たとえば、`global_instrumentation` が有効にされている場合、`thread_instrumentation` がチェックされます。`thread_instrumentation` が有効にされている場合、`events_xxx_current` コンシューマがチェックされます。これらのうち `events_waits_current` が有効にされている場合、`events_waits_history` および `events_waits_history_long` がチェックされます。

次の構成の各説明は、パフォーマンススキーマがチェックするセットアップ要素と、それが保守する出力テーブル（つまり、それが情報を収集するテーブル）を示します。

- [インストゥルメンテーションなし](#)
- [グローバルインストゥルメンテーションのみ](#)
- [グローバルおよびスレッドインストゥルメンテーションのみ](#)
- [グローバル、スレッド、および現在のイベントインストゥルメンテーション](#)
- [グローバル、スレッド、現在のイベント、およびイベント履歴インストゥルメンテーション](#)

インストゥルメンテーションなし

サーバー構成の状態:

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+-----+-----+
| NAME          | ENABLED |
+-----+-----+
| global_instrumentation | NO      |
| ...          | ...    |
+-----+-----+
```

この構成では、何もインストゥルメントされません。

チェックされるセットアップ要素:

- テーブル `setup_consumers`、コンシューマ `global_instrumentation`

保守される出力テーブル:

- なし

グローバルインストゥルメンテーションのみ

サーバー構成の状態:

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+-----+-----+
| NAME          | ENABLED |
```

```

+-----+-----+
| global_instrumentation | YES |
| thread_instrumentation | NO  |
| ...                    |    |
+-----+-----+

```

この構成では、インストゥルメンテーションがグローバル状態に対してのみ保守されます。スレッドごとのインストゥルメンテーションは無効にされます。

先述の構成に関連して、チェックされる追加のセットアップ要素:

- テーブル `setup_consumers`、コンシューマ `thread_instrumentation`
- テーブル `setup_instruments`
- テーブル `setup_objects`

先述の構成に関連して保守される追加の出力テーブル:

- `mutex_instances`
- `rwlock_instances`
- `cond_instances`
- `file_instances`
- `users`
- `hosts`
- `accounts`
- `socket_summary_by_event_name`
- `file_summary_by_instance`
- `file_summary_by_event_name`
- `objects_summary_global_by_type`
- `memory_summary_global_by_event_name`
- `table_lock_waits_summary_by_table`
- `table_io_waits_summary_by_index_usage`
- `table_io_waits_summary_by_table`
- `events_waits_summary_by_instance`
- `events_waits_summary_global_by_event_name`
- `events_stages_summary_global_by_event_name`
- `events_statements_summary_global_by_event_name`
- `events_transactions_summary_global_by_event_name`

グローバルおよびスレッドインストゥルメンテーションのみ

サーバー構成の状態:

```

mysql> SELECT * FROM performance_schema.setup_consumers;
+-----+-----+
| NAME          | ENABLED |
+-----+-----+
| global_instrumentation | YES  |
| thread_instrumentation | YES  |

```

```
| events_waits_current      | NO |
...
| events_stages_current    | NO |
...
| events_statements_current | NO |
...
| events_transactions_current | NO |
...
+-----+-----+
```

この構成では、インストゥルメンテーションがグローバルおよびスレッドごとに保守されます。現在のイベントまたはイベント履歴テーブルで、個々のイベントは収集されません。

先述の構成に関連して、チェックされる追加のセットアップ要素:

- xxx が `waits`, `stages`, `statements`, `transactions` であるテーブル `setup_consumers`、コンシューマ `events_xxx_current`
- テーブル `setup_actors`
- カラム `threads.instrumented`

先述の構成に関連して保守される追加の出力テーブル:

- `events_xxx_summary_by_yyy_by_event_name` (xxx は `waits`, `stages`, `statements`, `transactions`、yyy は `thread`, `user`, `host`, `account`)

グローバル、スレッド、および現在のイベントインストゥルメンテーション

サーバー構成の状態:

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+-----+-----+
| NAME                | ENABLED |
+-----+-----+
| global_instrumentation | YES    |
| thread_instrumentation | YES    |
| events_waits_current   | YES    |
| events_waits_history   | NO     |
| events_waits_history_long | NO    |
| events_stages_current  | YES    |
| events_stages_history   | NO     |
| events_stages_history_long | NO    |
| events_statements_current | YES   |
| events_statements_history | NO     |
| events_statements_history_long | NO  |
| events_transactions_current | YES   |
| events_transactions_history | NO     |
| events_transactions_history_long | NO  |
...
+-----+-----+
```

この構成では、インストゥルメンテーションがグローバルおよびスレッドごとに保守されます。個々のイベントが現在のイベントテーブルに収集されますが、イベント履歴テーブルには収集されません。

先述の構成に関連して、チェックされる追加のセットアップ要素:

- コンシューマ `events_xxx_history` (xxx は `waits`, `stages`, `statements`, `transactions`)
- コンシューマ `events_xxx_history_long` (xxx は `waits`, `stages`, `statements`, `transactions`)

先述の構成に関連して保守される追加の出力テーブル:

- `events_xxx_current` (xxx は `waits`, `stages`, `statements`, `transactions`)

グローバル、スレッド、現在のイベント、およびイベント履歴インストゥルメンテーション

`events_xxx_history` および `events_xxx_history_long` コンシューマは無効にされているため、先述の構成はイベント履歴を収集しません。それらのコンシューマは個別またはまとめて有効にして、スレッドごと、グローバルに、またはその両方でイベント履歴を収集できます。

この構成はスレッドごとにイベントを収集しますが、グローバルには収集しません。

```
mysql> SELECT * FROM performance_schema.setup_consumers;
```

NAME	ENABLED
global_instrumentation	YES
thread_instrumentation	YES
events_waits_current	YES
events_waits_history	YES
events_waits_history_long	NO
events_stages_current	YES
events_stages_history	YES
events_stages_history_long	NO
events_statements_current	YES
events_statements_history	YES
events_statements_history_long	NO
events_transactions_current	YES
events_transactions_history	YES
events_transactions_history_long	NO

この構成で保守されるイベント履歴テーブル:

- `events_xxx_history` (xxx は `waits`, `stages`, `statements`, `transactions`)

この構成はグローバルにイベント履歴を収集しますが、スレッドごとには収集しません。

```
mysql> SELECT * FROM performance_schema.setup_consumers;
```

NAME	ENABLED
global_instrumentation	YES
thread_instrumentation	YES
events_waits_current	YES
events_waits_history	NO
events_waits_history_long	YES
events_stages_current	YES
events_stages_history	NO
events_stages_history_long	YES
events_statements_current	YES
events_statements_history	NO
events_statements_history_long	YES
events_transactions_current	YES
events_transactions_history	NO
events_transactions_history_long	YES

この構成で保守されるイベント履歴テーブル:

- `events_xxx_history_long` (xxx は `waits`, `stages`, `statements`, `transactions`)

この構成はスレッドごとおよびグローバルにイベントを収集します。

```
mysql> SELECT * FROM performance_schema.setup_consumers;
```

NAME	ENABLED
global_instrumentation	YES
thread_instrumentation	YES
events_waits_current	YES
events_waits_history	YES
events_waits_history_long	YES
events_stages_current	YES
events_stages_history	YES
events_stages_history_long	YES
events_statements_current	YES
events_statements_history	YES
events_statements_history_long	YES
events_transactions_current	YES
events_transactions_history	YES

```
| events_transactions_history_long | YES |  
...  
+-----+-----+
```

この構成で保守されるイベント履歴テーブル:

- `events_xxx_history` (`xxx` は `waits`, `stages`, `statements`, `transactions`)
- `events_xxx_history_long` (`xxx` は `waits`, `stages`, `statements`, `transactions`)

27.4.9 フィルタリング操作のインストゥルメントまたはコンシューマの指定

フィルタリング操作に指定する名前は、必要に応じて具体的なものでも一般的なものでも指定できます。単一のインストゥルメントまたはコンシューマを示すには、その名前を省略せずに指定します。

```
UPDATE performance_schema.setup_instruments  
SET ENABLED = 'NO'  
WHERE NAME = 'wait/synch/mutex/myisammrg/MYRG_INFO::mutex';  
  
UPDATE performance_schema.setup_consumers  
SET ENABLED = 'NO'  
WHERE NAME = 'events_waits_current';
```

インストゥルメントまたはコンシューマのグループを指定するには、グループメンバーに一致するパターンを使用します。

```
UPDATE performance_schema.setup_instruments  
SET ENABLED = 'NO'  
WHERE NAME LIKE 'wait/synch/mutex/%';  
  
UPDATE performance_schema.setup_consumers  
SET ENABLED = 'NO'  
WHERE NAME LIKE '%history%';
```

パターンを使用する場合、それは関心のあるすべての項目に一致し、ほかの項目に一致しないように、選択してください。たとえば、すべてのファイル I/O インストゥルメントを選択するには、インストゥルメント名プリフィクス全体を含むパターンを使用することをお勧めします。

```
... WHERE NAME LIKE 'wait/io/file/%';
```

'%/file/%'のパターンは、名前に'/file/'の要素が含まれる他のインストゥルメントと一致します。パターン'%file%'は、名前のどこか (`wait/synch/mutex/innodb/file_open_mutex` など) の'`file`'とインストゥルメントが一致するため、あまり適切ではありません。

パターンに一致するインストゥルメントまたはコンシューマ名をチェックするには、簡単なテストを実行します。

```
SELECT NAME FROM performance_schema.setup_instruments  
WHERE NAME LIKE 'pattern';  
  
SELECT NAME FROM performance_schema.setup_consumers  
WHERE NAME LIKE 'pattern';
```

サポートされる名前の種類については、[セクション27.6「パフォーマンススキーマインストゥルメント命名規則」](#)を参照してください。

27.4.10 インストゥルメントされているものの特定

パフォーマンススキーマにどのインストゥルメントが含まれているかを特定するには、常に `setup_instruments` テーブルをチェックすることで可能です。たとえば、InnoDB ストレージエンジンに、どのファイル関連イベントがインストゥルメントされているかを確認するには、次のクエリーを使用します。

```
mysql> SELECT NAME, ENABLED, TIMED  
FROM performance_schema.setup_instruments  
WHERE NAME LIKE 'wait/io/file/innodb/%';  
+-----+-----+-----+  
| NAME | ENABLED | TIMED |  
+-----+-----+-----+  
| wait/io/file/innodb/innodb_tablespace_open_file | YES | YES |  
| wait/io/file/innodb/innodb_data_file | YES | YES |
```



```

wait/io/file/innodb/innodb_log_file      | YES | YES |
wait/io/file/innodb/innodb_temp_file     | YES | YES |
wait/io/file/innodb/innodb_arch_file     | YES | YES |
wait/io/file/innodb/innodb_clone_file    | YES | YES |
+-----+-----+

```

このドキュメントでは、いくつかの理由で、正確に何がインストールされるかについて詳細に説明していません。

- 何がインストールされるかは、サーバーコードです。このコードへの変更は頻繁に行われ、インストールのセットにも影響します。
- すべてのインストールは数百もあるため、それらを挙げることは現実的ではありません。
- 先述のように、`setup_instruments` テーブルをクエリーすることによって見つけることができます。この情報は使用している MySQL のバージョンに常に最新であり、コアサーバーに含まれておらず、自動化されたツールで使用可能な、インストールしている可能性があるインストール済みのプラグインのインストールメンションも含まれます。

27.5 パフォーマンススキーマクエリー

事前フィルタリングは収集されるイベント情報を制限し、特定のユーザーと関係ありません。対照的に、事後フィルタリングは、各ユーザーが、事前フィルタリングの適用後に使用可能なイベントから選択するイベント情報を制限する、適切な `WHERE` 句でクエリーを使用することによって実行されます。

[セクション27.4.3「イベントの事前フィルタリング」](#)で、ファイルインストールの事前フィルタリング方法の例を示しました。イベントテーブルにファイルと非ファイルの両方の情報が格納されている場合、事後フィルタリングはファイルイベントのみの情報を表示するもう1つの方法です。イベント選択を適切に制限するには、クエリーに `WHERE` 句を追加します。

```

mysql> SELECT THREAD_ID, NUMBER_OF_BYTES
FROM performance_schema.events_waits_history
WHERE EVENT_NAME LIKE 'wait/io/file/%'
AND NUMBER_OF_BYTES IS NOT NULL;
+-----+-----+
| THREAD_ID | NUMBER_OF_BYTES |
+-----+-----+
| 11 | 66 |
| 11 | 47 |
| 11 | 139 |
| 5 | 24 |
| 5 | 834 |
+-----+-----+

```

「最もパフォーマンスの高いスキーマ」テーブルには、全テーブルスキャン以外の実行計画へのアクセス権をオブジェクトに付与するインデックスがあります。これらのインデックスは、それらのテーブルを使用する `sys` スキーマビューなど、関連するオブジェクトのパフォーマンスも向上します。詳細は、[セクション8.2.4「パフォーマンススキーマクエリーの最適化」](#)を参照してください。

27.6 パフォーマンススキーマインストールメント命名規則

インストールメント名は、`/`文字で区切られた一連の要素で構成されます。名前の例:

```

wait/io/file/myisam/log
wait/io/file/mysys/charset
wait/lock/table/sql/handler
wait/synch/cond/mysys/COND_alarm
wait/synch/cond/sql/BINLOG::update_cond
wait/synch/mutex/mysys/BITMAP_mutex
wait/synch/mutex/sql/LOCK_delete
wait/synch/rwlock/sql/Query_cache_query::lock
stage/sql/closing tables
stage/sql/Sorting result
statement/com/Execute
statement/com/Query
statement/sql/create_table
statement/sql/lock_tables

```

errors

インストゥルメントの名前空間はツリー状の構造を持ちます。インストゥルメント名の要素は左から右に、より一般的なものからより具体的なものへと進行します。名前が持つ要素の数は、インストゥルメントのタイプによって異なります。

名前内の特定の要素の解釈は、その左側の要素によって異なります。たとえば、`myisam` は次の名前の両方に見られますが、最初の名前の `myisam` はファイル I/O に関連し、2 番目のそれは同期インストゥルメントに関連していません。

```
wait/io/file/myisam/log  
wait/synch/cond/myisam/MI_SORT_INFO::cond
```

インストゥルメント名は、パフォーマンススキーマ実装によって定義された構造を持つプリフィクスと、インストゥルメントコードを実装する開発者によって定義されるサフィクスから構成されます。インストゥルメント接頭辞の最上位要素はインストゥルメントのタイプを示します。この要素は、`performance_timers` テーブルのどのイベントタイマーがインストゥルメントに適用されるかも決定します。インストゥルメントのプリフィクス部分のトップレベルはインストゥルメントの種類を示します。

インストゥルメント名のサフィクス部分は、インストゥルメント自体のコードから生成されます。サフィクスには次のようなレベルが含まれることがあります。

- メジャー要素 (`myisam`, `innodb`, `mysys` や `sql` などのサーバーモジュール) の名前またはプラグイン名。
- 形式 `XXX` (グローバル変数) または `CCC::MMM` (クラス `CCC` のメンバー `MMM`) でのコード内の変数の名前。例: `COND_thread_cache`、`THR_LOCK_myisam`、`BINLOG::LOCK_index`。
- [最上位インストゥルメント要素](#)
- [アイドルインストゥルメント要素](#)
- [エラーインストゥルメント要素](#)
- [メモリーインストゥルメント要素](#)
- [ステージ証書要素](#)
- [ステートメント手段要素](#)
- [スレッドインストゥルメント要素](#)
- [待機インストゥルメントの要素](#)

最上位インストゥルメント要素

- `idle`: インストゥルメントされたアイドルイベント。このインストゥルメントにはこれ以上要素はありません。
- `error`: インストゥルメントされたエラーイベント。このインストゥルメントにはこれ以上要素はありません。
- `memory`: インストゥルメントされたメモリーイベント。
- `stage`: インストゥルメントされたステージイベント。
- `statement`: インストゥルメントされたステートメントイベント。
- `transaction`: インストゥルメントされたトランザクションイベント。このインストゥルメントにはこれ以上要素はありません。
- `wait`: インストゥルメントされた待機イベント。

アイドルインストゥルメント要素

`idle` インストゥルメントはアイドルイベントに使用されます。このイベントは、[セクション 27.12.3.5 「socket_instances テーブル」](#) の `socket_instances.STATE` カラムの説明に従ってパフォーマンススキーマによって生成されます。

エラーインストゥルメント要素

`error` インストゥルメントは、サーバーのエラーおよび警告に関する情報を収集するかどうかを示します。このインストゥルメントはデフォルトで有効になっています。タイミング情報が収集されないため、`setup_instruments` テーブルの `error` 行の `TIMED` カラムは適用できません。

メモリーインストゥルメント要素

メモリーインストゥルメンテーションはデフォルトで有効になっています。メモリーインストゥルメンテーションは、起動時に有効化または無効化できます。また、`setup_instruments` テーブル内の関連するインストゥルメントの `ENABLED` カラムを更新することで、実行時に動的に有効化または無効化できます。メモリーインストゥルメントの名前の形式は `memory/code_area/instrument_name` で、ここで、`code_area` は `sql` や `myisam` などの値で、`instrument_name` はインストゥルメントの詳細です。

接頭辞 `memory/performance_schema/` が付いたインストゥルメントは、パフォーマンススキーマ内の内部バッファに割り当てられているメモリー量を公開します。`memory/performance_schema/` インストゥルメントは組み込みであり、常に有効になっており、起動時または実行時に無効にすることはできません。組み込みメモリーインストゥルメントは、`memory_summary_global_by_event_name` テーブルにのみ表示されます。詳細は、[セクション27.17「パフォーマンススキーマのメモリー割り当てモデル」](#)を参照してください。

ステージ証書要素

ステージインストゥルメントは、形式 `stage/code_area/stage_name` の名前を持ちます。ここで `code_area` は `sql` や `myisam` などの名前で、`stage_name` は、`Sorting result` や `Sending data` などのステートメント処理のステージを示します。ステージは `SHOW PROCESLIST` によって表示されるか、または `INFORMATION_SCHEMA.PROCESLIST` テーブルに表示されるスレッドの状態に対応します。

ステートメント手段要素

- `statement/abstract/*`: ステートメント操作の抽象インストゥルメント。抽象インストゥルメントは、正確なステートメントの種類が分かる前のステートメント分類の初期段階時に使用され、その後種類がわかったときに、より具体的なステートメントインストゥルメントに変更されます。このプロセスの説明については、[セクション27.12.6「パフォーマンススキーマステートメントイベントテーブル」](#)を参照してください。
- `statement/com`: インストゥルメントされたコマンド操作。これらは `COM_xxx` 操作に対応する名前を持ちます (`mysql_com.h` ヘッダーファイルおよび `sql/sql_parse.cc` を参照してください)。たとえば、`statement/com/Connect` および `statement/com/Init DB` インストゥルメントは `COM_CONNECT` および `COM_INIT_DB` コマンドに対応します。
- `statement/scheduler/event`: イベントスケジューラによって実行されたすべてのイベントを追跡する単一のインストゥルメント。このインストゥルメントは、スケジュールされたイベントの実行が開始されると再生されます。
- `statement/sp`: ストアドプログラムによって実行されるインストゥルメントされた内部命令。たとえば、`statement/sp/cfetch` および `statement/sp/freturn` インストゥルメントは、カーソルフェッチおよび関数の戻り命令に使用されます。
- `statement/sql`: インストゥルメントされた SQL ステートメント操作。たとえば、`statement/sql/create_db` および `statement/sql/select` インストゥルメントは `CREATE DATABASE` および `SELECT` ステートメントに使用されます。

スレッドインストゥルメント要素

インストゥルメントされたスレッドは、スレッドクラス名と属性を公開する `setup_threads` テーブルに表示されます。

スレッドインストゥルメントは、`thread` (`thread/sql/parser_service` や `thread/performance_schema/setup` など) で始まります。

待機インストゥルメントの要素

- `wait/io`

インストゥルメントされた I/O 操作。

- [wait/io/file](#)

インストゥルメントされたファイル I/O 操作。ファイルの場合、待機はファイル操作の完了 (たとえば、[fwrite\(\)](#) の呼び出し) を待機する時間です。キャッシュのため、この呼び出し内で、ディスクへの物理的なファイル I/O は行われない可能性があります。

- [wait/io/socket](#)

インストゥルメントされたソケット操作。ソケットインストゥルメントは形式 [wait/io/socket/sql/socket_type](#) の名前を持ちます。サーバーには、それがサポートする各ネットワークプロトコルの待機ソケットがあります。TCP/IP または Unix ソケットファイル接続の待機ソケットに関連付けられているインストゥルメントは、それぞれ [server_tcpip_socket](#) または [server_unix_socket](#) の [socket_type](#) 値を持ちます。待機ソケットが接続を検出すると、サーバーは接続を、個別のスレッドによって管理される新しいソケットに転送します。新しい接続スレッドのインストゥルメントは、[client_connection](#) の [socket_type](#) 値を持ちます。

- [wait/io/table](#)

インストゥルメントされたテーブル I/O 操作。これらには、永続的ベーステーブルまたは一時テーブルへの行レベルアクセスが含まれます。行に影響する操作は、フェッチ、挿入、更新、および削除です。ビューの場合、待機はビューによって参照されるベーステーブルに関連付けられます。

ほとんどの待機と同様、テーブル I/O の待機にはほかの待機も含まれることがあります。たとえば、テーブル I/O にはファイル I/O またはメモリー操作が含まれることがあります。そのため、テーブル I/O 待機の [events_waits_current](#) には通常 2 行あります。詳細については、[セクション 27.8 「パフォーマンススキーマの原子的および分子的イベント」](#) を参照してください。

一部の行操作では、複数のテーブル I/O 待機が発生することがあります。たとえば、挿入は更新を発生させるトリガーをアクティブにすることがあります。

- [wait/lock](#)

インストゥルメントされたロック操作。

- [wait/lock/table](#)

インストゥルメントされたテーブルロック操作。

- [wait/lock/metadata/sql/mdl](#)

インストゥルメントされたメタデータロック操作。

- [wait/synch](#)

インストゥルメントされた同期オブジェクト。同期オブジェクトでは、[TIMER_WAIT](#) 時間には、オブジェクトへのロックがある場合に、その獲得を試みている間のブロックされる時間の量が含まれます。

- [wait/synch/cond](#)

条件は、1 つのスレッドによって、ほかのスレッドに、それらが待機している何かが発生したことを伝えるために使用されます。単一のスレッドが条件を待機していた場合、それはウェイクアップし、その実行を再開できます。複数のスレッドが待機していた場合、それらすべてがウェイクアップし、それらが待機していたリソースを奪い合うことがあります。

- [wait/synch/mutex](#)

ほかのスレッドのリソースへのアクセスを妨げながら、リソース (実行可能コードのセクションなど) へのアクセスを許可するために使用される相互排他オブジェクト。

- [wait/synch/prlock](#)

優先度 [rwlock](#) ロックオブジェクト。

- [wait/synch/rwlock](#)

他のスレッドによる使用を防止しながら、特定の 변수をアクセス用にロックするために使用されるプレーン [read/write lock](#) オブジェクト。共有読み取りロックは複数のスレッドによって同時に獲得できます。排他的書き込みロックは、一度に 1 つのスレッドだけが獲得できます。

- [wait/synch/sxlock](#)

共有排他 (SX) ロックは、他のスレッドによる一貫性のない読み取りを許可しながら、共通リソースへの書き込みアクセスを提供する [rwlock](#) ロックオブジェクトのタイプです。[sxlocks](#) は、同時実行性を最適化し、読み取り/書き込みワークロードのスケラビリティを向上させます。

27.7 パフォーマンススキーマステータスマニタリング

パフォーマンススキーマに関連付けられたいくつかのステータス変数があります。

```
mysql> SHOW STATUS LIKE 'perf%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Performance_schema_accounts_lost | 0 |
| Performance_schema_cond_classes_lost | 0 |
| Performance_schema_cond_instances_lost | 0 |
| Performance_schema_digest_lost | 0 |
| Performance_schema_file_classes_lost | 0 |
| Performance_schema_file_handles_lost | 0 |
| Performance_schema_file_instances_lost | 0 |
| Performance_schema_hosts_lost | 0 |
| Performance_schema_locker_lost | 0 |
| Performance_schema_memory_classes_lost | 0 |
| Performance_schema_metadata_lock_lost | 0 |
| Performance_schema_mutex_classes_lost | 0 |
| Performance_schema_mutex_instances_lost | 0 |
| Performance_schema_nested_statement_lost | 0 |
| Performance_schema_program_lost | 0 |
| Performance_schema_rwlock_classes_lost | 0 |
| Performance_schema_rwlock_instances_lost | 0 |
| Performance_schema_session_connect_attrs_lost | 0 |
| Performance_schema_socket_classes_lost | 0 |
| Performance_schema_socket_instances_lost | 0 |
| Performance_schema_stage_classes_lost | 0 |
| Performance_schema_statement_classes_lost | 0 |
| Performance_schema_table_handles_lost | 0 |
| Performance_schema_table_instances_lost | 0 |
| Performance_schema_thread_classes_lost | 0 |
| Performance_schema_thread_instances_lost | 0 |
| Performance_schema_users_lost | 0 |
+-----+-----+
```

パフォーマンススキーマステータス変数は、メモリー制約のためロードまたは作成できなかったインストールメンテーションに関する情報を提供します。これらの変数の名前にはいくつかの形式があります。

- [Performance_schema_xxx_classes_lost](#) は、種類 [xxx](#) のロードできなかったインストールメントの数を示します。
- [Performance_schema_xxx_instances_lost](#) は、オブジェクトの種類 [xxx](#) の作成できなかったインストールメントの数を示します。
- [Performance_schema_xxx_handles_lost](#) は、オブジェクトの種類 [xxx](#) のオープンできなかったインストールメントの数を示します。
- [Performance_schema_locker_lost](#) は、「失われた」か、または記録されなかったイベント数を示します。

たとえば、相互排他ロックがサーバーソースにインストールされているが、サーバーが実行時にインストールメンテーション用のメモリーを割り当てることができない場合、それは [Performance_schema_mutex_classes_lost](#) を増分します。mutex は同期オブジェクトとして機能しますが (つまり、サーバーは通常どおり機能します)、そのパフォーマンスデータは収集されません。インストールメントを割り当てることができる場合、それはインストールメントされた相互排他ロックインスタンスの初期化に使用できます。グローバル mutex などのシングルトン mutex の

場合、インスタンスは 1 つのみです。その他の相互排他ロックは、接続あたり、または各種キャッシュおよびデータバッファ内のページあたりに 1 つのインスタンスを持つため、インスタンスの数は時間の経過とともに変わります。最大接続数または一部のバッファの最大サイズを増やすと、一度に割り当てられるインスタンスの最大数が増えます。サーバーが特定のインストゥルメントされた相互排他ロックインスタンスを作成できない場合、それは `Performance_schema_mutex_instances_lost` を増分します。

次の条件が維持されているとします。

- サーバーは `--performance_schema_max_mutex_classes=200` オプションを使用して起動されているため、200 相互排他ロックインストゥルメントのための空きがあります。
- 150 相互排他ロックインストゥルメントはすでにロードされています。
- `plugin_a` という名前のプラグインには 40 相互排他ロックインストゥルメントが含まれます。
- `plugin_b` という名前のプラグインには 20 相互排他ロックインストゥルメントが含まれます。

サーバーは、次の一連のステートメントに示すように、プラグインに、それらが必要とする数と使用可能な数に応じて、相互排他ロックインストゥルメントを割り当てます。

```
INSTALL PLUGIN plugin_a
```

現在サーバーには $150+40 = 190$ 相互排他ロックインストゥルメントがあります。

```
UNINSTALL PLUGIN plugin_a;
```

サーバーにはまだ 190 インストゥルメントがあります。プラグインコードによって生成されたすべての履歴データはまだ使用できますが、インストゥルメントの新しいイベントは収集されません。

```
INSTALL PLUGIN plugin_a;
```

サーバーは 40 インストゥルメントがすでに定義されていることを検出したため、新しいインストゥルメントが作成されず、以前に割り当てられた内部メモリーバッファが再利用されます。サーバーにはまだ 190 インストゥルメントがあります。

```
INSTALL PLUGIN plugin_b;
```

サーバーは $200-190 = 10$ インストゥルメント (この例では、相互排他ロッククラス) の空きがあり、プラグインに 20 個の新しいインストゥルメントが含まれていることを認識します。10 個のインストゥルメントがロードされ、10 個は破棄されるか「失われます。」 `Performance_schema_mutex_classes_lost` は失われたインストゥルメント (相互排他ロッククラス) の数を示します。

```
mysql> SHOW STATUS LIKE "perf%mutex_classes_lost";
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Performance_schema_mutex_classes_lost | 10 |
+-----+-----+
1 row in set (0.10 sec)
```

インストゥルメンテーションはまだ機能し、`plugin_b` の (部分) データを収集します。

サーバーが相互排他ロックインストゥルメントを作成できないと、次の結果が発生します。

- インストゥルメントの行が `setup_instruments` テーブルに挿入されません。
- `Performance_schema_mutex_classes_lost` が 1 増加します。
- `Performance_schema_mutex_instances_lost` は変更されません。(相互排他ロックインストゥルメントが作成されない場合、あとでインストゥルメントされた相互排他ロックインスタンスの作成にそれを使用できません。)

先述のパターンは相互排他ロックだけでなく、すべての種類のインストゥルメントに当てはまります。

0 より大きい `Performance_schema_mutex_classes_lost` の値は 2 つのケースで発生する可能性があります。

- 数バイトのメモリーを節約するには、サーバーを `--performance_schema_max_mutex_classes=N` で起動します。ここで `N` はデフォルト値未満です。デフォルト値を選択すれば、MySQL 配布で提供されるすべてのプラグインを

ロードするために十分ですが、一部のプラグインをロードしない場合にこれを減らすことができます。たとえば、配布内の一部のストレージエンジンをロードしないように選択できます。

- パフォーマンススキーマ用にインストールされたサードパーティープラグインをロードしますが、サーバーの起動時に、プラグインのインストールメモリ要件を考慮しません。それはサードパーティー製であるため、このエンジンのインストールメモリ消費は `performance_schema_max_mutex_classes` で選択されたデフォルト値で考慮されていません。

サーバーにプラグインのインストール用の十分なリソースがなく、ユーザーが `--performance_schema_max_mutex_classes=N` を使用して、明示的に多く割り当てていない場合、このプラグインをロードすると、インストールが不足します。

`performance_schema_max_mutex_classes` に選択されている値が小さすぎる場合、エラーログにエラーが報告されず、実行時に障害が発生しません。ただし、`performance_schema` データベース内のテーブルの内容にイベントがありません。 `Performance_schema_mutex_classes_lost` ステータス変数は、インストールの作成の失敗のために、一部のイベントが内部で削除されたことを示す唯一の目に見えるしるしです。

インストールが失われていない場合、それはパフォーマンススキーマに認識され、インスタンスのインストール時に使用されます。たとえば、`wait/synch/mutex/sql/LOCK_delete` は `setup_instruments` テーブル内の相互排他ロックインストールの名前です。サーバーの実行時に相互排他ロックの多くのインスタンスが必要であっても、コード内 (`THD::LOCK_delete` 内) で相互排他ロックの作成時に、この単一のインストールが使用されます。この場合、`LOCK_delete` は接続 (`THD`) ごとの相互排他ロックであるため、サーバーに 1000 接続がある場合、1000 スレッドと 1000 個のインストールされた `LOCK_delete` 相互排他ロックインスタンス (`THD::LOCK_delete`) が存在します。

サーバーにこれらの 1000 個のインストールされた相互排他ロック (インスタンス) すべてのための空きがない場合、一部の相互排他ロックはインストールメンション付きで作成され、一部はインストールメンションなしで作成されます。サーバーが 800 インスタンスしか作成できない場合、200 インスタンスが失われます。サーバーは実行し続けますが、`Performance_schema_mutex_instances_lost` を 200 増分し、インスタンスを作成できなかったことを示します。

0 より大きい `Performance_schema_mutex_instances_lost` は、`--performance_schema_max_mutex_instances=N` で割り当てられたものより多くの相互排他ロックを、コードで実行時に初期化した場合に発生する可能性があります。

結論として、`SHOW STATUS LIKE 'perf%'` に何も失われていないことが示されている (すべての値がゼロ) 場合に、パフォーマンススキーマデータは正確で信頼できます。何かが失われた場合、データは不完全であり、使用するように入力されたメモリの量が不十分であったとすると、パフォーマンススキーマはすべてを記録できていません。この場合、特定の `Performance_schema_xxx_lost` 変数に問題の領域が示されます。

場合によって、意図的にインストールの不足を発生させることが適切なことがあります。たとえば、ファイル I/O のパフォーマンスデータに関心がない場合は、ファイル I/O に関連するすべてのパフォーマンススキーマのパラメータを 0 に設定して、サーバーを起動できます。ファイル関連のクラス、インスタンスまたはハンドルにメモリーは割り当てられず、すべてのファイルイベントが失われます。

`SHOW ENGINE PERFORMANCE_SCHEMA STATUS` を使用して、パフォーマンススキーマコードの内部操作を検査します。

```
mysql> SHOW ENGINE PERFORMANCE_SCHEMA STATUS\G
...
***** 3. row *****
Type: performance_schema
Name: events_waits_history.size
Status: 76
***** 4. row *****
Type: performance_schema
Name: events_waits_history.count
Status: 10000
***** 5. row *****
Type: performance_schema
Name: events_waits_history.memory
Status: 760000
...
***** 57. row *****
Type: performance_schema
Name: performance_schema.memory
Status: 26459600
```

このステートメントは、さまざまなパフォーマンススキーマオプションがメモリ要件に与える効果について、DBA が理解できるようにすることを目的としています。フィールドの意味の説明については、[セクション 13.7.7.15 「SHOW ENGINE ステートメント」](#)を参照してください。

27.8 パフォーマンススキーマの原子的および分子的イベント

テーブル I/O イベントの場合、`events_waits_current` には通常 1 行ではなく、2 行あります。たとえば、行フェッチにより、次のような行になる可能性があります。

Row#	EVENT_NAME	TIMER_START	TIMER_END
1	wait/io/file/myisam/dfile	10001	10002
2	wait/io/table/sql/handler	10000	NULL

行フェッチによりファイルが読み取られます。例では、テーブル I/O フェッチイベントがファイル I/O イベントの前に起動していますが、終了していません (その `TIMER_END` 値が `NULL` です)。ファイル I/O イベントはテーブル I/O イベント内に「ネスト」されます。

これは、相互排他ロックやファイル I/O などのほかの「原子的」待機イベントと異なり、テーブル I/O イベントは「分子的」で、ほかのイベントを含んで (重複して) います。`events_waits_current` で、テーブル I/O イベントには通常 2 つの行があります。

- 最新のテーブル I/O 待機イベントについての 1 行
- 任意の種類最新の待機イベントについての 1 行

通常、ただし常にではありませんが、「どの種類の」待機イベントもテーブル I/O イベントと異なります。各従属イベントが完了すると、それは `events_waits_current` から消去されます。この時点で、および次の従属イベントが開始されるまで、テーブル I/O 待機は、あらゆる種類の最新の待機でもあります。

27.9 現在および過去のイベントのパフォーマンススキーマテーブル

`wait`、`stage`、`statement`、および `transaction` イベントの場合、パフォーマンススキーマは現在のイベントをモニターおよび格納できます。また、イベントが終了すると、パフォーマンススキーマはそれらを履歴テーブルに格納できます。イベントタイプごとに、パフォーマンススキーマは現在のイベントと履歴イベントの格納に 3 つのテーブルを使用します。テーブルには次の形式の名前があります。ここで、`xxx` はイベントタイプ (`waits`、`stages`、`statements`、`transactions`) を示します:

- `events_xxx_current`: 「現在のイベント」テーブルには、各スレッド (スレッドごとに 1 行) の現在の監視対象イベントが格納されます。
- `events_xxx_history`: 「最近の履歴」テーブルには、スレッドごとに終了した最新のイベントが格納されます (スレッド当たりの最大行数まで)。
- `events_xxx_history_long`: 「長い履歴」テーブルには、グローバルに終了した最新のイベントが格納されます (すべてのスレッドで、テーブル当たりの最大行数まで)。

各イベントタイプの `_current` テーブルにはスレッドごとに 1 つの行が含まれるため、その最大サイズを構成するためのシステム変数はありません。パフォーマンススキーマは、個々の履歴テーブルを説明するセクションに示されているように、テーブル固有のシステム変数を使用して、サーバーの起動時に履歴テーブルのサイズを自動設定するか、サイズを明示的に構成できます。一般的な自動サイズ設定された値は、`_history` テーブルではスレッドごとに 10 行、`_history_long` テーブルでは合計 10,000 行です。

イベントタイプごとに、`_current`、`_history` および `_history_long` の各テーブルに同じカラムがあります。`_current` テーブルと `_history` テーブルのインデックス付けは同じです。`_history_long` テーブルにインデックス付けがありません。

`_current` テーブルには、サーバー内で現在何が起きているかが表示されます。現在のイベントが終了すると、その `_current` テーブルから削除されます。

`_history` および `_history_long` のテーブルには、最近行われたことが表示されます。履歴テーブルがいっぱいになると、新しいイベントが追加されると古いイベントは破棄されます。テーブルの目的が異なるため、`_history` テーブルと `_history_long` テーブルの行は様々な方法で期限切れになります:

- `_history` は、グローバルサーバーの負荷に関係なく、個々のスレッドを調査することを目的としています。
- `_history_long` は、各スレッドではなくグローバルにサーバーを調査することを目的としています。

2つのタイプの履歴テーブルの違いは、データ保存ポリシーに関連します。イベントが最初に表示されたときに、両方のテーブルに同じデータが含まれています。ただし、各テーブル内のデータは時間の経過とともに異なる期限が切れるため、各テーブルでデータが長期間または短時間保持される可能性があります。

- `_history` では、テーブルに特定のスレッドの最大行数が含まれている場合、そのスレッドの新しい行が追加されると、最も古いスレッド行は破棄されます。
- `_history_long` では、テーブルがいっぱいになると、いずれかの行を生成したスレッドに関係なく、新しい行が追加されたときに最も古い行が破棄されます。

スレッドが終了すると、そのすべての行は `_history` テーブルから破棄されますが、`_history_long` テーブルからは破棄されません。

次の例は、2つのタイプの履歴テーブルに対してイベントを追加および破棄する方法の違いを示しています。原則はすべてのイベントタイプに均等に適用されます。この例は、次の前提に基づいています：

- パフォーマンススキーマは、スレッドあたり 10 行を `_history` テーブルに保持し、合計 10,000 行を `_history_long` テーブルに保持するように構成されています。
- スレッド A は 1 秒あたり 1 つのイベントを生成します。
スレッド B は 100 イベント/秒を生成します。
- 他のスレッドは実行されていません。

5 秒後:

- A および B は、それぞれ 5 および 500 のイベントを生成しました。
- `_history` には、A に 5 行、B に 10 行が含まれています。スレッド当たりの記憶域は 10 行に制限されているため、A の行は破棄されていませんが、B の 490 行は破棄されています。
- `_history_long` には、A は 5 行、B は 500 行が含まれています。テーブルの最大サイズは 10,000 行であるため、いずれのスレッドでも破棄された行はありません。

実行の 5 分 (300 秒) 後:

- A および B は、それぞれ 300 および 30,000 個のイベントを生成しました。
- `_history` には、A に 10 行、B に 10 行が含まれています。スレッド当たりの記憶域は 10 行に制限されているため、A の 290 行は破棄されましたが、B の 29,990 行は破棄されました。A の行には最大 10 秒経過したデータが含まれ、B の行には最大 .1 秒経過したデータのみが含まれます。
- `_history_long` には 10,000 行が含まれます。A と B はともに 101 イベント/秒を生成するため、テーブルには A でなく B から約 100 から 1 の行が混在した、約 $10,000/101 = 99$ 秒前までのデータが含まれます。

27.10 パフォーマンススキーマのステートメントダイジェストとサンプリング

MySQL サーバーは、ステートメントダイジェスト情報を保守できます。ダイジェストプロセスは、各 SQL ステートメントを正規化された形式 (ステートメントダイジェスト) に変換し、正規化された結果から SHA-256 ハッシュ値 (ダイジェストハッシュ値) を計算します。正規化により、類似のステートメントがグループ化され、要約されて、サーバーが実行しているステートメントの種類とそれらが発生する頻度に関する情報が公開されます。ダイジェストごとに、ダイジェストを生成する代表的なステートメントがサンプルとして格納されます。このセクションでは、ステートメントのダイジェストとサンプリングがどのように行われるか、およびそれらがどのように役立つかについて説明します。

MySQL Enterprise Firewall やクエリーリライトプラグインなどの他の機能がステートメントダイジェストにアクセスできるように、パフォーマンススキーマが使用可能かどうかに関係なくパーサーでダイジェストが実行されます。

- [ステートメントダイジェストの一般概念](#)

- パフォーマンススキーマ内のステートメントダイジェスト
- ステートメントダイジェストメモリー使用
- ステートメントサンプリング

ステートメントダイジェストの一般概念

パーサーは、SQL ステートメントを受信すると、そのダイジェストが必要な場合にステートメントダイジェストを計算します。これは、次のいずれかの条件に該当する場合に当てはまります:

- パフォーマンススキーマダイジェストインストゥルメンテーションは有効です
- MySQL Enterprise Firewall は有効です
- クエリーリライトプラグインは有効です

パーサーは、アプリケーションが SQL ステートメントから正規化されたステートメントダイジェストおよびダイジェストハッシュ値を計算するために呼び出すことができる `STATEMENT_DIGEST_TEXT()` および `STATEMENT_DIGEST()` 関数でも使用されます。

`max_digest_length` システム変数値は、正規化されたステートメントダイジェストの計算に使用できるセッション当たりの最大バイト数を決定します。ダイジェスト計算中にその量の領域が使用されると、切捨てが発生: 解析されたステートメントからのそれ以上のトークンは収集されず、ダイジェスト値になりません。解析されたトークンのバイト数が同じ正規化されたステートメントダイジェストを生成し、比較された場合、またはダイジェスト統計のために集計された場合にのみ、ステートメントが同じであるとみなされます。

正規化されたステートメントが計算されると、そこから SHA-256 ハッシュ値が計算されます。さらに、次のようになります:

- MySQL Enterprise Firewall が有効な場合は呼び出され、計算されたダイジェストが使用可能になります。
- クエリーリライトプラグインが有効になっている場合は、そのプラグインがコールされ、ステートメントダイジェストおよびダイジェスト値が使用可能になります。
- パフォーマンススキーマでダイジェスト計測が有効になっている場合は、正規化されたステートメントダイジェストのコピーが作成され、最大 `performance_schema_max_digest_length` バイトが割り当てられます。したがって、`performance_schema_max_digest_length` が `max_digest_length` より小さい場合、コピーは元のコピーと相対的に切り捨てられます。正規化されたステートメントダイジェストのコピーは、元の正規化されたステートメントから計算された SHA-256 ハッシュ値とともに、適切な「パフォーマンススキーマ」テーブルに格納されます。(パフォーマンススキーマが元のステートメントダイジェストと相対的に正規化されたステートメントダイジェストのコピーを切り捨てる場合、SHA-256 ハッシュ値は再計算されません。)

ステートメントの正規化では、ステートメントのテキストがより標準化されたダイジェスト文字列表現に変換され、一般的なステートメントの構造は保持されますが、構造に不可欠でない情報は削除されます:

- データベースまたはテーブル名などのオブジェクト識別子は保持されます。
- リテラル値はパラメータマーカーに変換されます。正規化されたステートメントは、名前、パスワード、日付などの情報を保持しません。
- コメントが削除され、空白が調整されます。

これらのステートメントを考慮してください。

```
SELECT * FROM orders WHERE customer_id=10 AND quantity>20
SELECT * FROM orders WHERE customer_id = 20 AND quantity > 100
```

これらのステートメントを正規化するために、パーサーは ? によってデータ値を置換し、空白を調整します。どちらのステートメントも同じ正規化形式になるため、「同じ」とみなされます。

```
SELECT * FROM orders WHERE customer_id = ? AND quantity > ?
```

正規化されたステートメントは、格納される情報は少ないですが、引き続き元のステートメントを代表しています。異なるデータ値を持つ他の同様のステートメントは、同じ正規化形式になります。

ここで、これらのステートメントを考慮してください。

```
SELECT * FROM customers WHERE customer_id = 1000
SELECT * FROM orders WHERE customer_id = 1000
```

この場合、オブジェクト識別子が異なるため、正規化されたステートメントは異なります:

```
SELECT * FROM customers WHERE customer_id = ?
SELECT * FROM orders WHERE customer_id = ?
```

正規化によってダイジェストバッファで使用可能な領域を超えるステートメント (`max_digest_length` によって決定) が生成された場合、切捨てが発生し、テキストは「...」で終了します。「...」に続く部分のみが異なる長い正規化されたステートメントは、同じとみなされます。これらのステートメントを考慮してください。

```
SELECT * FROM mytable WHERE cola = 10 AND colb = 20
SELECT * FROM mytable WHERE cola = 10 AND colc = 20
```

AND の直後にカットオフが発生した場合、両方のステートメントの形式は次のように正規化されます:

```
SELECT * FROM mytable WHERE cola = ? AND ...
```

この場合、2 つ目のカラム名の違いが失われ、両方のステートメントが同じとみなされます。

パフォーマンススキーマ内のステートメントダイジェスト

パフォーマンススキーマでは、ステートメントダイジェストには次の要素が含まれます:

- `setup_consumers` テーブル内の `statements_digest` コンシューマは、パフォーマンススキーマがダイジェスト情報を保持するかどうかを制御します。 [ステートメントダイジェストコンシューマ](#) を参照してください。
- ステートメントイベントテーブル (`events_statements_current`、`events_statements_history` および `events_statements_history_long`) には、正規化されたステートメントダイジェストおよび対応するダイジェスト SHA-256 ハッシュ値を格納するためのカラムがあります:
 - `DIGEST_TEXT` は、正規化されたステートメントダイジェストのテキストです。これは、最大 `max_digest_length` バイトに計算され、必要に応じてさらに `performance_schema_max_digest_length` バイトに切り捨てられた元の正規化されたステートメントのコピーです。
 - `DIGEST` は、元の正規化されたステートメントから計算されたダイジェスト SHA-256 ハッシュ値です。

[セクション27.12.6「パフォーマンススキーマステートメントイベントテーブル」](#) を参照してください。

- `events_statements_summary_by_digest` サマリーテーブルには、集計されたステートメントダイジェスト情報が表示されます。このテーブルは、`SCHEMA_NAME` と `DIGEST` の組合せごとにステートメントの情報を集計します。パフォーマンススキーマは、SHA-256 ハッシュ値を集約に使用します。これらは、計算が高速で、衝突を最小限に抑える有利な統計分布を持つためです。 [セクション27.12.18.3「ステートメントサマリーテーブル」](#) を参照してください。

一部のパフォーマンステーブルには、ダイジェストの計算元となる元の SQL ステートメントを格納するカラムがあります:

- `events_statements_current`、`events_statements_history` および `events_statements_history_long` ステートメントイベントテーブルの `SQL_TEXT` カラム。
- `events_statements_summary_by_digest` サマリーテーブルの `QUERY_SAMPLE_TEXT` カラム。

ステートメントの表示に使用できる最大領域は、デフォルトで 1024 バイトです。この値を変更するには、サーバーの起動時に `performance_schema_max_sql_text_length` システム変数を設定します。変更は、指定したすべてのカラムに必要な記憶域に影響します。

`performance_schema_max_digest_length` システム変数は、パフォーマンススキーマ内のダイジェスト値の格納に使用できるステートメントあたりの最大バイト数を決定します。ただし、ステートメントのダイジェストの表示長は、キーワードやリテラル値などのステートメント要素の内部エンコーディングのため、使用可能なバッファサイズより長くなる場合があります。したがって、ステートメントイベントテーブルの `DIGEST_TEXT` カラムから選択された値は、`performance_schema_max_digest_length` 値を超えるように見える場合があります。

`events_statements_summary_by_digest` サマリーテーブルには、サーバーによって実行されるステートメントのプロファイルが表示されます。それは、アプリケーションが実行しているステートメントの種類とその頻度を示します。

アプリケーション開発者はこの情報をテーブル内のほかの情報と組み合わせて使用し、アプリケーションのパフォーマンス特性を査定できます。たとえば、待機時間、ロック時間、またはインデックスの使用を示すテーブルカラムは不十分なクエリーの種類を強調表示することができます。これにより、開発者が注意が必要なアプリケーションの部分を把握できます。

`events_statements_summary_by_digest` サマリーテーブルのサイズは固定です。デフォルトでは、パフォーマンススキーマは起動時に使用するサイズを見積もります。テーブルサイズを明示的に指定するには、サーバーの起動時に `performance_schema_digests_size` システム変数を設定します。テーブルがいっぱいになると、パフォーマンススキーマは、`SCHEMA_NAME` と `DIGEST` の値がテーブル内の既存の値と一致しないステートメントを、`SCHEMA_NAME` と `DIGEST` が `NULL` に設定された特殊な行にグループ化します。これにより、すべてのステートメントがカウントされます。ただし、特殊な行が実行されたステートメントのかなりの割合を占める場合は、`performance_schema_digests_size` を増やすことによってサマリーテーブルのサイズを増やすことをお勧めします。

ステートメントダイジェストメモリー使用

末尾のみが異なる非常に長いステートメントを生成するアプリケーションの場合、`max_digest_length` を増やすと、同じダイジェストに集約されるステートメントを区別するダイジェストの計算が可能になります。逆に、`max_digest_length` を減らすと、サーバーはダイジェスト記憶域専用のメモリーが少なくなります。同じダイジェストに集約する長いステートメントの可能性が高くなります。管理者は、特に多数の同時セッションを含むワークロード (サーバーがセッションごとに `max_digest_length` バイトを割り当てる) では、値を大きくすると、対応するメモリー要件が増加することに注意してください。

前述のように、パーサーによって計算される正規化されたステートメントダイジェストは最大 `max_digest_length` バイトに制約されますが、パフォーマンススキーマに格納される正規化されたステートメントダイジェストは `performance_schema_max_digest_length` バイトを使用します。 `max_digest_length` および `performance_schema_max_digest_length` の相対値に関して、次のメモリー使用上の考慮事項が適用されます:

- `max_digest_length` が `performance_schema_max_digest_length` より小さい場合:
 - パフォーマンススキーマ以外のサーバー機能は、最大 `max_digest_length` バイトを占める正規化されたステートメントダイジェストを使用します。
 - パフォーマンススキーマは、格納されている正規化されたステートメントダイジェストをさらに切り捨てることはありませんが、ダイジェストあたりの `max_digest_length` バイト数よりも多くのメモリーを割り当てます。これは不要です。
- `max_digest_length` が `performance_schema_max_digest_length` と等しい場合:
 - パフォーマンススキーマ以外のサーバー機能は、最大 `max_digest_length` バイトを占める正規化されたステートメントダイジェストを使用します。
 - パフォーマンススキーマは、格納されている正規化されたステートメントダイジェストをさらに切り捨てず、ダイジェストあたりの `max_digest_length` バイト数と同じ量のメモリーを割り当てます。
- `max_digest_length` が `performance_schema_max_digest_length` より大きい場合:
 - パフォーマンススキーマ以外のサーバー機能は、最大 `max_digest_length` バイトを占める正規化されたステートメントダイジェストを使用します。
 - パフォーマンススキーマは、格納されている正規化されたステートメントダイジェストをさらに切り捨て、ダイジェストあたりの `max_digest_length` バイト数より少ないメモリーを割り当てます。

パフォーマンススキーマのステートメントイベントテーブルには多くのダイジェストが格納される可能性があるため、`max_digest_length` より小さい `performance_schema_max_digest_length` を設定すると、管理者は次の要素のバランスを取ることができます:

- パフォーマンススキーマ外のサーバー機能で使用可能な長い正規化されたステートメントダイジェストが必要です
- 多くの同時セッション。それぞれがダイジェスト計算メモリーを割り当てます
- 多数のステートメントダイジェストを格納するときに、パフォーマンススキーマステートメントイベントテーブルによるメモリー消費を制限する必要がある

`performance_schema_max_digest_length` 設定はセッションごとではなく、ステートメントごとであり、セッションは `events_statements_history` テーブルに複数のステートメントを格納できます。このテーブルの典型的なステートメントの数はセッションごとに 10 であるため、各セッションは、このテーブルのみに対して `performance_schema_max_digest_length` 値で示されるメモリーの 10 倍を消費します。

また、多くのステートメント (およびダイジェスト) がグローバルに収集されており、特に `events_statements_history_long` テーブルにあります。ここでも、格納された `N` ステートメントは、`performance_schema_max_digest_length` 値で示されたメモリーの `N` 倍を消費します。

SQL ステートメントの格納およびダイジェスト計算に使用されるメモリー量を評価するには、`SHOW ENGINE PERFORMANCE_SCHEMA STATUS` ステートメントを使用するか、次のインストルメントを監視します:

```
mysql> SELECT NAME
      FROM performance_schema.setup_instruments
      WHERE NAME LIKE '%.sqltext';
+-----+-----+
| NAME                                     |
+-----+-----+
| memory/performance_schema/events_statements_history.sqltext |
| memory/performance_schema/events_statements_current.sqltext |
| memory/performance_schema/events_statements_history_long.sqltext |
+-----+-----+

mysql> SELECT NAME
      FROM performance_schema.setup_instruments
      WHERE NAME LIKE 'memory/performance_schema/%.tokens';
+-----+-----+
| NAME                                     |
+-----+-----+
| memory/performance_schema/events_statements_history.tokens    |
| memory/performance_schema/events_statements_current.tokens   |
| memory/performance_schema/events_statements_summary_by_digest.tokens |
| memory/performance_schema/events_statements_history_long.tokens |
+-----+-----+
```

ステートメントサンプリング

パフォーマンススキーマは、ステートメントサンプリングを使用して、`events_statements_summary_by_digest` テーブル内の各ダイジェスト値を生成する代表的なステートメントを収集します。これらのカラムには、サンプルステートメントの情報が格納されます: `QUERY_SAMPLE_TEXT` (ステートメントのテキスト)、`QUERY_SAMPLE_SEEN` (ステートメントが表示されたとき) および `QUERY_SAMPLE_TIMER_WAIT` (ステートメントの待機時間または実行時間)。パフォーマンススキーマは、サンプルステートメントを選択するたびに 3 つのカラムすべてを更新します。

新しいテーブル行が挿入されると、行ダイジェスト値を生成したステートメントは、ダイジェストに関連付けられた現在のサンプルステートメントとして格納されます。その後、同じダイジェスト値を持つほかのステートメントがサーバーに表示されるときに、新しいステートメントを使用して現在のサンプルステートメントを置き換えるかどうか (つまり、再サンプリングするかどうか) を決定します。リサンプリングポリシーは、現在のサンプルステートメントと新しいステートメントの比較待機時間、およびオプションで現在のサンプルステートメントの経過時間に基づきます:

- 待機時間に基づくリサンプリング: 新しいステートメントの待機時間が現在のサンプルステートメントの待機時間よりも長い場合は、現在のサンプルステートメントになります。
- 年齢に基づくリサンプリング: `performance_schema_max_digest_sample_age` システム変数の値がゼロより大きく、現在のサンプルステートメントが何秒以上経過している場合、現在のステートメントは「古すぎます」とみなされ、新しいステートメントで置き換えられます。これは、新しいステートメントの待機時間が現在のサンプルステートメントの待機時間より短い場合でも発生します。

デフォルトでは、`performance_schema_max_digest_sample_age` は 60 秒 (1 分) です。経過時間による「expire」ステートメントのサンプリング速度を変更するには、値を増減します。リサンプリングポリシーのエイジベース部分を無効にするには、`performance_schema_max_digest_sample_age` を 0 に設定します。

27.11 パフォーマンススキーマの一般的なテーブル特性

`performance_schema` データベースの名前は小文字で、その中のテーブルの名前も同様です。クエリーでは名前を小文字で指定してください。

`performance_schema` データベースの多くのテーブルは読取り専用で、変更できません:

```
mysql> TRUNCATE TABLE performance_schema.setup_instruments;  
ERROR 1683 (HY000): Invalid performance_schema usage.
```

セットアップテーブルの一部には、パフォーマンススキーマ操作に影響するように変更できるカラムがあります。さらに行の挿入や削除を許可するものもあります。収集されたイベントをクリアするために切り捨てが許可されるため、`events_waits_` のプリフィクスのある名前のテーブルなど、それらの種類の情報を格納するテーブルで、`TRUNCATE TABLE` を使用できます。

サマリーテーブルは `TRUNCATE TABLE` で切り捨てることができます。一般に、サマリーカラムは行を削除するのではなく、0 または `NULL` にリセットされます。これにより、収集された値をクリアし、アグリゲーションを再開できます。それは、実行時構成の変更を行なったあとなどに便利な場合があります。この切り捨て動作の例外は、個々のサマリーテーブルのセクションに記載されています。

ほかのデータベースやテーブルに対する権限は次のようになります。

- `performance_schema` テーブルから取得するには、`SELECT` 権限が必要です。
- 変更可能なそれらのカラムを変更するには、`UPDATE` 権限が必要です。
- 切り捨て可能なテーブルを切り捨てるには、`DROP` 権限が必要です。

「パフォーマンススキーマ」テーブルに適用される権限のセットは限られているため、データベースまたはテーブルのアーカイブで権限を付与するための短縮形として `GRANT ALL` を使用しようとする、次のエラーで失敗します:

```
mysql> GRANT ALL ON performance_schema.*  
TO 'u1'@'localhost';  
ERROR 1044 (42000): Access denied for user 'root'@'localhost'  
to database 'performance_schema'  
mysql> GRANT ALL ON performance_schema.setup_instruments  
TO 'u2'@'localhost';  
ERROR 1044 (42000): Access denied for user 'root'@'localhost'  
to database 'performance_schema'
```

かわりに、必要な権限を正確に付与します:

```
mysql> GRANT SELECT ON performance_schema.*  
TO 'u1'@'localhost';  
Query OK, 0 rows affected (0.03 sec)  
  
mysql> GRANT SELECT, UPDATE ON performance_schema.setup_instruments  
TO 'u2'@'localhost';  
Query OK, 0 rows affected (0.02 sec)
```

27.12 パフォーマンススキーマテーブルの説明

`performance_schema` データベース内のテーブルは次のようにグループ化できます。

- セットアップテーブル。これらのテーブルは、モニタリング特性の構成と表示に使われます。
- 現在のイベントテーブル。 `events_waits_current` テーブルには各スレッドの最新のイベントが格納されます。その他の類似テーブルには、イベント階層の様々なレベルの現在のイベントが含まれています: ステージイベントの場合は `events_stages_current`、ステートメントイベントの場合は `events_statements_current`、トランザクションイベントの場合は `events_transactions_current`。
- 履歴テーブル。これらのテーブルは現在のイベントテーブルと同じ構造を持ちますが、多くの行を格納します。たとえば、待機イベントの場合、 `events_waits_history` テーブルにはスレッドあたり最新の 10 イベントが格納されます。 `events_waits_history_long` には最新の 10,000 イベントが格納されます。ステージ、ステートメントおよびトランザクション履歴にも同様のテーブルがあります。

履歴テーブルのサイズを変更するには、サーバー起動時に、該当するシステム変数を設定します。たとえば、待機イベント履歴テーブルのサイズを設定するには、 `performance_schema_events_waits_history_size` および `performance_schema_events_waits_history_long_size` を設定します。

- サマリーテーブル。これらのテーブルには、履歴テーブルから破棄されたものを含むイベントのグループ全体で集計された情報が格納されます。

- インスタンステーブル。これらのテーブルは、インストールされたオブジェクトの種類を記述します。インストールされたオブジェクトは、サーバーによって使われると、イベントを生成します。これらのテーブルは、イベント名と説明のメモまたはステータス情報を提供します。
- その他のテーブル。これらはほかのどのテーブルグループにも分類されません。

27.12.1 パフォーマンススキーマテーブルインデックス

次の表に、各パフォーマンススキーマテーブルを一覧表示し、それぞれの簡単な説明を提供します。

表 27.1 パフォーマンススキーマテーブル

テーブル名	説明
accounts	クライアントアカウントごとの接続統計
binary_log_transaction_compression_stats	バイナリログトランザクション圧縮
clone_progress	クローン操作の進行状況
clone_status	クローン操作ステータス
cond_instances	同期オブジェクトインスタンス
data_lock_waits	データロック待機関係
data_locks	保持およびリクエストされたデータロック
error_log	サーバーエラーログの最近のエントリ
events_errors_summary_by_account_by_error	エラーコードおよびアカウントごとのエラー
events_errors_summary_by_host_by_error	エラーコードおよびホストごとのエラー
events_errors_summary_by_thread_by_error	エラーコードおよびホストごとのエラー
events_errors_summary_by_user_by_error	エラーコードおよびホストごとのエラー
events_errors_summary_global_by_error	エラーコードごとのエラー
events_stages_current	現在のステージイベント
events_stages_history	スレッド当たりの最新のステージイベント
events_stages_history_long	全体の最新ステージイベント
events_stages_summary_by_account_by_event_name	アカウントおよびイベント名ごとのステージイベント
events_stages_summary_by_host_by_event_name	ホスト名およびイベント名ごとのステージイベント
events_stages_summary_by_thread_by_event_name	スレッドおよびイベント名ごとのステージ待機
events_stages_summary_by_user_by_event_name	ユーザー名およびイベント名ごとのステージイベント
events_stages_summary_global_by_event_name	イベント名ごとのステージ待機
events_statements_current	現在のステートメントイベント
events_statements_histogram_by_digest	スキーマおよびダイジェスト値ごとのステートメントヒストグラム
events_statements_histogram_global	グローバルに要約されたステートメントヒストグラム
events_statements_history	スレッド当たりの最新のステートメントイベント
events_statements_history_long	全体の最新ステートメントイベント
events_statements_summary_by_account_by_event_name	アカウントおよびイベント名ごとのステートメントイベント
events_statements_summary_by_digest	スキーマおよびダイジェスト値ごとのステートメントイベント
events_statements_summary_by_host_by_event_name	ホスト名およびイベント名ごとのステートメントイベント
events_statements_summary_by_program	ストアードプログラムごとのステートメントイベント

テーブル名	説明
events_statements_summary_by_thread_by_event_name	スレッドおよびイベント名ごとのステートメントイベント
events_statements_summary_by_user_by_event_name	ユーザー名およびイベント名ごとのステートメントイベント
events_statements_summary_global_by_event_name	イベント名ごとのステートメントイベント
events_transactions_current	現在のトランザクションイベント
events_transactions_history	スレッド当たりの最新のトランザクションイベント
events_transactions_history_long	全体の最新のトランザクションイベント
events_transactions_summary_by_account_by_event_name	アカウントおよびイベント名ごとのトランザクションイベント
events_transactions_summary_by_host_by_event_name	ホスト名およびイベント名ごとのトランザクションイベント
events_transactions_summary_by_thread_by_event_name	スレッドおよびイベント名ごとのトランザクションイベント
events_transactions_summary_by_user_by_event_name	ユーザー名およびイベント名ごとのトランザクションイベント
events_transactions_summary_global_by_event_name	イベント名ごとのトランザクションイベント
events_waits_current	現在の待機イベント
events_waits_history	スレッド当たりの最新の待機イベント
events_waits_history_long	全体の最新待機イベント
events_waits_summary_by_account_by_event_name	アカウントおよびイベント名ごとの待機イベント
events_waits_summary_by_host_by_event_name	ホスト名およびイベント名ごとの待機イベント
events_waits_summary_by_instance	インスタンスごとの待機イベント
events_waits_summary_by_thread_by_event_name	スレッドおよびイベント名ごとの待機イベント
events_waits_summary_by_user_by_event_name	ユーザー名およびイベント名ごとの待機イベント
events_waits_summary_global_by_event_name	イベント名ごとの待機イベント
file_instances	ファイルインスタンス
file_summary_by_event_name	イベント名ごとのファイルイベント
file_summary_by_instance	ファイルインスタンスごとのファイルイベント
firewall_groups	キャッシュされた MySQL Enterprise Firewall グループプロファイルに表示
firewall_group_allowlist	キャッシュされた MySQL Enterprise Firewall グループプロファイル許可リストへの表示
firewall_membership	キャッシュされた MySQL Enterprise Firewall グループプロファイルメンバーへの表示
global_status	グローバルステータス変数
global_variables	グローバルシステム変数
host_cache	内部ホストキャッシュからの情報
hosts	クライアントホスト名ごとの接続統計
keyring_keys	キーリングキーのメタデータ
log_status	バックアップのためのサーバーログに関する情報
memory_summary_by_account_by_event_name	アカウントおよびイベント名ごとのメモリー操作
memory_summary_by_host_by_event_name	ホストおよびイベント名ごとのメモリー操作
memory_summary_by_thread_by_event_name	スレッドおよびイベント名ごとのメモリー操作

テーブル名	説明
memory_summary_by_user_by_event_name	ユーザーおよびイベント名ごとのメモリー操作
memory_summary_global_by_event_name	イベント名ごとのグローバルなメモリー操作
metadata_locks	メタデータロックおよびロックリクエスト
mutex_instances	相互排他ロック同期オブジェクトインスタンス
objects_summary_global_by_type	オブジェクトサマリー
performance_timers	使用可能なイベントタイマー
persisted_variables	mysqld-auto.cnf ファイルの内容
prepared_statements_instances	プリパードステートメントのインスタンスおよび統計
processlist	プロセスリスト情報
replication_applier_configuration	レプリカ上のレプリケーションアプライアンスの構成パラメータ
replication_applier_status	レプリカ上のレプリケーションアプライアンスの現在のステータス
replication_applier_status_by_coordinator	SQL またはコーディネータのスレッドアプライヤステータス
replication_applier_status_by_worker	ワーカースレッドアプライヤステータス (レプリカがマルチスレッドでない場合は空)
replication_asynchronous_connection_failover	非同期接続フェイルオーバーメカニズムのソースリスト
replication_connection_configuration	ソースに接続するための構成パラメータ
replication_connection_status	ソースへの接続の現在のステータス
rwlock_instances	ロック同期オブジェクトインスタンス
session_account_connect_attrs	現在のセッションごとの接続属性
session_connect_attrs	すべてのセッションの接続属性
session_status	現在のセッションのステータス変数
session_variables	現在のセッションのシステム変数
setup_actors	新しいフォアグラウンドスレッドのモニタリングの初期化方法
setup_consumers	イベント情報を格納できるコンシューマ
setup_instruments	イベントを収集できるインストゥルメントされたオブジェクトのクラス
setup_objects	モニターすべきオブジェクト
setup_threads	インストゥルメントされたスレッド名と属性
socket_instances	アクティブな接続インスタンス
socket_summary_by_event_name	イベント名ごとのソケット待機と I/O
socket_summary_by_instance	インスタンスごとのソケット待機と I/O
status_by_account	アカウントごとのセッションステータス変数
status_by_host	ホスト名ごとのセッションステータス変数
status_by_thread	セッションごとのセッションステータス変数
status_by_user	ユーザー名ごとのセッションステータス変数
table_handles	テーブルロックおよびロックリクエスト
table_io_waits_summary_by_index_usage	インデックスごとのテーブル I/O 待機
table_io_waits_summary_by_table	テーブルごとのテーブル I/O 待機
table_lock_waits_summary_by_table	テーブルごとのテーブルロック待機

テーブル名	説明
threads	サーバースレッドに関する情報
tls_channel_status	各接続インタフェースの TLS ステータス
tp_thread_group_state	スレッドプールのスレッドグループの状態に関する情報
tp_thread_group_stats	スレッドグループ統計
tp_thread_state	スレッドプールスレッドの状態に関する情報
user_defined_functions	登録済ユーザー定義関数
user_variables_by_thread	スレッド当たりのユーザー定義変数
users	クライアントユーザー名ごとの接続統計
variables_by_thread	セッションごとのセッションシステム変数
variables_info	システム変数が最近どのように設定されたか

27.12.2 パフォーマンススキーマセットアップテーブル

セットアップテーブルは現在のインストールメンテーションに関する情報を提供し、モニタリング構成の変更を可能にします。このため、[UPDATE](#) 権限を持つ場合、これらのテーブルの一部のカラムを変更できます。

セットアップ情報に個々の変数ではなく、テーブルを使用することで、パフォーマンススキーマ構成の変更の高度な柔軟性を提供します。たとえば、標準 SQL 構文の単一のステートメントを使用して、複数の同時の構成変更ができます。

これらのセットアップテーブルを使用できます。

- [setup_actors](#): 新しいフォアグラウンドスレッドのモニタリングの初期化方法
- [setup_consumers](#): イベント情報を送信および格納できる宛先
- [setup_instruments](#): イベントを収集できるインストールされたオブジェクトのクラス
- [setup_objects](#): モニターすべきオブジェクト
- [setup_threads](#): インストールされたスレッド名と属性

27.12.2.1 [setup_actors](#) テーブル

[setup_actors](#) テーブルには、新しいフォアグラウンドサーバースレッド (クライアント接続に関連付けられたスレッド) の監視および履歴イベントロギングを有効にするかどうかを決定する情報が含まれます。このテーブルはデフォルトで 100 行の最大サイズになります。テーブルサイズを変更するには、サーバー起動時に [performance_schema_setup_actors_size](#) システム変数を変更します。

新しいフォアグラウンドスレッドごとに、パフォーマンススキーマはスレッドのユーザーとホストを、[setup_actors](#) テーブルの行に対して照合します。そのテーブルの行が一致する場合、その [ENABLED](#) カラムと [HISTORY](#) カラムの値を使用して、スレッドの [threads](#) テーブルの行の [INSTRUMENTED](#) カラムと [HISTORY](#) カラムがそれぞれ設定されます。これにより、インストールおよび履歴イベントロギングをホスト、ユーザーまたはアカウント (ユーザーとホストの組合せ) ごとに選択的に適用できます。一致するものがない場合、スレッドの [INSTRUMENTED](#) カラムと [HISTORY](#) カラムは [NO](#) に設定されます。

バックグラウンドスレッドの場合、関連付けられたユーザーはありません。 [INSTRUMENTED](#) および [HISTORY](#) はデフォルトで [YES](#) であり、[setup_actors](#) は参照されません。

[setup_actors](#) テーブルの初期コンテンツは任意のユーザーとホストの組合せに一致するため、監視および履歴イベント収集はすべてのフォアグラウンドスレッドに対してデフォルトで有効になっています:

```
mysql> SELECT * FROM performance_schema.setup_actors;
+-----+-----+-----+-----+-----+
| HOST | USER | ROLE | ENABLED | HISTORY |
+-----+-----+-----+-----+-----+
| %   | %   | %   | YES   | YES   |
```


`setup_actors` テーブルを使用してイベント監視に影響を与える方法の詳細は、[セクション27.4.6「スレッドによる事前フィルタリング」](#)を参照してください。

`setup_actors` テーブルの変更は、変更後に作成されたフォアグラウンドスレッドにのみ影響し、既存のスレッドには影響しません。既存のスレッドに影響を与えるには、`threads` テーブルの行の `INSTRUMENTED` および `HISTORY` カラムを変更します。

`setup_actors` テーブルにはこれらのカラムがあります。

- **HOST**

ホスト名。これらはリテラル名または「任意のホスト」を意味する '%' であるべきです。

- **USER**

ユーザー名。これらはリテラル名または「任意のユーザー」を意味する '%' であるべきです。

- **ROLE**

使用されません。

- **ENABLED**

行に一致するフォアグラウンドスレッドのインストゥルメンテーションを有効にするかどうか。値は **YES** または **NO** です。

- **HISTORY**

行に一致するフォアグラウンドスレッドの履歴イベントをログに記録するかどうか。値は **YES** または **NO** です。

`setup_actors` テーブルには次のインデックスがあります：

- 主キー (**HOST**, **USER**, **ROLE**)

`TRUNCATE TABLE` は `setup_actors` テーブルに対して許可されています。行が削除されます。

27.12.2.2 `setup_consumers` テーブル

`setup_consumers` テーブルは、イベント情報を格納でき、有効にされているコンシューマの種類を一覧表示します。

```
mysql> SELECT * FROM performance_schema.setup_consumers;
```

NAME	ENABLED
events_stages_current	NO
events_stages_history	NO
events_stages_history_long	NO
events_statements_current	YES
events_statements_history	YES
events_statements_history_long	NO
events_transactions_current	YES
events_transactions_history	YES
events_transactions_history_long	NO
events_waits_current	NO
events_waits_history	NO
events_waits_history_long	NO
global_instrumentation	YES
thread_instrumentation	YES
statements_digest	YES

`setup_consumers` テーブル内のコンシューマ設定は、高いレベルから低いレベルまでの階層を形成します。さまざまなコンシューマを有効にする効果の詳細については、[セクション27.4.7「コンシューマによる事前フィルタリング」](#)を参照してください。

`setup_consumers` テーブルへの変更はただちにモニタリングに影響します。

`setup_consumers` テーブルにはこれらのカラムがあります。

- **NAME**

コンシューマ名。

- **ENABLED**

コンシューマが有効にされているかどうか。値は **YES** または **NO** です。このカラムは変更できます。コンシューマを無効にすると、サーバーはそれにイベント情報を追加する時間を費やさなくなります。

`setup_consumers` テーブルには次のインデックスがあります:

- 主キー (**NAME**)

TRUNCATE TABLE は、`setup_consumers` テーブルに対して許可されていません。

27.12.2.3 `setup_instruments` テーブル

`setup_instruments` テーブルは、イベントを収集できるインストゥルメントされたオブジェクトのクラスを一覧表示します。

```
mysql> SELECT * FROM performance_schema.setup_instruments\G
***** 1. row *****
  NAME: wait/synch/mutex/pfs/LOCK_pfs_share_list
  ENABLED: NO
  TIMED: NO
  PROPERTIES: singleton
  VOLATILITY: 1
  DOCUMENTATION: Components can provide their own performance_schema tables.
                  This lock protects the list of such tables definitions.
...
***** 369. row *****
  NAME: stage/sql/executing
  ENABLED: NO
  TIMED: NO
  PROPERTIES:
  VOLATILITY: 0
  DOCUMENTATION: NULL
...
***** 687. row *****
  NAME: statement/abstract/Query
  ENABLED: YES
  TIMED: YES
  PROPERTIES: mutable
  VOLATILITY: 0
  DOCUMENTATION: SQL query just received from the network. At this point,
                  the real statement type is unknown, the type will be
                  refined after SQL parsing.
...
***** 696. row *****
  NAME: memory/performance_schema/metadata_locks
  ENABLED: YES
  TIMED: NULL
  PROPERTIES: global_statistics
  VOLATILITY: 1
  DOCUMENTATION: Memory used for table performance_schema.metadata_locks
...

```

ソースコードに追加された各インストゥルメントは、インストゥルメントコードが実行されない場合でも、`setup_instruments` テーブルの行を提供します。インストゥルメントが有効化されて実行されると、インストゥルメントされたインスタンスが作成され、`file_instances` や `rwlock_instances` などの `xxx_instances` テーブルに表示されます。

ほとんどの `setup_instruments` 行を変更すると、すぐに監視に影響します。一部のインストゥルメントでは、変更はサーバーの起動時にのみ有効です。実行時に変更しても効果はありません。これは主にサーバー内の `mutex`、条件、および `rwlocks` に影響しますが、これが当てはまるほかのインストゥルメントが存在する可能性があります。

イベントフィルタリングにおける `setup_instruments` テーブルの役割の詳細については、[セクション27.4.3「イベントの事前フィルタリング」](#)を参照してください。

setup_instruments テーブルにはこれらのカラムがあります。

- **NAME**

インストゥルメント名。 [セクション27.6「パフォーマンススキーマインストゥルメント命名規則」](#)に説明するように、インストゥルメント名には複数の部分があり、階層を形成することがあります。インストゥルメントの実行から生成されるイベントには、インストゥルメント **NAME** 値から取得される **EVENT_NAME** 値があります。(イベントには実際には「名前」がありませんが、これによって、イベントをインストゥルメントに関連付ける方法を提供します。)

- **ENABLED**

インストゥルメントが有効にされているかどうか。値は **YES** または **NO** です。無効にされたインストゥルメントはイベントを生成しません。このカラムは変更できますが、**ENABLED** の設定は、すでに作成されているインストゥルメントには影響しません。

- **TIMED**

インストゥルメントの時間が測定されるかどうか。値は、**YES**、**NO** または **NULL** です。このカラムは変更できますが、**TIMED** の設定は、すでに作成されているインストゥルメントには影響しません。

TIMED 値 **NULL** は、インストゥルメントがタイミングをサポートしていないことを示します。たとえば、メモリー操作は時間指定されていないため、その **TIMED** カラムは **NULL** です。

タイミングをサポートするインストゥルメントに対して **TIMED** を **NULL** に設定しても、タイミングをサポートしないインストゥルメントに対して **TIMED** を **NULL** 以外に設定しても効果はありません。

有効にされたインストゥルメントの時間が測定されない場合、インストゥルメントコードは有効ですが、タイマーは有効ではありません。インストゥルメントによって生成されたイベントの **TIMER_START**、**TIMER_END**、および **TIMER_WAIT** タイマー値が **NULL** になります。これによって、サマリーテーブルの合計、最小、最大、および平均の時間値の計算時に、それらの値が無視されます。

- **PROPERTIES**

インストゥルメントプロパティ。このカラムは **SET** データ型を使用するため、インストゥルメントごとに次のリストの複数のフラグを設定できます：

- **global_statistics**: インストゥルメントはグローバルサマリーのみを生成します。スレッドごと、アカウントごと、ユーザーごと、ホストごとなど、より詳細なレベルのサマリーは使用できません。たとえば、ほとんどのメモリーインストゥルメントではグローバルサマリーのみが生成されます。
- **mutable**: インストゥルメントは、より具体的なものに「mutate」できます。このプロパティは、ステートメントインストゥルメントにのみ適用されます。
- **progress**: インストゥルメントは進捗データをレポートできます。このプロパティはステージインストゥルメントにのみ適用されます。
- **singleton**: インストゥルメントには単一のインスタンスがあります。たとえば、サーバー内のほとんどのグローバル mutex ロックはシングルトンであるため、対応するインストゥルメントも同様です。
- **user**: インストゥルメントは、(システムワークロードではなく) ユーザーワークロードに直接関連しています。そのようなインストゥルメントには、[wait/io/socket/sql/client_connection](#) があります。

- **VOLATILITY**

インストゥルメントのボラティリティ。ボラティリティ値の範囲は、低から高です。値は、[mysql/psi/psi_base.h](#) ヘッダーファイルで定義されている **PSI_VOLATILITY_xxx** 定数に対応します：

```
#define PSI_VOLATILITY_UNKNOWN 0
#define PSI_VOLATILITY_PERMANENT 1
#define PSI_VOLATILITY_PROVISIONING 2
#define PSI_VOLATILITY_DDL 3
#define PSI_VOLATILITY_CACHE 4
#define PSI_VOLATILITY_SESSION 5
#define PSI_VOLATILITY_TRANSACTION 6
```

```
#define PSI_VOLATILITY_QUERY 7  
#define PSI_VOLATILITY_INTRA_QUERY 8
```

VOLATILITY カラムは、ユーザー (およびパフォーマンススキーマコード) にインストールの実行時動作に関するヒントを提供するための単なる情報です。

低揮発性インデックス (PERMANENT = 1) を持つインストールは、サーバーの起動時に一度作成され、通常のサーバー操作中に破棄または再作成されることはありません。これらは、サーバーの停止中にのみ破棄されます。

たとえば、`wait/synch/mutex/pfs/LOCK_pfs_share_list` mutex は揮発性 1 で定義されており、これは一度作成されることを意味します。インストールメンターション自体から発生する可能性のあるオーバーヘッド (相互排他ロック初期化) は、このインストールには影響しません。実行時オーバーヘッドは、mutex をロックまたはロック解除する場合にのみ発生します。

ボラティリティインデックスが高いインストール (SESSION = 5 など) は、ユーザーセッションごとに作成および破棄されます。たとえば、`wait/synch/mutex/sql/THD::LOCK_query_plan` mutex は、セッションが接続されるたびに作成され、セッションが切断されると破棄されます。

この mutex は、パフォーマンススキーマのオーバーヘッドにより機密性が高くなります。これは、オーバーヘッドがロックおよびロック解除インストールメンターションだけでなく、より頻繁に実行される相互排他ロック作成および破棄インストールメンターションからも発生するためです。

ボラティリティの別の側面は、**ENABLED** カラムの更新が実際になんらかの影響を与えるかどうかと、そのタイミングに関するものです:

- **ENABLED** の更新は、後で作成されるインストールされたオブジェクトに影響しますが、すでに作成されているインストールには影響しません。
- より多くの「volatile」のインストールでは、`setup_instruments` テーブルの新しい設定がより早く使用されます。

たとえば、次のステートメントは既存のセッションの `LOCK_query_plan` mutex には影響しませんが、更新後に作成された新しいセッションには影響します:

```
UPDATE performance_schema.setup_instruments  
SET ENABLED=value  
WHERE NAME = 'wait/synch/mutex/sql/THD::LOCK_query_plan';
```

このステートメントは、実際には何の効果もありません:

```
UPDATE performance_schema.setup_instruments  
SET ENABLED=value  
WHERE NAME = 'wait/synch/mutex/pfs/LOCK_pfs_share_list';
```

この mutex は永続的であり、更新の実行前にすでに作成されています。mutex が再度作成されることはないため、`setup_instruments` の **ENABLED** 値は使用されません。この mutex を有効または無効にするには、かわりに `mutex_instances` テーブルを使用します。

• DOCUMENTATION

インストールの目的を説明する文字列。説明がない場合、値は **NULL** です。

`setup_instruments` テーブルには次のインデックスがあります:

- 主キー (NAME)

TRUNCATE TABLE は、`setup_instruments` テーブルに対して許可されていません。

27.12.2.4 setup_objects テーブル

`setup_objects` テーブルは、パフォーマンススキーマが特定のオブジェクトをモニターするかどうかを制御します。このテーブルはデフォルトで 100 行の最大サイズになります。テーブルサイズを変更するには、サーバー起動時に `performance_schema_setup_objects_size` システム変数を変更します。

初期 `setup_objects` の内容は次のように見えます。

```
mysql> SELECT * FROM performance_schema.setup_objects;
+-----+-----+-----+-----+
| OBJECT_TYPE | OBJECT_SCHEMA | OBJECT_NAME | ENABLED | TIMED |
+-----+-----+-----+-----+
| EVENT | mysql | % | NO | NO |
| EVENT | performance_schema | % | NO | NO |
| EVENT | information_schema | % | NO | NO |
| EVENT | % | % | YES | YES |
| FUNCTION | mysql | % | NO | NO |
| FUNCTION | performance_schema | % | NO | NO |
| FUNCTION | information_schema | % | NO | NO |
| FUNCTION | % | % | YES | YES |
| PROCEDURE | mysql | % | NO | NO |
| PROCEDURE | performance_schema | % | NO | NO |
| PROCEDURE | information_schema | % | NO | NO |
| PROCEDURE | % | % | YES | YES |
| TABLE | mysql | % | NO | NO |
| TABLE | performance_schema | % | NO | NO |
| TABLE | information_schema | % | NO | NO |
| TABLE | % | % | YES | YES |
| TRIGGER | mysql | % | NO | NO |
| TRIGGER | performance_schema | % | NO | NO |
| TRIGGER | information_schema | % | NO | NO |
| TRIGGER | % | % | YES | YES |
+-----+-----+-----+-----+
```

`setup_objects` テーブルへの変更はただちにオブジェクトモニタリングに影響します。

`setup_objects` に示されているオブジェクトの種類では、パフォーマンススキーマはそれらのモニター方法にテーブルを使用します。オブジェクトの一致は `OBJECT_SCHEMA` および `OBJECT_NAME` カラムに基づきます。一致のないオブジェクトはモニターされません。

デフォルトのオブジェクト構成の効果は、`mysql`、`INFORMATION_SCHEMA`、および `performance_schema` データベースのテーブルを除くすべてのテーブルをインストールすることです。(`INFORMATION_SCHEMA` データベース内のテーブルは、`setup_objects` の内容に関係なくインストールされず、`information_schema.%` の行は単にこのデフォルトを明示します。)

パフォーマンススキーマは、`setup_objects` の一致をチェックする場合、まずより詳細な一致を見つけようとします。たとえば、テーブル `db1.t1` では、`'db1'` と `'t1'`、次に `'db1'` と `'%'`、次に `'%'` と `'%'` の一致を検索します。さまざまな一致する `setup_objects` 行はさまざまな `ENABLED` 値と `TIMED` 値を持つ可能性があるため、一致が発生する順序が重要です。

テーブルへの `INSERT` または `DELETE` 権限を持つユーザーが、`setup_objects` に行を挿入したり、削除したりできます。既存の行では、テーブルへの `UPDATE` 権限を持つユーザーによって、`ENABLED` および `TIMED` カラムのみを変更できます。

イベントフィルタリングにおける `setup_objects` テーブルの役割の詳細については、[セクション27.4.3「イベントの事前フィルタリング」](#)を参照してください。

`setup_objects` テーブルにはこれらのカラムがあります。

- `OBJECT_TYPE`

インストールするオブジェクトの種類。値は、`'EVENT'` (イベントスケジューライベント)、`'FUNCTION'` (ストアドファンクション)、`'PROCEDURE'` (ストアドプロシージャ)、`'TABLE'` (実テーブル) または `'TRIGGER'` (トリガー) のいずれかです。

`TABLE` フィルタリングはテーブル I/O イベント (`wait/io/table/sql/handler` インストールメント) およびテーブルロックイベント (`wait/lock/table/sql/handler` インストールメント) に影響します。

- `OBJECT_SCHEMA`

オブジェクトを格納するスキーマ。これはリテラル名、または「任意のスキーマ」を意味する `'%'` であるべきです。

- `OBJECT_NAME`

インストールされたオブジェクトの名前。これはリテラル名、または「任意のオブジェクト」を意味する '%' であるべきです。

- **ENABLED**

オブジェクトのイベントがインストールされるかどうか。値は **YES** または **NO** です。このカラムは変更できません。

- **TIMED**

オブジェクトのイベントの時間が測定されるかどうか。このカラムは変更できます。

`setup_objects` テーブルには次のインデックスがあります:

- (OBJECT_TYPE, OBJECT_SCHEMA, OBJECT_NAME) のインデックス

`TRUNCATE TABLE` は `setup_objects` テーブルに対して許可されています。行が削除されます。

27.12.2.5 setup_threads テーブル

`setup_threads` テーブルには、インストールスレッドクラスがリストされます。スレッドクラス名と属性を公開します:

```
mysql> SELECT * FROM performance_schema.setup_threads\G
***** 1. row *****
  NAME: thread/performance_schema/setup
  ENABLED: YES
  HISTORY: YES
  PROPERTIES: singleton
  VOLATILITY: 0
  DOCUMENTATION: NULL
...
***** 4. row *****
  NAME: thread/sql/main
  ENABLED: YES
  HISTORY: YES
  PROPERTIES: singleton
  VOLATILITY: 0
  DOCUMENTATION: NULL
***** 5. row *****
  NAME: thread/sql/one_connection
  ENABLED: YES
  HISTORY: YES
  PROPERTIES: user
  VOLATILITY: 0
  DOCUMENTATION: NULL
...
***** 10. row *****
  NAME: thread/sql/event_scheduler
  ENABLED: YES
  HISTORY: YES
  PROPERTIES: singleton
  VOLATILITY: 0
  DOCUMENTATION: NULL
```

`setup_threads` テーブルには、次のカラムがあります:

- **NAME**

インストール名。スレッドインストールは、`thread` (`thread/sql/parser_service` や `thread/performance_schema/setup` など) で始まります。

- **ENABLED**

インストールが有効にされているかどうか。値は **YES** または **NO** です。このカラムは変更できますが、**ENABLED** を設定しても、すでに実行中のスレッドには影響しません。

バックグラウンドスレッドの場合、**ENABLED** 値を設定すると、このインストール用に後で作成され、`threads` テーブルにリストされるスレッドに対して、**INSTRUMENTED** が **YES** または **NO** に設定されるかどうか

かが制御されます。フォアグラウンドスレッドの場合、このカラムは効果がなく、`setup_actors` テーブルが優先されます。

• HISTORY

インストゥルメントの履歴イベントをログに記録するかどうか。値は `YES` または `NO` です。このカラムは変更できますが、`HISTORY` を設定しても、すでに実行中のスレッドには影響しません。

バックグラウンドスレッドの場合、`HISTORY` 値を設定すると、このインストゥルメント用に後で作成され、`threads` テーブルにリストされるスレッドに対して、`HISTORY` が `YES` または `NO` に設定されるかどうか制御されます。フォアグラウンドスレッドの場合、このカラムは効果がなく、`setup_actors` テーブルが優先されます。

• PROPERTIES

インストゥルメントプロパティ。このカラムは `SET` データ型を使用するため、インストゥルメントごとに次のリストの複数のフラグを設定できます:

- `singleton`: インストゥルメントには単一のインスタンスがあります。たとえば、`thread/sql/main` インストゥルメントのスレッドは 1 つのみです。
- `user`: インストゥルメントは、(システムワークロードではなく) ユーザーワークロードに直接関連しています。たとえば、ユーザーセッションを実行する `thread/sql/one_connection` などのスレッドには、システムスレッドと区別するための `user` プロパティがあります。

• VOLATILITY

インストゥルメントのボラティリティ。このカラムは、`setup_instruments` テーブルと同じ意味を持ちます。 [セクション27.12.2.3「setup_instruments テーブル」](#) を参照してください。

• DOCUMENTATION

インストゥルメントの目的を説明する文字列。説明がない場合、値は `NULL` です。

`setup_threads` テーブルには次のインデックスがあります:

- 主キー (`NAME`)

`TRUNCATE TABLE` は、`setup_threads` テーブルに対して許可されていません。

27.12.2.6 setup_timers テーブル

このテーブルは、MySQL 8.0.4 で削除されました。

27.12.3 パフォーマンススキーマインスタンステーブル

インスタンステーブルは、インストゥルメントされたオブジェクトの種類を記述します。それらは、イベント名と説明のメモまたはステータス情報を提供します。

- `cond_instances`: 条件同期オブジェクトインスタンス
- `file_instances`: ファイルインスタンス
- `mutex_instances`: 相互排他ロック同期オブジェクトインスタンス
- `rwlock_instances`: ロック同期オブジェクトインスタンス
- `socket_instances`: アクティブな接続インスタンス

これらのテーブルはインストゥルメントされた同期オブジェクト、ファイル、および接続を一覧表示します。3種類の同期オブジェクト `cond`、`mutex`、および `rwlock` があります。各インスタンステーブルには、各行に関連付けられているインストゥルメントを示す `EVENT_NAME` または `NAME` カラムがあります。 [セクション27.6「パフォーマンススキーマインストゥルメント命名規則」](#) に説明するように、インストゥルメント名には複数の部分があり、階層を形成することがあります。

`mutex_instances.LOCKED_BY_THREAD_ID` および `rwlock_instances.WRITE_LOCKED_BY_THREAD_ID` カラムは、パフォーマンスボトルネックまたはデッドロックの調査にぎわめて重要です。この目的でそれらを使用する方法の例については、[セクション27.19「問題を診断するためのパフォーマンススキーマの使用」](#)を参照してください

27.12.3.1 cond_instances テーブル

`cond_instances` テーブルは、サーバーの実行中にパフォーマンススキーマによって確認されるすべての条件を一覧表示します。条件は、この条件を待機しているスレッドが作業を再開できるように、特定のイベントが発生したことを伝えるために、コードで使用される同期メカニズムです。

スレッドが何かの発生を待機している場合、条件名はスレッドが何を待機しているかを示しますが、ほかのどのスレッド (スレッド) が原因であるかをすぐに通知する方法はありません。

`cond_instances` テーブルにはこれらのカラムがあります。

- `NAME`
条件に関連付けられているインストゥルメント名。
- `OBJECT_INSTANCE_BEGIN`
インストゥルメントされた条件のメモリー内のアドレス。

`cond_instances` テーブルには次のインデックスがあります:

- 主キー (`OBJECT_INSTANCE_BEGIN`)
- (`NAME`) のインデックス

`TRUNCATE TABLE` は、`cond_instances` テーブルに対して許可されていません。

27.12.3.2 file_instances テーブル

`file_instances` テーブルは、ファイル I/O インストゥルメンテーションの実行中にパフォーマンススキーマによって確認されるすべてのファイルを一覧表示します。ディスク上のファイルが開かれていない場合、`file_instances` には表示されません。ファイルがディスクから削除されると、`file_instances` テーブルからも削除されます。

`file_instances` テーブルにはこれらのカラムがあります。

- `FILE_NAME`
ファイル名。
- `EVENT_NAME`
ファイルに関連付けられているインストゥルメント名。
- `OPEN_COUNT`
ファイルへのオープンハンドルのカウント。ファイルを開いてから閉じた場合、そのファイルは 1 回開かれています。サーバーによって現在開かれているすべてのファイルを一覧表示するには、`WHERE OPEN_COUNT > 0` を使用します。

`file_instances` テーブルには次のインデックスがあります:

- 主キー (`FILE_NAME`)
- (`EVENT_NAME`) のインデックス

`TRUNCATE TABLE` は、`file_instances` テーブルに対して許可されていません。

27.12.3.3 mutex_instances テーブル

`mutex_instances` テーブルは、サーバーの実行中にパフォーマンススキーマによって確認されるすべての相互排他ロックを一覧表示します。相互排他ロックは、特定の時間に 1 つだけのスレッドが特定の共通リソースにアクセスできるようにする、コードで使用される同期メカニズムです。リソースは相互排他ロックによって「保護されている」と呼ばれます。

サーバーで実行されている 2 つのスレッド (たとえば、クエリーを同時に実行する 2 つのユーザーセッション) が同じリソース (ファイル、バッファまたは一部のデータ) にアクセスする必要がある場合、これらの 2 つのスレッドは互いに競合するため、mutex でロックを取得する最初のクエリーによって、最初のクエリーが完了するまで待機し、mutex のロックを解除します。

相互排他ロックの保持中に実行される作業は「クリティカルセクション」にあると呼ばれ、複数のクエリーがこのクリティカルセクションを連続して (一度に 1 つずつ) 実行するため、これは潜在的なボトルネックになります。

`mutex_instances` テーブルにはこれらのカラムがあります。

- `NAME`

相互排他ロックに関連付けられているインストゥルメント名。

- `OBJECT_INSTANCE_BEGIN`

インストゥルメントされた相互排他ロックのメモリー内のアドレス。

- `LOCKED_BY_THREAD_ID`

スレッドが現在相互排他ロックされている場合、`LOCKED_BY_THREAD_ID` はロックしているスレッドの `THREAD_ID` になり、そうでない場合、それは `NULL` になります。

`mutex_instances` テーブルには次のインデックスがあります:

- 主キー (`OBJECT_INSTANCE_BEGIN`)

- (`NAME`) のインデックス

- (`LOCKED_BY_THREAD_ID`) のインデックス

`TRUNCATE TABLE` は、`mutex_instances` テーブルに対して許可されていません。

コードにインストゥルメントされた各相互排他ロックについて、パフォーマンススキーマは次の情報を提供します。

- `setup_instruments` テーブルは、プリフィクス `wait/synch/mutex/` を付けて、インストゥルメンテーションポイントの名前を一覧表示します。

- 一部のコードで相互排他ロックが作成されると、行が `mutex_instances` テーブルに追加されます。`OBJECT_INSTANCE_BEGIN` カラムは相互排他ロックを一意に識別するプロパティです。

- スレッドが相互排他ロックのロックを試みた場合、`events_waits_current` テーブルにそのスレッドの行が表示され、それが相互排他ロックを待機していることが示され (`EVENT_NAME` カラム内)、待機されている相互排他ロックが示されます (`OBJECT_INSTANCE_BEGIN` カラム内)。

- スレッドが相互排他ロックのロックに成功した場合:

- `events_waits_current` は相互排他ロックへの待機が完了したことを示します (`TIMER_END` および `TIMER_WAIT` カラム内)

- 完了した待機イベントは `events_waits_history` および `events_waits_history_long` テーブルに追加されます。

- `mutex_instances` は相互排他ロックがスレッドによって所有されるようになったことを示します (`THREAD_ID` カラム内)。

- スレッドが相互排他ロックのロックを解除すると、`mutex_instances` は相互排他ロックに所有者がいなくなったことを示します (`THREAD_ID` カラムが `NULL` になります)。

- 相互排他ロックオブジェクトが破棄されると、対応する行が `mutex_instances` から削除されます。

次の両方のテーブルに対してクエリーを実行することによって、モニタリングアプリケーションまたは DBA は相互排他ロックを伴うスレッド間のボトルネックやデッドロックを検出できます。

- `events_waits_current`、スレッドが待機している相互排他ロックを確認する場合
- `mutex_instances`、相互排他ロックを現在所有しているほかのスレッドを確認する場合

27.12.3.4 `rwlock_instances` テーブル

`rwlock_instances` テーブルには、サーバーの実行中にパフォーマンススキーマによって認識されるすべての `rwlock` (読み取り/書き込みロック) インスタンスが一覧表示されます。 `rwlock` は、特定の時間にそのスレッドが、次の特定のルールに従って、一部の共通リソースにアクセスできるようにするために、コードで使用される同期メカニズムです。リソースは `rwlock` によって「保護されている」と呼ばれます。アクセスは、共有 (多くのスレッドが同時に読み取りロックを持つことができます)、排他 (特定の時間に書き込みロックを持つことができるのは 1 つのスレッドのみ) または共有排他 (スレッドは他のスレッドによる一貫性のない読み取りを許可しながら書き込みロックを持つことができます) のいずれかです。共有排他アクセスは、`sxlock` とも呼ばれ、同時実行性を最適化し、読み取り/書き込みワークロードのスケラビリティを向上させます。

ロックをリクエストしているスレッドの数およびリクエストされたロックの性質に応じて、他のスレッドが最初に終了するまで待機して、共有モード、排他モードまたは共有排他モードでアクセス権を付与することも、まったく付与しないこともできます。

`rwlock_instances` テーブルにはこれらのカラムがあります。

- `NAME`

ロックに関連付けられているインストゥルメント名。

- `OBJECT_INSTANCE_BEGIN`

インストゥルメントされたロックのメモリー内のアドレス。

- `WRITE_LOCKED_BY_THREAD_ID`

スレッドが現在排他 (書き込み) モードでロックされた `rwlock` を持つ場合、`WRITE_LOCKED_BY_THREAD_ID` はロックしているスレッドの `THREAD_ID` になり、そうでない場合、それは `NULL` になります。

- `READ_LOCKED_BY_COUNT`

スレッドが現在共有 (読み取り) モードでロックされた `rwlock` を持つ場合、`READ_LOCKED_BY_COUNT` が 1 増分されます。これはカウンタのみであるため、読み取りロックを保持するスレッドを見つけるためにそれを直接使用することはできませんが、`rwlock` に対して読み取りの競合があるかどうかを確認し、現在アクティブなリーダー数を確認するために使用することができます。

`rwlock_instances` テーブルには次のインデックスがあります:

- 主キー (`OBJECT_INSTANCE_BEGIN`)
- (`NAME`) のインデックス
- (`WRITE_LOCKED_BY_THREAD_ID`) のインデックス

`TRUNCATE TABLE` は、`rwlock_instances` テーブルに対して許可されていません。

次の両方のテーブルに対してクエリーを実行することによって、モニタリングアプリケーションまたは DBA はロックを伴うスレッド間の何らかのボトルネックやデッドロックを検出できます。

- `events_waits_current`、スレッドが待機している `rwlock` を確認する場合
- `rwlock_instances`、`rwlock` を現在所有しているほかのスレッドを確認する場合

制限があります。`rwlock_instances` は、書き込みロックを保持しているスレッドの識別にのみ使用できますが、読み取りロックを保持しているスレッドの識別には使用できません。

27.12.3.5 socket_instances テーブル

`socket_instances` テーブルは MySQL サーバーへのアクティブな接続のリアルタイムスナップショットを提供します。テーブルには、TCP/IP または Unix ソケットファイル接続ごとに 1 行含まれます。このテーブルで使用可能な情報は、サーバーへのアクティブな接続のリアルタイムスナップショットを提供します。(ソケット操作や送受信されたバイト数などのネットワークアクティビティを含む、追加の情報はソケットサマリーテーブルで入手できます。[セクション27.12.18.9「ソケットサマリーテーブル」](#)を参照してください)。

```
mysql> SELECT * FROM performance_schema.socket_instances\G
***** 1. row *****
EVENT_NAME: wait/io/socket/sql/server_unix_socket
OBJECT_INSTANCE_BEGIN: 4316619408
THREAD_ID: 1
SOCKET_ID: 16
IP:
PORT: 0
STATE: ACTIVE
***** 2. row *****
EVENT_NAME: wait/io/socket/sql/client_connection
OBJECT_INSTANCE_BEGIN: 4316644608
THREAD_ID: 21
SOCKET_ID: 39
IP: 127.0.0.1
PORT: 55233
STATE: ACTIVE
***** 3. row *****
EVENT_NAME: wait/io/socket/sql/server_tcpip_socket
OBJECT_INSTANCE_BEGIN: 4316699040
THREAD_ID: 1
SOCKET_ID: 14
IP: 0.0.0.0
PORT: 50603
STATE: ACTIVE
```

ソケットインストゥルメントは形式 `wait/io/socket/sql/socket_type` の名前を持ち、次のように使用されます。

1. サーバーには、それがサポートする各ネットワークプロトコルの待機ソケットがあります。TCP/IP または Unix ソケットファイル接続の待機ソケットに関連付けられているインストゥルメントは、それぞれ `server_tcpip_socket` または `server_unix_socket` の `socket_type` 値を持ちます。
2. 待機ソケットが接続を検出すると、サーバーは接続を、個別のスレッドによって管理される新しいソケットに転送します。新しい接続スレッドのインストゥルメントは、`client_connection` の `socket_type` 値を持ちます。
3. 接続が終了すると、`socket_instances` 内のそれに対応する行が削除されます。

`socket_instances` テーブルにはこれらのカラムがあります。

- `EVENT_NAME`

イベントを生成した `wait/io/socket/*` インストゥルメントの名前。これは `setup_instruments` テーブルからの `NAME` 値です。[セクション27.6「パフォーマンススキーマインストゥルメント命名規則」](#)に説明するように、インストゥルメント名には複数の部分があり、階層を形成することがあります。

- `OBJECT_INSTANCE_BEGIN`

このカラムは一意にソケットを識別します。この値はメモリー内のオブジェクトのアドレスです。

- `THREAD_ID`

サーバーによって割り当てられた内部スレッド識別子。各ソケットは単一のスレッドによって管理されるため、各ソケットはスレッドにマップでき、スレッドはサーバープロセスにマップできます。

- `SOCKET_ID`

ソケットに割り当てられている内部ファイルハンドル。

- `IP`

クライアントの IP アドレス。この値は IPv4 または IPv6 アドレスのいずれか、または Unix ソケットファイル接続を示すブランクになります。

- **PORT**

0 から 65535 の範囲の TCP/IP ポート番号。

- **STATE**

IDLE または **ACTIVE** のいずれかのソケットステータス。アクティブなソケットの待機時間は、対応するソケットインスタルメントを使用して追跡されます。アイドルソケットの待機時間は、**idle** インスタルメントを使用して追跡されます。

ソケットはクライアントからのリクエストを待機している場合、アイドルになります。ソケットがアイドルになると、ソケットを追跡している **socket_instances** 内のイベント行が **ACTIVE** のステータスから **IDLE** に切り替わります。**EVENT_NAME** 値は **wait/io/socket/*** のままになりますが、インスタルメントのタイミングは一時停止されます。代わりに、**idle** の **EVENT_NAME** 値で **events_waits_current** テーブルにイベントが生成されます。

次のリクエストを受信すると、**idle** イベントが終了し、ソケットインスタンスが **IDLE** から **ACTIVE** に切り替わり、ソケットインスタルメントのタイミングが再開します。

socket_instances テーブルには次のインデックスがあります:

- 主キー (**OBJECT_INSTANCE_BEGIN**)
- (**THREAD_ID**) のインデックス
- (**SOCKET_ID**) のインデックス
- (**IP**, **PORT**) のインデックス

TRUNCATE TABLE は、**socket_instances** テーブルに対して許可されていません。

IP:PORT カラムの組み合わせの値は接続を識別します。この組み合わせの値は、ソケットイベントの発生元の接続を識別するために、**events_waits_xxx** テーブルの **OBJECT_NAME** カラムで使用されます。

- Unix ドメインリスナーソケット (**server_unix_socket**) の場合、ポートは 0 で IP は "" です。
- Unix ドメインリスナー経由のクライアント接続 (**client_connection**) の場合、ポートは 0 で IP は "" です。
- TCP/IP サーバリスナーソケット (**server_tcpip_socket**) の場合、ポートは常にマスターポート (たとえば 3306) で、IP は常に **0.0.0.0** です。
- TCP/IP リスナー経由のクライアント接続 (**client_connection**) の場合、ポートはサーバーが割り当てたものになりますが、0 にはなりません。IP は発信元ホストの IP (ローカルホストの場合 **127.0.0.1** または **::1**) です

27.12.4 パフォーマンススキーマ待機イベントテーブル

パフォーマンススキーマインスタルメントは待機します。これは時間がかかるイベントです。イベント階層内では、待機イベントはステージイベント内にネストされ、ステージイベントはステートメントイベント内にネストされ、ステートメントイベントはトランザクションイベント内にネストされます。

これらのテーブルは待機イベントを格納します。

- **events_waits_current**: 各スレッドの現在の待機イベント。
- **events_waits_history**: スレッドごとに終了した最新の待機イベント。
- **events_waits_history_long**: グローバルに (すべてのスレッドで) 終了した最新の待機イベント。

次の各セクションでは、待機イベントテーブルについて説明します。待機イベントに関する情報を集計するサマリーテーブルもあります。[セクション27.12.18.1「待機イベント要約テーブル」](#)を参照してください。

3つの待機イベントテーブル間の関係の詳細は、[セクション27.9「現在および過去のイベントのパフォーマンススキーマテーブル」](#)を参照してください。

待機イベント収集の構成

待機イベントを収集するかどうかを制御するには、関連するインストゥルメントおよびコンシューマの状態を設定します:

- `setup_instruments` テーブルには、`wait` で始まる名前を持つインストゥルメントが格納されます。これらのインストゥルメントを使用して、個々の待機イベントクラスの収集を有効または無効にします。
- `setup_consumers` テーブルには、現在の待機イベントテーブル名と履歴待機イベントテーブル名に対応する名前のコンシューマ値が含まれます。これらのコンシューマを使用して、待機イベントのコレクションをフィルタします。

待機インストゥルメントの中には、デフォルトで有効になっているものと無効になっているものがあります。例:

```
mysql> SELECT NAME, ENABLED, TIMED
FROM performance_schema.setup_instruments
WHERE NAME LIKE 'wait/io/file/innodb%';
+-----+-----+-----+
| NAME                                     | ENABLED | TIMED |
+-----+-----+-----+
| wait/io/file/innodb/innodb_tablespace_open_file | YES     | YES   |
| wait/io/file/innodb/innodb_data_file           | YES     | YES   |
| wait/io/file/innodb/innodb_log_file            | YES     | YES   |
| wait/io/file/innodb/innodb_temp_file           | YES     | YES   |
| wait/io/file/innodb/innodb_arch_file           | YES     | YES   |
| wait/io/file/innodb/innodb_clone_file          | YES     | YES   |
+-----+-----+-----+
mysql> SELECT NAME, ENABLED, TIMED
FROM performance_schema.setup_instruments
WHERE NAME LIKE 'wait/io/socket%';
+-----+-----+-----+
| NAME                                     | ENABLED | TIMED |
+-----+-----+-----+
| wait/io/socket/sql/server_tcpip_socket | NO      | NO    |
| wait/io/socket/sql/server_unix_socket | NO      | NO    |
| wait/io/socket/sql/client_connection  | NO      | NO    |
+-----+-----+-----+
```

待機コンシューマはデフォルトで無効にされています。

```
mysql> SELECT *
FROM performance_schema.setup_consumers
WHERE NAME LIKE 'events_waits%';
+-----+-----+
| NAME                                     | ENABLED |
+-----+-----+
| events_waits_current                    | NO      |
| events_waits_history                    | NO      |
| events_waits_history_long               | NO      |
+-----+-----+
```

サーバー起動時の待機イベント収集を制御するには、`my.cnf` ファイルで次のような行を使用します:

- 有効化:

```
[mysqld]
performance-schema-instrument='wait/%=ON'
performance-schema-consumer-events-waits-current=ON
performance-schema-consumer-events-waits-history=ON
performance-schema-consumer-events-waits-history-long=ON
```

- 無効化:

```
[mysqld]
performance-schema-instrument='wait/%=OFF'
performance-schema-consumer-events-waits-current=OFF
performance-schema-consumer-events-waits-history=OFF
```

```
performance-schema-consumer-events-waits-history-long=OFF
```

実行時の待機イベント収集を制御するには、[setup_instruments](#) テーブルと [setup_consumers](#) テーブルを更新します:

- 有効化:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'YES', TIMED = 'YES'
WHERE NAME LIKE 'wait/%';
```

```
UPDATE performance_schema.setup_consumers
SET ENABLED = 'YES'
WHERE NAME LIKE 'events_waits%';
```

- 無効化:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'NO', TIMED = 'NO'
WHERE NAME LIKE 'wait/%';
```

```
UPDATE performance_schema.setup_consumers
SET ENABLED = 'NO'
WHERE NAME LIKE 'events_waits%';
```

特定の待機イベントのみを収集するには、対応する待機インストゥルメントのみを有効にします。特定の待機イベントテーブルについてのみ待機イベントを収集するには、待機インストゥルメントを有効にし、目的のテーブルに対応する待機コンシューマのみを有効にします。

イベント収集の構成の詳細は、[セクション27.3「パフォーマンススキーマ起動構成」](#) および [セクション27.4「パフォーマンススキーマ実行時構成」](#) を参照してください。

27.12.4.1 events_waits_current テーブル

[events_waits_current](#) テーブルには、現在の待機イベントが含まれます。テーブルには、スレッドごとに最新の監視対象待機イベントの現在のステータスを示す 1 行が格納されるため、テーブルサイズを構成するためのシステム変数はありません。

待機イベント行を格納するテーブルのうち、[events_waits_current](#) はもっとも基本的です。待機イベント行を格納するほかのテーブルは論理的に現在のイベントから派生します。たとえば、[events_waits_history](#) テーブルと [events_waits_history_long](#) テーブルは、終了した最新の待機イベントのコレクションで、スレッド当たりの最大行数まで、およびすべてのスレッドにわたってグローバルに終了します。

3 つの待機イベントテーブル間の関係の詳細は、[セクション27.9「現在および過去のイベントのパフォーマンススキーマテーブル」](#) を参照してください。

待機イベントを収集するかどうかの構成の詳細は、[セクション27.12.4「パフォーマンススキーマ待機イベントテーブル」](#) を参照してください。

[events_waits_current](#) テーブルにはこれらのカラムがあります。

- [THREAD_ID](#)、[EVENT_ID](#)

イベントに関連付けられたスレッドとイベントの起動時のスレッドの現在のイベント番号。ともに取得される [THREAD_ID](#) と [EVENT_ID](#) の値によって、行が一意に識別されます。同じ値のペアを持つ行は 2 つありません。

- [END_EVENT_ID](#)

このカラムは、イベントの起動時に [NULL](#) に設定され、イベントの終了時にスレッドの現在のイベント番号に更新されます。

- [EVENT_NAME](#)

イベントを生成したインストゥルメントの名前。これは [setup_instruments](#) テーブルからの [NAME](#) 値です。[セクション27.6「パフォーマンススキーマインストゥルメント命名規則」](#) に説明するように、インストゥルメント名には複数の部分があり、階層を形成することがあります。

- [SOURCE](#)

イベントを生成した、インストゥルメントされたコードを格納するソースファイルの名前と、インストゥルメントेशनが行われたファイルの行番号。これにより、ソースをチェックして、コードに含まれるものを正確に判断することができます。たとえば、相互排他ロックまたはロックがブロックされた場合、これが発生するコンテキストをチェックできます。

- **TIMER_START**、**TIMER_END**、**TIMER_WAIT**

イベントのタイミング情報。これらの値の単位はピコ秒 (秒の 1 兆分の 1) です。**TIMER_START** および **TIMER_END** 値は、イベントのタイミングが開始されたときと終了したときを示します。**TIMER_WAIT** はイベントの経過時間 (期間) です。

イベントが終了していない場合、**TIMER_END** は現在のタイマー値で、**TIMER_WAIT** はこれまでに経過した時間です (**TIMER_END** - **TIMER_START**)。

イベントが **TIMED = NO** のインストゥルメントから生成されている場合、タイミング情報は収集されず、**TIMER_START**、**TIMER_END**、および **TIMER_WAIT** はすべて **NULL** になります。

イベント時間の単位としてのピコ秒および時間値に影響する要因については、[セクション27.4.1「パフォーマンススキーマイベントタイミング」](#)を参照してください。

- **SPINS**

相互排他ロックの場合、スピンラウンドの数。値が **NULL** の場合、コードはスピンラウンドを使用しないか、スピニングがインストゥルメントされません。

- **OBJECT_SCHEMA**、**OBJECT_NAME**、**OBJECT_TYPE**、**OBJECT_INSTANCE_BEGIN**

これらのカラムは「作用している」オブジェクトを識別します。その意味は、オブジェクトの種類によって異なります。

同期オブジェクト (**cond**、**mutex**、および **rwlock**) の場合:

- **OBJECT_SCHEMA**、**OBJECT_NAME**、および **OBJECT_TYPE** は **NULL** です。
- **OBJECT_INSTANCE_BEGIN** はメモリー内の同期オブジェクトのアドレスです。

ファイル I/O オブジェクトの場合:

- **OBJECT_SCHEMA** は **NULL** です。
- **OBJECT_NAME** はファイル名です。
- **OBJECT_TYPE** は **FILE** です。
- **OBJECT_INSTANCE_BEGIN** はメモリー内のアドレスです。

ソケットオブジェクトの場合:

- **OBJECT_NAME** はソケットの **IP:PORT** 値です。
- **OBJECT_INSTANCE_BEGIN** はメモリー内のアドレスです。

テーブル I/O オブジェクトの場合:

- **OBJECT_SCHEMA** はテーブルを格納するスキーマの名前です。
- **OBJECT_NAME** はテーブル名です。
- **OBJECT_TYPE** は永続的ベーステーブルの **TABLE** または一時テーブルの **TEMPORARY TABLE** です。
- **OBJECT_INSTANCE_BEGIN** はメモリー内のアドレスです。

OBJECT_INSTANCE_BEGIN 値自体には、さまざまな値がさまざまなオブジェクトを示すことを除いて、意味がありません。**OBJECT_INSTANCE_BEGIN** はデバッグに使用できます。たとえば、それを **GROUP BY**

`OBJECT_INSTANCE_BEGIN` で使用して、1,000 相互排他ロック (つまり、データの 1,000 ページまたはブロックを保護する) の負荷が均等に広がっているか、または少数のボトルネックだけに関わっているかを確認できます。これにより、ログファイルやほかのデバッグまたはパフォーマンスツールで同じオブジェクトアドレスが見られた場合に、情報のほかのソースと関連付けることができます。

- `INDEX_NAME`

使用されるインデックスの名前。 `PRIMARY` はテーブルプライマリインデックスを示します。 `NULL` はインデックスが使用されなかったことを意味します。

- `NESTING_EVENT_ID`

このイベントが中にネストされているイベントの `EVENT_ID` 値。

- `NESTING_EVENT_TYPE`

ネストしているイベントの種類。 値は `TRANSACTION`, `STATEMENT`, `STAGE` または `WAIT` です。

- `OPERATION`

`lock`、`read`、または `write` などの実行される操作の種類。

- `NUMBER_OF_BYTES`

操作によって読み取りまたは書き込まれるバイト数。 テーブル I/O 待機 (`wait/io/table/sql/handler` インストゥルメントのイベント) の場合、`NUMBER_OF_BYTES` は行数を示します。 値が 1 より大きい場合、イベントはバッチ I/O 操作です。 次の説明では、単一行レポートとバッチ I/O を反映するレポートの違いについて説明します。

MySQL は、ネステッドループ実装を使用して結合を実行します。 パフォーマンススキーマインストゥルメントーションのジョブは、結合内のテーブルごとに行数と累積実行時間を提供することです。 `t1`, `t2`, `t3` のテーブル結合順序を使用して実行される次の形式の結合クエリーを想定します:

```
SELECT ... FROM t1 JOIN t2 ON ... JOIN t3 ON ...
```

テーブル「ファンアウト」は、結合処理中にテーブルを追加する行数の増減です。 テーブル `t3` のファンアウトが 1 より大きい場合、行フェッチ操作の大部分はそのテーブルに対するものです。 結合が `t1` から 10 行、`t1` から `t2` から 20 行、テーブル `t2` の行ごとに `t3` から 30 行にアクセスするとします。 単一行レポートでは、インストゥルメントされた操作の合計数は次のとおりです:

```
10 + (10 * 20) + (10 * 20 * 30) = 6210
```

インストゥルメント処理される操作の数は、スキャンごと (つまり、`t1` と `t2` の行の一意の組合せごと) に集計することで大幅に削減できます。 バッチ I/O レポートでは、パフォーマンススキーマは各行ではなくもっとも内側のテーブル `t3` のスキャンごとにイベントを生成し、計測される行操作の数は次のように減少します:

```
10 + (10 * 20) + (10 * 20) = 410
```

これは 93% の削減であり、レポートコールの数を減らすことで、バッチレポート戦略によってテーブル I/O のパフォーマンススキーマのオーバーヘッドが大幅に削減される方法を示しています。 トレードオフは、イベントタイミングの精度が低くなります。 バッチ I/O のタイミングには、行ごとのレポートのように個々の行操作の時間ではなく、結合バッファリング、集計、クライアントへの行の戻しなどの操作に費やされる時間が含まれます。

バッチ I/O レポートを実行するには、次の条件が満たされている必要があります:

- クエリー実行は、クエリーブロックの最も内側のテーブルにアクセスします (単一テーブルクエリーの場合、そのテーブルは最も内側としてカウントされます)
 - クエリーの実行では、テーブルの単一行は要求されません (たとえば、`eq_ref` アクセスではバッチレポートを使用できません)
 - クエリーの実行では、テーブルに対するテーブルアクセスを含むサブクエリーは評価されません
- `FLAGS`

将来使用するために予約されています。

`events_waits_current` テーブルには次のインデックスがあります:

- 主キー (`THREAD_ID`、`EVENT_ID`)

`TRUNCATE TABLE` は `events_waits_current` テーブルに対して許可されています。行が削除されます。

27.12.4.2 `events_waits_history` テーブル

`events_waits_history` テーブルには、スレッドごとに終了した `N` の最新の待機イベントが含まれます。待機イベントはそれらが終了するまでテーブルに追加されません。テーブルに特定のスレッドの最大行数が含まれている場合、そのスレッドの新しい行が追加されると、最も古いスレッド行は破棄されます。スレッドが終了すると、そのすべての行が破棄されます。

パフォーマンススキーマは、サーバーの起動時に `N` の値を自動サイズ調整します。スレッドごとの行数を明示的に設定するには、サーバーの起動時に `performance_schema_events_waits_history_size` システム変数を設定します。

`events_waits_history` テーブルには、`events_waits_current` と同じカラムおよびインデックス付けがあります。 [セクション27.12.4.1「events_waits_current テーブル」](#) を参照してください。

`TRUNCATE TABLE` は `events_waits_history` テーブルに対して許可されています。行が削除されます。

3つの待機イベントテーブル間の関係の詳細は、 [セクション27.9「現在および過去のイベントのパフォーマンススキーマテーブル」](#) を参照してください。

待機イベントを収集するかどうかの構成の詳細は、 [セクション27.12.4「パフォーマンススキーマ待機イベントテーブル」](#) を参照してください。

27.12.4.3 `events_waits_history_long` テーブル

`events_waits_history_long` テーブルには、すべてのスレッドでグローバルに終了した最新の待機イベントが `N` に含まれます。待機イベントはそれらが終了するまでテーブルに追加されません。テーブルがいっぱいになると、どちらのスレッドがどちらの行を生成したかに関係なく、新しい行が追加されたときにもっとも古い行が破棄されます。

パフォーマンススキーマは、サーバーの起動時に `N` の値を自動サイズ調整します。テーブルサイズを明示的に設定するには、サーバー起動時に `performance_schema_events_waits_history_long_size` システム変数を設定します。

`events_waits_history_long` テーブルには、`events_waits_current` と同じカラムがあります。 [セクション27.12.4.1「events_waits_current テーブル」](#) を参照してください。 `events_waits_current` とは異なり、`events_waits_history_long` にはインデックス付けはありません。

`TRUNCATE TABLE` は `events_waits_history_long` テーブルに対して許可されています。行が削除されます。

3つの待機イベントテーブル間の関係の詳細は、 [セクション27.9「現在および過去のイベントのパフォーマンススキーマテーブル」](#) を参照してください。

待機イベントを収集するかどうかの構成の詳細は、 [セクション27.12.4「パフォーマンススキーマ待機イベントテーブル」](#) を参照してください。

27.12.5 パフォーマンススキーマステージイベントテーブル

パフォーマンススキーマは、ステートメントの解析、テーブルのオープン、`filesort` 操作の実行など、ステートメント実行プロセス中のステップであるステージを計測します。ステージは `SHOW PROCESLIST` によって表示されるか、または `INFORMATION_SCHEMA.PROCESLIST` テーブルに表示されるスレッドの状態に対応します。ステージは、状態値が変化したときに開始および終了します。

イベント階層内では、待機イベントはステージイベント内にネストされ、ステージイベントはステートメントイベント内にネストされ、ステートメントイベントはトランザクションイベント内にネストされます。

これらのテーブルはステージイベントを格納します。

- `events_stages_current`: 各スレッドの現在のステージイベント。
- `events_stages_history`: スレッドごとに終了した最新のステージイベント。
- `events_stages_history_long`: グローバルに (すべてのスレッドで) 終了した最新のステージイベント。

次の各セクションでは、ステージイベントテーブルについて説明します。ステージイベントに関する情報を集計するサマリーテーブルもあります。セクション27.12.18.2「[ステージサマリーテーブル](#)」を参照してください。

3つのステージイベントテーブル間の関係の詳細は、セクション27.9「[現在および過去のイベントのパフォーマンススキーマテーブル](#)」を参照してください。

- [ステージイベント収集の構成](#)
- [ステージイベント進捗情報](#)

ステージイベント収集の構成

ステージイベントを収集するかどうかを制御するには、関連するインストゥルメントおよびコンシューマの状態を設定します:

- `setup_instruments` テーブルには、`stage` で始まる名前を持つインストゥルメントが格納されます。これらのインストゥルメントを使用して、個々のステージイベントクラスの収集を有効または無効にします。
- `setup_consumers` テーブルには、現在のステージイベントテーブル名と履歴ステージイベントテーブル名に対応する名前前のコンシューマ値が含まれます。これらのコンシューマを使用して、ステージイベントのコレクションをフィルタします。

ステージインストゥルメントは、ステートメントの進捗情報を提供するインストゥルメント以外、デフォルトでは無効になっています。例:

```
mysql> SELECT NAME, ENABLED, TIMED
       FROM performance_schema.setup_instruments
       WHERE NAME RLIKE 'stage/sql/[a-c]';
```

NAME	ENABLED	TIMED
stage/sql/After create	NO	NO
stage/sql/allocating local table	NO	NO
stage/sql/altering table	NO	NO
stage/sql/committing alter table to storage engine	NO	NO
stage/sql/Changing master	NO	NO
stage/sql/Checking master version	NO	NO
stage/sql/checking permissions	NO	NO
stage/sql/cleaning up	NO	NO
stage/sql/closing tables	NO	NO
stage/sql/Connecting to master	NO	NO
stage/sql/converting HEAP to MyISAM	NO	NO
stage/sql/Copying to group table	NO	NO
stage/sql/Copying to tmp table	NO	NO
stage/sql/copy to tmp table	NO	NO
stage/sql/Creating sort index	NO	NO
stage/sql/creating table	NO	NO
stage/sql/Creating tmp table	NO	NO

ステートメントの進捗情報を提供するステージイベントインストゥルメントは、デフォルトで有効化され、時間設定されます:

```
mysql> SELECT NAME, ENABLED, TIMED
       FROM performance_schema.setup_instruments
       WHERE ENABLED='YES' AND NAME LIKE "stage/%";
```

NAME	ENABLED	TIMED
stage/sql/copy to tmp table	YES	YES
stage/sql/Applying batch of row changes (write)	YES	YES
stage/sql/Applying batch of row changes (update)	YES	YES

stage/sql/Applying batch of row changes (delete)	YES	YES	
stage/innodb/alter table (end)	YES	YES	
stage/innodb/alter table (flush)	YES	YES	
stage/innodb/alter table (insert)	YES	YES	
stage/innodb/alter table (log apply index)	YES	YES	
stage/innodb/alter table (log apply table)	YES	YES	
stage/innodb/alter table (merge sort)	YES	YES	
stage/innodb/alter table (read PK and internal sort)	YES	YES	
stage/innodb/buffer pool load	YES	YES	
stage/innodb/clone (file copy)	YES	YES	
stage/innodb/clone (redo copy)	YES	YES	
stage/innodb/clone (page copy)	YES	YES	
+-----+-----+-----+			

ステージコンシューマはデフォルトで無効にされています。

```
mysql> SELECT *
      FROM performance_schema.setup_consumers
      WHERE NAME LIKE 'events_stages%';
+-----+-----+
| NAME          | ENABLED |
+-----+-----+
| events_stages_current | NO      |
| events_stages_history | NO      |
| events_stages_history_long | NO     |
+-----+-----+
```

サーバー起動時のステージイベント収集を制御するには、`my.cnf` ファイルで次のような行を使用します:

- 有効化:

```
[mysqld]
performance-schema-instrument='stage/%=ON'
performance-schema-consumer-events-stages-current=ON
performance-schema-consumer-events-stages-history=ON
performance-schema-consumer-events-stages-history-long=ON
```

- 無効化:

```
[mysqld]
performance-schema-instrument='stage/%=OFF'
performance-schema-consumer-events-stages-current=OFF
performance-schema-consumer-events-stages-history=OFF
performance-schema-consumer-events-stages-history-long=OFF
```

実行時にステージイベント収集を制御するには、`setup_instruments` テーブルと `setup_consumers` テーブルを更新します:

- 有効化:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'YES', TIMED = 'YES'
WHERE NAME LIKE 'stage/%';

UPDATE performance_schema.setup_consumers
SET ENABLED = 'YES'
WHERE NAME LIKE 'events_stages%';
```

- 無効化:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'NO', TIMED = 'NO'
WHERE NAME LIKE 'stage/%';

UPDATE performance_schema.setup_consumers
SET ENABLED = 'NO'
WHERE NAME LIKE 'events_stages%';
```

特定のステージイベントのみを収集するには、対応するステージインストゥルメントのみを有効にします。特定のステージイベントテーブルに対してのみステージイベントを収集するには、ステージインストゥルメントを有効にしますが、目的のテーブルに対応するステージコンシューマのみを有効にします。

イベント収集の構成の詳細は、[セクション27.3「パフォーマンススキーマ起動構成」](#) および [セクション27.4「パフォーマンススキーマ実行時構成」](#) を参照してください。

ステージイベント進捗情報

パフォーマンススキーマのステージイベントテーブルには、ともに各行にステージ進捗インジケータを提供する2つのカラムが含まれています:

- **WORK_COMPLETED**: ステージで完了した作業ユニットの数
- **WORK_ESTIMATED**: ステージに必要な作業ユニット数

インストゥルメントの進捗情報が指定されていない場合、各カラムは **NULL** です。情報が使用可能な場合、その解釈はインストゥルメントの実装によって完全に異なります。「パフォーマンススキーマ」テーブルは、進捗データを格納するコンテナを提供しますが、メトリック自体のセマンティクスについては想定しません:

- 「作業単位」は、処理されるバイト数、行数、ファイル数、テーブル数など、実行中に時間の経過とともに増加する整数メトリックです。特定のインストゥルメントの「作業単位」の定義は、データを提供するインストゥルメンテーションコードに残されます。
- **WORK_COMPLETED** 値は、インストゥルメントされたコードに応じて、一度に1つ以上の単位を増やすことができます。
- **WORK_ESTIMATED** 値は、インストゥルメントコードに応じてステージ中に変更できます。

ステージイベント進捗インジケータのインストゥルメンテーションでは、次のいずれかの動作を実装できます:

- 進捗インストゥルメンテーションなし

これは最も一般的なケースで、進捗データは提供されません。**WORK_COMPLETED** カラムと **WORK_ESTIMATED** カラムはどちらも **NULL** です。

- 無制限進捗インストゥルメンテーション

WORK_COMPLETED カラムのみが意味を持ちます。**WORK_ESTIMATED** カラムにはデータが提供されず、0が表示されます。

監視対象セッションの **events_stages_current** テーブルをクエリーすることで、監視アプリケーションはこれまでに実行された作業量をレポートできますが、ステージが完了間近であるかどうかはレポートできません。現在、このようなステージはインストゥルメントされていません。

- 有限進捗インストゥルメンテーション

WORK_COMPLETED カラムと **WORK_ESTIMATED** カラムの両方が意味を持ちます。

このタイプの進捗インジケータは、あとで説明するテーブルコピーインストゥルメントなど、完了基準が定義された操作に適しています。監視対象セッションの **events_stages_current** テーブルをクエリーすることで、監視アプリケーションはこれまでに実行された作業量をレポートし、**WORK_COMPLETED** / **WORK_ESTIMATED** 比率を計算してステージの全体的な完了率をレポートできます。

stage/sql/copy to tmp table インストゥルメントは、進捗インジケータがどのように機能するかを示します。**ALTER TABLE** ステートメントの実行中に **stage/sql/copy to tmp table** ステージが使用され、コピーするデータのサイズによっては、このステージが長時間実行される可能性があります。

テーブルコピータスクには終了(すべての行がコピーされます)が定義されており、**stage/sql/copy to tmp table** ステージは指定されたバインド済進捗情報にインストゥルメント処理されます: 使用される作業ユニットはコピーされた行数で、**WORK_COMPLETED** と **WORK_ESTIMATED** はどちらも意味があり、その比率はタスク完了率を示します。

インストゥルメントおよび関連するコンシューマを有効にするには、次のステートメントを実行します:

```
UPDATE performance_schema.setup_instruments
SET ENABLED='YES'
WHERE NAME='stage/sql/copy to tmp table';
```

```
UPDATE performance_schema.setup_consumers
SET ENABLED='YES'
WHERE NAME LIKE 'events_stages_%';
```

進行中の `ALTER TABLE` ステートメントの進行状況を確認するには、`events_stages_current` テーブルから選択します。

27.12.5.1 events_stages_current テーブル

`events_stages_current` テーブルには、現在のステージイベントが含まれます。テーブルには、スレッドごとに最新のモニター対象ステージイベントの現在のステータスを示す 1 行が格納されるため、テーブルサイズを構成するためのシステム変数はありません。

ステージイベント行を格納するテーブルのうち、`events_stages_current` はもっとも基本的です。ステージイベント行を格納するほかのテーブルは論理的に現在のイベントから派生します。たとえば、`events_stages_history` テーブルと `events_stages_history_long` テーブルは、終了した最新のステージイベントのコレクションで、スレッド当たりの最大行数まで、およびすべてのスレッドにわたってグローバルに終了します。

3 つのステージイベントテーブル間の関係の詳細は、[セクション27.9「現在および過去のイベントのパフォーマンススキーマテーブル」](#) を参照してください。

ステージイベントを収集するかどうかの構成の詳細は、[セクション27.12.5「パフォーマンススキーマステージイベントテーブル」](#) を参照してください。

`events_stages_current` テーブルにはこれらのカラムがあります。

- `THREAD_ID`、`EVENT_ID`

イベントに関連付けられたスレッドとイベントの起動時のスレッドの現在のイベント番号。ともに取得される `THREAD_ID` と `EVENT_ID` の値によって、行が一意に識別されます。同じ値のペアを持つ行は 2 つありません。

- `END_EVENT_ID`

このカラムは、イベントの起動時に `NULL` に設定され、イベントの終了時にスレッドの現在のイベント番号に更新されます。

- `EVENT_NAME`

イベントを生成したインストゥルメントの名前。これは `setup_instruments` テーブルからの `NAME` 値です。[セクション27.6「パフォーマンススキーマインストゥルメント命名規則」](#) に説明するように、インストゥルメント名には複数の部分があり、階層を形成することがあります。

- `SOURCE`

イベントを生成した、インストゥルメントされたコードを格納するソースファイルの名前と、インストゥルメンテーションが行われたファイルの行番号。これにより、ソースをチェックして、コードに含まれるものを正確に判断することができます。

- `TIMER_START`、`TIMER_END`、`TIMER_WAIT`

イベントのタイミング情報。これらの値の単位はピコ秒 (秒の 1 兆分の 1) です。`TIMER_START` および `TIMER_END` 値は、イベントのタイミングが開始されたときと終了したときを示します。`TIMER_WAIT` はイベントの経過時間 (期間) です。

イベントが終了していない場合、`TIMER_END` は現在のタイマー値で、`TIMER_WAIT` はこれまでに経過した時間です (`TIMER_END - TIMER_START`)。

イベントが `TIMED = NO` のインストゥルメントから生成されている場合、タイミング情報は収集されず、`TIMER_START`、`TIMER_END`、および `TIMER_WAIT` はすべて `NULL` になります。

イベント時間の単位としてのピコ秒および時間値に影響する要因については、[セクション27.4.1「パフォーマンススキーマイベントタイミング」](#) を参照してください。

- [WORK_COMPLETED](#), [WORK_ESTIMATED](#)

これらのカラムには、このような情報を生成するために実装されたインストゥルメントのステージ進捗情報が表示されます。 [WORK_COMPLETED](#) はステージで完了した作業単位の数を示し、 [WORK_ESTIMATED](#) はステージで予想される作業単位の数を示します。 詳細は、 [ステージイベント進捗情報](#) を参照してください。

- [NESTING_EVENT_ID](#)

このイベントが中にネストされているイベントの [EVENT_ID](#) 値。 ステージイベントのネストしているイベントは通常ステートメントイベントです。

- [NESTING_EVENT_TYPE](#)

ネストしているイベントの種類。 値は [TRANSACTION](#), [STATEMENT](#), [STAGE](#) または [WAIT](#) です。

[events_stages_current](#) テーブルには次のインデックスがあります:

- 主キー ([THREAD_ID](#)、[EVENT_ID](#))

[TRUNCATE TABLE](#) は [events_stages_current](#) テーブルに対して許可されています。 行が削除されます。

27.12.5.2 [events_stages_history](#) テーブル

[events_stages_history](#) テーブルには、スレッドごとに終了した [N](#) の最新のステージイベントが含まれます。 ステージイベントは終了するまでテーブルに追加されません。 テーブルに特定のスレッドの最大行数が含まれている場合、そのスレッドの新しい行が追加されると、最も古いスレッド行は破棄されます。 スレッドが終了すると、そのすべての行が破棄されます。

パフォーマンススキーマは、サーバーの起動時に [N](#) の値を自動サイズ調整します。 スレッドごとの行数を明示的に設定するには、サーバーの起動時に [performance_schema_events_stages_history_size](#) システム変数を設定します。

[events_stages_history](#) テーブルには、[events_stages_current](#) と同じカラムおよびインデックス付けがあります。 [セクション27.12.5.1「events_stages_current テーブル」](#) を参照してください。

[TRUNCATE TABLE](#) は [events_stages_history](#) テーブルに対して許可されています。 行が削除されます。

3つのステージイベントテーブル間の関係の詳細は、 [セクション27.9「現在および過去のイベントのパフォーマンススキーマテーブル」](#) を参照してください。

ステージイベントを収集するかどうかの構成の詳細は、 [セクション27.12.5「パフォーマンススキーマステージイベントテーブル」](#) を参照してください。

27.12.5.3 [events_stages_history_long](#) テーブル

[events_stages_history_long](#) テーブルには、すべてのスレッドでグローバルに終了した [N](#) の最新のステージイベントが含まれます。 ステージイベントは終了するまでテーブルに追加されません。 テーブルがいっぱいになると、どちらのスレッドがどちらの行を生成したかに関係なく、新しい行が追加されたときにもっとも古い行が破棄されます。

パフォーマンススキーマは、サーバーの起動時に [N](#) の値を自動サイズ調整します。 テーブルサイズを明示的に設定するには、サーバー起動時に [performance_schema_events_stages_history_long_size](#) システム変数を設定します。

[events_stages_history_long](#) テーブルには、[events_stages_current](#) と同じカラムがあります。 [セクション27.12.5.1「events_stages_current テーブル」](#) を参照してください。 [events_stages_current](#) とは異なり、[events_stages_history_long](#) にはインデックス付けはありません。

[TRUNCATE TABLE](#) は [events_stages_history_long](#) テーブルに対して許可されています。 行が削除されます。

3つのステージイベントテーブル間の関係の詳細は、 [セクション27.9「現在および過去のイベントのパフォーマンススキーマテーブル」](#) を参照してください。

ステージイベントを収集するかどうかの構成の詳細は、 [セクション27.12.5「パフォーマンススキーマステージイベントテーブル」](#) を参照してください。

27.12.6 パフォーマンススキーマステートメントイベントテーブル

パフォーマンススキーマインストゥルメントはステートメントの実行を計測します。ステートメントイベントは、イベント階層の上位レベルで発生します。イベント階層内では、待機イベントはステージイベント内にネストされ、ステージイベントはステートメントイベント内にネストされ、ステートメントイベントはトランザクションイベント内にネストされます。

これらのテーブルはステートメントイベントを格納します。

- `events_statements_current`: 各スレッドの現在のステートメントイベント。
- `events_statements_history`: スレッドごとに終了した最新のステートメントイベント。
- `events_statements_history_long`: グローバルに (すべてのスレッドで) 終了した最新のステートメントイベント。
- `prepared_statements_instances`: プリパードステートメントのインスタンスおよび統計

次の各セクションでは、ステートメントイベントテーブルについて説明します。ステートメントイベントに関する情報を集計するサマリーテーブルもあります。セクション27.12.18.3「ステートメントサマリーテーブル」を参照してください。

3つの `events_statements_xxx` イベントテーブル間の関係の詳細は、セクション27.9「現在および過去のイベントのパフォーマンススキーマテーブル」を参照してください。

- [ステートメントイベント収集の構成](#)
- [ステートメントモニタリング](#)

ステートメントイベント収集の構成

ステートメントイベントを収集するかどうかを制御するには、関連するインストゥルメントおよびコンシューマの状態を設定します:

- `setup_instruments` テーブルには、`statement` で始まる名前を持つインストゥルメントが格納されます。これらのインストゥルメントを使用して、個々のステートメントイベントクラスの収集を有効または無効にします。
- `setup_consumers` テーブルには、現在および過去のステートメントイベントテーブル名に対応する名前を持つコンシューマ値と、ステートメントダイジェストコンシューマが含まれます。これらのコンシューマを使用して、ステートメントイベントおよびステートメントダイジェストのコレクションをフィルタします。

ステートメントインストゥルメントはデフォルトで有効になっており、`events_statements_current`、`events_statements_history` および `statements_digest` ステートメントコンシューマはデフォルトで有効になっています:

```
mysql> SELECT NAME, ENABLED, TIMED
FROM performance_schema.setup_instruments
WHERE NAME LIKE 'statement/%';
+-----+-----+-----+
| NAME                                | ENABLED | TIMED |
+-----+-----+-----+
| statement/sql/select                 | YES     | YES   |
| statement/sql/create_table           | YES     | YES   |
| statement/sql/create_index           | YES     | YES   |
| ...                                  | ...     | ...   |
| statement/sp/stmt                    | YES     | YES   |
| statement/sp/set                     | YES     | YES   |
| statement/sp/set_trigger_field       | YES     | YES   |
| statement/scheduler/event            | YES     | YES   |
| statement/com/Sleep                  | YES     | YES   |
| statement/com/Quit                   | YES     | YES   |
| statement/com/Init DB                | YES     | YES   |
| ...                                  | ...     | ...   |
| statement/abstract/Query             | YES     | YES   |
| statement/abstract/new_packet        | YES     | YES   |
| statement/abstract/relay_log        | YES     | YES   |
```

```
mysql> SELECT *
      FROM performance_schema.setup_consumers
      WHERE NAME LIKE '%statements%';
+-----+-----+
| NAME                | ENABLED |
+-----+-----+
| events_statements_current | YES     |
| events_statements_history | YES     |
| events_statements_history_long | NO     |
| statements_digest      | YES     |
+-----+-----+
```

サーバー起動時のステートメントイベント収集を制御するには、`my.cnf` ファイルで次のような行を使用します:

- 有効化:

```
[mysqld]
performance-schema-instrument='statement%=ON'
performance-schema-consumer-events-statements-current=ON
performance-schema-consumer-events-statements-history=ON
performance-schema-consumer-events-statements-history-long=ON
performance-schema-consumer-statements-digest=ON
```

- 無効化:

```
[mysqld]
performance-schema-instrument='statement%=OFF'
performance-schema-consumer-events-statements-current=OFF
performance-schema-consumer-events-statements-history=OFF
performance-schema-consumer-events-statements-history-long=OFF
performance-schema-consumer-statements-digest=OFF
```

実行時にステートメントイベント収集を制御するには、`setup_instruments` テーブルおよび `setup_consumers` テーブルを更新します:

- 有効化:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'YES', TIMED = 'YES'
WHERE NAME LIKE 'statement%';

UPDATE performance_schema.setup_consumers
SET ENABLED = 'YES'
WHERE NAME LIKE '%statements%';
```

- 無効化:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'NO', TIMED = 'NO'
WHERE NAME LIKE 'statement%';

UPDATE performance_schema.setup_consumers
SET ENABLED = 'NO'
WHERE NAME LIKE '%statements%';
```

特定のステートメントイベントのみを収集するには、対応するステートメントインストゥルメントのみを有効にします。特定のステートメントイベントテーブルについてのみステートメントイベントを収集するには、ステートメントインストゥルメントを有効にしますが、目的のテーブルに対応するステートメントコンシューマのみを有効にします。

イベント収集の構成の詳細は、[セクション27.3「パフォーマンススキーマ起動構成」](#) および [セクション27.4「パフォーマンススキーマ実行時構成」](#) を参照してください。

ステートメントモニタリング

ステートメントのモニタリングは、サーバーがスレッドに対してアクティビティーがリクエストされていることを確認した時点から、すべてのアクティビティーが終了した時点までに開始されます。一般に、これはサーバーがクライ

アントから最初のパケットを受け取ったときから、サーバーが応答の送信を終了したときまでを意味します。ストッププログラム内のステートメントは、ほかのステートメントと同様にモニターされます。

パフォーマンススキーマがリクエスト (サーバーコマンドまたは SQL ステートメント) をインストールする場合、最終的なインストール名に到達するまで、より一般的 (または「抽象的」) から、より具体的へと段階を追って進むインストール名を使用します。

最終インストール名はサーバーコマンドと SQL ステートメントに対応します。

- サーバーコマンドは `mysql_com.h` ヘッダーファイルに定義され、`sql/sql_parse.cc` で処理される `COM_xxx codes` に対応します。例は `COM_PING` と `COM_QUIT` です。コマンドのインストール名は、`statement/com/Ping` や `statement/com/Quit` などの `statement/com` から始まる名前を持ちます。
- SQL ステートメントは `DELETE FROM t1` または `SELECT * FROM t2` などのテキストとして表されます。SQL ステートメントのインストール名は、`statement/sql/delete` や `statement/sql/select` などの `statement/sql` から始まる名前を持ちます。

いくつかの最終インストール名はエラー処理に固有です。

- `statement/com/Error` は帯域外のサーバーによって受信されたメッセージから構成されます。これはサーバーが理解しないクライアントによって送信されたコマンドを検出するために使用できます。これは、構成が誤っているか、サーバーよりも新しい MySQL のバージョンを使用しているクライアントや、サーバーへの攻撃を試みているクライアントの識別などの目的で役に立つことがあります。
- `statement/sql/error` は解析に失敗した SQL ステートメントから構成されます。これはクライアントによって送信された不正な形式のクエリーを検出するために使用できます。解析に失敗するクエリーは、解析するが、実行中のエラーのために失敗するクエリーと異なります。たとえば、`SELECT * FROM` は不正な形式で、`statement/sql/error` インストール名が使用されます。対照的に `SELECT *` は解析しますが、「表が指定されていません」エラーを伴って失敗します。この場合、`statement/sql/select` が使用され、ステートメントイベントにはエラーの性質を示す情報が含まれます。

リクエストはこれらの任意のソースから取得できます。

- リクエストをパケットとして送信するクライアントからのコマンドまたはステートメントリクエストとして
- レプリカのリレーログから読み取られたステートメント文字列として
- イベントスケジューラのイベントとして

リクエストの詳細は最初は不明で、パフォーマンススキーマはリクエストのソースに依存する順序で、抽象から特定のインストール名に進みます。

クライアントから受信したリクエストの場合:

1. サーバーがソケットレベルで新しいパケットを検出すると、新しいステートメントが `statement/abstract/new_packet` の抽象インストール名で開始されます。
2. サーバーはパケット番号を読み取ると、受信したリクエストの種類について詳しく知り、パフォーマンススキーマがインストール名を絞り込みます。たとえば、リクエストが `COM_PING` パケットの場合、インストール名は `statement/com/Ping` になり、それが最終名になります。リクエストが `COM_QUERY` パケットの場合、特定のタイプのステートメントではなく SQL ステートメントに対応することがわかっています。この場合、インストール名はある抽象名から、やや具体的だが、まだ抽象名である `statement/abstract/Query` に変更され、リクエストはさらに分類する必要があります。
3. リクエストがステートメントである場合、ステートメントテキストが読み取られ、パーサーに提供されます。解析後、正確なステートメントの種類が認識されます。リクエストがたとえば `INSERT` ステートメントの場合、パフォーマンススキーマはインストール名を `statement/abstract/Query` から最終名である `statement/sql/insert` に絞り込みます。

レプリカのリレーログからステートメントとして読み取られたリクエストの場合:

1. リレーログ内のステートメントはテキストとして保存され、そのように読み取られます。ネットワークプロトコルはないため、`statement/abstract/new_packet` インストール名は使用されません。代わりに、初期インストール名は `statement/abstract/relay_log` になります。

- ステートメントが解析されると、正確なステートメントの種類が認識されます。リクエストがたとえば `INSERT` ステートメントの場合、パフォーマンススキーマはインストゥルメント名を `statement/abstract/Query` から最終名である `statement/sql/insert` に絞り込みます。

先述の説明はステートメントベースのレプリケーションにのみ適用されます。行ベースレプリケーションの場合、行の変更を処理するレプリカで実行されるテーブル I/O は計測できますが、リレーログ内の行イベントは個別のステートメントとして表示されません。

イベントスケジューラから受信したリクエストの場合:

イベント実行は、`statement/scheduler/event` という名前を使用してインストゥルメントされます。これは最終名です。

イベント本体内で実行されるステートメントは、前の抽象インストゥルメントを使用せずに、`statement/sql/*` 名を使用してインストゥルメントされます。イベントはストアプログラムであり、ストアプログラムは実行前にメモリ内でプリコンパイルされます。したがって、実行時に解析は行われず、各ステートメントのタイプは実行時に認識されます。

イベント本体内で実行されるステートメントは子ステートメントです。たとえば、イベントが `INSERT` ステートメントを実行する場合、イベント自体の実行は親で、`statement/scheduler/event` を使用してインストゥルメント処理され、`INSERT` は `statement/sql/insert` を使用してインストゥルメント処理された子です。親子関係には、`between` の個別のインストゥルメントされた操作が保持されます。これは、範囲内で単一のインストゥルメント操作が発生する絞込みのシーケンスとは異なり、抽象インストゥルメント名から最終インストゥルメント名まで異なります。

ステートメントに対して収集される統計の場合、各ステートメントの種類に使用される最終 `statement/sql/*` インストゥルメントを有効にするだけでは十分ではありません。抽象 `statement/abstract/*` インストゥルメントも有効にする必要があります。すべてのステートメントインストゥルメントがデフォルトで有効にされるため、これは通常問題にならないはずですが、ただし、ステートメントインストゥルメントを選択して有効または無効にするアプリケーションは、抽象インストゥルメントを無効にすると、個々のステートメントインストゥルメントの統計収集も無効になることを考慮する必要があります。たとえば、`INSERT` ステートメントの統計を収集するには、`statement/sql/insert` を有効にする必要がありますが、`statement/abstract/new_packet` と `statement/abstract/Query` も有効にする必要があります。同様に、レプリケートされたステートメントをインストゥルメントするには、`statement/abstract/relay_log` が有効にされている必要があります。

ステートメントが最終ステートメント名として抽象インストゥルメントに分類されることはないため、`statement/abstract/Query` などの抽象インストゥルメントに対して統計は集計されません。

27.12.6.1 events_statements_current テーブル

`events_statements_current` テーブルには、現在のステートメントイベントが含まれます。テーブルには、スレッドごとに最新のモニター対象ステートメントイベントの現在のステータスを示す 1 行が格納されるため、テーブルサイズを構成するためのシステム変数はありません。

ステートメントイベント行を格納するテーブルのうち、`events_statements_current` はもっとも基本的です。ステートメントイベント行を格納するほかのテーブルは論理的に現在のイベントから派生します。たとえば、`events_statements_history` テーブルと `events_statements_history_long` テーブルは、終了した最新のステートメントイベントのコレクションで、スレッド当たりの最大行数まで、およびすべてのスレッドにわたってグローバルに終了します。

3 つの `events_statements_xxx` イベントテーブル間の関係の詳細は、[セクション27.9「現在および過去のイベントのパフォーマンススキーマテーブル」](#) を参照してください。

ステートメントイベントを収集するかどうかの構成については、[セクション27.12.6「パフォーマンススキーマステートメントイベントテーブル」](#) を参照してください。

`events_statements_current` テーブルにはこれらのカラムがあります。

- `THREAD_ID`、`EVENT_ID`

イベントに関連付けられたスレッドとイベントの起動時のスレッドの現在のイベント番号。ともに取得される `THREAD_ID` と `EVENT_ID` の値によって、行が一意に識別されます。同じ値のペアを持つ行は 2 つありません。

- `END_EVENT_ID`

このカラムは、イベントの起動時に `NULL` に設定され、イベントの終了時にスレッドの現在のイベント番号に更新されます。

- `EVENT_NAME`

イベントが収集されたインストゥルメントの名前。これは `setup_instruments` テーブルからの `NAME` 値です。セクション27.6「パフォーマンススキーマインストゥルメント命名規則」に説明するように、インストゥルメント名には複数の部分があり、階層を形成することがあります。

SQL ステートメントの場合、ステートメントが解析されるまで、`EVENT_NAME` 値は最初 `statement/com/Query` であり、その後セクション27.12.6「パフォーマンススキーマステートメントイベントテーブル」に説明するように、より適切な値に変更されます。

- `SOURCE`

イベントを生成した、インストゥルメントされたコードを格納するソースファイルの名前と、インストゥルメントेशनが行われたファイルの行番号。これにより、ソースをチェックして、コードに含まれるものを正確に判断することができます。

- `TIMER_START`、`TIMER_END`、`TIMER_WAIT`

イベントのタイミング情報。これらの値の単位はピコ秒 (秒の 1 兆分の 1) です。`TIMER_START` および `TIMER_END` 値は、イベントのタイミングが開始されたときと終了したときを示します。`TIMER_WAIT` はイベントの経過時間 (期間) です。

イベントが終了していない場合、`TIMER_END` は現在のタイマー値で、`TIMER_WAIT` はこれまでに経過した時間です (`TIMER_END - TIMER_START`)。

イベントが `TIMED = NO` のインストゥルメントから生成されている場合、タイミング情報は収集されず、`TIMER_START`、`TIMER_END`、および `TIMER_WAIT` はすべて `NULL` になります。

イベント時間の単位としてのピコ秒および時間値に影響する要因については、セクション27.4.1「パフォーマンススキーマイベントタイミング」を参照してください。

- `LOCK_TIME`

テーブルロックの待機に費やされた時間。この値はマイクロ秒で計算されますが、ほかのパフォーマンススキーマタイマーとの比較を容易にするため、ピコ秒に正規化されます。

- `SQL_TEXT`

SQL ステートメントのテキスト。SQL ステートメントに関連付けられていないコマンドの場合、値は `NULL` です。

ステートメントの表示に使用できる最大領域は、デフォルトで 1024 バイトです。この値を変更するには、サーバーの起動時に `performance_schema_max_sql_text_length` システム変数を設定します。(この値を変更すると、他の「パフォーマンススキーマ」テーブルのカラムにも影響します。セクション27.10「パフォーマンススキーマのステートメントダイジェストとサンプリング」を参照してください。)

- `DIGEST`

ステートメントは、64 16 進数文字の文字列、または `statements_digest` コンシューマが `no` の場合は `NULL` として SHA-256 値をダイジェストします。ステートメントダイジェストの詳細については、セクション27.10「パフォーマンススキーマのステートメントダイジェストとサンプリング」を参照してください。

- `DIGEST_TEXT`

正規化されたステートメントダイジェストテキスト (`statements_digest` コンシューマが `no` の場合は `NULL`)。ステートメントダイジェストの詳細については、セクション27.10「パフォーマンススキーマのステートメントダイジェストとサンプリング」を参照してください。

`performance_schema_max_digest_length` システム変数は、ダイジェスト値の格納に使用できるセッション当たりの最大バイト数を決定します。ただし、ダイジェストバッファ内のキーワードやリテラル値などのステートメン

ト要素のエンコーディングにより、ステートメントダイジェストの表示長が使用可能なバッファサイズより長くなる場合があります。したがって、ステートメントイベントテーブルの `DIGEST_TEXT` カラムから選択された値は、`performance_schema_max_digest_length` 値を超えるように見える場合があります。

- `CURRENT_SCHEMA`

ステートメントのデフォルトのデータベース、何も無い場合は `NULL`。

- `OBJECT_SCHEMA`、`OBJECT_NAME`、`OBJECT_TYPE`

ネストしたステートメント (ストアドプログラム) の場合、これらのカラムには親ステートメントに関する情報が含まれます。それ以外の場合は、`NULL` です。

- `OBJECT_INSTANCE_BEGIN`

このカラムはステートメントを識別します。この値はメモリー内のオブジェクトのアドレスです。

- `MYSQL_ERRNO`

ステートメント診断領域からのステートメントエラー番号。

- `RETURNED_SQLSTATE`

ステートメント診断領域からのステートメント `SQLSTATE` 値。

- `MESSAGE_TEXT`

ステートメント診断領域からのステートメントエラーメッセージ。

- `ERRORS`

ステートメントにエラーが発生したかどうか。 `SQLSTATE` 値が `00` (完了) または `01` (警告) から始まる場合、値は `0` です。 `SQLSTATE` 値がほかの値の場合、値は `1` です。

- `WARNINGS`

ステートメント診断領域からの警告数。

- `ROWS_AFFECTED`

ステートメントに影響を受けた行数。「影響を受けた」の意味については、`mysql_affected_rows()` を参照してください。

- `ROWS_SENT`

ステートメントから返された行数。

- `ROWS_EXAMINED`

サーバーレイヤーによって検査された行数 (ストレージエンジン内部の処理はカウントされません)。

- `CREATED_TMP_DISK_TABLES`

`Created_tmp_disk_tables` ステータス変数と同様ですが、ステートメントに固有です。

- `CREATED_TMP_TABLES`

`Created_tmp_tables` ステータス変数と同様ですが、ステートメントに固有です。

- `SELECT_FULL_JOIN`

`Select_full_join` ステータス変数と同様ですが、ステートメントに固有です。

- `SELECT_FULL_RANGE_JOIN`

`Select_full_range_join` ステータス変数と同様ですが、ステートメントに固有です。

- **SELECT_RANGE**
`Select_range` ステータス変数と同様ですが、ステートメントに固有です。
- **SELECT_RANGE_CHECK**
`Select_range_check` ステータス変数と同様ですが、ステートメントに固有です。
- **SELECT_SCAN**
`Select_scan` ステータス変数と同様ですが、ステートメントに固有です。
- **SORT_MERGE_PASSES**
`Sort_merge_passes` ステータス変数と同様ですが、ステートメントに固有です。
- **SORT_RANGE**
`Sort_range` ステータス変数と同様ですが、ステートメントに固有です。
- **SORT_ROWS**
`Sort_rows` ステータス変数と同様ですが、ステートメントに固有です。
- **SORT_SCAN**
`Sort_scan` ステータス変数と同様ですが、ステートメントに固有です。
- **NO_INDEX_USED**
ステートメントがインデックスを使用せずにテーブルスキャンを実行した場合は 1、そうでない場合は 0。
- **NO_GOOD_INDEX_USED**
サーバーがステートメントに使用する適切なインデックスを見つけられなかった場合は 1、そうでない場合は 0。追加の情報については、[セクション8.8.2「EXPLAIN 出力フォーマット」](#)で、EXPLAIN 出力の `Extra` カラムの `Range checked for each record` 値の説明を参照してください。
- **NESTING_EVENT_ID, NESTING_EVENT_TYPE, NESTING_EVENT_LEVEL**
これらの 3 つのカラムは、最上位 (ネストされていない) ステートメントおよびネストされたステートメント (ストアードプログラム内で実行される) に関する次のような情報を提供するために、他のカラムとともに使用されます。
トップレベルのステートメントの場合:

```
OBJECT_TYPE = NULL
OBJECT_SCHEMA = NULL
OBJECT_NAME = NULL
NESTING_EVENT_ID = NULL
NESTING_EVENT_TYPE = NULL
NESTING_LEVEL = 0
```

ネストしたステートメントの場合:

```
OBJECT_TYPE = the parent statement object type
OBJECT_SCHEMA = the parent statement object schema
OBJECT_NAME = the parent statement object name
NESTING_EVENT_ID = the parent statement EVENT_ID
NESTING_EVENT_TYPE = 'STATEMENT'
NESTING_LEVEL = the parent statement NESTING_LEVEL plus one
```
- **STATEMENT_ID**
SQL レベルでサーバーによって保持されるクエリー ID。これらの ID はアトミックに増分されるグローバルカウンタを使用して生成されるため、この値はサーバーインスタンスに対して一意です。このカラムは、MySQL 8.0.14 で追加されました。

`events_statements_current` テーブルには次のインデックスがあります:

- 主キー (THREAD_ID、EVENT_ID)

TRUNCATE TABLE は `events_statements_current` テーブルに対して許可されています。行が削除されます。

27.12.6.2 events_statements_history テーブル

`events_statements_history` テーブルには、スレッドごとに終了した `N` の最新のステートメントイベントが含まれます。ステートメントイベントは終了するまでテーブルに追加されません。テーブルに特定のスレッドの最大行数が含まれている場合、そのスレッドの新しい行が追加されると、最も古いスレッド行は破棄されます。スレッドが終了すると、そのすべての行が破棄されます。

パフォーマンススキーマは、サーバーの起動時に `N` の値を自動サイズ調整します。スレッドごとの行数を明示的に設定するには、サーバーの起動時に `performance_schema_events_statements_history_size` システム変数を設定します。

`events_statements_history` テーブルには、`events_statements_current` と同じカラムおよびインデックス付けがあります。セクション27.12.6.1「`events_statements_current` テーブル」を参照してください。

TRUNCATE TABLE は `events_statements_history` テーブルに対して許可されています。行が削除されます。

3つの `events_statements_xxx` イベントテーブル間の関係の詳細は、セクション27.9「現在および過去のイベントのパフォーマンススキーマテーブル」を参照してください。

ステートメントイベントを収集するかどうかの構成については、セクション27.12.6「パフォーマンススキーマステートメントイベントテーブル」を参照してください。

27.12.6.3 events_statements_history_long テーブル

`events_statements_history_long` テーブルには、すべてのスレッドでグローバルに終了した `N` の最新のステートメントイベントが含まれます。ステートメントイベントは終了するまでテーブルに追加されません。テーブルがいっぱいになると、どちらのスレッドがどちらの行を生成したかに関係なく、新しい行が追加されたときにもっとも古い行が破棄されます。

`N` の値はサーバー起動時に自動サイズ設定されます。テーブルサイズを明示的に設定するには、サーバー起動時に `performance_schema_events_statements_history_long_size` システム変数を設定します。

`events_statements_history_long` テーブルには、`events_statements_current` と同じカラムがあります。セクション27.12.6.1「`events_statements_current` テーブル」を参照してください。`events_statements_current` とは異なり、`events_statements_history_long` にはインデックス付けはありません。

TRUNCATE TABLE は `events_statements_history_long` テーブルに対して許可されています。行が削除されます。

3つの `events_statements_xxx` イベントテーブル間の関係の詳細は、セクション27.9「現在および過去のイベントのパフォーマンススキーマテーブル」を参照してください。

ステートメントイベントを収集するかどうかの構成については、セクション27.12.6「パフォーマンススキーマステートメントイベントテーブル」を参照してください。

27.12.6.4 prepared_statements_instances テーブル

パフォーマンススキーマはプリペアドステートメントのインストゥルメンテーションを提供しますが、次の2つのプロトコルがあります:

- バイナリプロトコル。これには MySQL C API を介してアクセスし、次のテーブルに示すように基礎となるサーバーコマンドにマップします。

C API 関数	対応するサーバーコマンド
<code>mysql_stmt_prepare()</code>	<code>COM_STMT_PREPARE</code>
<code>mysql_stmt_execute()</code>	<code>COM_STMT_EXECUTE</code>

C API 関数	対応するサーバーコマンド
mysql_stmt_close()	COM_STMT_CLOSE

- テキストプロトコル。これには SQL ステートメントを使用してアクセスし、次のテーブルに示すように基礎となるサーバーコマンドにマップします。

SQL ステートメント	対応するサーバーコマンド
PREPARE	SQLCOM_PREPARE
EXECUTE	SQLCOM_EXECUTE
DEALLOCATE PREPARE, DROP PREPARE	SQLCOM_DEALLOCATE PREPARE

パフォーマンススキーマプリペアドステートメントインストールメンテーションは、両方のプロトコルに対応しています。次の説明では、C API 関数または SQL ステートメントではなくサーバーコマンドについて説明します。

プリペアドステートメントに関する情報は、[prepared_statements_instances](#) テーブルにあります。このテーブルは、サーバーで使用されるプリペアドステートメントの検査を有効にし、それらに関する集計された統計を提供します。このテーブルのサイズを制御するには、サーバーの起動時に [performance_schema_max_prepared_statements_instances](#) システム変数を設定します。

プリペアドステートメント情報の収集は、次のテーブルに示すステートメントインストールメントによって異なります。これらのインストールメントはデフォルトで有効になっています。これらを変更するには、[setup_instruments](#) テーブルを更新します。

金融商品	サーバーコマンド
statement/com/Prepare	COM_STMT_PREPARE
statement/com/Execute	COM_STMT_EXECUTE
statement/sql/prepare_sql	SQLCOM_PREPARE
statement/sql/execute_sql	SQLCOM_EXECUTE

パフォーマンススキーマは、[prepared_statements_instances](#) テーブルの内容を次のように管理します：

- ステートメントの準備

[COM_STMT_PREPARE](#) または [SQLCOM_PREPARE](#) コマンドは、サーバーにプリペアドステートメントを作成します。ステートメントが正常にインストールメント処理されると、新しい行が [prepared_statements_instances](#) テーブルに追加されます。ステートメントを計測できない場合は、[Performance_schema_prepared_statements_lost](#) ステータス変数が増分されます。

- プリペアドステートメントの実行

インストールメント処理されたプリペアドステートメントインスタンスに対して [COM_STMT_EXECUTE](#) または [SQLCOM_PREPARE](#) コマンドを実行すると、対応する [prepared_statements_instances](#) テーブルの行が更新されません。

- プリペアドステートメントの割当て解除

インストールメント処理されたプリペアドステートメントインスタンスに対して [COM_STMT_CLOSE](#) または [SQLCOM_DEALLOCATE_PREPARE](#) コマンドを実行すると、対応する [prepared_statements_instances](#) テーブルの行が削除されます。リソースリークを回避するために、前述のプリペアドステートメントインストールメントが無効になっていても、削除が行われます。

[prepared_statements_instances](#) テーブルには、次のカラムがあります：

- [OBJECT_INSTANCE_BEGIN](#)

インストールメントされたプリペアドステートメントのメモリー内のアドレス。

- [STATEMENT_ID](#)

サーバーによって割り当てられた内部ステートメント ID。テキストプロトコルとバイナリプロトコルはどちらもステートメント ID を使用します。

- **STATEMENT_NAME**

バイナリプロトコルの場合、このカラムは **NULL** です。テキストプロトコルの場合、このカラムはユーザーによって割り当てられた外部ステートメントの名前です。たとえば、次の SQL ステートメントの場合、プリペアドステートメントの名前は **stmt** です:

```
PREPARE stmt FROM 'SELECT 1';
```

- **SQL_TEXT**

? プレースホルダマーカーを使用したプリペアドステートメントのテキスト。

- **OWNER_THREAD_ID, OWNER_EVENT_ID**

これらのカラムは、プリペアドステートメントを作成したイベントを示します。

- **OWNER_OBJECT_TYPE, OWNER_OBJECT_SCHEMA, OWNER_OBJECT_NAME**

クライアントセッションによって作成されたプリペアドステートメントの場合、これらのカラムは **NULL** です。ストアードプログラムによって作成されたプリペアドステートメントの場合、これらのカラムはストアードプログラムを指します。一般的なユーザーエラーは、プリペアドステートメントの割当て解除を忘れたことです。これらのカラムを使用すると、プリペアドステートメントをリークするストアードプログラムを見つけることができます:

```
SELECT  
  OWNER_OBJECT_TYPE, OWNER_OBJECT_SCHEMA, OWNER_OBJECT_NAME,  
  STATEMENT_NAME, SQL_TEXT  
FROM performance_schema.prepared_statements_instances  
WHERE OWNER_OBJECT_TYPE IS NOT NULL;
```

- **TIMER_PREPARE**

ステートメントの準備自体の実行に要した時間。

- **COUNT_REPREPARE**

ステートメントが内部的に再準備された回数 ([セクション8.10.3「プリペアドステートメントおよびストアードプログラムのキャッシュ」](#)を参照)。再準備のタイミング統計は、個別の操作としてではなくステートメントの実行の一部としてカウントされるため、使用できません。

- **COUNT_EXECUTE, SUM_TIMER_EXECUTE, MIN_TIMER_EXECUTE, AVG_TIMER_EXECUTE, MAX_TIMER_EXECUTE**

プリペアドステートメントの実行の集計統計。

- **SUM_xxx**

残りの **SUM_xxx** カラムは、ステートメントサマリーテーブルと同じです ([セクション27.12.18.3「ステートメントサマリーテーブル」](#)を参照)。

prepared_statements_instances テーブルには次のインデックスがあります:

- 主キー (**OBJECT_INSTANCE_BEGIN**)
- (**STATEMENT_ID**) のインデックス
- (**STATEMENT_NAME**) のインデックス
- (**OWNER_THREAD_ID, OWNER_EVENT_ID**) のインデックス
- (**OWNER_OBJECT_TYPE, OWNER_OBJECT_SCHEMA, OWNER_OBJECT_NAME**) のインデックス

TRUNCATE TABLE により、**prepared_statements_instances** テーブルの統計カラムがリセットされます。

27.12.7 パフォーマンススキーマのトランザクションテーブル

パフォーマンススキーマはトランザクションを計測します。イベント階層内では、待機イベントはステージイベント内にネストされ、ステージイベントはステートメントイベント内にネストされ、ステートメントイベントはトランザクションイベント内にネストされます。

次のテーブルには、トランザクションイベントが格納されます:

- `events_transactions_current`: 各スレッドの現在のトランザクションイベント。
- `events_transactions_history`: スレッドごとに終了した最新のトランザクションイベント。
- `events_transactions_history_long`: グローバルに (すべてのスレッドで) 終了した最新のトランザクションイベント。

次の各セクションでは、トランザクションイベントテーブルについて説明します。トランザクションイベントに関する情報を集計するサマリーテーブルもあります。セクション27.12.18.5「トランザクション要約テーブル」を参照してください。

3つのトランザクションイベントテーブル間の関係の詳細は、セクション27.9「現在および過去のイベントのパフォーマンススキーマテーブル」を参照してください。

- [トランザクションイベント収集の構成](#)
- [トランザクション境界](#)
- [トランザクション手段](#)
- [トランザクションおよびネストされたイベント](#)
- [トランザクションとストアプログラム](#)
- [トランザクションとセーブポイント](#)
- [トランザクションおよびエラー](#)

トランザクションイベント収集の構成

トランザクションイベントを収集するかどうかを制御するには、関連するインストゥルメントおよびコンシューマの状態を設定します:

- `setup_instruments` テーブルには、`transaction` という名前のインストゥルメントが含まれています。このインストゥルメントを使用して、個々のトランザクションイベントクラスの収集を有効または無効にします。
- `setup_consumers` テーブルには、現在のトランザクションイベントテーブル名と履歴トランザクションイベントテーブル名に対応する名前のコンシューマ値が含まれます。これらのコンシューマを使用して、トランザクションイベントのコレクションをフィルタします。

`transaction` インストゥルメント、`events_transactions_current` および `events_transactions_history` トランザクションコンシューマは、デフォルトで有効になっています:

```
mysql> SELECT NAME, ENABLED, TIMED
      FROM performance_schema.setup_instruments
      WHERE NAME = 'transaction';
+-----+-----+-----+
| NAME      | ENABLED | TIMED |
+-----+-----+-----+
| transaction | YES     | YES   |
+-----+-----+-----+
mysql> SELECT *
      FROM performance_schema.setup_consumers
      WHERE NAME LIKE 'events_transactions%';
+-----+-----+
| NAME                | ENABLED |
+-----+-----+
| events_transactions_current | YES     |
| events_transactions_history  | YES     |
```

```
| events_transactions_history_long | NO |  
+-----+-----+
```

サーバー起動時のトランザクションイベント収集を制御するには、`my.cnf` ファイルで次のような行を使用します:

- 有効化:

```
[mysqld]  
performance-schema-instrument='transaction=ON'  
performance-schema-consumer-events-transactions-current=ON  
performance-schema-consumer-events-transactions-history=ON  
performance-schema-consumer-events-transactions-history-long=ON
```

- 無効化:

```
[mysqld]  
performance-schema-instrument='transaction=OFF'  
performance-schema-consumer-events-transactions-current=OFF  
performance-schema-consumer-events-transactions-history=OFF  
performance-schema-consumer-events-transactions-history-long=OFF
```

実行時にトランザクションイベント収集を制御するには、`setup_instruments` テーブルと `setup_consumers` テーブルを更新します:

- 有効化:

```
UPDATE performance_schema.setup_instruments  
SET ENABLED = 'YES', TIMED = 'YES'  
WHERE NAME = 'transaction';  
  
UPDATE performance_schema.setup_consumers  
SET ENABLED = 'YES'  
WHERE NAME LIKE 'events_transactions%';
```

- 無効化:

```
UPDATE performance_schema.setup_instruments  
SET ENABLED = 'NO', TIMED = 'NO'  
WHERE NAME = 'transaction';  
  
UPDATE performance_schema.setup_consumers  
SET ENABLED = 'NO'  
WHERE NAME LIKE 'events_transactions%';
```

特定のトランザクションイベントテーブルに対してのみトランザクションイベントを収集するには、`transaction` インストルメントを有効にしますが、目的のテーブルに対応するトランザクションコンシューマのみを有効にします。

イベント収集の構成の詳細は、[セクション27.3「パフォーマンススキーマ起動構成」](#) および [セクション27.4「パフォーマンススキーマ実行時構成」](#) を参照してください。

トランザクション境界

MySQL Server では、トランザクションは次のステートメントで明示的に開始されます:

```
START TRANSACTION | BEGIN | XA START | XA BEGIN
```

トランザクションも暗黙的に開始されます。たとえば、`autocommit` システム変数が有効になっている場合、各ステートメントの開始によって新しいトランザクションが開始されます。

`autocommit` が無効になっている場合、コミットされたトランザクションに続く最初のステートメントによって、新しいトランザクションの開始がマークされます。後続のステートメントは、コミットされるまでトランザクションの一部です。

トランザクションは、次のステートメントで明示的に終了します:

```
COMMIT | ROLLBACK | XA COMMIT | XA ROLLBACK
```

トランザクションは、DDL ステートメント、ロックステートメントおよびサーバー管理ステートメントの実行によっても暗黙的に終了します。

次の説明では、[START TRANSACTION](#) への参照は [BEGIN](#)、[XA START](#) および [XA BEGIN](#) にも適用されます。同様に、[COMMIT](#) および [ROLLBACK](#) への参照は、それぞれ [XA COMMIT](#) および [XA ROLLBACK](#) に適用されます。

パフォーマンススキーマは、サーバーと同様のトランザクション境界を定義します。トランザクションイベントの開始と終了は、サーバー内の対応する状態遷移と密接に一致します:

- 明示的に開始されたトランザクションの場合、トランザクションイベントは [START TRANSACTION](#) ステートメントの処理中に開始されます。
- 暗黙的に開始されたトランザクションの場合、トランザクションイベントは、前のトランザクションの終了後にトランザクションエンジンを使用する最初のステートメントで開始されます。
- トランザクションについては、明示的に終了したか暗黙的に終了したかにかかわらず、[COMMIT](#) または [ROLLBACK](#) の処理中にサーバーがアクティブなトランザクション状態から遷移すると、トランザクションイベントが終了します。

このアプローチには微妙な影響があります:

- パフォーマンススキーマ内のトランザクションイベントには、対応する [START TRANSACTION](#)、[COMMIT](#)、または [ROLLBACK](#) ステートメントに関連付けられたステートメントイベントが完全には含まれていません。トランザクションイベントとこれらのステートメントの間には、わずかなタイミングの重複があります。
- 非トランザクションエンジンで動作するステートメントは、接続のトランザクション状態には影響しません。暗黙的トランザクションの場合、トランザクションイベントはトランザクションエンジンを使用する最初のステートメントから始まります。つまり、非トランザクションテーブルで排他的に動作するステートメントは、[START TRANSACTION](#) に続く場合でも無視されます。

説明するために、次のシナリオを考えてみます:

```
1. SET autocommit = OFF;
2. CREATE TABLE t1 (a INT) ENGINE = InnoDB;
3. START TRANSACTION;           -- Transaction 1 START
4. INSERT INTO t1 VALUES (1), (2), (3);
5. CREATE TABLE t2 (a INT) ENGINE = MyISAM; -- Transaction 1 COMMIT
   -- (implicit; DDL forces commit)
6. INSERT INTO t2 VALUES (1), (2), (3); -- Update nontransactional table
7. UPDATE t2 SET a = a + 1;         -- ... and again
8. INSERT INTO t1 VALUES (4), (5), (6); -- Write to transactional table
   -- Transaction 2 START (implicit)
9. COMMIT;                       -- Transaction 2 COMMIT
```

サーバーの観点からは、トランザクション 1 はテーブル `t2` が作成されると終了します。トランザクション 2 は、非トランザクションテーブルへの更新が介在しているにもかかわらず、トランザクションテーブルにアクセスするまで開始されません。

パフォーマンススキーマの観点からは、サーバーがアクティブなトランザクション状態に遷移すると、トランザクション 2 が開始されます。ステートメント 6 および 7 はトランザクション 2 の境界内に含まれません。これは、サーバーがバイナリログにトランザクションを書き込む方法と一致します。

トランザクション手段

トランザクションは、次の 3 つの属性で定義されます:

- アクセスモード (読取り専用、読取り/書き込み)
- 分離レベル ([SERIALIZABLE](#)、[REPEATABLE READ](#) など)
- 暗黙 ([autocommit](#) が有効) または明示 ([autocommit](#) が無効)

トランザクションインストゥルメンテーションの複雑さを軽減し、収集されたトランザクションデータが完全で意味のある結果を提供するようにするために、すべてのトランザクションはアクセスモード、分離レベルまたは自動コミットモードとは無関係にインストゥルメントされます。

トランザクション履歴を選択的に調べるには、トランザクションイベントテーブルの属性カラムを使用: [ACCESS_MODE](#)、[ISOLATION_LEVEL](#) および [AUTOCOMMIT](#)。

トランザクションインストゥルメンテーションのコストは、ユーザー、アカウント、ホストまたはスレッド (クライアント接続) に応じたトランザクションインストゥルメンテーションの有効化または無効化など、様々な方法で削減できます。

トランザクションおよびネストされたイベント

トランザクションイベントの親は、トランザクションを開始したイベントです。明示的に開始されたトランザクションの場合、これには **START TRANSACTION** および **COMMIT AND CHAIN** ステートメントが含まれます。暗黙的に開始されたトランザクションの場合は、前のトランザクションの終了後にトランザクションエンジンを使用する最初のステートメントです。

一般に、トランザクションは、**COMMIT** や **ROLLBACK** などのトランザクションを明示的に終了するステートメントを含め、トランザクション中に開始されるすべてのイベントの最上位レベルの親です。例外は、DDL ステートメントなど、トランザクションを暗黙的に終了するステートメントです。この場合、新しいステートメントを実行する前に現在のトランザクションをコミットする必要があります。

トランザクションとストアドプログラム

トランザクションおよびストアドプログラムイベントは、次のように関連付けられます：

- ストアドプロシージャ

ストアドプロシージャは、トランザクションとは無関係に動作します。ストアドプロシージャはトランザクション内で開始でき、トランザクションはストアドプロシージャ内から開始または終了できます。ストアドプロシージャは、トランザクション内からコールされた場合、親トランザクションのコミットを強制するステートメントを実行し、新しいトランザクションを開始できます。

ストアドプロシージャがトランザクション内で開始された場合、そのトランザクションはストアドプロシージャイベントの親になります。

ストアドプロシージャによってトランザクションが開始された場合、ストアドプロシージャはトランザクションイベントの親です。

- ストアドファンクション

ストアドファンクションは、明示的または暗黙的なコミットまたはロールバックの原因に制限されています。ストアドファンクションイベントは、親トランザクションイベント内に存在できます。

- トリガー

トリガーは、関連付けられているテーブルにアクセスするステートメントの一部としてアクティブ化されるため、トリガーイベントの親は常に、トリガーイベントをアクティブ化するステートメントです。

トリガーは、トランザクションを明示的または暗黙的にコミットまたはロールバックするステートメントを発行できません。

- 予定イベント

スケジュール済みイベントの本文でのステートメントの実行は、新しい接続で行われます。親トランザクション内でのスケジュール済みイベントのネストは適用できません。

トランザクションとセーブポイント

セーブポイントステートメントは、個別のステートメントイベントとして記録されます。トランザクションイベントには、トランザクション中に発行された **SAVEPOINT**、**ROLLBACK TO SAVEPOINT** および **RELEASE SAVEPOINT** ステートメントの個別のカウンタが含まれます。

トランザクションおよびエラー

トランザクション内で発生したエラーおよび警告は、ステートメントイベントに記録されますが、対応するトランザクションイベントには記録されません。これには、非トランザクションテーブルでのロールバックや GTID 整合性エラーなど、トランザクション固有のエラーおよび警告が含まれます。

27.12.7.1 events_transactions_current テーブル

`events_transactions_current` テーブルには、現在のトランザクションイベントが含まれます。テーブルには、スレッドごとに最新の監視対象トランザクションイベントの現在のステータスを示す 1 行が格納されるため、テーブルサイズを構成するためのシステム変数はありません。例:

```
mysql> SELECT *
      FROM performance_schema.events_transactions_current LIMIT 1\G
***** 1. row *****
      THREAD_ID: 26
      EVENT_ID: 7
      END_EVENT_ID: NULL
      EVENT_NAME: transaction
      STATE: ACTIVE
      TRX_ID: NULL
      GTID: 3E11FA47-71CA-11E1-9E33-C80AA9429562:56
      XID: NULL
      XA_STATE: NULL
      SOURCE: transaction.cc:150
      TIMER_START: 420833537900000
      TIMER_END: NULL
      TIMER_WAIT: NULL
      ACCESS_MODE: READ WRITE
      ISOLATION_LEVEL: REPEATABLE READ
      AUTOCOMMIT: NO
      NUMBER_OF_SAVEPOINTS: 0
      NUMBER_OF_ROLLBACK_TO_SAVEPOINT: 0
      NUMBER_OF_RELEASE_SAVEPOINT: 0
      OBJECT_INSTANCE_BEGIN: NULL
      NESTING_EVENT_ID: 6
      NESTING_EVENT_TYPE: STATEMENT
```

トランザクションイベント行を含むテーブルの中で、最も基本的なのは `events_transactions_current` です。トランザクションイベント行を含む他のテーブルは、現在のイベントから論理的に導出されます。たとえば、`events_transactions_history` テーブルと `events_transactions_history_long` テーブルは、終了した最新のトランザクションイベントのコレクションで、スレッド当たりの最大行数まで、およびすべてのスレッドにわたってグローバルに終了します。

3 つのトランザクションイベントテーブル間の関係の詳細は、[セクション27.9「現在および過去のイベントのパフォーマンススキーマテーブル」](#) を参照してください。

トランザクションイベントを収集するかどうかの構成の詳細は、[セクション27.12.7「パフォーマンススキーマのトランザクションテーブル」](#) を参照してください。

`events_transactions_current` テーブルには、次のカラムがあります:

- `THREAD_ID`、`EVENT_ID`

イベントに関連付けられたスレッドとイベントの起動時のスレッドの現在のイベント番号。ともに取得される `THREAD_ID` と `EVENT_ID` の値によって、行が一意に識別されます。同じ値のペアを持つ行は 2 つありません。

- `END_EVENT_ID`

このカラムは、イベントの起動時に `NULL` に設定され、イベントの終了時にスレッドの現在のイベント番号に更新されます。

- `EVENT_NAME`

イベントが収集されたインストゥルメントの名前。これは `setup_instruments` テーブルからの `NAME` 値です。[セクション27.6「パフォーマンススキーマインストゥルメント命名規則」](#) に説明するように、インストゥルメント名には複数の部分があり、階層を形成することがあります。

- `STATE`

現在のトランザクションの状態。値は、`ACTIVE` (`START TRANSACTION` または `BEGIN` の後)、`COMMITTED` (`COMMIT` の後) または `ROLLED BACK` (`ROLLBACK` の後) です。

- `TRX_ID`

使用されません。

- **GTID**

GTID カラムには `gtid_next` の値が含まれ、`ANONYMOUS`、`AUTOMATIC` または GTID のいずれかを `UUID:NUMBER` の形式で指定できます。 `gtid_next=AUTOMATIC`(すべての通常のクライアントトランザクション) を使用するトランザクションの場合、GTID カラムは、トランザクションがコミットされ、実際の GTID が割り当てられると変更されます。 `gtid_mode` が `ON` または `ON_PERMISSIVE` の場合、GTID カラムはトランザクション GTID に変更されます。 `gtid_mode` が `OFF` または `OFF_PERMISSIVE` の場合、GTID カラムは `ANONYMOUS` に変更されます。

- **XID_FORMAT_ID**、**XID_GTRID** および **XID_BQUAL**

XA トランザクション識別子の要素。これらの形式は、[セクション13.3.8.1「XA トランザクション SQL ステートメント」](#) で説明されている形式です。

- **XA_STATE**

XA トランザクションの状態。値は、`ACTIVE` (XA START の後)、`IDLE` (XA END の後)、`PREPARED` (XA PREPARE の後)、`ROLLED BACK` (XA ROLLBACK の後) または `COMMITTED` (XA COMMIT の後) です。

レプリカでは、`events_transactions_current` テーブルに同じ XA トランザクションが表示され、スレッドごとに状態が異なる場合があります。これは、XA トランザクションが準備された直後にレプリカアプライヤスレッドからデタッチされ、レプリカ上の任意のスレッドによってコミットまたはロールバックできるためです。 `events_transactions_current` テーブルには、スレッド上の最新の監視対象トランザクションイベントの現在のステータスが表示され、スレッドがアイドル状態の場合、このステータスは更新されません。そのため、XA トランザクションは、別のスレッドによって処理された後も、元のアプライヤスレッドの `PREPARED` 状態で表示できます。 `PREPARED` 状態のままではリカバリが必要な XA トランザクションを肯定的に識別するには、パフォーマンススキーマトランザクションテーブルではなく、`XA RECOVER` ステートメントを使用します。

- **SOURCE**

イベントを生成した、インストールされたコードを格納するソースファイルの名前と、インストールセッションが行われたファイルの行番号。これにより、ソースをチェックして、コードに含まれるものを正確に判断することができます。

- **TIMER_START**、**TIMER_END**、**TIMER_WAIT**

イベントのタイミング情報。これらの値の単位はピコ秒 (秒の 1 兆分の 1) です。 `TIMER_START` および `TIMER_END` 値は、イベントのタイミングが開始されたときと終了したときを示します。 `TIMER_WAIT` はイベントの経過時間 (期間) です。

イベントが終了していない場合、`TIMER_END` は現在のタイマー値で、`TIMER_WAIT` はこれまでに経過した時間です (`TIMER_END - TIMER_START`)。

イベントが `TIMED = NO` のインストールから生成されている場合、タイミング情報は収集されず、`TIMER_START`、`TIMER_END`、および `TIMER_WAIT` はすべて `NULL` になります。

イベント時間の単位としてのピコ秒および時間値に影響する要因については、[セクション27.4.1「パフォーマンススキーマイベントタイミング」](#) を参照してください。

- **ACCESS_MODE**

トランザクションアクセスモード。値は `READ WRITE` または `READ ONLY` です。

- **ISOLATION_LEVEL**

トランザクション分離レベル。値は `REPEATABLE READ`、`READ COMMITTED`、`READ UNCOMMITTED` または `SERIALIZABLE` です。

- **AUTOCOMMIT**

トランザクションの開始時に `autocommit` モードが有効化されたかどうか。

- `NUMBER_OF_SAVEPOINTS`, `NUMBER_OF_ROLLBACK_TO_SAVEPOINT`,
`NUMBER_OF_RELEASE_SAVEPOINT`

トランザクション中に発行された `SAVEPOINT`、`ROLLBACK TO SAVEPOINT` および `RELEASE SAVEPOINT` ステートメントの数。

- `OBJECT_INSTANCE_BEGIN`

使用されません。

- `NESTING_EVENT_ID`

このイベントが中にネストされているイベントの `EVENT_ID` 値。

- `NESTING_EVENT_TYPE`

ネストしているイベントの種類。値は `TRANSACTION`、`STATEMENT`、`STAGE` または `WAIT` です。(トランザクションはネストできないため、`TRANSACTION` は表示されません。)

`events_transactions_current` テーブルには次のインデックスがあります:

- 主キー (`THREAD_ID`、`EVENT_ID`)

`TRUNCATE TABLE` は `events_transactions_current` テーブルに対して許可されています。行が削除されます。

27.12.7.2 events_transactions_history テーブル

`events_transactions_history` テーブルには、スレッドごとに終了した `N` の最新のトランザクションイベントが含まれます。トランザクションイベントは、終了するまでテーブルに追加されません。テーブルに特定のスレッドの最大行数が含まれている場合、そのスレッドの新しい行が追加されると、最も古いスレッド行は破棄されます。スレッドが終了すると、そのすべての行が破棄されます。

パフォーマンススキーマは、サーバーの起動時に `N` の値を自動サイズ調整します。スレッドごとの行数を明示的に設定するには、サーバーの起動時に `performance_schema_events_transactions_history_size` システム変数を設定します。

`events_transactions_history` テーブルには、`events_transactions_current` と同じカラムおよびインデックス付けがあります。 [セクション27.12.7.1「events_transactions_current テーブル」](#) を参照してください。

`TRUNCATE TABLE` は `events_transactions_history` テーブルに対して許可されています。行が削除されます。

3つのトランザクションイベントテーブル間の関係の詳細は、 [セクション27.9「現在および過去のイベントのパフォーマンススキーマテーブル」](#) を参照してください。

トランザクションイベントを収集するかどうかの構成の詳細は、 [セクション27.12.7「パフォーマンススキーマのトランザクションテーブル」](#) を参照してください。

27.12.7.3 events_transactions_history_long テーブル

`events_transactions_history_long` テーブルには、すべてのスレッドでグローバルに終了した `N` の最新のトランザクションイベントが含まれます。トランザクションイベントは、終了するまでテーブルに追加されません。テーブルがいっぱいになると、どちらのスレッドがどちらの行を生成したかに関係なく、新しい行が追加されたときにもっとも古い行が破棄されます。

パフォーマンススキーマは、`N` の値をサーバーの起動時に自動サイズ調整します。テーブルサイズを明示的に設定するには、サーバーの起動時に `performance_schema_events_transactions_history_long_size` システム変数を設定します。

`events_transactions_history_long` テーブルには、`events_transactions_current` と同じカラムがあります。 [セクション27.12.7.1「events_transactions_current テーブル」](#) を参照してください。 `events_transactions_current` とは異なり、`events_transactions_history_long` にはインデックス付けはありません。

TRUNCATE TABLE は `events_transactions_history_long` テーブルに対して許可されています。行が削除されます。

3つのトランザクションイベントテーブル間の関係の詳細は、[セクション27.9「現在および過去のイベントのパフォーマンススキーマテーブル」](#)を参照してください。

トランザクションイベントを収集するかどうかの構成の詳細は、[セクション27.12.7「パフォーマンススキーマのトランザクションテーブル」](#)を参照してください。

27.12.8 パフォーマンススキーマ接続テーブル

クライアントは、MySQL サーバーに接続すると、特定のユーザー名で特定のホストから接続します。パフォーマンススキーマは、次のテーブルを使用して、これらの接続に関する統計情報を提供し、アカウント (ユーザーとホストの組み合わせ) ごとに追跡したり、ユーザー名とホスト名ごとに個別に追跡したりします:

- `accounts`: クライアントアカウントごとの接続統計
- `hosts`: クライアントホスト名ごとの接続統計
- `users`: クライアントユーザー名ごとの接続統計

接続テーブルの「`account`」の意味は、`mysql` システムデータベースの MySQL 付与テーブルでの意味と似ていますが、この用語はユーザーとホストの値の組合せを意味します。権限付与テーブルの場合、アカウントのホスト部分はパターンにできますが、「パフォーマンススキーマ」テーブルの場合、ホスト値は常に特定のパターン以外のホスト名になります。

各接続テーブルには、統計の基になる「トラッキング値」当たりの現在の接続数および合計接続数を追跡するための `CURRENT_CONNECTIONS` カラムと `TOTAL_CONNECTIONS` カラムがあります。テーブルは、それらが追跡値に使用するものに違いがあります。`accounts` テーブルには、ユーザーとホストの組合せごとに接続を追跡するための `USER` カラムと `HOST` カラムがあります。`users` テーブルと `hosts` テーブルには、ユーザー名とホスト名ごとに接続を追跡するための `USER` カラムと `HOST` カラムがそれぞれあります。

パフォーマンススキーマは、`NULL` の `USER` および `HOST` カラム値を持つ行を使用して、認証に失敗したユーザーセッションの内部スレッドとスレッドもカウントします。

`user1` と `user2` というクライアントがそれぞれ `hosta` と `hostb` から一度に接続するとします。パフォーマンススキーマは次のように接続を追跡します。

- `accounts` テーブルには、`user1/hosta`、`user1/hostb`、`user2/hosta` および `user2/hostb` アカウント値について、各行でアカウントごとに1つの接続がカウントされる4つの行があります。
- `hosts` テーブルには2つの行があり、`hosta` および `hostb` の場合、各行はホスト名ごとに2つの接続をカウントします。
- `users` テーブルには2つの行があり、`user1` および `user2` の場合、各行はユーザー名ごとに2つの接続をカウントします。

クライアントが接続すると、パフォーマンススキーマは、各テーブルに適した追跡値を使用して、各接続テーブルのどの行が適用されるかを決定します。そのような行がない場合、追加されます。次に、パフォーマンススキーマはその行の `CURRENT_CONNECTIONS` および `TOTAL_CONNECTIONS` カラムを1つ増分します。

クライアントが切断すると、パフォーマンススキーマはその行の `CURRENT_CONNECTIONS` カラムを1つ減分し、`TOTAL_CONNECTIONS` カラムは変更しないままにします。

TRUNCATE TABLE は接続テーブルに対して許可されます。これには次の効果があります:

- 現在の接続を持たないアカウント、ホストまたはユーザー (`CURRENT_CONNECTIONS = 0` の行) の行は削除されます。
- 削除されていない行はリセットされ、現在の接続のみがカウントされます: `CURRENT_CONNECTIONS > 0` の行の場合、`TOTAL_CONNECTIONS` は `CURRENT_CONNECTIONS` にリセットされます。
- このセクションの後半で説明するように、接続テーブルに依存するサマリーテーブルは暗黙的に切り捨てられません。

パフォーマンススキーマは、アカウント、ホスト、またはユーザーごとにさまざまなイベントタイプの接続統計を集約するサマリーテーブルを保持します。これらのテーブルの名前には、`_summary_by_account`、`_summary_by_host` または `_summary_by_user` が含まれます。識別するには、次のクエリーを使用します:

```
mysql> SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'performance_schema'
AND TABLE_NAME REGEXP '_summary_by_(account|host|user)'
ORDER BY TABLE_NAME;
+-----+
| TABLE_NAME |
+-----+
| events_errors_summary_by_account_by_error |
| events_errors_summary_by_host_by_error |
| events_errors_summary_by_user_by_error |
| events_stages_summary_by_account_by_event_name |
| events_stages_summary_by_host_by_event_name |
| events_stages_summary_by_user_by_event_name |
| events_statements_summary_by_account_by_event_name |
| events_statements_summary_by_host_by_event_name |
| events_statements_summary_by_user_by_event_name |
| events_transactions_summary_by_account_by_event_name |
| events_transactions_summary_by_host_by_event_name |
| events_transactions_summary_by_user_by_event_name |
| events_waits_summary_by_account_by_event_name |
| events_waits_summary_by_host_by_event_name |
| events_waits_summary_by_user_by_event_name |
| memory_summary_by_account_by_event_name |
| memory_summary_by_host_by_event_name |
| memory_summary_by_user_by_event_name |
+-----+
```

個々の接続サマリーテーブルの詳細は、サマリーが作成されたイベントタイプのテーブルについて説明しているセクションを参照してください:

- 待機イベントのサマリー: [セクション27.12.18.1「待機イベント要約テーブル」](#)
- ステージイベントサマリー: [セクション27.12.18.2「ステージサマリーテーブル」](#)
- ステートメントイベントサマリー: [セクション27.12.18.3「ステートメントサマリーテーブル」](#)
- トランザクションイベントサマリー: [セクション27.12.18.5「トランザクション要約テーブル」](#)
- メモリーイベントサマリー: [セクション27.12.18.10「メモリーサマリーテーブル」](#)
- エラーイベントサマリー: [セクション27.12.18.11「エラー要約テーブル」](#)

`TRUNCATE TABLE` は接続サマリーテーブルに使用できます。接続のないアカウント、ホストまたはユーザーの行を削除し、残りの行のサマリーカラムをゼロにリセットします。また、アカウント、ホスト、ユーザーまたはスレッド別に集計された各サマリーテーブルは、依存する接続テーブルの切捨てによって暗黙的に切り捨てられます。次のテーブルに、接続テーブルの切捨てと暗黙的に切り捨てられたテーブルの関係を示します。

表 27.2 接続テーブルの切捨ての暗黙的な影響

切り捨てられた接続テーブル	暗黙的に切り捨てられたサマリーテーブル
<code>accounts</code>	<code>_summary_by_account</code> 、 <code>_summary_by_thread</code> を含む名前を持つテーブル
<code>hosts</code>	<code>_summary_by_account</code> 、 <code>_summary_by_host</code> 、 <code>_summary_by_thread</code> を含む名前を持つテーブル
<code>users</code>	<code>_summary_by_account</code> 、 <code>_summary_by_user</code> 、 <code>_summary_by_thread</code> を含む名前を持つテーブル

`_summary_global` サマリーテーブルを切り捨てると、対応する接続およびスレッドサマリーテーブルも暗黙的に切り捨てられます。たとえば、`events_waits_summary_global_by_event_name` を切り捨てると、アカウント、ホスト、ユーザーまたはスレッドごとに集計された待機イベントサマリーテーブルが暗黙的に切り捨てられます。

27.12.8.1 accounts テーブル

`accounts` テーブルは、MySQL サーバーに接続した各アカウントの行を格納します。各アカウントについて、テーブルは現在と合計の接続数をカウントします。テーブルサイズはサーバー起動時に自動サイズ設定されます。テーブルサイズを明示的に設定するには、サーバー起動時に `performance_schema_accounts_size` システム変数を設定します。アカウント統計を無効にするには、この変数を 0 に設定します。

`accounts` テーブルには次のカラムがあります。 `TRUNCATE TABLE` の効果を含む、パフォーマンススキーマがこのテーブルの行を保守する方法については、[セクション27.12.8「パフォーマンススキーマ接続テーブル」](#)を参照してください。

- `USER`

接続のクライアントユーザー名。これは、内部スレッドまたは認証に失敗したユーザーセッションの `NULL` です。

- `HOST`

クライアントの接続元のホスト。これは、内部スレッドまたは認証に失敗したユーザーセッションの `NULL` です。

- `CURRENT_CONNECTIONS`

アカウントの現在の接続数。

- `TOTAL_CONNECTIONS`

アカウントの合計の接続数。

`accounts` テーブルには次のインデックスがあります:

- 主キー (`USER`、`HOST`)

27.12.8.2 hosts テーブル

`hosts` テーブルは、クライアントが MySQL サーバーに接続している各ホストの行を格納します。各ホスト名について、テーブルは現在と合計の接続数をカウントします。テーブルサイズはサーバー起動時に自動サイズ設定されます。テーブルサイズを明示的に設定するには、サーバー起動時に `performance_schema_hosts_size` システム変数を設定します。ホスト統計を無効にするには、この変数を 0 に設定します。

`hosts` テーブルには次のカラムがあります。 `TRUNCATE TABLE` の効果を含む、パフォーマンススキーマがこのテーブルの行を保守する方法については、[セクション27.12.8「パフォーマンススキーマ接続テーブル」](#)を参照してください。

- `HOST`

クライアントの接続元のホスト。これは、内部スレッドまたは認証に失敗したユーザーセッションの `NULL` です。

- `CURRENT_CONNECTIONS`

ホストの現在の接続数。

- `TOTAL_CONNECTIONS`

ホストの合計の接続数。

`hosts` テーブルには次のインデックスがあります:

- 主キー (`HOST`)

27.12.8.3 users テーブル

`users` テーブルは、MySQL サーバーに接続している各ユーザーの行を格納します。各ユーザー名について、テーブルは現在と合計の接続数をカウントします。テーブルサイズはサーバー起動時に自動サイズ設定されます。テーブルサイズを明示的に設定するには、サーバー起動時に `performance_schema_users_size` システム変数を設定します。ユーザー統計を無効にするには、この変数を 0 に設定します。

`users` テーブルには次のカラムがあります。 `TRUNCATE TABLE` の効果を含む、パフォーマンススキーマがこのテーブルの行を保守する方法については、 [セクション27.12.8「パフォーマンススキーマ接続テーブル」](#) を参照してください。

- `USER`

接続のクライアントユーザー名。これは、内部スレッドまたは認証に失敗したユーザーセッションの `NULL` です。

- `CURRENT_CONNECTIONS`

ユーザーの現在の接続数。

- `TOTAL_CONNECTIONS`

ユーザーの合計の接続数。

`users` テーブルには次のインデックスがあります:

- 主キー (`USER`)

27.12.9 パフォーマンススキーマ接続属性テーブル

接続属性は、アプリケーションプログラムが接続時にサーバーに渡すことができるキーと値のペアです。 `libmysqlclient` クライアントライブラリによって実装される C API に基づくアプリケーションの場合、 `mysql_options()` および `mysql_options4()` 関数によって接続属性セットが定義されます。他の MySQL コネクタは、独自の属性定義メソッドを提供する場合があります。

「これらのパフォーマンススキーマ」テーブルでは、属性情報が公開されます:

- `session_account_connect_attrs`: 現在のセッションおよびセッションアカウントに関連付けられている他のセッションの接続属性
- `session_connect_attrs`: すべてのセッションの接続属性。

また、監査ログに書き込まれる接続イベントには、接続属性が含まれる場合があります。 [セクション6.4.5.4「監査ログファイル形式」](#) を参照してください。

アンダースコア (`_`) で始まる属性名は内部使用のために予約されているため、アプリケーションプログラムで作成しないでください。この規則により、アプリケーション属性と競合せずに MySQL で新しい属性を導入でき、内部属性と競合しない独自の属性をアプリケーションプログラムで定義できます。

- [使用可能な接続属性](#)
- [接続属性制限](#)

使用可能な接続属性

特定の接続内に表示される接続属性のセットは、プラットフォーム、接続の確立に使用される MySQL Connector、クライアントプログラムなどの要因によって異なります。

`libmysqlclient` クライアントライブラリは、次の属性を設定します:

- `_client_name`: クライアント名 (クライアントライブラリの `libmysql`)。
- `_client_version`: クライアントライブラリのバージョン。
- `_os`: オペレーティングシステム (`Linux`、`Win64` など)。
- `_pid`: クライアントプロセス ID。
- `_platform`: マシンプラットフォーム (例: `x86_64`)。
- `_thread`: クライアントスレッド ID (Windows のみ)。

他の MySQL コネクタでは、独自の接続属性を定義できます。

MySQL Connector/C++ 8.0.16 以上では、X DevAPI または X DevAPI for C を使用するアプリケーションに対して次の属性が定義されます:

- `_client_license`: コネクタライセンス (GPL-2.0 など)。
- `_client_name`: コネクタ名 (`mysql-connector-cpp`)。
- `_client_version`: コネクタバージョン。
- `_os`: オペレーティングシステム (`Linux`、`Win64` など)。
- `_pid`: クライアントプロセス ID。
- `_platform`: マシンプラットフォーム (例: `x86_64`)。
- `_source_host`: クライアントが実行されているマシンのホスト名。
- `_thread`: クライアントスレッド ID (Windows のみ)。

MySQL Connector/J では、次の属性が定義されます:

- `_client_name`: クライアント名
- `_client_version`: クライアントライブラリバージョン
- `_os`: オペレーティングシステム (`Linux`、`Win64` など)
- `_client_license`: コネクタライセンスタイプ
- `_platform`: マシンプラットフォーム (`x86_64` など)
- `_runtime_vendor`: Java ランタイム環境 (JRE) ベンダー
- `_runtime_version`: Java ランタイム環境 (JRE) のバージョン

MySQL Connector/NET では、次の属性が定義されます:

- `_client_version`: クライアントライブラリのバージョン。
- `_os`: オペレーティングシステム (`Linux`、`Win64` など)。
- `_pid`: クライアントプロセス ID。
- `_platform`: マシンプラットフォーム (例: `x86_64`)。
- `_program_name`: クライアント名。
- `_thread`: クライアントスレッド ID (Windows のみ)。

PHP は、コンパイル方法に依存する属性を定義します:

- `libmysqlclient` を使用してコンパイル: 前に説明した標準の `libmysqlclient` 属性。
- `mysqlnd` を使用してコンパイル: `mysqlnd` の値を持つ `_client_name` 属性のみ。

多くの MySQL クライアントプログラムは、クライアント名と等しい値を持つ `program_name` 属性を設定します。たとえば、`mysqladmin` および `mysqldump` は、`program_name` を `mysqladmin` および `mysqldump` にそれぞれ設定します。MySQL Shell は、`program_name` を `mysqlsh` に設定します。

一部の MySQL クライアントプログラムでは、追加の属性が定義されます:

- `mysql` (MySQL 8.0.17 の時点):
 - `os_user`: プログラムを実行しているオペレーティングシステムユーザーの名前。Unix および Unix に似たシステムおよび Windows で使用できます。

- `os_sudouser`: `SUDO_USER` 環境変数の値。Unix および Unix に似たシステムで使用できます。値が空の `mysql` 接続属性は送信されません。
- `mysqlbinlog`:
 - `_client_role`: `binary_log_listener`
- レプリカ接続:
 - `program_name`: `mysqld`
 - `_client_role`: `binary_log_listener`
 - `_client_replication_channel_name`: チャンネル名。
- `FEDERATED` ストレージエンジン接続:
 - `program_name`: `mysqld`
 - `_client_role`: `federated_storage`

接続属性制限

クライアントからサーバーに送信される接続属性データの量には制限があります:

- 接続時間の前にクライアントによって課される固定制限。
- 接続時にサーバーによって課される固定制限。
- 接続時にパフォーマンススキーマによって課される構成可能な制限。

C API を使用して開始された接続の場合、`libmysqlclient` ライブラリでは、クライアント側の接続属性データの集計サイズに 64KB の制限が課されます: この制限を超える `mysql_options()` をコールすると、`CR_INVALID_PARAMETER_NO` エラーが発生します。他の MySQL コネクタでは、サーバーに送信できる接続属性データの量に独自のクライアント側制限が課される場合があります。

サーバー側では、接続属性データに対する次のサイズチェックが行われます:

- サーバーは、受け入れる接続属性データの集計サイズに 64KB の制限を課します。クライアントが 64KB を超える属性データを送信しようとする、サーバーは接続を拒否します。それ以外の場合、サーバーは属性バッファが有効であるとみなし、`Performance_schema_session_connect_attrs_longest_seen` ステータス変数で最も長いバッファのサイズを追跡します。
- 受け入れられる接続の場合、パフォーマンススキーマは `performance_schema_session_connect_attrs_size` システム変数の値に対して集約属性サイズをチェックします。属性サイズがこの値を超えると、次のアクションが実行されます:
 - パフォーマンススキーマは属性データを切り捨て、`Performance_schema_session_connect_attrs_lost` ステータス変数を増分します。これは、属性の切り捨てが発生した接続の数を示します。
 - `log_error_verbosity` システム変数が 1 より大きい場合、パフォーマンススキーマはエラーメッセージをエラーログに書き込みます:

```
Connection attributes of length N were truncated
(N bytes lost)
for connection N, user user_name@host_name
(as user_name), auth: {yes|no}
```

警告メッセージの情報は、DBA が属性の切り捨てが発生したクライアントを識別するのに役立ちます。

- 属性バッファに十分な領域がある場合は、失われたバイト数を示す値とともに `_truncated` 属性がセッション属性に追加されます。これにより、パフォーマンススキーマは接続属性テーブルで接続ごとの切り捨て情報を公開できます。この情報は、エラーログを確認せずに調べることができます。

27.12.9.1 session_account_connect_attrs テーブル

アプリケーションプログラムは、接続時にサーバーに渡されるキーと値の接続属性を提供できます。共通属性の詳細は、[セクション27.12.9「パフォーマンススキーマ接続属性テーブル」](#)を参照してください。

[session_account_connect_attrs](#) テーブルには、現在のセッションおよびセッションアカウントに関連付けられている他のセッションの接続属性のみが含まれます。すべてのセッションの接続属性を表示するには、[session_connect_attrs](#) テーブルを使用します。

[session_account_connect_attrs](#) テーブルには、次のカラムがあります：

- [PROCESSLIST_ID](#)

セッションの接続識別子。

- [ATTR_NAME](#)

属性名。

- [ATTR_VALUE](#)

属性値。

- [ORDINAL_POSITION](#)

属性が一連の接続属性に追加された順序。

[session_account_connect_attrs](#) テーブルには次のインデックスがあります：

- 主キー ([PROCESSLIST_ID](#)、[ATTR_NAME](#))

[TRUNCATE TABLE](#) は、[session_account_connect_attrs](#) テーブルに対して許可されていません。

27.12.9.2 session_connect_attrs テーブル

アプリケーションプログラムは、接続時にサーバーに渡されるキーと値の接続属性を提供できます。共通属性の詳細は、[セクション27.12.9「パフォーマンススキーマ接続属性テーブル」](#)を参照してください。

[session_connect_attrs](#) テーブルはすべてのセッションの接続属性を格納します。現在のセッションおよびセッションアカウントに関連付けられている他のセッションの接続属性のみを表示するには、[session_account_connect_attrs](#) テーブルを使用します。

[session_connect_attrs](#) テーブルには、次のカラムがあります：

- [PROCESSLIST_ID](#)

セッションの接続識別子。

- [ATTR_NAME](#)

属性名。

- [ATTR_VALUE](#)

属性値。

- [ORDINAL_POSITION](#)

属性が一連の接続属性に追加された順序。

[session_connect_attrs](#) テーブルには次のインデックスがあります：

- 主キー ([PROCESSLIST_ID](#)、[ATTR_NAME](#))

TRUNCATE TABLE は、`session_connect_attrs` テーブルに対して許可されていません。

27.12.10 パフォーマンススキーマのユーザー定義変数テーブル

パフォーマンススキーマは、ユーザー定義変数を公開する `user_variables_by_thread` テーブルを提供します。これらは特定のセッション内で定義される変数で、名前の前に@文字が含まれます。セクション9.4「ユーザー定義変数」を参照してください。

`user_variables_by_thread` テーブルには、次のカラムがあります：

- `THREAD_ID`
変数が定義されているセッションのスレッド識別子。
- `VARIABLE_NAME`
@の先頭文字を含まない変数名。
- `VARIABLE_VALUE`
変数値。

`user_variables_by_thread` テーブルには次のインデックスがあります：

- 主キー (`THREAD_ID`、`VARIABLE_NAME`)

TRUNCATE TABLE は、`user_variables_by_thread` テーブルに対して許可されていません。

27.12.11 パフォーマンススキーマレプリケーションテーブル

パフォーマンススキーマは、レプリケーション情報を公開するテーブルを提供します。これは、`SHOW REPLICASlave STATUS` ステートメントから入手できる情報に似ていますが、テーブル形式での表現はアクセスしやすく、ユーザビリティ上の利点があります：

- `SHOW REPLICASlave STATUS` 出力は、視覚的な検査に役立ちますが、プログラムでの使用にはあまり役立ちません。対照的に、「パフォーマンススキーマ」テーブルを使用すると、複雑な `WHERE` 条件、結合など、一般的な `SELECT` クエリーを使用してレプリカステータスに関する情報を検索できます。
- クエリー結果は、さらに分析するためにテーブルに保存することも、変数に割り当ててストアードプロシージャで使用することもできます。
- レプリケーションテーブルは、より適切な診断情報を提供します。マルチスレッドレプリカ操作の場合、`SHOW REPLICASlave STATUS` は `Last_SQL_Errno` および `Last_SQL_Error` フィールドを使用してすべてのコネクターおよびワーカースレッドエラーをレポートするため、最新のエラーのみが表示され、情報が失われる可能性があります。レプリケーションテーブルには、情報を失わずにスレッドごとにエラーが格納されます。
- 最後に表示されたトランザクションは、ワーカーごとにレプリケーションテーブルに表示されます。これは、`SHOW REPLICASlave STATUS` からは入手できない情報です。
- パフォーマンススキーマインタフェースに精通している開発者は、テーブルに行を追加することによって、レプリケーションテーブルを拡張して追加情報を提供できます。

レプリケーションテーブルの説明

パフォーマンススキーマは、次のレプリケーション関連テーブルを提供します：

- ソースへのレプリカの接続に関する情報を含むテーブル：
 - `replication_connection_configuration`: ソースに接続するための構成パラメータ
 - `replication_connection_status`: ソースへの接続の現在のステータス
 - `replication_asynchronous_connection_failover`: 非同期接続フェイルオーバーメカニズムのソースリスト

- トランザクションアプライアンスに関する一般的な (スレッド固有ではない) 情報を含むテーブル:
 - `replication_applier_configuration`: レプリカ上のトランザクションアプライアンスの構成パラメータ。
 - `replication_applier_status`: レプリカ上のトランザクションアプライアンスの現在のステータス。
- ソースから受信したトランザクションの適用を担当する特定のスレッドに関する情報を含むテーブル:
 - `replication_applier_status_by_coordinator`: コーディネータスレッドのステータス (レプリカがマルチスレッドでないかぎり空)。
 - `replication_applier_status_by_worker`: レプリカがマルチスレッドの場合のアプライヤスレッドまたはワーカー スレッドのステータス。
- チャンネルベースのレプリケーションフィルタに関する情報を含むテーブル:
 - `replication_applier_filters`: 特定のレプリケーションチャンネルに構成されているレプリケーションフィルタに関する情報を提供します。
 - `replication_applier_global_filters`: すべてのレプリケーションチャンネルに適用されるグローバルレプリケーションフィルタに関する情報を提供します。
- グループレプリケーションメンバーに関する情報を含むテーブル:
 - `replication_group_members`: グループメンバーのネットワークおよびステータス情報を提供します。
 - `replication_group_member_stats`: 参加しているグループメンバーおよびトランザクションに関する統計情報を提供します。

詳細は、[セクション18.3「グループレプリケーションの監視」](#) を参照してください。

パフォーマンススキーマが無効になっている場合、次のパフォーマンススキーマレプリケーションテーブルは引き続き移入されます:

- `replication_connection_configuration`
- `replication_connection_status`
- `replication_asynchronous_connection_failover`
- `replication_applier_configuration`
- `replication_applier_status`
- `replication_applier_status_by_coordinator`
- `replication_applier_status_by_worker`

例外は、レプリケーションテーブル `replication_connection_status`、`replication_applier_status_by_coordinator` および `replication_applier_status_by_worker` のローカルタイミング情報 (トランザクションの開始タイムスタンプと終了タイムスタンプ) です。パフォーマンススキーマが無効になっている場合、この情報は収集されません。

次の各セクションでは、[SHOW REPLICA | SLAVE STATUS](#) によって生成されるカラムと、同じ情報が表示されるレプリケーションテーブルのカラムとの対応など、各レプリケーションテーブルについて詳しく説明します。

このレプリケーションテーブルの残りの部分では、パフォーマンススキーマがそれらをどのように移入するか、および [SHOW REPLICA | SLAVE STATUS](#) のどのフィールドがテーブルに表示されないかについて説明します。

レプリケーションテーブルのライフサイクル

パフォーマンススキーマは、次のようにレプリケーションテーブルに移入します:

- [CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO](#) を実行する前は、テーブルは空です。

- [CHANGE REPLICATION SOURCE TO](#) | [CHANGE MASTER TO](#) の後、構成パラメータはテーブルに表示されます。現時点では、アクティブなレプリケーションスレッドはないため、[THREAD_ID](#) カラムは `NULL` で、[SERVICE_STATE](#) カラムの値は `OFF` です。
- [START REPLICA](#) | [SLAVE](#) の後、`NULL` 以外の [THREAD_ID](#) 値が表示されます。アイドル状態またはアクティブなスレッドの [SERVICE_STATE](#) 値は `ON` です。ソースに接続するスレッドは、接続の確立中は `CONNECTING` の値を持ち、接続が継続しているかぎり `ON` の値を持ちます。
- [STOP REPLICA](#) | [SLAVE](#) の後、[THREAD_ID](#) カラムは `NULL` になり、存在しなくなったスレッドの [SERVICE_STATE](#) カラムの値は `OFF` になります。
- テーブルは、[STOP REPLICA](#) | [SLAVE](#) またはスレッドがエラーのために停止した後も保持されます。
- [replication_applier_status_by_worker](#) テーブルは、レプリカがマルチスレッドモードで動作している場合にのみ空ではありません。つまり、[slave_parallel_workers](#) システム変数が 0 より大きい場合、このテーブルは [START REPLICA](#) | [SLAVE](#) の実行時に移入され、行数にワーカー数が表示されます。

レプリケーションテーブルにないレプリカステータス情報

パフォーマンススキーマレプリケーションテーブル内の情報は、[SHOW REPLICA](#) | [SLAVE STATUS](#) から使用可能な情報とは多少異なります。テーブルは、ファイル名や位置ではなくグローバルトランザクション識別子 (GTID) の使用に向けており、サーバー ID 値ではなくサーバー UUID 値を表しているためです。これらの違いのため、いくつかの [SHOW REPLICA](#) | [SLAVE STATUS](#) カラムはパフォーマンススキーマレプリケーションテーブルに保持されないか、別の方法で表されます:

- 次のフィールドはファイル名と位置を参照し、保持されません:

```
Master_Log_File  
Read_Master_Log_Pos  
Relay_Log_File  
Relay_Log_Pos  
Relay_Master_Log_File  
Exec_Master_Log_Pos  
Until_Condition  
Until_Log_File  
Until_Log_Pos
```

- [Master_Info_File](#) フィールドは保持されません。これは、レプリカソースメタデータリポジトリに使用される `master.info` ファイルを参照します。これは、リポジトリにクラッシュセーフテーブルを使用することで置き換えられています。
- 次のフィールドは、[server_uuid](#) ではなく [server_id](#) に基づいており、保持されません:

```
Master_Server_Id  
Replicate_Ignore_Server_Ids
```

- [Skip_Counter](#) フィールドは GTID ではなくイベント数に基づいており、保持されません。
- これらのエラーフィールドは [Last_SQL_Errno](#) および [Last_SQL_Error](#) のエイリアスであるため、保持されません:

```
Last_Errno  
Last_Error
```

パフォーマンススキーマでは、このエラー情報は [replication_applier_status_by_worker](#) テーブル (およびレプリカがマルチスレッドの場合は [replication_applier_status_by_coordinator](#)) の [LAST_ERROR_NUMBER](#) および [LAST_ERROR_MESSAGE](#) カラムで使用できます。これらのテーブルは、[Last_Errno](#) および [Last_Error](#) から入手できるよりも具体的なスレッドエラーごとの情報を提供します。

- コマンド行フィルタリングオプションに関する情報を提供するフィールドは保持されません:

```
Replicate_Do_DB  
Replicate_Ignore_DB  
Replicate_Do_Table  
Replicate_Ignore_Table  
Replicate_Wild_Do_Table  
Replicate_Wild_Ignore_Table
```

- [Replica_IO_State](#) および [Replica_SQL_Running_State](#) のフィールドは保持されません。必要に応じて、適切なレプリケーションテーブルの [THREAD_ID](#) カラムを使用し、[INFORMATION_SCHEMA PROCESLIST](#) テーブルの [ID](#) カラムと結合して、プロセスリストからこれらの値を取得し、後者のテーブルの [STATE](#) カラムを選択できます。
- [Executed_Gtid_Set](#) フィールドには、大量のテキストを含む大きなセットを表示できます。代わりに、「パフォーマンススキーマ」テーブルには、レプリカによって現在適用されているトランザクションの GTID が表示されます。または、実行された GTID のセットを [gtid_executed](#) システム変数の値から取得できます。
- [Seconds_Behind_Master](#) および [Relay_Log_Space](#) フィールドは指定予定のステータスであり、保持されません。

レプリケーションチャンネル

レプリケーション「パフォーマンススキーマ」テーブルの最初のカラムは [CHANNEL_NAME](#) です。これにより、レプリケーションチャンネルごとにテーブルを表示できます。非マルチソースレプリケーション設定には、単一のデフォルトレプリケーションチャンネルがあります。レプリカで複数のレプリケーションチャンネルを使用している場合は、レプリケーションチャンネルごとにテーブルをフィルタして、特定のレプリケーションチャンネルを監視できます。詳細は、[セクション17.2.2「レプリケーションチャンネル」](#) および [セクション17.1.5.8「マルチソースレプリケーションの監視」](#) を参照してください。

27.12.11.1 replication_connection_configuration テーブル

このテーブルは、レプリカがソースに接続するために使用する構成パラメータを示しています。テーブルに格納されているパラメータは、実行時に [CHANGE REPLICATION SOURCE TO](#) ステートメント (MySQL 8.0.23) または [CHANGE MASTER TO](#) ステートメント (MySQL 8.0.23 より前) を使用して変更できます。

[replication_connection_status](#) テーブルと比較すると、[replication_connection_configuration](#) の変更頻度は低くなります。これには、レプリカがソースに接続する方法を定義し、接続中に一定のままになる値が含まれますが、[replication_connection_status](#) には、接続中に変更される値が含まれます。

[replication_connection_configuration](#) テーブルには次のカラムがあります。カラムの説明には、カラム値の取得元となる対応する [CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO](#) オプションが示され、このセクションで後述するテーブルには、[replication_connection_configuration](#) カラムと [SHOW REPLICA | SLAVE STATUS](#) カラムの対応関係が示されています。

- [CHANNEL_NAME](#)

この行が表示しているレプリケーションチャンネル。常にデフォルトのレプリケーションチャンネルがあり、さらにレプリケーションチャンネルを追加できます。詳しくは[セクション17.2.2「レプリケーションチャンネル」](#)をご覧ください。(CHANGE REPLICATION SOURCE TO オプション: FOR CHANNEL、CHANGE MASTER TO オプション: FOR CHANNEL)

- [HOST](#)

レプリカの接続先のソースのホスト名。(CHANGE REPLICATION SOURCE TO オプション: SOURCE_HOST、CHANGE MASTER TO オプション: MASTER_HOST)

- [PORT](#)

ソースへの接続に使用されるポート。(CHANGE REPLICATION SOURCE TO オプション: SOURCE_PORT、CHANGE MASTER TO オプション: MASTER_PORT)

- [USER](#)

ソースへの接続に使用されるレプリケーションユーザーアカウントのユーザー名。(CHANGE REPLICATION SOURCE TO オプション: SOURCE_USER、CHANGE MASTER TO オプション: MASTER_USER)

- [NETWORK_INTERFACE](#)

レプリカがバインドされているネットワークインタフェース(ある場合)。(CHANGE REPLICATION SOURCE TO オプション: SOURCE_BIND、CHANGE MASTER TO オプション: MASTER_BIND)

- [AUTO_POSITION](#)

GTID 自動配置が使用されている場合は 1、それ以外の場合は 0。(CHANGE REPLICATION SOURCE TO オプション: SOURCE_AUTO_POSITION、CHANGE MASTER TO オプション: MASTER_AUTO_POSITION)

- SSL_ALLOWED, SSL_CA_FILE, SSL_CA_PATH, SSL_CERTIFICATE, SSL_CIPHER, SSL_KEY, SSL_VERIFY_SERVER_CERTIFICATE, SSL_CRL_FILE, SSL_CRL_PATH

これらのカラムには、レプリカがソースに接続するために使用する SSL パラメータが表示されます (存在する場合)。

SSL_ALLOWED には次の値があります:

- Yes(ソースへの SSL 接続が許可されている場合)
- ソースへの SSL 接続が許可されていない場合は No
- SSL 接続が許可されているが、レプリカで SSL サポートが有効になっていない場合は Ignored

(他の SSL カラムの CHANGE REPLICATION SOURCE TO オプション: SOURCE_SSL_CA, SOURCE_SSL_CAPATH, SOURCE_SSL_CERT, SOURCE_SSL_CIPHER, SOURCE_SSL_CRL, SOURCE_SSL_CRLPATH, SOURCE_SSL_KEY, SOURCE_SSL_VERIFY_SERVER_CERT。

その他の SSL カラムの CHANGE MASTER TO オプション: MASTER_SSL_CA, MASTER_SSL_CAPATH, MASTER_SSL_CERT, MASTER_SSL_CIPHER, MASTER_SSL_CRL, MASTER_SSL_CRLPATH, MASTER_SSL_KEY, MASTER_SSL_VERIFY_SERVER_CERT。

- CONNECTION_RETRY_INTERVAL

接続再試行間の秒数。(CHANGE REPLICATION SOURCE TO オプション: SOURCE_CONNECT_RETRY、CHANGE MASTER TO オプション: MASTER_CONNECT_RETRY)

- CONNECTION_RETRY_COUNT

接続が失われた場合にレプリカがソースへの再接続を試行できる回数。(CHANGE REPLICATION SOURCE TO オプション: SOURCE_RETRY_COUNT、CHANGE MASTER TO オプション: MASTER_RETRY_COUNT)

- HEARTBEAT_INTERVAL

レプリカのレプリケーションハートビート間隔 (秒単位)。(CHANGE REPLICATION SOURCE TO オプション: SOURCE_HEARTBEAT_PERIOD、CHANGE MASTER TO オプション: MASTER_HEARTBEAT_PERIOD)

- TLS_VERSION

レプリケーション接続のレプリカによって許可される TLS プロトコルバージョンのリスト。TLS バージョン情報については、[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#)を参照してください。(CHANGE REPLICATION SOURCE TO オプション: SOURCE_TLS_VERSION、CHANGE MASTER TO オプション: MASTER_TLS_VERSION)

- TLS_CIPHERSUITES

レプリケーション接続のレプリカによって許可される暗号スイートのリスト。TLS 暗号スイートの詳細は、[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#)を参照してください。(CHANGE REPLICATION SOURCE TO オプション: SOURCE_TLS_CIPHERSUITES、CHANGE MASTER TO オプション: MASTER_TLS_CIPHERSUITES)

- PUBLIC_KEY_PATH

RSA キーペアベースのパスワード交換のソースに必要な公開キーのレプリカ側コピーを含むファイルへのパス名。ファイルは PEM 形式である必要があります。このカラムは、sha256_password または caching_sha2_password 認証プラグインで認証されるレプリカに適用されます。(CHANGE REPLICATION SOURCE TO オプション: SOURCE_PUBLIC_KEY_PATH、CHANGE MASTER TO オプション: MASTER_PUBLIC_KEY_PATH)

PUBLIC_KEY_PATH が指定され、有効な公開キーファイルが指定されている場合は、GET_PUBLIC_KEY よりも優先されます。

- [GET_PUBLIC_KEY](#)

RSA キーペアベースのパスワード交換に必要な公開キーをソースからリクエストするかどうか。このカラムは、[caching_sha2_password](#) 認証プラグインで認証されるレプリカに適用されます。そのプラグインの場合、ソースは要求されないかぎり公開鍵を送信しません。(CHANGE REPLICATION SOURCE TO オプション: [SOURCE_GET_PUBLIC_KEY](#)、CHANGE MASTER TO オプション: [MASTER_GET_PUBLIC_KEY](#))

[PUBLIC_KEY_PATH](#) が指定され、有効な公開キーファイルが指定されている場合は、[GET_PUBLIC_KEY](#) よりも優先されます。

- [NETWORK_NAMESPACE](#)

ネットワーク名前空間名。接続でデフォルト (グローバル) 名前空間を使用する場合は空です。ネットワーク名前空間の詳細は、[セクション5.1.14「ネットワーク名前空間のサポート」](#) を参照してください。このカラムは、MySQL 8.0.22 で追加されました。

- [COMPRESSION_ALGORITHMS](#)

ソースへの接続に許可される圧縮アルゴリズム。(CHANGE REPLICATION SOURCE TO オプション: [SOURCE_COMPRESSION_ALGORITHMS](#)、CHANGE MASTER TO オプション: [MASTER_COMPRESSION_ALGORITHMS](#))

詳細は、[セクション4.2.8「接続圧縮制御」](#) を参照してください。

このカラムは、MySQL 8.0.18 で追加されました。

- [ZSTD_COMPRESSION_LEVEL](#)

[zstd](#) 圧縮アルゴリズムを使用するソースへの接続に使用する圧縮レベル。(CHANGE REPLICATION SOURCE TO オプション: [SOURCE_ZSTD_COMPRESSION_LEVEL](#)、CHANGE MASTER TO オプション: [MASTER_ZSTD_COMPRESSION_LEVEL](#))

詳細は、[セクション4.2.8「接続圧縮制御」](#) を参照してください。

このカラムは、MySQL 8.0.18 で追加されました。

- [SOURCE_CONNECTION_AUTO_FAILOVER](#)

このレプリケーションチャネルに対して非同期接続フェイルオーバーメカニズムがアクティブ化されているかどうか。(CHANGE REPLICATION SOURCE TO オプション: [SOURCE_CONNECTION_AUTO_FAILOVER](#)、CHANGE MASTER TO オプション: [SOURCE_CONNECTION_AUTO_FAILOVER](#))

このカラムは、MySQL 8.0.22 で追加されました。

[replication_connection_configuration](#) テーブルには次のインデックスがあります:

- 主キー ([CHANNEL_NAME](#))

TRUNCATE TABLE は、[replication_connection_configuration](#) テーブルに対して許可されていません。

次のテーブルに、[replication_connection_configuration](#) カラムと [SHOW REPLICA | SLAVE STATUS](#) カラムの対応を示します。

replication_connection_configuration カラム	SHOW REPLICA SLAVE STATUS カラム
CHANNEL_NAME	Channel_name
HOST	Source_Host
PORT	Source_Port
USER	Source_User
NETWORK_INTERFACE	Source_Bind
AUTO_POSITION	Auto_Position

replication_connection_configuration カラム	SHOW REPLICA SLAVE STATUS カラム
SSL_ALLOWED	Source_SSL_Allowed
SSL_CA_FILE	Source_SSL_CA_File
SSL_CA_PATH	Source_SSL_CA_Path
SSL_CERTIFICATE	Source_SSL_Cert
SSL_CIPHER	Source_SSL_Cipher
SSL_KEY	Source_SSL_Key
SSL_VERIFY_SERVER_CERTIFICATE	Source_SSL_Verify_Server_Cert
SSL_CRL_FILE	Source_SSL_Crl
SSL_CRL_PATH	Source_SSL_Crlpath
CONNECTION_RETRY_INTERVAL	Source_Connect_Retry
CONNECTION_RETRY_COUNT	Source_Retry_Count
HEARTBEAT_INTERVAL	なし
TLS_VERSION	Source_TLS_Version
PUBLIC_KEY_PATH	Source_public_key_path
GET_PUBLIC_KEY	Get_source_public_key
NETWORK_NAMESPACE	Network_Namespace
COMPRESSION_ALGORITHMS	[None]
ZSTD_COMPRESSION_LEVEL	[None]

27.12.11.2 replication_connection_status テーブル

このテーブルには、ソースへのレプリカ接続を処理する I/O スレッドの現在のステータス、リレーログにキューされている最後のトランザクションに関する情報、および現在リレーログにキューされているトランザクションに関する情報が表示されます。

replication_connection_configuration テーブルと比較すると、replication_connection_status はより頻繁に変更されます。これには、接続中に変更される値が含まれますが、replication_connection_configuration には、レプリカがソースに接続する方法を定義し、接続中に一定のままになる値が含まれます。

replication_connection_status テーブルには、次のカラムがあります:

- CHANNEL_NAME

この行が表示しているレプリケーションチャンネル。常にデフォルトのレプリケーションチャンネルがあり、さらにレプリケーションチャンネルを追加できます。詳しくは[セクション17.2.2「レプリケーションチャンネル」](#)をご覧ください。

- GROUP_NAME

このサーバーがグループのメンバーである場合、サーバーが属するグループの名前が表示されます。

- SOURCE_UUID

ソースからの server_uuid 値。

- THREAD_ID

I/O スレッド ID。

- SERVICE_STATE

ON (スレッドは存在し、アクティブまたはアイドル状態)、OFF (スレッドはすでに存在しません) または CONNECTING (スレッドは存在し、ソースに接続しています)。

- [RECEIVED_TRANSACTION_SET](#)

このレプリカによって受信されたすべてのトランザクションに対応するグローバルトランザクション ID (GTID) のセット。GTID が使用されていない場合は空です。詳しくは[GTID セット](#)をご覧ください。

- [LAST_ERROR_NUMBER](#), [LAST_ERROR_MESSAGE](#)

I/O スレッドを停止させた最新のエラーのエラー番号とエラーメッセージ。0 のエラー番号および空の文字列のメッセージは、「エラーなし」を示します。[LAST_ERROR_MESSAGE](#) 値が空でない場合、エラー値はレプリカエラーログにも表示されます。

[RESET MASTER](#) または [RESET REPLICA | SLAVE](#) を発行すると、これらのカラムに表示される値がリセットされます。

- [LAST_ERROR_TIMESTAMP](#)

最新の I/O エラーがいつ発生したかを示す 'YYYY-MM-DD hh:mm:ss[.fraction]' 形式のタイムスタンプ。

- [LAST_HEARTBEAT_TIMESTAMP](#)

レプリカが最新のハートビートシグナルをいつ受信したかを示す 'YYYY-MM-DD hh:mm:ss[.fraction]' 形式のタイムスタンプ。

- [COUNT_RECEIVED_HEARTBEATS](#)

レプリカが最後に再起動またはリセットされてから、あるいは [CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO](#) ステートメントが発行されてから受信したハートビートシグナルの合計数。

- [LAST_QUEUED_TRANSACTION](#)

リレーログにキューに入れられた最後のトランザクションのグローバルトランザクション ID (GTID)。

- [LAST_QUEUED_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP](#)

リレーログにキューされた最後のトランザクションが元のソースでコミットされた時間を示す 'YYYY-MM-DD hh:mm:ss[.fraction]' 形式のタイムスタンプ。

- [LAST_QUEUED_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP](#)

リレーログにキューされた最後のトランザクションが即時ソースでコミットされた時間を示す 'YYYY-MM-DD hh:mm:ss[.fraction]' 形式のタイムスタンプ。

- [LAST_QUEUED_TRANSACTION_START_QUEUE_TIMESTAMP](#)

この I/O スレッドによって最後のトランザクションがリレーログキューに置かれた時間を示す 'YYYY-MM-DD hh:mm:ss[.fraction]' 形式のタイムスタンプ。

- [LAST_QUEUED_TRANSACTION_END_QUEUE_TIMESTAMP](#)

最後のトランザクションがリレーログファイルにキューに入れられた時間を示す 'YYYY-MM-DD hh:mm:ss[.fraction]' 形式のタイムスタンプ。

- [QUEUEING_TRANSACTION](#)

リレーログ内の現在キューイングしているトランザクションのグローバルトランザクション ID (GTID)。

- [QUEUEING_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP](#)

現在キューイングしているトランザクションが元のソースでコミットされた時間を示す 'YYYY-MM-DD hh:mm:ss[.fraction]' 形式のタイムスタンプ。

- [QUEUEING_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP](#)

現在キューイングしているトランザクションが即時ソースでコミットされた時間を示す 'YYYY-MM-DD hh:mm:ss[.fraction]' 形式のタイムスタンプ。

- [QUEUEING_TRANSACTION_START_QUEUE_TIMESTAMP](#)

現在キューイングしているトランザクションの最初のイベントがこの I/O スレッドによってリレーログに書き込まれた時間を示す「YYYY-MM-DD hh:mm:ss[.fraction]」形式のタイムスタンプ。

パフォーマンススキーマが無効になっている場合、ローカルタイミング情報は収集されないため、キューに入れられたトランザクションの開始タイムスタンプと終了タイムスタンプを示すフィールドはゼロになります。

[replication_connection_status](#) テーブルには次のインデックスがあります:

- 主キー ([CHANNEL_NAME](#))
- ([THREAD_ID](#)) のインデックス

次のテーブルに、[replication_connection_status](#) カラムと [SHOW REPLICA | SLAVE STATUS](#) カラムの対応を示します。

replication_connection_status カラム	SHOW REPLICA SLAVE STATUS カラム
SOURCE_UUID	Master_UUID
THREAD_ID	なし
SERVICE_STATE	Replica_IO_Running
RECEIVED_TRANSACTION_SET	Retrieved_Gtid_Set
LAST_ERROR_NUMBER	Last_IO_Errno
LAST_ERROR_MESSAGE	Last_IO_Error
LAST_ERROR_TIMESTAMP	Last_IO_Error_Timestamp

27.12.11.3 [replication_asynchronous_connection_failover](#) テーブル

このテーブルには、非同期接続フェイルオーバーメカニズムの各レプリケーションチャンネルのレプリカソースリストが含まれます。非同期接続フェイルオーバーメカニズムは、レプリカからソースへの既存の接続が失敗した後、適切なリストから新しいソースへの非同期 (ソースからレプリカへの) レプリケーション接続を自動的に確立します。[asynchronous_connection_failover_add_source](#) および [asynchronous_connection_failover_delete_source](#) UDF を使用してソースリストを設定および管理し、レプリケーションチャンネルのソースリストに対してレプリケーションソースサーバーを追加および削除します。サーバーの管理対象グループを追加および削除するには、代わりに [asynchronous_connection_failover_add_managed](#) および [asynchronous_connection_failover_delete_managed](#) UDF を使用します。詳細は、[セクション17.4.9「非同期接続フェイルオーバーによるソースの切替え」](#)を参照してください。

[replication_asynchronous_connection_failover](#) テーブルには、次のカラムがあります:

- [CHANNEL_NAME](#)

このレプリケーションソースサーバーがソースリストの一部であるレプリケーションチャンネル。このチャンネルから現在のソースへの接続に失敗した場合、このレプリケーションソースサーバーは潜在的な新しいソースの1つです。

- [HOST](#)

このレプリケーションソースサーバーのホスト名。

- [PORT](#)

このレプリケーションソースサーバーのポート番号。

- [NETWORK_NAMESPACE](#)

このレプリケーションソースサーバーのネットワークネームスペース。この値が空の場合、接続ではデフォルト (グローバル) のネームスペースが使用されます。

- [WEIGHT](#)

レプリケーションチャンネルソースリスト内のこのレプリケーションソースサーバーの優先度。重みは 1 から 100 で、100 が最も高く、50 がデフォルトです。非同期接続フェイルオーバーメカニズムがアクティブ化されると、チャンネルのソースリストにリストされている代替ソースの中で重みが最も高いソースが最初の接続試行に対して選択されます。この試行が機能しない場合、レプリカは、リストされているすべてのソースを重みの降順で試行してから、最も高い重みのソースから再開します。複数のソースの重みが同じ場合、レプリカはそれらをランダムに順序付けします。

- **MANAGED_NAME**

サーバーが属する管理対象グループの識別子。GroupReplication 管理サービスの場合、identifier は `group_replication_group_name` システム変数の値です。

`replication_asynchronous_connection_failover` テーブルには次のインデックスがあります:

- 主キー (CHANNEL_NAME, HOST, PORT, NETWORK_NAMESPACE, MANAGED_NAME)

TRUNCATE TABLE は、`replication_asynchronous_connection_failover` テーブルに対して許可されていません。

27.12.11.4 replication_applier_configuration テーブル

このテーブルは、レプリカによって適用されるトランザクションに影響する構成パラメータを示しています。テーブルに格納されているパラメータは、実行時に `CHANGE REPLICATION SOURCE TO` ステートメント (MySQL 8.0.23) または `CHANGE MASTER TO` ステートメント (MySQL 8.0.23 より前) を使用して変更できます。

`replication_applier_configuration` テーブルには、次のカラムがあります:

- **CHANNEL_NAME**

この行が表示しているレプリケーションチャンネル。常にデフォルトのレプリケーションチャンネルがあり、さらにレプリケーションチャンネルを追加できます。詳しくは [セクション17.2.2 「レプリケーションチャンネル」](#) をご覧ください。

- **DESIRED_DELAY**

レプリカがソースを遅らせる必要がある秒数。(`CHANGE REPLICATION SOURCE TO` オプション: `SOURCE_DELAY`、`CHANGE MASTER TO` オプション: `MASTER_DELAY`) 詳細は、[セクション17.4.11 「遅延レプリケーション」](#) を参照してください。

- **PRIVILEGE_CHECKS_USER**

チャンネルのセキュリティコンテキストを提供するユーザーアカウント (`CHANGE REPLICATION SOURCE TO` オプション: `PRIVILEGE_CHECKS_USER`、`CHANGE MASTER TO` オプション: `PRIVILEGE_CHECKS_USER`)。これはエスケープされるため、SQL ステートメントにコピーして個々のトランザクションを実行できます。詳しくは [セクション17.3.3 「レプリケーション権限チェック」](#) をご覧ください。

- **REQUIRE_ROW_FORMAT**

チャンネルが行ベースのイベントのみを受け入れるかどうか (`CHANGE REPLICATION SOURCE TO` オプション: `REQUIRE_ROW_FORMAT`、`CHANGE MASTER TO` オプション: `REQUIRE_ROW_FORMAT`)。詳しくは [セクション17.3.3 「レプリケーション権限チェック」](#) をご覧ください。

- **REQUIRE_TABLE_PRIMARY_KEY_CHECK**

チャンネルで主キーが常に必要かどうか、またはソース設定に応じて必要かどうか (`CHANGE REPLICATION SOURCE TO` オプション: `REQUIRE_TABLE_PRIMARY_KEY_CHECK`、`CHANGE MASTER TO` オプション: `REQUIRE_TABLE_PRIMARY_KEY_CHECK`)。詳しくは [セクション17.3.3 「レプリケーション権限チェック」](#) をご覧ください。

- **ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS_TYPE**

GTID がまだないレプリケートされたトランザクションにチャンネルが GTID を割り当てるかどうか (`CHANGE REPLICATION SOURCE TO` オプション: `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS`、`CHANGE`

MASTER TO オプション: `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS`). OFF は GTID が割り当てられていないことを示します。LOCAL は、レプリカ独自の UUID (`server_uuid` 設定) を含む GTID が割り当てられていることを意味します。MANUAL は、手動で設定された UUID を含む GTID が割り当てられることを意味します。詳しくは [セクション 17.1.3.6 「GTID のないソースから GTID のあるレプリカへのレプリケーション」](#) をご覧ください。

- `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS_VALUE`

匿名トランザクションに割り当てられた GTID の一部として使用される UUID (`CHANGE REPLICATION SOURCE TO` オプション: `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS`, `CHANGE MASTER TO` オプション: `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS`). 詳しくは [セクション 17.1.3.6 「GTID のないソースから GTID のあるレプリカへのレプリケーション」](#) をご覧ください。

`replication_applier_configuration` テーブルには次のインデックスがあります:

- 主キー (`CHANNEL_NAME`)

TRUNCATE TABLE は、`replication_applier_configuration` テーブルに対して許可されていません。

次のテーブルに、`replication_applier_configuration` カラムと `SHOW REPLICA | SLAVE STATUS` カラムの対応を示します。

<code>replication_applier_configuration</code> カラム	<code>SHOW REPLICA SLAVE STATUS</code> カラム
<code>DESIRED_DELAY</code>	<code>SQL_Delay</code>

27.12.11.5 `replication_applier_status` テーブル

このテーブルには、レプリカの現在の一般的なトランザクション実行ステータスが表示されます。このテーブルは、関連するスレッドに固有ではないトランザクションアプライヤステータスの一般的な側面に関する情報を示しています。スレッド固有のステータス情報は、`replication_applier_status_by_coordinator` テーブル (およびレプリカがマルチスレッドの場合は `replication_applier_status_by_worker`) で使用できます。

`replication_applier_status` テーブルには、次のカラムがあります:

- `CHANNEL_NAME`

この行が表示しているレプリケーションチャンネル。常にデフォルトのレプリケーションチャンネルがあり、さらにレプリケーションチャンネルを追加できます。詳しくは [セクション 17.2.2 「レプリケーションチャンネル」](#) をご覧ください。

- `SERVICE_STATE`

レプリケーションチャンネルアプライヤスレッドがアクティブまたはアイドルの場合に `ON` を表示します。OFF は、アプライヤスレッドがアクティブでないことを意味します。

- `REMAINING_DELAY`

ソースがトランザクションを適用してから `DESIRED_DELAY` 秒が経過するまでレプリカが待機している場合、このフィールドには残りの遅延秒数が表示されます。ほかのときは、このフィールドは `NULL` です。(`DESIRED_DELAY` 値は `replication_applier_configuration` テーブルに格納されます。) 詳しくは [セクション 17.4.11 「遅延レプリケーション」](#) をご覧ください。

- `COUNT_TRANSACTIONS_RETRIES`

レプリケーション SQL スレッドがトランザクションの適用に失敗したために行われた再試行の回数を示します。特定のトランザクションの最大再試行回数は、`slave_transaction_retries` システム変数によって設定されます。`replication_applier_status_by_worker` テーブルには、シングルスレッドまたはマルチスレッドレプリカのトランザクション再試行に関する詳細情報が表示されます。

`replication_applier_status` テーブルには次のインデックスがあります:

- 主キー (`CHANNEL_NAME`)

TRUNCATE TABLE は、[replication_applier_status](#) テーブルに対して許可されていません。

次のテーブルに、[replication_applier_status](#) カラムと [SHOW REPLICA | SLAVE STATUS](#) カラムの対応を示します。

replication_applier_status カラム	SHOW REPLICA SLAVE STATUS カラム
SERVICE_STATE	なし
REMAINING_DELAY	SQL_Remaining_Delay

27.12.11.6 replication_applier_status_by_coordinator テーブル

マルチスレッドのレプリカの場合、レプリカは複数のワーカースレッドとコーディネータスレッドを使用してそれらを管理し、このテーブルにはコーディネータスレッドのステータスが表示されます。シングルスレッドレプリカの場合、このテーブルは空です。マルチスレッドのレプリカの場合、[replication_applier_status_by_worker](#) テーブルにワーカースレッドのステータスが表示されます。このテーブルには、コーディネータスレッドによってワーカーのキューにバッファリングされた最後のトランザクションと、現在バッファリングしているトランザクションに関する情報が表示されます。開始タイムスタンプは、このスレッドがトランザクションの最初のイベントをリレーログから読み取ってワーカーのキューにバッファするタイミングを表し、終了タイムスタンプは最後のイベントがワーカーのキューへのバッファリングを終了したタイミングを表します。

[replication_applier_status_by_coordinator](#) テーブルには、次のカラムがあります：

- [CHANNEL_NAME](#)

この行が表示しているレプリケーションチャンネル。常にデフォルトのレプリケーションチャンネルがあり、さらにレプリケーションチャンネルを追加できます。詳しくは[セクション17.2.2「レプリケーションチャンネル」](#)をご覧ください。

- [THREAD_ID](#)

SQL/コーディネータスレッド ID。

- [SERVICE_STATE](#)

[ON](#) (スレッドが存在し、アクティブまたはアイドル状態) または [OFF](#) (スレッドは存在しません)。

- [LAST_ERROR_NUMBER](#), [LAST_ERROR_MESSAGE](#)

SQL/コーディネータスレッドの停止の原因となった最新のエラーのエラー番号およびエラーメッセージ。エラー番号 0 および空の文字列であるメッセージは、「エラーなし」を意味します。[LAST_ERROR_MESSAGE](#) 値が空でない場合、エラー値はレプリカエラーログにも表示されます。

[RESET MASTER](#) または [RESET REPLICA | SLAVE](#) を発行すると、これらのカラムに表示される値がリセットされます。

[LAST_ERROR_NUMBER](#) カラムおよび [LAST_ERROR_MESSAGE](#) カラムに表示されるすべてのエラーコードおよびメッセージは、[Server Error Message Reference](#) にリストされているエラー値に対応します。

- [LAST_ERROR_TIMESTAMP](#)

最新の SQL/コーディネータエラーがいつ発生したかを示す'[YYYY-MM-DD hh:mm:ss\[.fraction\]](#)'形式のタイムスタンプ。

- [LAST_PROCESSED_TRANSACTION](#)

このコーディネータによって最後に処理されたトランザクションのグローバルトランザクション ID (GTID)。

- [LAST_PROCESSED_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP](#)

このコーディネータによって処理された最後のトランザクションが元のソースでコミットされた時間を示す'[YYYY-MM-DD hh:mm:ss\[.fraction\]](#)'形式のタイムスタンプ。

- [LAST_PROCESSED_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP](#)

このコーディネータによって処理された最後のトランザクションが即時ソースでコミットされた時間を示す'YYYY-MM-DD hh:mm:ss[.fraction]'形式のタイムスタンプ。

- `LAST_PROCESSED_TRANSACTION_START_BUFFER_TIMESTAMP`

このコーディネータスレッドがワーカースレッドのバッファへの最後のトランザクションの書き込みを開始した時間を示す'YYYY-MM-DD hh:mm:ss[.fraction]'形式のタイムスタンプ。

- `LAST_PROCESSED_TRANSACTION_END_BUFFER_TIMESTAMP`

このコーディネータスレッドによってワーカースレッドのバッファに最後のトランザクションが書き込まれた時間を示す'YYYY-MM-DD hh:mm:ss[.fraction]'形式のタイムスタンプ。

- `PROCESSING_TRANSACTION`

このコーディネータスレッドが現在処理しているトランザクションのグローバルトランザクション ID (GTID)。

- `PROCESSING_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP`

現在処理中のトランザクションが元のソースでコミットされた時間を示す'YYYY-MM-DD hh:mm:ss[.fraction]'形式のタイムスタンプ。

- `PROCESSING_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP`

現在処理中のトランザクションが即時ソースでコミットされた時間を示す'YYYY-MM-DD hh:mm:ss[.fraction]'形式のタイムスタンプ。

- `PROCESSING_TRANSACTION_START_BUFFER_TIMESTAMP`

このコーディネータスレッドが現在処理中のトランザクションのワーカースレッドのバッファへの書き込みを開始した時間を示す'YYYY-MM-DD hh:mm:ss[.fraction]'形式のタイムスタンプ。

パフォーマンススキーマが無効になっている場合、ローカルタイミング情報は収集されないため、バッファートランザクションの開始タイムスタンプと終了タイムスタンプを示すフィールドはゼロになります。

`replication_applier_status_by_coordinator` テーブルには次のインデックスがあります:

- 主キー (`CHANNEL_NAME`)
- (`THREAD_ID`) のインデックス

次のテーブルに、`replication_applier_status_by_coordinator` カラムと `SHOW REPLICA | SLAVE STATUS` カラムの対応を示します。

<code>replication_applier_status_by_coordinator</code> カラム	<code>SHOW REPLICA SLAVE STATUS</code> カラム
<code>THREAD_ID</code>	なし
<code>SERVICE_STATE</code>	<code>Replica_SQL_Running</code>
<code>LAST_ERROR_NUMBER</code>	<code>Last_SQL_Errno</code>
<code>LAST_ERROR_MESSAGE</code>	<code>Last_SQL_Error</code>
<code>LAST_ERROR_TIMESTAMP</code>	<code>Last_SQL_Error_Timestamp</code>

27.12.11.7 `replication_applier_status_by_worker` テーブル

このテーブルには、レプリカまたはグループレプリケーショングループメンバーの適用者スレッドによって処理されるトランザクションの詳細が示されます。シングルスレッドのレプリカの場合、レプリカの単一アプライヤスレッドのデータが表示されます。マルチスレッドのレプリカの場合、データはアプライヤスレッドごとに個別に表示されます。マルチスレッドレプリカ上のアプライヤスレッドは、ワーカーと呼ばれることもあります。レプリカまたはグループレプリケーショングループメンバー上のアプライヤスレッドの数は、シングルスレッドレプリカの場合はゼロに設定される `slave_parallel_workers` システム変数によって設定されます。マルチスレッドレ

リカには、アプライヤスレッドを管理するためのコーディネータスレッドもあり、このスレッドのステータスは `replication_applier_status_by_coordinator` テーブルに表示されます。

エラーに関連するカラムに表示されるすべてのエラーコードおよびメッセージは、[Server Error Message Reference](#) にリストされているエラー値に対応します。

パフォーマンススキーマが無効になっている場合、ローカルのタイミング情報は収集されないため、適用されるトランザクションの開始タイムスタンプと終了タイムスタンプを示すフィールドはゼロになります。このテーブルの開始タイムスタンプは、ワーカーが最初のイベントの適用を開始した時間を示し、終了タイムスタンプは、トランザクションの最後のイベントが適用された時間を示します。

`START REPLICASlave` ステートメントによってレプリカが再起動されると、`APPLYING_TRANSACTION` で始まるカラムがリセットされます。MySQL 8.0.13 より前は、これらのカラムはシングルスレッドモードで動作していたレプリカではリセットされず、マルチスレッドレプリカでのみリセットされていました。

`replication_applier_status_by_worker` テーブルには、次のカラムがあります:

- `CHANNEL_NAME`

この行が表示しているレプリケーションチャンネル。常にデフォルトのレプリケーションチャンネルがあり、さらにレプリケーションチャンネルを追加できます。詳しくは[セクション17.2.2「レプリケーションチャンネル」](#)をご覧ください。

- `WORKER_ID`

ワーカー識別子 (`mysql.slave_worker_info` テーブルの `id` カラムと同じ値)。`STOP REPLICASlave` の後、`THREAD_ID` カラムは `NULL` になりますが、`WORKER_ID` 値は保持されます。

- `THREAD_ID`

ワーカースレッド ID。

- `SERVICE_STATE`

`ON` (スレッドが存在し、アクティブまたはアイドル状態) または `OFF` (スレッドは存在しません)。

- `LAST_ERROR_NUMBER, LAST_ERROR_MESSAGE`

ワーカースレッドの停止の原因となった最新のエラーのエラー番号およびエラーメッセージ。0 のエラー番号および空の文字列のメッセージは、「エラーなし」を示します `LAST_ERROR_MESSAGE` 値が空でない場合、エラー値はレプリカエラーログにも表示されます。

`RESET MASTER` または `RESET REPLICASlave` を発行すると、これらのカラムに表示される値がリセットされます。

- `LAST_ERROR_TIMESTAMP`

最新のワーカーエラーがいつ発生したかを示す `'YYYY-MM-DD hh:mm:ss[.fraction]'` 形式のタイムスタンプ。

- `LAST_APPLIED_TRANSACTION`

このワーカーによって適用された最後のトランザクションのグローバルトランザクション ID (GTID)。

- `LAST_APPLIED_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP`

このワーカーによって適用された最後のトランザクションが元のソースでコミットされた時間を示す `'YYYY-MM-DD hh:mm:ss[.fraction]'` 形式のタイムスタンプ。

- `LAST_APPLIED_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP`

このワーカーによって適用された最後のトランザクションが即時ソースでコミットされた時間を示す `'YYYY-MM-DD hh:mm:ss[.fraction]'` 形式のタイムスタンプ。

- `LAST_APPLIED_TRANSACTION_START_APPLY_TIMESTAMP`

このワーカーが最後に適用されたトランザクションの適用をいつ開始したかを示す'YYYY-MM-DD hh:mm:ss[.fraction]'形式のタイムスタンプ。

- `LAST_APPLIED_TRANSACTION_END_APPLY_TIMESTAMP`

このワーカーが最後に適用されたトランザクションの適用を終了した時間を示す'YYYY-MM-DD hh:mm:ss[.fraction]'形式のタイムスタンプ。

- `APPLYING_TRANSACTION`

このワーカーが現在適用しているトランザクションのグローバルトランザクション ID (GTID)。

- `APPLYING_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP`

このワーカーが現在適用しているトランザクションが元のソースでコミットされた時間を示す'YYYY-MM-DD hh:mm:ss[.fraction]'形式のタイムスタンプ。

- `APPLYING_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP`

このワーカーが現在適用しているトランザクションが即時ソースでコミットされた時間を示す'YYYY-MM-DD hh:mm:ss[.fraction]'形式のタイムスタンプ。

- `APPLYING_TRANSACTION_START_APPLY_TIMESTAMP`

このワーカーが現在適用されているトランザクションを最初に適用しようとした時間を示す'YYYY-MM-DD hh:mm:ss[.fraction]'形式のタイムスタンプ。MySQL 8.0.13 より前は、このタイムスタンプは一時エラーのためにトランザクションが再試行されたときにリフレッシュされたため、トランザクションを適用しようとした最新のタイムスタンプが表示されていました。

- `LAST_APPLIED_TRANSACTION_RETRIES_COUNT`

最初の試行後に最後に適用されたトランザクションがワーカーによって再試行された回数。最初の試行でトランザクションが適用された場合、この数値はゼロです。

- `LAST_APPLIED_TRANSACTION_LAST_TRANSIENT_ERROR_NUMBER`

トランザクションが再試行される原因となった最後の一時エラーのエラー番号。

- `LAST_APPLIED_TRANSACTION_LAST_TRANSIENT_ERROR_MESSAGE`

トランザクションが再試行される原因となった最後の一時エラーのメッセージテキスト。

- `LAST_APPLIED_TRANSACTION_LAST_TRANSIENT_ERROR_TIMESTAMP`

トランザクションが再試行される原因となった最後の一時エラーの'YYYY-MM-DD hh:mm:ss[.fraction]'形式のタイムスタンプ。

- `APPLYING_TRANSACTION_RETRIES_COUNT`

現在適用されているトランザクションがこの時点までに再試行された回数。最初の試行でトランザクションが適用された場合、この数値はゼロです。

- `APPLYING_TRANSACTION_LAST_TRANSIENT_ERROR_NUMBER`

現在のトランザクションが再試行される原因となった最後の一時エラーのエラー番号。

- `APPLYING_TRANSACTION_LAST_TRANSIENT_ERROR_MESSAGE`

現在のトランザクションが再試行される原因となった最後の一時エラーのメッセージテキスト。

- `APPLYING_TRANSACTION_LAST_TRANSIENT_ERROR_TIMESTAMP`

現在のトランザクションが再試行される原因となった最後の一時エラーの'YYYY-MM-DD hh:mm:ss[.fraction]'形式のタイムスタンプ。

`replication_applier_status_by_worker` テーブルには次のインデックスがあります:

- 主キー (`CHANNEL_NAME`、`WORKER_ID`)
- (`THREAD_ID`) のインデックス

次のテーブルに、`replication_applier_status_by_worker` カラムと `SHOW REPLICA | SLAVE STATUS` カラムの対応を示します。

<code>replication_applier_status_by_worker</code> カラム	<code>SHOW REPLICA SLAVE STATUS</code> カラム
<code>WORKER_ID</code>	なし
<code>THREAD_ID</code>	なし
<code>SERVICE_STATE</code>	なし
<code>LAST_ERROR_NUMBER</code>	<code>Last_SQL_Errno</code>
<code>LAST_ERROR_MESSAGE</code>	<code>Last_SQL_Error</code>
<code>LAST_ERROR_TIMESTAMP</code>	<code>Last_SQL_Error_Timestamp</code>

27.12.11.8 `replication_applier_global_filters` テーブル

このテーブルには、このレプリカに構成されているグローバルレプリケーションフィルタが表示されます。
`replication_applier_global_filters` テーブルには、次のカラムがあります:

- `FILTER_NAME`

構成されているレプリケーションフィルタのタイプ。

- `FILTER_RULE`

`--replicate-*` コマンドオプションまたは `CHANGE REPLICATION FILTER` を使用してレプリケーションフィルタタイプに構成されたルール。

- `CONFIGURED_BY`

レプリケーションフィルタの構成に使用される方法は、次のいずれかです:

- `CHANGE REPLICATION FILTER` ステートメントを使用してグローバルレプリケーションフィルタによって構成された `CHANGE_REPLICATION_FILTER`。
- `--replicate-*` オプションを使用してグローバルレプリケーションフィルタによって構成された `STARTUP_OPTIONS`。

- `ACTIVE_SINCE`

レプリケーションフィルタが構成された時点のタイムスタンプ。

27.12.11.9 `replication_applier_filters` テーブル

このテーブルには、このレプリカに構成されているレプリケーションチャンネル固有のフィルタが表示されます。各行には、レプリケーションチャンネルで構成されたフィルタのタイプに関する情報が表示されます。
`replication_applier_filters` テーブルには、次のカラムがあります:

- `CHANNEL_NAME`

レプリケーションフィルタが構成されているレプリケーションチャンネルの名前。

- `FILTER_NAME`

このレプリケーションチャンネルに構成されているレプリケーションフィルタのタイプ。

- `FILTER_RULE`

--replicate-* コマンドオプションまたは [CHANGE REPLICATION FILTER](#) を使用してレプリケーションフィルタタイプに構成されたルール。

- [CONFIGURED_BY](#)

レプリケーションフィルタの構成に使用される方法は、次のいずれかです:

- [CHANGE REPLICATION FILTER](#) ステートメントを使用してグローバルレプリケーションフィルタによって構成された [CHANGE_REPLICATION_FILTER](#)。
- --replicate-* オプションを使用してグローバルレプリケーションフィルタによって構成された [STARTUP_OPTIONS](#)。
- [CHANGE REPLICATION FILTER FOR CHANNEL](#) ステートメントを使用してチャンネル固有のレプリケーションフィルタによって構成された [CHANGE_REPLICATION_FILTER_FOR_CHANNEL](#)。
- --replicate-* オプションを使用してチャンネル固有のレプリケーションフィルタによって構成された [STARTUP_OPTIONS_FOR_CHANNEL](#)。

- [ACTIVE_SINCE](#)

レプリケーションフィルタが構成された時点のタイムスタンプ。

- [COUNTER](#)

レプリケーションフィルタが構成されてから使用された回数。

27.12.11.10 replication_group_members テーブル

このテーブルは、レプリケーショングループメンバーのネットワークおよびステータス情報を示しています。表示されるネットワークアドレスは、クライアントをグループに接続するために使用されるアドレスであり、[group_replication_local_address](#) で指定されたメンバー内部グループ通信アドレスと混同しないでください。

[replication_group_members](#) テーブルには、次のカラムがあります:

- [CHANNEL_NAME](#)

グループレプリケーションチャンネルの名前。

- [MEMBER_ID](#)

メンバーサーバー UUID。これは、グループ内のメンバーごとに異なる値を持ちます。これは、各メンバーに一意であるため、キーとしても機能します。

- [MEMBER_HOST](#)

このメンバーのネットワークアドレス (ホスト名または IP アドレス)。メンバー [hostname](#) 変数から取得されます。これは、内部グループ通信に使用される [group_replication_local_address](#) とは異なり、クライアントが接続するアドレスです。

- [MEMBER_PORT](#)

サーバーがリスニングしているポート。メンバー [port](#) 変数から取得されます。

- [MEMBER_STATE](#)

このメンバーの現在の状態。次のいずれかになります:

- [ONLINE](#): メンバーは完全に機能している状態です。
- [RECOVERING](#): サーバーは、データの取得元のグループに参加しました。
- [OFFLINE](#): グループレプリケーションプラグインがインストールされていますが、起動されていません。

- **ERROR**: トランザクションの適用中またはリカバリフェーズ中にメンバーでエラーが発生し、グループトランザクションに参加していません。
- **UNREACHABLE**: 障害検出プロセスでは、グループメッセージがタイムアウトしたため、このメンバーに接続できないと疑われます。

セクション18.3.1「グループレプリケーションサーバーの状態」を参照してください。

- **MEMBER_ROLE**

グループ内のメンバーのロール (**PRIMARY** または **SECONDARY**)。

- **MEMBER_VERSION**

メンバーの MySQL バージョン。

`replication_group_members` テーブルには次のインデックスがあります:

- なし

`TRUNCATE TABLE` は、`replication_group_members` テーブルに対して許可されていません。

27.12.11.11 replication_group_member_stats テーブル

このテーブルは、レプリケーショングループメンバーの統計情報を示しています。これは、Group Replication が実行されている場合にのみ移入されます。

`replication_group_member_stats` テーブルには、次のカラムがあります:

- **CHANNEL_NAME**

グループレプリケーションチャンネルの名前

- **VIEW_ID**

このグループの現在のビュー識別子。

- **MEMBER_ID**

メンバーサーバー UUID。これは、グループ内のメンバーごとに異なる値を持ちます。これは、各メンバーに一意であるため、キーとしても機能します。

- **COUNT_TRANSACTIONS_IN_QUEUE**

キュー内の保留中の競合検出チェックのトランザクション数。トランザクションの競合がチェックされた後、チェックに合格すると、トランザクションも適用されるようにキューに入れられます。

- **COUNT_TRANSACTIONS_CHECKED**

競合がチェックされたトランザクションの数。

- **COUNT_CONFLICTS_DETECTED**

競合検出チェックに合格していないトランザクションの数。

- **COUNT_TRANSACTIONS_ROWS_VALIDATING**

動作保証に使用できるがガベージコレクションされていないトランザクション行の数。各トランザクションが動作保証されている競合検出データベースの現在のサイズと考えることができます。

- **TRANSACTIONS_COMMITTED_ALL_MEMBERS**

レプリケーショングループのすべてのメンバーで正常にコミットされたトランザクション (**GTID セット** として表示)。これは固定時間間隔で更新されます。

- [LAST_CONFLICT_FREE_TRANSACTION](#)
チェックされた最後の競合解消トランザクションのトランザクション識別子。
- [COUNT_TRANSACTIONS_REMOTE_IN_APPLIER_QUEUE](#)
このメンバーがレプリケーショングループから受信し、適用を待機しているトランザクションの数。
- [COUNT_TRANSACTIONS_REMOTE_APPLIED](#)
このメンバーがグループから受信して適用したトランザクションの数。
- [COUNT_TRANSACTIONS_LOCAL_PROPOSED](#)
このメンバーで発生し、グループに送信されたトランザクションの数。
- [COUNT_TRANSACTIONS_LOCAL_ROLLBACK](#)
このメンバーで発生し、グループによってロールバックされたトランザクションの数。

[replication_group_member_stats](#) テーブルには次のインデックスがあります:

- なし

[TRUNCATE TABLE](#) は、[replication_group_member_stats](#) テーブルに対して許可されていません。

27.12.11.12 [binary_log_transaction_compression_stats](#) テーブル

このテーブルは、バイナリログおよびリレーログに書き込まれたトランザクションペイロードの統計情報を示しており、バイナリログトランザクション圧縮を有効にした場合の影響を計算するために使用できます。バイナリログトランザクションの圧縮については、[セクション5.4.4.5「バイナリログトランザクション圧縮」](#)を参照してください。

[binary_log_transaction_compression_stats](#) テーブルは、サーバーインスタンスにバイナリログがあり、システム変数 [binlog_transaction_compression](#) が [ON](#) に設定されている場合のみ移入されます。統計には、サーバーの起動時またはテーブルの切り捨て時からバイナリログおよびリレーログに書き込まれたすべてのトランザクションが含まれます。圧縮されたトランザクションは使用される圧縮アルゴリズムによってグループ化され、圧縮されていないトランザクションは [NONE](#) として示された圧縮アルゴリズムとともにグループ化されるため、圧縮率を計算できます。

[binary_log_transaction_compression_stats](#) テーブルには、次のカラムがあります:

- [LOG_TYPE](#)
これらのトランザクションがバイナリログに書き込まれたかリレーログに書き込まれたか。
- [COMPRESSION_TYPE](#)
トランザクションペイロードの圧縮に使用される圧縮アルゴリズム。 [NONE](#) は、これらのトランザクションのペイロードが圧縮されなかったことを意味します。これは、多くの状況で適切です ([セクション5.4.4.5「バイナリログトランザクション圧縮」](#)を参照)。
- [TRANSACTION_COUNTER](#)
この圧縮タイプでこのログタイプに書き込まれたトランザクションの数。
- [COMPRESSED_BYTES](#)
圧縮され、圧縮後にこの圧縮タイプでこのログタイプに書き込まれた合計バイト数。
- [UNCOMPRESSED_BYTES](#)
このログタイプおよびこの圧縮タイプの圧縮前の合計バイト数。
- [COMPRESSION_PERCENTAGE](#)
このログタイプおよびこの圧縮タイプの圧縮率で、パーセンテージで表されます。

- [FIRST_TRANSACTION_ID](#)
この圧縮タイプでこのログタイプに書き込まれた最初のトランザクションの ID。
- [FIRST_TRANSACTION_COMPRESSED_BYTES](#)
圧縮後に最初のトランザクションで圧縮されてログに書き込まれた合計バイト数。
- [FIRST_TRANSACTION_UNCOMPRESSED_BYTES](#)
最初のトランザクションの圧縮前の合計バイト数。
- [FIRST_TRANSACTION_TIMESTAMP](#)
最初のトランザクションがログに書き込まれたときのタイムスタンプ。
- [LAST_TRANSACTION_ID](#)
この圧縮タイプでこのログタイプに書き込まれた最新のトランザクションの ID。
- [LAST_TRANSACTION_COMPRESSED_BYTES](#)
圧縮後にカウントされた、圧縮されて最新のトランザクションのログに書き込まれた合計バイト数。
- [LAST_TRANSACTION_UNCOMPRESSED_BYTES](#)
最新のトランザクションの圧縮前の合計バイト数。
- [LAST_TRANSACTION_TIMESTAMP](#)
最新のトランザクションがログに書き込まれたときのタイムスタンプ。

[binary_log_transaction_compression_stats](#) テーブルには次のインデックスがあります:

- なし

`TRUNCATE TABLE` は [binary_log_transaction_compression_stats](#) テーブルに対して許可されています。

27.12.12 パフォーマンススキーマ NDB Cluster テーブル

NDB 8.0.16 以降、NDB の自動同期は NDB Cluster 内部ディクショナリと MySQL Server データディクショナリとの間のメタデータのすべての不一致を自動的に検出して同期しようとしています。これは、[ndb_metadata_check](#) を使用して無効にしたり、[ndb_metadata_sync](#) を設定してオーバーライドしたりしないかぎり、デフォルトで [ndb_metadata_check_interval](#) システム変数によって決定される一定の間隔でバックグラウンドで実行されます。NDB 8.0.21 より前は、このプロセスに関してユーザーがすぐにアクセスできる情報は、ステータス変数 [Ndb_metadata_detected_count](#)、[Ndb_metadata_synced_count](#)、および [Ndb_metadata_excluded_count](#) (NDB 8.0.22 より前) として使用可能なロギングメッセージとオブジェクト数の形式のみで利用可能でした (NDB 8.0.18 から)、(ステータス変数 [Ndb_metadata_detected_count](#)、[Ndb_metadata_synced_count](#)、および [Ndb_metadata_excluded_count](#)) (NDB 8.0.22 以前、この変数は [Ndb_metadata_blacklist_size](#) という名前でした)。NDB 8.0.21 以降、自動同期の現在の状態に関するより詳細な情報は、NDB クラスター内で次の「パフォーマンススキーマ」テーブルで SQL ノードとして機能する MySQL サーバーによって公開されます:

- [ndb_sync_pending_objects](#): NDB ディクショナリと MySQL データディクショナリの間で不一致が検出された NDB データベースオブジェクトに関する情報を表示します。このようなオブジェクトを同期しようとする、NDB は同期を待機しているキューおよびこのテーブルからオブジェクトを削除し、不一致の調整を試みます。一時エラーのためにオブジェクトの同期が失敗した場合、次回 NDB が不一致検出を実行したときに、オブジェクトが取得されてキュー (およびこのテーブル) に追加されます。永続エラーのために試行が失敗した場合、オブジェクトは [ndb_sync_excluded_objects](#) テーブルに追加されます。
- [ndb_sync_excluded_objects](#): 不一致による永続的エラーのために自動同期が失敗した NDB データベースオブジェクトに関する情報を表示します。これらのオブジェクトは手動操作なしではリコンサイルできません。これらのオブジェクトはブロックされ、これが完了するまで不一致検出のために再度考慮されません。

[ndb_sync_pending_objects](#) および [ndb_sync_excluded_objects](#) テーブルは、MySQL で NDBCLUSTER ストレージエンジンのサポートが有効になっている場合にのみ存在します。

これらのテーブルについては、次の2つのセクションで詳しく説明します。

27.12.12.1 ndb_sync_pending_objects テーブル

このテーブルは、不一致が検出され、NDB デクシヨナリと MySQL データデクシヨナリ間の同期を待機している NDB データベースオブジェクトに関する情報を示します。

同期を待機している NDB データベースオブジェクトに関する情報の例:

```
mysql> SELECT * FROM performance_schema.ndb_sync_pending_objects;
+-----+-----+-----+
| SCHEMA_NAME | NAME | TYPE      |
+-----+-----+-----+
| NULL        | lg1  | LOGFILE GROUP |
| NULL        | ts1  | TABLESPACE |
| db1         | NULL | SCHEMA      |
| test        | t1   | TABLE      |
| test        | t2   | TABLE      |
| test        | t3   | TABLE      |
+-----+-----+-----+
```

ndb_sync_pending_objects テーブルには、次のカラムがあります:

- **SCHEMA_NAME**: 同期を待機しているオブジェクトが存在するスキーマ (データベース) の名前。これは、テーブルスペースおよびログファイルグループの **NULL** です
- **NAME**: 同期を待機しているオブジェクトの名前。オブジェクトがスキーマの場合は **NULL** です
- **TYPE**: 同期を待機しているオブジェクトのタイプ (**LOGFILE GROUP**, **TABLESPACE**, **SCHEMA** または **TABLE** のいずれか)

NDB 8.0.21 に ndb_sync_pending_objects テーブルが追加されました。

27.12.12.2 ndb_sync_excluded_objects テーブル

このテーブルは、NDB Cluster デクシヨナリと MySQL データデクシヨナリの間で自動的に同期できない NDB データベースオブジェクトに関する情報を提供します。

MySQL データデクシヨナリと同期できない NDB データベースオブジェクトに関する情報の例:

```
mysql> SELECT * FROM performance_schema.ndb_sync_excluded_objects\G
***** 1. row *****
SCHEMA_NAME: NULL
NAME: lg1
TYPE: LOGFILE GROUP
REASON: Injected failure
***** 2. row *****
SCHEMA_NAME: NULL
NAME: ts1
TYPE: TABLESPACE
REASON: Injected failure
***** 3. row *****
SCHEMA_NAME: db1
NAME: NULL
TYPE: SCHEMA
REASON: Injected failure
***** 4. row *****
SCHEMA_NAME: test
NAME: t1
TYPE: TABLE
REASON: Injected failure
***** 5. row *****
SCHEMA_NAME: test
NAME: t2
TYPE: TABLE
REASON: Injected failure
***** 6. row *****
SCHEMA_NAME: test
NAME: t3
TYPE: TABLE
```

REASON: Injected failure

`ndb_sync_excluded_objects` テーブルには、次のカラムがあります:

- **SCHEMA_NAME**: 同期に失敗したオブジェクトが存在するスキーマ (データベース) の名前。これは、テーブルスペースおよびログファイルグループの **NULL** です
- **NAME**: 同期に失敗したオブジェクトの名前。オブジェクトがスキーマの場合は **NULL** です
- **TYPE**: オブジェクトのタイプが同期に失敗しました。これは **LOGFILE GROUP**, **TABLESPACE**, **SCHEMA** または **TABLE** のいずれかです
- **REASON**: オブジェクトの除外 (ブロックリスト) の理由、つまり、このオブジェクトの同期化に失敗した理由
考えられる原因は次のとおりです:

- Injected failure
- Failed to determine if object existed in NDB
- Failed to determine if object existed in DD
- Failed to drop object in DD
- Failed to get undofiles assigned to logfile group
- Failed to get object id and version
- Failed to install object in DD
- Failed to get datafiles assigned to tablespace
- Failed to create schema
- Failed to determine if object was a local table
- Failed to invalidate table references
- Failed to set database name of NDB object
- Failed to get extra metadata of table
- Failed to migrate table with extra metadata version 1
- Failed to get object from DD
- Definition of table has changed in NDB Dictionary
- Failed to setup binlogging for table

このリストは必ずしも完全ではなく、将来の **NDB** リリースで変更される可能性があります。

NDB 8.0.21 に `ndb_sync_excluded_objects` テーブルが追加されました。

27.12.13 パフォーマンススキーマロックテーブル

パフォーマンススキーマは、次のテーブルを介してロック情報を公開します:

- **data_locks**: 保持およびリクエストされたデータロック
- **data_lock_waits**: データロック所有者とその所有者によってブロックされたデータロックリクエストの関係
- **metadata_locks**: メタデータロックの保持およびリクエスト
- **table_handles**: 保持およびリクエストされたテーブルロック

次の各セクションでは、これらのテーブルについて詳しく説明します。

27.12.13.1 data_locks テーブル

`data_locks` テーブルには、保持およびリクエストされたデータロックが表示されます。どのロック要求がどの保持ロックによってブロックされるかについては、[セクション27.12.13.2「data_lock_waits テーブル」](#)を参照してください。

データロック情報の例:

```
mysql> SELECT * FROM performance_schema.data_locks\G
***** 1. row *****
ENGINE: INNODB
ENGINE_LOCK_ID: 139664434886512:1059:139664350547912
ENGINE_TRANSACTION_ID: 2569
THREAD_ID: 46
EVENT_ID: 12
OBJECT_SCHEMA: test
OBJECT_NAME: t1
PARTITION_NAME: NULL
SUBPARTITION_NAME: NULL
INDEX_NAME: NULL
OBJECT_INSTANCE_BEGIN: 139664350547912
LOCK_TYPE: TABLE
LOCK_MODE: IX
LOCK_STATUS: GRANTED
LOCK_DATA: NULL
***** 2. row *****
ENGINE: INNODB
ENGINE_LOCK_ID: 139664434886512:2:4:1:139664350544872
ENGINE_TRANSACTION_ID: 2569
THREAD_ID: 46
EVENT_ID: 12
OBJECT_SCHEMA: test
OBJECT_NAME: t1
PARTITION_NAME: NULL
SUBPARTITION_NAME: NULL
INDEX_NAME: GEN_CLUST_INDEX
OBJECT_INSTANCE_BEGIN: 139664350544872
LOCK_TYPE: RECORD
LOCK_MODE: X
LOCK_STATUS: GRANTED
LOCK_DATA: supremum pseudo-record
```

ほとんどのパフォーマンススキーマデータ収集とは異なり、データロック情報を収集するか、データロックテーブルのサイズを制御するためのシステム変数を制御するためのインストゥルメントはありません。パフォーマンススキーマは、サーバーですでに使用可能な情報を収集するため、この情報を生成したり、その収集を制御するパラメータを必要とするメモリまたは CPU のオーバーヘッドはありません。

`data_locks` テーブルを使用すると、負荷が高いときに発生するパフォーマンスの問題の診断に役立ちます。InnoDB については、[セクション15.15.2「InnoDB INFORMATION_SCHEMA トランザクションおよびロック情報」](#)でこのトピックの説明を参照してください。

`data_locks` テーブルには、次のカラムがあります:

- `ENGINE`

ロックを保持または要求したストレージエンジン。

- `ENGINE_LOCK_ID`

ストレージエンジンによって保持または要求されたロックの ID。(ENGINE_LOCK_ID、ENGINE) 値のタプルは一意です。

「ロック ID」形式は内部形式であり、いつでも変更される可能性があります。アプリケーションは、特定の形式のロック ID に依存しないでください。

- `ENGINE_TRANSACTION_ID`

ロックを要求したトランザクションのストレージエンジン内部 ID。これはロックの所有者とみなすことができますが、ロックはまだ保留中であり、実際にはまだ付与されていない可能性があります (`LOCK_STATUS='WAITING'`)。

トランザクションがまだ書き込み操作を実行していない場合 (読み取り専用とみなされます)、カラムにはユーザーが解放しようとしていない内部データが含まれます。それ以外の場合、カラムはトランザクション ID です。

InnoDB の場合、トランザクションの詳細を取得するには、このカラムを `INFORMATION_SCHEMA INNODB_TRX` テーブルの `TRX_ID` カラムと結合します。

- `THREAD_ID`

ロックを作成したセッションのスレッド ID。スレッドの詳細を取得するには、このカラムをパフォーマンススキーマ `threads` テーブルの `THREAD_ID` カラムと結合します。

`THREAD_ID` を `EVENT_ID` とともに使用して、メモリー内にロックデータ構造が作成されたイベントを判別できます。(このイベントは、データ構造を使用して複数のロックが格納されている場合、この特定のロックリクエストが発生する前に発生した可能性があります。)

- `EVENT_ID`

ロックの原因となったパフォーマンススキーマイベント。(`THREAD_ID`、`EVENT_ID`) 値のタプルは、他の「パフォーマンススキーマ」テーブルの親イベントを暗黙的に識別します:

- `events_waits_xxx` テーブルの親待機イベント
- `events_stages_xxx` テーブルの親ステージイベント
- `events_statements_xxx` テーブルの親ステートメントイベント
- `events_transactions_current` テーブルの親トランザクションイベント

親イベントの詳細を取得するには、`THREAD_ID` カラムと `EVENT_ID` カラムを適切な親イベントテーブルの同名のカラムと結合します。 [セクション27.19.2「親イベント情報の取得」](#) を参照してください。

- `OBJECT_SCHEMA`

ロックされたテーブルを含むスキーマ。

- `OBJECT_NAME`

ロックされたテーブルの名前。

- `PARTITION_NAME`

ロックされたパーティションの名前 (存在する場合)。それ以外の場合は `NULL`。

- `SUBPARTITION_NAME`

ロックされたサブパーティションの名前 (存在する場合)。それ以外の場合は `NULL`。

- `INDEX_NAME`

ロックされたインデックスの名前 (存在する場合)。それ以外の場合は `NULL`。

実際には、InnoDB は常にインデックス (`GEN_CLUST_INDEX`) を作成するため、`INDEX_NAME` は InnoDB テーブルに対して `NULL` 以外です。

- `OBJECT_INSTANCE_BEGIN`

ロックのメモリー内のアドレス。

- `LOCK_TYPE`

ロックのタイプ。

この値はストレージエンジンに依存します。InnoDB の場合、許可される値は、行レベルロックの場合は RECORD、テーブルレベルロックの場合は TABLE です。

- LOCK_MODE

ロックのリクエスト方法。

この値はストレージエンジンに依存します。InnoDB の場合、許可される値は S[,GAP], X[,GAP], IS[,GAP], IX[,GAP], AUTO_INC および UNKNOWN です。AUTO_INC および UNKNOWN 以外のロックモードは、ギャップロック (存在する場合) を示します。S, X, IS, IX およびギャップロックの詳細は、[セクション15.7.1「InnoDB ロック」](#) を参照してください。

- LOCK_STATUS

ロックリクエストのステータス。

この値はストレージエンジンに依存します。InnoDB の場合、許可される値は、GRANTED (ロックが保持されている) および WAITING (ロックが待機されている) です。

- LOCK_DATA

ロックに関連付けられているデータ (ある場合)。この値はストレージエンジンに依存します。InnoDB の場合、LOCK_TYPE が RECORD の場合は値が表示され、それ以外の場合は NULL 値が表示されます。ロックされたレコードの主キー値は、主キーインデックスに設定されたロックに対して表示されます。ロックされたレコードのセカンダリインデックス値が表示され、セカンダリインデックスに配置されたロックに主キー値が追加されず。主キーがない場合、LOCK_DATA では、InnoDB クラスティンデックスの使用を制御するルールに従って、選択した一意インデックスのキー値または一意の InnoDB 内部行 ID 番号のいずれかが表示されます ([セクション 15.6.2.1「クラスタインデックスとセカンダリインデックス」](#) を参照)。LOCK_DATA は、supremum 擬似レコードで取得されたロックについて「「supremum 擬似レコード」」を報告します。ロックされたレコードを含むページが、ロックの保持中にディスクに書き込まれたためにバッファプールにない場合、InnoDB はディスクからページをフェッチしません。かわりに、LOCK_DATA は NULL をレポートします。

data_locks テーブルには次のインデックスがあります:

- 主キー (ENGINE_LOCK_ID、ENGINE)
- (ENGINE_TRANSACTION_ID、ENGINE) のインデックス
- (THREAD_ID、EVENT_ID) のインデックス
- (OBJECT_SCHEMA、OBJECT_NAME、PARTITION_NAME、SUBPARTITION_NAME) のインデックス

TRUNCATE TABLE は、data_locks テーブルに対して許可されていません。

27.12.13.2 data_lock_waits テーブル

data_lock_waits テーブルには、data_locks テーブルのどのデータロックリクエストが data_locks テーブルのどのデータロックによってブロックされているかを示す多対多関係が実装されています。data_locks で保持されているロックは、ロックリクエストがブロックされている場合にのみ data_lock_waits に表示されます。

この情報を使用すると、セッション間のデータロックの依存性を理解できます。テーブルには、セッションまたはトランザクションが待機しているロックだけでなく、そのロックを現在保持しているセッションまたはトランザクションが表示されます。

データロック待機情報の例:

```
mysql> SELECT * FROM performance_schema.data_lock_waits\G
***** 1. row *****
ENGINE: INNODB
REQUESTING_ENGINE_LOCK_ID: 140211201964816:2:4:2:140211086465800
REQUESTING_ENGINE_TRANSACTION_ID: 1555
REQUESTING_THREAD_ID: 47
REQUESTING_EVENT_ID: 5
```

```
REQUESTING_OBJECT_INSTANCE_BEGIN: 140211086465800  
BLOCKING_ENGINE_LOCK_ID: 140211201963888:2:4:2:140211086459880  
BLOCKING_ENGINE_TRANSACTION_ID: 1554  
BLOCKING_THREAD_ID: 46  
BLOCKING_EVENT_ID: 12  
BLOCKING_OBJECT_INSTANCE_BEGIN: 140211086459880
```

ほとんどのパフォーマンススキーマデータ収集とは異なり、データロック情報を収集するか、データロックテーブルのサイズを制御するためのシステム変数を制御するためのインストールはありません。パフォーマンススキーマは、サーバーですでに使用可能な情報を収集するため、この情報を生成したり、その収集を制御するパラメータを必要とするメモリーまたは CPU のオーバーヘッドはありません。

[data_lock_waits](#) テーブルを使用すると、負荷が高いときに発生するパフォーマンスの問題の診断に役立ちます。InnoDB については、[セクション15.15.2「InnoDB INFORMATION_SCHEMA トランザクションおよびロック情報」](#)でこのトピックの説明を参照してください。

[data_lock_waits](#) テーブルのカラムは [data_locks](#) テーブルのカラムと似ているため、ここでのカラムの説明は省略されています。カラムの詳細は、[セクション27.12.13.1「data_locks テーブル」](#)を参照してください。

[data_lock_waits](#) テーブルには、次のカラムがあります:

- [ENGINE](#)

ロックを要求したストレージエンジン。

- [REQUESTING_ENGINE_LOCK_ID](#)

ストレージエンジンによって要求されたロックの ID。ロックの詳細を取得するには、このカラムを [data_locks](#) テーブルの [ENGINE_LOCK_ID](#) カラムと結合します。

- [REQUESTING_ENGINE_TRANSACTION_ID](#)

ロックを要求したトランザクションのストレージエンジン内部 ID。

- [REQUESTING_THREAD_ID](#)

ロックをリクエストしたセッションのスレッド ID。

- [REQUESTING_EVENT_ID](#)

ロックを要求したセッションでロック要求を引き起こしたパフォーマンススキーマイベント。

- [REQUESTING_OBJECT_INSTANCE_BEGIN](#)

要求されたロックのメモリー内のアドレス。

- [BLOCKING_ENGINE_LOCK_ID](#)

ブロッキングロックの ID。ロックの詳細を取得するには、このカラムを [data_locks](#) テーブルの [ENGINE_LOCK_ID](#) カラムと結合します。

- [BLOCKING_ENGINE_TRANSACTION_ID](#)

ブロッキングロックを保持するトランザクションのストレージエンジン内部 ID。

- [BLOCKING_THREAD_ID](#)

ブロッキングロックを保持するセッションのスレッド ID。

- [BLOCKING_EVENT_ID](#)

パフォーマンススキーマを保持しているセッションでブロッキングロックの原因となったパフォーマンススキーマイベント。

- [BLOCKING_OBJECT_INSTANCE_BEGIN](#)

ブロッキングロックのメモリー内のアドレス。

`data_lock_waits` テーブルには次のインデックスがあります:

- (REQUESTING_ENGINE_LOCK_ID、ENGINE) のインデックス
- (BLOCKING_ENGINE_LOCK_ID、ENGINE) のインデックス
- (REQUESTING_ENGINE_TRANSACTION_ID、ENGINE) のインデックス
- (BLOCKING_ENGINE_TRANSACTION_ID、ENGINE) のインデックス
- (REQUESTING_THREAD_ID、REQUESTING_EVENT_ID) のインデックス
- (BLOCKING_THREAD_ID、BLOCKING_EVENT_ID) のインデックス

TRUNCATE TABLE は、`data_lock_waits` テーブルに対して許可されていません。

27.12.13.3 metadata_locks テーブル

MySQL では、メタデータロックを使用して、データベースオブジェクトへの同時アクセスを管理し、データの一貫性を確保します。セクション8.11.4「メタデータのロック」を参照してください。メタデータのロックは、テーブルのみでなく、スキーマ、ストアードプログラム (プロシージャ、ファンクション、トリガー、スケジュール済イベント)、テーブルスペース、GET_LOCK() 関数で取得されたユーザーロック (セクション12.15「ロック関数」を参照)、およびセクション5.6.8.1「ロックサービス」で説明されているロックサービスで取得されたロックにも適用されます。

パフォーマンススキーマは、`metadata_locks` テーブルを介してメタデータロック情報を公開します:

- 付与されているロック (現在どのセッションがどのメタデータロックを所有しているかが表示されます)。
- リクエストされたがまだ付与されていないロック (どのセッションがどのメタデータロックを待機しているかを示します)。
- デッドロック検出機能によって強制終了されたロックリクエスト。
- タイムアウトし、リクエストしているセッションロックリクエストが破棄されるのを待機しているロックリクエスト。

この情報を使用すると、セッション間のメタデータロックの依存性を理解できます。セッションが待機しているロックだけでなく、そのロックを現在保持しているセッションも表示できます。

`metadata_locks` テーブルは読み取り専用であり、更新できません。デフォルトでは自動サイズ設定されています。テーブルサイズを構成するには、サーバーの起動時に `performance_schema_max_metadata_locks` システム変数を設定します。

メタデータロックインストールメンターションでは、デフォルトで有効になっている `wait/lock/metadata/sql/mdl` インストールメントが使用されます。

サーバー起動時のメタデータロックインストールメンターションの状態を制御するには、`my.cnf` ファイルで次のような行を使用します:

- 有効化:

```
[mysqld]
performance-schema-instrument='wait/lock/metadata/sql/mdl=ON'
```

- 無効化:

```
[mysqld]
performance-schema-instrument='wait/lock/metadata/sql/mdl=OFF'
```

実行時にメタデータロックインストールメンターションの状態を制御するには、`setup_instruments` テーブルを更新します:

- 有効化:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'YES', TIMED = 'YES'
WHERE NAME = 'wait/lock/metadata/sql/mdl';
```

- 無効化:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'NO', TIMED = 'NO'
WHERE NAME = 'wait/lock/metadata/sql/mdl';
```

パフォーマンススキーマは、**LOCK_STATUS** カラムを使用して各ロックのステータスを示す **metadata_locks** テーブルの内容を次のように保守します:

- メタデータロックがリクエストされてすぐに取得されると、ステータスが **GRANTED** の行が挿入されます。
- メタデータロックがリクエストされ、すぐに取得されない場合、ステータスが **PENDING** の行が挿入されます。
- 以前にリクエストされたメタデータロックが付与されると、その行ステータスは **GRANTED** に更新されます。
- メタデータロックが解放されると、その行は削除されます。
- デッドロック検出でデッドロック (**ER_LOCK_DEADLOCK**) を解除するために保留中のロックリクエストが取り消されると、その行ステータスが **PENDING** から **VICTIM** に更新されます。
- 保留中のロックリクエスト (**ER_LOCK_WAIT_TIMEOUT**) がタイムアウトすると、その行ステータスが **PENDING** から **TIMEOUT** に更新されます。
- ロックまたは保留中のロックリクエストが強制終了されると、その行ステータスが **GRANTED** または **PENDING** から **KILLED** に更新されます。
- VICTIM**、**TIMEOUT** および **KILLED** のステータス値は簡潔で、ロック行が削除されようとしていることを示します。
- PRE_ACQUIRE_NOTIFY** および **POST_RELEASE_NOTIFY** のステータス値は簡潔であり、ロック取得操作の開始中またはロック解放操作の終了中に、メタデータロックサブシステムが関連するストレージエンジンに通知していることを示します。

metadata_locks テーブルには、次のカラムがあります:

- OBJECT_TYPE**

メタデータロックサブシステムで使用されるロックのタイプ。値は、**GLOBAL**、**SCHEMA**、**TABLE**、**FUNCTION**、**PROCEDURE**、**TRIGGER** (現在未使用)、**EVENT**、**COMMIT**、**USER LEVEL LOCK**、**TABLESPACE** または **LOCKING SERVICE** のいずれかです。

USER LEVEL LOCK の値は、**GET_LOCK()** で取得されたロックを示します。**LOCKING SERVICE** の値は、[セクション5.6.8.1「ロックサービス」](#) で説明されているロックサービスで取得されたロックを示します。

- OBJECT_SCHEMA**

オブジェクトを格納するスキーマ。

- OBJECT_NAME**

インストールメントされたオブジェクトの名前。

- OBJECT_INSTANCE_BEGIN**

インストールメントされたオブジェクトのメモリー内のアドレス。

- LOCK_TYPE**

メタデータロックサブシステムからのロックタイプ。値は、**INTENTION_EXCLUSIVE**、**SHARED**、**SHARED_HIGH_PRIO**、**SHARED_READ**、**SHARED_WRITE**、**SHARED_UPGRADABLE**、**SHARED_NO_WRITE**、**SHARED_NO_READ_WRITE** または **EXCLUSIVE** のいずれかです。

- [LOCK_DURATION](#)

メタデータロックサブシステムからのロック期間。値は、[STATEMENT](#)、[TRANSACTION](#) または [EXPLICIT](#) のいずれかです。 [STATEMENT](#) および [TRANSACTION](#) の値は、それぞれステートメントまたはトランザクションの終了時に暗黙的に解放されるロックを示します。 [EXPLICIT](#) の値は、残りのステートメントまたはトランザクションが終了し、[FLUSH TABLES WITH READ LOCK](#) で取得されたグローバルロックなどの明示的なアクションによって解放されるロックを示します。

- [LOCK_STATUS](#)

メタデータロックサブシステムからのロックステータス。値は、[PENDING](#)、[GRANTED](#)、[VICTIM](#)、[TIMEOUT](#)、[KILLED](#)、[PRE_ACQUIRE_NOTIFY](#) または [POST_RELEASE_NOTIFY](#) のいずれかです。 パフォーマンススキーマは、前述のようにこれらの値を割り当てます。

- [SOURCE](#)

イベントを生成した、インストゥルメントされたコードを格納するソースファイルの名前と、インストゥルメンテーションが行われたファイルの行番号。 これにより、ソースをチェックして、コードに含まれるものを正確に判断することができます。

- [OWNER_THREAD_ID](#)

メタデータロックをリクエストしているスレッド。

- [OWNER_EVENT_ID](#)

メタデータロックをリクエストしているイベント。

[metadata_locks](#) テーブルには次のインデックスがあります:

- 主キー ([OBJECT_INSTANCE_BEGIN](#))
- ([OBJECT_TYPE](#), [OBJECT_SCHEMA](#), [OBJECT_NAME](#)) のインデックス
- ([OWNER_THREAD_ID](#), [OWNER_EVENT_ID](#)) のインデックス

[TRUNCATE TABLE](#) は、[metadata_locks](#) テーブルに対して許可されていません。

27.12.13.4 [table_handles](#) テーブル

パフォーマンススキーマは、[table_handles](#) テーブルを介してテーブルロック情報を公開し、開いているテーブルハンドルごとに現在有効なテーブルロックを表示します。 [table_handles](#) では、テーブルロックインストゥルメンテーションによって記録される内容がレポートされます。 この情報には、サーバーが開いているテーブルハンドル、ロック方法、およびセッションが表示されます。

[table_handles](#) テーブルは読み取り専用であり、更新できません。 デフォルトでは自動サイズ設定されています。 テーブルサイズを構成するには、サーバーの起動時に [performance_schema_max_table_handles](#) システム変数を設定します。

テーブルロックのインストゥルメンテーションでは、デフォルトで有効になっている [wait/lock/table/sql/handler](#) インストゥルメントが使用されます。

サーバー起動時のテーブルロックのインストゥルメンテーション状態を制御するには、[my.cnf](#) ファイルで次のような行を使用します:

- 有効化:

```
[mysqld]
performance-schema-instrument='wait/lock/table/sql/handler=ON'
```

- 無効化:

```
[mysqld]
performance-schema-instrument='wait/lock/table/sql/handler=OFF'
```

実行時にテーブルロックのインストールメンテーション状態を制御するには、`setup_instruments` テーブルを更新します:

- 有効化:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'YES', TIMED = 'YES'
WHERE NAME = 'wait/lock/table/sql/handler';
```

- 無効化:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'NO', TIMED = 'NO'
WHERE NAME = 'wait/lock/table/sql/handler';
```

`table_handles` テーブルには、次のカラムがあります:

- `OBJECT_TYPE`

テーブルハンドルによってオープンされたテーブル。

- `OBJECT_SCHEMA`

オブジェクトを格納するスキーマ。

- `OBJECT_NAME`

インストールされたオブジェクトの名前。

- `OBJECT_INSTANCE_BEGIN`

メモリー内のテーブルハンドルアドレス。

- `OWNER_THREAD_ID`

テーブルハンドルを所有するスレッド。

- `OWNER_EVENT_ID`

テーブルハンドルがオープンされる原因となったイベント。

- `INTERNAL_LOCK`

SQL レベルで使用されるテーブルロック。値は、`READ`、`READ WITH SHARED LOCKS`、`READ HIGH PRIORITY`、`READ NO INSERT`、`WRITE ALLOW WRITE`、`WRITE CONCURRENT INSERT`、`WRITE LOW PRIORITY` または `WRITE` のいずれかです。これらのロックタイプについては、`include/thr_lock.h` ソースファイルを参照してください。

- `EXTERNAL_LOCK`

ストレージエンジンレベルで使用されるテーブルロック。値は、`READ EXTERNAL` または `WRITE EXTERNAL` のいずれかです。

`table_handles` テーブルには次のインデックスがあります:

- 主キー (`OBJECT_INSTANCE_BEGIN`)
- (`OBJECT_TYPE`, `OBJECT_SCHEMA`, `OBJECT_NAME`) のインデックス
- (`OWNER_THREAD_ID`, `OWNER_EVENT_ID`) のインデックス

`TRUNCATE TABLE` は、`table_handles` テーブルに対して許可されていません。

27.12.14 パフォーマンススキーマシステム変数テーブル

MySQL サーバーは、構成方法を示す多くのシステム変数を保持します ([セクション5.1.8「サーバーシステム変数」](#)を参照)。システム変数情報は、次の「パフォーマンススキーマ」テーブルで使用できます:

- [global_variables](#): グローバルシステム変数。グローバル値のみが必要なアプリケーションでは、このテーブルを使用する必要があります。
- [session_variables](#): 現在のセッションのシステム変数。独自のセッションのすべてのシステム変数値を必要とするアプリケーションでは、このテーブルを使用する必要があります。これには、セッションのセッション変数と、対応するセッションがないグローバル変数の値が含まれます。
- [variables_by_thread](#): 各アクティブセッションのセッションシステム変数。特定のセッションのセッション変数値を知るアプリケーションでは、このテーブルを使用する必要があります。これには、スレッド ID で識別されるセッション変数のみが含まれます。
- [persisted_variables](#): 永続化されたグローバルシステム変数設定を格納する `mysqld-auto.cnf` ファイルへの SQL インタフェースを提供します。 [セクション 27.12.14.1 「パフォーマンススキーマ persisted_variables テーブル」](#) を参照してください。
- [variables_info](#): システム変数ごとに、最後に設定されたソースとその値の範囲を示します。 [セクション 27.12.14.2 「パフォーマンススキーマ variables_info テーブル」](#) を参照してください。

セッション変数テーブル ([session_variables](#)、[variables_by_thread](#)) には、アクティブセッションの情報のみが含まれ、終了したセッションは含まれません。

[global_variables](#) テーブルと [session_variables](#) テーブルには、次のカラムがあります:

- [VARIABLE_NAME](#)
システム変数名。
- [VARIABLE_VALUE](#)
システム変数の値。 [global_variables](#) の場合、このカラムにはグローバル値が含まれます。 [session_variables](#) の場合、このカラムには現在のセッションで有効な変数値が含まれます。

[global_variables](#) テーブルと [session_variables](#) テーブルには、次のインデックスがあります:

- 主キー ([VARIABLE_NAME](#))

[variables_by_thread](#) テーブルには、次のカラムがあります:

- [THREAD_ID](#)
システム変数が定義されているセッションのスレッド識別子。
- [VARIABLE_NAME](#)
システム変数名。
- [VARIABLE_VALUE](#)
[THREAD_ID](#) カラムで指定されたセッションのセッション変数値。

[variables_by_thread](#) テーブルには次のインデックスがあります:

- 主キー ([THREAD_ID](#)、[VARIABLE_NAME](#))

[variables_by_thread](#) テーブルには、フォアグラウンドスレッドに関するシステム変数情報のみが含まれます。すべてのスレッドがパフォーマンススキーマによって計測されるわけではない場合、このテーブルにはいくつかの行がありません。この場合、[Performance_schema_thread_instances_lost](#) ステータス変数はゼロより大きくなります。

`TRUNCATE TABLE` は、パフォーマンススキーマシステム変数テーブルではサポートされていません。

27.12.14.1 パフォーマンススキーマ `persisted_variables` テーブル

[persisted_variables](#) テーブルは、永続化されたグローバルシステム変数設定を格納する `mysqld-auto.cnf` ファイルへの SQL インタフェースを提供し、実行時に `SELECT` ステートメントを使用してファイルの内容を検査できるように

します。変数は、`SET PERSIST` または `PERSIST_ONLY` ステートメントを使用して永続化されます。[セクション 13.7.6.1「変数代入の SET 構文」](#) を参照してください。このテーブルには、ファイル内の各永続システム変数の行が含まれています。永続化されていない変数はテーブルに表示されません。

永続システム変数の詳細は、[セクション 13.7.6.1「変数代入の SET 構文」](#) を参照してください。

`mysqld-auto.cnf` は次のようになります (少し再フォーマットされています):

```
{
  "Version": 1,
  "mysql_server": {
    "max_connections": {
      "Value": "1000",
      "Metadata": {
        "Timestamp": 1.519921706e+15,
        "User": "root",
        "Host": "localhost"
      }
    }
  },
  "autocommit": {
    "Value": "ON",
    "Metadata": {
      "Timestamp": 1.519921707e+15,
      "User": "root",
      "Host": "localhost"
    }
  }
}
```

その後、`persisted_variables` には次の内容が含まれます:

```
mysql> SELECT * FROM performance_schema.persisted_variables;
+-----+-----+
| VARIABLE_NAME | VARIABLE_VALUE |
+-----+-----+
| autocommit   | ON             |
| max_connections | 1000          |
+-----+-----+
```

`persisted_variables` テーブルには、次のカラムがあります:

- `VARIABLE_NAME`
mysqld-auto.cnf にリストされている変数名。
- `VARIABLE_VALUE`
mysqld-auto.cnf の変数にリストされている値。

`persisted_variables` には、次のインデックスがあります:

- 主キー (`VARIABLE_NAME`)

`TRUNCATE TABLE` は、`persisted_variables` テーブルに対して許可されていません。

27.12.14.2 パフォーマンススキーマ variables_info テーブル

`variables_info` テーブルには、システム変数ごとに、最後に設定されたソースとその値の範囲が表示されます。

`variables_info` テーブルには、次のカラムがあります:

- `VARIABLE_NAME`
変数名。
- `VARIABLE_SOURCE`
変数が最後に設定されたソース:

- **COMMAND_LINE**

コマンドラインで変数が設定されました。

- **COMPILED**

変数には、コンパイルされたデフォルト値があります。 **COMPILED** は、他の方法で設定されない変数に使用される値です。

- **DYNAMIC**

変数は実行時に設定されました。これには、 **init_file** システム変数を使用して指定されたファイル内に設定された変数が含まれます。

- **EXPLICIT**

変数は、 **--defaults-file** オプションで指定されたオプションファイルから設定されました。

- **EXTRA**

変数は、 **--defaults-extra-file** オプションで指定されたオプションファイルから設定されました。

- **GLOBAL**

変数がグローバルオプションファイルから設定されました。これには、 **EXPLICIT**, **EXTRA**, **LOGIN**, **PERSISTED**, **SERVER** または **USER** でカバーされないオプションファイルが含まれます。

- **LOGIN**

変数は、ユーザー固有のログインパスファイル (**~/mylogin.cnf**) から設定されました。

- **PERSISTED**

変数がサーバー固有の **mysqld-auto.cnf** オプションファイルから設定されました。 **persisted_globals_load** を無効にしてサーバーを起動した場合、この値は行に含まれません。

- **SERVER**

変数がサーバー固有の **\$MYSQL_HOME/my.cnf** オプションファイルから設定されました。 **MYSQL_HOME** の設定方法の詳細は、 [セクション4.2.2.2「オプションファイルの使用」](#) を参照してください。

- **USER**

変数がユーザー固有の **~/my.cnf** オプションファイルから設定されました。

- **VARIABLE_PATH**

変数がオプションファイルから設定された場合、 **VARIABLE_PATH** はそのファイルのパス名です。それ以外の場合、値は空の文字列です。

- **MIN_VALUE, MAX_VALUE**

変数に許可される最小値と最大値。両方とも、そのような値を持たない変数(つまり、数値ではない変数)の場合は0です。

- **SET_TIME**

変数が最後に設定された時刻。デフォルトは、起動時にサーバーがグローバルシステム変数を初期化した時刻です。

- **SET_USER, SET_HOST**

変数を最後に設定したクライアントユーザーのユーザー名とホスト名。クライアントが **'user17'@'%example.com** アカウントを使用してホスト **host34.example.com** から **user17** として接続する場合、 **SET_USER** および

`SET_HOST` はそれぞれ `user17` および `host34.example.com` です。プロキシユーザー接続の場合、これらの値は権限チェックが実行されるプロキシユーザーではなく、外部 (プロキシ) ユーザーに対応します。各カラムのデフォルトは空の文字列で、サーバーの起動後に変数が設定されていないことを示します。

`variables_info` テーブルには次のインデックスがあります:

- なし

`TRUNCATE TABLE` は、`variables_info` テーブルに対して許可されていません。

`DYNAMIC` 以外の `VARIABLE_SOURCE` 値を持つ変数が実行時に設定された場合、`VARIABLE_SOURCE` は `DYNAMIC` になり、`VARIABLE_PATH` は空の文字列になります。

セッション値のみを持つシステム変数 (`debug_sync` など) は、起動時または永続化時に設定できません。セッション専用システム変数の場合、`VARIABLE_SOURCE` には `COMPILED` または `DYNAMIC` のみを指定できます。

システム変数に予期しない `VARIABLE_SOURCE` 値が含まれている場合は、サーバーの起動方法を検討してください。たとえば、`mysqld_safe` はオプションファイルを読み取り、`mysqld` の起動に使用するコマンドラインの一部として検出された特定のオプションを渡します。したがって、オプションファイルに設定した一部のシステム変数は、予想されるおりに `GLOBAL` または `SERVER` としてではなく、`COMMAND_LINE` として `variables_info` に表示される場合があります。

`variables_info` テーブルを使用するいくつかのサンプルクエリーでは、次のよう出力されます:

- コマンドラインで設定された変数を表示します:

```
mysql> SELECT VARIABLE_NAME
        FROM performance_schema.variables_info
        WHERE VARIABLE_SOURCE = 'COMMAND_LINE'
        ORDER BY VARIABLE_NAME;
+-----+
| VARIABLE_NAME |
+-----+
| basedir      |
| datadir     |
| log_error   |
| pid_file    |
| plugin_dir  |
| port        |
+-----+
```

- 永続ストレージから設定された変数を表示します:

```
mysql> SELECT VARIABLE_NAME
        FROM performance_schema.variables_info
        WHERE VARIABLE_SOURCE = 'PERSISTED'
        ORDER BY VARIABLE_NAME;
+-----+
| VARIABLE_NAME |
+-----+
| event_scheduler |
| max_connections |
| validate_password.policy |
+-----+
```

- `variables_info` を `global_variables` テーブルと結合して、永続変数の現在の値とその値の範囲を表示します:

```
mysql> SELECT
        VI.VARIABLE_NAME, GV.VARIABLE_VALUE,
        VI.MIN_VALUE, VI.MAX_VALUE
        FROM performance_schema.variables_info AS VI
        INNER JOIN performance_schema.global_variables AS GV
        USING(VARIABLE_NAME)
        WHERE VI.VARIABLE_SOURCE = 'PERSISTED'
        ORDER BY VARIABLE_NAME;
+-----+-----+-----+-----+
| VARIABLE_NAME | VARIABLE_VALUE | MIN_VALUE | MAX_VALUE |
+-----+-----+-----+-----+
| event_scheduler | ON             | 0         | 0         |
+-----+-----+-----+-----+
```

max_connections	200	1	100000	
validate_password.policy	STRONG	0	0	

27.12.15 パフォーマンススキーマのステータス変数のテーブル

MySQL サーバーは、その操作に関する情報を提供する多くのステータス変数を保持します (セクション5.1.10「サーバスステータス変数」を参照)。ステータス変数情報は、次の「パフォーマンススキーマ」テーブルで使用できます:

- **global_status**: グローバルステータス変数。グローバル値のみが必要なアプリケーションでは、このテーブルを使用する必要があります。
- **session_status**: 現在のセッションのステータス変数。独自のセッションのすべてのステータス変数値を必要とするアプリケーションでは、このテーブルを使用する必要があります。これには、セッションのセッション変数と、対応するセッションがないグローバル変数の値が含まれます。
- **status_by_thread**: 各アクティブセッションのセッションステータス変数。特定のセッションのセッション変数値を知るアプリケーションでは、このテーブルを使用する必要があります。これには、スレッド ID で識別されるセッション変数のみが含まれます。

アカウント、ホスト名およびユーザー名ごとに集計されたステータス変数情報を提供するサマリーテーブルもあります。セクション27.12.18.12「ステータス変数サマリーテーブル」を参照してください。

セッション変数テーブル (**session_status**、**status_by_thread**) には、アクティブセッションの情報のみが含まれ、終了したセッションは含まれません。

パフォーマンススキーマは、**threads** テーブル内の **INSTRUMENTED** 値が **YES** であるスレッドについてのみ、グローバルステータス変数の統計を収集します。セッションステータス変数の統計は、**INSTRUMENTED** の値に関係なく常に収集されます。

パフォーマンススキーマは、ステータス変数テーブル内の **Com_xxx** ステータス変数の統計情報を収集しません。グローバルステートメントの実行数およびセッションごとのステートメントの実行数を取得するには、それぞれ **events_statements_summary_global_by_event_name** テーブルおよび **events_statements_summary_by_thread_by_event_name** テーブルを使用します。例:

```
SELECT EVENT_NAME, COUNT_STAR
FROM performance_schema.events_statements_summary_global_by_event_name
WHERE EVENT_NAME LIKE 'statement/sql/%';
```

global_status テーブルと **session_status** テーブルには、次のカラムがあります:

- **VARIABLE_NAME**
ステータス変数名。
- **VARIABLE_VALUE**
ステータス変数の値。 **global_status** の場合、このカラムにはグローバル値が含まれます。 **session_status** の場合、このカラムには現在のセッションの変数値が含まれます。

global_status テーブルと **session_status** テーブルには、次のインデックスがあります:

- 主キー (**VARIABLE_NAME**)

status_by_thread テーブルには、各アクティブスレッドのステータスが含まれます。これには次のカラムがあります。

- **THREAD_ID**
ステータス変数が定義されているセッションのスレッド識別子。
- **VARIABLE_NAME**
ステータス変数名。

- [VARIABLE_VALUE](#)

[THREAD_ID](#) カラムで指定されたセッションのセッション変数値。

[status_by_thread](#) テーブルには次のインデックスがあります:

- 主キー ([THREAD_ID](#)、[VARIABLE_NAME](#))

[status_by_thread](#) テーブルには、フォアグラウンドスレッドに関するステータス変数情報のみが含まれます。[performance_schema_max_thread_instances](#) システム変数が自動スケーリングされず (値-1 で示される)、インストールされるスレッドオブジェクトの最大許容数がバックグラウンドスレッドの数を超えていない場合、テーブルは空になります。

パフォーマンススキーマは、次のようなステータス変数テーブルの [TRUNCATE TABLE](#) をサポートします:

- [global_status](#): スレッド、アカウント、ホストおよびユーザーのステータスをリセットします。サーバーがリセットしないグローバルステータス変数をリセットします。
- [session_status](#): サポートされません。
- [status_by_thread](#): すべてのスレッドのステータスをグローバルステータスおよびアカウントステータスに集計し、スレッドステータスをリセットします。アカウント統計が収集されない場合、ホストおよびユーザーのステータスが収集されると、セッションステータスがホストおよびユーザーのステータスに追加されます。

[performance_schema_accounts_size](#)、[performance_schema_hosts_size](#) および [performance_schema_users_size](#) システム変数がそれぞれ 0 に設定されている場合、アカウント、ホストおよびユーザーの統計は収集されません。

[FLUSH STATUS](#) は、すべてのアクティブセッションのセッションステータスをグローバルステータス変数に追加し、すべてのアクティブセッションのステータスをリセットし、切断されたセッションから集計されたアカウント、ホストおよびユーザーステータス値をリセットします。

27.12.16 パフォーマンススキーマスレッドプールテーブル

注記

ここで説明する「パフォーマンススキーマ」テーブルは、MySQL 8.0.14 の時点で使用可能です。MySQL 8.0.14 より前は、かわりに対応する [INFORMATION_SCHEMA](#) テーブルを使用してください。[セクション26.52 「INFORMATION_SCHEMA スレッドプールテーブル」](#) を参照してください。

次のセクションでは、スレッドプールプラグインに関連付けられた「パフォーマンススキーマ」テーブルについて説明します ([セクション5.6.3 「MySQL Enterprise Thread Pool」](#) を参照)。スレッドプール操作に関する情報を示します。

- [tp_thread_group_state](#): スレッドプールのスレッドグループの状態に関する情報
- [tp_thread_group_stats](#): スレッドグループ統計
- [tp_thread_state](#): スレッドプールスレッドの状態に関する情報

これらのテーブル内の行は、特定時点のスナップショットを表します。[tp_thread_state](#) の場合、スレッドグループのすべての行は時間内のスナップショットを構成します。そのため、MySQL Server では、スナップショットの生成中にスレッドグループの相互排他ロックを保持します。ただし、[tp_thread_state](#) に対するステートメントが MySQL サーバー全体をブロックしないように、すべてのスレッドグループの mutex を同時に保持するわけではありません。

パフォーマンススキーマスレッドプールテーブルはスレッドプールプラグインによって実装され、そのプラグインがロードおよびアンロードされるときにロードおよびアンロードされます ([セクション5.6.3.2 「スレッドプールのインストール」](#) を参照)。テーブルの特別な構成ステップは必要ありません。ただし、これらのテーブルは、有効にするスレッドプールプラグインによって異なります。スレッドプールプラグインがロードされているが無効になっている場合、テーブルは作成されません。

27.12.16.1 [tp_thread_group_state](#) テーブル

注記

ここで説明する「パフォーマンススキーマ」テーブルは、MySQL 8.0.14 の時点で使用可能です。MySQL 8.0.14 より前は、かわりに対応する [INFORMATION_SCHEMA](#) テーブルを使用してください。 [セクション26.52.1「INFORMATION_SCHEMA TP_THREAD_GROUP_STATE テーブル」](#) を参照してください。

`tp_thread_group_state` テーブルには、スレッドプール内のスレッドグループごとに 1 つの行があります。それぞれの行には、グループの現在の状態に関する情報が表示されます。

`tp_thread_group_state` テーブルには、次のカラムがあります:

- `TP_GROUP_ID`

スレッドグループ ID です。これはテーブル内の一意のキーです。

- `CONSUMER_THREADS`

コンシューマスレッドの数です。アクティブなスレッドが停止またはブロックされると、実行開始可能なスレッドが最大 1 つ存在します。

- `RESERVE_THREADS`

予約状態のスレッドの数です。つまり、新しいスレッドをスリープ解除する必要があり、コンシューマスレッドが存在しなくなるまで、それらは起動されません。これは、通常の操作で必要になる数よりも多くのスレッドをスレッドグループが作成したときに、ほとんどのスレッドが最終的に達する状態です。多くの場合、スレッドグループは短い間追加スレッドを必要とし、その後しばらくの間はふたたび必要とすることはありません。この場合には予約状態になり、ふたたび必要とされるまでその状態のままです。ある程度のメモリーリソースをべつに使用しますが、追加のコンピューティングリソースは必要ありません。

- `CONNECT_THREAD_COUNT`

接続の初期化および認証の処理を処理中または待機中のスレッドの数。スレッドグループごとに最大 4 つの接続スレッドを使用できます。これらのスレッドは、非アクティブな期間が経過すると期限切れになります。

- `CONNECTION_COUNT`

このスレッドグループを使用した接続の数です。

- `QUEUED_QUERIES`

優先度の高いキューで待機しているステートメントの数です。

- `QUEUED_TRANSACTIONS`

優先度の低いキューで待機しているステートメントの数です。これらは、開始されていないトランザクションの初期ステートメントなので、キューに入れられたトランザクションも表します。

- `STALL_LIMIT`

スレッドグループの `thread_pool_stall_limit` システム変数の値。これはすべてのスレッドグループで同じ値です。

- `PRIO_KICKUP_TIMER`

スレッドグループの `thread_pool_prio_kickup_timer` システム変数の値。これはすべてのスレッドグループで同じ値です。

- `ALGORITHM`

スレッドグループの `thread_pool_algorithm` システム変数の値。これはすべてのスレッドグループで同じ値です。

- `THREAD_COUNT`

このスレッドグループの一部としてスレッドプール内で開始されたスレッドの数です。

- [ACTIVE_THREAD_COUNT](#)

ステートメントの実行中にアクティブなスレッドの数。

- [STALLED_THREAD_COUNT](#)

停止したスレッドグループ内のステートメントの数です。停止したステートメントが実行している可能性はありますが、スレッドプールから見ると停止して、進行していません。長時間実行しているステートメントはすぐにこのカテゴリで終了します。

- [WAITING_THREAD_NUMBER](#)

スレッドグループ内のステートメントのポーリングを処理するスレッドがある場合、これにより、このスレッドグループ内のスレッド番号が指定されます。このスレッドがステートメントを実行している可能性があります。

- [OLDEST_QUEUED](#)

もっとも古いキューに入れられたステートメントが実行待機した時間です (ミリ秒単位)。

- [MAX_THREAD_IDS_IN_GROUP](#)

グループ内のスレッドの最大スレッド ID です。これは、[tp_thread_state](#) テーブルから選択した場合のスレッドの [MAX\(TP_THREAD_NUMBER\)](#) と同じです。つまり、次の 2 つのクエリーは同等です。

```
SELECT TP_GROUP_ID, MAX_THREAD_IDS_IN_GROUP
FROM tp_thread_group_state;
```

```
SELECT TP_GROUP_ID, MAX(TP_THREAD_NUMBER)
FROM tp_thread_state GROUP BY TP_GROUP_ID;
```

[tp_thread_group_state](#) テーブルには次のインデックスがあります:

- [\(TP_GROUP_ID\)](#) の一意インデックス

[TRUNCATE TABLE](#) は、[tp_thread_group_state](#) テーブルに対して許可されていません。

27.12.16.2 [tp_thread_group_stats](#) テーブル

注記

ここで説明する「パフォーマンススキーマ」テーブルは、MySQL 8.0.14 の時点で使用可能です。MySQL 8.0.14 より前は、かわりに対応する [INFORMATION_SCHEMA](#) テーブルを使用してください。[セクション 26.52.2 「INFORMATION_SCHEMA TP_THREAD_GROUP_STATS テーブル」](#) を参照してください。

[tp_thread_group_stats](#) テーブルには、スレッドグループごとの統計がレポートされます。グループごとに 1 行があります。

[tp_thread_group_stats](#) テーブルには、次のカラムがあります:

- [TP_GROUP_ID](#)

スレッドグループ ID です。これはテーブル内の一意のキーです。

- [CONNECTIONS_STARTED](#)

開始した接続の数です。

- [CONNECTIONS_CLOSED](#)

終了した接続の数です。

- [QUERIES_EXECUTED](#)

実行したステートメントの数です。この数は、ステートメントが実行を終了したときではなく、開始したときに増えます。

- [QUERIES_QUEUED](#)

実行を待ってキューに入れられた、受け取ったステートメントの数です。これは、スレッドグループがキューに入れることなく即座に実行を開始できたステートメントはカウントされません。即座に実行を開始できるのは、[セクション5.6.3.3「スレッドプール操作」](#)に記した条件に該当する場合です。

- [THREADS_STARTED](#)

開始したスレッドの数です。

- [PRIO_KICKUPS](#)

[thread_pool_prio_kickup_timer](#) システム変数の値に基づいて、優先度の低いキューから優先度の高いキューに移動したステートメントの数です。この数が急速に増えた場合、変数の値を増やしてください。急速に増えるカウンタは、トランザクションが非常に早くから開始しないようにする優先度システムが機能していないことを意味します。[InnoDB](#) の場合、これは、同時トランザクションが多いためにパフォーマンスが低下している可能性が高くなっています。

- [STALLED_QUERIES_EXECUTED](#)

[thread_pool_stall_limit](#) システム変数の値より長く実行されたために停止として定義されたステートメントの数。

- [BECOME_CONSUMER_THREAD](#)

コンシューマスレッドロールがスレッドに割り当てられた回数です。

- [BECOME_RESERVE_THREAD](#)

予約スレッドロールがスレッドに割り当てられた回数です。

- [BECOME_WAITING_THREAD](#)

待機スレッドロールがスレッドに割り当てられた回数です。ステートメントがキューに入れられると、これは、通常の操作であっても、非常に頻繁に起こります。したがって、ステートメントがキューに入れられたシステムの負荷が高い場合には、この値が急速に増加しても正常です。

- [WAKE_THREAD_STALL_CHECKER](#)

複数のステートメントをできるだけ処理したり、待機スレッドロールに対処したりするために、スレッドのウェイクアップまたは作成を、停止チェックスレッドで決定した回数です。

- [SLEEP_WAITS](#)

[THD_WAIT_SLEEP](#) 待機の数です。これらは、スレッドがスリープ状態になるとき (たとえば、[SLEEP\(\)](#) 関数を呼び出すことによって) に発生します。

- [DISK_IO_WAITS](#)

[THD_WAIT_DISKIO](#) 待機の数です。ファイルシステムキャッシュにヒットしない可能性のあるディスク I/O をスレッドが実行すると発生します。ファイルに対する通常の読み取りおよび書き込みの場合ではなく、バッファプールがディスクに対してデータを読み取りおよび書き込むときに、このような待機が起こります。

- [ROW_LOCK_WAITS](#)

別のトランザクションによる行ロックの解放を待っている [THD_WAIT_ROW_LOCK](#) 待機の数です。

- [GLOBAL_LOCK_WAITS](#)

グローバルロックの解放を待っている [THD_WAIT_GLOBAL_LOCK](#) 待機の数です。

- [META_DATA_LOCK_WAITS](#)

メタデータロックの解放を待っている [THD_WAIT_META_DATA_LOCK](#) 待機の数です。

- [TABLE_LOCK_WAITS](#)

ステートメントがアクセスする必要のあるテーブルのロック解除を待っている `THD_WAIT_TABLE_LOCK` 待機の数です。

- `USER_LOCK_WAITS`

ユーザースレッドで構築された固有のロックを待っている `THD_WAIT_USER_LOCK` 待機の数です。

- `BINLOG_WAITS`

バイナリログの解放を待っている `THD_WAIT_BINLOG_WAITS` 待機の数です。

- `GROUP_COMMIT_WAITS`

`THD_WAIT_GROUP_COMMIT` 待機の数です。その他のパーティーがトランザクションの担当分を完了するまで、グループコミットが待機する必要があるときに発生します。

- `FSYNC_WAITS`

ファイル同期操作を待っている `THD_WAIT_SYNC` 待機の数です。

`tp_thread_group_stats` テーブルには次のインデックスがあります:

- `(TP_GROUP_ID)` の一意インデックス

`TRUNCATE TABLE` は、`tp_thread_group_stats` テーブルに対して許可されていません。

27.12.16.3 `tp_thread_state` テーブル

注記

ここで説明する「パフォーマンススキーマ」テーブルは、MySQL 8.0.14 の時点で使用可能です。MySQL 8.0.14 より前は、かわりに対応する `INFORMATION_SCHEMA` テーブルを使用してください。[セクション26.52.3「INFORMATION_SCHEMA TP_THREAD_STATE テーブル」](#) を参照してください。

`tp_thread_state` テーブルには、接続を処理するためにスレッドプールによって作成されたスレッドごとに 1 つの行があります。

`tp_thread_state` テーブルには、次のカラムがあります:

- `TP_GROUP_ID`

スレッドグループ ID です。

- `TP_THREAD_NUMBER`

スレッドグループ内のスレッドの ID です。 `TP_GROUP_ID` と `TP_THREAD_NUMBER` の組み合わせが、テーブル内での一意のキーになります。

- `PROCESS_COUNT`

このスレッドを使用しているステートメントが現在実行している 10 ミリ秒間隔です。0 は実行しているステートメントがないことを示し、1 はステートメントが最初の 10 ミリ秒に存在していることを示す、というようになります。

- `WAIT_TYPE`

スレッドの待機のタイプです。 `NULL` はスレッドがブロックされていることを示します。それ以外の場合、スレッドは `thd_wait_begin()` の呼び出しによってブロックされ、値は待機のタイプを指定します。 `tp_thread_group_stats` テーブルの `xxx_WAIT` カラムには、待機タイプごとにカウントが累積されます。

`WAIT_TYPE` 値は、次の表に示すように、待機のタイプを記述した文字列です。

表 27.3 tp_thread_state テーブルの WAIT_TYPE 値

待機タイプ	意味
THD_WAIT_SLEEP	スリープの待機
THD_WAIT_DISKIO	ディスク IO の待機
THD_WAIT_ROW_LOCK	行ロックの待機
THD_WAIT_GLOBAL_LOCK	グローバルロックの待機
THD_WAIT_META_DATA_LOCK	メタデータロックの待機
THD_WAIT_TABLE_LOCK	テーブルロックの待機
THD_WAIT_USER_LOCK	ユーザーロックの待機
THD_WAIT_BINLOG	binlog の待機
THD_WAIT_GROUP_COMMIT	グループコミットの待機
THD_WAIT_SYNC	fsync の待機

tp_thread_state テーブルには次のインデックスがあります:

- (TP_GROUP_ID、TP_THREAD_NUMBER) の一意インデックス

TRUNCATE TABLE は、tp_thread_state テーブルに対して許可されていません。

27.12.17 パフォーマンススキーマクローンテーブル

注記

ここで説明する「パフォーマンススキーマ」テーブルは、MySQL 8.0.17 の時点で使用可能です。

次の各セクションでは、クローンプラグインに関連付けられた「パフォーマンススキーマ」テーブルについて説明します(セクション5.6.7「クローンプラグイン」を参照)。これらのテーブルは、クローニング操作に関する情報を示しています。

- `clone_status`: 現在または最後に実行されたクローニング操作に関するステータス情報。
- `clone_progress`: 現在または最後に実行されたクローニング操作に関する進捗情報。

パフォーマンススキーマクローンテーブルはクローンプラグインによって実装され、そのプラグインがロードおよびアンロードされるときにロードおよびアンロードされます(セクション5.6.7.1「クローンプラグインのインストール」を参照)。テーブルの特別な構成ステップは必要ありません。ただし、これらのテーブルは、有効にするクローンプラグインによって異なります。クローンプラグインがロードされているが無効になっている場合、テーブルは作成されません。

パフォーマンススキーマクローンプラグインのテーブルは、受信者の MySQL サーバーインスタンスでのみ使用されます。データは、サーバーの停止および再起動後も保持されます。

27.12.17.1 clone_status テーブル

注記

ここで説明する「パフォーマンススキーマ」テーブルは、MySQL 8.0.17 の時点で使用可能です。

`clone_status` テーブルには、現在または最後に実行されたクローニング操作のステータスのみが表示されます。テーブルには、1 行のデータのみが含まれているか、空です。

`clone_status` テーブルには、次のカラムがあります:

- **ID**
現在の MySQL サーバーインスタンス内の一意のクローニング操作識別子。
- **PID**
クローニング操作を実行しているセッションのプロセスリスト ID。
- **STATE**
クローニング操作の現在の状態。値には、**Not Started**, **In Progress**, **Completed** および **Failed** が含まれます。
- **BEGIN_TIME**
クローニング操作がいつ開始されたかを示す 'YYYY-MM-DD hh:mm:ss[.fraction]' 形式のタイムスタンプ。
- **END_TIME**
クローニング操作が終了した時間を示す 'YYYY-MM-DD hh:mm:ss[.fraction]' 形式のタイムスタンプ。操作が終了していない場合は NULL を報告します。
- **SOURCE**
'HOST:PORT' 形式のドナー MySQL サーバーアドレス。このコラムには、ローカルクローニング操作の '**LOCAL INSTANCE**' が表示されます。
- **DESTINATION**
クローニング先のディレクトリ。
- **ERROR_NO**
失敗したクローニング操作について報告されたエラー番号。
- **ERROR_MESSAGE**
失敗したクローニング操作のエラーメッセージ文字列。
- **BINLOG_FILE**
データがクローニングされるバイナリログファイルの名前。
- **BINLOG_POSITION**
データのクローン先のバイナリログファイルのオフセット。
- **GTID_EXECUTED**
最後にクローニングされたトランザクションの GTID 値。

`clone_status` テーブルは読取り専用です。 `TRUNCATE TABLE` を含む DDL は許可されていません。

27.12.17.2 clone_progress テーブル

注記

ここで説明する「パフォーマンススキーマ」テーブルは、MySQL 8.0.17 の時点で使用可能です。

`clone_progress` テーブルには、現在または最後に実行されたクローニング操作の進捗情報のみが表示されます。

クローニング操作のステージには、**DROP DATA**, **FILE COPY**, **PAGE_COPY**, **REDO_COPY**, **FILE_SYNC**, **RESTART** および **RECOVERY** が含まれます。クローニング操作では、各ステージのレコードが生成されます。したがって、テーブルには 7 行のデータのみが含まれているか、空です。

`clone_progress` テーブルには、次のカラムがあります:

- **ID**
現在の MySQL サーバーインスタンス内の一意的クローニング操作識別子。
- **STAGE**
現在のクローニングステージの名前。ステージには、`DROP DATA`、`FILE COPY`、`PAGE_COPY`、`REDO_COPY`、`FILE_SYNC`、`RESTART` および `RECOVERY` が含まれます。
- **STATE**
クローニングステージの現在の状態。状態には、`Not Started`、`In Progress` および `Completed` が含まれます。
- **BEGIN_TIME**
クローニングステージがいつ開始されたかを示す 'YYYY-MM-DD hh:mm:ss[.fraction]' 形式のタイムスタンプ。ステージが開始されていない場合は `NULL` を報告します。
- **END_TIME**
クローニングステージが終了した時間を示す 'YYYY-MM-DD hh:mm:ss[.fraction]' 形式のタイムスタンプ。ステージが終了していない場合は `NULL` を報告します。
- **THREADS**
ステージで使用されるコンカレントスレッドの数。
- **ESTIMATE**
現在のステージの推定データ量 (バイト単位)。
- **DATA**
現在の状態で転送されたデータ量 (バイト単位)。
- **NETWORK**
現在の状態で転送されたネットワークデータの量 (バイト単位)。
- **DATA_SPEED**
データ転送の現在の実際の速度 (バイト/秒)。この値は、`clone_max_data_bandwidth` で定義されているリクエストされた最大データ転送レートとは異なる場合があります。
- **NETWORK_SPEED**
ネットワーク転送の現在の速度 (バイト/秒)。

`clone_progress` テーブルは読取り専用です。 `TRUNCATE TABLE` を含む DDL は許可されていません。

27.12.18 パフォーマンススキーマサマリーテーブル

サマリーテーブルは、時間の経過とともに強制終了されたイベントについて集計された情報を提供します。このグループのテーブルには、さまざまな方法で、イベントデータが要約されます。

待機イベントサマリー

- `events_waits_summary_by_account_by_event_name`: アカウントおよびイベント名ごとの待機イベント
- `events_waits_summary_by_host_by_event_name`: ホスト名およびイベント名ごとの待機イベント
- `events_waits_summary_by_instance`: インスタンス当たりの待機イベント

- [events_waits_summary_by_thread_by_event_name](#): スレッドおよびイベント名ごとの待機イベント
- [events_waits_summary_by_user_by_event_name](#): ユーザー名およびイベント名ごとの待機イベント
- [events_waits_summary_global_by_event_name](#): イベント名ごとの待機イベント

ステージサマリー

- [events_stages_summary_by_account_by_event_name](#): アカウントおよびイベント名ごとのステージイベント
- [events_stages_summary_by_host_by_event_name](#): ホスト名およびイベント名ごとのイベントのステージング
- [events_stages_summary_by_thread_by_event_name](#): スレッドおよびイベント名ごとのステージ待機
- [events_stages_summary_by_user_by_event_name](#): ユーザー名およびイベント名ごとのステージイベント
- [events_stages_summary_global_by_event_name](#): イベント名ごとのステージ待機

ステートメントサマリー

- [events_statements_histogram_by_digest](#): スキーマおよびダイジェスト値ごとのステートメントヒストグラム。
- [events_statements_histogram_global](#): グローバルに要約されたステートメントヒストグラム。
- [events_statements_summary_by_account_by_event_name](#): アカウントおよびイベント名ごとのステートメントイベント
- [events_statements_summary_by_digest](#): スキーマおよびダイジェスト値ごとのステートメントイベント
- [events_statements_summary_by_host_by_event_name](#): ホスト名およびイベント名ごとのステートメントイベント
- [events_statements_summary_by_program](#): ストアドプログラムごとのステートメントイベント (ストアドプロシージャとストアドファンクション、トリガー、およびイベント)
- [events_statements_summary_by_thread_by_event_name](#): スレッドおよびイベント名ごとのステートメントイベント
- [events_statements_summary_by_user_by_event_name](#): ユーザー名およびイベント名ごとのステートメントイベント
- [events_statements_summary_global_by_event_name](#): イベント名ごとのステートメントイベント
- [prepared_statements_instances](#): プリペアドステートメントのインスタンスおよび統計

トランザクション要約

- [events_transactions_summary_by_account_by_event_name](#): アカウントおよびイベント名ごとのトランザクションイベント
- [events_transactions_summary_by_host_by_event_name](#): ホスト名およびイベント名ごとのトランザクションイベント
- [events_transactions_summary_by_thread_by_event_name](#): スレッドおよびイベント名ごとのトランザクションイベント
- [events_transactions_summary_by_user_by_event_name](#): ユーザー名およびイベント名ごとのトランザクションイベント
- [events_transactions_summary_global_by_event_name](#): イベント名ごとのトランザクションイベント

オブジェクト待機サマリー

- [objects_summary_global_by_type](#): オブジェクトサマリー

ファイル I/O サマリー

- [file_summary_by_event_name](#): イベント名ごとのファイルイベント
- [file_summary_by_instance](#): ファイルインスタンスごとのファイルイベント

テーブル I/O およびロック待機サマリー

- [table_io_waits_summary_by_index_usage](#): インデックスごとのテーブル I/O 待機
- [table_io_waits_summary_by_table](#): テーブルごとのテーブル I/O 待機
- [table_lock_waits_summary_by_table](#): テーブルごとのテーブルロック待機

ソケットサマリー

- [socket_summary_by_instance](#): インスタンス当たりのソケット待機および I/O
- [socket_summary_by_event_name](#): イベント名ごとのソケット待機および I/O

メモリーサマリー

- [memory_summary_by_account_by_event_name](#): アカウントおよびイベント名ごとのメモリー操作
- [memory_summary_by_host_by_event_name](#): ホストおよびイベント名ごとのメモリー操作
- [memory_summary_by_thread_by_event_name](#): スレッドおよびイベント名ごとのメモリー操作
- [memory_summary_by_user_by_event_name](#): ユーザーおよびイベント名ごとのメモリー操作
- [memory_summary_global_by_event_name](#): イベント名ごとのグローバルなメモリー操作

エラーサマリー

- [events_errors_summary_by_account_by_error](#): エラーコードおよびアカウントごとのエラー
- [events_errors_summary_by_host_by_error](#): エラーコードおよびホストごとのエラー
- [events_errors_summary_by_thread_by_error](#): エラーコードおよびスレッドごとのエラー
- [events_errors_summary_by_user_by_error](#): エラーコードおよびユーザーごとのエラー
- [events_errors_summary_global_by_error](#): エラーコードごとのエラー

ステータス変数サマリー

- [status_by_account](#): アカウントごとのステータス変数
- [status_by_host](#): ホスト名ごとのステータス変数
- [status_by_user](#): ユーザー名ごとのステータス変数

各サマリーテーブルには、集計するデータのグループ化の方法を決定するグループ化カラムと、集計値を格納するサマリーカラムがあります。同様の方法でイベントを要約するテーブルには、多くの場合に類似の一連のサマリーカラムがあり、イベントの集計方法を決定するために使用されるグループ化カラムのみ異なります。

サマリーテーブルは `TRUNCATE TABLE` で切り捨てることができます。一般に、サマリーカラムは行を削除するのではなく、0 または `NULL` にリセットされます。これにより、収集された値をクリアし、アグリゲーションを再開できます。それは、実行時構成の変更を行なったあとなどに便利な場合があります。この切捨て動作の例外は、個々のサマリーテーブルのセクションに記載されています。

27.12.18.1 待機イベント要約テーブル

パフォーマンススキーマは現在および最近の待機イベントを収集するためのテーブルを保守し、その情報をサマリーテーブルに集計します。セクション27.12.4「パフォーマンススキーマ待機イベントテーブル」に待機サマリーが基づいているイベントについて説明しています。待機イベントの内容、現在および最近の待機イベントテーブル、およびデフォルトで無効になっている待機イベント収集の制御方法については、その説明を参照してください。

待機イベントサマリー情報の例:

```
mysql> SELECT *
      FROM performance_schema.events_waits_summary_global_by_event_name\G
...
***** 6. row *****
EVENT_NAME: wait/synch/mutex/sql/BINARY_LOG::LOCK_index
COUNT_STAR: 8
SUM_TIMER_WAIT: 2119302
MIN_TIMER_WAIT: 196092
AVG_TIMER_WAIT: 264912
MAX_TIMER_WAIT: 569421
...
***** 9. row *****
EVENT_NAME: wait/synch/mutex/sql/hash_filo::lock
COUNT_STAR: 69
SUM_TIMER_WAIT: 16848828
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 244185
MAX_TIMER_WAIT: 735345
...
```

各待機イベントサマリーテーブルには、テーブルでのイベントの集計方法を示す1つ以上のグループ化カラムがあります。イベント名は、[setup_instruments](#) テーブル内のイベントインストゥルメントの名前を参照します:

- [events_waits_summary_by_account_by_event_name](#) には、[EVENT_NAME](#)、[USER](#) および [HOST](#) カラムがあります。各行には、特定のアカウント (ユーザーとホストの組合せ) およびイベント名のイベントが要約されます。
- [events_waits_summary_by_host_by_event_name](#) には、[EVENT_NAME](#) カラムおよび [HOST](#) カラムがあります。各行には、特定のホストおよびイベント名のイベントが要約されます。
- [events_waits_summary_by_instance](#) には [EVENT_NAME](#) および [OBJECT_INSTANCE_BEGIN](#) カラムがあります。各行は特定のイベント名とオブジェクトのイベントを要約します。インストゥルメントを使用して複数のインスタンスを作成する場合、各インスタンスは一意的な [OBJECT_INSTANCE_BEGIN](#) 値を持ち、このテーブルに個別にまとめられます。
- [events_waits_summary_by_thread_by_event_name](#) には [THREAD_ID](#) および [EVENT_NAME](#) カラムがあります。各行は特定のスレッドおよびイベント名のイベントを要約します。
- [events_waits_summary_by_user_by_event_name](#) には、[EVENT_NAME](#) カラムおよび [USER](#) カラムがあります。各行には、特定のユーザーおよびイベント名のイベントが要約されます。
- [events_waits_summary_global_by_event_name](#) には [EVENT_NAME](#) カラムがあります。各行は特定のイベント名のイベントを要約します。インストゥルメントを使用して、インストゥルメントされるオブジェクトの複数のインスタンスを作成できます。たとえば、接続ごとに作成される相互排他ロックのインストゥルメントがある場合、接続と同じ数のインスタンスがあります。インストゥルメントのサマリー行は、これらのすべてのインスタンス全体を要約します。

各待機イベントサマリーテーブルには、集計値を含む次のサマリーカラムがあります:

- [COUNT_STAR](#)
要約されたイベントの数。この値には、時間付きが時間なしに関係なく、すべてのイベントが含まれます。
- [SUM_TIMER_WAIT](#)
要約された時間付きイベントの合計待機時間。時間なしイベントは [NULL](#) の待機時間を持つため、この値は時間付きイベントに対してのみ計算されます。同じことがほかの [xxx_TIMER_WAIT](#) 値にも当てはまります。
- [MIN_TIMER_WAIT](#)

要約された時間付きイベントの最小待機時間。

- [AVG_TIMER_WAIT](#)

要約された時間付きイベントの平均待機時間。

- [MAX_TIMER_WAIT](#)

要約された時間付きイベントの最大待機時間。

待機イベントのサマリーテーブルには、次のインデックスがあります:

- [events_waits_summary_by_account_by_event_name:](#)

- 主キー (USER, HOST, EVENT_NAME)

- [events_waits_summary_by_host_by_event_name:](#)

- 主キー (HOST, EVENT_NAME)

- [events_waits_summary_by_instance:](#)

- 主キー (OBJECT_INSTANCE_BEGIN)
- (EVENT_NAME) のインデックス

- [events_waits_summary_by_thread_by_event_name:](#)

- 主キー (THREAD_ID, EVENT_NAME)

- [events_waits_summary_by_user_by_event_name:](#)

- 主キー (USER, EVENT_NAME)

- [events_waits_summary_global_by_event_name:](#)

- 主キー (EVENT_NAME)

[TRUNCATE TABLE](#) は待機サマリーテーブルに使用できます。これには次の効果があります:

- アカウント、ホストまたはユーザーごとに集計されないサマリーテーブルの場合、切捨てによって、行が削除されるのではなくサマリーカラムがゼロにリセットされます。
- アカウント、ホストまたはユーザー別に集計されたサマリーテーブルの場合、切捨てによって、接続のないアカウント、ホストまたはユーザーの行が削除され、残りの行のサマリーカラムがゼロにリセットされます。

また、アカウント、ホスト、ユーザーまたはスレッド別に集計された各待機サマリーテーブルは、依存する接続テーブルの切捨てまたは [events_waits_summary_global_by_event_name](#) の切捨てによって暗黙的に切り捨てられます。詳細は、[セクション27.12.8「パフォーマンススキーマ接続テーブル」](#)を参照してください。

27.12.18.2 ステージサマリーテーブル

パフォーマンススキーマは、現在および最近のステージイベントを収集するためのテーブルを保持し、その情報をサマリーテーブルに集約します。[セクション27.12.5「パフォーマンススキーマステージイベントテーブル」](#)では、ステージサマリーの基になるイベントが記述されます。ステージイベントの内容、現在および過去のステージイベントテーブル、およびデフォルトで無効になっているステージイベント収集の制御方法の詳細は、その説明を参照してください。

ステージイベントサマリー情報の例:

```
mysql> SELECT *
      FROM performance_schema.events_stages_summary_global_by_event_name\G
...
***** 5. row *****
```

```
EVENT_NAME: stage/sql/checking permissions
COUNT_STAR: 57
SUM_TIMER_WAIT: 26501888880
MIN_TIMER_WAIT: 7317456
AVG_TIMER_WAIT: 464945295
MAX_TIMER_WAIT: 12858936792
...
***** 9. row *****
EVENT_NAME: stage/sql/closing tables
COUNT_STAR: 37
SUM_TIMER_WAIT: 662606568
MIN_TIMER_WAIT: 1593864
AVG_TIMER_WAIT: 17907891
MAX_TIMER_WAIT: 437977248
...
```

各ステージサマリーテーブルには、テーブルのイベントの集計方法を示す 1 つまたは複数のグループ化カラムがあります。イベント名は、[setup_instruments](#) テーブル内のイベントインストゥルメントの名前を参照します:

- [events_stages_summary_by_account_by_event_name](#) には、[EVENT_NAME](#)、[USER](#) および [HOST](#) カラムがあります。各行には、特定のアカウント (ユーザーとホストの組合せ) およびイベント名のイベントが要約されます。
- [events_stages_summary_by_host_by_event_name](#) には、[EVENT_NAME](#) カラムおよび [HOST](#) カラムがあります。各行には、特定のホストおよびイベント名のイベントが要約されます。
- [events_stages_summary_by_thread_by_event_name](#) には [THREAD_ID](#) および [EVENT_NAME](#) カラムがあります。各行は特定のスレッドおよびイベント名のイベントを要約します。
- [events_stages_summary_by_user_by_event_name](#) には、[EVENT_NAME](#) カラムおよび [USER](#) カラムがあります。各行には、特定のユーザーおよびイベント名のイベントが要約されます。
- [events_stages_summary_global_by_event_name](#) には [EVENT_NAME](#) カラムがあります。各行は特定のイベント名のイベントを要約します。

各ステージ要約テーブルには、集計値を含む次の要約カラムがあります: [COUNT_STAR](#), [SUM_TIMER_WAIT](#), [MIN_TIMER_WAIT](#), [AVG_TIMER_WAIT](#) および [MAX_TIMER_WAIT](#)。これらのカラムは、待機イベントサマリーテーブル ([セクション 27.12.18.1 「待機イベント要約テーブル」](#) を参照) の同じ名前のカラムに似ていますが、ステージサマリーテーブルでは、[events_waits_current](#) ではなく [events_stages_current](#) のイベントが集計される点が異なります。

ステージサマリーテーブルには、次のインデックスがあります:

- [events_stages_summary_by_account_by_event_name](#):
 - 主キー ([USER](#), [HOST](#), [EVENT_NAME](#))
- [events_stages_summary_by_host_by_event_name](#):
 - 主キー ([HOST](#), [EVENT_NAME](#))
- [events_stages_summary_by_thread_by_event_name](#):
 - 主キー ([THREAD_ID](#), [EVENT_NAME](#))
- [events_stages_summary_by_user_by_event_name](#):
 - 主キー ([USER](#), [EVENT_NAME](#))
- [events_stages_summary_global_by_event_name](#):
 - 主キー ([EVENT_NAME](#))

[TRUNCATE TABLE](#) はステージサマリーテーブルに使用できます。これには次の効果があります:

- アカウント、ホストまたはユーザーごとに集計されないサマリーテーブルの場合、切捨てによって、行が削除されるのではなくサマリーカラムがゼロにリセットされます。

- アカウント、ホストまたはユーザー別に集計されたサマリーテーブルの場合、切捨てによって、接続のないアカウント、ホストまたはユーザーの行が削除され、残りの行のサマリーカラムがゼロにリセットされます。

また、アカウント、ホスト、ユーザーまたはスレッド別に集計された各ステージサマリーテーブルは、依存する接続テーブルの切捨てまたは [events_stages_summary_global_by_event_name](#) の切捨てによって暗黙的に切り捨てられます。詳細は、[セクション27.12.8「パフォーマンススキーマ接続テーブル」](#)を参照してください。

27.12.18.3 ステートメントサマリーテーブル

パフォーマンススキーマは、現在および最近のステートメントイベントを収集するためのテーブルを保持し、その情報をサマリーテーブルに集約します。[セクション27.12.6「パフォーマンススキーマステートメントイベントテーブル」](#)は、ステートメントのサマリーの基になるイベントを記述します。ステートメントイベントの内容、現在および過去のステートメントイベントテーブル、およびステートメントイベント収集の制御方法 (デフォルトでは部分的に無効になっています) については、その説明を参照してください。

ステートメントイベントサマリー情報の例:

```
mysql> SELECT *
FROM performance_schema.events_statements_summary_global_by_event_name\G
***** 1. row *****
EVENT_NAME: statement/sql/select
COUNT_STAR: 25
SUM_TIMER_WAIT: 1535983999000
MIN_TIMER_WAIT: 209823000
AVG_TIMER_WAIT: 61439359000
MAX_TIMER_WAIT: 1363397650000
SUM_LOCK_TIME: 20186000000
SUM_ERRORS: 0
SUM_WARNINGS: 0
SUM_ROWS_AFFECTED: 0
SUM_ROWS_SENT: 388
SUM_ROWS_EXAMINED: 370
SUM_CREATED_TMP_DISK_TABLES: 0
SUM_CREATED_TMP_TABLES: 0
SUM_SELECT_FULL_JOIN: 0
SUM_SELECT_FULL_RANGE_JOIN: 0
SUM_SELECT_RANGE: 0
SUM_SELECT_RANGE_CHECK: 0
SUM_SELECT_SCAN: 6
SUM_SORT_MERGE_PASSES: 0
SUM_SORT_RANGE: 0
SUM_SORT_ROWS: 0
SUM_SORT_SCAN: 0
SUM_NO_INDEX_USED: 6
SUM_NO_GOOD_INDEX_USED: 0
...
```

各ステートメントサマリーテーブルには、テーブルのイベントの集計方法を示す 1 つまたは複数のグループ化カラムがあります。イベント名は、[setup_instruments](#) テーブル内のイベントインストゥルメントの名前を参照します:

- [events_statements_summary_by_account_by_event_name](#) には、[EVENT_NAME](#)、[USER](#) および [HOST](#) カラムがあります。各行には、特定のアカウント (ユーザーとホストの組合せ) およびイベント名のイベントが要約されます。
- [events_statements_summary_by_digest](#) には [SCHEMA_NAME](#) および [DIGEST](#) カラムがあります。各行には、スキーマおよびダイジェスト値ごとにイベントが要約されます。(DIGEST_TEXT カラムには、対応する正規化されたステートメントダイジェストテキストが含まれますが、グループ化カラムでもサマリーカラムでもありません。[QUERY_SAMPLE_TEXT](#)、[QUERY_SAMPLE_SEEN](#) および [QUERY_SAMPLE_TIMER_WAIT](#) カラムもグループ化カラムでもサマリーカラムでもありません。これらはステートメントサンプリングをサポートしています。)

テーブルの最大行数は、サーバー起動時に自動サイズ設定されます。この最大を明示的に設定するには、サーバー起動時に [performance_schema_digests_size](#) システム変数を設定します。

- [events_statements_summary_by_host_by_event_name](#) には、[EVENT_NAME](#) カラムおよび [HOST](#) カラムがあります。各行には、特定のホストおよびイベント名のイベントが要約されます。
- [events_statements_summary_by_program](#) には、[OBJECT_TYPE](#)、[OBJECT_SCHEMA](#) および [OBJECT_NAME](#) カラムがあります。各行には、特定のストアドプログラム (ストアドプロシージャーまたはストアドファンクション、トリガー、またはイベント) のイベントが要約されます。

- [events_statements_summary_by_thread_by_event_name](#) には `THREAD_ID` および `EVENT_NAME` カラムがあります。各行は特定のスレッドおよびイベント名のイベントを要約します。
- [events_statements_summary_by_user_by_event_name](#) には、`EVENT_NAME` カラムおよび `USER` カラムがあります。各行には、特定のユーザーおよびイベント名のイベントが要約されます。
- [events_statements_summary_global_by_event_name](#) には `EVENT_NAME` カラムがあります。各行は特定のイベント名のイベントを要約します。
- [prepared_statements_instances](#) には `OBJECT_INSTANCE_BEGIN` カラムがあります。各行には、指定されたプリペアドステートメントのイベントが要約されます。

各ステートメント要約テーブルには、集計値を含む次の要約カラムがあります (ただし、次の例外があります):

- `COUNT_STAR`、`SUM_TIMER_WAIT`、`MIN_TIMER_WAIT`、`AVG_TIMER_WAIT`、`MAX_TIMER_WAIT`

これらのカラムは、ステートメントサマリーテーブルが `events_waits_current` ではなく `events_statements_current` からイベントを集約する点を除き、待機イベントサマリーテーブル ([セクション27.12.18.1「待機イベント要約テーブル」](#) を参照) 内の同じ名前のカラムに似ています。

`prepared_statements_instances` テーブルには、これらのカラムはありません。

- `SUM_xxx`

`events_statements_current` テーブル内の対応する `xxx` カラムの集計。たとえば、ステートメントサマリーテーブルの `SUM_LOCK_TIME` および `SUM_ERRORS` カラムは `events_statements_current` テーブルの `LOCK_TIME` および `ERRORS` カラムの集計です。

`events_statements_summary_by_digest` テーブルにはこれらの追加のサマリーカラムがあります。

- `FIRST_SEEN`、`LAST_SEEN`

指定されたダイジェスト値を持つステートメントがいつ最初に見られ、最後に見られたかを示すタイムスタンプ。

- `QUANTILE_95`: ステートメントレイテンシの 95 パーセンタイル (ピコ秒)。このパーセンタイルは、収集されたヒストグラムデータから計算された高い見積りです。つまり、特定のダイジェストについて、測定されたステートメントの 95% のレイテンシは `QUANTILE_95` よりも低くなります。

ヒストグラムデータにアクセスするには、[セクション27.12.18.4「ステートメントヒストグラム要約テーブル」](#) で説明されているテーブルを使用します。

- `QUANTILE_99`: `QUANTILE_95` と似ていますが、99 パーセンタイルです。
- `QUANTILE_999`: `QUANTILE_95` と似ていますが、99.9 の第一パーセンタイル用です。

`events_statements_summary_by_digest` テーブルには、次のカラムが含まれます。これらはグループ化カラムでもサマリーカラムでもなく、ステートメントサンプリングをサポートしています:

- `QUERY_SAMPLE_TEXT`

行にダイジェスト値を生成するサンプル SQL ステートメント。このカラムを使用すると、アプリケーションは、特定のダイジェスト値について、そのダイジェストを生成するサーバーによって実際に表示されるステートメントにアクセスできます。これを使用するには、頻繁に発生するダイジェストに関連付けられた代表的なステートメントの実行計画を調べるために、ステートメントに対して `EXPLAIN` を実行します。

`QUERY_SAMPLE_TEXT` カラムに値が割り当てられると、`QUERY_SAMPLE_SEEN` および `QUERY_SAMPLE_TIMER_WAIT` カラムにも値が割り当てられます。

ステートメントの表示に使用できる最大領域は、デフォルトで 1024 バイトです。この値を変更するには、サーバーの起動時に `performance_schema_max_sql_text_length` システム変数を設定します。(この値を変更すると、他の「パフォーマンススキーマ」テーブルのカラムにも影響します。[セクション27.10「パフォーマンススキーマのステートメントダイジェストとサンプリング」](#) を参照してください。)

ステートメントサンプリングの詳細は、[セクション27.10「パフォーマンススキーマのステートメントダイジェストとサンプリング」](#) を参照してください。

- [QUERY_SAMPLE_SEEN](#)

[QUERY_SAMPLE_TEXT](#) カラムのステートメントがいつ見られたかを示すタイムスタンプ。

- [QUERY_SAMPLE_TIMER_WAIT](#)

[QUERY_SAMPLE_TEXT](#) カラムのサンプルステートメントの待機時間。

[events_statements_summary_by_program](#) テーブルには、次の追加のサマリーカラムがあります:

- [COUNT_STATEMENTS](#), [SUM_STATEMENTS_WAIT](#), [MIN_STATEMENTS_WAIT](#), [AVG_STATEMENTS_WAIT](#), [MAX_STATEMENTS_WAIT](#)

ストアプログラムの実行中に呼び出されたネストされたステートメントに関する統計。

[prepared_statements_instances](#) テーブルには、次の追加のサマリーカラムがあります:

- [COUNT_EXECUTE](#), [SUM_TIMER_EXECUTE](#), [MIN_TIMER_EXECUTE](#), [AVG_TIMER_EXECUTE](#), [MAX_TIMER_EXECUTE](#)

プリペアドステートメントの実行の集計統計。

ステートメントサマリーテーブルには、次のインデックスがあります:

- [events_transactions_summary_by_account_by_event_name](#):

- 主キー (USER, HOST, EVENT_NAME)

- [events_statements_summary_by_digest](#):

- 主キー (SCHEMA_NAME, DIGEST)

- [events_transactions_summary_by_host_by_event_name](#):

- 主キー (HOST, EVENT_NAME)

- [events_statements_summary_by_program](#):

- 主キー (OBJECT_TYPE, OBJECT_SCHEMA, OBJECT_NAME)

- [events_statements_summary_by_thread_by_event_name](#):

- 主キー (THREAD_ID, EVENT_NAME)

- [events_transactions_summary_by_user_by_event_name](#):

- 主キー (USER, EVENT_NAME)

- [events_statements_summary_global_by_event_name](#):

- 主キー (EVENT_NAME)

[TRUNCATE TABLE](#) はステートメントサマリーテーブルに使用できます。これには次の効果があります:

- [events_statements_summary_by_digest](#) の場合は、行を削除します。

- アカウント、ホストまたはユーザーごとに集計されない他のサマリーテーブルの場合、切捨てによって、行が削除されるのではなくサマリーカラムがゼロにリセットされます。

- アカウント、ホストまたはユーザー別に集計された他のサマリーテーブルの場合、切捨てによって、接続のないアカウント、ホストまたはユーザーの行が削除され、残りの行のサマリーカラムがゼロにリセットされます。

また、アカウント、ホスト、ユーザーまたはスレッド別に集計された各ステートメントサマリーテーブルは、依存する接続テーブルの切捨てまたは [events_statements_summary_global_by_event_name](#) の切捨てによって暗黙的に切り捨てられます。詳細は、[セクション27.12.8「パフォーマンススキーマ接続テーブル」](#)を参照してください。

また、`events_statements_summary_by_digest` を切り捨てると `events_statements_histogram_by_digest` が暗黙的に切り捨てられ、`events_statements_summary_global_by_event_name` を切り捨てると `events_statements_histogram_global` が暗黙的に切り捨てられます。

ステートメントダイジェストアグリゲーションルール

`statements_digest` コンシューマが有効な場合、`events_statements_summary_by_digest` への集計はステートメントの完了時に次のように行われます。アグリゲーションはステートメントに対して計算された `DIGEST` 値に基づきます。

- 完了したばかりのステートメントのダイジェスト値のある `events_statements_summary_by_digest` 行がすでに存在する場合、ステートメントの統計はその行に集計されます。 `LAST_SEEN` カラムは現在の時間に更新されます。
- 完了したばかりのステートメントのダイジェスト値のある行がなく、テーブルがいっぱいでない場合、そのステートメントに対して新しい行が作成されます。 `FIRST_SEEN` および `LAST_SEEN` カラムは現在の時間で初期化されます。
- 完了したばかりのステートメントのステートメントダイジェスト値のある行がなく、テーブルがいっぱいである場合、完了したばかりのステートメントの統計が、必要に応じて作成される特別な「多目的」行に、`DIGEST = NULL` で追加されます。この行が作成された場合、`FIRST_SEEN` および `LAST_SEEN` カラムは現在の時間で初期化されます。そうでない場合、`LAST_SEEN` カラムが現在の時間で更新されます。

パフォーマンススキーマテーブルには、メモリー制約による最大サイズがあるため、`DIGEST = NULL` の行は保守されます。 `DIGEST = NULL` 行は、ほかの行に一致しないダイジェストが、サマリーテーブルがいっぱいである場合でも、共通の「ほかの」バケットを使用して、カウントされることを許可します。この行は、ダイジェストサマリーが代表的であるかどうかを推定するのに役立ちます。

- すべてのダイジェストのうち 5% を表す `COUNT_STAR` 値がある `DIGEST = NULL` 行は、ダイジェストサマリーテーブルがきわめて代表的であることを示します。ほかの行が、存在するステートメントの 95% を占めます。
- すべてのダイジェストのうち 50% を表す `COUNT_STAR` 値がある `DIGEST = NULL` 行は、ダイジェストサマリーテーブルがあまり代表的でないことを示します。ほかの行は、存在するステートメントの半分しか占めません。たいいていの場合に DBA は `DIGEST = NULL` 行にカウントされる行の多くが、代わりにより具体的な行を使用してカウントされるように、最大テーブルサイズを拡大するべきです。これを実行するには、サーバーの起動時に、`performance_schema_digests_size` システム変数を大きな値に設定します。デフォルトサイズは 200 です。

ストアプログラムインツルメンテーションの動作

`setup_objects` テーブルでインツルメンテーションが有効になっているストアプログラムタイプの場合、`events_statements_summary_by_program` は次のようにストアプログラムの統計を保持します：

- 行は、サーバーで最初に使用されたときにオブジェクトに追加されます。
- オブジェクトの行は、オブジェクトが削除されると削除されます。
- 統計は、実行時にオブジェクトの行で集計されます。

[セクション27.4.3「イベントの事前フィルタリング」](#) も参照してください。

27.12.18.4 ステートメントヒストグラム要約テーブル

パフォーマンススキーマは、ステートメントの最小待機時間、最大待機時間、および平均待機時間に関する情報を含むステートメントイベントのサマリーテーブルを保守します ([セクション27.12.18.3「ステートメントサマリーテーブル」](#) を参照)。これらのテーブルでは、システムパフォーマンスの高度な評価が可能です。よりきめ細かいレベルでの評価を許可するために、パフォーマンススキーマはステートメントの待機時間のヒストグラムデータも収集します。これらのヒストグラムを使用すると、待機時間分布をさらに把握できます。

[セクション27.12.6「パフォーマンススキーマステートメントイベントテーブル」](#) は、ステートメントのサマリーの基になるイベントを記述します。ステートメントイベントの内容、現在および過去のステートメントイベントテーブル、およびステートメントイベント収集の制御方法 (デフォルトでは部分的に無効になっています) については、その説明を参照してください。

ステートメントヒストグラム情報の例:

```
mysql> SELECT *
```

```
FROM performance_schema.events_statements_histogram_by_digest
WHERE SCHEMA_NAME = 'mydb' AND DIGEST = 'bb3f69453119b2d7b3ae40673a9d4c7c'
AND COUNT_BUCKET > 0 ORDER BY BUCKET_NUMBER\G
***** 1. row *****
  SCHEMA_NAME: mydb
    DIGEST: bb3f69453119b2d7b3ae40673a9d4c7c
  BUCKET_NUMBER: 42
  BUCKET_TIMER_LOW: 66069344
  BUCKET_TIMER_HIGH: 69183097
  COUNT_BUCKET: 1
COUNT_BUCKET_AND_LOWER: 1
  BUCKET_QUANTILE: 0.058824
***** 2. row *****
  SCHEMA_NAME: mydb
    DIGEST: bb3f69453119b2d7b3ae40673a9d4c7c
  BUCKET_NUMBER: 43
  BUCKET_TIMER_LOW: 69183097
  BUCKET_TIMER_HIGH: 72443596
  COUNT_BUCKET: 1
COUNT_BUCKET_AND_LOWER: 2
  BUCKET_QUANTILE: 0.117647
***** 3. row *****
  SCHEMA_NAME: mydb
    DIGEST: bb3f69453119b2d7b3ae40673a9d4c7c
  BUCKET_NUMBER: 44
  BUCKET_TIMER_LOW: 72443596
  BUCKET_TIMER_HIGH: 75857757
  COUNT_BUCKET: 2
COUNT_BUCKET_AND_LOWER: 4
  BUCKET_QUANTILE: 0.235294
***** 4. row *****
  SCHEMA_NAME: mydb
    DIGEST: bb3f69453119b2d7b3ae40673a9d4c7c
  BUCKET_NUMBER: 45
  BUCKET_TIMER_LOW: 75857757
  BUCKET_TIMER_HIGH: 79432823
  COUNT_BUCKET: 6
COUNT_BUCKET_AND_LOWER: 10
  BUCKET_QUANTILE: 0.625000
...
```

たとえば、3 行目の次の値は、23.52 % のクエリーが 75.86 マイクロ秒未満で実行されることを示しています：

```
BUCKET_TIMER_HIGH: 75857757
BUCKET_QUANTILE: 0.235294
```

行 4 の次の値は、62.50 のクエリーの割合が 79.44 マイクロ秒未満で実行されていることを示しています：

```
BUCKET_TIMER_HIGH: 79432823
BUCKET_QUANTILE: 0.625000
```

各ステートメントヒストグラムサマリーテーブルには、テーブルがイベントを集計する方法を示す 1 つ以上のグループ化カラムがあります：

- `events_statements_histogram_by_digest` には、`SCHEMA_NAME`、`DIGEST` および `BUCKET_NUMBER` カラムがあります：
 - `SCHEMA_NAME` および `DIGEST` カラムは、`events_statements_summary_by_digest` テーブルのステートメントダイジェスト行を識別します。
 - `SCHEMA_NAME` と `DIGEST` の値が同じ `events_statements_histogram_by_digest` 行は、そのスキーマとダイジェストの組合せのヒストグラムを構成します。
 - 特定のヒストグラム内で、`BUCKET_NUMBER` カラムはバケット番号を示します。
- `events_statements_histogram_global` には `BUCKET_NUMBER` カラムがあります。このテーブルは、単一のヒストグラムを使用して、スキーマ名とダイジェスト値の間でグローバルに待機時間を要約しています。`BUCKET_NUMBER` カラムは、このグローバルヒストグラム内のバケット番号を示します。

ヒストグラムは `N` バケットで構成され、各行は単一のバケットを表し、バケット番号は `BUCKET_NUMBER` カラムで示されます。バケット番号は 0 で始まります。

各ステートメントヒストグラムサマリーテーブルには、集計値を含む次のサマリーカラムがあります:

- [BUCKET_TIMER_LOW](#), [BUCKET_TIMER_HIGH](#)

バケットは、[BUCKET_TIMER_LOW](#) と [BUCKET_TIMER_HIGH](#) の間で測定されたレイテンシを持つステートメントをピコ秒単位でカウントします:

- 最初のバケット ([BUCKET_NUMBER](#) = 0) の [BUCKET_TIMER_LOW](#) の値は 0 です。
- バケット ([BUCKET_NUMBER](#) = k) の [BUCKET_TIMER_LOW](#) の値は、前のバケット ([BUCKET_NUMBER](#) = k-1) の [BUCKET_TIMER_HIGH](#) と同じです
- 最後のバケットは、ヒストグラム内の前のバケットを超えるレイテンシを持つステートメントのキャッチオールです。

- [COUNT_BUCKET](#)

[BUCKET_TIMER_LOW](#) から [BUCKET_TIMER_HIGH](#) までの間隔の待機時間で測定されたが、[BUCKET_TIMER_HIGH](#) を含まないステートメントの数。

- [COUNT_BUCKET_AND_LOWER](#)

0 から [BUCKET_TIMER_HIGH](#) を含まないまでの間隔で待機時間で測定されたステートメントの数。

- [BUCKET_QUANTILE](#)

このバケットまたは下位バケットに該当するステートメントの比率。この比率は、定義によって [COUNT_BUCKET_AND_LOWER](#) / [SUM\(COUNT_BUCKET\)](#) に対応し、利便性の高いカラムとして表示されます。

ステートメントヒストグラムサマリーテーブルには、次のインデックスがあります:

- [events_statements_histogram_by_digest](#):
 - ([SCHEMA_NAME](#), [DIGEST](#), [BUCKET_NUMBER](#)) の一意インデックス
- [events_statements_histogram_global](#):
 - 主キー ([BUCKET_NUMBER](#))

[TRUNCATE TABLE](#) は、ステートメントヒストグラムサマリーテーブルに使用できます。切捨てでは、[COUNT_BUCKET](#) および [COUNT_BUCKET_AND_LOWER](#) のカラムが 0 に設定されます。

また、[events_statements_summary_by_digest](#) を切り捨てると [events_statements_histogram_by_digest](#) が暗黙的に切り捨てられ、[events_statements_summary_global_by_event_name](#) を切り捨てると [events_statements_histogram_global](#) が暗黙的に切り捨てられます。

27.12.18.5 トランザクション要約テーブル

パフォーマンススキーマは、現在および最近のトランザクションイベントを収集するためのテーブルを保持し、その情報をサマリーテーブルに集約します。 [セクション27.12.7「パフォーマンススキーマのトランザクションテーブル」](#) では、トランザクションサマリーの基になるイベントが記述されます。 トランザクションイベントの内容、現在および過去のトランザクションイベントテーブル、およびデフォルトで無効になっているトランザクションイベント収集の制御方法の詳細は、その説明を参照してください。

トランザクションイベントのサマリー情報の例:

```
mysql> SELECT *
FROM performance_schema.events_transactions_summary_global_by_event_name
LIMIT 1\G
***** 1. row *****
EVENT_NAME: transaction
COUNT_STAR: 5
SUM_TIMER_WAIT: 19550092000
MIN_TIMER_WAIT: 2954148000
AVG_TIMER_WAIT: 3910018000
```

```
MAX_TIMER_WAIT: 5486275000
COUNT_READ_WRITE: 5
SUM_TIMER_READ_WRITE: 19550092000
MIN_TIMER_READ_WRITE: 2954148000
AVG_TIMER_READ_WRITE: 3910018000
MAX_TIMER_READ_WRITE: 5486275000
COUNT_READ_ONLY: 0
SUM_TIMER_READ_ONLY: 0
MIN_TIMER_READ_ONLY: 0
AVG_TIMER_READ_ONLY: 0
MAX_TIMER_READ_ONLY: 0
```

各トランザクション要約テーブルには、テーブルでのイベントの集計方法を示す 1 つ以上のグループ化カラムがあります。イベント名は、[setup_instruments](#) テーブル内のイベントインストゥルメントの名前を参照します:

- [events_transactions_summary_by_account_by_event_name](#) には、[USER](#)、[HOST](#) および [EVENT_NAME](#) カラムがあります。各行には、特定のアカウント (ユーザーとホストの組合せ) およびイベント名のイベントが要約されます。
- [events_transactions_summary_by_host_by_event_name](#) には、[HOST](#) カラムおよび [EVENT_NAME](#) カラムがあります。各行には、特定のホストおよびイベント名のイベントが要約されます。
- [events_transactions_summary_by_thread_by_event_name](#) には、[THREAD_ID](#) カラムおよび [EVENT_NAME](#) カラムがあります。各行は特定のスレッドおよびイベント名のイベントを要約します。
- [events_transactions_summary_by_user_by_event_name](#) には、[USER](#) カラムおよび [EVENT_NAME](#) カラムがあります。各行には、特定のユーザーおよびイベント名のイベントが要約されます。
- [events_transactions_summary_global_by_event_name](#) には [EVENT_NAME](#) カラムがあります。各行は特定のイベント名のイベントを要約します。

各トランザクション要約テーブルには、集計値を含む次の要約カラムがあります:

- [COUNT_STAR](#)、[SUM_TIMER_WAIT](#)、[MIN_TIMER_WAIT](#)、[AVG_TIMER_WAIT](#)、[MAX_TIMER_WAIT](#)

これらのカラムは、待機イベントサマリーテーブル ([セクション 27.12.18.1 「待機イベント要約テーブル」](#) を参照) の同じ名前のカラムに似ていますが、トランザクションサマリーテーブルでは、[events_waits_current](#) ではなく [events_transactions_current](#) のイベントが集計される点が異なります。これらのカラムには、読取り/書込みトランザクションと読取り専用トランザクションがまとめられています。

- [COUNT_READ_WRITE](#)、[SUM_TIMER_READ_WRITE](#)、[MIN_TIMER_READ_WRITE](#)、[AVG_TIMER_READ_WRITE](#)、[MAX_TIMER_READ_WRITE](#)

これらは、[COUNT_STAR](#) カラムおよび [xxx_TIMER_WAIT](#) カラムに似ていますが、読取り/書込みトランザクションのみを要約します。トランザクションアクセスモードでは、トランザクションが読取り/書込みモードと読取り専用モードのどちらで動作するかを指定します。

- [COUNT_READ_ONLY](#)、[SUM_TIMER_READ_ONLY](#)、[MIN_TIMER_READ_ONLY](#)、[AVG_TIMER_READ_ONLY](#)、[MAX_TIMER_READ_ONLY](#)

これらは、[COUNT_STAR](#) カラムおよび [xxx_TIMER_WAIT](#) カラムに似ていますが、読取り専用トランザクションのみを要約します。トランザクションアクセスモードでは、トランザクションが読取り/書込みモードと読取り専用モードのどちらで動作するかを指定します。

トランザクション要約テーブルには、次のインデックスがあります:

- [events_transactions_summary_by_account_by_event_name](#):
 - 主キー ([USER](#)、[HOST](#)、[EVENT_NAME](#))
- [events_transactions_summary_by_host_by_event_name](#):
 - 主キー ([HOST](#)、[EVENT_NAME](#))
- [events_transactions_summary_by_thread_by_event_name](#):

- 主キー (THREAD_ID、EVENT_NAME)
- `events_transactions_summary_by_user_by_event_name`:
 - 主キー (USER、EVENT_NAME)
- `events_transactions_summary_global_by_event_name`:
 - 主キー (EVENT_NAME)

TRUNCATE TABLE は、トランザクション要約テーブルに対して許可されています。これには次の効果があります:

- アカウント、ホストまたはユーザーごとに集計されないサマリーテーブルの場合、切捨てによって、行が削除されるのではなくサマリーカラムがゼロにリセットされます。
- アカウント、ホストまたはユーザー別に集計されたサマリーテーブルの場合、切捨てによって、接続のないアカウント、ホストまたはユーザーの行が削除され、残りの行のサマリーカラムがゼロにリセットされます。

また、アカウント、ホスト、ユーザーまたはスレッド別に集計された各トランザクションサマリーテーブルは、依存する接続テーブルの切捨てまたは `events_transactions_summary_global_by_event_name` の切捨てによって暗黙的に切り捨てられます。詳細は、[セクション27.12.8「パフォーマンススキーマ接続テーブル」](#)を参照してください。

トランザクション集計ルール

トランザクションイベント収集は、分離レベル、アクセスモードまたは自動コミットモードに関係なく行われます。

トランザクションイベント収集は、サーバーによって開始された中断されていないすべてのトランザクション (空のトランザクションを含む) に対して発生します。

読取り/書き込みトランザクションは通常、読取り専用トランザクションよりもリソース集中型であるため、トランザクションサマリーテーブルには、読取り/書き込みトランザクションと読取り専用トランザクション用の個別の集計カラムが含まれます。

リソース要件は、トランザクション分離レベルによっても異なる場合があります。ただし、サーバーごとに1つの分離レベルのみが使用されることを前提としており、分離レベルによる集計は提供されません。

27.12.18.6 オブジェクト待機サマリーテーブル

パフォーマンススキーマは、オブジェクト待機イベントを集約するための `objects_summary_global_by_type` テーブルを保守します。

オブジェクト待機イベントサマリー情報の例:

```
mysql> SELECT * FROM performance_schema.objects_summary_global_by_type\G
...
***** 3. row *****
  OBJECT_TYPE: TABLE
  OBJECT_SCHEMA: test
  OBJECT_NAME: t
  COUNT_STAR: 3
  SUM_TIMER_WAIT: 263126976
  MIN_TIMER_WAIT: 1522272
  AVG_TIMER_WAIT: 87708678
  MAX_TIMER_WAIT: 258428280
...
***** 10. row *****
  OBJECT_TYPE: TABLE
  OBJECT_SCHEMA: mysql
  OBJECT_NAME: user
  COUNT_STAR: 14
  SUM_TIMER_WAIT: 365567592
  MIN_TIMER_WAIT: 1141704
  AVG_TIMER_WAIT: 26111769
  MAX_TIMER_WAIT: 334783032
```


...

`objects_summary_global_by_type` テーブルには、テーブルによるイベントの集計方法を示す次のグループ化カラムがあります: `OBJECT_TYPE`、`OBJECT_SCHEMA` および `OBJECT_NAME`。各行は特定のオブジェクトのイベントを要約します。

`objects_summary_global_by_type` には `events_waits_summary_by_xxx` テーブルと同じサマリーカラムがあります。[セクション27.12.18.1「待機イベント要約テーブル」](#)を参照してください。

`objects_summary_global_by_type` テーブルには次のインデックスがあります:

- 主キー (`OBJECT_TYPE`, `OBJECT_SCHEMA`, `OBJECT_NAME`)

`TRUNCATE TABLE` はオブジェクトサマリーテーブルに使用できます。それは、行を削除するのではなく、サマリーカラムを 0 にリセットします。

27.12.18.7 ファイル I/O サマリーテーブル

パフォーマンススキーマは、I/O 操作に関する情報を集約するファイル I/O サマリーテーブルを保守します。

ファイル I/O イベントサマリー情報の例:

```
mysql> SELECT * FROM performance_schema.file_summary_by_event_name\G
...
***** 2. row *****
      EVENT_NAME: wait/io/file/sql/binlog
      COUNT_STAR: 31
      SUM_TIMER_WAIT: 8243784888
      MIN_TIMER_WAIT: 0
      AVG_TIMER_WAIT: 265928484
      MAX_TIMER_WAIT: 6490658832
...
mysql> SELECT * FROM performance_schema.file_summary_by_instance\G
...
***** 2. row *****
      FILE_NAME: /var/mysql/share/english/errmsg.sys
      EVENT_NAME: wait/io/file/sql/ERRMSG
      EVENT_NAME: wait/io/file/sql/ERRMSG
      OBJECT_INSTANCE_BEGIN: 4686193384
      COUNT_STAR: 5
      SUM_TIMER_WAIT: 13990154448
      MIN_TIMER_WAIT: 26349624
      AVG_TIMER_WAIT: 2798030607
      MAX_TIMER_WAIT: 8150662536
...
```

各ファイル I/O サマリーテーブルには、テーブルのイベントの集計方法を示す 1 つまたは複数のグループ化カラムがあります。イベント名は、`setup_instruments` テーブル内のイベントインストゥルメントの名前を参照します:

- `file_summary_by_event_name` には `EVENT_NAME` カラムがあります。各行は特定のイベント名のイベントを要約します。
- `file_summary_by_instance` には、`FILE_NAME`、`EVENT_NAME` および `OBJECT_INSTANCE_BEGIN` カラムがあります。各行は特定のファイルおよびイベント名のイベントを要約します。

各ファイル I/O サマリーテーブルには、集約された値を含む次のサマリーカラムがあります。一部のカラムは一般的で、より詳細なカラムの値の合計と同じ値を持ちます。このように、低レベルカラムを合計するユーザー定義ビューを必要とせずに、高レベルでのアグリゲーションを直接取得できます。

- `COUNT_STAR`、`SUM_TIMER_WAIT`、`MIN_TIMER_WAIT`、`AVG_TIMER_WAIT`、`MAX_TIMER_WAIT`

これらのカラムはすべての I/O 操作を集計します。

- `COUNT_READ`、`SUM_TIMER_READ`、`MIN_TIMER_READ`、`AVG_TIMER_READ`、`MAX_TIMER_READ`、`SUM_NUMBER_OF_BYTES_READ`

これらのカラムは `FGETS`、`FGETC`、`FREAD`、および `READ` を含むすべての読み取り操作を集計します。

- [COUNT_WRITE](#)、[SUM_TIMER_WRITE](#)、[MIN_TIMER_WRITE](#)、[AVG_TIMER_WRITE](#)、[MAX_TIMER_WRITE](#)、[SUM_NUMBER_O](#)

これらのカラムは [FPUTS](#)、[FPUTC](#)、[FPRINTF](#)、[VFPRINTF](#)、[FWRITE](#)、および [PWRITE](#) を含むすべての書き込み操作を集計します。

- [COUNT_MISC](#)、[SUM_TIMER_MISC](#)、[MIN_TIMER_MISC](#)、[AVG_TIMER_MISC](#)、[MAX_TIMER_MISC](#)

これらのカラムは [CREATE](#)、[DELETE](#)、[OPEN](#)、[CLOSE](#)、[STREAM_OPEN](#)、[STREAM_CLOSE](#)、[SEEK](#)、[TELL](#)、[FLUSH](#)、[STAT](#)、[FSTAT](#)、[CHSIZE](#) および [SYNC](#) を含むその他のすべての I/O 操作を集計します。これらの操作のバイトカウントはありません。

ファイル I/O サマリーテーブルには、次のインデックスがあります：

- [file_summary_by_event_name](#):

- 主キー ([EVENT_NAME](#))

- [file_summary_by_instance](#):

- 主キー ([OBJECT_INSTANCE_BEGIN](#))
- ([FILE_NAME](#)) のインデックス
- ([EVENT_NAME](#)) のインデックス

[TRUNCATE TABLE](#) はファイル I/O サマリーテーブルに使用できます。それは、行を削除するのではなく、サマリーカラムを 0 にリセットします。

MySQL サーバーでは、複数の手法を使用して、ファイルから読み取られた情報をキャッシュすることで I/O 操作を回避するため、I/O イベントが発生すると予想されるステートメントではそうしない可能性があります。キャッシュをフラッシュするか、サーバーを再起動して、その状態をリセットすることによって、I/O を発生させることができる場合があります。

27.12.18.8 テーブル I/O およびロック待機サマリーテーブル

次のセクションでは、テーブル I/O およびロック待機サマリーテーブルについて説明します。

- [table_io_waits_summary_by_index_usage](#): インデックスごとのテーブル I/O 待機
- [table_io_waits_summary_by_table](#): テーブルごとのテーブル I/O 待機
- [table_lock_waits_summary_by_table](#): テーブルごとのテーブルロック待機

[table_io_waits_summary_by_table](#) テーブル

[table_io_waits_summary_by_table](#) テーブルは、[wait/io/table/sql/handler](#) インストゥルメントによって生成されるすべてのテーブル I/O 待機イベントを集計します。グループ化はテーブル単位です。

[table_io_waits_summary_by_table](#) テーブルには、テーブルのイベントの集計方法を示すこれらのグループ化カラムがあります。[OBJECT_TYPE](#)、[OBJECT_SCHEMA](#)、および [OBJECT_NAME](#)。これらのカラムは、[events_waits_current](#) テーブル内と同じ意味を持ちます。それらは、行の適用先のテーブルを識別します。

[table_io_waits_summary_by_table](#) には集計された値を格納する次のサマリーカラムがあります。カラムの説明に示すように、一部のカラムは一般的で、より詳細なカラムの値の合計と同じ値を持ちます。たとえば、すべての書き込みを集計するカラムは、挿入、更新、および削除を集計する対応するカラムの合計を保持します。このように、低レベルカラムを合計するユーザー定義ビューを必要とせずに、高レベルでのアグリゲーションを直接取得できます。

- [COUNT_STAR](#)、[SUM_TIMER_WAIT](#)、[MIN_TIMER_WAIT](#)、[AVG_TIMER_WAIT](#)、[MAX_TIMER_WAIT](#)

これらのカラムはすべての I/O 操作を集計します。それらは対応する [xxx_READ](#) および [xxx_WRITE](#) カラムの合計と同じです。

- [COUNT_READ](#)、[SUM_TIMER_READ](#)、[MIN_TIMER_READ](#)、[AVG_TIMER_READ](#)、[MAX_TIMER_READ](#)

これらのカラムはすべての読み取り操作を集計します。それらに対応する `xxx_FETCH` カラムの合計と同じです。

- `COUNT_WRITE`、`SUM_TIMER_WRITE`、`MIN_TIMER_WRITE`、`AVG_TIMER_WRITE`、`MAX_TIMER_WRITE`
これらのカラムはすべての書き込み操作を集計します。それらに対応する `xxx_INSERT`、`xxx_UPDATE`、および `xxx_DELETE` カラムの合計と同じです。
- `COUNT_FETCH`、`SUM_TIMER_FETCH`、`MIN_TIMER_FETCH`、`AVG_TIMER_FETCH`、`MAX_TIMER_FETCH`
これらのカラムはすべてのフェッチ操作を集計します。
- `COUNT_INSERT`、`SUM_TIMER_INSERT`、`MIN_TIMER_INSERT`、`AVG_TIMER_INSERT`、`MAX_TIMER_INSERT`
これらのカラムはすべての挿入操作を集計します。
- `COUNT_UPDATE`、`SUM_TIMER_UPDATE`、`MIN_TIMER_UPDATE`、`AVG_TIMER_UPDATE`、`MAX_TIMER_UPDATE`
これらのカラムはすべての更新操作を集計します。
- `COUNT_DELETE`、`SUM_TIMER_DELETE`、`MIN_TIMER_DELETE`、`AVG_TIMER_DELETE`、`MAX_TIMER_DELETE`
これらのカラムはすべての削除操作を集計します。

`table_io_waits_summary_by_table` テーブルには次のインデックスがあります:

- `(OBJECT_TYPE, OBJECT_SCHEMA, OBJECT_NAME)` の一意インデックス

`TRUNCATE TABLE` はテーブル I/O サマリーテーブルに使用できます。それは、行を削除するのではなく、サマリーカラムを 0 にリセットします。このテーブルを切り捨てると、`table_io_waits_summary_by_index_usage` テーブルも切り捨てられます。

`table_io_waits_summary_by_index_usage` テーブル

`table_io_waits_summary_by_index_usage` テーブルは、`wait/io/table/sql/handler` インストゥルメントによって生成されるすべてのテーブルインデックス I/O 待機イベントを集計します。グループ化はテーブルインデックス単位です。

`table_io_waits_summary_by_index_usage` のカラムは、`table_io_waits_summary_by_table` とほぼ同じです。唯一の違いは、テーブル I/O 待機イベントが記録されたときに使用されたインデックスの名前に対応する、追加のグループカラム `INDEX_NAME` です。

- `PRIMARY` の値はテーブル I/O でプライマリインデックスが使用されたことを示します。
- `NULL` の値はテーブル I/O でインデックスが使用されなかったことを示します。
- 挿入は `INDEX_NAME = NULL` に対してカウントされます。

`table_io_waits_summary_by_index_usage` テーブルには次のインデックスがあります:

- `(OBJECT_TYPE, OBJECT_SCHEMA, OBJECT_NAME, INDEX_NAME)` の一意インデックス

`TRUNCATE TABLE` はテーブル I/O サマリーテーブルに使用できます。それは、行を削除するのではなく、サマリーカラムを 0 にリセットします。このテーブルも、`table_io_waits_summary_by_table` テーブルの切り捨てによって切り捨てられます。テーブルのインデックス構造を変更する DDL 操作により、インデックスごとの統計がリセットされることがあります。

`table_lock_waits_summary_by_table` テーブル

`table_lock_waits_summary_by_table` テーブルは、`wait/lock/table/sql/handler` インストゥルメントによって生成されるすべてのテーブルロック待機イベントを集計します。グループ化はテーブル単位です。

このテーブルには、内部および外部ロックに関する情報が格納されます。

- 内部ロックは、SQL レイヤーのロックに対応します。これは現在 `thr_lock()` への呼び出しによって実装されます。イベント行では、これらのロックは次のいずれかの値を持つ `OPERATION` カラムによって区別されます:

```
read normal
read with shared locks
read high priority
read no insert
write allow write
write concurrent insert
write delayed
write low priority
write normal
```

- 外部ロックはストレージエンジンレイヤーのロックに対応します。これは現在 `handler::external_lock()` への呼び出しによって実装されます。イベント行では、これらのロックは次のいずれかの値を持つ `OPERATION` カラムによって区別されます:

```
read external
write external
```

`table_lock_waits_summary_by_table` テーブルには、テーブルのイベントの集計方法を示すこれらのグループ化カラムがあります。 `OBJECT_TYPE`、 `OBJECT_SCHEMA`、および `OBJECT_NAME`。これらのカラムは、 `events_waits_current` テーブル内と同じ意味を持ちます。それらは、行の適用先のテーブルを識別します。

`table_lock_waits_summary_by_table` には集計された値を格納する次のサマリーカラムがあります。カラムの説明に示すように、一部のカラムは一般的で、より詳細なカラムの値の合計と同じ値を持ちます。たとえば、すべてのロックを集計するカラムは、読み取りおよび書き込みロックを集計する対応するカラムの合計を保持します。このように、低レベルカラムを合計するユーザー定義ビューを必要とせずに、高レベルでのアグリゲーションを直接取得できます。

- `COUNT_STAR`、 `SUM_TIMER_WAIT`、 `MIN_TIMER_WAIT`、 `AVG_TIMER_WAIT`、 `MAX_TIMER_WAIT`

これらのカラムはすべてのロック操作を集計します。それらは対応する `xxx_READ` および `xxx_WRITE` カラムの合計と同じです。

- `COUNT_READ`、 `SUM_TIMER_READ`、 `MIN_TIMER_READ`、 `AVG_TIMER_READ`、 `MAX_TIMER_READ`

これらのカラムはすべての読み取りロック操作を集計します。それらは対応する `xxx_READ_NORMAL`、 `xxx_READ_WITH_SHARED_LOCKS`、 `xxx_READ_HIGH_PRIORITY`、および `xxx_READ_NO_INSERT` カラムの合計と同じです。

- `COUNT_WRITE`、 `SUM_TIMER_WRITE`、 `MIN_TIMER_WRITE`、 `AVG_TIMER_WRITE`、 `MAX_TIMER_WRITE`

これらのカラムはすべての書き込みロック操作を集計します。これらは、対応する `xxx_WRITE_ALLOW_WRITE`、 `xxx_WRITE_CONCURRENT_INSERT`、 `xxx_WRITE_LOW_PRIORITY` および `xxx_WRITE_NORMAL` のカラムの合計と同じです。

- `COUNT_READ_NORMAL`、 `SUM_TIMER_READ_NORMAL`、 `MIN_TIMER_READ_NORMAL`、 `AVG_TIMER_READ_NORMAL`、 `MAX_TIMER_READ_NORMAL`

これらのカラムは内部読み取りロックを集計します。

- `COUNT_READ_WITH_SHARED_LOCKS`、 `SUM_TIMER_READ_WITH_SHARED_LOCKS`、 `MIN_TIMER_READ_WITH_SHARED_LOCKS`、 `AVG_TIMER_READ_WITH_SHARED_LOCKS`、 `MAX_TIMER_READ_WITH_SHARED_LOCKS`

これらのカラムは内部読み取りロックを集計します。

- `COUNT_READ_HIGH_PRIORITY`、 `SUM_TIMER_READ_HIGH_PRIORITY`、 `MIN_TIMER_READ_HIGH_PRIORITY`、 `AVG_TIMER_READ_HIGH_PRIORITY`、 `MAX_TIMER_READ_HIGH_PRIORITY`

これらのカラムは内部読み取りロックを集計します。

- `COUNT_READ_NO_INSERT`、 `SUM_TIMER_READ_NO_INSERT`、 `MIN_TIMER_READ_NO_INSERT`、 `AVG_TIMER_READ_NO_INSERT`、 `MAX_TIMER_READ_NO_INSERT`

これらのカラムは内部読み取りロックを集計します。

- `COUNT_READ_EXTERNAL`、 `SUM_TIMER_READ_EXTERNAL`、 `MIN_TIMER_READ_EXTERNAL`、 `AVG_TIMER_READ_EXTERNAL`、 `MAX_TIMER_READ_EXTERNAL`

これらのカラムは外部読み取りロックを集計します。

- `COUNT_WRITE_ALLOW_WRITE`、 `SUM_TIMER_WRITE_ALLOW_WRITE`、 `MIN_TIMER_WRITE_ALLOW_WRITE`、 `AVG_TIMER_WRITE_ALLOW_WRITE`、 `MAX_TIMER_WRITE_ALLOW_WRITE`

これらのカラムは内部書き込みロックを集計します。

- `COUNT_WRITE_CONCURRENT_INSERT`、`SUM_TIMER_WRITE_CONCURRENT_INSERT`、`MIN_TIMER_WRITE_CONCURRENT_INSERT`、`AVG_TIMER_WRITE_CONCURRENT_INSERT`

これらのカラムは内部書き込みロックを集計します。

- `COUNT_WRITE_LOW_PRIORITY`、`SUM_TIMER_WRITE_LOW_PRIORITY`、`MIN_TIMER_WRITE_LOW_PRIORITY`、`AVG_TIMER_WRITE_LOW_PRIORITY`

これらのカラムは内部書き込みロックを集計します。

- `COUNT_WRITE_NORMAL`、`SUM_TIMER_WRITE_NORMAL`、`MIN_TIMER_WRITE_NORMAL`、`AVG_TIMER_WRITE_NORMAL`

これらのカラムは内部書き込みロックを集計します。

- `COUNT_WRITE_EXTERNAL`、`SUM_TIMER_WRITE_EXTERNAL`、`MIN_TIMER_WRITE_EXTERNAL`、`AVG_TIMER_WRITE_EXTERNAL`

これらのカラムは外部書き込みロックを集計します。

`table_lock_waits_summary_by_table` テーブルには次のインデックスがあります:

- `(OBJECT_TYPE, OBJECT_SCHEMA, OBJECT_NAME)` の一意インデックス

`TRUNCATE TABLE` はテーブルロックサマリーテーブルに使用できます。それは、行を削除するのではなく、サマリーカラムを 0 にリセットします。

27.12.18.9 ソケットサマリーテーブル

これらのソケットサマリーテーブルは、ソケット操作のタイマーおよびバイトカウント情報を集計します。

- `socket_summary_by_event_name`: ソケットインストゥルメントごとに、すべてのソケット I/O 操作の `wait/io/socket/*` インストゥルメントによって生成されたタイマーおよびバイトカウント統計を集計します。
- `socket_summary_by_instance`: ソケットインスタンスごとに、すべてのソケット I/O 操作の `wait/io/socket/*` インストゥルメントによって生成されたタイマーおよびバイトカウント統計を集計します。接続が終了すると、`socket_summary_by_instance` 内のそれに対応する行が削除されます。

ソケットサマリーテーブルは、ソケットがクライアントからの次のリクエストを待っている間に、`idle` イベントによって生成される待機は集計しません。`idle` イベントアグリゲーションには、待機イベントサマリーテーブルを使用します。[セクション27.12.18.1「待機イベント要約テーブル」](#)を参照してください。

各ソケットサマリーテーブルには、テーブルのイベントの集計方法を示す 1 つまたは複数のグループ化カラムがあります。イベント名は、`setup_instruments` テーブル内のイベントインストゥルメントの名前を参照します:

- `socket_summary_by_event_name` には `EVENT_NAME` カラムがあります。各行は特定のイベント名のイベントを要約します。
- `socket_summary_by_instance` には `OBJECT_INSTANCE_BEGIN` カラムがあります。各行は特定のオブジェクトのイベントを要約します。

各ソケットサマリーテーブルには、集約された値を含む次のサマリーカラムがあります:

- `COUNT_STAR`、`SUM_TIMER_WAIT`、`MIN_TIMER_WAIT`、`AVG_TIMER_WAIT`、`MAX_TIMER_WAIT`

これらのカラムはすべての操作を集計します。

- `COUNT_READ`、`SUM_TIMER_READ`、`MIN_TIMER_READ`、`AVG_TIMER_READ`、`MAX_TIMER_READ`、`SUM_NUMBER_RECEIVED`

これらのカラムはすべての受信操作 (`RECV`、`RECVFROM`、および `RECVMSG`) を集計します。

- `COUNT_WRITE`、`SUM_TIMER_WRITE`、`MIN_TIMER_WRITE`、`AVG_TIMER_WRITE`、`MAX_TIMER_WRITE`、`SUM_NUMBER_SENT`

これらのカラムはすべての送信操作 (`SEND`、`SENDTO`、および `SENDMSG`) を集計します。

- `COUNT_MISC`、`SUM_TIMER_MISC`、`MIN_TIMER_MISC`、`AVG_TIMER_MISC`、`MAX_TIMER_MISC`

これらのカラムは [CONNECT](#)、[LISTEN](#)、[ACCEPT](#)、[CLOSE](#)、および [SHUTDOWN](#) などのほかのすべてのソケット操作を集計します。これらの操作のバイトカウントはありません。

[socket_summary_by_instance](#) テーブルには、ソケットのクラス ([client_connection](#)、[server_tcpip_socket](#)、[server_unix_socket](#)) を示す [EVENT_NAME](#) カラムもあります。このカラムは、たとえば、クライアントアクティビティをサーバー待機ソケットのそれから分離するために、グループ化できます。

ソケットサマリーテーブルには、次のインデックスがあります:

- [socket_summary_by_event_name](#):
 - 主キー ([EVENT_NAME](#))
- [socket_summary_by_instance](#):
 - 主キー ([OBJECT_INSTANCE_BEGIN](#))
 - ([EVENT_NAME](#)) のインデックス

[TRUNCATE TABLE](#) はソケットサマリーテーブルに使用できます。 [events_statements_summary_by_digest](#) を除き、`tt` は行を削除するのではなく、サマリーカラムを 0 にリセットします。

27.12.18.10 メモリーサマリーテーブル

パフォーマンススキーマは、メモリー使用量を計測し、メモリー使用量の統計を集約します。詳細は、次の要因を参照してください:

- 使用されるメモリーのタイプ (様々なキャッシュ、内部バッファなど)
- メモリー操作を間接的に実行するスレッド、アカウント、ユーザー、ホスト

パフォーマンススキーマは、メモリー使用の次の側面を計測

- 使用済メモリーサイズ
- 操作数
- 最低水位標と最高水位標

メモリーサイズは、サーバーのメモリー消費を理解またはチューニングするのに役立ちます。

操作数は、サーバーがメモリーアロケータに与える全体的な圧力を理解またはチューニングするのに役立ちます。これはパフォーマンスに影響します。シングルバイトを 100 万回割り当てることは、100 万バイトを一度に割り当てることと同じではありません。サイズとカウントの両方を追跡すると、違いが生じる可能性があります。

最低水位標と最高水位標は、ワークロードのスパイク、全体的なワークロードの安定性およびメモリーリークの可能性を検出するために重要です。

メモリーサマリーテーブルには、メモリーイベントが時間指定されていないため、タイミング情報は含まれません。

メモリー使用量データの収集の詳細は、[メモリーインストゥルメンテーションの動作](#) を参照してください。

メモリーイベントのサマリー情報の例:

```
mysql> SELECT *
FROM performance_schema.memory_summary_global_by_event_name
WHERE EVENT_NAME = 'memory/sql/TABLE\G
***** 1. row *****
EVENT_NAME: memory/sql/TABLE
COUNT_ALLOC: 1381
COUNT_FREE: 924
SUM_NUMBER_OF_BYTES_ALLOC: 2059873
SUM_NUMBER_OF_BYTES_FREE: 1407432
```



```
LOW_COUNT_USED: 0
CURRENT_COUNT_USED: 457
HIGH_COUNT_USED: 461
LOW_NUMBER_OF_BYTES_USED: 0
CURRENT_NUMBER_OF_BYTES_USED: 652441
HIGH_NUMBER_OF_BYTES_USED: 669269
```

各メモリーサマリーテーブルには、テーブルがイベントを集約する方法を示す 1 つまたは複数のグループ化カラムがあります。イベント名は、[setup_instruments](#) テーブル内のイベントインストゥルメントの名前を参照します:

- [memory_summary_by_account_by_event_name](#) には、[USER](#)、[HOST](#) および [EVENT_NAME](#) カラムがあります。各行には、特定のアカウント (ユーザーとホストの組合せ) およびイベント名のイベントが要約されます。
- [memory_summary_by_host_by_event_name](#) には、[HOST](#) カラムおよび [EVENT_NAME](#) カラムがあります。各行には、特定のホストおよびイベント名のイベントが要約されます。
- [memory_summary_by_thread_by_event_name](#) には、[THREAD_ID](#) カラムおよび [EVENT_NAME](#) カラムがあります。各行は特定のスレッドおよびイベント名のイベントを要約します。
- [memory_summary_by_user_by_event_name](#) には、[USER](#) カラムおよび [EVENT_NAME](#) カラムがあります。各行には、特定のユーザーおよびイベント名のイベントが要約されます。
- [memory_summary_global_by_event_name](#) には [EVENT_NAME](#) カラムがあります。各行は特定のイベント名のイベントを要約します。

各メモリーサマリーテーブルには、集計値を含む次のサマリーカラムがあります:

- [COUNT_ALLOC](#), [COUNT_FREE](#)
メモリー割り当て関数およびメモリー解放関数への呼び出しの総数。
- [SUM_NUMBER_OF_BYTES_ALLOC](#), [SUM_NUMBER_OF_BYTES_FREE](#)
割り当てられたメモリーブロックおよび解放されたメモリーブロックの集計サイズ。
- [CURRENT_COUNT_USED](#)
まだ解放されていない現在割り当てられているブロックの集計数。これは、[COUNT_ALLOC-COUNT_FREE](#) と同等の便利なカラムです。
- [CURRENT_NUMBER_OF_BYTES_USED](#)
まだ解放されていない、現在割り当てられているメモリーブロックの集計サイズ。これは、[SUM_NUMBER_OF_BYTES_ALLOC-SUM_NUMBER_OF_BYTES_FREE](#) と同等の便利なカラムです。
- [LOW_COUNT_USED](#), [HIGH_COUNT_USED](#)
[CURRENT_COUNT_USED](#) カラムに対応する最低水位標および最高水位標。
- [LOW_NUMBER_OF_BYTES_USED](#), [HIGH_NUMBER_OF_BYTES_USED](#)
[CURRENT_NUMBER_OF_BYTES_USED](#) カラムに対応する最低水位標および最高水位標。

メモリーサマリーテーブルには、次のインデックスがあります:

- [memory_summary_by_account_by_event_name](#):
 - 主キー ([USER](#), [HOST](#), [EVENT_NAME](#))
- [memory_summary_by_host_by_event_name](#):
 - 主キー ([HOST](#), [EVENT_NAME](#))
- [memory_summary_by_thread_by_event_name](#):
 - 主キー ([THREAD_ID](#), [EVENT_NAME](#))

- [memory_summary_by_user_by_event_name](#):

- 主キー (USER、EVENT_NAME)

- [memory_summary_global_by_event_name](#):

- 主キー (EVENT_NAME)

TRUNCATE TABLE はメモリーサマリーテーブルに対して許可されています。これには次の効果があります:

- 通常、切捨てによって統計のベースラインがリセットされますが、サーバーの状態は変更されません。つまり、メモリーテーブルを切り捨ててもメモリーは解放されません。
- [COUNT_ALLOC](#) および [COUNT_FREE](#) は、各カウンタを同じ値だけ減らすことで、新しいベースラインにリセットされます。
- 同様に、[SUM_NUMBER_OF_BYTES_ALLOC](#) および [SUM_NUMBER_OF_BYTES_FREE](#) は新しいベースラインにリセットされます。
- [LOW_COUNT_USED](#) および [HIGH_COUNT_USED](#) は [CURRENT_COUNT_USED](#) にリセットされます。
- [LOW_NUMBER_OF_BYTES_USED](#) および [HIGH_NUMBER_OF_BYTES_USED](#) は [CURRENT_NUMBER_OF_BYTES_USED](#) にリセットされます。

また、アカウント、ホスト、ユーザーまたはスレッド別に集計された各メモリーサマリーテーブルは、依存する接続テーブルの切捨てまたは [memory_summary_global_by_event_name](#) の切捨てによって暗黙的に切り捨てられます。詳細は、[セクション27.12.8「パフォーマンススキーマ接続テーブル」](#)を参照してください。

メモリーインストゥルメンテーションの動作

メモリーインストゥルメントは [setup_instruments](#) テーブルにリストされ、[memory/code_area/instrument_name](#) という形式の名前を持ちます。メモリーインストゥルメンテーションはデフォルトで有効になっています。

接頭辞 [memory/performance_schema/](#)が付いたインストゥルメントは、パフォーマンススキーマ自体の内部バッファに割り当てられているメモリー量を公開します。[memory/performance_schema/](#)インストゥルメントは組込みであり、常に有効になっており、起動時または実行時に無効にすることはできません。組込みメモリーインストゥルメントは、[memory_summary_global_by_event_name](#) テーブルにのみ表示されます。

サーバー起動時のメモリーインストゥルメンテーションの状態を制御するには、[my.cnf](#) ファイルで次のような行を使用します:

- 有効化:

```
[mysqld]
performance-schema-instrument='memory/%=ON'
```

- 無効化:

```
[mysqld]
performance-schema-instrument='memory/%=OFF'
```

実行時のメモリーインストゥルメンテーションの状態を制御するには、[setup_instruments](#) テーブルの関連インストゥルメントの [ENABLED](#) カラムを更新します:

- 有効化:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'YES'
WHERE NAME LIKE 'memory/%';
```

- 無効化:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'NO'
WHERE NAME LIKE 'memory/%';
```

メモリーインストゥルメントの場合、メモリー操作は時間外であるため、`setup_instruments` の `TIMED` カラムは無視されます。

サーバー内のスレッドがインストゥルメントされたメモリー割当てを実行する場合、次のルールが適用されます:

- スレッドがインストゥルメントされていない場合、またはメモリーインストゥルメントが有効になっていない場合、割り当てられたメモリーブロックはインストゥルメントされません。
- それ以外の場合 (つまり、スレッドとインストゥルメントの両方が有効になっている場合)、割り当てられたメモリーブロックがインストゥルメントされます。

割当て解除には、次のルールが適用されます:

- メモリー割当て操作がインストゥルメントされた場合、現在のインストゥルメントまたはスレッド有効ステータスに関係なく、対応する空き操作がインストゥルメントされます。
- メモリー割当て操作がインストゥルメントされていない場合、現在のインストゥルメントまたはスレッド有効ステータスに関係なく、対応する空き操作はインストゥルメントされません。

スレッドごとの統計には、次のルールが適用されます。

計測されたサイズ `N` のメモリーブロックが割り当てられると、パフォーマンススキーマはメモリーサマリーテーブルのカラムを次のように更新します:

- `COUNT_ALLOC`: 増加 1
- `CURRENT_COUNT_USED`: 増加 1
- `HIGH_COUNT_USED`: `CURRENT_COUNT_USED` が新しい最大値の場合に増加
- `SUM_NUMBER_OF_BYTES_ALLOC`: `N` による増加
- `CURRENT_NUMBER_OF_BYTES_USED`: `N` による増加
- `HIGH_NUMBER_OF_BYTES_USED`: `CURRENT_NUMBER_OF_BYTES_USED` が新しい最大値の場合に増加

インストゥルメントされたメモリーブロックの割り当てが解除されると、パフォーマンススキーマはメモリーサマリーテーブルのカラムを次のように更新します:

- `COUNT_FREE`: 増加 1
- `CURRENT_COUNT_USED`: 減少 1
- `LOW_COUNT_USED`: `CURRENT_COUNT_USED` が新しい最小値の場合は減少
- `SUM_NUMBER_OF_BYTES_FREE`: `N` による増加
- `CURRENT_NUMBER_OF_BYTES_USED`: `N` による減少
- `LOW_NUMBER_OF_BYTES_USED`: `CURRENT_NUMBER_OF_BYTES_USED` が新しい最小値の場合は減少

高レベルの集計 (アカウント別、ユーザー別、ホスト別) では、最低水位標と最高水位標に同じルールが適用されます。

- `LOW_COUNT_USED` および `LOW_NUMBER_OF_BYTES_USED` は、より低い見積りです。パフォーマンススキーマによって報告される値は、実行時に効率的に使用されるメモリーの最小数または最小サイズ以下であることが保証されます。
- `HIGH_COUNT_USED` および `HIGH_NUMBER_OF_BYTES_USED` は、より高い見積りです。パフォーマンススキーマによって報告される値は、実行時に効率的に使用されるメモリーの最大数または最大サイズ以上であることが保証されます。

`memory_summary_global_by_event_name` 以外のサマリーテーブルの見積りが低い場合、スレッド間でメモリー所有権が転送されると、値が負になる可能性があります。

見積計算の例を次に示しますが、見積実装は変更される可能性があることに注意してください:

スレッド 1 は、`memory_summary_by_thread_by_event_name` テーブルの `LOW_NUMBER_OF_BYTES_USED` カラムおよび `HIGH_NUMBER_OF_BYTES_USED` カラムで報告されているように、実行中に 1MB から 2MB の範囲のメモリーを使用します。

スレッド 2 は、同様に、実行中に 10MB から 12MB の範囲のメモリーを使用します。

これらの 2 つのスレッドが同じユーザーアカウントに属している場合、アカウントごとのサマリーでは、このアカウントが 11MB から 14MB の範囲でメモリーを使用したと推定されます。つまり、上位レベルの集計の `LOW_NUMBER_OF_BYTES_USED` は、各 `LOW_NUMBER_OF_BYTES_USED` の合計です (最悪の場合)。同様に、上位レベルの集計の `HIGH_NUMBER_OF_BYTES_USED` は、各 `HIGH_NUMBER_OF_BYTES_USED` の合計です (最悪の場合)。

11MB は、両方のスレッドが同時に低い使用マークに達した場合にのみ発生する可能性のある低い見積りです。

14MB は、両方のスレッドが同時に高使用量マークに達した場合にのみ発生する可能性のある高い見積りです。

このアカウントの実際のメモリー使用量は、11.5 MB から 13.5 MB の範囲内である可能性があります。

キャパシティプランニングでは、セッションが相関していない場合に発生する可能性があることが示されているため (通常は、最悪のケースのレポートが望ましい動作です)。

27.12.18.11 エラー要約テーブル

パフォーマンススキーマは、サーバーエラー (および警告) に関する統計情報を集約するためのサマリーテーブルを保持します。サーバーエラーのリストは、[Server Error Message Reference](#) を参照してください。

エラー情報の収集は、デフォルトで有効になっている `error` インストゥルメントによって制御されます。タイミング情報は収集されません。

各エラーサマリーテーブルには、エラーを識別する 3 つのカラムがあります:

- `ERROR_NUMBER` は数値のエラー値です。値は一意です。
- `ERROR_NAME` は、`ERROR_NUMBER` 値に対応するシンボリックエラー名です。値は一意です。
- `SQLSTATE` は、`ERROR_NUMBER` 値に対応する `SQLSTATE` 値です。値は必ずしも一意ではありません。

たとえば、`ERROR_NUMBER` が 1050 の場合、`ERROR_NAME` は `ER_TABLE_EXISTS_ERROR` で、`SQLSTATE` は `42S01` です。

エラーイベントのサマリー情報の例:

```
mysql> SELECT *
FROM performance_schema.events_errors_summary_global_by_error
WHERE SUM_ERROR_RAISED <> 0\G
***** 1. row *****
ERROR_NUMBER: 1064
ERROR_NAME: ER_PARSE_ERROR
SQL_STATE: 42000
SUM_ERROR_RAISED: 1
SUM_ERROR_HANDLED: 0
FIRST_SEEN: 2016-06-28 07:34:02
LAST_SEEN: 2016-06-28 07:34:02
***** 2. row *****
ERROR_NUMBER: 1146
ERROR_NAME: ER_NO_SUCH_TABLE
SQL_STATE: 42S02
SUM_ERROR_RAISED: 2
SUM_ERROR_HANDLED: 0
FIRST_SEEN: 2016-06-28 07:34:05
LAST_SEEN: 2016-06-28 07:36:18
***** 3. row *****
ERROR_NUMBER: 1317
ERROR_NAME: ER_QUERY_INTERRUPTED
SQL_STATE: 70100
SUM_ERROR_RAISED: 1
```

```
SUM_ERROR_HANDLED: 0  
FIRST_SEEN: 2016-06-28 11:01:49  
LAST_SEEN: 2016-06-28 11:01:49
```

各エラーサマリーテーブルには、テーブルでのエラーの集計方法を示す 1 つ以上のグループ化カラムがあります:

- `events_errors_summary_by_account_by_error` には、`USER`、`HOST` および `ERROR_NUMBER` カラムがあります。各行には、特定のアカウント (ユーザーとホストの組合せ) およびエラーのイベントが要約されます。
- `events_errors_summary_by_host_by_error` には、`HOST` カラムおよび `ERROR_NUMBER` カラムがあります。各行には、特定のホストおよびエラーのイベントが要約されます。
- `events_errors_summary_by_thread_by_error` には、`THREAD_ID` カラムおよび `ERROR_NUMBER` カラムがあります。各行には、特定のスレッドおよびエラーのイベントが要約されます。
- `events_errors_summary_by_user_by_error` には、`USER` カラムおよび `ERROR_NUMBER` カラムがあります。各行には、特定のユーザーおよびエラーのイベントが要約されます。
- `events_errors_summary_global_by_error` には `ERROR_NUMBER` カラムがあります。各行には、特定のエラーのイベントが要約されます。

各エラーサマリーテーブルには、集計値を含む次のサマリーカラムがあります:

- `SUM_ERROR_RAISED`
このカラムには、エラーが発生した回数が集計されます。
- `SUM_ERROR_HANDLED`
このカラムは、SQL 例外ハンドラによってエラーが処理された回数を集計します。
- `FIRST_SEEN`, `LAST_SEEN`
エラーが最初に表示された時間と最後に表示された時間を示すタイムスタンプ。

各エラーサマリーテーブルの `NULL` 行は、インストゥルメントされたエラーの範囲外にあるすべてのエラーの統計を集計するために使用されます。たとえば、MySQL Server エラーが `M` から `N` の範囲内にあり、その範囲内には番号 `Q` でエラーが発生した場合、エラーは `NULL` 行に集計されます。`NULL` の行は、`ERROR_NUMBER=0`、`ERROR_NAME=NULL` および `SQLSTATE=NULL` を含む行です。

エラーサマリーテーブルには、次のインデックスがあります:

- `events_errors_summary_by_account_by_error`:
 - 主キー (`USER`, `HOST`, `ERROR_NUMBER`)
- `events_errors_summary_by_host_by_error`:
 - 主キー (`HOST`, `ERROR_NUMBER`)
- `events_errors_summary_by_thread_by_error`:
 - 主キー (`THREAD_ID`, `ERROR_NUMBER`)
- `events_errors_summary_by_user_by_error`:
 - 主キー (`USER`, `ERROR_NUMBER`)
- `events_errors_summary_global_by_error`:
 - 主キー (`ERROR_NUMBER`)

`TRUNCATE TABLE` はエラーサマリーテーブルに対して許可されています。これには次の効果があります:

- アカウント、ホストまたはユーザーごとに集計されないサマリーテーブルの場合、切捨てによって、行が削除されるのではなく、サマリーカラムがゼロまたは `NULL` にリセットされます。

- アカウント、ホストまたはユーザー別に集計されたサマリーテーブルの場合、切捨てによって、接続のないアカウント、ホストまたはユーザーの行が削除され、残りの行のサマリーカラムがゼロまたは `NULL` にリセットされます。

また、アカウント、ホスト、ユーザーまたはスレッド別に集計された各エラーサマリーテーブルは、依存する接続テーブルの切捨てまたは `events_errors_summary_global_by_error` の切捨てによって暗黙的に切り捨てられます。詳細は、[セクション27.12.8「パフォーマンススキーマ接続テーブル」](#)を参照してください。

27.12.18.12 ステータス変数サマリーテーブル

パフォーマンススキーマは、[セクション27.12.15「パフォーマンススキーマのステータス変数のテーブル」](#)で説明されているテーブルでステータス変数情報を使用できるようにします。また、ここで説明するように、集計されたステータス変数情報をサマリーテーブルで使用できるようにします。各ステータス変数サマリーテーブルには、テーブルでのステータス値の集計方法を示す1つ以上のグループ化カラムがあります：

- `status_by_account` には、アカウント別にステータス変数を要約するための `USER`、`HOST` および `VARIABLE_NAME` のカラムがあります。
- `status_by_host` には、クライアントの接続元のホスト別にステータス変数を要約するための `HOST` および `VARIABLE_NAME` のカラムがあります。
- `status_by_user` には、クライアントユーザー名別にステータス変数を要約するための `USER` および `VARIABLE_NAME` のカラムがあります。

各ステータス変数サマリーテーブルには、集計値を含む次のサマリーカラムがあります：

- `VARIABLE_VALUE`

アクティブセッションおよび終了したセッションの集計ステータス変数値。

ステータス変数サマリーテーブルには、次のインデックスがあります：

- `status_by_account`:
 - 主キー (`USER`, `HOST`, `VARIABLE_NAME`)
- `status_by_host`:
 - 主キー (`HOST`, `VARIABLE_NAME`)
- `status_by_user`:
 - 主キー (`USER`, `VARIABLE_NAME`)

これらのテーブルの「account」の意味は、`mysql` システムデータベースの MySQL 付与テーブルでの意味と似ていますが、用語はユーザーとホストの値の組合せを意味します。権限付与テーブルの場合、アカウントのホスト部分はパターンにできますが、「パフォーマンススキーマ」テーブルの場合、ホスト値は常に特定のパターン以外のホスト名になります。

アカウントステータスは、セッションの終了時に収集されます。セッションステータスカウンタは、グローバルステータスカウンタおよび対応するアカウントステータスカウンタに追加されます。アカウント統計が収集されない場合、ホストおよびユーザーのステータスが収集されると、セッションステータスがホストおよびユーザーのステータスに追加されます。

`performance_schema_accounts_size`、`performance_schema_hosts_size` および `performance_schema_users_size` システム変数がそれぞれ0に設定されている場合、アカウント、ホストおよびユーザーの統計は収集されません。

パフォーマンススキーマは、次のようなステータス変数サマリーテーブルの `TRUNCATE TABLE` をサポートしています。すべての場合、アクティブセッションのステータスは影響を受けません：

- `status_by_account`: 終了したセッションからユーザーおよびホストのステータスにアカウントステータスを集計し、アカウントステータスをリセットします。
- `status_by_host`: 終了したセッションから集計ホストのステータスをリセットします。

- `status_by_user`: 終了したセッションから集計済ユーザーステータスをリセットします。

`FLUSH STATUS` は、すべてのアクティブセッションのセッションステータスをグローバルステータス変数に追加し、すべてのアクティブセッションのステータスをリセットし、切断されたセッションから集計されたアカウント、ホストおよびユーザーステータス値をリセットします。

27.12.19 パフォーマンススキーマのその他のテーブル

次のセクションでは、先述のセクションで説明したテーブルカテゴリに収まらないテーブルについて説明します。

- `error_log`: エラーログに書き込まれた最新のイベント。
- `host_cache`: 内部ホストキャッシュからの情報
- `keyring_keys`: MySQL キーリングのキーのメタデータ。
- `log_status`: バックアップのためのサーバーログに関する情報。
- `performance_timers`: 使用可能なイベントタイマー
- `threads`: サーバースレッドに関する情報
- `tls_channel_status`: 接続インターフェースの TLS コンテキストプロパティ。
- `user_defined_functions`: コンポーネント、プラグイン、または `CREATE FUNCTION` ステートメントによって登録されたユーザー定義関数。

27.12.19.1 error_log テーブル

MySQL サーバーが保持するログのうち、診断メッセージを書き込むエラーログです ([セクション5.4.2「エラーログ」](#)を参照)。通常、サーバーはサーバーホスト上のファイルまたはシステムログサービスに診断を書き込みます。MySQL 8.0.22 の時点では、エラーログの構成に応じて、サーバーは最新のエラーイベントをパフォーマンススキーマの `error_log` テーブルに書き込むこともできます。したがって、`error_log` テーブルに対する `SELECT` 権限を付与すると、クライアントおよびアプリケーションは SQL クエリーを使用してエラーログの内容にアクセスできるため、DBA はサーバーホストでのファイルシステムへの直接アクセスを許可せずにログにアクセスできます。

`error_log` テーブルでは、より構造化されたカラムに基づいてフォーカスされたクエリーがサポートされます。また、より自由形式の分析をサポートするエラーメッセージの全文も含まれています。

テーブルの実装では、固定サイズのメモリー内リングバッファが使用され、必要に応じて古いイベントが自動的に破棄され、新しいイベント用の領域が確保されます。

`error_log` コンテンツの例:

```
mysql> SELECT * FROM performance_schema.error_log\G
***** 1. row *****
LOGGED: 2020-08-06 09:25:00.338624
THREAD_ID: 0
  PRIO: System
ERROR_CODE: MY-010116
SUBSYSTEM: Server
  DATA: mysqld (mysqld 8.0.23) starting as process 96344
***** 2. row *****
LOGGED: 2020-08-06 09:25:00.363521
THREAD_ID: 1
  PRIO: System
ERROR_CODE: MY-013576
SUBSYSTEM: InnoDB
  DATA: InnoDB initialization has started.
...
***** 65. row *****
LOGGED: 2020-08-06 09:25:02.936146
THREAD_ID: 0
  PRIO: Warning
ERROR_CODE: MY-010068
SUBSYSTEM: Server
```

```
DATA: CA certificate /var/mysql/sslinfo/cacert.pem is self signed.
...
***** 89. row *****
LOGGED: 2020-08-06 09:25:03.112801
THREAD_ID: 0
  PRIO: System
ERROR_CODE: MY-013292
SUBSYSTEM: Server
  DATA: Admin interface ready for connections, address: '127.0.0.1' port: 33062
```

`error_log` テーブルには次のカラムがあります。説明に示されているように、`DATA` カラム以外のすべてのカラムは、[セクション5.4.2.3「エラーイベントフィールド」](#)で説明されている基礎となるエラーイベント構造のフィールドに対応します。

• LOGGED

マイクロ秒精度のイベントタイムスタンプ。`LOGGED` は、エラーイベントの `time` フィールドに対応していますが、次のような違いが考えられます:

- エラーログの `time` 値は、`log_timestamps` システム変数の設定に従って表示されます。[初期起動時のロギング出力形式](#) を参照してください。
- `LOGGED` カラムには、`TIMESTAMP` データ型を使用して値が格納されます。値は UTC で格納されますが、現在のセッションタイムゾーンで取得されると表示されます。[セクション11.2.2「DATE、DATETIME、およびTIMESTAMP型」](#) を参照してください。

エラーログファイルに表示されているのと同じタイムゾーンで `LOGGED` 値を表示するには、最初にセッションタイムゾーンを次のように設定します:

```
SET @@session.time_zone = @@global.log_timestamps;
```

`log_timestamps` 値が UTC で、システムに名前付きタイムゾーンサポートがインストールされていない場合 ([セクション5.1.15「MySQL Serverでのタイムゾーンのサポート」](#) を参照)、タイムゾーンを次のように設定します:

```
SET @@session.time_zone = '+00:00';
```

• THREAD_ID

MySQL スレッド ID。`THREAD_ID` は、エラーイベントの `thread` フィールドに対応します。

パフォーマンススキーマ内では、`error_log` テーブルの `THREAD_ID` カラムは `threads` テーブルの `PROCESLIST_ID` カラムにもっとも似ています:

- フォアグラウンドスレッドの場合、`THREAD_ID` および `PROCESLIST_ID` は接続識別子を表します。これは、`INFORMATION_SCHEMA PROCESLIST` テーブルの `ID` カラムに表示される値と同じで、`SHOW PROCESLIST` 出力の `id` カラムに表示され、スレッド内の `CONNECTION_ID()` 関数によって返されます。
- バックグラウンドスレッドの場合、`THREAD_ID` は 0、`PROCESLIST_ID` は NULL です。

`error_log` 以外の「多数のパフォーマンススキーマ」テーブルには `THREAD_ID` という名前のカラムがありますが、これらのテーブルでは、`THREAD_ID` カラムはパフォーマンススキーマによって内部的に割り当てられた値です。

• PRIO

イベントの優先度。許可される値は `System`、`Error`、`Warning`、`Note` です。`PRIO` カラムは、エラーイベントの `label` フィールドに基づいており、それ自体は基礎となる数値 `prio` フィールド値に基づいています。

• ERROR_CODE

数値のイベントエラーコード。`ERROR_CODE` は、エラーイベントの `error_code` フィールドに対応します。

• SUBSYSTEM

イベントが発生したサブシステム。`SUBSYSTEM` は、エラーイベントの `subsystem` フィールドに対応します。

• DATA

エラーイベントのテキスト表現。この値の形式は、`error_log` 行を生成するログシンクコンポーネントによって生成される形式によって異なります。たとえば、ログシンクが `log_sink_internal` または `log_sink_json` の場合、`DATA` の値はそれぞれ従来の形式または JSON 形式のエラーイベントを表します。(セクション5.4.2.9「エラーログ出力形式」を参照してください。)

エラーログを再構成して、`error_log` テーブルに行を提供するログシンクコンポーネントを変更できます。また、異なるシンクによって異なる出力形式が生成されるため、異なるタイミングで `error_log` テーブルに書き込まれる行に異なる `DATA` 形式を設定できます。

`error_log` テーブルには次のインデックスがあります:

- 主キー (LOGGED)
- (THREAD_ID) のインデックス
- (PRIO) のインデックス
- (ERROR_CODE) のインデックス
- (SUBSYSTEM) のインデックス

`TRUNCATE TABLE` は、`error_log` テーブルに対して許可されていません。

`error_log` テーブルの実装および構成

パフォーマンススキーマ `error_log` テーブルは、エラーログにフォーマットされたエラーイベントを書き込むだけでなく、テーブルに書き込むエラーログシンクコンポーネントによって移入されます。ログシンクによるパフォーマンススキーマのサポートには、次の2つの部分があります:

- ログシンクは、発生時に新しいエラーイベントを `error_log` テーブルに書き込むことができます。
- ログシンクは、以前に書き込まれたエラーメッセージを抽出するためのパーサーを提供できます。これにより、サーバーインスタンスは前のインスタンスによってエラーログファイルに書き込まれたメッセージを読み取って、`error_log` テーブルに格納できます。前のインスタンスによって停止中に書き込まれたメッセージは、停止が発生した理由の診断に役立つ場合があります。

現在、従来の形式の `log_sink_internal` および JSON 形式の `log_sink_json` シンクは、`error_log` テーブルへの新しいイベントの書き込みをサポートし、以前に書き込まれたエラーログファイルを読み取るパーサーを提供しています。

`log_error_services` システム変数は、エラーロギングを有効にするログコンポーネントを制御します。この値は、エラーイベントが発生したときに左から右の順に実行されるログフィルタおよびログシンクコンポーネントのパイプラインです。`log_error_services` の値は、次のように `error_log` テーブルへの移入に関連します:

- 起動時に、サーバーは `log_error_services` 値を調べ、次の条件を満たす左端のログシンクから選択します:

- `error_log` テーブルをサポートし、パーサーを提供するシンク。
- 存在しない場合、`error_log` テーブルをサポートするがパーサーを提供しないシンク。

これらの条件を満たすログシンクがない場合、`error_log` テーブルは空のままです。それ以外の場合、シンクがパーサーを提供し、以前に書き込まれたエラーログファイルを検出できるようにすると、サーバーはシンクパーサーを使用してファイルの最後の部分を読み取り、そこに含まれる古いイベントをテーブルに書き込みます。その後、シンクは新しいエラーイベントを発生時にテーブルに書き込みます。

- 実行時に、`log_error_services` の値が変更されると、サーバーは再度その値を調べ、今度は `error_log` テーブルをサポートする左端に有効なログシンクを探して、パーサーを提供するかどうかを確認します。

そのようなログシンクが存在しない場合、追加のエラーイベントは `error_log` テーブルに書き込まれません。それ以外の場合、新しく構成されたシンクは、新しいエラーイベントを発生時にテーブルに書き込みます。

エラーログに書き込まれる出力に影響する構成は、`error_log` テーブルの内容に影響します。これには、冗長性、メッセージ抑制、メッセージフィルタリングなどの設定が含まれます。また、起動時に以前のログファイルから読み取られた情報にも適用されます。たとえば、冗長性が低い状態で構成された以前のサーバーインスタンス中に書き込まれ

なかったメッセージは、冗長性が高い状態で構成された現在のインスタンスによってファイルが読み取られた場合は使用できなくなります。

`error_log` テーブルは固定サイズのメモリ内リングバッファのビューで、新しいイベント用の領域を確保するために、必要に応じて古いイベントが自動的に破棄されます。次のテーブルに示すように、いくつかのステータス変数は、進行中の `error_log` 操作に関する情報を提供します。

ステータス変数	意味
<code>Error_log_buffered_bytes</code>	テーブルで使用されるバイト数
<code>Error_log_buffered_events</code>	テーブルに存在するイベント
<code>Error_log_expired_events</code>	テーブルから破棄されたイベント
<code>Error_log_latest_write</code>	テーブルへの最終書き込み時間

27.12.19.2 firewall_groups テーブル

`firewall_groups` テーブルは、MySQL Enterprise Firewall のインメモリーデータキャッシュのビューを提供します。登録されているファイアウォールグループプロファイルの名前と動作モードを一覧表示します。ファイアウォールデータの永続的な記憶域を提供する `mysql.firewall_groups` システムテーブルとともに使用されます。[MySQL Enterprise Firewall テーブル](#) を参照してください。

`firewall_groups` テーブルには、次のカラムがあります:

- **NAME**
グループプロファイル名。
- **MODE**
プロファイルの現在の操作モード。許可されるモード値は、`OFF`、`DETECTING`、`PROTECTING` および `RECORDING` です。意味の詳細は、[ファイアウォール操作の概念](#) を参照してください。
- **USERHOST**
プロファイルが `RECORDING` モードの場合に使用される、グループプロファイルのトレーニングアカウント。値は、`NULL` または `user_name@host_name` 形式の `NULL` 以外のアカウントです:
 - 値が `NULL` の場合、ファイアウォールはグループのメンバーであるアカウントから受信したステートメントの許可リストルールを記録します。
 - 値が `NULL` 以外の場合、ファイアウォールは、指定されたアカウント (グループのメンバーである必要があります) から受信したステートメントの許可リストルールのみを記録します。

`firewall_groups` テーブルには次のインデックスがあります:

- なし

`TRUNCATE TABLE` は、`firewall_groups` テーブルに対して許可されていません。

`firewall_groups` テーブルが MySQL 8.0.23 に追加されました。

27.12.19.3 firewall_group_allowlist テーブル

`firewall_group_allowlist` テーブルは、MySQL Enterprise Firewall のインメモリーデータキャッシュのビューを提供します。登録されているファイアウォールグループプロファイルの許可リストルールを一覧表示します。ファイアウォールデータの永続的な記憶域を提供する `mysql.firewall_group_allowlist` システムテーブルとともに使用されます。[MySQL Enterprise Firewall テーブル](#) を参照してください。

`firewall_group_allowlist` テーブルには、次のカラムがあります:

- **NAME**

グループプロファイル名。

- **RULE**

プロファイルの許容可能なステートメントパターンを示す正規化されたステートメント。プロファイル許可リストは、そのルールの和集合です。

`firewall_group_allowlist` テーブルには次のインデックスがあります:

- なし

`TRUNCATE TABLE` は、`firewall_group_allowlist` テーブルに対して許可されていません。

`firewall_group_allowlist` テーブルが MySQL 8.0.23 に追加されました。

27.12.19.4 `firewall_membership` テーブル

`firewall_membership` テーブルは、MySQL Enterprise Firewall のインメモリーデータキャッシュのビューを提供します。登録済のファイアウォールグループプロファイルのメンバー (アカウント) がリストされます。ファイアウォールデータの永続的な記憶域を提供する `mysql.firewall_membership` システムテーブルとともに使用されます。[MySQL Enterprise Firewall テーブル](#) を参照してください。

`firewall_membership` テーブルには、次のカラムがあります:

- **GROUP_ID**

グループプロファイル名。

- **MEMBER_ID**

プロファイルのメンバーであるアカウントの名前。

`firewall_membership` テーブルには次のインデックスがあります:

- なし

`TRUNCATE TABLE` は、`firewall_membership` テーブルに対して許可されていません。

`firewall_membership` テーブルが MySQL 8.0.23 に追加されました。

27.12.19.5 `host_cache` テーブル

MySQL サーバーは、クライアントホスト名および IP アドレス情報を含むインメモリーホストキャッシュを保持し、ドメインネームシステム (DNS) のルックアップを回避するために使用されます。`host_cache` テーブルには、このキャッシュの内容が公開されます。`host_cache_size` システム変数は、ホストキャッシュのサイズと `host_cache` テーブルのサイズを制御します。ホストキャッシュの操作および構成については、[セクション5.1.12.3「DNS ルックアップとホストキャッシュ」](#) を参照してください。

`host_cache` テーブルではホストキャッシュの内容が公開されるため、`SELECT` ステートメントを使用して調べることができます。これは、接続の問題の原因の診断に役立つことがあります。

`host_cache` テーブルにはこれらのカラムがあります。

- **IP**

文字列で表された、サーバーに接続しているクライアントの IP アドレス。

- **HOST**

そのクライアント IP の解決済みの DNS ホスト名、または名前が不明な場合は `NULL`。

- **HOST_VALIDATED**

クライアント IP に対し、IP からホスト名、ホスト名から IP の DNS 解決が正常に実行されたかどうか。
`HOST_VALIDATED` が `YES` の場合、DNS への追加コールを回避できるように、IP に対応するホスト名として `HOST` カラムが使用されます。`HOST_VALIDATED` が `NO` である間、DNS 解決は、最終的に有効な結果または永続エラーのいずれかで完了するまで、接続試行ごとに試行されます。この情報により、サーバーは一時的な DNS 障害中に不正または欠落したホスト名のキャッシュを回避でき、クライアントに永久的な影響を及ぼす可能性があります。

- `SUM_CONNECT_ERRORS`

「ブロック」とみなされる接続エラーの数 (`max_connect_errors` システム変数に対して評価される)。プロトコルハンドシェイクエラーのみがカウントされ、検証 (`HOST_VALIDATED = YES`) に合格したホストに対してのみカウントされます。

特定のホストの `SUM_CONNECT_ERRORS` が `max_connect_errors` の値に達すると、そのホストからの新しい接続はブロックされます。ホストがブロックされていない間にホストからの複数の接続が同時に試行される可能性があるため、`SUM_CONNECT_ERRORS` 値は `max_connect_errors` 値を超えることがあります。これらのいずれかまたはすべてに障害が発生し、`SUM_CONNECT_ERRORS` が `max_connect_errors` の値を超える可能性があります。

`max_connect_errors` が 200 で、特定のホストの `SUM_CONNECT_ERRORS` が 199 であるとします。10 クライアントがそのホストから同時に接続しようとした場合、`SUM_CONNECT_ERRORS` が 200 に達していないため、これらはブロックされません。5 つのクライアントでブロッキングエラーが発生した場合、`SUM_CONNECT_ERRORS` の値は 204 になり、クライアントごとに増加します。接続試行が開始されたときの `SUM_CONNECT_ERRORS` の値が 200 に達していないため、他の 5 つのクライアントは成功し、ブロックされません。`SUM_CONNECT_ERRORS` が 200 に達した後に開始するホストからの新しい接続はブロックされます。

- `COUNT_HOST_BLOCKED_ERRORS`

`SUM_CONNECT_ERRORS` が `max_connect_errors` システム変数の値を超えたため、ブロックされた接続の数。

- `COUNT_NAMEINFO_TRANSIENT_ERRORS`

IP からホスト名への DNS 解決時の一時的なエラーの数。

- `COUNT_NAMEINFO_PERMANENT_ERRORS`

IP からホスト名への DNS 解決時の永続的なエラーの数。

- `COUNT_FORMAT_ERRORS`

ホスト名形式エラーの数。MySQL では、`1.2.example.com` など、名前の最初のコンポーネントが完全に数値であるホスト名に対して、`mysql.user` システムテーブルの `Host` カラム値の照合は実行されません。代わりにクライアント IP アドレスが使用されます。この種類の照合が行われない理由については、[セクション 6.2.4「アカウント名の指定」](#)を参照してください。

- `COUNT_ADDRINFO_TRANSIENT_ERRORS`

ホスト名から IP への逆引き DNS 解決時の一時的なエラーの数。

- `COUNT_ADDRINFO_PERMANENT_ERRORS`

ホスト名から IP への逆引き DNS 解決時の永続的なエラーの数。

- `COUNT_FCRDNS_ERRORS`

Forward-confirmed reverse DNS エラーの数。これらのエラーは、IP からホスト名、ホスト名から IP の DNS 解決で、クライアントの発信元の IP アドレスに一致しない IP アドレスが生成された場合に発生します。

- `COUNT_HOST_ACL_ERRORS`

クライアントホストからの接続がユーザーに許可されていないために発生したエラーの数。そのような場合、サーバーは `ER_HOST_NOT_PRIVILEGED` を返し、ユーザー名やパスワードも要求しません。

- `COUNT_NO_AUTH_PLUGIN_ERRORS`

使用できない認証プラグインのリクエストによるエラーの数。プラグインが使用できない可能性があるのは、たとえば、それがロードされていないか、ロードの試みに失敗した場合です。

- [COUNT_AUTH_PLUGIN_ERRORS](#)

認証プラグインによって報告されるエラーの数。

認証プラグインは、障害の原因を示すために、さまざまなエラーコードを報告することがあります。エラーの種類に応じて、これらのいずれかのカラムが増分されます。[COUNT_AUTHENTICATION_ERRORS](#)、[COUNT_AUTH_PLUGIN_ERRORS](#)、[COUNT_HANDSHAKE_ERRORS](#)。新しいリターンコードは、既存のプラグイン API へのオプションの拡張です。不明または予期しないプラグインエラーは [COUNT_AUTH_PLUGIN_ERRORS](#) カラムにカウントされます。

- [COUNT_HANDSHAKE_ERRORS](#)

有線プロトコルレベルで検出されたエラーの数。

- [COUNT_PROXY_USER_ERRORS](#)

プロキシユーザー A が存在しない別のユーザー B にプロキシ設定されているときに検出されたエラーの数。

- [COUNT_PROXY_USER_ACL_ERRORS](#)

プロキシユーザー A が、存在するが A が [PROXY](#) 権限を持たない別のユーザー B にプロキシされたときに検出されたエラーの数。

- [COUNT_AUTHENTICATION_ERRORS](#)

失敗した認証によって発生したエラーの数。

- [COUNT_SSL_ERRORS](#)

SSL の問題によるエラーの数。

- [COUNT_MAX_USER_CONNECTIONS_ERRORS](#)

ユーザーごとの接続割り当てを超えることによって発生したエラーの数。 [セクション6.2.20「アカウントリソース制限の設定」](#) を参照してください。

- [COUNT_MAX_USER_CONNECTIONS_PER_HOUR_ERRORS](#)

時間あたりのユーザーごとの接続割り当てを超えることによって発生したエラーの数。 [セクション6.2.20「アカウントリソース制限の設定」](#) を参照してください。

- [COUNT_DEFAULT_DATABASE_ERRORS](#)

デフォルトのデータベースに関連するエラーの数。たとえば、データベースが存在しないか、ユーザーにデータベースへのアクセス権がありません。

- [COUNT_INIT_CONNECT_ERRORS](#)

[init_connect](#) システム変数値内のステートメントの実行の失敗によって発生したエラーの数。

- [COUNT_LOCAL_ERRORS](#)

サーバー実装にローカルで、ネットワーク、認証、または承認に関連しないエラーの数。たとえば、メモリー不足状況はこのカテゴリに収まります。

- [COUNT_UNKNOWN_ERRORS](#)

このテーブルのほかのカラムによって報告されない、ほかの不明なエラーの数。新しいエラー条件を報告する必要があり、[host_cache](#) テーブルの下位互換性および構造を保持する必要がある場合に備えて、このカラムは将来の使用のために予約されています。

- [FIRST_SEEN](#)

IP カラム内のクライアントから確認された最初の接続の試みのタイムスタンプ。

- [LAST_SEEN](#)

IP カラムのクライアントから見た最新の接続試行のタイムスタンプ。

- [FIRST_ERROR_SEEN](#)

IP カラム内のクライアントから確認された最初のエラーのタイムスタンプ。

- [LAST_ERROR_SEEN](#)

IP カラムでクライアントから検出された最新のエラーのタイムスタンプ。

[host_cache](#) テーブルには次のインデックスがあります:

- 主キー (IP)
- (HOST) のインデックス

[TRUNCATE TABLE](#) は [host_cache](#) テーブルに対して許可されています。テーブルに対する [DROP](#) 権限が必要です。テーブルを切り捨てると、ホストキャッシュがフラッシュされます。これは、[ホストキャッシュのフラッシュ](#) で説明されている効果があります。

27.12.19.6 keyring_keys テーブル

MySQL Server は、内部サーバーコンポーネントおよびプラグインが機密情報を安全に格納して後で取得できるようにするキーリングをサポートしています。 [セクション6.4.4「MySQL キーリング」](#) を参照してください。

MySQL 8.0.16 では、[keyring_keys](#) テーブルはキーリング内のキーのメタデータを公開します。キーメタデータには、キー ID、キー所有者およびバックエンドキー ID が含まれます。 [keyring_keys](#) テーブルでは、キーの内容などの機密キーリングデータは公開されません。

[keyring_keys](#) テーブルには、次のカラムがあります:

- [KEY_ID](#)
キー識別子。
- [KEY_OWNER](#)
キーの所有者。
- [BACKEND_KEY_ID](#)
キーリングバックエンドによってキーに使用される ID。

[keyring_keys](#) テーブルにインデックスがありません。

[TRUNCATE TABLE](#) は、[keyring_keys](#) テーブルに対して許可されていません。

27.12.19.7 log_status テーブル

[log_status](#) テーブルには、オンラインバックアップツールが、コピープロセス中に必要なログファイルをロックせずにコピーできるようにする情報が示されます。

[log_status](#) テーブルをクエリーすると、サーバーはロギングおよび関連する管理上の変更を、テーブルに移入するのに十分な時間ブロックしてから、リソースを解放します。 [log_status](#) テーブルは、ソースバイナリログと [gtid_executed](#) レコード内のコピー先のオンラインバックアップ、および各レプリケーションチャンネルのリレーログを通知します。また、個々のストレージエンジンに関連する情報 (最後のログシーケンス番号 (LSN) や、[InnoDB](#) ストレージエンジンで最後に取得されたチェックポイントの LSN など) も提供します。

[log_status](#) テーブルには、次のカラムがあります:

- [SERVER_UUID](#)

このサーバーインスタンスのサーバー UUID。これは、読み取り専用システム変数 `server_uuid` の生成された一意の値です。

- LOCAL

次のキーを持つ単一の JSON オブジェクトとして提供される、ソースからのログ位置状態情報:

<code>binary_log_file</code>	現在のバイナリログファイルの名前。
<code>binary_log_position</code>	<code>log_status</code> テーブルへのアクセス時の現在のバイナリログの位置。
<code>gtid_executed</code>	<code>log_status</code> テーブルへのアクセス時のグローバルサーバー変数 <code>gtid_executed</code> の現在の値。この情報は、 <code>binary_log_file</code> および <code>binary_log_position</code> のキーと一致します。

- REPLICATION

チャンネルの JSON 配列。それぞれに次の情報が含まれます:

<code>channel_name</code>	レプリケーションチャンネルの名前。デフォルトのレプリケーションチャンネル名は空の文字列 (「」) です。
<code>relay_log_file</code>	レプリケーションチャンネルの現在のリレーログファイルの名前。
<code>relay_log_pos</code>	<code>log_status</code> テーブルへのアクセス時の現在のリレーログの位置。

- STORAGE_ENGINES

個々のストレージエンジンからの関連情報。適用可能なストレージエンジンごとに 1 つの鍵を持つ JSON オブジェクトとして提供されます。

`log_status` テーブルにインデックスがありません。

`log_status` テーブルにアクセスするには、`BACKUP_ADMIN` 権限および `SELECT` 権限が必要です。

`TRUNCATE TABLE` は、`log_status` テーブルに対して許可されていません。

27.12.19.8 performance_timers テーブル

`performance_timers` テーブルは使用可能なイベントタイマーを示します。

```
mysql> SELECT * FROM performance_schema.performance_timers;
+-----+-----+-----+
| TIMER_NAME | TIMER_FREQUENCY | TIMER_RESOLUTION | TIMER_OVERHEAD |
+-----+-----+-----+
| CYCLE      | 2389029850      | 1                | 72              |
| NANOSECOND | 1000000000      | 1                | 112             |
| MICROSECOND | 1000000         | 1                | 136             |
| MILLISECOND | 1036           | 1                | 168             |
+-----+-----+-----+
```

特定のタイマーに関連付けられた値が `NULL` の場合、そのタイマーはプラットフォームでサポートされていません。イベントタイミングがどのように行われるかの説明については、[セクション 27.4.1 「パフォーマンススキーマイベントタイミング」](#) を参照してください。

`performance_timers` テーブルにはこれらのカラムがあります。

- `TIMER_NAME`

タイマー名。

- `TIMER_FREQUENCY`

秒あたりのタイマーユニットの数。サイクルタイマーの場合、頻度は一般に CPU 速度に関連します。たとえば、2.4GHz プロセッサを搭載するシステムでは、`CYCLE` は 2400000000 に近い可能性があります。

- `TIMER_RESOLUTION`

タイマー値が増加するタイマーユニット数を示します。タイマーの分解能が 10 の場合、その値は毎回 10 ずつ増加します。

- **TIMER_OVERHEAD**

特定のタイマーで 1 つのタイミングを取得するためのオーバーヘッドの最小サイクル数。パフォーマンススキーマは、初期化時にタイマーを 20 回呼び出し、最小値を選択することによって、この値を決定します。インストールメンテーションは、各イベントの開始と終了でタイマーを呼び出すため、実際の合計のオーバーヘッドはこの量の 2 倍になります。タイマーコードは、時間付きイベントにのみ呼び出されるため、このオーバーヘッドは時間なしイベントには適用されません。

`performance_timers` テーブルには次のインデックスがあります:

- なし

`TRUNCATE TABLE` は、`performance_timers` テーブルに対して許可されていません。

27.12.19.9 processlist テーブル

MySQL プロセスリストには、サーバー内で実行されているスレッドのセットによって現在実行されている操作が示されます。`processlist` テーブルは、プロセス情報のソースです。このテーブルと他のソースの比較は、[プロセス情報のソース](#) を参照してください。

`processlist` テーブルは直接クエリーすることができます。`PROCESS` 権限を持っている場合は、他のユーザーに属するスレッドも含めて、すべてのスレッドを表示できます。それ以外の場合 (`PROCESS` 権限なし)、非匿名ユーザーは自分のスレッドに関する情報にはアクセスできますが、他のユーザーのスレッドにはアクセスできず、匿名ユーザーはスレッド情報にアクセスできません。

注記

`performance_schema_show_processlist` システム変数が有効になっている場合、`processlist` テーブルは `SHOW PROCESSLIST` ステートメントの基礎となる代替実装の基礎としても機能します。詳細は、このセクションの後半のを参照してください。

`processlist` テーブルには、各サーバープロセスの行が含まれます:

```
mysql> SELECT * FROM performance_schema.processlist\G
***** 1. row *****
  ID: 5
  USER: event_scheduler
  HOST: localhost
  DB: NULL
  COMMAND: Daemon
  TIME: 137
  STATE: Waiting on empty queue
  INFO: NULL
***** 2. row *****
  ID: 9
  USER: me
  HOST: localhost:58812
  DB: NULL
  COMMAND: Sleep
  TIME: 95
  STATE:
  INFO: NULL
***** 3. row *****
  ID: 10
  USER: me
  HOST: localhost:58834
  DB: test
  COMMAND: Query
  TIME: 0
  STATE: executing
  INFO: SELECT * FROM performance_schema.processlist
...
```

`processlist` テーブルには、次のカラムがあります:

- ID

接続識別子。これは、`SHOW PROCESSLIST` ステートメントの `Id` カラムに表示される値と同じで、パフォーマンススキーマ `threads` テーブルの `PROCESSLIST_ID` カラムに表示され、スレッド内の `CONNECTION_ID()` 関数によって返されます。

- USER

このステートメントを発行した MySQL ユーザー。 `system user` の値は、遅延行ハンドラスレッド、レプリカホストで使用される I/O または SQL スレッドなど、タスクを内部的に処理するためにサーバーによって起動される非クライアントスレッドを指します。 `system user` の場合、`Host` カラムにホストが指定されていません。 `unauthenticated user` は、クライアント接続に関連付けられたが、クライアントユーザーの認証がまだ行われていないスレッドを参照します。 `event_scheduler` は、スケジュールされたイベントをモニターするスレッドを指します (セクション25.4「イベントスケジューラの使用」を参照)。

注記

`system user` の `USER` 値は、`SYSTEM_USER` 権限とは異なります。前者は内部スレッドを指定します。後者は、システムユーザーと通常のユーザーアカウントカテゴリを区別します (セクション6.2.11「アカウントカテゴリ」を参照)。

- HOST

ステートメントを発行するクライアントのホスト名 (ホストがない `system user` を除く)。TCP/IP 接続のホスト名は、`host_name:client_port` 形式でレポートされるため、どのクライアントが何を実行しているかを簡単に判別できます。

- DB

スレッドのデフォルトデータベース。選択されていない場合は `NULL`。

- COMMAND

スレッドがクライアントのかわりに実行しているコマンドのタイプ。セッションがアイドル状態の場合は `Sleep`。スレッドコマンドの説明については、セクション8.14「サーバスレッド (プロセス) 情報の確認」を参照してください。このカラムの値は、クライアント/サーバープロトコルの `COM_xxx` コマンドと `Com_xxx` ステータス変数に対応します。セクション5.1.10「サーバスレッドステータス変数」を参照してください

- TIME

スレッドが現在の状態になってからの秒数。レプリカ SQL スレッドの場合、この値は、最後にレプリケートされたイベントのタイムスタンプとレプリカホストのリアルタイムの間の秒数です。セクション17.2.3「レプリケーションスレッド」を参照してください。

- STATE

スレッドが行なっていることを示すアクション、イベント、または状態。 `STATE` の値の詳細は、セクション8.14「サーバスレッド (プロセス) 情報の確認」を参照してください。

ほとんどの状態がきわめてすばやい操作に対応します。スレッドの状態が何秒間も特定の状態にとどまっている場合は、調査が必要な問題が発生している可能性があります。

- INFO

スレッドが実行しているステートメント。ステートメントを実行していない場合は `NULL`。このステートメントは、サーバーに送信されるステートメント、またはこのステートメントがほかのステートメントを実行する場合は、もっとも内側のステートメントである可能性があります。たとえば、`CALL` ステートメントが `SELECT` ステートメントを実行しているストアドプロシージャを実行する場合、`INFO` 値には `SELECT` ステートメントが表示されます。

`processlist` テーブルには次のインデックスがあります:

- 主キー (ID)

`TRUNCATE TABLE` は、`processlist` テーブルに対して許可されていません。

前述のように、`performance_schema_show_processlist` システム変数が有効になっている場合、`processlist` テーブルは他のプロセス情報ソースの代替実装の基礎となります:

- `SHOW PROCESSLIST` ステートメント。
- `mysqladmin processlist` コマンド (`SHOW PROCESSLIST` ステートメントを使用)。

デフォルトの `SHOW PROCESSLIST` 実装は、グローバル `mutex` を保持しながら、スレッドマネージャ内からアクティブスレッド間で繰り返されます。これは、特にビジー状態のシステムではパフォーマンスに悪影響を及ぼします。代替の `SHOW PROCESSLIST` 実装は、パフォーマンススキーマ `processlist` テーブルに基づいています。この実装は、スレッドマネージャではなくパフォーマンススキーマからアクティブなスレッドデータをクエリーするため、`mutex` は必要ありません。

MySQL の構成は、次のように `processlist` テーブルの内容に影響します:

- 最小限必要な構成:
 - MySQL サーバーは、スレッドインストゥルメンテーションを有効にして構成および構築する必要があります。これはデフォルトで `true` で、`DISABLE_PSI_THREAD_CMake` オプションを使用して制御されます。
 - サーバーの起動時にパフォーマンススキーマを有効にする必要があります。これはデフォルトで `true` で、`performance_schema` システム変数を使用して制御されます。

この構成が満たされると、`performance_schema_show_processlist` は代替の `SHOW PROCESSLIST` 実装を有効または無効にします。最小構成が満たされていない場合、`processlist` テーブル (したがって `SHOW PROCESSLIST`) はすべてのデータを返すわけではありません。

- 推奨構成:
 - 一部のスレッドを無視しないようにするには:
 - `performance_schema_max_thread_instances` システム変数はデフォルトのままにするか、少なくとも `max_connections` システム変数と同じ大きさに設定します。
 - `performance_schema_max_thread_classes` システム変数はデフォルトのままにします。
 - 一部の `STATE` カラムの値が空にならないようにするには、`performance_schema_max_stage_classes` システム変数をデフォルトのままにします。

これらの構成パラメータのデフォルトは `-1` で、パフォーマンススキーマはサーバーの起動時にそれらのサイズを自動設定します。パラメータが指定どおりに設定されている場合、`processlist` テーブル (および `SHOW PROCESSLIST`) は完全なプロセス情報を生成します。

前述の構成パラメータは、`processlist` テーブルの内容に影響します。ただし、特定の構成では、`processlist` の内容は `performance_schema_show_processlist` 設定の影響を受けません。

代替プロセスリストの実装は、MySQL クライアント/サーバープロトコルの `INFORMATION_SCHEMA_PROCESSLIST` テーブルまたは `COM_PROCESS_INFO` コマンドには適用されません。

27.12.19.10 スレッドテーブル

`threads` テーブルは各サーバスレッドの行を格納します。各行にはスレッドに関する情報が含まれ、監視および履歴イベントロギングが有効かどうかを示されます:

```
mysql> SELECT * FROM performance_schema.threads\G
***** 1. row *****
  THREAD_ID: 1
    NAME: thread/sql/main
   TYPE: BACKGROUND
PROCESSLIST_ID: NULL
PROCESSLIST_USER: NULL
PROCESSLIST_HOST: NULL
PROCESSLIST_DB: NULL
PROCESSLIST_COMMAND: NULL
PROCESSLIST_TIME: 80284
PROCESSLIST_STATE: NULL
```



```
PROCESSLIST_INFO: NULL
PARENT_THREAD_ID: NULL
  ROLE: NULL
  INSTRUMENTED: YES
  HISTORY: YES
CONNECTION_TYPE: NULL
  THREAD_OS_ID: 489803
  RESOURCE_GROUP: SYS_default
...
***** 4. row *****
  THREAD_ID: 51
  NAME: thread/sql/one_connection
  TYPE: FOREGROUND
  PROCESSLIST_ID: 34
  PROCESSLIST_USER: isabella
  PROCESSLIST_HOST: localhost
  PROCESSLIST_DB: performance_schema
PROCESSLIST_COMMAND: Query
PROCESSLIST_TIME: 0
PROCESSLIST_STATE: Sending data
PROCESSLIST_INFO: SELECT * FROM performance_schema.threads
PARENT_THREAD_ID: 1
  ROLE: NULL
  INSTRUMENTED: YES
  HISTORY: YES
CONNECTION_TYPE: SSL/TLS
  THREAD_OS_ID: 755399
  RESOURCE_GROUP: USR_default
...
```

パフォーマンススキーマが初期化されると、存在するスレッドに基づいて `threads` テーブルが生成されます。その後、サーバーがスレッドを作成するたびに新しい行が追加されます。

新しいスレッドの `INSTRUMENTED` および `HISTORY` カラムの値は、`setup_actors` テーブルの内容によって決まります。`setup_actors` テーブルを使用してこれらのカラムを制御する方法の詳細は、[セクション27.4.6「スレッドによる事前フィルタリング」](#)を参照してください。

スレッドの終了時に、`threads` テーブルからの行の削除が行われます。クライアントセッションに関連付けられたスレッドでは、セッションの終了時に削除が行われます。クライアントで自動再接続が有効になっていて、切断後にセッションが再接続された場合、セッションは異なる `PROCESSLIST_ID` 値を持つ `threads` テーブルの新しい行に関連付けられます。新しいスレッドの `INSTRUMENTED` および `HISTORY` の初期値は、元のスレッドの値と異なる場合があります。一方、`setup_actors` テーブルは変更された可能性があり、行の初期化後に元のスレッドの `INSTRUMENTED` または `HISTORY` の値が変更された場合、変更は新しいスレッドに引き継がれません。

スレッドモニタリング (スレッドによって実行されるイベントがインストゥルメントされているかどうか) および履歴イベントロギングを有効または無効にできます。新しいフォアグラウンドスレッドの `INSTRUMENTED` および `HISTORY` の初期値を制御するには、`setup_actors` テーブルを使用します。既存のスレッドのこれらの側面を制御するには、`threads` テーブルの行の `INSTRUMENTED` および `HISTORY` カラムを設定します。(スレッドモニタリングおよび履歴イベントロギングが発生する条件の詳細は、`INSTRUMENTED` および `HISTORY` のカラムの説明を参照してください。)

接頭辞が `PROCESSLIST_` の名前と他のプロセス情報ソースとの `threads` テーブルのカラムの比較は、[プロセス情報のソース](#)を参照してください。

重要

`threads` テーブル以外のスレッド情報ソースの場合、他のユーザーのスレッドに関する情報は、現在のユーザーが `PROCESS` 権限を持っている場合にのみ表示されます。これは `threads` テーブルには当てはまりません。テーブルの `SELECT` 権限を持つすべてのユーザーに、すべての行が表示されます。`threads` テーブルにアクセスして他のユーザーのスレッドを表示できないようにするユーザーには、そのユーザーに対する `SELECT` 権限を付与しないでください。

`threads` テーブルにはこれらのカラムがあります。

- `THREAD_ID`

一意のスレッド識別子。

- **NAME**

サーバーのスレッドインストゥルメンテーションコードに関連付けられている名前。たとえば、`thread/sql/one_connection` はユーザー接続の処理を担当するコード内のスレッド関数に対応し、`thread/sql/main` はサーバーの `main()` 関数を表します。

- **TYPE**

FOREGROUND または **BACKGROUND** のスレッドの種類。ユーザー接続スレッドはフォアグラウンドスレッドです。内部サーバーアクティビティーに関連付けられているスレッドはバックグラウンドスレッドです。たとえば、内部 **InnoDB** スレッド、レプリカに情報を送信する「binlog dump」スレッド、レプリケーション I/O スレッドおよび SQL スレッドなどです。

- **PROCESSLIST_ID**

フォアグラウンドスレッド (ユーザー接続に関連付けられている) の場合、これは接続識別子です。これは、**INFORMATION_SCHEMA PROCESSLIST** テーブルの **ID** カラムに表示される値と同じで、**SHOW PROCESSLIST** 出力の **Id** カラムに表示され、スレッド内の **CONNECTION_ID()** 関数によって返されます。

バックグラウンドスレッド (ユーザー接続に関連付けられていない) の場合、**PROCESSLIST_ID** は **NULL** であるため、値は一意ではありません。

- **PROCESSLIST_USER**

フォアグラウンドスレッドに関連付けられているユーザー、バックグラウンドスレッドの場合は **NULL**。

- **PROCESSLIST_HOST**

フォアグラウンドスレッドに関連付けられているクライアントのホスト名、バックグラウンドスレッドの場合は **NULL**。

INFORMATION_SCHEMA PROCESSLIST テーブルの **HOST** カラムまたは **SHOW PROCESSLIST** 出力の **Host** カラムとは異なり、**PROCESSLIST_HOST** カラムには TCP/IP 接続のポート番号は含まれません。パフォーマンススキーマからこの情報を取得するには、ソケットインストゥルメンテーションを有効にし (デフォルトでは有効になっていません)、**socket_instances** テーブルを調べます:

```
mysql> SELECT NAME, ENABLED, TIMED
      FROM performance_schema.setup_instruments
      WHERE NAME LIKE 'wait/io/socket%';
+-----+-----+-----+
| NAME                                | ENABLED | TIMED |
+-----+-----+-----+
| wait/io/socket/sql/server_tcpip_socket | NO      | NO    |
| wait/io/socket/sql/server_unix_socket  | NO      | NO    |
| wait/io/socket/sql/client_connection   | NO      | NO    |
+-----+-----+-----+
3 rows in set (0.01 sec)

mysql> UPDATE performance_schema.setup_instruments
      SET ENABLED='YES'
      WHERE NAME LIKE 'wait/io/socket%';
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0

mysql> SELECT * FROM performance_schema.socket_instances\G
***** 1. row *****
EVENT_NAME: wait/io/socket/sql/client_connection
OBJECT_INSTANCE_BEGIN: 140612577298432
THREAD_ID: 31
SOCKET_ID: 53
IP: ::ffff:127.0.0.1
PORT: 55642
STATE: ACTIVE
...
```

- **PROCESSLIST_DB**

スレッドのデフォルトデータベース。選択されていない場合は **NULL**。

- **PROCESSLIST_COMMAND**

フォアグラウンドスレッドの場合、スレッドがクライアントのかわりに実行しているコマンドのタイプ、またはセッションがアイドル状態の場合は **Sleep**。スレッドコマンドの説明については、[セクション8.14「サーバースレッド \(プロセス\) 情報の確認」](#)を参照してください。このコラムの値は、クライアント/サーバープロトコルの **COM_xxx** コマンドと **Com_xxx** ステータス変数に対応します。[セクション5.1.10「サーバーステータス変数」](#)を参照してください

バックグラウンドスレッドはクライアントのかわりにコマンドを実行しないため、このコラムは **NULL** である可能性があります。

- **PROCESSLIST_TIME**

スレッドが現在の状態になってからの秒数。レプリカ SQL スレッドの場合、この値は、最後にレプリケートされたイベントのタイムスタンプとレプリカホストのリアルタイムの間の秒数です。[セクション17.2.3「レプリケーションスレッド」](#)を参照してください。

- **PROCESSLIST_STATE**

スレッドが行なっていることを示すアクション、イベント、または状態。**PROCESSLIST_STATE** 値の説明については、[セクション8.14「サーバースレッド \(プロセス\) 情報の確認」](#)を参照してください。値が **NULL** の場合、スレッドはアイドルクライアントセッションに対応しているか、スレッドが実行している作業がステージでインストールメントされていません。

ほとんどの状態がきわめてすばやい操作に対応します。スレッドが何秒間も特定の状態にとどまっている場合は、問題が発生している可能性があり、調査が必要です。

- **PROCESSLIST_INFO**

スレッドが実行しているステートメント。ステートメントを実行していない場合は **NULL**。このステートメントは、サーバーに送信されるステートメント、またはこのステートメントがほかのステートメントを実行する場合は、もっとも内側のステートメントである可能性があります。たとえば、**CALL** ステートメントが **SELECT** ステートメントを実行しているストアードプロシージャを実行する場合、**PROCESSLIST_INFO** 値には **SELECT** ステートメントが示されます。

- **PARENT_THREAD_ID**

このスレッドがサブスレッド (別のスレッドによって生成される) である場合、これは生成されるスレッドの **THREAD_ID** 値です。

- **ROLE**

使用されません。

- **INSTRUMENTED**

スレッドによって実行されるイベントがインストールメントされるかどうか。値は **YES** または **NO** です。

- フォアグラウンドスレッドでは、初期 **INSTRUMENTED** 値は、スレッドに関連付けられているユーザーアカウントが **setup_actors** テーブル内の任意の行に一致するかどうかによって決定されます。照合は **PROCESSLIST_USER** および **PROCESSLIST_HOST** コラムの値に基づきます。

スレッドがサブスレッドを生成すると、そのサブスレッドに対して作成された **threads** テーブルの行に対して照合が再度行われます。

- バックグラウンドスレッドの場合、**INSTRUMENTED** はデフォルトで **YES** です。バックグラウンドスレッドに関連付けられたユーザーはないため、**setup_actors** は参照されません。
- どのスレッドでも、スレッドの有効期間の間にその **INSTRUMENTED** 値が変更されることがあります。

スレッドによって実行されるイベントのモニタリングが行われる場合、これらのことが当てはまる必要があります。

- **setup_consumers** テーブル内の **thread_instrumentation** コンシューマは **YES** である必要があります。

- `threads.INSTRUMENTED` カラムは `YES` である必要があります。
- 監視は、`setup_instruments` テーブルで `ENABLED` カラムが `YES` に設定されているインストゥルメントから生成されたスレッドイベントに対してのみ行われます。

- **HISTORY**

スレッドの履歴イベントをログに記録するかどうか。値は `YES` または `NO` です。

- フォアグラウンドスレッドの場合、初期 `HISTORY` 値は、スレッドに関連付けられたユーザーアカウントが `setup_actors` テーブルのいずれかの行と一致するかどうかによって決まります。照合は `PROCESLIST_USER` および `PROCESLIST_HOST` カラムの値に基づきます。

スレッドがサブスレッドを生成すると、そのサブスレッドに対して作成された `threads` テーブルの行に対して照合が再度行われます。

- バックグラウンドスレッドの場合、`HISTORY` はデフォルトで `YES` です。バックグラウンドスレッドに関連付けられたユーザーがないため、`setup_actors` は参照されません。
- どのスレッドでも、その `HISTORY` 値はスレッドの存続期間中に変更できます。

スレッドの履歴イベントロギングを実行するには、次のことが当てはまる必要があります：

- `setup_consumers` テーブルの適切な履歴関連コンシューマを有効にする必要があります。たとえば、`events_waits_history` および `events_waits_history_long` テーブルの待機イベントロギングでは、対応する `events_waits_history` および `events_waits_history_long` コンシューマが `YES` である必要があります。
- `threads.HISTORY` カラムは `YES` である必要があります。
- ロギングは、`setup_instruments` テーブルで `ENABLED` カラムが `YES` に設定されているインストゥルメントから生成されたスレッドイベントに対してのみ発生します。

- **CONNECTION_TYPE**

接続の確立に使用されるプロトコル、またはバックグラウンドスレッド用の `NULL`。許可される値は、`TCP/IP` (暗号化なしで確立された TCP/IP 接続)、`SSL/TLS` (暗号化で確立された TCP/IP 接続)、`Socket` (Unix ソケットファイル接続)、`Named Pipe` (Windows 名前付きパイプ接続) および `Shared Memory` (Windows 共有メモリー接続) です。

- **THREAD_OS_ID**

基礎となるオペレーティングシステムで定義されているスレッドまたはタスク識別子 (存在する場合)：

- MySQL スレッドが存続期間中に同じオペレーティングシステムスレッドに関連付けられている場合、`THREAD_OS_ID` にはオペレーティングシステムスレッド ID が含まれます。
- MySQL スレッドが存続期間中に同じオペレーティングシステムスレッドに関連付けられていない場合、`THREAD_OS_ID` には `NULL` が含まれます。これは、スレッドプールプラグインが使用されている場合のユーザーセッションに一般的です (セクション 5.6.3 「MySQL Enterprise Thread Pool」を参照)。

Windows の場合、`THREAD_OS_ID` はプロセスエクスプローラ (<https://technet.microsoft.com/en-us/sysinternals/bb896653.aspx>) に表示されるスレッド ID に対応します。

Linux の場合、`THREAD_OS_ID` は `gettid()` 関数の値に対応します。この値は、たとえば、`perf` または `ps -L` コマンドを使用するか、`proc` ファイルシステム (`/proc/[pid]/task/[tid]`) で公開されます。詳細は、`perf-stat(1)`、`ps(1)`、および `proc(5)` のマニュアルページを参照してください。

- **RESOURCE_GROUP**

リソースグループラベル。リソースグループが現在のプラットフォームまたはサーバー構成でサポートされていない場合、この値は `NULL` です (リソースグループの制限を参照)。

`threads` テーブルには次のインデックスがあります：

- 主キー (`THREAD_ID`)

- (NAME) のインデックス
- (PROCESSLIST_ID) のインデックス
- (PROCESSLIST_USER、PROCESSLIST_HOST) のインデックス
- (PROCESSLIST_HOST) のインデックス
- (THREAD_OS_ID) のインデックス
- (RESOURCE_GROUP) のインデックス

TRUNCATE TABLE は、threads テーブルに対して許可されていません。

27.12.19.11 tls_channel_status テーブル

接続インタフェース TLS プロパティはサーバーの起動時に設定され、実行時に ALTER INSTANCE RELOAD TLS ステートメントを使用して更新できます。サーバー側のランタイム構成および暗号化された接続の監視を参照してください。

tls_channel_status テーブル (MySQL 8.0.21 で使用可能) は、接続インタフェース TLS プロパティに関する情報を提供します:

```
mysql> SELECT * FROM performance_schema.tls_channel_status\G
***** 1. row *****
CHANNEL: mysql_main
PROPERTY: Enabled
VALUE: Yes
***** 2. row *****
CHANNEL: mysql_main
PROPERTY: ssl_accept_renegotiates
VALUE: 0
***** 3. row *****
CHANNEL: mysql_main
PROPERTY: Ssl_accepts
VALUE: 2
...
***** 29. row *****
CHANNEL: mysql_admin
PROPERTY: Enabled
VALUE: No
***** 30. row *****
CHANNEL: mysql_admin
PROPERTY: ssl_accept_renegotiates
VALUE: 0
***** 31. row *****
CHANNEL: mysql_admin
PROPERTY: Ssl_accepts
VALUE: 0
...
```

tls_channel_status テーブルには、次のカラムがあります:

- CHANNEL

TLS プロパティ行が適用される接続インタフェースの名前。mysql_main と mysql_admin は、それぞれメイン接続インタフェースと管理接続インタフェースのチャンネル名です。様々なインタフェースの詳細は、[セクション 5.1.12.1「接続インタフェース」](#) を参照してください。

- PROPERTY

TLS プロパティ名。Enabled プロパティの行は、インタフェース全体のステータスを示し、インタフェースとそのステータスはそれぞれ CHANNEL カラムと VALUE カラムで指定されます。その他のプロパティ名は、特定の TLS プロパティを示します。これらは多くの場合、TLS 関連のステータス変数の名前に対応します。

- VALUE

TLS プロパティ値。

このテーブルで公開されるプロパティは固定されず、各チャンネルで実装されるインストールメンテーションに依存します。

チャンネルごとに、`PROPERTY` 値が `Enabled` の行はチャンネルが暗号化された接続をサポートしているかどうかを示し、その他のチャンネル行は TLS コンテキストプロパティを示します:

- `mysql_main` の場合、`Enabled` プロパティは `yes` または `no` で、メインインタフェースが暗号化された接続をサポートするかどうかを示します。その他のチャンネル行には、メインインタフェースの TLS コンテキストプロパティが表示されます。

メインインタフェースでは、次のステートメントを使用して同様のステータス情報を取得できます:

```
SHOW GLOBAL STATUS LIKE 'current_tls%';  
SHOW GLOBAL STATUS LIKE 'ssl%';
```

- `mysql_admin` では、管理インタフェースが有効になっていない場合、または有効になっているが暗号化された接続をサポートしていない場合、`Enabled` プロパティは `no` です。インタフェースが有効で、暗号化された接続をサポートしている場合、`Enabled` は `yes` です。

`Enabled` が `yes` の場合、他の `mysql_admin` 行は、そのインタフェースにデフォルト以外の TLS パラメータ値が構成されている場合にのみ、管理インタフェース TLS コンテキストのチャンネルプロパティを示します。(これは、`admin_tls_xxx` または `admin_ssl_xxx` システム変数がデフォルトとは異なる値に設定されている場合です。) それ以外の場合、管理インタフェースはメインインタフェースと同じ TLS コンテキストを使用します。

`tls_channel_status` テーブルには次のインデックスがあります:

- なし

`TRUNCATE TABLE` は、`tls_channel_status` テーブルに対して許可されていません。

27.12.19.12 user_defined_functions テーブル

`user_defined_functions` テーブルには、コンポーネントまたはプラグインによって自動的に、または `CREATE FUNCTION` ステートメントによって手動で登録された各ユーザー定義関数 (UDF) の行が含まれています。テーブルの行を追加または削除する操作の詳細は、[セクション 5.7.1 「ユーザー定義関数のインストールおよびアンインストール」](#) を参照してください。

`user_defined_functions` テーブルには、次のカラムがあります:

- `UDF_NAME`
SQL ステートメントで参照される UDF 名。関数が `CREATE FUNCTION` ステートメントによって登録され、アンロード処理中の場合、値は `NULL` です。
- `UDF_RETURN_TYPE`
UDF 戻り値のタイプ。値は、`int`、`decimal`、`real`、`char` または `row` のいずれかです。
- `UDF_TYPE`
UDF タイプ。値は、`function` (スカラー) または `aggregate` のいずれかです。
- `UDF_LIBRARY`
実行可能 UDF コードを含むライブラリファイルの名前。このファイルは、`plugin_dir` システム変数で指定されたディレクトリにあります。UDF が `CREATE FUNCTION` ステートメントではなくコンポーネントまたはプラグインによって登録された場合、値は `NULL` です。
- `UDF_USAGE_COUNT`
現在の UDF 使用数。これは、ステートメントが現在 UDF にアクセスしているかどうかを判断するために使用されます。

`user_defined_functions` テーブルには次のインデックスがあります:

- 主キー (`UDF_NAME`)

TRUNCATE TABLE は、user_defined_functions テーブルに対して許可されていません。

27.13 パフォーマンススキーマオプションおよび変数リファレンス

表 27.4 「パフォーマンススキーマ変数リファレンス」

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
performance_schema	はい	はい	はい		グローバル	いいえ
Performance_schema_accounts_lost				はい	グローバル	いいえ
performance_schema_accounts_lost	はい	はい	はい		グローバル	いいえ
Performance_schema_cond_classes_lost				はい	グローバル	いいえ
Performance_schema_cond_instances_lost				はい	グローバル	いいえ
performance-schema-consumer-events-stages-current	はい	はい				
performance-schema-consumer-events-stages-history	はい	はい				
performance-schema-consumer-events-stages-history-long	はい	はい				
performance-schema-consumer-events-statements-current	はい	はい				
performance-schema-consumer-events-statements-history	はい	はい				
performance-schema-consumer-events-statements-history-long	はい	はい				
performance-schema-consumer-events-transactions-current	はい	はい				
performance-schema-consumer-	はい	はい				

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
events- transactions- history						
performance- schema- consumer- events- transactions- history-long	はい	はい				
performance- schema- consumer- events-waits- current	はい	はい				
performance- schema- consumer- events-waits- history	はい	はい				
performance- schema- consumer- events-waits- history-long	はい	はい				
performance- schema- consumer- global- instrumentation	はい	はい				
performance- schema- consumer- statements- digest	はい	はい				
performance- schema- consumer- thread- instrumentation	はい	はい				
Performance_schema_digest_lost				はい	グローバル	いいえ
performance_schema_digests_size	はい	はい	はい		グローバル	いいえ
performance_schema_events_stages_history_long	はい	はい	はい		グローバル	いいえ
performance_schema_events_stages_history_size	はい	はい	はい		グローバル	いいえ
performance_schema_events_statements_history_long	はい	はい	はい		グローバル	いいえ
performance_schema_events_statements_history_size	はい	はい	はい		グローバル	いいえ
performance_schema_events_transactions_history_long	はい	はい	はい		グローバル	いいえ
performance_schema_events_transactions_history_size	はい	はい	はい		グローバル	いいえ
performance_schema_events_waits_history_long	はい	はい	はい		グローバル	いいえ
performance_schema_events_waits_history_size	はい	はい	はい		グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
Performance_schema_file_classes_lost				はい	グローバル	いいえ
Performance_schema_file_handles_lost				はい	グローバル	いいえ
Performance_schema_file_instances_lost				はい	グローバル	いいえ
Performance_schema_hosts_lost				はい	グローバル	いいえ
performance_schema_hosts_size	はい	はい	はい		グローバル	いいえ
performance-schema-instrument	はい	はい				
Performance_schema_locker_lost				はい	グローバル	いいえ
performance_schema_max_cond_classes	はい	はい	はい		グローバル	いいえ
performance_schema_max_cond_instances	はい	はい	はい		グローバル	いいえ
performance_schema_max_digest_length	はい	はい	はい		グローバル	いいえ
performance_schema_max_file_classes	はい	はい	はい		グローバル	いいえ
performance_schema_max_file_handles	はい	はい	はい		グローバル	いいえ
performance_schema_max_file_instances	はい	はい	はい		グローバル	いいえ
performance_schema_max_memory_classes	はい	はい	はい		グローバル	いいえ
performance_schema_max_metadata_locks	はい	はい	はい		グローバル	いいえ
performance_schema_max_mutex_classes	はい	はい	はい		グローバル	いいえ
performance_schema_max_mutex_instances	はい	はい	はい		グローバル	いいえ
performance_schema_max_prepared_statements_instances	はい	はい	はい		グローバル	いいえ
performance_schema_max_program_instances	はい	はい	はい		グローバル	いいえ
performance_schema_max_rwlock_classes	はい	はい	はい		グローバル	いいえ
performance_schema_max_rwlock_instances	はい	はい	はい		グローバル	いいえ
performance_schema_max_socket_classes	はい	はい	はい		グローバル	いいえ
performance_schema_max_socket_instances	はい	はい	はい		グローバル	いいえ
performance_schema_max_stage_classes	はい	はい	はい		グローバル	いいえ
performance_schema_max_statement_classes	はい	はい	はい		グローバル	いいえ
performance_schema_max_statement_stack	はい	はい	はい		グローバル	いいえ
performance_schema_max_table_handles	はい	はい	はい		グローバル	いいえ
performance_schema_max_table_instances	はい	はい	はい		グローバル	いいえ
performance_schema_max_thread_classes	はい	はい	はい		グローバル	いいえ
performance_schema_max_thread_instances	はい	はい	はい		グローバル	いいえ
Performance_schema_memory_classes_lost				はい	グローバル	いいえ
Performance_schema_metadata_lock_lost				はい	グローバル	いいえ
Performance_schema_mutex_classes_lost				はい	グローバル	いいえ
Performance_schema_mutex_instances_lost				はい	グローバル	いいえ
Performance_schema_nested_statement_lost				はい	グローバル	いいえ
Performance_schema_prepared_statements_lost				はい	グローバル	いいえ
Performance_schema_program_lost				はい	グローバル	いいえ
Performance_schema_rwlock_classes_lost				はい	グローバル	いいえ
Performance_schema_rwlock_instances_lost				はい	グローバル	いいえ

名前	コマンド行	オプションファイル	システム変数	ステータス変数	変数スコープ	動的
Performance_schema_session_connect_attrs_lost				はい	グローバル	いいえ
performance_schema_session_connect_attrs_size	はい	はい	はい		グローバル	いいえ
performance_schema_setup_actor_size	はい	はい	はい		グローバル	いいえ
performance_schema_setup_object_size	はい	はい	はい		グローバル	いいえ
Performance_schema_socket_classes_lost				はい	グローバル	いいえ
Performance_schema_socket_instances_lost				はい	グローバル	いいえ
Performance_schema_stage_classes_lost				はい	グローバル	いいえ
Performance_schema_statement_classes_lost				はい	グローバル	いいえ
Performance_schema_table_handles_lost				はい	グローバル	いいえ
Performance_schema_table_instances_lost				はい	グローバル	いいえ
Performance_schema_thread_classes_lost				はい	グローバル	いいえ
Performance_schema_thread_instances_lost				はい	グローバル	いいえ
Performance_schema_users_lost				はい	グローバル	いいえ
performance_schema_users_size	はい	はい	はい		グローバル	いいえ

27.14 パフォーマンススキーマコマンドオプション

パフォーマンススキーマパラメータは、サーバー起動時にコマンド行またはオプションファイルに指定して、パフォーマンススキーマのインストゥルメントおよびコンシューマを構成できます。多くの場合に実行時構成も可能ですが ([セクション27.4「パフォーマンススキーマ実行時構成」](#)を参照)、実行時構成で、起動プロセス中にすでに初期化されているインストゥルメントに影響を与えるには遅すぎる場合は、起動時構成を使用する必要があります。

パフォーマンススキーマのコンシューマおよびインストゥルメントは、次の構文を使用して起動時に構成できます。追加の詳細については、[セクション27.3「パフォーマンススキーマ起動構成」](#)を参照してください。

- `--performance-schema-consumer-consumer_name=value`

パフォーマンススキーマコンシューマを構成します。 `setup_consumers` テーブル内のコンシューマ名は下線が使われますが、起動時に設定されたコンシューマでは、名前の中のダッシュと下線は同等です。各コンシューマを構成するためのオプションについては、このセクションの後半で詳しく説明します。

- `--performance-schema-instrument=instrument_name=value`

パフォーマンススキーマインストゥルメントを構成します。名前をパターンとして指定して、パターンに一致するインストゥルメントを構成できます。

次の項目は各コンシューマを構成します。

- `--performance-schema-consumer-events-stages-current=value`
`events-stages-current` コンシューマを構成します。
- `--performance-schema-consumer-events-stages-history=value`
`events-stages-history` コンシューマを構成します。
- `--performance-schema-consumer-events-stages-history-long=value`
`events-stages-history-long` コンシューマを構成します。
- `--performance-schema-consumer-events-statements-current=value`
`events-statements-current` コンシューマを構成します。
- `--performance-schema-consumer-events-statements-history=value`

`events-statements-history` コンシューマを構成します。

- `--performance-schema-consumer-events-statements-history-long=value`
`events-statements-history-long` コンシューマを構成します。
- `--performance-schema-consumer-events-transactions-current=value`
パフォーマンススキーマ `events-transactions-current` コンシューマを構成します。
- `--performance-schema-consumer-events-transactions-history=value`
パフォーマンススキーマ `events-transactions-history` コンシューマを構成します。
- `--performance-schema-consumer-events-transactions-history-long=value`
パフォーマンススキーマ `events-transactions-history-long` コンシューマを構成します。
- `--performance-schema-consumer-events-waits-current=value`
`events-waits-current` コンシューマを構成します。
- `--performance-schema-consumer-events-waits-history=value`
`events-waits-history` コンシューマを構成します。
- `--performance-schema-consumer-events-waits-history-long=value`
`events-waits-history-long` コンシューマを構成します。
- `--performance-schema-consumer-global-instrumentation=value`
`global-instrumentation` コンシューマを構成します。
- `--performance-schema-consumer-statements-digest=value`
`statements-digest` コンシューマを構成します。
- `--performance-schema-consumer-thread-instrumentation=value`
`thread-instrumentation` コンシューマを構成します。

27.15 パフォーマンススキーマシステム変数

パフォーマンススキーマは、構成情報を提供するいくつかのシステム変数を実装しています。

```
mysql> SHOW VARIABLES LIKE 'perf%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| performance_schema | ON |
| performance_schema_accounts_size | -1 |
| performance_schema_digests_size | 10000 |
| performance_schema_events_stages_history_long_size | 10000 |
| performance_schema_events_stages_history_size | 10 |
| performance_schema_events_statements_history_long_size | 10000 |
| performance_schema_events_statements_history_size | 10 |
| performance_schema_events_transactions_history_long_size | 10000 |
| performance_schema_events_transactions_history_size | 10 |
| performance_schema_events_waits_history_long_size | 10000 |
| performance_schema_events_waits_history_size | 10 |
| performance_schema_hosts_size | -1 |
| performance_schema_max_cond_classes | 80 |
| performance_schema_max_cond_instances | -1 |
| performance_schema_max_digest_length | 1024 |
| performance_schema_max_file_classes | 50 |
| performance_schema_max_file_handles | 32768 |
| performance_schema_max_file_instances | -1 |
| performance_schema_max_index_stat | -1 |
```

```

performance_schema_max_memory_classes      | 320 |
performance_schema_max_metadata_locks     | -1  |
performance_schema_max_mutex_classes      | 220 |
performance_schema_max_mutex_instances    | -1  |
performance_schema_max_prepared_statements_instances | -1  |
performance_schema_max_program_instances  | -1  |
performance_schema_max_rwlock_classes     | 40  |
performance_schema_max_rwlock_instances   | -1  |
performance_schema_max_socket_classes     | 10  |
performance_schema_max_socket_instances   | -1  |
performance_schema_max_sql_text_length    | 1024 |
performance_schema_max_stage_classes      | 150 |
performance_schema_max_statement_classes  | 192 |
performance_schema_max_statement_stack    | 10  |
performance_schema_max_table_handles      | -1  |
performance_schema_max_table_instances    | -1  |
performance_schema_max_table_lock_stat    | -1  |
performance_schema_max_thread_classes     | 50  |
performance_schema_max_thread_instances   | -1  |
performance_schema_session_connect_attrs_size | 512 |
performance_schema_setup_actors_size      | -1  |
performance_schema_setup_objects_size     | -1  |
performance_schema_users_size             | -1  |
+-----+-----+

```

パフォーマンススキーマシステム変数は、コマンド行またはオプションファイルでサーバーの起動時に設定でき、多くは実行時に設定できます。 [セクション27.13「パフォーマンススキーマオプションおよび変数リファレンス」](#)を参照してください。

パフォーマンススキーマは、明示的に設定されていない場合、サーバーの起動時にいくつかのパラメータの値のサイズを自動的に設定します。 詳細については、 [セクション27.3「パフォーマンススキーマ起動構成」](#)を参照してください。

パフォーマンススキーマシステム変数には次の意味があります。

- [performance_schema](#)

コマンド行形式	<code>--performance-schema[={OFF ON}]</code>
システム変数	performance_schema
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	ON

この変数の値は **ON** または **OFF** で、パフォーマンススキーマが有効にされているかどうかを示します。 デフォルト値は **ON** です。 サーバーの起動時に、この変数を値なし、またはそれを有効にする **ON** または **1** の値、またはそれを無効にする **OFF** または **0** の値で指定できます。

パフォーマンススキーマが無効になっていても、引き続き [global_variables](#)、[session_variables](#)、[global_status](#) および [session_status](#) テーブルにデータが移入されます。 これは、必要に応じて、[SHOW VARIABLES](#) および [SHOW STATUS](#) ステートメントの結果をこれらのテーブルから取得できるようにするために発生します。 また、パフォーマンススキーマは、一部のレプリケーションテーブルが無効になっている場合はそれらのテーブルにデータを移入します。

- [performance_schema_accounts_size](#)

コマンド行形式	<code>--performance-schema-accounts-size=#</code>
システム変数	performance_schema_accounts_size
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ

型	Integer
デフォルト値	-1 (自動スケーリングを示します。このリテラル値を割り当てないでください)
最小値	-1 (自動スケーリングを示します。このリテラル値を割り当てないでください)
最大値	1048576

`accounts` テーブルの行数。この変数が 0 の場合、パフォーマンススキーマは `accounts` テーブル内の接続統計情報または `status_by_account` テーブル内のステータス変数情報を保持しません。

- [performance_schema_digests_size](#)

コマンド行形式	<code>--performance-schema-digests-size=#</code>
システム変数	performance_schema_digests_size
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動サイズ設定を示します。このリテラル値を割り当てないでください)
最小値	-1
最大値	1048576

`events_statements_summary_by_digest` テーブル内の最大行数。この最大を超えたため、ダイジェストをインストールできない場合、パフォーマンススキーマは `Performance_schema_digest_lost` ステータス変数を増分します。

ステートメントダイジェストの詳細については、[セクション27.10「パフォーマンススキーマのステートメントダイジェストとサンプリング」](#)を参照してください。

- [performance_schema_error_size](#)

コマンド行形式	<code>--performance-schema-error-size=#</code>
システム変数	performance_schema_error_size
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	number of server error codes
最小値	0
最大値	1048576

インストールされたサーバーエラーコードの数。デフォルト値は、サーバーエラーコードの実際の数です。値は 0 から最大までの任意の場所に設定できますが、使用目的はデフォルト (すべてのエラーを計測する) または 0 (エラーを計測しない) に設定することです。

エラー情報はサマリーテーブルに集約されます。[セクション27.12.18.11「エラー要約テーブル」](#)を参照してください。インストールされていないエラーが発生した場合、発生した情報は各サマリーテーブルの NULL 行、つまり `ERROR_NUMBER=0`、`ERROR_NAME=NULL` および `SQLSTATE=NULL` のある行に集計されます。

- [performance_schema_events_stages_history_long_size](#)

コマンド行形式	<code>--performance-schema-events-stages-history-long-size=#</code>
システム変数	performance_schema_events_stages_history_long_size
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動サイズ設定を示します。このリテラル値を割り当てないでください)

[events_stages_history_long](#) テーブルの行数。

- [performance_schema_events_stages_history_size](#)

コマンド行形式	<code>--performance-schema-events-stages-history-size=#</code>
システム変数	performance_schema_events_stages_history_size
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動サイズ設定を示します。このリテラル値を割り当てないでください)

[events_stages_history](#) テーブルのスレッドあたりの行数。

- [performance_schema_events_statements_history_long_size](#)

コマンド行形式	<code>--performance-schema-events-statements-history-long-size=#</code>
システム変数	performance_schema_events_statements_history_long_size
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動サイズ設定を示します。このリテラル値を割り当てないでください)

[events_statements_history_long](#) テーブル内の行数。

- [performance_schema_events_statements_history_size](#)

コマンド行形式	<code>--performance-schema-events-statements-history-size=#</code>
システム変数	performance_schema_events_statements_history_size
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer

デフォルト値	-1 (自動サイズ設定を示します。このリテラル値を割り当てないでください)
--------	---------------------------------------

[events_statements_history](#) テーブル内のスレッドあたりの行数。

- [performance_schema_events_transactions_history_long_size](#)

コマンド行形式	<code>--performance-schema-events-transactions-history-long-size=#</code>
システム変数	performance_schema_events_transactions_history_long_size
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動サイズ設定を示します。このリテラル値を割り当てないでください)

[events_transactions_history_long](#) テーブルの行数。

- [performance_schema_events_transactions_history_size](#)

コマンド行形式	<code>--performance-schema-events-transactions-history-size=#</code>
システム変数	performance_schema_events_transactions_history_size
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動サイズ設定を示します。このリテラル値を割り当てないでください)

[events_transactions_history](#) テーブルのスレッド当たりの行数。

- [performance_schema_events_waits_history_long_size](#)

コマンド行形式	<code>--performance-schema-events-waits-history-long-size=#</code>
システム変数	performance_schema_events_waits_history_long_size
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動サイズ設定を示します。このリテラル値を割り当てないでください)

[events_waits_history_long](#) テーブル内の行数。

- [performance_schema_events_waits_history_size](#)

コマンド行形式	<code>--performance-schema-events-waits-history-size=#</code>
システム変数	performance_schema_events_waits_history_size
スコープ	グローバル

動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動サイズ設定を示します。このリテラル値を割り当てないでください)

events_waits_history テーブル内のスレッドあたりの行数。

- [performance_schema_hosts_size](#)

コマンド行形式	--performance-schema-hosts-size=#
システム変数	performance_schema_hosts_size
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動スケーリングを示します。このリテラル値を割り当てないでください)
最小値	-1 (自動スケーリングを示します。このリテラル値を割り当てないでください)
最大値	1048576

hosts テーブル内の行数。この変数が 0 の場合、パフォーマンススキーマは hosts テーブル内の接続統計情報または status_by_host テーブル内のステータス変数情報を保持しません。

- [performance_schema_max_cond_classes](#)

コマンド行形式	--performance-schema-max-cond-classes=#
システム変数	performance_schema_max_cond_classes
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値 (≥ 8.0.13)	100
デフォルト値 (≤ 8.0.12)	80
最小値	0
最大値 (≥ 8.0.12)	1024
最大値 (8.0.11)	256

条件インストールメントの最大数。この変数の設定方法および使用方法の詳細は、[セクション27.7「パフォーマンススキーマステータスモニタリング」](#)を参照してください。

- [performance_schema_max_cond_instances](#)

コマンド行形式	--performance-schema-max-cond-instances=#
システム変数	performance_schema_max_cond_instances
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ

型	Integer
デフォルト値	-1 (自動スケーリングを示します。このリテラル値を割り当てないでください)

インストールされた条件オブジェクトの最大数。この変数の設定方法および使用方法の詳細は、[セクション 27.7「パフォーマンススキーマステータスマニタリング」](#)を参照してください。

- [performance_schema_max_digest_length](#)

コマンド行形式	<code>--performance-schema-max-digest-length=#</code>
システム変数	performance_schema_max_digest_length
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1024
最小値	0
最大値	1048576

パフォーマンススキーマ内の正規化されたステートメントダイジェスト値を計算するためにステートメントごとに予約されるメモリの最大バイト数。この変数は [max_digest_length](#) に関連しています。[セクション 5.1.8「サーバーシステム変数」](#) でその変数の説明を参照してください。

メモリ使用に関する考慮事項など、ステートメントダイジェストの詳細は、[セクション 27.10「パフォーマンススキーマのステートメントダイジェストとサンプリング」](#)を参照してください。

- [performance_schema_max_digest_sample_age](#)

コマンド行形式	<code>--performance-schema-max-digest-sample-age=#</code>
システム変数	performance_schema_max_digest_sample_age
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	60
最小値	0
最大値	1048576

この変数は、[events_statements_summary_by_digest](#) テーブルのステートメントサンプリングに影響します。新しいテーブル行が挿入されると、行ダイジェスト値を生成したステートメントは、ダイジェストに関連付けられた現在のサンプルステートメントとして格納されます。その後、同じダイジェスト値を持つほかのステートメントがサーバーに表示されるときに、新しいステートメントを使用して現在のサンプルステートメントを置き換えるかどうか（つまり、再サンプリングするかどうか）を決定します。リサンプリングポリシーは、現在のサンプルステートメントと新しいステートメントの比較待機時間、およびオプションで現在のサンプルステートメントの経過時間に基づきます：

- 待機時間に基づくリサンプリング: 新しいステートメントの待機時間が現在のサンプルステートメントの待機時間よりも長い場合は、現在のサンプルステートメントになります。
- 年齢に基づくリサンプリング: [performance_schema_max_digest_sample_age](#) システム変数の値がゼロより大きく、現在のサンプルステートメントが何秒以上経過している場合、現在のステートメントは「古すぎま

す」)とみなされ、新しいステートメントで置き換えられます。これは、新しいステートメントの待機時間が現在のサンプルステートメントの待機時間より短い場合でも発生します。

ステートメントサンプリングの詳細は、[セクション27.10「パフォーマンススキーマのステートメントダイジェストとサンプリング」](#)を参照してください。

- [performance_schema_max_file_classes](#)

コマンド行形式	<code>--performance-schema-max-file-classes=#</code>
システム変数	performance_schema_max_file_classes
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	80
最小値	0
最大値 (≥ 8.0.12)	1024
最大値 (8.0.11)	256

ファイルインストゥルメントの最大数。この変数の設定方法および使用方法の詳細は、[セクション27.7「パフォーマンススキーマステータスマニタリング」](#)を参照してください。

- [performance_schema_max_file_handles](#)

コマンド行形式	<code>--performance-schema-max-file-handles=#</code>
システム変数	performance_schema_max_file_handles
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	32768

オープンしているファイルオブジェクトの最大数。この変数の設定方法および使用方法の詳細は、[セクション27.7「パフォーマンススキーマステータスマニタリング」](#)を参照してください。

[performance_schema_max_file_handles](#) の値は、[open_files_limit](#) の値より大きくしてください。[open_files_limit](#) は、サーバーがサポート可能なオープンファイルハンドルの最大数に影響し、[performance_schema_max_file_handles](#) はこれらのファイルハンドルのうちインストゥルメント可能な数に影響します。

- [performance_schema_max_file_instances](#)

コマンド行形式	<code>--performance-schema-max-file-instances=#</code>
システム変数	performance_schema_max_file_instances
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer

デフォルト値	-1 (自動スケーリングを示します。このリテラル値を割り当てないでください)
--------	--

インストールされるファイルオブジェクトの最大数。この変数の設定方法および使用方法の詳細は、[セクション27.7「パフォーマンススキーマステータスマニタリング」](#)を参照してください。

- [performance_schema_max_index_stat](#)

コマンド行形式	<code>--performance-schema-max-index-stat=#</code>
システム変数	performance_schema_max_index_stat
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動サイズ設定を示します。このリテラル値を割り当てないでください)

パフォーマンススキーマが統計情報を保持するインデックスの最大数。インデックス統計が失われるようにこの最大値を超えると、パフォーマンススキーマは [Performance_schema_index_stat_lost](#) ステータス変数を増分します。デフォルト値は、[performance_schema_max_table_instances](#) の値を使用して自動サイズ設定されます。

- [performance_schema_max_memory_classes](#)

コマンド行形式	<code>--performance-schema-max-memory-classes=#</code>
システム変数	performance_schema_max_memory_classes
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	450

メモリーインストールの最大数 この変数の設定方法および使用方法の詳細は、[セクション27.7「パフォーマンススキーマステータスマニタリング」](#)を参照してください。

- [performance_schema_max_metadata_locks](#)

コマンド行形式	<code>--performance-schema-max-metadata-locks=#</code>
システム変数	performance_schema_max_metadata_locks
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動スケーリングを示します。このリテラル値を割り当てないでください)

メタデータロックインストールの最大数。この値は、[metadata_locks](#) テーブルのサイズを制御します。メタデータロックを計測できないようにこの最大値を超えると、パフォーマンススキーマは [Performance_schema_metadata_lock_lost](#) ステータス変数を増分します。

- [performance_schema_max_mutex_classes](#)

コマンド行形式	<code>--performance-schema-max-mutex-classes=#</code>	4427
---------	---	------

システム変数	performance_schema_max_mutex_classes
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値 (≥ 8.0.12)	300
デフォルト値 (8.0.11)	250
最小値	0
最大値 (≥ 8.0.12)	1024
最大値 (8.0.11)	256

相互排他ロックインストゥルメントの最大数。この変数の設定方法および使用方法の詳細は、[セクション27.7「パフォーマンススキーマステータスマニタリング」](#)を参照してください。

- [performance_schema_max_mutex_instances](#)

コマンド行形式	<code>--performance-schema-max-mutex-instances=#</code>
システム変数	performance_schema_max_mutex_instances
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動スケーリングを示します。このリテラル値を割り当てないでください)

インストゥルメントされる相互排他ロックオブジェクトの最大数。この変数の設定方法および使用方法の詳細は、[セクション27.7「パフォーマンススキーマステータスマニタリング」](#)を参照してください。

- [performance_schema_max_prepared_statements_instances](#)

コマンド行形式	<code>--performance-schema-max-prepared-statements-instances=#</code>
システム変数	performance_schema_max_prepared_statements_instances
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動スケーリングを示します。このリテラル値を割り当てないでください)

[prepared_statements_instances](#) テーブルの最大行数。プリペアドステートメントを計測できないようにこの最大値を超えると、パフォーマンススキーマは [Performance_schema_prepared_statements_lost](#) ステータス変数を増分します。この変数の設定方法および使用方法の詳細は、[セクション27.7「パフォーマンススキーマステータスマニタリング」](#)を参照してください。

この変数のデフォルト値は、[max_prepared_stmt_count](#) システム変数の値に基づいて自動サイズ設定されます。

- [performance_schema_max_rwlock_classes](#)

コマンド行形式	<code>--performance-schema-max-rwlock-classes=#</code>
システム変数	performance_schema_max_rwlock_classes

スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	60
最小値	0
最大値 (≥ 8.0.12)	1024
最大値 (8.0.11)	256

rwlock インストゥルメントの最大数。この変数の設定方法および使用方法の詳細は、[セクション27.7「パフォーマンススキーマステータスマニタリング」](#)を参照してください。

- [performance_schema_max_program_instances](#)

コマンド行形式	<code>--performance-schema-max-program-instances=#</code>
システム変数	performance_schema_max_program_instances
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動スケーリングを示します。このリテラル値を割り当てないでください)

パフォーマンススキーマが統計情報を保持するストアプログラムの最大数。この最大値を超えると、パフォーマンススキーマは [Performance_schema_program_lost](#) ステータス変数を増分します。この変数の設定方法および使用方法の詳細は、[セクション27.7「パフォーマンススキーマステータスマニタリング」](#)を参照してください。

- [performance_schema_max_rwlock_instances](#)

コマンド行形式	<code>--performance-schema-max-rwlock-instances=#</code>
システム変数	performance_schema_max_rwlock_instances
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動スケーリングを示します。このリテラル値を割り当てないでください)

インストゥルメントされる rwlock オブジェクトの最大数。この変数の設定方法および使用方法の詳細は、[セクション27.7「パフォーマンススキーマステータスマニタリング」](#)を参照してください。

- [performance_schema_max_socket_classes](#)

コマンド行形式	<code>--performance-schema-max-socket-classes=#</code>
システム変数	performance_schema_max_socket_classes
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer

デフォルト値	10
最小値	0
最大値 (≥ 8.0.12)	1024
最大値 (8.0.11)	256

ソケットインストールの最大数。この変数の設定方法および使用方法の詳細は、[セクション27.7「パフォーマンススキーマステータスマニタリング」](#)を参照してください。

• [performance_schema_max_socket_instances](#)

コマンド行形式	<code>--performance-schema-max-socket-instances=#</code>
システム変数	performance_schema_max_socket_instances
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動スケーリングを示します。このリテラル値を割り当てないでください)

インストールされるソケットオブジェクトの最大数。この変数の設定方法および使用方法の詳細は、[セクション27.7「パフォーマンススキーマステータスマニタリング」](#)を参照してください。

• [performance_schema_max_sql_text_length](#)

コマンド行形式	<code>--performance-schema-max-sql-text-length=#</code>
システム変数	performance_schema_max_sql_text_length
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	1024
最小値	0
最大値	1048576

SQL ステートメントの格納に使用される最大バイト数。この値は、次のコラムに必要な記憶域に適用されます:

- [events_statements_current](#)、[events_statements_history](#) および [events_statements_history_long](#) ステートメントイベントテーブルの [SQL_TEXT](#) コラム。
- [events_statements_summary_by_digest](#) サマリーテーブルの [QUERY_SAMPLE_TEXT](#) コラム。

[performance_schema_max_sql_text_length](#) を超えるバイトは破棄され、コラムには表示されません。コラム内の初期バイト数が多くなった後にのみ異なるステートメントは区別できません。

[performance_schema_max_sql_text_length](#) 値を減らすとメモリー使用量は減少しますが、末尾のみが異なる場合は、より多くのステートメントが区別できなくなります。値を大きくするとメモリー使用量が増加しますが、長いステートメントを区別できます。

• [performance_schema_max_stage_classes](#)

コマンド行形式	<code>--performance-schema-max-stage-classes=#</code>
システム変数	performance_schema_max_stage_classes
スコープ	グローバル

動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値 (≥ 8.0.13)	175
デフォルト値 (≤ 8.0.12)	150
最小値	0
最大値 (≥ 8.0.12)	1024
最大値 (8.0.11)	256

ステージインストゥルメントの最大数。この変数の設定方法および使用方法の詳細は、[セクション27.7「パフォーマンススキーマステータスマニタリング」](#)を参照してください。

- [performance_schema_max_statement_classes](#)

コマンド行形式	<code>--performance-schema-max-statement-classes=#</code>
システム変数	performance_schema_max_statement_classes
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動サイズ設定を示します。このリテラル値を割り当てないでください)

ステートメントインストゥルメントの最大数。この変数の設定方法および使用方法の詳細は、[セクション27.7「パフォーマンススキーマステータスマニタリング」](#)を参照してください。

デフォルト値は、サーバー構築時に、クライアント/サーバープロトコルのコマンド数とサーバーでサポートされるSQLステートメントの種類の数に基づいて計算されます。

この変数は、それを0に設定して、すべてのステートメントインストゥルメンテーションを無効にし、それに関連付けられているすべてのメモリーを節約しないがぎり、変更するべきではありません。変数をデフォルトでない0以外の値に設定してもメリットはありません。特にデフォルトより大きい値は、必要以上のメモリーが割り当てられます。

- [performance_schema_max_statement_stack](#)

コマンド行形式	<code>--performance-schema-max-statement-stack=#</code>
システム変数	performance_schema_max_statement_stack
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	10

パフォーマンススキーマが統計を保持するネストされたストアプログラムコールの最大深度。この最大値を超えると、パフォーマンススキーマはストアプログラムステートメントが実行されるたびに [Performance_schema_nested_statement_lost](#) ステータス変数を増分します。

- [performance_schema_max_table_handles](#)

コマンド行形式	<code>--performance-schema-max-table-handles=#</code>
システム変数	performance_schema_max_table_handles

スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動スケーリングを示します。このリテラル値を割り当てないでください)

オープンしているテーブルオブジェクトの最大数。この値は、`table_handles` テーブルのサイズを制御します。テーブルハンドルを計測できないようにこの最大値を超えると、パフォーマンススキーマは `Performance_schema_table_handles_lost` ステータス変数を増分します。この変数の設定方法および使用方法の詳細は、[セクション27.7「パフォーマンススキーマステータスマニタリング」](#) を参照してください。

- [performance_schema_max_table_instances](#)

コマンド行形式	<code>--performance-schema-max-table-instances=#</code>
システム変数	performance_schema_max_table_instances
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動スケーリングを示します。このリテラル値を割り当てないでください)

インストールされるテーブルオブジェクトの最大数。この変数の設定方法および使用方法の詳細は、[セクション27.7「パフォーマンススキーマステータスマニタリング」](#) を参照してください。

- [performance_schema_max_table_lock_stat](#)

コマンド行形式	<code>--performance-schema-max-table-lock-stat=#</code>
システム変数	performance_schema_max_table_lock_stat
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動サイズ設定を示します。このリテラル値を割り当てないでください)

パフォーマンススキーマがロック統計を保持するテーブルの最大数。テーブルロック統計が失われるようにこの最大値を超えると、パフォーマンススキーマは `Performance_schema_table_lock_stat_lost` ステータス変数を増分しません。

- [performance_schema_max_thread_classes](#)

コマンド行形式	<code>--performance-schema-max-thread-classes=#</code>
システム変数	performance_schema_max_thread_classes
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	100

最小値	0
最大値 (≥ 8.0.12)	1024
最大値 (8.0.11)	256

スレッドインストゥルメントの最大数。この変数の設定方法および使用方法の詳細は、[セクション27.7「パフォーマンススキーマステータスマニタリング」](#)を参照してください。

- [performance_schema_max_thread_instances](#)

コマンド行形式	<code>--performance-schema-max-thread-instances=#</code>
システム変数	performance_schema_max_thread_instances
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動スケールリングを示します。このリテラル値を割り当てないでください)

インストゥルメントされるスレッドオブジェクトの最大数。この値は `threads` テーブルのサイズを制御します。この最大を超えたため、スレッドをインストゥルメントできない場合、パフォーマンススキーマは `Performance_schema_thread_instances_lost` ステータス変数を増分します。この変数の設定方法および使用方法の詳細は、[セクション27.7「パフォーマンススキーマステータスマニタリング」](#)を参照してください。

`max_connections` システム変数は、サーバーで実行できるスレッドの数に影響します。`performance_schema_max_thread_instances` は、これらの実行中のスレッドのうちインストゥルメントできる数に影響します。

`variables_by_thread` テーブルおよび `status_by_thread` テーブルには、フォアグラウンドスレッドに関するシステム変数およびステータス変数の情報のみが含まれます。すべてのスレッドがパフォーマンススキーマによって計測されるわけではない場合、このテーブルにはいくつかの行がありません。この場合、`Performance_schema_thread_instances_lost` ステータス変数はゼロより大きくなります。

- [performance_schema_session_connect_attrs_size](#)

コマンド行形式	<code>--performance-schema-session-connect-attrs-size=#</code>
システム変数	performance_schema_session_connect_attrs_size
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動サイズ設定を示します。このリテラル値を割り当てないでください)
最小値	-1
最大値	1048576

接続属性のキーと値のペアを保持するために予約されているスレッド当たりの事前割当て済メモリーの量。クライアントによって送信される接続属性データの集約サイズがこの量より大きい場合、パフォーマンススキーマは属性データを切り捨て、`Performance_schema_session_connect_attrs_lost` ステータス変数を増分して、`log_error_verbosity` システム変数が 1 より大きい場合に切り捨てが発生したことを示すメッセージをエラーログに書き込みます。`_truncated` 属性は、属性バッファに十分な領域がある場合、失われたバイト数を示す値とともに

にセッション属性にも追加されます。これにより、パフォーマンススキーマは接続属性テーブルで接続ごとの切り捨て情報を公開できます。この情報は、エラーログを確認せずに調べることができます。

`performance_schema_session_connect_attrs_size` のデフォルト値は、サーバーの起動時に自動サイズ設定されます。この値は小さい場合があるため、切捨て (`Performance_schema_session_connect_attrs_lost` がゼロ以外になる) が発生した場合は、`performance_schema_session_connect_attrs_size` を明示的に大きい値に設定することをお勧めします。

最大許容 `performance_schema_session_connect_attrs_size` 値は 1MB ですが、サーバーは受け入れる接続属性データの集計サイズに 64KB の制限を課すため、有効な最大値は 64KB です。クライアントが 64KB を超える属性データを送信しようとする、サーバーは接続を拒否します。詳細は、[セクション 27.12.9 「パフォーマンススキーマ接続属性テーブル」](#) を参照してください。

- `performance_schema_setup_actors_size`

コマンド行形式	<code>--performance-schema-setup-actors-size=#</code>
システム変数	<code>performance_schema_setup_actors_size</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動スケーリングを示します。このリテラル値を割り当てないでください)

`setup_actors` テーブル内の行数。

- `performance_schema_setup_objects_size`

コマンド行形式	<code>--performance-schema-setup-objects-size=#</code>
システム変数	<code>performance_schema_setup_objects_size</code>
スコープ	グローバル
動的	いいえ
SET_VAR ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動スケーリングを示します。このリテラル値を割り当てないでください)

`setup_objects` テーブル内の行数。

- `performance_schema_show_processlist`

コマンド行形式	<code>--performance-schema-show-processlist[={OFF ON}]</code>
導入	8.0.22
システム変数	<code>performance_schema_show_processlist</code>
スコープ	グローバル
動的	はい
SET_VAR ヒントの適用	いいえ
型	Boolean
デフォルト値	OFF

`SHOW PROCESSLIST` ステートメントは、すべてのアクティブスレッドからスレッドデータを収集することで、プロセス情報を提供します。`performance_schema_show_processlist` 変数は、使用する `SHOW PROCESSLIST` 実装を決定します:

- デフォルトの実装は、グローバル mutex を保持しながら、スレッドマネージャ内からアクティブスレッド間で繰り返されます。これは、特にビジー状態のシステムではパフォーマンスに悪影響を及ぼします。
- 代替の `SHOW PROCESSLIST` 実装は、パフォーマンススキーマ `processlist` テーブルに基づいています。この実装は、スレッドマネージャーではなくパフォーマンススキーマからアクティブなスレッドデータをクエリーするため、mutex は必要ありません。

代替実装を有効にするには、`performance_schema_show_processlist` システム変数を有効にします。デフォルトおよび代替の実装で同じ情報が得られるようにするには、特定の構成要件を満たす必要があります。[セクション 27.12.19.9 「processlist テーブル」](#) を参照してください。

- [performance_schema_users_size](#)

コマンド行形式	<code>--performance-schema-users-size=#</code>
システム変数	<code>performance_schema_users_size</code>
スコープ	グローバル
動的	いいえ
<code>SET_VAR</code> ヒントの適用	いいえ
型	Integer
デフォルト値	-1 (自動スケーリングを示します。このリテラル値を割り当てないでください)
最小値	-1 (自動スケーリングを示します。このリテラル値を割り当てないでください)
最大値	1048576

`users` テーブル内の行数。この変数が 0 の場合、パフォーマンススキーマは `users` テーブル内の接続統計情報または `status_by_user` テーブル内のステータス変数情報を保持しません。

27.16 パフォーマンススキーマステータス変数

パフォーマンススキーマは、メモリー制約のためロードまたは作成できなかったインストゥルメンテーションについての情報を提供するいくつかのステータス変数を実装しています。

```
mysql> SHOW STATUS LIKE 'perf%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Performance_schema_accounts_lost | 0 |
| Performance_schema_cond_classes_lost | 0 |
| Performance_schema_cond_instances_lost | 0 |
| Performance_schema_file_classes_lost | 0 |
| Performance_schema_file_handles_lost | 0 |
| Performance_schema_file_instances_lost | 0 |
| Performance_schema_hosts_lost | 0 |
| Performance_schema_locker_lost | 0 |
| Performance_schema_mutex_classes_lost | 0 |
| Performance_schema_mutex_instances_lost | 0 |
| Performance_schema_rwlock_classes_lost | 0 |
| Performance_schema_rwlock_instances_lost | 0 |
| Performance_schema_socket_classes_lost | 0 |
| Performance_schema_socket_instances_lost | 0 |
| Performance_schema_stage_classes_lost | 0 |
| Performance_schema_statement_classes_lost | 0 |
| Performance_schema_table_handles_lost | 0 |
| Performance_schema_table_instances_lost | 0 |
| Performance_schema_thread_classes_lost | 0 |
| Performance_schema_thread_instances_lost | 0 |
| Performance_schema_users_lost | 0 |
+-----+-----+
```

これらの変数を使用して、パフォーマンススキーマのステータスをチェックすることについては、[セクション 27.7 「パフォーマンススキーマステータスマニタリング」](#) を参照してください。

パフォーマンススキーマステータス変数には次の意味があります。

- [Performance_schema_accounts_lost](#)
accounts テーブルがいっぱいであったため、行を追加できなかった回数。
- [Performance_schema_cond_classes_lost](#)
ロードできなかった条件インストゥルメントの数。
- [Performance_schema_cond_instances_lost](#)
作成できなかった条件インストゥルメントインスタンスの数。
- [Performance_schema_digest_lost](#)
[events_statements_summary_by_digest](#) テーブルにインストゥルメントできなかったダイジェストインスタンスの数。 [performance_schema_digests_size](#) の値が小さすぎる場合、これは 0 以外になることがあります。
- [Performance_schema_file_classes_lost](#)
ロードできなかったファイルインストゥルメントの数。
- [Performance_schema_file_handles_lost](#)
オープンできなかったファイルインストゥルメントインスタンスの数。
- [Performance_schema_file_instances_lost](#)
作成できなかったファイルインストゥルメントインスタンスの数。
- [Performance_schema_hosts_lost](#)
hosts テーブルがいっぱいであったため、行を追加できなかった回数。
- [Performance_schema_index_stat_lost](#)
統計が失われたインデックスの数。 [performance_schema_max_index_stat](#) の値が小さすぎる場合は、ゼロ以外にすることができます。
- [Performance_schema_locker_lost](#)
次の状況のため、「失われる」が記録されないイベント数。
 - イベントが再帰的です (たとえば、A を待機することによって B で待機が発生し、それによって C で待機が発生したなど)。
 - ネストしたイベントスタックの深さが、実装によって課せられる制限より大きいです。パフォーマンススキーマによって記録されるイベントは再帰的ではないため、この変数は常に 0 であるべきです。
- [Performance_schema_memory_classes_lost](#)
メモリーインストゥルメントをロードできなかった回数。
- [Performance_schema_metadata_lock_lost](#)
[metadata_locks](#) テーブルでインストゥルメントできなかったメタデータロックの数。 [performance_schema_max_metadata_locks](#) の値が小さすぎる場合は、ゼロ以外にすることができます。
- [Performance_schema_mutex_classes_lost](#)
ロードできなかった相互排他ロックインストゥルメントの数。
- [Performance_schema_mutex_instances_lost](#)

作成できなかった相互排他ロックインストゥルメントインスタンスの数。

- [Performance_schema_nested_statement_lost](#)

統計が失われたストアプログラムステートメントの数。 [performance_schema_max_statement_stack](#) の値が小さすぎる場合は、ゼロ以外にすることができます。

- [Performance_schema_prepared_statements_lost](#)

[prepared_statements_instances](#) テーブルでインストゥルメントできなかったプリペアドステートメントの数。 [performance_schema_max_prepared_statements_instances](#) の値が小さすぎる場合は、ゼロ以外にすることができます。

- [Performance_schema_program_lost](#)

統計が失われたストアプログラムの数。 [performance_schema_max_program_instances](#) の値が小さすぎる場合は、ゼロ以外にすることができます。

- [Performance_schema_rwlock_classes_lost](#)

ロードできなかった rwlock インストゥルメントの数。

- [Performance_schema_rwlock_instances_lost](#)

作成できなかった rwlock インストゥルメントインスタンスの数。

- [Performance_schema_session_connect_attrs_longest_seen](#)

[performance_schema_session_connect_attrs_size](#) システム変数の値に対してパフォーマンススキーマによって実行される接続属性 size-limit チェックに加えて、サーバーは事前チェックを実行し、受け入れる接続属性データの集約サイズに 64KB の制限を課します。クライアントが 64KB を超える属性データを送信しようとする、サーバーは接続を拒否します。それ以外の場合、サーバーは属性バッファが有効であるとみなし、[Performance_schema_session_connect_attrs_longest_seen](#) ステータス変数で最も長いバッファのサイズを追跡します。この値が [performance_schema_session_connect_attrs_size](#) より大きい場合、DBA は後者の値を増やすことも、大量の属性データを送信しているクライアントを調査することもできます。

接続属性の詳細は、[セクション27.12.9「パフォーマンススキーマ接続属性テーブル」](#) を参照してください。

- [Performance_schema_session_connect_attrs_lost](#)

接続属性の切捨てが発生した接続の数。特定の接続について、クライアントが、集約サイズが [performance_schema_session_connect_attrs_size](#) システム変数の値で許可されている予約済みストレージより大きい接続属性のキーと値のペアを送信した場合、パフォーマンススキーマは属性データを切り捨て、[Performance_schema_session_connect_attrs_lost](#) を増分します。この値がゼロ以外の場合、[performance_schema_session_connect_attrs_size](#) を大きい値に設定できます。

接続属性の詳細は、[セクション27.12.9「パフォーマンススキーマ接続属性テーブル」](#) を参照してください。

- [Performance_schema_socket_classes_lost](#)

ロードできなかったソケットインストゥルメントの数。

- [Performance_schema_socket_instances_lost](#)

作成できなかったソケットインストゥルメントインスタンスの数。

- [Performance_schema_stage_classes_lost](#)

ロードできなかったステージインストゥルメントの数。

- [Performance_schema_statement_classes_lost](#)

ロードできなかったステートメントインストゥルメントの数。

- [Performance_schema_table_handles_lost](#)

オープンできなかったテーブルインストゥルメントインスタンスの数。 `performance_schema_max_table_handles` の値が小さすぎる場合は、ゼロ以外にすることができます。

- `Performance_schema_table_instances_lost`

作成できなかったテーブルインストゥルメントインスタンスの数。

- `Performance_schema_table_lock_stat_lost`

ロック統計が失われたテーブルの数。 `performance_schema_max_table_lock_stat` の値が小さすぎる場合は、ゼロ以外にすることができます。

- `Performance_schema_thread_classes_lost`

ロードできなかったスレッドインストゥルメントの数。

- `Performance_schema_thread_instances_lost`

`threads` テーブルにインストゥルメントできなかったスレッドインスタンスの数。
`performance_schema_max_thread_instances` の値が小さすぎる場合、これは 0 以外になることがあります。

- `Performance_schema_users_lost`

`users` テーブルがいっぱいであったため、行を追加できなかった回数。

27.17 パフォーマンススキーマのメモリー割り当てモデル

パフォーマンススキーマは、次のメモリー割り当てモデルを使用します:

- サーバー起動時にメモリーを割り当てることができます
- サーバー操作中に追加のメモリーを割り当てることができます
- サーバー操作中にメモリーを解放しないでください (ただし、リサイクルされている可能性があります)
- シャットダウン時に使用されたすべてのメモリーを解放

その結果、パフォーマンススキーマを少ない構成で使えるようにメモリー制約が緩和され、サーバー負荷で消費量がスケールされるようにメモリーフットプリントが減少します。使用されるメモリーは、見積もられた負荷や明示的に構成された負荷ではなく、実際に見られる負荷によって異なります。

いくつかのパフォーマンススキーマのサイズ変更パラメータは自動調整されるため、メモリー割り当ての明示的な制限を確立しないかぎり、明示的に構成する必要はありません:

```
performance_schema_accounts_size
performance_schema_hosts_size
performance_schema_max_cond_instances
performance_schema_max_file_instances
performance_schema_max_index_stat
performance_schema_max_metadata_locks
performance_schema_max_mutex_instances
performance_schema_max_prepared_statements_instances
performance_schema_max_program_instances
performance_schema_max_rwlock_instances
performance_schema_max_socket_instances
performance_schema_max_table_handles
performance_schema_max_table_instances
performance_schema_max_table_lock_stat
performance_schema_max_thread_instances
performance_schema_users_size
```

自動スケールパラメータの場合、構成は次のように機能します:

- 値を -1 (デフォルト) に設定すると、パラメータは自動スケールされます:
 - 対応する内部バッファは最初は空で、メモリーは割り当てられません。

- パフォーマンススキーマがデータを収集すると、対応するバッファにメモリーが割り当てられます。バッファサイズは無制限で、負荷に伴って大きくなる可能性があります。
- 値を 0 に設定した場合:
 - 対応する内部バッファは最初は空で、メモリーは割り当てられません。
- 値が $N > 0$ に設定されている場合:
 - 対応する内部バッファは最初は空で、メモリーは割り当てられません。
 - パフォーマンススキーマがデータを収集すると、バッファサイズが N に達するまで、対応するバッファにメモリーが割り当てられます。
 - バッファサイズが N に達すると、メモリーはこれ以上割り当てられません。このバッファのパフォーマンススキーマによって収集されたデータは失われ、対応する「失われたインスタンス」カウンタは増分されます。

パフォーマンススキーマが使用しているメモリー量を確認するには、その目的に合わせて設計されたインストゥルメントを確認します。パフォーマンススキーマはメモリーを内部的に割り当て、各バッファを専用インストゥルメントに関連付けて、メモリー消費を個々のバッファにトレースできるようにします。接頭辞 `memory/performance_schema/` で指定されたインストゥルメントは、これらの内部バッファに割り当てられているメモリー量を公開します。バッファはサーバーに対してグローバルであるため、インストゥルメントは `memory_summary_global_by_event_name` テーブルにのみ表示され、他の `memory_summary_by_xxx_by_event_name` テーブルには表示されません。

このクエリーは、メモリーインストゥルメントに関連付けられた情報を表示します:

```
SELECT * FROM performance_schema.memory_summary_global_by_event_name
WHERE EVENT_NAME LIKE 'memory/performance_schema/%';
```

27.18 パフォーマンススキーマとプラグイン

`UNINSTALL PLUGIN` によってプラグインを削除することは、そのプラグインのコードですでに収集された情報には影響しません。プラグインのロード中にコードの実行に費やされた時間は、プラグインがあとでアンロードされた場合でも費やされます。集計情報を含む関連付けられたイベント情報は、`performance_schema` データベーステーブルで読み取り可能なままになります。プラグインのインストールと削除の効果の追加情報については、[セクション 27.7「パフォーマンススキーマステータスマニタリング」](#)を参照してください。

プラグインコードをインストゥルメントするプラグイン実装者は、プラグインをロードするユーザーがその要件を把握できるように、インストゥルメンテーションの特性をドキュメント化してください。たとえば、サードパーティーストレージエンジンでは、そのドキュメントに、エンジンが相互排他ロックおよびその他のインストゥルメントに必要なとするメモリーの量を記載してください。

27.19 問題を診断するためのパフォーマンススキーマの使用

パフォーマンススキーマは、DBA が「大まかな推量」ではなく、実際に測定して、パフォーマンスのチューニングを行うのに役立つツールです。このセクションでは、この目的でパフォーマンススキーマを使用するいくつかの方法について説明します。ここでの説明は、[セクション 27.4.2「パフォーマンススキーマイベントフィルタリング」](#)で説明しているイベントフィルタリングの使用に依存します。

次の例では、パフォーマンスボトルネックの調査など、反復される問題の分析に使用できる 1 つの方法を示しています。開始するには、パフォーマンスが「遅すぎる」とみなされ、最適化が必要な反復可能なユースケースが必要であり、すべてのインストゥルメンテーションを有効にすべきです (事前フィルタリングなし)。

- ユースケースを実行します。
- パフォーマンススキーマテーブルを使用して、パフォーマンスの問題の原因を分析します。この分析は、フィルタリング後に大きく依存します。
- 除外する問題領域については、対応するインストゥルメントを無効にします。たとえば、問題が特定のストレージエンジンのファイル I/O に関連していないことが分析に示されている場合、そのエンジンのファイル I/O インス

ツールメントを無効にします。次に、履歴およびサマリーテーブルを切り捨て、これまでに収集されたイベントを削除します。

4. ステップ 1 のプロセスを繰り返します。

反復のたびに、パフォーマンススキーマの出力 (特に `events_waits_history_long` テーブル) に含まれる「「ノイズ」」の数が少なく、重要でないインストゥルメントによって発生する「「ノイズ」」が少なくなり、このテーブルのサイズが固定されている場合は、手動での問題の分析に関連するより多くのデータが含まれます。

「signal/noise」比率が改善され、分析が容易になるため、反復ごとに調査が問題の根本原因に近づいて近づく必要があります。

5. パフォーマンスボトルネックの原因を突き止めたら、次のような適切な修正措置をとります。

- サーバーパラメータ (キャッシュサイズ、メモリーなど) をチューニングします。
- クエリーを違う方法で書いてチューニングします。
- データベーススキーマ (テーブル、インデックスなど) をチューニングします。
- コードをチューニングします (これはストレージエンジンまたはサーバー開発者のみに適用されます)。

6. ステップ 1 から再開して、パフォーマンスへの変更の効果を確認します。

`mutex_instances.LOCKED_BY_THREAD_ID` および `rwlock_instances.WRITE_LOCKED_BY_THREAD_ID` カラムは、パフォーマンスボトルネックまたはデッドロックの調査にきわめて重要です。これは、次のようなパフォーマンススキーマインストゥルメンテーションによって可能になります。

1. スレッド 1 は相互排他ロックを待機してスタックしているとします。
2. スレッドが何を待機しているかを特定できます。

```
SELECT * FROM performance_schema.events_waits_current
WHERE THREAD_ID = thread_1;
```

たとえば、`events_waits_current.OBJECT_INSTANCE_BEGIN` にあるように、スレッドが相互排他ロック A を待機していることがクエリー結果に示されます。

3. 相互排他ロック A を保持しているスレッドを特定できます。

```
SELECT * FROM performance_schema.mutex_instances
WHERE OBJECT_INSTANCE_BEGIN = mutex_A;
```

たとえば、`mutex_instances.LOCKED_BY_THREAD_ID` にあるように、相互排他ロック A を保持しているのがスレッド 2 であることがクエリー結果に示されます。

4. スレッド 2 が何を実行しているかを確認できます。

```
SELECT * FROM performance_schema.events_waits_current
WHERE THREAD_ID = thread_2;
```

27.19.1 パフォーマンススキーマを使用したクエリープロファイリング

次の例は、パフォーマンススキーマのステートメントイベントとステージイベントを使用して、`SHOW PROFILES` および `SHOW PROFILE` ステートメントによって提供されるプロファイリング情報と同等のデータを取得する方法を示しています。

`setup_actors` テーブルを使用すると、ホスト、ユーザーまたはアカウントごとに履歴イベントの収集を制限して、実行時のオーバーヘッドおよび履歴テーブルに収集されるデータ量を減らすことができます。例の最初のステップは、履歴イベントの収集を特定のユーザーに制限する方法を示しています。

パフォーマンススキーマは、タイミングデータを標準単位に正規化するために、イベントタイマー情報をピコ秒 (1 秒に 1 兆) で表示します。次の例では、`TIMER_WAIT` 値を 1000000000000 で除算して、データを秒単位で表示します。また、値は小数点以下 6 桁に切り捨てられ、`SHOW PROFILES` および `SHOW PROFILE` ステートメントと同じ形式でデータが表示されます。

- 履歴イベントの収集を、クエリーを実行するユーザーに制限します。デフォルトでは、`setup_actors` は、すべてのフォアグラウンドスレッドのモニタリングおよび履歴イベント収集を許可するように構成されています:

```
mysql> SELECT * FROM performance_schema.setup_actors;
+-----+-----+-----+-----+
| HOST | USER | ROLE | ENABLED | HISTORY |
+-----+-----+-----+-----+
| %    | %    | %    | YES     | YES     |
+-----+-----+-----+-----+
```

`setup_actors` テーブルのデフォルト行を更新して、すべてのフォアグラウンドスレッドの履歴イベント収集および監視を無効にし、クエリーを実行するユーザーの監視および履歴イベント収集を有効にする新しい行を挿入します:

```
mysql> UPDATE performance_schema.setup_actors
  SET ENABLED = 'NO', HISTORY = 'NO'
  WHERE HOST = '%' AND USER = '%';

mysql> INSERT INTO performance_schema.setup_actors
  (HOST,USER,ROLE,ENABLED,HISTORY)
  VALUES('localhost','test_user','%','YES','YES');
```

これで、`setup_actors` テーブルのデータは次のようになります:

```
mysql> SELECT * FROM performance_schema.setup_actors;
+-----+-----+-----+-----+
| HOST | USER | ROLE | ENABLED | HISTORY |
+-----+-----+-----+-----+
| %    | %    | %    | NO      | NO      |
| localhost | test_user | %    | YES     | YES     |
+-----+-----+-----+-----+
```

- `setup_instruments` テーブルを更新して、ステートメントおよびステージインストゥルメンテーションが有効になっていることを確認します。一部のインストゥルメントは、デフォルトですでに有効になっている場合があります。

```
mysql> UPDATE performance_schema.setup_instruments
  SET ENABLED = 'YES', TIMED = 'YES'
  WHERE NAME LIKE '%statement%';

mysql> UPDATE performance_schema.setup_instruments
  SET ENABLED = 'YES', TIMED = 'YES'
  WHERE NAME LIKE '%stage%';
```

- `events_statements_*` および `events_stages_*` コンシューマが有効になっていることを確認します。一部のコンシューマは、デフォルトですでに有効になっている場合があります。

```
mysql> UPDATE performance_schema.setup_consumers
  SET ENABLED = 'YES'
  WHERE NAME LIKE '%events_statements_%';

mysql> UPDATE performance_schema.setup_consumers
  SET ENABLED = 'YES'
  WHERE NAME LIKE '%events_stages_%';
```

- 監視しているユーザーアカウントで、プロファイリングするステートメントを実行します。例:

```
mysql> SELECT * FROM employees.employees WHERE emp_no = 10001;
+-----+-----+-----+-----+
| emp_no | birth_date | first_name | last_name | gender | hire_date |
+-----+-----+-----+-----+
| 10001 | 1953-09-02 | Georgi    | Facello  | M      | 1986-06-26 |
+-----+-----+-----+-----+
```

- `events_statements_history_long` テーブルをクエリーして、ステートメントの `EVENT_ID` を識別します。このステップは、`Query_ID` を識別するための `SHOW PROFILES` の実行に似ています。次のクエリーでは、`SHOW PROFILES` のような出力が生成されます:

```
mysql> SELECT EVENT_ID, TRUNCATE(TIMER_WAIT/1000000000000,6) as Duration, SQL_TEXT
  FROM performance_schema.events_statements_history_long WHERE SQL_TEXT like '%10001%';
+-----+-----+-----+
| EVENT_ID | Duration | SQL_TEXT |
+-----+-----+-----+
```

```
| event_id | duration | sql_text |
+-----+-----+-----+
| 31 | 0.028310 | SELECT * FROM employees.employees WHERE emp_no = 10001 |
+-----+-----+-----+
```

6. `events_stages_history_long` テーブルをクエリーして、ステートメントステージイベントを取得します。ステージは、イベントネストを使用してステートメントにリンクされます。各ステージイベントレコードには、親ステートメントの `EVENT_ID` を含む `NESTING_EVENT_ID` カラムがあります。

```
mysql> SELECT event_name AS Stage, TRUNCATE(TIMER_WAIT/1000000000000,6) AS Duration
FROM performance_schema.events_stages_history_long WHERE NESTING_EVENT_ID=31;
+-----+-----+
| Stage | Duration |
+-----+-----+
| stage/sql/starting | 0.000080 |
| stage/sql/checking permissions | 0.000005 |
| stage/sql/Opening tables | 0.027759 |
| stage/sql/init | 0.000052 |
| stage/sql/System lock | 0.000009 |
| stage/sql/optimizing | 0.000006 |
| stage/sql/statistics | 0.000082 |
| stage/sql/preparing | 0.000008 |
| stage/sql/executing | 0.000000 |
| stage/sql/Sending data | 0.000017 |
| stage/sql/end | 0.000001 |
| stage/sql/query end | 0.000004 |
| stage/sql/closing tables | 0.000006 |
| stage/sql/freeing items | 0.000272 |
| stage/sql/cleaning up | 0.000001 |
+-----+-----+
```

27.19.2 親イベント情報の取得

`data_locks` テーブルには、保持およびリクエストされたデータロックが表示されます。このテーブルの行には、ロックを所有するセッションのスレッド ID を示す `THREAD_ID` カラムと、ロックの原因となったパフォーマンススキーマイベントを示す `EVENT_ID` カラムがあります。(`THREAD_ID`、`EVENT_ID`) 値のタプルは、他の「パフォーマンススキーマ」テーブルの親イベントを暗黙的に識別します:

- `events_waits_xxx` テーブルの親待機イベント
- `events_stages_xxx` テーブルの親ステージイベント
- `events_statements_xxx` テーブルの親ステートメントイベント
- `events_transactions_current` テーブルの親トランザクションイベント

親イベントの詳細を取得するには、`THREAD_ID` カラムと `EVENT_ID` カラムを適切な親イベントテーブルの同名のカラムと結合します。リレーションはネストされたセットデータモデルに基づいているため、結合には複数の句があります。 `parent` および `child` でそれぞれ表される親テーブルと子テーブルの場合、結合は次のようになります:

```
WHERE
parent.THREAD_ID = child.THREAD_ID /* 1 */
AND parent.EVENT_ID < child.EVENT_ID /* 2 */
AND (
child.EVENT_ID <= parent.END_EVENT_ID /* 3a */
OR parent.END_EVENT_ID IS NULL /* 3b */
)
```

結合の条件は次のとおりです:

1. 親イベントと子イベントは同じスレッド内にあります。
2. 子イベントは親イベントの後に開始されるため、その `EVENT_ID` 値は親の値より大きくなります。
3. 親イベントが完了したか、まだ実行中です。

ロック情報を検索するために、`data_locks` は子イベントを含むテーブルです。

`data_locks` テーブルには既存のロックのみが表示されるため、親イベントを含むテーブルに関して次の考慮事項が適用されます:

- トランザクションの場合、唯一の選択肢は `events_transactions_current` です。トランザクションが完了した場合、そのトランザクションはトランザクション履歴テーブルにある可能性があります、ロックはすでに解除されています。
- ステートメントの場合、ロックを取得したステートメントがすでに完了したトランザクション内のステートメントであるか (`events_statements_history` を使用)、ステートメントがまだ実行中であるか (`events_statements_current` を使用)、によって異なります。
- ステージの場合、ロジックはステートメントのロジックと似ています。 `events_stages_history` または `events_stages_current` を使用します。
- 待機の場合、ロジックはステートメントのロジックと似ています。 `events_waits_history` または `events_waits_current` を使用します。ただし、多くの待機が記録されるため、ロックの原因となった待機はすでに履歴テーブルからなくなっている可能性があります。

待機イベント、ステージイベントおよびステートメントイベントは、履歴からすぐに消えます。長時間前に実行されたステートメントがロックを取得したが、まだオープンしているトランザクションにある場合、ステートメントを見つけることはできない可能性があります、トランザクションを見つけることはできます。

このため、ネストされたセットデータモデルは親イベントの検索に適しています。中間ノードがすでに履歴テーブルから削除されている場合、親/子関係の次のリンク (データロック -> 親待機 -> 親ステージ -> 親トランザクション) は適切に機能しません。

次のシナリオは、ロックが取得されたステートメントの親トランザクションを検索する方法を示しています:

セッション A:

```
[1] START TRANSACTION;
[2] SELECT * FROM t1 WHERE pk = 1;
[3] SELECT 'Hello, world';
```

セッション B:

```
SELECT ...
FROM performance_schema.events_transactions_current AS parent
  INNER JOIN performance_schema.data_locks AS child
WHERE
  parent.THREAD_ID = child.THREAD_ID
  AND parent.EVENT_ID < child.EVENT_ID
  AND (
    child.EVENT_ID <= parent.END_EVENT_ID
    OR parent.END_EVENT_ID IS NULL
  );
```

セッション B のクエリーでは、`pk=1` を使用してレコードのデータロックを所有しているステートメント[2]を表示する必要があります。

セッション A がさらにステートメントを実行すると、[2]は履歴テーブルからフェードアウトします。

クエリーでは、実行されたステートメント、ステージまたは待機の数に関係なく、[1]で開始されたトランザクションが表示されます。

サーバーで他のクエリーが実行されないことを前提として (履歴が保持されるように)、`events_xxx_history_long` テーブルを使用してさらにデータを表示することもできます (トランザクションを除く)。

27.20 パフォーマンススキーマの制約

パフォーマンススキーマでは、データの収集または生成に相互排他ロックを使用できないので、一貫性は保証されず、適切な結果にならないことがあります。 `performance_schema` テーブルのイベント値は、非決定的であり、反復不可です。

イベント情報を別のテーブルに保存する場合は、元のイベントが後で使用可能なままであると想定しないでください。たとえば、`performance_schema` テーブルから一時テーブルにイベントを選択し、あとからそのテーブルと元のテーブルを結合させる場合、一致するものがない可能性があります。

`mysqldump` と `BACKUP DATABASE` は、`performance_schema` データベース内のテーブルを無視します。

`performance_schema` データベース内のテーブルは、`LOCK TABLES` でロックできませんが、`setup_xxx` テーブルは除きます。

`performance_schema` データベース内のテーブルにインデックスを設定できません。

`performance_schema` データベース内のテーブルは複製されません。

タイマーの種類は、プラットフォームごとに異なります。`performance_timers` テーブルは使用可能なイベントタイマーを示します。特定のタイマー名に対するこのテーブルでの値が `NULL` の場合、そのタイマーはプラットフォームでサポートされていません。

ストレージエンジンに適用されるインストゥルメントは、すべてのストレージエンジンに実装されていないことがあります。各サードパーティーエンジンのインストゥルメンテーションはエンジン管理者の責任です。

第 28 章 MySQL sys スキーマ

目次

28.1 sys スキーマを使用するための前提条件	4445
28.2 sys スキーマの使用	4446
28.3 sys スキーマ進捗レポート	4447
28.4 sys スキーマオブジェクトリファレンス	4447
28.4.1 sys スキーマオブジェクトインデックス	4447
28.4.2 sys スキーマのテーブルおよびトリガー	4451
28.4.3 sys スキーマビュー	4453
28.4.4 sys スキーマストアードプロシージャ	4491
28.4.5 sys スキーマストアードファンクション	4509

MySQL 8.0 には、パフォーマンススキーマによって収集されたデータを DBA および開発者が解釈するのに役立つ一連のオブジェクトである `sys` スキーマが含まれています。`sys` スキーマオブジェクトは、一般的なチューニングおよび診断ユースケースに使用できます。このスキーマのオブジェクトは次のとおりです:

- パフォーマンススキーマデータをわかりやすい形式に要約するビュー。
- パフォーマンススキーマの構成や診断レポートの生成などの操作を実行するストアードプロシージャ。
- パフォーマンススキーマ構成をクエリーして、フォーマットサービスを提供するストアードファンクション。

新規インストールでは、`--initialize` または `--initialize-insecure` オプションを指定して `mysqld` を使用する場合、`sys` スキーマはデータディレクトリの初期化時にデフォルトでインストールされます。これが不要な場合は、初期化後に `sys` スキーマを手動で削除できます。

MySQL のアップグレード手順では、`sys` スキーマは存在するが `version` ビューがない場合、このビューが存在しないことはユーザーが作成した `sys` スキーマを示していることを前提として、エラーが生成されます。この場合にアップグレードするには、まず既存の `sys` スキーマを削除するか、名前を変更します。

`sys` スキーマオブジェクトには、`'mysql.sys'@'localhost'` の `DEFINER` があります。専用の `mysql.sys` アカウントを使用すると、DBA が `root` アカウントの名前を変更または削除した場合に発生する問題を回避できます。

28.1 sys スキーマを使用するための前提条件

`sys` スキーマを使用する前に、このセクションで説明する前提条件を満たしている必要があります。

`sys` スキーマはパフォーマンススキーマにアクセスする代替手段を提供するため、`sys` スキーマが機能するにはパフォーマンススキーマを有効にする必要があります。 [セクション27.3「パフォーマンススキーマ起動構成」](#) を参照してください。

`sys` スキーマへの完全なアクセス権を持つユーザーには、次の権限が必要です:

- すべての `sys` テーブルおよびビューに対する `SELECT`
- すべての `sys` ストアドプロシージャおよびファンクションに対する `EXECUTE`
- `sys_config` テーブルの `INSERT` および `UPDATE`(変更が行われる場合)
- 特定の `sys` スキーマストアードプロシージャおよびファンクションに対する追加の権限。説明に記載されています (`ps_setup_save()` プロシージャなど)

`sys` スキーマオブジェクトの基礎となるオブジェクトに対する権限も必要です:

- `sys` スキーマオブジェクトによってアクセスされる「パフォーマンススキーマ」テーブルに対する `SELECT`、および `sys` スキーマオブジェクトを使用して更新されるテーブルに対する `UPDATE`
- `INFORMATION_SCHEMA INNODB_BUFFER_PAGE` テーブル用の `PROCESS`

`sys` スキーマ機能を最大限に活用するには、特定のパフォーマンススキーマインストールメントおよびコンシューマを有効にし、(インストールメント用に) 時間を設定する必要があります:

- すべての `wait` インストゥルメント
- すべての `stage` インストゥルメント
- すべての `statement` インストゥルメント
- すべてのイベントの `xxx_current` および `xxx_history_long` コンシューマ

`sys` スキーマ自体を使用して、追加のインストゥルメントおよびコンシューマをすべて有効にできます:

```
CALL sys.ps_setup_enable_instrument('wait');
CALL sys.ps_setup_enable_instrument('stage');
CALL sys.ps_setup_enable_instrument('statement');
CALL sys.ps_setup_enable_consumer('current');
CALL sys.ps_setup_enable_consumer('history_long');
```

注記

`sys` スキーマの多くの用途では、データ収集にはデフォルトのパフォーマンススキーマで十分です。前述のすべてのインストゥルメントおよびコンシューマを有効にするとパフォーマンスに影響するため、必要な追加の構成のみを有効にすることをお勧めします。また、追加の構成を有効にすると、次のようにデフォルトの構成を簡単に復元できることに注意してください:

```
CALL sys.ps_setup_reset_to_default(TRUE);
```

28.2 sys スキーマの使用

`sys` スキーマをデフォルトスキーマにすると、そのオブジェクトへの参照をスキーマ名で修飾する必要がなくなります:

```
mysql> USE sys;
Database changed
mysql> SELECT * FROM version;
+-----+-----+
| sys_version | mysql_version |
+-----+-----+
| 2.0.0      | 8.0.13-debug  |
+-----+-----+
```

(`version` ビューには、`sys` スキーマおよび MySQL サーバーのバージョンが表示されます。)

別のスキーマがデフォルトである (または単に明示的である) ときに `sys` スキーマオブジェクトにアクセスするには、オブジェクト参照をスキーマ名で修飾します:

```
mysql> SELECT * FROM sys.version;
+-----+-----+
| sys_version | mysql_version |
+-----+-----+
| 2.0.0      | 8.0.13-debug  |
+-----+-----+
```

`sys` スキーマには、様々な方法で「パフォーマンススキーマ」テーブルを要約する多数のビューが含まれています。これらのビューのほとんどはベアになっているため、ベアのメンバーは他のメンバーと同じ名前に `x$` 接頭辞を付けたものになります。たとえば、`host_summary_by_file_io` ビューでは、ホスト別にグループ化されたファイル I/O が要約され、ピコ秒からより読みやすい値 (単位付き) に変換された待機時間が表示されます

```
mysql> SELECT * FROM sys.host_summary_by_file_io;
+-----+-----+-----+
| host      | ios  | io_latency |
+-----+-----+-----+
| localhost | 67570 | 5.38 s    |
| background | 3468 | 4.18 s    |
+-----+-----+-----+
```

`x$host_summary_by_file_io` ビューには、同じデータが要約されますが、書式設定されていないピコ秒待機時間が表示されます:

```
mysql> SELECT * FROM sys.x$host_summary_by_file_io;
+-----+-----+-----+
| host      | ios  | io_latency |
+-----+-----+-----+
```

```
| host | ios | io_latency |
+-----+-----+-----+
| localhost | 67574 | 5380678125144 |
| background | 3474 | 4758696829416 |
+-----+-----+-----+
```

`x$` 接頭辞のないビューは、人間が読みやすくわかりやすい出力を提供することを目的としています。同じ値を RAW 形式で表示する `x$` 接頭辞を持つビューは、データに対して独自の処理を実行する他のツールでの使用を目的としています。 `x$` 以外と `x$` 以外のビューの違いの詳細は、[セクション28.4.3「sys スキーマビュー」](#) を参照してください。

`sys` スキーマオブジェクト定義を調べるには、適切な `SHOW` ステートメントまたは `INFORMATION_SCHEMA` クエリを使用します。たとえば、`session` ビューおよび `format_bytes()` 関数の定義を調べるには、次のステートメントを使用します:

```
mysql> SHOW CREATE VIEW sys.session;
mysql> SHOW CREATE FUNCTION sys.format_bytes;
```

ただし、これらのステートメントは、比較的フォーマットされていない形式で定義を表示します。読みやすい形式でオブジェクト定義を表示するには、MySQL ソース配布の `scripts/sys_schema` にある個々の `.sql` ファイルにアクセスします。MySQL 8.0.18 より前は、ソースは <https://github.com/mysql/mysql-sys> の `sys` スキーマ開発 web サイトから入手可能な個別の配布で管理されます。

デフォルトでは、`mysqldump` も `mysqlpump` も `sys` スキーマをダンプしません。ダンプファイルを生成するには、次のいずれかのコマンドを使用して、コマンドラインで `sys` スキーマに明示的に名前を付けます:

```
mysqldump --databases --routines sys > sys_dump.sql
mysqlpump sys > sys_dump.sql
```

ダンプファイルからスキーマを再インストールするには、次のコマンドを使用します:

```
mysql < sys_dump.sql
```

28.3 sys スキーマ進捗レポート

次の `sys` スキーマビューは、長時間実行トランザクションの進捗レポートを提供します:

```
processlist
session
x$processlist
x$session
```

必要なインストールメントおよびコンシューマが有効になっている場合、これらのビューの `progress` カラムには、進捗レポートをサポートするステージで完了した作業の割合が表示されます。

ステージ進捗レポートでは、`events_stages_current` コンシューマおよび進捗情報が必要なインストールメントを有効にする必要があります。現在、次のステージの手段は進捗レポートをサポートしています:

```
stage/sql/Copying to tmp table
stage/innodb/alter table (end)
stage/innodb/alter table (flush)
stage/innodb/alter table (insert)
stage/innodb/alter table (log apply index)
stage/innodb/alter table (log apply table)
stage/innodb/alter table (merge sort)
stage/innodb/alter table (read PK and internal sort)
stage/innodb/buffer pool load
```

見積および完了した作業レポートをサポートしていないステージの場合、または必要なインストールメントまたはコンシューマが有効になっていない場合、`progress` カラムは `NULL` です。

28.4 sys スキーマオブジェクトリファレンス

`sys` スキーマには、テーブルとトリガー、ビューおよびストアドプロシージャとストアドファンクションが含まれます。次の各セクションでは、これらの各オブジェクトについて詳しく説明します。

28.4.1 sys スキーマオブジェクトインデックス

次のテーブルに、`sys` スキーマオブジェクトをリストし、それぞれについて簡単に説明します。

表 28.1 sys スキーマのテーブルおよびトリガー

テーブルまたはトリガー名	説明
sys_config	sys スキーマ構成オプション
sys_config_insert_set_user	sys_config 挿入トリガー
sys_config_update_set_user	sys_config 更新トリガー

表 28.2 sys スキーマビュー

ビュー名	説明
host_summary, x\$host_summary	ホスト別にグループ化されたステートメントアクティビティ、ファイル I/O,および接続
host_summary_by_file_io, x\$host_summary_by_file_io	ホスト別にグループ化されたファイル I/O,
host_summary_by_file_io_type, x\$host_summary_by_file_io_type	ホストおよびイベントタイプ別にグループ化されたファイル I/O,
host_summary_by_stages, x\$host_summary_by_stages	ホスト別にグループ化されたステートメントステージ
host_summary_by_statement_latency, x\$host_summary_by_statement_latency	ホスト別にグループ化されたステートメント統計
host_summary_by_statement_type, x\$host_summary_by_statement_type	実行されたステートメント (ホストおよびステートメント別にグループ化)
innodb_buffer_stats_by_schema, x\$innodb_buffer_stats_by_schema	スキーマ別にグループ化された InnoDB バッファ情報
innodb_buffer_stats_by_table, x\$innodb_buffer_stats_by_table	スキーマおよびテーブル別にグループ化された InnoDB バッファ情報
innodb_lock_waits, x\$innodb_lock_waits	InnoDB ロック情報
io_by_thread_by_latency, x\$io_by_thread_by_latency	スレッド別にグループ化された I/O コンシューマ
io_global_by_file_by_bytes, x\$io_global_by_file_by_bytes	ファイルおよびバイトでグループ化されたグローバル I/O コンシューマ
io_global_by_file_by_latency, x\$io_global_by_file_by_latency	ファイルおよび待機時間別にグループ化されたグローバル I/O コンシューマ
io_global_by_wait_by_bytes, x\$io_global_by_wait_by_bytes	バイト単位でグループ化されたグローバル I/O コンシューマ
io_global_by_wait_by_latency, x\$io_global_by_wait_by_latency	待機時間別にグループ化されたグローバル I/O コンシューマ
latest_file_io, x\$latest_file_io	ファイルおよびスレッド別にグループ化された最新の I/O,
memory_by_host_by_current_bytes, x\$memory_by_host_by_current_bytes	ホスト別にグループ化されたメモリー使用量
memory_by_thread_by_current_bytes, x\$memory_by_thread_by_current_bytes	スレッド別にグループ化されたメモリー使用量
memory_by_user_by_current_bytes, x\$memory_by_user_by_current_bytes	ユーザー別にグループ化されたメモリー使用量
memory_global_by_current_bytes, x\$memory_global_by_current_bytes	割当てタイプ別にグループ化されたメモリー使用
memory_global_total, x\$memory_global_total	合計メモリー使用量
metrics	サーバーメトリック
processlist, x\$processlist	プロセスリスト情報
ps_check_lost_instrumentation	インストゥルメントが失われた変数
schema_auto_increment_columns	AUTO_INCREMENT のカラム情報

ビュー名	説明
schema_index_statistics, x\$schema_index_statistics	インデックス統計
schema_object_overview	各スキーマ内のオブジェクトのタイプ
schema_redundant_indexes	重複または重複したインデックス
schema_table_lock_waits, x\$schema_table_lock_waits	メタデータロックを待機しているセッション
schema_table_statistics, x\$schema_table_statistics	テーブル統計
schema_table_statistics_with_buffer, x \$schema_table_statistics_with_buffer	テーブル統計 (InnoDB バッファプール統計を含む)
schema_tables_with_full_table_scans, x \$schema_tables_with_full_table_scans	全スキャンでアクセスされているテーブル
schema_unused_indexes	アクティブに使用されていないインデックス
session, x\$session	ユーザーセッションのプロセスリスト情報
session_ssl_status	接続 SSL 情報
statement_analysis, x\$statement_analysis	ステートメント集計統計
statements_with_errors_or_warnings, x \$statements_with_errors_or_warnings	エラーまたは警告が生成されたステートメント
statements_with_full_table_scans, x \$statements_with_full_table_scans	全テーブルスキャンを実行したステートメント
statements_with_runtimes_in_95th_percentile, x \$statements_with_runtimes_in_95th_percentile	平均ランタイムが最も高いステートメント
statements_with_sorting, x\$statements_with_sorting	ソートを実行したステートメント
statements_with_temp_tables, x \$statements_with_temp_tables	一時テーブルを使用したステートメント
user_summary, x\$user_summary	ユーザーステートメントおよび接続アクティビティ
user_summary_by_file_io, x\$user_summary_by_file_io	ユーザー別にグループ化したファイル I/O,
user_summary_by_file_io_type, x \$user_summary_by_file_io_type	ユーザーおよびイベント別にグループ化されたファイル I/O,
user_summary_by_stages, x\$user_summary_by_stages	ユーザー別にグループ化されたステージイベント
user_summary_by_statement_latency, x \$user_summary_by_statement_latency	ユーザー別にグループ化されたステートメント統計
user_summary_by_statement_type, x \$user_summary_by_statement_type	実行されたステートメント (ユーザーおよびステートメント別にグループ化)
version	現在の sys スキーマおよび MySQL サーバーのバージョン
wait_classes_global_by_avg_latency, x \$wait_classes_global_by_avg_latency	待機クラスの平均待機時間 (イベントクラス別にグループ化)
wait_classes_global_by_latency, x \$wait_classes_global_by_latency	待機クラスの合計待機時間 (イベントクラス別にグループ化)
waits_by_host_by_latency, x\$waits_by_host_by_latency	ホストおよびイベント別にグループ化された待機イベント
waits_by_user_by_latency, x\$waits_by_user_by_latency	ユーザーおよびイベント別にグループ化された待機イベント
waits_global_by_latency, x\$waits_global_by_latency	イベント別にグループ化された待機イベント
x\$ps_digest_95th_percentile_by_avg_us	95th-percentile ビューのヘルパービュー
x\$ps_digest_avg_latency_distribution	95th-percentile ビューのヘルパービュー
x\$ps_schema_table_statistics_io	テーブル統計ビューのヘルパービュー

ビュー名	説明
x\$schema_flattened_keys	schema_redundant_indexes のヘルパービュー

表 28.3 sys スキーマストアードプロシージャ

プロシージャ名	説明
create_synonym_db()	スキーマのシノニムの作成
diagnostics()	システム診断情報の収集
execute_prepared_stmt()	プリアドステートメントの実行
ps_setup_disable_background_threads()	バックグラウンドスレッドのインストールメンテーションの無効化
ps_setup_disable_consumer()	コンシューマの無効化
ps_setup_disable_instrument()	インストゥルメントの無効化
ps_setup_disable_thread()	スレッドのインストールメンテーションの無効化
ps_setup_enable_background_threads()	バックグラウンドスレッドのインストールメンテーションの有効化
ps_setup_enable_consumer()	コンシューマの有効化
ps_setup_enable_instrument()	インストゥルメントの有効化
ps_setup_enable_thread()	スレッドのインストールメンテーションの有効化
ps_setup_reload_saved()	保存されたパフォーマンススキーマ構成のリロード
ps_setup_reset_to_default()	保存されたパフォーマンススキーマ構成のリセット
ps_setup_save()	パフォーマンススキーマ構成の保存
ps_setup_show_disabled()	無効になっているパフォーマンススキーマ構成の表示
ps_setup_show_disabled_consumers()	無効になっているパフォーマンススキーマコンシューマを表示
ps_setup_show_disabled_instruments()	無効になっているパフォーマンススキーマインストゥルメントを表示
ps_setup_show_enabled()	有効なパフォーマンススキーマ構成を表示
ps_setup_show_enabled_consumers()	有効なパフォーマンススキーマコンシューマを表示
ps_setup_show_enabled_instruments()	有効なパフォーマンススキーマインストゥルメントを表示
ps_statement_avg_latency_histogram()	ステートメント待機時間ヒストグラムの表示
ps_trace_statement_digest()	ダイジェストのトレースパフォーマンススキーマインストールメンテーション
ps_trace_thread()	スレッドのパフォーマンススキーマデータのダンプ
ps_truncate_all_tables()	パフォーマンススキーマサマリーテーブルの切捨て
statement_performance_analyzer()	サーバーで実行されているステートメントのレポート
table_exists()	テーブルが存在するかどうか

表 28.4 sys スキーマストアードファンクション

関数名	説明
extract_schema_from_file_name()	ファイルパス名からスキーマ名を抽出
extract_table_from_file_name()	ファイルパス名からテーブル名を抽出
format_bytes()	バイト数を単位付きの値に変換
format_path()	パス名の data および temp-file ディレクトリをシンボリック値に置き換えます

関数名	説明
<code>format_statement()</code>	長いステートメントを固定長に切り捨てる
<code>format_time()</code>	ピコ秒値を単位付きの値に変換
<code>list_add()</code>	リストにアイテムを追加
<code>list_drop()</code>	リストからアイテムを削除
<code>ps_is_account_enabled()</code>	アカウントインツルメンテーションが有効かどうかを確認
<code>ps_is_consumer_enabled()</code>	コンシューマが有効かどうかを確認
<code>ps_is_instrument_default_enabled()</code>	インストゥルメントが有効かどうかを確認
<code>ps_is_instrument_default_timed()</code>	インストゥルメントが時間指定かどうかを確認
<code>ps_is_thread_instrumented()</code>	スレッドがインストゥルメントされているかどうかを確認
<code>ps_thread_account()</code>	スレッド ID のアカウントを返します
<code>ps_thread_id()</code>	接続 ID のスレッド ID を返します
<code>ps_thread_stack()</code>	スレッド ID のイベント情報を返します
<code>ps_thread_trx_info()</code>	スレッド ID のトランザクション情報を返します
<code>quote_identifier()</code>	引用識別子として文字列を返します
<code>sys_get_config()</code>	<code>sys</code> スキーマ構成オプションを返します
<code>version_major()</code>	MySQL サーバーのメジャーバージョン番号
<code>version_minor()</code>	MySQL サーバーのマイナーバージョン番号
<code>version_patch()</code>	MySQL サーバーパッチのリリースバージョン番号

28.4.2 sys スキーマのテーブルおよびトリガー

次の各セクションでは、`sys` のスキーマテーブルおよびトリガーについて説明します。

28.4.2.1 sys_config テーブル

このテーブルには、オプションごとに 1 行ずつ、`sys` スキーマ構成オプションが含まれています。このテーブルを更新して行われた構成変更は、クライアントセッションおよびサーバーの再起動後も保持されます。

`sys_config` テーブルには、次のカラムがあります:

- `variable`
構成オプション名。
- `value`
構成オプションの値。
- `set_time`
行に対する最新の変更のタイムスタンプ。
- `set_by`
行を最後に変更したアカウント。`sys` スキーマのインストール後に行が変更されていない場合、値は `NULL` です。

`sys_config` テーブルからの直接読取りの数を最小限に抑える効率的な方法として、このテーブルの値を使用する `sys` スキーマ関数は、対応する名前を持つユーザー定義変数 (同じ名前と `@sys.` 接頭辞を持つユーザー定義変数) をチェックします。(たとえば、`diagnostics.include_raw` オプションに対応する変数は `@sys.diagnostics.include_raw` です。) ユーザー定義変数が現在のセッションに存在し、`NULL` 以外の場合、関数は `sys_config` テーブルの値よりも優先してその値を使用します。それ以外の場合、関数はテーブルの値を読み取って使用します。後者の場合、コール側関数は

従来どおり、対応するユーザー定義変数をテーブルの値に設定するため、同じセッション内の構成オプションへのさらなる参照で変数が使用され、テーブルを再度読み取る必要はありません。

たとえば、`statement_truncate_len` オプションは、`format_statement()` 関数によって戻されるステートメントの最大長を制御します。デフォルトは 64 です。現在のセッションの値を一時的に 32 に変更するには、対応する `@sys.statement_truncate_len` ユーザー定義変数を設定します：

```
mysql> SET @stmt = 'SELECT variable, value, set_time, set_by FROM sys_config';
mysql> SELECT sys.format_statement(@stmt);
+-----+
| sys.format_statement(@stmt) |
+-----+
| SELECT variable, value, set_time, set_by FROM sys_config |
+-----+
mysql> SET @sys.statement_truncate_len = 32;
mysql> SELECT sys.format_statement(@stmt);
+-----+
| sys.format_statement(@stmt) |
+-----+
| SELECT variabl ... ROM sys_config |
+-----+
```

セッション内でのその後の `format_statement()` の起動では、テーブル (64) に格納されている値ではなく、ユーザー定義の変数値 (32) が引き続き使用されます。

ユーザー定義変数の使用を停止し、テーブルの値の使用に戻すには、セッション内で変数を `NULL` に設定します：

```
mysql> SET @sys.statement_truncate_len = NULL;
mysql> SELECT sys.format_statement(@stmt);
+-----+
| sys.format_statement(@stmt) |
+-----+
| SELECT variable, value, set_time, set_by FROM sys_config |
+-----+
```

または、現在のセッションを終了して (ユーザー定義変数が存在しなくなります)、新しいセッションを開始します。

`sys_config` テーブルのオプションとユーザー定義変数の間に記述されている従来の関係を利用して、セッションの終了時に終了する一時的な構成変更を行うことができます。ただし、ユーザー定義変数を設定してから、同じセッション内で対応するテーブルの値を変更した場合、`NULL` 以外の値を持つユーザー定義変数が存在するかぎり、変更されたテーブルの値はそのセッションでは使用されません。(変更されたテーブルの値は、ユーザー定義変数が割り当てられていない他のセッションで使用されます。)

次のリストでは、`sys_config` テーブルのオプションおよび対応するユーザー定義変数について説明します：

- `diagnostics.allow_i_s_tables`, `@sys.diagnostics.allow_i_s_tables`

このオプションが `ON` の場合、`diagnostics()` プロシージャは `INFORMATION_SCHEMA.TABLES` テーブルに対してテーブルスキャンを実行できます。多くのテーブルがある場合、これはコストがかかる可能性があります。デフォルトは `OFF` です。

- `diagnostics.include_raw`, `@sys.diagnostics.include_raw`

このオプションが `ON` の場合、`diagnostics()` プロシージャには `metrics` ビューのクエリーからの RAW 出力が含まれます。デフォルトは `OFF` です。

- `ps_thread_trx_info.max_length`, `@sys.ps_thread_trx_info.max_length`

`ps_thread_trx_info()` 関数によって生成される JSON 出力の最大長。デフォルトは 65535 です。

- `statement_performance_analyzer.limit`, `@sys.statement_performance_analyzer.limit`

組込み制限のないビューに対して返す行の最大数。(たとえば、`statements_with_runtimes_in_95th_percentile` ビューには 95 パーセントイルの平均実行時間を持つステートメントのみが返されるという意味で組込みの制限があります。) デフォルトは 100 です。

- `statement_performance_analyzer.view`, `@sys.statement_performance_analyzer.view`

`statement_performance_analyzer()` プロシージャ (`diagnostics()` プロシージャによって起動される) で使用されるカスタムクエリまたはビュー。オプション値に空白が含まれている場合は、クエリとして解釈されます。それ以外の場合は、パフォーマンススキーマ `events_statements_summary_by_digest` テーブルをクエリする既存のビューの名前である必要があります。 `statement_performance_analyzer.limit` 構成オプションが 0 より大きい場合、クエリまたはビュー定義に `LIMIT` 句を含めることはできません。デフォルトは `NULL` です (カスタムビューが定義されていません)。

- `statement_truncate_len`, `@sys.statement_truncate_len`

`format_statement()` 関数によって戻されるステートメントの最大長。長いステートメントはこの長さに切り捨てられます。デフォルトは 64 です。

その他のオプションは、`sys_config` テーブルに追加できます。たとえば、`diagnostics()` および `execute_prepared_stmt()` プロシージャでは、`debug` オプションが存在する場合にそれを使用しますが、デバッグ出力は通常、対応する `@sys.debug` ユーザー定義変数を設定することによって一時的にのみ有効になるため、このオプションはデフォルトで `sys_config` テーブルの一部ではありません。個々のセッションで変数を設定せずにデバッグ出力を有効にするには、テーブルにオプションを追加します:

```
mysql> INSERT INTO sys.sys_config (variable, value) VALUES('debug', 'ON');
```

テーブルのデバッグ設定を変更するには、次の 2 つの操作を行います。まず、テーブル自体の値を変更します:

```
mysql> UPDATE sys.sys_config
SET value = 'OFF'
WHERE variable = 'debug';
```

次に、現在のセッション内でプロシージャを起動する際に、テーブルの変更された値が使用されるようにするには、対応するユーザー定義変数を `NULL` に設定します:

```
mysql> SET @sys.debug = NULL;
```

28.4.2.2 sys_config_insert_set_user トリガー

`INSERT` ステートメントによって `sys_config` テーブルに追加された行の場合、`sys_config_insert_set_user` トリガーは `set_by` カラムを現行ユーザーに設定します。

28.4.2.3 sys_config_update_set_user トリガー

`sys_config` テーブルの `sys_config_update_set_user` トリガーは、`sys_config_insert_set_user` トリガーと似ていますが、`UPDATE` ステートメントの場合です。

28.4.3 sys スキーマビュー

次の各セクションでは、`sys` スキーマビューについて説明します。

`sys` スキーマには、様々な方法で「パフォーマンススキーマ」テーブルを要約する多数のビューが含まれています。これらのビューのほとんどはペアになっているため、ペアのメンバーは他のメンバーと同じ名前に `x$` 接頭辞を付けたものになります。たとえば、`host_summary_by_file_io` ビューでは、ホスト別にグループ化されたファイル I/O が要約され、ピコ秒からより読みやすい値 (単位付き) に変換された待機時間が表示されます

```
mysql> SELECT * FROM sys.host_summary_by_file_io;
+-----+-----+-----+
| host   | ios  | io_latency |
+-----+-----+-----+
| localhost | 67570 | 5.38 s   |
| background | 3468 | 4.18 s   |
+-----+-----+-----+
```

`x$host_summary_by_file_io` ビューには、同じデータが要約されますが、書式設定されていないピコ秒待機時間が表示されます:

```
mysql> SELECT * FROM sys.x$host_summary_by_file_io;
+-----+-----+-----+
| host   | ios  | io_latency |
+-----+-----+-----+
| localhost | 67574 | 5380678125144 |
```

```
| background | 3474 | 4758696829416 |  
+-----+-----+-----+-----+
```

x\$ 接頭辞のないビューは、わかりやすく読みやすい出力を提供することを目的としています。同じ値を RAW 形式で表示する x\$ 接頭辞を持つビューは、データに対して独自の処理を実行する他のツールでの使用を目的としています。

x\$ 接頭辞のないビューは、次の点で対応する x\$ ビューと異なります：

- バイト数は、`format_bytes()` を使用してサイズ単位でフォーマットされます。
- 時間値は、`format_time()` を使用して時間単位で書式設定されます。
- SQL ステートメントは、`format_statement()` を使用して最大表示幅に切り捨てられます。
- パス名は、`format_path()` を使用して短縮されます。

28.4.3.1 host_summary および x\$host_summary のビュー

これらのビューには、ホストごとにグループ化されたステートメントアクティビティ、ファイル I/O、および接続が要約されます。

host_summary ビューと x\$host_summary ビューには、次のカラムがあります：

- **host**
クライアントの接続元のホスト。基礎となる「パフォーマンススキーマ」テーブルの `HOST` カラムが `NULL` である行はバックグラウンドスレッド用とみなされ、`background` のホスト名でレポートされます。
- **statements**
ホストのステートメントの合計数。
- **statement_latency**
ホストの時間指定ステートメントの合計待機時間。
- **statement_avg_latency**
ホストの時間指定ステートメント当たりの平均待機時間。
- **table_scans**
ホストのテーブルスキャンの合計数。
- **file_ios**
ホストのファイル I/O イベントの合計数。
- **file_io_latency**
ホストの時間指定ファイル I/O イベントの合計待機時間。
- **current_connections**
ホストの現在の接続数。
- **total_connections**
ホストの合計の接続数。
- **unique_users**
ホストの個別ユーザーの数。
- **current_memory**
ホストに割り当てられている現在のメモリー量。

- [total_memory_allocated](#)

ホストに割り当てられたメモリーの合計量。

28.4.3.2 [host_summary_by_file_io](#) および [x\\$host_summary_by_file_io](#) のビュー

これらのビューは、ホストごとにグループ化されたファイル I/O を要約します。デフォルトでは、行は合計ファイル I/O レイテンシの降順でソートされます。

[host_summary_by_file_io](#) ビューと [x\\$host_summary_by_file_io](#) ビューには、次のカラムがあります:

- [host](#)

クライアントの接続元のホスト。基礎となる「パフォーマンススキーマ」テーブルの [HOST](#) カラムが [NULL](#) である行はバックグラウンドスレッド用とみなされ、[background](#) のホスト名でレポートされます。

- [ios](#)

ホストのファイル I/O イベントの合計数。

- [io_latency](#)

ホストの時間指定ファイル I/O イベントの合計待機時間。

28.4.3.3 [host_summary_by_file_io_type](#) および [x\\$host_summary_by_file_io_type](#) ビュー

これらのビューには、ホストおよびイベントタイプ別にグループ化されたファイル I/O が要約されます。デフォルトでは、行はホストでソートされ、合計 I/O レイテンシが降順になります。

[host_summary_by_file_io_type](#) ビューと [x\\$host_summary_by_file_io_type](#) ビューには、次のカラムがあります:

- [host](#)

クライアントの接続元のホスト。基礎となる「パフォーマンススキーマ」テーブルの [HOST](#) カラムが [NULL](#) である行はバックグラウンドスレッド用とみなされ、[background](#) のホスト名でレポートされます。

- [event_name](#)

ファイル I/O イベント名。

- [total](#)

ホストで発生したファイル I/O イベントの合計数。

- [total_latency](#)

ホストのファイル I/O イベントの発生時間の合計待機時間。

- [max_latency](#)

ホストのファイル I/O イベントの時間指定発生の最大単一待機時間。

28.4.3.4 [host_summary_by_stages](#) および [x\\$host_summary_by_stages](#) のビュー

これらのビューは、ホストごとにグループ化されたステートメントステージを要約します。デフォルトでは、行はホストおよび合計レイテンシの降順でソートされます。

[host_summary_by_stages](#) ビューと [x\\$host_summary_by_stages](#) ビューには、次のカラムがあります:

- [host](#)

クライアントの接続元のホスト。基礎となる「パフォーマンススキーマ」テーブルの [HOST](#) カラムが [NULL](#) である行はバックグラウンドスレッド用とみなされ、[background](#) のホスト名でレポートされます。

- [event_name](#)

ステージイベント名。

- [total](#)
ホストのステージイベントの発生の合計数。
- [total_latency](#)
ホストのステージイベントの時間指定発生の合計待機時間。
- [avg_latency](#)
ホストのステージイベントの発生時間ごとの平均待機時間。

28.4.3.5 `host_summary_by_statement_latency` および `x$host_summary_by_statement_latency` ビュー

これらのビューには、ホストごとにグループ化されたステートメント全体の統計が要約されます。デフォルトでは、行は合計レイテンシの降順でソートされます。

`host_summary_by_statement_latency` ビューと `x$host_summary_by_statement_latency` ビューには、次のカラムがあります:

- [ホスト](#)
クライアントの接続元のホスト。基礎となる「パフォーマンススキーマ」テーブルの `HOST` カラムが `NULL` である行はバックグラウンドスレッド用とみなされ、`background` のホスト名でレポートされます。
- [total](#)
ホストのステートメントの合計数。
- [total_latency](#)
ホストの時間指定ステートメントの合計待機時間。
- [max_latency](#)
ホストの時間指定ステートメントの最大単一待機時間。
- [lock_latency](#)
ホストの時間指定ステートメントによるロックを待機する合計時間。
- [rows_sent](#)
ホストのステートメントによって返された行の合計数。
- [rows_examined](#)
ホストのステートメントによってストレージエンジンから読み取られた行の合計数。
- [rows_affected](#)
ホストのステートメントの影響を受ける行の合計数。
- [full_scans](#)
ホストのステートメントによる全テーブルスキャンの合計数。

28.4.3.6 `host_summary_by_statement_type` および `x$host_summary_by_statement_type` のビュー

これらのビューには、実行されたステートメントに関する情報が、ホストおよびステートメントタイプ別にグループ化されて要約されます。デフォルトでは、行はホストおよび合計レイテンシの降順でソートされます。

`host_summary_by_statement_type` ビューと `x$host_summary_by_statement_type` ビューには、次のカラムがあります:

- `host`

クライアントの接続元のホスト。基礎となる「パフォーマンススキーマ」テーブルの `HOST` カラムが `NULL` である行はバックグラウンドスレッド用とみなされ、`background` のホスト名でレポートされます。

- `statement`

ステートメントイベント名の最終コンポーネント。

- `total`

ホストに対するステートメントイベントの発生の合計数。

- `total_latency`

ホストに対するステートメントイベントの発生時間の合計待機時間。

- `max_latency`

ホストのステートメントイベントの時間指定発生の最大単一待機時間。

- `lock_latency`

ホストのステートメントイベントの発生時間によるロックの合計待機時間。

- `rows_sent`

ホストのステートメントイベントの発生によって返された行の合計数。

- `rows_examined`

ホストのステートメントイベントの発生によってストレージエンジンから読み取られた行の合計数。

- `rows_affected`

ホストのステートメントイベントの発生によって影響を受ける行の合計数。

- `full_scans`

ホストのステートメントイベントの発生による全テーブルスキャンの合計数。

28.4.3.7 `innodb_buffer_stats_by_schema` および `x$innodb_buffer_stats_by_schema` のビュー

これらのビューには、スキーマ別にグループ化された `INFORMATION_SCHEMA.INNODB_BUFFER_PAGE` テーブルの情報がまとめられています。デフォルトでは、行は降順のバッファサイズでソートされます。

警告

`INNODB_BUFFER_PAGE` テーブルにアクセスするビューをクエリーすると、パフォーマンスに影響する可能性があります。パフォーマンスへの影響を認識し、許容できると判断した場合を除き、本番システムでこれらのビューをクエリーしないでください。本番システムのパフォーマンスへの影響を回避するには、調査する問題を再現し、テストインスタンスのバッファプール統計をクエリーします。

`innodb_buffer_stats_by_schema` ビューと `x$innodb_buffer_stats_by_schema` ビューには、次のカラムがあります:

- `object_schema`

オブジェクトのスキーマ名。テーブルが `InnoDB` ストレージエンジンに属している場合は `InnoDB System`。

- `allocated`

スキーマに割り当てられた合計バイト数。

- [data](#)

スキーマに割り当てられたデータバイトの合計数。

- [pages](#)

スキーマに割り当てられたページの合計数。

- [pages_hashed](#)

スキーマに割り当てられたハッシュページの合計数。

- [pages_old](#)

スキーマに割り当てられた古いページの合計数。

- [rows_cached](#)

スキーマのキャッシュされた行の合計数。

28.4.3.8 innodb_buffer_stats_by_table および x\$innodb_buffer_stats_by_table のビュー

これらのビューには、スキーマおよびテーブル別にグループ化された [INFORMATION_SCHEMA](#) [INNODB_BUFFER_PAGE](#) テーブルの情報がまとめられています。デフォルトでは、行は降順のバッファサイズでソートされます。

警告

[INNODB_BUFFER_PAGE](#) テーブルにアクセスするビューをクエリーすると、パフォーマンスに影響する可能性があります。パフォーマンスへの影響を認識し、許容できると判断した場合を除き、本番システムでこれらのビューをクエリーしないでください。本番システムのパフォーマンスへの影響を回避するには、調査する問題を再現し、テストインスタンスのバッファプール統計をクエリーします。

[innodb_buffer_stats_by_table](#) ビューと [x\\$innodb_buffer_stats_by_table](#) ビューには、次のカラムがあります:

- [object_schema](#)

オブジェクトのスキーマ名。テーブルが [InnoDB](#) ストレージエンジンに属している場合は [InnoDB System](#)。

- [object_name](#)

テーブル名

- [allocated](#)

テーブルに割り当てられた合計バイト数。

- [data](#)

テーブルに割り当てられたデータバイト数。

- [pages](#)

テーブルに割り当てられたページの合計数。

- [pages_hashed](#)

テーブルに割り当てられたハッシュページの数。

- [pages_old](#)

テーブルに割り当てられた古いページの数。

- [rows_cached](#)

テーブルのキャッシュされた行数。

28.4.3.9 innodb_lock_waits および x\$innodb_lock_waits のビュー

これらのビューには、トランザクションが待機している InnoDB ロックの概要が示されます。デフォルトでは、行はロック期間の降順でソートされます。

[innodb_lock_waits](#) ビューと [x\\$innodb_lock_waits](#) ビューには、次のカラムがあります:

- [wait_started](#)

ロック待機が開始された時刻。

- [wait_age](#)

ロックが待機された期間 (TIME 値)。

- [wait_age_secs](#)

ロックが待機された時間 (秒)。

- [locked_table_schema](#)

ロックされたテーブルを含むスキーマ。

- [locked_table_name](#)

ロックされたテーブルの名前。

- [locked_table_partition](#)

ロックされたパーティションの名前 (存在する場合)。それ以外の場合は [NULL](#)。

- [locked_table_subpartition](#)

ロックされたサブパーティションの名前 (存在する場合)。それ以外の場合は [NULL](#)。

- [locked_index](#)

ロックされたインデックスの名前。

- [locked_type](#)

待機中のロックのタイプ。

- [waiting_trx_id](#)

待機中のトランザクションの ID。

- [waiting_trx_started](#)

待機中のトランザクションが開始された時刻。

- [waiting_trx_age](#)

待機中のトランザクションが待機していた時間 (TIME 値)。

- [waiting_trx_rows_locked](#)

待機中のトランザクションによってロックされた行数。

- [waiting_trx_rows_modified](#)

待機中のトランザクションによって変更された行数。

- [waiting_pid](#)
待機中のトランザクションのプロセスリスト ID。
- [waiting_query](#)
ロックを待機しているステートメント。
- [waiting_lock_id](#)
待機中のロックの ID。
- [waiting_lock_mode](#)
待機ロックのモード。
- [blocking_trx_id](#)
待機ロックをブロックしているトランザクションの ID。
- [blocking_pid](#)
ブロッキングトランザクションのプロセスリスト ID。
- [blocking_query](#)
ブロックしているトランザクションが実行しているステートメント。ブロッキングクエリーを発行したセッションがアイドル状態になった場合、このフィールドは NULL を報告します。詳細は、[発行セッションがアイドル状態になった後のブロッキングクエリーの識別](#)を参照してください。
- [blocking_lock_id](#)
待機中のロックをブロックしているロックの ID。
- [blocking_lock_mode](#)
待機中のロックをブロックしているロックのモード。
- [blocking_trx_started](#)
ブロッキングトランザクションが開始された時刻。
- [blocking_trx_age](#)
ブロックしているトランザクションが実行されている期間 (TIME 値)。
- [blocking_trx_rows_locked](#)
ブロックしているトランザクションによってロックされた行数。
- [blocking_trx_rows_modified](#)
ブロックしているトランザクションによって変更された行数。
- [sql_kill_blocking_query](#)
ブロッキングステートメントを強制終了するために実行する KILL ステートメント。
- [sql_kill_blocking_connection](#)
ブロッキングステートメントを実行しているセッションを強制終了するために実行する KILL ステートメント。

28.4.3.10 io_by_thread_by_latency および x\$io_by_thread_by_latency のビュー

これらのビューは、I/O コンシューマを要約して、I/O を待機している時間をスレッド別にグループ化して表示します。デフォルトでは、行は合計 I/O レイテンシの降順でソートされます。

`io_by_thread_by_latency` ビューと `x$io_by_thread_by_latency` ビューには、次のカラムがあります:

- `user`
フォアグラウンドスレッドの場合、スレッドに関連付けられたアカウント。バックグラウンドスレッドの場合は、スレッド名。
- `total`
スレッドの I/O イベントの合計数。
- `total_latency`
スレッドの時間指定 I/O イベントの合計待機時間。
- `min_latency`
スレッドの時間指定 I/O イベントの最小単一待機時間。
- `avg_latency`
スレッドの時間指定 I/O イベント当たりの平均待機時間。
- `max_latency`
スレッドの時間指定 I/O イベントの最大単一待機時間。
- `thread_id`
スレッド ID。
- `processlist_id`
フォアグラウンドスレッドの場合、スレッドのプロセスリスト ID。バックグラウンドスレッドの場合、`NULL`。

28.4.3.11 `io_global_by_file_by_bytes` および `x$io_global_by_file_by_bytes` のビュー

これらのビューは、グローバル I/O コンシューマを要約して、ファイルごとにグループ化された I/O の量を表示します。デフォルトでは、行は合計 I/O (読取りおよび書込みバイト数) の降順でソートされます。

`io_global_by_file_by_bytes` ビューと `x$io_global_by_file_by_bytes` ビューには、次のカラムがあります:

- `file`
ファイルパス名。
- `count_read`
ファイルの読取りイベントの合計数。
- `total_read`
ファイルから読み取られた合計バイト数。
- `avg_read`
ファイルからの読取り当たりの平均バイト数。
- `count_write`
ファイルの書込みイベントの合計数。
- `total_written`
ファイルに書き込まれた合計バイト数。

- [avg_write](#)
ファイルへの書き込み当たりの平均バイト数。
- [total](#)
ファイルに対して読取りおよび書き込みが行われた合計バイト数。
- [write_pct](#)
書き込まれた I/O の合計バイト数の割合。

28.4.3.12 [io_global_by_file_by_latency](#) および [x\\$io_global_by_file_by_latency](#) のビュー

これらのビューは、グローバル I/O コンシューマを要約して、I/O を待機している時間をファイル別にグループ化して表示します。デフォルトでは、行は合計レイテンシの降順でソートされます。

[io_global_by_file_by_latency](#) ビューと [x\\$io_global_by_file_by_latency](#) ビューには、次のカラムがあります:

- [file](#)
ファイルパス名。
- [total](#)
ファイルの I/O イベントの合計数。
- [total_latency](#)
ファイルの時間指定 I/O イベントの合計待機時間。
- [count_read](#)
ファイルの読取り I/O イベントの合計数。
- [read_latency](#)
ファイルの時間指定読取り I/O イベントの合計待機時間。
- [count_write](#)
ファイルの書き込み I/O イベントの合計数。
- [write_latency](#)
ファイルの時間指定書き込み I/O イベントの合計待機時間。
- [count_misc](#)
ファイルの他の I/O イベントの合計数。
- [misc_latency](#)
ファイルの他の時間指定 I/O イベントの合計待機時間。

28.4.3.13 [io_global_by_wait_by_bytes](#) および [x\\$io_global_by_wait_by_bytes](#) のビュー

これらのビューには、グローバル I/O コンシューマの概要が示され、I/O を待機している時間がイベント別にグループ化されて表示されます。デフォルトでは、行は合計 I/O (読取りおよび書き込みバイト数) の降順でソートされます。

[io_global_by_wait_by_bytes](#) ビューと [x\\$io_global_by_wait_by_bytes](#) ビューには、次のカラムがあります:

- [event_name](#)
`wait/io/file/`接頭辞が削除された I/O イベント名。

- [total](#)
I/O イベントの発生の合計数。
- [total_latency](#)
I/O イベントの発生時間の合計待機時間。
- [min_latency](#)
I/O イベントの発生時間の最小単一待機時間。
- [avg_latency](#)
I/O イベントの発生時間ごとの平均待機時間。
- [max_latency](#)
I/O イベントの時間指定発生の最大単一待機時間。
- [count_read](#)
I/O イベントの読取りリクエストの数。
- [total_read](#)
I/O イベントに対して読み取られたバイト数。
- [avg_read](#)
I/O イベントの読取り当たりの平均バイト数。
- [count_write](#)
I/O イベントの書込みリクエストの数。
- [total_written](#)
I/O イベントに対して書き込まれたバイト数。
- [avg_written](#)
I/O イベントの書込み当たりの平均バイト数。
- [total_requested](#)
I/O イベントに対して読取りおよび書込みが行われた合計バイト数。

28.4.3.14 [io_global_by_wait_by_latency](#) および [x\\$io_global_by_wait_by_latency](#) ビュー

これらのビューには、グローバル I/O コンシューマの概要が示され、I/O を待機している時間がイベント別にグループ化されて表示されます。デフォルトでは、行は合計レイテンシの降順でソートされます。

[io_global_by_wait_by_latency](#) ビューと [x\\$io_global_by_wait_by_latency](#) ビューには、次のカラムがあります:

- [event_name](#)
[wait/io/file/](#)接頭辞が削除された I/O イベント名。
- [total](#)
I/O イベントの発生の合計数。
- [total_latency](#)
I/O イベントの発生時間の合計待機時間。

- [avg_latency](#)
I/O イベントの発生時間ごとの平均待機時間。
- [max_latency](#)
I/O イベントの時間指定発生最大の単一待機時間。
- [read_latency](#)
I/O イベントの時間指定読取り発生合計待機時間。
- [write_latency](#)
I/O イベントの時間指定書込み発生合計待機時間。
- [misc_latency](#)
I/O イベントの他の時間発生合計待機時間。
- [count_read](#)
I/O イベントの読取りリクエストの数。
- [total_read](#)
I/O イベントに対して読み取られたバイト数。
- [avg_read](#)
I/O イベントの読取り当たりの平均バイト数。
- [count_write](#)
I/O イベントの書込みリクエストの数。
- [total_written](#)
I/O イベントに対して書き込まれたバイト数。
- [avg_written](#)
I/O イベントの書込み当たりの平均バイト数。

28.4.3.15 latest_file_io および x\$latest_file_io のビュー

これらのビューには、ファイルおよびスレッド別にグループ化されたファイル I/O アクティビティが要約されます。デフォルトでは、行は最新の I/O から順にソートされます。

[latest_file_io](#) ビューと [x\\$latest_file_io](#) ビューには、次のカラムがあります:

- [スレッド](#)
フォアグラウンドスレッドの場合、スレッドに関連付けられたアカウント (TCP/IP 接続の場合はポート番号)。バックグラウンドスレッドの場合は、スレッド名とスレッド ID
- [file](#)
ファイルパス名。
- [latency](#)
ファイル I/O イベントの待機時間。
- [operation](#)

操作のタイプ。

- [requested](#)

ファイル I/O イベントに対してリクエストされたデータバイト数。

28.4.3.16 [memory_by_host_by_current_bytes](#) および [x\\$memory_by_host_by_current_bytes](#) ビュー

これらのビューには、ホストごとにグループ化されたメモリー使用量の概要が示されます。デフォルトでは、行は使用されているメモリー量の降順でソートされます。

[memory_by_host_by_current_bytes](#) ビューと [x\\$memory_by_host_by_current_bytes](#) ビューには、次のカラムがあります:

- [ホスト](#)

クライアントの接続元のホスト。基礎となる「パフォーマンススキーマ」テーブルの [HOST](#) カラムが [NULL](#) である行はバックグラウンドスレッド用とみなされ、[background](#) のホスト名でレポートされます。

- [current_count_used](#)

ホストに対してまだ解放されていない割当て済メモリーブロックの現在の数。

- [current_allocated](#)

ホストに対してまだ解放されていない割当て済バイトの現在の数。

- [current_avg_alloc](#)

ホストのメモリーブロック当たりの現在の割当てバイト数。

- [current_max_alloc](#)

ホストの現在の最大メモリー割当て (バイト単位)。

- [total_allocated](#)

ホストの合計メモリー割当て (バイト)。

28.4.3.17 [memory_by_thread_by_current_bytes](#) および [x\\$memory_by_thread_by_current_bytes](#) のビュー

これらのビューには、スレッドごとにグループ化されたメモリー使用量の概要が示されます。デフォルトでは、行は使用されているメモリー量の降順でソートされます。

[memory_by_thread_by_current_bytes](#) ビューと [x\\$memory_by_thread_by_current_bytes](#) ビューには、次のカラムがあります:

- [thread_id](#)

スレッド ID。

- [user](#)

スレッドユーザーまたはスレッド名。

- [current_count_used](#)

スレッドに対してまだ解放されていない割当て済メモリーブロックの現在の数。

- [current_allocated](#)

スレッドに対してまだ解放されていない、現在割り当てられているバイト数。

- [current_avg_alloc](#)

スレッドのメモリーブロック当たりの現在の割当てバイト数。

- [current_max_alloc](#)

スレッドの現在の最大メモリー割当て (バイト単位)。

- [total_allocated](#)

スレッドの合計メモリー割当て (バイト)。

28.4.3.18 [memory_by_user_by_current_bytes](#) および [x\\$memory_by_user_by_current_bytes](#) のビュー

これらのビューには、ユーザー別にグループ化されたメモリー使用量の概要が示されます。デフォルトでは、行は使用されているメモリー量の降順でソートされます。

[memory_by_user_by_current_bytes](#) ビューと [x\\$memory_by_user_by_current_bytes](#) ビューには、次のカラムがあります:

- [user](#)

クライアントユーザー名。基礎となる「パフォーマンススキーマ」テーブルの [USER](#) カラムが [NULL](#) である行はバックグラウンドスレッド用とみなされ、[background](#) のホスト名でレポートされます。

- [current_count_used](#)

ユーザーに対してまだ解放されていない割当て済メモリーブロックの現在の数。

- [current_allocated](#)

ユーザーに対してまだ解放されていない割当て済バイトの現在の数。

- [current_avg_alloc](#)

ユーザーに割り当てられているメモリーブロック当たりの現在のバイト数。

- [current_max_alloc](#)

ユーザーの現在の最大メモリー割当て (バイト)。

- [total_allocated](#)

ユーザーの合計メモリー割当て (バイト)。

28.4.3.19 [memory_global_by_current_bytes](#) および [x\\$memory_global_by_current_bytes](#) のビュー

これらのビューには、割当てタイプ (つまり、イベント別) 別にグループ化されたメモリー使用量が要約されます。デフォルトでは、行は使用されているメモリー量の降順でソートされます。

[memory_global_by_current_bytes](#) ビューと [x\\$memory_global_by_current_bytes](#) ビューには、次のカラムがあります:

- [event_name](#)

メモリーイベント名。

- [current_count](#)

イベントの発生の合計数。

- [current_alloc](#)

イベントに対してまだ解放されていない、現在割り当てられているバイト数。

- [current_avg_alloc](#)
イベントのメモリーブロック当たりの現在の割当てバイト数。
- [high_count](#)
イベントに割り当てられたメモリーブロック数の最高水位標。
- [high_alloc](#)
イベントに割り当てられるバイト数の最高水位標。
- [high_avg_alloc](#)
イベントに割り当てられたメモリーブロック当たりの平均バイト数の最高水位標。

28.4.3.20 `memory_global_total` および `x$memory_global_total` のビュー

これらのビューには、サーバー内の合計メモリー使用量のサマリーが表示されます。

`memory_global_total` ビューと `x$memory_global_total` ビューには、次のカラムがあります:

- [total_allocated](#)
サーバー内で割り当てられたメモリーの合計バイト数。

28.4.3.21 メトリックビュー

このビューには、MySQL サーバメトリックの概要が示され、変数名、値、タイプおよびそれらが有効かどうかが表示されます。デフォルトでは、行は変数の型と名前ですべてソートされます。

`metrics` ビューには、次の情報が含まれます:

- パフォーマンススキーマ `global_status` テーブルのグローバルステータス変数
- `INFORMATION_SCHEMA INNODB_METRICS` テーブルからの InnoDB メトリック
- パフォーマンススキーマメモリーインストゥルメンテーションに基づく、現在および合計のメモリー割り当て
- 現在の時刻 (人間が読める形式と Unix タイムスタンプ形式)

`global_status` テーブルと `INNODB_METRICS` テーブルの間には、`metrics` ビューによって排除される情報の重複がいくつかあります。

`metrics` ビューには、次のカラムがあります:

- [Variable_name](#)
メトリック名。メトリックタイプによって、名前の取得元のソースが決まります:
 - グローバルステータス変数用: `global_status` テーブルの `VARIABLE_NAME` カラム
 - InnoDB メトリックの場合: `INNODB_METRICS` テーブルの `NAME` カラム
 - その他のメトリック用: ビュー提供の説明文字列
- [Variable_value](#)
メトリック値。メトリックタイプによって、値の取得元のソースが決まります:
 - グローバルステータス変数用: `global_status` テーブルの `VARIABLE_VALUE` カラム
 - InnoDB メトリックの場合: `INNODB_METRICS` テーブルの `COUNT` カラム

- メモリーメトリック用: パフォーマンススキーマ `memory_summary_global_by_event_name` テーブルの関連するカラム
- 現在の時刻: `NOW(3)` または `UNIX_TIMESTAMP(NOW(3))` の値
- **Type**
メトリックタイプ:
 - グローバルステータス変数用: `Global Status`
 - InnoDB メトリックの場合: InnoDB Metrics - `%(%)` は `INNODB_METRICS` テーブルの `SUBSYSTEM` カラムの値に置き換えられます)
 - メモリーメトリック用: `Performance Schema`
 - 現在の時刻: `System Time`
- **Enabled**
メトリックが有効かどうか:
 - グローバルステータス変数用: `YES`
 - InnoDB メトリックの場合: `INNODB_METRICS` テーブルの `STATUS` カラムが `enabled` の場合は `YES`、それ以外の場合は `NO`
 - メモリーメトリック用: `NO`、`YES` または `PARTIAL` (現在、`PARTIAL` はメモリーメトリックに対してのみ発生し、すべての `memory/%` インストゥルメントが有効ではないことを示します。パフォーマンススキーマメモリーインストゥルメントは常に有効です)
 - 現在の時刻: `YES`

28.4.3.22 processlist ビューと x\$processlist ビュー

MySQL プロセスリストには、サーバー内で実行されているスレッドのセットによって現在実行されている操作が示されます。 `processlist` および `x$processlist` ビューには、プロセス情報が要約されています。これらは、`SHOW PROCESSLIST` ステートメントおよび `INFORMATION_SCHEMA_PROCESSLIST` テーブルよりも完全な情報を提供し、非ブロック化も行います。デフォルトでは、行はプロセス時間の降順および待機時間の降順でソートされます。プロセス情報ソースの比較については、[プロセス情報のソース](#) を参照してください。

ここでのカラムの説明は簡単です。詳細は、[セクション27.12.19.10「スレッドテーブル」](#) のパフォーマンススキーマ `threads` テーブルの説明を参照してください。

`processlist` ビューと `x$processlist` ビューには、次のカラムがあります:

- `thd_id`
スレッド ID。
- `conn_id`
接続 ID。
- `user`
スレッドユーザーまたはスレッド名。
- `db`
スレッドのデフォルトのデータベース、何もなければ `NULL`。
- `command`

フォアグラウンドスレッドの場合、スレッドがクライアントのかわりに実行しているコマンドのタイプ、またはセッションがアイドル状態の場合は [Sleep](#)。

- [state](#)
スレッドが行なっていることを示すアクション、イベント、または状態。
- [time](#)
スレッドが現在の状態になってからの秒数。
- [current_statement](#)
スレッドが実行しているステートメント、またはそれがどのステートメントも実行していない場合は `NULL`。
- [statement_latency](#)
ステートメントが実行されている期間。
- [progress](#)
進捗レポートをサポートするステージで完了した作業の割合。 [セクション28.3「sys スキーマ進捗レポート」](#) を参照してください。
- [lock_latency](#)
現在のステートメントによるロックの待機に費やされた時間。
- [rows_examined](#)
現在のステートメントによってストレージエンジンから読み取られた行数。
- [rows_sent](#)
現行のステートメントによって戻された行数。
- [rows_affected](#)
現在のステートメントの影響を受ける行数。
- [tmp_tables](#)
現在のステートメントによって作成された内部インメモリー一時テーブルの数。
- [tmp_disk_tables](#)
現在のステートメントによって作成されたディスク上の内部一時テーブルの数。
- [full_scan](#)
現在のステートメントによって実行された全テーブルスキャンの数。
- [last_statement](#)
スレッドによって実行された最後のステートメント (現在実行中のステートメントまたは待機がない場合)。
- [last_statement_latency](#)
最後のステートメントが実行された時間。
- [current_memory](#)
スレッドによって割り当てられたバイト数。
- [last_wait](#)

スレッドの最新の待機イベントの名前。

- [last_wait_latency](#)

スレッドの最新の待機イベントの待機時間。

- [source](#)

イベントを生成したインストゥルメントされたコードを含むソースファイルおよび行番号。

- [trx_latency](#)

スレッドの現在のトランザクションの待機時間。

- [trx_state](#)

スレッドの現在のトランザクションの状態。

- [trx_autocommit](#)

現在のトランザクションの開始時に自動コミットモードが有効になっていたかどうか。

- [pid](#)

クライアントプロセス ID。

- [program_name](#)

クライアントプログラム名。

28.4.3.23 ps_check_lost_instrumentation ビュー

このビューは、失われたパフォーマンススキーマインストゥルメントに関する情報を返し、パフォーマンススキーマがすべての実行時データをモニターできないかどうかを示します。

[ps_check_lost_instrumentation](#) ビューには、次のカラムがあります:

- [variable_name](#)

失われたインストゥルメントのタイプを示すパフォーマンススキーマのステータス変数名。

- [variable_value](#)

失われたインストゥルメントの数。

28.4.3.24 schema_auto_increment_columns ビュー

このビューは、[AUTO_INCREMENT](#) カラムを持つテーブルを示し、現在のカラム値と最大カラム値、使用率 (使用可能な値に対する使用率) など、これらのカラムに関する情報を提供します。デフォルトでは、行は使用率と最大カラム値の降順でソートされます。

これらのスキーマのテーブルはビュー出力から除外されます: [mysql](#), [sys](#), [INFORMATION_SCHEMA](#), [performance_schema](#)。

[schema_auto_increment_columns](#) ビューには、次のカラムがあります:

- [table_schema](#)

テーブルを含むスキーマ。

- [table_name](#)

[AUTO_INCREMENT](#) カラムを含むテーブル。

- `column_name`
AUTO_INCREMENT カラムの名前。
- `data_type`
カラムのデータ型。
- `column_type`
カラムのカラムタイプ。これは、データ型に他の情報を加えたものです。たとえば、`bigint(20) unsigned` カラムタイプのカラムの場合、データ型は単なる `bigint` です。
- `is_signed`
カラムタイプが符号付きかどうか。
- `is_unsigned`
カラムの型が符号なしかどうか。
- `max_value`
カラムに許可される最大値。
- `auto_increment`
カラムの現在の AUTO_INCREMENT 値。
- `auto_increment_ratio`
カラムに許可された値に使用されるの比率。これは、値の順序が「使用済」である量を示します。

28.4.3.25 `schema_index_statistics` および `x$schema_index_statistics` のビュー

これらのビューは、インデックス統計を提供します。デフォルトでは、行は合計インデックスレイテンシの降順でソートされます。

`schema_index_statistics` ビューと `x$schema_index_statistics` ビューには、次のカラムがあります:

- `table_schema`
テーブルを含むスキーマ。
- `table_name`
インデックスを含むテーブル。
- `index_name`
インデックスの名前。
- `rows_selected`
インデックスを使用して読み取られた行の合計数。
- `select_latency`
インデックスを使用した時間読取りの合計待機時間。
- `rows_inserted`
インデックスに挿入された行の合計数。
- `insert_latency`

インデックスへの時間挿入の合計待機時間。

- [rows_updated](#)

インデックスで更新された行の合計数。

- [update_latency](#)

インデックス内の時間指定更新の合計待機時間。

- [rows_deleted](#)

インデックスから削除された行の合計数。

- [delete_latency](#)

インデックスからの時間付き削除の合計待機時間。

28.4.3.26 [schema_object_overview](#) ビュー

このビューには、各スキーマ内のオブジェクトのタイプがまとめられています。デフォルトでは、行はスキーマおよびオブジェクトタイプでソートされます。

注記

多数のオブジェクトを持つ MySQL インスタンスでは、このビューの実行に時間がかかる場合があります。

[schema_object_overview](#) ビューには、次のカラムがあります:

- [db](#)

スキーマ名。

- [object_type](#)

オブジェクトタイプ: [BASE TABLE](#)、[INDEX \(index_type\)](#)、[EVENT](#)、[FUNCTION](#)、[PROCEDURE](#)、[TRIGGER](#)、[VIEW](#)。

- [count](#)

指定されたタイプのスキーマ内のオブジェクトの数。

28.4.3.27 [schema_redundant_indexes](#) および [x\\$schema_flattened_keys](#) のビュー

[schema_redundant_indexes](#) ビューには、他のインデックスを複製するインデックス、またはそれらによって冗長化されたインデックスが表示されます。[x\\$schema_flattened_keys](#) ビューは、[schema_redundant_indexes](#) のヘルパービューです。

次のカラムの説明では、冗長インデックスを冗長化するインデックスが主要なインデックスです。

[schema_redundant_indexes](#) ビューには、次のカラムがあります:

- [table_schema](#)

テーブルを含むスキーマ。

- [table_name](#)

インデックスを含むテーブル。

- [redundant_index_name](#)

冗長インデックスの名前。

- [redundant_index_columns](#)
冗長インデックスのカラムの名前。
- [redundant_index_non_unique](#)
冗長インデックス内の一意でないカラムの数。
- [dominant_index_name](#)
主要なインデックスの名前。
- [dominant_index_columns](#)
主要インデックスのカラムの名前。
- [dominant_index_non_unique](#)
主要インデックス内の一意でないカラムの数。
- [subpart_exists](#)
インデックスがカラムの一部のみをインデックス付けするかどうか。
- [sql_drop_index](#)
冗長インデックスを削除するために実行するステートメント。

[x\\$schema_flattened_keys](#) ビューには、次のカラムがあります:

- [table_schema](#)
テーブルを含むスキーマ。
- [table_name](#)
インデックスを含むテーブル。
- [index_name](#)
インデックス名。
- [non_unique](#)
インデックス内の一意でないカラムの数。
- [subpart_exists](#)
インデックスがカラムの一部のみをインデックス付けするかどうか。
- [index_columns](#)
インデックス内のカラムの名前。

28.4.3.28 [schema_table_lock_waits](#) および [x\\$schema_table_lock_waits](#) のビュー

これらのビューには、メタデータロックで待機中にブロックされているセッションとブロックされているセッションが表示されます。

ここでのカラムの説明は簡単です。詳細は、[セクション27.12.13.3「metadata_locks テーブル」](#) のパフォーマンススキーマ [metadata_locks](#) テーブルの説明を参照してください。

[schema_table_lock_waits](#) ビューと [x\\$schema_table_lock_waits](#) ビューには、次のカラムがあります:

- [object_schema](#)

ロックするオブジェクトを含むスキーマ。

- [object_name](#)
インストールされたオブジェクトの名前。
- [waiting_thread_id](#)
ロックを待機中のスレッドのスレッド ID。
- [waiting_pid](#)
ロックを待機しているスレッドのプロセスリスト ID。
- [waiting_account](#)
ロックを待機しているセッションに関連付けられているアカウント。
- [waiting_lock_type](#)
待機中のロックのタイプ。
- [waiting_lock_duration](#)
待機中のロックが待機している時間。
- [waiting_query](#)
ロックを待機しているステートメント。
- [waiting_query_secs](#)
ステートメントが待機している時間 (秒)。
- [waiting_query_rows_affected](#)
ステートメントに影響を受けた行数。
- [waiting_query_rows_examined](#)
ステートメントによってストレージエンジンから読み取られた行数。
- [blocking_thread_id](#)
待機中のロックをブロックしているスレッドのスレッド ID。
- [blocking_pid](#)
待機中のロックをブロックしているスレッドのプロセスリスト ID。
- [blocking_account](#)
待機ロックをブロックしているスレッドに関連付けられたアカウント。
- [blocking_lock_type](#)
待機中のロックをブロックしているロックのタイプ。
- [blocking_lock_duration](#)
ブロッキングロックが保持されている期間。
- [sql_kill_blocking_query](#)
ブロッキングステートメントを強制終了するために実行する [KILL](#) ステートメント。

- [sql_kill_blocking_connection](#)

ブロッキングステートメントを実行しているセッションを強制終了するために実行する [KILL](#) ステートメント。

28.4.3.29 [schema_table_statistics](#) および [x\\$schema_table_statistics](#) のビュー

これらのビューには、テーブルの統計がまとめられています。デフォルトでは、行は合計待機時間の降順 (競合が最も多いテーブルが最初) でソートされます。

これらのビューはヘルパービュー [x\\$ps_schema_table_statistics_io](#) を使用します。

[schema_table_statistics](#) ビューと [x\\$schema_table_statistics](#) ビューには、次のカラムがあります:

- [table_schema](#)
テーブルを含むスキーマ。
- [table_name](#)
テーブル名
- [total_latency](#)
テーブルの時間指定 I/O イベントの合計待機時間。
- [rows_fetched](#)
テーブルから読み取られた行の合計数。
- [fetch_latency](#)
テーブルの時間指定読取り I/O イベントの合計待機時間。
- [rows_inserted](#)
テーブルに挿入された行の合計数。
- [insert_latency](#)
テーブルの時間指定挿入 I/O イベントの合計待機時間。
- [rows_updated](#)
テーブルで更新された行の合計数。
- [update_latency](#)
テーブルの時間指定更新 I/O イベントの合計待機時間。
- [rows_deleted](#)
テーブルから削除された行の合計数。
- [delete_latency](#)
テーブルの時間指定削除 I/O イベントの合計待機時間。
- [io_read_requests](#)
テーブルに対する読取りリクエストの合計数。
- [io_read](#)
テーブルから読み取られた合計バイト数。
- [io_read_latency](#)

テーブルからの読取りの合計待機時間。

- [io_write_requests](#)

テーブルに対する書き込みリクエストの合計数。

- [io_write](#)

テーブルに書き込まれた合計バイト数。

- [io_write_latency](#)

テーブルへの書き込みの合計待機時間。

- [io_misc_requests](#)

テーブルに対するその他の I/O リクエストの合計数。

- [io_misc_latency](#)

テーブルに対するその他の I/O リクエストの合計待機時間。

28.4.3.30 [schema_table_statistics_with_buffer](#) および [x\\$schema_table_statistics_with_buffer](#) のビュー

これらのビューは、[InnoDB](#) バッファプール統計を含むテーブル統計を要約します。デフォルトでは、行は合計待機時間の降順 (競合が最も多いテーブルが最初) でソートされます。

これらのビューはヘルパービュー [x\\$ps_schema_table_statistics_io](#) を使用します。

[schema_table_statistics_with_buffer](#) ビューと [x\\$schema_table_statistics_with_buffer](#) ビューには、次のカラムがあります:

- [table_schema](#)

テーブルを含むスキーマ。

- [table_name](#)

テーブル名

- [rows_fetched](#)

テーブルから読み取られた行の合計数。

- [fetch_latency](#)

テーブルの時間指定読取り I/O イベントの合計待機時間。

- [rows_inserted](#)

テーブルに挿入された行の合計数。

- [insert_latency](#)

テーブルの時間指定挿入 I/O イベントの合計待機時間。

- [rows_updated](#)

テーブルで更新された行の合計数。

- [update_latency](#)

テーブルの時間指定更新 I/O イベントの合計待機時間。

- [rows_deleted](#)
テーブルから削除された行の合計数。
- [delete_latency](#)
テーブルの時間指定削除 I/O イベントの合計待機時間。
- [io_read_requests](#)
テーブルに対する読取りリクエストの合計数。
- [io_read](#)
テーブルから読み取られた合計バイト数。
- [io_read_latency](#)
テーブルからの読取りの合計待機時間。
- [io_write_requests](#)
テーブルに対する書込みリクエストの合計数。
- [io_write](#)
テーブルに書き込まれた合計バイト数。
- [io_write_latency](#)
テーブルへの書込みの合計待機時間。
- [io_misc_requests](#)
テーブルに対するその他の I/O リクエストの合計数。
- [io_misc_latency](#)
テーブルに対するその他の I/O リクエストの合計待機時間。
- [innodb_buffer_allocated](#)
テーブルに割り当てられた InnoDB バッファバイトの合計数。
- [innodb_buffer_data](#)
テーブルに割り当てられた InnoDB データバイトの合計数。
- [innodb_buffer_free](#)
テーブル ([innodb_buffer_allocated](#) - [innodb_buffer_data](#)) に割り当てられた InnoDB 非データバイトの合計数。
- [innodb_buffer_pages](#)
テーブルに割り当てられた InnoDB ページの合計数。
- [innodb_buffer_pages_hashed](#)
テーブルに割り当てられた InnoDB ハッシュページの合計数。
- [innodb_buffer_pages_old](#)
テーブルに割り当てられた InnoDB の古いページの合計数。
- [innodb_buffer_rows_cached](#)
テーブルに対してキャッシュされた InnoDB 行の合計数。

28.4.3.31 schema_tables_with_full_table_scans および x\$schema_tables_with_full_table_scans のビュー

これらのビューには、全テーブルスキャンでアクセスされているテーブルが表示されます。デフォルトでは、行はスキャンされた降順の行でソートされます。

[schema_tables_with_full_table_scans](#) ビューと [x\\$schema_tables_with_full_table_scans](#) ビューには、次のカラムがあります:

- [object_schema](#)
スキーマ名。
- [object_name](#)
テーブル名
- [rows_full_scanned](#)
テーブルの全体スキャンでスキャンされた行の合計数。
- [latency](#)
テーブルの全スキャンの合計待機時間。

28.4.3.32 schema_unused_indexes ビュー

これらのビューには、使用されていないことを示すイベントがないインデックスが表示されます。デフォルトでは、行はスキーマおよびテーブルでソートされます。

このビューは、サーバーが稼働しており、ワークロードが代表的であるほど長く処理されている場合に最も役立ちます。そうしないと、このビューにインデックスが存在しても意味がない場合があります。

[schema_unused_indexes](#) ビューには、次のカラムがあります:

- [object_schema](#)
スキーマ名。
- [object_name](#)
テーブル名
- [index_name](#)
未使用のインデックス名。

28.4.3.33 セッションおよび x\$session ビュー

これらのビューは、[processlist](#) および [x\\$processlist](#) に似ていますが、ユーザーセッションのみを表示するようにバックグラウンドプロセスをフィルタで除外します。カラムの説明は、[セッション28.4.3.22「processlist ビューと x\\$processlist ビュー」](#) を参照してください。

28.4.3.34 session_ssl_status ビュー

このビューには、接続ごとに SSL バージョン、暗号および再利用された SSL セッションの数が表示されます。

[session_ssl_status](#) ビューには、次のカラムがあります:

- [thread_id](#)
接続のスレッド ID。

- [ssl_version](#)
接続に使用される SSL のバージョン。
- [ssl_cipher](#)
接続に使用される SSL 暗号。
- [ssl_sessions_reused](#)
接続に再利用された SSL セッションの数。

28.4.3.35 `statement_analysis` および `x$statement_analysis` のビュー

これらのビューには、集計された統計とともに正規化されたステートメントがリストされます。コンテンツは、MySQL Enterprise Monitor クエリー分析ビューに似ています。デフォルトでは、行は合計レイテンシの降順でソートされます。

`statement_analysis` ビューと `x$statement_analysis` ビューには、次のカラムがあります:

- [クエリー](#)
正規化されたステートメントの文字列。
- [db](#)
ステートメントのデフォルトデータベース。存在しない場合は `NULL`。
- [full_scan](#)
ステートメントの発生によって実行された全テーブルスキャンの合計数。
- [exec_count](#)
ステートメントが実行された合計回数。
- [err_count](#)
ステートメントの発生によって生成されたエラーの合計数。
- [warn_count](#)
ステートメントの発生によって生成された警告の合計数。
- [total_latency](#)
ステートメントの時間指定発生の合計待機時間。
- [max_latency](#)
ステートメントの時間指定発生の最大単一待機時間。
- [avg_latency](#)
ステートメントの発生時間ごとの平均待機時間。
- [lock_latency](#)
ステートメントの発生時間でロックを待機する合計時間。
- [rows_sent](#)
ステートメントの発生によって返された行の合計数。
- [rows_sent_avg](#)

ステートメントの出現ごとに返される平均行数。

- [rows_examined](#)

ステートメントの発生によってストレージエンジンから読み取られた行の合計数。

- [rows_examined_avg](#)

ステートメントの発生ごとにストレージエンジンから読み取られた行の平均数。

- [rows_affected](#)

ステートメントの発生によって影響を受ける行の合計数。

- [rows_affected_avg](#)

ステートメントの発生ごとに影響を受ける行の平均数。

- [tmp_tables](#)

ステートメントの発生によって作成された内部インメモリー一時テーブルの合計数。

- [tmp_disk_tables](#)

ステートメントの発生によって作成されたディスク上の内部一時テーブルの合計数。

- [rows_sorted](#)

ステートメントの出現順にソートされた行の合計数。

- [sort_merge_passes](#)

ステートメントの出現によるソートマージパスの合計数。

- [digest](#)

ステートメントダイジェスト。

- [first_seen](#)

ステートメントが最初に表示された時刻。

- [last_seen](#)

ステートメントが最後に表示された時刻。

28.4.3.36 `statements_with_errors_or_warnings` および `x$statements_with_errors_or_warnings` ビュー

これらのビューには、エラーまたは警告が生成された正規化されたステートメントが表示されます。デフォルトでは、行は降順のエラー数および警告数でソートされます。

`statements_with_errors_or_warnings` ビューと `x$statements_with_errors_or_warnings` ビューには、次のカラムがあります:

- [クエリー](#)

正規化されたステートメントの文字列。

- [db](#)

ステートメントのデフォルトデータベース。存在しない場合は `NULL`。

- [exec_count](#)

ステートメントが実行された合計回数。

- [エラー](#)を参照してください。

ステートメントの発生によって生成されたエラーの合計数。

- [error_pct](#)

エラーが発生したステートメントの発生率。

- [warnings](#)

ステートメントの発生によって生成された警告の合計数。

- [warning_pct](#)

警告を生成したステートメントの発生率。

- [first_seen](#)

ステートメントが最初に表示された時刻。

- [last_seen](#)

ステートメントが最後に表示された時刻。

- [digest](#)

ステートメントダイジェスト。

28.4.3.37 statements_with_full_table_scans および x\$statements_with_full_table_scans のビュー

これらのビューには、全テーブルスキャンを実行した正規化されたステートメントが表示されます。デフォルトでは、行は全体スキャンが実行された時間の降順および合計待機時間の降順でソートされます。

[statements_with_full_table_scans](#) ビューと [x\\$statements_with_full_table_scans](#) ビューには、次のカラムがあります:

- [クエリー](#)

正規化されたステートメントの文字列。

- [db](#)

ステートメントのデフォルトデータベース。存在しない場合は `NULL`。

- [exec_count](#)

ステートメントが実行された合計回数。

- [total_latency](#)

ステートメントの時間指定ステートメントイベントの合計待機時間。

- [no_index_used_count](#)

テーブルのスキャンにインデックスが使用されなかった合計回数。

- [no_good_index_used_count](#)

テーブルのスキャンに適切なインデックスが使用されなかった合計回数。

- [no_index_used_pct](#)

テーブルのスキャンにインデックスが使用されなかった時間の割合。

- `rows_sent`
テーブルから返された行の合計数。
- `rows_examined`
テーブルのストレージエンジンから読み取られた行の合計数。
- `rows_sent_avg`
テーブルから返された行の平均数。
- `rows_examined_avg`
テーブルのストレージエンジンから読み取られた行の平均数。
- `first_seen`
ステートメントが最初に表示された時刻。
- `last_seen`
ステートメントが最後に表示された時刻。
- `digest`
ステートメントダイジェスト。

28.4.3.38 `statements_with_runtimes_in_95th_percentile` および `x$statements_with_runtimes_in_95th_percentile` のビュー

これらのビューには、95 パーセンタイルのランタイムを含むステートメントがリストされます。デフォルトでは、行は平均レイテンシの降順でソートされます。

どちらのビューも、`x$ps_digest_avg_latency_distribution` と `x$ps_digest_95th_percentile_by_avg_us` の 2 つのヘルパービューを使用します。

`statements_with_runtimes_in_95th_percentile` ビューと `x$statements_with_runtimes_in_95th_percentile` ビューには、次のカラムがあります:

- `クエリー`
正規化されたステートメントの文字列。
- `db`
ステートメントのデフォルトデータベース。存在しない場合は `NULL`。
- `full_scan`
ステートメントの発生によって実行された全テーブルスキャンの合計数。
- `exec_count`
ステートメントが実行された合計回数。
- `err_count`
ステートメントの発生によって生成されたエラーの合計数。
- `warn_count`
ステートメントの発生によって生成された警告の合計数。
- `total_latency`

ステートメントの時間指定発生の合計待機時間。

- [max_latency](#)

ステートメントの時間指定発生の最大単一待機時間。

- [avg_latency](#)

ステートメントの発生時間ごとの平均待機時間。

- [rows_sent](#)

ステートメントの発生によって返された行の合計数。

- [rows_sent_avg](#)

ステートメントの出現ごとに返される平均行数。

- [rows_examined](#)

ステートメントの発生によってストレージエンジンから読み取られた行の合計数。

- [rows_examined_avg](#)

ステートメントの発生ごとにストレージエンジンから読み取られた行の平均数。

- [first_seen](#)

ステートメントが最初に表示された時刻。

- [last_seen](#)

ステートメントが最後に表示された時刻。

- [digest](#)

ステートメントダイジェスト。

28.4.3.39 statements_with_sorting および x\$statements_with_sorting のビュー

これらのビューには、ソートを実行した正規化されたステートメントがリストされます。デフォルトでは、行は合計レイテンシの降順でソートされます。

[statements_with_sorting](#) ビューと [x\\$statements_with_sorting](#) ビューには、次のカラムがあります:

- [クエリー](#)

正規化されたステートメントの文字列。

- [db](#)

ステートメントのデフォルトデータベース。存在しない場合は `NULL`。

- [exec_count](#)

ステートメントが実行された合計回数。

- [total_latency](#)

ステートメントの時間指定発生の合計待機時間。

- [sort_merge_passes](#)

ステートメントの出現によるソートマージパスの合計数。

- [avg_sort_merges](#)
ステートメントの出現ごとのソートマージパスの平均数。
- [sorts_using_scans](#)
ステートメントの発生によるテーブルスキャンを使用したソートの合計数。
- [sort_using_range](#)
ステートメントの出現による範囲アクセスを使用したソートの合計数。
- [rows_sorted](#)
ステートメントの出現順にソートされた行の合計数。
- [avg_rows_sorted](#)
ステートメントの発生ごとにソートされた行の平均数。
- [first_seen](#)
ステートメントが最初に表示された時刻。
- [last_seen](#)
ステートメントが最後に表示された時刻。
- [digest](#)
ステートメントダイジェスト。

28.4.3.40 statements_with_temp_tables および x\$statements_with_temp_tables のビュー

これらのビューには、一時テーブルを使用した正規化されたステートメントがリストされます。デフォルトでは、使用されるディスク上の一時テーブルの数が降順で、使用されるインメモリー一時テーブルの数が降順でソートされません。

[statements_with_temp_tables](#) ビューと [x\\$statements_with_temp_tables](#) ビューには、次のカラムがあります:

- [クエリー](#)
正規化されたステートメントの文字列。
- [db](#)
ステートメントのデフォルトデータベース。存在しない場合は `NULL`。
- [exec_count](#)
ステートメントが実行された合計回数。
- [total_latency](#)
ステートメントの時間指定発生の合計待機時間。
- [memory_tmp_tables](#)
ステートメントの発生によって作成された内部インメモリー一時テーブルの合計数。
- [disk_tmp_tables](#)
ステートメントの発生によって作成されたディスク上の内部一時テーブルの合計数。
- [avg_tmp_tables_per_query](#)

ステートメントの発生ごとに作成された内部一時テーブルの平均数。

- [tmp_tables_to_disk_pct](#)

ディスク上のテーブルに変換された内部インメモリー一時テーブルの割合。

- [first_seen](#)

ステートメントが最初に表示された時刻。

- [last_seen](#)

ステートメントが最後に表示された時刻。

- [digest](#)

ステートメントダイジェスト。

28.4.3.41 user_summary および x\$user_summary のビュー

これらのビューには、ステートメントアクティビティ、ファイル I/O、および接続がユーザー別にグループ化されて要約されます。デフォルトでは、行は合計レイテンシの降順でソートされます。

`user_summary` ビューと `x$user_summary` ビューには、次のカラムがあります:

- [user](#)

クライアントユーザー名。基礎となる「パフォーマンススキーマ」テーブルの `USER` カラムが `NULL` である行はバックグラウンドスレッド用とみなされ、`background` のホスト名でレポートされます。

- [statements](#)

ユーザーのステートメントの合計数。

- [statement_latency](#)

ユーザーの時間指定ステートメントの合計待機時間。

- [statement_avg_latency](#)

ユーザーの時間指定ステートメント当たりの平均待機時間。

- [table_scans](#)

ユーザーのテーブルスキャンの合計数。

- [file_ios](#)

ユーザーのファイル I/O イベントの合計数。

- [file_io_latency](#)

ユーザーの時間指定ファイル I/O イベントの合計待機時間。

- [current_connections](#)

ユーザーの現在の接続数。

- [total_connections](#)

ユーザーの合計の接続数。

- [unique_hosts](#)

ユーザーの接続元である個別のホストの数。

- [current_memory](#)

ユーザーに割り当てられているメモリーの現在の量。

- [total_memory_allocated](#)

ユーザーに割り当てられたメモリーの合計量。

28.4.3.42 [user_summary_by_file_io](#) および [x\\$user_summary_by_file_io](#) のビュー

これらのビューには、ユーザー別にグループ化されたファイル I/O が要約されます。デフォルトでは、行は合計ファイル I/O レイテンシの降順でソートされます。

[user_summary_by_file_io](#) ビューと [x\\$user_summary_by_file_io](#) ビューには、次のカラムがあります:

- [user](#)

クライアントユーザー名。基礎となる「パフォーマンススキーマ」テーブルの [USER](#) カラムが [NULL](#) である行はバックグラウンドスレッド用とみなされ、[background](#) のホスト名でレポートされます。

- [ios](#)

ユーザーのファイル I/O イベントの合計数。

- [io_latency](#)

ユーザーの時間指定ファイル I/O イベントの合計待機時間。

28.4.3.43 [user_summary_by_file_io_type](#) および [x\\$user_summary_by_file_io_type](#) のビュー

これらのビューには、ユーザーおよびイベントタイプ別にグループ化されたファイル I/O が要約されます。デフォルトでは、行はユーザーおよび合計レイテンシの降順でソートされます。

[user_summary_by_file_io_type](#) ビューと [x\\$user_summary_by_file_io_type](#) ビューには、次のカラムがあります:

- [user](#)

クライアントユーザー名。基礎となる「パフォーマンススキーマ」テーブルの [USER](#) カラムが [NULL](#) である行はバックグラウンドスレッド用とみなされ、[background](#) のホスト名でレポートされます。

- [event_name](#)

ファイル I/O イベント名。

- [total](#)

ユーザーのファイル I/O イベントの発生の合計数。

- [latency](#)

ユーザーのファイル I/O イベントの発生時間の合計待機時間。

- [max_latency](#)

ユーザーのファイル I/O イベントの時間指定発生の最大単一待機時間。

28.4.3.44 [user_summary_by_stages](#) および [x\\$user_summary_by_stages](#) のビュー

これらのビューには、ユーザー別にグループ化されたステージが要約されます。デフォルトでは、行はユーザーでソートされ、ステージ待機時間の合計が降順になります。

[user_summary_by_stages](#) ビューと [x\\$user_summary_by_stages](#) ビューには、次のカラムがあります:

- [user](#)

クライアントユーザー名。基礎となる「パフォーマンススキーマ」テーブルの `USER` カラムが `NULL` である行はバックグラウンドスレッド用とみなされ、`background` のホスト名でレポートされます。

- `event_name`
ステージイベント名。
- `total`
ユーザーのステージイベントの発生の合計数。
- `total_latency`
ユーザーのステージイベントの時間指定発生の合計待機時間。
- `avg_latency`
ユーザーのステージイベントの発生時間ごとの平均待機時間。

28.4.3.45 `user_summary_by_statement_latency` および `x$user_summary_by_statement_latency` のビュー

これらのビューには、ユーザー別にグループ化されたステートメント全体の統計が要約されます。デフォルトでは、行は合計レイテンシの降順でソートされます。

`user_summary_by_statement_latency` ビューと `x$user_summary_by_statement_latency` ビューには、次のカラムがあります:

- `user`
クライアントユーザー名。基礎となる「パフォーマンススキーマ」テーブルの `USER` カラムが `NULL` である行はバックグラウンドスレッド用とみなされ、`background` のホスト名でレポートされます。
- `total`
ユーザーのステートメントの合計数。
- `total_latency`
ユーザーの時間指定ステートメントの合計待機時間。
- `max_latency`
ユーザーの時間指定ステートメントの最大単一待機時間。
- `lock_latency`
ユーザーの時間指定ステートメントによるロック待機の合計時間。
- `rows_sent`
ユーザーに対してステートメントによって返された行の合計数。
- `rows_examined`
ユーザーのステートメントによってストレージエンジンから読み取られた行の合計数。
- `rows_affected`
ユーザーのステートメントの影響を受ける行の合計数。
- `full_scans`
ユーザーのステートメントによる全テーブルスキャンの合計数。

28.4.3.46 user_summary_by_statement_type および x\$user_summary_by_statement_type のビュー

これらのビューには、ユーザーおよびステートメントタイプ別にグループ化された、実行されたステートメントに関する情報が要約されます。デフォルトでは、行はユーザーおよび合計レイテンシの降順でソートされます。

`user_summary_by_statement_type` ビューと `x$user_summary_by_statement_type` ビューには、次のカラムがありません:

- `user`

クライアントユーザー名。基礎となる「パフォーマンススキーマ」テーブルの `USER` カラムが `NULL` である行はバックグラウンドスレッド用とみなされ、`background` のホスト名でレポートされます。

- `statement`

ステートメントイベント名の最終コンポーネント。

- `total`

ユーザーのステートメントイベントの発生の合計数。

- `total_latency`

ユーザーのステートメントイベントの発生時間の合計待機時間。

- `max_latency`

ユーザーのステートメントイベントの時間指定発生の最大単一待機時間。

- `lock_latency`

ユーザーのステートメントイベントの発生時間によるロック待機の合計時間。

- `rows_sent`

ユーザーのステートメントイベントの発生によって返された行の合計数。

- `rows_examined`

ユーザーのステートメントイベントの発生によってストレージエンジンから読み取られた行の合計数。

- `rows_affected`

ユーザーのステートメントイベントの発生の影響を受ける行の合計数。

- `full_scans`

ユーザーのステートメントイベントの発生による全テーブルスキャンの合計数。

28.4.3.47 バージョンビュー

このビューには、現在の `sys` スキーマおよび MySQL サーバーのバージョンが表示されます。

注記

MySQL 8.0.18 では、このビューは非推奨です。将来のバージョンの MySQL で削除される予定です。影響を受けるアプリケーションは、かわりに代替を使用するように調整する必要があります。たとえば、`VERSION()` 関数を使用して、MySQL サーバーのバージョンを取得します。

`version` ビューには、次のカラムがあります:

- `sys_version`

`sys` スキーマのバージョン。

- `mysql_version`

MySQL サーバーのバージョン。

28.4.3.48 `wait_classes_global_by_avg_latency` および `x$wait_classes_global_by_avg_latency` のビュー

これらのビューには、待機クラスの平均待機時間がイベントクラス別にグループ化されて要約されます。デフォルトでは、行は平均レイテンシの降順でソートされます。アイドルイベントは無視されます。

イベントクラスは、最初の 3 つのコンポーネントの後にあるすべてのものをイベント名から削除することによって決定されます。たとえば、`wait/io/file/sql/slow_log` のクラスは `wait/io/file` です。

`wait_classes_global_by_avg_latency` ビューと `x$wait_classes_global_by_avg_latency` ビューには、次のカラムがあります:

- `event_class`

イベントクラス。

- `total`

クラス内で発生したイベントの合計数。

- `total_latency`

クラス内のイベントの発生時間の合計待機時間。

- `min_latency`

クラス内のイベントの発生時間の最小単一待機時間。

- `avg_latency`

クラス内のイベントの発生時間ごとの平均待機時間。

- `max_latency`

クラス内で発生したイベントの最大単一待機時間。

28.4.3.49 `wait_classes_global_by_latency` および `x$wait_classes_global_by_latency` ビュー

これらのビューには、待機クラスの合計待機時間がイベントクラス別にグループ化されて要約されます。デフォルトでは、行は合計レイテンシの降順でソートされます。アイドルイベントは無視されます。

イベントクラスは、最初の 3 つのコンポーネントの後にあるすべてのものをイベント名から削除することによって決定されます。たとえば、`wait/io/file/sql/slow_log` のクラスは `wait/io/file` です。

`wait_classes_global_by_latency` ビューと `x$wait_classes_global_by_latency` ビューには、次のカラムがあります:

- `event_class`

イベントクラス。

- `total`

クラス内で発生したイベントの合計数。

- `total_latency`

クラス内のイベントの発生時間の合計待機時間。

- [min_latency](#)
クラス内のイベントの発生時間の最小単一待機時間。
- [avg_latency](#)
クラス内のイベントの発生時間ごとの平均待機時間。
- [max_latency](#)
クラス内で発生したイベントの最大単一待機時間。

28.4.3.50 [waits_by_host_by_latency](#) および [x\\$waits_by_host_by_latency](#) のビュー

これらのビューには、ホストおよびイベント別にグループ化された待機イベントが要約されます。デフォルトでは、行はホストおよび合計レイテンシの降順でソートされます。アイドルイベントは無視されます。

[waits_by_host_by_latency](#) ビューと [x\\$waits_by_host_by_latency](#) ビューには、次のカラムがあります:

- [host](#)
接続元のホスト。
- [event](#)
イベント名。
- [total](#)
ホストで発生したイベントの合計数。
- [total_latency](#)
ホストのイベントの発生時間の合計待機時間。
- [avg_latency](#)
ホストのイベントの発生時間ごとの平均待機時間。
- [max_latency](#)
ホストのイベントの時間指定発生最大の単一待機時間。

28.4.3.51 [waits_by_user_by_latency](#) および [x\\$waits_by_user_by_latency](#) のビュー

これらのビューには、ユーザーおよびイベント別にグループ化された待機イベントが要約されます。デフォルトでは、行はユーザーおよび合計レイテンシの降順でソートされます。アイドルイベントは無視されます。

[waits_by_user_by_latency](#) ビューと [x\\$waits_by_user_by_latency](#) ビューには、次のカラムがあります:

- [user](#)
接続に関連付けられているユーザー。
- [event](#)
イベント名。
- [total](#)
ユーザーのイベントの発生の合計数。
- [total_latency](#)
ユーザーのイベントの発生時間の合計待機時間。

- [avg_latency](#)
ユーザーのイベントの発生時間ごとの平均待機時間。
- [max_latency](#)
ユーザーのイベントの時間指定発生最大の単一待機時間。

28.4.3.52 `waits_global_by_latency` および `x$waits_global_by_latency` のビュー

これらのビューには、イベント別にグループ化された待機イベントが要約されます。デフォルトでは、行は合計レイテンシの降順でソートされます。アイドルイベントは無視されます。

`waits_global_by_latency` ビューと `x$waits_global_by_latency` ビューには、次のカラムがあります:

- [events](#)
イベント名。
- [total](#)
イベントの発生の合計数。
- [total_latency](#)
イベントの発生時間の合計待機時間。
- [avg_latency](#)
イベントの発生時間ごとの平均待機時間。
- [max_latency](#)
イベントの時間指定発生最大の単一待機時間。

28.4.4 sys スキーマストアドプロシージャ

次の各セクションでは、`sys` スキーマストアドプロシージャについて説明します。

28.4.4.1 `create_synonym_db()` プロシージャ

このプロシージャは、スキーマ名を指定すると、元のスキーマ内のすべてのテーブルおよびビューを参照するビューを含むシノニムスキーマを作成します。たとえば、これを使用して、長い名前 (`INFORMATION_SCHEMA` ではなく `info` など) のスキーマを参照するための短い名前を作成できます。

パラメータ

- `in_db_name` `VARCHAR(64)`: シノニムを作成するスキーマの名前。
- `in_synonym` `VARCHAR(64)`: シノニムスキーマに使用する名前。このスキーマは存在していない必要があります。

例

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
| world |
+-----+
mysql> CALL sys.create_synonym_db('INFORMATION_SCHEMA', 'info');
```

```

+-----+
| summary          |
+-----+
| Created 63 views in the info database |
+-----+
mysql> SHOW DATABASES;
+-----+
| Database        |
+-----+
| information_schema |
| info            |
| mysql           |
| performance_schema |
| sys             |
| world           |
+-----+
mysql> SHOW FULL TABLES FROM info;
+-----+
| Tables_in_info    | Table_type |
+-----+
| character_sets    | VIEW      |
| collation_character_set_applicability | VIEW      |
| collations        | VIEW      |
| column_privileges | VIEW      |
| columns           | VIEW      |
| ...

```

28.4.4.2 diagnostics() プロシージャ

診断目的で現在のサーバステータスのレポートを作成します。

このプロシージャは、`sql_log_bin` システム変数のセッション値を操作して、実行中にバイナリロギングを無効にします。これは制限付き操作であるため、このプロシージャには制限付きセッション変数を設定するのに十分な権限が必要です。 [セクション5.1.9.1「システム変数権限」](#)を参照してください。

`diagnostics()` 用に収集されるデータには、次の情報が含まれます:

- `metrics` ビューからの情報 ([セクション28.4.3.21「メトリックビュー」](#)を参照)
- 95 パーセントイルでクエリーを検出するビューなど、その他の関連する `sys` スキーマビューからの情報
- MySQL サーバーが NDB Cluster の一部である場合、`ndbinfo` スキーマからの情報
- レプリケーションステータス (ソースとレプリカの両方)

`sys` スキーマビューの一部は、初期値 (オプション)、全体値およびデルタ値として計算されます:

- 初期ビューは、`diagnostics()` プロシージャの開始時のビューの内容です。この出力は、デルタ表示に使用される開始値と同じです。初期ビューは、`diagnostics.include_raw` 構成オプションが `ON` の場合に含まれます。
- 全体的なビューは、`diagnostics()` プロシージャの最後にあるビューの内容です。この出力は、デルタ表示に使用される終了値と同じです。ビュー全体が常に含まれます。
- デルタビューは、プロシージャの実行の開始と終了の違いです。最小値と最大値は、それぞれ終了ビューの最小値と最大値です。監視対象期間の最小値および最大値を反映しているとはかぎりません。`metrics` ビューを除き、デルタは最初の出力と最後の出力の間でのみ計算されます。

パラメータ

- `in_max_runtime` `INT UNSIGNED`: データ収集の最大時間 (秒)。 `NULL` を使用して、デフォルトの 60 秒のデータを収集します。それ以外の場合は、0 より大きい値を使用します。
- `in_interval` `INT UNSIGNED`: データ収集間のスリープ時間 (秒)。 `NULL` を使用して、デフォルトの 30 秒間スリープします。それ以外の場合は、0 より大きい値を使用します。
- `in_auto_config` `ENUM('current', 'medium', 'full')`: 使用するパフォーマンススキーマ構成。許可される値は次のとおりです:

- **current**: 現在のインストールメントおよびコンシューマ設定を使用します。
- **medium**: 一部のインストールメントおよびコンシューマを有効にします。
- **full**: すべてのインストールメントおよびコンシューマを有効にします。

注記

インストールメントおよびコンシューマが有効になるほど、MySQL サーバーのパフォーマンスへの影響が大きくなります。 **medium** 設定、特にパフォーマンスに大きな影響を与える **full** 設定には注意してください。

medium または **full** 設定を使用するには、**SUPER** 権限が必要です。

current 以外の設定を選択すると、現在の設定がプロシージャの最後にリストアされます。

構成オプション

`diagnostics()` 操作は、次の構成オプションまたは対応するユーザー定義変数を使用して変更できます (セクション 28.4.2.1 「`sys_config` テーブル」を参照):

- `debug`, `@sys.debug`

このオプションが **ON** の場合は、デバッグ出力を生成します。デフォルトは **OFF** です。

- `diagnostics.allow_i_s_tables`, `@sys.diagnostics.allow_i_s_tables`

このオプションが **ON** の場合、`diagnostics()` プロシージャは `INFORMATION_SCHEMA.TABLES` テーブルに対してテーブルスキャンを実行できます。多くのテーブルがある場合、これはコストがかかる可能性があります。デフォルトは **OFF** です。

- `diagnostics.include_raw`, `@sys.diagnostics.include_raw`

このオプションが **ON** の場合、`diagnostics()` プロシージャの出力には、`metrics` ビューのクエリーからの RAW 出力が含まれます。デフォルトは **OFF** です。

- `statement_truncate_len`, `@sys.statement_truncate_len`

`format_statement()` 関数によって戻されるステートメントの最大長。長いステートメントはこの長さに切り捨てられます。デフォルトは 64 です。

例

30 秒ごとに反復を開始し、現在のパフォーマンススキーマ設定を使用して最大 120 秒間実行する診断レポートを作成します:

```
mysql> CALL sys.diagnostics(120, 30, 'current');
```

`diagnostics()` プロシージャの実行時にファイル内の出力を取得するには、`mysql` クライアントの `tee filename` および `notee` コマンドを使用します (セクション 4.5.1.2 「`mysql` クライアントコマンド」を参照):

```
mysql> tee diag.out;
mysql> CALL sys.diagnostics(120, 30, 'current');
mysql> notee;
```

28.4.4.3 `execute_prepared_stmt()` プロシージャ

SQL ステートメントを文字列として指定すると、準備されたステートメントとして実行されます。プリパードステートメントは実行後に割当て解除されるため、再利用されません。したがって、このプロシージャは主に動的ステートメントを一度に実行する場合に役立ちます。

このプロシージャは、準備されたステートメントの名前として `sys_execute_prepared_stmt` を使用します。プロシージャのコール時にそのステートメントの名前が存在する場合は、以前の内容が破棄されます。

パラメータ

- `in_query LONGTEXT CHARACTER SET utf8`: 実行するステートメントの文字列。

構成オプション

`execute_prepared_stmt()` 操作は、次の構成オプションまたは対応するユーザー定義変数を使用して変更できます ([セクション28.4.2.1「sys_config テーブル」](#) を参照):

- `debug, @sys.debug`

このオプションが **ON** の場合は、デバッグ出力を生成します。デフォルトは **OFF** です。

例

```
mysql> CALL sys.execute_prepared_stmt("SELECT COUNT(*) FROM mysql.user");
+-----+
| COUNT(*) |
+-----+
|      15 |
+-----+
```

28.4.4.4 ps_setup_disable_background_threads() プロシージャ

すべてのバックグラウンドスレッドのパフォーマンススキーマインストゥルメンテーションを無効にします。無効化されたバックグラウンドスレッドの数を示す結果セットを生成します。すでに無効になっているスレッドはカウントされません。

パラメータ

なし

例

```
mysql> CALL sys.ps_setup_disable_background_threads();
+-----+
| summary |
+-----+
| Disabled 24 background threads |
+-----+
```

28.4.4.5 ps_setup_disable_consumer() プロシージャ

引数を含む名前を持つパフォーマンススキーマコンシューマを無効にします。無効化されたコンシューマの数を示す結果セットを生成します。すでに無効になっているコンシューマはカウントされません。

パラメータ

- `consumer VARCHAR(128)`: コンシューマ名の照合に使用される値。LIKE パターン一致のオペランドとして `%consumer%` を使用して識別されます。

"の値は、すべてのコンシューマに一致します。

例

すべてのステートメントコンシューマを無効にします:

```
mysql> CALL sys.ps_setup_disable_consumer('statement');
+-----+
| summary |
+-----+
| Disabled 4 consumers |
+-----+
```

28.4.4.6 ps_setup_disable_instrument() プロシージャ

引数を含む名前を持つパフォーマンススキーマインストゥルメントを無効にします。無効化されたインストゥルメントの数を示す結果セットを生成します。すでに無効になっているインストゥルメントはカウントされません。

パラメータ

- `in_pattern` `VARCHAR(128)`: インストゥルメント名の照合に使用される値。LIKE パターン一致のオペランドとして `%in_pattern%` を使用して識別されます。

"の値は、すべてのインストゥルメントに一致します。

例

特定のインストゥルメントを無効にします。

```
mysql> CALL sys.ps_setup_disable_instrument('wait/lock/metadata/sql/mdl');
+-----+
| summary |
+-----+
| Disabled 1 instrument |
+-----+
```

すべての mutex インストゥルメントを無効にします:

```
mysql> CALL sys.ps_setup_disable_instrument('mutex');
+-----+
| summary |
+-----+
| Disabled 177 instruments |
+-----+
```

28.4.4.7 ps_setup_disable_thread() プロシージャ

接続 ID を指定すると、スレッドのパフォーマンススキーマインストゥルメンテーションが無効になります。無効化されたスレッドの数を示す結果セットを生成します。すでに無効になっているスレッドはカウントされません。

パラメータ

- `in_connection_id` `BIGINT`: 接続 ID。これは、パフォーマンススキーマ `threads` テーブルの `PROCESLIST_ID` カラムまたは `SHOW PROCESLIST` 出力の `id` カラムで指定されたタイプの値です。

例

接続 ID で特定の接続を無効にします:

```
mysql> CALL sys.ps_setup_disable_thread(225);
+-----+
| summary |
+-----+
| Disabled 1 thread |
+-----+
```

現在の接続を無効にします:

```
mysql> CALL sys.ps_setup_disable_thread(CONNECTION_ID());
+-----+
| summary |
+-----+
| Disabled 1 thread |
+-----+
```

28.4.4.8 ps_setup_enable_background_threads() プロシージャ

すべてのバックグラウンドスレッドのパフォーマンススキーマインストゥルメンテーションを有効にします。有効化されたバックグラウンドスレッドの数を示す結果セットを生成します。すでに有効になっているスレッドはカウントされません。

パラメータ

なし

例

```
mysql> CALL sys.ps_setup_enable_background_threads();
+-----+
| summary |
+-----+
| Enabled 24 background threads |
+-----+
```

28.4.4.9 ps_setup_enable_consumer() プロシージャ

引数を含む名前を持つパフォーマンススキーマコンシューマを有効にします。有効化されたコンシューマの数を示す結果セットを生成します。すでに有効になっているコンシューマはカウントされません。

パラメータ

- **consumer VARCHAR(128)**: コンシューマ名の照合に使用される値。LIKE パターン一致のオペランドとして `%consumer%` を使用して識別されます。

"の値は、すべてのコンシューマに一致します。

例

すべてのステートメントコンシューマを有効にします:

```
mysql> CALL sys.ps_setup_enable_consumer('statement');
+-----+
| summary |
+-----+
| Enabled 4 consumers |
+-----+
```

28.4.4.10 ps_setup_enable_instrument() プロシージャ

引数を含む名前を持つパフォーマンススキーマインストゥルメントを有効にします。有効化されたインストゥルメントの数を示す結果セットを生成します。すでに有効になっているインストゥルメントはカウントされません。

パラメータ

- **in_pattern VARCHAR(128)**: インストゥルメント名の照合に使用される値。LIKE パターン一致のオペランドとして `%in_pattern%` を使用して識別されます。

"の値は、すべてのインストゥルメントに一致します。

例

特定のインストゥルメントを有効にします:

```
mysql> CALL sys.ps_setup_enable_instrument('wait/lock/metadata/sql/mdl');
+-----+
| summary |
+-----+
| Enabled 1 instrument |
+-----+
```

すべての mutex インストゥルメントを有効にします:

```
mysql> CALL sys.ps_setup_enable_instrument('mutex');
+-----+
| summary |
+-----+
| Enabled 177 instruments |
+-----+
```

```
+-----+
```

28.4.4.11 ps_setup_enable_thread() プロシージャ

接続 ID を指定すると、スレッドのパフォーマンススキーマインストゥルメンテーションが有効になります。有効化されたスレッドの数を示す結果セットを生成します。すでに有効になっているスレッドはカウントされません。

パラメータ

- `in_connection_id` BIGINT: 接続 ID。これは、パフォーマンススキーマ `threads` テーブルの `PROCESSLIST_ID` カラムまたは `SHOW PROCESSLIST` 出力の `id` カラムで指定されたタイプの値です。

例

接続 ID で特定の接続を有効にします:

```
mysql> CALL sys.ps_setup_enable_thread(225);
+-----+
| summary |
+-----+
| Enabled 1 thread |
+-----+
```

現在の接続を有効にします:

```
mysql> CALL sys.ps_setup_enable_thread(CONNECTION_ID());
+-----+
| summary |
+-----+
| Enabled 1 thread |
+-----+
```

28.4.4.12 ps_setup_reload_saved() プロシージャ

`ps_setup_save()` を使用して、同じセッション内で以前に保存したパフォーマンススキーマ構成をリロードします。詳細は、`ps_setup_save()` の説明を参照してください。

このプロシージャは、`sql_log_bin` システム変数のセッション値を操作して、実行中にバイナリロギングを無効にします。これは制限付き操作であるため、このプロシージャには制限付きセッション変数を設定するのに十分な権限が必要です。セクション5.1.9.1「システム変数権限」を参照してください。

パラメータ

なし

28.4.4.13 ps_setup_reset_to_default() プロシージャ

パフォーマンススキーマ構成をデフォルト設定にリセットします。

パラメータ

- `in_verbose` BOOLEAN: プロシージャの実行中に各設定ステージに関する情報を表示するかどうか。これには、実行された SQL ステートメントが含まれます。

例

```
mysql> CALL sys.ps_setup_reset_to_default(TRUE)G
***** 1. row *****
status: Resetting: setup_actors
DELETE
FROM performance_schema.setup_actors
WHERE NOT (HOST = '%' AND USER = '%' AND ROLE = '%')
***** 1. row *****
status: Resetting: setup_actors
```

```
INSERT IGNORE INTO performance_schema.setup_actors
VALUES ('%', '%', '%')
...
```

28.4.4.14 ps_setup_save() プロシージャ

現在のパフォーマンススキーマ構成を保存します。これにより、デバッグまたはその他の目的で構成を一時的に変更し、`ps_setup_reload_saved()` プロシージャを起動して以前の状態にリストアできます。

構成を保存するための他の同時コールを防止するために、`ps_setup_save()` では、`GET_LOCK()` 関数をコールして `sys.ps_setup_save` という名前のアドバイザロックを取得します。`ps_setup_save()` はタイムアウトパラメータを使用して、ロックがすでに存在する場合に待機する秒数を示します (他のセッションに未処理の構成が保存されていることを示します)。ロックを取得せずにタイムアウトが期限切れになると、`ps_setup_save()` は失敗します。

構成は `TEMPORARY` テーブルに保存されるため、後で same セッション内で `ps_setup_save()` として `ps_setup_reload_saved()` をコールすることを目的としています。`ps_setup_save()` によって一時テーブルが削除され、ロックが解除されます。`ps_setup_save()` を起動せずにセッションを終了すると、テーブルおよびロックが自動的に消えます。

このプロシージャは、`sql_log_bin` システム変数のセッション値を操作して、実行中にバイナリロギングを無効にします。これは制限付き操作であるため、このプロシージャには制限付きセッション変数を設定するのに十分な権限が必要です。[セクション5.1.9.1「システム変数権限」](#)を参照してください。

パラメータ

- `in_timeout` INT: `sys.ps_setup_save` ロックの取得を待機する秒数。負の `timeout` 値は、無限のタイムアウトを表します。

例

```
mysql> CALL sys.ps_setup_save(10);
... make Performance Schema configuration changes ...
mysql> CALL sys.ps_setup_reload_saved();
```

28.4.4.15 ps_setup_show_disabled() プロシージャ

現在無効になっているすべてのパフォーマンススキーマ構成を表示します。

パラメータ

- `in_show_instruments` BOOLEAN: 無効なインストゥルメントを表示するかどうか。これは長いリストである場合があります。
- `in_show_threads` BOOLEAN: 無効なスレッドを表示するかどうか。

例

```
mysql> CALL sys.ps_setup_show_disabled(TRUE, TRUE);
+-----+
| performance_schema_enabled |
+-----+
|          1 |
+-----+

+-----+
| enabled_users |
+-----+
| '%@%' |
+-----+

+-----+-----+-----+
| object_type | objects | enabled | timed |
+-----+-----+-----+
```



```
EVENT | mysql.% | NO | NO |
EVENT | performance_schema.% | NO | NO |
EVENT | information_schema.% | NO | NO |
FUNCTION | mysql.% | NO | NO |
FUNCTION | performance_schema.% | NO | NO |
FUNCTION | information_schema.% | NO | NO |
PROCEDURE | mysql.% | NO | NO |
PROCEDURE | performance_schema.% | NO | NO |
PROCEDURE | information_schema.% | NO | NO |
TABLE | mysql.% | NO | NO |
TABLE | performance_schema.% | NO | NO |
TABLE | information_schema.% | NO | NO |
TRIGGER | mysql.% | NO | NO |
TRIGGER | performance_schema.% | NO | NO |
TRIGGER | information_schema.% | NO | NO |
+-----+-----+-----+-----+
...
```

28.4.4.16 ps_setup_show_disabled_consumers() プロシージャ

現在無効になっているすべてのパフォーマンススキーマコンシューマを表示します。

パラメータ

なし

例

```
mysql> CALL sys.ps_setup_show_disabled_consumers();
+-----+
| disabled_consumers |
+-----+
| events_stages_current |
| events_stages_history |
| events_stages_history_long |
| events_statements_history |
| events_statements_history_long |
| events_transactions_history |
| events_transactions_history_long |
| events_waits_current |
| events_waits_history |
| events_waits_history_long |
+-----+
```

28.4.4.17 ps_setup_show_disabled_instruments() プロシージャ

現在無効になっているすべてのパフォーマンススキーマインストゥルメントを表示します。これは長いリストである場合があります。

パラメータ

なし

例

```
mysql> CALL sys.ps_setup_show_disabled_instruments()G
***** 1. row *****
disabled_instruments: wait/synch/mutex/sql/TC_LOG_MMAP::LOCK_tc
timed: NO
***** 2. row *****
disabled_instruments: wait/synch/mutex/sql/THD::LOCK_query_plan
timed: NO
***** 3. row *****
disabled_instruments: wait/synch/mutex/sql/MYSQL_BIN_LOG::LOCK_commit
timed: NO
...
```

28.4.4.18 ps_setup_show_enabled() プロシージャ

現在有効になっているすべてのパフォーマンススキーマ構成を表示します。

パラメータ

- `in_show_instruments` **BOOLEAN**: 有効なインストゥルメントを表示するかどうか。これは長いリストである場合があります。
- `in_show_threads` **BOOLEAN**: 有効なスレッドを表示するかどうか。

例

```
mysql> CALL sys.ps_setup_show_enabled(FALSE, FALSE);
+-----+
| performance_schema_enabled |
+-----+
|          1 |
+-----+
1 row in set (0.01 sec)

+-----+
| enabled_users |
+-----+
| '%@%' |
+-----+
1 row in set (0.01 sec)

+-----+-----+-----+-----+
| object_type | objects | enabled | timed |
+-----+-----+-----+-----+
| EVENT      | %.%    | YES    | YES   |
| FUNCTION   | %.%    | YES    | YES   |
| PROCEDURE  | %.%    | YES    | YES   |
| TABLE     | %.%    | YES    | YES   |
| TRIGGER    | %.%    | YES    | YES   |
+-----+-----+-----+-----+
5 rows in set (0.02 sec)

+-----+
| enabled_consumers |
+-----+
| events_statements_current |
| events_statements_history |
| events_transactions_current |
| events_transactions_history |
| global_instrumentation |
| statements_digest |
| thread_instrumentation |
+-----+
```

28.4.4.19 ps_setup_show_enabled_consumers() プロシージャ

現在有効になっているすべてのパフォーマンススキーマコンシューマを表示します。

パラメータ

なし

例

```
mysql> CALL sys.ps_setup_show_enabled_consumers();
+-----+
| enabled_consumers |
+-----+
| events_statements_current |
| events_statements_history |
| events_transactions_current |
| events_transactions_history |
| global_instrumentation |
| statements_digest |
| thread_instrumentation |
+-----+
```

28.4.4.20 ps_setup_show_enabled_instruments() プロシージャ

現在有効になっているパフォーマンススキーマインストゥルメントをすべて表示します。これは長いリストである場合があります。

パラメータ

なし

例

```
mysql> CALL sys.ps_setup_show_enabled_instruments()\G
***** 1. row *****
enabled_instruments: wait/io/file/sql/map
timed: YES
***** 2. row *****
enabled_instruments: wait/io/file/sql/binlog
timed: YES
***** 3. row *****
enabled_instruments: wait/io/file/sql/binlog_cache
timed: YES
...
```

28.4.4.21 ps_statement_avg_latency_histogram() プロシージャ

パフォーマンススキーマ [events_statements_summary_by_digest](#) テーブル内で追跡されたすべての正規化ステートメントの平均待機時間値のテキストヒストグラムグラフを表示します。

このプロシージャを使用すると、この MySQL インスタンス内で実行されているステートメントのレイテンシ分散の概要を表示できます。

パラメータ

なし

例

ステートメント単位でのヒストグラム出力。たとえば、ヒストグラム凡例の `* = 2 units` は、各 `*` 文字が 2 つのステートメントを表すことを意味します。

```
mysql> CALL sys.ps_statement_avg_latency_histogram()\G
***** 1. row *****
Performance Schema Statement Digest Average Latency Histogram:

. = 1 unit
* = 2 units
# = 3 units

(0 - 66ms) 88 |#####
(66 - 133ms) 14 |.....
(133 - 199ms) 4 |....
(199 - 265ms) 5 |**
(265 - 332ms) 1 |.
(332 - 398ms) 0 |
(398 - 464ms) 1 |.
(464 - 531ms) 0 |
(531 - 597ms) 0 |
(597 - 663ms) 0 |
(663 - 730ms) 0 |
(730 - 796ms) 0 |
(796 - 863ms) 0 |
(863 - 929ms) 0 |
(929 - 995ms) 0 |
(995 - 1062ms) 0 |

Total Statements: 114; Buckets: 16; Bucket Size: 66 ms;
```

28.4.4.22 ps_trace_statement_digest() プロシージャ

特定のステートメントダイジェストのすべてのパフォーマンススキーマインストゥルメンテーションをトレースします。

パフォーマンススキーマ `events_statements_summary_by_digest` テーブル内で目的のステートメントが見つかった場合は、その `DIGEST` カラムの MD5 値をこのプロシージャに指定し、ポーリング期間と間隔を指定します。結果は、そのダイジェストについてパフォーマンススキーマ内で追跡されたすべての統計のレポートです。

また、この手順では、間隔中にダイジェストの最長実行例に対して `EXPLAIN` の実行も試行されます。パフォーマンススキーマが長い `SQL_TEXT` 値を切り捨てるため、この試行は失敗する可能性があります。その結果、解析エラーのため、`EXPLAIN` は失敗します。

このプロシージャは、`sql_log_bin` システム変数のセッション値を操作して、実行中にバイナリロギングを無効にします。これは制限付き操作であるため、このプロシージャには制限付きセッション変数を設定するのに十分な権限が必要です。セクション 5.1.9.1「システム変数権限」を参照してください。

パラメータ

- `in_digest` VARCHAR(32): 分析するステートメントダイジェスト識別子。
- `in_runtime` INT: 分析を実行する時間 (秒)。
- `in_interval` DECIMAL(2,2): スナップショットを取得しようとする秒単位の分析間隔 (小数も可)。
- `in_start_fresh` BOOLEAN: 起動前にパフォーマンススキーマ `events_statements_history_long` および `events_stages_history_long` テーブルを切り捨てるかどうか。
- `in_auto_enable` BOOLEAN: 必要なコンシューマを自動的に有効にするかどうか。

例

```
mysql> CALL sys.ps_trace_statement_digest('891ec6860f98ba46d89dd20b0c03652c', 10, 0.1, TRUE, TRUE);
+-----+
| SUMMARY STATISTICS |
+-----+
| SUMMARY STATISTICS |
+-----+
1 row in set (9.11 sec)

+-----+-----+-----+-----+-----+
| executions | exec_time | lock_time | rows_sent | rows_examined | tmp_tables | full_scans |
+-----+-----+-----+-----+-----+
| 21 | 4.11 ms | 2.00 ms | 0 | 21 | 0 | 0 |
+-----+-----+-----+-----+-----+
1 row in set (9.11 sec)

+-----+-----+-----+
| event_name | count | latency |
+-----+-----+-----+
| stage/sql/statistics | 16 | 546.92 us |
| stage/sql/freeing items | 18 | 520.11 us |
| stage/sql/init | 51 | 466.80 us |
...
| stage/sql/cleaning up | 18 | 11.92 us |
| stage/sql/executing | 16 | 6.95 us |
+-----+-----+-----+
17 rows in set (9.12 sec)

+-----+
| LONGEST RUNNING STATEMENT |
+-----+
| LONGEST RUNNING STATEMENT |
+-----+
1 row in set (9.16 sec)

+-----+-----+-----+-----+-----+
| thread_id | exec_time | lock_time | rows_sent | rows_examined | tmp_tables | full_scan |
```

```

+-----+-----+-----+-----+-----+-----+
| 166646 | 618.43 us | 1.00 ms | 0 | 1 | 0 | 0 |
+-----+-----+-----+-----+-----+
1 row in set (9.16 sec)

# Truncated for clarity...
+-----+-----+
| sql_text |
+-----+-----+
| select hibeventhe0_id as id1382_, hibeventhe0_createdTime ... |
+-----+-----+
1 row in set (9.17 sec)

+-----+-----+
| event_name | latency |
+-----+-----+
| stage/sql/init | 8.61 us |
| stage/sql/init | 331.07 ns |
...
| stage/sql/freeing items | 30.46 us |
| stage/sql/cleaning up | 662.13 ns |
+-----+-----+
18 rows in set (9.23 sec)

+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | hibeventhe0_ | const | fixedTime | fixedTime | 775 | const,const | 1 | NULL |
+-----+-----+-----+-----+-----+-----+
1 row in set (9.27 sec)

Query OK, 0 rows affected (9.28 sec)

```

28.4.4.23 ps_trace_thread() プロシージャ

インストゥルメントされたスレッドのすべてのパフォーマンススキーマデータを `.dot` 形式のグラフファイルにダンプします (DOT グラフ記述言語の場合)。プロシージャから返される各結果セットは、完全なグラフに使用する必要があります。

このプロシージャは、`sql_log_bin` システム変数のセッション値を操作して、実行中にバイナリロギングを無効にします。これは制限付き操作であるため、このプロシージャには制限付きセッション変数を設定するのに十分な権限が必要です。 [セクション5.1.9.1「システム変数権限」](#) を参照してください。

パラメータ

- `in_thread_id` INT: トレースするスレッド。
- `in_outfile` VARCHAR(255): `.dot` 出力ファイルに使用する名前。
- `in_max_runtime` DECIMAL(20,2): データを収集する最大秒数 (小数も可)。 `NULL` を使用して、デフォルトの 60 秒のデータを収集します。
- `in_interval` DECIMAL(20,2): データ収集間でスリープする秒数 (小数も可)。 `NULL` を使用して、デフォルトの 1 秒間スリープします。
- `in_start_fresh` BOOLEAN: トレースの前にすべてのパフォーマンススキーマデータをリセットするかどうか。
- `in_auto_setup` BOOLEAN: 他のすべてのスレッドを無効にし、すべてのインストゥルメントおよびコンシューマを有効にするかどうか。これにより、実行終了時の設定もリセットされます。
- `in_debug` BOOLEAN : グラフに `file:lineno` 情報を含めるかどうか。

例

```

mysql> CALL sys.ps_trace_thread(25, CONCAT('/tmp/stack-', REPLACE(NOW(), '-', '-')), '.dot'), NULL, NULL, TRUE, TRUE, TRUE);
+-----+
| summary |
+-----+
| Disabled 1 thread |

```

```
+-----+
1 row in set (0.00 sec)

+-----+
| Info |
+-----+
| Data collection starting for THREAD_ID = 25 |
+-----+
1 row in set (0.03 sec)

+-----+
| Info |
+-----+
| Stack trace written to /tmp/stack-2014-02-16-21:18:41.dot |
+-----+
1 row in set (60.07 sec)

+-----+
| Convert to PDF |
+-----+
| dot -Tpdf -o /tmp/stack_25.pdf /tmp/stack-2014-02-16-21:18:41.dot |
+-----+
1 row in set (60.07 sec)

+-----+
| Convert to PNG |
+-----+
| dot -Tpng -o /tmp/stack_25.png /tmp/stack-2014-02-16-21:18:41.dot |
+-----+
1 row in set (60.07 sec)

+-----+
| summary |
+-----+
| Enabled 1 thread |
+-----+
1 row in set (60.32 sec)
```

28.4.4.24 ps_truncate_all_tables() プロシージャ

すべてのパフォーマンススキーマサマリーテーブルを切り捨てて、すべての集約インストゥルメンテーションをスナップショットとしてリセットします。切り捨てられたテーブルの数を示す結果セットを生成します。

パラメータ

- `in_verbose` **BOOLEAN**: 実行前に各 **TRUNCATE TABLE** ステートメントを表示するかどうか。

例

```
mysql> CALL sys.ps_truncate_all_tables(FALSE);
+-----+
| summary |
+-----+
| Truncated 49 tables |
+-----+
```

28.4.4.25 statement_performance_analyzer() プロシージャ

サーバーで実行されているステートメントのレポートを作成します。ビューは、全体またはデルタ (あるいはその両方) のアクティビティに基づいて計算されます。

このプロシージャは、`sql_log_bin` システム変数のセッション値を操作して、実行中にバイナリロギングを無効にします。これは制限付き操作であるため、このプロシージャには制限付きセッション変数を設定するのに十分な権限が必要です。 [セクション5.1.9.1「システム変数権限」](#)を参照してください。

パラメータ

- `in_action` **ENUM('snapshot', 'overall', 'delta', 'create_tmp', 'create_table', 'save', 'cleanup')**: 実行するアクション。次の値を使用できます:

- **snapshot**: スナップショットを格納します。デフォルトでは、パフォーマンススキーマ `events_statements_summary_by_digest` テーブルの現在の内容のスナップショットが作成されます。 `in_table` を設定すると、これを上書きして、指定したテーブルのコンテンツをコピーできます。スナップショットは、`sys` スキーマの `tmp_digests` 一時テーブルに格納されます。
- **overall**: `in_table` で指定されたテーブルのコンテンツに基づいて分析を生成します。全体的な分析では、`in_table` を `NOW()` にして新しいスナップショットを使用できます。これにより、既存のスナップショットが上書きされます。既存のスナップショットを使用するには、`NULL for in_table` を使用します。 `in_table` が `NULL` で、スナップショットが存在しない場合は、新しいスナップショットが作成されます。 `in_views` パラメータおよび `statement_performance_analyzer.limit` 構成オプションは、このプロシージャの操作に影響します。
- **delta**: デルタ分析を生成します。デルタは、`in_table` で指定された参照テーブルとスナップショット (存在する必要があります) の間で計算されます。このアクションでは、`sys` スキーマの `tmp_digests_delta` 一時テーブルが使用されます。 `in_views` パラメータおよび `statement_performance_analyzer.limit` 構成オプションは、このプロシージャの操作に影響します。
- **create_table**: 後で使用するためのスナップショットの格納に適した通常のテーブルを作成します (デルタの計算など)。
- **create_tmp**: 後で使用するためのスナップショットの格納に適した一時テーブルを作成します (デルタの計算など)。
- **save**: `in_table` で指定されたテーブルにスナップショットを保存します。テーブルが存在し、正しい構造である必要があります。スナップショットが存在しない場合は、新しいスナップショットが作成されます。
- **cleanup**: スナップショットおよびデルタに使用されている一時テーブルを削除します。
- **in_table VARCHAR(129)**: `in_action` パラメータで指定される一部のアクションに使用されるテーブルパラメータ。バックティック (``) 識別子引用符文字を使用せずに、`db_name.tbl_name` または `tbl_name` の形式を使用します。ピリオド (.) は、データベース名およびテーブル名ではサポートされていません。
各 `in_action` 値の `in_table` 値の意味の詳細は、個々の `in_action` 値の説明を参照してください。
- **in_views SET ('with_runtimes_in_95th_percentile', 'analysis', 'with_errors_or_warnings', 'with_full_table_scans', 'with_sorting', 'with_temp_tables', 'custom')**: 含めるビュー。このパラメータは `SET` 値であるため、複数のビュー名をカンマで区切って含めることができます。デフォルトでは、`custom` を除くすべてのビューが含まれます。次の値を使用できます:
 - `with_runtimes_in_95th_percentile`: `statements_with_runtimes_in_95th_percentile` ビューを使用します。
 - `analysis`: `statement_analysis` ビューを使用します。
 - `with_errors_or_warnings`: `statements_with_errors_or_warnings` ビューを使用します。
 - `with_full_table_scans`: `statements_with_full_table_scans` ビューを使用します。
 - `with_sorting`: `statements_with_sorting` ビューを使用します。
 - `with_temp_tables`: `statements_with_temp_tables` ビューを使用します。
 - `custom`: カスタムビューを使用します。クエリーまたは既存のビューに名前を付けるには、`statement_performance_analyzer.view` 構成オプションを使用してこのビューを指定する必要があります。

構成オプション

`statement_performance_analyzer()` 操作は、次の構成オプションまたは対応するユーザー定義変数を使用して変更できます (セクション28.4.2.1「`sys_config` テーブル」を参照):

- `debug, @sys.debug`
このオプションが `ON` の場合は、デバッグ出力を生成します。デフォルトは `OFF` です。
- `statement_performance_analyzer.limit, @sys.statement_performance_analyzer.limit`

組み込み制限のないビューに対して返す行の最大数。デフォルトは 100 です。

- `statement_performance_analyzer.view`, `@sys.statement_performance_analyzer.view`

使用するカスタムクエリーまたはビュー。オプション値に空白が含まれている場合は、クエリーとして解釈されます。それ以外の場合は、パフォーマンススキーマ `events_statements_summary_by_digest` テーブルをクエリーする既存のビューの名前である必要があります。 `statement_performance_analyzer.limit` 構成オプションが 0 より大きい場合、クエリーまたはビュー定義に `LIMIT` 句を含めることはできません。ビューを指定する場合は、`in_table` パラメータと同じ形式を使用します。デフォルトは `NULL` です (カスタムビューが定義されていません)。

例

`events_statements_summary_by_digest` の最後の切捨て以降の 95 パーセンタイルのクエリーと 1 分間のデルタ期間を使用してレポートを作成するには:

1. 初期スナップショットを格納する一時テーブルを作成します。
2. 初期スナップショットを作成します。
3. 初期スナップショットを一時テーブルに保存します。
4. 1 分間待ちます。
5. 新しいスナップショットを作成します。
6. 新しいスナップショットに基づいて分析を実行します。
7. 初期スナップショットと新規スナップショットの差分に基づいて分析を実行します。

```
mysql> CALL sys.statement_performance_analyzer('create_tmp', 'mydb.tmp_digests_ini', NULL);
Query OK, 0 rows affected (0.08 sec)

mysql> CALL sys.statement_performance_analyzer('snapshot', NULL, NULL);
Query OK, 0 rows affected (0.02 sec)

mysql> CALL sys.statement_performance_analyzer('save', 'mydb.tmp_digests_ini', NULL);
Query OK, 0 rows affected (0.00 sec)

mysql> DO SLEEP(60);
Query OK, 0 rows affected (1 min 0.00 sec)

mysql> CALL sys.statement_performance_analyzer('snapshot', NULL, NULL);
Query OK, 0 rows affected (0.02 sec)

mysql> CALL sys.statement_performance_analyzer('overall', NULL, 'with_runtimes_in_95th_percentile');
+-----+
| Next Output          |
+-----+
| Queries with Runtime in 95th Percentile |
+-----+
1 row in set (0.05 sec)

...

mysql> CALL sys.statement_performance_analyzer('delta', 'mydb.tmp_digests_ini', 'with_runtimes_in_95th_percentile');
+-----+
| Next Output          |
+-----+
| Queries with Runtime in 95th Percentile |
+-----+
1 row in set (0.03 sec)

...
```

95 パーセンタイルクエリーおよび全テーブルスキャンを使用した上位 10 クエリーの全体的なレポートを作成します:

```
mysql> CALL sys.statement_performance_analyzer('snapshot', NULL, NULL);
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> SET @sys.statement_performance_analyzer.limit = 10;
Query OK, 0 rows affected (0.00 sec)

mysql> CALL sys.statement_performance_analyzer('overall', NULL, 'with_runtimes_in_95th_percentile,with_full_table_scans');
+-----+
| Next Output |
+-----+
| Queries with Runtime in 95th Percentile |
+-----+
1 row in set (0.01 sec)

...

+-----+
| Next Output |
+-----+
| Top 10 Queries with Full Table Scan |
+-----+
1 row in set (0.09 sec)

...
```

合計実行時間でソートされた上位 10 のクエリーを示すカスタムビューを使用し、Linux の `watch` コマンドを使用してビューを毎分リフレッシュします:

```
mysql> CREATE OR REPLACE VIEW mydb.my_statements AS
SELECT sys.format_statement(DIGEST_TEXT) AS query,
       SCHEMA_NAME AS db,
       COUNT_STAR AS exec_count,
       sys.format_time(SUM_TIMER_WAIT) AS total_latency,
       sys.format_time(AVG_TIMER_WAIT) AS avg_latency,
       ROUND(IFNULL(SUM_ROWS_SENT / NULLIF(COUNT_STAR, 0), 0)) AS rows_sent_avg,
       ROUND(IFNULL(SUM_ROWS_EXAMINED / NULLIF(COUNT_STAR, 0), 0)) AS rows_examined_avg,
       ROUND(IFNULL(SUM_ROWS_AFFECTED / NULLIF(COUNT_STAR, 0), 0)) AS rows_affected_avg,
       DIGEST AS digest
FROM performance_schema.events_statements_summary_by_digest
ORDER BY SUM_TIMER_WAIT DESC;
Query OK, 0 rows affected (0.10 sec)

mysql> CALL sys.statement_performance_analyzer('create_table', 'mydb.digests_prev', NULL);
Query OK, 0 rows affected (0.10 sec)

shell> watch -n 60 "mysql sys --table -e \"
> SET @sys.statement_performance_analyzer.view = 'mydb.my_statements';
> SET @sys.statement_performance_analyzer.limit = 10;
> CALL statement_performance_analyzer('snapshot', NULL, NULL);
> CALL statement_performance_analyzer('delta', 'mydb.digests_prev', 'custom');
> CALL statement_performance_analyzer('save', 'mydb.digests_prev', NULL);
> \"";

Every 60.0s: mysql sys --table -e " ... Mon Dec 22 10:58:51 2014

+-----+
| Next Output |
+-----+
| Top 10 Queries Using Custom View |
+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| query      | db | exec_count | total_latency | avg_latency | rows_sent_avg | rows_examined_avg | rows_affected_avg | digest |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
...
```

28.4.4.26 table_exists() プロシージャ

特定のテーブルが通常のテーブル、**TEMPORARY** テーブルまたはビューとして存在するかどうかをテストします。このプロシージャは、**OUT** パラメータでテーブルタイプを戻します。指定された名前の一時的テーブルと永続テーブルの両方が存在する場合は、**TEMPORARY** が返されます。

パラメータ

- `in_db VARCHAR(64)`: テーブルの存在をチェックするデータベースの名前。

- `in_table VARCHAR(64)`: 存在をチェックするテーブルの名前。
- `out_exists ENUM('','BASE TABLE','VIEW','TEMPORARY')`: 戻り値。これは `OUT` パラメータであるため、テーブル型を格納できる変数である必要があります。プロシージャが戻ると、変数にはテーブルが存在するかどうかを示す次のいずれかの値が設定されます:
 - `''`: テーブル名が実テーブル、`TEMPORARY` テーブルまたはビューとして存在しません。
 - `BASE TABLE`: テーブル名は実 (永続) テーブルとして存在します。
 - `VIEW`: テーブル名はビューとして存在します。
 - `TEMPORARY`: テーブル名は `TEMPORARY` テーブルとして存在します。

例

```
mysql> CREATE DATABASE db1;
Query OK, 1 row affected (0.01 sec)

mysql> USE db1;
Database changed
mysql> CREATE TABLE t1 (id INT PRIMARY KEY);
Query OK, 0 rows affected (0.03 sec)

mysql> CREATE TABLE t2 (id INT PRIMARY KEY);
Query OK, 0 rows affected (0.20 sec)

mysql> CREATE view v_t1 AS SELECT * FROM t1;
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TEMPORARY TABLE t1 (id INT PRIMARY KEY);
Query OK, 0 rows affected (0.00 sec)

mysql> CALL sys.table_exists('db1', 't1', @exists); SELECT @exists;
Query OK, 0 rows affected (0.01 sec)

+-----+
| @exists |
+-----+
| TEMPORARY |
+-----+
1 row in set (0.00 sec)

mysql> CALL sys.table_exists('db1', 't2', @exists); SELECT @exists;
Query OK, 0 rows affected (0.02 sec)

+-----+
| @exists |
+-----+
| BASE TABLE |
+-----+
1 row in set (0.00 sec)

mysql> CALL sys.table_exists('db1', 'v_t1', @exists); SELECT @exists;
Query OK, 0 rows affected (0.02 sec)

+-----+
| @exists |
+-----+
| VIEW |
+-----+
1 row in set (0.00 sec)

mysql> CALL sys.table_exists('db1', 't3', @exists); SELECT @exists;
Query OK, 0 rows affected (0.00 sec)

+-----+
| @exists |
+-----+
| |
+-----+
1 row in set (0.00 sec)
```

28.4.5 sys スキーマストアドファンクション

次の各セクションでは、`sys` スキーマストアドファンクションについて説明します。

28.4.5.1 `extract_schema_from_file_name()` 関数

ファイルパス名を指定すると、スキーマ名を表すパスコンポーネントを返します。この関数は、ファイル名がスキーマディレクトリ内にあることを前提としています。このため、独自の `DATA_DIRECTORY` テーブルオプションを使用して定義されたパーティションまたはテーブルでは機能しません。

この関数は、ファイルパス名を含むパフォーマンススキーマからファイル I/O 情報を抽出する場合に役立ちます。完全パス名よりも理解しやすく、オブジェクトスキーマ名に対する結合で使用できるスキーマ名を表示する便利な方法を提供します。

パラメータ

- `path VARCHAR(512)`: スキーマ名の抽出元のデータファイルへのフルパス。

戻り値

`VARCHAR(64)` 値。

例

```
mysql> SELECT sys.extract_schema_from_file_name('/usr/local/mysql/data/world/City.ibd');
+-----+
| sys.extract_schema_from_file_name('/usr/local/mysql/data/world/City.ibd') |
+-----+
| world                               |
+-----+
```

28.4.5.2 `extract_table_from_file_name()` 関数

ファイルパス名を指定すると、テーブル名を表すパスコンポーネントを返します。

この関数は、ファイルパス名を含むパフォーマンススキーマからファイル I/O 情報を抽出する場合に役立ちます。完全パス名よりも理解しやすく、オブジェクトテーブル名に対する結合で使用できるテーブル名を表示する便利な方法を提供します。

パラメータ

- `path VARCHAR(512)`: テーブル名の抽出元のデータファイルへのフルパス。

戻り値

`VARCHAR(64)` 値。

例

```
mysql> SELECT sys.extract_table_from_file_name('/usr/local/mysql/data/world/City.ibd');
+-----+
| sys.extract_table_from_file_name('/usr/local/mysql/data/world/City.ibd') |
+-----+
| City                               |
+-----+
```

28.4.5.3 `format_bytes()` 関数

注記

MySQL 8.0.16 では、`format_bytes()` は非推奨です。将来のバージョンの MySQL で削除される予定です。かわりに、組込みの `FORMAT_BYTES()` 関数を使用するようにアプリケーションを移行する必要があります。セクション12.22「パフォーマンススキーマ関数」を参照してください

バイト数を指定すると、人間が読める形式に変換され、値と単位インジケータで構成される文字列が返されます。値のサイズに応じて、単位部分は **bytes**、**KiB** (キビバイト)、**MiB** (メビバイト)、**GiB** (ギビバイト)、**TiB** (テビバイト) または **PiB** (ペビバイト) になります。

パラメータ

- **bytes TEXT**: フォーマットするバイト数。

戻り値

TEXT 値。

例

```
mysql> SELECT sys.format_bytes(512), sys.format_bytes(18446644073709551615);
+-----+-----+
| sys.format_bytes(512) | sys.format_bytes(18446644073709551615) |
+-----+-----+
| 512 bytes           | 16383.91 PiB                          |
+-----+-----+
```

28.4.5.4 format_path() 関数

パス名を指定すると、次のシステム変数の値と一致するサブパスを次の順序で置換した後に、変更されたパス名を返します:

```
datadir
tmpdir
slave_load_tmpdir
innodb_data_home_dir
innodb_log_group_home_dir
innodb_undo_directory
basedir
```

システム変数 **sysvar** の値と一致する値は、文字列 **@@GLOBAL.sysvar** に置き換えられます。

パラメータ

- **path VARCHAR(512)**: 書式設定するパス名。

戻り値

VARCHAR(512) CHARACTER SET utf8 値。

例

```
mysql> SELECT sys.format_path('/usr/local/mysql/data/world/City.ibd');
+-----+-----+
| sys.format_path('/usr/local/mysql/data/world/City.ibd') |
+-----+-----+
| /usr/local/mysql/data/world/City.ibd                    |
+-----+-----+
```

28.4.5.5 format_statement() 関数

文字列 (通常は SQL ステートメントを表します) を指定すると、**statement_truncate_len** 構成オプションで指定された長さまで短くなり、結果が返されます。文字列が **statement_truncate_len** より短い場合、切捨ては行われません。それ以外の場合、文字列の中央部分は省略記号 (...) に置き換えられます。

この関数は、「パフォーマンススキーマ」テーブルから取得した長いステートメントを既知の固定最大長に書式設定する場合に便利です。

パラメータ

- **statement LONGTEXT**: 書式設定するステートメント。

構成オプション

`format_statement()` 操作は、次の構成オプションまたは対応するユーザー定義変数を使用して変更できます ([セクション 28.4.2.1 「sys_config テーブル」](#) を参照):

- `statement_truncate_len`, `@sys.statement_truncate_len`

`format_statement()` 関数によって戻されるステートメントの最大長。長いステートメントはこの長さに切り捨てられます。デフォルトは 64 です。

戻り値

LONGTEXT 値。

例

デフォルトでは、`format_statement()` は 64 文字以下のステートメントを切り捨てます。
`@sys.statement_truncate_len` を設定すると、現在のセッションの切捨て長が変更されます:

```
mysql> SET @stmt = 'SELECT variable, value, set_time, set_by FROM sys_config';
mysql> SELECT sys.format_statement(@stmt);
+-----+
| sys.format_statement(@stmt) |
+-----+
| SELECT variable, value, set_time, set_by FROM sys_config |
+-----+
mysql> SET @sys.statement_truncate_len = 32;
mysql> SELECT sys.format_statement(@stmt);
+-----+
| sys.format_statement(@stmt) |
+-----+
| SELECT variabl ... ROM sys_config |
+-----+
```

28.4.5.6 format_time() 関数

注記

MySQL 8.0.16 では、`format_time()` は非推奨です。将来のバージョンの MySQL で削除される予定です。かわりに、組み込みの `FORMAT_PICO_TIME()` 関数を使用するようにアプリケーションを移行する必要があります。 [セクション 12.22 「パフォーマンススキーマ関数」](#) を参照してください

ピコ秒単位のパフォーマンススキーマレイテンシまたは待機時間を指定すると、それを人間が読める形式に変換し、値と単位インジケータで構成される文字列を返します。値のサイズに応じて、単位部分は `ps` (ピコ秒)、`ns` (ナノ秒)、`us` (マイクロ秒)、`ms` (ms)、`s` (秒)、`m` (分)、`h` (時間)、`d` (日) または `w` (週) です。

パラメータ

- `picoseconds` TEXT: 書式設定するピコ秒値。

戻り値

TEXT 値。

例

```
mysql> SELECT sys.format_time(3501), sys.format_time(188732396662000);
+-----+-----+
| sys.format_time(3501) | sys.format_time(188732396662000) |
+-----+-----+
| 3.50 ns              | 3.15 m                            |
+-----+-----+
```

28.4.5.7 list_add() 関数

カンマ区切りの値リストに値を追加し、結果を返します。

この関数および `list_drop()` は、カンマ区切りの値リストを取る `sql_mode` や `optimizer_switch` などのシステム変数の値を操作する場合に役立ちます。

パラメータ

- `in_list` TEXT: 変更するリスト。
- `in_add_value` TEXT: リストに追加する値。

戻り値

TEXT 値。

例

```
mysql> SELECT @@sql_mode;
+-----+
| @@sql_mode |
+-----+
| ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES |
+-----+
mysql> SET @@sql_mode = sys.list_add(@@sql_mode, 'NO_ENGINE_SUBSTITUTION');
mysql> SELECT @@sql_mode;
+-----+
| @@sql_mode |
+-----+
| ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION |
+-----+
mysql> SET @@sql_mode = sys.list_drop(@@sql_mode, 'ONLY_FULL_GROUP_BY');
mysql> SELECT @@sql_mode;
+-----+
| @@sql_mode |
+-----+
| STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION |
+-----+
```

28.4.5.8 list_drop() 関数

カンマ区切りの値リストから値を削除し、結果を返します。詳細は、`list_add()` の説明を参照してください

パラメータ

- `in_list` TEXT: 変更するリスト。
- `in_drop_value` TEXT: リストから削除する値。

戻り値

TEXT 値。

28.4.5.9 ps_is_account_enabled() 関数

指定されたアカウントのパフォーマンススキーマインストールメンテーションが有効かどうかを示す `YES` または `NO` を返します。

パラメータ

- `in_host` VARCHAR(60): チェックするアカウントのホスト名。
- `in_user` VARCHAR(32): チェックするアカウントのユーザー名。

戻り値

ENUM('YES','NO') 値。

例

```
mysql> SELECT sys.ps_is_account_enabled('localhost', 'root');
+-----+
| sys.ps_is_account_enabled('localhost', 'root') |
+-----+
| YES |
+-----+
```

28.4.5.10 ps_is_consumer_enabled() 関数

指定されたパフォーマンススキーマコンシューマが有効かどうかを示す **YES** または **NO** を返します。引数が **NULL** の場合は **NULL** を返します。引数が有効なコンシューマ名でない場合は、エラーが発生します。(MySQL 8.0.18 より前では、引数が有効なコンシューマ名でない場合、この関数は **NULL** を返します。)

この関数はコンシューマ階層を考慮しているため、依存するすべてのコンシューマも有効になっていないかぎり、コンシューマは有効とみなされません。コンシューマ階層の詳細は、[セクション27.4.7「コンシューマによる事前ファイルタリング」](#)を参照してください。

パラメータ

- **in_consumer** **VARCHAR(64)**: チェックするコンシューマの名前。

戻り値

ENUM('YES','NO') 値。

例

```
mysql> SELECT sys.ps_is_consumer_enabled('thread_instrumentation');
+-----+
| sys.ps_is_consumer_enabled('thread_instrumentation') |
+-----+
| YES |
+-----+
```

28.4.5.11 ps_is_instrument_default_enabled() 関数

指定されたパフォーマンススキーマインストゥルメントがデフォルトで有効になっているかどうかを示す **YES** または **NO** を返します。

パラメータ

- **in_instrument** **VARCHAR(128)**: チェックするインストゥルメントの名前。

戻り値

ENUM('YES','NO') 値。

例

```
mysql> SELECT sys.ps_is_instrument_default_enabled('memory/innodb/row_log_buf');
+-----+
| sys.ps_is_instrument_default_enabled('memory/innodb/row_log_buf') |
+-----+
| NO |
+-----+
mysql> SELECT sys.ps_is_instrument_default_enabled('statement/sql/alter_user');
+-----+
| sys.ps_is_instrument_default_enabled('statement/sql/alter_user') |
+-----+
| YES |
+-----+
```

28.4.5.12 ps_is_instrument_default_timed() 関数

指定されたパフォーマンススキーマインストゥルメントがデフォルトで時間指定されているかどうかを示す **YES** または **NO** を返します。

パラメータ

- `in_instrument` **VARCHAR(128)**: チェックするインストゥルメントの名前。

戻り値

ENUM('YES','NO') 値。

例

```
mysql> SELECT sys.ps_is_instrument_default_timed('memory/innodb/row_log_buf');
+-----+
| sys.ps_is_instrument_default_timed('memory/innodb/row_log_buf') |
+-----+
| NO |
+-----+
mysql> SELECT sys.ps_is_instrument_default_timed('statement/sql/alter_user');
+-----+
| sys.ps_is_instrument_default_timed('statement/sql/alter_user') |
+-----+
| YES |
+-----+
```

28.4.5.13 ps_is_thread_instrumented() 関数

指定された接続 ID のパフォーマンススキーマインストゥルメンテーションが有効かどうかを示す **YES** または **NO** を返します。ID が不明な場合は **UNKNOWN**、ID が **NULL** の場合は **NULL** を返します。

パラメータ

- `in_connection_id` **BIGINT UNSIGNED**: 接続 ID。これは、パフォーマンススキーマ `threads` テーブルの `PROCESSLIST_ID` カラムまたは `SHOW PROCESSLIST` 出力の `Id` カラムで指定されたタイプの値です。

戻り値

ENUM('YES','NO','UNKNOWN') 値。

例

```
mysql> SELECT sys.ps_is_thread_instrumented(43);
+-----+
| sys.ps_is_thread_instrumented(43) |
+-----+
| UNKNOWN |
+-----+
mysql> SELECT sys.ps_is_thread_instrumented(CONNECTION_ID());
+-----+
| sys.ps_is_thread_instrumented(CONNECTION_ID()) |
+-----+
| YES |
+-----+
```

28.4.5.14 ps_thread_account() 関数

パフォーマンススキーマスレッド ID を指定すると、スレッドに関連付けられた `user_name@host_name` アカウントを返します。

パラメータ

- `in_thread_id` **BIGINT UNSIGNED**: アカウントを返すスレッド ID。この値は、一部のパフォーマンススキーマ `threads` テーブル行の `THREAD_ID` カラムと一致する必要があります。

戻り値

TEXT 値。

例

```
mysql> SELECT sys.ps_thread_account(sys.ps_thread_id(CONNECTION_ID()));
+-----+
| sys.ps_thread_account(sys.ps_thread_id(CONNECTION_ID())) |
+-----+
| root@localhost |
+-----+
```

28.4.5.15 ps_thread_id() 関数

注記

MySQL 8.0.16 では、`ps_thread_id()` は非推奨です。将来のバージョンの MySQL で削除される予定です。かわりに、組み込みの `PS_THREAD_ID()` および `PS_CURRENT_THREAD_ID()` 関数を使用するようにアプリケーションを移行する必要があります。セクション12.22「パフォーマンススキーマ関数」を参照してください

指定された接続 ID に割り当てられたパフォーマンススキーマスレッド ID、または接続 ID が NULL の場合は現在の接続のスレッド ID を返します。

パラメータ

- `in_connection_id` BIGINT UNSIGNED: スレッド ID を返す接続の ID。これは、パフォーマンススキーマ `threads` テーブルの `PROCESSLIST_ID` カラムまたは `SHOW PROCESSLIST` 出力の `id` カラムで指定されたタイプの値です。

戻り値

BIGINT UNSIGNED 値。

例

```
mysql> SELECT sys.ps_thread_id(260);
+-----+
| sys.ps_thread_id(260) |
+-----+
| 285 |
+-----+
```

28.4.5.16 ps_thread_stack() 関数

指定されたスレッド ID のパフォーマンススキーマ内のすべてのステートメント、ステージおよびイベントの JSON 形式のスタックを返します。

パラメータ

- `in_thread_id` BIGINT: トレースするスレッドの ID。この値は、一部のパフォーマンススキーマ `threads` テーブル行の `THREAD_ID` カラムと一致する必要があります。
- `in_verbose` BOOLEAN: イベントに `file:lineno` 情報を含めるかどうか。

戻り値

LONGTEXT CHARACTER SET latin1 値。

例

```
mysql> SELECT sys.ps_thread_stack(37, FALSE) AS thread_stack\G
***** 1. row *****
```

```
thread_stack: {"rankdir": "LR", "nodesep": "0.10",
"stack_created": "2014-02-19 13:39:03", "mysql_version": "8.0.2-dmr-debug-log",
"mysql_user": "root@localhost", "events": [{"nesting_event_id": "0",
"event_id": "10", "timer_wait": 256.35, "event_info": "sql/select",
"wait_info": "select @@version comment limit 1\nerrors: 0\nwarnings: 0\nlock time:
...

```

28.4.5.17 ps_thread_trx_info() 関数

指定されたスレッドに関する情報を含む JSON オブジェクトを返します。この情報には、現在のトランザクションと、パフォーマンススキーマ `events_transactions_current` および `events_statements_history` テーブルから導出された、すでに実行されているステートメントが含まれます。(JSON オブジェクトの完全なデータを取得するには、これらのテーブルのコンシューマを有効にする必要があります。)

出力が切捨て長 (デフォルトで 65535) を超えると、次のような JSON エラーオブジェクトが返されます:

```
{ "error": "Trx info truncated: Row 6 was cut by GROUP_CONCAT()" }
```

関数の実行中に発生したその他の警告および例外についても、同様のエラーオブジェクトが返されます。

パラメータ

- `in_thread_id` **BIGINT UNSIGNED**: トランザクション情報を返すスレッド ID。この値は、一部のパフォーマンススキーマ `threads` テーブル行の `THREAD_ID` カラムと一致する必要があります。

構成オプション

`ps_thread_trx_info()` 操作は、次の構成オプションまたは対応するユーザー定義変数を使用して変更できます ([セクション 28.4.2.1 「sys_config テーブル」](#) を参照):

- `ps_thread_trx_info.max_length`, `@sys.ps_thread_trx_info.max_length`

出力の最大長。デフォルトは 65535 です。

戻り値

LONGTEXT 値。

例

```
mysql> SELECT sys.ps_thread_trx_info(48)\G
***** 1. row *****
sys.ps_thread_trx_info(48): [
  {
    "time": "790.70 us",
    "state": "COMMITTED",
    "mode": "READ WRITE",
    "autocommitted": "NO",
    "gtid": "AUTOMATIC",
    "isolation": "REPEATABLE READ",
    "statements_executed": [
      {
        "sql_text": "INSERT INTO info VALUES (1, 'foo')",
        "time": "471.02 us",
        "schema": "trx",
        "rows_examined": 0,
        "rows_affected": 1,
        "rows_sent": 0,
        "tmp_tables": 0,
        "tmp_disk_tables": 0,
        "sort_rows": 0,
        "sort_merge_passes": 0
      },
      {
        "sql_text": "COMMIT",
        "time": "254.42 us",
        "schema": "trx",
        "rows_examined": 0,

```



```
"rows_affected": 0,
"rows_sent": 0,
"tmp_tables": 0,
"tmp_disk_tables": 0,
"sort_rows": 0,
"sort_merge_passes": 0
}
],
{
  "time": "426.20 us",
  "state": "COMMITTED",
  "mode": "READ WRITE",
  "autocommitted": "NO",
  "gtid": "AUTOMATIC",
  "isolation": "REPEATABLE READ",
  "statements_executed": [
    {
      "sql_text": "INSERT INTO info VALUES (2, 'bar')",
      "time": "107.33 us",
      "schema": "trx",
      "rows_examined": 0,
      "rows_affected": 1,
      "rows_sent": 0,
      "tmp_tables": 0,
      "tmp_disk_tables": 0,
      "sort_rows": 0,
      "sort_merge_passes": 0
    },
    {
      "sql_text": "COMMIT",
      "time": "213.23 us",
      "schema": "trx",
      "rows_examined": 0,
      "rows_affected": 0,
      "rows_sent": 0,
      "tmp_tables": 0,
      "tmp_disk_tables": 0,
      "sort_rows": 0,
      "sort_merge_passes": 0
    }
  ]
}
]
```

28.4.5.18 quote_identifier() 関数

文字列引数を指定すると、この関数は SQL ステートメントに含めるのに適した引用符付き識別子を生成します。これは、識別子として使用される値が予約語であるか、バックティック (`) 文字を含む場合に便利です。

パラメータ

in_identifier TEXT: 引用する識別子。

戻り値

TEXT 値。

例

```
mysql> SELECT sys.quote_identifier('plain');
+-----+
| sys.quote_identifier('plain') |
+-----+
| `plain`                       |
+-----+
mysql> SELECT sys.quote_identifier('trick`ier');
+-----+
| sys.quote_identifier('trick`ier') |
+-----+
| `trick``ier`                   |
+-----+
```

```
+-----+
mysql> SELECT sys.quote_identifier('integer');
+-----+
| sys.quote_identifier('integer') |
+-----+
| `integer`                       |
+-----+
```

28.4.5.19 sys_get_config() 関数

構成オプション名を指定すると、`sys_config` テーブルからオプション値を返します。オプションがテーブルに存在しない場合は、指定されたデフォルト値 (`NULL` の可能性があります) を返します。

`sys_get_config()` がデフォルト値を返し、その値が `NULL` である場合、呼出し側は指定された構成オプションの `NULL` を処理できると予想されます。

慣例上、`sys_get_config()` をコールするルーチンはまず、対応するユーザー定義変数が存在し、`NULL` 以外であるかどうかをチェックします。その場合、ルーチンは `sys_config` テーブルを読み取らずに変数値を使用します。変数が存在しないか、`NULL` である場合、ルーチンはテーブルからオプション値を読み取り、ユーザー定義変数をその値に設定します。構成オプションとそれに対応するユーザー定義変数の関係の詳細は、[セクション28.4.2.1「sys_config テーブル」](#)を参照してください。

構成オプションがすでに設定されているかどうかを確認し、設定されていない場合は `sys_get_config()` の戻り値を使用する場合は、`IFNULL(...)` を使用できます (後の例を参照)。ただし、最初の反復でのみ代入が必要な繰返しコールの場合、`IFNULL(...)` の使用は `IF (...) THEN ... END IF;` ブロックを使用するより大幅に遅くなることが予想されるため、ループ内で (たとえば、結果セットの各行に対して)、これを実行しないでください (後述の例を参照)。

パラメータ

- `in_variable_name` `VARCHAR(128)`: 値を返す構成オプションの名前。
- `in_default_value` `VARCHAR(128)`: 構成オプションが `sys_config` テーブルに見つからない場合に返すデフォルト値。

戻り値

`VARCHAR(128)` 値。

例

オプションがテーブルに存在しない場合、`sys_config` テーブルから構成値を取得し、デフォルトとして 128 に戻します:

```
mysql> SELECT sys.sys_get_config('statement_truncate_len', 128) AS Value;
+-----+
| Value |
+-----+
| 64    |
+-----+
```

ワンライナーの例: オプションがすでに設定されているかどうかを確認します。設定されていない場合は、(`sys_config` テーブルの値を使用して)`IFNULL(...)` 結果を割り当てます:

```
mysql> SET @sys.statement_truncate_len =
IFNULL(@sys.statement_truncate_len,
sys.sys_get_config('statement_truncate_len', 64));
```

`IF (...) THEN ... END IF;` ブロックの例: オプションがすでに設定されているかどうかを確認します。設定されていない場合は、`sys_config` テーブルから値を割り当てます:

```
IF (@sys.statement_truncate_len IS NULL) THEN
SET @sys.statement_truncate_len = sys.sys_get_config('statement_truncate_len', 64);
END IF;
```

28.4.5.20 version_major() 関数

この関数は、MySQL サーバーのメジャーバージョンを返します。

パラメータ

なし

戻り値

TINYINT UNSIGNED 値。

例

```
mysql> SELECT VERSION(), sys.version_major();
+-----+
| VERSION() | sys.version_major() |
+-----+
| 8.0.13-debug | 8 |
+-----+
```

28.4.5.21 version_minor() 関数

この関数は、MySQL サーバーのマイナーバージョンを戻します。

パラメータ

なし

戻り値

TINYINT UNSIGNED 値。

例

```
mysql> SELECT VERSION(), sys.version_minor();
+-----+
| VERSION() | sys.version_minor() |
+-----+
| 8.0.13-debug | 0 |
+-----+
```

28.4.5.22 version_patch() 関数

この関数は、MySQL サーバーのパッチリリースバージョンを戻します。

パラメータ

なし

戻り値

TINYINT UNSIGNED 値。

例

```
mysql> SELECT VERSION(), sys.version_patch();
+-----+
| VERSION() | sys.version_patch() |
+-----+
| 8.0.13-debug | 13 |
+-----+
```


第 29 章 Connector および API

目次

29.1 MySQL Connector/C++	4524
29.2 MySQL Connector/J	4524
29.3 MySQL Connector/NET	4524
29.4 MySQL Connector/ODBC	4524
29.5 MySQL Connector/Python	4525
29.6 MySQL Connector/Node.js	4525
29.7 MySQL C API	4525
29.8 MySQL PHP API	4525
29.9 MySQL Perl API	4525
29.10 MySQL Python API	4526
29.11 MySQL Ruby API	4526
29.11.1 MySQL/Ruby API	4526
29.11.2 Ruby/MySQL API	4526
29.12 MySQL Tcl API	4526
29.13 MySQL Eiffel ラッパー	4526

MySQL Connector はクライアントプログラムに MySQL サーバーへの接続を提供します。API は、クラシック MySQL プロトコル または X プロトコル を使用した MySQL リソースへの低レベルのアクセスを提供します。コネクタと API の両方を使用すると、ODBC、Java (JDBC)、C++、Python、Node.js、PHP、Perl、Ruby、C などの別の言語または環境から MySQL ステートメントに接続して実行できます。

MySQL Connector

Oracle では多数のコネクタを開発しています。

- [Connector/C++](#) により、C++ アプリケーションは MySQL に接続できます。
- [Connector/J](#) は標準 JDBC (Java Database Connectivity) API を使用して、Java アプリケーションから、MySQL に接続するためのドライバサポートを提供します。
- [Connector/NET](#) を使用すると、開発者は MySQL に接続する .NET アプリケーションを作成できます。Connector/NET は、完全に機能する ADO.NET インタフェースを実装し、ADO.NET 対応ツールでの使用をサポートします。Connector/NET を使用するアプリケーションは、サポートされている任意の .NET 言語で記述できます。

[MySQL for Visual Studio](#) は、Connector/NET および Microsoft Visual Studio 2012、2013、2015 および 2017 で動作します。MySQL for Visual Studio は、Visual Studio から MySQL オブジェクトおよびデータへのアクセスを提供します。Visual Studio パッケージとして、サーバーエクスプローラに直接統合され、新しい接続を作成して MySQL データベースオブジェクトを操作できます。

- [Connector/ODBC](#) は ODBC (Open Database Connectivity) API を使用して、MySQL に接続するためのドライバサポートを提供します。Windows、Unix および macOS プラットフォームからの ODBC 接続をサポートしています。
- [Connector/Python](#) は [Python DB API バージョン 2.0](#) に準拠する API を使用して、Python アプリケーションから、MySQL に接続するためのドライバサポートを提供します。追加の Python モジュールまたは MySQL クライアントライブラリは必要ありません。
- [Connector/Node.js](#) には、X プロトコル を使用して Node.js アプリケーションから MySQL に接続するための非同期 API が用意されています。Connector/Node.js では、データベースセッションおよびスキーマの管理、MySQL ドキュメントストアコレクションの操作および RAW SQL ステートメントの使用がサポートされています。

MySQL C API

C アプリケーション内で MySQL をネイティブに使用するための直接アクセスのために、[C API](#) は `libmysqlclient` クライアントライブラリを介して MySQL クライアント/サーバープロトコルへの低レベルのアクセスを提供します。これ

は、MySQL サーバーのインスタンスに接続するために使用する主な方法で、MySQL コマンド行クライアントと、ここで詳しく説明している多くの MySQL Connector およびサードパーティー API のどちらにも使用されています。

`libmysqlclient` は、MySQL ディストリビューションに含まれています。

[MySQL C API Implementations](#) も参照してください。

C アプリケーションから MySQL にアクセスするか、この章の Connector や API でサポートされていない言語で、MySQL へのインタフェースを構築するには、[C API](#) から始めます。このプロセスに役立つ、多くのプログラム向けユーティリティがあります。[セクション4.7「プログラム開発ユーティリティ」](#)を参照してください。

サードパーティー MySQL API

この章で説明している残りの API は、特定のアプリケーション言語から MySQL へのインタフェースを提供します。これらのサードパーティーソリューションは Oracle で開発されていないが、サポートされていません。それらの使用と機能に関する基本情報は、参考目的のみここで提供しています。

すべてのサードパーティー言語 API は、`libmysqlclient` を使用するか、または ネイティブドライバを実装するか、2つの方法のいずれかを使用して開発されています。2つのソリューションには異なるメリットがあります。

- `libmysqlclient` は MySQL クライアントアプリケーションと同じライブラリを使用するため、MySQL と完全に互換性があります。ただし、機能セットは、`libmysqlclient` から公開された実装とインタフェースに制限され、データがネイティブ言語と MySQL API コンポーネント間でコピーされるため、パフォーマンスが低下することがあります。
- ネイティブドライバはホスト言語または環境内に完全に収まる MySQL ネットワークプロトコルの実装です。ネイティブドライバはコンポーネント間でのデータのコピーが少ないため高速であり、標準 MySQL API によって使用できない高度な機能を提供できます。さらに、ネイティブドライバコンポーネントの構築には、MySQL クライアントライブラリのコピーが必要ないため、ネイティブドライバは、エンドユーザーにとって構築とデプロイが簡単です。

[表29.1「MySQL API およびインタフェース」](#)には、MySQL で使用可能なライブラリおよびインタフェースの多くがリストされています。

表 29.1 MySQL API およびインタフェース

環境	API	型	メモ
Ada	GNU Ada MySQL バインディング	libmysqlclient	GNU Ada 用の MySQL バインディングに関するドキュメント を参照してください。
C	C API	libmysqlclient	MySQL 8.0 C API Developer Guide を参照してください。
C++	Connector/C++	libmysqlclient	MySQL Connector/C++ 8.0 Developer Guide を参照してください。
	MySQL++	libmysqlclient	「MySQL++ web サイト」 を参照してください。
	MySQL wrapped	libmysqlclient	「MySQL wrapped」 を参照してください。
Cocoa	MySQL-Cocoa	libmysqlclient	Objective-C Cocoa 環境と互換性があります。 http://mysql-cocoa.sourceforge.net/ を参照してください
D	MySQL for D	libmysqlclient	MySQL for D を参照してください。

環境	API	型	メモ
Eiffel	Eiffel MySQL	libmysqlclient	セクション29.13「MySQL Eiffel ラッパー」を参照してください。
Erlang	erlang-mysql-driver	libmysqlclient	「 erlang-mysql-driver 」を参照してください。
Haskell	Haskell MySQL バインディング	ネイティブドライバ	Brian O'Sullivan のピュア Haskell MySQL バインディングに関するドキュメントを参照してください。
	hsqldb-mysql	libmysqlclient	「 Haskell 用の MySQL ドライバ 」を参照してください。
Java/JDBC	Connector/J	ネイティブドライバ	MySQL Connector/J 5.1 Developer Guide を参照してください。
Kaya	MyDB	libmysqlclient	MyDB に関するドキュメント を参照してください。
Lua	LuaSQL	libmysqlclient	LuaSQL を参照してください。
.NET/Mono	Connector/NET	ネイティブドライバ	「 MySQL Connector/NET Developer Guide 」を参照してください。
Objective Caml	Objective Caml MySQL バインディング	libmysqlclient	Objective Caml 用の MySQL バインディングに関するドキュメントを参照してください。
Octave	GNU Octave 用データベースバインディング	libmysqlclient	GNU Octave 用データベースバインディングに関するドキュメントを参照してください。
ODBC	Connector/ODBC	libmysqlclient	「 MySQL Connector/ODBC Developer Guide 」を参照してください。
Perl	DBI/DBD::mysql	libmysqlclient	セクション29.9「 MySQL Perl API 」を参照してください。
	Net::MySQL	ネイティブドライバ	CPAN の「 Net::MySQL 」を参照してください。
PHP	mysql 、 ext/mysql インタフェース (非推奨)	libmysqlclient	「 MySQL and PHP 」を参照してください。
	mysqli 、 ext/mysqli インタフェース	libmysqlclient	「 MySQL and PHP 」を参照してください。
	PDO_MYSQL	libmysqlclient	「 MySQL and PHP 」を参照してください。
	PDO mysqlnd	ネイティブドライバ	
Python	Connector/Python	ネイティブドライバ	「 MySQL Connector/Python Developer Guide 」を参照してください。

環境	API	型	メモ
Python	Connector/Python C 拡張機能	libmysqlclient	「 MySQL Connector/Python Developer Guide 」を参照してください。
	MySQLdb	libmysqlclient	セクション29.10「MySQL Python API」 を参照してください。
Ruby	MySQL/Ruby	libmysqlclient	libmysqlclient を使用します。 セクション29.11.1「MySQL/Ruby API」 を参照してください。
	Ruby/MySQL	ネイティブドライバ	セクション29.11.2「Ruby/MySQL API」 を参照してください。
Scheme	Myscsh	libmysqlclient	Myscsh に関するドキュメント を参照してください。
SPL	sql_mysql	libmysqlclient	SPL の sql_mysql を参照してください。
Tcl	MySQLtcl	libmysqlclient	セクション29.12「MySQL Tcl API」 を参照してください。

29.1 MySQL Connector/C++

MySQL Connector/C++ マニュアルは、MySQL リファレンスマニュアルの一部としてではなく、スタンドアロン形式で公開されています。詳細については、これらのドキュメントを参照してください。

- 主要マニュアル: [MySQL Connector/C++ 8.0 Developer Guide](#)
- リリースノート: [MySQL Connector/C++ リリースノート](#)

29.2 MySQL Connector/J

MySQL Connector/J マニュアルは、MySQL リファレンスマニュアルの一部としてではなく、スタンドアロン形式で公開されています。詳細については、これらのドキュメントを参照してください。

- メインマニュアル: [MySQL Connector/J 開発者ガイド](#)
- リリースノート: [MySQL Connector/J リリースノート](#)

29.3 MySQL Connector/NET

MySQL Connector/NET マニュアルは、MySQL リファレンスマニュアルの一部としてではなく、スタンドアロン形式で公開されています。詳細については、これらのドキュメントを参照してください。

- 主要マニュアル: [MySQL Connector/NET Developer Guide](#)
- リリースノート: [MySQL Connector/NET リリースノート](#)

29.4 MySQL Connector/ODBC

MySQL Connector/ODBC マニュアルは、MySQL リファレンスマニュアルの一部としてではなく、スタンドアロン形式で公開されています。詳細については、これらのドキュメントを参照してください。

- 主要マニュアル: [MySQL Connector/ODBC Developer Guide](#)
- リリースノート: [MySQL Connector/ODBC リリースノート](#)

29.5 MySQL Connector/Python

MySQL Connector/Python マニュアルは、MySQL リファレンスマニュアルの一部としてではなく、スタンドアロン形式で公開されています。詳細については、これらのドキュメントを参照してください。

- 主要マニュアル: [MySQL Connector/Python Developer Guide](#)
- リリースノート: [MySQL Connector/Python リリースノート](#)

29.6 MySQL Connector/Node.js

MySQL Connector/Node.js マニュアルは、MySQL リファレンスマニュアルの一部としてではなく、スタンドアロン形式で公開されています。詳細については、これらのドキュメントを参照してください。

- リリースノート: [MySQL Connector/Node.js リリースノート](#)

29.7 MySQL C API

『MySQL C API 開発者ガイド』は、MySQL リファレンスマニュアルの一部としてではなく、スタンドアロン形式で公開されています。 [MySQL 8.0 C API Developer Guide](#)を参照してください。

29.8 MySQL PHP API

MySQL PHP API のマニュアルは、MySQL リファレンスマニュアルの一部としてではなく、独立した形式で発行されるようになりました。 [MySQL and PHP](#)を参照してください。

29.9 MySQL Perl API

PerlDBIモジュールはデータベースアクセスのための一般的なインタフェースを提供します。変更せずに、多くのさまざまなデータベースエンジンで動作する DBI スクリプトを書くことができます。MySQL で DBI を使用するには、次をインストールします。

1. DBI モジュール。
2. `DBD::mysql` モジュール。これは Perl のデータベースドライバ (DBD) モジュールです。
3. オプションで、アクセスするほかの任意の種類のデータベースサーバーの DBD モジュール。

Perl DBI は推薦される Perl インタフェースです。それは、廃止とみなされるべき `mysqlperl` と呼ばれる古いインタフェースを置き換えます。

これらのセクションには、MySQL と Perl の使用および Perl での MySQL アプリケーションの作成に関する情報が含まれます。

- Perl DBI サポートのインストール手順については、[セクション2.13「Perl のインストールに関する注釈」](#)を参照してください。
- オプションファイルからオプションを読み取る例については、[セクション5.8.4「複数サーバー環境でのクライアントプログラムの使用」](#)を参照してください。
- セキュアなコーディングのヒントについては、[セクション6.1.1「セキュリティガイドライン」](#)を参照してください。
- デバッグのヒントについては、[セクション5.9.1.4「gdb での mysqld のデバッグ」](#)を参照してください。
- いくつかの Perl 固有の環境変数については、[セクション4.9「環境変数」](#)を参照してください。
- macOS での実行に関する考慮事項は、[セクション2.4「macOS への MySQL のインストール」](#)を参照してください。
- 文字列リテラルを引用する方法については、[セクション9.1.1「文字列リテラル」](#)を参照してください。

DBI 情報はコマンド行、オンライン、または印刷物で取得できます。

- DBI と `DBD::mysql` モジュールをインストールしたら、コマンド行で `perldoc` コマンドを使って、それらに関する情報を得ることができます。

```
shell> perldoc DBI
shell> perldoc DBI::FAQ
shell> perldoc DBD::mysql
```

`pod2man`、`pod2html` などを使用して、この情報をほかのフォーマットに変換することもできます。

- Perl DBI のオンライン情報は、DBI web サイト (<http://dbi.perl.org/>) を参照してください。そのサイトでは一般的な DBI メーリングリストをホストしています。
- 印刷された情報として、公式の DBI の書籍は『Programming the Perl DBI』(Alligator Descartes および Tim Bunce 著、O'Reilly & Associates 発行、2000 年) があります。このマニュアルに関する情報は、DBI web サイト <http://dbi.perl.org/> で入手できます。

29.10 MySQL Python API

`MySQLdb` は Python DB API バージョン 2.0 に準拠した MySQL の Python のサポートを提供するサードパーティードライバです。それは <http://sourceforge.net/projects/mysql-python/> にあります。

新しい MySQL Connector/Python コンポーネントは同じ Python API へのインタフェースを提供し、MySQL サーバーに組み込まれ、Oracle によってサポートされます。Connector の詳細と、Python アプリケーションのコーディングガイドラインおよびサンプル Python コードについては、[MySQL Connector/Python Developer Guide](#) を参照してください。

29.11 MySQL Ruby API

MySQL アプリケーションを開発する Ruby プログラマは 2 つの API を使用できます。

- MySQL/Ruby API は `libmysqlclient` API ライブラリに基づいています。MySQL/Ruby API のインストールと使用については、[セクション29.11.1「MySQL/Ruby API」](#) を参照してください。
- Ruby/MySQL API はネイティブ MySQL ネットワークプロトコル (ネイティブドライバ) を使用するために書かれています。Ruby/Ruby API のインストールと使用については、[セクション29.11.2「Ruby/MySQL API」](#) を参照してください。

Ruby 言語に関するバックグラウンドと構文情報については、[Ruby プログラミング言語](#) を参照してください。

29.11.1 MySQL/Ruby API

MySQL/Ruby モジュールは、`libmysqlclient` 経由で Ruby を使用した MySQL データベースへのアクセスを提供します。

モジュールのインストールと公開されている関数については、[「MySQL/Ruby」](#) を参照してください。

29.11.2 Ruby/MySQL API

MySQL/Ruby モジュールは、MySQL ネットワークプロトコルを使用して、ネイティブドライバインタフェース経由で、Ruby を使用した MySQL データベースへのアクセスを提供します。

モジュールのインストールと公開されている関数については、[「Ruby/MySQL」](#) を参照してください。

29.12 MySQL Tcl API

`MySQLtcl` は [Tcl プログラミング言語](#) から MySQL データベースサーバーにアクセスするための簡単な API です。それは <http://www.xdobry.de/mysqltcl/> にあります。

29.13 MySQL Eiffel ラッパー

このページは機械翻訳したものです。

MySQL Eiffel ラッパー

Eiffel MySQL は、Michael Ravits によって書かれた Eiffel プログラミング言語を使用した MySQL データベースサーバーへのインタフェースです。それは <http://efsa.sourceforge.net/archive/ravits/mysql.htm> にあります。

第 30 章 MySQL Enterprise Edition

目次

30.1 MySQL Enterprise Monitor の概要	4529
30.2 MySQL Enterprise Backup の概要	4530
30.3 MySQL Enterprise Security の概要	4531
30.4 MySQL Enterprise Encryption の概要	4531
30.5 MySQL Enterprise Audit の概要	4531
30.6 MySQL Enterprise Firewall の概要	4531
30.7 MySQL Enterprise Thread Pool の概要	4532
30.8 MySQL Enterprise Data Masking and De-Identification の概要	4532

MySQL Enterprise Edition は商用製品です。MySQL Community Edition と同様に、MySQL Enterprise Edition には、完全なコミット、ロールバック、クラッシュリカバリおよび行レベルロック機能を備えた完全に統合されたトランザクションセーフな ACID 準拠のデータベースである MySQL Server が含まれます。また、MySQL Enterprise Edition には、監視およびオンラインバックアップを提供するように設計された次のコンポーネントと、セキュリティおよびスケーラビリティの向上が含まれています:

以降のセクションでは、これらの各コンポーネントについて簡単に説明し、より詳細な情報がある場所を示します。商用製品の詳細は、<https://www.mysql.com/products/> を参照してください。

- [MySQL Enterprise Monitor](#)
- [MySQL Enterprise Backup](#)
- [MySQL Enterprise Security](#)
- [MySQL Enterprise Encryption](#)
- [MySQL Enterprise Audit](#)
- [MySQL Enterprise Firewall](#)
- [MySQL Enterprise Thread Pool](#)
- [MySQL Enterprise Data Masking and De-Identification](#)

30.1 MySQL Enterprise Monitor の概要

MySQL Enterprise Monitor は、MySQL のエンタープライズモニタリングシステムで、MySQL サーバーに目を通し、潜在的な問題と問題を通知して、問題の修正方法をアドバイスします。MySQL Enterprise Monitor は、ビジネスにとって重要な単一の MySQL サーバーから、ビジョ状態の web サイトに電源を供給する MySQL サーバーの大規模なファームまで、あらゆる種類の構成を監視できます。

以降では、MySQL Enterprise Monitor 製品を構成する基本的なコンポーネントについて簡単に概要を示します。詳細は、<https://dev.mysql.com/doc/mysql-monitor/en/> で閲覧できる MySQL Enterprise Monitor のマニュアルを参照してください。

MySQL Enterprise Monitor コンポーネントは、データベースおよびネットワークのトポロジに応じてさまざまな構成でインストールでき、データベースサーバーのマシンのオーバーヘッドを最小限にして、信頼できる応答のよいモニタリングデータの最適な組み合わせが提供されます。MySQL Enterprise Monitor の一般的なインストール環境は次のもので構成されています。

- モニターする 1 つ以上の MySQL サーバー。MySQL Enterprise Monitor は MySQL サーバーの Community リリースおよび Enterprise リリースの両方をモニターできます。
- モニターされる各ホストの MySQL Enterprise Monitor Agent。

- エージェントからの情報を照合して、収集されたデータへのユーザーインターフェースを提供する単一の MySQL Enterprise Service Manager。

MySQL Enterprise Monitor は 1 つ以上の MySQL サーバーをモニターするように設計されています。モニタリング情報は、エージェント MySQL Enterprise Monitor Agent を使用して収集されます。このエージェントは、モニターするホストおよび MySQL サーバーとやり取りして、変数、ステータス、およびヘルス情報を収集し、この情報を MySQL Enterprise Service Manager に送信します。

エージェントによって収集されたモニタリングしている各 MySQL サーバーおよびホストの情報は、MySQL Enterprise Service Manager に送信されます。このサーバーは、エージェントからのすべての情報を照合します。エージェントによって送信された情報を照合するときに、MySQL Enterprise Service Manager はサーバーのステータスを適切な値と比較して、収集されたデータを頻繁にテストします。しきい値に達すると、サーバーはイベント（アラームおよび通知を含む）をトリガーして潜在的な問題（メモリー不足、高い CPU 使用率、より複雑な状況（バッファサイズ不足、ステータス情報など）など）を強調できます。各テストは、それに関連付けられているしきい値と合わせて、ルールと呼ばれます。

これらのルール、アラーム、および通知は、それぞれ MySQL Enterprise Advisor と呼ばれます。Advisors は MySQL Enterprise Service Manager の重要な部分を形成しており、警告情報および潜在的な問題に関するトラブルシューティングの推奨事項を提供します。

MySQL Enterprise Service Manager には Web サーバーが含まれており、ユーザーは Web ブラウザを使用してやり取りします。このインターフェース (MySQL Enterprise Monitor User Interface) には、エージェントによって収集されたすべての情報が表示され、すべてのサーバーおよびそれらの現在のステータスをグループで表示したり、個別に表示したりできます。サービスのすべての特性を MySQL Enterprise Monitor User Interface を使用して制御および構成します。

MySQL Enterprise Monitor Agent の処理によって提供される情報には、グラフ形式で表示できる統計およびクエリーの情報も含まれています。たとえば、サーバーの負荷、クエリーの数、インデックスの使用状況の情報などの特性をグラフとして時間の経過に従って表示できます。このグラフを使用すると、サーバー上での問題や潜在的な問題が正確に特定され、特定の期間のデータを検査することによって、データベースの問題または外部の問題（外部システムまたはネットワーク障害など）からの影響を診断するために役立ちます。

MySQL Enterprise Monitor Agent は、サーバーで実行されたクエリーに関する詳細な情報（行数および各クエリーを実行するためにかかった実行時間を含む）を収集するように構成することもできます。詳細なクエリーデータをグラフィカルな情報に関連付けることによって、著しく高い負荷やインデックスなどの問題が発生したときに実行されていたクエリーを識別できます。クエリーデータはクエリーアナライザと呼ばれるシステムによってサポートされており、ニーズに応じて異なる方法でデータを表示できます。

30.2 MySQL Enterprise Backup の概要

MySQL Enterprise Backup は、MySQL データベースに対してホットバックアップ操作を実行します。この製品は、InnoDB ストレージエンジンによって作成されたテーブルの信頼できる効率的なバックアップが行われるように設計されています。補完するために、MyISAM およびほかのストレージエンジンのテーブルをバックアップすることもできます。

以降では、MySQL Enterprise Backup の概要を簡単に示します。詳細は、<https://dev.mysql.com/doc/mysql-enterprise-backup/en/> で閲覧できる MySQL Enterprise Backup のマニュアルを参照してください。

ホットバックアップは、データベースが実行されていて、アプリケーションがそれに対して読み取りおよび書き込みを行なっている間に実行します。このタイプのバックアップは、通常のデータベースの操作をブロックせず、バックアップが行われているときに発生したすべての変更を取得します。これらの理由から、データベース「拡大」の場合はホットバックアップが望ましいです。データがバックアップにかなりの時間を要する大きさであり、ビジネスにとってデータが十分重要な場合は、アプリケーション、web サイトまたは Web サービスをオフラインにせずに、最後のすべての変更を取得する必要があります。

MySQL Enterprise Backup は、InnoDB ストレージエンジンを使用するすべてのテーブルのホットバックアップを実行します。MyISAM またはその他の InnoDB 以外のストレージエンジンを使用しているテーブルの場合、データベースの実行は継続されるがバックアップされている間はテーブルを変更できない「ウォーム」バックアップが行われます。バックアップ操作を効率的にするために、InnoDB を新しいテーブルのデフォルトのストレージエンジンに指定するか、InnoDB ストレージエンジンを使用するように既存のテーブルを変更できます。

30.3 MySQL Enterprise Security の概要

MySQL Enterprise Edition には、外部サービスを使用してセキュリティ機能を実装するプラグインが用意されています:

- MySQL Enterprise Edition には、MySQL Server が PAM (Pluggable Authentication Module) を使用して MySQL ユーザーを認証できるようにする認証プラグインが含まれています。PAM を使用すると、システムは標準インタフェースを使用して、さまざまな種類の認証方式 (Unix パスワードや LDAP ディレクトリなど) にアクセスできます。詳細は、[セクション6.4.1.5 「PAM プラガブル認証」](#)を参照してください。
- MySQL Enterprise Edition には、Windows で外部認証を実行する認証プラグインが含まれているため、MySQL Server はネイティブ Windows サービスを使用してクライアント接続を認証できます。Windows にログインしたユーザーは、追加のパスワードを指定せずに、自分の環境内の情報に基づいて MySQL クライアントプログラムからサーバーに接続できます。詳細は、[セクション6.4.1.6 「Windows プラガブル認証」](#)を参照してください。
- MySQL Enterprise Edition には、SQL レベルで OpenSSL 機能を公開する OpenSSL ライブラリに基づく一連の暗号化機能が含まれています。これらの関数を使用すると、不明瞭化 (識別特性の削除)、フォーマットされたランダムデータの生成、データの置換または置換など、複数の方法を使用して既存のデータをマスキングできます。詳細は、[セクション30.4 「MySQL Enterprise Encryption の概要」](#)を参照してください。
- MySQL Enterprise Edition 5.7 以上には、Oracle Key Vault をキーリング記憶域のバックエンドとして使用するキーリングプラグインが含まれています。詳細は、[セクション6.4.4 「MySQL キーリング」](#)を参照してください。

ほかの関連するエンタープライズセキュリティー機能については、[セクション30.4 「MySQL Enterprise Encryption の概要」](#)を参照してください。

30.4 MySQL Enterprise Encryption の概要

MySQL Enterprise Edition には、SQL レベルで OpenSSL 機能を公開する OpenSSL ライブラリに基づく一連の暗号化機能が含まれています。これらの関数を使用することによって、エンタープライズアプリケーションが次の操作を実行できるようになります。

- 公開鍵非対称暗号方式を使用した、追加のデータ保護の実装
- 公開鍵、秘密鍵、およびデジタル署名の作成
- 非対称暗号化および非対称復号化の実行
- デジタル署名およびデータの検証や妥当性検査に対する暗号化ハッシュの使用

詳細は、[セクション6.6 「MySQL Enterprise Encryption」](#)を参照してください。

ほかの関連するエンタープライズセキュリティー機能については、[セクション30.3 「MySQL Enterprise Security の概要」](#)を参照してください。

30.5 MySQL Enterprise Audit の概要

MySQL Enterprise Edition には、サーバープラグインを使用して実装される MySQL Enterprise Audit が含まれています。MySQL Enterprise Audit では、オープン MySQL 監査 API を使用して、特定の MySQL サーバーで実行される接続およびクエリーアクティビティの標準のポリシーベースの監視およびロギングを有効にします。MySQL Enterprise Audit は、Oracle 監査仕様を満たすように設計されており、内部および外部の規制ガイドラインによって管理されるアプリケーションに対して、すぐに使用できる監査およびコンプライアンスソリューションを提供します。

インストール時に監査プラグインを使用すると、MySQL サーバーはサーバーアクティビティの監査レコードを含むログファイルを生成できます。ログの内容には、クライアントが接続および切断した時間、接続中に実行したアクション (アクセスしたデータベースおよびテーブルなど) が含まれます。

詳細は、[セクション6.4.5 「MySQL Enterprise Audit」](#)を参照してください。

30.6 MySQL Enterprise Firewall の概要

MySQL Enterprise Edition には、アプリケーションレベルのファイアウォールである MySQL Enterprise Firewall が含まれています。これにより、データベース管理者は、受け入れられたステートメントパターンの許可リストに対する照合に基づいて SQL ステートメントの実行を許可または拒否できます。これにより、SQL インジェクションなどの攻撃や、正当なクエリーワークロード特性の外部でアプリケーションを使用することで、アプリケーションを利用しようとする攻撃に対して MySQL Server を強化できます。

ファイアウォールに登録された各 MySQL アカウントには独自のステートメント allowlist があり、アカウントごとに保護を調整できます。特定のアカウントについて、ファイアウォールは、承認されたステートメントパターンでのトレーニングまたは許可できないステートメントに対する保護のために、記録または保護モードで動作できます。

詳細は、[セクション6.4.7「MySQL Enterprise Firewall」](#)を参照してください。

30.7 MySQL Enterprise Thread Pool の概要

MySQL Enterprise Edition には、サーバープラグインを使用して実装される MySQL Enterprise Thread Pool が含まれています。MySQL サーバーのデフォルトのスレッド処理モデルでは、クライアント接続ごとに 1 つのスレッドを使用してステートメントが実行されます。より多くのクライアントがサーバーに接続してステートメントを実行すると、全体的なパフォーマンスが低下します。MySQL Enterprise Edition では、スレッドプールプラグインは、オーバーヘッドを減らしてパフォーマンスを向上させるために設計された代替のスレッド処理モデルを提供します。このプラグインは、多数のクライアント接続に対してステートメント実行スレッドを効率的に管理することによってサーバーのパフォーマンスを向上させるスレッドプールを実装します。

詳細は、[セクション5.6.3「MySQL Enterprise Thread Pool」](#)を参照してください。

30.8 MySQL Enterprise Data Masking and De-Identification の概要

MySQL Enterprise Edition 5.7 以上には、プラグインおよび一連のユーザー定義関数を含むプラグインライブラリとして実装される MySQL Enterprise Data Masking and De-Identification が含まれています。データマスキングでは、実際の値を置換で置換することで機密情報が非表示になります。MySQL Enterprise Data Masking and De-Identification 関数を使用すると、不明瞭化 (識別特性の削除)、フォーマットされたランダムデータの生成、データの置換または置換など、いくつかの方法を使用して既存のデータをマスキングできます。

詳細は、[セクション6.5「MySQL Enterprise Data Masking and De-Identification」](#)を参照してください。

第 31 章 MySQL Workbench

MySQL Workbench には、MySQL サーバーおよびデータベースを操作するためのグラフィカルツールが用意されています。MySQL Workbench は、MySQL バージョン 5.5 以上を完全にサポートしています。

以降では、MySQL Workbench の機能について簡単に説明します。詳細は、<https://dev.mysql.com/doc/workbench/en/> で閲覧できる MySQL Workbench のマニュアルを参照してください。

MySQL Workbench は 5 つの主な機能領域を提供しています。

- SQL の開発: データベースサーバーへの接続を作成および管理できます。接続パラメータを構成できるほかに、MySQL Workbench は組み込みの SQL Editor を使用してデータベース接続で SQL クエリーを実行する機能を提供しています。この機能は、以前は Query Browser スタンドアロンアプリケーションによって提供されていた機能と置き換わるものです。
- データモデリング: データベーススキーマのモデルのグラフィカルな作成、スキーマとライブデータベースの間のリバースエンジニアリングとフォワードエンジニアリング、および包括的な Table Editor を使用したデータベースのすべての特性の編集を行うことができます。Table Editor は、テーブル、カラム、インデックス、トリガー、パーティション化、オプション、挿入と権限、ルーチン、およびビューを編集するための使いやすい機能を提供しています。
- サーバー管理: サーバーインスタンスを作成および管理できます。
- データ移行: Microsoft SQL Server、Sybase ASE、SQLite、SQL Anywhere、PostgreSQL、およびその他の RDBMS のテーブル、オブジェクト、およびデータを MySQL に移行できます。移行では、以前のバージョンの MySQL から最新のリリースへの移行もサポートされます。
- MySQL エンタープライズサポート: MySQL Enterprise Backup および MySQL Audit などのエンタープライズ製品をサポートします。

MySQL Workbench は Community Edition と Commercial Edition の 2 つのエディションで使用できます。Community Edition は無料で使用できます。Commercial Edition では、データベースドキュメントの生成などの追加のエンタープライズ機能が低価格で提供されています。

第 32 章 OCI マーケットプレイス上の MySQL

目次

32.1 Oracle Cloud Infrastructure に MySQL EE をデプロイするための前提条件	4535
32.2 Oracle Cloud Infrastructure での MySQL EE のデプロイ	4535
32.3 ネットワークアクセスの構成	4537
32.4 Connecting	4537
32.5 Maintenance	4538

この章では、MySQL Enterprise Edition を Oracle Cloud Infrastructure (OCI) マーケットプレイスアプリケーションとしてデプロイする方法について説明します。これは BYOL 製品です。

注記

OCI マーケットプレイスの詳細は、「[Marketplace の概要](#)」を参照してください。

MySQL Enterprise Edition Marketplace アプリケーションは、Oracle Linux 7.7 と MySQL EE 8.0 を実行する OCI コンピュートインスタンスです。デプロイされたイメージへの MySQL EE インストールは、[セクション 2.5.4 「Oracle の RPM パッケージを使用した Linux への MySQL のインストール](#)」で説明されている RPM インストールと似ています。

MySQL Enterprise Edition の詳細は、[第 30 章 「MySQL Enterprise Edition](#)」を参照してください。

MySQL の拡張構成の詳細は、「[セキュアデプロイメントガイド](#)」を参照してください。

Oracle Linux 7 の詳細は、「[Oracle Linux ドキュメント](#)」を参照してください

この製品はユーザー管理であり、アップグレードおよびメンテナンスを担当します。

32.1 Oracle Cloud Infrastructure に MySQL EE をデプロイするための前提条件

次の前提があります:

- OCI 用語に精通していること。OCI を初めて使用する場合は、「[はじめに](#)」を参照してください。
- 適切に構成された仮想クラウドネットワーク (VCN) およびサブネットにアクセスできます。詳細は、「[仮想ネットワーク](#)」を参照してください。
- OCI マーケットプレイスアプリケーションをテナンシのコンパートメントにデプロイするために必要な権限があります。詳細は、「[ポリシーの仕組み](#)」を参照してください。

32.2 Oracle Cloud Infrastructure での MySQL EE のデプロイ

Oracle Cloud Infrastructure に MySQL EE をデプロイするには、次の手順を実行します:

1. OCI マーケットプレイスを開き、MySQL を選択します。

MySQL リストが表示されます。

2. 「インスタンスの起動」をクリックして、アプリケーションの起動プロセスを開始します。

「コンピューティングインスタンスの作成」ダイアログが表示されます。

フィールドへの入力方法の詳細は、「[Linux インスタンスを作成するには](#)」を参照してください。

デフォルトでは、MySQL サーバーはポート 3306 でリスニングし、単一のユーザー root で構成されます。

重要

デプロイメントが完了すると、MySQL サーバーが起動し、コンピュートインスタンスに接続して、MySQL ログファイルに書き込まれたデフォルトのルートパスワードを取得する必要があります。

詳しくは[SSH での接続](#)をご覧ください。

次の MySQL ソフトウェアがインストールされています:

- MySQL Server EE
- MySQL Enterprise Backup
- MySQL Shell
- MySQL Router

MySQL 構成

セキュリティのために、次のものが有効になります:

- SELinux: 詳細は、「[SELinux の構成および使用](#)」を参照してください
- firewalld: 詳細は、[firewalld の現在のステータスと設定](#)を参照してください

次の MySQL プラグインが有効になります:

- [thread_pool](#)
- [validate_password](#)

起動時には、次の処理が行われます:

- MySQL Server は、`/etc/my.cnf.d/`で`etc/my.cnf` および `*.cnf` という名前のすべてのファイルを読み取ります。
- `/etc/my.cnf.d/perf-tuning.cnf` は、選択した OCI シェイプに基づいて`/usr/bin/mkcnf` によって作成されます。

注記

このメカニズムを無効にするには、`/etc/systemd/system/mysqld.service.d/perf-tuning.conf` を削除します。

パフォーマンスチューニングは、次のシェイプに対して構成されます:

- VM.Standard2.1
- VM.Standard2.2
- VM.Standard2.4
- VM.Standard2.8
- VM.Standard2.16
- VM.Standard2.24
- VM.Standard.E2.1
- VM.Standard.E2.2
- VM.Standard.E2.4
- VM.Standard.E2.8

- BM.Standard2.52

その他のすべてのシェイプでは、VM.Standard 2.1 のチューニングが使用されます。

32.3 ネットワークアクセスの構成

OCI セキュリティルールの詳細は、「[セキュリティルール](#)」を参照してください。

重要

MySQL X プロトコルを使用する場合は、ポート 22 (SSH) および 3306 (MySQL) およびオプションで 33060 でインGRES を有効にする必要があります。

32.4 Connecting

このセクションでは、OCI コンピュートインスタンスにデプロイされた MySQL サーバーに接続するための様々な接続方法について説明します。

SSH での接続

このセクションでは、UNIX 系プラットフォームから OCI Compute への接続について詳しく説明します。SSH での接続の詳細は、「[SSH を使用した Oracle Linux インスタンスへのアクセス](#)」および「[インスタンスへの接続](#)」を参照してください。

SSH を使用してコンピュートインスタンスで実行されている Oracle Linux に接続するには、次のコマンドを実行します:

```
ssh opc@computeIP
```

ここで、`opc` はコンピュートユーザーで、`computeIP` はコンピュートインスタンスの IP アドレスです。

root ユーザー用に作成された一時 root パスワードを検索するには、次のコマンドを実行します:

```
sudo grep 'temporary password' /var/log/mysqlid.log
```

デフォルトのパスワードを変更するには、次のコマンドを使用して、生成された一時パスワードを使用してサーバーにログイン: `mysql -uroot -p`。次に、次のコマンドを実行します:

```
ALTER USER 'root'@'localhost' IDENTIFIED BY 'MyNewPass4!';
```

MySQL クライアントとの接続

注記

ローカル MySQL クライアントから接続するには、まず、リモートログインを許可するユーザーを MySQL サーバーに作成する必要があります。

ローカル MySQL クライアントから MySQL Server に接続するには、シェルセッションから次のコマンドを実行します:

```
mysql -uroot -p -hcomputeIP
```

ここで、`computeIP` はコンピュートインスタンスの IP アドレスです。

MySQL Shell を使用した接続

ローカルの MySQL Shell から MySQL Server に接続するには、次のコマンドを実行してシェルセッションを開始します:

```
mysqlsh \connect root@computeIP
```

ここで、[computelIP](#) はコンピュータインスタンスの IP アドレスです。

MySQL Shell 接続の詳細は、[MySQL Shell 接続](#) を参照してください。

ワークベンチとの接続

MySQL Workbench から MySQL Server に接続するには、[Connections in MySQL Workbench](#) を参照してください。

32.5 Maintenance

この製品はユーザー管理であり、アップグレードおよびメンテナンスを担当します。

MySQL のアップグレード

既存のインストールは RPM ベースです。MySQL サーバーをアップグレードするには、[セクション2.11.6「Unix/Linux での MySQL バイナリまたはパッケージベースのインストールのアップグレード」](#) を参照してください。

`scp` を使用して、必要な RPM を OCI コンピュートインスタンスにコピーするか、OCI Object Storage からコピーできます (アクセス権が構成されている場合)。ファイルストレージもオプションです。詳細は、「[ファイルストレージおよび NFS](#)」を参照してください。

バックアップとリストア

MySQL Enterprise Backup は、推奨されるバックアップおよびリストアソリューションです。詳細は、[Backing Up to Cloud Storage](#) を参照してください。

MySQL Enterprise Backup の詳細は、[Getting Started with MySQL Enterprise Backup](#) を参照してください。

MySQL のデフォルトのバックアップおよびリストアの詳細は、[第7章「バックアップとリカバリ」](#) を参照してください。

付録 A MySQL 8.0 のよくある質問

目次

A.1 MySQL 8.0 FAQ: 全般	4539
A.2 MySQL 8.0 FAQ: ストレージエンジン	4541
A.3 MySQL 8.0 FAQ: サーバー SQL モード	4541
A.4 MySQL 8.0 FAQ: ストアドプロシージャおよびストアドファンクション	4542
A.5 MySQL 8.0 FAQ: トリガー	4546
A.6 MySQL 8.0 FAQ: ビュー	4548
A.7 MySQL 8.0 FAQ: INFORMATION_SCHEMA	4549
A.8 MySQL 8.0 FAQ: 移行	4549
A.9 MySQL 8.0 FAQ: セキュリティー	4550
A.10 MySQL 8.0 FAQ: NDB Cluster	4551
A.11 MySQL 8.0 FAQ: MySQL の中国語、日本語、および韓国語の文字セット	4563
A.12 MySQL 8.0 FAQ: コネクタおよび API	4573
A.13 MySQL 8.0 FAQ : C API、libmysql	4573
A.14 MySQL 8.0 FAQ: レプリケーション	4574
A.15 MySQL 8.0 FAQ: MySQL Enterprise Thread Pool	4577
A.16 MySQL 8.0 FAQ : InnoDB 変更バッファ	4578
A.17 MySQL 8.0 FAQ : InnoDB 保存データ暗号化	4580
A.18 MySQL 8.0 FAQ : 仮想化のサポート	4582

A.1 MySQL 8.0 FAQ: 全般

A.1.1 本番で使用 (GA) できる MySQL はどのバージョンですか。	4539
A.1.2 MySQL のバージョン番号付けでバージョン 6 および 7 がスキップされ、8.0 に直接移動したのはなぜですか。	4540
A.1.3 MySQL 8.0 ではサブクエリーを実行できますか。	4540
A.1.4 MySQL 8.0 では複数テーブルへの挿入、更新、および削除を実行できますか。	4540
A.1.5 MySQL 8.0 にはシーケンスはありますか。	4540
A.1.6 MySQL 8.0 には秒の小数部が返される NOW() 関数はありますか。	4540
A.1.7 MySQL 8.0 はマルチコアプロセッサで動作しますか。	4540
A.1.8 <code>mysqld</code> の複数のプロセスが表示されるのはなぜですか。	4540
A.1.9 MySQL 8.0 は ACID トランザクションを実行できますか。	4540

A.1.1. 本番で使用 (GA) できる MySQL はどのバージョンですか。

MySQL 8.0、5.7 および MySQL 5.6 は、本番用にサポートされています。

MySQL 8.0 は、2018 年 4 月 19 日に本番使用のためにリリースされた MySQL 8.0.11 で General Availability (GA) ステータスを達成しました。

MySQL 5.7 は MySQL 5.7.9 で一般提供 (GA) ステータスとなり、2015 年 10 月 21 日に本番使用のためにリリースされました。

MySQL 5.6 は MySQL 5.6.10 で一般提供 (GA) ステータスとなり、2013 年 2 月 5 日に本番使用のためにリリースされました。

MySQL 5.5 は MySQL 5.5.8 で一般提供 (GA) ステータスとなり、2010 年 12 月 3 日に本番使用のためにリリースされました。MySQL 5.5 のアクティブな開発が終了しました。

MySQL 5.1 は MySQL 5.1.30 で一般提供 (GA) ステータスとなり、2008 年 11 月 14 日に本番使用のためにリリースされました。MySQL 5.1 のアクティブな開発が終了しました。

MySQL 5.0 は MySQL 5.0.15 で一般提供 (GA) ステータスとなり、2005 年 10 月 19 日に本番使用のためにリリースされました。MySQL 5.0 のアクティブな開発が終了しました。

A.1.2. MySQL のバージョン番号付けでバージョン 6 および 7 がスキップされ、8.0 に直接移動したのはなぜですか。

この MySQL バージョンで導入された多くの新機能と重要な機能のため、新しいシリーズを開始することになりました。シリーズ番号 6 および 7 は MySQL によって実際に使用されていたため、8.0 に移動しました。

A.1.3. MySQL 8.0 ではサブクエリーを実行できますか。

はい。 [セクション13.2.11「サブクエリー」](#) を参照してください。

A.1.4. MySQL 8.0 では複数テーブルへの挿入、更新、および削除を実行できますか。

はい。複数テーブルへの更新の実行に必要な構文については、 [セクション13.2.13「UPDATE ステートメント」](#) を参照してください。複数テーブルでの削除の実行に必要な構文については、 [セクション13.2.2「DELETE ステートメント」](#) を参照してください。

複数テーブルへの挿入は、 [FOR EACH ROW](#) 句の [BEGIN ... END](#) ブロック内に複数の [INSERT](#) ステートメントが含まれているトリガーを使用して実行できます。 [セクション25.3「トリガーの使用」](#) を参照してください。

A.1.5. MySQL 8.0 にはシーケンスはありますか。

いいえ。ただし、MySQL には [AUTO_INCREMENT](#) システムがあり、MySQL 8.0 でマルチソースレプリケーション設定の挿入を処理することもできます。 [auto_increment_increment](#) および [auto_increment_offset](#) システム変数を使用して、各サーバーがほかのサーバーと競合しない自動インクリメント値を生成するように設定できます。 [auto_increment_increment](#) にはサーバーの数より大きい値を指定し、各サーバーが一意的なオフセットを持つようにします。

A.1.6. MySQL 8.0 には秒の小数部が返される [NOW\(\)](#) 関数はありますか。

はい。 [セクション11.2.6「時間値での小数秒」](#) を参照してください。

A.1.7. MySQL 8.0 はマルチコアプロセッサで動作しますか。

はい。MySQL はマルチスレッド化されており、使用可能なすべての CPU を使用します。すべての CPU を使用できるわけではありません。最新のオペレーティングシステムでは、基礎となるすべての CPU を使用できる必要がありますが、プロセスを特定の CPU または CPU のセットに制限することもできます。

Windows では、現在、 [mysqld](#) が使用できる (論理) プロセッサの数に制限があります: シングルプロセッサグループ。最大 64 個の論理プロセッサに制限されます。

複数のコアの使用方法は次のとおりです:

- 単一のコアは通常、1 つのセッションから発行されたコマンドを処理するために使用されます。
- いくつかのバックグラウンドスレッドでは、バックグラウンド I/O タスクの移動を維持するためなど、余分なコアの使用が制限されています。
- データベースが I/O-bound (容量未満の CPU 消費で示される) である場合、CPU の追加は複雑になります。データベースが I/O-bound 部分と CPU ボンド部分にパーティション化されている場合でも、CPU の追加が役立つことがあります。

A.1.8. [mysqld](#) の複数のプロセスが表示されるのはなぜですか。

[mysqld](#) は、マルチプロセスプログラムではなく単一プロセスプログラムであり、他のプロセスをフォークまたは起動しません。ただし、[mysqld](#) はマルチスレッド化されており、一部のプロセスレポートシステムユーティリティでは、マルチスレッドプロセスのスレッドごとに個別のエントリが表示されるため、実際には複数の [mysqld](#) プロセスが表示される場合があります。

A.1.9. MySQL 8.0 は ACID トランザクションを実行できますか。

はい。MySQL の現在のすべてのバージョンでトランザクションがサポートされます。 [InnoDB](#) ストレージエンジンは、行レベルのロック、マルチバージョン、非ロックの反復可能読み取り、および 4 つのすべての SQL 標準分離レベルを持つ、完全な ACID トランザクションを提供しています。

[NDB](#) ストレージエンジンは、 [READ COMMITTED](#) トランザクション分離レベルのみをサポートしています。

A.2 MySQL 8.0 FAQ: ストレージエンジン

A.2.1 MySQL のストレージエンジンの完全なドキュメントはどこで入手できますか。	4541
A.2.2 MySQL 8.0 に新しいストレージエンジンはありますか。	4541
A.2.3 MySQL 8.0 で削除されたストレージエンジンはありますか。	4541
A.2.4 特定のストレージエンジンの使用を防止できますか。	4541
A.2.5 InnoDB ストレージエンジンと InnoDB 以外のストレージエンジンの組み合わせとは対照的に、InnoDB ストレージエンジンを排他的に使用する利点がありますか。	4541
A.2.6 ARCHIVE ストレージエンジンに固有の利点は何ですか。	4541

A.2.1. MySQL のストレージエンジンの完全なドキュメントはどこで入手できますか。

第16章「代替ストレージエンジン」を参照してください。この章では、InnoDB ストレージエンジンと NDB ストレージエンジン (MySQL Cluster に使用) を除く、すべての MySQL ストレージエンジンについて説明します。InnoDB については、第15章「InnoDB ストレージエンジン」を参照してください。NDB については、第23章「MySQL NDB Cluster 8.0」を参照してください。

A.2.2. MySQL 8.0 に新しいストレージエンジンはありますか。

いいえ。InnoDB は新しいテーブルのデフォルトのストレージエンジンです。詳細は、セクション 15.1「InnoDB 入門」を参照してください。

A.2.3. MySQL 8.0 で削除されたストレージエンジンはありますか。

パーティショニングサポートを提供した PARTITION ストレージエンジンプラグインは、ネイティブのパーティショニングハンドラに置き換えられます。この変更の一環として、サーバーは - DWITH_PARTITION_STORAGE_ENGINE を使用して構築できなくなりました。partition は、SHOW PLUGINS の出力または INFORMATION_SCHEMA.PLUGINS テーブルにも表示されなくなりました。

特定のテーブルのパーティション化をサポートするには、テーブルに使用されるストレージエンジンが独自の (「native」) パーティショニングハンドラを提供する必要があります。InnoDB は、ネイティブのパーティショニングハンドラを含む MySQL 8.0 でサポートされている唯一のストレージエンジンです。ほかのストレージエンジンを使用して MySQL 8.0 でパーティション化されたテーブルを作成しようとすると、失敗します。(MySQL Cluster で使用される NDB ストレージエンジンも独自のパーティショニングハンドラを提供しますが、MySQL 8.0 では現在サポートされていません。)

A.2.4. 特定のストレージエンジンの使用を防止できますか。

はい。disabled_storage_engines 構成オプションは、テーブルまたはテーブルスペースの作成に使用できないストレージエンジンを定義します。デフォルトでは、disabled_storage_engines は空 (エンジンが無効になっていない) ですが、1 つ以上のエンジンのカンマ区切りリストに設定できます。

A.2.5. InnoDB ストレージエンジンと InnoDB 以外のストレージエンジンの組み合わせとは対照的に、InnoDB ストレージエンジンを排他的に使用する利点がありますか。

はい。InnoDB テーブルを排他的に使用すると、バックアップおよびリカバリ操作を簡略化できます。MySQL Enterprise Backup は、InnoDB ストレージエンジンを使用するすべてのテーブルの hot backup を実行します。MyISAM またはその他の InnoDB 以外のストレージエンジンを使用するテーブルの場合は、「ウォーム」バックアップが実行され、データベースは引き続き実行されますが、バックアップ中にこれらのテーブルを変更することはできません。セクション 30.2「MySQL Enterprise Backup の概要」を参照してください。

A.2.6. ARCHIVE ストレージエンジンに固有の利点は何ですか。

ARCHIVE ストレージエンジンは、インデックスなしで大量のデータを格納します。フットプリントは小さく、テーブルスキャンを使用して選択を実行します。詳細は、セクション 16.5「ARCHIVE ストレージエンジン」を参照してください。

A.3 MySQL 8.0 FAQ: サーバー SQL モード

A.3.1 サーバー SQL モードとは何ですか。	4542
A.3.2 サーバー SQL モードはいくつありますか。	4542
A.3.3 サーバー SQL モードを判別するにはどうすればよいですか。	4542
A.3.4 モードはデータベースまたは接続に依存していますか。	4542

A.3.5 厳密モードにルールを追加できますか。	4542
A.3.6 厳密モードはパフォーマンスに影響しますか。	4542
A.3.7 MySQL 8.0 をインストールしたときのデフォルトのサーバー SQL モードは何ですか。	4542

A.3.1. サーバー SQL モードとは何ですか。

サーバー SQL モードは、MySQL でサポートされる SQL 構文、および実行されるデータ妥当性チェックの種類を定義します。これにより、MySQL をさまざまな環境で使用したり、MySQL をほかのデータベースサーバーと一緒に使用したりすることが、さらに容易になります。MySQL サーバーは、これらのモードを各クライアントに個別に適用します。詳細は、[セクション5.1.11「サーバー SQL モード」](#)を参照してください。

A.3.2. サーバー SQL モードはいくつありますか。

各モードは、個別にオン/オフを切り替えることができます。使用可能なモードの完全なリストについては、[セクション5.1.11「サーバー SQL モード」](#)を参照してください。

A.3.3. サーバー SQL モードを判別するにはどうすればよいですか。

`--sql-mode` オプションを使用すると、デフォルトの SQL モード (`mysqld` を起動する場合) を設定できます。`SET [GLOBAL|SESSION] sql_mode='modes'` ステートメントを使用すると、ローカルに接続に対して、またはグローバルに適用されるように、接続内から設定を変更できます。現在のモードを取得するには、`SELECT @@sql_mode` ステートメントを発行します。

A.3.4. モードはデータベースまたは接続に依存していますか。

モードは特定のデータベースにリンクされていません。モードは、ローカルにセッション (接続) に対して設定するか、グローバルにサーバーに対して設定できます。これらの設定は、`SET [GLOBAL|SESSION] sql_mode='modes'` を使用して変更できます。

A.3.5. 厳密モードにルールを追加できますか。

厳密モードと呼ぶ場合は、`TRADITIONAL`、`STRICT_TRANS_TABLES`、または `STRICT_ALL_TABLES` モードの少なくとも 1 つが有効にされているモードを意味します。オプションは組み合わせることができるため、モードに制約を追加できます。詳細は、[セクション5.1.11「サーバー SQL モード」](#)を参照してください。

A.3.6. 厳密モードはパフォーマンスに影響しますか。

一部の設定での入力データの厳密な検証では、検証を行わない場合より時間がかかります。パフォーマンスへの影響はそれほど大きくありませんが、そのような検証が必要ない場合 (アプリケーションでそのすべてをすでに処理している場合)、MySQL には厳密モードを無効にするオプションがあります。ただし、必要な場合は、厳密モードでこのような検証を行うことができます。

A.3.7. MySQL 8.0 をインストールしたときのデフォルトのサーバー SQL モードは何ですか。

MySQL 8.0 のデフォルトの SQL モードには、次のモードが含まれます: `ONLY_FULL_GROUP_BY`、`STRICT_TRANS_TABLES`、`NO_ZERO_IN_DATE`、`NO_ZERO_DATE`、`ERROR_FOR_DIVISION_BY_ZERO` および `NO_ENGINE_SUBSTITUTION`。

使用可能なすべてのモードおよびデフォルトの MySQL 動作の詳細は、[セクション5.1.11「サーバー SQL モード」](#)を参照してください。

A.4 MySQL 8.0 FAQ: ストアドプロシージャおよびストアドファンクション

A.4.1 MySQL 8.0 はストアドプロシージャおよびストアドファンクションをサポートしていますか。	4543
A.4.2 MySQL のストアドプロシージャおよびストアドファンクションについてのドキュメントはどこにありますか。	4543
A.4.3 MySQL のストアドプロシージャのディスカッションフォーラムはありますか。	4543
A.4.4 ストアドプロシージャの ANSI SQL 2003 仕様はどこにありますか。	4543
A.4.5 ストアドルーチンを管理するにはどうすればよいですか。	4543
A.4.6 特定のデータベースのすべてのストアドプロシージャおよびストアドファンクションを表示する方法はありますか。	4543
A.4.7 ストアドプロシージャはどこに格納されますか。	4544

- A.4.8 ストアドプロシージャーまたはストアドファンクションをパッケージにグループ化することはできませんか。 4544
- A.4.9 ストアドプロシージャーは別のストアドプロシージャーを呼び出すことができますか。 4544
- A.4.10 ストアドプロシージャーはトリガーを呼び出すことができますか。 4544
- A.4.11 ストアドプロシージャーはテーブルにアクセスできますか。 4544
- A.4.12 ストアドプロシージャーには、アプリケーションエラーを発生させるステートメントはありますか。 4544
- A.4.13 ストアドプロシージャーには例外処理はありますか。 4544
- A.4.14 MySQL 8.0 のストアドルーチンは結果セットを返すことができますか。 4544
- A.4.15 ストアドプロシージャーで `WITH RECOMPILE` はサポートされますか。 4544
- A.4.16 `mod_plsql` を Apache のゲートウェイとして使用してデータベース内のストアドプロシージャーと直接やり取りするのと同様の機能は MySQL にありますか。 4544
- A.4.17 ストアドプロシージャーに入力として配列を渡すことはできますか。 4544
- A.4.18 ストアドプロシージャーに `IN` パラメータとしてカーソルを渡すことはできますか。 4545
- A.4.19 ストアドプロシージャーの `OUT` パラメータとしてカーソルを返すことはできますか。 4545
- A.4.20 デバッグのために、ストアドルーチン内の変数の値を出力できますか。 4545
- A.4.21 ストアドプロシージャー内でトランザクションをコミットまたはロールバックできますか。 4545
- A.4.22 MySQL 8.0 のストアドプロシージャーおよびストアドファンクションはレプリケーションで動作しますか。 4545
- A.4.23 レプリケーションソースサーバー上に作成されたストアドプロシージャーおよびストアドファンクションはレプリカにレプリケートされますか。 4545
- A.4.24 ストアドプロシージャーおよびストアドファンクション内で実行されたアクションはどのようにレプリケートされますか。 4545
- A.4.25 レプリケーションでストアドプロシージャーおよびストアドファンクションを使用するための特別なセキュリティ要件はありますか。 4545
- A.4.26 ストアドプロシージャーおよびストアドファンクションのアクションをレプリケートする場合の制限は何ですか。 4545
- A.4.27 前述の制限は、MySQL が point-in-time 4546
- A.4.28 前述の制限を修正するために何が行われていますか。 4546
- A.4.1. MySQL 8.0 はストアドプロシージャーおよびストアドファンクションをサポートしていますか。
- はい。MySQL 8.0 は、ストアドプロシージャーとストアドファンクションの 2 種類のストアドルーチンをサポートしています。
- A.4.2. MySQL のストアドプロシージャーおよびストアドファンクションについてのドキュメントはどこにありますか。
- [セクション25.2「ストアドルーチンの使用」](#)を参照してください。
- A.4.3. MySQL のストアドプロシージャーのディスカッションフォーラムはありますか。
- はい。 <https://forums.mysql.com/list.php?98> を参照してください。
- A.4.4. ストアドプロシージャーの ANSI SQL 2003 仕様はどこにありますか。
- 残念ながら、正式な仕様は無料では入手できません (ANSI は有料で販売しています)。ただし、「SQL-99 完全、本当に」 by Peter Gulutzan や Trudy Pelzer など、ストアドプロシージャーの適用範囲を含む標準の包括的な概要を提供する書籍があります。
- A.4.5. ストアドルーチンを管理するにはどうすればよいですか。
- ストアドルーチンに明かな命名スキームを使用することはよい管理方法です。ストアドプロシージャーは、`CREATE [FUNCTION|PROCEDURE]`、`ALTER [FUNCTION|PROCEDURE]`、`DROP [FUNCTION|PROCEDURE]`、および `SHOW CREATE [FUNCTION|PROCEDURE]` を使用して管理できます。既存のストアドプロシージャーに関する情報を取得するには、`INFORMATION_SCHEMA` データベースの `ROUTINES` テーブル ([セクション26.30「INFORMATION_SCHEMA ROUTINES テーブル」](#)を参照してください) を使用します。
- A.4.6. 特定のデータベースのすべてのストアドプロシージャーおよびストアドファンクションを表示する方法はありますか。
- はい。 `dbname` という名前のデータベースの場合、`INFORMATION_SCHEMA.ROUTINES` テーブルに対して次のクエリーを使用します。

```
SELECT ROUTINE_TYPE, ROUTINE_NAME
FROM INFORMATION_SCHEMA.ROUTINES
WHERE ROUTINE_SCHEMA='dbname';
```

詳細は、[セクション26.30「INFORMATION_SCHEMA ROUTINES テーブル」](#)を参照してください。

ストアドルーチンの本体は、[SHOW CREATE FUNCTION](#) (ストアドファンクションの場合) または [SHOW CREATE PROCEDURE](#) (ストアドプロシージャの場合) を使用して表示できます。詳細は、[セクション13.7.7.9「SHOW CREATE PROCEDURE ステートメント」](#)を参照してください。

A.4.7. ストアドプロシージャはどこに格納されますか。

ストアドプロシージャは、データディクショナリの一部である `mysql.routines` テーブルおよび `mysql.parameters` テーブルに格納されます。これらのテーブルに直接アクセスすることはできません。かわりに、`INFORMATION_SCHEMA ROUTINES` テーブルおよび `PARAMETERS` テーブルをクエリーします。[セクション26.30「INFORMATION_SCHEMA ROUTINES テーブル」](#) および [セクション26.20「INFORMATION_SCHEMA PARAMETERS テーブル」](#)を参照してください。

[SHOW CREATE FUNCTION](#) を使用してストアドファンクションに関する情報を取得し、[SHOW CREATE PROCEDURE](#) を使用してストアドプロシージャに関する情報を取得することもできます。[セクション13.7.7.9「SHOW CREATE PROCEDURE ステートメント」](#)を参照してください。

A.4.8. ストアドプロシージャまたはストアドファンクションをパッケージにグループ化することはできますか。

いいえ。これは MySQL 8.0 ではサポートされません。

A.4.9. ストアドプロシージャは別のストアドプロシージャを呼び出すことができますか。

はい。

A.4.10. ストアドプロシージャはトリガーを呼び出すことができますか。

ストアドプロシージャでは、トリガーが実行される `UPDATE` などの SQL ステートメントを実行できます。

A.4.11. ストアドプロシージャはテーブルにアクセスできますか。

はい。ストアドプロシージャは、必要に応じて 1 つ以上のテーブルにアクセスできます。

A.4.12. ストアドプロシージャには、アプリケーションエラーを発生させるステートメントはありますか。

はい。MySQL 8.0 には、SQL 標準の `SIGNAL` ステートメントおよび `RESIGNAL` ステートメントが実装されています。[セクション13.6.7「条件の処理」](#)を参照してください。

A.4.13. ストアドプロシージャには例外処理はありますか。

MySQL には、SQL 標準に従った `HANDLER` 定義が実装されています。詳細は、[セクション13.6.7.2「DECLARE ... HANDLER ステートメント」](#)を参照してください。

A.4.14. MySQL 8.0 のストアドルーチンは結果セットを返すことができますか。

ストアドプロシージャは返すことができますが、ストアドファンクションは返すことができません。ストアドプロシージャ内で通常の `SELECT` を実行すると、結果セットがクライアントに直接返されます。これを機能させるには、MySQL 4.1 以上のクライアント/サーバプロトコルを使用する必要があります。つまり、PHP では、古い `mysql` 拡張機能ではなく `mysqli` 拡張機能を使用する必要があります。

A.4.15. ストアドプロシージャで `WITH RECOMPILE` はサポートされますか。

MySQL 8.0 にはありません。

A.4.16. `mod_plsql` を Apache のゲートウェイとして使用してデータベース内のストアドプロシージャと直接やり取りするのと同等の機能は MySQL にありますか。

MySQL 8.0 には同等の機能はありません。

A.4.17. ストアドプロシージャに入力として配列を渡すことはできますか。

MySQL 8.0 にはありません。

A.4.18. ストアドプロシージャに `IN` パラメータとしてカーソルを渡すことはできますか。

MySQL 8.0 では、カーソルはストアドプロシージャの内部でのみ使用できます。

A.4.19. ストアドプロシージャの `OUT` パラメータとしてカーソルを返すことはできますか。

MySQL 8.0 では、カーソルはストアドプロシージャの内部でのみ使用できます。ただし、`SELECT` でカーソルをオープンしない場合、結果はクライアントに直接送信されます。変数に対して `SELECT INTO` を発行することもできます。セクション13.2.10「`SELECT` ステートメント」を参照してください。

A.4.20. デバッグのために、ストアドルーチン内の変数の値を出力できますか。

はい。これは、ストアドプロシージャでは行うことができますが、ストアドファンクションでは行うことはできません。ストアドプロシージャ内で通常の `SELECT` を実行すると、結果セットがクライアントに直接返されます。これを機能させるには、MySQL 4.1 (以上) のクライアント/サーバープロトコルを使用する必要があります。つまり、PHP では、古い `mysql` 拡張機能ではなく `mysqli` 拡張機能を使用する必要があります。

A.4.21. ストアドプロシージャ内でトランザクションをコミットまたはロールバックできますか。

はい。ただし、ストアドファンクション内でトランザクション操作を実行することはできません。

A.4.22. MySQL 8.0 のストアドプロシージャおよびストアドファンクションはレプリケーションで動作しますか。

はい。ストアドプロシージャおよびファンクションで実行される標準アクションは、レプリケーションソースサーバーからレプリカにレプリケートされます。セクション25.7「ストアドプログラムバイナリロギング」で詳しく説明されているいくつかの制限があります。

A.4.23. レプリケーションソースサーバー上に作成されたストアドプロシージャおよびストアドファンクションはレプリカにレプリケートされますか。

はい。オブジェクトが両方のサーバーに存在するように、レプリケーションソースサーバーで通常の DDL ステートメントを介して実行されるストアドプロシージャおよびファンクションの作成はレプリカにレプリケートされます。ストアドプロシージャおよびストアドファンクションに対する `ALTER` ステートメントおよび `DROP` ステートメントもレプリケートされます。

A.4.24. ストアドプロシージャおよびストアドファンクション内で実行されたアクションはどのようにレプリケートされますか。

MySQL は、ストアドプロシージャで発生した各 DML イベントを記録し、それらの個々のアクションをレプリカにレプリケートします。ストアドプロシージャを実行するために行われた実際の呼び出しはレプリケートされません。

データを変更するストアドファンクションは、各ファンクション内で行われた DML イベントとしてではなく、関数呼び出しとしてログ記録されます。

A.4.25. レプリケーションでストアドプロシージャおよびストアドファンクションを使用するための特別なセキュリティ要件はありますか。

はい。レプリカにはソースバイナリログから読み取られたステートメントを実行する権限があるため、レプリケーションでストアドファンクションを使用するための特別なセキュリティ制約が存在します。レプリケーションまたは一般のバイナリロギング (ポイントインタイムリカバリのための) がアクティブである場合、MySQL の DBA には選択できるセキュリティオプションが 2 つあります。

1. ストアドファンクションを作成するユーザーに、`SUPER` 権限を付与する必要があります。
2. または、DBA は `log_bin_trust_function_creators` システム変数に 1 を設定できます。これにより、標準の `CREATE ROUTINE` 権限を持ったユーザーがストアドファンクションを作成できます。

A.4.26. ストアドプロシージャおよびストアドファンクションのアクションをレプリケートする場合の制限は何ですか。

ストアドプロシージャに埋め込まれている決定性のない (ランダムな) アクションまたは時間ベースのアクションは、正しくレプリケートされないことがあります。その性質上、ランダムに生成された結果は予測できず、正確には再現できません。したがって、レプリカにレプリケートされたランダムなアクションは、ソースで実行されたものを反映しません。ストアドファンクションを `DETERMINISTIC` として宣言する

か、[log_bin_trust_function_creators](#) システム変数を 0 に設定すると、ランダムな操作によってランダムな値が生成されなくなります。

また、ストアードプロシージャでのアクションのタイミングはレプリケーションに使用されるバイナリログを介して再現できないため、レプリカでは時間ベースのアクションを再現できません。バイナリログには、DML イベントのみが記録され、タイミング制約は含まれていません。

最後に、大規模な DML アクション (一括挿入など) 中にエラーが発生した非トランザクションテーブルでは、ソースが DML アクティビティから部分的に更新される可能性があるというレプリケーションの問題が発生する可能性があります。エラーが発生したためレプリカは更新されません。回避策として、[IGNORE](#) キーワードを使用して関数 DML アクションを実行すると、エラーの原因となったソースの更新が無視され、エラーの原因とならない更新がレプリカにレプリケートされます。

A.4.27 前述の制限は、MySQL が point-in-time

レプリケーションに影響する制限が、ポイントインタイムリカバリに同様に影響します。

A.4.28 前述の制限を修正するために何が行われていますか。

ステートメントベースのレプリケーションまたは行ベースのレプリケーションのいずれかを選択できます。元のレプリケーションの実装は、ステートメントベースのバイナリロギングに基づいています。行ベースのバイナリロギングによって、前述の制限が解決されます。

複合レプリケーションも使用できます (`--binlog-format=mixed` を指定してサーバーを起動します)。このハイブリッド形式のレプリケーション「[knows](#)」ステートメントレベルレプリケーションを安全に使用できるかどうか、または行レベルレプリケーションが必要かどうか。

追加情報については、[セクション 17.2.1 「レプリケーション形式」](#) を参照してください。

A.5 MySQL 8.0 FAQ: トリガー

A.5.1 MySQL 8.0 のトリガーについてのドキュメントはどこにありますか。	4546
A.5.2 MySQL のトリガーについてのディスカッションフォーラムはありますか。	4546
A.5.3 MySQL 8.0 にはステートメントレベルまたは行レベルのトリガーはありますか。	4546
A.5.4 デフォルトのトリガーはありますか。	4546
A.5.5 MySQL でトリガーを管理するにはどうすればよいですか。	4547
A.5.6 特定のデータベースのすべてのトリガーを表示する方法はありますか。	4547
A.5.7 トリガーはどこに格納されますか。	4547
A.5.8 トリガーはストアードプロシージャを呼び出すことができますか。	4547
A.5.9 トリガーはテーブルにアクセスできますか。	4547
A.5.10 1 つのテーブルに同じトリガーイベントおよびアクション時間のトリガーを複数定義することはできますか。	4547
A.5.11 トリガーはリモートサーバー上のテーブルを更新できますか。	4547
A.5.12 トリガーはレプリケーションで動作しますか。	4547
A.5.13 ソースのトリガーを介して実行されるアクションはレプリカにどのようにレプリケートされますか。	4548

A.5.1. MySQL 8.0 のトリガーについてのドキュメントはどこにありますか。

[セクション 25.3 「トリガーの使用」](#) を参照してください。

A.5.2. MySQL のトリガーについてのディスカッションフォーラムはありますか。

はい。 <https://forums.mysql.com/list.php?99> にあります。

A.5.3. MySQL 8.0 にはステートメントレベルまたは行レベルのトリガーはありますか。

MySQL 8.0 では、すべてのトリガーは [FOR EACH ROW](#) です。つまり、トリガーは、挿入、更新または削除される各行に対してアクティブ化されます。MySQL 8.0 は [FOR EACH STATEMENT](#) を使用したトリガーをサポートしていません。

A.5.4. デフォルトのトリガーはありますか。

明示的にはありません。MySQL は、一部の [TIMESTAMP](#) カラム、および [AUTO_INCREMENT](#) を使用して定義されたカラムに特殊な動作を割り当てています。

A.5.5. MySQL でトリガーを管理するにはどうすればよいですか。

MySQL 8.0 では、トリガーを作成する場合は `CREATE TRIGGER` ステートメントを使用し、削除する場合は `DROP TRIGGER` を使用します。これらのステートメントについては、[セクション13.1.22「CREATE TRIGGER ステートメント」](#) および [セクション13.1.34「DROP TRIGGER ステートメント」](#) を参照してください。

トリガーに関する情報は、`INFORMATION_SCHEMA.TRIGGERS` テーブルをクエリーすることによって取得できます。[セクション26.45「INFORMATION_SCHEMA TRIGGERS テーブル」](#) を参照してください。

A.5.6. 特定のデータベースのすべてのトリガーを表示する方法はありますか。

はい。データベース `dbname` に定義されているすべてのトリガーのリストを取得するには、`INFORMATION_SCHEMA.TRIGGERS` テーブルに対して次のようなクエリーを使用します。

```
SELECT TRIGGER_NAME, EVENT_MANIPULATION, EVENT_OBJECT_TABLE, ACTION_STATEMENT
FROM INFORMATION_SCHEMA.TRIGGERS
WHERE TRIGGER_SCHEMA='dbname';
```

このテーブルについては、[セクション26.45「INFORMATION_SCHEMA TRIGGERS テーブル」](#) を参照してください。

MySQL に固有の `SHOW TRIGGERS` ステートメントを使用することもできます。[セクション13.7.7.40「SHOW TRIGGERS ステートメント」](#) を参照してください。

A.5.7. トリガーはどこに格納されますか。

トリガーは、データディクショナリの一部である `mysql.triggers` システムテーブルに格納されます。

A.5.8. トリガーはストアードプロシージャを呼び出すことができますか。

はい。

A.5.9. トリガーはテーブルにアクセスできますか。

トリガーは、それ自体が定義されているテーブルの古いデータおよび新しいデータの両方にアクセスできます。トリガーはほかのテーブルに影響を与えることもできますが、その関数またはトリガーを呼び出したステートメントによってすでに使用されている (読み取りまたは書き込みのために) テーブルを変更することはできません。

A.5.10.1 つのテーブルに同じトリガーイベントおよびアクション時間のトリガーを複数定義することはできますか。

MySQL 8.0 では、同じトリガーイベントおよびアクション時間を持つ特定のテーブルに対して複数のトリガーを定義できます。たとえば、1 つのテーブルに対して 2 つの `BEFORE UPDATE` トリガーを定義できます。デフォルトでは、同じトリガーイベントおよびアクション時間を持つトリガーは、作成された順序で実行されます。トリガーの順序を指定するには、`FOR EACH ROW` のあとに `FOLLOWS` または `PRECEDES` を示す句、および同じトリガーイベントとアクション時間を持つ既存のトリガーの名前を指定します。`FOLLOWS` を指定すると、新しいトリガーは既存のトリガーのあとに実行されます。`PRECEDES` を指定すると、新しいトリガーは既存のトリガーの前に実行されます。

A.5.11. トリガーはリモートサーバー上のテーブルを更新できますか。

はい。リモートサーバー上のテーブルは、`FEDERATED` ストレージエンジンを使用して更新できます。[\(セクション16.8「FEDERATED ストレージエンジン」](#) を参照してください)。

A.5.12. トリガーはレプリケーションで動作しますか。

はい。ただし、それらがどのように機能するかは、MySQL 「クラシック」ステートメントベースと行ベースのどちらのレプリケーション形式を使用しているかによって異なります。

ステートメントベースレプリケーションを使用する場合、レプリカ上のトリガーは、ソース上で実行される (およびレプリカにレプリケートされる) ステートメントによって実行されます。

行ベースのレプリケーションを使用している場合、ソースで実行されてレプリカにレプリケートされたステートメントのため、トリガーはレプリカで実行されません。かわりに、行ベースのレプリケーションを使用すると、ソースでトリガーを実行したことによって発生した変更がレプリカに適用されます。

詳細は、[セクション17.5.1.36「レプリケーションとトリガー」](#)を参照してください。

A.5.13. ソースのトリガーを介して実行されるアクションはレプリカにどのようにレプリケートされますか。

これは、ステートメントベースのレプリケーションまたは行ベースのレプリケーションのいずれを使用しているかによって異なります。

ステートメントベースのレプリケーション. まず、ソースに存在するトリガーをレプリカサーバーに再作成する必要があります。これが行われると、レプリケーションに関与するほかの標準の DML ステートメントと同様に、レプリケーションフローが動作します。たとえば、レプリケーションソースサーバーに存在する **AFTER** 挿入トリガーを持つテーブル **EMP** について考えてみます。レプリカサーバーにも同じ **EMP** テーブルおよび **AFTER** 挿入トリガーが存在します。レプリケーションフローは次のようになります。

1. **INSERT** ステートメントが **EMP** に発行されます。
2. **EMP** の **AFTER** トリガーが実行されます。
3. **INSERT** ステートメントがバイナリログに書き込まれます。
4. レプリカは、**INSERT** ステートメントを **EMP** に取得して実行します。
5. レプリカに存在する **EMP** の **AFTER** トリガーがアクティブ化されます。

行ベースのレプリケーション. 行ベースのレプリケーションを使用すると、ソースでトリガーを実行したことによって発生した変更がレプリカに適用されます。ただし、トリガー自体は、実際には行ベースレプリケーションの下でレプリカでは実行されません。これは、ソースとレプリカの両方がソースからの変更を適用し、さらにこれらの変更の原因となったトリガーがレプリカに適用された場合、変更がレプリカに 2 回適用され、ソースとレプリカのデータが異なるためです。

ほとんどの場合、行ベースのレプリケーションおよびステートメントベースのレプリケーションで結果は同じです。ただし、ソースとレプリカで異なるトリガーを使用する場合は、行ベースのレプリケーションを使用できません。(これは、行ベースの形式では、トリガーが実行される原因となったステートメントではなく、ソースで実行されているトリガーによって行われた変更がレプリカにレプリケートされ、レプリカ上の対応するトリガーが実行されないためです。) 代わりに、そのようなトリガーが実行されるきっかけとなったステートメントを、ステートメントベースのレプリケーションを使用してレプリケートする必要があります。

詳細は、[セクション17.5.1.36「レプリケーションとトリガー」](#)を参照してください。

A.6 MySQL 8.0 FAQ: ビュー

A.6.1 MySQL のビューについて説明しているドキュメントはどこにありますか。	4548
A.6.2 MySQL のビューについてのディスカッションフォーラムはありますか。	4548
A.6.3 基礎テーブルが削除または名前変更された場合、ビューはどうなりますか。	4548
A.6.4 MySQL 8.0 にはテーブルのスナップショットはありますか。	4548
A.6.5 MySQL 8.0 にはマテリアライズドビューはありますか。	4549
A.6.6 結合に基づいているビューに挿入できますか。	4549

A.6.1. MySQL のビューについて説明しているドキュメントはどこにありますか。

[セクション25.5「ビューの使用」](#)を参照してください。

A.6.2. MySQL のビューについてのディスカッションフォーラムはありますか。

はい。 <https://forums.mysql.com/list.php?100> を参照してください。

A.6.3. 基礎テーブルが削除または名前変更された場合、ビューはどうなりますか。

ビューが作成されたあとに、その定義が参照しているテーブルまたはビューを削除または変更できます。この種類の問題に関してビュー定義を確認するには、[CHECK TABLE](#) ステートメントを使用します。([セクション13.7.3.2「CHECK TABLE ステートメント」](#)を参照してください。)

A.6.4. MySQL 8.0 にはテーブルのスナップショットはありますか。

いいえ。

A.6.5. MySQL 8.0 にはマテリアライズドビューはありますか。

いいえ。

A.6.6. 結合に基づいているビューに挿入できますか。

`INSERT` ステートメントに、関連するテーブルが 1 つのみであることを明確にするカラムリストがある場合は、実行できます。

ビューに対する単一の挿入で、複数のテーブルに挿入することはできません。

A.7 MySQL 8.0 FAQ: INFORMATION_SCHEMA

A.7.1 MySQL の <code>INFORMATION_SCHEMA</code> データベースについてのドキュメントはどこにありますか。	4549
A.7.2 <code>INFORMATION_SCHEMA</code> についてのディスカッションフォーラムはありますか。	4549
A.7.3 <code>INFORMATION_SCHEMA</code> の ANSI SQL 2003 仕様はどこにありますか。	4549
A.7.4 Oracle Data Dictionary と MySQL <code>INFORMATION_SCHEMA</code> の違いは何ですか。	4549
A.7.5 <code>INFORMATION_SCHEMA</code> データベースのテーブルに挿入または変更を行うことはできますか。	4549

A.7.1. MySQL の `INFORMATION_SCHEMA` データベースについてのドキュメントはどこにありますか。

[第26章「INFORMATION_SCHEMA テーブル」](#)を参照してください。

A.7.2. `INFORMATION_SCHEMA` についてのディスカッションフォーラムはありますか。

<https://forums.mysql.com/list.php?101> を参照してください。

A.7.3. `INFORMATION_SCHEMA` の ANSI SQL 2003 仕様はどこにありますか。

残念ながら、正式な仕様は無料では入手できません。(ANSI は有料で販売しています。)ただし、「SQL-99 完全、本当に」 by Peter Gulutzan や Trudy Pelzer など、`INFORMATION_SCHEMA` を含む標準の包括的な概要を提供するブックがあります。

A.7.4. Oracle Data Dictionary と MySQL `INFORMATION_SCHEMA` の違いは何ですか。

Oracle および MySQL は両方とも、テーブルにメタデータを提供しています。ただし、Oracle と MySQL では異なるテーブル名およびカラム名が使用されています。MySQL の実装は、SQL 標準で定義されている `INFORMATION_SCHEMA` もサポートする DB2 および SQL Server の実装に似ています。

A.7.5. `INFORMATION_SCHEMA` データベースのテーブルに挿入または変更を行うことはできますか。

いいえ。アプリケーションが特定の標準構造に依存していることがあるため、変更しないでください。このため、`INFORMATION_SCHEMA` テーブルまたはデータを変更したことによるバグまたはその他の問題はサポートできません。

A.8 MySQL 8.0 FAQ: 移行

A.8.1 MySQL 5.7 から MySQL 8.0 に移行する方法に関する情報はどこにありますか。	4549
A.8.2 MySQL 8.0 のストレージエンジン (テーブルタイプ) のサポートは、以前のバージョンからどのように変更されましたか。	4549

A.8.1. MySQL 5.7 から MySQL 8.0 に移行する方法に関する情報はどこにありますか。

アップグレード情報については、[セクション2.11「MySQL のアップグレード」](#)を参照してください。アップグレード時はメジャーバージョンをスキップせずに、各手順でメジャーバージョンから次のメジャーバージョンにアップグレードして、手順の操作を完了してください。これはより複雑に見える可能性がありますが、最終的に時間とトラブルを節約します。アップグレード中に問題が発生した場合、その起点は、ユーザーまたは MySQL Enterprise サブスクリプションを保有している場合は MySQL サポートによって識別しやすくなります。

A.8.2. MySQL 8.0 のストレージエンジン (テーブルタイプ) のサポートは、以前のバージョンからどのように変更されましたか。

ストレージエンジンのサポートは次のように変更されました。

- **ISAM** テーブルのサポートは MySQL 5.0 で削除されたため、**ISAM** の代わりに **MyISAM** ストレージエンジンを使用してください。テーブル `tblname` を **ISAM** から **MyISAM** に変更するには、次のようなステートメントを発行します。

```
ALTER TABLE tblname ENGINE=MYISAM;
```

- **MyISAM** テーブルの内部 **RAID** も MySQL 5.0 で削除されました。これは、2G バイトを超えるファイルサイズをサポートしていないファイルシステムで大きいテーブルを許容するために以前使用されていました。現在のすべてのファイルシステムではより大きいテーブルが許容されます。**MERGE** テーブル、ビューなどの新しいほかのソリューションもあります。
- すべてのストレージエンジンの **VARCHAR** カラム型で、末尾のスペースが維持されるようになりました。
- **MEMORY** テーブル (以前は **HEAP** テーブルと呼ばれました) にも **VARCHAR** カラムを含めることができます。

A.9 MySQL 8.0 FAQ: セキュリティー

A.9.1 MySQL のセキュリティの問題に関するドキュメントはどこにありますか。	4550
A.9.2 MySQL 8.0 のデフォルトの認証プラグインは何ですか。	4550
A.9.3 MySQL 8.0 には SSL に対するネイティブなサポートはありますか。	4551
A.9.4 SSL サポートは MySQL バイナリに組み込まれていますか。または、バイナリを有効にするにはバイナリを自分で再コンパイルする必要がありますか。	4551
A.9.5 MySQL 8.0 には LDAP ディレクトリに対する組み込みの認証はありますか。	4551
A.9.6 MySQL 8.0 にはロールベースのアクセス制御 (RBAC) のサポートは含まれていますか。	4551

A.9.1. MySQL のセキュリティの問題に関するドキュメントはどこにありますか。

最初に第6章「[セキュリティ](#)」をお読みください。

特定のセキュリティの問題に関して役に立つことがある MySQL ドキュメントのほかの部分としては、次のセクションがあります。

- [セクション6.1.1「セキュリティガイドライン」](#)。
- [セクション6.1.3「攻撃者に対する MySQL のセキュアな状態の維持」](#)。
- [セクションB.3.3.2「root のパスワードをリセットする方法」](#)。
- [セクション6.1.5「MySQL を通常ユーザーとして実行する方法」](#)。
- [セクション6.1.4「セキュリティ関連の mysqld オプションおよび変数」](#)。
- [セクション6.1.6「LOAD DATA LOCAL のセキュリティ上の考慮事項」](#)。
- [セクション2.10「インストール後のセットアップとテスト」](#)。
- [セクション6.3「暗号化された接続の使用」](#)。
- [Loadable Function Security Precautions](#)。

A.9.2. MySQL 8.0 のデフォルトの認証プラグインは何ですか。

MySQL 8.0 のデフォルトの認証プラグインは `caching_sha2_password` です。このプラグインについては、[セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#)を参照してください。

`caching_sha2_password` プラグインは、`mysql_native_password` プラグイン (以前の MySQL シリーズのデフォルトプラグイン) よりもセキュアなパスワード暗号化を提供します。サーバー操作のためのデフォルトのプラグインのこの変更による影響、およびサーバーとクライアントおよびコネクタとの互換性については、[優先認証プラグインとしての caching_sha2_password](#) を参照してください。

プラガブル認証およびその他の使用可能な認証プラグインの一般情報は、[セクション6.2.17「プラガブル認証」](#) および [セクション6.4.1「認証プラグイン」](#) を参照してください。

A.9.3. MySQL 8.0 には SSL に対するネイティブなサポートはありますか。

ほとんどの 8.0 バイナリには、クライアントおよびサーバー間の SSL 接続のサポートがあります。 [セクション 6.3「暗号化された接続の使用」](#) を参照してください。

(たとえば) クライアントアプリケーションで SSL 接続がサポートされない場合は、SSH を使用して接続をトンネルすることもできます。例については、 [セクション 6.3.4「SSH を使用した Windows から MySQL へのリモート接続」](#) を参照してください。

A.9.4. SSL サポートは MySQL バイナリに組み込まれていますか。または、バイナリを有効にするにはバイナリを自分で再コンパイルする必要がありますか。

ほとんどの 8.0 バイナリでは、保護、認証、またはその両方のクライアント/サーバー接続に対して SSL が有効になっています。 [セクション 6.3「暗号化された接続の使用」](#) を参照してください。

A.9.5. MySQL 8.0 には LDAP ディレクトリに対する組み込みの認証はありますか。

Enterprise エディションには、LDAP ディレクトリに対する認証をサポートする [PAM Authentication Plugin](#) が含まれています。

A.9.6. MySQL 8.0 にはロールベースのアクセス制御 (RBAC) のサポートは含まれていますか。

現在はありません。

A.10 MySQL 8.0 FAQ: NDB Cluster

次のセクションでは、MySQL NDB Cluster および [NDB](#) ストレージエンジンについてよくある質問に答えます。

A.10.1 NDB Cluster をサポートする MySQL ソフトウェアのバージョンはどれですか。ソースからコンパイルする必要がありますか。	4552
A.10.2 「NDB」 および 「NDBCLUSTER」 は何を意味していますか。	4553
A.10.3 NDB Cluster の使用と MySQL レプリケーションの使用の違いは何ですか。	4553
A.10.4 NDB Cluster を実行するために特別なネットワークは必要ですか。クラスタ内のコンピュータはどのように通信しますか。	4553
A.10.5 NDB Cluster を実行するために必要なコンピュータの数とその理由を教えてください。	4553
A.10.6 NDB Cluster では、どのようなコンピュータが機能しますか。	4553
A.10.7 NDB Cluster 管理クライアントで SHOW コマンドを実行すると、次のような出力行が表示されます:	4554
A.10.8 NDB Cluster はどのオペレーティングシステムで使用できますか。	4554
A.10.9 NDB Cluster を実行するためのハードウェア要件は何ですか。	4555
A.10.10 NDB Cluster を使用するには、どのくらいの RAM が必要ですか。ディスクメモリーを使用することはできますか。	4555
A.10.11 NDB Cluster ではどのファイルシステムを使用できますか。ネットワークファイルシステムまたはネットワーク共有は使用できますか。	4556
A.10.12 NDB Cluster ノードを仮想マシン (VMWare、VirtualBox、Parallels、Xen によって作成されたものなど) 内で実行できますか。	4556
A.10.13 NDB Cluster データベースにデータを移入しようとしています。ロード処理が予期せず終了し、次のようなエラーメッセージが表示されます。	4556
A.10.14 NDB Cluster は TCP/IP を使用します。これは、1 つ以上のノードをリモートの場所に配置して、インターネット経由で実行できることを意味しますか。	4557
A.10.15 NDB Cluster を使用するには、新しいプログラミング言語またはクエリー言語を学習する必要がありますか。	4557
A.10.16 NDB Cluster でサポートされているプログラミング言語および API は何ですか。	4557
A.10.17 NDB Cluster には管理ツールは含まれていますか。	4557
A.10.18 NDB Cluster の使用時にエラーまたは警告メッセージが何を意味しているかを調べるにはどうすればよいですか。	4558
A.10.19 NDB Cluster トランザクションセーフですか。どのような分離レベルがサポートされますか。	4558
A.10.20 NDB Cluster でサポートされているストレージエンジンを教えてください。	4558
A.10.21 致命的な障害が発生した場合 (たとえば、都市全体の電力が失われ、UPS で障害が発生した場合、すべてのデータが失われますか。	4558
A.10.22 NDB Cluster で FULLTEXT インデックスを使用できますか。	4558
A.10.23 単一のコンピュータ上で複数のノードを実行できますか。	4558

A.10.24 NDB Cluster を再起動せずにデータノードを追加できますか。	4559
A.10.25 NDB Cluster を使用するときの注意すべき制限はありますか。	4559
A.10.26 NDB Cluster は外部キーをサポートしていますか。	4560
A.10.27 NDB Cluster に既存の MySQL データベースをインポートするにはどうすればよいですか。	4560
A.10.28 NDB Cluster ノードはどのように相互に通信しますか。	4560
A.10.29 アービトラータとは何ですか。	4560
A.10.30 NDB Cluster でサポートされているデータ型は何ですか。	4560
A.10.31 NDB Cluster を起動および停止するにはどうすればよいですか。	4561
A.10.32 NDB Cluster をシャットダウンすると、NDB Cluster データはどうなりますか。	4561
A.10.33 NDB Cluster に複数の管理ノードを設定することをお勧めしますか。	4561
A.10.34 1つの NDB Cluster に異なる種類のハードウェアとオペレーティングシステムを混在させることはできますか。	4562
A.10.35 単一のホスト上で2つのデータノードを実行できますか。2つの SQL ノードは実行できますか。	4562
A.10.36 NDB Cluster でホスト名を使用できますか。	4562
A.10.37 NDB Cluster は IPv6 をサポートしていますか。	4562
A.10.38 複数の MySQL サーバーを持つ NDB Cluster で MySQL ユーザーを処理するにはどうすればよいですか。	4562
A.10.39 SQL ノードの1つで障害が発生した場合に、クエリーの送信を続けるにはどうすればよいですか。	4562
A.10.40 NDB Cluster をバックアップおよび復元するにはどうすればよいですか。	4562
A.10.41 「エンジェルプロセス」とは何ですか。	4562

A.10.1 NDB Cluster をサポートする MySQL ソフトウェアのバージョンはどれですか。ソースからコンパイルする必要がありますか。

NDB Cluster は、標準の MySQL Server 8.0 リリースではサポートされません。代わりに、MySQL NDB Cluster は別の製品として提供されます。使用可能な NDB Cluster リリースシリーズには、次のものが含まれます:

- NDB Cluster 7.2. このシリーズは、新しいデプロイメントまたはメンテナンスではサポートされなくなりました。NDB Cluster 7.2 のユーザーは、できるだけ早く新しいリリースシリーズにアップグレードするようにしてください。新しい配備では、最新の NDB Cluster 8.0 リリースを使用することをお勧めします。
- NDB Cluster 7.3. このシリーズは NDB Cluster の以前の General Availability (GA) バージョンであり、本番で引き続き使用できますが、新しい配備では最新の NDB Cluster 8.0 リリースを使用することをお勧めします。最新の NDB Cluster 7.3 リリースは、<https://dev.mysql.com/downloads/cluster/> から取得できます。
- NDB Cluster 7.4. このシリーズは NDB Cluster の以前の General Availability (GA) バージョンであり、本番で引き続き使用できますが、新しい配備では最新の NDB Cluster 8.0 リリースを使用することをお勧めします。最新の NDB Cluster 7.4 リリースは、<https://dev.mysql.com/downloads/cluster/> から取得できます。
- NDB Cluster 7.5. このシリーズは NDB Cluster の以前の General Availability (GA) バージョンであり、本番で引き続き使用できますが、新しい配備では最新の NDB Cluster 7.6 リリースを使用することをお勧めします。最新の NDB Cluster 7.5 リリースは、<https://dev.mysql.com/downloads/cluster/> から取得できます。
- NDB Cluster 7.6. このシリーズは NDB Cluster の以前の General Availability (GA) バージョンであり、本番で引き続き使用できますが、新しい配備では最新の NDB Cluster 8.0 リリースを使用することをお勧めします。最新の NDB Cluster 7.6 リリースは、<https://dev.mysql.com/downloads/cluster/> から取得できます。
- NDB Cluster 8.0. このシリーズは、NDB ストレージエンジンおよび MySQL Server 8.0 のバージョン 8.0 に基づいた NDB Cluster の最新の General Availability (GA) バージョンです。NDB Cluster 8.0 は本番で使用できます。本番用の新しい配備では、現在 NDB Cluster 8.0.23 であるこのシリーズの最新の GA リリースを使用するようにしてください。<https://dev.mysql.com/downloads/cluster/> から最新の NDB Cluster 8.0 リリースを取得できます。このシリーズの新機能およびその他の重要な変更の詳細は、[What is New in NDB Cluster](#) を参照してください。

NDB Cluster はソース ([セクション23.2.1.4「Linux でのソースからの NDB Cluster の構築」](#) および [セクション23.2.2「Windows でのソースからの NDB Cluster のコンパイルとインストール」](#) を参照) から取得およびコンパイルできますが、もっとも特殊なケースを除くすべてのケースで、使用しているオペレーティングシステムおよび状況に適した Oracle によって提供される次のインストーラのいずれかを使用することをお勧めします:

- web ベースの [NDB Cluster Auto-Installer](#) (NDB でサポートされているすべてのプラットフォームで動作)

- Linux [binary release](#) (tar.gz ファイル)
- Linux [RPM package](#)
- Linux [.deb file](#)
- Windows [binary 「no-install」 release](#)
- Windows [MSI Installer](#)

インストールパッケージは、プラットフォームパッケージ管理システムからも入手できます。

使用している MySQL Server で NDB がサポートされるかどうかを判別するには、[SHOW VARIABLES LIKE 'have_%'](#)、[SHOW ENGINES](#)、または [SHOW PLUGINS](#) ステートメントのいずれかを使用します。

A.10.2. 「NDB」 および 「NDBCLUSTER」 は何を意味していますか。

「NDB」は「Network Database」を意味しています。NDB と NDBCLUSTER はどちらも、MySQL でのクラスタリングのサポートを可能にするストレージエンジンの名前です。NDB をお勧めしますが、どちらの名前も正しいです。

A.10.3.NDB Cluster の使用と MySQL レプリケーションの使用の違いは何ですか。

従来の MySQL レプリケーションでは、ソース MySQL サーバーが 1 つ以上のレプリカを更新します。トランザクションは順次コミットされ、遅いトランザクションによってレプリカがソースより遅れる可能性があります。これは、ソースに障害が発生した場合、レプリカが最後のいくつかのトランザクションを記録していない可能性があることを意味します。InnoDB などのトランザクションセーフエンジンが使用されている場合、トランザクションはレプリカで完了するか、まったく適用されませんが、レプリケーションではソースとレプリカのすべてのデータが常に一貫していることは保証されません。NDB Cluster では、すべてのデータノードの同期が維持され、いずれかのデータノードによってコミットされたトランザクションがすべてのデータノードに対してコミットされます。データノードの障害が発生した場合、ほかのすべてのデータノードでは整合性のある状態が維持されます。

つまり、標準の MySQL レプリケーションは非同期ですが、NDB Cluster は同期です。

非同期レプリケーションは NDB Cluster でも使用できます。NDB Cluster レプリケーション（「geo-replication」とも呼ばれる）には、2 つの NDB Cluster 間および NDB Cluster から非クラスタ MySQL サーバーへの両方をレプリケートする機能が含まれています。[セクション23.6 「NDB Cluster レプリケーション」](#)を参照してください。

A.10.4.NDB Cluster を実行するために特別なネットワークは必要ですか。クラスタ内のコンピュータはどのように通信しますか。

NDB Cluster は、TCP/IP を使用してコンピュータを接続する高帯域幅環境で使用することを目的としています。そのパフォーマンスは、クラスタ内のコンピュータ間の接続速度に直接関係しています。NDB Cluster の最小接続要件には、標準的な 100 メガビットイーサネットネットワークまたは同等のものが含まれます。可能な場合はギガビット Ethernet を使用することをお勧めします。

A.10.5.NDB Cluster を実行するために必要なコンピュータの数とその理由を教えてください。

実行可能なクラスタの実行には、少なくとも 3 台のコンピュータが必要となります。ただし、NDB Cluster 内のコンピュータの最小推奨数は 4 です: それぞれが管理ノードと SQL ノードを実行し、2 台のコンピュータがデータノードとして機能します。データノードを 2 台にする目的は冗長性を持たせるためです。管理ノードは別個のマシンで実行して、いずれかのデータノードで障害が発生した場合にアービトレーションサービスが継続されることを保証する必要があります。

スループットおよび高可用性を向上させるには、複数の SQL ノード (クラスタに接続された MySQL サーバー) を使用してください。複数の管理サーバーを実行することもできます (必須ではありません)。

A.10.6.NDB Cluster では、どのようなコンピュータが機能しますか。

NDB Cluster には物理的な編成と論理的な編成の両方があり、コンピュータは物理的な要素です。クラスタの論理要素または機能要素はノードと呼ばれ、クラスタのノードを収容するコンピュータはクラスタホストと呼

ばれることがあります。クラスタ内の特定のロールにそれぞれ対応する 3 つのタイプのノードがあります。これらを次に示します。

- 管理ノード. このノードはクラスタ全体の管理サービス (起動、シャットダウン、バックアップ、およびほかのノードの構成データを含む) を提供します。管理ノードサーバーはアプリケーション `ndb_mgmd` として実装され、NDB Cluster の制御に使用される管理クライアントは `ndb_mgm` です。これらのプログラムについては、[セクション 23.4.4 「ndb_mgmd — NDB Cluster 管理サーバーデーモン」](#) および [セクション 23.4.5 「ndb_mgm — NDB Cluster 管理クライアント」](#) を参照してください。
- データノード. このタイプのノードは、データを格納およびレプリケートします。データノードの機能は、NDB データノードプロセス `ndbd` のインスタンスによって処理されます。詳細は、[セクション 23.4.1 「ndbd — NDB Cluster データノードデーモン」](#) を参照してください。
- SQL ノード. これは、NDBCLUSTER ストレージエンジンをサポートして構築され、エンジンを有効にするための `--ndb-cluster` オプションと NDB Cluster 管理サーバーに接続できるようにするための `--ndb-connectstring` オプションで起動される MySQL Server (`mysqld`) の単なるインスタンスです。これらのオプションについては、[NDB Cluster の MySQL Server オプション](#) を参照してください。

注記

API ノードは、データの格納および取得のためにクラスタのデータノードを直接使用するアプリケーションです。このため、SQL ノードは MySQL サーバーを使用してクラスタへの SQL インタフェースを提供する API ノードの一種であると考えられます。NDB Cluster データに直接のオブジェクト指向トランザクションおよびスキーマインタフェースを提供する NDB API を使用して、そのようなアプリケーション (MySQL Server に依存しない) を記述できます。詳細は、[NDB Cluster API Overview: The NDB API](#) を参照してください。

A.10.7.NDB Cluster 管理クライアントで `SHOW` コマンドを実行すると、次のような出力行が表示されます:

```
id=2 @10.100.10.32 (Version: 8.0.23-ndb-8.0.23 Nodegroup: 0, *)
```

「*」は何を意味していますか。このノードはほかのノードとどのように異なりますか。

最も簡単な回答は、「NDB Cluster のソースコードを記述または分析するソフトウェアエンジニアでない限り、これは制御できないため、どのような場合でも心配する必要はありません」です。

この回答に満足できない場合、より長くテクニカルなバージョンは次のとおりです。

NDB Cluster 内の多くのメカニズムでは、データノード間で分散調整が必要です。これらの分散アルゴリズムおよびプロトコルには、グローバルチェックポイント、DDL (スキーマ) の変更、およびノードの再起動処理が含まれます。この調整を単純にするために、それらのメンバーのいずれかがリーダーとして動作するようにデータノードで「選出」されます。この選択に影響を与えるユーザー向けメカニズムはありません。これは完全に自動です。これは NDB Cluster 内部アーキテクチャーの重要な部分です。

ノードがこれらのメカニズムの「リーダー」として動作する場合、通常、それがアクティビティの調整の中心となり、「フォロワー」として動作するほかのノードは、リーダーによって指示されたアクティビティの各自の担当分を実行します。リーダーとして動作するノードで障害が発生すると、残りのノードによって新しいリーダーが選出されます。古いリーダーによって調整されていた進行中のタスクは、実際に関係するメカニズムに従って、失敗するか、新しいリーダーによって続行されます。

これらの各種のメカニズムおよびプロトコルの一部で別のリーダーノードが使用されることがありますが、一般的には、それらのすべてで同じリーダーが選択されます。管理クライアントの `SHOW` の出力でリーダーとして示されるノードは、DDL およびメタデータアクティビティの調整を担当する `DICT` マネージャと内部的に呼ばれます。

NDB Cluster は、リーダーの選択がクラスタ自体の外部で認識できないように設計されています。たとえば、現在のリーダーの CPU またはリソースの使用量がほかのデータノードより著しく高いことはなく、リーダーで障害が発生した場合にほかのデータノードで障害が発生した場合よりクラスタに対して特別に大きい影響があるということはありません。

A.10.8.NDB Cluster はどのオペレーティングシステムで使用できますか。

NDB Cluster は、ほとんどの Unix に似たオペレーティングシステムでサポートされています。NDB Cluster は、Microsoft Windows オペレーティングシステムの本番設定でもサポートされています。

さまざまなオペレーティングシステムバージョン、オペレーティングシステム配布、およびハードウェアプラットフォームで NDB Cluster に提供されるサポートのレベルに関する詳細は、<https://www.mysql.com/support/supportedplatforms/cluster.html> を参照してください。

A.10.9.NDB Cluster を実行するためのハードウェア要件は何ですか。

NDB Cluster は、NDB 対応バイナリが使用可能な任意のプラットフォームで実行するようにしてください。データノードおよび API ノードの場合、より速い CPU およびより多くのメモリによって、パフォーマンスが向上することがあり、64 ビットの CPU は 32 ビットのプロセッサよりも効率的である場合があります。各ノードのデータベースの負担を保持するために、データノードに使用されるマシンに十分なメモリがある必要があります (詳細は、「必要な RAM のサイズはどれくらいですか」を参照してください)。NDB Cluster 管理サーバーの実行にのみ使用されるコンピュータの場合、要件は最小限です。一般に、このタスクには共通のデスクトップ PC (または同等のもの) で十分です。ノードは標準の TCP/IP ネットワークおよびハードウェアを使用して通信できます。高速の SCI プロトコルを使用することもできます。ただし、SCI を使用するには、特殊なネットワークハードウェアおよびソフトウェアが必要となります (セクション 23.3.4 「NDB Cluster での高速インターコネクトの使用」を参照してください)。

A.10.10.NDB Cluster を使用するには、どのくらいの RAM が必要ですか。ディスクメモリーを使用することはできますか。

NDB Cluster は最初はインメモリーとしてのみ実装されましたが、現在使用可能なすべてのバージョンで NDB Cluster をディスクに格納する機能も提供されています。詳細は、セクション 23.5.10 「NDB Cluster ディスクデータテーブル」を参照してください。

インメモリーの NDB テーブルの場合は、クラスタ内の各データノードで必要となる RAM の容量のおおよその見積もりを取得するために次の式を使用できます。

```
(SizeofDatabase × NumberOfReplicas × 1.1) / NumberOfDataNodes
```

メモリー要件をより正確に計算するには、クラスタデータベースの各テーブルで行ごとに必要となる格納領域 (詳細は、セクション 11.7 「データ型のストレージ要件」を参照してください) を判別して、それを行数で乗算する必要があります。カラムインデックスについては、次のことを考慮する必要があります。

- NDBCLUSTER テーブルに作成される各主キーまたはハッシュインデックスには、レコードごとに 21-25 バイトが必要となります。これらのインデックスは `IndexMemory` を使用します。
- 順序付けされた各インデックスでは、`DataMemory` を使用して、レコードごとに 10 バイトのストレージが必要となります。
- また、主キーまたは一意のインデックスを作成すると、このインデックスが `USING HASH` を指定して作成された場合を除き、順序付けされたインデックスが作成されます。言い換えると、次のようになります。
 - 通常、クラスタテーブルの主キーまたは一意のインデックスには、レコードごとに 31 - 35 バイトが使用されます。
 - ただし、主キーまたは一意のインデックスが `USING HASH` を指定して作成されている場合は、レコードごとに 21 バイトから 25 バイトのみが必要となります。

すべての主キーおよび一意インデックスに対して `USING HASH` を使用する「NDB Cluster の作成」テーブルでは、通常、`USING HASH` が主キーおよび一意キーの作成に使用されなかったテーブルの更新よりも 20 から 30% 速く、テーブルの更新が高速に実行されます。これは、必要なメモリーが少なくなり (順序付けされたインデックスが作成されないため)、使用される CPU が少なくなる (読み取りおよび (場合によっては) 更新する必要があるインデックスが少なくなるため) ためです。ただし、これは、本来範囲スキャンを使用するクエリーをほかの方法で実行する必要があることも意味し、選択が低速になることがあります。

クラスタのメモリー要件を計算するときに、最新の MySQL 8.0 リリースに付属している `ndb_size.pl` ユーティリティーが役に立つことがあります。この Perl スクリプトは、現在の (クラスタではない) MySQL データベースに接続して、NDBCLUSTER ストレージエンジンを使用した場合にデータベースで必要となる容量に関する報告を作成します。詳細は、セクション 23.4.28 「`ndb_size.pl` — NDBCLUSTER サイズ要件エスティメータ」を参照してください。

すべての「NDB Cluster」テーブルに主キーが必要であることを覚えておくことは特に重要です。NDB ストレージエンジンは、主キーが定義されていない場合、主キーを自動的に作成します。この主キーは `USING HASH` を指定せずに作成されます。

NDB Cluster データおよびインデックスの格納に使用されているメモリーの量は、`ndb_mgm` クライアントの `REPORT MEMORYUSAGE` コマンドを使用していつでも確認できます。詳細は、[セクション23.5.1「NDB Cluster 管理クライアントのコマンド」](#) を参照してください。また、使用可能な `DataMemory` または NDB 7.6 より前の `IndexMemory` の 80% が使用されているとき、および使用率が 90%、99% および 100% に達したときに、警告がクラスタログに書き込まれます。

- A.10.1 NDB Cluster ではどのファイルシステムを使用できますか。 ネットワークファイルシステムまたはネットワーク共有は使用できますか。

一般に、ホストオペレーティングシステムにネイティブなファイルシステムは NDB Cluster で正常に動作します。NDB Cluster で特定のファイルシステムが特に適切に動作する (または特にうまく動作しない) ことがなかった場合は、「[NDB Cluster フォーラム](#)」での結果について話し合うように招待します。

Windows の場合は、標準の MySQL の場合と同様に、NDB Cluster に `NTFS` ファイルシステムを使用することをお勧めします。`FAT` または `VFAT` ファイルシステムで NDB Cluster をテストしません。このため、MySQL または NDB Cluster での使用はお勧めしません。

NDB Cluster はシェアードナッシングのソリューションとして実装されています。これの背後にある考えは、単一のハードウェアの障害によって複数のクラスタノードの障害が発生したり、場合によってはクラスタ全体の障害が発生したりしないことです。このため、NDB Cluster ではネットワーク共有またはネットワークファイルシステムの使用はサポートされていません。これは、SAN などの共有ストレージデバイスにも当てはまりません。

- A.10.1 NDB Cluster ノードを仮想マシン (VMWare、VirtualBox、Parallels、Xen によって作成されたものなど) 内で実行できますか。

NDB Cluster は、仮想マシンでの使用がサポートされています。現在、[Oracle VM](#) を使用してサポートおよびテストが行われています。

一部の NDB Cluster ユーザーは、ほかの仮想化製品を使用して NDB Cluster を正常に配備しました。このような場合、Oracle は NDB Cluster のサポートを提供できますが、仮想環境に固有の問題はその製品ベンダーに示す必要があります。

- A.10.1 NDB Cluster データベースにデータを移入しようとしています。ロード処理が予期せずに終了し、次のようなエラーメッセージが表示されます。

```
「ERROR 1114: The table 'my_cluster_table' is full」
```

これが発生するのはなぜですか。

原因はすべてのテーブルデータおよびすべてのインデックス (テーブル定義に主キーの定義が含まれていない場合に自動的に作成される、NDB ストレージエンジンによって要求される主キーを含む) のための十分な RAM が、使用しているセットアップで提供されていないことである可能性があります。

すべてのデータノードで同じ容量の RAM が使用されることにも注意してください。クラスタ内のデータノードは、データノードの中で使用可能なメモリーの容量がもっとも少ないノードより多いメモリーを使用することはできないためです。たとえば、クラスタデータノードをホストしているコンピュータが 4 台あり、これらのうち 3 台にクラスタデータを格納できる 3GB の RAM があり、残りのデータノードには 1GB の RAM しかない場合、各データノードは NDB Cluster データおよびインデックス専用にすることができます。

場合によっては、`ndb_mgm -e "ALL REPORT MEMORYUSAGE"` で `DataMemory` に大きい空き領域が示されていても `Table is full` というエラーが MySQL クライアントアプリケーションで表示されることがあります。`CREATE TABLE` の `MAX_ROWS` オプションを使用すると、NDB で強制的に「NDB Cluster」テーブルの追加パーティションを作成し、ハッシュインデックスに使用可能なメモリーを増やすことができます。通常、テーブルに格納することが予期されている行数の 2 倍の数値を `MAX_ROWS` に設定すれば十分です。

同様の理由で、データが大量にロードされたノードで、データノードが再起動される問題が発生することもあります。`MinFreePct` パラメータは、`DataMemory` および (NDB 7.6 より前の) `IndexMemory` の一部 (デフォルト

トでは 5%) を再起動で使用するために予約することで、この問題を解決できます。この予約されたメモリーは、NDB テーブルまたはデータの格納には使用できません。

A.10.14 NDB Cluster は TCP/IP を使用します。これは、1 つ以上のノードをリモートの場所に配置して、インターネット経由で実行できることを意味しますか。

NDB Cluster は 100 Mbps またはギガビット Ethernet を使用した LAN 設定で見つかったような専用の高速接続を保証する条件で実行されることを前提として設計および実装されているため、このような状況でクラスタが確実に実行される可能性は very ではありません。これより遅い環境で使用したときのパフォーマンスはテストされておらず保証されません。

また、NDB Cluster 内のノード間の通信はセキュリティー保護されておらず、ほかの保護メカニズムによって暗号化も保護もされていないことに注意してください。クラスタのもっともセキュアな構成は、外部からクラスタのデータまたは管理ノードに直接アクセスされない、ファイアウォールの内側のプライベートネットワークです。(SQL ノードの場合は、MySQL サーバーのほかのインスタンスの場合と同様の予防措置を取るようになっています。) 詳細は、[セクション 23.5.17 「NDB Cluster のセキュリティーの問題」](#) を参照してください。

A.10.15 NDB Cluster を使用するには、新しいプログラミング言語またはクエリー言語を学習する必要がありますか。

いいえ。クラスタ自体の管理および構成にはいくつかの特殊なコマンドが使用されますが、次の操作には標準の (MySQL) ステートメントのみが必要となります。

- テーブルの作成、変更、および削除
- テーブルデータの挿入、更新、および削除
- プライマリインデックスおよび一意のインデックスの作成、変更、および削除

NDB Cluster を設定するには、いくつかの特殊な構成パラメータおよびファイルが必要です。これらについては、[セクション 23.3.3 「NDB Cluster 構成ファイル」](#) を参照してください。

NDB Cluster 管理クライアント (`ndb_mgm`) では、クラスタノードの起動や停止などのタスクにいくつかの単純なコマンドが使用されます。[セクション 23.5.1 「NDB Cluster 管理クライアントのコマンド」](#) を参照してください。

A.10.16 NDB Cluster でサポートされているプログラミング言語および API は何ですか。

NDB Cluster は、ODBC、.Net、MySQL C API、PHP、Perl、Python などの一般的なスクリプト言語用の多数のドライバなど、標準の MySQL Server と同じプログラミング API および言語をサポートしています。これらの API を使用して記述された NDB Cluster アプリケーションは、ほかの MySQL アプリケーションと同様に動作し、SQL ステートメントを MySQL Server (NDB Cluster の場合は SQL ノード) に送信し、データ行を含む応答を受信します。これらの API については、[第 29 章 「Connector および API」](#) を参照してください。

NDB Cluster は NDB API を使用したアプリケーションプログラミングもサポートしています。NDB API は、MySQL Server を経由せずに NDB Cluster データへの下位レベルの C++ インタフェースを提供します。[The NDB API](#) を参照してください。また、多くの `NDBCLUSTER` 管理関数が C 言語の MGM API によって提供されています。詳細は、[The MGM API](#) を参照してください。

NDB Cluster は、セッションおよびトランザクションを使用したデータのドメインオブジェクトモデルをサポートする ClusterJ を使用した Java アプリケーションプログラミングもサポートしています。詳細は、[Java and NDB Cluster](#) を参照してください。

さらに、NDB Cluster は `memcached` をサポートしているため、開発者は `memcached` インタフェースを使用して NDB Cluster に格納されているデータにアクセスできます。詳細は、[ndbmemcache—Memcache API for NDB Cluster \(NO LONGER SUPPORTED\)](#) を参照してください。

NDB Cluster には、NDB Cluster をデータストアとして、`Node.js` に対して記述された NoSQL アプリケーションをサポートするアダプタも含まれています。詳細は、[MySQL NoSQL Connector for JavaScript](#) を参照してください。

A.10.17 NDB Cluster には管理ツールは含まれていますか。

NDB Cluster には、基本管理機能を実行するためのコマンドラインクライアントが含まれています。 [セクション23.4.5「ndb_mgm — NDB Cluster 管理クライアント」](#) および [セクション23.5.1「NDB Cluster 管理クライアントのコマンド」](#) を参照してください。

NDB Cluster 7.6 以前は、ローリング再起動や構成変更などの多くの NDB Cluster 管理タスクを自動化できる高度なコマンド行インターフェースを提供する個別の製品である MySQL Cluster Manager でもサポートされています。バージョン 1.4.8 以降、MySQL Cluster Manager は NDB Cluster 8.0 の実験的なサポートも提供します。MySQL Cluster Manager については、 [MySQL Cluster Manager 1.4.8 User Manual](#) を参照してください。

NDB Cluster は、NDB Cluster ソフトウェアディストリビューションの一部として、NDB Cluster を設定および配備するためのグラフィカルブラウザベースの Auto-Installer も提供します。詳細は、 [The NDB Cluster Auto-Installer \(NDB 7.5\) \(No longer supported\)](#) を参照してください。

A.10.1 ~~N~~NDB Cluster の使用時にエラーまたは警告メッセージが何を意味しているかを調べるにはどうすればよいですか。

これを行うことができる方法は 2 つあります。

- エラー状態または警告状態を通知された直後に、mysql クライアント内で `SHOW ERRORS` または `SHOW WARNINGS` を使用します。
- システムのシェルプロンプトで、`peror --ndb_error_code` を使用します。

A.10.1 ~~N~~NDB Cluster トランザクションセーフですか。どのような分離レベルがサポートされますか。

はい。NDB ストレージエンジンを指定して作成されたテーブルの場合は、トランザクションがサポートされます。現在、NDB Cluster は `READ COMMITTED` トランザクション分離レベルのみをサポートしています。

A.10.2 ~~N~~NDB Cluster でサポートされているストレージエンジンを教えてください。

NDB Cluster には NDB ストレージエンジンが必要です。つまり、NDB Cluster 内のノード間でテーブルを共有するには、`ENGINE=NDB` (または同等のオプション `ENGINE=NDBCLUSTER`) を使用してテーブルを作成する必要があります。

NDB Cluster で使用されている MySQL サーバー上で、ほかのストレージエンジン (InnoDB や MyISAM など) を使用してテーブルを作成することは可能ですが、これらのテーブルは NDB を使用しないため、クラスタ化に参加しません。このような各テーブルは、作成される個々の MySQL サーバーインスタンスに対して厳密にローカルです。

NDB Cluster は、アーキテクチャー、要件、および実装に関して InnoDB クラスタリングとは大きく異なります。名前に類似点があるにもかかわらず、両者は互換性がありません。InnoDB クラスタリングの詳細は、 [MySQL AdminAPI の使用](#) を参照してください。NDB ストレージエンジンと InnoDB ストレージエンジンの違いについては、 [セクション23.1.6「MySQL Server NDB Cluster と比較した InnoDB の使用」](#) も参照してください。

A.10.2 ~~N~~致命的な障害が発生した場合 (たとえば、都市全体の電力が失われ、UPS で障害が発生した場合、すべてのデータが失われますか。

コミットされたすべてのトランザクションはログ記録されます。このため、大災害が起こった場合に一部のデータが失われることはありますが、それはごく限定的なものとなります。データの損失は、トランザクションごとの操作の数を最小限にすることによって、さらに減らすことができます。(どのような場合でも、1 つのトランザクションで多数の操作を実行するのはよい考えではありません。)

A.10.2 ~~N~~NDB Cluster で `FULLTEXT` インデックスを使用できますか。

`FULLTEXT` のインデックス作成は現在、InnoDB および MyISAM ストレージエンジンでのみサポートされています。詳細は、 [セクション12.10「全文検索関数」](#) を参照してください。

A.10.2 ~~N~~単一のコンピュータ上で複数のノードを実行できますか。

実行できますが常に推奨できるとは限りません。クラスタを実行する主な理由の 1 つは冗長性を持たせるためです。この冗長性の利点を完全に享受するには、各ノードを別個のマシン上に配置してください。単一のマシンに複数のノードを配置した場合、そのマシンで障害が発生したときに、それらのすべてのノードが失われ

ます。このため、単一のマシンで複数のデータノードを実行する場合は、そのマシンの障害によってノードグループのすべてのデータノードが失われることがないように設定することが非常に重要です。

NDB Cluster は、低コスト (またはコストなし) のオペレーティングシステムでロードされたコモディティハードウェア上で実行できるため、余分なマシンまたは 2 台のマシンを使用すると、ミッションクリティカルなデータを保護する価値があります。管理ノードを実行するクラスタホストの要件は最低限のものであることにも注意してください。このタスクは、300 MHz の Pentium または同等の CPU、オペレーティングシステムのための十分な RAM、および `ndb_mgmd` プロセスと `ndb_mgm` プロセスのための少量のオーバーヘッドに対応した装備があれば実行できます。

複数の CPU、コア、またはその両方を持つ単一のホストで、複数のクラスタデータノードを実行することは容認できます。NDB Cluster ディストリビューションは、このようなシステムでの使用を目的としたマルチスレッドバージョンのデータノードバイナリも提供します。詳細は、[セクション23.4.3「ndbmt — NDB Cluster データノードデーモン \(マルチスレッド\)」](#)を参照してください。

同じマシン上でデータノードと SQL ノードを同時に実行できる場合もあります。そのような配置での実行状態は、さまざまな要因 (コアおよび CPU の数、データノードおよび SQL ノードのプロセスが使用できるディスクおよびメモリの容量など) によって異なります。そのような構成を計画する場合は、これらの要因を考慮する必要があります。

A.10.24 NDB Cluster を再起動せずにデータノードを追加できますか。

クラスタをオフラインにせずに、実行中の NDB Cluster に新しいデータノードを追加できます。詳細は、[セクション23.5.7「NDB Cluster データノードのオンラインでの追加」](#)を参照してください。

ほかのタイプの NDB Cluster ノードの場合は、ローリング再起動がすべて必要です ([セクション23.5.5「NDB Cluster のローリング再起動の実行」](#)を参照)。

A.10.25 NDB Cluster を使用するときの注意すべき制限はありますか。

MySQL NDB Cluster での NDB テーブルの制限事項は次のとおりです:

- 一時テーブルはサポートされません。ENGINE=NDB または ENGINE=NDBCLUSTER を使用した CREATE TEMPORARY TABLE ステートメントは、エラーが発生して失敗します。
- NDBCLUSTER テーブルでサポートされるユーザー定義のパーティション化のタイプは、KEY および LINEAR KEY のみです。ほかのパーティショニングタイプを使用して NDB テーブルを作成しようとすると、エラーが発生して失敗します。
- FULLTEXT インデックスはサポートされません。
- インデックスのプリフィクスはサポートされません。インデックスを作成できるのは完全なカラムのみです。
- 空間インデックスはサポートされません (空間カラムは使用できます)。 [セクション11.4「空間データ型」](#)を参照してください。
- 部分的なトランザクションおよび部分的なロールバックのサポートは、ほかのトランザクションストレージエンジン (個別のステートメントをロールバックできる InnoDB など) と同等です。
- テーブルごとに許可される属性の最大数は 512 個です。属性名は 31 文字以内である必要があります。各テーブルで、テーブル名とデータベース名を合わせた最大長は 122 文字です。
- NDB 8.0 では、テーブル行の最大サイズは 14 K バイトであり、BLOB 値はカウントされません。NDB 8.0 では、この最大値は 30000 バイトに増加します。詳細は、[セクション23.1.7.5「NDB Cluster 内のデータベースオブジェクトに関連付けられる制限」](#)を参照してください。

NDB テーブルごとの行数の制限はありません。テーブルサイズの制限は多くの要因によって異なり、各データノードで使用できる RAM の容量に特に関係しています。

NDB Cluster の制限の完全なリストについては、[セクション23.1.7「NDB Cluster の既知の制限事項」](#)を参照してください。[セクション23.1.7.11「前 NDB Cluster 8.0 で解決される NDB Cluster の問題」](#)も参照してください。

A.10.2 NDB Cluster は外部キーをサポートしていますか。

NDB Cluster は、**InnoDB** ストレージエンジンで検出されるものと同等の外部キー制約のサポートを提供します。詳細は、[セクション1.7.3.2「FOREIGN KEY の制約」](#)、および [セクション13.1.20.5「FOREIGN KEY の制約」](#) を参照してください。外部キーのサポートが必要なアプリケーションは NDB Cluster 7.3, 7.4, 7.5 以降を使用するようにしてください。

A.10.2 NDB Cluster に既存の MySQL データベースをインポートするにはどうすればよいですか。

NDB Cluster には、ほかのバージョンの MySQL と同様にデータベースをインポートできます。この FAQ の各所で説明されている制限以外の唯一の特別な要件は、クラスタに含まれるテーブルが **NDB** ストレージエンジンを使用する必要があることです。これは、**ENGINE=NDB** または **ENGINE=NDBCLUSTER** を指定してテーブルを作成する必要があることを意味します。

ほかのストレージエンジンを使用しているテーブルを、1つ以上の **ALTER TABLE** ステートメントを使用して、**NDBCLUSTER** に変更することもできます。ただし、変更を行う前に、テーブルの定義が **NDBCLUSTER** ストレージエンジンと互換性がある必要があります。MySQL 8.0 では、追加の回避策も必要となります。詳細は、[セクション23.1.7「NDB Cluster の既知の制限事項」](#) を参照してください。

A.10.2 NDB Cluster ノードはどのように相互に通信しますか。

クラスタのノードは、3種類の転送メカニズム (TCP/IP、SHM (共有メモリー)、および SCI (スケーラブルコヒーレントインタフェース)) を使用して通信できます。使用可能な場合、同じクラスタホスト上にあるノード間では SHM がデフォルトで使用されます。ただし、これは実験的とみなされます。SCI は、スケーラブルなマルチプロセッサシステムを構築するために使用される高速 (1 Gbps 以上) で高可用性のプロトコルであり、特殊なハードウェアおよびドライバが必要となります。NDB Cluster のトランスポートメカニズムとして SCI を使用する方法的詳細は、[セクション23.3.4「NDB Cluster での高速インターコネクタの使用」](#) を参照してください。

A.10.2 アービトレータとは何ですか。

クラスタ内の1つ以上のデータノードで障害が発生した場合、すべてのクラスタデータノードが相互に「参照」できるわけではありません。実際に、2つのセットのデータノードがネットワークのパーティション化で別々に分離されることがあります（「スプリットブレイン」シナリオとも呼ばれます）。各セットのデータノードがクラスタ全体のように動作しようとするため、このタイプの状況は好ましくありません。競合するデータノードのセットのいずれかを選択するために、アービトレータが必要となります。

少なくとも1つのノードグループのすべてのデータノードが存続した場合、クラスタの単一のサブセットが独自に機能するクラスタを形成できないため、ネットワークのパーティション化は問題とはなりません。本当の問題はすべてのノードが存続している単一のノードグループがない場合に発生し、その場合はネットワークのパーティション化（「スプリットブレイン」シナリオ）が発生する可能性があります。そしてアービトレータが必要となります。すべてのクラスタノードは、同じノードをアービトレータとして認識しますが、通常、これは管理サーバーです。ただし、クラスタ内の MySQL サーバーのいずれかを代わりにアービトレータとして動作するように構成できます。アービトレータは、クラスタノードの最初のセットがアクセスすることを受け入れ、残りのセットにシャットダウンするように指示します。アービトレータの選択は、MySQL サーバーおよび管理サーバーノードの **ArbitrationRank** 構成パラメータによって制御されます。**ArbitrationRank** 構成パラメータを使用してアービトレータの選択プロセスを制御することもできます。これらのパラメータについては、[セクション23.3.3.5「NDB Cluster 管理サーバーの定義」](#) を参照してください。

アービトレータの役割によって、指定されたホストに重い要求が課されることはないため、アービトレータのホストはこの目的のために特別に処理が速いマシン、または追加のメモリーがあるマシンである必要はありません。

A.10.3 NDB Cluster でサポートされているデータ型は何ですか。

NDB Cluster は、MySQL 空間拡張機能に関連付けられたデータ型を含め、通常の MySQL データ型をすべてサポートしますが、**NDB** ストレージエンジンは空間インデックスをサポートしていません。（空間インデックスは **MyISAM** によってのみサポートされます。詳細は、[セクション11.4「空間データ型」](#) を参照してください。）また、**NDB** テーブルとともに使用された場合、インデックスに関していくつかの違いがあります。

注記

「NDB Cluster ディスクデータ」テーブル (`TABLESPACE ... STORAGE DISK ENGINE=NDB` または `TABLESPACE ... STORAGE DISK ENGINE=NDBCLUSTER` で作成されたテーブル) には固定幅の行のみがあります。これは、(たとえば) `VARCHAR(255)` カラムが含まれている各ディスクデータテーブルレコードで、実際に格納される文字数に関係なく、255 文字 (テーブルに使用されている文字セットおよび照合順序に必要となります) の領域が必要となることを意味します。

これらの問題については、[セクション23.1.7「NDB Cluster の既知の制限事項」](#)を参照してください。

A.10.3 **NDB Cluster を起動および停止するにはどうすればよいですか。**

クラスタ内の各ノードを次の順序で個別に起動する必要があります。

1. `ndb_mgmd` コマンドを使用して、管理ノードを起動します。

`-f` オプションまたは `--config-file` オプションを指定して、構成ファイルのある場所を管理ノードに通知する必要があります。

2. `ndbd` コマンドを使用して、各データノードを起動します。

データノードが管理サーバーに接続する方法を認識するように、`-c` オプションまたは `--ndb-connectstring` オプションを指定して各データノードを起動する必要があります。

3. 任意の起動スクリプト (`mysqld_safe` など) を使用して各 MySQL サーバー (SQL ノード) を起動します。

各 MySQL サーバーは、`--ndbcluster` オプションおよび `--ndb-connectstring` オプションを指定して起動する必要があります。これらのオプションにより、`mysqld` は `NDBCLUSTER` ストレージエンジンのサポートおよび管理サーバーへの接続方法を有効にします。

影響を受けるノードがあるマシンのシステムシェルで、これらの各コマンドを実行する必要があります。(そのマシンを物理的にその場で操作する必要はありません。リモートログインシェルをこの目的に使用できます。) クラスタが実行されていることを確認するには、管理ノードが収容されているマシンで `NDB` 管理クライアント `ndb_mgm` を起動して、`SHOW` コマンドまたは `ALL STATUS` コマンドを発行します。

実行されているクラスタをシャットダウンするには、管理クライアントで `SHUTDOWN` コマンドを発行します。または、システムシェルに次のコマンドを入力できます。

```
shell> ndb_mgm -e "SHUTDOWN"
```

(この例の引用符はオプションです。 `-e` オプションの後ろのコマンド文字列にスペースがないためです。また、管理クライアントのほかのコマンドと同様に、`SHUTDOWN` コマンドでは大文字/小文字が区別されません。)

これらのコマンドのいずれかによって、`ndb_mgm`、`ndb_mgm`、および `ndbd` プロセスが正常に終了します。SQL ノードとして実行されている MySQL サーバーは、`mysqladmin shutdown` を使用して停止できます。

詳細は、[セクション23.5.1「NDB Cluster 管理クライアントのコマンド」](#) および [セクション23.2.6「NDB Cluster の安全なシャットダウンと再起動」](#)を参照してください。

MySQL Cluster Manager および NDB Cluster Auto-Installer には、NDB Cluster ノードの応答なし停止の開始を処理する追加の方法が用意されています。これらのツールの詳細は、[MySQL Cluster Manager 1.4.8 User Manual](#) および [セクション23.2.8「NDB Cluster Auto-Installer \(サポートされなくなりました\)」](#)を参照してください。

A.10.3 **NDB Cluster をシャットダウンすると、NDB Cluster データはどうなりますか。**

クラスタのデータノードによってメモリーに保持されていたデータがディスクに書き込まれ、次回クラスタが起動されたときにメモリーにリロードされます。

A.10.3 **NDB Cluster に複数の管理ノードを設定することをお勧めしますが。**

フェイルセーフとして役に立つことがあります。特定の時点でクラスタを制御しているのは 1 つの管理ノードのみですが、1 つの管理ノードをプライマリとして構成し、プライマリ管理ノードで障害が発生した場合に、1 つ以上の追加の管理ノードが引き継ぐようにすることができます。

NDB Cluster 管理ノードを構成する方法については、[セクション23.3.3「NDB Cluster 構成ファイル」](#)を参照してください。

A.10.34. つの NDB Cluster に異なる種類のハードウェアとオペレーティングシステムを混在させることはできますか。

はい。すべてのマシンおよびオペレーティングシステムが同じ「エンディアン」（すべてがビッグエンディアンまたはすべてがリトルエンディアン）であれば可能です。

異なる NDB Cluster リリースのソフトウェアを異なるノードで使用することもできます。ただし、このような使用はローリングアップグレード手順の一部としてのみサポートされています ([セクション23.5.5「NDB Cluster のローリング再起動の実行」](#)を参照)。

A.10.35. 単一のホスト上で 2 つのデータノードを実行できますか。2 つの SQL ノードは実行できますか。

はい。実行できます。複数のデータノードの場合は、各ノードで別のデータディレクトリを使用することをお勧めします (必須ではありません)。単一のマシン上で複数の SQL ノードを実行する場合は、`mysqld` の各インスタンスで別の TCP/IP ポートを使用する必要があります。

データノードと SQL ノードを同じホスト上で同時に実行することは可能ですが、`ndbd` または `ndbmt` プロセスが `mysqld` とのメモリーを競合する可能性があることに注意してください。

A.10.36. NDB Cluster でホスト名を使用できますか。

はい。クラスタのホストに DNS および DHCP を使用できます。ただし、アプリケーションが「ファイブナイン」可用性を要求する場合は、固定 (数値) IP アドレスを使用してください。クラスタホスト間通信は DNS、DHCP などのサービスに依存させると、潜在的な障害点が増えるためです。

A.10.37. NDB Cluster は IPv6 をサポートしていますか。

IPv6 は SQL ノード (MySQL サーバー) 間の接続でサポートされていますが、ほかのすべてのタイプの NDB Cluster ノード間の接続は IPv4 を使用する必要があります。

これは、実質的には NDB Cluster 間のレプリケーションに IPv6 を使用できるが、同じ NDB Cluster 内のノード間の接続には IPv4 を使用する必要があることを意味します。詳細は、[セクション23.6.3「NDB Cluster レプリケーションの既知の問題」](#)を参照してください。

A.10.38. 複数の MySQL サーバーを持つ NDB Cluster で MySQL ユーザーを処理するにはどうすればよいですか。

通常、MySQL ユーザーアカウントと特権は、同じ NDB Cluster にアクセスする異なる MySQL サーバー間で自動的に伝播されません。MySQL NDB Cluster は、`NDB_STORED_USER` 権限を使用した共有および同期されたユーザーおよび権限のサポートを提供します。詳細は、[セクション23.5.12「NDB_STORED_USER での分散 MySQL 権限」](#)を参照してください。この実装は NDB 8.0 の新機能であり、NDB 8.0 でサポートされなくなった以前のバージョンの NDB Cluster で採用されていた共有特権メカニズムと互換性がないことに注意してください。

A.10.39. SQL ノードの 1 つで障害が発生した場合に、クエリーの送信を続けるにはどうすればよいですか。

MySQL NDB Cluster では、SQL ノード間の自動フェイルオーバーは一切提供されません。アプリケーションは、SQL ノードの損失を処理し、それらの間でフェイルオーバーする準備ができていなければならない必要があります。

A.10.40. NDB Cluster をバックアップおよび復元するにはどうすればよいですか。

NDB 管理クライアントおよび `ndb_restore` プログラムで NDB Cluster のネイティブバックアップおよび復元機能を使用できます。[セクション23.5.8「NDB Cluster のオンラインバックアップ」](#) および [セクション23.4.23「ndb_restore — NDB Cluster バックアップの復元」](#)を参照してください。

このために `mysqldump` および MySQL サーバーで提供されている従来の機能を使用することもできます。詳細は、[セクション4.5.4「mysqldump — データベースバックアッププログラム」](#)を参照してください。

A.10.41. 「エンジェルプロセス」とは何ですか。

このプロセスは、データノードプロセスをモニターし、必要に応じて再起動を試みます。 `ndbd` を起動したあとに、システムでアクティブなプロセスのリストをチェックすると、その名前で実行されているプロセスが実際には 2 つあることを確認できます (簡潔にするために、出力では `ndb_mgmd` および `ndbd` を省略しました)。

```
shell> ./ndb_mgmd

shell> ps aux | grep ndb
me 23002 0.0 0.0 122948 3104 ? Ssl 14:14 0:00 ./ndb_mgmd
me 23025 0.0 0.0 5284 820 pts/2 S+ 14:14 0:00 grep ndb

shell> ./ndbd -c 127.0.0.1 --initial

shell> ps aux | grep ndb
me 23002 0.0 0.0 123080 3356 ? Ssl 14:14 0:00 ./ndb_mgmd
me 23096 0.0 0.0 35876 2036 ? Ss 14:14 0:00 ./ndbmt-d 127.0.0.1 --initial
me 23097 1.0 2.4 524116 91096 ? Sl 14:14 0:00 ./ndbmt-d 127.0.0.1 --initial
me 23168 0.0 0.0 5284 812 pts/2 R+ 14:15 0:00 grep ndb
```

メモリー使用率と CPU 使用率の両方について 0.0 を示す `ndbd` プロセスは、`angel` プロセスです (ただし、実際には非常に少量のプロセスを使用します)。このプロセスは、`ndbd` または `ndbmt-d` のメインプロセス (実際にデータを処理するプライマリデータノードプロセス) が実行されているかどうかのみをチェックします。許可されている場合 (たとえば、`StopOnError` 構成パラメータが `false` に設定されている場合)、`angel` プロセスはプライマリデータノードプロセスの再起動を試みます。

A.11 MySQL 8.0 FAQ: MySQL の中国語、日本語、および韓国語の文字セット

この一連のよくある質問は、CJK (中国語、日本語、韓国語) の問題に関する多くの問い合わせに対応している MySQL のサポートグループおよび開発グループの経験から得られたものです。

- A.11.1 MySQL で使用できる CJK 文字セットはどの文字セットですか。 4563
 - A.11.2 CJK 文字をテーブルに挿入しました。 `SELECT` でそれらが「？」文字として表示されるのはなぜですか。 4564
 - A.11.3 Big5 中国語文字セットを使用する場合は、どのような問題に注意すべきですか。 4566
 - A.11.4 日本語文字セットの変換が失敗するのはなぜですか。 4566
 - A.11.5 SJIS の `81CA` を `cp932` に変換する場合はどうすればよいですか。 4568
 - A.11.6 MySQL では円 (¥) 記号をどのように表しますか。 4568
 - A.11.7 MySQL で韓国語文字セットを使用する場合に注意する問題はありますか。 4568
 - A.11.8 なぜ「`Incorrect string value`」というエラーメッセージが表示されるのですか。 4568
 - A.11.9 Access、PHP または別の API を使用して、アプリケーションで GUI フロントエンドまたはブラウザに CJK 文字が正しく表示されないのはなぜですか。 4569
 - A.11.10 MySQL 8.0 にアップグレードしました。文字セットに関して、MySQL 4.0 の動作に戻すにはどうすればよいですか。 4569
 - A.11.11 CJK 文字での `LIKE` 検索および `FULLTEXT` 検索が失敗することがあるのはなぜですか。 4570
 - A.11.12 文字 `X` がすべての文字セットで使用可能であるかどうかを判別するにはどうすればよいですか。 4571
 - A.11.13 CJK 文字列が Unicode で間違っソートされるのはなぜですか。 (I) 4572
 - A.11.14 CJK 文字列が Unicode で間違っソートされるのはなぜですか。 (II) 4572
 - A.11.15 補助文字が MySQL で拒否されるのはなぜですか。 4572
 - A.11.16 「CJK」は「CJKV」である必要がありますか。 4572
 - A.11.17 MySQL では、CJK 文字をデータベース名およびテーブル名で使用できますか。 4572
 - A.11.18 MySQL マニュアルの中国語版、日本語版、および韓国語版はどこにありますか。 4572
 - A.11.19 MySQL での CJK および関連する問題についての支援はどこで受けられますか。 4572
- A.11.1 MySQL で使用できる CJK 文字セットはどの文字セットですか。

CJK 文字セットのリストは、MySQL のバージョンによって異なることがあります。たとえば、`gb18030` 文字セットは MySQL 5.7.4 より前はサポートされていません。ただし、`INFORMATION_SCHEMA.CHARACTER_SETS` 表のすべてのエントリの `DESCRIPTION` カラムには適用可能な言語の名前が表示されるため、次のクエリーを使用して Unicode 以外のすべての CJK 文字セットの最新のリストを取得できます。

```
mysql> SELECT CHARACTER_SET_NAME, DESCRIPTION
FROM INFORMATION_SCHEMA.CHARACTER_SETS
WHERE DESCRIPTION LIKE '%Chin%'
```

```

OR DESCRIPTION LIKE '%Japanese%'
OR DESCRIPTION LIKE '%Korean%'
ORDER BY CHARACTER_SET_NAME;
+-----+-----+
| CHARACTER_SET_NAME | DESCRIPTION |
+-----+-----+
| big5                | Big5 Traditional Chinese |
| cp932              | SJIS for Windows Japanese |
| eucjpms            | UJIS for Windows Japanese |
| euckr              | EUC-KR Korean |
| gb18030            | China National Standard GB18030 |
| gb2312             | GB2312 Simplified Chinese |
| gbk                | GBK Simplified Chinese |
| sjis               | Shift-JIS Japanese |
| ujis               | EUC-JP Japanese |
+-----+-----+

```

(詳細は [セクション26.4「INFORMATION_SCHEMA CHARACTER_SETS テーブル」](#) を参照してください。)

MySQL では、中国人民共和国で正式な GB (Guojia Biaozhun、国民標準または簡体字中国語) 文字セットの 3 つのバリエーションがサポートされています: [gb2312](#)、[gbk](#) および (MySQL 5.7.4 の時点) [gb18030](#)。

[gbk](#) は [gb2312](#) のスーパーセットであるため、ユーザーが [gbk](#) 文字を [gb2312](#) に挿入しようとする場合があります。ただし、最終的には、まれな中国語の文字を挿入しようとしたが、機能しません。(例は、[Bug #16072](#) を参照してください)。

ここでは、[gb2312](#) または [gbk](#) で正当な文字を明らかにして、正式なドキュメントへの参照を示します。[gb2312](#) または [gbk](#) のバグを報告する前に、次のリファレンスを確認してください:

- MySQL [gbk](#) 文字セットは実際には「Microsoft コードページ 936」です。これは、正式な [gbk](#) とは文字 [A1A4](#) (中黒)、[A1AA](#) (エムダツシユ)、[A6E0-A6F5](#)、および [A8BB-A8C0](#) が異なります。
- [gbk](#)/Unicode マッピングのリストについては、<http://www.unicode.org/Public/MAPPINGS/VENDORS/MICSFT/WINDOWS/CP936.TXT> を参照してください。

また、CJK 文字を Unicode 文字セットで格納することもできますが、使用可能な照合順序では文字が予期したとおりにソートされない場合があります:

- [utf8](#) および [ucs2](#) 文字セットでは、Unicode Basic Multilingual Plane (BMP) の文字がサポートされています。これらの文字には、[U+0000](#) と [U+FFFF](#) の間のコードポイント値があります。
- [utf8mb4](#)、[utf16](#)、[utf16le](#) および [utf32](#) の文字セットでは、BMP 文字と BMP の外部にある補助文字がサポートされています。補助文字には、[U+10000](#) と [U+10FFFF](#) の間のコードポイント値があります。

Unicode 文字セットに使用される照合順序によって、セット内の文字をソート (区別) する機能が決まります:

- Unicode 照合アルゴリズム (UCA) に基づく照合では、4.0.0 は BMP 文字のみを区別します。
- UCA 5.2.0 または 9.0.0 に基づく照合では、BMP と補助文字が区別されます。
- UCA 以外の照合順序では、すべての Unicode 文字が区別されない場合があります。たとえば、[utf8mb4](#) のデフォルトの照合は [utf8mb4_general_ci](#) で、BMP 文字のみが区別されます。

さらに、文字の区別は、特定の CJK 言語の表記規則に従って文字を順序付けすることと同じではありません。現在、MySQL には CJK 固有の UCA 照合順序である [gb18030_unicode_520_ci](#) のみがあります (Unicode 以外の [gb18030](#) 文字セットを使用する必要があります)。

Unicode 照合およびそれらの区別プロパティ (補助文字の照合プロパティを含む) の詳細は、[セクション 10.10.1「Unicode 文字セット」](#) を参照してください。

A.11.2.CJK 文字をテーブルに挿入しました。SELECT でそれらが「?」文字として表示されるのはなぜですか。

この問題は通常、MySQL の設定がアプリケーションプログラムまたはオペレーティングシステムの設定と一致しないことが原因です。これらのタイプの問題を修正するための一般的な手順を次に示します。

- 使用している MySQL のバージョンを確認します。

これを判別するには、`SELECT VERSION();` ステートメントを使用します。

- 意図した文字セットがデータベースで実際に使用されていることを確認します。

ユーザーは多くの場合、クライアントの文字セットはサーバーの文字セットまたは表示のために使用される文字セットと常に同じであると考えます。ただし、これらは両方とも間違っただけです。次のステートメントを使用して、`SHOW CREATE TABLE tablename` の結果を確認するか、またはさらに適切に確認できます:

```
SELECT character_set_name, collation_name
FROM information_schema.columns
WHERE table_schema = your_database_name
AND table_name = your_table_name
AND column_name = your_column_name;
```

- 正しく表示されない文字の 16 進値を判別します。

テーブル `table_name` のカラム `column_name` のこの情報を取得するには、次のクエリーを使用します。

```
SELECT HEX(column_name)
FROM table_name;
```

`3F` は `?` 文字のエンコードです。これは、`?` が実際にカラムに格納される文字であることを意味します。これは、ほとんどの場合、特定の文字をクライアントの文字セットからターゲットの文字セットに変換するときの問題のために発生します。

- Make では、ラウンドトリップが可能であることを確認します。 `literal` (または `_introducer hexadecimal-value`) を選択した場合、`literal` を result として取得しますが。

たとえば、日本語のカタカナ文字 `Pe` (ペ) はすべての CJK 文字セットに存在し、コードポイント値 (16 進コーディング) は `0x30da` です。この文字のラウンドトリップをテストするには、次のクエリーを使用します。

```
SELECT 'ペ' AS `ペ`; /* or SELECT _ucs2 0x30da; */
```

結果が `ペ` でもない場合、ラウンドトリップは失敗します。

そのような失敗に関するバグレポートの場合は、`SELECT HEX('ペ');` も試すように要請されることがあります。これにより、クライアントのエンコードが正しいかどうかを判断できます。

- 問題が MySQL 以外のブラウザまたはほかのアプリケーションの問題ではないことを確認します。

このタスクを実行するには、`mysql` クライアントプログラムを使用します。`mysql` で文字が正しく表示されるが、アプリケーションで文字が表示されない場合は、おそらくシステム設定が原因です。

設定を確認するには、`SHOW VARIABLES` ステートメントを使用します。このステートメントの出力は次のようになります:

```
mysql> SHOW VARIABLES LIKE 'char%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| character_set_client | utf8 |
| character_set_connection | utf8 |
| character_set_database | latin1 |
| character_set_filesystem | binary |
| character_set_results | utf8 |
| character_set_server | latin1 |
| character_set_system | utf8 |
| character_sets_dir | /usr/local/mysql/share/mysql/charsets/ |
```

これらは、West ([latin1](#) は西ヨーロッパ言語の文字セット) のサーバーに接続された国際指向クライアント ([utf8](#) Unicode の使用に注意) の一般的な文字セット設定です。

Latin よりも Unicode (通常、Unix での [utf8](#) バリエント、および Windows での [ucs2](#) バリエント) の方が望ましいですが、オペレーティングシステムのユーティリティで最適にサポートされる文字セットではないことがあります。多くの Windows ユーザーは、Microsoft の文字セット (日本語 Windows 用の [cp932](#) など) が適当であると判断します。

サーバー設定を制御できず、基礎となるコンピュータで使用される設定がわからない場合は、使用している国に共通の文字セット ([euckr](#) = Korea、[gb18030](#)、[gb2312](#) または [gbk](#) = People Republic of China; [big5](#) = Taiwan; [sjis](#)、[ujis](#)、[cp932](#)、または [eucjpms](#) = Japan; [ucs2](#) または [utf8](#) = anywhere) に変更してみてください。通常、クライアント、接続、および結果の設定のみを変更する必要があります。SET NAMES. ステートメントは、3 つすべてを一度に変更します。例:

```
SET NAMES 'big5';
```

設定が正しい場合は、[my.cnf](#) または [my.ini](#) を編集することによってそれを永続的なものにできます。たとえば、次のような行を追加できます。

```
[mysqld]
character-set-server=big5
[client]
default-character-set=big5
```

アプリケーションで使用されている API 構成の設定に問題があることもあります。詳細は、「GUI フロントエンドまたはブラウザで CJK 文字が正しく表示されないのはなぜですか」を参照してください。

A.11.3. Big5 中国語文字セットを使用する場合は、どのような問題に注意すべきですか。

MySQL は、香港および台湾 (中華民国) で一般的な Big5 文字セットをサポートしています。MySQL [big5](#) 文字セットは、実際には元の [big5](#) 文字セットと非常によく似た Microsoft コードページ 950 にあります。

[HKSCS](#) 拡張を追加する機能要求は申請されています。この拡張を必要とするユーザーの場合、Bug #13577 のための推奨パッチが役に立つことがあります。

A.11.4. 日本語文字セットの変換が失敗するのはなぜですか。

MySQL は、[sjis](#)、[ujis](#)、[cp932](#)、および [eucjpms](#) 文字セットと Unicode をサポートしています。一般的な二重文字セット間で変換を行うことです。たとえば、Unix のサーバー (通常は [sjis](#) または [ujis](#) が使用されます) と Windows のクライアント (通常は [cp932](#) が使用されます) を使用している場合があります。

次の変換テーブルでは、[ucs2](#) カラムはソースを表し、[sjis](#)、[cp932](#)、[ujis](#) および [eucjpms](#) カラムは宛先を表します。つまり、[CONVERT\(ucs2\)](#) を使用するか、値を含む [ucs2](#) カラムを [sjis](#)、[cp932](#)、[ujis](#) または [eucjpms](#) カラムに割り当てると、最後の 4 カラムは 16 進数の結果を提供します。

文字名	ucs2	sjis	cp932	ujis	eucjpms
BROKEN BAR (破断線)	00A6	3F	3F	8FA2C3	3F
FULLWIDTH BROKEN BAR (全角破断線)	FFE4	3F	FA55	3F	8FA2
YEN SIGN (円記号)	00A5	3F	3F	20	3F
FULLWIDTH YEN SIGN (全角円記号)	FFE5	818F	818F	A1EF	3F
TILDE (チルダ)	007E	7E	7E	7E	7E
OVERLINE (オーバーライン)	203E	3F	3F	20	3F

文字名	ucs2	sjis	cp932	ujis	eucjpms
HORIZONTAL BAR (水平線)	2015	815C	815C	A1BD	A1BD
EM DASH (エムダッシュ)	2014	3F	3F	3F	3F
REVERSE SOLIDUS (リバースソリダス)	005C	815F	5C	5C	5C
全角逆斜線	FF3C	3F	815F	3F	A1C0
WAVE DASH (波ダッシュ)	301C	8160	3F	A1C1	3F
FULLWIDTH TILDE (全角チルダ)	FF5E	3F	8160	3F	A1C1
DOUBLE VERTICAL LINE (双柱)	2016	8161	3F	A1C2	3F
PARALLEL TO (平行)	2225	3F	8161	3F	A1C2
MINUS SIGN (マイナス記号)	2212	817C	3F	A1DD	3F
FULLWIDTH HYPHEN-MINUS (全角ハイフンマイナス)	FF0D	3F	817C	3F	A1DD
CENT SIGN (セント記号)	00A2	8191	3F	A1F1	3F
FULLWIDTH CENT SIGN (全角セント記号)	FFE0	3F	8191	3F	A1F1
POUND SIGN (ポンド記号)	00A3	8192	3F	A1F2	3F
FULLWIDTH POUND SIGN (全角ポンド記号)	FFE1	3F	8192	3F	A1F2
NOT SIGN (否定記号)	00AC	81CA	3F	A2CC	3F
FULLWIDTH NOT SIGN (全角否定記号)	FFE2	3F	81CA	3F	A2CC

では、この表の次の部分について考えてみましょう。

	ucs2	sjis	cp932
NOT SIGN (否定記号)	00AC	81CA	3F
FULLWIDTH NOT SIGN (全角否定記号)	FFE2	3F	81CA

つまり、MySQL は **NOT SIGN** (Unicode **U+00AC**) を **sjis** コードポイント **0x81CA** および **cp932** コードポイント **3F** に変換します。(3F は疑問符 (「?」) です。これは、変換を実行できない場合に常に使用されるものです。)

A.11.5.SJIS の 81CA を cp932 に変換する場合はどうすればよいですか。

答えは「?」です。これにはデメリットがあり、多くのユーザーが「**「ルーズ」**」変換を希望するため、**sjis** の **81CA (NOT SIGN)** は **cp932** の **81CA (FULLWIDTH NOT SIGN)** になります。

A.11.6.MySQL では円 (¥) 記号をどのように表しますか。

一部のバージョンの日本語文字セット (**sjis** と **eur**) では **5C** が逆斜線 (\、バックスラッシュとも呼ばれる) として扱われ、他のバージョンでは円記号 (¥) として扱われるため、問題が発生します。

MySQL は JIS (Japanese Industrial Standards) 標準に記述されている 1 つのバージョンのみに従っています。MySQL では **5C** は常にリバースソリダス (\) です。

A.11.7.MySQL で韓国語文字セットを使用する場合に注意する問題はありますか。

理論的には、複数のバージョンの **euocr** (Extended Unix Code Korea) 文字セットがありますが、認識されている問題は 1 つだけです。コードポイント **0x5c** が **ウオン記号 (₩)** である EUC-KR の「KS-Roman」バリエーションではなく、コードポイント **0x5c** がリバースソリダス (つまり、\) である EUC-KR の「ASCII」バリエーションが使用されています。これは、Unicode の **U+20A9** を **euocr** に変換できないことを意味します。

```
mysql> SELECT
  CONVERT(₩ USING euocr) AS euocr,
  HEX(CONVERT(₩ USING euocr)) AS hexeuocr;
+-----+-----+
| euocr | hexeuocr |
+-----+-----+
| ?    | 3F      |
+-----+-----+
```

A.11.8.なぜ「Incorrect string value」というエラーメッセージが表示されるのですか。

この問題を確認するには、Unicode (**ucs2**) カラムと中国語 (**gb2312**) カラムを含むテーブルを作成します。

```
mysql> CREATE TABLE ch
  (ucs2 CHAR(3) CHARACTER SET ucs2,
  gb2312 CHAR(3) CHARACTER SET gb2312);
```

非厳密 SQL モードでは、両方のカラムにまれな文字 **洵** を配置してみてください。

```
mysql> SET sql_mode = "";
mysql> INSERT INTO ch VALUES ('A洵B','A洵B');
Query OK, 1 row affected, 1 warning (0.00 sec)
```

INSERT によって警告が生成されます。次のステートメントを使用してその内容を確認します。

```
mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Warning
Code: 1366
Message: Incorrect string value: '\xE6\xB1\x8CB' for column 'gb2312' at row 1
```

gb2312 カラムのみに関する警告でした。

```
mysql> SELECT ucs2,HEX(ucs2),gb2312,HEX(gb2312) FROM ch;
+-----+-----+-----+-----+
| ucs2 | HEX(ucs2) | gb2312 | HEX(gb2312) |
+-----+-----+-----+-----+
| A洵B | 00416C4C0042 | A?B    | 413F42      |
+-----+-----+-----+-----+
```

いくつかのことを説明する必要があります。

1. 前述のように、**洵**文字は **gb2312** 文字セットに含まれていません。
2. 古いバージョンの MySQL を使用している場合は、別のメッセージが表示されることがあります。
3. MySQL は厳密な SQL モードを使用するように設定されていないため、エラーではなく警告が発生します。非厳密モードでは、MySQL は放棄するのではなく、最良の適合を得るために実行しようとします。

厳密な SQL モードでは、「不正な文字列値」メッセージは警告ではなくエラーとして発生し、`INSERT` は失敗します。

A.11.9 Access、PHP または別の API を使用して、アプリケーションで GUI フロントエンドまたはブラウザに CJK 文字が正しく表示されないのはなぜですか。

`mysql` クライアントを使用してサーバーへの直接接続を取得し、そこで同じクエリーを試行します。`mysql` が正しく応答する場合、問題はアプリケーションインタフェースの初期化が必要である可能性があります。`mysql` で `SHOW VARIABLES LIKE 'char%'`; ステートメントを使用して、使用される文字セットを確認します。Access を使用している場合、コネクタ/ODBC を使用して接続している可能性があります。この場合は、[Configuring Connector/ODBC](#)を確認してください。たとえば、`big5` を使用する場合は、`SET NAMES 'big5'`と入力します。(この場合、`;`文字は必要ありません。) ASP を使用している場合は、コードに `SET NAMES` を追加する必要があることがあります。過去に作成された例を次に示します。

```
<%
Session.CodePage=0
Dim strConnection
Dim Conn
strConnection="driver={MySQL ODBC 3.51 Driver};server=server;uid=username;" \
    & "pwd=password;database=database;stmt=SET NAMES 'big5';"
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open strConnection
%>
```

ほぼ同様に、Connector/NET で `latin1` 以外の文字セットを使用している場合は、接続文字列に文字セットを指定する必要があります。詳細は、[Connector/NET Connections](#)を参照してください。

PHP を使用している場合は、次のコードを試してください。

```
<?php
$link = new mysqli($host, $usr, $pwd, $db);

if( mysqli_connect_errno() )
{
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$link->query("SET NAMES 'utf8'");
?>
```

この場合、`SET NAMES` を使用して `character_set_client`、`character_set_connection` および `character_set_results` を変更しました。

PHP アプリケーションで頻繁に発生する別の問題は、ブラウザによって行われる想定に関連しています。問題を修正するために `<meta>` タグの追加または変更で十分な場合があります: たとえば、ユーザーエージェントがページコンテンツを `UTF-8` として解釈するようにするには、HTML ページの `<head>` セクションに `<meta http-equiv="Content-Type" content="text/html; charset=utf-8">` を含めます。

Connector/J を使用している場合は、[Using Character Sets and Unicode](#)を参照してください。

A.11.10 MySQL 8.0 にアップグレードしました。文字セットに関して、MySQL 4.0 の動作に戻すにはどうすればよいですか。

MySQL バージョン 4.0 では、サーバーおよびクライアントの両方のための単一の「グローバル」文字セットがあり、使用する文字の決定はサーバー管理者が行なっていました。これは MySQL バージョン 4.1 以降で変更されました。現在行われるのは、[セクション10.4「接続文字セットおよび照合順序」](#)に説明されているように、「ハンドシェイク」です。

クライアントが接続するときに、使用する文字セットの名前をサーバーに送信します。サーバーはこの名前を使用して、`character_set_client`、`character_set_results`、および `character_set_connection` システム変数を設定します。実際には、サーバーは文字セット名を使用して `SET NAMES` 操作を実行します。

このことの影響は、`--character-set-server=utf8` を指定して `mysqld` を開始することによって、クライアントの文字セットを制御できないことです。ただし、アジアの顧客の中には、MySQL 4.0 の動作を好むものもありま

す。この動作を維持できるように、`--skip-character-set-client-handshake` を使用してオフにできる `mysqld` スイッチ `--character-set-client-handshake` が追加されました。`--skip-character-set-client-handshake` を使用して `mysqld` を起動すると、クライアントは接続時に、使用する文字セットの名前をサーバーに送信します。ただし、サーバーはクライアントからのこのリクエストを無視。

たとえば、お気に入りのサーバー文字セットが `latin1` であるとします。クライアントのオペレーティングシステムでサポートされている文字セットであるため、クライアントが `utf8` を使用しているとします。デフォルトの文字セットとして `latin1` を使用してサーバーを起動します:

```
mysqld --character-set-server=latin1
```

そのあと、クライアントをデフォルトの文字セット `utf8` で起動します。

```
mysql --default-character-set=utf8
```

結果の設定は、`SHOW VARIABLES` の出力を表示することで確認できます:

```
mysql> SHOW VARIABLES LIKE 'char%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| character_set_client | utf8 |
| character_set_connection | utf8 |
| character_set_database | latin1 |
| character_set_filesystem | binary |
| character_set_results | utf8 |
| character_set_server | latin1 |
| character_set_system | utf8 |
| character_sets_dir | /usr/local/mysql/share/mysqlCharsets/ |
+-----+-----+
```

次に、クライアントを停止し、`mysqldadmin` を使用してサーバーを停止します。今度はハンドシェイクをスキップするように通知して、サーバーをふたたび起動します。

```
mysqld --character-set-server=utf8 --skip-character-set-client-handshake
```

`utf8` をデフォルトの文字セットとして再度使用してクライアントを起動し、結果の設定を表示します:

```
mysql> SHOW VARIABLES LIKE 'char%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| character_set_client | latin1 |
| character_set_connection | latin1 |
| character_set_database | latin1 |
| character_set_filesystem | binary |
| character_set_results | latin1 |
| character_set_server | latin1 |
| character_set_system | utf8 |
| character_sets_dir | /usr/local/mysql/share/mysqlCharsets/ |
+-----+-----+
```

`SHOW VARIABLES` とは異なる結果を比較することでわかるように、`--skip-character-set-client-handshake` オプションが使用されている場合、サーバーはクライアントの初期設定を無視します。

A.11.1 CJK 文字での `LIKE` 検索および `FULLTEXT` 検索が失敗することがあるのはなぜですか。

`LIKE` 検索では、`BINARY` や `BLOB` などのバイナリ文字列カラム型に非常に単純な問題があります: 文字の終わりを知っている必要があります。マルチバイト文字セットでは、各文字のオクテット長が異なることがあります。たとえば、`utf8` の場合、次に示すように `A` は 1 バイトを必要としますが、`べ` は 3 バイトを必要とします。

```
+-----+-----+
| OCTET_LENGTH(_utf8 'A') | OCTET_LENGTH(_utf8 'べ') |
+-----+-----+
| 1 | 3 |
+-----+-----+
```


文字列の最初の文字がどこで終わるかわからない場合は、2番目の文字がどこで始まるかわからないため、`LIKE 'A%'`などの非常に単純な検索も失敗します。解決策は、適切なCJK文字セットを持つように定義された非バイナリ文字列カラム型を使用することです。次に例を示します: `mysql TEXT CHARACTER SET sjis`。または、比較する前にCJK文字セットに変換してください。

これは、MySQL が存在しない文字のエンコーディングを許可できない理由の1つです。不正な入力の拒否に厳密でない場合は、文字の終わりを知る方法はありません。

`FULLTEXT` の検索では、単語の開始位置と終了位置を知る必要があります。西欧言語では、ほとんど(すべてではない)が識別しやすい単語境界: スペース文字を使用するため、これはほとんど問題ではありません。ただし、通常、アジアの言語ではこれは異なります。独自の判断で不完全な方法を使用して、すべての漢字が単語を表すと想定したり、(日本語の場合) 文法的な終わりに従ったカタカナからひらがなへの変化に応じるようにしたりすることもできます。ただし、唯一の確実な解決策では、包括的な単語のリストが必要となります。これは、サポートされる各アジア言語のサーバーに辞書を含める必要があることを意味します。これは簡単に言って実現可能ではありません。

A.11.1 文字 X がすべての文字セットで使用可能かどうかを判別するにはどうすればよいですか。

簡体字中国語および半角ではない基本的な日本語のかな文字のほとんどは、すべてのCJK文字セットで表示されます。次のストアードプロシージャは、`UCS-2 Unicode` 文字を受け入れて他の文字セットに変換し、結果を16進数で表示します。

```
DELIMITER //

CREATE PROCEDURE p_convert(ucs2_char CHAR(1) CHARACTER SET ucs2)
BEGIN
CREATE TABLE tj
(ucs2 CHAR(1) character set ucs2,
 utf8 CHAR(1) character set utf8,
 big5 CHAR(1) character set big5,
 cp932 CHAR(1) character set cp932,
 eucjpms CHAR(1) character set eucjpms,
 euckr CHAR(1) character set euckr,
 gb2312 CHAR(1) character set gb2312,
 gbk CHAR(1) character set gbk,
 sjis CHAR(1) character set sjis,
 ujis CHAR(1) character set ujis);

INSERT INTO tj (ucs2) VALUES (ucs2_char);

UPDATE tj SET utf8=ucs2,
 big5=ucs2,
 cp932=ucs2,
 eucjpms=ucs2,
 euckr=ucs2,
 gb2312=ucs2,
 gbk=ucs2,
 sjis=ucs2,
 ujis=ucs2;

/* If there are conversion problems, UPDATE produces warnings. */

SELECT hex(ucs2) AS ucs2,
 hex(utf8) AS utf8,
 hex(big5) AS big5,
 hex(cp932) AS cp932,
 hex(eucjpms) AS eucjpms,
 hex(euckr) AS euckr,
 hex(gb2312) AS gb2312,
 hex(gbk) AS gbk,
 hex(sjis) AS sjis,
 hex(ujis) AS ujis
FROM tj;

DROP TABLE tj;

END//
```

```
DELIMITER ;
```

入力には、単一の `ucs2` 文字を使用することも、その文字のコード値 (16 進表記) を使用することもできます。たとえば、`ucs2` エンコーディングおよび名前 (<http://www.unicode.org/Public/UNIDATA/UnicodeData.txt>) の Unicode リストから、Katakana 文字 `Pe` がすべての CJK 文字セットに表示され、そのコード値が `X'30DA'` であることがわかります。この値を `p_convert()` の引数として使用すると、次のような結果となります。

```
mysql> CALL p_convert(X'30DA');
+-----+-----+-----+-----+-----+-----+-----+-----+
| ucs2 | utf8 | big5 | cp932 | eucjpms | euckr | gb2312 | gbk | sjis | ujis |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 30DA | E3839A | C772 | 8379 | A5DA | ABDA | A5DA | A5DA | 8379 | A5DA |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

いずれのカラム値も `3F` (疑問符文字、`?`) ではないため、すべての変換が機能したことがわかります。

A.11.13 CJK 文字列が Unicode で間違っソートされるのはなぜですか。 (I)

古い MySQL バージョンで発生した CJK ソートの問題は、`utf8mb4` 文字セットおよび `utf8mb4_ja_0900_as_cs` 照合順序を使用して、MySQL 8.0 の時点で解決できます。

A.11.14 CJK 文字列が Unicode で間違っソートされるのはなぜですか。 (II)

古い MySQL バージョンで発生した CJK ソートの問題は、`utf8mb4` 文字セットおよび `utf8mb4_ja_0900_as_cs` 照合順序を使用して、MySQL 8.0 の時点で解決できます。

A.11.15 補助文字が MySQL で拒否されるのはなぜですか。

補助文字は Unicode 基本多言語面 / 平面 0 の外部にあります。BMP 文字には、`U+0000` と `U+FFFF` の間のコードポイント値があります。補助文字には、`U+10000` と `U+10FFFF` の間のコードポイント値があります。

補助文字を格納するには、それらを許可する文字セットを使用する必要があります:

- `utf8` および `ucs2` 文字セットでは BMP 文字のみがサポートされます。

`utf8` 文字セットでは、最大 3 バイトを超える UTF-8 文字のみが許可されます。このことが Bug #12600 など報告され、「バグではありません」として拒否されました。`utf8` では、認識できないバイトが検出された場合、MySQL は入力文字列を切り捨てる必要があります。それ以外の場合、不正なマルチバイト文字の長さは不明です。

考えられる回避策の 1 つは、`utf8` のかわりに `ucs2` を使用することです。この場合、「bad」文字は疑問符に変更されます。ただし、切捨てるは行われません。妥当性チェックが行われない `BLOB` または `BINARY` にデータ型を変更することもできます。

- `utf8mb4`, `utf16`, `utf16le` および `utf32` 文字セットでは、BMP 文字と BMP 外の補助文字がサポートされています。

A.11.16 「CJK」は「CJKV」である必要がありますか。

いいえ。用語「CJKV」(Chinese Japanese Korean Vietnamese) は漢字 (もともとは中国語) が含まれているベトナム文字セットを指しています。MySQL は、西部文字を含む最新のベトナム語スクリプトをサポートしていますが、ハン文字を使用した古いベトナム語スクリプトはサポートしていません。

MySQL 5.6 の時点では、[セクション 10.10.1 「Unicode 文字セット」](#) で説明しているように、Unicode 文字セットにベトナム語の照合順序があります。

A.11.17 MySQL では、CJK 文字をデータベース名およびテーブル名で使用できますか。

はい。

A.11.18 MySQL マニュアルの中国語版、日本語版、および韓国語版はどこにありますか。

MySQL 5.6 マニュアルの日本語版は、<https://dev.mysql.com/doc/> からダウンロードできます。

A.11.19 MySQL での CJK および関連する問題についての支援はどこで受けることができますか。

次のリソースを利用できます。

- MySQL ユーザーグループのリストは、<https://wikis.oracle.com/display/mysql/List+of+MySQL+User+Groups>にあります。
- 文字セットの問題に関連する機能要求については、<http://tinyurl.com/y6xcuf> を参照してください。
- MySQL 「[文字セット](#)、[照合順序](#)、[Unicode フォーラム](#)」にアクセスします。<http://forums.mysql.com/> には外国語フォーラムも用意されています。

A.12 MySQL 8.0 FAQ: コネクタおよび API

MySQL Connector およびその他の API に関する一般的な質問、問題、および回答については、マニュアルの次の部分を参照してください。

- [Using C API Features](#)
- [Connector/ODBC Notes and Tips](#)
- [Connector/NET Programming](#)
- [MySQL Connector/J 8.0 Developer Guide](#)

A.13 MySQL 8.0 FAQ : C API、libmysql

MySQL C API および libmysql に関するよくある質問。

A.13.1 「MySQL ネイティブ C API」とは一般的なメリットとユースケースとは何ですか。	4573
A.13.2 どのバージョンの libmysql を使用すればよいですか。	4573
A.13.3 「NoSQL」 X DevAPI を使用する場合はどうなりますか。	4573
A.13.4 libmysql をダウンロードするにはどうすればよいですか。	4573
A.13.5 ドキュメントはどこにありますか。	4573
A.13.6 バグはどのようにしてレポートするのですか。	4573
A.13.7 ライブラリを自分でコンパイルできますか。	4574

A.13.1. 「MySQL ネイティブ C API」とは一般的なメリットとユースケースとは何ですか。

libmysql は、C アプリケーションで MySQL データベースサーバーに接続するために使用できる C ベースの API です。ODBC、Perl DBI、Python DB API などの標準データベース API のドライバの基盤としても使用されます。

A.13.2. どのバージョンの libmysql を使用すればよいですか。

MySQL 8.0, 5.7, 5.6 および 5.5 の場合は、libmysql 8.0 を推奨します。

A.13.3. 「NoSQL」 X DevAPI を使用する場合はどうなりますか。

C 言語および X DevApi Document Store for MySQL 8.0 の場合は、MySQL Connector/C++ をお勧めします。Connector/C++ 8.0 には互換性のある C ヘッダーがあります。(これは、MySQL 5.7 以前には適用されません。)

A.13.4. libmysql をダウンロードするにはどうすればよいですか。

- Linux : Client Utilities Package は、「[MySQL コミュニティサーバー](#)」のダウンロードページから入手できます。
- Repos : Client Utilities Package は、[Yum](#)、[APT](#)、「[SuSE リポジトリ](#)」から入手できます。
- Windows : Client Utilities Package は、「[Windows インストーラ](#)」から入手できます。

A.13.5. ドキュメントはどこにありますか。

[MySQL 8.0 C API Developer Guide](#)を参照してください。

A.13.6. バグはどのようにしてレポートするのですか。

バグまたは不整合にお気づきの場合は、[バグデータベース](#)で報告してください。次に示すように、C API クライアントを選択します。

A.13.7.ライブラリを自分でコンパイルできますか。

はい、libmysqlclient ソースコードをダウンロードして、自分でコンパイルできます。次に例を示します：

```
$ git clone --depth 1 https://github.com/mysql/mysql-server
$ cd mysql-server
$ mkdir build
$ cd build
$ cmake .. -GNinja -DDOWNLOAD_BOOST=1 \
  -DWITH_BOOST=/tmp -DCMAKE_BUILD_TYPE=Release -DWITHOUT_SERVER=ON \
  -DWITH_SSL=system
$ ninja libmysqlclient.a
$ ls -la archive_output_directory/libmysqlclient.a
-rw-rw-r-- 1 kg kg 8,5M wrz 5 04:57 archive_output_directory/libmysqlclient.a
```

注記

この例では、make ではなく <https://ninja-build.org/> をビルドシステムとして使用します。

A.14 MySQL 8.0 FAQ: レプリケーション

次のセクションでは、MySQL レプリケーションに関するよくある質問に回答しています。

A.14.1 レプリカは常にソースに接続する必要がありますか。	4574
A.14.2 レプリケーションを有効にするには、ソースとレプリカでネットワーキングを有効にする必要がありますか。	4574
A.14.3 レプリカがソースと比較してどの程度遅れているかを知るにはどうすればよいですか。つまり、レプリカによってレプリケートされた最後のステートメントの日付を知るにはどうすればよいですか。	4575
A.14.4 レプリカがキャッチアップされるまでソースで更新を強制的にブロックするにはどうすればよいですか。	4575
A.14.5 二方向レプリケーションを設定する場合に注意する問題がありますか。	4575
A.14.6 レプリケーションを使用してシステムのパフォーマンスを改善するにはどうすればよいですか。	4575
A.14.7 パフォーマンスがより高いレプリケーションを使用するには、アプリケーションのクライアントコードを準備するときに何を行えばよいですか。	4576
A.14.8 MySQL レプリケーションによってシステムのパフォーマンスが向上するのは、どのような場合でどの程度向上しますか。	4576
A.14.9 冗長性または高可用性を持たせるには、レプリケーションをどのように使用すればよいですか。	4577
A.14.10 レプリケーションソースサーバーがステートメントベースまたは行ベースのバイナリロギング形式を使用しているかどうかを確認するにはどうすればよいですか。	4577
A.14.11 行ベースのレプリケーションを使用するようレプリカに指示するにはどうすればよいですか。	4577
A.14.12 GRANT および REVOKE ステートメントがレプリカマシンにレプリケートされないようにするにはどうすればよいですか。	4577
A.14.13 レプリケーションは混合オペレーティングシステムで機能しますか (たとえば、ソースは Linux 上で実行され、レプリカは macOS と Windows 上で実行されます)。	4577
A.14.14 混在ハードウェアアーキテクチャでレプリケーションは機能しますか (たとえば、ソースは 64 ビットマシン上で実行され、レプリカは 32 ビットマシン上で実行されます)。	4577

A.14.1.レプリカは常にソースに接続する必要がありますか。

いいえ、その必要はありません。レプリカは、数時間または数日間停止するか切断されたままにしてから、再接続して更新に追いつけます。たとえば、リンクが散發的にのみ、短時間にわたって稼働しているダイアルアップリンクを介して、ソース/レプリカ関係を設定できます。これは、特別な対策をとらないかぎり、いつでもレプリカがソースと同期していることが保証されないことを意味します。

切断されたレプリカに対してキャッチアップが発生するようにするには、まだレプリカにレプリケートされていない情報を含むバイナリログファイルをソースから削除しないでください。非同期レプリケーションは、レプリカが最後にイベントを読み取った時点からバイナリログの読み取りを続行できる場合にのみ機能します。

A.14.2.レプリケーションを有効にするには、ソースとレプリカでネットワーキングを有効にする必要がありますか。

はい、ソースおよびレプリカでネットワーキングを有効にする必要があります。ネットワーキングが有効になっていない場合、レプリカはソースに接続してバイナリログを転送できません。どちらのサーバーの構成ファイルでも、`skip_networking` システム変数が有効になっていないことを確認します。

- A.14.3.レプリカがソースと比較してどの程度遅れているかを知るにはどうすればよいですか。つまり、レプリカによってレプリケートされた最後のステートメントの日付を知るにはどうすればよいですか。

`SHOW REPLICA | SLAVE STATUS` からの出力の `Seconds_Behind_Master` カラムを確認します。 [セクション 17.1.7.1「レプリケーションステータスの確認」](#) を参照してください。

レプリケーション SQL スレッドは、ソースから読み取られたイベントを実行すると、独自の時間をイベントタイムスタンプに変更します。(これにより、`TIMESTAMP` が正常にレプリケートされます。) `SHOW PROCESSLIST` の出力の `Time` カラムでは、レプリケーション SQL スレッドに表示される秒数は、最後にレプリケートされたイベントのタイムスタンプとレプリカマシンのリアルタイムの間の秒数です。これを使用して、最後にレプリケートされたイベントの日付を判別できます。レプリカがソースから 1 時間切断されてから再接続すると、`SHOW PROCESSLIST` のレプリケーション SQL スレッドに 3600 などの大きな `Time` 値がすぐに表示される場合があります。これは、レプリカが 1 時間経過したステートメントを実行しているためです。 [セクション 17.2.3「レプリケーションスレッド」](#) を参照してください。

- A.14.4.レプリカがキャッチアップされるまでソースで更新を強制的にブロックするにはどうすればよいですか。

次の手順を使用します。

1. ソースで、次のステートメントを実行します:

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SHOW MASTER STATUS;
```

`SHOW` ステートメントの出力から、レプリケーション座標 (現在のバイナリログファイル名および位置) を記録します。

2. レプリカで、次のステートメントを発行します。ここで、`MASTER_POS_WAIT()` 関数の引数は、前のステップで取得したレプリケーション座標値です:

```
mysql> SELECT MASTER_POS_WAIT('log_name', log_pos);
```

`SELECT` ステートメントは、レプリカが指定されたログファイルおよび位置に達するまでブロックされます。その時点で、レプリカはソースと同期しており、ステートメントは戻ります。

3. ソースで次のステートメントを発行して、ソースが更新の処理を再度開始できるようにします:

```
mysql> UNLOCK TABLES;
```

- A.14.5.二方向レプリケーションを設定する場合に注意する問題はありますか。

現在、MySQL レプリケーションでは、分散 (クロスサーバー) 更新の原子性を保証するために、ソースとレプリカ間のロックプロトコルはサポートされていません。つまり、クライアント A が共同ソース 1 を更新し、その間にクライアント B が共同ソース 2 に伝播する前に、クライアント A の更新が共同ソース 1 とは異なる方法で動作するように共同ソース 2 を更新できます。したがって、クライアント A の更新によって共同ソース 2 が作成されると、共同ソース 2 からのすべての更新も伝播された後でも、共同ソース 1 とは異なるテーブルが生成されます。これは、どのような順序でも更新が安全に行われるという確証がある場合、またはクライアントコードで更新順序の間違いに対処できる場合を除いて、2 つのサーバーを二方向レプリケーション関係にするべきではないことを意味します。

二方向レプリケーションでは、実際には、更新に関してパフォーマンスはそれほど向上しません。各サーバーは、単一のサーバーで更新を行うときと同量の更新を行う必要があります。唯一の違いは、別のサーバーで発生した更新があるレプリケーションスレッドでシリアライズされるため、ロックの競合が少し少なくなることです。この利点さえも、ネットワーク遅延によって相殺されてしまうことがあります。

- A.14.6.レプリケーションを使用してシステムのパフォーマンスを改善するにはどうすればよいですか。

1 つのサーバをソースとして設定し、すべての書き込みをそのサーバに指示します。次に、予算およびラックスペースがある数のレプリカを構成し、ソースとレプリカ間で読取りを分散します。レプリカは、`--skip-innodb` オプションを使用して開始し、`low_priority_updates` システム変数を有効にし、`delay_key_write` システム変数を `ALL` に設定してレプリカの端の速度を向上させることもできます。この場合、レプリカは `InnoDB`

テーブルではなく非トランザクション **MyISAM** テーブルを使用して、トランザクションのオーバーヘッドをなくすことで高速になります。

- A.14.7 パフォーマンスがより高いレプリケーションを使用するには、アプリケーションのクライアントコードを準備するとき何を行えばよいですか。

スケールアウトソリューションとしてレプリケーションを使用するためのガイドについては、[セクション 17.4.5 「スケールアウトのためにレプリケーションを使用する」](#)を参照してください。

- A.14.8 MySQL レプリケーションによってシステムのパフォーマンスが向上するのは、どのような場合でどの程度向上しますか。

MySQL レプリケーションは、読み取りが頻繁に行われ、書き込みはそれほどないシステムにもっとも適しています。理論上、単一ソース/複数レプリケーション設定を使用すると、ネットワーク帯域幅が不足するか、ソースが処理できない時点まで更新負荷が増大するまで、レプリカを追加することでシステムをスケールアップできます。

追加されたメリットが平準化を開始する前に使用できるレプリカの数、およびサイトのパフォーマンスを向上させるためには、クエリーパターンを把握し、典型的なソースと典型的なレプリカでの読み取りおよび書き込みのスループット間の関係をベンチマークして経験的に判断する必要があります。ここでは、架空のシステムのレプリケーションの構成を単純に計算した例を示します。reads および writes は、1 秒当たりの読み取りおよび書き込みの数をそれぞれ示しています。

システムのロードは 10% の書き込みと 90% の読み取りで構成されていて、ベンチマークによって reads が $1200 - 2 * \text{writes}$ であることが判別されたとします。つまり、書き込みがない場合、システムは毎秒 1,200 回の読み取りを実行できます。書き込みの平均は読み取りの平均の 2 倍の時間がかかり、この関係はリニア (直線的) です。ソースと各レプリカの容量が同じで、ソースと N のレプリカが 1 つあるとします。次に、各サーバー (ソースまたはレプリカ) について次のことを行います:

$$\text{reads} = 1200 - 2 * \text{writes}$$

$$\text{reads} = 9 * \text{writes} / (N + 1) \text{ (読み取りは分割されますが、書き込みはすべてのレプリカにレプリケートされます)}$$

$$9 * \text{writes} / (N + 1) + 2 * \text{writes} = 1200$$

$$\text{writes} = 1200 / (2 + 9/(N + 1))$$

最後の方程式は、最大可能読み取り速度が 1,200/秒で、書き込みあたり 9 読み取りの比率が指定された場合の、 N レプリカの最大書き込み数を示します。

この分析によって、次の結論が導かれます。

- $N = 0$ (レプリケーションがないことを意味します) の場合、システムは毎秒 $1200/11 = 109$ 回の書き込みを処理できます。
- $N = 1$ の場合、毎秒最大 184 回の書き込みを実行できます。
- $N = 8$ の場合、毎秒最大 400 回の書き込みを実行できます。
- $N = 17$ の場合、毎秒最大 480 回の書き込みを実行できます。
- N が無限に近づくと (予算も負の無限大になり)、毎秒 600 回の書き込みに近くなり、システムスループットは 5.5 倍になります。ただし、8 サーバーのみでも 4 倍近くに増えます。

これらの計算では、無限のネットワーク帯域幅が想定され、システムで重要となる可能性のあるその他のいくつかの要因は無視されます。多くの場合、 N レプリカを追加した場合、システムで何が起るかを正確に予測するような計算を実行できないことがあります。ただし、次の質問に答えると、レプリケーションによってシステムのパフォーマンスが向上する可能性があるかどうかとその程度を判断するのに役立ちます:

- システムの読み取りと書き込みの比率はどれくらいですか。
- 読み取りを減らした場合、単一のサーバーで書き込みのロードをどれくらい処理できますか。
- ネットワーク上で使用可能な帯域幅があるレプリカはいくつありますか。

A.14.9 冗長性または高可用性を持たせるには、レプリケーションをどのように使用すればよいですか。

冗長性の実装方法は、アプリケーションおよび状況によってまったく異なります。高可用性ソリューション（自動フェイルオーバーを使用）では、アクティブな監視と、元の MySQL サーバーからレプリカへのフェイルオーバーサポートを提供するカスタムスクリプトまたはサードパーティツールが必要です。

プロセスを手動で処理するには、新しいサーバーと通信するようにアプリケーションを変更するか、MySQL サーバーの DNS を障害が発生したサーバーから新しいサーバーに調整して、障害が発生したソースから事前構成済レプリカに切り替えることができます。

詳細およびソリューションの例については、[セクション 17.4.8 「フェイルオーバー中のソースの切替え」](#)を参照してください。

A.14.10 レプリケーションソースサーバーがステートメントベースまたは行ベースのバイナリロギング形式を使用しているかどうかを確認するにはどうすればよいですか。

`binlog_format` システム変数の値を確認します。

```
mysql> SHOW VARIABLES LIKE 'binlog_format';
```

表示される値は、常に `STATEMENT`、`ROW` または `MIXED` のいずれかです。`MIXED` モードでは、ステートメントベースのロギングがデフォルトで使用されますが、レプリケーションは安全でないステートメントなどの特定の条件下で行ベースのロギングに自動的に切り替わります。これが発生する可能性がある状況の詳細は、[セクション 5.4.4.3 「混合形式のバイナリロギング形式」](#)を参照してください。

A.14.11 行ベースのレプリケーションを使用するようレプリカに指示するにはどうすればよいですか。

レプリカは、使用するフォーマットを自動的に認識します。

A.14.12 `GRANT` および `REVOKE` ステートメントがレプリカマシンにレプリケートされないようにするにはどうすればよいですか。

`--replicate-wild-ignore-table=mysql.%` オプションを指定してサーバーを起動し、`mysql` データベースのテーブルのレプリケーションが無視されるようにします。

A.14.13 レプリケーションは混合オペレーティングシステムで機能しますか（たとえば、ソースは Linux 上で実行され、レプリカは macOS と Windows 上で実行されます）。

はい。

A.14.14 混在ハードウェアアーキテクチャでレプリケーションは機能しますか（たとえば、ソースは 64 ビットマシン上で実行され、レプリカは 32 ビットマシン上で実行されます）。

はい。

A.15 MySQL 8.0 FAQ: MySQL Enterprise Thread Pool

A.15.1 スレッドプールとは何ですか。これはどのような問題を解決しますか。 4577

A.15.2 スレッドプールでは、最適なパフォーマンスとスループットのために同時セッションおよびトランザクションをどのように制限および管理しますか。 4578

A.15.3 スレッドプールは、クライアント側の接続プールとどのように異なりますか。 4578

A.15.4 スレッドプールはどのような場合に使用しますか。 4578

A.15.5 推奨されるスレッドプールの構成はありますか。 4578

A.15.1 スレッドプールとは何ですか。これはどのような問題を解決しますか。

MySQL スレッドプールは、MySQL サーバーのデフォルトの接続処理機能を拡張する MySQL サーバーのプラグインであり、ステートメント/クエリーおよびトランザクションの並列実行数を制限して、タスクを完了するための CPU およびメモリーのリソースがそれぞれに十分に確保されるようにします。MySQL 8.0 の場合、スレッドプールプラグインは商用製品である MySQL Enterprise Edition に含まれています。

MySQL サーバーのデフォルトのスレッド処理モデルでは、クライアント接続ごとに 1 つのスレッドを使用してステートメントが実行されます。より多くのクライアントがサーバーに接続してステートメントを実行す

ると、全体的なパフォーマンスが低下します。スレッドプールプラグインは、オーバーヘッドを軽減してパフォーマンスを向上させるように設計された代替のスレッド処理モデルを提供します。スレッドプールプラグインは、特に現在のマルチ CPU/コアシステムでの多数のクライアント接続において、ステートメント実行スレッドを効率的に管理することによって、サーバーのパフォーマンスを向上させます。

詳細は、[セクション5.6.3「MySQL Enterprise Thread Pool」](#)を参照してください。

- A.15.2.スレッドプールでは、最適なパフォーマンスとスループットのために同時セッションおよびトランザクションをどのように制限および管理しますか。

スレッドプールは、「分割統治」手法を使用して並列性を制限および調整します。MySQL サーバーのデフォルトの接続処理と異なり、スレッドプールでは接続とスレッドが分離され、接続およびそれらの接続から受け取ったステートメントを実行するスレッドが固定された関係を持たないようにになっています。スレッドプールは、構成可能なスレッドグループ内でクライアント接続を管理します。スレッドグループでは、実行される処理の性質に基づいて優先順位が付けられ、キューに入れられます。

詳細は、[セクション5.6.3.3「スレッドプール操作」](#)を参照してください。

- A.15.3.スレッドプールは、クライアント側の接続プールとどのように異なりますか。

MySQL 接続プールはクライアント側で動作し、MySQL サーバーに対して MySQL クライアントで接続および切断が繰り返されないようにします。MySQL クライアントのアイドル状態の接続をキャッシュし、必要に応じてほかのユーザーが使用できるように設計されています。これにより、クエリーが MySQL サーバーに発行されたときに、接続の確立および切断のオーバーヘッドおよびコストが最小化されます。MySQL 接続プールには、バックエンド MySQL サーバーのクエリー処理機能またはロードに関する可視性はありません。対照的に、スレッドプールは MySQL サーバー側で動作し、バックエンド MySQL データベースにアクセスするクライアント接続から受信されるインバウンド同時接続およびクエリーの実行を管理するように設計されています。機能が分離されているため、MySQL 接続プールとスレッドプールは次元の異なるものであり、相互に独立して使用できます。

MySQL Connector を使用した MySQL の接続プールについては、[第29章「Connector および API」](#)で説明しています。

- A.15.4.スレッドプールはどのような場合に使用しますか。

スレッドプールを最適に使用するには、考慮すべきいくつかの経験則があります。

MySQL の `Threads_running` 変数は、MySQL サーバーで現在実行されている並列ステートメントの数を追跡します。この変数が、サーバーが最適に動作しない (通常は InnoDB ワークロードで 40 を超える) リージョンを一貫して超えている場合、特に極端な並列オーバーロードの状況ではスレッドプールが役立ちます。

`innodb_thread_concurrency` を使用して同時に実行するステートメントの数を制限する場合は、スレッドプールがスレッドグループに接続を割り当て、トランザクションコンテンツ、ユーザー定義の指定などに基づいて実行をキューイングすることで、同じ問題をより適切に解決する必要があります。

最後に、ワークロードが主に短いクエリーで構成されている場合は、スレッドプールが有益です。

詳細は、[セクション5.6.3.4「スレッドプールのチューニング」](#)を参照してください。

- A.15.5.推奨されるスレッドプールの構成はありますか。

スレッドプールには、パフォーマンスに影響する、ユーザー事例から追加された多数の構成パラメータがあります。これらの詳細、およびチューニングのヒントについては、[セクション5.6.3.4「スレッドプールのチューニング」](#)を参照してください。

A.16 MySQL 8.0 FAQ : InnoDB 変更バッファ

- A.16.1 セカンダリインデックスを変更してバッファリングを変更する操作のタイプは何ですか。 4579
A.16.2 InnoDB 変更バッファにはどのようなメリットがありますか。 4579
A.16.3 変更バッファは他のタイプのインデックスをサポートしていますか。 4579
A.16.4 InnoDB は変更バッファにどのくらいの領域を使用しますか。 4579
A.16.5 変更バッファの現在のサイズを確認するにはどうすればよいですか。 4579

A.16.6 変更バッファのマージはいつ行われますか。	4579
A.16.7 変更バッファはいつフラッシュされますか。	4580
A.16.8 変更バッファはいつ使用しますか。	4580
A.16.9 変更バッファを使用しないのはいつですか。	4580
A.16.10 変更バッファに関する追加情報はどこで入手できますか。	4580

A.16.1.セカンダリインデックスを変更してバッファリングを変更する操作のタイプは何ですか。

INSERT、**UPDATE** および **DELETE** 操作では、セカンダリインデックスを変更できます。影響を受けるインデックスページがバッファプールにない場合、変更は変更バッファにバッファできます。

A.16.2.InnoDB 変更バッファにはどのようなメリットがありますか。

セカンダリインデックスページがバッファプールにない場合にセカンダリインデックスの変更をバッファリングすると、影響を受けるインデックスページをディスクからすぐに読み取るために必要な負荷の高いランダムアクセス I/O 操作が回避されます。バッファされた変更は、他の読取り操作によってバッファプールにページが読み取られるため、後でバッチで適用できます。

A.16.3.変更バッファは他のタイプのインデックスをサポートしていますか。

いいえ。変更バッファでは、セカンダリインデックスのみがサポートされます。クラスタインデックス、全文インデックスおよび空間インデックスはサポートされていません。全文インデックスには、独自のキャッシュメカニズムがあります。

A.16.4.InnoDB は変更バッファにどのくらいの領域を使用しますか。

MySQL 5.6 で `innodb_change_buffer_max_size` 構成オプションが導入される前は、システムテーブルスペースのディスク上の変更バッファの最大サイズは InnoDB バッファプールサイズの 1/3 でした。

MySQL 5.6 以降では、`innodb_change_buffer_max_size` 構成オプションによって、変更バッファの最大サイズがバッファプールの合計サイズに対する割合として定義されます。デフォルトでは、`innodb_change_buffer_max_size` は 25 に設定されます。最大設定は 50 です。

ディスク上の変更バッファが定義された制限を超える場合、InnoDB は操作をバッファしません。

変更バッファページは、バッファプールに永続化する必要はなく、LRU 操作によって削除される場合があります。

A.16.5.変更バッファの現在のサイズを確認するにはどうすればよいですか。

変更バッファの現在のサイズは、`SHOW ENGINE INNODB STATUS \G` によって **INSERT BUFFER AND ADAPTIVE HASH INDEX** ヘッダーの下にレポートされます。例:

```
-----  
INSERT BUFFER AND ADAPTIVE HASH INDEX  
-----  
lbuf: size 1, free list len 0, seg size 2, 0 merges
```

関連するデータポイントは次のとおりです:

- **size**: 変更バッファ内で使用されるページ数。変更バッファサイズは `seg size - (1 + free list len)` と同じです。1 + 値は、変更バッファヘッダーページを表します。
- **seg size**: 変更バッファのサイズ (ページ単位)。

変更バッファステータスの監視の詳細は、[セクション15.5.2「変更バッファ」](#) を参照してください。

A.16.6.変更バッファのマージはいつ行われますか。

- ページがバッファプールに読み込まれると、バッファされた変更は、ページが使用可能になる前に、読取りの完了時にマージされます。
- 変更バッファのマージはバックグラウンドタスクとして実行されます。`innodb_io_capacity` パラメータは、変更バッファからのデータのマージなど、InnoDB バックグラウンドタスクによって実行される I/O アクティビティの上限を設定します。

- クラッシュリカバリ中に変更バッファのマー지가実行されます。インデックスページがバッファプールに読み込まれると、変更は変更バッファ (システムテーブルスペース内) からセカンダリインデックスのリーフページに適用されます。
- 変更バッファは完全に永続的であり、システムクラッシュが発生する可能性があります。再起動時に、変更バッファのマージ操作は通常の操作の一部として再開されます。
- 変更バッファの完全マージは、`--innodb-fast-shutdown=0` を使用した低速なサーバー停止の一部として強制できます。

A.16.7 変更バッファはいつフラッシュされますか。

更新されたページは、バッファプールを占有するほかのページをフラッシュするのと同じフラッシュメカニズムによってフラッシュされます。

A.16.8 変更バッファはいつ使用しますか。

変更バッファは、インデックスが大きくなり、InnoDB バッファプールに収まらなくなったときに、ランダムな I/O をセカンダリインデックスに減らすように設計された機能です。一般に、データセット全体がバッファプールに収まらない場合、セカンダリインデックスページを変更する大量の DML アクティビティがある場合、または DML アクティビティによって定期的に変更されるセカンダリインデックスが多数ある場合は、変更バッファを使用する必要があります。

A.16.9 変更バッファを使用しないのはいつですか。

データセット全体が InnoDB バッファプール内に収まる場合、比較的少ないセカンダリインデックスがある場合、またはソリッドステートストレージを使用している場合 (ランダム読み取りは順次読み取りとほぼ同じくらい高速)、変更バッファを無効にすることを検討してください。構成を変更する前に、代表的なワークロードを使用してテストを実行し、変更バッファを無効にすると利点があるかどうかを判断することをお勧めします。

A.16.10 変更バッファに関する追加情報はどこで入手できますか。

[セクション15.5.2「変更バッファ」](#)を参照してください。

A.17 MySQL 8.0 FAQ : InnoDB 保存データ暗号化

A.17.1 データを表示する権限を持つユーザーのデータは復号化されますか。	4580
A.17.2 InnoDB の保存データ暗号化に関連するオーバーヘッドは何ですか。	4581
A.17.3 InnoDB の保存データ暗号化で使用される暗号化アルゴリズムは何ですか。	4581
A.17.4 InnoDB の保存データ暗号化機能によって提供されるアルゴリズムのかわりにサードパーティの暗号化アルゴリズムを使用できますか。	4581
A.17.5 インデックス付きカラムを暗号化できますか。	4581
A.17.6 InnoDB の保存データ暗号化では、どのようなデータ型とデータ長がサポートされていますか。	4581
A.17.7 データはネットワーク上で暗号化されたままですか。	4581
A.17.8 データベースメモリーにクリアテキストまたは暗号化されたデータが含まれていますか。	4581
A.17.9 暗号化するデータを知るにはどうすればよいですか。	4581
A.17.10 InnoDB の保存データ暗号化は、MySQL ですでに提供されている暗号化機能とどのように異なりますか。	4581
A.17.11 トランスポータブルテーブルスペース機能は、InnoDB の保存データ暗号化で動作しますか。	4581
A.17.12 圧縮は、InnoDB の保存データ暗号化で機能しますか。	4581
A.17.13 暗号化されたテーブルで <code>mysqlpump</code> または <code>mysqldump</code> を使用できますか。	4582
A.17.14 マスター暗号化キーを変更 (ローテーション、キー更新) するにはどうすればよいですか。	4582
A.17.15 クリアテキストの InnoDB テーブルスペースから暗号化された InnoDB テーブルスペースにデータを移行するには、どのようにしますか。	4582

A.17.1 データを表示する権限を持つユーザーのデータは復号化されますか。

はい。InnoDB の保存データ暗号化は、既存のアプリケーションに影響を与えずにデータベース内で暗号化を透過的に適用するように設計されています。暗号化された形式でデータを返すと、ほとんどの既存のアプリケーションが破損します。InnoDB の保存データ暗号化は、従来のデータベース暗号化ソリューションに関連

するオーバーヘッドなしで暗号化の利点を提供します。通常、アプリケーション、データベーストリガーおよびビューに対するコストのかかる大幅な変更が必要になります。

A.17.2.InnoDB の保存データ暗号化に関連するオーバーヘッドは何ですか。

追加の記憶域オーバーヘッドはありません。内部ベンチマークによると、パフォーマンスオーバーヘッド金額は単一桁のパーセント差異になります。

A.17.3.InnoDB の保存データ暗号化で使用する暗号化アルゴリズムは何ですか。

InnoDB の保存データ暗号化では、Advanced Encryption Standard (AES256) ブロックベースの暗号化アルゴリズムがサポートされています。テーブルスペースキー暗号化には電子コードブック (ECB) ブロック暗号化モードを使用し、データ暗号化には暗号ブロックチェーン (CBC) ブロック暗号化モードを使用します。

A.17.4.InnoDB の保存データ暗号化機能によって提供されるアルゴリズムのかわりにサードパーティの暗号化アルゴリズムを使用できますか。

いいえ。他の暗号化アルゴリズムは使用できません。指定された暗号化アルゴリズムは広範囲に受け入れられます。

A.17.5.インデックス付きカラムを暗号化できますか。

InnoDB の保存データ暗号化では、すべてのインデックスが透過的にサポートされます。

A.17.6.InnoDB の保存データ暗号化では、どのようなデータ型とデータ長がサポートされていますか。

InnoDB の保存データ暗号化では、サポートされているすべてのデータ型がサポートされます。データ長の制限はありません。

A.17.7.データはネットワーク上で暗号化されたままですか。

InnoDB の保存データ機能によって暗号化されたデータは、テーブルスペースファイルから読み取られるときに復号化されます。したがって、データがネットワーク上にある場合は、クリアテキスト形式になります。ただし、ネットワーク上のデータは、SSL/TLS を使用してデータベースとの間を移動するデータを暗号化する MySQL ネットワーク暗号化を使用して暗号化できます。

A.17.8.データベースメモリーにクリアテキストまたは暗号化されたデータが含まれていますか。

InnoDB の保存データ暗号化では、メモリー内データが復号化され、完全な透過性が提供されます。

A.17.9.暗号化するデータを知るにはどうすればよいですか。

PCI-DSS 標準に準拠するには、クレジットカード番号 (プライマリアカウント番号または PAN) を暗号化された形式で格納する必要があります。違反通知法 (CA SB 1386、CA AB 1950、43 以上の米国の州における同様の法律など) では、名、姓、運転免許証番号およびその他の PII データの暗号化が必要です。2008 年初頭、CA AB 1298 は PII データに医療保険および健康保険情報を追加しました。さらに、業界固有のプライバシーおよびセキュリティ標準では、特定のアセットの暗号化が必要になる場合があります。たとえば、医薬品研究結果、油田探索結果、金融契約、法律執行情報の個人データなどの資産では、暗号化が必要になる場合があります。医療業界では、患者データ、健康記録、X 線画像のプライバシーが最も重要です。

A.17.10.InnoDB の保存データ暗号化は、MySQL ですでに提供されている暗号化機能とどのように異なりますか。

MySQL には、データベース内のデータを手動で暗号化するために使用できる対称および非対称の暗号化 API があります。ただし、アプリケーションは暗号化キーを管理し、API 関数をコールして必要な暗号化および復号化操作を実行する必要があります。InnoDB の保存データ暗号化は、アプリケーションの変更を必要とせず、エンドユーザーに対して透過的であり、自動化された組込みキー管理を提供します。

A.17.11.トランスポータブルテーブルスペース機能は、InnoDB の保存データ暗号化で動作しますか。

はい。暗号化された file-per-table テーブルスペースでサポートされています。詳細は、[暗号化されたテーブルスペースのエクスポート](#)を参照してください。

A.17.12.圧縮は、InnoDB の保存データ暗号化で機能しますか。

圧縮はデータブロックが暗号化される前に適用されるため、InnoDB の保存データ暗号化を使用している顧客には圧縮の完全な利点があります。

A.17.13 暗号化されたテーブルで `mysqldump` または `mysqldump` を使用できますか。

はい。これらのユーティリティは論理バックアップを作成するため、暗号化されたテーブルからダンプされたデータは暗号化されません。

A.17.14 マスター暗号化キーを変更 (ローテーション、キー更新) するにはどうすればよいですか。

InnoDB の保存データ暗号化では、2 層キーメカニズムが使用されます。保存データ暗号化を使用すると、個々のテーブルスペースキーが基礎となるテーブルスペースデータファイルのヘッダーに格納されます。テーブルスペースキーは、マスター暗号化キーを使用して暗号化されます。マスター暗号化キーは、テーブルスペースの暗号化が有効な場合に生成され、データベースの外部に格納されます。マスター暗号化キーは、新しいマスター暗号化キーを生成し、キーを格納し、使用するキーをローテーションする `ALTER INSTANCE ROTATE INNODB MASTER KEY` ステートメントを使用してローテーションされます。

A.17.15 リアテキストの InnoDB テーブルスペースから暗号化された InnoDB テーブルスペースにデータを移行するには、どのようにしますか。

あるテーブルスペースから別のテーブルスペースへのデータの転送は不要です。InnoDB file-per-table テーブルスペースのデータを暗号化するには、`ALTER TABLE tbl_name ENCRYPTION = 'Y'` を実行します。一般テーブルスペースまたは `mysql` テーブルスペースを暗号化するには、`ALTER TABLESPACE tablespace_name ENCRYPTION = 'Y'` を実行します。一般テーブルスペースの暗号化サポートは、MySQL 8.0.13 で導入されました。mysql システムテーブルスペースの暗号化サポートは、MySQL 8.0.16 の時点で使用できます。

A.18 MySQL 8.0 FAQ : 仮想化のサポート

A.18.1 MySQL は、Oracle VM、VMWare、Docker、Microsoft Hyper-V などの仮想環境でサポートされていますか。 4582

A.18.1 MySQL は、Oracle VM、VMWare、Docker、Microsoft Hyper-V などの仮想環境でサポートされていますか。

MySQL は仮想化環境でサポートされていますが、Oracle VM でのみ動作保証されています。詳細は、Oracle Support にお問い合わせください。

仮想化ソフトウェアを使用する場合は、潜在的な問題に注意してください。通常は、ディスク、I/O、ネットワークおよびメモリーのパフォーマンス、パフォーマンスの低下、速度または予測不可能性に関連していません。

付録 B エラーメッセージと一般的な問題

目次

B.1 エラーメッセージのソースと要素	4583
B.2 エラー情報インタフェース	4585
B.3 問題および一般的なエラー	4587
B.3.1 問題の原因を判別する方法	4587
B.3.2 MySQL プログラム使用時の一般的なエラー	4588
B.3.3 管理関連の問題	4598
B.3.4 クエリー関連の問題	4605
B.3.5 オプティマイザ関連の問題	4611
B.3.6 テーブル定義関連の問題	4612
B.3.7 MySQL の既知の問題	4613

この付録では、MySQL で提供されるエラー情報のタイプとその情報の取得方法について説明します。最後のセクションでは、トラブルシューティングについて説明します。発生する可能性のある一般的な問題とエラー、および潜在的な解決策について説明します。

追加のリソース

その他のエラー関連ドキュメントは次のとおりです:

- サーバーがエラーログを書き込む場所と方法の構成に関する情報: [セクション5.4.2「エラーログ」](#)
- エラーメッセージに使用される文字セットに関する情報: [セクション10.6「エラーメッセージ文字セット」](#)
- エラーメッセージに使用される言語に関する情報: [セクション10.12「エラーメッセージ言語の設定」](#)
- InnoDB に関連するエラーに関する情報: [セクション15.21.4「InnoDB のエラー処理」](#)
- MySQL サーバーおよびクライアントプログラムによって生成されるエラーメッセージの説明: [MySQL 8.0 Error Message Reference](#)

B.1 エラーメッセージのソースと要素

このセクションでは、エラーメッセージが MySQL 内でどのように生成されるか、およびエラーメッセージに含まれる要素について説明します。

- [エラーメッセージのソース](#)
- [エラーメッセージの要素](#)
- [エラーコード範囲](#)

エラーメッセージのソース

エラーメッセージは、サーバー側またはクライアント側で発生する可能性があります:

- サーバー側では、SQL ステートメントの実行中に発生する問題などの結果として、起動および停止プロセス中にエラーメッセージが表示される場合があります。
- MySQL サーバーは、エラーログにいくつかのエラーメッセージを書き込みます。これらは、データベース管理者にとって関心のある問題、または DBA アクションが必要な問題を示します。
- サーバーは、ほかのエラーメッセージをクライアントプログラムに送信します。これらは、特定のクライアントにのみ関連する問題を示します。MySQL クライアントライブラリは、サーバーから受信したエラーを取得し、ホストクライアントプログラムで使用できるようにします。

- クライアント側のエラーメッセージは MySQL クライアントライブラリ内から生成され、通常はサーバーとの通信に問題が発生します。

エラーログに書き込まれるサーバー側のエラーメッセージの例:

- 起動プロセス中に生成されるこのメッセージには、ステータスインジケータまたは進捗インジケータが表示されません:

```
2018-10-28T13:01:32.735983Z 0 [Note] [MY-010303] [Server] Skipping generation of SSL certificates as options related to SSL are specified.
```

- このメッセージは、DBA アクションが必要な問題を示しています:

```
2018-10-02T03:20:39.410387Z 768 [ERROR] [MY-010045] [Server] Event Scheduler: [evtuser@localhost][myschema.e_daily] Unknown database 'mydb'
```

mysql クライアントによって表示される、クライアントプログラムに送信されるサーバー側のエラーメッセージの例:

```
mysql> SELECT * FROM no_such_table;
ERROR 1146 (42S02): Table 'test.no_such_table' doesn't exist
```

mysql クライアントによって表示される、クライアントライブラリ内からのクライアント側エラーメッセージの例:

```
shell> mysql -h no-such-host
ERROR 2005 (HY000): Unknown MySQL server host 'no-such-host' (0)
```

エラーがクライアントライブラリ内から発生したか、サーバーから受信されたかにかかわらず、MySQL クライアントプログラムは様々な方法で応答できます。図のように、ユーザーが修正策を講じることができるよう、クライアントにエラーメッセージが表示される場合があります。かわりに、クライアントは内部的に失敗した操作を解決または再試行するか、他のアクションを実行できます。

エラーメッセージの要素

エラーが発生した場合、エラー情報には複数の要素が含まれます: エラーコード、SQLSTATE 値、およびメッセージ文字列。これらの要素には、次の特性があります:

- エラーコード: この値は数値です。MySQL 固有であり、他のデータベースシステムには移植できません。

各エラー番号には対応するシンボリック値があります。例:

- サーバーエラー番号 **1146** の記号は **ER_NO_SUCH_TABLE** です。
- クライアントエラー番号 **2005** の記号は **CR_UNKNOWN_HOST** です。

エラーメッセージで使用されるエラーコードのセットは、個別の範囲にパーティション化されます。[エラーコード範囲](#) を参照してください。

エラーコードは、特定の MySQL シリーズの General Availability (GA) リリース間で安定しています。シリーズが GA ステータスになる前に、新しいコードがまだ開発中であり、変更される可能性があります。

- SQLSTATE 値: この値は 5 文字の文字列 ('42S02' など) です。SQLSTATE 値は ANSI SQL および ODBC から取得され、数値エラーコードよりも標準化されています。SQLSTATE 値の最初の 2 文字はエラークラスを示しています。

- クラス = '00' は成功を示しています。
- クラス = '01' は警告を示しています。
- クラス = '02' は「Not Found」を示しています。これは、カーソルのコンテキストに関係しており、カーソルがデータセットの最後に達したときの動作を制御するために使用します。この状況は、行が取得されない **SELECT ... INTO var_list** ステートメントでも発生します。
- クラス > '02' は例外を示しています。

サーバー側エラーの場合、すべての MySQL エラー番号に対応する SQLSTATE 値があるわけではありません。それらの場合は、**'HY000'** (一般エラー) が使用されます。

クライアント側のエラーの場合、SQLSTATE 値は常に'HY000' (一般エラー) であるため、あるクライアントエラーを別のクライアントエラーと区別することは意味がありません。

- メッセージ文字列: この文字列は、エラーのテキストによる説明を提供します。

エラーコード範囲

エラーメッセージで使用されるエラーコードのセットは、それぞれ独自の目的を持つ個別の範囲にパーティション化されます:

- 1 から 999: グローバルエラーコード。このエラーコード範囲は、サーバーおよびクライアントによって使用される共有範囲であるため、「global」と呼ばれます。

この範囲のエラーがサーバー側で発生すると、サーバーはエラーコードをエラーログに書き込み、エラーコードの先頭にゼロを付けて 6 桁にし、**MY-**の接頭辞を追加します。

この範囲のエラーがクライアント側で発生すると、クライアントライブラリはゼロ埋込みや接頭辞なしでクライアントプログラムで使用できるようになります。

- 1,000 から 1,999: クライアントに送信されるメッセージ用に予約されているサーバーエラーコード。
- 2,000 から 2,999: クライアントライブラリで使用するために予約されているクライアントエラーコード。
- 3,000 から 4,999: クライアントに送信されるメッセージ用に予約されているサーバーエラーコード。
- 5,000 から 5,999: クライアントに送信されるメッセージ用に X プラグイン で使用するために予約されているエラーコード。
- 10,000 から 49,999: エラーログに書き込まれるメッセージ用に予約されているサーバーエラーコード (クライアントには送信されません)。

この範囲のエラーが発生すると、サーバーはエラーログに書き込み、エラーコードの先頭にゼロを 6 桁に埋め込み、**MY-**の接頭辞を追加します。

- 50,000 から 51,999: サードパーティが使用するために予約されたエラーコード。

サーバーは、エラーログに書き込まれたエラーメッセージを、クライアントに送信されたエラーメッセージとは異なる方法で処理します:

- サーバーがエラーログにメッセージを書き込むと、エラーコードが先頭にゼロを付けて 6 桁に埋め込まれ、**MY-**の接頭辞が追加されます (例: **MY-000022**, **MY-010048**) 。
- サーバーがクライアントプログラムにメッセージを送信すると、エラーコードにゼロ埋込みや接頭辞は追加されません (例: **1036**, **3013**) 。

B.2 エラー情報インタフェース

[セクションB.1「エラーメッセージのソースと要素」](#) で説明されているように、エラーメッセージはサーバー側またはクライアント側で生成され、各エラーメッセージにはエラーコード、SQLSTATE 値、およびメッセージ文字列が含まれます。サーバー側、クライアント側およびグローバル (サーバーとクライアント間で共有) のエラーのリストは、[MySQL 8.0 Error Message Reference](#) を参照してください。

プログラム内からのエラーチェックには、エラーメッセージ文字列ではなく、エラーコード番号または記号を使用します。メッセージ文字列は頻繁には変更されませんが、変更は可能です。また、データベース管理者がメッセージ文字列の言語に影響する言語設定を変更した場合は、[セクション10.12「エラーメッセージ言語の設定」](#) を参照してください。

MySQL のエラー情報は、サーバーエラーログ、SQL レベル、クライアントプログラムおよびコマンドラインから入手できます。

- [エラーログ](#)
- [SQL エラーメッセージインタフェース](#)

- [クライアントエラーメッセージインタフェース](#)
- [コマンドラインエラーメッセージインタフェース](#)

エラーログ

サーバー側では、一部のメッセージはエラーログ用です。サーバーがログを書き込む場所と方法の構成の詳細は、[セクション5.4.2「エラーログ」](#)を参照してください。

その他のサーバーエラーメッセージはクライアントプログラムに送信することを目的としており、[クライアントエラーメッセージインタフェース](#)で説明されているように使用できます。

特定のエラーコードが存在する範囲によって、サーバーがエラーメッセージをエラーログに書き込むか、クライアントに送信するかが決まります。これらの範囲の詳細は、[エラーコード範囲](#)を参照してください。

SQL エラーメッセージインタフェース

SQL レベルでは、MySQL にエラー情報のソースがいくつかあります:

- SQL ステートメントの警告およびエラーの情報は、[SHOW WARNINGS](#) ステートメントおよび [SHOW ERRORS](#) ステートメントを使用して表示できます。 `warning_count` システム変数は、エラー、警告およびノートの数を示します (`sql_notes` システム変数が無効な場合、ノートは除外されます)。 `error_count` システム変数はエラー数を示します。この値には警告および注意は含まれていません。
- [GET DIAGNOSTICS](#) ステートメントは、診断領域の診断情報を調査するために使用できます。[セクション13.6.7.3「GET DIAGNOSTICS ステートメント」](#)を参照してください。
- [SHOW SLAVE STATUS](#) ステートメントの出力には、レプリカサーバーで発生したレプリケーションエラーに関する情報が含まれます。
- InnoDB テーブルに対する [CREATE TABLE](#) ステートメントが失敗した場合、[SHOW ENGINE INNODB STATUS](#) ステートメントの出力には、最後の外部キーエラーに関する情報が含まれています。

クライアントエラーメッセージインタフェース

クライアントプログラムは、次の2つのソースからエラーを受け取ります:

- MySQL クライアントライブラリ内からクライアント側で発生したエラー。
- サーバー側で発生し、サーバーによってクライアントに送信されるエラー。これらはクライアントライブラリ内で受信され、ホストクライアントプログラムで使用できるようになります。

特定のエラーコードが存在する範囲によって、クライアントライブラリ内から発生したものが、クライアントがサーバーから受信したものが決まります。これらの範囲の詳細は、[エラーコード範囲](#)を参照してください。

エラーがクライアントライブラリ内から発生したか、サーバーから受信されたかに関係なく、MySQL クライアントプログラムは、クライアントライブラリで C API 関数をコールして、エラーコード、SQLSTATE 値、メッセージ文字列およびその他の関連情報を取得します:

- `mysql_errno()` は、MySQL エラーコードを返します。
- `mysql_sqlstate()` は SQLSTATE 値を返します。
- `mysql_error()` はメッセージ文字列を返します。
- `mysql_stmt_errno()`、`mysql_stmt_sqlstate()` および `mysql_stmt_error()` は、プリペアドステートメントに対応するエラー関数です。
- `mysql_warning_count()` は、最新のステートメントのエラー、警告およびノートの数を返します。

クライアントライブラリのエラー関数については、[MySQL 8.0 C API Developer Guide](#)を参照してください。

MySQL クライアントプログラムは、様々な方法でエラーに応答する場合があります。クライアントはエラーメッセージを表示して、ユーザーが修正措置を講じるか、内部的に失敗した操作の解決または再試行を試みるか、または

他のアクションを実行できるようにすることができます。たとえば、(mysql クライアントを使用して) サーバーへの接続に失敗すると、次のメッセージが表示されることがあります:

```
shell> mysql -h no-such-host
ERROR 2005 (HY000): Unknown MySQL server host 'no-such-host' (0)
```

コマンドラインエラーメッセージインタフェース

`perror` プログラムは、コマンド行からエラー番号に関する情報を表示します。 [セクション4.8.2「 perror — MySQL エラーメッセージ情報の表示」](#) を参照してください。

```
shell> perror 1231
MySQL error code MY-001231 (ER_WRONG_VALUE_FOR_VAR): Variable '%-.64s'
can't be set to the value of '%-.200s'
```

MySQL NDB Cluster エラーの場合は、 `ndb_perror` を使用します。 [セクション23.4.16「 ndb_perror — NDB エラーメッセージ情報の取得」](#) を参照してください。

```
shell> ndb_perror 323
NDB error code 323: Invalid nodegroup id, nodegroup already existing:
Permanent error: Application error
```

B.3 問題および一般的なエラー

このセクションでは、発生する可能性がある一般的な問題およびエラーメッセージを一覧表示しています。問題の原因を判別する方法およびそれらを解決するための手順について説明しています。

B.3.1 問題の原因を判別する方法

問題が発生したときに最初にすべきことは、原因となっているプログラムまたはユニットを特定することです。

- 次のいずれかの症状が発生している場合は、ハードウェアの問題 (メモリー、マザーボード、CPU、ハードディスクなど) またはカーネルの問題である可能性があります。
 - キーボードが動作しない。通常、これは Caps Lock キーを押すことによって確認できます。Caps Lock のランプが変わらない場合は、キーボードを交換する必要があります。(これを行う前に、コンピュータの再起動を試みて、キーボードのすべてのケーブルを確認する必要があります。)
 - マウスポインタが動かない。
 - リモートマシンからの ping にマシンが応答しない。
 - MySQL に関連しないその他のプログラムが正常に動作しない。
 - システムが突然再起動される。(ユーザーレベルの欠陥のあるプログラムがシステムを停止できないようにしてください。)

この場合は、すべてのケーブルを確認し、診断ツールを実行してハードウェアをチェックすることから開始してください。問題を解決できる可能性があるオペレーティングシステムのパッチ、アップデート、またはサービスパックがあるかどうかを確認してください。すべてのライブラリ (`glibc` など) が最新であることも確認してください。

メモリーの問題を早期に発見するために、ECC メモリーを持つマシンを使用することは良いことです。

- キーボードがロックアップした場合は、別のマシンから自分のマシンにログインして `kbd_mode -a` を実行することによってリカバリできることがあります。
- システムのログファイル (`/var/log/messages` または同様のログファイル) で問題の原因を調べてください。問題の原因が MySQL にあると思われる場合は、MySQL のログファイルも調べてください。 [セクション5.4「MySQL Server ログ」](#) を参照してください。
- ハードウェアに問題がないと思われる場合は、問題の原因となっているプログラムを見つけてください。 `top` 、 `ps` 、タスクマネージャー、または同様のプログラムを使用して、すべての CPU を使用しているプログラムまたはマシンをロックしているプログラムを確認します。

- `top`、`df`、または同様のプログラムを使用して、メモリー、ディスク領域、ファイルディスクリプタ、またはその他の重要なリソースが不足しているかどうかを確認します。
- 問題の原因が暴走したプロセスにある場合は、そのプロセスの強制終了を試みることができます。プロセスが停止しない場合は、オペレーティングシステムにバグがある可能性があります。

他のすべての可能性を調べ、MySQL サーバーまたは MySQL クライアントが問題の原因となっていると判断した場合は、バグレポートを作成します。セクション1.6「質問またはバグをレポートする方法」を参照してください。バグレポートで、システムがどのように動作しているか、および何が起きていると思われるかについて完全に説明してみてください。また、MySQL が問題の原因と考えている理由も記載します。この章で説明するすべての状況を考慮してください。システムを検査したときの問題の状況を正確に記述します。プログラムおよびログファイルの出力やエラーメッセージを「コピー&ペースト」します。

動作していないプログラムおよびすべての症状について詳しく記述してください。「システムが動作しない」とのみ記述されたバグレポートを過去に多数受け取りました。これでは、問題の原因に関する情報が提供されません。

プログラムで障害が発生した場合は、次の情報を知ることが常に役に立ちます。

- 問題のプログラムでセグメンテーション違反が発生したかどうか (コアがダンプされたかどうか)。
- プログラムが使用可能なすべての CPU 時間を使用しているかどうか。 `top` を使用して確認してください。プログラムをしばらく実行したままにしてみてください。計算の多い処理が行われているだけである可能性があります。
- `mysqld` サーバーが問題の原因である場合は、`mysqladmin -u root ping` または `mysqladmin -u root processlist` で応答を取得できますか。
- MySQL サーバーに接続しようとしたときに、クライアントプログラムはどのように動作しますか。(たとえば、`mysql` を実行します。) クライアントが動作しなくなりますか。プログラムから出力はありますか。

バグレポートを送信する場合は、セクション1.6「質問またはバグをレポートする方法」に説明されている手順に従ってください。

B.3.2 MySQL プログラム使用時の一般的なエラー

このセクションでは、MySQL プログラムを実行したときによく発生するエラーを一覧表示しています。問題はクライアントプログラムを実行しようとしたときに発生しますが、多くの問題の解決策では MySQL サーバーの構成を変更します。

B.3.2.1 アクセスは拒否されました

「アクセスは拒否されました」というエラーには多くの原因があります。多くの場合、問題はサーバーが接続時にクライアントプログラムに使用を許可する MySQL アカウントに関連しています。セクション6.2「アクセス制御とアカウント管理」およびセクション6.2.21「MySQL への接続の問題のトラブルシューティング」を参照してください。

B.3.2.2 [ローカルの] MySQL サーバーに接続できません

UNIX 上の MySQL クライアントは `mysqld` サーバーに 2 つの方法で接続できます。UNIX ソケットファイルを使用してファイルシステム内のファイル (デフォルトは `/tmp/mysql.sock`) を介して接続するか、TCP/IP を使用してポート番号を介して接続します。UNIX ソケットファイルでの接続は TCP/IP よりも高速ですが、同じコンピュータ上にあるサーバーに接続するときのみ使用できます。UNIX ソケットファイルは、ホスト名を指定しない場合、または特殊なホスト名 `localhost` を指定する場合に使用されます。

MySQL サーバーが Windows 上で実行されている場合は、TCP/IP を使用して接続できます。 `named_pipe` システム変数を有効にしてサーバーを起動した場合、サーバーが実行されているホストでクライアントを実行していれば、名前付きパイプで接続することもできます。デフォルトでは、名前付きパイプの名前は `MySQL` です。 `mysqld` に接続するときにホスト名を指定しない場合、MySQL クライアントは最初に名前付きパイプに接続しようとします。接続できない場合は、TCP/IP ポートに接続します。Windows で名前付きパイプの使用を強制するには、ホスト名として `.` を使用します。

エラー (2002) 「... に接続できません」は、通常、サーバーに接続しようとしたときに、システムで MySQL サーバーが実行されていなかったこと、あるいは間違った UNIX ソケットファイル名または TCP/IP ポート番号を使用していることを意味しています。使用している TCP/IP ポートがファイアウォールまたはポートブロックサービスによってブロックされていないことも確認してください。

エラー (2003) 「'server' の MySQL サーバーに接続できません (10061)」は、ネットワーク接続が拒否されたことを示しています。MySQL サーバーが実行されていること、ネットワーク接続が有効にされていること、および指定したネットワークポートがサーバーに構成されていることを確認してください。

サーバーのホストで `mysqld` という名前のプロセスが実行されているかどうかを確認することから始めます。(UNIX では `ps xa | grep mysqld`、Windows ではタスクマネージャーを使用します。) プロセスがない場合は、サーバーを起動してください。 [セクション2.10.2「サーバーの起動」](#) を参照してください。

`mysqld` プロセスが実行されている場合は、次のコマンドを使用してチェックできます。ポート番号または UNIX ソケットファイル名は、使用している環境では異なる場合があります。`host_ip` は、サーバーが実行されているマシンの IP アドレスを表しています。

```
shell> mysqladmin version
shell> mysqladmin variables
shell> mysqladmin -h 'hostname' version variables
shell> mysqladmin -h 'hostname' --port=3306 version
shell> mysqladmin -h host_ip version
shell> mysqladmin --protocol=SOCKET --socket=/tmp/mysql.sock version
```

`hostname` コマンドでは通常の引用符ではなく逆引用符が使用されています。これにより、`hostname` の出力(つまり、現在のホスト名)が `mysqladmin` コマンドに渡されます。`hostname` コマンドがないか、Windows 上で実行している場合は、`-h` オプションに続けて、マシンのホスト名を手動で入力できます(逆引用符なし)。`-h 127.0.0.1` を使用して、TCP/IP でローカルホストへの接続を試みることもできます。

サーバーがネットワーク接続を無視するように構成されていないこと、または(リモート側から接続しようとする場合に)サーバーのネットワークインタフェース上でローカル側でのみ待機するように構成されていないことを確認します。`skip_networking` システム変数を有効にしてサーバーを起動した場合、TCP/IP 接続を受け入れることはできません。`bind_address` システム変数を `127.0.0.1` に設定してサーバーを起動した場合、サーバーはループバックインタフェースでローカルでのみ TCP/IP 接続をリスニングし、リモート接続を受け入れません。

ファイアウォールが MySQL へのアクセスをブロックしていないか確認します。ファイアウォールは、実行中のアプリケーションまたは MySQL によって通信用に使用されるポート番号(デフォルトは 3306)を基準として構成されることがあります。Linux または Unix の場合、IP テーブル(または同様の機能の)構成を調べてポートがブロックされていないことを確認します。Windows では、ZoneAlarm や Windows ファイアウォールなどのアプリケーションを、MySQL ポートをブロックしないように構成する必要がある場合があります。

「ローカルの MySQL サーバーに接続できません」というエラーが発生する可能性があるいくつかの原因を次に示します。

- `mysqld` がローカルホストで実行されていない。オペレーティングシステムのプロセスリストをチェックして、`mysqld` プロセスが存在することを確認します。
- 多数の TCP/IP 接続がある Windows 上で MySQL サーバーを実行している。クライアントで頻繁にそのエラーが発生している場合は、[Windows で MySQL サーバーへの接続に失敗するに回避策](#)があります。
- `mysqld` が使用する UNIX ソケットファイル(デフォルトでは `/tmp/mysql.sock`)をほかのユーザーが削除した。たとえば、`/tmp` ディレクトリから古いファイルを削除する `cron` ジョブがある可能性があります。`mysqladmin version` を実行すると、`mysqladmin` が使用する UNIX ソケットファイルが実際に存在するかどうかを確認できます。この場合の対処方法は、`mysql.sock` を削除しないように `cron` ジョブを変更するか、ソケットファイルを別の場所に配置することです。[セクションB.3.3.6「MySQL の UNIX ソケットファイルを保護または変更する方法」](#)を参照してください。
- `--socket=/path/to/socket` オプションを指定して `mysqld` サーバーを起動したが、クライアントプログラムにソケットファイルの新しい名前を指定し忘れた。サーバーのソケットパス名を変更したら、MySQL クライアントにも通知する必要があります。これを行うには、クライアントプログラムを実行するときと同じ `--socket` オプションを指定します。`mysql.sock` ファイルにアクセスするための権限がクライアントにあることも確認する必要があります。ソケットファイルがある場所を見つけるには、次のコマンドを実行します。

```
shell> netstat -ln | grep mysql
```

[セクションB.3.3.6「MySQL の UNIX ソケットファイルを保護または変更する方法」](#)を参照してください。

- Linux を使用していて、1つのサーバスレッドが停止した(コアがダンプされた)。この場合は、MySQL サーバーを再起動する前に、ほかの `mysqld` スレッドを強制終了する(たとえば、`kill` を使用します)必要があります。[セクションB.3.3.3「MySQL が繰り返しクラッシュする場合の対処方法」](#)を参照してください。

- UNIX ソケットファイルが保持されているディレクトリまたはソケットファイル自体に対する適切なアクセス権限が、サーバーまたはクライアントプログラムにない可能性がある。この場合は、ディレクトリまたはソケットファイルのアクセス権限を変更して、サーバーおよびクライアントがそれらにアクセスできるようにするか、サーバーがソケットファイルを作成でき、クライアントプログラムがアクセスできるディレクトリのソケットファイル名を指定する `--socket` オプションを指定して `mysqld` を再起動します。

「`some_host` 上の MySQL サーバーに接続できません」というエラーメッセージが表示された場合は、次のことを行って問題の原因を見つけてください。

- `telnet some_host 3306` を実行して Enter キーを何度か押すことによって、サーバーがそのホスト上で実行されているかどうかを確認します (3306 は MySQL のデフォルトのポート番号です。サーバーが別のポートで待機している場合は、値を変更します。) MySQL サーバーが実行されていて、そのポートで待機している場合は、サーバーのバージョン番号を含む応答を受け取ります。「`telnet: リモートホストに接続できません: 接続は拒否されました`」などのエラーを受け取った場合、指定したポートでサーバーは実行されていません。
- サーバーがローカルホストで実行されている場合は、`mysqladmin -h localhost variables` を使用して UNIX ソケットファイルを使用して接続することを試みてください。サーバーが待機するように構成されている TCP/IP ポート番号を確認します (これは `port` 変数の値です。)
- Linux で実行しており、セキュリティ強化された Linux (SELinux) が有効になっている場合は、[セクション 6.7「SELinux」](#) を参照してください。

Windows で MySQL サーバーへの接続に失敗する

多数の TCP/IP 接続のある Windows 上で MySQL サーバーを実行していて、「MySQL サーバーに接続できません」というエラーが頻繁に発生している場合は、それらの接続に対応するための十分なエフェメラル (一時的な) ポートを Windows が許可していないことが原因である可能性があります。

`TIME_WAIT` の目的は、接続が閉じられたあとでも、接続がパケットを受け入れるようにすることです。これは、インターネットのルーティングによってパケットに低速なルートが割り当てられ、双方がクローズに同意したあとにパケットが受信されることがあるためです。そのポートが新しい接続に使用された場合は、古い接続からのパケットによってプロトコルが断絶したり、元の接続からの個人情報が漏えいしたりすることがあります。`TIME_WAIT` による遅延は、それらの遅延したパケットを受信するために一定の時間を猶予してからポートが再使用されるようにすることによってこれを防ぎます。

LAN 接続では、距離と遅延が比較的大きいインターネットとは異なり、パケットが大きく遅延して受信されることはほとんどないため、`TIME_WAIT` を大幅に減らしたほうが安全です。

Windows はエフェメラル (一時的な) TCP ポートをユーザーに許可しています。ポートが閉じられると、120 秒間 `TIME_WAIT` ステータスのままになります。この時間が経過するまで、ポートは再び使用できません。ポート番号のデフォルトの範囲は、Windows のバージョンによって異なります。古いバージョンではポートの数がより制限されます。

- Windows Server 2003 まで: 1025-5000 の範囲のポート
- Windows Vista、Server 2008 以降: 49152-65535 の範囲のポート

使用可能な TCP ポートが少なく (5000)、`TIME_WAIT` ステータスで多数の TCP ポートが短い期間にオープンおよびクローズされると、ポートが不足する可能性が高くなります。この問題に対処する方法は 2 つあります。

- 原因となっている可能性のある接続プールまたは永続的な接続を調査することによって、急速に消費される TCP ポートの数を減らします
- Windows レジストリのいくつかの設定をチューニングします (次の手順を参照してください)

重要

次の手順では、Windows レジストリを変更します。レジストリを変更する前に、必ずレジストリをバックアップし、問題が発生した場合の復元方法を理解しておいてください。レジストリをバックアップ、復旧、および編集する方法については、Microsoft サポート技術情報の記事 (<http://support.microsoft.com/kb/256986/EN-US/>) を参照してください。

1. レジストリエディタ (`Regedt32.exe`) を開始します。

- レジストリの次のキーを見つけます。

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
```

- 「編集」メニューで、「値の追加」をクリックして次のレジストリ値を追加します。

```
Value Name: MaxUserPort  
Data Type: REG_DWORD  
Value: 65534
```

これにより、ユーザーが使用できるエフェメラルポートの番号が設定されます。有効な範囲は 5000 から 65534 (10 進数) までです。デフォルト値は 0x1388 (10 進数の 5000) です。

- 「編集」メニューで、「値の追加」をクリックして次のレジストリ値を追加します。

```
Value Name: TcpTimedWaitDelay  
Data Type: REG_DWORD  
Value: 30
```

これにより、閉じる前に `TIME_WAIT` 状態で TCP ポートの接続を保持する秒数が設定されます。有効な範囲は 30 から 300 桁ですが、Microsoft で許可されている最新の値を確認できます。デフォルト値は 0x78 (10 進数の 120) です。

- レジストリエディタを終了します。

- マシンをリブートします。

注意: 前述した設定を元に戻すには、作成したレジストリのエントリを削除します。

B.3.2.3 MySQL サーバーへの接続が失われました

このエラーメッセージには 3 つの原因が考えられます。

通常、これはネットワーク接続の問題を示しており、このエラーが頻繁に発生する場合は、ネットワークの状態を確認してください。エラーメッセージに「クエリー中」が含まれている場合、発生している問題はこれに該当する可能性があります。

「クエリー中」の状態は、1 つ以上のクエリーの一部分として数百万件の行が送信されているときに発生することがあります。この問題が発生していることがわかった場合は、`net_read_timeout` をデフォルトの 30 秒から 60 秒またはデータ転送が完了するのに十分な時間に増やすことを試みてください。

まれに、クライアントがサーバーに初期接続を試みるときに発生することがあります。この場合、`connect_timeout` 値の設定が数秒であるときは、10 秒 (距離が非常に遠い場合、または低速な接続の場合はさらに長く) に増やすことによって、この問題を解決できることがあります。より一般的ではない原因によってこの問題が発生しているかどうかを判別するには、`SHOW GLOBAL STATUS LIKE 'Aborted_connects'` を使用します。サーバーが中断する初期接続試行ごとに 1 つずつ増加します。エラーメッセージの一部として「認証パケットを読み込んでいます」が表示されることがあります。その場合も、必要となる解決策がこの方法であることを示しています。

前述の原因ではない場合は、一部のクライアントで発生することがある `max_allowed_packet` より大きい `BLOB` 値に関する問題が発生している可能性があります。`ER_NET_PACKET_TOO_LARGE` エラーが発生することがありますが、それは `max_allowed_packet` を増やす必要があることを示しています。

B.3.2.4 パスワードをインタラクティブに入力すると失敗する

`--password` オプションまたは `-p` オプションをパスワード値を指定せずに使用して起動すると、MySQL クライアントプログラムはパスワードの入力を求めます。

```
shell> mysql -u user_name -p  
Enter password:
```

一部のシステムでは、オプションファイルまたはコマンド行で指定するとパスワードが機能するが、`Enter password:` プロンプトでインタラクティブに入力すると機能しないことがあります。これは、パスワードを読み取るためにシステムによって提供されたライブラリが、パスワード値を少ない文字数 (通常、8 文字) に制限したときに発生します。これはシステムライブラリの問題であり、MySQL の問題ではありません。これを回避するには、MySQL のパスワードを 8 文字以下の値に変更するか、パスワードをオプションファイルに格納します。

B.3.2.5 接続が多すぎます

mysqld サーバーに接続しようとしたときにクライアントで **Too many connections** エラーが発生した場合、使用可能なすべての接続が他のクライアントによって使用されています。

許可される接続数は、`max_connections` システム変数によって制御されます。より多くの接続をサポートするには、`max_connections` をより大きな値に設定します。

mysqld では、実際には `max_connections + 1` クライアント接続が許可されます。追加の接続は、`CONNECTION_ADMIN` 権限 (または非推奨の `SUPER` 権限) を持つアカウントで使用するために予約されています。通常のユーザー (必要ないユーザー) ではなく管理者に権限を付与することで、管理者はサーバーに接続し、権限のないクライアントの最大数が接続されている場合でも `SHOW PROCESSLIST` を使用して問題を診断できます。セクション 13.7.7.29 「`SHOW PROCESSLIST` ステートメント」を参照してください。

サーバーは、専用インタフェースでの管理接続も許可します。サーバーがクライアント接続を処理する方法の詳細は、セクション 5.1.12.1 「`接続インタフェース`」を参照してください。

B.3.2.6 メモリ不足

mysql クライアントプログラムを使用してクエリーを発行し、次のようなエラーを受け取った場合は、mysql にクエリーの結果全体を格納するための十分なメモリがないことを意味しています。

```
mysql: Out of memory at line 42, 'malloc.c'  
mysql: needed 8136 byte (8k), memory in use: 12481367 bytes (12189k)  
ERROR 2008: MySQL client ran out of memory
```

この問題を解決するには、最初にクエリーが正しいかどうかを確認します。そのような多数の行が返されることが予想されるクエリーですか。そうではない場合は、クエリーを修正して再試行してください。予想される場合は、`--quick` オプションを指定して mysql を呼び出すことができます。これにより、`mysql_use_result()` C API 関数を使用して結果セットが取得されるようになり、クライアントへのロードが少なくなります (サーバーへのロードが増えます)。

B.3.2.7 MySQL サーバーが存在しなくなりました

このセクションでは、関連する「クエリー中にサーバーへの接続が失われました」というエラーについても説明します。

「MySQL サーバーが存在しなくなりました」というエラーのもっとも一般的な原因は、サーバーがタイムアウトして接続が閉じられた場合です。この場合は、通常、次のいずれかのエラーコードを受け取ります (受け取るエラーコードはオペレーティングシステムによって異なります)。

エラーコード	説明
<code>CR_SERVER_GONE_ERROR</code>	クライアントはサーバーに問い合わせを送信できませんでした。
<code>CR_SERVER_LOST</code>	クライアントはサーバーに書き込むときにエラーを受け取りませんでした。問い合わせに対する完全な応答 (または何らかの応答) が得られませんでした。

デフォルトでは、何も発生しなかった場合、サーバーは 8 時間後に接続を閉じます。この時間制限を変更するには、mysqld を起動するときに `wait_timeout` 変数を設定します。セクション 5.1.8 「`サーバーシステム変数`」を参照してください。

スクリプトがある場合、クライアントが自動再接続を行うには、クエリーを再発行する必要があるだけです。これは、クライアントの自動再接続を有効にしている (mysql コマンド行クライアントのデフォルト) ことが前提です。

「MySQL サーバーが存在しなくなりました」というエラーのほかの一般的な原因を次に示します。

- ユーザー (またはデータベース管理者) が実行中のスレッドを `KILL` ステートメントまたは `mysqladmin kill` コマンドを使用して強制終了した。
- サーバーへの接続を閉じたあとにクエリーを実行しようとした。これはアプリケーションの論理エラーを示しており、修正する必要があります。

- 別のホスト上で実行されているクライアントアプリケーションに、そのホストから MySQL サーバーに接続するために必要な権限がない。
- クライアント側で TCP/IP 接続がタイムアウトした。これは、コマンド `mysql_options(..., MYSQL_OPT_READ_TIMEOUT,...)` または `mysql_options(..., MYSQL_OPT_WRITE_TIMEOUT,...)` を使用している場合に発生することがあります。この場合は、タイムアウト値を増やすと問題が解決されることがあります。
- サーバー側でタイムアウトが発生し、クライアントの自動再接続が無効にされている (MySQL 構造体の `reconnect` フラグが 0 と等しい)。
- Windows クライアントを使用していて、コマンドが発行される前にサーバーによって接続がドロップされた (おそらく `wait_timeout` が期限切れになったため)。

Windows での問題は、TCP/IP 接続をサーバーに書き込むときに MySQL が OS からエラーを受け取らず、接続から応答を読み取ろうとしたときにエラーを受け取ることです。

この問題の解決策は、最後のクエリーから長い時間がたっている場合に接続に対して `mysql_ping()` を実行するか (これは Connector/ODBC が行なっていることです)、`mysqld` サーバーで `wait_timeout` に事実上タイムアウトすることがないような高い値を設定することです。

- 間違ったクエリーまたは長すぎるクエリーをサーバーに送信した場合にも、これらのエラーを受け取ることがあります。`mysqld` が長すぎるパケットまたは順序が間違っているパケットを受け取ると、クライアントで何らかの問題が発生していると見なして、接続を閉じます。大きなクエリーが必要な場合 (たとえば、大きな BLOB カラムを操作している場合)、クエリーの制限を緩和するには、サーバーの `max_allowed_packet` 変数 (デフォルト値は 64M バイト) を設定します。クライアント側の最大パケットサイズも増やす必要がある場合もあります。パケットサイズの設定については、[セクション B.3.2.8 「パケットが大きすぎます」](#) を参照してください。

多数の行を挿入する `INSERT` ステートメントまたは `REPLACE` ステートメントも、これらの種類のエラーの原因となることがあります。これらのステートメントのいずれかは、挿入される行数に関係なくサーバーに単一の要求を送信します。このため、1つの `INSERT` または `REPLACE` で送信される行数を減らすことによって多くの場合エラーを防ぐことができます。

- ホスト名のルックアップが失敗した場合に、このエラーが発生することもあります (たとえば、サーバーまたはネットワークが依存する DNS サーバーが停止した場合)。これは、MySQL が名前解決についてホストシステムに依存しているが、それが動作しているかどうかは知ることができないためです。MySQL が認識している範囲では、この問題はほかのネットワークタイムアウトと区別が付きません。

`skip_networking` システム変数を有効にして MySQL を起動すると、`MySQL server has gone away` エラーが表示されることもあります。

MySQL のポート (デフォルトは 3306) がファイアウォールによってブロックされていて、MySQL サーバーへのすべての接続が遮断される場合、このエラーの原因となることがある別のネットワークの問題が発生します。

- アプリケーションで複数の子プロセスがフォークされ、すべての子プロセスが MySQL サーバーへの同じ接続を使用しようとした場合に、このエラーが発生することもあります。これは、各子プロセスが個別の接続を使用することによって防ぐことができます。
- クエリーの実行中にサーバーが停止するバグが発生した。

MySQL サーバーが停止して再起動されたかどうかを確認するには、`mysqladmin version` を実行してサーバーの稼働時間を検査します。`mysqld` がクラッシュして再起動されたためにクライアント接続が切断された場合は、そのクラッシュの原因を見つけることに集中してください。クエリーを再び発行するとサーバーが再び強制終了されるかどうかを確認することから始めます。[セクション B.3.3.3 「MySQL が繰り返しクラッシュする場合の対処方法」](#) を参照してください。

失われた接続の詳細を取得するには、`log_error_verbosity` システム変数を 3 に設定して `mysqld` を起動します。これにより、切断メッセージの一部が `hostname.err` ファイルに記録されます。[セクション 5.4.2 「エラーログ」](#) を参照してください。

この問題に関するバグレポートを作成する場合は、次の情報を含めてください。

- MySQL サーバーが停止したかどうかを示します。これに関する情報はサーバーのエラーログにあります。[セクション B.3.3.3 「MySQL が繰り返しクラッシュする場合の対処方法」](#) を参照してください。

- 特定のクエリーによって `mysqld` が強制終了し、クエリーを実行する前に関係するテーブルが `CHECK TABLE` でチェックされていた場合は、再現可能なテストケースを提供してください。 [セクション5.9「MySQL のデバッグ」](#) を参照してください。
- MySQL サーバーの `wait_timeout` システム変数の値 (`mysqldadmin variables` を使用すると、この変数の値を取得できません。)
- 一般クエリーログを有効にして `mysqld` を実行し、問題のクエリーがログに出力されるかどうかを確認することを試みましたか。([セクション5.4.3「一般クエリーログ」](#) を参照してください。)

[セクションB.3.2.9「通信エラーおよび中止された接続」](#) および [セクション1.6「質問またはバグをレポートする方法」](#) も参照してください。

B.3.2.8 パケットが大きすぎます

通信パケットは、MySQL サーバーに送信される単一の SQL ステートメント、クライアントに送信される単一行、またはソースレプリケーションサーバーからレプリカに送信されるバイナリログイベントです。

MySQL 8.0 Server およびクライアント間で転送可能なパケットの最大サイズは 1G バイトです。

MySQL クライアントまたは `mysqld` サーバーが `max_allowed_packet` バイトより大きいパケットを受け取ると、`ER_NET_PACKET_TOO_LARGE` エラーが発行され、接続が失われます。一部のクライアントでは、パケットが大きすぎる場合、「クエリー中に MySQL サーバーへの接続が失われました」というエラーを受け取ることもあります。

クライアントとサーバーの両方にそれぞれ `max_allowed_packet` 変数があるため、大きなパケットを処理する場合は、クライアントとサーバーの両方のこの変数を増やす必要があります。

`mysql` クライアントプログラムを使用している場合、`max_allowed_packet` 変数のデフォルトは 16M バイトです。大きな値を設定するには、`mysql` を次のように起動します。

```
shell> mysql --max_allowed_packet=32M
```

これにより、パケットサイズが 32M バイトに設定されます。

サーバーのデフォルトの `max_allowed_packet` 値は 64M バイトです。サーバーが大きなクエリーを処理する必要がある場合 (たとえば、大きい `BLOB` カラムを操作している場合) は、この値を増やすことができます。たとえば、この変数に 128M バイトを設定するには、サーバーを次のように起動します。

```
shell> mysqld --max_allowed_packet=128M
```

オプションファイルを使用して `max_allowed_packet` を設定することもできます。たとえば、サーバー側のサイズを 128M バイトに設定するには、次の行をオプションファイルに追加します。

```
[mysqld]  
max_allowed_packet=128M
```

追加のメモリーは必要なときのみ割り当てられるため、この変数の値を増やしておく安全です。たとえば、`mysqld` が追加のメモリーを割り当てるのは、長いクエリーが発行された場合、または `mysqld` が大きな結果行を返す必要がある場合のみです。この変数のデフォルト値が小さいのは、クライアントとサーバーの間の不正なパケットを捕捉するための予防措置であり、誤って大きなパケットが使用されてメモリー不足にならないようにするためでもあります。

大きい `BLOB` 値を使用しているが、そのクエリーを処理するための十分なメモリーへのアクセスを `mysqld` に与えていない場合にも、大きいパケットに関する予期しない問題が発生することがあります。これに当てはまると思われる場合は、`mysqld_safe` スクリプトの先頭に `ulimit -d 256000` を追加して、`mysqld` を再起動してください。

B.3.2.9 通信エラーおよび中止された接続

通信エラー、中止された接続などの接続の問題が発生した場合は、次の情報ソースを使用して問題を診断してください。

- エラーログ。 [セクション5.4.2「エラーログ」](#) を参照してください。

- 一般クエリーログ。 [セクション5.4.3「一般クエリーログ」](#) を参照してください。
- `Aborted_xxx` ステータス変数および `Connection_errors_xxx` ステータス変数。 [セクション5.1.10「サーバーステータス変数」](#) を参照してください。
- ホストキャッシュ。パフォーマンススキーマ `host_cache` テーブルを使用してアクセスできます。 [セクション5.1.12.3「DNS ルックアップとホストキャッシュ」](#) および [セクション27.12.19.5「host_cache テーブル」](#) を参照してください。

`log_error_verbosity` システム変数が 3 に設定されている場合、次のようなメッセージがエラーログに記録されることがあります:

```
[Note] Aborted connection 854 to db: 'employees' user: 'josh'
```

クライアントも接続できない場合、サーバーは `Aborted_connects` ステータス変数をインクリメントします。接続の失敗は、次の原因で発生することがあります。

- クライアントがデータベースにアクセスしようとしたが、データベースに対する権限がありません。
- クライアントが不正なパスワードを使用している。
- パケットに正しい情報が含まれていない。
- 接続パケットの取得には `connect_timeout` 秒より長くかかります。 [セクション5.1.8「サーバーシステム変数」](#) を参照してください。

これらのことが発生した場合は、何かがサーバーに侵入しようとしていることを示している可能性があります。一般クエリーログが有効になっている場合は、これらのタイプの問題に関するメッセージがログに記録されます。

クライアントが正常に接続したが、その後不適切に切断または終了した場合、サーバーは `Aborted_clients` ステータス変数をインクリメントし、「**接続が中止されました**」というメッセージをエラーログに記録します。この原因は次のいずれかです。

- クライアントプログラムが、終了する前に `mysql_close()` を呼び出さなかった。
- クライアントが、サーバーに要求を発行せずに `wait_timeout` 秒または `interactive_timeout` 秒を超えてスリープしている。 [セクション5.1.8「サーバーシステム変数」](#) を参照してください。
- クライアントプログラムがデータ転送中に突然終了した。

中断された接続または中断されたクライアントに関するその他の問題の理由は次のとおりです:

- `max_allowed_packet` 変数の値が小さすぎるか、クエリーが `mysqld` 用に割り当てられているメモリーより大きいメモリーの領域を要求した。 [セクションB.3.2.8「パケットが大きすぎます」](#) を参照してください。
- Linux で半二重および全二重の両方の Ethernet プロトコルが使用された。一部の Linux イーサネットドライバにはこのバグがあります。FTP を使用して、クライアントマシンとサーバーマシン間で大きなファイルを転送することによって、このバグをテストしてください。転送がバースト-ポーズ-バースト-ポーズモードになる場合は、Linux の二重化シンドロームに陥っています。ネットワークカードとハブ/スイッチの二重化モードを全二重または半二重に切り替えて結果をテストし、最適な設定を判別します。
- 読み取りで中断が発生するスレッドライブラリの問題。
- 不適切に構成されている TCP/IP。
- 障害のある Ethernet、ハブ、スイッチ、ケーブルなど。これは、ハードウェアを交換することによってのみ適切に診断できます。

[セクションB.3.2.7「MySQL サーバーが存在しなくなりました」](#) も参照してください。

B.3.2.10 テーブルが満杯です

テーブルが満杯であるというエラーが発生した場合は、ディスクが満杯であるか、テーブルが最大サイズに達した可能性があります。MySQL データベースの事実上の最大テーブルサイズは、通常、MySQL の内部制限ではなくオペ

レーティングシステムのファイルサイズに関する制約によって判断します。 [セクション8.4.6「テーブルサイズの制限」](#)を参照してください。

B.3.2.11 ファイルを作成/書き込みできない

一部のクエリーで次のタイプのエラーを受け取る場合は、MySQL が一時ディレクトリに結果セットの一時ファイルを作成できないことを意味します。

```
Can't create/write to file '\sqla3fe_0.ism'.
```

上記のエラーは Windows での一般的なメッセージです。UNIX のメッセージも似ています。

解決策の 1 つは、`--tmpdir` オプションを指定して `mysqld` を起動するか、オプションファイルの `[mysqld]` セクションにこのオプションを追加することです。たとえば、`C:\temp` ディレクトリを指定するには、次の行を使用します。

```
[mysqld]  
tmpdir=C:/temp
```

`C:\temp` ディレクトリが存在していて、MySQL サーバーが書き込むための十分な領域がある必要があります。 [セクション4.2.2.2「オプションファイルの使用」](#)を参照してください。

このエラーの別の原因は権限の問題です。MySQL サーバーが `tmpdir` ディレクトリに書き込めることを確認してください。

`pererror` で表示されるエラーコードも確認します。サーバーがテーブルに書き込むことができない原因の 1 つは、ファイルシステムが満杯であるためです。

```
shell> pererror 28  
OS error code 28: No space left on device
```

起動時に次のタイプのエラーを受け取る場合は、データファイルの格納に使用されるファイルシステムまたはディレクトリが書き込み保護されていることを示しています。書き込みエラーがテストファイルに対するものであれば、このエラーは重大ではなく、無視しても安全です。

```
Can't create test file /usr/local/mysql/data/master.lower-test
```

B.3.2.12 コマンドは同期されていません

クライアントのコードで「[コマンドは同期されていません。このコマンドは現在実行できません](#)」というメッセージを受け取る場合は、クライアント関数を間違った順序で呼び出しています。

たとえば、これは、`mysql_free_result()` を呼び出す前に、`mysql_use_result()` を使用して、新しいクエリーを実行しようとした場合に発生することがあります。これは、データを返す 2 つのクエリーの間に `mysql_use_result()` または `mysql_store_result()` を呼び出さずに実行した場合にも発生することがあります。

B.3.2.13 ユーザーを無視します

次のエラーが表示される場合は、`mysqld` が起動されたときまたは付与テーブルをリロードしたときに、`user` テーブルで不正なパスワードを持つアカウントが見つかったことを意味します。

```
ユーザー 'some_user'@'some_host' のパスワードが不正です。ユーザーを無視します
```

その結果、このアカウントは権限システムによって無視されます。この問題を解決するには、新しい有効なパスワードをアカウントに割り当てます。

B.3.2.14 表 'tbl_name' は存在しません

次のエラーが表示される場合は、通常、指定された名前のデフォルトデータベースにテーブルが存在しないことを意味します。

```
Table 'tbl_name' doesn't exist  
Can't find file: 'tbl_name' (errno: 2)
```

テーブルは存在しているが、誤った名前参照している場合もあります。

- MySQL ではディレクトリとファイルを使用してデータベースとテーブルを格納するため、大/小文字が区別されるファイル名を持つファイルシステム上にある場合、データベースとテーブルの名前は大小文字が区別されます。
- Windows など、大文字と小文字が区別されないファイルシステムの場合でも、クエリー内の特定のテーブルへのすべての参照で同じ大文字と小文字を使用する必要があります。

デフォルトデータベースにあるテーブルを確認するには、`SHOW TABLES` を使用します。 [セクション13.7.7「SHOW ステートメント」](#) を参照してください。

B.3.2.15 文字セットを初期化できません

文字セットの問題がある場合は、次のようなエラーが表示されることがあります。

```
MySQL Connection Failed: Can't initialize character set charset_name
```

このエラーには次のいずれかの原因がある可能性があります。

- 文字セットがマルチバイト文字セットであり、クライアントでその文字セットがサポートされていない。この場合、`-DDEFAULT_CHARSET=charset_name` オプションを指定して `CMake` を実行し、クライアントを再コンパイルする必要があります。 [セクション2.9.7「MySQL ソース構成オプション」](#) を参照してください。

すべての標準 MySQL バイナリは、すべてのマルチバイト文字セットをサポートしてコンパイルされます。

- 文字セットは `mysqld` にコンパイルされない単純な文字セットであり、文字セットの定義ファイルがクライアントが予期している場所にありません。

この場合は、次のいずれかの方法を使用して問題を解決する必要があります。

- その文字セットがサポートされるようにクライアントを再コンパイルします。 [セクション2.9.7「MySQL ソース構成オプション」](#) を参照してください。
- 文字セットの定義ファイルがあるディレクトリをクライアントに指定します。多くのクライアントでは、`--character-sets-dir` オプションを指定することによってこれを行うことができます。
- 文字定義ファイルをクライアントが予期しているパスにコピーします。

B.3.2.16 ファイルが見つからず同様のエラーが発生しました

MySQL から `errno 23` または `errno 24` で `ERROR 'file_name' not found (errno: 23)`、`Can't open file: file_name (errno: 24)`、またはその他のエラーが発生した場合は、MySQL サーバーに十分なファイル記述子が割り当てられていないことを意味します。 `percona` ユーティリティを使用すると、エラー番号の意味の説明を取得できます。

```
shell> perror 23
OS error code 23: File table overflow
shell> perror 24
OS error code 24: Too many open files
shell> perror 11
OS error code 11: Resource temporarily unavailable
```

ここでの問題は、`mysqld` が同時にオープンしたままにしようとしているファイルが多すぎることです。一度に多数のファイルをオープンしないように `mysqld` に通知するか、`mysqld` が使用できるファイルディスクリプタの数を増やします。

一度にオープンするファイル数を少なくするように `mysqld` に通知するには、`table_open_cache` システム変数の値 (デフォルト値は 64) を減らすことによってテーブルキャッシュを小さくします。 [セクション8.4.3.1「MySQL でのテーブルのオープンとクローズの方法」](#) で説明されているように、状況によっては、サーバーがキャッシュサイズを一時的に拡張しようとする可能性があるため、これによってファイルディスクリプタの不足を完全に防ぐことはできません。 `max_connections` の値を減らすことによっても、オープンファイルの数が減少します (デフォルト値は 100)。

`mysqld` が使用できるファイルディスクリプタの数を変更するには、`mysqld_safe` に `--open-files-limit` オプションを使用するか、`open_files_limit` システム変数を設定します。 [セクション5.1.8「サーバーシステム変数」](#) を参照してください。これらの値を設定するもっとも簡単な方法は、オプションファイルにオプションを追加することです。 [セクション4.2.2.2「オプションファイルの使用」](#) を参照してください。オープンファイルの制限の設定をサポートしてい

古いバージョンの `mysqld` を使用している場合は、`mysqld_safe` スクリプトを編集できます。このスクリプトには、コメントアウトされた行 `ulimit -n 256` があります。# 文字を削除してこの行をコメント解除し、数字 `256` を変更して、`mysqld` が使用できるファイルディスクリプタの数を設定します。

`--open-files-limit` および `ulimit` を使用すると、ファイルディスクリプタの数を増やすことができますが、オペレーティングシステムが課している制限が上限となります。`mysqld_safe` または `mysqld` を `root` として起動した場合にのみオーバーライドできる「堅固な」制限もあります(この場合、起動後に `root` として実行され続けないように、`--user` オプションを指定してサーバーを起動する必要があります)。各プロセスで使用できるファイルディスクリプタの数に関するオペレーティングシステムの制限を緩める必要がある場合は、システムのドキュメントを参照してください。

注記

`tcsh` シェルを実行している場合、`ulimit` は機能しません。また、`tcsh` では、現在の制限を問い合わせたときに不正な値が報告されます。この場合は、`sh` を使用して `mysqld_safe` を起動してください。

B.3.2.17 テーブルの破損の問題

`myisam_recover_options` システム変数を設定して `mysqld` を起動した場合、MyISAM テーブルが正しくクローズされていないまたはクラッシュとマークされていれば、MySQL によって自動的にチェックされ、修復が試行されます。これが発生した場合、MySQL は `hostname.err` ファイルに「警告: テーブル ... をチェックしています」と書き込み、テーブルを修復する必要がある場合は、「警告: テーブルを修復しています」がそのあとに書き込まれます。これらのエラーを多数受け取り、その直前に予期しない `mysqld` の停止がなかった場合は、何らかの問題があるため、さらに調査する必要があります。

サーバーは、MyISAM テーブルの破損を検出すると、ソースファイルの名前や行番号、テーブルにアクセスするスレッドのリストなどの追加情報をエラーログに書き込みます。たとえば、「`thread_id=1` からエラーを受け取りました。`mi_dynrec.c:368`」です。これは、バグレポートに含めると役に立つ情報です。

セクション5.1.7「サーバーコマンドオプション」およびセクション5.9.1.7「テーブルが破損した場合のテストケースの作成」も参照してください。

B.3.3 管理関連の問題

B.3.3.1 ファイル権限の問題

ファイル権限に問題がある場合、`mysqld` の起動時に `UMASK` または `UMASK_DIR` 環境変数が正しく設定されないことがあります。たとえば、`mysqld` では、テーブルの作成時に次のエラーメッセージが発行される場合があります:

```
ERROR: Can't find file: 'path/with/file_name' (Errcode: 13)
```

`UMASK` および `UMASK_DIR` のデフォルト値は、それぞれ `0640` および `0750` です。`mysqld` では、`UMASK` または `UMASK_DIR` の値がゼロで始まる場合、その値は 8 進数であるとみなされます。たとえば、`0600` オクタルは `384` 桁であるため、`UMASK=0600` の設定は `UMASK=384` と同等です。

`mysqld_safe` を使用して `mysqld` を起動する場合は、デフォルトの `UMASK` 値を次のように変更します:

```
UMASK=384 # = 600 in octal
export UMASK
mysqld_safe &
```

注記

`UMASK` を考慮しない `mysqld_safe` を使用して `mysqld` を起動すると、エラーログファイルに例外が適用されます: `mysqld` の起動前にエラーログファイルが存在しない場合、`mysqld_safe` はエラーログファイルを作成し、`mysqld_safe` は厳密な値 `0137` に設定された `umask` を使用します。これが適切でない場合は、`mysqld_safe` を実行する前に、目的のアクセスモードでエラーファイルを手動で作成します。

デフォルトでは、`mysqld` はアクセス権限の値が `0750` のデータベースディレクトリを作成します。この動作を変更するには、`UMASK_DIR` 変数を設定します。この値を設定すると、新しいディレクトリは `UMASK` 値と `UMASK_DIR`

値を組み合わせたもので作成されます。たとえば、グループにすべての新しいディレクトリへのアクセス権を付与するには、次のように `mysqld_safe` を起動します:

```
UMASK_DIR=504 # = 770 in octal
export UMASK_DIR
mysqld_safe &
```

追加の詳細については、[セクション4.9「環境変数」](#)を参照してください。

B.3.3.2 root のパスワードをリセットする方法

MySQL の `root` パスワードを割り当てていない場合、サーバーは `root` として接続するためのパスワードをまったく必要としません。ただし、これはセキュリティー保護されていません。パスワードの割当て手順は、[セクション2.10.4「初期 MySQL アカウントの保護」](#)を参照してください。

`root` パスワードがわかっている、それを変更する場合は、[セクション13.7.1.1「ALTER USER ステートメント」](#) および [セクション13.7.1.10「SET PASSWORD ステートメント」](#) を参照してください。

以前に `root` パスワードを割り当てたが、忘れた場合は、新しいパスワードを割り当てることができます。次のセクションでは、Windows、Unix および Unix に似たシステムの手順と、任意のシステムに適用される一般的な手順について説明します。

root のパスワードのリセット: Windows システム

Windows では、次の手順を使用して MySQL `'root'@'localhost'` アカウントのパスワードをリセットします。別のホスト名部分を持つ `root` アカウントのパスワードを変更するには、そのホスト名を使用するように指示を変更します。

1. Administrator としてシステムにログオンします。
2. MySQL サーバーが実行されている場合は停止します。Windows サービスとして実行されているサーバーの場合は、サービスマネージャーを開きます (「スタート」メニューから、「コントロール パネル」、「管理ツール」、「サービス」の順に選択します)。リスト内で MySQL サービスを見つけて、それを停止します。

サーバーがサービスとして実行されていない場合は、タスクマネージャーを使用して強制的に停止する必要があることがあります。

3. 単一行にパスワード割当てステートメントを含むテキストファイルを作成します。パスワードを使用するパスワードに置き換えます。

```
ALTER USER 'root'@'localhost' IDENTIFIED BY 'MyNewPass';
```

4. ファイルを保存します。この例では、ファイルに `C:\mysql-init.txt` という名前を付けることを前提としています。
5. コンソールウィンドウを開いて、コマンドプロンプトを表示します (「スタート」メニューから「ファイル名を指定して実行」を選択し、実行するコマンドとして `cmd` を入力します)。
6. ファイルに名前を付けるように `init_file` システム変数を設定して MySQL サーバーを起動します (オプション値のバックスラッシュが二重になることに注意してください):

```
C:\> cd "C:\Program Files\MySQL\MySQL Server 8.0\bin"
C:\> mysqld --init-file=C:\mysql-init.txt
```

MySQL を別の場所にインストールした場合は、`cd` コマンドを適宜調整します。

サーバーは、起動時に `init_file` システム変数で指定されたファイルの内容を実行し、`'root'@'localhost'` アカウントのパスワードを変更します。

サーバー出力をログファイルではなくコンソールウィンドウに表示するには、`mysqld` コマンドに `--console` オプションを追加します。

MySQL インストールウィザードを使用して MySQL をインストールした場合、`--defaults-file` オプションの指定が必要になることがあります。例:

```
C:\> mysqld
--defaults-file="C:\ProgramData\MySQL\MySQL Server 8.0\my.ini"
--init-file=C:\mysql-init.txt
```


適切な `--defaults-file` 設定はサービスマネージャーを使用して見つけることができます(「スタート」メニューから「コントロールパネル」、「管理ツール」、「サービス」の順に選択します)。リスト内で MySQL サービスを見つけて、それを右クリックし、「プロパティ」オプションを選択します。「実行ファイルのパス」フィールドに `--defaults-file` 設定が含まれています。

7. サーバーが正常に起動されたら `C:\mysql-init.txt` を削除します。

新しいパスワードを使用して `root` として MySQL サーバーに接続できるようになりました。MySQL サーバーを停止し、通常どおりに再起動します。サーバーをサービスとして実行している場合は、Windows の「サービス」ウィンドウから開始します。サーバーを手動で起動している場合は、通常使用するコマンドを使用してください。

root パスワードのリセット: Unix および Unix- 類似システム

Unix では、次の手順を使用して MySQL `'root'@'localhost'` アカウントのパスワードをリセットします。別のホスト名部分を持つ `root` アカウントのパスワードを変更するには、そのホスト名を使用するように指示を変更します。

この手順では、通常実行に使用する Unix ログインアカウントから MySQL サーバーを起動することを前提としています。たとえば、`mysql` のログインアカウントを使用してサーバーを実行する場合は、この手順を使用する前に `mysql` としてログインしてください。または、`root` としてログインすることもできますが、この場合は `--user=mysql` オプションを指定して `mysqld` を起動する必要があります。 `--user=mysql` を使用せずに `root` としてサーバーを起動した場合、サーバーは `root` が所有するファイル(ログファイルなど)をデータディレクトリに作成することがあり、以降のサーバーの起動で権限関連の問題の原因となることがあります。その場合は、ファイルの所有権を `mysql` に変更するか、削除する必要があります。

1. MySQL サーバーを実行する Unix ユーザー(`mysql` など)としてシステムにログオンします。
2. MySQL サーバーが実行されている場合は停止します。サーバーのプロセス ID が含まれている `.pid` ファイルを見つけます。このファイルの正確な場所と名前は、配布、ホスト名、および構成によって異なります。一般的な場所は、`/var/lib/mysql/`、`/var/run/mysqld/`、および `/usr/local/mysql/data/` です。通常、ファイル名には `.pid` という拡張子が付けられており、`mysqld` またはシステムのホスト名で始まります。

通常の `kill` (`kill -9` ではなく)を `mysqld` プロセスに送信して、MySQL サーバーを停止します。次のコマンドで、`.pid` ファイルの実際のパス名を使用します:

```
shell> kill `cat /mysql-data-directory/host_name.pid`
```

`cat` コマンドには逆引用符(通常の引用符ではなく)を使用します。これにより、`cat` の出力が `kill` コマンドに指定されます。

3. 単一行にパスワード割当てステートメントを含むテキストファイルを作成します。パスワードを使用するパスワードに置き換えます。

```
ALTER USER 'root'@'localhost' IDENTIFIED BY 'MyNewPass';
```

4. ファイルを保存します。この例では、ファイルに `/home/me/mysql-init` という名前を付けることを前提としています。ファイルにはパスワードが含まれているため、他のユーザーが読み取れる場所に保存しないでください。`mysql` (サーバーが実行されるときユーザー)としてログインしていない場合は、`mysql` による読み取りを許可する権限がファイルに設定されていることを確認してください。

5. ファイルに名前を付けるように `init_file` システム変数を設定して、MySQL サーバーを起動します:

```
shell> mysqld --init-file=/home/me/mysql-init &
```

サーバーは、起動時に `init_file` システム変数で指定されたファイルの内容を実行し、`'root'@'localhost'` アカウントのパスワードを変更します。

サーバーの通常の起動方法によっては、その他のオプションも必要になる場合があります。たとえば、`init_file` 引数の前に `--defaults-file` が必要な場合があります。

6. サーバーが正常に起動されたら `/home/me/mysql-init` を削除します。

新しいパスワードを使用して `root` として MySQL サーバーに接続できるようになりました。サーバーを停止して、通常モードで再起動します。

root のパスワードのリセット: 一般的な手順

前述のセクションでは、Windows、Unix および Unix に似たシステム専用のパスワードリセット手順について説明します。または、どのプラットフォームでも、`mysql` クライアントを使用してパスワードをリセットできます (ただし、このアプローチの安全性は低くなります):

1. 必要に応じて MySQL サーバーを停止し、`--skip-grant-tables` オプションを使用して再起動します。これにより、すべてのユーザーがパスワードなしですべての権限で接続できるようになり、`ALTER USER` や `SET PASSWORD` などのアカウント管理ステートメントが無効になります。これはセキュアではないため、`--skip-grant-tables` オプションを使用してサーバーを起動すると、`skip_networking` を有効にしてリモート接続も無効になります。
2. `mysql` クライアントを使用して MySQL サーバーに接続します。サーバーは `--skip-grant-tables` で起動されたため、パスワードは必要ありません:

```
shell> mysql
```

3. `mysql` クライアントで、`account-management` ステートメントが機能するように付与テーブルをリロードするようにサーバーに指示します:

```
mysql> FLUSH PRIVILEGES;
```

次に、`'root'@'localhost'` アカウントのパスワードを変更します。パスワードを使用するパスワードに置き換えます。別のホスト名部分を持つ `root` アカウントのパスワードを変更するには、そのホスト名を使用するように指示を変更します。

```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'MyNewPass';
```

新しいパスワードを使用して `root` として MySQL サーバーに接続できるようになりました。サーバーを停止し、通常どおりに再起動します (`--skip-grant-tables` オプションを指定せず、`skip_networking` システム変数を有効にせずに)。

B.3.3.3 MySQL が繰り返しクラッシュする場合の対処方法

各 MySQL バージョンは、リリース前に多くのプラットフォームでテストされています。これは、MySQL にバグがないということではありませんが、バグがあってもごく少数であり、見つかることはまれです。問題が発生した場合は、システムがクラッシュした正確な原因を探ることが常に役に立ちます。問題の修正が迅速に得られる可能性が高まるためです。

まず、問題は `mysqld` サーバーが停止したことであるかどうか、またはクライアントに関連しているかどうかを判断してください。 `mysqld` サーバーが稼働している時間を確認するには、`mysqladmin version` を実行します。 `mysqld` が停止して再起動された場合は、サーバーのエラーログを確認すると原因が見つかる可能性があります。 [セクション 5.4.2 「エラーログ」](#) を参照してください。

一部のシステムでは、`mysqld` が停止した場所のスタックトレースがエラーログに記録されます。エラーログに書き込まれる変数値は、常に 100% 正しいとは限りません。

予期しないサーバーイグジットの多くは、データファイルまたはインデックスファイルが破損していることが原因です。MySQL は、各 SQL ステートメントの実行後、クライアントに結果を通知する前に、ディスク上のファイルを書き込みシステムコールを使用して更新します。(`delay_key_write` システム変数を有効にして実行している場合、データファイルは書き込まれますがインデックスファイルは書き込まれません。) これは、`mysqld` がクラッシュしてもデータファイルのコンテンツは安全であることを意味します。フラッシュされていないデータがオペレーティングシステムによってディスクに書き込まれることが保証されているためです。各 SQL ステートメントのあとに MySQL がすべてのデータをディスクにフラッシュするには、`--flush` オプションを指定して `mysqld` を再起動します。

これは、通常、次のいずれかが発生していなければ、データが損なわれたテーブルができることはないことを意味します。

- MySQL サーバーまたはサーバーのホストが更新中に強制終了された。
- 更新の途中で `mysqld` が停止するバグが見つかった。
- 一部の外部プログラムは、テーブルを適切にロックせずに、`mysqld` と同時にデータファイルまたはインデックスファイルを操作しています。

- 適切なファイルシステムのロック (通常は `lockd` ロックマネージャーによって処理されます) をサポートしていないシステムで同じデータディレクトリを使用して多数の `mysqld` サーバーを実行しているか、外部ロックを無効にして複数のサーバーを実行している。
- 大きく破損したデータが含まれているクラッシュしたデータファイルまたはインデックスファイルがあり、`mysqld` が混乱した。
- データストレージのコードにバグが見つかった。可能性は低いですがあり得ることです。この場合は、修復されたテーブルのコピーに対して `ALTER TABLE` を使用することによって、別のエンジンへのストレージエンジンの変更を試みることができます。

何らかのクラッシュが発生している理由を知ることは非常に困難であるため、まず他のユーザーのために機能するものが予期しない終了になるかどうかを確認してみてください。次のことを試してください。

- `mysqladmin shutdown` を使用して `mysqld` サーバーを停止し、データディレクトリから `myisamchk --silent --force */*.MYI` を実行して、すべての MyISAM テーブルを確認し、`mysqld` を再起動します。これにより、クリーンな状態から実行していることが保証されます。第5章「MySQL サーバーの管理」を参照してください。
- 一般クエリログを有効にして (セクション5.4.3「一般クエリログ」を参照してください) `mysqld` を起動します。ログに書き込まれた情報から、特定のクエリによってサーバーが強制終了したかどうかを判断してください。すべてのバグの95%は特定のクエリに関連しています。通常、これは、サーバーが再起動される前のログファイルの最後のクエリのいずれかです。セクション5.4.3「一般クエリログ」を参照してください。クエリを発行する前にすべてのテーブルを確認しても、特定のクエリによってMySQLが繰り返し強制終了される場合は、バグとして判断されましたのでバグレポートを送信してください。セクション1.6「質問またはバグをレポートする方法」を参照してください。
- 問題を再現するために使用できるテストケースを作成してください。セクション5.9「MySQL のデバッグ」を参照してください。
- `fork_big.pl` スクリプトを試してください。(ソース配布の `tests` ディレクトリにあります。)
- MySQL をデバッグ用に構成すると、何らかの問題がある場合に、考えられるエラーに関する情報を収集しやすくなります。`-DWITH_DEBUG=1` オプションを指定してMySQLを再構成し、`CMake` を実行して再コンパイルします。セクション5.9「MySQL のデバッグ」を参照してください。
- オペレーティングシステムに最新のパッチが適用されていることを確認してください。
- `--skip-external-locking` オプションを使用して `mysqld` を起動します。一部のシステムでは `lockd` ロックマネージャーが正常に動作しません。`--skip-external-locking` オプションは `mysqld` に外部ロックを使用しないように通知します。(これは、同じデータディレクトリで2つの `mysqld` サーバーは実行できず、`myisamchk` を使用する場合は注意する必要があることを意味します。それでも、テストとしてこのオプションを試すことには価値があります。)
- `mysqld` は実行されているが応答しない場合は、`mysqladmin -u root processlist` を試してください。`mysqld` が応答しないように見えても、ハングアップしていないことがあります。すべての接続が使用されているか、内部ロックの問題である可能性があります。`mysqladmin -u root processlist` は、通常、これらの場合でも接続を確立することができ、現在の接続数およびそのステータスに関する役に立つ情報が表示されます。
- ほかのクエリを実行しているときに、別のウィンドウで `mysqladmin -i 5 status` コマンドまたは `mysqladmin -i 5 -r status` コマンドを実行して統計を生成します。
- 次の手順を試してください。
 - `gdb` (または別のデバッガ) から `mysqld` を起動します。セクション5.9「MySQL のデバッグ」を参照してください。
 - テストスクリプトを実行します。
 - もっとも低い3つのレベルでバックトレースおよびローカル変数を出力します。`gdb` でこれを行うには、`mysqld` が `gdb` 内でクラッシュしたときに、次のコマンドを使用します。

```
backtrace
info local
up
```

```
info local
up
info local
```

`gdb` で、`info threads` を使用して存在するスレッドを検査し、`thread N` (ここで `N` はスレッド ID です) を使用して特定のスレッドに切り替えることもできます。

- Perl スクリプトを使用してアプリケーションをシミュレートし、MySQL を強制的に終了または誤動作させてください。
- 通常のバグレポートを送信します。 [セクション1.6「質問またはバグをレポートする方法」](#) を参照してください。通常より詳しく記述してください。MySQL は多くのユーザーの環境では動作しているため、クラッシュは問題となっているコンピュータにのみ存在する何かによって発生している可能性があります (たとえば、特定のシステムライブラリに関連するエラー)。
- 動的長の行を含むテーブルで問題があり、`VARCHAR` カラム (`BLOB` カラムまたは `TEXT` カラムではなく) のみを使用している場合は、`ALTER TABLE` を使用して、すべての `VARCHAR` を `CHAR` に変更してみてください。これにより、MySQL が固定長の行を使用ようになります。固定長の行では若干追加の領域が使用されますが、破損に対してより耐性があります。

現在の動的行のコードは数年使用されており、問題はほとんど発生していませんが、動的長の行はその性質のためエラーが発生しやすい傾向があるので、この方法がうまくいくかどうかを試すのは良い考えである可能性があります。
- 問題を診断する場合は、ハードウェアの障害の可能性を考慮に入れてください。欠陥のあるハードウェアは、データ破損の原因となることがあります。ハードウェアをトラブルシューティングする場合は、メモリーおよびディスクのサブシステムに特に注意してください。

B.3.3.4 MySQL が満杯のディスクを処理する方法

このセクションでは、MySQL がディスク満杯エラー (「デバイスに領域が残っていない」など)、および割り当て超過エラー (「書き込みに失敗しました」、「ユーザーブロックの制限に達しました」など) に対処する方法について説明します。

このセクションは、`MyISAM` テーブルへの書き込みに関連しています。「行」および「レコード」への言及が「イベント」を意味すると理解する必要のあることを除き、バイナリログファイルおよびバイナリログインデックスファイルへの書き込みにも当てはまります。

ディスク満杯状態が発生すると、MySQL は次のことを行います。

- 現在の行を書き込むための十分な領域があるかどうかを 1 分おきに確認します。十分な領域がある場合は、何事もなかったかのように稼働し続けます。
- ディスク満杯状態について警告するエントリをログファイルに 10 分おきに書き込みます。

問題を軽減するには次のアクションを行います。

- 続行する場合は、すべてのレコードを挿入するための十分なディスク領域を解放する必要があるだけです。
- または、スレッドを中止するには `mysqladmin kill` を使用します。スレッドは次回ディスクがチェックされるとき (1 分以内) に中止されます。
- ほかのスレッドが、ディスク満杯状態の原因となったテーブルを待機している可能性があります。複数の「ロックされた」スレッドがある場合は、ディスク満杯状態を待機していた 1 つのスレッドを強制終了すると、ほかのスレッドが続行できるようになります。

前述の動作の例外は、`REPAIR TABLE` または `OPTIMIZE TABLE` を使用する場合、または `LOAD DATA` の後または `ALTER TABLE` ステートメントの後にインデックスがバッチで作成される場合です。これらのすべてのステートメントでは大きい一時ファイルが作成されることがあり、それがそのまま残された場合、システムのほかの部分で大きな問題となる可能性があります。MySQL がこれらのいずれかの操作を実行していて、ディスクが満杯になった場合は、大きい一時ファイルが削除され、テーブルがクラッシュとしてマークされます。例外は `ALTER TABLE` の場合で、古いテーブルは変更されないままになります。

B.3.3.5 MySQL が一時ファイルを格納する場所

UNIX では、MySQL は一時ファイルを格納するディレクトリのパス名として、`TMPDIR` 環境変数の値を使用します。`TMPDIR` が設定されていない場合、MySQL はシステムのデフォルトを使用します。通常、これは `/tmp`、`/var/tmp`、または `/usr/tmp` です。

Windows では、MySQL は `TMPDIR`、`TEMP`、および `TMP` 環境変数の値を順番にチェックします。MySQL は最初に見つかった設定されている変数を使用し、残りの変数はチェックしません。`TMPDIR`、`TEMP`、および `TMP` がいずれも設定されていない場合、MySQL は Windows システムのデフォルトを使用します。通常、これは `C:\windows\temp\` です。

一時ファイルディレクトリを含むファイルシステムが小さすぎる場合は、`mysqld --tmpdir` オプションを使用して、十分な領域があるファイルシステム内のディレクトリを指定できます。

`--tmpdir` オプションには、ラウンドロビン方式で使用される複数のパスのリストを設定できます。パスは、Unix ではコロン文字 (`:`) で区切り、Windows ではセミコロン文字 (`;`) で区切る必要があります。

注記

負荷を効果的に分散するには、これらのパスに同じディスクの個別のパーティションではなく、個別の物理ディスクを指定してください。

MySQL サーバーがレプリカとして機能している場合は、`slave_load_tmpdir` システム変数を設定して、`LOAD DATA` ステートメントのレプリケート時に一時ファイルを保持するための個別のディレクトリを指定できます。`LOAD DATA` の複製に使用される一時ファイルがマシンの再起動後も存続できるように、このディレクトリは (メモリーベースのファイルシステムではなく) ディスクベースのファイルシステム内にある必要があります。このディレクトリは、システム起動プロセス中にオペレーティングシステムによってクリアされるものはいけません。ただし、一時ファイルが削除されている場合は、再起動後にレプリケーションを続行できるようになりました。

MySQL は、`mysqld` が終了したら一時ファイルが削除されるようにしています。これがサポートされるプラットフォームでは (UNIX など)、ファイルをオープンしたあとにリンク解除することによってこれが行われます。この方法のデメリットは、名前がディレクトリのリストに表示されないことであり、一時ファイルディレクトリがあるファイルシステムを満杯にしている大きい一時ファイルが表示されません。(そのような場合は、`mysqld` に関連付けられている大きいファイルを識別するために、`IsOf +L1` が役に立つことがあります。)

通常、MySQL はソート (`ORDER BY` または `GROUP BY`) を行うときに、1 つまたは 2 つの一時ファイルを使用します。必要となる最大のディスク領域は次の式によって判別されます。

```
(length of what is sorted + sizeof(row pointer))
* number of matched rows
* 2
```

行ポインタのサイズは通常 4 バイトですが、大きいテーブルの場合は将来拡張される可能性があります。

一部のステートメントでは、MySQL によって、非表示ではなく、`#sql` で始まる名前を持つ一時 SQL テーブルが作成されます。

一部の `SELECT` クエリーでは、中間結果を保持するための一時 SQL テーブルが作成されます。

テーブルを再構築し、`ALGORITHM=INPLACE` 技術を使用してオンラインで実行されない DDL 操作では、元のテーブルと同じディレクトリに元のテーブルの一時コピーが作成されます。

オンライン DDL 操作では、同時 DML の記録に一時ログファイル、インデックスの作成時に一時ソートファイル、テーブルの再構築時に一時中間テーブルファイルを使用できます。詳細は、[セクション15.12.3「オンライン DDL 領域の要件」](#)を参照してください。

InnoDB のユーザー作成一時テーブルおよびディスク上の内部一時テーブルは、MySQL データディレクトリの `ibtmp1` という名前の一時テーブルスペースファイルに作成されます。詳細は、[セクション15.6.3.5「一時テーブルスペース」](#)を参照してください。

[セクション15.15.7「InnoDB INFORMATION_SCHEMA 一時テーブル情報テーブル」](#)も参照してください。

オプションの `EXTENDED` 修飾子を使用すると、失敗した `ALTER TABLE` ステートメントによって作成された非表示のテーブルが `SHOW TABLES` にリストされます。[セクション13.7.7.39「SHOW TABLES ステートメント」](#)を参照してください。

B.3.3.6 MySQL の UNIX ソケットファイルを保護または変更する方法

サーバーがローカルクライアントと通信するために使用する UNIX ソケットファイルのデフォルトの場所は、`/tmp/mysql.sock` です。(一部の配布形式ではディレクトリが異なる場合があります、たとえば RPM の場合は `/var/lib/mysql` です。)

UNIX の一部のバージョンでは、`/tmp` ディレクトリ内のファイル、または一時ファイルに使用されるほかの同様のディレクトリ内のファイルをだれでも削除できます。ソケットファイルがシステム上のそのようなディレクトリに配置されている場合は、問題となることがあります。

ほとんどのバージョンの UNIX では、`/tmp` ディレクトリを保護して、所有者またはスーパーユーザー (`root`) のみがファイルを削除できるようにすることができます。これを行うには、`root` としてログインして次のコマンドを使用することによって、`/tmp` ディレクトリに `スティッキービット` を設定します。

```
shell> chmod +t /tmp
```

`スティッキービット` が設定されたかどうかを確認するには、`ls -ld /tmp` を実行します。いちばん最後の権限文字が `t` である場合は、このビットが設定されています。

別の方法は、サーバーが UNIX ソケットファイルを作成する場所を変更することです。これを行う場合は、クライアントプログラムにもファイルの新しい場所を認識させてください。ファイルの場所を指定する方法はいくつかあります。

- グローバルまたはローカルのオプションファイルにパスを指定します。たとえば、次の行を `/etc/my.cnf` に指定します。

```
[mysqld]
socket=/path/to/socket

[client]
socket=/path/to/socket
```

[セクション4.2.2.2「オプションファイルの使用」](#)を参照してください。

- `mysqld_safe` およびクライアントプログラムを実行するときに、`--socket` オプションをコマンド行に指定します。
- `MYSQL_UNIX_PORT` 環境変数に UNIX ソケットファイルのパスを設定します。
- 別のデフォルトの UNIX ソケットファイルの場所を使用するように、ソースから MySQL を再コンパイルします。`CMake` を実行するときに `MYSQL_UNIX_ADDR` オプションでファイルへのパスを定義します。[セクション2.9.7「MySQL ソース構成オプション」](#)を参照してください。

新しいソケットの場所が動作しているかどうかをテストするには、次のコマンドを使用してサーバーに接続します。

```
shell> mysqladmin --socket=/path/to/socket version
```

B.3.3.7 タイムゾーンの問題

`SELECT NOW()` でローカルの時間ではなく UTC で値が返される問題がある場合は、サーバーに現在のタイムゾーンを通知する必要があります。`UNIX_TIMESTAMP()` が間違った値を返す場合も同様です。これは、サーバーが実行されている環境 (`mysqld_safe` や `mysql.server` など) で実行する必要があります。[セクション4.9「環境変数」](#)を参照してください。

サーバーのタイムゾーンを設定するには、`mysqld_safe` に `--timezone=timezone_name` オプションを指定します。`mysqld` を起動する前に、`TZ` 環境変数を設定することによって設定することもできます。

`--timezone` または `TZ` に許可される値は、システムによって異なります。許容可能な値を確認するには、オペレーティングシステムのドキュメントを参照してください。

B.3.4 クエリー関連の問題

B.3.4.1 文字列検索での大文字/小文字の区別

非バイナリ文字列の場合 (`CHAR`、`VARCHAR`、`TEXT`)、文字列検索では比較オペランドの照合順序が使用されます。バイナリ文字列 (`BINARY`、`VARBINARY`、`BLOB`) の場合、比較ではオペランド内のバイトの数値が使用されます。つまり、アルファベット文字の場合、比較では大文字と小文字が区別されます。

非バイナリ文字列とバイナリ文字列の比較は、バイナリ文字列の比較として扱われます。

単純な比較操作 (`>=`、`>`、`=`、`<`、`<=`、ソート、およびグループ化) は、各文字の「ソート値」に基づきます。同じソート値を持つ文字は同じ文字として扱われます。たとえば、特定の照合で `e` と `é` のソート値が同じである場合、それらは等しいと比較されます。

デフォルトの文字セットおよび照合順序は `utf8mb4` および `utf8mb4_0900_ai_ci` であるため、非バイナリ文字列比較ではデフォルトで大文字と小文字が区別されません。これは、`col_name LIKE 'a%'` を使用して検索した場合、`A` または `a` で始まるすべてのカラム値が取得されることを意味します。この検索で大文字と小文字を区別するには、オペランドのいずれかに大文字と小文字を区別する照合順序またはバイナリ照合順序があることを確認します。たとえば、両方とも `utf8mb4` 文字セットを持つカラムと文字列を比較する場合は、`COLLATE` 演算子を使用して、いずれかのオペランドに `utf8mb4_0900_as_cs` 照合順序または `utf8mb4_bin` 照合順序を設定できます:

```
col_name COLLATE utf8mb4_0900_as_cs LIKE 'a%'
col_name LIKE 'a%' COLLATE utf8mb4_0900_as_cs
col_name COLLATE utf8mb4_bin LIKE 'a%'
col_name LIKE 'a%' COLLATE utf8mb4_bin
```

カラムを常に大文字と小文字を区別して処理する場合は、大文字と小文字を区別する照合順序またはバイナリ照合順序で宣言します。 [セクション13.1.20「CREATE TABLE ステートメント」](#) を参照してください。

非バイナリ文字列の大/小文字を区別する比較で大/小文字を区別しないようにするには、`COLLATE` を使用して大/小文字を区別しない照合に名前を付けます。次の例の文字列では、通常、大/小文字が区別されますが、`COLLATE` は比較を大/小文字を区別しないように変更します:

```
mysql> SET NAMES 'utf8mb4';
mysql> SET @s1 = 'MySQL' COLLATE utf8mb4_bin,
        @s2 = 'mysql' COLLATE utf8mb4_bin;
mysql> SELECT @s1 = @s2;
+-----+
| @s1 = @s2 |
+-----+
|          0 |
+-----+
mysql> SELECT @s1 COLLATE utf8mb4_0900_ai_ci = @s2;
+-----+
| @s1 COLLATE utf8mb4_0900_ai_ci = @s2 |
+-----+
|                                  1 |
+-----+
```

バイナリ文字列では、比較で大/小文字が区別されます。文字列を大/小文字を区別しないものとして比較するには、文字列を非バイナリ文字列に変換し、`COLLATE` を使用して大/小文字を区別しない照合に名前を付けます:

```
mysql> SET @s = BINARY 'MySQL';
mysql> SELECT @s = 'mysql';
+-----+
| @s = 'mysql' |
+-----+
|          0 |
+-----+
mysql> SELECT CONVERT(@s USING utf8mb4) COLLATE utf8mb4_0900_ai_ci = 'mysql';
+-----+
| CONVERT(@s USING utf8mb4) COLLATE utf8mb4_0900_ai_ci = 'mysql' |
+-----+
|                                  1 |
+-----+
```

値を非バイナリ文字列とバイナリ文字列のどちらとして比較するかを決定するには、`COLLATION()` 関数を使用します。この例では、`VERSION()` が大文字と小文字を区別しない照合順序を持つ文字列を返すため、比較では大文字と小文字が区別されません:

```
mysql> SELECT COLLATION(VERSION());
+-----+
| COLLATION(VERSION()) |
+-----+
```



```
| utf8_general_ci |
+-----+
```

バイナリ文字列の場合、照合値は `binary` であるため、比較では大文字と小文字が区別されます。 `binary` が予想されるコンテキストの 1 つは、一般的なルールとしてバイナリ文字列を返す圧縮関数です: `string`:

```
mysql> SELECT COLLATION(COMPRESS('x'));
+-----+
| COLLATION(COMPRESS('x')) |
+-----+
| binary                    |
+-----+
```

文字列のソート値を確認する場合は、`WEIGHT_STRING()` が役に立つことがあります。 [セクション12.8「文字列関数および演算子」](#) を参照してください。

B.3.4.2 DATE カラムの使用に関する問題

`DATE` 値の形式は `'YYYY-MM-DD'` です。標準 SQL に従うと、ほかの形式は許可されません。 `UPDATE` の式および `SELECT` ステートメントの `WHERE` 句では、この形式を使用してください。例:

```
SELECT * FROM t1 WHERE date >= '2003-05-05';
```

便宜上、日付が数値コンテキストで使用されている場合、MySQL は自動的に日付を数値に変換します。逆の場合も同様です。また、MySQL は、更新時、および日付を `DATE`、`DATETIME`、または `TIMESTAMP` カラムと比較する `WHERE` 句で、「緩やかな」文字列形式を許可します。「緩やかな」形式とは、各部分の区切り文字として句読点文字を使用できることを意味します。たとえば、`'2004-08-15'` と `'2004#08#15'` は同等です。MySQL は、日付として解釈できる場合、区切り文字が含まれていない文字列 (`'20040815'` など) も変換できます。

`<`、`<=`、`=`、`>=`、`>`、または `BETWEEN` 演算子を使用して、`DATE`、`TIME`、`DATETIME`、または `TIMESTAMP` を定数文字列と比較する場合、通常、MySQL はより速く比較するために (および「緩やかな」文字列チェックのため) 文字列を内部長整数に変換します。ただし、この変換には次の例外があります。

- 2 つのカラムを比較する場合
- `DATE`、`TIME`、`DATETIME`、または `TIMESTAMP` カラムと式を比較する場合
- 上記で一覧表示した比較方法以外の比較方法を使用する場合 (`IN`、`STRCMP()` など)。

これらの例外の場合、比較はオブジェクトを文字列に変換して文字列比較を実行することによって行われます。

安全に処理を行うには、時間値と文字列を比較する場合、文字列が文字列として比較されると想定し、適切な文字列関数を使用します。

特殊な「ゼロ」日付 `'0000-00-00'` は、`'0000-00-00'` として格納および取得できます。`'0000-00-00'` 日付が Connector/ODBC を介して使用される場合、ODBC はそのような日付を処理できないため、`NULL` に自動的に変換されます。

MySQL が前述の変換を実行するため、次のステートメントは動作します (`idate` が `DATE` カラムであると想定しています)。

```
INSERT INTO t1 (idate) VALUES (19970505);
INSERT INTO t1 (idate) VALUES ('19970505');
INSERT INTO t1 (idate) VALUES ('97-05-05');
INSERT INTO t1 (idate) VALUES ('1997.05.05');
INSERT INTO t1 (idate) VALUES ('1997 05 05');
INSERT INTO t1 (idate) VALUES ('0000-00-00');

SELECT idate FROM t1 WHERE idate >= '1997-05-05';
SELECT idate FROM t1 WHERE idate >= 19970505;
SELECT MOD(idate,100) FROM t1 WHERE idate >= 19970505;
SELECT idate FROM t1 WHERE idate >= '19970505';
```

ただし、次のステートメントは動作しません。

```
SELECT idate FROM t1 WHERE STRCMP(idate,'20030505')=0;
```

`STRCMP()` は文字列関数であるため、`idate` を `'YYYY-MM-DD'` 形式の文字列に変換し、文字列比較を実行します。`'20030505'` は日付 `'2003-05-05'` に変換されずに、日付比較が実行されます。

`ALLOW_INVALID_DATES` SQL モードを有効にしている場合、MySQL は限定的なチェックのみが行われた日付を格納することを許可します。MySQL は、日が 1 から 31 までの範囲内にあり、月が 1 から 12 までの範囲内にあることのみを要求します。これにより、Web アプリケーションで年、月、および日を 3 つの別個のフィールドで取得して、ユーザーが入力したとおりに格納する (日付検証なしで) 場合に、MySQL が非常に便利になります。

MySQL は、日または月と日がゼロである日付の格納を許可します。これは、生年月日を `DATE` カラムに格納するが、その日付の一部のみがわかっている場合に便利です。日付にゼロの月または日の部分を許可しないようにするには、`NO_ZERO_IN_DATE` モードを有効にします。

MySQL では、「ダミーの日付」として `'0000-00-00'` の「ゼロ」の値を格納できます。これは、`NULL` 値を使用するよりも便利な場合があります。`DATE` カラムに格納される日付を妥当な値に変換できない場合、MySQL は `'0000-00-00'` を格納します。`'0000-00-00'` を禁止するには、`NO_ZERO_DATE` モードを有効にします。

MySQL がすべての日付をチェックして、有効な日付のみを受け入れるようにするには (`IGNORE` によってオーバーライドされないかぎり)、`sql_mode` システム変数に `"NO_ZERO_IN_DATE,NO_ZERO_DATE"` を設定します。

B.3.4.3 NULL 値に関する問題

`NULL` 値の概念については、`NULL` が空の文字列 `"` と同じであると考えがちな SQL の初心者が混乱することがよくあります。これらは同一ではありません。たとえば、次の 2 つのステートメントは完全に異なります。

```
mysql> INSERT INTO my_table (phone) VALUES (NULL);
mysql> INSERT INTO my_table (phone) VALUES ("");
```

両方のステートメントで `phone` カラムに値が挿入されていますが、最初のステートメントは `NULL` 値を挿入しており、2 番目のステートメントは空の文字列を挿入しています。最初のステートメントの意味は「電話番号がわからない」、2 番目のステートメントの意味は「この人は電話を持っていないため、電話番号がない」と見なすことができます。

`NULL` を処理する場合は、`IS NULL` 演算子と `IS NOT NULL` 演算子、および `IFNULL()` 関数を使用できます。

SQL では、`NULL` 値はほかの値 (`NULL` を含む) との比較で `true` になることはありません。`NULL` を含む式は、式に関連する演算子および関数のドキュメントに示されている場合を除き、常に `NULL` 値を生成します。次の例のすべてのカラムは `NULL` を返します。

```
mysql> SELECT NULL, 1+NULL, CONCAT('Invisible',NULL);
```

`NULL` であるカラム値を検索する場合、`expr = NULL` テストは使用できません。`expr = NULL` はどのような式の場合でも `true` にならないため、次のステートメントは行を返しません。

```
mysql> SELECT * FROM my_table WHERE phone = NULL;
```

`NULL` 値を検索するには、`IS NULL` テストを使用する必要があります。次のステートメントは、`NULL` の電話番号および空の電話番号を検索する方法を示しています。

```
mysql> SELECT * FROM my_table WHERE phone IS NULL;
mysql> SELECT * FROM my_table WHERE phone = "";
```

追加情報および例については、[セクション 3.3.4.6 「NULL 値の操作」](#) を参照してください。

`MyISAM`、`InnoDB`、または `MEMORY` ストレージエンジンを使用している場合は、`NULL` 値を持つことができるカラムにインデックスを追加できます。それ以外の場合は、インデックスが付けられるカラムを `NOT NULL` と宣言する必要があり、そのカラムには `NULL` を挿入できません。

`LOAD DATA` でデータを読み取ると、空または欠落しているカラムが `"` で更新されます。`NULL` 値をカラムにロードするには、データファイルで `\N` を使用します。状況によっては、リテラル文字 `NULL` も使用できます。[セクション 13.2.7 「LOAD DATA ステートメント」](#) を参照してください。

`DISTINCT`、`GROUP BY`、または `ORDER BY` が使用された場合、すべての `NULL` 値は等しいと見なされます。

`ORDER BY` を使用した場合、`NULL` 値は最初 (`DESC` を指定してソートを降順にした場合は最後) に表示されます。

`COUNT()`、`MIN()`、`SUM()` などの集計 (グループ) 関数では、`NULL` 値は無視されます。例外は個別のカラム値ではなく行数をカウントする `COUNT(*)` です。たとえば、次のステートメントは 2 つのカウントを生成します。最初のカウントはテーブル内の行数のカウントであり、2 番目のカウントは `age` カラムの `NULL` 以外の値の数のカウントです。

```
mysql> SELECT COUNT(*), COUNT(age) FROM person;
```

一部のデータ型では、MySQL は `NULL` 値に対して特殊な処理を行います。`NULL` を `TIMESTAMP` カラムに挿入すると、現在の日付と時間が挿入されます。`NULL` を `AUTO_INCREMENT` 属性を持つ整数カラムまたは浮動小数点カラムに挿入すると、シーケンスの次の数値が挿入されます。

B.3.4.4 カラムエイリアスに関する問題

エイリアスをクエリーの選択リストに使用すると、カラムを別の名前にすることができます。`GROUP BY`、`ORDER BY`、または `HAVING` 句でエイリアスを使用して、カラムを参照できます。

```
SELECT SQRT(a*b) AS root FROM tbl_name
GROUP BY root HAVING root > 0;
SELECT id, COUNT(*) AS cnt FROM tbl_name
GROUP BY id HAVING cnt > 0;
SELECT id AS 'Customer identity' FROM tbl_name;
```

標準 SQL では、`WHERE` 句でのカラムエイリアスへの参照は許可されません。`WHERE` 句が評価されるときに、カラム値がまだ判別されていない場合があるため、この制限が課されています。たとえば、次のクエリーは不正です。

```
SELECT id, COUNT(*) AS cnt FROM tbl_name
WHERE cnt > 0 GROUP BY id;
```

`WHERE` 句は `GROUP BY` 句に含められる行を判別しますが、行が選択されるまでわからないカラム値のエイリアスを参照して `GROUP BY` によってグループ化しています。

クエリーの選択リストで、引用したカラムエイリアスを指定するには、識別子または文字列引用文字を使用します。

```
SELECT 1 AS `one`, 2 AS `two`;
```

ステートメント内のどこに指定する場合でも、エイリアスへの引用した参照には、識別子引用符を使用する必要があります。そうしないと、参照は文字列リテラルとして扱われます。たとえば、次のステートメントはカラム `id` の値によってグループ化され、エイリアス ``a`` を使用して参照されます。

```
SELECT id AS `a`, COUNT(*) AS cnt FROM tbl_name
GROUP BY `a`;
```

このステートメントはリテラル文字列 ``a`` でグループ化され、次のようには機能しません:

```
SELECT id AS `a`, COUNT(*) AS cnt FROM tbl_name
GROUP BY `a`;
```

B.3.4.5 非トランザクションテーブルのロールバックの失敗

`ROLLBACK` を実行しようとしたときに次のメッセージを受け取った場合は、トランザクションで使用された 1 つ以上のテーブルがトランザクションをサポートしていないことを意味します。

```
Warning: Some non-transactional changed tables couldn't be rolled back
```

これらの非トランザクションテーブルは、`ROLLBACK` ステートメントの影響を受けません。

トランザクション内でトランザクションテーブルと非トランザクションテーブルを意図的に混在させていない場合、このメッセージの原因は、トランザクションテーブルと考えていたテーブルが実際にはそうではなかったことである可能性があります。これは、`mysqld` サーバーによってサポートされていない (または起動オプションで無効にされた) トランザクションストレージエンジンを使用してテーブルを作成しようとした場合に発生することがあります。`mysqld` がストレージエンジンをサポートしない場合は、非トランザクションである `MyISAM` テーブルとしてテーブルが作成されます。

テーブルのストレージエンジンを確認するには、次のいずれかのステートメントを使用します。

```
SHOW TABLE STATUS LIKE 'tbl_name';
SHOW CREATE TABLE tbl_name;
```

セクション13.7.7.38 「`SHOW TABLE STATUS` ステートメント」 およびセクション13.7.7.10 「`SHOW CREATE TABLE` ステートメント」を参照してください。

`mysqld` サーバーによってサポートされるストレージエンジンを確認するには、次のステートメントを使用します。

```
SHOW ENGINES;
```

詳細は、[セクション13.7.7.16「SHOW ENGINES ステートメント」](#)を参照してください。

B.3.4.6 関連するテーブルからの行の削除

`related_table` の `DELETE` ステートメントの合計長が `max_allowed_packet` システム変数のデフォルト値より大きい場合は、小さい部分に分割して、複数の `DELETE` ステートメントを実行する必要があります。 `related_column` にインデックスが付けられている場合は、ステートメントごとに 100 から 1,000 の `related_column` 値のみを指定することによって、`DELETE` が最速になる可能性があります。 `related_column` にインデックスが付けられていない場合、速度は `IN` 句の引数の数の影響を受けません。

B.3.4.7 一致する行がない場合の問題の解決

多数のテーブルを使用する複雑なクエリーで行が返されない場合は、次の手順を使用して問題を判別してください。

1. `EXPLAIN` を指定してクエリーをテストし、明らかな間違いが見つかるかどうかを確認します。 [セクション13.8.2「EXPLAIN ステートメント」](#)を参照してください。
2. `WHERE` 句で使用されているカラムのみを選択します。
3. 行が返されるまで、クエリーから一度に 1 つずつテーブルを削除します。テーブルが大きい場合、クエリーに `LIMIT 10` を使用するのはいくつかの方法です。
4. クエリーから最後に削除したテーブルに対して一致する行を持つカラムに `SELECT` を発行します。
5. `FLOAT` カラムまたは `DOUBLE` カラムと 10 進数の数値を比較している場合、等式 (=) 比較は使用できません。すべての浮動小数点値が正確な精度で格納されるとはかぎらないため、この問題はほとんどのコンピュータ言語で一般的です。 `FLOAT` を `DOUBLE` に変更すると解決することがあります。 [セクションB.3.4.8「浮動小数点値に関する問題」](#)を参照してください。
6. 問題がまだ特定されない場合は、`mysql test < query.sql` を使用して実行可能な、問題が再現される最小限のテストを作成します。テストファイルを作成するには、`mysqldump --quick db_name tbl_name_1 ... tbl_name_n > query.sql` を使用してテーブルをダンプします。エディタでファイルを開いて、一部の挿入行を削除し (問題の再現に必要な分量以上にある場合)、ファイルの最後に `SELECT` ステートメントを追加します。

次のコマンドを実行して、テストファイルで問題が再現されることを確認します。

```
shell> mysqladmin create test2
shell> mysql test2 < query.sql
```

テストファイルをバグレポートに添付します ([セクション1.6「質問またはバグをレポートする方法」](#)の手順を参照してください)。

B.3.4.8 浮動小数点値に関する問題

浮動小数点数は、近似値であり正確な値として格納されないため、混乱の原因となることがあります。SQL ステートメントで出力される浮動小数点値は、内部で表された値と同じではないことがあります。比較で浮動小数点値を正確な値として扱おうとすると、問題となることがあります。これらはまた、プラットフォームまたは実装の依存関係にも従います。 `FLOAT` データ型および `DOUBLE` データ型では、これらの問題が発生することがあります。 `DECIMAL` カラムの場合、MySQL は演算を 65 桁 (10 進数) の精度で実行するため、ほとんどの一般的な精度の問題が解決されます。

次の例では、`DOUBLE` を使用し、浮動小数点演算を使用して行われる計算がどのように浮動小数点エラーとなるかを示しています。

```
mysql> CREATE TABLE t1 (i INT, d1 DOUBLE, d2 DOUBLE);
mysql> INSERT INTO t1 VALUES (1, 101.40, 21.40), (1, -80.00, 0.00),
-> (2, 0.00, 0.00), (2, -13.20, 0.00), (2, 59.60, 46.40),
-> (2, 30.40, 30.40), (3, 37.00, 7.40), (3, -29.60, 0.00),
-> (4, 60.00, 15.40), (4, -10.60, 0.00), (4, -34.00, 0.00),
-> (5, 33.00, 0.00), (5, -25.80, 0.00), (5, 0.00, 7.20),
-> (6, 0.00, 0.00), (6, -51.40, 0.00);
```

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b
-> FROM t1 GROUP BY i HAVING a <= b;
```

```
+-----+-----+-----+
|i | a | b |
+-----+-----+
| 1 | 21.4 | 21.4 |
| 2 | 76.8 | 76.8 |
| 3 | 7.4 | 7.4 |
| 4 | 15.4 | 15.4 |
| 5 | 7.2 | 7.2 |
| 6 | -51.4 | 0 |
+-----+-----+-----+
```

正しい結果です。最初の5レコードは比較を満たしていないように見えますが(aとbの値は異なるように見えませんが)、コンピュータのアーキテクチャー、コンパイラのバージョン、最適化レベルなどの要因によって、小数点以下1桁などの数字が異なるためにこのような結果となっている可能性があります。たとえば、CPUが異なると、浮動小数点数の評価が異なることがあります。

カラム d1 および d2 が **DOUBLE** ではなく **DECIMAL** として定義されていた場合、**SELECT** クエリーの結果は1行のみ(上記の最後の行)となります。

浮動小数点数の比較を正しく行うには、最初に数値の差異に関して受け入れられる許容度を決定し、許容値に対して比較を行います。たとえば、1万分の1(0.0001)の精度内で同じであれば浮動小数点数が同じであると見なす場合は、許容値より大きい差異を見つけるように比較を記述してください。

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b FROM t1
-> GROUP BY i HAVING ABS(a - b) > 0.0001;
```

```
+-----+-----+-----+
|i | a | b |
+-----+-----+
| 6 | -51.4 | 0 |
+-----+-----+
1 row in set (0.00 sec)
```

逆に、数値が同じである行を取得する場合は、テストで許容値内での差異を判断するようにします。

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b FROM t1
-> GROUP BY i HAVING ABS(a - b) <= 0.0001;
```

```
+-----+-----+-----+
|i | a | b |
+-----+-----+
| 1 | 21.4 | 21.4 |
| 2 | 76.8 | 76.8 |
| 3 | 7.4 | 7.4 |
| 4 | 15.4 | 15.4 |
| 5 | 7.2 | 7.2 |
+-----+-----+
5 rows in set (0.03 sec)
```

浮動小数点値はプラットフォームまたは実装の依存関係の影響を受けます。次のステートメントを実行するとします。

```
CREATE TABLE t1(c1 FLOAT(53,0), c2 FLOAT(53,0));
INSERT INTO t1 VALUES('1e+52','1e+52');
SELECT * FROM t1;
```

一部のプラットフォームでは、**SELECT** ステートメントは **inf** および **-inf** を返します。ほかのプラットフォームでは、**0** および **-0** が返されます。

前述の問題の影響は、ソースで **mysqldump** を使用してテーブルの内容をダンプし、ダンプファイルをレプリカにリロードしてレプリカを作成しようとする、浮動小数点カラムを含むテーブルが2つのホスト間で異なる可能性があることです。

B.3.5 オプティマイザ関連の問題

MySQL はコストベースのオプティマイザを使用して、クエリーを実行する最適な方法を判別しています。多くの場合、MySQL は実行可能な最適なクエリー計画を計算できますが、データに関する情報を十分に取得できず、データについて「学習による」推測を行う必要がある場合があります。

MySQL で「適切」に処理されなかった場合に、MySQL に指示を送るために使用できるツールを次に示します。

- **EXPLAIN** ステートメントを使用して、MySQL がクエリーを処理する方法に関する情報を取得します。これを使用するには、キーワード **EXPLAIN** を **SELECT** ステートメントの前に追加します。

```
mysql> EXPLAIN SELECT * FROM t1, t2 WHERE t1.i = t2.i;
```

EXPLAIN については、[セクション13.8.2「EXPLAIN ステートメント」](#)で詳しく説明しています。

- **ANALYZE TABLE tbl_name** を使用して、スキャンされるテーブルのキー分布を更新します。[セクション13.7.3.1「ANALYZE TABLE ステートメント」](#)を参照してください。
- スキャンされるテーブルに **FORCE INDEX** を使用して、テーブルスキャンは該当するインデックスを使用した場合と比較して著しく負荷が高いことを MySQL に通知します。

```
SELECT * FROM t1, t2 FORCE INDEX (index_for_column)  
WHERE t1.col_name=t2.col_name;
```

USE INDEX および **IGNORE INDEX** も役に立つことがあります。[セクション8.9.4「インデックスヒント」](#)を参照してください。

- グローバルおよびテーブルレベルの **STRAIGHT_JOIN**。[セクション13.2.10「SELECT ステートメント」](#)を参照してください。
- グローバルなシステム変数またはスレッド固有のシステム変数をチューニングできます。たとえば、キースキャンでキー検索が 1,000 回を超えて行われることはない想定するように最適化に通知するには、**--max-seeks-for-key=1000** オプションを指定して **mysqld** を起動するか、**SET max_seeks_for_key=1000** を使用します。[セクション5.1.8「サーバースystem変数」](#)を参照してください。

B.3.6 テーブル定義関連の問題

B.3.6.1 ALTER TABLE での問題

ALTER TABLE を使用して文字セットまたは文字カラムの照合順序を変更するときに、重複キーエラーを受け取った場合、原因は新しいカラムの照合順序が同じ値に対して 2 つのキーをマップしたか、テーブルが破損していることです。後者の場合は、そのテーブルに対して **REPAIR TABLE** を実行してください。**REPAIR TABLE** は、**MyISAM**、**ARCHIVE** および **CSV** テーブルに対して機能します。

トランザクションテーブルに対して **ALTER TABLE** を使用したとき、または Windows を使用しているときに、**LOCK TABLE** が発行されていた場合、**ALTER TABLE** はテーブルをロック解除します。これが行われるのは、**InnoDB** およびこれらのオペレーティングシステムは使用されているテーブルをドロップできないためです。

B.3.6.2 TEMPORARY テーブルに関する問題

CREATE TEMPORARY TABLE で作成される一時テーブルには、次の制限があります：

- **TEMPORARY** テーブルは、**InnoDB**、**MEMORY**、**MyISAM** および **MERGE** ストレージエンジンでのみサポートされます。
- **NDB Cluster** では一時テーブルはサポートされていません。
- **SHOW TABLES** ステートメントでは **TEMPORARY** テーブルは一覧表示されません。
- **TEMPORARY** テーブルの名前を変更する場合、**RENAME TABLE** は機能しません。かわりに **ALTER TABLE** を使用します：

```
ALTER TABLE old_name RENAME new_name;
```

- 同じクエリーで **TEMPORARY** テーブルを複数回参照することはできません。たとえば、次のステートメントは動作しません。

```
SELECT * FROM temp_table JOIN temp_table AS t2;
```

このステートメントによって次のエラーが生成されます：


```
ERROR 1137: Can't reopen table: 'temp_table'
```

クエリーで **TEMPORARY** テーブルではなく共通テーブル式 (CTE) の使用が許可されている場合は、この問題を回避できます。たとえば、これは「**テーブルを再オープンできません**」エラーで失敗します:

```
CREATE TEMPORARY TABLE t SELECT 1 AS col_a, 2 AS col_b;
SELECT * FROM t AS t1 JOIN t AS t2;
```

エラーを回避するには、**TEMPORARY** テーブルではなく CTE を定義する **WITH** 句を使用します:

```
WITH cte AS (SELECT 1 AS col_a, 2 AS col_b)
SELECT * FROM cte AS t1 JOIN cte AS t2;
```

- ・ ストアドファンクションで一時テーブルを異なるエイリアスで複数回参照すると、関数内の異なるステートメントで参照が発生しても、「**テーブルを再オープンできません**」エラーが発生します。これは、ストアドファンクションの外部で作成され、複数の呼び出し元および呼び出し先関数にわたって参照される一時テーブルで発生することがあります。
- ・ **TEMPORARY** が既存の **TEMPORARY** 以外のテーブルと同じ名前で作成された場合、**TEMPORARY** 以外のテーブルは、別の記憶域エンジンを使用しているても、**TEMPORARY** テーブルが削除されるまで非表示になります。
- ・ レプリケーションで一時テーブルを使用する場合の既知の問題があります。詳しくは[セクション 17.5.1.31 「レプリケーションと一時テーブル」](#)をご覧ください。

B.3.7 MySQL の既知の問題

このセクションでは、最新バージョンの MySQL の既知の問題を一覧表示します。

プラットフォーム固有の問題の詳細は、[セクション 2.1 「一般的なインストールガイド」](#) および [セクション 5.9 「MySQL のデバッグ」](#) のインストールおよびデバッグの手順を参照してください。

次の問題は既知の問題です。

- ・ **IN** のサブクエリーの最適化は、= ほど効果はありません。
- ・ **lower_case_table_names=2** (データベース名およびテーブル名に大文字/小文字のどちらが使用されたかを MySQL が認識するようになります) を使用しているても、MySQL が関数 **DATABASE()** のデータベース名、またはさまざまなログ内 (大文字/小文字が区別されないシステムの) で使用された表記を識別できません。
- ・ **FOREIGN KEY** 制約の削除はレプリケーションでは機能しません。これは、制約がレプリカ上に別の名前を持つ可能性があるためです。
- ・ **REPLACE** (および **REPLACE** オプションを指定した **LOAD DATA**) で **ON DELETE CASCADE** がトリガーされません。
- ・ **DISTINCT** リストに指定されたすべてのカラムのみを使用しない場合、**GROUP_CONCAT()** 内で **ORDER BY** を指定した **DISTINCT** が動作しません。
- ・ 数値は符号付き整数コンテキストで評価されるため、小数カラムまたは文字列カラムに大きい整数値 (2^{63} から $2^{64}-1$) を挿入すると、負の値として挿入されます。
- ・ ステートメントベースのバイナリロギングでは、ソースサーバーは実行されたクエリーをバイナリログに書き込みます。これは、ほとんどの場合に理想的に動作する非常に高速かつコンパクトで効率的なロギング方法です。ただし、データ変更が非決定的に行われるようにクエリーが設計されている場合は、ソースとレプリカのデータが異なる可能性があります (通常、レプリケーションの外部であっても推奨されません)。

例:

- ・ ゼロ値または **NULL** 値を **AUTO_INCREMENT** カラムに挿入する **CREATE TABLE ... SELECT** ステートメントまたは **INSERT ... SELECT** ステートメント。
- ・ **ON DELETE CASCADE** プロパティが指定された外部キーを持つテーブルから行を削除する場合の **DELETE**。
- ・ 挿入されるデータに重複キー値がある場合の **REPLACE ... SELECT**、**INSERT IGNORE ... SELECT**。

これは、前述したクエリーに決定性順序を保証する `ORDER BY` 句がない場合にのみ発生することがあります。

たとえば、`ORDER BY` を使用しない `INSERT ... SELECT` の場合、`SELECT` は、ソースおよびレプリカでの最適化による選択に応じて、異なる順序で行を返すことがあります (これにより、行のランクが異なるため、`AUTO_INCREMENT` カラムで異なる番号が取得されます)。

クエリーは、次の場合にのみ、ソースとレプリカで異なる方法で最適化されます：

- テーブルは、レプリカとは異なるストレージエンジンを使用してソースに格納されます。(ソースとレプリカで異なるストレージエンジンを使用できます。たとえば、ソースでは `InnoDB` を使用できますが、レプリカに使用可能なディスク領域が少ない場合はレプリカで `MyISAM` を使用できます。)
- MySQL バッファサイズ (`key_buffer_size` など) は、ソースとレプリカで異なります。
- ソースとレプリカは異なる MySQL バージョンを実行し、最適化コードはこれらのバージョン間で異なります。

この問題は、`mysqlbinlog|mysql` を使用したデータベースのリストアに影響することもあります。

この問題を回避するもっとも簡単な方法は、行が常に同じ順序で格納または変更されるように、`ORDER BY` 句を前述の非決定性クエリーに追加することです。行ベースのロギング形式または混合したロギング形式を使用することでも、この問題が回避されます。

- スタートアップオプションにファイル名を指定しない場合、ログファイル名はサーバーのホスト名に基づいています。ホスト名を別の名前に変更した場合に同じログファイル名のままにするには、`--log-bin=old_host_name-bin` などのオプションを明示的に使用する必要があります。セクション 5.1.7 「サーバーコマンドオプション」を参照してください。または、ホスト名の変更が反映されるように、古いファイルを名前変更します。バイナリログの場合は、バイナリログのインデックスファイルを編集して、そこにあるバイナリログファイル名も修正する必要があります。(レプリカ上のリレーログにも同じことが当てはまります。)
- `mysqlbinlog` では、`LOAD DATA` ステートメントの後に残っている一時ファイルは削除されません。セクション 4.6.8 「`mysqlbinlog` — バイナリログファイル処理のためのユーティリティ
- `RENAME` が `TEMPORARY` テーブル、または `MERGE` テーブルで使用されているテーブルで動作しません。
- `SET CHARACTER SET` を使用したときに、データベース、テーブル、およびカラムの名前に変換された文字を使用できません。
- MySQL 8.0.17 より前は、`LIKE ... ESCAPE` で `ESCAPE` とともに `_` または `%` を使用することはできません。
- サーバーは、データ値を比較するときに最初の `max_sort_length` バイトのみを使用します。つまり、値が最初の `max_sort_length` バイトの後にのみ異なる場合、`GROUP BY`、`ORDER BY` または `DISTINCT` で値を確実に使用することはできません。これを回避するには、変数値を増やします。`max_sort_length` のデフォルト値は 1024 であり、サーバーの起動時または実行時に変更できます。
- 数値計算は `BIGINT` または `DOUBLE` (通常、どちらも長さは 64 ビットです) で行われます。返される精度は関数によって異なります。一般的なルールとしては、ビット関数は `BIGINT` の精度、`IF()` と `ELT()` は `BIGINT` または `DOUBLE` の精度、および残りは `DOUBLE` の精度で実行されます。符号なしの long long 値がビットフィールド以外で 63 ビット (9223372036854775807) を超える値に解決される場合は、使用しないようにしてください。
- 1 つのテーブルには、最大 255 個の `ENUM` カラムおよび `SET` カラムを作成できます。
- 現在、`MIN()`、`MAX()`、およびその他の集約関数では、MySQL はセット内の文字列の相対位置ではなく文字列値で `ENUM` カラムおよび `SET` カラムを比較します。
- `UPDATE` ステートメントでは、カラムは左から右に更新されます。更新されたカラムを参照している場合は、元の値ではなく更新された値が取得されます。たとえば、次のステートメントでは `KEY` に 1 ではなく 2 がインクリメントされます。

```
mysql> UPDATE tbl_name SET KEY=KEY+1,KEY=KEY+1;
```
- 同じクエリーで複数の一時テーブルを参照することはできますが、特定の一時テーブルを複数回参照することはできません。たとえば、次のステートメントは動作しません。

```
mysql> SELECT * FROM temp_table, temp_table AS t2;  
ERROR 1137: Can't reopen table: 'temp_table'
```

- 結合で「隠し」カラムを使用している場合は、オプティマイザでの **DISTINCT** の処理が異なることがあります。結合では、隠しカラムは結果の一部としてカウントされますが (表示されていない場合)、通常のクエリーでは、隠しカラムは **DISTINCT** 比較で考慮されません。

次に例を示します。

```
SELECT DISTINCT mp3id FROM band_downloads  
WHERE userid = 9 ORDER BY id DESC;
```

および

```
SELECT DISTINCT band_downloads.mp3id  
FROM band_downloads,band_mp3  
WHERE band_downloads.userid = 9  
AND band_mp3.id = band_downloads.mp3id  
ORDER BY band_downloads.id DESC;
```

2 番目のケースでは、結果セットに同一の行が 2 つ表示される場合があります (非表示の **id** カラムの値が異なる可能性があるため)。

これは、結果に **ORDER BY** のカラムがないクエリーでのみ発生します。

- 空のセットを返すクエリーに関する **PROCEDURE** を実行すると、**PROCEDURE** でカラムが変換されないことがあります。
- **MERGE** タイプのテーブルの作成で、基礎テーブルが互換性のあるタイプであるかどうかチェックされません。
- **ALTER TABLE** を使用して、**MERGE** テーブルで使用されるテーブルに **UNIQUE** インデックスを追加し、次に **MERGE** テーブルに通常のインデックスを追加したときに、テーブルに古い **UNIQUE** ではないキーがあった場合、それらのテーブルのキー順序は異なります。これは、重複キーをできるだけ早く検出できるように、**ALTER TABLE** が通常のインデックスより **UNIQUE** インデックスを優先するためです。

付録 C インデックス

目次

一般的な索引	4617
C 関数の索引	4873
コマンドの索引	4876
関数の索引	4921
INFORMATION_SCHEMA の索引	4966
結合型の索引	4981
演算子の索引	4983
オプションの索引	4992
権限の索引	5089
SQL モードの索引	5105
ステートメント/構文の索引	5109
ステータス変数の索引	5204
システム変数の索引	5237
トランザクション分離レベルの索引	5331

一般的な索引

シンボル

!

- 非推奨となった機能, 36
- ! (論理 NOT), 1866
- != (等しくない), 1862
- ", 1636
- " (二重引用符), 1628, 2070
- %, 1874
- % (modulo), 1879
- % (ワイルドカード文字), 1628
- & & (論理 AND), 1867
- & (ビット単位 AND), 1983
- &&
 - 非推奨となった機能, 36
- 'on clause'のカラム ... が不明です, 2338, 2339
- () (括弧), 1860
- (Ctrl+Z) \Z, 1628, 2313
- * (multiplication), 1874
- + (addition), 1873
- (subtraction), 1873
- (単項マイナス), 1874
- bootstrap
 - 削除された機能, 41
- compress
 - 非推奨となった機能, 37
- des-key-file
 - 削除された機能, 40
- disable オプションプリフィクス, 308
- enable オプションプリフィクス, 308
- fix-db-names
 - 削除された機能, 41
- fix-table-names
 - 削除された機能, 41
- ignore-db-dir

- 削除された機能, 39
- log-warnings
 - 削除された機能, 39
- loose オプションプリフィクス, 308
- master-info-file
 - 非推奨となった機能, 37
- maximum オプションプリフィクス, 308
- no-dd-upgrade
 - 非推奨となった機能, 37
- partition
 - 削除された機能, 41
- password オプション, 1034
- secure-auth
 - 削除された機能, 39
- skip オプションプリフィクス, 308
- skip-partition
 - 削除された機能, 41
- ssl
 - 削除された機能, 40
- ssl-verify-server-cert
 - 削除された機能, 40
- temp-pool
 - 削除された機能, 40
- >, 2051
- >>, 2053
- ? オプション (NDB Cluster プログラム), 3836
- c オプション (NDB Cluster プログラム), 3837
- c オプション (ndb_mgmd) (OBSOLETE), 3736
- d オプション
 - ndb_index_stat, 3777
 - ndb_mgmd, 3737
- e オプション
 - ndb_mgm, 3742
- f オプション
 - ndb_mgmd, 3736
- I オプション
 - ndbinfo_select_all, 3732
- n オプション
 - ndbd, 3729
 - ndbmt, 3729
- p オプション, 1034
- P オプション
 - ndb_mgmd, 3740
- V オプション (NDB Cluster プログラム), 3838
- v オプション
 - ndb_mgmd, 3740
- .ARM ファイル, 5335
- .ARZ ファイル, 5335
- .cfg ファイル, 5335
- .frm ファイル, 5335
- .ibd ファイル, 2243, 5335
- .ibz ファイル, 5335
- .MRG ファイル, 5335
- .my.cnf オプションファイル, 302, 303, 321, 1016, 1035, 1125
- .MYD ファイル, 2243
- .MYD ファイル, 5335
- .MYI ファイル, 2243
- .MYI ファイル, 5336
- .mylogin.cnf オプションファイル, 302, 514
- .mysql_history ファイル, 386, 1035

.NET, 5336
.OPT ファイル, 5336
.par ファイル, 5336
.pid (プロセス ID) ファイル, 1427
.sdi ファイル, 2297
/ (division), 1874
/etc/passwd, 1037, 2332
1 つのデータベースオプション
 mysql, 374
10 進演算, 2142
16 進数リテラル, 1632
 ビット操作, 1633
16 進数リテラルの概要, 1632
2 フェーズコミット, 840, 840, 5336
3306 port, 208, 659
33060 ポート, 208
5.1 より前のデータベース名変換
 削除された機能, 41
:= (assignment), 1673
:= (代入演算子), 1868
< (より小さい), 1862
<< (左シフト), 1984
<< (左シフト), 291
<= (以下), 1862
<=> (等しい), 1862
<> (等しくない), 1862
= (assignment), 1673
= (equal), 1861
= (代入演算子), 1868
> (より大きい), 1863
>> (右シフト), 1984
>= (以上), 1862
[api] (NDB Cluster), 3550
[computer] (NDB Cluster), 3551
[mgm] (NDB Cluster), 3549
[mysqld] (NDB Cluster), 3550
[ndbd default] (NDB Cluster), 3543
[ndbd] (NDB Cluster), 3543
[ndb_mgmd] (NDB Cluster), 3549
[shm] (NDB Cluster), 3551
[tcp] (NDB Cluster), 3551
\ (単一引用符), 1628
\. (mysql クライアントコマンド), 287, 389
\0 (ASCII NUL), 1628, 2312
\b (backspace), 1628, 2070, 2312
\f (formfeed), 2070
\n (linefeed), 1628, 2070, 2312
\n (newline), 1628, 2070, 2312
\N (NULL), 2313
\r (キャリッジリターン), 1628, 2070, 2312
\t (tab), 1628, 2070, 2312
\u (Unicode 文字), 2070
\Z (Ctrl+Z) ASCII 26, 1628, 2313
\\ (エスケープ), 1628, 2070
^(ビット単位 XOR), 1984
_ (ファイルドカード文字), 1629
_ai 照合順序接尾辞, 1692
_as 照合接尾辞, 1692
_bin 照合接尾辞, 1692, 1712
_ci 照合接尾辞, 1692

- _cs 照合接尾辞, 1692
- _ks 照合接尾辞, 1692
- _rowid
 - SELECT ステートメント, 2199, 2226, 2226
- ` , 1636
- | (ビット単位 OR), 1983
- || (論理 OR), 1867
- ||
 - 非推奨となった機能, 36
- ~ (反転ビット), 1985
- アーリーアダプタ, 5353
- アイデンティティシステム変数, 711
- アカウントカテゴリ, 1087
- アカウント管理, 1043
- アカウントのロック, 1066, 1122
 - ALTER USER, 2489
 - CREATE USER ステートメント, 2500
 - Locked_connects ステータス変数, 846
- アカウント名, 1070
- 悪意のある SQL ステートメント
 - NDB Cluster, 3987
- アクセス権限, 1043
- アクセス制御, 1043, 1073
- アクセスは拒否されましたエラー, 4588
- アセンブリ, 5353
- 新しい, 5354
- 新しいシステム変数, 743
- 圧縮, 2761, 5353
 - KEY_BLOCK_SIZE, 2766
 - 圧縮済みページサイズ, 2766
 - アプリケーションとスキーマ設計, 2765
 - アルゴリズム, 2767
 - 概要, 2761
 - 構成の特性, 2766
 - 実装, 2767
 - 情報スキーマ, 2895
 - 調整, 2763
 - データとインデックス, 2768
 - データの特性, 2764
 - バッファープールの考慮事項, 2769
 - 変更ログ, 2768
 - ログファイル形式, 2770
 - ワークロードの特性, 2766
- 圧縮失敗, 5354
- 圧縮ステータス変数, 837
- 圧縮テーブル, 509, 2993, 5353
- 圧縮バックアップ, 5353
- 圧縮行形式, 2780
- 圧縮行フォーマット, 5354
- アップグレード, 233
 - MySQL SLES リポジトリ, 257, 257
 - MySQL Yum リポジトリを使用して, 255
 - MySQL の Docker インストール, 259
 - NDB Cluster, 3516, 3871
 - 新機能, 8
 - 別のアーキテクチャー, 261
- アップグレードオプション
 - mysqld, 667
 - MySQLInstallerConsole, 126
- アップグレードおよびダウングレード (NDB Cluster)

- バージョン間の互換性, 3516
- アトミック DDL, 2150
 - 新機能, 8
- アトミック DDL, 5353
- アトミック命令, 5353
- アプリケーションプログラミングインタフェース (API), 5353
- アライアンスキーマネージャ
 - keyring_okv キーリングプラグイン, 1240
- 暗号化, 1036, 1129
- 暗号化関数, 1985
- 暗号化された接続の確立, 1130
- 暗号化接続, 1129
 - コマンドオプション, 314
 - 必須として, 1135
- 安全でないステートメント (レプリケーション), 3156
 - 定義済み, 3155
- 安全なステートメント (レプリケーション)
 - defined, 3155
- 暗黙的 GROUP BY ソート, 1469
- 暗黙のカラム変更, 2257
- 暗黙の行ロック, 5354
- 暗黙のデフォルト値, 1824
- アンロード
 - テーブル, 273
- 以下 (<=), 1862
- 以上 (>=), 1862
- 移植性, 1432
 - 型, 1830
- 以前の起動後の CMake の実行, 188, 218
- 一意キー
 - およびパーティショニングキー, 4088
 - 制約, 71
- 一意制約, 5356
- 一意のインデックス, 5356
- 一意のキー, 5356
- 一時テーブル, 5356
 - およびレプリケーション, 3228
 - 内部, 1518
 - 問題, 4612
- 一時テーブルスペース, 5356
- 一時ファイル, 4603
- 一括ロード
 - InnoDB テーブルの, 1528
 - MyISAM テーブルの, 1536
- 一貫性読み取り, 2709, 5357
- 一致する行がない, 4610
- 一般クエリログ, 921, 5356
- 一般情報, 1
- 一般テーブルスペース, 5357
- イベント, 4104
 - 削除, 2282
 - 作成, 2189
 - ステータス変数, 4110
 - 変更, 2160
 - メタデータ, 4107
- イベントグループ, 3149
- イベントスケジューラ, 4095, 4104
 - SQL ステートメント, 4107
 - イベントの削除, 2282
 - イベントの作成, 2189

- イベントの変更, 2160
- イベントメタデータ, 4107
- および MySQL 権限, 4108
- および mysqldadmin デバッグ, 4108
- および SHOW PROCESSLIST, 4105
- およびレプリケーション, 3221, 3221
- 開始および停止, 4105
- 概念, 4104
- 時間表現, 4107
- ステータス情報の取得, 4108
- スレッドの状態, 1625
- 有効および無効, 4105
- イベントタイプ (NDB Cluster), 3858, 3861
- イベントテーブル
 - データディクショナリテーブル, 896
- イベントの重大度レベル (NDB Cluster), 3860
- イベントログ (NDB Cluster), 3858, 3859, 3860
- イベントログ形式 (NDB Cluster), 3861
- 意味のある JSON 値, 1814
- 色オプション
 - ndb_top, 3827
- 印刷オプション
 - ndb_restore, 3803
- インスタンス, 5354
- インストゥルメンテーション, 5354
- インストール, 123
 - Linux RPM パッケージ, 157
 - macOS DMG パッケージ, 139
 - Perl, 262
 - Perl を Windows に, 263
 - Solaris PKG パッケージ, 180
 - 概要, 82
 - ソース配布, 183
 - バイナリ配布, 98
- インストールオプション
 - mysqld, 652
 - MySQLInstallerConsole, 125
 - ndbd, 3728
 - ndbmt, 3728
 - ndb_mgmd, 3737
- インストール後
 - セットアップとテスト, 220
 - 複数サーバー, 1009
- インストールの概要, 183
- インストールのレイアウト, 97, 97
- インデックス, 2194, 5355
 - BLOB カラム, 1500, 2195
 - NULL 値, 2223
 - TEXT カラム, 1500, 2195
 - TIMESTAMP ルックアップ, 1512
 - および BLOB カラム, 2222
 - および IS NULL, 1505
 - および LIKE, 1505
 - および ndb_restore, 3805
 - および TEXT カラム, 2222
- カラム, 1500
- カラム接頭辞, 1500
- キーキャッシュへの割当て, 2598
- 降順, 1511
- 再作成, 259

- 削除, 2175, 2282
- 名前, 1635
- の使用, 1498
- 左端のプリフィクス, 1498, 1503
- ブロックサイズ, 715
- マルチカラム, 1502
- マルチパート, 2194
- インデックス拡張子, 1506
- インデックスキャッシュ, 5355
- インデックス結合タイプ
 - オブティマイザ, 1545
- インデックス条件プッシュダウン, 5355
- インデックス接頭辞
 - partitioning, 4085
- インデックス付き一時テーブル
 - 準結合方式, 1482
- インデックステーブル
 - データディクショナリテーブル, 896
- インデックス統計
 - NDB, 3646
- インデックスの左端のプリフィクス, 1498, 1503
- インデックスヒント, 1579, 2327, 5355
- インデックスプリフィクス, 5355
- インデックスレコードロック
 - InnoDB, 2717
- インテンションロック, 2701, 5354
- インテンションロックの挿入, 5355
- イントロデューサ
 - 16 進数リテラル, 1632
 - バイナリ文字セット, 1740
 - ビット値リテラル, 1634
 - 文字セット, 1698
 - 文字列リテラル, 1628, 1696
- インポート
 - データ, 389, 439
- インメモリーデータベース, 5356
- 引用
 - カラムエイリアス, 4609
- 引用符
 - 文字列内, 1629
- 引用符を使用
 - カラムエイリアス, 1636
- ウィンドウ
 - ウィンドウ関数, 2117
- ウィンドウ関数, 289, 2110
 - EXPLAIN, 1478
 - spatial, 2040
 - 構文, 2116
 - 最適化, 1477
 - 新機能, 25
 - 制限事項, 2124
 - 名前付きウィンドウ, 2123
- ウォームアップ, 5357
- ウォームバックアップ, 5357
- 埋込み, 5357
- 埋込みサーバーライブラリ
 - 削除された機能, 42
- 永続的統計, 5358
- エイリアス
 - GROUP BY 句, 2106

- 式, 2106
- 式に関する, 2327
- 大/小文字の区別, 1639
- テーブル, 2327
- 名前, 1635
- エクステンツ, 5358
- エスケープ (\), 1628, 2070
- エスケープシーケンス
 - オプションファイル, 305
 - 文字列, 1627
- エビクション, 5358
- エボック, 919, 1771
- エラー
 - アクセスは拒否されました, 4588
 - 一般的, 4587
 - 既知, 4613
 - サブクエリー内, 2356
 - 情報のソース, 4585
 - 接続が失われました, 4591
 - テーブルのチェック, 1424
 - リスト, 4588
 - レポート, 62, 62
- エラーコード, 549
 - 削除された機能, 42
- エラー番号, 549
- エラーメッセージ
 - 言語, 1740, 1740
 - テーブルを再オープンできません, 4612
 - 表示, 549
 - ファイルが見つかりません, 4598
 - ローカルデータのロードは無効です。これはクライアント側とサーバー側の両方で有効にする必要があります, 1041
- エラーログ (NDB Cluster), 3730
- エラーログ, 5358
- 円記号 (日本語), 4563
- 演算
 - 算術, 1873
- 演算子, 1834
 - bit, 1975
 - string, 1901
 - キャスト, 1873, 1952
 - 算術, 1975
 - 代入, 1673
 - 文字列の比較, 1914
 - 優先順位, 1859
 - 論理, 1866
 - 割り当て, 1868
- エンジンオプション
 - mysqlslap, 479
- エンジン条件プッシュダウン, 1446
- エンタープライズ拡張
 - MySQL Enterprise Audit, 1279
 - MySQL Enterprise Data Masking and De-Identification, 1371
 - MySQL Enterprise Encryption, 1388
 - MySQL Enterprise Firewall, 1348
 - MySQL Enterprise Security, 1162, 1171, 1176
 - MySQL Enterprise Thread Pool, 951
- エンタープライズコンポーネント
 - MySQL Enterprise Audit, 4531
 - MySQL Enterprise Backup, 4530
 - MySQL Enterprise Data Masking and De-Identification, 4532

- MySQL Enterprise Encryption, 4531
- MySQL Enterprise Firewall, 4532
- MySQL Enterprise Monitor, 4529
- MySQL Enterprise Security, 4531
- MySQL Enterprise Thread Pool, 4532
- 大きいテーブル
 - NDB Cluster, 2233
- 多くのサーバーの起動, 1009
- オーダーオプション
 - ndb_select_all, 3815
- オーバーフローの処理, 1768
- オーバーフローページ, 5359
- オープンソース
 - 定義, 4
- オープンテーブル, 1516
- オープンの数, 396
- 大文字/小文字の区別
 - 検索での, 4605
 - 文字列比較, 1914
- 大文字と小文字の区別
 - レプリケーションフィルタリングオプション, 3169
- 遅いクエリー, 396
- 同じ値が優先 (競合解決), 4019
- オブジェクト
 - stored, 4095
- オプション, 5358
 - CMake, 191
 - myisamchk, 495
 - MySQL で提供されている, 265
 - コマンド行
 - mysql, 363
 - mysqladmin, 396
 - ブール値, 308
- オプションの例
 - mysqld_multi, 343
- オプションファイル, 302, 1125, 5358
 - .my.cnf, 302, 303, 321, 1016, 1035, 1125
 - .mylogin.cnf, 302, 514
 - C:\my.cnf, 1016
 - my.cnf, 3212
 - mysqld-auto.cnf, 300, 302, 754, 806, 809, 826, 830, 1341, 2542, 2608, 4359
 - エスケープシーケンス, 305
- オプションプリフィクス
 - disable, 308
 - enable, 308
 - loose, 308
 - maximum, 308
 - skip, 308
- オプティマイザ, 5358
 - およびレプリケーション, 3225
 - 切り替え可能な最適化, 1556
 - クエリー計画評価, 1555
 - コストモデル, 1581
 - 新機能, 22
 - 制御, 1555
- オプティマイザ統計, 2753
 - InnoDB テーブル, 2747
- オプティマイザヒント, 1565
- オプティミスティック, 5358
- オフページカラム, 5358

- オペレーティングシステム
 - サポート対象, 83, 84
 - ファイルサイズの制限, 1521
- オンライン, 5359
- オンライン DDL, 5359
- オンライン DDL, 2786, 2787
 - 制限, 2804
 - 並列性, 2799
- オンラインアップグレードおよびダウングレード (NDB Cluster), 3871
 - ノード更新の順序, 3872
- カーソル, 2450, 5360
- カーディナリティー, 1479, 5360
- 開始
 - コメント, 71
- 解析可能オプション
 - ndb_show_tables, 3822
- 回転のコミットを待機しています
 - スレッドの状態, 1623
- 開発ソースツリー, 189
- 外部キー, 70, 290, 2176, 5361
 - 削除, 2176, 2252
 - 制約, 71, 72
 - メタデータのロック, 1599
- 外部キー制約, 2248
 - とオンライン DDL, 2804
- 外部結合
 - 最適化, 1456
- 外部ロック, 650, 774, 1423, 1600, 1620
- 概要, 1
- カウンタ, 5359
- カウント
 - テーブルの行, 281
- カウントオプション
 - innochecksum, 487
 - myisam_ftdump, 491
 - mysqladmin, 399
 - mysqlshow, 468
- 書き込み結合, 5361
- 鍵リングサービス関数
 - my_key_fetch(), 1006
 - my_key_generate(), 1006
 - my_key_remove(), 1007
 - my_key_store(), 1007
- 角括弧, 1762
- 拡張オプション
 - mysqlcheck, 410
- 拡張機能
 - 標準 SQL への, 66
- 加算 (+), 1873
- 仮想インデックス, 5360
- 仮想生成カラム, 5360
- 型
 - 移植性, 1830
 - カラム, 1761, 1830
 - データ, 1761
 - テーブルの, 2983
- 型変換, 1856, 1861
- カタログテーブル
 - データディクショナリテーブル, 895
- 括弧 (と), 1860

カッコで囲まれたクエリー式, 2342
稼働時間, 396
稼働時間ステータス変数, 854
カバリングインデックス, 5359
可変長型, 5361
可用性, 5361
カラム, 5359
カラム
 インデックス, 1500
 型, 1761
 選択, 274
 その他の型, 1830
 表示, 465
 変更, 2173
カラムインデックス, 5360
カラムエイリアス
 引用, 4609
 引用符を使用, 1636
 問題, 4609
カラムオプション
 mysqlimport, 442
カラム形式, 2224
カラムコメント, 2224
カラムテーブル
 データディクショナリテーブル, 896
カラムの格納, 2224
カラムプリフィクス, 5360
カラム名
 大/小文字の区別, 1639
カレンダー, 1927
環境変数, 299, 333, 1125
 AUTHENTICATION_LDAP_CLIENT_LOG, 550, 1206
 AUTHENTICATION_PAM_LOG, 550, 1171
 CC, 218, 550
 CXX, 218, 550
 DBI_TRACE, 550, 1019
 DBI_USER, 550
 HOME, 386, 550
 LDAPNOINIT, 1181
 LD_LIBRARY_PATH, 264
 LD_PRELOAD, 177
 LD_RUN_PATH, 264, 550
 LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN, 550
 LIBMYSQL_PLUGINS, 550
 LIBMYSQL_PLUGIN_DIR, 550
 MYSQLD_OPTS, 177
 MYSQLX_TCP_PORT, 550
 MYSQLX_UNIX_PORT, 550
 MYSQL_DEBUG, 299, 550, 1022
 MYSQL_GROUP_SUFFIX, 550
 MYSQL_HISTFILE, 386, 550
 MYSQL_HISTIGNORE, 386, 550
 MYSQL_HOME, 550
 MYSQL_HOST, 321, 550
 MYSQL_OPENSSL_UDF_DH_BITS_THRESHOLD, 550, 1391
 MYSQL_OPENSSL_UDF_DSA_BITS_THRESHOLD, 550, 1391
 MYSQL_OPENSSL_UDF_RSA_BITS_THRESHOLD, 550, 1391
 MYSQL_PS1, 550
 MYSQL_PWD, 550
 MYSQL_TCP_PORT, 299, 550, 1015, 1016

MYSQL_TEST_LOGIN_FILE, 308, 515, 550
MYSQL_TEST_TRACE_CRASH, 550
MYSQL_TEST_TRACE_DEBUG, 550
MYSQL_UNIX_PORT, 299, 550, 1015, 1016
NOTIFY_SOCKET, 177, 550
PATH, 131, 136, 228, 300, 550
PKG_CONFIG_PATH, 550
SUDO_USER, 4327
TMPDIR, 299, 550, 4603
TZ, 177, 550, 880, 4605
UMASK, 550, 4598
UMASK_DIR, 550, 4598
USER, 321, 550
のリスト, 550
韓国語, 4563
監査 API UDF
 audit_api_message_emit_udf(), 1346
監査プラグイン
 sha2_cache_cleaner, 1156
監査ログ UDF の読み取り
 audit_log_read(), 1307, 1332
 audit_log_read_bookmark(), 1307, 1334
監査ログ暗号化 UDF
 audit_log_encryption_password_get(), 1302, 1328
 audit_log_encryption_password_set(), 1302, 1329
監査ログのフィルタリング
 ルールベース, 1311
 レガシーモード, 1312, 1322, 1325
監査ログフィルタ UDF
 audit_log_filter_flush(), 1330
 audit_log_filter_remove_filter(), 1330
 audit_log_filter_remove_user(), 1331
 audit_log_filter_set_filter(), 1331
 audit_log_filter_set_user(), 1331
監視, 990, 1607, 2635, 2640, 2672, 2732, 2739, 2766, 2811, 2926, 2928
 threads, 1612
 マルチソースレプリケーション, 3064
関数, 1834
 GTID, 2088
 SELECT および WHERE 句の, 1834
 暗号化, 1985
 およびレプリケーション, 3219
 キャスト, 1952
 グループ化, 1860
 情報, 1993
 数学, 1875
 その他, 2128
関数従属性, 859, 2104, 2107
関数名
 あいまいさの解決, 1642
 構文解析, 1642
間接インデックス
 NDB Cluster, 2262
完全バックアップ, 5361
カンマ区切り値データ、読み取り, 2311, 2333
管理
 サーバー, 393
管理クライアント (NDB Cluster), 3741
 (参照 mgm)
管理接続インタフェース, 865, 867

- 管理ノード (NDB Cluster), 3734
 - (参照 [mgmd](#))
 - 定義済, 3450
- 管理プログラム, 298
- 関連性, 5361
- キー, 1500
 - 2 つを使用した検索, 291
 - 外部, 70, 290
 - マルチカラム, 1502
- キー管理
 - keyring, 1256
- キーキャッシュ
 - MyISAM, 1587
 - インデックスの割当て, 2598
- キー値ストア, 1505
- キー容量
 - MyISAM, 2991
- キーリングシステム変数, 1265
- キーリングプラグイン
 - keyring_aws, 1241
 - keyring_encrypted_file, 1235
 - keyring_file, 1234
 - keyring_hashicorp, 1243
 - keyring_oci, 1250
 - keyring_okv, 1236
- キーワード, 1645
- 記憶域要件
 - InnoDB テーブル, 1827
 - NDB Cluster, 1827
 - 空間データ型, 1830
 - 時間データ型, 1828
 - 数値データ型, 1827
 - データ型, 1826
 - 日付データ型, 1828
 - 文字列データ型, 1828
- 期限切れのパスワード, 1108
 - リセット, 1100
- 期限切れパスワードのリセット, 1100
- 疑似レコード, 5362
- 規則
 - 構文, 2
 - 表記, 2
- 既知のエラー, 4613
- 起動, 5362
 - mysqld, 1038
 - サーバーの, 221
 - サーバーを自動的に, 232
 - スレッドの状態, 1620
- 起動オプション
 - default, 302
 - レプリケーションチャンネル, 3159
- 起動パラメータ, 556
 - mysql, 363
 - mysqladmin, 396
- 機能, 削除 (参照 [削除された機能](#))
- 機能, 新機能 (参照 [新機能](#))
- 機能, 非推奨 (参照 [非推奨となった機能](#))
- 逆引用符, 5362
- キャスト, 1856, 1861, 1952
- キャスト演算子, 1952

- キャスト関数, 1952
 - 新機能, 26
- キャッシュ, 5361
 - クリア, 2600
- ギャップ, 5361
- ギャップイベント, 3993
- ギャップロック, 2701, 5362
 - InnoDB, 2717
- キャリッジリターン (`\r`), 1628, 2070, 2312
- 競合解決
 - mysqld 起動オプション, 4017
 - 有効化, 4018
- 競合解消
 - NDB Cluster レプリケーション, 4016
 - ndb_replication システムテーブル, 4018
- 競合検出ステータス変数
 - NDB Cluster レプリケーション, 4021
- 強制オプション
 - myisamchk, 498
 - myisampack, 510
 - mysql, 372
 - mysqldadmin, 400
 - mysqlcheck, 410
 - mysqldump, 427
 - mysqlexport, 443
 - mysql_upgrade, 359
- 強制性
 - collation, 1711
- 共通テーブル式, 289, 2366
 - 最適化, 1479, 1488
 - 新機能, 25
- 共有テーブルスペース, 5362
- 共有メモリートランスポート (参照 [NDB Cluster](#))
- 共有ロック, 2701, 5362
- 拒否オプション
 - ndb_import, 3773
- 切り捨て, 5362
- 近似値数値リテラル, 1630, 2142
- 近似値リテラル, 2142
- 近接検索, 1932
- クイックオプション
 - myisamchk, 500
 - mysql, 375
 - mysqlcheck, 412
 - mysqldump, 436
- 空間ウィンドウ関数, 2040
- 空間関数, 2003
 - 削除された機能, 40
- 空間クエリー
 - 最適化, 1501
- 空間値
 - 構文的に整形形式, 1803
- 空間データ型, 1793
 - SRID 属性, 1794
 - 記憶域要件, 1830
- クエリー, 5362
 - 速度, 1432
 - 入力, 266
 - パフォーマンスの推定, 1555
 - 例, 287

- クエリーオプション
 - mysqlslap, 481
 - ndb_config, 3750, 3750
 - ndb_index_stat, 3777
- クエリー拡張, 1937
- クエリーキャストの注入
 - 新機能, 28
- クエリーキャッシュ
 - および ndbinfo データベーステーブル, 3912
 - 削除された機能, 39
- クエリー後オプション
 - mysqlslap, 481
- クエリー実行計画, 5363
- クエリーステータス変数, 848
- クエリー属性, 1680
- クエリーリライトプラグイン
 - ddl_rewriter, 966
 - リライタ, 958
- 区切り文字がない文字列, 1631
- 行, 5364
 - 一致の問題, 4610
 - カウント, 281
 - 削除, 4610
 - 選択, 273
 - ソート, 275
- 行コンストラクタ, 2348
 - 最適化, 1478
- 行サイズ
 - maximum, 1523
- 行サブクエリー, 2348
- 行フォーマット, 5364
- 行ベースレプリケーション, 5364
 - デメリット, 3153
 - メリット, 3153
- 行レベルロック, 1593, 5365
- 行ロック, 5365
- 組み込み, 5364
- 組み込み一時テーブル, 5364
- クライアント, 5363
- クライアント
 - デバッグ, 1022
- クライアントからの受信
 - スレッドの状態, 1619
- クライアント側のプリペアドステートメント, 5363
- クライアント接続, 865
- クライアントツール, 4521
- クライアントに送信中
 - スレッドの状態, 1620
- クライアントプログラム, 297
- クライアントライブラリ, 5363
- クラスタ化されたインデックス, 5363
 - InnoDB, 2665
- クラスタデータベース (OBSOLETE) (参照 [NDB Cluster レプリケーション](#))
- クラスタリング (参照 [NDB Cluster](#))
- クラスタログ, 3858, 3859
- クラッシュ, 1016, 5363
 - 繰り返される, 4601
 - リカバリ, 1423
 - レプリケーション, 3227
- クラッシュセーフレプリケーション, 3095, 3193

- クラッシュリカバリ, 5363
 - InnoDB, 2939, 2941
- グラフオプション
 - ndb_top, 3827
- クリア
 - キャッシュ, 2600
- クリーンシャットダウン, 893, 1011, 3227, 5364
- クリーンページ, 5364
- グループ (NDB Cluster), 3716
- グループ化
 - 式, 1860
- グループ書き込みコンセンサス, 3271
- グループコミット, 2695, 5364
- グループプリファレンス
 - LDAP 認証, 1186
- グループレプリケーション UDF
 - group_replication_get_communication_protocol(), 2436
 - group_replication_get_write_concurrency(), 2435
 - group_replication_set_as_primary(), 2434
 - group_replication_set_communication_protocol(), 2437
 - group_replication_set_write_concurrency(), 2436
 - group_replication_switch_to_multi_primary_mode(), 2435
 - group_replication_switch_to_single_primary_mode(), 2435
- グループレプリケーション, 3241
 - 2 番目のインスタンスの追加, 3261
 - allowlist, 3304
 - background, 3242
 - details, 3250
 - group_replication_get_write_concurrency() UDF, 3271
 - group_replication_ip_allowlist, 3304
 - group_replication_ip_whitelist, 3304
 - group_replication_set_as_primary() UDF, 3270
 - group_replication_set_write_concurrency() UDF, 3271
 - group_replication_switch_to_multi_primary_mode() UDF, 3271
 - group_replication_switch_to_single_primary_mode() UDF, 3270
 - IP アドレス権限, 3304
 - ipv4 と ipv6 の混在, 3297
 - ipv6, 3297
 - MySQL Enterprise Backup, 3299
 - operations, 3269
 - Paxos, 3252
 - performance, 3311
 - replication_group_members テーブル, 3268
 - security, 3304
 - SSL サポート, 3306
 - summary, 3244
 - UDF, 3269, 3270, 3270, 3271, 3271, 3271
 - view, 3250
 - アップグレード, 3321
 - ある時点からのリカバリ, 3287
 - 一貫性による影響の保証, 3276
 - 一貫性保証, 3273
 - 一貫性保証でレベルを選択, 3275
 - 一貫性保証とデータフロー, 3273
 - 一貫性保証の構成, 3274
 - インスタンスの構成, 3255
 - インスタンスの追加, 3261, 3263
 - インスタンスのデプロイ, 3255
 - オフラインアップグレード, 3324
 - オンラインアップグレード, 3324

オンラインアップグレードに関する考慮事項, 3324
オンラインアップグレード方法, 3326
オンライングループの構成, 3269
可観測性, 3252
監視, 3266
起動, 3260
グループ, 3250
グループ書込みコンセンサス, 3271
グループ書込み同時実行性の検査, 3271
グループ書込み同時実行性の構成, 3271
グループ通信システム, 3252
グループ通信スレッド (GCT), 3311
グループ通信スレッドの微調整, 3311
グループメンバーシップ, 3250
グループモードの変更, 3270
サーバーの状態, 3267
システム変数, 3328
障害検出, 3251, 3316
障害検出に対するレスポンス, 3316
シングルプライマリモード, 3247
シングルプライマリモードへの変更, 3270
スロットル, 3312
制限事項, 3363
整合性によって同期ポイントが保証される, 3274
セキュアソケットレイヤーのサポート, 3306
選挙プロセス, 3247
単一プライマリモードでのデプロイ, 3254
通信プロトコル, 3271
データ定義言語ステートメント, 3249
トランザクションの一貫性保証, 3273
トランザクションの一貫性保証の理解, 3273
ネットワークのパーティション化, 3292
ネットワークパーティション, 3316
バージョンの結合, 3322
はじめに, 3254
パフォーマンス xcom キャッシュ, 3315
パフォーマンスメッセージの断片化, 3314
非同期レプリケーション, 3243
プライマリセカンダリレプリケーション, 3243
プライマリ選択への影響を保証する一貫性, 3277
プライマリであるメンバーの変更, 3270
プライマリの検索, 3248
プライマリフェイルオーバー, 3273
プライマリメンバーの変更, 3270
プラグインアーキテクチャー, 3252
フロー制御, 3312
プローブと統計, 3312
分散リカバリ, 3278
分散リカバリの構成, 3285
変更の表示, 3287
マルチプライマリモード, 3248
マルチプライマリモードとシングルプライマリモード, 3246
マルチプライマリモードへの変更, 3271
メッセージ圧縮, 3313
メンバーのアップグレード, 3324
モード, 3246
モードの選択, 3246
ユーザー資格証明, 3258
ユースケース, 3245
要件, 3361

- 要件と制限事項, 3361
- よくある質問, 3365
- 例ユースケースシナリオ, 3246
- レプリケーショングループメンバー統計, 3268
- レプリケーションテクノロジー, 3243
- グローバル一時テーブルスペース, 5364
- グローバル権限, 2502, 2515
- グローバルトランザクション, 5364
- クローンプラグイン, 978
 - clone_progress テーブル, 991
 - clone_status テーブル, 991
 - Com_clone ステータス変数, 994
 - 圧縮データのクローニング, 987
 - 暗号化された接続の構成, 986
 - 暗号化データのクローニング, 987
 - インストール, 980
 - 監視, 990
 - クローニング操作の停止, 994
 - システム変数, 995
 - 障害処理, 990
 - 新機能, 27
 - ステージイベントの監視, 992
 - 制限事項, 1000
 - ディレクトリとファイル, 989
 - 名前付きディレクトリへのクローニング, 985
 - パフォーマンススキーマインストゥルメント, 993
 - リモートクローニングの前提条件, 983
 - リモートデータのクローニング, 982, 985
 - レプリケーション用のクローニング, 987
 - ローカルでのデータのクローニング, 981
- 計画安定性, 5365
- 計算, 2142
 - カーディナリティ, 2569
 - 行セットの集計値, 2090
 - 日付, 276
- 形状的に有効
 - GIS 値, 1803
 - 空間値, 1803
- 桁, 1761
- 結合, 5365
 - Nested Loop アルゴリズム, 1453
 - USING と ON, 2338
- 結合アルゴリズム
 - Block Nested Loop, 1449
 - Nested Loop, 1449
- 結合オプション
 - myisampack, 510
- 結合タイプ
 - ALL, 1545
 - const, 1543
 - eq_ref, 1543
 - fulltext, 1544
 - index, 1545
 - index_merge, 1544
 - index_subquery, 1545
 - range, 1545
 - ref, 1544
 - ref_or_null, 1544
 - system, 1543
 - unique_subquery, 1544

- 権限
 - 変更, 1096
- 権限情報
 - ロケーション, 1062
- 権限チェック
 - 速度への影響, 1497, 1497
- 権限の制限
 - GRANT ステートメント, 2512
 - 部分的な取消し, 1091
- 権限の変更, 1096
- 言語オプション
 - mysqld, 652
- 言語サポート
 - エラーメッセージ, 1740
- 現在の行
 - ウィンドウ関数, 2117
- 検索
 - 2 つのキー, 291
 - full-text, 1927
 - MySQL Web ページ, 62
 - 大文字/小文字の区別, 4605
- 検索インデックス, 5365
- 減算 (-), 1873
- 原子的, 5365
- 厳密値リテラル, 2142
- 厳密モード, 5365
- 高位境界値, 5367
- 攻撃者
 - セキュリティ, 1036
- 貢献した会社
 - のリスト, 79
- 貢献者
 - のリスト, 73
- 降順インデックス, 1511, 5367
- 更新オプション
 - MySQLInstallerConsole, 126
 - ndb_index_stat, 3777
- 更新可能なビュー, 4112
- 構成オプション
 - MySQLInstallerConsole, 124
- 合成キー, 5366
- 高精度計算, 2142
- 構成ファイル, 1125, 5366
- 高速インデックス作成, 5367
- 高速オプション
 - mysamchk, 498
 - mysqlcheck, 410
- 高速シャットダウン, 5367
- 構文
 - 正規表現, 1917
- 構文規則, 2
- 構文的に整形式
 - GIS 値, 1803
 - 空間値, 1803
- コーディネータからのイベントの待機中
 - スレッドの状態, 1624
- コールドバックアップ, 5366
- 互換性
 - ODBC との, 1765, 1856, 1865, 2224, 2336
 - ODBC を使用, 778

- Oracle との, 2173, 2611
- Oracle を使用, 68, 2097
- PostgreSQL との, 69
- 標準 SQL との, 66
- コストモデル
 - オブティマイザ, 1581
- 固定小数点数演算, 2142
- 固定行フォーマット, 5366
- 子テーブル, 5366
- コネクタ, 5365
- コネクタ, 4521
- コマンド
 - バイナリ配布, 99
- コマンドインタセプタ, 5365
- コマンドオプション (NDB Cluster)
 - mysqld, 3657
 - ndbd, 3725
 - ndbinfo_select_all, 3732
 - ndb_mgm, 3741
 - ndb_mgmd, 3734
- コマンドオプション
 - mysql, 363
 - mysqladmin, 396
 - mysqld, 644
- コマンドオプションの優先順位, 300
- コマンド行オプション (NDB Cluster), 3834
- コマンド行履歴
 - mysql, 386
- コマンド構文, 3
- コマンドは同期されていません, 4596
- コマンド行ツール, 123, 362
- コミット, 5366
- コミットオプション
 - mysqslap, 477
- コメント
 - 開始, 71
 - 追加, 1683
- コメントオプション
 - mysql, 369
 - mysqldump, 426
- コメントの構文, 1683
- 孤立したストアオブジェクト, 4117
- 混合ステートメント (レプリケーション), 3232
- 混在モード挿入, 5367
- コンソールオプション
 - mysqld, 646
- コンテキストオプション
 - ndb_desc, 3759
- コンパクト行フォーマット, 5366
- コンパクトな行形式, 2779
- コンポーネント, 942
 - log_filter_dragnet, 944
 - log_filter_internal, 944
 - log_sink_internal, 944
 - log_sink_json, 944
 - log_sink_syseventlog, 945
 - log_sink_test, 945
 - query_attributes, 945
 - security, 1151
 - アンインストール, 942, 2540

- インストール, 942, 2538
- コンポーネントテーブル
 - システムテーブル, 898
- コンポーネント取り付け
 - validate_password, 1223
- コンポーネントのアンインストール, 942, 2540
 - validate_password, 1223
- コンポーネントのインストール, 942, 2538
- サーバー, 5368
 - 起動, 221
 - 起動と停止, 232
 - 起動の問題, 226
 - 再起動, 229
 - 接続, 265, 319
 - 切断, 265
 - デバッグ, 1016
 - 複数, 1009
 - ログ, 900
- サーバー側カーソル
 - 制限事項, 2451
- サーバー側のプリヘアドステートメント, 5368
- サーバー構成, 556
- サーバー接続
 - コマンドオプション, 312
- サーバーの管理, 393
- サーバープラグイン, 945
- サーバー変数, 2594 (参照 [システム変数](#))
- サービス
 - プラグインのための, 1000
- 再開オプション
 - ndb_import, 3773
- 再起動
 - サーバー, 229
- 再構成, 218
- 最小境界矩形, 2034
- 最小上限レコード, 5368
- サイズ
 - display, 1761
- 再接続
 - 自動, 4409
- 最大下限レコード, 5368
- 最適化 (NDB), 1446, 3674
- 最適化, 1430, 1441, 1490, 1530
 - Batched Key Access, 1460, 1462
 - BLOB 型, 1516
 - Block Nested Loop, 1460, 1461
 - DELETE ステートメント, 1497
 - DISTINCT, 1472
 - filesort, 1469, 1583
 - GROUP BY, 1470
 - INFORMATION_SCHEMA クエリー, 1492
 - InnoDB テーブル, 1525
 - INSERT ステートメント, 1496
 - LEFT JOIN, 1456
 - LIMIT 句, 1473
 - MEMORY ストレージエンジン, 1501
 - MEMORY テーブル, 1538
 - Multi-Range Read, 1459
 - MyISAM テーブル, 1535
 - ORDER BY, 1467

PERFORMANCE_SCHEMA, 1612
REPAIR TABLE ステートメント, 1537
RIGHT JOIN, 1456
SELECT ステートメント, 1432
SPATIAL インデックス, 1499
SQL ステートメント, 1432
UPDATE ステートメント, 1497
WHERE 句, 1433
インデックス, 1498
ウィンドウ関数, 1477
外部キー, 1500
外部結合, 1456
共通テーブル式, 1479
空間クエリー, 1501
行コンストラクタ, 1478
サブクエリー, 1479, 1484
サブクエリー実体化, 1483
主キー, 1499
数値型, 1515
全文クエリー, 1501
多数のテーブル, 1516
ディスク I/O, 1601
データサイズ, 1514
データ変更ステートメント, 1496
テーブル, 1426
導出テーブル, 1479
パフォーマンススキーマクエリー, 1494
非決定的関数, 1475
ビュー, 1479
ヒント, 1497
フルテーブルスキャン, 1479
ベンチマーク, 1611
メモリーの使用, 1605
文字および文字列型, 1516

最適化オプション
mysqlcheck, 411

サイレントオプション
myisamchk, 496
myisampack, 510
mysql, 377
mysqldadmin, 402
mysqlcheck, 412
mysqld_multi, 344
mysqlexport, 446
mysqslap, 482
ndb_perror, 3783
peror, 549

先読み, 5368

削減
データサイズ, 1514

削除, 5368
accounts, 1080, 1080
function, 2537
mysql.sock, 4605
user, 2501, 2501
users, 2501
インデックス, 2175, 2282
外部キー, 2176, 2252
行, 4610
主キー, 2175

- スキーマ, 2281
- データベース, 2281
- テーブル, 2284
- 削除オプション
 - mysqlimport, 443
 - ndb_index_stat, 3777
- 削除された機能, 37
 - bootstrap, 41
 - des-key-file, 40
 - fix-db-names, 41
 - fix-table-names, 41
 - ignore-db-dir, 39
 - log-warnings, 39
 - partition, 41
 - secure-auth, 39
 - skip-partition, 41
 - ssl, 40
 - ssl-verify-server-cert, 40
 - temp-pool, 40
- 5.1 より前のデータベース名変換, 41
- ALTER DATABASE, 41
- ALTER TABLE ... UPGRADE PARTITIONING, 44
- Area(), 40
- AsBinary(), 40
- AsText(), 40
- AsWKB(), 40
- AsWKT(), 40
- Buffer(), 40
- Centroid(), 40
- Com_alter_db_upgrade, 41
- Contains(), 40
- ConvexHull(), 40
- Crosses(), 40
- datetime_format, 39
- date_format, 39
- DB2, 39
- DECODE(), 40
- DES_DECRYPT(), 40
- DES_ENCRYPT(), 40
- DES_KEY_FILE, 40
- Dimension(), 40
- DISABLE_SHARED, 44
- Disjoint(), 40
- Distance(), 40
- DTrace, 44
- ENCODE(), 40
- ENCRYPT(), 40
- EndPoint(), 40
- Envelope(), 40
- Equals(), 40
- EXPLAIN EXTENDED, 39
- EXPLAIN PARTITIONS, 39
- ExteriorRing(), 40
- FILE_FORMAT, 44
- FLUSH QUERY CACHE, 39
- GeomCollFromText(), 40
- GeomCollFromWKB(), 40
- GeometryCollectionFromText(), 40
- GeometryCollectionFromWKB(), 40
- GeometryFromText(), 40

GeometryFromWKB(), 40
GeometryN(), 40
GeometryType(), 40
GeomFromText(), 40
GeomFromWKB(), 40
GLength(), 40
GLOBAL_STATUS, 41
GLOBAL_VARIABLES, 41
GRANT, 38
GROUP BY ソート, 39
have_crypt, 40
HAVE_CRYPT, 40
IDENTIFIED BY PASSWORD, 38
ignore_builtin_innodb, 40
ignore_db_dirs, 39
information_schema_stats, 38
InnoDB 圧縮一時テーブル, 43
InnoDB 共有テーブルスペース, 44
InnoDB リモートテーブルスペース, 44
Innodb_available_undo_logs, 44
innodb_file_format, 44
innodb_file_format_check, 44
innodb_file_format_max, 44
innodb_large_prefix, 44
INNODB_LOCKS, 43
innodb_locks_unsafe_for_binlog, 38
INNODB_LOCK_WAITS, 43
innodb_support_xa, 44
INNODB_SYS_COLUMNS, 38
INNODB_SYS_DATAFILES, 38
INNODB_SYS_FIELDS, 38
INNODB_SYS_FOREIGN, 38
INNODB_SYS_FOREIGN_COLS, 38
INNODB_SYS_INDEXES, 38
INNODB_SYS_TABLES, 38
INNODB_SYS_TABLESPACES, 38
INNODB_SYS_TABLESTATS, 38
INNODB_SYS_VIRTUAL, 38
innodb_undo_logs, 44
innodb_undo_tablespaces, 44
INSTALL_SCRIPTDIR, 41
InteriorRingN(), 40
internal_tmp_disk_storage_engine, 44
Intersects(), 40
IsClosed(), 40
IsEmpty(), 40
IsSimple(), 40
JSON_APPEND(), 44
libmysqld, 42
LineFromText(), 40
LineFromWKB(), 40
LineStringFromText(), 40
LineStringFromWKB(), 40
log_builtin_as_identified_by_password, 38
log_warnings, 39
M LineFromText(), 40
M LineFromWKB(), 40
MAXDB, 39
max_tmp_tables, 39
metadata_locks_cache_size, 39

metadata_locks_hash_instances, 39
MPointFromText(), 40
MPointFromWKB(), 40
MPolyFromText(), 40
MPolyFromWKB(), 40
MSSQL, 39
MultiPointFromText(), 40
MultiPointFromWKB(), 40
MultiPolygonFromText(), 40
MultiPolygonFromWKB(), 40
multi_range_count, 39
MYSQL323, 39
MYSQL40, 39
mysql_install_db, 41
MYSQL_OPT_SSL_ENFORCE, 40
MYSQL_OPT_SSL_VERIFY_SERVER_CERT, 40
mysql_plugin, 42
MYSQL_SECURE_AUTH, 39
NO_FIELD_OPTIONS, 39
NO_KEY_OPTIONS, 39
NO_TABLE_OPTIONS, 39
NULL としての\N, 40
NumGeometries(), 40
NumInteriorRings(), 40
NumPoints(), 40
old_passwords, 38
ORACLE, 39
Overlaps(), 40
PASSWORD(), 38
performance_timers, 42
perror, 44
PointFromText(), 40
PointFromWKB(), 40
PointN(), 40
PolyFromText(), 40
PolyFromWKB(), 40
PolygonFromText(), 40
PolygonFromWKB(), 40
POSTGRESQL, 39
PROCEDURE ANALYSE(), 40
Qcache_free_blocks, 39
Qcache_free_memory, 39
Qcache_inserts, 39, 39
Qcache_lowmem_prunes, 39
Qcache_not_cached, 39
Qcache_queries_in_cache, 39
Qcache_total_blocks, 39
query_cache_limit, 39
query_cache_min_res_unit, 39
query_cache_size, 39
query_cache_type, 39
query_cache_wlock_invalidate, 39
RESET QUERY CACHE, 39
resolveip, 42
resolve_stack_dump, 42
secure_auth, 39
SESSION_STATUS, 41
SESSION_VARIABLES, 41
setup_timers, 42
show_compatibility_56, 41

- Slave_heartbeat_period, 41
- Slave_last_heartbeat, 41
- Slave_received_heartbeats, 41
- Slave_retried_transactions, 41
- Slave_running, 41
- SQL モード, 39
- SQL_CACHE, 39
- sql_log_bin, 39
- SRID(), 40
- StartPoint(), 40
- sync_frm, 39
- time_format, 39
- Touches(), 40
- tx_isolation, 39
- tx_read_only, 39
- Within(), 40
- X(), 40
- Y(), 40
- 埋込みサーバーライブラリ, 42
- エラーコード, 42
- 空間関数, 40
- クエリーキャッシュ, 39
- スレッドの状態, 39
- マルチ LineStringFromText(), 40
- マルチ LineStringFromWKB(), 40
- ユーザー変数, 44
- 削除バッファリング, 5368
- 作成
 - function, 2536
 - スキーマ, 2189
 - データベース, 269, 2189
 - テーブル, 270
 - デフォルトの起動オプション, 302
 - バグレポート, 62
- 作成オプション
 - mysqlslap, 478
- サブクエリー, 2344 (参照 [サブクエリー](#))
 - ALL を使用した, 2347
 - ANY、IN、SOME を使用した, 2346
 - EXISTS を使用した, 2349
 - FROM 句 (参照 [導出テーブル](#))
 - NOT EXISTS を使用した, 2349
 - エラー, 2356
 - 行コンストラクタ, 2348
 - 最適化, 1479, 1484
 - 制約, 2357
 - 相関, 2350
- サブクエリー実体化, 1483
- サブパーティショニング, 4051, 4051
- サブパーティション
 - 既知の問題, 4087
- サブリスト, 5367
- サロゲートキー, 5367
- 算術演算子, 1975
- 算術関数, 1975
- 算術式, 1873
- 参照結合タイプ
 - オブティマイザ, 1544
- 参照整合性, 2626, 5368
- サンドボックスモード

- 期限切れのパスワードアカウント, 1108
- サンプリング
 - statement, 4378
- シーケンス, 292
- シェル構文, 3
- ジオメトリ, 1793
- ジオメトリ値
 - WKB 形式, 1802
 - WKT 形式, 1801
 - 内部記憶域形式, 1802
- 時間隔構文, 1678
- 時間間隔の構文, 1678
- 時間値
 - JSON, 1811
- 時間データ型
 - 記憶域要件, 1828
- 時間表現
 - イベントスケジューラ, 4107
- 時間リテラル, 1630
- 式
 - 拡張, 279
- 式のエイリアス, 2106, 2327
- 式の構文, 1676
- 識別子, 1635
 - 引用符付与, 1636
 - 大/小文字の区別, 1639
- 識別子の引用符付与, 1636
- 識別名
 - LDAP 認証, 1178
- シグナル
 - クライアント応答, 554
 - サーバーレスポンス, 552
 - 制限事項, 2476
- シグナル処理, 552
- 事後フィルタリング
 - パフォーマンススキーマ, 4250
- 辞書照合、ドイツ語, 1732, 1732
- システム
 - セキュリティー, 1032
- システムアカウント
 - アカウントカテゴリ, 1087
- システムオプション
 - ndb_config, 3750
- システムセッション
 - セッションカテゴリ, 1089
- システムテーブル
 - audit_log_filter テーブル, 899
 - audit_log_user テーブル, 899
 - columns_priv テーブル, 898, 1063
 - column_statistics テーブル, 896, 1584
 - db table, 230, 897, 1063
 - default_roles テーブル, 898, 1063
 - engine_cost, 1582
 - engine_cost テーブル, 899
 - firewall_groups テーブル, 899
 - firewall_group_allowlist テーブル, 899
 - firewall_membership テーブル, 899
 - firewall_users テーブル, 899
 - firewall_whitelist テーブル, 899
 - func table, 898, 1009

general_log テーブル, 898
global_grants テーブル, 897, 1061, 1063
gtid_executed テーブル, 899, 3035
help_category テーブル, 899
help_keyword テーブル, 899
help_relation テーブル, 899
help_topic テーブル, 899
innodb_dynamic_metadata テーブル, 900
innodb_index_stats テーブル, 899, 2747
innodb_table_stats テーブル, 899, 2747
ndb_binlog_index テーブル, 899, 3999
password_history テーブル, 898, 1063
procs_priv テーブル, 898, 1063
proxies_priv テーブル, 230, 898, 1063
role_edges テーブル, 898, 1063
servers テーブル, 900
server_cost, 1582
server_cost テーブル, 899
slave_master_info テーブル, 899
slave_relay_log_info テーブル, 899
slave_worker_info テーブル, 899
slow_log テーブル, 898
tables_priv テーブル, 897, 1063
time_zone テーブル, 899
time_zone_leap_second テーブル, 899
time_zone_name テーブル, 899
time_zone_transition テーブル, 899
time_zone_transition_type テーブル, 899
user table, 230, 897, 1062
オブティマイザ, 1543, 2331
コンポーネントテーブル, 898
タイムゾーンテーブル, 899
プラグインテーブル, 898
ヘルプテーブル, 898
システムテーブルスペース, 5368
システムテーブルのロック
スレッドの状態, 1618
システムテーブルを開く
スレッドの状態, 1618
システム変数, 669, 806, 2594
activate_all_roles_on_login, 671
admin_address, 671
admin_port, 672
admin_ssl_ca, 672
admin_ssl_capath, 673
admin_ssl_cert, 673
admin_ssl_cipher, 673
admin_ssl_crl, 674
admin_ssl_crlpath, 674
admin_ssl_key, 674
admin_tls_ciphersuites, 675
admin_tls_version, 675
audit_log_buffer_size, 1336
audit_log_compression, 1336
audit_log_connection_policy, 1336
audit_log_current_session, 1337
audit_log_encryption, 1337
audit_log_exclude_accounts, 1338
audit_log_file, 1307, 1338
audit_log_filter_id, 1339

audit_log_flush, 1339
audit_log_format, 1339
audit_log_include_accounts, 1339
audit_log_password_history_keep_days, 1340
audit_log_policy, 1341
audit_log_prune_seconds, 1342
audit_log_read_buffer_size, 1310, 1342
audit_log_rotate_on_size, 1343
audit_log_statement_policy, 1343
audit_log_strategy, 1344
authentication_ldap_sasl_auth_method_name, 1201
authentication_ldap_sasl_bind_base_dn, 1202
authentication_ldap_sasl_bind_root_dn, 1202
authentication_ldap_sasl_bind_root_pwd, 1203
authentication_ldap_sasl_ca_path, 1203
authentication_ldap_sasl_group_search_attr, 1203
authentication_ldap_sasl_group_search_filter, 1204
authentication_ldap_sasl_init_pool_size, 1204
authentication_ldap_sasl_log_status, 1205
authentication_ldap_sasl_max_pool_size, 1206
authentication_ldap_sasl_referral, 1206
authentication_ldap_sasl_server_host, 1206
authentication_ldap_sasl_server_port, 1207
authentication_ldap_sasl_tls, 1207
authentication_ldap_sasl_user_search_attr, 1208
authentication_ldap_simple_auth_method_name, 1208
authentication_ldap_simple_bind_base_dn, 1208
authentication_ldap_simple_bind_root_dn, 1209
authentication_ldap_simple_bind_root_pwd, 1209
authentication_ldap_simple_ca_path, 1210
authentication_ldap_simple_group_search_attr, 1210
authentication_ldap_simple_group_search_filter, 1210
authentication_ldap_simple_init_pool_size, 1211
authentication_ldap_simple_log_status, 1212
authentication_ldap_simple_max_pool_size, 1212
authentication_ldap_simple_referral, 1213
authentication_ldap_simple_server_host, 1213
authentication_ldap_simple_server_port, 1214
authentication_ldap_simple_tls, 1215
authentication_ldap_simple_user_search_attr, 1215
authentication_windows_log_level, 675
authentication_windows_use_principal_name, 676
autocommit, 676
automatic_sp_privileges, 677
auto_generate_certs, 677
auto_increment_increment, 3073
auto_increment_offset, 3075
avoid_temporal_upgrade, 678
back_log, 678
basedir, 679
big_tables, 679
bind_address, 679
binlog_cache_size, 3116
binlog_checksum, 3117
binlog_direct_non_transactional_updates, 3117
binlog_encryption, 3118
binlog_error_action, 3119
binlog_expire_logs_seconds, 3119
binlog_format, 3120
binlog_group_commit_sync_delay, 3122

binlog_group_commit_sync_no_delay_count, 3122
binlog_gtid_simple_recovery, 3138
binlog_max_flush_queue_time, 3123
binlog_order_commits, 3123
binlog_rotate_encryption_master_key_at_startup, 3124
binlog_rows_query_log_events, 3127
binlog_row_event_max_size, 3124
binlog_row_image, 3124
binlog_row_metadata, 3126
binlog_row_value_options, 3126
binlog_stmt_cache_size, 3128
binlog_transaction_compression, 3128
binlog_transaction_compression_level_zstd, 3129
binlog_transaction_dependency_history_size, 3130
binlog_transaction_dependency_tracking, 3129
block_encryption_mode, 681
bulk_insert_buffer_size, 682, 2990
caching_sha2_password_auto_generate_rsa_keys, 683
caching_sha2_password_digest_rounds, 682
caching_sha2_password_private_key_path, 683
caching_sha2_password_public_key_path, 684
character_sets_dir, 686
character_set_client, 684
character_set_connection, 685
character_set_database, 685
character_set_filesystem, 685
character_set_results, 686
character_set_server, 686
character_set_system, 686
check_proxy_users, 687, 1120
clone_autotune_concurrency, 995
clone_buffer_size, 996
clone_ddl_timeout, 996
clone_enable_compression, 997
clone_max_concurrency, 997
clone_max_data_bandwidth, 997
clone_max_network_bandwidth, 998
clone_ssl_ca, 998
clone_ssl_cert, 999
clone_ssl_key, 999
clone_valid_donor_list, 999
collation_connection, 687
collation_database, 687
collation_server, 688
completion_type, 688
concurrent_insert, 689
connection_control_failed_connections_threshold, 1219
connection_control_max_connection_delay, 1220
connection_control_min_connection_delay, 1220
connect_timeout, 690
core_file, 690
create_admin_listener_thread, 690
cte_max_recursion_depth, 691
daemon_memcached_enable_binlog, 2820
daemon_memcached_engine_lib_name, 2821
daemon_memcached_engine_lib_path, 2821
daemon_memcached_option, 2821
daemon_memcached_r_batch_size, 2822
daemon_memcached_w_batch_size, 2822
datadir, 691

debug, 691
debug_sync, 692
default_authentication_plugin, 693
default_collation_for_utf8mb4, 693
default_password_lifetime, 694
default_storage_engine, 694
default_table_encryption, 695
default_tmp_storage_engine, 695
default_week_format, 696
delayed_insert_limit, 697
delayed_insert_timeout, 697
delayed_queue_size, 698
delay_key_write, 696, 2990
disabled_storage_engines, 698
disconnect_on_expired_password, 699
div_precision_increment, 699
dragnet.log_error_filter_rules, 700
end_markers_in_json, 700
enforce_gtid_consistency, 3139
error_count, 701
event_scheduler, 701
expire_logs_days, 3131
explicit_defaults_for_timestamp, 702
external_user, 703
flush, 704
flush_time, 704
foreign_key_checks, 704
ft_boolean_syntax, 705
ft_max_word_len, 705
ft_min_word_len, 706
ft_query_expansion_limit, 706
ft_stopword_file, 706
general_log, 707
general_log_file, 707
generated_random_password_length, 707
group_concat_max_len, 708
group_replication_advertise_recovery_endpoints, 3329
group_replication_allow_local_lower_version_join, 3330
group_replication_autorejoin_tries, 3332
group_replication_auto_increment_increment, 3331
group_replication_bootstrap_group, 3332
group_replication_clone_threshold, 3333
group_replication_communication_debug_options, 3334
group_replication_communication_max_message_size, 3334
group_replication_components_stop_timeout, 3335
group_replication_compression_threshold, 3335
group_replication_consistency, 3336
group_replication_enforce_update_everywhere_checks, 3337
group_replication_exit_state_action, 3338
group_replication_flow_control_applier_threshold, 3339
group_replication_flow_control_certifier_threshold, 3339
group_replication_flow_control_hold_percent, 3340
group_replication_flow_control_max_commit_quota, 3340
group_replication_flow_control_member_quota_percent, 3341
group_replication_flow_control_min_quota, 3341
group_replication_flow_control_min_recovery_quota, 3341
group_replication_flow_control_mode, 3342
group_replication_flow_control_period, 3342
group_replication_flow_control_release_percent, 3342
group_replication_force_members, 3343

group_replication_group_name, 3343
group_replication_group_seeds, 3344
group_replication_gtid_assignment_block_size, 3345
group_replication_ip_allowlist, 3345
group_replication_ip_whitelist, 3346
group_replication_local_address, 3347
group_replication_member_expel_timeout, 3348
group_replication_member_weight, 3349
group_replication_message_cache_size, 3349
group_replication_poll_spin_loops, 3350
group_replication_recovery_complete_at, 3350
group_replication_recovery_compression_algorithm, 3351
group_replication_recovery_get_public_key, 3351
group_replication_recovery_public_key_path, 3352
group_replication_recovery_reconnect_interval, 3352
group_replication_recovery_retry_count, 3353
group_replication_recovery_ssl_ca, 3353
group_replication_recovery_ssl_capath, 3353
group_replication_recovery_ssl_cert, 3354
group_replication_recovery_ssl_cipher, 3354
group_replication_recovery_ssl_crl, 3354
group_replication_recovery_ssl_crlpath, 3355
group_replication_recovery_ssl_key, 3355
group_replication_recovery_ssl_verify_server_cert, 3355
group_replication_recovery_tls_ciphersuites, 3356
group_replication_recovery_tls_version, 3356
group_replication_recovery_use_ssl, 3357
group_replication_recovery_zstd_compression_level, 3357
group_replication_single_primary_mode, 3357
group_replication_ssl_mode, 3358
group_replication_start_on_boot, 3358
group_replication_tls_source, 3359
group_replication_transaction_size_limit, 3359
group_replication_unreachable_majority_timeout, 3360
gtid_executed, 3140
gtid_executed_compression_period, 3140
gtid_mode, 3141
gtid_next, 3142
gtid_owned, 3142
gtid_purged, 3143
have_compress, 708
have_dynamic_loading, 708
have_geometry, 708
have_openssl, 708
have_profiling, 708
have_query_cache, 708
have_rtree_keys, 708
have_ssl, 709
have_statement_timeout, 709
have_symlink, 709
hintable, 1576
histogram_generation_max_mem_size, 709
hostname, 710
identity, 711
immediate_server_version, 3076
information_schema_stats_expiry, 711
init_connect, 711
init_file, 712
init_slave, 3091
innodb_adaptive_flushing, 2822

innodb_adaptive_flushing_lwm, 2823
innodb_adaptive_hash_index, 2823
innodb_adaptive_hash_index_parts, 2823
innodb_adaptive_max_sleep_delay, 2824
innodb_api_bk_commit_interval, 2824
innodb_api_disable_rowlock, 2825
innodb_api_enable_binlog, 2825
innodb_api_enable_mdln, 2825
innodb_api_trx_level, 2825
innodb_autoextend_increment, 2826
innodb_autoinc_lock_mode, 2826
innodb_background_drop_list_empty, 2827
innodb_buffer_pool_chunk_size, 2827
innodb_buffer_pool_debug, 2828
innodb_buffer_pool_dump_at_shutdown, 2828
innodb_buffer_pool_dump_now, 2829
innodb_buffer_pool_dump_pct, 2829
innodb_buffer_pool_filename, 2829
innodb_buffer_pool_instances, 2830
innodb_buffer_pool_in_core_file, 2830
innodb_buffer_pool_load_abort, 2831
innodb_buffer_pool_load_at_startup, 2831
innodb_buffer_pool_load_now, 2831
innodb_buffer_pool_size, 2832
innodb_change_buffering, 2833
innodb_change_buffering_debug, 2834
innodb_change_buffer_max_size, 2833
innodb_checkpoint_disabled, 2834
innodb_checksum_algorithm, 2835
innodb_cmp_per_index_enabled, 2836
innodb_commit_concurrency, 2836
innodb_compression_failure_threshold_pct, 2837
innodb_compression_level, 2838
innodb_compression_pad_pct_max, 2838
innodb_compress_debug, 2837
innodb_concurrency_tickets, 2838
innodb_data_file_path, 2839
innodb_data_home_dir, 2840
innodb_ddl_log_crash_reset_debug, 2840
innodb_deadlock_detect, 2840
innodb_dedicated_server, 2841
innodb_default_row_format, 2841
innodb_directories, 2842
innodb_disable_sort_file_cache, 2842
innodb_doublewrite, 2843
innodb_doublewrite_batch_size, 2843
innodb_doublewrite_dir, 2843
innodb_doublewrite_files, 2844
innodb_doublewrite_pages, 2844
innodb_extend_and_initialize, 2844
innodb_fast_shutdown, 2845
innodb_file_per_table, 2846
innodb_fill_factor, 2846
innodb_fil_make_page_dirty_debug, 2845
innodb_flushing_avg_loops, 2851
innodb_flush_log_at_timeout, 2847
innodb_flush_log_at_trx_commit, 2847
innodb_flush_method, 2848
innodb_flush_neighbors, 2850
innodb_flush_sync, 2850

innodb_force_load_corrupted, 2851
innodb_force_recovery, 2851
innodb_fsync_threshold, 2852
innodb_ft_aux_table, 2852
innodb_ft_cache_size, 2853
innodb_ft_enable_diag_print, 2853
innodb_ft_enable_stopword, 2854
innodb_ft_max_token_size, 2854
innodb_ft_min_token_size, 2854
innodb_ft_num_word_optimize, 2855
innodb_ft_result_cache_limit, 2855
innodb_ft_server_stopword_table, 2856
innodb_ft_sort_pll_degree, 2856
innodb_ft_total_cache_size, 2856
innodb_ft_user_stopword_table, 2857
innodb_idle_flush_pct, 2857
innodb_io_capacity, 2858
innodb_io_capacity_max, 2858
innodb_limit_optimistic_insert_debug, 2858
innodb_lock_wait_timeout, 2859
innodb_log_buffer_size, 2859
innodb_log_checkpoint_fuzzy_now, 2860
innodb_log_checkpoint_now, 2860
innodb_log_checksums, 2860
innodb_log_compressed_pages, 2861
innodb_log_files_in_group, 2862
innodb_log_file_size, 2861
innodb_log_group_home_dir, 2862
innodb_log_spin_cpu_abs_lwm, 2862
innodb_log_spin_cpu_pct_hwm, 2863
innodb_log_wait_for_flush_spin_hwm, 2863
innodb_log_writer_threads, 2864
innodb_log_write_ahead_size, 2864
innodb_lru_scan_depth, 2864
innodb_max_dirty_pages_pct, 2865
innodb_max_dirty_pages_pct_lwm, 2866
innodb_max_purge_lag, 2866
innodb_max_purge_lag_delay, 2866
innodb_max_undo_log_size, 2867
innodb_merge_threshold_set_all_debug, 2867
innodb_monitor_disable, 2867
innodb_monitor_enable, 2867
innodb_monitor_reset, 2868
innodb_monitor_reset_all, 2868
innodb_numa_interleave, 2869
innodb_old_blocks_pct, 2869
innodb_old_blocks_time, 2869
innodb_online_alter_log_max_size, 2870
innodb_open_files, 2870
innodb_optimize_fulltext_only, 2871
innodb_page_cleaners, 2871
innodb_page_size, 2872
innodb_parallel_read_threads, 2873
innodb_print_ddl_logs, 2874
innodb_purge_batch_size, 2874
innodb_purge_rseg_truncate_frequency, 2875
innodb_purge_threads, 2875
innodb_random_read_ahead, 2875
innodb_read_ahead_threshold, 2876
innodb_read_io_threads, 2876

innodb_read_only, 2877
innodb_redo_log_archive_dirs, 2878
innodb_redo_log_encrypt, 2878
innodb_replication_delay, 2878
innodb_rollback_on_timeout, 2879
innodb_rollback_segments, 2879
innodb_saved_page_number_debug, 2879
innodb_sort_buffer_size, 2880
innodb_spin_wait_delay, 2880
innodb_spin_wait_pause_multiplier, 2881
innodb_stats_auto_recalc, 2881
innodb_stats_include_delete_marked, 2749, 2882
innodb_stats_method, 2882
innodb_stats_on_metadata, 2882
innodb_stats_persistent_sample_pages, 2883
innodb_stats_transient_sample_pages, 2884
innodb_status_output, 2884
innodb_status_output_locks, 2885
innodb_strict_mode, 2885
innodb_sync_array_size, 2886
innodb_sync_debug, 2886
innodb_sync_spin_loops, 2886
innodb_table_locks, 2886
innodb_temp_data_file_path, 2887
innodb_temp_tablespaces_dir, 2888
innodb_thread_concurrency, 2888
innodb_thread_sleep_delay, 2889
innodb_tmpdir, 2890
innodb_trx_purge_view_update_only_debug, 2891
innodb_trx_rseg_n_slots_debug, 2891
innodb_undo_directory, 2891
innodb_undo_log_encrypt, 2892
innodb_undo_log_truncate, 2892
innodb_undo_tablespaces, 2892
innodb_use_native_aio, 2893
innodb_validate_tablespace_paths, 2893
innodb_version, 2894
innodb_write_io_threads, 2894
insert_id, 713
interactive_timeout, 713
internal_tmp_disk_storage_engine, 713
internal_tmp_mem_storage_engine, 714
join_buffer_size, 714
keep_files_on_create, 715
keyring_aws_cmk_id, 1265
keyring_aws_conf_file, 1265
keyring_aws_data_file, 1266
keyring_aws_region, 1266
keyring_encrypted_file_data, 1267
keyring_encrypted_file_password, 1268
keyring_file_data, 1268
keyring_hashicorp_auth_path, 1269
keyring_hashicorp_caching, 1270
keyring_hashicorp_ca_path, 1270
keyring_hashicorp_commit_auth_path, 1270
keyring_hashicorp_commit_caching, 1271
keyring_hashicorp_commit_ca_path, 1271
keyring_hashicorp_commit_role_id, 1271
keyring_hashicorp_commit_server_url, 1271
keyring_hashicorp_commit_store_path, 1272

keyring_hashicorp_role_id, 1272
keyring_hashicorp_secret_id, 1272
keyring_hashicorp_server_url, 1273
keyring_hashicorp_store_path, 1273
keyring_oci_ca_certificate, 1273
keyring_oci_compartment, 1274
keyring_oci_encryption_endpoint, 1274
keyring_oci_key_file, 1274
keyring_oci_key_fingerprint, 1275
keyring_oci_management_endpoint, 1275
keyring_oci_master_key, 1276
keyring_oci_secrets_endpoint, 1276
keyring_oci_tenancy, 1276
keyring_oci_user, 1277
keyring_oci_vaults_endpoint, 1277
keyring_oci_virtual_vault, 1277
keyring_okv_conf_dir, 1278
keyring_operations, 1278
key_buffer_size, 715
key_cache_age_threshold, 716
key_cache_block_size, 717
key_cache_division_limit, 717
large_files_support, 717
large_pages, 717
large_page_size, 718
last_insert_id, 718
lc_messages, 718
lc_messages_dir, 718
lc_time_names, 719
license, 719
local_infile, 719, 1040
locked_in_memory, 720
lock_order, 1023
lock_order_debug_loop, 1024
lock_order_debug_missing_arc, 1024
lock_order_debug_missing_key, 1024
lock_order_debug_missing_unlock, 1025
lock_order_dependencies, 1025
lock_order_extra_dependencies, 1025
lock_order_output_directory, 1025
lock_order_print_txt, 1026
lock_order_trace_loop, 1026
lock_order_trace_missing_arc, 1026
lock_order_trace_missing_key, 1027
lock_order_trace_missing_unlock, 1027
lock_wait_timeout, 719
log_bin, 3131
log_bin_basename, 3132
log_bin_index, 3132
log_bin_trust_function_creators, 3132
log_bin_use_v1_row_events, 3133
log_error, 720
log_error_services, 720
log_error_suppression_list, 721
log_error_verbosity, 721
log_output, 722
log_queries_not_using_indexes, 723
log_raw, 723
log_slave_updates, 3133
log_slow_extra, 723

log_slow_slave_statements, 3091
log_statements_unsafe_for_binlog, 3133
log_syslog, 724
log_syslog_facility, 724
log_syslog_include_pid, 724
log_syslog_tag, 725
log_throttle_queries_not_using_indexes, 725
log_timestamps, 725
long_query_time, 726
lower_case_file_system, 726
lower_case_table_names, 727
low_priority_updates, 726
mandatory_roles, 728
master_info_repository, 3091
master_verify_checksum, 3134
max_allowed_packet, 728
max_binlog_cache_size, 3134
max_binlog_size, 3134
max_binlog_stmt_cache_size, 3135
max_connections, 729
max_connect_errors, 729
max_delayed_threads, 730
max_digest_length, 730
max_error_count, 731
max_execution_time, 731
max_heap_table_size, 731
max_insert_delayed_threads, 732
max_join_size, 391, 732
max_length_for_sort_data, 733
max_points_in_geometry, 733
max_prepared_stmt_count, 733
max_relay_log_size, 3092
max_seeks_for_key, 734
max_sort_length, 734
max_sp_recursion_depth, 734
max_user_connections, 735
max_write_lock_count, 735
mecab_rc_file, 736
metadata_locks_cache_size, 736
metadata_locks_hash_instances, 736
min_examined_row_limit, 737
myisam_data_pointer_size, 737
myisam_max_sort_file_size, 737, 2990
myisam_mmap_size, 738
myisam_recover_options, 738, 2990
myisam_repair_threads, 739
myisam_sort_buffer_size, 739, 2990
myisam_stats_method, 740
myisam_use_mmap, 740
mysqld, 557
mysqlx_bind_address, 3422
mysqlx_compression_algorithms, 3423
mysqlx_connect_timeout, 3424
mysqlx_deflate_default_compression_level, 3424
mysqlx_deflate_max_client_compression_level, 3424
mysqlx_document_id_unique_prefix, 3425
mysqlx_enable_hello_notice, 3425
mysqlx_idle_worker_thread_timeout, 3425
mysqlx_interactive_timeout, 3425
mysqlx_lz4_default_compression_level, 3426

mysqlx_lz4_max_client_compression_level, 3426
mysqlx_max_allowed_packet, 3426
mysqlx_max_connections, 3427
mysqlx_min_worker_threads, 3427
mysqlx_port, 3427
mysqlx_port_open_timeout, 3428
mysqlx_read_timeout, 3428
mysqlx_socket, 3428
mysqlx_ssl_ca, 3429
mysqlx_ssl_capath, 3429
mysqlx_ssl_cert, 3430
mysqlx_ssl_cipher, 3430
mysqlx_ssl_crl, 3430
mysqlx_ssl_crlpath, 3430
mysqlx_ssl_key, 3431
mysqlx_wait_timeout, 3431
mysqlx_write_timeout, 3431
mysqlx_zstd_default_compression_level, 3432
mysqlx_zstd_max_client_compression_level, 3432
mysql_firewall_mode, 1370
mysql_firewall_trace, 1371
mysql_native_password_proxy_users, 740, 1121
named_pipe, 741
named_pipe_full_access_group, 741
ndbinfo_database, 3687
ndbinfo_max_bytes, 3687
ndbinfo_max_rows, 3687
ndbinfo_offline, 3688
ndbinfo_show_hidden, 3688
ndbinfo_table_prefix, 3688
ndbinfo_version, 3688, 3688
ndb_autoincrement_prefetch_sz, 3667
ndb_cache_check_time, 3667
ndb_clear_apply_status, 3668
ndb_conflict_role, 3668
ndb_data_node_neighbour, 3669
ndb_dbg_check_shares, 3669
ndb_default_column_format, 3670
ndb_deferred_constraints, 3670
ndb_distribution, 3670
ndb_eventbuffer_free_percent, 3671
ndb_eventbuffer_max_alloc, 3671
ndb_extra_logging, 3671
ndb_force_send, 3672
ndb_fully_replicated, 3672
ndb_index_stat_enable, 3672
ndb_index_stat_option, 3672
ndb_join_pushdown, 3674
ndb_log_apply_status, 3675
ndb_log_bin, 3676
ndb_log_binlog_index, 3676
ndb_log_empty_epochs, 3676
ndb_log_empty_update, 3676
ndb_log_exclusive_reads, 3677
ndb_log_orig, 3677
ndb_log_transaction_id, 3677
ndb_metadata_check, 3678
ndb_metadata_check_interval, 3678
ndb_metadata_sync, 3678
ndb_optimized_node_selection, 3679

ndb_read_backup, 3680
ndb_recv_thread_activation_threshold, 3680
ndb_recv_thread_cpu_mask, 3680
ndb_report_thresh_binlog_epoch_slip, 3681
ndb_report_thresh_binlog_mem_usage, 3681
ndb_row_checksum, 3681
ndb_schema_dist_lock_wait_timeout, 3682
ndb_schema_dist_timeout, 3682
ndb_schema_dist_upgrade_allowed, 3682
ndb_show_foreign_key_mock_tables, 3683
ndb_slave_conflict_role, 3683
ndb_table_no_logging, 3683
ndb_table_temporary, 3684
ndb_use_copying_alter_table, 3684
ndb_use_exact_count, 3685
ndb_use_transactions, 3685
ndb_version, 3685
ndb_version_string, 3685
net_buffer_length, 742
net_read_timeout, 742
net_retry_count, 742
net_write_timeout, 743
new, 743
ngram_token_size, 743
offline_mode, 744
old, 744
old_alter_table, 744
open_files_limit, 745
optimizer_prune_level, 746
optimizer_search_depth, 746
optimizer_switch, 746
optimizer_trace, 750
optimizer_trace_features, 751
optimizer_trace_limit, 751
optimizer_trace_max_mem_size, 751
optimizer_trace_offset, 751
original_commit_timestamp, 3135
original_server_version, 3076
parser_max_mem_size, 752
partial_revokes, 752
password_history, 753
password_require_current, 753
password_reuse_interval, 754
performance_schema, 4420
performance_schema_accounts_size, 4420
performance_schema_digests_size, 4421
performance_schema_error_size, 4421
performance_schema_events_stages_history_long_size, 4422
performance_schema_events_stages_history_size, 4422
performance_schema_events_statements_history_long_size, 4422
performance_schema_events_statements_history_size, 4422
performance_schema_events_transactions_history_long_size, 4423
performance_schema_events_transactions_history_size, 4423
performance_schema_events_waits_history_long_size, 4423
performance_schema_events_waits_history_size, 4423
performance_schema_hosts_size, 4424
performance_schema_max_cond_classes, 4424
performance_schema_max_cond_instances, 4424
performance_schema_max_digest_length, 4425
performance_schema_max_digest_sample_age, 4425

performance_schema_max_file_classes, 4426
performance_schema_max_file_handles, 4426
performance_schema_max_file_instances, 4426
performance_schema_max_index_stat, 4427
performance_schema_max_memory_classes, 4427
performance_schema_max_metadata_locks, 4427
performance_schema_max_mutex_classes, 4427
performance_schema_max_mutex_instances, 4428
performance_schema_max_prepared_statements_instances, 4428
performance_schema_max_program_instances, 4429
performance_schema_max_rwlock_classes, 4428
performance_schema_max_rwlock_instances, 4429
performance_schema_max_socket_classes, 4429
performance_schema_max_socket_instances, 4430
performance_schema_max_sql_text_length, 4430
performance_schema_max_stage_classes, 4430
performance_schema_max_statement_classes, 4431
performance_schema_max_statement_stack, 4431
performance_schema_max_table_handles, 4431
performance_schema_max_table_instances, 4432
performance_schema_max_table_lock_stat, 4432
performance_schema_max_thread_classes, 4432
performance_schema_max_thread_instances, 4433
performance_schema_session_connect_attrs_size, 4433
performance_schema_setup_actors_size, 4434
performance_schema_setup_objects_size, 4434
performance_schema_show_processlist, 4434
performance_schema_users_size, 4435
persist-restricted, 830
persisted_globals_load, 754, 826
persist_only_admin_x509_subject, 754
pid_file, 755
plugin_dir, 755
port, 755
preload_buffer_size, 756
print_identified_with_as_hex, 756
profiling_history_size, 756
protocol_compression_algorithms, 757
protocol_version, 757
proxy_user, 758
pseudo_slave_mode, 758
pseudo_thread_id, 759
query_alloc_block_size, 759
query_prealloc_size, 759
rand_seed1, 760
rand_seed2, 760
range_alloc_block_size, 760
range_optimizer_max_mem_size, 760
rbr_exec_mode, 761
read_buffer_size, 761
read_only, 762
read_rnd_buffer_size, 763
regexp_stack_limit, 763
regexp_time_limit, 764
relay_log, 3092
relay_log_basename, 3093
relay_log_index, 3093
relay_log_info_file, 3094
relay_log_info_repository, 3094
relay_log_purge, 3095

relay_log_recovery, 3095
relay_log_space_limit, 3096
replication_optimize_for_static_plugin_config, 3096
replication_sender_observe_commit_only, 3096
report_host, 3097
report_password, 3097
report_port, 3097
report_user, 3098
require_row_format, 764
require_secure_transport, 764
resultset_metadata, 765
rewriter_enabled, 965
rewriter_verbose, 965
rpl_read_size, 3098
rpl_semi_sync_master_enabled, 3077
rpl_semi_sync_master_timeout, 3077
rpl_semi_sync_master_trace_level, 3077
rpl_semi_sync_master_wait_for_slave_count, 3078
rpl_semi_sync_master_wait_no_slave, 3078
rpl_semi_sync_master_wait_point, 3079
rpl_semi_sync_slave_enabled, 3099
rpl_semi_sync_slave_trace_level, 3099
rpl_stop_slave_timeout, 3099
schema_definition_cache, 766
secondary_engine_cost_threshold, 765
secure_file_priv, 766
select_into_buffer_size, 767
select_into_disk_sync, 767
select_into_disk_sync_delay, 768
server_id, 3065
server_id_bits, 3686, 3686
session_track_gtids, 768
session_track_schema, 769
session_track_state_change, 769
session_track_system_variables, 770
session_track_transaction_info, 770
SET_VAR オブティマイザヒント, 1576
sha256_password_auto_generate_rsa_keys, 771
sha256_password_private_key_path, 771
sha256_password_proxy_users, 772, 1121
sha256_password_public_key_path, 772
shared_memory, 773
shared_memory_base_name, 773
show_create_table_skip_secondary_engine, 773
show_create_table_verbosity, 774
show_old_temporals, 774
skip_external_locking, 774
skip_name_resolve, 775
skip_networking, 775
skip_show_database, 776
slave_allow_batching, 3686
slave_checkpoint_group, 3100
slave_checkpoint_period, 3100
slave_compressed_protocol, 3101
slave_exec_mode, 3102
slave_load_tmpdir, 3102
slave_max_allowed_packet, 3103
slave_net_timeout, 3103
slave_parallel_type, 3104
slave_parallel_workers, 3105

slave_pending_jobs_size_max, 3105
slave_preserve_commit_order, 3106
slave_rows_search_algorithms, 3107
slave_skip_errors, 3108
slave_sql_verify_checksum, 3108
slave_transaction_retries, 3108
slave_type_conversions, 3109
slow_launch_time, 776
slow_query_log, 776
slow_query_log_file, 777
socket, 777
sort_buffer_size, 777
sql_auto_is_null, 778
sql_big_selects, 778
sql_buffer_result, 779
sql_log_bin, 3136
sql_log_off, 779
sql_mode, 779
sql_notes, 780
sql_quote_show_create, 781
sql_require_primary_key, 781
sql_safe_updates, 391, 782
sql_select_limit, 391, 782
sql_slave_skip_counter, 3109
sql_warnings, 782
ssl_ca, 783
ssl_capath, 783
ssl_cert, 783
ssl_cipher, 784
ssl_crl, 784
ssl_crlpath, 785
ssl_fips_mode, 785
ssl_key, 786
stored_program_cache, 786
stored_program_definition_cache, 786
super_read_only, 787
sync_binlog, 3136
sync_master_info, 3110
sync_relay_log, 3110
sync_relay_log_info, 3111
syseventlog.facility, 788
syseventlog.include_pid, 788
syseventlog.tag, 788
system_time_zone, 789
tablespace_definition_cache, 791
table_definition_cache, 789
table_encryption_privilege_check, 790
table_open_cache, 790
table_open_cache_instances, 791
temptable_max_mmap, 791
temptable_max_ram, 792
temptable_use_mmap, 792
thread_cache_size, 792
thread_handling, 793
thread_pool_algorithm, 793
thread_pool_high_priority_connection, 794
thread_pool_max_active_query_threads, 794
thread_pool_max_unused_threads, 795
thread_pool_prio_kickup_timer, 795
thread_pool_size, 795

thread_pool_stall_limit, 796
thread_stack, 796
timestamp, 797
time_zone, 797
tls_ciphersuites, 798
tls_version, 798
tmpdir, 799
tmp_table_size, 799
transaction_alloc_block_size, 800
transaction_allow_batching, 3686
transaction_isolation, 800
transaction_prealloc_size, 801
transaction_read_only, 802
transaction_write_set_extraction, 3137
unique_checks, 803
updatable_views_with_limit, 803
use_secondary_engine, 804
validate_password.check_user_name, 1224
validate_password.dictionary_file, 1224
validate_password.length, 1225
validate_password.mixed_case_count, 1225
validate_password.number_count, 1226
validate_password.policy, 1226
validate_password.special_char_count, 1227
validate_password_check_user_name, 1228
validate_password_dictionary_file, 1228
validate_password_length, 1229
validate_password_mixed_case_count, 1229
validate_password_number_count, 1229
validate_password_policy, 1230
validate_password_special_char_count, 1230
validate_user_plugins, 804
version, 805
version_comment, 805
version_compile_machine, 805
version_compile_os, 805
version_compile_zlib, 805
version_tokens_session, 977
version_tokens_session_number, 978
wait_timeout, 806
warning_count, 806
windowing_use_high_precision, 806
およびレプリケーション, 3234
非永続性, 830
必要な権限, 809
プロファイリング, 756
事前クエリーオプション
 mysqslap, 481
事前フィルタリング
 パフォーマンススキーマ, 4250
実験システム変数, 206
実行
 ANSI モード, 66
 クエリー, 266
 バッチモード, 286
 複数サーバー, 1009
実行オプション
 mysql, 372
 ndb_mgm, 3742
実行スレッド (NDB Cluster), 3628

実体化

- 共通テーブル式, 1488, 1571
- サブクエリー, 1483
- 参照の表示, 1488, 1571
- 導出テーブル, 1488, 1571

質問数, 396

- 質問ステータス変数, 848
- 自動インクリメント, 2659, 2664, 5370
- 自動インクリメントロック, 5370

自動コミット, 5370

自動コミットモード, 2708

自動修復オプション

- mysqlcheck, 408

自動ラップされた JSON 値, 1814

シャープチェックポイント, 5369

シャットダウン, 5369

- サーバー, 229

集計関数, 2090

充填係数, 2666

修復

- テーブル, 404, 1424

修復オプション

- myisamchk, 499

- mysqlcheck, 412

重複キーエラー, 2665

重複の除去

- 準結合方式, 1482

主キー, 5369

- およびパーティショニングキー, 4088

- 削除, 2175

- 制約, 71

循環レプリケーション

- NDB Cluster 内, 3993, 4013, 4016

準結合, 1480

順序エミュレーション, 2001

準同期レプリケーション, 3204

- インストール, 3207

- 管理インタフェース, 3206

- 構成, 3207

- モニタリング, 3209

準備されたバックアップ, 5370

準備済みステートメント, 2441, 2443, 2443

準備の割当て解除, 2437

障害検出

- グループレプリケーション, 3251

条件, 2452

条件の処理

- INOUT パラメータ, 2476

- OUT パラメータ, 2476

照合, 1686

- binary, 1712, 1739

- NO PAD, 1714, 1726, 1785

- PAD SPACE, 1714, 1726, 1785

- Unicode, 1725

- _ai 接尾辞, 1692

- _as 接尾辞, 1692

- _bin 接尾辞, 1692, 1712

- _ci 接尾辞, 1692

- _ks 接尾辞, 1692

- _ss 接尾辞, 1692

- アジア系, 1735
- キリル文字, 1734
- 中央ヨーロッパ言語, 1733
- 中東, 1733
- 西ヨーロッパ言語, 1731
- バルト語, 1734
- 南ヨーロッパ, 1733
- 命名規則, 1692
- 文字列, 1743
- 照合順序
 - INFORMATION_SCHEMA, 1716
 - 追加, 1744
 - 変更, 1745
- 照合テーブル
 - データディクショナリテーブル, 895
- 乗算 (*), 1874
- 小数点, 1761
- 小数秒
 - およびレプリケーション, 3221
- 小数秒の精度, 1762, 1771
- 冗長行フォーマット, 5369
- 冗長な行形式, 2778
- 情報オプション
 - myisamchk, 498
- 情報関数, 1993
- 情報スキーマ
 - InnoDB テーブル, 2894
- ショートフォームオプション
 - mysqlbinlog, 533
- 初期オプション
 - ndbd, 3727
 - ndbmt, 3727
 - ndb_mgmd, 3737
- 初期化オプション
 - mysqld, 651
- シリアライズされたディクショナリ情報 (参照 [SDI](#))
- シリアライズディクショナリ情報 (SDI), 5369
- 親イベント
 - performance_schema, 4442
- 新機能, 8
 - C API, 26
 - configuration, 26
 - EXPLAIN ANALYZE, 28
 - InnoDB, 11
 - innodb_deadlock_detect, 11
 - JSON, 20
 - JSON スキーマ CHECK 制約, 30
 - JSON スキーマ検証, 26
 - logging, 25
 - ON DUPLICATE KEY UPDATE, 30
 - redo ログのアーカイブ, 27
 - security, 9
 - TABLE ステートメント, 30
 - time_zone, 27
 - VALUES ステートメント, 30
 - アップグレード, 8
 - アトミック DDL, 8
 - ウィンドウ関数, 25
 - オプティマイザ, 22
 - キャスト関数, 26

- 共通テーブル式, 25
- クエリーキャストの注入, 28
- クローンプラグイン, 27
- 正規表現, 25
- 接続管理, 26
- タイムゾーンのサポート, 29
- データ型, 22
- データディクショナリ, 8
- テーブルの暗号化, 10
- テーブルのエイリアスと DELETE, 25
- 内部一時テーブル, 25
- バックアップロック, 25
- ハッシュ結合, 27
- 複数値インデックス, 27
- プラグイン, 26
- 文字セット, 19
- ラテラル導出テーブル, 25
- リソース管理, 10
- レプリケーション, 25
- シングルユーザーオプション
 - ndb_waiter, 3831
- シングルユーザーモード (NDB Cluster), 3841, 3873
 - および ndb_restore, 3789
- 親テーブル, 5370
- シンボリックリンク, 1603, 1604
 - Windows, 1604
 - データベース, 1603
 - テーブル, 1603
- 垂直オプション
 - mysql, 379
 - mysqladmin, 403
- 推定
 - クエリーパフォーマンス, 1555
- 数学関数, 1875
- 数値, 1630
- 数値スケール, 1761
- 数値精度, 1761
- 数値データ型, 1762
 - 記憶域要件, 1827
- 数値リテラル
 - approximate-value, 1630, 2142
 - exact-value, 1630, 2142
- スーパーユーザー, 230
- スカラー
 - JSON, 1811
- スキーマ, 5370
 - 削除, 2281
 - 作成, 2189
 - 変更, 2156
- スクリプト, 334, 342
 - SQL, 362
- スクリプトファイル, 286
- スケラビリティ, 5371
- スケラブルコヒーレントインタフェース (NDB Cluster) (廃止), 3724
- スケール
 - 演算, 2142
 - 数値, 1761
- スケールアウト, 5371
- スケールアップ, 5371
- スタンドアロンオプション

- mysqld, 664
- ステータスオプション
 - MySQLInstallerConsole, 126
 - mysqlshow, 472
- ステータス変数, 834, 2588
 - Aborted_clients, 835
 - Aborted_connects, 835
 - Acl_cache_items_count, 835, 835
 - Audit_log_current_size, 1345
 - Audit_log_events, 1345
 - Audit_log_events_filtered, 1345
 - Audit_log_events_lost, 1345
 - Audit_log_events_written, 1345
 - Audit_log_event_max_drop_size, 1345
 - Audit_log_total_size, 1345
 - Audit_log_write_waits, 1345
 - Authentication_ldap_sasl_supported_methods, 835
 - Binlog_cache_disk_use, 835
 - Binlog_cache_use, 835
 - Binlog_stmt_cache_disk_use, 835
 - Binlog_stmt_cache_use, 836
 - Bytes_received, 836
 - Bytes_sent, 836
 - Caching_sha2_password_rsa_public_key, 836
 - Compression_algorithm, 837
 - Compression_level, 837
 - Connections, 837
 - Connection_control_delay_generated, 1220
 - Connection_errors_accept, 837
 - Connection_errors_internal, 837
 - Connection_errors_max_connections, 837
 - Connection_errors_peer_address, 837
 - Connection_errors_select, 837
 - Connection_errors_tcpwrap, 837
 - Created_tmp_disk_tables, 838
 - Created_tmp_files, 838
 - Created_tmp_tables, 838
 - Current_tls_ca, 838
 - Current_tls_capath, 838
 - Current_tls_cert, 838
 - Current_tls_cipher, 838
 - Current_tls_ciphersuites, 839
 - Current_tls_crl, 839
 - Current_tls_crlpath, 839
 - Current_tls_key, 839
 - Current_tls_version, 839
 - Delayed_errors, 839
 - Delayed_insert_threads, 839
 - Delayed_writes, 839
 - dragnet.Status, 839, 839
 - Error_log_buffered_bytes, 839
 - Error_log_buffered_events, 840
 - Error_log_expired_events, 840
 - Error_log_latest_write, 840
 - Firewall_access_denied, 1371
 - Firewall_access_granted, 1371
 - Firewall_access_suspicious, 1371
 - Firewall_cached_entries, 1371
 - Flush_commands, 840
 - group_replication_primary_member, 840

Handler_commit, 840
Handler_delete, 840
Handler_discover, 3689
Handler_external_lock, 840
Handler_mrr_init, 840
Handler_prepare, 840
Handler_read_first, 840
Handler_read_key, 841
Handler_read_last, 841
Handler_read_next, 841
Handler_read_prev, 841
Handler_read_rnd, 841
Handler_read_rnd_next, 841
Handler_rollback, 841
Handler_savepoint, 841
Handler_savepoint_rollback, 841
Handler_update, 841
Handler_write, 841
Innodb_buffer_pool_bytes_data, 842
Innodb_buffer_pool_bytes_dirty, 842
Innodb_buffer_pool_dump_status, 841
Innodb_buffer_pool_load_status, 841
Innodb_buffer_pool_pages_data, 842
Innodb_buffer_pool_pages_dirty, 842
Innodb_buffer_pool_pages_flushed, 842
Innodb_buffer_pool_pages_free, 842
Innodb_buffer_pool_pages_latched, 842
Innodb_buffer_pool_pages_misc, 842
Innodb_buffer_pool_pages_total, 842
Innodb_buffer_pool_reads, 843
Innodb_buffer_pool_read_ahead, 842
Innodb_buffer_pool_read_ahead_evicted, 842
Innodb_buffer_pool_read_ahead_rnd, 842
Innodb_buffer_pool_read_requests, 843
Innodb_buffer_pool_resize_status, 843
Innodb_buffer_pool_wait_free, 843
Innodb_buffer_pool_write_requests, 843
Innodb_data_fsyncs, 843
Innodb_data_pending_fsyncs, 843
Innodb_data_pending_reads, 843
Innodb_data_pending_writes, 843
Innodb_data_read, 843
Innodb_data_reads, 843
Innodb_data_writes, 843
Innodb_data_written, 843
Innodb_dblwr_pages_written, 843
Innodb_dblwr_writes, 844
Innodb_have_atomic_builtins, 844
Innodb_log_waits, 844
Innodb_log_writes, 844
Innodb_log_write_requests, 844
Innodb_num_open_files, 844
Innodb_os_log_fsyncs, 844
Innodb_os_log_pending_fsyncs, 844
Innodb_os_log_pending_writes, 844
Innodb_os_log_written, 844
Innodb_pages_created, 844
Innodb_pages_read, 844
Innodb_pages_written, 844
Innodb_page_size, 844

Innodb_redo_log_enabled, 844
Innodb_rows_deleted, 845
Innodb_rows_inserted, 845
Innodb_rows_read, 845
Innodb_rows_updated, 845
Innodb_row_lock_current_waits, 844
Innodb_row_lock_time, 845
Innodb_row_lock_time_avg, 845
Innodb_row_lock_time_max, 845
Innodb_row_lock_waits, 845
Innodb_system_rows_deleted, 845
Innodb_system_rows_inserted, 845
Innodb_system_rows_read, 845
Innodb_truncated_status_writes, 845
Innodb_undo_tablespaces_active, 845
Innodb_undo_tablespaces_explicit, 845
Innodb_undo_tablespaces_implicit, 845
Innodb_undo_tablespaces_total, 846
Key_blocks_not_flushed, 846
Key_blocks_unused, 846
Key_blocks_used, 846
Key_reads, 846
Key_read_requests, 846
Key_writes, 846
Key_write_requests, 846
Last_query_cost, 846
Last_query_partial_plans, 846
Locked_connects, 846
Max_execution_time_exceeded, 846
Max_execution_time_set, 847
Max_execution_time_set_failed, 847
Max_used_connections, 847
Max_used_connections_time, 847
mecab_charset, 847
NDB Cluster, 3689
NDB Cluster レプリケーションの競合検出, 4021
Ndb_api_adaptive_send_deferred_count, 3689
Ndb_api_adaptive_send_deferred_count_replica, 3689
Ndb_api_adaptive_send_deferred_count_session, 3689
Ndb_api_adaptive_send_deferred_count_slave, 3689
Ndb_api_adaptive_send_forced_count, 3689
Ndb_api_adaptive_send_forced_count_replica, 3689
Ndb_api_adaptive_send_forced_count_session, 3689
Ndb_api_adaptive_send_forced_count_slave, 3689
Ndb_api_adaptive_send_unforced_count, 3690
Ndb_api_adaptive_send_unforced_count_replica, 3690
Ndb_api_adaptive_send_unforced_count_session, 3690
Ndb_api_adaptive_send_unforced_count_slave, 3690
Ndb_api_bytes_received_count, 3692
Ndb_api_bytes_received_count_replica, 3691
Ndb_api_bytes_received_count_session, 3691
Ndb_api_bytes_received_count_slave, 3691
Ndb_api_bytes_sent_count, 3691
Ndb_api_bytes_sent_count_replica, 3690
Ndb_api_bytes_sent_count_session, 3690
Ndb_api_bytes_sent_count_slave, 3691
Ndb_api_event_bytes_count, 3692
Ndb_api_event_bytes_count_injector, 3692
Ndb_api_event_data_count, 3692
Ndb_api_event_data_count_injector, 3692

Ndb_api_event_nodata_count, 3692
Ndb_api_event_nodata_count_injector, 3692
Ndb_api_pk_op_count, 3693
Ndb_api_pk_op_count_replica, 3693
Ndb_api_pk_op_count_session, 3693
Ndb_api_pk_op_count_slave, 3693
Ndb_api_pruned_scan_count, 3694
Ndb_api_pruned_scan_count_replica, 3694
Ndb_api_pruned_scan_count_session, 3693
Ndb_api_pruned_scan_count_slave, 3694
Ndb_api_range_scan_count, 3695
Ndb_api_range_scan_count_replica, 3694
Ndb_api_range_scan_count_session, 3694
Ndb_api_range_scan_count_slave, 3695
Ndb_api_read_row_count, 3696
Ndb_api_read_row_count_replica, 3695
Ndb_api_read_row_count_session, 3695
Ndb_api_read_row_count_slave, 3695
Ndb_api_scan_batch_count, 3696
Ndb_api_scan_batch_count_replica, 3696
Ndb_api_scan_batch_count_session, 3696
Ndb_api_scan_batch_count_slave, 3696
Ndb_api_table_scan_count, 3697
Ndb_api_table_scan_count_replica, 3697
Ndb_api_table_scan_count_session, 3697
Ndb_api_table_scan_count_slave, 3697
Ndb_api_trans_abort_count, 3698
Ndb_api_trans_abort_count_replica, 3697
Ndb_api_trans_abort_count_session, 3697
Ndb_api_trans_abort_count_slave, 3698
Ndb_api_trans_close_count, 3699
Ndb_api_trans_close_count_replica, 3698
Ndb_api_trans_close_count_session, 3698
Ndb_api_trans_close_count_slave, 3698
Ndb_api_trans_commit_count, 3699
Ndb_api_trans_commit_count_replica, 3699
Ndb_api_trans_commit_count_session, 3699
Ndb_api_trans_commit_count_slave, 3699
Ndb_api_trans_local_read_row_count, 3700
Ndb_api_trans_local_read_row_count_replica, 3699
Ndb_api_trans_local_read_row_count_session, 3699
Ndb_api_trans_local_read_row_count_slave, 3700
Ndb_api_trans_start_count, 3701
Ndb_api_trans_start_count_replica, 3700
Ndb_api_trans_start_count_session, 3700
Ndb_api_trans_start_count_slave, 3701
Ndb_api_uk_op_count, 3702
Ndb_api_uk_op_count_replica, 3701
Ndb_api_uk_op_count_session, 3701
Ndb_api_uk_op_count_slave, 3701
Ndb_api_wait_exec_complete_count, 3702
Ndb_api_wait_exec_complete_count_replica, 3702
Ndb_api_wait_exec_complete_count_session, 3702
Ndb_api_wait_exec_complete_count_slave, 3702
Ndb_api_wait_meta_request_count, 3703
Ndb_api_wait_meta_request_count_replica, 3703
Ndb_api_wait_meta_request_count_session, 3702
Ndb_api_wait_meta_request_count_slave, 3703
Ndb_api_wait_nanos_count, 3704
Ndb_api_wait_nanos_count_replica, 3703

Ndb_api_wait_nanos_count_session, 3703
Ndb_api_wait_nanos_count_slave, 3704
Ndb_api_wait_scan_result_count, 3705
Ndb_api_wait_scan_result_count_replica, 3704
Ndb_api_wait_scan_result_count_session, 3704
Ndb_api_wait_scan_result_count_slave, 3704
Ndb_cluster_node_id, 3705
Ndb_config_from_host, 3705
Ndb_config_from_port, 3705
Ndb_config_generation, 3705
Ndb_conflict_fn_epoch, 3705
Ndb_conflict_fn_epoch2, 3706
Ndb_conflict_fn_epoch2_trans, 3706
Ndb_conflict_fn_epoch_trans, 3706
Ndb_conflict_fn_max, 3705
Ndb_conflict_fn_max_del_win, 3705
Ndb_conflict_fn_old, 3705
Ndb_conflict_last_conflict_epoch, 3706
Ndb_conflict_last_stable_epoch, 3706
Ndb_conflict_reflected_op_discard_count, 3706
Ndb_conflict_reflected_op_prepare_count, 3706
Ndb_conflict_refresh_op_count, 3706
Ndb_conflict_trans_conflict_commit_count, 3707
Ndb_conflict_trans_detect_iter_count, 3707
Ndb_conflict_trans_reject_count, 3707
Ndb_conflict_trans_row_conflict_count, 3706
Ndb_conflict_trans_row_reject_count, 3707
Ndb_epoch_delete_delete_count, 3707
Ndb_execute_count, 3707
Ndb_last_commit_epoch_server, 3707
Ndb_last_commit_epoch_session, 3707
Ndb_metadata_blacklist_size (OBSOLETE), 3707
Ndb_metadata_detected_count, 3707
Ndb_metadata_excluded_count, 3707
Ndb_metadata_synced_count, 3708
Ndb_number_of_data_nodes, 3708
Ndb_pruned_scan_count, 3708
Ndb_pushed_queries_defined, 3708
Ndb_pushed_queries_dropped, 3708
Ndb_pushed_queries_executed, 3708
Ndb_pushed_reads, 3708
Ndb_replica_max_replicated_epoch, 3708
Ndb_scan_count, 3708
Ndb_slave_max_replicated_epoch, 3709
Ndb_system_name, 3709
Ndb_trans_hint_count_session, 3709
Not_flushed_delayed_rows, 847
Ongoing_anonymous_gtid_violating_transaction_count, 847
Ongoing_anonymous_transaction_count, 847
Ongoing_automatic_gtid_violating_transaction_count, 847
Opened_files, 847
Opened_tables, 848
Opened_table_definitions, 848
Open_files, 847
Open_streams, 847
Open_tables, 847
Open_table_definitions, 847
Performance_schema_accounts_lost, 4436
Performance_schema_cond_classes_lost, 4436
Performance_schema_cond_instances_lost, 4436

Performance_schema_digest_lost, 4436
Performance_schema_file_classes_lost, 4436
Performance_schema_file_handles_lost, 4436
Performance_schema_file_instances_lost, 4436
Performance_schema_hosts_lost, 4436
Performance_schema_index_stat_lost, 4436
Performance_schema_locker_lost, 4436
Performance_schema_memory_classes_lost, 4436
Performance_schema_metadata_lock_lost, 4436
Performance_schema_mutex_classes_lost, 4436
Performance_schema_mutex_instances_lost, 4437
Performance_schema_nested_statement_lost, 4437
Performance_schema_prepared_statements_lost, 4437
Performance_schema_program_lost, 4437
Performance_schema_rwlock_classes_lost, 4437
Performance_schema_rwlock_instances_lost, 4437
Performance_schema_session_connect_attrs_longest_seen, 4437
Performance_schema_session_connect_attrs_lost, 4437
Performance_schema_socket_classes_lost, 4437
Performance_schema_socket_instances_lost, 4437
Performance_schema_stage_classes_lost, 4437
Performance_schema_statement_classes_lost, 4437
Performance_schema_table_handles_lost, 4438
Performance_schema_table_instances_lost, 4438
Performance_schema_table_lock_stat_lost, 4438
Performance_schema_thread_classes_lost, 4438
Performance_schema_thread_instances_lost, 4438
Performance_schema_users_lost, 4438
Prepared_stmt_count, 848
Rewriter_number_loaded_rules, 965
Rewriter_number_reloads, 966
Rewriter_number_rewritten_queries, 966
Rewriter_reload_error, 966
Rpl_semi_sync_master_clients, 848
Rpl_semi_sync_master_net_avg_wait_time, 848
Rpl_semi_sync_master_net_waits, 848
Rpl_semi_sync_master_net_wait_time, 848
Rpl_semi_sync_master_no_times, 848
Rpl_semi_sync_master_no_tx, 849
Rpl_semi_sync_master_status, 849
Rpl_semi_sync_master_timefunc_failures, 849
Rpl_semi_sync_master_tx_avg_wait_time, 849
Rpl_semi_sync_master_tx_waits, 849
Rpl_semi_sync_master_tx_wait_time, 849
Rpl_semi_sync_master_wait_pos_backtraverse, 849
Rpl_semi_sync_master_wait_sessions, 849
Rpl_semi_sync_master_yes_tx, 849
Rpl_semi_sync_slave_status, 850
Rsa_public_key, 850
Secondary_engine_execution_count, 850
Select_full_join, 850
Select_full_range_join, 850
Select_range, 850
Select_range_check, 850
Select_scan, 850
Slave_open_temp_tables, 850
Slave_rows_last_search_algorithm_used, 850
Slow_launch_threads, 850
Slow_queries, 851
Sort_merge_passes, 851

Sort_range, 851
Sort_rows, 851
Sort_scan, 851
Ssl_accepts, 851
Ssl_accept_renegotiates, 851
Ssl_callback_cache_hits, 851
Ssl_cipher, 851
Ssl_cipher_list, 851
Ssl_client_connects, 851
Ssl_connect_renegotiates, 851
Ssl_ctx_verify_depth, 851
Ssl_ctx_verify_mode, 851
Ssl_default_timeout, 851
Ssl_finished_accepts, 852
Ssl_finished_connects, 852
Ssl_server_not_after, 852
Ssl_server_not_before, 852
Ssl_sessions_reused, 852
Ssl_session_cache_hits, 852
Ssl_session_cache_misses, 852
Ssl_session_cache_mode, 852
Ssl_session_cache_overflows, 852
Ssl_session_cache_size, 852
Ssl_session_cache_timeouts, 852
Ssl_used_session_cache_entries, 852
Ssl_verify_depth, 852
Ssl_verify_mode, 852
Ssl_version, 853
Table_locks_immediate, 853
Table_locks_waited, 853
Table_open_cache_hits, 853
Table_open_cache_misses, 853
Table_open_cache_overflows, 853
Tc_log_max_pages_used, 853
Tc_log_page_siz, 853
Tc_log_page_waits, 853
Threads_cached, 853
Threads_connected, 853
Threads_created, 854
Threads_running, 854
Uptime_since_flush_status, 854
validate_password.dictionary_file_last_parsed, 1227
validate_password.dictionary_file_words_count, 1227
validate_password_dictionary_file_last_parsed, 1230
validate_password_dictionary_file_words_count, 1230
圧縮, 837
稼働時間, 854
クエリー, 848
質問, 848
ステートメント
CREATE USER, 1077
DROP USER, 1077
GRANT, 1077
REVOKE, 1077
複合, 2443
レプリカ, 2398
レプリケーションサーバー, 2433
レプリケーションソース, 2395
ステートメントインターセプタ, 5371
ステートメントサンプリング, 4378

- ステートメントの終了
 - Control+C, 363, 377, 2372
- ステートメントベースレプリケーション, 5371
 - 安全でないステートメント, 3152
 - デメリット, 3152
 - メリット, 3151
- ステミング, 5371
- ストアドオブジェクト, 4095, 5371
 - 孤立, 4117
- ストアドオブジェクト権限, 4116
- ストアド生成カラム, 5372
- ストアドファンクション, 4097
- ストアドプログラム, 2443, 4095, 5371
 - roles, 1085
 - 再解析, 1592
- ストアドプロシージャ, 4097
- ストアドルーチン, 4095, 4097, 5372
 - LAST_INSERT_ID(), 4099
 - およびレプリケーション, 3221
 - 制限事項, 4125
 - メタデータ, 4099
- ストップワード, 1937, 5372
- ストップワードリスト
 - ユーザー定義の, 1940
- ストライピング
 - 定義, 1602
- ストレージエンジン, 5372
 - ARCHIVE, 3001
 - InnoDB, 2626
 - NDB と InnoDB の違い, 3479
 - PERFORMANCE_SCHEMA, 4238
 - アプリケーション機能の要件, 3481
 - 可用性, 3479
 - サポートされるアプリケーション, 3480
 - 使用シナリオ, 3481
 - 選択, 2983
- ストレージノード - データノード, ndbd, ndbmtd を参照 (参照 データノード, ndbd, ndbmtd)
- ストレージノード - データノードを参照, ndbd (参照 データノード, ndbd)
- ストレージ領域
 - 最適化, 1514
- スナップショット, 5372
- スパースファイル, 5372
- スピン, 5372
- スピンロックポーリング, 2744
- スペース ID, 5372
- スレーブワーカーが処理待ちイベントを解放するのを待機中
 - スレッドの状態, 1624
- スレッド, 5372
 - モニタリング, 4163, 4408
- スレッドキャッシュ, 865
- スレッド数, 396
- スレッドテーブル
 - performance_schema, 4408
- スレッドのコマンド, 1614
 - Binlog Dump, 1614
 - Change user, 1614
 - Close stmt, 1614
 - Connect, 1614
 - Connect Out, 1614
 - Create DB, 1614

- Daemon, 1614
- Debug, 1614
- Delayed insert, 1614
- Drop DB, 1615
- Error, 1615
- Execute, 1615
- Fetch, 1615
- Field List, 1615
- Init DB, 1615
- Kill, 1615
- Long Data, 1615
- Ping, 1615
- Prepare, 1615
- Processlist, 1615
- Query, 1615
- Quit, 1615
- Refresh, 1615
- Register Slave, 1615
- Reset stmt, 1615
- Set option, 1615
- Shutdown, 1616
- Sleep, 1616
- Statistics, 1616
- Time, 1616
- スレッドの状態, 1612
 - After create, 1616
 - altering table, 1616
 - Analyzing, 1616
 - Changing master, 1624
 - Checking master version, 1622
 - checking permissions, 1616
 - Checking table, 1616
 - cleaning up, 1616
 - Clearing, 1625
 - closing tables, 1616
 - committing alter table to storage engine, 1617
 - Committing events to binlog, 1625
 - Connecting to master, 1622
 - copy to tmp table, 1616
 - Copying to group table, 1616
 - Copying to tmp table, 1617
 - Copying to tmp table on disk, 1617
 - Creating index, 1617
 - Creating sort index, 1617
 - creating table, 1617
 - Creating tmp table, 1617
 - deleting from main table, 1617
 - deleting from reference tables, 1617
 - discard_or_import_tablespace, 1617
 - end, 1617
 - executing, 1617
 - Execution of init_command, 1617
 - Finished reading one binlog; switching to next binlog, 1622
 - freeing items, 1618
 - FULLTEXT initialization, 1618
 - HEAP から ondisk への変換, 1616
 - init, 1618
 - Initialized, 1625
 - Killed, 1618
 - Killing slave, 1624

- logging slow query, 1618
- login, 1618
- Making temporary file (append) before replaying LOAD DATA INFILE, 1623
- Making temporary file (create) before replaying LOAD DATA INFILE, 1623
- manage keys, 1618
- NDB Cluster, 1624
- Opening master dump table, 1624
- Opening mysql.ndb_apply_status, 1625
- optimizing, 1618
- preparing for alter table, 1619
- preparing, 1618
- Processing events from schema table, 1625
- Processing events, 1625
- Purging old relay logs, 1618
- query end, 1618
- Queueing master event to the relay log, 1622
- Reading event from the relay log, 1624
- Reading master dump table data, 1624
- Rebuilding the index on master dump table, 1624
- Reconnecting after a failed binlog dump request, 1622
- Reconnecting after a failed master event read, 1622
- Registering slave on master, 1623
- Removing duplicates, 1619
- removing tmp table, 1619
- rename, 1619
- rename result table, 1619
- Reopen tables, 1619
- Repair by sorting, 1619
- Repair done, 1619
- Repair with keycache, 1619
- Requesting binlog dump, 1623
- Rolling back, 1619
- Saving state, 1619
- Searching rows for update, 1619
- Sending binlog event to slave, 1622
- setup, 1620
- Shutting down, 1625
- Slave has read all relay log; waiting for more updates, 1624
- Sorting for group, 1620
- Sorting for order, 1620
- Sorting index, 1620
- Sorting result, 1620
- statistics, 1620
- Syncing ndb table schema operation and binlog, 1625
- System lock, 1620
- update, 1620
- updating main table, 1620
- updating reference tables, 1621
- Updating, 1620
- User lock, 1621
- User sleep, 1621
- Waiting for allowed to take ndbcluster global schema lock, 1625
- Waiting for commit lock, 1621
- Waiting for event from ndbcluster, 1625
- Waiting for first event from ndbcluster, 1625
- Waiting for global read lock, 1621
- Waiting for master to send event, 1623
- Waiting for master update, 1623
- Waiting for ndbcluster binlog update to reach current position, 1625
- Waiting for ndbcluster global schema lock, 1625

- Waiting for ndbcluster to start, 1625
- Waiting for next activation, 1625
- Waiting for scheduler to stop, 1625
- Waiting for schema epoch, 1625
- Waiting for schema metadata lock, 1621
- Waiting for slave mutex on exit, 1623, 1624
- Waiting for stored function metadata lock, 1621
- Waiting for stored procedure metadata lock, 1621
- Waiting for table level lock, 1621
- Waiting for table metadata lock, 1621
- Waiting for the next event in relay log, 1624
- Waiting for the slave SQL thread to free enough relay log space, 1623
- Waiting for trigger metadata lock, 1621
- Waiting on cond, 1622
- Waiting on empty queue, 1625
- Waiting to finalize termination, 1622
- Waiting to reconnect after a failed binlog dump request, 1623
- Waiting to reconnect after a failed master event read, 1623
- Waiting until MASTER_DELAY seconds after master executed event, 1624
- Writing to net, 1622
- 一般的な, 1616
- イベントスケジューラ, 1625
- 回転のコミットを待機しています, 1623
- 起動, 1620
- クライアントからの受信, 1619
- クライアントに送信中, 1620
- コーディネータからのイベントの待機中, 1624
- 削除された機能, 39
- システムテーブルのロック, 1618
- システムテーブルを開く, 1618
- スレーブワーカーが処理待ちイベントを解放するのを待機中, 1624
- テーブルの待機中, 1621
- テーブルフラッシュの待機中, 1621
- テーブルを開く, 1618
- ハンドラコミットの待機中, 1621
- マスターはすべての binlog をスレーブに送信しました。更新を待機しています, 1622
- レプリケーション, 1622, 1623, 1624
- レプリケーションソース, 1622
- スレッドプールプラグイン
 - リソースグループ, 888
- スロークエリーログ, 937, 5373
- 既読現象, 5373
- 正確な値の数値リテラル, 1630, 2142
- 正規化, 5374
- 正規化された JSON 値, 1814
- 正規表現
 - JSON スキーマ, 2081
 - 新機能, 25
- 正規表現の構文, 1917
- 制限事項
 - events, 4125
 - InnoDB, 2981, 2981
 - ウィンドウ関数, 2124
 - サーバー側カーソル, 2451
 - シグナル, 2476
 - ストアルーチン, 4125
 - トリガー, 4125
 - ビュー, 4128
 - リソースグループ, 888, 888
 - レプリケーション, 3212

静止, 5375
整数, 1630
整数演算, 2142
生成されるカラム, 5374
 ALTER TABLE, 2183
 CREATE TABLE, 2257
 CREATE TRIGGER, 2276
 CREATE VIEW, 2280
 INFORMATION_SCHEMA.COLUMNS テーブル, 4140
 INSERT, 2301
 REPLACE, 2324
 SHOW COLUMNS ステートメント, 2552, 4140
 UPDATE, 2362
 セカンダリインデックス, 2260
 ビュー, 4113
静的権限, 1060
精度
 numeric, 1761
 演算, 2142
 小数秒, 1762, 1771
正当な名前, 1635
制約, 71, 5374
 performance_schema データベース, 4443
 XA トランザクション, 2393
 外部キー, 2248
 サブクエリー, 2357
 プラグブルな認証, 1113
 文字セット, 1740
セーブポイント, 5373
セカンダリインデックス, 5373
 InnoDB, 2665
セカンダリパスワード, 1104
セキュアな接続, 1129
 コマンドオプション, 314
セキュリティー
 InnoDB memcached インタフェース用, 2954
 攻撃者に対する, 1036
セキュリティシステム, 1043
セグメント, 5373
設計
 問題, 4613
セッション一時テーブルスペース, 5373
セッションカテゴリ, 1089
セッションステート
 変更トラッキング, 890
セッションステート情報, 769, 769, 770
セッショントラック gtid, 768
セッションビュー
 sys スキーマ, 4478
セッション変数
 およびレプリケーション, 3234
接続, 5374
 DNS SRV レコードの使用, 327
 parameters, 322
 SSH を使用したリモート, 1150
 URI のような接続文字列の使用, 322, 325
 キーと値のペアの使用, 322, 326
 検証, 1073
 サーバーに, 265, 319
 中止, 4594

- 接続圧縮
 - X プロトコル, 3415
 - クラシック MySQL プロトコル, 330
- 接続インタフェース
 - main, 865
 - 管理, 865
- 接続オプション
 - ndb_restore, 3795
- 接続が失われましたエラー, 4591
- 接続管理, 864
 - 新機能, 26
- 接続ステータス変数, 837
- 接続プール, 5374
- 接続文字列, 5374 (参照 [NDB Cluster](#))
- 切断
 - サーバーから, 265
- 設定
 - passwords, 1097
- セットアップ
 - インストール後, 220
- 接尾辞オプション
 - mysql_ssl_rsa_setup, 352
- 説明オプション
 - myisamchk, 500
- 選択
 - MySQL のバージョン, 84
 - データ型, 1830
 - データベース, 270
- 選択性, 5374
- 全文
 - ストップワードリスト, 1940
- 全文クエリー
 - 最適化, 1501
- 全文検索, 1927, 5373
- 関連サブクエリー, 2350
- 相互排他ロック, 5375
- 相対オプション
 - mysqladmin, 402
- 挿入, 5375
 - 同時, 1594, 1597
 - の速度, 1496
- 挿入可能なビュー
 - insertable, 4112
- 挿入バッファ, 5375
- 挿入バッファリング, 5375
 - 無効化, 2639
- 増分バックアップ, 5375
- 増分リカバリ, 1419
 - NDB Cluster レプリケーションの使用, 4012
- 双方向レプリケーション
 - NDB Cluster 内, 4013, 4016
- ソースとレプリカの同期
 - NDB Cluster レプリケーション, 4009
- ソース配布
 - インストール, 183
- ソート
 - データ, 275
 - テーブルの行, 275
 - 付与テーブル, 1075, 1076
- ソートオプション

- ndb_top, 3829
- ソートバッファ, 5375
- 属性降格
 - レプリケーション, 3216
- 属性昇格
 - レプリケーション, 3216
- 速度
 - 挿入, 1496
 - レプリケーションで増加, 3019
- ソケットシステム変数, 777
- その他の関数, 2128
- ダーティーページ, 2823, 5376
- ダーティー読み取り, 5376
- 大/小文字の区別
 - アカウント名, 1071
 - アクセス検査, 1069
 - 識別子, 1639
 - データベース名, 67
 - テーブル名, 67
 - 名前, 1639
- 大カッコ
 - 平方, 1762
- 待機, 5376
- 代入演算子
 - :=, 1868
 - =, 1868
- タイプオプション
 - ibd2sdi, 485
 - ndb_config, 3751
 - ndb_show_tables, 3822
- タイムアウト (レプリケーション), 3229
- タイムゾーン
 - アップグレード, 882
 - うるう秒, 884
 - およびレプリケーション, 3229
 - サポート, 880
- タイムゾーンテーブル, 353
 - システムテーブル, 899
- タイムゾーンのサポート
 - 新機能, 29
- タイムゾーンの問題, 4605
- 対話型オプション
 - ndb_mgmd, 3737
- ダウングレード, 262
 - NDB Cluster, 3516, 3871
- ダウンロード, 85
- タブオプション
 - mysqldump, 432
 - ndb_restore, 3808
- タプル, 5376
- 単一引用符 (\), 1628
- 単一トランザクションオプション
 - mysqldump, 437
 - mysqlpump, 461
- 単項マイナス (-), 1874
- ダンプ
 - データベースおよびテーブル, 447
 - データベースとテーブル, 414
- ダンプオプション
 - myisam_ftdump, 492

- ndb_index_stat, 3777
- ndb_redo_log_reader, 3787
- 端末モニター
 - 定義, 265
- チェック
 - テーブルのエラー, 1424
- チェックオプション
 - myisamchk, 498
 - mysqlcheck, 408
- チェックサム (NDB Cluster), 3716
- チェックサム, 5376
- チェックサムエラー, 180
- チェックポイント, 5376
- 遅延レプリケーション, 3209
- 置換オプション
 - mysqldump, 426
 - mysqlimport, 445
 - mysqlpump, 459
- 中規模の信頼, 5376
- 中国語、日本語、韓国語文字セット
 - よくある質問, 4563
- 中止されたクライアント, 4594
- 中止された接続, 4594
- 抽出
 - 日付, 276
- チュートリアル, 265
- チューニング, 1430
- 調整
 - InnoDB 圧縮テーブル, 2763
- 地理空間特性, 1793
- 地理的特性, 1793
- 追加
 - 新規アカウント権限, 1077
 - 新規ユーザー権限, 1077
 - 文字セット, 1741
- 通常のアカント
 - アカウントカテゴリ, 1087
- 通常のセッション
 - セッションカテゴリ, 1089
- ツール
 - mysqld_multi, 342
 - mysqld_safe, 334
 - コマンド行, 123, 362
 - のリスト, 79
- 低位境界値, 5379
- ディクショナリオブジェクトキャッシュ, 2619, 5378
- 停止
 - サーバー, 232
- 定数テーブル, 1434
- ディスク I/O, 1530
- ディスクオプション
 - ndb_select_all, 3816
- ディスク障害
 - InnoDB, 2939
- ディスク上のデータ (NDB Cluster)
 - INFORMATION_SCHEMA.FILES テーブル, 4147
 - 「ディスクデータ」テーブル (NDB Cluster) (参照 [NDB Cluster ディスクデータ](#))
- ディスクバウンド, 5378
- ディスクパフォーマンス, 1601
- ディスクベース, 5378

低速シャットダウン, 5380
低優先度オプション
 mysqlexport, 444
ディレクトリ構造
 デフォルト, 97
データ
 インポート, 389, 439
 サイズ, 1514
 テーブルへのロード, 272
 取り出し, 273
データ暗号化, 2805
データウェアハウス, 5379
データ型, 1761
 BIGINT, 1764
 BINARY, 1783, 1786
 BIT, 1762
 BLOB, 1783, 1787
 BOOL, 1763, 1830
 BOOLEAN, 1763, 1830
 CHAR, 1781, 1782
 CHAR VARYING, 1783
 CHARACTER, 1782
 CHARACTER VARYING, 1783
 DATE, 1771, 1772
 DATETIME, 1771, 1772
 DEC, 1764
 DECIMAL, 1764, 2142
 DOUBLE, 1765
 DOUBLE PRECISION, 1765
 ENUM, 1784, 1788
 FIXED, 1764
 FLOAT, 1764, 1765, 1765
 GEOMETRY, 1794
 GEOMETRYCOLLECTION, 1794
 INT, 1764
 INTEGER, 1764
 LINESTRING, 1794
 LONG, 1787
 LONGBLOB, 1784
 LONGTEXT, 1784
 MEDIUMBLOB, 1783
 MEDIUMINT, 1763
 MEDIUMTEXT, 1783
 MULTILINESTRING, 1794
 MULTIPOINT, 1794
 MULTIPOLYGON, 1794
 NATIONAL CHAR, 1782
 NATIONAL VARCHAR, 1783
 NCHAR, 1782
 numeric, 1762
 NUMERIC, 1764
 NVARCHAR, 1783
 POINT, 1794
 POLYGON, 1794
 REAL, 1765
 SET, 1784, 1791
 SMALLINT, 1763
 string, 1781
 TEXT, 1783, 1787
 TIME, 1772, 1775

TIMESTAMP, 1771, 1772
TINYBLOB, 1783
TINYINT, 1763
TINYTEXT, 1783
VARBINARY, 1783, 1786
VARCHAR, 1781, 1783
VARCHARACTER, 1783
YEAR, 1772, 1775
新機能, 22
非推奨となった機能, 36, 36, 36
日付と時刻, 1769
データが無効です
 constraint, 72
データディクショナリ, 2617, 5379
 INFORMATION_SCHEMA 統合, 2620
 schema, 2617
 運用上の影響, 2622
 新機能, 8
 制限事項, 2624
 ディクショナリオブジェクトキャッシュ, 2619
 トランザクション記憶域, 2619
 メタデータファイルの削除, 2618
 利点, 2617
データディクショナリテーブル
 character_sets テーブル, 895
 check_constraints テーブル, 895
 column_type_elements テーブル, 896
 dd_properties テーブル, 896
 foreign_keys テーブル, 896
 foreign_key_column_usage テーブル, 896
 index_column_usage テーブル, 896
 index_partitions テーブル, 896
 index_stats テーブル, 896
 innodb_ddl_log テーブル, 896
 parameter_type_elements テーブル, 896
 resource_groups テーブル, 896
 schemata テーブル, 896
 st_spatial_reference_systems テーブル, 896
 tablespace_files テーブル, 896
 table_partitions テーブル, 896
 table_partition_values テーブル, 896
 table_stats テーブル, 896
 view_routine_usage テーブル, 896
 view_table_usage テーブル, 896
 イベントテーブル, 896
 インデックステーブル, 896
 カタログテーブル, 895
 カラムテーブル, 896
 照合テーブル, 895
 テーブルスペーステーブル, 896
 テーブルテーブル, 896
 トリガータブル, 896
 パラメータテーブル, 896
 ルーチンテーブル, 896
データディレクトリ, 5379
 mysql_upgrade_info ファイル, 237, 355
データノード (NDB Cluster), 3725, 3733
 定義済, 3450
データノード
 メモリー割当て, 3625

- データのクローニング, 979
- データファイル, 5379
- データベース, 5379
 - コピー, 261
 - 削除, 2281
 - 作成, 269, 2189, 2189
 - 使用, 269
 - 情報, 285
 - シンボリックリンク, 1603
 - 選択, 270, 2615
 - ダンプ, 414, 447
 - 定義, 4
 - 名前, 1635
 - バックアップ, 1405
 - 表示, 465
 - 複製, 3019
 - 変更, 2156
- データベースオブジェクト
 - メタデータ, 1691
- データベースオプション
 - mysql, 370
 - mysqlbinlog, 527
 - mysqlcheck, 409
 - mysqldump, 433
 - mysqlpump, 455
 - ndb_blob_tool, 3744
 - ndb_desc, 3759
 - ndb_index_stat, 3777
 - ndb_move_data, 3780
 - ndb_show_tables, 3822
- データベース情報
 - 取得, 2547
- データベースのコピー, 261
- データベース名
 - 大/小文字の区別, 67, 1639
- データベースメタデータ, 4132
- データマスキングプラグイン
 - アンインストール, 1373
 - インストール, 1373
- テーブル, 5377
 - BLACKHOLE, 3002
 - const, 1543
 - CSV, 2999
 - EXAMPLE, 3014
 - FEDERATED, 3009
 - HEAP, 2995
 - InnoDB, 2626
 - MEMORY, 2995
 - MERGE, 3004
 - MyISAM, 2987
 - system, 1543
 - TEMPORARY, 2244
 - 圧縮フォーマット, 2993
 - インポート, 2646
 - エラーチェック, 1424
 - 多すぎる, 1518
 - オープン, 1516
 - カラムの選択, 274
 - 行のカウント, 281
 - 行のサイズ, 1827

- 行の削除, 4610
- 行の選択, 273
- 行のソート, 275
- クローニング, 2244
- コピー, 2245
- 再作成, 259
- 最大サイズ, 1521
- 最適化, 1426
- 削除, 2284
- 作成, 270
- 修復, 259, 404, 1424
- 情報, 285, 501
- シンボリックリンク, 1603
- ステータスの表示, 2589
- ダンプ, 414, 447
- 断片化, 2532
- チェック, 498
- 定数, 1434
- データの取り出し, 273
- データのロード, 272
- デフラグ, 1427, 2532, 2992
- 動的, 2992
- 閉じる, 1516
- 名前, 1635
- パーティション化, 3004
- パフォーマンスの向上, 1514
- 表示, 465
- 開く, 1516
- 複数, 283
- 変更, 2165, 2175, 4612
- 保守, 404
- 保守スケジュール, 1427
- マージ, 3004
- テーブルオプション
 - mysql, 378
 - mysqlcheck, 413
 - mysqldump, 434
 - ndb_desc, 3760
- テーブルがいっぱいのエラーです (NDB Cluster), 3580
- テーブル型
 - 選択, 2983
- テーブルが満杯です, 679, 4595
- テーブルキャッシュ, 1516
- テーブルごとのカラム数
 - maximum, 1523
- テーブル情報
 - myisamchk, 501
- テーブル数
 - フラッシュ, 396
- テーブルスキャン, 2733
- テーブルスペース, 2675, 5377
- テーブルスペーステーブル
 - データディクショナリテーブル, 896
- テーブルスペースの暗号化
 - 監視, 2811
- テーブルタイプ, 5378
- テーブル値コンストラクタ
 - TABLE, 2358
 - VALUES ステートメント, 2364
- テーブルテーブル

- データディクショナリテーブル, 896
- テーブルの暗号化
 - 新機能, 10
- テーブルのエイリアス, 2327
- テーブルのエイリアスと DELETE
 - 新機能, 25
- テーブルの完全スキャン, 5377
- テーブルのクローニング, 2244
- テーブルのコピー, 2245
- テーブルのサイズ, 1521
- テーブルの待機中
 - スレッドの状態, 1621
- テーブルフラッシュの待機中
 - スレッドの状態, 1621
- テーブルプルアウト
 - 準結合方式, 1482
- テーブル名
 - 大/小文字の区別, 67, 1639
- テーブルレベルロック, 1593
- テーブルロック, 5378
- テーブルを再オープンできません
 - エラーメッセージ, 4612
- テーブルを開く
 - スレッドの状態, 1618
- デーモンオプション
 - ndb_mgmd, 3737
- 適応型ハッシュインデックス, 2641, 5380
- 適応型フラッシュ, 5380
- テキストオプション
 - ndb_top, 3829
- テキストコレクション, 5377
- テキストファイル
 - インポート, 389, 439, 2307
- テキストファイルから SQL ステートメントを実行する, 389
- テキストファイルからの SQL ステートメントの実行, 286
- 適用, 5380
- テスト
 - インストール, 221
 - インストール後, 220
 - サーバーへの接続, 1073
- テストオプション
 - myisampack, 510
- デッドロック, 1593, 2385, 2713, 2717, 2718, 2718, 2874, 4439, 5378
- デッドロック検出, 5379
- デッドロック対象, 5378
- デバッグ
 - MySQL, 1016
 - クライアント, 1022
 - サーバー, 1016
- デバッグオプション
 - comp_err, 346
 - ibd2sdi, 484
 - myisamchk, 495
 - myisampack, 509
 - mysql, 370
 - mysqladmin, 399
 - mysqlbinlog, 528
 - mysqlcheck, 409
 - mysqld, 647
 - mysqldump, 427

- mysqldumpslow, 545
- mysqlimport, 442
- mysqlpump, 455
- mysqlshow, 468
- mysqlslap, 478
- mysql_config_editor, 517
- mysql_upgrade, 358
- my_print_defaults, 548
- デバッグサポート, 191
- デバッグレベルオプション
 - ndb_setup.py, 3820
- デフォルトアカウント, 230
- デフォルトオプション, 302
- デフォルト値, 1824, 2223, 2301
 - BLOB および TEXT カラム, 1787
 - 暗黙の, 1824
 - 明示的な, 1824
- デフォルトのインストール場所, 97
- デフォルトのプロキシユーザー, 1118
- デフォルトのホスト名, 319
- デフォルトロール, 2516
 - ALTER USER, 2484
 - CREATE USER ステートメント, 2496
- デュアルパスワード, 1104
- 転置インデックス, 5380
- 電話帳の照合、ドイツ語, 1732, 1732
- ドイツ語辞書照合, 1732, 1732
- ドイツ語の電話帳照合, 1732, 1732
- 透過的データ暗号化, 2805
- 透過的ページ圧縮, 5382
- 統計, 5382
- 同時実行オプション
 - mysqlslap, 478
- 同時実行性
 - コミット, 2836
 - スレッド, 2888
 - チケット, 2838
- 同時挿入, 1594, 1597
- 導出条件プッシュダウン, 1490
- 導出テーブル, 2350
 - 更新可能なビュー, 4112
 - 最適化, 1479, 1488
 - 実体化防止, 1489
 - ラテラル, 1547, 2353
- 動的 SQL, 5381
- 動的カーソル, 5381
- 動的行フォーマット, 2780, 5382
- 動的権限, 1060
- 動的ステートメント, 5381
- 動的テーブルの特徴, 2992
- ドキュメント ID, 5381
- ドキュメント
 - 韓国語, 4563
 - 中国語, 4563
 - 日本語, 4563
- ドキュメント作成者
 - のリスト, 77
- ドキュメントストア, 3373
 - としての MySQL, 3373
- 匿名ユーザー, 1073, 1076

閉じる

- テーブル, 1516
- 特権システム, 1043
- トラストストア, 5380
- トラブルシューティング, 4585, 5380
 - ALTER TABLE での問題, 4612
 - InnoDB テーブルのデフラグ, 2786
 - InnoDB デッドロック, 2717, 2718
 - InnoDB のエラー, 2980
 - InnoDB のリカバリに関する問題, 2977
 - MySQL Enterprise Monitor を使用した, 4529
 - MySQL Server のコンパイル, 218
 - MySQL パフォーマンススキーマによる, 4439
 - 起動の問題, 226
 - 接続の問題, 1124
 - レプリケーション, 3238
- トランザクション ID, 5381
- トランザクション, 5381
 - およびレプリケーション, 3231
 - サポート, 2626
- トランザクションアクセスモード, 2387
- トランザクションオプション
 - ndb_delete_all, 3754
- トランザクションセーフテーブル, 2626
- トランザクションの一貫性保証
 - 理解, 3273
- トランザクションの状態
 - 変更トラッキング, 890
- トランザクション分離レベル, 2387
 - NDB Cluster, 3485
 - READ COMMITTED, 2706
 - READ UNCOMMITTED, 2707
 - REPEATABLE READ, 2705
 - SERIALIZABLE, 2707
- トランスポートブルテーブルスペース, 2646, 5381
- トリガー, 2275, 2286, 2593, 4095, 4099
 - LAST_INSERT_ID(), 4099
 - およびレプリケーション, 3221, 3233
 - 制限事項, 4125
 - メタデータ, 4104
- トリガーオプション
 - mysqldump, 434
 - mysqlpump, 462
- トリガーテーブル
 - データディクショナリテーブル, 896
- トリガーの表示, 2593
- 取消し
 - privileges, 2515
- 取り出し
 - テーブルのデータ, 273
- トレースファイル (NDB Cluster), 3731
- トレースファイル
 - ndbmt, 3733
- ドロップ, 5381
- 内部一時テーブル
 - 新機能, 25
- 内部関数, 2127
- 内部記憶域形式
 - ジオメトリ値, 1802
- 内部ロック, 1593

ナチュラルキー, 5382
夏時間, 883, 1512, 1898
名前, 1635, 3656
 大/小文字の区別, 1639
 変数, 1673
名前付きウィンドウ
 ウィンドウ関数, 2123
名前付きタイムゾーンのサポート
 不明または不正なタイムゾーン, 881
名前付きパイプ, 129, 134
名前のないビュー, 2350
二重引用符 ("), 1628, 2070
二重書き込みバッファ, 843, 2843
二重書き込みバッファアー, 2784, 5382
日時データ型, 1769
日付値
 問題点, 1774
日本語、韓国語、中国語文字セット
 よくある質問, 4563
日本語文字セット
 変換, 4563
入力
 クエリー, 266
認証
 InnoDB memcached インタフェース用, 2954
認証プラグイン
 authentication_ldap_sasl, 1176
 authentication_ldap_sasl_client, 1176
 authentication_ldap_simple, 1176
 authentication_pam, 1162
 authentication_windows, 1171
 authentication_windows_client, 1171
 auth_socket, 1196
 auth_test_plugin, 1198
 caching_sha2_password, 1152
 mysql_clear_password, 1161
 mysql_clear_plugin, 1176
 mysql_native_password, 1151
 mysql_no_login, 1194
 sha256_password, 1157
 test_plugin_server, 1198
 クライアント/サーバーの互換性, 1112
 クライアント/サーバープロトコル, 1113
ネイティブ C API, 5383
ネイティブのバックアップとリストア
 バックアップ識別子, 3886
ネーミング
 MySQL のリリース, 84
ネクストキーロック, 2701, 5383
 InnoDB, 2717
ネストされたクエリー, 2344
ネットマスク表記法
 アカウント名, 1071
ネットワークネームスペース, 875
ネットワークポート
 NDB Cluster, 3986
年齢
 計算, 276
ノードグループ (NDB Cluster), 3452
ノードログ (NDB Cluster), 3858

- ページ, 5384
- ページ構成, 2745
- ページスケジューリング, 2745
- ページスレッド, 5385
- ページ遅延, 5385
- ページバッファリング, 5385
- バージョン
 - 最新, 85
 - 選択, 84
- バージョンオプション
 - comp_err, 347
 - ibd2sdi, 484
 - innochecksum, 487
 - myisamchk, 496
 - myisampack, 510
 - mysql, 379
 - mysqladmin, 403
 - mysqlbinlog, 536
 - mysqlcheck, 413
 - mysqld, 669
 - mysqldump, 428
 - mysqld_multi, 344
 - mysqlimport, 447
 - mysqlpump, 463
 - mysqlshow, 472
 - mysqlslap, 483
 - mysql_config, 547
 - mysql_config_editor, 518
 - mysql_ssl_rsa_setup, 353
 - my_print_defaults, 548
 - ndbxfm, 3834
 - ndb_config, 3751
 - ndb_perror, 3783
 - perror, 549
- バージョントークン UDF
 - version_tokens_delete(), 975
 - version_tokens_edit(), 975
 - version_tokens_lock_exclusive(), 976
 - version_tokens_lock_shared(), 976
 - version_tokens_set(), 976
 - version_tokens_show(), 976
 - version_tokens_unlock(), 976
- バージョントークン, 968
- バージョントークンプラグイン
 - reference, 975
 - アンインストール, 969
 - インストール, 968
 - 使用, 969
 - 要素, 968
- バージョンビュー
 - sys スキーマ, 4488
- パーティショニング
 - COLUMNS, 4040
 - および FULLTEXT インデックス, 4085
 - および SQL モード, 3228, 4082
 - および一時テーブル, 4085, 4088
 - および外部キー, 4084
 - およびサブクエリー, 4085
 - および日付, 4033
 - およびレプリケーション, 3228

- 概念, 4030
- キーによる, 4049
- 最適化, 4074
- サブパーティショニング, 4087
- ストレージエンジン (制限), 4091
- 制限, 4082
- 線形ハッシュによる, 4048
- タイプ, 4032
- パーティショニングキーのデータ型, 4085
- パーティショニング式で許可されない演算子, 4082
- パーティショニング式で許可される関数, 4092
- パーティショニング式でサポートされる演算子, 4082
- パーティションの最大数, 4084
- ハッシュによる, 4047
- 範囲による, 4034
- リストによる, 4038
- リソース, 4030
- 利点, 4032
- リニアキーによる, 4051
- パーティショニングキーおよび一意キー, 4088
- パーティショニングキーおよび主キー, 4088
- パーティショニング情報ステートメント, 4072
- パーティション (NDB Cluster), 3452
- パーティション
 - 管理, 4057
 - 切り捨て, 4057
 - 最適化, 4071
 - 修復, 4071
 - チェック, 4071
 - 追加および削除, 4057
 - 分割およびマージ, 4057
 - 分析, 4071
 - 変更, 4057
- パーティション管理, 4057
- パーティションに関する情報の取得, 4072
- パーティションプルーニング, 4074
- ハートビート, 5383
- 排他ロック, 2701, 5385
- バイナリ照合, 1712
- バイナリデータで引用符を使用, 1629
- バイナリ配布
 - インストール, 98
- バイナリロギング
 - ALTER USER, 2490
 - CREATE USER, 2501
 - NDB Cluster, 3489
- バイナリログ, 922, 5383
 - イベントグループ, 3149
 - 不可視のカラム, 2266
- バイナリログの暗号化, 3179
- バウンス, 5383
- バグ
 - NDB Cluster
 - レポート, 3761
 - 既知, 4613
 - レポート, 2, 62
- バグデータベース, 62
- パス名区切り文字
 - Windows, 305
- パスワード

- InnoDB memcached インタフェース用, 2954
- 管理者ガイドライン, 1035
- 失くした, 4599
- セキュリティー, 1034
- ユーザーガイドライン, 1034
- ユーザーの, 1044
- リセット, 4599
- ログイン, 1035
- 忘れた, 4599
- パスワードオプション, 312
 - mysql, 375
 - mysqladmin, 401
 - mysqlbinlog, 530
 - mysqlcheck, 411
 - mysqldump, 423
 - mysqld_multi, 344
 - mysqlimport, 445
 - mysqlpump, 459
 - mysqlshow, 470
 - mysqlslap, 480
 - mysql_secure_installation, 349
 - mysql_upgrade, 360
 - ndb_top, 3828
- 「パスワードが現在のポリシー要件を満たしていません」
 - パスワードエラー, 1221
- パスワード管理, 1098
- パスワードの検証, 1221
- パスワードの設定, 2517
- パスワードポリシー, 1221
- 破損, 2977
 - InnoDB, 2939
- 破損ページ, 2784, 5385
- パターンマッチング, 279, 1917
- 発音
 - MySQL, 5
- バックアップ,トラブルシューティング
 - NDB Cluster 内, 3888
- バックアップ, 1405, 4530, 5384
 - InnoDB, 2938
 - mysqldump による, 1414
 - データベースとテーブル, 414
- バックアップオプション
 - myisamchk, 499
 - myisampack, 509
- バックアップからのリストア
 - NDB Cluster レプリケーション, 4007
- バックアップ識別子
 - ネイティブのバックアップとリストア, 3886
- バックアップの構成
 - NDB Cluster 内, 3887
- バックアップのリストア
 - NDB Cluster 内, 3788
- バックアップロック
 - 新機能, 25
- バックグラウンドスレッド, 2743
 - read, 2742
 - write, 2742
- バックスペース (\b)), 1628, 2070, 2312
- バックスラッシュ
 - エスケープ文字, 1627

- パッケージ
 - のリスト, 78
- ハッシュインデックス, 1504, 5383
- ハッシュ結合
 - 新機能, 27
- バッチ SQL ファイル, 362
- バッチオプション
 - mysql, 368
- バッチ更新 (NDB Cluster レプリケーション), 4004
- バッチモード, 286
- パッド属性
 - 照合, 1714, 1785, 1915
- バッファアー, 5384
- バッファアーサイズ, 1587, 2728
- バッファアープール, 1587, 2728, 5384
 - と圧縮テーブル, 2769
- バッファアープールインスタンス, 5384
- バッファサイズ
 - client, 4521
- バッファプール, 2733, 2733, 2734, 2735, 2737
 - 監視, 2635, 2732, 2739
- バッファリングの変更
 - 無効化, 2639
- バディアーロケータ, 2895, 5384
- 幅
 - display, 1761
- パフォーマンス, 1430
 - 推定, 1555
 - ディスク I/O, 1601
 - ベンチマーク, 1611
- パフォーマンススキーマ, 2924, 4237, 5384
 - data_locks テーブル, 2897
 - data_lock_waits テーブル, 2897
 - イベントフィルタリング, 4250
 - スレッドプールテーブル, 4364
 - メモリーの使用, 4246
- パフォーマンススキーマ関数, 2124
- パフォーマンススキーマクエリー
 - 最適化, 1494
- パラメータテーブル
 - データディクショナリテーブル, 896
- 範囲外の処理, 1768
- 範囲結合タイプ
 - オブティマイザ, 1545
- 半一貫性読み取り, 5385
- ハンドラ, 2453
- ハンドラコミットの待機中
 - スレッドの状態, 1621
- 反復オプション
 - mysqlslap, 479
- 反復不可能読み取り, 5385
- ピア行
 - ウィンドウ関数, 2119
- 比較
 - アカウント名, 1071
 - アクセス検査, 1069
 - 末尾のスペース, 1714
- 比較演算子, 1860
- 比較の末尾の空白, 1714
- 非決定的関数

- 最適化, 1475
- レプリケーション, 1475
- ビジネスルール, 5385
- 非推奨となった機能, 35
- !, 36
- &&, 36
- compress, 37
- master-info-file, 37
- no-dd-upgrade, 37
- BINARY, 36
- CREATE TEMPORARY TABLE, 36
- ENGINE, 36
- FOUND_ROWS(), 36
- InnoDB memcached プラグイン, 37
- INSERT ... ON DUPLICATE KEY UPDATE, 37
- INTO, 36
- JSON_MERGE(), 36
- JSON_TABLE() 構文, 37
- KEY によるカラムインデックス接頭辞およびパーティション, 37
- max_length_for_sort_data, 37
- MYSQL_OPT_COMPRESS, 37
- MYSQL_PWD, 37
- mysql_upgrade, 37
- mysql_upgrade_info ファイル, 37
- PAD_CHAR_TO_FULL_LENGTH, 36
- relay_log_info_file, 37
- sha256_password, 35
- slave_compressed_protocol, 37
- SQL_CALC_FOUND_ROWS, 36
- UNION, 36
- utf8mb3, 35
- validate_password プラグイン, 36
- VALUES(), 37
- ||, 36
- データ型, 36, 36, 36
- 非正規化, 5386
- 日付
 - パーティショニングで使用される (例), 4036, 4047, 4052, 4075
 - パーティショニングで使用される, 4033
- 日付データ型
 - 記憶域要件, 1828
- 日付と時刻の関数, 1883
- 日付の計算, 276
- 日付リテラル, 1630
- ビット演算子, 1975
- ビット関数
 - 例, 291
- ビット操作
 - 16 進数リテラル, 1633
 - ビット値リテラル, 1635
- ビット値リテラル, 1634
 - ビット操作, 1635
- ビット値リテラルの概要, 1634
- 非同期 I/O, 2742
- 非同期 I/O, 5386
- 非同期レプリケーション (参照 [NDB Cluster レプリケーション](#))
- 等しくない (!=), 1862
- 等しくない (<>), 1862
- 非トランザクションテーブル, 4609
- 非ブロック化 I/O, 5386

- ビュー, 2277, 4095, 4111
 - privileges, 4129
 - roles, 1085
 - アルゴリズム, 4111
 - およびレプリケーション, 3236
 - 更新可能, 2277, 4112
 - 最適化, 1479, 1488
 - 実体化防止, 1489
 - 制限事項, 4128, 4129
 - メタデータ, 4116
 - 問題, 4129
- 表記規則, 2
- 表示
 - information
 - SHOW, 2547, 2550, 2592
 - SHOW ステートメント, 2568, 2570
 - カーディナリティ, 2569
 - 照合, 2569
 - データベース情報, 465, 465
 - テーブルステータス, 2589
- 表示サイズ, 1761
- 表示幅, 1761
- 標準 SQL
 - との違い, 69, 2514
 - に対する拡張機能, 67
 - への拡張, 66
- 標準からの読み取り
 - innochecksum, 490
- 標準互換性, 66
- 標準モニター, 2931, 2933, 2936
- 開いているテーブル数, 396
- 開く
 - テーブル, 1516
- 非ロック読み取り, 5386
- ヒント, 67
 - インデックス, 1579, 2327
 - オブティマイザ, 1565
 - 最適化, 1497
- ファイル
 - Not Found メッセージ, 4598
 - 一般クエリーログ, 921
 - エラーメッセージ, 1740
 - 権限, 4598
 - 修復, 499
 - スクリプト, 286
 - スロークエリーログ, 937
 - テキスト, 389, 439
 - バイナリログ, 922
 - ログ, 940
- ファイル形式, 5386
- ファイル内オプション
 - comp_err, 346
- ファイルを作成/書き込みできない, 4596
- ファジーチェックポイント, 5386
- ファントム, 5386
- ファントム行, 2716
- フィールド
 - 変更, 2173
- フィールドオプション
 - ndb_config, 3748

- フィルファクタ, 5386
- ブール検索, 1932
- ブール値オプション, 308
- ブールリテラル, 1635
- フォーラム, 61
- 負荷エミュレーション, 472
- 不可視インデックス, 1509, 2175, 2207, 2228
 - メタデータ, 1509
- 不可視のカラム, 2223, 2263
 - バイナリログ, 2266
 - メタデータ, 2265
- 複合インデックス, 5388
- 複合ステートメント, 2443
- 複合パーティショニング, 4051
- 複数サーバー, 1009
- 複数值インデックス, 2199
 - 新機能, 27
- 複数の mysqld, 342
- 複数のディスクを使用したデータの起動, 1604
- 複数バッファプール, 2733
- 副選択, 2344
- プッシュダウン結合 (NDB), 3674
- 物理, 5388
- 物理バックアップ, 5388
- 浮動小数点数, 1765
- 浮動小数点値
 - およびレプリケーション, 3219
- 部分インデックス, 5388
- 部分更新
 - レプリケーション, 3227
- 部分信頼, 5388
- 部分的な取消し, 1076, 1091
- 部分バックアップ, 5388
- 不明または不正なタイムゾーン
 - error, 881
- 付与
 - privileges, 2502
- 付与テーブル
 - columns_priv テーブル, 898, 1063
 - db table, 230, 897, 1063
 - default_roles テーブル, 898, 1063
 - global_grants テーブル, 897, 1061, 1063
 - password_history テーブル, 898, 1063
 - procs_priv テーブル, 898, 1063
 - proxies_priv, 1117
 - proxies_priv テーブル, 230, 898, 1063
 - role_edges テーブル, 898, 1063
 - tables_priv テーブル, 897, 1063
 - user table, 230, 897, 1062
 - 構造, 1062
 - ソート, 1075, 1076
- プライマリパスワード, 1104
- ブラインドクエリー拡張, 1937, 5387
- プラグブルな認証
 - 制約, 1113
- プラグブル認証
 - PAM, 1162
 - Windows, 1171
- プラグイン API, 945
- プラグイン

- security, 1151
- アクティブ化, 946
- アンインストール, 946, 2540
- インストール, 946, 2538
- クローン, 979
- サーバー, 945
- 新機能, 26
- プラグインアクティベーションオプション
 - FORCE, 949
 - FORCE_PLUS_PERMANENT, 949
 - OFF, 949
 - ON, 949
- プラグインアンインストール
 - データマスキング, 1373
 - バージョントークン, 969
 - リライタクエリーリライトプラグイン, 958
- プラグインインストール
 - audit_log, 1280
 - CONNECTION_CONTROL, 1216
 - CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS, 1216
 - ddl_rewriter, 966
 - keyring_aws, 1233
 - keyring_encrypted_file, 1233
 - keyring_file, 1233
 - keyring_hashicorp, 1233
 - keyring_oci, 1233
 - keyring_okv, 1233
 - keyring_udf, 1256
 - MySQL Enterprise Firewall プラグイン, 1349
 - MySQL Enterprise Thread Pool, 952
 - クローン, 980
 - データマスキング, 1373
 - バージョントークン, 969
 - リライタクエリーリライトプラグイン, 958
- プラグインサービス, 1000
 - locking_service, 1000
 - mysql_keyring, 1005
- プラグインテーブル
 - システムテーブル, 898
- プラグインのアクティブ化, 946
- プラグインのアンインストール, 946, 2540
- プラグインのインストール, 946, 2538
- フラグメントレプリカ (NDB Cluster), 3452
- フラッシュ, 2735, 5387
- フラッシュオプション
 - mysqld, 650
- フラッシュテーブル数, 396
- フラッシュリスト, 5387
- プラットフォーム
 - サポート対象, 83
- プリアードステートメント, 2437, 5387
 - 再準備, 1592
- 古いシステム変数, 744
- フルテーブルスキャン
 - 回避, 1479
- フロー制御関数, 1869
- プロキシユーザー, 1115
 - LDAP 認証, 1184
 - PAM 認証, 1168
 - PROXY 権限, 1117

- Windows 認証, 1175
- サーバーユーザーマッピング, 1120
- システム変数, 1121
- デフォルトのプロキシユーザー, 1118
- 匿名ユーザーとの競合, 1119
- プロキシユーザーマッピング
 - LDAP 認証, 1186
- プログラム
 - stored, 4095
 - 管理, 298
 - クライアント, 297
 - ストアド, 2443
 - ユーティリティ, 298
- プログラムオプション (NDB Cluster), 3834
- プログラム開発ユーティリティ, 298
- プログラム変数
 - 設定, 309
- プログラム変数の設定, 309
- プロシージャ
 - stored, 4097
- プロセス, 5387
 - display, 2574
 - ndbinfo テーブル, 3957
 - 監視, 1612
- プロセス管理 (NDB Cluster), 3724
- ブロックされたホストのブロック解除, 871
- ブロックされるホスト
 - ブロック解除, 871
- プロファイリングシステム変数, 756
- プロンプト
 - 意味, 268
- 分散特権 (NDB Cluster), 3899
- 分析オプション
 - mysamchk, 500
 - mysqlcheck, 408
- 分離レベル, 2705, 5387
- 並列処理オプション
 - ndb_restore, 3802
- 並列性, 2626, 5389
- ページ, 5388
- ページ圧縮, 2774
- ページオプション
 - innochecksum, 488
- ページクリーナー, 5388
- ページサイズ
 - InnoDB, 2666
- ベシミスティック, 5388
- ヘッダーオプション
 - ndb_select_all, 3816
- ヘッダーファイル
 - keyword_list.h, 4157
- ヘルプオプション
 - comp_err, 346
 - ibd2sdi, 483
 - innochecksum, 486
 - mysamchk, 495
 - mysampack, 509
 - mysam_ftdump, 491
 - mysql, 367
 - mysqladmin, 399

- mysqlbinlog, 526
- mysqlcheck, 408
- mysqld, 645
- mysqldump, 427
- mysqldumpslow, 545
- mysqld_multi, 343
- mysqld_safe, 336
- mysqlexport, 442
- MySQLInstallerConsole, 124
- mysqlpump, 453
- mysqlshow, 468
- mysqlslap, 477
- mysql_config_editor, 517
- mysql_secure_installation, 348
- mysql_ssl_rsa_setup, 352
- mysql_upgrade, 358
- my_print_defaults, 547
- ndbxfm, 3833
- ndb_perror, 3783
- ndb_setup.py, 3820
- ndb_top, 3827
- peror, 549
- ヘルプテーブル
 - システムテーブル, 898
- 変更
 - カラム, 2173
 - スキーマ, 2156
 - ソケットの場所, 341, 4605
 - データベース, 2156
 - テーブル, 2165, 2175, 4612
 - フィールド, 2173
- 変更バッファ, 2638
 - 監視, 2640
- 変更バッファアー, 5389
- 変更バッファリング, 5389
- 変数
 - server, 2594
 - status, 2588
 - system, 806, 2594
 - およびレプリケーション, 3234
 - 環境, 299
 - システム, 669
 - ステータス, 834
 - ユーザー定義, 1673
- 変数オプション
 - mysql_config, 547
- ベトナム語, 4563
- ベンチマーク, 1611
- ベースカラム, 5389
- ベータ, 5389
- ポイントインタイムリカバリ, 1419, 5390
- 方法
 - ロック, 1593
- ポート, 226, 319, 550, 1014, 1032, 1124, 1150, 4293, 4588
- ホールパンチング, 5390
- 保守
 - テーブル, 404, 1427
 - ログファイル, 940
- ホスト, 5389
- ホストオプション, 312

- mysql, 372
- mysqladmin, 401
- mysqlbinlog, 529
- mysqlcheck, 410
- mysqldump, 422
- mysqlimport, 444
- mysqlpump, 457
- mysqlshow, 470
- mysqslap, 479
- mysql_secure_installation, 349
- mysql_upgrade, 359
- ndb_config, 3749
- ndb_top, 3827
- ホストキャッシュ, 868
- ホスト名, 319
 - アカウント名, 1070
 - 最大長, 26
 - デフォルト, 319
 - デフォルトアカウント, 230
 - ロール名, 1072
- ホスト名解決, 868
- ホスト名キャッシュ, 868
- 保存データの暗号化, 2805
- ホット, 5390
- ホットバックアップ, 5390
- ボトルネック, 5390
- 翻訳者
 - のリスト, 77
- マージ, 5391
 - 共通テーブル式, 1488
 - 参照の表示, 1488
 - 導出テーブル, 1488
- マイナス
 - 単項 (-), 1874
- 負の値, 1630
- マスタースレッド, 5390
- マスターはすべての binlog をスレーブに送信しました。更新を待機しています
 - スレッドの状態, 1622
- マッチング
 - パターン, 279
- 末尾のスペース
 - CHAR, 1782, 1784
 - ENUM, 1789
 - SET, 1791
 - VARCHAR, 1783, 1784
 - 比較, 1784
- マニュアル
 - オンラインでの場所, 2
 - 構文規則, 2
 - 表記規則, 2
- マニュアルのオンラインでの場所, 2
- マルチ LineStringFromText()
 - 削除された機能, 40
- マルチ LineStringFromWKB()
 - 削除された機能, 40
- マルチカラムインデックス, 1501
- マルチコア, 5390
- マルチソースレプリケーション, 3059
 - GTID ソースの追加, 3061
 - NDB Cluster 内, 4016

- エラーメッセージ, 3059
 - 概要, 3059
 - 監視, 3064
 - 構成, 3059
 - チュートリアル, 3059
 - バイナリログソースの追加, 3062
 - パフォーマンススキーマ, 3064
 - プロビジョニング, 3060
 - レプリカの開始, 3063
 - レプリカの停止, 3063
 - レプリカのリセット, 3063
- マルチパートインデックス, 2194
- マルチバイト文字, 1743
- マルチバイト文字セット, 4597
- 丸め, 2142
- 丸め誤差, 1764
- 満杯のディスク, 4603, 4603
- 未修飾オプション
 - ndb_desc, 3760
 - ndb_show_tables, 3822
- ミッドポイント挿入, 2733
- ミッドポイント挿入戦略, 5391
- 見積, 1629
 - アカウント名, 1070
 - アカウント名のホスト名, 1070
 - アカウント名のユーザー名, 1070
 - スキーマオブジェクト, 2504
- ミニトランザクション, 5391
- ミラーサイト, 85
- 明示的なデフォルト値, 1824
- メイン接続インタフェース, 865
- メタデータ
 - InnoDB, 4191
 - ストアドルーチン, 4099
 - データベース, 4132
 - データベースオブジェクト, 1691
 - トリガー, 4103
 - ビュー, 4116
 - 不可視インデックス, 1509
 - 不可視のカラム, 2265
- メタデータのロック, 1597, 4355
- メタデータロック, 5391
- メトリックカウンタ, 5391
- メトリックビュー
 - sys スキーマ, 4467
- メモリー使用量
 - myisamchk, 507
 - NDB Cluster 内, 3484
 - 監視, 1607
- メモリーの使用, 1605
 - パフォーマンススキーマ, 4246
- メモリー割り当てライブラリ, 177, 337
- モード
 - バッチ, 286
- 文字
 - マルチバイト, 1743
- 文字セット, 1686
 - binary, 1739
 - Unicode, 1725
 - アジア系, 1735

- およびレプリケーション, 3213
- キリル文字, 1734
- 新機能, 19
- 制約, 1740
- 中央ヨーロッパ言語, 1733
- 中東, 1733
- 追加, 1741
- 西ヨーロッパ言語, 1731
- バルト語, 1734
- 南ヨーロッパ, 1733
- レパートリー, 1689
- 文字セットの概要, 1698
- 文字セットレパートリー, 1720
- モジユロ (%), 1879
- 文字列
 - エスケープシーケンス, 1627
 - 区切り文字がない, 1631
 - 定義済, 1627
 - レパートリー, 1689
- 文字列演算子, 1901
- 文字列型, 1781
- 文字列関数, 1901
- 文字列照合, 1743
- 文字列データ型, 1781
 - 記憶域要件, 1828
- 文字列の連結, 1627, 1904
- 文字列比較
 - 大文字/小文字の区別, 1914
- 文字列比較演算子, 1914
- 文字列比較関数, 1914
- 文字列リテラル, 1627
- 文字列リテラルイントロデューサ, 1628, 1696
- もっとも大きいタイムスタンプ、削除が優先 (競合解決), 4020
- もっとも大きいタイムスタンプが優先 (競合解決), 4020
- モニター, 2931
 - InnoDB, 2976
 - output, 2933
 - 端末, 265
 - 有効化, 2932
- モニターオプション
 - ndb_import, 3771
- モニタリング, 4529
- モノ, 5391
- 問題
 - DATE カラム, 4607
 - MySQL Server のコンパイル, 218
 - Perl のインストール, 263
 - Solaris へのインストール, 180
 - アクセスは拒否されましたエラー, 4588
 - 一般的なエラー, 4587
 - サーバーの起動, 226
 - 接続が失われましたエラー, 4591
 - タイムゾーン, 4605
 - テーブルロック, 1595
 - レポート, 2, 62
- 問題点
 - 日付値, 1774
- 有効な JSON 値, 1811
- 有効な数値
 - 例, 1630

- ユーザーアカウント
 - reserved, 1081
 - 作成, 1077, 2490
 - デュアルパスワード, 1104
 - 名前変更, 2514
 - 変更, 2477
 - リソース制限, 735, 1122, 2486, 2498
- ユーザーアカウントの作成, 2490
- ユーザーアカウントの名前変更, 2514
- ユーザーアカウントの変更, 2477
- ユーザーオプション, 314
 - mysql, 379
 - mysqladmin, 403
 - mysqlbinlog, 535
 - mysqlcheck, 413
 - mysqld, 669
 - mysqldump, 424
 - mysqld_multi, 344
 - mysqld_safe, 339
 - mysqlimport, 447
 - mysqlpump, 462, 462
 - mysqlshow, 472
 - mysqslap, 482
 - mysql_secure_installation, 350
 - mysql_upgrade, 362
 - ndb_top, 3829
- ユーザー管理, 1043
- ユーザー権限
 - 削除, 2501, 2501
 - チェック, 1080
 - 追加, 1077
 - 取消し, 1080
- ユーザー定義関数 (参照 [UDF](#))
 - 削除, 2537
 - 作成, 2536
- ユーザー定義変数, 1673
- ユーザー変数
 - およびレプリケーション, 3234
 - 削除された機能, 44
- ユーザー名
 - アカウント名, 1070
 - デフォルトアカウント, 230
 - とパスワード, 1044
 - ロール名, 1072
- ユーザー名の長さ
 - レプリケーション, 3234
- 優先順位
 - 演算子, 1859
- ユーティリティ
 - プログラム開発, 298
- ユーティリティプログラム, 298
- 予期しない停止
 - レプリケーション, 3095, 3193
 - 「よく知られたテキスト」形式
 - ジオメトリ値, 1801
 - 「よく知られているバイナリ」形式
 - ジオメトリ値, 1802
- 読取り競合の検出および解決
 - NDB Cluster レプリケーション, 4026
- 読取り専用オプション

- myisamchk, 498
- 読取り専用データベース
 - ALTER DATABASE, 2156
- 読み取り専用トランザクション, 5391
- 読み取りのロック, 2712
 - NOWAIT, 2712
 - SKIP LOCKED, 2712
- 読み取りビュー, 5391
- 予約語, 1645
 - およびレプリケーション, 3225
- 予約済ユーザーアカウント, 1081
- より大きい (>), 1863
- より小さい (<), 1862
- ラージページオプション
 - mysqld, 653
- ラージページのサポート, 1609
- ライフサイクルインサータ, 5392
- ラッチ, 5392
- ラッパー
 - Eiffel, 4527
- ラテラル導出テーブル, 1547, 2353
 - 新機能, 25
- ラベル
 - ストアドプログラムブロック, 2444
- ランダムダイブ, 5392
- リカバリ
 - クラッシュから, 1423
 - 増分, 1419
 - ポイントインタイム, 1419
- リカバリオプション
 - myisamchk, 500
- リスト, 5392
- リストア, 5392
- リストオプション
 - MySQLInstallerConsole, 126
- リソース管理
 - 新機能, 10
- リソースグループ, 885
 - 名前, 1635
- リソースグループ名
 - 大/小文字の区別, 1639
- リソース制限
 - ユーザーアカウント, 735, 1122, 2486, 2498
- リソース不足エラー
 - およびパーティション化されたテーブル, 4084
- リテラル, 1627
 - 16進数, 1632
 - date, 1630
 - numeric, 1630
 - string, 1627
 - time, 1630
 - ビット値, 1634
 - ブール, 1635
- リモート管理 (NDB Cluster)
 - セキュリティの問題, 3987
- リライタ UDF
 - load_rewrite_rules(), 965
- リライタクエリーリライトプラグイン, 958
 - アンインストール, 958
 - インストール, 958

- リリース
 - DMR, 84
 - GA, 84
 - RC, 84
 - ネーミングスキーム, 84
- リリース番号, 84
- リレーショナル, 5392
- リレーショナルデータベース
 - 定義, 4
- リレーログ (レプリケーション), 3163
- 履歴リスト, 5392
- リンク
 - シンボリック, 1603
- 隣接ページ, 5393
- ルーズインデックススキャン
 - GROUP BY 最適化, 1471
- ルーチン
 - stored, 4095, 4097
- ルーチンオプション
 - mysqldump, 434
 - mysqlpump, 460
- ルーチンテーブル
 - データディクショナリテーブル, 896
- 例
 - myisamchk の出力, 501
 - 圧縮テーブル, 510
 - クエリー, 287
- 例外インターセプタ, 5393
- 例外テーブル
 - NDB Cluster レプリケーション, 4023
- レコードレベルロック
 - InnoDB, 2717
- レコードロック, 5393
- レパートリー, 5393
 - string, 1689
 - 文字セット, 1689, 1720
- レプリカ, 5393
 - ステートメント, 2398
- レプリケーション, 非同期 (参照 [NDB Cluster レプリケーション](#))
- レプリケーション UDF
 - asynchronous_connection_failover_add_managed(), 2432
 - asynchronous_connection_failover_add_source(), 2431
 - asynchronous_connection_failover_delete_managed(), 2433
 - asynchronous_connection_failover_delete_source(), 2431
- レプリケーション, 3019, 5393
 - BLACKHOLE, 3213
 - CHECKSUM TABLE ステートメント, 3213
 - group, 3241
 - mysql (システム) スキーマ, 3225
 - NDB Cluster 内, 3990
 - (参照 [NDB Cluster レプリケーション](#))
 - REPAIR TABLE ステートメント, 2535
 - STATEMENT 形式と互換性のないステートメント, 3152
 - 安全および安全でないステートメント, 3155
 - 円形, 3993
 - および AUTO_INCREMENT, 3212
 - および CREATE ... IF NOT EXISTS, 3213
 - および CREATE TABLE ... SELECT, 3214
 - および DATA DIRECTORY, 3218
 - および DROP ... IF EXISTS, 3218

および FLUSH, 3219
および INDEX DIRECTORY, 3218
および LAST_INSERT_ID(), 3212
および LIMIT, 3223
および LOAD DATA, 3223
および max_allowed_packet, 3223
および MEMORY テーブル, 3224
および REPAIR TABLE ステートメント, 3225
および SQL モード, 3228
および TIMESTAMP, 3212
および TRUNCATE TABLE, 3234
および一時テーブル, 3228
および関数, 3219
およびクエリーオプティマイザ, 3225
および小数秒, 3221
およびスケジュールされたイベント, 3221, 3221
およびストアドルーチン, 3221
およびタイムゾーン, 3229
およびトランザクション, 3231
およびトリガー, 3221, 3233
およびパーティショニング, 3228
およびビュー, 3236
および浮動小数点値, 3219
および変数, 3234
および文字セット, 3213
および呼び出される機能, 3221
および予約語, 3225
行ベース対ステートメントベース, 3151
クラッシュ, 3227
権限, 3225
シャットダウンおよび再起動, 3228
シャットダウンと再起動, 3227
準同期, 3204
新機能, 25
スレッドの状態, 1622, 1623, 1624
ソースとレプリカで異なるテーブルを使用, 3215
属性降格, 3216
属性昇格, 3216
タイムアウト, 3229
遅延, 3209
トランザクション, 3229
パーティションテーブル, 3225
非決定的関数, 1475
部分更新, 3227
ユーザー名の長さ, 3234
予期しない停止, 3193
リソースグループ, 888
リレーログ, 3163
レプリカでのエラー, 3227
レプリケーションメタデータリポジトリ, 3163
レプリケーションオプション, 3212
レプリケーション形式
 比較, 3151
レプリケーションサーバー
 ステートメント, 2433
レプリケーションソース
 ステートメント, 2395
 スレッドの状態, 1622
レプリケーションソースを次に変更, 2411
 NDB Cluster 内, 4002

- レプリケーションチャンネル, 3157
 - 起動オプション, 3159
 - 互換性, 3159
 - コマンド, 3158
 - 命名規則, 3160
 - レプリケーションチャンネルベースのフィルタ, 3174
 - レプリケーションテクノロジー, 3243
 - レプリケーションで増加
 - 速度, 3019
 - レプリケーションの実装, 3150
 - レプリケーションの制限事項, 3212
 - レプリケーションフィルタリングオプション
 - および大文字と小文字の区別, 3169
 - レプリケーションメタデータリポジトリ, 3163
 - レプリケーションモード, 3053
 - オンラインでの無効化, 3057
 - オンラインでの有効化, 3055
 - 概念, 3053
 - 匿名トランザクションの検証, 3058
 - レポート
 - エラー, 62
 - バグ, 2, 62
 - 問題, 2
 - 連結
 - string, 1904
 - 文字列, 1627
 - 連結解除オプション
 - mysqslap, 479
 - ローカルオプション
 - mysqlimport, 444, 1040
 - ローカルデータのロードは無効です。これはクライアント側とサーバー側の両方で有効にする必要があります
 - エラーメッセージ, 1041
 - ロード
 - テーブル, 272
 - ロードバランシング, 5394
 - ローリング再起動 (NDB Cluster), 3871
 - ロールの削除, 2501
 - ロールの作成, 2490
 - ロールの取消し, 2515
 - ロールの付与, 2502
 - ロールの割当て, 2519
 - ロールバック, 5395
 - ロールバックセグメント, 2682, 2685, 5395
 - ロール名, 1072
- ロギング
 - パスワード, 1035
 - ロギングコマンド (NDB Cluster), 3859
 - ログ, 5394
 - サーバー, 900
 - フラッシュ, 900
 - ロググループ, 5394
 - ログコンポーネント
 - log_filter_dragnet, 944
 - log_filter_internal, 944
 - log_sink_internal, 944
 - log_sink_json, 944
 - log_sink_syseventlog, 945
 - log_sink_test, 945
 - ログバッファ, 5394
 - ログファイル (NDB Cluster), 3730

- ndbmysd, 3733
- ログファイル, 5394
 - 保守, 940
- ログレベルオプション
 - ndb_import, 3770
- ロック, 2700, 5394, 5394
 - external, 650, 774
 - InnoDB, 2701
 - 外部, 1423, 1600, 1620
 - 行レベル, 1593
 - 情報スキーマ, 2896
 - テーブルレベル, 1593
 - 内部, 1593
 - パフォーマンススキーマ, 2896
 - メタデータ, 1597
- ロックエスケレーション, 5394
- ロック関数, 1990
- ロックサービス
 - mysql_acquire_locking_service_locks() C 関数, 1001
 - mysql_release_locking_service_locks() C 関数, 1002
 - service_get_read_locks() UDF, 1005
 - service_get_write_locks() UDF, 1005
 - service_release_locks() UDF, 1005
 - アンインストール, 1002
 - インストール, 1002
- ロック方法, 1593
- ロックモニター, 2931, 2933
- ロック読み取り, 5394
- 論理, 5395
- 論理演算子, 1866
- 論理バックアップ, 5395
- ワークロード, 5395
- ワイルドカード
 - mysql.columns_priv テーブル内, 1076
 - mysql.db テーブル内, 1076
 - mysql.procs_priv テーブル内, 1076
 - mysql.tables_priv テーブル内, 1076
 - アカウント名, 1071
 - および LIKE, 1505
- ワイルドカード文字 (%), 1628
- ワイルドカード文字 (_, 1629
- 割り当て演算子, 1868

A

- abort-on-error オプション
 - ndb_import, 3766
 - ndb_move_data, 3780
- abort-slave-event-count オプション
 - mysqld, 3090
- Aborted_clients ステータス変数, 835
- Aborted_connects ステータス変数, 835
- ABS(), 1875
- account
 - default, 230
 - root, 230
- accounts
 - reserved, 1081
 - 権限の追加, 1077
 - 削除, 1080

作成, 1077
accounts テーブル
 performance_schema, 4324
account_locked カラム
 user table, 1066
ACID, 5336
ACID, 2626, 2630
ACLs, 1043
Acl_cache_items_count ステータス変数, 835, 835
ACOS(), 1875
activate_all_roles_on_login システム変数, 671
ActiveState Perl, 263
add-drop-database オプション
 mysqldump, 425
 mysqlpump, 453
add-drop-table オプション
 mysqldump, 426
 mysqlpump, 453
add-drop-trigger オプション
 mysqldump, 426
add-drop-user オプション
 mysqlpump, 453
add-locks オプション
 mysqldump, 436
 mysqlpump, 453
add-missing オプション
 ndb_blob_tool, 3744
ADDDATE(), 1885
ADDTIME(), 1885
ADD_GDB_INDEX オプション
 CMake, 204
admin-ssl オプション
 mysqld, 645
ADMINISTRABLE_ROLE_AUTHORIZATIONS
 INFORMATION_SCHEMA テーブル, 4135
admin_address システム変数, 671
admin_port システム変数, 672
admin_ssl_ca システム変数, 672
admin_ssl_capath システム変数, 673
admin_ssl_cert システム変数, 673
admin_ssl_cipher システム変数, 673
admin_ssl_crl システム変数, 674
admin_ssl_crlpath システム変数, 674
admin_ssl_key システム変数, 674
admin_tls_ciphersuites システム変数, 675
admin_tls_version システム変数, 675
ADO.NET, 5337
AES_DECRYPT(), 1986
AES_ENCRYPT(), 1987
After create
 スレッドの状態, 1616
ai-increment オプション
 ndb_import, 3766
ai-off オプション
 ndb_import, 3767
ai-prefetch-sz オプション
 ndb_import, 3767
AIO, 5337
ALL, 2347
 SELECT 修飾子, 2330

- ALL PRIVILEGES 権限, 1048
- ALL 結合タイプ
 - オプティマイザ, 1545
- ALL 権限, 1048
- all-databases オプション
 - mysqlcheck, 408
 - mysqldump, 433
 - mysqlpump, 454
- all-in-1 オプション
 - mysqlcheck, 408
- all-tables オプション
 - mysqldump, 426
- allow-keywords オプション
 - mysqldump, 426
- allow-mismatches オプション
 - innochecksum, 488
- allow-pk-changes オプション
 - ndb_restore, 3792
- allow-suspicious-udfs オプション
 - mysqld, 645
- AllowSpinOverhead, 3626
- AllowUnresolvedHostNames, 3709
- ALLOW_INVALID_DATES SQL モード, 856
- ALTER COLUMN, 2175
- ALTER DATABASE, 2156
 - 削除された機能, 41
- ALTER EVENT, 2160
 - およびレプリケーション, 3221
- ALTER FUNCTION, 2162
- ALTER INSTANCE, 2162
- ALTER LOGFILE GROUP, 2163
 - (参照 [NDB Cluster データ](#))
- ALTER privilege, 1048
- ALTER PROCEDURE, 2165
- ALTER RESOURCE GROUP ステートメント, 2520
- ALTER ROUTINE 権限, 1048
- ALTER SCHEMA, 2156
- ALTER SERVER, 2165
- ALTER TABLE ... UPGRADE PARTITIONING
 - 削除された機能, 44
- ALTER TABLE, 2165, 2175, 4612
 - ROW_FORMAT, 2781
 - 監視, 2926
 - レプリケーションメタデータリポジトリ, 3163
- ALTER TABLESPACE
 - NDB Cluster データ, 2187
 - undo テーブルスペース, 2187
 - 一般テーブルスペース, 2187
- ALTER USER ステートメント, 1097, 2477
- ALTER VIEW, 2188
- altering table
 - スレッドの状態, 1616
- ANALYZE TABLE
 - およびパーティショニング, 4071
- ANALYZE TABLE ステートメント, 2523
- Analyzing
 - スレッドの状態, 1616
- AND
 - logical, 1867
 - ビット単位, 1983

ANSI, 5337
ANSI SQL モード, 855, 860
ansi オプション
 mysqld, 646
ANSI モード
 実行, 67
ANSI_QUOTES SQL モード, 856
ANY, 2346
ANY_VALUE(), 2129
Apache, 294
API, 5337
API
 Perl, 4525
 のリスト, 78
API ノード (NDB Cluster)
 定義済, 3450
API ノード (参照 SQL ノード)
APIs, 4521
append オプション
 ndb_restore, 3793
APPLICABLE_ROLES
 INFORMATION_SCHEMA テーブル, 4136
APPLICATION_PASSWORD_ADMIN 権限, 1054
apply-slave-statements オプション
 mysqldump, 428
apply_status テーブル (OBSOLETE), 4000
 (参照 [NDB Cluster レプリケーション](#))
Arbitration, 3614
ArbitrationDelay, 3573, 3651
ArbitrationRank, 3572, 3651
ArbitrationTimeout, 3614
arbitrator_validity_detail
 ndbinfo テーブル, 3914
arbitrator_validity_summary
 ndbinfo テーブル, 3914
ARCHIVE ストレージエンジン, 2983, 3001
Area()
 削除された機能, 40
array
 JSON, 1811
AS, 2327, 2334
AsBinary()
 削除された機能, 40
ASCII(), 1903
ASIN(), 1876
ASP.net, 5337
AsText()
 削除された機能, 40
AsWKB()
 削除された機能, 40
AsWKT()
 削除された機能, 40
asymmetric_decrypt(), 1392
asymmetric_derive(), 1393
asymmetric_encrypt(), 1393
asymmetric_sign(), 1394
asymmetric_verify(), 1394
asynchronous_connection_failover_add_managed() UDF, 2432
asynchronous_connection_failover_add_source() UDF, 2431
asynchronous_connection_failover_delete_managed() UDF, 2433

asynchronous_connection_failover_delete_source() UDF, 2431
ATAN(), 1876
ATAN2(), 1876
attributes
 リソースグループ, 885
audit-log オプション
 mysqld, 1335
AUDIT_ADMIN 権限, 1055
audit_api_message_emit_udf() 監査 API UDF, 1346
audit_log プラグイン, 1279
 インストール, 1280
audit_log_buffer_size システム変数, 1336
audit_log_compression システム変数, 1336
audit_log_connection_policy システム変数, 1336
audit_log_current_session システム変数, 1337
Audit_log_current_size ステータス変数, 1345
audit_log_encryption システム変数, 1337
audit_log_encryption_password_get() 監査ログ暗号化 UDF, 1302, 1328
audit_log_encryption_password_set() 監査ログ暗号化 UDF, 1302, 1329
Audit_log_events ステータス変数, 1345
Audit_log_events_filtered ステータス変数, 1345
Audit_log_events_lost ステータス変数, 1345
Audit_log_events_written ステータス変数, 1345
Audit_log_event_max_drop_size ステータス変数, 1345
audit_log_exclude_accounts システム変数, 1338
audit_log_file システム変数, 1307, 1338
audit_log_filter テーブル
 システムテーブル, 899
audit_log_filter_flush() 監査ログフィルタ UDF, 1330
audit_log_filter_id システム変数, 1339
audit_log_filter_remove_filter() 監査ログフィルタ UDF, 1330
audit_log_filter_remove_user() 監査ログフィルタ UDF, 1331
audit_log_filter_set_filter() 監査ログフィルタ UDF, 1331
audit_log_filter_set_user() 監査ログフィルタ UDF, 1331
audit_log_flush システム変数, 1339
audit_log_format システム変数, 1339
audit_log_include_accounts システム変数, 1339
audit_log_password_history_keep_days システム変数, 1340
audit_log_policy システム変数, 1341
audit_log_prune_seconds システム変数, 1342
audit_log_read_buffer_size システム変数, 1310, 1342
audit_log_rotate_on_size システム変数, 1343
audit_log_statement_policy システム変数, 1343
audit_log_strategy システム変数, 1344
Audit_log_total_size ステータス変数, 1345
audit_log_user テーブル
 システムテーブル, 899
Audit_log_write_waits ステータス変数, 1345
authentication
 LDAP, 1176
 SASL, 1176
AUTHENTICATION_LDAP_CLIENT_LOG 環境変数, 550, 1206
authentication_ldap_sasl_auth_method_name システム変数, 1201
authentication_ldap_sasl_bind_base_dn システム変数, 1202
authentication_ldap_sasl_bind_root_dn システム変数, 1202
authentication_ldap_sasl_bind_root_pwd システム変数, 1203
authentication_ldap_sasl_ca_path システム変数, 1203
authentication_ldap_sasl_group_search_attr システム変数, 1203
authentication_ldap_sasl_group_search_filter システム変数, 1204
authentication_ldap_sasl_init_pool_size システム変数, 1204

authentication_ldap_sasl_log_status システム変数, 1205
authentication_ldap_sasl_max_pool_size システム変数, 1206
authentication_ldap_sasl_referral システム変数, 1206
authentication_ldap_sasl_server_host システム変数, 1206
authentication_ldap_sasl_server_port システム変数, 1207
Authentication_ldap_sasl_supported_methods ステータス変数, 835
authentication_ldap_sasl_tls システム変数, 1207
authentication_ldap_sasl_user_search_attr システム変数, 1208
authentication_ldap_simple_auth_method_name システム変数, 1208
authentication_ldap_simple_bind_base_dn システム変数, 1208
authentication_ldap_simple_bind_root_dn システム変数, 1209
authentication_ldap_simple_bind_root_pwd システム変数, 1209
authentication_ldap_simple_ca_path システム変数, 1210
authentication_ldap_simple_group_search_attr システム変数, 1210
authentication_ldap_simple_group_search_filter システム変数, 1210
authentication_ldap_simple_init_pool_size システム変数, 1211
authentication_ldap_simple_log_status システム変数, 1212
authentication_ldap_simple_max_pool_size システム変数, 1212
authentication_ldap_simple_referral システム変数, 1213
authentication_ldap_simple_server_host システム変数, 1213
authentication_ldap_simple_server_port システム変数, 1214
authentication_ldap_simple_tls システム変数, 1215
authentication_ldap_simple_user_search_attr システム変数, 1215
authentication_pam 認証プラグイン, 1162
AUTHENTICATION_PAM_LOG 環境変数, 550, 1171
authentication_windows 認証プラグイン, 1171
authentication_windows_client 認証プラグイン, 1171
authentication_windows_log_level システム変数, 675
authentication_windows_use_principal_name システム変数, 676
auth_socket 認証プラグイン, 1196
auth_test_plugin 認証プラグイン, 1198
auto-generate-sql オプション
 mysqlslap, 477
auto-generate-sql-add-autoincrement オプション
 mysqlslap, 477
auto-generate-sql-execute-number オプション
 mysqlslap, 477
auto-generate-sql-guid-primary オプション
 mysqlslap, 477
auto-generate-sql-load-type オプション
 mysqlslap, 477
auto-generate-sql-secondary-indexes オプション
 mysqlslap, 477
auto-generate-sql-unique-query-number オプション
 mysqlslap, 477
auto-generate-sql-unique-write-number オプション
 mysqlslap, 477
auto-generate-sql-write-number オプション
 mysqlslap, 477
auto-inc オプション
 ndb_desc, 3759
auto-inc 口ツク, 2701
auto-increment, 2658, 2665
auto-rehash オプション
 mysql, 367
auto-vertical-output オプション
 mysql, 368
auto.cnf ファイル, 3066
 SHOW REPLICAS | SHOW SLAVE HOSTS ステートメント, 2580
autocommit システム変数, 676

AutomaticThreadConfig, 3628
automatic_sp_privileges システム変数, 677
AutoReconnect
 API および SQL ノード, 3653
auto_generate_certs システム変数, 677
AUTO_INCREMENT, 292, 1767
 および NULL 値, 4609
 およびレプリケーション, 3212
auto_increment_increment システム変数, 3073
auto_increment_offset システム変数, 3075
AVG(), 2091
AVG(DISTINCT), 2091
avoid_temporal_upgrade システム変数, 678

B

B ツリー, 5337
B ツリーインデックス, 1504, 2666
BACKUP イベント (NDB Cluster), 3866
backup-password オプション
 ndb_restore, 3794
backup-path オプション
 ndb_restore, 3793
BackupDataBufferSize, 3621, 3887
BackupDataDir, 3579
BackupDiskWriteSpeedPct, 3621
backupid オプション
 ndb_restore, 3794
BackupLogBufferSize, 3621, 3887
BackupMaxWriteSize, 3623, 3888
BackupMemory, 3622, 3887
BackupReportFrequency, 3622
backups
 NDB Cluster 内, 3788, 3884, 3884, 3884, 3887
 NDB Cluster レプリケーション, 4007
 データベースおよびテーブル, 447
BackupWriteSize, 3622, 3887
BACKUP_ADMIN 権限, 1055
back_log システム変数, 678
base64-output オプション
 mysqlbinlog, 526
basedir オプション
 mysql.server, 342
 mysqld, 646
 mysqld_safe, 336
basedir システム変数, 679
BatchByteSize, 3651
Batched Key Access
 最適化, 1460, 1462
BatchSize, 3651
BatchSizePerLocalScan, 3590
BEGIN, 2376, 2443
 XA トランザクション, 2391
 ラベル, 2444
BENCHMARK(), 1993
BETWEEN ... AND, 1863
big5, 4563
BIGINT データ型, 1764
big_tables システム変数, 679
BIN(), 1903

BINARY, 1955
非推奨となった機能, 36
BINARY データ型, 1783, 1786
binary-as-hex オプション
mysql, 368
binary-mode オプション
mysql, 369
bind-address オプション
mysql, 369
mysqldadmin, 399
mysqlbinlog, 526
mysqlcheck, 408
mysqldump, 422
mysqlimport, 442
mysqlpump, 454
mysqlshow, 468
mysql_upgrade, 358
ndb_mgmd, 3735
bind_address システム変数, 679
binlog, 5337
Binlog Dump
スレッドのコマンド, 1614
BINLOG ステートメント, 2598
mysqlbinlog 出力, 538
binlog-checksum オプション
mysqld, 3115
binlog-do-db オプション
mysqld, 3113
binlog-ignore-db オプション
mysqld, 3115
binlog-row-event-max-size オプション
mysqlbinlog, 526
mysqld, 3111
BINLOG_ADMIN 権限, 1055
Binlog_cache_disk_use ステータス変数, 835
binlog_cache_size システム変数, 3116
Binlog_cache_use ステータス変数, 835
binlog_checksum システム変数, 3117
binlog_direct_non_transactional_updates システム変数, 3117
binlog_encryption システム変数, 3118
BINLOG_ENCRYPTION_ADMIN 権限, 1055
binlog_error_action システム変数, 3119
binlog_expire_logs_seconds, 3119
binlog_format
BLACKHOLE, 3213
binlog_format システム変数, 3120
binlog_group_commit_sync_delay, 3122
binlog_group_commit_sync_no_delay_count, 3122
binlog_gtid_simple_recovery, 3138
binlog_index テーブル (OBSOLETE) (参照 [NDB Cluster レプリケーション](#))
binlog_max_flush_queue_time システム変数, 3123
binlog_order_commits システム変数, 3123
binlog_rotate_encryption_master_key_at_startup システム変数, 3124
binlog_rows_query_log_events システム変数, 3127
binlog_row_event_max_size システム変数, 3124
binlog_row_image システム変数, 3124
binlog_row_metadata システム変数, 3126
binlog_row_value_options システム変数, 3126
Binlog_stmt_cache_disk_use ステータス変数, 835
binlog_stmt_cache_size システム変数, 3128

Binlog_stmt_cache_use ステータス変数, 836
binlog_transaction_compression システム変数, 3128
binlog_transaction_compression_level_zstd システム変数, 3129
binlog_transaction_dependency_history_size システム変数, 3130
binlog_transaction_dependency_tracking システム変数, 3129
BIN_TO_UUID(), 2130
bit 関数, 1975
BIT データ型, 1762
BIT_AND(), 2091
BIT_COUNT, 291
BIT_COUNT(), 1985
BIT_LENGTH(), 1903
BIT_OR, 291
BIT_OR(), 2092
BIT_XOR(), 2092
BLACKHOLE
 binlog_format, 3213
 レプリケーション, 3213
BLACKHOLE ストレージエンジン, 2983, 3002
BLOB, 5337
BLOB カラム
 インデックス設定, 2222
 インデックス付け, 1500
 サイズ, 1829
 デフォルト値, 1787
 バイナリデータの挿入, 1629
BLOB データ型, 1783, 1787
blob-info オプション
 ndb_desc, 3759
Block Nested Loop
 最適化, 1460, 1461
Block Nested Loop 結合アルゴリズム, 1449
block-search オプション
 myisamchk, 500
blocks
 ndbinfo テーブル, 3916
block_encryption_mode システム変数, 681
BOOL データ型, 1763
BOOLEAN データ型, 1763
browser-start-page オプション
 ndb_setup.py, 3819
Buffer()
 削除された機能, 40
bugs.mysql.com, 62
BuildIndexThreads, 3624
BUILD_CONFIG オプション
 CMake, 200
bulk_insert_buffer_size システム変数, 682, 2990
BUNDLE_RUNTIME_LIBRARIES オプション
 CMake, 200
Bytes_received ステータス変数, 836
Bytes_sent ステータス変数, 836

C

C, 5338
C API, 5338
C API, 4521, 4525
 FAQ, 4573
 新機能, 26

C API UDFs
 mysql_query_attribute_string(), 1682
C#, 5338
C++, 5338
C++, 4524
C:\my.cnf オプションファイル, 1016
ca-certs-file オプション
 ndb_setup.py, 3819
CACHE INDEX ステートメント, 2598
cache_policies テーブル, 2971
caching_sha2_password
 サポートされませんエラー, 239
 認証方式がクライアントに不明ですエラー, 239
 ロードできませんエラー, 239
caching_sha2_password 認証プラグイン, 1152
 互換性, 238
caching_sha2_password_auto_generate_rsa_keys システム変数, 683
caching_sha2_password_digest_rounds システム変数, 682
caching_sha2_password_private_key_path システム変数, 683
caching_sha2_password_public_key_path システム変数, 684
Caching_sha2_password_rsa_public_key ステータス変数, 836
CALL, 2290
CAN_ACCESS_COLUMN(), 2127
CAN_ACCESS_DATABASE(), 2127
CAN_ACCESS_TABLE(), 2127
CAN_ACCESS_USER(), 2128
CAN_ACCESS_VIEW(), 2128
CASE, 1869, 2446
CAST, 1956
CC 環境変数, 218, 550
CEIL(), 1876
CEILING(), 1876
Centroid()
 削除された機能, 40
cert-file オプション
 ndb_setup.py, 3819
cflags オプション
 mysql_config, 546
CHANGE MASTER TO, 2398
 NDB Cluster 内, 4002
CHANGE REPLICATION FILTER, 2409
Change user
 スレッドのコマンド, 1614
Changing master
 スレッドの状態, 1624
channel, 3157
 コマンド, 3158
CHAR VARYING データ型, 1783
CHAR データ型, 1781, 1782
CHAR(), 1903
CHARACTER VARYING データ型, 1783
CHARACTER データ型, 1782
character-set-client-handshake オプション
 mysqld, 646
character-sets-dir オプション (NDB Cluster プログラム), 3835, 3835
character-sets-dir オプション
 myisamchk, 499
 myisampack, 509
 mysql, 369
 mysqladmin, 399

- mysqlbinlog, 526
- mysqlcheck, 408
- mysqldump, 428
- mysqlimport, 442
- mysqlpump, 454
- mysqlshow, 468
- mysql_upgrade, 358
- ndb_move_data, 3780
- CHARACTER_LENGTH(), 1904
- CHARACTER_SETS
 - INFORMATION_SCHEMA テーブル, 4136
- character_sets テーブル
 - データディクショナリテーブル, 895
- character_sets_dir システム変数, 686
- character_set_client システム変数, 684
- character_set_connection システム変数, 685
- character_set_database システム変数, 685
- character_set_filesystem システム変数, 685
- character_set_results システム変数, 686
- character_set_server システム変数, 686
- character_set_system システム変数, 686
- charset オプション
 - comp_err, 346
- charset コマンド
 - mysql, 380
- CHARSET(), 1994
- CHAR_LENGTH(), 1904
- CHECK TABLE
 - およびパーティショニング, 4071
- CHECK TABLE ステートメント, 2527
- check オプション
 - myisamchk, 498
- CHECK 制約
 - ALTER TABLE, 2176
 - CREATE TABLE, 2255
 - JSON_SCHEMA_VALID() を使用, 2080
 - RENAME TABLE, 2288
 - SHOW CREATE TABLE, 2554
- check-missing オプション
 - ndb_blob_tool, 3744
- check-only-changed オプション
 - myisamchk, 498
 - mysqlcheck, 408
- check-orphans オプション
 - ndb_blob_tool, 3744
- check-upgrade オプション
 - mysqlcheck, 408
- Checking master version
 - スレッドの状態, 1622
- checking permissions
 - スレッドの状態, 1616
- Checking table
 - スレッドの状態, 1616
- CHECKPOINT イベント (NDB Cluster), 3861
- Checksum, 3710
- CHECKSUM TABLE
 - レプリケーション, 3213
- CHECKSUM TABLE ステートメント, 2531
- CHECK_CONSTRAINTS
 - INFORMATION_SCHEMA テーブル, 4137

check_constraints テーブル
データディクショナリテーブル, 895

check_proxy_users システム変数, 687, 1120

chroot オプション
mysqld, 646

CIDR 表記法
アカウント名, 1072

CJK (中国語、日本語、韓国語)
Access、PHP などの問題, 4563
big5, 4563
Big5 文字セットの問題 (中国語), 4563
CJKV, 4563
euckr 文字セットの問題 (韓国語), 4563
FAQ, 4563
gb2312、GBK, 4563
GB 文字セットの問題 (中国語), 4563
LIKE および FULLTEXT, 4563
LIKE および FULLTEXT の問題, 4563
MySQL 4.0 の動作, 4563
ORDER BY 処置, 4563, 4563
Unicode 照合, 4563
アクセス、PHP など, 4563
円記号, 4563
円記号の問題 (日本語), 4563
韓国語のドキュメント, 4563
韓国語文字セット, 4563
疑問符として表示される文字, 4563
拒否された文字, 4563
使用可能な文字セット, 4563
照合, 4563, 4563
ソート順序の問題, 4563, 4563
ソートの問題, 4563, 4563
中国語のドキュメント, 4563
データ切捨ての問題, 4563
データの切捨て, 4563
データベース名とテーブル名, 4563
特定の文字の使用可能性, 4563
日本語のドキュメント, 4563
日本語文字セット, 4563
日本語文字セットの変換の問題, 4563
ベトナム語, 4563
文字の可用性のテスト, 4563

CJK 文字によるデータの切捨て, 4563

ClassicFragmentation
ndbmt, 3628

cleaning up
スレッドの状態, 1616

clear command
mysql, 381

Clearing
スレッドの状態, 1625

client
シグナル処理, 554

CLOB, 5338

CLONE, 2540

CLONE_ADMIN 権限, 1055

clone_autotune_concurrency システム変数, 995

clone_buffer_size システム変数, 996

clone_ddl_timeout システム変数, 996

clone_enable_compression システム変数, 997

clone_max_concurrency システム変数, 997
clone_max_data_bandwidth システム変数, 997
clone_max_network_bandwidth システム変数, 998
clone_progress テーブル, 991
 performance_schema, 4370
clone_ssl_ca システム変数, 998
clone_ssl_cert システム変数, 999
clone_ssl_key システム変数, 999
clone_status テーブル, 991
 performance_schema, 4369
clone_valid_donor_list システム変数, 999
CLOSE, 2451
Close stmt
 スレッドのコマンド, 1614
closing tables
 スレッドの状態, 1616
cluster-config-suffix オプション
 ndb_config, 3747
 ndb_mgmd, 3735
cluster.binlog_index テーブル (OBSOLETE) (参照 [NDB Cluster レプリケーション](#))
CLUSTERLOG STATISTICS コマンド (NDB Cluster), 3867
CLUSTERLOG コマンド (NDB Cluster), 3859
cluster_locks
 ndbinfo テーブル, 3916
cluster_operations
 ndbinfo テーブル, 3917
cluster_replication データベース (OBSOLETE) (参照 [NDB Cluster レプリケーション](#))
cluster_transactions
 ndbinfo テーブル, 3919
CMake
 ADD_GDB_INDEX オプション, 204
 BUILD_CONFIG オプション, 200
 BUNDLE_RUNTIME_LIBRARIES オプション, 200
 CMAKE_BUILD_TYPE オプション, 200
 CMAKE_CXX_FLAGS オプション, 216
 CMAKE_C_FLAGS オプション, 216
 CMAKE_INSTALL_PREFIX オプション, 200
 COMPILATION_COMMENT オプション, 204
 COMPILATION_COMMENT_SERVER オプション, 204
 COMPRESS_DEBUG_SECTIONS オプション, 204
 CPACK_MONOLITHIC_INSTALL オプション, 200
 DEFAULT_CHARSET オプション, 204
 DEFAULT_COLLATION オプション, 204
 DISABLE_PSI_COND オプション, 204
 DISABLE_PSI_DATA_LOCK オプション, 206
 DISABLE_PSI_ERROR オプション, 206
 DISABLE_PSI_FILE オプション, 204
 DISABLE_PSI_IDLE オプション, 205
 DISABLE_PSI_MEMORY オプション, 205
 DISABLE_PSI_METADATA オプション, 205
 DISABLE_PSI_MUTEX オプション, 205
 DISABLE_PSI_PS オプション, 205
 DISABLE_PSI_RWLOCK オプション, 205
 DISABLE_PSI_SOCKET オプション, 205
 DISABLE_PSI_SP オプション, 205
 DISABLE_PSI_STAGE オプション, 205
 DISABLE_PSI_STATEMENT オプション, 205
 DISABLE_PSI_STATEMENT_DIGEST オプション, 205
 DISABLE_PSI_TABLE オプション, 205
 DISABLE_PSI_THREAD オプション, 206

DISABLE_PSI_TRANSACTION オプション, 206
DISABLE_SHARED オプション, 205
DOWNLOAD_BOOST オプション, 206
DOWNLOAD_BOOST_TIMEOUT オプション, 206
ENABLED_LOCAL_INFILE オプション, 207, 1040
ENABLED_PROFILING オプション, 207
ENABLE_DOWNLOADS オプション, 206
ENABLE_EXPERIMENTAL_SYSVARS オプション, 206
ENABLE_GCOV オプション, 206
ENABLE_GPROF オプション, 206
FORCE_INSOURCE_BUILD オプション, 200
FORCE_UNSUPPORTED_COMPILER オプション, 207
FPROFILE_GENERATE オプション, 207
FPROFILE_USE オプション, 207
IGNORE_AIO_CHECK オプション, 207
INSTALL_BINDIR オプション, 200
INSTALL_DOCDIR オプション, 200
INSTALL_DOCREADMEDIR オプション, 200
INSTALL_INCLUDEDIR オプション, 200
INSTALL_INFODIR オプション, 201
INSTALL_LAYOUT オプション, 201
INSTALL_LIBDIR オプション, 201
INSTALL_MANDIR オプション, 201
INSTALL_MYSQLKEYRINGDIR オプション, 201
INSTALL_MYSQLSHAREDIR オプション, 201
INSTALL_MYSQLTESTDIR オプション, 201
INSTALL_PKGCONFIGDIR オプション, 201
INSTALL_PLUGINDIR オプション, 201
INSTALL_PRIV_LIBDIR オプション, 201
INSTALL_SBINDIR オプション, 202
INSTALL_SECURE_FILE_PRIVDIR オプション, 202
INSTALL_SHAREDIR オプション, 202
INSTALL_STATIC_LIBRARIES オプション, 202
INSTALL_SUPPORTFILESDIR オプション, 202
LINK_RANDOMIZE オプション, 202
LINK_RANDOMIZE_SEED オプション, 202
MAX_INDEXES オプション, 207
MEMCACHED_HOME オプション, 217
MUTEX_TYPE オプション, 208
MYSQLX_TCP_PORT オプション, 208
MYSQLX_UNIX_ADDR オプション, 208
MYSQL_DATADIR オプション, 202
MYSQL_MAINTAINER_MODE オプション, 208
MYSQL_PROJECT_NAME オプション, 208
MYSQL_TCP_PORT オプション, 208
MYSQL_UNIX_ADDR オプション, 208
NDB_UTILS_LINK_DYNAMIC, 217
ODBC_INCLUDES オプション, 202
ODBC_LIB_DIR オプション, 202
OPTIMIZER_TRACE オプション, 208
REPRODUCIBLE_BUILD オプション, 208
SYSCONFDIR オプション, 202
SYSTEMD_PID_DIR オプション, 202
SYSTEMD_SERVICE_NAME オプション, 203
TMPDIR オプション, 203
USE_LD_GOLD オプション, 208
USE_LD_LLD オプション, 209
VERSION ファイル, 219
WIN_DEBUG_NO_INLINE オプション, 209
WITH_ANT オプション, 209

WITH_ASAN オプション, 209
WITH_ASAN_SCOPE オプション, 209
WITH_AUTHENTICATION_LDAP オプション, 209
WITH_AUTHENTICATION_PAM オプション, 209
WITH_AWS_SDK オプション, 209
WITH_BOOST オプション, 209
WITH_BUNDLED_LIBEVENT オプション, 217
WITH_BUNDLED_MEMCACHED オプション, 217
WITH_CLASSPATH オプション, 217
WITH_CLIENT_PROTOCOL_TRACING オプション, 210
WITH_CURL オプション, 210
WITH_DEBUG オプション, 210
WITH_DEFAULT_COMPILER_OPTIONS オプション, 216
WITH_DEFAULT_FEATURE_SET オプション, 211
WITH_EDITLINE オプション, 211
WITH_ERROR_INSERT オプション, 217
WITH_GMOCK オプション, 211
WITH_ICU オプション, 211
WITH_INNODB_EXTRA_DEBUG オプション, 211
WITH_INNODB_MEMCACHED オプション, 211
WITH_JEMALLOC オプション, 211
WITH_KEYRING_TEST オプション, 211
WITH_LIBEVENT オプション, 211
WITH_LIBWRAP オプション, 212
WITH_LOCK_ORDER オプション, 212
WITH_LSAN オプション, 212
WITH_LTO オプション, 212
WITH_LZ4 オプション, 212
WITH_LZMA オプション, 212
WITH_MECAB オプション, 212
WITH_MSAN オプション, 213
WITH_MSCRT_DEBUG オプション, 213
WITH_MYSQLX オプション, 213
WITH_NDBCLUSTER オプション, 217
WITH_NDBCLUSTER_STORAGE_ENGINE オプション, 217
WITH_NDBMTD オプション, 217
WITH_NDB_BINLOG オプション, 218
WITH_NDB_DEBUG オプション, 218
WITH_NDB_JAVA オプション, 218
WITH_NDB_PORT オプション, 218
WITH_NDB_TEST オプション, 218
WITH_NUMA オプション, 213
WITH_PLUGIN_NDBCLUSTER オプション, 218
WITH_PROTOBUF オプション, 213
WITH_RAPID オプション, 213
WITH_RAPIDJSON オプション, 213
WITH_RE2 オプション, 213
WITH_ROUTER オプション, 214
WITH_SSL オプション, 214
WITH_SYSTEMD オプション, 214
WITH_SYSTEMD_DEBUG オプション, 214
WITH_SYSTEM_LIBS オプション, 214
WITH_TCMALLOC オプション, 215
WITH_TEST_TRACE_PLUGIN オプション, 215
WITH_TSAN オプション, 215
WITH_UBSAN オプション, 215
WITH_UNIT_TESTS オプション, 215
WITH_UNIXODBC オプション, 215
WITH_VALGRIND オプション, 215
WITH_ZLIB オプション, 216

- WITH_ZSTD オプション, 216
- 以前の起動後の実行, 188, 218
- オプション, 191
- CMakeCache.txt ファイル, 218
- CMAKE_BUILD_TYPE オプション
 - CMake, 200
- CMAKE_CXX_FLAGS オプション
 - CMake, 216
- CMAKE_C_FLAGS オプション
 - CMake, 216
- CMAKE_INSTALL_PREFIX オプション
 - CMake, 200
- COALESCE(), 1863
- COERCIBILITY(), 1994
- collation
 - 強制性, 1711
- COLLATION(), 1994
- COLLATIONS
 - INFORMATION_SCHEMA テーブル, 4137
- COLLATION_CHARACTER_SET_APPLICABILITY
 - INFORMATION_SCHEMA テーブル, 4138
- collation_connection システム変数, 687
- collation_database システム変数, 687
- collation_server システム変数, 688
- column-names オプション
 - mysql, 369
- column-statistics オプション
 - mysqldump, 435
 - mysqlpump, 454
- column-type-info オプション
 - mysql, 369
- columns
 - 記憶域要件, 1826
 - 名前, 1635
- COLUMNS
 - INFORMATION_SCHEMA テーブル, 4138
- COLUMNS パーティショニング, 4040
- COLUMNS_EXTENSIONS
 - INFORMATION_SCHEMA テーブル, 4141
- columns_priv テーブル
 - システムテーブル, 898, 1063
- COLUMN_PRIVILEGES
 - INFORMATION_SCHEMA テーブル, 4141
- COLUMN_STATISTICS
 - INFORMATION_SCHEMA テーブル, 4142
- column_statistics テーブル
 - システムテーブル, 896, 1584
- column_type_elements テーブル
 - データディクショナリテーブル, 896
- COMMIT, 2376
 - XA トランザクション, 2391
- committing alter table to storage engine
 - スレッドの状態, 1617
- Committing events to binlog
 - スレッドの状態, 1625
- compact オプション
 - mysqldump, 431
- compatible オプション
 - mysqldump, 431
- COMPILATION_COMMENT オプション

- CMake, 204
- COMPILATION_COMMENT_SERVER オプション
 - CMake, 204
- complete-insert オプション
 - mysqldump, 431
 - mysqlpump, 454
- completion_type システム変数, 688
- compress オプション, 319
 - mysql, 369
 - mysqladmin, 399
 - mysqlbinlog, 526
 - mysqlcheck, 408
 - mysqldump, 422
 - mysqlimport, 442
 - mysqlpump, 454
 - mysqlshow, 468
 - mysqlslap, 477
 - mysql_upgrade, 358
 - ndbxfm, 3833
- COMPRESS(), 1988
- compress-output オプション
 - mysqlpump, 455
- CompressedBackup, 3623
- CompressedLCP, 3601
- compression, 2774
 - BLOB、VARCHAR および TEXT, 2769
 - connection, 330, 3415
 - オーバーフローページ, 2769
 - 監視, 2766
 - テーブルに対する有効化, 2761
- compression-algorithms オプション, 319
 - mysql, 369
 - mysqladmin, 399
 - mysqlbinlog, 527
 - mysqlcheck, 408
 - mysqldump, 422
 - mysqlimport, 442
 - mysqlpump, 455
 - mysqlshow, 468
 - mysqlslap, 477
 - mysql_upgrade, 358
- Compression_algorithm ステータス変数, 837
- Compression_level ステータス変数, 837
- COMPRESS_DEBUG_SECTIONS オプション
 - CMake, 204
- comp_err, 296, 346
 - charset オプション, 346
 - debug-info オプション, 346
 - errmsg-file オプション, 346
 - header-file オプション, 346
 - in-file-errlog オプション, 347
 - in-file-toclient オプション, 347
 - name-file オプション, 347
 - out-dir オプション, 347
 - out-file オプション, 347
 - デバッグオプション, 346
 - バージョンオプション, 347
 - ファイル内オプション, 346
 - ヘルプオプション, 346
- Com_alter_db_upgrade

- 削除された機能, 41
- CONCAT(), 1904
- CONCAT_WS(), 1904
- concurrent_insert システム変数, 689
- conditions, 2563, 2596
- cond_instances テーブル
 - performance_schema, 4290
- config-cache オプション
 - ndb_mgmd, 3735
- config-file オプション
 - my_print_defaults, 548
 - ndb_config, 3748
 - ndb_mgmd, 3736
- config.ini (NDB Cluster), 3509, 3560, 3561, 3741
- configdir オプション
 - ndb_mgmd, 3736
- ConfigGenerationNumber, 3656
- configinfo オプション
 - ndb_config, 3747
- configuration
 - NDB Cluster, 3543
 - server, 556
 - 新機能, 26
- config_from_node オプション
 - ndb_config, 3748
- config_nodes
 - ndbinfo テーブル, 3920
- config_options テーブル, 2971
- config_params
 - ndbinfo テーブル, 3920
- config_values
 - ndbinfo テーブル, 3921
- Connect
 - スレッドのコマンド, 1614
- Connect Out
 - スレッドのコマンド, 1614
- CONNECT コマンド (NDB Cluster), 3839
- connect コマンド
 - mysql, 381
- connect-delay オプション
 - ndbd, 3726
 - ndbmt, 3726
- connect-expired-password オプション
 - mysql, 370
- connect-retries オプション (NDB Cluster プログラム), 3835
- connect-retries オプション
 - ndbd, 3726
 - ndbmt, 3726
 - ndb_mgm, 3742
- connect-retry-delay オプション (NDB Cluster プログラム), 3835
- connect-retry-delay オプション
 - ndbd, 3726
 - ndbmt, 3726
- connect-string オプション (NDB Cluster プログラム), 3837
- connect-timeout オプション
 - mysql, 370
 - mysqladmin, 399
- ConnectBackoffMaxTime, 3654
- ConnectCheckIntervalDelay, 3609
- Connecting to master

- スレッドの状態, 1622
- CONNECTION イベント (NDB Cluster), 3861
- connection-server-id オプション
 - mysqlbinlog, 527
- connection-timeout オプション
 - ndb_error_reporter, 3762
- ConnectionMap, 3649
- connections オプション
 - ndb_config, 3748
 - ndb_import, 3767
- CONNECTION_ADMIN 権限, 1055
- CONNECTION_CONTROL プラグイン
 - インストール, 1216
 - システム変数, 1219
 - ステータス変数, 1220
- Connection_control_delay_generated ステータス変数, 1220
- connection_control_failed_connections_threshold システム変数, 1219
- CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS INFORMATION_SCHEMA テーブル, 4233
- CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS プラグイン
 - インストール, 1216
- connection_control_max_connection_delay システム変数, 1220
- connection_control_min_connection_delay システム変数, 1220
- Connection_errors_accept ステータス変数, 837
- Connection_errors_internal ステータス変数, 837
- Connection_errors_max_connections ステータス変数, 837
- Connection_errors_peer_address ステータス変数, 837
- Connection_errors_select ステータス変数, 837
- Connection_errors_tcpwrap ステータス変数, 837
- CONNECTION_ID(), 1995
- Connector/C++, 5338
- Connector/C++, 4521, 4524
- Connector/J, 5338
- Connector/J, 4521, 4524
- Connector/NET, 5338
- Connector/NET, 4521, 4524
- Connector/Node.js, 4521, 4525
- Connector/ODBC, 5338
- Connector/ODBC, 4521, 4524
- Connector/PHP, 5338
- Connector/Python, 4521, 4525
- connect_timeout システム変数, 690
- const table
 - オペティマイザ, 1543, 2331
- containers テーブル, 2971
- Contains()
 - 削除された機能, 40
- continue オプション
 - ndb_import, 3767
- Control+C
 - ステートメントの終了, 363, 377, 2372
- CONV(), 1876
- CONVERT, 1957
- CONVERT TO, 2177
- CONVERT_TZ(), 1885
- ConvexHull()
 - 削除された機能, 40
- copy to tmp table
 - スレッドの状態, 1616
- Copying to group table

- スレッドの状態, 1616
- Copying to tmp table
 - スレッドの状態, 1617
- Copying to tmp table on disk
 - スレッドの状態, 1617
- core-file オプション (NDB Cluster プログラム), 3835
- core-file オプション, 2740
 - mysqld, 647
- core-file-size オプション
 - mysqld_safe, 336
- core_file システム変数, 690, 2740
- correct-checksum オプション
 - myisamchk, 499
- COS(), 1877
- COT(), 1877
- COUNT(), 2092
- COUNT(DISTINCT), 2093
- counters
 - ndbinfo テーブル, 3924
- CPACK_MONOLITHIC_INSTALL オプション
 - CMake, 200
- CPU バウンド, 5339
- cpudata
 - ndbinfo テーブル, 3925
- cpudata_1sec
 - ndbinfo テーブル, 3926
- cpudata_20sec
 - ndbinfo テーブル, 3926
- cpudata_50ms
 - ndbinfo テーブル, 3927
- cpuinfo
 - ndbinfo テーブル, 3928
- cpustat
 - ndbinfo テーブル, 3929
- cpustat_1sec
 - ndbinfo テーブル, 3930
- cpustat_20sec
 - ndbinfo テーブル, 3931
- cpustat_50ms
 - ndbinfo テーブル, 3929
- CrashOnCorruptedTuple, 3602
- CRC32(), 1877
- CREATE ... IF NOT EXISTS
 - およびレプリケーション, 3213
- CREATE DATABASE, 2189
- Create DB
 - スレッドのコマンド, 1614
- CREATE EVENT, 2189
 - およびレプリケーション, 3221
- CREATE FUNCTION, 2209
- CREATE FUNCTION ステートメント, 2536
- CREATE INDEX, 2194
- CREATE LOGFILE GROUP, 2207
 - (参照 [NDB Cluster ディスクデータ](#))
- CREATE NODEGROUP コマンド (NDB Cluster), 3842
- CREATE PROCEDURE, 2209
- CREATE RESOURCE GROUP ステートメント, 2521
- CREATE ROLE 権限, 1048
- CREATE ROLE ステートメント, 2490
- CREATE ROUTINE 権限, 1049

CREATE SCHEMA, 2189
CREATE SERVER, 2213
CREATE SPATIAL REFERENCE SYSTEM, 2214
CREATE TABLE ... SELECT
 およびレプリケーション, 3214
CREATE TABLE, 2218
 DIRECTORY オプション
 およびレプリケーション, 3218
 KEY_BLOCK_SIZE, 2766
 NDB_TABLE オプション, 2266
 ROW_FORMAT, 2781
 テーブル圧縮のオプション, 2761
CREATE TABLESPACE, 2268
 undo テーブルスペース, 2268
 一般テーブルスペース, 2268
CREATE TABLESPACE 権限, 1049
CREATE TEMPORARY TABLE
 非推奨となった機能, 36
CREATE TEMPORARY TABLES 権限, 1049
CREATE TRIGGER, 2275
CREATE USER 権限, 1049
CREATE USER ステートメント, 1077, 1097, 2490
CREATE VIEW, 2277
CREATE VIEW 権限, 1049
CREATE 権限, 1048
create-options オプション
 mysqldump, 431
create-schema オプション
 mysqlslap, 478
Created_tmp_disk_tables ステータス変数, 838
Created_tmp_files ステータス変数, 838
Created_tmp_tables ステータス変数, 838
create_admin_listener_thread システム変数, 690
create_asymmetric_priv_key(), 1394
create_asymmetric_pub_key(), 1395
create_dh_parameters(), 1395
create_digest(), 1395
create_synonym_db() プロシージャ
 sys スキーマ, 4491
Creating index
 スレッドの状態, 1617
Creating sort index
 スレッドの状態, 1617
creating table
 スレッドの状態, 1617
Creating tmp table
 スレッドの状態, 1617
CROSS JOIN, 2334
Crosses()
 削除された機能, 40
CRUD, 5339
CR_SERVER_GONE_ERROR, 4592
CR_SERVER_LOST_ERROR, 4592
csv オプション
 mysqlslap, 478
CSV ストレージエンジン, 2983, 2999
CSV データ、読み取り, 2311, 2333
cte_max_recursion_depth システム変数, 691
CUME_DIST(), 2111
CURDATE(), 1886

CURRENT_DATE, 1886
CURRENT_ROLE(), 1995
CURRENT_TIME, 1886
CURRENT_TIMESTAMP, 1886
Current_tls_ca ステータス変数, 838
Current_tls_cpath ステータス変数, 838
Current_tls_cert ステータス変数, 838
Current_tls_cipher ステータス変数, 838
Current_tls_ciphersuites ステータス変数, 839
Current_tls_crl ステータス変数, 839
Current_tls_crlpath ステータス変数, 839
Current_tls_key ステータス変数, 839
Current_tls_version ステータス変数, 839
CURRENT_USER(), 1995
CURTIME(), 1886
CXX 環境変数, 218, 550
cxxflags オプション
 mysql_config, 546

D

Daemon
 スレッドのコマンド, 1614
daemonize オプション
 mysqld, 647
daemon_memcached_enable_binlog システム変数, 2820
daemon_memcached_engine_lib_name システム変数, 2821
daemon_memcached_engine_lib_path システム変数, 2821
daemon_memcached_option システム変数, 2821
daemon_memcached_r_batch_size システム変数, 2822
daemon_memcached_w_batch_size システム変数, 2822
DATA DIRECTORY
 およびレプリケーション, 3218
data-file-length オプション
 myisamchk, 499
database
 名前変更, 2288
DATABASE(), 1996
DataDir, 3573, 3579
datadir オプション
 mysql.server, 342
 mysqld, 647
 mysqld_safe, 336
 mysql_ssl_rsa_setup, 352
datadir システム変数, 691
DataMemory, 3580
data_locks テーブル
 performance_schema, 4351, 4442
data_lock_waits テーブル
 performance_schema, 4353
DATE, 4607
DATE カラム
 問題, 4607
DATE データ型, 1771, 1772
DATE(), 1886
DATEDIFF(), 1886
DATETIME データ型, 1771, 1772
datetime_format
 削除された機能, 39
DATE_ADD(), 1886

- date_format
 - 削除された機能, 39
- DATE_FORMAT(), 1887
- DATE_SUB(), 1886, 1889
- DAY(), 1889
- DAYNAME(), 1889
- DAYOFMONTH(), 1889
- DAYOFWEEK(), 1889
- DAYOFYEAR(), 1889
- db table
 - システムテーブル, 230, 897, 1063
 - ソート, 1076
- db-workers オプション
 - ndb_import, 3767
- DB2
 - 削除された機能, 39
- DBI インタフェース, 4525
- DBI->quote, 1629
- DBI->trace, 1019
- DBI/DBD インタフェース, 4525
- DBI_TRACE 環境変数, 550, 1019
- DBI_USER 環境変数, 550
- DEBUG パッケージ, 1027
- DCL, 5339
- DCL, 2502, 2515
- DDEX プロバイダ, 5339
- DDL, 5339
- DDL, 2150, 2150
- ddl-rewriter オプション
 - mysqld, 967
- ddl_rewriter プラグイン, 966
 - インストール, 966
- dd_properties テーブル
 - データディクショナリテーブル, 896
- DEALLOCATE PREPARE, 2443
- deb ファイル
 - MySQL APT リポジトリ, 157
 - MySQL SLES リポジトリ, 157
- Debug
 - スレッドのコマンド, 1614
- debug オプション (NDB Cluster プログラム), 3835
- debug システム変数, 691
- debug-check オプション
 - mysql, 370
 - mysqladmin, 399
 - mysqlbinlog, 528
 - mysqlcheck, 409
 - mysqldump, 427
 - mysqlexport, 442
 - mysqlpump, 455
 - mysqlshow, 468
 - mysqlslap, 478
 - mysql_upgrade, 358
- debug-info オプション
 - comp_err, 346
 - mysql, 370
 - mysqladmin, 399
 - mysqlbinlog, 528
 - mysqlcheck, 409
 - mysqldump, 427

- mysqlimport, 442
- mysqlpump, 455
- mysqlshow, 468
- mysqlslap, 478
- mysql_upgrade, 358
- debug-sync-timeout オプション
 - mysqld, 648
- debug_sync システム変数, 692
- DEC データ型, 1764
- DECIMAL データ型, 1764, 2142
- DECLARE, 2444
- DECODE()
 - 削除された機能, 40
- decode_bits myisamchk 変数, 497
- decrypt オプション
 - ndb_restore, 3795
- decrypt-password オプション
 - ndbxfm, 3833
- DedicatedNode
 - API ノード, 3650
 - 管理サーバー, 3570
 - データノード, 3576
- default
 - privileges, 230
- DEFAULT value 句, 1824, 2223
- DEFAULT(), 2131
- default-auth オプション, 312
 - mysql, 370
 - mysqladmin, 400
 - mysqlbinlog, 528
 - mysqlcheck, 410
 - mysqldump, 422
 - mysqlimport, 443
 - mysqlpump, 455
 - mysqlshow, 469
 - mysqlslap, 478
 - mysql_upgrade, 358
- default-character-set オプション
 - mysql, 370
 - mysqladmin, 400
 - mysqlcheck, 409
 - mysqldump, 428
 - mysqlimport, 443
 - mysqlpump, 455
 - mysqlshow, 469
 - mysql_upgrade, 358
- default-parallel オプション
 - mysqlpump, 456
- default-time-zone オプション
 - mysqld, 648
- DefaultHashMapSize, 3593, 3653
- DefaultOperationRedoProblemAction
 - API および SQL ノード, 3653
- defaults-extra-file オプション, 307
 - myisamchk, 495
 - mysql, 370
 - mysqladmin, 400
 - mysqlbinlog, 528
 - mysqlcheck, 409
 - mysqld, 648

mysqldump, 425
mysqld_multi, 343
mysqld_safe, 336
mysqlexport, 443
mysqlpump, 456
mysqlshow, 469
mysqlslap, 478
mysql_secure_installation, 348
mysql_upgrade, 358
my_print_defaults, 548
NDB クライアントプログラム, 3836
defaults-file オプション, 307
myisamchk, 496
mysql, 371
mysqladmin, 400
mysqlbinlog, 528
mysqlcheck, 409
mysqld, 649
mysqldump, 425
mysqld_multi, 343
mysqld_safe, 336
mysqlexport, 443
mysqlpump, 456
mysqlshow, 469
mysqlslap, 478
mysql_secure_installation, 349
mysql_upgrade, 359
my_print_defaults, 548
NDB クライアントプログラム, 3836
defaults-group-suffix オプション, 307
myisamchk, 496
mysql, 371
mysqladmin, 400
mysqlbinlog, 529
mysqlcheck, 410
mysqld, 649
mysqldump, 425
mysqlexport, 443
mysqlpump, 456
mysqlshow, 469
mysqlslap, 479
mysql_secure_installation, 349
mysql_upgrade, 359
my_print_defaults, 548
NDB クライアントプログラム, 3836
default_authentication_plugin システム変数, 693
DEFAULT_CHARSET オプション
CMake, 204
DEFAULT_COLLATION オプション
CMake, 204
default_collation_for_utf8mb4 システム変数, 693
default_password_lifetime システム変数, 694
default_roles テーブル
システムテーブル, 898, 1063
default_storage_engine システム変数, 694
default_table_encryption, 2807
default_table_encryption 変数, 695
default_tmp_storage_engine システム変数, 695
default_week_format システム変数, 696
defer-table-indexes オプション

- mysqlpump, 456
- DEFINER 権限, 2566, 4116
- DEGREES(), 1877
- delay オプション
 - ndbinfo_select_all, 3732
- DELAYED, 2306
 - INSERT 修飾子, 2303
- Delayed insert
 - スレッドのコマンド, 1614
- Delayed_errors ステータス変数, 839
- delayed_insert_limit システム変数, 697
- Delayed_insert_threads ステータス変数, 839
- delayed_insert_timeout システム変数, 697
- delayed_queue_size システム変数, 698
- Delayed_writes ステータス変数, 839
- delay_key_write システム変数, 696, 2990
- DELETE, 2291
 - NDB Cluster, 3484
- DELETE 権限, 1049
- delete-master-logs オプション
 - mysqldump, 428
- delete-orphans オプション
 - ndb_blob_tool, 3744
- deleting from main table
 - スレッドの状態, 1617
- deleting from reference tables
 - スレッドの状態, 1617
- delimiter オプション
 - mysql, 371
 - mysqlslap, 479
 - ndb_select_all, 3816
- delimiter コマンド
 - mysql, 381
- demo_test テーブル, 2948
- DENSE_RANK(), 2111
- DESC, 2610
- descending オプション
 - ndb_select_all, 3815
- DESCRIBE, 285, 2610
- DES_DECRYPT()
 - 削除された機能, 40
- DES_ENCRYPT()
 - 削除された機能, 40
- DES_KEY_FILE
 - 削除された機能, 40
- diagnostics() プロシージャ
 - sys スキーマ, 4492
- DictTrace, 3620
- dict_obj_info
 - ndbinfo テーブル, 3932, 3932
- dict_obj_types
 - ndbinfo テーブル, 3935
- diff-default オプション
 - ndb_config, 3748
- Dimension()
 - 削除された機能, 40
- disable named コマンド
 - mysql, 371
- disable-indexes オプション
 - ndb_restore, 3795

disable-keys オプション
mysqldump, 435

disable-log-bin オプション
mysqlbinlog, 529

disabled_storage_engines システム変数, 698

DISABLE_PSI_COND オプション
CMake, 204

DISABLE_PSI_DATA_LOCK オプション
CMake, 206

DISABLE_PSI_ERROR オプション
CMake, 206

DISABLE_PSI_FILE オプション
CMake, 204

DISABLE_PSI_IDLE オプション
CMake, 205

DISABLE_PSI_MEMORY オプション
CMake, 205

DISABLE_PSI_METADATA オプション
CMake, 205

DISABLE_PSI_MUTEX オプション
CMake, 205

DISABLE_PSI_PS オプション
CMake, 205

DISABLE_PSI_RWLOCK オプション
CMake, 205

DISABLE_PSI_SOCKET オプション
CMake, 205

DISABLE_PSI_SP オプション
CMake, 205

DISABLE_PSI_STAGE オプション
CMake, 205

DISABLE_PSI_STATEMENT オプション
CMake, 205

DISABLE_PSI_STATEMENT_DIGEST オプション
CMake, 205

DISABLE_PSI_TABLE オプション
CMake, 205

DISABLE_PSI_THREAD オプション
CMake, 206

DISABLE_PSI_TRANSACTION オプション
CMake, 206

DISABLE_SHARED
削除された機能, 44

DISABLE_SHARED オプション
CMake, 205

DISCARD TABLESPACE, 2178, 2653

discard_or_import_tablespace
スレッドの状態, 1617

disconnect-slave-event-count オプション
mysqld, 3090

disconnect_on_expired_password システム変数, 699

Disjoint()
削除された機能, 40

DiskDataUsingSameDisk, 3644

DiskIOThreadPool, 3640, 3644

Diskless, 3602

diskpagebuffer
ndbinfo テーブル, 3938

DiskPageBufferEntries, 3639

DiskPageBufferMemory, 3639, 3644

disks
データの分割, 1604

diskstat
ndbinfo テーブル, 3939

diskstats_1sec
ndbinfo テーブル, 3940

DiskSyncSize, 3612

disk_write_speed_aggregate
ndbinfo テーブル, 3936

disk_write_speed_aggregate_node
ndbinfo テーブル, 3937

disk_write_speed_base
ndbinfo テーブル, 3935

Distance()
削除された機能, 40

DISTINCT, 275, 1472
AVG(), 2091
COUNT(), 2093
MAX(), 2097
MIN(), 2097
SELECT 修飾子, 2330
SUM(), 2098

DISTINCTROW
SELECT 修飾子, 2330

DIV, 1874

division (/), 1874

div_precision_increment システム変数, 699

DML, 5339

DML, 2290
DELETE ステートメント, 2291
INSERT ステートメント, 2299
TABLE ステートメント, 2358
UPDATE ステートメント, 2361
VALUES ステートメント, 2364

DMR
MySQL のリリース, 84

DN (参照 [識別名](#))

DNS, 868

DNS SRV レコード, 327, 371, 1213

dns-srv-name オプション
mysql, 371

DO, 2295

DocBook XML
ドキュメントソース形式, 4

Docker, 259

Docker イメージ
Windows の場合, 172

dont-ignore-systab-0 オプション
ndb_restore, 3795

DOUBLE PRECISION データ型, 1765

DOUBLE データ型, 1765

DOWNLOAD_BOOST オプション
CMake, 206

DOWNLOAD_BOOST_TIMEOUT オプション
CMake, 206

dragnet.log_error_filter_rules システム変数, 700

dragnet.Status ステータス変数, 839, 839

DROP ... IF EXISTS
およびレプリケーション, 3218

DROP DATABASE, 2281

Drop DB

- スレッドのコマンド, 1615
- DROP EVENT, 2282
- DROP FOREIGN KEY, 2176, 2252
- DROP FUNCTION, 2283
- DROP FUNCTION ステートメント, 2537
- DROP INDEX, 2175, 2282
- DROP LOGFILE GROUP, 2282
 - (参照 [NDB Cluster ディスクデータ](#))
- DROP NODEGROUP コマンド (NDB Cluster), 3842
- DROP PREPARE, 2443
- DROP PRIMARY KEY, 2175
- DROP PROCEDURE, 2283
- DROP RESOURCE GROUP ステートメント, 2522
- DROP ROLE 権限, 1049
- DROP ROLE ステートメント, 2501
- DROP SCHEMA, 2281
- DROP SERVER, 2283
- DROP SPATIAL REFERENCE SYSTEM, 2283
- DROP TABLE, 2284
 - NDB Cluster, 3484
- DROP TABLESPACE
 - NDB Cluster ディスクデータ, 2285
 - undo テーブルスペース, 2285
 - 一般テーブルスペース, 2285
- DROP TRIGGER, 2286
- DROP USER ステートメント, 1077, 2501
- DROP VIEW, 2286
- DROP 権限, 1049
- drop-source オプション
 - ndb_move_data, 3781
- dry-scp オプション
 - ndb_error_reporter, 3762
- DSN, 5339
- DTrace
 - 削除された機能, 44
- DUAL, 2326
- dump-date オプション
 - mysqldump, 427
- dump-file オプション
 - ibd2sdi, 484
 - ndb_blob_tool, 3744
- dump-slave オプション
 - mysqldump, 428
- DUMPFIL, 2333

E

- early-plugin-load オプション
 - mysqld, 649
- edit コマンド
 - mysql, 381
- ego コマンド
 - mysql, 381
- Eiffel, 5339
- Eiffel ラッパー, 4526
- ELT(), 1904
- enable-cleartext-plugin オプション
 - mysql, 372
 - mysqladmin, 400

- mysqlcheck, 410
- mysqldump, 422
- mysqlimport, 443
- mysqlshow, 469
- mysqlslap, 479
- EnableAdaptiveSpinning, 3626
- ENABLED_LOCAL_INFILE オプション
 - CMake, 207, 1040
- ENABLED_PROFILING オプション
 - CMake, 207
- ENABLED_ROLES
 - INFORMATION_SCHEMA テーブル, 4143
- EnablePartialLcp, 3594
- EnableRedoControl, 3598
- ENABLE_DOWNLOADS オプション
 - CMake, 206
- ENABLE_EXPERIMENTAL_SYSVARS オプション
 - CMake, 206
- ENABLE_GCOV オプション
 - CMake, 206
- ENABLE_GPROF オプション
 - CMake, 206
- ENCODE()
 - 削除された機能, 40
- ENCRYPT()
 - 削除された機能, 40
- encrypt-kdf-iter-count オプション
 - ndbxfm, 3833
- encrypt-password オプション
 - ndbxfm, 3833
- encryption, 2805
 - バイナリログファイル, 3180
- ENCRYPTION_KEY_ADMIN 権限, 1055
- end
 - スレッドの状態, 1617
- END, 2443
- end-page オプション
 - innochecksum, 487
- EndPoint()
 - 削除された機能, 40
- end_markers_in_json システム変数, 700
- enforce_gtid_consistency システム変数, 3139
- ENGINE
 - 非推奨となった機能, 36
- ENGINES
 - INFORMATION_SCHEMA テーブル, 4143
- engine_cost
 - システムテーブル, 1582
- engine_cost テーブル
 - システムテーブル, 899
- ENTER SINGLE USER MODE コマンド (NDB Cluster), 3841
- ENUM
 - サイズ, 1830
- ENUM データ型, 1784, 1788
- Envelope()
 - 削除された機能, 40
- equal (=), 1861
- Equals()
 - 削除された機能, 40
- eq_ref 結合タイプ

- オブティマイザ, 1543
- errins-delay オプション
 - ndb_import, 3768
- errins-type オプション
 - ndb_import, 3767
- errmsg-file オプション
 - comp_err, 346
- Error
 - スレッドのコマンド, 1615
- ERROR イベント (NDB Cluster), 3866
- error-insert オプション
 - ndb_move_data, 3781
- errors
 - ディレクトリチェックサム, 180
 - レプリケーション, 3227
- error_count システム変数, 701
- ERROR_FOR_DIVISION_BY_ZERO SQL モード, 856
- error_log テーブル
 - performance_schema, 4397
- Error_log_buffered_bytes ステータス変数, 839
- Error_log_buffered_events ステータス変数, 840
- Error_log_expired_events ステータス変数, 840
- Error_log_latest_write ステータス変数, 840
- error_messages
 - ndbinfo テーブル, 3941
- EVENT 権限, 1049
- EventLogBufferSize, 3617
- events, 4095
 - 制限事項, 4125
- EVENTS
 - INFORMATION_SCHEMA テーブル, 4109, 4144
- events オプション
 - mysqldump, 434
 - mysqlpump, 456
- events_errors_summary_by_account_by_error テーブル
 - performance_schema, 4394
- events_errors_summary_by_host_by_error テーブル
 - performance_schema, 4394
- events_errors_summary_by_thread_by_error テーブル
 - performance_schema, 4394
- events_errors_summary_by_user_by_error テーブル
 - performance_schema, 4394
- events_errors_summary_global_by_error テーブル
 - performance_schema, 4394
- events_stages_current テーブル
 - performance_schema, 4303
- events_stages_history テーブル
 - performance_schema, 4304
- events_stages_history_long テーブル
 - performance_schema, 4304
- events_stages_summary_by_account_by_event_name テーブル
 - performance_schema, 4375
- events_stages_summary_by_host_by_event_name テーブル
 - performance_schema, 4375
- events_stages_summary_by_thread_by_event_name テーブル
 - performance_schema, 4375
- events_stages_summary_by_user_by_event_name テーブル
 - performance_schema, 4375
- events_stages_summary_global_by_event_name テーブル
 - performance_schema, 4375

events_statements_current テーブル
performance_schema, 4308

events_statements_histogram_by_digest テーブル
performance_schema, 4380

events_statements_histogram_global テーブル
performance_schema, 4380

events_statements_history テーブル
performance_schema, 4312

events_statements_history_long テーブル
performance_schema, 4312

events_statements_summary_by_account_by_event_name テーブル
performance_schema, 4377

events_statements_summary_by_digest テーブル
performance_schema, 4377

events_statements_summary_by_host_by_event_name テーブル
performance_schema, 4377

events_statements_summary_by_program テーブル
performance_schema, 4377

events_statements_summary_by_thread_by_event_name テーブル
performance_schema, 4377

events_statements_summary_by_user_by_event_name テーブル
performance_schema, 4377

events_statements_summary_global_by_event_name テーブル
performance_schema, 4377

events_transactions_current テーブル
performance_schema, 4319

events_transactions_history テーブル
performance_schema, 4321

events_transactions_history_long テーブル
performance_schema, 4321

events_transactions_summary_by_account_by_event テーブル
performance_schema, 4382

events_transactions_summary_by_host_by_event_name テーブル
performance_schema, 4382

events_transactions_summary_by_thread_by_event_name テーブル
performance_schema, 4382

events_transactions_summary_by_user_by_event_name テーブル
performance_schema, 4382

events_transactions_summary_global_by_event_name テーブル
performance_schema, 4382

events_waits_current テーブル
performance_schema, 4296

events_waits_history テーブル
performance_schema, 4299

events_waits_history_long テーブル
performance_schema, 4299

events_waits_summary_by_account_by_event_name テーブル
performance_schema, 4374

events_waits_summary_by_host_by_event_name テーブル
performance_schema, 4374

events_waits_summary_by_instance テーブル
performance_schema, 4374

events_waits_summary_by_thread_by_event_name テーブル
performance_schema, 4374

events_waits_summary_by_user_by_event_name テーブル
performance_schema, 4374

events_waits_summary_global_by_event_name テーブル
performance_schema, 4374

event_scheduler システム変数, 701

EXAMPLE ストレージエンジン, 2983, 3014

- exclude-databases オプション
 - mysqlpump, 456
 - ndb_restore, 3795
- exclude-events オプション
 - mysqlpump, 457
- exclude-gtids オプション
 - mysqlbinlog, 529
- exclude-intermediate-sql-tables オプション
 - ndb_restore, 3796
- exclude-missing-columns オプション
 - ndb_move_data, 3781
 - ndb_restore, 3796
- exclude-missing-tables オプション
 - ndb_restore, 3796
- exclude-routines オプション
 - mysqlpump, 457
- exclude-tables オプション
 - mysqlpump, 457
 - ndb_restore, 3796
- exclude-trigger オプション
 - mysqlpump, 457
- exclude-users オプション
 - mysqlpump, 457
- Execute
 - スレッドのコマンド, 1615
- EXECUTE, 2437, 2443
- EXECUTE 権限, 1049
- ExecuteOnComputer, 3570, 3576, 3649
- execute_prepared_stmt() プロシージャ
 - sys スキーマ, 4493
- executing
 - スレッドの状態, 1617
- Execution of init_command
 - スレッドの状態, 1617
- EXISTS
 - サブクエリーを使用した, 2349
- EXIT SINGLE USER MODE コマンド (NDB Cluster), 3841
- EXIT コマンド (NDB Cluster), 3841
- exit コマンド
 - mysql, 381
- exit-info オプション
 - mysqld, 650
- EXP(), 1877
- expire_logs_days システム変数, 3131
- EXPLAIN ANALYZE
 - 新機能, 28
- EXPLAIN, 1539, 2611, 4072, 4073
 - ウィンドウ関数, 1478
- EXPLAIN EXTENDED
 - 削除された機能, 39
- EXPLAIN PARTITIONS
 - 削除された機能, 39
- EXPLAIN がパーティション化されたテーブルで使用される, 4072
- explicit_defaults_for_timestamp システム変数, 702
- EXPORT_SET(), 1905
- extend-check オプション
 - myisamchk, 498, 499
- extended-insert オプション
 - mysqldump, 435
 - mysqlpump, 457

ExteriorRing()
削除された機能, 40
external-locking オプション
mysqld, 650
external_user システム変数, 703
extra-file オプション
my_print_defaults, 548
extra-node-info オプション
ndb_desc, 3760
extra-partition-info オプション
ndb_desc, 3760
EXTRACT(), 1889
ExtractValue(), 1967
extract_schema_from_file_name() 関数
sys スキーマ, 4509
extract_table_from_file_name() 関数
sys スキーマ, 4509
ExtraSendBufferMemory
API ノード, 3652
管理ノード, 3574
データノード, 3644

F

failover, 5340
failover
Java クライアント, 3451
NDB Cluster レプリケーション, 4006
FALSE, 1630, 1635
テスト, 1864, 1865
false リテラル
JSON, 1811
FAQ
コネクタおよび API, 4573
レプリケーション, 4574
FAQs
C API, 4573
InnoDB 保存データ暗号化, 4580
libmysql, 4573
NDB Cluster, 4551
仮想化のサポート, 4582
FEDERATED ストレージエンジン, 2983, 3009
Fetch
スレッドのコマンド, 1615
FETCH, 2451
Field List
スレッドのコマンド, 1615
FIELD(), 1905
fields-enclosed-by オプション
mysqldump, 431, 443
ndb_import, 3768
ndb_restore, 3797
fields-escaped-by オプション
mysqldump, 431, 443
ndb_import, 3768
fields-optionally-enclosed-by オプション
mysqldump, 431, 443
ndb_import, 3768
ndb_restore, 3797
fields-terminated-by オプション

- mysqldump, 431, 443
- ndb_import, 3768
- ndb_restore, 3797
- FILE, 1907
- FILE 権限, 1049
- file-per-table, 5340
- file-per-table, 2646
- files
 - CREATE TABLE による作成, 2243
 - サイズ制限, 1521
- FILES
 - INFORMATION_SCHEMA テーブル, 4147
- filesort の最適化, 1469, 1583
- FileSystemPath, 3579
- FileSystemPathDataFiles, 3641
- FileSystemPathDD, 3641
- FileSystemPathUndoFiles, 3641
- FILE_FORMAT
 - 削除された機能, 44
- file_instances テーブル
 - performance_schema, 4290
- file_summary_by_event_name テーブル
 - performance_schema, 4385
- file_summary_by_instance テーブル
 - performance_schema, 4385
- fill_help_tables.sql, 889
- FIND_IN_SET(), 1905
- Finished reading one binlog; switching to next binlog
 - スレッドの状態, 1622
- FIPS モード, 1401
- firewall (ソフトウェア)
 - NDB Cluster, 3984, 3986
- Firewall_access_denied ステータス変数, 1371
- Firewall_access_granted ステータス変数, 1371
- Firewall_access_suspicious ステータス変数, 1371
- FIREWALL_ADMIN 権限, 1056
- Firewall_cached_entries ステータス変数, 1371
- firewall_groups MySQL Enterprise Firewall テーブル, 1364
- firewall_groups テーブル
 - performance_schema, 4400
 - システムテーブル, 899
- firewall_group_allowlist MySQL Enterprise Firewall テーブル, 1364
- firewall_group_allowlist テーブル
 - performance_schema, 4400
 - システムテーブル, 899
- firewall_group_delist() MySQL Enterprise Firewall UDF, 1369
- firewall_group_enlist() MySQL Enterprise Firewall UDF, 1369
- firewall_membership MySQL Enterprise Firewall テーブル, 1364
- firewall_membership テーブル
 - performance_schema, 4401
 - システムテーブル, 899
- FIREWALL_USER 権限, 1056
- firewall_users MySQL Enterprise Firewall テーブル, 1363
- firewall_users テーブル
 - システムテーブル, 899
- firewall_whitelist MySQL Enterprise Firewall テーブル, 1363
- firewall_whitelist テーブル
 - システムテーブル, 899
- FirstMatch
 - 準結合方式, 1482

FIRST_VALUE(), 2111
FIXED データ型, 1764
FLOAT データ型, 1764, 1765, 1765
floats, 1630
FLOOR(), 1877
FLUSH
 およびレプリケーション, 3219
FLUSH QUERY CACHE
 削除された機能, 39
flush システム変数, 704
FLUSH ステートメント, 2600
flush-logs オプション
 mysqldump, 436
flush-privileges オプション
 mysqldump, 436
Flush_commands ステータス変数, 840
FLUSH_OPTIMIZER_COSTS 権限, 1056
FLUSH_STATUS 権限, 1056
FLUSH_TABLES 権限, 1056
flush_time システム変数, 704
FLUSH_USER_RESOURCES 権限, 1056
FOR SHARE, 2330
FOR UPDATE, 2330
FORCE
 プラグインアクティベーションオプション, 949
FORCE INDEX, 1579, 4612
FORCE KEY, 1579
force オプション
 myisamchk, 499
force-if-open オプション
 mysqlbinlog, 529
force-read オプション
 mysqlbinlog, 529
FORCE_INSOURCE_BUILD オプション
 CMake, 200
FORCE_PLUS_PERMANENT
 プラグインアクティベーションオプション, 949
FORCE_UNSUPPORTED_COMPILER オプション
 CMake, 207
FOREIGN KEY 制約, 5340
foreign_keys テーブル
 データディクショナリテーブル, 896
foreign_key_checks システム変数, 704
foreign_key_column_usage テーブル
 データディクショナリテーブル, 896
FORMAT(), 1905
FORMAT_BYTES() 関数, 2125
format_bytes() 関数
 sys スキーマ, 4509
format_path() 関数
 sys スキーマ, 4510
FORMAT_PICO_TIME() 関数, 2125
format_statement() 関数
 sys スキーマ, 4510
format_time() 関数
 sys スキーマ, 4511
formfeed (lf), 2070
FOUND_ROWS(), 1996
 非推奨となった機能, 36
FPROFILE_GENERATE オプション

- CMake, 207
- FPROFILE_USE オプション
 - CMake, 207
- FragmentLogFileSize, 3593
- FRAGMENT_COUNT_TYPE (NDB_TABLE) (OBSOLETE)
 - NDB Cluster, 2267
- frame
 - ウィンドウ関数, 2119, 2120
- FreeBSD トラブルシューティング, 219
- freeing items
 - スレッドの状態, 1618
- FROM, 2327
- FROM_BASE64(), 1905
- FROM_DAYS(), 1889
- FROM_UNIXTIME(), 1889
- fs オプション
 - ndb_error_reporter, 3762
- FTS, 5340
- ft_boolean_syntax システム変数, 705
- ft_max_word_len myisamchk 変数, 497
- ft_max_word_len システム変数, 705
- ft_min_word_len myisamchk 変数, 497
- ft_min_word_len システム変数, 706
- ft_query_expansion_limit システム変数, 706
- ft_stopword_file myisamchk 変数, 497
- ft_stopword_file システム変数, 706
- FULLTEXT, 1927
- FULLTEXT initialization
 - スレッドの状態, 1618
- FULLTEXT インデックス
 - InnoDB, 2667
 - 監視, 2671
- FULLTEXT インデックス, 5340
- fulltext 結合タイプ
 - オプティマイザ, 1544
- FULLY_REPLICATED (NDB_TABLE)
 - NDB Cluster, 2267
- func table
 - システムテーブル, 898, 1009
- function
 - 削除, 2537
 - 作成, 2536
- functions
 - aggregate, 2090
 - bit, 1975
 - GROUP BY, 2090
 - internal, 2127
 - stored, 4097
 - string, 1901
 - user-defined, 2536, 2537
 - 算術, 1975
 - パフォーマンススキーマ, 2124
 - 日付と時刻, 1883
 - フロー制御, 1869
 - 文字列の比較, 1914
 - ロック, 1990

G

GA, 5340

GA
 MySQL のリリース, 84
GAC, 5340
gb2312、GBK, 4563
gci オプション
 ndb_select_all, 3816
gci64 オプション
 ndb_select_all, 3816
GCP 停止エラー (NDB Cluster), 3644
gdb
 使用, 1018
gdb オプション
 mysqld, 651
Gemalto SafeNet KeySecure アプリケーション
 keyring_okv キーリングプラグイン, 1239
General Public License, 4
general_log システム変数, 707
general_log テーブル
 システムテーブル, 898
general_log_file システム変数, 707
generated_random_password_length システム変数, 707
gen_blacklist() MySQL Enterprise Data Masking and De-Identification UDF, 1385
gen_blocklist() MySQL Enterprise Data Masking and De-Identification UDF, 1386
gen_dictionary() MySQL Enterprise Data Masking and De-Identification UDF, 1386
gen_dictionary_drop() MySQL Enterprise Data Masking and De-Identification UDF, 1387
gen_dictionary_load() MySQL Enterprise Data Masking and De-Identification UDF, 1388
gen_range() MySQL Enterprise Data Masking and De-Identification UDF, 1383
gen_rnd_email() MySQL Enterprise Data Masking and De-Identification UDF, 1384
gen_rnd_pan() MySQL Enterprise Data Masking and De-Identification UDF, 1384
gen_rnd_ssn() MySQL Enterprise Data Masking and De-Identification UDF, 1385
gen_rnd_us_phone() MySQL Enterprise Data Masking and De-Identification UDF, 1385
GeomCollection(), 2011
GeomCollFromText()
 削除された機能, 40
GeomCollFromWKB()
 削除された機能, 40
GEOMETRY データ型, 1794
GEOMETRYCOLLECTION データ型, 1794
GeometryCollection(), 2011
GeometryCollectionFromText()
 削除された機能, 40
GeometryCollectionFromWKB()
 削除された機能, 40
GeometryFromText()
 削除された機能, 40
GeometryFromWKB()
 削除された機能, 40
GeometryN()
 削除された機能, 40
GeometryType()
 削除された機能, 40
GeomFromText()
 削除された機能, 40
GeomFromWKB()
 削除された機能, 40
GET DIAGNOSTICS, 2456
get-server-public-key オプション, 314
 mysql, 372
 mysqladmin, 400
 mysqlbinlog, 529

- mysqlcheck, 410
- mysqldump, 422
- mysqlimport, 444
- mysqlpump, 457
- mysqlshow, 469
- mysqlslap, 479
- mysql_upgrade, 359
- GET_DD_COLUMN_PRIVILEGES(), 2128
- GET_DD_CREATE_OPTIONS(), 2128
- GET_DD_INDEX_SUB_PART_LENGTH(), 2128
- GET_FORMAT(), 1890
- GET_LOCK(), 1991
- GIS, 1793
- GIS データ型
 - 記憶域要件, 1830
- GIS 値
 - 形状的に有効, 1803
- Git ツリー, 189
- Glassfish, 5340
- GLength()
 - 削除された機能, 40
- GLOBAL
 - SET ステートメント, 2542
- global_grants テーブル
 - システムテーブル, 897, 1061, 1063
- GLOBAL_STATUS
 - 削除された機能, 41
- GLOBAL_VARIABLES
 - 削除された機能, 41
- go コマンド
 - mysql, 381
- Google テスト, 206
- GRANT
 - 削除された機能, 38
- GRANT OPTION 権限, 1050
- GRANT ステートメント, 1077, 2502
 - 権限の制限, 2512
- grants
 - display, 2565
- GREATEST(), 1863
- GROUP BY
 - WITH ROLLUP, 2098
 - 暗黙的ソート, 1469
 - エイリアス, 2106
 - 最大ソート長, 2328
 - 標準 SQL の拡張機能, 2104, 2328
- GROUP BY 関数, 2090
- GROUP BY 最適化, 1470
- GROUP BY ソート
 - 削除された機能, 39
- GROUPING(), 2098, 2131
- GROUP_CONCAT(), 2093
- group_concat_max_len システム変数, 708
- GROUP_INDEX, 1572
- GROUP_REPLICATION_ADMIN 権限, 1056
- group_replication_advertise_recovery_endpoints, 3329
- group_replication_allow_local_lower_version_join システム変数, 3330
- group_replication_autorejoin_tries システム変数, 3332
- group_replication_auto_increment_increment システム変数, 3331
- group_replication_bootstrap_group システム変数, 3332

group_replication_clone_threshold システム変数, 3333
group_replication_communication_debug_options システム変数, 3334
group_replication_communication_max_message_size システム変数, 3334
group_replication_components_stop_timeout システム変数, 3335
group_replication_compression_threshold システム変数, 3335
group_replication_consistency システム変数, 3336
group_replication_enforce_update_everywhere_checks システム変数, 3337
group_replication_exit_state_action システム変数, 3338
group_replication_flow_control_applier_threshold システム変数, 3339
group_replication_flow_control_certifier_threshold システム変数, 3339
group_replication_flow_control_hold_percent システム変数, 3340
group_replication_flow_control_max_commit_quota システム変数, 3340
group_replication_flow_control_member_quota_percent システム変数, 3341
group_replication_flow_control_min_quota システム変数, 3341
group_replication_flow_control_min_recovery_quota システム変数, 3341
group_replication_flow_control_mode システム変数, 3342
group_replication_flow_control_period システム変数, 3342
group_replication_flow_control_release_percent システム変数, 3342
group_replication_force_members システム変数, 3343
group_replication_get_communication_protocol() UDF, 2436
group_replication_get_write_concurrency() UDF, 2435, 3271
group_replication_group_name システム変数, 3343
group_replication_group_seeds システム変数, 3344
group_replication_gtid_assignment_block_size システム変数, 3345
group_replication_ip_allowlist, 3345
group_replication_ip_whitelist, 3346
group_replication_local_address システム変数, 3347
group_replication_member_expel_timeout システム変数, 3348
group_replication_member_weight システム変数, 3349
group_replication_message_cache_size システム変数, 3349
group_replication_poll_spin_loops システム変数, 3350
group_replication_primary_member ステータス変数, 840
group_replication_recovery_complete_at システム変数, 3350
group_replication_recovery_compression_algorithm システム変数, 3351
group_replication_recovery_get_public_key システム変数, 3351
group_replication_recovery_public_key_path システム変数, 3352
group_replication_recovery_reconnect_interval システム変数, 3352
group_replication_recovery_retry_count システム変数, 3353
group_replication_recovery_ssl_ca システム変数, 3353
group_replication_recovery_ssl_capath システム変数, 3353
group_replication_recovery_ssl_cert システム変数, 3354
group_replication_recovery_ssl_cipher システム変数, 3354
group_replication_recovery_ssl_crl システム変数, 3354
group_replication_recovery_ssl_crlpath システム変数, 3355
group_replication_recovery_ssl_key システム変数, 3355
group_replication_recovery_ssl_verify_server_cert システム変数, 3355
group_replication_recovery_tls_ciphersuites システム変数, 3356
group_replication_recovery_tls_version システム変数, 3356
group_replication_recovery_use_ssl システム変数, 3357
group_replication_recovery_zstd_compression_level システム変数, 3357
group_replication_set_as_primary() UDF, 2434, 3270
group_replication_set_communication_protocol() UDF, 2437
group_replication_set_write_concurrency() UDF, 2436, 3271
group_replication_single_primary_mode システム変数, 3357
group_replication_ssl_mode システム変数, 3358
group_replication_start_on_boot システム変数, 3358
group_replication_switch_to_multi_primary_mode() UDF, 2435, 3271
group_replication_switch_to_single_primary_mode() UDF, 2435, 3270
group_replication_tls_source システム変数, 3359
group_replication_transaction_size_limit システム変数, 3359

- group_replication_unreachable_majority_timeout, 3360
- GSSAPI 認証方式
 - LDAP 認証, 1188, 1188
- GTID 関数, 2088
- GTID セット
 - 表現, 3034
- GTIDs, 3032
 - auto-positioning, 3041
 - gtid_purged, 3039
 - logging, 3035
 - 概念, 3033
 - スケールアウト, 3044
 - 制限事項, 3048
 - トランザクションへの割当て, 3047
 - フェイルオーバー, 3044
 - ライフサイクル, 3037
 - レプリケーション, 3042
- gtid_executed システム変数, 3140
- gtid_executed テーブル
 - システムテーブル, 899, 3035
- gtid_executed_compression_period, 3140
- gtid_executed_compression_period システム変数
 - mysql.gtid_executed テーブル, 3036
- gtid_mode システム変数, 3141
- gtid_next システム変数, 3142
- gtid_owned システム変数, 3142
- gtid_purged, 3039
- gtid_purged システム変数, 3143
- GTID_SUBSET(), 2088
- GTID_SUBTRACT(), 2089
- GUID, 5341

H

- HANDLER, 2295
- Handler_commit ステータス変数, 840
- Handler_delete ステータス変数, 840
- Handler_discover ステータス変数, 3689
- Handler_external_lock ステータス変数, 840
- Handler_mrr_init ステータス変数, 840
- Handler_prepare ステータス変数, 840
- Handler_read_first ステータス変数, 840
- Handler_read_key ステータス変数, 841
- Handler_read_last ステータス変数, 841
- Handler_read_next ステータス変数, 841
- Handler_read_prev ステータス変数, 841
- Handler_read_rnd ステータス変数, 841
- Handler_read_rnd_next ステータス変数, 841
- Handler_rollback ステータス変数, 841
- Handler_savepoint ステータス変数, 841
- Handler_savepoint_rollback ステータス変数, 841
- Handler_update ステータス変数, 841
- Handler_write ステータス変数, 841
- HASH パーティショニング, 4047
- HASH パーティション
 - 管理, 4063
 - 分割およびマージ, 4063
- HashiCorp Vault
 - 構成, 1246
- HashiCorp Vault の証明書およびキーファイル

- 構成, 1244
- have_compress システム変数, 708
- have_crypt
 - 削除された機能, 40
- HAVE_CRYPT
 - 削除された機能, 40
- have_dynamic_loading システム変数, 708
- have_geometry システム変数, 708
- have_openssl システム変数, 708
- have_profiling システム変数, 708
- have_query_cache システム変数, 708
- have_rtree_keys システム変数, 708
- have_ssl システム変数, 709
- have_statement_timeout システム変数, 709
- have_symlink システム変数, 709
- HAVING, 2328
- HDD, 5341
- header-file オプション
 - comp_err, 346
- HEAP から ondisk への変換
 - スレッドの状態, 1616
- HEAP ストレージエンジン, 2983, 2995
- HeartbeatIntervalDbApi, 3607
- HeartbeatIntervalDbDb, 3607
- HeartbeatIntervalMgmdMgmd
 - 管理ノード, 3574
- HeartbeatOrder, 3608
- HeartbeatThreadPriority, 3574, 3652
- help オプション (NDB Cluster プログラム), 3836
- HELP オプション
 - mysamchk, 495
- HELP コマンド (NDB Cluster), 3839
- help コマンド
 - mysql, 380
- HELP ステートメント, 2614
- help_category テーブル
 - システムテーブル, 899
- help_keyword テーブル
 - システムテーブル, 899
- help_relation テーブル
 - システムテーブル, 899
- help_topic テーブル
 - システムテーブル, 899
- hex オプション
 - ndb_restore, 3798
- HEX(), 1878, 1906
- hex-blob オプション
 - mysqldump, 431
 - mysqlpump, 457
- hexdump オプション
 - mysqlbinlog, 529
- HIGH_NOT_PRECEDENCE SQL モード, 856
- HIGH_PRIORITY
 - INSERT 修飾子, 2303
 - SELECT 修飾子, 2331
- hintable
 - システム変数, 1576
- histignore オプション
 - mysql, 372
- histogram_generation_max_mem_size システム変数, 709

HOME 環境変数, 386, 550
HostName (NDB Cluster), 3983
HostName, 3570, 3576, 3650
hostname システム変数, 710
HostName1, 3710, 3716
HostName2, 3710, 3717
hosts テーブル
 performance_schema, 4324
host_cache テーブル
 performance_schema, 868, 4401
host_summary ビュー
 sys スキーマ, 4454
host_summary_by_file_io ビュー
 sys スキーマ, 4455
host_summary_by_file_io_type ビュー
 sys スキーマ, 4455
host_summary_by_stages ビュー
 sys スキーマ, 4455
host_summary_by_statement_latency ビュー
 sys スキーマ, 4456
host_summary_by_statement_type ビュー
 sys スキーマ, 4456
HOUR(), 1891
html オプション
 mysql, 372
hwinfo
 ndbinfo テーブル, 3942

I

i-am-a-dummy オプション
 mysql, 376
ib-file セット, 5341
ibbackup_logfile, 5341
ibd2sdi, 483
 id オプション, 484
 no-check オプション, 486
 pretty オプション, 486
 skip-data オプション, 484
 strict-check オプション, 486
 カウントオプション, 484
 タイプオプション, 485
 デバッグオプション, 484
 バージョンオプション, 484
 ヘルプオプション, 483
ibdata ファイル, 2243, 5341
ibtmp ファイル, 5341
ib_logfile, 5341
icc
 MySQL ビルド, 97
ICU_VERSION(), 1998
Id, 3569, 3648
id オプション
 ibd2sdi, 484
idempotent オプション
 mysqlbinlog, 529
IDENTIFIED BY PASSWORD
 削除された機能, 38
idlesleep オプション
 ndb_import, 3769

idlespin オプション
 ndb_import, 3769
IF, 2447
IF(), 1871
IFNULL(), 1871
IGNORE
 DELETE 修飾子, 2293, 2309
 INSERT 修飾子, 2303
 UPDATE 修飾子, 2362
 パーティション化されたテーブル, 2303
 パーティションテーブル, 863
IGNORE INDEX, 1579
IGNORE KEY, 1579
ignore オプション
 mysqlimport, 444
ignore-error オプション
 mysqldump, 434
ignore-extended-pk-updates オプション
 ndb_restore, 3798
ignore-lines オプション
 mysqlimport, 444
 ndb_import, 3769
ignore-spaces オプション
 mysql, 372
ignore-table オプション
 mysqldump, 434
IGNORE_AIO_CHECK オプション
 CMake, 207
ignore_builtin_innodb
 削除された機能, 40
ignore_db_dirs
 削除された機能, 39
IGNORE_SPACE SQL モード, 856
ilist, 5342
immediate_commit_timestamp, 3209
immediate_server_version システム変数, 3076
IMPORT TABLE, 2297
IMPORT TABLESPACE, 2178, 2653
IN, 2346
IN(), 1864
in-file-errlog オプション
 comp_err, 347
in-file-toclient オプション
 comp_err, 347
include オプション
 mysql_config, 546
include-databases オプション
 mysqlpump, 457
 ndb_restore, 3798
include-events オプション
 mysqlpump, 457
include-gtids オプション
 mysqlbinlog, 530
include-master-host-port オプション
 mysqldump, 429
include-routines オプション
 mysqlpump, 458
include-stored-grants オプション
 ndb_restore, 3798
include-tables オプション

- mysqlpump, 458
- ndb_restore, 3798
- include-trigger オプション
 - mysqlpump, 458
- include-users オプション
 - mysqlpump, 458
- INDEX DIRECTORY
 - およびレプリケーション, 3218
- index dives (統計推定用), 2753
- index dives
 - 範囲の最適化, 1438
- INDEX, 1572
- index
 - ソートされたインデックス構築, 2666
- INDEX 権限, 1050
- IndexMemory, 3581
- IndexStatAutoCreate
 - データノード, 3646
- IndexStatAutoUpdate
 - データノード, 3646
- IndexStatSaveScale
 - データノード, 3647
- IndexStatSaveSize
 - データノード, 3647
- IndexStatTriggerPct
 - データノード, 3647
- IndexStatTriggerScale
 - データノード, 3647
- IndexStatUpdateDelay
 - データノード, 3648

- index_column_usage テーブル
 - データディクショナリテーブル, 896
- INDEX_MERGE, 1572
- index_merge 結合タイプ
 - オプティマイザ, 1544
- index_partitions テーブル
 - データディクショナリテーブル, 896
- index_stats テーブル
 - データディクショナリテーブル, 896
- index_subquery 結合タイプ
 - オプティマイザ, 1545

- INET6_ATON(), 2134
- INET6_NTOA(), 2135
- INET_ATON(), 2134
- INET_NTOA(), 2134
- INFO イベント (NDB Cluster), 3866

- info オプション
 - innochecksum, 487
 - ndbxfrm, 3833

- INFORMATION_SCHEMA, 5342
- INFORMATION_SCHEMA, 4132
 - InnoDB テーブル, 4191
 - INNODB_CMP テーブル, 2895
 - INNODB_CMPMEM テーブル, 2895
 - INNODB_CMPMEM_RESET テーブル, 2895
 - INNODB_CMP_RESET テーブル, 2895
 - INNODB_TRX テーブル, 2896
 - MySQL Enterprise Firewall テーブル, 4233
 - 照合順序と検索, 1716
 - スレッドプールテーブル, 4231

- セキュリティの問題, 3988
- 接続制御テーブル, 4233
- INFORMATION_SCHEMA クエリー
 - 最適化, 1492
- INFORMATION_SCHEMA.ENGINES テーブル
 - NDB Cluster, 3977
- INFORMATION_SCHEMA.PLUGINS テーブル
 - NDB Cluster, 3982
- information_schema_stats
 - 削除された機能, 38
- information_schema_stats_expiry システム変数, 711
- INFO_BIN ファイル
 - バイナリ配布構成オプション, 64, 183
- Init DB
 - スレッドのコマンド, 1615
- init
 - スレッドの状態, 1618
- init-command オプション
 - mysql, 372
- InitFragmentLogFiles, 3594
- initial-start オプション
 - ndbd, 3728
 - ndbmt, 3728
- initialize-insecure オプション
 - mysqld, 651
- Initialized
 - スレッドの状態, 1625
- InitialLogFileGroup, 3642
- InitialNoOfOpenFiles, 3594
- InitialTablespace, 3643
- init_connect システム変数, 711
- init_file システム変数, 712
- init_slave システム変数, 3091
- INNER JOIN, 2334
- innochecksum, 298, 486
 - allow-mismatches オプション, 488
 - end-page オプション, 487
 - info オプション, 487
 - log オプション, 489
 - no-check オプション, 488
 - page-type-dump オプション, 489
 - page-type-summary オプション, 489
 - read from standard in オプション, 490
 - start-page オプション, 487
 - strict-check オプション, 488
 - verbose オプション, 487
 - write オプション, 488
 - カウントオプション, 487
 - バージョンオプション, 487
 - ページオプション, 488
 - ヘルプオプション, 486
- InnoDB, 5342
- InnoDB, 2626
 - auto-inc ロック, 2701
 - COMPACT 行形式, 2779
 - DYNAMIC 行形式, 2780, 2781
 - file-per-table テーブルスペース, 2675
 - files, 2658
 - FULLTEXT インデックス, 2667
 - insert-intention ロック, 2701

limits, 2980
Linux, 2742
NDB Cluster との比較, 3479, 3479, 3480, 3481
NFS, 2722
point-in-time リカバリ, 2939
RAW パーティション, 2674
recovery, 2938
redo ログ, 2695, 2695
REDUNDANT 行形式, 2778
Solaris の問題, 180
storage, 2657
transactions, 2655
アーキテクチャ, 2632
アプリケーション機能の要件, 3481
アプリケーションパフォーマンス, 2658
一貫性読み取り, 2709
インデックスレコードロック, 2717
インテンションロック, 2701
インメモリ構造, 2633
オンライン DDL, 2786
可用性, 3479
記憶域レイアウト, 2656
ギャップロック, 2701, 2717
共有ロック, 2701
行フォーマット, 2642, 2782
クラスタ化されたインデックス, 2665
クラッシュリカバリ, 2939, 2939, 2941
構成パラメータ, 2813
サポートされるアプリケーション, 3480
システム変数, 2813
自動インクリメントカラム, 2658
自動コミットモード, 2708, 2708
主キー, 2643, 2657
新機能, 11
制限事項, 2981, 2981
セカンダリインデックス, 2665
ソートされたインデックス構築, 2666
チェックポイント, 2785
ディスク I/O 最適化, 1530
ディスク I/O, 2783
ディスク障害, 2939
ディスク上の構造, 2642
データの転送, 2656
データファイル, 2672
テーブル, 2642
 その他のストレージエンジンからの変換, 2654
テーブルの移行, 2652
テーブルの作成, 2642
テーブルのプロパティ, 2643
適応型ハッシュインデックス, 2641
デッドロック, 2656, 2717, 2718
デッドロック検出, 2718
デッドロックの例, 2718
トラブルシューティング, 2976
 I/O に関する問題, 2977
 SQL のエラー, 2980
 オンライン DDL, 2804
 孤立 ibd ファイルの復元, 2979
 データディクショナリに関する問題, 2979
 データファイルをオープンできません, 2979

- テーブルのデフラグ, 2785
- デッドロック, 2717, 2718
- パフォーマンスの問題, 1525
- リカバリに関する問題, 2977
- トランザクションモデル, 2700, 2705
- ネクストキーロック, 2701, 2717
- 排他ロック, 2701
- 破損, 2939
- バックアップ, 2938
- バッファプール, 2740
- 非同期 I/O, 2742
- ファイル領域管理, 2784
- 物理インデックス構造, 2666
- ページサイズ, 2666
- 変更バッファ, 2638
- マルチバージョン, 2631
- メモリー使用量, 2655
- モニター, 2976
- レコードレベルロック, 2717
- レプリケーション, 2941
- ロック, 2700, 2701, 2713
- ロックモード, 2701
- ロック読み取り, 2711
- InnoDB memcached プラグイン
 - 非推奨となった機能, 37
- InnoDB ReplicaSet
 - はじめに, 3443
- InnoDB 圧縮一時テーブル
 - 削除された機能, 43
- innodb オプション
 - mysqld, 2819
- InnoDB 共有テーブルスペース
 - 削除された機能, 44
- InnoDB クラスタ
 - はじめに, 3439
- InnoDB 述語ロック, 2705
- InnoDB ストレージエンジン, 2626, 2983
- InnoDB テーブル
 - 記憶域要件, 1827
- InnoDB バッファプール, 1587, 2728
- InnoDB バッファプール, 2635, 2733, 2733, 2734, 2737
- InnoDB モニター, 2931
 - output, 2933
 - 有効化, 2932
- InnoDB リモートテーブルスペース
 - 削除された機能, 44
- innodb-status-file オプション
 - mysqld, 2820
- innodb_adaptive_flushing システム変数, 2822
- innodb_adaptive_flushing_lwm システム変数, 2823
- innodb_adaptive_hash_index
 - および innodb_thread_concurrency, 2741
- innodb_adaptive_hash_index システム変数, 2823
- innodb_adaptive_hash_index_parts 変数, 2823
- innodb_adaptive_max_sleep_delay システム変数, 2824
- innodb_api_bk_commit_interval システム変数, 2824
- innodb_api_disable_rowlock システム変数, 2825
- innodb_api_enable_binlog システム変数, 2825
- innodb_api_enable_mdlog システム変数, 2825
- innodb_api_trx_level システム変数, 2825

innodb_autoextend_increment システム変数, 2826
innodb_autoinc_lock_mode, 5342
innodb_autoinc_lock_mode システム変数, 2826
Innodb_available_undo_logs
削除された機能, 44
innodb_background_drop_list_empty システム変数, 2827
INNODB_BUFFER_PAGE
INFORMATION_SCHEMA テーブル, 4191
INNODB_BUFFER_PAGE_LRU
INFORMATION_SCHEMA テーブル, 4195
Innodb_buffer_pool_bytes_data ステータス変数, 842
Innodb_buffer_pool_bytes_dirty ステータス変数, 842
innodb_buffer_pool_chunk_size システム変数, 2827
innodb_buffer_pool_debug, 2828
innodb_buffer_pool_dump_at_shutdown システム変数, 2828
innodb_buffer_pool_dump_now システム変数, 2829
innodb_buffer_pool_dump_pct システム変数, 2829
Innodb_buffer_pool_dump_status ステータス変数, 841
innodb_buffer_pool_filename システム変数, 2829
innodb_buffer_pool_instances システム変数, 2830
innodb_buffer_pool_in_core_file オプション, 2740
innodb_buffer_pool_in_core_file システム変数, 2830
innodb_buffer_pool_load_abort システム変数, 2831
innodb_buffer_pool_load_at_startup システム変数, 2831
innodb_buffer_pool_load_now システム変数, 2831
Innodb_buffer_pool_load_status ステータス変数, 841
Innodb_buffer_pool_pages_data ステータス変数, 842
Innodb_buffer_pool_pages_dirty ステータス変数, 842
Innodb_buffer_pool_pages_flushed ステータス変数, 842
Innodb_buffer_pool_pages_free ステータス変数, 842
Innodb_buffer_pool_pages_latched ステータス変数, 842
Innodb_buffer_pool_pages_misc ステータス変数, 842
Innodb_buffer_pool_pages_total ステータス変数, 842
Innodb_buffer_pool_reads ステータス変数, 843
Innodb_buffer_pool_read_ahead ステータス変数, 842
Innodb_buffer_pool_read_ahead_evicted ステータス変数, 842
Innodb_buffer_pool_read_ahead_rnd ステータス変数, 842
Innodb_buffer_pool_read_requests ステータス変数, 843
Innodb_buffer_pool_resize_status ステータス変数, 843
innodb_buffer_pool_size システム変数, 2832
INNODB_BUFFER_POOL_STATS
INFORMATION_SCHEMA テーブル, 4198
Innodb_buffer_pool_wait_free ステータス変数, 843
Innodb_buffer_pool_write_requests ステータス変数, 843
innodb_buffer_stats_by_schema ビュー
sys スキーマ, 4457
innodb_buffer_stats_by_table ビュー
sys スキーマ, 4458
INNODB_CACHED_INDEXES
INFORMATION_SCHEMA テーブル, 4201
innodb_change_buffering, 2639
innodb_change_buffering システム変数, 2833
innodb_change_buffering_debug, 2834
innodb_change_buffer_max_size システム変数, 2833
innodb_checkpoint_disabled システム変数, 2834
innodb_checksum_algorithm システム変数, 2835
INNODB_CMP
INFORMATION_SCHEMA テーブル, 4201
INNODB_CMPMEM
INFORMATION_SCHEMA テーブル, 4203

INNODB_CMPMEM_RESET
 INFORMATION_SCHEMA テーブル, 4203
INNODB_CMP_PER_INDEX
 INFORMATION_SCHEMA テーブル, 4204
innodb_cmp_per_index_enabled システム変数, 2836
INNODB_CMP_PER_INDEX_RESET
 INFORMATION_SCHEMA テーブル, 4204
INNODB_CMP_RESET
 INFORMATION_SCHEMA テーブル, 4202
INNODB_COLUMNS
 INFORMATION_SCHEMA テーブル, 4205
innodb_commit_concurrency システム変数, 2836
innodb_compression_failure_threshold_pct システム変数, 2837
innodb_compression_level システム変数, 2838
innodb_compression_pad_pct_max システム変数, 2838
innodb_compress_debug, 2837
innodb_concurrency_tickets, 2741
innodb_concurrency_tickets システム変数, 2838
INNODB_DATAFILES
 INFORMATION_SCHEMA テーブル, 4207
innodb_data_file_path システム変数, 2839
Innodb_data_fsyncs ステータス変数, 843
innodb_data_home_dir システム変数, 2840
Innodb_data_pending_fsyncs ステータス変数, 843
Innodb_data_pending_reads ステータス変数, 843
Innodb_data_pending_writes ステータス変数, 843
Innodb_data_read ステータス変数, 843
Innodb_data_reads ステータス変数, 843
Innodb_data_writes ステータス変数, 843
Innodb_data_written ステータス変数, 843
Innodb_dblwr_pages_written ステータス変数, 843
Innodb_dblwr_writes ステータス変数, 844
innodb_ddl_log テーブル
 データディクショナリテーブル, 896
innodb_ddl_log_crash_reset_debug システム変数, 2840
innodb_deadlock_detect
 新機能, 11
innodb_deadlock_detect システム変数, 2840
innodb_dedicated_server システム変数, 2841
innodb_default_row_format, 2781
innodb_default_row_format システム変数, 2841
innodb_directories システム変数, 2842
innodb_disable_sort_file_cache システム変数, 2842
innodb_doublewrite システム変数, 2843
innodb_doublewrite_batch_size, 2843
innodb_doublewrite_dir, 2843
innodb_doublewrite_files, 2844
innodb_doublewrite_pages, 2844
innodb_dynamic_metadata テーブル
 システムテーブル, 900
innodb_extend_and_initialize, 2691
innodb_extend_and_initialize システム変数, 2844
innodb_fast_shutdown システム変数, 2845
INNODB_FIELDS
 INFORMATION_SCHEMA テーブル, 4207
innodb_file_format
 削除された機能, 44
innodb_file_format_check
 削除された機能, 44
innodb_file_format_max

削除された機能, 44
innodb_file_per_table, 5342
innodb_file_per_table, 2761
innodb_file_per_table システム変数, 2846
innodb_fill_factor システム変数, 2846
innodb_fil_make_page_dirty_debug, 2845
innodb_flushing_avg_loops システム変数, 2851
innodb_flush_log_at_timeout システム変数, 2847
innodb_flush_log_at_trx_commit システム変数, 2847
innodb_flush_method システム変数, 2848
innodb_flush_neighbors システム変数, 2850
innodb_flush_sync システム変数, 2850
innodb_force_load_corrupted システム変数, 2851
innodb_force_recovery システム変数, 2851
DROP TABLE, 2285
INNODB_FOREIGN
INFORMATION_SCHEMA テーブル, 4208
INNODB_FOREIGN_COLS
INFORMATION_SCHEMA テーブル, 4209
innodb_fsync_threshold システム変数, 2852
innodb_ft_aux_table システム変数, 2852
INNODB_FT_BEING_DELETED
INFORMATION_SCHEMA テーブル, 4209
innodb_ft_cache_size システム変数, 2853
INNODB_FT_CONFIG
INFORMATION_SCHEMA テーブル, 4210
INNODB_FT_DEFAULT_STOPWORD
INFORMATION_SCHEMA テーブル, 4211
INNODB_FT_DELETED
INFORMATION_SCHEMA テーブル, 4212
innodb_ft_enable_diag_print システム変数, 2853
innodb_ft_enable_stopword システム変数, 2854
INNODB_FT_INDEX_CACHE
INFORMATION_SCHEMA テーブル, 4213
INNODB_FT_INDEX_TABLE
INFORMATION_SCHEMA テーブル, 4214
innodb_ft_max_token_size システム変数, 2854
innodb_ft_min_token_size システム変数, 2854
innodb_ft_num_word_optimize システム変数, 2855
innodb_ft_result_cache_limit システム変数, 2855
innodb_ft_server_stopword_table システム変数, 2856
innodb_ft_sort_pll_degree システム変数, 2856
innodb_ft_total_cache_size システム変数, 2856
innodb_ft_user_stopword_table システム変数, 2857
Innodb_have_atomic_builtins ステータス変数, 844
innodb_idle_flush_pct システム変数, 2857
INNODB_INDEXES
INFORMATION_SCHEMA テーブル, 4215
innodb_index_stats テーブル
システムテーブル, 899, 2747
innodb_io_capacity, 2743
innodb_io_capacity システム変数, 2858
innodb_io_capacity_max システム変数, 2858
innodb_large_prefix
削除された機能, 44
innodb_limit_optimistic_insert_debug, 2858
INNODB_LOCKS
INFORMATION_SCHEMA テーブル, 4217
削除された機能, 43
innodb_locks_unsafe_for_binlog

削除された機能, 38
INNODB_LOCK_WAITS
 INFORMATION_SCHEMA テーブル, 4218
 削除された機能, 43
innodb_lock_waits ビュー
 sys スキーマ, 4459
innodb_lock_wait_timeout, 5343
innodb_lock_wait_timeout システム変数, 2859
innodb_log_buffer_size システム変数, 2859
innodb_log_checkpoint_fuzzy_now システム変数, 2860
innodb_log_checkpoint_now システム変数, 2860
innodb_log_checksums システム変数, 2860
innodb_log_compressed_pages システム変数, 2861
innodb_log_files_in_group システム変数, 2862
innodb_log_file_size システム変数, 2861
innodb_log_group_home_dir システム変数, 2862
innodb_log_spin_cpu_abs_lwm システム変数, 2862
innodb_log_spin_cpu_pct_hwm システム変数, 2863
InnoDB_log_waits ステータス変数, 844
innodb_log_wait_for_flush_spin_hwm システム変数, 2863
innodb_log_writer_threads システム変数, 2864
InnoDB_log_writes ステータス変数, 844
innodb_log_write_ahead_size システム変数, 2864
InnoDB_log_write_requests ステータス変数, 844
innodb_lru_scan_depth システム変数, 2864
innodb_max_dirty_pages_pct システム変数, 2865
innodb_max_dirty_pages_pct_lwm システム変数, 2866
innodb_max_purge_lag システム変数, 2866
innodb_max_purge_lag_delay システム変数, 2866
innodb_max_undo_log_size システム変数, 2867
innodb_memcache データベース, 2948, 2971
innodb_memcached_config.sql スクリプト, 2948
innodb_merge_threshold_set_all_debug, 2867
INNODB_METRICS
 INFORMATION_SCHEMA テーブル, 4218
innodb_monitor_disable システム変数, 2867
innodb_monitor_enable システム変数, 2867
innodb_monitor_reset システム変数, 2868
innodb_monitor_reset_all システム変数, 2868
innodb_numa_interleave 変数, 2869
InnoDB_num_open_files ステータス変数, 844
innodb_old_blocks_pct, 2733
innodb_old_blocks_pct システム変数, 2869
innodb_old_blocks_time, 2733
innodb_old_blocks_time システム変数, 2869
innodb_online_alter_log_max_size システム変数, 2870
innodb_open_files システム変数, 2870
innodb_optimize_fulltext_only システム変数, 2871
InnoDB_os_log_fsyncs ステータス変数, 844
InnoDB_os_log_pending_fsyncs ステータス変数, 844
InnoDB_os_log_pending_writes ステータス変数, 844
InnoDB_os_log_written ステータス変数, 844
InnoDB_pages_created ステータス変数, 844
InnoDB_pages_read ステータス変数, 844
InnoDB_pages_written ステータス変数, 844
innodb_page_cleaners システム変数, 2871
innodb_page_size システム変数, 2872
InnoDB_page_size ステータス変数, 844
innodb_parallel_read_threads システム変数, 2873
innodb_print_all_deadlocks システム変数, 2874

innodb_print_all_deadlocks, 2874
innodb_print_ddl_logs システム変数, 2874
innodb_purge_batch_size システム変数, 2874
innodb_purge_rseg_truncate_frequency システム変数, 2875
innodb_purge_threads システム変数, 2875
innodb_random_read_ahead システム変数, 2875
innodb_read_ahead_threshold, 2734
innodb_read_ahead_threshold システム変数, 2876
innodb_read_io_threads, 2742
innodb_read_io_threads システム変数, 2876
innodb_read_only システム変数, 2877
INNODB_REDO_LOG_ARCHIVE 権限, 1056
innodb_redo_log_archive_dirs システム変数, 2878
INNODB_REDO_LOG_ENABLE 権限, 1056
Innodb_redo_log_enabled ステータス変数, 844
innodb_redo_log_encrypt システム変数, 2878
innodb_replication_delay システム変数, 2878
innodb_rollback_on_timeout システム変数, 2879
innodb_rollback_segments システム変数, 2879
Innodb_rows_deleted ステータス変数, 845
Innodb_rows_inserted ステータス変数, 845
Innodb_rows_read ステータス変数, 845
Innodb_rows_updated ステータス変数, 845
Innodb_row_lock_current_waits ステータス変数, 844
Innodb_row_lock_time ステータス変数, 845
Innodb_row_lock_time_avg ステータス変数, 845
Innodb_row_lock_time_max ステータス変数, 845
Innodb_row_lock_waits ステータス変数, 845
innodb_saved_page_number_debug, 2879
INNODB_SESSION_TEMP_TABLESPACES
INFORMATION_SCHEMA テーブル, 4220
innodb_sort_buffer_size システム変数, 2880
innodb_spin_wait_delay, 2744
innodb_spin_wait_delay システム変数, 2880
innodb_spin_wait_pause_multiplier, 2744
innodb_spin_wait_pause_multiplier システム変数, 2881
innodb_stats_auto_recalc システム変数, 2881
innodb_stats_include_delete_marked システム変数, 2749, 2882
innodb_stats_method システム変数, 2882
innodb_stats_on_metadata システム変数, 2882
innodb_stats_persistent システム変数
innodb_stats_persistent, 2883
innodb_stats_persistent_sample_pages システム変数, 2883
innodb_stats_transient_sample_pages, 2753
innodb_stats_transient_sample_pages システム変数, 2884
innodb_status_output システム変数, 2884
innodb_status_output_locks システム変数, 2885
innodb_stat_persistent システム変数, 2883
innodb_strict_mode, 5343
innodb_strict_mode システム変数, 2885
innodb_support_xa
削除された機能, 44
innodb_sync_array_size システム変数, 2886
innodb_sync_debug, 2886
innodb_sync_spin_loops システム変数, 2886
Innodb_system_rows_deleted ステータス変数, 845
Innodb_system_rows_inserted ステータス変数, 845
Innodb_system_rows_read ステータス変数, 845
INNODB_SYS_COLUMNS
削除された機能, 38

INNODB_SYS_DATAFILES
削除された機能, 38

INNODB_SYS_FIELDS
削除された機能, 38

INNODB_SYS_FOREIGN
削除された機能, 38

INNODB_SYS_FOREIGN_COLS
削除された機能, 38

INNODB_SYS_INDEXES
削除された機能, 38

INNODB_SYS_TABLES
削除された機能, 38

INNODB_SYS_TABLESPACES
削除された機能, 38

INNODB_SYS_TABLESTATS
削除された機能, 38

INNODB_SYS_VIRTUAL
削除された機能, 38

INNODB_TABLES
INFORMATION_SCHEMA テーブル, 4221

INNODB_TABLESPACES
INFORMATION_SCHEMA テーブル, 4222

INNODB_TABLESPACES_BRIEF
INFORMATION_SCHEMA テーブル, 4224

INNODB_TABLESTATS
INFORMATION_SCHEMA テーブル, 4225

innodb_table_locks システム変数, 2886

innodb_table_stats テーブル
システムテーブル, 899, 2747

innodb_temp_data_file_path システム変数, 2887

innodb_temp_tablespaces_dir システム変数, 2888

INNODB_TEMP_TABLE_INFO
INFORMATION_SCHEMA テーブル, 4226

innodb_thread_concurrency, 2741

innodb_thread_concurrency システム変数, 2888

innodb_thread_sleep_delay, 2741

innodb_thread_sleep_delay システム変数, 2889

innodb_tmpdir システム変数, 2890

Innodb_truncated_status_writes ステータス変数, 845

INNODB_TRX
INFORMATION_SCHEMA テーブル, 4227

innodb_trx_purge_view_update_only_debug, 2891

innodb_trx_rseg_n_slots_debug, 2891

innodb_undo_directory システム変数, 2891

innodb_undo_logs
削除された機能, 44

innodb_undo_log_encrypt システム変数, 2892

innodb_undo_log_truncate システム変数, 2892

innodb_undo_tablespaces
削除された機能, 44

innodb_undo_tablespaces システム変数, 2892

Innodb_undo_tablespaces_active ステータス変数, 845

Innodb_undo_tablespaces_explicit ステータス変数, 845

Innodb_undo_tablespaces_implicit ステータス変数, 845

Innodb_undo_tablespaces_total ステータス変数, 846

innodb_use_native_aio, 2742

innodb_use_native_aio システム変数, 2893

innodb_validate_tablespace_paths システム変数, 2893

innodb_version システム変数, 2894

INNODB_VIRTUAL

- INFORMATION_SCHEMA テーブル, 4230
- innodb_write_io_threads, 2742
- innodb_write_io_threads システム変数, 2894
- INOUT パラメータ
 - 条件の処理, 2476
- input-type オプション
 - ndb_import, 3769
- input-workers オプション
 - ndb_import, 3770
- INSERT ... ON DUPLICATE KEY UPDATE
 - 非推奨となった機能, 37
- INSERT ... SELECT, 2303
- INSERT ... TABLE, 2303
- INSERT DELAYED, 2306, 2306
- INSERT, 1496, 2299
- INSERT 権限, 1050
- INSERT(), 1906
- insert-ignore オプション
 - mysqldump, 435
 - mysqlpump, 458
- insert-intention ロック, 2701
- InsertRecoveryWork, 3597
- insert_id システム変数, 713
- INSTALL COMPONENT ステートメント, 2538
- INSTALL PLUGIN ステートメント, 2538
- install-manual オプション
 - mysqld, 652
- INSTALL_BINDIR オプション
 - CMake, 200
- INSTALL_DOCDIR オプション
 - CMake, 200
- INSTALL_DOCREADMEDIR オプション
 - CMake, 200
- INSTALL_INCLUDEDIR オプション
 - CMake, 200
- INSTALL_INFODIR オプション
 - CMake, 201
- INSTALL_LAYOUT オプション
 - CMake, 201
- INSTALL_LIBDIR オプション
 - CMake, 201
- INSTALL_MANDIR オプション
 - CMake, 201
- INSTALL_MYSQLKEYRINGDIR オプション
 - CMake, 201
- INSTALL_MYSQLSHAREDIR オプション
 - CMake, 201
- INSTALL_MYSQLTESTDIR オプション
 - CMake, 201
- INSTALL_PKGCONFIGDIR オプション
 - CMake, 201
- INSTALL_PLUGINDIR オプション
 - CMake, 201
- INSTALL_PRIV_LIBDIR オプション
 - CMake, 201
- INSTALL_SBINDIR オプション
 - CMake, 202
- INSTALL_SCRIPTDIR
 - 削除された機能, 41
- INSTALL_SECURE_FILE_PRIVDIR オプション

CMake, 202
INSTALL_SHARED_DIR オプション
CMake, 202
INSTALL_STATIC_LIBRARIES オプション
CMake, 202
INSTALL_SUPPORTFILES_DIR オプション
CMake, 202
INSTR(), 1906
INT データ型, 1764
INTEGER データ型, 1764
interactive_timeout システム変数, 713
interceptor, 5343
InteriorRingN()
削除された機能, 40
INTERNAL_AUTO_INCREMENT(), 2128
INTERNAL_AVG_ROW_LENGTH(), 2128
INTERNAL_CHECKSUM(), 2128
INTERNAL_CHECK_TIME(), 2128
INTERNAL_DATA_FREE(), 2128
INTERNAL_DATA_LENGTH(), 2128
INTERNAL_DD_CHAR_LENGTH(), 2128
INTERNAL_GET_COMMENT_OR_ERROR(), 2128
INTERNAL_GET_ENABLED_ROLE_JSON() 関数, 2128
INTERNAL_GET_HOSTNAME() 関数, 2128
INTERNAL_GET_USERNAME() 関数, 2128
INTERNAL_GET_VIEW_WARNING_OR_ERROR(), 2128
INTERNAL_INDEX_COLUMN_CARDINALITY(), 2128
INTERNAL_INDEX_LENGTH(), 2128
INTERNAL_IS_ENABLED_ROLE() 関数, 2128
INTERNAL_IS_MANDATORY_ROLE() 関数, 2128
INTERNAL_KEYS_DISABLED(), 2128
INTERNAL_MAX_DATA_LENGTH(), 2128
INTERNAL_TABLE_ROWS(), 2128
internal_tmp_disk_storage_engine
削除された機能, 44
internal_tmp_disk_storage_engine システム変数, 713
internal_tmp_mem_storage_engine システム変数, 714
INTERNAL_UPDATE_TIME(), 2128
Intersects()
削除された機能, 40
INTERVAL
時間間隔の構文, 1678
INTERVAL(), 1864
INTO
SELECT, 2331
TABLE ステートメント, 2332
VALUES ステートメント, 2332
カッコで囲まれたクエリー式, 2342
非推奨となった機能, 36
INTO OUTFILE
TABLE ステートメント, 2333
INVOKER 権限, 2566, 4116
IOPS, 5343
io_by_thread_by_latency ビュー
sys スキーマ, 4460
io_global_by_file_by_bytes ビュー
sys スキーマ, 4461
io_global_by_file_by_latency ビュー
sys スキーマ, 4462
io_global_by_wait_by_bytes ビュー

- sys スキーマ, 4462
- io_global_by_wait_by_latency ビュー
 - sys スキーマ, 4463
- IP アドレス
 - アカウント名, 1070
- IPv6 アドレス
 - アカウント名, 1070
- IPv6 接続, 775
- IS boolean_value, 1864
- IS NOT boolean_value, 1865
- IS NOT DISTINCT FROM 演算子, 1862
- IS NOT NULL, 1865
- IS NULL, 1466, 1865
 - およびインデックス, 1505
- IsClosed()
 - 削除された機能, 40
- IsEmpty()
 - 削除された機能, 40
- ISNULL(), 1865
- ISOLATION LEVEL, 2387
- IsSimple()
 - 削除された機能, 40
- IS_FREE_LOCK(), 1992
- IS_IPV4(), 2135
- IS_IPV4_COMPAT(), 2136
- IS_IPV4_MAPPED(), 2136
- IS_IPV6(), 2136
- IS_USED_LOCK(), 1992
- IS_UUID(), 2137
- IS_VISIBLE_DD_OBJECT(), 2128
- ITERATE, 2448

J

- J2EE, 5343
- Java, 5343
- Java, 4524
- JBoss, 5343
- JDBC, 5343
- JDBC, 4521
- jdbc:mysql:loadbalance://, 3451
- JNDI, 5343
- JOIN, 2334
- join_buffer_size システム変数, 714
- JOIN_INDEX, 1572
- JSON
 - array, 1811
 - false リテラル, 1811
 - NDB Cluster, 2262
 - null リテラル, 1811
 - null、true および false リテラル, 1813
 - object, 1811
 - string, 1811
 - true リテラル, 1811
 - 意味のある値, 1814
 - 引用符の処理, 1813
 - 時間値, 1811
 - 自動ラップされた値, 1814
 - 新機能, 20
 - スカラー, 1811

- 正規化値, 1814
- 有効な値, 1811
- JSON 関数, 2045, 2046
- JSON スキーマ CHECK 制約
 - 新機能, 30
- JSON スキーマ検証, 2078
 - 新機能, 26
- JSON データ型, 1809
- JSON 値のマージ, 1815
- JSON ポインタ URI フラグメント識別子, 2081
- JSON_APPEND()
 - 削除された機能, 44
- JSON_ARRAY(), 2048
- JSON_ARRAYAGG(), 2094
- JSON_ARRAY_APPEND(), 2063
- JSON_ARRAY_INSERT(), 2064
- JSON_CONTAINS(), 2049
- JSON_CONTAINS_PATH(), 2049
- JSON_DEPTH(), 2071
- JSON_EXTRACT(), 2050
- JSON_INSERT(), 2065
- JSON_KEYS(), 2054
- JSON_LENGTH(), 2072
- JSON_MERGE() (deprecated), 1815
- JSON_MERGE(), 2065
 - 非推奨となった機能, 36
- JSON_MERGE_PATCH(), 1815, 2066
- JSON_MERGE_PRESERVE(), 1815, 2068
- JSON_OBJECT(), 2048
- JSON_OBJECTAGG(), 2094
- JSON_OVERLAPS(), 2055
- JSON_PRETTY(), 2083
- JSON_QUOTE(), 2048
- JSON_REMOVE(), 2069
- JSON_REPLACE(), 2069
- JSON_SCHEMA_VALID(), 2078
 - CHECK 制約, 2080
- JSON_SCHEMA_VALIDATION_REPORT(), 2081
- JSON_SEARCH(), 2056
- JSON_SET(), 2069
- JSON_STORAGE_FREE(), 2084
- JSON_STORAGE_SIZE(), 2086
- JSON_TABLE(), 2074
- JSON_TABLE() 構文
 - 非推奨となった機能, 37
- JSON_TYPE(), 2072
- JSON_UNQUOTE(), 2070
- JSON_VALID(), 2073
- JSON_VALUE(), 2059

K

- keep-state オプション
 - ndb_import, 3770
- keep_files_on_create システム変数, 715
- KEY によるカラムインデックス接頭辞およびパーティション
 - 非推奨となった機能, 37
- KEY パーティショニング, 4049
- KEY パーティション
 - 管理, 4063

- 分割およびマージ, 4063
- key-file オプション
 - ndb_setup.py, 3820
- keyring, 1231
 - キー管理, 1256
- keyring-migration-destination オプション
 - mysqld, 1263
- keyring-migration-host オプション
 - mysqld, 1264
- keyring-migration-password オプション
 - mysqld, 1264
- keyring-migration-port オプション
 - mysqld, 1264
- keyring-migration-socket オプション
 - mysqld, 1264
- keyring-migration-source オプション
 - mysqld, 1264
- keyring-migration-user オプション
 - mysqld, 1265
- keyring_aws UDFs
 - keyring_aws_rotate_cmk(), 1262
 - keyring_aws_rotate_keys(), 1263
- keyring_aws キーリングプラグイン, 1241
- keyring_aws プラグイン
 - インストール, 1233
- keyring_aws_cmk_id システム変数, 1265
- keyring_aws_conf_file システム変数, 1265
- keyring_aws_data_file システム変数, 1266
- keyring_aws_region システム変数, 1266
- keyring_aws_rotate_cmk() keyring_aws UDF, 1262
- keyring_aws_rotate_keys() keyring_aws UDF, 1263
- keyring_encrypted_file キーリングプラグイン, 1235
- keyring_encrypted_file プラグイン
 - インストール, 1233
- keyring_encrypted_file_data システム変数, 1267
- keyring_encrypted_file_password システム変数, 1268
- keyring_file キーリングプラグイン, 1234
- keyring_file プラグイン, 2805
 - インストール, 1233
- keyring_file_data システム変数, 1268
- keyring_hashicorp UDFs
 - keyring_hashicorp_update_config(), 1263
- keyring_hashicorp キーリングプラグイン, 1243
 - 構成, 1248
- keyring_hashicorp プラグイン
 - インストール, 1233
- keyring_hashicorp_auth_path システム変数, 1269
- keyring_hashicorp_caching システム変数, 1270
- keyring_hashicorp_ca_path システム変数, 1270
- keyring_hashicorp_commit_auth_path システム変数, 1270
- keyring_hashicorp_commit_caching システム変数, 1271
- keyring_hashicorp_commit_ca_path システム変数, 1271
- keyring_hashicorp_commit_role_id システム変数, 1271
- keyring_hashicorp_commit_server_url システム変数, 1271
- keyring_hashicorp_commit_store_path システム変数, 1272
- keyring_hashicorp_role_id システム変数, 1272
- keyring_hashicorp_secret_id システム変数, 1272
- keyring_hashicorp_server_url システム変数, 1273
- keyring_hashicorp_store_path システム変数, 1273
- keyring_hashicorp_update_config() keyring_hashicorp UDF, 1263

keyring_keys テーブル
performance_schema, 1232, 1302, 1329, 4404

keyring_key_fetch() キーリング UDF, 1260

keyring_key_generate() キーリング UDF, 1261

keyring_key_length_fetch() キーリング UDF, 1261

keyring_key_remove() キーリング UDF, 1261

keyring_key_store() キーリング UDF, 1262

keyring_key_type_fetch() キーリング UDF, 1262

keyring_oci キーリングプラグイン, 1250
構成, 1250

keyring_oci プラグイン
インストール, 1233

keyring_oci_ca_certificate システム変数, 1273

keyring_oci_compartment システム変数, 1274

keyring_oci_encryption_endpoint システム変数, 1274

keyring_oci_key_file システム変数, 1274

keyring_oci_key_fingerprint システム変数, 1275

keyring_oci_management_endpoint システム変数, 1275

keyring_oci_master_key システム変数, 1276

keyring_oci_secrets_endpoint システム変数, 1276

keyring_oci_tenancy システム変数, 1276

keyring_oci_user システム変数, 1277

keyring_oci_vaults_endpoint システム変数, 1277

keyring_oci_virtual_vault システム変数, 1277

keyring_okv キーリングプラグイン, 1236
Gemalto SafeNet KeySecure アプリケーション, 1239
Oracle Key Vault, 1238
Townsend Alliance Key Manager, 1240
構成, 1237

keyring_okv プラグイン, 2805
インストール, 1233

keyring_okv_conf_dir システム変数, 1278

keyring_operations システム変数, 1278

keyring_udf プラグイン
アンインストール, 1256
インストール, 1256

keys オプション
mysqlshow, 470

keys-used オプション
myisamchk, 499

keystore, 5343

KEYWORDS
INFORMATION_SCHEMA テーブル, 4157

keyword_list.h ヘッダーファイル, 4157

Key_blocks_not_flushed ステータス変数, 846

Key_blocks_unused ステータス変数, 846

Key_blocks_used ステータス変数, 846

KEY_BLOCK_SIZE, 5344

KEY_BLOCK_SIZE, 2761, 2766

key_buffer_size myisamchk 変数, 497

key_buffer_size システム変数, 715

key_cache_age_threshold システム変数, 716

key_cache_block_size システム変数, 717

key_cache_division_limit システム変数, 717

KEY_COLUMN_USAGE
INFORMATION_SCHEMA テーブル, 4155

Key_reads ステータス変数, 846

Key_read_requests ステータス変数, 846

Key_writes ステータス変数, 846

Key_write_requests ステータス変数, 846

Kill

- スレッドのコマンド, 1615
- KILL ステートメント, 2606
- Killed
 - スレッドの状態, 1618
- Killing slave
 - スレッドの状態, 1624
- krb5.conf ファイル
 - LDAP 認証, 1189

L

- LAG(), 2112
- lap オプション
 - ndb_redo_log_reader, 3787
- large_files_support システム変数, 717
- large_pages システム変数, 717
- large_page_size システム変数, 718
- LAST_DAY(), 1891
- last_insert_id システム変数, 718
- LAST_INSERT_ID(), 1998, 2302
 - およびストアドルーチン, 4099
 - およびトリガー, 4099
 - およびレプリケーション, 3212
- Last_query_cost ステータス変数, 846
- Last_query_partial_plans ステータス変数, 846
- LAST_VALUE(), 2113
- LateAlloc, 3602
- latest_file_io ビュー
 - sys スキーマ, 4464
- lc-messages オプション
 - mysqld, 653
- lc-messages-dir オプション
 - mysqld, 653
- LCASE(), 1906
- LcpScanProgressTimeout, 3595
- lcp_simulator.cc (テストプログラム), 3597
- lc_messages システム変数, 718
- lc_messages_dir システム変数, 718
- lc_time_names システム変数, 719
- LDAP
 - authentication, 1176
- LDAP 認証
 - GSSAPI 認証方式, 1188
 - Kerberos 認証方式, 1188
 - krb5.conf ファイル, 1189
 - ldap_destroy_tgt パラメータ, 1193
 - ldap_server_host パラメータ, 1193
 - 「WITH_AUTHENTICATION_LDAP CMake」オプション, 209
 - クライアントサイドロギング, 1206
 - サーバー側ロギング, 1205, 1212
- ldap.conf 構成ファイル, 1181
- LDAPNOINIT 環境変数, 1181
- ldap_destroy_tgt パラメータ
 - LDAP 認証, 1193
- ldap_server_host パラメータ
 - LDAP 認証, 1193
- LDML 構文, 1751
- LD_LIBRARY_PATH 環境変数, 264
- LD_PRELOAD 環境変数, 177

LD_RUN_PATH 環境変数, 264, 550
LEAD(), 2114
LEAST(), 1866
LEAVE, 2448
ledir オプション
 mysqld_safe, 337
LEFT JOIN, 1456, 2334
LEFT OUTER JOIN, 2334
LEFT(), 1906
length オプション
 mysam_ftdump, 492
LENGTH(), 1907
libaio, 98, 162, 207
libmysql, 5344
libmysql
 FAQ, 4573
libmysqlclient, 5344
libmysqlclient ライブラリ, 4521
libmysqld, 5344
libmysqld
 削除された機能, 42
LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN 環境変数, 550
LIBMYSQL_PLUGINS 環境変数, 550
LIBMYSQL_PLUGIN_DIR 環境変数, 550
library
 libmysqlclient, 4521
libs オプション
 mysql_config, 546
libs_r オプション
 mysql_config, 546
license システム変数, 719
LIKE, 1914
 およびインデックス, 1504
 およびワイルドカード, 1505
LIMIT, 1996, 2329
 およびレプリケーション, 3223
 カッコで囲まれたクエリー式, 2342
 最適化, 1473
limits
 file-size, 1521
 InnoDB, 2980
 結合当たりの最大テーブル数, 2336
 最大行サイズ, 1523
 データベースの最大数, 1521, 2189
 テーブル当たりの最大カラム数, 1523
 テーブルサイズ, 1521
 テーブルの最大数, 1521, 2221
 ビュー当たりの最大テーブル数, 4128
line-numbers オプション
 mysql, 372
LINEAR HASH パーティショニング, 4048
LINEAR KEY パーティショニング, 4051
linefeed (\n), 1628, 2070, 2312
LineFromText()
 削除された機能, 40
LineFromWKB()
 削除された機能, 40
lines-terminated-by オプション
 mysqldump, 431, 444
 ndb_import, 3770

- ndb_restore, 3799
- LINESTRING データ型, 1794
- LineString(), 2011
- LineStringFromText()
 - 削除された機能, 40
- LineStringFromWKB()
 - 削除された機能, 40
- LINK_RANDOMIZE オプション
 - CMake, 202
- LINK_RANDOMIZE_SEED オプション
 - CMake, 202
- LIST パーティショニング, 4038, 4040
- LIST パーティション
 - 管理, 4057
 - 追加および削除, 4058
- list_add() 関数
 - sys スキーマ, 4511
- list_drop() 関数
 - sys スキーマ, 4512
- LN(), 1878
- LOAD DATA, 2307, 4608
 - LOCAL ロード, 1039
 - およびレプリケーション, 3223
- LOAD XML, 2316
- load-data-local-dir オプション
 - mysql, 373, 1041
- LOAD_FILE(), 1907
- load_rewrite_rules() リライタ UDF, 965
- local-infile オプション
 - mysql, 373, 1040
- local-load オプション
 - mysqlbinlog, 530, 1042
- local-service オプション
 - mysqld, 654
- localhost, 5344
- localhost
 - 特別な処理, 320
- LOCALTIME, 1891
- LOCALTIMESTAMP, 1891
- local_infile システム変数, 719, 1040
- LOCATE(), 1907
- LocationDomainId (API ノード), 3650
- LocationDomainId (管理ノード), 3570
- LocationDomainId (データノード), 3578
- LOCK IN SHARE MODE, 2330
- LOCK INSTANCE FOR BACKUP, 2381
- lock mode, 5344
- LOCK TABLES, 2382
- LOCK TABLES 権限, 1050
- lock オプション
 - ndb_select_all, 3815
- lock-all-tables オプション
 - mysqldump, 436
- lock-tables オプション
 - mysqldump, 436
 - mysqlimport, 444
- Locked_connects ステータス変数, 846
- locked_in_memory システム変数, 720
- LockExecuteThreadToCPU, 3624
- locking_service サービス, 1000

LockMaintThreadsToCPU, 3625
LockPagesInMainMemory, 3603
locks_per_fragment
 ndbinfo テーブル, 3943
lock_order システム変数, 1023
LOCK_ORDER ツール, 1022
lock_order_debug_loop システム変数, 1024
lock_order_debug_missing_arc システム変数, 1024
lock_order_debug_missing_key システム変数, 1024
lock_order_debug_missing_unlock システム変数, 1025
lock_order_dependencies システム変数, 1025
lock_order_extra_dependencies システム変数, 1025
lock_order_output_directory システム変数, 1025
lock_order_print_txt システム変数, 1026
lock_order_trace_loop システム変数, 1026
lock_order_trace_missing_arc システム変数, 1026
lock_order_trace_missing_key システム変数, 1027
lock_order_trace_missing_unlock システム変数, 1027
lock_wait_timeout システム変数, 719
log オプション
 innochecksum, 489
 mysqld_multi, 343
LOG(), 1878
log-bin オプション
 mysqld, 3112
log-bin-index オプション
 mysqld, 3113
log-error オプション
 mysqld, 654
 mysqldump, 427
 mysqld_safe, 337
log-error-file オプション
 mysqldump, 458
log-isam オプション
 mysqld, 654
log-name オプション
 ndb_mgmd, 3738
log-raw オプション
 mysqld, 654
log-short-format オプション
 mysqld, 655
log-tc オプション
 mysqld, 655
log-tc-size オプション
 mysqld, 655
LOG10(), 1878
LOG2(), 1878
logbuffer-size オプション
 ndbd, 3729
 ndbmtd, 3729
logbuffers
 ndbinfo テーブル, 3945
LogDestination, 3571
logging
 新機能, 25
logging slow query
 スレッドの状態, 1618
login
 スレッドの状態, 1618
login-path オプション, 307

- mysql, 373
- mysqladmin, 401
- mysqlbinlog, 530
- mysqlcheck, 410
- mysqldump, 422
- mysqlimport, 444
- mysqlpump, 458
- mysqlshow, 470
- mysqslap, 479
- mysql_upgrade, 359
- my_print_defaults, 548
- NDB クライアントプログラム, 3836
- LogLevelCheckpoint, 3618
- LogLevelCongestion, 3619
- LogLevelConnection, 3618
- LogLevelError, 3618
- LogLevelInfo, 3619
- LogLevelNodeRestart, 3618
- LogLevelShutdown, 3617
- LogLevelStartup, 3617
- LogLevelStatistic, 3618
- logspaces
 - ndbinfo テーブル, 3946
- log_bin システム変数, 3131
- log_bin_basename システム変数, 3132
- log_bin_index システム変数, 3132
- log_bin_trust_function_creators システム変数, 3132
- log_bin_use_v1_row_events システム変数, 3133
- log_builtin_as_identified_by_password
 - 削除された機能, 38
- log_error システム変数, 720
- log_error_services システム変数, 720
- log_error_suppression_list システム変数, 721
- log_error_verbosity システム変数, 721
- log_filter_dagnet ログコンポーネント, 944
- log_filter_internal ログコンポーネント, 944
- log_output システム変数, 722
- log_queries_not_using_indexes システム変数, 723
- log_raw システム変数, 723
- log_sink_internal ログコンポーネント, 944
- log_sink_json ログコンポーネント, 944
- log_sink_syseventlog ログコンポーネント, 945
- log_sink_test ログコンポーネント, 945
- log_slave_updates システム変数, 3133
- log_slow_admin_statements システム変数
 - mysqld, 723
- log_slow_extra システム変数, 723
- log_slow_slave_statements システム変数, 3091
- log_statements_unsafe_for_binlog システム変数, 3133
- log_status テーブル
 - performance_schema, 4404
- log_syslog システム変数, 724
- log_syslog_facility システム変数, 724
- log_syslog_include_pid システム変数, 724
- log_syslog_tag システム変数, 725
- log_throttle_queries_not_using_indexes システム変数, 725
- log_timestamps システム変数, 725
- log_warnings
 - 削除された機能, 39
- Long Data

- スレッドのコマンド, 1615
- LONG データ型, 1787
- LONGBLOB データ型, 1784
- LongMessageBuffer, 3590
- LONGTEXT データ型, 1784
- long_query_time システム変数, 726
- LOOP, 2448
 - ラベル, 2444
- loops オプション
 - ndbinfo_select_all, 3732
 - ndb_index_stat, 3779
 - ndb_show_tables, 3822
- LooseScan
 - 準結合方式, 1482
- loose_, 5344
- lossy-conversions オプション
 - ndb_move_data, 3781
 - ndb_restore, 3800
- LOWER(), 1907
- lower_case_file_system システム変数, 726
 - GRANT, 2509
- lower_case_table_names システム変数, 727
- LOW_PRIORITY
 - DELETE 修飾子, 2293
 - INSERT 修飾子, 2302
 - UPDATE 修飾子, 2362
- low_priority_updates システム変数, 726
- LPAD(), 1908
- LRU, 5344
- LRU ページの置換, 2733
- LSN, 5345
- LTRIM(), 1908
- lz4_decompress, 299, 548

M

- M LineFromText()
 - 削除された機能, 40
- M LineFromWKB()
 - 削除された機能, 40
- macOS
 - インストール, 139
- MAKEDATE(), 1891
- MAKETIME(), 1891
- MAKE_SET(), 1908
- Making temporary file (append) before replaying LOAD DATA INFILE
 - スレッドの状態, 1623
- Making temporary file (create) before replaying LOAD DATA INFILE
 - スレッドの状態, 1623
- manage keys
 - スレッドの状態, 1618
- management
 - リソースグループ, 886
- mandatory_roles システム変数, 728
- manual
 - 使用可能なフォーマット, 2
- mask_inner() MySQL Enterprise Data Masking and De-Identification UDF, 1380
- mask_outer() MySQL Enterprise Data Masking and De-Identification UDF, 1381
- mask_pan() MySQL Enterprise Data Masking and De-Identification UDF, 1382
- mask_pan_relaxed() MySQL Enterprise Data Masking and De-Identification UDF, 1382

mask_ssn() MySQL Enterprise Data Masking and De-Identification UDF, 1383
master-data オプション
mysqldump, 429
master-info-file オプション
mysqld, 3080
master-retry-count オプション
mysqld, 3080
master_info_repository システム変数, 3091, 3163
MASTER_POS_WAIT(), 2137, 2423
master_verify_checksum システム変数, 3134
MATCH ... AGAINST(), 1927
MAX(), 2097
MAX(DISTINCT), 2097
max-allowed-packet オプション
mysql, 373
mysqldump, 435
mysqldump, 458
mysql_upgrade, 359
max-binlog-dump-events オプション
mysqld, 3116
max-join-size オプション
mysql, 373
max-record-length オプション
myisamchk, 499
max-relay-log-size オプション
mysqld, 3081
max-rows オプション
ndb_import, 3770
MaxAllocate, 3592
MaxBufferedEpochBytes, 3611
MaxBufferedEpochs, 3611
MAXDB
削除された機能, 39
MaxDiskDataLatency, 3643
MaxDiskWriteSpeed, 3613
MaxDiskWriteSpeedOtherNodeRestart, 3613
MaxDiskWriteSpeedOwnRestart, 3613
MaxDMLOperationsPerTransaction, 3586
MaxFKBuildBatchSize, 3590
maximums
結合当たりの最大テーブル数, 2336
最大行サイズ, 1523
データベースの最大数, 1521, 2189
テーブル当たりの最大カラム数, 1523
テーブルサイズ, 1521
テーブルの最大数, 1521, 2221
ビュー当たりの最大テーブル数, 4128
MaxLCPStartDelay, 3596
MaxNoOfAttributes, 3598
MaxNoOfConcurrentIndexOperations, 3586
MaxNoOfConcurrentOperations, 3585
MaxNoOfConcurrentScans, 3591
MaxNoOfConcurrentSubOperations, 3601
MaxNoOfConcurrentTransactions, 3584
MaxNoOfExecutionThreads
ndbmt, 3628
MaxNoOfFiredTriggers, 3587
MaxNoOfLocalOperations, 3585
MaxNoOfLocalScans, 3591
MaxNoOfOpenFiles, 3595

MaxNoOfOrderedIndexes, 3599
MaxNoOfSavedMessages, 3595
MaxNoOfSubscribers, 3601
MaxNoOfSubscriptions, 3600
MaxNoOfTables, 3599
MaxNoOfTriggers, 3600
MaxNoOfUniqueHashIndexes, 3600
MaxParallelCopyInstances, 3591
MaxParallelScansPerFragment, 3592
MaxReorgBuildBatchSize, 3592
MaxScanBatchSize, 3652
MaxStartFailRetries, 3646
MaxUIBuildBatchSize, 3592
max_allowed_packet
 およびレプリケーション, 3223
max_allowed_packet システム変数, 728
max_binlog_cache_size システム変数, 3134
max_binlog_size システム変数, 3134
max_binlog_stmt_cache_size システム変数, 3135
max_connections システム変数, 729
MAX_CONNECTIONS_PER_HOUR, 1122
max_connect_errors システム変数, 729
max_delayed_threads システム変数, 730
max_digest_length システム変数, 730
max_error_count システム変数, 731
max_execution_time システム変数, 731
Max_execution_time_exceeded ステータス変数, 846
Max_execution_time_set ステータス変数, 847
Max_execution_time_set_failed ステータス変数, 847
max_heap_table_size システム変数, 731
MAX_INDEXES オプション
 CMake, 207
max_insert_delayed_threads システム変数, 732
max_join_size システム変数, 391, 732
max_length_for_sort_data
 非推奨となった機能, 37
max_length_for_sort_data システム変数, 733
max_points_in_geometry システム変数, 733
max_prepared_stmt_count システム変数, 733
MAX_QUERIES_PER_HOUR, 1122
max_relay_log_size システム変数, 3092
MAX_ROWS
 and NDB Cluster, 4032
 NDB Cluster, 2233
 および DataMemory (NDB Cluster), 3580
max_seeks_for_key システム変数, 734
max_sort_length システム変数, 734
max_sp_recursion_depth システム変数, 734
max_tmp_tables
 削除された機能, 39
MAX_UPDATES_PER_HOUR, 1122
Max_used_connections ステータス変数, 847
Max_used_connections_time ステータス変数, 847
MAX_USER_CONNECTIONS, 1122
max_user_connections システム変数, 735
max_write_lock_count システム変数, 735
MBR, 2034
MBRContains(), 2035
MBRCoveredBy(), 2035
MBRCovers(), 2035

MBRDisjoint(), 2036
MBREquals(), 2036
MBRIntersects(), 2036
MBROverlaps(), 2036
MBRTouches(), 2036
MBRWithin(), 2036
MD5(), 1988
MDL, 5345
measured-load オプション
 ndb_top, 3827
mecab_charset ステータス変数, 847
mecab_rc_file システム変数, 736
medium-check オプション
 mysamchk, 498
 mysqlcheck, 411
MEDIUMBLOB データ型, 1783
MEDIUMINT データ型, 1763
MEDIUMTEXT データ型, 1783
MEMBER OF(), 2062
membership
 ndbinfo テーブル, 3946
memcached, 5345
memcached, 2942
MEMCACHED_HOME オプション
 CMake, 217
MEMCACHED_SASL_PWDB 環境変数, 2954
memcapable コマンド, 2944
memlock オプション
 mysqld, 655
MEMORY ストレージエンジン, 2983, 2995
 およびレプリケーション, 3224
 最適化, 1501
memoryusage
 ndbinfo テーブル, 3948
memory_by_host_by_current_bytes ビュー
 sys スキーマ, 4465
memory_by_thread_by_current_bytes ビュー
 sys スキーマ, 4465
memory_by_user_by_current_bytes ビュー
 sys スキーマ, 4466
memory_global_by_current_bytes ビュー
 sys スキーマ, 4466
memory_global_total ビュー
 sys スキーマ, 4467
memory_per_fragment
 ndbinfo テーブル, 3949
memory_summary_by_account_by_event_name テーブル
 performance_schema, 4390
memory_summary_by_host_by_event_name テーブル
 performance_schema, 4390
memory_summary_by_thread_by_event_name テーブル
 performance_schema, 4390
memory_summary_by_user_by_event_name テーブル
 performance_schema, 4390
memory_summary_global_by_event_name テーブル
 performance_schema, 4390
MemReportFrequency, 3619
MERGE ストレージエンジン, 2983, 3004
MERGE テーブル
 定義済み, 3004

metadata_locks テーブル
performance_schema, 4355

metadata_locks_cache_size
削除された機能, 39

metadata_locks_cache_size システム変数, 736

metadata_locks_hash_instances
削除された機能, 39

metadata_locks_hash_instances システム変数, 736

mgmd (NDB Cluster)
定義済, 3450
(参照 [管理ノード \(NDB Cluster\)](#))

MICROSECOND(), 1891

MID(), 1908

MIN(), 2097

MIN(DISTINCT), 2097

MinDiskWriteSpeed, 3614

MinFreePct, 3580, 3583

MINUTE(), 1891

min_examined_row_limit システム変数, 737

MM.MySQL, 5345

MOD (モジュロ), 1879

MOD(), 1879

modify オプション
MySQLInstallerConsole, 126

modulo (MOD), 1879

MONTH(), 1891

MONTHNAME(), 1892

MPointFromText()
削除された機能, 40

MPointFromWKB()
削除された機能, 40

MPolyFromText()
削除された機能, 40

MPolyFromWKB()
削除された機能, 40

MRR, 1572

MSSQL
削除された機能, 39

Multi-Range Read
最適化, 1459

MULTILINESTRING データ型, 1794

MultiLineString(), 2011

MULTIPOINT データ型, 1794

MultiPoint(), 2011

MultiPointFromText()
削除された機能, 40

MultiPointFromWKB()
削除された機能, 40

MULTIPOLYGON データ型, 1794

MultiPolygon(), 2011

MultiPolygonFromText()
削除された機能, 40

MultiPolygonFromWKB()
削除された機能, 40

multi_range_count
削除された機能, 39

mutex wait
監視, 2928

mutex_instances テーブル
performance_schema, 4290

MUTEX_TYPE オプション
 CMake, 208
MVCC, 5345
MVCC (マルチバージョン並列処理制御), 2631
My
 由来, 8
my.cnf, 5345
my.cnf
 NDB Cluster, 3509, 3560, 3561
 NDB Cluster 内, 3889
my.cnf オプションファイル, 3212
my.ini, 5345
mycnf オプション
 ndb_config, 3749
 ndb_mgmd, 3738
MyISAM
 InnoDB へのテーブルの変換, 2654
 圧縮テーブル, 509, 2993
MyISAM キーキャッシュ, 1587
MyISAM ストレージエンジン, 2983, 2987
mysam-block-size オプション
 mysqld, 656
mysamchk, 298, 492
 block-search オプション, 500
 character-sets-dir オプション, 499
 check-only-changed オプション, 498
 correct-checksum オプション, 499
 data-file-length オプション, 499
 defaults-extra-file オプション, 495
 defaults-file オプション, 496
 defaults-group-suffix オプション, 496
 extend-check オプション, 498, 499
 force オプション, 499
 HELP オプション, 495
 keys-used オプション, 499
 max-record-length オプション, 499
 medium-check オプション, 498
 no-defaults オプション, 496
 no-symlinks オプション, 499
 parallel-recover オプション, 499
 print-defaults オプション, 496
 safe-recover オプション, 500
 set-auto-increment[option, 501
 set-collation オプション, 500
 sort-index オプション, 501
 sort-records オプション, 501
 sort-recover オプション, 500
 tmpdir オプション, 500
 unpack オプション, 500
 update-state オプション, 499
 verbose オプション, 496
 wait オプション, 496
 オプション, 495
 強制オプション, 498
 クイックオプション, 500
 高速オプション, 498
 サイレントオプション, 496
 出力例, 501
 情報オプション, 498
 説明オプション, 500

- チェックオプション, 498
- デバッグオプション, 495
- バージョンオプション, 496
- バックアップオプション, 499
- 分析オプション, 500
- ヘルプオプション, 495
- 読取り専用オプション, 498
- リカバリオプション, 500
- mysamlog, 298, 508
- mysampack, 298, 509, 2257, 2993
 - character-sets-dir オプション, 509
 - tmpdir オプション, 510
 - verbose オプション, 510
 - wait オプション, 510
 - 強制オプション, 510
 - 結合オプション, 510
 - サイレントオプション, 510
 - テストオプション, 510
 - デバッグオプション, 509
 - バージョンオプション, 510
 - バックアップオプション, 509
 - ヘルプオプション, 509
- mysam_block_size mysamchk 変数, 497
- mysam_data_pointer_size システム変数, 737
- mysam_ftdump, 298, 491
 - length オプション, 492
 - stats オプション, 492
 - verbose オプション, 492
 - カウントオプション, 491
 - ダンプオプション, 492
 - ヘルプオプション, 491
- mysam_max_sort_file_size システム変数, 737, 2990
- mysam_mmap_size システム変数, 738
- mysam_recover_options システム変数, 738, 2990
- mysam_repair_threads システム変数, 739
- mysam_sort_buffer_size mysamchk 変数, 497
- mysam_sort_buffer_size システム変数, 739, 2990
- mysam_stats_method システム変数, 740
- mysam_use_mmap システム変数, 740
- MyODBC ドライバ, 5346
- mysql, 5346
- MySQL APT リポジトリ, 157, 257
- MySQL C API, 4525
- MySQL Cluster Manager
 - および ndb_mgm, 3839
- MySQL Dolphin の名前, 8
- MySQL Enterprise Audit, 1279, 4531
- MySQL Enterprise Backup, 5346
- MySQL Enterprise Backup, 4530
 - グループレプリケーション, 3299
- MySQL Enterprise Data Masking and De-Identification, 1371, 4532
- MySQL Enterprise Data Masking and De-Identification UDFs
 - gen_blacklist(), 1385
 - gen_blocklist(), 1386
 - gen_dictionary(), 1386
 - gen_dictionary_drop(), 1387
 - gen_dictionary_load(), 1388
 - gen_range(), 1383
 - gen_rnd_email(), 1384
 - gen_rnd_pan(), 1384

- gen_rnd_ssn(), 1385
- gen_rnd_us_phone(), 1385
- mask_inner(), 1380
- mask_outer(), 1381
- mask_pan(), 1382
- mask_pan_relaxed(), 1382
- mask_ssn(), 1383
- MySQL Enterprise Data Masking and De-Identification プラグイン
 - 要素, 1373
- MySQL Enterprise Encryption, 1388, 4531
- MySQL Enterprise Firewall, 1348, 4531
 - インストール, 1349
 - 使用, 1352
- MySQL Enterprise Firewall UDFs
 - firewall_group_delist(), 1369
 - firewall_group_enlist(), 1369
 - mysql_firewall_flush_status(), 1370
 - normalize_statement(), 1370
 - read_firewall_groups(), 1369
 - read_firewall_group_allowlist(), 1369
 - read_firewall_users(), 1368
 - read_firewall_whitelist(), 1368
 - set_firewall_group_mode(), 1369
 - set_firewall_mode(), 1369
- MySQL Enterprise Firewall ストアドプロシージャ
 - sp_firewall_group_delist(), 1366
 - sp_firewall_group_enlist(), 1366
 - sp_reload_firewall_group_rules(), 1366
 - sp_reload_firewall_rules(), 1365
 - sp_set_firewall_group_mode(), 1367
 - sp_set_firewall_group_mode_and_user(), 1368
 - sp_set_firewall_mode(), 1365
- MySQL Enterprise Firewall テーブル
 - firewall_groups, 1364
 - firewall_group_allowlist, 1364
 - firewall_membership, 1364
 - firewall_users, 1363
 - firewall_whitelist, 1363
- MySQL Enterprise Monitor, 4529
- MySQL Enterprise Security, 1162, 1171, 1176, 4531
- MySQL Enterprise Thread Pool, 951, 4532
 - インストール, 952
 - 要素, 952
- MySQL Enterprise 透過的データ暗号化, 2805
- MySQL Installer, 105
- MySQL
 - websites, 61
 - アップグレード, 353
 - 定義, 4
 - デバッグ, 1016
 - はじめに, 4
 - 発音, 5
 - フォーラム, 61
- mysql, 297, 362
 - 1つのデータベースオプション, 374
 - auto-rehash オプション, 367
 - auto-vertical-output オプション, 368
 - binary-as-hex オプション, 368
 - binary-mode オプション, 369
 - bind-address オプション, 369

character-sets-dir オプション, 369
charset コマンド, 380
clear コマンド, 381
column-names オプション, 369
column-type-info オプション, 369
compress オプション, 369
compression-algorithms オプション, 369
connect コマンド, 381
connect-expired-password オプション, 370
connect-timeout オプション, 370
debug-check オプション, 370
debug-info オプション, 370
default-auth オプション, 370
default-character-set オプション, 370
defaults-extra-file オプション, 370
defaults-file オプション, 371
defaults-group-suffix オプション, 371
delimiter オプション, 371
delimiter コマンド, 381
disable named コマンド, 371
dns-srv-name オプション, 371
edit コマンド, 381
ego コマンド, 381
enable-cleartext-plugin オプション, 372
exit コマンド, 381
get-server-public-key オプション, 372
go コマンド, 381
help コマンド, 380
histignore オプション, 372
html オプション, 372
i-am-a-dummy オプション, 376
ignore-spaces オプション, 372
init-command オプション, 372
line-numbers オプション, 372
load-data-local-dir オプション, 373, 1041
local-infile オプション, 373, 1040
login-path オプション, 373
max-allowed-packet オプション, 373
max-join-size オプション, 373
named-commands オプション, 373
net-buffer-length オプション, 374
network-namespace オプション, 374
no-auto-rehash オプション, 374
no-beep オプション, 374
no-defaults オプション, 374
nopager コマンド, 381
notee コマンド, 382
nowarning コマンド, 382
pager オプション, 375
pager コマンド, 382
pipe オプション, 375
plugin-dir オプション, 375
port オプション, 375
print コマンド, 382
print-defaults オプション, 375
prompt オプション, 375
prompt コマンド, 382
protocol オプション, 375
query_attributes コマンド, 382
quit コマンド, 382

- raw オプション, 376
- reconnect オプション, 376
- rehash コマンド, 383
- resetconnection コマンド, 383
- safe-updates オプション, 376
- select-limit オプション, 376
- server-public-key-path オプション, 376
- shared-memory-base-name オプション, 377
- show-warnings オプション, 377
- sigint-ignore オプション, 377
- skip-column-names オプション, 377
- skip-line-numbers オプション, 377
- socket オプション, 377
- source コマンド, 383
- SSL オプション, 377
- ssl-fips-mode オプション, 378
- status コマンド, 383
- syslog オプション, 378
- system コマンド, 383
- tee オプション, 378
- tee コマンド, 383
- tls-ciphersuites オプション, 378
- tls-version オプション, 378
- unbuffered オプション, 379
- use コマンド, 384
- verbose オプション, 379
- wait オプション, 379
- warnings コマンド, 384
- xml オプション, 379
- zstd-compression-level オプション, 379
- 強制オプション, 372
- クイックオプション, 375
- コメントオプション, 369
- サイレントオプション, 377
- 実行オプション, 372
- 垂直オプション, 379
- データベースオプション, 370
- テーブルオプション, 378
- デバッグオプション, 370
- バージョンオプション, 379
- パスワードオプション, 375
- パッチオプション, 368
- ヘルプオプション, 367
- ホストオプション, 372
- ユーザーオプション, 379
- mysql prompt コマンド, 384
- MySQL Server
 - mysqld, 334, 556
- MySQL Server のコンパイル
 - 問題, 218
- MySQL Shell JavaScript チュートリアル, 3375
 - append insert delete, 3385
 - drop table, 3393
 - MySQL Shell の使用, 3376
 - MySQL Shell のヘルプ, 3377
 - quit MySQL Shell, 3377
 - 一意インデックス, 3388
 - 一意でないインデックス, 3387
 - インデックスの削除, 3388
 - 完全なレコードの挿入, 3389

- 結果の制限、ソートおよびスキップ, 3384
- コレクション操作, 3379
- コレクションの作成, 3379
- コレクションの取得, 3379
- コレクションのドロップ, 3380
- 最後のドキュメントの削除, 3387
- 最初のドキュメントの削除, 3387
- 最初のレコードの削除, 3393
- 条件によるドキュメントの削除, 3386
- 条件を使用したレコードの削除, 3393
- スキーマの確認, 3379
- すべてのドキュメントを検索, 3381
- すべてのドキュメントを削除, 3387
- すべてのレコードの削除, 3393
- すべてのレコードを選択, 3390
- 制限、順序、オフセット結果, 3391
- テーブル挿入レコード, 3389
- テーブル内のドキュメント, 3393
- テーブルの選択, 3389
- テーブルレコードの更新, 3392
- ドキュメントおよびコレクション, 3378
- ドキュメントのインデックス付け, 3387
- ドキュメントの検索, 3381
- ドキュメントの削除, 3386
- ドキュメントの追加, 3380
- ドキュメントの変更, 3385
- フィールドの設定および設定解除, 3385
- フィルタ検索, 3390
- フィルタ検索によるドキュメントの検索, 3381
- プロジェクト結果, 3384, 3391
- リレーショナルテーブル, 3388
- レコードの一部を挿入, 3389
- レコードの選択, 3394
- レコードの挿入, 3394
- ワールド x, 3377
- MySQL Shell, 3371
- NoSQL, 3371
- MySQL Shell Python チュートリアル, 3394
 - append insert delete, 3405
 - collections get, 3398
 - drop table, 3412
 - MySQL Shell の使用, 3395
 - MySQL Shell のヘルプ, 3396
 - quit MySQL Shell, 3396
 - 一意インデックス, 3407
 - 一意でないインデックス, 3406
 - インデックスの削除, 3407
 - オーダーオフセット結果の制限, 3410
 - 完全なレコードの挿入, 3408
 - 結果の制限、ソートおよびスキップ, 3403
 - コレクション作成, 3398
 - コレクション操作, 3398
 - コレクションドロップ, 3399
 - 最後のドキュメントの削除, 3406
 - 最初のドキュメントの削除, 3406
 - 最初のレコードの削除, 3412
 - 条件によるドキュメントの削除, 3406
 - 条件を使用したレコードの削除, 3412
 - スキーマの確認, 3398
 - すべてのドキュメントを検索, 3400

- すべてのドキュメントを削除, 3406
- すべてのレコードの削除, 3412
- すべてのレコードを選択, 3409
- テーブル選択, 3409
- テーブル挿入, 3408
- テーブル内のドキュメント, 3412
- テーブルレコードの更新, 3411
- ドキュメントインデックス, 3406
- ドキュメントおよびコレクション, 3397
- ドキュメント削除, 3405
- ドキュメントの検索, 3400
- ドキュメントの追加, 3399
- ドキュメントの変更, 3404
- フィールドの設定および設定解除, 3404
- フィルタ検索, 3409
- フィルタ検索によるドキュメントの検索, 3401
- プロジェクト結果, 3403, 3410
- リレーショナルテーブル, 3407
- レコードの一部を挿入, 3408
- レコードの選択, 3413
- レコードの挿入, 3413
- ワールド x, 3396
- MySQL SLES リポジトリ, 157, 257
- mysql source (テキストファイルから読み取るためのコマンド), 287, 389
- MySQL Yum リポジトリ, 153, 255
- mysql\ (テキストファイルから読み取るためのコマンド), 287, 389
- mysql クライアントパーサー
 - mysqld パーサーとの比較, 393
- MySQL 権限
 - NDB Cluster, 3987
- mysql コマンド
 - のリスト, 380
- mysql コマンドオプション, 363
- MySQL システムテーブル
 - NDB Cluster, 3987
 - レプリケーション, 3225
- MySQL ストレージエンジン, 2983
- MySQL ソース配布, 84
- MySQL データディクショナリ, 2617
- mysql データベース
 - gtid_executed テーブル, 3035
- MySQL のアップグレード, 353
- MySQL の主な機能, 5
- MySQL の機能, 5
- MySQL の空間拡張, 1793
- MySQL の取得, 85
- MySQL の名前, 8
- MySQL のバージョン, 85
- MySQL の歴史, 8, 8
- MySQL バイナリ配布, 84
- mysql 履歴ファイル, 386
- MySQL をダウンロードするための URL, 85
- mysql.gtid_executed テーブル, 3034, 3035
 - compression, 3036
 - thread/sql/compress_gtid_table, 3036
- mysql.ndb_binlog_index テーブル, 3999
 - (参照 [NDB Cluster レプリケーション](#))
- mysql.server, 296, 340
 - basedir オプション, 342
 - datadir オプション, 342

- pid-file オプション, 342
- service-startup-timeout オプション, 342
- mysql.slave_master_info テーブル, 3163
- mysql.slave_relay_log_info テーブル, 3163
- mysql.sock
 - 保護, 4605
- MYSQL323
 - 削除された機能, 39
- MYSQL40
 - 削除された機能, 39
- mysqldadmin, 297, 393, 2189, 2281, 2588, 2594, 2600, 2606
 - bind-address オプション, 399
 - character-sets-dir オプション, 399
 - compress オプション, 399
 - compression-algorithms オプション, 399
 - connect-timeout オプション, 399
 - debug-check オプション, 399
 - debug-info オプション, 399
 - default-auth オプション, 400
 - default-character-set オプション, 400
 - defaults-extra-file オプション, 400
 - defaults-file オプション, 400
 - defaults-group-suffix オプション, 400
 - enable-clear-text-plugin オプション, 400
 - get-server-public-key オプション, 400
 - login-path オプション, 401
 - no-beep オプション, 401
 - no-defaults オプション, 401
 - pipe オプション, 401
 - plugin-dir オプション, 401
 - port オプション, 401
 - print-defaults オプション, 402
 - protocol オプション, 402
 - server-public-key-path オプション, 402
 - shared-memory-base-name オプション, 402
 - show-warnings オプション, 402
 - shutdown-timeout オプション, 402
 - sleep オプション, 402
 - socket オプション, 403
 - SSL オプション, 403
 - ssl-fips-mode オプション, 403
 - tls-ciphersuites オプション, 403
 - tls-version オプション, 403
 - verbose オプション, 403
 - wait オプション, 404
 - zstd-compression-level オプション, 404
- カウントオプション, 399
- 強制オプション, 400
- サイレントオプション, 402
- 垂直オプション, 403
- 相対オプション, 402
- デバッグオプション, 399
- バージョンオプション, 403
- パスワードオプション, 401
- ヘルプオプション, 399
- ホストオプション, 401
- ユーザーオプション, 403
- mysqldadmin オプション
 - mysqld_multi, 343
- mysqldadmin コマンドオプション, 396

mysqlbackup コマンド, 5346
mysqlbinlog, 298, 520
 base64-output オプション, 526
 bind-address オプション, 526
 binlog-row-event-max-size オプション, 526
 character-sets-dir オプション, 526
 compress オプション, 526
 compression-algorithms オプション, 527
 connection-server-id オプション, 527
 debug-check オプション, 528
 debug-info オプション, 528
 default-auth オプション, 528
 defaults-extra-file オプション, 528
 defaults-file オプション, 528
 defaults-group-suffix オプション, 529
 disable-log-bin オプション, 529
 exclude-gtids オプション, 529
 force-if-open オプション, 529
 force-read オプション, 529
 get-server-public-key オプション, 529
 hexdump オプション, 529
 idempotent オプション, 529
 include-gtids オプション, 530
 local-load オプション, 530, 1042
 login-path オプション, 530
 no-defaults オプション, 530
 offset オプション, 530
 open-files-limit オプション, 530
 plugin-dir オプション, 531
 port オプション, 531
 print-defaults オプション, 531
 print-table-metadata オプション, 531
 protocol オプション, 531
 raw オプション, 531
 read-from-remote-master オプション, 531
 read-from-remote-server オプション, 531
 require-row-format オプション, 532
 result-file オプション, 532
 rewrite-db オプション, 532
 server-id オプション, 533
 server-id-bit オプション, 533
 server-public-key-path オプション, 533
 set-charset オプション, 533
 shared-memory-base-name オプション, 533
 skip-gtids オプション, 533
 socket オプション, 534
 SSL オプション, 534
 ssl-fips-mode オプション, 534
 start-datetime オプション, 534
 start-position オプション, 534
 stop-datetime オプション, 534
 stop-never オプション, 535
 stop-never-slave-server-id オプション, 535
 stop-position オプション, 535
 tls-ciphersuites オプション, 535
 tls-version オプション, 535
 to-last-log オプション, 535
 verbose オプション, 535
 verify-binlog-checksum オプション, 536
 zstd-compression-level オプション, 536

ショートフォームオプション, 533
データベースオプション, 527
デバッグオプション, 528
バージョンオプション, 536
パスワードオプション, 530
ヘルプオプション, 526
ホストオプション, 529
ユーザーオプション, 535
mysqlcheck, 297, 404
 all-databases オプション, 408
 all-in-1 オプション, 408
 bind-address オプション, 408
 character-sets-dir オプション, 408
 check-only-changed オプション, 408
 check-upgrade オプション, 408
 compress オプション, 408
 compression-algorithms オプション, 408
 debug-check オプション, 409
 debug-info オプション, 409
 default-auth オプション, 410
 default-character-set オプション, 409
 defaults-extra-file オプション, 409
 defaults-file オプション, 409
 defaults-group-suffix オプション, 410
 enable-clear-text-plugin オプション, 410
 get-server-public-key オプション, 410
 login-path オプション, 410
 medium-check オプション, 411
 no-defaults オプション, 411
 pipe オプション, 411
 plugin-dir オプション, 411
 port オプション, 411
 print-defaults オプション, 411
 protocol オプション, 411
 server-public-key-path オプション, 412
 shared-memory-base-name オプション, 412
 skip-database オプション, 412
 socket オプション, 412
 SSL オプション, 412
 ssl-fips-mode オプション, 412
 tls-ciphersuites オプション, 413
 tls-version オプション, 413
 use-frm オプション, 413
 verbose オプション, 413
 write-binlog オプション, 413
 zstd-compression-level オプション, 413
拡張オプション, 410
強制オプション, 410
クイックオプション, 412
高速オプション, 410
最適化オプション, 411
サイレントオプション, 412
自動修復オプション, 408
修復オプション, 412
チェックオプション, 408
データベースオプション, 409
テーブルオプション, 413
デバッグオプション, 409
バージョンオプション, 413
パスワードオプション, 411

分析オプション, 408
ヘルプオプション, 408
ホストオプション, 410
ユーザーオプション, 413
mysqlclient, 5346
mysqld, 5346
mysqld (NDB Cluster), 3724
mysqld, 296
 abort-slave-event-count オプション, 3090
 admin-ssl オプション, 645
 allow-suspicious-udfs オプション, 645
 ansi オプション, 646
 audit-log オプション, 1335
 basedir オプション, 646
 binlog-checksum オプション, 3115
 binlog-do-db オプション, 3113
 binlog-ignore-db オプション, 3115
 binlog-row-event-max-size オプション, 3111
 character-set-client-handshake オプション, 646
 chroot オプション, 646
 core-file オプション, 647
 daemonize オプション, 647
 datadir オプション, 647
 ddl-rewriter オプション, 967
 debug-sync-timeout オプション, 648
 default-time-zone オプション, 648
 defaults-extra-file オプション, 648
 defaults-file オプション, 649
 defaults-group-suffix オプション, 649
 disconnect-slave-event-count オプション, 3090
 early-plugin-load オプション, 649
 exit-info オプション, 650
 external-locking オプション, 650
 gdb オプション, 651
 initialize-insecure オプション, 651
 innodb オプション, 2819
 innodb-status-file オプション, 2820
 install-manual オプション, 652
 keyring-migration-destination オプション, 1263
 keyring-migration-host オプション, 1264
 keyring-migration-password オプション, 1264
 keyring-migration-port オプション, 1264
 keyring-migration-socket オプション, 1264
 keyring-migration-source オプション, 1264
 keyring-migration-user オプション, 1265
 lc-messages オプション, 653
 lc-messages-dir オプション, 653
 local-service オプション, 654
 log-bin オプション, 3112
 log-bin-index オプション, 3113
 log-error オプション, 654
 log-isam オプション, 654
 log-raw オプション, 654
 log-short-format オプション, 655
 log-tc オプション, 655
 log-tc-size オプション, 655
 log_slow_admin_statements システム変数, 723
 master-info-file オプション, 3080
 master-retry-count オプション, 3080
 max-binlog-dump-events オプション, 3116

max-relay-log-size オプション, 3081
memlock オプション, 655
mysam-block-size オプション, 656
MySQL Server, 334, 556
NDB Cluster での役割 (参照 [SQL ノード \(NDB Cluster\)](#))
NDB Cluster プロセスとして, 3657, 3889
ndb-allow-copying-alter-table オプション, 3657
ndb-batch-size オプション, 3658
ndb-blob-read-batch-bytes オプション, 3659
ndb-blob-write-batch-bytes オプション, 3660
ndb-cluster-connection-pool オプション, 3658
ndb-cluster-connection-pool-nodeids オプション, 3659
ndb-connectstring オプション, 3660
ndb-log-apply-status, 3662
ndb-log-empty-epochs, 3662
ndb-log-empty-update, 3663
ndb-log-exclusive-reads, 3663
ndb-log-fail-terminate, 3663
ndb-log-orig, 3663
ndb-log-transaction-id, 3664
ndb-nodeid, 3665
ndb-optimization-delay オプション, 3665
ndb-schema-dist-timeout オプション, 3661
ndb-transid-mysql-connection-map オプション, 3666
ndb-wait-connected オプション, 3666
ndb-wait-setup オプション, 3666
ndbcluster オプション, 3657
ndbinfo オプション, 3665
no-dd-upgrade オプション, 656
no-defaults オプション, 656
no-monitor オプション, 657
old-style-user-limits オプション, 657
performance-schema-consumer-events-stages-current オプション, 4418
performance-schema-consumer-events-stages-history オプション, 4418
performance-schema-consumer-events-stages-history-long オプション, 4418
performance-schema-consumer-events-statements-current オプション, 4418
performance-schema-consumer-events-statements-history オプション, 4419
performance-schema-consumer-events-statements-history-long オプション, 4419
performance-schema-consumer-events-transactions-current オプション, 4419
performance-schema-consumer-events-transactions-history オプション, 4419
performance-schema-consumer-events-transactions-history-long オプション, 4419
performance-schema-consumer-events-waits-current オプション, 4419
performance-schema-consumer-events-waits-history オプション, 4419
performance-schema-consumer-events-waits-history-long オプション, 4419
performance-schema-consumer-global-instrumentation オプション, 4419
performance-schema-consumer-statements-digest オプション, 4419
performance-schema-consumer-thread-instrumentation オプション, 4419
performance-schema-consumer-xxx オプション, 4418
performance-schema-instrument オプション, 4418
plugin オプション接頭辞, 659
plugin-load オプション, 658
plugin-load-add オプション, 658
port オプション, 659
port-open-timeout オプション, 660
print-defaults オプション, 660
relay-log-purge オプション, 3081
relay-log-space-limit オプション, 3081
remove オプション, 660
replicate-do-db オプション, 3082
replicate-do-table オプション, 3085

replicate-ignore-db オプション, 3084
replicate-ignore-table オプション, 3086
replicate-rewrite-db オプション, 3086
replicate-same-server-id オプション, 3087
replicate-wild-do-table オプション, 3088
replicate-wild-ignore-table オプション, 3088
safe-user-create オプション, 660
server_uuid 変数, 3066
show-slave-auth-info オプション, 3072
skip-admin-ssl オプション, 645
skip-grant-tables オプション, 660
skip-host-cache オプション, 661
skip-innodb オプション, 662, 2820
skip-ndbcluster オプション, 3667
skip-new オプション, 662
skip-show-database オプション, 662
skip-slave-start オプション, 3089
skip-ssl オプション, 664
skip-stack-trace オプション, 662
skip-symbolic-links オプション, 665
slave-skip-errors オプション, 3089
slave-sql-verify-checksum オプション, 3090
slow-start-timeout オプション, 662
socket オプション, 662
sporadic-binlog-dump-fail オプション, 3116
sql-mode オプション, 663
ssl オプション, 664
super-large-pages オプション, 664
symbolic-links オプション, 665
sysdate-is-no オプション, 665
tc-heuristic-recover オプション, 665
tmpdir オプション, 666
transaction-isolation オプション, 666
transaction-read-only オプション, 666
validate-config オプション, 669
validate-password オプション, 1227
verbose オプション, 669
アップグレードオプション, 667
インストールオプション, 652
起動, 1039
言語オプション, 652
コマンドオプション, 644
コンソールオプション, 646
終了コード, 894
初期化オプション, 651
スタンドアロンオプション, 664
デバッグオプション, 647
バージョンオプション, 669
フラッシュオプション, 650
ヘルプオプション, 645
ユーザーオプション, 669
ラージページオプション, 653
mysqld オプション, 557
 malloc-lib, 337
 mysqld_multi, 344
 mysqld_safe, 338
mysqld システム変数, 557
mysqld のオプションと変数
 NDB Cluster, 3657
mysqld パーサー

mysql クライアントパーサーとの比較, 393
mysqld-auto.cnf オプションファイル, 300, 302, 754, 806, 809, 826, 830, 1341, 2542, 2608, 4359
mysqld-safe-log-timestamps オプション
 mysqld_safe, 337
mysqld-version オプション
 mysqld_safe, 338
MySQLdb, 5346
mysqldump, 5346
mysqldump, 261, 297, 414
 add-drop-database オプション, 425
 add-drop-table オプション, 426
 add-drop-trigger オプション, 426
 add-locks オプション, 436
 all-databases オプション, 433
 all-tables オプション, 426
 allow-keywords オプション, 426
 apply-slave-statements オプション, 428
 bind-address オプション, 422
 character-sets-dir オプション, 428
 column-statistics オプション, 435
 compact オプション, 431
 compatible オプション, 431
 complete-insert オプション, 431
 compress オプション, 422
 compression-algorithms オプション, 422
 create-options オプション, 431
 debug-check オプション, 427
 debug-info オプション, 427
 default-auth オプション, 422
 default-character-set オプション, 428
 defaults-extra-file オプション, 425
 defaults-file オプション, 425
 defaults-group-suffix オプション, 425
 delete-master-logs オプション, 428
 disable-keys オプション, 435
 dump-date オプション, 427
 dump-slave オプション, 428
 enable-cleartext-plugin オプション, 422
 events オプション, 434
 extended-insert オプション, 435
 fields-enclosed-by オプション, 431, 443
 fields-escaped-by オプション, 431, 443
 fields-optionally-enclosed-by オプション, 431, 443
 fields-terminated-by オプション, 431, 443
 flush-logs オプション, 436
 flush-privileges オプション, 436
 get-server-public-key オプション, 422
 hex-blob オプション, 431
 ignore-error オプション, 434
 ignore-table オプション, 434
 include-master-host-port オプション, 429
 insert-ignore オプション, 435
 lines-terminated-by オプション, 431, 444
 lock-all-tables オプション, 436
 lock-tables オプション, 436
 log-error オプション, 427
 login-path オプション, 422
 master-data オプション, 429
 max-allowed-packet オプション, 435
 net-buffer-length オプション, 435

network-timeout オプション, 435
no-autocommit オプション, 437
no-create-db オプション, 426
no-create-info オプション, 426
no-data オプション, 434
no-defaults オプション, 425
no-set-names オプション, 428
no-tablespaces オプション, 426
opt オプション, 435
order-by-primary オプション, 437
pipe オプション, 423
plugin-dir オプション, 423
port オプション, 423
print-defaults オプション, 425
protocol オプション, 423
quote-names オプション, 431
result-file オプション, 431
server-public-key-path オプション, 423
set-charset オプション, 428
set-gtid-purge オプション, 429
shared-memory-base-name オプション, 437
show-create-skip-secondary-engine オプション, 432
skip-comments オプション, 427
skip-opt オプション, 436
socket オプション, 424
SSL オプション, 424
ssl-fips-mode オプション, 424
tls-ciphersuites オプション, 424
tls-version オプション, 424
tz-utc オプション, 432
verbose オプション, 427
where オプション, 434
xml オプション, 432
zstd-compression-level オプション, 424
回避策, 4129
回避方法, 438
強制オプション, 427
クイックオプション, 436
コメントオプション, 426
タブオプション, 432
単一トランザクションオプション, 437
置換オプション, 426
データベースオプション, 433
テーブルオプション, 434
デバッグオプション, 427
トリガーオプション, 434
バージョンオプション, 428
パスワードオプション, 423
バックアップに使用, 1414
ビュー, 438, 4129
ヘルプオプション, 427
ホストオプション, 422
問題, 438, 4129
ユーザーオプション, 424
ルーチンオプション, 434
mysqldumpslow, 298, 544
verbose オプション, 545
デバッグオプション, 545
ヘルプオプション, 545
mysqld_multi, 296, 342

defaults-extra-file オプション, 343
defaults-file オプション, 343
log オプション, 343
mysqladmin オプション, 343
mysqld オプション, 344
no-defaults オプション, 343
no-log オプション, 344
tcp-ip オプション, 344
verbose オプション, 344
オプションの例, 343
サイレントオプション, 344
バージョンオプション, 344
パスワードオプション, 344
ヘルプオプション, 343
ユーザーオプション, 344

MYSQLD_OPTS 環境変数, 177

mysqld_safe, 296, 334

- basedir オプション, 336
- core-file-size オプション, 336
- datadir オプション, 336
- defaults-extra-file オプション, 336
- defaults-file オプション, 336
- ledir オプション, 337
- log-error オプション, 337
- malloc-lib オプション, 337
- mysqld オプション, 338
- mysqld-safe-log-timestamps オプション, 337
- mysqld-version オプション, 338
- nice オプション, 338
- no-defaults オプション, 338
- open-files-limit オプション, 338
- pid-file オプション, 338
- plugin-dir オプション, 338
- port オプション, 338
- skip-kill-mysqld オプション, 339
- skip-syslog オプション, 339
- socket オプション, 339
- syslog オプション, 339
- syslog-tag オプション, 339
- timezone オプション, 339
- ヘルプオプション, 336
- ユーザーオプション, 339

mysqlimport, 261, 297, 439, 2307

- bind-address オプション, 442
- character-sets-dir オプション, 442
- compress オプション, 442
- compression-algorithms オプション, 442
- debug-check オプション, 442
- debug-info オプション, 442
- default-auth オプション, 443
- default-character-set オプション, 443
- defaults-extra-file オプション, 443
- defaults-file オプション, 443
- defaults-group-suffix オプション, 443
- enable-cleartext-plugin オプション, 443
- get-server-public-key オプション, 444
- ignore オプション, 444
- ignore-lines オプション, 444
- lock-tables オプション, 444
- login-path オプション, 444

- no-defaults オプション, 444
- pipe オプション, 445
- plugin-dir オプション, 445
- port オプション, 445
- print-defaults オプション, 445
- protocol オプション, 445
- server-public-key-path オプション, 445
- shared-memory-base-name オプション, 446
- socket オプション, 446
- SSL オプション, 446
- ssl-fips-mode オプション, 446
- tls-ciphersuites オプション, 446
- tls-version オプション, 447
- use-threads オプション, 447
- verbose オプション, 447
- zstd-compression-level オプション, 447
- カラムオプション, 442
- 強制オプション, 443
- サイレントオプション, 446
- 削除オプション, 443
- 置換オプション, 445
- 低優先度オプション, 444
- デバッグオプション, 442
- バージョンオプション, 447
- パスワードオプション, 445
- ヘルプオプション, 442
- ホストオプション, 444
- ユーザーオプション, 447
- ローカルオプション, 444, 1040
- MySQLInstallerConsole, 123
 - modify オプション, 126
 - remove オプション, 126
 - アップグレードオプション, 126
 - インストールオプション, 125
 - 更新オプション, 126
 - 構成オプション, 124
 - ステータスオプション, 126
 - ヘルプオプション, 124
 - リストオプション, 126
- mysqlpump, 297, 447
 - add-drop-database オプション, 453
 - add-drop-table オプション, 453
 - add-drop-user オプション, 453
 - add-locks オプション, 453
 - all-databases オプション, 454
 - bind-address オプション, 454
 - character-sets-dir オプション, 454
 - column-statistics オプション, 454
 - complete-insert オプション, 454
 - compress オプション, 454
 - compress-output オプション, 455
 - compression-algorithms オプション, 455
 - debug-check オプション, 455
 - debug-info オプション, 455
 - default-auth オプション, 455
 - default-character-set オプション, 455
 - default-parallel オプション, 456
 - defaults-extra-file オプション, 456
 - defaults-file オプション, 456
 - defaults-group-suffix オプション, 456

defer-table-indexes オプション, 456
events オプション, 456
exclude-databases オプション, 456
exclude-events オプション, 457
exclude-routines オプション, 457
exclude-tables オプション, 457
exclude-trigger オプション, 457
exclude-users オプション, 457
extended-insert オプション, 457
get-server-public-key オプション, 457
hex-blob オプション, 457
include-databases オプション, 457
include-events オプション, 457
include-routines オプション, 458
include-tables オプション, 458
include-trigger オプション, 458
include-users オプション, 458
insert-ignore オプション, 458
log-error-file オプション, 458
login-path オプション, 458
max-allowed-packet オプション, 458
net-buffer-length オプション, 458
no-create-db オプション, 458
no-create-info オプション, 458
no-defaults オプション, 459
parallel-schemas オプション, 459
plugin-dir オプション, 459
port オプション, 459
print-defaults オプション, 459
protocol オプション, 459
result-file オプション, 460
server-public-key-path オプション, 460
set-charset オプション, 460
set-gtid-purge オプション, 460
skip-definer オプション, 461
skip-dump-rows オプション, 461
socket オプション, 461
SSL オプション, 461
ssl-fips-mode オプション, 462
tls-ciphersuites オプション, 462
tls-version オプション, 462
tz-utc オプション, 462
watch-progress オプション, 463
zstd-compression-level オプション, 463
オブジェクトの選択, 463
制限事項, 465
単一トランザクションオプション, 461
置換オプション, 459
データベースオプション, 455
デバッグオプション, 455
トリガーオプション, 462
バージョンオプション, 463
パスワードオプション, 459
並列処理, 464
ヘルプオプション, 453
ホストオプション, 457
ユーザーオプション, 462, 462
ルーチンオプション, 460
mysqlsh, 297
mysqlshow, 298, 465

bind-address オプション, 468
character-sets-dir オプション, 468
compress オプション, 468
compression-algorithms オプション, 468
debug-check オプション, 468
debug-info オプション, 468
default-auth オプション, 469
default-character-set オプション, 469
defaults-extra-file オプション, 469
defaults-file オプション, 469
defaults-group-suffix オプション, 469
enable-cleartext-plugin オプション, 469
get-server-public-key オプション, 469
keys オプション, 470
login-path オプション, 470
no-defaults オプション, 470
pipe オプション, 470
plugin-dir オプション, 470
port オプション, 470
print-defaults オプション, 470
protocol オプション, 471
server-public-key-path オプション, 471
shared-memory-base-name オプション, 471
show-table-type オプション, 471
socket オプション, 471
SSL オプション, 471
ssl-fips-mode オプション, 471
tls-ciphersuites オプション, 472
tls-version オプション, 472
verbose オプション, 472
zstd-compression-level オプション, 472
カウントオプション, 468
ステータスオプション, 472
デバッグオプション, 468
バージョンオプション, 472
パスワードオプション, 470
ヘルプオプション, 468
ホストオプション, 470
ユーザーオプション, 472
mysqlslap, 298, 472
auto-generate-sql オプション, 477
auto-generate-sql-add-autoincrement オプション, 477
auto-generate-sql-execute-number オプション, 477
auto-generate-sql-guid-primary オプション, 477
auto-generate-sql-load-type オプション, 477
auto-generate-sql-secondary-indexes オプション, 477
auto-generate-sql-unique-query-number オプション, 477
auto-generate-sql-unique-write-number オプション, 477
auto-generate-sql-write-number オプション, 477
compress オプション, 477
compression-algorithms オプション, 477
create-schema オプション, 478
csv オプション, 478
debug-check オプション, 478
debug-info オプション, 478
default-auth オプション, 478
defaults-extra-file オプション, 478
defaults-file オプション, 478
defaults-group-suffix オプション, 479
delimiter オプション, 479

enable-clear-text-plugin オプション, 479
get-server-public-key オプション, 479
login-path オプション, 479
no-defaults オプション, 480
no-drop オプション, 480
number-char-cols オプション, 480
number-int-cols オプション, 480
number-of-queries オプション, 480
only-print オプション, 480
pipe オプション, 480
plugin-dir オプション, 480
port オプション, 481
post-system オプション, 481
pre-system オプション, 481
print-defaults オプション, 481
protocol オプション, 481
server-public-key-path オプション, 481
shared-memory-base-name オプション, 481
socket オプション, 482
sql-mode オプション, 482
SSL オプション, 482
ssl-fips-mode オプション, 482
tls-ciphersuites オプション, 482
tls-version オプション, 482
verbose オプション, 482
zstd-compression-level オプション, 483
エンジンオプション, 479
クエリーオプション, 481
クエリー後オプション, 481
コミットオプション, 477
サイレントオプション, 482
作成オプション, 478
事前クエリーオプション, 481
デバッグオプション, 478
同時実行オプション, 478
バージョンオプション, 483
パスワードオプション, 480
反復オプション, 479
ヘルプオプション, 477
ホストオプション, 479
ユーザーオプション, 482
連結解除オプション, 479
mysqlx X プラグイン オプション, 3421
mysqlx_bind_address システム変数, 3422
mysqlx_compression_algorithms システム変数, 3423
mysqlx_connect_timeout システム変数, 3424
mysqlx_deflate_default_compression_level システム変数, 3424
mysqlx_deflate_max_client_compression_level システム変数, 3424
mysqlx_document_id_unique_prefix システム変数, 3425
mysqlx_enable_hello_notice システム変数, 3425
mysqlx_idle_worker_thread_timeout システム変数, 3425
mysqlx_interactive_timeout システム変数, 3425
mysqlx_lz4_default_compression_level システム変数, 3426
mysqlx_lz4_max_client_compression_level システム変数, 3426
mysqlx_max_allowed_packet システム変数, 3426
mysqlx_max_connections システム変数, 3427
mysqlx_min_worker_threads システム変数, 3427
mysqlx_port システム変数, 3427
mysqlx_port_open_timeout システム変数, 3428
mysqlx_read_timeout システム変数, 3428

mysqlx_socket システム変数, 3428
mysqlx_ssl_ca システム変数, 3429
mysqlx_ssl_capath システム変数, 3429
mysqlx_ssl_cert システム変数, 3430
mysqlx_ssl_cipher システム変数, 3430
mysqlx_ssl_crl システム変数, 3430
mysqlx_ssl_crlpath システム変数, 3430
mysqlx_ssl_key システム変数, 3431
MYSQLX_TCP_PORT オプション
 CMake, 208
MYSQLX_TCP_PORT 環境変数, 550
MYSQLX_UNIX_ADDR オプション
 CMake, 208
MYSQLX_UNIX_PORT 環境変数, 550
mysqlx_wait_timeout システム変数, 3431
mysqlx_write_timeout システム変数, 3431
mysqlx_zstd_default_compression_level システム変数, 3432
mysqlx_zstd_max_client_compression_level システム変数, 3432
mysql_acquire_locking_service_locks() C 関数
 ロックサービス, 1001
mysql_clear_password 認証プラグイン, 1161
mysql_config, 546
 cflags オプション, 546
 cxxflags オプション, 546
 include オプション, 546
 libs オプション, 546
 libs_r オプション, 546
 plugindir オプション, 547
 port オプション, 547
 socket オプション, 547
 バージョンオプション, 547
 変数オプション, 547
mysql_config_editor, 298, 514
 verbose オプション, 518
 デバッグオプション, 517
 バージョンオプション, 518
 ヘルプオプション, 517
mysql_config_server, 546
MYSQL_DATADIR オプション
 CMake, 202
MYSQL_DEBUG 環境変数, 299, 550, 1022
mysql_firewall_flush_status() MySQL Enterprise Firewall UDF, 1370
mysql_firewall_mode システム変数, 1370
mysql_firewall_trace システム変数, 1371
MYSQL_FIREWALL_USERS
 INFORMATION_SCHEMA テーブル, 4233
MYSQL_FIREWALL_WHITELIST
 INFORMATION_SCHEMA テーブル, 4234
MYSQL_GROUP_SUFFIX 環境変数, 550
MYSQL_HISTFILE 環境変数, 386, 550
MYSQL_HISTIGNORE 環境変数, 386, 550
MYSQL_HOME 環境変数, 550
MYSQL_HOST 環境変数, 321, 550
mysql_info(), 2167, 2302, 2316, 2362
mysql_insert_id(), 2302
mysql_install_db
 削除された機能, 41
mysql_keyring サービス, 1005
MYSQL_MAINTAINER_MODE オプション
 CMake, 208

mysql_native_password 認証プラグイン, 1151
mysql_native_password_proxy_users システム変数, 740, 1121
mysql_no_login 認証プラグイン, 1194
MYSQL_OPENSSL_UDF_DH_BITS_THRESHOLD 環境変数, 550, 1391
MYSQL_OPENSSL_UDF_DSA_BITS_THRESHOLD 環境変数, 550, 1391
MYSQL_OPENSSL_UDF_RSA_BITS_THRESHOLD 環境変数, 550, 1391
mysql_options()
 MYSQL_OPT_LOAD_DATA_LOCAL_DIR, 1041
 MYSQL_OPT_LOCAL_INFILE, 1040, 1041
MYSQL_OPT_COMPRESS
 非推奨となった機能, 37
MYSQL_OPT_SSL_ENFORCE
 削除された機能, 40
MYSQL_OPT_SSL_VERIFY_SERVER_CERT
 削除された機能, 40
mysql_plugin
 削除された機能, 42
MYSQL_PROJECT_NAME オプション
 CMake, 208
MYSQL_PS1 環境変数, 550
MYSQL_PWD
 非推奨となった機能, 37
MYSQL_PWD 環境変数, 550
mysql_query_attribute_string() C API UDF, 1682
mysql_real_escape_string_quote(), 1629, 1909
mysql_release_locking_service_locks() C 関数
 ロックサービス, 1002
MYSQL_SECURE_AUTH
 削除された機能, 39
mysql_secure_installation, 296, 347
 defaults-extra-file オプション, 348
 defaults-file オプション, 349
 defaults-group-suffix オプション, 349
 no-defaults オプション, 349
 port オプション, 349
 print-defaults オプション, 349
 protocol オプション, 349
 socket オプション, 349
 SSL オプション, 350
 ssl-fips-mode オプション, 350
 tls-ciphersuites オプション, 350
 tls-version オプション, 350
 use-default オプション, 350
 パスワードオプション, 349
 ヘルプオプション, 348
 ホストオプション, 349
 ユーザーオプション, 350
mysql_ssl_rsa_setup, 297, 350
 datadir オプション, 352
 uid オプション, 353
 verbose オプション, 353
 接尾辞オプション, 352
 バージョンオプション, 353
 ヘルプオプション, 352
MYSQL_TCP_PORT オプション
 CMake, 208
MYSQL_TCP_PORT 環境変数, 299, 550, 1015, 1016
MYSQL_TEST_LOGIN_FILE 環境変数, 308, 515, 550
MYSQL_TEST_TRACE_CRASH 環境変数, 550
MYSQL_TEST_TRACE_DEBUG 環境変数, 550

mysql_tzinfo_to_sql, 297, 353
MYSQL_UNIX_ADDR オプション
 CMake, 208
MYSQL_UNIX_PORT 環境変数, 299, 550, 1015, 1016
mysql_upgrade, 297, 353
 bind-address オプション, 358
 character-sets-dir オプション, 358
 compress オプション, 358
 compression-algorithms オプション, 358
 debug-check オプション, 358
 debug-info オプション, 358
 default-auth オプション, 358
 default-character-set オプション, 358
 defaults-extra-file オプション, 358
 defaults-file オプション, 359
 defaults-group-suffix オプション, 359
 get-server-public-key オプション, 359
 login-path オプション, 359
 max-allowed-packet オプション, 359
 mysql_upgrade_info ファイル, 237, 355
 net-buffer-length オプション, 359
 no-defaults オプション, 359
 pipe オプション, 360
 plugin-dir オプション, 360
 port オプション, 360
 print-defaults オプション, 360
 protocol オプション, 360
 server-public-key-path オプション, 360
 shared-memory-base-name オプション, 361
 skip-sys-schema オプション, 361
 socket オプション, 361
 SSL オプション, 361
 ssl-fips-mode オプション, 361
 tls-ciphersuites オプション, 361
 tls-version オプション, 362
 upgrade-system-tables オプション, 362
 verbose オプション, 362
 version-check オプション, 362
 write-binlog オプション, 362
 zstd-compression-level オプション, 362
 強制オプション, 359
 デバッグオプション, 358
 パスワードオプション, 360
 非推奨となった機能, 37
 ヘルプオプション, 358
 ホストオプション, 359
 ユーザーオプション, 362
mysql_upgrade_info ファイル
 mysql_upgrade, 237, 355
 非推奨となった機能, 37
my_key_fetch() キーリングサービス関数, 1006
my_key_generate() キーリングサービス関数, 1006
my_key_remove() キーリングサービス関数, 1007
my_key_store() キーリングサービス関数, 1007
my_print_defaults, 298, 547
 config-file オプション, 548
 defaults-extra-file オプション, 548
 defaults-file オプション, 548
 defaults-group-suffix オプション, 548
 extra-file オプション, 548

login-path オプション, 548
no-defaults オプション, 548
show オプション, 548
verbose オプション, 548
デバッグオプション, 548
バージョンオプション, 548
ヘルプオプション, 547

N

name-file オプション
 comp_err, 347
named-commands オプション
 mysql, 373
named_pipe システム変数, 741
named_pipe_full_access_group システム変数, 741
namespaces
 network, 875
NAME_CONST(), 2138, 4124
NATIONAL CHAR データ型, 1782
NATIONAL VARCHAR データ型, 1783
NATURAL INNER JOIN, 2334
NATURAL JOIN, 2334
NATURAL LEFT JOIN, 2334
NATURAL LEFT OUTER JOIN, 2334
NATURAL RIGHT JOIN, 2334
NATURAL RIGHT OUTER JOIN, 2334
NCHAR データ型, 1782
NDB API カウンタ (NDB Cluster), 3900
 types, 3903
 関連付けられたステータス変数, 3905
 スコープ, 3903
NDB API データベースオブジェクト
 NDB Cluster レプリケーション, 3992
NDB API レプリカステータス変数
 NDB Cluster レプリケーション, 3991
NDB Cluster 8.0, 3456
NDB Cluster Auto-Installer (サポートされなくなりました), 3518
 and setup.bat (Windows), 3522
 アーキテクチャ, 3518
 および ndb_setup.py, 3522
 および Python, 3518
 起動, 3521
 「クラスタの定義」画面, 3524
 「構成のデプロイ」画面, 3536
 サポートされる web ブラウザ, 3518
 サポートされるプラットフォーム, 3518
 使用, 3519
 セキュリティの問題, 3519
 ソフトウェア要件, 3518
 「パラメータの定義」画面, 3534
 プロセスの削除, 3534
 プロセスの追加, 3533
 「プロセスの定義」画面, 3532
 ホストの追加と削除, 3529
 「ホストの定義」画面, 3527
 要件, 3518
 ようこそ画面, 3522
 リモートホストでの認証, 3519
 リモートホストとローカルホスト, 3519, 3519

NDB Cluster, 3446
 .deb ファイルのインストール (Linux), 3500
 API ノード, 3450, 3648
 Auto-Installer による配備 (サポートされなくなりました), 3518
 BACKUP イベント, 3866
 backups, 3788, 3884, 3884, 3884, 3887, 3888
 CHECKPOINT イベント, 3861
 CLUSTERLOG STATISTICS コマンド, 3867
 CLUSTERLOG コマンド, 3859
 configuration, 3492, 3540, 3541, 3568, 3569, 3575, 3648, 3741, 3889
 CONNECT コマンド, 3839
 CONNECTION イベント, 3861
 CREATE NODEGROUP コマンド, 3842
 DROP NODEGROUP コマンド, 3842
 ENTER SINGLE USER MODE コマンド, 3841
 ERROR イベント, 3866
 EXIT SINGLE USER MODE コマンド, 3841
 EXIT コマンド, 3841
 FAQ, 4551
 FULLY_REPLICATED (NDB_TABLE), 2267
 GCP 停止エラー, 3644
 HELP コマンド, 3839
 HostName パラメータ
 セキュリティ, 3983
 INFO イベント, 3866
 InnoDB との比較, 3479, 3479, 3480, 3481
 IP アドレス指定, 3493
 Java クライアント, 3451
 JSON, 2262
 MAX_ROWS, 2233
 mgm, 3834
 mgm 管理クライアント, 3867
 mgm クライアント, 3839
 mgm プロセス, 3741
 mgmd, 3834
 mgmd プロセス, 3734
 MySQL root ユーザー, 3987, 3989
 MySQL 権限, 3987
 mysqld のオプションと変数, 3657
 mysqld プロセス, 3657, 3889
 nbd, 3725, 3834
 nbd プロセス, 3725, 3869
 nbdinfo_select_all, 3731
 nbdmtd, 3733
 ndb_mgm, 3511, 3741
 ndb_mgmd プロセス, 3734
 NODELOG DEBUG コマンド, 3843
 NODERESTART イベント, 3863
 NOLOGGING (NDB_TABLE), 2266
 PARTITION_BALANCE (NDB_TABLE), 2267
 PROMPT コマンド, 3843
 QUIT コマンド, 3841
 READ_BACKUP (NDB_TABLE), 2266
 REPORT コマンド, 3840
 RESTART コマンド, 3840
 RPM (Linux) のインストール, 3496
 SCHEMA イベント, 3865
 security, 3982
 HostName パラメータ, 3983
 ネットワーキング, 3983

- ネットワーク構成, 3983
- ネットワークポート, 3986
- ファイアウォール, 3984, 3986
- リモート管理, 3987
- SHOW コマンド, 3839
- SHUTDOWN コマンド, 3842
- SINGLEUSER イベント, 3866
- SQL ノード, 3450, 3648, 3889
- START BACKUP コマンド, 4007
- START コマンド, 3839
- STARTUP イベント, 3862
- STATISTICS イベント, 3865
- STATUS コマンド, 3840
- STOP コマンド, 3839
- USING HASH, 2205
- アップグレードとダウングレード, 3516, 3871
- アプリケーション機能の要件, 3481
- 一般説明, 3448
- イベントタイプ, 3858, 3861
- イベントの重大度, 3860
- イベントロギングしきい値, 3860
- イベントログ, 3858, 3859
- イベントログ形式, 3861
- インストール (Linux), 3494
- インストール (Windows), 3502
- インストール, 3492
- インターコネクト, 3724
- エラーログ, 3730
- 大きいテーブル, 2233
- および DNS, 3493
- および INFORMATION_SCHEMA, 3988
- 開始フェーズ (要約), 3869
- 概念, 3450
- 可用性, 3479
- 監視, 3900
- 監視用の SQL ステートメント, 3976
- 間接インデックス, 2262
- 管理, 3657, 3725, 3734, 3741, 3741, 3834, 3838, 3839, 3867
- 管理クライアント (ndb_mgm), 3741
- 管理コマンド, 3867
- 管理ノード, 3450, 3568, 3734
- 記憶域要件, 1827
- 起動, 3541
- 起動または再起動, 3869
- 共有メモリートランスポート, 3714
- クイック構成, 3541
- クエリーの実行, 3512
- クラスタログ, 3858, 3859
- 構成 (例), 3561
- 構成, 3887
- 構成の変更, 3871
- 構成パラメータ, 3543, 3543, 3549, 3550, 3551
- 構成ファイル, 3509, 3560
- コマンド, 3657, 3725, 3734, 3741, 3839
- 再起動, 3515
- サポートされるアプリケーション, 3480
- 実行時統計, 3867
- 実行スレッド, 3628
- シャットダウン, 3515
- 使用可能なプラットフォーム, 3447

- 情報ソース, 3448
- スタンドアロン MySQL Server との比較, 3479, 3479, 3480, 3481
- ステータス変数, 3689
- スレッドの状態, 1624
- セキュリティ手順, 3988
- 接続文字列, 3566
- ソースからのインストール (Linux), 3500
- ソースからのインストール (Windows), 3505
- 単一ユーザーモード, 3841, 3873
- 通信プロトコルのセキュリティの確保, 3983
- 「ディスクデータ」テーブル (参照 [NDB Cluster ディスクデータ](#))
- データノード, 3450, 3575, 3725, 3733
- テーブルおよびデータの使用, 3512
- トランザクション, 3581
- トランザクション処理, 3487
- トランザクション分離レベル, 3485
- トランスポート
 - TCP/IP, 3714
 - 共有メモリー (SHM), 3714
- トレースファイル, 3731
- ネットワーキング, 3455, 3714, 3714
- ネットワーク構成
 - セキュリティ, 3983
- ネットワークトランスポート, 3724
- ノードおよびタイプ, 3450
- ノード間の直接接続, 3714
- ノード識別子, 3717, 3717
- ノード障害 (シングルユーザーモード), 3873
- ノードとノードグループ, 3452
- ノードの起動, 3505, 3511
- ノードホストの定義, 3568
- ノードログ, 3858
- パーティション, 3452
- パーティション化のサポート, 3483
- バイナリのインストール (Windows), 3502
- バイナリリリースのインストール (Linux), 3494
- バックアップのトラブルシューティング, 3888
- バックアップのリストア, 3788
- 複数の管理サーバー, 3872
- フラグメントレプリカ, 3452
- プロセス管理, 3724
- メモリー使用量およびリカバリ, 3484, 3872
- 要件, 3455
- リセット, 3871
- レプリケーション, 3989
 - (参照 [NDB Cluster レプリケーション](#))
- レプリケーションの準備, 4001
- ローリング再起動 (複数の管理サーバー), 3872
- ロギングコマンド, 3859
- ログファイル, 3730, 3733
- NDB Cluster 自動インストーラ
 - セットアッププログラム (Windows), 3818
 - セットアッププログラム, 3818
- NDB Cluster ディスクデータ, 3890
 - 記憶域要件, 3895
 - ディスクデータオブジェクトの削除, 3895
 - テーブルスペースの作成, 3892
 - テーブルの作成, 3891, 3893
 - ログファイルグループの作成, 3891
- NDB Cluster に関連する SQL ステートメント, 3976

- NDB Cluster のインストール, 3492
 - Debian Linux, 3500
 - Linux, 3494
 - Linux RPM, 3496
 - Linux ソースリリース, 3500
 - Linux バイナリリリース, 3494
 - Ubuntu Linux, 3500
 - Windows, 3502
 - Windows ソース, 3505
 - Windows バイナリリリース, 3502
- NDB Cluster の開発, 3456
- NDB Cluster の管理, 3741, 3838
- NDB Cluster の構成 (概念), 3450
- NDB Cluster の構成, 3492, 3540, 3741, 3889
- NDB Cluster の新機能, 3456
- NDB Cluster の制限, 3481, 3481
 - JSON カラム, 3484
 - partitioning, 3483
 - performance, 3489
 - transactions, 3485
 - 以前のバージョンの現在のバージョンで解決済, 3491
 - エラー処理およびレポート, 3487
 - 構成による強制, 3484
 - 構文, 3482
 - サポートされない機能, 3488
 - ジオメトリデータ型, 3483
 - 実装, 3489
 - ディスクデータストレージ, 3490
 - データベースオブジェクト, 3488
 - バイナリロギング, 3489
 - 標準の MySQL 制限との相違点, 3484
 - 複数の MySQL サーバー, 3491
 - 複数の管理サーバー, 3491
 - メモリー使用量およびトランザクション処理, 3487
 - レプリケーション, 3484
- NDB Cluster のセキュリティー保護, 3988
- NDB Cluster の使い方, 3492
- NDB Cluster プログラム, 3724
- NDB Cluster プログラムの使用, 3724
- NDB Cluster プロセス, 3724
- NDB Cluster プロセスの管理, 3724
- NDB Cluster レプリケーション, 3990
 - initial オプション, 3996
 - backups, 4007
 - failover, 4005, 4005
 - NDB API データベースオブジェクト, 3992
 - NDB API レプリカステータス変数, 3991
 - NDB 以外のストレージエンジン (レプリカ上), 3997
 - point-in-time リカバリ, 4012
 - reset-replica.pl バックアップ自動化スクリプト, 4009
 - 一意キー, 3995
 - 概念, 3991, 3991
 - 既知の問題, 3992
 - 起動, 4003
 - ギャップイベント, 3993
 - 競合解消, 4016
 - 主キー, 3995
 - 循環レプリケーション, 3993, 4013
 - 準備, 4001
 - 使用されるシステムテーブル, 3999

- シングルポイント障害, 4005
- 接続の損失, 3992
- 双方向レプリケーション, 4013
- ソースとレプリカの同期, 4009
- バックアップからのリストア, 4007
- 要件, 3991
- 読取り競合の検出および解決, 4026
- NDB Cluster レプリケーション競合解決
 - 例外テーブル, 4023
- ndb オプション
 - ndb_perror, 3783
 - pererror, 549
- NDB クライアントプログラム
 - defaults-extra-file オプション, 3836
 - defaults-file オプション, 3836
 - defaults-group-suffix オプション, 3836
 - login-path オプション, 3836
 - no-defaults オプション, 3837
 - print-defaults オプション, 3837
- NDB ストレージエンジン (参照 [NDB Cluster](#))
 - FAQ, 4551
- NDB テーブル
 - MySQL root ユーザー, 3987
- NDB 統計変数 (NDB Cluster), 3900
 - types, 3903
 - スコープ, 3903
- NDB 統計変数
 - NDB API カウンタ, 3905
- NDB バックアップの復元
 - NDB の新しいバージョンへ, 3810
 - NDB の以前のバージョンへ, 3809
 - NDB リリースシリーズ間, 3809
- NDB ユーティリティー
 - セキュリティの問題, 3989
- NDB\$CFT_CAUSE, 4024
- NDB\$EPOCH(), 4020
 - 制限事項, 4021
- NDB\$EPOCH2(), 4022
- NDB\$EPOCH2_TRANS(), 4022
- NDB\$EPOCH_TRANS(), 4020, 4022
- NDB\$MAX(), 4020
- NDB\$MAX_DELETE_WIN(), 4020
- NDB\$OLD, 4019
- NDB\$OP_TYPE, 4024
- NDB\$ORIG_TRANSID, 4024
- ndb-allow-copying-alter-table オプション
 - mysqld, 3657
- ndb-batch-size オプション
 - mysqld, 3658
- ndb-blob-read-batch-bytes オプション
 - mysqld, 3659
- ndb-blob-write-batch-bytes オプション
 - mysqld, 3660
- ndb-cluster-connection-pool オプション
 - mysqld, 3658
- ndb-cluster-connection-pool-nodeids オプション
 - mysqld, 3659
- ndb-connectstring オプション
 - mysqld, 3660
 - ndb_config, 3749

ndb-default-column-format オプション (NDB Cluster), 3660
ndb-deferred-constraints オプション (NDB Cluster), 3661
NDB-distribution オプション (NDB Cluster), 3661
NDB-gmd-host オプション (NDB Cluster プログラム), 3837
NDB-gmd-host オプション (NDB Cluster), 3664
ndb-log-apply-status オプション
mysql, 3662
ndb-log-empty-epochs オプション
mysql, 3662
ndb-log-empty-update オプション
mysql, 3663
ndb-log-exclusive-reads オプション
mysql, 3663
ndb-log-fail-terminate オプション
mysql, 3663
ndb-log-orig オプション
mysql, 3663
ndb-log-transaction-id オプション
mysql, 3664
ndb-log-update-as-write (mysql オプション), 4017
ndb-log-update-minimal オプション (NDB Cluster), 3664
NDB-nectstring オプション (NDB Cluster プログラム), 3837
ndb-nodegroup-map オプション
ndb_restore, 3800
NDB-nodeid オプション (NDB Cluster プログラム), 3837
ndb-nodeid オプション
mysql, 3665
ndb-optimization-delay オプション
mysql, 3665
ndb-optimized-node-selection オプション (NDB Cluster), 3837
ndb-schema-dist-timeout オプション
mysql, 3661
ndb-transid-mysql-connection-map オプション
mysql, 3666
ndb-wait-connected オプション
mysql, 3666
ndb-wait-setup オプション
mysql, 3666
ndbcluster オプション
mysql, 3657
NDBCLUSTER ストレージエンジン (参照 [NDB Cluster](#))
ndbd (NDB Cluster)
定義済, 3450
(参照 [データノード \(NDB Cluster\)](#))
ndbd, 3724, 3724
-n オプション, 3729
connect-delay オプション, 3726
connect-retries オプション, 3726
connect-retry-delay オプション, 3726
initial-start オプション, 3728
logbuffer-size オプション, 3729
nostart オプション, 3729
nowait-nodes オプション, 3729
remove オプション, 3730
verbose オプション, 3730
インストールオプション, 3728
初期オプション, 3727
ndbinfo オプション
mysql, 3665
ndbinfo データベース, 3910

およびクエリーキャッシュ, 3912
基本的な使用方法, 3913
サポートの判断, 3910

ndbinfo_database システム変数, 3687
ndbinfo_max_bytes システム変数, 3687
ndbinfo_max_rows システム変数, 3687
ndbinfo_offline システム変数, 3688
ndbinfo_select_all, 3724, 3731
 -I オプション, 3732
 delay オプション, 3732
 loops オプション, 3732

ndbinfo_show_hidden システム変数, 3688
ndbinfo_table_prefix システム変数, 3688
ndbinfo_version システム変数, 3688, 3688
ndbmtd, 3724, 3733
 -n オプション, 3729
 ClassicFragmentation, 3628
 configuration, 3631, 3632
 connect-delay オプション, 3726
 connect-retries オプション, 3726
 connect-retry-delay オプション, 3726
 initial-start オプション, 3728
 logbuffer-size オプション, 3729
 MaxNoOfExecutionThreads, 3628
 nstart オプション, 3729
 nowait-nodes オプション, 3729
 remove オプション, 3730
 verbose オプション, 3730
 インストールオプション, 3728
 初期オプション, 3727
 トレースファイル, 3733, 3733

ndbxfrm, 3724, 3832
 compress オプション, 3833
 decrypt-password オプション, 3833
 encrypt-kdf-iter-count オプション, 3833
 encrypt-password オプション, 3833
 info オプション, 3833
 usage オプション, 3834
 バージョンオプション, 3834
 ヘルプオプション, 3833

Ndb_api_adaptive_send_deferred_count ステータス変数, 3689
Ndb_api_adaptive_send_deferred_count_replica ステータス変数, 3689
Ndb_api_adaptive_send_deferred_count_session ステータス変数, 3689
Ndb_api_adaptive_send_deferred_count_slave ステータス変数, 3689
Ndb_api_adaptive_send_forced_count ステータス変数, 3689
Ndb_api_adaptive_send_forced_count_replica ステータス変数, 3689
Ndb_api_adaptive_send_forced_count_session ステータス変数, 3689
Ndb_api_adaptive_send_forced_count_slave ステータス変数, 3689
Ndb_api_adaptive_send_unforced_count ステータス変数, 3690
Ndb_api_adaptive_send_unforced_count_replica ステータス変数, 3690
Ndb_api_adaptive_send_unforced_count_slave ステータス変数, 3690
Ndb_api_adaptive_send_unforced_count_slave セッション変数, 3690
Ndb_api_bytes_received_count ステータス変数, 3692
Ndb_api_bytes_received_count_replica ステータス変数, 3691
Ndb_api_bytes_received_count_session ステータス変数, 3691
Ndb_api_bytes_received_count_slave ステータス変数, 3691
Ndb_api_bytes_sent_count ステータス変数, 3691
Ndb_api_bytes_sent_count_replica ステータス変数, 3690
Ndb_api_bytes_sent_count_session ステータス変数, 3690
Ndb_api_bytes_sent_count_slave ステータス変数, 3691

Ndb_api_event_bytes_count ステータス変数, 3692
Ndb_api_event_bytes_count_injector ステータス変数, 3692
Ndb_api_event_data_count ステータス変数, 3692
Ndb_api_event_data_count_injector ステータス変数, 3692
Ndb_api_event_nondata_count ステータス変数, 3692
Ndb_api_event_nondata_count_injector ステータス変数, 3692
Ndb_api_pk_op_count ステータス変数, 3693
Ndb_api_pk_op_count_replica ステータス変数, 3693
Ndb_api_pk_op_count_session ステータス変数, 3693
Ndb_api_pk_op_count_slave ステータス変数, 3693
Ndb_api_pruned_scan_count ステータス変数, 3694
Ndb_api_pruned_scan_count_replica ステータス変数, 3694
Ndb_api_pruned_scan_count_session ステータス変数, 3693
Ndb_api_pruned_scan_count_slave ステータス変数, 3694
Ndb_api_range_scan_count ステータス変数, 3695
Ndb_api_range_scan_count_replica ステータス変数, 3694
Ndb_api_range_scan_count_session ステータス変数, 3694
Ndb_api_range_scan_count_slave ステータス変数, 3695
Ndb_api_read_row_count ステータス変数, 3696
Ndb_api_read_row_count_replica ステータス変数, 3695
Ndb_api_read_row_count_session ステータス変数, 3695
Ndb_api_read_row_count_slave ステータス変数, 3695
Ndb_api_scan_batch_count ステータス変数, 3696
Ndb_api_scan_batch_count_replica ステータス変数, 3696
Ndb_api_scan_batch_count_session ステータス変数, 3696
Ndb_api_scan_batch_count_slave ステータス変数, 3696
Ndb_api_table_scan_count ステータス変数, 3697
Ndb_api_table_scan_count_replica ステータス変数, 3697
Ndb_api_table_scan_count_session ステータス変数, 3697
Ndb_api_table_scan_count_slave ステータス変数, 3697
Ndb_api_trans_abort_count ステータス変数, 3698
Ndb_api_trans_abort_count_replica ステータス変数, 3697
Ndb_api_trans_abort_count_session ステータス変数, 3697
Ndb_api_trans_abort_count_slave ステータス変数, 3698
Ndb_api_trans_close_count ステータス変数, 3699
Ndb_api_trans_close_count_replica ステータス変数, 3698
Ndb_api_trans_close_count_session ステータス変数, 3698
Ndb_api_trans_close_count_slave ステータス変数, 3698
Ndb_api_trans_commit_count ステータス変数, 3699
Ndb_api_trans_commit_count_replica ステータス変数, 3699
Ndb_api_trans_commit_count_session ステータス変数, 3699
Ndb_api_trans_commit_count_slave ステータス変数, 3699
Ndb_api_trans_local_read_row_count ステータス変数, 3700
Ndb_api_trans_local_read_row_count_replica ステータス変数, 3699
Ndb_api_trans_local_read_row_count_session ステータス変数, 3699
Ndb_api_trans_local_read_row_count_slave ステータス変数, 3700
Ndb_api_trans_start_count ステータス変数, 3701
Ndb_api_trans_start_count_replica ステータス変数, 3700
Ndb_api_trans_start_count_session ステータス変数, 3700
Ndb_api_trans_start_count_slave ステータス変数, 3701
Ndb_api_uk_op_count ステータス変数, 3702
Ndb_api_uk_op_count_replica ステータス変数, 3701
Ndb_api_uk_op_count_session ステータス変数, 3701
Ndb_api_uk_op_count_slave ステータス変数, 3701
Ndb_api_wait_exec_complete_count ステータス変数, 3702
Ndb_api_wait_exec_complete_count_replica ステータス変数, 3702
Ndb_api_wait_exec_complete_count_session ステータス変数, 3702
Ndb_api_wait_exec_complete_count_slave ステータス変数, 3702
Ndb_api_wait_meta_request_count ステータス変数, 3703
Ndb_api_wait_meta_request_count_replica ステータス変数, 3703

Ndb_api_wait_meta_request_count_session ステータス変数, 3702
Ndb_api_wait_meta_request_count_slave ステータス変数, 3703
Ndb_api_wait_nanos_count ステータス変数, 3704
Ndb_api_wait_nanos_count_replica ステータス変数, 3703
Ndb_api_wait_nanos_count_session ステータス変数, 3703
Ndb_api_wait_nanos_count_slave ステータス変数, 3704
Ndb_api_wait_scan_result_count ステータス変数, 3705
Ndb_api_wait_scan_result_count_replica ステータス変数, 3704
Ndb_api_wait_scan_result_count_session ステータス変数, 3704
Ndb_api_wait_scan_result_count_slave ステータス変数, 3704
ndb_apply_status テーブル (NDB Cluster レプリケーション), 4000, 4001, 4006
(参照 [NDB Cluster レプリケーション](#))
ndb_autoincrement_prefetch_sz システム変数, 3667
ndb_binlog_index テーブル (NDB Cluster レプリケーション), 3999, 4006
(参照 [NDB Cluster レプリケーション](#))
ndb_binlog_index テーブル
システムテーブル, 899, 3999
ndb_blob_tool, 3724, 3743
add-missing オプション, 3744
check-missing オプション, 3744
check-orphans オプション, 3744
delete-orphans オプション, 3744
dump-file オプション, 3744
verbose オプション, 3744
データベースオプション, 3744
ndb_cache_check_time システム変数, 3667
ndb_clear_apply_status システム変数, 3668
Ndb_cluster_node_id ステータス変数, 3705
ndb_config, 3724, 3745
cluster-config-suffix オプション, 3747
config-file オプション, 3748
configinfo オプション, 3747
config_from_node オプション, 3748
connections オプション, 3748
diff-default オプション, 3748
mycnf オプション, 3749
ndb-connectstring オプション, 3749
nodeid オプション, 3749
nodes オプション, 3749
query-all オプション, 3750
rows オプション, 3750
usage オプション, 3751
xml オプション, 3751
クエリーオプション, 3750, 3750
システムオプション, 3750
タイプオプション, 3751
バージョンオプション, 3751
フィールドオプション, 3748
ホストオプション, 3749
Ndb_config_from_host ステータス変数, 3705
Ndb_config_from_port ステータス変数, 3705
Ndb_config_generation ステータス変数, 3705
Ndb_conflict_fn_epoch ステータス変数, 3705
Ndb_conflict_fn_epoch2 ステータス変数, 3706
Ndb_conflict_fn_epoch2_trans ステータス変数, 3706
Ndb_conflict_fn_epoch_trans ステータス変数, 3706
Ndb_conflict_fn_max ステータス変数, 3705
Ndb_conflict_fn_max_del_win ステータス変数, 3705
Ndb_conflict_fn_old ステータス変数, 3705
Ndb_conflict_last_conflict_epoch ステータス変数, 3706

Ndb_conflict_last_stable_epoch ステータス変数, 3706
Ndb_conflict_reflected_op_discard_count ステータス変数, 3706
Ndb_conflict_reflected_op_prepare_count ステータス変数, 3706
Ndb_conflict_refresh_op_count ステータス変数, 3706
ndb_conflict_role システム変数, 3668
Ndb_conflict_trans_conflict_commit_count ステータス変数, 3707
Ndb_conflict_trans_detect_iter_count ステータス変数, 3707
Ndb_conflict_trans_reject_count ステータス変数, 3707
Ndb_conflict_trans_row_conflict_count ステータス変数, 3706
Ndb_conflict_trans_row_reject_count ステータス変数, 3707
ndb_cpcd, 3724
ndb_data_node_neighbour システム変数, 3669
ndb_dbg_check_shares システム変数, 3669
ndb_default_column_format システム変数, 3670
ndb_deferred_constraints システム変数, 3670
ndb_delete_all, 3724, 3753
 トランザクションオプション, 3754
ndb_desc, 3724, 3754
 auto-inc オプション, 3759
 blob-info オプション, 3759
 extra-node-info オプション, 3760
 extra-partition-info オプション, 3760
 retries オプション, 3760
 コンテキストオプション, 3759
 データベースオプション, 3759
 テーブルオプション, 3760
 未修飾オプション, 3760
ndb_distribution システム変数, 3670
ndb_drop_index, 3724, 3760
ndb_drop_table, 3724, 3761
Ndb_epoch_delete_delete_count ステータス変数, 3707
ndb_error_reporter, 3724, 3761
 connection-timeout オプション, 3762
 dry-scp オプション, 3762
 fs オプション, 3762
 skip-nodegroup オプション, 3762
 オプション, 3762
ndb_eventbuffer_free_percent システム変数, 3671
ndb_eventbuffer_max_alloc システム変数, 3671
Ndb_execute_count ステータス変数, 3707
ndb_extra_logging システム変数, 3671
ndb_force_send システム変数, 3672
ndb_fully_replicated システム変数, 3672
ndb_import, 3724, 3763
 abort-on-error オプション, 3766
 ai-increment オプション, 3766
 ai-off オプション, 3767
 ai-prefetch-sz オプション, 3767
 connections オプション, 3767
 continue オプション, 3767
 db-workers オプション, 3767
 errins-delay オプション, 3768
 errins-type オプション, 3767
 fields-enclosed-by オプション, 3768
 fields-escaped-by オプション, 3768
 fields-optionally-enclosed-by オプション, 3768
 fields-terminated-by オプション, 3768
 idlesleep オプション, 3769
 idlespin オプション, 3769
 ignore-lines オプション, 3769

input-type オプション, 3769
input-workers オプション, 3770
keep-state オプション, 3770
lines-terminated-by オプション, 3770
max-rows オプション, 3770
no-asynch オプション, 3771
no-hint オプション, 3771
opbatch オプション, 3771
opbytes オプション, 3771
output-type オプション, 3771
output-workers オプション, 3772
pagecnt オプション, 3772
pagesize オプション, 3772
polltimeout オプション, 3772
rowbatch オプション, 3773
rowbytes オプション, 3773
state-dir オプション, 3774
stats オプション, 3773
tempdelay オプション, 3774
temperrors オプション, 3774
verbose オプション, 3774
拒否オプション, 3773
再開オプション, 3773
モニターオプション, 3771
ログレベルオプション, 3770
ndb_index_stat, 3724, 3774
-d オプション, 3777
loops オプション, 3779
sys-check オプション, 3778
sys-create オプション, 3778
sys-create-if-not-exist オプション, 3778
sys-create-if-not-valid オプション, 3778
sys-drop オプション, 3778
sys-skip-events オプション, 3779
sys-skip-tables オプション, 3779
verbose オプション, 3779
クエリーオプション, 3777
更新オプション, 3777
削除オプション, 3777
出力の解釈, 3775
ダンプオプション, 3777
データベースオプション, 3777
例, 3775
ndb_index_stat_enable システム変数, 3672
ndb_index_stat_option システム変数, 3672
ndb_join_pushdown システム変数, 3674
Ndb_last_commit_epoch_server ステータス変数, 3707
Ndb_last_commit_epoch_session ステータス変数, 3707
ndb_log_apply_status システム変数, 3675
ndb_log_apply_status 変数 (NDB Cluster レプリケーション), 4006
ndb_log_bin システム変数, 3676
ndb_log_binlog_index システム変数, 3676
ndb_log_empty_epochs システム変数, 3676
ndb_log_empty_update システム変数, 3676
ndb_log_exclusive_reads (システム変数), 4027
ndb_log_exclusive_reads システム変数, 3677
ndb_log_orig システム変数, 3677
ndb_log_transaction_id システム変数, 3677
Ndb_metadata_blacklist_size ステータス変数 (OBSOLETE), 3707
ndb_metadata_check システム変数, 3678

ndb_metadata_check_interval システム変数, 3678
Ndb_metadata_detected_count ステータス変数, 3707
Ndb_metadata_excluded_count ステータス変数, 3707
ndb_metadata_sync システム変数, 3678
Ndb_metadata_synced_count ステータス変数, 3708
ndb_mgm (NDB Cluster 管理ノードクライアント), 3511
ndb_mgm, 3724, 3741 (参照 mgm)
 -e オプション, 3742
 connect-retries オプション, 3742
 MySQL Cluster Manager での使用, 3839
 実行オプション, 3742
ndb_mgmd (NDB Cluster プロセス), 3734
ndb_mgmd (NDB Cluster)
 定義済, 3450
 (参照 [管理ノード \(NDB Cluster\)](#))
ndb_mgmd, 3724 (参照 mgmd)
 -d オプション, 3737
 -f オプション, 3736
 -P オプション, 3740
 -v オプション, 3740
 bind-address オプション, 3735
 cluster-config-suffix オプション, 3735
 config-cache オプション, 3735
 config-file オプション, 3736
 configdir オプション, 3736
 log-name オプション, 3738
 mycnf オプション, 3738
 no-nodeid-checks オプション, 3738
 nodaemon オプション, 3738
 nowait-nodes オプション, 3738
 print-full-config オプション, 3740
 reload オプション, 3740
 remove オプション, 3740
 verbose オプション, 3740
 インストールオプション, 3737
 初期オプション, 3737
 対話型オプション, 3737
 デーモンオプション, 3737
ndb_move_data, 3724, 3779
 abort-on-error オプション, 3780
 character-sets-dir オプション, 3780
 drop-source オプション, 3781
 error-insert オプション, 3781
 exclude-missing-columns オプション, 3781
 lossy-conversions オプション, 3781
 promote-attributes オプション, 3781
 staging-tries オプション, 3781
 verbose オプション, 3781
 データベースオプション, 3780
Ndb_number_of_data_nodes ステータス変数, 3708
ndb_optimized_node_selection システム変数, 3679
ndb_perror, 3782
 ndb オプション, 3783
 verbose オプション, 3783
 サイレントオプション, 3783
 バージョンオプション, 3783
 ヘルプオプション, 3783
ndb_print_backup_file, 3724, 3783
ndb_print_file, 3724, 3784
ndb_print_frag_file, 3724, 3784

ndb_print_schema_file, 3724, 3785
ndb_print_sys_file, 3724, 3785
Ndb_pruned_scan_count ステータス変数, 3708
Ndb_pushed_queries_defined ステータス変数, 3708
Ndb_pushed_queries_dropped ステータス変数, 3708
Ndb_pushed_queries_executed ステータス変数, 3708
Ndb_pushed_reads ステータス変数, 3708
ndb_read_backup
 および NDB_TABLE, 2266
ndb_read_backup システム変数, 3680
ndb_rcv_thread_activation_threshold システム変数, 3680
ndb_rcv_thread_cpu_mask システム変数, 3680
ndb_redo_log_reader, 3786
 lap オプション, 3787
 twiddle オプション, 3788
 ダンプオプション, 3787
ndb_replication システムテーブル, 4018
Ndb_replica_max_replicated_epoch ステータス変数, 3708
ndb_report_thresh_binlog_epoch_slip システム変数, 3681
ndb_report_thresh_binlog_mem_usage システム変数, 3681
ndb_restore, 3788
 allow-pk-changes オプション, 3792
 append オプション, 3793
 backup-password オプション, 3794
 backup-path オプション, 3793
 backupid オプション, 3794
 decrypt オプション, 3795
 disable-indexes オプション, 3795
 dont-ignore-systab-0 オプション, 3795
 exclude-databases オプション, 3795
 exclude-intermediate-sql-tables オプション, 3796
 exclude-missing-columns オプション, 3796
 exclude-missing-tables オプション, 3796
 exclude-tables オプション, 3796
 fields-enclosed-by オプション, 3797
 fields-optionally-enclosed-by オプション, 3797
 fields-terminated-by オプション, 3797
 hex オプション, 3798
 ignore-extended-pk-updates オプション, 3798
 include-databases オプション, 3798
 include-stored-grants オプション, 3798
 include-tables オプション, 3798
 lines-terminated-by オプション, 3799
 lossy-conversions オプション, 3800
 ndb-nodgroup-map オプション, 3800
 no-binlog オプション, 3800
 no-restore-disk-objects オプション, 3800
 no-upgrade オプション, 3800
 nodeid オプション, 3800
 num-slices オプション, 3801
 preserve-trailing-spaces オプション, 3803
 print-data オプション, 3803
 print-log オプション, 3803
 print-meta オプション, 3803
 print-sql-log オプション, 3804
 progress-frequency オプション, 3804
 promote-attributes オプション, 3804
 rebuild-indexes オプション, 3805
 remap-column オプション, 3805
 restore-data オプション, 3806

restore-epoch オプション, 3806
restore-meta オプション, 3806
restore-privilege-tables オプション, 3807
rewrite-database オプション, 3807
skip-broken-objects オプション, 3808
skip-table-check オプション, 3808
skip-unknown-objects オプション, 3808
slice-id オプション, 3808
verbose オプション, 3809
一般的なオプションおよび必須のオプション, 3792
印刷オプション, 3803
エラー, 3809
循環レプリケーション, 4014
接続オプション, 3795
タブオプション, 3808
並列処理オプション, 3802
ndb_row_checksum システム変数, 3681
Ndb_scan_count ステータス変数, 3708
ndb_schema_dist_lock_wait_timeout システム変数, 3682
ndb_schema_dist_timeout システム変数, 3682
ndb_schema_dist_upgrade_allowed システム変数, 3682
ndb_select_all, 3724, 3814
 delimiter オプション, 3816
 descending オプション, 3815
 gci オプション, 3816
 gci64 オプション, 3816
 lock オプション, 3815
 nodata オプション, 3816
 rowid オプション, 3816
 tupscan オプション, 3816
 useHexFormat オプション, 3816
 オーダーオプション, 3815
 ディスクオプション, 3816
 データベースオプション, 3815
 並列処理オプション, 3815
 ヘッダーオプション, 3816
ndb_select_count, 3724, 3817
ndb_setup.py, 3724, 3818
 browser-start-page オプション, 3819
 ca-certs-file オプション, 3819
 cert-file オプション, 3819
 key-file オプション, 3820
 no-browser オプション, 3820
 port オプション, 3820
 server-log-file オプション, 3820
 server-name オプション, 3821
 use-http オプション, 3821, 3821
 デバッグレベルオプション, 3820
 ヘルプオプション, 3820
ndb_show_foreign_key_mock_tables システム変数, 3683
ndb_show_tables, 3724, 3821
 loops オプション, 3822
 show-temp-status オプション, 3822
 解析可能オプション, 3822
 タイプオプション, 3822
 データベースオプション, 3822
 未修飾オプション, 3822
ndb_size.pl, 3724, 3822
ndb_size.pl スクリプト, 1827
ndb_slave_conflict_role システム変数, 3683

Ndb_slave_max_replicated_epoch ステータス変数, 3709
NDB_STORED_USER, 1056, 3899
Ndb_system_name ステータス変数, 3709
NDB_TABLE, 2230, 2266, 3580
ndb_table_no_logging システム変数, 3683
ndb_table_temporary システム変数, 3684
ndb_top, 3724, 3825
 measured-load オプション, 3827
 node-id オプション, 3827
 os-load オプション, 3828
 passwd オプション, 3828
 port オプション, 3828
 sleep-time オプション, 3828
 socket オプション, 3828
 色オプション, 3827
 グラフオプション, 3827
 ソートオプション, 3829
 テキストオプション, 3829
 パスワードオプション, 3828
 ヘルプオプション, 3827
 ホストオプション, 3827
 ユーザーオプション, 3829
ndb_transid_mysql_connection_map
 INFORMATION_SCHEMA テーブル, 4156
Ndb_trans_hint_count_session ステータス変数, 3709
ndb_use_copying_alter_table システム変数, 3684
ndb_use_exact_count システム変数, 3685
ndb_use_transactions システム変数, 3685
NDB_UTILS_LINK_DYNAMIC
 CMake, 217
ndb_version システム変数, 3685
ndb_version_string システム変数, 3685
ndb_waiter, 3724, 3830
 no-contact オプション, 3831
 not-started オプション, 3831
 nowait-nodes オプション, 3831
 timeout オプション, 3831
 wait-nodes オプション, 3831
 シングルユーザーオプション, 3831
Nested Loop 結合アルゴリズム, 1449, 1453
net-buffer-length オプション
 mysql, 374
 mysqldump, 435
 mysqlpump, 458
 mysql_upgrade, 359
network-namespace オプション
 mysql, 374
network-timeout オプション
 mysqldump, 435
net_buffer_length システム変数, 742
net_read_timeout システム変数, 742
net_retry_count システム変数, 742
net_write_timeout システム変数, 743
newline (\n), 1628, 2070, 2312
NFS
 InnoDB, 2722
ngram_token_size システム変数, 743
nice オプション
 mysqld_safe, 338
NO PAD 照合, 1714, 1726, 1785

no-asynch オプション
 ndb_import, 3771

no-auto-rehash オプション
 mysql, 374

no-autocommit オプション
 mysqldump, 437

no-beep オプション
 mysql, 374
 mysqladmin, 401

no-binlog オプション
 ndb_restore, 3800

no-browser オプション
 ndb_setup.py, 3820

no-check オプション
 ibd2sdi, 486
 innochecksum, 488

no-contact オプション
 ndb_waiter, 3831

no-create-db オプション
 mysqldump, 426
 mysqlpump, 458

no-create-info オプション
 mysqldump, 426
 mysqlpump, 458

no-data オプション
 mysqldump, 434

no-dd-upgrade オプション
 mysqld, 656

no-defaults オプション, 308
 myisamchk, 496
 mysql, 374
 mysqladmin, 401
 mysqlbinlog, 530
 mysqlcheck, 411
 mysqld, 656
 mysqldump, 425
 mysqld_multi, 343
 mysqld_safe, 338
 mysqlimport, 444
 mysqlpump, 459
 mysqlshow, 470
 mysqlslap, 480
 mysql_secure_installation, 349
 mysql_upgrade, 359
 my_print_defaults, 548
 NDB クライアントプログラム, 3837

no-drop オプション
 mysqlslap, 480

no-hint オプション
 ndb_import, 3771

no-log オプション
 mysqld_multi, 344

no-monitor オプション
 mysqld, 657

no-nodeid-checks オプション
 ndb_mgmd, 3738

no-restore-disk-objects オプション
 ndb_restore, 3800

no-set-names オプション
 mysqldump, 428

- no-symlinks オプション
 - mysamchk, 499
- no-tablespaces オプション
 - mysqldump, 426
- no-upgrade オプション
 - ndb_restore, 3800
- nodaemon オプション
 - ndb_mgmd, 3738
- nodata オプション
 - ndb_select_all, 3816
- node-id オプション
 - ndb_top, 3827
- Node.js, 4525
- NodeGroup, 3577
- NodeGroupTransporters, 3593
- Nodeld, 3569, 3576, 3649
- nodeid オプション
 - ndb_config, 3749
 - ndb_restore, 3800
- Nodeld1, 3710, 3717
- Nodeld2, 3711, 3717
- NodeldServer, 3717
- NODELOG DEBUG コマンド (NDB Cluster), 3843
- NODERESTART イベント (NDB Cluster), 3863
- nodes
 - ndbinfo テーブル, 3951
- nodes オプション
 - ndb_config, 3749
- NOLOGGING (NDB_TABLE)
 - NDB Cluster, 2266
- NOLOGGING, 2266
- NoOfFragmentLogFiles, 3596
- NoOfFragmentLogParts, 3631
- NoOfReplicas, 3578
- nopager コマンド
 - mysql, 381
- normalize_statement() MySQL Enterprise Firewall UDF, 1370
- NoSQL, 5346
- NoSQL, 3371
- NoSQL データベース
 - としての MySQL, 3373
- nostart オプション
 - ndbd, 3729
 - ndbmtd, 3729
- NOT BETWEEN, 1863
- NOT EXISTS
 - サブクエリーを使用した, 2349
- NOT IN, 1864
- NOT LIKE, 1916
- NOT
 - logical, 1866
- NOT NULL 制約, 5346
- NOT REGEXP, 1918
- not-started オプション
 - ndb_waiter, 3831
- notee コマンド
 - mysql, 382
- NOTIFY_SOCKET 環境変数, 177, 550
- Not_flushed_delayed_rows ステータス変数, 847
- NOW(), 1892

NOWAIT (START BACKUP コマンド), 3885
NOWAIT, 2330
nowait-nodes オプション
 ndbd, 3729
 ndbmt, 3729
 ndb_mgmd, 3738
 ndb_waiter, 3831
nowarning コマンド
 mysql, 382
NO_AUTO_VALUE_ON_ZERO SQL モード, 857
NO_BACKSLASH_ESCAPES SQL モード, 857
NO_DIR_IN_CREATE SQL モード, 857
NO_ENGINE_SUBSTITUTION SQL モード, 857
NO_FIELD_OPTIONS
 削除された機能, 39
NO_GROUP_INDEX, 1572
NO_ICP, 1572
NO_INDEX, 1572
NO_INDEX_MERGE, 1572
NO_JOIN_INDEX, 1572
NO_KEY_OPTIONS
 削除された機能, 39
NO_MRR, 1572
NO_ORDER_INDEX, 1573
NO_RANGE_OPTIMIZATION, 1572
NO_SKIP_SCAN, 1573
NO_TABLE_OPTIONS
 削除された機能, 39
NO_UNSIGNED_SUBTRACTION SQL モード, 857
NO_ZERO_DATE SQL モード, 858
NO_ZERO_IN_DATE SQL モード, 858
NTH_VALUE(), 2114
NTILE(), 2115
NUL, 1628, 2312
NULL, 5347
NULL, 278, 4608
 null のテスト, 1862, 1863, 1865, 1865, 1871
 ORDER BY, 1469
NULL 拒否条件, 1457
NULL で補完された行, 1453, 1457
NULL としての\N
 削除された機能, 40
NULL 値, 278, 1635
 ORDER BY, 1635
 インデックス, 2223
 および AUTO_INCREMENT カラム, 4609
 および TIMESTAMP カラム, 4609
 と空の値, 4608
null リテラル
 JSON, 1811
NULLIF(), 1871
num-slices オプション
 ndb_restore, 3801
Nump, 3625
number-char-cols オプション
 mysqlslap, 480
number-int-cols オプション
 mysqlslap, 480
number-of-queries オプション
 mysqlslap, 480

NumCPUs, 3632
NUMERIC データ型, 1764
NumGeometries()
削除された機能, 40
NumInteriorRings()
削除された機能, 40
NumPoints()
削除された機能, 40
NVARCHAR データ型, 1783

O

object
JSON, 1811
objects_summary_global_by_type テーブル
performance_schema, 4384
OCT(), 1908
OCTET_LENGTH(), 1908
ODBC, 5347
ODBC 互換性, 778, 1856, 1865, 2224
ODBC との互換性, 1765
ODBC の互換性, 2336
ODBC_INCLUDES= option
CMake, 202
ODBC_LIB_DIR オプション
CMake, 202
ODirect, 3603
ODirectSyncFlag, 3604
OFF
プラグインアクティベーションオプション, 949
offline_mode システム変数, 744
offset オプション
mysqlbinlog, 530
OGC (参照 [Open Geospatial Consortium](#))
OLAP, 2098
old-style-user-limits オプション
mysqld, 657
old_alter_table システム変数, 744
old_passwords
削除された機能, 38
OLTP, 5347
ON DUPLICATE KEY
INSERT 修飾子, 2303
ON DUPLICATE KEY UPDATE, 2299
新機能, 30
ON
プラグインアクティベーションオプション, 949
ON と USING
結合, 2338
Ongoing_anonymous_gtid_violating_transaction_count ステータス変数, 847
Ongoing_anonymous_transaction_count ステータス変数, 847
Ongoing_automatic_gtid_violating_transaction_count ステータス変数, 847
only-print オプション
mysqlslap, 480
ONLY_FULL_GROUP_BY
SQL モード, 2104
ONLY_FULL_GROUP_BY SQL モード, 859
opbatch オプション
ndb_import, 3771
opbytes オプション

- ndb_import, 3771
- Open Geospatial Consortium, 1793
- OPEN, 2451
- open-files-limit オプション
 - mysqlbinlog, 530
 - mysqld_safe, 338
- Opened_files ステータス変数, 847
- Opened_tables ステータス変数, 848
- Opened_table_definitions ステータス変数, 848
- OpenGIS, 1793
- Opening master dump table
 - スレッドの状態, 1624
- Opening mysql.ndb_apply_status
 - スレッドの状態, 1625
- OpenLDAP 構成
 - ldap.conf ファイル, 1181
- OpenSSL FIPS オブジェクトモジュール, 1401
- OpenSSL, 191, 1129
 - FIPS モード, 1401
- Open_files ステータス変数, 847
- open_files_limit システム変数, 745
- Open_streams ステータス変数, 847
- Open_tables ステータス変数, 847
- Open_table_definitions ステータス変数, 847
- operations_per_fragment
 - ndbinfo テーブル, 3953
- opt オプション
 - mysqldump, 435
- OPTIMIZE TABLE
 - およびパーティショニング, 4071
- OPTIMIZE TABLE ステートメント, 2532
- optimizer_prune_level システム変数, 746
- optimizer_search_depth システム変数, 746
- optimizer_switch システム変数, 746, 1556
 - use_invisible_indexes フラグ, 1510
- OPTIMIZER_TRACE
 - INFORMATION_SCHEMA テーブル, 4157
- OPTIMIZER_TRACE オプション
 - CMake, 208
- optimizer_trace システム変数, 750
- optimizer_trace_features システム変数, 751
- optimizer_trace_limit システム変数, 751
- optimizer_trace_max_mem_size システム変数, 751
- optimizer_trace_offset システム変数, 751
- optimizing
 - スレッドの状態, 1618
- options
 - mysqld, 557
 - レプリケーション, 3212
- OR, 291, 1441
 - ビット単位, 1983
 - 論理, 1867
- OR インデックスマージの最適化, 1441
- Oracle Key Vault, 2805
 - keyring_okv キーリングプラグイン, 1238
- ORACLE
 - 削除された機能, 39
- Oracle の互換性, 68, 2097, 2173, 2611
- ORD(), 1908
- ORDER BY, 275, 2178, 2328

NULL, 1469
NULL 値, 1635
WITH ROLLUP, 2328
ウィンドウ関数, 2119
カッコで囲まれたクエリー式, 2342
最大ソート長, 2328
ORDER BY 最適化, 1467
order-by-primary オプション
mysqldump, 437
ORDER_INDEX, 1573
original_commit_timestamp, 3209
original_commit_timestamp システム変数, 3135
original_server_version システム変数, 3076
os-load オプション
ndb_top, 3828
OUT パラメータ
条件の処理, 2476
out-dir オプション
comp_err, 347
out-file オプション
comp_err, 347
OUTFILE, 2332
output-type オプション
ndb_import, 3771
output-workers オプション
ndb_import, 3772
OVER 句
ウィンドウ関数, 2117
Overlaps()
削除された機能, 40
OverloadLimit, 3711, 3718

P

PAD SPACE 照合順序, 1714, 1726, 1785
PAD_CHAR_TO_FULL_LENGTH
非推奨となった機能, 36
PAD_CHAR_TO_FULL_LENGTH SQL モード, 859
page size, 5347
page-type-dump オプション
innochecksum, 489
page-type-summary オプション
innochecksum, 489
pagecnt オプション
ndb_import, 3772
pager オプション
mysql, 375
pager コマンド
mysql, 382
pagesize オプション
ndb_import, 3772
PAM
ブラガブル認証, 1162
parallel-recover オプション
myisamchk, 499
parallel-schemas オプション
mysqlpump, 459
parameters
server, 556
PARAMETERS

- INFORMATION_SCHEMA テーブル, 4158
- parameter_type_elements テーブル
 - データディクショナリテーブル, 896
- parser_max_mem_size システム変数, 752
- partial_revokes システム変数, 752
- PARTITION BY LIST COLUMNS, 4040
- PARTITION BY
 - ウィンドウ関数, 2119
- PARTITION BY RANGE COLUMNS, 4040
- PARTITION, 4029
- partitioning, 4029
 - NDB Cluster でのサポート, 3483
 - support, 4029
 - インデックス接頭辞, 4085
 - ウィンドウ関数, 2119
 - キー, 4032
 - 最適化, 4073
 - テーブル, 4029
 - パーティショニング式, 4032
 - 有効化, 4029
 - レプリケーション, 3225
- PARTITIONS
 - INFORMATION_SCHEMA テーブル, 4159
 - PartitionsPerNode, 3632
- PARTITION_BALANCE (NDB_TABLE)
 - NDB Cluster, 2267
- PARTITION_BALANCE, 2266, 3580
- passwd オプション
 - ndb_top, 3828
- password
 - root ユーザー, 230
 - 期限切れのリセット, 1100
- PASSWORD()
 - 削除された機能, 38
- passwords
 - expiration, 1108
 - security, 1043
 - 設定, 1097, 2517
 - リセット, 1108
- password_history システム変数, 753
- password_history テーブル
 - システムテーブル, 898, 1063
- password_require_current システム変数, 753
- password_reuse_interval システム変数, 754
- PATH 環境変数, 131, 136, 228, 300, 550
- PERCENT_RANK(), 2115
- performance-schema-consumer-events-stages-current オプション
 - mysqld, 4418
- performance-schema-consumer-events-stages-history オプション
 - mysqld, 4418
- performance-schema-consumer-events-stages-history-long オプション
 - mysqld, 4418
- performance-schema-consumer-events-statements-current オプション
 - mysqld, 4418
- performance-schema-consumer-events-statements-history オプション
 - mysqld, 4419
- performance-schema-consumer-events-statements-history-long オプション
 - mysqld, 4419
- performance-schema-consumer-events-transactions-current オプション
 - mysqld, 4419

performance-schema-consumer-events-transactions-history オプション
mysql, 4419

performance-schema-consumer-events-transactions-history-long オプション
mysql, 4419

performance-schema-consumer-events-waits-current オプション
mysql, 4419

performance-schema-consumer-events-waits-history オプション
mysql, 4419

performance-schema-consumer-events-waits-history-long オプション
mysql, 4419

performance-schema-consumer-global-instrumentation オプション
mysql, 4419

performance-schema-consumer-statements-digest オプション
mysql, 4419

performance-schema-consumer-thread-instrumentation オプション
mysql, 4419

performance-schema-consumer-xxx オプション
mysql, 4418

performance-schema-instrument オプション
mysql, 4418

performance_schema

- accounts テーブル, 4323
- clone_progress テーブル, 4370
- clone_status テーブル, 4369
- cond_instances テーブル, 4290
- data_locks テーブル, 4351, 4442
- data_lock_waits テーブル, 4353
- error_log テーブル, 4397
- events_errors_summary_by_account_by_error テーブル, 4394
- events_errors_summary_by_host_by_error テーブル, 4394
- events_errors_summary_by_thread_by_error テーブル, 4394
- events_errors_summary_by_user_by_error テーブル, 4394
- events_errors_summary_global_by_error テーブル, 4394
- events_stages_current テーブル, 4303
- events_stages_history テーブル, 4304
- events_stages_history_long テーブル, 4304
- events_stages_summary_by_account_by_event_name テーブル, 4375
- events_stages_summary_by_host_by_event_name テーブル, 4375
- events_stages_summary_by_thread_by_event_name テーブル, 4375
- events_stages_summary_by_user_by_event_name テーブル, 4375
- events_stages_summary_global_by_event_name テーブル, 4375
- events_statements_current テーブル, 4308
- events_statements_histogram_by_digest テーブル, 4380
- events_statements_histogram_global テーブル, 4380
- events_statements_history テーブル, 4312
- events_statements_history_long テーブル, 4312
- events_statements_summary_by_account_by_event_name テーブル, 4377
- events_statements_summary_by_digest テーブル, 4377
- events_statements_summary_by_host_by_event_name テーブル, 4377
- events_statements_summary_by_program テーブル, 4377
- events_statements_summary_by_thread_by_event_name テーブル, 4377
- events_statements_summary_by_user_by_event_name テーブル, 4377
- events_statements_summary_global_by_event_name テーブル, 4377
- events_transactions_current テーブル, 4319
- events_transactions_history テーブル, 4321
- events_transactions_history_long テーブル, 4321
- events_transactions_summary_by_account_by_event_name テーブル, 4382
- events_transactions_summary_by_host_by_event_name テーブル, 4382
- events_transactions_summary_by_thread_by_event_name テーブル, 4382
- events_transactions_summary_by_user_by_event_name テーブル, 4382

events_transactions_summary_global_by_event_name テーブル, 4382
events_waits_current テーブル, 4296
events_waits_history テーブル, 4299
events_waits_history_long テーブル, 4299
events_waits_summary_by_account_by_event_name テーブル, 4373
events_waits_summary_by_host_by_event_name テーブル, 4374
events_waits_summary_by_instance テーブル, 4374
events_waits_summary_by_thread_by_event_name テーブル, 4374
events_waits_summary_by_user_by_event_name テーブル, 4374
events_waits_summary_global_by_event_name テーブル, 4374
file_instances テーブル, 4290
file_summary_by_event_name テーブル, 4385
file_summary_by_instance テーブル, 4385
firewall_groups テーブル, 4400
firewall_group_allowlist テーブル, 4400
firewall_membership テーブル, 4401
hosts テーブル, 4324
host_cache テーブル, 868, 4401
keyring_keys テーブル, 1232, 1302, 1329, 4404
log_status テーブル, 4404
memory_summary_by_account_by_event_name テーブル, 4390
memory_summary_by_host_by_event_name テーブル, 4390
memory_summary_by_thread_by_event_name テーブル, 4390
memory_summary_by_user_by_event_name テーブル, 4390
memory_summary_global_by_event_name テーブル, 4390
metadata_locks テーブル, 4355
mutex_instances テーブル, 4290
objects_summary_global_by_type テーブル, 4384
performance_timers テーブル, 4405
prepared_statements_instances テーブル, 4377
processlist テーブル, 4406
replication_applier_configuration, 4338
replication_applier_status, 4339
replication_applier_status_by_coordinator, 4340
replication_applier_status_by_worker, 4341
replication_asynchronous_connection_failover, 4337
replication_connection_configuration, 4332
replication_connection_status, 4335
rwlock_instances テーブル, 4292
session_account_connect_attrs テーブル, 4328
session_connect_attrs テーブル, 4328
setup_actors テーブル, 4282
setup_consumers テーブル, 4283
setup_instruments テーブル, 4284
setup_objects テーブル, 4286
setup_threads テーブル, 4288
socket_instances テーブル, 4293
socket_summary_by_event_name テーブル, 4389
socket_summary_by_instance テーブル, 4389
table_handles テーブル, 4357
table_io_waits_summary_by_index_usage テーブル, 4387
table_io_waits_summary_by_table テーブル, 4386
table_lock_waits_summary_by_table テーブル, 4387
tls_channel_status テーブル, 4413
tp_thread_group_state テーブル, 4364
tp_thread_group_stats テーブル, 4366
tp_thread_state テーブル, 4368
users テーブル, 4324
user_defined_functions テーブル, 1009, 4414
user_variables_by_thread テーブル, 4329

親イベント, 4442
スレッドテーブル, 4408
performance_schema システム変数, 4420
PERFORMANCE_SCHEMA ストレージエンジン, 4238
performance_schema データベース, 4238
TRUNCATE TABLE, 4277, 4443
制約, 4443
performance_schema.global_status テーブル
NDB Cluster, 3982
performance_schema.global_variables テーブル
NDB Cluster, 3978
Performance_schema_accounts_lost ステータス変数, 4436
performance_schema_accounts_size システム変数, 4420
Performance_schema_cond_classes_lost ステータス変数, 4436
Performance_schema_cond_instances_lost ステータス変数, 4436
performance_schema_digests_size システム変数, 4421
Performance_schema_digest_lost ステータス変数, 4436
performance_schema_error_size システム変数, 4421
performance_schema_events_stages_history_long_size システム変数, 4422
performance_schema_events_stages_history_size システム変数, 4422
performance_schema_events_statements_history_long_size システム変数, 4422
performance_schema_events_statements_history_size システム変数, 4422
performance_schema_events_transactions_history_long_size システム変数, 4423
performance_schema_events_transactions_history_size システム変数, 4423
performance_schema_events_waits_history_long_size システム変数, 4423
performance_schema_events_waits_history_size システム変数, 4423
Performance_schema_file_classes_lost ステータス変数, 4436
Performance_schema_file_handles_lost ステータス変数, 4436
Performance_schema_file_instances_lost ステータス変数, 4436
Performance_schema_hosts_lost ステータス変数, 4436
performance_schema_hosts_size システム変数, 4424
Performance_schema_index_stat_lost ステータス変数, 4436
Performance_schema_locker_lost ステータス変数, 4436
performance_schema_max_cond_classes システム変数, 4424
performance_schema_max_cond_instances システム変数, 4424
performance_schema_max_digest_length システム変数, 4425
performance_schema_max_digest_sample_age システム変数, 4425
performance_schema_max_file_classes システム変数, 4426
performance_schema_max_file_handles システム変数, 4426
performance_schema_max_file_instances システム変数, 4426
performance_schema_max_index_stat システム変数, 4427
performance_schema_max_memory_classes システム変数, 4427
performance_schema_max_metadata_locks システム変数, 4427
performance_schema_max_mutex_classes システム変数, 4427
performance_schema_max_mutex_instances システム変数, 4428
performance_schema_max_prepared_statements_instances システム変数, 4428
performance_schema_max_program_instances システム変数, 4429
performance_schema_max_rwlock_classes システム変数, 4428
performance_schema_max_rwlock_instances システム変数, 4429
performance_schema_max_socket_classes システム変数, 4429
performance_schema_max_socket_instances システム変数, 4430
performance_schema_max_sql_text_length システム変数, 4430
performance_schema_max_stage_classes システム変数, 4430
performance_schema_max_statement_classes システム変数, 4431
performance_schema_max_statement_stack システム変数, 4431
performance_schema_max_table_handles システム変数, 4431
performance_schema_max_table_instances システム変数, 4432
performance_schema_max_table_lock_stat システム変数, 4432
performance_schema_max_thread_classes システム変数, 4432
performance_schema_max_thread_instances システム変数, 4433

Performance_schema_memory_classes_lost ステータス変数, 4436
Performance_schema_metadata_lock_lost ステータス変数, 4436
Performance_schema_mutex_classes_lost ステータス変数, 4436
Performance_schema_mutex_instances_lost ステータス変数, 4437
Performance_schema_nested_statement_lost ステータス変数, 4437
Performance_schema_prepared_statements_lost ステータス変数, 4437
Performance_schema_program_lost ステータス変数, 4437
Performance_schema_rwlock_classes_lost ステータス変数, 4437
Performance_schema_rwlock_instances_lost ステータス変数, 4437
Performance_schema_session_connect_attrs_longest_seen ステータス変数, 4437
Performance_schema_session_connect_attrs_lost ステータス変数, 4437
performance_schema_session_connect_attrs_size システム変数, 4433
performance_schema_setup_actors_size システム変数, 4434
performance_schema_setup_objects_size システム変数, 4434
performance_schema_show_processlist システム変数, 4434
Performance_schema_socket_classes_lost ステータス変数, 4437
Performance_schema_socket_instances_lost ステータス変数, 4437
Performance_schema_stage_classes_lost ステータス変数, 4437
Performance_schema_statement_classes_lost ステータス変数, 4437
Performance_schema_table_handles_lost ステータス変数, 4438
Performance_schema_table_instances_lost ステータス変数, 4438
Performance_schema_table_lock_stat_lost ステータス変数, 4438
Performance_schema_thread_classes_lost ステータス変数, 4438
Performance_schema_thread_instances_lost ステータス変数, 4438
Performance_schema_users_lost ステータス変数, 4438
performance_schema_users_size システム変数, 4435
performance_timers
 削除された機能, 42
performance_timers テーブル
 performance_schema, 4405
PERIOD_ADD(), 1892
PERIOD_DIFF(), 1893
Perl, 5348
Perl API, 5348
Perl API, 4525
Perl DBI/DBD
 インストールに関する問題, 263
Perl
 Windows にインストール, 263
 インストール, 262
perror, 299, 549
 ndb オプション, 549
 verbose オプション, 549
 サイレントオプション, 549
 削除された機能, 44
 バージョンオプション, 549
 ヘルプオプション, 549
PERSIST
 SET ステートメント, 826, 2542
persisted_globals_load システム変数, 754, 826
PERSIST_ONLY
 SET ステートメント, 826, 2542
persist_only_admin_x509_subject システム変数, 754
PERSIST_RO_VARIABLES_ADMIN 権限, 1057
pgman_time_track_stats
 ndbinfo テーブル, 3956
PHP, 5348
PHP API, 5348
PI(), 1879
pid-file オプション

- mysql.server, 342
- mysqld_safe, 338
- pid_file システム変数, 755
- Ping
 - スレッドのコマンド, 1615
- pipe オプション, 313
 - mysql, 375, 411
 - mysqladmin, 401
 - mysqldump, 423
 - mysqlimport, 445
 - mysqlshow, 470
 - mysqlslap, 480
 - mysql_upgrade, 360
- PIPES_AS_CONCAT SQL モード, 860
- PITR, 5348
- PKG_CONFIG_PATH 環境変数, 550
- plugin
 - audit_log, 1279
- plugin オプション接頭辞
 - mysqld, 659
- plugin-dir オプション, 313
 - mysql, 375
 - mysqladmin, 401
 - mysqlbinlog, 531
 - mysqlcheck, 411
 - mysqldump, 423
 - mysqld_safe, 338
 - mysqlimport, 445
 - mysqlpump, 459
 - mysqlshow, 470
 - mysqlslap, 480
 - mysql_upgrade, 360
- plugin-load オプション
 - mysqld, 658
- plugin-load-add オプション
 - mysqld, 658
- plugindir オプション
 - mysql_config, 547
- PLUGINS
 - INFORMATION_SCHEMA テーブル, 4162
- plugin_dir システム変数, 755
- POINT データ型, 1794
- Point(), 2011
- point-in-time リカバリ
 - InnoDB, 2939
 - NDB Cluster レプリケーションの使用, 4012
- PointFromText()
 - 削除された機能, 40
- PointFromWKB()
 - 削除された機能, 40
- PointN()
 - 削除された機能, 40
- polltimeout オプション
 - ndb_import, 3772
- PolyFromText()
 - 削除された機能, 40
- PolyFromWKB()
 - 削除された機能, 40
- POLYGON データ型, 1794
- Polygon(), 2011

PolygonFromText()
削除された機能, 40

PolygonFromWKB()
削除された機能, 40

port, 5348

port オプション, 313

- mysql, 375
- mysqladmin, 401
- mysqlbinlog, 531
- mysqlcheck, 411
- mysqld, 659
- mysqldump, 423
- mysqld_safe, 338
- mysqlimport, 445
- mysqlpump, 459
- mysqlshow, 470
- mysqlslap, 481
- mysql_config, 547
- mysql_secure_installation, 349
- mysql_upgrade, 360
- ndb_setup.py, 3820
- ndb_top, 3828

port システム変数, 755

port-open-timeout オプション

- mysqld, 660

PortNumber, 3570

PortNumberStats, 3573

ports, 208, 208, 531

POSITION(), 1909

post-system オプション

- mysqlslap, 481

POSTGRES SQL

- 削除された機能, 39

PostgreSQL の互換性, 69

POW(), 1879

POWER(), 1879

pre-system オプション

- mysqlslap, 481

precedence

- コマンドオプション, 300

preload_buffer_size システム変数, 756

Prepare

- スレッドのコマンド, 1615

PREPARE, 2437, 2441

- XA トランザクション, 2391

prepared_statements_instances テーブル

performance_schema, 4377

Prepared_stmt_count ステータス変数, 848

preparing for alter table

- スレッドの状態, 1619

preparing

- スレッドの状態, 1618

PreSendChecksum, 3711, 3718

preserve-trailing-spaces オプション

- ndb_restore, 3803

pretty オプション

- ibd2sdi, 486

PRIMARY KEY, 2175, 2225

PrimaryMGNode, 3656

print コマンド

- mysql, 382
- print-data オプション
 - ndb_restore, 3803
- print-defaults オプション, 308
 - myisamchk, 496
 - mysql, 375
 - mysqladmin, 402
 - mysqlbinlog, 531
 - mysqlcheck, 411
 - mysqld, 660
 - mysqldump, 425
 - mysqlimport, 445
 - mysqlpump, 459
 - mysqlshow, 470
 - mysqlslap, 481
 - mysql_secure_installation, 349
 - mysql_upgrade, 360
 - NDB クライアントプログラム, 3837
- print-full-config オプション
 - ndb_mgmd, 3740
- print-log オプション
 - ndb_restore, 3803
- print-meta オプション
 - ndb_restore, 3803
- print-sql-log オプション
 - ndb_restore, 3804
- print-table-metadata オプション
 - mysqlbinlog, 531
- print_identified_with_as_hex システム変数, 756
- privileges
 - access, 1043
 - ALL, 1048
 - ALL PRIVILEGES, 1048
 - ALTER, 1048
 - ALTER ROUTINE, 1048
 - APPLICATION_PASSWORD_ADMIN, 1054
 - AUDIT_ADMIN, 1055
 - BACKUP_ADMIN, 1055
 - BINLOG_ADMIN, 1055
 - BINLOG_ENCRYPTION_ADMIN, 1055
 - CLONE_ADMIN, 1055
 - CONNECTION_ADMIN, 1055
 - CREATE, 1048
 - CREATE ROLE, 1048
 - CREATE ROUTINE, 1049
 - CREATE TABLESPACE, 1049
 - CREATE TEMPORARY TABLES, 1049
 - CREATE USER, 1049
 - CREATE VIEW, 1049
 - default, 230
 - DEFINER, 2566, 4116
 - DELETE, 1049
 - display, 2565
 - DROP, 1049
 - DROP ROLE, 1049
 - ENCRYPTION_KEY_ADMIN, 1055
 - EVENT, 1049
 - EXECUTE, 1049
 - FILE, 1049
 - FIREWALL_ADMIN, 1056

FIREWALL_USER, 1056
FLUSH_OPTIMIZER_COSTS, 1056
FLUSH_STATUS, 1056
FLUSH_TABLES, 1056
FLUSH_USER_RESOURCES, 1056
GRANT OPTION, 1050
GROUP_REPLICATION_ADMIN, 1056
INDEX, 1050
INNODB_REDO_LOG_ARCHIVE, 1056
INNODB_REDO_LOG_ENABLE, 1056
INSERT, 1050
INVOKER, 2566, 4116
LOCK TABLES, 1050
NDB_STORED_USER, 1056
PERSIST_RO_VARIABLES_ADMIN, 1057
PROCESS, 1050
PROXY, 1050
REFERENCES, 1050
RELOAD, 1051
REPLICATION CLIENT, 1051
REPLICATION SLAVE, 1051
REPLICATION_APPLIER, 1057
REPLICATION_SLAVE_ADMIN, 1057
RESOURCE_GROUP_ADMIN, 1057
RESOURCE_GROUP_USER, 1057
ROLE_ADMIN, 1057
SELECT, 1051
SERVICE_CONNECTION_ADMIN, 1057
SESSION_VARIABLES_ADMIN, 1057
SET_USER_ID, 1058
SHOW DATABASES, 1051
SHOW VIEW, 1051
SHOW_ROUTINE, 1058
SHUTDOWN, 1052
SQL SECURITY, 4116
SUPER, 1053
SYSTEM_USER, 1058, 1087
SYSTEM_VARIABLES_ADMIN, 1059
TABLE_ENCRYPTION_ADMIN, 1059
TEMPORARY テーブル, 1049, 2244, 2511
TRIGGER, 1054
UPDATE, 1054
USAGE, 1054
VERSION_TOKEN_ADMIN, 1059
XA_RECOVER_ADMIN, 1059
削除, 2501, 2501
ストアオブジェクト, 4116
静的と動的, 1060
チェック, 1079
追加, 1077
取消し, 1080, 2515
付与, 2502
レプリケーション, 3225
PROCEDURE ANALYSE()
削除された機能, 40
PROCESS 権限, 1050
Processing events from schema table
スレッドの状態, 1625
Processing events
スレッドの状態, 1625

Processlist
スレッドのコマンド, 1615
PROCESSLIST, 2574
 INFORMATION_SCHEMA テーブル, 4163
 INFORMATION_SCHEMA テーブルとの不整合の可能性, 2902
processlist
 監視, 4406
processlist テーブル
 performance_schema, 4406
processlist ビュー
 sys スキーマ, 4468
procs_priv テーブル
 システムテーブル, 898, 1063
PROFILING
 INFORMATION_SCHEMA テーブル, 4165
profiling_history_size システム変数, 756
progress-frequency オプション
 ndb_restore, 3804
promote-attributes オプション
 ndb_move_data, 3781
 ndb_restore, 3804
prompt オプション
 mysql, 375
PROMPT コマンド (NDB Cluster), 3843
prompt コマンド
 mysql, 382
protocol オプション, 313
 mysql, 375
 mysqladmin, 402
 mysqlbinlog, 531
 mysqlcheck, 411
 mysqldump, 423
 mysqlexport, 445
 mysqlpump, 459
 mysqlshow, 471
 mysqlslap, 481
 mysql_secure_installation, 349
 mysql_upgrade, 360
protocol_compression_algorithms システム変数, 757
protocol_version システム変数, 757
proxies_priv
 付与テーブル, 1117
proxies_priv テーブル
 システムテーブル, 230, 898, 1063
PROXY 権限, 1050
proxy_user システム変数, 758
pseudo_slave_mode システム変数, 758
pseudo_thread_id システム変数, 759
ps_check_lost_instrumentation ビュー
 sys スキーマ, 4470
PS_CURRENT_THREAD_ID() 関数, 2126
ps_is_account_enabled() 関数
 sys スキーマ, 4512
ps_is_consumer_enabled() 関数
 sys スキーマ, 4513
ps_is_instrument_default_enabled() 関数
 sys スキーマ, 4513
ps_is_instrument_default_timed() 関数
 sys スキーマ, 4513
ps_is_thread_instrumented() 関数

sys スキーマ, 4514
ps_setup_disable_background_threads() プロシージャ
 sys スキーマ, 4494
ps_setup_disable_consumer() プロシージャ
 sys スキーマ, 4494
ps_setup_disable_instrument() プロシージャ
 sys スキーマ, 4494
ps_setup_disable_thread() プロシージャ
 sys スキーマ, 4495
ps_setup_enable_background_threads() プロシージャ
 sys スキーマ, 4495
ps_setup_enable_consumer() プロシージャ
 sys スキーマ, 4496
ps_setup_enable_instrument() プロシージャ
 sys スキーマ, 4496
ps_setup_enable_thread() プロシージャ
 sys スキーマ, 4497
ps_setup_reload_saved() プロシージャ
 sys スキーマ, 4497
ps_setup_reset_to_default() プロシージャ
 sys スキーマ, 4497
ps_setup_save() プロシージャ
 sys スキーマ, 4498
ps_setup_show_disabled() プロシージャ
 sys スキーマ, 4498
ps_setup_show_disabled_consumers() プロシージャ
 sys スキーマ, 4499
ps_setup_show_disabled_instruments() プロシージャ
 sys スキーマ, 4499
ps_setup_show_enabled() プロシージャ
 sys スキーマ, 4499
ps_setup_show_enabled_consumers() プロシージャ
 sys スキーマ, 4500
ps_setup_show_enabled_instruments() プロシージャ
 sys スキーマ, 4501
ps_statement_avg_latency_histogram() プロシージャ
 sys スキーマ, 4501
ps_thread_account() 関数
 sys スキーマ, 4514
PS_THREAD_ID() 関数, 2126
ps_thread_id() 関数
 sys スキーマ, 4515
ps_thread_stack() 関数
 sys スキーマ, 4515
ps_thread_trx_info() 関数
 sys スキーマ, 4516
ps_trace_statement_digest() プロシージャ
 sys スキーマ, 4502
ps_trace_thread() プロシージャ
 sys スキーマ, 4503
ps_truncate_all_tables() プロシージャ
 sys スキーマ, 4504
Pthreads, 5348
PURGE BINARY LOGS, 2395
PURGE MASTER LOGS, 2395
purge, 2745
Purging old relay logs
 スレッドの状態, 1618
Python, 5348
Python API, 5348

Python, 4525
 サードパーティードライバ, 4526

Q

Qcache_free_blocks
 削除された機能, 39
Qcache_free_memory
 削除された機能, 39
Qcache_inserts
 削除された機能, 39, 39
Qcache_lowmem_prunes
 削除された機能, 39
Qcache_not_cached
 削除された機能, 39
Qcache_queries_in_cache
 削除された機能, 39
Qcache_total_blocks
 削除された機能, 39
QUARTER(), 1893
query end
 スレッドの状態, 1618
Query
 スレッドのコマンド, 1615
query-all オプション
 ndb_config, 3750
query_alloc_block_size システム変数, 759
query_attributes コマンド
 mysql, 382
query_attributes コンポーネント, 945
query_cache_limit
 削除された機能, 39
query_cache_min_res_unit
 削除された機能, 39
query_cache_size
 削除された機能, 39
query_cache_type
 削除された機能, 39
query_cache_wlock_invalidate
 削除された機能, 39
query_prealloc_size システム変数, 759
Queueing master event to the relay log
 スレッドの状態, 1622
QUICK
 DELETE 修飾子, 2293
Quit
 スレッドのコマンド, 1615
QUIT コマンド (NDB Cluster), 3841
quit コマンド
 mysql, 382
QUOTE(), 1629, 1909
quote-names オプション
 mysqldump, 431
quote_identifier() 関数
 sys スキーマ, 4517

R

R-tree, 5348
RADIANS(), 1879
RAID, 5349

RAND(), 1879
RANDOM_BYTES(), 1989
rand_seed1 システム変数, 760
rand_seed2 システム変数, 760
RANGE パーティショニング, 4034, 4040
RANGE パーティション
 管理, 4058
 追加および削除, 4058
range_alloc_block_size システム変数, 760
range_optimizer_max_mem_size システム変数, 760
RANK(), 2116
raw オプション
 mysql, 376
 mysqlbinlog, 531
RAW パーティション, 2674
raw バックアップ, 5349
rbr_exec_mode システム変数, 761
RC
 MySQL のリリース, 84
READ COMMITTED, 5349
READ COMMITTED
 NDB Cluster での実装, 3485
 トランザクション分離レベル, 2706
READ UNCOMMITTED, 5349
READ UNCOMMITTED
 トランザクション分離レベル, 2707
read-ahead
 random, 2734
 線形, 2734
read-from-remote-master オプション
 mysqlbinlog, 531
read-from-remote-server オプション
 mysqlbinlog, 531
Reading event from the relay log
 スレッドの状態, 1624
Reading master dump table data
 スレッドの状態, 1624
READ_BACKUP (NDB_TABLE)
 NDB Cluster, 2266
READ_BACKUP, 2266
read_buffer_size myisamchk 変数, 497
read_buffer_size システム変数, 761
read_firewall_groups() MySQL Enterprise Firewall UDF, 1369
read_firewall_group_allowlist() MySQL Enterprise Firewall UDF, 1369
read_firewall_users() MySQL Enterprise Firewall UDF, 1368
read_firewall_whitelist() MySQL Enterprise Firewall UDF, 1368
read_only システム変数, 762
read_rnd_buffer_size システム変数, 763
REAL データ型, 1765
RealtimeScheduler, 3625
REAL_AS_FLOAT SQL モード, 860
rebuild-indexes オプション
 ndb_restore, 3805
Rebuilding the index on master dump table
 スレッドの状態, 1624
ReceiveBufferMemory, 3711
reconnect オプション
 mysql, 376
Reconnecting after a failed binlog dump request
 スレッドの状態, 1622

Reconnecting after a failed master event read
スレッドの状態, 1622

RECOVER
XA トランザクション, 2391

recovery
InnoDB, 2938

RecoveryWork, 3597

Redo, 5349

redo ログ, 2695, 2695

Redo ログ, 5349

redo ログのアーカイブ, 2695, 5349
新機能, 27

RedoBuffer, 3616

RedoOverCommitCounter
データノード, 3645

RedoOverCommitLimit
データノード, 3645

references, 2176

REFERENCES 権限, 1050

REFERENTIAL_CONSTRAINTS
INFORMATION_SCHEMA テーブル, 4166

Refresh
スレッドのコマンド, 1615

ref_or_null, 1466

ref_or_null 結合タイプ
オプティマイザ, 1544

REGEXP, 1918

REGEXP 演算子, 1917

REGEXP_INSTR(), 1918

REGEXP_LIKE(), 1919

REGEXP_REPLACE(), 1920

regexp_stack_limit システム変数, 763

REGEXP_SUBSTR(), 1921

regexp_time_limit システム変数, 764

Register Slave
スレッドのコマンド, 1615

Registering slave on master
スレッドの状態, 1623

rehash コマンド
mysql, 383

relay-log-purge オプション
mysqld, 3081

relay-log-space-limit オプション
mysqld, 3081

relay_log システム変数, 3092

relay_log_basename システム変数, 3093

relay_log_index システム変数, 3093

relay_log_info_file
非推奨となった機能, 37

relay_log_info_file システム変数, 3094

relay_log_info_repository システム変数, 3094, 3163

relay_log_purge システム変数, 3095

relay_log_recovery システム変数, 3095

relay_log_space_limit システム変数, 3096

RELEASE SAVEPOINT, 2380

RELEASE_ALL_LOCKS(), 1992

RELEASE_LOCK(), 1992

reload オプション
ndb_mgmd, 3740

RELOAD 権限, 1051

- remap-column オプション
 - ndb_restore, 3805
- remove オプション
 - mysqld, 660
 - MySQLInstallerConsole, 126
 - ndbd, 3730
 - ndbmtd, 3730
 - ndb_mgmd, 3740
- Removing duplicates
 - スレッドの状態, 1619
- removing tmp table
 - スレッドの状態, 1619
- rename database, 2288
- rename
 - スレッドの状態, 1619
- rename result table
 - スレッドの状態, 1619
- RENAME TABLE, 2287
- RENAME USER ステートメント, 2514
- Reopen tables
 - スレッドの状態, 1619
- Repair by sorting
 - スレッドの状態, 1619
- Repair done
 - スレッドの状態, 1619
- REPAIR TABLE
 - およびパーティショニング, 4071
 - およびレプリケーション, 3225
- REPAIR TABLE ステートメント, 2534
 - options, 2535
 - output, 2536
 - ストレージエンジンのサポート, 2535
 - パーティション化のサポート, 2535
 - レプリケーション, 2535
- Repair with keycache
 - スレッドの状態, 1619
- REPEAT, 2449
 - ラベル, 2444
- REPEAT(), 1909
- REPEATABLE READ, 5350
- REPEATABLE READ
 - トランザクション分離レベル, 2705
- REPLACE, 2323
- REPLACE(), 1909
- replicate-do-db オプション
 - mysqld, 3082
- replicate-do-table オプション
 - mysqld, 3085
- replicate-ignore-db オプション
 - mysqld, 3084
- replicate-ignore-table オプション
 - mysqld, 3086
- replicate-rewrite-db オプション
 - mysqld, 3086
- replicate-same-server-id オプション
 - mysqld, 3087
- replicate-wild-do-table オプション
 - mysqld, 3088
- replicate-wild-ignore-table オプション
 - mysqld, 3088

REPLICATION CLIENT 権限, 1051
REPLICATION SLAVE 権限, 1051
REPLICATION_APPLIER 権限, 1057
replication_applier_configuration
 performance_schema, 4338
replication_applier_status
 performance_schema, 4339
replication_applier_status_by_coordinator
 performance_schema, 4340
replication_applier_status_by_worker
 performance_schema, 4341
replication_asynchronous_connection_failover
 performance_schema, 4337
replication_connection_configuration
 performance_schema, 4332
replication_connection_status
 performance_schema, 4335
replication_optimize_for_static_plugin_config システム変数, 3096
replication_sender_observe_commit_only システム変数, 3096
REPLICATION_SLAVE_ADMIN 権限, 1057
REPORT コマンド (NDB Cluster), 3840
report_host システム変数, 3097
report_password システム変数, 3097
report_port システム変数, 3097
report_user システム変数, 3098
REPRODUCIBLE_BUILD オプション
 CMake, 208
Requesting binlog dump
 スレッドの状態, 1623
REQUIRE オプション
 ALTER USER, 2485
 CREATE USER ステートメント, 2496
require-row-format オプション
 mysqlbinlog, 532
RequireEncryptedBackup, 3623
require_row_format システム変数, 764
require_secure_transport システム変数, 764
ReservedConcurrentIndexOperations, 3588
ReservedConcurrentOperations, 3588
ReservedConcurrentScans, 3588
ReservedConcurrentTransactions, 3588
ReservedFiredTriggers, 3588
ReservedLocalScans, 3589
ReservedTransactionBufferMemory, 3589
RESET MASTER, 2396
RESET MASTER ステートメント, 2608
RESET PERSIST ステートメント, 809, 826, 2608
RESET QUERY CACHE
 削除された機能, 39
RESET REPLICA | SLAVE ALL, 2423
RESET REPLICA | SLAVE, 2423
RESET REPLICA | SLAVE ステートメント, 2608
RESET SLAVE | REPLICA ALL, 2424
RESET SLAVE | REPLICA, 2424
Reset stmt
 スレッドのコマンド, 1615
reset-replica.pl
 NDB Cluster レプリケーション, 4009
resetconnection コマンド
 mysql, 383

RESIGNAL, 2461
resolveip
 削除された機能, 42
resolve_stack_dump
 削除された機能, 42
resources
 ndbinfo テーブル, 3958
RESOURCE_GROUPS
 INFORMATION_SCHEMA テーブル, 4167
resource_groups テーブル
 データディクショナリテーブル, 896, 2520
RESOURCE_GROUP_ADMIN 権限, 1057
RESOURCE_GROUP_USER 権限, 1057
RESTART コマンド (NDB Cluster), 3840
RESTART ステートメント, 2609
RestartOnErrorInsert, 3604
RestartSubscriberConnectTimeout, 3615
restart_info
 ndbinfo テーブル, 3959
restore-data オプション
 ndb_restore, 3806
restore-epoch オプション
 ndb_restore, 3806
restore-meta オプション
 ndb_restore, 3806
restore-privilege-tables オプション
 ndb_restore, 3807
result-file オプション
 mysqlbinlog, 532
 mysqldump, 431
 mysqlpump, 460
resultset_metadata システム変数, 765
retries オプション
 ndb_desc, 3760
return (r), 1628, 2070, 2312
RETURN, 2449
REVERSE(), 1909
REVOKE ステートメント, 1077, 2515
rewrite-database オプション
 ndb_restore, 3807
rewrite-db オプション
 mysqlbinlog, 532
rewriter_enabled システム変数, 965
Rewriter_number_loaded_rules ステータス変数, 965
Rewriter_number_reloads ステータス変数, 966
Rewriter_number_rewritten_queries ステータス変数, 966
Rewriter_reload_error ステータス変数, 966
rewriter_verbose システム変数, 965
RIGHT JOIN, 1456, 2334
RIGHT OUTER JOIN, 2334
RIGHT(), 1909
RLIKE, 1918
roles, 1081
 default, 2516
 削除, 2501
 作成, 2490
 ストアドプログラム, 1085
 取消し, 2515
 ビュー, 1085
 付与, 2502

割当て, 2519
ROLES_GRAPHML(), 2001
ROLE_ADMIN 権限, 1057
ROLE_COLUMN_GRANTS
 INFORMATION_SCHEMA テーブル, 4167
role_edges テーブル
 システムテーブル, 898, 1063
ROLE_ROUTINE_GRANTS
 INFORMATION_SCHEMA テーブル, 4168
ROLE_TABLE_GRANTS
 INFORMATION_SCHEMA テーブル, 4169
ROLLBACK, 2376
 XA トランザクション, 2391
ROLLBACK TO SAVEPOINT, 2380
Rolling back
 スレッドの状態, 1619
ROLLUP, 2098
root パスワード, 230
root ユーザー, 1032
 パスワードのリセット, 4599
ROUND(), 1881
ROUTINES
 INFORMATION_SCHEMA テーブル, 4169
ROW, 2348
rowbatch オプション
 ndb_import, 3773
rowbytes オプション
 ndb_import, 3773
rowid オプション
 ndb_select_all, 3816
rows オプション
 ndb_config, 3750
ROW_COUNT(), 2002
ROW_FORMAT
 COMPACT, 2779
 COMPRESSED, 2761, 2780
 DYNAMIC, 2780
 REDUNDANT, 2778
ROW_NUMBER(), 2116
RPAD(), 1910
rpl_read_size システム変数, 3098
Rpl_semi_sync_master_clients ステータス変数, 848
rpl_semi_sync_master_enabled システム変数, 3077
Rpl_semi_sync_master_net_avg_wait_time ステータス変数, 848
Rpl_semi_sync_master_net_waits ステータス変数, 848
Rpl_semi_sync_master_net_wait_time ステータス変数, 848
Rpl_semi_sync_master_no_times ステータス変数, 848
Rpl_semi_sync_master_no_tx ステータス変数, 849
Rpl_semi_sync_master_status ステータス変数, 849
Rpl_semi_sync_master_timefunc_failures ステータス変数, 849
rpl_semi_sync_master_timeout システム変数, 3077
rpl_semi_sync_master_trace_level システム変数, 3077
Rpl_semi_sync_master_tx_avg_wait_time ステータス変数, 849
Rpl_semi_sync_master_tx_waits ステータス変数, 849
Rpl_semi_sync_master_tx_wait_time ステータス変数, 849
rpl_semi_sync_master_wait_for_slave_count システム変数, 3078
rpl_semi_sync_master_wait_no_slave システム変数, 3078
rpl_semi_sync_master_wait_point システム変数, 3079
Rpl_semi_sync_master_wait_pos_backtraverse ステータス変数, 849
Rpl_semi_sync_master_wait_sessions ステータス変数, 849

Rpl_semi_sync_master_yes_tx ステータス変数, 849
rpl_semi_sync_slave_enabled システム変数, 3099
Rpl_semi_sync_slave_status ステータス変数, 850
rpl_semi_sync_slave_trace_level システム変数, 3099
rpl_stop_slave_timeout システム変数, 3099
RPM パッケージマネージャー, 157
RPM ファイル, 153, 157
Rsa_public_key ステータス変数, 850
RTRIM(), 1910
Ruby, 5350
Ruby API, 5350
Ruby API, 4526
rw ロック (読み書きロック), 5350
rlock_instances テーブル
 performance_schema, 4292

S

safe-recover オプション
 myisamchk, 500
safe-updates オプション
 mysql, 376, 391
safe-updates モード, 391
safe-user-create オプション
 mysqld, 660
SafeNet KeySecure アプリケーション
 keyring_okv キーリングプラグイン, 1239
Sakila, 8
SASL, 2954
 authentication, 1176
SAVEPOINT, 2380
Saving state
 スレッドの状態, 1619
SchedulerExecutionTimer, 3626
SchedulerResponsiveness, 3626
SchedulerSpinTimer, 3626
SCHEMA イベント (NDB Cluster), 3865
SCHEMA(), 2002
SCHEMATA
 INFORMATION_SCHEMA テーブル, 4172
schemata テーブル
 データディクショナリテーブル, 896
SCHEMATA_EXTENSIONS
 INFORMATION_SCHEMA テーブル, 2157, 4173
schema_auto_increment_columns ビュー
 sys スキーマ, 4470
schema_definition_cache システム変数, 766
schema_index_statistics ビュー
 sys スキーマ, 4471
schema_object_overview ビュー
 sys スキーマ, 4472
SCHEMA_PRIVILEGES
 INFORMATION_SCHEMA テーブル, 4173
schema_redundant_indexes ビュー
 sys スキーマ, 4472
schema_tables_with_full_table_scans ビュー
 sys スキーマ, 4478
schema_table_lock_waits ビュー
 sys スキーマ, 4473
schema_table_statistics ビュー

- sys スキーマ, 4475
- schema_table_statistics_with_buffer ビュー
 - sys スキーマ, 4476
- schema_unused_indexes ビュー
 - sys スキーマ, 4478
- SCI (NDB Cluster) (廃止), 3724
- SDI, 5350
- SDI, 483, 2297, 5350
- Searching rows for update
 - スレッドの状態, 1619
- SECOND(), 1893
- secondary_engine_cost_threshold システム変数, 765
- Secondary_engine_execution_count ステータス変数, 850
- secure_auth
 - 削除された機能, 39
- secure_file_priv システム変数, 766
- security
 - NDB ユーティリティー, 3989
 - 悪意のある SQL ステートメント, 3987
 - コンポーネント, 1151
 - 新機能, 9
 - プラグイン, 1151
- SEC_TO_TIME(), 1893
- SELECT INTO TABLE, 70
- SELECT
 - INTO, 2331
 - LIMIT, 2325
 - 最適化, 1539, 2611
- SELECT 権限, 1051
- select-limit オプション
 - mysql, 376
- Select_full_join ステータス変数, 850
- Select_full_range_join ステータス変数, 850
- select_into_buffer_size, 767
- select_into_disk_sync, 767
- select_into_disk_sync_delay, 768
- Select_range ステータス変数, 850
- Select_range_check ステータス変数, 850
- Select_scan ステータス変数, 850
- SELinux, 1395
 - LDAP 認証, 1180
 - mode, 1396
 - MySQL Router TCP ポートコンテキスト, 1399
 - MySQL Server ポリシー, 1397
 - MySQL 機能 TCP ポートコンテキスト, 1399
 - MySQL データディレクトリコンテキスト, 1397
 - mysqld TCP ポートコンテキスト, 1399
 - PID ファイルコンテキスト, 1398
 - secure_file_priv ディレクトリコンテキスト, 1398
 - status, 1396
 - TCP ポートコンテキスト, 1398
 - Unix ドメインファイルコンテキスト, 1398
 - エラーログファイルコンテキスト, 1398
 - グループアプリケーション TCP ポートコンテキスト, 1399
 - ドキュメントストア TCP ポートコンテキスト, 1399
 - トラブルシューティング, 1400
 - ファイルコンテキスト, 1397
- SendBufferMemory, 3712, 3718
- Sending binlog event to slave
 - スレッドの状態, 1622

SendSignalId, 3712, 3718
SEQUENCE, 292
SERIAL DEFAULT VALUE, 1824
SERIAL, 1762, 1764
SERIALIZABLE, 5350
SERIALIZABLE
 トランザクション分離レベル, 2707
server
 shutdown, 229
 シグナル処理, 552
server-id オプション
 mysqlbinlog, 533
server-id-bit オプション
 mysqlbinlog, 533
server-log-file オプション
 ndb_setup.py, 3820
server-name オプション
 ndb_setup.py, 3821
server-public-key-path オプション, 315
 mysql, 376
 mysqladmin, 402
 mysqlbinlog, 533
 mysqlcheck, 412
 mysqldump, 423
 mysqlimport, 445
 mysqlpump, 460
 mysqlshow, 471
 mysqlslap, 481
 mysql_upgrade, 360
ServerPort, 3577
servers テーブル
 システムテーブル, 900
server_cost
 システムテーブル, 1582
server_cost テーブル
 システムテーブル, 899
server_id システム変数, 3065
server_id_bits システム変数, 3686, 3686
server_locks
 ndbinfo テーブル, 3962
server_operations
 ndbinfo テーブル, 3963
server_transactions
 ndbinfo テーブル, 3965
server_uuid システム変数
 mysqld, 3066
service-startup-timeout オプション
 mysql.server, 342
SERVICE_CONNECTION_ADMIN 権限, 1057
service_get_read_locks() UDF
 ロックサービス, 1005
service_get_write_locks() UDF
 ロックサービス, 1005
service_release_locks() UDF
 ロックサービス, 1005
servlet, 5351
SESSION
 SET ステートメント, 2542
session_account_connect_attrs テーブル
 performance_schema, 4328

session_connect_attrs テーブル
performance_schema, 4328

session_ssl_status ビュー
sys スキーマ, 4478

SESSION_STATUS
削除された機能, 41

session_track_gtids, 768

session_track_schema システム変数, 769

session_track_state_change システム変数, 769

session_track_system_variables システム変数, 770

session_track_transaction_info システム変数, 770

SESSION_USER(), 2002

SESSION_VARIABLES
削除された機能, 41

SESSION_VARIABLES_ADMIN 権限, 1057

SET CHARACTER SET ステートメント, 2546

SET CHARSET ステートメント, 2546

SET DEFAULT ROLE ステートメント, 2516

SET GLOBAL sql_slave_skip_counter, 3149

SET GLOBAL ステートメント, 809

SET NAMES, 1707

SET NAMES ステートメント, 2546

Set option
スレッドのコマンド, 1615

SET PASSWORD ステートメント, 2517

SET PERSIST ステートメント, 809

SET PERSIST_ONLY ステートメント, 809

SET RESOURCE GROUP ステートメント, 2523

SET ROLE ステートメント, 2519

SET SESSION ステートメント, 809

SET
CHARACTER SET, 1701
NAMES, 1701
サイズ, 1830

SET sql_log_bin, 2397

SET TRANSACTION, 2387

SET ステートメント
CHARACTER SET, 2546
CHARSET, 2546
NAMES, 2546
代入演算子, 1868
変数の割当て, 826, 2542

SET データ型, 1784, 1791

set-auto-increment[option
myisamchk, 501

set-charset オプション
mysqlbinlog, 533
mysqldump, 428
mysqlpump, 460

set-collation オプション
myisamchk, 500

set-gtid-purge オプション
mysqldump, 429
mysqlpump, 460

setup
スレッドの状態, 1620

setup.bat
NDB Cluster (Windows), 3818

setup_actors テーブル
performance_schema, 4282

setup_consumers テーブル
 performance_schema, 4283
setup_instruments テーブル
 performance_schema, 4284
setup_objects テーブル
 performance_schema, 4286
setup_threads テーブル
 performance_schema, 4288
setup_timers
 削除された機能, 42
set_firewall_group_mode() MySQL Enterprise Firewall UDF, 1369
set_firewall_mode() MySQL Enterprise Firewall UDF, 1369
SET_USER_ID 権限, 1058
 孤立したストアオブジェクト, 4117
 ストアオブジェクトの作成, 4117
SET_VAR オプティマイザヒント, 1576
SHA(), 1989
SHA1(), 1989
SHA2(), 1989
sha256_password
 非推奨となった機能, 35
sha256_password 認証プラグイン, 1157
sha256_password_auto_generate_rsa_keys システム変数, 771
sha256_password_private_key_path システム変数, 771
sha256_password_proxy_users システム変数, 772, 1121
sha256_password_public_key_path システム変数, 772
sha2_cache_cleaner 監査プラグイン, 1156
shared-memory-base-name オプション, 313
 mysql, 377
 mysqladmin, 402
 mysqlbinlog, 533
 mysqlcheck, 412
 mysqldump, 437
 mysqlexport, 446
 mysqlshow, 471
 mysqlslap, 481
 mysql_upgrade, 361
SharedGlobalMemory, 3640
shared_memory システム変数, 773
shared_memory_base_name システム変数, 773
ShmKey, 3718
ShmSize, 3719
ShmSpinTime, 3719
SHOW BINARY LOGS ステートメント, 2547, 2547
SHOW BINLOG EVENTS ステートメント, 2547, 2548
SHOW CHARACTER SET ステートメント, 2547, 2549
SHOW COLLATION ステートメント, 2547, 2549
SHOW COLUMNS ステートメント, 2547, 2550
SHOW CREATE DATABASE ステートメント, 2547, 2552
SHOW CREATE EVENT ステートメント, 2547
SHOW CREATE FUNCTION ステートメント, 2547, 2553
SHOW CREATE PROCEDURE ステートメント, 2547, 2553
SHOW CREATE SCHEMA ステートメント, 2547, 2552
SHOW CREATE TABLE ステートメント, 2547, 2554
SHOW CREATE TRIGGER ステートメント, 2547, 2555
SHOW CREATE USER ステートメント, 2556
SHOW CREATE VIEW ステートメント, 2547, 2556
SHOW DATABASES 権限, 1051
SHOW DATABASES ステートメント, 2547, 2557
SHOW ENGINE INNODB STATUS ステートメント, 2557

SHOW ENGINE NDB STATUS, 3976
SHOW ENGINE NDBCLUSTER STATUS, 3976
SHOW ENGINE
 and NDB Cluster, 3976
SHOW ENGINE ステートメント, 2547, 2557
SHOW ENGINES
 and NDB Cluster, 3976
SHOW ENGINES ステートメント, 2547, 2561
SHOW ERRORS ステートメント, 2547, 2563
SHOW EVENTS ステートメント, 2547, 2563
SHOW FIELDS ステートメント, 2547, 2550
SHOW FUNCTION CODE ステートメント, 2547, 2565
SHOW FUNCTION STATUS ステートメント, 2547, 2565
SHOW GRANTS ステートメント, 2547, 2565
SHOW INDEX ステートメント, 2547, 2568
SHOW KEYS ステートメント, 2547, 2568
SHOW MASTER LOGS ステートメント, 2547, 2547
SHOW MASTER STATUS ステートメント, 2547, 2570
SHOW OPEN TABLES ステートメント, 2547, 2570
SHOW PLUGINS ステートメント, 2547, 2571
SHOW PRIVILEGES ステートメント, 2547, 2572
SHOW PROCEDURE CODE ステートメント, 2547, 2572
SHOW PROCEDURE STATUS ステートメント, 2547, 2573
SHOW PROCESSLIST ステートメント, 2547, 2574
SHOW PROFILE ステートメント, 2547, 2576
SHOW PROFILES ステートメント, 2547, 2576, 2578
SHOW RELAYLOG EVENTS ステートメント, 2547, 2579
SHOW REPLICA | SLAVE STATUS ステートメント, 2547, 2581
SHOW REPLICAS | SHOW SLAVE HOSTS ステートメント, 2547, 2580
SHOW SCHEDULER STATUS, 4108
SHOW SCHEMAS ステートメント, 2557
SHOW
 NDB Cluster 管理クライアント, 3542
SHOW SLAVE HOSTS | SHOW REPLICAS ステートメント, 2580
SHOW SLAVE | REPLICA STATUS ステートメント, 2588
SHOW STATUS
 and NDB Cluster, 3980
SHOW STATUS ステートメント, 2547, 2588
SHOW STORAGE ENGINES ステートメント, 2561
SHOW TABLE STATUS ステートメント, 2547, 2589
SHOW TABLES ステートメント, 2547, 2592
SHOW TRIGGERS ステートメント, 2547, 2593
SHOW VARIABLES
 and NDB Cluster, 3977
SHOW VARIABLES ステートメント, 2547, 2594
SHOW VIEW 権限, 1051
SHOW WARNINGS ステートメント, 2547, 2596
show オプション
 my_print_defaults, 548
SHOW 拡張, 4234
SHOW コマンド (NDB Cluster), 3839
show-create-skip-secondary-engine オプション
 mysqldump, 432
show-slave-auth-info オプション
 mysqld, 3072
show-table-type オプション
 mysqlshow, 471
show-temp-status オプション
 ndb_show_tables, 3822
show-warnings オプション

- mysql, 377
- mysqladmin, 402
- show_compatibility_56
 - 削除された機能, 41
- show_create_table_skip_secondary_engine システム変数, 773
- show_create_table_verbosity システム変数, 774
- show_old_temporals システム変数, 774
- SHOW_ROUTINE 権限, 1058
- shutdown
 - server, 892
- Shutdown
 - スレッドのコマンド, 1616
- SHUTDOWN 権限, 1052
- SHUTDOWN コマンド (NDB Cluster), 3842
- SHUTDOWN ステートメント, 2610
- shutdown-timeout オプション
 - mysqladmin, 402
- Shutting down
 - スレッドの状態, 1625
- SIGHUP シグナル
 - サーバーレスポンス, 552, 2601
 - ログメンテナンス, 941
- SIGINT シグナル
 - mysql クライアント, 377
 - クライアント応答, 554
 - サーバーレスポンス, 552, 1019
- sigint-ignore オプション
 - mysql, 377
- SIGN(), 1882
- SIGNAL, 2465
- SigNum, 3719
- SIGPIPE シグナル
 - クライアント応答, 554
- SIGTERM シグナル
 - サーバーレスポンス, 552, 2610
- SIGUSR1 シグナル
 - サーバーレスポンス, 552, 2601
 - ログメンテナンス, 941
- SIN(), 1882
- SINGLEUSER イベント (NDB Cluster), 3866
- SKIP LOCKED, 2330
- skip-admin-ssl オプション
 - mysqld, 645
- skip-broken-objects オプション
 - ndb_restore, 3808
- skip-column-names オプション
 - mysql, 377
- skip-comments オプション
 - mysqldump, 427
- skip-data オプション
 - ibd2sdi, 484
- skip-database オプション
 - mysqlcheck, 412
- skip-definer オプション
 - mysqlpump, 461
- skip-dump-rows オプション
 - mysqlpump, 461
- skip-grant-tables オプション
 - mysqld, 660
- skip-gtids オプション

- mysqlbinlog, 533
- skip-host-cache オプション
 - mysqld, 661
- skip-innodb オプション
 - mysqld, 662, 2820
- skip-kill-mysqld オプション
 - mysqld_safe, 339
- skip-line-numbers オプション
 - mysql, 377
- skip-ndbcluster オプション
 - mysqld, 3667
- skip-new オプション
 - mysqld, 662
- skip-nodegroup オプション
 - ndb_error_reporter, 3762
- skip-opt オプション
 - mysqldump, 436
- skip-show-database オプション
 - mysqld, 662
- skip-slave-start オプション
 - mysqld, 3089
- skip-ssl オプション
 - mysqld, 664
- skip-stack-trace オプション
 - mysqld, 662
- skip-symbolic-links オプション
 - mysqld, 665
- skip-sys-schema オプション
 - mysql_upgrade, 361
- skip-syslog オプション
 - mysqld_safe, 339
- skip-table-check オプション
 - ndb_restore, 3808
- skip-unknown-objects オプション
 - ndb_restore, 3808
- skip_external_locking システム変数, 774
- skip_name_resolve システム変数, 775
- skip_networking システム変数, 775
- SKIP_SCAN, 1573
- skip_show_database システム変数, 776
- Slave has read all relay log; waiting for more updates
 - スレッドの状態, 1624
- slave-skip-errors オプション
 - mysqld, 3089
- slave-sql-verify-checksum オプション
 - mysqld, 3090
- slave_allow_batching, 4004
- slave_allow_batching システム変数, 3686
- slave_checkpoint_group システム変数, 3100
- slave_checkpoint_period システム変数, 3100
- slave_compressed_protocol
 - 非推奨となった機能, 37
- slave_compressed_protocol システム変数, 3101
- slave_exec_mode システム変数, 3102
- Slave_heartbeat_period
 - 削除された機能, 41
- Slave_last_heartbeat
 - 削除された機能, 41
- slave_load_tmpdir システム変数, 3102
- slave_master_info テーブル

- システムテーブル, 899
- slave_max_allowed_packet システム変数, 3103
- slave_net_timeout システム変数, 3103
- Slave_open_temp_tables ステータス変数, 850
- slave_parallel_type システム変数, 3104
- slave_parallel_workers システム変数, 3105
- slave_pending_jobs_size_max システム変数, 3105
- slave_preserve_commit_order, 3106
- Slave_received_heartbeats
 - 削除された機能, 41
- slave_relay_log_info テーブル
 - システムテーブル, 899
- Slave_retried_transactions
 - 削除された機能, 41
- Slave_rows_last_search_algorithm_used ステータス変数, 850
- slave_rows_search_algorithms システム変数, 3107
- Slave_running
 - 削除された機能, 41
- slave_skip_errors システム変数, 3108
- slave_sql_verify_checksum システム変数, 3108
- slave_transaction_retries システム変数, 3108
- slave_type_conversions システム変数, 3109
- slave_worker_info テーブル
 - システムテーブル, 899
- Sleep
 - スレッドのコマンド, 1616
- sleep オプション
 - mysqladmin, 402
- SLEEP(), 2138
- sleep-time オプション
 - ndb_top, 3828
- slice-id オプション
 - ndb_restore, 3808
- slow-start-timeout オプション
 - mysqld, 662
- Slow_launch_threads ステータス変数, 850
- slow_launch_time システム変数, 776
- slow_log テーブル
 - システムテーブル, 898
- Slow_queries ステータス変数, 851
- slow_query_log システム変数, 776
- slow_query_log_file システム変数, 777
- SMALLINT データ型, 1763
- SNAPSHOTEND (START BACKUP コマンド), 3885
- SNAPSHOTSTART (START BACKUP コマンド), 3885
- socket オプション, 313
 - mysql, 377
 - mysqladmin, 403
 - mysqlbinlog, 534
 - mysqlcheck, 412
 - mysqld, 662
 - mysqldump, 424
 - mysqld_safe, 339
 - mysqlimport, 446
 - mysqlpump, 461
 - mysqlshow, 471
 - mysqlslap, 482
 - mysql_config, 547
 - mysql_secure_installation, 349
 - mysql_upgrade, 361

- ndb_top, 3828
- socket_instances テーブル
 - performance_schema, 4293
- socket_summary_by_event_name テーブル
 - performance_schema, 4389
- socket_summary_by_instance テーブル
 - performance_schema, 4389
- Solaris
 - インストール, 180
- Solaris x86_64 の問題, 1530
- Solaris トラブルシューティング, 219
- Solaris のインストールの問題, 180
- SOME, 2346
- sort-index オプション
 - myisamchk, 501
- sort-records オプション
 - myisamchk, 501
- sort-recover オプション
 - myisamchk, 500
- Sorting for group
 - スレッドの状態, 1620
- Sorting for order
 - スレッドの状態, 1620
- Sorting index
 - スレッドの状態, 1620
- Sorting result
 - スレッドの状態, 1620
- sort_buffer_size myisamchk 変数, 497
- sort_buffer_size システム変数, 777
- sort_key_blocks myisamchk 変数, 497
- Sort_merge_passes ステータス変数, 851
- Sort_range ステータス変数, 851
- Sort_rows ステータス変数, 851
- Sort_scan ステータス変数, 851
- SOUNDEX(), 1910
- SOUNDS LIKE, 1910
- source, 5351
- source (mysql クライアントコマンド), 287, 389
- source コマンド
 - mysql, 383
- SPACE(), 1910
- SPATIAL インデックス
 - InnoDB 述語ロック, 2705
 - 最適化, 1499
- speed
 - クエリー, 1432
- SpinMethod, 3626
- sporadic-binlog-dump-fail オプション
 - mysqld, 3116
- Spring, 5351
- sp_firewall_group_delist() MySQL Enterprise Firewall ストアドプロシージャ, 1366
- sp_firewall_group_enlist() MySQL Enterprise Firewall ストアドプロシージャ, 1366
- sp_reload_firewall_group_rules() MySQL Enterprise Firewall ストアドプロシージャ, 1366
- sp_reload_firewall_rules() MySQL Enterprise Firewall ストアドプロシージャ, 1365
- sp_set_firewall_group_mode() MySQL Enterprise Firewall ストアドプロシージャ, 1367
- sp_set_firewall_group_mode_and_user() MySQL Enterprise Firewall ストアドプロシージャ, 1368
- sp_set_firewall_mode() MySQL Enterprise Firewall ストアドプロシージャ, 1365
- SQL, 5351
- SQL SECURITY
 - 権限への影響, 4116

SQL
 定義, 4
SQL スクリプト, 362
SQL ステートメント
 レプリカ, 2398
 レプリケーションサーバー, 2433
 レプリケーションソース, 2395
SQL ノード (NDB Cluster), 3889
 定義済, 3450
SQL モード, 854
 ALLOW_INVALID_DATES, 856
 ANSI, 855, 860
 ANSI_QUOTES, 856
 ERROR_FOR_DIVISION_BY_ZERO, 856
 HIGH_NOT_PRECEDENCE, 856
 IGNORE_SPACE, 856
 NO_AUTO_VALUE_ON_ZERO, 857
 NO_BACKSLASH_ESCAPES, 857
 NO_DIR_IN_CREATE, 857
 NO_ENGINE_SUBSTITUTION, 857
 NO_UNSIGNED_SUBTRACTION, 857
 NO_ZERO_DATE, 858
 NO_ZERO_IN_DATE, 858
 ONLY_FULL_GROUP_BY, 859, 2104
 PAD_CHAR_TO_FULL_LENGTH, 859
 PIPES_AS_CONCAT, 860
 REAL_AS_FLOAT, 860
 strict, 855
 STRICT_ALL_TABLES, 860
 STRICT_TRANS_TABLES, 855, 860
 TIME_TRUNCATE_FRACTIONAL, 860
 TRADITIONAL, 855, 861
 およびパーティショニング, 3228, 4082
 およびレプリケーション, 3228
 削除された機能, 39
SQL-92
 への拡張, 66
sql-mode オプション
 mysqld, 663
 mysqlslap, 482
SQLState, 5351
sql_auto_is_null システム変数, 778
SQL_BIG_RESULT
 SELECT 修飾子, 2331
sql_big_selects システム変数, 778
SQL_BUFFER_RESULT
 SELECT 修飾子, 2331
sql_buffer_result システム変数, 779
SQL_CACHE
 SELECT 修飾子, 2331
 削除された機能, 39
SQL_CALC_FOUND_ROWS, 1473
 SELECT 修飾子, 2331
 非推奨となった機能, 36
sql_log_bin
 削除された機能, 39
sql_log_bin システム変数, 3136
sql_log_off システム変数, 779
sql_mode システム変数, 779
sql_notes システム変数, 780

SQL_NO_CACHE

SELECT 修飾子, 2331

sql_quote_show_create システム変数, 781

sql_require_primary_key システム変数, 781

sql_safe_updates システム変数, 391, 782

sql_select_limit システム変数, 391, 782

sql_slave_skip_counter, 3149

sql_slave_skip_counter システム変数, 3109

SQL_SMALL_RESULT

SELECT 修飾子, 2331

sql_warnings システム変数, 782

SQRT(), 1882

SRID 属性

空間データ型, 1794

SRID 値

空間関数による処理, 2007

SRID()

削除された機能, 40

SSD, 5351

SSD, 2761

SSH, 1036, 1150

SSL, 5351

SSL, 1129

X.509 基本, 1129

コマンドオプション, 314

接続の確立, 1130

SSL オプション

mysql, 377

mysqladmin, 403

mysqlbinlog, 534

mysqlcheck, 412

mysqldump, 424

mysqlimport, 446

mysqlpump, 461

mysqlshow, 471

mysqlslap, 482

mysql_secure_installation, 350

mysql_upgrade, 361

ssl オプション

mysqld, 664

SSL 関連オプション

ALTER USER, 2485

CREATE USER ステートメント, 2496

SSL ライブラリ

構成, 191

ssl-ca オプション, 315

ssl-capath オプション, 315

ssl-cert オプション, 315

ssl-cipher オプション, 316

ssl-crl オプション, 316

ssl-crlpath オプション, 316

ssl-fips-mode オプション, 316

mysql, 378

mysqladmin, 403

mysqlbinlog, 534

mysqlcheck, 412

mysqldump, 424

mysqlimport, 446

mysqlpump, 462

mysqlshow, 471

- mysqlslap, 482
- mysql_secure_installation, 350
- mysql_upgrade, 361
- ssl-key オプション, 317
- ssl-mode オプション, 317
- Ssl_accepts ステータス変数, 851
- Ssl_accept_renegotiates ステータス変数, 851
- ssl_ca システム変数, 783
- Ssl_callback_cache_hits ステータス変数, 851
- ssl_capath システム変数, 783
- ssl_cert システム変数, 783
- ssl_cipher システム変数, 784
- Ssl_cipher ステータス変数, 851
- Ssl_cipher_list ステータス変数, 851
- Ssl_client_connects ステータス変数, 851
- Ssl_connect_renegotiates ステータス変数, 851
- ssl_crl システム変数, 784
- ssl_crlpath システム変数, 785
- Ssl_ctx_verify_depth ステータス変数, 851
- Ssl_ctx_verify_mode ステータス変数, 851
- Ssl_default_timeout ステータス変数, 851
- Ssl_finished_accepts ステータス変数, 852
- Ssl_finished_connects ステータス変数, 852
- ssl_fips_mode システム変数, 785
- ssl_key システム変数, 786
- Ssl_server_not_after ステータス変数, 852
- Ssl_server_not_before ステータス変数, 852
- Ssl_sessions_reused ステータス変数, 852
- Ssl_session_cache_hits ステータス変数, 852
- Ssl_session_cache_misses ステータス変数, 852
- Ssl_session_cache_mode ステータス変数, 852
- Ssl_session_cache_overflows ステータス変数, 852
- Ssl_session_cache_size ステータス変数, 852
- Ssl_session_cache_timeouts ステータス変数, 852
- Ssl_used_session_cache_entries ステータス変数, 852
- Ssl_verify_depth ステータス変数, 852
- Ssl_verify_mode ステータス変数, 852
- Ssl_version ステータス変数, 853
- staging-tries オプション
 - ndb_move_data, 3781
- START BACKUP
 - NOWAIT, 3885
 - SNAPSHOTEND, 3885
 - SNAPSHOTSTART, 3885
 - WAIT COMPLETED, 3885
 - WAIT STARTED, 3885
 - 構文, 3884
- START GROUP_REPLICATION, 2433
- START REPLICA | SLAVE, 2425
- START SLAVE | REPLICA, 2428
- START
 - XA トランザクション, 2391
- START TRANSACTION, 2376
- START コマンド (NDB Cluster), 3839
- start-datetime オプション
 - mysqlbinlog, 534
- start-page オプション
 - innochecksum, 487
- start-position オプション
 - mysqlbinlog, 534

StartConnectBackoffMaxTime, 3655
StartFailRetryDelay, 3646
StartFailureTimeout, 3606
StartNoNodeGroupTimeout, 3606
StartPartialTimeout, 3606
StartPartitionedTimeout, 3606
StartPoint()
 削除された機能, 40
STARTUP イベント (NDB Cluster), 3862
StartupStatusReportFrequency, 3620
state-dir オプション
 ndb_import, 3774
statements_with_errors_or_warnings ビュー
 sys スキーマ, 4480
statements_with_full_table_scans ビュー
 sys スキーマ, 4481
statements_with_runtimes_in_95th_percentile ビュー
 sys スキーマ, 4482
statements_with_sorting ビュー
 sys スキーマ, 4483
statements_with_temp_tables ビュー
 sys スキーマ, 4484
statement_analysis ビュー
 sys スキーマ, 4479
STATEMENT_DIGEST(), 1989
STATEMENT_DIGEST_TEXT(), 1990
statement_performance_analyzer() プロシージャ
 sys スキーマ, 4504
Statistics
 スレッドのコマンド, 1616
statistics
 スレッドの状態, 1620
STATISTICS
 INFORMATION_SCHEMA テーブル, 4174
STATISTICS イベント (NDB Cluster), 3865
stats オプション
 myisam_ftdump, 492
 ndb_import, 3773
stats_method myisamchk 変数, 497
status
 テーブル, 2589
STATUS コマンド (NDB Cluster), 3840
status コマンド
 mysql, 383
 結果, 396
STD(), 2097
STDDEV(), 2097
STDDEV_POP(), 2098
STDDEV_SAMP(), 2098
STOP GROUP_REPLICATION, 2434
STOP REPLICAS | SLAVE, 2429
STOP SLAVE | REPLICAS, 2430
STOP コマンド (NDB Cluster), 3839
stop-datetime オプション
 mysqlbinlog, 534
stop-never オプション
 mysqlbinlog, 535
stop-never-slave-server-id オプション
 mysqlbinlog, 535
stop-position オプション

- mysqlbinlog, 535
- StopOnError, 3604
- stored_program_cache システム変数, 786
- stored_program_definition_cache システム変数, 786
- STRAIGHT_JOIN, 1457, 1539, 1552, 2334, 2612
 - SELECT 修飾子, 1481, 2331
 - 結合タイプ, 1481
- STRCMP(), 1917
- strict SQL モード, 855
- strict モード
 - default, 72
- strict-check オプション
 - ibd2sdi, 486
 - innochecksum, 488
- STRICT_ALL_TABLES SQL モード, 860
- STRICT_TRANS_TABLES SQL モード, 855, 860
- string
 - JSON, 1811
- StringMemory, 3582
- STR_TO_DATE(), 1893
- ST_Area(), 2021
- ST_AsBinary(), 2012
- ST_AsGeoJSON(), 2038
- ST_AsText(), 2013
- ST_Buffer(), 2024
- ST_Buffer_Strategy(), 2026
- ST_Centroid(), 2022
- ST_Collect(), 2040
- ST_Contains(), 2031
- ST_ConvexHull(), 2026
- ST_Crosses(), 2031
- ST_Difference(), 2026
- ST_Dimension(), 2014
- ST_Disjoint(), 2031
- ST_Distance(), 2031
- ST_Distance_Sphere(), 2042
- ST_EndPoint(), 2018
- ST_Envelope(), 2014
- ST_Equals(), 2032
- ST_ExteriorRing(), 2023
- ST_FrechetDistance(), 2032
- ST_GeoHash(), 2037
- ST_GeomCollFromText(), 2008
- ST_GeomCollFromWKB(), 2010
- ST_GeometryCollectionFromText(), 2008
- ST_GeometryCollectionFromWKB(), 2010
- ST_GeometryFromText(), 2008
- ST_GeometryFromWKB(), 2010
- ST_GeometryN(), 2024
- ST_GeometryType(), 2014
- ST_GEOMETRY_COLUMNS
 - INFORMATION_SCHEMA テーブル, 4176
- ST_GeomFromGeoJSON(), 2039
- ST_GeomFromText(), 2008
- ST_GeomFromWKB(), 2010
- ST_HausdorffDistance(), 2033
- ST_InteriorRingN(), 2023
- ST_Intersection(), 2026
- ST_Intersects(), 2034
- ST_IsClosed(), 2018

ST_IsEmpty(), 2015
ST_IsSimple(), 2015
ST_IsValid(), 2043
ST_LatFromGeoHash(), 2037
ST_Latitude(), 2016
ST_Length(), 2019
ST_LineFromText(), 2009
ST_LineFromWKB(), 2010
ST_LineInterpolatePoint(), 2027, 2028
ST_LineInterpolatePoints(), 2027
ST_LineStringFromText(), 2009
ST_LineStringFromWKB(), 2010
ST_LongFromGeoHash(), 2037
ST_Longitude(), 2017
ST_MakeEnvelope(), 2043
ST_MLineFromText(), 2009
ST_MLineFromWKB(), 2010
ST_MPointFromText(), 2009
ST_MPointFromWKB(), 2010
ST_MPolyFromText(), 2009
ST_MPolyFromWKB(), 2010
ST_MultiLineStringFromText(), 2009
ST_MultiLineStringFromWKB(), 2010
ST_MultiPointFromText(), 2009
ST_MultiPointFromWKB(), 2010
ST_MultiPolygonFromText(), 2009
ST_MultiPolygonFromWKB(), 2010
ST_NumGeometries(), 2024
ST_NumInteriorRing(), 2023
ST_NumInteriorRings(), 2023
ST_NumPoints(), 2020
ST_Overlaps(), 2034
ST_PointFromGeoHash(), 2038
ST_PointFromText(), 2009
ST_PointFromWKB(), 2010
ST_PointN(), 2021
ST_PolyFromText(), 2009
ST_PolyFromWKB(), 2010
ST_PolygonFromText(), 2009
ST_PolygonFromWKB(), 2010
ST_Simplify(), 2044
ST_SPATIAL_REFERENCE_SYSTEMS
INFORMATION_SCHEMA テーブル, 4177
st_spatial_reference_systems テーブル
データディクショナリテーブル, 896
ST_SRID(), 2015
ST_StartPoint(), 2021
ST_SwapXY(), 2013
ST_SymDifference(), 2028
ST_Touches(), 2034
ST_Transform(), 2028
ST_Union(), 2029
ST_UNITS_OF_MEASURE
INFORMATION_SCHEMA テーブル, 4178
ST_Validate(), 2045
ST_Within(), 2034
ST_X(), 2017
ST_Y(), 2018
SUBDATE(), 1894
SUBPARTITION BY KEY

既知の問題, 4087
SUBSTR(), 1910
SUBSTRING(), 1911
SUBSTRING_INDEX(), 1911
SUBTIME(), 1894
SUDDO_USER 環境変数, 4327
SUM(), 2098
SUM(DISTINCT), 2098
SUPER 権限, 1053
super-large-pages オプション
 mysqld, 664
super_read_only システム変数, 787
support
 オペレーティングシステム, 83, 84
 プラットフォーム用, 83
symbolic-links オプション
 mysqld, 665
Syncing ndb table schema operation and binlog
 スレッドの状態, 1625
sync_binlog システム変数, 3136
sync_frm
 削除された機能, 39
sync_master_info システム変数, 3110
sync_relay_log システム変数, 3110
sync_relay_log_info システム変数, 3111
sys スキーマ, 4239
 create_synonym_db() プロシージャ, 4491
 diagnostics() プロシージャ, 4492
 execute_prepared_stmt() プロシージャ, 4493
 extract_schema_from_file_name() 関数, 4509
 extract_table_from_file_name() 関数, 4509
 format_bytes() 関数, 4509
 format_path() 関数, 4510
 format_statement() 関数, 4510
 format_time() 関数, 4511
 host_summary ビュー, 4454
 host_summary_by_file_io ビュー, 4455
 host_summary_by_file_io_type ビュー, 4455
 host_summary_by_stages ビュー, 4455
 host_summary_by_statement_latency ビュー, 4456
 host_summary_by_statement_type ビュー, 4456
 innodb_buffer_stats_by_schema ビュー, 4457
 innodb_buffer_stats_by_table ビュー, 4458
 innodb_lock_waits ビュー, 4459
 io_by_thread_by_latency ビュー, 4460
 io_global_by_file_by_bytes ビュー, 4461
 io_global_by_file_by_latency ビュー, 4462
 io_global_by_wait_by_bytes ビュー, 4462
 io_global_by_wait_by_latency ビュー, 4463
 latest_file_io ビュー, 4464
 list_add() 関数, 4511
 list_drop() 関数, 4512
 memory_by_host_by_current_bytes ビュー, 4465
 memory_by_thread_by_current_bytes ビュー, 4465
 memory_by_user_by_current_bytes ビュー, 4466
 memory_global_by_current_bytes ビュー, 4466
 memory_global_total ビュー, 4467
 processlist ビュー, 4468
 ps_check_lost_instrumentation ビュー, 4470
 ps_is_account_enabled() 関数, 4512

ps_is_consumer_enabled() 関数, 4513
ps_is_instrument_default_enabled() 関数, 4513
ps_is_instrument_default_timed() 関数, 4514
ps_is_thread_instrumented() 関数, 4514
ps_setup_disable_background_threads() プロシージャ, 4494
ps_setup_disable_consumer() プロシージャ, 4494
ps_setup_disable_instrument() プロシージャ, 4495
ps_setup_disable_thread() プロシージャ, 4495
ps_setup_enable_background_threads() プロシージャ, 4495
ps_setup_enable_consumer() プロシージャ, 4496
ps_setup_enable_instrument() プロシージャ, 4496
ps_setup_enable_thread() プロシージャ, 4497
ps_setup_reload_saved() プロシージャ, 4497
ps_setup_reset_to_default() プロシージャ, 4497
ps_setup_save() プロシージャ, 4498
ps_setup_show_disabled() プロシージャ, 4498
ps_setup_show_disabled_consumers() プロシージャ, 4499
ps_setup_show_disabled_instruments() プロシージャ, 4499
ps_setup_show_enabled() プロシージャ, 4499
ps_setup_show_enabled_consumers() プロシージャ, 4500
ps_setup_show_enabled_instruments() プロシージャ, 4501
ps_statement_avg_latency_histogram() プロシージャ, 4501
ps_thread_account() 関数, 4514
ps_thread_id() 関数, 4515
ps_thread_stack() 関数, 4515
ps_thread_trx_info() 関数, 4516
ps_trace_statement_digest() プロシージャ, 4502
ps_trace_thread() プロシージャ, 4503
ps_truncate_all_tables() プロシージャ, 4504
quote_identifier() 関数, 4517
schema_auto_increment_columns ビュー, 4470
schema_index_statistics ビュー, 4471
schema_object_overview ビュー, 4472
schema_redundant_indexes ビュー, 4472
schema_tables_with_full_table_scans ビュー, 4478
schema_table_lock_waits ビュー, 4473
schema_table_statistics ビュー, 4475
schema_table_statistics_with_buffer ビュー, 4476
schema_unused_indexes ビュー, 4478
session_ssl_status ビュー, 4478
statements_with_errors_or_warnings ビュー, 4480
statements_with_full_table_scans ビュー, 4481
statements_with_runtimes_in_95th_percentile ビュー, 4482
statements_with_sorting ビュー, 4483
statements_with_temp_tables ビュー, 4484
statement_analysis ビュー, 4479
statement_performance_analyzer() プロシージャ, 4504
sys_config テーブル, 4451
sys_get_config() 関数, 4518
table_exists() プロシージャ, 4507
user_summary ビュー, 4485
user_summary_by_file_io ビュー, 4486
user_summary_by_file_io_type ビュー, 4486
user_summary_by_stages ビュー, 4486
user_summary_by_statement_latency ビュー, 4487
user_summary_by_statement_type ビュー, 4488
version_major() 関数, 4518
version_minor() 関数, 4519
version_patch() 関数, 4519
waits_by_host_by_latency ビュー, 4490

waits_by_user_by_latency ビュー, 4490
waits_global_by_latency ビュー, 4491
wait_classes_global_by_avg_latency ビュー, 4489
wait_classes_global_by_latency ビュー, 4489
x\$ ビュー, 4453
x\$host_summary ビュー, 4454
x\$host_summary_by_file_io ビュー, 4455
x\$host_summary_by_file_io_type ビュー, 4455
x\$host_summary_by_stages ビュー, 4455
x\$host_summary_by_statement_latency ビュー, 4456
x\$host_summary_by_statement_type ビュー, 4456
x\$innodb_buffer_stats_by_schema ビュー, 4457
x\$innodb_buffer_stats_by_table ビュー, 4458
x\$innodb_lock_waits ビュー, 4459
x\$io_by_thread_by_latency ビュー, 4460
x\$io_global_by_file_by_bytes ビュー, 4461
x\$io_global_by_file_by_latency ビュー, 4462
x\$io_global_by_wait_by_bytes ビュー, 4462
x\$io_global_by_wait_by_latency ビュー, 4463
x\$latest_file_io ビュー, 4464
x\$memory_by_host_by_current_bytes ビュー, 4465
x\$memory_by_thread_by_current_bytes ビュー, 4465
x\$memory_by_user_by_current_bytes ビュー, 4466
x\$memory_global_by_current_bytes ビュー, 4466
x\$memory_global_total ビュー, 4467
x\$processlist ビュー, 4468
x\$schema_flattened_keys ビュー, 4472
x\$schema_index_statistics ビュー, 4471
x\$schema_tables_with_full_table_scans ビュー, 4478
x\$schema_table_lock_waits ビュー, 4473
x\$schema_table_statistics ビュー, 4475
x\$schema_table_statistics_with_buffer ビュー, 4476
x\$session ビュー, 4478
x\$statements_with_errors_or_warnings ビュー, 4480
x\$statements_with_full_table_scans ビュー, 4481
x\$statements_with_runtimes_in_95th_percentile ビュー, 4482
x\$statements_with_sorting ビュー, 4483
x\$statements_with_temp_tables ビュー, 4484
x\$statement_analysis ビュー, 4479
x\$user_summary ビュー, 4485
x\$user_summary_by_file_io ビュー, 4486
x\$user_summary_by_file_io_type ビュー, 4486
x\$user_summary_by_stages ビュー, 4486
x\$user_summary_by_statement_latency ビュー, 4487
x\$user_summary_by_statement_type ビュー, 4488
x\$waits_by_host_by_latency ビュー, 4490
x\$waits_by_user_by_latency ビュー, 4490
x\$waits_global_by_latency ビュー, 4491
x\$wait_classes_global_by_avg_latency ビュー, 4489
x\$wait_classes_global_by_latency ビュー, 4489
オブジェクトの所有権, 4445
セッションビュー, 4478
バージョンビュー, 4488
メトリックビュー, 4467
sys-check オプション
 ndb_index_stat, 3778
sys-create オプション
 ndb_index_stat, 3778
sys-create-if-not-exist オプション
 ndb_index_stat, 3778

sys-create-if-not-valid オプション
 ndb_index_stat, 3778
sys-drop オプション
 ndb_index_stat, 3778
sys-skip-events オプション
 ndb_index_stat, 3779
sys-skip-tables オプション
 ndb_index_stat, 3779
SYSCONFDIR オプション
 CMake, 202
SYSDATE(), 1894
sysdate-is-no オプション
 mysqld, 665
syseventlog.facility システム変数, 788
syseventlog.include_pid システム変数, 788
syseventlog.tag システム変数, 788
syslog オプション
 mysql, 378
 mysqld_safe, 339
syslog-tag オプション
 mysqld_safe, 339
System lock
 スレッドの状態, 1620
system
 privilege, 1043
system コマンド
 mysql, 383
systemd
 「CMake SYSTEMD_PID_DIR」オプション, 202
 「CMake SYSTEMD_SERVICE_NAME」オプション, 203
 「CMake WITH_SYSTEMD」オプション, 214
 mysqld daemonize オプション, 647
 mysqld 終了コード, 894
 mysqld の管理, 175
SYSTEMD_PID_DIR オプション
 CMake, 202
SYSTEMD_SERVICE_NAME オプション
 CMake, 203
system_time_zone システム変数, 789
SYSTEM_USER 権限, 1058, 1087
SYSTEM_USER(), 2003
SYSTEM_VARIABLES_ADMIN 権限, 1059
sys_config テーブル
 sys スキーマ, 4451
sys_get_config() 関数
 sys スキーマ, 4518

T

tab (t), 1628, 2070, 2312
TABLE, 2358
TABLE ステートメント
 INTO を使用, 2332
 新機能, 30
tables
 圧縮, 509
TABLES
 INFORMATION_SCHEMA テーブル, 4179
TABLESPACES
 INFORMATION_SCHEMA テーブル, 4183

TABLESPACES_EXTENSIONS
 INFORMATION_SCHEMA テーブル, 4183
tablespace_definition_cache システム変数, 791
tablespace_files テーブル
 データディクショナリテーブル, 896
TABLE_EXTENSIONS
 INFORMATION_SCHEMA テーブル, 4182
tables_priv テーブル
 システムテーブル, 897, 1063
TABLE_CONSTRAINTS
 INFORMATION_SCHEMA テーブル, 4183
TABLE_CONSTRAINTS_EXTENSIONS
 INFORMATION_SCHEMA テーブル, 4184
table_definition_cache システム変数, 789
table_distribution_status
 ndbinfo テーブル, 3966
TABLE_ENCRYPTION_ADMIN 権限, 1059
table_encryption_privilege_check 変数, 790
table_exists() プロシージャ
 sys スキーマ, 4507
table_fragments
 ndbinfo テーブル, 3967
table_handles テーブル
 performance_schema, 4357
table_info
 ndbinfo テーブル, 3968
table_io_waits_summary_by_index_usage テーブル
 performance_schema, 4387
table_io_waits_summary_by_table テーブル
 performance_schema, 4386
Table_locks_immediate ステータス変数, 853
Table_locks_waited ステータス変数, 853
table_lock_waits_summary_by_table テーブル
 performance_schema, 4387
table_open_cache, 1516
table_open_cache システム変数, 790
Table_open_cache_hits ステータス変数, 853
table_open_cache_instances システム変数, 791
Table_open_cache_misses ステータス変数, 853
Table_open_cache_overflows ステータス変数, 853
table_partitions テーブル
 データディクショナリテーブル, 896
table_partition_values テーブル
 データディクショナリテーブル, 896
TABLE_PRIVILEGES
 INFORMATION_SCHEMA テーブル, 4184
table_replicas
 ndbinfo テーブル, 3969
table_stats テーブル
 データディクショナリテーブル, 896
TAN(), 1882
tar
 Solaris の問題, 180, 180
tc-heuristic-recover オプション
 mysqld, 665
Tcl, 5351
Tcl API, 4526
tcp-ip オプション
 mysqld_multi, 344
TCP/IP, 129, 134, 208, 208, 319, 338, 349, 375, 531, 547, 550, 659, 865, 1010, 1036, 1124, 4293, 4588

TcpSpinTime, 3712
TCP_MAXSEG_SIZE, 3713
TCP_RCV_BUF_SIZE, 3713
TCP_SND_BUF_SIZE, 3713
Tc_log_max_pages_used ステータス変数, 853
Tc_log_page_size ステータス変数, 853
Tc_log_page_waits ステータス変数, 853
tc_time_track_stats
 ndbinfo テーブル, 3970
tee オプション
 mysql, 378
tee コマンド
 mysql, 383
tempdelay オプション
 ndb_import, 3774
temperrors オプション
 ndb_import, 3774
TEMPORARY テーブル, 2244
 名前変更, 2288
TEMPORARY テーブルの権限, 1049, 2244, 2511
temptable_max_mmap システム変数, 791
temptable_max_ram システム変数, 792
temptable_use_mmap システム変数, 792
test_plugin_server 認証プラグイン, 1198
TEXT
 サイズ, 1829
TEXT カラム
 インデックス, 2195, 2195
 インデックス設定, 2222
 インデックス付け, 1500
 デフォルト値, 1787
TEXT データ型, 1783, 1787
thread/sql/compress_gtid_table, 3036
threadblocks
 ndbinfo テーブル, 3971
ThreadConfig, 3632
ThreadPool (参照 [DiskIOThreadPool](#))
threads, 2574
 display, 2574
 ndbinfo テーブル, 3972
 監視, 1612, 2574
threadstat
 ndbinfo テーブル, 3972
Threads_cached ステータス変数, 853
Threads_connected ステータス変数, 853
Threads_created ステータス変数, 854
Threads_running ステータス変数, 854
thread_cache_size システム変数, 792
thread_handling システム変数, 793
thread_pool_algorithm システム変数, 793
thread_pool_high_priority_connection システム変数, 794
thread_pool_max_active_query_threads システム変数, 794
thread_pool_max_unused_threads システム変数, 795
thread_pool_prio_kickup_timer システム変数, 795
thread_pool_size システム変数, 795
thread_pool_stall_limit システム変数, 796
thread_stack システム変数, 796
Time
 スレッドのコマンド, 1616
TIME データ型, 1772, 1775

TIME(), 1895
TimeBetweenEpochs, 3610
TimeBetweenEpochsTimeout, 3610
TimeBetweenGlobalCheckpoints, 3609, 3644
TimeBetweenGlobalCheckpointsTimeout, 3610
TimeBetweenInactiveTransactionAbortCheck, 3611
TimeBetweenLocalCheckpoints, 3609
TimeBetweenWatchDogCheck, 3605
TimeBetweenWatchDogCheckInitial, 3605
TIMEDIFF(), 1895
timeout, 690, 1991
timeout オプション
 ndb_waiter, 3831
TIMESTAMP
 インデックス, 1512
 および NULL 値, 4609
 およびレプリケーション, 3212
 初期化および更新機能, 1776
timestamp システム変数, 797
TIMESTAMP データ型, 1771, 1772
TIMESTAMP(), 1896
TIMESTAMPADD(), 1896
TIMESTAMPDIFF(), 1896
timezone オプション
 mysqld_safe, 339
time_format
 削除された機能, 39
TIME_FORMAT(), 1896
TIME_TO_SEC(), 1896
TIME_TRUNCATE_FRACTIONAL SQL モード, 860
time_zone
 新機能, 27
time_zone システム変数, 797
time_zone テーブル
 システムテーブル, 899
time_zone_leap_second テーブル
 システムテーブル, 899
time_zone_name テーブル
 システムテーブル, 899
time_zone_transition テーブル
 システムテーブル, 899
time_zone_transition_type テーブル
 システムテーブル, 899
TINYBLOB データ型, 1783
TINYINT データ型, 1763
TINYTEXT データ型, 1783
TLS, 1129
 コマンドオプション, 314
 接続の確立, 1130
TLS 関連オプション
 ALTER USER, 2485
 CREATE USER ステートメント, 2496
tls-ciphersuites オプション, 318
 mysql, 378
 mysqladmin, 403
 mysqlbinlog, 535
 mysqlcheck, 413
 mysqldump, 424
 mysqlimport, 446
 mysqlpump, 462

- mysqlshow, 472
- mysqlslap, 482
- mysql_secure_installation, 350
- mysql_upgrade, 361
- tls-version オプション, 318
- mysql, 378
 - mysqladmin, 403
 - mysqlbinlog, 535
 - mysqlcheck, 413
 - mysqldump, 424
 - mysqlimport, 447
 - mysqlpump, 462
 - mysqlshow, 472
 - mysqlslap, 482
 - mysql_secure_installation, 350
 - mysql_upgrade, 362
- tls_channel_status テーブル
 - performance_schema, 4413
- tls_ciphersuites システム変数, 798
- tls_version システム変数, 798
- TMPDIR オプション
 - CMake, 203
- tmpdir オプション
 - myisamchk, 500
 - myisampack, 510
 - mysqld, 666
- TMPDIR 環境変数, 299, 550, 4603
- tmpdir システム変数, 799
- tmp_table_size システム変数, 799
- to-last-log オプション
 - mysqlbinlog, 535
- Tomcat, 5351
- TotalSendBufferMemory
 - API および SQL ノード, 3652
 - 管理ノード, 3574
 - データノード, 3644
- Touches()
 - 削除された機能, 40
- Townsend Alliance Key Manager
 - keyring_okv キーリングプラグイン, 1240
- TO_BASE64(), 1911
- TO_DAYS(), 1897
- TO_SECONDS(), 1897
- TPS, 5351
- TP_THREAD_GROUP_STATE
 - INFORMATION_SCHEMA テーブル, 4231
- tp_thread_group_state テーブル
 - performance_schema, 4364
- TP_THREAD_GROUP_STATS
 - INFORMATION_SCHEMA テーブル, 4232
- tp_thread_group_stats テーブル
 - performance_schema, 4366
- TP_THREAD_STATE
 - INFORMATION_SCHEMA テーブル, 4232
- tp_thread_state テーブル
 - performance_schema, 4368
- trace DBI メソッド, 1019
- TRADITIONAL SQL モード, 855, 861
- transaction-isolation オプション
 - mysqld, 666

- transaction-read-only オプション
 - mysqld, 666
- TransactionBufferMemory, 3587
- TransactionDeadlockDetectionTimeout, 3612
- TransactionInactiveTimeout, 3612
- TransactionMemory, 3589
- transactions, 2700
 - 分離レベル, 2705
 - メタデータのロック, 1597
 - レプリケーション, 3229
- transaction_alloc_block_size システム変数, 800
- transaction_allow_batching セッション変数 (NDB Cluster), 3686
- transaction_isolation システム変数, 800
- transaction_prealloc_size システム変数, 801
- transaction_read_only システム変数, 802
- transaction_write_set_extraction, 3137
- transporters
 - ndbinfo テーブル, 3974
- TRIGGER 権限, 1054
- TRIGGERS
 - INFORMATION_SCHEMA テーブル, 4185
- TRIM(), 1911
- TRUE, 1630, 1635
 - テスト, 1864, 1865
- true リテラル
 - JSON, 1811
- TRUNCATE TABLE, 2288
 - NDB Cluster, 3484
 - performance_schema データベース, 4277, 4443
 - およびレプリケーション, 3234
- TRUNCATE(), 1882
- tupscan オプション
 - ndb_select_all, 3816
- twiddle オプション
 - ndb_redo_log_reader, 3788
- TwoPassInitialNodeRestartCopy, 3627
- tx_isolation
 - 削除された機能, 39
- tx_read_only
 - 削除された機能, 39
- types
 - numeric, 1762
 - string, 1781
 - 日付と時刻, 1769
- TZ 環境変数, 177, 550, 880, 4605
- tz-utc オプション
 - mysqldump, 432
 - mysqlpump, 462

U

- UCASE(), 1912
- UCS-2, 1686
- ucs2 文字セット, 1721
 - クライアント文字セットとして, 1702
- UDF API, 1007
- UDF, 1007, 2536, 2537
 - アンインストール, 1008
 - インストール, 1008
- UDF インストール

- keyring, 1256
- UDF のアンインストール, 1008
- UDF のインストール, 1008
- UDF のキー設定
 - keyring_key_fetch(), 1260
 - keyring_key_generate(), 1261
 - keyring_key_length_fetch(), 1261
 - keyring_key_remove(), 1261
 - keyring_key_store(), 1262
 - keyring_key_type_fetch(), 1262
 - アンインストール, 1256
 - インストール, 1256
 - 使用, 1257
 - 汎用, 1256
 - プラグイン固有, 1262
- UDF を読み取る audit_log_read() 監査ログ, 1307, 1332
- UDF を読み取る audit_log_read_bookmark() 監査ログ, 1307, 1334
- UDFs
 - reference, 1852
- uid オプション
 - mysql_ssl_rsa_setup, 353
- ulimit, 4597
- UMASK 環境変数, 550, 4598
- UMASK_DIR 環境変数, 550, 4598
- unbuffered オプション
 - mysql, 379
- UNCOMPRESS(), 1990
- UNCOMPRESSED_LENGTH(), 1990
- Undo, 5352
- undo テーブルスペース, 2682, 2685
- Undo テーブルスペース, 5352
- undo ログ, 2682, 2685
- Undo ログ, 5352
- undo ログセグメント, 5352
- UndoDataBuffer, 3616
- UndoIndexBuffer, 3615
- UNHEX(), 1912
- Unicode, 5352
- Unicode Collation Algorithm, 1726
- Unicode, 1686
- Unicode 文字 (U)), 2070
- UNINSTALL COMPONENT ステートメント, 2540
- UNINSTALL PLUGIN ステートメント, 2540
- UNION, 291, 2339
 - カッコで囲まれたクエリー式, 2342
 - 非推奨となった機能, 36
- UNIQUE, 2175
- unique_checks システム変数, 803
- unique_subquery 結合タイプ
 - オブティマイザ, 1544
- Unix シグナル処理, 552
- UNIX_TIMESTAMP(), 1898
- UNKNOWN
 - テスト, 1864, 1865
- UNLOCK INSTANCE, 2381
- UNLOCK TABLES, 2382
- unpack オプション
 - myisamchk, 500
- UNSIGNED, 1762, 1767
- UNTIL, 2449

updatable_views_with_limit システム変数, 803
UPDATE, 70, 2360
update
 スレッドの状態, 1620
UPDATE 権限, 1054
update-state オプション
 myisamchk, 499
UpdateXML(), 1969
updating main table
 スレッドの状態, 1620
updating reference tables
 スレッドの状態, 1621
Updating
 スレッドの状態, 1620
upgrade-system-tables オプション
 mysql_upgrade, 362
UPPER(), 1912
Uptime_since_flush_status ステータス変数, 854
URI のような接続文字列, 322
usage オプション (NDB Cluster プログラム), 3836
usage オプション
 ndbxfm, 3834
 ndb_config, 3751
USAGE 権限, 1054
USE INDEX, 1579
USE KEY, 1579
USE, 2615
use コマンド
 mysql, 384
use-default オプション
 mysql_secure_installation, 350
use-frm オプション
 mysqlcheck, 413
use-http オプション
 ndb_setup.py, 3821, 3821
use-threads オプション
 mysqlimport, 447
useHexFormat オプション
 ndb_select_all, 3816
User lock
 スレッドの状態, 1621
User sleep
 スレッドの状態, 1621
user table
 account_locked カラム, 1066
 システムテーブル, 230, 897, 1062
user
 root, 230
USER 環境変数, 321, 550
user テーブル
 ソート, 1075
USER(), 2003
users
 削除, 1080, 2501
users テーブル
 performance_schema, 4324
USER_ATTRIBUTES
 INFORMATION_SCHEMA テーブル, 4187
user_defined_functions テーブル
 performance_schema, 1009, 4414

USER_PRIVILEGES
 INFORMATION_SCHEMA テーブル, 4188
user_summary ビュー
 sys スキーマ, 4485
user_summary_by_file_io ビュー
 sys スキーマ, 4486
user_summary_by_file_io_type ビュー
 sys スキーマ, 4486
user_summary_by_stages ビュー
 sys スキーマ, 4486
user_summary_by_statement_latency ビュー
 sys スキーマ, 4487
user_summary_by_statement_type ビュー
 sys スキーマ, 4488
user_variables_by_thread テーブル
 performance_schema, 4329
UseShm, 3605
use_invisible_indexes フラグ
 optimizer_switch システム変数, 1510
USE_LD_GOLD オプション
 CMake, 208
USE_LD_LLD オプション
 CMake, 209
use_secondary_engine システム変数, 804
USING HASH
 NDB テーブル, 2205
USING と ON
 結合, 2338
UTC_DATE(), 1899
UTC_TIME(), 1899
UTC_TIMESTAMP(), 1899
UTF-8, 1686
 データベースオブジェクトメタデータ, 1691
utf16 文字セット, 1721
 クライアント文字セットとして, 1702
utf16le 文字セット, 1721
 クライアント文字セットとして, 1702
utf16_bin 照合順序, 1731
utf32 文字セット, 1721
 クライアント文字セットとして, 1702
utf8 文字セット, 1720
 utf8mb3 のエイリアス, 1720, 1720
utf8mb3
 非推奨となった機能, 35
utf8mb3 文字セット, 1720
 utf8 エイリアス, 1720, 1720
utf8mb4 照合, 1727
utf8mb4 文字セット, 1719
utf8mb4_0900_bin
 対 utf8mb4_bin, 1725
utf8mb4_bin
 対 utf8mb4_0900_bin, 1725
UUID(), 2139
UUID_SHORT(), 2140
UUID_TO_BIN(), 2140

V

valid
 GIS 値, 1803

- 空間値, 1803
- validate-config オプション
 - mysqld, 669
- validate-password オプション
 - mysqld, 1227
- validate_password コンポーネント, 1221
 - アンインストール, 1223
 - インストール, 1222
 - システム変数, 1223
 - ステータス変数, 1227
- validate_password プラグイン, 1221
 - options, 1227
 - validate_password コンポーネントへの移行, 1231
 - 構成, 1223
 - システム変数, 1228
 - ステータス変数, 1230
 - 非推奨となった機能, 36
- validate_password.check_user_name システム変数, 1224
- validate_password.dictionary_file システム変数, 1224
- validate_password.dictionary_file_last_parse ステータス変数, 1227
- validate_password.dictionary_file_words_count ステータス変数, 1227
- validate_password.length システム変数, 1225
- validate_password.mixed_case_count システム変数, 1225
- validate_password.number_count システム変数, 1226
- validate_password.policy システム変数, 1226
- validate_password.special_char_count システム変数, 1227
- validate_password_check_user_name システム変数, 1228
- validate_password_dictionary_file システム変数, 1228
- validate_password_dictionary_file_last_parsed ステータス変数, 1230
- validate_password_dictionary_file_words_count ステータス変数, 1230
- validate_password_length システム変数, 1229
- validate_password_mixed_case_count システム変数, 1229
- validate_password_number_count システム変数, 1229
- validate_password_policy システム変数, 1230
- validate_password_special_char_count システム変数, 1230
- VALIDATE_PASSWORD_STRENGTH(), 1990
- validate_user_plugins システム変数, 804
- VALUES ステートメント, 2364
 - INTO を使用, 2332
 - 新機能, 30
- VALUES(), 2141
 - 非推奨となった機能, 37
- VARBINARY データ型, 1783, 1786
- VARCHAR
 - サイズ, 1829
- VARCHAR データ型, 1781, 1783
- VARCHARACTER データ型, 1783
- VARIANCE(), 2098
- VAR_POP(), 2098
- VAR_SAMP(), 2098
- verbose オプション
 - innochecksum, 487
 - myisamchk, 496
 - myisampack, 510
 - myisam_ftdump, 492
 - mysql, 379
 - mysqldadmin, 403
 - mysqlbinlog, 535
 - mysqlcheck, 413
 - mysqld, 669

mysqldump, 427
mysqldumpslow, 545
mysqld_multi, 344
mysqlimport, 447
mysqlshow, 472
mysqlslap, 482
mysql_config_editor, 518
mysql_ssl_rsa_setup, 353
mysql_upgrade, 362
my_print_defaults, 548
nbd, 3730
nbdmtd, 3730
nbd_blob_tool, 3744
nbd_import, 3774
nbd_index_stat, 3779
nbd_mgmd, 3740
nbd_move_data, 3781
nbd_perror, 3783
nbd_restore, 3809
perror, 549
verify-binlog-checksum オプション
 mysqlbinlog, 536
version オプション (NDB Cluster プログラム), 3838
version システム変数, 805
VERSION ファイル
 CMake, 219
VERSION(), 2003
version-check オプション
 mysql_upgrade, 362
version_comment システム変数, 805
version_compile_machine システム変数, 805
version_compile_os システム変数, 805
version_compile_zlib システム変数, 805
version_major() 関数
 sys スキーマ, 4518
version_minor() 関数
 sys スキーマ, 4519
version_patch() 関数
 sys スキーマ, 4519
version_tokens_delete() バージョントークン UDF, 975
version_tokens_edit() バージョントークン UDF, 975
version_tokens_lock_exclusive() バージョントークン UDF, 976
version_tokens_lock_shared() バージョントークン UDF, 976
version_tokens_session システム変数, 977
version_tokens_session_number システム変数, 978
version_tokens_set() バージョントークン UDF, 976
version_tokens_show() バージョントークン UDF, 976
version_tokens_unlock() バージョントークン UDF, 976
VERSION_TOKEN_ADMIN 権限, 1059
view, 5352
VIEWS
 INFORMATION_SCHEMA テーブル, 4189
VIEW_ROUTINE_USAGE
 INFORMATION_SCHEMA テーブル, 4190
view_routine_usage テーブル
 データディクショナリテーブル, 896
VIEW_TABLE_USAGE
 INFORMATION_SCHEMA テーブル, 4191
view_table_usage テーブル
 データディクショナリテーブル, 896

Visual Studio, 5352

W

WAIT COMPLETED (START BACKUP コマンド), 3885

WAIT STARTED (START BACKUP コマンド), 3885

wait オプション

 mysamchk, 496

 mysampack, 510

 mysql, 379

 mysqladmin, 404

wait-nodes オプション

 ndb_waiter, 3831

Waiting for allowed to take ndbcluster global schema lock

 スレッドの状態, 1625

Waiting for commit lock

 スレッドの状態, 1621

Waiting for event from ndbcluster

 スレッドの状態, 1625

Waiting for event metadata lock

 スレッドの状態, 1621

Waiting for event read lock

 スレッドの状態, 1621

Waiting for first event from ndbcluster

 スレッドの状態, 1625

Waiting for master to send event

 スレッドの状態, 1623

Waiting for master update

 スレッドの状態, 1623

Waiting for ndbcluster binlog update to reach current position

 スレッドの状態, 1625

Waiting for ndbcluster global schema lock

 スレッドの状態, 1625

Waiting for ndbcluster to start

 スレッドの状態, 1625

Waiting for next activation

 スレッドの状態, 1625

Waiting for scheduler to stop

 スレッドの状態, 1625

Waiting for schema epoch

 スレッドの状態, 1625

Waiting for schema metadata lock

 スレッドの状態, 1621

Waiting for slave mutex on exit

 スレッドの状態, 1623, 1624

Waiting for stored function metadata lock

 スレッドの状態, 1621

Waiting for stored procedure metadata lock

 スレッドの状態, 1621

Waiting for table level lock

 スレッドの状態, 1621

Waiting for table metadata lock

 スレッドの状態, 1621

Waiting for the next event in relay log

 スレッドの状態, 1624

Waiting for the slave SQL thread to free enough relay log space

 スレッドの状態, 1623

Waiting for trigger metadata lock

 スレッドの状態, 1621

Waiting on cond

- スレッドの状態, 1622
- Waiting on empty queue
 - スレッドの状態, 1625
- Waiting to finalize termination
 - スレッドの状態, 1622
- Waiting to reconnect after a failed binlog dump request
 - スレッドの状態, 1623
- Waiting to reconnect after a failed master event read
 - スレッドの状態, 1623
- Waiting until MASTER_DELAY seconds after master executed event
 - スレッドの状態, 1624
- waits_by_host_by_latency ビュー
 - sys スキーマ, 4490
- waits_by_user_by_latency ビュー
 - sys スキーマ, 4490
- waits_global_by_latency ビュー
 - sys スキーマ, 4491
- wait_classes_global_by_avg_latency ビュー
 - sys スキーマ, 4489
- wait_classes_global_by_latency ビュー
 - sys スキーマ, 4489
- WAIT_FOR_EXECUTED_GTID_SET(), 2089
- wait_timeout システム変数, 806
- WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS(), 2090
- Wan, 3573, 3654
- warnings コマンド
 - mysql, 384
- warning_count システム変数, 806
- watch-progress オプション
 - mysqldump, 463
- WatchdogImmediateKill, 3620
- websites
 - MySQL, 61
- WEEK(), 1899
- WEEKDAY(), 1900
- WEEKOFYEAR(), 1901
- WEIGHT_STRING(), 1913
- well-formed
 - GIS 値, 1803
 - 空間値, 1803
- WHERE, 1433
 - SHOW を伴う, 4132, 4234
- where オプション
 - mysqldump, 434
- WHERE を伴う SHOW, 4132, 4234
- WHILE, 2450
 - ラベル, 2444
- windowing_use_high_precision システム変数, 806
- Windows
 - MySQL の制限事項, 137
 - アップグレード, 257
 - インタラクティブ履歴, 390
 - パス名区切り文字, 305
 - プラグابل認証, 1171
- WIN_DEBUG_NO_INLINE オプション
 - CMake, 209
- WITH ROLLUP, 2098
- Within()
 - 削除された機能, 40
- WITH_ANT オプション

CMake, 209
WITH_ASAN オプション
CMake, 209
WITH_ASAN_SCOPE オプション
CMake, 209
WITH_AUTHENTICATION_LDAP オプション
CMake, 209
WITH_AUTHENTICATION_PAM オプション
CMake, 209
WITH_AWS_SDK オプション
CMake, 209
WITH_BOOST オプション
CMake, 209
WITH_BUNDLED_LIBEVENT オプション
CMake, 217
WITH_BUNDLED_MEMCACHED オプション
CMake, 217
WITH_CLASSPATH オプション
CMake, 217
WITH_CLIENT_PROTOCOL_TRACING オプション
CMake, 210
WITH_CURL オプション
CMake, 210
WITH_DEBUG オプション
CMake, 210
WITH_DEFAULT_COMPILER_OPTIONS オプション
CMake, 216
WITH_DEFAULT_FEATURE_SET オプション
CMake, 211
WITH_EDITLINE オプション
CMake, 211
WITH_ERROR_INSERT オプション
CMake, 217
WITH_GMOCK オプション
CMake, 211
WITH_ICU オプション
CMake, 211
WITH_INNODB_EXTRA_DEBUG オプション
CMake, 211
WITH_INNODB_MEMCACHED オプション
CMake, 211
WITH_JEMALLOC オプション
CMake, 211
WITH_KEYRING_TEST オプション
CMake, 211
WITH_LIBEVENT オプション
CMake, 211
WITH_LIBWRAP オプション
CMake, 212
WITH_LOCK_ORDER オプション
CMake, 212
WITH_LSAN オプション
CMake, 212
WITH_LTO オプション
CMake, 212
WITH_LZ4 オプション
CMake, 212
WITH_LZMA オプション
CMake, 212
WITH_MECAB オプション

CMake, 212
WITH_MSAN オプション
CMake, 213
WITH_MSCRT_DEBUG オプション
CMake, 213
WITH_MYSQLX オプション
CMake, 213
WITH_NDBCLUSTER オプション
CMake, 217
WITH_NDBCLUSTER_STORAGE_ENGINE オプション
CMake, 217
WITH_NDBMTD オプション
CMake, 217
WITH_NDB_BINLOG オプション
CMake, 218
WITH_NDB_DEBUG オプション
CMake, 218
WITH_NDB_JAVA オプション
CMake, 218
WITH_NDB_PORT オプション
CMake, 218
WITH_NDB_TEST オプション
CMake, 218
WITH_NUMA オプション
CMake, 213
WITH_PLUGIN_NDBCLUSTER オプション
CMake, 218
WITH_PROTOBUF オプション
CMake, 213
WITH_RAPID オプション
CMake, 213
WITH_RAPIDJSON オプション
CMake, 213
WITH_RE2 オプション
CMake, 213
WITH_ROUTER オプション
CMake, 214
WITH_SSL オプション
CMake, 214
WITH_SYSTEMD オプション
CMake, 214
WITH_SYSTEMD_DEBUG オプション
CMake, 214
WITH_SYSTEM_LIBS オプション
CMake, 214
WITH_TCMALLOC オプション
CMake, 215
WITH_TEST_TRACE_PLUGIN オプション
CMake, 215
WITH_TSAN オプション
CMake, 215
WITH_UBSAN オプション
CMake, 215
WITH_UNIT_TESTS オプション
CMake, 215
WITH_UNIXODBC オプション
CMake, 215
WITH_VALGRIND オプション
CMake, 215
WITH_ZLIB オプション

CMake, 216
WITH_ZSTD オプション
CMake, 216
WKB 形式
ジオメトリ値, 1802
WKT 形式
ジオメトリ値, 1801
wolfSSL, 1130
write オプション
innochecksum, 488
write-binlog オプション
mysqlcheck, 413
mysql_upgrade, 362
write_buffer_size myisamchk 変数, 497
Writing to net
スレッドの状態, 1622

X

X プラグイン, 3413
X プラグイン オプション
mysqlx, 3421
x\$ ビュー
sys スキーマ, 4453
x\$host_summary ビュー
sys スキーマ, 4454
x\$host_summary_by_file_io ビュー
sys スキーマ, 4455
x\$host_summary_by_file_io_type ビュー
sys スキーマ, 4455
x\$host_summary_by_stages ビュー
sys スキーマ, 4455
x\$host_summary_by_statement_latency ビュー
sys スキーマ, 4456
x\$host_summary_by_statement_type ビュー
sys スキーマ, 4456
x\$innodb_buffer_stats_by_schema ビュー
sys スキーマ, 4457
x\$innodb_buffer_stats_by_table ビュー
sys スキーマ, 4458
x\$innodb_lock_waits ビュー
sys スキーマ, 4459
x\$io_by_thread_by_latency ビュー
sys スキーマ, 4460
x\$io_global_by_file_by_bytes ビュー
sys スキーマ, 4461
x\$io_global_by_file_by_latency ビュー
sys スキーマ, 4462
x\$io_global_by_wait_by_bytes ビュー
sys スキーマ, 4462
x\$io_global_by_wait_by_latency ビュー
sys スキーマ, 4463
x\$latest_file_io ビュー
sys スキーマ, 4464
x\$memory_by_host_by_current_bytes ビュー
sys スキーマ, 4465
x\$memory_by_thread_by_current_bytes ビュー
sys スキーマ, 4465
x\$memory_by_user_by_current_bytes ビュー
sys スキーマ, 4466

x\$memory_global_by_current_bytes ビュー
sys スキーマ, 4466

x\$memory_global_total ビュー
sys スキーマ, 4467

x\$processlist ビュー
sys スキーマ, 4468

x\$schema_flattened_keys ビュー
sys スキーマ, 4472

x\$schema_index_statistics ビュー
sys スキーマ, 4471

x\$schema_tables_with_full_table_scans ビュー
sys スキーマ, 4478

x\$schema_table_lock_waits ビュー
sys スキーマ, 4473

x\$schema_table_statistics ビュー
sys スキーマ, 4475

x\$schema_table_statistics_with_buffer ビュー
sys スキーマ, 4476

x\$session ビュー
sys スキーマ, 4478

x\$statements_with_errors_or_warnings ビュー
sys スキーマ, 4480

x\$statements_with_full_table_scans ビュー
sys スキーマ, 4481

x\$statements_with_runtimes_in_95th_percentile ビュー
sys スキーマ, 4482

x\$statements_with_sorting ビュー
sys スキーマ, 4483

x\$statements_with_temp_tables ビュー
sys スキーマ, 4484

x\$statement_analysis ビュー
sys スキーマ, 4479

x\$user_summary ビュー
sys スキーマ, 4485

x\$user_summary_by_file_io ビュー
sys スキーマ, 4486

x\$user_summary_by_file_io_type ビュー
sys スキーマ, 4486

x\$user_summary_by_stages ビュー
sys スキーマ, 4486

x\$user_summary_by_statement_latency ビュー
sys スキーマ, 4487

x\$user_summary_by_statement_type ビュー
sys スキーマ, 4488

x\$waits_by_host_by_latency ビュー
sys スキーマ, 4490

x\$waits_by_user_by_latency ビュー
sys スキーマ, 4490

x\$waits_global_by_latency ビュー
sys スキーマ, 4491

x\$wait_classes_global_by_avg_latency ビュー
sys スキーマ, 4489

x\$wait_classes_global_by_latency ビュー
sys スキーマ, 4489

X()
削除された機能, 40

X.509/Certificate, 1130

XA, 5353

XA BEGIN, 2391

XA COMMIT, 2391

XA PREPARE, 2391
XA RECOVER, 2391
XA ROLLBACK, 2391
XA START, 2391
XA トランザクション, 2390
 制約, 2393
 トランザクション識別子, 2391
XA_RECOVER_ADMIN 権限, 1059
xid
 XA トランザクション識別子, 2391
xml オプション
 mysql, 379
 mysqldump, 432
 ndb_config, 3751
XOR
 logical, 1867
 ビット単位, 1984

Y

Y()
 削除された機能, 40
YEAR データ型, 1772, 1775
YEAR(), 1901
YEARWEEK(), 1901

Z

ZEROFILL, 1762, 1767
zlib_decompress, 299, 549
zstd-compression-level オプション, 319
 mysql, 379
 mysqladmin, 404
 mysqlbinlog, 536
 mysqlcheck, 413
 mysqldump, 424
 mysqlimport, 447
 mysqlpump, 463
 mysqlshow, 472
 mysqslap, 483
 mysql_upgrade, 362

C 関数の索引

mysql_affected_rows()

[セクション13.2.1 「CALL ステートメント」](#)
[セクション13.2.6 「INSERT ステートメント」](#)
[セクション13.2.9 「REPLACE ステートメント」](#)
[セクション12.16 「情報関数」](#)

mysql_bind_param()

[セクション9.6 「クエリー属性」](#)

mysql_change_user()

[セクション4.5.1.2 「mysql クライアントコマンド」](#)

mysql_close()

[セクションB.3.2.9 「通信エラーおよび中止された接続」](#)

mysql_errno()

セクション13.6.7.5「[SIGNAL ステートメント](#)」
セクションB.2「[エラー情報インタフェース](#)」
セクション6.4.5.4「[監査ログファイル形式](#)」

mysql_error()

セクション13.6.7.5「[SIGNAL ステートメント](#)」
セクションB.2「[エラー情報インタフェース](#)」

mysql_escape_string()

セクション6.1.7「[クライアントプログラミングのセキュリティーガイドライン](#)」

mysql_fetch_row()

セクション16.8.1「[FEDERATED ストレージエンジンの概要](#)」

mysql_free_result()

セクションB.3.2.12「[コマンドは同期されていません](#)」

mysql_get_character_set_info()

セクション10.14.2「[照合順序 ID の選択](#)」

mysql_info()

セクション13.1.9「[ALTER TABLE ステートメント](#)」
セクション13.2.6「[INSERT ステートメント](#)」
セクション13.2.7「[LOAD DATA ステートメント](#)」
セクション1.7.3.1「[PRIMARY KEY および UNIQUE インデックス制約](#)」
セクション13.2.13「[UPDATE ステートメント](#)」

mysql_insert_id()

セクション3.6.9「[AUTO_INCREMENT の使用](#)」
セクション13.1.20「[CREATE TABLE ステートメント](#)」
セクション13.2.6「[INSERT ステートメント](#)」
セクション5.1.8「[サーバーシステム変数](#)」
セクション12.16「[情報関数](#)」

mysql_next_result()

セクション13.2.1「[CALL ステートメント](#)」

mysql_options()

セクション6.1.6「[LOAD DATA LOCAL のセキュリティー上の考慮事項](#)」
セクション2.11.4「[MySQL 8.0 での変更](#)」
セクション1.3「[MySQL 8.0 の新機能](#)」
セクションB.3.2.7「[MySQL サーバーが存在しなくなりました](#)」
セクション6.4.1.2「[SHA-2 プラガブル認証のキャッシュ](#)」
セクション6.4.1.3「[SHA-256 プラガブル認証](#)」
セクション6.2.1「[アカウントのユーザー名とパスワード](#)」
セクション6.4.1.4「[クライアント側クリアテキストプラガブル認証](#)」
セクション27.12.9「[パフォーマンススキーマ接続属性テーブル](#)」
セクション6.2.17「[プラガブル認証](#)」
セクション4.2.8「[接続圧縮制御](#)」
セクション10.4「[接続文字セットおよび照合順序](#)」
セクション6.2.16「[期限切れパスワードのサーバー処理](#)」
セクション5.8.4「[複数サーバー環境でのクライアントプログラムの使用](#)」

mysql_options4()

セクション27.12.9「[パフォーマンススキーマ接続属性テーブル](#)」

mysql_ping()

セクションB.3.2.7「MySQL サーバーが存在しなくなりました」

mysql_query()

セクション13.2.1「CALL ステートメント」

mysql_real_connect()

セクション13.2.1「CALL ステートメント」

セクション4.2.6「DNS SRV レコードを使用したサーバーへの接続」

セクション13.2.6.2「INSERT ... ON DUPLICATE KEY UPDATE ステートメント」

セクション13.2.6「INSERT ステートメント」

セクション4.10「MySQL での Unix シグナル処理」

セクション4.5.1.1「mysql クライアントオプション」

セクション5.1.8「サーバーシステム変数」

セクション25.2.1「ストアードルーチンの構文」

セクション13.5「プリヘアドステートメント」

セクション12.16「情報関数」

セクション6.2.16「期限切れパスワードのサーバー処理」

セクション5.8.4「複数サーバー環境でのクライアントプログラムの使用」

第12章「関数と演算子」

mysql_real_connect_dns_srv()

セクション4.2.6「DNS SRV レコードを使用したサーバーへの接続」

セクション4.5.1.1「mysql クライアントオプション」

mysql_real_escape_string_quote()

セクション6.1.7「クライアントプログラミングのセキュリティーガイドライン」

セクション9.1.1「文字列リテラル」

セクション11.4.7「空間カラムへのデータ移入」

mysql_real_query()

セクション13.2.1「CALL ステートメント」

セクション16.8.1「FEDERATED ストレージエンジンの概要」

mysql_session_track_get_first()

セクション5.1.18「クライアントセッション状態の変更のサーバートラッキング」

mysql_session_track_get_next()

セクション5.1.18「クライアントセッション状態の変更のサーバートラッキング」

mysql_shutdown()

セクション6.2.2「MySQL で提供される権限」

セクション13.7.8.9「SHUTDOWN ステートメント」

mysql_sqlstate()

セクション13.6.7.5「SIGNAL ステートメント」

セクションB.2「エラー情報インタフェース」

mysql_stmt_attr_set()

セクション13.6.6.5「サーバー側のカーソルの制約」

mysql_stmt_close()

セクション27.12.6.4「prepared_statements_instances テーブル」

mysql_stmt_errno()

セクションB.2「エラー情報インタフェース」

mysql_stmt_error()

[セクションB.2「エラー情報インタフェース」](#)

mysql_stmt_execute()

[セクション27.12.6.4「prepared_statements_instances テーブル」](#)

mysql_stmt_next_result()

[セクション13.2.1「CALL ステートメント」](#)

mysql_stmt_prepare()

[セクション27.12.6.4「prepared_statements_instances テーブル」](#)

[セクション13.5「プリペアドステートメント」](#)

[セクション8.10.3「プリペアドステートメントおよびストアプログラムのキャッシュ」](#)

mysql_stmt_send_long_data()

[セクション5.1.8「サーバーシステム変数」](#)

mysql_stmt_sqlstate()

[セクションB.2「エラー情報インタフェース」](#)

mysql_store_result()

[セクション16.8.1「FEDERATED ストレージエンジンの概要」](#)

[セクション4.5.1「mysql — MySQL コマンドラインクライアント」](#)

[セクションB.3.2.12「コマンドは同期されていません」](#)

mysql_use_result()

[セクション4.5.1「mysql — MySQL コマンドラインクライアント」](#)

[セクションB.3.2.12「コマンドは同期されていません」](#)

[セクションB.3.2.6「メモリー不足」](#)

mysql_warning_count()

[セクション13.7.7.42「SHOW WARNINGS ステートメント」](#)

[セクションB.2「エラー情報インタフェース」](#)

コマンドの索引

記号 | [A](#) | [B](#) | [C](#) | [D](#) | [F](#) | [G](#) | [H](#) | [I](#) | [J](#) | [K](#) | [L](#) | [M](#) | [N](#) | [O](#) | [P](#) | [R](#) | [S](#) | [T](#) | [U](#) | [V](#) | [W](#) | [X](#) | [Y](#) | [Z](#)

記号

[\[index top\]](#)

このスタイルのテキスト

[セクション1.1「このマニュアルについて」](#)

サービス管理マネージャー

[セクション2.3「Microsoft Windows に MySQL をインストールする」](#)

ディレクトリユーティリティー

[セクション2.4.1「macOS への MySQL のインストールに関する一般的なノート」](#)

A

[\[index top\]](#)

Access

[セクション13.2.2 「DELETE ステートメント」](#)

addgroup

[セクション23.2.1.1 「Linux への NDB Cluster バイナリリリースのインストール」](#)
[セクション2.2 「一般的なバイナリを使用した MySQL の Unix/Linux へのインストール」](#)

addr2line

[セクション5.9.1.5 「スタックトレースの使用」](#)

adduser

[セクション23.2.1.1 「Linux への NDB Cluster バイナリリリースのインストール」](#)
[セクション2.2 「一般的なバイナリを使用した MySQL の Unix/Linux へのインストール」](#)

ant

[セクション2.9.7 「MySQL ソース構成オプション」](#)

APF

[セクション23.5.17.1 「NDB Cluster のセキュリティーおよびネットワークの問題」](#)

apt-get

[セクション15.20.5 「InnoDB memcached プラグインのセキュリティーに関する考慮事項」](#)
[セクション2.5.5 「オラクルからの Debian パッケージを使用して MySQL を Linux にインストールする」](#)
[セクション2.5.7 「ネイティブソフトウェアリポジトリから MySQL を Linux にインストールする」](#)

audit2allow

[セクション6.7.6 「SELinux のトラブルシューティング」](#)

B

[\[index top\]](#)

bash

[セクション2.4.1 「macOS への MySQL のインストールに関する一般的なノート」](#)
[セクション4.2.1 「MySQL プログラムの起動」](#)
[セクション17.1.6.3 「Replica Server のオプションと変数」](#)
[セクション13.7.8.8 「RESTART ステートメント」](#)
[セクション1.1 「このマニュアルについて」](#)
[セクション6.1.2.1 「パスワードセキュリティーのためのエンドユーザーガイドライン」](#)
[セクション4.2.9 「環境変数の設定」](#)

bison

[セクション2.9.8 「MySQL のコンパイルに関する問題」](#)
[セクション1.8.1 「MySQL への貢献者」](#)
[セクション2.9.2 「ソースインストールの前提条件」](#)

C

[\[index top\]](#)

cat

[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)
[セクション4.5.1.1 「mysql クライアントオプション」](#)

cd

[root のパスワードのリセット: Windows システム](#)

chkconfig

セクション23.2.1.1「Linux への NDB Cluster バイナリリリースのインストール」

セクション4.3.3「mysql.server — MySQL サーバー起動スクリプト」

CMake

セクション16.5「ARCHIVE ストレージエンジン」

セクション16.6「BLACKHOLE ストレージエンジン」

セクション16.9「EXAMPLE ストレージエンジン」

セクション16.8「FEDERATED ストレージエンジン」

セクション15.14「InnoDB の起動オプションおよびシステム変数」

セクション23.2.1.4「Linux でのソースからの NDB Cluster の構築」

セクション6.1.6「LOAD DATA LOCAL のセキュリティー上の考慮事項」

セクション5.9.3「LOCK_ORDER ツール」

セクション1.3「MySQL 8.0 の新機能」

第23章「MySQL NDB Cluster 8.0」

セクションB.3.3.3「MySQL が繰り返しクラッシュする場合の対処方法」

セクションB.3.3.6「MySQL の UNIX ソケットファイルを保護または変更する方法」

セクション2.9.8「MySQL のコンパイルに関する問題」

セクション1.2.2「MySQL の主な機能」

セクション2.9.7「MySQL ソース構成オプション」

セクション23.5.9「NDB Cluster での MySQL Server の使用」

セクション27.12.19.9「processlist テーブル」

セクション2.9.6「SSL ライブラリサポートの構成」

セクション2.5.9「systemd を使用した MySQL Server の管理」

セクション5.8.3「Unix 上での複数の MySQL インスタンスの実行」

セクション23.2.2.2「Windows でのソースからの NDB Cluster のコンパイルとインストール」

セクション10.5「アプリケーションの文字セットおよび照合順序の構成」

セクション4.2.2.2「オプションファイルの使用」

セクション6.4.4.13「キーリングシステム変数」

セクション5.1.7「サーバーコマンドオプション」

セクション5.1.8「サーバーシステム変数」

セクション10.3.2「サーバー文字セットおよび照合順序」

セクション2.9.2「ソースインストールの前提条件」

セクション14.1「データディクショナリスキーマ」

セクション27.2「パフォーマンススキーマビルド構成」

セクション10.13「文字セットの追加」

セクションB.3.2.15「文字セットを初期化できません」

セクション2.9.4「標準ソース配布を使用して MySQL をインストールする」

セクション4.9「環境変数」

セクション2.9.5「開発ソースツリーを使用して MySQL をインストールする」

cmake

セクション15.20.5「InnoDB memcached プラグインのセキュリティーに関する考慮事項」

セクション2.9.10「MySQL Doxygen ドキュメントコンテンツの生成」

セクション2.9.7「MySQL ソース構成オプション」

セクション2.9.4「標準ソース配布を使用して MySQL をインストールする」

cmd

root のパスワードのリセット: Windows システム

cmd.exe

セクション4.6.2「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティー」

セクション4.2.1「MySQL プログラムの起動」

セクション1.1「このマニュアルについて」

command.com

セクション4.2.1「MySQL プログラムの起動」

[セクション1.1「このマニュアルについて」](#)

comp_err

[セクション4.4.1「comp_err — MySQL エラーメッセージファイルのコンパイル」](#)

[セクション4.1「MySQL プログラムの概要」](#)

configure

[セクション23.4.32「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#)

[セクション1.1「このマニュアルについて」](#)

[セクション1.6「質問またはバグをレポートする方法」](#)

copy

[ローデータファイルを使用したデータスナップショットの作成](#)

coreadm

[セクション2.7「Solaris への MySQL のインストール」](#)

[セクション5.1.7「サーバーコマンドオプション」](#)

cp

[セクション7.1「バックアップとリカバリの種類」](#)

[セクション17.4.1.2「レプリカからの RAW データのバックアップ」](#)

[セクション17.1.2.8「レプリケーション環境へのレプリカの追加」](#)

[ローデータファイルを使用したデータスナップショットの作成](#)

cron

[セクションB.3.2.2「\[ローカルの\] MySQL サーバーに接続できません」](#)

[セクション13.7.3.2「CHECK TABLE ステートメント」](#)

[セクション7.6.5「MyISAM テーブル保守スケジュールのセットアップ」](#)

[セクション16.2.1「MyISAM 起動オプション」](#)

[セクション5.4.6「サーバーログの保守」](#)

[セクション3.5「バッチモードでの MySQL の使用」](#)

csh

[セクション4.2.1「MySQL プログラムの起動」](#)

[セクション1.1「このマニュアルについて」](#)

[セクション4.2.9「環境変数の設定」](#)

curl

[セクション2.9.7「MySQL ソース構成オプション」](#)

D

[\[index top\]](#)

daemon_memcached

[セクション15.20.2「InnoDB memcached のアーキテクチャー」](#)

[セクション15.20.6.2「InnoDB memcached プラグインに対する memcached アプリケーションの適応」](#)

date

[セクション4.3.2「mysqld_safe — MySQL サーバー起動スクリプト」](#)

df

[セクションB.3.1「問題の原因を判別する方法」](#)

dig

[セクション1.3「MySQL 8.0 の新機能」](#)

dnf

セクション2.5.1 「MySQL Yum リポジトリを使用して MySQL を Linux にインストールする」
セクション2.11.7 「MySQL Yum リポジトリを使用する MySQL のアップグレード」
セクション2.5.4 「Oracle の RPM パッケージを使用した Linux への MySQL のインストール」
セクション23.2.1.2 「RPM から NDB Cluster をインストール」

dnf config-manager

セクション2.5.1 「MySQL Yum リポジトリを使用して MySQL を Linux にインストールする」

dnf upgrade

セクション2.5.1 「MySQL Yum リポジトリを使用して MySQL を Linux にインストールする」

docker exec

セクション2.5.6.1 「Docker を使用した MySQL Server デプロイメントの基本ステップ」

docker images

セクション2.5.6.1 「Docker を使用した MySQL Server デプロイメントの基本ステップ」

docker inspect

セクション2.5.6.2 「Docker での MySQL Server のデプロイに関するその他のトピック」

docker logs mysqlid-container

セクション2.5.6.2 「Docker での MySQL Server のデプロイに関するその他のトピック」

docker ps

セクション2.5.6.1 「Docker を使用した MySQL Server デプロイメントの基本ステップ」

docker pull

セクション2.5.6.2 「Docker での MySQL Server のデプロイに関するその他のトピック」
セクション2.5.6.1 「Docker を使用した MySQL Server デプロイメントの基本ステップ」

docker rm

セクション2.5.6.1 「Docker を使用した MySQL Server デプロイメントの基本ステップ」

docker run

セクション2.5.6.2 「Docker での MySQL Server のデプロイに関するその他のトピック」
セクション2.5.6.1 「Docker を使用した MySQL Server デプロイメントの基本ステップ」

docker stop

セクション2.5.6.1 「Docker を使用した MySQL Server デプロイメントの基本ステップ」

dot

セクション2.9.10 「MySQL Doxygen ドキュメントコンテンツの生成」

doxygen

セクション2.9.10 「MySQL Doxygen ドキュメントコンテンツの生成」

dpkg

セクション2.5.5 「オラクルからの Debian パッケージを使用して MySQL を Linux にインストールする」

dump

ローデータファイルを使用したデータスナップショットの作成

F

[\[index top\]](#)

flex

[セクション5.9.3 「LOCK_ORDER ツール」](#)
[セクション2.9.7 「MySQL ソース構成オプション」](#)

G

[\[index top\]](#)

gcc

[セクション1.8.4 「MySQL の作成に使用されたツール」](#)
[セクション2.13.3 「Perl DBI/DBD インタフェース使用の問題」](#)

gdb

[セクション5.9.1.4 「gdb での mysqld のデバッグ」](#)
[セクションB.3.3.3 「MySQL が繰り返しクラッシュする場合の対処方法」](#)
[セクション1.8.4 「MySQL の作成に使用されたツール」](#)
[セクション5.9.1.1 「デバッグのための MySQL のコンパイル」](#)

getcap

[セクション5.1.16 「リソースグループ」](#)

getenforce

[セクション6.7.2 「SELinux モードの変更」](#)

git branch

[セクション2.9.5 「開発ソースツリーを使用して MySQL をインストールする」](#)

git checkout

[セクション2.9.5 「開発ソースツリーを使用して MySQL をインストールする」](#)

gmake

[セクション2.8 「FreeBSD に MySQL をインストールする」](#)
[セクション2.9.2 「ソースインストールの前提条件」](#)
[セクション2.9.4 「標準ソース配布を使用して MySQL をインストールする」](#)

GnuPG

[セクション2.1.4.2 「GnuPG を使用した署名確認」](#)

gnutar

[セクション2.9.2 「ソースインストールの前提条件」](#)
[セクション2.2 「一般的なバイナリを使用した MySQL の Unix/Linux へのインストール」](#)

gogoc

[セクション5.1.13.5 「ブローカからの IPv6 アドレスの入手」](#)

gold

[セクション2.9.7 「MySQL ソース構成オプション」](#)

gpg

[セクション2.1.4.2 「GnuPG を使用した署名確認」](#)

grep

[セクション4.6.9 「mysqldumpslow — スロークエリーログファイルの要約」](#)

[セクション3.3.4.7 「パターンマッチング」](#)

groupadd

[セクション23.2.1.1 「Linux への NDB Cluster バイナリリリースのインストール」](#)

[セクション2.7 「Solaris への MySQL のインストール」](#)

[セクション2.2 「一般的なバイナリを使用した MySQL の Unix/Linux へのインストール」](#)

gtar

[セクション2.7 「Solaris への MySQL のインストール」](#)

[セクション2.9.2 「ソースインストールの前提条件」](#)

[セクション2.2 「一般的なバイナリを使用した MySQL の Unix/Linux へのインストール」](#)

gunzip

[セクション2.9.4 「標準ソース配布を使用して MySQL をインストールする」](#)

[セクション6.4.5.5 「監査ロギング特性の構成」](#)

gzip

[セクション2.4 「macOS への MySQL のインストール」](#)

[セクション23.3.3.6 「NDB Cluster データノードの定義」](#)

[セクション6.4.5.5 「監査ロギング特性の構成」](#)

[セクション1.6 「質問またはバグをレポートする方法」](#)

H

[\[index top\]](#)

help contents

[セクション4.5.1.4 「mysql クライアントのサーバー側ヘルプ」](#)

host

[セクション1.3 「MySQL 8.0 の新機能」](#)

hostname

[セクションB.3.2.2 「\[ローカルの\] MySQL サーバーに接続できません」](#)

I

[\[index top\]](#)

ibd2sdi

[セクション4.6.1 「ibd2sdi — InnoDB テーブルスペース SDI 抽出ユーティリティ」](#)

[セクション1.3 「MySQL 8.0 の新機能」](#)

[MySQL 用語集](#)

[セクション14.6 「シリアライズディクショナリ情報 \(SDI\)」](#)

icc

[セクション2.1.6 「コンパイラ固有のビルドの特徴」](#)

ifconfig

[セクション5.1.13.1 「IPv6 用のシステムサポートの確認」](#)

innochecksum

[セクション13.7.3.2 「CHECK TABLE ステートメント」](#)

[セクション4.6.2 「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティー」](#)

[セクション4.1 「MySQL プログラムの概要」](#)
[MySQL 用語集](#)

ip

[セクション5.1.14 「ネットワークネームスペースのサポート」](#)

iptables

[セクション23.5.17.1 「NDB Cluster のセキュリティーおよびネットワークの問題」](#)
[セクション18.10 「よくある質問」](#)

J

[\[index top\]](#)

java

[セクション6.4.4.4 「keyring_okv KMIP プラグインの使用」](#)

K

[\[index top\]](#)

kill

[セクションB.3.2.2 「\[ローカルの\] MySQL サーバーに接続できません」](#)
[セクション4.10 「MySQL での Unix シグナル処理」](#)
[セクション23.5.5 「NDB Cluster のローリング再起動の実行」](#)
[セクション23.4.1 「ndbd — NDB Cluster データノードデーモン」](#)

kinit

[セクション6.4.1.7 「LDAP プラガブル認証」](#)

klist

[セクション6.4.1.7 「LDAP プラガブル認証」](#)

ksh

[セクション4.2.1 「MySQL プログラムの起動」](#)
[セクション4.2.9 「環境変数の設定」](#)

kswapd

[セクション23.3.3.6 「NDB Cluster データノードの定義」](#)

kswitch

[セクション6.4.1.7 「LDAP プラガブル認証」](#)

L

[\[index top\]](#)

ldapsearch

[セクション6.4.1.7 「LDAP プラガブル認証」](#)

ldd mysqld

[セクション2.8 「FreeBSD に MySQL をインストールする」](#)

less

[セクション4.5.1.1 「mysql クライアントオプション」](#)
[セクション4.5.1.2 「mysql クライアントコマンド」](#)

lld

[セクション2.9.7 「MySQL ソース構成オプション」](#)

llvm

[セクション2.9.7 「MySQL ソース構成オプション」](#)

ln

[セクション8.12.2.2 「Unix 上の MyISAM へのシンボリックリンクの使用」](#)

logger

[セクション4.3.2 「mysqld_safe — MySQL サーバー起動スクリプト」](#)

ls

[セクション6.7 「SELinux」](#)

lsof +L1

[セクションB.3.3.5 「MySQL が一時ファイルを格納する場所」](#)

lz4

[セクション4.8.1 「lz4_decompress — mysqlpump LZ4-Compressed 出力の解凍」](#)

[セクション2.9.7 「MySQL ソース構成オプション」](#)

[セクション4.5.6 「mysqlpump — データベースバックアッププログラム」](#)

lz4_decompress

[セクション4.8.1 「lz4_decompress — mysqlpump LZ4-Compressed 出力の解凍」](#)

[セクション2.9.7 「MySQL ソース構成オプション」](#)

[セクション4.1 「MySQL プログラムの概要」](#)

[セクション4.5.6 「mysqlpump — データベースバックアッププログラム」](#)

[セクション4.8.3 「zlib_decompress — mysqlpump ZLIB 圧縮出力の解凍」](#)

M

[\[index top\]](#)

m4

[セクション2.9.2 「ソースインストールの前提条件」](#)

make

[セクション2.8 「FreeBSD に MySQL をインストールする」](#)

[セクション2.9.8 「MySQL のコンパイルに関する問題」](#)

[セクション2.13.3 「Perl DBI/DBD インタフェース使用の問題」](#)

[セクション2.9.2 「ソースインストールの前提条件」](#)

[セクション2.9.4 「標準ソース配布を使用して MySQL をインストールする」](#)

make && make install

[セクション23.2.1.4 「Linux でのソースからの NDB Cluster の構築」](#)

make install

[セクション23.2.1.4 「Linux でのソースからの NDB Cluster の構築」](#)

[セクション2.9.7 「MySQL ソース構成オプション」](#)

make package

[セクション2.9.7 「MySQL ソース構成オプション」](#)

[セクション2.9.4 「標準ソース配布を使用して MySQL をインストールする」](#)

make test

セクション2.13.1「Unix に Perl をインストールする」
セクション2.9.5「開発ソースツリーを使用して MySQL をインストールする」

make VERBOSE=1

セクション2.9.8「MySQL のコンパイルに関する問題」

md5

セクション2.1.4.1「MD5 チェックサムの確認」

md5.exe

セクション2.1.4.1「MD5 チェックサムの確認」

md5sum

セクション2.1.4.1「MD5 チェックサムの確認」

memcached

セクション15.20.2「InnoDB memcached のアーキテクチャー」
セクション15.20.4「InnoDB memcached の複数の get および Range クエリーのサポート」
セクション15.20「InnoDB memcached プラグイン」
セクション15.20.7「InnoDB memcached プラグインとレプリケーション」
セクション15.20.6.2「InnoDB memcached プラグインに対する memcached アプリケーションの適応」
セクション15.20.5「InnoDB memcached プラグインのセキュリティーに関する考慮事項」
セクション15.20.9「InnoDB memcached プラグインのトラブルシューティング」
セクション15.20.6.4「InnoDB memcached プラグインのトランザクション動作の制御」
セクション15.20.6.3「InnoDB memcached プラグインのパフォーマンスのチューニング」
セクション15.20.8「InnoDB memcached プラグインの内部」
セクション15.20.1「InnoDB memcached プラグインの利点」
セクション15.20.3「InnoDB memcached プラグインの設定」
セクション15.20.6「InnoDB memcached プラグイン用のアプリケーションの記述」
セクション15.20.6.1「InnoDB memcached プラグイン用の既存の MySQL スキーマの適応」
セクション15.14「InnoDB の起動オプションおよびシステム変数」
セクション15.20.6.5「memcached 操作に合わせた DML ステートメントの改変」
セクション1.3「MySQL 8.0 の新機能」
セクション2.9.7「MySQL ソース構成オプション」
MySQL 用語集
セクション15.20.6.6「ベースとなる InnoDB テーブルでの DML および DDL ステートメントの実行」

memcapable

セクション15.20.2「InnoDB memcached のアーキテクチャー」

memcat

セクション15.20.2「InnoDB memcached のアーキテクチャー」

memcp

セクション15.20.2「InnoDB memcached のアーキテクチャー」

memflush

セクション15.20.2「InnoDB memcached のアーキテクチャー」

memrm

セクション15.20.2「InnoDB memcached のアーキテクチャー」

memslap

セクション15.20.6.3「InnoDB memcached プラグインのパフォーマンスのチューニング」

mgmd

- セクション23.2 「NDB Cluster のインストール」
- セクション23.1.4 「NDB Cluster の新機能」
- セクション23.2.2.1 「バイナリリリースから Windows への NDB Cluster のインストール」

mkdir

- セクション13.1.12 「CREATE DATABASE ステートメント」
- セクション14.8 「データディクショナリの制限事項」

mklink

- セクション8.12.2.3 「Windows 上のデータベースへのシンボリックリンクの使用」

more

- セクション4.5.1.1 「mysql クライアントオプション」
- セクション4.5.1.2 「mysql クライアントコマンド」

mv

- セクション8.12.2.1 「Unix 上のデータベースへのシンボリックリンクの使用」
- セクション5.4.2.10 「エラーログファイルのフラッシュおよび名前変更」
- セクション5.4.6 「サーバーログの保守」
- セクション5.4.3 「一般クエリーログ」

my_print_defaults

- セクション4.7.2 「my_print_defaults — オプションファイルからのオプションの表示」
- セクション4.1 「MySQL プログラムの概要」
- セクション4.7 「プログラム開発ユーティリティ」

myisam_ftdump

- セクション4.6.3 「myisam_ftdump — 全文インデックス情報の表示」
- セクション4.1 「MySQL プログラムの概要」
- セクション12.10 「全文検索関数」

myisamchk

- セクション13.7.3.1 「ANALYZE TABLE ステートメント」
- セクション13.7.3.2 「CHECK TABLE ステートメント」
- セクション13.2.2 「DELETE ステートメント」
- セクション8.8.2 「EXPLAIN 出力フォーマット」
- セクション26.34 「INFORMATION_SCHEMA STATISTICS テーブル」
- セクション26.38 「INFORMATION_SCHEMA TABLES テーブル」
- セクション8.3.8 「InnoDB および MyISAM インデックス統計コレクション」
- セクション13.7.8.5 「LOAD INDEX INTO CACHE ステートメント」
- セクション8.6.1 「MyISAM クエリーの最適化」
- セクション16.2 「MyISAM ストレージエンジン」
- セクション7.6.2 「MyISAM テーブルのエラーのチェック方法」
- セクション16.2.3 「MyISAM テーブルのストレージフォーマット」
- セクション8.6.2 「MyISAM テーブルの一括データロード」
- セクション7.6 「MyISAM テーブルの保守とクラッシュリカバリ」
- セクション7.6.3 「MyISAM テーブルの修復方法」
- セクション7.6.4 「MyISAM テーブルの最適化」
- セクション16.2.4.1 「MyISAM テーブルの破損」
- セクション7.6.5 「MyISAM テーブル保守スケジュールのセットアップ」
- セクション16.2.1 「MyISAM 起動オプション」
- セクション4.6.4 「myisamchk — MyISAM テーブルメンテナンスユーティリティ」
- セクション4.6.4.5 「myisamchk によるテーブル情報の取得」
- セクション4.6.4.2 「myisamchk のチェックオプション」
- セクション4.6.4.1 「myisamchk の一般オプション」
- セクション4.6.4.3 「myisamchk の修復オプション」

セクション4.6.4.6 「myisamchk メモリー使用量」
セクション4.6.6 「myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成」
セクションB.3.3.3 「MySQL が繰り返しクラッシュする場合の対処方法」
セクション1.2.2 「MySQL の主な機能」
セクション12.10.6 「MySQL の全文検索の微調整」
セクション5.9.1 「MySQL サーバーのデバッグ」
セクション4.1 「MySQL プログラムの概要」
セクション5.9.1.6 「mysqld でのエラーの原因を見つけるためのサーバーログの使用」
セクション13.7.3.5 「REPAIR TABLE ステートメント」
セクション8.6.3 「REPAIR TABLE ステートメントの最適化」
セクション13.7.7.22 「SHOW INDEX ステートメント」
セクション13.7.7.38 「SHOW TABLE STATUS ステートメント」
セクション8.12.2.2 「Unix 上の MyISAM へのシンボリックリンクの使用」
セクション4.6.4.4 「その他の myisamchk オプション」
セクション7.6.1 「クラッシュリカバリへの myisamchk の使用」
セクション5.1.8 「サーバーシステム変数」
セクション5.9.1.7 「テーブルが破損した場合のテストケースの作成」
セクション8.4.6 「テーブルサイズの制限」
セクション7.2 「データベースバックアップ方法」
セクション16.2.3.2 「動的テーブルの特徴」
セクション16.2.3.3 「圧縮テーブルの特徴」
セクション8.11.5 「外部ロック」
セクション1.6 「質問またはバグをレポートする方法」
セクション16.2.4.2 「適切に閉じられなかったテーブルの問題」
セクション16.2.3.1 「静的 (固定長) テーブルの特長」

myisamchk *.MYI

セクション7.6.3 「MyISAM テーブルの修復方法」

myisamchk tbl_name

セクション7.6.2 「MyISAM テーブルのエラーのチェック方法」

myisamlog

セクション4.6.5 「myisamlog — MyISAM ログファイルの内容の表示」
セクション4.1 「MySQL プログラムの概要」

myisampack

セクション13.1.20 「CREATE TABLE ステートメント」
セクション16.7 「MERGE ストレージエンジン」
セクション16.7.1 「MERGE テーブルの長所と短所」
セクション16.2 「MyISAM ストレージエンジン」
セクション16.2.3 「MyISAM テーブルのストレージフォーマット」
セクション8.6.2 「MyISAM テーブルの一括データロード」
セクション4.6.4.5 「myisamchk によるテーブル情報の取得」
セクション4.6.4.3 「myisamchk の修復オプション」
セクション4.6.6 「myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成」
セクション4.1 「MySQL プログラムの概要」
セクション8.4.6 「テーブルサイズの制限」
セクション8.4.1 「データサイズの最適化」
セクション16.2.3.3 「圧縮テーブルの特徴」
セクション8.11.5 「外部ロック」
セクション13.1.20.7 「暗黙のカラム指定の変更」

mysql

セクション13.6.1 「BEGIN ... END 複合ステートメント」
セクション11.3.4 「BLOB 型と TEXT 型」
セクション13.1.17 「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」
セクション28.4.4.2 「diagnostics() プロシージャ」

セクション4.2.6 「DNS SRV レコードを使用したサーバーへの接続」
セクション2.5.6.2 「Docker での MySQL Server のデプロイに関するその他のトピック」
セクション2.5.6.1 「Docker を使用した MySQL Server デプロイメントの基本ステップ」
セクション13.6.7.3 「GET DIAGNOSTICS ステートメント」
セクション13.7.1.6 「GRANT ステートメント」
セクション17.1.5.2 「GTID ベースのレプリケーション用のマルチソースレプリカのプロビジョニング」
セクション13.8.3 「HELP ステートメント」
セクション13.2.5 「IMPORT TABLE ステートメント」
セクション15.18.2 「InnoDB のリカバリ」
セクション15.17.2 「InnoDB モニターの有効化」
セクション5.1.13.3 「IPv6 ローカルホストアドレスを使用した接続」
セクション5.1.13.4 「IPv6 非ローカルホストアドレスを使用した接続」
セクション12.18.3 「JSON 値を検索する関数」
セクション6.4.1.7 「LDAP プラガブル認証」
セクション8.2.1.19 「LIMIT クエリーの最適化」
セクション6.1.6 「LOAD DATA LOCAL のセキュリティ上の考慮事項」
セクション13.2.7 「LOAD DATA ステートメント」
セクション13.2.8 「LOAD XML ステートメント」
セクション2.4.1 「macOS への MySQL のインストールに関する一般的なノート」
セクション15.6.1.5 「MyISAM から InnoDB へのテーブルの変換」
セクションA.11 「MySQL 8.0 FAQ: MySQL の中国語、日本語、および韓国語の文字セット」
セクションA.10 「MySQL 8.0 FAQ: NDB Cluster」
セクション2.11.4 「MySQL 8.0 での変更」
セクション1.3 「MySQL 8.0 の新機能」
セクション6.4.7.3 「MySQL Enterprise Firewall の使用」
セクション2.3.3.1 「MySQL Installer の初期設定」
第23章 「MySQL NDB Cluster 8.0」
セクション5.1.15 「MySQL Server でのタイムゾーンのサポート」
第19章 「MySQL Shell」
セクション4.5.1 「mysql — MySQL コマンドラインクライアント」
セクション4.10 「MySQL での Unix シグナル処理」
セクション2.11 「MySQL のアップグレード」
セクション6.2.21 「MySQL への接続の問題のトラブルシューティング」
セクション1.8.1 「MySQL への貢献者」
セクション6.1.5 「MySQL を通常ユーザーとして実行する方法」
セクション4.5.1.4 「mysql クライアントのサーバー側ヘルプ」
セクション5.9.2 「MySQL クライアントのデバッグ」
セクション4.5.1.6 「mysql クライアントのヒント」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.5.1.2 「mysql クライアントコマンド」
セクション4.5.1.3 「mysql クライアントロギング」
セクション2.3.4.7 「MySQL ツールの PATH をカスタマイズする」
セクション2.11.14 「MySQL データベースのほかのマシンへのコピー」
セクション4.1 「MySQL プログラムの概要」
セクション4.2.1 「MySQL プログラムの起動」
MySQL 用語集
セクション4.3.3 「mysql.server — MySQL サーバー起動スクリプト」
セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティー」
セクション4.4.4 「mysql_tzinfo_to_sql — タイムゾーンテーブルのロード」
セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」
セクション5.9.1.6 「mysqld でのエラーの原因を見つけるためのサーバーログの使用」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション23.5.13 「NDB API 統計のカウントと変数」
セクション23.5.9 「NDB Cluster での MySQL Server の使用」
セクション23.3.3.7 「NDB Cluster での SQL およびその他の API ノードの定義」
セクション23.6.5 「NDB Cluster のレプリケーションの準備」
NDB Cluster の新しいバージョンへの NDB バックアップの復元

- セクション23.1.4 「NDB Cluster の新機能」
- セクション23.3 「NDB Cluster の構成」
- セクション23.5 「NDB Cluster の管理」
- セクション23.5.10.1 「NDB Cluster データオブジェクト」
- セクション23.5.7.1 「NDB Cluster データノードのオンラインでの追加: 一般的な問題」
- セクション23.5.7.2 「NDB Cluster データノードのオンラインでの追加: 基本手順」
- セクション23.5.7.3 「NDB Cluster データノードのオンラインでの追加: 詳細な例」
- セクション23.6 「NDB Cluster レプリケーション」
- セクション23.6.9 「NDB Cluster レプリケーションによる NDB Cluster バックアップ」
- セクション23.6.6 「NDB Cluster レプリケーションの開始 (シングルレプリケーションチャンネル)」
- セクション23.1.7.8 「NDB Cluster 専用の問題」
- セクション23.5.1 「NDB Cluster 管理クライアントのコマンド」
- セクション23.4.9 「ndb_desc — NDB テーブルの表示」
- セクション23.4.11 「ndb_drop_table — NDB テーブルの削除」
- セクション23.4.13 「ndb_import — NDB への CSV データのインポート」
- セクション23.4.14 「ndb_index_stat — NDB インデックス統計ユーティリティー」
- セクション23.4.5 「ndb_mgm — NDB Cluster 管理クライアント」
- セクション23.5.12 「NDB_STORED_USER での分散 MySQL 権限」
- セクション23.5.14.3 「ndbinfo backup_id テーブル」
- セクション23.5.14.37 「ndbinfo memory_per_fragment テーブル」
- セクション23.5.14.55 「ndbinfo transporters テーブル」
- セクション23.5.14 「ndbinfo: NDB Cluster 情報データベース」
- セクション23.4.2 「ndbinfo_select_all — ndbinfo テーブルからの選択」
- セクション6.4.1.5 「PAM プラガブル認証」
- セクション24.2.3.1 「RANGE COLUMNS パーティショニング」
- セクション13.7.1.8 「REVOKE ステートメント」
- root のパスワードのリセット: 一般的な手順
- セクション23.2.1.2 「RPM から NDB Cluster をインストール」
- セクション6.4.1.2 「SHA-2 プラガブル認証のキャッシュ」
- セクション6.4.1.3 「SHA-256 プラガブル認証」
- セクション13.7.7.35 「SHOW REPLICA | SLAVE STATUS ステートメント」
- セクション13.7.7.42 「SHOW WARNINGS ステートメント」
- セクション13.6.7.5 「SIGNAL ステートメント」
- セクション7.4.2 「SQL フォーマットバックアップのリロード」
- セクション4.2.5 「URI 類似文字列またはキーと値のペアを使用したサーバーへの接続」
- セクション23.2.2.3 「Windows での NDB Cluster の初期起動」
- セクション2.3.6 「Windows でのインストール後の手順」
- セクション2.3.4.8 「Windows のサービスとして MySQL を起動する」
- セクション13.2.15 「WITH (共通テーブル式)」
- セクション12.12 「XML 関数」
- セクション1.1 「このマニュアルについて」
- セクション6.2.8 「アカウントの追加、権限の割当ておよびアカウントの削除」
- セクション10.5 「アプリケーションの文字セットおよび照合順序の構成」
- セクション25.4.2 「イベントスケジューラの構成」
- セクションB.1 「エラーメッセージのソースと要素」
- セクションB.2 「エラー情報インタフェース」
- セクション4.2.2.6 「オプションのデフォルト、値を想定するオプション、および = 記号」
- セクション4.2.2.2 「オプションファイルの使用」
- セクション4.2.2.3 「オプションファイルの処理に影響するコマンド行オプション」
- セクション3.2 「クエリーの入力」
- セクション9.6 「クエリー属性」
- セクション6.4.1.4 「クライアント側クリアテキストプラガブル認証」
- セクション23.5.2.3 「クラスタログでのイベントバッファのレポート」
- セクション18.4.6 「グループレプリケーションでの MySQL Enterprise Backup の使用」
- セクション4.2.4 「コマンドオプションを使用した MySQL Server への接続」
- セクション4.2.2.1 「コマンド行でのオプションの使用」
- セクション9.7 「コメント」
- セクション1.7.2.4 「コメントの先頭としての "--」
- セクション2.10.3 「サーバーのテスト」
- セクション3.1 「サーバーへの接続とサーバーからの切断」

セクション5.1.8 「サーバーシステム変数」
セクション5.1.17 「サーバー側ヘルプのサポート」
セクション25.1 「ストアプログラムの定義」
セクション6.4.1.9 「ソケットピア資格証明プラグブル認証」
第3章 「チュートリアル」
セクション4.5.1.5 「テキストファイルから SQL ステートメントを実行する」
セクション23.2.5 「テーブルとデータを含む NDB Cluster の例」
セクション2.11.13 「テーブルまたはインデックスの再作成または修復」
セクション15.7.5.3 「デッドロックを最小化および処理する方法」
セクション14.1 「データディクショナリスキーマ」
セクション7.4.5.1 「データベースのコピーの作成」
セクション3.3.1 「データベースの作成と選択」
セクション25.3.1 「トリガーの構文と例」
セクション5.1.14 「ネットワークネームスペースのサポート」
セクション23.2.2.1 「バイナリリリースから Windows への NDB Cluster のインストール」
セクション7.5.1 「バイナリログを使用したポイントインタイムリカバリ」
セクション4.6.8.3 「バイナリログファイルのバックアップのための mysqlbinlog の使用」
セクション7.3 「バックアップおよびリカバリ戦略の例」
セクション7.1 「バックアップとリカバリの種類」
セクション7.4 「バックアップへの mysqldump の使用」
セクション3.5 「バッチモードでの MySQL の使用」
セクションB.3.2.8 「バケットが大きすぎます」
セクション6.1.2.1 「パスワードセキュリティのためのエンドユーザーガイドライン」
セクション27.12.9 「パフォーマンススキーマ接続属性テーブル」
セクション24.3.3 「パーティションとサブパーティションをテーブルと交換する」
セクション17.1.3.5 「フェイルオーバーおよびスケールアウトでの GTID の使用」
セクション6.2.17 「プラグブル認証」
セクション13.5 「プリペアドステートメント」
セクション4.2.2 「プログラムオプションの指定」
セクション4.2.2.4 「プログラムオプション修飾子」
セクション4.2.2.5 「プログラム変数の設定へのオプションの使用」
セクション17.1.5.5 「マルチソースレプリカの開始」
セクション17.1.5.3 「マルチソースレプリカへの GTID ベースのソースの追加」
セクション17.1.5.4 「マルチソースレプリカへのバイナリログベースレプリケーションソースの追加」
セクション17.1.5.1 「マルチソースレプリケーションの構成」
セクションB.3.2.6 「メモリー不足」
セクション7.3.2 「リカバリへのバックアップの使用」
セクション3.6 「一般的なクエリーの例」
セクション2.2 「一般的なバイナリを使用した MySQL の Unix/Linux へのインストール」
元のノードより多くのノードへのリストア
セクション2.10.4 「初期 MySQL アカウントの保護」
セクション7.4.4 「区切りテキストフォーマットバックアップのリロード」
セクションB.3.1 「問題の原因を判別する方法」
セクション8.13.1 「式と関数の速度の測定」
セクション12.16 「情報関数」
セクション4.2.7 「接続トランスポートプロトコル」
セクション4.2.8 「接続圧縮制御」
セクション10.4 「接続文字セットおよび照合順序」
セクション6.1.3 「攻撃者に対する MySQL のセキュアな状態の維持」
セクション9.1.1 「文字列リテラル」
セクション6.3.1 「暗号化接続を使用するための MySQL の構成」
セクション6.2.16 「期限切れパスワードのサーバー処理」
セクション4.9 「環境変数」
セクション1.6 「質問またはバグをレポートする方法」
第12章 「関数と演算子」
セクション15.8.10.2 「非永続的オプティマイザ統計のパラメータの構成」

mysql ...

セクション5.9.1.1 「デバッグのための MySQL のコンパイル」

mysql-server

セクション2.8 「FreeBSD に MySQL をインストールする」

mysql-test-run.pl

セクション5.9.3 「LOCK_ORDER ツール」

セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティー」

セクション4.2.2.2 「オプションファイルの使用」

mysql.exe

セクション4.5.1.6 「mysql クライアントのヒント」

セクション23.2.2.3 「Windows での NDB Cluster の初期起動」

セクション23.2.2.1 「バイナリリリースから Windows への NDB Cluster のインストール」

mysql.server

セクション2.5 「Linux に MySQL をインストールする」

セクション2.10.5 「MySQL を自動的に起動および停止する」

セクション4.1 「MySQL プログラムの概要」

セクション4.3.3 「mysql.server — MySQL サーバー起動スクリプト」

セクション4.6.9 「mysqldumpslow — スロークエリーログファイルの要約」

セクション23.2.1.2 「RPM から NDB Cluster をインストール」

セクション5.1.7 「サーバーコマンドオプション」

セクションB.3.3.7 「タイムゾーンの問題」

セクション6.1.3 「攻撃者に対する MySQL のセキュアな状態の維持」

mysql.server stop

セクション4.3.3 「mysql.server — MySQL サーバー起動スクリプト」

mysql_client_test_embedded

セクション1.3 「MySQL 8.0 の新機能」

mysql_config

セクション1.3 「MySQL 8.0 の新機能」

セクション2.9.8 「MySQL のコンパイルに関する問題」

セクション4.1 「MySQL プログラムの概要」

セクション4.7.1 「mysql_config — クライアントのコンパイル用オプションの表示」

mysql_config_editor

セクション4.7.2 「my_print_defaults — オプションファイルからのオプションの表示」

セクション4.6.4.1 「myisamchk の一般オプション」

セクション4.5.1.1 「mysql クライアントオプション」

セクション4.1 「MySQL プログラムの概要」

セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティー」

セクション4.4.2 「mysql_secure_installation — MySQL インストールのセキュリティー改善」

セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」

セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.5 「mysqlimport — データインポートプログラム」

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」

セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」

セクション4.2.2.2 「オプションファイルの使用」

セクション4.2.2.3 「オプションファイルの処理に影響するコマンド行オプション」

セクション6.1.2.1 「パスワードセキュリティーのためのエンドユーザーガイドライン」

セクション4.9 「環境変数」

mysql_install_db

セクション1.3「MySQL 8.0 の新機能」

mysql_plugin

セクション1.3「MySQL 8.0 の新機能」

mysql_secure_installation

セクション4.1「MySQL プログラムの概要」

セクション4.4.2「mysql_secure_installation — MySQL インストールのセキュリティー改善」

セクション2.7.1「Solaris PKG を使用して Solaris に MySQL をインストールする」

セクション2.5.5「オラクルからの Debian パッケージを使用して MySQL を Linux にインストールする」

セクション2.10.1「データディレクトリの初期化」

セクション2.5.7「ネイティブソフトウェアリポジトリから MySQL を Linux にインストールする」

セクション2.10.4「初期 MySQL アカウントの保護」

mysql_setpermission

セクション1.8.1「MySQL への貢献者」

mysql_ssl_rsa_setup

セクション6.3.3.1「MySQL を使用した SSL および RSA 証明書とキーの作成」

セクション4.1「MySQL プログラムの概要」

セクション4.4.3「mysql_ssl_rsa_setup — SSL/RSA ファイルの作成」

セクション6.3.3.3「openssl を使用した RSA キーの作成」

セクション6.3.3.2「openssl を使用した SSL 証明書およびキーの作成」

セクション4.2.3「サーバーに接続するためのコマンドオプション」

セクション2.10.1「データディレクトリの初期化」

セクション6.3.1「暗号化接続を使用するための MySQL の構成」

mysql_stmt_execute()

セクション5.1.10「サーバーステータス変数」

mysql_stmt_prepare()

セクション5.1.10「サーバーステータス変数」

mysql_tzinfo_to_sql

セクション5.1.15「MySQL Server でのタイムゾーンをサポート」

セクション4.1「MySQL プログラムの概要」

セクション4.4.4「mysql_tzinfo_to_sql — タイムゾーンテーブルのロード」

mysql_upgrade

セクション17.1.3.7「GTID ベースレプリケーションの制約」

セクション1.3「MySQL 8.0 の新機能」

セクション2.11.7「MySQL Yum リポジトリを使用する MySQL のアップグレード」

セクション2.11.3「MySQL のアップグレードプロセスの内容」

セクション4.1「MySQL プログラムの概要」

セクション4.4.5「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

セクション23.6.4「NDB Cluster レプリケーションスキーマおよびテーブル」

セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」

セクション6.4.1.3「SHA-256 プラガブル認証」

セクション2.11.6「Unix/Linux での MySQL バイナリまたはパッケージベースのインストールのアップグレード」

セクション2.11.10「Windows 上の MySQL をアップグレードする」

セクション18.2.1.2「グループレプリケーション用のインスタンスの構成」

セクション5.1.8「サーバーシステム変数」

セクション2.11.13「テーブルまたはインデックスの再作成または修復」

セクション17.5.3「レプリケーションセットアップをアップグレードする」

セクション4.2.8「接続圧縮制御」

mysqlaccess

セクション1.8.1「MySQL への貢献者」

mysqladmin

セクションB.3.2.2「[ローカルの] MySQL サーバーに接続できません」
セクション13.1.12「CREATE DATABASE ステートメント」
セクション13.1.24「DROP DATABASE ステートメント」
セクション13.7.8.3「FLUSH ステートメント」
セクション17.1.3.4「GTID を使用したレプリケーションのセットアップ」
セクション2.4.1「macOS への MySQL のインストールに関する一般的なノート」
セクション7.6.3「MyISAM テーブルの修復方法」
セクションA.11「MySQL 8.0 FAQ: MySQL の中国語、日本語、および韓国語の文字セット」
セクション2.11.4「MySQL 8.0 での変更」
セクション2.3.3.1「MySQL Installer の初期設定」
セクション5.4「MySQL Server ログ」
セクションB.3.3.3「MySQL が繰り返しクラッシュする場合の対処方法」
セクション6.2.2「MySQL で提供される権限」
セクション1.2.2「MySQL の主な機能」
セクション1.8.1「MySQL への貢献者」
セクション5.9.1「MySQL サーバーのデバッグ」
セクション2.3.4.7「MySQL ツールの PATH をカスタマイズする」
セクション4.1「MySQL プログラムの概要」
セクション4.6.7「mysql_config_editor — MySQL 構成ユーティリティー」
セクション4.5.2「mysqladmin — A MySQL Server 管理プログラム」
セクション4.3.4「mysqld_multi — 複数の MySQL サーバーの管理」
セクション17.4.1.1「mysqldump を使用したレプリカのバックアップ」
セクション23.2.1.2「RPM から NDB Cluster をインストール」
セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」
セクション6.4.1.3「SHA-256 プラガブル認証」
セクション5.8.3「Unix 上での複数の MySQL インスタンスの実行」
セクション2.3.4.6「Windows のコマンド行からの MySQL の起動」
セクション2.3.4.8「Windows のサービスとして MySQL を起動する」
セクション2.11.10「Windows 上の MySQL をアップグレードする」
セクション6.2.14「アカウントパスワードの割り当て」
セクション4.2.2.2「オプションファイルの使用」
セクション6.4.1.4「クライアント側クリアテキストプラガブル認証」
セクション4.2.4「コマンドオプションを使用した MySQL Server への接続」
セクション4.2.2.1「コマンド行でのオプションの使用」
セクション2.10.3「サーバーのテスト」
セクション5.1.19「サーバーの停止プロセス」
セクション5.1.1「サーバーの構成」
セクション27.12.9「パフォーマンススキーマ接続属性テーブル」
セクション6.2.17「プラガブル認証」
セクション2.10.4「初期 MySQL アカウントの保護」
セクションB.3.1「問題の原因を判別する方法」
セクション4.2.8「接続圧縮制御」
セクション10.4「接続文字セットおよび照合順序」
セクション1.6「質問またはバグをレポートする方法」

mysqladmin debug

セクション13.7.1.6「GRANT ステートメント」
セクション6.2.2「MySQL で提供される権限」
セクション5.9.1「MySQL サーバーのデバッグ」
セクション25.4.5「イベントスケジューラのスレータス」

mysqladmin extended-status

セクション13.7.7.37「SHOW STATUS ステートメント」

mysqladmin flush-hosts

セクション5.1.12.3「DNS ルックアップとホストキャッシュ」
セクション6.2.21「MySQL への接続の問題のトラブルシューティング」

mysqladmin flush-logs

セクション4.5.2「mysqladmin — A MySQL Server 管理プログラム」
セクション5.4.2.10「エラーログファイルのフラッシュおよび名前変更」
セクション5.4.6「サーバーログの保守」
セクション5.4.4「バイナリログ」
セクション7.3.1「バックアップポリシーの確立」
セクション7.3.3「バックアップ戦略サマリー」
セクション17.2.4.1「リレーログ」

mysqladmin flush-privileges

セクション6.2.21「MySQL への接続の問題のトラブルシューティング」
セクション2.11.14「MySQL データベースのほかのマシンへのコピー」
セクション4.5.2「mysqladmin — A MySQL Server 管理プログラム」
セクション5.1.7「サーバーコマンドオプション」
セクション6.2.3「付与テーブル」
セクション6.2.13「権限変更が有効化される時期」

mysqladmin flush-tables

セクション8.6.2「MyISAM テーブルの一括データロード」
セクション4.6.6「mysampack — 圧縮された読み取り専用の MyISAM テーブルの生成」
セクション8.4.3.1「MySQL でのテーブルのオープンとクローズの方法」
セクション8.12.3.1「MySQL のメモリーの使用方法」
セクション7.6.1「クラッシュリカバリへの myisamchk の使用」
セクション8.11.5「外部ロック」

mysqladmin flush-xxx

セクション6.2.8「アカウントの追加、権限の割当ておよびアカウントの削除」

mysqladmin kill

セクション13.7.8.4「KILL ステートメント」
セクションB.3.3.4「MySQL が満杯のディスクを処理する方法」
セクション6.2.2「MySQL で提供される権限」
セクションB.3.2.7「MySQL サーバーが存在しなくなりました」
セクション12.15「ロック関数」

mysqladmin password

セクション4.5.2「mysqladmin — A MySQL Server 管理プログラム」

mysqladmin processlist

セクション13.7.8.4「KILL ステートメント」
セクション6.2.2「MySQL で提供される権限」
セクション27.12.19.9「processlist テーブル」
セクション6.2.8「アカウントの追加、権限の割当ておよびアカウントの削除」
セクション8.14.1「プロセスリストへのアクセス」
セクション6.1.3「攻撃者に対する MySQL のセキュアな状態の維持」

mysqladmin processlist status

セクション5.9.1「MySQL サーバーのデバッグ」

mysqladmin refresh

セクション8.4.3.1「MySQL でのテーブルのオープンとクローズの方法」
セクション6.2.8「アカウントの追加、権限の割当ておよびアカウントの削除」
セクション5.4.6「サーバーログの保守」

mysqladmin reload

セクション6.2.8 「アカウントの追加、権限の割当ておよびアカウントの削除」
セクション6.2.20 「アカウントリソース制限の設定」
セクション5.1.7 「サーバーコマンドオプション」
セクション6.2.3 「付与テーブル」
セクション6.2.13 「権限変更が有効化される時期」
セクション1.6 「質問またはバグをレポートする方法」

mysqladmin reload version

セクション1.6 「質問またはバグをレポートする方法」

mysqladmin shutdown

セクション13.7.1.6 「GRANT ステートメント」
セクション7.6.3 「MyISAM テーブルの修復方法」
セクションA.10 「MySQL 8.0 FAQ: NDB Cluster」
セクションB.3.3.3 「MySQL が繰り返しクラッシュする場合の対処方法」
セクション6.2.2 「MySQL で提供される権限」
セクション6.1.5 「MySQL を通常ユーザーとして実行する方法」
セクション4.3.3 「mysql.server — MySQL サーバー起動スクリプト」
セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」
セクション23.2.6 「NDB Cluster の安全なシャットダウンと再起動」
セクション23.5.7.3 「NDB Cluster データノードのオンラインでの追加: 詳細な例」
セクション23.4.13 「ndb_import — NDB への CSV データのインポート」
セクション13.7.8.9 「SHUTDOWN ステートメント」
セクション2.3.4.8 「Windows のサービスとして MySQL を起動する」
セクション6.2.7 「アクセス制御、ステージ 2: リクエストの確認」
セクション18.7.3.2 「グループレプリケーションメンバーのアップグレード」
セクション5.1.19 「サーバーの停止プロセス」
セクション5.9.1.7 「テーブルが破損した場合のテストケースの作成」
セクション5.9.1.2 「トレースファイルの作成」
セクション2.4.2 「ネイティブパッケージを使用した macOS への MySQL のインストール」
セクション17.5.1.31 「レプリケーションと一時テーブル」

mysqladmin status

セクション8.4.3.1 「MySQL でのテーブルのオープンとクローズの方法」
セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」

mysqladmin variables

セクションB.3.2.7 「MySQL サーバーが存在しなくなりました」
セクション13.7.7.41 「SHOW VARIABLES ステートメント」

mysqladmin variables extended-status processlist

セクション1.6 「質問またはバグをレポートする方法」

mysqladmin ver

セクション5.9.1.1 「デバッグのための MySQL のコンパイル」

mysqladmin version

セクションB.3.2.2 「[ローカルの] MySQL サーバーに接続できません」
セクションB.3.3.3 「MySQL が繰り返しクラッシュする場合の対処方法」
セクションB.3.2.7 「MySQL サーバーが存在しなくなりました」
セクション2.10.3 「サーバーのテスト」
セクション1.6 「質問またはバグをレポートする方法」

mysqlanalyze

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

mysqlbackup

セクション2.5.6.2 「Docker での MySQL Server のデプロイに関するその他のトピック」

セクション15.18.1 「InnoDB バックアップ」

MySQL 用語集

セクション18.7.3.4 「mysqlbackup を使用したグループレプリケーションアップグレード」

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション18.4.6 「グループレプリケーションでの MySQL Enterprise Backup の使用」

セクション7.1 「バックアップとリカバリの種類」

セクション17.1.3.5 「フェイルオーバーおよびスケールアウトでの GTID の使用」

ローデータファイルを使用したデータスナップショットの作成

mysqlbinlog

セクション13.7.8.1 「BINLOG ステートメント」

セクション17.1.3.2 「GTID ライフサイクル」

セクション17.1.3.1 「GTID 形式および格納」

セクション15.18.2 「InnoDB のリカバリ」

セクション6.1.6 「LOAD DATA LOCAL のセキュリティー上の考慮事項」

セクション6.2.2 「MySQL で提供される権限」

セクションB.3.7 「MySQL の既知の問題」

セクション4.5.1.1 「mysql クライアントオプション」

セクション4.1 「MySQL プログラムの概要」

MySQL 用語集

セクション4.6.8.1 「mysqlbinlog 16 進ダンプ形式」

セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」

セクション4.6.8.4 「mysqlbinlog サーバー ID の指定」

セクション4.6.8.2 「mysqlbinlog 行イベントの表示」

NDB Cluster システム変数

セクション23.6.9.2 「NDB Cluster レプリケーションを使用したポイントインタイムリカバリ」

セクション6.4.1.2 「SHA-2 プラガブル認証のキャッシュ」

セクション6.4.1.3 「SHA-256 プラガブル認証」

セクション13.7.7.2 「SHOW BINLOG EVENTS ステートメント」

セクション13.7.7.32 「SHOW RELAYLOG EVENTS ステートメント」

セクション13.4.2.7 「START REPLICAS | SLAVE ステートメント」

セクション12.24 「その他の関数」

セクション7.5.2 「イベントの位置を使用したポイントインタイムリカバリ」

セクション5.1.8 「サーバーシステム変数」

セクション17.2.1.1 「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」

セクション17.1.7.3 「トランザクションのスキップ」

セクション17.1.6.4 「バイナリロギングのオプションと変数」

セクション5.4.4 「バイナリログ」

バイナリログのトランザクション圧縮のモニタリング

セクション17.3.2.1 「バイナリログの暗号化の範囲」

セクション7.5.1 「バイナリログを使用したポイントインタイムリカバリ」

セクション5.4.4.5 「バイナリログトランザクション圧縮」

セクション17.3.2 「バイナリログファイルとリレーログファイルの暗号化」

セクション4.6.8.3 「バイナリログファイルのバックアップのための mysqlbinlog の使用」

セクション27.12.9 「パフォーマンススキーマ接続属性テーブル」

セクション24.3.5 「パーティションに関する情報を取得する」

セクション17.1.3.5 「フェイルオーバーおよびスケールアウトでの GTID の使用」

セクション7.3.2 「リカバリへのバックアップの使用」

セクション17.2.4.1 「リレーログ」

セクション17.3.3.1 「レプリケーション PRIVILEGE_CHECKS_USER アカウントの権限」

セクション17.5.1.19 「レプリケーションと LOAD DATA」

セクション17.5.1.39 「レプリケーションと変数」

セクション17.5.5 「レプリケーションバグまたは問題を報告する方法」

セクション17.3.3 「レプリケーション権限チェック」

セクション5.4.3 「一般クエリーログ」

セクション17.3.3.3 「失敗したレプリケーション権限チェックからのリカバリ」

セクション4.2.8 「接続圧縮制御」

セクション17.2.1.2 「行ベースロギングおよびレプリケーションの使用」
セクション17.4.11 「遅延レプリケーション」
セクション13.1.20.10 「非表示カラム」

mysqlbinlog binary-log-file | mysql

セクション5.9.1.7 「テーブルが破損した場合のテストケースの作成」

mysqlbinlog|mysql

セクションB.3.7 「MySQL の既知の問題」

mysqlcheck

セクション4.7.2 「my_print_defaults — オプションファイルからのオプションの表示」
セクション16.2 「MyISAM ストレージエンジン」
セクション7.6 「MyISAM テーブルの保守とクラッシュリカバリ」
セクション2.11.4 「MySQL 8.0 での変更」
セクション1.3 「MySQL 8.0 の新機能」
セクション1.2.2 「MySQL の主な機能」
セクション4.1 「MySQL プログラムの概要」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション6.4.1.2 「SHA-2 プラガブル認証のキャッシュ」
セクション6.4.1.3 「SHA-256 プラガブル認証」
セクション2.11.5 「アップグレード用のインストールの準備」
セクション6.4.1.4 「クライアント側クリアテキストプラガブル認証」
セクション5.1.8 「サーバーシステム変数」
セクション2.11.13 「テーブルまたはインデックスの再作成または修復」
セクション4.2.8 「接続圧縮制御」
セクション10.4 「接続文字セットおよび照合順序」

mysqld

セクション5.8 「1 つのマシン上での複数の MySQL インスタンスの実行」
セクションB.3.2.2 「[ローカルの] MySQL サーバーに接続できません」
セクション13.1.2 「ALTER DATABASE ステートメント」
セクション16.6 「BLACKHOLE ストレージエンジン」
セクション13.4.2.2 「CHANGE REPLICATION FILTER ステートメント」
セクション4.4.1 「comp_err — MySQL エラーメッセージファイルのコンパイル」
セクション13.1.20 「CREATE TABLE ステートメント」
セクション5.9.4 「DEBUG パッケージ」
セクション5.1.12.3 「DNS ルックアップとホストキャッシュ」
セクション2.5.6.1 「Docker を使用した MySQL Server デプロイメントの基本ステップ」
セクション15.6.3.2 「File-Per-Table テーブルスペース」
セクション5.9.1.4 「gdb での mysqld のデバッグ」
セクション17.1.3.4 「GTID を使用したレプリケーションのセットアップ」
セクション15.20.2 「InnoDB memcached のアーキテクチャー」
セクション15.20.7 「InnoDB memcached プラグインとレプリケーション」
セクション15.20.6.4 「InnoDB memcached プラグインのトランザクション動作の制御」
セクション15.20.3 「InnoDB memcached プラグインの設定」
セクション8.5.4 「InnoDB redo ロギングの最適化」
セクション15.21.1 「InnoDB の I/O に関する問題のトラブルシューティング」
セクション15.7.5 「InnoDB のデッドロック」
セクション15.21 「InnoDB のトラブルシューティング」
セクション15.18.2 「InnoDB のリカバリ」
セクション15.21.2 「InnoDB のリカバリの強制的な実行」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.8.1 「InnoDB の起動構成」
セクション15.6.1.4 「InnoDB テーブルの移動またはコピー」
セクション15.11.1 「InnoDB ディスク I/O」
セクション15.18.1 「InnoDB バックアップ」
セクション15.17.2 「InnoDB モニターの有効化」

セクション13.2.6.2 「INSERT ... ON DUPLICATE KEY UPDATE ステートメント」
セクション13.2.6 「INSERT ステートメント」
セクション13.7.8.4 「KILL ステートメント」
セクション23.2.1 「Linux での NDB Cluster のインストール」
セクション23.2.1.4 「Linux でのソースからの NDB Cluster の構築」
セクション23.2.1.1 「Linux への NDB Cluster バイナリリリースのインストール」
セクション6.1.6 「LOAD DATA LOCAL のセキュリティー上の考慮事項」
セクション13.2.7 「LOAD DATA ステートメント」
セクション2.3.5 「Microsoft Windows MySQL Server インストールのトラブルシューティング」
セクション2.3 「Microsoft Windows に MySQL をインストールする」
セクション2.3.1 「Microsoft Windows 上での MySQL のインストールレイアウト」
セクション16.2 「MyISAM ストレージエンジン」
セクション7.6.3 「MyISAM テーブルの修復方法」
セクション16.2.4.1 「MyISAM テーブルの破損」
セクション7.6.5 「MyISAM テーブル保守スケジュールのセットアップ」
セクション16.2.1 「MyISAM 起動オプション」
セクション4.6.4 「myisamchk — MyISAM テーブルメンテナンスユーティリティー」
セクション4.6.4.2 「myisamchk のチェックオプション」
セクション4.6.4.1 「myisamchk の一般オプション」
セクション4.6.6 「myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成」
セクションA.11 「MySQL 8.0 FAQ: MySQL の中国語、日本語、および韓国語の文字セット」
セクションA.10 「MySQL 8.0 FAQ: NDB Cluster」
セクションA.3 「MySQL 8.0 FAQ: サーバー SQL モード」
セクションA.1 「MySQL 8.0 FAQ: 全般」
セクション1.3 「MySQL 8.0 の新機能」
第23章 「MySQL NDB Cluster 8.0」
セクション5.1 「MySQL Server」
セクション23.1.6 「MySQL Server NDB Cluster と比較した InnoDB の使用」
セクション5.1.15 「MySQL Server でのタイムゾーンのサポート」
セクション2.10.2.1 「MySQL Server の起動時の問題のトラブルシューティング」
セクション5.4 「MySQL Server ログ」
第28章 「MySQL sys スキーマ」
セクションB.3.3.5 「MySQL が一時ファイルを格納する場所」
セクションB.3.3.3 「MySQL が繰り返しクラッシュする場合の対処方法」
セクション4.10 「MySQL での Unix シグナル処理」
セクション2.11.3 「MySQL のアップグレードプロセスの内容」
セクション1.8.5 「MySQL のサポータ」
セクション8.12.3.1 「MySQL のメモリーの使用方法」
セクション12.10.6 「MySQL の全文検索の微調整」
セクション1.7 「MySQL の標準への準拠」
セクション6.2.21 「MySQL への接続の問題のトラブルシューティング」
セクション2.10.5 「MySQL を自動的に起動および停止する」
セクション6.1.5 「MySQL を通常ユーザーとして実行する方法」
セクション2.3.4.9 「MySQL インストールのテスト」
セクション4.5.1.6 「mysql クライアントのヒント」
セクションB.3.2.7 「MySQL サーバーが存在しなくなりました」
セクション5.9.1 「MySQL サーバーのデバッグ」
第5章 「MySQL サーバーの管理」
セクション2.3.4.3 「MySQL サーバータイプの選択」
セクション2.9.7 「MySQL ソース構成オプション」
セクション4.1 「MySQL プログラムの概要」
セクション6.7.5.2 「MySQL 機能の TCP ポートコンテキストの設定」
MySQL 用語集
セクション4.3.3 「mysql.server — MySQL サーバー起動スクリプト」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.3.1 「mysqld — MySQL サーバー」
セクション5.9.1.6 「mysqld でのエラーの原因を見つけるためのサーバーログの使用」
セクション6.7.5.1 「mysqld の TCP ポートコンテキストの設定」
セクション4.3.4 「mysqld_multi — 複数の MySQL サーバーの管理」

セクション4.3.2 「mysqld_safe — MySQL サーバー起動スクリプト」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション23.1.5 「NDB 8.0 で追加、非推奨または削除されたオプション、変数、およびパラメータ」
セクション23.5.13 「NDB API 統計のカウンタと変数」
セクション23.2.8.2 「NDB Cluster Auto-Installer の使用」
セクション23.3.2.5 「NDB Cluster mysqld オプションおよび変数のリファレンス」
セクション23.5.17.3 「NDB Cluster および MySQL セキュリティー手順」
セクション23.5.9 「NDB Cluster での MySQL Server の使用」
セクション23.3.3.7 「NDB Cluster での SQL およびその他の API ノードの定義」
NDB Cluster の MySQL Server オプション
セクション23.1.7.1 「NDB Cluster の SQL 構文に準拠していません」
セクション23.2.7 「NDB Cluster のアップグレードおよびダウングレード」
セクション23.2 「NDB Cluster のインストール」
セクション23.1.1 「NDB Cluster のコア概念」
セクション23.6.5 「NDB Cluster のレプリケーションの準備」
セクション23.5.5 「NDB Cluster のローリング再起動の実行」
セクション23.2.3 「NDB Cluster の初期構成」
セクション23.2.4 「NDB Cluster の初期起動」
セクション23.3.3.2 「NDB Cluster の推奨開始構成」
NDB Cluster の新しいバージョンへの NDB バックアップの復元
セクション23.1.4 「NDB Cluster の新機能」
セクション23.1 「NDB Cluster の概要」
セクション23.3 「NDB Cluster の構成」
セクション23.5 「NDB Cluster の管理」
NDB Cluster システム変数
NDB Cluster ステータス変数
セクション23.3.3.6 「NDB Cluster データノードの定義」
セクション23.1.2 「NDB Cluster ノード、ノードグループ、フラグメントレプリカ、およびパーティション」
セクション23.4 「NDB Cluster プログラム」
セクション23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」
セクション23.2.2.4 「NDB Cluster プロセスを Windows サービスとしてインストール」
セクション23.6 「NDB Cluster レプリケーション」
セクション23.6.10 「NDB Cluster レプリケーション: 双方向および循環レプリケーション」
セクション23.6.7 「NDB Cluster レプリケーションでの 2 つのレプリケーションチャネルの使用」
セクション23.6.8 「NDB Cluster レプリケーションによるフェイルオーバーの実装」
セクション23.6.2 「NDB Cluster レプリケーションの一般的な要件」
セクション23.6.3 「NDB Cluster レプリケーションの既知の問題」
セクション23.6.11 「NDB Cluster レプリケーションの競合解決」
セクション23.6.6 「NDB Cluster レプリケーションの開始 (シングルレプリケーションチャネル)」
セクション23.6.4 「NDB Cluster レプリケーションスキーマおよびテーブル」
セクション23.1.7.8 「NDB Cluster 専用の問題」
セクション23.3.3.1 「NDB Cluster 構成: 基本例」
セクション23.3.2 「NDB Cluster 構成パラメータ、オプション、および変数の概要」
セクション23.5.1 「NDB Cluster 管理クライアントのコマンド」
セクション23.2.8.1 「NDB Cluster 自動インストーラの要件」
NDB バックアップを以前のバージョンの NDB Cluster に復元
セクション23.4.13 「ndb_import — NDB への CSV データのインポート」
セクション23.4.4 「ndb_mgrmd — NDB Cluster 管理サーバーデーモン」
セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」
セクション23.4.27 「ndb_show_tables — NDB テーブルのリストの表示」
セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」
セクション23.5.14.39 「ndbinfo operations_per_fragment テーブル」
セクション23.5.14.44 「ndbinfo server_locks テーブル」
セクション23.5.14 「ndbinfo: NDB Cluster 情報データベース」
セクション13.7.3.4 「OPTIMIZE TABLE ステートメント」
セクション2.5.4 「Oracle の RPM パッケージを使用した Linux への MySQL のインストール」
セクション13.7.3.5 「REPAIR TABLE ステートメント」
セクション13.4.2.5 「RESET REPLICA | SLAVE ステートメント」
セクション13.7.8.8 「RESTART ステートメント」
root のパスワードのリセット: Windows システム

root パスワードのリセット: Unix および Unix- 類似システム
セクション23.2.1.2 「RPM から NDB Cluster をインストール」
セクション13.2.10.1 「SELECT ... INTO ステートメント」
セクション6.7 「SELinux」
セクション13.7.7.15 「SHOW ENGINE ステートメント」
セクション2.7 「Solaris への MySQL のインストール」
セクション2.9.6 「SSL ライブラリサポートの構成」
セクション13.4.2.7 「START REPLICAS | SLAVE ステートメント」
セクション2.5.9 「systemd を使用した MySQL Server の管理」
セクション8.12.2.2 「Unix 上の MyISAM へのシンボリックリンクの使用」
セクション2.11.6 「Unix/Linux での MySQL バイナリまたはパッケージベースのインストールのアップグレード」
セクション5.9.1.3 「WER と PDB を使用した Windows クラッシュダンプの作成」
セクション2.3.4.6 「Windows のコマンド行からの MySQL の起動」
セクション2.3.4.8 「Windows のサービスとして MySQL を起動する」
セクション5.8.2.1 「Windows コマンド行での複数の MySQL インスタンスの起動」
セクション5.8.2.2 「Windows サービスとして複数の MySQL インスタンスの起動」
セクション2.11.10 「Windows 上の MySQL をアップグレードする」
セクション1.1 「このマニュアルについて」
セクション12.24 「その他の関数」
セクション2.11.12 「アップグレードのトラブルシューティング」
セクション2.1.2 「インストールする MySQL のバージョンと配布の選択」
セクション5.4.2.3 「エラーイベントフィールド」
セクション10.12 「エラーメッセージ言語の設定」
セクション5.4.2 「エラーログ」
セクション5.4.2.10 「エラーログファイルのフラッシュおよび名前変更」
セクション5.4.2.9 「エラーログ出力形式」
セクション8.2.1.5 「エンジンコンディションプッシュダウンの最適化」
セクション4.2.2.2 「オプションファイルの使用」
セクション4.2.2.3 「オプションファイルの処理に影響するコマンド行オプション」
セクションB.3.5 「オプティマイザ関連の問題」
セクション6.4.4.9 「キーリングキーストア間のキーの移行」
セクション23.5.2.3 「クラスタログでのイベントバッファのレポート」
セクション7.6.1 「クラッシュリカバリへの myisamchk の使用」
セクション15.8.3.7 「コアファイルからのバッファープールページの除外」
セクション9.7 「コメント」
セクション4.3 「サーバーおよびサーバーの起動プログラム」
セクション2.10.3 「サーバーのテスト」
セクション5.1.1 「サーバーの構成」
セクション5.1.7 「サーバーコマンドオプション」
セクション5.1.8 「サーバーシステム変数」
セクション5.1.10 「サーバーステータス変数」
セクション5.4.6 「サーバーログの保守」
セクション10.3.2 「サーバー文字セットおよび照合順序」
セクション5.4.2.8 「システムログへのエラーロギング」
セクション5.9.1.5 「スタックトレースの使用」
セクション5.4.5 「スロークエリログ」
セクション6.1.4 「セキュリティ関連の mysqld オプションおよび変数」
セクションB.3.3.7 「タイムゾーンの問題」
セクション5.9.1.7 「テーブルが破損した場合のテストケースの作成」
セクションB.3.2.17 「テーブルの破損の問題」
セクション8.11.2 「テーブルロックの問題」
セクション5.9.1.1 「デバッグのための MySQL のコンパイル」
セクション5.4.2.2 「デフォルトのエラーログ保存先の構成」
セクション14.1 「データディクショナリスキーマ」
セクション2.10.1 「データディレクトリの初期化」
セクション5.9.1.2 「トレースファイルの作成」
セクション2.4.2 「ネイティブパッケージを使用した macOS への MySQL のインストール」
セクション5.1.14 「ネットワークネームスペースのサポート」
セクション23.2.2.1 「バイナリリリースから Windows への NDB Cluster のインストール」
セクション17.1.6.4 「バイナリロギングのオプションと変数」

セクション5.4.4.1「バイナリロギング形式」
セクション5.4.4「バイナリログ」
セクションB.3.2.8「バケットが大きすぎます」
セクション27.12.14.2「パフォーマンススキーマ variables_info テーブル」
セクション27.3「パフォーマンススキーマ起動構成」
セクションB.3.2.16「ファイルが見つからず同様のエラーが発生しました」
セクションB.3.2.11「ファイルを作成/書き込みできない」
セクションB.3.3.1「ファイル権限の問題」
セクション17.4.8「フェイルオーバー中のソースの切替え」
セクション4.2.2「プログラムオプションの指定」
セクション4.2.2.4「プログラムオプション修飾子」
セクションB.3.2.13「ユーザーを無視します」
セクション5.1.16「リソースグループ」
セクション17.1.6.1「レプリケーション、バイナリロギングオプション、および変数のリファレンス」
セクション17.1.6「レプリケーションおよびバイナリロギングのオプションと変数」
セクション17.5.1.28「レプリケーションとソースまたはレプリカの停止」
セクション17.5.1.34「レプリケーションとトランザクションの非一貫性」
セクション17.1.6.2「レプリケーションソースのオプションと変数」
セクション17.2.4.2「レプリケーションメタデータリポジトリ」
セクション17.3.3「レプリケーション権限チェック」
セクション5.4.3「一般クエリーログ」
セクション8.14.3「一般的なスレッドの状態」
セクション2.2「一般的なバイナリを使用した MySQL の Unix/Linux へのインストール」
セクション15.6.4「二重書き込みバッファ」
元のノードより多くのノードへのリストア
セクション8.2.1.23「全テーブルスキャンの回避」
セクション2.10.4「初期 MySQL アカウントの保護」
セクションB.3.1「問題の原因を判別する方法」
セクション8.11.5「外部ロック」
セクション12.16「情報関数」
セクションB.3.2.5「接続が多すぎます」
セクション5.1.12.1「接続インタフェース」
セクション6.1.3「攻撃者に対する MySQL のセキュアな状態の維持」
セクションB.3.2.15「文字セットを初期化できません」
セクション6.2.13「権限変更が有効化される時期」
セクション5.4.4.3「混合形式のバイナリロギング形式」
セクション4.9「環境変数」
セクション1.6「質問またはバグをレポートする方法」
セクション17.2.2.3「起動オプションとレプリケーションチャンネル」
セクションB.3.2.9「通信エラーおよび中止された接続」
セクション16.2.4.2「適切に閉じられなかったテーブルの問題」
セクションB.3.4.5「非トランザクションテーブルのロールバックの失敗」

mysqld mysqld.trace

セクション5.9.1.2「トレースファイルの作成」

mysqld-auto.cnf

セクション4.2.2「プログラムオプションの指定」

mysqld-debug

セクション2.3.4.3「MySQL サーバータイプの選択」
セクション4.3.1「mysqld — MySQL サーバー」
セクション4.3.2「mysqld_safe — MySQL サーバー起動スクリプト」
セクション5.9.1.2「トレースファイルの作成」
セクション2.2「一般的なバイナリを使用した MySQL の Unix/Linux へのインストール」

mysqld.exe

セクション23.2.2.4「NDB Cluster プロセスを Windows サービスとしてインストール」

セクション23.2.2.3 「Windows での NDB Cluster の初期起動」
セクション23.2.2.1 「バイナリリリースから Windows への NDB Cluster のインストール」

mysqld_multi

セクション4.1 「MySQL プログラムの概要」
セクション4.3.4 「mysqld_multi — 複数の MySQL サーバーの管理」
セクション2.5.9 「systemd を使用した MySQL Server の管理」
セクション5.8.3 「Unix 上での複数の MySQL インスタンスの実行」

mysqld_multi.server

セクション2.5.9 「systemd を使用した MySQL Server の管理」

mysqld_safe

セクション5.8 「1 つのマシン上での複数の MySQL インスタンスの実行」
セクション15.21 「InnoDB のトラブルシューティング」
セクションA.10 「MySQL 8.0 FAQ: NDB Cluster」
セクション5.1.15 「MySQL Server でのタイムゾーンのサポート」
セクション2.10.2.1 「MySQL Server の起動時の問題のトラブルシューティング」
セクションB.3.3.6 「MySQL の UNIX ソケットファイルを保護または変更する方法」
セクション2.10.5 「MySQL を自動的に起動および停止する」
セクション2.9.7 「MySQL ソース構成オプション」
セクション4.1 「MySQL プログラムの概要」
セクション4.3.3 「mysql.server — MySQL サーバー起動スクリプト」
セクション4.3.4 「mysqld_multi — 複数の MySQL サーバーの管理」
セクション4.3.2 「mysqld_safe — MySQL サーバー起動スクリプト」
セクション23.5.17.3 「NDB Cluster および MySQL セキュリティー手順」
セクション23.5.7.3 「NDB Cluster データノードのオンラインでの追加: 詳細な例」
セクション13.7.8.8 「RESTART ステートメント」
セクション23.2.1.2 「RPM から NDB Cluster をインストール」
セクション2.5.9 「systemd を使用した MySQL Server の管理」
セクション5.8.3 「Unix 上での複数の MySQL インスタンスの実行」
セクション2.11.6 「Unix/Linux での MySQL バイナリまたはパッケージベースのインストールのアップグレード」
セクション5.4.2 「エラーログ」
セクション4.2.2.6 「オプションのデフォルト、値を想定するオプション、および = 記号」
セクション4.2.2.2 「オプションファイルの使用」
セクション2.10.3 「サーバーのテスト」
セクション5.1.1 「サーバーの構成」
セクション2.10.2 「サーバーの起動」
セクション5.1.7 「サーバーコマンドオプション」
セクション5.1.8 「サーバーシステム変数」
セクションB.3.3.7 「タイムゾーンの問題」
セクション5.9.1.1 「デバッグのための MySQL のコンパイル」
セクション5.4.2.2 「デフォルトのエラーログ保存先の構成」
セクションB.3.2.8 「バケットが大きすぎます」
セクション27.12.14.2 「パフォーマンススキーマ variables_info テーブル」
セクションB.3.2.16 「ファイルが見つからず同様のエラーが発生しました」
セクションB.3.3.1 「ファイル権限の問題」
セクション8.12.3.2 「ラージページのサポートの有効化」
セクション6.1.3 「攻撃者に対する MySQL のセキュアな状態の維持」

mysqldump

セクション10.9.8 「3 バイト Unicode 文字セットと 4 バイト Unicode 文字セット間の変換」
セクション13.1.2 「ALTER DATABASE ステートメント」
セクション11.3.4 「BLOB 型と TEXT 型」
セクション13.1.20 「CREATE TABLE ステートメント」
セクション13.1.21 「CREATE TABLESPACE ステートメント」
セクション13.1.20.5 「FOREIGN KEY の制約」
セクション17.1.3.8 「GTID を操作するストアードファンクションの例」

セクション17.1.5.2 「GTID ベースのレプリケーション用のマルチソースレプリカのプロビジョニング」
セクション13.2.5 「IMPORT TABLE ステートメント」
セクション4.6.2 「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティー」
セクション15.20.7 「InnoDB memcached プラグインとレプリケーション」
セクション8.5.5 「InnoDB テーブルの一括データロード」
セクション15.6.1.1 「InnoDB テーブルの作成」
セクション15.6.1.4 「InnoDB テーブルの移動またはコピー」
セクション15.18.1 「InnoDB バックアップ」
セクション13.2.7 「LOAD DATA ステートメント」
セクション13.2.8 「LOAD XML ステートメント」
セクションA.10 「MySQL 8.0 FAQ: NDB Cluster」
セクション2.11.4 「MySQL 8.0 での変更」
セクション1.3 「MySQL 8.0 の新機能」
セクション5.4 「MySQL Server ログ」
セクション6.2.2 「MySQL で提供される権限」
セクション1.2.2 「MySQL の主な機能」
セクション6.2.21 「MySQL への接続の問題のトラブルシューティング」
セクション1.8.1 「MySQL への貢献者」
セクション4.5.1.1 「mysql クライアントオプション」
セクション2.3.4.7 「MySQL ツールの PATH をカスタマイズする」
セクション2.11.14 「MySQL データベースのほかのマシンへのコピー」
セクション4.1 「MySQL プログラムの概要」
セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティー」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション7.4.1 「mysqldump による SQL フォーマットでのデータのダンプ」
セクション7.4.3 「mysqldump による区切りテキストフォーマットでのデータのダンプ」
セクション7.4.5 「mysqldump のヒント」
mysqldump を使用したデータスナップショットの作成
セクション17.4.1.1 「mysqldump を使用したレプリカのバックアップ」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション23.5.8 「NDB Cluster のオンラインバックアップ」
セクション23.6.5 「NDB Cluster のレプリケーションの準備」
セクション23.5.5 「NDB Cluster のローリング再起動の実行」
NDB Cluster の新しいバージョンへの NDB バックアップの復元
セクション23.1 「NDB Cluster の概要」
セクション23.6.9.2 「NDB Cluster レプリケーションを使用したポイントインタイムリカバリ」
NDB バックアップを以前のバージョンの NDB Cluster に復元
セクション23.4.13 「ndb_import — NDB への CSV データのインポート」
セクション23.5.14 「ndbinfo: NDB Cluster 情報データベース」
セクション23.2.1.2 「RPM から NDB Cluster をインストール」
セクション13.4.1.3 「SET sql_log_bin ステートメント」
セクション6.4.1.2 「SHA-2 プラガブル認証のキャッシュ」
セクション6.4.1.3 「SHA-256 プラガブル認証」
セクション7.4.2 「SQL フォーマットバックアップのリロード」
セクション28.2 「sys スキーマの使用」
セクション2.6 「Unbreakable Linux Network (ULN) を使用した MySQL のインストール」
セクション13.7.4.6 「UNINSTALL PLUGIN ステートメント」
セクション2.11.6 「Unix/Linux での MySQL バイナリまたはパッケージベースのインストールのアップグレード」
セクション4.2.5 「URI 類似文字列またはキーと値のペアを使用したサーバーへの接続」
セクション2.3.4.8 「Windows のサービスとして MySQL を起動する」
セクション12.12 「XML 関数」
セクション4.2.2.2 「オプションファイルの使用」
セクション6.4.1.4 「クライアント側クリアテキストプラガブル認証」
セクション17.1.6.5 「グローバルトランザクション ID システム変数」
セクション4.2.4 「コマンドオプションを使用した MySQL Server への接続」
セクション5.1.11 「サーバー SQL モード」
セクション5.1.8 「サーバーシステム変数」
セクション5.4.6 「サーバーログの保守」
セクション7.4.5.2 「サーバー間でのデータベースのコピー」
セクション7.4.5.3 「ストアードプログラムのダンプ」

セクション17.4.1.3「ソースまたはレプリカを読み取り専用にするによるバックアップ」
セクション23.2.5「テーブルとデータを含む NDB Cluster の例」
セクション15.11.4「テーブルのデフラグ」
セクション2.11.13「テーブルまたはインデックスの再作成または修復」
セクション7.4.5.4「テーブル定義と内容の個別のダンプ」
セクション17.1.2.5「データスナップショットの方法の選択」
セクション14.7「データディクショナリの使用法の違い」
セクション7.4.5.1「データベースのコピーの作成」
セクション7.2「データベースバックアップ方法」
セクション4.6.8.3「バイナリログファイルのバックアップのための mysqlbinlog の使用」
セクション7.3「バックアップおよびリカバリ戦略の例」
第7章「バックアップとリカバリ」
セクション7.1「バックアップとリカバリの種類」
セクション7.4「バックアップへの mysqldump の使用」
セクション7.3.1「バックアップポリシーの確立」
セクション7.3.3「バックアップ戦略サマリー」
セクション17.4.1「バックアップ用にレプリケーションを使用する」
セクション15.5.1「バッファプール」
セクション27.20「パフォーマンススキーマの制約」
セクション27.12.9「パフォーマンススキーマ接続属性テーブル」
セクション15.8.9「ページ構成」
セクション25.9「ビューの制約」
セクション17.1.3.5「フェイルオーバーおよびスケールアウトでの GTID の使用」
セクション17.5.1.34「レプリケーションとトランザクションの非一貫性」
セクション17.5.3「レプリケーションセットアップをアップグレードする」
セクション17.1.2.4「レプリケーションソースのバイナリログ座標の取得」
セクションB.3.4.7「一致する行がない場合の問題の解決」
セクション5.4.1「一般クエリログおよびスロークエリログの出力先の選択」
元のノードより多くのノードへのリストア
セクション7.4.4「区切りテキストフォーマットバックアップのリロード」
セクション4.2.7「接続トランスポートプロトコル」
セクション4.2.8「接続圧縮制御」
既存のデータによるレプリケーションのセットアップ
セクションB.3.4.8「浮動小数点値に関する問題」
セクション17.4.4「異なるソースおよびレプリカのストレージエンジンでのレプリケーションの使用」
セクション17.4.6「異なるレプリカへの異なるデータベースのレプリケート」
セクション1.6「質問またはバグをレポートする方法」
セクション13.1.20.10「非表示カラム」

mysqldump mysql

セクション6.2.21「MySQL への接続の問題のトラブルシューティング」

mysqldump's

セクション2.11.14「MySQL データベースのほかのマシンへのコピー」

mysqldumpslow

セクション4.1「MySQL プログラムの概要」
セクション4.6.9「mysqldumpslow — スロークエリログファイルの要約」
セクション5.4.5「スロークエリログ」

mysqlhotcopy

セクション1.8.1「MySQL への貢献者」

mysqlimport

セクション6.1.6「LOAD DATA LOCAL のセキュリティー上の考慮事項」
セクション13.2.7「LOAD DATA ステートメント」
セクション2.11.14「MySQL データベースのほかのマシンへのコピー」
セクション4.1「MySQL プログラムの概要」

セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション6.4.1.2 「SHA-2 プラガブル認証のキャッシュ」
セクション6.4.1.3 「SHA-256 プラガブル認証」
セクション7.2 「データベースバックアップ方法」
セクション7.1 「バックアップとリカバリの種類」
セクション7.4.4 「区切りテキストフォーマットバックアップのリロード」
セクション4.2.8 「接続圧縮制御」
セクション10.4 「接続文字セットおよび照合順序」

MySQLInstallerConsole.exe

セクション2.3.3.5 「MySQLInstallerConsole リファレンス」

mysqloptimize

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

mysqlpump

セクション13.1.2 「ALTER DATABASE ステートメント」
セクション13.1.21 「CREATE TABLESPACE ステートメント」
セクション17.1.3.8 「GTID を操作するストアドファンクションの例」
セクション4.8.1 「lz4_decompress — mysqlpump LZ4-Compressed 出力の解凍」
セクションA.17 「MySQL 8.0 FAQ : InnoDB 保存データ暗号化」
セクション4.1 「MySQL プログラムの概要」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション6.4.1.2 「SHA-2 プラガブル認証のキャッシュ」
セクション6.4.1.3 「SHA-256 プラガブル認証」
セクション28.2 「sys スキーマの使用」
セクション2.6 「Unbreakable Linux Network (ULN) を使用した MySQL のインストール」
セクション2.11.6 「Unix/Linux での MySQL バイナリまたはパッケージベースのインストールのアップグレード」
セクション4.8.3 「zlib_decompress — mysqlpump ZLIB 圧縮出力の解凍」
セクション14.7 「データディクショナリを使用する方法の違い」
セクション4.2.8 「接続圧縮制御」

mysqlrepair

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

mysqlsh

セクション20.3.1 「MySQL Shell」
セクション20.4.1 「MySQL Shell」
セクション4.1 「MySQL プログラムの概要」
セクション6.4.1.2 「SHA-2 プラガブル認証のキャッシュ」

mysqlshow

セクション2.3.4.9 「MySQL インストールのテスト」
セクション4.1 「MySQL プログラムの概要」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション6.4.1.2 「SHA-2 プラガブル認証のキャッシュ」
セクション6.4.1.3 「SHA-256 プラガブル認証」
セクション13.7.7.14 「SHOW DATABASES ステートメント」
セクション13.7.7.22 「SHOW INDEX ステートメント」
セクション13.7.7.38 「SHOW TABLE STATUS ステートメント」
セクション2.3.6 「Windows でのインストール後の手順」
セクション6.4.1.4 「クライアント側クリアテキストプラガブル認証」
セクション4.2.4 「コマンドオプションを使用した MySQL Server への接続」
セクション2.10.3 「サーバーのテスト」
セクション4.2.8 「接続圧縮制御」
セクション10.4 「接続文字セットおよび照合順序」

mysqlshow db_name

[セクション13.7.7.39「SHOW TABLES ステートメント」](#)

mysqlshow db_name tbl_name

[セクション13.7.7.5「SHOW COLUMNS ステートメント」](#)

mysqlslap

[セクション4.1「MySQL プログラムの概要」](#)

[セクション4.5.8「mysqlslap — ロードエミュレーションクライアント」](#)

[セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#)

[セクション6.4.1.3「SHA-256 プラガブル認証」](#)

[セクション6.4.1.4「クライアント側クリアテキストプラガブル認証」](#)

[セクション15.16.2「パフォーマンススキーマを使用した InnoDB Mutex 待機のモニタリング」](#)

[セクション4.2.8「接続圧縮制御」](#)

[セクション8.13.2「独自のベンチマークの使用」](#)

mysqltest

[セクション1.3「MySQL 8.0 の新機能」](#)

[セクション23.2.1.2「RPM から NDB Cluster をインストール」](#)

[セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#)

[セクション6.4.1.3「SHA-256 プラガブル認証」](#)

[セクション5.1.18「クライアントセッション状態の変更のサーバートラッキング」](#)

[セクション4.2.8「接続圧縮制御」](#)

mysqltest_embedded

[セクション1.3「MySQL 8.0 の新機能」](#)

mysqlxtest

[セクション5.1.14「ネットワークネームスペースのサポート」](#)

N

[\[index top\]](#)

ndbmdtd

[セクション23.3.3.6「NDB Cluster データノードの定義」](#)

ndb_blob_tool

[セクション23.1.4「NDB Cluster の新機能」](#)

[セクション23.4.6「ndb_blob_tool — NDB Cluster テーブルの BLOB および TEXT カラムのチェックおよび修復」](#)

ndb_config

[セクション23.4「NDB Cluster プログラム」](#)

[セクション23.4.7「ndb_config — NDB Cluster 構成情報の抽出」](#)

[セクション23.2.1.2「RPM から NDB Cluster をインストール」](#)

ndb_delete_all

[セクション23.1.4「NDB Cluster の新機能」](#)

[セクション23.4.8「ndb_delete_all — NDB テーブルからのすべての行の削除」](#)

ndb_desc

[セクション26.15「INFORMATION_SCHEMA FILES テーブル」](#)

[セクション26.21「INFORMATION_SCHEMA PARTITIONS テーブル」](#)

[セクション24.2.5「KEY パーティショニング」](#)

[セクション23.5.17.3「NDB Cluster および MySQL セキュリティー手順」](#)

NDB Cluster の MySQL Server オプション

セクション23.1.4 「NDB Cluster の新機能」

セクション23.5.10.1 「NDB Cluster ディスクデータオブジェクト」

セクション23.5.7.3 「NDB Cluster データノードのオンラインでの追加: 詳細な例」

セクション23.4.9 「ndb_desc — NDB テーブルの表示」

セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

セクション13.1.20.11 「NDB_TABLE オプションの設定」

セクション23.5.14.6 「ndbinfo cluster_operations テーブル」

セクション23.5.14.45 「ndbinfo server_operations テーブル」

セクション23.2.1.2 「RPM から NDB Cluster をインストール」

ndb_drop_index

セクション23.4.10 「ndb_drop_index — NDB テーブルからのインデックスの削除」

ndb_drop_table

セクション2.9.7 「MySQL ソース構成オプション」

セクション23.2.7 「NDB Cluster のアップグレードおよびダウングレード」

セクション23.1.4 「NDB Cluster の新機能」

セクション23.4.10 「ndb_drop_index — NDB テーブルからのインデックスの削除」

セクション23.4.11 「ndb_drop_table — NDB テーブルの削除」

ndb_error_reporter

セクション23.4.12 「ndb_error_reporter — NDB エラーレポートユーティリティー」

ndb_import

セクション23.4.13 「ndb_import — NDB への CSV データのインポート」

ndb_index_stat

セクション23.4.14 「ndb_index_stat — NDB インデックス統計ユーティリティー」

ndb_mgm

セクション23.2.1.3 「.deb ファイルを使用した NDB Cluster のインストール」

セクション23.2.1 「Linux での NDB Cluster のインストール」

セクション23.2.1.4 「Linux でのソースからの NDB Cluster の構築」

セクション23.2.1.1 「Linux への NDB Cluster バイナリリリースのインストール」

セクションA.10 「MySQL 8.0 FAQ: NDB Cluster」

第23章 「MySQL NDB Cluster 8.0」

NDB Cluster の MySQL Server オプション

セクション23.5.8 「NDB Cluster のオンラインバックアップ」

セクション23.1.1 「NDB Cluster のコア概念」

セクション23.5.6 「NDB Cluster のシングルユーザーモード」

セクション23.5.17.1 「NDB Cluster のセキュリティおよびネットワークの問題」

セクション23.5.5 「NDB Cluster のローリング再起動の実行」

セクション23.2.4 「NDB Cluster の初期起動」

セクション23.2.6 「NDB Cluster の安全なシャットダウンと再起動」

NDB Cluster の新しいバージョンへの NDB バックアップの復元

セクション23.1.4 「NDB Cluster の新機能」

セクション23.5.7.1 「NDB Cluster データノードのオンラインでの追加: 一般的な問題」

セクション23.5.7.3 「NDB Cluster データノードのオンラインでの追加: 詳細な例」

セクション23.3.3.6 「NDB Cluster データノードの定義」

セクション23.4 「NDB Cluster プログラム」

セクション23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」

セクション23.6.9.2 「NDB Cluster レプリケーションを使用したポイントインタイムリカバリ」

セクション23.5.3.1 「NDB Cluster ロギング管理コマンド」

セクション23.5.1 「NDB Cluster 管理クライアントのコマンド」

セクション23.5.8.2 「NDB Cluster 管理クライアントを使用したバックアップの作成」

セクション23.4.5 「ndb_mgm — NDB Cluster 管理クライアント」

セクション23.4.4 「ndb_mgmd — NDB Cluster 管理サーバーデーモン」
セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」
セクション23.5.14.1 「ndbinfo arbitrator_validity_detail テーブル」
セクション23.5.14.3 「ndbinfo backup_id テーブル」
セクション23.5.14.35 「ndbinfo membership テーブル」
セクション23.5.14.36 「ndbinfo memoryusage テーブル」
セクション23.5.14.38 「ndbinfo nodes テーブル」
セクション23.5.14.55 「ndbinfo transporters テーブル」
セクション23.2.1.2 「RPM から NDB Cluster をインストール」
セクション23.2.2.3 「Windows での NDB Cluster の初期起動」
元のノードより多くのノードへのリストア

ndb_mgm.exe

セクション23.2.2.3 「Windows での NDB Cluster の初期起動」
セクション23.2.2.1 「バイナリリリースから Windows への NDB Cluster のインストール」

ndb_mgmd

セクション23.2.1.3 「.deb ファイルを使用した NDB Cluster のインストール」
セクション23.2.1 「Linux での NDB Cluster のインストール」
セクション23.2.1.4 「Linux でのソースからの NDB Cluster の構築」
セクション23.2.1.1 「Linux への NDB Cluster バイナリリリースのインストール」
セクションA.10 「MySQL 8.0 FAQ: NDB Cluster」
セクション2.9.7 「MySQL ソース構成オプション」
セクション23.2.8.2 「NDB Cluster Auto-Installer の使用」
セクション23.3.3.10 「NDB Cluster TCP/IP 接続」
NDB Cluster の MySQL Server オプション
セクション23.3.1 「NDB Cluster のクイックテスト設定」
セクション23.1.1 「NDB Cluster のコア概念」
セクション23.5.5 「NDB Cluster のローリング再起動の実行」
セクション23.2.4 「NDB Cluster の初期起動」
セクション23.2.6 「NDB Cluster の安全なシャットダウンと再起動」
セクション23.1.4 「NDB Cluster の新機能」
セクション23.3.3.6 「NDB Cluster データノードの定義」
セクション23.1.2 「NDB Cluster ノード、ノードグループ、フラグメントレプリカ、およびパーティション」
セクション23.4 「NDB Cluster プログラム」
セクション23.5.3.1 「NDB Cluster ロギング管理コマンド」
セクション23.3.3.3 「NDB Cluster 接続文字列」
セクション23.3.3.1 「NDB Cluster 構成: 基本例」
セクション23.5.1 「NDB Cluster 管理クライアントのコマンド」
セクション23.3.3.5 「NDB Cluster 管理サーバーの定義」
セクション23.5.4 「NDB Cluster 起動フェーズのサマリー」
セクション23.4.13 「ndb_import — NDB への CSV データのインポート」
セクション23.4.4 「ndb_mgmd — NDB Cluster 管理サーバーデーモン」
セクション23.4.1 「ndbd — NDB Cluster データノードデーモン」
セクション23.2.1.2 「RPM から NDB Cluster をインストール」
セクション23.2.2.3 「Windows での NDB Cluster の初期起動」
セクション23.2.2.1 「バイナリリリースから Windows への NDB Cluster のインストール」

ndb_mgmd.exe

セクション23.2.2.4 「NDB Cluster プロセスを Windows サービスとしてインストール」
セクション23.2.2.3 「Windows での NDB Cluster の初期起動」
セクション23.2.2.1 「バイナリリリースから Windows への NDB Cluster のインストール」

ndb_move_data

セクション23.4.15 「ndb_move_data — NDB データコピーユーティリティ」

ndb_perror

セクション1.3 「MySQL 8.0 の新機能」

[セクション23.1.4 「NDB Cluster の新機能」](#)
[セクション23.4.16 「ndb_perror — NDB エラーメッセージ情報の取得」](#)
[セクション23.5.14.30 「ndbinfo error_messages テーブル」](#)
[セクション4.8.2 「 perror — MySQL エラーメッセージ情報の表示」](#)
[セクションB.2 「エラー情報インタフェース」](#)

ndb_print_backup

[セクション23.4.31 「ndbxfm — NDB Cluster によって作成されたファイルの圧縮、圧縮解除、暗号化、および復号化」](#)

ndb_print_backup_file

[セクション23.1.4 「NDB Cluster の新機能」](#)
[セクション23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#)
[セクション23.5.8.2 「NDB Cluster 管理クライアントを使用したバックアップの作成」](#)
[セクション23.4.17 「ndb_print_backup_file — NDB バックアップファイルの内容の出力」](#)
[セクション23.4.19 「ndb_print_frag_file — NDB フラグメントリストファイルの内容の出力」](#)
[セクション23.4.20 「ndb_print_schema_file — NDB スキーマファイル内容の出力」](#)
[セクション23.4.21 「ndb_print_sys_file — NDB システムファイル内容の出力」](#)
[セクション23.4.22 「ndb_redo_log_reader — クラスタ redo ログの内容の確認および印刷」](#)

ndb_print_file

[セクション23.4.18 「ndb_print_file — NDB ディスクデータファイル内容の出力」](#)

ndb_print_frag_file

[セクション23.4.19 「ndb_print_frag_file — NDB フラグメントリストファイルの内容の出力」](#)

ndb_print_schema_file

[セクション23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#)
[セクション23.4.17 「ndb_print_backup_file — NDB バックアップファイルの内容の出力」](#)
[セクション23.4.18 「ndb_print_file — NDB ディスクデータファイル内容の出力」](#)
[セクション23.4.19 「ndb_print_frag_file — NDB フラグメントリストファイルの内容の出力」](#)
[セクション23.4.20 「ndb_print_schema_file — NDB スキーマファイル内容の出力」](#)
[セクション23.4.21 「ndb_print_sys_file — NDB システムファイル内容の出力」](#)
[セクション23.4.22 「ndb_redo_log_reader — クラスタ redo ログの内容の確認および印刷」](#)

ndb_print_sys_file

[セクション23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#)
[セクション23.4.17 「ndb_print_backup_file — NDB バックアップファイルの内容の出力」](#)
[セクション23.4.18 「ndb_print_file — NDB ディスクデータファイル内容の出力」](#)
[セクション23.4.19 「ndb_print_frag_file — NDB フラグメントリストファイルの内容の出力」](#)
[セクション23.4.20 「ndb_print_schema_file — NDB スキーマファイル内容の出力」](#)
[セクション23.4.21 「ndb_print_sys_file — NDB システムファイル内容の出力」](#)

ndb_redo_log_reader

[セクション23.4.22 「ndb_redo_log_reader — クラスタ redo ログの内容の確認および印刷」](#)

ndb_restore

[セクションA.10 「MySQL 8.0 FAQ: NDB Cluster」](#)
[セクション23.5.8 「NDB Cluster のオンラインバックアップ」](#)
[セクション23.1.1 「NDB Cluster のコア概念」](#)
[セクション23.5.6 「NDB Cluster のシングルユーザーモード」](#)
[セクション23.5.5 「NDB Cluster のローリング再起動の実行」](#)
[NDB Cluster の新しいバージョンへの NDB バックアップの復元](#)
[セクション23.1.4 「NDB Cluster の新機能」](#)
[セクション23.1 「NDB Cluster の概要」](#)
[セクション23.3.3.6 「NDB Cluster データノードの定義」](#)

[セクション23.4「NDB Cluster プログラム」](#)
[セクション23.6.10「NDB Cluster レプリケーション: 双方向および循環レプリケーション」](#)
[セクション23.6.9「NDB Cluster レプリケーションによる NDB Cluster バックアップ」](#)
[セクション23.6.9.2「NDB Cluster レプリケーションを使用したポイントインタイムリカバリ」](#)
[セクション23.6.4「NDB Cluster レプリケーションスキーマおよびテーブル」](#)
[セクション23.5.8.2「NDB Cluster 管理クライアントを使用したバックアップの作成」](#)
[NDB バックアップを以前のバージョンの NDB Cluster に復元](#)
[セクション23.4.23「ndb_restore — NDB Cluster バックアップの復元」](#)
[セクション23.4.31「ndbxfrm — NDB Cluster によって作成されたファイルの圧縮、圧縮解除、暗号化、および復号化」](#)
[セクション7.1「バックアップとリカバリの種類」](#)
[パラレルバックアップのシリアルリストア](#)
[パラレルバックアップのリストア](#)
[セクション23.5.8.5「並列データノードを使用した NDB バックアップの作成」](#)
[元のノードより多くのノードへのリストア](#)
[元のノードより少ないノードへのリストア](#)

ndb_select_all

[セクション23.5.17.3「NDB Cluster および MySQL セキュリティー手順」](#)
[セクション23.1.4「NDB Cluster の新機能」](#)
[セクション23.4.24「ndb_select_all — NDB テーブルの行の出力」](#)
[セクション23.4.27「ndb_show_tables — NDB テーブルのリストの表示」](#)
[セクション23.5.12「NDB_STORED_USER での分散 MySQL 権限」](#)

ndb_select_count

[セクション23.4.25「ndb_select_count — NDB テーブルの行数の出力」](#)

ndb_setup.py

[セクション23.2.8「NDB Cluster Auto-Installer \(サポートされなくなりました\)」](#)
[セクション23.2.8.2「NDB Cluster Auto-Installer の使用」](#)
[セクション23.1.4「NDB Cluster の新機能」](#)
[セクション23.4「NDB Cluster プログラム」](#)
[セクション23.2.8.1「NDB Cluster 自動インストーラの要件」](#)
[セクション23.4.26「ndb_setup.py — NDB Cluster 用ブラウザベースの Auto-Installer の起動 \(非推奨\)」](#)

ndb_show_tables

[セクション23.5.17.3「NDB Cluster および MySQL セキュリティー手順」](#)
[NDB Cluster の MySQL Server オプション](#)
[セクション23.1.4「NDB Cluster の新機能」](#)
[セクション23.4「NDB Cluster プログラム」](#)
[セクション23.4.27「ndb_show_tables — NDB テーブルのリストの表示」](#)
[セクション23.5.14.5「ndbinfo cluster_locks テーブル」](#)
[セクション23.5.14.6「ndbinfo cluster_operations テーブル」](#)
[セクション23.5.14.32「ndbinfo locks_per_fragment テーブル」](#)
[セクション23.5.14.39「ndbinfo operations_per_fragment テーブル」](#)
[セクション23.5.14.44「ndbinfo server_locks テーブル」](#)
[セクション23.5.14.45「ndbinfo server_operations テーブル」](#)

ndb_size.pl

[セクションA.10「MySQL 8.0 FAQ: NDB Cluster」](#)
[NDB Cluster の MySQL Server オプション](#)
[セクション23.4.28「ndb_size.pl — NDBCLUSTER サイズ要件エスティメータ」](#)
[セクション11.7「データ型のストレージ要件」](#)

ndb_top

[セクション23.1.4「NDB Cluster の新機能」](#)
[セクション23.4.29「ndb_top — NDB スレッドの CPU 使用率情報の表示」](#)

ndb_waiter

セクション23.1.4 「NDB Cluster の新機能」

セクション23.4.30 「ndb_waiter — NDB Cluster が特定のステータスに達するまで待機」

ndbd

セクション23.2.1.3 「.deb ファイルを使用した NDB Cluster のインストール」

セクション23.2.1 「Linux での NDB Cluster のインストール」

セクション23.2.1.4 「Linux でのソースからの NDB Cluster の構築」

セクション23.2.1.1 「Linux への NDB Cluster バイナリリリースのインストール」

セクションA.10 「MySQL 8.0 FAQ: NDB Cluster」

セクション23.2.8.2 「NDB Cluster Auto-Installer の使用」

NDB Cluster の MySQL Server オプション

セクション23.2 「NDB Cluster のインストール」

セクション23.3.1 「NDB Cluster のクイックテスト設定」

セクション23.1.1 「NDB Cluster のコア概念」

セクション23.5.5 「NDB Cluster のローリング再起動の実行」

セクション23.2.4 「NDB Cluster の初期起動」

セクション23.2.6 「NDB Cluster の安全なシャットダウンと再起動」

セクション23.3.3.2 「NDB Cluster の推奨開始構成」

セクション23.1.4 「NDB Cluster の新機能」

セクション23.5 「NDB Cluster の管理」

セクション23.5.7.2 「NDB Cluster データノードのオンラインでの追加: 基本手順」

セクション23.5.7.3 「NDB Cluster データノードのオンラインでの追加: 詳細な例」

セクション23.3.3.6 「NDB Cluster データノードの定義」

セクション23.3.2.1 「NDB Cluster データノード構成パラメータ」

セクション23.1.2 「NDB Cluster ノード、ノードグループ、フラグメントレプリカ、およびパーティション」

セクション23.4 「NDB Cluster プログラム」

セクション23.6.9 「NDB Cluster レプリケーションによる NDB Cluster バックアップ」

セクション23.6.9.2 「NDB Cluster レプリケーションを使用したポイントインタイムリカバリ」

セクション23.3.3.1 「NDB Cluster 構成: 基本例」

セクション23.3.2 「NDB Cluster 構成パラメータ、オプション、および変数の概要」

セクション23.5.3.3 「NDB Cluster 管理クライアントでの CLUSTERLOG STATISTICS の使用」

セクション23.5.1 「NDB Cluster 管理クライアントのコマンド」

セクション23.5.4 「NDB Cluster 起動フェーズのサマリー」

セクション23.4.4 「ndb_mgmd — NDB Cluster 管理サーバーデーモン」

セクション23.4.30 「ndb_waiter — NDB Cluster が特定のステータスに達するまで待機」

セクション23.4.1 「ndbd — NDB Cluster データノードデーモン」

セクション23.5.14.38 「ndbinfo nodes テーブル」

セクション23.4.3 「ndbmtd — NDB Cluster データノードデーモン (マルチスレッド)」

セクション23.2.1.2 「RPM から NDB Cluster をインストール」

セクション23.2.2.1 「バイナリリリースから Windows への NDB Cluster のインストール」

セクション23.5.8.5 「並列データノードを使用した NDB バックアップの作成」

ndbd.exe

セクション23.2.2.4 「NDB Cluster プロセスを Windows サービスとしてインストール」

セクション23.2.2.3 「Windows での NDB Cluster の初期起動」

セクション23.2.2.1 「バイナリリリースから Windows への NDB Cluster のインストール」

ndbinfo_select_all

セクション23.4.2 「ndbinfo_select_all — ndbinfo テーブルからの選択」

ndbmtd

セクション23.2.1 「Linux での NDB Cluster のインストール」

セクション23.2.1.4 「Linux でのソースからの NDB Cluster の構築」

セクション23.2.1.1 「Linux への NDB Cluster バイナリリリースのインストール」

セクションA.10 「MySQL 8.0 FAQ: NDB Cluster」

セクション2.9.7 「MySQL ソース構成オプション」

セクション23.2.8.2 「NDB Cluster Auto-Installer の使用」

セクション23.1.1 「NDB Cluster のコア概念」
セクション23.5.5 「NDB Cluster のローリング再起動の実行」
セクション23.2.6 「NDB Cluster の安全なシャットダウンと再起動」
セクション23.3.3.2 「NDB Cluster の推奨開始構成」
セクション23.1.4 「NDB Cluster の新機能」
セクション23.5.7.2 「NDB Cluster データノードのオンラインでの追加: 基本手順」
セクション23.5.7.3 「NDB Cluster データノードのオンラインでの追加: 詳細な例」
セクション23.3.3.6 「NDB Cluster データノードの定義」
セクション23.3.2.1 「NDB Cluster データノード構成パラメータ」
セクション23.1.2 「NDB Cluster ノード、ノードグループ、フラグメントレプリカ、およびパーティション」
セクション23.4 「NDB Cluster プログラム」
セクション23.6.9.2 「NDB Cluster レプリケーションを使用したポイントインタイムリカバリ」
セクション23.4.4 「ndb_mgmd — NDB Cluster 管理サーバーデーモン」
セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」
セクション23.4.1 「ndbd — NDB Cluster データノードデーモン」
セクション23.5.14.38 「ndbinfo nodes テーブル」
セクション23.5.14.42 「ndbinfo resources テーブル」
セクション23.4.3 「ndbmtdd — NDB Cluster データノードデーモン (マルチスレッド)」
セクション23.2.1.2 「RPM から NDB Cluster をインストール」
セクション23.4.23.3 「パラレルで作成されたバックアップからのリストア」
セクション23.5.8.5 「並列データノードを使用した NDB バックアップの作成」
元のノードより少ないノードへのリストア

ndbmtdd.exe

セクション23.2.2.4 「NDB Cluster プロセスを Windows サービスとしてインストール」
セクション23.2.2.3 「Windows での NDB Cluster の初期起動」
セクション23.2.2.1 「バイナリリリースから Windows への NDB Cluster のインストール」

ndbxfm

セクション23.1.4 「NDB Cluster の新機能」
セクション23.5.8.2 「NDB Cluster 管理クライアントを使用したバックアップの作成」
セクション23.4.31 「ndbxfm — NDB Cluster によって作成されたファイルの圧縮、圧縮解除、暗号化、および復号化」

NET

セクション23.2.2.4 「NDB Cluster プロセスを Windows サービスとしてインストール」
セクション23.3.4.8 「Windows のサービスとして MySQL を起動する」

NET START

セクション23.2.2.4 「NDB Cluster プロセスを Windows サービスとしてインストール」
セクション5.8.2.2 「Windows サービスとして複数の MySQL インスタンスの起動」

NET START mysqld_service_name

セクション2.3.5 「Microsoft Windows MySQL Server インストールのトラブルシューティング」
セクション2.3.4.8 「Windows のサービスとして MySQL を起動する」
セクション2.11.10 「Windows 上の MySQL をアップグレードする」

NET STOP

セクション23.2.2.4 「NDB Cluster プロセスを Windows サービスとしてインストール」
セクション5.8.2.2 「Windows サービスとして複数の MySQL インスタンスの起動」

NET STOP mysqld_service_name

セクション2.3.4.8 「Windows のサービスとして MySQL を起動する」
セクション2.11.10 「Windows 上の MySQL をアップグレードする」

NET STOP service_name

セクション23.2.6 「NDB Cluster の安全なシャットダウンと再起動」

nslookup

セクション1.3 「MySQL 8.0 の新機能」

numactl

セクション23.3.3.6 「NDB Cluster データノードの定義」

O

[\[index top\]](#)

openssl

セクション6.4.4.4 「keyring_okv KMIP プラグインの使用」
セクション6.3.3.1 「MySQL を使用した SSL および RSA 証明書とキーの作成」
セクション4.4.3 「mysql_ssl_rsa_setup — SSL/RSA ファイルの作成」
セクション6.3.3.3 「openssl を使用した RSA キーの作成」
セクション6.3.3.2 「openssl を使用した SSL 証明書およびキーの作成」
セクション6.3.3 「SSL および RSA 証明書とキーの作成」
セクション5.1.9.4 「永続的で永続的に制限されないシステム変数」
セクション6.4.5.5 「監査ロギング特性の構成」

openssl md5 package_name

セクション2.1.4.1 「MD5 チェックサムの確認」

openssl zlib

セクション2.9.7 「MySQL ソース構成オプション」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション4.8.3 「zlib_decompress — mysqlpump ZLIB 圧縮出力の解凍」

P

[\[index top\]](#)

patchelf

セクション2.9.7 「MySQL ソース構成オプション」

perf

セクション27.12.19.10 「スレッドテーブル」

perror

セクション7.6.3 「MyISAM テーブルの修復方法」
セクションA.10 「MySQL 8.0 FAQ: NDB Cluster」
セクション1.3 「MySQL 8.0 の新機能」
セクション4.1 「MySQL プログラムの概要」
セクション23.1.4 「NDB Cluster の新機能」
セクション23.4.16 「ndb_perror — NDB エラーメッセージ情報の取得」
セクション23.5.14.30 「ndbinfo error_messages テーブル」
セクション4.8.2 「pererror — MySQL エラーメッセージ情報の表示」
セクションB.2 「エラー情報インタフェース」
セクションB.3.2.16 「ファイルが見つからず同様のエラーが発生しました」
セクションB.3.2.11 「ファイルを作成/書き込みできない」
セクション5.4.2.6 「ルールベースのエラーログのフィルタリング (log_filter_dragnet)」

pfexec

セクション2.2 「一般的なバイナリを使用した MySQL の Unix/Linux へのインストール」

PGP

[セクション2.1.4.2 「GnuPG を使用した署名確認」](#)

ping6

[セクション5.1.13.5 「ブローカからの IPv6 アドレスの入手」](#)

pkg-config

[セクション2.9.7 「MySQL ソース構成オプション」](#)

[セクション4.7.1 「mysql_config — クライアントのコンパイル用オプションの表示」](#)

[セクション4.9 「環境変数」](#)

pkgadd

[セクション2.7.1 「Solaris PKG を使用して Solaris に MySQL をインストールする」](#)

pkgrm

[セクション2.7.1 「Solaris PKG を使用して Solaris に MySQL をインストールする」](#)

ppm

[セクション2.13 「Perl のインストールに関する注釈」](#)

ps

[セクション2.10.2.1 「MySQL Server の起動時の問題のトラブルシューティング」](#)

[セクション8.12.3.1 「MySQL のメモリーの使用方法」](#)

[セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」](#)

[セクション6.7 「SELinux」](#)

[セクション6.2.14 「アカウントパスワードの割り当て」](#)

[セクション4.2.4 「コマンドオプションを使用した MySQL Server への接続」](#)

[セクション27.12.19.10 「スレッドテーブル」](#)

[セクション6.1.2.1 「パスワードセキュリティのためのエンドユーザーガイドライン」](#)

[セクションB.3.1 「問題の原因を判別する方法」](#)

[セクション4.9 「環境変数」](#)

ps xa | grep mysqld

[セクションB.3.2.2 「\[ローカルの\] MySQL サーバーに接続できません」](#)

R

[\[index top\]](#)

rename

[セクション5.4.2.10 「エラーログファイルのフラッシュおよび名前変更」](#)

[セクション5.4.6 「サーバーログの保守」](#)

[セクション5.4.3 「一般クエリログ」](#)

resolve_stack_dump

[セクション1.3 「MySQL 8.0 の新機能」](#)

resolveip

[セクション1.3 「MySQL 8.0 の新機能」](#)

restart

[セクション2.5.1 「MySQL Yum リポジトリを使用して MySQL を Linux にインストールする」](#)

[セクション2.5.4 「Oracle の RPM パッケージを使用した Linux への MySQL のインストール」](#)

rm

[セクション13.4.1.1「PURGE BINARY LOGS ステートメント」](#)

rpm

[セクション2.5.4「Oracle の RPM パッケージを使用した Linux への MySQL のインストール」](#)

[セクション2.1.4.4「RPM を使用した署名確認」](#)

[セクション2.9.4「標準ソース配布を使用して MySQL をインストールする」](#)

rpmbuild

[セクション2.5.4「Oracle の RPM パッケージを使用した Linux への MySQL のインストール」](#)

[セクション2.9.2「ソースインストールの前提条件」](#)

[セクション2.9.4「標準ソース配布を使用して MySQL をインストールする」](#)

rsync

[セクション7.1「バックアップとリカバリの種類」](#)

[セクション17.1.2.8「レプリケーション環境へのレプリカの追加」](#)

[ローデータファイルを使用したデータスナップショットの作成](#)

S

[\[index top\]](#)

SC

[セクション2.3.4.8「Windows のサービスとして MySQL を起動する」](#)

SC DELETE

[セクション23.2.2.4「NDB Cluster プロセスを Windows サービスとしてインストール」](#)

SC DELETE mysql_d_service_name

[セクション2.3.4.8「Windows のサービスとして MySQL を起動する」](#)

[セクション5.8.2.2「Windows サービスとして複数の MySQL インスタンスの起動」](#)

SC DELETE service_name

[セクション23.2.2.4「NDB Cluster プロセスを Windows サービスとしてインストール」](#)

SC START

[セクション23.5.5「NDB Cluster のローリング再起動の実行」](#)

[セクション23.2.2.4「NDB Cluster プロセスを Windows サービスとしてインストール」](#)

[セクション5.8.2.2「Windows サービスとして複数の MySQL インスタンスの起動」](#)

SC START mysql_d_service_name

[セクション2.3.5「Microsoft Windows MySQL Server インストールのトラブルシューティング」](#)

[セクション2.11.10「Windows 上の MySQL をアップグレードする」](#)

sc start mysql_d_service_name

[セクション2.3.4.8「Windows のサービスとして MySQL を起動する」](#)

SC STOP

[セクション23.5.5「NDB Cluster のローリング再起動の実行」](#)

[セクション23.2.2.4「NDB Cluster プロセスを Windows サービスとしてインストール」](#)

[セクション5.8.2.2「Windows サービスとして複数の MySQL インスタンスの起動」](#)

SC STOP mysql_d_service_name

[セクション2.3.4.8「Windows のサービスとして MySQL を起動する」](#)

sc stop mysql_service_name

セクション2.3.4.8 「Windows のサービスとして MySQL を起動する」

SC STOP service_name

セクション23.2.6 「NDB Cluster の安全なシャットダウンと再起動」

scp

セクション7.1 「バックアップとリカバリの種類」

ローデータファイルを使用したデータスナップショットの作成

sed

セクション3.3.4.7 「パターンマッチング」

SELECT

セクション23.2.5 「テーブルとデータを含む NDB Cluster の例」

semanage

セクション6.7.6 「SELinux のトラブルシューティング」

semodule

セクション6.7.3 「MySQL Server SELinux ポリシー」

service

セクション2.5.1 「MySQL Yum リポジトリを使用して MySQL を Linux にインストールする」

セクション2.5.4 「Oracle の RPM パッケージを使用した Linux への MySQL のインストール」

セクション2.5.9 「systemd を使用した MySQL Server の管理」

Service Control Manager

セクション2.3.4.8 「Windows のサービスとして MySQL を起動する」

Services

セクション23.2.2.4 「NDB Cluster プロセスを Windows サービスとしてインストール」

セクション2.3.4.8 「Windows のサービスとして MySQL を起動する」

sestatus

セクション6.7.1 「SELinux が有効かどうかの確認」

セクション6.7.2 「SELinux モードの変更」

セクション18.10 「よくある質問」

setcap

セクション5.1.14 「ネットワークネームスペースのサポート」

セクション5.1.16 「リソースグループ」

setenforce

セクション6.7.6 「SELinux のトラブルシューティング」

セクション6.7.2 「SELinux モードの変更」

setenv

セクション4.2.9 「環境変数の設定」

setup.bat

セクション23.2.8.2 「NDB Cluster Auto-Installer の使用」

sh

セクション4.2.1 「MySQL プログラムの起動」

セクション1.1「このマニュアルについて」
セクションB.3.2.16「ファイルが見つからず同様のエラーが発生しました」
セクション4.2.9「環境変数の設定」

SHOW

セクション23.3.1「NDB Cluster のクイックテスト設定」

SHOW ERRORS

セクションA.10「MySQL 8.0 FAQ: NDB Cluster」

SHOW WARNINGS

セクションA.10「MySQL 8.0 FAQ: NDB Cluster」

sleep

セクション4.3.2「mysqld_safe — MySQL サーバー起動スクリプト」

ssh

セクション23.5.17.1「NDB Cluster のセキュリティーおよびネットワークの問題」

start

セクション2.5.1「MySQL Yum リポジトリを使用して MySQL を Linux にインストールする」
セクション2.5.4「Oracle の RPM パッケージを使用した Linux への MySQL のインストール」

Start>Run>cmd.exe

セクション6.3.3.2「openssl を使用した SSL 証明書およびキーの作成」

status

セクション2.5.1「MySQL Yum リポジトリを使用して MySQL を Linux にインストールする」
セクション2.5.4「Oracle の RPM パッケージを使用した Linux への MySQL のインストール」

stop

セクション2.5.1「MySQL Yum リポジトリを使用して MySQL を Linux にインストールする」
セクション2.5.4「Oracle の RPM パッケージを使用した Linux への MySQL のインストール」

strings

セクション6.1.1「セキュリティーガイドライン」

su root

セクション23.2.1.1「Linux への NDB Cluster バイナリリリースのインストール」

sudo

セクション23.2.1.1「Linux への NDB Cluster バイナリリリースのインストール」
セクション5.1.14「ネットワークネームスペースのサポート」
セクション5.1.16「リソースグループ」
セクション2.2「一般的なバイナリを使用した MySQL の Unix/Linux へのインストール」

sudo ip

セクション5.1.14「ネットワークネームスペースのサポート」

sudo setcap

セクション5.1.14「ネットワークネームスペースのサポート」

systemctl

セクション2.5.1「MySQL Yum リポジトリを使用して MySQL を Linux にインストールする」

[セクション2.5.4 「Oracle の RPM パッケージを使用した Linux への MySQL のインストール」](#)
[セクション2.5.9 「systemd を使用した MySQL Server の管理」](#)
[セクション2.5.7 「ネイティブソフトウェアリポジトリから MySQL を Linux にインストールする」](#)

T

[\[index top\]](#)

tar

[セクション6.4.4.4 「keyring_okv KMIP プラグインの使用」](#)
[セクション2.4 「macOS への MySQL のインストール」](#)
[セクション2.5.4 「Oracle の RPM パッケージを使用した Linux への MySQL のインストール」](#)
[セクション23.2.1.2 「RPM から NDB Cluster をインストール」](#)
[セクション2.7.1 「Solaris PKG を使用して Solaris に MySQL をインストールする」](#)
[セクション2.7 「Solaris への MySQL のインストール」](#)
[セクション2.13.1 「Unix に Perl をインストールする」](#)
[セクション8.12.2.1 「Unix 上のデータベースへのシンボリックリンクの使用」](#)
[セクション2.1.2 「インストールする MySQL のバージョンと配布の選択」](#)
[セクション2.9.1 「ソースのインストール方法」](#)
[セクション2.9.2 「ソースインストールの前提条件」](#)
[セクション3.3 「データベースの作成と使用」](#)
[セクション7.1 「バックアップとリカバリの種類」](#)
[セクション17.4.1.2 「レプリカからの RAW データのバックアップ」](#)
[セクション17.1.2.8 「レプリケーション環境へのレプリカの追加」](#)
[ローデータファイルを使用したデータスナップショットの作成](#)
[セクション2.2 「一般的なバイナリを使用した MySQL の Unix/Linux へのインストール」](#)
[セクション2.9.4 「標準ソース配布を使用して MySQL をインストールする」](#)
[セクション1.6 「質問またはバグをレポートする方法」](#)

tcpdump

[セクション6.1.1 「セキュリティガイドライン」](#)

tcsh

[セクション2.4.1 「macOS への MySQL のインストールに関する一般的なノート」](#)
[セクション4.2.1 「MySQL プログラムの起動」](#)
[セクション1.1 「このマニュアルについて」](#)
[セクションB.3.2.16 「ファイルが見つからず同様のエラーが発生しました」](#)
[セクション4.2.9 「環境変数の設定」](#)

tee

[セクション4.5.1.2 「mysql クライアントコマンド」](#)

telnet

[セクション15.20.2 「InnoDB memcached のアーキテクチャー」](#)
[セクション15.20.3 「InnoDB memcached プラグインの設定」](#)
[セクション6.1.1 「セキュリティガイドライン」](#)

Terminal

[セクション2.4 「macOS への MySQL のインストール」](#)

top

[セクション23.5.14.41 「ndbinfo processes テーブル」](#)
[セクションB.3.1 「問題の原因を判別する方法」](#)

U

[\[index top\]](#)

ulimit

[セクション4.3.2 「mysqld_safe — MySQL サーバー起動スクリプト」](#)
[セクション23.3.3.6 「NDB Cluster データノードの定義」](#)
[セクション23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」](#)
[セクション5.1.7 「サーバーコマンドオプション」](#)
[セクション5.1.8 「サーバーシステム変数」](#)
[セクションB.3.2.8 「バケットが大きすぎます」](#)
[セクションB.3.2.16 「ファイルが見つからず同様のエラーが発生しました」](#)
[セクション8.12.3.2 「ラージページのサポートの有効化」](#)

unix_chkpwd

[セクション6.4.1.5 「PAM プラガブル認証」](#)

update-rc.d

[セクション23.2.1.1 「Linux への NDB Cluster バイナリリリースのインストール」](#)

useradd

[セクション23.2.1.1 「Linux への NDB Cluster バイナリリリースのインストール」](#)
[セクション2.7 「Solaris への MySQL のインストール」](#)
[セクション2.2 「一般的なバイナリを使用した MySQL の Unix/Linux へのインストール」](#)

V

[\[index top\]](#)

vault

[セクション6.4.4.6 「HashiCorp Vault キーリングプラグインの使用」](#)

vault server

[セクション6.4.4.6 「HashiCorp Vault キーリングプラグインの使用」](#)

vi

[セクション4.5.1.2 「mysql クライアントコマンド」](#)
[セクション23.2.3 「NDB Cluster の初期構成」](#)
[セクション3.3.4.7 「パターンマッチング」](#)

W

[\[index top\]](#)

watch

[セクション28.4.4.25 「statement_performance_analyzer\(\) プロシージャ」](#)

WinDbg

[セクション5.9.1.3 「WER と PDB を使用した Windows クラッシュダンプの作成」](#)

windbg.exe

[セクション5.9.1.3 「WER と PDB を使用した Windows クラッシュダンプの作成」](#)

winMd5Sum

[セクション2.1.4.1 「MD5 チェックサムの確認」](#)

WinZip

[セクション2.9.2 「ソースインストールの前提条件」](#)
[セクション17.4.1.2 「レプリカからの RAW データのバックアップ」](#)

[セクション2.9.4「標準ソース配布を使用して MySQL をインストールする」](#)

WordPad

[セクション13.2.7「LOAD DATA ステートメント」](#)

X

[\[index top\]](#)

XZ

[セクション2.2「一般的なバイナリを使用した MySQL の Unix/Linux へのインストール」](#)

Y

[\[index top\]](#)

yacc

[セクション2.9.8「MySQL のコンパイルに関する問題」](#)

[セクション9.3「キーワードと予約語」](#)

yum

[セクション2.5.1「MySQL Yum リポジトリを使用して MySQL を Linux にインストールする」](#)

[セクション2.11.7「MySQL Yum リポジトリを使用する MySQL のアップグレード」](#)

[セクション2.5.4「Oracle の RPM パッケージを使用した Linux への MySQL のインストール」](#)

[セクション23.2.1.2「RPM から NDB Cluster をインストール」](#)

[セクション2.5.7「ネイティブソフトウェアリポジトリから MySQL を Linux にインストールする」](#)

yum install

[セクション2.5.4「Oracle の RPM パッケージを使用した Linux への MySQL のインストール」](#)

yum update

[セクション2.5.1「MySQL Yum リポジトリを使用して MySQL を Linux にインストールする」](#)

yum-config-manager

[セクション2.5.1「MySQL Yum リポジトリを使用して MySQL を Linux にインストールする」](#)

Z

[\[index top\]](#)

zip

[ローデータファイルを使用したデータスナップショットの作成](#)

[セクション1.6「質問またはバグをレポートする方法」](#)

zlib_decompress

[セクション4.8.1「lz4_decompress — mysqlpump LZ4-Compressed 出力の解凍」](#)

[セクション2.9.7「MySQL ソース構成オプション」](#)

[セクション4.1「MySQL プログラムの概要」](#)

[セクション4.5.6「mysqlpump — データベースバックアッププログラム」](#)

[セクション4.8.3「zlib_decompress — mysqlpump ZLIB 圧縮出力の解凍」](#)

zsh

[セクション4.2.9「環境変数の設定」](#)

zypper

[セクション2.5.4「Oracle の RPM パッケージを使用した Linux への MySQL のインストール」](#)

関数の索引

[記号](#) | [A](#) | [B](#) | [C](#) | [D](#) | [E](#) | [F](#) | [G](#) | [H](#) | [I](#) | [J](#) | [K](#) | [L](#) | [M](#) | [N](#) | [O](#) | [P](#) | [Q](#) | [R](#) | [S](#) | [T](#) | [U](#) | [V](#) | [W](#) | [Y](#)

記号

[\[index top\]](#)

%

[セクション1.7.1 「標準 SQL に対する MySQL 拡張機能」](#)

A

[\[index top\]](#)

ABS()

[セクション8.9.6 「オプティマイザ統計」](#)

[セクション13.7.4.1 「ユーザー定義関数用の CREATE FUNCTION ステートメント」](#)

[セクション12.6.2 「数学関数」](#)

[セクション24.6.3 「関数に関連するパーティショニング制限」](#)

ACOS()

[セクション12.6.2 「数学関数」](#)

ADDDATE()

[セクション12.7 「日付および時間関数」](#)

addslashes()

[セクション6.1.7 「クライアントプログラミングのセキュリティーガイドライン」](#)

ADDTIME()

[セクション12.7 「日付および時間関数」](#)

AES_DECRYPT()

[セクション1.3 「MySQL 8.0 の新機能」](#)

[セクション6.6.4 「MySQL Enterprise Encryption ユーザー定義関数の説明」](#)

[セクション5.1.8 「サーバーシステム変数」](#)

[セクション12.14 「暗号化関数と圧縮関数」](#)

AES_ENCRYPT()

[セクション1.3 「MySQL 8.0 の新機能」](#)

[セクション6.6.4 「MySQL Enterprise Encryption ユーザー定義関数の説明」](#)

[セクション5.1.8 「サーバーシステム変数」](#)

[セクション12.14 「暗号化関数と圧縮関数」](#)

ANY_VALUE()

[セクション12.20.3 「MySQL での GROUP BY の処理」](#)

[セクション12.24 「その他の関数」](#)

ASCII()

[セクション13.8.3 「HELP ステートメント」](#)

[セクション12.8 「文字列関数および演算子」](#)

ASIN()

[セクション12.6.2 「数学関数」](#)

asymmetric_decrypt()

セクション6.6.4 「MySQL Enterprise Encryption ユーザー定義関数の説明」

asymmetric_derive()

セクション6.6.4 「MySQL Enterprise Encryption ユーザー定義関数の説明」

asymmetric_encrypt()

セクション6.6.4 「MySQL Enterprise Encryption ユーザー定義関数の説明」

asymmetric_sign()

セクション6.6.4 「MySQL Enterprise Encryption ユーザー定義関数の説明」

asymmetric_verify()

セクション6.6.4 「MySQL Enterprise Encryption ユーザー定義関数の説明」

asynchronous_connection_failover_add_managed

セクション13.4.2.1 「CHANGE MASTER TO ステートメント」

セクション13.4.2.3 「CHANGE REPLICATION SOURCE TO ステートメント」

セクション27.12.11.3 「replication_asynchronous_connection_failover テーブル」

セクション17.4.9 「非同期接続フェイルオーバーによるソースの切替え」

asynchronous_connection_failover_add_managed()

セクション13.4.2.11 「ソースリストを構成する関数」

asynchronous_connection_failover_add_source

セクション13.4.2.1 「CHANGE MASTER TO ステートメント」

セクション13.4.2.3 「CHANGE REPLICATION SOURCE TO ステートメント」

セクション27.12.11.3 「replication_asynchronous_connection_failover テーブル」

セクション17.4.9 「非同期接続フェイルオーバーによるソースの切替え」

asynchronous_connection_failover_add_source()

セクション13.4.2.11 「ソースリストを構成する関数」

asynchronous_connection_failover_delete_managed

セクション13.4.2.1 「CHANGE MASTER TO ステートメント」

セクション13.4.2.3 「CHANGE REPLICATION SOURCE TO ステートメント」

セクション27.12.11.3 「replication_asynchronous_connection_failover テーブル」

セクション17.4.9 「非同期接続フェイルオーバーによるソースの切替え」

asynchronous_connection_failover_delete_managed()

セクション13.4.2.11 「ソースリストを構成する関数」

asynchronous_connection_failover_delete_source

セクション13.4.2.1 「CHANGE MASTER TO ステートメント」

セクション13.4.2.3 「CHANGE REPLICATION SOURCE TO ステートメント」

セクション27.12.11.3 「replication_asynchronous_connection_failover テーブル」

セクション17.4.9 「非同期接続フェイルオーバーによるソースの切替え」

asynchronous_connection_failover_delete_source()

セクション13.4.2.11 「ソースリストを構成する関数」

ATAN()

セクション12.6.2 「数学関数」

ATAN2()

セクション12.6.2「数学関数」

audit_api_message_emit_udf()

セクション6.4.6「監査メッセージコンポーネント」
セクション6.4.5.8「監査ログフィルタ定義の書込み」

audit_log_encryption_password_get()

セクション6.4.5.5「監査ロギング特性の構成」
セクション6.4.5.10「監査ログ参照」

audit_log_encryption_password_set()

セクション6.4.5.5「監査ロギング特性の構成」
セクション6.4.5.10「監査ログ参照」

audit_log_filter_flush()

セクション6.4.5.7「監査ログのフィルタリング」
セクション6.4.5.10「監査ログ参照」

audit_log_filter_remove_filter()

セクション6.4.5.7「監査ログのフィルタリング」
セクション6.4.5.10「監査ログ参照」

audit_log_filter_remove_user()

セクション6.4.5.7「監査ログのフィルタリング」
セクション6.4.5.10「監査ログ参照」

audit_log_filter_set_filter()

セクション6.4.5.7「監査ログのフィルタリング」
セクション6.4.5.10「監査ログ参照」

audit_log_filter_set_user()

セクション6.4.5.7「監査ログのフィルタリング」
セクション6.4.5.10「監査ログ参照」

audit_log_read()

セクション6.4.5.6「監査ログファイルの読取り」
セクション6.4.5.10「監査ログ参照」

audit_log_read_bookmark()

セクション6.4.5.6「監査ログファイルの読取り」
セクション6.4.5.10「監査ログ参照」

AVG()

セクション12.20.1「集計関数の説明」

AVG()

セクション11.3.5「ENUM 型」
セクション8.2.1.17「GROUP BY の最適化」
セクション1.3「MySQL 8.0 の新機能」
セクション1.2.2「MySQL の主な機能」
セクション11.3.6「SET 型」
セクション12.21.3「ウィンドウ機能フレーム仕様」
セクション11.2.1「日時データ型の構文」
セクション12.20.1「集計関数の説明」

B

[\[index top\]](#)

BENCHMARK()

[セクション13.2.11.8「導出テーブル」](#)
[セクション8.13.1「式と関数の速度の測定」](#)
[セクション12.16「情報関数」](#)

BIN()

[セクション9.1.5「ビット値リテラル」](#)
[セクション12.8「文字列関数および演算子」](#)

BIN_TO_UUID()

[セクション12.24「その他の関数」](#)
[セクション12.8.3「関数結果の文字セットと照合順序」](#)

BIT_AND()

[セクション12.13「ビット関数と演算子」](#)
[セクション1.7.1「標準 SQL に対する MySQL 拡張機能」](#)
[セクション12.20.1「集計関数の説明」](#)

BIT_COUNT()

[セクション12.13「ビット関数と演算子」](#)
[セクション1.7.1「標準 SQL に対する MySQL 拡張機能」](#)

BIT_LENGTH()

[セクション12.8「文字列関数および演算子」](#)

BIT_OR()

[セクション12.13「ビット関数と演算子」](#)
[セクション1.7.1「標準 SQL に対する MySQL 拡張機能」](#)
[セクション12.20.1「集計関数の説明」](#)

BIT_XOR()

[セクション12.13「ビット関数と演算子」](#)
[セクション1.7.1「標準 SQL に対する MySQL 拡張機能」](#)
[セクション12.20.1「集計関数の説明」](#)

C

[\[index top\]](#)

CAN_ACCESS_COLUMN()

[セクション12.23「内部関数」](#)

CAN_ACCESS_DATABASE()

[セクション12.23「内部関数」](#)

CAN_ACCESS_TABLE()

[セクション12.23「内部関数」](#)

CAN_ACCESS_USER()

[セクション12.23「内部関数」](#)

CAN_ACCESS_VIEW()

セクション12.23「内部関数」

CAST()

セクション9.1.4「16進数リテラル」
セクション13.1.15「CREATE INDEX ステートメント」
セクション11.2.2「DATE、DATETIME、および TIMESTAMP 型」
セクション11.5「JSON データ型」
セクション12.18.2「JSON 値を作成する関数」
セクション12.18.3「JSON 値を検索する関数」
セクション1.3「MySQL 8.0 の新機能」
セクション1.7.2「MySQL と標準 SQL との違い」
セクション13.5.1「PREPARE ステートメント」
セクション13.2.15「WITH (共通テーブル式)」
セクション12.11「キャスト関数と演算子」
セクション9.1.5「ビット値リテラル」
セクション12.13「ビット関数と演算子」
セクション9.4「ユーザー定義変数」
セクション9.5「式」
セクション12.3「式評価での型変換」
セクション12.7「日付および時間関数」
セクション11.2.7「日付と時間型間での変換」
セクション12.4.2「比較関数と演算子」
セクション12.8.3「関数結果の文字セットと照合順序」

CEIL()

セクション12.6.2「数学関数」

CEILING()

セクション24.2.4.1「LINEAR HASH パーティショニング」
セクション12.6.2「数学関数」
セクション24.6.3「関数に関連するパーティショニング制限」

CHAR()

セクション4.5.1.1「mysql クライアントオプション」
セクション12.8「文字列関数および演算子」
セクション1.7.1「標準 SQL に対する MySQL 拡張機能」

CHAR_LENGTH()

セクション10.10.1「Unicode 文字セット」
セクション12.8「文字列関数および演算子」

CHARACTER_LENGTH()

セクション12.8「文字列関数および演算子」

CHARSET()

セクション12.16「情報関数」
セクション12.8.3「関数結果の文字セットと照合順序」

COALESCE

セクション13.5.1「PREPARE ステートメント」

COALESCE()

セクション13.2.10.2「JOIN 句」
セクション13.5.1「PREPARE ステートメント」

セクション13.2.15 「WITH (共通テーブル式)」
セクション12.4.2 「比較関数と演算子」

COERCIBILITY()

セクション10.8.4 「式での照合の強制性」
セクション12.16 「情報関数」

COLLATION()

セクション12.16 「情報関数」
セクションB.3.4.1 「文字列検索での大文字/小文字の区別」
セクション12.8.3 「関数結果の文字セットと照合順序」

COMPRESS()

セクション2.9.7 「MySQL ソース構成オプション」
セクション5.1.8 「サーバーシステム変数」
セクション12.14 「暗号化関数と圧縮関数」

CONCAT()

セクション26.48 「INFORMATION_SCHEMA VIEWS テーブル」
セクション13.7.7.13 「SHOW CREATE VIEW ステートメント」
セクション12.12 「XML 関数」
セクション12.11 「キャスト関数と演算子」
セクション5.1.11 「サーバー SQL モード」
セクション13.7.4.1 「ユーザー定義関数用の CREATE FUNCTION ステートメント」
セクション10.8.4 「式での照合の強制性」
セクション12.3 「式評価での型変換」
セクション10.2.1 「文字セットレパートリー」
セクション12.8 「文字列関数および演算子」
セクション1.7.1 「標準 SQL に対する MySQL 拡張機能」
セクション12.4.3 「論理演算子」
セクション12.8.3 「関数結果の文字セットと照合順序」
セクション12.20.1 「集計関数の説明」

CONCAT_WS()

セクション12.8 「文字列関数および演算子」
セクション12.20.1 「集計関数の説明」

CONNECTION_ID()

セクション13.1.20.6 「CHECK 制約」
セクション13.1.20.8 「CREATE TABLE および生成されるカラム」
セクション27.12.19.1 「error_log テーブル」
セクション26.23 「INFORMATION_SCHEMA PROCESSLIST テーブル」
セクション13.7.8.4 「KILL ステートメント」
セクション4.5.1.3 「mysql クライアントログイン」
セクション27.12.19.9 「processlist テーブル」
セクション13.7.7.29 「SHOW PROCESSLIST ステートメント」
セクション5.1.8 「サーバーシステム変数」
セクション27.12.19.10 「スレッドテーブル」
セクション17.2.1.3 「バイナリログインでの安全および安全でないステートメントの判断」
セクション12.22 「パフォーマンススキーマ関数」
セクション12.16 「情報関数」
セクション6.4.5.4 「監査ログファイル形式」

CONV()

セクション12.6.2 「数学関数」
セクション12.8 「文字列関数および演算子」
セクション12.8.3 「関数結果の文字セットと照合順序」

CONVERT()

[セクションA.11「MySQL 8.0 FAQ: MySQL の中国語、日本語、および韓国語の文字セット」](#)
[セクション1.3「MySQL 8.0 の新機能」](#)
[セクション6.5.3「MySQL Enterprise Data Masking and De-Identification の使用」](#)
[セクション4.5.1.1「mysql クライアントオプション」](#)
[セクション12.11「キャスト関数と演算子」](#)
[セクション10.3.8「文字セットイントロデューサ」](#)
[セクション10.3.6「文字列リテラルの文字セットおよび照合順序」](#)
[セクション12.4.2「比較関数と演算子」](#)

CONVERT_TZ()

[セクション13.3.6「LOCK TABLES および UNLOCK TABLES ステートメント」](#)
[セクション8.3.14「TIMESTAMP カラムからのインデックス付きルックアップ」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション5.4.5「スロークエリログ」](#)
[セクション5.4.3「一般クエリログ」](#)
[セクション12.7「日付および時間関数」](#)

COS()

[セクション12.6.2「数学関数」](#)

COT()

[セクション12.6.2「数学関数」](#)

COUNT()

[セクション8.2.1.17「GROUP BY の最適化」](#)
[セクション15.23「InnoDB の制限および制限事項」](#)
[セクション8.4.4「MySQL での内部一時テーブルの使用」](#)
[セクション1.2.2「MySQL の主な機能」](#)
[セクション23.1.7.3「NDB Cluster でのトランザクション処理に関する制限」](#)
[NDB Cluster ステータス変数](#)
[セクションB.3.4.3「NULL 値に関する問題」](#)
[セクション8.2.1.1「WHERE 句の最適化」](#)
[セクション12.24「その他の関数」](#)
[セクション5.1.11「サーバー SQL モード」](#)
[セクション8.2.2.4「マージまたは実体化を使用した導出テーブル、ビュー参照および共通テーブル式の最適化」](#)
[セクション13.7.4.1「ユーザー定義関数用の CREATE FUNCTION ステートメント」](#)
[セクション12.16「情報関数」](#)
[セクション25.5.3「更新可能および挿入可能なビュー」](#)
[セクション1.7.1「標準 SQL に対する MySQL 拡張機能」](#)
[セクション3.3.4.8「行のカウント」](#)
[セクション12.20.1「集計関数の説明」](#)

CRC32()

[セクション12.6.2「数学関数」](#)

create_asymmetric_priv_key()

[セクション6.6.2「MySQL Enterprise Encryption の使用方法と例」](#)
[セクション6.6.4「MySQL Enterprise Encryption ユーザー定義関数の説明」](#)
[セクション4.9「環境変数」](#)

create_asymmetric_pub_key()

[セクション6.6.4「MySQL Enterprise Encryption ユーザー定義関数の説明」](#)

create_dh_parameters()

[セクション6.6.2「MySQL Enterprise Encryption の使用方法と例」](#)

セクション6.6.4「MySQL Enterprise Encryption ユーザー定義関数の説明」
セクション4.9「環境変数」

create_digest()

セクション6.6.4「MySQL Enterprise Encryption ユーザー定義関数の説明」

CUME_DIST()

セクション12.21.1「Window 関数の説明」

CURDATE()

セクション17.2.1.3「バイナリロギングでの安全および安全でないステートメントの判断」
セクション12.7「日付および時間関数」
セクション3.3.4.5「日付の計算」

CURRENT_DATE

セクション11.6「データ型デフォルト値」
セクション12.7「日付および時間関数」

CURRENT_DATE()

セクション17.2.1.3「バイナリロギングでの安全および安全でないステートメントの判断」
セクション12.7「日付および時間関数」
セクション11.2.7「日付と時間型間での変換」

CURRENT_ROLE()

セクション6.2.10「ロールの使用」
セクション12.16「情報関数」

CURRENT_TIME

セクション12.7「日付および時間関数」

CURRENT_TIME()

セクション17.2.1.3「バイナリロギングでの安全および安全でないステートメントの判断」
セクション12.7「日付および時間関数」

CURRENT_TIMESTAMP

セクション13.1.13「CREATE EVENT ステートメント」
セクション13.1.20.8「CREATE TABLE および生成されるカラム」
セクション11.2.5「TIMESTAMP および DATETIME の自動初期化および更新機能」
セクション5.1.8「サーバーシステム変数」
セクション11.6「データ型デフォルト値」
セクション12.7「日付および時間関数」

CURRENT_TIMESTAMP()

セクション11.2.5「TIMESTAMP および DATETIME の自動初期化および更新機能」
セクション17.2.1.3「バイナリロギングでの安全および安全でないステートメントの判断」
セクション12.7「日付および時間関数」

CURRENT_USER

セクション13.7.1.1「ALTER USER ステートメント」
セクション13.1.13「CREATE EVENT ステートメント」
セクション13.1.17「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」
セクション13.1.22「CREATE TRIGGER ステートメント」
セクション13.1.23「CREATE VIEW ステートメント」
セクション17.5.1.8「CURRENT_USER() のレプリケーション」

セクション13.7.1.6 「GRANT ステートメント」
セクション13.7.7.12 「SHOW CREATE USER ステートメント」
セクション6.2.4 「アカウント名の指定」
セクション25.6 「ストアオブジェクトのアクセス制御」
セクション17.5.1.14 「レプリケーションとシステム関数」
セクション6.2.3 「付与テーブル」
セクション12.16 「情報関数」
セクション5.4.4.3 「混合形式のバイナリロギング形式」

CURRENT_USER()

セクション13.7.1.1 「ALTER USER ステートメント」
セクション13.1.20.6 「CHECK 制約」
セクション6.4.2.1 「Connection-Control プラグインのインストール」
セクション13.1.13 「CREATE EVENT ステートメント」
セクション13.1.17 「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」
セクション13.1.20.8 「CREATE TABLE および生成されるカラム」
セクション13.1.22 「CREATE TRIGGER ステートメント」
セクション13.1.23 「CREATE VIEW ステートメント」
セクション17.5.1.8 「CURRENT_USER() のレプリケーション」
セクション13.7.1.6 「GRANT ステートメント」
セクション6.4.1.7 「LDAP プラガブル認証」
セクション13.7.1.10 「SET PASSWORD ステートメント」
セクション13.7.7.12 「SHOW CREATE USER ステートメント」
セクション6.2.22 「SQL ベースのアカウントアクティビティ 監査」
セクション6.2.4 「アカウント名の指定」
セクション6.2.6 「アクセス制御、ステージ 1: 接続の検証」
セクション5.6.8.2 「キーリングサービス」
セクション25.6 「ストアオブジェクトのアクセス制御」
セクション6.4.3.2 「パスワード検証オプションおよび変数」
セクション6.2.15 「パスワード管理」
セクション6.2.18 「プロキシユーザー」
セクション10.2.2 「メタデータ用の UTF-8」
セクション17.5.1.14 「レプリケーションとシステム関数」
セクション6.2.5 「ロール名の指定」
セクション12.16 「情報関数」
セクション6.4.4.10 「汎用キーリングキー管理関数」
セクション5.4.4.3 「混合形式のバイナリロギング形式」
セクション6.4.5.4 「監査ログファイル形式」

CURTIME()

セクション5.1.15 「MySQL Server でのタイムゾーンのサポート」
セクション17.2.1.3 「バイナリロギングでの安全および安全でないステートメントの判断」
セクション12.7 「日付および時間関数」

D

[\[index top\]](#)

DATABASE()

セクション13.1.24 「DROP DATABASE ステートメント」
セクションB.3.7 「MySQL の既知の問題」
セクション3.4 「データベースとテーブルに関する情報の取得」
セクション3.3.1 「データベースの作成と選択」
セクション10.2.2 「メタデータ用の UTF-8」
セクション12.16 「情報関数」

DATE()

セクション12.7 「日付および時間関数」

DATE_ADD()

セクション12.21.3「ウィンドウ機能フレーム仕様」
セクション9.5「式」
セクション12.7「日付および時間関数」
セクション3.3.4.5「日付の計算」
セクション11.2「日時データ型」
セクション12.6.1「算術演算子」

DATE_FORMAT()

セクション10.16「MySQL Server のロケールサポート」
セクション12.11「キャスト関数と演算子」
セクション5.1.8「サーバーシステム変数」
セクション12.7「日付および時間関数」

DATE_SUB()

セクション9.5「式」
セクション12.7「日付および時間関数」
セクション11.2「日時データ型」

DATEDIFF()

セクション12.7「日付および時間関数」
セクション24.6.3「関数に関連するパーティショニング制限」

DAY()

セクション12.7「日付および時間関数」
セクション24.6.3「関数に関連するパーティショニング制限」

DAYNAME()

セクション10.16「MySQL Server のロケールサポート」
セクション5.1.8「サーバーシステム変数」
セクション12.7「日付および時間関数」

DAYOFMONTH()

セクション12.7「日付および時間関数」
セクション3.3.4.5「日付の計算」
セクション24.6.3「関数に関連するパーティショニング制限」

DAYOFWEEK()

セクション12.7「日付および時間関数」
セクション24.6.3「関数に関連するパーティショニング制限」

DAYOFYEAR()

セクション24.2「パーティショニングタイプ」
セクション12.7「日付および時間関数」
セクション24.6.3「関数に関連するパーティショニング制限」

DEFAULT()

セクション13.1.9.2「ALTER TABLE および生成されるカラム」
セクション13.2.6「INSERT ステートメント」
セクション13.2.9「REPLACE ステートメント」
セクション12.24「その他の関数」
セクション11.6「データ型デフォルト値」

DEGREES()

セクション12.6.2「数学関数」

DENSE_RANK()

セクション12.21.1「Window 関数の説明」

E

[[index top](#)]

ELT()

セクションB.3.7「MySQL の既知の問題」
セクション12.8「文字列関数および演算子」
セクション1.7.1「標準 SQL に対する MySQL 拡張機能」
セクション12.8.3「関数結果の文字セットと照合順序」

EXP()

セクション13.1.20「CREATE TABLE ステートメント」
セクション12.6.2「数学関数」

EXPORT_SET()

セクション12.8「文字列関数および演算子」

EXTRACT()

セクション12.11「キャスト関数と演算子」
セクション9.5「式」
セクション12.7「日付および時間関数」
セクション24.6.3「関数に関連するパーティショニング制限」

ExtractValue()

セクション12.12「XML 関数」

F

[[index top](#)]

FIELD()

セクション12.8「文字列関数および演算子」

FIND_IN_SET()

セクション11.3.6「SET 型」
セクション12.8「文字列関数および演算子」

firewall_group_delist()

セクション6.4.7.4「MySQL Enterprise Firewall リファレンス」

firewall_group_enlist()

セクション6.4.7.4「MySQL Enterprise Firewall リファレンス」

FIRST_VALUE()

セクション12.21.1「Window 関数の説明」
セクション12.21.3「ウィンドウ機能フレーム仕様」

FLOOR()

セクション12.20.3「MySQL での GROUP BY の処理」
セクション8.9.6「オプティマイザ統計」
セクション12.6.2「数学関数」
セクション24.6.3「関数に関連するパーティショニング制限」

FORMAT()

[セクション10.16「MySQL Server のロケールサポート」](#)
[セクション12.24「その他の関数」](#)
[セクション12.6.2「数学関数」](#)
[セクション12.8「文字列関数および演算子」](#)
[セクション1.7.1「標準 SQL に対する MySQL 拡張機能」](#)
[セクション12.8.3「関数結果の文字セットと照合順序」](#)

FORMAT_BYTES()

[セクション28.4.5.3「format_bytes\(\) 関数」](#)
[セクション8.12.3.1「MySQL のメモリーの使用方法」](#)
[セクション12.22「パフォーマンススキーマ関数」](#)

FORMAT_PICO_TIME()

[セクション28.4.5.6「format_time\(\) 関数」](#)
[セクション12.22「パフォーマンススキーマ関数」](#)

FOUND_ROWS()

[セクション1.3「MySQL 8.0 の新機能」](#)
[セクション13.2.10「SELECT ステートメント」](#)
[セクション17.2.1.1「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」](#)
[セクション17.2.1.3「バイナリロギングでの安全および安全でないステートメントの判断」](#)
[セクション17.5.1.14「レプリケーションとシステム関数」](#)
[セクション12.16「情報関数」](#)
[セクション5.4.4.3「混合形式のバイナリロギング形式」](#)

FROM_BASE64()

[セクション12.8「文字列関数および演算子」](#)

FROM_DAYS()

[セクション12.7「日付および時間関数」](#)
[セクション1.7.1「標準 SQL に対する MySQL 拡張機能」](#)

FROM_UNIXTIME()

[セクション8.3.14「TIMESTAMP カラムからのインデックス付きルックアップ」](#)
[セクション17.5.1.33「レプリケーションとタイムゾーン」](#)
[セクション12.7「日付および時間関数」](#)

G

[\[index top\]](#)

gen_blacklist()

[セクション6.5.5「MySQL Enterprise Data Masking and De-Identification ユーザー定義関数の説明」](#)

gen_blocklist()

[セクション6.5.3「MySQL Enterprise Data Masking and De-Identification の使用」](#)
[セクション6.5.5「MySQL Enterprise Data Masking and De-Identification ユーザー定義関数の説明」](#)

gen_dictionary()

[セクション6.5.3「MySQL Enterprise Data Masking and De-Identification の使用」](#)
[セクション6.5.5「MySQL Enterprise Data Masking and De-Identification ユーザー定義関数の説明」](#)

gen_dictionary_drop()

[セクション6.5.5「MySQL Enterprise Data Masking and De-Identification ユーザー定義関数の説明」](#)

gen_dictionary_load()

セクション6.5.3「MySQL Enterprise Data Masking and De-Identification の使用」

セクション6.5.5「MySQL Enterprise Data Masking and De-Identification ユーザー定義関数の説明」

gen_range()

セクション6.5.3「MySQL Enterprise Data Masking and De-Identification の使用」

セクション6.5.5「MySQL Enterprise Data Masking and De-Identification ユーザー定義関数の説明」

gen_rnd_email()

セクション6.5.3「MySQL Enterprise Data Masking and De-Identification の使用」

セクション6.5.5「MySQL Enterprise Data Masking and De-Identification ユーザー定義関数の説明」

gen_rnd_pan()

セクション6.5.3「MySQL Enterprise Data Masking and De-Identification の使用」

セクション6.5.5「MySQL Enterprise Data Masking and De-Identification ユーザー定義関数の説明」

gen_rnd_ssn()

セクション6.5.3「MySQL Enterprise Data Masking and De-Identification の使用」

セクション6.5.5「MySQL Enterprise Data Masking and De-Identification ユーザー定義関数の説明」

gen_rnd_us_phone()

セクション6.5.3「MySQL Enterprise Data Masking and De-Identification の使用」

セクション6.5.5「MySQL Enterprise Data Masking and De-Identification ユーザー定義関数の説明」

GeomCollection()

セクション12.17.5「ジオメトリ値を作成する MySQL 固有の関数」

GeometryCollection()

セクション12.17.5「ジオメトリ値を作成する MySQL 固有の関数」

セクション12.17.6「ジオメトリ形式変換関数」

GET_DD_COLUMN_PRIVILEGES()

セクション12.23「内部関数」

GET_DD_CREATE_OPTIONS()

セクション12.23「内部関数」

GET_DD_INDEX_SUB_PART_LENGTH()

セクション12.23「内部関数」

GET_FORMAT()

セクション10.16「MySQL Server のロケールサポート」

セクション12.7「日付および時間関数」

GET_LOCK()

セクション13.1.13「CREATE EVENT ステートメント」

セクション13.7.8.4「KILL ステートメント」

セクション13.3.6「LOCK TABLES および UNLOCK TABLES ステートメント」

セクション27.12.13.3「metadata_locks テーブル」

セクション28.4.4.14「ps_setup_save() プロシージャ」

セクション25.4.1「イベントスケジューラの概要」

セクション18.9.2「グループレプリケーションの制限事項」

セクション17.2.1.1「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」

セクション17.2.1.3「バイナリロギングでの安全および安全でないステートメントの判断」

セクション8.11.4「メタデータのロック」

セクション17.5.1.14 「レプリケーションとシステム関数」

セクション5.6.8.1 「ロックサービス」

ロックサービス UDF インタフェース

セクション12.15 「ロック関数」

セクション8.14.3 「一般的なスレッドの状態」

セクション8.11.1 「内部ロック方法」

gethostbyaddr()

セクション5.1.12.3 「DNS ルックアップとホストキャッシュ」

gethostbyname()

セクション5.1.12.3 「DNS ルックアップとホストキャッシュ」

getrusage()

セクション23.5.14.54 「ndbinfo threadstat テーブル」

gettimeofday()

セクション23.5.14.54 「ndbinfo threadstat テーブル」

GREATEST()

セクション11.5 「JSON データ型」

セクション12.4.2 「比較関数と演算子」

セクション12.8.3 「関数結果の文字セットと照合順序」

GROUP_CONCAT()

セクション11.5 「JSON データ型」

セクション8.4.4 「MySQL での内部一時テーブルの使用」

セクション1.2.2 「MySQL の主な機能」

セクションB.3.7 「MySQL の既知の問題」

セクション5.1.8 「サーバーシステム変数」

セクション1.7.1 「標準 SQL に対する MySQL 拡張機能」

セクション12.20.1 「集計関数の説明」

group_replication_get_communication_protocol()

セクション18.8 「グループレプリケーションシステム変数」

セクション18.7.1.2 「グループレプリケーション通信プロトコルのバージョン」

セクション13.4.3.6 「グループレプリケーション通信プロトコルのバージョンを検査および設定する関数」

セクション18.4.1.4 「グループ通信プロトコルバージョンの設定」

セクション18.6.4 「メッセージの断片化」

group_replication_get_write_concurrency()

セクション13.4.3.5 「グループの最大コンセンサスインスタンスを検査および構成する関数」

セクション18.4.1.3 「グループレプリケーショングループ書き込みコンセンサスの使用」

group_replication_set_as_primary()

セクション18.7.1.1 「アップグレード中のメンバーバージョン」

セクション18.4.1.1 「グループプライマリメンバーの変更」

セクション18.7.3.3 「グループレプリケーションのオンラインアップグレード方法」

セクション13.4.3.3 「グループレプリケーションプライマリを構成する機能」

セクション18.1.3.1 「シングルプライマリモード」

セクション18.4.2.1 「トランザクションの一貫性保証の理解」

group_replication_set_communication_protocol()

セクション6.2.2 「MySQL で提供される権限」

セクション18.7.2 「グループレプリケーションのオフラインアップグレード」

セクション18.8 「グループレプリケーションシステム変数」

セクション18.7.1.2「グループレプリケーション通信プロトコルのバージョン」
セクション13.4.3.6「グループレプリケーション通信プロトコルのバージョンを検査および設定する関数」
セクション18.4.1.4「グループ通信プロトコルバージョンの設定」
セクション18.6.4「メッセージの断片化」

group_replication_set_write_concurrency()

セクション6.2.2「MySQL で提供される権限」
セクション13.4.3.5「グループの最大コンセンサスインスタンスを検査および構成する関数」
セクション18.4.1.3「グループレプリケーショングループ書き込みコンセンサスの使用」

group_replication_switch_to_multi_primary_mode()

セクション18.4.1.2「グループモードの変更」
セクション18.8「グループレプリケーションシステム変数」
セクション13.4.3.4「グループレプリケーションモードを構成する関数」
セクション18.7.1「グループ内の異なるメンバーバージョンの組合せ」
バージョンの互換性
セクション18.1.3「マルチプライマリモードとシングルプライマリモード」

group_replication_switch_to_single_primary_mode()

セクション18.4.1.2「グループモードの変更」
セクション18.8「グループレプリケーションシステム変数」
セクション13.4.3.4「グループレプリケーションモードを構成する関数」
セクション18.1.3.1「シングルプライマリモード」
セクション18.1.3「マルチプライマリモードとシングルプライマリモード」

GROUPING()

セクション12.20.2「GROUP BY 修飾子」
セクション12.21.2「Window 関数の概念と構文」
セクション12.24「その他の関数」

GTID_INTERSECTION_WITH_UUID

セクション17.1.3.8「GTID を操作するストアドファンクションの例」

GTID_IS_DISJOINT

セクション17.1.3.8「GTID を操作するストアドファンクションの例」

GTID_IS_DISJOINT_UNION

セクション17.1.3.8「GTID を操作するストアドファンクションの例」

GTID_IS_EQUAL

セクション17.1.3.8「GTID を操作するストアドファンクションの例」

GTID_SUBSET

セクション17.1.3.8「GTID を操作するストアドファンクションの例」

GTID_SUBSET()

セクション17.1.3.8「GTID を操作するストアドファンクションの例」
セクション17.1.3.1「GTID 形式および格納」
セクション12.19「グローバルトランザクション識別子 (GTID) で使用される機能」

GTID_SUBTRACT

セクション17.1.3.8「GTID を操作するストアドファンクションの例」

GTID_SUBTRACT()

セクション17.1.3.8「GTID を操作するストアドファンクションの例」
セクション17.1.3.1「GTID 形式および格納」

セクション17.1.6.5「グローバルトランザクション ID システム変数」
セクション12.19「グローバルトランザクション識別子 (GTID) で使用される機能」

GTID_SUBTRACT_UUID

セクション17.1.3.8「GTID を操作するストアドファンクションの例」

GTID_UNION

セクション17.1.3.8「GTID を操作するストアドファンクションの例」

H

[\[index top\]](#)

HEX()

セクション9.1.4「16 進数リテラル」
セクション12.24「その他の関数」
セクション9.1.5「ビット値リテラル」
セクション12.6.2「数学関数」
セクション10.3.6「文字列リテラルの文字セットおよび照合順序」
セクション12.8「文字列関数および演算子」
セクション6.4.4.10「汎用キーリングキー管理関数」
セクション12.8.3「関数結果の文字セットと照合順序」

HOUR()

セクション12.7「日付および時間関数」
セクション24.6.3「関数に関連するパーティショニング制限」

I

[\[index top\]](#)

ICU_VERSION()

セクション12.16「情報関数」

IF

セクション13.5.1「PREPARE ステートメント」

IF()

セクション13.6.5.2「IF ステートメント」
セクション15.15.3「InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル」
セクションB.3.7「MySQL の既知の問題」
セクション12.5「フロー制御関数」
セクション1.7.1「標準 SQL に対する MySQL 拡張機能」
セクション12.8.3「関数結果の文字セットと照合順序」

IFNULL

セクション13.5.1「PREPARE ステートメント」

IFNULL()

セクションB.3.4.3「NULL 値に関する問題」
セクション12.5「フロー制御関数」

INET6_ATON()

セクション5.1.13「IPv6 サポート」
セクション12.24「その他の関数」
セクション12.13「ビット関数と演算子」

INET6_NTOA()

セクション5.1.13 「IPv6 サポート」

セクション12.24 「その他の関数」

INET_ATON()

セクション5.1.13 「IPv6 サポート」

セクション12.24 「その他の関数」

INET_NTOA()

セクション5.1.13 「IPv6 サポート」

セクション12.24 「その他の関数」

INSERT()

セクション12.8 「文字列関数および演算子」

INSTR()

セクション12.8 「文字列関数および演算子」

セクション12.8.3 「関数結果の文字セットと照合順序」

INTERNAL_AUTO_INCREMENT()

セクション12.23 「内部関数」

INTERNAL_AVG_ROW_LENGTH()

セクション12.23 「内部関数」

INTERNAL_CHECK_TIME()

セクション12.23 「内部関数」

INTERNAL_CHECKSUM()

セクション12.23 「内部関数」

INTERNAL_DATA_FREE()

セクション12.23 「内部関数」

INTERNAL_DATA_LENGTH()

セクション12.23 「内部関数」

INTERNAL_DD_CHAR_LENGTH()

セクション12.23 「内部関数」

INTERNAL_GET_COMMENT_OR_ERROR()

セクション12.23 「内部関数」

INTERNAL_GET_ENABLED_ROLE_JSON()

セクション12.23 「内部関数」

INTERNAL_GET_HOSTNAME()

セクション12.23 「内部関数」

INTERNAL_GET_USERNAME()

セクション12.23 「内部関数」

INTERNAL_GET_VIEW_WARNING_OR_ERROR()

セクション12.23 「内部関数」

INTERNAL_INDEX_COLUMN_CARDINALITY()

セクション12.23 「内部関数」

INTERNAL_INDEX_LENGTH()

セクション12.23 「内部関数」

INTERNAL_IS_ENABLED_ROLE()

セクション12.23 「内部関数」

INTERNAL_IS_MANDATORY_ROLE()

セクション12.23 「内部関数」

INTERNAL_KEYS_DISABLED()

セクション12.23 「内部関数」

INTERNAL_MAX_DATA_LENGTH()

セクション12.23 「内部関数」

INTERNAL_TABLE_ROWS()

セクション12.23 「内部関数」

INTERNAL_UPDATE_TIME()

セクション12.23 「内部関数」

INTERVAL()

セクション12.4.2 「比較関数と演算子」

IS_FREE_LOCK()

セクション17.2.1.1 「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」

セクション17.2.1.3 「バイナリロギングでの安全および安全でないステートメントの判断」

セクション17.5.1.14 「レプリケーションとシステム関数」

セクション12.15 「ロック関数」

IS_IPV4()

セクション12.24 「その他の関数」

IS_IPV4_COMPAT()

セクション12.24 「その他の関数」

IS_IPV4_MAPPED()

セクション12.24 「その他の関数」

IS_IPV6()

セクション12.24 「その他の関数」

IS_USED_LOCK()

セクション17.2.1.1 「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」

セクション17.2.1.3 「バイナリロギングでの安全および安全でないステートメントの判断」

セクション17.5.1.14 「レプリケーションとシステム関数」

セクション12.15 「ロック関数」

IS_UUID()

セクション12.24 「その他の関数」

IS_VISIBLE_DD_OBJECT()

セクション12.23「内部関数」

ISNULL()

セクション12.4.2「比較関数と演算子」

J

[\[index top\]](#)

JSON_ARRAY()

セクション11.5「JSON データ型」
セクション12.18.2「JSON 値を作成する関数」
セクション12.18.3「JSON 値を検索する関数」

JSON_ARRAY_APPEND()

セクション12.18.4「JSON 値を変更する関数」
セクション1.3「MySQL 8.0 の新機能」

JSON_ARRAY_INSERT()

セクション12.18.4「JSON 値を変更する関数」

JSON_ARRAYAGG()

セクション12.18.2「JSON 値を作成する関数」
セクション12.18.1「JSON 関数リファレンス」
セクション1.3「MySQL 8.0 の新機能」
セクション12.20.1「集計関数の説明」

JSON_CONTAINS()

セクション13.1.15「CREATE INDEX ステートメント」
セクション12.18.3「JSON 値を検索する関数」
セクション1.3「MySQL 8.0 の新機能」

JSON_CONTAINS_PATH()

セクション11.5「JSON データ型」
セクション12.18.3「JSON 値を検索する関数」

JSON_DEPTH()

セクション12.18.5「JSON 値属性を返す関数」

JSON_EXTRACT()

セクション12.18.7「JSON スキーマ検証関数」
セクション11.5「JSON データ型」
セクション12.18.3「JSON 値を検索する関数」
セクション1.3「MySQL 8.0 の新機能」
セクション13.1.20.9「セカンダリインデックスと生成されたカラム」

JSON_INSERT()

セクション11.5「JSON データ型」
セクション12.18.4「JSON 値を変更する関数」

JSON_KEYS()

セクション12.18.3「JSON 値を検索する関数」
セクション6.2.3「付与テーブル」

JSON_LENGTH()

[セクション12.18.5「JSON 値属性を返す関数」](#)

JSON_MERGE()

[セクション12.18.4「JSON 値を変更する関数」](#)

[セクション1.3「MySQL 8.0 の新機能」](#)

JSON_MERGE_PATCH()

[セクション13.7.1.1「ALTER USER ステートメント」](#)

[セクション11.5「JSON データ型」](#)

[セクション12.18.4「JSON 値を変更する関数」](#)

[セクション1.3「MySQL 8.0 の新機能」](#)

JSON_MERGE_PRESERVE()

[セクション11.5「JSON データ型」](#)

[セクション12.18.4「JSON 値を変更する関数」](#)

[セクション1.3「MySQL 8.0 の新機能」](#)

JSON_OBJECT()

[セクション11.5「JSON データ型」](#)

[セクション12.18.2「JSON 値を作成する関数」](#)

[セクション12.18.3「JSON 値を検索する関数」](#)

JSON_OBJECTAGG()

[セクション12.18.2「JSON 値を作成する関数」](#)

[セクション12.18.1「JSON 関数リファレンス」](#)

[セクション1.3「MySQL 8.0 の新機能」](#)

[セクション12.20.1「集計関数の説明」](#)

JSON_OVERLAPS()

[セクション13.1.15「CREATE INDEX ステートメント」](#)

[セクション12.18.3「JSON 値を検索する関数」](#)

[セクション1.3「MySQL 8.0 の新機能」](#)

JSON_PRETTY()

[セクション12.18.7「JSON スキーマ検証関数」](#)

[セクション12.18.8「JSON ユーティリティ関数」](#)

[セクション12.18.1「JSON 関数リファレンス」](#)

[セクション1.3「MySQL 8.0 の新機能」](#)

JSON_QUOTE()

[セクション12.18.8「JSON ユーティリティ関数」](#)

[セクション12.18.2「JSON 値を作成する関数」](#)

JSON_REMOVE()

[セクション11.5「JSON データ型」](#)

[セクション12.18.8「JSON ユーティリティ関数」](#)

[セクション12.18.4「JSON 値を変更する関数」](#)

[セクション1.3「MySQL 8.0 の新機能」](#)

[セクション17.1.6.4「バイナリロギングのオプションと変数」](#)

JSON_REPLACE()

[セクション11.5「JSON データ型」](#)

[セクション12.18.8「JSON ユーティリティ関数」](#)

[セクション12.18.4「JSON 値を変更する関数」](#)

[セクション1.3「MySQL 8.0の新機能」](#)
[セクション17.1.6.4「バイナリロギングのオプションと変数」](#)

JSON_SCHEMA_VALID()

[セクション12.18.7「JSONスキーマ検証関数」](#)
[セクション1.3「MySQL 8.0の新機能」](#)

JSON_SCHEMA_VALIDATION_REPORT()

[セクション12.18.7「JSONスキーマ検証関数」](#)
[セクション1.3「MySQL 8.0の新機能」](#)

JSON_SEARCH()

[セクション11.5「JSONデータ型」](#)
[セクション12.18.3「JSON値を検索する関数」](#)

JSON_SET()

[セクション11.5「JSONデータ型」](#)
[セクション12.18.8「JSONユーティリティ関数」](#)
[セクション12.18.4「JSON値を変更する関数」](#)
[セクション1.3「MySQL 8.0の新機能」](#)
[セクション17.1.6.4「バイナリロギングのオプションと変数」](#)
[セクション6.4.5.6「監査ログファイルの読取り」](#)

JSON_STORAGE_FREE()

[セクション11.5「JSONデータ型」](#)
[セクション12.18.8「JSONユーティリティ関数」](#)
[セクション12.18.1「JSON関数リファレンス」](#)
[セクション1.3「MySQL 8.0の新機能」](#)

JSON_STORAGE_SIZE()

[セクション11.5「JSONデータ型」](#)
[セクション12.18.8「JSONユーティリティ関数」](#)
[セクション12.18.1「JSON関数リファレンス」](#)
[セクション1.3「MySQL 8.0の新機能」](#)

JSON_TABLE()

[セクション12.18.6「JSONテーブル関数」](#)
[セクション1.3「MySQL 8.0の新機能」](#)
[セクション13.2.11.9「ラテラル導出テーブル」](#)
[セクション13.2.11.8「導出テーブル」](#)

JSON_TYPE()

[セクション11.5「JSONデータ型」](#)
[セクション12.18.3「JSON値を検索する関数」](#)
[セクション12.18.5「JSON値属性を返す関数」](#)

JSON_UNQUOTE()

[セクション12.18.3「JSON値を検索する関数」](#)

JSON_UNQUOTE()

[セクション13.1.15「CREATE INDEXステートメント」](#)
[セクション11.5「JSONデータ型」](#)
[セクション12.18.4「JSON値を変更する関数」](#)
[セクション1.3「MySQL 8.0の新機能」](#)
[セクション13.1.20.9「セカンダリインデックスと生成されたカラム」](#)
[セクション8.3.11「生成されたカラムインデックスのオプティマイザによる使用」](#)

JSON_VALID()

[セクション12.18.5「JSON 値属性を返す関数」](#)

JSON_VALUE()

[セクション12.18.3「JSON 値を検索する関数」](#)

[セクション1.3「MySQL 8.0 の新機能」](#)

[セクション12.11「キャスト関数と演算子」](#)

[セクション13.1.20.9「セカンダリインデックスと生成されたカラム」](#)

K

[\[index top\]](#)

keyring_aws_rotate_cmk()

[セクション6.4.4.5「keyring_aws Amazon Web Services キーリングプラグインの使用」](#)

[セクション6.4.4.11「プラグイン固有のキーリングキー管理関数」](#)

keyring_aws_rotate_keys()

[セクション6.4.4.5「keyring_aws Amazon Web Services キーリングプラグインの使用」](#)

[セクション6.4.4.11「プラグイン固有のキーリングキー管理関数」](#)

keyring_hashicorp_update_config()

[セクション6.4.4.6「HashiCorp Vault キーリングプラグインの使用」](#)

[セクション6.4.4.11「プラグイン固有のキーリングキー管理関数」](#)

keyring_key_fetch()

[セクション6.4.4.10「汎用キーリングキー管理関数」](#)

keyring_key_generate()

[セクション6.4.4.10「汎用キーリングキー管理関数」](#)

keyring_key_length_fetch()

[セクション6.4.4.10「汎用キーリングキー管理関数」](#)

keyring_key_remove()

[セクション6.4.4.10「汎用キーリングキー管理関数」](#)

keyring_key_store()

[セクション6.4.4.10「汎用キーリングキー管理関数」](#)

keyring_key_type_fetch()

[セクション6.4.4.10「汎用キーリングキー管理関数」](#)

L

[\[index top\]](#)

LAG()

[セクション1.3「MySQL 8.0 の新機能」](#)

[セクション12.21.1「Window 関数の説明」](#)

LAST_DAY()

[セクション12.7「日付および時間関数」](#)

LAST_INSERT_ID()

[セクション3.6.9「AUTO_INCREMENT の使用」](#)
[セクション13.1.20「CREATE TABLE ステートメント」](#)
[セクション13.2.6.2「INSERT ... ON DUPLICATE KEY UPDATE ステートメント」](#)
[セクション13.2.6「INSERT ステートメント」](#)
[セクション13.3.6「LOCK TABLES および UNLOCK TABLES ステートメント」](#)
[セクション23.4.23「ndb_restore — NDB Cluster バックアップの復元」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション25.2.4「ストアードプロシージャ、関数、トリガー、および LAST_INSERT_ID\(\)」](#)
[セクション17.2.1.3「バイナリロギングでの安全および安全でないステートメントの判断」](#)
[セクション17.5.1.1「レプリケーションと AUTO_INCREMENT」](#)
[セクション17.5.4「レプリケーションのトラブルシューティング」](#)
[セクション12.16「情報関数」](#)
[セクション25.5.3「更新可能および挿入可能なビュー」](#)
[セクション1.7.1「標準 SQL に対する MySQL 拡張機能」](#)
[セクション12.4.2「比較関数と演算子」](#)

LAST_VALUE()

[セクション12.21.1「Window 関数の説明」](#)
[セクション12.21.3「ウィンドウ機能フレーム仕様」](#)

LCASE()

[セクション12.8「文字列関数および演算子」](#)
[セクション12.8.3「関数結果の文字セットと照合順序」](#)

LEAD()

[セクション1.3「MySQL 8.0 の新機能」](#)
[セクション12.21.1「Window 関数の説明」](#)

LEAST()

[セクション11.5「JSON データ型」](#)
[セクション12.4.2「比較関数と演算子」](#)
[セクション12.8.3「関数結果の文字セットと照合順序」](#)

LEFT()

[セクション12.8「文字列関数および演算子」](#)

LENGTH()

[セクション11.4.3「サポートされる空間データ形式」](#)
[セクション11.7「データ型のストレージ要件」](#)
[セクション12.8「文字列関数および演算子」](#)

LineString()

[セクション12.17.5「ジオメトリ値を作成する MySQL 固有の関数」](#)

LN()

[セクション12.6.2「数学関数」](#)

LOAD_FILE()

[セクション13.2.8「LOAD XML ステートメント」](#)
[セクション6.6.2「MySQL Enterprise Encryption の使用方法と例」](#)
[セクション6.2.2「MySQL で提供される権限」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション17.2.1.1「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」](#)
[セクション17.2.1.3「バイナリロギングでの安全および安全でないステートメントの判断」](#)
[セクション17.5.1.14「レプリケーションとシステム関数」](#)

[セクション12.8「文字列関数および演算子」](#)
[セクション5.4.4.3「混合形式のバイナリロギング形式」](#)

load_rewrite_rules()

[リライタのクエリーリライトプロシージャおよび関数](#)
[セクション5.6.4「リライタクエリーリライトプラグイン」](#)

LOCALTIME

[セクション11.2.5「TIMESTAMP および DATETIME の自動初期化および更新機能」](#)
[セクション12.7「日付および時間関数」](#)

LOCALTIME()

[セクション11.2.5「TIMESTAMP および DATETIME の自動初期化および更新機能」](#)
[セクション17.2.1.3「バイナリロギングでの安全および安全でないステートメントの判断」](#)
[セクション12.7「日付および時間関数」](#)

LOCALTIMESTAMP

[セクション11.2.5「TIMESTAMP および DATETIME の自動初期化および更新機能」](#)
[セクション12.7「日付および時間関数」](#)

LOCALTIMESTAMP()

[セクション11.2.5「TIMESTAMP および DATETIME の自動初期化および更新機能」](#)
[セクション17.2.1.3「バイナリロギングでの安全および安全でないステートメントの判断」](#)
[セクション12.7「日付および時間関数」](#)

LOCATE()

[セクション12.8「文字列関数および演算子」](#)

LOG()

[セクション24.2.4.1「LINEAR HASH パーティショニング」](#)
[セクション12.6.2「数学関数」](#)

LOG10()

[セクション12.6.2「数学関数」](#)

LOG2()

[セクション12.6.2「数学関数」](#)

LOWER()

[セクション10.8.7「INFORMATION_SCHEMA 検索での照合の使用」](#)
[セクション10.10.1「Unicode 文字セット」](#)
[セクション12.11「キャスト関数と演算子」](#)
[セクション12.8「文字列関数および演算子」](#)
[セクション12.8.3「関数結果の文字セットと照合順序」](#)

LPAD()

[セクション1.3「MySQL 8.0 の新機能」](#)
[セクション12.13「ビット関数と演算子」](#)
[セクション11.1.1「数値データ型の構文」](#)
[セクション11.1.6「数値型の属性」](#)
[セクション12.8「文字列関数および演算子」](#)

LTRIM()

[セクション12.8「文字列関数および演算子」](#)
[セクション12.8.3「関数結果の文字セットと照合順序」](#)

M

[\[index top\]](#)

MAKE_SET()

セクション12.8「文字列関数および演算子」

MAKEDATE()

セクション12.7「日付および時間関数」

MAKETIME()

セクション12.7「日付および時間関数」

mask_inner()

セクション6.5.3「MySQL Enterprise Data Masking and De-Identification の使用」

セクション6.5.5「MySQL Enterprise Data Masking and De-Identification ユーザー定義関数の説明」

mask_outer()

セクション6.5.3「MySQL Enterprise Data Masking and De-Identification の使用」

セクション6.5.5「MySQL Enterprise Data Masking and De-Identification ユーザー定義関数の説明」

mask_pan()

セクション6.5.3「MySQL Enterprise Data Masking and De-Identification の使用」

セクション6.5.5「MySQL Enterprise Data Masking and De-Identification ユーザー定義関数の説明」

mask_pan_relaxed()

セクション6.5.3「MySQL Enterprise Data Masking and De-Identification の使用」

セクション6.5.5「MySQL Enterprise Data Masking and De-Identification ユーザー定義関数の説明」

mask_ssn()

セクション6.5.3「MySQL Enterprise Data Masking and De-Identification の使用」

セクション6.5.5「MySQL Enterprise Data Masking and De-Identification ユーザー定義関数の説明」

MASTER_POS_WAIT()

セクションA.14「MySQL 8.0 FAQ: レプリケーション」

セクション12.24「その他の関数」

セクション17.2.1.1「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」

セクション17.2.1.3「バイナリロギングでの安全および安全でないステートメントの判断」

バイナリログトランザクション圧縮が有効な場合の動作

セクション17.2.2.2「以前のレプリケーションステートメントとの互換性」

セクション17.2.2.1「単一チャンネルで操作するためのコマンド」

MATCH

セクション9.5「式」

MATCH ()

セクション12.10「全文検索関数」

MATCH()

セクション15.6.2.4「InnoDB FULLTEXT インデックス」

セクション12.10.6「MySQL の全文検索の微調整」

MySQL 用語集

セクション12.10.2「ブール全文検索」

セクション12.10.5「全文制限」

[セクション12.10「全文検索関数」](#)
[セクション12.10.1「自然言語全文検索」](#)

MAX()

[セクション12.20.1「集計関数の説明」](#)

MAX()

[セクション3.6.9「AUTO_INCREMENT の使用」](#)
[セクション8.2.1.17「GROUP BY の最適化」](#)
[セクション11.5「JSON データ型」](#)
[セクション8.3.1「MySQL のインデックスの使用の仕組み」](#)
[セクション1.2.2「MySQL の主な機能」](#)
[セクションB.3.7「MySQL の既知の問題」](#)
[セクション23.4.23「ndb_restore — NDB Cluster バックアップの復元」](#)
[セクション27.12.16.1「tp_thread_group_state テーブル」](#)
[セクション13.2.15「WITH \(共通テーブル式\)」](#)
[セクション8.3.10「インデックス拡張の使用」](#)
[セクション8.2.1.21「ウィンドウ機能最適化」](#)
[セクション5.1.11「サーバー SQL モード」](#)
[セクション8.2.2.4「マージまたは実体化を使用した導出テーブル、ビュー参照および共通テーブル式の最適化」](#)
[セクション8.9.2「切り替え可能な最適化」](#)
[セクション11.1.1「数値データ型の構文」](#)
[セクション11.2.8「日付の 2 桁の年」](#)
[セクション25.5.3「更新可能および挿入可能なビュー」](#)
[セクション8.3.13「降順インデックス」](#)
[セクション12.20.1「集計関数の説明」](#)

MBRContains()

[セクション12.17.9.2「最小境界矩形を使用する空間リレーション関数」](#)
[セクション11.4.11「空間インデックスの使用」](#)

MBRCoveredBy()

[セクション12.17.9.2「最小境界矩形を使用する空間リレーション関数」](#)

MBRCovers()

[セクション12.17.9.2「最小境界矩形を使用する空間リレーション関数」](#)

MBRDisjoint()

[セクション12.17.9.2「最小境界矩形を使用する空間リレーション関数」](#)

MBREquals()

[セクション12.17.9.2「最小境界矩形を使用する空間リレーション関数」](#)

MBRIntersects()

[セクション12.17.9.2「最小境界矩形を使用する空間リレーション関数」](#)

MBROverlaps()

[セクション12.17.9.2「最小境界矩形を使用する空間リレーション関数」](#)

MBRTouches()

[セクション12.17.9.2「最小境界矩形を使用する空間リレーション関数」](#)

MBRWithin()

[セクション12.17.9.2「最小境界矩形を使用する空間リレーション関数」](#)
[セクション11.4.11「空間インデックスの使用」](#)

MD5()

[セクション24.2.5「KEY パーティショニング」](#)
[セクション9.2「スキーマオブジェクト名」](#)
[セクション12.14「暗号化関数と圧縮関数」](#)
[セクション1.7.1「標準 SQL に対する MySQL 拡張機能」](#)

MICROSECOND()

[セクション12.7「日付および時間関数」](#)
[セクション24.6.3「関数に関連するパーティショニング制限」](#)

MID()

[セクション12.8「文字列関数および演算子」](#)
[セクション12.8.3「関数結果の文字セットと照合順序」](#)

MIN()

[セクション12.20.1「集計関数の説明」](#)

MIN()

[セクション8.2.1.17「GROUP BY の最適化」](#)
[セクション11.5「JSON データ型」](#)
[セクション8.3.1「MySQL のインデックスの使用の仕組み」](#)
[セクション1.2.2「MySQL の主な機能」](#)
[セクションB.3.7「MySQL の既知の問題」](#)
[セクション23.4.23「ndb_restore — NDB Cluster バックアップの復元」](#)
[セクションB.3.4.3「NULL 値に関する問題」](#)
[セクション8.2.1.1「WHERE 句の最適化」](#)
[セクション8.3.10「インデックス拡張の使用」](#)
[セクション8.2.1.21「ウィンドウ機能最適化」](#)
[セクション8.2.2.4「マージまたは実体化を使用した導出テーブル、ビュー参照および共通テーブル式の最適化」](#)
[セクション8.9.2「切り替え可能な最適化」](#)
[セクション11.1.1「数値データ型の構文」](#)
[セクション11.2.8「日付の 2 桁の年」](#)
[セクション25.5.3「更新可能および挿入可能なビュー」](#)
[セクション8.3.13「降順インデックス」](#)
[セクション12.20.1「集計関数の説明」](#)

MINUTE()

[セクション12.7「日付および時間関数」](#)
[セクション24.6.3「関数に関連するパーティショニング制限」](#)

MOD()

[セクション5.1.11「サーバー SQL モード」](#)
[セクション12.6.2「数学関数」](#)
[セクション3.3.4.5「日付の計算」](#)
[セクション1.7.1「標準 SQL に対する MySQL 拡張機能」](#)
[セクション12.6.1「算術演算子」](#)
[セクション24.6.3「関数に関連するパーティショニング制限」](#)

MONTH()

[セクション24.2「パーティショニングタイプ」](#)
[セクション12.7「日付および時間関数」](#)
[セクション3.3.4.5「日付の計算」](#)
[セクション24.6.3「関数に関連するパーティショニング制限」](#)

MONTHNAME()

[セクション10.16「MySQL Server のロケールサポート」](#)

セクション5.1.8「サーバーシステム変数」

セクション12.7「日付および時間関数」

MultiLineString()

セクション12.17.5「ジオメトリ値を作成する MySQL 固有の関数」

MultiPoint()

セクション12.17.5「ジオメトリ値を作成する MySQL 固有の関数」

MultiPolygon()

セクション12.17.5「ジオメトリ値を作成する MySQL 固有の関数」

my_open()

セクション5.1.10「サーバーステータス変数」

mysql_firewall_flush_status()

セクション6.4.7.4「MySQL Enterprise Firewall リファレンス」

mysql_query_attribute_string()

セクション9.6「クエリー属性」

セクション5.5.4「クエリー属性コンポーネント」

N

[\[index top\]](#)

NAME_CONST()

セクション12.24「その他の関数」

セクション25.7「ストアードプログラムバイナリロギング」

normalize_statement()

セクション6.4.7.4「MySQL Enterprise Firewall リファレンス」

NOW()

セクション13.1.20.6「CHECK 制約」

セクション13.1.17「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」

セクション13.1.20.8「CREATE TABLE および生成されるカラム」

セクションA.1「MySQL 8.0 FAQ: 全般」

セクション5.1.15「MySQL Server でのタイムゾーンのサポート」

セクション28.4.4.25「statement_performance_analyzer() プロシージャ」

セクション11.2.5「TIMESTAMP および DATETIME の自動初期化および更新機能」

セクション11.2.4「YEAR 型」

セクション5.1.7「サーバーコマンドオプション」

セクション5.1.8「サーバーシステム変数」

セクション17.2.1.1「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」

セクション11.6「データ型デフォルト値」

セクション17.2.1.3「バイナリロギングでの安全および安全でないステートメントの判断」

セクション28.4.3.21「メトリックビュー」

セクション17.5.1.14「レプリケーションとシステム関数」

セクション17.5.1.33「レプリケーションとタイムゾーン」

セクション12.7「日付および時間関数」

セクション11.2.6「時間値での小数秒」

NTH_VALUE()

セクション1.3「MySQL 8.0 の新機能」

[セクション12.21.1「Window 関数の説明」](#)
[セクション12.21.3「ウィンドウ機能フレーム仕様」](#)

NTILE()

[セクション1.3「MySQL 8.0 の新機能」](#)
[セクション12.21.1「Window 関数の説明」](#)

NULLIF

[セクション13.5.1「PREPARE ステートメント」](#)

NULLIF()

[セクション12.5「フロー制御関数」](#)

O

[\[index top\]](#)

OCT()

[セクション12.8「文字列関数および演算子」](#)

OCTET_LENGTH()

[セクション12.8「文字列関数および演算子」](#)

ORD()

[セクション12.8「文字列関数および演算子」](#)

P

[\[index top\]](#)

PERCENT_RANK()

[セクション12.21.1「Window 関数の説明」](#)

PERIOD_ADD()

[セクション12.7「日付および時間関数」](#)
[セクション1.7.1「標準 SQL に対する MySQL 拡張機能」](#)

PERIOD_DIFF()

[セクション12.7「日付および時間関数」](#)
[セクション1.7.1「標準 SQL に対する MySQL 拡張機能」](#)

PI()

[セクション12.6.2「数学関数」](#)
[セクション9.2.5「関数名の構文解析と解決」](#)

Point()

[セクション11.4.3「サポートされる空間データ形式」](#)
[セクション12.17.5「ジオメトリ値を作成する MySQL 固有の関数」](#)

Polygon()

[セクション12.17.5「ジオメトリ値を作成する MySQL 固有の関数」](#)

POSITION()

[セクション12.8「文字列関数および演算子」](#)

POW()

[セクション24.2.4 「HASH パーティショニング」](#)
[セクション12.6.2 「数学関数」](#)
[セクション8.2.1.20 「関数コールの最適化」](#)

POWER()

[セクション24.2.4.1 「LINEAR HASH パーティショニング」](#)
[セクション12.6.2 「数学関数」](#)

PS_CURRENT_THREAD_ID()

[セクション28.4.5.15 「ps_thread_id\(\) 関数」](#)
[セクション12.22 「パフォーマンススキーマ関数」](#)

PS_THREAD_ID()

[セクション28.4.5.15 「ps_thread_id\(\) 関数」](#)
[セクション12.22 「パフォーマンススキーマ関数」](#)

pthread_mutex()

[セクション1.8.1 「MySQL への貢献者」](#)

Q

[\[index top\]](#)

QUARTER()

[セクション12.7 「日付および時間関数」](#)
[セクション24.6.3 「関数に関連するパーティショニング制限」](#)

QUOTE()

[セクション9.1.1 「文字列リテラル」](#)
[セクション12.8 「文字列関数および演算子」](#)

R

[\[index top\]](#)

RADIANS()

[セクション12.6.2 「数学関数」](#)

RAND()

[セクション13.1.17 「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」](#)
[セクション5.1.8 「サーバーシステム変数」](#)
[セクション17.2.1.1 「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」](#)
[セクション17.2.1.3 「バイナリロギングでの安全および安全でないステートメントの判断」](#)
[セクション17.5.1.14 「レプリケーションとシステム関数」](#)
[セクション8.9.2 「切り替え可能な最適化」](#)
[セクション12.6.2 「数学関数」](#)
[セクション8.2.1.20 「関数コールの最適化」](#)

RANDOM_BYTES()

[セクション12.14 「暗号化関数と圧縮関数」](#)

RANK()

[セクション1.3 「MySQL 8.0 の新機能」](#)

[セクション12.21.1 「Window 関数の説明」](#)

read_firewall_group_allowlist()

[セクション6.4.7.4 「MySQL Enterprise Firewall リファレンス」](#)

read_firewall_groups()

[セクション6.4.7.4 「MySQL Enterprise Firewall リファレンス」](#)

read_firewall_users()

[セクション6.4.7.4 「MySQL Enterprise Firewall リファレンス」](#)

read_firewall_whitelist()

[セクション6.4.7.4 「MySQL Enterprise Firewall リファレンス」](#)

REGEXP_INSTR()

[セクション1.3 「MySQL 8.0 の新機能」](#)

[セクション12.8.2 「正規表現」](#)

REGEXP_LIKE()

[セクション1.3 「MySQL 8.0 の新機能」](#)

[セクション5.1.8 「サーバーシステム変数」](#)

[セクション3.3.4.7 「パターンマッチング」](#)

[セクション12.8.2 「正規表現」](#)

REGEXP_REPLACE()

[セクション1.3 「MySQL 8.0 の新機能」](#)

[セクション12.8.2 「正規表現」](#)

REGEXP_SUBSTR()

[セクション1.3 「MySQL 8.0 の新機能」](#)

[セクション12.8.2 「正規表現」](#)

RELEASE_ALL_LOCKS()

[セクション12.15 「ロック関数」](#)

RELEASE_LOCK()

[セクション13.2.3 「DO ステートメント」](#)

[セクション13.3.6 「LOCK TABLES および UNLOCK TABLES ステートメント」](#)

[セクション17.2.1.1 「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」](#)

[セクション17.2.1.3 「バイナリロギングでの安全および安全でないステートメントの判断」](#)

[セクション17.5.1.14 「レプリケーションとシステム関数」](#)

[セクション12.15 「ロック関数」](#)

[セクション8.11.1 「内部ロック方法」](#)

REPEAT()

[セクション12.8 「文字列関数および演算子」](#)

[セクション12.8.3 「関数結果の文字セットと照合順序」](#)

REPLACE()

[セクション12.8 「文字列関数および演算子」](#)

[セクション12.8.3 「関数結果の文字セットと照合順序」](#)

REVERSE()

[セクション12.8 「文字列関数および演算子」](#)

[セクション12.8.3 「関数結果の文字セットと照合順序」](#)

RIGHT()

[セクション12.8「文字列関数および演算子」](#)
[セクション12.8.3「関数結果の文字セットと照合順序」](#)

ROLES_GRAPHML()

[セクション6.2.2「MySQL で提供される権限」](#)
[セクション12.16「情報関数」](#)

ROUND()

[セクション15.15.3「InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル」](#)
[セクション12.25.4「丸め動作」](#)
[セクション12.6.2「数学関数」](#)
[セクション12.25「高精度計算」](#)
[セクション12.25.5「高精度計算の例」](#)

ROW_COUNT()

[セクション13.2.1「CALL ステートメント」](#)
[セクション13.2.2「DELETE ステートメント」](#)
[セクション13.2.6「INSERT ステートメント」](#)
[セクション13.6.7.7「MySQL の診断領域」](#)
[セクション17.2.1.3「バイナリロギングでの安全および安全でないステートメントの判断」](#)
[セクション17.5.1.14「レプリケーションとシステム関数」](#)
[セクション12.16「情報関数」](#)
[セクション5.4.4.3「混合形式のバイナリロギング形式」](#)

ROW_NUMBER()

[セクション12.21.2「Window 関数の概念と構文」](#)
[セクション12.21.1「Window 関数の説明」](#)

RPAD()

[セクション12.13「ビット関数と演算子」](#)
[セクション12.8「文字列関数および演算子」](#)
[セクション12.8.3「関数結果の文字セットと照合順序」](#)

RTRIM()

[セクション12.8「文字列関数および演算子」](#)
[セクション12.8.3「関数結果の文字セットと照合順序」](#)

S

[\[index top\]](#)

SCHEMA()

[セクション12.16「情報関数」](#)

SEC_TO_TIME()

[セクション12.7「日付および時間関数」](#)

SECOND()

[セクション12.7「日付および時間関数」](#)
[セクション24.6.3「関数に関連するパーティショニング制限」](#)

service_get_read_locks()

[ロックサービス UDF インタフェース](#)

service_get_write_locks()

ロックサービス UDF インタフェース

service_release_locks()

ロックサービス UDF インタフェース

SESSION_USER()

セクション17.2.1.3 「バイナリロギングでの安全および安全でないステートメントの判断」

セクション10.2.2 「メタデータ用の UTF-8」

セクション12.16 「情報関数」

set_firewall_group_mode()

セクション6.4.7.4 「MySQL Enterprise Firewall リファレンス」

set_firewall_mode()

セクション6.4.7.4 「MySQL Enterprise Firewall リファレンス」

setrlimit()

セクション5.1.8 「サーバーシステム変数」

SHA()

セクション12.14 「暗号化関数と圧縮関数」

SHA1()

セクション12.14 「暗号化関数と圧縮関数」

SHA2()

セクション1.3 「MySQL 8.0 の新機能」

セクション6.1.1 「セキュリティーガイドライン」

セクション12.14 「暗号化関数と圧縮関数」

SIGN()

セクション12.6.2 「数学関数」

SIN()

セクション12.6.2 「数学関数」

SLEEP()

セクション27.12.16.2 「tp_thread_group_stats テーブル」

セクション12.24 「その他の関数」

セクション17.2.1.1 「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」

セクション17.2.1.3 「バイナリロギングでの安全および安全でないステートメントの判断」

セクション8.14.3 「一般的なスレッドの状態」

SOUNDEX()

セクション12.8 「文字列関数および演算子」

セクション12.8.3 「関数結果の文字セットと照合順序」

SPACE()

セクション12.8 「文字列関数および演算子」

セクション12.8.3 「関数結果の文字セットと照合順序」

SQRT()

セクション12.6.2 「数学関数」

ST_Area()

[セクション12.17.7.4「Polygon および MultiPolygon プロパティ関数」](#)

[セクション12.17.7「ジオメトリプロパティ関数」](#)

ST_AsBinary()

[セクション12.17.6「ジオメトリ形式変換関数」](#)

[セクション11.4.8「空間データのフェッチ」](#)

ST_AsGeoJSON()

[セクション11.5「JSON データ型」](#)

[セクション12.17.11「空間 GeoJSON 関数」](#)

ST_AsText()

[セクション12.17.6「ジオメトリ形式変換関数」](#)

[セクション11.4.8「空間データのフェッチ」](#)

ST_AsWKB()

[セクション12.17.6「ジオメトリ形式変換関数」](#)

ST_AsWKT()

[セクション12.17.3「WKT 値からジオメトリ値を作成する関数」](#)

[セクション12.17.6「ジオメトリ形式変換関数」](#)

ST_Buffer()

[セクション12.17.8「空間演算子関数」](#)

ST_Buffer_Strategy()

[セクション5.1.8「サーバーシステム変数」](#)

[セクション12.17.8「空間演算子関数」](#)

ST_Centroid()

[セクション12.17.7.4「Polygon および MultiPolygon プロパティ関数」](#)

ST_Collect()

[セクション12.17.12「空間集計関数」](#)

ST_Collect()

[セクション12.17.12「空間集計関数」](#)

ST_Contains()

[セクション12.17.9.1「オブジェクト形状を使用する空間関係関数」](#)

ST_ConvexHull()

[セクション12.17.8「空間演算子関数」](#)

ST_Crosses()

[セクション12.17.9.1「オブジェクト形状を使用する空間関係関数」](#)

ST_Difference()

[セクション12.17.8「空間演算子関数」](#)

ST_Dimension()

[セクション12.17.7.1「一般的なジオメトリプロパティ関数」](#)

ST_Disjoint()

セクション12.17.9.1「オブジェクト形状を使用する空間関係関数」

ST_Distance()

セクション26.37「INFORMATION_SCHEMA ST_UNITS_OF_MEASURE テーブル」

セクション12.17.9.1「オブジェクト形状を使用する空間関係関数」

セクション12.17.13「空間の便利な関数」

ST_Distance_Sphere()

セクション12.17.9.1「オブジェクト形状を使用する空間関係関数」

セクション12.17.13「空間の便利な関数」

ST_EndPoint()

セクション12.17.7.3「LineString および MultiLineString のプロパティ関数」

セクション12.17.8「空間演算子関数」

ST_Envelope()

セクション12.17.7.1「一般的なジオメトリプロパティ関数」

セクション12.17.8「空間演算子関数」

ST_Equals()

セクション12.17.9.1「オブジェクト形状を使用する空間関係関数」

ST_ExteriorRing()

セクション12.17.7.4「Polygon および MultiPolygon プロパティ関数」

セクション12.17.8「空間演算子関数」

ST_FrechetDistance()

セクション12.17.9.1「オブジェクト形状を使用する空間関係関数」

ST_GeoHash()

セクション12.17.10「空間 Geohash 関数」

ST_GeomCollFromText()

セクション12.17.3「WKT 値からジオメトリ値を作成する関数」

ST_GeomCollFromTxt()

セクション12.17.3「WKT 値からジオメトリ値を作成する関数」

ST_GeomCollFromWKB()

セクション12.17.4「WKB 値からジオメトリ値を作成する関数」

ST_GeometryCollectionFromText()

セクション12.17.3「WKT 値からジオメトリ値を作成する関数」

ST_GeometryCollectionFromWKB()

セクション12.17.4「WKB 値からジオメトリ値を作成する関数」

ST_GeometryFromText()

セクション12.17.3「WKT 値からジオメトリ値を作成する関数」

ST_GeometryFromWKB()

セクション12.17.4「WKB 値からジオメトリ値を作成する関数」

ST_GeometryN()

[セクション12.17.7.5「GeometryCollection プロパティ関数」](#)

[セクション12.17.8「空間演算子関数」](#)

ST_GeometryType()

[セクション12.17.7.1「一般的なジオメトリプロパティ関数」](#)

ST_GeomFromGeoJSON()

[セクション11.5「JSON データ型」](#)

[セクション12.17.11「空間 GeoJSON 関数」](#)

ST_GeomFromText()

[セクション12.17.3「WKT 値からジオメトリ値を作成する関数」](#)

[セクション11.4.3「サポートされる空間データ形式」](#)

[セクション12.17.5「ジオメトリ値を作成する MySQL 固有の関数」](#)

[セクション12.17.6「ジオメトリ形式変換関数」](#)

[セクション11.4.7「空間カラムへのデータ移入」](#)

ST_GeomFromWKB()

[セクション12.17.4「WKB 値からジオメトリ値を作成する関数」](#)

ST_HausdorffDistance()

[セクション12.17.9.1「オブジェクト形状を使用する空間関係関数」](#)

ST_InteriorRingN()

[セクション12.17.7.4「Polygon および MultiPolygon プロパティ関数」](#)

[セクション12.17.8「空間演算子関数」](#)

ST_Intersection()

[セクション12.17.8「空間演算子関数」](#)

ST_Intersects()

[セクション12.17.9.1「オブジェクト形状を使用する空間関係関数」](#)

ST_IsClosed()

[セクション12.17.7.3「LineString および MultiLineString のプロパティ関数」](#)

ST_IsEmpty()

[セクション12.17.7.1「一般的なジオメトリプロパティ関数」](#)

ST_IsSimple()

[セクション12.17.7.1「一般的なジオメトリプロパティ関数」](#)

ST_IsValid()

[セクション11.4.4「ジオメトリの整形形式と妥当性」](#)

[セクション12.17.13「空間の便利な関数」](#)

ST_LatFromGeoHash()

[セクション12.17.10「空間 Geohash 関数」](#)

ST_Latitude()

[セクション12.17.7.2「Pointプロパティ関数」](#)

ST_Length()

セクション12.17.7.3「LineString および MultiLineString のプロパティ関数」
セクション1.3「MySQL 8.0 の新機能」
セクション12.8「文字列関数および演算子」
セクション12.17.8「空間演算子関数」

ST_LineFromText()

セクション12.17.3「WKT 値からジオメトリ値を作成する関数」

ST_LineFromWKB()

セクション12.17.4「WKB 値からジオメトリ値を作成する関数」

ST_LineInterpolatePoint()

セクション12.17.8「空間演算子関数」

ST_LineInterpolatePoints()

セクション12.17.8「空間演算子関数」

ST_LineStringFromText()

セクション12.17.3「WKT 値からジオメトリ値を作成する関数」

ST_LineStringFromWKB()

セクション12.17.4「WKB 値からジオメトリ値を作成する関数」

ST_LongFromGeoHash()

セクション12.17.10「空間 Geohash 関数」

ST_Longitude()

セクション12.17.7.2「Pointプロパティ関数」

ST_MakeEnvelope()

セクション12.17.13「空間の便利な関数」

ST_MLineFromText()

セクション12.17.3「WKT 値からジオメトリ値を作成する関数」

ST_MLineFromWKB()

セクション12.17.4「WKB 値からジオメトリ値を作成する関数」

ST_MPointFromText()

セクション12.17.3「WKT 値からジオメトリ値を作成する関数」
セクション11.4.3「サポートされる空間データ形式」

ST_MPointFromWKB()

セクション12.17.4「WKB 値からジオメトリ値を作成する関数」

ST_MPolyFromText()

セクション12.17.3「WKT 値からジオメトリ値を作成する関数」

ST_MPolyFromWKB()

セクション12.17.4「WKB 値からジオメトリ値を作成する関数」

ST_MultiLineStringFromText()

セクション12.17.3「WKT 値からジオメトリ値を作成する関数」

ST_MultiLineStringFromWKB()

セクション12.17.4 「WKB 値からジオメトリ値を作成する関数」

ST_MultiPointFromText()

セクション12.17.3 「WKT 値からジオメトリ値を作成する関数」

ST_MultiPointFromWKB()

セクション12.17.4 「WKB 値からジオメトリ値を作成する関数」

ST_MultiPolygonFromText()

セクション12.17.3 「WKT 値からジオメトリ値を作成する関数」

ST_MultiPolygonFromWKB()

セクション12.17.4 「WKB 値からジオメトリ値を作成する関数」

ST_NumGeometries()

セクション12.17.7.5 「GeometryCollection プロパティ関数」

ST_NumInteriorRing()

セクション12.17.7.4 「Polygon および MultiPolygon プロパティ関数」

ST_NumInteriorRings()

セクション12.17.7.4 「Polygon および MultiPolygon プロパティ関数」

ST_NuminteriorRings()

セクション12.17.7.4 「Polygon および MultiPolygon プロパティ関数」

ST_NumPoints()

セクション12.17.7.3 「LineString および MultiLineString のプロパティ関数」

ST_Overlaps()

セクション12.17.9.1 「オブジェクト形状を使用する空間関係関数」

ST_PointAtDistance()

セクション12.17.8 「空間演算子関数」

ST_PointFromGeoHash()

セクション12.17.10 「空間 Geohash 関数」

ST_PointFromText()

セクション12.17.3 「WKT 値からジオメトリ値を作成する関数」

ST_PointFromWKB()

セクション12.17.4 「WKB 値からジオメトリ値を作成する関数」

ST_PointN()

セクション12.17.7.3 「LineString および MultiLineString のプロパティ関数」

セクション12.17.8 「空間演算子関数」

ST_PolyFromText()

セクション12.17.3 「WKT 値からジオメトリ値を作成する関数」

ST_PolyFromWKB()

セクション12.17.4「WKB 値からジオメトリ値を作成する関数」

ST_PolygonFromText()

セクション12.17.3「WKT 値からジオメトリ値を作成する関数」

ST_PolygonFromWKB()

セクション12.17.4「WKB 値からジオメトリ値を作成する関数」

ST_Simplify()

セクション12.17.13「空間の便利な関数」

ST_SRID()

セクション12.17.7.1「一般的なジオメトリプロパティ関数」

セクション12.17.8「空間演算子関数」

ST_StartPoint()

セクション12.17.7.3「LineString および MultiLineString のプロパティ関数」

セクション12.17.8「空間演算子関数」

ST_SwapXY()

セクション12.17.6「ジオメトリ形式変換関数」

ST_SymDifference()

セクション12.17.8「空間演算子関数」

ST_Touches()

セクション12.17.9.1「オブジェクト形状を使用する空間関係関数」

ST_Transform()

セクション12.17.7.1「一般的なジオメトリプロパティ関数」

セクション12.17.8「空間演算子関数」

ST_Union()

セクション12.17.8「空間演算子関数」

ST_Validate()

セクション12.17.13「空間の便利な関数」

ST_Within()

セクション12.17.9.1「オブジェクト形状を使用する空間関係関数」

ST_X()

セクション12.17.7.2「Pointプロパティ関数」

セクション11.4.3「サポートされる空間データ形式」

ST_Y()

セクション12.17.7.2「Pointプロパティ関数」

STATEMENT_DIGEST()

セクション27.10「パフォーマンススキーマのステートメントダイジェストとサンプリング」

セクション12.14「暗号化関数と圧縮関数」

STATEMENT_DIGEST_TEXT()

セクション6.4.7.4「MySQL Enterprise Firewall リファレンス」
セクション27.10「パフォーマンススキーマのステートメントダイジェストとサンプリング」
セクション12.14「暗号化関数と圧縮関数」

STD()

セクション1.2.2「MySQL の主な機能」
セクション1.7.1「標準 SQL に対する MySQL 拡張機能」
セクション12.20.1「集計関数の説明」

STDDEV()

セクション12.20.1「集計関数の説明」

STDDEV_POP()

セクション8.2.1.21「ウィンドウ機能最適化」
セクション12.20.1「集計関数の説明」

STDDEV_SAMP()

セクション8.2.1.21「ウィンドウ機能最適化」
セクション12.20.1「集計関数の説明」

STR_TO_DATE()

セクション10.16「MySQL Server のロケールサポート」
セクション12.7「日付および時間関数」
セクション11.2「日時データ型」

STRCMP()

セクションB.3.4.2「DATE カラムの使用に関する問題」
セクション12.8.1「文字列比較関数および演算子」

SUBDATE()

セクション12.7「日付および時間関数」

SUBSTR()

セクション12.13「ビット関数と演算子」
セクション12.8「文字列関数および演算子」

SUBSTRING()

セクション13.1.15「CREATE INDEX ステートメント」
セクション12.8「文字列関数および演算子」
セクション12.8.3「関数結果の文字セットと照合順序」

SUBSTRING_INDEX()

セクション6.2.22「SQL ベースのアカウントアクティビティ監査」
セクション12.8「文字列関数および演算子」

SUBTIME()

セクション12.7「日付および時間関数」

SUM()

セクション12.20.1「集計関数の説明」

SUM()

セクション11.3.5「ENUM 型」

[セクション8.2.1.17「GROUP BY の最適化」](#)
[セクション1.3「MySQL 8.0 の新機能」](#)
[セクション1.2.2「MySQL の主な機能」](#)
[セクションB.3.4.3「NULL 値に関する問題」](#)
[セクション11.3.6「SET 型」](#)
[セクション12.21.2「Window 関数の概念と構文」](#)
[セクション12.24「その他の関数」](#)
[セクション12.21.3「ウィンドウ機能フレーム仕様」](#)
[セクション8.2.2.4「マージまたは実体化を使用した導出テーブル、ビュー参照および共通テーブル式の最適化」](#)
[セクション13.7.4.1「ユーザー定義関数用の CREATE FUNCTION ステートメント」](#)
[セクション11.2.1「日時データ型の構文」](#)
[セクション25.5.3「更新可能および挿入可能なビュー」](#)
[セクション12.20.1「集計関数の説明」](#)

SYSDATE()

[セクション5.1.7「サーバーコマンドオプション」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション17.2.1.1「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」](#)
[セクション17.2.1.3「バイナリロギングでの安全および安全でないステートメントの判断」](#)
[セクション17.5.1.14「レプリケーションとシステム関数」](#)
[セクション12.7「日付および時間関数」](#)

SYSTEM_USER()

[セクション17.2.1.3「バイナリロギングでの安全および安全でないステートメントの判断」](#)
[セクション10.2.2「メタデータ用の UTF-8」](#)
[セクション12.16「情報関数」](#)

T

[\[index top\]](#)

TAN()

[セクション12.6.2「数学関数」](#)

TIME()

[セクション12.7「日付および時間関数」](#)

TIME_FORMAT()

[セクション12.11「キャスト関数と演算子」](#)
[セクション12.7「日付および時間関数」](#)

TIME_TO_SEC()

[セクション12.7「日付および時間関数」](#)
[セクション24.6.3「関数に関連するパーティショニング制限」](#)

TIMEDIFF()

[セクション12.7「日付および時間関数」](#)

TIMESTAMP()

[セクション12.7「日付および時間関数」](#)

TIMESTAMPADD()

[セクション12.7「日付および時間関数」](#)

TIMESTAMPDIFF()

[セクション12.7「日付および時間関数」](#)

[セクション3.3.4.5「日付の計算」](#)

TO_BASE64()

[セクション12.8「文字列関数および演算子」](#)

TO_DAYS()

[セクション24.2.4「HASH パーティショニング」](#)

[セクション24.2「パーティショニングタイプ」](#)

[セクション24.4「パーティションプルーニング」](#)

[セクション12.7「日付および時間関数」](#)

[セクション1.7.1「標準 SQL に対する MySQL 拡張機能」](#)

[セクション24.6.3「関数に関連するパーティショニング制限」](#)

TO_SECONDS()

[セクション24.2「パーティショニングタイプ」](#)

[セクション24.4「パーティションプルーニング」](#)

[セクション12.7「日付および時間関数」](#)

[セクション24.6.3「関数に関連するパーティショニング制限」](#)

TRIM()

[セクション10.7「カラム文字セットの変換」](#)

[セクション12.8「文字列関数および演算子」](#)

[セクション1.7.1「標準 SQL に対する MySQL 拡張機能」](#)

[セクション12.8.3「関数結果の文字セットと照合順序」](#)

TRUNCATE()

[セクション12.6.2「数学関数」](#)

U

[\[index top\]](#)

UCASE()

[セクション12.8「文字列関数および演算子」](#)

[セクション12.8.3「関数結果の文字セットと照合順序」](#)

UNCOMPRESS()

[セクション2.9.7「MySQL ソース構成オプション」](#)

[セクション5.1.8「サーバーシステム変数」](#)

[セクション12.14「暗号化関数と圧縮関数」](#)

UNCOMPRESSED_LENGTH()

[セクション12.14「暗号化関数と圧縮関数」](#)

UNHEX()

[セクション4.5.1.1「mysql クライアントオプション」](#)

[セクション12.8「文字列関数および演算子」](#)

[セクション12.14「暗号化関数と圧縮関数」](#)

UNIX_TIMESTAMP()

[セクション24.2.1「RANGE パーティショニング」](#)

[セクション8.3.14「TIMESTAMP カラムからのインデックス付きルックアップ」](#)

[セクション5.1.8「サーバーシステム変数」](#)

[セクションB.3.3.7「タイムゾーンの問題」](#)

[セクション17.2.1.3「バイナリロギングでの安全および安全でないステートメントの判断」](#)

[セクション28.4.3.21「メトリックビュー」](#)
[セクション12.7「日付および時間関数」](#)
[セクション24.6.3「関数に関連するパーティショニング制限」](#)

UpdateXML()

[セクション12.12「XML 関数」](#)

UPPER()

[セクション10.8.7「INFORMATION_SCHEMA 検索での照合の使用」](#)
[セクション10.10.1「Unicode 文字セット」](#)
[セクション12.11「キャスト関数と演算子」](#)
[セクション10.2.1「文字セットレパートリー」](#)
[セクション12.8「文字列関数および演算子」](#)
[セクション12.8.3「関数結果の文字セットと照合順序」](#)

USER()

[セクション13.7.1.1「ALTER USER ステートメント」](#)
[セクション6.4.1.7「LDAP プラガブル認証」](#)
[セクション6.2.22「SQL ベースのアカウントアクティビティ 監査」](#)
[セクション17.2.1.1「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」](#)
[セクション5.1.14「ネットワークネームスペースのサポート」](#)
[セクション17.2.1.3「バイナリロギングでの安全および安全でないステートメントの判断」](#)
[セクション6.4.3.2「パスワード検証オプションおよび変数」](#)
[セクション6.2.15「パスワード管理」](#)
[セクション6.2.18「プロキシユーザー」](#)
[セクション10.2.2「メタデータ用の UTF-8」](#)
[セクション17.5.1.14「レプリケーションとシステム関数」](#)
[セクション10.8.4「式での照合の強制性」](#)
[セクション12.16「情報関数」](#)
[セクション5.4.4.3「混合形式のバイナリロギング形式」](#)

UTC_DATE

[セクション12.7「日付および時間関数」](#)

UTC_DATE()

[セクション17.2.1.3「バイナリロギングでの安全および安全でないステートメントの判断」](#)
[セクション12.7「日付および時間関数」](#)

UTC_TIME

[セクション12.7「日付および時間関数」](#)

UTC_TIME()

[セクション17.2.1.3「バイナリロギングでの安全および安全でないステートメントの判断」](#)
[セクション12.7「日付および時間関数」](#)

UTC_TIMESTAMP

[セクション12.7「日付および時間関数」](#)

UTC_TIMESTAMP()

[セクション5.1.15「MySQL Server でのタイムゾーンのサポート」](#)
[セクション17.2.1.3「バイナリロギングでの安全および安全でないステートメントの判断」](#)
[セクション12.7「日付および時間関数」](#)

UUID()

[セクション12.24「その他の関数」](#)

セクション17.2.1.1「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」
セクション25.7「ストアプログラムバイナリロギング」
セクション17.2.1.3「バイナリロギングでの安全および安全でないステートメントの判断」
セクション17.1.6.4「バイナリロギングのオプションと変数」
セクション17.5.1.14「レプリケーションとシステム関数」
セクション5.4.4.3「混合形式のバイナリロギング形式」
セクション8.2.1.20「関数コールの最適化」

UUID_SHORT()

セクション12.24「その他の関数」
セクション17.2.1.1「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」
セクション17.2.1.3「バイナリロギングでの安全および安全でないステートメントの判断」

UUID_TO_BIN()

セクション12.24「その他の関数」
セクション12.13「ビット関数と演算子」

V

[\[index top\]](#)

VALIDATE_PASSWORD_STRENGTH()

セクション6.4.3.2「パスワード検証オプションおよび変数」
セクション6.4.3「パスワード検証コンポーネント」
セクション12.14「暗号化関数と圧縮関数」

VALUES()

セクション13.2.6.2「INSERT ... ON DUPLICATE KEY UPDATE ステートメント」
セクション1.3「MySQL 8.0 の新機能」
セクション13.2.14「VALUES ステートメント」
セクション12.24「その他の関数」

VAR_POP()

セクション8.2.1.21「ウィンドウ機能最適化」
セクション12.20.1「集計関数の説明」

VAR_SAMP()

セクション8.2.1.21「ウィンドウ機能最適化」
セクション12.20.1「集計関数の説明」

VARIANCE()

セクション12.20.1「集計関数の説明」

VERSION()

セクション17.2.1.1「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」
セクション28.4.3.47「バージョンビュー」
セクション10.2.2「メタデータ用の UTF-8」
セクション17.5.1.14「レプリケーションとシステム関数」
セクション10.8.4「式での照合の強制性」
セクション12.16「情報関数」
セクションB.3.4.1「文字列検索での大文字/小文字の区別」
セクション6.4.5.4「監査ログファイル形式」

version_tokens_delete()

セクション5.6.6.3「バージョントークンの使用」
セクション5.6.6.4「バージョントークン参照」

version_tokens_edit()

セクション5.6.6.3「バージョントークンの使用」

セクション5.6.6.4「バージョントークン参照」

version_tokens_lock_exclusive()

セクション5.6.6.3「バージョントークンの使用」

セクション5.6.6.4「バージョントークン参照」

version_tokens_lock_shared()

セクション5.6.6.3「バージョントークンの使用」

セクション5.6.6.4「バージョントークン参照」

version_tokens_set()

セクション5.6.6.3「バージョントークンの使用」

セクション5.6.6.4「バージョントークン参照」

version_tokens_show()

セクション5.6.6.3「バージョントークンの使用」

セクション5.6.6.4「バージョントークン参照」

version_tokens_unlock()

セクション5.6.6.3「バージョントークンの使用」

セクション5.6.6.4「バージョントークン参照」

W

[\[index top\]](#)

WAIT_FOR_EXECUTED_GTID_SET()

セクション17.1.3.8「GTID を操作するストアドファンクションの例」

セクション12.19「グローバルランザクション識別子 (GTID) で使用される機能」

セクション17.1.4.1「レプリケーションモードの概念」

WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS()

セクション12.19「グローバルランザクション識別子 (GTID) で使用される機能」

WEEK()

セクション5.1.8「サーバーシステム変数」

セクション12.7「日付および時間関数」

WEEKDAY()

セクション24.2「パーティショニングタイプ」

セクション12.7「日付および時間関数」

セクション1.7.1「標準 SQL に対する MySQL 拡張機能」

セクション24.6.3「関数に関連するパーティショニング制限」

WEEKOFYEAR()

セクション12.7「日付および時間関数」

WEIGHT_STRING()

セクション10.10.1「Unicode 文字セット」

セクション10.14「文字セットへの照合順序の追加」

セクションB.3.4.1「文字列検索での大文字/小文字の区別」

セクション12.8「文字列関数および演算子」

Y

[\[index top\]](#)

YEAR()

[セクション24.2.4 「HASH パーティショニング」](#)
[セクション24.2.7 「MySQL パーティショニングによる NULL の扱い」](#)
[セクション24.3.1 「RANGE および LIST パーティションの管理」](#)
[セクション24.2.1 「RANGE パーティショニング」](#)
[セクション24.2 「パーティショニングタイプ」](#)
[セクション24.4 「パーティションプルーニング」](#)
[セクション12.7 「日付および時間関数」](#)
[セクション3.3.4.5 「日付の計算」](#)
[セクション24.6.3 「関数に関連するパーティショニング制限」](#)

YEARWEEK()

[セクション12.7 「日付および時間関数」](#)
[セクション24.6.3 「関数に関連するパーティショニング制限」](#)

INFORMATION_SCHEMA の索引

[A](#) | [C](#) | [E](#) | [F](#) | [I](#) | [K](#) | [M](#) | [N](#) | [O](#) | [P](#) | [R](#) | [S](#) | [T](#) | [U](#) | [V](#)

A

[\[index top\]](#)

ADMINISTRABLE_ROLE_AUTHORIZATIONS

[セクション26.2 「INFORMATION_SCHEMA ADMINISTRABLE_ROLE_AUTHORIZATIONS テーブル」](#)

APPLICABLE_ROLES

[セクション26.3 「INFORMATION_SCHEMA APPLICABLE_ROLES テーブル」](#)

C

[\[index top\]](#)

CHARACTER_SETS

[セクション26.4 「INFORMATION_SCHEMA CHARACTER_SETS テーブル」](#)
[セクション14.5 「INFORMATION_SCHEMA とデータディクショナリの統合」](#)
[セクション10.2 「MySQL での文字セットと照合順序」](#)
[セクション13.7.7.3 「SHOW CHARACTER SET ステートメント」](#)
[セクション10.3.5 「カラム文字セットおよび照合順序」](#)
[セクション10.10 「サポートされる文字セットおよび照合順序」](#)
[セクション10.3.4 「テーブル文字セットおよび照合順序」](#)
[セクション10.3.3 「データベース文字セットおよび照合順序」](#)
[セクション10.3.8 「文字セットイントロデューサ」](#)
[セクション10.3.6 「文字列リテラルの文字セットおよび照合順序」](#)

CHECK_CONSTRAINTS

[セクション26.5 「INFORMATION_SCHEMA CHECK_CONSTRAINTS テーブル」](#)
[セクション14.5 「INFORMATION_SCHEMA とデータディクショナリの統合」](#)

COLLATION_CHARACTER_SET_APPLICABILITY

[セクション26.7 「INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY テーブル」](#)

セクション14.5「INFORMATION_SCHEMA とデータディクショナリの統合」
セクション8.2.3「INFORMATION_SCHEMA クエリーの最適化」

COLLATIONS

セクション11.3.2「CHAR および VARCHAR 型」
セクション26.6「INFORMATION_SCHEMA COLLATIONS テーブル」
セクション14.5「INFORMATION_SCHEMA とデータディクショナリの統合」
セクション10.2「MySQL での文字セットと照合順序」
セクション13.7.7.4「SHOW COLLATION ステートメント」
セクション10.10.1「Unicode 文字セット」
セクション10.8.5「バイナリ照合順序と_bin 照合順序」
セクション10.15「文字セットの構成」

COLUMN_PRIVILEGES

セクション26.10「INFORMATION_SCHEMA COLUMN_PRIVILEGES テーブル」

COLUMN_STATISTICS

セクション26.11「INFORMATION_SCHEMA COLUMN_STATISTICS テーブル」
セクション14.5「INFORMATION_SCHEMA とデータディクショナリの統合」
セクション8.9.6「オプティマイザ統計」

COLUMNS

セクション26.8「INFORMATION_SCHEMA COLUMNS テーブル」
セクション26.51.1「INFORMATION_SCHEMA INNODB_BUFFER_PAGE テーブル」
セクション26.51.2「INFORMATION_SCHEMA INNODB_BUFFER_PAGE_LRU テーブル」
セクション26.51.3「INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS テーブル」
セクション26.51.4「INFORMATION_SCHEMA INNODB_CACHED_INDEXES テーブル」
セクション26.51.5「INFORMATION_SCHEMA INNODB_CMP および INNODB_CMP_RESET テーブル」
セクション26.51.7「INFORMATION_SCHEMA INNODB_CMP_PER_INDEX および INNODB_CMP_PER_INDEX_RESET テーブル」
セクション26.51.6「INFORMATION_SCHEMA INNODB_CMPMEM および INNODB_CMPMEM_RESET テーブル」
セクション26.51.8「INFORMATION_SCHEMA INNODB_COLUMNS テーブル」
セクション26.51.9「INFORMATION_SCHEMA INNODB_DATAFILES テーブル」
セクション26.51.10「INFORMATION_SCHEMA INNODB_FIELDS テーブル」
セクション26.51.11「INFORMATION_SCHEMA INNODB_FOREIGN テーブル」
セクション26.51.12「INFORMATION_SCHEMA INNODB_FOREIGN_COLS テーブル」
セクション26.51.13「INFORMATION_SCHEMA INNODB_FT_BEING_DELETED テーブル」
セクション26.51.14「INFORMATION_SCHEMA INNODB_FT_CONFIG テーブル」
セクション26.51.15「INFORMATION_SCHEMA INNODB_FT_DEFAULT_STOPWORD テーブル」
セクション26.51.16「INFORMATION_SCHEMA INNODB_FT_DELETED テーブル」
セクション26.51.17「INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE テーブル」
セクション26.51.18「INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE テーブル」
セクション26.51.19「INFORMATION_SCHEMA INNODB_INDEXES テーブル」
セクション26.51.22「INFORMATION_SCHEMA INNODB_METRICS テーブル」
セクション26.51.23「INFORMATION_SCHEMA INNODB_SESSION_TEMP_TABLESPACES テーブル」
セクション26.51.24「INFORMATION_SCHEMA INNODB_TABLES テーブル」
セクション26.51.25「INFORMATION_SCHEMA INNODB_TABLESPACES テーブル」
セクション26.51.26「INFORMATION_SCHEMA INNODB_TABLESPACES_BRIEF テーブル」
セクション26.51.27「INFORMATION_SCHEMA INNODB_TABLESTATS ビュー」
セクション26.51.28「INFORMATION_SCHEMA INNODB_TEMP_TABLE_INFO テーブル」
セクション26.51.29「INFORMATION_SCHEMA INNODB_TRX テーブル」
セクション26.51.30「INFORMATION_SCHEMA INNODB_VIRTUAL テーブル」
セクション26.35「INFORMATION_SCHEMA ST_GEOMETRY_COLUMNS テーブル」
セクション14.5「INFORMATION_SCHEMA とデータディクショナリの統合」
セクション13.7.7.5「SHOW COLUMNS ステートメント」

COLUMNS_EXTENSIONS

セクション26.9「INFORMATION_SCHEMA COLUMNS_EXTENSIONS テーブル」

CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS

[セクション6.4.2.2 「Connection-Control のシステム変数とステータス変数」](#)

[セクション6.4.2 「Connection-Control プラグイン」](#)

[セクション6.4.2.1 「Connection-Control プラグインのインストール」](#)

[セクション26.53.1 「INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS テーブル」](#)

E

[\[index top\]](#)

ENABLED_ROLES

[セクション26.12 「INFORMATION_SCHEMA ENABLED_ROLES テーブル」](#)

ENGINES

[セクション26.13 「INFORMATION_SCHEMA ENGINES テーブル」](#)

[セクション13.7.7.16 「SHOW ENGINES ステートメント」](#)

[セクション23.5.16 「クイックリファレンス: NDB Cluster SQL ステートメント」](#)

[セクション5.1.8 「サーバーシステム変数」](#)

EVENTS

[セクション26.14 「INFORMATION_SCHEMA EVENTS テーブル」](#)

[セクション14.5 「INFORMATION_SCHEMA とデータディクショナリの統合」](#)

[セクション13.7.7.18 「SHOW EVENTS ステートメント」](#)

[セクション25.4.2 「イベントスケジューラの構成」](#)

[セクション25.4.4 「イベントメタデータ」](#)

[セクション17.1.2.8 「レプリケーション環境へのレプリカの追加」](#)

[セクション17.5.1.16 「呼び出される機能のレプリケーション」](#)

[既存のデータによるレプリケーションのセットアップ](#)

F

[\[index top\]](#)

FILES

[セクション26.15 「INFORMATION_SCHEMA FILES テーブル」](#)

[セクション26.51.9 「INFORMATION_SCHEMA INNODB_DATAFILES テーブル」](#)

[セクション26.51.25 「INFORMATION_SCHEMA INNODB_TABLESPACES テーブル」](#)

[セクション26.40 「INFORMATION_SCHEMA TABLESPACES テーブル」](#)

[セクション14.5 「INFORMATION_SCHEMA とデータディクショナリの統合」](#)

[セクション6.2.2 「MySQL で提供される権限」](#)

[セクション23.5.15 「NDB Cluster の INFORMATION_SCHEMA テーブル」](#)

[セクション23.5.10.1 「NDB Cluster ディスクデータオブジェクト」](#)

[セクション23.5.14 「ndbinfo: NDB Cluster 情報データベース」](#)

I

[\[index top\]](#)

INFORMATION_SCHEMA

[セクション14.5 「INFORMATION_SCHEMA とデータディクショナリの統合」](#)

[セクション15.15 「InnoDB INFORMATION_SCHEMA テーブル」](#)

[セクション2.11.4 「MySQL 8.0 での変更」](#)

[セクション1.3 「MySQL 8.0 の新機能」](#)

[第14章 「MySQL データディクショナリ」](#)

[セクション5.2 「MySQL データディレクトリ」](#)

[MySQL 用語集](#)

セクション23.5.15「NDB Cluster の INFORMATION_SCHEMA テーブル」
セクション28.2「sys スキーマの使用」
セクション14.1「データディクショナリスキーマ」
セクション6.2.9「予約済アカウント」

INFORMATION_SCHEMA.CHARACTER_SETS

セクションA.11「MySQL 8.0 FAQ: MySQL の中国語、日本語、および韓国語の文字セット」

INFORMATION_SCHEMA.COLLATIONS

セクション8.9.6「オブティマイザ統計」
セクション10.14.2「照合順序 ID の選択」

INFORMATION_SCHEMA.COLUMN_STATISTICS

セクション13.7.3.1「ANALYZE TABLE ステートメント」
セクション8.9.6「オブティマイザ統計」
セクション8.9「クエリーオブティマイザの制御」

INFORMATION_SCHEMA.COLUMNS

セクション23.4.14「ndb_index_stat — NDB インデックス統計ユーティリティー」
セクション2.11.6「Unix/Linux での MySQL バイナリまたはパッケージベースのインストールのアップグレード」
セクション5.1.8「サーバーシステム変数」
セクション27.1「パフォーマンススキーマクイックスタート」
セクション13.1.20.10「非表示カラム」

INFORMATION_SCHEMA.ENGINES

セクション15.1.3「InnoDB がデフォルトのストレージエンジンであるかどうかの確認」
セクション23.5.16「クイックリファレンス: NDB Cluster SQL ステートメント」
セクション27.1「パフォーマンススキーマクイックスタート」

INFORMATION_SCHEMA.EVENTS

セクション25.4.6「イベントスケジューラと MySQL 権限」
セクション25.4.4「イベントメタデータ」
セクション25.8「ストアプログラムの制約」
セクション17.5.1.16「呼び出される機能のレプリケーション」

INFORMATION_SCHEMA.FILES

セクション13.1.6「ALTER LOGFILE GROUP ステートメント」
セクション13.1.10「ALTER TABLESPACE ステートメント」
セクション13.1.16「CREATE LOGFILE GROUP ステートメント」
セクション13.1.21「CREATE TABLESPACE ステートメント」
セクション15.15.8「INFORMATION_SCHEMA.FILES からの InnoDB テーブルスペースメタデータの取得」
セクション2.11.4「MySQL 8.0 での変更」
セクション23.1.4「NDB Cluster の新機能」
セクション23.5.10.2「NDB Cluster データストレージの要件」
セクション23.4.9「ndb_desc — NDB テーブルの表示」
セクション23.5.14.21「ndbinfo dict_obj_info テーブル」
セクション23.5.14.22「ndbinfo dict_obj_tree テーブル」
セクション23.5.14.33「ndbinfo logbuffers テーブル」
セクション23.5.14.34「ndbinfo logspaces テーブル」
セクション15.6.3.4「undo テーブルスペース」
セクション15.6.3.6「サーバーがオフラインのときのテーブルスペースファイルの移動」
セクション5.6.7.3「リモートデータのクローニング」
セクション15.6.3.5「一時テーブルスペース」

INFORMATION_SCHEMA.INNODB_BUFFER_PAGE

セクション15.5.2「変更バッファ」

INFORMATION_SCHEMA.INNODB_CACHED_INDEXES

セクション1.3「MySQL 8.0の新機能」

INFORMATION_SCHEMA.INNODB_CMP

セクション15.9.1.3「InnoDB テーブルの圧縮の調整」

MySQL 用語集

セクション15.15.1.3「圧縮情報スキーマテーブルの使用」

INFORMATION_SCHEMA.INNODB_CMP_PER_INDEX

セクション15.14「InnoDB の起動オプションおよびシステム変数」

セクション15.9.1.3「InnoDB テーブルの圧縮の調整」

INFORMATION_SCHEMA.INNODB_CMPMEM

セクション15.15.1.3「圧縮情報スキーマテーブルの使用」

INFORMATION_SCHEMA.INNODB_COLUMNS

セクション15.12.1「オンライン DDL 操作」

INFORMATION_SCHEMA.INNODB_DATAFILES

セクション2.11.4「MySQL 8.0 での変更」

INFORMATION_SCHEMA.INNODB_FT_CONFIG

セクション15.6.2.4「InnoDB FULLTEXT インデックス」

INFORMATION_SCHEMA.INNODB_FT_DEFAULT_STOPWORD

セクション12.10.4「全文ストップワード」

INFORMATION_SCHEMA.INNODB_FT_INDEX_CACHE

セクション15.6.2.4「InnoDB FULLTEXT インデックス」

セクション12.10.9「MeCab フルテキストパーサープラグイン」

セクション12.10.8「ngram 全文パーサー」

INFORMATION_SCHEMA.INNODB_FT_INDEX_TABLE

セクション12.10.4「全文ストップワード」

INFORMATION_SCHEMA.INNODB_INDEXES

セクション15.6.2.4「InnoDB FULLTEXT インデックス」

INFORMATION_SCHEMA.INNODB_METRICS

セクション15.15.6「InnoDB INFORMATION_SCHEMA メトリックテーブル」

セクション15.14「InnoDB の起動オプションおよびシステム変数」

セクション15.6.3.4「undo テーブルスペース」

セクション15.5.2「変更バッファ」

INFORMATION_SCHEMA.INNODB_TABLES

セクション15.15.3「InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル」

セクション15.10「InnoDB の行フォーマット」

セクション15.12.1「オンライン DDL 操作」

セクション24.3.3「パーティションとサブパーティションをテーブルと交換する」

INFORMATION_SCHEMA.INNODB_TABLESPACES

セクション15.15.3「InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル」

セクション15.9.2「InnoDB ページ圧縮」

セクション15.13「InnoDB 保存データ暗号化」

セクション1.3 「MySQL 8.0 の新機能」
セクション15.6.3.4 「undo テーブルスペース」
セクション15.6.3.9 「テーブルスペースの AUTOEXTEND_SIZE 構成」

INFORMATION_SCHEMA.INNODB_TABLESPACES_BRIEF

セクション1.3 「MySQL 8.0 の新機能」

INFORMATION_SCHEMA.INNODB_TABLESTATS

セクション13.7.3.1 「ANALYZE TABLE ステートメント」
セクション15.15.3 「InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル」

INFORMATION_SCHEMA.INNODB_TEMP_TABLE_INFO

セクション15.6.3.5 「一時テーブルスペース」

INFORMATION_SCHEMA.INNODB_TRX

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.7.6 「トランザクションスケジューリング」

INFORMATION_SCHEMA.KEY_COLUMN_USAGE

セクション1.7.3.2 「FOREIGN KEY の制約」
セクション13.1.20.5 「FOREIGN KEY の制約」

INFORMATION_SCHEMA.MYSQL_FIREWALL_USERS

セクション6.4.7.4 「MySQL Enterprise Firewall リファレンス」

INFORMATION_SCHEMA.MYSQL_FIREWALL_WHITELIST

セクション6.4.7.4 「MySQL Enterprise Firewall リファレンス」

INFORMATION_SCHEMA.OPTIMIZER_TRACE

セクション8.2.1.2 「range の最適化」

INFORMATION_SCHEMA.PARTITIONS

セクション24.2.5 「KEY パーティショニング」
セクション24.2.7 「MySQL パーティショニングによる NULL の扱い」
セクション24.2.3.1 「RANGE COLUMNS パーティショニング」
セクション24.3.3 「パーティションとサブパーティションをテーブルと交換する」
セクション24.3.5 「パーティションに関する情報を取得する」

INFORMATION_SCHEMA.PLUGINS

セクション6.4.2.1 「Connection-Control プラグインのインストール」
セクション5.6.5.1 「ddl_rewriter のインストールまたはアンインストール」
セクション15.13 「InnoDB 保存データ暗号化」
セクション6.4.1.7 「LDAP プラガブル認証」
セクションA.2 「MySQL 8.0 FAQ: ストレージエンジン」
セクション1.3 「MySQL 8.0 の新機能」
セクション6.4.5.2 「MySQL Enterprise Audit のインストールまたはアンインストール」
セクション6.4.1.5 「PAM プラガブル認証」
セクション6.4.1.6 「Windows プラガブル認証」
セクション6.4.4.1 「キーリングプラグインのインストール」
セクション23.5.16 「クイックリファレンス: NDB Cluster SQL ステートメント」
セクション5.6.7.1 「クローンプラグインのインストール」
セクション5.6.2 「サーバープラグイン情報の取得」
セクション5.6.3.2 「スレッドプールのインストール」
セクション6.4.1.9 「ソケットピア資格証明プラガブル認証」
セクション6.4.1.10 「プラガブル認証のテスト」
セクション5.6.1 「プラグインのインストールおよびアンインストール」

[セクション5.6.7.3「リモートデータのクローニング」](#)
[セクション6.4.1.8「ログインなしのプラグブル認証」](#)
[セクション17.4.10.2「準同期レプリケーションのインストールと構成」](#)

INFORMATION_SCHEMA.PROCESSLIST

[セクション6.2.2「MySQL で提供される権限」](#)
[セクション27.6「パフォーマンススキーマインストール命名規則」](#)
[セクション27.12.5「パフォーマンススキーマステージイベントテーブル」](#)
[セクション12.16「情報関数」](#)

INFORMATION_SCHEMA.RESOURCE_GROUPS

[セクション5.1.16「リソースグループ」](#)

INFORMATION_SCHEMA.ROUTINES

[セクションA.4「MySQL 8.0 FAQ: ストアドプロシージャおよびストアドファンクション」](#)
[セクション6.2.2「MySQL で提供される権限」](#)
[セクション25.2.2「ストアドルーチンと MySQL 権限」](#)

INFORMATION_SCHEMA.SCHEMATA

[セクション15.13「InnoDB 保存データ暗号化」](#)
[セクション10.3.3「データベース文字セットおよび照合順序」](#)

INFORMATION_SCHEMA.STATISTICS

[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)
[セクション5.3「mysql システムスキーマ」](#)
[セクション23.4.14「ndb_index_stat — NDB インデックス統計ユーティリティ」](#)
[セクション8.9.4「インデックスヒント」](#)
[セクション8.3.12「不可視のインデックス」](#)
[セクション15.8.10.2「非永続的オプティマイザ統計のパラメータの構成」](#)

INFORMATION_SCHEMA.TABLE_CONSTRAINTS

[セクション15.12.1「オンライン DDL 操作」](#)

INFORMATION_SCHEMA.TABLES

[セクション13.1.9「ALTER TABLE ステートメント」](#)
[セクション28.4.4.2「diagnostics\(\) プロシージャ」](#)
[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)
[セクション15.9.2「InnoDB ページ圧縮」](#)
[セクション15.13「InnoDB 保存データ暗号化」](#)
[セクション23.5.7.3「NDB Cluster データノードのオンラインでの追加: 詳細な例」](#)
[セクション28.4.2.1「sys_config テーブル」](#)
[セクション5.6.3.2「スレッドプールのインストール」](#)
[セクション14.2「ファイルベースのメタデータ記憶域の削除」](#)
[セクション15.8.10.2「非永続的オプティマイザ統計のパラメータの構成」](#)

INFORMATION_SCHEMA.TRIGGERS

[セクションA.5「MySQL 8.0 FAQ: トリガー」](#)

INFORMATION_SCHEMA.USER_ATTRIBUTES

[セクション13.7.1.1「ALTER USER ステートメント」](#)
[セクション13.7.1.3「CREATE USER ステートメント」](#)
[セクション1.3「MySQL 8.0 の新機能」](#)

INFORMATION_SCHEMA.VIEWS

[セクション2.11.5「アップグレード用のインストールの準備」](#)
[セクション25.5.3「更新可能および挿入可能なビュー」](#)

INNODB_BUFFER_PAGE

セクション26.51.1 「INFORMATION_SCHEMA INNODB_BUFFER_PAGE テーブル」
セクション26.51.2 「INFORMATION_SCHEMA INNODB_BUFFER_PAGE_LRU テーブル」
セクション15.15.5 「InnoDB INFORMATION_SCHEMA バッファプールテーブル」
セクション28.4.3.7 「innodb_buffer_stats_by_schema および x\$innodb_buffer_stats_by_schema のビュー」
セクション28.4.3.8 「innodb_buffer_stats_by_table および x\$innodb_buffer_stats_by_table のビュー」
セクション28.1 「sys スキーマを使用するための前提条件」
セクション15.5.2 「変更バッファ」

INNODB_BUFFER_PAGE_LRU

セクション26.51.2 「INFORMATION_SCHEMA INNODB_BUFFER_PAGE_LRU テーブル」
セクション15.15.5 「InnoDB INFORMATION_SCHEMA バッファプールテーブル」
セクション15.8.3.6 「バッファプールの状態の保存と復元」

INNODB_BUFFER_POOL_STATS

セクション26.51.3 「INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS テーブル」
セクション15.15.5 「InnoDB INFORMATION_SCHEMA バッファプールテーブル」
セクション15.5.1 「バッファプール」

INNODB_CACHED_INDEXES

セクション26.51.4 「INFORMATION_SCHEMA INNODB_CACHED_INDEXES テーブル」

INNODB_CMP

セクション26.51.5 「INFORMATION_SCHEMA INNODB_CMP および INNODB_CMP_RESET テーブル」
セクション15.15.1.1 「INNODB_CMP および INNODB_CMP_RESET」
セクション15.15.1.2 「INNODB_CMPMEM および INNODB_CMPMEM_RESET」
セクション15.15.1 「圧縮に関する InnoDB INFORMATION_SCHEMA テーブル」
セクション15.15.1.3 「圧縮情報スキーマテーブルの使用」
セクション15.9.1.4 「実行時の InnoDB テーブル圧縮の監視」

INNODB_CMP_PER_INDEX

セクション26.51.7 「INFORMATION_SCHEMA INNODB_CMP_PER_INDEX および
INNODB_CMP_PER_INDEX_RESET テーブル」
セクション15.15.1.3 「圧縮情報スキーマテーブルの使用」
セクション15.9.1.4 「実行時の InnoDB テーブル圧縮の監視」

INNODB_CMP_PER_INDEX_RESET

セクション26.51.7 「INFORMATION_SCHEMA INNODB_CMP_PER_INDEX および
INNODB_CMP_PER_INDEX_RESET テーブル」

INNODB_CMP_RESET

セクション26.51.5 「INFORMATION_SCHEMA INNODB_CMP および INNODB_CMP_RESET テーブル」
セクション15.15.1.1 「INNODB_CMP および INNODB_CMP_RESET」
セクション15.15.1.2 「INNODB_CMPMEM および INNODB_CMPMEM_RESET」
セクション15.15.1 「圧縮に関する InnoDB INFORMATION_SCHEMA テーブル」

INNODB_CMPMEM

セクション26.51.6 「INFORMATION_SCHEMA INNODB_CMPMEM および INNODB_CMPMEM_RESET テーブル」
セクション15.15.1.2 「INNODB_CMPMEM および INNODB_CMPMEM_RESET」
セクション15.15.1 「圧縮に関する InnoDB INFORMATION_SCHEMA テーブル」
セクション15.15.1.3 「圧縮情報スキーマテーブルの使用」

INNODB_CMPMEM_RESET

セクション26.51.6 「INFORMATION_SCHEMA INNODB_CMPMEM および INNODB_CMPMEM_RESET テーブル」
セクション15.15.1.2 「INNODB_CMPMEM および INNODB_CMPMEM_RESET」

[セクション15.15.1「圧縮に関する InnoDB INFORMATION_SCHEMA テーブル」](#)

INNODB_COLUMNS

[セクション26.51.8「INFORMATION_SCHEMA INNODB_COLUMNS テーブル」](#)

[セクション26.51.30「INFORMATION_SCHEMA INNODB_VIRTUAL テーブル」](#)

[セクション14.5「INFORMATION_SCHEMA とデータディクショナリの統合」](#)

[セクション15.15.3「InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル」](#)

INNODB_DATAFILES

[セクション26.15「INFORMATION_SCHEMA FILES テーブル」](#)

[セクション26.51.9「INFORMATION_SCHEMA INNODB_DATAFILES テーブル」](#)

[セクション26.51.26「INFORMATION_SCHEMA INNODB_TABLESPACES_BRIEF テーブル」](#)

[セクション26.40「INFORMATION_SCHEMA TABLESPACES テーブル」](#)

[セクション14.5「INFORMATION_SCHEMA とデータディクショナリの統合」](#)

[セクション15.15.8「INFORMATION_SCHEMA.FILES から InnoDB テーブルスペースメタデータの取得」](#)

[セクション15.15.3「InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル」](#)

INNODB_FIELDS

[セクション26.51.10「INFORMATION_SCHEMA INNODB_FIELDS テーブル」](#)

[セクション14.5「INFORMATION_SCHEMA とデータディクショナリの統合」](#)

[セクション15.15.3「InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル」](#)

INNODB_FOREIGN

[セクション1.7.3.2「FOREIGN KEY の制約」](#)

[セクション13.1.20.5「FOREIGN KEY の制約」](#)

[セクション26.51.11「INFORMATION_SCHEMA INNODB_FOREIGN テーブル」](#)

[セクション14.5「INFORMATION_SCHEMA とデータディクショナリの統合」](#)

[セクション15.15.3「InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル」](#)

INNODB_FOREIGN_COLS

[セクション1.7.3.2「FOREIGN KEY の制約」](#)

[セクション13.1.20.5「FOREIGN KEY の制約」](#)

[セクション26.51.12「INFORMATION_SCHEMA INNODB_FOREIGN_COLS テーブル」](#)

[セクション14.5「INFORMATION_SCHEMA とデータディクショナリの統合」](#)

[セクション15.15.3「InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル」](#)

INNODB_FT_BEING_DELETED

[セクション26.51.13「INFORMATION_SCHEMA INNODB_FT_BEING_DELETED テーブル」](#)

[セクション15.6.2.4「InnoDB FULLTEXT インデックス」](#)

[セクション15.15.4「InnoDB INFORMATION_SCHEMA FULLTEXT インデックステーブル」](#)

[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)

INNODB_FT_CONFIG

[セクション26.51.14「INFORMATION_SCHEMA INNODB_FT_CONFIG テーブル」](#)

[セクション15.6.2.4「InnoDB FULLTEXT インデックス」](#)

[セクション15.15.4「InnoDB INFORMATION_SCHEMA FULLTEXT インデックステーブル」](#)

[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)

INNODB_FT_DEFAULT_STOPWORD

[セクション26.51.15「INFORMATION_SCHEMA INNODB_FT_DEFAULT_STOPWORD テーブル」](#)

[セクション15.6.2.4「InnoDB FULLTEXT インデックス」](#)

[セクション15.15.4「InnoDB INFORMATION_SCHEMA FULLTEXT インデックステーブル」](#)

[セクション12.10.4「全文ストップワード」](#)

INNODB_FT_DELETED

[セクション26.51.13「INFORMATION_SCHEMA INNODB_FT_BEING_DELETED テーブル」](#)

[セクション26.51.16「INFORMATION_SCHEMA INNODB_FT_DELETED テーブル」](#)

[セクション15.6.2.4 「InnoDB FULLTEXT インデックス」](#)
[セクション15.15.4 「InnoDB INFORMATION_SCHEMA FULLTEXT インデックステーブル」](#)
[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)

INNODB_FT_INDEX_CACHE

[セクション26.51.17 「INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE テーブル」](#)
[セクション15.6.2.4 「InnoDB FULLTEXT インデックス」](#)
[セクション15.15.4 「InnoDB INFORMATION_SCHEMA FULLTEXT インデックステーブル」](#)
[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)

INNODB_FT_INDEX_TABLE

[セクション26.51.13 「INFORMATION_SCHEMA INNODB_FT_BEING_DELETED テーブル」](#)
[セクション26.51.16 「INFORMATION_SCHEMA INNODB_FT_DELETED テーブル」](#)
[セクション26.51.18 「INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE テーブル」](#)
[セクション15.6.2.4 「InnoDB FULLTEXT インデックス」](#)
[セクション15.15.4 「InnoDB INFORMATION_SCHEMA FULLTEXT インデックステーブル」](#)
[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)

INNODB_INDEXES

[セクション26.51.4 「INFORMATION_SCHEMA INNODB_CACHED_INDEXES テーブル」](#)
[セクション26.51.19 「INFORMATION_SCHEMA INNODB_INDEXES テーブル」](#)
[セクション14.5 「INFORMATION_SCHEMA とデータディクショナリの統合」](#)
[セクション15.6.2.4 「InnoDB FULLTEXT インデックス」](#)
[セクション15.15.3 「InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル」](#)
[セクション15.8.11 「インデックスページのマージしきい値の構成」](#)

INNODB_LOCK_WAITS

[セクション26.51.21 「INFORMATION_SCHEMA INNODB_LOCK_WAITS テーブル」](#)
[セクション15.15.2 「InnoDB INFORMATION_SCHEMA トランザクションおよびロック情報」](#)
[セクション15.15.2.2 「InnoDB のロックおよびロック待機情報」](#)
[セクション15.15.2.3 「InnoDB トランザクションおよびロック情報の永続性と一貫性」](#)
[セクション15.15.2.1 「InnoDB トランザクションの使用および情報のロック」](#)
[セクション1.3 「MySQL 8.0 の新機能」](#)

INNODB_LOCKS

[セクション26.51.20 「INFORMATION_SCHEMA INNODB_LOCKS テーブル」](#)
[セクション15.15.2 「InnoDB INFORMATION_SCHEMA トランザクションおよびロック情報」](#)
[セクション15.15.2.2 「InnoDB のロックおよびロック待機情報」](#)
[セクション15.15.2.3 「InnoDB トランザクションおよびロック情報の永続性と一貫性」](#)
[セクション15.15.2.1 「InnoDB トランザクションの使用および情報のロック」](#)
[セクション1.3 「MySQL 8.0 の新機能」](#)

INNODB_METRICS

[セクション13.7.3.1 「ANALYZE TABLE ステートメント」](#)
[セクション26.51.22 「INFORMATION_SCHEMA INNODB_METRICS テーブル」](#)
[セクション15.15.6 「InnoDB INFORMATION_SCHEMA メトリックテーブル」](#)
[セクション1.3 「MySQL 8.0 の新機能」](#)
MySQL 用語集
[セクション15.8.11 「インデックスページのマージしきい値の構成」](#)
[セクション15.7.6 「トランザクションスケジューリング」](#)
[セクション28.4.3.21 「メトリックビュー」](#)
[セクション15.5.2 「変更バッファ」](#)

INNODB_SESSION_TEMP_TABLESPACES

[セクション26.51.23 「INFORMATION_SCHEMA INNODB_SESSION_TEMP_TABLESPACES テーブル」](#)
[セクション1.3 「MySQL 8.0 の新機能」](#)
[セクション15.6.3.5 「一時テーブルスペース」](#)

INNODB_TABLES

[セクション26.51.4 「INFORMATION_SCHEMA INNODB_CACHED_INDEXES テーブル」](#)
[セクション26.51.24 「INFORMATION_SCHEMA INNODB_TABLES テーブル」](#)
[セクション14.5 「INFORMATION_SCHEMA とデータディクショナリの統合」](#)
[セクション15.6.2.4 「InnoDB FULLTEXT インデックス」](#)
[セクション15.15.3 「InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル」](#)
[セクション1.3 「MySQL 8.0 の新機能」](#)

INNODB_TABLESPACES

[セクション26.15 「INFORMATION_SCHEMA FILES テーブル」](#)
[セクション26.51.25 「INFORMATION_SCHEMA INNODB_TABLESPACES テーブル」](#)
[セクション26.51.26 「INFORMATION_SCHEMA INNODB_TABLESPACES_BRIEF テーブル」](#)
[セクション26.38 「INFORMATION_SCHEMA TABLES テーブル」](#)
[セクション26.40 「INFORMATION_SCHEMA TABLESPACES テーブル」](#)
[セクション14.5 「INFORMATION_SCHEMA とデータディクショナリの統合」](#)
[セクション15.15.8 「INFORMATION_SCHEMA.FILES からの InnoDB テーブルスペースメタデータの取得」](#)
[セクション15.15.3 「InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル」](#)
[セクション1.3 「MySQL 8.0 の新機能」](#)
[セクション13.7.7.38 「SHOW TABLE STATUS ステートメント」](#)

INNODB_TABLESPACES_BRIEF

[セクション26.51.26 「INFORMATION_SCHEMA INNODB_TABLESPACES_BRIEF テーブル」](#)
[セクション14.5 「INFORMATION_SCHEMA とデータディクショナリの統合」](#)
[セクション15.15.3 「InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル」](#)

INNODB_TABLESTATS

[セクション26.51.27 「INFORMATION_SCHEMA INNODB_TABLESTATS ビュー」](#)
[セクション14.5 「INFORMATION_SCHEMA とデータディクショナリの統合」](#)
[セクション15.15.3 「InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル」](#)

INNODB_TEMP_TABLE_INFO

[セクション26.51.28 「INFORMATION_SCHEMA INNODB_TEMP_TABLE_INFO テーブル」](#)
[セクション15.15.7 「InnoDB INFORMATION_SCHEMA 一時テーブル情報テーブル」](#)

INNODB_TRX

[セクション27.12.13.1 「data_locks テーブル」](#)
[セクション26.51.29 「INFORMATION_SCHEMA INNODB_TRX テーブル」](#)
[セクション15.15.2 「InnoDB INFORMATION_SCHEMA トランザクションおよびロック情報」](#)
[セクション15.15.2.2 「InnoDB のロックおよびロック待機情報」](#)
[セクション15.15.2.3 「InnoDB トランザクションおよびロック情報の永続性と一貫性」](#)
[セクション15.15.2.1 「InnoDB トランザクションの使用および情報のロック」](#)

INNODB_VIRTUAL

[セクション26.51.30 「INFORMATION_SCHEMA INNODB_VIRTUAL テーブル」](#)

K

[\[index top\]](#)

KEY_COLUMN_USAGE

[セクション26.16 「INFORMATION_SCHEMA KEY_COLUMN_USAGE テーブル」](#)
[セクション14.5 「INFORMATION_SCHEMA とデータディクショナリの統合」](#)
[セクション5.3 「mysql システムスキーマ」](#)

KEYWORDS

[セクション26.18 「INFORMATION_SCHEMA KEYWORDS テーブル」](#)

M

[\[index top\]](#)

MYSQL_FIREWALL_USERS

セクション26.54.1 「INFORMATION_SCHEMA MYSQL_FIREWALL_USERS テーブル」

MYSQL_FIREWALL_WHITELIST

セクション26.54.2 「INFORMATION_SCHEMA MYSQL_FIREWALL_WHITELIST テーブル」

N

[\[index top\]](#)

NDB_TRANSID_MYSQL_CONNECTION_MAP

セクション23.5.14.45 「ndbinfo server_operations テーブル」

セクション23.5.14.46 「ndbinfo server_transactions テーブル」

ndb_transid_mysql_connection_map

セクション23.5.15 「NDB Cluster の INFORMATION_SCHEMA テーブル」

NDB Cluster の MySQL Server オプション

セクション23.5.14 「ndbinfo: NDB Cluster 情報データベース」

O

[\[index top\]](#)

OPTIMIZER_TRACE

セクション26.19 「INFORMATION_SCHEMA OPTIMIZER_TRACE テーブル」

P

[\[index top\]](#)

PARAMETERS

セクション26.20 「INFORMATION_SCHEMA PARAMETERS テーブル」

セクション26.30 「INFORMATION_SCHEMA ROUTINES テーブル」

セクション14.5 「INFORMATION_SCHEMA とデータディクショナリの統合」

セクションA.4 「MySQL 8.0 FAQ: ストアドプロシージャおよびストアドファンクション」

セクション13.7.7.28 「SHOW PROCEDURE STATUS ステートメント」

PARTITIONS

セクション26.21 「INFORMATION_SCHEMA PARTITIONS テーブル」

セクション26.38 「INFORMATION_SCHEMA TABLES テーブル」

セクション14.5 「INFORMATION_SCHEMA とデータディクショナリの統合」

セクション24.2.7 「MySQL パーティショニングによる NULL の扱い」

セクション13.7.7.38 「SHOW TABLE STATUS ステートメント」

セクション24.3.3 「パーティションとサブパーティションをテーブルと交換する」

セクション24.3.5 「パーティションに関する情報を取得する」

第24章 「パーティション化」

PLUGINS

セクション26.22 「INFORMATION_SCHEMA PLUGINS テーブル」

セクション13.7.4.4 「INSTALL PLUGIN ステートメント」

セクション23.5.16 「クイックリファレンス: NDB Cluster SQL ステートメント」

セクション5.6.2 「サーバープラグイン情報の取得」

PROCESLIST

[セクション27.12.19.1「error_log テーブル」](#)
[セクション26.51.29「INFORMATION_SCHEMA INNODB_TRX テーブル」](#)
[セクション26.23「INFORMATION_SCHEMA PROCESLIST テーブル」](#)
[セクション15.15.2.3「InnoDB トランザクションおよびロック情報の永続性と一貫性」](#)
[セクション15.15.2.1「InnoDB トランザクションの使用および情報のロック」](#)
[セクション13.7.8.4「KILL ステートメント」](#)
[セクション27.12.19.9「processlist テーブル」](#)
[セクション28.4.3.22「processlist ビューと x\\$processlist ビュー」](#)
[セクション13.7.7.29「SHOW PROCESLIST ステートメント」](#)
[セクション5.6.7.10「クローニング操作の停止」](#)
[セクション27.12.19.10「スレッドテーブル」](#)
[セクション27.12.11「パフォーマンススキーマレプリケーションテーブル」](#)
[セクション8.14.1「プロセスリストへのアクセス」](#)
[セクション8.8.4「名前付き接続の実行計画情報の取得」](#)

PROFILING

[セクション26.24「INFORMATION_SCHEMA PROFILING テーブル」](#)
[セクション13.7.7.30「SHOW PROFILE ステートメント」](#)

R

[\[index top\]](#)

REFERENTIAL_CONSTRAINTS

[セクション26.25「INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS テーブル」](#)
[セクション14.5「INFORMATION_SCHEMA とデータディクショナリの統合」](#)
[セクション5.3「mysql システムスキーマ」](#)

RESOURCE_GROUPS

[セクション26.26「INFORMATION_SCHEMA RESOURCE_GROUPS テーブル」](#)
[セクション14.5「INFORMATION_SCHEMA とデータディクショナリの統合」](#)
[セクション5.1.16「リソースグループ」](#)

ROLE_COLUMN_GRANTS

[セクション26.27「INFORMATION_SCHEMA ROLE_COLUMN_GRANTS テーブル」](#)

ROLE_ROUTINE_GRANTS

[セクション26.28「INFORMATION_SCHEMA ROLE_ROUTINE_GRANTS テーブル」](#)

ROLE_TABLE_GRANTS

[セクション26.29「INFORMATION_SCHEMA ROLE_TABLE_GRANTS テーブル」](#)

ROUTINES

[セクション26.30「INFORMATION_SCHEMA ROUTINES テーブル」](#)
[セクション14.5「INFORMATION_SCHEMA とデータディクショナリの統合」](#)
[セクションA.4「MySQL 8.0 FAQ: ストアドプロシージャおよびストアドファンクション」](#)
[セクション13.7.7.28「SHOW PROCEDURE STATUS ステートメント」](#)
[セクション26.1「はじめに」](#)
[セクション25.6「ストアドオブジェクトのアクセス制御」](#)
[セクション25.2.3「ストアドルーチンのメタデータ」](#)

S

[\[index top\]](#)

SCHEMA_PRIVILEGES

[セクション26.33 「INFORMATION_SCHEMA SCHEMA_PRIVILEGES テーブル」](#)

SCHEMATA

[セクション26.31 「INFORMATION_SCHEMA SCHEMATA テーブル」](#)

[セクション26.32 「INFORMATION_SCHEMA SCHEMATA_EXTENSIONS テーブル」](#)

[セクション14.5 「INFORMATION_SCHEMA とデータディクショナリの統合」](#)

[セクション6.2.2 「MySQL で提供される権限」](#)

[セクション13.7.7.14 「SHOW DATABASES ステートメント」](#)

[セクション5.1.8 「サーバーシステム変数」](#)

[セクション6.2.3 「付与テーブル」](#)

SCHEMATA_EXTENSIONS

[セクション13.1.2 「ALTER DATABASE ステートメント」](#)

[セクション26.31 「INFORMATION_SCHEMA SCHEMATA テーブル」](#)

[セクション26.32 「INFORMATION_SCHEMA SCHEMATA_EXTENSIONS テーブル」](#)

ST_GEOMETRY_COLUMNS

[セクション26.8 「INFORMATION_SCHEMA COLUMNS テーブル」](#)

[セクション26.35 「INFORMATION_SCHEMA ST_GEOMETRY_COLUMNS テーブル」](#)

[セクション14.5 「INFORMATION_SCHEMA とデータディクショナリの統合」](#)

ST_SPATIAL_REFERENCE_SYSTEMS

[セクション13.1.19 「CREATE SPATIAL REFERENCE SYSTEM ステートメント」](#)

[セクション26.36 「INFORMATION_SCHEMA ST_SPATIAL_REFERENCE_SYSTEMS テーブル」](#)

[セクション14.5 「INFORMATION_SCHEMA とデータディクショナリの統合」](#)

[セクション11.4.5 「空間参照システムのサポート」](#)

ST_UNITS_OF_MEASURE

[セクション26.37 「INFORMATION_SCHEMA ST_UNITS_OF_MEASURE テーブル」](#)

[セクション12.17.7.3 「LineString および MultiLineString のプロパティ関数」](#)

[セクション12.17.9.1 「オブジェクト形状を使用する空間関係関数」](#)

STATISTICS

[セクション13.7.3.1 「ANALYZE TABLE ステートメント」](#)

[セクション26.34 「INFORMATION_SCHEMA STATISTICS テーブル」](#)

[セクション14.5 「INFORMATION_SCHEMA とデータディクショナリの統合」](#)

[セクション13.7.7.22 「SHOW INDEX ステートメント」](#)

[セクション14.7 「データディクショナリの使用方法の違い」](#)

T

[\[index top\]](#)

TABLE_CONSTRAINTS

[セクション26.25 「INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS テーブル」](#)

[セクション26.42 「INFORMATION_SCHEMA TABLE_CONSTRAINTS テーブル」](#)

[セクション14.5 「INFORMATION_SCHEMA とデータディクショナリの統合」](#)

TABLE_CONSTRAINTS_EXTENSIONS

[セクション26.43 「INFORMATION_SCHEMA TABLE_CONSTRAINTS_EXTENSIONS テーブル」](#)

TABLE_PRIVILEGES

[セクション26.44 「INFORMATION_SCHEMA TABLE_PRIVILEGES テーブル」](#)

TABLES

[セクション13.1.20「CREATE TABLE ステートメント」](#)
[セクション26.38「INFORMATION_SCHEMA TABLES テーブル」](#)
[セクション14.5「INFORMATION_SCHEMA とデータディクショナリの統合」](#)
[セクション13.1.20.11「NDB_TABLE オプションの設定」](#)
[セクション13.7.7.38「SHOW TABLE STATUS ステートメント」](#)
[セクション13.7.7.39「SHOW TABLES ステートメント」](#)
[セクション26.1「はじめに」](#)
[セクション14.7「データディクショナリの使用法の違い」](#)

TABLES_EXTENSIONS

[セクション26.39「INFORMATION_SCHEMA TABLES_EXTENSIONS テーブル」](#)

TABLESPACES_EXTENSIONS

[セクション26.41「INFORMATION_SCHEMA TABLESPACES_EXTENSIONS テーブル」](#)

TP_THREAD_GROUP_STATE

[セクション26.52「INFORMATION_SCHEMA スレッドプールテーブル」](#)

TP_THREAD_GROUP_STATS

[セクション26.52「INFORMATION_SCHEMA スレッドプールテーブル」](#)

TP_THREAD_STATE

[セクション26.52「INFORMATION_SCHEMA スレッドプールテーブル」](#)
[セクション5.6.3.2「スレッドプールのインストール」](#)

TRIGGERS

[セクション26.45「INFORMATION_SCHEMA TRIGGERS テーブル」](#)
[セクション14.5「INFORMATION_SCHEMA とデータディクショナリの統合」](#)
[セクション13.7.7.11「SHOW CREATE TRIGGER ステートメント」](#)
[セクション13.7.7.40「SHOW TRIGGERS ステートメント」](#)
[セクション2.11.5「アップグレード用のインストールの準備」](#)
[セクション25.3.2「トリガーのメタデータ」](#)

U

[\[index top\]](#)

USER_ATTRIBUTES

[セクション13.7.1.1「ALTER USER ステートメント」](#)
[セクション13.7.1.3「CREATE USER ステートメント」](#)
[セクション26.46「INFORMATION_SCHEMA USER_ATTRIBUTES テーブル」](#)

USER_PRIVILEGES

[セクション26.47「INFORMATION_SCHEMA USER_PRIVILEGES テーブル」](#)
[セクション6.2.2「MySQL で提供される権限」](#)

V

[\[index top\]](#)

VIEW_ROUTINE_USAGE

[セクション26.49「INFORMATION_SCHEMA VIEW_ROUTINE_USAGE テーブル」](#)
[セクション14.5「INFORMATION_SCHEMA とデータディクショナリの統合」](#)

VIEW_TABLE_USAGE

[セクション26.50「INFORMATION_SCHEMA VIEW_TABLE_USAGE テーブル」](#)

[セクション14.5「INFORMATION_SCHEMA とデータディクショナリの統合」](#)

VIEWS

[セクション26.48「INFORMATION_SCHEMA VIEWS テーブル」](#)

[セクション14.5「INFORMATION_SCHEMA とデータディクショナリの統合」](#)

[セクション13.7.7.13「SHOW CREATE VIEW ステートメント」](#)

[セクション25.5.5「ビューのメタデータ」](#)

結合型の索引

[記号](#) | [A](#) | [C](#) | [E](#) | [F](#) | [I](#) | [R](#) | [S](#) | [U](#)

記号

[\[index top\]](#)

インデックス

[セクション8.8.2「EXPLAIN 出力フォーマット」](#)

A

[\[index top\]](#)

ALL

[セクション8.2.1.12「Block Nested Loop 結合と Batched Key Access 結合」](#)

[セクション8.8.2「EXPLAIN 出力フォーマット」](#)

[セクション8.2.1.7「Nested Loop 結合アルゴリズム」](#)

[セクション8.2.1.23「全テーブルスキャンの回避」](#)

C

[\[index top\]](#)

const

[セクション8.8.2「EXPLAIN 出力フォーマット」](#)

[NDB Cluster システム変数](#)

[セクション8.2.1.16「ORDER BY の最適化」](#)

[セクション8.2.1.2「range の最適化」](#)

[セクション13.2.10「SELECT ステートメント」](#)

[セクション8.9.3「オプティマイザヒント」](#)

[セクション8.8.3「拡張 EXPLAIN 出力形式」](#)

E

[\[index top\]](#)

eq_ref

[セクション8.2.1.12「Block Nested Loop 結合と Batched Key Access 結合」](#)

[セクション27.12.4.1「events_waits_current テーブル」](#)

[セクション8.2.2.3「EXISTS 戦略を使用したサブクエリーの最適化」](#)

[セクション8.8.2「EXPLAIN 出力フォーマット」](#)

[セクション16.7.1「MERGE テーブルの長所と短所」](#)

[NDB Cluster システム変数](#)

[セクション8.2.1.6 「インデックスコンディションプッシュダウンの最適化」](#)

F

[\[index top\]](#)

fulltext

[セクション8.8.2 「EXPLAIN 出力フォーマット」](#)

I

[\[index top\]](#)

index

[セクション8.2.1.12 「Block Nested Loop 結合と Batched Key Access 結合」](#)

[セクション8.8.2 「EXPLAIN 出力フォーマット」](#)

[セクション8.2.1.7 「Nested Loop 結合アルゴリズム」](#)

index_merge

[セクション8.8.2 「EXPLAIN 出力フォーマット」](#)

[セクション8.2.1.3 「インデックスマージの最適化」](#)

index_subquery

[セクション8.2.2.3 「EXISTS 戦略を使用したサブクエリーの最適化」](#)

[セクション8.8.2 「EXPLAIN 出力フォーマット」](#)

R

[\[index top\]](#)

range

[セクション8.2.1.12 「Block Nested Loop 結合と Batched Key Access 結合」](#)

[セクション8.8.2 「EXPLAIN 出力フォーマット」](#)

[セクション8.2.1.17 「GROUP BY の最適化」](#)

[セクション8.2.1.7 「Nested Loop 結合アルゴリズム」](#)

[セクション8.2.1.2 「range の最適化」](#)

[セクション8.2.1.6 「インデックスコンディションプッシュダウンの最適化」](#)

[セクション8.2.1.3 「インデックスマージの最適化」](#)

[セクション8.9.3 「オプティマイザヒント」](#)

ref

[セクション8.2.1.12 「Block Nested Loop 結合と Batched Key Access 結合」](#)

[セクション8.2.2.3 「EXISTS 戦略を使用したサブクエリーの最適化」](#)

[セクション8.8.2 「EXPLAIN 出力フォーマット」](#)

[セクション8.3.8 「InnoDB および MyISAM インデックス統計コレクション」](#)

[セクション16.7.1 「MERGE テーブルの長所と短所」](#)

[NDB Cluster システム変数](#)

[セクション8.2.1.6 「インデックスコンディションプッシュダウンの最適化」](#)

[セクション8.9.3 「オプティマイザヒント」](#)

[セクション8.2.2.4 「マージまたは実体化を使用した導出テーブル、ビュー参照および共通テーブル式の最適化」](#)

[セクション8.8.3 「拡張 EXPLAIN 出力形式」](#)

ref_or_null

[セクション8.2.2.3 「EXISTS 戦略を使用したサブクエリーの最適化」](#)

[セクション8.8.2 「EXPLAIN 出力フォーマット」](#)

セクション8.2.1.15 「IS NULL の最適化」
セクション8.2.1.6 「インデックスコンディションプッシュダウンの最適化」

S

[\[index top\]](#)

system

セクション8.8.2 「EXPLAIN 出力フォーマット」
セクション8.2.1.2 「range の最適化」
セクション13.2.10 「SELECT ステートメント」
セクション8.8.3 「拡張 EXPLAIN 出力形式」

U

[\[index top\]](#)

unique_subquery

セクション8.2.2.3 「EXISTS 戦略を使用したサブクエリーの最適化」
セクション8.8.2 「EXPLAIN 出力フォーマット」

演算子の索引

[記号](#) | [A](#) | [B](#) | [C](#) | [D](#) | [E](#) | [I](#) | [L](#) | [M](#) | [N](#) | [O](#) | [R](#) | [V](#) | [X](#)

記号

[\[index top\]](#)

-

セクション12.11 「キャスト関数と演算子」
セクション24.6 「パーティショニングの制約と制限」
セクション9.5 「式」
セクション11.1.1 「数値データ型の構文」
セクション12.6.1 「算術演算子」

!

セクション1.3 「MySQL 8.0 の新機能」
セクション9.5 「式」
セクション12.4.1 「演算子の優先順位」
セクション12.4.3 「論理演算子」

!=

セクション11.5 「JSON データ型」
セクション1.3 「MySQL 8.0 の新機能」
セクション8.2.1.2 「range の最適化」
セクション12.4.2 「比較関数と演算子」
セクション12.4.1 「演算子の優先順位」

%

セクション12.6.1 「算術演算子」

&

セクション13.1.20 「CREATE TABLE ステートメント」
セクション24.6 「パーティショニングの制約と制限」

セクション12.13「ビット関数と演算子」

&&

セクション1.3「MySQL 8.0 の新機能」
セクション1.7.1「標準 SQL に対する MySQL 拡張機能」
セクション12.4.3「論理演算子」

>

セクション8.3.9「B ツリーインデックスとハッシュインデックスの比較」
セクション8.8.2「EXPLAIN 出力フォーマット」
セクション11.5「JSON データ型」
セクション1.3「MySQL 8.0 の新機能」
セクション8.2.1.2「range の最適化」
セクション1.7.1「標準 SQL に対する MySQL 拡張機能」
セクション12.4.2「比較関数と演算子」
セクション12.4.1「演算子の優先順位」
セクション8.3.11「生成されたカラムインデックスのオプティマイザによる使用」

->

セクション11.5「JSON データ型」
セクション12.18.3「JSON 値を検索する関数」
セクション1.3「MySQL 8.0 の新機能」
セクション13.1.20.9「セカンダリインデックスと生成されたカラム」

>>

セクション24.6「パーティショニングの制約と制限」
セクション12.13「ビット関数と演算子」
セクション1.7.1「標準 SQL に対する MySQL 拡張機能」

->>

セクション13.1.15「CREATE INDEX ステートメント」
セクション11.5「JSON データ型」
セクション1.3「MySQL 8.0 の新機能」
セクション13.1.20.9「セカンダリインデックスと生成されたカラム」

>=

セクション8.3.9「B ツリーインデックスとハッシュインデックスの比較」
セクション8.8.2「EXPLAIN 出力フォーマット」
セクション11.5「JSON データ型」
セクション1.3「MySQL 8.0 の新機能」
セクション8.2.1.2「range の最適化」
セクション1.7.1「標準 SQL に対する MySQL 拡張機能」
セクション12.4.2「比較関数と演算子」
セクション12.4.1「演算子の優先順位」
セクション8.3.11「生成されたカラムインデックスのオプティマイザによる使用」

<

セクション8.3.9「B ツリーインデックスとハッシュインデックスの比較」
セクション8.8.2「EXPLAIN 出力フォーマット」
セクション11.5「JSON データ型」
セクション1.3「MySQL 8.0 の新機能」
セクション3.3.4.6「NULL 値の操作」
セクション8.2.1.2「range の最適化」
セクション1.7.1「標準 SQL に対する MySQL 拡張機能」
セクション12.4.2「比較関数と演算子」
セクション12.4.1「演算子の優先順位」
セクション8.3.11「生成されたカラムインデックスのオプティマイザによる使用」

<>

セクション8.8.2「EXPLAIN 出力フォーマット」
セクション11.5「JSON データ型」
セクション1.3「MySQL 8.0 の新機能」
セクション3.3.4.6「NULL 値の操作」
セクション8.2.1.2「range の最適化」
セクション1.7.1「標準 SQL に対する MySQL 拡張機能」
セクション12.4.2「比較関数と演算子」
セクション12.4.1「演算子の優先順位」

<<

セクション24.6「パーティショニングの制約と制限」
セクション12.13「ビット関数と演算子」
セクション1.7.1「標準 SQL に対する MySQL 拡張機能」

<=

セクション8.3.9「B ツリーインデックスとハッシュインデックスの比較」
セクション8.8.2「EXPLAIN 出力フォーマット」
セクション11.5「JSON データ型」
セクション1.3「MySQL 8.0 の新機能」
セクション8.2.1.2「range の最適化」
セクション1.7.1「標準 SQL に対する MySQL 拡張機能」
セクション12.4.2「比較関数と演算子」
セクション12.4.1「演算子の優先順位」
セクション8.3.11「生成されたカラムインデックスのオプティマイザによる使用」

<=>

セクション8.8.2「EXPLAIN 出力フォーマット」
セクション11.5「JSON データ型」
セクション1.3「MySQL 8.0 の新機能」
セクション8.2.1.2「range の最適化」
セクション12.3「式評価での型変換」
セクション1.7.1「標準 SQL に対する MySQL 拡張機能」
セクション12.4.2「比較関数と演算子」
セクション12.4.1「演算子の優先順位」

*

セクション24.6「パーティショニングの制約と制限」
セクション11.1.1「数値データ型の構文」
セクション12.6.1「算術演算子」

+

セクション12.11「キャスト関数と演算子」
セクション24.6「パーティショニングの制約と制限」
セクション9.5「式」
セクション11.1.1「数値データ型の構文」
セクション12.6.1「算術演算子」

/

セクション5.1.8「サーバーシステム変数」
セクション24.6「パーティショニングの制約と制限」
セクション12.6.1「算術演算子」

:=

セクション9.4「ユーザー定義変数」
セクション12.4.4「割り当て演算子」

セクション13.7.6.1「変数代入の SET 構文」
セクション12.4.1「演算子の優先順位」

=

セクション8.3.9「B ツリーインデックスとハッシュインデックスの比較」
セクション8.8.2「EXPLAIN 出力フォーマット」
セクション11.5「JSON データ型」
セクション1.3「MySQL 8.0 の新機能」
セクション3.3.4.6「NULL 値の操作」
セクション8.2.1.2「range の最適化」
セクション13.2.11.12「サブクエリーの制約」
セクション9.4「ユーザー定義変数」
セクション12.4.4「割り当て演算子」
セクション13.7.6.1「変数代入の SET 構文」
セクション12.8.1「文字列比較関数および演算子」
セクション1.7.1「標準 SQL に対する MySQL 拡張機能」
セクション12.4.2「比較関数と演算子」
セクション12.4.1「演算子の優先順位」
セクション8.3.11「生成されたカラムインデックスのオプティマイザによる使用」

^

セクション24.6「パーティショニングの制約と制限」
セクション12.13「ビット関数と演算子」
セクション9.5「式」
セクション12.4.1「演算子の優先順位」

|

セクション24.6「パーティショニングの制約と制限」
セクション12.13「ビット関数と演算子」

||

セクション10.8.2「COLLATE 句の優先順位」
セクション1.3「MySQL 8.0 の新機能」
セクション5.1.11「サーバー SQL モード」
セクション9.5「式」
セクション1.7.1「標準 SQL に対する MySQL 拡張機能」
セクション12.4.1「演算子の優先順位」
セクション12.4.3「論理演算子」
セクション12.8.3「関数結果の文字セットと照合順序」

~

セクション24.6「パーティショニングの制約と制限」
セクション12.13「ビット関数と演算子」

A

[[index top](#)]

AND

セクション3.6.7「2 つのキーを使用した検索」
セクション8.3.9「B ツリーインデックスとハッシュインデックスの比較」
セクション13.1.20「CREATE TABLE ステートメント」
セクション8.2.2.3「EXISTS 戦略を使用したサブクエリーの最適化」
セクション1.3「MySQL 8.0 の新機能」
セクション8.2.1.2「range の最適化」
セクション8.2.1.3「インデックスマージの最適化」
セクション13.2.11.12「サブクエリーの制約」

[セクション20.3.4.2「テーブルの選択」](#)
[セクション20.4.4.2「テーブルの選択」](#)
[セクション25.5.2「ビュー処理アルゴリズム」](#)
[セクション12.8.1「文字列比較関数および演算子」](#)
[セクション1.7.1「標準 SQL に対する MySQL 拡張機能」](#)
[セクション3.3.4.2「特定の行の選択」](#)
[セクション8.2.1.22「行コンストラクタ式の最適化」](#)
[セクション12.4.3「論理演算子」](#)

B

[\[index top\]](#)

BETWEEN

[セクション8.3.9「B ツリーインデックスとハッシュインデックスの比較」](#)
[セクション8.8.2「EXPLAIN 出力フォーマット」](#)
[セクション11.5「JSON データ型」](#)
[セクション13.5.1「PREPARE ステートメント」](#)
[セクション8.2.1.2「range の最適化」](#)
[セクション12.3「式評価での型変換」](#)
[セクション8.2.1.13「条件フィルタ」](#)
[セクション12.4.2「比較関数と演算子」](#)
[セクション8.3.11「生成されたカラムインデックスのオプティマイザによる使用」](#)

BINARY

[セクション12.11「キャスト関数と演算子」](#)
[セクション3.3.4.7「パターンマッチング」](#)
[セクション12.13「ビット関数と演算子」](#)
[セクション8.4.2.2「文字および文字列型の最適化」](#)
[セクション3.3.4.4「行のソート」](#)

C

[\[index top\]](#)

CASE

[セクション13.6.5.1「CASE ステートメント」](#)
[セクション13.5.1「PREPARE ステートメント」](#)
[セクション12.5「フロー制御関数」](#)
[セクション9.5「式」](#)
[セクション1.7.1「標準 SQL に対する MySQL 拡張機能」](#)

CASE value WHEN compare_value THEN result END

[セクション12.5「フロー制御関数」](#)

CASE WHEN condition THEN result END

[セクション12.5「フロー制御関数」](#)

CASE WHEN expr1 = expr2 THEN NULL ELSE expr1 END

[セクション12.5「フロー制御関数」](#)

column->>path

[セクション12.18.3「JSON 値を検索する関数」](#)

column->path

[セクション11.5「JSON データ型」](#)
[セクション12.18.3「JSON 値を検索する関数」](#)

D

[\[index top\]](#)

DIV

[セクション24.6「パーティショニングの制約と制限」](#)
[セクション12.6.1「算術演算子」](#)

E

[\[index top\]](#)

expr BETWEEN min AND max

[セクション12.4.2「比較関数と演算子」](#)

expr IN ()

[セクション12.4.2「比較関数と演算子」](#)

expr LIKE pat

[セクション12.8.1「文字列比較関数および演算子」](#)

expr NOT BETWEEN min AND max

[セクション12.4.2「比較関数と演算子」](#)

expr NOT IN ()

[セクション12.4.2「比較関数と演算子」](#)

expr NOT LIKE pat

[セクション12.8.1「文字列比較関数および演算子」](#)

expr NOT REGEXP pat

[セクション12.8.2「正規表現」](#)

expr NOT RLIKE pat

[セクション12.8.2「正規表現」](#)

expr REGEXP pat

[セクション12.8.2「正規表現」](#)

expr RLIKE pat

[セクション12.8.2「正規表現」](#)

expr1 SOUNDS LIKE expr2

[セクション12.8「文字列関数および演算子」](#)

I

[\[index top\]](#)

IN()

[セクション8.8.2「EXPLAIN 出力フォーマット」](#)
[セクション11.5「JSON データ型」](#)
[セクション8.2.1.2「range の最適化」](#)
[セクション12.3「式評価での型変換」](#)

セクション12.4.1 「演算子の優先順位」
セクション8.3.11 「生成されたカラムインデックスのオプティマイザによる使用」
セクション8.2.1.22 「行コンストラクタ式の最適化」

IS

セクション12.4.1 「演算子の優先順位」

IS boolean_value

セクション12.4.2 「比較関数と演算子」

IS NOT boolean_value

セクション12.4.2 「比較関数と演算子」

IS NOT NULL

セクションB.3.4.3 「NULL 値に関する問題」
セクション3.3.4.6 「NULL 値の操作」
セクション8.2.1.2 「range の最適化」
セクション12.4.2 「比較関数と演算子」

IS NULL

セクション8.2.2.3 「EXISTS 戦略を使用したサブクエリーの最適化」
セクション8.8.2 「EXPLAIN 出力フォーマット」
セクション8.2.1.15 「IS NULL の最適化」
セクションB.3.4.3 「NULL 値に関する問題」
セクション3.3.4.6 「NULL 値の操作」
セクション8.2.1.2 「range の最適化」
セクション5.1.8 「サーバーシステム変数」
セクション12.4.2 「比較関数と演算子」

L

[[index top](#)]

LIKE

セクション8.3.9 「B ツリーインデックスとハッシュインデックスの比較」
セクション8.8.2 「EXPLAIN 出力フォーマット」
セクション13.8.3 「HELP ステートメント」
セクション12.18.3 「JSON 値を検索する関数」
セクションA.11 「MySQL 8.0 FAQ: MySQL の中国語、日本語、および韓国語の文字セット」
セクション10.2 「MySQL での文字セットと照合順序」
セクション4.5.1.4 「mysql クライアントのサーバー側ヘルプ」
セクション23.6.11 「NDB Cluster レプリケーションの競合解決」
セクション28.4.4.5 「ps_setup_disable_consumer() プロシージャ」
セクション28.4.4.6 「ps_setup_disable_instrument() プロシージャ」
セクション28.4.4.9 「ps_setup_enable_consumer() プロシージャ」
セクション28.4.4.10 「ps_setup_enable_instrument() プロシージャ」
セクション8.2.1.2 「range の最適化」
セクション17.1.6.3 「Replica Server のオプションと変数」
セクション11.3.6 「SET 型」
セクション13.7.7.3 「SHOW CHARACTER SET ステートメント」
セクション13.7.7.4 「SHOW COLLATION ステートメント」
セクション13.7.7.5 「SHOW COLUMNS ステートメント」
セクション13.7.7.14 「SHOW DATABASES ステートメント」
セクション13.7.7.18 「SHOW EVENTS ステートメント」
セクション13.7.7.24 「SHOW OPEN TABLES ステートメント」
セクション13.7.7.28 「SHOW PROCEDURE STATUS ステートメント」
セクション13.7.7.37 「SHOW STATUS ステートメント」

[セクション13.7.7.38 「SHOW TABLE STATUS ステートメント」](#)
[セクション13.7.7.39 「SHOW TABLES ステートメント」](#)
[セクション13.7.7.40 「SHOW TRIGGERS ステートメント」](#)
[セクション13.7.7.41 「SHOW VARIABLES ステートメント」](#)
[セクション26.55 「SHOW ステートメントの拡張」](#)
[セクション6.2.4 「アカウント名の指定」](#)
[セクション6.2.7 「アクセス制御、ステージ 2: リクエストの確認」](#)
[セクション27.4.4 「インストゥルメントによる事前フィルタリング」](#)
[セクション12.11 「キャスト関数と演算子」](#)
[セクション23.5.16 「クイックリファレンス: NDB Cluster SQL ステートメント」](#)
[セクション5.1.9 「システム変数の使用」](#)
[セクション10.8.5 「バイナリ照合順序と_bin 照合順序」](#)
[セクション3.3.4.7 「パターンマッチング」](#)
[セクション9.1.1 「文字列リテラル」](#)
[セクション12.8.1 「文字列比較関数および演算子」](#)
[セクション5.1.9.5 「構造化システム変数」](#)
[セクション1.7.1 「標準 SQL に対する MySQL 拡張機能」](#)
[セクション12.4.1 「演算子の優先順位」](#)

LIKE '_A%'

[セクションA.11 「MySQL 8.0 FAQ: MySQL の中国語、日本語、および韓国語の文字セット」](#)

LIKE 'pattern'

[セクション8.2.1.2 「range の最適化」](#)
[セクション13.7.7 「SHOW ステートメント」](#)

LIKE ... ESCAPE

[セクションB.3.7 「MySQL の既知の問題」](#)

M

[\[index top\]](#)

MEMBER OF()

[セクション13.1.15 「CREATE INDEX ステートメント」](#)
[セクション12.18.3 「JSON 値を検索する関数」](#)
[セクション1.3 「MySQL 8.0 の新機能」](#)

N

[\[index top\]](#)

N % M

[セクション12.6.2 「数学関数」](#)
[セクション12.6.1 「算術演算子」](#)

N MOD M

[セクション12.6.2 「数学関数」](#)
[セクション12.6.1 「算術演算子」](#)

NOT

[セクション1.3 「MySQL 8.0 の新機能」](#)
[セクション5.1.11 「サーバー SQL モード」](#)
[セクション12.4.3 「論理演算子」](#)

NOT IN()

[セクション8.2.1.2 「range の最適化」](#)

NOT LIKE

[セクション3.3.4.7「パターンマッチング」](#)
[セクション12.8.1「文字列比較関数および演算子」](#)

NOT REGEXP

[セクション12.8.1「文字列比較関数および演算子」](#)
[セクション1.7.1「標準 SQL に対する MySQL 拡張機能」](#)

NOT RLIKE

[セクション12.8.1「文字列比較関数および演算子」](#)

O

[\[index top\]](#)

OR

[セクション3.6.7「2つのキーを使用した検索」](#)
[セクション8.2.2.3「EXISTS 戦略を使用したサブクエリーの最適化」](#)
[セクション13.7.1.6「GRANT ステートメント」](#)
[セクション1.3「MySQL 8.0 の新機能」](#)
[セクション8.2.1.2「range の最適化」](#)
[セクション8.2.1.3「インデックスマージの最適化」](#)
[セクション5.1.11「サーバー SQL モード」](#)
[セクション20.3.4.2「テーブルの選択」](#)
[セクション20.4.4.2「テーブルの選択」](#)
[セクション9.5「式」](#)
[セクション12.8.1「文字列比較関数および演算子」](#)
[セクション1.7.1「標準 SQL に対する MySQL 拡張機能」](#)
[セクション12.4.1「演算子の優先順位」](#)
[セクション3.3.4.2「特定の行の選択」](#)
[セクション8.2.1.22「行コンストラクタ式の最適化」](#)
[セクション12.4.3「論理演算子」](#)

R

[\[index top\]](#)

REGEXP

[セクション1.3「MySQL 8.0 の新機能」](#)
[セクション3.3.4.7「パターンマッチング」](#)
[セクション10.11「文字セットの制約」](#)
[セクション1.7.1「標準 SQL に対する MySQL 拡張機能」](#)
[セクション12.8.2「正規表現」](#)
[セクション12.4.1「演算子の優先順位」](#)

RLIKE

[セクション1.3「MySQL 8.0 の新機能」](#)
[セクション3.3.4.7「パターンマッチング」](#)
[セクション10.11「文字セットの制約」](#)
[セクション12.8.2「正規表現」](#)

V

[\[index top\]](#)

value MEMBER OF()

[セクション12.18.3「JSON 値を検索する関数」](#)

X

[\[index top\]](#)

XOR

[セクション12.4.3 「論理演算子」](#)

[セクション12.20.1 「集計関数の説明」](#)

オプションの索引

[記号](#) | [A](#) | [B](#) | [C](#) | [D](#) | [E](#) | [F](#) | [G](#) | [H](#) | [I](#) | [J](#) | [K](#) | [L](#) | [M](#) | [N](#) | [O](#) | [P](#) | [Q](#) | [R](#) | [S](#) | [T](#) | [U](#) | [V](#) | [W](#) | [X](#) | [Y](#) | [Z](#)

記号

[\[index top\]](#)

--

[セクション1.7.2.4 「コメントの先頭としての「--」](#)

-#

[セクション4.4.1 「comp_err — MySQL エラーメッセージファイルのコンパイル」](#)

[セクション5.9.4 「DEBUG パッケージ」](#)

[セクション4.6.1 「ibd2sdi — InnoDB テーブルスペース SDI 抽出ユーティリティ」](#)

[セクション4.7.2 「my_print_defaults — オプションファイルからのオプションの表示」](#)

[セクション4.6.4.1 「myisamchk の一般オプション」](#)

[セクション4.6.6 「myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成」](#)

[セクション4.5.1.1 「mysql クライアントオプション」](#)

[セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティ」](#)

[セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」](#)

[セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」](#)

[セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」](#)

[セクション4.5.3 「mysqlcheck — テーブル保守プログラム」](#)

[セクション4.5.4 「mysqldump — データベースバックアッププログラム」](#)

[セクション4.5.5 「mysqlimport — データインポートプログラム」](#)

[セクション4.5.6 「mysqlpump — データベースバックアッププログラム」](#)

[セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」](#)

[セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」](#)

[セクション5.1.7 「サーバーコマンドオプション」](#)

-1

[セクション4.5.3 「mysqlcheck — テーブル保守プログラム」](#)

1231

[セクション4.8.2 「perror — MySQL エラーメッセージ情報の表示」](#)

-?

[セクション4.4.1 「comp_err — MySQL エラーメッセージファイルのコンパイル」](#)

[セクション4.6.2 「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティ」](#)

[セクション4.7.2 「my_print_defaults — オプションファイルからのオプションの表示」](#)

[セクション4.6.3 「myisam_ftdump — 全文インデックス情報の表示」](#)

[セクション4.6.4.1 「myisamchk の一般オプション」](#)

[セクション4.6.5 「myisamlog — MyISAM ログファイルの内容の表示」](#)

[セクション4.6.6 「myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成」](#)

[セクション1.2.2 「MySQL の主な機能」](#)

[セクション4.5.1.1 「mysql クライアントオプション」](#)

[セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティ」](#)

セクション4.4.2 「mysql_secure_installation — MySQL インストールのセキュリティ改善」
セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.5 「mysqlexport — データインポートプログラム」
セクション4.5.6 「mysqldump — データベースバックアッププログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」
セクション23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」
セクション23.4.7 「ndb_config — NDB Cluster 構成情報の抽出」
セクション23.4.16 「ndb_perror — NDB エラーメッセージ情報の取得」
セクション23.4.18 「ndb_print_file — NDB デスクデータファイル内容の出力」
セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」
セクション23.4.31 「ndbxfrm — NDB Cluster によって作成されたファイルの圧縮、圧縮解除、暗号化、および復号化」
セクション4.8.2 「perror — MySQL エラーメッセージ情報の表示」
セクション4.2.2.1 「コマンド行でのオプションの使用」
セクション5.1.7 「サーバーコマンドオプション」

?

セクション4.4.3 「mysql_ssl_rsa_setup — SSL/RSA ファイルの作成」

このスタイルのテキスト

セクション1.1 「このマニュアルについて」

A

[index top]

-A

セクション4.5.1.1 「mysql クライアントオプション」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.6 「mysqldump — データベースバックアッププログラム」
セクション23.4.15 「ndb_move_data — NDB データコピーユーティリティ」
セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」
セクション4.6.4.4 「その他の myisamchk オプション」

-a

セクション6.4.4.6 「HashiCorp Vault キーリングプラグインの使用」
セクション4.6.2 「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティ」
セクション7.6.4 「MyISAM テーブルの最適化」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.6.9 「mysqldumpslow — スロークエリログファイルの要約」
セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」
セクション23.1.4 「NDB Cluster の新機能」
セクション23.5.1 「NDB Cluster 管理クライアントのコマンド」
セクション23.2.8.1 「NDB Cluster 自動インストーラの要件」
セクション23.4.7 「ndb_config — NDB Cluster 構成情報の抽出」
セクション23.4.9 「ndb_desc — NDB テーブルの表示」
セクション23.4.26 「ndb_setup.py — NDB Cluster 用ブラウザベースの Auto-Installer の起動 (非推奨)」
セクション4.6.4.4 「その他の myisamchk オプション」

--abort-on-error

セクション23.4.13 「ndb_import — NDB への CSV データのインポート」
セクション23.4.15 「ndb_move_data — NDB データコピーユーティリティ」

--abort-slave-event-count

セクション17.1.6.3「Replica Server のオプションと変数」

--add-drop-database

セクション4.5.4「mysqldump — データベースバックアッププログラム」

セクション7.4.1「mysqldump による SQL フォーマットでのデータのダンプ」

セクション4.5.6「mysqlpump — データベースバックアッププログラム」

--add-drop-table

セクション4.5.4「mysqldump — データベースバックアッププログラム」

セクション4.5.6「mysqlpump — データベースバックアッププログラム」

--add-drop-trigger

セクション4.5.4「mysqldump — データベースバックアッププログラム」

--add-drop-user

セクション4.5.6「mysqlpump — データベースバックアッププログラム」

--add-locks

セクション4.5.4「mysqldump — データベースバックアッププログラム」

セクション4.5.6「mysqlpump — データベースバックアッププログラム」

--add-missing

セクション23.1.4「NDB Cluster の新機能」

セクション23.4.6「ndb_blob_tool — NDB Cluster テーブルの BLOB および TEXT カラムのチェックおよび修復」

--admin-ssl

セクション5.1.7「サーバーコマンドオプション」

セクション6.3.1「暗号化接続を使用するための MySQL の構成」

セクション5.1.12.2「管理接続管理」

admin-ssl

セクション5.1.12.2「管理接続管理」

--ai-increment

セクション23.4.13「ndb_import — NDB への CSV データのインポート」

--ai-offset

セクション23.4.13「ndb_import — NDB への CSV データのインポート」

--ai-prefetch-sz

セクション23.4.13「ndb_import — NDB への CSV データのインポート」

--all

セクション4.6.7「mysql_config_editor — MySQL 構成ユーティリティー」

--all-databases

セクション4.5.3「mysqlcheck — テーブル保守プログラム」

セクション4.5.4「mysqldump — データベースバックアッププログラム」

セクション7.4.1「mysqldump による SQL フォーマットでのデータのダンプ」

mysqldump を使用したデータスナップショットの作成

セクション4.5.6「mysqlpump — データベースバックアッププログラム」

セクション23.5.14「ndbinfo: NDB Cluster 情報データベース」

セクション7.4.2「SQL フォーマットバックアップのリロード」

セクション2.11.6「Unix/Linux での MySQL バイナリまたはパッケージベースのインストールのアップグレード」

セクション2.11.13「テーブルまたはインデックスの再作成または修復」
セクション14.7「データディクショナリの使用法の違い」
セクション4.6.8.3「バイナリログファイルのバックアップのための mysqlbinlog の使用」

--all-in-1

セクション4.5.3「mysqlcheck — テーブル保守プログラム」

--all-tablespaces

セクション4.5.4「mysqldump — データベースバックアッププログラム」

--allow-keywords

セクション4.5.4「mysqldump — データベースバックアッププログラム」

--allow-mismatches

セクション4.6.2「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティ」

--allow-pk-changes

セクション23.1.4「NDB Cluster の新機能」
セクション23.4.23「ndb_restore — NDB Cluster バックアップの復元」

--allow-suspicious-udfs

セクション5.1.7「サーバーコマンドオプション」

--analyze

セクション7.6.4「MyISAM テーブルの最適化」
セクション4.6.4.1「myisamchk の一般オプション」
セクション4.5.3「mysqlcheck — テーブル保守プログラム」
セクション4.6.4.4「その他の myisamchk オプション」

--ansi

セクション1.7「MySQL の標準への準拠」
セクション5.1.7「サーバーコマンドオプション」

antonio

セクション6.4.1.5「PAM プラガブル認証」

--append

セクション23.4.23「ndb_restore — NDB Cluster バックアップの復元」

--apply-slave-statements

セクション4.5.4「mysqldump — データベースバックアッププログラム」

--audit-log

セクション6.4.5.2「MySQL Enterprise Audit のインストールまたはアンインストール」
セクション6.4.5.10「監査ログ参照」

--auto-generate-sql

セクション4.5.8「mysqslap — ロードエミュレーションクライアント」

--auto-generate-sql-add-autoincrement

セクション4.5.8「mysqslap — ロードエミュレーションクライアント」

--auto-generate-sql-execute-number

セクション4.5.8「mysqslap — ロードエミュレーションクライアント」

--auto-generate-sql-guid-primary

セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」

--auto-generate-sql-load-type

セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」

--auto-generate-sql-secondary-indexes

セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」

--auto-generate-sql-unique-query-number

セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」

--auto-generate-sql-unique-write-number

セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」

--auto-generate-sql-write-number

セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」

--auto-inc

セクション23.1.4 「NDB Cluster の新機能」

セクション23.4.9 「ndb_desc — NDB テーブルの表示」

--auto-rehash

セクション4.5.1.1 「mysql クライアントオプション」

セクション4.5.1.2 「mysql クライアントコマンド」

セクション15.8.10.2 「非永続的オブティマイザ統計のパラメータの構成」

auto-rehash

セクション15.8.10.2 「非永続的オブティマイザ統計のパラメータの構成」

--auto-repair

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

--auto-vertical-output

セクション4.5.1.1 「mysql クライアントオプション」

--autocommit

セクション5.1.8 「サーバーシステム変数」

B

[\[index top\]](#)

-B

セクション4.6.4.3 「myisamchk の修復オプション」

セクション4.5.1.1 「mysql クライアントオプション」

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

-b

セクション4.6.6 「myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成」

セクション4.5.1.1 「mysql クライアントオプション」

セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」

セクション23.4.9 「ndb_desc — NDB テーブルの表示」

セクション23.4.23 「`ndb_restore` — NDB Cluster バックアップの復元」
セクション4.6.4.4 「その他の `myisamchk` オプション」
セクション5.1.7 「サーバーコマンドオプション」

--back_log

セクション2.7 「Solaris への MySQL のインストール」

--backup

セクション4.6.4.3 「`myisamchk` の修復オプション」
セクション4.6.6 「`mysampack` — 圧縮された読み取り専用の MyISAM テーブルの生成」

--backup-password

セクション23.1.4 「NDB Cluster の新機能」
セクション23.5.8.2 「NDB Cluster 管理クライアントを使用したバックアップの作成」
セクション23.4.23 「`ndb_restore` — NDB Cluster バックアップの復元」

--backup-path

セクション23.4.23 「`ndb_restore` — NDB Cluster バックアップの復元」
パラレルバックアップのシリアルリストア
パラレルバックアップのリストア
元のノードより少ないノードへのリストア

backup-path

セクション23.4.23 「`ndb_restore` — NDB Cluster バックアップの復元」

backup-to-image

セクション2.5.6.2 「Docker での MySQL Server のデプロイに関するその他のトピック」

--backupid

セクション23.1.4 「NDB Cluster の新機能」
セクション23.4.23 「`ndb_restore` — NDB Cluster バックアップの復元」
元のノードより少ないノードへのリストア

--base64-output

セクション4.6.8 「`mysqlbinlog` — バイナリログファイルを処理するためのユーティリティー」
セクション4.6.8.2 「`mysqlbinlog` 行イベントの表示」
セクション17.2.1.1 「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」
セクション17.2.1.2 「行ベースロギングおよびレプリケーションの使用」

--basedir

セクション5.8 「1 つのマシン上での複数の MySQL インスタンスの実行」
セクション2.10.2.1 「MySQL Server の起動時の問題のトラブルシューティング」
セクション2.9.7 「MySQL ソース構成オプション」
セクション4.3.2 「`mysqld_safe` — MySQL サーバー起動スクリプト」
セクション23.1.7.1 「NDB Cluster の SQL 構文に準拠していません」
セクション5.1.7 「サーバーコマンドオプション」
セクション2.10.1 「データディレクトリの初期化」

basedir

セクション2.3.5 「Microsoft Windows MySQL Server インストールのトラブルシューティング」
セクション4.3.3 「`mysql.server` — MySQL サーバー起動スクリプト」
セクション2.3.4.2 「オプションファイルの作成」

--batch

セクション4.5.1.1 「`mysql` クライアントオプション」
セクション4.5.1.3 「`mysql` クライアントロギング」

--binary-as-hex

セクション4.5.1.1「mysql クライアントオプション」

--binary-mode

セクション4.5.1.6「mysql クライアントのヒント」

セクション4.5.1.1「mysql クライアントオプション」

セクション4.5.1.2「mysql クライアントコマンド」

セクション4.6.8「mysqlbinlog — バイナリログファイル进行处理するためのユーティリティ」

セクション9.7「コメント」

セクション7.5.1「バイナリログを使用したポイントインタイムリカバリ」

--bind-address

セクション4.5.1.1「mysql クライアントオプション」

セクション4.4.5「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

セクション4.5.2「mysqladmin — A MySQL Server 管理プログラム」

セクション4.6.8「mysqlbinlog — バイナリログファイル进行处理するためのユーティリティ」

セクション4.5.3「mysqlcheck — テーブル保守プログラム」

セクション4.5.4「mysqldump — データベースバックアッププログラム」

セクション4.5.5「mysqlexport — データインポートプログラム」

セクション4.5.6「mysqmpump — データベースバックアッププログラム」

セクション4.5.7「mysqlshow — データベース、テーブル、およびカラム情報の表示」

セクション23.1.4「NDB Cluster の新機能」

セクション23.4.4「ndb_mgmd — NDB Cluster 管理サーバーデーモン」

セクション23.4.1「ndbd — NDB Cluster データノードデーモン」

--binlog-checksum

セクション17.1.6.4「バイナリロギングのオプションと変数」

--binlog-do-db

セクション4.6.8「mysqlbinlog — バイナリログファイル进行处理するためのユーティリティ」

セクション23.6.3「NDB Cluster レプリケーションの既知の問題」

セクション17.1.6.3「Replica Server のオプションと変数」

セクション17.2.5「サーバーがレプリケーションフィルタリングルールをどのように評価するか」

セクション17.2.5.1「データベースレベルレプリケーションオプションおよびバイナリロギングオプションの評価」

セクション17.1.6.4「バイナリロギングのオプションと変数」

セクション5.4.4「バイナリログ」

--binlog-format

セクションA.4「MySQL 8.0 FAQ: ストアドプロシージャおよびストアドファンクション」

セクション23.1.7.6「NDB Cluster でサポートされない機能または欠落している機能」

セクション23.6.2「NDB Cluster レプリケーションの一般的な要件」

セクション23.6.6「NDB Cluster レプリケーションの開始 (シングルレプリケーションチャンネル)」

セクション5.4.4.1「バイナリロギング形式」

セクション5.4.4.2「バイナリログ形式の設定」

--binlog-ignore-db

セクション23.6.3「NDB Cluster レプリケーションの既知の問題」

セクション17.1.6.3「Replica Server のオプションと変数」

セクション17.2.5「サーバーがレプリケーションフィルタリングルールをどのように評価するか」

セクション17.2.5.1「データベースレベルレプリケーションオプションおよびバイナリロギングオプションの評価」

セクション17.1.6.4「バイナリロギングのオプションと変数」

セクション5.4.4「バイナリログ」

セクション17.3.3.1「レプリケーション PRIVILEGE_CHECKS_USER アカウントの権限」

--binlog-row-event-max-size

セクション4.6.8「mysqlbinlog — バイナリログファイル进行处理するためのユーティリティ」

セクション17.1.6.4「バイナリロギングのオプションと変数」

セクション5.4.4.2 「バイナリログ形式の設定」

--blob-info

セクション23.4.9 「ndb_desc — NDB テーブルの表示」

--block-search

セクション4.6.4.4 「その他の myisamchk オプション」

--bootstrap

セクション1.3 「MySQL 8.0 の新機能」

--browser-start-page

セクション23.4.26 「ndb_setup.py — NDB Cluster 用ブラウザベースの Auto-Installer の起動 (非推奨)」

C

[index top]

-C

セクション4.4.1 「comp_err — MySQL エラーメッセージファイルのコンパイル」
セクション4.6.2 「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティ」
セクション4.6.4.2 「myisamchk のチェックオプション」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」
セクション4.2.3 「サーバーに接続するためのコマンドオプション」

-C

セクション4.4.1 「comp_err — MySQL エラーメッセージファイルのコンパイル」
セクション6.4.4.6 「HashiCorp Vault キーリングプラグインの使用」
セクション4.6.1 「ibd2sdi — InnoDB テーブルスペース SDI 抽出ユーティリティ」
セクション4.6.2 「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティ」
セクション4.7.2 「my_print_defaults — オプションファイルからのオプションの表示」
セクション4.6.3 「myisam_ftdump — 全文インデックス情報の表示」
セクション4.6.4.2 「myisamchk のチェックオプション」
セクション4.6.5 「myisamlog — MyISAM ログファイルの内容の表示」
セクションA.10 「MySQL 8.0 FAQ: NDB Cluster」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」
セクション23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」
セクション23.2.8.1 「NDB Cluster 自動インストーラの要件」
セクション23.4.7 「ndb_config — NDB Cluster 構成情報の抽出」
セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」
セクション23.4.26 「ndb_setup.py — NDB Cluster 用ブラウザベースの Auto-Installer の起動 (非推奨)」
セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」
セクション23.4.31 「ndbxfm — NDB Cluster によって作成されたファイルの圧縮、圧縮解除、暗号化、および復号化」

セクション23.2.2.1「バイナリリリースから Windows への NDB Cluster のインストール」
元のノードより少ないノードへのリストア

--ca-certs-file

セクション23.2.8.1「NDB Cluster 自動インストーラの要件」
セクション23.4.26「ndb_setup.py — NDB Cluster 用ブラウザベースの Auto-Installer の起動 (非推奨)」

--cert-file

セクション23.2.8.1「NDB Cluster 自動インストーラの要件」
セクション23.4.26「ndb_setup.py — NDB Cluster 用ブラウザベースの Auto-Installer の起動 (非推奨)」

--cflags

セクション2.9.8「MySQL のコンパイルに関する問題」
セクション4.7.1「mysql_config — クライアントのコンパイル用オプションの表示」

--character-set-client-handshake

セクション10.10.7.1「cp932 文字セット」
セクションA.11「MySQL 8.0 FAQ: MySQL の中国語、日本語、および韓国語の文字セット」
セクション5.1.7「サーバーコマンドオプション」

--character-set-server

セクションA.11「MySQL 8.0 FAQ: MySQL の中国語、日本語、および韓国語の文字セット」
セクション10.5「アプリケーションの文字セットおよび照合順序の構成」
セクション10.3.2「サーバー文字セットおよび照合順序」
セクション10.15「文字セットの構成」

--character-sets-dir

セクション4.6.4.3「myisamchk の修復オプション」
セクション4.6.6「myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成」
セクション4.5.1.1「mysql クライアントオプション」
セクション4.4.5「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
セクション4.5.2「mysqladmin — A MySQL Server 管理プログラム」
セクション4.6.8「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」
セクション4.5.3「mysqlcheck — テーブル保守プログラム」
セクション4.5.4「mysqldump — データベースバックアッププログラム」
セクション4.5.5「mysqlimport — データインポートプログラム」
セクション4.5.6「mysqlpump — データベースバックアッププログラム」
セクション4.5.7「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション23.4.32「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」
セクション23.4.15「ndb_move_data — NDB データコピーユーティリティ」
セクション10.15「文字セットの構成」
セクションB.3.2.15「文字セットを初期化できません」

--character_set_server

セクション2.9.7「MySQL ソース構成オプション」

--charset

セクション4.4.1「comp_err — MySQL エラーメッセージファイルのコンパイル」

--check

セクション4.6.4.2「myisamchk のチェックオプション」
セクション4.5.3「mysqlcheck — テーブル保守プログラム」

--check-missing

セクション23.1.4「NDB Cluster の新機能」
セクション23.4.6「ndb_blob_tool — NDB Cluster テーブルの BLOB および TEXT カラムのチェックおよび修復」

--check-only-changed

セクション4.6.4.2 「mysamchk のチェックオプション」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

--check-orphans

セクション23.4.6 「ndb_blob_tool — NDB Cluster テーブルの BLOB および TEXT カラムのチェックおよび修復」

--check-upgrade

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

--chroot

セクション5.1.7 「サーバーコマンドオプション」

--clone

セクション5.6.7.1 「クローンプラグインのインストール」

--cluster-config-suffix

セクション23.4.7 「ndb_config — NDB Cluster 構成情報の抽出」
セクション23.4.4 「ndb_mgmd — NDB Cluster 管理サーバーデーモン」

cluster-config-suffix

セクション23.4.7 「ndb_config — NDB Cluster 構成情報の抽出」
セクション23.4.4 「ndb_mgmd — NDB Cluster 管理サーバーデーモン」

CMAKE_BUILD_TYPE

セクション2.9.7 「MySQL ソース構成オプション」

CMAKE_C_FLAGS

セクション2.9.8 「MySQL のコンパイルに関する問題」
セクション2.9.7 「MySQL ソース構成オプション」
セクション5.9.1.1 「デバッグのための MySQL のコンパイル」

CMAKE_C_FLAGS_build_type

セクション2.9.7 「MySQL ソース構成オプション」

CMAKE_C_FLAGS_RELWITHDEBINFO

セクション2.9.7 「MySQL ソース構成オプション」

CMAKE_CXX_FLAGS

セクション2.9.8 「MySQL のコンパイルに関する問題」
セクション2.9.7 「MySQL ソース構成オプション」
セクション5.9.1.1 「デバッグのための MySQL のコンパイル」

CMAKE_CXX_FLAGS_build_type

セクション2.9.7 「MySQL ソース構成オプション」

CMAKE_CXX_FLAGS_RELWITHDEBINFO

セクション2.9.7 「MySQL ソース構成オプション」

CMAKE_INSTALL_PREFIX

セクション2.9.7 「MySQL ソース構成オプション」
セクション5.8.3 「Unix 上での複数の MySQL インスタンスの実行」
セクション6.4.4.13 「キーリングシステム変数」

セクション5.1.8 「サーバーシステム変数」
セクション2.9.5 「開発ソースツリーを使用して MySQL をインストールする」

CMAKE_PREFIX_PATH

セクション2.9.7 「MySQL ソース構成オプション」

--collation-server

セクション10.5 「アプリケーションの文字セットおよび照合順序の構成」
セクション10.3.2 「サーバー文字セットおよび照合順序」
セクション10.15 「文字セットの構成」

--collation_server

セクション2.9.7 「MySQL ソース構成オプション」

--color

セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」

--column-names

セクション4.5.1.1 「mysql クライアントオプション」
セクション4.2.2.4 「プログラムオプション修飾子」

--column-statistics

セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

--column-type-info

セクション8.2.1.19 「LIMIT クエリーの最適化」
セクション4.5.1.1 「mysql クライアントオプション」

--columns

セクション4.5.5 「mysqlimport — データインポートプログラム」

--comments

セクション4.5.1.1 「mysql クライアントオプション」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」

--commit

セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」

--compact

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

--compatible

セクション2.11.4 「MySQL 8.0 での変更」
セクション1.3 「MySQL 8.0 の新機能」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」

COMPILATION_COMMENT

セクション2.9.7 「MySQL ソース構成オプション」
セクション5.1.8 「サーバーシステム変数」

COMPILATION_COMMENT_SERVER

セクション2.9.7 「MySQL ソース構成オプション」
セクション5.1.8 「サーバーシステム変数」

--complete-insert

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

--compress

セクション1.3 「MySQL 8.0 の新機能」

セクション4.5.1.1 「mysql クライアントオプション」

セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」

セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.5 「mysqlimport — データインポートプログラム」

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」

セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」

セクション23.4.31 「ndb_xfrm — NDB Cluster によって作成されたファイルの圧縮、圧縮解除、暗号化、および復号化」

セクション4.2.3 「サーバーに接続するためのコマンドオプション」

セクション4.2.8 「接続圧縮制御」

--compress-output

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

--compression-algorithms

セクション4.5.1.1 「mysql クライアントオプション」

セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」

セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.5 「mysqlimport — データインポートプログラム」

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」

セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」

セクション4.2.3 「サーバーに接続するためのコマンドオプション」

セクション4.2.8 「接続圧縮制御」

--concurrency

セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」

--config-cache

セクション23.3.3 「NDB Cluster 構成ファイル」

セクション23.4.4 「ndb_mgmd — NDB Cluster 管理サーバーデーモン」

--config-dir

セクション23.4.4 「ndb_mgmd — NDB Cluster 管理サーバーデーモン」

--config-file

セクション4.7.2 「my_print_defaults — オプションファイルからのオプションの表示」

セクションA.10 「MySQL 8.0 FAQ: NDB Cluster」

セクション23.2.4 「NDB Cluster の初期起動」

セクション23.2.2.4 「NDB Cluster プロセスを Windows サービスとしてインストール」

セクション23.3.3.1 「NDB Cluster 構成: 基本例」

セクション23.4.7 「ndb_config — NDB Cluster 構成情報の抽出」

セクション23.4.4 「ndb_mgmd — NDB Cluster 管理サーバーデーモン」

セクション23.2.2.3 「Windows での NDB Cluster の初期起動」

--config-from-node

セクション23.4.7 「[ndb_config](#) — NDB Cluster 構成情報の抽出」

--config_from_node

セクション23.4.7 「[ndb_config](#) — NDB Cluster 構成情報の抽出」

--configdir

セクション23.3.3 「[NDB Cluster 構成ファイル](#)」

セクション23.4.4 「[ndb_mgmd](#) — NDB Cluster 管理サーバーデーモン」

--configinfo

セクション23.4.7 「[ndb_config](#) — NDB Cluster 構成情報の抽出」

--connect

セクション23.4.23 「[ndb_restore](#) — NDB Cluster バックアップの復元」

--connect-delay

セクション23.4.1 「[ndbd](#) — NDB Cluster データノードデーモン」

--connect-expired-password

セクション4.5.1.1 「[mysql クライアントオプション](#)」

セクション6.2.16 「[期限切れパスワードのサーバー処理](#)」

--connect-retries

セクション23.4.32 「[NDB Cluster プログラムに共通のオプション](#) — NDB Cluster プログラムに共通のオプション」

セクション23.4.5 「[ndb_mgm](#) — NDB Cluster 管理クライアント」

セクション23.4.1 「[ndbd](#) — NDB Cluster データノードデーモン」

--connect-retry-delay

セクション23.4.32 「[NDB Cluster プログラムに共通のオプション](#) — NDB Cluster プログラムに共通のオプション」

セクション23.4.5 「[ndb_mgm](#) — NDB Cluster 管理クライアント」

セクション23.4.1 「[ndbd](#) — NDB Cluster データノードデーモン」

--connect-string

セクション23.4.32 「[NDB Cluster プログラムに共通のオプション](#) — NDB Cluster プログラムに共通のオプション」

--connect-timeout

セクション4.5.1.1 「[mysql クライアントオプション](#)」

セクション4.5.2 「[mysqladmin](#) — A MySQL Server 管理プログラム」

--connection-control

セクション6.4.2.1 「[Connection-Control プラグインのインストール](#)」

--connection-control-failed-login-attempts

セクション6.4.2.1 「[Connection-Control プラグインのインストール](#)」

--connection-server-id

セクション4.6.8 「[mysqlbinlog](#) — バイナリログファイル进行处理するためのユーティリティー」

セクション4.6.8.4 「[mysqlbinlog](#) サーバー ID の指定」

セクション4.6.8.3 「[バイナリログファイルのバックアップのための mysqlbinlog の使用](#)」

--connection-timeout

セクション23.4.12 「[ndb_error_reporter](#) — NDB エラーレポートユーティリティー」

--connections

セクション23.4.7 「[ndb_config](#) — NDB Cluster 構成情報の抽出」
セクション23.4.13 「[ndb_import](#) — NDB への CSV データのインポート」

--console

セクション15.21 「[InnoDB のトラブルシューティング](#)」
セクション15.17.2 「[InnoDB モニターの有効化](#)」
root のパスワードのリセット: Windows システム
セクション23.2.2.3 「[Windows での NDB Cluster の初期起動](#)」
セクション2.3.4.6 「[Windows のコマンド行からの MySQL の起動](#)」
セクション5.4.2.1 「[エラーログ構成](#)」
セクション2.3.4.5 「[サーバーをはじめて起動する](#)」
セクション5.1.7 「[サーバーコマンドオプション](#)」
セクション5.4.2.2 「[デフォルトのエラーログ保存先の構成](#)」
セクション2.10.1 「[データディレクトリの初期化](#)」

--context

セクション23.1.4 「[NDB Cluster の新機能](#)」
セクション23.4.9 「[ndb_desc](#) — NDB テーブルの表示」

--continue

セクション23.4.13 「[ndb_import](#) — NDB への CSV データのインポート」

copy-back-and-apply-log

セクション2.5.6.2 「[Docker での MySQL Server のデプロイに関するその他のトピック](#)」

--core-file

セクション5.9.1.4 「[gdb での mysqld のデバッグ](#)」
セクション23.4.32 「[NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション](#)」
セクション15.8.3.7 「[コアファイルからのバッファプールページの除外](#)」
セクション5.1.7 「[サーバーコマンドオプション](#)」
セクション5.1.8 「[サーバーシステム変数](#)」

core-file

セクション5.9.1.3 「[WER と PDB を使用した Windows クラッシュダンプの作成](#)」

--core-file-size

セクション4.3.2 「[mysqld_safe](#) — MySQL サーバー起動スクリプト」
セクション2.5.9 「[systemd を使用した MySQL Server の管理](#)」
セクション5.1.7 「[サーバーコマンドオプション](#)」

--correct-checksum

セクション4.6.4.3 「[myisamchk](#) の修復オプション」

--count

セクション4.6.2 「[innochecksum](#) — オフライン InnoDB ファイルチェックサムユーティリティー」
セクション4.6.3 「[myisam_ftdump](#) — 全文インデックス情報の表示」
セクション4.5.2 「[mysqldadmin](#) — A MySQL Server 管理プログラム」
セクション4.5.7 「[mysqlshow](#) — データベース、テーブル、およびカラム情報の表示」

--create

セクション4.5.8 「[mysqlslap](#) — ロードエミュレーションクライアント」

--create-options

セクション4.5.4 「[mysqldump](#) — データベースバックアッププログラム」

--create-schema

セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」

--CSV

セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」

--cxxflags

セクション2.9.8 「MySQL のコンパイルに関する問題」

セクション4.7.1 「mysql_config — クライアントのコンパイル用オプションの表示」

D

[[index top](#)]

-D

セクション16.5 「ARCHIVE ストレージエンジン」

セクション16.6 「BLACKHOLE ストレージエンジン」

セクション4.4.1 「comp_err — MySQL エラーメッセージファイルのコンパイル」

セクション16.9 「EXAMPLE ストレージエンジン」

セクション16.8 「FEDERATED ストレージエンジン」

セクション4.6.2 「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティー」

セクション15.20.3 「InnoDB memcached プラグインの設定」

セクション23.2.1.4 「Linux でのソースからの NDB Cluster の構築」

セクション5.9.3 「LOCK_ORDER ツール」

セクション4.8.1 「lz4_decompress — mysqlpump LZ4-Compressed 出力の解凍」

セクション4.6.4.3 「myisamchk の修復オプション」

セクションA.2 「MySQL 8.0 FAQ: ストレージエンジン」

第23章 「MySQL NDB Cluster 8.0」

セクションB.3.3.3 「MySQL が繰り返しクラッシュする場合の対処方法」

セクション5.9.2 「MySQL クライアントのデバッグ」

セクション4.5.1.1 「mysql クライアントオプション」

セクション2.9.7 「MySQL ソース構成オプション」

セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」

セクション4.5.5 「mysqlimport — データインポートプログラム」

セクション23.5.9 「NDB Cluster での MySQL Server の使用」

セクション23.1.4 「NDB Cluster の新機能」

セクション23.4.24 「ndb_select_all — NDB テーブルの行の出力」

セクション17.1.6.3 「Replica Server のオプションと変数」

セクション2.9.6 「SSL ライブラリサポートの構成」

セクション2.5.9 「systemd を使用した MySQL Server の管理」

セクション23.2.2.2 「Windows でのソースからの NDB Cluster のコンパイルとインストール」

セクション20.5.2 「X プラグイン の無効化」

セクション4.8.3 「zlib_decompress — mysqlpump ZLIB 圧縮出力の解凍」

セクション2.1.2 「インストールする MySQL のバージョンと配布の選択」

セクション5.1.7 「サーバーコマンドオプション」

セクション5.1.8 「サーバーシステム変数」

セクション5.9.1.1 「デバッグのための MySQL のコンパイル」

セクション10.13 「文字セットの追加」

セクションB.3.2.15 「文字セットを初期化できません」

セクション2.9.4 「標準ソース配布を使用して MySQL をインストールする」

-d

セクション2.5.6.1 「Docker を使用した MySQL Server デプロイメントの基本ステップ」

セクション4.6.1 「ibd2sdi — InnoDB テーブルスペース SDI 抽出ユーティリティー」

セクション6.4.4.4 「keyring_okv KMIP プラグインの使用」

セクション4.6.3 「myisam_ftdump — 全文インデックス情報の表示」

セクション4.6.4.1 「myisamchk の一般オプション」

セクション4.6.8 「mysqlbinlog — バイナリログファイル进行处理するためのユーティリティー」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.6.9 「mysqldumpslow — スロークエリログファイルの要約」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション23.4.6 「ndb_blob_tool — NDB Cluster テーブルの BLOB および TEXT カラムのチェックおよび修復」
セクション23.4.8 「ndb_delete_all — NDB テーブルからのすべての行の削除」
セクション23.4.9 「ndb_desc — NDB テーブルの表示」
セクション23.4.10 「ndb_drop_index — NDB テーブルからのインデックスの削除」
セクション23.4.11 「ndb_drop_table — NDB テーブルの削除」
セクション23.4.14 「ndb_index_stat — NDB インデックス統計ユーティリティー」
セクション23.4.4 「ndb_mgmd — NDB Cluster 管理サーバーデーモン」
セクション23.4.15 「ndb_move_data — NDB データコピーユーティリティー」
セクション23.4.22 「ndb_redo_log_reader — クラスタ redo ログの内容の確認および印刷」
セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」
セクション23.4.24 「ndb_select_all — NDB テーブルの行の出力」
セクション23.4.25 「ndb_select_count — NDB テーブルの行数の出力」
セクション23.4.26 「ndb_setup.py — NDB Cluster 用ブラウザベースの Auto-Installer の起動 (非推奨)」
セクション23.4.27 「ndb_show_tables — NDB テーブルのリストの表示」
セクション23.4.1 「ndbd — NDB Cluster データノードデーモン」
セクション23.4.2 「ndbinfo_select_all — ndbinfo テーブルからの選択」
セクション4.6.4.4 「その他の myisamchk オプション」
セクション5.1.8 「サーバーシステム変数」

--daemon

セクション23.4.4 「ndb_mgmd — NDB Cluster 管理サーバーデーモン」
セクション23.4.1 「ndbd — NDB Cluster データノードデーモン」

--daemonize

セクション5.1.7 「サーバーコマンドオプション」

--data-file-length

セクション4.6.4.3 「myisamchk の修復オプション」

--database

セクション4.5.1.1 「mysql クライアントオプション」
セクション4.6.8 「mysqlbinlog — バイナリログファイル进行处理するためのユーティリティー」
セクション23.4.6 「ndb_blob_tool — NDB Cluster テーブルの BLOB および TEXT カラムのチェックおよび修復」
セクション23.4.8 「ndb_delete_all — NDB テーブルからのすべての行の削除」
セクション23.4.9 「ndb_desc — NDB テーブルの表示」
セクション23.4.10 「ndb_drop_index — NDB テーブルからのインデックスの削除」
セクション23.4.11 「ndb_drop_table — NDB テーブルの削除」
セクション23.4.14 「ndb_index_stat — NDB インデックス統計ユーティリティー」
セクション23.4.15 「ndb_move_data — NDB データコピーユーティリティー」
セクション23.4.24 「ndb_select_all — NDB テーブルの行の出力」
セクション23.4.25 「ndb_select_count — NDB テーブルの行数の出力」
セクション23.4.27 「ndb_show_tables — NDB テーブルのリストの表示」
セクション23.4.28 「ndb_size.pl — NDBCLUSTER サイズ要件エスエイメータ」
セクション23.4.2 「ndbinfo_select_all — ndbinfo テーブルからの選択」

--databases

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション7.4.1 「mysqldump による SQL フォーマットでのデータのダンプ」
mysqldump を使用したデータスナップショットの作成
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション23.5.14 「ndbinfo: NDB Cluster 情報データベース」
セクション7.4.2 「SQL フォーマットバックアップのリロード」
セクション7.4.5.2 「サーバー間でのデータベースのコピー」

セクション2.11.13 「テーブルまたはインデックスの再作成または修復」
セクション7.4.5.1 「データベースのコピーの作成」

--datadir

セクション5.8 「1つのマシン上での複数のMySQLインスタンスの実行」
セクション2.10.2.1 「MySQL Serverの起動時の問題のトラブルシューティング」
セクション2.9.7 「MySQLソース構成オプション」
セクション5.2 「MySQLデータディレクトリ」
セクション4.4.3 「mysql_ssl_rsa_setup — SSL/RSAファイルの作成」
セクション4.3.2 「mysqld_safe — MySQLサーバー起動スクリプト」
セクション23.5.17.3 「NDB ClusterおよびMySQLセキュリティー手順」
セクション5.8.3 「Unix上での複数のMySQLインスタンスの実行」
セクション2.3.4.2 「オプションファイルの作成」
セクション4.2.2.2 「オプションファイルの使用」
セクション5.1.7 「サーバーコマンドオプション」
セクション2.10.1 「データディレクトリの初期化」
セクション5.8.1 「複数のデータディレクトリのセットアップ」

datadir

セクション2.4.1 「macOSへのMySQLのインストールに関する一般的なノート」
セクション2.3.5 「Microsoft Windows MySQL Serverインストールのトラブルシューティング」
セクション4.3.3 「mysql.server — MySQLサーバー起動スクリプト」
セクション2.3.7 「Windowsプラットフォームの制限事項」
セクション2.3.4.2 「オプションファイルの作成」

--db-workers

セクション23.4.13 「ndb_import — NDBへのCSVデータのインポート」

--ddl-rewriter

セクション5.6.5.1 「ddl_rewriterのインストールまたはアンインストール」
セクション5.6.5.2 「ddl_rewriterプラグインオプション」

--debug

セクション4.4.1 「comp_err — MySQLエラーメッセージファイルのコンパイル」
セクション5.9.4 「DEBUGパッケージ」
セクション4.6.1 「ibd2sdi — InnoDBテーブルスペースSDI抽出ユーティリティ」
セクション4.7.2 「my_print_defaults — オプションファイルからのオプションの表示」
セクション4.6.4.1 「myisamchkの一般オプション」
セクション4.6.6 「myisampack — 圧縮された読み取り専用のMyISAMテーブルの生成」
セクション2.10.2.1 「MySQL Serverの起動時の問題のトラブルシューティング」
セクション6.2.21 「MySQLへの接続の問題のトラブルシューティング」
セクション4.5.1.1 「mysqlクライアントオプション」
セクション2.9.7 「MySQLソース構成オプション」
セクション4.6.7 「mysql_config_editor — MySQL構成ユーティリティー」
セクション4.4.5 「mysql_upgrade — MySQLテーブルのチェックとアップグレード」
セクション4.5.2 「mysqladmin — A MySQL Server管理プログラム」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.6.9 「mysqldumpslow — スロークエリログファイルの要約」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」
セクション23.4.32 「NDB Clusterプログラムに共通のオプション — NDB Clusterプログラムに共通のオプション」
セクション2.3.4.6 「Windowsのコマンド行からのMySQLの起動」
セクション5.1.7 「サーバーコマンドオプション」
セクション5.1.8 「サーバーシステム変数」

セクション5.9.1.1「デバッグのためのMySQLのコンパイル」

debug

セクション14.1「データディクショナリスキーマ」

--debug-check

セクション4.5.1.1「mysqlクライアントオプション」

セクション4.4.5「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

セクション4.5.2「mysqldadmin — A MySQL Server 管理プログラム」

セクション4.6.8「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」

セクション4.5.3「mysqlcheck — テーブル保守プログラム」

セクション4.5.4「mysqldump — データベースバックアッププログラム」

セクション4.5.5「mysqlimport — データインポートプログラム」

セクション4.5.6「mysqlpump — データベースバックアッププログラム」

セクション4.5.7「mysqlshow — データベース、テーブル、およびカラム情報の表示」

セクション4.5.8「mysqslap — ロードエミュレーションクライアント」

--debug-info

セクション4.4.1「comp_err — MySQL エラーメッセージファイルのコンパイル」

セクション4.5.1.1「mysqlクライアントオプション」

セクション4.4.5「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

セクション4.5.2「mysqldadmin — A MySQL Server 管理プログラム」

セクション4.6.8「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」

セクション4.5.3「mysqlcheck — テーブル保守プログラム」

セクション4.5.4「mysqldump — データベースバックアッププログラム」

セクション4.5.5「mysqlimport — データインポートプログラム」

セクション4.5.6「mysqlpump — データベースバックアッププログラム」

セクション4.5.7「mysqlshow — データベース、テーブル、およびカラム情報の表示」

セクション4.5.8「mysqslap — ロードエミュレーションクライアント」

--debug-level

セクション23.4.26「ndb_setup.py — NDB Cluster 用ブラウザベースの Auto-Installer の起動 (非推奨)」

--debug-sync-timeout

セクション2.9.7「MySQL ソース構成オプション」

セクション5.1.7「サーバーコマンドオプション」

セクション5.1.8「サーバーシステム変数」

--decrypt

セクション23.1.4「NDB Cluster の新機能」

セクション23.5.8.2「NDB Cluster 管理クライアントを使用したバックアップの作成」

セクション23.4.23「ndb_restore — NDB Cluster バックアップの復元」

--decrypt-password

セクション23.1.4「NDB Cluster の新機能」

セクション23.5.8.2「NDB Cluster 管理クライアントを使用したバックアップの作成」

セクション23.4.31「ndbxfrm — NDB Cluster によって作成されたファイルの圧縮、圧縮解除、暗号化、および復号化」

--default-auth

セクション2.11.4「MySQL 8.0 での変更」

セクション4.5.1.1「mysqlクライアントオプション」

セクション4.4.5「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

セクション4.5.2「mysqldadmin — A MySQL Server 管理プログラム」

セクション4.6.8「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」

セクション4.5.3「mysqlcheck — テーブル保守プログラム」

セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」
セクション4.2.3 「サーバーに接続するためのコマンドオプション」
セクション6.4.1.1 「ネイティブプラグブル認証」
セクション6.2.17 「プラグブル認証」

--default-authentication-plugin

セクション2.11.4 「MySQL 8.0 での変更」

--default-character-set

セクション13.2.7 「LOAD DATA ステートメント」
セクション4.5.1.6 「mysql クライアントのヒント」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション6.2.1 「アカウントのユーザー名とパスワード」
セクション10.5 「アプリケーションの文字セットおよび照合順序の構成」
セクション5.1.8 「サーバーシステム変数」
セクション4.5.1.5 「テキストファイルから SQL ステートメントを実行する」
セクション10.4 「接続文字セットおよび照合順序」
セクション10.15 「文字セットの構成」

--default-parallelism

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

--default-storage-engine

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション5.1.7 「サーバーコマンドオプション」

--default-time-zone

セクション5.1.15 「MySQL Server でのタイムゾーンのサポート」
セクション5.1.7 「サーバーコマンドオプション」
セクション5.1.8 「サーバーシステム変数」

--default-tmp-storage-engine

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション5.1.7 「サーバーコマンドオプション」

--default.key_buffer_size

セクション5.1.9.5 「構造化システム変数」

DEFAULT_CHARSET

セクション10.5 「アプリケーションの文字セットおよび照合順序の構成」
セクション10.3.2 「サーバー文字セットおよび照合順序」

DEFAULT_COLLATION

セクション10.5 「アプリケーションの文字セットおよび照合順序の構成」
セクション10.3.2 「サーバー文字セットおよび照合順序」

--defaults-extra-file

セクション4.7.2 「my_print_defaults — オプションファイルからのオプションの表示」
セクション4.6.4.1 「myisamchk の一般オプション」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.4.2 「mysql_secure_installation — MySQL インストールのセキュリティー改善」
セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.3.4 「mysqld_multi — 複数の MySQL サーバーの管理」
セクション4.3.2 「mysqld_safe — MySQL サーバー起動スクリプト」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」
セクション23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」
セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」
セクション4.2.2.2 「オプションファイルの使用」
セクション4.2.2.3 「オプションファイルの処理に影響するコマンド行オプション」
セクション5.1.7 「サーバーコマンドオプション」
セクション2.10.1 「データディレクトリの初期化」
セクション27.12.14.2 「パフォーマンススキーマ variables_info テーブル」

--defaults-file

セクション5.8 「1 つのマシン上での複数の MySQL インスタンスの実行」
セクション15.8.1 「InnoDB の起動構成」
セクション4.7.2 「my_print_defaults — オプションファイルからのオプションの表示」
セクション4.6.4.1 「myisamchk の一般オプション」
セクション2.11.4 「MySQL 8.0 での変更」
セクション4.5.1.1 「mysql クライアントオプション」
セクション2.9.7 「MySQL ソース構成オプション」
セクション4.4.2 「mysql_secure_installation — MySQL インストールのセキュリティー改善」
セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.3.4 「mysqld_multi — 複数の MySQL サーバーの管理」
セクション4.3.2 「mysqld_safe — MySQL サーバー起動スクリプト」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」
セクション23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」
セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」
root のパスワードのリセット: Windows システム
root パスワードのリセット: Unix および Unix- 類似システム
セクション5.8.3 「Unix 上での複数の MySQL インスタンスの実行」
セクション2.3.4.8 「Windows のサービスとして MySQL を起動する」
セクション5.8.2.1 「Windows コマンド行での複数の MySQL インスタンスの起動」
セクション5.8.2.2 「Windows サービスとして複数の MySQL インスタンスの起動」
セクション4.2.2.3 「オプションファイルの処理に影響するコマンド行オプション」
セクション6.4.4.9 「キーリングキーストア間のキーの移行」
セクション5.1.7 「サーバーコマンドオプション」
セクション5.1.3 「サーバー構成の検証」
セクション2.10.1 「データディレクトリの初期化」
セクション6.1.2.1 「パスワードセキュリティーのためのエンドユーザーガイドライン」
セクション27.12.14.2 「パフォーマンススキーマ variables_info テーブル」

--defaults-group-suffix

セクション4.7.2 「my_print_defaults — オプションファイルからのオプションの表示」
セクション4.6.4.1 「myisamchk の一般オプション」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.4.2 「mysql_secure_installation — MySQL インストールのセキュリティ改善」
セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション4.5.6 「mysqldump — データベースバックアッププログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」
セクション23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」
セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」
セクション4.2.2.3 「オプションファイルの処理に影響するコマンド行オプション」
セクション5.1.7 「サーバーコマンドオプション」
セクション4.9 「環境変数」

--defer-table-indexes

セクション4.5.6 「mysqldump — データベースバックアッププログラム」

--delay

セクション23.4.2 「ndbinfo_select_all — ndbinfo テーブルからの選択」

--delay-key-write

セクション8.11.5 「外部ロック」

--delay_key_write

セクション5.1.9 「システム変数の使用」

delay_key_write

セクション16.2.1 「MyISAM 起動オプション」

--delete

セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション23.4.14 「ndb_index_stat — NDB インデックス統計ユーティリティー」

--delete-master-logs

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

--delete-orphans

セクション23.4.6 「ndb_blob_tool — NDB Cluster テーブルの BLOB および TEXT カラムのチェックおよび修復」

--delimiter

セクション4.5.1.1 「mysql クライアントオプション」
セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」
セクション23.4.24 「ndb_select_all — NDB テーブルの行の出力」

--des-key-file

セクション1.3 「MySQL 8.0 の新機能」

--descending

セクション23.4.24 「ndb_select_all — NDB テーブルの行の出力」

--description

セクション4.6.4.4「その他の myisamchk オプション」

--detach

セクション4.5.8「mysqslap — ロードエミュレーションクライアント」

--diff-default

セクション23.4.7「ndb_config — NDB Cluster 構成情報の抽出」

--disable

セクション4.2.2.4「プログラムオプション修飾子」

--disable-admin-ssl

セクション5.1.7「サーバーコマンドオプション」

--disable-auto-rehash

セクション4.5.1.1「mysql クライアントオプション」

セクション15.8.10.2「非永続的オブティマイザ統計のパラメータの構成」

--disable-indexes

セクション23.4.23「ndb_restore — NDB Cluster バックアップの復元」

元のノードより多くのノードへのリストア

--disable-keys

セクション4.5.4「mysqldump — データベースバックアッププログラム」

--disable-log-bin

セクション17.1.3.1「GTID 形式および格納」

セクション4.6.8「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」

セクション17.1.6.4「バイナリロギングのオプションと変数」

セクション5.4.4「バイナリログ」

セクション17.3.3.1「レプリケーション PRIVILEGE_CHECKS_USER アカウントの権限」

--disable-named-commands

セクション4.5.1.1「mysql クライアントオプション」

--disable-plugin_name

セクション5.6.1「プラグインのインストールおよびアンインストール」

--disable-ssl

セクション1.3「MySQL 8.0 の新機能」

セクション5.1.7「サーバーコマンドオプション」

DISABLE_PSI_THREAD

セクション27.12.19.9「processlist テーブル」

DISABLE_SHARED

セクション1.3「MySQL 8.0 の新機能」

--disconnect-slave-event-count

セクション17.1.6.3「Replica Server のオプションと変数」

--disk

セクション23.4.24「ndb_select_all — NDB テーブルの行の出力」

--diskscan

セクション23.4.8 「`ndb_delete_all` — NDB テーブルからのすべての行の削除」

--dns-srv-name

セクション4.2.6 「DNS SRV レコードを使用したサーバーへの接続」

セクション4.5.1.1 「mysql クライアントオプション」

セクション4.5.1.2 「mysql クライアントコマンド」

do-*

セクション17.2.5 「サーバーがレプリケーションフィルタリングルールをどのように評価するか」

--dont-ignore-systab-0

セクション23.4.23 「`ndb_restore` — NDB Cluster バックアップの復元」

--drop-source

セクション23.4.15 「`ndb_move_data` — NDB データコピーユーティリティー」

--dry-scp

セクション23.4.12 「`ndb_error_reporter` — NDB エラーレポートユーティリティー」

--dump

セクション4.6.3 「`myisam_ftdump` — 全文インデックス情報の表示」

セクション23.4.14 「`ndb_index_stat` — NDB インデックス統計ユーティリティー」

--dump-date

セクション4.5.4 「`mysqldump` — データベースバックアッププログラム」

--dump-file

セクション4.6.1 「`ibd2sdi` — InnoDB テーブルスペース SDI 抽出ユーティリティー」

セクション23.4.6 「`ndb_blob_tool` — NDB Cluster テーブルの BLOB および TEXT カラムのチェックおよび修復」

--dump-slave

セクション4.5.4 「`mysqldump` — データベースバックアッププログラム」

セクション17.5.1.34 「レプリケーションとトランザクションの非一貫性」

E

[[index top](#)]

-E

セクション4.5.1.1 「mysql クライアントオプション」

セクション4.5.2 「`mysqladmin` — A MySQL Server 管理プログラム」

セクション4.5.4 「`mysqldump` — データベースバックアッププログラム」

-e

セクション4.4.1 「`comp_err` — MySQL エラーメッセージファイルのコンパイル」

セクション2.5.6.2 「Docker での MySQL Server のデプロイに関するその他のトピック」

セクション4.6.2 「`innochecksum` — オフライン InnoDB ファイルチェックサムユーティリティー」

セクション13.2.8 「LOAD XML ステートメント」

セクション4.7.2 「`my_print_defaults` — オプションファイルからのオプションの表示」

セクション7.6.2 「MyISAM テーブルのエラーのチェック方法」

セクション4.6.4.5 「`myisamchk` によるテーブル情報の取得」

セクション4.6.4.2 「`myisamchk` のチェックオプション」

セクション4.6.4.1 「`myisamchk` の一般オプション」

セクション4.6.4.3 「`myisamchk` の修復オプション」

セクションA.10 「MySQL 8.0 FAQ: NDB Cluster」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」
セクション23.2.6 「NDB Cluster の安全なシャットダウンと再起動」
セクション23.6.9 「NDB Cluster レプリケーションによる NDB Cluster バックアップ」
セクション23.5.8.2 「NDB Cluster 管理クライアントを使用したバックアップの作成」
セクション23.4.5 「ndb_mgm — NDB Cluster 管理クライアント」
セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」
セクション4.2.2.1 「コマンド行でのオプションの使用」

--early-plugin-load

セクション6.4.4.6 「HashiCorp Vault キーリングプラグインの使用」
セクション6.4.4.5 「keyring_aws Amazon Web Services キーリングプラグインの使用」
セクション6.4.4.3 「keyring_encrypted_file キーリングプラグインの使用」
セクション6.4.4.2 「keyring_file ファイルベースプラグインの使用」
セクション6.4.4.4 「keyring_okv KMIP プラグインの使用」
セクション2.4.3 「MySQL 起動デーモンのインストールおよび使用」
セクション6.4.4.7 「Oracle Cloud Infrastructure Vault キーリングプラグインの使用」
セクション2.11.6 「Unix/Linux での MySQL バイナリまたはパッケージベースのインストールのアップグレード」
セクション6.4.4.1 「キーリングプラグインのインストール」
セクション5.1.7 「サーバーコマンドオプション」
セクション5.6.1 「プラグインのインストールおよびアンインストール」

early-plugin-load

セクション15.13 「InnoDB 保存データ暗号化」

--embedded

セクション1.3 「MySQL 8.0 の新機能」

--embedded-libs

セクション1.3 「MySQL 8.0 の新機能」

--embedded-server

セクション1.3 「MySQL 8.0 の新機能」

--enable-cleartext-plugin

セクション6.4.1.7 「LDAP プラガブル認証」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」
セクション6.4.1.5 「PAM プラガブル認証」
セクション6.4.1.4 「クライアント側クリアテキストプラガブル認証」

--enable-plugin_name

セクション5.6.1 「プラグインのインストールおよびアンインストール」

--enable-ssl

セクション1.3 「MySQL 8.0 の新機能」

enabled

セクション2.5.1 「MySQL Yum リポジトリを使用して MySQL を Linux にインストールする」

ENABLED_LOCAL_INFILE

セクション6.1.6「LOAD DATA LOCAL のセキュリティー上の考慮事項」

セクション2.9.7「MySQL ソース構成オプション」

--encrypt-kdf-iter-count

セクション23.4.31「ndbxfm — NDB Cluster によって作成されたファイルの圧縮、圧縮解除、暗号化、および復号化」

--encrypt-password

セクション23.4.31「ndbxfm — NDB Cluster によって作成されたファイルの圧縮、圧縮解除、暗号化、および復号化」

--end-page

セクション4.6.2「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティー」

--enforce-gtid-consistency

セクション17.1.3.7「GTID ベースレプリケーションの制約」

セクション17.1.6.5「グローバルトランザクション ID システム変数」

--engine

セクション4.5.8「mysqslap — ロードエミュレーションクライアント」

--env

セクション2.5.6.2「Docker での MySQL Server のデプロイに関するその他のトピック」

--errins-delay

セクション23.4.13「ndb_import — NDB への CSV データのインポート」

--errins-type

セクション23.4.13「ndb_import — NDB への CSV データのインポート」

--errmsg-file

セクション4.4.1「comp_err — MySQL エラーメッセージファイルのコンパイル」

--error-insert

セクション23.4.15「ndb_move_data — NDB データコピーユーティリティー」

--event-scheduler

セクション25.4.2「イベントスケジューラの構成」

event-scheduler

セクション25.4.2「イベントスケジューラの構成」

--events

セクション4.5.4「mysqldump — データベースバックアッププログラム」

セクション4.5.6「mysqlpump — データベースバックアッププログラム」

セクション2.11.6「Unix/Linux での MySQL バイナリまたはパッケージベースのインストールのアップグレード」

セクション7.4.5.3「ストアプログラムのダンプ」

セクション7.4.5.4「テーブル定義と内容の個別のダンプ」

セクション14.7「データディクショナリを使用する方法の違い」

セクション4.6.8.3「バイナリログファイルのバックアップのための mysqlbinlog の使用」

--example

セクション4.3.4「mysqld_multi — 複数の MySQL サーバーの管理」

--exclude-*

セクション23.4.23 「[ndb_restore](#) — NDB Cluster バックアップの復元」

--exclude-databases

セクション4.5.6 「[mysqlpump](#) — データベースバックアッププログラム」

セクション23.4.23 「[ndb_restore](#) — NDB Cluster バックアップの復元」

--exclude-events

セクション4.5.6 「[mysqlpump](#) — データベースバックアッププログラム」

--exclude-gtids

セクション4.6.8 「[mysqlbinlog](#) — バイナリログファイルを処理するためのユーティリティー」

--exclude-intermediate-sql-tables

セクション23.4.23 「[ndb_restore](#) — NDB Cluster バックアップの復元」

--exclude-missing-columns

セクション23.4.15 「[ndb_move_data](#) — NDB データコピーユーティリティー」

セクション23.4.23 「[ndb_restore](#) — NDB Cluster バックアップの復元」

--exclude-missing-tables

セクション23.4.23 「[ndb_restore](#) — NDB Cluster バックアップの復元」

--exclude-routines

セクション4.5.6 「[mysqlpump](#) — データベースバックアッププログラム」

--exclude-tables

セクション4.5.6 「[mysqlpump](#) — データベースバックアッププログラム」

セクション23.4.23 「[ndb_restore](#) — NDB Cluster バックアップの復元」

--exclude-triggers

セクション4.5.6 「[mysqlpump](#) — データベースバックアッププログラム」

--exclude-users

セクション4.5.6 「[mysqlpump](#) — データベースバックアッププログラム」

--excludedbs

セクション23.4.28 「[ndb_size.pl](#) — NDBCLUSTER サイズ要件エスティメータ」

--excludetables

セクション23.4.28 「[ndb_size.pl](#) — NDBCLUSTER サイズ要件エスティメータ」

--execute

セクション4.5.1.1 「[mysql](#) クライアントオプション」

セクション4.5.1.3 「[mysql](#) クライアントロギング」

セクション23.5.8.2 「NDB Cluster 管理クライアントを使用したバックアップの作成」

セクション23.4.5 「[ndb_mgm](#) — NDB Cluster 管理クライアント」

セクション4.2.2.1 「コマンド行でのオプションの使用」

--exit-info

セクション5.1.7 「サーバーコマンドオプション」

--extend-check

セクション4.6.4.2 「[myisamchk](#) のチェックオプション」

セクション4.6.4.1 「myisamchk の一般オプション」

セクション4.6.4.3 「myisamchk の修復オプション」

--extended

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

--extended-insert

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

--external-locking

セクション16.2.1 「MyISAM 起動オプション」

セクション5.1.7 「サーバーコマンドオプション」

セクション5.1.8 「サーバーシステム変数」

セクション8.11.5 「外部ロック」

--extra-file

セクション4.7.2 「my_print_defaults — オプションファイルからのオプションの表示」

--extra-node-info

セクション23.4.9 「ndb_desc — NDB テーブルの表示」

--extra-partition-info

セクション23.4.9 「ndb_desc — NDB テーブルの表示」

セクション23.5.14.6 「ndbinfo cluster_operations テーブル」

セクション23.5.14.45 「ndbinfo server_operations テーブル」

F

[[index top](#)]

-F

セクション4.4.1 「comp_err — MySQL エラーメッセージファイルのコンパイル」

セクション4.6.4.2 「myisamchk のチェックオプション」

セクション4.6.5 「myisamlog — MyISAM ログファイルの内容の表示」

セクション4.5.1.2 「mysql クライアントコマンド」

セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」

-f

セクション4.6.4.2 「myisamchk のチェックオプション」

セクション4.6.4.3 「myisamchk の修復オプション」

セクション4.6.5 「myisamlog — MyISAM ログファイルの内容の表示」

セクション4.6.6 「myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成」

セクションA.10 「MySQL 8.0 FAQ: NDB Cluster」

セクション4.5.1.1 「mysql クライアントオプション」

セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」

セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.5 「mysqlimport — データインポートプログラム」

セクション23.2.4 「NDB Cluster の初期起動」

セクション23.5.1 「NDB Cluster 管理クライアントのコマンド」

セクション23.4.7 「ndb_config — NDB Cluster 構成情報の抽出」

セクション23.4.4 「ndb_mgmd — NDB Cluster 管理サーバーデーモン」
セクション23.4.22 「ndb_redo_log_reader — クラスタ redo ログの内容の確認および印刷」
セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」
セクション23.2.2.3 「Windows での NDB Cluster の初期起動」
セクション5.9.1.5 「スタックトレースの使用」

--fast

セクション4.6.4.2 「myisamchk のチェックオプション」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

--federated

セクション16.8 「FEDERATED ストレージエンジン」

--fields

セクション23.4.7 「ndb_config — NDB Cluster 構成情報の抽出」

--fields-enclosed-by

セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション7.4.3 「mysqldump による区切りテキストフォーマットでのデータのダンプ」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション23.4.13 「ndb_import — NDB への CSV データのインポート」
セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

--fields-escaped-by

セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション7.4.3 「mysqldump による区切りテキストフォーマットでのデータのダンプ」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション23.4.13 「ndb_import — NDB への CSV データのインポート」

--fields-optionally-enclosed-by

セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション7.4.3 「mysqldump による区切りテキストフォーマットでのデータのダンプ」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション23.4.13 「ndb_import — NDB への CSV データのインポート」
セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

--fields-terminated-by

セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション7.4.3 「mysqldump による区切りテキストフォーマットでのデータのダンプ」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション23.4.13 「ndb_import — NDB への CSV データのインポート」
セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

--fields-xxx

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

--fix-db-names

セクション1.3 「MySQL 8.0 の新機能」

--fix-table-names

セクション1.3 「MySQL 8.0 の新機能」

--flush

セクションB.3.3.3 「MySQL が繰り返しクラッシュする場合の対処方法」
セクション5.1.7 「サーバーコマンドオプション」
セクション5.1.8 「サーバーシステム変数」

--flush-logs

セクション5.4「MySQL Server ログ」
セクション6.2.2「MySQL で提供される権限」
セクション4.5.4「mysqldump — データベースバックアッププログラム」
セクション7.3.1「バックアップポリシーの確立」

--flush-privileges

セクション4.5.4「mysqldump — データベースバックアッププログラム」

--force

セクション4.6.4.2「myisamchk のチェックオプション」
セクション4.6.4.3「myisamchk の修復オプション」
セクション4.6.6「myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成」
セクション2.11.3「MySQL のアップグレードプロセスの内容」
セクション4.5.1.1「mysql クライアントオプション」
セクション4.4.5「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
セクション4.5.2「mysqladmin — A MySQL Server 管理プログラム」
セクション4.5.3「mysqlcheck — テーブル保守プログラム」
セクション4.5.4「mysqldump — データベースバックアッププログラム」
セクション4.5.5「mysqlimport — データインポートプログラム」
セクション23.6.4「NDB Cluster レプリケーションスキーマおよびテーブル」
セクション3.5「バッチモードでの MySQL の使用」

--force-if-open

セクション4.6.8「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」

--force-read

セクション4.6.8「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」

--foreground

セクション23.4.1「ndbd — NDB Cluster データノードデーモン」

--format

セクション23.4.28「ndb_size.pl — NDBCLUSTER サイズ要件エスティメータ」

FPROFILE_GENERATE

セクション2.9.7「MySQL ソース構成オプション」

FPROFILE_USE

セクション2.9.7「MySQL ソース構成オプション」

--fs

セクション23.4.12「ndb_error_reporter — NDB エラーレポートユーティリティー」

G

[index top]

-G

セクション4.5.1.1「mysql クライアントオプション」
セクション4.6.7「mysql_config_editor — MySQL 構成ユーティリティー」

-g

セクション4.7.2「my_print_defaults — オプションファイルからのオプションの表示」

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.6.9 「mysqldumpslow — スロークエリールログファイルの要約」
セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」
セクション5.9.1.1 「デバッグのための MySQL のコンパイル」

--gci

セクション23.4.24 「ndb_select_all — NDB テーブルの行の出力」

--gci64

セクション23.4.24 「ndb_select_all — NDB テーブルの行の出力」

--gdb

セクション5.9.1.4 「gdb での mysqld のデバッグ」
セクション4.10 「MySQL での Unix シグナル処理」
セクション13.7.8.8 「RESTART ステートメント」
セクション5.1.7 「サーバーコマンドオプション」

--general-log

セクション4.2.2.1 「コマンド行でのオプションの使用」
セクション5.1.9 「システム変数の使用」

--general_log

セクション4.2.2.1 「コマンド行でのオプションの使用」
セクション5.1.9 「システム変数の使用」
セクション5.4.3 「一般クエリールログ」
セクション5.4.1 「一般クエリールログおよびスロークエリールログの出力先の選択」

--general_log_file

セクション5.8 「1 つのマシン上での複数の MySQL インスタンスの実行」
セクション5.1.8 「サーバーシステム変数」
セクション5.4.3 「一般クエリールログ」

--get-server-public-key

セクション4.5.1.1 「mysql クライアントオプション」
セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション4.5.6 「mysqldumpslow — データベースバックアッププログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション4.5.8 「mysqldumpslow — ロードエミュレーションクライアント」
セクション6.4.1.2 「SHA-2 プラガブル認証のキャッシュ」
セクション4.2.3 「サーバーに接続するためのコマンドオプション」

--graph

セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」

H

[[index top](#)]

-H

セクション4.4.1 「comp_err — MySQL エラーメッセージファイルのコンパイル」
セクション4.6.4.1 「myisamchk の一般オプション」

セクション4.5.1.1 「mysql クライアントオプション」
セクション4.6.8 「mysqlbinlog — バイナリログファイル进行处理するためのユーティリティ」
セクション23.4.26 「ndb_setup.py — NDB Cluster 用ブラウザベースの Auto-Installer の起動 (非推奨)」

-h

セクション4.6.1 「ibd2sdi — InnoDB テーブルスペース SDI 抽出ユーティリティ」
セクション4.6.3 「mysam_ftdump — 全文インデックス情報の表示」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.2.1 「MySQL プログラムの起動」
セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティ」
セクション4.4.2 「mysql_secure_installation — MySQL インストールのセキュリティ改善」
セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」
セクション4.6.8 「mysqlbinlog — バイナリログファイル进行处理するためのユーティリティ」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.6.9 「mysqldumpslow — スロークエリログファイルの要約」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」
セクション23.4.18 「ndb_print_file — NDB ディスクデータファイル内容の出力」
セクション23.4.24 「ndb_select_all — NDB テーブルの行の出力」
セクション23.4.26 「ndb_setup.py — NDB Cluster 用ブラウザベースの Auto-Installer の起動 (非推奨)」
セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」
セクション1.1 「このマニュアルについて」
セクション4.2.4 「コマンドオプションを使用した MySQL Server への接続」
セクション4.2.2.1 「コマンド行でのオプションの使用」
セクション4.2.3 「サーバーに接続するためのコマンドオプション」
セクション5.1.7 「サーバーコマンドオプション」

HAVE_CRYPT

セクション1.3 「MySQL 8.0 の新機能」

--header

セクション23.4.24 「ndb_select_all — NDB テーブルの行の出力」

--header-file

セクション4.4.1 「comp_err — MySQL エラーメッセージファイルのコンパイル」

--HELP

セクション4.6.4.1 「mysamchk の一般オプション」

--help

セクション4.4.1 「comp_err — MySQL エラーメッセージファイルのコンパイル」
セクション4.6.1 「ibd2sdi — InnoDB テーブルスペース SDI 抽出ユーティリティ」
セクション4.6.2 「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティ」
セクション4.7.2 「my_print_defaults — オプションファイルからのオプションの表示」
セクション4.6.3 「mysam_ftdump — 全文インデックス情報の表示」
セクション4.6.4.1 「mysamchk の一般オプション」
セクション4.6.6 「mysampack — 圧縮された読み取り専用の MyISAM テーブルの生成」
セクション2.10.2.1 「MySQL Server の起動時の問題のトラブルシューティング」
セクション1.2.2 「MySQL の主な機能」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.1 「MySQL プログラムの概要」
セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティ」
セクション4.4.2 「mysql_secure_installation — MySQL インストールのセキュリティ改善」

セクション4.4.3 「mysql_ssl_rsa_setup — SSL/RSA ファイルの作成」
セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
セクション4.5.2 「mysqldadmin — A MySQL Server 管理プログラム」
セクション4.6.8 「mysqlbinlog — バイナリログファイル処理するためのユーティリティ」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.3.4 「mysqld_multi — 複数の MySQL サーバーの管理」
セクション4.3.2 「mysqld_safe — MySQL サーバー起動スクリプト」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.6.9 「mysqldumpslow — スロークエリログファイルの要約」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」
セクション23.1.4 「NDB Cluster の新機能」
セクション23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」
セクション23.2.2.4 「NDB Cluster プロセスを Windows サービスとしてインストール」
セクション23.4.6 「ndb_blob_tool — NDB Cluster テーブルの BLOB および TEXT カラムのチェックおよび修復」
セクション23.4.7 「ndb_config — NDB Cluster 構成情報の抽出」
セクション23.4.16 「ndb_perror — NDB エラーメッセージ情報の取得」
セクション23.4.18 「ndb_print_file — NDB ディスクデータファイル内容の出力」
セクション23.4.22 「ndb_redo_log_reader — クラスタ redo ログの内容の確認および印刷」
セクション23.4.26 「ndb_setup.py — NDB Cluster 用ブラウザベースの Auto-Installer の起動 (非推奨)」
セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」
セクション23.4.31 「ndbxfm — NDB Cluster によって作成されたファイルの圧縮、圧縮解除、暗号化、および復号化」
セクション4.8.2 「 perror — MySQL エラーメッセージ情報の表示」
セクション4.2.2.2 「オプションファイルの使用」
セクション4.2.2.1 「コマンド行でのオプションの使用」
セクション2.10.3 「サーバーのテスト」
セクション5.1.7 「サーバーコマンドオプション」
第3章 「チュートリアル」

--hex

セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

--hex-blob

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

--hexdump

セクション4.6.8.1 「mysqlbinlog 16 進ダンプ形式」

セクション4.6.8 「mysqlbinlog — バイナリログファイル処理するためのユーティリティ」

--histignore

セクション4.5.1.6 「mysql クライアントのヒント」

セクション4.5.1.1 「mysql クライアントオプション」

セクション4.5.1.3 「mysql クライアントロギング」

--host

セクション6.2.21 「MySQL への接続の問題のトラブルシューティング」

セクション4.5.1.1 「mysql クライアントオプション」

セクション4.2.1 「MySQL プログラムの起動」

セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティ」

セクション4.4.2 「mysql_secure_installation — MySQL インストールのセキュリティ改善」

セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

セクション4.5.2 「mysqldadmin — A MySQL Server 管理プログラム」

セクション4.6.8 「mysqlbinlog — バイナリログファイル処理するためのユーティリティ」

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」
セクション23.4.7 「ndb_config — NDB Cluster 構成情報の抽出」
セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」
セクション20.5.6.2 「X プラグイン のオプションとシステム変数」
セクション1.1 「このマニュアルについて」
セクション4.2.2.6 「オプションのデフォルト、値を想定するオプション、および = 記号」
セクション4.2.2.2 「オプションファイルの使用」
セクション4.2.4 「コマンドオプションを使用した MySQL Server への接続」
セクション4.2.2.1 「コマンド行でのオプションの使用」
セクション4.2.3 「サーバーに接続するためのコマンドオプション」
セクション5.1.8 「サーバーシステム変数」
セクション2.10.1 「データディレクトリの初期化」
セクション4.6.8.3 「バイナリログファイルのバックアップのための mysqlbinlog の使用」
セクション5.8.4 「複数サーバー環境でのクライアントプログラムの使用」

host

セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティ」
セクション4.2.2.2 「オプションファイルの使用」

--hostname

セクション23.4.28 「ndb_size.pl — NDBCLUSTER サイズ要件エスティメータ」

--html

セクション4.5.1.1 「mysql クライアントオプション」

|

[index top]

-|

セクション4.6.2 「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティ」
セクション4.6.5 「myisamlog — MyISAM ログファイルの内容の表示」
セクション4.8.2 「perror — MySQL エラーメッセージ情報の表示」
セクション5.1.7 「サーバーコマンドオプション」

-i

セクション4.6.1 「ibd2sdi — InnoDB テーブルスペース SDI 抽出ユーティリティ」
セクション7.6.2 「MyISAM テーブルのエラーのチェック方法」
セクション4.6.4.2 「myisamchk のチェックオプション」
セクション4.6.5 「myisamlog — MyISAM ログファイルの内容の表示」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.6.9 「mysqldumpslow — スロークエリログファイルの要約」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」
セクション23.5.1 「NDB Cluster 管理クライアントのコマンド」
セクション23.4.31 「ndbxfrm — NDB Cluster によって作成されたファイルの圧縮、圧縮解除、暗号化、および復号化」

--i-am-a-dummy

セクション4.5.1.6 「mysql クライアントのヒント」

セクション4.5.1.1 「mysql クライアントオプション」

--id

セクション4.6.1 「ibd2sdi — InnoDB テーブルスペース SDI 抽出ユーティリティ」

--idempotent

セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」

セクション5.1.8 「サーバーシステム変数」

--idlesleep

セクション23.4.13 「ndb_import — NDB への CSV データのインポート」

--idlespin

セクション23.4.13 「ndb_import — NDB への CSV データのインポート」

--ignore

セクション4.5.5 「mysqlimport — データインポートプログラム」

ignore-*

セクション17.2.5 「サーバーがレプリケーションフィルタリングルールをどのように評価するか」

--ignore-db-dir

セクション1.3 「MySQL 8.0 の新機能」

セクション2.11.5 「アップグレード用のインストールの準備」

--ignore-error

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

--ignore-extended-pk-updates

セクション23.1.4 「NDB Cluster の新機能」

セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

--ignore-lines

セクション4.5.5 「mysqlimport — データインポートプログラム」

セクション23.4.13 「ndb_import — NDB への CSV データのインポート」

--ignore-spaces

セクション4.5.1.1 「mysql クライアントオプション」

--ignore-table

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

mysqldump を使用したデータスナップショットの作成

--in-file

セクション4.4.1 「comp_err — MySQL エラーメッセージファイルのコンパイル」

--in-file-errlog

セクション4.4.1 「comp_err — MySQL エラーメッセージファイルのコンパイル」

--in-file-toclient

セクション4.4.1 「comp_err — MySQL エラーメッセージファイルのコンパイル」

--include

セクション4.7.1 「mysql_config — クライアントのコンパイル用オプションの表示」

--include-*

セクション23.4.23 「[ndb_restore](#) — NDB Cluster バックアップの復元」

--include-databases

セクション4.5.6 「[mysqldump](#) — データベースバックアッププログラム」

セクション23.4.23 「[ndb_restore](#) — NDB Cluster バックアップの復元」

--include-events

セクション4.5.6 「[mysqldump](#) — データベースバックアッププログラム」

--include-gtids

セクション4.6.8 「[mysqlbinlog](#) — バイナリログファイルを処理するためのユーティリティー」

--include-master-host-port

セクション4.5.4 「[mysqldump](#) — データベースバックアッププログラム」

--include-routines

セクション4.5.6 「[mysqldump](#) — データベースバックアッププログラム」

--include-stored-grants

セクション23.1.4 「[NDB Cluster](#) の新機能」

セクション23.4.23 「[ndb_restore](#) — NDB Cluster バックアップの復元」

--include-tables

セクション4.5.6 「[mysqldump](#) — データベースバックアッププログラム」

セクション23.4.23 「[ndb_restore](#) — NDB Cluster バックアップの復元」

--include-triggers

セクション4.5.6 「[mysqldump](#) — データベースバックアッププログラム」

--include-users

セクション4.5.6 「[mysqldump](#) — データベースバックアッププログラム」

--info

セクション4.6.2 「[innochecksum](#) — オフライン InnoDB ファイルチェックサムユーティリティー」

セクション23.4.31 「[ndbxfm](#) — NDB Cluster によって作成されたファイルの圧縮、圧縮解除、暗号化、および復号化」

セクション4.8.2 「[perror](#) — MySQL エラーメッセージ情報の表示」

--information

セクション4.6.4.2 「[myisamchk](#) のチェックオプション」

--init-command

セクション4.5.1.1 「[mysql](#) クライアントオプション」

--init_connect

セクション10.5 「アプリケーションの文字セットおよび照合順序の構成」

--initial

セクション23.3.3.10 「[NDB Cluster TCP/IP](#) 接続」

セクション23.3.3.7 「[NDB Cluster](#) での SQL およびその他の API ノードの定義」

セクション23.3.3.4 「[NDB Cluster](#) でのコンピュータの定義」

セクション23.2.7 「NDB Cluster のアップグレードおよびダウングレード」
セクション23.5.5 「NDB Cluster のローリング再起動の実行」
セクション23.3.3.12 「NDB Cluster の共有メモリー接続」
セクション23.1.4 「NDB Cluster の新機能」
セクション23.5.10.2 「NDB Cluster ディスクデータストレージの要件」
セクション23.5.7.2 「NDB Cluster データノードのオンラインでの追加: 基本手順」
セクション23.5.7.3 「NDB Cluster データノードのオンラインでの追加: 詳細な例」
セクション23.3.3.6 「NDB Cluster データノードの定義」
セクション23.6.3 「NDB Cluster レプリケーションの既知の問題」
セクション23.6.9.2 「NDB Cluster レプリケーションを使用したポイントインタイムリカバリ」
セクション23.3.2 「NDB Cluster 構成パラメータ、オプション、および変数の概要」
セクション23.3.3 「NDB Cluster 構成ファイル」
セクション23.5.1 「NDB Cluster 管理クライアントのコマンド」
セクション23.3.3.5 「NDB Cluster 管理サーバーの定義」
セクション23.5.4 「NDB Cluster 起動フェーズのサマリー」
セクション23.4.7 「ndb_config — NDB Cluster 構成情報の抽出」
セクション23.4.4 「ndb_mgmd — NDB Cluster 管理サーバーデーモン」
セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」
セクション23.4.1 「ndbd — NDB Cluster データノードデーモン」
セクション23.4.3 「ndbmtd — NDB Cluster データノードデーモン (マルチスレッド)」
セクション23.2.2.3 「Windows での NDB Cluster の初期起動」
セクション23.3.3.8 「システムの定義」
元のノードより少ないノードへのリストア
セクション23.1.7.10 「複数の NDB Cluster ノードに関する制限事項」

--initial-start

セクション23.4.1 「ndbd — NDB Cluster データノードデーモン」

--initialize

セクション23.2.1.1 「Linux への NDB Cluster バイナリリリースのインストール」
セクション2.11.4 「MySQL 8.0 での変更」
セクション1.3 「MySQL 8.0 の新機能」
第28章 「MySQL sys スキーマ」
NDB Cluster の MySQL Server オプション
セクション2.3.4.2 「オプションファイルの作成」
セクション5.1.7 「サーバーコマンドオプション」
セクション5.1.8 「サーバーシステム変数」
セクション2.10.1 「データディレクトリの初期化」
セクション17.1.6.4 「バイナリロギングのオプションと変数」
セクション5.4.4 「バイナリログ」

--initialize-insecure

セクション23.2.1.1 「Linux への NDB Cluster バイナリリリースのインストール」
セクション2.11.4 「MySQL 8.0 での変更」
セクション1.3 「MySQL 8.0 の新機能」
第28章 「MySQL sys スキーマ」
セクション2.3.4.2 「オプションファイルの作成」
セクション5.1.7 「サーバーコマンドオプション」
セクション5.1.8 「サーバーシステム変数」
セクション2.10.1 「データディレクトリの初期化」
セクション17.1.6.4 「バイナリロギングのオプションと変数」
セクション5.4.4 「バイナリログ」

--innodb

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

--innodb-adaptive-hash-index

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

--innodb-file-per-table

[セクション5.1.7「サーバーコマンドオプション」](#)

innodb-file-per-table

[セクション5.1.7「サーバーコマンドオプション」](#)

--innodb-rollback-on-timeout

[セクション15.21.4「InnoDBのエラー処理」](#)

[セクション15.14「InnoDBの起動オプションおよびシステム変数」](#)

--innodb-status-file

[セクション15.14「InnoDBの起動オプションおよびシステム変数」](#)

[セクション15.17.2「InnoDB モニターの有効化」](#)

--innodb-xxx

[セクション5.1.7「サーバーコマンドオプション」](#)

innodb_file_per_table

[ローデータファイルを使用したデータスナップショットの作成](#)

--input-type

[セクション23.4.13「ndb_import — NDB への CSV データのインポート」](#)

--input-workers

[セクション23.4.13「ndb_import — NDB への CSV データのインポート」](#)

--insert-ignore

[セクション4.5.4「mysqldump — データベースバックアッププログラム」](#)

[セクション4.5.6「mysqlpump — データベースバックアッププログラム」](#)

--install

[セクション23.2.2.4「NDB Cluster プロセスを Windows サービスとしてインストール」](#)

[セクション23.4.4「ndb_mgrmd — NDB Cluster 管理サーバーデーモン」](#)

[セクション23.4.1「ndbd — NDB Cluster データノードデーモン」](#)

[セクション2.3.4.8「Windows のサービスとして MySQL を起動する」](#)

[セクション5.8.2.2「Windows サービスとして複数の MySQL インスタンスの起動」](#)

[セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」](#)

[セクション5.1.7「サーバーコマンドオプション」](#)

--install-manual

[セクション2.3.4.8「Windows のサービスとして MySQL を起動する」](#)

[セクション5.8.2.2「Windows サービスとして複数の MySQL インスタンスの起動」](#)

[セクション5.1.7「サーバーコマンドオプション」](#)

INSTALL_LAYOUT

[セクション2.9.7「MySQL ソース構成オプション」](#)

[セクション6.4.4.13「キーリングシステム変数」](#)

[セクション5.1.8「サーバーシステム変数」](#)

INSTALL_LIBDIR

[セクション2.9.7「MySQL ソース構成オプション」](#)

INSTALL_MYSQLKEYRINGDIR

[セクション6.4.4.13「キーリングシステム変数」](#)

INSTALL_PRIV_LIBDIR

セクション2.9.7「MySQL ソース構成オプション」

INSTALL_SCRIPTDIR

セクション1.3「MySQL 8.0 の新機能」

INSTALL_SECURE_FILE_PRIV_EMBEDDEDIR

セクション1.3「MySQL 8.0 の新機能」

INSTALL_SECURE_FILE_PRIVDIR

セクション5.1.8「サーバーシステム変数」

--interactive

セクション23.4.4「ndb_mgmd — NDB Cluster 管理サーバーデーモン」

--iterations

セクション4.5.8「mysqlslap — ロードエミュレーションクライアント」

J

[\[index top\]](#)

-j

セクション4.6.6「myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成」

セクション4.5.1.1「mysql クライアントオプション」

セクション4.6.8「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」

--join

セクション4.6.6「myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成」

K

[\[index top\]](#)

-K

セクション4.5.4「mysqldump — データベースバックアッププログラム」

-k

セクション4.6.4.3「myisamchk の修復オプション」

セクション4.4.5「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

セクション4.5.7「mysqlshow — データベース、テーブル、およびカラム情報の表示」

セクション23.4.26「ndb_setup.py — NDB Cluster 用ブラウザベースの Auto-Installer の起動 (非推奨)」

セクション23.4.31「ndbxfrm — NDB Cluster によって作成されたファイルの圧縮、圧縮解除、暗号化、および復号化」

--keep-state

セクション23.4.13「ndb_import — NDB への CSV データのインポート」

--keep_files_on_create

セクション13.1.20「CREATE TABLE ステートメント」

--key-file

セクション23.4.26「ndb_setup.py — NDB Cluster 用ブラウザベースの Auto-Installer の起動 (非推奨)」

--keyring-migration-destination

セクション6.4.4.9 「キーリングキーストア間のキーの移行」
セクション6.4.4.12 「キーリングコマンドのオプション」

--keyring-migration-host

セクション6.4.4.9 「キーリングキーストア間のキーの移行」
セクション6.4.4.12 「キーリングコマンドのオプション」

--keyring-migration-password

セクション6.4.4.9 「キーリングキーストア間のキーの移行」
セクション6.4.4.12 「キーリングコマンドのオプション」

--keyring-migration-port

セクション6.4.4.9 「キーリングキーストア間のキーの移行」
セクション6.4.4.12 「キーリングコマンドのオプション」

--keyring-migration-socket

セクション6.4.4.9 「キーリングキーストア間のキーの移行」
セクション6.4.4.12 「キーリングコマンドのオプション」

--keyring-migration-source

セクション6.4.4.9 「キーリングキーストア間のキーの移行」
セクション6.4.4.12 「キーリングコマンドのオプション」

--keyring-migration-user

セクション6.4.4.9 「キーリングキーストア間のキーの移行」
セクション6.4.4.12 「キーリングコマンドのオプション」

--keys

セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」

--keys-used

セクション4.6.4.3 「myisamchk の修復オプション」

L

[[index top](#)]

-L

セクション4.5.1.1 「mysql クライアントオプション」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」
セクション2.13.3 「Perl DBI/DBD インタフェース使用の問題」

-l

セクション4.6.2 「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティ」
セクション4.7.2 「my_print_defaults — オプションファイルからのオプションの表示」
セクション4.6.3 「myisam_ftdump — 全文インデックス情報の表示」
セクション4.6.4.3 「myisamchk の修復オプション」
セクション2.9.7 「MySQL ソース構成オプション」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.6.9 「mysqldumpslow — スロークエリログファイルの要約」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション23.4.15 「ndb_move_data — NDB データコピーユーティリティ」

セクション23.4.22 「[ndb_redo_log_reader](#) — クラスタ redo ログの内容の確認および印刷」
セクション23.4.24 「[ndb_select_all](#) — NDB テーブルの行の出力」
セクション23.4.25 「[ndb_select_count](#) — NDB テーブルの行数の出力」
セクション23.4.27 「[ndb_show_tables](#) — NDB テーブルのリストの表示」
セクション23.4.2 「[ndbinfo_select_all](#) — ndbinfo テーブルからの選択」

--language

セクション5.1.7 「[サーバーコマンドオプション](#)」

--large-pages

セクション5.1.7 「[サーバーコマンドオプション](#)」
セクション5.1.8 「[サーバーシステム変数](#)」
セクション8.12.3.2 「[ラージページのサポートの有効化](#)」

--lc-messages

セクション5.1.7 「[サーバーコマンドオプション](#)」

--lc-messages-dir

セクション5.1.7 「[サーバーコマンドオプション](#)」

--ledir

セクション4.3.2 「[mysqld_safe](#) — MySQL サーバー起動スクリプト」

--length

セクション4.6.3 「[myisam_ftdump](#) — 全文インデックス情報の表示」

--libmysqld-libs

セクション1.3 「[MySQL 8.0 の新機能](#)」

--libs

セクション4.7.1 「[mysql_config](#) — クライアントのコンパイル用オプションの表示」

--libs_r

セクション4.7.1 「[mysql_config](#) — クライアントのコンパイル用オプションの表示」

--line-numbers

セクション4.5.1.1 「[mysql](#) クライアントオプション」

--lines-terminated-by

セクション4.5.4 「[mysqldump](#) — データベースバックアッププログラム」
セクション7.4.3 「[mysqldump](#) による区切りテキストフォーマットでのデータのダンプ」
セクション4.5.5 「[mysqlimport](#) — データインポートプログラム」
セクション23.4.13 「[ndb_import](#) — NDB への CSV データのインポート」
セクション23.4.23 「[ndb_restore](#) — NDB Cluster バックアップの復元」

LINK_RANDOMIZE

セクション2.9.7 「[MySQL ソース構成オプション](#)」

--load-data-local-dir

セクション6.1.6 「[LOAD DATA LOCAL](#) のセキュリティ上の考慮事項」
セクション4.5.1.1 「[mysql](#) クライアントオプション」
セクション4.6.8 「[mysqlbinlog](#) — バイナリログファイルを処理するためのユーティリティ」

--loadqueries

セクション23.4.28 「[ndb_size.pl](#) — NDBCLUSTER サイズ要件エスティメータ」

--local

[セクション6.1.6「LOAD DATA LOCAL のセキュリティー上の考慮事項」](#)

[セクション4.5.5「mysqlimport — データインポートプログラム」](#)

--local-infile

[セクション6.1.6「LOAD DATA LOCAL のセキュリティー上の考慮事項」](#)

[セクション13.2.8「LOAD XML ステートメント」](#)

[セクション4.5.1.1「mysql クライアントオプション」](#)

--local-load

[セクション6.1.6「LOAD DATA LOCAL のセキュリティー上の考慮事項」](#)

[セクション4.6.8「mysqlbinlog — バイナリログファイル进行处理するためのユーティリティー」](#)

--local-service

[セクション2.3.4.8「Windows のサービスとして MySQL を起動する」](#)

[セクション5.1.7「サーバーコマンドオプション」](#)

--lock

[セクション23.4.24「ndb_select_all — NDB テーブルの行の出力」](#)

[セクション23.4.25「ndb_select_count — NDB テーブルの行数の出力」](#)

--lock-all-tables

[セクション4.5.4「mysqldump — データベースバックアッププログラム」](#)

--lock-order

[セクション5.9.3「LOCK_ORDER ツール」](#)

--lock-tables

[セクション4.5.4「mysqldump — データベースバックアッププログラム」](#)

[セクション4.5.5「mysqlimport — データインポートプログラム」](#)

--log

[セクション4.6.2「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティー」](#)

[セクション4.3.4「mysqld_multi — 複数の MySQL サーバーの管理」](#)

--log-bin

[セクション5.8「1 つのマシン上での複数の MySQL インスタンスの実行」](#)

[セクションB.3.7「MySQL の既知の問題」](#)

[セクション23.1.7.1「NDB Cluster の SQL 構文に準拠していません」](#)

[セクション13.4.1.1「PURGE BINARY LOGS ステートメント」](#)

[セクション17.1.6.3「Replica Server のオプションと変数」](#)

[セクション18.4.6「グループレプリケーションでの MySQL Enterprise Backup の使用」](#)

[セクション15.6.3.6「サーバーがオフラインのときのテーブルスペースファイルの移動」](#)

[セクション17.1.6.4「バイナリロギングのオプションと変数」](#)

[セクション5.4.4「バイナリログ」](#)

[セクション7.5.1「バイナリログを使用したポイントインタイムリカバリ」](#)

[セクション7.3.1「バックアップポリシーの確立」](#)

[セクション17.4.8「フェイルオーバー中のソースの切替え」](#)

[セクション7.3.2「リカバリへのバックアップの使用」](#)

[セクション17.1.2.1「レプリケーションソース構成の設定」](#)

--log-bin-index

[セクション23.1.7.1「NDB Cluster の SQL 構文に準拠していません」](#)

[セクション17.1.6.3「Replica Server のオプションと変数」](#)

[セクション17.1.6.4「バイナリロギングのオプションと変数」](#)

セクション5.4.4 「バイナリログ」

--log-error

セクション5.8 「1つのマシン上での複数のMySQLインスタンスの実行」
セクション2.5.6.2 「DockerでのMySQL Serverのデプロイに関するその他のトピック」
セクション4.3.2 「mysqld_safe — MySQLサーバー起動スクリプト」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション2.3.4.6 「Windowsのコマンド行からのMySQLの起動」
セクション5.4.2.1 「エラーログ構成」
セクション4.2.2.6 「オプションのデフォルト、値を想定するオプション、および = 記号」
セクション2.3.4.5 「サーバーをはじめて起動する」
セクション5.1.7 「サーバーコマンドオプション」
セクション5.4.6 「サーバーログの保守」
セクション5.4.2.2 「デフォルトのエラーログ保存先の構成」

--log-error-file

セクション4.5.6 「mysqldump — データベースバックアッププログラム」

--log-isam

セクション4.6.5 「myisamlog — MyISAM ログファイルの内容の表示」
セクション5.1.7 「サーバーコマンドオプション」

--log-level

セクション23.4.13 「ndb_import — NDB への CSV データのインポート」

--log-name

セクション23.4.4 「ndb_mgmd — NDB Cluster 管理サーバーデーモン」

--log-raw

セクション5.1.7 「サーバーコマンドオプション」
セクション5.1.8 「サーバーシステム変数」
セクション6.1.2.3 「パスワードおよびロギング」
セクション5.4.3 「一般クエリーログ」

--log-short-format

セクション5.1.7 「サーバーコマンドオプション」
セクション5.4.5 「スロークエリーログ」

--log-slave-updates

セクション17.1.3.4 「GTIDを使用したレプリケーションのセットアップ」
セクション17.1.4.3 「GTID トランザクションのオンラインでの無効化」
セクション17.1.6.3 「Replica Server のオプションと変数」
セクション17.1.6.4 「バイナリロギングのオプションと変数」
セクション5.4.4 「バイナリログ」
セクション17.4.8 「フェイルオーバー中のソースの切替え」
セクション17.1.2.2 「レプリカ構成の設定」
セクション17.5.5 「レプリケーションバグまたは問題を報告する方法」
セクション17.4.7 「レプリケーションパフォーマンスを改善する」

--log-tc

セクション5.1.7 「サーバーコマンドオプション」

--log-tc-size

セクション5.1.7 「サーバーコマンドオプション」
セクション5.1.10 「サーバーステータス変数」

--log-warnings

セクション1.3「MySQL 8.0の新機能」

--log_output

セクション5.4.1「一般クエリログおよびスロークエリログの出力先の選択」

--log_timestamps

セクション4.3.2「mysqld_safe — MySQL サーバー起動スクリプト」

--logbuffer-size

セクション23.4.1「ndbd — NDB Cluster データノードデーモン」

--login-path

セクション4.7.2「my_print_defaults — オプションファイルからのオプションの表示」

セクション4.5.1.1「mysql クライアントオプション」

セクション4.6.7「mysql_config_editor — MySQL 構成ユーティリティー」

セクション4.4.5「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

セクション4.5.2「mysqladmin — A MySQL Server 管理プログラム」

セクション4.6.8「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」

セクション4.5.3「mysqlcheck — テーブル保守プログラム」

セクション4.5.4「mysqldump — データベースバックアッププログラム」

セクション4.5.5「mysqlimport — データインポートプログラム」

セクション4.5.6「mysqlpump — データベースバックアッププログラム」

セクション4.5.7「mysqlshow — データベース、テーブル、およびカラム情報の表示」

セクション4.5.8「mysqlslap — ロードエミュレーションクライアント」

セクション23.4.32「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」

セクション4.2.2.2「オプションファイルの使用」

セクション4.2.2.3「オプションファイルの処理に影響するコマンド行オプション」

--loops

セクション23.4.14「ndb_index_stat — NDB インデックス統計ユーティリティー」

セクション23.4.27「ndb_show_tables — NDB テーブルのリストの表示」

セクション23.4.2「ndbinfo_select_all — ndbinfo テーブルからの選択」

--loose

セクション4.2.2.4「プログラムオプション修飾子」

--loose-opt_name

セクション4.2.2.2「オプションファイルの使用」

--lossy-conversions

セクション23.4.15「ndb_move_data — NDB データコピーユーティリティー」

セクション23.4.23「ndb_restore — NDB Cluster バックアップの復元」

--low-priority

セクション4.5.5「mysqlimport — データインポートプログラム」

--low-priority-updates

セクション13.2.6「INSERT ステートメント」

セクション8.11.2「テーブルロックの問題」

セクション8.11.3「同時挿入」

--lower-case-table-names

セクション9.2.3「識別子の太文字と小文字の区別」

M

[\[index top\]](#)

-M

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

-m

セクション4.6.4.2 「myisamchk のチェックオプション」

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

セクション23.6.9 「NDB Cluster レプリケーションによる NDB Cluster バックアップ」

セクション23.4.22 「ndb_redo_log_reader — クラスタ redo ログの内容の確認および印刷」

セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」

元のノードより多くのノードへのリストア

--malloc-lib

セクション4.3.2 「mysqld_safe — MySQL サーバー起動スクリプト」

セクション2.5.9 「systemd を使用した MySQL Server の管理」

--master-data

セクション5.4 「MySQL Server ログ」

セクション6.2.2 「MySQL で提供される権限」

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

mysqldump を使用したデータスナップショットの作成

セクション4.6.8.3 「バイナリログファイルのバックアップのための mysqlbinlog の使用」

セクション7.3.1 「バックアップポリシーの確立」

セクション17.1.3.5 「フェイルオーバーおよびスケールアウトでの GTID の使用」

--master-info-file

セクション1.3 「MySQL 8.0 の新機能」

セクション17.1.6.3 「Replica Server のオプションと変数」

セクション17.2.4.2 「レプリケーションメタデータリポジトリ」

--master-retry-count

セクション13.4.2.1 「CHANGE MASTER TO ステートメント」

セクション13.4.2.3 「CHANGE REPLICATION SOURCE TO ステートメント」

セクション17.1.6.3 「Replica Server のオプションと変数」

セクション13.7.7.35 「SHOW REPLICA | SLAVE STATUS ステートメント」

--max-allowed-packet

セクション4.5.1.1 「mysql クライアントオプション」

セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

--max-binlog-dump-events

セクション17.1.6.4 「バイナリロギングのオプションと変数」

--max-binlog-size

セクション17.1.6.3 「Replica Server のオプションと変数」

--max-join-size

セクション4.5.1.6 「mysql クライアントのヒント」

セクション4.5.1.1 「mysql クライアントオプション」

--max-record-length

セクション4.6.4.3 「myisamchk の修復オプション」

セクション13.7.3.5 「REPAIR TABLE ステートメント」

--max-relay-log-size

セクション17.1.6.3 「Replica Server のオプションと変数」

セクション17.2.2.3 「起動オプションとレプリケーションチャンネル」

--max-rows

セクション23.4.13 「ndb_import — NDB への CSV データのインポート」

--max-seeks-for-key

セクションB.3.5 「最適化に関連の問題」

セクション8.2.1.23 「全テーブルスキャンの回避」

--maximum

セクション4.2.2.4 「プログラムオプション修飾子」

--maximum-back_log

セクション4.2.2.4 「プログラムオプション修飾子」

--maximum-innodb-log-file-size

セクション5.1.9 「システム変数の使用」

--maximum-max_heap_table_size

セクション4.2.2.4 「プログラムオプション修飾子」

--maximum-var_name

セクション5.1.7 「サーバーコマンドオプション」

セクション5.1.9 「システム変数の使用」

--measured-load

セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」

--medium-check

セクション4.6.4.2 「myisamchk のチェックオプション」

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

--memlock

セクション5.1.7 「サーバーコマンドオプション」

セクション5.1.8 「サーバーシステム変数」

セクション15.6.3.1 「システムテーブルスペース」

--monitor

セクション23.4.13 「ndb_import — NDB への CSV データのインポート」

--mount

セクション2.5.6.2 「Docker での MySQL Server のデプロイに関するその他のトピック」

--my-plugin

セクション5.6.1 「プラグインのインストールおよびアンインストール」

--my_plugin

セクション5.6.1「プラグインのインストールおよびアンインストール」

--mycnf

セクション23.4.7「ndb_config — NDB Cluster 構成情報の抽出」

セクション23.4.4「ndb_mgmd — NDB Cluster 管理サーバーデーモン」

--myisam-block-size

セクション8.10.2.5「キーキャッシュブロックサイズ」

セクション5.1.7「サーバーコマンドオプション」

--myisam_sort_buffer_size

セクション4.6.4.6「myisamchk メモリ使用量」

MYSQL_ALLOW_EMPTY_PASSWORD

セクション2.5.6.2「Docker での MySQL Server のデプロイに関するその他のトピック」

MYSQL_DATABASE

セクション2.5.6.2「Docker での MySQL Server のデプロイに関するその他のトピック」

MYSQL_LOG_CONSOLE

セクション2.5.6.2「Docker での MySQL Server のデプロイに関するその他のトピック」

MYSQL_MAINTAINER_MODE

セクション2.9.8「MySQL のコンパイルに関する問題」

MYSQL_ONETIME_PASSWORD

セクション2.5.6.2「Docker での MySQL Server のデプロイに関するその他のトピック」

セクション2.5.6.1「Docker を使用した MySQL Server デプロイメントの基本ステップ」

MYSQL_PASSWORD

セクション2.5.6.2「Docker での MySQL Server のデプロイに関するその他のトピック」

MYSQL_RANDOM_ROOT_PASSWORD

セクション2.5.6.2「Docker での MySQL Server のデプロイに関するその他のトピック」

MYSQL_ROOT_HOST

セクション2.5.6.2「Docker での MySQL Server のデプロイに関するその他のトピック」

MYSQL_ROOT_PASSWORD

セクション2.5.6.2「Docker での MySQL Server のデプロイに関するその他のトピック」

MYSQL_TCP_PORT

セクション2.9.7「MySQL ソース構成オプション」

セクション2.9.5「開発ソースツリーを使用して MySQL をインストールする」

MYSQL_UNIX_ADDR

セクションB.3.3.6「MySQL の UNIX ソケットファイルを保護または変更する方法」

セクション2.9.7「MySQL ソース構成オプション」

セクション20.5.6.2「X プラグイン のオプションとシステム変数」

セクション2.9.5「開発ソースツリーを使用して MySQL をインストールする」

MYSQL_USER

セクション2.5.6.2「Docker での MySQL Server のデプロイに関するその他のトピック」

--mysqladmin

セクション4.3.4 「mysqld_multi — 複数の MySQL サーバーの管理」

--mysqld

セクション4.3.4 「mysqld_multi — 複数の MySQL サーバーの管理」
セクション4.3.2 「mysqld_safe — MySQL サーバー起動スクリプト」

--mysqld-safe-log-timestamps

セクション4.3.2 「mysqld_safe — MySQL サーバー起動スクリプト」

--mysqld-version

セクション4.3.2 「mysqld_safe — MySQL サーバー起動スクリプト」

--mysqlx

セクション20.5.6.2 「X プラグイン のオプションとシステム変数」
セクション20.5.2 「X プラグイン の無効化」

mysqlx

セクション20.5.2 「X プラグイン の無効化」

MYSQLX_UNIX_ADDR

セクション20.5.6.2 「X プラグイン のオプションとシステム変数」

N

[\[index top\]](#)

-N

セクション4.4.1 「comp_err — MySQL エラーメッセージファイルのコンパイル」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション23.4.26 「ndb_setup.py — NDB Cluster 用ブラウザベースの Auto-Installer の起動 (非推奨)」

-n

セクション4.6.1 「ibd2sdi — InnoDB テーブルスペース SDI 抽出ユーティリティ」
セクション4.6.2 「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティ」
セクション4.7.2 「my_print_defaults — オプションファイルからのオプションの表示」
セクション4.6.4.3 「myisamchk の修復オプション」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.6.9 「mysqldumpslow — スロークエリログファイルの要約」
セクション23.5.1 「NDB Cluster 管理クライアントのコマンド」
セクション23.4.9 「ndb_desc — NDB テーブルの表示」
セクション23.4.22 「ndb_redo_log_reader — クラスター redo ログの内容の確認および印刷」
セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」
セクション23.4.26 「ndb_setup.py — NDB Cluster 用ブラウザベースの Auto-Installer の起動 (非推奨)」
セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」
セクション23.4.30 「ndb_waiter — NDB Cluster が特定のステータスに達するまで待機」
セクション23.4.1 「ndbd — NDB Cluster データノードデーモン」

--name

セクション2.5.6.1 「Docker を使用した MySQL Server デプロイメントの基本ステップ」

--name-file

セクション4.4.1 「comp_err — MySQL エラーメッセージファイルのコンパイル」

--named-commands

セクション4.5.1.1「mysql クライアントオプション」

--ndb

セクション1.3「MySQL 8.0 の新機能」

セクション23.1.4「NDB Cluster の新機能」

セクション23.4.16「ndb_perror — NDB エラーメッセージ情報の取得」

セクション23.5.14.30「ndbinfo error_messages テーブル」

セクション4.8.2「 perror — MySQL エラーメッセージ情報の表示」

--ndb-allow-copying-alter-table

NDB Cluster の MySQL Server オプション

--ndb-batch-size

NDB Cluster の MySQL Server オプション

セクション23.6.5「NDB Cluster のレプリケーションの準備」

セクション23.1.4「NDB Cluster の新機能」

--ndb-blob-read-batch-bytes

NDB Cluster の MySQL Server オプション

--ndb-blob-write-batch-bytes

NDB Cluster の MySQL Server オプション

セクション23.6.5「NDB Cluster のレプリケーションの準備」

セクション23.1.4「NDB Cluster の新機能」

--ndb-cluster

セクションA.10「MySQL 8.0 FAQ: NDB Cluster」

--ndb-cluster-connection-pool

NDB Cluster の MySQL Server オプション

--ndb-cluster-connection-pool-nodeids

NDB Cluster の MySQL Server オプション

--ndb-connectstring

セクションA.10「MySQL 8.0 FAQ: NDB Cluster」

セクション23.5.17.2「NDB Cluster および MySQL の権限」

セクション23.5.9「NDB Cluster での MySQL Server の使用」

NDB Cluster の MySQL Server オプション

セクション23.1.1「NDB Cluster のコア概念」

セクション23.6.5「NDB Cluster のレプリケーションの準備」

セクション23.3.3.2「NDB Cluster の推奨開始構成」

セクション23.1.4「NDB Cluster の新機能」

セクション23.4.32「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」

セクション23.4.7「ndb_config — NDB Cluster 構成情報の抽出」

セクション23.4.23「ndb_restore — NDB Cluster バックアップの復元」

セクション23.2.2.1「バイナリリリースから Windows への NDB Cluster のインストール」

元のノードより多くのノードへのリストア

元のノードより少ないノードへのリストア

--ndb-default-column-format

NDB Cluster の MySQL Server オプション

--ndb-deferred-constraints

NDB Cluster の MySQL Server オプション

--ndb-distribution

NDB Cluster の MySQL Server オプション

--ndb-force-send

セクション23.3.3.2 「NDB Cluster の推奨開始構成」

--ndb-index-stat-enable

セクション23.3.3.2 「NDB Cluster の推奨開始構成」

--ndb-log-apply-status

NDB Cluster の MySQL Server オプション
NDB Cluster システム変数

--ndb-log-bin

セクション23.6.6 「NDB Cluster レプリケーションの開始 (シングルレプリケーションチャンネル)」

--ndb-log-empty-epochs

NDB Cluster の MySQL Server オプション
セクション23.6.4 「NDB Cluster レプリケーションスキーマおよびテーブル」

--ndb-log-empty-update

NDB Cluster の MySQL Server オプション

--ndb-log-exclusive-reads

NDB Cluster の MySQL Server オプション

--ndb-log-fail-terminate

NDB Cluster の MySQL Server オプション
セクション23.1.4 「NDB Cluster の新機能」

--ndb-log-orig

NDB Cluster の MySQL Server オプション
NDB Cluster システム変数
セクション23.6.4 「NDB Cluster レプリケーションスキーマおよびテーブル」

--ndb-log-transaction-id

NDB Cluster の MySQL Server オプション
NDB Cluster システム変数
セクション23.6.11 「NDB Cluster レプリケーションの競合解決」

--ndb-log-update-as-write

セクション23.6.3 「NDB Cluster レプリケーションの既知の問題」
セクション23.6.11 「NDB Cluster レプリケーションの競合解決」

--ndb-log-update-minimal

NDB Cluster の MySQL Server オプション

--ndb-log-updated-only

セクション23.6.11 「NDB Cluster レプリケーションの競合解決」

--ndb-mgmd-host

NDB Cluster の MySQL Server オプション

セクション23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」

--ndb-nodegroup-map

セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

--ndb-nodeid

NDB Cluster の MySQL Server オプション

セクション23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」

セクション23.4.4 「ndb_mgmd — NDB Cluster 管理サーバーデーモン」

セクション23.4.1 「ndbd — NDB Cluster データノードデーモン」

--ndb-optimization-delay

NDB Cluster の MySQL Server オプション

セクション13.7.3.4 「OPTIMIZE TABLE ステートメント」

--ndb-optimized-node-selection

セクション23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」

--ndb-schema-dist-timeout

NDB Cluster の MySQL Server オプション

セクション23.1.4 「NDB Cluster の新機能」

--ndb-transid-mysql-connection-map

セクション26.17 「INFORMATION_SCHEMA ndb_transid_mysql_connection_map テーブル」

NDB Cluster の MySQL Server オプション

--ndb-use-exact-count

セクション23.3.3.2 「NDB Cluster の推奨開始構成」

--ndb-wait-connected

NDB Cluster の MySQL Server オプション

--ndb-wait-setup

NDB Cluster の MySQL Server オプション

--ndbcluster

セクション26.13 「INFORMATION_SCHEMA ENGINES テーブル」

セクションA.10 「MySQL 8.0 FAQ: NDB Cluster」

セクション23.5.17.2 「NDB Cluster および MySQL の権限」

セクション23.5.9 「NDB Cluster での MySQL Server の使用」

NDB Cluster の MySQL Server オプション

セクション23.1.1 「NDB Cluster のコア概念」

セクション23.3.3.2 「NDB Cluster の推奨開始構成」

セクション23.3 「NDB Cluster の構成」

セクション23.5.14 「ndbinfo: NDB Cluster 情報データベース」

セクション13.7.7.16 「SHOW ENGINES ステートメント」

セクション5.1.7 「サーバーコマンドオプション」

セクション23.2.2.1 「バイナリリリースから Windows への NDB Cluster のインストール」

--ndbinfo

NDB Cluster の MySQL Server オプション

--net-buffer-length

セクション4.5.1.1 「mysql クライアントオプション」

セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

net_retry_count

セクション17.2.3.1 「レプリケーションメインスレッドの監視」

net_write_timeout

セクション17.2.3.1 「レプリケーションメインスレッドの監視」

--network

セクション2.5.6.2 「Docker での MySQL Server のデプロイに関するその他のトピック」

--network-namespace

セクション4.5.1.1 「mysql クライアントオプション」

セクション5.1.14 「ネットワークネームスペースのサポート」

--network-timeout

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

--nice

セクション4.3.2 「mysqld_safe — MySQL サーバー起動スクリプト」

セクション2.5.9 「systemd を使用した MySQL Server の管理」

--no-asynch

セクション23.4.13 「ndb_import — NDB への CSV データのインポート」

--no-auto-rehash

セクション4.5.1.1 「mysql クライアントオプション」

--no-autocommit

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

--no-beep

セクション4.5.1.1 「mysql クライアントオプション」

セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」

--no-binlog

セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

--no-browser

セクション23.4.26 「ndb_setup.py — NDB Cluster 用ブラウザベースの Auto-Installer の起動 (非推奨)」

--no-check

セクション4.6.1 「ibd2sdi — InnoDB テーブルスペース SDI 抽出ユーティリティ」

セクション4.6.2 「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティー」

--no-contact

セクション23.4.30 「ndb_waiter — NDB Cluster が特定のステータスに達するまで待機」

--no-create-db

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

--no-create-info

セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション7.4.5.4 「テーブル定義と内容の個別のダンプ」

--no-data

セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション7.4.5.4 「テーブル定義と内容の個別のダンプ」

--no-dd-upgrade

セクション1.3 「MySQL 8.0 の新機能」
セクション2.11.3 「MySQL のアップグレードプロセスの内容」
セクション5.1.7 「サーバーコマンドオプション」
セクション14.1 「データディクショナリスキーマ」

--no-defaults

セクション4.7.2 「my_print_defaults — オプションファイルからのオプションの表示」
セクション4.6.4.1 「myisamchk の一般オプション」
セクション6.2.21 「MySQL への接続の問題のトラブルシューティング」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティー」
セクション4.4.2 「mysql_secure_installation — MySQL インストールのセキュリティ改善」
セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.3.4 「mysqld_multi — 複数の MySQL サーバーの管理」
セクション4.3.2 「mysqld_safe — MySQL サーバー起動スクリプト」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」
セクション23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」
セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」
セクション4.2.2.2 「オプションファイルの使用」
セクション4.2.2.3 「オプションファイルの処理に影響するコマンド行オプション」
セクション5.1.7 「サーバーコマンドオプション」
セクション5.1.9.3 「永続化されるシステム変数」

--no-drop

セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」

--no-hint

セクション23.4.13 「ndb_import — NDB への CSV データのインポート」

--no-history-logging

セクション18.4.6 「グループレプリケーションでの MySQL Enterprise Backup の使用」

--no-log

セクション4.3.4 「mysqld_multi — 複数の MySQL サーバーの管理」

--no-monitor

セクション2.3 「Microsoft Windows に MySQL をインストールする」
セクション13.7.8.8 「RESTART ステートメント」
セクション5.1.7 「サーバーコマンドオプション」

--no-nodeid-checks

セクション23.4.4 「[ndb_mgmd — NDB Cluster 管理サーバーデーモン](#)」

--no-restore-disk-objects

セクション23.4.23 「[ndb_restore — NDB Cluster バックアップの復元](#)」

--no-set-names

セクション4.5.4 「[mysqldump — データベースバックアッププログラム](#)」

--no-symlinks

セクション4.6.4.3 「[myisamchk の修復オプション](#)」

--no-tablespaces

セクション4.5.4 「[mysqldump — データベースバックアッププログラム](#)」

--no-upgrade

セクション23.4.23 「[ndb_restore — NDB Cluster バックアップの復元](#)」

--nodaemon

セクション23.4.4 「[ndb_mgmd — NDB Cluster 管理サーバーデーモン](#)」

セクション23.4.1 「[ndbd — NDB Cluster データノードデーモン](#)」

--nodata

セクション23.4.24 「[ndb_select_all — NDB テーブルの行の出力](#)」

--node-id

セクション23.4.29 「[ndb_top — NDB スレッドの CPU 使用率情報の表示](#)」

--nodeid

セクション23.1.4 「[NDB Cluster の新機能](#)」

セクション23.4.7 「[ndb_config — NDB Cluster 構成情報の抽出](#)」

セクション23.4.23 「[ndb_restore — NDB Cluster バックアップの復元](#)」

元のノードより少ないノードへのリストア

--nodes

セクション23.4.7 「[ndb_config — NDB Cluster 構成情報の抽出](#)」

--nostart

セクション23.5.1 「[NDB Cluster 管理クライアントのコマンド](#)」

セクション23.4.1 「[ndbd — NDB Cluster データノードデーモン](#)」

--not-started

セクション23.4.30 「[ndb_waiter — NDB Cluster が特定のステータスに達するまで待機](#)」

--nowait-nodes

セクション23.5.7.3 「[NDB Cluster データノードのオンラインでの追加: 詳細な例](#)」

セクション23.3.3.6 「[NDB Cluster データノードの定義](#)」

セクション23.4.4 「[ndb_mgmd — NDB Cluster 管理サーバーデーモン](#)」

セクション23.4.30 「[ndb_waiter — NDB Cluster が特定のステータスに達するまで待機](#)」

セクション23.4.1 「[ndbd — NDB Cluster データノードデーモン](#)」

--num-slices

セクション23.1.4 「[NDB Cluster の新機能](#)」

[セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」](#)

--number-char-cols

[セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」](#)

--number-int-cols

[セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」](#)

--number-of-queries

[セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」](#)

O

[\[index top\]](#)

-O

[セクション4.4.1 「comp_err — MySQL エラーメッセージファイルのコンパイル」](#)

[セクション2.9.7 「MySQL ソース構成オプション」](#)

-o

[セクション4.6.4.3 「myisamchk の修復オプション」](#)

[セクション4.6.5 「myisamlog — MyISAM ログファイルの内容の表示」](#)

[セクション4.5.1.1 「mysql クライアントオプション」](#)

[セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」](#)

[セクション4.5.3 「mysqlcheck — テーブル保守プログラム」](#)

[セクション23.4.24 「ndb_select_all — NDB テーブルの行の出力」](#)

[セクション23.4.26 「ndb_setup.py — NDB Cluster 用ブラウザベースの Auto-Installer の起動 \(非推奨\)」](#)

[セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」](#)

[セクション8.12.1 「ディスク I/O の最適化」](#)

--offset

[セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」](#)

--old-style-user-limits

[セクション6.2.20 「アカウントリソース制限の設定」](#)

[セクション5.1.7 「サーバーコマンドオプション」](#)

old_passwords

[セクション2.11.4 「MySQL 8.0 での変更」](#)

--oldpackage

[セクション2.5.4 「Oracle の RPM パッケージを使用した Linux への MySQL のインストール」](#)

ON

[セクション3.3.4.9 「複数のテーブルの使用」](#)

--one-database

[セクション4.5.1.1 「mysql クライアントオプション」](#)

--only-print

[セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」](#)

--opbatch

[セクション23.4.13 「ndb_import — NDB への CSV データのインポート」](#)

--opbytes

セクション23.4.13 「[ndb_import](#) — NDB への CSV データのインポート」

--open-files-limit

セクション4.6.8 「[mysqlbinlog](#) — バイナリログファイルを処理するためのユーティリティー」

セクション4.3.2 「[mysqld_safe](#) — MySQL サーバー起動スクリプト」

セクション2.5.9 「[systemd](#) を使用した MySQL Server の管理」

セクション5.1.8 「サーバーシステム変数」

セクションB.3.2.16 「ファイルが見つからず同様のエラーが発生しました」

--opt

セクション8.5.5 「[InnoDB](#) テーブルの一括データロード」

セクション4.5.4 「[mysqldump](#) — データベースバックアッププログラム」

--opt_name

セクション4.2.2.2 「オプションファイルの使用」

--optimize

セクション4.5.3 「[mysqlcheck](#) — テーブル保守プログラム」

--optimizer-switch

セクション23.3.3.2 「[NDB Cluster](#) の推奨開始構成」

options

セクション12.17.4 「[WKB](#) 値からジオメトリ値を作成する関数」

セクション12.17.3 「[WKT](#) 値からジオメトリ値を作成する関数」

セクション12.17.6 「[ジオメトリ形式変換関数](#)」

--order

セクション23.4.24 「[ndb_select_all](#) — NDB テーブルの行の出力」

--order-by-primary

セクション4.5.4 「[mysqldump](#) — データベースバックアッププログラム」

--os-load

セクション23.4.29 「[ndb_top](#) — NDB スレッドの CPU 使用率情報の表示」

--out-dir

セクション4.4.1 「[comp_err](#) — MySQL エラーメッセージファイルのコンパイル」

--out-file

セクション4.4.1 「[comp_err](#) — MySQL エラーメッセージファイルのコンパイル」

--output-type

セクション23.4.13 「[ndb_import](#) — NDB への CSV データのインポート」

--output-workers

セクション23.4.13 「[ndb_import](#) — NDB への CSV データのインポート」

P

[\[index top\]](#)

-P

- セクション4.5.1.1 「mysql クライアントオプション」
- セクション4.2.1 「MySQL プログラムの起動」
- セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティー」
- セクション4.4.2 「mysql_secure_installation — MySQL インストールのセキュリティ改善」
- セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
- セクション4.5.2 「mysqldadmin — A MySQL Server 管理プログラム」
- セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」
- セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
- セクション4.5.4 「mysqldump — データベースバックアッププログラム」
- セクション4.5.5 「mysqlimport — データインポートプログラム」
- セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
- セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
- セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」
- セクション23.1.4 「NDB Cluster の新機能」
- セクション23.5.8.2 「NDB Cluster 管理クライアントを使用したバックアップの作成」
- セクション23.4.4 「ndb_mgmd — NDB Cluster 管理サーバーデーモン」
- セクション23.4.17 「ndb_print_backup_file — NDB バックアップファイルの内容の出力」
- セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」
- セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」
- セクション4.2.4 「コマンドオプションを使用した MySQL Server への接続」
- セクション4.2.3 「サーバーに接続するためのコマンドオプション」
- セクション5.1.7 「サーバーコマンドオプション」

-p

- セクション4.6.1 「ibd2sdi — InnoDB テーブルスペース SDI 抽出ユーティリティー」
- セクション4.6.2 「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティー」
- セクション24.2.5 「KEY パーティショニング」
- セクション4.6.4.3 「myisamchk の修復オプション」
- セクション4.6.5 「myisamlog — MyISAM ログファイルの内容の表示」
- セクション2.11 「MySQL のアップグレード」
- セクション6.2.21 「MySQL への接続の問題のトラブルシューティング」
- セクション2.3.4.9 「MySQL インストールのテスト」
- セクション4.5.1.1 「mysql クライアントオプション」
- セクション4.2.1 「MySQL プログラムの起動」
- セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティー」
- セクション4.4.2 「mysql_secure_installation — MySQL インストールのセキュリティ改善」
- セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
- セクション4.5.2 「mysqldadmin — A MySQL Server 管理プログラム」
- セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」
- セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
- セクション4.5.4 「mysqldump — データベースバックアッププログラム」
- セクション4.5.5 「mysqlimport — データインポートプログラム」
- セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
- セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
- セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」
- セクション23.5.7.3 「NDB Cluster データノードのオンラインでの追加: 詳細な例」
- セクション23.4.9 「ndb_desc — NDB テーブルの表示」
- セクション23.4.22 「ndb_redo_log_reader — クラスタ redo ログの内容の確認および印刷」
- セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」
- セクション23.4.24 「ndb_select_all — NDB テーブルの行の出力」
- セクション23.4.25 「ndb_select_count — NDB テーブルの行数の出力」
- セクション23.4.26 「ndb_setup.py — NDB Cluster 用ブラウザベースの Auto-Installer の起動 (非推奨)」
- セクション23.4.27 「ndb_show_tables — NDB テーブルのリストの表示」
- セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」
- セクション23.5.14.6 「ndbinfo cluster_operations テーブル」
- セクション23.5.14.45 「ndbinfo server_operations テーブル」
- セクション23.4.2 「ndbinfo_select_all — ndbinfo テーブルからの選択」

セクション2.3.6 「Windows でのインストール後の手順」
セクション2.3.4.6 「Windows のコマンド行からの MySQL の起動」
セクション2.3.4.8 「Windows のサービスとして MySQL を起動する」
セクション2.11.10 「Windows 上の MySQL をアップグレードする」
セクション6.2.1 「アカウントのユーザー名とパスワード」
セクション4.2.4 「コマンドオプションを使用した MySQL Server への接続」
セクション4.2.2.1 「コマンド行でのオプションの使用」
セクション4.2.3 「サーバーに接続するためのコマンドオプション」
セクション2.10.3 「サーバーのテスト」
セクションB.3.2.4 「パスワードをインタラクティブに入力すると失敗する」
セクション6.1.2.1 「パスワードセキュリティのためのエンドユーザーガイドライン」

--page

セクション4.6.2 「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティ」

--page-type-dump

セクション4.6.2 「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティ」

--page-type-summary

セクション4.6.2 「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティ」

--pagecnt

セクション23.4.13 「ndb_import — NDB への CSV データのインポート」

--pager

セクション4.5.1.1 「mysql クライアントオプション」

セクション4.5.1.2 「mysql クライアントコマンド」

--pagesize

セクション23.4.13 「ndb_import — NDB への CSV データのインポート」

--parallel-recover

セクション4.6.4.3 「myisamchk の修復オプション」

--parallel-schemas

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

--parallelism

セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

セクション23.4.24 「ndb_select_all — NDB テーブルの行の出力」

セクション23.4.25 「ndb_select_count — NDB テーブルの行数の出力」

セクション23.4.2 「ndbinfo_select_all — ndbinfo テーブルからの選択」

parallelism

セクション23.4.24 「ndb_select_all — NDB テーブルの行の出力」

--parsable

セクション23.4.27 「ndb_show_tables — NDB テーブルのリストの表示」

--partition

セクション1.3 「MySQL 8.0 の新機能」

--passwd

セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」

--password

セクション6.2.21 「MySQL への接続の問題のトラブルシューティング」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.2.1 「MySQL プログラムの起動」
セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティ」
セクション4.4.2 「mysql_secure_installation — MySQL インストールのセキュリティ改善」
セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」
セクション4.6.8 「mysqlbinlog — バイナリログファイル処理のためのユーティリティ」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.3.4 「mysqld_multi — 複数の MySQL サーバーの管理」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」
セクション23.4.28 「ndb_size.pl — NDBCLUSTER サイズ要件エスティメータ」
セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」
セクション6.2.1 「アカウントのユーザー名とパスワード」
セクション4.2.4 「コマンドオプションを使用した MySQL Server への接続」
セクション4.2.2.1 「コマンド行でのオプションの使用」
セクション4.2.3 「サーバーに接続するためのコマンドオプション」
セクション4.6.8.3 「バイナリログファイルのバックアップのための mysqlbinlog の使用」
セクション7.3 「バックアップおよびリカバリ戦略の例」
セクションB.3.2.4 「パスワードをインタラクティブに入力すると失敗する」
セクション6.1.2.1 「パスワードセキュリティのためのエンドユーザーガイドライン」
セクション6.4.1.10 「プラグブル認証のテスト」
元のノードより多くのノードへのリストア

password

セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティ」
セクション4.2.2.2 「オプションファイルの使用」

--performance-schema-consumer-consumer_name

セクション27.14 「パフォーマンススキーマコマンドオプション」

--performance-schema-consumer-events-stages-current

セクション27.14 「パフォーマンススキーマコマンドオプション」

--performance-schema-consumer-events-stages-history

セクション27.14 「パフォーマンススキーマコマンドオプション」

--performance-schema-consumer-events-stages-history-long

セクション27.14 「パフォーマンススキーマコマンドオプション」

--performance-schema-consumer-events-statements-current

セクション27.14 「パフォーマンススキーマコマンドオプション」

--performance-schema-consumer-events-statements-history

セクション27.14 「パフォーマンススキーマコマンドオプション」

--performance-schema-consumer-events-statements-history-long

セクション27.14 「パフォーマンススキーマコマンドオプション」

--performance-schema-consumer-events-transactions-current

セクション27.14 「パフォーマンススキーマコマンドオプション」

--performance-schema-consumer-events-transactions-history

セクション27.14「パフォーマンススキーマコマンドオプション」

--performance-schema-consumer-events-transactions-history-long

セクション27.14「パフォーマンススキーマコマンドオプション」

--performance-schema-consumer-events-waits-current

セクション27.14「パフォーマンススキーマコマンドオプション」

--performance-schema-consumer-events-waits-history

セクション27.14「パフォーマンススキーマコマンドオプション」

--performance-schema-consumer-events-waits-history-long

セクション27.14「パフォーマンススキーマコマンドオプション」

--performance-schema-consumer-global-instrumentation

セクション27.14「パフォーマンススキーマコマンドオプション」

--performance-schema-consumer-statements-digest

セクション27.14「パフォーマンススキーマコマンドオプション」

--performance-schema-consumer-thread-instrumentation

セクション27.14「パフォーマンススキーマコマンドオプション」

--performance-schema-instrument

セクション27.14「パフォーマンススキーマコマンドオプション」

セクション27.3「パフォーマンススキーマ起動構成」

--performance-schema-xxx

セクション5.1.7「サーバーコマンドオプション」

--performance_schema_max_mutex_classes

セクション27.7「パフォーマンススキーマステータスマニタリング」

--performance_schema_max_mutex_instances

セクション27.7「パフォーマンススキーマステータスマニタリング」

--pid-file

セクション5.8「1つのマシン上での複数のMySQLインスタンスの実行」

セクション4.3.4「mysqld_multi — 複数のMySQLサーバーの管理」

セクション4.3.2「mysqld_safe — MySQLサーバー起動スクリプト」

セクション2.5.9「systemdを使用したMySQL Serverの管理」

セクション5.4.2.1「エラーログ構成」

セクション5.1.7「サーバーコマンドオプション」

セクション5.4.2.2「デフォルトのエラーログ保存先の構成」

pid-file

セクション4.3.3「mysql.server — MySQLサーバー起動スクリプト」

--pipe

セクション2.3.4.9「MySQLインストールのテスト」

セクション4.5.1.1「mysqlクライアントオプション」

セクション4.4.5「mysql_upgrade — MySQLテーブルのチェックとアップグレード」

セクション4.5.2「mysqladmin — A MySQL Server 管理プログラム」

- セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
- セクション4.5.4 「mysqldump — データベースバックアッププログラム」
- セクション4.5.5 「mysqlimport — データインポートプログラム」
- セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
- セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」
- セクション4.2.4 「コマンドオプションを使用した MySQL Server への接続」
- セクション4.2.3 「サーバーに接続するためのコマンドオプション」

--plugin

- セクション5.1.7 「サーバーコマンドオプション」

--plugin-dir

- セクション6.4.1.7 「LDAP プラガブル認証」
- セクション4.5.1.1 「mysql クライアントオプション」
- セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
- セクション4.5.2 「mysqldadmin — A MySQL Server 管理プログラム」
- セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」
- セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
- セクション4.3.2 「mysqld_safe — MySQL サーバー起動スクリプト」
- セクション4.5.4 「mysqldump — データベースバックアッププログラム」
- セクション4.5.5 「mysqlimport — データインポートプログラム」
- セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
- セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
- セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」
- セクション4.2.3 「サーバーに接続するためのコマンドオプション」
- セクション6.2.17 「プラグブル認証」

--plugin-innodb-file-per-table

- セクション5.1.7 「サーバーコマンドオプション」

--plugin-load

- セクション5.6.5.2 「ddl_rewriter プラグインオプション」
- セクション13.7.4.4 「INSTALL PLUGIN ステートメント」
- セクション1.3 「MySQL 8.0 の新機能」
- セクション2.9.7 「MySQL ソース構成オプション」
- セクション6.4.4.12 「キーリングコマンドのオプション」
- セクション6.4.4.1 「キーリングプラグインのインストール」
- セクション5.1.7 「サーバーコマンドオプション」
- セクション6.4.3.3 「パスワード検証コンポーネントへの移行」
- セクション5.6.1 「プラグインのインストールおよびアンインストール」
- セクション6.4.5.10 「監査ログ参照」

--plugin-load-add

- セクション6.4.2.1 「Connection-Control プラグインのインストール」
- セクション5.6.5.2 「ddl_rewriter プラグインオプション」
- セクション6.4.1.7 「LDAP プラガブル認証」
- セクション1.3 「MySQL 8.0 の新機能」
- セクション6.4.1.5 「PAM プラガブル認証」
- セクション6.4.1.6 「Windows プラガブル認証」
- セクション6.4.4.1 「キーリングプラグインのインストール」
- セクション5.6.7.1 「クローンプラグインのインストール」
- セクション5.1.7 「サーバーコマンドオプション」
- セクション5.6.3.2 「スレッドプールのインストール」
- セクション6.4.1.9 「ソケットピア資格証明プラグブル認証」
- セクション6.4.3.2 「パスワード検証オプションおよび変数」
- セクション6.4.3.3 「パスワード検証コンポーネントへの移行」
- セクション6.4.1.10 「プラグブル認証のテスト」
- セクション5.6.1 「プラグインのインストールおよびアンインストール」

セクション6.4.1.8 「ログインなしのプラグブル認証」

セクション6.4.5.10 「監査ログ参照」

plugin-load-add

セクション5.6.7.1 「クローンプラグインのインストール」

セクション18.2.1.2 「グループレプリケーション用のインスタンスの構成」

--plugin-sql-mode

セクション5.1.7 「サーバーコマンドオプション」

--plugin-xxx

セクション5.1.7 「サーバーコマンドオプション」

--plugin_dir

セクション2.9.7 「MySQL ソース構成オプション」

--plugin_name

セクション5.6.1 「プラグインのインストールおよびアンインストール」

--plugindir

セクション4.7.1 「mysql_config — クライアントのコンパイル用オプションの表示」

--polltimeout

セクション23.4.13 「ndb_import — NDB への CSV データのインポート」

--port

セクション5.8 「1 つのマシン上での複数の MySQL インスタンスの実行」

セクション2.10.2.1 「MySQL Server の起動時の問題のトラブルシューティング」

セクション6.2.21 「MySQL への接続の問題のトラブルシューティング」

セクション4.5.1.1 「mysql クライアントオプション」

セクション2.9.7 「MySQL ソース構成オプション」

セクション4.2.1 「MySQL プログラムの起動」

セクション4.7.1 「mysql_config — クライアントのコンパイル用オプションの表示」

セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティー」

セクション4.4.2 「mysql_secure_installation — MySQL インストールのセキュリティ改善」

セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」

セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

セクション4.3.2 「mysqld_safe — MySQL サーバー起動スクリプト」

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.5 「mysqlimport — データインポートプログラム」

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」

セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」

セクション23.4.26 「ndb_setup.py — NDB Cluster 用ブラウザベースの Auto-Installer の起動 (非推奨)」

セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」

セクション5.8.3 「Unix 上での複数の MySQL インスタンスの実行」

セクション4.2.4 「コマンドオプションを使用した MySQL Server への接続」

セクション4.2.3 「サーバーに接続するためのコマンドオプション」

セクション5.1.7 「サーバーコマンドオプション」

セクション5.1.8 「サーバーシステム変数」

セクション5.8.4 「複数サーバー環境でのクライアントプログラムの使用」

port

セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティー」

セクション4.2.2.2 「オプションファイルの使用」

--port-open-timeout

セクション5.1.7 「サーバーコマンドオプション」

--post-query

セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」

--post-system

セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」

--pre-query

セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」

--pre-system

セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」

PREFIX

セクション23.2.1.4 「Linux でのソースからの NDB Cluster の構築」

--preserve-trailing-spaces

セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

--pretty

セクション4.6.1 「ibd2sdi — InnoDB テーブルスペース SDI 抽出ユーティリティ」

--print

セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

--print-data

セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

--print-defaults

セクション4.6.4.1 「myisamchk の一般オプション」

セクション4.5.1.1 「mysql クライアントオプション」

セクション4.4.2 「mysql_secure_installation — MySQL インストールのセキュリティ改善」

セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

セクション4.5.2 「mysqldadmin — A MySQL Server 管理プログラム」

セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.5 「mysqlimport — データインポートプログラム」

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」

セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」

セクション23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」

セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」

セクション2.11.12 「アップグレードのトラブルシューティング」

セクション4.2.2.3 「オプションファイルの処理に影響するコマンド行オプション」

セクション5.1.7 「サーバーコマンドオプション」

--print-full-config

セクション23.4.4 「ndb_mgmd — NDB Cluster 管理サーバーデーモン」

--print-log

セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

--print-meta

セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

--print-sql-log

セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

print-sql-log

セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

--print-table-metadata

セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」

--print_*

セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

--progress-frequency

セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

--promote-attributes

セクション23.4.15 「ndb_move_data — NDB データコピーユーティリティ」

セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

--prompt

セクション4.5.1.1 「mysql クライアントオプション」

セクション4.5.1.2 「mysql クライアントコマンド」

--protocol

セクション1.2.2 「MySQL の主な機能」

セクション2.3.4.9 「MySQL インストールのテスト」

セクション4.5.1.1 「mysql クライアントオプション」

セクション4.4.2 「mysql_secure_installation — MySQL インストールのセキュリティ改善」

セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」

セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.5 「mysqlimport — データインポートプログラム」

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」

セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」

セクション5.8.3 「Unix 上での複数の MySQL インスタンスの実行」

セクション4.2.4 「コマンドオプションを使用した MySQL Server への接続」

セクション4.2.3 「サーバーに接続するためのコマンドオプション」

セクション2.3.4.5 「サーバーをはじめて起動する」

セクション4.2.7 「接続トランスポートプロトコル」

セクション5.8.4 「複数サーバー環境でのクライアントプログラムの使用」

Q

[index top]

-Q

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

-q

セクション4.6.4.3 「myisamchk の修復オプション」

セクション4.5.1.1 「mysql クライアントオプション」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」
セクション23.4.7 「ndb_config — NDB Cluster 構成情報の抽出」
セクション23.4.18 「ndb_print_file — NDB データファイル内容の出力」

--query

セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」
セクション23.4.7 「ndb_config — NDB Cluster 構成情報の抽出」
セクション23.4.14 「ndb_index_stat — NDB インデックス統計ユーティリティー」

--query-all

セクション23.4.7 「ndb_config — NDB Cluster 構成情報の抽出」

--quick

セクション4.6.4.3 「myisamchk の修復オプション」
セクション4.6.4.6 「myisamchk メモリ使用量」
セクション4.5.1 「mysql — MySQL コマンドラインクライアント」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.2.2.2 「オプションファイルの使用」
セクション7.6.1 「クラッシュリカバリへの myisamchk の使用」
セクションB.3.2.6 「メモリ不足」

--quote-names

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

R

[\[index top\]](#)

-R

セクション7.6.4 「MyISAM テーブルの最適化」
セクション4.6.5 「myisamlog — MyISAM ログファイルの内容の表示」
セクション6.2.2 「MySQL で提供される権限」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.6.4.4 「その他の myisamchk オプション」
セクション4.6.8.3 「バイナリログファイルのバックアップのための mysqlbinlog の使用」

-r

セクション7.6.3 「MyISAM テーブルの修復方法」
セクション4.6.4.2 「myisamchk のチェックオプション」
セクション4.6.4.3 「myisamchk の修復オプション」
セクション4.6.5 「myisamlog — MyISAM ログファイルの内容の表示」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.6.9 「mysqldumpslow — スロークエリーログファイルの要約」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション23.4.7 「ndb_config — NDB Cluster 構成情報の抽出」
セクション23.4.9 「ndb_desc — NDB テーブルの表示」
セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」
セクション5.1.7 「サーバーコマンドオプション」
セクション2.2 「一般的なバイナリを使用した MySQL の Unix/Linux へのインストール」

--raw

セクション4.5.1.1 「mysql クライアントオプション」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」
セクション4.6.8.3 「バイナリログファイルのバックアップのための mysqlbinlog の使用」
セクション17.1.3.5 「フェイルオーバーおよびスケールアウトでの GTID の使用」

--read-from-remote-master

セクション6.2.2 「MySQL で提供される権限」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」
セクション17.1.3.5 「フェイルオーバーおよびスケールアウトでの GTID の使用」

--read-from-remote-server

セクション6.2.2 「MySQL で提供される権限」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」
セクション4.6.8.4 「mysqlbinlog サーバー ID の指定」
セクション7.5.1 「バイナリログを使用したポイントインタイムリカバリ」
セクション17.3.2 「バイナリログファイルとリレーログファイルの暗号化」
セクション4.6.8.3 「バイナリログファイルのバックアップのための mysqlbinlog の使用」
セクション17.1.3.5 「フェイルオーバーおよびスケールアウトでの GTID の使用」

--read-only

セクション4.6.4.2 「myisamchk のチェックオプション」

--real_table_name

セクション23.4.28 「ndb_size.pl — NDBCLUSTER サイズ要件エスティメータ」

--rebuild-indexes

セクション23.1.4 「NDB Cluster の新機能」
セクション23.3.3.6 「NDB Cluster データノードの定義」
セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

--reconnect

セクション4.5.1.1 「mysql クライアントオプション」

--recover

セクション4.6.4.2 「myisamchk のチェックオプション」
セクション4.6.4.1 「myisamchk の一般オプション」
セクション4.6.4.3 「myisamchk の修復オプション」
セクション4.6.4.6 「myisamchk メモリ使用量」

--rejects

セクション23.4.13 「ndb_import — NDB への CSV データのインポート」

--relative

セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」

--relay-log

セクション18.4.6 「グループレプリケーションでの MySQL Enterprise Backup の使用」
セクション17.2.2.3 「起動オプションとレプリケーションチャンネル」

--relay-log-index

セクション17.2.2.3 「起動オプションとレプリケーションチャンネル」

--relay-log-purge

セクション17.1.6.3 「Replica Server のオプションと変数」

--relay-log-recovery

セクション17.1.6.3 「Replica Server のオプションと変数」

セクション13.7.7.35 「SHOW REPLICAS | SLAVE STATUS ステートメント」

セクション5.1.19 「サーバーの停止プロセス」

セクション17.4.2 「レプリカの予期しない停止の処理」

セクション17.5.1.34 「レプリケーションとトランザクションの非一貫性」

--relay-log-space-limit

セクション17.1.6.3 「Replica Server のオプションと変数」

セクション17.2.2.3 「起動オプションとレプリケーションチャンネル」

--reload

セクション23.5.5 「NDB Cluster のローリング再起動の実行」

セクション23.5.7.2 「NDB Cluster データノードのオンラインでの追加: 基本手順」

セクション23.5.7.3 「NDB Cluster データノードのオンラインでの追加: 詳細な例」

セクション23.3.3 「NDB Cluster 構成ファイル」

セクション23.4.4 「ndb_mgmd — NDB Cluster 管理サーバーデーモン」

セクション23.2.2.3 「Windows での NDB Cluster の初期起動」

セクション23.1.7.10 「複数の NDB Cluster ノードに関する制限事項」

--remap-column

セクション23.1.4 「NDB Cluster の新機能」

セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

--remove

セクション23.2.2.4 「NDB Cluster プロセスを Windows サービスとしてインストール」

セクション23.4.4 「ndb_mgmd — NDB Cluster 管理サーバーデーモン」

セクション23.4.1 「ndbd — NDB Cluster データノードデーモン」

セクション2.3.4.8 「Windows のサービスとして MySQL を起動する」

セクション5.8.2.2 「Windows サービスとして複数の MySQL インスタンスの起動」

セクション5.1.7 「サーバーコマンドオプション」

--remove{

セクション23.4.4 「ndb_mgmd — NDB Cluster 管理サーバーデーモン」

--repair

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

--replace

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.5 「mysqlimport — データインポートプログラム」

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

--replicate-*

セクション13.4.2.2 「CHANGE REPLICATION FILTER ステートメント」

セクション17.1.6.3 「Replica Server のオプションと変数」

セクション27.12.11.9 「replication_applier_filters テーブル」

セクション27.12.11.8 「replication_applier_global_filters テーブル」

セクション17.2.5 「サーバーがレプリケーションフィルタリングルールをどのように評価するか」

セクション17.2.5.4 「レプリケーションチャンネルベースのフィルタ」

--replicate-*-db

セクション17.1.6.3 「Replica Server のオプションと変数」

セクション25.8「ストアプログラムの制約」

--replicate-do-*

セクション23.6.3「NDB Cluster レプリケーションの既知の問題」

--replicate-do-db

セクション13.4.2.2「CHANGE REPLICATION FILTER ステートメント」

セクション23.6.3「NDB Cluster レプリケーションの既知の問題」

セクション17.1.6.3「Replica Server のオプションと変数」

セクション13.7.7.35「SHOW REPLICATION | SLAVE STATUS ステートメント」

セクション13.3.1「START TRANSACTION、COMMIT および ROLLBACK ステートメント」

セクション17.2.5「サーバーがレプリケーションフィルタリングルールをどのように評価するか」

セクション17.2.5.1「データベースレベルレプリケーションオプションおよびバイナリロギングオプションの評価」

セクション17.1.6.4「バイナリロギングのオプションと変数」

セクション5.4.4「バイナリログ」

セクション17.5.1.31「レプリケーションと一時テーブル」

セクション17.5.1.26「レプリケーションと予約語」

セクション17.2.5.4「レプリケーションチャンネルベースのフィルタ」

セクション17.4.6「異なるレプリカへの異なるデータベースのレプリケート」

--replicate-do-db:channel_1

セクション17.1.6.3「Replica Server のオプションと変数」

--replicate-do-table

セクション16.6「BLACKHOLE ストレージエンジン」

セクション13.4.2.2「CHANGE REPLICATION FILTER ステートメント」

セクション23.6.3「NDB Cluster レプリケーションの既知の問題」

セクション17.1.6.3「Replica Server のオプションと変数」

セクション13.7.7.35「SHOW REPLICATION | SLAVE STATUS ステートメント」

セクション17.2.5.2「テーブルレベルレプリケーションオプションの評価」

セクション17.5.1.31「レプリケーションと一時テーブル」

セクション17.5.1.26「レプリケーションと予約語」

セクション17.2.5.3「レプリケーションフィルタリングオプション間の相互作用」

--replicate-do-table:channel_1

セクション17.1.6.3「Replica Server のオプションと変数」

--replicate-ignore-*

セクション23.6.3「NDB Cluster レプリケーションの既知の問題」

--replicate-ignore-db

セクション13.4.2.2「CHANGE REPLICATION FILTER ステートメント」

セクション23.6.3「NDB Cluster レプリケーションの既知の問題」

セクション17.1.6.3「Replica Server のオプションと変数」

セクション13.7.7.35「SHOW REPLICATION | SLAVE STATUS ステートメント」

セクション13.3.1「START TRANSACTION、COMMIT および ROLLBACK ステートメント」

セクション17.2.5「サーバーがレプリケーションフィルタリングルールをどのように評価するか」

セクション17.2.5.1「データベースレベルレプリケーションオプションおよびバイナリロギングオプションの評価」

セクション17.1.6.4「バイナリロギングのオプションと変数」

セクション5.4.4「バイナリログ」

セクション17.5.1.26「レプリケーションと予約語」

セクション17.2.5.4「レプリケーションチャンネルベースのフィルタ」

セクション17.2.5.3「レプリケーションフィルタリングオプション間の相互作用」

--replicate-ignore-db:channel_1

セクション17.1.6.3「Replica Server のオプションと変数」

--replicate-ignore-table

セクション16.6「BLACKHOLE ストレージエンジン」
セクション13.4.2.2「CHANGE REPLICATION FILTER ステートメント」
セクション23.6.3「NDB Cluster レプリケーションの既知の問題」
セクション17.1.6.3「Replica Server のオプションと変数」
セクション13.4.2.5「RESET REPLICA | SLAVE ステートメント」
セクション13.7.7.35「SHOW REPLICA | SLAVE STATUS ステートメント」
セクション17.2.5.2「テーブルレベルレプリケーションオプションの評価」
セクション17.5.1.31「レプリケーションと一時テーブル」
セクション17.5.1.26「レプリケーションと予約語」

--replicate-ignore-table:channel_1

セクション17.1.6.3「Replica Server のオプションと変数」

--replicate-rewrite-db

セクション13.4.2.2「CHANGE REPLICATION FILTER ステートメント」
セクション17.1.6.3「Replica Server のオプションと変数」
セクション17.2.5「サーバーがレプリケーションフィルタリングルールをどのように評価するか」

--replicate-same-server-id

セクション13.4.2.1「CHANGE MASTER TO ステートメント」
セクション13.4.2.3「CHANGE REPLICATION SOURCE TO ステートメント」
セクション17.1.4.3「GTID トランザクションのオンラインでの無効化」
セクション17.1.6.3「Replica Server のオプションと変数」
セクション17.1.6「レプリケーションおよびバイナリロギングのオプションと変数」

--replicate-wild-do-table

セクション13.4.2.2「CHANGE REPLICATION FILTER ステートメント」
セクション17.1.6.3「Replica Server のオプションと変数」
セクション13.7.7.35「SHOW REPLICA | SLAVE STATUS ステートメント」
セクション17.2.5「サーバーがレプリケーションフィルタリングルールをどのように評価するか」
セクション25.8「ストアプログラムの制約」
セクション17.2.5.2「テーブルレベルレプリケーションオプションの評価」
セクション17.5.1.31「レプリケーションと一時テーブル」
セクション17.4.6「異なるレプリカへの異なるデータベースのレプリケート」

--replicate-wild-do-table:channel_1

セクション17.1.6.3「Replica Server のオプションと変数」

--replicate-wild-ignore-table

セクション13.4.2.2「CHANGE REPLICATION FILTER ステートメント」
セクションA.14「MySQL 8.0 FAQ: レプリケーション」
セクション23.6.3「NDB Cluster レプリケーションの既知の問題」
セクション17.1.6.3「Replica Server のオプションと変数」
セクション13.7.7.35「SHOW REPLICA | SLAVE STATUS ステートメント」
セクション17.2.5.2「テーブルレベルレプリケーションオプションの評価」
セクション17.5.1.31「レプリケーションと一時テーブル」

--replicate-wild-ignore:channel_1

セクション17.1.6.3「Replica Server のオプションと変数」

replication-ignore-table

セクション17.5.1.40「レプリケーションとビュー」

--report-host

セクション13.7.7.33「SHOW REPLICAS | SHOW SLAVE HOSTS ステートメント」

セクション18.10「よくある質問」
セクション17.1.7.1「レプリケーションステータスの確認」

--report-password

セクション13.7.7.33「SHOW REPLICAS | SHOW SLAVE HOSTS ステートメント」
セクション17.1.6.2「レプリケーションソースのオプションと変数」

--report-port

セクション13.7.7.33「SHOW REPLICAS | SHOW SLAVE HOSTS ステートメント」

--report-user

セクション13.7.7.33「SHOW REPLICAS | SHOW SLAVE HOSTS ステートメント」
セクション17.1.6.2「レプリケーションソースのオプションと変数」

--require-row-format

セクション4.6.8「mysqlbinlog — バイナリログファイル进行处理するためのユーティリティー」
セクション17.3.3「レプリケーション権限チェック」

--restart

セクション2.5.6.1「Docker を使用した MySQL Server デプロイメントの基本ステップ」

--restore-data

セクション23.1.4「NDB Cluster の新機能」
NDB バックアップを以前のバージョンの NDB Cluster に復元
セクション23.4.23「ndb_restore — NDB Cluster バックアップの復元」
元のノードより少ないノードへのリストア

--restore-epoch

セクション23.6.9「NDB Cluster レプリケーションによる NDB Cluster バックアップ」
セクション23.6.9.2「NDB Cluster レプリケーションを使用したポイントインタイムリカバリ」
セクション23.4.23「ndb_restore — NDB Cluster バックアップの復元」

--restore-meta

NDB Cluster の新しいバージョンへの NDB バックアップの復元
セクション23.4.23「ndb_restore — NDB Cluster バックアップの復元」
元のノードより多くのノードへのリストア

--restore-privilege-tables

セクション23.1.4「NDB Cluster の新機能」
セクション23.4.23「ndb_restore — NDB Cluster バックアップの復元」

--result-file

セクション4.6.8「mysqlbinlog — バイナリログファイル进行处理するためのユーティリティー」
セクション4.5.4「mysqldump — データベースバックアッププログラム」
セクション4.5.6「mysqlpump — データベースバックアッププログラム」
セクション4.6.8.3「バイナリログファイルのバックアップのための mysqlbinlog の使用」

--resume

セクション23.4.13「ndb_import — NDB への CSV データのインポート」

--retries

セクション23.4.9「ndb_desc — NDB テーブルの表示」

--rewrite-database

セクション23.4.23「ndb_restore — NDB Cluster バックアップの復元」

--rewrite-db

セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」

--routines

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

セクション2.11.6 「Unix/Linux での MySQL バイナリまたはパッケージベースのインストールのアップグレード」

セクション7.4.5.3 「ストアードプログラムのダンプ」

セクション7.4.5.4 「テーブル定義と内容の個別のダンプ」

セクション14.7 「データディクショナリの使用法の違い」

セクション4.6.8.3 「バイナリログファイルのバックアップのための mysqlbinlog の使用」

--rowbatch

セクション23.4.13 「ndb_import — NDB への CSV データのインポート」

--rowbytes

セクション23.4.13 「ndb_import — NDB への CSV データのインポート」

--rowid

セクション23.4.24 「ndb_select_all — NDB テーブルの行の出力」

--ROWS

セクション23.4.7 「ndb_config — NDB Cluster 構成情報の抽出」

S

[[index top](#)]

-S

セクション4.6.2 「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティ」

セクション7.6.4 「MyISAM テーブルの最適化」

セクション4.5.1.1 「mysql クライアントオプション」

セクション4.5.1.2 「mysql クライアントコマンド」

セクション4.2.1 「MySQL プログラムの起動」

セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティ」

セクション4.4.2 「mysql_secure_installation — MySQL インストールのセキュリティ改善」

セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」

セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.5 「mysqlimport — データインポートプログラム」

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」

セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」

セクション23.2.8.2 「NDB Cluster Auto-Installer の使用」

セクション23.2.8.1 「NDB Cluster 自動インストーラの要件」

セクション23.4.26 「ndb_setup.py — NDB Cluster 用ブラウザベースの Auto-Installer の起動 (非推奨)」

セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」

セクション4.6.4.4 「その他の myisamchk オプション」

セクション4.2.3 「サーバーに接続するためのコマンドオプション」

-S

セクション4.6.1 「ibd2sdi — InnoDB テーブルスペース SDI 抽出ユーティリティ」

セクション4.6.2 「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティ」

セクション4.7.2 「my_print_defaults — オプションファイルからのオプションの表示」

セクション7.6.2 「MyISAM テーブルのエラーのチェック方法」
セクション7.6.3 「MyISAM テーブルの修復方法」
セクション7.6.5 「MyISAM テーブル保守スケジュールのセットアップ」
セクション4.6.3 「mysiam_ftdump — 全文インデックス情報の表示」
セクション4.6.4.1 「mysiamchk の一般オプション」
セクション4.6.6 「mysiampack — 圧縮された読み取り専用の MyISAM テーブルの生成」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」
セクション4.6.8 「mysqlbinlog — バイナリログファイル进行处理するためのユーティリティー」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.6.9 「mysqldumpslow — スロークエリーログファイルの要約」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」
セクション23.4.16 「ndb_perror — NDB エラーメッセージ情報の取得」
セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」
セクション23.4.26 「ndb_setup.py — NDB Cluster 用ブラウザベースの Auto-Installer の起動 (非推奨)」
セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」
セクション4.8.2 「 perror — MySQL エラーメッセージ情報の表示」
セクション2.2 「一般的なバイナリを使用した MySQL の Unix/Linux へのインストール」

--safe-recover

セクション4.6.4.1 「mysiamchk の一般オプション」
セクション4.6.4.3 「mysiamchk の修復オプション」
セクション4.6.4.6 「mysiamchk メモリー使用量」

--safe-updates

セクション4.5.1.6 「mysql クライアントのヒント」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.5.1.2 「mysql クライアントコマンド」
セクション5.1.8 「サーバーシステム変数」

--safe-user-create

セクション5.1.7 「サーバーコマンドオプション」

--savequeries

セクション23.4.28 「ndb_size.pl — NDBCLUSTER サイズ要件エスティメータ」

--secure-auth

セクション1.3 「MySQL 8.0 の新機能」
セクション6.2.17 「プラグブル認証」

--select-limit

セクション4.5.1.6 「mysql クライアントのヒント」
セクション4.5.1.1 「mysql クライアントオプション」

--server-arg

セクション1.3 「MySQL 8.0 の新機能」

--server-file

セクション1.3 「MySQL 8.0 の新機能」

--server-id

セクション4.6.8 「mysqlbinlog — バイナリログファイル进行处理するためのユーティリティー」
セクション23.6.2 「NDB Cluster レプリケーションの一般的な要件」
セクション13.7.7.33 「SHOW REPLICAS | SHOW SLAVE HOSTS ステートメント」

--server-id-bits

セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」

--server-log-file

セクション23.4.26 「ndb_setup.py — NDB Cluster 用ブラウザベースの Auto-Installer の起動 (非推奨)」

--server-name

セクション23.4.26 「ndb_setup.py — NDB Cluster 用ブラウザベースの Auto-Installer の起動 (非推奨)」

--server-public-key-path

セクション4.5.1.1 「mysql クライアントオプション」

セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」

セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.5 「mysqlexport — データインポートプログラム」

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」

セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」

セクション6.4.1.2 「SHA-2 プラガブル認証のキャッシュ」

セクション6.4.1.3 「SHA-256 プラガブル認証」

セクション4.2.3 「サーバーに接続するためのコマンドオプション」

service-startup-timeout

セクション4.3.3 「mysql.server — MySQL サーバー起動スクリプト」

--set-auto-increment

セクション4.6.4.4 「その他の myisamchk オプション」

--set-charset

セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

--set-collation

セクション4.6.4.3 「myisamchk の修復オプション」

--set-gtid-purged

セクション17.1.3.8 「GTID を操作するストアドファンクションの例」

セクション2.11.14 「MySQL データベースのほかのマシンへのコピー」

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション7.4.1 「mysqldump による SQL フォーマットでのデータのダンプ」

mysqldump を使用したデータスナップショットの作成

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

セクション17.1.3.5 「フェイルオーバーおよびスケールアウトでの GTID の使用」

--shared-memory-base-name

セクション5.8 「1 つのマシン上での複数の MySQL インスタンスの実行」

セクション4.5.1.1 「mysql クライアントオプション」

セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」

セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.5 「mysqlexport — データインポートプログラム」

セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」
セクション4.2.3 「サーバーに接続するためのコマンドオプション」
セクション5.8.4 「複数サーバー環境でのクライアントプログラムの使用」

--short-form

セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」

--show

セクション4.7.2 「my_print_defaults — オプションファイルからのオプションの表示」

--show-create-skip-secondary-engine

セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション5.1.8 「サーバーシステム変数」

--show-slave-auth-info

セクション17.1.6.3 「Replica Server のオプションと変数」
セクション13.7.7.33 「SHOW REPLICAS | SHOW SLAVE HOSTS ステートメント」
セクション17.1.6.2 「レプリケーションソースのオプションと変数」

--show-table-type

セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」

--show-temp-status

セクション23.4.27 「ndb_show_tables — NDB テーブルのリストの表示」

--show-warnings

セクション4.5.1.1 「mysql クライアントオプション」
セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」

--shutdown-timeout

セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」

--sigint-ignore

セクション4.10 「MySQL での Unix シグナル処理」
セクション4.5.1.1 「mysql クライアントオプション」

--silent

セクション7.6.5 「MyISAM テーブル保守スケジュールのセットアップ」
セクション4.6.4.1 「myisamchk の一般オプション」
セクション4.6.6 「myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.3.4 「mysqld_multi — 複数の MySQL サーバーの管理」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」
セクション23.4.16 「ndb_perror — NDB エラーメッセージ情報の取得」
セクション4.8.2 「 perror — MySQL エラーメッセージ情報の表示」

--single-transaction

セクション15.18.1 「InnoDB バックアップ」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション7.2 「データベースバックアップ方法」
セクション7.3.1 「バックアップポリシーの確立」

セクション15.8.9「ページ構成」

--single-user

セクション23.4.30「ndb_waiter — NDB Cluster が特定のステータスに達するまで待機」

--skip

セクション4.5.4「mysqldump — データベースバックアッププログラム」

セクション5.1.7「サーバーコマンドオプション」

セクション4.2.2.4「プログラムオプション修飾子」

--skip-add-drop-table

セクション4.5.4「mysqldump — データベースバックアッププログラム」

--skip-add-locks

セクション4.5.4「mysqldump — データベースバックアッププログラム」

--skip-admin-ssl

セクション5.1.7「サーバーコマンドオプション」

--skip-auto-rehash

セクション4.5.1.1「mysql クライアントオプション」

--skip-binary-as-hex

セクション4.5.1.1「mysql クライアントオプション」

--skip-binlog

セクション18.4.6「グループレプリケーションでの MySQL Enterprise Backup の使用」

--skip-broken-objects

セクション23.4.23「ndb_restore — NDB Cluster バックアップの復元」

--skip-character-set-client-handshake

セクション10.10.7.1「cp932 文字セット」

セクションA.11「MySQL 8.0 FAQ: MySQL の中国語、日本語、および韓国語の文字セット」

セクション5.1.7「サーバーコマンドオプション」

セクション5.1.8「サーバーシステム変数」

--skip-color

セクション23.4.29「ndb_top — NDB スレッドの CPU 使用率情報の表示」

--skip-colors

セクション23.4.29「ndb_top — NDB スレッドの CPU 使用率情報の表示」

--skip-column-names

セクション4.5.1.1「mysql クライアントオプション」

--skip-comments

セクション4.5.1.1「mysql クライアントオプション」

セクション4.5.4「mysqldump — データベースバックアッププログラム」

--skip-config-cache

セクション23.4.4「ndb_mgmd — NDB Cluster 管理サーバーデーモン」

--skip-data

セクション4.6.1「ibd2sdi — InnoDB テーブルスペース SDI 抽出ユーティリティ」

--skip-database

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

--skip-defer-table-indexes

セクション4.5.6 「mysqldump — データベースバックアッププログラム」

--skip-definer

セクション4.5.6 「mysqldump — データベースバックアッププログラム」

--skip-disable-keys

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

--skip-dump-date

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

--skip-dump-rows

セクション4.5.6 「mysqldump — データベースバックアッププログラム」

--skip-engine_name

セクション26.13 「INFORMATION_SCHEMA ENGINES テーブル」

セクション13.7.7.16 「SHOW ENGINES ステートメント」

--skip-events

セクション4.5.6 「mysqldump — データベースバックアッププログラム」

セクション7.4.5.3 「ストアドプログラムのダンプ」

--skip-extended-insert

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

--skip-external-locking

セクションB.3.3.3 「MySQL が繰り返しクラッシュする場合の対処方法」

セクション5.1.7 「サーバーコマンドオプション」

セクション5.1.8 「サーバーシステム変数」

セクション8.14.3 「一般的なスレッドの状態」

セクション8.11.5 「外部ロック」

--skip-federated

セクション17.4.4 「異なるソースおよびレプリカのストレージエンジンでのレプリケーションの使用」

--skip-grant-tables

セクション13.7.8.3 「FLUSH ステートメント」

セクション26.46 「INFORMATION_SCHEMA USER_ATTRIBUTES テーブル」

セクション13.7.4.3 「INSTALL COMPONENT ステートメント」

セクション13.7.4.4 「INSTALL PLUGIN ステートメント」

セクション6.2.2 「MySQL で提供される権限」

セクション6.2.21 「MySQL への接続の問題のトラブルシューティング」

セクション5.3 「mysql システムスキーマ」

セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」

root のパスワードのリセット: 一般的な手順

セクション25.4.2 「イベントスケジューラの構成」

セクション4.2.2.1 「コマンド行でのオプションの使用」

セクション5.5.1 「コンポーネントのインストールおよびアンインストール」

セクション5.1.7 「サーバーコマンドオプション」

セクション5.1.8 「サーバーシステム変数」

セクション6.2.15 「パスワード管理」

セクション6.2.17「プラグブル認証」
セクション5.6.1「プラグインのインストールおよびアンインストール」
セクション5.7.1「ユーザー定義関数のインストールおよびアンインストール」
セクション13.7.4.1「ユーザー定義関数用の CREATE FUNCTION ステートメント」
セクション6.2.13「権限変更が有効化される時期」

--skip-graphs

セクション23.4.29「ndb_top — NDB スレッドの CPU 使用率情報の表示」

--skip-gtids

セクション4.6.8「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」
セクション7.5.1「バイナリログを使用したポイントインタイムリカバリ」

--skip-host-cache

セクション5.1.12.3「DNS ルックアップとホストキャッシュ」
セクション6.2.21「MySQL への接続の問題のトラブルシューティング」
セクション5.1.7「サーバーコマンドオプション」
セクション5.1.8「サーバーシステム変数」

--skip-innodb

セクション15.14「InnoDB の起動オプションおよびシステム変数」
セクションA.14「MySQL 8.0 FAQ: レプリケーション」
セクション5.1.7「サーバーコマンドオプション」
セクション5.6.1「プラグインのインストールおよびアンインストール」

--skip-innodb-adaptive-hash-index

セクション15.14「InnoDB の起動オプションおよびシステム変数」

--skip-kill-mysqld

セクション4.3.2「mysqld_safe — MySQL サーバー起動スクリプト」

--skip-line-numbers

セクション4.5.1.1「mysql クライアントオプション」

--skip-lock-tables

セクション4.5.4「mysqldump — データベースバックアッププログラム」

--skip-log-bin

セクション17.1.3.4「GTID を使用したレプリケーションのセットアップ」
セクション17.1.3.1「GTID 形式および格納」
セクション17.1.6.4「バイナリロギングのオプションと変数」
セクション5.4.4「バイナリログ」
セクション17.1.2.2「レプリカ構成の設定」
セクション17.5.4「レプリケーションのトラブルシューティング」
セクション17.5.3「レプリケーションセットアップをアップグレードする」
セクション17.1.2.1「レプリケーションソース構成の設定」

--skip-mysqlex

セクション20.5.2「X プラグイン の無効化」

--skip-named-commands

セクション4.5.1.1「mysql クライアントオプション」

--skip-ndbcluster

セクション23.3.2.5「NDB Cluster mysqld オプションおよび変数のリファレンス」

NDB Cluster の MySQL Server オプション

--skip-network-timeout

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

--skip-new

セクション5.9.1 「MySQL サーバーのデバッグ」
セクション13.7.3.4 「OPTIMIZE TABLE ステートメント」
セクション5.1.7 「サーバーコマンドオプション」
セクション5.1.8 「サーバーシステム変数」

--skip-nodegroup

セクション23.4.12 「ndb_error_reporter — NDB エラーレポートユーティリティー」

--skip-opt

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

--skip-pager

セクション4.5.1.1 「mysql クライアントオプション」

--skip-partition

セクション1.3 「MySQL 8.0 の新機能」

--skip-password

セクション4.5.1.1 「mysql クライアントオプション」
セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」
セクション4.2.4 「コマンドオプションを使用した MySQL Server への接続」
セクション4.2.3 「サーバーに接続するためのコマンドオプション」

--skip-plugin-innodb-file-per-table

セクション5.1.7 「サーバーコマンドオプション」

--skip-plugin_name

セクション5.6.1 「プラグインのインストールおよびアンインストール」

--skip-quick

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

--skip-quote-names

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

--skip-reconnect

セクション4.5.1.6 「mysql クライアントのヒント」
セクション4.5.1.1 「mysql クライアントオプション」

--skip-relaylog

セクション18.4.6 「グループレプリケーションでの MySQL Enterprise Backup の使用」

--skip-routines

セクション4.5.6 「mysqldump — データベースバックアッププログラム」
セクション7.4.5.3 「ストアドプログラムのダンプ」

--skip-safe-updates

セクション4.5.1.1 「mysql クライアントオプション」

--skip-set-charset

セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.6 「mysqldump — データベースバックアッププログラム」

--skip-show-database

セクション6.2.2 「MySQL で提供される権限」
セクション1.8.5 「MySQL のサポータ」
セクション13.7.7.14 「SHOW DATABASES ステートメント」
セクション5.1.7 「サーバーコマンドオプション」

--skip-slave-preserve-commit-order

セクション17.1.6.4 「バイナリロギングのオプションと変数」
セクション5.4.4 「バイナリログ」

--skip-slave-start

セクション17.1.3.4 「GTID を使用したレプリケーションのセットアップ」
セクション23.6.5 「NDB Cluster のレプリケーションの準備」
セクション23.6.9 「NDB Cluster レプリケーションによる NDB Cluster バックアップ」
セクション23.6.6 「NDB Cluster レプリケーションの開始 (シングルレプリケーションチャンネル)」
セクション17.1.6.3 「Replica Server のオプションと変数」
セクション13.4.2.7 「START REPLICA | SLAVE ステートメント」
セクション17.5.4 「レプリケーションのトラブルシューティング」
セクション17.5.3 「レプリケーションセットアップをアップグレードする」
セクション17.1.2.8 「レプリケーション環境へのレプリカの追加」
既存のデータによるレプリケーションのセットアップ
セクション17.3.1 「暗号化接続を使用するためのレプリケーションの設定」
セクション17.2.2.3 「起動オプションとレプリケーションチャンネル」

--skip-sort

セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」

--skip-ssl

セクション1.3 「MySQL 8.0 の新機能」
セクション5.1.7 「サーバーコマンドオプション」

--skip-stack-trace

セクション5.9.1.4 「gdb での mysqld のデバッグ」
セクション5.1.7 「サーバーコマンドオプション」

--skip-super-large-pages

セクション5.1.7 「サーバーコマンドオプション」
セクション8.12.3.2 「ラージページのサポートの有効化」

--skip-symbolic-links

セクション13.1.20 「CREATE TABLE ステートメント」
セクション8.12.2.2 「Unix 上の MyISAM へのシンボリックリンクの使用」
セクション5.1.7 「サーバーコマンドオプション」
セクション5.1.8 「サーバーシステム変数」
セクション6.1.3 「攻撃者に対する MySQL のセキュアな状態の維持」

--skip-sys-schema

セクション2.11.3 「MySQL のアップグレードプロセスの内容」

セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

--skip-syslog

セクション4.3.2 「mysqld_safe — MySQL サーバー起動スクリプト」

--skip-table-check

セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

--skip-triggers

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

セクション7.4.5.3 「ストアードプログラムのダンプ」

--skip-tz-utc

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

--skip-unknown-objects

セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

--skip-version-check

セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

--skip-warn

セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティー」

--skip-watch-progress

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

--skip-write-binlog

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

--skip_grant_tables

セクション4.2.2.1 「コマンド行でのオプションの使用」

--slave-parallel-workers

セクション17.2.2.3 「起動オプションとレプリケーションチャンネル」

--slave-preserve-commit-order

セクション17.1.6.4 「バイナリロギングのオプションと変数」

セクション5.4.4 「バイナリログ」

--slave-skip-counter

セクション17.2.2.3 「起動オプションとレプリケーションチャンネル」

--slave-skip-errors

セクション23.6.8 「NDB Cluster レプリケーションによるフェイルオーバーの実装」

セクション17.1.6.3 「Replica Server のオプションと変数」

セクション17.5.1.29 「レプリケーション中のレプリカエラー」

--slave-sql-verify-checksum

セクション17.1.6.3 「Replica Server のオプションと変数」

セクション17.1.6.4 「バイナリロギングのオプションと変数」

--slave_net_timeout

セクション17.2.2.3 「起動オプションとレプリケーションチャンネル」

--sleep

セクション4.5.2 「mysqldadmin — A MySQL Server 管理プログラム」

--sleep-time

セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」

--slice-id

セクション23.1.4 「NDB Cluster の新機能」

セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

slice-id

セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

--slow-start-timeout

セクション5.1.7 「サーバーコマンドオプション」

--slow_query_log

セクション5.4.5 「スロークエリエラーログ」

セクション5.4.1 「一般クエリエラーログおよびスロークエリエラーログの出力先の選択」

--slow_query_log_file

セクション5.8 「1 つのマシン上での複数の MySQL インスタンスの実行」

セクション5.4.5 「スロークエリエラーログ」

--socket

セクション5.8 「1 つのマシン上での複数の MySQL インスタンスの実行」

セクションB.3.2.2 「[ローカルの] MySQL サーバーに接続できません」

セクション2.5.6.3 「Docker を使用した Windows およびその他の Linux 以外のプラットフォームへの MySQL のデプロイ」

セクションB.3.3.6 「MySQL の UNIX ソケットファイルを保護または変更する方法」

セクション6.2.21 「MySQL への接続の問題のトラブルシューティング」

セクション2.3.4.9 「MySQL インストールのテスト」

セクション4.5.1.1 「mysql クライアントオプション」

セクション2.9.7 「MySQL ソース構成オプション」

セクション4.2.1 「MySQL プログラムの起動」

セクション4.7.1 「mysql_config — クライアントのコンパイル用オプションの表示」

セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティー」

セクション4.4.2 「mysql_secure_installation — MySQL インストールのセキュリティ改善」

セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

セクション4.5.2 「mysqldadmin — A MySQL Server 管理プログラム」

セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

セクション4.3.2 「mysqld_safe — MySQL サーバー起動スクリプト」

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.5 「mysqlimport — データインポートプログラム」

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」

セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」

セクション23.4.28 「ndb_size.pl — NDBCLUSTER サイズ要件エスティメータ」

セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」

セクション5.8.3 「Unix 上での複数の MySQL インスタンスの実行」

セクション4.2.4 「コマンドオプションを使用した MySQL Server への接続」
セクション4.2.3 「サーバーに接続するためのコマンドオプション」
セクション5.1.7 「サーバーコマンドオプション」
セクション5.8.4 「複数サーバー環境でのクライアントプログラムの使用」

socket

セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティー」
セクション4.2.2.2 「オプションファイルの使用」

--sort

セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」

--sort-index

セクション7.6.4 「MyISAM テーブルの最適化」
セクション4.6.4.4 「その他の myisamchk オプション」

--sort-records

セクション7.6.4 「MyISAM テーブルの最適化」
セクション4.6.4.4 「その他の myisamchk オプション」

--sort-recover

セクション4.6.4.1 「myisamchk の一般オプション」
セクション4.6.4.3 「myisamchk の修復オプション」
セクション4.6.4.6 「myisamchk メモリー使用量」

--sort_buffer_size

セクション5.1.7 「サーバーコマンドオプション」

--sporadic-binlog-dump-fail

セクション17.1.6.4 「バイナリロギングのオプションと変数」

--sql-mode

セクションA.3 「MySQL 8.0 FAQ: サーバー SQL モード」
セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」
セクション5.1.11 「サーバー SQL モード」
セクション5.1.7 「サーバーコマンドオプション」
第12章 「関数と演算子」

sql-mode

セクション5.1.11 「サーバー SQL モード」

--ssl

セクション1.3 「MySQL 8.0 の新機能」
セクション6.3.3.1 「MySQL を使用した SSL および RSA 証明書とキーの作成」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.4.2 「mysql_secure_installation — MySQL インストールのセキュリティ改善」
セクション4.4.3 「mysql_ssl_rsa_setup — SSL/RSA ファイルの作成」
セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」

セクション5.1.7 「サーバーコマンドオプション」
セクション5.1.8 「サーバーシステム変数」
セクション6.3.1 「暗号化接続を使用するための MySQL の構成」
セクション5.1.12.2 「管理接続管理」

--ssl*

セクション4.5.1.1 「mysql クライアントオプション」
セクション4.4.2 「mysql_secure_installation — MySQL インストールのセキュリティー改善」
セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」

--ssl-ca

セクション13.7.1.1 「ALTER USER ステートメント」
セクション13.7.1.3 「CREATE USER ステートメント」
セクション4.4.3 「mysql_ssl_rsa_setup — SSL/RSA ファイルの作成」
セクション6.3.3.2 「openssl を使用した SSL 証明書およびキーの作成」
セクション4.2.3 「サーバーに接続するためのコマンドオプション」
セクション5.1.7 「サーバーコマンドオプション」
セクション6.3.1 「暗号化接続を使用するための MySQL の構成」
セクション5.1.9.4 「永続的で永続的に制限されないシステム変数」

--ssl-capath

セクション4.2.3 「サーバーに接続するためのコマンドオプション」
セクション5.1.7 「サーバーコマンドオプション」
セクション6.3.1 「暗号化接続を使用するための MySQL の構成」

--ssl-cert

セクション13.7.1.1 「ALTER USER ステートメント」
セクション13.7.1.3 「CREATE USER ステートメント」
セクション4.4.3 「mysql_ssl_rsa_setup — SSL/RSA ファイルの作成」
セクション6.3.3.2 「openssl を使用した SSL 証明書およびキーの作成」
セクション4.2.3 「サーバーに接続するためのコマンドオプション」
セクション6.3.1 「暗号化接続を使用するための MySQL の構成」
セクション5.1.9.4 「永続的で永続的に制限されないシステム変数」

--ssl-cipher

セクション4.2.3 「サーバーに接続するためのコマンドオプション」
セクション6.3.2 「暗号化された接続 TLS プロトコルおよび暗号」
セクション6.3.1 「暗号化接続を使用するための MySQL の構成」

--ssl-crl

セクション4.2.3 「サーバーに接続するためのコマンドオプション」
セクション6.3.1 「暗号化接続を使用するための MySQL の構成」

--ssl-crlpath

セクション4.2.3 「サーバーに接続するためのコマンドオプション」
セクション6.3.1 「暗号化接続を使用するための MySQL の構成」

--ssl-fips-mode

セクション6.8 「FIPS のサポート」

セクション4.5.1.1 「mysql クライアントオプション」
セクション4.4.2 「mysql_secure_installation — MySQL インストールのセキュリティ改善」
セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.5 「mysqlexport — データインポートプログラム」
セクション4.5.6 「mysqlexport — データベースバックアッププログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」
セクション4.2.3 「サーバーに接続するためのコマンドオプション」

--ssl-key

セクション13.7.1.1 「ALTER USER ステートメント」
セクション13.7.1.3 「CREATE USER ステートメント」
セクション4.4.3 「mysql_ssl_rsa_setup — SSL/RSA ファイルの作成」
セクション6.3.3.2 「openssl を使用した SSL 証明書およびキーの作成」
セクション4.2.3 「サーバーに接続するためのコマンドオプション」
セクション6.3.1 「暗号化接続を使用するための MySQL の構成」
セクション5.1.9.4 「永続的で永続的に制限されないシステム変数」

--ssl-mode

セクション13.7.1.1 「ALTER USER ステートメント」
セクション13.7.1.3 「CREATE USER ステートメント」
セクション6.1.6 「LOAD DATA LOCAL のセキュリティ上の考慮事項」
セクション1.3 「MySQL 8.0 の新機能」
セクション4.2.5 「URI 類似文字列またはキーと値のペアを使用したサーバーへの接続」
セクション4.2.3 「サーバーに接続するためのコマンドオプション」
セクション6.3.1 「暗号化接続を使用するための MySQL の構成」

--ssl-verify-server-cert

セクション1.3 「MySQL 8.0 の新機能」

--ssl-xxx

セクション4.5.1.1 「mysql クライアントオプション」
セクション4.4.2 「mysql_secure_installation — MySQL インストールのセキュリティ改善」
セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.5 「mysqlexport — データインポートプログラム」
セクション4.5.6 「mysqlexport — データベースバックアッププログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」
セクション2.9.6 「SSL ライブラリサポートの構成」
セクション4.2.3 「サーバーに接続するためのコマンドオプション」
セクション6.3.1 「暗号化接続を使用するための MySQL の構成」

--staging-tries

セクション23.4.15 「ndb_move_data — NDB データコピーユーティリティ」

--standalone

セクション2.3.4.6 「Windows のコマンド行からの MySQL の起動」
セクション5.1.7 「サーバーコマンドオプション」
セクション5.9.1.2 「トレースファイルの作成」

--start-datetime

セクション4.6.8 「mysqlbinlog — バイナリログファイル进行处理するためのユーティリティ」
セクション7.5.2 「イベントの位置を使用したポイントインタイムリカバリ」

--start-page

セクション4.6.2 「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティ」

--start-position

セクション4.6.8 「mysqlbinlog — バイナリログファイル进行处理するためのユーティリティ」
セクション7.5.2 「イベントの位置を使用したポイントインタイムリカバリ」

--state-dir

セクション23.4.13 「ndb_import — NDB への CSV データのインポート」

--stats

セクション4.6.3 「myisam_ftdump — 全文インデックス情報の表示」
セクション23.4.13 「ndb_import — NDB への CSV データのインポート」

--status

セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」

--stop-datetime

セクション4.6.8 「mysqlbinlog — バイナリログファイル进行处理するためのユーティリティ」
セクション7.5.2 「イベントの位置を使用したポイントインタイムリカバリ」

--stop-never

セクション4.6.8 「mysqlbinlog — バイナリログファイル进行处理するためのユーティリティ」
セクション4.6.8.4 「mysqlbinlog サーバー ID の指定」
セクション4.6.8.3 「バイナリログファイルのバックアップのための mysqlbinlog の使用」

--stop-never-slave-server-id

セクション4.6.8 「mysqlbinlog — バイナリログファイル进行处理するためのユーティリティ」

--stop-position

セクション4.6.8 「mysqlbinlog — バイナリログファイル进行处理するためのユーティリティ」
セクション7.5.2 「イベントの位置を使用したポイントインタイムリカバリ」

--strict-check

セクション4.6.1 「ibd2sdi — InnoDB テーブルスペース SDI 抽出ユーティリティ」
セクション4.6.2 「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティ」

--suffix

セクション6.3.3.1 「MySQL を使用した SSL および RSA 証明書とキーの作成」
セクション4.4.3 「mysql_ssl_rsa_setup — SSL/RSA ファイルの作成」

--super-large-pages

セクション5.1.7 「サーバーコマンドオプション」
セクション8.12.3.2 「ラージページのサポートの有効化」

--symbolic-links

セクション8.12.2.2 「Unix 上の MyISAM へのシンボリックリンクの使用」
セクション5.1.7 「サーバーコマンドオプション」
セクション5.1.8 「サーバーシステム変数」

--sys-*

セクション23.4.14 「[ndb_index_stat](#) — NDB インデックス統計ユーティリティー」

--sys-check

セクション23.4.14 「[ndb_index_stat](#) — NDB インデックス統計ユーティリティー」

--sys-create

セクション23.4.14 「[ndb_index_stat](#) — NDB インデックス統計ユーティリティー」

--sys-create-if-not-exist

セクション23.4.14 「[ndb_index_stat](#) — NDB インデックス統計ユーティリティー」

sys-create-if-not-exist

セクション23.4.14 「[ndb_index_stat](#) — NDB インデックス統計ユーティリティー」

--sys-create-if-not-valid

セクション23.4.14 「[ndb_index_stat](#) — NDB インデックス統計ユーティリティー」

--sys-drop

セクション23.4.14 「[ndb_index_stat](#) — NDB インデックス統計ユーティリティー」

--sys-skip-events

セクション23.4.14 「[ndb_index_stat](#) — NDB インデックス統計ユーティリティー」

--sys-skip-tables

セクション23.4.14 「[ndb_index_stat](#) — NDB インデックス統計ユーティリティー」

SYSCONFDIR

セクション4.2.2.2 「[オプションファイルの使用](#)」

--sysdate-is-now

セクション5.1.7 「[サーバーコマンドオプション](#)」

セクション5.1.8 「[サーバーシステム変数](#)」

セクション17.2.1.1 「[ステートメントベースおよび行ベースレプリケーションのメリットとデメリット](#)」

セクション17.5.1.14 「[レプリケーションとシステム関数](#)」

セクション12.7 「[日付および時間関数](#)」

--syslog

セクション4.5.1.6 「[mysql クライアントのヒント](#)」

セクション4.5.1.1 「[mysql クライアントオプション](#)」

セクション4.5.1.3 「[mysql クライアントロギング](#)」

セクション4.3.2 「[mysqld_safe](#) — MySQL サーバー起動スクリプト」

セクション2.5.9 「[systemd を使用した MySQL Server の管理](#)」

セクション4.9 「[環境変数](#)」

--syslog-tag

セクション4.3.2 「[mysqld_safe](#) — MySQL サーバー起動スクリプト」

--system

セクション23.4.7 「[ndb_config](#) — NDB Cluster 構成情報の抽出」

T

[\[index top\]](#)

-T

セクション4.4.1 「comp_err — MySQL エラーメッセージファイルのコンパイル」
セクション4.6.4.2 「myisamchk のチェックオプション」
セクション4.6.6 「myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」
セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」
セクション5.1.7 「サーバーコマンドオプション」

-t

セクション4.6.1 「ibd2sdi — InnoDB テーブルスペース SDI 抽出ユーティリティ」
セクション4.6.4.3 「myisamchk の修復オプション」
セクション4.6.6 「myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.6.9 「mysqldumpslow — スロークエリログファイルの要約」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション23.4.8 「ndb_delete_all — NDB テーブルからのすべての行の削除」
セクション23.4.9 「ndb_desc — NDB テーブルの表示」
セクション23.4.5 「ndb_mgm — NDB Cluster 管理クライアント」
セクション23.4.22 「ndb_redo_log_reader — クラスタ redo ログの内容の確認および印刷」
セクション23.4.24 「ndb_select_all — NDB テーブルの行の出力」
セクション23.4.27 「ndb_show_tables — NDB テーブルのリストの表示」
セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」
セクション23.4.30 「ndb_waiter — NDB Cluster が特定のステータスに達するまで待機」
セクション5.1.7 「サーバーコマンドオプション」

--tab

セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション7.4.3 「mysqldump による区切りテキストフォーマットでのデータのダンプ」
セクション23.4.13 「ndb_import — NDB への CSV データのインポート」
セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」
セクション7.2 「データベースバックアップ方法」
セクション7.1 「バックアップとリカバリの種類」
セクション7.4 「バックアップへの mysqldump の使用」

--table

セクション4.5.1.1 「mysql クライアントオプション」
セクション23.4.9 「ndb_desc — NDB テーブルの表示」

--tables

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」

--tc-heuristic-recover

セクション5.1.7 「サーバーコマンドオプション」

--tcp-ip

セクション4.3.4 「mysqld_multi — 複数の MySQL サーバーの管理」

--tee

セクション4.5.1.1 「mysql クライアントオプション」

セクション4.5.1.2 「mysql クライアントコマンド」

--temp-pool

セクション1.3 「MySQL 8.0 の新機能」

--tempdelay

セクション23.4.13 「ndb_import — NDB への CSV データのインポート」

--temperrors

セクション23.4.13 「ndb_import — NDB への CSV データのインポート」

--test

セクション4.6.6 「myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成」

--text

セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」

--thread_cache_size

セクション5.9.1.4 「gdb での mysqld のデバッグ」

--timeout

セクション23.4.30 「ndb_waiter — NDB Cluster が特定のステータスに達するまで待機」

--timezone

セクション5.1.15 「MySQL Server でのタイムゾーンのサポート」

セクション4.3.2 「mysqld_safe — MySQL サーバー起動スクリプト」

セクション5.1.8 「サーバーシステム変数」

セクションB.3.3.7 「タイムゾーンの問題」

--tls-ciphersuites

セクション4.5.1.1 「mysql クライアントオプション」

セクション4.4.2 「mysql_secure_installation — MySQL インストールのセキュリティ改善」

セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」

セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.5 「mysqlexport — データインポートプログラム」

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」

セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」

セクション4.2.3 「サーバーに接続するためのコマンドオプション」

セクション6.3.2 「暗号化された接続 TLS プロトコルおよび暗号」

セクション6.3.1 「暗号化接続を使用するための MySQL の構成」

--tls-version

セクション4.5.1.1 「mysql クライアントオプション」

セクション4.4.2 「mysql_secure_installation — MySQL インストールのセキュリティ改善」

セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」

セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.5 「mysqlexport — データインポートプログラム」

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」

セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」
セクション4.2.3 「サーバーに接続するためのコマンドオプション」
セクション6.3.2 「暗号化された接続 TLS プロトコルおよび暗号」
セクション6.3.1 「暗号化接続を使用するための MySQL の構成」

--tmpdir

セクション5.8 「1 つのマシン上での複数の MySQL インスタンスの実行」
セクション4.6.4.3 「myisamchk の修復オプション」
セクション4.6.4.6 「myisamchk メモリ使用量」
セクション4.6.6 「myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成」
セクションB.3.3.5 「MySQL が一時ファイルを格納する場所」
セクション2.3.4.8 「Windows のサービスとして MySQL を起動する」
セクション5.1.7 「サーバーコマンドオプション」
セクションB.3.2.11 「ファイルを作成/書き込みできない」

tmpdir

セクション2.3 「Microsoft Windows に MySQL をインストールする」

--to-last-log

セクション4.6.8 「mysqlbinlog — バイナリログファイル进行处理するためのユーティリティ」
セクション4.6.8.4 「mysqlbinlog サーバー ID の指定」
セクション4.6.8.3 「バイナリログファイルのバックアップのための mysqlbinlog の使用」

--transaction-isolation

セクション13.3.7 「SET TRANSACTION ステートメント」
セクション5.1.7 「サーバーコマンドオプション」
セクション5.1.8 「サーバーシステム変数」
セクション15.7.2.1 「トランザクション分離レベル」

--transaction-read-only

セクション13.3.7 「SET TRANSACTION ステートメント」
セクション5.1.7 「サーバーコマンドオプション」
セクション5.1.8 「サーバーシステム変数」

--transactional

セクション23.4.8 「ndb_delete_all — NDB テーブルからのすべての行の削除」

--triggers

セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.6 「mysqldump — データベースバックアッププログラム」
セクション7.4.5.3 「ストアドプログラムのダンプ」

--try-reconnect

セクション23.4.5 「ndb_mgm — NDB Cluster 管理クライアント」

--tupscan

セクション23.4.8 「ndb_delete_all — NDB テーブルからのすべての行の削除」
セクション23.4.24 「ndb_select_all — NDB テーブルの行の出力」

--type

セクション4.6.1 「ibd2sdi — InnoDB テーブルスペース SDI 抽出ユーティリティ」
セクション23.4.7 「ndb_config — NDB Cluster 構成情報の抽出」
セクション23.4.27 「ndb_show_tables — NDB テーブルのリストの表示」

--tz-utc

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

U

[index top]

-U

セクション4.6.4.2 「myisamchk のチェックオプション」

セクション4.5.1.1 「mysql クライアントオプション」

-u

セクション4.6.4.3 「myisamchk の修復オプション」

セクション4.6.5 「myisamlog — MyISAM ログファイルの内容の表示」

セクション2.11 「MySQL のアップグレード」

セクション2.3.4.9 「MySQL インストールのテスト」

セクション4.5.1.1 「mysql クライアントオプション」

セクション4.2.1 「MySQL プログラムの起動」

セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティー」

セクション4.4.2 「mysql_secure_installation — MySQL インストールのセキュリティ改善」

セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」

セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.5 「mysqlexport — データインポートプログラム」

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」

セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」

セクション23.4.9 「ndb_desc — NDB テーブルの表示」

セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

セクション23.4.27 「ndb_show_tables — NDB テーブルのリストの表示」

セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」

セクション2.3.6 「Windows でのインストール後の手順」

セクション6.2.1 「アカウントのユーザー名とパスワード」

セクション4.2.3 「サーバーに接続するためのコマンドオプション」

セクション2.10.3 「サーバーのテスト」

セクション5.1.7 「サーバーコマンドオプション」

--uid

セクション6.3.3.1 「MySQL を使用した SSL および RSA 証明書とキーの作成」

セクション4.4.3 「mysql_ssl_rsa_setup — SSL/RSA ファイルの作成」

--unbuffered

セクション4.5.1.1 「mysql クライアントオプション」

--unpack

セクション16.2.3 「MyISAM テーブルのストレージフォーマット」

セクション4.6.4.3 「myisamchk の修復オプション」

セクション4.6.6 「myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成」

--unqualified

セクション23.4.9 「ndb_desc — NDB テーブルの表示」

セクション23.4.27 「ndb_show_tables — NDB テーブルのリストの表示」

--update

セクション23.4.14 「ndb_index_stat — NDB インデックス統計ユーティリティー」

--update-state

セクション16.2 「MyISAM ストレージエンジン」
セクション7.6.3 「MyISAM テーブルの修復方法」
セクション4.6.4.2 「myisamchk のチェックオプション」

--upgrade

セクション2.11.4 「MySQL 8.0 での変更」
セクション1.3 「MySQL 8.0 の新機能」
セクション2.3.3.4 「MySQL Installer 製品カタログおよびダッシュボード」
セクション2.11.3 「MySQL のアップグレードプロセスの内容」
セクション23.6.4 「NDB Cluster レプリケーションスキーマおよびテーブル」
セクション2.11.6 「Unix/Linux での MySQL バイナリまたはパッケージベースのインストールのアップグレード」
セクション5.1.7 「サーバーコマンドオプション」

--upgrade-system-tables

セクション2.11.3 「MySQL のアップグレードプロセスの内容」
セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
セクション23.6.4 「NDB Cluster レプリケーションスキーマおよびテーブル」

--uri

セクション4.2.5 「URI 類似文字列またはキーと値のペアを使用したサーバーへの接続」

--usage

セクション23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」
セクション23.4.7 「ndb_config — NDB Cluster 構成情報の抽出」
セクション23.4.31 「ndbxfm — NDB Cluster によって作成されたファイルの圧縮、圧縮解除、暗号化、および復号化」

--use-default

セクション4.4.2 「mysql_secure_installation — MySQL インストールのセキュリティー改善」

--use-frm

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

--use-http

セクション23.4.26 「ndb_setup.py — NDB Cluster 用ブラウザベースの Auto-Installer の起動 (非推奨)」

--use-https

セクション23.2.8.1 「NDB Cluster 自動インストーラの要件」
セクション23.4.26 「ndb_setup.py — NDB Cluster 用ブラウザベースの Auto-Installer の起動 (非推奨)」

--use-threads

セクション4.5.5 「mysqlimport — データインポートプログラム」

--useHexFormat

セクション23.4.24 「ndb_select_all — NDB テーブルの行の出力」

--user

セクション6.1.5 「MySQL を通常ユーザーとして実行する方法」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.5.1.3 「mysql クライアントロギング」
セクション4.2.1 「MySQL プログラムの起動」
セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティー」
セクション4.4.2 「mysql_secure_installation — MySQL インストールのセキュリティー改善」
セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

セクション4.5.2 「mysqldadmin — A MySQL Server 管理プログラム」
セクション4.6.8 「mysqlbinlog — バイナリログファイル进行处理するためのユーティリティー」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.3.4 「mysqld_multi — 複数の MySQL サーバーの管理」
セクション4.3.2 「mysqld_safe — MySQL サーバー起動スクリプト」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」
セクション23.5.17.3 「NDB Cluster および MySQL セキュリティー手順」
セクション23.4.28 「ndb_size.pl — NDBCLUSTER サイズ要件エスティメータ」
セクション23.4.29 「ndb_top — NDB スレッドの CPU 使用率情報の表示」
root パスワードのリセット: Unix および Unix- 類似システム
セクション6.2.1 「アカウントのユーザー名とパスワード」
セクション4.2.2.6 「オプションのデフォルト、値を想定するオプション、および = 記号」
セクション4.2.2.2 「オプションファイルの使用」
セクション6.4.4.9 「キーリングキーストア間のキーの移行」
セクション4.2.3 「サーバーに接続するためのコマンドオプション」
セクション2.10.2 「サーバーの起動」
セクション5.1.7 「サーバーコマンドオプション」
セクション6.4.1.9 「ソケットピア資格証明プラグブル認証」
セクション2.10.1 「データディレクトリの初期化」
セクション4.6.8.3 「バイナリログファイルのバックアップのための mysqlbinlog の使用」
セクション7.3 「バックアップおよびリカバリ戦略の例」
セクションB.3.2.16 「ファイルが見つからず同様のエラーが発生しました」
セクション6.4.1.10 「プラグブル認証のテスト」
セクション4.2.2 「プログラムオプションの指定」
元のノードより多くのノードへのリストア
セクション6.1.3 「攻撃者に対する MySQL のセキュアな状態の維持」

user

セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティー」
セクション4.2.2.2 「オプションファイルの使用」

--users

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

V

[index top]

-V

セクション4.4.1 「comp_err — MySQL エラーメッセージファイルのコンパイル」
セクション4.6.2 「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティー」
セクション4.7.2 「my_print_defaults — オプションファイルからのオプションの表示」
セクション4.6.4.1 「myisamchk の一般オプション」
セクション4.6.5 「myisamlog — MyISAM ログファイルの内容の表示」
セクション4.6.6 「myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティー」
セクション4.4.3 「mysql_ssl_rsa_setup — SSL/RSA ファイルの作成」
セクション4.5.2 「mysqldadmin — A MySQL Server 管理プログラム」
セクション4.6.8 「mysqlbinlog — バイナリログファイル进行处理するためのユーティリティー」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」

セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」
セクション23.1.4 「NDB Cluster の新機能」
セクション23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」
セクション23.4.7 「ndb_config — NDB Cluster 構成情報の抽出」
セクション23.4.16 「ndb_perror — NDB エラーメッセージ情報の取得」
セクション23.4.31 「ndbxfm — NDB Cluster によって作成されたファイルの圧縮、圧縮解除、暗号化、および復号化」
セクション4.8.2 「perror — MySQL エラーメッセージ情報の表示」
セクション4.2.2.1 「コマンド行でのオプションの使用」
セクション5.1.7 「サーバーコマンドオプション」

-V

セクション4.6.1 「ibd2sdi — InnoDB テーブルスペース SDI 抽出ユーティリティ」
セクション4.6.2 「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティ」
セクション6.4.4.4 「keyring_okv KMIP プラグインの使用」
セクション4.7.2 「my_print_defaults — オプションファイルからのオプションの表示」
セクション7.6.2 「MyISAM テーブルのエラーのチェック方法」
セクション4.6.3 「myisam_ftdump — 全文インデックス情報の表示」
セクション4.6.4.5 「myisamchk によるテーブル情報の取得」
セクション4.6.4.1 「myisamchk の一般オプション」
セクション4.6.5 「myisamlog — MyISAM ログファイルの内容の表示」
セクション4.6.6 「myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティ」
セクション4.4.3 「mysql_ssl_rsa_setup — SSL/RSA ファイルの作成」
セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」
セクション4.6.8.2 「mysqlbinlog 行イベントの表示」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.6.9 「mysqldumpslow — スロークエリログファイルの要約」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」
セクション23.4.6 「ndb_blob_tool — NDB Cluster テーブルの BLOB および TEXT カラムのチェックおよび修復」
セクション23.4.13 「ndb_import — NDB への CSV データのインポート」
セクション23.4.14 「ndb_index_stat — NDB インデックス統計ユーティリティ」
セクション23.4.4 「ndb_mgmd — NDB Cluster 管理サーバーデーモン」
セクション23.4.16 「ndb_perror — NDB エラーメッセージ情報の取得」
セクション23.4.18 「ndb_print_file — NDB データファイル内容の出力」
セクション23.4.1 「ndbd — NDB Cluster データノードデーモン」
セクション4.8.2 「perror — MySQL エラーメッセージ情報の表示」
セクション4.2.2.1 「コマンド行でのオプションの使用」
セクション5.1.7 「サーバーコマンドオプション」
セクション17.1.6.4 「バイナリロギングのオプションと変数」

--validate-config

セクション5.1.7 「サーバーコマンドオプション」
セクション5.1.3 「サーバー構成の検証」

--validate-password

セクション6.4.3.2 「パスワード検証オプションおよび変数」
セクション6.4.3.3 「パスワード検証コンポーネントへの移行」

--var_name

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション4.6.4.1 「myisamchk の一般オプション」
セクション5.1.7 「サーバーコマンドオプション」

--variable

セクション4.7.1 「mysql_config — クライアントのコンパイル用オプションの表示」

--verbose

セクション4.6.2 「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティ」
セクション4.7.2 「my_print_defaults — オプションファイルからのオプションの表示」
セクション4.6.3 「myisam_ftdump — 全文インデックス情報の表示」
セクション4.6.4.1 「myisamchk の一般オプション」
セクション4.6.6 「myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成」
セクション2.10.2.1 「MySQL Server の起動時の問題のトラブルシューティング」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティ」
セクション4.4.3 「mysql_ssl_rsa_setup — SSL/RSA ファイルの作成」
セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」
セクション4.6.8.2 「mysqlbinlog 行イベントの表示」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.3.4 「mysqld_multi — 複数の MySQL サーバーの管理」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.6.9 「mysqldumpslow — スロークエリログファイルの要約」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」
セクション23.5.1 「NDB Cluster 管理クライアントのコマンド」
セクション23.4.6 「ndb_blob_tool — NDB Cluster テーブルの BLOB および TEXT カラムのチェックおよび修復」
セクション23.4.13 「ndb_import — NDB への CSV データのインポート」
セクション23.4.14 「ndb_index_stat — NDB インデックス統計ユーティリティ」
セクション23.4.4 「ndb_mgmd — NDB Cluster 管理サーバーデーモン」
セクション23.4.15 「ndb_move_data — NDB データコピーユーティリティ」
セクション23.4.16 「ndb_perror — NDB エラーメッセージ情報の取得」
セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」
セクション23.4.1 「ndbd — NDB Cluster データノードデーモン」
セクション4.8.2 「perror — MySQL エラーメッセージ情報の表示」
セクション4.6.4.4 「その他の myisamchk オプション」
セクション4.2.2.2 「オプションファイルの使用」
セクション4.2.2.1 「コマンド行でのオプションの使用」
セクション5.1.7 「サーバーコマンドオプション」
セクション17.2.1.1 「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」
セクション4.5.1.5 「テキストファイルから SQL ステートメントを実行する」
セクション17.1.6.4 「バイナリロギングのオプションと変数」
バイナリログのトランザクション圧縮のモニタリング
セクション17.2.1.2 「行ベースロギングおよびレプリケーションの使用」

--verify-binlog-checksum

セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」

--version

セクション4.4.1 「comp_err — MySQL エラーメッセージファイルのコンパイル」
セクション4.6.1 「ibd2sdi — InnoDB テーブルスペース SDI 抽出ユーティリティ」
セクション4.6.2 「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティ」
セクション4.7.2 「my_print_defaults — オプションファイルからのオプションの表示」
セクション4.6.4.1 「myisamchk の一般オプション」
セクション4.6.6 「myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.7.1 「mysql_config — クライアントのコンパイル用オプションの表示」
セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティ」
セクション4.4.3 「mysql_ssl_rsa_setup — SSL/RSA ファイルの作成」

セクション4.5.2 「mysqldadmin — A MySQL Server 管理プログラム」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.3.4 「mysqld_multi — 複数の MySQL サーバーの管理」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション4.5.6 「mysqldump — データベースバックアッププログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」
セクション23.4.32 「NDB Cluster プログラムに共通のオプション — NDB Cluster プログラムに共通のオプション」
セクション23.4.7 「ndb_config — NDB Cluster 構成情報の抽出」
セクション23.4.16 「ndb_perror — NDB エラーメッセージ情報の取得」
セクション23.4.31 「ndbxfm — NDB Cluster によって作成されたファイルの圧縮、圧縮解除、暗号化、および復号化」
セクション4.8.2 「 perror — MySQL エラーメッセージ情報の表示」
セクション4.2.2.1 「コマンド行でのオプションの使用」
セクション5.1.7 「サーバーコマンドオプション」

--version-check

セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

--vertical

セクション4.5.1.1 「mysql クライアントオプション」
セクション4.5.2 「mysqldadmin — A MySQL Server 管理プログラム」
セクション1.6 「質問またはバグをレポートする方法」

W

[[index top](#)]

-W

セクション4.5.1.1 「mysql クライアントオプション」
セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
セクション4.5.2 「mysqldadmin — A MySQL Server 管理プログラム」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」
セクション4.2.3 「サーバーに接続するためのコマンドオプション」

-W

セクション4.6.2 「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティ」
セクション4.6.4.1 「myisamchk の一般オプション」
セクション4.6.5 「myisamlog — MyISAM ログファイルの内容の表示」
セクション4.6.6 「myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティ」
セクション4.5.2 「mysqldadmin — A MySQL Server 管理プログラム」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション23.4.30 「ndb_waiter — NDB Cluster が特定のステータスに達するまで待機」

--wait

セクション4.6.4.1 「myisamchk の一般オプション」
セクション4.6.6 「myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.5.2 「mysqldadmin — A MySQL Server 管理プログラム」

--wait-nodes

セクション23.4.30 「ndb_waiter — NDB Cluster が特定のステータスに達するまで待機」

--warn

セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティー」

--watch-progress

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

--where

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

WITH_ANT

セクション2.9.7 「MySQL ソース構成オプション」

WITH_BOOST

セクション2.9.7 「MySQL ソース構成オプション」

セクション2.9.2 「ソースインストールの前提条件」

WITH_BUNDLED_MEMCACHED

セクション2.9.7 「MySQL ソース構成オプション」

WITH_CLASSPATH

セクション23.2.1.4 「Linux でのソースからの NDB Cluster の構築」

セクション23.2.2.2 「Windows でのソースからの NDB Cluster のコンパイルとインストール」

WITH_CLIENT_PROTOCOL_TRACING

セクション2.9.7 「MySQL ソース構成オプション」

WITH_CURL

セクション2.9.7 「MySQL ソース構成オプション」

WITH_DEBUG

セクション4.6.1 「ibd2sdi — InnoDB テーブルスペース SDI 抽出ユーティリティー」

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

セクション4.6.4.1 「myisamchk の一般オプション」

セクション4.6.6 「myisampack — 圧縮された読み取り専用の MyISAM テーブルの生成」

セクション4.5.1.1 「mysql クライアントオプション」

セクション2.9.7 「MySQL ソース構成オプション」

セクション4.6.7 「mysql_config_editor — MySQL 構成ユーティリティー」

セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」

セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.6.9 「mysqldumpslow — スロークエリーログファイルの要約」

セクション4.5.5 「mysqlimport — データインポートプログラム」

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」

セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」

WITH_EDITLINE

セクション2.9.7 「MySQL ソース構成オプション」

WITH_EMBEDDED_SERVER

セクション1.3 「MySQL 8.0 の新機能」

WITH_EMBEDDED_SHARED_LIBRARY

[セクション1.3「MySQL 8.0の新機能」](#)

WITH_GMOCK

[セクション2.9.7「MySQL ソース構成オプション」](#)

WITH_ICU

[セクション2.9.7「MySQL ソース構成オプション」](#)

WITH_JEMALLOC

[セクション2.9.7「MySQL ソース構成オプション」](#)

WITH_LIBEVENT

[セクション2.9.7「MySQL ソース構成オプション」](#)

WITH_LOCK_ORDER

[セクション5.9.3「LOCK_ORDER ツール」](#)
[セクション2.9.7「MySQL ソース構成オプション」](#)

WITH_LTO

[セクション2.9.7「MySQL ソース構成オプション」](#)

WITH_LZ4

[セクション2.9.7「MySQL ソース構成オプション」](#)

WITH_LZMA

[セクション2.9.7「MySQL ソース構成オプション」](#)

WITH_MECAB

[セクション12.10.9「MeCab フルテキストパーサープラグイン」](#)

WITH_NDB_JAVA

[セクション23.2.1.4「Linux でのソースからの NDB Cluster の構築」](#)
[セクション23.2.2.2「Windows でのソースからの NDB Cluster のコンパイルとインストール」](#)

WITH_NDBCLUSTER

[セクション23.2.1.4「Linux でのソースからの NDB Cluster の構築」](#)
[セクション2.9.7「MySQL ソース構成オプション」](#)
[セクション23.2.2.2「Windows でのソースからの NDB Cluster のコンパイルとインストール」](#)

WITH_NDBCLUSTER_STORAGE_ENGINE

[セクション23.2.1.4「Linux でのソースからの NDB Cluster の構築」](#)
[セクション23.2.2.2「Windows でのソースからの NDB Cluster のコンパイルとインストール」](#)

WITH_NUMA

[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)
[セクション2.9.7「MySQL ソース構成オプション」](#)

WITH_PLUGIN_NDBCLUSTER

[セクション23.2.1.4「Linux でのソースからの NDB Cluster の構築」](#)
[セクション23.2.2.2「Windows でのソースからの NDB Cluster のコンパイルとインストール」](#)

WITH_PROTOBUF

[セクション2.9.7「MySQL ソース構成オプション」](#)

WITH_RE2

セクション2.9.7「MySQL ソース構成オプション」

WITH_SSL

セクション2.9.7「MySQL ソース構成オプション」
セクション2.9.6「SSL ライブラリサポートの構成」
セクション2.9.2「ソースインストールの前提条件」

WITH_SYSTEMD

セクション2.9.7「MySQL ソース構成オプション」

WITH_TCMALLOC

セクション2.9.7「MySQL ソース構成オプション」

WITH_TEST_TRACE_PLUGIN

セクション2.9.7「MySQL ソース構成オプション」

WITH_ZLIB

セクション2.9.7「MySQL ソース構成オプション」

WITH_ZSTD

セクション2.9.7「MySQL ソース構成オプション」

--write

セクション4.6.2「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティ」

--write-binlog

セクション17.1.3.7「GTID ベースレプリケーションの制約」
セクション4.4.5「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
セクション4.5.3「mysqlcheck — テーブル保守プログラム」
セクション17.5.3「レプリケーションセットアップをアップグレードする」

X

[[index top](#)]

-X

セクション4.5.1.1「mysql クライアントオプション」
セクション4.5.1.2「mysql クライアントコマンド」
セクション4.5.4「mysqldump — データベースバックアッププログラム」

-X

セクション4.5.4「mysqldump — データベースバックアッププログラム」
セクション4.5.8「mysqslap — ロードエミュレーションクライアント」
セクション23.1.4「NDB Cluster の新機能」
セクション23.4.9「ndb_desc — NDB テーブルの表示」
セクション23.4.24「ndb_select_all — NDB テーブルの行の出力」
セクション23.4.29「ndb_top — NDB スレッドの CPU 使用率情報の表示」

--xml

セクション13.2.8「LOAD XML ステートメント」
セクション4.5.1.1「mysql クライアントオプション」
セクション4.5.4「mysqldump — データベースバックアッププログラム」
セクション23.4.7「ndb_config — NDB Cluster 構成情報の抽出」
セクション12.12「XML 関数」

Y

[\[index top\]](#)

-Y

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

-y

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」

Z

[\[index top\]](#)

-Z

セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

セクション23.4.24 「ndb_select_all — NDB テーブルの行の出力」

--zstd-compression-level

セクション4.5.1.1 「mysql クライアントオプション」

セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」

セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」

セクション4.5.3 「mysqlcheck — テーブル保守プログラム」

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.5 「mysqlimport — データインポートプログラム」

セクション4.5.6 「mysqmpump — データベースバックアッププログラム」

セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」

セクション4.5.8 「mysqslap — ロードエミュレーションクライアント」

セクション4.2.3 「サーバーに接続するためのコマンドオプション」

セクション4.2.8 「接続圧縮制御」

権限の索引

[A](#) | [B](#) | [C](#) | [D](#) | [E](#) | [F](#) | [G](#) | [H](#) | [I](#) | [L](#) | [N](#) | [P](#) | [R](#) | [S](#) | [T](#) | [U](#) | [V](#) | [X](#)

A

[\[index top\]](#)

ALL

セクション13.7.1.6 「GRANT ステートメント」

セクション6.2.2 「MySQL で提供される権限」

ALL PRIVILEGES

セクション6.2.2 「MySQL で提供される権限」

セクション13.7.7.21 「SHOW GRANTS ステートメント」

ALTER

セクション13.1.2 「ALTER DATABASE ステートメント」

セクション13.1.9 「ALTER TABLE ステートメント」

セクション13.7.1.6 「GRANT ステートメント」

セクション6.2.2 「MySQL で提供される権限」

セクション13.1.36 「RENAME TABLE ステートメント」

[セクション24.3.3「パーティションとサブパーティションをテーブルと交換する」](#)

ALTER ROUTINE

[セクション13.1.4「ALTER FUNCTION ステートメント」](#)
[セクション13.1.7「ALTER PROCEDURE ステートメント」](#)
[セクション13.1.17「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」](#)
[セクション13.1.29「DROP PROCEDURE および DROP FUNCTION ステートメント」](#)
[セクション13.7.1.6「GRANT ステートメント」](#)
[セクション26.30「INFORMATION_SCHEMA ROUTINES テーブル」](#)
[セクション6.2.2「MySQL で提供される権限」](#)
[セクション13.7.7.9「SHOW CREATE PROCEDURE ステートメント」](#)
[セクション13.7.7.28「SHOW PROCEDURE STATUS ステートメント」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション25.7「ストアードプログラムバイナリロギング」](#)
[セクション25.2.2「ストアードルーチンと MySQL 権限」](#)
[セクション17.1.6.4「バイナリロギングのオプションと変数」](#)

APPLICATION_PASSWORD_ADMIN

[セクション13.7.1.1「ALTER USER ステートメント」](#)
[セクション13.7.1.6「GRANT ステートメント」](#)
[セクション6.2.2「MySQL で提供される権限」](#)
[セクション13.7.1.10「SET PASSWORD ステートメント」](#)
[セクション6.2.15「パスワード管理」](#)

AUDIT_ADMIN

[セクション13.7.1.6「GRANT ステートメント」](#)
[セクション6.4.5.1「MySQL Enterprise Audit の要素」](#)
[セクション6.2.2「MySQL で提供される権限」](#)
[セクション6.4.5.10「監査ログ参照」](#)

B

[\[index top\]](#)

BACKUP_ADMIN

[セクション13.7.1.6「GRANT ステートメント」](#)
[セクション13.3.5「LOCK INSTANCE FOR BACKUP および UNLOCK INSTANCE ステートメント」](#)
[セクション2.11.4「MySQL 8.0 での変更」](#)
[セクション1.3「MySQL 8.0 の新機能」](#)
[セクション6.2.2「MySQL で提供される権限」](#)
[クローニングの前提条件](#)
[セクション5.6.7.3「リモートデータのクローニング」](#)
[セクション5.6.7.2「ローカルでのデータのクローニング」](#)
[セクション18.4.3.2「分散リカバリのためのクローニング」](#)
[セクション18.2.1.3「分散リカバリのユーザー資格証明」](#)

BINLOG_ADMIN

[セクション13.7.8.1「BINLOG ステートメント」](#)
[セクション13.7.1.6「GRANT ステートメント」](#)
[セクション6.2.2「MySQL で提供される権限」](#)
[セクション4.6.8「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」](#)
[セクション13.4.1.1「PURGE BINARY LOGS ステートメント」](#)
[セクション6.2.12「部分取消しを使用した権限の制限」](#)

BINLOG_ENCRYPTION_ADMIN

[セクション13.1.5「ALTER INSTANCE ステートメント」](#)
[セクション13.7.1.6「GRANT ステートメント」](#)

[セクション6.2.2「MySQL で提供される権限」](#)
[セクション17.3.2「バイナリログファイルとリレーログファイルの暗号化」](#)
[セクション17.3.2.3「バイナリログマスターキーのローテーション」](#)

C

[\[index top\]](#)

CLONE_ADMIN

[セクション13.7.1.6「GRANT ステートメント」](#)
[セクション6.2.2「MySQL で提供される権限」](#)
[クローニングの前提条件](#)
[セクション5.6.7.12「クローンシステム変数」](#)
[セクション5.6.7.3「リモートデータのクローニング」](#)

CONNECTION_ADMIN

[セクション13.1.5「ALTER INSTANCE ステートメント」](#)
[セクション13.7.1.1「ALTER USER ステートメント」](#)
[セクション13.7.1.2「CREATE ROLE ステートメント」](#)
[セクション13.7.1.3「CREATE USER ステートメント」](#)
[セクション13.7.1.4「DROP ROLE ステートメント」](#)
[セクション13.7.1.5「DROP USER ステートメント」](#)
[セクション13.7.1.6「GRANT ステートメント」](#)
[セクション13.7.8.4「KILL ステートメント」](#)
[セクション6.2.2「MySQL で提供される権限」](#)
[セクション4.5.2「mysqldadmin — A MySQL Server 管理プログラム」](#)
[セクション13.7.1.7「RENAME USER ステートメント」](#)
[セクション13.7.1.8「REVOKE ステートメント」](#)
[セクション13.7.1.10「SET PASSWORD ステートメント」](#)
[セクション13.3.7「SET TRANSACTION ステートメント」](#)
[セクション13.7.7.29「SHOW PROCESSLIST ステートメント」](#)
[セクション13.3.1「START TRANSACTION、COMMIT および ROLLBACK ステートメント」](#)
[セクション6.2.11「アカウントカテゴリ」](#)
[セクション6.2.14「アカウントパスワードの割り当て」](#)
[セクション13.7.1「アカウント管理ステートメント」](#)
[セクション10.5「アプリケーションの文字セットおよび照合順序の構成」](#)
[セクション18.8「グループレプリケーションシステム変数」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクションB.3.2.5「接続が多すぎます」](#)
[セクション5.1.12.1「接続インターフェース」](#)
[セクション6.1.3「攻撃者に対する MySQL のセキュアな状態の維持」](#)
[セクション6.3.1「暗号化接続を使用するための MySQL の構成」](#)
[セクション5.1.12.2「管理接続管理」](#)
[セクション18.6.6.4「終了処理」](#)

CREATE

[セクション13.1.9「ALTER TABLE ステートメント」](#)
[セクション13.1.12「CREATE DATABASE ステートメント」](#)
[セクション13.1.20「CREATE TABLE ステートメント」](#)
[セクション13.7.1.6「GRANT ステートメント」](#)
[セクション13.2.5「IMPORT TABLE ステートメント」](#)
[セクション6.2.2「MySQL で提供される権限」](#)
[セクション13.1.36「RENAME TABLE ステートメント」](#)
[セクション24.3.3「パーティションとサブパーティションをテーブルと交換する」](#)

CREATE ROLE

[セクション13.7.1.2「CREATE ROLE ステートメント」](#)
[セクション13.7.1.6「GRANT ステートメント」](#)

セクション6.2.2 「MySQL で提供される権限」
セクション17.3.3.1 「レプリケーション PRIVILEGE_CHECKS_USER アカountの権限」
セクション6.2.10 「ロールの使用」

CREATE ROUTINE

セクション13.1.17 「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」
セクション13.7.1.6 「GRANT ステートメント」
セクション26.30 「INFORMATION_SCHEMA ROUTINES テーブル」
セクションA.4 「MySQL 8.0 FAQ: ストアドプロシージャおよびストアドファンクション」
セクション6.2.2 「MySQL で提供される権限」
セクション13.7.7.9 「SHOW CREATE PROCEDURE ステートメント」
セクション13.7.7.28 「SHOW PROCEDURE STATUS ステートメント」
セクション25.7 「ストアドプログラムバイナリロギング」
セクション25.2.2 「ストアドルーチンと MySQL 権限」
セクション17.1.6.4 「バイナリロギングのオプションと変数」

CREATE TABLESPACE

セクション13.1.10 「ALTER TABLESPACE ステートメント」
セクション13.7.1.6 「GRANT ステートメント」
セクション15.13 「InnoDB 保存データ暗号化」
セクション6.2.2 「MySQL で提供される権限」
セクション15.6.3.3 「一般テーブルスペース」

CREATE TEMPORARY TABLES

セクション13.1.20.2 「CREATE TEMPORARY TABLE ステートメント」
セクション13.7.1.6 「GRANT ステートメント」
セクション6.2.2 「MySQL で提供される権限」

CREATE USER

セクション13.7.1.1 「ALTER USER ステートメント」
セクション13.7.1.2 「CREATE ROLE ステートメント」
セクション13.7.1.3 「CREATE USER ステートメント」
セクション13.7.1.4 「DROP ROLE ステートメント」
セクション13.7.1.5 「DROP USER ステートメント」
セクション13.7.1.6 「GRANT ステートメント」
セクション26.46 「INFORMATION_SCHEMA USER_ATTRIBUTES テーブル」
セクション6.2.2 「MySQL で提供される権限」
セクション13.7.1.7 「RENAME USER ステートメント」
セクション13.7.1.8 「REVOKE ステートメント」
セクション13.7.1.9 「SET DEFAULT ROLE ステートメント」
セクション13.7.1.10 「SET PASSWORD ステートメント」
セクション6.2.8 「アカウントの追加、権限の割当ておよびアカウントの削除」
セクション6.2.11 「アカウントカテゴリ」
セクション6.2.14 「アカウントパスワードの割り当て」
セクション6.2.15 「パスワード管理」
セクション17.3.3.1 「レプリケーション PRIVILEGE_CHECKS_USER アカountの権限」
セクション6.2.10 「ロールの使用」

CREATE VIEW

セクション13.1.11 「ALTER VIEW ステートメント」
セクション13.1.23 「CREATE VIEW ステートメント」
セクション13.7.1.6 「GRANT ステートメント」
セクション6.2.2 「MySQL で提供される権限」
セクション25.9 「ビューの制約」

D

[[index top](#)]

DELETE

[セクション13.1.20「CREATE TABLE ステートメント」](#)
[セクション13.2.2「DELETE ステートメント」](#)
[セクション13.7.1.5「DROP USER ステートメント」](#)
[セクション13.7.1.6「GRANT ステートメント」](#)
[セクション16.7「MERGE ストレージエンジン」](#)
[セクション6.2.2「MySQL で提供される権限」](#)
[セクション23.5.17.2「NDB Cluster および MySQL の権限」](#)
[セクション13.2.9「REPLACE ステートメント」](#)
[セクション27.12.2.4「setup_objects テーブル」](#)
[セクション13.7.4.5「UNINSTALL COMPONENT ステートメント」](#)
[セクション13.7.4.6「UNINSTALL PLUGIN ステートメント」](#)
[セクション6.2.7「アクセス制御、ステージ 2: リクエストの確認」](#)
[セクション5.6.1「プラグインのインストールおよびアンインストール」](#)
[セクション13.7.4.2「ユーザー定義関数に対する DROP FUNCTION ステートメント」](#)
[セクション5.7.1「ユーザー定義関数のインストールおよびアンインストール」](#)
[セクション17.3.3.1「レプリケーション PRIVILEGE_CHECKS_USER アカウントの権限」](#)
[セクション6.2.10「ロールの使用」](#)
[セクション15.7.2.4「読取りのロック」](#)
[セクション6.2.12「部分取消しを使用した権限の制限」](#)

DROP

[セクション13.1.9「ALTER TABLE ステートメント」](#)
[セクション13.1.11「ALTER VIEW ステートメント」](#)
[セクション13.1.23「CREATE VIEW ステートメント」](#)
[セクション5.1.12.3「DNS ルックアップとホストキャッシュ」](#)
[セクション13.1.24「DROP DATABASE ステートメント」](#)
[セクション13.1.32「DROP TABLE ステートメント」](#)
[セクション13.1.35「DROP VIEW ステートメント」](#)
[セクション13.7.8.3「FLUSH ステートメント」](#)
[セクション13.7.1.6「GRANT ステートメント」](#)
[セクション27.12.19.5「host_cache テーブル」](#)
[セクション1.3「MySQL 8.0 の新機能」](#)
[セクション6.6.1「MySQL Enterprise Encryption のインストール」](#)
[セクション6.2.2「MySQL で提供される権限」](#)
[セクション24.3.1「RANGE および LIST パーティションの管理」](#)
[セクション13.1.36「RENAME TABLE ステートメント」](#)
[セクション13.1.37「TRUNCATE TABLE ステートメント」](#)
[セクション6.2「アクセス制御とアカウント管理」](#)
[セクション27.11「パフォーマンススキーマの一般的なテーブル特性」](#)
[セクション24.3.3「パーティションとサブパーティションをテーブルと交換する」](#)

DROP ROLE

[セクション13.7.1.4「DROP ROLE ステートメント」](#)
[セクション13.7.1.6「GRANT ステートメント」](#)
[セクション6.2.2「MySQL で提供される権限」](#)
[セクション17.3.3.1「レプリケーション PRIVILEGE_CHECKS_USER アカウントの権限」](#)
[セクション6.2.10「ロールの使用」](#)

E

[\[index top\]](#)

ENCRYPTION_KEY_ADMIN

[セクション13.1.5「ALTER INSTANCE ステートメント」](#)
[セクション13.7.1.6「GRANT ステートメント」](#)
[セクション15.13「InnoDB 保存データ暗号化」](#)
[セクション6.2.2「MySQL で提供される権限」](#)

セクション6.4.4.9 「キーリングキーストア間のキーの移行」
セクション6.4.4.13 「キーリングシステム変数」

EVENT

セクション13.1.3 「ALTER EVENT ステートメント」
セクション13.1.13 「CREATE EVENT ステートメント」
セクション13.1.25 「DROP EVENT ステートメント」
セクション13.7.1.6 「GRANT ステートメント」
セクション6.2.2 「MySQL で提供される権限」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション13.7.7.7 「SHOW CREATE EVENT ステートメント」
セクション13.7.7.18 「SHOW EVENTS ステートメント」
セクション25.4.6 「イベントスケジューラと MySQL 権限」
セクション25.4.1 「イベントスケジューラの概要」
セクション25.4.3 「イベント構文」

EXECUTE

セクション13.1.17 「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」
セクション13.1.29 「DROP PROCEDURE および DROP FUNCTION ステートメント」
セクション13.7.1.6 「GRANT ステートメント」
セクション26.30 「INFORMATION_SCHEMA ROUTINES テーブル」
セクション6.4.7.3 「MySQL Enterprise Firewall の使用」
セクション6.2.2 「MySQL で提供される権限」
セクション13.7.7.9 「SHOW CREATE PROCEDURE ステートメント」
セクション13.7.7.28 「SHOW PROCEDURE STATUS ステートメント」
セクション28.1 「sys スキーマを使用するための前提条件」
セクション5.6.7.9 「クローニング操作の監視」
セクション5.1.8 「サーバーシステム変数」
セクション25.6 「ストアオブジェクトのアクセス制御」
セクション25.2.2 「ストアルーチンと MySQL 権限」
セクション6.4.4.10 「汎用キーリングキー管理関数」

F

[[index top](#)]

FILE

セクション11.3.4 「BLOB 型と TEXT 型」
セクション13.1.20 「CREATE TABLE ステートメント」
セクション13.7.1.6 「GRANT ステートメント」
セクション13.2.5 「IMPORT TABLE ステートメント」
セクション13.2.7 「LOAD DATA ステートメント」
セクション13.2.8 「LOAD XML ステートメント」
セクション1.3 「MySQL 8.0 の新機能」
セクション6.6.2 「MySQL Enterprise Encryption の使用方法と例」
セクション6.2.2 「MySQL で提供される権限」
セクション6.2.21 「MySQL への接続の問題のトラブルシューティング」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション7.4.3 「mysqldump による区切りテキストフォーマットでのデータのダンプ」
セクション13.2.10.1 「SELECT ... INTO ステートメント」
セクション5.1.8 「サーバーシステム変数」
セクション17.3.3.1 「レプリケーション PRIVILEGE_CHECKS_USER アカウントの権限」
セクション17.5.1.19 「レプリケーションと LOAD DATA」
セクション6.2.3 「付与テーブル」
セクション6.1.3 「攻撃者に対する MySQL のセキュアな状態の維持」
セクション12.8 「文字列関数および演算子」
セクション6.2.12 「部分取消しを使用した権限の制限」

FIREWALL_ADMIN

[セクション13.7.1.6 「GRANT ステートメント」](#)
[セクション6.4.7.3 「MySQL Enterprise Firewall の使用」](#)
[セクション6.4.7.1 「MySQL Enterprise Firewall の要素」](#)
[セクション6.4.7.4 「MySQL Enterprise Firewall リファレンス」](#)
[セクション6.2.2 「MySQL で提供される権限」](#)

FIREWALL_USER

[セクション13.7.1.6 「GRANT ステートメント」](#)
[セクション6.4.7.3 「MySQL Enterprise Firewall の使用」](#)
[セクション6.4.7.1 「MySQL Enterprise Firewall の要素」](#)
[セクション6.2.2 「MySQL で提供される権限」](#)

FLUSH_OPTIMIZER_COSTS

[セクション13.7.8.3 「FLUSH ステートメント」](#)
[セクション13.7.1.6 「GRANT ステートメント」](#)
[セクション6.2.2 「MySQL で提供される権限」](#)

FLUSH_STATUS

[セクション13.7.8.3 「FLUSH ステートメント」](#)
[セクション13.7.1.6 「GRANT ステートメント」](#)
[セクション6.2.2 「MySQL で提供される権限」](#)

FLUSH_TABLES

[セクション13.7.8.3 「FLUSH ステートメント」](#)
[セクション13.7.1.6 「GRANT ステートメント」](#)
[セクション6.2.2 「MySQL で提供される権限」](#)

FLUSH_USER_RESOURCES

[セクション13.7.8.3 「FLUSH ステートメント」](#)
[セクション13.7.1.6 「GRANT ステートメント」](#)
[セクション6.2.2 「MySQL で提供される権限」](#)

G

[\[index top\]](#)

GRANT OPTION

[セクション13.7.1.6 「GRANT ステートメント」](#)
[セクション26.10 「INFORMATION_SCHEMA COLUMN_PRIVILEGES テーブル」](#)
[セクション26.33 「INFORMATION_SCHEMA SCHEMA_PRIVILEGES テーブル」](#)
[セクション26.44 「INFORMATION_SCHEMA TABLE_PRIVILEGES テーブル」](#)
[セクション26.47 「INFORMATION_SCHEMA USER_PRIVILEGES テーブル」](#)
[セクション6.2.2 「MySQL で提供される権限」](#)
[セクション13.7.1.8 「REVOKE ステートメント」](#)
[セクション13.7.7.21 「SHOW GRANTS ステートメント」](#)
[セクション17.3.3.1 「レプリケーション PRIVILEGE_CHECKS_USER アカウントの権限」](#)

GROUP_REPLICATION_ADMIN

[セクション13.7.1.6 「GRANT ステートメント」](#)
[セクション6.2.2 「MySQL で提供される権限」](#)
[セクション13.4.3.1 「START GROUP_REPLICATION ステートメント」](#)
[セクション13.4.3.2 「STOP GROUP_REPLICATION ステートメント」](#)
[セクション13.4.3.5 「グループの最大コンセンサスインスタンスを検査および構成する関数」](#)
[セクション18.4.1.3 「グループレプリケーショングループ書込みコンセンサスの使用」](#)
[セクション18.8 「グループレプリケーションシステム変数」](#)

[セクション13.4.3.6 「グループレプリケーション通信プロトコルのバージョンを検査および設定する関数」](#)
[セクション18.4.1.4 「グループ通信プロトコルバージョンの設定」](#)

I

[\[index top\]](#)

INDEX

[セクション13.1.9 「ALTER TABLE ステートメント」](#)
[セクション13.7.1.6 「GRANT ステートメント」](#)
[セクション6.2.2 「MySQL で提供される権限」](#)

INNODB_REDO_LOG_ARCHIVE

[セクション13.7.1.6 「GRANT ステートメント」](#)
[セクション6.2.2 「MySQL で提供される権限」](#)
[セクション15.6.5 「redo ログ」](#)

INNODB_REDO_LOG_ENABLE

[セクション13.7.1.6 「GRANT ステートメント」](#)
[セクション1.3 「MySQL 8.0 の新機能」](#)
[セクション6.2.2 「MySQL で提供される権限」](#)
[セクション15.6.5 「redo ログ」](#)

INSERT

[セクション13.1.9 「ALTER TABLE ステートメント」](#)
[セクション13.7.3.1 「ANALYZE TABLE ステートメント」](#)
[セクション13.7.1.3 「CREATE USER ステートメント」](#)
[セクション13.1.23 「CREATE VIEW ステートメント」](#)
[セクション13.7.1.6 「GRANT ステートメント」](#)
[セクション13.2.6 「INSERT ステートメント」](#)
[セクション13.7.4.3 「INSTALL COMPONENT ステートメント」](#)
[セクション13.7.4.4 「INSTALL PLUGIN ステートメント」](#)
[セクション6.6.1 「MySQL Enterprise Encryption のインストール」](#)
[セクション6.2.2 「MySQL で提供される権限」](#)
[セクション13.7.3.4 「OPTIMIZE TABLE ステートメント」](#)
[セクション13.1.36 「RENAME TABLE ステートメント」](#)
[セクション13.7.3.5 「REPAIR TABLE ステートメント」](#)
[セクション13.2.9 「REPLACE ステートメント」](#)
[セクション27.12.2.4 「setup_objects テーブル」](#)
[セクション28.1 「sys スキーマを使用するための前提条件」](#)
[セクション6.2.14 「アカウントパスワードの割り当て」](#)
[セクション6.2.7 「アクセス制御、ステージ 2: リクエストの確認」](#)
[セクション25.4.6 「イベントスケジューラと MySQL 権限」](#)
[セクション5.1.7 「サーバーコマンドオプション」](#)
[セクション25.6 「ストアードオブジェクトのアクセス制御」](#)
[セクション24.3.3 「パーティションとサブパーティションをテーブルと交換する」](#)
[セクション16.11.1 「プラグブルストレージエンジンのアーキテクチャー」](#)
[セクション5.6.1 「プラグインのインストールおよびアンインストール」](#)
[セクション5.7.1 「ユーザー定義関数のインストールおよびアンインストール」](#)
[セクション13.7.4.1 「ユーザー定義関数用の CREATE FUNCTION ステートメント」](#)
[セクション17.3.3.1 「レプリケーション PRIVILEGE_CHECKS_USER アカウントの権限」](#)
[セクション17.3.3 「レプリケーション権限チェック」](#)
[セクション6.2.10 「ロールの使用」](#)
[セクション6.2.12 「部分取消しを使用した権限の制限」](#)

L

[\[index top\]](#)

LOCK TABLES

セクション13.7.8.3 「FLUSH ステートメント」
セクション13.7.1.6 「GRANT ステートメント」
セクション13.3.6 「LOCK TABLES および UNLOCK TABLES ステートメント」
セクション6.2.2 「MySQL で提供される権限」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.6 「mysqmpump — データベースバックアッププログラム」
セクション15.7.2.4 「読取りのロック」

N

[\[index top\]](#)

NDB_STORED_USER

セクション13.7.1.6 「GRANT ステートメント」
セクションA.10 「MySQL 8.0 FAQ: NDB Cluster」
セクション6.2.2 「MySQL で提供される権限」
セクション23.1.4 「NDB Cluster の新機能」
NDB バックアップを以前のバージョンの NDB Cluster に復元
セクション23.5.12 「NDB_STORED_USER での分散 MySQL 権限」

P

[\[index top\]](#)

PERSIST_RO_VARIABLES_ADMIN

セクション13.7.1.6 「GRANT ステートメント」
セクション6.2.2 「MySQL で提供される権限」
セクション13.7.8.7 「RESET PERSIST ステートメント」
セクション5.1.9.1 「システム変数権限」

PROCESS

セクション13.8.2 「EXPLAIN ステートメント」
セクション13.7.1.6 「GRANT ステートメント」
セクション26.15 「INFORMATION_SCHEMA FILES テーブル」
セクション26.51.1 「INFORMATION_SCHEMA INNODB_BUFFER_PAGE テーブル」
セクション26.51.2 「INFORMATION_SCHEMA INNODB_BUFFER_PAGE_LRU テーブル」
セクション26.51.3 「INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS テーブル」
セクション26.51.4 「INFORMATION_SCHEMA INNODB_CACHED_INDEXES テーブル」
セクション26.51.5 「INFORMATION_SCHEMA INNODB_CMP および INNODB_CMP_RESET テーブル」
セクション26.51.7 「INFORMATION_SCHEMA INNODB_CMP_PER_INDEX および INNODB_CMP_PER_INDEX_RESET テーブル」
セクション26.51.6 「INFORMATION_SCHEMA INNODB_CMPMEM および INNODB_CMPMEM_RESET テーブル」
セクション26.51.8 「INFORMATION_SCHEMA INNODB_COLUMNS テーブル」
セクション26.51.9 「INFORMATION_SCHEMA INNODB_DATAFILES テーブル」
セクション26.51.10 「INFORMATION_SCHEMA INNODB_FIELDS テーブル」
セクション26.51.11 「INFORMATION_SCHEMA INNODB_FOREIGN テーブル」
セクション26.51.12 「INFORMATION_SCHEMA INNODB_FOREIGN_COLS テーブル」
セクション26.51.13 「INFORMATION_SCHEMA INNODB_FT_BEING_DELETED テーブル」
セクション26.51.14 「INFORMATION_SCHEMA INNODB_FT_CONFIG テーブル」
セクション26.51.15 「INFORMATION_SCHEMA INNODB_FT_DEFAULT_STOPWORD テーブル」
セクション26.51.16 「INFORMATION_SCHEMA INNODB_FT_DELETED テーブル」
セクション26.51.17 「INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE テーブル」
セクション26.51.18 「INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE テーブル」
セクション26.51.19 「INFORMATION_SCHEMA INNODB_INDEXES テーブル」
セクション26.51.21 「INFORMATION_SCHEMA INNODB_LOCK_WAITS テーブル」
セクション26.51.20 「INFORMATION_SCHEMA INNODB_LOCKS テーブル」

セクション26.51.22 「INFORMATION_SCHEMA INNODB_METRICS テーブル」
セクション26.51.23 「INFORMATION_SCHEMA INNODB_SESSION_TEMP_TABLESPACES テーブル」
セクション26.51.24 「INFORMATION_SCHEMA INNODB_TABLES テーブル」
セクション26.51.25 「INFORMATION_SCHEMA INNODB_TABLESPACES テーブル」
セクション26.51.26 「INFORMATION_SCHEMA INNODB_TABLESPACES_BRIEF テーブル」
セクション26.51.27 「INFORMATION_SCHEMA INNODB_TABLESTATS ビュー」
セクション26.51.28 「INFORMATION_SCHEMA INNODB_TEMP_TABLE_INFO テーブル」
セクション26.51.29 「INFORMATION_SCHEMA INNODB_TRX テーブル」
セクション26.51.30 「INFORMATION_SCHEMA INNODB_VIRTUAL テーブル」
セクション26.23 「INFORMATION_SCHEMA PROCESSLIST テーブル」
セクション15.17.2 「InnoDB モニターの有効化」
セクション13.7.8.4 「KILL ステートメント」
セクション6.2.2 「MySQL で提供される権限」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション23.5.9 「NDB Cluster での MySQL Server の使用」
セクション27.12.19.9 「processlist テーブル」
セクション13.7.7.15 「SHOW ENGINE ステートメント」
セクション13.7.7.29 「SHOW PROCESSLIST ステートメント」
セクション28.1 「sys スキーマを使用するための前提条件」
セクション26.1 「はじめに」
セクション6.2.8 「アカウントの追加、権限の割当ておよびアカウントの削除」
セクション25.4.2 「イベントスケジューラの構成」
セクション27.12.19.10 「スレッドテーブル」
セクション8.14.1 「プロセスリストへのアクセス」
セクション8.8.4 「名前付き接続の実行計画情報の取得」
セクション6.1.3 「攻撃者に対する MySQL のセキュアな状態の維持」

PROXY

セクション13.7.1.6 「GRANT ステートメント」
セクション27.12.19.5 「host_cache テーブル」
セクション6.4.1.7 「LDAP プラガブル認証」
セクション6.2.2 「MySQL で提供される権限」
セクション6.4.1.5 「PAM プラガブル認証」
セクション6.4.1.6 「Windows プラガブル認証」
セクション6.2.18 「プロキシユーザー」
セクション6.2.3 「付与テーブル」
セクション2.10.4 「初期 MySQL アカウントの保護」

PROXY ... WITH GRANT OPTION

セクション6.2.18 「プロキシユーザー」

R

[\[index top\]](#)

REFERENCES

セクション13.1.20.5 「FOREIGN KEY の制約」
セクション13.7.1.6 「GRANT ステートメント」
セクション6.2.2 「MySQL で提供される権限」

RELOAD

セクション5.1.12.3 「DNS ルックアップとホストキャッシュ」
セクション13.7.8.3 「FLUSH ステートメント」
セクション13.7.1.6 「GRANT ステートメント」
セクション13.3.5 「LOCK INSTANCE FOR BACKUP および UNLOCK INSTANCE ステートメント」
セクション2.11.4 「MySQL 8.0 での変更」
セクション6.2.2 「MySQL で提供される権限」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション13.4.1.2 「RESET MASTER ステートメント」
セクション13.4.2.5 「RESET REPLICA | SLAVE ステートメント」
セクション13.7.8.6 「RESET ステートメント」
セクション6.2.8 「アカウントの追加、権限の割当ておよびアカウントの削除」
セクション6.2.7 「アクセス制御、ステージ 2: リクエストの確認」
セクション5.4.6 「サーバーログの保守」
セクション6.2.3 「付与テーブル」

REPLICATION CLIENT

セクション13.7.1.6 「GRANT ステートメント」
セクション6.2.2 「MySQL で提供される権限」
セクション13.7.7.1 「SHOW BINARY LOGS ステートメント」
セクション13.7.7.23 「SHOW MASTER STATUS ステートメント」
セクション13.7.7.35 「SHOW REPLICA | SLAVE STATUS ステートメント」

REPLICATION SLAVE

セクション13.7.1.6 「GRANT ステートメント」
セクション6.2.2 「MySQL で提供される権限」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」
セクション13.7.7.2 「SHOW BINLOG EVENTS ステートメント」
セクション13.7.7.32 「SHOW RELAYLOG EVENTS ステートメント」
セクション13.7.7.33 「SHOW REPLICAS | SHOW SLAVE HOSTS ステートメント」
セクション17.1.5.1 「マルチソースレプリケーションの構成」
セクション17.1.2.3 「レプリケーション用ユーザーの作成」
セクション18.2.1.3 「分散リカバリのユーザー資格証明」
セクション17.3.1 「暗号化接続を使用するためのレプリケーションの設定」

REPLICATION_APPLIER

セクション13.7.8.1 「BINLOG ステートメント」
セクション13.4.2.1 「CHANGE MASTER TO ステートメント」
セクション13.4.2.3 「CHANGE REPLICATION SOURCE TO ステートメント」
セクション13.7.1.6 「GRANT ステートメント」
セクション6.2.2 「MySQL で提供される権限」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」
セクション17.1.6.5 「グローバルトランザクション ID システム変数」
セクション5.1.8 「サーバーシステム変数」
セクション17.1.6.4 「バイナリロギングのオプションと変数」
セクション17.3.3.1 「レプリケーション PRIVILEGE_CHECKS_USER アカウントの権限」
セクション17.1.6.2 「レプリケーションソースのオプションと変数」
セクション17.3.3 「レプリケーション権限チェック」

REPLICATION_SLAVE_ADMIN

セクション13.4.2.1 「CHANGE MASTER TO ステートメント」
セクション13.4.2.2 「CHANGE REPLICATION FILTER ステートメント」
セクション13.4.2.3 「CHANGE REPLICATION SOURCE TO ステートメント」
セクション13.7.1.6 「GRANT ステートメント」
セクション6.2.2 「MySQL で提供される権限」
セクション13.4.2.7 「START REPLICA | SLAVE ステートメント」
セクション13.4.2.9 「STOP REPLICA | SLAVE ステートメント」
セクション17.4.10.2 「準同期レプリケーションのインストールと構成」

RESOURCE_GROUP_ADMIN

セクション13.7.2.1 「ALTER RESOURCE GROUP ステートメント」
セクション13.7.2.2 「CREATE RESOURCE GROUP ステートメント」
セクション13.7.2.3 「DROP RESOURCE GROUP ステートメント」
セクション13.7.1.6 「GRANT ステートメント」
セクション6.2.2 「MySQL で提供される権限」
セクション13.7.2.4 「SET RESOURCE GROUP ステートメント」

[セクション8.9.3「オプティマイザヒント」](#)
[セクション5.1.16「リソースグループ」](#)

RESOURCE_GROUP_USER

[セクション13.7.1.6「GRANT ステートメント」](#)
[セクション6.2.2「MySQL で提供される権限」](#)
[セクション13.7.2.4「SET RESOURCE GROUP ステートメント」](#)
[セクション8.9.3「オプティマイザヒント」](#)
[セクション5.1.16「リソースグループ」](#)

ROLE_ADMIN

[セクション13.7.1.6「GRANT ステートメント」](#)
[セクション6.2.2「MySQL で提供される権限」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション6.2.10「ロールの使用」](#)
[セクション12.16「情報関数」](#)

S

[\[index top\]](#)

SELECT

[セクション13.7.3.1「ANALYZE TABLE ステートメント」](#)
[セクション13.7.3.3「CHECKSUM TABLE ステートメント」](#)
[セクション13.1.17「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」](#)
[セクション13.1.20.3「CREATE TABLE ... LIKE ステートメント」](#)
[セクション13.1.20「CREATE TABLE ステートメント」](#)
[セクション13.1.22「CREATE TRIGGER ステートメント」](#)
[セクション13.1.23「CREATE VIEW ステートメント」](#)
[セクション13.2.2「DELETE ステートメント」](#)
[セクション27.12.19.1「error_log テーブル」](#)
[セクション13.7.8.3「FLUSH ステートメント」](#)
[セクション13.7.1.6「GRANT ステートメント」](#)
[セクション26.51.21「INFORMATION_SCHEMA INNODB_LOCK_WAITS テーブル」](#)
[セクション26.51.20「INFORMATION_SCHEMA INNODB_LOCKS テーブル」](#)
[セクション26.30「INFORMATION_SCHEMA ROUTINES テーブル」](#)
[セクション26.46「INFORMATION_SCHEMA USER_ATTRIBUTES テーブル」](#)
[セクション13.2.6「INSERT ステートメント」](#)
[セクション13.3.6「LOCK TABLES および UNLOCK TABLES ステートメント」](#)
[セクション16.7「MERGE ストレージエンジン」](#)
[セクション6.4.7.4「MySQL Enterprise Firewall リファレンス」](#)
[セクション6.2.2「MySQL で提供される権限」](#)
[セクション4.5.4「mysqldump — データベースバックアッププログラム」](#)
[セクション4.5.6「mysqlpump — データベースバックアッププログラム」](#)
[セクション23.5.17.2「NDB Cluster および MySQL の権限」](#)
[セクション13.7.3.4「OPTIMIZE TABLE ステートメント」](#)
[セクション13.7.3.5「REPAIR TABLE ステートメント」](#)
[セクション13.7.1.8「REVOKE ステートメント」](#)
[セクション13.7.7.9「SHOW CREATE PROCEDURE ステートメント」](#)
[セクション13.7.7.12「SHOW CREATE USER ステートメント」](#)
[セクション13.7.7.13「SHOW CREATE VIEW ステートメント」](#)
[セクション13.7.7.21「SHOW GRANTS ステートメント」](#)
[セクション13.7.7.27「SHOW PROCEDURE CODE ステートメント」](#)
[セクション13.7.7.28「SHOW PROCEDURE STATUS ステートメント」](#)
[セクション28.1「sys スキーマを使用するための前提条件」](#)
[セクション13.2.13「UPDATE ステートメント」](#)
[セクション6.2.11「アカウントカテゴリ」](#)
[セクション6.2.7「アクセス制御、ステージ 2: リクエストの確認」](#)

セクション6.2「アクセス制御とアカウント管理」
セクション25.4.6「イベントスケジューラと MySQL 権限」
セクション5.6.7.9「クローニング操作の監視」
セクション25.6「ストアオブジェクトのアクセス制御」
セクション25.2.2「ストアルーチンと MySQL 権限」
セクション27.12.19.10「スレッドテーブル」
セクション14.7「データディクショナリの使用法の違い」
セクション25.3.1「トリガーの構文と例」
セクション27.11「パフォーマンススキーマの一般的なテーブル特性」
セクション25.9「ビューの制約」
セクション15.7.2.4「読取りのロック」
セクション6.2.12「部分取消しを使用した権限の制限」

SERVICE_CONNECTION_ADMIN

セクション6.2.2「MySQL で提供される権限」
分散リカバリエンドポイントのアドレスの選択
セクション5.1.12.2「管理接続管理」

SESSION_VARIABLES_ADMIN

セクション13.7.1.6「GRANT ステートメント」
セクション6.2.2「MySQL で提供される権限」
セクション5.1.9.1「システム変数権限」
セクション17.3.3.1「レプリケーション PRIVILEGE_CHECKS_USER アカウントの権限」

SET_USER_ID

セクション13.1.11「ALTER VIEW ステートメント」
セクション13.7.1.3「CREATE USER ステートメント」
セクション13.7.1.5「DROP USER ステートメント」
セクション13.7.1.6「GRANT ステートメント」
セクション6.2.2「MySQL で提供される権限」
セクション13.7.1.7「RENAME USER ステートメント」
セクション6.2.11「アカウントカテゴリ」
セクション25.6「ストアオブジェクトのアクセス制御」
セクション25.7「ストアプログラムバイナリロギング」

SHOW DATABASES

セクション13.7.1.6「GRANT ステートメント」
セクション6.2.2「MySQL で提供される権限」
セクション13.7.7.14「SHOW DATABASES ステートメント」
セクション5.1.8「サーバーシステム変数」

SHOW VIEW

セクション13.8.2「EXPLAIN ステートメント」
セクション13.7.1.6「GRANT ステートメント」
セクション26.48「INFORMATION_SCHEMA VIEWS テーブル」
セクション6.2.2「MySQL で提供される権限」
セクション4.5.4「mysqldump — データベースバックアッププログラム」
セクション4.5.6「mysqlpump — データベースバックアッププログラム」
セクション13.7.7.13「SHOW CREATE VIEW ステートメント」
セクション25.9「ビューの制約」

SHOW_ROUTINE

セクション13.7.1.6「GRANT ステートメント」
セクション26.30「INFORMATION_SCHEMA ROUTINES テーブル」
セクション6.2.2「MySQL で提供される権限」
セクション13.7.7.9「SHOW CREATE PROCEDURE ステートメント」
セクション13.7.7.27「SHOW PROCEDURE CODE ステートメント」

セクション13.7.7.28 「SHOW PROCEDURE STATUS ステートメント」
セクション25.2.2 「ストアドルーチンと MySQL 権限」

SHUTDOWN

セクション13.7.1.6 「GRANT ステートメント」
セクション17.1.3.4 「GTID を使用したレプリケーションのセットアップ」
セクション4.10 「MySQL での Unix シグナル処理」
セクション6.2.2 「MySQL で提供される権限」
セクション4.3.4 「mysqld_multi — 複数の MySQL サーバーの管理」
セクション13.7.8.8 「RESTART ステートメント」
セクション13.7.8.9 「SHUTDOWN ステートメント」
セクション6.2.7 「アクセス制御、ステージ 2: リクエストの確認」
セクション5.1.19 「サーバーの停止プロセス」
セクション5.6.7.3 「リモートデータのクローニング」
セクション6.2.3 「付与テーブル」

SUPER

セクション13.1.4 「ALTER FUNCTION ステートメント」
セクション13.1.5 「ALTER INSTANCE ステートメント」
セクション13.1.8 「ALTER SERVER ステートメント」
セクション13.7.1.1 「ALTER USER ステートメント」
セクション13.1.11 「ALTER VIEW ステートメント」
セクション13.7.8.1 「BINLOG ステートメント」
セクション13.4.2.1 「CHANGE MASTER TO ステートメント」
セクション13.4.2.2 「CHANGE REPLICATION FILTER ステートメント」
セクション13.4.2.3 「CHANGE REPLICATION SOURCE TO ステートメント」
セクション13.1.17 「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」
セクション13.7.1.2 「CREATE ROLE ステートメント」
セクション13.1.18 「CREATE SERVER ステートメント」
セクション13.1.19 「CREATE SPATIAL REFERENCE SYSTEM ステートメント」
セクション13.1.22 「CREATE TRIGGER ステートメント」
セクション13.7.1.3 「CREATE USER ステートメント」
セクション28.4.4.2 「diagnostics() プロシージャ」
セクション5.1.12.3 「DNS ルックアップとホストキャッシュ」
セクション13.7.1.4 「DROP ROLE ステートメント」
セクション13.1.30 「DROP SERVER ステートメント」
セクション13.1.31 「DROP SPATIAL REFERENCE SYSTEM ステートメント」
セクション13.7.1.5 「DROP USER ステートメント」
セクション13.7.1.6 「GRANT ステートメント」
セクション17.1.3.4 「GTID を使用したレプリケーションのセットアップ」
セクション15.13 「InnoDB 保存データ暗号化」
セクション13.7.8.4 「KILL ステートメント」
セクションA.4 「MySQL 8.0 FAQ: ストアドプロシージャおよびストアドファンクション」
セクション6.5.5 「MySQL Enterprise Data Masking and De-Identification ユーザー定義関数の説明」
セクション6.5.1 「MySQL Enterprise Data Masking and De-Identification 要素」
セクション6.4.7.3 「MySQL Enterprise Firewall の使用」
セクション6.4.7.4 「MySQL Enterprise Firewall リファレンス」
セクション5.1.15 「MySQL Server でのタイムゾーンのサポート」
セクション6.2.2 「MySQL で提供される権限」
セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」
セクション13.7.1.7 「RENAME USER ステートメント」
セクション13.7.8.7 「RESET PERSIST ステートメント」
セクション13.7.1.8 「REVOKE ステートメント」
セクション13.7.1.10 「SET PASSWORD ステートメント」
セクション13.3.7 「SET TRANSACTION ステートメント」
セクション13.7.7.1 「SHOW BINARY LOGS ステートメント」
セクション13.7.7.23 「SHOW MASTER STATUS ステートメント」
セクション13.7.7.29 「SHOW PROCESSLIST ステートメント」

セクション13.7.7.35 「SHOW REPLICA | SLAVE STATUS ステートメント」
セクション13.4.3.1 「START GROUP_REPLICATION ステートメント」
セクション13.4.2.7 「START REPLICA | SLAVE ステートメント」
セクション13.3.1 「START TRANSACTION、COMMIT および ROLLBACK ステートメント」
セクション13.4.3.2 「STOP GROUP_REPLICATION ステートメント」
セクション13.4.2.9 「STOP REPLICA | SLAVE ステートメント」
セクション6.2.11 「アカウントカテゴリ」
セクション6.2.14 「アカウントパスワードの割り当て」
セクション13.7.1 「アカウント管理ステートメント」
セクション10.5 「アプリケーションの文字セットおよび照合順序の構成」
セクション6.4.4.9 「キーリングキーストア間のキーの移行」
セクション6.4.4.13 「キーリングシステム変数」
セクション5.6.7.12 「クローンシステム変数」
セクション18.8 「グループレプリケーションシステム変数」
セクション5.1.11 「サーバー SQL モード」
セクション5.1.8 「サーバーシステム変数」
セクション5.1.9.1 「システム変数権限」
セクション25.6 「ストアオブジェクトのアクセス制御」
セクション25.7 「ストアプログラムバイナリロギング」
セクション17.1.6.4 「バイナリロギングのオプションと変数」
セクション17.1.2 「バイナリログファイルの位置ベースのレプリケーションの設定」
セクション5.6.6.3 「バージョントークンの使用」
セクション5.6.6.4 「バージョントークン参照」
セクション5.6.6.1 「バージョントークン要素」
セクション6.4.4.11 「プラグイン固有のキーリングキー管理関数」
セクション6.2.10 「ロールの使用」
セクション12.16 「情報関数」
セクションB.3.2.5 「接続が多すぎます」
セクション5.1.12.1 「接続インターフェース」
セクション6.1.3 「攻撃者に対する MySQL のセキュアな状態の維持」
セクション17.4.10.2 「準同期レプリケーションのインストールと構成」
セクション6.4.5.7 「監査ログのフィルタリング」
セクション6.4.5.10 「監査ログ参照」
セクション5.1.12.2 「管理接続管理」
セクション18.6.6.4 「終了処理」

SYSTEM_USER

セクション13.7.1.6 「GRANT ステートメント」
セクション26.23 「INFORMATION_SCHEMA PROCESSLIST テーブル」
セクション26.46 「INFORMATION_SCHEMA USER_ATTRIBUTES テーブル」
セクション13.7.8.4 「KILL ステートメント」
セクション1.3 「MySQL 8.0 の新機能」
セクション6.2.2 「MySQL で提供される権限」
セクション27.12.19.9 「processlist テーブル」
セクション13.7.7.29 「SHOW PROCESSLIST ステートメント」
セクション6.2.11 「アカウントカテゴリ」
セクション5.1.8 「サーバーシステム変数」
セクション25.6 「ストアオブジェクトのアクセス制御」
セクション6.2.10 「ロールの使用」
セクション12.16 「情報関数」

SYSTEM_VARIABLES_ADMIN

セクション5.1.12.3 「DNS ルックアップとホストキャッシュ」
セクション13.7.1.6 「GRANT ステートメント」
セクション5.1.15 「MySQL Server でのタイムゾーンのサポート」
セクション6.2.2 「MySQL で提供される権限」
セクション13.7.8.7 「RESET PERSIST ステートメント」
セクション6.4.4.9 「キーリングキーストア間のキーの移行」
セクション6.4.4.13 「キーリングシステム変数」

セクション5.6.7.12 「クローンシステム変数」
セクション5.1.11 「サーバー SQL モード」
セクション5.1.8 「サーバーシステム変数」
セクション5.1.9.1 「システム変数権限」
セクション6.4.4.11 「プラグイン固有のキーリングキー管理関数」
セクション5.6.7.3 「リモートデータのクローニング」
セクション6.2.10 「ロールの使用」
セクション6.2.3 「付与テーブル」
セクション6.4.5.10 「監査ログ参照」

T

[\[index top\]](#)

TABLE_ENCRYPTION_ADMIN

セクション13.1.2 「ALTER DATABASE ステートメント」
セクション13.1.9 「ALTER TABLE ステートメント」
セクション13.1.10 「ALTER TABLESPACE ステートメント」
セクション13.1.12 「CREATE DATABASE ステートメント」
セクション13.1.20 「CREATE TABLE ステートメント」
セクション13.1.21 「CREATE TABLESPACE ステートメント」
セクション13.7.1.6 「GRANT ステートメント」
セクション15.13 「InnoDB 保存データ暗号化」
セクション1.3 「MySQL 8.0 の新機能」
セクション6.2.2 「MySQL で提供される権限」
セクション13.1.36 「RENAME TABLE ステートメント」
セクション5.1.8 「サーバーシステム変数」
セクション17.3.3.1 「レプリケーション PRIVILEGE_CHECKS_USER アカウントの権限」

TRIGGER

セクション13.1.22 「CREATE TRIGGER ステートメント」
セクション13.1.34 「DROP TRIGGER ステートメント」
セクション13.7.1.6 「GRANT ステートメント」
セクション26.45 「INFORMATION_SCHEMA TRIGGERS テーブル」
セクション6.2.2 「MySQL で提供される権限」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.6 「mysqldump — データベースバックアッププログラム」
セクション13.7.7.11 「SHOW CREATE TRIGGER ステートメント」
セクション13.7.7.40 「SHOW TRIGGERS ステートメント」

U

[\[index top\]](#)

UPDATE

セクション13.7.1.1 「ALTER USER ステートメント」
セクション13.1.20 「CREATE TABLE ステートメント」
セクション13.1.22 「CREATE TRIGGER ステートメント」
セクション13.7.1.6 「GRANT ステートメント」
セクション26.46 「INFORMATION_SCHEMA USER_ATTRIBUTES テーブル」
セクション13.2.6 「INSERT ステートメント」
セクション16.7 「MERGE ストレージエンジン」
セクション6.2.2 「MySQL で提供される権限」
セクション23.5.17.2 「NDB Cluster および MySQL の権限」
セクション13.7.1.7 「RENAME USER ステートメント」
セクション13.7.1.8 「REVOKE ステートメント」
セクション13.7.1.9 「SET DEFAULT ROLE ステートメント」
セクション13.7.1.10 「SET PASSWORD ステートメント」

[セクション27.12.2.4 「setup_objects テーブル」](#)
[セクション28.1 「sys スキーマを使用するための前提条件」](#)
[セクション13.2.13 「UPDATE ステートメント」](#)
[セクション6.2.11 「アカウントカテゴリ」](#)
[セクション6.2.14 「アカウントパスワードの割り当て」](#)
[セクション5.1.8 「サーバーシステム変数」](#)
[セクション25.6 「ストアオブジェクトのアクセス制御」](#)
[セクション25.3.1 「トリガーの構文と例」](#)
[セクション6.2.15 「パスワード管理」](#)
[セクション27.11 「パフォーマンススキーマの一般的なテーブル特性」](#)
[セクション27.12.2 「パフォーマンススキーマセットアップテーブル」](#)
[セクション27.4 「パフォーマンススキーマ実行時構成」](#)
[セクション17.3.3.1 「レプリケーション PRIVILEGE_CHECKS_USER アカウントの権限」](#)
[セクション6.2.10 「ロールの使用」](#)
[セクション15.7.2.4 「読取りのロック」](#)
[セクション6.2.12 「部分取消しを使用した権限の制限」](#)

USAGE

[セクション13.7.1.6 「GRANT ステートメント」](#)
[セクション6.2.2 「MySQL で提供される権限」](#)

V

[\[index top\]](#)

VERSION_TOKEN_ADMIN

[セクション13.7.1.6 「GRANT ステートメント」](#)
[セクション6.2.2 「MySQL で提供される権限」](#)
[セクション5.6.6.3 「バージョントークンの使用」](#)
[セクション5.6.6.4 「バージョントークン参照」](#)
[セクション5.6.6.1 「バージョントークン要素」](#)

X

[\[index top\]](#)

XA_RECOVER_ADMIN

[セクション13.7.1.6 「GRANT ステートメント」](#)
[セクション6.2.2 「MySQL で提供される権限」](#)
[セクション13.3.8.1 「XA トランザクション SQL ステートメント」](#)
[セクション5.1.8 「サーバーシステム変数」](#)

SQL モードの索引

[A](#) | [E](#) | [H](#) | [I](#) | [N](#) | [O](#) | [P](#) | [R](#) | [S](#) | [T](#)

A

[\[index top\]](#)

ALLOW_INVALID_DATES

[セクションB.3.4.2 「DATE カラムの使用に関する問題」](#)
[セクション11.2.2 「DATE、DATETIME、および TIMESTAMP 型」](#)
[セクション5.1.11 「サーバー SQL モード」](#)
[セクション12.7 「日付および時間関数」](#)
[セクション11.2 「日時データ型」](#)

ANSI

[セクション26.48「INFORMATION_SCHEMA VIEWS テーブル」](#)
[セクション13.7.7.13「SHOW CREATE VIEW ステートメント」](#)
[セクション5.1.11「サーバー SQL モード」](#)
[セクション9.2.5「関数名の構文解析と解決」](#)

ANSI_QUOTES

[セクション13.1.20.5「FOREIGN KEY の制約」](#)
[セクション4.5.1.6「mysql クライアントのヒント」](#)
[セクション4.5.4「mysqldump — データベースバックアッププログラム」](#)
[セクション8.9.3「オプティマイザヒント」](#)
[セクション5.1.11「サーバー SQL モード」](#)
[セクション9.2「スキーマオブジェクト名」](#)
[セクション9.1.1「文字列リテラル」](#)
[セクション1.7.1「標準 SQL に対する MySQL 拡張機能」](#)

E

[\[index top\]](#)

ERROR_FOR_DIVISION_BY_ZERO

[セクションA.3「MySQL 8.0 FAQ: サーバー SQL モード」](#)
[セクション5.1.11「サーバー SQL モード」](#)
[セクション12.25.3「式の処理」](#)
[セクション12.25.5「高精度計算の例」](#)

H

[\[index top\]](#)

HIGH_NOT_PRECEDENCE

[セクション5.1.11「サーバー SQL モード」](#)
[セクション9.5「式」](#)
[セクション12.4.1「演算子の優先順位」](#)

I

[\[index top\]](#)

IGNORE_SPACE

[セクション13.1.17「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」](#)
[セクション4.5.1.1「mysql クライアントオプション」](#)
[セクション5.1.11「サーバー SQL モード」](#)
[セクション9.2.5「関数名の構文解析と解決」](#)

N

[\[index top\]](#)

NO_AUTO_VALUE_ON_ZERO

[セクション3.6.9「AUTO_INCREMENT の使用」](#)
[セクション13.1.20「CREATE TABLE ステートメント」](#)
[セクション5.1.11「サーバー SQL モード」](#)
[セクション11.1.6「数値型の属性」](#)

NO_BACKSLASH_ESCAPES

[セクション11.5「JSON データ型」](#)

セクション12.18.4 「JSON 値を変更する関数」
セクション5.1.11 「サーバー SQL モード」
セクション9.1.1 「文字列リテラル」
セクション12.8.1 「文字列比較関数および演算子」

NO_DIR_IN_CREATE

セクション13.1.20 「CREATE TABLE ステートメント」
セクション5.1.11 「サーバー SQL モード」
セクション5.4.4 「バイナリログ」
セクション17.5.1.10 「レプリケーションと DIRECTORY テーブルオプション」
セクション17.5.1.39 「レプリケーションと変数」

NO_ENGINE_SUBSTITUTION

セクション13.1.9 「ALTER TABLE ステートメント」
セクション13.1.20 「CREATE TABLE ステートメント」
セクションA.3 「MySQL 8.0 FAQ: サーバー SQL モード」
セクション5.1.11 「サーバー SQL モード」
セクション16.1 「ストレージエンジンの設定」
セクション5.6.1 「プラグインのインストールおよびアンインストール」
セクション17.4.4 「異なるソースおよびレプリカのストレージエンジンでのレプリケーションの使用」

NO_UNSIGNED_SUBTRACTION

セクション12.11 「キャスト関数と演算子」
セクション5.1.11 「サーバー SQL モード」
セクション24.6 「パーティショニングの制約と制限」
セクション11.1.1 「数値データ型の構文」
セクション12.6.1 「算術演算子」
セクション11.1.7 「範囲外およびオーバーフローの処理」

NO_ZERO_DATE

セクション13.1.20 「CREATE TABLE ステートメント」
セクションB.3.4.2 「DATE カラムの使用に関する問題」
セクション11.2.2 「DATE、DATETIME、および TIMESTAMP 型」
セクションA.3 「MySQL 8.0 FAQ: サーバー SQL モード」
セクション11.2.5 「TIMESTAMP および DATETIME の自動初期化および更新機能」
セクション12.11 「キャスト関数と演算子」
セクション5.1.11 「サーバー SQL モード」
セクション5.1.8 「サーバーシステム変数」
セクション12.7 「日付および時間関数」
セクション11.2 「日時データ型」

NO_ZERO_IN_DATE

セクション13.1.20 「CREATE TABLE ステートメント」
セクションB.3.4.2 「DATE カラムの使用に関する問題」
セクションA.3 「MySQL 8.0 FAQ: サーバー SQL モード」
セクション5.1.11 「サーバー SQL モード」
セクション11.2 「日時データ型」

O

[\[index top\]](#)

ONLY_FULL_GROUP_BY

セクション12.20.2 「GROUP BY 修飾子」
セクションA.3 「MySQL 8.0 FAQ: サーバー SQL モード」
セクション12.20.3 「MySQL での GROUP BY の処理」
セクション12.24 「その他の関数」

[セクション5.1.11「サーバー SQL モード」](#)
[セクション3.3.4.8「行のカウント」](#)

P

[\[index top\]](#)

PAD_CHAR_TO_FULL_LENGTH

[セクション11.3.2「CHAR および VARCHAR 型」](#)
[セクション1.3「MySQL 8.0 の新機能」](#)
[セクション5.1.11「サーバー SQL モード」](#)
[セクション11.3.1「文字列データ型の構文」](#)

PIPES_AS_CONCAT

[セクション1.3「MySQL 8.0 の新機能」](#)
[セクション5.1.11「サーバー SQL モード」](#)
[セクション9.5「式」](#)
[セクション12.4.1「演算子の優先順位」](#)
[セクション12.4.3「論理演算子」](#)

R

[\[index top\]](#)

REAL_AS_FLOAT

[セクション5.1.11「サーバー SQL モード」](#)
[セクション11.1「数値データ型」](#)
[セクション11.1.1「数値データ型の構文」](#)

S

[\[index top\]](#)

STRICT_ALL_TABLES

[セクションA.3「MySQL 8.0 FAQ: サーバー SQL モード」](#)
[セクション5.1.11「サーバー SQL モード」](#)
[セクション17.5.3「レプリケーションセットアップをアップグレードする」](#)
[セクション12.25.3「式の処理」](#)

STRICT_TRANS_TABLES

[セクションA.3「MySQL 8.0 FAQ: サーバー SQL モード」](#)
[セクション5.1.11「サーバー SQL モード」](#)
[セクション17.5.3「レプリケーションセットアップをアップグレードする」](#)
[セクション12.25.3「式の処理」](#)

T

[\[index top\]](#)

TIME_TRUNCATE_FRACTIONAL

[セクション5.1.11「サーバー SQL モード」](#)
[セクション11.2.6「時間値での小数秒」](#)

TRADITIONAL

[セクション13.2.7「LOAD DATA ステートメント」](#)

[セクションA.3「MySQL 8.0 FAQ: サーバー SQL モード」](#)
[セクション11.2.5「TIMESTAMP および DATETIME の自動初期化および更新機能」](#)
[セクション5.1.11「サーバー SQL モード」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション12.25.3「式の処理」](#)

ステートメント/構文の索引

[A](#) | [B](#) | [C](#) | [D](#) | [E](#) | [F](#) | [G](#) | [H](#) | [I](#) | [K](#) | [L](#) | [O](#) | [P](#) | [R](#) | [S](#) | [T](#) | [U](#) | [V](#) | [W](#) | [X](#)

A

[\[index top\]](#)

ADD PARTITION

[セクション15.12.1「オンライン DDL 操作」](#)

ALTER DATABASE

[セクション13.1.2「ALTER DATABASE ステートメント」](#)
[セクション1.3「MySQL 8.0 の新機能」](#)
[セクション4.6.8「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」](#)
[セクション4.5.4「mysqldump — データベースバックアッププログラム」](#)
[セクション23.1.4「NDB Cluster の新機能」](#)
[セクション17.1.6.3「Replica Server のオプションと変数」](#)
[セクション10.5「アプリケーションの文字セットおよび照合順序の構成」](#)
[セクション17.2.5「サーバーがレプリケーションフィルタリングルールをどのように評価するか」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション17.2.5.1「データベースレベルレプリケーションオプションおよびバイナリロギングオプションの評価」](#)
[セクション10.3.3「データベース文字セットおよび照合順序」](#)
[セクション1.7.1「標準 SQL に対する MySQL 拡張機能」](#)

ALTER EVENT

[セクション13.1.2「ALTER DATABASE ステートメント」](#)
[セクション13.1.3「ALTER EVENT ステートメント」](#)
[セクション13.1.13「CREATE EVENT ステートメント」](#)
[セクション17.5.1.8「CURRENT_USER\(\) のレプリケーション」](#)
[セクション26.14「INFORMATION_SCHEMA EVENTS テーブル」](#)
[セクション13.7.7.18「SHOW EVENTS ステートメント」](#)
[セクション25.4.6「イベントスケジューラと MySQL 権限」](#)
[セクション25.4.1「イベントスケジューラの概要」](#)
[セクション25.4.4「イベントメタデータ」](#)
[セクション25.4.3「イベント構文」](#)
[セクション25.6「ストアドオブジェクトのアクセス制御」](#)
[セクション25.8「ストアドプログラムの制約」](#)
[セクション25.7「ストアドプログラムバイナリロギング」](#)
[セクション17.1.2.8「レプリケーション環境へのレプリカの追加」](#)
[セクション17.5.1.16「呼び出される機能のレプリケーション」](#)
[セクション12.16「情報関数」](#)
[既存のデータによるレプリケーションのセットアップ](#)
[セクション13.3.3「暗黙的なコミットを発生させるステートメント」](#)

ALTER EVENT event_name ENABLE

[セクション17.5.1.16「呼び出される機能のレプリケーション」](#)

ALTER FUNCTION

[セクション13.1.4「ALTER FUNCTION ステートメント」](#)
[セクション25.7「ストアドプログラムバイナリロギング」](#)

[セクション25.2.1「ストアドルーチンの構文」](#)
[セクション13.3.3「暗黙的なコミットを発生させるステートメント」](#)

ALTER IGNORE TABLE

[セクション24.3.4「パーティションの保守」](#)

ALTER INSTANCE

[セクション13.1.5「ALTER INSTANCE ステートメント」](#)
[セクション15.6.5「redo ログ」](#)

ALTER INSTANCE DISABLE INNODB REDO_LOG

[セクション15.6.5「redo ログ」](#)

ALTER INSTANCE INNODB REDO_LOG

[セクション13.1.5「ALTER INSTANCE ステートメント」](#)
[セクション15.6.5「redo ログ」](#)

ALTER INSTANCE RELOAD TLS

[セクション18.5.2「Secure Socket Layer \(SSL\) を使用したグループ通信接続の保護」](#)
[セクション27.12.19.11「tls_channel_status テーブル」](#)
[セクション5.1.7「サーバーコマンドオプション」](#)
[セクション5.1.10「サーバーステータス変数」](#)
[セクション6.3.1「暗号化接続を使用するための MySQL の構成」](#)

ALTER INSTANCE ROTATE BINLOG MASTER KEY

[セクション17.3.2.3「バイナリログマスターキーのローテーション」](#)

ALTER INSTANCE ROTATE INNODB MASTER KEY

[セクション15.13「InnoDB 保存データ暗号化」](#)
[セクションA.17「MySQL 8.0 FAQ : InnoDB 保存データ暗号化」](#)

ALTER INSTANCE {ENABLE|DISABLE} INNODB REDO_LOG

[セクション1.3「MySQL 8.0 の新機能」](#)
[セクション6.2.2「MySQL で提供される権限」](#)

ALTER LOGFILE GROUP

[セクション13.1.6「ALTER LOGFILE GROUP ステートメント」](#)
[セクション26.15「INFORMATION_SCHEMA FILES テーブル」](#)
[セクション23.5.10.1「NDB Cluster ディスクデータオブジェクト」](#)
[セクション23.3.3.6「NDB Cluster データノードの定義」](#)
[セクション23.1.7.8「NDB Cluster 専用の問題」](#)

ALTER PROCEDURE

[セクション13.1.7「ALTER PROCEDURE ステートメント」](#)
[セクション25.7「ストアドプログラムバイナリロギング」](#)
[セクション25.2.1「ストアドルーチンの構文」](#)
[セクション13.3.3「暗黙的なコミットを発生させるステートメント」](#)

ALTER RESOURCE GROUP

[セクション13.7.2.1「ALTER RESOURCE GROUP ステートメント」](#)
[セクション5.1.16「リソースグループ」](#)

ALTER SCHEMA

[セクション13.1.2「ALTER DATABASE ステートメント」](#)
[セクション15.13「InnoDB 保存データ暗号化」](#)

ALTER SERVER

- セクション17.5.1.5「CREATE SERVER、ALTER SERVER、および DROP SERVER のレプリケーション」
- セクション6.2.2「MySQL で提供される権限」
- セクション13.1.1「アトミックデータ定義ステートメントのサポート」
- セクション13.3.3「暗黙的なコミットを発生させるステートメント」

ALTER TABLE

- セクション10.9.8「3 バイト Unicode 文字セットと 4 バイト Unicode 文字セット間の変換」
- セクション13.1.2「ALTER DATABASE ステートメント」
- セクション13.1.9.2「ALTER TABLE および生成されるカラム」
- セクションB.3.6.1「ALTER TABLE での問題」
- セクション13.1.9.3「ALTER TABLE の例」
- セクション13.1.9「ALTER TABLE ステートメント」
- セクション13.1.9.1「ALTER TABLE パーティション操作」
- セクション13.7.3.1「ANALYZE TABLE ステートメント」
- セクション3.6.9「AUTO_INCREMENT の使用」
- セクション13.7.3.2「CHECK TABLE ステートメント」
- セクション13.1.15「CREATE INDEX ステートメント」
- セクション13.1.20.4「CREATE TABLE ... SELECT ステートメント」
- セクション13.1.20「CREATE TABLE ステートメント」
- セクション13.1.21「CREATE TABLESPACE ステートメント」
- セクション13.1.27「DROP INDEX ステートメント」
- セクション8.8.2「EXPLAIN 出力フォーマット」
- セクション16.8.3「FEDERATED ストレージエンジンの注記とヒント」
- セクション15.6.3.2「File-Per-Table テーブルスペース」
- セクション1.7.3.2「FOREIGN KEY の制約」
- セクション13.1.20.5「FOREIGN KEY の制約」
- セクション13.7.1.6「GRANT ステートメント」
- セクション24.3.2「HASH および KEY パーティションの管理」
- セクション26.51.24「INFORMATION_SCHEMA INNODB_TABLES テーブル」
- セクション26.21「INFORMATION_SCHEMA PARTITIONS テーブル」
- セクション26.34「INFORMATION_SCHEMA STATISTICS テーブル」
- セクション15.6.2.4「InnoDB FULLTEXT インデックス」
- セクション15.20.6.4「InnoDB memcached プラグインのトランザクション動作の制御」
- セクション8.3.8「InnoDB および MyISAM インデックス統計コレクション」
- セクション15.12「InnoDB とオンライン DDL」
- セクション15.16「InnoDB の MySQL パフォーマンススキーマとの統合」
- セクション15.8.10「InnoDB のオプティマイザ統計の構成」
- セクション15.21.2「InnoDB のリカバリの強制的な実行」
- セクション15.10「InnoDB の行フォーマット」
- セクション15.14「InnoDB の起動オプションおよびシステム変数」
- セクション15.9.1.5「InnoDB テーブルでの圧縮の動作」
- セクション15.8.10.3「InnoDB テーブルに対する ANALYZE TABLE の複雑さの推定」
- セクション15.6.1.1「InnoDB テーブルの作成」
- セクション15.9.1「InnoDB テーブルの圧縮」
- セクション15.6.1.4「InnoDB テーブルの移動またはコピー」
- セクション15.9.2「InnoDB ページ圧縮」
- セクション15.13「InnoDB 保存データ暗号化」
- セクション13.7.8.4「KILL ステートメント」
- セクション13.3.6「LOCK TABLES および UNLOCK TABLES ステートメント」
- セクション12.10.9「MeCab フルテキストパーサープラグイン」
- セクション16.3「MEMORY ストレージエンジン」
- セクション16.7.2「MERGE テーブルの問題点」
- セクション15.6.1.5「MyISAM から InnoDB へのテーブルの変換」
- セクション16.2「MyISAM ストレージエンジン」
- セクション16.2.3「MyISAM テーブルのストレージフォーマット」
- セクション7.6.3「MyISAM テーブルの修復方法」
- セクション16.2.1「MyISAM 起動オプション」
- セクション4.6.4.1「myisamchk の一般オプション」

セクションA.10 「MySQL 8.0 FAQ: NDB Cluster」
セクション2.11.4 「MySQL 8.0 での変更」
セクション1.3 「MySQL 8.0 の新機能」
セクションB.3.3.5 「MySQL が一時ファイルを格納する場所」
セクションB.3.3.4 「MySQL が満杯のディスクを処理する方法」
セクションB.3.3.3 「MySQL が繰り返しクラッシュする場合の対処方法」
セクション6.2.2 「MySQL で提供される権限」
セクション24.1 「MySQL のパーティショニングの概要」
セクション12.10.6 「MySQL の全文検索の微調整」
セクションB.3.7 「MySQL の既知の問題」
MySQL 用語集
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション23.5.11 「NDB Cluster での ALTER TABLE を使用したオンライン操作」
セクション23.1.7.3 「NDB Cluster でのトランザクション処理に関する制限」
セクション23.1.7.6 「NDB Cluster でサポートされない機能または欠落している機能」
セクション23.1.7.2 「NDB Cluster と標準の MySQL 制限の制限と相違点」
NDB Cluster の MySQL Server オプション
セクション23.1.7.1 「NDB Cluster の SQL 構文に準拠していません」
セクション23.2.3 「NDB Cluster の初期構成」
NDB Cluster システム変数
セクション23.5.10.1 「NDB Cluster ディスクデータオブジェクト」
セクション23.5.7.3 「NDB Cluster データノードのオンラインでの追加: 詳細な例」
セクション23.3.3.6 「NDB Cluster データノードの定義」
セクション23.6.4 「NDB Cluster レプリケーションスキーマおよびテーブル」
セクション23.1.7.8 「NDB Cluster 専用の問題」
セクション23.3.3.1 「NDB Cluster 構成: 基本例」
セクション23.4.9 「ndb_desc — NDB テーブルの表示」
セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」
セクション13.1.20.11 「NDB_TABLE オプションの設定」
セクション12.10.8 「ngram 全文パーサー」
セクション13.7.3.4 「OPTIMIZE TABLE ステートメント」
セクション24.2.3.1 「RANGE COLUMNS パーティショニング」
セクション24.3.1 「RANGE および LIST パーティショニングの管理」
セクション24.2.1 「RANGE パーティショニング」
セクション13.1.36 「RENAME TABLE ステートメント」
セクション13.7.7.15 「SHOW ENGINE ステートメント」
セクション13.7.7.22 「SHOW INDEX ステートメント」
セクション13.7.7.39 「SHOW TABLES ステートメント」
セクション13.7.7.42 「SHOW WARNINGS ステートメント」
セクション15.9.1.7 「SQL 圧縮構文の警告とエラー」
セクションB.3.6.2 「TEMPORARY テーブルに関する問題」
セクション8.12.2.2 「Unix 上の MyISAM へのシンボリックリンクの使用」
セクション2.11.5 「アップグレード用のインストールの準備」
セクション13.1.1 「アトミックデータ定義ステートメントのサポート」
セクション15.8.11 「インデックスページのマーじしきい値の構成」
セクション15.12.2 「オンライン DDL のパフォーマンスと同時実行性」
セクション15.12.6 「オンライン DDL の制限事項」
セクション15.12.4 「オンライン DDL を使用した DDL ステートメントの簡略化」
セクション15.12.5 「オンライン DDL 失敗条件」
セクション15.12.1 「オンライン DDL 操作」
セクション8.3.5 「カラムインデックス」
セクション10.3.5 「カラム文字セットおよび照合順序」
セクション10.7 「カラム文字セットの変換」
セクション12.11 「キャスト関数と演算子」
セクション5.1.11 「サーバー SQL モード」
セクション5.1.7 「サーバーコマンドオプション」
セクション5.1.8 「サーバーシステム変数」
セクション24.6.2 「ストレージエンジンに関連するパーティショニング制限」
セクション16.1 「ストレージエンジンの設定」

セクション5.4.5「スロークエリーログ」
ソースまたはレプリカにカラムが多いレプリケーション
セクション23.2.5「テーブルとデータを含む NDB Cluster の例」
セクション15.11.4「テーブルのデフラグ」
セクション3.3.2「テーブルの作成」
セクション2.11.13「テーブルまたはインデックスの再作成または修復」
セクション8.4.6「テーブルサイズの制限」
セクション15.6.3.9「テーブルスペースの AUTOEXTEND_SIZE 構成」
セクション15.9.1.1「テーブル圧縮の概要」
セクション10.3.4「テーブル文字セットおよび照合順序」
セクション8.4.1「データサイズの最適化」
セクション11.6「データ型デフォルト値」
セクション17.2.1.3「バイナリロギングでの安全および安全でないステートメントの判断」
セクション17.1.6.4「バイナリロギングのオプションと変数」
セクション5.4.4.2「バイナリログ形式の設定」
セクション15.16.1「パフォーマンススキーマを使用した InnoDB テーブルの ALTER TABLE の進行状況のモニタリング」
セクション27.12.5「パフォーマンススキーマステージイベントテーブル」
セクション24.6「パーティショニングの制約と制限」
セクション24.6.1「パーティショニングキー、主キー、および一意キー」
セクション24.3.3「パーティションとサブパーティションをテーブルと交換する」
セクション24.3.4「パーティションの保守」
セクション24.3「パーティション管理」
セクション25.9「ビューの制約」
セクション14.2「ファイルベースのメタデータ記憶域の削除」
セクション17.5.1.1「レプリケーションと AUTO_INCREMENT」
セクション17.5.1.26「レプリケーションと予約語」
セクション5.4.1「一般クエリーログおよびスロークエリーログの出力先の選択」
セクション15.6.3.3「一般テーブルスペース」
セクション8.14.3「一般的なスレッドの状態」
セクション15.7.2.3「一貫性非ロック読み取り」
セクション8.3.12「不可視のインデックス」
個々のテーブルのオプティマイザ統計パラメータの構成
セクション12.10「全文検索関数」
セクション15.9.1.2「圧縮テーブルの作成」
セクション12.16「情報関数」
セクション11.3.1「文字列データ型の構文」
セクション13.1.20.7「暗黙のカラム指定の変更」
セクション13.3.3「暗黙的なコミットを発生させるステートメント」
セクション1.7.1「標準 SQL に対する MySQL 拡張機能」
セクション17.4.4「異なるソースおよびレプリカのストレージエンジンでのレプリケーションの使用」
セクション11.4.10「空間インデックスの作成」
セクション11.4.6「空間カラムの作成」
セクション11.1.7「範囲外およびオーバーフローの処理」
セクション23.1.7.10「複数の NDB Cluster ノードに関する制限事項」
セクション13.1.20.10「非表示カラム」

ALTER TABLE ... ADD COLUMN

セクション26.51.8「INFORMATION_SCHEMA INNODB_COLUMNS テーブル」

ALTER TABLE ... ADD FOREIGN KEY

セクション13.1.9「ALTER TABLE ステートメント」

ALTER TABLE ... ADD PARTITION

セクション24.3.1「RANGE および LIST パーティションの管理」

ALTER TABLE ... ALGORITHM=COPY

セクション13.1.9「ALTER TABLE ステートメント」

[セクション13.1.20.5「FOREIGN KEY の制約」](#)

ALTER TABLE ... ALGORITHM=INPLACE

[セクション13.1.9「ALTER TABLE ステートメント」](#)
[セクション13.1.20.5「FOREIGN KEY の制約」](#)
[セクション15.12.6「オンライン DDL の制限事項」](#)

ALTER TABLE ... ALGORITHM=INPLACE, REORGANIZE PARTITION

[セクション23.5.7.2「NDB Cluster データノードのオンラインでの追加: 基本手順」](#)
[セクション23.5.7.3「NDB Cluster データノードのオンラインでの追加: 詳細な例」](#)

ALTER TABLE ... AUTO_INCREMENT = N

[セクション15.6.1.6「InnoDB での AUTO_INCREMENT 処理」](#)

ALTER TABLE ... COMPRESSION

[セクション15.9.2「InnoDB ページ圧縮」](#)

ALTER TABLE ... COMPRESSION=None

[セクション15.9.2「InnoDB ページ圧縮」](#)

ALTER TABLE ... CONVERT TO CHARACTER SET

[セクション13.7.3.1「ANALYZE TABLE ステートメント」](#)

ALTER TABLE ... DISABLE KEYS

[セクション13.2.7「LOAD DATA ステートメント」](#)

ALTER TABLE ... DISCARD PARTITION ... TABLESPACE

[セクション15.6.1.3「InnoDB テーブルのインポート」](#)

ALTER TABLE ... DISCARD TABLESPACE

[セクション13.1.21「CREATE TABLESPACE ステートメント」](#)
[セクション26.5.1.25「INFORMATION_SCHEMA INNODB_TABLESPACES テーブル」](#)
[セクション15.6.1.3「InnoDB テーブルのインポート」](#)
MySQL 用語集
[セクション15.6.3.3「一般テーブルスペース」](#)

ALTER TABLE ... DROP FOREIGN KEY

[セクション13.1.9「ALTER TABLE ステートメント」](#)

ALTER TABLE ... DROP PARTITION

[セクション17.5.1.24「レプリケーションおよびパーティション化」](#)

ALTER TABLE ... ENABLE KEYS

[セクション13.2.7「LOAD DATA ステートメント」](#)

ALTER TABLE ... ENCRYPTION

[セクション13.1.5「ALTER INSTANCE ステートメント」](#)

ALTER TABLE ... ENGINE

[セクション5.1.8「サーバーシステム変数」](#)

ALTER TABLE ... ENGINE = MEMORY

[セクション17.5.1.21「レプリケーションと MEMORY テーブル」](#)

ALTER TABLE ... ENGINE permitted_engine

[セクション5.1.8「サーバーシステム変数」](#)

ALTER TABLE ... ENGINE=INNODB

[セクション1.3「MySQL 8.0 の新機能」](#)

[セクション23.6.4「NDB Cluster レプリケーションスキーマおよびテーブル」](#)

ALTER TABLE ... ENGINE=NDB

[セクション23.4.13「ndb_import — NDB への CSV データのインポート」](#)

ALTER TABLE ... EXCHANGE PARTITION

[セクション13.1.9.1「ALTER TABLE パーティション操作」](#)

[セクション24.3.3「パーティションとサブパーティションをテーブルと交換する」](#)

ALTER TABLE ... FORCE

[セクション13.7.3.4「OPTIMIZE TABLE ステートメント」](#)

ALTER TABLE ... IMPORT PARTITION ... TABLESPACE

[セクション15.6.1.3「InnoDB テーブルのインポート」](#)

ALTER TABLE ... IMPORT TABLESPACE

[セクション15.6.1.3「InnoDB テーブルのインポート」](#)

[セクション15.6.1.4「InnoDB テーブルの移動またはコピー」](#)

[MySQL 用語集](#)

ALTER TABLE ... OPTIMIZE PARTITION

[セクション24.6.2「ストレージエンジンに関連するパーティショニング制限」](#)

[セクション24.3.4「パーティションの保守」](#)

ALTER TABLE ... PARTITION BY

[セクション24.6.1「パーティショニングキー、主キー、および一意キー」](#)

ALTER TABLE ... PARTITION BY ...

[セクション24.3.1「RANGE および LIST パーティションの管理」](#)

[セクション24.6「パーティショニングの制約と制限」](#)

ALTER TABLE ... REMOVE PARTITIONING

[セクション1.3「MySQL 8.0 の新機能」](#)

ALTER TABLE ... RENAME

[セクション8.12.2.2「Unix 上の MyISAM へのシンボリックリンクの使用」](#)

ALTER TABLE ... REORGANIZE PARTITION

[セクション23.5.7.1「NDB Cluster データノードのオンラインでの追加: 一般的な問題」](#)

[セクション23.5.7.3「NDB Cluster データノードのオンラインでの追加: 詳細な例」](#)

[セクション23.5.1「NDB Cluster 管理クライアントのコマンド」](#)

[セクション2.11.5「アップグレード用のインストールの準備」](#)

ALTER TABLE ... REPAIR PARTITION

[セクション24.3.3「パーティションとサブパーティションをテーブルと交換する」](#)

[セクション24.3.4「パーティションの保守」](#)

ALTER TABLE ... TABLESPACE

[セクション13.1.21「CREATE TABLESPACE ステートメント」](#)

ALTER TABLE ... TRUNCATE PARTITION

[セクション24.3.4「パーティションの保守」](#)

[セクション24.3「パーティション管理」](#)

ALTER TABLE ... TRUNCATE PARTITION ALL

[セクション24.3.4「パーティションの保守」](#)

ALTER TABLE ...IMPORT TABLESPACE

[セクション13.1.21「CREATE TABLESPACE ステートメント」](#)

[セクション15.6.3.3「一般テーブルスペース」](#)

ALTER TABLE EXCHANGE PARTITION

[セクション24.3.3「パーティションとサブパーティションをテーブルと交換する」](#)

ALTER TABLE mysql.ndb_apply_status ENGINE=MyISAM

[セクション23.6.3「NDB Cluster レプリケーションの既知の問題」](#)

ALTER TABLE ndb_table ... ALGORITHM=INPLACE, TABLESPACE=new_tablespace

[セクション23.5.11「NDB Cluster での ALTER TABLE を使用したオンライン操作」](#)

ALTER TABLE t TRUNCATE PARTITION ()

[セクション13.2.2「DELETE ステートメント」](#)

ALTER TABLE table ENGINE = NDB

[セクション23.1.4「NDB Cluster の新機能」](#)

ALTER TABLE table_name ENGINE=InnoDB;

[セクション15.1.4「InnoDB を使用したテストおよびベンチマーク」](#)

ALTER TABLE table_name REORGANIZE PARTITION

[セクション23.5.7.3「NDB Cluster データノードのオンラインでの追加: 詳細な例」](#)

ALTER TABLE tbl_name ENCRYPTION = 'Y'

[セクションA.17「MySQL 8.0 FAQ : InnoDB 保存データ暗号化」](#)

ALTER TABLE tbl_name ENGINE=engine_name

[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)

[セクション14.7「データディクショナリを使用する方法の違い」](#)

[セクション15.8.2「読み取り専用操作作用の InnoDB の構成」](#)

ALTER TABLE tbl_name ENGINE=INNODB

[セクション13.1.9「ALTER TABLE ステートメント」](#)

[セクション15.11.4「テーブルのデフラグ」](#)

ALTER TABLE tbl_name FORCE

[セクション13.1.9「ALTER TABLE ステートメント」](#)

[セクション15.11.4「テーブルのデフラグ」](#)

ALTER TABLE tbl_name TABLESPACE tablespace_name

[セクション13.1.21「CREATE TABLESPACE ステートメント」](#)

MySQL 用語集

[セクション15.6.3.3 「一般テーブルスペース」](#)

ALTER TABLESPACE

[セクション13.1.10 「ALTER TABLESPACE ステートメント」](#)
[セクション13.1.21 「CREATE TABLESPACE ステートメント」](#)
[セクション13.1.33 「DROP TABLESPACE ステートメント」](#)
[セクション26.15 「INFORMATION_SCHEMA FILES テーブル」](#)
[セクション15.13 「InnoDB 保存データ暗号化」](#)
[セクション1.3 「MySQL 8.0 の新機能」](#)
[セクション23.5.10.1 「NDB Cluster デイスクデータオブジェクト」](#)
[セクション23.3.3.6 「NDB Cluster データノードの定義」](#)
[セクション23.1.7.8 「NDB Cluster 専用の問題」](#)
[セクション15.6.3.9 「テーブルスペースの AUTOEXTEND_SIZE 構成」](#)

ALTER TABLESPACE ... ADD DATAFILE

[セクション13.1.21 「CREATE TABLESPACE ステートメント」](#)

ALTER TABLESPACE ... DROP DATAFILE

[セクション13.1.33 「DROP TABLESPACE ステートメント」](#)

ALTER TABLESPACE ... DROP DATATFILE

[セクション13.1.21 「CREATE TABLESPACE ステートメント」](#)

ALTER TABLESPACE ... ENCRYPTION

[セクション15.12.1 「オンライン DDL 操作」](#)

ALTER TABLESPACE ... ENGINE

[セクション5.1.8 「サーバーシステム変数」](#)

ALTER TABLESPACE ... RENAME TO

[セクション1.3 「MySQL 8.0 の新機能」](#)
[セクション15.12.1 「オンライン DDL 操作」](#)
[セクション15.6.3.3 「一般テーブルスペース」](#)

ALTER TABLESPACE tablespace_name ENCRYPTION = 'Y'

[セクションA.17 「MySQL 8.0 FAQ : InnoDB 保存データ暗号化」](#)

ALTER UNDO TABLESPACE

[セクション2.11.4 「MySQL 8.0 での変更」](#)
[セクション1.3 「MySQL 8.0 の新機能」](#)
[セクション15.6.3.4 「undo テーブルスペース」](#)

ALTER UNDO TABLESPACE ... SET INACTIVE

[セクション26.51.25 「INFORMATION_SCHEMA INNODB_TABLESPACES テーブル」](#)

ALTER UNDO TABLESPACE tablespace_name SET ACTIVE

[セクション15.6.3.4 「undo テーブルスペース」](#)

ALTER UNDO TABLESPACE tablespace_name SET INACTIVE

[セクション15.6.3.4 「undo テーブルスペース」](#)

ALTER USER

[セクション13.7.1.1 「ALTER USER ステートメント」](#)

セクション13.7.1.2 「CREATE ROLE ステートメント」
セクション13.7.1.3 「CREATE USER ステートメント」
セクション6.8 「FIPS のサポート」
セクション13.7.1.6 「GRANT ステートメント」
セクション6.4.1.7 「LDAP プラガブル認証」
セクション2.11.4 「MySQL 8.0 での変更」
セクション1.3 「MySQL 8.0 の新機能」
セクション6.2.2 「MySQL で提供される権限」
セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
root のパスワードのリセット: 一般的な手順
セクション13.7.1.9 「SET DEFAULT ROLE ステートメント」
セクション13.7.1.10 「SET PASSWORD ステートメント」
セクション6.2.14 「アカウントパスワードの割り当て」
セクション6.2.20 「アカウントリソース制限の設定」
セクション6.2.19 「アカウントロック」
セクション6.2.6 「アクセス制御、ステージ 1: 接続の検証」
セクション4.2.3 「サーバーに接続するためのコマンドオプション」
セクション5.1.8 「サーバーシステム変数」
セクション6.4.1.9 「ソケットピア資格証明プラガブル認証」
セクション2.10.1 「データディレクトリの初期化」
セクション6.1.2.1 「パスワードセキュリティのためのエンドユーザーガイドライン」
セクション6.4.3.2 「パスワード検証オプションおよび変数」
セクション6.4.3 「パスワード検証コンポーネント」
セクション6.2.15 「パスワード管理」
セクション6.2.18 「プロキシユーザー」
セクション6.2.10 「ロールの使用」
セクション6.2.3 「付与テーブル」
セクション6.1.3 「攻撃者に対する MySQL のセキュアな状態の維持」
セクション6.3.1 「暗号化接続を使用するための MySQL の構成」
セクション13.3.3 「暗黙的なコミットを発生させるステートメント」
セクション6.2.16 「期限切れパスワードのサーバー処理」

ALTER USER ... ATTRIBUTE ...

セクション26.46 「INFORMATION_SCHEMA.USER_ATTRIBUTES テーブル」

ALTER USER ... DEFAULT ROLE

セクション13.7.1.1 「ALTER USER ステートメント」
セクション13.7.1.9 「SET DEFAULT ROLE ステートメント」

ALTER USER ... UNLOCK

セクション13.7.1.1 「ALTER USER ステートメント」
セクション6.2.19 「アカウントロック」
セクション6.2.15 「パスワード管理」

ALTER VIEW

セクション13.1.2 「ALTER DATABASE ステートメント」
セクション13.1.11 「ALTER VIEW ステートメント」
セクション17.5.1.8 「CURRENT_USER() のレプリケーション」
セクション25.6 「ストアオブジェクトのアクセス制御」
セクション25.8 「ストアプログラムの制約」
セクション25.5.1 「ビューの構文」
セクション25.5.2 「ビュー処理アルゴリズム」
セクション12.16 「情報関数」
セクション13.3.3 「暗黙的なコミットを発生させるステートメント」

ANALYZE PARTITION

セクション15.12.1 「オンライン DDL 操作」

ANALYZE TABLE

セクション13.1.9 「ALTER TABLE ステートメント」
セクション13.7.3.1 「ANALYZE TABLE ステートメント」
セクション13.1.15 「CREATE INDEX ステートメント」
セクション13.1.20 「CREATE TABLE ステートメント」
セクション8.8.1 「EXPLAIN によるクエリーの最適化」
セクション13.8.2 「EXPLAIN ステートメント」
セクション8.8.2 「EXPLAIN 出力フォーマット」
セクション26.51.27 「INFORMATION_SCHEMA INNODB_TABLESTATS ビュー」
セクション26.34 「INFORMATION_SCHEMA STATISTICS テーブル」
セクション26.38 「INFORMATION_SCHEMA TABLES テーブル」
セクション14.5 「INFORMATION_SCHEMA とデータディクショナリの統合」
セクション8.2.3 「INFORMATION_SCHEMA クエリーの最適化」
セクション8.3.8 「InnoDB および MyISAM インデックス統計コレクション」
セクション15.8.10 「InnoDB のオプティマイザ統計の構成」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
InnoDB オプティマイザ統計でサンプリングされるページの数の構成
セクション15.8.10.3 「InnoDB テーブルに対する ANALYZE TABLE の複雑さの推定」
InnoDB 永続的統計テーブル
InnoDB 永続的統計テーブルの例
セクション16.7.2 「MERGE テーブルの問題点」
セクション8.6.1 「MyISAM クエリーの最適化」
セクション7.6 「MyISAM テーブルの保守とクラッシュリカバリ」
セクション4.6.4.1 「myisamchk の一般オプション」
セクション2.11.4 「MySQL 8.0 での変更」
セクション6.2.2 「MySQL で提供される権限」
セクション12.10.6 「MySQL の全文検索の微調整」
セクション5.3 「mysql システムスキーマ」
MySQL 用語集
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション23.4.14 「ndb_index_stat — NDB インデックス統計ユーティリティ」
セクション8.2.1.2 「range の最適化」
セクション8.2.1 「SELECT ステートメントの最適化」
セクション13.7.7.22 「SHOW INDEX ステートメント」
セクション8.9.6 「オプティマイザ統計」
セクション8.9 「クエリーオプティマイザの制御」
セクション5.1.8 「サーバーシステム変数」
セクション5.4.5 「スロークエリーログ」
セクション14.7 「データディクショナリの使用法の違い」
セクション24.6 「パーティショニングの制約と制限」
セクション24.3.4 「パーティションの保守」
セクション17.5.1.13 「レプリケーションと FLUSH」
セクション8.14.3 「一般的なスレッドの状態」
個々のテーブルのオプティマイザ統計パラメータの構成
セクション13.3.3 「暗黙的なコミットを発生させるステートメント」
セクション1.7.1 「標準 SQL に対する MySQL 拡張機能」
永続オプティマイザ統計の自動統計計算の構成
永続統計計算への削除マーク付きレコードの組込み
セクション15.8.2 「読み取り専用操作の InnoDB の構成」
セクション15.8.10.2 「非永続的オプティマイザ統計のパラメータの構成」

autocommit = 0 の場合、

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

B

[\[index top\]](#)

BEGIN

- セクション13.6.1 「BEGIN ... END 複合ステートメント」
- セクション27.12.7.1 「events_transactions_current テーブル」
- セクション23.6.11 「NDB Cluster レプリケーションの競合解決」
- セクション17.1.6.3 「Replica Server のオプションと変数」
- セクション5.1.8 「サーバーシステム変数」
- セクション25.8 「ストアードプログラムの制約」
- セクション25.7 「ストアードプログラムバイナリロギング」
- セクション27.12.7 「パフォーマンススキーマのトランザクションテーブル」
- セクション17.5.1.35 「レプリケーションとトランザクション」
- セクション13.3.3 「暗黙的なコミットを発生させるステートメント」
- セクション15.7.2.2 「自動コミット、コミットおよびロールバック」

BEGIN ... END

- セクション13.6.1 「BEGIN ... END 複合ステートメント」
- セクション13.6.5.1 「CASE ステートメント」
- セクション13.1.22 「CREATE TRIGGER ステートメント」
- セクション13.6.7.2 「DECLARE ... HANDLER ステートメント」
- セクション13.6.3 「DECLARE ステートメント」
- セクション13.6.5.4 「LEAVE ステートメント」
- セクション13.3.1 「START TRANSACTION、COMMIT および ROLLBACK ステートメント」
- セクション25.4.1 「イベントスケジューラの概要」
- セクション13.6.6.1 「カーソル CLOSE ステートメント」
- セクション13.6.6.3 「カーソル FETCH ステートメント」
- セクション13.6.2 「ステートメントラベル」
- セクション25.8 「ストアードプログラムの制約」
- セクション25.1 「ストアードプログラムの定義」
- セクション25.3.1 「トリガーの構文と例」
- セクション13.6.7.6 「ハンドラのスコープに関するルール」
- セクション13.6.4.1 「ローカル変数 DECLARE ステートメント」
- セクション13.6.4.2 「ローカル変数のスコープと解決」
- セクション13.3.3 「暗黙的なコミットを発生させるステートメント」
- セクション13.6 「複合ステートメントの構文」

BINLOG

- セクション13.7.8.1 「BINLOG ステートメント」
- セクション6.2.2 「MySQL で提供される権限」
- セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」
- セクション4.6.8.2 「mysqlbinlog 行イベントの表示」
- セクション5.1.8 「サーバーシステム変数」
- セクション17.1.6.4 「バイナリロギングのオプションと変数」
- セクション17.3.3.1 「レプリケーション PRIVILEGE_CHECKS_USER アカウントの権限」
- セクション17.3.3 「レプリケーション権限チェック」

C

[[index top](#)]

CACHE INDEX

- セクション13.7.8.2 「CACHE INDEX ステートメント」
- セクション13.7.8.5 「LOAD INDEX INTO CACHE ステートメント」
- セクション8.10.2.4 「インデックスプリロード」
- セクション13.3.3 「暗黙的なコミットを発生させるステートメント」
- セクション8.10.2.2 「複合キーキャッシュ」

CALL

- セクション13.1.2 「ALTER DATABASE ステートメント」
- セクション13.2.1 「CALL ステートメント」

セクション13.1.17 「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」
第25章 「ストアドオブジェクト」
セクション25.6 「ストアドオブジェクトのアクセス制御」
セクション25.7 「ストアドプログラムバイナリロギング」
セクション25.2.1 「ストアドルーチンの構文」
セクション25.3.1 「トリガーの構文と例」
セクション13.5 「プリペアドステートメント」

CALL p()

セクション13.6.7.4 「RESIGNAL ステートメント」

CASE

セクション13.6.5.1 「CASE ステートメント」
セクション13.6.5 「フロー制御ステートメント」
セクション12.5 「フロー制御関数」
セクション8.10.3 「プリペアドステートメントおよびストアドプログラムのキャッシュ」

CHANGE MASTER TO

2 番目のインスタンスの追加

CHANGE MASTER TO でのトランザクションのスキップ

セクション13.4.2.1 「CHANGE MASTER TO ステートメント」
セクション13.4.2.3 「CHANGE REPLICATION SOURCE TO ステートメント」
セクション13.7.1.6 「GRANT ステートメント」
GTID のないトランザクションのスキップ
セクション17.1.3.4 「GTID を使用したレプリケーションのセットアップ」
セクション17.1.3.7 「GTID ベースレプリケーションの制約」
セクション17.1.3.3 「GTID 自動配置」
セクション15.20.7 「InnoDB memcached プラグインとレプリケーション」
セクション2.11.4 「MySQL 8.0 での変更」
セクション1.3 「MySQL 8.0 の新機能」
セクション6.2.2 「MySQL で提供される権限」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
mysqldump を使用したデータスナップショットの作成
セクション23.6.5 「NDB Cluster のレプリケーションの準備」
セクション23.6.10 「NDB Cluster レプリケーション: 双方向および循環レプリケーション」
セクション23.6.9 「NDB Cluster レプリケーションによる NDB Cluster バックアップ」
セクション23.6.8 「NDB Cluster レプリケーションによるフェイルオーバーの実装」
セクション17.1.6.3 「Replica Server のオプションと変数」
セクション27.12.11.4 「replication_applier_configuration テーブル」
セクション27.12.11.1 「replication_connection_configuration テーブル」
セクション13.4.2.5 「RESET REPLICA | SLAVE ステートメント」
セクション6.4.1.2 「SHA-2 プラガブル認証のキャッシュ」
セクション6.4.1.3 「SHA-256 プラガブル認証」
セクション13.7.7.35 「SHOW REPLICA | SLAVE STATUS ステートメント」
セクション13.4.3.1 「START GROUP_REPLICATION ステートメント」
セクション13.4.2.7 「START REPLICA | SLAVE ステートメント」
セクション13.4.2.9 「STOP REPLICA | SLAVE ステートメント」
セクション18.10 「よくある質問」
セクション6.2.14 「アカウントパスワードの割り当て」
インスタンスの追加
クローニングの前提条件
クローニング操作
セクション18.9.1 「グループレプリケーションの要件」
セクション18.8 「グループレプリケーションシステム変数」
セクション17.3.3.2 「グループレプリケーションチャンネルの権限チェック」
セクション5.1.8 「サーバーシステム変数」
セクション5.1.14 「ネットワークネームスペースのサポート」
バイナリログのトランザクション圧縮のモニタリング

セクション17.1.1 「バイナリログファイルの位置ベースのレプリケーション構成の概要」
セクション27.12.11 「パフォーマンススキーマレプリケーションテーブル」
セクション17.1.3.5 「フェイルオーバーおよびスケールアウトでの GTID の使用」
セクション17.4.8 「フェイルオーバー中のソースの切替え」
セクション17.1.5.3 「マルチソースレプリカへの GTID ベースのソースの追加」
セクション17.1.5.4 「マルチソースレプリカへのバイナリログベースレプリケーションソースの追加」
セクション17.4.1.2 「レプリカからの RAW データのバックアップ」
セクション17.1.2.7 「レプリカでのソース構成の設定」
セクション17.4.2 「レプリカの予期しない停止の処理」
セクション8.14.5 「レプリケーション I/O スレッドの状態」
セクション17.3.3.1 「レプリケーション PRIVILEGE_CHECKS_USER アカウントの権限」
セクション8.14.6 「レプリケーション SQL スレッドの状態」
セクション17.1.6 「レプリケーションおよびバイナリロギングのオプションと変数」
セクション17.5.1.28 「レプリケーションとソースまたはレプリカの停止」
セクション17.5.1.34 「レプリケーションとトランザクションの非一貫性」
セクション17.3 「レプリケーションのセキュリティ」
セクション17.1.7.1 「レプリケーションステータスの確認」
セクション17.2.4.2 「レプリケーションメタデータリポジトリ」
セクション17.1.4.1 「レプリケーションモードの概念」
レプリケーションユーザー資格証明のセキュアな提供
セクション8.14.7 「レプリケーション接続スレッドの状態」
セクション17.3.3 「レプリケーション権限チェック」
セクション5.6.7.6 「レプリケーション用のクローニング」
セクション17.1.2.3 「レプリケーション用ユーザーの作成」
セクション17.2.2.2 「以前のレプリケーションステートメントとの互換性」
セクション18.2.1.3 「分散リカバリのユーザー資格証明」
セクション17.2.2.1 「単一チャンネルで操作するためのコマンド」
セクション4.2.8 「接続圧縮制御」
新しいソースおよびレプリカを使用したレプリケーションの設定
既存のデータによるレプリケーションのセットアップ
セクション6.3.2 「暗号化された接続 TLS プロトコルおよび暗号」
セクション17.3.1 「暗号化接続を使用するためのレプリケーションの設定」
セクション13.3.3 「暗黙的なコミットを発生させるステートメント」
セクション17.2.1.2 「行ベースロギングおよびレプリケーションの使用」
セクション17.4.9 「非同期接続フェイルオーバーによるソースの切替え」

CHANGE REPLICATION FILTER

セクション13.4.2.2 「CHANGE REPLICATION FILTER ステートメント」
セクション6.2.2 「MySQL で提供される権限」
セクション27.12.11.9 「replication_applier_filters テーブル」
セクション27.12.11.8 「replication_applier_global_filters テーブル」
セクション13.7.7.35 「SHOW REPLICAS | SLAVE STATUS ステートメント」
セクション5.1.10 「サーバーステータス変数」
セクション17.1.5.3 「マルチソースレプリカへの GTID ベースのソースの追加」
セクション17.1.5.4 「マルチソースレプリカへのバイナリログベースレプリケーションソースの追加」
セクション17.2.5.4 「レプリケーションチャンネルベースのフィルタ」

CHANGE REPLICATION FILTER REPLICATE_DO_DB

セクション17.1.6.3 「Replica Server のオプションと変数」

CHANGE REPLICATION FILTER REPLICATE_DO_TABLE

セクション17.1.6.3 「Replica Server のオプションと変数」

CHANGE REPLICATION FILTER REPLICATE_IGNORE_DB

セクション17.1.6.3 「Replica Server のオプションと変数」

CHANGE REPLICATION FILTER REPLICATE_IGNORE_TABLE

セクション17.1.6.3 「Replica Server のオプションと変数」

CHANGE REPLICATION FILTER REPLICATE_REWRITE_DB

セクション17.1.6.3「Replica Server のオプションと変数」

CHANGE REPLICATION FILTER REPLICATE_WILD_DO_TABLE

セクション17.1.6.3「Replica Server のオプションと変数」

CHANGE REPLICATION FILTER REPLICATE_WILD_IGNORE_TABLE

セクション17.1.6.3「Replica Server のオプションと変数」

CHANGE REPLICATION SOURCE TO

2 番目のインスタンスの追加

セクション13.4.2.1「CHANGE MASTER TO ステートメント」

セクション13.4.2.3「CHANGE REPLICATION SOURCE TO ステートメント」

セクション13.7.1.6「GRANT ステートメント」

GTID のあるトランザクションのスキップ

セクション17.1.3.6「GTID のないソースから GTID のあるレプリカへのレプリケーション」

GTID のないトランザクションのスキップ

セクション17.1.3.4「GTID を使用したレプリケーションのセットアップ」

セクション17.1.3.7「GTID ベースレプリケーションの制約」

セクション17.1.3.3「GTID 自動配置」

セクション15.20.7「InnoDB memcached プラグインとレプリケーション」

セクション6.2.2「MySQL で提供される権限」

セクション4.6.8「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」

セクション4.5.4「mysqldump — データベースバックアッププログラム」

mysqldump を使用したデータスナップショットの作成

セクション23.6.5「NDB Cluster のレプリケーションの準備」

セクション23.6.10「NDB Cluster レプリケーション: 双方向および循環レプリケーション」

セクション23.6.9「NDB Cluster レプリケーションによる NDB Cluster バックアップ」

セクション23.6.8「NDB Cluster レプリケーションによるフェイルオーバーの実装」

セクション17.1.6.3「Replica Server のオプションと変数」

セクション27.12.11.4「replication_applier_configuration テーブル」

セクション27.12.11.1「replication_connection_configuration テーブル」

セクション27.12.11.2「replication_connection_status テーブル」

セクション13.4.2.5「RESET REPLICAS | SLAVE ステートメント」

セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」

セクション6.4.1.3「SHA-256 プラガブル認証」

セクション13.7.7.35「SHOW REPLICAS | SLAVE STATUS ステートメント」

セクション13.4.3.1「START GROUP_REPLICATION ステートメント」

セクション13.4.2.7「START REPLICAS | SLAVE ステートメント」

セクション13.4.2.9「STOP REPLICAS | SLAVE ステートメント」

セクション18.10「よくある質問」

セクション6.2.14「アカウントパスワードの割り当て」

インスタンスの追加

クローニングの前提条件

クローニング操作

セクション18.9.1「グループレプリケーションの要件」

セクション18.8「グループレプリケーションシステム変数」

セクション17.3.3.2「グループレプリケーションチャンネルの権限チェック」

セクション5.1.8「サーバーシステム変数」

セクション17.1.7.3「トランザクションのスキップ」

セクション5.1.14「ネットワークネームスペースのサポート」

バイナリログのトランザクション圧縮のモニタリング

セクション17.1.1「バイナリログファイルの位置ベースのレプリケーション構成の概要」

セクション27.12.11「パフォーマンススキーマレプリケーションテーブル」

セクション17.1.3.5「フェイルオーバーおよびスケールアウトでの GTID の使用」

セクション17.4.8「フェイルオーバー中のソースの切替え」

セクション17.1.5.3「マルチソースレプリカへの GTID ベースのソースの追加」

セクション17.1.5.4「マルチソースレプリカへのバイナリログベースレプリケーションソースの追加」

セクション17.4.1.2 「レプリカからの RAW データのバックアップ」
セクション17.1.2.7 「レプリカでのソース構成の設定」
セクション17.4.2 「レプリカの予期しない停止の処理」
セクション8.14.5 「レプリケーション I/O スレッドの状態」
セクション17.3.3.1 「レプリケーション PRIVILEGE_CHECKS_USER アカウントの権限」
セクション8.14.6 「レプリケーション SQL スレッドの状態」
セクション17.1.6 「レプリケーションおよびバイナリロギングのオプションと変数」
セクション17.5.1.28 「レプリケーションとソースまたはレプリカの停止」
セクション17.5.1.34 「レプリケーションとトランザクションの非一貫性」
セクション17.3 「レプリケーションのセキュリティ」
セクション17.2.4.2 「レプリケーションメタデータリポジトリ」
セクション17.1.4.1 「レプリケーションモードの概念」
レプリケーションユーザー資格証明のセキュアな提供
セクション8.14.7 「レプリケーション接続スレッドの状態」
セクション17.3.3 「レプリケーション権限チェック」
セクション5.6.7.6 「レプリケーション用のクローニング」
セクション17.1.2.3 「レプリケーション用ユーザーの作成」
セクション17.2.2.2 「以前のレプリケーションステートメントとの互換性」
セクション18.2.1.3 「分散リカバリのユーザー資格証明」
セクション17.2.2.1 「単一チャンネルで操作するためのコマンド」
セクション4.2.8 「接続圧縮制御」
新しいソースおよびレプリカを使用したレプリケーションの設定
既存のデータによるレプリケーションのセットアップ
セクション6.3.2 「暗号化された接続 TLS プロトコルおよび暗号」
セクション17.3.1 「暗号化接続を使用するためのレプリケーションの設定」
セクション13.3.3 「暗黙的なコミットを発生させるステートメント」
セクション17.2.1.2 「行ベースロギングおよびレプリケーションの使用」
セクション17.4.9 「非同期接続フェイルオーバーによるソースの切替え」

CHANGE REPLICATION SOURCE TO SOURCE_DELAY=N

セクション17.4.11 「遅延レプリケーション」

CHECK PARTITION

セクション15.12.1 「オンライン DDL 操作」

CHECK TABLE

セクション13.1.9.1 「ALTER TABLE パーティション操作」
セクション16.5 「ARCHIVE ストレージエンジン」
セクション13.7.3.2 「CHECK TABLE ステートメント」
セクション13.1.17 「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」
セクション13.1.23 「CREATE VIEW ステートメント」
セクション16.4.1 「CSV テーブルの修復と確認」
セクション4.6.2 「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティ」
セクション15.21 「InnoDB のトラブルシューティング」
セクション15.18.2 「InnoDB のリカバリ」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション16.7 「MERGE ストレージエンジン」
セクション7.6 「MyISAM テーブルの保守とクラッシュリカバリ」
セクション7.6.3 「MyISAM テーブルの修復方法」
セクション16.2.4.1 「MyISAM テーブルの破損」
セクション7.6.5 「MyISAM テーブル保守スケジュールのセットアップ」
セクション4.6.4 「myisamchk — MyISAM テーブルメンテナンスユーティリティ」
セクションA.6 「MySQL 8.0 FAQ: ビュー」
セクション1.3 「MySQL 8.0 の新機能」
セクション2.11.3 「MySQL のアップグレードプロセスの内容」
セクションB.3.2.7 「MySQL サーバーが存在しなくなりました」
セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
NDB Cluster の新しいバージョンへの NDB バックアップの復元

セクション5.1.8「サーバーシステム変数」
セクション13.6.6.5「サーバー側のカーソルの制約」
セクション25.8「ストアプログラムの制約」
セクション5.4.5「スロークエリーログ」
セクション2.11.13「テーブルまたはインデックスの再作成または修復」
セクション24.6「パーティショニングの制約と制限」
セクション24.3.4「パーティションの保守」
セクション25.9「ビューの制約」
セクション5.4.1「一般クエリーログおよびスロークエリーログの出力先の選択」
セクション8.11.5「外部ロック」
セクション13.3.3「暗黙的なコミットを発生させるステートメント」
セクション1.7.1「標準 SQL に対する MySQL 拡張機能」
セクション1.6「質問またはバグをレポートする方法」
セクション16.2.4.2「適切に閉じられなかったテーブルの問題」

CHECK TABLE ... EXTENDED

セクション13.7.3.2「CHECK TABLE ステートメント」

CHECK TABLE ... FOR UPGRADE

セクション13.7.3.2「CHECK TABLE ステートメント」

セクション13.7.3.5「REPAIR TABLE ステートメント」

CHECK TABLE FOR UPGRADE

セクション14.6「シリアライズディクショナリ情報 (SDI)」

CHECK TABLE QUICK

セクション13.7.3.2「CHECK TABLE ステートメント」

CHECKSUM TABLE

セクション13.7.3.3「CHECKSUM TABLE ステートメント」

セクション13.1.20「CREATE TABLE ステートメント」

セクション17.5.1.4「レプリケーションと CHECKSUM TABLE」

CHECKSUM TABLE ... QUICK

セクション13.7.3.3「CHECKSUM TABLE ステートメント」

CLONE

セクション13.7.5「CLONE ステートメント」

セクション5.6.7.9「クローニング操作の監視」

セクション5.6.7.3「リモートデータのクローニング」

セクション5.6.7.2「ローカルでのデータのクローニング」

CLONE INSTANCE

セクション13.7.5「CLONE ステートメント」

セクション5.6.7.9「クローニング操作の監視」

セクション5.6.7.13「クローンプラグインの制限事項」

セクション5.6.7.8「リモートクローニング操作の失敗処理」

セクション5.6.7.3「リモートデータのクローニング」

セクション5.6.7.4「暗号化データのクローニング」

CLONE LOCAL

セクション5.6.7.9「クローニング操作の監視」

CLONE LOCAL DATA DIRECTORY

セクション13.7.5「CLONE ステートメント」

セクション5.6.7.2「ローカルでのデータのクローニング」

COALESCE PARTITION

セクション15.12.1「オンライン DDL 操作」

COMMIT

セクション13.1.17「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」

セクション27.12.7.1「events_transactions_current テーブル」

セクション26.38「INFORMATION_SCHEMA TABLES テーブル」

セクション15.2「InnoDB および ACID モデル」

セクション15.7.3「InnoDB のさまざまな SQL ステートメントで設定されたロック」

セクション8.5.3「InnoDB の読み取り専用トランザクションの最適化」

セクション15.14「InnoDB の起動オプションおよびシステム変数」

セクション8.5.5「InnoDB テーブルの一括データロード」

セクション15.6.1.5「MyISAM から InnoDB へのテーブルの変換」

セクション4.5.4「mysqldump — データベースバックアッププログラム」

NDB Cluster システム変数

セクション17.1.6.3「Replica Server のオプションと変数」

セクション13.3.4「SAVEPOINT、ROLLBACK TO SAVEPOINT および RELEASE SAVEPOINT ステートメント」

セクション13.7.7.38「SHOW TABLE STATUS ステートメント」

セクション13.3.1「START TRANSACTION、COMMIT および ROLLBACK ステートメント」

セクション13.1.1「アトミックデータ定義ステートメントのサポート」

セクション5.1.18「クライアントセッション状態の変更のサーバートラッキング」

セクション5.1.8「サーバースystem変数」

セクション5.1.10「サーバーステータス変数」

セクション25.7「ストアードプログラムバイナリロギング」

データ定義ステートメント

セクション13.3「トランザクションステートメントおよびロックステートメント」

セクション25.3.1「トリガーの構文と例」

セクション5.4.4「バイナリログ」

セクション27.12.7「パフォーマンススキーマのトランザクションテーブル」

ライタのクエリーリライトプロシージャおよび関数

セクション17.5.1.35「レプリケーションとトランザクション」

セクション17.1.2.4「レプリケーションソースのバイナリログ座標の取得」

セクション13.3.3「暗黙的なコミットを発生させるステートメント」

セクション15.7.2.2「自動コミット、コミットおよびロールバック」

COMMIT AND CHAIN

セクション5.1.18「クライアントセッション状態の変更のサーバートラッキング」

セクション27.12.7「パフォーマンススキーマのトランザクションテーブル」

COMPRESSION

セクション15.13「InnoDB 保存データ暗号化」

CREATE DATABASE

セクション13.1.2「ALTER DATABASE ステートメント」

セクション13.1.12「CREATE DATABASE ステートメント」

セクション4.6.8「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」

セクション4.5.4「mysqldump — データベースバックアッププログラム」

セクション7.4.1「mysqldump による SQL フォーマットでのデータのダンプ」

セクション4.5.6「mysqlpump — データベースバックアッププログラム」

セクション23.1.4「NDB Cluster の新機能」

セクション23.6.9「NDB Cluster レプリケーションによる NDB Cluster バックアップ」

セクション23.1.7.8「NDB Cluster 専用の問題」

セクション17.1.6.3「Replica Server のオプションと変数」

セクション13.7.7.6「SHOW CREATE DATABASE ステートメント」

セクション7.4.2「SQL フォーマットバックアップのリロード」

セクション8.12.2.1「Unix 上のデータベースへのシンボリックリンクの使用」

セクション10.5「アプリケーションの文字セットおよび照合順序の構成」
セクション17.2.5「サーバーがレプリケーションフィルタリングルールをどのように評価するか」
セクション5.1.8「サーバーシステム変数」
セクション10.3.2「サーバー文字セットおよび照合順序」
セクション7.4.5.2「サーバー間でのデータベースのコピー」
セクション17.2.5.1「データベースレベルレプリケーションオプションおよびバイナリロギングオプションの評価」
セクション10.3.3「データベース文字セットおよび照合順序」
セクション7.1「バックアップとリカバリの種類」
セクション27.6「パフォーマンススキーマインストールメント命名規則」
セクション13.3.3「暗黙的なコミットを発生させるステートメント」
セクション1.7.1「標準 SQL に対する MySQL 拡張機能」
セクション9.2.3「識別子の大きい文字と小さい文字の区別」

CREATE DATABASE IF NOT EXISTS

セクション17.5.1.6「CREATE ... IF NOT EXISTS ステートメントのレプリケーション」

CREATE EVENT

セクション13.1.2「ALTER DATABASE ステートメント」
セクション13.1.3「ALTER EVENT ステートメント」
セクション13.1.5「ALTER INSTANCE ステートメント」
セクション13.1.13「CREATE EVENT ステートメント」
セクション17.5.1.8「CURRENT_USER() のレプリケーション」
セクション26.14「INFORMATION_SCHEMA EVENTS テーブル」
セクション4.5.4「mysqldump — データベースバックアッププログラム」
セクション4.5.6「mysqlpump — データベースバックアッププログラム」
セクション13.7.7.7「SHOW CREATE EVENT ステートメント」
セクション13.7.7.18「SHOW EVENTS ステートメント」
セクション25.4.6「イベントスケジューラと MySQL 権限」
セクション25.4.4「イベントメタデータ」
セクション25.4.3「イベント構文」
セクション5.1.7「サーバーコマンドオプション」
第25章「ストアオブジェクト」
セクション25.8「ストアプログラムの制約」
セクション25.7「ストアプログラムバイナリロギング」
セクション17.3.2.3「バイナリログマスターキーのローテーション」
セクション17.5.1.16「呼び出される機能のレプリケーション」
セクション9.5「式」
セクション12.16「情報関数」
セクション13.3.3「暗黙的なコミットを発生させるステートメント」

CREATE EVENT IF NOT EXISTS

セクション17.5.1.6「CREATE ... IF NOT EXISTS ステートメントのレプリケーション」

CREATE FULLTEXT INDEX

セクション8.5.5「InnoDB テーブルの一括データロード」

CREATE FUNCTION

セクション13.1.2「ALTER DATABASE ステートメント」
セクション13.1.4「ALTER FUNCTION ステートメント」
セクション13.1.14「CREATE FUNCTION ステートメント」
セクション13.1.17「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」
セクション17.5.1.8「CURRENT_USER() のレプリケーション」
セクション15.14「InnoDB の起動オプションおよびシステム変数」
セクション6.5.2「MySQL Enterprise Data Masking and De-Identification のインストールまたはアンインストール」
セクション6.6.1「MySQL Enterprise Encryption のインストール」
セクション1.8.1「MySQL への貢献者」
セクション5.3「mysql システムスキーマ」
セクション4.5.4「mysqldump — データベースバックアッププログラム」

セクション4.5.6「mysqldump — データベースバックアッププログラム」
セクション27.12.19.12「user_defined_functions テーブル」
セクション2.11.12「アップグレードのトラブルシューティング」
セクション5.1.7「サーバーコマンドオプション」
第25章「ストアオブジェクト」
セクション25.8「ストアプログラムの制約」
セクション25.7「ストアプログラムバイナリロギング」
セクション25.2.1「ストアルーチンの構文」
セクション5.6.6.2「バージョントークンのインストールまたはアンインストール」
セクション27.12.19「パフォーマンススキーマのその他のテーブル」
セクション13.7.4.2「ユーザー定義関数に対する DROP FUNCTION ステートメント」
セクション5.7.1「ユーザー定義関数のインストールおよびアンインストール」
セクション5.7.2「ユーザー定義関数情報の取得」
セクション13.7.4.1「ユーザー定義関数用の CREATE FUNCTION ステートメント」
ロックサービス UDF インタフェース
セクション17.5.1.16「呼び出される機能のレプリケーション」
セクション12.16「情報関数」
セクション13.3.3「暗黙的なコミットを発生させるステートメント」
セクション6.4.4.10「汎用キーリングキー管理関数」
セクション9.2.5「関数名の構文解析と解決」

CREATE INDEX

セクション13.1.2「ALTER DATABASE ステートメント」
セクション13.1.15「CREATE INDEX ステートメント」
セクション13.1.20「CREATE TABLE ステートメント」
セクション26.34「INFORMATION_SCHEMA STATISTICS テーブル」
セクション15.6.2.4「InnoDB FULLTEXT インデックス」
セクション15.20.6.4「InnoDB memcached プラグインのトランザクション動作の制御」
セクション15.14「InnoDB の起動オプションおよびシステム変数」
セクション15.9.1.5「InnoDB テーブルでの圧縮の動作」
セクション12.10.9「MeCab フルテキストパーサープラグイン」
セクション8.7「MEMORY テーブルの最適化」
MySQL 用語集
セクション23.5.11「NDB Cluster での ALTER TABLE を使用したオンライン操作」
セクション23.1.7.6「NDB Cluster でサポートされない機能または欠落している機能」
セクション12.10.8「ngram 全文パーサー」
セクション13.7.7.22「SHOW INDEX ステートメント」
セクション15.8.11「インデックスページのマージしきい値の構成」
セクション15.12.1「オンライン DDL 操作」
セクション8.3.5「カラムインデックス」
セクション12.11「キャスト関数と演算子」
セクション5.1.8「サーバーシステム変数」
セクション5.4.5「スロークエリロギング」
セクション8.3.12「不可視のインデックス」
セクション12.10「全文検索関数」
セクション13.3.3「暗黙的なコミットを発生させるステートメント」
セクション11.4.10「空間インデックスの作成」

CREATE LOGFILE GROUP

セクション13.1.6「ALTER LOGFILE GROUP ステートメント」
セクション13.1.16「CREATE LOGFILE GROUP ステートメント」
セクション13.1.21「CREATE TABLESPACE ステートメント」
セクション26.15「INFORMATION_SCHEMA FILES テーブル」
セクション4.5.4「mysqldump — データベースバックアッププログラム」
セクション23.5.10.1「NDB Cluster データオブジェクト」
セクション23.3.3.6「NDB Cluster データノードの定義」
セクション23.1.7.8「NDB Cluster 専用の問題」
セクション23.5.14.42「ndbinfo resources テーブル」
セクション23.3.3.13「データノードのメモリー管理」

CREATE OR REPLACE VIEW

[セクション13.1.23「CREATE VIEW ステートメント」](#)
[セクション25.9「ビューの制約」](#)

CREATE PROCEDURE

[セクション13.1.2「ALTER DATABASE ステートメント」](#)
[セクション13.1.7「ALTER PROCEDURE ステートメント」](#)
[セクション13.2.1「CALL ステートメント」](#)
[セクション13.1.17「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」](#)
[セクション17.5.1.8「CURRENT_USER\(\) のレプリケーション」](#)
[セクション4.5.4「mysqldump — データベースバックアッププログラム」](#)
[セクション4.5.6「mysqlpump — データベースバックアッププログラム」](#)
[第25章「ストアドオブジェクト」](#)
[セクション25.8「ストアドプログラムの制約」](#)
[セクション25.7「ストアドプログラムバイナリロギング」](#)
[セクション25.2.1「ストアドルーチンの構文」](#)
[セクション17.5.1.16「呼び出される機能のレプリケーション」](#)
[セクション12.16「情報関数」](#)
[セクション13.3.3「暗黙的なコミットを発生させるステートメント」](#)

CREATE RESOURCE GROUP

[セクション13.7.2.1「ALTER RESOURCE GROUP ステートメント」](#)
[セクション13.7.2.2「CREATE RESOURCE GROUP ステートメント」](#)
[セクション5.1.16「リソースグループ」](#)

CREATE ROLE

[セクション13.7.1.2「CREATE ROLE ステートメント」](#)
[セクション6.2.2「MySQL で提供される権限」](#)
[セクション6.2.10「ロールの使用」](#)
[セクション13.3.3「暗黙的なコミットを発生させるステートメント」](#)

CREATE SCHEMA

[セクション13.1.12「CREATE DATABASE ステートメント」](#)
[セクション15.13「InnoDB 保存データ暗号化」](#)
[セクション23.6.9「NDB Cluster レプリケーションによる NDB Cluster バックアップ」](#)
[セクション23.1.7.8「NDB Cluster 専用の問題」](#)

CREATE SERVER

[セクション13.1.8「ALTER SERVER ステートメント」](#)
[セクション16.8.2.2「CREATE SERVER を使用した FEDERATED テーブルの作成」](#)
[セクション17.5.1.5「CREATE SERVER、ALTER SERVER、および DROP SERVER のレプリケーション」](#)
[セクション16.8.3「FEDERATED ストレージエンジンの注記とヒント」](#)
[セクション16.8.2「FEDERATED テーブルの作成方法」](#)
[セクション13.7.8.3「FLUSH ステートメント」](#)
[セクション6.2.2「MySQL で提供される権限」](#)
[セクション8.12.3.1「MySQL のメモリーの使用方法」](#)
[セクション13.1.1「アトミックデータ定義ステートメントのサポート」](#)
[セクション13.3.3「暗黙的なコミットを発生させるステートメント」](#)

CREATE SPATIAL REFERENCE SYSTEM

[セクション13.1.19「CREATE SPATIAL REFERENCE SYSTEM ステートメント」](#)
[セクション26.36「INFORMATION_SCHEMA ST_SPATIAL_REFERENCE_SYSTEMS テーブル」](#)
[セクション13.3.3「暗黙的なコミットを発生させるステートメント」](#)
[セクション11.4.5「空間参照システムのサポート」](#)

CREATE TABLE

[セクション13.1.2「ALTER DATABASE ステートメント」](#)

セクション13.1.9.3「ALTER TABLE の例」
セクション13.1.9「ALTER TABLE ステートメント」
セクション13.1.9.1「ALTER TABLE パーティション操作」
セクション16.5「ARCHIVE ストレージエンジン」
セクション3.6.9「AUTO_INCREMENT の使用」
セクション13.1.20.6「CHECK 制約」
セクション16.8.2.1「CONNECTION を使用した FEDERATED テーブルの作成」
セクション13.1.13「CREATE EVENT ステートメント」
セクション13.1.15「CREATE INDEX ステートメント」
セクション13.1.18「CREATE SERVER ステートメント」
セクション13.1.20.3「CREATE TABLE ... LIKE ステートメント」
セクション13.1.20.4「CREATE TABLE ... SELECT ステートメント」
セクション17.5.1.7「CREATE TABLE ... SELECT ステートメントのレプリケーション」
セクション13.1.20.8「CREATE TABLE および生成されるカラム」
セクション13.1.20「CREATE TABLE ステートメント」
セクション13.1.21「CREATE TABLESPACE ステートメント」
セクション13.1.20.2「CREATE TEMPORARY TABLE ステートメント」
セクション5.6.5「ddl_rewriter プラグイン」
セクション11.3.5「ENUM 型」
セクション15.6.3.2「File-Per-Table テーブルスペース」
セクション17.3.2「FOREIGN KEY の制約」
セクション13.1.20.5「FOREIGN KEY の制約」
セクション17.1.3.2「GTID ライフサイクル」
セクション17.1.3.1「GTID 形式および格納」
セクション24.2.4「HASH パーティショニング」
セクション13.8.3「HELP ステートメント」
セクション26.51.24「INFORMATION_SCHEMA INNODB_TABLES テーブル」
セクション26.21「INFORMATION_SCHEMA PARTITIONS テーブル」
セクション26.34「INFORMATION_SCHEMA STATISTICS テーブル」
セクション26.38「INFORMATION_SCHEMA TABLES テーブル」
セクション8.5.7「InnoDB DDL 操作の最適化」
セクション15.6.2.4「InnoDB FULLTEXT インデックス」
セクション15.15.3「InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル」
セクション15.19「InnoDB と MySQL レプリケーション」
セクション15.8.10「InnoDB のオプティマイザ統計の構成」
セクション15.10「InnoDB の行フォーマット」
セクション15.14「InnoDB の起動オプションおよびシステム変数」
セクション15.1.4「InnoDB を使用したテストおよびベンチマーク」
セクション15.9.1.5「InnoDB テーブルでの圧縮の動作」
セクション15.8.10.3「InnoDB テーブルに対する ANALYZE TABLE の複雑さの推定」
セクション15.6.1.3「InnoDB テーブルのインポート」
セクション15.1.2「InnoDB テーブルのベストプラクティス」
セクション15.6.1.1「InnoDB テーブルの作成」
セクション15.9.1「InnoDB テーブルの圧縮」
セクション15.9.2「InnoDB ページ圧縮」
セクション15.13「InnoDB 保存データ暗号化」
セクション15.1「InnoDB 入門」
セクション12.18.6「JSON テーブル関数」
セクション24.2.5「KEY パーティショニング」
セクション24.2.2「LIST パーティショニング」
セクション13.2.8「LOAD XML ステートメント」
セクション13.3.6「LOCK TABLES および UNLOCK TABLES ステートメント」
セクション12.10.9「MeCab フルテキストパーサープラグイン」
セクション16.3「MEMORY ストレージエンジン」
セクション15.6.1.5「MyISAM から InnoDB へのテーブルの変換」
セクション16.2「MyISAM ストレージエンジン」
セクション16.2.3「MyISAM テーブルのストレージフォーマット」
セクションA.10「MySQL 8.0 FAQ: NDB Cluster」
セクション2.11.4「MySQL 8.0 での変更」
セクション1.3「MySQL 8.0 の新機能」

セクション8.4.4 「MySQL での内部一時テーブルの使用」
セクション6.2.2 「MySQL で提供される権限」
セクション24.1 「MySQL のパーティショニングの概要」
セクション8.12.3.1 「MySQL のメモリーの使用方法」
セクション4.5.1.1 「mysql クライアントオプション」
セクション5.4.4.4 「mysql データベーステーブルへの変更に対するロギング形式」
セクション24.2.7 「MySQL パーティショニングによる NULL の扱い」
MySQL 用語集
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション7.4.3 「mysqldump による区切りテキストフォーマットでのデータのダンプ」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション23.5.13 「NDB API 統計のカウントと変数」
セクション23.5.11 「NDB Cluster での ALTER TABLE を使用したオンライン操作」
セクション23.1.7.3 「NDB Cluster でのトランザクション処理に関する制限」
セクション23.1.7.6 「NDB Cluster でサポートされない機能または欠落している機能」
セクション23.1.7.1 「NDB Cluster の SQL 構文に準拠していません」
セクション23.2.3 「NDB Cluster の初期構成」
セクション23.1.4 「NDB Cluster の新機能」
NDB Cluster システム変数
セクション23.5.10.1 「NDB Cluster ディスクデータオブジェクト」
セクション23.3.3.6 「NDB Cluster データノードの定義」
セクション23.1.7.5 「NDB Cluster 内のデータベースオブジェクトに関連付けられる制限」
セクション23.1.7.8 「NDB Cluster 専用の問題」
セクション23.3.3.1 「NDB Cluster 構成: 基本例」
NDB バックアップを以前のバージョンの NDB Cluster に復元
セクション23.4.6 「ndb_blob_tool — NDB Cluster テーブルの BLOB および TEXT カラムのチェックおよび修復」
セクション23.4.9 「ndb_desc — NDB テーブルの表示」
セクション12.10.8 「ngram 全文パーサー」
セクション24.2.3.1 「RANGE COLUMNS パーティショニング」
セクション24.3.1 「RANGE および LIST パーティションの管理」
セクション24.2.1 「RANGE パーティショニング」
セクション13.2.9 「REPLACE ステートメント」
セクション13.7.7.5 「SHOW COLUMNS ステートメント」
セクション13.7.7.10 「SHOW CREATE TABLE ステートメント」
セクション13.7.7.15 「SHOW ENGINE ステートメント」
セクション13.7.7.22 「SHOW INDEX ステートメント」
セクション13.7.7.38 「SHOW TABLE STATUS ステートメント」
セクション13.7.7.42 「SHOW WARNINGS ステートメント」
セクション15.9.1.7 「SQL 圧縮構文の警告とエラー」
セクション13.1.37 「TRUNCATE TABLE ステートメント」
セクション13.7.4.6 「UNINSTALL PLUGIN ステートメント」
セクション8.12.2.2 「Unix 上の MyISAM へのシンボリックリンクの使用」
セクション2.3.7 「Windows プラットフォームの制限事項」
セクション13.2.15 「WITH (共通テーブル式)」
セクション13.1.1 「アトミックデータ定義ステートメントのサポート」
セクション15.8.11 「インデックスページのマージしきい値の構成」
セクションB.2 「エラー情報インタフェース」
セクション15.12.1 「オンライン DDL 操作」
セクション8.3.5 「カラムインデックス」
セクション10.3.5 「カラム文字セットおよび照合順序」
セクション12.11 「キャスト関数と演算子」
セクション24.2.6 「サブパーティショニング」
セクション5.1.11 「サーバー SQL モード」
セクション5.1.7 「サーバーコマンドオプション」
セクション5.1.8 「サーバーシステム変数」
セクション8.12.2 「シンボリックリンクの使用」
セクション13.2.11.1 「スカラーオペランドとしてのサブクエリー」
セクション24.6.2 「ストレージエンジンに関連するパーティショニング制限」
セクション16.1 「ストレージエンジンの設定」
ソースまたはレプリカにカラムが多いレプリケーション

セクション3.3.2 「テーブルの作成」
セクション3.3.3 「テーブルへのデータのロード」
セクション8.4.6 「テーブルサイズの制限」
セクション15.6.3.9 「テーブルスペースの AUTOEXTEND_SIZE 構成」
セクション15.9.1.1 「テーブル圧縮の概要」
セクション10.3.4 「テーブル文字セットおよび照合順序」
セクション8.4.1 「データサイズの最適化」
セクション3.4 「データベースとテーブルに関する情報の取得」
セクション7.2 「データベースバックアップ方法」
セクション10.3.3 「データベース文字セットおよび照合順序」
セクション17.1.6.4 「バイナリロギングのオプションと変数」
セクション5.4.4.2 「バイナリログ形式の設定」
セクション7.1 「バックアップとリカバリの種類」
セクション7.4 「バックアップへの mysqldump の使用」
セクション24.6 「パーティショニングの制約と制限」
セクション24.6.1 「パーティショニングキー、主キー、および一意キー」
セクション24.2 「パーティショニングタイプ」
セクション24.3 「パーティション管理」
セクション17.5.1.1 「レプリケーションと AUTO_INCREMENT」
セクション17.5.1.10 「レプリケーションと DIRECTORY テーブルオプション」
セクション17.5.1.14 「レプリケーションとシステム関数」
セクション17.5.1.3 「レプリケーションと文字セット」
セクション17.2.5.3 「レプリケーションフィルタリングオプション間の相互作用」
セクション5.4.1 「一般クエリログおよびスロークエリログの出力先の選択」
セクション15.6.3.3 「一般テーブルスペース」
セクション8.3.12 「不可視のインデックス」
第16章 「代替ストレージエンジン」
個々のテーブルの最適化統計パラメータの構成
セクション12.10 「全文検索関数」
セクション7.4.4 「区切りテキストフォーマットバックアップのリロード」
セクション15.9.1.2 「圧縮テーブルの作成」
セクション12.16 「情報関数」
セクション11.3.1 「文字列データ型の構文」
セクション13.1.20.7 「暗黙のカラム指定の変更」
セクション13.3.3 「暗黙的なコミットを発生させるステートメント」
セクション1.7.1 「標準 SQL に対する MySQL 拡張機能」
セクション17.4.4 「異なるソースおよびレプリカのストレージエンジンでのレプリケーションの使用」
セクション11.4.10 「空間インデックスの作成」
セクション11.4.6 「空間カラムの作成」
セクション3.3.4.9 「複数のテーブルの使用」
セクション9.2.3 「識別子の大文字と小文字の区別」
セクション24.6.3 「関数に関連するパーティショニング制限」
セクション13.1.20.10 「非表示カラム」

CREATE TABLE ... DATA DIRECTORY

セクション15.6.3.6 「サーバーがオフラインのときのテーブルスペースファイルの移動」

CREATE TABLE ... ENCRYPTION

セクション13.1.5 「ALTER INSTANCE ステートメント」

CREATE TABLE ... LIKE

セクション13.1.15 「CREATE INDEX ステートメント」
セクション13.1.20.3 「CREATE TABLE ... LIKE ステートメント」
セクション13.1.20.8 「CREATE TABLE および生成されるカラム」
セクション13.3.6 「LOCK TABLES および UNLOCK TABLES ステートメント」
セクション16.7 「MERGE ストレージエンジン」
セクション11.6 「データ型デフォルト値」
セクション17.5.1.1 「レプリケーションと AUTO_INCREMENT」
セクション13.1.20.10 「非表示カラム」

CREATE TABLE ... ROW_FORMAT=COMPRESSED

セクション2.11.4「MySQL 8.0 での変更」

CREATE TABLE ... SELECT

セクション13.1.20.4「CREATE TABLE ... SELECT ステートメント」
セクション17.5.1.7「CREATE TABLE ... SELECT ステートメントのレプリケーション」
セクション13.1.20.8「CREATE TABLE および生成されるカラム」
セクション13.1.23「CREATE VIEW ステートメント」
セクション17.1.3.7「GTID ベースレプリケーションの制約」
セクション17.1.3.2「GTID ライフサイクル」
セクション1.3「MySQL 8.0 の新機能」
セクションB.3.7「MySQL の既知の問題」
セクション5.4.4.4「mysql データベーステーブルへの変更に対するロギング形式」
セクション1.7.2.1「SELECT INTO TABLE の違い」
セクション13.2.12「TABLE ステートメント」
セクション13.2.14「VALUES ステートメント」
セクション13.1.1「アトミックデータ定義ステートメントのサポート」
セクション12.11「キャスト関数と演算子」
セクション17.1.6.5「グローバルトランザクション ID システム変数」
セクション5.1.11「サーバー SQL モード」
セクション17.2.1.1「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」
セクション25.7「ストアドプログラムバイナリロギング」
セクション11.6「データ型デフォルト値」
セクション15.7.2.3「一貫性非ロック読み取り」
セクション13.3.3「暗黙的なコミットを発生させるステートメント」
セクション13.1.20.10「非表示カラム」

CREATE TABLE ... SELECT ...

セクション15.7.3「InnoDB のさまざまな SQL ステートメントで設定されたロック」
セクション23.1.4「NDB Cluster の新機能」
セクション24.3.1「RANGE および LIST パーティションの管理」

CREATE TABLE ... TABLESPACE

セクション13.1.9「ALTER TABLE ステートメント」
セクション13.1.21「CREATE TABLESPACE ステートメント」
セクション15.6.1.1「InnoDB テーブルの作成」
セクション15.6.1.2「外部でのテーブルの作成」

CREATE TABLE dst_tbl LIKE src_tbl

セクション14.7「データディクショナリの使用法の違い」

CREATE TABLE IF NOT EXISTS

セクション17.5.1.6「CREATE ... IF NOT EXISTS ステートメントのレプリケーション」

CREATE TABLE IF NOT EXISTS ... LIKE

セクション17.5.1.6「CREATE ... IF NOT EXISTS ステートメントのレプリケーション」

CREATE TABLE IF NOT EXISTS ... SELECT

セクション17.5.1.6「CREATE ... IF NOT EXISTS ステートメントのレプリケーション」

CREATE TABLE LIKE

セクション23.4.13「ndb_import — NDB への CSV データのインポート」

CREATE TABLE new_table SELECT ... FROM old_table ...

セクション13.1.20.4「CREATE TABLE ... SELECT ステートメント」
セクション13.2.10「SELECT ステートメント」

CREATE TABLE tbl_name ... TABLESPACE tablespace_name

セクション13.1.21「CREATE TABLESPACE ステートメント」

MySQL 用語集

セクション15.6.3.3「一般テーブルスペース」

CREATE TABLE ts VALUES ROW()

セクション13.2.11「サブクエリー」

CREATE TABLE...AS SELECT

セクション8.2.1「SELECT ステートメントの最適化」

CREATE TABLESPACE

セクション13.1.10「ALTER TABLESPACE ステートメント」

セクション13.1.20「CREATE TABLE ステートメント」

セクション13.1.21「CREATE TABLESPACE ステートメント」

セクション13.1.33「DROP TABLESPACE ステートメント」

セクション26.15「INFORMATION_SCHEMA FILES テーブル」

セクション26.51.24「INFORMATION_SCHEMA INNODB_TABLES テーブル」

セクション15.13「InnoDB 保存データ暗号化」

セクション1.3「MySQL 8.0 の新機能」

MySQL 用語集

セクション4.5.4「mysqldump — データベースバックアッププログラム」

セクション4.5.6「mysqlpump — データベースバックアッププログラム」

セクション23.5.10.1「NDB Cluster ディスクデータオブジェクト」

セクション23.3.3.6「NDB Cluster データノードの定義」

セクション23.1.7.8「NDB Cluster 専用の問題」

セクション13.1.1「アトミックデータ定義ステートメントのサポート」

セクション5.1.8「サーバーシステム変数」

セクション15.6.3.9「テーブルスペースの AUTOEXTEND_SIZE 構成」

セクション15.11.2「ファイル領域管理」

セクション15.6.3.3「一般テーブルスペース」

CREATE TABLESPACE ... ADD DATAFILE

セクション2.11.4「MySQL 8.0 での変更」

セクション15.6.3.6「サーバーがオフラインのときのテーブルスペースファイルの移動」

CREATE TEMPORARY TABLE

セクション13.1.20「CREATE TABLE ステートメント」

セクション13.1.20.2「CREATE TEMPORARY TABLE ステートメント」

セクション13.7.1.6「GRANT ステートメント」

セクション17.1.3.7「GTID ベースレプリケーションの制約」

セクション13.2.5「IMPORT TABLE ステートメント」

セクションA.10「MySQL 8.0 FAQ: NDB Cluster」

セクション1.3「MySQL 8.0 の新機能」

セクション6.2.2「MySQL で提供される権限」

セクション4.6.8「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」

セクションB.3.6.2「TEMPORARY テーブルに関する問題」

セクション17.1.6.5「グローバルトランザクション ID システム変数」

セクション5.1.8「サーバーシステム変数」

セクション16.1「ストレージエンジンの設定」

セクション7.5.1「バイナリログを使用したポイントインタイムリカバリ」

セクション17.5.1.31「レプリケーションと一時テーブル」

セクション15.9.1.2「圧縮テーブルの作成」

セクション13.3.3「暗黙的なコミットを発生させるステートメント」

CREATE TRIGGER

セクション13.1.2「ALTER DATABASE ステートメント」

セクション13.1.22 「CREATE TRIGGER ステートメント」
セクション17.5.1.8 「CURRENT_USER() のレプリケーション」
セクション8.2.2.3 「EXISTS 戦略を使用したサブクエリーの最適化」
セクションA.5 「MySQL 8.0 FAQ: トリガー」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション13.7.7.11 「SHOW CREATE TRIGGER ステートメント」
第25章 「ストアドオブジェクト」
セクション25.8 「ストアドプログラムの制約」
セクション25.7 「ストアドプログラムバイナリロギング」
セクション25.3.1 「トリガーの構文と例」
セクション17.5.1.16 「呼び出される機能のレプリケーション」
セクション12.16 「情報関数」
セクション13.3.3 「暗黙的なコミットを発生させるステートメント」

CREATE UNDO TABLESPACE

セクション13.1.33 「DROP TABLESPACE ステートメント」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション1.3 「MySQL 8.0 の新機能」
MySQL 用語集
セクション15.6.3.4 「undo テーブルスペース」

CREATE USER

セクション13.7.1.3 「CREATE USER ステートメント」
セクション6.8 「FIPS のサポート」
セクション13.7.8.3 「FLUSH ステートメント」
セクション13.7.1.6 「GRANT ステートメント」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション5.1.13 「IPv6 サポート」
セクション5.1.13.3 「IPv6 ローカルホストアドレスを使用した接続」
セクション6.4.1.7 「LDAP プラガブル認証」
セクション1.3 「MySQL 8.0 の新機能」
セクション6.4.7.3 「MySQL Enterprise Firewall の使用」
セクション6.2.2 「MySQL で提供される権限」
セクション8.12.3.1 「MySQL のメモリーの使用方法」
セクション5.3 「mysql システムスキーマ」
セクション5.4.4.4 「mysql データベーステーブルへの変更に対するロギング形式」
セクション4.5.6 「mysqldump — データベースバックアッププログラム」
セクション23.5.17.2 「NDB Cluster および MySQL の権限」
セクション23.6.5 「NDB Cluster のレプリケーションの準備」
セクション23.5.12 「NDB_STORED_USER での分散 MySQL 権限」
セクション6.4.1.5 「PAM プラガブル認証」
セクション6.4.1.2 「SHA-2 プラガブル認証のキャッシュ」
セクション6.4.1.3 「SHA-256 プラガブル認証」
セクション13.7.7.12 「SHOW CREATE USER ステートメント」
セクション6.4.1.6 「Windows プラガブル認証」
セクション20.5.3 「X プラグイン での暗号化接続の使用」
セクション6.2.1 「アカウントのユーザー名とパスワード」
セクション6.2.8 「アカウントの追加、権限の割当ておよびアカウントの削除」
セクション6.2.11 「アカウントカテゴリ」
セクション6.2.14 「アカウントパスワードの割り当て」
セクション6.2.20 「アカウントリソース制限の設定」
セクション6.2.19 「アカウントロック」
セクション6.2.4 「アカウント名の指定」
セクション6.2.6 「アクセス制御、ステージ 1: 接続の検証」
セクション6.2 「アクセス制御とアカウント管理」
セクション13.1.1 「アトミックデータ定義ステートメントのサポート」
セクション4.2.3 「サーバーに接続するためのコマンドオプション」
セクション5.1.8 「サーバーシステム変数」
セクション25.6 「ストアドオブジェクトのアクセス制御」

セクション6.4.1.9 「ソケットピア資格証明プラグブル認証」
セクション2.10.1 「データディレクトリの初期化」
セクション6.1.2.3 「パスワードおよびロギング」
セクション6.1.2.1 「パスワードセキュリティーのためのエンドユーザーガイドライン」
セクション6.4.3 「パスワード検証コンポーネント」
セクション6.2.15 「パスワード管理」
セクション6.2.18 「プロキシユーザー」
セクション17.3.3 「レプリケーション権限チェック」
セクション17.1.2.3 「レプリケーション用ユーザーの作成」
セクション6.4.1.8 「ログインなしのプラグブル認証」
セクション6.2.10 「ロールの使用」
セクション6.2.3 「付与テーブル」
セクション6.1.3 「攻撃者に対する MySQL のセキュアな状態の維持」
セクション6.3 「暗号化された接続の使用」
セクション6.3.1 「暗号化接続を使用するための MySQL の構成」
セクション13.3.3 「暗黙的なコミットを発生させるステートメント」
セクション15.8.2 「読み取り専用操作の InnoDB の構成」
セクション6.2.12 「部分取消しを使用した権限の制限」

**CREATE USER 'bill'@'localhost' COMMENT 'A comment' ATTRIBUTE
{'foo': 'bar', 'bazz': 'fazz'}**

セクション26.46 「INFORMATION_SCHEMA USER_ATTRIBUTES テーブル」

CREATE USER ... ATTRIBUTE ...

セクション26.46 「INFORMATION_SCHEMA USER_ATTRIBUTES テーブル」

CREATE USER ... REQUIRE SUBJECT

セクション5.1.9.4 「永続的で永続的に制限されないシステム変数」

CREATE VIEW

セクション13.1.2 「ALTER DATABASE ステートメント」
セクション13.1.11 「ALTER VIEW ステートメント」
セクション13.1.23 「CREATE VIEW ステートメント」
セクション17.5.1.8 「CURRENT_USER() のレプリケーション」
セクション26.48 「INFORMATION_SCHEMA VIEWS テーブル」
セクション13.3.6 「LOCK TABLES および UNLOCK TABLES ステートメント」
セクション6.2.2 「MySQL で提供される権限」
セクション13.7.7.13 「SHOW CREATE VIEW ステートメント」
第25章 「ストアドオブジェクト」
セクション25.9 「ビューの制約」
セクション25.5.1 「ビューの構文」
セクション25.5.2 「ビュー処理アルゴリズム」
セクション8.2.2.4 「マージまたは実体化を使用した導出テーブル、ビュー参照および共通テーブル式の最適化」
セクション8.14.3 「一般的なスレッドの状態」
セクション12.16 「情報関数」
セクション13.3.3 「暗黙的なコミットを発生させるステートメント」
セクション25.5.3 「更新可能および挿入可能なビュー」
セクション9.2.1 「識別子の長さ制限」

CREATE VIEW ... SELECT

セクション13.2.12 「TABLE ステートメント」
セクション13.2.14 「VALUES ステートメント」

D

[\[index top\]](#)

DEALLOCATE PREPARE

セクション13.5.3「DEALLOCATE PREPARE ステートメント」
セクション13.5.1「PREPARE ステートメント」
セクション27.12.6.4「prepared_statements_instances テーブル」
セクション5.1.10「サーバーステータス変数」
セクション25.8「ストアードプログラムの制約」
セクション13.5「プリペアドステートメント」

DECLARE

セクション13.1.17「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」
セクション13.6.3「DECLARE ステートメント」
セクション13.6.7.3「GET DIAGNOSTICS ステートメント」
セクション13.6.7.5「SIGNAL ステートメント」
セクション13.6.4「ストアードプログラム内の変数」

DECLARE ... CONDITION

セクション13.6.7.1「DECLARE ... CONDITION ステートメント」
セクション13.6.7.2「DECLARE ... HANDLER ステートメント」
セクション13.6.7.5「SIGNAL ステートメント」
セクション13.6.7「条件の処理」

DECLARE ... HANDLER

セクション13.6.7.1「DECLARE ... CONDITION ステートメント」
セクション13.6.7.2「DECLARE ... HANDLER ステートメント」
セクション13.6.7.5「SIGNAL ステートメント」
セクション13.6.7「条件の処理」

DEFAULT ENCRYPTION

セクション13.1.10「ALTER TABLESPACE ステートメント」

DEFAULT ENCRYPTION='N'

セクション13.1.10「ALTER TABLESPACE ステートメント」

DEFAULT ENCRYPTION='Y'

セクション13.1.10「ALTER TABLESPACE ステートメント」

DELETE

セクション13.1.2「ALTER DATABASE ステートメント」
セクション13.1.9.1「ALTER TABLE パーティション操作」
セクション16.5「ARCHIVE ストレージエンジン」
セクション16.6「BLACKHOLE ストレージエンジン」
セクション13.1.17「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」
セクション13.1.22「CREATE TRIGGER ステートメント」
セクション13.1.23「CREATE VIEW ステートメント」
セクション13.2.2「DELETE ステートメント」
セクション8.8.1「EXPLAIN によるクエリーの最適化」
セクション13.8.2「EXPLAIN ステートメント」
セクション8.8.2「EXPLAIN 出力フォーマット」
セクション16.8.3「FEDERATED ストレージエンジンの注記とヒント」
セクション13.1.20.5「FOREIGN KEY の制約」
セクション13.7.1.6「GRANT ステートメント」
セクション26.51.27「INFORMATION_SCHEMA INNODB_TABLESTATS ビュー」
セクション26.38「INFORMATION_SCHEMA TABLES テーブル」
セクション26.48「INFORMATION_SCHEMA VIEWS テーブル」
セクション15.19「InnoDB と MySQL レプリケーション」
セクション15.7.3「InnoDB のさまざまな SQL ステートメントで設定されたロック」

セクション15.21.2 「InnoDB のリカバリの強制的な実行」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.1.2 「InnoDB テーブルのベストプラクティス」
セクション13.2.10.2 「JOIN 句」
セクション13.7.8.4 「KILL ステートメント」
セクション24.2.2 「LIST パーティショニング」
セクション15.20.6.5 「memcached 操作に合わせた DML ステートメントの改変」
セクション16.3 「MEMORY ストレージエンジン」
セクション16.7 「MERGE ストレージエンジン」
セクション16.7.2 「MERGE テーブルの問題点」
セクション15.6.1.5 「MyISAM から InnoDB へのテーブルの変換」
セクション8.6.2 「MyISAM テーブルの一括データロード」
セクション1.3 「MySQL 8.0 の新機能」
セクション6.2.2 「MySQL で提供される権限」
セクション24.1 「MySQL のパーティショニングの概要」
セクション1.2.2 「MySQL の主な機能」
セクションB.3.7 「MySQL の既知の問題」
セクション6.2.21 「MySQL への接続の問題のトラブルシューティング」
セクション4.5.1.6 「mysql クライアントのヒント」
セクション4.5.1.1 「mysql クライアントオプション」
セクション5.4.4.4 「mysql データベーステーブルへの変更に対するロギング形式」
MySQL 用語集
セクション23.5.17.3 「NDB Cluster および MySQL セキュリティー手順」
セクション23.1.7.3 「NDB Cluster でのトランザクション処理に関する制限」
セクション23.1.7.2 「NDB Cluster と標準の MySQL 制限の制限と相違点」
セクション23.5.5 「NDB Cluster のローリング再起動の実行」
セクション23.1.4 「NDB Cluster の新機能」
セクション23.5.10.1 「NDB Cluster ディスクデータオブジェクト」
セクション23.4.8 「ndb_delete_all — NDB テーブルからのすべての行の削除」
セクション15.9.1.6 「OLTP ワークロードの圧縮」
セクション24.3.1 「RANGE および LIST パーティションの管理」
セクション8.2.1.2 「range の最適化」
セクション24.2.1 「RANGE パーティショニング」
セクション17.1.6.3 「Replica Server のオプションと変数」
セクション8.2.1 「SELECT ステートメントの最適化」
セクション13.7.7.38 「SHOW TABLE STATUS ステートメント」
セクション8.3.3 「SPATIAL インデックス最適化」
セクション13.1.37 「TRUNCATE TABLE ステートメント」
セクション15.6.6 「undo ログ」
セクション8.2.1.1 「WHERE 句の最適化」
セクション13.2.15 「WITH (共通テーブル式)」
セクション3.3.4.1 「すべてのデータの選択」
セクション26.1 「はじめに」
セクション6.2.8 「アカウントの追加、権限の割当ておよびアカウントの削除」
セクション6.2 「アクセス制御とアカウント管理」
セクション12.21.5 「ウィンドウ機能の制限事項」
セクション8.9.3 「オプティマイザヒント」
セクション15.12.1 「オンライン DDL 操作」
セクション9.3 「キーワードと予約語」
セクション13.2.11 「サブクエリー」
セクション8.2.2 「サブクエリー、導出テーブル、ビュー参照および共通テーブル式の最適化」
セクション5.1.11 「サーバー SQL モード」
セクション5.1.8 「サーバーシステム変数」
セクション5.1.10 「サーバーステータス変数」
セクション17.2.1.1 「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」
セクション8.11.2 「テーブルロックの問題」
セクション8.2.5 「データ変更ステートメントの最適化」
セクション15.7.2.1 「トランザクション分離レベル」
セクション25.3.1 「トリガーの構文と例」
セクション17.1.6.4 「バイナリロギングのオプションと変数」

セクション5.4.4 「バイナリログ」
セクション15.8.9 「パーティション構成」
セクション24.4 「パーティションプルーニング」
セクション24.5 「パーティション選択」
セクション5.6.4 「リライタクエリーリライトプラグイン」
セクション5.6.4.2 「リライタクエリーリライトプラグインの使用」
セクション17.5.1.27 「レプリケーションおよび行検索」
セクション17.5.1.18 「レプリケーションと LIMIT」
セクション17.5.1.21 「レプリケーションと MEMORY テーブル」
セクション17.5.1.23 「レプリケーションとクエリー最適化」
セクション17.5.1.36 「レプリケーションとトリガー」
セクション5.4.1 「一般クエリーログおよびスロークエリーログの出力先の選択」
セクション8.14.3 「一般的なスレッドの状態」
セクション15.7.2.3 「一貫性非ロック読み取り」
セクション6.2.3 「付与テーブル」
セクション12.10.5 「全文制限」
セクション8.11.1 「内部ロック方法」
セクション8.8.4 「名前付き接続の実行計画情報の取得」
セクション15.5.2 「変更バッファ」
セクション8.2.2.2 「実体化を使用したサブクエリーの最適化」
セクション12.16 「情報関数」
セクション8.8.3 「拡張 EXPLAIN 出力形式」
セクション25.5.3 「更新可能および挿入可能なビュー」
セクション1.7.1 「標準 SQL に対する MySQL 拡張機能」
セクション6.2.13 「権限変更が有効化される時期」
セクション8.2.2.1 「準結合変換による IN および EXISTS サブクエリー述語の最適化」
セクション6.4.5.7 「監査ログのフィルタリング」
セクション6.4.5.8 「監査ログフィルタ定義の書込み」
セクション6.4.5.10 「監査ログ参照」
セクション17.2.1.2 「行ベースロギングおよびレプリケーションの使用」
第12章 「関数と演算子」
セクションB.3.4.6 「関連するテーブルからの行の削除」

DELETE FROM ... WHERE ...

セクション15.7.3 「InnoDB のさまざまな SQL ステートメントで設定されたロック」

DESCRIBE

セクション13.8.1 「DESCRIBE ステートメント」
セクション13.8.2 「EXPLAIN ステートメント」
セクション8.4.4 「MySQL での内部一時テーブルの使用」
セクション13.7.7.5 「SHOW COLUMNS ステートメント」
セクション26.55 「SHOW ステートメントの拡張」
セクション3.3.2 「テーブルの作成」
セクション3.4 「データベースとテーブルに関する情報の取得」
セクション10.2.2 「メタデータ用の UTF-8」
セクション3.6.6 「外部キーの使用」
セクション13.1.20.7 「暗黙のカラム指定の変更」

DISCARD PARTITION

セクション15.12.1 「オンライン DDL 操作」

DISCARD PARTITION ... TABLESPACE

セクション13.1.9.1 「ALTER TABLE パーティション操作」

DO

セクション13.1.3 「ALTER EVENT ステートメント」
セクション13.1.13 「CREATE EVENT ステートメント」
セクション13.2.3 「DO ステートメント」

セクション26.14 「INFORMATION_SCHEMA EVENTS テーブル」
セクション5.4.4.4 「mysql データベーステーブルへの変更に対するロギング形式」
セクション13.2.11 「サブクエリー」
セクション25.8 「ストアードプログラムの制約」
セクション25.7 「ストアードプログラムバイナリロギング」
セクション18.4.2.2 「トランザクション一貫性保証の構成」
セクション12.15 「ロック関数」
セクション1.7.1 「標準 SQL に対する MySQL 拡張機能」

DROP DATABASE

セクション13.1.2 「ALTER DATABASE ステートメント」
セクション13.7.3.1 「ANALYZE TABLE ステートメント」
セクション13.1.24 「DROP DATABASE ステートメント」
セクション13.1.33 「DROP TABLESPACE ステートメント」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション7.4.1 「mysqldump による SQL フォーマットでのデータのダンプ」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション23.1.4 「NDB Cluster の新機能」
セクション23.1.7.8 「NDB Cluster 専用の問題」
セクション23.5.12 「NDB_STORED_USER での分散 MySQL 権限」
セクション17.1.6.3 「Replica Server のオプションと変数」
セクション2.3.7 「Windows プラットフォームの制限事項」
セクション2.11.5 「アップグレード用のインストールの準備」
セクション13.1.1 「アトミックデータ定義ステートメントのサポート」
セクション17.2.5 「サーバーがレプリケーションフィルタリングルールをどのように評価するか」
セクション17.2.5.1 「データベースレベルレプリケーションオプションおよびバイナリロギングオプションの評価」
セクション15.6.3.3 「一般テーブルスペース」
セクション13.3.3 「暗黙的なコミットを発生させるステートメント」
セクション1.7.1 「標準 SQL に対する MySQL 拡張機能」

DROP DATABASE IF EXISTS

セクション17.5.1.11 「DROP ... IF EXISTS ステートメントのレプリケーション」

DROP EVENT

セクション13.1.2 「ALTER DATABASE ステートメント」
セクション25.4.6 「イベントスケジューラと MySQL 権限」
セクション25.4.3 「イベント構文」
セクション25.7 「ストアードプログラムバイナリロギング」
セクション17.5.1.16 「呼び出される機能のレプリケーション」
セクション13.3.3 「暗黙的なコミットを発生させるステートメント」

DROP FUNCTION

セクション13.1.2 「ALTER DATABASE ステートメント」
セクション13.1.4 「ALTER FUNCTION ステートメント」
セクション13.1.26 「DROP FUNCTION ステートメント」
セクション13.1.29 「DROP PROCEDURE および DROP FUNCTION ステートメント」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション6.5.2 「MySQL Enterprise Data Masking and De-identification のインストールまたはアンインストール」
セクション6.6.1 「MySQL Enterprise Encryption のインストール」
セクション1.8.1 「MySQL への貢献者」
セクション2.11.12 「アップグレードのトラブルシューティング」
セクション25.7 「ストアードプログラムバイナリロギング」
セクション25.2.1 「ストアードルーチンの構文」
セクション5.6.6.2 「バージョントークンのインストールまたはアンインストール」
セクション13.7.4.2 「ユーザー定義関数に対する DROP FUNCTION ステートメント」
セクション5.7.1 「ユーザー定義関数のインストールおよびアンインストール」
セクション13.7.4.1 「ユーザー定義関数用の CREATE FUNCTION ステートメント」

[ロックサービス UDF インタフェース](#)
[セクション17.5.1.16 「呼び出される機能のレプリケーション」](#)
[セクション13.3.3 「暗黙的なコミットを発生させるステートメント」](#)
[セクション6.4.4.10 「汎用キーリングキー管理関数」](#)
[セクション9.2.5 「関数名の構文解析と解決」](#)

DROP INDEX

[セクション13.1.2 「ALTER DATABASE ステートメント」](#)
[セクション13.1.9 「ALTER TABLE ステートメント」](#)
[セクション13.1.27 「DROP INDEX ステートメント」](#)
[MySQL 用語集](#)
[セクション23.5.11 「NDB Cluster での ALTER TABLE を使用したオンライン操作」](#)
[セクション15.12.1 「オンライン DDL 操作」](#)
[セクション5.1.8 「サーバーシステム変数」](#)
[セクション5.4.5 「スロークエリーログ」](#)
[セクション13.3.3 「暗黙的なコミットを発生させるステートメント」](#)
[セクション1.7.1 「標準 SQL に対する MySQL 拡張機能」](#)
[セクション11.4.10 「空間インデックスの作成」](#)

DROP LOGFILE GROUP

[セクション13.1.28 「DROP LOGFILE GROUP ステートメント」](#)
[セクション23.1.7.8 「NDB Cluster 専用の問題」](#)

DROP PARTITION

[セクション15.12.1 「オンライン DDL 操作」](#)

DROP PREPARE

[セクション27.12.6.4 「prepared_statements_instances テーブル」](#)

DROP PROCEDURE

[セクション13.1.2 「ALTER DATABASE ステートメント」](#)
[セクション13.1.7 「ALTER PROCEDURE ステートメント」](#)
[セクション25.7 「ストアードプログラムバイナリロギング」](#)
[セクション25.2.1 「ストアードルーチンの構文」](#)
[セクション17.5.1.16 「呼び出される機能のレプリケーション」](#)
[セクション13.3.3 「暗黙的なコミットを発生させるステートメント」](#)

DROP RESOURCE GROUP

[セクション13.7.2.3 「DROP RESOURCE GROUP ステートメント」](#)
[セクション5.1.16 「リソースグループ」](#)

DROP ROLE

[セクション13.7.1.4 「DROP ROLE ステートメント」](#)
[セクション6.2.2 「MySQL で提供される権限」](#)
[セクション5.1.8 「サーバーシステム変数」](#)
[セクション6.2.10 「ロールの使用」](#)
[セクション13.3.3 「暗黙的なコミットを発生させるステートメント」](#)

DROP SCHEMA

[セクション13.1.24 「DROP DATABASE ステートメント」](#)
[セクション23.1.7.8 「NDB Cluster 専用の問題」](#)
[セクション5.1.8 「サーバーシステム変数」](#)

DROP SERVER

[セクション17.5.1.5 「CREATE SERVER、ALTER SERVER、および DROP SERVER のレプリケーション」](#)
[セクション13.7.8.3 「FLUSH ステートメント」](#)

セクション6.2.2「MySQL で提供される権限」
セクション8.12.3.1「MySQL のメモリーの使用方法」
セクション13.1.1「アトミックデータ定義ステートメントのサポート」
セクション13.3.3「暗黙的なコミットを発生させるステートメント」

DROP SPATIAL REFERENCE SYSTEM

セクション13.1.31「DROP SPATIAL REFERENCE SYSTEM ステートメント」
セクション26.36「INFORMATION_SCHEMA ST_SPATIAL_REFERENCE_SYSTEMS テーブル」
セクション13.3.3「暗黙的なコミットを発生させるステートメント」
セクション11.4.5「空間参照システムのサポート」

DROP TABLE

セクション13.1.2「ALTER DATABASE ステートメント」
セクション13.1.9「ALTER TABLE ステートメント」
セクション13.7.3.1「ANALYZE TABLE ステートメント」
セクション13.1.21「CREATE TABLESPACE ステートメント」
セクション13.1.20.2「CREATE TEMPORARY TABLE ステートメント」
セクション13.1.22「CREATE TRIGGER ステートメント」
セクション13.1.32「DROP TABLE ステートメント」
セクション16.8.3「FEDERATED ストレージエンジンの注記とヒント」
セクション15.6.3.2「File-Per-Table テーブルスペース」
セクション17.1.3.2「GTID ライフサイクル」
セクション8.5.7「InnoDB DDL 操作の最適化」
セクション15.21.2「InnoDB のリカバリの強制的な実行」
セクション15.21.3「InnoDB データディクショナリの操作のトラブルシューティング」
セクション13.3.6「LOCK TABLES および UNLOCK TABLES ステートメント」
セクション16.3「MEMORY ストレージエンジン」
セクション16.7「MERGE ストレージエンジン」
セクション16.7.2「MERGE テーブルの問題点」
セクション6.2.2「MySQL で提供される権限」
セクション13.6.7.7「MySQL の診断領域」
セクション4.5.1.1「mysql クライアントオプション」
MySQL 用語集
セクション4.5.4「mysqldump — データベースバックアッププログラム」
セクション4.5.6「mysqlpump — データベースバックアッププログラム」
セクション23.1.7.2「NDB Cluster と標準の MySQL 制限の制限と相違点」
セクション23.5.7.1「NDB Cluster データノードのオンラインでの追加: 一般的な問題」
セクション23.6.9.2「NDB Cluster レプリケーションを使用したポイントインタイムリカバリ」
セクション23.1.7.8「NDB Cluster 専用の問題」
セクション23.5.1「NDB Cluster 管理クライアントのコマンド」
セクション23.4.10「ndb_drop_index — NDB テーブルからのインデックスの削除」
セクション23.4.11「ndb_drop_table — NDB テーブルの削除」
セクション13.7.7.39「SHOW TABLES ステートメント」
セクション13.6.7.5「SIGNAL ステートメント」
セクション13.4.2.7「START REPLICA | SLAVE ステートメント」
セクション13.1.37「TRUNCATE TABLE ステートメント」
セクション13.7.4.6「UNINSTALL PLUGIN ステートメント」
セクション13.1.1「アトミックデータ定義ステートメントのサポート」
セクション17.1.6.5「グローバルトランザクション ID システム変数」
セクション5.1.8「サーバーシステム変数」
セクション13.6.7.6「ハンドラのスコープに関するルール」
セクション7.5.1「バイナリログを使用したポイントインタイムリカバリ」
セクション5.4.4.2「バイナリログ形式の設定」
セクション25.9「ビューの制約」
セクション14.2「ファイルベースのメタデータ記憶域の削除」
セクション5.4.1「一般クエリーログおよびスロークエリーログの出力先の選択」
セクション15.6.3.3「一般テーブルスペース」
セクション15.7.2.3「一貫性非ロック読み取り」
セクション12.16「情報関数」

セクション13.3.3「暗黙的なコミットを発生させるステートメント」
セクション1.7.1「標準 SQL に対する MySQL 拡張機能」
セクション6.4.5.4「監査ログファイル形式」

DROP TABLE IF EXISTS

セクション17.5.1.11「DROP ... IF EXISTS ステートメントのレプリケーション」

DROP TABLESPACE

セクション1.3「MySQL 8.0 の新機能」
セクション23.1.7.8「NDB Cluster 専用の問題」
セクション13.1.1「アトミックデータ定義ステートメントのサポート」
セクション5.1.8「サーバーシステム変数」
セクション15.6.3.3「一般テーブルスペース」

DROP TABLESPACE tablespace_name

セクション15.6.3.3「一般テーブルスペース」

DROP TEMPORARY TABLE

セクション17.1.3.7「GTID ベースレプリケーションの制約」
セクション17.1.6.5「グローバルトランザクション ID システム変数」
セクション17.5.1.31「レプリケーションと一時テーブル」

DROP TRIGGER

セクション13.1.2「ALTER DATABASE ステートメント」
セクション13.1.34「DROP TRIGGER ステートメント」
セクションA.5「MySQL 8.0 FAQ: トリガー」
セクション4.5.4「mysqldump — データベースバックアッププログラム」
セクション25.3.1「トリガーの構文と例」
セクション17.5.1.16「呼び出される機能のレプリケーション」
セクション13.3.3「暗黙的なコミットを発生させるステートメント」

DROP UNDO TABALESPACE

セクション15.6.3.4「undo テーブルスペース」

DROP UNDO TABLESPACE

セクション2.11.4「MySQL 8.0 での変更」
セクション1.3「MySQL 8.0 の新機能」

DROP USER

セクション17.5.1.8「CURRENT_USER() のレプリケーション」
セクション13.7.1.5「DROP USER ステートメント」
セクション13.7.8.3「FLUSH ステートメント」
セクション13.7.1.6「GRANT ステートメント」
セクション1.3「MySQL 8.0 の新機能」
セクション6.2.2「MySQL で提供される権限」
セクション8.12.3.1「MySQL のメモリーの使用法」
セクション4.5.6「mysqlpump — データベースバックアッププログラム」
セクション23.5.17.2「NDB Cluster および MySQL の権限」
セクション23.5.12「NDB_STORED_USER での分散 MySQL 権限」
セクション13.7.1.8「REVOKE ステートメント」
セクション6.2.1「アカウントのユーザー名とパスワード」
セクション6.2.8「アカウントの追加、権限の割当ておよびアカウントの削除」
セクション25.4.6「イベントスケジューラと MySQL 権限」
セクション5.1.8「サーバーシステム変数」
セクション25.6「ストアドオブジェクトのアクセス制御」
セクション6.2.10「ロールの使用」
セクション12.16「情報関数」

セクション13.3.3「暗黙的なコミットを発生させるステートメント」

DROP VIEW

セクション13.1.2「ALTER DATABASE ステートメント」

セクション13.1.35「DROP VIEW ステートメント」

セクション13.3.6「LOCK TABLES および UNLOCK TABLES ステートメント」

セクション13.1.1「アトミックデータ定義ステートメントのサポート」

セクション25.9「ビューの制約」

セクション25.5.1「ビューの構文」

セクション13.3.3「暗黙的なコミットを発生させるステートメント」

DROP VIEW IF EXISTS

セクション17.5.1.11「DROP ... IF EXISTS ステートメントのレプリケーション」

E

[[index top](#)]

ENCRYPTION

セクション15.13「InnoDB 保存データ暗号化」

EXCHANGE PARTITION

セクション15.12.1「オンライン DDL 操作」

EXECUTE

セクション13.2.1「CALL ステートメント」

セクション13.5.2「EXECUTE ステートメント」

セクション13.5.1「PREPARE ステートメント」

セクション27.12.6.4「prepared_statements_instances テーブル」

セクション5.1.10「サーバーステータス変数」

セクション25.8「ストアードプログラムの制約」

セクション13.5「プリペアドステートメント」

EXPLAIN

セクション13.1.9「ALTER TABLE ステートメント」

セクション8.2.1.12「Block Nested Loop 結合と Batched Key Access 結合」

セクション13.1.15「CREATE INDEX ステートメント」

セクション13.1.17「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」

セクション13.8.1「DESCRIBE ステートメント」

セクション8.2.1.18「DISTINCT の最適化」

セクション8.2.2.3「EXISTS 戦略を使用したサブクエリーの最適化」

セクション8.8.1「EXPLAIN によるクエリーの最適化」

セクション13.8.2「EXPLAIN ステートメント」

セクション8.8.2「EXPLAIN 出力フォーマット」

セクション8.2.1.17「GROUP BY の最適化」

セクション8.2.3「INFORMATION_SCHEMA クエリーの最適化」

InnoDB オプティマイザ統計でサンプリングされるページの数の構成

セクション8.2.1.15「IS NULL の最適化」

セクション12.18.3「JSON 値を検索する関数」

セクション8.2.1.19「LIMIT クエリーの最適化」

セクション8.2.1.11「Multi-Range Read の最適化」

セクション1.3「MySQL 8.0 の新機能」

セクション8.4.4「MySQL での内部一時テーブルの使用」

セクション6.2.2「MySQL で提供される権限」

セクション1.2.2「MySQL の主な機能」

セクション4.5.1.6「mysql クライアントのヒント」

セクション5.9.1「MySQL サーバーのデバッグ」

第27章「MySQL パフォーマンススキーマ」

セクション5.9.1.6 「mysqld でのエラーの原因を見つけるためのサーバーログの使用」
セクション23.1.4 「NDB Cluster の新機能」
NDB Cluster システム変数
NDB Cluster ステータス変数
セクション8.2.1.16 「ORDER BY の最適化」
セクション28.4.4.22 「ps_trace_statement_digest() プロシージャ」
セクション8.2.1.2 「range の最適化」
セクション13.2.10 「SELECT ステートメント」
セクション8.2.1 「SELECT ステートメントの最適化」
セクション13.7.7.42 「SHOW WARNINGS ステートメント」
セクション13.2.15 「WITH (共通テーブル式)」
セクション12.24 「その他の関数」
セクション26.1 「はじめに」
セクション8.3.7 「インデックスの使用の確認」
セクション8.2.1.6 「インデックスコンディションプッシュダウンの最適化」
セクション8.9.4 「インデックスヒント」
セクション8.2.1.3 「インデックスマージの最適化」
セクション8.3.10 「インデックス拡張の使用」
セクション8.2.1.21 「ウィンドウ機能最適化」
セクション8.2.1.5 「エンジンコンディションプッシュダウンの最適化」
セクション8.9.3 「オプティマイザヒント」
セクション8.9.6 「オプティマイザ統計」
セクションB.3.5 「オプティマイザ関連の問題」
セクション8.3.5 「カラムインデックス」
セクション8.8 「クエリー実行プランの理解」
セクション27.12.18.3 「ステートメントサマリーテーブル」
セクション25.8 「ストアプログラムの制約」
セクション13.1.20.9 「セカンダリインデックスと生成されたカラム」
セクション8.2.1.4 「ハッシュ結合の最適化」
セクション8.2.4 「パフォーマンススキーマクエリーの最適化」
セクション24.3.5 「パーティションに関する情報を取得する」
セクション24.4 「パーティションプルーニング」
セクション8.2.2.4 「マージまたは実体化を使用した導出テーブル、ビュー参照および共通テーブル式の最適化」
セクションB.3.4.7 「一致する行がない場合の問題の解決」
セクション8.3.12 「不可視のインデックス」
セクション8.2.1.23 「全テーブルスキャンの回避」
セクション8.9.2 「切り替え可能な最適化」
セクション8.8.4 「名前付き接続の実行計画情報の取得」
セクション8.2.2.2 「実体化を使用したサブクエリーの最適化」
セクション13.2.11.8 「導出テーブル」
セクション8.8.3 「拡張 EXPLAIN 出力形式」
セクション8.2.1.13 「条件フィルタ」
セクション8.2.2.1 「準結合変換による IN および EXISTS サブクエリー述語の最適化」
セクション8.3.11 「生成されたカラムインデックスのオプティマイザによる使用」
セクション11.4.11 「空間インデックスの使用」

EXPLAIN ... FOR CONNECTION

セクション13.8.2 「EXPLAIN ステートメント」

EXPLAIN ANALYZE

セクション13.7.8.4 「KILL ステートメント」
セクション1.3 「MySQL 8.0 の新機能」
セクション8.2.1.4 「ハッシュ結合の最適化」

EXPLAIN FOR CONNECTION

セクション8.8.2 「EXPLAIN 出力フォーマット」
セクション8.2.1.2 「range の最適化」
セクション5.1.10 「サーバステータス変数」
セクション8.8.4 「名前付き接続の実行計画情報の取得」

EXPLAIN FORMAT=JSON

セクション8.2.1.21「ウィンドウ機能最適化」

EXPLAIN FORMAT=TREE

セクション1.3「MySQL 8.0 の新機能」

EXPLAIN SELECT

セクション8.8.2「EXPLAIN 出力フォーマット」
セクション15.7.5.3「デッドロックを最小化および処理する方法」
セクション24.3.5「パーティションに関する情報を取得する」
セクション13.2.11.8「導出テーブル」
セクション1.7.1「標準 SQL に対する MySQL 拡張機能」
セクション1.6「質問またはバグをレポートする方法」

EXPLAIN SELECT COUNT()

セクション24.2.1「RANGE パーティショニング」

EXPLAIN tbl_name

セクション8.8.1「EXPLAIN によるクエリーの最適化」

F

[\[index top\]](#)

FETCH

セクション13.6.6.2「カーソル DECLARE ステートメント」
セクション13.6.6.3「カーソル FETCH ステートメント」
セクション25.8「ストアプログラムの制約」

FETCH ... INTO var_list

セクション13.6.4「ストアプログラム内の変数」

FLUSH

セクション13.7.8.3「FLUSH ステートメント」
セクション13.7.1.6「GRANT ステートメント」
セクション1.3「MySQL 8.0 の新機能」
セクション4.10「MySQL での Unix シグナル処理」
セクション6.2.2「MySQL で提供される権限」
セクション4.5.4「mysqldump — データベースバックアッププログラム」
セクション13.7.8.6「RESET ステートメント」
セクション25.8「ストアプログラムの制約」
セクション7.3.1「バックアップポリシーの確立」
セクション17.5.1.13「レプリケーションと FLUSH」
セクション13.3.3「暗黙的なコミットを発生させるステートメント」
セクション1.7.1「標準 SQL に対する MySQL 拡張機能」

FLUSH BINARY LOGS

セクション13.7.8.3「FLUSH ステートメント」
セクション13.4.1.2「RESET MASTER ステートメント」
セクション5.4.6「サーバーログの保守」

FLUSH ENGINE LOGS

セクション13.7.8.3「FLUSH ステートメント」

FLUSH ERROR LOGS

セクション13.7.8.3「FLUSH ステートメント」

[セクション5.4.2.10「エラーログファイルのフラッシュおよび名前変更」](#)

FLUSH GENERAL LOGS

[セクション13.7.8.3「FLUSH ステートメント」](#)

FLUSH HOSTS

[セクション5.1.12.3「DNS ルックアップとホストキャッシュ」](#)

[セクション13.7.8.3「FLUSH ステートメント」](#)

[セクション1.3「MySQL 8.0 の新機能」](#)

FLUSH LOGS

[セクション13.7.8.3「FLUSH ステートメント」](#)

[セクション17.1.3.4「GTID を使用したレプリケーションのセットアップ」](#)

[セクション17.1.4.2「GTID トランザクションのオンラインでの有効化」](#)

[セクション17.1.4.3「GTID トランザクションのオンラインでの無効化」](#)

[セクション5.4「MySQL Server ログ」](#)

[セクション4.5.2「mysqldadmin — A MySQL Server 管理プログラム」](#)

[セクション23.1.4「NDB Cluster の新機能」](#)

[セクション5.4.2.10「エラーログファイルのフラッシュおよび名前変更」](#)

[セクション5.1.10「サーバーステータス変数」](#)

[セクション5.4.6「サーバーログの保守」](#)

[セクション7.2「データベースバックアップ方法」](#)

[セクション7.3.1「バックアップポリシーの確立」](#)

[セクション7.3.3「バックアップ戦略サマリー」](#)

[セクション17.1.3.5「フェイルオーバーおよびスケールアウトでの GTID の使用」](#)

[セクション17.2.4.1「リレーログ」](#)

[セクション17.5.1.13「レプリケーションと FLUSH」](#)

[セクション5.4.1「一般クエリーログおよびスロークエリーログの出力先の選択」](#)

FLUSH OPTIMIZER_COSTS

[セクション13.7.8.3「FLUSH ステートメント」](#)

[セクション6.2.2「MySQL で提供される権限」](#)

[セクション8.9.5「オプティマイザコストモデル」](#)

FLUSH PRIVILEGES

[セクション13.7.8.3「FLUSH ステートメント」](#)

[セクション6.2.2「MySQL で提供される権限」](#)

[セクション8.12.3.1「MySQL のメモリーの使用方法」](#)

[セクション6.2.21「MySQL への接続の問題のトラブルシューティング」](#)

[セクション4.5.4「mysqldump — データベースバックアッププログラム」](#)

[セクション23.5.17.3「NDB Cluster および MySQL セキュリティー手順」](#)

[セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#)

[セクション1.1「このマニュアルについて」](#)

[セクション6.2.20「アカウントリソース制限の設定」](#)

[セクション5.1.7「サーバーコマンドオプション」](#)

[セクション6.2.15「パスワード管理」](#)

[セクション17.5.1.13「レプリケーションと FLUSH」](#)

[セクション6.2.10「ロールの使用」](#)

[セクション6.2.3「付与テーブル」](#)

[セクション6.2.13「権限変更が有効化される時期」](#)

FLUSH RELAY LOGS

[セクション13.7.8.3「FLUSH ステートメント」](#)

[セクション17.2.2.1「単一チャンネルで操作するためのコマンド」](#)

FLUSH RELAY LOGS FOR CHANNEL channel

[セクション13.7.8.3「FLUSH ステートメント」](#)

FLUSH SLOW LOGS

[セクション13.7.8.3「FLUSH ステートメント」](#)

FLUSH STATUS

[セクション13.7.8.3「FLUSH ステートメント」](#)

[セクション6.2.2「MySQL で提供される権限」](#)

[セクション8.3.10「インデックス拡張の使用」](#)

[セクション5.1.10「サーバーステータス変数」](#)

[セクション27.12.18.12「ステータス変数サマリーテーブル」](#)

[セクション27.12.15「パフォーマンススキーマのステータス変数のテーブル」](#)

FLUSH TABLE

[セクション13.7.8.3「FLUSH ステートメント」](#)

FLUSH TABLES

[セクション13.7.8.3「FLUSH ステートメント」](#)

[セクション13.2.4「HANDLER ステートメント」](#)

[セクション16.7.2「MERGE テーブルの問題点」](#)

[セクション8.6.2「MyISAM テーブルの一括データロード」](#)

[セクション4.6.4「myisamchk — MyISAM テーブルメンテナンユーティリティー」](#)

[セクション8.4.3.1「MySQL でのテーブルのオープンとクローズの方法」](#)

[セクション6.2.2「MySQL で提供される権限」](#)

[セクション8.12.3.1「MySQL のメモリーの使用方法」](#)

[セクション8.3.10「インデックス拡張の使用」](#)

[セクション5.1.8「サーバーシステム変数」](#)

[セクション5.1.10「サーバーステータス変数」](#)

[セクション8.10.3「プリペアドステートメントおよびストアードプログラムのキャッシュ」](#)

[セクション17.5.1.13「レプリケーションと FLUSH」](#)

[セクション17.1.2.4「レプリケーションソースのバイナリログ座標の取得」](#)

[セクション5.4.1「一般クエリログおよびスロークエリログの出力先の選択」](#)

[セクション8.14.3「一般的なスレッドの状態」](#)

[セクション16.2.4.2「適切に閉じられなかったテーブルの問題」](#)

FLUSH TABLES ... FOR EXPORT

[セクション13.7.8.3「FLUSH ステートメント」](#)

[セクション15.6.1.3「InnoDB テーブルのインポート」](#)

[MySQL 用語集](#)

[セクション15.6.1.2「外部でのテーブルの作成」](#)

FLUSH TABLES ...FOR EXPORT

[セクション13.7.8.3「FLUSH ステートメント」](#)

FLUSH TABLES tbl_name ...

[セクション13.7.8.3「FLUSH ステートメント」](#)

FLUSH TABLES tbl_name ... FOR EXPORT

[セクション13.7.8.3「FLUSH ステートメント」](#)

FLUSH TABLES tbl_name ... WITH READ LOCK

[セクション13.7.8.3「FLUSH ステートメント」](#)

[セクション13.3.5「LOCK INSTANCE FOR BACKUP および UNLOCK INSTANCE ステートメント」](#)

FLUSH TABLES tbl_name WITH READ LOCK

[セクション13.2.4「HANDLER ステートメント」](#)

FLUSH TABLES WITH READ LOCK

- セクション13.1.10 「ALTER TABLESPACE ステートメント」
- セクション13.7.8.3 「FLUSH ステートメント」
- セクション13.3.6 「LOCK TABLES および UNLOCK TABLES ステートメント」
- セクション27.12.13.3 「metadata_locks テーブル」
- セクション4.5.4 「mysqldump — データベースバックアッププログラム」
- セクション13.3.1 「START TRANSACTION、COMMIT および ROLLBACK ステートメント」
- セクション5.1.8 「サーバーシステム変数」
- セクション7.2 「データベースバックアップ方法」
- セクション7.3.1 「バックアップポリシーの確立」
- セクション17.5.1.13 「レプリケーションと FLUSH」
- セクション17.1.2.4 「レプリケーションソースのバイナリログ座標の取得」
- セクション5.4.1 「一般クエリーログおよびスロークエリーログの出力先の選択」
- セクション15.6.3.3 「一般テーブルスペース」
- セクション8.14.3 「一般的なスレッドの状態」
- セクション13.3.3 「暗黙的なコミットを発生させるステートメント」

FLUSH USER_RESOURCES

- セクション13.7.8.3 「FLUSH ステートメント」
- セクション6.2.2 「MySQL で提供される権限」
- セクション6.2.20 「アカウントリソース制限の設定」

G

[[index top](#)]

GET DIAGNOSTICS

- セクション13.6.7.3 「GET DIAGNOSTICS ステートメント」
- セクション13.6.7.7 「MySQL の診断領域」
- セクション13.6.7.4 「RESIGNAL ステートメント」
- セクション13.7.7.42 「SHOW WARNINGS ステートメント」
- セクション13.6.7.5 「SIGNAL ステートメント」
- セクションB.2 「エラー情報インタフェース」
- セクション5.1.8 「サーバーシステム変数」
- セクション25.8 「ストアプログラムの制約」
- セクション13.6.7 「条件の処理」
- セクション13.6.8 「条件処理の制約」

GET STACKED DIAGNOSTICS

- セクション13.6.7.3 「GET DIAGNOSTICS ステートメント」
- セクション13.6.7.7 「MySQL の診断領域」

GRANT

- セクション13.7.1.3 「CREATE USER ステートメント」
- セクション17.5.1.8 「CURRENT_USER() のレプリケーション」
- セクション13.7.8.3 「FLUSH ステートメント」
- セクション13.7.1.6 「GRANT ステートメント」
- セクション15.14 「InnoDB の起動オプションおよびシステム変数」
- セクション5.1.13 「IPv6 サポート」
- セクション5.1.13.3 「IPv6 ローカルホストアドレスを使用した接続」
- セクションA.14 「MySQL 8.0 FAQ: レプリケーション」
- セクション2.11.4 「MySQL 8.0 での変更」
- セクション1.3 「MySQL 8.0 の新機能」
- セクション6.4.7.3 「MySQL Enterprise Firewall の使用」
- セクション6.2.2 「MySQL で提供される権限」
- セクション8.12.3.1 「MySQL のメモリーの使用方法」
- セクション5.3 「mysql システムスキーマ」

セクション17.5.1.22 「mysql システムスキーマのレプリケーション」
セクション5.4.4.4 「mysql データベーステーブルへの変更に対するロギング形式」
MySQL 用語集
セクション4.5.6 「mysqldump — データベースバックアッププログラム」
セクション23.5.17.2 「NDB Cluster および MySQL の権限」
セクション23.6.5 「NDB Cluster のレプリケーションの準備」
セクション23.1.4 「NDB Cluster の新機能」
セクション23.5.12 「NDB_STORED_USER での分散 MySQL 権限」
セクション13.7.1.8 「REVOKE ステートメント」
セクション13.7.7.21 「SHOW GRANTS ステートメント」
セクション6.4.1.6 「Windows プラガブル認証」
セクション6.2.1 「アカウントのユーザー名とパスワード」
セクション6.2.8 「アカウントの追加、権限の割当ておよびアカウントの削除」
セクション6.2.11 「アカウントカテゴリ」
セクション6.2.4 「アカウント名の指定」
セクション6.2.7 「アクセス制御、ステージ 2: リクエストの確認」
セクション6.2 「アクセス制御とアカウント管理」
セクション13.1.1 「アトミックデータ定義ステートメントのサポート」
セクション25.4.6 「イベントスケジューラと MySQL 権限」
セクション5.1.7 「サーバーコマンドオプション」
セクション5.1.8 「サーバーシステム変数」
セクション5.1.9.1 「システム変数権限」
セクション17.2.1.1 「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」
セクション6.1.1 「セキュリティガイドライン」
セクション2.10.1 「データディレクトリの初期化」
セクション8.2.6 「データベース権限の最適化」
セクション6.1.2.3 「パスワードおよびロギング」
セクション6.2.18 「プロキシユーザー」
セクション17.5.1.13 「レプリケーションと FLUSH」
セクション17.3.3 「レプリケーション権限チェック」
セクション17.1.2.3 「レプリケーション用ユーザーの作成」
セクション6.2.10 「ロールの使用」
セクション6.2.3 「付与テーブル」
セクション12.16 「情報関数」
セクション13.3.3 「暗黙的なコミットを発生させるステートメント」
セクション6.2.13 「権限変更が有効化される時期」
セクション15.8.2 「読み取り専用操作の InnoDB の構成」
セクション6.2.12 「部分取消しを使用した権限の制限」

GRANT ALL

セクション13.7.1.6 「GRANT ステートメント」

GRANT EVENT

セクション25.4.6 「イベントスケジューラと MySQL 権限」

GRANT PROXY

セクション6.4.1.7 「LDAP プラガブル認証」

GROUP BY

セクション15.1.1 「InnoDB テーブルを使用する利点」

H

[\[index top\]](#)

HANDLER

セクション16.8.3 「FEDERATED ストレージエンジンの注記とヒント」
セクション13.7.8.3 「FLUSH ステートメント」

[セクションA.4「MySQL 8.0 FAQ: ストアドプロシージャーおよびストアドファンクション」](#)
[セクション1.7「MySQL の標準への準拠」](#)
[セクション5.1.8「サーバーシステム変数」](#)

HANDLER ... CLOSE

[セクション13.7.7.24「SHOW OPEN TABLES ステートメント」](#)

HANDLER ... OPEN

[セクション13.7.7.24「SHOW OPEN TABLES ステートメント」](#)

HANDLER ... READ

[セクション25.8「ストアドプログラムの制約」](#)

HANDLER OPEN

[セクション13.2.4「HANDLER ステートメント」](#)
[セクション13.1.37「TRUNCATE TABLE ステートメント」](#)

HELP

[セクション13.8.3「HELP ステートメント」](#)
[セクション13.3.6「LOCK TABLES および UNLOCK TABLES ステートメント」](#)
[セクション5.1.17「サーバー側ヘルプのサポート」](#)
[セクション2.10.1「データディレクトリの初期化」](#)

|

[\[index top\]](#)

IF

[セクション13.6.7.2「DECLARE ... HANDLER ステートメント」](#)
[セクション13.6.5.2「IF ステートメント」](#)
[セクション13.6.5「フロー制御ステートメント」](#)
[セクション12.5「フロー制御関数」](#)
[セクション8.10.3「プリペアドステートメントおよびストアドプログラムのキャッシュ」](#)

IMPORT PARTITION

[セクション15.12.1「オンライン DDL 操作」](#)

IMPORT PARTITION ... TABLESPACE

[セクション13.1.9.1「ALTER TABLE パーティション操作」](#)

IMPORT TABLE

[セクション13.1.2「ALTER DATABASE ステートメント」](#)
[セクション13.2.5「IMPORT TABLE ステートメント」](#)
[セクション13.2.7「LOAD DATA ステートメント」](#)
MySQL 用語集
[セクション14.6「シリアライズディクショナリ情報 \(SDI\)」](#)

INSERT

[セクション13.1.2「ALTER DATABASE ステートメント」](#)
[セクション16.5「ARCHIVE ストレージエンジン」](#)
[セクション16.6「BLACKHOLE ストレージエンジン」](#)
[セクション13.1.20.6「CHECK 制約」](#)
[セクション16.8.2.1「CONNECTION を使用した FEDERATED テーブルの作成」](#)
[セクション13.1.15「CREATE INDEX ステートメント」](#)
[セクション13.1.17「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」](#)

セクション13.1.20.8 「CREATE TABLE および生成されるカラム」
セクション13.1.20.2 「CREATE TEMPORARY TABLE ステートメント」
セクション13.1.22 「CREATE TRIGGER ステートメント」
セクション13.1.23 「CREATE VIEW ステートメント」
セクション13.6.7.2 「DECLARE ... HANDLER ステートメント」
セクション13.2.2 「DELETE ステートメント」
セクション8.8.1 「EXPLAIN によるクエリーの最適化」
セクション13.8.2 「EXPLAIN ステートメント」
セクション8.8.2 「EXPLAIN 出力フォーマット」
セクション16.8.3 「FEDERATED ストレージエンジンの注記とヒント」
セクション13.1.20.5 「FOREIGN KEY の制約」
セクション13.7.1.6 「GRANT ステートメント」
セクション26.38 「INFORMATION_SCHEMA TABLES テーブル」
セクション26.48 「INFORMATION_SCHEMA VIEWS テーブル」
セクション15.20.9 「InnoDB memcached プラグインのトラブルシューティング」
セクション15.6.1.6 「InnoDB での AUTO_INCREMENT 処理」
セクション15.7.3 「InnoDB のさまざまな SQL ステートメントで設定されたロック」
セクション15.21.2 「InnoDB のリカバリの強制的な実行」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.1.2 「InnoDB テーブルのベストプラクティス」
セクション8.5.5 「InnoDB テーブルの一括データロード」
セクション15.15.2.1 「InnoDB トランザクションの使用および情報のロック」
セクション15.7.1 「InnoDB ロック」
セクション13.2.6.2 「INSERT ... ON DUPLICATE KEY UPDATE ステートメント」
セクション13.2.6.1 「INSERT ... SELECT ステートメント」
セクション13.2.6.3 「INSERT DELAYED ステートメント」
セクション13.2.6 「INSERT ステートメント」
セクション8.2.5.1 「INSERT ステートメントの最適化」
セクション12.18.7 「JSON スキーマ検証関数」
セクション24.2.2 「LIST パーティショニング」
セクション13.2.7 「LOAD DATA ステートメント」
セクション13.3.6 「LOCK TABLES および UNLOCK TABLES ステートメント」
セクション16.7 「MERGE ストレージエンジン」
セクション16.7.2 「MERGE テーブルの問題点」
セクション15.6.1.5 「MyISAM から InnoDB へのテーブルの変換」
セクション8.6.1 「MyISAM クエリーの最適化」
セクション16.2 「MyISAM ストレージエンジン」
セクション8.6.2 「MyISAM テーブルの一括データロード」
セクションA.11 「MySQL 8.0 FAQ: MySQL の中国語、日本語、および韓国語の文字セット」
セクションA.5 「MySQL 8.0 FAQ: トリガー」
セクションA.6 「MySQL 8.0 FAQ: ビュー」
セクションA.1 「MySQL 8.0 FAQ: 全般」
セクション2.11.4 「MySQL 8.0 での変更」
セクション1.3 「MySQL 8.0 の新機能」
セクション6.6.2 「MySQL Enterprise Encryption の使用方法と例」
セクション6.2.2 「MySQL で提供される権限」
セクション24.1 「MySQL のパーティショニングの概要」
セクション1.2.2 「MySQL の主な機能」
セクション6.2.21 「MySQL への接続の問題のトラブルシューティング」
セクション4.5.1.1 「mysql クライアントオプション」
セクションB.3.2.7 「MySQL サーバーが存在しなくなりました」
セクション5.4.4.4 「mysql データベーステーブルへの変更に対するログイン形式」
MySQL 用語集
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション23.5.13 「NDB API 統計のカウントと変数」
セクション23.1.7.3 「NDB Cluster でのトランザクション処理に関する制限」
セクション23.5.5 「NDB Cluster のローリング再起動の実行」
セクション23.1.4 「NDB Cluster の新機能」

セクション23.5.10.1「NDB Cluster ディスクデータオブジェクト」
セクション23.6.11「NDB Cluster レプリケーションの競合解決」
セクション15.9.1.6「OLTP ワークロードの圧縮」
セクション13.7.3.4「OPTIMIZE TABLE ステートメント」
セクション13.5.1「PREPARE ステートメント」
セクション1.7.3.1「PRIMARY KEY および UNIQUE インデックス制約」
セクション24.3.1「RANGE および LIST パーティションの管理」
セクション24.2.1「RANGE パーティショニング」
セクション13.2.9「REPLACE ステートメント」
セクション13.7.7.27「SHOW PROCEDURE CODE ステートメント」
セクション13.7.7.38「SHOW TABLE STATUS ステートメント」
セクション13.7.7.42「SHOW WARNINGS ステートメント」
セクション8.3.3「SPATIAL インデックス最適化」
セクション13.2.12「TABLE ステートメント」
セクション15.6.6「undo ログ」
セクション13.2.13「UPDATE ステートメント」
セクション13.2.14「VALUES ステートメント」
セクション12.24「その他の関数」
セクション26.1「はじめに」
セクション6.2.8「アカウントの追加、権限の割当ておよびアカウントの削除」
セクション6.2.11「アカウントカテゴリ」
セクション6.2.7「アクセス制御、ステージ 2: リクエストの確認」
セクション6.2「アクセス制御とアカウント管理」
セクション8.9.3「オプティマイザヒント」
セクション15.12.1「オンライン DDL 操作」
セクション10.7「カラム文字セットの変換」
セクション13.2.11「サブクエリー」
セクション5.1.11「サーバー SQL モード」
セクション5.1.19「サーバーの停止プロセス」
セクション5.1.8「サーバーシステム変数」
セクション17.2.1.1「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」
セクション25.7「ストアドプログラムバイナリロギング」
セクション27.4.6「スレッドによる事前フィルタリング」
セクション13.1.20.9「セカンダリインデックスと生成されたカラム」
セクション23.2.5「テーブルとデータを含む NDB Cluster の例」
セクション3.3.3「テーブルへのデータのロード」
セクション8.11.2「テーブルロックの問題」
セクション11.6「データ型デフォルト値」
セクション8.2.5「データ変更ステートメントの最適化」
セクション25.3「トリガーの使用」
セクション25.3.1「トリガーの構文と例」
セクション17.2.1.3「バイナリロギングでの安全および安全でないステートメントの判断」
セクション5.4.4「バイナリログ」
セクション10.8.5「バイナリ照合順序と_bin 照合順序」
セクション7.1「バックアップとリカバリの種類」
セクション7.3.1「バックアップポリシーの確立」
セクション6.1.2.3「パスワードおよびロギング」
セクション27.12.6「パフォーマンススキーマステートメントイベントテーブル」
セクション15.8.9「ページ構成」
セクション24.6「パーティショニングの制約と制限」
セクション24.4「パーティションプルーニング」
セクション24.5「パーティション選択」
セクション8.10.3「プリペアドステートメントおよびストアドプログラムのキャッシュ」
セクション8.11.4「メタデータのロック」
セクション5.6.4「リライタクエリーリライトプラグイン」
セクション5.6.4.2「リライタクエリーリライトプラグインの使用」
セクション17.5.1.1「レプリケーションと AUTO_INCREMENT」
セクション17.5.1.30「レプリケーションとサーバー SQL モード」
セクション17.5.1.14「レプリケーションとシステム関数」
セクション17.5.1.36「レプリケーションとトリガー」

セクション17.5.1.39「レプリケーションと変数」
セクション17.1.6.2「レプリケーションソースのオプションと変数」
セクション17.2.5.3「レプリケーションフィルタリングオプション間の相互作用」
セクション17.5.1.29「レプリケーション中のレプリカエラー」
セクション5.4.1「一般クエリログおよびスロークエリログの出力先の選択」
セクション8.14.3「一般的なスレッドの状態」
セクション6.2.3「付与テーブル」
セクション12.10.5「全文制限」
セクション8.11.1「内部ロック方法」
セクション8.11.3「同時挿入」
セクション8.8.4「名前付き接続の実行計画情報の取得」
セクション15.5.2「変更バッファ」
セクション12.25.3「式の処理」
セクション12.16「情報関数」
セクション8.8.3「拡張 EXPLAIN 出力形式」
セクション11.2.1「日時データ型の構文」
セクション25.5.3「更新可能および挿入可能なビュー」
セクション1.7.1「標準 SQL に対する MySQL 拡張機能」
セクション6.2.13「権限変更が有効化される時期」
セクション6.4.5.7「監査ログのフィルタリング」
セクション6.4.5.8「監査ログフィルタ定義の書込み」
セクション6.4.5.10「監査ログ参照」
セクション11.4.7「空間カラムへのデータ移入」
セクション11.1.7「範囲外およびオーバーフローの処理」
セクション15.7.2.2「自動コミット、コミットおよびロールバック」
セクション13.1.20.10「非表示カラム」

INSERT ... ON DUPLICATE KEY UPDATE

セクション16.8.3「FEDERATED ストレージエンジンの注記とヒント」
セクション15.6.1.6「InnoDB での AUTO_INCREMENT 処理」
セクション15.7.3「InnoDB のさまざまな SQL ステートメントで設定されたロック」
セクション13.2.6.2「INSERT ... ON DUPLICATE KEY UPDATE ステートメント」
セクション13.2.6「INSERT ステートメント」
セクション16.7.2「MERGE テーブルの問題点」
セクション1.3「MySQL 8.0 の新機能」
MySQL 用語集
セクション23.6.3「NDB Cluster レプリケーションの既知の問題」
セクション13.2.14「VALUES ステートメント」
セクション12.24「その他の関数」
セクション17.2.1.3「バイナリロギングでの安全および安全でないステートメントの判断」
セクション12.16「情報関数」
セクション13.1.20.10「非表示カラム」

INSERT ... SELECT

セクション15.6.1.6「InnoDB での AUTO_INCREMENT 処理」
セクション15.7.3「InnoDB のさまざまな SQL ステートメントで設定されたロック」
セクション13.2.6.2「INSERT ... ON DUPLICATE KEY UPDATE ステートメント」
セクション13.2.6.1「INSERT ... SELECT ステートメント」
セクション13.2.6「INSERT ステートメント」
セクション1.3「MySQL 8.0 の新機能」
セクション8.4.4「MySQL での内部一時テーブルの使用」
セクションB.3.7「MySQL の既知の問題」
NDB Cluster システム変数
セクション5.1.8「サーバーシステム変数」
セクション17.2.1.1「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」
セクション5.4.4「バイナリログ」
セクション24.5「パーティション選択」
セクション17.5.1.18「レプリケーションと LIMIT」
セクション8.11.3「同時挿入」

INSERT ... SELECT ON DUPLICATE KEY UPDATE

[セクション13.2.6.2「INSERT ... ON DUPLICATE KEY UPDATE ステートメント」](#)

[セクション13.2.6.1「INSERT ... SELECT ステートメント」](#)

INSERT ... SET

[セクション13.2.6「INSERT ステートメント」](#)

INSERT ... TABLE

[セクション13.2.6「INSERT ステートメント」](#)

INSERT ... VALUES

[セクション13.2.6「INSERT ステートメント」](#)

INSERT ... VALUES ROW()

[セクション13.2.6「INSERT ステートメント」](#)

INSERT DELAYED

[セクション13.2.6.3「INSERT DELAYED ステートメント」](#)

[セクション13.2.6「INSERT ステートメント」](#)

INSERT IGNORE

[セクション13.1.20.6「CHECK 制約」](#)

[セクション1.7.3.4「ENUM および SET の制約」](#)

[セクション13.2.6「INSERT ステートメント」](#)

[セクション4.5.4「mysqldump — データベースバックアッププログラム」](#)

[セクション4.5.6「mysqlpump — データベースバックアッププログラム」](#)

[セクション5.1.11「サーバー SQL モード」](#)

[セクション12.16「情報関数」](#)

INSERT IGNORE ... SELECT

[セクション13.2.6.1「INSERT ... SELECT ステートメント」](#)

INSERT INTO ... SELECT

[セクション13.1.13「CREATE EVENT ステートメント」](#)

[セクション13.2.6「INSERT ステートメント」](#)

[セクション16.3「MEMORY ストレージエンジン」](#)

[セクション1.7.2.1「SELECT INTO TABLE の違い」](#)

[セクション6.2.7「アクセス制御、ステージ 2: リクエストの確認」](#)

[セクション15.7.2.3「一貫性非ロック読み取り」](#)

[セクション6.4.5.8「監査ログフィルタ定義の書込み」](#)

INSERT INTO ... SELECT *

[セクション13.1.20.10「非表示カラム」](#)

INSERT INTO ... SELECT FROM memory_table

[セクション17.5.1.21「レプリケーションと MEMORY テーブル」](#)

INSERT INTO...SELECT

[セクション8.2.1「SELECT ステートメントの最適化」](#)

INSTALL COMPONENT

[セクション13.7.4.3「INSTALL COMPONENT ステートメント」](#)

[セクション6.2.2「MySQL で提供される権限」](#)

[セクション5.3「mysql システムスキーマ」](#)

[セクション13.7.4.5「UNINSTALL COMPONENT ステートメント」](#)

セクション13.1.1「アトミックデータ定義ステートメントのサポート」
セクション5.5.3「エラーログコンポーネント」
セクション5.4.2.1「エラーログ構成」
セクション9.6「クエリー属性」
セクション5.5.1「コンポーネントのインストールおよびアンインストール」
セクション5.5.2「コンポーネント情報の取得」
セクション5.1.8「サーバーシステム変数」
セクション6.4.3.1「パスワード検証コンポーネントのインストールおよびアンインストール」
セクション6.4.6「監査メッセージコンポーネント」

INSTALL PLUGIN

セクション6.4.2.1「Connection-Control プラグインのインストール」
セクション5.6.5.1「ddl_rewriter のインストールまたはアンインストール」
セクション5.6.5.2「ddl_rewriter プラグインオプション」
セクション13.7.8.3「FLUSH ステートメント」
セクション26.22「INFORMATION_SCHEMA PLUGINS テーブル」
セクション15.20.2「InnoDB memcached のアーキテクチャー」
セクション15.20.9「InnoDB memcached プラグインのトラブルシューティング」
セクション15.20.3「InnoDB memcached プラグインの設定」
セクション15.14「InnoDB の起動オプションおよびシステム変数」
セクション13.7.4.4「INSTALL PLUGIN ステートメント」
セクション6.4.4.5「keyring_aws Amazon Web Services キーリングプラグインの使用」
セクション6.4.1.7「LDAP プラガブル認証」
セクション12.10.9「MeCab フルテキストパーサープラグイン」
セクション1.3「MySQL 8.0 の新機能」
セクション6.4.5.2「MySQL Enterprise Audit のインストールまたはアンインストール」
セクション6.5.2「MySQL Enterprise Data Masking and De-Identification のインストールまたはアンインストール」
セクション6.4.7.2「MySQL Enterprise Firewall のインストールまたはアンインストール」
セクション8.12.3.1「MySQL のメモリーの使用方法」
セクション5.3「mysql システムスキーマ」
セクション2.9.7「MySQL ソース構成オプション」
セクション6.4.1.5「PAM プラガブル認証」
セクション13.7.7.25「SHOW PLUGINS ステートメント」
セクション13.7.4.6「UNINSTALL PLUGIN ステートメント」
セクション6.4.1.6「Windows プラガブル認証」
セクション13.1.1「アトミックデータ定義ステートメントのサポート」
セクション6.4.4.1「キーリングプラグインのインストール」
セクション5.6.7.1「クローンプラグインのインストール」
セクション5.5.1「コンポーネントのインストールおよびアンインストール」
セクション5.1.7「サーバーコマンドオプション」
セクション5.1.8「サーバーシステム変数」
セクション5.6.2「サーバープラグイン情報の取得」
セクション6.4.1.9「ソケットピア資格証明プラガブル認証」
セクション5.6.6.2「バージョントークンのインストールまたはアンインストール」
セクション6.4.3.2「パスワード検証オプションおよび変数」
セクション16.11.1「プラガブルストレージエンジンのアーキテクチャー」
セクション6.4.1.10「プラガブル認証のテスト」
セクション5.6.1「プラグインのインストールおよびアンインストール」
セクション6.4.1.8「ログインなしのプラガブル認証」
セクション13.3.3「暗黙的なコミットを発生させるステートメント」
セクション6.4.4.10「汎用キーリングキー管理関数」
セクション17.4.10.2「準同期レプリケーションのインストールと構成」
セクション17.4.10.1「準同期レプリケーション管理インタフェース」
セクション6.4.5.7「監査ログのフィルタリング」
セクション6.4.5.10「監査ログ参照」
セクション15.8.2「読み取り専用操作の InnoDB の構成」

ITERATE

セクション13.6.7.2「DECLARE ... HANDLER ステートメント」

セクション13.6.5.3 「ITERATE ステートメント」
セクション13.6.2 「ステートメントラベル」
セクション13.6.5 「フロー制御ステートメント」

K

[[index top](#)]

KILL

セクション13.7.1.6 「GRANT ステートメント」
セクション17.1.3.2 「GTID ライフサイクル」
セクション28.4.3.9 「innodb_lock_waits および x\$innodb_lock_waits のビュー」
セクション13.7.8.4 「KILL ステートメント」
セクション13.3.6 「LOCK TABLES および UNLOCK TABLES ステートメント」
セクション6.2.2 「MySQL で提供される権限」
セクションB.3.2.7 「MySQL サーバーが存在しなくなりました」
セクション28.4.3.28 「schema_table_lock_waits および x\$schema_table_lock_waits のビュー」
セクション13.7.7.29 「SHOW PROCESSLIST ステートメント」
セクション13.4.2.9 「STOP REPLICA | SLAVE ステートメント」
セクション8.14 「サーバスレッド (プロセス) 情報の確認」
セクション17.5.1.34 「レプリケーションとトランザクションの非一貫性」
セクション8.14.3 「一般的なスレッドの状態」
セクション6.3.1 「暗号化接続を使用するための MySQL の構成」

KILL CONNECTION

セクション13.7.8.4 「KILL ステートメント」
セクション13.4.2.9 「STOP REPLICA | SLAVE ステートメント」
セクション5.1.19 「サーバーの停止プロセス」

KILL QUERY

セクション13.8.2 「EXPLAIN ステートメント」
セクション13.7.8.4 「KILL ステートメント」
セクション13.4.2.9 「STOP REPLICA | SLAVE ステートメント」
セクション13.2.15 「WITH (共通テーブル式)」
セクション12.24 「その他の関数」
セクション5.1.19 「サーバーの停止プロセス」

KILL QUERY processlist_id

セクション5.6.7.10 「クローニング操作の停止」

L

[[index top](#)]

LEAVE

セクション13.6.7.2 「DECLARE ... HANDLER ステートメント」
セクション13.6.5.4 「LEAVE ステートメント」
セクション13.6.5.5 「LOOP ステートメント」
セクション13.6.5.7 「RETURN ステートメント」
セクション13.6.2 「ステートメントラベル」
セクション25.8 「ストアプログラムの制約」
セクション13.6.5 「フロー制御ステートメント」

LOAD DATA

セクション13.1.2 「ALTER DATABASE ステートメント」
セクション13.1.20.6 「CHECK 制約」
セクション13.1.22 「CREATE TRIGGER ステートメント」

セクション11.3.5 「ENUM 型」
セクション13.1.20.5 「FOREIGN KEY の制約」
セクション13.2.5 「IMPORT TABLE ステートメント」
セクション15.6.1.6 「InnoDB での AUTO_INCREMENT 処理」
セクション8.2.5.1 「INSERT ステートメントの最適化」
セクション6.1.6 「LOAD DATA LOCAL のセキュリティー上の考慮事項」
セクション13.2.7 「LOAD DATA ステートメント」
セクション13.2.8 「LOAD XML ステートメント」
セクション16.3 「MEMORY ストレージエンジン」
セクション8.6.2 「MyISAM テーブルの一括データロード」
セクション16.2.1 「MyISAM 起動オプション」
セクション1.3 「MySQL 8.0 の新機能」
セクションB.3.3.5 「MySQL が一時ファイルを格納する場所」
セクションB.3.3.4 「MySQL が満杯のディスクを処理する方法」
セクション6.2.2 「MySQL で提供される権限」
セクション24.1 「MySQL のパーティショニングの概要」
セクションB.3.7 「MySQL の既知の問題」
セクション6.2.21 「MySQL への接続の問題のトラブルシューティング」
セクション4.5.1.1 「mysql クライアントオプション」
セクション5.4.4.4 「mysql データベーステーブルへの変更に対するロギング形式」
セクション4.1 「MySQL プログラムの概要」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション23.1.7.3 「NDB Cluster でのトランザクション処理に関する制限」
セクション23.5.5 「NDB Cluster のローリング再起動の実行」
セクション23.4.13 「ndb_import — NDB への CSV データのインポート」
セクション23.4.27 「ndb_show_tables — NDB テーブルのリストの表示」
セクション9.1.7 「NULL 値」
セクションB.3.4.3 「NULL 値に関する問題」
セクション17.1.6.3 「Replica Server のオプションと変数」
セクション13.2.10.1 「SELECT ... INTO ステートメント」
セクション13.7.7.42 「SHOW WARNINGS ステートメント」
セクション2.3.7 「Windows プラットフォームの制限事項」
セクション3.3.4.1 「すべてのデータの選択」
セクション5.1.11 「サーバー SQL モード」
セクション5.1.7 「サーバーコマンドオプション」
セクション5.1.8 「サーバーシステム変数」
セクション13.2.11.1 「スカラーオペランドとしてのサブクエリー」
セクション25.8 「ストアプログラムの制約」
セクション3.3.3 「テーブルへのデータのロード」
セクション7.2 「データベースバックアップ方法」
セクション10.3.3 「データベース文字セットおよび照合順序」
セクション25.3 「トリガーの使用」
セクション17.2.1.3 「バイナリロギングでの安全および安全でないステートメントの判断」
セクション17.3.2.1 「バイナリログの暗号化の範囲」
セクション7.1 「バックアップとリカバリの種類」
セクション24.6 「パーティショニングの制約と制限」
セクション24.5 「パーティション選択」
セクション9.4 「ユーザー定義変数」
セクション17.4.1.2 「レプリカからの RAW データのバックアップ」
セクション17.3.3.1 「レプリケーション PRIVILEGE_CHECKS_USER アカウントの権限」
セクション8.14.6 「レプリケーション SQL スレッドの状態」
セクション17.5.1.19 「レプリケーションと LOAD DATA」
セクション7.4.4 「区切りテキストフォーマットバックアップのリロード」
セクション8.11.3 「同時挿入」
セクション12.16 「情報関数」
セクション6.1.3 「攻撃者に対する MySQL のセキュアな状態の維持」
セクション10.11 「文字セットの制約」
セクション13.3.3 「暗黙的なコミットを発生させるステートメント」

[セクション1.7.1「標準 SQL に対する MySQL 拡張機能」](#)
[セクション6.4.5.11「監査ログの制限」](#)
[セクション6.4.5.4「監査ログファイル形式」](#)
[セクション11.1.7「範囲外およびオーバーフローの処理」](#)
[セクション13.1.20.10「非表示カラム」](#)

LOAD DATA ... IGNORE

[セクション13.1.20.6「CHECK 制約」](#)

LOAD DATA ... REPLACE

[セクション13.2.9「REPLACE ステートメント」](#)

LOAD DATA INFILE

[セクション23.1.4「NDB Cluster の新機能」](#)

LOAD DATA LOCAL

[セクション6.1.6「LOAD DATA LOCAL のセキュリティー上の考慮事項」](#)
[セクション13.2.7「LOAD DATA ステートメント」](#)
[セクション4.5.1.1「mysql クライアントオプション」](#)
[セクション2.9.7「MySQL ソース構成オプション」](#)
[セクション4.6.8「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」](#)
[セクション5.1.8「サーバーシステム変数」](#)

LOAD INDEX INTO CACHE

[セクション13.7.8.2「CACHE INDEX ステートメント」](#)
[セクション13.7.8.5「LOAD INDEX INTO CACHE ステートメント」](#)
[セクション8.10.2.4「インデックスプリロード」](#)
[セクション13.3.3「暗黙的なコミットを発生させるステートメント」](#)

LOAD INDEX INTO CACHE ... IGNORE LEAVES

[セクション13.7.8.5「LOAD INDEX INTO CACHE ステートメント」](#)

LOAD XML

[セクション13.1.2「ALTER DATABASE ステートメント」](#)
[セクション13.1.20.6「CHECK 制約」](#)
[セクション13.2.8「LOAD XML ステートメント」](#)
[セクション1.3「MySQL 8.0 の新機能」](#)
[セクション24.1「MySQL のパーティショニングの概要」](#)
[セクション5.1.11「サーバー SQL モード」](#)
[セクション24.5「パーティション選択」](#)
[セクション13.1.20.10「非表示カラム」](#)

LOAD XML ... IGNORE

[セクション13.1.20.6「CHECK 制約」](#)

LOAD XML LOCAL

[セクション13.2.8「LOAD XML ステートメント」](#)

LOCK INSTANCE FOR BACKUP

[セクション1.3「MySQL 8.0 の新機能」](#)
[セクション6.2.2「MySQL で提供される権限」](#)
[セクション5.1.8「サーバーシステム変数」](#)

LOCK TABLE

[セクションB.3.6.1「ALTER TABLE での問題」](#)
[セクション8.14.3「一般的なスレッドの状態」](#)

セクション8.11.3「同時挿入」

LOCK TABLES

セクション13.1.10「ALTER TABLESPACE ステートメント」
セクション13.1.12「CREATE DATABASE ステートメント」
セクション13.1.20.3「CREATE TABLE ... LIKE ステートメント」
セクション13.1.22「CREATE TRIGGER ステートメント」
セクション13.7.8.3「FLUSH ステートメント」
セクション13.1.20.5「FOREIGN KEY の制約」
セクション13.7.1.6「GRANT ステートメント」
セクション15.7.3「InnoDB のさまざまな SQL ステートメントで設定されたロック」
セクション15.7.5「InnoDB のデッドロック」
セクション15.14「InnoDB の起動オプションおよびシステム変数」
セクション15.1.2「InnoDB テーブルのベストプラクティス」
セクション13.3.6「LOCK TABLES および UNLOCK TABLES ステートメント」
セクション16.7.2「MERGE テーブルの問題点」
セクション8.6.2「MyISAM テーブルの一括データロード」
セクション6.2.2「MySQL で提供される権限」
セクション4.5.4「mysqldump — データベースバックアッププログラム」
セクション4.5.6「mysqlpump — データベースバックアッププログラム」
セクション13.1.36「RENAME TABLE ステートメント」
セクション13.3.1「START TRANSACTION、COMMIT および ROLLBACK ステートメント」
セクション5.1.8「サーバーシステム変数」
セクション25.8「ストアードプログラムの制約」
セクション8.11.2「テーブルロックの問題」
セクション15.7.5.3「デッドロックを最小化および処理する方法」
セクション15.7.5.2「デッドロック検出」
セクション8.11.4「メタデータのロック」
セクション5.4.1「一般クエリログおよびスロークエリログの出力先の選択」
セクション15.6.3.3「一般テーブルスペース」
セクション8.14.3「一般的なスレッドの状態」
セクション8.11.1「内部ロック方法」
セクション13.3.3「暗黙的なコミットを発生させるステートメント」
セクション23.1.7.10「複数の NDB Cluster ノードに関する制限事項」
セクション16.2.4.2「適切に閉じられなかったテーブルの問題」

LOCK TABLES ... READ

セクション13.7.8.3「FLUSH ステートメント」
セクション15.7.3「InnoDB のさまざまな SQL ステートメントで設定されたロック」
セクション15.14「InnoDB の起動オプションおよびシステム変数」
セクション8.11.4「メタデータのロック」

LOCK TABLES ... WRITE

セクション15.7.3「InnoDB のさまざまな SQL ステートメントで設定されたロック」
セクション15.14「InnoDB の起動オプションおよびシステム変数」
セクション15.7.1「InnoDB ロック」

LOCK TABLES READ

セクション13.1.20.5「FOREIGN KEY の制約」
セクション13.3.6「LOCK TABLES および UNLOCK TABLES ステートメント」

LOCK TABLES WRITE

セクション13.1.20.5「FOREIGN KEY の制約」
セクション13.3.6「LOCK TABLES および UNLOCK TABLES ステートメント」

LOOP

セクション13.6.5.3「ITERATE ステートメント」

セクション13.6.5.4 「LEAVE ステートメント」
セクション13.6.5.5 「LOOP ステートメント」
セクション13.6.2 「ステートメントラベル」
セクション13.6.5 「フロー制御ステートメント」

O

[\[index top\]](#)

OPTIMIZE PARTITION

セクション15.12.1 「オンライン DDL 操作」

OPTIMIZE TABLE

セクション16.5 「ARCHIVE ストレージエンジン」
セクション13.2.2 「DELETE ステートメント」
セクション26.51.13 「INFORMATION_SCHEMA INNODB_FT_BEING_DELETED テーブル」
セクション26.51.14 「INFORMATION_SCHEMA INNODB_FT_CONFIG テーブル」
セクション26.51.16 「INFORMATION_SCHEMA INNODB_FT_DELETED テーブル」
セクション26.51.17 「INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE テーブル」
セクション26.51.18 「INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE テーブル」
セクション15.15.4 「InnoDB INFORMATION_SCHEMA FULLTEXT インデックステーブル」
セクション15.10 「InnoDB の行フォーマット」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.6.1.3 「InnoDB テーブルのインポート」
セクション15.9.2 「InnoDB ページ圧縮」
セクション13.7.8.4 「KILL ステートメント」
セクション13.3.5 「LOCK INSTANCE FOR BACKUP および UNLOCK INSTANCE ステートメント」
セクション16.7.2 「MERGE テーブルの問題点」
セクション8.6.1 「MyISAM クエリーの最適化」
セクション7.6 「MyISAM テーブルの保守とクラッシュリカバリ」
セクション7.6.4 「MyISAM テーブルの最適化」
セクション7.6.5 「MyISAM テーブル保守スケジュールのセットアップ」
セクション4.6.4.1 「myisamchk の一般オプション」
セクションB.3.3.4 「MySQL が満杯のディスクを処理する方法」
セクション6.2.2 「MySQL で提供される権限」
セクション12.10.6 「MySQL の全文検索の微調整」
セクション5.9.1 「MySQL サーバーのデバッグ」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション23.5.11 「NDB Cluster での ALTER TABLE を使用したオンライン操作」
セクション23.1.7.2 「NDB Cluster と標準の MySQL 制限の制限と相違点」
NDB Cluster の MySQL Server オプション
セクション23.5.10.2 「NDB Cluster ディスクデータストレージの要件」
セクション23.5.7.2 「NDB Cluster データノードのオンラインでの追加: 基本手順」
セクション23.5.7.3 「NDB Cluster データノードのオンラインでの追加: 詳細な例」
セクション13.7.3.4 「OPTIMIZE TABLE ステートメント」
セクション8.12.2.2 「Unix 上の MyISAM へのシンボリックリンクの使用」
セクション8.2.5.2 「UPDATE ステートメントの最適化」
セクション8.2.7 「その他の最適化のヒント」
セクション15.12.6 「オンライン DDL の制限事項」
セクション5.1.19 「サーバーの停止プロセス」
セクション5.1.7 「サーバーコマンドオプション」
セクション5.1.8 「サーバーシステム変数」
セクション5.4.5 「スロークエリーログ」
セクション24.6 「パーティショニングの制約と制限」
セクション24.3.4 「パーティションの保守」
セクション17.5.1.13 「レプリケーションと FLUSH」
セクション8.14.3 「一般的なスレッドの状態」
セクション16.2.3.2 「動的テーブルの特徴」
セクション13.3.3 「暗黙的なコミットを発生させるステートメント」

[セクション1.7.1「標準 SQL に対する MySQL 拡張機能」](#)
[セクション16.2.3.1「静的 \(固定長\) テーブルの特長」](#)

ORDER BY

[セクション15.1.1「InnoDB テーブルを使用する利点」](#)

P

[\[index top\]](#)

PARTITION BY

[セクション15.12.1「オンライン DDL 操作」](#)

PREPARE

[セクション13.2.1「CALL ステートメント」](#)
[セクション13.5.3「DEALLOCATE PREPARE ステートメント」](#)
[セクション13.5.2「EXECUTE ステートメント」](#)
[セクション1.3「MySQL 8.0 の新機能」](#)
[セクション13.5.1「PREPARE ステートメント」](#)
[セクション27.12.6.4「prepared_statements_instances テーブル」](#)
[セクション9.6「クエリー属性」](#)
[セクション5.1.10「サーバーステータス変数」](#)
[セクション25.8「ストアプログラムの制約」](#)
[セクション13.5「プリペアドステートメント」](#)
[セクション8.10.3「プリペアドステートメントおよびストアプログラムのキャッシュ」](#)
[セクション8.11.4「メタデータのロック」](#)
[セクション9.2.3「識別子の大文字と小文字の区別」](#)

PURGE BINARY LOGS

[セクション13.7.1.6「GRANT ステートメント」](#)
[セクション6.2.2「MySQL で提供される権限」](#)
[セクション4.5.4「mysqldump — データベースバックアッププログラム」](#)
[セクション13.4.1.1「PURGE BINARY LOGS ステートメント」](#)
[セクション13.4.1.2「RESET MASTER ステートメント」](#)
[セクション5.4.6「サーバーログの保守」](#)
[セクション17.1.6.4「バイナリログのオプションと変数」](#)
[セクション5.4.4「バイナリログ」](#)
[セクション7.3.1「バックアップポリシーの確立」](#)
[セクション17.1.3.5「フェイルオーバーおよびスケールアウトでの GTID の使用」](#)

PURGE BINARY LOGS TO

[セクション13.4.1.2「RESET MASTER ステートメント」](#)

R

[\[index top\]](#)

REBUILD PARTITION

[セクション15.12.1「オンライン DDL 操作」](#)

RELEASE SAVEPOINT

[セクション27.12.7.1「events_transactions_current テーブル」](#)
[セクション13.3.4「SAVEPOINT、ROLLBACK TO SAVEPOINT および RELEASE SAVEPOINT ステートメント」](#)
[セクション27.12.7「パフォーマンススキーマのトランザクションテーブル」](#)

REMOVE PARTITIONING

[セクション15.12.1「オンライン DDL 操作」](#)

RENAME TABLE

[セクション13.1.2 「ALTER DATABASE ステートメント」](#)
[セクション13.1.9 「ALTER TABLE ステートメント」](#)
[セクション13.7.3.1 「ANALYZE TABLE ステートメント」](#)
[セクション13.2.2 「DELETE ステートメント」](#)
[セクション15.6.1.4 「InnoDB テーブルの移動またはコピー」](#)
[セクション4.5.4 「mysqldump — データベースバックアッププログラム」](#)
[セクション4.5.6 「mysqlpump — データベースバックアッププログラム」](#)
[セクション13.1.36 「RENAME TABLE ステートメント」](#)
[セクション8.12.2.2 「Unix 上の MyISAM へのシンボリックリンクの使用」](#)
[セクション2.11.5 「アップグレード用のインストールの準備」](#)
[セクション13.1.1 「アトミックデータ定義ステートメントのサポート」](#)
[セクション15.12.1 「オンライン DDL 操作」](#)
[セクション8.11.4 「メタデータのロック」](#)
[セクション5.4.1 「一般クエリーログおよびスロークエリーログの出力先の選択」](#)
[セクション8.14.3 「一般的なスレッドの状態」](#)
[セクション13.3.3 「暗黙的なコミットを発生させるステートメント」](#)
[セクション1.7.1 「標準 SQL に対する MySQL 拡張機能」](#)

RENAME USER

[セクション17.5.1.8 「CURRENT_USER\(\) のレプリケーション」](#)
[セクション13.7.1.6 「GRANT ステートメント」](#)
[セクション6.2.2 「MySQL で提供される権限」](#)
[セクション5.4.4.4 「mysql データベーステーブルへの変更に対するロギング形式」](#)
[セクション13.7.1.7 「RENAME USER ステートメント」](#)
[セクション6.4.1.2 「SHA-2 プラガブル認証のキャッシュ」](#)
[セクション25.4.6 「イベントスケジューラと MySQL 権限」](#)
[セクション25.6 「ストアオブジェクトのアクセス制御」](#)
[セクション6.2.10 「ロールの使用」](#)
[セクション12.16 「情報関数」](#)
[セクション13.3.3 「暗黙的なコミットを発生させるステートメント」](#)
[セクション6.2.13 「権限変更が有効化される時期」](#)

REORGANIZE PARTITION

[セクション15.12.1 「オンライン DDL 操作」](#)

REPAIR PARTITION

[セクション15.12.1 「オンライン DDL 操作」](#)

REPAIR TABLE

[セクションB.3.6.1 「ALTER TABLE での問題」](#)
[セクション13.1.9 「ALTER TABLE ステートメント」](#)
[セクション13.1.9.1 「ALTER TABLE パーティション操作」](#)
[セクション16.5 「ARCHIVE ストレージエンジン」](#)
[セクション13.7.3.2 「CHECK TABLE ステートメント」](#)
[セクション16.4.1 「CSV テーブルの修復と確認」](#)
[セクション13.2.5 「IMPORT TABLE ステートメント」](#)
[セクション13.7.8.4 「KILL ステートメント」](#)
[セクション13.2.7 「LOAD DATA ステートメント」](#)
[セクション13.3.5 「LOCK INSTANCE FOR BACKUP および UNLOCK INSTANCE ステートメント」](#)
[セクション16.7.2 「MERGE テーブルの問題点」](#)
[セクション7.6 「MyISAM テーブルの保守とクラッシュリカバリ」](#)
[セクション7.6.3 「MyISAM テーブルの修復方法」](#)
[セクション16.2.4.1 「MyISAM テーブルの破損」](#)
[セクション7.6.5 「MyISAM テーブル保守スケジュールのセットアップ」](#)
[セクション16.2.1 「MyISAM 起動オプション」](#)
[セクション4.6.4 「mysamchk — MyISAM テーブルメンテナンスユーティリティ」](#)

セクション4.6.4.1 「myisamchk の一般オプション」
セクションB.3.3.4 「MySQL が満杯のディスクを処理する方法」
セクション6.2.2 「MySQL で提供される権限」
セクション12.10.6 「MySQL の全文検索の微調整」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
NDB Cluster の新しいバージョンへの NDB バックアップの復元
セクション13.7.3.5 「REPAIR TABLE ステートメント」
セクション8.6.3 「REPAIR TABLE ステートメントの最適化」
セクション8.12.2.2 「Unix 上の MyISAM へのシンボリックリンクの使用」
セクション2.11.5 「アップグレード用のインストールの準備」
セクション5.1.19 「サーバーの停止プロセス」
セクション5.1.8 「サーバーシステム変数」
セクション5.4.5 「スロークエリログ」
セクション2.11.13 「テーブルまたはインデックスの再作成または修復」
セクション7.2 「データベースバックアップ方法」
セクション24.6 「パーティショニングの制約と制限」
セクション24.3.3 「パーティションとサブパーティションをテーブルと交換する」
セクション24.3.4 「パーティションの保守」
セクション17.5.1.13 「レプリケーションと FLUSH」
セクション17.5.1.25 「レプリケーションと REPAIR TABLE」
セクション8.14.3 「一般的なスレッドの状態」
セクション8.11.5 「外部ロック」
セクション13.3.3 「暗黙的なコミットを発生させるステートメント」
セクション1.7.1 「標準 SQL に対する MySQL 拡張機能」
セクション1.6 「質問またはバグをレポートする方法」
セクション16.2.4.2 「適切に閉じられなかったテーブルの問題」

REPEAT

セクション13.6.7.2 「DECLARE ... HANDLER ステートメント」
セクション13.6.5.3 「ITERATE ステートメント」
セクション13.6.5.4 「LEAVE ステートメント」
セクション13.6.5.6 「REPEAT ステートメント」
セクション13.6.2 「ステートメントラベル」
セクション25.1 「ストアードプログラムの定義」
セクション13.6.5 「フロー制御ステートメント」

REPLACE

セクション13.1.2 「ALTER DATABASE ステートメント」
セクション16.5 「ARCHIVE ストレージエンジン」
セクション13.1.20.6 「CHECK 制約」
セクション13.1.20.8 「CREATE TABLE および生成されるカラム」
セクション13.1.22 「CREATE TRIGGER ステートメント」
セクション8.8.1 「EXPLAIN によるクエリーの最適化」
セクション13.8.2 「EXPLAIN ステートメント」
セクション8.8.2 「EXPLAIN 出力フォーマット」
セクション15.6.1.6 「InnoDB での AUTO_INCREMENT 処理」
セクション15.7.3 「InnoDB のさまざまな SQL ステートメントで設定されたロック」
セクション13.2.6 「INSERT ステートメント」
セクション16.7.2 「MERGE テーブルの問題点」
セクション1.3 「MySQL 8.0 の新機能」
セクション24.1 「MySQL のパーティショニングの概要」
セクション1.2.2 「MySQL の主な機能」
セクションB.3.7 「MySQL の既知の問題」
セクションB.3.2.7 「MySQL サーバーが存在しなくなりました」
セクション5.4.4.4 「mysql データベーステーブルへの変更に対するログイン形式」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.6 「mysqldump — データベースバックアッププログラム」
セクション23.5.14.39 「ndbinfo operations_per_fragment テーブル」
セクション13.2.9 「REPLACE ステートメント」

[セクション13.2.12「TABLE ステートメント」](#)
[セクション13.2.13「UPDATE ステートメント」](#)
[セクション13.2.14「VALUES ステートメント」](#)
[セクション8.9.3「オプティマイザヒント」](#)
[セクション17.2.1.1「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」](#)
[セクション11.6「データ型デフォルト値」](#)
[セクション24.6「パーティショニングの制約と制限」](#)
[セクション24.5「パーティション選択」](#)
[セクション5.6.4「リライタクエリーリライトプラグイン」](#)
[セクション5.6.4.2「リライタクエリーリライトプラグインの使用」](#)
[セクション8.8.4「名前付き接続の実行計画情報の取得」](#)
[セクション12.16「情報関数」](#)
[セクション8.8.3「拡張 EXPLAIN 出力形式」](#)
[セクション1.7.1「標準 SQL に対する MySQL 拡張機能」](#)
[セクション6.4.5.8「監査ログフィルタ定義の書込み」](#)
[セクション13.1.20.10「非表示カラム」](#)

REPLACE ... SELECT

[セクション15.6.1.6「InnoDB での AUTO_INCREMENT 処理」](#)
[セクションB.3.7「MySQL の既知の問題」](#)

REPLACE INTO ... SELECT *

[セクション13.1.20.10「非表示カラム」](#)

RESET

[セクション13.7.8.3「FLUSH ステートメント」](#)
[セクション13.7.8.7「RESET PERSIST ステートメント」](#)
[セクション13.7.8.6「RESET ステートメント」](#)
[セクション13.3.3「暗黙的なコミットを発生させるステートメント」](#)
[セクション1.7.1「標準 SQL に対する MySQL 拡張機能」](#)

RESET MASTER

[セクション17.1.5.2「GTID ベースのレプリケーション用のマルチソースレプリカのプロビジョニング」](#)
[セクション17.1.3.2「GTID ライフサイクル」](#)
[セクション17.1.3.1「GTID 形式および格納」](#)
[セクション6.2.2「MySQL で提供される権限」](#)
[セクション23.1.4「NDB Cluster の新機能」](#)
[セクション23.6.3「NDB Cluster レプリケーションの既知の問題」](#)
[セクション27.12.11.6「replication_applier_status_by_coordinator テーブル」](#)
[セクション27.12.11.7「replication_applier_status_by_worker テーブル」](#)
[セクション27.12.11.2「replication_connection_status テーブル」](#)
[セクション13.4.1.2「RESET MASTER ステートメント」](#)
[セクション13.4.2.5「RESET REPLICA | SLAVE ステートメント」](#)
[セクション13.7.8.6「RESET ステートメント」](#)
[セクション13.7.7.35「SHOW REPLICA | SLAVE STATUS ステートメント」](#)
[セクション17.1.6.5「グローバルトランザクション ID システム変数」](#)
[セクション5.4.4「バイナリログ」](#)
[セクション17.4.8「フェイルオーバー中のソースの切替え」](#)
[セクション17.1.5.7「マルチソースレプリカのリセット」](#)

RESET PERSIST

[セクション6.4.1.7「LDAP プラガブル認証」](#)
[セクション13.7.8.7「RESET PERSIST ステートメント」](#)
[セクション13.7.8.6「RESET ステートメント」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション5.1.9.1「システム変数権限」](#)
[セクション18.4.2.2「トランザクション一貫性保証の構成」](#)
[セクション13.7.6.1「変数代入の SET 構文」](#)

セクション13.3.3 「暗黙的なコミットを発生させるステートメント」
セクション5.1.9.3 「永続化されるシステム変数」

RESET PERSIST var_name

セクション5.6.1 「プラグインのインストールおよびアンインストール」

RESET REPLICA

セクション13.4.2.3 「CHANGE REPLICATION SOURCE TO ステートメント」
セクション13.4.2.5 「RESET REPLICA | SLAVE ステートメント」
セクション13.4.2.6 「RESET SLAVE | REPLICA ステートメント」

RESET REPLICA ALL

セクション13.4.2.3 「CHANGE REPLICATION SOURCE TO ステートメント」

RESET REPLICA | SLAVE

セクション13.4.2.1 「CHANGE MASTER TO ステートメント」
セクション17.1.3.2 「GTID ライフサイクル」
セクション6.2.2 「MySQL で提供される権限」
セクション23.6.3 「NDB Cluster レプリケーションの既知の問題」
セクション27.12.11.6 「replication_applier_status_by_coordinator テーブル」
セクション27.12.11.7 「replication_applier_status_by_worker テーブル」
セクション27.12.11.2 「replication_connection_status テーブル」
セクション13.4.1.2 「RESET MASTER ステートメント」
セクション13.7.8.6 「RESET ステートメント」
セクション13.7.7.35 「SHOW REPLICA | SLAVE STATUS ステートメント」
セクション13.4.2.7 「START REPLICA | SLAVE ステートメント」
セクション17.1.5.7 「マルチソースレプリカのリセット」
セクション17.1.6 「レプリケーションおよびバイナリロギングのオプションと変数」
セクション17.5.1.34 「レプリケーションとトランザクションの非一貫性」
セクション17.2.4.2 「レプリケーションメタデータリポジトリ」
セクション17.3.3 「レプリケーション権限チェック」
セクション17.2.2.2 「以前のレプリケーションステートメントとの互換性」
セクション17.2.2.1 「単一チャンネルで操作するためのコマンド」
セクション13.3.3 「暗黙的なコミットを発生させるステートメント」
セクション17.4.11 「遅延レプリケーション」

RESET REPLICA | SLAVE ALL

セクション13.4.2.1 「CHANGE MASTER TO ステートメント」
セクション13.4.2.2 「CHANGE REPLICATION FILTER ステートメント」
セクション17.1.5.7 「マルチソースレプリカのリセット」
セクション17.2.5.4 「レプリケーションチャンネルベースのフィルタ」

RESET SLAVE

セクション17.1.3.6 「GTID のないソースから GTID のあるレプリカへのレプリケーション」
NDB Cluster システム変数
セクション13.4.2.5 「RESET REPLICA | SLAVE ステートメント」
セクション13.4.2.6 「RESET SLAVE | REPLICA ステートメント」

RESET SLAVE ALL

セクション17.1.3.6 「GTID のないソースから GTID のあるレプリカへのレプリケーション」

RESIGNAL

セクション13.6.7.1 「DECLARE ... CONDITION ステートメント」
セクション13.6.7.2 「DECLARE ... HANDLER ステートメント」
セクション13.6.7.3 「GET DIAGNOSTICS ステートメント」
セクション13.6.7.7 「MySQL の診断領域」

セクション13.6.7.4 「RESIGNAL ステートメント」
セクション13.6.7.5 「SIGNAL ステートメント」
セクション25.8 「ストアプログラムの制約」
セクション13.6.7.6 「ハンドラのスコープに関するルール」
セクション13.6.7 「条件の処理」
セクション13.6.7.8 「条件の処理と OUT または INOUT パラメータ」
セクション13.6.8 「条件処理の制約」

RESTART

セクション2.3 「Microsoft Windows に MySQL をインストールする」
セクション6.2.2 「MySQL で提供される権限」
セクション23.1.4 「NDB Cluster の新機能」
セクション13.7.8.8 「RESTART ステートメント」
セクション5.1.7 「サーバーコマンドオプション」
セクション5.1.10 「サーバーステータス変数」
セクション5.6.7.3 「リモートデータのクローニング」
セクション5.1.9.3 「永続化されるシステム変数」

RETURN

セクション13.1.17 「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」
セクション13.6.5.5 「LOOP ステートメント」
セクション13.6.7.7 「MySQL の診断領域」
セクション13.6.5.7 「RETURN ステートメント」
セクション13.6.7.5 「SIGNAL ステートメント」
セクション25.8 「ストアプログラムの制約」
セクション13.6.5 「フロー制御ステートメント」
セクション8.10.3 「プリペアドステートメントおよびストアプログラムのキャッシュ」

REVOKE

セクション17.5.1.8 「CURRENT_USER() のレプリケーション」
セクション13.7.8.3 「FLUSH ステートメント」
セクション13.7.1.6 「GRANT ステートメント」
セクション5.1.13 「IPv6 サポート」
セクションA.14 「MySQL 8.0 FAQ: レプリケーション」
セクション6.2.2 「MySQL で提供される権限」
セクション1.7.2 「MySQL と標準 SQL との違い」
セクション8.12.3.1 「MySQL のメモリーの使用法」
セクション17.5.1.22 「mysql システムスキーマのレプリケーション」
セクション5.4.4.4 「mysql データベーステーブルへの変更に対するロギング形式」

MySQL 用語集

セクション23.5.17.2 「NDB Cluster および MySQL の権限」
セクション23.5.12 「NDB_STORED_USER での分散 MySQL 権限」
セクション13.7.1.8 「REVOKE ステートメント」
セクション6.2.1 「アカウントのユーザー名とパスワード」
セクション6.2.8 「アカウントの追加、権限の割当ておよびアカウントの削除」
セクション6.2 「アクセス制御とアカウント管理」
セクション13.1.1 「アトミックデータ定義ステートメントのサポート」
セクション25.4.6 「イベントスケジューラと MySQL 権限」
セクション5.1.8 「サーバーシステム変数」
セクション5.1.9.1 「システム変数権限」
セクション17.2.1.1 「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」
セクション6.1.1 「セキュリティガイドライン」
セクション6.2.18 「プロキシユーザー」
セクション6.2.10 「ロールの使用」
セクション6.2.3 「付与テーブル」
セクション12.16 「情報関数」
セクション13.3.3 「暗黙的なコミットを発生させるステートメント」
セクション6.2.13 「権限変更が有効化される時期」
セクション15.8.2 「読み取り専用操作の InnoDB の構成」

[セクション6.2.12 「部分取消しを使用した権限の制限」](#)

REVOKE ALL PRIVILEGES

[セクション13.7.1.6 「GRANT ステートメント」](#)
[セクション6.2.2 「MySQL で提供される権限」](#)
[セクション6.2.10 「ロールの使用」](#)

ROLLBACK

[セクション27.12.7.1 「events_transactions_current テーブル」](#)
[セクション4.6.1 「ibd2sdi — InnoDB テーブルスペース SDI 抽出ユーティリティ」](#)
[セクション15.2 「InnoDB および ACID モデル」](#)
[セクション15.6.1.6 「InnoDB での AUTO_INCREMENT 処理」](#)
[セクション15.21.4 「InnoDB のエラー処理」](#)
[セクション13.3.6 「LOCK TABLES および UNLOCK TABLES ステートメント」](#)
[セクション15.6.1.5 「MyISAM から InnoDB へのテーブルの変換」](#)
[セクション1.3 「MySQL 8.0 の新機能」](#)
[セクション17.1.6.3 「Replica Server のオプションと変数」](#)
[セクション13.3.4 「SAVEPOINT、ROLLBACK TO SAVEPOINT および RELEASE SAVEPOINT ステートメント」](#)
[セクション13.3.1 「START TRANSACTION、COMMIT および ROLLBACK ステートメント」](#)
[セクション5.1.8 「サーバースステム変数」](#)
[セクション25.7 「ストアードプログラムバイナリロギング」](#)
[セクション13.3 「トランザクションステートメントおよびロックステートメント」](#)
[セクション25.3.1 「トリガーの構文と例」](#)
[セクション5.4.4 「バイナリログ」](#)
[セクション27.12.7 「パフォーマンススキーマのトランザクションテーブル」](#)
[セクション17.5.1.35 「レプリケーションとトランザクション」](#)
[セクション13.3.2 「ロールバックできないステートメント」](#)
[セクション12.16 「情報関数」](#)
[セクション13.3.3 「暗黙的なコミットを発生させるステートメント」](#)
[セクション15.7.2.2 「自動コミット、コミットおよびロールバック」](#)
[セクションB.3.4.5 「非トランザクションテーブルのロールバックの失敗」](#)

ROLLBACK TO SAVEPOINT

[セクション27.12.7.1 「events_transactions_current テーブル」](#)
[セクション13.3.4 「SAVEPOINT、ROLLBACK TO SAVEPOINT および RELEASE SAVEPOINT ステートメント」](#)
[セクション27.12.7 「パフォーマンススキーマのトランザクションテーブル」](#)

ROLLBACK to SAVEPOINT

[セクション25.3.1 「トリガーの構文と例」](#)

S

[\[index top\]](#)

SAVEPOINT

[セクション27.12.7.1 「events_transactions_current テーブル」](#)
[セクション13.3.4 「SAVEPOINT、ROLLBACK TO SAVEPOINT および RELEASE SAVEPOINT ステートメント」](#)
[セクション27.12.7 「パフォーマンススキーマのトランザクションテーブル」](#)

SE PERSIST_ONLY

[セクション4.2.2.2 「オプションファイルの使用」](#)

SELECT

[セクション3.6.7 「2 つのキーを使用した検索」](#)
[セクション13.1.2 「ALTER DATABASE ステートメント」](#)
[セクション13.1.9 「ALTER TABLE ステートメント」](#)
[セクション13.1.11 「ALTER VIEW ステートメント」](#)

セクション16.5「ARCHIVE ストレージエンジン」
セクション8.3.9「B ツリーインデックスとハッシュインデックスの比較」
セクション16.8.2.1「CONNECTION を使用した FEDERATED テーブルの作成」
セクション17.5.1.6「CREATE ... IF NOT EXISTS ステートメントのレプリケーション」
セクション13.1.13「CREATE EVENT ステートメント」
セクション13.1.15「CREATE INDEX ステートメント」
セクション13.1.17「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」
セクション13.1.20.4「CREATE TABLE ... SELECT ステートメント」
セクション13.1.20.8「CREATE TABLE および生成されるカラム」
セクション13.1.20「CREATE TABLE ステートメント」
セクション13.1.20.2「CREATE TEMPORARY TABLE ステートメント」
セクション13.1.23「CREATE VIEW ステートメント」
セクションB.3.4.2「DATE カラムの使用に関する問題」
セクション13.2.2「DELETE ステートメント」
セクション5.1.12.3「DNS ルックアップとホストキャッシュ」
セクション13.2.3「DO ステートメント」
セクション11.3.5「ENUM 型」
セクション13.2.11.6「EXISTS または NOT EXISTS を使用したサブクエリー」
セクション8.2.2.3「EXISTS 戦略を使用したサブクエリーの最適化」
セクション8.8.1「EXPLAIN によるクエリーの最適化」
セクション13.8.2「EXPLAIN ステートメント」
セクション8.8.2「EXPLAIN 出力フォーマット」
セクション16.8.3「FEDERATED ストレージエンジンの注記とヒント」
セクション13.1.20.5「FOREIGN KEY の制約」
セクション13.7.1.6「GRANT ステートメント」
セクション13.2.4「HANDLER ステートメント」
セクション27.12.19.5「host_cache テーブル」
セクション26.8「INFORMATION_SCHEMA COLUMNS テーブル」
セクション26.14「INFORMATION_SCHEMA EVENTS テーブル」
セクション26.51.29「INFORMATION_SCHEMA INNODB_TRX テーブル」
セクション26.23「INFORMATION_SCHEMA PROCESSLIST テーブル」
セクション26.48「INFORMATION_SCHEMA VIEWS テーブル」
セクション15.6.1.6「InnoDB での AUTO_INCREMENT 処理」
セクション15.7.3「InnoDB のさまざまな SQL ステートメントで設定されたロック」
セクション15.21.2「InnoDB のリカバリの強制的な実行」
セクション8.5.3「InnoDB の読み取り専用トランザクションの最適化」
セクション15.15.2.3「InnoDB トランザクションおよびロック情報の永続性と一貫性」
セクション15.15.2.1「InnoDB トランザクションの使用および情報のロック」
セクション8.5.2「InnoDB トランザクション管理の最適化」
セクション13.2.6.2「INSERT ... ON DUPLICATE KEY UPDATE ステートメント」
セクション13.2.6.1「INSERT ... SELECT ステートメント」
セクション13.2.6「INSERT ステートメント」
セクション13.2.10.2「JOIN 句」
セクション11.5「JSON データ型」
セクション12.18.3「JSON 値を検索する関数」
セクション13.7.8.4「KILL ステートメント」
セクション13.2.8「LOAD XML ステートメント」
セクション13.3.6「LOCK TABLES および UNLOCK TABLES ステートメント」
セクション16.7「MERGE ストレージエンジン」
セクション16.7.2「MERGE テーブルの問題点」
セクション15.6.1.5「MyISAM から InnoDB へのテーブルの変換」
セクション8.6.1「MyISAM クエリーの最適化」
セクション8.6.2「MyISAM テーブルの一括データロード」
セクション7.6.4「MyISAM テーブルの最適化」
セクションA.11「MySQL 8.0 FAQ: MySQL の中国語、日本語、および韓国語の文字セット」
セクションA.4「MySQL 8.0 FAQ: ストアドプロシージャおよびストアドファンクション」
セクションA.14「MySQL 8.0 FAQ: レプリケーション」
セクション1.3「MySQL 8.0 の新機能」
セクション6.6.2「MySQL Enterprise Encryption の使用方法と例」
セクション6.4.7.4「MySQL Enterprise Firewall リファレンス」

セクションB.3.3.5 「MySQL が一時ファイルを格納する場所」
セクション12.20.3 「MySQL での GROUP BY の処理」
セクション8.4.4 「MySQL での内部一時テーブルの使用」
セクション6.2.2 「MySQL で提供される権限」
セクション1.2.2 「MySQL の主な機能」
セクションB.3.7 「MySQL の既知の問題」
セクション4.5.1.6 「mysql クライアントのヒント」
セクション4.5.1.1 「mysql クライアントオプション」
セクション5.3 「mysql システムスキーマ」
セクション5.4.4.4 「mysql データベーステーブルへの変更に対するログイン形式」
第27章 「MySQL パフォーマンススキーマ」
セクション24.2.7 「MySQL パーティショニングによる NULL の扱い」
MySQL 用語集
セクション5.9.1.6 「mysqld でのエラーの原因を見つけるためのサーバーログの使用」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.6 「mysqldump — データベースバックアッププログラム」
セクション4.5.8 「mysqldump — ロードエミュレーションクライアント」
セクション23.1.7.3 「NDB Cluster でのトランザクション処理に関する制限」
セクション23.1.4 「NDB Cluster の新機能」
NDB Cluster システム変数
NDB Cluster ステータス変数
セクション23.5.10.1 「NDB Cluster ディスクデータオブジェクト」
セクション23.6.11 「NDB Cluster レプリケーションの競合解決」
セクション23.6.4 「NDB Cluster レプリケーションスキーマおよびテーブル」
セクション23.4.24 「ndb_select_all — NDB テーブルの行の出力」
セクション23.5.14.38 「ndbinfo nodes テーブル」
セクション23.5.14 「ndbinfo: NDB Cluster 情報データベース」
セクション13.5.1 「PREPARE ステートメント」
セクション27.12.19.9 「processlist テーブル」
セクション24.2.3.1 「RANGE COLUMNS パーティショニング」
セクション24.3.1 「RANGE および LIST パーティショニングの管理」
セクション8.2.1.2 「range の最適化」
セクション13.2.9 「REPLACE ステートメント」
セクション13.2.10.1 「SELECT ... INTO ステートメント」
セクション13.2.10 「SELECT ステートメント」
セクション8.2.1 「SELECT ステートメントの最適化」
セクション13.7.7.2 「SHOW BINLOG EVENTS ステートメント」
セクション13.7.7.13 「SHOW CREATE VIEW ステートメント」
セクション13.7.7.17 「SHOW ERRORS ステートメント」
セクション13.7.7.29 「SHOW PROCESSLIST ステートメント」
セクション13.7.7.32 「SHOW RELAYLOG EVENTS ステートメント」
セクション13.7.7.41 「SHOW VARIABLES ステートメント」
セクション13.7.7.42 「SHOW WARNINGS ステートメント」
セクション13.7.7 「SHOW ステートメント」
セクション13.3.1 「START TRANSACTION、COMMIT および ROLLBACK ステートメント」
セクション13.2.12 「TABLE ステートメント」
セクション13.2.10.3 「UNION 句」
セクション13.2.13 「UPDATE ステートメント」
セクション8.2.5.2 「UPDATE ステートメントの最適化」
セクション13.2.14 「VALUES ステートメント」
セクション8.2.1.1 「WHERE 句の最適化」
セクション13.2.15 「WITH (共通テーブル式)」
セクション1.1 「このマニュアルについて」
セクション3.3.4.1 「すべてのデータの選択」
セクション4.6.4.4 「その他の myisamchk オプション」
セクション26.1 「はじめに」
セクション6.2 「アクセス制御とアカウント管理」
セクション25.4.2 「イベントスケジューラの構成」
セクション8.9.4 「インデックスヒント」
セクション8.9.3 「オプティマイザヒント」

セクションB.3.5「オペティマイザ関連の問題」
セクション15.12.2「オンライン DDL のパフォーマンスと同時実行性」
セクション18.4.1「オンライングループの構成」
セクション13.2.10.4「カッコで囲まれたクエリー式」
セクション13.6.6.2「カーソル DECLARE ステートメント」
セクション13.6.6.3「カーソル FETCH ステートメント」
セクション9.3「キーワードと予約語」
セクション23.5.16「クイックリファレンス: NDB Cluster SQL ステートメント」
セクション3.2「クエリーの入力」
セクション4.2.2.1「コマンド行でのオプションの使用」
セクション13.2.11「サブクエリー」
セクション5.1.11「サーバー SQL モード」
セクション5.1.8「サーバーシステム変数」
セクション5.1.10「サーバーステータス変数」
セクション13.2.11.1「スカラーオペランドとしてのサブクエリー」
セクション25.8「ストアードプログラムの制約」
セクション25.7「ストアードプログラムバイナリロギング」
セクション25.2.1「ストアードルーチンの構文」
セクション27.12.19.10「スレッドテーブル」
セクション13.1.20.9「セカンダリインデックスと生成されたカラム」
セクション3.3.4「テーブルからの情報の取り出し」
セクション23.2.5「テーブルとデータを含む NDB Cluster の例」
セクション8.11.2「テーブルロックの問題」
セクション15.7.5.3「デッドロックを最小化および処理する方法」
セクション3.3.1「データベースの作成と選択」
セクション18.4.2.2「トランザクション一貫性保証の構成」
セクション15.7.2.1「トランザクション分離レベル」
セクション25.3.1「トリガーの構文と例」
セクション5.4.4「バイナリログ」
セクション5.6.6.4「バージョントークン参照」
セクション27.12.14.1「パフォーマンススキーマ persisted_variables テーブル」
セクション27.6「パフォーマンススキーマインストールメント命名規則」
セクション27.12.11「パフォーマンススキーマレプリケーションテーブル」
セクション15.8.9「パーティション構成」
セクション24.3.5「パーティションに関する情報を取得する」
セクション24.4「パーティションプルーニング」
セクション24.5「パーティション選択」
セクション25.5.1「ビューの構文」
セクション15.7.4「ファントム行」
セクション8.10.3「プリペアドステートメントおよびストアードプログラムのキャッシュ」
セクション8.3.6「マルチカラムインデックス」
セクション8.2.2.4「マージまたは実体化を使用した導出テーブル、ビュー参照および共通テーブル式の最適化」
セクション10.2.2「メタデータ用の UTF-8」
セクション9.4「ユーザー定義変数」
セクション13.2.11.9「ラテラル導出テーブル」
セクション5.6.4「リライタクエリーリライトプラグイン」
セクション5.6.4.2「リライタクエリーリライトプラグインの使用」
セクション6.4.5.9「レガシーモード監査ログのフィルタリング」
セクション17.2「レプリケーションの実装」
セクション17.1.6.2「レプリケーションソースのオプションと変数」
セクション13.6.4.2「ローカル変数のスコープと解決」
セクションB.3.4.7「一致する行がない場合の問題の解決」
セクション8.14.3「一般的なスレッドの状態」
セクション15.7.2.3「一貫性非ロック読み取り」
セクション6.2.3「付与テーブル」
セクション8.11.1「内部ロック方法」
セクション12.4.4「割り当て演算子」
セクション8.4.3.2「同じデータベースに大量のテーブルを作成することの短所」
セクション8.11.3「同時挿入」
セクション8.8.4「名前付き接続の実行計画情報の取得」

[セクション17.5.1.16「呼び出される機能のレプリケーション」](#)
[セクション13.7.6.1「変数代入の SET 構文」](#)
[セクション13.2.11.8「導出テーブル」](#)
[セクション12.3「式評価での型変換」](#)
[セクション12.16「情報関数」](#)
[セクション8.8.3「拡張 EXPLAIN 出力形式」](#)
[セクション10.4「接続文字セットおよび照合順序」](#)
[セクション6.1.3「攻撃者に対する MySQL のセキュアな状態の維持」](#)
[セクション9.1.1「文字列リテラル」](#)
[セクション8.3「最適化とインデックス」](#)
[セクション1.7.1「標準 SQL に対する MySQL 拡張機能」](#)
[セクション12.4.2「比較関数と演算子」](#)
[セクション5.1.9.3「永続化されるシステム変数」](#)
[セクションB.3.4.8「浮動小数点値に関する問題」](#)
[セクション8.2.2.1「準結合変換による IN および EXISTS サブクエリ述語の最適化」](#)
[セクション10.8.6「照合順序の効果の例」](#)
[セクション3.3.4.2「特定の行の選択」](#)
[セクション6.4.5.4「監査ログファイル形式」](#)
[セクション6.4.5.8「監査ログフィルタ定義の書き込み」](#)
[セクション11.4.11「空間インデックスの使用」](#)
[セクション15.7.2.2「自動コミット、コミットおよびロールバック」](#)
[セクション12.10.1「自然言語全文検索」](#)
[セクション15.7.2.4「読取りのロック」](#)
[セクション1.6「質問またはバグをレポートする方法」](#)
第12章「関数と演算子」
[セクション8.2.1.20「関数コールの最適化」](#)
[セクション12.20.1「集計関数の説明」](#)
[セクション13.1.20.10「非表示カラム」](#)

SELECT *

[セクション11.3.4「BLOB 型と TEXT 型」](#)
[セクション13.2.12「TABLE ステートメント」](#)

SELECT * FROM

[セクション23.5.14.37「ndbinfo memory_per_fragment テーブル」](#)

SELECT * FROM t PARTITION ()

[セクション24.1「MySQL のパーティショニングの概要」](#)

SELECT * INTO OUTFILE 'file_name' FROM tbl_name

[セクション7.2「データベースバックアップ方法」](#)

SELECT ... FOR SHARE

[セクション15.7.3「InnoDB のさまざまな SQL ステートメントで設定されたロック」](#)
[セクション15.7.1「InnoDB ロック」](#)
[セクション1.3「MySQL 8.0 の新機能」](#)
[セクション17.2.1.1「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」](#)
[セクション15.7.2.1「トランザクション分離レベル」](#)
[セクション6.2.3「付与テーブル」](#)
[セクション15.7.2.4「読取りのロック」](#)

SELECT ... FOR UPDATE

[セクション15.7.3「InnoDB のさまざまな SQL ステートメントで設定されたロック」](#)
[セクション15.7.5「InnoDB のデッドロック」](#)
[セクション15.1.2「InnoDB テーブルのベストプラクティス」](#)
[セクション15.7.1「InnoDB ロック」](#)
[セクション17.2.1.1「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」](#)

セクション15.7.5.3「デッドロックを最小化および処理する方法」
セクション15.7.2.4「読取りのロック」

SELECT ... FROM

セクション15.7.3「InnoDB のさまざまな SQL ステートメントで設定されたロック」

SELECT ... INTO

セクション13.1.13「CREATE EVENT ステートメント」
セクション13.2.10.1「SELECT ... INTO ステートメント」
セクション1.7.2.1「SELECT INTO TABLE の違い」
セクション13.2.10「SELECT ステートメント」
セクション17.5.1.14「レプリケーションとシステム関数」
セクション13.6.4.2「ローカル変数のスコープと解決」

SELECT ... INTO DUMPFILE

セクション5.1.8「サーバースystem変数」
セクション6.1.3「攻撃者に対する MySQL のセキュアな状態の維持」

SELECT ... INTO outfile

セクション15.21.2「InnoDB のリカバリの強制的な実行」
セクション13.2.7「LOAD DATA ステートメント」
セクション1.3「MySQL 8.0 の新機能」
セクション6.2.2「MySQL で提供される権限」
セクション6.2.21「MySQL への接続の問題のトラブルシューティング」
セクション7.4.3「mysqldump による区切りテキストフォーマットでのデータのダンプ」
セクション9.1.7「NULL 値」
セクション13.2.10.1「SELECT ... INTO ステートメント」
セクション1.7.2.1「SELECT INTO TABLE の違い」
セクション2.3.7「Windows プラットフォームの制限事項」
セクション1.1「このマニュアルについて」
セクション5.1.7「サーバーコマンドオプション」
セクション5.1.8「サーバースystem変数」
セクション7.1「バックアップとリカバリの種類」
セクション6.1.3「攻撃者に対する MySQL のセキュアな状態の維持」

SELECT ... INTO outfile 'file_name'

セクション13.2.10.1「SELECT ... INTO ステートメント」

SELECT ... INTO var_list

セクション25.8「ストアプログラムの制約」
セクション13.6.4「ストアプログラム内の変数」

SELECT ... LOCK IN SHARE MODE

セクション23.1.7.3「NDB Cluster でのトランザクション処理に関する制限」

SELECT DISTINCT

InnoDB オプティマイザ統計でサンプリングされるページの数の構成
セクション8.14.3「一般的なスレッドの状態」
セクション8.2.2.1「準結合変換による IN および EXISTS サブクエリー述語の最適化」

SELECT FROM table

セクション23.1.4「NDB Cluster の新機能」

SELECT INTO ... outfile

セクション23.4.13「ndb_import — NDB への CSV データのインポート」

SELECT INTO DUMPFILE

セクション1.3「MySQL 8.0 の新機能」
セクション5.1.8「サーバーシステム変数」

SELECT INTO OUTFILE

セクション23.4.13「ndb_import — NDB への CSV データのインポート」
セクション5.1.8「サーバーシステム変数」

SELECT SLEEP()

セクション5.1.11「サーバー SQL モード」

SET

セクション6.4.4.6「HashiCorp Vault キーリングプラグインの使用」
セクション15.8.7「InnoDB I/O Capacity の構成」
セクション15.14「InnoDB の起動オプションおよびシステム変数」
セクション15.8.3.1「InnoDB バッファプールサイズの構成」
セクション15.13「InnoDB 保存データ暗号化」
セクション1.3「MySQL 8.0 の新機能」
セクション6.6.2「MySQL Enterprise Encryption の使用方法と例」
セクション13.6.7.7「MySQL の診断領域」
セクション4.5.1.6「mysql クライアントのヒント」
セクション23.6.11「NDB Cluster レプリケーションの競合解決」
セクション17.1.6.3「Replica Server のオプションと変数」
セクション13.4.2.5「RESET REPLICAS | SLAVE ステートメント」
セクション13.3.7「SET TRANSACTION ステートメント」
セクション13.7.6「SET ステートメント」
セクション13.7.7.41「SHOW VARIABLES ステートメント」
セクション12.1「SQL 関数および演算子リファレンス」
セクション15.6.3.4「undo テーブルスペース」
セクション25.4.2「イベントスケジューラの構成」
セクション8.9.3「オプティマイザヒント」
セクション5.6.7.12「クローンシステム変数」
セクション17.1.6.5「グローバルトランザクション ID システム変数」
セクション15.8.3.7「コアファイルからのバッファプールページの除外」
セクション4.2.2.1「コマンド行でのオプションの使用」
セクション13.2.11「サブクエリー」
セクション5.1.11「サーバー SQL モード」
セクション5.1.7「サーバーコマンドオプション」
セクション5.1.8「サーバーシステム変数」
セクション5.1.9「システム変数の使用」
セクション5.1.9.1「システム変数権限」
セクション25.1「ストアードプログラムの定義」
セクション25.7「ストアードプログラムバイナリロギング」
セクション13.6.4「ストアードプログラム内の変数」
セクション5.4.5「スロークエリーログ」
セクション14.1「データディクショナリスキーマ」
セクション18.4.2.2「トランザクション一貫性保証の構成」
セクション25.3.1「トリガーの構文と例」
セクション17.1.6.4「バイナリロギングのオプションと変数」
セクション4.2.2.5「プログラム変数の設定へのオプションの使用」
セクション9.4「ユーザー定義変数」
セクション17.1.6.2「レプリケーションソースのオプションと変数」
セクション12.4.4「割り当て演算子」
セクション15.9.1.2「圧縮テーブルの作成」
セクション13.7.6.1「変数代入の SET 構文」
セクション12.16「情報関数」
セクション10.4「接続文字セットおよび照合順序」
セクション6.2.16「期限切れパスワードのサーバー処理」

セクション1.7.1 「標準 SQL に対する MySQL 拡張機能」
セクション5.1.9.3 「永続化されるシステム変数」
セクション12.4 「演算子」
セクション6.4.5.10 「監査ログ参照」
第12章 「関数と演算子」
セクション15.8.10.2 「非永続的オブティマイザ統計のパラメータの構成」

SET @@GLOBAL.gtid_purged

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

SET @@GLOBAL.ndb_slave_conflict_role = 'NONE'

NDB Cluster システム変数

SET @x=2, @y=4, @z=8

セクション13.2.10.1 「SELECT ... INTO ステートメント」

SET autocommit

セクション8.5.5 「InnoDB テーブルの一括データロード」
セクション13.3 「トランザクションステートメントおよびロックステートメント」

SET autocommit = 0

セクション17.4.10 「準同期レプリケーション」

SET CHARACTER SET

セクション13.7.6.2 「SET CHARACTER SET ステートメント」
セクション13.7.6 「SET ステートメント」
セクション10.9 「Unicode のサポート」
セクション10.4 「接続文字セットおよび照合順序」

SET CHARACTER SET 'charset_name'

セクション10.4 「接続文字セットおよび照合順序」

SET CHARACTER SET charset_name

セクション10.4 「接続文字セットおよび照合順序」

SET DEFAULT ROLE

セクション13.7.1.1 「ALTER USER ステートメント」
セクション13.7.1.9 「SET DEFAULT ROLE ステートメント」
セクション13.7.1.11 「SET ROLE ステートメント」
セクション13.7.6 「SET ステートメント」
セクション5.1.8 「サーバーシステム変数」
セクション6.2.10 「ロールの使用」

SET GLOBAL

セクション15.6.3.2 「File-Per-Table テーブルスペース」
セクション13.7.1.6 「GRANT ステートメント」
セクション15.8.7 「InnoDB I/O Capacity の構成」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.8.3.4 「InnoDB バッファープールのプリフェッチ (先読み) の構成」
セクション6.4.4.5 「keyring_aws Amazon Web Services キーリングプラグインの使用」
セクション6.2.2 「MySQL で提供される権限」
セクション5.1.9.1 「システム変数権限」
セクション15.8.8 「スピンロックのポーリングの構成」
セクション15.8.3.3 「バッファープールをスキャンに耐えられるようにする」
セクション5.4.2.6 「ルールベースのエラーログのフィルタリング (log_filter_dragnet)」

セクション13.7.6.1「変数代入の SET 構文」
セクション15.5.2「変更バッファ」
セクション6.3.1「暗号化接続を使用するための MySQL の構成」
セクション5.1.9.3「永続化されるシステム変数」
セクション17.4.10.2「準同期レプリケーションのインストールと構成」
セクション8.10.2.2「複合キーキャッシュ」

SET NAMES

セクション10.10.7.2「gb18030 文字セット」
セクション13.2.7「LOAD DATA ステートメント」
セクションA.11「MySQL 8.0 FAQ: MySQL の中国語、日本語、および韓国語の文字セット」
セクション4.5.1.2「mysql クライアントコマンド」
セクション4.5.4「mysqldump — データベースバックアッププログラム」
セクション4.5.6「mysqlpump — データベースバックアッププログラム」
セクション13.7.6.3「SET NAMES ステートメント」
セクション13.7.6「SET ステートメント」
セクション10.9「Unicode のサポート」
セクション10.5「アプリケーションの文字セットおよび照合順序の構成」
セクション10.6「エラーメッセージ文字セット」
セクション5.1.8「サーバーシステム変数」
セクション10.2.2「メタデータ用の UTF-8」
セクション12.3「式評価での型変換」
セクション10.4「接続文字セットおよび照合順序」
セクション10.3.6「文字列リテラルの文字セットおよび照合順序」

SET NAMES 'charset_name'

セクション10.4「接続文字セットおよび照合順序」

SET NAMES 'cp1251'

セクション10.4「接続文字セットおよび照合順序」

SET NAMES charset_name

セクション4.6.8「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」

SET NAMES default_character_set

セクション4.5.4「mysqldump — データベースバックアッププログラム」
セクション4.5.6「mysqlpump — データベースバックアッププログラム」

SET PASSWORD

セクション17.5.1.8「CURRENT_USER() のレプリケーション」
セクション6.2.2「MySQL で提供される権限」
セクション5.4.4.4「mysql データベーステーブルへの変更に対するログイン形式」
root のパスワードのリセット: 一般的な手順
セクション13.7.1.10「SET PASSWORD ステートメント」
セクション13.7.6「SET ステートメント」
セクション6.2.4「アカウント名の指定」
セクション5.1.8「サーバーシステム変数」
セクション6.1.2.3「パスワードおよびログイン」
セクション6.4.3.2「パスワード検証オプションおよび変数」
セクション6.4.3「パスワード検証コンポーネント」
セクション6.2.15「パスワード管理」
セクション17.5.1.39「レプリケーションと変数」
セクション6.2.3「付与テーブル」
セクション12.16「情報関数」
セクション13.3.3「暗黙的なコミットを発生させるステートメント」
セクション6.2.16「期限切れパスワードのサーバー処理」
セクション6.2.13「権限変更が有効化される時期」

SET PASSWORD ... = PASSWORD()

セクション1.3「MySQL 8.0の新機能」

SET PERSIST

セクション6.4.2.1「Connection-Control プラグインのインストール」
セクション6.4.1.7「LDAP プラガブル認証」
セクション6.4.7.3「MySQL Enterprise Firewall の使用」
セクション6.2.2「MySQL で提供される権限」
セクション5.4.2.1「エラーログ構成」
セクション4.2.2.2「オプションファイルの使用」
セクション5.1.8「サーバーシステム変数」
セクション5.1.9.1「システム変数権限」
セクション6.2.15「パスワード管理」
セクション27.12.14.1「パフォーマンススキーマ persisted_variables テーブル」
セクション5.4.2.6「ルールベースのエラーログのフィルタリング (log_filter_dragnet)」
セクション6.2.10「ロールの使用」
セクション13.7.6.1「変数代入の SET 構文」
セクション4.2.8「接続圧縮制御」
セクション6.3.1「暗号化接続を使用するための MySQL の構成」
セクション5.1.9.3「永続化されるシステム変数」
セクション5.1.9.4「永続的で永続的に制限されないシステム変数」
セクション6.2.12「部分取消しを使用した権限の制限」

SET PERSIST_ONLY

セクション6.2.2「MySQL で提供される権限」
セクション6.4.4.7「Oracle Cloud Infrastructure Vault キーリングプラグインの使用」
セクション13.7.8.8「RESTART ステートメント」
セクション4.2.2.2「オプションファイルの使用」
セクション5.1.8「サーバーシステム変数」
セクション5.1.9.1「システム変数権限」
セクション5.1.9.3「永続化されるシステム変数」
セクション5.1.9.4「永続的で永続的に制限されないシステム変数」

SET RESOURCE GROUP

セクション6.2.2「MySQL で提供される権限」
セクション13.7.2.4「SET RESOURCE GROUP ステートメント」
セクション8.9.3「オプティマイザヒント」
セクション5.1.16「リソースグループ」

SET ROLE

セクション13.7.1.11「SET ROLE ステートメント」
セクション13.7.6「SET ステートメント」
セクション13.7.7.21「SHOW GRANTS ステートメント」
セクション5.1.8「サーバーシステム変数」
セクション6.2.10「ロールの使用」
セクション12.16「情報関数」
セクション6.2.13「権限変更が有効化される時期」

SET ROLE DEFAULT

セクション13.7.1.1「ALTER USER ステートメント」
セクション13.7.1.3「CREATE USER ステートメント」
セクション5.3「mysql システムスキーマ」
セクション13.7.1.9「SET DEFAULT ROLE ステートメント」
セクション13.7.1.11「SET ROLE ステートメント」

SET SESSION

セクション5.1.9.1「システム変数権限」

SET SESSION TRANSACTION ISOLATION LEVEL

セクション5.1.8「サーバーシステム変数」

SET SESSION TRANSACTION {READ WRITE | READ ONLY}

セクション5.1.8「サーバーシステム変数」

SET sql_log_bin = 0

セクション4.6.8「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」

SET sql_log_bin=OFF

セクション5.4.4「バイナリログ」

SET sql_mode='modes'

セクションA.3「MySQL 8.0 FAQ: サーバー SQL モード」

SET TIMESTAMP = value

セクション8.14.1「プロセスリストへのアクセス」

SET TRANSACTION

セクション13.3.7「SET TRANSACTION ステートメント」

セクション13.3.1「START TRANSACTION、COMMIT および ROLLBACK ステートメント」

セクション5.1.18「クライアントセッション状態の変更のサーバートラッキング」

セクション5.1.7「サーバーコマンドオプション」

セクション5.1.8「サーバーシステム変数」

セクション15.7.2.1「トランザクション分離レベル」

SET TRANSACTION ISOLATION LEVEL

セクション13.7.6「SET ステートメント」

セクション13.3.1「START TRANSACTION、COMMIT および ROLLBACK ステートメント」

セクション5.1.8「サーバーシステム変数」

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED

セクション15.20.7「InnoDB memcached プラグインとレプリケーション」

SET TRANSACTION {READ WRITE | READ ONLY}

セクション5.1.8「サーバーシステム変数」

SET var_name = value

セクション13.7.6「SET ステートメント」

SHOW

セクション13.1.13「CREATE EVENT ステートメント」

セクション13.1.17「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」

セクション14.5「INFORMATION_SCHEMA とデータディクショナリの統合」

セクションA.14「MySQL 8.0 FAQ: レプリケーション」

セクション1.2.2「MySQL の主な機能」

セクション4.5.7「mysqlshow — データベース、テーブル、およびカラム情報の表示」

セクション13.7.7.5「SHOW COLUMNS ステートメント」

セクション13.7.7.39「SHOW TABLES ステートメント」

セクション13.7.7「SHOW ステートメント」

セクション26.55「SHOW ステートメントの拡張」

セクション28.2「sys スキーマの使用」

セクション26.1「はじめに」

セクション13.6.6.2「カーソル DECLARE ステートメント」
セクション25.8「ストアプログラムの制約」
セクション13.4.1「ソースサーバーを制御する SQL ステートメント」
セクション14.1「データディクショナリスキーマ」
セクション3.3「データベースの作成と使用」
セクション18.4.2.2「トランザクション一貫性保証の構成」
セクション5.4.4「バイナリログ」
セクション27.1「パフォーマンススキーマクイックスタート」
セクション10.2.2「メタデータ用の UTF-8」
セクション17.1.2.8「レプリケーション環境へのレプリカの追加」
既存のデータによるレプリケーションのセットアップ
セクション1.7.1「標準 SQL に対する MySQL 拡張機能」

SHOW BINARY LOGS

セクション6.2.2「MySQL で提供される権限」
セクション4.6.8「mysqlbinlog — バイナリログファイル进行处理するためのユーティリティ」
セクション13.4.1.1「PURGE BINARY LOGS ステートメント」
セクション13.7.7.1「SHOW BINARY LOGS ステートメント」
セクション13.4.1「ソースサーバーを制御する SQL ステートメント」
セクション17.3.2「バイナリログファイルとリレーログファイルの暗号化」
セクション4.6.8.3「バイナリログファイルのバックアップのための mysqlbinlog の使用」

SHOW BINLOG EVENTS

GTID のあるトランザクションのスキップ
セクション6.2.2「MySQL で提供される権限」
セクション23.1.4「NDB Cluster の新機能」
セクション23.6.4「NDB Cluster レプリケーションスキーマおよびテーブル」
セクション13.7.7.2「SHOW BINLOG EVENTS ステートメント」
セクション13.4.2.7「START REPLICA | SLAVE ステートメント」
セクション13.6.6.5「サーバー側のカーソルの制約」
セクション13.4.1「ソースサーバーを制御する SQL ステートメント」
セクション17.1.7.3「トランザクションのスキップ」
セクション5.4.4.5「バイナリログトランザクション圧縮」
バイナリログトランザクション圧縮が有効な場合の動作

SHOW CHARACTER SET

セクション13.1.2「ALTER DATABASE ステートメント」
セクション13.1.12「CREATE DATABASE ステートメント」
セクション26.4「INFORMATION_SCHEMA CHARACTER_SETS テーブル」
セクション10.2「MySQL での文字セットと照合順序」
セクション13.7.7.3「SHOW CHARACTER SET ステートメント」
セクション26.55「SHOW ステートメントの拡張」
セクション10.3.5「カラム文字セットおよび照合順序」
セクション10.10「サポートされる文字セットおよび照合順序」
セクション5.1.8「サーバーシステム変数」
セクション10.3.4「テーブル文字セットおよび照合順序」
セクション10.3.3「データベース文字セットおよび照合順序」
セクション10.3.8「文字セットイントロデューサ」
セクション10.3.6「文字列リテラルの文字セットおよび照合順序」

SHOW COLLATION

セクション13.1.2「ALTER DATABASE ステートメント」
セクション13.1.12「CREATE DATABASE ステートメント」
セクション26.7「INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY テーブル」
セクション26.6「INFORMATION_SCHEMA COLLATIONS テーブル」
セクション10.2「MySQL での文字セットと照合順序」
セクション2.9.7「MySQL ソース構成オプション」
セクション13.7.7.4「SHOW COLLATION ステートメント」

[セクション5.1.8「サーバーシステム変数」](#)
[セクション10.15「文字セットの構成」](#)
[セクション10.14.2「照合順序 ID の選択」](#)

SHOW COLUMNS

[セクション13.8.2「EXPLAIN ステートメント」](#)
[セクション26.8「INFORMATION_SCHEMA COLUMNS テーブル」](#)
[セクション26.51.1「INFORMATION_SCHEMA INNODB_BUFFER_PAGE テーブル」](#)
[セクション26.51.2「INFORMATION_SCHEMA INNODB_BUFFER_PAGE_LRU テーブル」](#)
[セクション26.51.3「INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS テーブル」](#)
[セクション26.51.4「INFORMATION_SCHEMA INNODB_CACHED_INDEXES テーブル」](#)
[セクション26.51.5「INFORMATION_SCHEMA INNODB_CMP および INNODB_CMP_RESET テーブル」](#)
[セクション26.51.7「INFORMATION_SCHEMA INNODB_CMP_PER_INDEX および INNODB_CMP_PER_INDEX_RESET テーブル」](#)
[セクション26.51.6「INFORMATION_SCHEMA INNODB_CMPMEM および INNODB_CMPMEM_RESET テーブル」](#)
[セクション26.51.8「INFORMATION_SCHEMA INNODB_COLUMNS テーブル」](#)
[セクション26.51.9「INFORMATION_SCHEMA INNODB_DATAFILES テーブル」](#)
[セクション26.51.10「INFORMATION_SCHEMA INNODB_FIELDS テーブル」](#)
[セクション26.51.11「INFORMATION_SCHEMA INNODB_FOREIGN テーブル」](#)
[セクション26.51.12「INFORMATION_SCHEMA INNODB_FOREIGN_COLS テーブル」](#)
[セクション26.51.13「INFORMATION_SCHEMA INNODB_FT_BEING_DELETED テーブル」](#)
[セクション26.51.14「INFORMATION_SCHEMA INNODB_FT_CONFIG テーブル」](#)
[セクション26.51.15「INFORMATION_SCHEMA INNODB_FT_DEFAULT_STOPWORD テーブル」](#)
[セクション26.51.16「INFORMATION_SCHEMA INNODB_FT_DELETED テーブル」](#)
[セクション26.51.17「INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE テーブル」](#)
[セクション26.51.18「INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE テーブル」](#)
[セクション26.51.19「INFORMATION_SCHEMA INNODB_INDEXES テーブル」](#)
[セクション26.51.22「INFORMATION_SCHEMA INNODB_METRICS テーブル」](#)
[セクション26.51.23「INFORMATION_SCHEMA INNODB_SESSION_TEMP_TABLESPACES テーブル」](#)
[セクション26.51.24「INFORMATION_SCHEMA INNODB_TABLES テーブル」](#)
[セクション26.51.25「INFORMATION_SCHEMA INNODB_TABLESPACES テーブル」](#)
[セクション26.51.26「INFORMATION_SCHEMA INNODB_TABLESPACES_BRIEF テーブル」](#)
[セクション26.51.27「INFORMATION_SCHEMA INNODB_TABLESTATS ビュー」](#)
[セクション26.51.28「INFORMATION_SCHEMA INNODB_TEMP_TABLE_INFO テーブル」](#)
[セクション26.51.29「INFORMATION_SCHEMA INNODB_TRX テーブル」](#)
[セクション26.51.30「INFORMATION_SCHEMA INNODB_VIRTUAL テーブル」](#)
[セクション8.4.4「MySQL での内部一時テーブルの使用」](#)
[セクション13.7.7.5「SHOW COLUMNS ステートメント」](#)
[セクション26.55「SHOW ステートメントの拡張」](#)
[セクション27.1「パフォーマンススキーマクイックスタート」](#)
[セクション13.1.20.10「非表示カラム」](#)

SHOW COLUMNS FROM tbl_name LIKE 'enum_col'

[セクション11.3.5「ENUM 型」](#)

SHOW COUNT()

[セクション13.7.7.17「SHOW ERRORS ステートメント」](#)
[セクション13.7.7.42「SHOW WARNINGS ステートメント」](#)

SHOW CREATE DATABASE

[セクション13.1.2「ALTER DATABASE ステートメント」](#)
[セクション13.7.7.6「SHOW CREATE DATABASE ステートメント」](#)
[セクション5.1.8「サーバーシステム変数」](#)

SHOW CREATE EVENT

[セクション13.7.7.18「SHOW EVENTS ステートメント」](#)
[セクション25.4.6「イベントスケジューラと MySQL 権限」](#)
[セクション25.4.4「イベントメタデータ」](#)

SHOW CREATE FUNCTION

セクション13.1.17「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」
セクションA.4「MySQL 8.0 FAQ: ストアドプロシージャーおよびストアドファンクション」
セクション6.2.2「MySQL で提供される権限」
セクション13.7.7.9「SHOW CREATE PROCEDURE ステートメント」
セクション25.2.2「ストアドルーチンと MySQL 権限」
セクション25.2.3「ストアドルーチンのメタデータ」
セクション1.6「質問またはバグをレポートする方法」

SHOW CREATE PROCEDURE

セクション13.1.17「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」
セクションA.4「MySQL 8.0 FAQ: ストアドプロシージャーおよびストアドファンクション」
セクション6.2.2「MySQL で提供される権限」
セクション13.7.7.8「SHOW CREATE FUNCTION ステートメント」
セクション25.2.2「ストアドルーチンと MySQL 権限」
セクション25.2.3「ストアドルーチンのメタデータ」
セクション1.6「質問またはバグをレポートする方法」

SHOW CREATE SCHEMA

セクション15.13「InnoDB 保存データ暗号化」
セクション13.7.7.6「SHOW CREATE DATABASE ステートメント」

SHOW CREATE TABLE

セクション13.1.9「ALTER TABLE ステートメント」
セクション13.1.20「CREATE TABLE ステートメント」
セクション13.8.2「EXPLAIN ステートメント」
セクション16.8.2「FEDERATED テーブルの作成方法」
セクション13.1.20.5「FOREIGN KEY の制約」
セクション15.6.1.3「InnoDB テーブルのインポート」
セクション15.9.2「InnoDB ページ圧縮」
セクション24.2.5「KEY パーティショニング」
セクション7.6.3「MyISAM テーブルの修復方法」
セクション23.1.7.1「NDB Cluster の SQL 構文に準拠していません」
NDB Cluster システム変数
セクション23.5.10.1「NDB Cluster ディスクデータオブジェクト」
セクション23.4.9「ndb_desc — NDB テーブルの表示」
セクション23.4.23「ndb_restore — NDB Cluster バックアップの復元」
セクション13.1.20.11「NDB_TABLE オプションの設定」
セクション23.5.14.22「ndbinfo dict_obj_tree テーブル」
セクション24.3.1「RANGE および LIST パーティションの管理」
セクション13.7.7.5「SHOW COLUMNS ステートメント」
セクション13.7.7.10「SHOW CREATE TABLE ステートメント」
セクション15.8.11「インデックスページのマージしきい値の構成」
セクション5.1.8「サーバーシステム変数」
セクション2.11.13「テーブルまたはインデックスの再作成または修復」
セクション15.6.3.9「テーブルスペースの AUTOEXTEND_SIZE 構成」
セクション14.1「データディクショナリスキーマ」
セクション3.4「データベースとテーブルに関する情報の取得」
セクション11.6「データ型デフォルト値」
セクション27.1「パフォーマンススキーマクイックスタート」
セクション8.2.4「パフォーマンススキーマクエリーの最適化」
セクション24.3.5「パーティションに関する情報を取得する」
セクション15.9.1.2「圧縮テーブルの作成」
セクション3.6.6「外部キーの使用」
セクション13.1.20.7「暗黙のカラム指定の変更」

SHOW CREATE TRIGGER

セクション13.7.7.11「SHOW CREATE TRIGGER ステートメント」

セクション25.3.2「トリガーのメタデータ」

SHOW CREATE USER

セクション13.7.1.1「ALTER USER ステートメント」
セクション13.7.1.3「CREATE USER ステートメント」
セクション13.7.7.12「SHOW CREATE USER ステートメント」
セクション13.7.7.21「SHOW GRANTS ステートメント」
セクション6.2.8「アカウントの追加、権限の割当ておよびアカウントの削除」
セクション6.2.19「アカウントロック」
セクション5.1.8「サーバーシステム変数」
セクション6.2.3「付与テーブル」

SHOW CREATE VIEW

セクション13.7.1.6「GRANT ステートメント」
セクション26.48「INFORMATION_SCHEMA VIEWS テーブル」
セクション6.2.2「MySQL で提供される権限」
セクション13.7.7.13「SHOW CREATE VIEW ステートメント」
セクション2.11.5「アップグレード用のインストールの準備」
セクション25.5.5「ビューのメタデータ」
セクション25.9「ビューの制約」

SHOW DATABASES

セクション13.7.1.6「GRANT ステートメント」
セクション26.31「INFORMATION_SCHEMA SCHEMATA テーブル」
セクション14.5「INFORMATION_SCHEMA とデータディクショナリの統合」
セクション6.2.2「MySQL で提供される権限」
セクション23.5.17.2「NDB Cluster および MySQL の権限」
セクション23.5.14「ndbinfo: NDB Cluster 情報データベース」
セクション13.7.7.14「SHOW DATABASES ステートメント」
セクション26.55「SHOW ステートメントの拡張」
セクション26.1「はじめに」
セクション5.1.7「サーバーコマンドオプション」
セクション5.1.8「サーバーシステム変数」
セクション3.4「データベースとテーブルに関する情報の取得」
セクション3.3「データベースの作成と使用」
セクション6.2.3「付与テーブル」
セクション9.2.3「識別子の太文字と小文字の区別」

SHOW ENGINE

セクション6.2.2「MySQL で提供される権限」
セクション13.7.7.15「SHOW ENGINE ステートメント」

SHOW ENGINE INNODB MUTEX

セクション15.14「InnoDB の起動オプションおよびシステム変数」
セクション15.17.3「InnoDB 標準モニターおよびロックモニターの出力」
セクション13.7.7.15「SHOW ENGINE ステートメント」

SHOW ENGINE INNODB STATUS

セクション13.1.20.5「FOREIGN KEY の制約」
セクション26.51.3「INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS テーブル」
セクション15.15.3「InnoDB INFORMATION_SCHEMA スキーマオブジェクトテーブル」
セクション15.15.5「InnoDB INFORMATION_SCHEMA バッファプールテーブル」
セクション15.15.6「InnoDB INFORMATION_SCHEMA メトリックテーブル」
セクション15.7.5「InnoDB のデッドロック」
セクション8.5.3「InnoDB の読み取り専用トランザクションの最適化」
セクション15.14「InnoDB の起動オプションおよびシステム変数」
セクション15.6.1.4「InnoDB テーブルの移動またはコピー」
セクション8.5.8「InnoDB ディスク I/O の最適化」

セクション15.17.2 「InnoDB モニターの有効化」
セクション15.7.1 「InnoDB ロック」
セクション15.17.3 「InnoDB 標準モニターおよびロックモニターの出力」
セクション15.6.1.5 「MyISAM から InnoDB へのテーブルの変換」

MySQL 用語集

セクション13.7.7.15 「SHOW ENGINE ステートメント」
セクションB.2 「エラー情報インターフェース」
セクション15.7.5.3 「デッドロックを最小化および処理する方法」
セクション15.5.1 「バッファプール」
セクション15.8.9 「ページ構成」
セクション15.8.2 「読み取り専用操作の InnoDB の構成」
セクション15.5.3 「適応型ハッシュインデックス」

SHOW ENGINE NDB STATUS

セクション23.5 「NDB Cluster の管理」
セクション23.6.4 「NDB Cluster レプリケーションスキーマおよびテーブル」
セクション23.2.2.3 「Windows での NDB Cluster の初期起動」
セクション23.5.16 「クイックリファレンス: NDB Cluster SQL ステートメント」

SHOW ENGINE NDBCLUSTER STATUS

NDB Cluster の MySQL Server オプション
セクション23.5.16 「クイックリファレンス: NDB Cluster SQL ステートメント」

SHOW ENGINE PERFORMANCE_SCHEMA STATUS

セクション13.7.7.15 「SHOW ENGINE ステートメント」
セクション27.10 「パフォーマンススキーマのステートメントダイジェストとサンプリング」
セクション27.7 「パフォーマンススキーマステータスマonitoring」

SHOW ENGINES

セクション16.5 「ARCHIVE ストレージエンジン」
セクション16.6 「BLACKHOLE ストレージエンジン」
セクション26.13 「INFORMATION_SCHEMA ENGINES テーブル」
セクション15.1.3 「InnoDB がデフォルトのストレージエンジンであるかどうかの確認」
セクションA.10 「MySQL 8.0 FAQ: NDB Cluster」
セクション2.3.4.3 「MySQL サーバタイプの選択」
セクション23.5.9 「NDB Cluster での MySQL Server の使用」
セクション23.5.14 「ndbinfo: NDB Cluster 情報データベース」
セクション13.7.7.16 「SHOW ENGINES ステートメント」
セクション23.5.16 「クイックリファレンス: NDB Cluster SQL ステートメント」
セクション5.1.8 「サーバーシステム変数」
セクション27.1 「パフォーマンススキーマクイックスタート」
セクション27.2 「パフォーマンススキーマビルド構成」
第16章 「代替ストレージエンジン」

SHOW ERRORS

セクション13.6.7.3 「GET DIAGNOSTICS ステートメント」
セクション13.6.7.7 「MySQL の診断領域」
セクション13.6.7.4 「RESIGNAL ステートメント」
セクション13.7.7.17 「SHOW ERRORS ステートメント」
セクション13.7.7.42 「SHOW WARNINGS ステートメント」
セクション13.6.7.5 「SIGNAL ステートメント」
セクションB.2 「エラー情報インターフェース」
セクション5.1.8 「サーバーシステム変数」

SHOW EVENTS

セクション26.14 「INFORMATION_SCHEMA EVENTS テーブル」
セクション13.7.7.18 「SHOW EVENTS ステートメント」
セクション25.4.6 「イベントスケジューラと MySQL 権限」

[セクション25.4.4 「イベントメタデータ」](#)
[セクション17.5.1.16 「呼び出される機能のレプリケーション」](#)

SHOW FULL COLUMNS

[セクション13.1.20 「CREATE TABLE ステートメント」](#)
[セクション26.10 「INFORMATION_SCHEMA COLUMN_PRIVILEGES テーブル」](#)

SHOW FULL PROCESSLIST

[セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」](#)
[セクション15.12.2 「オンライン DDL のパフォーマンスと同時実行性」](#)
[セクション8.14.1 「プロセスリストへのアクセス」](#)

SHOW FULL TABLES

[セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」](#)

SHOW FUNCTION CODE

[セクション6.2.2 「MySQL で提供される権限」](#)
[セクション13.7.7.27 「SHOW PROCEDURE CODE ステートメント」](#)
[セクション25.2.2 「ストアドルーチンと MySQL 権限」](#)
[セクション25.2.3 「ストアドルーチンのメタデータ」](#)

SHOW FUNCTION STATUS

[セクション6.2.2 「MySQL で提供される権限」](#)
[セクション13.7.7.28 「SHOW PROCEDURE STATUS ステートメント」](#)
[セクション25.2.2 「ストアドルーチンと MySQL 権限」](#)
[セクション25.2.3 「ストアドルーチンのメタデータ」](#)

SHOW GLOBAL STATUS

[NDB Cluster ステータス変数](#)
[セクション5.1.8 「サーバーシステム変数」](#)

SHOW GRANTS

[セクション13.7.1.6 「GRANT ステートメント」](#)
[セクション6.2.2 「MySQL で提供される権限」](#)
[セクション6.2.21 「MySQL への接続の問題のトラブルシューティング」](#)
[セクション23.5.12 「NDB_STORED_USER での分散 MySQL 権限」](#)
[セクション13.7.1.8 「REVOKE ステートメント」](#)
[セクション13.7.7.12 「SHOW CREATE USER ステートメント」](#)
[セクション13.7.7.21 「SHOW GRANTS ステートメント」](#)
[セクション13.7.7.26 「SHOW PRIVILEGES ステートメント」](#)
[セクション6.2.8 「アカウントの追加、権限の割当ておよびアカウントの削除」](#)
[セクション6.2 「アクセス制御とアカウント管理」](#)
[セクション5.1.8 「サーバーシステム変数」](#)
[セクション6.1.1 「セキュリティガイドライン」](#)
[セクション6.2.10 「ロールの使用」](#)
[セクション6.2.3 「付与テーブル」](#)
[セクション6.2.12 「部分取消しを使用した権限の制限」](#)

SHOW GRANTS FOR CURRENT_USER

[セクション13.7.7.21 「SHOW GRANTS ステートメント」](#)

SHOW GRANTS FOR user

[セクション13.7.7.21 「SHOW GRANTS ステートメント」](#)

SHOW INDEX

[セクション13.7.3.1 「ANALYZE TABLE ステートメント」](#)

セクション13.8.2「EXPLAIN ステートメント」
セクション8.8.2「EXPLAIN 出力フォーマット」
セクション26.34「INFORMATION_SCHEMA STATISTICS テーブル」
セクション26.42「INFORMATION_SCHEMA TABLE_CONSTRAINTS テーブル」
セクション8.3.8「InnoDB および MyISAM インデックス統計コレクション」
セクション23.4.14「ndb_index_stat — NDB インデックス統計ユーティリティ」
セクション13.7.7.5「SHOW COLUMNS ステートメント」
セクション13.7.7.22「SHOW INDEX ステートメント」
セクション4.6.4.4「その他の myisamchk オプション」
セクション8.9.4「インデックスヒント」
セクション15.8.11「インデックスページのマージしきい値の構成」
セクション8.9.3「オプティマイザヒント」
セクション8.2.4「パフォーマンススキーマクエリーの最適化」
セクション8.3.12「不可視のインデックス」
セクション15.8.10.2「非永続的オプティマイザ統計のパラメータの構成」

SHOW MASTER LOGS

セクション13.7.7.1「SHOW BINARY LOGS ステートメント」

SHOW MASTER STATUS

セクション15.20.7「InnoDB memcached プラグインとレプリケーション」
セクション6.2.2「MySQL で提供される権限」
セクション4.5.4「mysqldump — データベースバックアッププログラム」
セクション23.6.9「NDB Cluster レプリケーションによる NDB Cluster バックアップ」
セクション13.7.7.35「SHOW REPLICHA | SLAVE STATUS ステートメント」
セクション17.1.6.5「グローバルトランザクション ID システム変数」
セクション13.4.1「ソースサーバーを制御する SQL ステートメント」
セクション17.5.4「レプリケーションのトラブルシューティング」
セクション17.1.2.4「レプリケーションソースのバイナリログ座標の取得」
セクション17.5.5「レプリケーションバグまたは問題を報告する方法」

SHOW OPEN TABLES

セクション13.7.7.24「SHOW OPEN TABLES ステートメント」

SHOW PLUGINS

セクション6.4.2.1「Connection-Control プラグインのインストール」
セクション5.6.5.1「ddl_rewriter のインストールまたはアンインストール」
セクション26.17「INFORMATION_SCHEMA ndb_transid_mysql_connection_map テーブル」
セクション26.22「INFORMATION_SCHEMA PLUGINS テーブル」
セクション15.13「InnoDB 保存データ暗号化」
セクション13.7.4.4「INSTALL PLUGIN ステートメント」
セクション6.4.1.7「LDAP プラガブル認証」
セクション12.10.9「MeCab フルテキストパーサープラグイン」
セクションA.10「MySQL 8.0 FAQ: NDB Cluster」
セクションA.2「MySQL 8.0 FAQ: ストレージエンジン」
セクション1.3「MySQL 8.0 の新機能」
セクション6.4.5.2「MySQL Enterprise Audit のインストールまたはアンインストール」
NDB Cluster の MySQL Server オプション
セクション23.5.14「ndbinfo: NDB Cluster 情報データベース」
セクション6.4.1.5「PAM プラガブル認証」
セクション13.7.7.25「SHOW PLUGINS ステートメント」
セクション6.4.1.6「Windows プラガブル認証」
セクション6.4.4.1「キーリングプラグインのインストール」
セクション23.5.16「クイックリファレンス: NDB Cluster SQL ステートメント」
セクション5.6.7.1「クローンプラグインのインストール」
セクション5.6.2「サーバープラグイン情報の取得」
セクション5.6.3.2「スレッドプールのインストール」
セクション6.4.1.9「ソケットピア資格証明プラガブル認証」

セクション6.4.1.10「プラグブル認証のテスト」
セクション5.6.1「プラグインのインストールおよびアンインストール」
セクション5.6.7.3「リモートデータのクローニング」
セクション6.4.1.8「ログインなしのプラグブル認証」
セクション17.4.10.2「準同期レプリケーションのインストールと構成」

SHOW plugins

セクション20.5.1「Xプラグインのインストールの確認」

SHOW PRIVILEGES

セクション13.7.7.26「SHOW PRIVILEGES ステートメント」

SHOW PROCEDURE CODE

セクション6.2.2「MySQLで提供される権限」
セクション13.7.7.19「SHOW FUNCTION CODE ステートメント」
セクション25.2.2「ストアドルーチンとMySQL権限」
セクション25.2.3「ストアドルーチンのメタデータ」

SHOW PROCEDURE STATUS

セクション6.2.2「MySQLで提供される権限」
セクション13.7.7.20「SHOW FUNCTION STATUS ステートメント」
セクション25.2.2「ストアドルーチンとMySQL権限」
セクション25.2.3「ストアドルーチンのメタデータ」

SHOW PROCESSLIST

セクション13.4.2.1「CHANGE MASTER TO ステートメント」
セクション13.4.2.3「CHANGE REPLICATION SOURCE TO ステートメント」
セクション27.12.19.1「error_log テーブル」
セクション13.7.1.6「GRANT ステートメント」
セクション17.1.3.1「GTID形式および格納」
セクション26.17「INFORMATION_SCHEMA ndb_transid_mysql_connection_map テーブル」
セクション26.23「INFORMATION_SCHEMA PROCESSLIST テーブル」
セクション15.21.4「InnoDBのエラー処理」
セクション13.7.8.4「KILL ステートメント」
セクションA.14「MySQL 8.0 FAQ: レプリケーション」
セクション1.3「MySQL 8.0の新機能」
セクション6.2.2「MySQLで提供される権限」
セクション4.5.2「mysqladmin — A MySQL Server 管理プログラム」
セクション23.5.9「NDB ClusterでのMySQL Serverの使用」
セクション23.5.14.44「ndbinfo server_locks テーブル」
セクション23.5.14.45「ndbinfo server_operations テーブル」
セクション23.5.14.46「ndbinfo server_transactions テーブル」
セクション27.12.19.9「processlist テーブル」
セクション28.4.3.22「processlist ビューと x\$processlist ビュー」
セクション28.4.5.13「ps_is_thread_instrumented() 関数」
セクション28.4.4.7「ps_setup_disable_thread() プロシージャ」
セクション28.4.4.11「ps_setup_enable_thread() プロシージャ」
セクション28.4.5.15「ps_thread_id() 関数」
セクション13.7.7.29「SHOW PROCESSLIST ステートメント」
セクション13.7.7.30「SHOW PROFILE ステートメント」
セクション13.7.7.35「SHOW REPLICA | SLAVE STATUS ステートメント」
セクション13.4.2.7「START REPLICA | SLAVE ステートメント」
セクション13.3.1「START TRANSACTION、COMMIT および ROLLBACK ステートメント」
セクション25.4.2「イベントスケジューラの構成」
セクション5.6.7.10「クローニング操作の停止」
セクション27.12.19.10「スレッドテーブル」
セクション27.6「パフォーマンススキーマインストゥルメント命名規則」
セクション27.15「パフォーマンススキーマシステム変数」

セクション27.12.5 「パフォーマンススキーマステージイベントテーブル」
セクション12.22 「パフォーマンススキーマ関数」
セクション17.4.8 「フェイルオーバー中のソースの切替え」
セクション8.14.1 「プロセスリストへのアクセス」
セクション17.5.4 「レプリケーションのトラブルシューティング」
セクション17.1.7.1 「レプリケーションステータスの確認」
セクション17.2.3 「レプリケーションスレッド」
セクション17.2.3.1 「レプリケーションメインスレッドの監視」
セクション8.8.4 「名前付き接続の実行計画情報の取得」
セクション12.16 「情報関数」
セクションB.3.2.5 「接続が多すぎます」
セクション5.1.12.1 「接続インタフェース」
セクション6.1.3 「攻撃者に対する MySQL のセキュアな状態の維持」
セクション17.4.11 「遅延レプリケーション」

SHOW PROFILE

セクション26.24 「INFORMATION_SCHEMA PROFILING テーブル」
セクション2.9.7 「MySQL ソース構成オプション」
セクション13.7.7.30 「SHOW PROFILE ステートメント」
セクション13.7.7.31 「SHOW PROFILES ステートメント」
セクション5.1.8 「サーバーシステム変数」
セクション27.19.1 「パフォーマンススキーマを使用したクエリープロファイリング」
セクション8.14.3 「一般的なスレッドの状態」

SHOW PROFILES

セクション26.24 「INFORMATION_SCHEMA PROFILING テーブル」
セクション2.9.7 「MySQL ソース構成オプション」
セクション13.7.7.30 「SHOW PROFILE ステートメント」
セクション13.7.7.31 「SHOW PROFILES ステートメント」
セクション5.1.8 「サーバーシステム変数」
セクション27.19.1 「パフォーマンススキーマを使用したクエリープロファイリング」

SHOW RELAYLOG EVENTS

GTID のあるトランザクションのスキップ
セクション6.2.2 「MySQL で提供される権限」
セクション13.7.7.2 「SHOW BINLOG EVENTS ステートメント」
セクション13.7.7.32 「SHOW RELAYLOG EVENTS ステートメント」
セクション17.1.7.3 「トランザクションのスキップ」
セクション5.4.4.5 「バイナリログトランザクション圧縮」
バイナリログトランザクション圧縮が有効な場合の動作
セクション13.4.2 「レプリケーションサーバーを制御するための SQL ステートメント」
セクション17.2.2.2 「以前のレプリケーションステートメントとの互換性」
セクション17.2.2.1 「単一チャンネルで操作するためのコマンド」

SHOW REPLICA STATUS

セクション13.4.2.3 「CHANGE REPLICATION SOURCE TO ステートメント」
セクション13.7.7.35 「SHOW REPLICA | SLAVE STATUS ステートメント」
セクション13.7.7.36 「SHOW SLAVE | REPLICA STATUS ステートメント」
セクション17.1.7.1 「レプリケーションステータスの確認」

SHOW REPLICA | SLAVE STATUS

セクション13.4.2.1 「CHANGE MASTER TO ステートメント」
セクション17.1.3.7 「GTID ベースレプリケーションの制約」
セクションA.14 「MySQL 8.0 FAQ: レプリケーション」
セクション6.2.2 「MySQL で提供される権限」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション23.6.8 「NDB Cluster レプリケーションによるフェイルオーバーの実装」
セクション23.6.3 「NDB Cluster レプリケーションの既知の問題」

セクション13.4.1.1 「PURGE BINARY LOGS ステートメント」
セクション17.1.6.3 「Replica Server のオプションと変数」
セクション27.12.11.4 「replication_applier_configuration テーブル」
セクション27.12.11.5 「replication_applier_status テーブル」
セクション27.12.11.6 「replication_applier_status_by_coordinator テーブル」
セクション27.12.11.7 「replication_applier_status_by_worker テーブル」
セクション27.12.11.1 「replication_connection_configuration テーブル」
セクション27.12.11.2 「replication_connection_status テーブル」
セクション13.7.7.23 「SHOW MASTER STATUS ステートメント」
セクション13.7.7.35 「SHOW REPLICAS | SLAVE STATUS ステートメント」
セクション13.4.2.7 「START REPLICAS | SLAVE ステートメント」
セクション17.1.6.5 「グローバルトランザクション ID システム変数」
セクション5.1.14 「ネットワークネームスペースのサポート」
セクション27.12.11 「パフォーマンススキーマレプリケーションテーブル」
セクション17.1.5.5 「マルチソースレプリカの開始」
セクション8.14.5 「レプリケーション I/O スレッドの状態」
セクション17.1.6 「レプリケーションおよびバイナリロギングのオプションと変数」
セクション17.5.1.34 「レプリケーションとトランザクションの非一貫性」
セクション17.5.4 「レプリケーションのトラブルシューティング」
セクション13.4.2 「レプリケーションサーバーを制御するための SQL ステートメント」
セクション17.1.7.1 「レプリケーションステータスの確認」
セクション17.5.5 「レプリケーションバグまたは問題を報告する方法」
セクション17.2.3.1 「レプリケーションメインスレッドの監視」
セクション17.2.4.2 「レプリケーションメタデータリポジトリ」
セクション17.1.4.1 「レプリケーションモードの概念」
セクション17.5.1.29 「レプリケーション中のレプリカエラー」
セクション17.1.2.8 「レプリケーション環境へのレプリカの追加」
セクション17.2.2.2 「以前のレプリケーションステートメントとの互換性」
セクション17.2.2.1 「単一チャンネルで操作するためのコマンド」
セクション17.3.1 「暗号化接続を使用するためのレプリケーションの設定」
セクション17.4.11 「遅延レプリケーション」

SHOW REPLICAS

セクション13.7.7.33 「SHOW REPLICAS | SHOW SLAVE HOSTS ステートメント」
セクション13.7.7.34 「SHOW SLAVE HOSTS | SHOW REPLICAS ステートメント」

SHOW REPLICAS | SHOW SLAVE HOSTS

セクション6.2.2 「MySQL で提供される権限」
セクション17.1.6.3 「Replica Server のオプションと変数」
セクション13.4.1 「ソースサーバーを制御する SQL ステートメント」
セクション17.1.6 「レプリケーションおよびバイナリロギングのオプションと変数」
セクション17.1.7.1 「レプリケーションステータスの確認」
セクション17.1.6.2 「レプリケーションソースのオプションと変数」

SHOW SCHEMAS

セクション13.7.7.14 「SHOW DATABASES ステートメント」

SHOW SESSION STATUS

NDB Cluster ステータス変数

SHOW SLAVE HOSTS

セクション13.7.7.33 「SHOW REPLICAS | SHOW SLAVE HOSTS ステートメント」
セクション13.7.7.34 「SHOW SLAVE HOSTS | SHOW REPLICAS ステートメント」

SHOW SLAVE STATUS

セクション13.7.7.35 「SHOW REPLICAS | SLAVE STATUS ステートメント」
セクション13.7.7.36 「SHOW SLAVE | REPLICAS STATUS ステートメント」

セクションB.2「エラー情報インタフェース」
セクション17.1.7.1「レプリケーションステータスの確認」
セクション17.2.3「レプリケーションスレッド」

SHOW STATUS

セクション23.5.13「NDB API 統計のカウンタと変数」
セクション23.3.3.7「NDB Cluster での SQL およびその他の API ノードの定義」
NDB Cluster の MySQL Server オプション
セクション23.1.4「NDB Cluster の新機能」
セクション23.5「NDB Cluster の管理」
セクション23.6「NDB Cluster レプリケーション」
セクション13.7.7.37「SHOW STATUS ステートメント」
セクション8.3.10「インデックス拡張の使用」
セクション23.5.16「クイックリファレンス: NDB Cluster SQL ステートメント」
セクション5.1.8「サーバーシステム変数」
セクション5.1.10「サーバーステータス変数」
セクション25.8「ストアードプログラムの制約」
セクション27.15「パフォーマンススキーマシステム変数」
セクション17.5.1.31「レプリケーションと一時テーブル」
セクション17.4.10.3「準同期レプリケーションモニタリング」

SHOW STATUS LIKE 'perf%'

セクション27.7「パフォーマンススキーマステータスモニタリング」

SHOW TABLE STATUS

セクション16.5「ARCHIVE ストレージエンジン」
セクション13.1.20「CREATE TABLE ステートメント」
セクション13.8.2「EXPLAIN ステートメント」
セクション26.38「INFORMATION_SCHEMA TABLES テーブル」
セクション15.6.1.6「InnoDB での AUTO_INCREMENT 処理」
セクション15.23「InnoDB の制限および制限事項」
セクション15.10「InnoDB の行フォーマット」
セクション15.14「InnoDB の起動オプションおよびシステム変数」
セクション15.6.1.1「InnoDB テーブルの作成」
MySQL 用語集
セクション13.7.7.5「SHOW COLUMNS ステートメント」
セクション13.7.7.10「SHOW CREATE TABLE ステートメント」
セクション13.7.7.38「SHOW TABLE STATUS ステートメント」
セクション24.3.5「パーティションに関する情報を取得する」
セクション15.11.2「ファイル領域管理」
セクション12.20.1「集計関数の説明」
セクション15.8.10.2「非永続的オブティマイザ統計のパラメータの構成」

SHOW TABLES

セクション26.38「INFORMATION_SCHEMA TABLES テーブル」
セクション15.15「InnoDB INFORMATION_SCHEMA テーブル」
セクション6.4.7.3「MySQL Enterprise Firewall の使用」
セクションB.3.3.5「MySQL が一時ファイルを格納する場所」
セクション5.3「mysql システムスキーマ」
MySQL 用語集
セクション23.6.10「NDB Cluster レプリケーション: 双方向および循環レプリケーション」
セクション23.4.23「ndb_restore — NDB Cluster バックアップの復元」
セクション23.5.14「ndbinfo: NDB Cluster 情報データベース」
セクション13.7.7.38「SHOW TABLE STATUS ステートメント」
セクション13.7.7.39「SHOW TABLES ステートメント」
セクション26.55「SHOW ステートメントの拡張」
セクションB.3.6.2「TEMPORARY テーブルに関する問題」
セクション26.1「はじめに」

セクション5.6.3.2「スレッドプールのインストール」
セクション3.3.2「テーブルの作成」
セクション14.1「データディクショナリスキーマ」
セクションB.3.2.14「表 'tbl_name' は存在しません」
セクション9.2.3「識別子の大文字と小文字の区別」

SHOW TABLES FROM some_ndb_database

セクション23.5.17.2「NDB Cluster および MySQL の権限」

SHOW TRIGGERS

セクション26.45「INFORMATION_SCHEMA TRIGGERS テーブル」
セクションA.5「MySQL 8.0 FAQ: トリガー」
セクション13.7.7.40「SHOW TRIGGERS ステートメント」
セクション2.11.5「アップグレード用のインストールの準備」
セクション25.3.2「トリガーのメタデータ」

SHOW VARIABLES

セクション5.8「1つのマシン上での複数のMySQLインスタンスの実行」
セクション15.6.1.3「InnoDB テーブルのインポート」
セクションA.11「MySQL 8.0 FAQ: MySQL の中国語、日本語、および韓国語の文字セット」
セクション23.6.6「NDB Cluster レプリケーションの開始 (シングルレプリケーションチャンネル)」
セクション13.7.7.41「SHOW VARIABLES ステートメント」
セクション25.4.2「イベントスケジューラの構成」
セクション6.4.4.13「キーリングシステム変数」
セクション23.5.16「クイックリファレンス: NDB Cluster SQL ステートメント」
セクション5.1.7「サーバーコマンドオプション」
セクション5.1.8「サーバーシステム変数」
セクション5.1.9「システム変数の使用」
セクション27.15「パフォーマンススキーマシステム変数」
セクション27.3「パフォーマンススキーマ起動構成」
セクション17.1.5.8「マルチソースレプリケーションの監視」
セクション5.6.7.3「リモートデータのクローニング」
セクション6.4.5.9「レガシーモード監査ログのフィルタリング」
セクション13.7.6.1「変数代入の SET 構文」
セクション17.4.10.3「準同期レプリケーションモニタリング」
セクション6.4.5.8「監査ログフィルタ定義の書込み」

SHOW WARNINGS

セクション13.1.9「ALTER TABLE ステートメント」
セクション13.1.29「DROP PROCEDURE および DROP FUNCTION ステートメント」
セクション13.1.32「DROP TABLE ステートメント」
セクション8.2.2.3「EXISTS 戦略を使用したサブクエリーの最適化」
セクション8.8.1「EXPLAIN によるクエリーの最適化」
セクション13.8.2「EXPLAIN ステートメント」
セクション8.8.2「EXPLAIN 出力フォーマット」
セクション13.6.7.3「GET DIAGNOSTICS ステートメント」
セクション10.14.4.3「Index.xml の構文解析中の診断」
セクション8.2.3「INFORMATION_SCHEMA クエリーの最適化」
セクション12.18.7「JSON スキーマ検証関数」
セクション13.2.7「LOAD DATA ステートメント」
セクション1.3「MySQL 8.0 の新機能」
セクション13.6.7.7「MySQL の診断領域」
セクション4.5.1.6「mysql クライアントのヒント」
セクション1.7.3.1「PRIMARY KEY および UNIQUE インデックス制約」
セクション13.7.7.17「SHOW ERRORS ステートメント」
セクション13.7.7.42「SHOW WARNINGS ステートメント」
セクション13.6.7.5「SIGNAL ステートメント」
セクションB.2「エラー情報インタフェース」

- セクション8.9.3 「オプティマイザヒント」
- セクション5.1.8 「サーバーシステム変数」
- セクション17.2.1.1 「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」
- セクション13.1.20.9 「セカンダリインデックスと生成されたカラム」
- セクション5.6.4.2 「リライタクエリーリライトプラグインの使用」
- セクション12.25.4 「丸め動作」
- セクション8.9.2 「切り替え可能な最適化」
- セクション8.2.2.2 「実体化を使用したサブクエリーの最適化」
- セクション8.8.3 「拡張 EXPLAIN 出力形式」
- セクション5.4.4.3 「混合形式のバイナリロギング形式」
- セクション8.2.2.1 「準結合変換による IN および EXISTS サブクエリー述語の最適化」
- セクション8.3.11 「生成されたカラムインデックスのオプティマイザによる使用」
- セクション9.2.5 「関数名の構文解析と解決」

SHUTDOWN

- セクション4.10 「MySQL での Unix シグナル処理」
- セクション6.2.2 「MySQL で提供される権限」
- セクション13.7.8.9 「SHUTDOWN ステートメント」
- セクション18.7.3.2 「グループレプリケーションメンバーのアップグレード」
- セクション5.1.10 「サーバーステータス変数」
- セクション18.4.2.2 「トランザクション一貫性保証の構成」

SIGNAL

- セクション13.6.7.1 「DECLARE ... CONDITION ステートメント」
- セクション13.6.7.2 「DECLARE ... HANDLER ステートメント」
- セクション13.6.7.7 「MySQL の診断領域」
- セクション13.6.7.4 「RESIGNAL ステートメント」
- セクション13.6.7.5 「SIGNAL ステートメント」
- セクション25.8 「ストアプログラムの制約」
- セクション13.6.7.6 「ハンドラのスコープに関するルール」
- セクション12.16 「情報関数」
- セクション13.6.7 「条件の処理」
- セクション13.6.8 「条件処理の制約」

START GROUP REPLICATION

- セクション6.2.2 「MySQL で提供される権限」

START GROUP_REPLICATION

2 番目のインスタンスの追加

- セクション13.4.2.1 「CHANGE MASTER TO ステートメント」
- セクション13.4.2.3 「CHANGE REPLICATION SOURCE TO ステートメント」
- セクション18.5.2 「Secure Socket Layer (SSL) を使用したグループ通信接続の保護」
- セクション13.4.3.2 「STOP GROUP_REPLICATION ステートメント」
- セクション18.10 「よくある質問」

インスタンスの追加

クローニングの前提条件

クローニング操作

- セクション18.2.1.5 「グループのブートストラップ」
- セクション18.5.1 「グループレプリケーション IP アドレスの権限」
- セクション18.8 「グループレプリケーションシステム変数」
- セクション18.7.3.2 「グループレプリケーションメンバーのアップグレード」
- セクション18.7.1 「グループ内の異なるメンバーバージョンの組合せ」
- セクション18.4.4 「ネットワークパーティション化」
- セクション6.1.2.3 「パスワードおよびロギング」
- セクション17.2.4.2 「レプリケーションメタデータリポジトリ」
- レプリケーションユーザー資格証明のセキュアな提供
- セクション5.6.7.6 「レプリケーション用のクローニング」
- セクション18.4.3 「分散リカバリ」

セクション18.2.1.3 「分散リカバリのユーザー資格証明」
分散リカバリエンドポイントのアドレスの選択
セクション6.3.1 「暗号化接続を使用するための MySQL の構成」
セクション18.6.6.4 「終了処理」

START REPLICA

セクション13.4.2.3 「CHANGE REPLICATION SOURCE TO ステートメント」
セクション17.1.3.6 「GTID のないソースから GTID のあるレプリカへのレプリケーション」
セクション13.4.2.7 「START REPLICA | SLAVE ステートメント」
セクション13.4.2.8 「START SLAVE | REPLICA ステートメント」
セクション17.1.7.2 「レプリカでのレプリケーションの一時停止」

START REPLICA SQL_THREAD

セクション13.4.2.3 「CHANGE REPLICATION SOURCE TO ステートメント」

START REPLICA UNTIL SQL_AFTER_MTS_GAPS

セクション13.4.2.3 「CHANGE REPLICATION SOURCE TO ステートメント」

START REPLICA | SLAVE

セクション13.4.2.1 「CHANGE MASTER TO ステートメント」
GTID のあるトランザクションのスキップ
セクション17.1.3.1 「GTID 形式および格納」
セクション6.2.2 「MySQL で提供される権限」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション23.6.10 「NDB Cluster レプリケーション: 双方向および循環レプリケーション」
セクション23.6.7 「NDB Cluster レプリケーションでの 2 つのレプリケーションチャンネルの使用」
セクション23.6.8 「NDB Cluster レプリケーションによるフェイルオーバーの実装」
セクション23.6.6 「NDB Cluster レプリケーションの開始 (シングルレプリケーションチャンネル)」
セクション17.1.6.3 「Replica Server のオプションと変数」
セクション27.12.11.7 「replication_applier_status_by_worker テーブル」
セクション13.4.2.5 「RESET REPLICA | SLAVE ステートメント」
SET GLOBAL sql_slave_skip_counter でのトランザクションのスキップ
セクション13.7.7.35 「SHOW REPLICA | SLAVE STATUS ステートメント」
セクション13.4.2.9 「STOP REPLICA | SLAVE ステートメント」
セクション17.1.7.3 「トランザクションのスキップ」
バイナリログトランザクション圧縮が有効な場合の動作
セクション6.1.2.3 「パスワードおよびロギング」
セクション27.12.11 「パフォーマンススキーマレプリケーションテーブル」
セクション17.4.8 「フェイルオーバー中のソースの切替え」
セクション17.1.5.5 「マルチソースレプリカの開始」
セクション17.1.7.2 「レプリカでのレプリケーションの一時停止」
セクション17.1.6 「レプリケーションおよびバイナリロギングのオプションと変数」
セクション17.5.1.34 「レプリケーションとトランザクションの非一貫性」
セクション17.5.4 「レプリケーションのトラブルシューティング」
セクション17.2.3 「レプリケーションスレッド」
セクション17.2.4.2 「レプリケーションメタデータリポジトリ」
セクション17.5.1.29 「レプリケーション中のレプリカエラー」
セクション17.3.3 「レプリケーション権限チェック」
セクション17.1.2.8 「レプリケーション環境へのレプリカの追加」
セクション17.2.2.2 「以前のレプリケーションステートメントとの互換性」
セクション17.2.2.1 「単一チャンネルで操作するためのコマンド」
セクション17.3.3.3 「失敗したレプリケーション権限チェックからのリカバリ」
既存のデータによるレプリケーションのセットアップ
セクション13.3.3 「暗黙的なコミットを発生させるステートメント」
セクション17.4.10.2 「準同期レプリケーションのインストールと構成」
セクション17.4.6 「異なるレプリカへの異なるデータベースのレプリケート」
セクション17.4.11 「遅延レプリケーション」
セクション17.4.9 「非同期接続フェイルオーバーによるソースの切替え」

START REPLICA | SLAVE SQL_THREAD

セクション13.4.2.1 「CHANGE MASTER TO ステートメント」

セクション13.4.2.2 「CHANGE REPLICATION FILTER ステートメント」

START REPLICA | SLAVE UNTIL SQL_AFTER_MTS_GAPS

セクション13.4.2.1 「CHANGE MASTER TO ステートメント」

セクション17.1.6.3 「Replica Server のオプションと変数」

セクション13.7.7.35 「SHOW REPLICA | SLAVE STATUS ステートメント」

セクション17.4.2 「レプリカの予期しない停止の処理」

セクション17.5.1.34 「レプリケーションとトランザクションの非一貫性」

START SLAVE

セクション23.4.23 「ndb_restore — NDB Cluster バックアップの復元」

セクション13.4.2.7 「START REPLICA | SLAVE ステートメント」

セクション13.4.2.8 「START SLAVE | REPLICATION ステートメント」

セクション17.1.7.2 「レプリカでのレプリケーションの一時停止」

START TRANSACTION

セクション13.6.1 「BEGIN ... END 複合ステートメント」

セクション27.12.7.1 「events_transactions_current テーブル」

セクション13.7.8.3 「FLUSH ステートメント」

セクション8.5.3 「InnoDB の読み取り専用トランザクションの最適化」

セクション13.3.6 「LOCK TABLES および UNLOCK TABLES ステートメント」

セクション4.5.4 「mysqldump — データベースバックアッププログラム」

セクション4.5.6 「mysqlpump — データベースバックアッププログラム」

セクション13.3.7 「SET TRANSACTION ステートメント」

セクション13.3.1 「START TRANSACTION、COMMIT および ROLLBACK ステートメント」

セクション13.3.8.2 「XA トランザクションの状態」

セクション5.1.18 「クライアントセッション状態の変更のサーバトラッキング」

セクション5.1.8 「サーバーシステム変数」

セクション25.8 「ストアードプログラムの制約」

セクション15.7.5.3 「デッドロックを最小化および処理する方法」

セクション13.3 「トランザクションステートメントおよびロックステートメント」

セクション25.3.1 「トリガーの構文と例」

セクション27.12.7 「パフォーマンススキーマのトランザクションテーブル」

セクション13.3.3 「暗黙的なコミットを発生させるステートメント」

セクション17.4.10 「準同期レプリケーション」

セクション15.7.2.2 「自動コミット、コミットおよびロールバック」

セクション15.7.2.4 「読取りのロック」

START TRANSACTION ... COMMIT

セクション13.1.1 「アトミックデータ定義ステートメントのサポート」

データ定義ステートメント

START TRANSACTION READ ONLY

セクション8.5.3 「InnoDB の読み取り専用トランザクションの最適化」

MySQL 用語集

START TRANSACTION WITH CONSISTENT SNAPSHOT

セクション15.7.2.3 「一貫性非ロック読み取り」

STATS_PERSISTENT=0

セクション15.8.10.2 「非永続的オブティマイザ統計のパラメータの構成」

STATS_PERSISTENT=1

セクション15.8.10.1 「永続的オブティマイザ統計のパラメータの構成」

STOP GROUP REPLICATION

セクション6.2.2「MySQL で提供される権限」
セクション13.4.2.5「RESET REPLICAS | SLAVE ステートメント」

STOP GROUP_REPLICATION

セクション18.6.6.3「Auto-Rejoin」
セクション18.5.2「Secure Socket Layer (SSL) を使用したグループ通信接続の保護」
セクション13.4.3.2「STOP GROUP_REPLICATION ステートメント」
セクション18.10「よくある質問」
セクション18.1.4.1「グループメンバーシップ」
セクション18.5.1「グループレプリケーション IP アドレスの権限」
セクション18.8「グループレプリケーションシステム変数」
セクション18.7.3.2「グループレプリケーションメンバーのアップグレード」
セクション18.4.2.2「トランザクション一貫性保証の構成」
レプリケーションユーザー資格証明のセキュアな提供
セクション18.6.6.2「使用できない大多数のタイムアウト」
セクション18.2.1.3「分散リカバリのユーザー資格証明」
セクション6.3.1「暗号化接続を使用するための MySQL の構成」

STOP REPLICA

セクション13.4.2.3「CHANGE REPLICATION SOURCE TO ステートメント」
セクション13.4.2.9「STOP REPLICA | SLAVE ステートメント」
セクション13.4.2.10「STOP SLAVE | REPLICA ステートメント」
セクション17.1.7.2「レプリカでのレプリケーションの一時停止」

STOP REPLICA | SLAVE

セクション13.4.2.1「CHANGE MASTER TO ステートメント」
セクション17.1.3.2「GTID ライフサイクル」
セクション6.2.2「MySQL で提供される権限」
セクション4.5.4「mysqldump — データベースバックアッププログラム」
セクション17.1.6.3「Replica Server のオプションと変数」
セクション27.12.11.7「replication_applier_status_by_worker テーブル」
セクション13.4.1.2「RESET MASTER ステートメント」
セクション13.4.2.5「RESET REPLICAS | SLAVE ステートメント」
セクション13.7.7.35「SHOW REPLICAS | SLAVE STATUS ステートメント」
セクション13.4.2.7「START REPLICA | SLAVE ステートメント」
セクション27.12.11「パフォーマンススキーマレプリケーションテーブル」
セクション17.4.8「フェイルオーバー中のソースの切替え」
セクション17.1.5.6「マルチソースレプリカの停止」
セクション17.1.7.2「レプリカでのレプリケーションの一時停止」
セクション17.1.6「レプリケーションおよびバイナリロギングのオプションと変数」
セクション17.5.1.34「レプリケーションとトランザクションの非一貫性」
セクション17.1.7.1「レプリケーションステータスの確認」
セクション17.3.3「レプリケーション権限チェック」
セクション17.1.2.8「レプリケーション環境へのレプリカの追加」
セクション17.2.2.2「以前のレプリケーションステートメントとの互換性」
セクション17.2.2.1「単一チャンネルで操作するためのコマンド」
セクション13.3.3「暗黙的なコミットを発生させるステートメント」
セクション17.4.10.2「準同期レプリケーションのインストールと構成」
セクション17.2.1.2「行ベースロギングおよびレプリケーションの使用」
セクション17.4.11「遅延レプリケーション」
セクション17.4.9「非同期接続フェイルオーバーによるソースの切替え」

STOP REPLICA | SLAVE IO_THREAD

セクション17.4.8「フェイルオーバー中のソースの切替え」

STOP REPLICA | SLAVE SQL_THREAD

セクション13.4.2.2「CHANGE REPLICATION FILTER ステートメント」

セクション17.2.1.2 「行ベースロギングおよびレプリケーションの使用」

STOP SLAVE

セクション13.4.2.9 「STOP REPLICA | SLAVE ステートメント」

セクション13.4.2.10 「STOP SLAVE | REPLICA ステートメント」

セクション17.1.7.2 「レプリカでのレプリケーションの一時停止」

T

[\[index top\]](#)

TABLE

セクション13.2.11.4 「ALL を使用したサブクエリー」

セクション13.2.11.3 「ANY、IN、または SOME を使用したサブクエリー」

セクション13.1.20.4 「CREATE TABLE ... SELECT ステートメント」

セクション13.1.23 「CREATE VIEW ステートメント」

セクション13.2.11.6 「EXISTS または NOT EXISTS を使用したサブクエリー」

セクション13.8.2 「EXPLAIN ステートメント」

セクション13.2.6.1 「INSERT ... SELECT ステートメント」

セクション13.2.6 「INSERT ステートメント」

セクション1.3 「MySQL 8.0 の新機能」

セクション13.2.9 「REPLACE ステートメント」

セクション13.2.10.1 「SELECT ... INTO ステートメント」

セクション13.2.12 「TABLE ステートメント」

セクション13.2.10.3 「UNION 句」

セクション13.2.14 「VALUES ステートメント」

セクション13.2.11 「サブクエリー」

セクション13.2.11.10 「サブクエリーのエラー」

セクション13.2.11.1 「スカラーオペランドとしてのサブクエリー」

TRUNCATE PARTITION

セクション15.12.1 「オンライン DDL 操作」

TRUNCATE TABLE

セクション27.12.8.1 「accounts テーブル」

セクション13.1.2 「ALTER DATABASE ステートメント」

セクション27.12.11.12 「binary_log_transaction_compression_stats テーブル」

セクション27.12.17.2 「clone_progress テーブル」

セクション27.12.17.1 「clone_status テーブル」

セクション27.12.3.1 「cond_instances テーブル」

セクション13.1.22 「CREATE TRIGGER ステートメント」

セクション27.12.13.2 「data_lock_waits テーブル」

セクション27.12.13.1 「data_locks テーブル」

セクション13.2.2 「DELETE ステートメント」

セクション5.1.12.3 「DNS ルックアップとホストキャッシュ」

セクション27.12.19.1 「error_log テーブル」

セクション27.12.5.1 「events_stages_current テーブル」

セクション27.12.5.2 「events_stages_history テーブル」

セクション27.12.5.3 「events_stages_history_long テーブル」

セクション27.12.6.1 「events_statements_current テーブル」

セクション27.12.6.2 「events_statements_history テーブル」

セクション27.12.6.3 「events_statements_history_long テーブル」

セクション27.12.7.1 「events_transactions_current テーブル」

セクション27.12.7.2 「events_transactions_history テーブル」

セクション27.12.7.3 「events_transactions_history_long テーブル」

セクション27.12.4.1 「events_waits_current テーブル」

セクション27.12.4.2 「events_waits_history テーブル」

セクション27.12.4.3 「events_waits_history_long テーブル」

セクション16.8.3 「FEDERATED ストレージエンジンの注記とヒント」
セクション15.6.3.2 「File-Per-Table テーブルスペース」
セクション27.12.3.2 「file_instances テーブル」
セクション27.12.19.3 「firewall_group_allowlist テーブル」
セクション27.12.19.2 「firewall_groups テーブル」
セクション27.12.19.4 「firewall_membership テーブル」
セクション13.7.8.3 「FLUSH ステートメント」
セクション13.2.4 「HANDLER ステートメント」
セクション27.12.19.5 「host_cache テーブル」
セクション27.12.8.2 「hosts テーブル」
セクション26.51.19 「INFORMATION_SCHEMA INNODB_INDEXES テーブル」
セクション26.51.24 「INFORMATION_SCHEMA INNODB_TABLES テーブル」
セクション8.5.7 「InnoDB DDL 操作の最適化」
セクション15.20.7 「InnoDB memcached プラグインとレプリケーション」
セクション15.20.8 「InnoDB memcached プラグインの内部」
セクション27.12.19.6 「keyring_keys テーブル」
セクション13.3.5 「LOCK INSTANCE FOR BACKUP および UNLOCK INSTANCE ステートメント」
セクション13.3.6 「LOCK TABLES および UNLOCK TABLES ステートメント」
セクション27.12.19.7 「log_status テーブル」
セクション15.20.6.5 「memcached 操作に合わせた DML ステートメントの改変」
セクション16.3 「MEMORY ストレージエンジン」
セクション16.7.2 「MERGE テーブルの問題点」
セクション27.12.13.3 「metadata_locks テーブル」
セクション27.12.3.3 「mutex_instances テーブル」
セクション15.6.1.5 「MyISAM から InnoDB へのテーブルの変換」
セクション1.3 「MySQL 8.0 の新機能」
セクション6.2.2 「MySQL で提供される権限」
セクション5.4.4.4 「mysql データベーステーブルへの変更に対するロギング形式」
MySQL 用語集
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.6 「mysqldump — データベースバックアッププログラム」
セクション23.1.7.3 「NDB Cluster でのトランザクション処理に関する制限」
セクション23.1.7.2 「NDB Cluster と標準の MySQL 制限の制限と相違点」
セクション23.5.7.1 「NDB Cluster データノードのオンラインでの追加: 一般的な問題」
セクション23.6.9.2 「NDB Cluster レプリケーションを使用したポイントインタイムリカバリ」
セクション23.5.1 「NDB Cluster 管理クライアントのコマンド」
セクション23.4.8 「ndb_delete_all — NDB テーブルからのすべての行の削除」
セクション27.12.19.8 「performance_timers テーブル」
セクション27.12.6.4 「prepared_statements_instances テーブル」
セクション27.12.19.9 「processlist テーブル」
セクション28.4.4.24 「ps_truncate_all_tables() プロシージャ」
セクション24.3.1 「RANGE および LIST パーティションの管理」
セクション27.12.11.4 「replication_applier_configuration テーブル」
セクション27.12.11.5 「replication_applier_status テーブル」
セクション27.12.11.3 「replication_asynchronous_connection_failover テーブル」
セクション27.12.11.1 「replication_connection_configuration テーブル」
セクション27.12.11.11 「replication_group_member_stats テーブル」
セクション27.12.11.10 「replication_group_members テーブル」
セクション27.12.3.4 「rlock_instances テーブル」
セクション27.12.9.1 「session_account_connect_attrs テーブル」
セクション27.12.9.2 「session_connect_attrs テーブル」
セクション27.12.2.1 「setup_actors テーブル」
セクション27.12.2.2 「setup_consumers テーブル」
セクション27.12.2.3 「setup_instruments テーブル」
セクション27.12.2.4 「setup_objects テーブル」
セクション27.12.2.5 「setup_threads テーブル」
セクション27.12.3.5 「socket_instances テーブル」
セクション27.12.13.4 「table_handles テーブル」
table_io_waits_summary_by_index_usage テーブル
table_io_waits_summary_by_table テーブル

table_lock_waits_summary_by_table テーブル
セクション27.12.19.11 「tls_channel_status テーブル」
セクション27.12.16.1 「tp_thread_group_state テーブル」
セクション27.12.16.2 「tp_thread_group_stats テーブル」
セクション27.12.16.3 「tp_thread_state テーブル」
セクション13.1.37 「TRUNCATE TABLE ステートメント」
セクション27.12.19.12 「user_defined_functions テーブル」
セクション27.12.8.3 「users テーブル」
セクション13.1.1 「アトミックデータ定義ステートメントのサポート」
セクション27.4.3 「イベントの事前フィルタリング」
セクション27.12.18.11 「エラー要約テーブル」
セクション27.12.18.6 「オブジェクト待機サマリーテーブル」
セクション5.6.7.13 「クローンプラグインの制限事項」
セクション5.1.8 「サーバーシステム変数」
セクション27.12.18.2 「ステージサマリーテーブル」
セクション27.12.18.12 「ステータス変数サマリーテーブル」
セクション27.12.18.3 「ステートメントサマリーテーブル」
セクション27.12.18.4 「ステートメントヒストグラム要約テーブル」
セクション27.12.19.10 「スレッドテーブル」
セクション27.12.18.9 「ソケットサマリーテーブル」
セクション27.12.18.5 「トランザクション要約テーブル」
セクション27.12.14.1 「パフォーマンススキーマ persisted_variables テーブル」
セクション27.12.14.2 「パフォーマンススキーマ variables_info テーブル」
セクション27.12.15 「パフォーマンススキーマのステータス変数のテーブル」
セクション27.12.10 「パフォーマンススキーマのユーザー定義変数テーブル」
セクション27.11 「パフォーマンススキーマの一般的なテーブル特性」
セクション27.12.18 「パフォーマンススキーマサマリーテーブル」
セクション27.12.14 「パフォーマンススキーマシステム変数テーブル」
セクション27.12.8 「パフォーマンススキーマ接続テーブル」
セクション24.3.4 「パーティションの保守」
セクション27.12.18.7 「ファイル I/O サマリーテーブル」
セクション27.12.18.10 「メモリーサマリーテーブル」
セクション17.5.1.21 「レプリケーションと MEMORY テーブル」
セクション17.5.1.37 「レプリケーションと TRUNCATE TABLE」
セクション5.4.1 「一般クエリーログおよびスロークエリーログの出力先の選択」
セクション16.2.3.3 「圧縮テーブルの特徴」
セクション27.12.18.1 「待機イベント要約テーブル」
セクション13.3.3 「暗黙的なコミットを発生させるステートメント」
セクション6.4.5.8 「監査ログフィルタ定義の書込み」

U

[[index top](#)]

UNINSTALL COMPONENT

セクション6.2.2 「MySQL で提供される権限」
セクション13.7.4.5 「UNINSTALL COMPONENT ステートメント」
セクション13.1.1 「アトミックデータ定義ステートメントのサポート」
セクション5.5.3 「エラーログコンポーネント」
セクション5.4.2.1 「エラーログ構成」
セクション9.6 「クエリー属性」
セクション5.5.1 「コンポーネントのインストールおよびアンインストール」
セクション6.4.3.1 「パスワード検証コンポーネントのインストールおよびアンインストール」
セクション6.4.6 「監査メッセージコンポーネント」

UNINSTALL PLUGIN

セクション5.6.5.1 「ddl_rewriter のインストールまたはアンインストール」
セクション13.7.8.3 「FLUSH ステートメント」
セクション26.22 「INFORMATION_SCHEMA PLUGINS テーブル」

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション13.7.4.4 「INSTALL PLUGIN ステートメント」
セクション6.4.4.5 「keyring_aws Amazon Web Services キーリングプラグインの使用」
セクション6.4.1.7 「LDAP プラガブル認証」
セクション6.5.2 「MySQL Enterprise Data Masking and De-Identification のインストールまたはアンインストール」
セクション8.12.3.1 「MySQL のメモリーの使用方法」
セクション6.4.1.5 「PAM プラガブル認証」
セクション13.7.7.25 「SHOW PLUGINS ステートメント」
セクション13.7.4.6 「UNINSTALL PLUGIN ステートメント」
セクション6.4.1.6 「Windows プラガブル認証」
セクション13.1.1 「アトミックデータ定義ステートメントのサポート」
セクション6.4.1.9 「ソケットピア資格証明プラグブル認証」
セクション5.6.6.2 「バージョントークンのインストールまたはアンインストール」
セクション27.18 「パフォーマンススキーマとプラグイン」
セクション16.11.1 「プラグブルストレージエンジンのアーキテクチャー」
セクション6.4.1.10 「プラグブル認証のテスト」
セクション5.6.1 「プラグインのインストールおよびアンインストール」
セクション6.4.1.8 「ログインなしのプラグブル認証」
セクション13.3.3 「暗黙的なコミットを発生させるステートメント」
セクション6.4.4.10 「汎用キーリングキー管理関数」
セクション6.4.5.7 「監査ログのフィルタリング」
セクション6.4.5.10 「監査ログ参照」

UNION

セクション3.6.7 「2 つのキーを使用した検索」
セクション13.1.20 「CREATE TABLE ステートメント」
セクション13.1.23 「CREATE VIEW ステートメント」
セクション8.8.2 「EXPLAIN 出力フォーマット」
セクション15.7.3 「InnoDB のさまざまな SQL ステートメントで設定されたロック」
セクション13.2.6.2 「INSERT ... ON DUPLICATE KEY UPDATE ステートメント」
セクション16.7 「MERGE ストレージエンジン」
セクション1.3 「MySQL 8.0 の新機能」
セクション8.4.4 「MySQL での内部一時テーブルの使用」
セクション8.2.1.2 「range の最適化」
セクション13.2.10.1 「SELECT ... INTO ステートメント」
セクション13.2.10 「SELECT ステートメント」
セクション13.2.12 「TABLE ステートメント」
セクション13.2.10.3 「UNION 句」
セクション13.2.14 「VALUES ステートメント」
セクション13.2.15 「WITH (共通テーブル式)」
セクション12.12 「XML 関数」
セクション13.2.10.4 「カッコで囲まれたクエリー式」
セクション13.2.11 「サブクエリー」
セクション5.1.10 「サーバーステータス変数」
セクション25.5.1 「ビューの構文」
セクション8.2.2.4 「マージまたは実体化を使用した導出テーブル、ビュー参照および共通テーブル式の最適化」
セクション8.2.2.5 「導出条件プッシュダウン最適化」
セクション12.16 「情報関数」
セクション11.1.6 「数値型の属性」
セクション25.5.3 「更新可能および挿入可能なビュー」
セクション8.2.2.1 「準結合変換による IN および EXISTS サブクエリー述語の最適化」
セクション12.8.3 「関数結果の文字セットと照合順序」

UNION ALL

セクション8.4.4 「MySQL での内部一時テーブルの使用」
セクション13.2.10.3 「UNION 句」
セクション8.2.2.4 「マージまたは実体化を使用した導出テーブル、ビュー参照および共通テーブル式の最適化」
セクション12.16 「情報関数」
セクション25.5.3 「更新可能および挿入可能なビュー」

UNION DISTINCT

セクション13.2.10.3「UNION 句」
セクション13.2.15「WITH (共通テーブル式)」

UNLOCK INSTANCE

セクション1.3「MySQL 8.0 の新機能」

UNLOCK TABLES

セクション13.7.8.3「FLUSH ステートメント」
セクション15.7.3「InnoDB のさまざまな SQL ステートメントで設定されたロック」
セクション15.6.1.3「InnoDB テーブルのインポート」
セクション13.3.6「LOCK TABLES および UNLOCK TABLES ステートメント」
セクション8.6.2「MyISAM テーブルの一括データロード」
セクション4.5.4「mysqldump — データベースバックアッププログラム」
セクション4.5.6「mysqlpump — データベースバックアッププログラム」
セクション13.3.1「START TRANSACTION、COMMIT および ROLLBACK ステートメント」
セクション25.8「ストアドプログラムの制約」
セクション15.7.5.3「デッドロックを最小化および処理する方法」
セクション7.2「データベースバックアップ方法」
セクション13.3.3「暗黙的なコミットを発生させるステートメント」

UPDATE

セクション13.1.2「ALTER DATABASE ステートメント」
セクション16.5「ARCHIVE ストレージエンジン」
セクション16.6「BLACKHOLE ストレージエンジン」
セクション13.7.3.2「CHECK TABLE ステートメント」
セクション13.1.20.6「CHECK 制約」
セクション16.8.2.1「CONNECTION を使用した FEDERATED テーブルの作成」
セクション13.1.20.8「CREATE TABLE および生成されるカラム」
セクション13.1.20.2「CREATE TEMPORARY TABLE ステートメント」
セクション13.1.22「CREATE TRIGGER ステートメント」
セクション13.1.23「CREATE VIEW ステートメント」
セクションB.3.4.2「DATE カラムの使用に関する問題」
セクション8.8.1「EXPLAIN によるクエリーの最適化」
セクション13.8.2「EXPLAIN ステートメント」
セクション8.8.2「EXPLAIN 出力フォーマット」
セクション16.8.3「FEDERATED ストレージエンジンの注記とヒント」
セクション13.1.20.5「FOREIGN KEY の制約」
セクション1.7.2.3「FOREIGN KEY 制約の違い」
セクション13.7.1.6「GRANT ステートメント」
セクション26.51.27「INFORMATION_SCHEMA INNODB_TABLESTATS ビュー」
セクション26.38「INFORMATION_SCHEMA TABLES テーブル」
セクション26.48「INFORMATION_SCHEMA VIEWS テーブル」
セクション15.6.1.6「InnoDB での AUTO_INCREMENT 処理」
セクション15.7.3「InnoDB のさまざまな SQL ステートメントで設定されたロック」
セクション15.7.5「InnoDB のデッドロック」
セクション15.21.2「InnoDB のリカバリの強制的な実行」
セクション15.14「InnoDB の起動オプションおよびシステム変数」
セクション15.1.2「InnoDB テーブルのベストプラクティス」
セクション8.5.5「InnoDB テーブルの一括データロード」
セクション15.7.1「InnoDB ロック」
セクション13.2.6.2「INSERT ... ON DUPLICATE KEY UPDATE ステートメント」
セクション13.2.6「INSERT ステートメント」
セクション13.2.10.2「JOIN 句」
セクション11.5「JSON データ型」
セクション12.18.8「JSON ユーティリティ関数」
セクション13.7.8.4「KILL ステートメント」
セクション13.2.7「LOAD DATA ステートメント」

セクション13.3.6「LOCK TABLES および UNLOCK TABLES ステートメント」
セクション16.7「MERGE ストレージエンジン」
セクション15.6.1.5「MyISAM から InnoDB へのテーブルの変換」
セクション16.2「MyISAM ストレージエンジン」
セクション8.6.2「MyISAM テーブルの一括データロード」
セクションA.4「MySQL 8.0 FAQ: ストアドプロシージャおよびストアドファンクション」
セクション2.11.4「MySQL 8.0 での変更」
セクション1.3「MySQL 8.0 の新機能」
セクション8.4.4「MySQL での内部一時テーブルの使用」
セクション6.2.2「MySQL で提供される権限」
セクション24.1「MySQL のパーティショニングの概要」
セクション1.2.2「MySQL の主な機能」
セクションB.3.7「MySQL の既知の問題」
セクション6.2.21「MySQL への接続の問題のトラブルシューティング」
セクション4.5.1.6「mysql クライアントのヒント」
セクション4.5.1.1「mysql クライアントオプション」
セクション5.4.4.4「mysql データベーステーブルへの変更に対するロギング形式」
MySQL 用語集
セクション4.6.8.2「mysqlbinlog 行イベントの表示」
セクション23.1.4「NDB Cluster の新機能」
セクション23.5.10.1「NDB Cluster ディスクデータオブジェクト」
セクション23.6.3「NDB Cluster レプリケーションの既知の問題」
セクション23.6.11「NDB Cluster レプリケーションの競合解決」
セクション15.9.1.6「OLTP ワークロードの圧縮」
セクション13.5.1「PREPARE ステートメント」
セクション1.7.3.1「PRIMARY KEY および UNIQUE インデックス制約」
セクション8.2.1.2「range の最適化」
セクション17.1.6.3「Replica Server のオプションと変数」
セクション13.7.7.38「SHOW TABLE STATUS ステートメント」
セクション13.7.7.42「SHOW WARNINGS ステートメント」
セクション8.3.3「SPATIAL インデックス最適化」
セクション12.1「SQL 関数および演算子リファレンス」
セクション28.4.2.3「sys_config_update_set_user トリガー」
セクション15.6.6「undo ログ」
セクション1.7.2.2「UPDATE の違い」
セクション13.2.13「UPDATE ステートメント」
セクション8.2.1.1「WHERE 句の最適化」
セクション13.2.15「WITH (共通テーブル式)」
セクション3.3.4.1「すべてのデータの選択」
セクション12.24「その他の関数」
セクション26.1「はじめに」
セクション6.2.8「アカウントの追加、権限の割当ておよびアカウントの削除」
セクション6.2.11「アカウントカテゴリ」
セクション6.2.7「アクセス制御、ステージ 2: リクエストの確認」
セクション6.2「アクセス制御とアカウント管理」
セクション8.9.4「インデックスヒント」
セクション15.8.11「インデックスページのマージしきい値の構成」
セクション12.21.5「ウィンドウ機能の制限事項」
セクション8.9.3「オプティマイザヒント」
セクション15.12.1「オンライン DDL 操作」
セクション10.7「カラム文字セットの変換」
セクション13.2.11「サブクエリー」
セクション8.2.2「サブクエリー、導出テーブル、ビュー参照および共通テーブル式の最適化」
セクション5.1.11「サーバー SQL モード」
セクション5.1.19「サーバーの停止プロセス」
セクション5.1.8「サーバーシステム変数」
セクション5.1.10「サーバーステータス変数」
セクション17.2.1.1「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」
セクション27.4.6「スレッドによる事前フィルタリング」
セクション13.1.20.9「セカンダリインデックスと生成されたカラム」

セクション8.11.2「テーブルロックの問題」
セクション11.6「データ型デフォルト値」
セクション8.2.5「データ変更ステートメントの最適化」
セクション15.7.2.1「トランザクション分離レベル」
セクション25.3.1「トリガーの構文と例」
セクション17.2.1.3「バイナリロギングでの安全および安全でないステートメントの判断」
セクション17.1.6.4「バイナリロギングのオプションと変数」
セクション5.4.4「バイナリログ」
セクション10.8.5「バイナリ照合順序と_bin 照合順序」
セクション6.1.2.3「パスワードおよびロギング」
セクション15.8.9「ページ構成」
セクション24.6「パーティショニングの制約と制限」
セクション24.4「パーティションプルニング」
セクション24.5「パーティション選択」
セクション8.10.3「プリペアドステートメントおよびストアードプログラムのキャッシュ」
セクション5.6.4「リライタクエリーリライトプラグイン」
セクション5.6.4.2「リライタクエリーリライトプラグインの使用」
セクション17.5.1.27「レプリケーションおよび行検索」
セクション17.5.1.18「レプリケーションとLIMIT」
セクション17.5.1.23「レプリケーションとクエリー最適マイザ」
セクション17.5.1.36「レプリケーションとトリガー」
セクション17.2.5.3「レプリケーションフィルタリングオプション間の相互作用」
セクション17.5.1.29「レプリケーション中のレプリカエラー」
セクション5.4.1「一般クエリーログおよびスロークエリーログの出力先の選択」
セクション8.14.3「一般的なスレッドの状態」
セクション15.7.2.3「一貫性非ロック読み取り」
セクション6.2.3「付与テーブル」
セクション12.10.5「全文制限」
セクション8.11.1「内部ロック方法」
セクション12.4.4「割り当て演算子」
セクション8.8.4「名前付き接続の実行計画情報の取得」
セクション15.5.2「変更バッファ」
セクション8.2.2.2「実体化を使用したサブクエリーの最適化」
セクション12.16「情報関数」
セクション8.8.3「拡張 EXPLAIN 出力形式」
セクション11.2.1「日時データ型の構文」
セクション25.5.3「更新可能および挿入可能なビュー」
セクション1.7.1「標準 SQL に対する MySQL 拡張機能」
セクション6.2.13「権限変更が有効化される時期」
セクション8.2.2.1「準結合変換による IN および EXISTS サブクエリー述語の最適化」
セクション12.4「演算子」
セクション6.4.5.7「監査ログのフィルタリング」
セクション6.4.5.8「監査ログフィルタ定義の書込み」
セクション6.4.5.10「監査ログ参照」
セクション11.1.7「範囲外およびオーバーフローの処理」
セクション17.2.1.2「行ベースロギングおよびレプリケーションの使用」
セクション15.7.2.4「読み取りのロック」
第12章「関数と演算子」
セクション8.2.1.20「関数コールの最適化」
セクション13.1.20.10「非表示カラム」

UPDATE ... ()

セクション15.7.2.3「一貫性非ロック読み取り」

UPDATE ... WHERE

セクション15.7.5「InnoDB のデッドロック」

UPDATE ... WHERE ...

セクション15.7.3「InnoDB のさまざまな SQL ステートメントで設定されたロック」

UPDATE IGNORE

[セクション13.1.20.6「CHECK 制約」](#)
[セクション13.2.13「UPDATE ステートメント」](#)
[セクション5.1.11「サーバー SQL モード」](#)

USE

[セクション13.1.17「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」](#)
[セクション4.5.1.1「mysql クライアントオプション」](#)
[セクション4.6.8「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」](#)
[セクション4.5.4「mysqldump — データベースバックアッププログラム」](#)
[セクション7.4.1「mysqldump による SQL フォーマットでのデータのダンプ」](#)
[セクション4.5.6「mysqlpump — データベースバックアッププログラム」](#)
[セクション23.5.14「ndbinfo: NDB Cluster 情報データベース」](#)
[セクション17.1.6.3「Replica Server のオプションと変数」](#)
[セクション7.4.2「SQL フォーマットバックアップのリロード」](#)
[セクション13.8.4「USE ステートメント」](#)
[セクション26.1「はじめに」](#)
[セクション7.4.5.2「サーバー間でのデータベースのコピー」](#)
[セクション25.2.1「ストアドルーチンの構文」](#)
[セクション3.3「データベースの作成と使用」](#)
[セクション3.3.1「データベースの作成と選択」](#)
[セクション17.2.5.1「データベースレベルレプリケーションオプションおよびバイナリロギングオプションの評価」](#)
[セクション18.4.2.2「トランザクション一貫性保証の構成」](#)
[セクション17.1.6.4「バイナリロギングのオプションと変数」](#)
[セクション17.2.5.3「レプリケーションフィルタリングオプション間の相互作用」](#)

USE db2

[セクション4.6.8「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」](#)

USE db_name

[セクション4.5.1.1「mysql クライアントオプション」](#)

USE test

[セクション4.6.8「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」](#)

V

[\[index top\]](#)

VALUES

[セクション13.1.20.4「CREATE TABLE ... SELECT ステートメント」](#)
[セクション13.1.23「CREATE VIEW ステートメント」](#)
[セクション1.3「MySQL 8.0 の新機能」](#)
[セクション13.2.10.1「SELECT ... INTO ステートメント」](#)
[セクション13.2.12「TABLE ステートメント」](#)
[セクション13.2.10.3「UNION 句」](#)
[セクション13.2.14「VALUES ステートメント」](#)
[セクション13.2.11「サブクエリー」](#)

VALUES ROW()

[セクション13.2.6「INSERT ステートメント」](#)
[セクション13.2.9「REPLACE ステートメント」](#)

W

[\[index top\]](#)

WHERE

[セクション15.1.1 「InnoDB テーブルを使用する利点」](#)

WHILE

[セクション13.6.5.3 「ITERATE ステートメント」](#)
[セクション13.6.5.4 「LEAVE ステートメント」](#)
[セクション13.6.5.8 「WHILE ステートメント」](#)
[セクション13.6.2 「ステートメントラベル」](#)
[セクション13.6.5 「フロー制御ステートメント」](#)

WITH

[セクション13.2.2 「DELETE ステートメント」](#)
[セクション1.3 「MySQL 8.0 の新機能」](#)
[セクション13.2.10 「SELECT ステートメント」](#)
[セクションB.3.6.2 「TEMPORARY テーブルに関する問題」](#)
[セクション13.2.13 「UPDATE ステートメント」](#)
[セクション13.2.15 「WITH \(共通テーブル式\)」](#)
[セクション8.2.2.4 「マージまたは実体化を使用した導出テーブル、ビュー参照および共通テーブル式の最適化」](#)

X

[\[index top\]](#)

XA BEGIN

[セクション27.12.7 「パフォーマンススキーマのトランザクションテーブル」](#)

XA COMMIT

[セクション27.12.7.1 「events_transactions_current テーブル」](#)
[セクション2.11.6 「Unix/Linux での MySQL バイナリまたはパッケージベースのインストールのアップグレード」](#)
[セクション13.3.8.2 「XA トランザクションの状態」](#)
[セクション5.1.8 「サーバーシステム変数」](#)
[セクション27.12.7 「パフォーマンススキーマのトランザクションテーブル」](#)
[セクション8.11.4 「メタデータのロック」](#)

XA END

[セクション27.12.7.1 「events_transactions_current テーブル」](#)
[セクション13.3.8.1 「XA トランザクション SQL ステートメント」](#)
[セクション13.3.8.3 「XA トランザクションの制約」](#)
[セクション13.3.8.2 「XA トランザクションの状態」](#)

XA PREPARE

[セクション27.12.7.1 「events_transactions_current テーブル」](#)
[セクション13.3.8.2 「XA トランザクションの状態」](#)
[セクション5.1.8 「サーバーシステム変数」](#)

XA RECOVER

[セクション27.12.7.1 「events_transactions_current テーブル」](#)
[セクション13.7.1.6 「GRANT ステートメント」](#)
[セクション6.2.2 「MySQL で提供される権限」](#)
[セクション2.11.6 「Unix/Linux での MySQL バイナリまたはパッケージベースのインストールのアップグレード」](#)
[セクション13.3.8.1 「XA トランザクション SQL ステートメント」](#)
[セクション13.3.8.3 「XA トランザクションの制約」](#)
[セクション13.3.8.2 「XA トランザクションの状態」](#)

XA ROLLBACK

[セクション27.12.7.1 「events_transactions_current テーブル」](#)

[セクション2.11.6「Unix/LinuxでのMySQLバイナリまたはパッケージベースのインストールのアップグレード」](#)
[セクション13.3.8.2「XAトランザクションの状態」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション27.12.7「パフォーマンススキーマのトランザクションテーブル」](#)
[セクション8.11.4「メタデータのロック」](#)

XA START

[セクション27.12.7.1「events_transactions_currentテーブル」](#)
[セクション13.3.8.1「XAトランザクションSQLステートメント」](#)
[セクション13.3.8.3「XAトランザクションの制約」](#)
[セクション13.3.8.2「XAトランザクションの状態」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション27.12.7「パフォーマンススキーマのトランザクションテーブル」](#)

XA START xid

[セクション13.3.8.1「XAトランザクションSQLステートメント」](#)

ステータス変数の索引

記号 | [A](#) | [B](#) | [C](#) | [D](#) | [E](#) | [F](#) | [G](#) | [H](#) | [I](#) | [K](#) | [L](#) | [M](#) | [N](#) | [O](#) | [P](#) | [Q](#) | [R](#) | [S](#) | [T](#) | [U](#) | [V](#)

記号

[\[index top\]](#)

質問

[セクション5.1.10「サーバーステータス変数」](#)

A

[\[index top\]](#)

Aborted_clients

[セクション5.1.10「サーバーステータス変数」](#)
[セクションB.3.2.9「通信エラーおよび中止された接続」](#)

Aborted_connects

[セクション5.1.10「サーバーステータス変数」](#)
[セクションB.3.2.9「通信エラーおよび中止された接続」](#)

Acl_cache_items_count

[セクション5.1.10「サーバーステータス変数」](#)

Audit_log_current_size

[セクション6.4.5.10「監査ログ参照」](#)

Audit_log_event_max_drop_size

[セクション6.4.5.10「監査ログ参照」](#)

Audit_log_events

[セクション6.4.5.10「監査ログ参照」](#)

Audit_log_events_filtered

[セクション6.4.5.10「監査ログ参照」](#)

Audit_log_events_lost

セクション6.4.5.10「監査ログ参照」

Audit_log_events_written

セクション6.4.5.10「監査ログ参照」

Audit_log_total_size

セクション6.4.5.10「監査ログ参照」

Audit_log_write_waits

セクション6.4.5.10「監査ログ参照」

Authentication_ldap_sasl_supported_methods

セクション6.4.1.7「LDAP プラガブル認証」

セクション5.1.10「サーバーステータス変数」

B

[\[index top\]](#)

Binlog_cache_disk_use

セクション5.1.10「サーバーステータス変数」

セクション17.1.6.4「バイナリロギングのオプションと変数」

セクション5.4.4「バイナリログ」

Binlog_cache_use

セクション5.1.10「サーバーステータス変数」

セクション17.1.6.4「バイナリロギングのオプションと変数」

セクション5.4.4「バイナリログ」

Binlog_stmt_cache_disk_use

セクション5.1.10「サーバーステータス変数」

セクション17.1.6.4「バイナリロギングのオプションと変数」

Binlog_stmt_cache_use

セクション5.1.10「サーバーステータス変数」

セクション17.1.6.4「バイナリロギングのオプションと変数」

Bytes_received

セクション5.1.10「サーバーステータス変数」

セクション5.4.5「スロークエリーログ」

Bytes_sent

セクション5.1.10「サーバーステータス変数」

セクション5.4.5「スロークエリーログ」

C

[\[index top\]](#)

Caching_sha2_password_rsa_public_key

セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」

セクション5.1.10「サーバーステータス変数」

Com_flush

セクション5.1.10「サーバーステータス変数」

Com_restart

セクション13.7.8.8「RESTART ステートメント」

Com_shutdown

セクション13.7.8.9「SHUTDOWN ステートメント」

Com_stmt_reprepare

セクション8.10.3「プリペアドステートメントおよびストアプログラムのキャッシュ」

Compression

セクション5.1.10「サーバーステータス変数」

セクション4.2.8「接続圧縮制御」

Compression_algorithm

セクション5.1.10「サーバーステータス変数」

セクション4.2.8「接続圧縮制御」

Compression_level

セクション5.1.10「サーバーステータス変数」

セクション4.2.8「接続圧縮制御」

Connection_control_delay_generated

セクション6.4.2.2「Connection-Control のシステム変数とステータス変数」

セクション6.4.2.1「Connection-Control プラグインのインストール」

Connection_errors_accept

セクション5.1.10「サーバーステータス変数」

Connection_errors_internal

セクション5.1.10「サーバーステータス変数」

Connection_errors_max_connections

セクション5.1.10「サーバーステータス変数」

セクション5.1.12.1「接続インタフェース」

Connection_errors_peer_address

セクション5.1.10「サーバーステータス変数」

Connection_errors_select

セクション5.1.10「サーバーステータス変数」

Connection_errors_tcpwrap

セクション5.1.10「サーバーステータス変数」

Connection_errors_xxx

セクション5.1.12.3「DNS ルックアップとホストキャッシュ」

セクション5.1.10「サーバーステータス変数」

Connections

セクション5.1.8「サーバーシステム変数」

セクション5.1.10「サーバーステータス変数」

Created_tmp_disk_tables

セクション27.12.6.1「events_statements_current テーブル」

セクション8.4.4「MySQL での内部一時テーブルの使用」

セクション5.1.8「サーバーシステム変数」

セクション5.1.10「サーバーステータス変数」

セクション5.4.5「スロークエリログ」

Created_tmp_files

セクション5.1.10「サーバーステータス変数」

Created_tmp_tables

セクション27.12.6.1「events_statements_current テーブル」

セクション8.4.4「MySQL での内部一時テーブルの使用」

セクション13.7.7.37「SHOW STATUS ステートメント」

セクション5.1.8「サーバーシステム変数」

セクション5.1.10「サーバーステータス変数」

セクション5.4.5「スロークエリログ」

Current_tls_ca

セクション5.1.10「サーバーステータス変数」

セクション6.3.1「暗号化接続を使用するための MySQL の構成」

Current_tls_capath

セクション5.1.10「サーバーステータス変数」

セクション6.3.1「暗号化接続を使用するための MySQL の構成」

Current_tls_cert

セクション5.1.10「サーバーステータス変数」

セクション6.3.1「暗号化接続を使用するための MySQL の構成」

Current_tls_cipher

セクション5.1.10「サーバーステータス変数」

セクション6.3.1「暗号化接続を使用するための MySQL の構成」

Current_tls_ciphersuites

セクション5.1.10「サーバーステータス変数」

セクション6.3.1「暗号化接続を使用するための MySQL の構成」

Current_tls_crl

セクション5.1.10「サーバーステータス変数」

セクション6.3.1「暗号化接続を使用するための MySQL の構成」

Current_tls_crlpath

セクション5.1.10「サーバーステータス変数」

セクション6.3.1「暗号化接続を使用するための MySQL の構成」

Current_tls_key

セクション5.1.10「サーバーステータス変数」

セクション6.3.1「暗号化接続を使用するための MySQL の構成」

Current_tls_version

セクション5.1.10「サーバーステータス変数」

セクション6.3.1「暗号化接続を使用するための MySQL の構成」

D

[\[index top\]](#)

Delayed_errors

[セクション5.1.10 「サーバーステータス変数」](#)

Delayed_insert_threads

[セクション5.1.10 「サーバーステータス変数」](#)

Delayed_writes

[セクション5.1.10 「サーバーステータス変数」](#)

dragnet.Status

[セクション5.1.8 「サーバーシステム変数」](#)

[セクション5.1.10 「サーバーステータス変数」](#)

E

[\[index top\]](#)

Error_log_buffered_bytes

[セクション27.12.19.1 「error_log テーブル」](#)

[セクション5.1.10 「サーバーステータス変数」](#)

Error_log_buffered_events

[セクション27.12.19.1 「error_log テーブル」](#)

[セクション5.1.10 「サーバーステータス変数」](#)

Error_log_expired_events

[セクション27.12.19.1 「error_log テーブル」](#)

[セクション5.1.10 「サーバーステータス変数」](#)

Error_log_latest_write

[セクション27.12.19.1 「error_log テーブル」](#)

[セクション5.1.10 「サーバーステータス変数」](#)

F

[\[index top\]](#)

Firewall_access_denied

[セクション6.4.7.4 「MySQL Enterprise Firewall リファレンス」](#)

Firewall_access_granted

[セクション6.4.7.3 「MySQL Enterprise Firewall の使用」](#)

[セクション6.4.7.4 「MySQL Enterprise Firewall リファレンス」](#)

Firewall_access_suspicious

[セクション6.4.7.4 「MySQL Enterprise Firewall リファレンス」](#)

Firewall_cached_entries

[セクション6.4.7.4 「MySQL Enterprise Firewall リファレンス」](#)

Flush_commands

セクション5.1.10「サーバーステータス変数」

G

[\[index top\]](#)

group_replication_primary_member

セクション18.8「グループレプリケーションシステム変数」
セクション5.1.10「サーバーステータス変数」
[プライマリの検索](#)

H

[\[index top\]](#)

Handler_commit

セクション5.1.10「サーバーステータス変数」

Handler_delete

セクション5.1.10「サーバーステータス変数」

Handler_discover

NDB Cluster ステータス変数

Handler_external_lock

セクション5.1.10「サーバーステータス変数」

Handler_mrr_init

セクション5.1.10「サーバーステータス変数」

Handler_prepare

セクション5.1.10「サーバーステータス変数」

Handler_read_first

セクション5.1.10「サーバーステータス変数」
セクション5.4.5「スロークエリローグ」

Handler_read_key

セクション5.1.10「サーバーステータス変数」
セクション5.4.5「スロークエリローグ」

Handler_read_last

セクション5.1.10「サーバーステータス変数」
セクション5.4.5「スロークエリローグ」

Handler_read_next

セクション8.3.10「インデックス拡張の使用」
セクション5.1.10「サーバーステータス変数」
セクション5.4.5「スロークエリローグ」

Handler_read_prev

セクション5.1.10「サーバーステータス変数」
セクション5.4.5「スロークエリローグ」

Handler_read_rnd

[セクション5.1.10「サーバーステータス変数」](#)

[セクション5.4.5「スロークエリールログ」](#)

Handler_read_rnd_next

[セクション5.1.10「サーバーステータス変数」](#)

[セクション5.4.5「スロークエリールログ」](#)

Handler_rollback

[セクション5.1.10「サーバーステータス変数」](#)

Handler_savepoint

[セクション5.1.10「サーバーステータス変数」](#)

Handler_savepoint_rollback

[セクション5.1.10「サーバーステータス変数」](#)

Handler_update

[セクション5.1.10「サーバーステータス変数」](#)

Handler_write

[セクション5.1.10「サーバーステータス変数」](#)

|

[\[index top\]](#)

Innodb_buffer_pool_bytes_data

[セクション5.1.10「サーバーステータス変数」](#)

Innodb_buffer_pool_bytes_dirty

[セクション5.1.10「サーバーステータス変数」](#)

Innodb_buffer_pool_dump_status

[セクション5.1.10「サーバーステータス変数」](#)

Innodb_buffer_pool_load_status

[セクション5.1.10「サーバーステータス変数」](#)

Innodb_buffer_pool_pages_data

[セクション5.1.10「サーバーステータス変数」](#)

Innodb_buffer_pool_pages_dirty

[セクション5.1.10「サーバーステータス変数」](#)

Innodb_buffer_pool_pages_flushed

[セクション5.1.10「サーバーステータス変数」](#)

Innodb_buffer_pool_pages_free

[セクション5.1.10「サーバーステータス変数」](#)

Innodb_buffer_pool_pages_latched

[セクション5.1.10「サーバーステータス変数」](#)

InnoDB_buffer_pool_pages_misc

セクション5.1.10 「サーバーステータス変数」

InnoDB_buffer_pool_pages_total

セクション5.1.10 「サーバーステータス変数」

InnoDB_buffer_pool_read_ahead

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.8.3.4 「InnoDB バッファープールのプリフェッチ (先読み) の構成」
セクション5.1.10 「サーバーステータス変数」

InnoDB_buffer_pool_read_ahead_evicted

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.8.3.4 「InnoDB バッファープールのプリフェッチ (先読み) の構成」
セクション5.1.10 「サーバーステータス変数」

InnoDB_buffer_pool_read_ahead_rnd

セクション15.8.3.4 「InnoDB バッファープールのプリフェッチ (先読み) の構成」
セクション5.1.10 「サーバーステータス変数」

InnoDB_buffer_pool_read_requests

セクション5.1.10 「サーバーステータス変数」

InnoDB_buffer_pool_reads

セクション5.1.10 「サーバーステータス変数」

InnoDB_buffer_pool_resize_status

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.8.3.1 「InnoDB バッファープールサイズの構成」
セクション5.1.10 「サーバーステータス変数」

InnoDB_buffer_pool_wait_free

セクション5.1.10 「サーバーステータス変数」

InnoDB_buffer_pool_write_requests

セクション5.1.10 「サーバーステータス変数」

InnoDB_data_fsyncs

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション5.1.10 「サーバーステータス変数」

InnoDB_data_pending_fsyncs

セクション5.1.10 「サーバーステータス変数」

InnoDB_data_pending_reads

セクション5.1.10 「サーバーステータス変数」

InnoDB_data_pending_writes

セクション5.1.10 「サーバーステータス変数」

InnoDB_data_read

セクション5.1.10 「サーバーステータス変数」

InnoDB_data_reads

セクション5.1.10「サーバーステータス変数」

InnoDB_data_writes

セクション5.1.10「サーバーステータス変数」

InnoDB_data_written

セクション5.1.10「サーバーステータス変数」

InnoDB_dblwr_pages_written

セクション5.1.10「サーバーステータス変数」

InnoDB_dblwr_writes

セクション5.1.10「サーバーステータス変数」

InnoDB_have_atomic_builtins

セクション5.1.10「サーバーステータス変数」

InnoDB_log_waits

セクション5.1.10「サーバーステータス変数」

InnoDB_log_write_requests

セクション5.1.10「サーバーステータス変数」

InnoDB_log_writes

セクション5.1.10「サーバーステータス変数」

InnoDB_num_open_files

セクション5.1.10「サーバーステータス変数」

InnoDB_os_log_fsyncs

セクション5.1.10「サーバーステータス変数」

InnoDB_os_log_pending_fsyncs

セクション5.1.10「サーバーステータス変数」

InnoDB_os_log_pending_writes

セクション5.1.10「サーバーステータス変数」

InnoDB_os_log_written

セクション15.14「InnoDBの起動オプションおよびシステム変数」

セクション5.1.10「サーバーステータス変数」

InnoDB_page_size

セクション5.1.10「サーバーステータス変数」

InnoDB_pages_created

セクション5.1.10「サーバーステータス変数」

InnoDB_pages_read

セクション5.1.10「サーバーステータス変数」

InnoDB_pages_written

[セクション5.1.10「サーバーステータス変数」](#)

InnoDB_redo_log_enabled

[セクション1.3「MySQL 8.0の新機能」](#)

[セクション15.6.5「redo ログ」](#)

[セクション5.1.10「サーバーステータス変数」](#)

InnoDB_row_lock_current_waits

[セクション5.1.10「サーバーステータス変数」](#)

InnoDB_row_lock_time

[セクション5.1.10「サーバーステータス変数」](#)

InnoDB_row_lock_time_avg

[セクション5.1.10「サーバーステータス変数」](#)

InnoDB_row_lock_time_max

[セクション5.1.10「サーバーステータス変数」](#)

InnoDB_row_lock_waits

[セクション5.1.10「サーバーステータス変数」](#)

InnoDB_rows_deleted

[セクション5.1.10「サーバーステータス変数」](#)

InnoDB_rows_inserted

[セクション5.1.10「サーバーステータス変数」](#)

InnoDB_rows_read

[セクション5.1.10「サーバーステータス変数」](#)

InnoDB_rows_updated

[セクション5.1.10「サーバーステータス変数」](#)

InnoDB_system_rows_deleted

[セクション5.1.10「サーバーステータス変数」](#)

InnoDB_system_rows_inserted

[セクション5.1.10「サーバーステータス変数」](#)

InnoDB_system_rows_read

[セクション5.1.10「サーバーステータス変数」](#)

InnoDB_truncated_status_writes

[セクション5.1.10「サーバーステータス変数」](#)

InnoDB_undo_tablespaces_active

[セクション5.1.10「サーバーステータス変数」](#)

InnoDB_undo_tablespaces_explicit

[セクション5.1.10「サーバーステータス変数」](#)

Innodb_undo_tablespaces_implicit

[セクション5.1.10「サーバーステータス変数」](#)

Innodb_undo_tablespaces_total

[セクション5.1.10「サーバーステータス変数」](#)

K

[\[index top\]](#)

Key_blocks_not_flushed

[セクション5.1.10「サーバーステータス変数」](#)

Key_blocks_unused

[セクション5.1.8「サーバーシステム変数」](#)

[セクション5.1.10「サーバーステータス変数」](#)

Key_blocks_used

[セクション5.1.10「サーバーステータス変数」](#)

Key_read_requests

[セクション5.1.8「サーバーシステム変数」](#)

[セクション5.1.10「サーバーステータス変数」](#)

Key_reads

[セクション5.1.8「サーバーシステム変数」](#)

[セクション5.1.10「サーバーステータス変数」](#)

Key_write_requests

[セクション5.1.8「サーバーシステム変数」](#)

[セクション5.1.10「サーバーステータス変数」](#)

Key_writes

[セクション5.1.8「サーバーシステム変数」](#)

[セクション5.1.10「サーバーステータス変数」](#)

L

[\[index top\]](#)

Last_query_cost

[セクション5.1.10「サーバーステータス変数」](#)

Last_query_partial_plans

[セクション5.1.10「サーバーステータス変数」](#)

Locked_connects

[セクション6.2.19「アカウントロック」](#)

[セクション5.1.10「サーバーステータス変数」](#)

M

[\[index top\]](#)

Max_execution_time_exceeded

セクション5.1.10「サーブステータス変数」

Max_execution_time_set

セクション5.1.10「サーブステータス変数」

Max_execution_time_set_failed

セクション5.1.10「サーブステータス変数」

Max_used_connections

セクション5.1.10「サーブステータス変数」

Max_used_connections_time

セクション5.1.10「サーブステータス変数」

mecab_charset

セクション12.10.9「MeCab フルテキストパーサープラグイン」

セクション5.1.10「サーブステータス変数」

Mysqlx_aborted_clients

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_address

セクション20.5.6.2「Xプラグイン のオプションとシステム変数」

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_bytes_received

セクション20.5.5「Xプラグイン での接続圧縮」

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_bytes_received_compressed_payload

セクション20.5.5「Xプラグイン での接続圧縮」

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_bytes_received_uncompressed_frame

セクション20.5.5「Xプラグイン での接続圧縮」

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_bytes_sent

セクション20.5.5「Xプラグイン での接続圧縮」

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_bytes_sent_compressed_payload

セクション20.5.5「Xプラグイン での接続圧縮」

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_bytes_sent_uncompressed_frame

セクション20.5.5「Xプラグイン での接続圧縮」

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_compression_algorithm

セクション20.5.5「Xプラグイン での接続圧縮」

[セクション20.5.6.3「X プラグイン ステータス変数」](#)

Mysqlx_compression_level

[セクション20.5.5「X プラグイン での接続圧縮」](#)
[セクション20.5.6.3「X プラグイン ステータス変数」](#)

Mysqlx_connection_accept_errors

[セクション20.5.6.3「X プラグイン ステータス変数」](#)

Mysqlx_connection_errors

[セクション20.5.6.3「X プラグイン ステータス変数」](#)

Mysqlx_connections_accepted

[セクション20.5.6.3「X プラグイン ステータス変数」](#)

Mysqlx_connections_closed

[セクション20.5.6.3「X プラグイン ステータス変数」](#)

Mysqlx_connections_rejected

[セクション20.5.6.3「X プラグイン ステータス変数」](#)

Mysqlx_crud_create_view

[セクション20.5.6.3「X プラグイン ステータス変数」](#)

Mysqlx_crud_delete

[セクション20.5.6.3「X プラグイン ステータス変数」](#)

Mysqlx_crud_drop_view

[セクション20.5.6.3「X プラグイン ステータス変数」](#)

Mysqlx_crud_find

[セクション20.5.6.3「X プラグイン ステータス変数」](#)

Mysqlx_crud_insert

[セクション20.5.6.3「X プラグイン ステータス変数」](#)

Mysqlx_crud_modify_view

[セクション20.5.6.3「X プラグイン ステータス変数」](#)

Mysqlx_crud_update

[セクション20.5.6.3「X プラグイン ステータス変数」](#)

Mysqlx_cursor_close

[セクション20.5.6.3「X プラグイン ステータス変数」](#)

Mysqlx_cursor_fetch

[セクション20.5.6.3「X プラグイン ステータス変数」](#)

Mysqlx_cursor_open

[セクション20.5.6.3「X プラグイン ステータス変数」](#)

Mysqlx_errors_sent

[セクション20.5.6.3「X プラグイン ステータス変数」](#)

Mysqlx_expect_close

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_expect_open

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_init_error

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_messages_sent

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_notice_global_sent

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_notice_other_sent

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_notice_warning_sent

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_notified_by_group_replication

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_port

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_prep_deallocate

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_prep_execute

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_prep_prepare

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_rows_sent

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_sessions

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_sessions_accepted

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_sessions_closed

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_sessions_fatal_error

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_sessions_killed

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_sessions_rejected

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_socket

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_ssl_accept_renegotiates

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_ssl_accepts

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_ssl_active

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_ssl_cipher

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_ssl_cipher_list

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_ssl_ctx_verify_depth

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_ssl_ctx_verify_mode

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_ssl_finished_accepts

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_ssl_server_not_after

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_ssl_server_not_before

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_ssl_verify_depth

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_ssl_verify_mode

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_ssl_version

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_stmt_create_collection

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_stmt_create_collection_index

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_stmt_disable_notices

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_stmt_drop_collection

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_stmt_drop_collection_index

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_stmt_enable_notices

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_stmt_ensure_collection

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_stmt_execute_mysqlx

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_stmt_execute_sql

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_stmt_execute_xplugin

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_stmt_get_collection_options

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_stmt_kill_client

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_stmt_list_clients

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_stmt_list_notices

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_stmt_list_objects

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_stmt_modify_collection_options

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_stmt_ping

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_worker_threads

セクション20.5.6.3「Xプラグイン ステータス変数」

Mysqlx_worker_threads_active

セクション20.5.6.3「Xプラグイン ステータス変数」

N

[\[index top\]](#)

Ndb_api_adaptive_send_deferred_count

NDB Cluster ステータス変数

Ndb_api_adaptive_send_deferred_count_replica

セクション23.1.4 「NDB Cluster の新機能」

NDB Cluster ステータス変数

Ndb_api_adaptive_send_deferred_count_session

NDB Cluster ステータス変数

Ndb_api_adaptive_send_deferred_count_slave

セクション23.1.4 「NDB Cluster の新機能」

NDB Cluster ステータス変数

Ndb_api_adaptive_send_forced_count

NDB Cluster ステータス変数

Ndb_api_adaptive_send_forced_count_replica

セクション23.1.4 「NDB Cluster の新機能」

NDB Cluster ステータス変数

Ndb_api_adaptive_send_forced_count_session

NDB Cluster ステータス変数

Ndb_api_adaptive_send_forced_count_slave

セクション23.1.4 「NDB Cluster の新機能」

NDB Cluster ステータス変数

Ndb_api_adaptive_send_unforced_count

NDB Cluster ステータス変数

Ndb_api_adaptive_send_unforced_count_replica

セクション23.1.4 「NDB Cluster の新機能」

NDB Cluster ステータス変数

Ndb_api_adaptive_send_unforced_count_session

NDB Cluster ステータス変数

Ndb_api_adaptive_send_unforced_count_slave

セクション23.1.4 「NDB Cluster の新機能」

NDB Cluster ステータス変数

Ndb_api_bytes_received_count

セクション23.5.13 「NDB API 統計のカウンタと変数」

NDB Cluster ステータス変数

Ndb_api_bytes_received_count_replica

セクション23.1.4 「NDB Cluster の新機能」

NDB Cluster ステータス変数

Ndb_api_bytes_received_count_session

セクション23.5.13 「NDB API 統計のカウンタと変数」

NDB Cluster ステータス変数

Ndb_api_bytes_received_count_slave

セクション23.5.13 「NDB API 統計のカウンタと変数」

セクション23.1.4 「NDB Cluster の新機能」

NDB Cluster ステータス変数

Ndb_api_bytes_sent_count

セクション23.5.13 「NDB API 統計のカウントと変数」
NDB Cluster ステータス変数

Ndb_api_bytes_sent_count_replica

セクション23.1.4 「NDB Cluster の新機能」
NDB Cluster ステータス変数

Ndb_api_bytes_sent_count_session

セクション23.5.13 「NDB API 統計のカウントと変数」
NDB Cluster ステータス変数

Ndb_api_bytes_sent_count_slave

セクション23.5.13 「NDB API 統計のカウントと変数」
セクション23.1.4 「NDB Cluster の新機能」
NDB Cluster ステータス変数

Ndb_api_event_bytes_count

セクション23.5.13 「NDB API 統計のカウントと変数」
NDB Cluster ステータス変数

Ndb_api_event_bytes_count_injector

セクション23.5.13 「NDB API 統計のカウントと変数」
NDB Cluster ステータス変数

Ndb_api_event_data_count

セクション23.5.13 「NDB API 統計のカウントと変数」
NDB Cluster ステータス変数

Ndb_api_event_data_count_injector

セクション23.5.13 「NDB API 統計のカウントと変数」
NDB Cluster ステータス変数

Ndb_api_event_nondata_count

セクション23.5.13 「NDB API 統計のカウントと変数」
NDB Cluster ステータス変数

Ndb_api_event_nondata_count_injector

セクション23.5.13 「NDB API 統計のカウントと変数」
NDB Cluster ステータス変数

Ndb_api_pk_op_count

セクション23.5.13 「NDB API 統計のカウントと変数」
NDB Cluster ステータス変数

Ndb_api_pk_op_count_replica

セクション23.1.4 「NDB Cluster の新機能」
NDB Cluster ステータス変数

Ndb_api_pk_op_count_session

セクション23.5.13 「NDB API 統計のカウントと変数」
NDB Cluster ステータス変数

Ndb_api_pk_op_count_slave

[セクション23.5.13「NDB API 統計のカウンタと変数」](#)

[セクション23.1.4「NDB Cluster の新機能」](#)

NDB Cluster ステータス変数

Ndb_api_pruned_scan_count

[セクション23.5.13「NDB API 統計のカウンタと変数」](#)

NDB Cluster ステータス変数

Ndb_api_pruned_scan_count_replica

[セクション23.1.4「NDB Cluster の新機能」](#)

NDB Cluster ステータス変数

Ndb_api_pruned_scan_count_session

[セクション23.5.13「NDB API 統計のカウンタと変数」](#)

NDB Cluster ステータス変数

Ndb_api_pruned_scan_count_slave

[セクション23.5.13「NDB API 統計のカウンタと変数」](#)

[セクション23.1.4「NDB Cluster の新機能」](#)

NDB Cluster ステータス変数

Ndb_api_range_scan_count

[セクション23.5.13「NDB API 統計のカウンタと変数」](#)

NDB Cluster ステータス変数

Ndb_api_range_scan_count_replica

[セクション23.1.4「NDB Cluster の新機能」](#)

NDB Cluster ステータス変数

Ndb_api_range_scan_count_session

[セクション23.5.13「NDB API 統計のカウンタと変数」](#)

NDB Cluster ステータス変数

Ndb_api_range_scan_count_slave

[セクション23.5.13「NDB API 統計のカウンタと変数」](#)

[セクション23.1.4「NDB Cluster の新機能」](#)

NDB Cluster ステータス変数

Ndb_api_read_row_count

[セクション23.5.13「NDB API 統計のカウンタと変数」](#)

NDB Cluster ステータス変数

Ndb_api_read_row_count_replica

[セクション23.1.4「NDB Cluster の新機能」](#)

NDB Cluster ステータス変数

Ndb_api_read_row_count_session

[セクション23.5.13「NDB API 統計のカウンタと変数」](#)

NDB Cluster ステータス変数

Ndb_api_read_row_count_slave

[セクション23.5.13「NDB API 統計のカウンタと変数」](#)

[セクション23.1.4「NDB Cluster の新機能」](#)

NDB Cluster ステータス変数

Ndb_api_scan_batch_count

セクション23.5.13 「NDB API 統計のカウントと変数」
NDB Cluster ステータス変数

Ndb_api_scan_batch_count_replica

セクション23.1.4 「NDB Cluster の新機能」
NDB Cluster ステータス変数

Ndb_api_scan_batch_count_session

セクション23.5.13 「NDB API 統計のカウントと変数」
NDB Cluster ステータス変数

Ndb_api_scan_batch_count_slave

セクション23.5.13 「NDB API 統計のカウントと変数」
セクション23.1.4 「NDB Cluster の新機能」
NDB Cluster ステータス変数

Ndb_api_table_scan_count

セクション23.5.13 「NDB API 統計のカウントと変数」
NDB Cluster ステータス変数

Ndb_api_table_scan_count_replica

セクション23.1.4 「NDB Cluster の新機能」
NDB Cluster ステータス変数

Ndb_api_table_scan_count_session

セクション23.5.13 「NDB API 統計のカウントと変数」
NDB Cluster ステータス変数

Ndb_api_table_scan_count_slave

セクション23.5.13 「NDB API 統計のカウントと変数」
セクション23.1.4 「NDB Cluster の新機能」
NDB Cluster ステータス変数

Ndb_api_trans_abort_count

セクション23.5.13 「NDB API 統計のカウントと変数」
NDB Cluster ステータス変数

Ndb_api_trans_abort_count_replica

セクション23.1.4 「NDB Cluster の新機能」
NDB Cluster ステータス変数

Ndb_api_trans_abort_count_session

セクション23.5.13 「NDB API 統計のカウントと変数」
NDB Cluster ステータス変数

Ndb_api_trans_abort_count_slave

セクション23.5.13 「NDB API 統計のカウントと変数」
セクション23.1.4 「NDB Cluster の新機能」
NDB Cluster ステータス変数

Ndb_api_trans_close_count

セクション23.5.13 「NDB API 統計のカウントと変数」
NDB Cluster ステータス変数

Ndb_api_trans_close_count_replica

セクション23.1.4 「NDB Cluster の新機能」

NDB Cluster ステータス変数

Ndb_api_trans_close_count_session

セクション23.5.13 「NDB API 統計のカウンタと変数」

NDB Cluster ステータス変数

Ndb_api_trans_close_count_slave

セクション23.5.13 「NDB API 統計のカウンタと変数」

セクション23.1.4 「NDB Cluster の新機能」

NDB Cluster ステータス変数

Ndb_api_trans_commit_count

セクション23.5.13 「NDB API 統計のカウンタと変数」

NDB Cluster ステータス変数

Ndb_api_trans_commit_count_replica

セクション23.1.4 「NDB Cluster の新機能」

NDB Cluster ステータス変数

Ndb_api_trans_commit_count_session

セクション23.5.13 「NDB API 統計のカウンタと変数」

NDB Cluster ステータス変数

Ndb_api_trans_commit_count_slave

セクション23.5.13 「NDB API 統計のカウンタと変数」

セクション23.1.4 「NDB Cluster の新機能」

NDB Cluster ステータス変数

Ndb_api_trans_local_read_row_count

セクション23.5.13 「NDB API 統計のカウンタと変数」

NDB Cluster ステータス変数

Ndb_api_trans_local_read_row_count_replica

セクション23.1.4 「NDB Cluster の新機能」

NDB Cluster ステータス変数

Ndb_api_trans_local_read_row_count_session

セクション23.5.13 「NDB API 統計のカウンタと変数」

NDB Cluster ステータス変数

Ndb_api_trans_local_read_row_count_slave

セクション23.5.13 「NDB API 統計のカウンタと変数」

セクション23.1.4 「NDB Cluster の新機能」

NDB Cluster ステータス変数

Ndb_api_trans_start_count

セクション23.5.13 「NDB API 統計のカウンタと変数」

NDB Cluster ステータス変数

Ndb_api_trans_start_count_replica

セクション23.1.4 「NDB Cluster の新機能」

NDB Cluster ステータス変数

Ndb_api_trans_start_count_session

セクション23.5.13 「NDB API 統計のカウントと変数」
NDB Cluster ステータス変数

Ndb_api_trans_start_count_slave

セクション23.5.13 「NDB API 統計のカウントと変数」
セクション23.1.4 「NDB Cluster の新機能」
NDB Cluster ステータス変数

Ndb_api_uk_op_count

セクション23.5.13 「NDB API 統計のカウントと変数」
NDB Cluster ステータス変数

Ndb_api_uk_op_count_replica

セクション23.1.4 「NDB Cluster の新機能」
NDB Cluster ステータス変数

Ndb_api_uk_op_count_session

セクション23.5.13 「NDB API 統計のカウントと変数」
NDB Cluster ステータス変数

Ndb_api_uk_op_count_slave

セクション23.5.13 「NDB API 統計のカウントと変数」
セクション23.1.4 「NDB Cluster の新機能」
NDB Cluster ステータス変数

Ndb_api_wait_exec_complete_count

セクション23.5.13 「NDB API 統計のカウントと変数」
NDB Cluster ステータス変数

Ndb_api_wait_exec_complete_count_replica

セクション23.1.4 「NDB Cluster の新機能」
NDB Cluster ステータス変数

Ndb_api_wait_exec_complete_count_session

セクション23.5.13 「NDB API 統計のカウントと変数」
NDB Cluster ステータス変数

Ndb_api_wait_exec_complete_count_slave

セクション23.5.13 「NDB API 統計のカウントと変数」
セクション23.1.4 「NDB Cluster の新機能」
NDB Cluster ステータス変数

Ndb_api_wait_meta_request_count

セクション23.5.13 「NDB API 統計のカウントと変数」
NDB Cluster ステータス変数

Ndb_api_wait_meta_request_count_replica

セクション23.1.4 「NDB Cluster の新機能」
NDB Cluster ステータス変数

Ndb_api_wait_meta_request_count_session

セクション23.5.13 「NDB API 統計のカウントと変数」
NDB Cluster ステータス変数

Ndb_api_wait_meta_request_count_slave

[セクション23.5.13 「NDB API 統計のカウンタと変数」](#)

[セクション23.1.4 「NDB Cluster の新機能」](#)

NDB Cluster ステータス変数

Ndb_api_wait_nanos_count

[セクション23.5.13 「NDB API 統計のカウンタと変数」](#)

NDB Cluster ステータス変数

Ndb_api_wait_nanos_count_replica

[セクション23.1.4 「NDB Cluster の新機能」](#)

NDB Cluster ステータス変数

Ndb_api_wait_nanos_count_session

[セクション23.5.13 「NDB API 統計のカウンタと変数」](#)

NDB Cluster ステータス変数

Ndb_api_wait_nanos_count_slave

[セクション23.5.13 「NDB API 統計のカウンタと変数」](#)

[セクション23.1.4 「NDB Cluster の新機能」](#)

NDB Cluster ステータス変数

Ndb_api_wait_scan_result_count

[セクション23.5.13 「NDB API 統計のカウンタと変数」](#)

NDB Cluster ステータス変数

Ndb_api_wait_scan_result_count_replica

[セクション23.1.4 「NDB Cluster の新機能」](#)

NDB Cluster ステータス変数

Ndb_api_wait_scan_result_count_session

[セクション23.5.13 「NDB API 統計のカウンタと変数」](#)

NDB Cluster ステータス変数

Ndb_api_wait_scan_result_count_slave

[セクション23.5.13 「NDB API 統計のカウンタと変数」](#)

[セクション23.1.4 「NDB Cluster の新機能」](#)

NDB Cluster ステータス変数

Ndb_cluster_node_id

NDB Cluster ステータス変数

Ndb_config_from_host

NDB Cluster ステータス変数

Ndb_config_from_port

NDB Cluster ステータス変数

Ndb_config_generation

NDB Cluster ステータス変数

Ndb_conflict_fn_epoch

NDB Cluster ステータス変数

[セクション23.6.11 「NDB Cluster レプリケーションの競合解決」](#)

Ndb_conflict_fn_epoch2

NDB Cluster ステータス変数
セクション23.6.11 「NDB Cluster レプリケーションの競合解決」

Ndb_conflict_fn_epoch2_trans

NDB Cluster ステータス変数
セクション23.6.11 「NDB Cluster レプリケーションの競合解決」

Ndb_conflict_fn_epoch_trans

NDB Cluster ステータス変数
セクション23.6.11 「NDB Cluster レプリケーションの競合解決」

Ndb_conflict_fn_max

NDB Cluster ステータス変数
セクション23.6.11 「NDB Cluster レプリケーションの競合解決」

Ndb_conflict_fn_max_del_win

NDB Cluster ステータス変数

Ndb_conflict_fn_old

NDB Cluster ステータス変数
セクション23.6.11 「NDB Cluster レプリケーションの競合解決」

Ndb_conflict_last_conflict_epoch

NDB Cluster ステータス変数

Ndb_conflict_last_stable_epoch

NDB Cluster ステータス変数

Ndb_conflict_reflected_op_discard_count

NDB Cluster ステータス変数
セクション23.6.11 「NDB Cluster レプリケーションの競合解決」

Ndb_conflict_reflected_op_prepare_count

NDB Cluster ステータス変数
セクション23.6.11 「NDB Cluster レプリケーションの競合解決」

Ndb_conflict_refresh_op_count

NDB Cluster ステータス変数

Ndb_conflict_trans_conflict_commit_count

NDB Cluster ステータス変数

Ndb_conflict_trans_detect_iter_count

NDB Cluster ステータス変数

Ndb_conflict_trans_reject_count

NDB Cluster ステータス変数

Ndb_conflict_trans_row_conflict_count

NDB Cluster ステータス変数

Ndb_conflict_trans_row_reject_count

NDB Cluster ステータス変数

セクション23.6.11 「NDB Cluster レプリケーションの競合解決」

Ndb_epoch_delete_delete_count

NDB Cluster ステータス変数

Ndb_execute_count

NDB Cluster ステータス変数

Ndb_last_commit_epoch_server

NDB Cluster ステータス変数

Ndb_last_commit_epoch_session

NDB Cluster ステータス変数

Ndb_metadata_detected_count

セクション23.1.4 「NDB Cluster の新機能」

NDB Cluster ステータス変数

セクション27.12.12 「パフォーマンススキーマ NDB Cluster テーブル」

Ndb_metadata_excluded_count

セクション23.1.4 「NDB Cluster の新機能」

NDB Cluster ステータス変数

セクション27.12.12 「パフォーマンススキーマ NDB Cluster テーブル」

Ndb_metadata_synced_count

セクション23.1.4 「NDB Cluster の新機能」

NDB Cluster ステータス変数

セクション27.12.12 「パフォーマンススキーマ NDB Cluster テーブル」

Ndb_number_of_data_nodes

NDB Cluster ステータス変数

Ndb_pruned_scan_count

NDB Cluster ステータス変数

Ndb_pushed_queries_defined

NDB Cluster システム変数

NDB Cluster ステータス変数

Ndb_pushed_queries_dropped

NDB Cluster システム変数

NDB Cluster ステータス変数

Ndb_pushed_queries_executed

NDB Cluster システム変数

NDB Cluster ステータス変数

Ndb_pushed_reads

NDB Cluster システム変数

NDB Cluster ステータス変数

Ndb_replica_max_replicated_epoch

セクション23.1.4 「NDB Cluster の新機能」

NDB Cluster ステータス変数

Ndb_scan_count

NDB Cluster ステータス変数

Ndb_slave_max_replicated_epoch

セクション23.1.4 「NDB Cluster の新機能」
NDB Cluster ステータス変数

Ndb_system_name

NDB Cluster ステータス変数
セクション23.3.3.8 「システムの定義」

Ndb_trans_hint_count_session

NDB Cluster ステータス変数

Not_flushed_delayed_rows

セクション5.1.10 「サーバーステータス変数」

O

[\[index top\]](#)

Ongoing_anonymous_gtid_violating_transaction_count

セクション5.1.10 「サーバーステータス変数」

Ongoing_anonymous_transaction_count

セクション5.1.10 「サーバーステータス変数」

Ongoing_automatic_gtid_violating_transaction_count

セクション5.1.10 「サーバーステータス変数」

Open_files

セクション5.1.10 「サーバーステータス変数」

Open_streams

セクション5.1.10 「サーバーステータス変数」

Open_table_definitions

セクション5.1.10 「サーバーステータス変数」

Open_tables

セクション5.1.10 「サーバーステータス変数」

Opened_files

セクション5.1.10 「サーバーステータス変数」

Opened_table_definitions

セクション5.1.10 「サーバーステータス変数」

Opened_tables

セクション8.4.3.1 「MySQL でのテーブルのオープンとクローズの方法」
セクション5.1.8 「サーバーシステム変数」
セクション5.1.10 「サーバーステータス変数」

P

[\[index top\]](#)

Performance_schema_accounts_lost

[セクション27.16 「パフォーマンススキーマステータス変数」](#)

Performance_schema_cond_classes_lost

[セクション27.16 「パフォーマンススキーマステータス変数」](#)

Performance_schema_cond_instances_lost

[セクション27.16 「パフォーマンススキーマステータス変数」](#)

Performance_schema_digest_lost

[セクション27.15 「パフォーマンススキーマシステム変数」](#)

[セクション27.16 「パフォーマンススキーマステータス変数」](#)

Performance_schema_file_classes_lost

[セクション27.16 「パフォーマンススキーマステータス変数」](#)

Performance_schema_file_handles_lost

[セクション27.16 「パフォーマンススキーマステータス変数」](#)

Performance_schema_file_instances_lost

[セクション27.16 「パフォーマンススキーマステータス変数」](#)

Performance_schema_hosts_lost

[セクション27.16 「パフォーマンススキーマステータス変数」](#)

Performance_schema_index_stat_lost

[セクション27.15 「パフォーマンススキーマシステム変数」](#)

[セクション27.16 「パフォーマンススキーマステータス変数」](#)

Performance_schema_locker_lost

[セクション27.16 「パフォーマンススキーマステータス変数」](#)

Performance_schema_memory_classes_lost

[セクション27.16 「パフォーマンススキーマステータス変数」](#)

Performance_schema_metadata_lock_lost

[セクション27.15 「パフォーマンススキーマシステム変数」](#)

[セクション27.16 「パフォーマンススキーマステータス変数」](#)

Performance_schema_mutex_classes_lost

[セクション27.7 「パフォーマンススキーマステータスモニタリング」](#)

[セクション27.16 「パフォーマンススキーマステータス変数」](#)

Performance_schema_mutex_instances_lost

[セクション27.7 「パフォーマンススキーマステータスモニタリング」](#)

[セクション27.16 「パフォーマンススキーマステータス変数」](#)

Performance_schema_nested_statement_lost

[セクション27.15 「パフォーマンススキーマシステム変数」](#)

[セクション27.16 「パフォーマンススキーマステータス変数」](#)

Performance_schema_prepared_statements_lost

セクション27.12.6.4「prepared_statements_instances テーブル」

セクション27.15「パフォーマンススキーマシステム変数」

セクション27.16「パフォーマンススキーマステータス変数」

Performance_schema_program_lost

セクション27.15「パフォーマンススキーマシステム変数」

セクション27.16「パフォーマンススキーマステータス変数」

Performance_schema_rwlock_classes_lost

セクション27.16「パフォーマンススキーマステータス変数」

Performance_schema_rwlock_instances_lost

セクション27.16「パフォーマンススキーマステータス変数」

Performance_schema_session_connect_attrs_longest_seen

セクション27.16「パフォーマンススキーマステータス変数」

セクション27.12.9「パフォーマンススキーマ接続属性テーブル」

Performance_schema_session_connect_attrs_lost

セクション27.15「パフォーマンススキーマシステム変数」

セクション27.16「パフォーマンススキーマステータス変数」

セクション27.12.9「パフォーマンススキーマ接続属性テーブル」

Performance_schema_socket_classes_lost

セクション27.16「パフォーマンススキーマステータス変数」

Performance_schema_socket_instances_lost

セクション27.16「パフォーマンススキーマステータス変数」

Performance_schema_stage_classes_lost

セクション27.16「パフォーマンススキーマステータス変数」

Performance_schema_statement_classes_lost

セクション27.16「パフォーマンススキーマステータス変数」

Performance_schema_table_handles_lost

セクション27.15「パフォーマンススキーマシステム変数」

セクション27.16「パフォーマンススキーマステータス変数」

Performance_schema_table_instances_lost

セクション27.16「パフォーマンススキーマステータス変数」

Performance_schema_table_lock_stat_lost

セクション27.15「パフォーマンススキーマシステム変数」

セクション27.16「パフォーマンススキーマステータス変数」

Performance_schema_thread_classes_lost

セクション27.16「パフォーマンススキーマステータス変数」

Performance_schema_thread_instances_lost

セクション27.15「パフォーマンススキーマシステム変数」

セクション27.12.14「パフォーマンススキーマシステム変数テーブル」

セクション27.16「パフォーマンススキーマステータス変数」

[セクション12.22 「パフォーマンススキーマ関数」](#)

Performance_schema_users_lost

[セクション27.16 「パフォーマンススキーマステータス変数」](#)

Prepared_stmt_count

[セクション5.1.10 「サーバーステータス変数」](#)

Q

[\[index top\]](#)

Queries

[セクション5.1.10 「サーバーステータス変数」](#)

Questions

[セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」](#)

[セクション5.1.10 「サーバーステータス変数」](#)

R

[\[index top\]](#)

Rewriter_number_loaded_rules

[リライタのクエリーリライトプラグインステータス変数](#)

Rewriter_number_reloads

[リライタのクエリーリライトプラグインステータス変数](#)

Rewriter_number_rewritten_queries

[リライタのクエリーリライトプラグインステータス変数](#)

Rewriter_reload_error

[リライタのクエリーリライトプラグインステータス変数](#)

[リライタのクエリーリライトプロシージャおよび関数](#)

[セクション5.6.4.2 「リライタクエリーリライトプラグインの使用」](#)

[リライタクエリーリライトプラグインルールテーブル](#)

Rpl_semi_sync_master_clients

[セクション5.1.10 「サーバーステータス変数」](#)

[セクション17.4.10.3 「準同期レプリケーションモニタリング」](#)

[セクション17.4.10.1 「準同期レプリケーション管理インタフェース」](#)

Rpl_semi_sync_master_net_avg_wait_time

[セクション5.1.10 「サーバーステータス変数」](#)

Rpl_semi_sync_master_net_wait_time

[セクション5.1.10 「サーバーステータス変数」](#)

Rpl_semi_sync_master_net_waits

[セクション5.1.10 「サーバーステータス変数」](#)

Rpl_semi_sync_master_no_times

[セクション5.1.10 「サーバーステータス変数」](#)

Rpl_semi_sync_master_no_tx

セクション5.1.10「サーバーステータス変数」
セクション17.4.10.3「準同期レプリケーションモニタリング」
セクション17.4.10.1「準同期レプリケーション管理インタフェース」

Rpl_semi_sync_master_status

セクション5.1.10「サーバーステータス変数」
セクション17.4.10.3「準同期レプリケーションモニタリング」
セクション17.4.10.1「準同期レプリケーション管理インタフェース」

Rpl_semi_sync_master_timefunc_failures

セクション5.1.10「サーバーステータス変数」

Rpl_semi_sync_master_tx_avg_wait_time

セクション5.1.10「サーバーステータス変数」

Rpl_semi_sync_master_tx_wait_time

セクション5.1.10「サーバーステータス変数」

Rpl_semi_sync_master_tx_waits

セクション5.1.10「サーバーステータス変数」

Rpl_semi_sync_master_wait_pos_backtraverse

セクション5.1.10「サーバーステータス変数」

Rpl_semi_sync_master_wait_sessions

セクション5.1.10「サーバーステータス変数」

Rpl_semi_sync_master_yes_tx

セクション5.1.10「サーバーステータス変数」
セクション17.4.10.3「準同期レプリケーションモニタリング」
セクション17.4.10.1「準同期レプリケーション管理インタフェース」

Rpl_semi_sync_slave_status

セクション5.1.10「サーバーステータス変数」
セクション17.4.10.3「準同期レプリケーションモニタリング」
セクション17.4.10.1「準同期レプリケーション管理インタフェース」

Rsa_public_key

セクション6.4.1.3「SHA-256 プラガブル認証」
セクション5.1.10「サーバーステータス変数」

S

[\[index top\]](#)

Secondary_engine_execution_count

セクション5.1.10「サーバーステータス変数」

Select_full_join

セクション27.12.6.1「events_statements_current テーブル」
セクション5.1.10「サーバーステータス変数」

Select_full_range_join

セクション27.12.6.1「events_statements_current テーブル」

[セクション5.1.10「サーバーステータス変数」](#)

Select_range

[セクション27.12.6.1「events_statements_current テーブル」](#)

[セクション5.1.10「サーバーステータス変数」](#)

Select_range_check

[セクション27.12.6.1「events_statements_current テーブル」](#)

[セクション5.1.10「サーバーステータス変数」](#)

Select_scan

[セクション27.12.6.1「events_statements_current テーブル」](#)

[セクション5.1.10「サーバーステータス変数」](#)

Slave_open_temp_tables

[セクション13.4.2.1「CHANGE MASTER TO ステートメント」](#)

[セクション13.4.2.3「CHANGE REPLICATION SOURCE TO ステートメント」](#)

[セクション13.4.2.9「STOP REPLICATION | SLAVE ステートメント」](#)

[セクション5.1.10「サーバーステータス変数」](#)

[セクション17.5.1.31「レプリケーションと一時テーブル」](#)

Slave_rows_last_search_algorithm_used

[セクション5.1.10「サーバーステータス変数」](#)

Slow_launch_threads

[セクション5.1.8「サーバーシステム変数」](#)

[セクション5.1.10「サーバーステータス変数」](#)

Slow_queries

[セクション5.1.8「サーバーシステム変数」](#)

[セクション5.1.10「サーバーステータス変数」](#)

Sort_merge_passes

[セクション27.12.6.1「events_statements_current テーブル」](#)

[セクション8.2.1.16「ORDER BY の最適化」](#)

[セクション5.1.8「サーバーシステム変数」](#)

[セクション5.1.10「サーバーステータス変数」](#)

[セクション5.4.5「スロークエリログ」](#)

Sort_range

[セクション27.12.6.1「events_statements_current テーブル」](#)

[セクション5.1.10「サーバーステータス変数」](#)

[セクション5.4.5「スロークエリログ」](#)

Sort_rows

[セクション27.12.6.1「events_statements_current テーブル」](#)

[セクション5.1.10「サーバーステータス変数」](#)

[セクション5.4.5「スロークエリログ」](#)

Sort_scan

[セクション27.12.6.1「events_statements_current テーブル」](#)

[セクション5.1.10「サーバーステータス変数」](#)

[セクション5.4.5「スロークエリログ」](#)

Ssl_accept_renegotiates

[セクション5.1.10「サーバーステータス変数」](#)

Ssl_accepts

セクション5.1.10「サーバーステータス変数」

Ssl_callback_cache_hits

セクション5.1.10「サーバーステータス変数」

Ssl_cipher

セクション5.1.10「サーバーステータス変数」

セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」

セクション6.3.1「暗号化接続を使用するための MySQL の構成」

Ssl_cipher_list

セクション5.1.10「サーバーステータス変数」

セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」

Ssl_client_connects

セクション5.1.10「サーバーステータス変数」

Ssl_connect_renegotiates

セクション5.1.10「サーバーステータス変数」

Ssl_ctx_verify_depth

セクション5.1.10「サーバーステータス変数」

Ssl_ctx_verify_mode

セクション5.1.10「サーバーステータス変数」

Ssl_default_timeout

セクション5.1.10「サーバーステータス変数」

Ssl_finished_accepts

セクション5.1.10「サーバーステータス変数」

Ssl_finished_connects

セクション5.1.10「サーバーステータス変数」

Ssl_server_not_after

セクション5.1.10「サーバーステータス変数」

Ssl_server_not_before

セクション5.1.10「サーバーステータス変数」

Ssl_session_cache_hits

セクション5.1.10「サーバーステータス変数」

Ssl_session_cache_misses

セクション5.1.10「サーバーステータス変数」

Ssl_session_cache_mode

セクション5.1.10「サーバーステータス変数」

Ssl_session_cache_overflows

セクション5.1.10「サーバーステータス変数」

Ssl_session_cache_size

[セクション5.1.10「サーバーステータス変数」](#)

Ssl_session_cache_timeouts

[セクション5.1.10「サーバーステータス変数」](#)

Ssl_sessions_reused

[セクション5.1.10「サーバーステータス変数」](#)

Ssl_used_session_cache_entries

[セクション5.1.10「サーバーステータス変数」](#)

Ssl_verify_depth

[セクション5.1.10「サーバーステータス変数」](#)

Ssl_verify_mode

[セクション5.1.10「サーバーステータス変数」](#)

Ssl_version

[セクション5.1.10「サーバーステータス変数」](#)

[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#)

T

[\[index top\]](#)

Table_locks_immediate

[セクション5.1.10「サーバーステータス変数」](#)

[セクション8.11.1「内部ロック方法」](#)

Table_locks_waited

[セクション5.1.10「サーバーステータス変数」](#)

[セクション8.11.1「内部ロック方法」](#)

Table_open_cache_hits

[セクション5.1.10「サーバーステータス変数」](#)

Table_open_cache_misses

[セクション5.1.10「サーバーステータス変数」](#)

Table_open_cache_overflows

[セクション5.1.10「サーバーステータス変数」](#)

Tc_log_max_pages_used

[セクション5.1.10「サーバーステータス変数」](#)

Tc_log_page_size

[セクション5.1.10「サーバーステータス変数」](#)

Tc_log_page_waits

[セクション5.1.10「サーバーステータス変数」](#)

Threads_cached

[セクション5.1.10「サーバーステータス変数」](#)

[セクション5.1.12.1「接続インタフェース」](#)

Threads_connected

[セクション5.1.10「サーバーステータス変数」](#)

Threads_created

[セクション5.1.8「サーバーシステム変数」](#)
[セクション5.1.10「サーバーステータス変数」](#)
[セクション5.1.12.1「接続インタフェース」](#)

Threads_running

[セクションA.15「MySQL 8.0 FAQ: MySQL Enterprise Thread Pool」](#)
[セクション5.1.10「サーバーステータス変数」](#)

U

[\[index top\]](#)

Uptime

[セクション4.5.2「mysqldadmin — A MySQL Server 管理プログラム」](#)
[セクション5.1.10「サーバーステータス変数」](#)

Uptime_since_flush_status

[セクション5.1.10「サーバーステータス変数」](#)

V

[\[index top\]](#)

validate_password.dictionary_file_last_parsed

[セクション6.4.3.2「パスワード検証オプションおよび変数」](#)

validate_password.dictionary_file_words_count

[セクション6.4.3.2「パスワード検証オプションおよび変数」](#)

validate_password_dictionary_file_last_parsed

[セクション6.4.3.2「パスワード検証オプションおよび変数」](#)

validate_password_dictionary_file_words_count

[セクション6.4.3.2「パスワード検証オプションおよび変数」](#)

システム変数の索引

[A](#) | [B](#) | [C](#) | [D](#) | [E](#) | [F](#) | [G](#) | [H](#) | [I](#) | [J](#) | [K](#) | [L](#) | [M](#) | [N](#) | [O](#) | [P](#) | [Q](#) | [R](#) | [S](#) | [T](#) | [U](#) | [V](#) | [W](#)

A

[\[index top\]](#)

activate_all_roles_on_login

[セクション13.7.1.11「SET ROLE ステートメント」](#)
[セクション13.7.7.21「SHOW GRANTS ステートメント」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション17.3.3「レプリケーション権限チェック」](#)
[セクション6.2.10「ロールの使用」](#)

admin_address

セクション5.1.8「サーバーシステム変数」
セクション5.1.14「ネットワークネームスペースのサポート」
分散リカバリエンドポイントのアドレスの選択
セクション5.1.12.2「管理接続管理」

admin_port

セクション18.8「グループレプリケーションシステム変数」
セクション5.1.8「サーバーシステム変数」
分散リカバリエンドポイントのアドレスの選択
セクション5.1.12.2「管理接続管理」

admin_ssl_ca

セクション5.1.8「サーバーシステム変数」
セクション5.1.12.2「管理接続管理」

admin_ssl_capath

セクション5.1.8「サーバーシステム変数」

admin_ssl_cert

セクション5.1.7「サーバーコマンドオプション」
セクション5.1.8「サーバーシステム変数」

admin_ssl_cipher

セクション5.1.8「サーバーシステム変数」

admin_ssl_crl

セクション5.1.8「サーバーシステム変数」

admin_ssl_crlpath

セクション5.1.8「サーバーシステム変数」

admin_ssl_key

セクション5.1.7「サーバーコマンドオプション」
セクション5.1.8「サーバーシステム変数」

admin_tls_ciphersuites

セクション5.1.8「サーバーシステム変数」

admin_tls_version

セクション5.1.8「サーバーシステム変数」

audit_log_buffer_size

セクション6.4.5.5「監査ロギング特性の構成」
セクション6.4.5.10「監査ログ参照」

audit_log_compression

セクション6.4.5.5「監査ロギング特性の構成」
セクション6.4.5.10「監査ログ参照」

audit_log_connection_policy

セクション6.4.5.9「レガシーモード監査ログのフィルタリング」
セクション6.4.5.8「監査ログフィルタ定義の書込み」

[セクション6.4.5.10「監査ログ参照」](#)

audit_log_current_session

[セクション6.4.5.10「監査ログ参照」](#)

audit_log_encryption

[セクション6.4.5.5「監査ロギング特性の構成」](#)

[セクション6.4.5.10「監査ログ参照」](#)

audit_log_exclude_accounts

[セクション6.4.5.9「レガシーモード監査ログのフィルタリング」](#)

[セクション6.4.5.8「監査ログフィルタ定義の書込み」](#)

[セクション6.4.5.10「監査ログ参照」](#)

audit_log_file

[セクション2.5.6.2「DockerでのMySQL Serverのデプロイに関するその他のトピック」](#)

[セクション6.4.5「MySQL Enterprise Audit」](#)

[セクション6.4.5.3「MySQL Enterprise Auditのセキュリティに関する考慮事項」](#)

[セクション6.4.5.5「監査ロギング特性の構成」](#)

[セクション6.4.5.6「監査ログファイルの読み取り」](#)

[セクション6.4.5.10「監査ログ参照」](#)

audit_log_filter_id

[セクション6.4.5.7「監査ログのフィルタリング」](#)

[セクション6.4.5.8「監査ログフィルタ定義の書込み」](#)

[セクション6.4.5.10「監査ログ参照」](#)

audit_log_flush

[セクション6.4.5.5「監査ロギング特性の構成」](#)

[セクション6.4.5.10「監査ログ参照」](#)

audit_log_format

[セクション6.4.5「MySQL Enterprise Audit」](#)

[セクション6.4.6「監査メッセージコンポーネント」](#)

[セクション6.4.5.5「監査ロギング特性の構成」](#)

[セクション6.4.5.4「監査ログファイル形式」](#)

[セクション6.4.5.10「監査ログ参照」](#)

audit_log_include_accounts

[セクション6.4.5.9「レガシーモード監査ログのフィルタリング」](#)

[セクション6.4.5.8「監査ログフィルタ定義の書込み」](#)

[セクション6.4.5.10「監査ログ参照」](#)

audit_log_password_history_keep_days

[セクション6.4.5.5「監査ロギング特性の構成」](#)

[セクション6.4.5.10「監査ログ参照」](#)

audit_log_policy

[セクション5.1.9「システム変数の使用」](#)

[セクション6.4.5.9「レガシーモード監査ログのフィルタリング」](#)

[セクション6.4.5.8「監査ログフィルタ定義の書込み」](#)

[セクション6.4.5.10「監査ログ参照」](#)

audit_log_prune_seconds

[セクション6.4.5.5「監査ロギング特性の構成」](#)

[セクション6.4.5.10「監査ログ参照」](#)

audit_log_read_buffer_size

セクション6.4.5.6「監査ログファイルの読取り」

セクション6.4.5.10「監査ログ参照」

audit_log_rotate_on_size

セクション6.4.5.5「監査ロギング特性の構成」

セクション6.4.5.10「監査ログ参照」

audit_log_statement_policy

セクション6.4.5.9「レガシーモード監査ログのフィルタリング」

セクション6.4.5.8「監査ログフィルタ定義の書込み」

セクション6.4.5.10「監査ログ参照」

audit_log_strategy

セクション6.4.5.5「監査ロギング特性の構成」

セクション6.4.5.10「監査ログ参照」

authentication_ldap_sasl_auth_method_name

セクション6.4.1.7「LDAP プラガブル認証」

セクション6.4.1.11「プラガブル認証システム変数」

authentication_ldap_sasl_bind_base_dn

セクション6.4.1.7「LDAP プラガブル認証」

セクション6.4.1.11「プラガブル認証システム変数」

authentication_ldap_sasl_bind_root_dn

セクション6.4.1.7「LDAP プラガブル認証」

セクション6.4.1.11「プラガブル認証システム変数」

authentication_ldap_sasl_bind_root_pwd

セクション6.4.1.7「LDAP プラガブル認証」

セクション6.4.1.11「プラガブル認証システム変数」

authentication_ldap_sasl_ca_path

セクション6.4.1.7「LDAP プラガブル認証」

セクション6.4.1.11「プラガブル認証システム変数」

authentication_ldap_sasl_group_search_attr

セクション6.4.1.7「LDAP プラガブル認証」

セクション6.4.1.11「プラガブル認証システム変数」

authentication_ldap_sasl_group_search_filter

セクション6.4.1.11「プラガブル認証システム変数」

authentication_ldap_sasl_init_pool_size

セクション6.4.1.11「プラガブル認証システム変数」

authentication_ldap_sasl_log_status

セクション6.4.1.11「プラガブル認証システム変数」

authentication_ldap_sasl_max_pool_size

セクション6.4.1.11「プラガブル認証システム変数」

authentication_ldap_sasl_referral

セクション6.4.1.7「LDAP プラガブル認証」

セクション6.4.1.11 「プラグブル認証システム変数」

authentication_ldap_sasl_server_host

セクション6.4.1.7 「LDAP プラグブル認証」

セクション6.4.1.11 「プラグブル認証システム変数」

authentication_ldap_sasl_server_port

セクション6.4.1.7 「LDAP プラグブル認証」

セクション6.4.1.11 「プラグブル認証システム変数」

authentication_ldap_sasl_tls

セクション6.4.1.7 「LDAP プラグブル認証」

セクション6.4.1.11 「プラグブル認証システム変数」

authentication_ldap_sasl_user_search_attr

セクション6.4.1.7 「LDAP プラグブル認証」

セクション6.4.1.11 「プラグブル認証システム変数」

authentication_ldap_simple_auth_method_name

セクション6.4.1.7 「LDAP プラグブル認証」

セクション6.4.1.11 「プラグブル認証システム変数」

authentication_ldap_simple_bind_base_dn

セクション6.4.1.11 「プラグブル認証システム変数」

authentication_ldap_simple_bind_root_dn

セクション6.4.1.11 「プラグブル認証システム変数」

authentication_ldap_simple_bind_root_pwd

セクション6.4.1.11 「プラグブル認証システム変数」

authentication_ldap_simple_ca_path

セクション6.4.1.7 「LDAP プラグブル認証」

セクション6.4.1.11 「プラグブル認証システム変数」

authentication_ldap_simple_group_search_attr

セクション6.4.1.7 「LDAP プラグブル認証」

セクション6.4.1.11 「プラグブル認証システム変数」

authentication_ldap_simple_group_search_filter

セクション6.4.1.11 「プラグブル認証システム変数」

authentication_ldap_simple_init_pool_size

セクション6.4.1.11 「プラグブル認証システム変数」

authentication_ldap_simple_log_status

セクション6.4.1.11 「プラグブル認証システム変数」

authentication_ldap_simple_max_pool_size

セクション6.4.1.11 「プラグブル認証システム変数」

authentication_ldap_simple_referral

セクション6.4.1.7 「LDAP プラグブル認証」

セクション6.4.1.11 「プラグブル認証システム変数」

authentication_ldap_simple_server_host

セクション6.4.1.11「プラグブル認証システム変数」

authentication_ldap_simple_server_port

セクション6.4.1.11「プラグブル認証システム変数」

authentication_ldap_simple_tls

セクション6.4.1.7「LDAP プラグブル認証」

セクション6.4.1.11「プラグブル認証システム変数」

authentication_ldap_simple_user_search_attr

セクション6.4.1.7「LDAP プラグブル認証」

セクション6.4.1.11「プラグブル認証システム変数」

authentication_windows_log_level

セクション6.4.1.6「Windows プラグブル認証」

セクション5.1.8「サーバーシステム変数」

authentication_windows_use_principal_name

セクション6.4.1.6「Windows プラグブル認証」

セクション5.1.8「サーバーシステム変数」

auto_generate_certs

セクション6.3.3.1「MySQL を使用した SSL および RSA 証明書とキーの作成」

セクション5.1.8「サーバーシステム変数」

auto_increment_increment

セクション3.6.9「AUTO_INCREMENT の使用」

セクション15.6.1.6「InnoDB での AUTO_INCREMENT 処理」

セクションA.1「MySQL 8.0 FAQ: 全般」

セクション23.4.13「ndb_import — NDB への CSV データのインポート」

セクション18.10「よくある質問」

セクション18.8「グループレプリケーションシステム変数」

セクション17.5.1.39「レプリケーションと変数」

セクション17.1.6.2「レプリケーションソースのオプションと変数」

セクション5.4.4.3「混合形式のバイナリロギング形式」

auto_increment_offset

セクション3.6.9「AUTO_INCREMENT の使用」

セクション15.6.1.6「InnoDB での AUTO_INCREMENT 処理」

セクションA.1「MySQL 8.0 FAQ: 全般」

セクション23.4.13「ndb_import — NDB への CSV データのインポート」

セクション18.10「よくある質問」

セクション18.8「グループレプリケーションシステム変数」

セクション17.5.1.39「レプリケーションと変数」

セクション17.1.6.2「レプリケーションソースのオプションと変数」

セクション5.4.4.3「混合形式のバイナリロギング形式」

AUTOCOMMIT

セクション17.5.1.35「レプリケーションとトランザクション」

autocommit

セクション13.1.10「ALTER TABLESPACE ステートメント」

セクション13.2.2「DELETE ステートメント」

セクション17.1.3.7「GTID ベースレプリケーションの制約」

セクション26.51.29「INFORMATION_SCHEMA INNODB_TRX テーブル」

セクション15.7.3 「InnoDB のさまざまな SQL ステートメントで設定されたロック」
セクション15.7 「InnoDB のロックおよびトランザクションモデル」
セクション8.5.3 「InnoDB の読み取り専用トランザクションの最適化」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.6.1.4 「InnoDB テーブルの移動またはコピー」
セクション13.3.6 「LOCK TABLES および UNLOCK TABLES ステートメント」
セクション15.6.1.5 「MyISAM から InnoDB へのテーブルの変換」
NDB Cluster システム変数
セクション13.3.1 「START TRANSACTION、COMMIT および ROLLBACK ステートメント」
セクション5.1.8 「サーバーシステム変数」
セクション5.6.3.3 「スレッドプール操作」
セクション15.7.5.2 「デッドロック検出」
セクション15.7.2.1 「トランザクション分離レベル」
セクション27.12.7 「パフォーマンススキーマのトランザクションテーブル」
セクション15.8.9 「ページ構成」
セクション17.5.1.35 「レプリケーションとトランザクション」
セクション17.5.1.31 「レプリケーションと一時テーブル」
セクション15.6.3.3 「一般テーブルスペース」
セクション15.7.2.2 「自動コミット、コミットおよびロールバック」
セクション15.7.2.4 「読み取りのロック」

automatic_sp_privileges

セクション13.1.7 「ALTER PROCEDURE ステートメント」
セクション13.1.17 「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」
セクション5.1.7 「サーバーコマンドオプション」
セクション5.1.8 「サーバーシステム変数」
セクション25.2.2 「ストアードルーチンと MySQL 権限」

avoid_temporal_upgrade

セクション13.7.3.2 「CHECK TABLE ステートメント」
セクション13.7.3.5 「REPAIR TABLE ステートメント」
セクション5.1.8 「サーバーシステム変数」

B

[[index top](#)]

back_log

セクション5.1.8 「サーバーシステム変数」

basedir

セクション13.7.4.4 「INSTALL PLUGIN ステートメント」
セクション2.4.3 「MySQL 起動デーモンのインストールおよび使用」
セクション5.1.7 「サーバーコマンドオプション」
セクション5.1.8 「サーバーシステム変数」

big_tables

セクション8.4.4 「MySQL での内部一時テーブルの使用」
セクション5.1.8 「サーバーシステム変数」

bind_address

セクション5.8 「1 つのマシン上での複数の MySQL インスタンスの実行」
セクションB.3.2.2 「[ローカルの] MySQL サーバーに接続できません」
セクション18.4.5 「IPv6 および IPv6 と IPv4 の混合グループのサポート」
セクション5.1.13 「IPv6 サポート」
セクション5.1.13.3 「IPv6 ローカルホストアドレスを使用した接続」
セクション5.1.13.2 「IPv6 接続を許可するための MySQL Server の構成」
セクション5.1.13.4 「IPv6 非ローカルホストアドレスを使用した接続」

セクション6.2.21「MySQL への接続の問題のトラブルシューティング」
セクション4.3.4「mysqld_multi — 複数の MySQL サーバーの管理」
セクション23.5.14.41「ndbinfo processes テーブル」
セクション20.5.6.2「X プラグイン のオプションとシステム変数」
セクション18.5.1「グループレプリケーション IP アドレスの権限」
セクション5.1.8「サーバーシステム変数」
セクション5.1.14「ネットワークネームスペースのサポート」
セクション5.1.13.5「ブローカからの IPv6 アドレスの入手」
分散リカバリエンドポイントのアドレスの選択
セクション5.1.9.4「永続的で永続的に制限されないシステム変数」

binlog

セクション4.6.8「mysqlbinlog — バイナリログファイル进行处理するためのユーティリティ」
セクション18.9.1「グループレプリケーションの要件」

binlog_cache_size

セクション5.1.10「サーバーステータス変数」
セクション17.1.6.4「バイナリロギングのオプションと変数」
セクション5.4.4「バイナリログ」

binlog_checksum

MySQL 用語集
セクション18.9.2「グループレプリケーションの制限事項」
セクション17.1.6.4「バイナリロギングのオプションと変数」
セクション5.4.4「バイナリログ」
セクション17.5.1.35「レプリケーションとトランザクション」

binlog_direct_non_transactional_updates

セクション17.1.6.4「バイナリロギングのオプションと変数」
セクション17.5.1.35「レプリケーションとトランザクション」

binlog_encryption

セクション13.1.5「ALTER INSTANCE ステートメント」
セクション6.2.2「MySQL で提供される権限」
セクション4.6.8「mysqlbinlog — バイナリログファイル进行处理するためのユーティリティ」
セクション13.7.7.1「SHOW BINARY LOGS ステートメント」
セクション17.1.6.4「バイナリロギングのオプションと変数」
セクション5.4.4「バイナリログ」
セクション17.3.2「バイナリログファイルとリレーログファイルの暗号化」
セクション17.3.2.3「バイナリログマスターキーのローテーション」
セクション17.3「レプリケーションのセキュリティ」
セクション6.1.3「攻撃者に対する MySQL のセキュアな状態の維持」

binlog_error_action

セクション17.1.3.1「GTID 形式および格納」
セクション17.1.6.5「グローバルトランザクション ID システム変数」
セクション17.1.6.4「バイナリロギングのオプションと変数」
セクション5.4.4「バイナリログ」

binlog_expire_logs_seconds

セクション17.1.3.3「GTID 自動配置」
セクション13.4.1.1「PURGE BINARY LOGS ステートメント」
セクション5.4.6「サーバーログの保守」
セクション17.1.6.4「バイナリロギングのオプションと変数」

binlog_format

セクション16.6「BLACKHOLE ストレージエンジン」

セクション17.1.3.7 「GTID ベースレプリケーションの制約」
セクション17.1.3.2 「GTID ライフサイクル」
セクション15.20.7 「InnoDB memcached プラグインとレプリケーション」
セクション13.2.7 「LOAD DATA ステートメント」
セクションA.14 「MySQL 8.0 FAQ: レプリケーション」
セクション6.2.2 「MySQL で提供される権限」
セクション17.5.1.22 「mysql システムスキーマのレプリケーション」
セクション5.4.4.4 「mysql データベーステーブルへの変更に対するロギング形式」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」
セクション17.1.6.3 「Replica Server のオプションと変数」
セクション13.3.8.3 「XA トランザクションの制約」
セクション12.24 「その他の関数」
セクション5.1.9.1 「システム変数権限」
セクション25.7 「ストアードプログラムバイナリロギング」
セクション5.4.5 「スロークエリーログ」
セクション17.2.5.2 「テーブルレベルレプリケーションオプションの評価」
セクション17.2.5.1 「データベースレベルレプリケーションオプションおよびバイナリロギングオプションの評価」
セクション11.6 「データ型デフォルト値」
セクション15.7.2.1 「トランザクション分離レベル」
セクション17.2.1.3 「バイナリロギングでの安全および安全でないステートメントの判断」
セクション17.1.6.4 「バイナリロギングのオプションと変数」
セクション17.3.2.1 「バイナリログの暗号化の範囲」
セクション5.4.4.2 「バイナリログ形式の設定」
セクション24.3.5 「パーティションに関する情報を取得する」
セクション17.3.3.1 「レプリケーション PRIVILEGE_CHECKS_USER アカウントの権限」
セクション17.5.1.2 「レプリケーションと BLACKHOLE テーブル」
セクション17.5.1.19 「レプリケーションと LOAD DATA」
セクション17.5.1.21 「レプリケーションと MEMORY テーブル」
セクション17.5.1.35 「レプリケーションとトランザクション」
セクション17.5.1.31 「レプリケーションと一時テーブル」
セクション17.5.3 「レプリケーションセットアップをアップグレードする」
セクション17.2.5.3 「レプリケーションフィルタリングオプション間の相互作用」
セクション17.2.1 「レプリケーション形式」
セクション17.3.3 「レプリケーション権限チェック」
セクション12.15 「ロック関数」
セクション5.4.3 「一般クエリーログ」
セクション6.2.3 「付与テーブル」
セクション12.16 「情報関数」
セクション12.6.2 「数学関数」
セクション12.7 「日付および時間関数」
セクション5.4.4.3 「混合形式のバイナリロギング形式」
セクション17.2.1.2 「行ベースロギングおよびレプリケーションの使用」

binlog_group_commit_sync_delay

セクション17.1.6.4 「バイナリロギングのオプションと変数」

binlog_group_commit_sync_no_delay_count

セクション17.1.6.4 「バイナリロギングのオプションと変数」

binlog_gtid_simple_recovery

セクション17.1.3.2 「GTID ライフサイクル」

セクション17.1.6.5 「グローバルトランザクション ID システム変数」

binlog_max_flush_queue_time

セクション17.1.6.4 「バイナリロギングのオプションと変数」

binlog_order_commits

セクション17.1.6.4 「バイナリロギングのオプションと変数」

binlog_rotate_encryption_master_key_at_startup

[セクション6.2.2「MySQL で提供される権限」](#)
[セクション17.1.6.4「バイナリロギングのオプションと変数」](#)
[セクション17.3.2.3「バイナリログマスターキーのローテーション」](#)

binlog_row_event_max_size

[セクション17.1.6.4「バイナリロギングのオプションと変数」](#)
[セクション5.4.4.2「バイナリログ形式の設定」](#)

binlog_row_image

[セクション8.5.8「InnoDB ディスク I/O の最適化」](#)
[セクション17.2.1.1「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」](#)
[セクション17.1.6.4「バイナリロギングのオプションと変数」](#)
[セクション13.1.20.10「非表示カラム」](#)

binlog_row_metadata

[セクション17.1.6.4「バイナリロギングのオプションと変数」](#)

binlog_row_value_options

[セクション11.5「JSON データ型」](#)
[セクション17.5.1.17「JSON ドキュメントのレプリケーション」](#)
[セクション1.3「MySQL 8.0 の新機能」](#)
[セクション17.1.6.4「バイナリロギングのオプションと変数」](#)

binlog_rows_query_log_events

[セクション4.6.8「mysqlbinlog — バイナリログファイル进行处理するためのユーティリティー」](#)
[セクション4.6.8.2「mysqlbinlog 行イベントの表示」](#)
[セクション17.2.1.1「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」](#)
[セクション17.4.3「行ベースのレプリケーションの監視」](#)

binlog_stmt_cache_size

[セクション5.1.10「サーバーステータス変数」](#)
[セクション17.1.6.4「バイナリロギングのオプションと変数」](#)

binlog_transaction_compression

[セクション27.12.11.12「binary_log_transaction_compression_stats テーブル」](#)
[セクション13.4.2.1「CHANGE MASTER TO ステートメント」](#)
[セクション13.4.2.3「CHANGE REPLICATION SOURCE TO ステートメント」](#)
[セクション4.6.8「mysqlbinlog — バイナリログファイル进行处理するためのユーティリティー」](#)
[セクション17.1.6.3「Replica Server のオプションと変数」](#)
[セクション17.1.6.4「バイナリロギングのオプションと変数」](#)
[セクション5.4.4.5「バイナリログトランザクション圧縮」](#)
[セクション18.6.3「メッセージ圧縮」](#)
圧縮トランザクションペイロードと非圧縮トランザクションペイロードの組合せ

binlog_transaction_compression_level_zstd

[セクション17.1.6.4「バイナリロギングのオプションと変数」](#)
[セクション5.4.4.5「バイナリログトランザクション圧縮」](#)

binlog_transaction_dependency_history_size

[セクション8.12.3.1「MySQL のメモリーの使用方法」](#)
[セクション17.1.6.4「バイナリロギングのオプションと変数」](#)

binlog_transaction_dependency_tracking

[セクション17.1.6.3「Replica Server のオプションと変数」](#)

セクション18.9.1「グループレプリケーションの要件」
セクション17.1.6.4「バイナリロギングのオプションと変数」

block_encryption_mode

セクション5.1.8「サーバーシステム変数」
セクション12.14「暗号化関数と圧縮関数」

bulk_insert_buffer_size

セクション8.2.5.1「INSERT ステートメントの最適化」
セクション16.2.1「MyISAM 起動オプション」
セクション5.1.8「サーバーシステム変数」

C

[\[index top\]](#)

caching_sha

セクション6.3.3.1「MySQL を使用した SSL および RSA 証明書とキーの作成」
セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」
セクション5.1.8「サーバーシステム変数」
セクション5.1.10「サーバステータス変数」

character_set_client

セクション26.14「INFORMATION_SCHEMA EVENTS テーブル」
セクション26.30「INFORMATION_SCHEMA ROUTINES テーブル」
セクション26.45「INFORMATION_SCHEMA TRIGGERS テーブル」
セクション26.48「INFORMATION_SCHEMA VIEWS テーブル」
セクション13.2.7「LOAD DATA ステートメント」
セクションA.11「MySQL 8.0 FAQ: MySQL の中国語、日本語、および韓国語の文字セット」
セクション13.7.6.2「SET CHARACTER SET ステートメント」
セクション13.7.6.3「SET NAMES ステートメント」
セクション13.7.7「SHOW CREATE EVENT ステートメント」
セクション13.7.7.9「SHOW CREATE PROCEDURE ステートメント」
セクション13.7.7.11「SHOW CREATE TRIGGER ステートメント」
セクション13.7.7.13「SHOW CREATE VIEW ステートメント」
セクション13.7.7.18「SHOW EVENTS ステートメント」
セクション13.7.7.28「SHOW PROCEDURE STATUS ステートメント」
セクション13.7.7.40「SHOW TRIGGERS ステートメント」
セクション9.6「クエリー属性」
セクション5.1.8「サーバーシステム変数」
セクション5.4.4「バイナリログ」
セクション5.6.4.2「リライタクエリーリライトプラグインの使用」
セクション17.5.1.39「レプリケーションと変数」
セクション10.4「接続文字セットおよび照合順序」
セクション10.15「文字セットの構成」
セクション5.4.4.3「混合形式のバイナリロギング形式」

character_set_connection

セクションA.11「MySQL 8.0 FAQ: MySQL の中国語、日本語、および韓国語の文字セット」
セクション10.16「MySQL Server の口ケールサポート」
セクション13.7.6.2「SET CHARACTER SET ステートメント」
セクション13.7.6.3「SET NAMES ステートメント」
セクション12.11「キャスト関数と演算子」
セクション5.1.8「サーバーシステム変数」
セクション17.5.1.39「レプリケーションと変数」
セクション10.8.4「式での照合の強制性」
セクション12.3「式評価での型変換」
セクション10.4「接続文字セットおよび照合順序」

セクション10.3.8 「文字セットイントロデューサ」
セクション10.2.1 「文字セットレパートリー」
セクション9.1.1 「文字列リテラル」
セクション10.3.6 「文字列リテラルの文字セットおよび照合順序」
セクション12.7 「日付および時間関数」
セクション12.14 「暗号化関数と圧縮関数」
セクション5.4.4.3 「混合形式のバイナリロギング形式」
セクション12.8.3 「関数結果の文字セットと照合順序」

character_set_database

セクション13.1.9 「ALTER TABLE ステートメント」
セクション13.1.17 「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」
セクション13.2.7 「LOAD DATA ステートメント」
セクション2.11.4 「MySQL 8.0 での変更」
セクション23.1.4 「NDB Cluster の新機能」
セクション13.7.6.2 「SET CHARACTER SET ステートメント」
セクション5.1.8 「サーバーシステム変数」
セクション10.3.3 「データベース文字セットおよび照合順序」
セクション17.5.1.39 「レプリケーションと変数」
セクション10.4 「接続文字セットおよび照合順序」
セクション5.4.4.3 「混合形式のバイナリロギング形式」

character_set_filesystem

セクション13.2.7 「LOAD DATA ステートメント」
セクション13.2.10.1 「SELECT ... INTO ステートメント」
セクション5.1.8 「サーバーシステム変数」
セクション12.8 「文字列関数および演算子」

character_set_results

セクションA.11 「MySQL 8.0 FAQ: MySQL の中国語、日本語、および韓国語の文字セット」
セクション13.7.6.2 「SET CHARACTER SET ステートメント」
セクション13.7.6.3 「SET NAMES ステートメント」
セクション10.6 「エラーメッセージ文字セット」
セクション5.1.8 「サーバーシステム変数」
セクション10.2.2 「メタデータ用の UTF-8」
セクション10.4 「接続文字セットおよび照合順序」

character_set_server

セクション2.11.4 「MySQL 8.0 での変更」
セクション5.1.8 「サーバーシステム変数」
セクション10.3.2 「サーバー文字セットおよび照合順序」
セクション10.3.3 「データベース文字セットおよび照合順序」
セクション17.5.1.39 「レプリケーションと変数」
セクション17.5.1.3 「レプリケーションと文字セット」
セクション12.10.4 「全文ストップワード」
セクション10.4 「接続文字セットおよび照合順序」
セクション10.15 「文字セットの構成」
セクション5.4.4.3 「混合形式のバイナリロギング形式」

character_set_system

セクション5.1.8 「サーバーシステム変数」
セクション10.2.2 「メタデータ用の UTF-8」
セクション10.15 「文字セットの構成」

character_sets_dir

セクション10.14.3 「8ビットの文字セットへの単純な照合順序の追加」
セクション10.14.4.1 「LDML 構文を使用した UCA 照合順序の定義」

セクション5.1.8「サーバーシステム変数」

check_proxy_users

セクション5.1.8「サーバーシステム変数」
セクション6.2.18「プロキシユーザー」

clone_autotune_concurrency

セクション5.6.7.9「クローニング操作の監視」
セクション5.6.7.12「クローンシステム変数」

clone_buffer_size

セクション5.6.7.12「クローンシステム変数」

clone_ddl_timeout

セクション5.6.7.12「クローンシステム変数」
セクション5.6.7.8「リモートクローニング操作の失敗処理」

clone_enable_compression

クローニングの前提条件
セクション5.6.7.12「クローンシステム変数」
セクション18.8「グループレプリケーションシステム変数」
分散リカバリの圧縮

clone_max_concurrency

セクション5.6.7.9「クローニング操作の監視」
セクション5.6.7.12「クローンシステム変数」

clone_max_data_bandwidth

セクション27.12.17.2「clone_progress テーブル」
セクション5.6.7.12「クローンシステム変数」

clone_max_network_bandwidth

セクション5.6.7.12「クローンシステム変数」

clone_ssl_ca

クローニングの前提条件
セクション5.6.7.12「クローンシステム変数」
セクション18.8「グループレプリケーションシステム変数」
セクション5.6.7.3「リモートデータのクローニング」
分散リカバリの SSL および認証
セクション18.5.3.2「分散リカバリのための Secure Socket Layer (SSL) 接続」

clone_ssl_cert

クローニングの前提条件
セクション5.6.7.12「クローンシステム変数」
セクション18.8「グループレプリケーションシステム変数」
セクション5.6.7.3「リモートデータのクローニング」
分散リカバリの SSL および認証
セクション18.5.3.2「分散リカバリのための Secure Socket Layer (SSL) 接続」

clone_ssl_key

クローニングの前提条件
セクション5.6.7.12「クローンシステム変数」
セクション18.8「グループレプリケーションシステム変数」
セクション5.6.7.3「リモートデータのクローニング」
分散リカバリの SSL および認証

clone_valid_donor_list

クローニングの前提条件
セクション5.6.7.12「クローンシステム変数」
セクション5.6.7.3「リモートデータのクローニング」

collation_connection

セクション26.14「INFORMATION_SCHEMA EVENTS テーブル」
セクション26.30「INFORMATION_SCHEMA ROUTINES テーブル」
セクション26.45「INFORMATION_SCHEMA TRIGGERS テーブル」
セクション26.48「INFORMATION_SCHEMA VIEWS テーブル」
セクション13.5.1「PREPARE ステートメント」
セクション13.7.6.3「SET NAMES ステートメント」
セクション13.7.7.7「SHOW CREATE EVENT ステートメント」
セクション13.7.7.9「SHOW CREATE PROCEDURE ステートメント」
セクション13.7.7.11「SHOW CREATE TRIGGER ステートメント」
セクション13.7.7.13「SHOW CREATE VIEW ステートメント」
セクション13.7.7.18「SHOW EVENTS ステートメント」
セクション13.7.7.28「SHOW PROCEDURE STATUS ステートメント」
セクション13.7.7.40「SHOW TRIGGERS ステートメント」
セクション12.11「キャスト関数と演算子」
セクション5.1.8「サーバーシステム変数」
セクション5.4.4「バイナリログ」
セクション17.5.1.39「レプリケーションと変数」
セクション10.8.4「式での照合の強制性」
セクション12.3「式評価での型変換」
セクション10.4「接続文字セットおよび照合順序」
セクション10.3.8「文字セットイントロデューサ」
セクション10.3.6「文字列リテラルの文字セットおよび照合順序」
セクション12.7「日付および時間関数」
セクション12.14「暗号化関数と圧縮関数」
セクション5.4.4.3「混合形式のバイナリロギング形式」
セクション12.8.3「関数結果の文字セットと照合順序」

collation_database

セクション13.1.17「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」
セクション2.11.4「MySQL 8.0 での変更」
セクション23.1.4「NDB Cluster の新機能」
セクション5.1.8「サーバーシステム変数」
セクション10.3.3「データベース文字セットおよび照合順序」
セクション5.4.4「バイナリログ」
セクション17.5.1.39「レプリケーションと変数」
セクション10.4「接続文字セットおよび照合順序」
セクション5.4.4.3「混合形式のバイナリロギング形式」

collation_server

セクション2.11.4「MySQL 8.0 での変更」
セクション5.1.8「サーバーシステム変数」
セクション10.3.2「サーバー文字セットおよび照合順序」
セクション10.3.3「データベース文字セットおよび照合順序」
セクション5.4.4「バイナリログ」
セクション17.5.1.39「レプリケーションと変数」
セクション12.10.4「全文ストップワード」
セクション10.4「接続文字セットおよび照合順序」
セクション5.4.4.3「混合形式のバイナリロギング形式」

completion_type

セクション13.3.1「START TRANSACTION、COMMIT および ROLLBACK ステートメント」
セクション5.1.8「サーバーシステム変数」

concurrent_insert

[セクション8.6.1「MyISAM クエリーの最適化」](#)
[セクション5.1.7「サーバーコマンドオプション」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション8.11.1「内部ロック方法」](#)
[セクション8.11.3「同時挿入」](#)

connect_timeout

[セクションB.3.2.3「MySQL サーバーへの接続が失われました」](#)
[セクション20.5.6.2「X プラグイン のオプションとシステム変数」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクションB.3.2.9「通信エラーおよび中止された接続」](#)

connection_control_failed_connections_threshold

[セクション6.4.2.2「Connection-Control のシステム変数とステータス変数」](#)
[セクション6.4.2.1「Connection-Control プラグインのインストール」](#)
[セクション26.53.1「INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS テーブル」](#)

connection_control_max_connection_delay

[セクション6.4.2.2「Connection-Control のシステム変数とステータス変数」](#)
[セクション6.4.2.1「Connection-Control プラグインのインストール」](#)

connection_control_min_connection_delay

[セクション6.4.2.2「Connection-Control のシステム変数とステータス変数」](#)
[セクション6.4.2.1「Connection-Control プラグインのインストール」](#)

core_file

[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)
[セクション1.3「MySQL 8.0 の新機能」](#)
[セクション15.8.3.7「コアファイルからのバッファプールページの除外」](#)
[セクション5.1.8「サーバーシステム変数」](#)

create_admin_listener_thread

[セクション5.1.8「サーバーシステム変数」](#)
[セクション5.1.12.2「管理接続管理」](#)

cte_max_recursion_depth

[セクション13.2.15「WITH \(共通テーブル式\)」](#)
[セクション5.1.8「サーバーシステム変数」](#)

D

[\[index top\]](#)

daemon_memcached_enable_binlog

[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)

daemon_memcached_engine_lib_name

[セクション15.20.3「InnoDB memcached プラグインの設定」](#)
[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)

daemon_memcached_engine_lib_path

[セクション15.20.3「InnoDB memcached プラグインの設定」](#)

[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)

daemon_memcached_option

[セクション15.20.2 「InnoDB memcached のアーキテクチャー」](#)
[セクション15.20.5 「InnoDB memcached プラグインのセキュリティに関する考慮事項」](#)
[セクション15.20.9 「InnoDB memcached プラグインのトラブルシューティング」](#)
[セクション15.20.3 「InnoDB memcached プラグインの設定」](#)
[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)

daemon_memcached_r_batch_size

[セクション15.20.2 「InnoDB memcached のアーキテクチャー」](#)
[セクション15.20.7 「InnoDB memcached プラグインとレプリケーション」](#)
[セクション15.20.6.3 「InnoDB memcached プラグインのパフォーマンスのチューニング」](#)
[セクション15.20.3 「InnoDB memcached プラグインの設定」](#)
[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)
[セクション15.20.6.6 「ベースとなる InnoDB テーブルでの DML および DDL ステートメントの実行」](#)

daemon_memcached_w_batch_size

[セクション15.20.2 「InnoDB memcached のアーキテクチャー」](#)
[セクション15.20.7 「InnoDB memcached プラグインとレプリケーション」](#)
[セクション15.20.6.4 「InnoDB memcached プラグインのトランザクション動作の制御」](#)
[セクション15.20.6.3 「InnoDB memcached プラグインのパフォーマンスのチューニング」](#)
[セクション15.20.3 「InnoDB memcached プラグインの設定」](#)
[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)
[セクション15.20.6.6 「ベースとなる InnoDB テーブルでの DML および DDL ステートメントの実行」](#)

datadir

[セクション26.15 「INFORMATION_SCHEMA FILES テーブル」](#)
[セクション15.18.2 「InnoDB のリカバリ」](#)
[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)
[セクション15.8.1 「InnoDB の起動構成」](#)
[セクション2.3 「Microsoft Windows に MySQL をインストールする」](#)
[セクション2.11.4 「MySQL 8.0 での変更」](#)
[セクション1.3 「MySQL 8.0 の新機能」](#)
MySQL 用語集
[セクション2.4.3 「MySQL 起動デーモンのインストールおよび使用」](#)
[セクション15.6.5 「redo ログ」](#)
[セクション13.1.37 「TRUNCATE TABLE ステートメント」](#)
[セクション15.6.3.4 「undo テーブルスペース」](#)
[セクション18.2.1.2 「グループレプリケーション用のインスタンスの構成」](#)
[セクション15.6.3.6 「サーバーがオフラインのときのテーブルスペースファイルの移動」](#)
[セクション5.1.7 「サーバーコマンドオプション」](#)
[セクション5.1.8 「サーバーシステム変数」](#)
[セクション15.6.3.3 「一般テーブルスペース」](#)
[セクション15.6.1.2 「外部でのテーブルの作成」](#)

debug

[セクション5.9.4 「DEBUG パッケージ」](#)
[セクション5.1.8 「サーバーシステム変数」](#)

debug_sync

[セクション5.1.8 「サーバーシステム変数」](#)
[セクション27.12.14.2 「パフォーマンススキーマ variables_info テーブル」](#)

default

[セクション15.1.4 「InnoDB を使用したテストおよびベンチマーク」](#)
[セクション18.9.1 「グループレプリケーションの要件」](#)

[セクション16.1「ストレージエンジンの設定」](#)

default_authentication_plugin

[セクション13.7.1.1「ALTER USER ステートメント」](#)
[セクション13.7.1.3「CREATE USER ステートメント」](#)
[セクション2.11.4「MySQL 8.0 での変更」](#)
[セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」](#)
[セクション6.4.1.3「SHA-256 プラガブル認証」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション6.2.17「プラガブル認証」](#)
[セクション6.4.1「認証プラグイン」](#)

default_collation_for_utf

[セクション5.1.8「サーバーシステム変数」](#)

default_password_lifetime

[セクション13.7.1.1「ALTER USER ステートメント」](#)
[セクション13.7.1.3「CREATE USER ステートメント」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション6.2.15「パスワード管理」](#)
[セクション6.2.3「付与テーブル」](#)

default_storage_engine

[セクション13.1.16「CREATE LOGFILE GROUP ステートメント」](#)
[セクション13.1.20「CREATE TABLE ステートメント」](#)
[セクション13.1.21「CREATE TABLESPACE ステートメント」](#)
[セクション13.1.33「DROP TABLESPACE ステートメント」](#)
[セクション24.2.2「LIST パーティショニング」](#)
[セクション24.1「MySQL のパーティショニングの概要」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション16.1「ストレージエンジンの設定」](#)
[セクション5.6.1「プラグインのインストールおよびアンインストール」](#)
[セクション17.5.1.39「レプリケーションと変数」](#)
[セクション15.6.3.3「一般テーブルスペース」](#)
[セクション17.4.4「異なるソースおよびレプリカのストレージエンジンでのレプリケーションの使用」](#)

default_table_encryption

[セクション13.1.2「ALTER DATABASE ステートメント」](#)
[セクション13.1.10「ALTER TABLESPACE ステートメント」](#)
[セクション13.1.12「CREATE DATABASE ステートメント」](#)
[セクション13.1.21「CREATE TABLESPACE ステートメント」](#)
[セクション15.13「InnoDB 保存データ暗号化」](#)
[セクション1.3「MySQL 8.0 の新機能」](#)
[セクション18.8「グループレプリケーションシステム変数」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション5.1.9.1「システム変数権限」](#)
[セクション17.3.3.1「レプリケーション PRIVILEGE_CHECKS_USER アカウントの権限」](#)

default_tmp_storage_engine

[セクション5.1.8「サーバーシステム変数」](#)
[セクション16.1「ストレージエンジンの設定」](#)
[セクション5.6.1「プラグインのインストールおよびアンインストール」](#)

default_week_format

[セクション5.1.8「サーバーシステム変数」](#)
[セクション12.7「日付および時間関数」](#)
[セクション24.6.3「関数に関連するパーティショニング制限」](#)

delay_key_write

[セクション13.1.20「CREATE TABLE ステートメント」](#)
[セクションA.14「MySQL 8.0 FAQ: レプリケーション」](#)
[セクションB.3.3.3「MySQL が繰り返しクラッシュする場合の対処方法」](#)
[セクション5.1.7「サーバーコマンドオプション」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション8.11.5「外部ロック」](#)

delayed_insert_limit

[セクション5.1.8「サーバーシステム変数」](#)

delayed_insert_timeout

[セクション5.1.8「サーバーシステム変数」](#)

delayed_queue_size

[セクション5.1.8「サーバーシステム変数」](#)

disabled_storage_engines

[セクションA.2「MySQL 8.0 FAQ: ストレージエンジン」](#)
[セクション4.4.5「mysql_upgrade — MySQL テーブルのチェックとアップグレード」](#)
[セクション18.9.1「グループレプリケーションの要件」](#)
[セクション18.2.1.2「グループレプリケーション用のインスタンスの構成」](#)
[セクション5.1.7「サーバーコマンドオプション」](#)
[セクション5.1.8「サーバーシステム変数」](#)

disconnect_on_expired_password

[セクション5.1.8「サーバーシステム変数」](#)
[セクション6.2.16「期限切れパスワードのサーバー処理」](#)

div_precision_increment

[セクション5.1.8「サーバーシステム変数」](#)
[セクション12.6.1「算術演算子」](#)

dragnet

[セクション5.5「MySQL のコンポーネント」](#)
[セクション5.5.3「エラーログコンポーネント」](#)
[セクション5.4.2.4「エラーログフィルタリングのタイプ」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション5.1.10「サーバーステータス変数」](#)
[セクション5.1.9「システム変数の使用」](#)
[セクション5.4.2.6「ルールベースのエラーログのフィルタリング \(log_filter_dragnet\)」](#)

E

[\[index top\]](#)

end_markers_in_json

[セクション5.1.8「サーバーシステム変数」](#)

enforce_gtid_consistency

[セクション17.1.3.4「GTID を使用したレプリケーションのセットアップ」](#)
[セクション17.1.4.3「GTID トランザクションのオンラインでの無効化」](#)
[セクション17.1.3.7「GTID ベースレプリケーションの制約」](#)
[セクション18.9.1「グループレプリケーションの要件」](#)
[セクション17.1.6.5「グローバルトランザクション ID システム変数」](#)
[セクション17.5.1.31「レプリケーションと一時テーブル」](#)

[セクション17.1.4.1「レプリケーションモードの概念」](#)

eq_range_index_dive_limit

[セクション8.2.1.2「rangeの最適化」](#)
[セクション5.1.8「サーバーシステム変数」](#)

error_count

[セクション13.6.7.7「MySQLの診断領域」](#)
[セクション13.7.7.17「SHOW ERRORS ステートメント」](#)
[セクションB.2「エラー情報インタフェース」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション13.5「プリペアドステートメント」](#)

event

[セクション15.8.2「読み取り専用操作のInnoDBの構成」](#)

event_scheduler

[セクション25.4.6「イベントスケジューラとMySQL権限」](#)
[セクション25.4.2「イベントスケジューラの構成」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション17.1.2.8「レプリケーション環境へのレプリカの追加」](#)
[既存のデータによるレプリケーションのセットアップ](#)

expire_logs_days

[セクション17.1.6.4「バイナリロギングのオプションと変数」](#)

explicit_defaults_for_timestamp

[セクション13.1.20.8「CREATE TABLE および生成されるカラム」](#)
[セクション11.2.5「TIMESTAMP および DATETIME の自動初期化および更新機能」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション11.6「データ型デフォルト値」](#)
[セクション11.2.1「日時データ型の構文」](#)

external_user

[セクション5.1.8「サーバーシステム変数」](#)
[セクション6.2.18「プロキシユーザー」](#)
[セクション6.4.5.4「監査ログファイル形式」](#)

F

[\[index top\]](#)

flush

[セクション5.1.8「サーバーシステム変数」](#)

flush_time

[セクション5.1.7「サーバーコマンドオプション」](#)
[セクション5.1.8「サーバーシステム変数」](#)

foreign_key_checks

[セクション13.1.9「ALTER TABLE ステートメント」](#)
[セクション13.1.20.5「FOREIGN KEY の制約」](#)
[セクション15.6.1.3「InnoDB テーブルのインポート」](#)
[NDB Cluster システム変数](#)
[セクション15.12.1「オンライン DDL 操作」](#)
[セクション5.1.11「サーバー SQL モード」](#)

[セクション5.1.8「サーバーシステム変数」](#)
[セクション5.4.4「バイナリログ」](#)
[セクション17.5.1.39「レプリケーションと変数」](#)
[セクション5.4.4.3「混合形式のバイナリロギング形式」](#)

ft_boolean_syntax

[セクション12.10.6「MySQLの全文検索の微調整」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション12.10.2「ブール全文検索」](#)

ft_max_word_len

[セクション12.10.6「MySQLの全文検索の微調整」](#)
[セクション12.10.8「ngram全文パーサー」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション12.10.2「ブール全文検索」](#)
[ローデータファイルを使用したデータスナップショットの作成](#)

ft_min_word_len

[セクション12.10.9「MeCabフルテキストパーサープラグイン」](#)
[セクション12.10.6「MySQLの全文検索の微調整」](#)
[セクション12.10.8「ngram全文パーサー」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション12.10.2「ブール全文検索」](#)
[ローデータファイルを使用したデータスナップショットの作成](#)
[セクション12.10.1「自然言語全文検索」](#)

ft_query_expansion_limit

[セクション5.1.8「サーバーシステム変数」](#)

ft_stopword_file

[セクション12.10.6「MySQLの全文検索の微調整」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション12.10.2「ブール全文検索」](#)
[ローデータファイルを使用したデータスナップショットの作成](#)
[セクション12.10.4「全文ストップワード」](#)
[セクション12.10.1「自然言語全文検索」](#)

G

[\[index top\]](#)

general_log

[MySQL用語集](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション5.4.3「一般クエリーログ」](#)
[セクション5.4.1「一般クエリーログおよびスロークエリーログの出力先の選択」](#)

general_log_file

[セクション1.3「MySQL 8.0の新機能」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション5.4.3「一般クエリーログ」](#)
[セクション5.4.1「一般クエリーログおよびスロークエリーログの出力先の選択」](#)

generated_random_password_length

[セクション5.1.8「サーバーシステム変数」](#)
[セクション6.2.15「パスワード管理」](#)

group_concat_max_len

セクション5.1.8「サーバーシステム変数」

セクション12.20.1「集計関数の説明」

group_replication_advertise_recovery_endpoints

セクション18.4.5「IPv6 および IPv6 と IPv4 の混合グループのサポート」

セクション18.8「グループレプリケーションシステム変数」

セクション18.4.3.1「分散リカバリの接続」

分散リカバリエンドポイントのアドレスの選択

group_replication_allow_local_lower_version_join

セクション18.7.1.1「アップグレード中のメンバーバージョン」

セクション18.8「グループレプリケーションシステム変数」

group_replication_auto_increment_increment

セクション18.10「よくある質問」

セクション18.8「グループレプリケーションシステム変数」

セクション17.1.6.2「レプリケーションソースのオプションと変数」

group_replication_autorejoin_tries

セクション18.6.6.3「Auto-Rejoin」

セクション18.6.6.1「Expel タイムアウト」

セクション18.10「よくある質問」

セクション18.8「グループレプリケーションシステム変数」

セクション18.6.6.2「使用できない大多数のタイムアウト」

セクション18.6.6.4「終了処理」

group_replication_bootstrap_group

2番目のインスタンスの追加

セクション18.2.1.5「グループのブートストラップ」

セクション18.8「グループレプリケーションシステム変数」

セクション18.2.1.2「グループレプリケーション用のインスタンスの構成」

セクション18.1.3「マルチプライマリモードとシングルプライマリモード」

group_replication_clone_threshold

クローニングのしきい値

セクション18.8「グループレプリケーションシステム変数」

セクション18.4.3.2「分散リカバリのためのクローニング」

セクション18.4.3.5「分散リカバリの仕組み」

group_replication_communication_debug_options

セクション18.8「グループレプリケーションシステム変数」

group_replication_communication_max_message_size

セクション18.9.2「グループレプリケーションの制限事項」

セクション18.8「グループレプリケーションシステム変数」

セクション18.6.4「メッセージの断片化」

group_replication_components_stop_timeout

セクション18.8「グループレプリケーションシステム変数」

group_replication_compression_threshold

セクション18.9.2「グループレプリケーションの制限事項」

セクション18.8「グループレプリケーションシステム変数」

バイナリログのトランザクション圧縮のモニタリング

セクション18.6.3「メッセージ圧縮」

group_replication_consistency

セクション6.2.2「MySQL で提供される権限」
セクション18.8「グループレプリケーションシステム変数」
セクション18.1.3.1「シングルプライマリモード」
セクション18.4.2.1「トランザクションの一貫性保証の理解」
セクション18.4.2.2「トランザクション一貫性保証の構成」
セクション18.1.3.2「マルチプライマリモード」

group_replication_enforce_update_everywhere_checks

セクション18.4.1.2「グループモードの変更」
セクション18.9.2「グループレプリケーションの制限事項」
セクション18.8「グループレプリケーションシステム変数」
セクション13.4.3.4「グループレプリケーションモードを構成する関数」
セクション18.1.3.1「シングルプライマリモード」
トランザクションチェック

group_replication_exit_state_action

セクション18.6.6.3「Auto-Rejoin」
セクション18.6.6.1「Expel タイムアウト」
セクション18.3.1「グループレプリケーションサーバーの状態」
セクション18.8「グループレプリケーションシステム変数」
セクション18.6.6.2「使用できない大多数のタイムアウト」
セクション18.4.3.4「分散リカバリのフォルトトレランス」
セクション18.6.6.4「終了処理」

group_replication_flow_control_applier_threshold

セクション18.8「グループレプリケーションシステム変数」

group_replication_flow_control_certifier_threshold

セクション18.8「グループレプリケーションシステム変数」

group_replication_flow_control_hold_percent

セクション18.8「グループレプリケーションシステム変数」

group_replication_flow_control_max_commit_quota

セクション18.8「グループレプリケーションシステム変数」

group_replication_flow_control_member_quota_percent

セクション18.8「グループレプリケーションシステム変数」

group_replication_flow_control_min_quota

セクション18.8「グループレプリケーションシステム変数」

group_replication_flow_control_min_recovery_quota

セクション18.8「グループレプリケーションシステム変数」

group_replication_flow_control_mode

セクション18.8「グループレプリケーションシステム変数」

group_replication_flow_control_period

セクション18.3.3「replication_group_member_stats テーブル」
セクション18.8「グループレプリケーションシステム変数」

group_replication_flow_control_release_percent

セクション18.8「グループレプリケーションシステム変数」

group_replication_force_members

セクション18.8「グループレプリケーションシステム変数」
セクション18.4.4「ネットワークパーティション化」

group_replication_group_name

セクション27.12.11.3「replication_asynchronous_connection_failover テーブル」
セクション18.8「グループレプリケーションシステム変数」
セクション18.2.1.2「グループレプリケーション用のインスタンスの構成」
セクション13.4.2.11「ソースリストを構成する関数」
セクション17.4.9「非同期接続フェイルオーバーによるソースの切替え」

group_replication_group_seeds

セクション18.4.5「IPv6 および IPv6 と IPv4 の混合グループのサポート」
セクション18.5.1「グループレプリケーション IP アドレスの権限」
セクション18.2.2「グループレプリケーションのローカルでのデプロイ」
セクション18.8「グループレプリケーションシステム変数」
セクション18.2.1.2「グループレプリケーション用のインスタンスの構成」
セクション18.4.3.1「分散リカバリの接続」

group_replication_gtid_assignment_block_size

セクション18.8「グループレプリケーションシステム変数」

group_replication_ip_allowlist

セクション18.4.5「IPv6 および IPv6 と IPv4 の混合グループのサポート」
セクション18.5.1「グループレプリケーション IP アドレスの権限」
セクション18.8「グループレプリケーションシステム変数」
分散リカバリエンドポイントのアドレスの選択

group_replication_ip_whitelist

セクション18.4.5「IPv6 および IPv6 と IPv4 の混合グループのサポート」
セクション18.5.1「グループレプリケーション IP アドレスの権限」
セクション18.8「グループレプリケーションシステム変数」
分散リカバリエンドポイントのアドレスの選択

group_replication_local_address

セクション18.4.5「IPv6 および IPv6 と IPv4 の混合グループのサポート」
セクション6.7.5.2「MySQL 機能の TCP ポートコンテキストの設定」
セクション27.12.11.10「replication_group_members テーブル」
セクション18.10「よくある質問」
セクション18.5.1「グループレプリケーション IP アドレスの権限」
セクション18.2.2「グループレプリケーションのローカルでのデプロイ」
セクション18.8「グループレプリケーションシステム変数」
セクション18.2.1.2「グループレプリケーション用のインスタンスの構成」
セクション18.4.3.1「分散リカバリの接続」
分散リカバリエンドポイントのアドレスの選択

group_replication_member_expel_timeout

セクション18.6.6.1「Expel タイムアウト」
セクション18.6.5「XCom キャッシュ管理」
セクション18.10「よくある質問」
セクション18.6.5.1「キャッシュサイズの増加」
セクション18.1.4.1「グループメンバーシップ」
セクション18.9.2「グループレプリケーションの制限事項」
セクション18.8「グループレプリケーションシステム変数」
セクション18.6.4「メッセージの断片化」
セクション18.6.6.4「終了処理」
セクション18.6.6「障害検出およびネットワークパーティション化へのレスポンス」

group_replication_member_weight

セクション18.8「グループレプリケーションシステム変数」
プライマリ選択アルゴリズム

group_replication_message_cache_size

セクション18.6.6.1「Expel タイムアウト」
セクション18.6.5「XCom キャッシュ管理」
セクション18.6.5.2「キャッシュサイズの削減」
セクション18.6.5.1「キャッシュサイズの増加」
セクション18.8「グループレプリケーションシステム変数」

group_replication_poll_spin_loops

セクション18.8「グループレプリケーションシステム変数」
セクション18.6.1「グループ通信スレッドの微調整」

group_replication_recovery_complete_at

セクション18.8「グループレプリケーションシステム変数」
セクション18.4.3.3「分散リカバリの構成」

group_replication_recovery_compression_algorithm

セクション18.8「グループレプリケーションシステム変数」
バイナリログのトランザクション圧縮のモニタリング
セクション18.6.3「メッセージ圧縮」
分散リカバリの圧縮
セクション4.2.8「接続圧縮制御」

group_replication_recovery_get_public_key

セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」
セクション6.4.1.3「SHA-256 プラガブル認証」
キャッシュ SHA-2 認証プラグインを使用するレプリケーションユーザー
セクション18.8「グループレプリケーションシステム変数」
分散リカバリの SSL および認証

group_replication_recovery_public_key_path

セクション6.4.1.2「SHA-2 プラガブル認証のキャッシュ」
キャッシュ SHA-2 認証プラグインを使用するレプリケーションユーザー
セクション18.8「グループレプリケーションシステム変数」
分散リカバリの SSL および認証

group_replication_recovery_reconnect_interval

セクション18.8「グループレプリケーションシステム変数」
セクション18.4.3.3「分散リカバリの構成」

group_replication_recovery_retry_count

セクション18.8「グループレプリケーションシステム変数」
セクション18.4.3.4「分散リカバリのフォルトトレランス」
セクション18.4.3.3「分散リカバリの構成」

group_replication_recovery_ssl_ca

クローニングの前提条件
セクション18.8「グループレプリケーションシステム変数」
分散リカバリの SSL および認証
セクション18.5.3.2「分散リカバリのための Secure Socket Layer (SSL) 接続」

group_replication_recovery_ssl_capath

セクション18.8「グループレプリケーションシステム変数」

[セクション18.5.3.2「分散リカバリのための Secure Socket Layer \(SSL\) 接続」](#)

group_replication_recovery_ssl_cert

クローニングの前提条件

[セクション18.8「グループレプリケーションシステム変数」](#)

[分散リカバリの SSL および認証](#)

[セクション18.5.3.2「分散リカバリのための Secure Socket Layer \(SSL\) 接続」](#)

group_replication_recovery_ssl_cipher

[セクション18.8「グループレプリケーションシステム変数」](#)

[セクション18.5.3.2「分散リカバリのための Secure Socket Layer \(SSL\) 接続」](#)

[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#)

group_replication_recovery_ssl_crl

[セクション18.8「グループレプリケーションシステム変数」](#)

[セクション18.5.3.2「分散リカバリのための Secure Socket Layer \(SSL\) 接続」](#)

group_replication_recovery_ssl_crlpath

[セクション18.8「グループレプリケーションシステム変数」](#)

[セクション18.5.3.2「分散リカバリのための Secure Socket Layer \(SSL\) 接続」](#)

group_replication_recovery_ssl_key

クローニングの前提条件

[セクション18.8「グループレプリケーションシステム変数」](#)

[分散リカバリの SSL および認証](#)

[セクション18.5.3.2「分散リカバリのための Secure Socket Layer \(SSL\) 接続」](#)

group_replication_recovery_ssl_verify_server_cert

[セクション18.8「グループレプリケーションシステム変数」](#)

[セクション18.5.3.2「分散リカバリのための Secure Socket Layer \(SSL\) 接続」](#)

group_replication_recovery_tls_ciphersuites

[セクション18.5.2「Secure Socket Layer \(SSL\) を使用したグループ通信接続の保護」](#)

[セクション18.9.2「グループレプリケーションの制限事項」](#)

[セクション18.8「グループレプリケーションシステム変数」](#)

[セクション18.5.3.2「分散リカバリのための Secure Socket Layer \(SSL\) 接続」](#)

[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#)

group_replication_recovery_tls_version

[セクション18.5.2「Secure Socket Layer \(SSL\) を使用したグループ通信接続の保護」](#)

[セクション18.9.2「グループレプリケーションの制限事項」](#)

[セクション18.8「グループレプリケーションシステム変数」](#)

[セクション18.5.3.2「分散リカバリのための Secure Socket Layer \(SSL\) 接続」](#)

[セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」](#)

group_replication_recovery_use_ssl

クローニングの前提条件

[セクション18.8「グループレプリケーションシステム変数」](#)

[分散リカバリの SSL および認証](#)

[セクション18.5.3.2「分散リカバリのための Secure Socket Layer \(SSL\) 接続」](#)

group_replication_recovery_zstd_compression_level

[セクション18.8「グループレプリケーションシステム変数」](#)

[バイナリログのトランザクション圧縮のモニタリング](#)

[セクション18.6.3「メッセージ圧縮」](#)

分散リカバリの圧縮

セクション4.2.8「接続圧縮制御」

group_replication_single_primary_mode

セクション18.9.2「グループレプリケーションの制限事項」

セクション18.8「グループレプリケーションシステム変数」

セクション18.1.3.1「シングルプライマリモード」

セクション18.1.3.2「マルチプライマリモード」

セクション18.1.3「マルチプライマリモードとシングルプライマリモード」

group_replication_ssl_mode

セクション18.5.2「Secure Socket Layer (SSL) を使用したグループ通信接続の保護」

セクション18.8「グループレプリケーションシステム変数」

分散リカバリの SSL および認証

セクション18.5.3.2「分散リカバリのための Secure Socket Layer (SSL) 接続」

group_replication_start_on_boot

セクション18.6.6.1「Expel タイムアウト」

セクション13.4.3.1「START GROUP_REPLICATION ステートメント」

セクション18.10「よくある質問」

クローニングの前提条件

クローニング操作

セクション18.8「グループレプリケーションシステム変数」

セクション18.7.3.2「グループレプリケーションメンバーのアップグレード」

セクション18.2.1.2「グループレプリケーション用のインスタンスの構成」

レプリケーションユーザー資格証明のセキュアな提供

セクション18.2.1.3「分散リカバリのユーザー資格証明」

セクション18.6.6.4「終了処理」

group_replication_tls_source

セクション18.8「グループレプリケーションシステム変数」

group_replication_transaction_size_limit

セクション18.9.2「グループレプリケーションの制限事項」

セクション18.8「グループレプリケーションシステム変数」

セクション18.6.4「メッセージの断片化」

group_replication_unreachable_majority_timeout

セクション18.8「グループレプリケーションシステム変数」

セクション18.6.6.2「使用できない大多数のタイムアウト」

セクション18.6.6.4「終了処理」

gtid_executed

セクション17.1.3.6「GTID のないソースから GTID のあるレプリカへのレプリケーション」

セクション17.1.3.8「GTID を操作するストアドファンクションの例」

セクション17.1.5.2「GTID ベースのレプリケーション用のマルチソースレプリカのプロビジョニング」

セクション17.1.3.7「GTID ベースレプリケーションの制約」

セクション17.1.3.2「GTID ライフサイクル」

セクション17.1.3.1「GTID 形式および格納」

セクション17.1.3.3「GTID 自動配置」

セクション27.12.19.7「log_status テーブル」

セクション2.11.14「MySQL データベースのほかのマシンへのコピー」

セクション4.5.4「mysqldump — データベースバックアッププログラム」

セクション7.4.1「mysqldump による SQL フォーマットでのデータのダンプ」

mysqldump を使用したデータスナップショットの作成

セクション17.1.6.3「Replica Server のオプションと変数」

セクション13.4.1.2「RESET MASTER ステートメント」

[セクション13.7.7.23 「SHOW MASTER STATUS ステートメント」](#)
[セクション13.7.7.35 「SHOW REPLICA | SLAVE STATUS ステートメント」](#)
[クローニングのしきい値](#)
[セクション18.4.6 「グループレプリケーションでの MySQL Enterprise Backup の使用」](#)
[セクション17.1.6.5 「グローバルトランザクション ID システム変数」](#)
[セクション5.1.8 「サーバーシステム変数」](#)
[セクション27.12.11 「パフォーマンススキーマレプリケーションテーブル」](#)
[セクション17.1.3.5 「フェイルオーバーおよびスケールアウトでの GTID の使用」](#)
[セクション17.2.5.4 「レプリケーションチャンネルベースのフィルタ」](#)
[セクション17.1.4.1 「レプリケーションモードの概念」](#)
[セクション5.6.7.6 「レプリケーション用のクローニング」](#)

gtid_executed_compression_period

[セクション17.1.3.1 「GTID 形式および格納」](#)
[セクション17.1.6.5 「グローバルトランザクション ID システム変数」](#)

GTID_MODE

[セクション13.4.2.1 「CHANGE MASTER TO ステートメント」](#)
[セクション13.4.2.3 「CHANGE REPLICATION SOURCE TO ステートメント」](#)
[セクション17.1.3.3 「GTID 自動配置」](#)
[セクション17.2.1.2 「行ベースロギングおよびレプリケーションの使用」](#)

gtid_mode

[セクション13.4.2.1 「CHANGE MASTER TO ステートメント」](#)
[セクション13.4.2.3 「CHANGE REPLICATION SOURCE TO ステートメント」](#)
[セクション27.12.7.1 「events_transactions_current テーブル」](#)
[GTID のあるトランザクションのスキップ](#)
[セクション17.1.3.6 「GTID のないソースから GTID のあるレプリカへのレプリケーション」](#)
[GTID のないトランザクションのスキップ](#)
[セクション17.1.3.4 「GTID を使用したレプリケーションのセットアップ」](#)
[セクション17.1.4.2 「GTID トランザクションのオンラインでの有効化」](#)
[セクション17.1.4.3 「GTID トランザクションのオンラインでの無効化」](#)
[セクション17.1.3.7 「GTID ベースレプリケーションの制約」](#)
[セクション17.1.3.1 「GTID 形式および格納」](#)
[セクション2.11.14 「MySQL データベースのほかのマシンへのコピー」](#)
[セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」](#)
[セクション4.5.4 「mysqldump — データベースバックアッププログラム」](#)
[セクション7.4.1 「mysqldump による SQL フォーマットでのデータのダンプ」](#)
[mysqldump を使用したデータスナップショットの作成](#)
[セクション17.1.6.3 「Replica Server のオプションと変数」](#)
[セクション13.4.1.2 「RESET MASTER ステートメント」](#)
[セクション13.4.2.5 「RESET REPLICA | SLAVE ステートメント」](#)
[SET GLOBAL sql_slave_skip_counter でのトランザクションのスキップ](#)
[セクション2.11.6 「Unix/Linux での MySQL バイナリまたはパッケージベースのインストールのアップグレード」](#)
[セクション18.9.1 「グループレプリケーションの要件」](#)
[セクション17.1.6.5 「グローバルトランザクション ID システム変数」](#)
[セクション12.19 「グローバルトランザクション識別子 \(GTID\) で使用される機能」](#)
[セクション17.1.7.3 「トランザクションのスキップ」](#)
[セクション17.1.3.5 「フェイルオーバーおよびスケールアウトでの GTID の使用」](#)
[セクション17.1.5.3 「マルチソースレプリカへの GTID ベースのソースの追加」](#)
[セクション17.4.2 「レプリカの予期しない停止の処理」](#)
[セクション17.5.3 「レプリケーションセットアップをアップグレードする」](#)
[セクション17.1.4.1 「レプリケーションモードの概念」](#)
[セクション5.6.7.6 「レプリケーション用のクローニング」](#)
[セクション17.4.9 「非同期接続フェイルオーバーによるソースの切替え」](#)

gtid_next

[セクション27.12.7.1 「events_transactions_current テーブル」](#)

[セクション17.1.3.2 「GTID ライフサイクル」](#)
[セクション17.1.3.1 「GTID 形式および格納」](#)
[セクション13.4.2.7 「START REPLICA | SLAVE ステートメント」](#)
[セクション13.4.2.9 「STOP REPLICA | SLAVE ステートメント」](#)
[セクション17.1.6.5 「グローバルトランザクション ID システム変数」](#)
[セクション5.1.10 「サーバーステータス変数」](#)
[セクション17.3.3.1 「レプリケーション PRIVILEGE_CHECKS_USER アカウントの権限」](#)
[セクション17.1.4.1 「レプリケーションモードの概念」](#)

gtid_owned

[セクション17.1.3.2 「GTID ライフサイクル」](#)
[セクション17.1.6.5 「グローバルトランザクション ID システム変数」](#)

gtid_purged

[セクション13.4.2.1 「CHANGE MASTER TO ステートメント」](#)
[セクション13.4.2.3 「CHANGE REPLICATION SOURCE TO ステートメント」](#)
[セクション17.1.3.8 「GTID を操作するストアドファンクションの例」](#)
[セクション17.1.5.2 「GTID ベースのレプリケーション用のマルチソースレプリカのプロビジョニング」](#)
[セクション17.1.3.2 「GTID ライフサイクル」](#)
[セクション17.1.3.1 「GTID 形式および格納」](#)
[セクション17.1.3.3 「GTID 自動配置」](#)
[セクション4.5.4 「mysqldump — データベースバックアッププログラム」](#)
[セクション7.4.1 「mysqldump による SQL フォーマットでのデータのダンプ」](#)
[mysqldump を使用したデータスナップショットの作成](#)
[セクション13.4.1.2 「RESET MASTER ステートメント」](#)
[クローニングのしきい値](#)
[セクション17.1.6.5 「グローバルトランザクション ID システム変数」](#)
[セクション17.1.3.5 「フェイルオーバーおよびスケールアウトでの GTID の使用」](#)
[セクション17.1.4.1 「レプリケーションモードの概念」](#)

H

[\[index top\]](#)

have_compress

[セクション5.1.8 「サーバースystem変数」](#)

have_dynamic_loading

[セクション5.1.8 「サーバースystem変数」](#)
[セクション17.4.10.2 「準同期レプリケーションのインストールと構成」](#)

have_geometry

[セクション5.1.8 「サーバースystem変数」](#)

have_openssl

[セクション5.1.8 「サーバースystem変数」](#)

have_profiling

[セクション5.1.8 「サーバースystem変数」](#)

have_query_cache

[セクション5.1.8 「サーバースystem変数」](#)

have_rtree_keys

[セクション5.1.8 「サーバースystem変数」](#)

have_ssl

セクション2.9.6「SSL ライブラリサポートの構成」
セクション5.1.8「サーバーシステム変数」

have_statement_timeout

セクション5.1.8「サーバーシステム変数」

have_symlink

セクション8.12.2.2「Unix 上の MyISAM へのシンボリックリンクの使用」
セクション5.1.7「サーバーコマンドオプション」
セクション5.1.8「サーバーシステム変数」

histogram_generation_max_mem_size

セクション13.7.3.1「ANALYZE TABLE ステートメント」
セクション5.1.8「サーバーシステム変数」

host_cache_size

セクション5.1.12.3「DNS ルックアップとホストキャッシュ」
セクション27.12.19.5「host_cache テーブル」
セクション5.1.7「サーバーコマンドオプション」
セクション5.1.8「サーバーシステム変数」

hostname

セクション13.7.5「CLONE ステートメント」
セクション27.12.11.10「replication_group_members テーブル」
セクション18.10「よくある質問」
セクション18.8「グループレプリケーションシステム変数」
セクション18.2.1.2「グループレプリケーション用のインスタンスの構成」
セクション5.1.8「サーバーシステム変数」
セクション5.6.7.3「リモートデータのクローニング」
セクション18.4.3.1「分散リカバリの接続」
セクション18.5.3「分散リカバリ接続の保護」

|

[[index top](#)]

identity

セクション5.1.8「サーバーシステム変数」
セクション17.5.1.39「レプリケーションと変数」
セクション5.4.4.3「混合形式のバイナリロギング形式」

immediate_server_version

セクション17.5.2「MySQL バージョン間のレプリケーション互換性」
セクション17.3.3.1「レプリケーション PRIVILEGE_CHECKS_USER アカウントの権限」
セクション17.1.6.2「レプリケーションソースのオプションと変数」

information_schema_stats_expiry

セクション13.7.3.1「ANALYZE TABLE ステートメント」
セクション26.34「INFORMATION_SCHEMA STATISTICS テーブル」
セクション26.38「INFORMATION_SCHEMA TABLES テーブル」
セクション14.5「INFORMATION_SCHEMA とデータディクショナリの統合」
セクション8.2.3「INFORMATION_SCHEMA クエリーの最適化」
セクション15.14「InnoDB の起動オプションおよびシステム変数」
セクション1.3「MySQL 8.0 の新機能」

[セクション5.1.8「サーバーシステム変数」](#)
[セクション14.7「データディクショナリの使用法の違い」](#)

init_connect

[セクション27.12.19.5「host_cache テーブル」](#)
[セクション6.2.2「MySQL で提供される権限」](#)
[セクション17.1.6.3「Replica Server のオプションと変数」](#)
[セクション10.5「アプリケーションの文字セットおよび照合順序の構成」](#)
[セクション5.1.8「サーバーシステム変数」](#)

init_file

[セクション13.1.2「ALTER DATABASE ステートメント」](#)
[セクション16.3「MEMORY ストレージエンジン」](#)
[root のパスワードのリセット: Windows システム](#)
[root パスワードのリセット: Unix および Unix- 類似システム](#)
[セクション5.1.7「サーバーコマンドオプション」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション2.10.1「データディレクトリの初期化」](#)
[セクション27.12.14.2「パフォーマンススキーマ variables_info テーブル」](#)
[セクション27.4「パフォーマンススキーマ実行時構成」](#)
[セクション17.5.1.21「レプリケーションと MEMORY テーブル」](#)
[セクション8.10.2.2「複合キーキャッシュ」](#)

init_slave

[セクション17.1.6.3「Replica Server のオプションと変数」](#)
[セクション17.3.3.1「レプリケーション PRIVILEGE_CHECKS_USER アカウントの権限」](#)

innodb

[セクションA.16「MySQL 8.0 FAQ : InnoDB 変更バッファ」](#)
[セクション2.11.4「MySQL 8.0 での変更」](#)
[セクション15.8.3.7「コアファイルからのバッファプールページの除外」](#)
[セクション15.8.2「読み取り専用操作の InnoDB の構成」](#)

innodb_adaptive_flushing

[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)
[セクション15.8.3.5「バッファプールのフラッシュの構成」](#)

innodb_adaptive_flushing_lwm

[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)
[セクション15.8.3.5「バッファプールのフラッシュの構成」](#)

innodb_adaptive_hash_index

[セクション15.8.4「InnoDB のスレッド並列性の構成」](#)
[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)
[セクション8.5.9「InnoDB 構成変数の最適化」](#)
MySQL 用語集
[セクション13.1.37「TRUNCATE TABLE ステートメント」](#)
[セクション15.5.3「適応型ハッシュインデックス」](#)

innodb_adaptive_hash_index_parts

[セクション26.51.29「INFORMATION_SCHEMA INNODB_TRX テーブル」](#)
[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)
[セクション15.5.3「適応型ハッシュインデックス」](#)

innodb_adaptive_max_sleep_delay

[セクション15.8.4「InnoDB のスレッド並列性の構成」](#)

[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)

innodb_api_bk_commit_interval

[セクション15.20.2 「InnoDB memcached のアーキテクチャー」](#)

[セクション15.20.6.4 「InnoDB memcached プラグインのトランザクション動作の制御」](#)

[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)

innodb_api_disable_rowlock

[セクション15.20.6.4 「InnoDB memcached プラグインのトランザクション動作の制御」](#)

[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)

innodb_api_enable_binlog

[セクション15.20.7 「InnoDB memcached プラグインとレプリケーション」](#)

[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)

innodb_api_enable_mdl

[セクション15.20.2 「InnoDB memcached のアーキテクチャー」](#)

[セクション15.20.6.4 「InnoDB memcached プラグインのトランザクション動作の制御」](#)

[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)

innodb_api_trx_level

[セクション15.20.2 「InnoDB memcached のアーキテクチャー」](#)

[セクション15.20.6.4 「InnoDB memcached プラグインのトランザクション動作の制御」](#)

[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)

innodb_autoextend_increment

[セクション15.6.3.2 「File-Per-Table テーブルスペース」](#)

[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)

[セクション15.8.1 「InnoDB の起動構成」](#)

[MySQL 用語集](#)

[セクション15.6.3.1 「システムテーブルスペース」](#)

innodb_autoinc_lock_mode

[セクション15.6.1.6 「InnoDB での AUTO_INCREMENT 処理」](#)

[セクション15.7.3 「InnoDB のさまざまな SQL ステートメントで設定されたロック」](#)

[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)

[セクション8.5.5 「InnoDB テーブルの一括データロード」](#)

[セクション15.7.1 「InnoDB ロック」](#)

[セクション1.3 「MySQL 8.0 の新機能」](#)

[MySQL 用語集](#)

[セクション12.16 「情報関数」](#)

innodb_background_drop_list_empty

[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)

innodb_buffer_pool_chunk_size

[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)

[セクション15.8.3.1 「InnoDB バッファプールサイズの構成」](#)

innodb_buffer_pool_debug

[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)

innodb_buffer_pool_dump_at_shutdown

[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)

[MySQL 用語集](#)

セクション15.8.3.6「バッファープールの状態の保存と復元」

innodb_buffer_pool_dump_now

セクション15.14「InnoDBの起動オプションおよびシステム変数」

セクション15.8.3.6「バッファープールの状態の保存と復元」

innodb_buffer_pool_dump_pct

セクション15.14「InnoDBの起動オプションおよびシステム変数」

セクション15.8.3.6「バッファープールの状態の保存と復元」

innodb_buffer_pool_filename

セクション15.14「InnoDBの起動オプションおよびシステム変数」

セクション15.8.3.6「バッファープールの状態の保存と復元」

innodb_buffer_pool_in_core_file

セクション15.14「InnoDBの起動オプションおよびシステム変数」

セクション4.3.2「mysqld_safe — MySQL サーバー起動スクリプト」

セクション15.8.3.7「コアファイルからのバッファープールページの除外」

セクション5.1.7「サーバーコマンドオプション」

innodb_buffer_pool_instances

セクション15.20.6.3「InnoDB memcached プラグインのパフォーマンスのチューニング」

セクション15.14「InnoDBの起動オプションおよびシステム変数」

セクション15.8.1「InnoDBの起動構成」

セクション15.8.3.1「InnoDB バッファープールサイズの構成」

セクション8.12.3.1「MySQLのメモリーの使用方法」

MySQL 用語集

セクション15.8.3.5「バッファープールのフラッシュの構成」

セクション15.6.4「二重書き込みバッファ」

セクション15.8.3.2「複数のバッファープールインスタンスの構成」

innodb_buffer_pool_load_abort

セクション15.14「InnoDBの起動オプションおよびシステム変数」

セクション5.1.10「サーバーステータス変数」

innodb_buffer_pool_load_at_startup

セクション15.14「InnoDBの起動オプションおよびシステム変数」

MySQL 用語集

セクション5.1.10「サーバーステータス変数」

セクション15.8.3.6「バッファープールの状態の保存と復元」

innodb_buffer_pool_load_now

セクション15.14「InnoDBの起動オプションおよびシステム変数」

セクション5.1.10「サーバーステータス変数」

セクション15.8.3.6「バッファープールの状態の保存と復元」

innodb_buffer_pool_size

セクション15.20.2「InnoDB memcached のアーキテクチャー」

セクション15.20.6.3「InnoDB memcached プラグインのパフォーマンスのチューニング」

セクション15.14「InnoDBの起動オプションおよびシステム変数」

セクション15.8.1「InnoDBの起動構成」

セクション8.5.8「InnoDB ディスク I/O の最適化」

セクション15.8.3.1「InnoDB バッファープールサイズの構成」

セクション15.6.1.5「MyISAM から InnoDB へのテーブルの変換」

セクション1.3「MySQL 8.0 の新機能」

セクション8.12.3.1「MySQLのメモリーの使用方法」

MySQL 用語集

セクション15.9.1.6 「OLTP ワークロードの圧縮」

セクション5.1.10 「サーバーステータス変数」

セクション15.8.12 「専用 MySQL Server の自動構成の有効化」

セクション15.8.3.2 「複数のバッファプールインスタンスの構成」

innodb_change_buffer_max_size

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

セクションA.16 「MySQL 8.0 FAQ : InnoDB 変更バッファ」

MySQL 用語集

セクション15.5.2 「変更バッファ」

innodb_change_buffering

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

セクション8.5.2 「InnoDB トランザクション管理の最適化」

MySQL 用語集

セクション15.5.2 「変更バッファ」

セクション15.8.2 「読み取り専用操作の InnoDB の構成」

innodb_change_buffering_debug

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

innodb_checkpoint_disabled

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

innodb_checksum_algorithm

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

セクション8.5.8 「InnoDB ディスク I/O の最適化」

MySQL 用語集

innodb_checksums

MySQL 用語集

innodb_cmp_per_index_enabled

セクション26.51.7 「INFORMATION_SCHEMA INNODB_CMP_PER_INDEX および INNODB_CMP_PER_INDEX_RESET テーブル」

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

セクション15.9.1.3 「InnoDB テーブルの圧縮の調整」

セクション15.9.1.4 「実行時の InnoDB テーブル圧縮の監視」

innodb_commit_concurrency

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

innodb_compress_debug

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

innodb_compression_failure_threshold_pct

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

セクション15.9.1.5 「InnoDB テーブルでの圧縮の動作」

セクション15.9.1.3 「InnoDB テーブルの圧縮の調整」

MySQL 用語集

セクション15.9.1.6 「OLTP ワークロードの圧縮」

innodb_compression_level

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

セクション15.9.1.5 「InnoDB テーブルでの圧縮の動作」
セクション15.9.1.3 「InnoDB テーブルの圧縮の調整」
MySQL 用語集
セクション15.9.1.6 「OLTP ワークロードの圧縮」

innodb_compression_pad_pct_max

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.9.1.5 「InnoDB テーブルでの圧縮の動作」
セクション15.9.1.3 「InnoDB テーブルの圧縮の調整」
MySQL 用語集
セクション15.9.1.6 「OLTP ワークロードの圧縮」

innodb_concurrency_tickets

セクション26.51.29 「INFORMATION_SCHEMA INNODB_TRX テーブル」
セクション15.8.4 「InnoDB のスレッド並列性の構成」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション8.5.9 「InnoDB 構成変数の最適化」

innodb_data_file_path

セクション26.15 「INFORMATION_SCHEMA FILES テーブル」
セクション4.6.2 「innochecksum — オフライン InnoDB ファイルチェックサムユーティリティー」
セクション15.21.1 「InnoDB の I/O に関する問題のトラブルシューティング」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.8.1 「InnoDB の起動構成」
セクション15.6.3.1 「システムテーブルスペース」
セクション2.10.1 「データディレクトリの初期化」
セクション15.11.2 「ファイル領域管理」
セクション5.6.7.3 「リモートデータのクローニング」

innodb_data_home_dir

セクション15.21.1 「InnoDB の I/O に関する問題のトラブルシューティング」
セクション15.18.2 「InnoDB のリカバリ」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.8.1 「InnoDB の起動構成」
セクション2.11.4 「MySQL 8.0 での変更」
セクション1.3 「MySQL 8.0 の新機能」
セクション15.6.5 「redo ログ」
セクション13.1.37 「TRUNCATE TABLE ステートメント」
セクション15.6.3.4 「undo テーブルスペース」
セクション18.4.6 「グループレプリケーションでの MySQL Enterprise Backup の使用」
セクション15.6.3.6 「サーバーがオフラインのときのテーブルスペースファイルの移動」
セクション2.10.1 「データディレクトリの初期化」
セクション5.6.7.3 「リモートデータのクローニング」
セクション15.6.3.5 「一時テーブルスペース」
セクション15.6.3.3 「一般テーブルスペース」
セクション15.6.4 「二重書き込みバッファ」
セクション15.6.1.2 「外部でのテーブルの作成」

innodb_ddl_log_crash_reset_debug

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

innodb_deadlock_detect

セクション15.7.5 「InnoDB のデッドロック」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.6.1.5 「MyISAM から InnoDB へのテーブルの変換」
セクション1.3 「MySQL 8.0 の新機能」
MySQL 用語集

[セクション15.7.5.2「デッドロック検出」](#)
[セクション8.11.1「内部ロック方法」](#)

innodb_dedicated_server

[セクション15.14「InnoDBの起動オプションおよびシステム変数」](#)
[セクション1.3「MySQL 8.0の新機能」](#)
[セクション15.8.12「専用MySQL Serverの自動構成の有効化」](#)

innodb_default_row_format

[セクション13.1.20「CREATE TABLE ステートメント」](#)
[セクション15.10「InnoDBの行フォーマット」](#)
[セクション15.14「InnoDBの起動オプションおよびシステム変数」](#)
[セクション15.6.1.3「InnoDBテーブルのインポート」](#)
[セクション15.6.1.1「InnoDBテーブルの作成」](#)
MySQL用語集
[セクション8.4.1「データサイズの最適化」](#)

innodb_directories

[セクション13.1.21「CREATE TABLESPACE ステートメント」](#)
[セクション15.18.2「InnoDBのリカバリ」](#)
[セクション15.14「InnoDBの起動オプションおよびシステム変数」](#)
[セクション2.11.4「MySQL 8.0での変更」](#)
[セクション1.3「MySQL 8.0の新機能」](#)
[セクション15.6.5「redo ログ」](#)
[セクション13.1.37「TRUNCATE TABLE ステートメント」](#)
[セクション15.6.3.4「undo テーブルスペース」](#)
[セクション15.6.3.6「サーバーがオフラインのときのテーブルスペースファイルの移動」](#)
[セクション15.6.3.7「テーブルスペースパス検証の無効化」](#)
[セクション15.6.3.3「一般テーブルスペース」](#)
[セクション15.6.1.2「外部でのテーブルの作成」](#)

innodb_disable_sort_file_cache

[セクション15.14「InnoDBの起動オプションおよびシステム変数」](#)

innodb_doublewrite

[セクション15.20.6.3「InnoDB memcached プラグインのパフォーマンスのチューニング」](#)
[セクション15.2「InnoDB および ACID モデル」](#)
[セクション15.14「InnoDBの起動オプションおよびシステム変数」](#)
[セクション15.8.1「InnoDBの起動構成」](#)
[セクション15.11.1「InnoDB ディスク I/O」](#)
MySQL用語集
[セクション15.6.4「二重書き込みバッファ」](#)

innodb_doublewrite_batch_size

[セクション15.14「InnoDBの起動オプションおよびシステム変数」](#)
[セクション1.3「MySQL 8.0の新機能」](#)
[セクション15.6.4「二重書き込みバッファ」](#)

innodb_doublewrite_dir

[セクション15.14「InnoDBの起動オプションおよびシステム変数」](#)
[セクション15.8.1「InnoDBの起動構成」](#)
[セクション1.3「MySQL 8.0の新機能」](#)
[セクション15.6.4「二重書き込みバッファ」](#)

innodb_doublewrite_files

[セクション15.14「InnoDBの起動オプションおよびシステム変数」](#)

[セクション1.3「MySQL 8.0の新機能」](#)
[セクション15.6.4「二重書き込みバッファ」](#)

innodb_doublewrite_pages

[セクション15.14「InnoDBの起動オプションおよびシステム変数」](#)
[セクション1.3「MySQL 8.0の新機能」](#)
[セクション15.6.4「二重書き込みバッファ」](#)

innodb_extend_and_initialize

[セクション15.14「InnoDBの起動オプションおよびシステム変数」](#)
[セクション15.6.3.8「Linuxでのテーブルスペースの領域割当ての最適化」](#)
[セクション1.3「MySQL 8.0の新機能」](#)

innodb_fast_shutdown

[セクション15.18.2「InnoDBのリカバリ」](#)
[セクション15.14「InnoDBの起動オプションおよびシステム変数」](#)
MySQL用語集
[セクション2.11.6「Unix/LinuxでのMySQLバイナリまたはパッケージベースのインストールのアップグレード」](#)
[セクション2.11.12「アップグレードのトラブルシューティング」](#)
[セクション2.11.5「アップグレード用のインストールの準備」](#)
[セクション5.1.19「サーバーの停止プロセス」](#)

innodb_fil_make_page_dirty_debug

[セクション15.14「InnoDBの起動オプションおよびシステム変数」](#)

innodb_file_per

[セクション13.1.20.2「CREATE TEMPORARY TABLE ステートメント」](#)
[セクション15.14「InnoDBの起動オプションおよびシステム変数」](#)

innodb_file_per_table

[セクション13.1.20.3「CREATE TABLE ... LIKE ステートメント」](#)
[セクション13.1.20.1「CREATE TABLE によって作成されるファイル」](#)
[セクション13.1.20「CREATE TABLE ステートメント」](#)
[セクション13.1.21「CREATE TABLESPACE ステートメント」](#)
[セクション15.6.3.2「File-Per-Table テーブルスペース」](#)
[セクション13.7.8.3「FLUSH ステートメント」](#)
[セクション15.2「InnoDBおよびACIDモデル」](#)
[セクション15.10「InnoDBの行フォーマット」](#)
[セクション15.14「InnoDBの起動オプションおよびシステム変数」](#)
[セクション15.9.1.5「InnoDBテーブルでの圧縮の動作」](#)
[セクション15.6.1.3「InnoDBテーブルのインポート」](#)
[セクション15.1.2「InnoDBテーブルのベストプラクティス」](#)
[セクション15.6.1.1「InnoDBテーブルの作成」](#)
[セクション15.21.3「InnoDBデータディクショナリの操作のトラブルシューティング」](#)
[セクション15.6.1.5「MyISAMからInnoDBへのテーブルの変換」](#)
MySQL用語集
[セクション13.7.3.4「OPTIMIZE TABLE ステートメント」](#)
[セクション15.9.1.7「SQL圧縮構文の警告とエラー」](#)
[セクション15.11.5「TRUNCATE TABLEによるディスク領域の再利用」](#)
[セクション24.6「パーティショニングの制約と制限」](#)
[セクション15.11.2「ファイル領域管理」](#)
[セクション15.9.1.2「圧縮テーブルの作成」](#)
[セクション15.6.1.2「外部でのテーブルの作成」](#)
[セクション17.4.6「異なるレプリカへの異なるデータベースのレプリケート」](#)

innodb_fill_factor

[セクション15.14「InnoDBの起動オプションおよびシステム変数」](#)

セクション15.6.2.2 「InnoDB インデックスの物理構造」
セクション15.6.2.3 「ソートされたインデックス構築」

innodb_flush_log_at_timeout

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.5.4 「ログバッファ」

innodb_flush_log_at_trx_commit

セクション15.20.6.3 「InnoDB memcached プラグインのパフォーマンスのチューニング」
セクション15.2 「InnoDB および ACID モデル」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション8.5.2 「InnoDB トランザクション管理の最適化」
セクション13.1.1 「アトミックデータ定義ステートメントのサポート」
セクション17.1.6.4 「バイナリロギングのオプションと変数」
セクション17.4.2 「レプリカの予期しない停止の処理」
セクション17.5.1.28 「レプリケーションとソースまたはレプリカの停止」
セクション15.5.4 「ログバッファ」

innodb_flush_method

セクション15.6.3.2 「File-Per-Table テーブルスペース」
セクション15.20.6.3 「InnoDB memcached プラグインのパフォーマンスのチューニング」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション8.5.8 「InnoDB ディスク I/O の最適化」
セクション1.3 「MySQL 8.0 の新機能」
セクション5.1.10 「サーバーステータス変数」
セクション15.6.4 「二重書き込みバッファ」
セクション15.8.12 「専用 MySQL Server の自動構成の有効化」

innodb_flush_neighbors

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション8.5.8 「InnoDB ディスク I/O の最適化」
MySQL 用語集
セクション15.8.3.5 「バッファープールのフラッシュの構成」

innodb_flush_sync

セクション15.8.7 「InnoDB I/O Capacity の構成」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」

innodb_flushing_avg_loops

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.8.3.5 「バッファープールのフラッシュの構成」

innodb_force_load_corrupted

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

innodb_force_recovery

セクション13.1.32 「DROP TABLE ステートメント」
セクション15.18.2 「InnoDB のリカバリ」
セクション15.21.2 「InnoDB のリカバリの強制的な実行」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション8.5.2 「InnoDB トランザクション管理の最適化」
セクション2.11.13 「テーブルまたはインデックスの再作成または修復」
セクション1.6 「質問またはバグをレポートする方法」

innodb_fsync_threshold

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

セクション8.5.8 「InnoDB ディスク I/O の最適化」

innodb_ft_aux_table

セクション26.51.13 「INFORMATION_SCHEMA INNOODB_FT_BEING_DELETED テーブル」
セクション26.51.14 「INFORMATION_SCHEMA INNOODB_FT_CONFIG テーブル」
セクション26.51.16 「INFORMATION_SCHEMA INNOODB_FT_DELETED テーブル」
セクション26.51.17 「INFORMATION_SCHEMA INNOODB_FT_INDEX_CACHE テーブル」
セクション26.51.18 「INFORMATION_SCHEMA INNOODB_FT_INDEX_TABLE テーブル」
セクション15.15.4 「InnoDB INFORMATION_SCHEMA FULLTEXT インデックステーブル」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」

innodb_ft_cache_size

セクション26.51.17 「INFORMATION_SCHEMA INNOODB_FT_INDEX_CACHE テーブル」
セクション15.6.2.4 「InnoDB FULLTEXT インデックス」
セクション15.15.4 「InnoDB INFORMATION_SCHEMA FULLTEXT インデックステーブル」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
MySQL 用語集

innodb_ft_enable_diag_print

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

innodb_ft_enable_stopword

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション12.10.6 「MySQL の全文検索の微調整」
セクション12.10.2 「ブール全文検索」
セクション12.10.1 「自然言語全文検索」

innodb_ft_max_token_size

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション12.10.6 「MySQL の全文検索の微調整」
セクション12.10.8 「ngram 全文パーサー」
セクション12.10.2 「ブール全文検索」
セクション12.10.4 「全文ストップワード」

innodb_ft_min_token_size

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション12.10.9 「MeCab フルテキストパーサープラグイン」
セクション12.10.6 「MySQL の全文検索の微調整」
セクション12.10.8 「ngram 全文パーサー」
セクション12.10.2 「ブール全文検索」
セクション12.10.4 「全文ストップワード」
セクション12.10.1 「自然言語全文検索」

innodb_ft_num_word_optimize

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション12.10.6 「MySQL の全文検索の微調整」
セクション13.7.3.4 「OPTIMIZE TABLE ステートメント」

innodb_ft_result_cache_limit

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

innodb_ft_server_stopword_table

セクション26.51.15 「INFORMATION_SCHEMA INNOODB_FT_DEFAULT_STOPWORD テーブル」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション12.10.6 「MySQL の全文検索の微調整」
セクション12.10.2 「ブール全文検索」

[セクション12.10.4 「全文ストップワード」](#)
[セクション12.10.1 「自然言語全文検索」](#)

innodb_ft_sort_pll_degree

[セクション15.6.2.4 「InnoDB FULLTEXT インデックス」](#)
[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)

innodb_ft_total_cache_size

[セクション26.51.17 「INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE テーブル」](#)
[セクション15.6.2.4 「InnoDB FULLTEXT インデックス」](#)
[セクション15.15.4 「InnoDB INFORMATION_SCHEMA FULLTEXT インデックステーブル」](#)
[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)

innodb_ft_user_stopword_table

[セクション26.51.15 「INFORMATION_SCHEMA INNODB_FT_DEFAULT_STOPWORD テーブル」](#)
[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)
[セクション12.10.6 「MySQL の全文検索の微調整」](#)
[セクション12.10.2 「ブール全文検索」](#)
[セクション12.10.4 「全文ストップワード」](#)
[セクション12.10.1 「自然言語全文検索」](#)

innodb_idle_flush_pct

[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)
[セクション8.5.8 「InnoDB ディスク I/O の最適化」](#)
[セクション1.3 「MySQL 8.0 の新機能」](#)
[セクション15.8.3.5 「バッファープールのフラッシュの構成」](#)

innodb_io_capacity

[セクション15.8.7 「InnoDB I/O Capacity の構成」](#)
[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)
[セクション8.5.8 「InnoDB ディスク I/O の最適化」](#)
[セクションA.16 「MySQL 8.0 FAQ : InnoDB 変更バッファ」](#)
[セクション15.8.3.5 「バッファープールのフラッシュの構成」](#)

innodb_io_capacity_max

[セクション15.8.7 「InnoDB I/O Capacity の構成」](#)
[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)
[セクション8.5.8 「InnoDB ディスク I/O の最適化」](#)
[セクション15.8.3.5 「バッファープールのフラッシュの構成」](#)

innodb_limit_optimistic_insert_debug

[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)

innodb_lock_wait_timeout

[セクション15.7.5 「InnoDB のデッドロック」](#)
[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)
[セクション15.6.1.5 「MyISAM から InnoDB へのテーブルの変換」](#)
[セクション1.3 「MySQL 8.0 の新機能」](#)
MySQL 用語集
[セクション17.1.6.3 「Replica Server のオプションと変数」](#)
[セクション15.7.5.2 「デッドロック検出」](#)
[セクション17.5.1.32 「レプリケーション再試行とタイムアウト」](#)
[セクション8.11.1 「内部ロック方法」](#)

innodb_log_buffer_size

[セクション8.5.4 「InnoDB redo ログの最適化」](#)

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.8.1 「InnoDB の起動構成」
セクション1.3 「MySQL 8.0 の新機能」
MySQL 用語集
セクション15.5.4 「ログバッファ」

innodb_log_checkpoint_fuzzy_now

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

innodb_log_checkpoint_now

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

innodb_log_checksums

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

innodb_log_compressed_pages

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション8.5.8 「InnoDB ディスク I/O の最適化」
セクション15.9.1.6 「OLTP ワークロードの圧縮」

innodb_log_file_size

セクション8.5.4 「InnoDB redo ロギングの最適化」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.8.1 「InnoDB の起動構成」
セクション8.5.8 「InnoDB ディスク I/O の最適化」
セクション1.3 「MySQL 8.0 の新機能」
MySQL 用語集
セクション15.6.5 「redo ログ」
セクション5.1.9 「システム変数の使用」
セクション2.10.1 「データディレクトリの初期化」
セクション15.8.3.5 「バッファープールのフラッシュの構成」
セクション15.8.12 「専用 MySQL Server の自動構成の有効化」
セクション15.8.2 「読み取り専用操作の InnoDB の構成」

innodb_log_files_in_group

セクション8.5.4 「InnoDB redo ロギングの最適化」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.8.1 「InnoDB の起動構成」
MySQL 用語集
セクション15.6.5 「redo ログ」
セクション15.8.12 「専用 MySQL Server の自動構成の有効化」

innodb_log_group_home_dir

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.8.1 「InnoDB の起動構成」
MySQL 用語集
セクション15.6.5 「redo ログ」
セクション15.6.3.4 「undo テーブルスペース」
セクション18.4.6 「グループレプリケーションでの MySQL Enterprise Backup の使用」
セクション2.10.1 「データディレクトリの初期化」
セクション5.6.7.3 「リモートデータのクローニング」

innodb_log_spin_cpu_abs_lwm

セクション8.5.4 「InnoDB redo ロギングの最適化」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション1.3 「MySQL 8.0 の新機能」

innodb_log_spin_cpu_pct_hwm

セクション8.5.4 「InnoDB redo ログの最適化」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション1.3 「MySQL 8.0 の新機能」

innodb_log_wait_for_flush_spin_hwm

セクション8.5.4 「InnoDB redo ログの最適化」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション1.3 「MySQL 8.0 の新機能」

innodb_log_write_ahead_size

セクション8.5.4 「InnoDB redo ログの最適化」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」

innodb_log_writer_threads

セクション8.5.4 「InnoDB redo ログの最適化」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」

innodb_lru_scan_depth

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.8.3.5 「バッファープールのフラッシュの構成」

innodb_max_dirty_pages_pct

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション8.5.8 「InnoDB ディスク I/O の最適化」
セクション1.3 「MySQL 8.0 の新機能」
セクション15.8.3.5 「バッファープールのフラッシュの構成」

innodb_max_dirty_pages_pct_lwm

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション1.3 「MySQL 8.0 の新機能」
セクション15.8.3.5 「バッファープールのフラッシュの構成」

innodb_max_purge_lag

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.3 「InnoDB マルチバージョン」
MySQL 用語集
セクション15.8.9 「ページ構成」

innodb_max_purge_lag_delay

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.8.9 「ページ構成」

innodb_max_undo_log_size

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.6.3.4 「undo テーブルスペース」

innodb_merge_threshold_set_all_debug

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

innodb_monitor_disable

セクション26.51.22 「INFORMATION_SCHEMA INNODB_METRICS テーブル」
セクション15.15.6 「InnoDB INFORMATION_SCHEMA メトリックテーブル」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション13.7.7.15 「SHOW ENGINE ステートメント」

innodb_monitor_enable

[セクション26.51.22「INFORMATION_SCHEMA INNODB_METRICS テーブル」](#)
[セクション15.15.6「InnoDB INFORMATION_SCHEMA メトリックテーブル」](#)
[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)
[セクション13.7.7.15「SHOW ENGINE ステートメント」](#)

innodb_monitor_reset

[セクション26.51.22「INFORMATION_SCHEMA INNODB_METRICS テーブル」](#)
[セクション15.15.6「InnoDB INFORMATION_SCHEMA メトリックテーブル」](#)
[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)

innodb_monitor_reset_all

[セクション26.51.22「INFORMATION_SCHEMA INNODB_METRICS テーブル」](#)
[セクション15.15.6「InnoDB INFORMATION_SCHEMA メトリックテーブル」](#)
[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)

innodb_numa_interleave

[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)

innodb_old_blocks_pct

[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)
[MySQL 用語集](#)
[セクション15.8.3.3「バッファプールをスキャンに耐えられるようにする」](#)

innodb_old_blocks_time

[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)
[セクション15.5.1「バッファプール」](#)
[セクション15.8.3.3「バッファプールをスキャンに耐えられるようにする」](#)

innodb_online_alter_log_max_size

[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)
[MySQL 用語集](#)
[セクション15.12.5「オンライン DDL 失敗条件」](#)
[セクション15.12.3「オンライン DDL 領域の要件」](#)

innodb_open_files

[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)
[セクション5.1.8「サーバーシステム変数」](#)

innodb_optimize_fulltext_only

[セクション26.51.18「INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE テーブル」](#)
[セクション15.6.2.4「InnoDB FULLTEXT インデックス」](#)
[セクション15.15.4「InnoDB INFORMATION_SCHEMA FULLTEXT インデックステーブル」](#)
[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)
[セクション12.10.6「MySQL の全文検索の微調整」](#)
[セクション13.7.3.4「OPTIMIZE TABLE ステートメント」](#)

innodb_page_cleaners

[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)
[MySQL 用語集](#)
[セクション15.8.3.5「バッファプールのフラッシュの構成」](#)

innodb_page_size

[セクション13.1.20「CREATE TABLE ステートメント」](#)
[セクション13.1.21「CREATE TABLESPACE ステートメント」](#)
[セクション26.15「INFORMATION_SCHEMA FILES テーブル」](#)

セクション15.20.9 「InnoDB memcached プラグインのトラブルシューティング」
セクション8.5.4 「InnoDB redo ログの最適化」
セクション15.22 「InnoDB の制限」
セクション15.23 「InnoDB の制限および制限事項」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.8.1 「InnoDB の起動構成」
セクション15.6.2.2 「InnoDB インデックスの物理構造」
セクション15.9.1.5 「InnoDB テーブルでの圧縮の動作」
セクション15.8.10.3 「InnoDB テーブルに対する ANALYZE TABLE の複雑さの推定」
セクション15.6.1.3 「InnoDB テーブルのインポート」
セクション8.5.8 「InnoDB ディスク I/O の最適化」
セクション15.9.2 「InnoDB ページ圧縮」
セクション15.6.1.5 「MyISAM から InnoDB へのテーブルの変換」
MySQL 用語集
セクション15.6.3.4 「undo テーブルスペース」
セクション15.8.3.7 「コアファイルからのバッファプールページの除外」
セクション15.6.3.1 「システムテーブルスペース」
セクション8.4.7 「テーブルカラム数と行サイズの制限」
セクション15.6.3.9 「テーブルスペースの AUTOEXTEND_SIZE 構成」
セクション15.9.1.1 「テーブル圧縮の概要」
セクション15.11.2 「ファイル領域管理」
セクション5.6.7.3 「リモートデータのクローニング」
セクション15.6.3.3 「一般テーブルスペース」
セクション15.9.1.2 「圧縮テーブルの作成」

innodb_parallel_read_threads

セクション13.7.3.2 「CHECK TABLE ステートメント」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション1.3 「MySQL 8.0 の新機能」

innodb_print_all_deadlocks

セクション15.7.5 「InnoDB のデッドロック」
セクション15.21 「InnoDB のトラブルシューティング」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.6.1.5 「MyISAM から InnoDB へのテーブルの変換」
セクション15.7.5.3 「デッドロックを最小化および処理する方法」

innodb_print_ddl_logs

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション13.1.1 「アトミックデータ定義ステートメントのサポート」

innodb_purge_batch_size

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.8.9 「ページ構成」

innodb_purge_rseg_truncate_frequency

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.6.3.4 「undo テーブルスペース」
セクション15.8.9 「ページ構成」

innodb_purge_threads

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
MySQL 用語集
セクション15.8.9 「ページ構成」

innodb_random_read_ahead

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

セクション15.8.3.4 「InnoDB バッファプールのプリフェッチ (先読み) の構成」
MySQL 用語集

innodb_read_ahead_threshold

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.8.3.4 「InnoDB バッファプールのプリフェッチ (先読み) の構成」

innodb_read_io_threads

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.8.5 「InnoDB バックグラウンド I/O スレッドの数の構成」
セクション15.17.3 「InnoDB 標準モニターおよびロックモニターの出力」
セクション15.8.6 「Linux での非同期 I/O の使用」

innodb_read_only

セクション13.7.3.1 「ANALYZE TABLE ステートメント」
セクション26.34 「INFORMATION_SCHEMA STATISTICS テーブル」
セクション26.38 「INFORMATION_SCHEMA TABLES テーブル」
セクション8.2.3 「INFORMATION_SCHEMA クエリーの最適化」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション5.1.8 「サーバーシステム変数」
セクション14.7 「データディクショナリの使用法の違い」
セクション15.8.2 「読み取り専用操作の InnoDB の構成」

innodb_redo_log_archive_dirs

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.6.5 「redo ログ」

innodb_redo_log_encrypt

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.13 「InnoDB 保存データ暗号化」
セクション5.6.7.4 「暗号化データのクローニング」

innodb_replication_delay

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

innodb_rollback_on_timeout

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

innodb_rollback_segments

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション1.3 「MySQL 8.0 の新機能」
セクション15.6.3.4 「undo テーブルスペース」
セクション15.6.6 「undo ログ」

innodb_saved_page_number_debug

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

innodb_sort_buffer_size

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
MySQL 用語集
セクション15.12.3 「オンライン DDL 領域の要件」

innodb_spin_wait_delay

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.8.8 「スピンロックのポーリングの構成」

innodb_spin_wait_pause_multiplier

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション1.3 「MySQL 8.0 の新機能」
セクション15.8.8 「スピンロックのポーリングの構成」

innodb_stats_auto_recalc

セクション13.1.20 「CREATE TABLE ステートメント」
セクション15.8.10 「InnoDB のオプティマイザ統計の構成」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
InnoDB 永続的統計テーブルの例
個々のテーブルのオプティマイザ統計パラメータの構成
永続オプティマイザ統計の自動統計計算の構成

innodb_stats_include_delete_marked

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
永続統計計算への削除マーク付きレコードの組込み

innodb_stats_method

セクション8.3.8 「InnoDB および MyISAM インデックス統計コレクション」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
MySQL 用語集

innodb_stats_on_metadata

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.8.10.2 「非永続的オプティマイザ統計のパラメータの構成」

innodb_stats_persistent

セクション13.7.3.1 「ANALYZE TABLE ステートメント」
セクション13.1.15 「CREATE INDEX ステートメント」
セクション13.1.20 「CREATE TABLE ステートメント」
セクション15.8.10 「InnoDB のオプティマイザ統計の構成」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.8.10.3 「InnoDB テーブルに対する ANALYZE TABLE の複雑さの推定」
個々のテーブルのオプティマイザ統計パラメータの構成
セクション8.5.10 「多くのテーブルのあるシステムに対する InnoDB の最適化」
セクション15.8.10.1 「永続的オプティマイザ統計のパラメータの構成」
セクション15.8.10.2 「非永続的オプティマイザ統計のパラメータの構成」

innodb_stats_persistent_sample_pages

セクション13.7.3.1 「ANALYZE TABLE ステートメント」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
InnoDB オプティマイザ統計でサンプリングされるページの数の構成
セクション15.8.10.3 「InnoDB テーブルに対する ANALYZE TABLE の複雑さの推定」
個々のテーブルのオプティマイザ統計パラメータの構成

innodb_stats_sample_pages

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

innodb_stats_transient_sample_pages

セクション13.7.3.1 「ANALYZE TABLE ステートメント」
セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.8.10.3 「InnoDB テーブルに対する ANALYZE TABLE の複雑さの推定」
セクション15.8.10.2 「非永続的オプティマイザ統計のパラメータの構成」

innodb_status_output

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

[セクション15.17.2 「InnoDB モニターの有効化」](#)

[innodb_status_output_locks](#)

[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)

[セクション15.17.2 「InnoDB モニターの有効化」](#)

[innodb_strict_mode](#)

[セクション13.1.20 「CREATE TABLE ステートメント」](#)

[セクション13.1.21 「CREATE TABLESPACE ステートメント」](#)

[セクション13.1.20.2 「CREATE TEMPORARY TABLE ステートメント」](#)

[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)

[セクション15.9.1.5 「InnoDB テーブルでの圧縮の動作」](#)

[セクション1.3 「MySQL 8.0 の新機能」](#)

[MySQL 用語集](#)

[セクション15.9.1.7 「SQL 圧縮構文の警告とエラー」](#)

[セクション5.1.11 「サーバー SQL モード」](#)

[セクション15.9.1.2 「圧縮テーブルの作成」](#)

[innodb_sync_array_size](#)

[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)

[innodb_sync_debug](#)

[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)

[セクション2.9.7 「MySQL ソース構成オプション」](#)

[innodb_sync_spin_loops](#)

[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)

[innodb_table_locks](#)

[セクション15.7.3 「InnoDB のさまざまな SQL ステートメントで設定されたロック」](#)

[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)

[innodb_temp_data_file_path](#)

[セクション26.15 「INFORMATION_SCHEMA FILES テーブル」](#)

[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)

[セクション15.8.1 「InnoDB の起動構成」](#)

[MySQL 用語集](#)

[セクション15.6.3.5 「一時テーブルスペース」](#)

[セクション15.8.2 「読み取り専用操作用の InnoDB の構成」](#)

[innodb_temp_tablespaces_dir](#)

[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)

[セクション15.8.1 「InnoDB の起動構成」](#)

[セクション1.3 「MySQL 8.0 の新機能」](#)

[セクション15.6.5 「redo ログ」](#)

[セクション15.6.3.5 「一時テーブルスペース」](#)

[innodb_thread_concurrency](#)

[セクション15.8.4 「InnoDB のスレッド並列性の構成」](#)

[セクション15.14 「InnoDB の起動オプションおよびシステム変数」](#)

[セクション8.5.9 「InnoDB 構成変数の最適化」](#)

[セクション15.17.3 「InnoDB 標準モニターおよびロックモニターの実出力」](#)

[セクションA.15 「MySQL 8.0 FAQ: MySQL Enterprise Thread Pool」](#)

[innodb_thread_sleep_delay](#)

[セクション15.8.4 「InnoDB のスレッド並列性の構成」](#)

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

innodb_tmpdir

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

セクション15.6.5 「redo ログ」

セクション15.12.5 「オンライン DDL 失敗条件」

セクション15.12.3 「オンライン DDL 領域の要件」

innodb_trx_purge_view_update_only_debug

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

innodb_trx_rseg_n_slots_debug

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

innodb_undo_directory

セクション13.1.21 「CREATE TABLESPACE ステートメント」

セクション15.18.2 「InnoDB のリカバリ」

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

セクション15.8.1 「InnoDB の起動構成」

セクション2.11.4 「MySQL 8.0 での変更」

セクション1.3 「MySQL 8.0 の新機能」

セクション15.6.5 「redo ログ」

セクション15.6.3.4 「undo テーブルスペース」

セクション18.4.6 「グループレプリケーションでの MySQL Enterprise Backup の使用」

セクション15.6.3.6 「サーバーがオフラインのときのテーブルスペースファイルの移動」

セクション5.6.7.3 「リモートデータのクローニング」

セクション15.6.3.3 「一般テーブルスペース」

セクション15.6.1.2 「外部でのテーブルの作成」

セクション15.8.2 「読み取り専用操作の InnoDB の構成」

innodb_undo_log_encrypt

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

セクション15.13 「InnoDB 保存データ暗号化」

セクション5.6.7.4 「暗号化データのクローニング」

innodb_undo_log_truncate

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

セクション1.3 「MySQL 8.0 の新機能」

セクション15.6.3.4 「undo テーブルスペース」

innodb_undo_tablespaces

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

セクション2.11.4 「MySQL 8.0 での変更」

セクション1.3 「MySQL 8.0 の新機能」

MySQL 用語集

セクション15.6.3.4 「undo テーブルスペース」

セクション15.8.2 「読み取り専用操作の InnoDB の構成」

innodb_use_native_aio

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

セクション8.5.8 「InnoDB ディスク I/O の最適化」

セクション15.8.6 「Linux での非同期 I/O の使用」

MySQL 用語集

innodb_validate_tablespace_paths

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

セクション1.3 「MySQL 8.0 の新機能」
セクション15.6.3.7 「テーブルスペースパス検証の無効化」

innodb_version

セクション15.14 「InnoDB の起動オプションおよびシステム変数」

innodb_write_io_threads

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.8.5 「InnoDB バックグラウンド I/O スレッドの数の構成」
セクション15.17.3 「InnoDB 標準モニターおよびロックモニターの出力」
セクション15.8.6 「Linux での非同期 I/O の使用」
セクション15.6.4 「二重書き込みバッファ」

insert_id

セクション16.8.3 「FEDERATED ストレージエンジンの注記とヒント」
セクション5.1.8 「サーバーシステム変数」

interactive_timeout

セクション5.1.8 「サーバーシステム変数」
セクションB.3.2.9 「通信エラーおよび中止された接続」

internal_tmp_disk_storage_engine

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション15.8.1 「InnoDB の起動構成」
セクション1.3 「MySQL 8.0 の新機能」
セクション8.4.4 「MySQL での内部一時テーブルの使用」
セクション8.12.3.1 「MySQL のメモリーの使用方法」
セクション5.1.8 「サーバーシステム変数」
セクション8.2.2.4 「マージまたは実体化を使用した導出テーブル、ビュー参照および共通テーブル式の最適化」
セクション15.6.3.5 「一時テーブルスペース」

internal_tmp_mem_storage_engine

セクション1.3 「MySQL 8.0 の新機能」
セクション8.4.4 「MySQL での内部一時テーブルの使用」
セクション5.1.8 「サーバーシステム変数」

J

[\[index top\]](#)

join_buffer_size

セクション8.2.1.12 「Block Nested Loop 結合と Batched Key Access 結合」
セクション1.3 「MySQL 8.0 の新機能」
セクション8.2.1.7 「Nested Loop 結合アルゴリズム」
セクション5.1.8 「サーバーシステム変数」
セクション8.2.1.4 「ハッシュ結合の最適化」

K

[\[index top\]](#)

keep_files_on_create

セクション5.1.8 「サーバーシステム変数」

key_buffer_size

セクション8.2.5.3 「DELETE ステートメントの最適化」

セクション15.8.1 「InnoDB の起動構成」
セクション15.6.1.5 「MyISAM から InnoDB へのテーブルの変換」
セクション8.10.2 「MyISAM キーキャッシュ」
セクション8.6.2 「MyISAM テーブルの一括データロード」
セクション7.6.3 「MyISAM テーブルの修復方法」
セクション8.12.3.1 「MySQL のメモリーの使用方法」
セクションB.3.7 「MySQL の既知の問題」
セクション8.6.3 「REPAIR TABLE ステートメントの最適化」
セクション8.10.2.6 「キーキャッシュの再構築」
セクション8.8.5 「クエリーパフォーマンスの推定」
セクション5.1.8 「サーバーシステム変数」
セクション5.1.10 「サーバーステータス変数」
セクション5.1.9.5 「構造化システム変数」
セクション8.10.2.2 「複合キーキャッシュ」

key_cache_age_threshold

セクション5.1.8 「サーバーシステム変数」
セクション8.10.2.3 「ミッドポイント挿入戦略」
セクション5.1.9.5 「構造化システム変数」

key_cache_block_size

セクション8.10.2.6 「キーキャッシュの再構築」
セクション8.10.2.5 「キーキャッシュブロックサイズ」
セクション5.1.8 「サーバーシステム変数」
セクション5.1.9.5 「構造化システム変数」

key_cache_division_limit

セクション5.1.8 「サーバーシステム変数」
セクション8.10.2.3 「ミッドポイント挿入戦略」
セクション5.1.9.5 「構造化システム変数」

keyring_aws_cmk_id

セクション6.4.4.5 「keyring_aws Amazon Web Services キーリングプラグインの使用」
セクション6.4.4.13 「キーリングシステム変数」
セクション6.4.4.11 「プラグイン固有のキーリングキー管理関数」

keyring_aws_conf_file

セクション6.4.4.5 「keyring_aws Amazon Web Services キーリングプラグインの使用」
セクション6.4.4.13 「キーリングシステム変数」

keyring_aws_data_file

セクション6.4.4.5 「keyring_aws Amazon Web Services キーリングプラグインの使用」
セクション6.4.4.13 「キーリングシステム変数」
セクション6.4.4.11 「プラグイン固有のキーリングキー管理関数」

keyring_aws_region

セクション6.4.4.13 「キーリングシステム変数」

keyring_encrypted_file_data

セクション15.13 「InnoDB 保存データ暗号化」
セクション6.4.4.3 「keyring_encrypted_file キーリングプラグインの使用」
セクション2.9.7 「MySQL ソース構成オプション」
セクション6.4.4.13 「キーリングシステム変数」

keyring_encrypted_file_password

セクション6.4.4.3 「keyring_encrypted_file キーリングプラグインの使用」
セクション6.4.4.13 「キーリングシステム変数」

keyring_file_data

セクション15.13 「InnoDB 保存データ暗号化」
セクション6.4.4.2 「keyring_file ファイルベースプラグインの使用」
セクション2.9.7 「MySQL ソース構成オプション」
セクション2.4.3 「MySQL 起動デーモンのインストールおよび使用」
セクション6.4.4.9 「キーリングキーストア間のキーの移行」
セクション6.4.4.13 「キーリングシステム変数」

keyring_hashicorp_auth_path

セクション6.4.4.6 「HashiCorp Vault キーリングプラグインの使用」
セクション6.4.4.13 「キーリングシステム変数」

keyring_hashicorp_ca_path

セクション6.4.4.6 「HashiCorp Vault キーリングプラグインの使用」
セクション6.4.4.13 「キーリングシステム変数」

keyring_hashicorp_caching

セクション6.4.4.6 「HashiCorp Vault キーリングプラグインの使用」
セクション6.4.4.13 「キーリングシステム変数」

keyring_hashicorp_commit_auth_path

セクション6.4.4.13 「キーリングシステム変数」

keyring_hashicorp_commit_ca_path

セクション6.4.4.13 「キーリングシステム変数」

keyring_hashicorp_commit_caching

セクション6.4.4.13 「キーリングシステム変数」

keyring_hashicorp_commit_role_id

セクション6.4.4.6 「HashiCorp Vault キーリングプラグインの使用」
セクション6.4.4.13 「キーリングシステム変数」

keyring_hashicorp_commit_server_url

セクション6.4.4.13 「キーリングシステム変数」

keyring_hashicorp_commit_store_path

セクション6.4.4.6 「HashiCorp Vault キーリングプラグインの使用」
セクション6.4.4.13 「キーリングシステム変数」

keyring_hashicorp_role_id

セクション6.4.4.6 「HashiCorp Vault キーリングプラグインの使用」
セクション6.4.4.13 「キーリングシステム変数」

keyring_hashicorp_secret_id

セクション6.4.4.6 「HashiCorp Vault キーリングプラグインの使用」
セクション6.4.4.13 「キーリングシステム変数」

keyring_hashicorp_server_url

セクション6.4.4.6 「HashiCorp Vault キーリングプラグインの使用」
セクション6.4.4.13 「キーリングシステム変数」

keyring_hashicorp_store_path

セクション6.4.4.6 「HashiCorp Vault キーリングプラグインの使用」
セクション6.4.4.13 「キーリングシステム変数」

keyring_oci_ca_certificate

セクション6.4.4.7 「Oracle Cloud Infrastructure Vault キーリングプラグインの使用」
セクション6.4.4.13 「キーリングシステム変数」

keyring_oci_compartment

セクション6.4.4.7 「Oracle Cloud Infrastructure Vault キーリングプラグインの使用」
セクション6.4.4.13 「キーリングシステム変数」

keyring_oci_encryption_endpoint

セクション6.4.4.7 「Oracle Cloud Infrastructure Vault キーリングプラグインの使用」
セクション6.4.4.13 「キーリングシステム変数」

keyring_oci_key_file

セクション6.4.4.7 「Oracle Cloud Infrastructure Vault キーリングプラグインの使用」
セクション6.4.4.13 「キーリングシステム変数」

keyring_oci_key_fingerprint

セクション6.4.4.7 「Oracle Cloud Infrastructure Vault キーリングプラグインの使用」
セクション6.4.4.13 「キーリングシステム変数」

keyring_oci_management_endpoint

セクション6.4.4.7 「Oracle Cloud Infrastructure Vault キーリングプラグインの使用」
セクション6.4.4.13 「キーリングシステム変数」

keyring_oci_master_key

セクション6.4.4.7 「Oracle Cloud Infrastructure Vault キーリングプラグインの使用」
セクション6.4.4.13 「キーリングシステム変数」

keyring_oci_secrets_endpoint

セクション6.4.4.7 「Oracle Cloud Infrastructure Vault キーリングプラグインの使用」
セクション6.4.4.13 「キーリングシステム変数」

keyring_oci_tenancy

セクション6.4.4.7 「Oracle Cloud Infrastructure Vault キーリングプラグインの使用」
セクション6.4.4.13 「キーリングシステム変数」

keyring_oci_user

セクション6.4.4.7 「Oracle Cloud Infrastructure Vault キーリングプラグインの使用」
セクション6.4.4.13 「キーリングシステム変数」

keyring_oci_vaults_endpoint

セクション6.4.4.7 「Oracle Cloud Infrastructure Vault キーリングプラグインの使用」
セクション6.4.4.13 「キーリングシステム変数」

keyring_oci_virtual_vault

セクション6.4.4.7 「Oracle Cloud Infrastructure Vault キーリングプラグインの使用」
セクション6.4.4.13 「キーリングシステム変数」

keyring_okv_conf_dir

セクション6.4.4.4 「keyring_okv KMIP プラグインの使用」
セクション6.4.4.13 「キーリングシステム変数」

keyring_operations

セクション6.4.4.9 「キーリングキーストア間のキーの移行」

[セクション6.4.4.13 「キーリングシステム変数」](#)

L

[\[index top\]](#)

large_files_support

[セクション5.1.8 「サーバーシステム変数」](#)
[セクション24.6 「パーティショニングの制約と制限」](#)

large_page_size

[セクション5.1.8 「サーバーシステム変数」](#)

large_pages

[セクション5.1.8 「サーバーシステム変数」](#)

last_insert_id

[セクション5.1.8 「サーバーシステム変数」](#)
[セクション17.5.1.39 「レプリケーションと変数」](#)
[セクション5.4.4.3 「混合形式のバイナリログイン形式」](#)

lc_messages

[セクション10.12 「エラーメッセージ言語の設定」](#)
[セクション5.1.8 「サーバーシステム変数」](#)

lc_messages_dir

[セクション10.12 「エラーメッセージ言語の設定」](#)
[セクション5.1.8 「サーバーシステム変数」](#)

lc_time_names

[セクション10.16 「MySQL Server のロケールサポート」](#)
[セクション5.1.8 「サーバーシステム変数」](#)
[セクション17.5.1.39 「レプリケーションと変数」](#)
[セクション12.8 「文字列関数および演算子」](#)
[セクション12.7 「日付および時間関数」](#)
[セクション5.4.4.3 「混合形式のバイナリログイン形式」](#)

license

[セクション5.1.8 「サーバーシステム変数」](#)

local

[セクション13.2.8 「LOAD XML ステートメント」](#)

local_infile

[セクション6.1.6 「LOAD DATA LOCAL のセキュリティー上の考慮事項」](#)
[セクション13.2.7 「LOAD DATA ステートメント」](#)
[セクション2.9.7 「MySQL ソース構成オプション」](#)
[セクション5.1.8 「サーバーシステム変数」](#)

lock_order

[セクション5.9.3 「LOCK_ORDER ツール」](#)

lock_order_debug_loop

[セクション5.9.3 「LOCK_ORDER ツール」](#)

lock_order_debug_missing_arc

セクション5.9.3「LOCK_ORDER ツール」

lock_order_debug_missing_key

セクション5.9.3「LOCK_ORDER ツール」

lock_order_debug_missing_unlock

セクション5.9.3「LOCK_ORDER ツール」

lock_order_dependencies

セクション5.9.3「LOCK_ORDER ツール」

lock_order_extra_dependencies

セクション5.9.3「LOCK_ORDER ツール」

lock_order_output_directory

セクション5.9.3「LOCK_ORDER ツール」

lock_order_print_txt

セクション5.9.3「LOCK_ORDER ツール」

lock_order_trace_loop

セクション5.9.3「LOCK_ORDER ツール」

lock_order_trace_missing_arc

セクション5.9.3「LOCK_ORDER ツール」

lock_order_trace_missing_key

セクション5.9.3「LOCK_ORDER ツール」

lock_order_trace_missing_unlock

セクション5.9.3「LOCK_ORDER ツール」

lock_wait_timeout

セクション13.3.5「LOCK INSTANCE FOR BACKUP および UNLOCK INSTANCE ステートメント」

セクション5.1.8「サーバーシステム変数」

locked_in_memory

セクション5.1.8「サーバーシステム変数」

log

セクション15.20.7「InnoDB memcached プラグインとレプリケーション」

セクション15.14「InnoDB の起動オプションおよびシステム変数」

セクション18.9.1「グループレプリケーションの要件」

セクション17.2.2.3「起動オプションとレプリケーションチャンネル」

log_bin

NDB Cluster システム変数

セクション17.1.6.3「Replica Server のオプションと変数」

セクション13.4.1.2「RESET MASTER ステートメント」

セクション17.1.6.4「バイナリロギングのオプションと変数」

セクション5.4.4「バイナリログ」

セクション17.5.1.34「レプリケーションとトランザクションの非一貫性」

[セクション17.1.2.1「レプリケーションソース構成の設定」](#)
[セクション17.5.5「レプリケーションバグまたは問題を報告する方法」](#)

log_bin_basename

[セクション17.1.6.4「バイナリロギングのオプションと変数」](#)
[セクション5.4.4「バイナリログ」](#)

log_bin_index

[セクション17.1.6.4「バイナリロギングのオプションと変数」](#)

log_bin_trust_function_creators

[セクションA.4「MySQL 8.0 FAQ: ストアドプロシージャおよびストアドファンクション」](#)
[セクション25.7「ストアドプログラムバイナリロギング」](#)
[セクション17.1.6.4「バイナリロギングのオプションと変数」](#)

log_bin_use_v

[NDB Cluster の MySQL Server オプション](#)
[セクション23.6.11「NDB Cluster レプリケーションの競合解決」](#)
[セクション17.1.6.4「バイナリロギングのオプションと変数」](#)

log_error

[セクション2.5.6.2「Docker での MySQL Server のデプロイに関するその他のトピック」](#)
[セクション5.4.2.7「JSON 形式でのエラーロギング」](#)
[セクション1.3「MySQL 8.0 の新機能」](#)
[セクション2.4.3「MySQL 起動デーモンのインストールおよび使用」](#)
[セクション5.5.3「エラーログコンポーネント」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション5.4.2.2「デフォルトのエラーログ保存先の構成」](#)

log_error_services

[セクション27.12.19.1「error_log テーブル」](#)
[セクション5.4.2.7「JSON 形式でのエラーロギング」](#)
[セクション1.3「MySQL 8.0 の新機能」](#)
[セクション5.5.3「エラーログコンポーネント」](#)
[セクション5.4.2.1「エラーログ構成」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション5.4.2.8「システムログへのエラーロギング」](#)
[セクション5.4.2.2「デフォルトのエラーログ保存先の構成」](#)
[セクション5.4.2.6「ルールベースのエラーログのフィルタリング \(log_filter_dragonet\)」](#)

log_error_suppression_list

[セクション5.5.3「エラーログコンポーネント」](#)
[セクション5.4.2.4「エラーログフィルタリングのタイプ」](#)
[セクション5.4.2.1「エラーログ構成」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション5.4.2.5「優先度ベースのエラーログのフィルタリング \(log_filter_internal\)」](#)

log_error_verbosity

[セクション1.3「MySQL 8.0 の新機能」](#)
[セクション6.4.7.3「MySQL Enterprise Firewall の使用」](#)
[セクションB.3.2.7「MySQL サーバーが存在しなくなりました」](#)
[セクション5.5.3「エラーログコンポーネント」](#)
[セクション5.4.2.4「エラーログフィルタリングのタイプ」](#)
[セクション5.4.2.9「エラーログ出力形式」](#)
[セクション5.4.2.1「エラーログ構成」](#)
[セクション18.3「グループレプリケーションの監視」](#)

セクション5.1.7 「サーバーコマンドオプション」
セクション5.1.8 「サーバーシステム変数」
セクション5.1.3 「サーバー構成の検証」
セクション5.4.2.8 「システムログへのエラーロギング」
セクション15.6.3.7 「テーブルスペースパス検証の無効化」
セクション27.15 「パフォーマンススキーマシステム変数」
セクション27.12.9 「パフォーマンススキーマ接続属性テーブル」
セクション5.4.2.6 「ルールベースのエラーログのフィルタリング (log_filter_dragonet)」
セクション5.4.2.5 「優先度ベースのエラーログのフィルタリング (log_filter_internal)」
セクション5.4.4.3 「混合形式のバイナリロギング形式」
セクションB.3.2.9 「通信エラーおよび中止された接続」

log_output

セクション5.1.8 「サーバーシステム変数」
セクション5.4.5 「スロークエリーログ」
セクション5.4.3 「一般クエリーログ」
セクション5.4.1 「一般クエリーログおよびスロークエリーログの出力先の選択」

log_queries_not_using_indexes

セクション5.1.8 「サーバーシステム変数」
セクション5.4.5 「スロークエリーログ」

log_raw

セクション5.1.8 「サーバーシステム変数」

log_slave_updates

セクション17.1.3.8 「GTID を操作するストアドファンクションの例」
セクション17.1.3.1 「GTID 形式および格納」
NDB Cluster の MySQL Server オプション
NDB Cluster システム変数
セクション23.6.10 「NDB Cluster レプリケーション: 双方向および循環レプリケーション」
セクション23.6.3 「NDB Cluster レプリケーションの既知の問題」
セクション17.1.6.3 「Replica Server のオプションと変数」
セクション17.1.6.4 「バイナリロギングのオプションと変数」
セクション5.4.4 「バイナリログ」
セクション17.1.2.2 「レプリカ構成の設定」
セクション17.5.1.34 「レプリケーションとトランザクションの非一貫性」
セクション17.4.7 「レプリケーションパフォーマンスを改善する」

log_slow_admin_statements

セクション5.1.8 「サーバーシステム変数」
セクション5.4.5 「スロークエリーログ」

log_slow_extra

セクション5.1.8 「サーバーシステム変数」
セクション5.4.5 「スロークエリーログ」

log_slow_slave_statements

セクション17.1.6.3 「Replica Server のオプションと変数」
セクション5.4.5 「スロークエリーログ」

log_statements_unsafe_for_binlog

セクション17.1.6.4 「バイナリロギングのオプションと変数」

log_syslog

セクション5.1.8 「サーバーシステム変数」

log_syslog_facility

セクション4.3.2 「mysqld_safe — MySQL サーバー起動スクリプト」
セクション5.1.8 「サーバーシステム変数」
セクション5.4.2.8 「システムログへのエラーロギング」

log_syslog_include_pid

セクション5.1.8 「サーバーシステム変数」
セクション5.4.2.8 「システムログへのエラーロギング」

log_syslog_tag

セクション4.3.2 「mysqld_safe — MySQL サーバー起動スクリプト」
セクション5.1.8 「サーバーシステム変数」
セクション5.4.2.8 「システムログへのエラーロギング」

log_throttle_queries_not_using_indexes

セクション5.1.8 「サーバーシステム変数」
セクション5.4.5 「スロークエリーログ」

log_timestamps

セクション27.12.19.1 「error_log テーブル」
セクション5.4.2.9 「エラーログ出力形式」
セクション5.1.8 「サーバーシステム変数」
セクション5.4.5 「スロークエリーログ」
セクション5.4.3 「一般クエリーログ」

long_query_time

セクション5.4 「MySQL Server ログ」
セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」
セクション17.1.6.3 「Replica Server のオプションと変数」
セクション5.1.8 「サーバーシステム変数」
セクション5.1.10 「サーバーステータス変数」
セクション5.4.5 「スロークエリーログ」

low_priority_updates

セクションA.14 「MySQL 8.0 FAQ: レプリケーション」
セクション5.1.8 「サーバーシステム変数」
セクション8.11.2 「テーブルロックの問題」

lower

セクション18.9.1 「グループレプリケーションの要件」

lower_case_file_system

セクション5.1.8 「サーバーシステム変数」

lower_case_table_names

セクション13.1.20.5 「FOREIGN KEY の制約」
セクション13.7.1.6 「GRANT ステートメント」
セクション13.2.5 「IMPORT TABLE ステートメント」
セクション26.51.8 「INFORMATION_SCHEMA INNODB_COLUMNS テーブル」
セクション26.51.24 「INFORMATION_SCHEMA INNODB_TABLES テーブル」
セクション10.8.7 「INFORMATION_SCHEMA 検索での照合の使用」
セクション15.6.1.3 「InnoDB テーブルのインポート」
セクション15.6.1.4 「InnoDB テーブルの移動またはコピー」
セクション2.11.4 「MySQL 8.0 での変更」
セクション23.1.4 「NDB Cluster の新機能」
セクション13.7.1.8 「REVOKE ステートメント」

セクション13.7.7.39「SHOW TABLES ステートメント」
セクション2.11.5「アップグレード用のインストールの準備」
セクション18.7.3.1「オンラインアップグレードに関する考慮事項」
セクション18.8「グループレプリケーションシステム変数」
セクション17.2.5「サーバーがレプリケーションフィルタリングルールをどのように評価するか」
セクション5.1.8「サーバーシステム変数」
セクション17.5.1.39「レプリケーションと変数」
セクション9.2.3「識別子の太文字と小文字の区別」
セクション1.6「質問またはバグをレポートする方法」
高度なオプション

M

[\[index top\]](#)

mandatory_roles

セクション13.7.1.4「DROP ROLE ステートメント」
セクション13.7.1.5「DROP USER ステートメント」
セクション6.2.2「MySQL で提供される権限」
セクション13.7.1.8「REVOKE ステートメント」
セクション13.7.1.11「SET ROLE ステートメント」
セクション13.7.7.21「SHOW GRANTS ステートメント」
セクション6.2.11「アカウントカテゴリ」
セクション5.1.8「サーバーシステム変数」
セクション5.1.9.1「システム変数権限」
セクション6.2.10「ロールの使用」

master_info_repository

セクション1.3「MySQL 8.0 の新機能」
セクション17.1.6.3「Replica Server のオプションと変数」
セクション13.4.2.5「RESET REPLICA | SLAVE ステートメント」
セクション18.9.1「グループレプリケーションの要件」
セクション17.2.4.2「レプリケーションメタデータリポジトリ」
セクション5.6.7.6「レプリケーション用のクローニング」
セクション17.2.2.3「起動オプションとレプリケーションチャンネル」

master_verify_checksum

MySQL 用語集
セクション17.1.6.4「バイナリロギングのオプションと変数」
セクション5.4.4「バイナリログ」

max_allowed_packet

セクション11.3.4「BLOB 型と TEXT 型」
セクション11.5「JSON データ型」
セクション8.12.3.1「MySQL のメモリーの使用方法」
セクションB.3.2.7「MySQL サーバーが存在しなくなりました」
セクションB.3.2.3「MySQL サーバーへの接続が失われました」
セクション23.6.11「NDB Cluster レプリケーションの競合解決」
セクション17.1.6.3「Replica Server のオプションと変数」
セクション20.5.6.2「X プラグイン のオプションとシステム変数」
セクション5.1.8「サーバーシステム変数」
セクション11.7「データ型のストレージ要件」
バイナリログトランザクション圧縮が有効な場合の動作
セクション5.6.6.3「バージョントークンの使用」
セクションB.3.2.8「パケットが大きすぎます」
セクション5.6.7.3「リモートデータのクローニング」
セクション17.5.1.20「レプリケーションと max_allowed_packet」
セクション12.8「文字列関数および演算子」

セクション12.4.2 「比較関数と演算子」
セクションB.3.2.9 「通信エラーおよび中止された接続」
セクションB.3.4.6 「関連するテーブルからの行の削除」
セクション12.20.1 「集計関数の説明」

max_binlog_cache_size

セクション8.12.3.1 「MySQL のメモリーの使用法」
セクション17.1.6.4 「バイナリロギングのオプションと変数」
セクション5.4.4 「バイナリログ」

max_binlog_size

セクション17.1.3.1 「GTID 形式および格納」
セクション5.4 「MySQL Server ログ」
セクション17.1.6.3 「Replica Server のオプションと変数」
セクション5.4.6 「サーバーログの保守」
セクション17.1.6.4 「バイナリロギングのオプションと変数」
セクション5.4.4 「バイナリログ」
セクション17.2.4.1 「リレーログ」

max_binlog_stmt_cache_size

セクション8.12.3.1 「MySQL のメモリーの使用法」
セクション17.1.6.4 「バイナリロギングのオプションと変数」

max_connect_errors

セクション5.1.12.3 「DNS ルックアップとホストキャッシュ」
セクション27.12.19.5 「host_cache テーブル」
セクション5.1.8 「サーバーシステム変数」

max_connections

セクション5.9.1.4 「gdb での mysqld のデバッグ」
セクション1.3 「MySQL 8.0 の新機能」
セクション8.4.3.1 「MySQL でのテーブルのオープンとクローズの方法」
セクション6.2.2 「MySQL で提供される権限」
セクション27.12.19.9 「processlist テーブル」
セクション20.5.6.2 「X プラグイン のオプションとシステム変数」
セクション5.1.8 「サーバーシステム変数」
セクション5.1.10 「サーバーステータス変数」
セクション5.6.3.3 「スレッドプール操作」
セクション14.4 「ディクショナリオブジェクトキャッシュ」
セクション27.15 「パフォーマンススキーマシステム変数」
セクションB.3.2.16 「ファイルが見つからず同様のエラーが発生しました」
セクションB.3.2.5 「接続が多すぎます」
セクション5.1.12.1 「接続インターフェース」
セクション5.1.12.2 「管理接続管理」

max_delayed_threads

セクション5.1.8 「サーバーシステム変数」

max_digest_length

セクション8.12.3.1 「MySQL のメモリーの使用法」
セクション5.1.8 「サーバーシステム変数」
セクション27.10 「パフォーマンススキーマのステートメントダイジェストとサンプリング」
セクション27.15 「パフォーマンススキーマシステム変数」
セクション12.14 「暗号化関数と圧縮関数」

max_error_count

セクション13.2.7 「LOAD DATA ステートメント」

セクション13.6.7.7 「MySQL の診断領域」
セクション13.6.7.4 「RESIGNAL ステートメント」
セクション13.7.7.17 「SHOW ERRORS ステートメント」
セクション13.7.7.42 「SHOW WARNINGS ステートメント」
セクション5.1.8 「サーバーシステム変数」

max_execution_time

セクション13.2.15 「WITH (共通テーブル式)」
セクション8.9.3 「オブティマイザヒント」
セクション5.1.8 「サーバーシステム変数」
セクション5.1.10 「サーバーステータス変数」

max_heap_table_size

セクション16.3 「MEMORY ストレージエンジン」
セクション8.4.4 「MySQL での内部一時テーブルの使用」
セクション8.12.3.1 「MySQL のメモリーの使用方法」
セクション5.1.8 「サーバーシステム変数」
セクション13.6.6.5 「サーバー側のカーソルの制約」
セクション8.4.6 「テーブルサイズの制限」
セクション17.5.1.21 「レプリケーションと MEMORY テーブル」
セクション17.5.1.39 「レプリケーションと変数」

max_insert_delayed_threads

セクション5.1.8 「サーバーシステム変数」

max_join_size

セクション8.8.2 「EXPLAIN 出力フォーマット」
セクション4.5.1.6 「mysql クライアントのヒント」
セクション5.1.8 「サーバーシステム変数」
セクション13.7.6.1 「変数代入の SET 構文」

max_length_for_sort_data

セクション1.3 「MySQL 8.0 の新機能」
セクション8.2.1.16 「ORDER BY の最適化」
セクション5.1.8 「サーバーシステム変数」

max_points_in_geometry

セクション5.1.8 「サーバーシステム変数」
セクション12.17.8 「空間演算子関数」

max_prepared_stmt_count

セクション13.5.3 「DEALLOCATE PREPARE ステートメント」
セクション5.1.8 「サーバーシステム変数」
セクション5.1.10 「サーバーステータス変数」
セクション27.15 「パフォーマンススキーマシステム変数」
セクション13.5 「プリペアドステートメント」
セクション8.10.3 「プリペアドステートメントおよびストアードプログラムのキャッシュ」

max_relay_log_size

セクション17.1.6.3 「Replica Server のオプションと変数」
セクション17.1.6.4 「バイナリロギングのオプションと変数」
セクション17.2.4.1 「リレーログ」

max_seeks_for_key

セクション13.7.3.1 「ANALYZE TABLE ステートメント」
セクション5.1.8 「サーバーシステム変数」

max_sort_length

セクション11.3.4「BLOB 型と TEXT 型」
セクション11.5「JSON データ型」
セクションB.3.7「MySQL の既知の問題」
セクション8.2.1.16「ORDER BY の最適化」
セクション13.2.10「SELECT ステートメント」
セクション5.1.8「サーバーシステム変数」

max_sp_recursion_depth

セクション5.1.8「サーバーシステム変数」
セクション25.2.1「ストアドルーチンの構文」

max_user_connections

セクション13.7.1.1「ALTER USER ステートメント」
セクション13.7.1.3「CREATE USER ステートメント」
セクション13.7.8.3「FLUSH ステートメント」
セクション6.2.20「アカウントリソース制限の設定」
セクション5.1.8「サーバーシステム変数」
セクション6.1.3「攻撃者に対する MySQL のセキュアな状態の維持」

max_write_lock_count

セクション13.3.6「LOCK TABLES および UNLOCK TABLES ステートメント」
セクション5.1.8「サーバーシステム変数」
セクション8.11.2「テーブルロックの問題」
セクション8.11.4「メタデータのロック」

mecab_rc_file

セクション12.10.9「MeCab フルテキストパーサープラグイン」
セクション5.1.8「サーバーシステム変数」

metadata_locks_cache_size

セクション5.1.8「サーバーシステム変数」

metadata_locks_hash_instances

セクション5.1.8「サーバーシステム変数」

min_examined_row_limit

セクション5.1.8「サーバーシステム変数」
セクション5.4.5「スロークエリログ」

myisam_data_pointer_size

セクション13.1.20「CREATE TABLE ステートメント」
セクション5.1.8「サーバーシステム変数」
セクション8.4.6「テーブルサイズの制限」

myisam_max_sort_file_size

セクション16.2.1「MyISAM 起動オプション」
セクション8.6.3「REPAIR TABLE ステートメントの最適化」
セクション5.1.8「サーバーシステム変数」

myisam_mmap_size

セクション5.1.8「サーバーシステム変数」

myisam_recover_options

セクション8.6.1「MyISAM クエリーの最適化」

[セクション16.2「MyISAM ストレージエンジン」](#)
[セクション7.6.5「MyISAM テーブル保守スケジュールのセットアップ」](#)
[セクション16.2.1「MyISAM 起動オプション」](#)
[セクション5.9.1.6「mysqld でのエラーの原因を見つけるためのサーバーログの使用」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクションB.3.2.17「テーブルの破損の問題」](#)

myisam_repair_threads

[セクション5.1.8「サーバーシステム変数」](#)

myisam_sort_buffer_size

[セクション13.1.9「ALTER TABLE ステートメント」](#)
[セクション16.2.1「MyISAM 起動オプション」](#)
[セクション8.6.3「REPAIR TABLE ステートメントの最適化」](#)
[セクション5.1.8「サーバーシステム変数」](#)

myisam_stats_method

[セクション8.3.8「InnoDB および MyISAM インデックス統計コレクション」](#)
[セクション5.1.8「サーバーシステム変数」](#)

myisam_use_mmap

[セクション8.12.3.1「MySQL のメモリーの使用方法」](#)
[セクション5.1.8「サーバーシステム変数」](#)

mysql_firewall_mode

[セクション6.4.7.3「MySQL Enterprise Firewall の使用」](#)
[セクション6.4.7.4「MySQL Enterprise Firewall リファレンス」](#)

mysql_firewall_trace

[セクション6.4.7.3「MySQL Enterprise Firewall の使用」](#)
[セクション6.4.7.4「MySQL Enterprise Firewall リファレンス」](#)

mysql_native_password_proxy_users

[セクション5.1.8「サーバーシステム変数」](#)
[セクション6.2.18「プロキシユーザー」](#)

mysqlx_bind_address

[セクション20.5.6.2「X プラグイン のオプションとシステム変数」](#)
[セクション20.5.6.3「X プラグイン ステータス変数」](#)
[セクション5.1.14「ネットワークネームスペースのサポート」](#)

mysqlx_compression_algorithms

[セクション20.5.5「X プラグイン での接続圧縮」](#)
[セクション20.5.6.2「X プラグイン のオプションとシステム変数」](#)
[セクション20.5.6.3「X プラグイン ステータス変数」](#)

mysqlx_connect_timeout

[セクション20.5.6.2「X プラグイン のオプションとシステム変数」](#)

mysqlx_deflate_default_compression_level

[セクション20.5.5「X プラグイン での接続圧縮」](#)
[セクション20.5.6.2「X プラグイン のオプションとシステム変数」](#)

mysqlx_deflate_max_client_compression_level

[セクション20.5.5「X プラグイン での接続圧縮」](#)

セクション20.5.6.2「X プラグイン のオプションとシステム変数」

mysqlx_document_id_unique_prefix

セクション20.5.6.2「X プラグイン のオプションとシステム変数」

mysqlx_enable_hello_notice

セクション20.5.6.2「X プラグイン のオプションとシステム変数」

mysqlx_idle_worker_thread_timeout

セクション20.5.6.2「X プラグイン のオプションとシステム変数」

mysqlx_interactive_timeout

セクション20.5.6.2「X プラグイン のオプションとシステム変数」

mysqlx_lz

セクション20.5.5「X プラグイン での接続圧縮」

セクション20.5.6.2「X プラグイン のオプションとシステム変数」

mysqlx_max_allowed_packet

セクション20.5.5「X プラグイン での接続圧縮」

セクション20.5.6.2「X プラグイン のオプションとシステム変数」

mysqlx_max_connections

セクション20.5.6.2「X プラグイン のオプションとシステム変数」

mysqlx_min_worker_threads

セクション20.5.6.2「X プラグイン のオプションとシステム変数」

mysqlx_port

セクション13.7.5「CLONE ステートメント」

セクション20.3.1「MySQL Shell」

セクション20.4.1「MySQL Shell」

セクション2.9.7「MySQL ソース構成オプション」

セクション6.7.5.2「MySQL 機能の TCP ポートコンテキストの設定」

セクション20.5.6.2「X プラグイン のオプションとシステム変数」

セクション5.6.7.13「クローンプラグインの制限事項」

セクション5.6.7.3「リモートデータのクローニング」

mysqlx_port_open_timeout

セクション20.5.6.2「X プラグイン のオプションとシステム変数」

mysqlx_read_timeout

セクション20.5.6.2「X プラグイン のオプションとシステム変数」

mysqlx_socket

セクション20.5.6.2「X プラグイン のオプションとシステム変数」

mysqlx_ssl_ca

セクション20.5.6.2「X プラグイン のオプションとシステム変数」

mysqlx_ssl_capath

セクション20.5.6.2「X プラグイン のオプションとシステム変数」

mysqlx_ssl_cert

セクション20.5.6.2「X プラグイン のオプションとシステム変数」

mysqlx_ssl_cipher

セクション20.5.6.2「X プラグイン のオプションとシステム変数」

mysqlx_ssl_crl

セクション20.5.6.2「X プラグイン のオプションとシステム変数」

mysqlx_ssl_crlpath

セクション20.5.6.2「X プラグイン のオプションとシステム変数」

mysqlx_ssl_key

セクション20.5.6.2「X プラグイン のオプションとシステム変数」

mysqlx_wait_timeout

セクション20.5.6.2「X プラグイン のオプションとシステム変数」

mysqlx_write_timeout

セクション20.5.6.2「X プラグイン のオプションとシステム変数」

mysqlx_zstd_default_compression_level

セクション20.5.5「X プラグイン での接続圧縮」

セクション20.5.6.2「X プラグイン のオプションとシステム変数」

mysqlx_zstd_max_client_compression_level

セクション20.5.5「X プラグイン での接続圧縮」

セクション20.5.6.2「X プラグイン のオプションとシステム変数」

N

[\[index top\]](#)

named_pipe

セクションB.3.2.2「[ローカルの] MySQL サーバーに接続できません」

セクション1.2.2「MySQL の主な機能」

セクション4.5.1.1「mysql クライアントオプション」

セクション2.3.4.3「MySQL サーバータイプの選択」

セクション4.4.2「mysql_secure_installation — MySQL インストールのセキュリティ改善」

セクション4.4.5「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

セクション4.5.2「mysqladmin — A MySQL Server 管理プログラム」

セクション4.6.8「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」

セクション4.5.3「mysqlcheck — テーブル保守プログラム」

セクション4.5.4「mysqldump — データベースバックアッププログラム」

セクション4.5.5「mysqlimport — データインポートプログラム」

セクション4.5.6「mysqlpump — データベースバックアッププログラム」

セクション4.5.7「mysqlshow — データベース、テーブル、およびカラム情報の表示」

セクション4.5.8「mysqlslap — ロードエミュレーションクライアント」

セクション4.2.4「コマンドオプションを使用した MySQL Server への接続」

セクション4.2.3「サーバーに接続するためのコマンドオプション」

セクション5.1.8「サーバーシステム変数」

タイプとネットワーキング

named_pipe_full_access_group

セクション1.3「MySQL 8.0 の新機能」

セクション4.5.1.1「mysql クライアントオプション」

セクション4.4.2「mysql_secure_installation — MySQL インストールのセキュリティ改善」

セクション4.4.5「mysql_upgrade — MySQL テーブルのチェックとアップグレード」

セクション4.5.2 「mysqladmin — A MySQL Server 管理プログラム」
セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」
セクション4.2.4 「コマンドオプションを使用した MySQL Server への接続」
セクション4.2.3 「サーバーに接続するためのコマンドオプション」
セクション5.1.8 「サーバーシステム変数」
セクション4.2.7 「接続トランスポートプロトコル」

ndb_autoincrement_prefetch_sz

セクション23.1.4 「NDB Cluster の新機能」
NDB Cluster システム変数
セクション23.4.13 「ndb_import — NDB への CSV データのインポート」

ndb_cache_check_time

NDB Cluster システム変数

ndb_clear_apply_status

NDB Cluster システム変数
セクション13.4.2.5 「RESET REPLICAS | SLAVE ステートメント」

ndb_conflict_role

セクション23.1.4 「NDB Cluster の新機能」
NDB Cluster システム変数

ndb_data_node_neighbour

NDB Cluster システム変数
セクション13.1.20.11 「NDB_TABLE オプションの設定」

ndb_dbg_check_shares

NDB Cluster システム変数

ndb_default_column_format

NDB Cluster システム変数

ndb_deferred_constraints

NDB Cluster システム変数

ndb_distribution

NDB Cluster システム変数

ndb_eventbuffer_free_percent

NDB Cluster システム変数
セクション23.5.2.3 「クラスタログでのイベントバッファのレポート」

ndb_eventbuffer_max_alloc

NDB Cluster システム変数
セクション23.5.2.3 「クラスタログでのイベントバッファのレポート」

ndb_extra_logging

NDB Cluster システム変数

ndb_force_send

NDB Cluster システム変数

ndb_fully_replicated

NDB Cluster システム変数

ndb_index_stat_enable

NDB Cluster システム変数

ndb_index_stat_option

NDB Cluster システム変数

ndb_join_pushdown

セクション8.8.2「EXPLAIN 出力フォーマット」

NDB Cluster システム変数

ndb_log_apply_status

NDB Cluster システム変数

セクション23.6.10「NDB Cluster レプリケーション: 双方向および循環レプリケーション」

ndb_log_bin

セクション23.1.4「NDB Cluster の新機能」

NDB Cluster システム変数

ndb_log_binlog_index

NDB Cluster システム変数

ndb_log_empty_epochs

NDB Cluster システム変数

ndb_log_empty_update

NDB Cluster システム変数

ndb_log_exclusive_reads

NDB Cluster の MySQL Server オプション

NDB Cluster システム変数

セクション23.6.11「NDB Cluster レプリケーションの競合解決」

ndb_log_orig

NDB Cluster システム変数

ndb_log_transaction_id

NDB Cluster システム変数

ndb_metadata_check

セクション23.1.4「NDB Cluster の新機能」

NDB Cluster システム変数

セクション23.4.23「ndb_restore — NDB Cluster バックアップの復元」

セクション27.12.12「パフォーマンススキーマ NDB Cluster テーブル」

ndb_metadata_check_interval

セクション23.1.4「NDB Cluster の新機能」

NDB Cluster システム変数

[セクション27.12.12「パフォーマンススキーマ NDB Cluster テーブル」](#)

ndb_metadata_sync

[セクション23.1.4「NDB Cluster の新機能」](#)

NDB Cluster システム変数

[セクション27.12.12「パフォーマンススキーマ NDB Cluster テーブル」](#)

ndb_optimized_node_selection

[セクション23.3.3.10「NDB Cluster TCP/IP 接続」](#)

NDB Cluster システム変数

[セクション23.5.3.3「NDB Cluster 管理クライアントでの CLUSTERLOG STATISTICS の使用」](#)

ndb_read_backup

[セクション23.1.4「NDB Cluster の新機能」](#)

NDB Cluster システム変数

[セクション13.1.20.11「NDB_TABLE オプションの設定」](#)

ndb_recv_thread_activation_threshold

NDB Cluster システム変数

ndb_recv_thread_cpu_mask

NDB Cluster システム変数

ndb_report_thresh_binlog_epoch_slip

NDB Cluster システム変数

[セクション23.5.2.3「クラスタログでのイベントバッファのレポート」](#)

ndb_report_thresh_binlog_mem_usage

NDB Cluster システム変数

[セクション23.5.2.3「クラスタログでのイベントバッファのレポート」](#)

ndb_row_checksum

NDB Cluster システム変数

ndb_schema_dist_lock_wait_timeout

NDB Cluster システム変数

ndb_schema_dist_timeout

NDB Cluster システム変数

ndb_schema_dist_upgrade_allowed

NDB Cluster システム変数

ndb_show_foreign_key_mock_tables

NDB Cluster システム変数

ndb_slave_conflict_role

[セクション23.1.4「NDB Cluster の新機能」](#)

NDB Cluster システム変数

[セクション23.6.11「NDB Cluster レプリケーションの競合解決」](#)

ndb_table_no_logging

NDB Cluster システム変数

セクション13.1.20.11 「NDB_TABLE オプションの設定」

ndb_table_temporary

NDB Cluster システム変数

ndb_use_copying_alter_table

NDB Cluster システム変数

ndb_use_exact_count

NDB Cluster システム変数

ndb_use_transactions

NDB Cluster システム変数

ndb_version

NDB Cluster システム変数

ndb_version_string

NDB Cluster システム変数

ndbinfo_database

NDB Cluster システム変数

ndbinfo_max_bytes

NDB Cluster システム変数

ndbinfo_max_rows

NDB Cluster システム変数

ndbinfo_offline

NDB Cluster システム変数

ndbinfo_show_hidden

NDB Cluster システム変数

セクション23.5.14.6 「ndbinfo cluster_operations テーブル」
セクション23.5.14.7 「ndbinfo cluster_transactions テーブル」
セクション23.5.14.45 「ndbinfo server_operations テーブル」
セクション23.5.14.46 「ndbinfo server_transactions テーブル」

ndbinfo_table_prefix

NDB Cluster システム変数

ndbinfo_version

NDB Cluster システム変数

net_buffer_length

セクション8.12.3.1 「MySQL のメモリーの使用方法」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション5.1.8 「サーバーシステム変数」

net_read_timeout

セクションB.3.2.3 「MySQL サーバーへの接続が失われました」
セクション5.1.8 「サーバーシステム変数」

net_retry_count

セクション5.1.8「サーバーシステム変数」

net_write_timeout

セクション5.1.8「サーバーシステム変数」

new

セクション5.1.8「サーバーシステム変数」

セクション24.6.2「ストレージエンジンに関連するパーティショニング制限」

ngram_token_size

セクション12.10.6「MySQLの全文検索の微調整」

セクション12.10.8「ngram全文パーサー」

セクション5.1.8「サーバーシステム変数」

セクション12.10.2「ブール全文検索」

セクション12.10.4「全文ストップワード」

セクション12.10.1「自然言語全文検索」

O

[[index top](#)]

offline_mode

セクション6.2.2「MySQLで提供される権限」

セクション18.3.1「グループレプリケーションサーバーの状態」

セクション18.8「グループレプリケーションシステム変数」

セクション5.1.8「サーバーシステム変数」

セクション18.6.6.4「終了処理」

old

セクション8.9.4「インデックスヒント」

セクション5.1.8「サーバーシステム変数」

old_alter_table

セクション13.1.9「ALTER TABLEステートメント」

セクション15.12「InnoDBとオンラインDDL」

セクション1.3「MySQL 8.0の新機能」

セクション13.7.3.4「OPTIMIZE TABLEステートメント」

セクション24.3.1「RANGEおよびLISTパーティションの管理」

セクション15.12.2「オンラインDDLのパフォーマンスと同時実行性」

セクション15.12.4「オンラインDDLを使用したDDLステートメントの簡略化」

セクション15.12.1「オンラインDDL操作」

セクション5.1.8「サーバーシステム変数」

open_files_limit

セクション15.14「InnoDBの起動オプションおよびシステム変数」

セクション1.3「MySQL 8.0の新機能」

セクション8.4.3.1「MySQLでのテーブルのオープンとクローズの方法」

セクション2.5.9「systemdを使用したMySQL Serverの管理」

セクション5.1.8「サーバーシステム変数」

セクション8.2.1.4「ハッシュ結合の最適化」

セクション27.15「パフォーマンススキーマシステム変数」

セクション24.6「パーティショニングの制約と制限」

セクションB.3.2.16「ファイルが見つからず同様のエラーが発生しました」

セクション5.4.1「一般クエリログおよびスロークエリログの出力先の選択」

セクション5.1.12.1「接続インタフェース」

optimizer_prune_level

セクション8.9.3「オプティマイザヒント」
セクション8.9.1「クエリー計画評価の制御」
セクション5.1.8「サーバーシステム変数」
セクション8.2.2.1「準結合変換による IN および EXISTS サブクエリー述語の最適化」

optimizer_search_depth

セクション8.9.1「クエリー計画評価の制御」
セクション5.1.8「サーバーシステム変数」

optimizer_switch

セクション8.2.1.12「Block Nested Loop 結合と Batched Key Access 結合」
セクション8.2.2.3「EXISTS 戦略を使用したサブクエリーの最適化」
セクション8.2.1.19「LIMIT クエリーの最適化」
セクション28.4.5.7「list_add() 関数」
セクション8.2.1.11「Multi-Range Read の最適化」
セクション1.3「MySQL 8.0 の新機能」
セクション8.2.1.2「range の最適化」
セクション13.2.13「UPDATE ステートメント」
セクション8.2.1.6「インデックスコンディションプッシュダウンの最適化」
セクション8.2.1.3「インデックスマージの最適化」
セクション8.3.10「インデックス拡張の使用」
セクション8.2.1.5「エンジンコンディションプッシュダウンの最適化」
セクション8.9.3「オプティマイザヒント」
セクション8.9.6「オプティマイザ統計」
セクション5.1.8「サーバーシステム変数」
セクション8.2.1.4「ハッシュ結合の最適化」
セクション25.5.2「ビュー処理アルゴリズム」
セクション8.2.2.4「マージまたは実体化を使用した導出テーブル、ビュー参照および共通テーブル式の最適化」
セクション8.3.12「不可視のインデックス」
セクション8.9.2「切り替え可能な最適化」
セクション8.2.2.2「実体化を使用したサブクエリーの最適化」
セクション8.2.2.5「導出条件プッシュダウン最適化」
セクション8.2.1.13「条件フィルタ」
セクション8.2.2.1「準結合変換による IN および EXISTS サブクエリー述語の最適化」

optimizer_trace

セクション26.19「INFORMATION_SCHEMA OPTIMIZER_TRACE テーブル」
セクション5.1.8「サーバーシステム変数」

optimizer_trace_features

セクション5.1.8「サーバーシステム変数」

optimizer_trace_limit

セクション5.1.8「サーバーシステム変数」

optimizer_trace_max_mem_size

セクション26.19「INFORMATION_SCHEMA OPTIMIZER_TRACE テーブル」
セクション5.1.8「サーバーシステム変数」

optimizer_trace_offset

セクション5.1.8「サーバーシステム変数」

original_commit_timestamp

セクション17.1.6.4「バイナリロギングのオプションと変数」
セクション17.3.3.1「レプリケーション PRIVILEGE_CHECKS_USER アカウントの権限」

original_server_version

[セクション17.5.2「MySQLバージョン間のレプリケーション互換性」](#)
[セクション4.6.8「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション17.3.3.1「レプリケーション PRIVILEGE_CHECKS_USER アカウントの権限」](#)
[セクション17.1.6.2「レプリケーションソースのオプションと変数」](#)

P

[\[index top\]](#)

parser_max_mem_size

[セクション5.1.8「サーバーシステム変数」](#)

partial_revokes

[セクション13.7.1.6「GRANT ステートメント」](#)
[セクション1.3「MySQL 8.0 の新機能」](#)
[セクション6.2.11「アカウントカテゴリ」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション6.2.12「部分取消しを使用した権限の制限」](#)

password_history

[セクション13.7.1.1「ALTER USER ステートメント」](#)
[セクション13.7.1.3「CREATE USER ステートメント」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション6.2.15「パスワード管理」](#)

password_require_current

[セクション13.7.1.1「ALTER USER ステートメント」](#)
[セクション13.7.1.3「CREATE USER ステートメント」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション6.2.15「パスワード管理」](#)

password_reuse_interval

[セクション13.7.1.1「ALTER USER ステートメント」](#)
[セクション13.7.1.3「CREATE USER ステートメント」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション6.2.15「パスワード管理」](#)

performance_schema

[セクション27.12.19.9「processlist テーブル」](#)
[セクション27.1「パフォーマンススキーマクイックスタート」](#)
[セクション27.15「パフォーマンススキーマシステム変数」](#)
[セクション27.3「パフォーマンススキーマ起動構成」](#)

performance_schema_accounts_size

[セクション27.12.8.1「accounts テーブル」](#)
[セクション27.12.18.12「ステータス変数サマリーテーブル」](#)
[セクション27.12.15「パフォーマンススキーマのステータス変数のテーブル」](#)
[セクション27.15「パフォーマンススキーマシステム変数」](#)

performance_schema_digests_size

[セクション27.12.18.3「ステートメントサマリーテーブル」](#)
[セクション27.10「パフォーマンススキーマのステートメントダイジェストとサンプリング」](#)
[セクション27.15「パフォーマンススキーマシステム変数」](#)

セクション27.16 「パフォーマンススキーマステータス変数」

performance_schema_error_size

セクション27.15 「パフォーマンススキーマシステム変数」

performance_schema_events_stages_history_long_size

セクション27.12.5.3 「events_stages_history_long テーブル」

セクション27.15 「パフォーマンススキーマシステム変数」

performance_schema_events_stages_history_size

セクション27.12.5.2 「events_stages_history テーブル」

セクション27.15 「パフォーマンススキーマシステム変数」

performance_schema_events_statements_history_long_size

セクション27.12.6.3 「events_statements_history_long テーブル」

セクション27.15 「パフォーマンススキーマシステム変数」

performance_schema_events_statements_history_size

セクション27.12.6.2 「events_statements_history テーブル」

セクション27.15 「パフォーマンススキーマシステム変数」

performance_schema_events_transactions_history_long_size

セクション27.12.7.3 「events_transactions_history_long テーブル」

セクション27.15 「パフォーマンススキーマシステム変数」

performance_schema_events_transactions_history_size

セクション27.12.7.2 「events_transactions_history テーブル」

セクション27.15 「パフォーマンススキーマシステム変数」

performance_schema_events_waits_history_long_size

セクション27.12.4.3 「events_waits_history_long テーブル」

セクション13.7.7.15 「SHOW ENGINE ステートメント」

セクション27.15 「パフォーマンススキーマシステム変数」

セクション27.12 「パフォーマンススキーマテーブルの説明」

performance_schema_events_waits_history_size

セクション27.12.4.2 「events_waits_history テーブル」

セクション13.7.7.15 「SHOW ENGINE ステートメント」

セクション27.15 「パフォーマンススキーマシステム変数」

セクション27.12 「パフォーマンススキーマテーブルの説明」

performance_schema_hosts_size

セクション27.12.8.2 「hosts テーブル」

セクション27.12.18.12 「ステータス変数サマリーテーブル」

セクション27.12.15 「パフォーマンススキーマのステータス変数のテーブル」

セクション27.15 「パフォーマンススキーマシステム変数」

performance_schema_max_cond_classes

セクション27.15 「パフォーマンススキーマシステム変数」

performance_schema_max_cond_instances

セクション27.15 「パフォーマンススキーマシステム変数」

performance_schema_max_digest_length

セクション27.12.6.1 「events_statements_current テーブル」

セクション5.1.8「サーバーシステム変数」

セクション27.10「パフォーマンススキーマのステートメントダイジェストとサンプリング」

セクション27.15「パフォーマンススキーマシステム変数」

performance_schema_max_digest_sample_age

セクション27.10「パフォーマンススキーマのステートメントダイジェストとサンプリング」

セクション27.15「パフォーマンススキーマシステム変数」

performance_schema_max_file_classes

セクション27.15「パフォーマンススキーマシステム変数」

performance_schema_max_file_handles

セクション27.15「パフォーマンススキーマシステム変数」

performance_schema_max_file_instances

セクション27.15「パフォーマンススキーマシステム変数」

performance_schema_max_index_stat

セクション27.15「パフォーマンススキーマシステム変数」

セクション27.16「パフォーマンススキーマステータス変数」

performance_schema_max_memory_classes

セクション27.15「パフォーマンススキーマシステム変数」

performance_schema_max_metadata_locks

セクション27.12.13.3「metadata_locks テーブル」

セクション27.15「パフォーマンススキーマシステム変数」

セクション27.16「パフォーマンススキーマステータス変数」

performance_schema_max_mutex_classes

セクション27.15「パフォーマンススキーマシステム変数」

セクション27.7「パフォーマンススキーマステータスモニタリング」

performance_schema_max_mutex_instances

セクション27.15「パフォーマンススキーマシステム変数」

performance_schema_max_prepared_statements_instances

セクション27.12.6.4「prepared_statements_instances テーブル」

セクション27.15「パフォーマンススキーマシステム変数」

セクション27.16「パフォーマンススキーマステータス変数」

performance_schema_max_program_instances

セクション27.15「パフォーマンススキーマシステム変数」

セクション27.16「パフォーマンススキーマステータス変数」

performance_schema_max_rwlock_classes

セクション27.15「パフォーマンススキーマシステム変数」

performance_schema_max_rwlock_instances

セクション27.15「パフォーマンススキーマシステム変数」

performance_schema_max_socket_classes

セクション27.15「パフォーマンススキーマシステム変数」

performance_schema_max_socket_instances

セクション27.15 「パフォーマンススキーマシステム変数」

performance_schema_max_sql_text_length

セクション27.12.6.1 「events_statements_current テーブル」

セクション27.12.18.3 「ステートメントサマリーテーブル」

セクション27.10 「パフォーマンススキーマのステートメントダイジェストとサンプリング」

セクション27.15 「パフォーマンススキーマシステム変数」

performance_schema_max_stage_classes

セクション27.12.19.9 「processlist テーブル」

セクション27.15 「パフォーマンススキーマシステム変数」

performance_schema_max_statement_classes

セクション27.15 「パフォーマンススキーマシステム変数」

performance_schema_max_statement_stack

セクション27.15 「パフォーマンススキーマシステム変数」

セクション27.16 「パフォーマンススキーマステータス変数」

performance_schema_max_table_handles

セクション27.12.13.4 「table_handles テーブル」

セクション27.15 「パフォーマンススキーマシステム変数」

セクション27.16 「パフォーマンススキーマステータス変数」

performance_schema_max_table_instances

セクション27.15 「パフォーマンススキーマシステム変数」

performance_schema_max_table_lock_stat

セクション27.15 「パフォーマンススキーマシステム変数」

セクション27.16 「パフォーマンススキーマステータス変数」

performance_schema_max_thread_classes

セクション27.12.19.9 「processlist テーブル」

セクション27.15 「パフォーマンススキーマシステム変数」

performance_schema_max_thread_instances

セクション27.12.19.9 「processlist テーブル」

セクション13.7.7.15 「SHOW ENGINE ステートメント」

セクション27.12.15 「パフォーマンススキーマのステータス変数のテーブル」

セクション27.15 「パフォーマンススキーマシステム変数」

セクション27.16 「パフォーマンススキーマステータス変数」

セクション12.22 「パフォーマンススキーマ関数」

performance_schema_session_connect_attrs_size

セクション27.15 「パフォーマンススキーマシステム変数」

セクション27.16 「パフォーマンススキーマステータス変数」

セクション27.12.9 「パフォーマンススキーマ接続属性テーブル」

performance_schema_setup_actors_size

セクション27.12.2.1 「setup_actors テーブル」

セクション27.15 「パフォーマンススキーマシステム変数」

performance_schema_setup_objects_size

セクション27.12.2.4 「setup_objects テーブル」

セクション27.15 「パフォーマンススキーマシステム変数」

performance_schema_show_processlist

セクション27.12.19.9 「processlist テーブル」

セクション27.15 「パフォーマンススキーマシステム変数」

performance_schema_users_size

セクション27.12.8.3 「users テーブル」

セクション27.12.18.12 「ステータス変数サマリーテーブル」

セクション27.12.15 「パフォーマンススキーマのステータス変数のテーブル」

セクション27.15 「パフォーマンススキーマシステム変数」

persist_only_admin_x

セクション5.1.8 「サーバーシステム変数」

セクション5.1.9.1 「システム変数権限」

セクション5.1.9.4 「永続的で永続的に制限されないシステム変数」

persisted_globals_load

セクション13.7.8.7 「RESET PERSIST ステートメント」

セクション4.2.2.2 「オプションファイルの使用」

セクション5.1.8 「サーバーシステム変数」

セクション27.12.14.2 「パフォーマンススキーマ variables_info テーブル」

セクション5.1.9.3 「永続化されるシステム変数」

セクション5.1.9.4 「永続的で永続的に制限されないシステム変数」

pid

セクション15.8.2 「読み取り専用操作の InnoDB の構成」

pid_file

セクション1.3 「MySQL 8.0 の新機能」

セクション2.4.3 「MySQL 起動デーモンのインストールおよび使用」

セクション5.1.8 「サーバーシステム変数」

plugin_dir

セクション6.4.2.1 「Connection-Control プラグインのインストール」

セクション5.6.5.1 「ddl_rewriter のインストールまたはアンインストール」

セクション26.22 「INFORMATION_SCHEMA PLUGINS テーブル」

セクション15.20.3 「InnoDB memcached プラグインの設定」

セクション13.7.4.3 「INSTALL COMPONENT ステートメント」

セクション13.7.4.4 「INSTALL PLUGIN ステートメント」

セクション6.4.1.7 「LDAP プラガブル認証」

セクション6.4.5.2 「MySQL Enterprise Audit のインストールまたはアンインストール」

セクション6.5.2 「MySQL Enterprise Data Masking and De-Identification のインストールまたはアンインストール」

セクション6.6.1 「MySQL Enterprise Encryption のインストール」

セクション2.4.3 「MySQL 起動デーモンのインストールおよび使用」

セクション2.5.4 「Oracle の RPM パッケージを使用した Linux への MySQL のインストール」

セクション6.4.1.5 「PAM プラガブル認証」

セクション13.7.7.25 「SHOW PLUGINS ステートメント」

セクション27.12.19.12 「user_defined_functions テーブル」

セクション6.4.1.6 「Windows プラガブル認証」

セクション6.4.4.12 「キーリングコマンドのオプション」

セクション6.4.4.1 「キーリングプラグインのインストール」

セクション5.6.7.1 「クローンプラグインのインストール」

セクション5.1.7 「サーバーコマンドオプション」

セクション5.1.8 「サーバーシステム変数」

セクション5.6.3.2 「スレッドプールのインストール」

セクション6.4.1.9 「ソケットピア資格証明プラガブル認証」

セクション5.6.6.2 「バージョントークンのインストールまたはアンインストール」
セクション6.1.2.2 「パスワードセキュリティについての管理者ガイドライン」
セクション6.4.3.1 「パスワード検証コンポーネントのインストールおよびアンインストール」
セクション16.11.1 「プラグブルストレージエンジンのアーキテクチャー」
セクション6.2.17 「プラグブル認証」
セクション6.4.1.10 「プラグブル認証のテスト」
セクション5.6.1 「プラグインのインストールおよびアンインストール」
セクション13.7.4.1 「ユーザー定義関数用の CREATE FUNCTION ステートメント」
セクション6.4.1.8 「ログインなしのプラグブル認証」
ロックサービス UDF インタフェース
セクション6.1.3 「攻撃者に対する MySQL のセキュアな状態の維持」
セクション6.4.4.10 「汎用キーリングキー管理関数」
セクション17.4.10.2 「準同期レプリケーションのインストールと構成」
セクション6.4.6 「監査メッセージコンポーネント」

port

セクションB.3.2.2 「[ローカルの] MySQL サーバーに接続できません」
セクション13.7.5 「CLONE ステートメント」
セクション20.3.1 「MySQL Shell」
セクション20.4.1 「MySQL Shell」
セクション6.7.5.1 「mysqld の TCP ポートコンテキストの設定」
セクション27.12.11.10 「replication_group_members テーブル」
セクション20.5.6.2 「X プラグイン のオプションとシステム変数」
セクション18.10 「よくある質問」
セクション18.8 「グループレプリケーションシステム変数」
セクション18.2.1.2 「グループレプリケーション用のインスタンスの構成」
セクション5.1.8 「サーバーシステム変数」
セクション5.6.7.3 「リモートデータのクローニング」
セクション18.4.3.1 「分散リカバリの接続」
分散リカバリエンドポイントのアドレスの選択
セクション18.5.3 「分散リカバリ接続の保護」

preload_buffer_size

セクション5.1.8 「サーバーシステム変数」

print_identified_with_as_hex

セクション13.7.1.1 「ALTER USER ステートメント」
セクション13.7.1.3 「CREATE USER ステートメント」
セクション13.7.7.12 「SHOW CREATE USER ステートメント」
セクション6.2.8 「アカウントの追加、権限の割当ておよびアカウントの削除」
セクション5.1.8 「サーバーシステム変数」

profiling

セクション26.24 「INFORMATION_SCHEMA PROFILING テーブル」
セクション13.7.7.30 「SHOW PROFILE ステートメント」
セクション5.1.8 「サーバーシステム変数」

profiling_history_size

セクション13.7.7.30 「SHOW PROFILE ステートメント」
セクション5.1.8 「サーバーシステム変数」

protocol_compression_algorithms

セクション13.4.2.1 「CHANGE MASTER TO ステートメント」
セクション13.4.2.3 「CHANGE REPLICATION SOURCE TO ステートメント」
セクション4.5.1.1 「mysql クライアントオプション」
セクション4.4.5 「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
セクション4.5.2 「mysqldadmin — A MySQL Server 管理プログラム」

セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」
セクション4.5.3 「mysqlcheck — テーブル保守プログラム」
セクション4.5.4 「mysqldump — データベースバックアッププログラム」
セクション4.5.5 「mysqlimport — データインポートプログラム」
セクション4.5.6 「mysqlpump — データベースバックアッププログラム」
セクション4.5.7 「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション4.5.8 「mysqlslap — ロードエミュレーションクライアント」
セクション20.5.5 「X プラグイン での接続圧縮」
セクション18.8 「グループレプリケーションシステム変数」
セクション4.2.3 「サーバーに接続するためのコマンドオプション」
セクション5.1.8 「サーバーシステム変数」
セクション5.1.10 「サーバーステータス変数」
セクション4.2.8 「接続圧縮制御」

protocol_version

セクション5.1.8 「サーバーシステム変数」
セクション5.1.9.4 「永続的で永続的に制限されないシステム変数」

proxy_user

セクション5.1.8 「サーバーシステム変数」
セクション6.2.18 「プロキシユーザー」

pseudo_slave_mode

セクション4.6.8 「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」
セクション5.1.8 「サーバーシステム変数」
セクション17.3.3.1 「レプリケーション PRIVILEGE_CHECKS_USER アカウントの権限」

pseudo_thread_id

セクション5.1.8 「サーバーシステム変数」
セクション17.5.1.39 「レプリケーションと変数」
セクション17.3.3 「レプリケーション権限チェック」
セクション12.16 「情報関数」
セクション5.4.4.3 「混合形式のバイナリロギング形式」

Q

[\[index top\]](#)

query_alloc_block_size

セクション5.1.8 「サーバーシステム変数」

query_prealloc_size

セクション5.1.8 「サーバーシステム変数」

R

[\[index top\]](#)

rand_seed

セクション5.1.8 「サーバーシステム変数」

range_alloc_block_size

セクション5.1.8 「サーバーシステム変数」

range_optimizer_max_mem_size

セクション4.5.1.6 「mysql クライアントのヒント」

[セクション8.2.1.2「rangeの最適化」](#)
[セクション5.1.8「サーバーシステム変数」](#)

rbr_exec_mode

[セクション5.1.8「サーバーシステム変数」](#)

read_buffer_size

[セクション8.12.3.1「MySQLのメモリーの使用方法」](#)
[セクション8.6.3「REPAIR TABLE ステートメントの最適化」](#)
[セクション5.1.8「サーバーシステム変数」](#)

read_only

[セクション13.7.1.1「ALTER USER ステートメント」](#)
[セクション13.7.1.2「CREATE ROLE ステートメント」](#)
[セクション13.7.1.3「CREATE USER ステートメント」](#)
[セクション13.7.1.4「DROP ROLE ステートメント」](#)
[セクション13.7.1.5「DROP USER ステートメント」](#)
[セクション13.7.1.6「GRANT ステートメント」](#)
[セクション17.1.3.4「GTIDを使用したレプリケーションのセットアップ」](#)
[セクション8.2.3「INFORMATION_SCHEMA クエリーの最適化」](#)
[セクション6.2.2「MySQLで提供される権限」](#)
[セクション13.7.1.7「RENAME USER ステートメント」](#)
[セクション13.7.1.8「REVOKE ステートメント」](#)
[セクション13.7.1.10「SET PASSWORD ステートメント」](#)
[セクション13.3.1「START TRANSACTION、COMMIT および ROLLBACK ステートメント」](#)
[セクション6.2.14「アカウントパスワードの割り当て」](#)
[セクション13.7.1「アカウント管理ステートメント」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション17.4.1.3「ソースまたはレプリカを読み取り専用にするによるバックアップ」](#)
[セクション17.5.1.39「レプリケーションと変数」](#)
[セクション5.4.1「一般クエリーログおよびスロークエリーログの出力先の選択」](#)

read_rnd_buffer_size

[セクション8.2.1.11「Multi-Range Readの最適化」](#)
[セクション8.12.3.1「MySQLのメモリーの使用方法」](#)
[セクション8.2.1.16「ORDER BYの最適化」](#)
[セクション5.1.8「サーバーシステム変数」](#)

regex_stack_limit

[セクション1.3「MySQL 8.0の新機能」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション12.8.2「正規表現」](#)

regex_time_limit

[セクション1.3「MySQL 8.0の新機能」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション12.8.2「正規表現」](#)

relay

[セクション17.2.2.3「起動オプションとレプリケーションチャンネル」](#)

relay_log

[セクション1.3「MySQL 8.0の新機能」](#)
[セクション17.1.6.3「Replica Serverのオプションと変数」](#)
[セクション18.10「よくある質問」](#)
[セクション17.2.4.1「リレーログ」](#)

[セクション17.2.2.4「レプリケーションチャネルのネーミング規則」](#)
[セクション17.4.7「レプリケーションパフォーマンスを改善する」](#)
[セクション17.1.2.8「レプリケーション環境へのレプリカの追加」](#)

relay_log_basename

[セクション17.1.6.3「Replica Server のオプションと変数」](#)

relay_log_index

[セクション17.1.6.3「Replica Server のオプションと変数」](#)
[セクション17.2.4.1「リレーログ」](#)
[セクション17.1.2.8「レプリケーション環境へのレプリカの追加」](#)

relay_log_info_file

[セクション17.1.6.3「Replica Server のオプションと変数」](#)
[セクション17.2.4.2「レプリケーションメタデータリポジトリ」](#)

relay_log_info_repository

[セクション1.3「MySQL 8.0 の新機能」](#)
[セクション17.1.6.3「Replica Server のオプションと変数」](#)
[セクション13.4.2.5「RESET REPLICAS | SLAVE ステートメント」](#)
[セクション18.9.1「グループレプリケーションの要件」](#)
[セクション17.4.2「レプリカの予期しない停止の処理」](#)
[セクション17.2.4.2「レプリケーションメタデータリポジトリ」](#)
[セクション5.6.7.6「レプリケーション用のクローニング」](#)
[セクション17.2.2.3「起動オプションとレプリケーションチャネル」](#)

relay_log_purge

[セクション17.1.6.3「Replica Server のオプションと変数」](#)
[セクション13.7.7.35「SHOW REPLICAS | SLAVE STATUS ステートメント」](#)

relay_log_recovery

[セクション17.1.6.3「Replica Server のオプションと変数」](#)
[セクション17.4.2「レプリカの予期しない停止の処理」](#)

relay_log_space_limit

[セクション17.1.6.3「Replica Server のオプションと変数」](#)
[セクション8.14.5「レプリケーション I/O スレッドの状態」](#)
[セクション17.2.2.3「起動オプションとレプリケーションチャネル」](#)

replication_optimize_for_static_plugin_config

[セクション17.1.6.3「Replica Server のオプションと変数」](#)
[セクション17.4.10.1「準同期レプリケーション管理インタフェース」](#)

replication_sender_observe_commit_only

[セクション17.1.6.3「Replica Server のオプションと変数」](#)
[セクション17.4.10.1「準同期レプリケーション管理インタフェース」](#)

report_host

[セクション17.1.6.3「Replica Server のオプションと変数」](#)
[セクション18.2.2「グループレプリケーションのローカルでのデプロイ」](#)
[セクション18.2.1.2「グループレプリケーション用のインスタンスの構成」](#)
[セクション18.4.3.1「分散リカバリの接続」](#)

report_password

[セクション17.1.6.3「Replica Server のオプションと変数」](#)

report_port

セクション17.1.6.3「Replica Server のオプションと変数」
セクション18.10「よくある質問」
セクション18.8「グループレプリケーションシステム変数」
セクション18.4.3.1「分散リカバリの接続」
分散リカバリエンドポイントのアドレスの選択

report_user

セクション17.1.6.3「Replica Server のオプションと変数」

require_row_format

セクション5.1.8「サーバーシステム変数」

require_secure_transport

セクション6.8「FIPS のサポート」
セクション20.5.3「X プラグイン での暗号化接続の使用」
セクション5.1.8「サーバーシステム変数」
セクション4.2.7「接続トランスポートプロトコル」
セクション6.3「暗号化された接続の使用」
セクション6.3.1「暗号化接続を使用するための MySQL の構成」

resultset_metadata

セクション5.1.8「サーバーシステム変数」

rewriter_enabled

リライタのクエリーリライトプラグインのシステム変数
セクション5.6.4.2「リライタクエリーリライトプラグインの使用」

rewriter_verbose

リライタのクエリーリライトプラグインのシステム変数

rpl_read_size

セクション8.12.3.1「MySQL のメモリーの使用方法」
セクション17.1.6.3「Replica Server のオプションと変数」
セクション17.4.7「レプリケーションパフォーマンスを改善する」

rpl_semi_sync_master_enabled

セクション17.1.6.2「レプリケーションソースのオプションと変数」
セクション17.4.10.2「準同期レプリケーションのインストールと構成」
セクション17.4.10.3「準同期レプリケーションモニタリング」
セクション17.4.10.1「準同期レプリケーション管理インタフェース」

rpl_semi_sync_master_timeout

セクション17.1.6.2「レプリケーションソースのオプションと変数」
セクション17.4.10.2「準同期レプリケーションのインストールと構成」
セクション17.4.10.1「準同期レプリケーション管理インタフェース」

rpl_semi_sync_master_trace_level

セクション17.1.6.3「Replica Server のオプションと変数」
セクション17.1.6.2「レプリケーションソースのオプションと変数」

rpl_semi_sync_master_wait_for_slave_count

セクション17.1.6.2「レプリケーションソースのオプションと変数」
セクション17.4.10「準同期レプリケーション」

rpl_semi_sync_master_wait_no_slave

セクション17.1.6.2「レプリケーションソースのオプションと変数」

rpl_semi_sync_master_wait_point

セクション17.1.6.2「レプリケーションソースのオプションと変数」

セクション17.4.10「準同期レプリケーション」

rpl_semi_sync_slave_enabled

セクション17.1.6.3「Replica Server のオプションと変数」

セクション17.4.10.2「準同期レプリケーションのインストールと構成」

セクション17.4.10.1「準同期レプリケーション管理インタフェース」

rpl_semi_sync_slave_trace_level

セクション17.1.6.3「Replica Server のオプションと変数」

rpl_stop_slave_timeout

セクション17.1.6.3「Replica Server のオプションと変数」

セクション13.4.2.9「STOP REPLICAS | SLAVE ステートメント」

セクション17.5.1.34「レプリケーションとトランザクションの非一貫性」

S

[\[index top\]](#)

schema_definition_cache

セクション5.1.8「サーバーシステム変数」

セクション14.4「ディクショナリオブジェクトキャッシュ」

secondary_engine_cost_threshold

セクション5.1.8「サーバーシステム変数」

secure_file_priv

セクション13.2.5「IMPORT TABLE ステートメント」

セクション13.2.7「LOAD DATA ステートメント」

セクション6.5.3「MySQL Enterprise Data Masking and De-Identification の使用」

セクション6.5.5「MySQL Enterprise Data Masking and De-Identification ユーザー定義関数の説明」

セクション6.2.2「MySQL で提供される権限」

セクション2.9.7「MySQL ソース構成オプション」

セクション2.5.4「Oracle の RPM パッケージを使用した Linux への MySQL のインストール」

セクション15.6.5「redo ログ」

セクション13.2.10.1「SELECT ... INTO ステートメント」

セクション6.7.4「SELinux ファイルコンテキスト」

セクション5.1.8「サーバーシステム変数」

セクション2.10.1「データディレクトリの初期化」

セクション6.1.3「攻撃者に対する MySQL のセキュアな状態の維持」

セクション12.8「文字列関数および演算子」

select_into_buffer_size

セクション1.3「MySQL 8.0 の新機能」

セクション13.2.10.1「SELECT ... INTO ステートメント」

セクション5.1.8「サーバーシステム変数」

select_into_disk_sync

セクション1.3「MySQL 8.0 の新機能」

セクション13.2.10.1「SELECT ... INTO ステートメント」

セクション5.1.8「サーバーシステム変数」

select_into_disk_sync_delay

セクション1.3「MySQL 8.0 の新機能」

セクション13.2.10.1「SELECT ... INTO ステートメント」

セクション5.1.8「サーバーシステム変数」

server_id

2 番目のインスタンスの追加

セクション17.1.3.4「GTID を使用したレプリケーションのセットアップ」

セクション4.6.8「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」

NDB Cluster システム変数

セクション23.6.11「NDB Cluster レプリケーションの競合解決」

セクション17.1.6.3「Replica Server のオプションと変数」

セクション13.7.7.35「SHOW REPLICAS | SLAVE STATUS ステートメント」

セクション12.24「その他の関数」

セクション18.9.1「グループレプリケーションの要件」

セクション18.2.1.2「グループレプリケーション用のインスタンスの構成」

セクション17.1.6.4「バイナリロギングのオプションと変数」

セクション5.4.4「バイナリログ」

セクション17.1.1「バイナリログファイルの位置ベースのレプリケーション構成の概要」

セクション27.12.11「パフォーマンススキーマレプリケーションテーブル」

セクション17.1.2.2「レプリカ構成の設定」

セクション17.1.6「レプリケーションおよびバイナリロギングのオプションと変数」

セクション17.5.4「レプリケーションのトラブルシューティング」

セクション17.1.6.2「レプリケーションソースのオプションと変数」

セクション17.1.2.1「レプリケーションソース構成の設定」

セクション17.1.2.8「レプリケーション環境へのレプリカの追加」

セクション6.4.5.4「監査ログファイル形式」

セクション17.2.1.2「行ベースロギングおよびレプリケーションの使用」

高度なオプション

server_id_bits

NDB Cluster システム変数

server_uuid

セクション13.4.2.1「CHANGE MASTER TO ステートメント」

セクション13.4.2.3「CHANGE REPLICATION SOURCE TO ステートメント」

セクション17.1.3.6「GTID のないソースから GTID のあるレプリカへのレプリケーション」

セクション17.1.3.8「GTID を操作するストアドファンクションの例」

セクション17.1.3.1「GTID 形式および格納」

セクション27.12.19.7「log_status テーブル」

セクション27.12.11.4「replication_applier_configuration テーブル」

セクション27.12.11.2「replication_connection_status テーブル」

セクション18.3.2「replication_group_members テーブル」

セクション13.7.7.35「SHOW REPLICAS | SLAVE STATUS ステートメント」

セクション13.4.2.7「START REPLICAS | SLAVE ステートメント」

セクション18.4.1.1「グループプライマリメンバーの変更」

セクション18.4.1.2「グループモードの変更」

セクション18.4.6「グループレプリケーションでの MySQL Enterprise Backup の使用」

セクション13.4.3.3「グループレプリケーションプライマリを構成する機能」

セクション17.3.2.2「バイナリログ暗号化キー」

セクション27.12.11「パフォーマンススキーマレプリケーションテーブル」

プライマリ選択アルゴリズム

セクション17.1.6「レプリケーションおよびバイナリロギングのオプションと変数」

session_track_gtid

セクション5.1.18「クライアントセッション状態の変更のサーバートラッキング」

セクション17.1.6.5「グローバルトランザクション ID システム変数」
セクション5.1.8「サーバーシステム変数」
セクション17.1.4.1「レプリケーションモードの概念」

session_track_schema

セクション5.1.18「クライアントセッション状態の変更のサーバートラッキング」
セクション5.1.8「サーバーシステム変数」

session_track_state_change

セクション5.1.18「クライアントセッション状態の変更のサーバートラッキング」
セクション5.1.8「サーバーシステム変数」

session_track_system_variables

セクション5.1.18「クライアントセッション状態の変更のサーバートラッキング」
セクション5.1.8「サーバーシステム変数」

session_track_transaction_info

セクション5.1.18「クライアントセッション状態の変更のサーバートラッキング」
セクション5.1.8「サーバーシステム変数」

sha

セクション6.3.3.1「MySQL を使用した SSL および RSA 証明書とキーの作成」
セクション6.4.1.3「SHA-256 プラガブル認証」
セクション5.1.8「サーバーシステム変数」
セクション5.1.10「サーバーステータス変数」
セクション6.2.18「プロキシユーザー」

shared_memory

セクション1.2.2「MySQL の主な機能」
セクション4.5.1.1「mysql クライアントオプション」
セクション4.4.5「mysql_upgrade — MySQL テーブルのチェックとアップグレード」
セクション4.5.2「mysqladmin — A MySQL Server 管理プログラム」
セクション4.6.8「mysqlbinlog — バイナリログファイルを処理するためのユーティリティー」
セクション4.5.3「mysqlcheck — テーブル保守プログラム」
セクション4.5.4「mysqldump — データベースバックアッププログラム」
セクション4.5.5「mysqlexport — データインポートプログラム」
セクション4.5.7「mysqlshow — データベース、テーブル、およびカラム情報の表示」
セクション4.5.8「mysqlslap — ロードエミュレーションクライアント」
セクション5.8.2.1「Windows コマンド行での複数の MySQL インスタンスの起動」
セクション4.2.4「コマンドオプションを使用した MySQL Server への接続」
セクション4.2.3「サーバーに接続するためのコマンドオプション」
セクション2.3.4.5「サーバーをはじめて起動する」
セクション5.1.8「サーバーシステム変数」
タイプとネットワーキング

shared_memory_base_name

セクション5.8.2.1「Windows コマンド行での複数の MySQL インスタンスの起動」
セクション5.1.8「サーバーシステム変数」

show_create_table_skip_secondary_engine

セクション4.5.4「mysqldump — データベースバックアッププログラム」
セクション5.1.8「サーバーシステム変数」

show_create_table_verbosity

セクション5.1.8「サーバーシステム変数」

show_old_temporals

セクション5.1.8 「サーバーシステム変数」

skip_external_locking

セクション5.1.8 「サーバーシステム変数」

セクション8.11.5 「外部ロック」

skip_name_resolve

セクション5.1.12.3 「DNS ルックアップとホストキャッシュ」

セクション6.2.21 「MySQL への接続の問題のトラブルシューティング」

セクション2.3.4.9 「MySQL インストールのテスト」

セクション18.10 「よくある質問」

セクション5.1.8 「サーバーシステム変数」

セクション2.10.1 「データディレクトリの初期化」

セクション5.1.14 「ネットワークネームスペースのサポート」

skip_networking

セクションB.3.2.2 「[ローカルの] MySQL サーバーに接続できません」

セクション5.1.12.3 「DNS ルックアップとホストキャッシュ」

セクションA.14 「MySQL 8.0 FAQ: レプリケーション」

セクション6.2.21 「MySQL への接続の問題のトラブルシューティング」

セクションB.3.2.7 「MySQL サーバーが存在しなくなりました」

セクション23.5.14.41 「ndbinfo processes テーブル」

root のパスワードのリセット: 一般的な手順

セクション20.5.6.3 「X プラグイン ステータス変数」

セクション5.1.7 「サーバーコマンドオプション」

セクション5.1.8 「サーバーシステム変数」

セクション6.2.17 「プラグイン認証」

セクション17.5.4 「レプリケーションのトラブルシューティング」

セクション17.5.3 「レプリケーションセットアップをアップグレードする」

セクション17.1.2.1 「レプリケーションソース構成の設定」

skip_show_database

セクション5.1.7 「サーバーコマンドオプション」

セクション5.1.8 「サーバーシステム変数」

slave

セクション17.1.3.6 「GTID のないソースから GTID のあるレプリカへのレプリケーション」

セクション17.2.2.3 「起動オプションとレプリケーションチャンネル」

slave_allow_batching

セクション23.6.5 「NDB Cluster のレプリケーションの準備」

セクション23.1.4 「NDB Cluster の新機能」

NDB Cluster システム変数

セクション23.6.6 「NDB Cluster レプリケーションの開始 (シングルレプリケーションチャンネル)」

slave_checkpoint_group

セクション23.6.3 「NDB Cluster レプリケーションの既知の問題」

セクション17.1.6.3 「Replica Server のオプションと変数」

セクション12.24 「その他の関数」

セクション17.2.2.3 「起動オプションとレプリケーションチャンネル」

slave_checkpoint_period

セクション17.1.6.3 「Replica Server のオプションと変数」

セクション12.24 「その他の関数」

slave_compressed_protocol

セクション13.4.2.1「CHANGE MASTER TO ステートメント」
セクション13.4.2.3「CHANGE REPLICATION SOURCE TO ステートメント」
セクション1.3「MySQL 8.0 の新機能」
セクション17.1.6.3「Replica Server のオプションと変数」
バイナリログのトランザクション圧縮のモニタリング
セクション4.2.8「接続圧縮制御」

slave_exec_mode

セクション23.6.3「NDB Cluster レプリケーションの既知の問題」
セクション17.1.6.3「Replica Server のオプションと変数」
セクション17.5.1.21「レプリケーションと MEMORY テーブル」
セクション17.2.1.2「行ベースロギングおよびレプリケーションの使用」

slave_load_tmpdir

セクションB.3.3.5「MySQL が一時ファイルを格納する場所」
セクション17.1.6.3「Replica Server のオプションと変数」
セクション5.1.7「サーバーコマンドオプション」
セクション5.1.8「サーバーシステム変数」
セクション7.2「データベースバックアップ方法」
セクション17.4.1.2「レプリカからの RAW データのバックアップ」

slave_max_allowed_packet

セクション17.1.6.3「Replica Server のオプションと変数」
セクション18.9.2「グループレプリケーションの制限事項」
セクション18.8「グループレプリケーションシステム変数」
バイナリログトランザクション圧縮が有効な場合の動作
セクション18.6.4「メッセージの断片化」
セクション17.5.1.20「レプリケーションと max_allowed_packet」

slave_net_timeout

セクション13.4.2.1「CHANGE MASTER TO ステートメント」
セクション13.4.2.3「CHANGE REPLICATION SOURCE TO ステートメント」
セクション17.1.6.3「Replica Server のオプションと変数」
セクション5.1.8「サーバーシステム変数」
セクション8.14.5「レプリケーション I/O スレッドの状態」
セクション17.5.1.28「レプリケーションとソースまたはレプリカの停止」
セクション17.1.7.1「レプリケーションステータスの確認」

slave_parallel_type

セクション17.1.6.3「Replica Server のオプションと変数」
セクション18.9.1「グループレプリケーションの要件」
セクション17.1.6.4「バイナリロギングのオプションと変数」
バイナリログトランザクション圧縮が有効な場合の動作
セクション17.5.1.34「レプリケーションとトランザクションの非一貫性」
セクション17.2.3.2「レプリケーションアプライアンスワーカー スレッドの監視」

slave_parallel_workers

セクション13.4.2.1「CHANGE MASTER TO ステートメント」
セクション13.4.2.3「CHANGE REPLICATION SOURCE TO ステートメント」
セクション17.1.3.2「GTID ライフサイクル」
セクション17.1.5「MySQL マルチソースレプリケーション」
セクション23.6.3「NDB Cluster レプリケーションの既知の問題」
セクション17.1.6.3「Replica Server のオプションと変数」
セクション27.12.11.7「replication_applier_status_by_worker テーブル」
セクション13.4.2.9「STOP REPLICA | SLAVE ステートメント」
セクション18.9.1「グループレプリケーションの要件」

[セクション17.1.6.4「バイナリロギングのオプションと変数」](#)
[セクション27.12.11「パフォーマンススキーマレプリケーションテーブル」](#)
[セクション8.14.6「レプリケーション SQL スレッドの状態」](#)
[セクション17.5.1.20「レプリケーションと max_allowed_packet」](#)
[セクション17.5.1.34「レプリケーションとトランザクションの非一貫性」](#)
[セクション17.2.3「レプリケーションスレッド」](#)
[セクション17.2.2「レプリケーションチャネル」](#)
[セクション5.6.7.6「レプリケーション用のクローニング」](#)

slave_pending_jobs_size_max

[セクション8.12.3.1「MySQL のメモリーの使用方法」](#)
[セクション17.1.6.3「Replica Server のオプションと変数」](#)
[セクション8.14.6「レプリケーション SQL スレッドの状態」](#)
[セクション17.5.1.20「レプリケーションと max_allowed_packet」](#)
[セクション17.2.3.2「レプリケーションアプライアンスワーカースレッドの監視」](#)

slave_preserve_commit_order

[セクション17.1.3.2「GTID ライフサイクル」](#)
[セクション17.1.6.3「Replica Server のオプションと変数」](#)
[セクション18.9.1「グループレプリケーションの要件」](#)
[セクション17.1.6.4「バイナリロギングのオプションと変数」](#)
[セクション8.14.5「レプリケーション I/O スレッドの状態」](#)
[セクション17.5.1.34「レプリケーションとトランザクションの非一貫性」](#)

slave_rows_search_algorithms

[セクション17.1.6.3「Replica Server のオプションと変数」](#)
[セクション5.1.10「サーバーステータス変数」](#)
[セクション17.5.1.27「レプリケーションおよび行検索」](#)
[セクション13.1.20.10「非表示カラム」](#)

slave_skip_errors

[セクション17.1.6.3「Replica Server のオプションと変数」](#)

slave_sql_verify_checksum

MySQL 用語集
[セクション17.1.6.3「Replica Server のオプションと変数」](#)
[セクション5.4.4「バイナリログ」](#)

slave_transaction_retries

[セクション17.1.6.3「Replica Server のオプションと変数」](#)
[セクション27.12.11.5「replication_applier_status テーブル」](#)
[セクション17.5.1.32「レプリケーション再試行とタイムアウト」](#)
[セクション17.2.2.3「起動オプションとレプリケーションチャネル」](#)

slave_type_conversions

[セクション23.6.3「NDB Cluster レプリケーションの既知の問題」](#)
[セクション17.1.6.3「Replica Server のオプションと変数」](#)

slow_launch_time

[セクション5.1.8「サーバースステム変数」](#)
[セクション5.1.10「サーバーステータス変数」](#)

slow_query_log

[セクション5.1.8「サーバースステム変数」](#)
[セクション5.4.5「スロークエリーログ」](#)
[セクション5.4.1「一般クエリーログおよびスロークエリーログの出力先の選択」](#)

slow_query_log_file

セクション1.3「MySQL 8.0の新機能」
セクション5.1.8「サーバーシステム変数」
セクション5.4.5「スロークエリログ」
セクション5.4.1「一般クエリログおよびスロークエリログの出力先の選択」

socket

セクション20.5.6.2「X プラグイン のオプションとシステム変数」
セクション5.1.8「サーバーシステム変数」

sort_buffer_size

セクション7.6.3「MyISAM テーブルの修復方法」
セクション8.2.1.16「ORDER BY の最適化」
セクション5.1.7「サーバーコマンドオプション」
セクション5.1.8「サーバーシステム変数」
セクション5.1.10「サーバーステータス変数」
セクション13.7.6.1「変数代入の SET 構文」

sql_auto_is_null

セクション13.1.20「CREATE TABLE ステートメント」
セクション5.1.8「サーバーシステム変数」
セクション5.4.4「バイナリログ」
セクション17.5.1.39「レプリケーションと変数」
セクション12.4.2「比較関数と演算子」
セクション5.4.4.3「混合形式のバイナリロギング形式」

sql_big_selects

セクション5.1.8「サーバーシステム変数」

sql_buffer_result

セクション5.1.8「サーバーシステム変数」

sql_log_bin

セクション28.4.4.2「diagnostics() プロシージャ」
セクション1.3「MySQL 8.0の新機能」
セクション6.2.2「MySQL で提供される権限」
セクション4.6.8「mysqlbinlog — バイナリログファイルを処理するためのユーティリティ」
セクション23.1.7.1「NDB Cluster の SQL 構文に準拠していません」
セクション23.6.3「NDB Cluster レプリケーションの既知の問題」
セクション23.1.7.8「NDB Cluster 専用の問題」
セクション28.4.4.12「ps_setup_reload_saved() プロシージャ」
セクション28.4.4.14「ps_setup_save() プロシージャ」
セクション28.4.4.22「ps_trace_statement_digest() プロシージャ」
セクション28.4.4.23「ps_trace_thread() プロシージャ」
セクション13.4.1.3「SET sql_log_bin ステートメント」
セクション28.4.4.25「statement_performance_analyzer() プロシージャ」
セクション5.1.9.1「システム変数権限」
セクション17.1.6.4「バイナリロギングのオプションと変数」
セクション17.5.3「レプリケーションセットアップをアップグレードする」

sql_log_off

セクション6.2.2「MySQL で提供される権限」
MySQL 用語集
セクション5.1.8「サーバーシステム変数」
セクション5.4.3「一般クエリログ」
セクション5.4.1「一般クエリログおよびスロークエリログの出力先の選択」

SQL_MODE

セクション15.12.1「オンライン DDL 操作」

sql_mode

セクション13.1.13「CREATE EVENT ステートメント」

セクション13.1.17「CREATE PROCEDURE ステートメントおよび CREATE FUNCTION ステートメント」

セクション13.1.22「CREATE TRIGGER ステートメント」

セクションB.3.4.2「DATE カラムの使用に関する問題」

セクション26.48「INFORMATION_SCHEMA VIEWS テーブル」

セクション15.1.2「InnoDB テーブルのベストプラクティス」

セクション28.4.5.7「list_add() 関数」

セクション13.2.7「LOAD DATA ステートメント」

セクション2.11.4「MySQL 8.0 での変更」

セクション1.7「MySQL の標準への準拠」

セクション13.7.7.13「SHOW CREATE VIEW ステートメント」

セクション13.6.7.5「SIGNAL ステートメント」

セクション2.11.5「アップグレード用のインストールの準備」

セクション4.2.2.2「オプションファイルの使用」

セクション5.1.11「サーバー SQL モード」

セクション5.1.8「サーバーシステム変数」

セクション5.1.9「システム変数の使用」

セクション5.4.4「バイナリログ」

セクション17.5.1.39「レプリケーションと変数」

セクション12.25.3「式の処理」

セクション5.4.4.3「混合形式のバイナリロギング形式」

セクション1.6「質問またはバグをレポートする方法」

sql_notes

セクション13.6.7.7「MySQL の診断領域」

セクション13.7.7.42「SHOW WARNINGS ステートメント」

セクションB.2「エラー情報インタフェース」

セクション5.1.8「サーバーシステム変数」

sql_quote_show_create

セクション13.7.7.6「SHOW CREATE DATABASE ステートメント」

セクション13.7.7.10「SHOW CREATE TABLE ステートメント」

セクション5.1.8「サーバーシステム変数」

セクション12.16「情報関数」

sql_require_primary_key

セクション13.1.9「ALTER TABLE ステートメント」

セクション13.4.2.1「CHANGE MASTER TO ステートメント」

セクション13.4.2.3「CHANGE REPLICATION SOURCE TO ステートメント」

セクション18.4.6「グループレプリケーションでの MySQL Enterprise Backup の使用」

セクション18.9.1「グループレプリケーションの要件」

セクション17.3.3.2「グループレプリケーションチャンネルの権限チェック」

セクション5.1.8「サーバーシステム変数」

セクション17.3.3.1「レプリケーション PRIVILEGE_CHECKS_USER アカウントの権限」

セクション17.3.3「レプリケーション権限チェック」

sql_safe_updates

セクション4.5.1.6「mysql クライアントのヒント」

セクション8.2.1.2「range の最適化」

セクション5.1.8「サーバーシステム変数」

sql_select_limit

セクション4.5.1.6「mysql クライアントのヒント」

セクション5.1.8 「サーバーシステム変数」

sql_slave_skip_counter

セクション17.1.3.7 「GTID ベースレプリケーションの制約」

セクション17.1.6.3 「Replica Server のオプションと変数」

SET GLOBAL sql_slave_skip_counter でのトランザクションのスキップ

セクション13.7.7.35 「SHOW REPLICA | SLAVE STATUS ステートメント」

バイナリログトランザクション圧縮が有効な場合の動作

sql_warnings

セクション5.1.8 「サーバーシステム変数」

ssl_ca

セクション6.3.3.1 「MySQL を使用した SSL および RSA 証明書とキーの作成」

セクション6.3.3.2 「openssl を使用した SSL 証明書およびキーの作成」

セクション18.5.2 「Secure Socket Layer (SSL) を使用したグループ通信接続の保護」

セクション20.5.6.2 「X プラグイン のオプションとシステム変数」

セクション4.2.3 「サーバーに接続するためのコマンドオプション」

セクション5.1.8 「サーバーシステム変数」

セクション5.1.10 「サーバーステータス変数」

セクション6.3.1 「暗号化接続を使用するための MySQL の構成」

セクション17.3.1 「暗号化接続を使用するためのレプリケーションの設定」

ssl_capath

セクション18.5.2 「Secure Socket Layer (SSL) を使用したグループ通信接続の保護」

セクション20.5.6.2 「X プラグイン のオプションとシステム変数」

セクション4.2.3 「サーバーに接続するためのコマンドオプション」

セクション5.1.8 「サーバーシステム変数」

セクション5.1.10 「サーバーステータス変数」

セクション6.3.1 「暗号化接続を使用するための MySQL の構成」

セクション17.3.1 「暗号化接続を使用するためのレプリケーションの設定」

ssl_cert

セクション6.3.3.1 「MySQL を使用した SSL および RSA 証明書とキーの作成」

セクション6.3.3.2 「openssl を使用した SSL 証明書およびキーの作成」

セクション18.5.2 「Secure Socket Layer (SSL) を使用したグループ通信接続の保護」

セクション20.5.6.2 「X プラグイン のオプションとシステム変数」

セクション4.2.3 「サーバーに接続するためのコマンドオプション」

セクション5.1.7 「サーバーコマンドオプション」

セクション5.1.8 「サーバーシステム変数」

セクション5.1.10 「サーバーステータス変数」

セクション6.3.2 「暗号化された接続 TLS プロトコルおよび暗号」

セクション6.3.1 「暗号化接続を使用するための MySQL の構成」

セクション17.3.1 「暗号化接続を使用するためのレプリケーションの設定」

ssl_cipher

セクション4.4.3 「mysql_ssl_rsa_setup — SSL/RSA ファイルの作成」

セクション18.5.2 「Secure Socket Layer (SSL) を使用したグループ通信接続の保護」

セクション20.5.6.2 「X プラグイン のオプションとシステム変数」

セクション4.2.3 「サーバーに接続するためのコマンドオプション」

セクション5.1.8 「サーバーシステム変数」

セクション5.1.10 「サーバーステータス変数」

セクション6.3.2 「暗号化された接続 TLS プロトコルおよび暗号」

セクション6.3.1 「暗号化接続を使用するための MySQL の構成」

ssl_crl

セクション18.5.2 「Secure Socket Layer (SSL) を使用したグループ通信接続の保護」

セクション20.5.6.2「X プラグイン のオプションとシステム変数」
セクション4.2.3「サーバーに接続するためのコマンドオプション」
セクション5.1.8「サーバーシステム変数」
セクション5.1.10「サーバーステータス変数」
セクション6.3.1「暗号化接続を使用するための MySQL の構成」

ssl_crlpath

セクション18.5.2「Secure Socket Layer (SSL) を使用したグループ通信接続の保護」
セクション20.5.6.2「X プラグイン のオプションとシステム変数」
セクション4.2.3「サーバーに接続するためのコマンドオプション」
セクション5.1.8「サーバーシステム変数」
セクション5.1.10「サーバーステータス変数」
セクション6.3.1「暗号化接続を使用するための MySQL の構成」

ssl_fips_mode

セクション6.8「FIPS のサポート」
セクション4.2.3「サーバーに接続するためのコマンドオプション」
セクション5.1.8「サーバーシステム変数」

ssl_key

セクション6.3.3.1「MySQL を使用した SSL および RSA 証明書とキーの作成」
セクション6.3.3.2「openssl を使用した SSL 証明書およびキーの作成」
セクション18.5.2「Secure Socket Layer (SSL) を使用したグループ通信接続の保護」
セクション20.5.6.2「X プラグイン のオプションとシステム変数」
セクション4.2.3「サーバーに接続するためのコマンドオプション」
セクション5.1.7「サーバーコマンドオプション」
セクション5.1.8「サーバーシステム変数」
セクション5.1.10「サーバーステータス変数」
セクション6.3.1「暗号化接続を使用するための MySQL の構成」
セクション17.3.1「暗号化接続を使用するためのレプリケーションの設定」

storage_engine

セクション13.1.16「CREATE LOGFILE GROUP ステートメント」

stored_program_cache

セクション5.1.8「サーバーシステム変数」
セクション14.4「ディクショナリオブジェクトキャッシュ」
セクション8.10.3「プリペアドステートメントおよびストアードプログラムのキャッシュ」

stored_program_definition_cache

セクション5.1.8「サーバーシステム変数」
セクション14.4「ディクショナリオブジェクトキャッシュ」

super_read_only

2 番目のインスタンスの追加

セクション8.2.3「INFORMATION_SCHEMA クエリーの最適化」
セクション13.4.3.1「START GROUP_REPLICATION ステートメント」
セクション13.4.3.2「STOP GROUP_REPLICATION ステートメント」
セクション18.7.1.1「アップグレード中のメンバーバージョン」
セクション18.7.3.1「オンラインアップグレードに関する考慮事項」
セクション18.7.3.3「グループレプリケーションのオンラインアップグレード方法」
セクション18.3.1「グループレプリケーションサーバーの状態」
セクション18.8「グループレプリケーションシステム変数」
セクション18.7.3.2「グループレプリケーションメンバーのアップグレード」
セクション5.1.8「サーバーシステム変数」
セクション18.1.3.1「シングルプライマリモード」
セクション18.6.6.4「終了処理」

sync_binlog

[セクション17.1.3.3「GTID 自動配置」](#)
[セクション15.2「InnoDB および ACID モデル」](#)
[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)
[セクション17.1.6.4「バイナリロギングのオプションと変数」](#)
[セクション5.4.4「バイナリログ」](#)
[セクション17.5.1.28「レプリケーションとソースまたはレプリカの停止」](#)

sync_master_info

[セクション17.1.6.3「Replica Server のオプションと変数」](#)

sync_relay_log

[セクション17.1.6.3「Replica Server のオプションと変数」](#)
[セクション17.4.2「レプリカの予期しない停止の処理」](#)

sync_relay_log_info

[セクション17.1.6.3「Replica Server のオプションと変数」](#)
[セクション17.5.1.28「レプリケーションとソースまたはレプリカの停止」](#)

syseventlog

[セクション5.1.8「サーバーシステム変数」](#)
[セクション5.4.2.8「システムログへのエラーロギング」](#)

system_time_zone

[セクション5.1.15「MySQL Server でのタイムゾーンのサポート」](#)
[セクション5.1.7「サーバーコマンドオプション」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション17.5.1.33「レプリケーションとタイムゾーン」](#)

T

[\[index top\]](#)

table_definition_cache

[セクション8.12.3.1「MySQL のメモリーの使用方法」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション14.4「ディクショナリオブジェクトキャッシュ」](#)

table_encryption_privilege_check

[セクション13.1.2「ALTER DATABASE ステートメント」](#)
[セクション13.1.9「ALTER TABLE ステートメント」](#)
[セクション13.1.10「ALTER TABLESPACE ステートメント」](#)
[セクション13.1.12「CREATE DATABASE ステートメント」](#)
[セクション13.1.20「CREATE TABLE ステートメント」](#)
[セクション13.1.21「CREATE TABLESPACE ステートメント」](#)
[セクション15.13「InnoDB 保存データ暗号化」](#)
[セクション1.3「MySQL 8.0 の新機能」](#)
[セクション6.2.2「MySQL で提供される権限」](#)
[セクション13.1.36「RENAME TABLE ステートメント」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション17.3.3.1「レプリケーション PRIVILEGE_CHECKS_USER アカウントの権限」](#)

table_open_cache

[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)
[セクション8.4.3.1「MySQL でのテーブルのオープンとクローズの方法」](#)
[セクション8.12.3.1「MySQL のメモリーの使用方法」](#)

[セクション5.1.8 「サーバーシステム変数」](#)
[セクション5.1.10 「サーバーステータス変数」](#)
[セクションB.3.2.16 「ファイルが見つからず同様のエラーが発生しました」](#)
[セクション8.14.3 「一般的なスレッドの状態」](#)

table_open_cache_instances

[セクション5.1.8 「サーバーシステム変数」](#)
[セクション5.1.10 「サーバーステータス変数」](#)
[セクション5.4.1 「一般クエリログおよびスロークエリログの出力先の選択」](#)

tablespace_definition_cache

[セクション5.1.8 「サーバーシステム変数」](#)
[セクション14.4 「ディクショナリオブジェクトキャッシュ」](#)

temptable_max_mmap

[セクション1.3 「MySQL 8.0 の新機能」](#)
[セクション8.4.4 「MySQL での内部一時テーブルの使用」](#)
[セクション5.1.8 「サーバーシステム変数」](#)

temptable_max_ram

[セクション1.3 「MySQL 8.0 の新機能」](#)
[セクション8.4.4 「MySQL での内部一時テーブルの使用」](#)
[セクション5.1.8 「サーバーシステム変数」](#)

temptable_use_mmap

[セクション1.3 「MySQL 8.0 の新機能」](#)
[セクション8.4.4 「MySQL での内部一時テーブルの使用」](#)
[セクション5.1.8 「サーバーシステム変数」](#)
[セクション5.1.10 「サーバーステータス変数」](#)

thread_cache_size

[セクション5.9.1.4 「gdb での mysqld のデバッグ」](#)
[セクション5.1.8 「サーバーシステム変数」](#)
[セクション5.1.10 「サーバーステータス変数」](#)
[セクション5.1.12.1 「接続インターフェース」](#)

thread_handling

[セクション5.1.8 「サーバーシステム変数」](#)
[セクション5.6.3.1 「スレッドプール要素」](#)

thread_pool_algorithm

[セクション27.12.16.1 「tp_thread_group_state テーブル」](#)
[セクション5.1.8 「サーバーシステム変数」](#)
[セクション5.6.3.3 「スレッドプール操作」](#)

thread_pool_high_priority_connection

[セクション5.1.8 「サーバーシステム変数」](#)
[セクション5.6.3.3 「スレッドプール操作」](#)

thread_pool_max_active_query_threads

[セクション5.1.8 「サーバーシステム変数」](#)
[セクション5.6.3.3 「スレッドプール操作」](#)

thread_pool_max_unused_threads

[セクション5.1.8 「サーバーシステム変数」](#)
[セクション5.6.3.3 「スレッドプール操作」](#)

thread_pool_prio_kickup_timer

セクション27.12.16.1「tp_thread_group_state テーブル」
セクション27.12.16.2「tp_thread_group_stats テーブル」
セクション5.1.8「サーバーシステム変数」
セクション5.6.3.4「スレッドプールのチューニング」
セクション5.6.3.3「スレッドプール操作」

thread_pool_size

セクション5.1.8「サーバーシステム変数」
セクション5.6.3.4「スレッドプールのチューニング」
セクション5.6.3.3「スレッドプール操作」

thread_pool_stall_limit

セクション27.12.16.1「tp_thread_group_state テーブル」
セクション27.12.16.2「tp_thread_group_stats テーブル」
セクション5.1.8「サーバーシステム変数」
セクション5.6.3.4「スレッドプールのチューニング」
セクション5.6.3.3「スレッドプール操作」

thread_stack

セクション8.12.3.1「MySQL のメモリーの使用方法」
セクション5.1.8「サーバーシステム変数」
セクション25.2.1「ストアルーチンの構文」
セクション5.1.12.1「接続インターフェース」

time_zone

セクション13.1.13「CREATE EVENT ステートメント」
セクション11.2.2「DATE、DATETIME、および TIMESTAMP 型」
セクション1.3「MySQL 8.0 の新機能」
セクション5.1.15「MySQL Server でのタイムゾーンのサポート」
セクション25.4.4「イベントメタデータ」
セクション5.1.7「サーバーコマンドオプション」
セクション5.1.8「サーバーシステム変数」
セクション5.4.5「スロークエリールログ」
セクション17.5.1.33「レプリケーションとタイムゾーン」
セクション17.5.1.39「レプリケーションと変数」
セクション5.4.3「一般クエリールログ」
セクション12.7「日付および時間関数」
セクション5.4.4.3「混合形式のバイナリロギング形式」

timestamp

セクション16.8.3「FEDERATED ストレージエンジンの注記とヒント」
セクション5.1.8「サーバーシステム変数」
セクション17.5.1.39「レプリケーションと変数」
セクション5.4.4.3「混合形式のバイナリロギング形式」

tls_ciphersuites

セクション18.5.2「Secure Socket Layer (SSL) を使用したグループ通信接続の保護」
セクション4.2.3「サーバーに接続するためのコマンドオプション」
セクション5.1.8「サーバーシステム変数」
セクション5.1.10「サーバーステータス変数」
セクション6.3.2「暗号化された接続 TLS プロトコルおよび暗号」
セクション6.3.1「暗号化接続を使用するための MySQL の構成」

tls_version

セクション18.5.2「Secure Socket Layer (SSL) を使用したグループ通信接続の保護」

セクション20.5.3 「X プラグイン での暗号化接続の使用」
セクション4.2.3 「サーバーに接続するためのコマンドオプション」
セクション5.1.8 「サーバーシステム変数」
セクション5.1.10 「サーバーステータス変数」
セクション6.3.2 「暗号化された接続 TLS プロトコルおよび暗号」
セクション6.3.1 「暗号化接続を使用するための MySQL の構成」
セクション17.3.1 「暗号化接続を使用するためのレプリケーションの設定」

tmp_table_size

セクション8.4.4 「MySQL での内部一時テーブルの使用」
セクション8.12.3.1 「MySQL のメモリーの使用方法」
セクション5.1.8 「サーバーシステム変数」
セクション13.6.6.5 「サーバー側のカーソルの制約」

tmpdir

セクション15.14 「InnoDB の起動オプションおよびシステム変数」
セクション8.4.4 「MySQL での内部一時テーブルの使用」
セクション2.9.7 「MySQL ソース構成オプション」
セクション8.2.1.16 「ORDER BY の最適化」
セクション17.1.6.3 「Replica Server のオプションと変数」
セクション15.12.5 「オンライン DDL 失敗条件」
セクション15.12.3 「オンライン DDL 領域の要件」
セクション5.1.8 「サーバーシステム変数」
セクション7.2 「データベースバックアップ方法」
セクションB.3.2.11 「ファイルを作成/書き込みできない」
セクション17.4.1.2 「レプリカからの RAW データのバックアップ」

transaction

セクション18.9.1 「グループレプリケーションの要件」

transaction_alloc_block_size

セクション5.1.8 「サーバーシステム変数」

transaction_allow_batching

NDB Cluster システム変数

transaction_isolation

セクション13.3.7 「SET TRANSACTION ステートメント」
セクション5.1.18 「クライアントセッション状態の変更のサーバートラッキング」
セクション5.1.7 「サーバーコマンドオプション」
セクション5.1.8 「サーバーシステム変数」

transaction_prealloc_size

セクション5.1.8 「サーバーシステム変数」

transaction_read_only

セクション8.2.3 「INFORMATION_SCHEMA クエリーの最適化」
セクション13.3.7 「SET TRANSACTION ステートメント」
セクション5.1.18 「クライアントセッション状態の変更のサーバートラッキング」
セクション5.1.7 「サーバーコマンドオプション」
セクション5.1.8 「サーバーシステム変数」

transaction_write_set_extraction

セクション18.8 「グループレプリケーションシステム変数」
セクション17.1.6.4 「バイナリロギングのオプションと変数」

U

[\[index top\]](#)

unique_checks

[セクション15.6.1.5「MyISAM から InnoDB へのテーブルの変換」](#)

[セクション5.1.8「サーバーシステム変数」](#)

[セクション5.4.4「バイナリログ」](#)

[セクション17.5.1.39「レプリケーションと変数」](#)

[セクション5.4.4.3「混合形式のバイナリロギング形式」](#)

updatable_views_with_limit

[セクション5.1.8「サーバーシステム変数」](#)

[セクション25.5.3「更新可能および挿入可能なビュー」](#)

use_secondary_engine

[セクション5.1.8「サーバーシステム変数」](#)

V

[\[index top\]](#)

validate_password

[セクション6.4.3.2「パスワード検証オプションおよび変数」](#)

[セクション6.4.3「パスワード検証コンポーネント」](#)

[セクション12.14「暗号化関数と圧縮関数」](#)

validate_password_check_user_name

[セクション6.4.3.2「パスワード検証オプションおよび変数」](#)

validate_password_dictionary_file

[セクション6.4.3.2「パスワード検証オプションおよび変数」](#)

validate_password_length

[セクション6.4.3.2「パスワード検証オプションおよび変数」](#)

validate_password_mixed_case_count

[セクション6.4.3.2「パスワード検証オプションおよび変数」](#)

validate_password_number_count

[セクション6.4.3.2「パスワード検証オプションおよび変数」](#)

validate_password_policy

[セクション6.4.3.2「パスワード検証オプションおよび変数」](#)

validate_password_special_char_count

[セクション6.4.3.2「パスワード検証オプションおよび変数」](#)

validate_user_plugins

[セクション5.1.8「サーバーシステム変数」](#)

version

[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)

[セクション5.1.8「サーバーシステム変数」](#)
[セクション12.16「情報関数」](#)
[セクション6.4.5.4「監査ログファイル形式」](#)

version_comment

[セクション2.9.7「MySQL ソース構成オプション」](#)
[セクション13.7.7.41「SHOW VARIABLES ステートメント」](#)
[セクション5.1.8「サーバーシステム変数」](#)

version_compile_machine

[セクション5.1.8「サーバーシステム変数」](#)

version_compile_os

[セクション5.1.8「サーバーシステム変数」](#)

version_compile_zlib

[セクション5.1.8「サーバーシステム変数」](#)

version_tokens_session

[セクション5.6.6.3「バージョントークンの使用」](#)
[セクション5.6.6.4「バージョントークン参照」](#)

version_tokens_session_number

[セクション5.6.6.4「バージョントークン参照」](#)

W

[\[index top\]](#)

wait_timeout

[セクションB.3.2.7「MySQL サーバーが存在しなくなりました」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション18.4.2.2「トランザクション一貫性保証の構成」](#)
[セクションB.3.2.9「通信エラーおよび中止された接続」](#)

warning_count

[セクション13.6.7.7「MySQL の診断領域」](#)
[セクション13.7.7.17「SHOW ERRORS ステートメント」](#)
[セクション13.7.7.42「SHOW WARNINGS ステートメント」](#)
[セクション13.6.7.5「SIGNAL ステートメント」](#)
[セクションB.2「エラー情報インタフェース」](#)
[セクション5.1.8「サーバーシステム変数」](#)
[セクション13.5「プリペアドステートメント」](#)

windowing_use_high_precision

[セクション8.2.1.21「ウィンドウ機能最適化」](#)
[セクション5.1.8「サーバーシステム変数」](#)

トランザクション分離レベルの索引

[R](#) | [S](#)

[R](#)

[\[index top\]](#)

READ COMMITTED

セクション27.12.7.1「events_transactions_current テーブル」
セクション15.7.3「InnoDB のさまざまな SQL ステートメントで設定されたロック」
セクション15.14「InnoDB の起動オプションおよびシステム変数」
セクション8.5.2「InnoDB トランザクション管理の最適化」
セクション15.7.1「InnoDB ロック」
セクションA.10「MySQL 8.0 FAQ: NDB Cluster」
セクションA.1「MySQL 8.0 FAQ: 全般」
セクション1.3「MySQL 8.0 の新機能」
セクション23.1.7.3「NDB Cluster でのトランザクション処理に関する制限」
セクション23.3.3.6「NDB Cluster データノードの定義」
セクション23.1.6.3「NDB および InnoDB の機能使用のサマリー」
セクション23.1.6.1「NDB および InnoDB ストレージエンジンの違い」
セクション13.3.7「SET TRANSACTION ステートメント」
セクション13.1.37「TRUNCATE TABLE ステートメント」
セクション18.9.2「グループレプリケーションの制限事項」
セクション15.7.5.3「デッドロックを最小化および処理する方法」
セクション15.7.2.1「トランザクション分離レベル」
セクション5.4.4.2「バイナリログ形式の設定」
セクション15.7.2.3「一貫性非ロック読み取り」

READ UNCOMMITTED

セクション27.12.7.1「events_transactions_current テーブル」
セクション15.20.2「InnoDB memcached のアーキテクチャー」
セクション15.14「InnoDB の起動オプションおよびシステム変数」
セクション23.1.7.3「NDB Cluster でのトランザクション処理に関する制限」
セクション13.3.7「SET TRANSACTION ステートメント」
セクション13.1.37「TRUNCATE TABLE ステートメント」
セクション15.7.2.1「トランザクション分離レベル」
セクション5.4.4.2「バイナリログ形式の設定」
セクション15.20.6.6「ベースとなる InnoDB テーブルでの DML および DDL ステートメントの実行」
セクション15.7.2.3「一貫性非ロック読み取り」
永続統計計算への削除マーク付きレコードの組込み

READ-COMMITTED

セクション13.3.7「SET TRANSACTION ステートメント」
セクション5.1.7「サーバーコマンドオプション」

READ-UNCOMMITTED

セクション13.3.7「SET TRANSACTION ステートメント」
セクション5.1.7「サーバーコマンドオプション」

REPEATABLE READ

セクション27.12.7.1「events_transactions_current テーブル」
セクション15.20.6.4「InnoDB memcached プラグインのトランザクション動作の制御」
セクション15.14「InnoDB の起動オプションおよびシステム変数」
セクション8.5.2「InnoDB トランザクション管理の最適化」
セクション15.7.1「InnoDB ロック」
セクション1.3「MySQL 8.0 の新機能」
セクション4.5.4「mysqldump — データベースバックアッププログラム」
セクション4.5.6「mysqlpump — データベースバックアッププログラム」
セクション23.1.7.3「NDB Cluster でのトランザクション処理に関する制限」
セクション13.3.7「SET TRANSACTION ステートメント」
セクション13.3.1「START TRANSACTION、COMMIT および ROLLBACK ステートメント」
セクション13.3.8「XA トランザクション」
セクション18.9.2「グループレプリケーションの制限事項」
セクション15.7.2.1「トランザクション分離レベル」

[セクション27.12.7「パフォーマンススキーマのトランザクションテーブル」](#)
[セクション15.7.2.3「一貫性非ロック読み取り」](#)
[セクション5.4.4.3「混合形式のバイナリロギング形式」](#)

REPEATABLE-READ

[セクション13.3.7「SET TRANSACTION ステートメント」](#)
[セクション5.1.7「サーバーコマンドオプション」](#)
[セクション5.1.8「サーバーシステム変数」](#)

S

[\[index top\]](#)

SERIALIZABLE

[セクション27.12.7.1「events_transactions_current テーブル」](#)
[セクション15.7.3「InnoDB のさまざまな SQL ステートメントで設定されたロック」](#)
[セクション15.14「InnoDB の起動オプションおよびシステム変数」](#)
[セクション15.7.1「InnoDB ロック」](#)
[セクション23.1.7.3「NDB Cluster でのトランザクション処理に関する制限」](#)
[セクション13.3.7「SET TRANSACTION ステートメント」](#)
[セクション13.3.1「START TRANSACTION、COMMIT および ROLLBACK ステートメント」](#)
[セクション13.3.8「XA トランザクション」](#)
[セクション18.9.2「グループレプリケーションの制限事項」](#)
[セクション5.1.7「サーバーコマンドオプション」](#)
[セクション15.7.2.1「トランザクション分離レベル」](#)
[セクション27.12.7「パフォーマンススキーマのトランザクションテーブル」](#)
[セクション5.4.4.3「混合形式のバイナリロギング形式」](#)

MySQL 用語集

これらの用語は、MySQL データベースサーバーに関する情報で一般的に使用されます。この用語集は、InnoDB ストレージエンジンに関する用語のリファレンスとして作成され、大部分の定義は InnoDB 関連です。

.ARM ファイル

ARCHIVE テーブルのメタデータ。 .ARZ ファイルと対比してください。この拡張子を持つファイルは常に、MySQL Enterprise Backup 製品の `mysqlbackup` コマンドで生成されるバックアップに含まれます。
[.ARM ファイル](#), [MySQL Enterprise Backup](#), [mysqlbackup コマンド](#)も参照

.ARZ ファイル

ARCHIVE テーブルのデータ。 .ARM ファイルと対比してください。この拡張子を持つファイルは常に、MySQL Enterprise Backup 製品の `mysqlbackup` コマンドで生成されるバックアップに含まれます。
[.ARM ファイル](#), [MySQL Enterprise Backup](#), [mysqlbackup コマンド](#)も参照

.cfg ファイル

InnoDB トランスポータブルテーブルスペース機能で使用するメタデータファイル。これは、コマンド `FLUSH TABLES ... FOR EXPORT` で生成され、1 つまたは複数のテーブルを、別のサーバーにコピーできる一貫した状態にします。 .cfg ファイルは、対応する .ibd ファイルとともにコピーされ、 `ALTER TABLE ... IMPORT TABLESPACE` ステップ中に space ID などの .ibd ファイルの内部値を調整するために使用されます。
[.ibd ファイル](#), [スペース ID](#), [トランスポータブルテーブルスペース](#)も参照

.frm ファイル

MySQL テーブルのメタデータ (テーブル定義など) を含むファイル。 .frm ファイルは MySQL 8.0 で削除されましたが、以前の MySQL リリースでは引き続き使用されています。 MySQL 8.0 では、以前に .frm ファイルに格納されたデータはデータディクショナリテーブルに格納されます。
[データディクショナリ](#), [MySQL Enterprise Backup](#), [システムテーブルスペース](#)も参照

.ibd ファイル

file-per-table テーブルスペースおよび一般テーブルスペースのデータファイル。 File-per-table テーブルスペースの .ibd ファイルには、単一のテーブルおよび関連するインデックスデータが含まれます。 一般テーブルスペース .ibd ファイルには、複数のテーブルのテーブルおよびインデックスデータを含めることができます。

.ibd ファイル拡張子は、1 つ以上の ibdata ファイルで構成されるシステムテーブルスペースには適用されません。

file-per-table テーブルスペースまたは一般テーブルスペースが `DATA DIRECTORY =` 句を使用して作成された場合、.ibd ファイルは通常のデータディレクトリ外の指定されたパスにあります。

MySQL Enterprise Backup 製品によって .ibd ファイルが圧縮バックアップに含まれるとき、圧縮版は .ibz ファイルです。
[データベース](#), [file-per-table](#), [一般テーブルスペース](#), [ibdata ファイル](#), [.ibz ファイル](#), [innodb_file_per_table](#), [MySQL Enterprise Backup](#), [システムテーブルスペース](#)も参照

.ibz ファイル

MySQL Enterprise Backup 製品が圧縮バックアップを実行するときに、これは、file-per-table 設定を使用して作成される各テーブルスペースファイルを、.ibd 拡張子から .ibz 拡張子に変換します。

バックアップ中に適用される圧縮は、通常操作中にテーブルデータを圧縮されたままにする圧縮行フォーマットとは異なります。圧縮バックアップ操作では、すでに圧縮行フォーマットであるテーブルスペースについては圧縮ステップをスキップします。2 回目の圧縮では、バックアップ速度を低下させるだけで、ほとんどまたはまったく領域を節約しないためです。
[圧縮バックアップ](#), [圧縮行フォーマット](#), [file-per-table](#), [.ibd ファイル](#), [MySQL Enterprise Backup](#), [テーブルスペース](#)も参照

.MRG ファイル

MERGE ストレージエンジンで使用される、ほかのテーブルへの参照を含むファイル。この拡張子を持つファイルは常に、MySQL Enterprise Backup 製品の `mysqlbackup` コマンドで生成されるバックアップに含まれます。
[MySQL Enterprise Backup](#), [mysqlbackup コマンド](#)も参照

.MYD ファイル

MySQL が MyISAM テーブルのデータを格納するために使用するファイル。

[.MYI ファイル](#), [MySQL Enterprise Backup](#), [mysqlbackup コマンド](#)も参照

.MYI ファイル

MySQL が [MyISAM](#) テーブルのインデックスを格納するために使用するファイル。

[.MYD ファイル](#), [MySQL Enterprise Backup](#), [mysqlbackup コマンド](#)も参照

.NET

[ADO.NET](#), [ASP.net](#), [Connector/NET](#), [モノ](#), [Visual Studio](#)も参照

.OPT ファイル

データベース構成情報を含むファイル。この拡張子のファイルは、MySQL Enterprise Backup 製品の [mysqlbackup](#) コマンドによって生成されたバックアップに含まれます。

[MySQL Enterprise Backup](#), [mysqlbackup コマンド](#)も参照

.par ファイル

パーティション定義を含むファイル。この拡張子のファイルは、MySQL Enterprise Backup 製品の [mysqlbackup](#) コマンドによって生成されたバックアップに含まれます。

MySQL 5.7.6 での [InnoDB](#) テーブルのネイティブパーティション化サポートの導入により、パーティション化された [InnoDB](#) テーブルの [.par](#) ファイルは作成されなくなりました。パーティション化された [MyISAM](#) テーブルでは、MySQL 5.7 の [.par](#) ファイルが引き続き使用されます。MySQL 8.0 では、パーティション分割のサポートは [InnoDB](#) ストレージエンジンによってのみ提供されます。そのため、[.par](#) ファイルは MySQL 8.0 の時点では使用されなくなりました。

[MySQL Enterprise Backup](#), [mysqlbackup コマンド](#)も参照

2

2 フェーズコミット

XA 仕様に基づく分散型トランザクションの一部である操作。(2PC と略記されることがあります。)複数のデータベースがトランザクションに参加する場合、すべてのデータベースが変更をコミットするか、すべてのデータベースが変更をロールバックします。

[コミット](#), [ロールバック](#), [トランザクション](#), [XA](#)も参照

A

ACID

原子性 (atomicity)、一貫性 (consistency)、分離性 (isolation)、持続性 (durability) を表す頭字語。これらの特性はすべてデータベースシステムで望ましく、すべてトランザクションの概念に密接に結び付けられています。[InnoDB](#) のトランザクション機能は ACID の原則に準拠しています。

トランザクションは、コミットまたはロールバックできる原子的な作業単位です。トランザクションによってデータベースに複数の変更が行われた場合、トランザクションがコミットされるとすべての変更が完了し、トランザクションがロールバックされるとすべての変更が元に戻されます。

データベースは常に一貫性のある状態のままです - 各コミットまたはロールバックの後、およびトランザクションの進行中。関連データが複数のテーブルにわたって更新されている場合、クエリーは、古い値と新しい値の混合ではなく、すべて古い値が、すべて新しい値のどちらかを見ます。

トランザクションは進行中、互いから保護 (分離) されます。それらは互いに干渉できず、互いのコミットされていないデータを見ることはできません。この分離性は、ロックメカニズムを通じて実現します。経験豊富なユーザーは、実際にトランザクションが互いに干渉しないと確信できれば、パフォーマンスと並列性の向上の代わりに保護の低下をトレードオフするように分離レベルを調整できます。

トランザクションの結果は持続的です。コミット操作が成功すると、そのトランザクションによって行われた変更は、データベース以外の多くのアプリケーションが脆弱である停電、システムのクラッシュ、競合状況、またはほかの潜在的な危険から保護されます。持続性には通常、ディスクストレージへの書き込みがかかっており、書き込み操作中の停電またはソフトウェアクラッシュに対して保護するために一定の冗長性を備えています。(InnoDB では、二重書き込みバッファは永続性を支援します。)

[原子的](#), [コミット](#), [並列性](#), [二重書き込みバッファ](#), [分離レベル](#), [ロック](#), [ロールバック](#), [トランザクション](#)も参照

ADO.NET

ASP.NET などの .NET テクノロジーを使用して構築されたアプリケーション用のオブジェクトリレーショナルマッピング (ORM) フレームワーク。このようなアプリケーションは、Connector/NET コンポーネントを介して MySQL とインタフェースできます。

[.NET](#), [ASP.net](#), [Connector/NET](#), [モノ](#), [Visual Studio](#)も参照

AIO

非同期 I/O (Asynchronous I/O) の頭字語。この頭字語は、[InnoDB](#) メッセージまたはキーワードに表示される場合があります。

[非同期 I/O](#)も参照

ANSI

ODBC では、文字セットおよびその他の国際化の側面をサポートする代替方法。Unicode と対比してください。Connector/ODBC 3.51 は ANSI ドライバですが、Connector/ODBC 5.1 は Unicode ドライバです。

[Connector/ODBC](#), [ODBC](#), [Unicode](#)も参照

API

API は、client プログラムから MySQL プロトコルおよび MySQL リソースへの低レベルのアクセスを提供します。Connector によって提供される上位レベルのアクセスと対比してください。

[C API](#), [クライアント](#), [コネクタ](#), [ネイティブ C API](#), [Perl API](#), [PHP API](#), [Python API](#), [Ruby API](#)も参照

ASP.net

.NET テクノロジーおよび言語を使用して web ベースのアプリケーションを開発するためのフレームワーク。このようなアプリケーションは、Connector/NET コンポーネントを介して MySQL とインタフェースできます。

MySQL を使用してサーバー側の web ページを作成する別のテクノロジーは、PHP です。

[.NET](#), [ADO.NET](#), [Connector/NET](#), [モノ](#), [PHP](#), [Visual Studio](#)も参照

B

B ツリー

データベースインデックスに一般的に使用されるツリーデータ構造。この構造は、常にソートされ続け、正確な一致 (等号演算子) および範囲 (大なり、小なり、[BETWEEN](#) 演算子など) の高速ルックアップを可能にします。このタイプのインデックスは、[InnoDB](#) や [MyISAM](#) などのほとんどのストレージエンジンで使用できます。

B ツリーノードには多くの子を含むことができるので、B ツリーは、ノードごとに 2 つの子に限られているバイナリツリーと同じではありません。

[MEMORY](#) ストレージエンジンでのみ使用可能なハッシュインデックスと対比してください。[MEMORY](#) ストレージエンジンでは B ツリーインデックスも使用でき、一部のクエリーで範囲演算子が使用されている場合は、[MEMORY](#) テーブルに B ツリーインデックスを選択する必要があります。

B ツリーという用語の使用は、インデックス設計の一般クラスへの参照として意図されています。MySQL ストレージエンジンで使用される B ツリー構造は、従来の B ツリー設計には存在しないため、バリエーションとみなされることがあります。関連情報は、[MySQL Internals Manual](#) の [InnoDB Page Structure](#) 「[ファイルヘッダー](#)」のセクションを参照してください。

[ハッシュインデックス](#)も参照

binlog

バイナリログファイルの非公式名。たとえば、電子メールメッセージやフォーラムディスカッションでこの略語を見ることがあります。

[バイナリログ](#)も参照

BLOB

任意のサイズの任意の種類のバイナリデータを含むオブジェクトの SQL データ型 ([TINYBLOB](#), [BLOB](#), [MEDIUMBLOB](#) および [LONGBLOB](#))。MySQL テーブル内の行およびカラムに簡単に分解できないドキュメント、イメージ、サウンドファイルおよびその他の種類の情報を格納するために使用されます。MySQL アプリケーション内で BLOB を処理する手法は、Connector および API ごとに異なります。MySQL [Connector/ODBC](#) では、[BLOB](#) 値を [LONGVARBINARY](#) として定義します。文字データの大規模で自由形式のコレクションの場合、業界用語は MySQL [TEXT](#) データ型で表される CLOB です。

[API](#), [CLOB](#), [コネクタ](#), [Connector/ODBC](#)も参照

C

C

移植性とパフォーマンスを組み合わせ、低レベルのハードウェア機能にアクセスし、オペレーティングシステム、ドライバ、およびその他の種類のシステムソフトウェアを記述するための一般的な選択を可能にするプログラミング言語。C で記述された複雑なアプリケーション、言語および再利用可能なモジュールの多くは、他の言語で記述された高レベルのコンポーネントと結び付けられています。そのコア構文は、C++、Java および C#の開発者によく知られています。

[C API](#), [C++](#), [C#](#), [Java](#)も参照

C API

C API コードは、MySQL とともに配布されます。これは libmysqlclient ライブラリに含まれ、C プログラムがデータベースにアクセスできるようにします。

[API](#), [C](#), [libmysqlclient](#)も参照

C#

強力な型付け機能とオブジェクト指向機能を組み合わせたプログラミング言語で、Microsoft .NET フレームワークまたはそれに対応するオープンソースのモノ内で実行されます。多くの場合、ASP.net フレームワークを使用したアプリケーションの作成に使用されます。この構文は、C、C++ および Java 開発者によく知られています。

[.NET](#), [ASP.net](#), [C](#), [Connector/.NET](#), [C++](#), [Java](#), [モノ](#)も参照

C++

C 開発者に精通したコア構文を持つプログラミング言語。高レベルのデータ型、オブジェクト指向機能およびガベージコレクションと組み合わせて、パフォーマンスのための低レベルの操作へのアクセスを提供します。MySQL 用の C++ アプリケーションを作成するには、Connector/C++ コンポーネントを使用します。

[C](#), [Connector/C++](#)も参照

CLOB

任意のサイズの任意の種類の文字データを含むオブジェクトの SQL データ型 ([TINYTEXT](#), [TEXT](#), [MEDIUMTEXT](#) または [LONGTEXT](#))。関連付けられた文字セットおよび照合順序とともにテキストベースのドキュメントを格納するために使用されます。MySQL アプリケーション内で CLOB を処理する手法は、Connector および API ごとに異なります。MySQL Connector/ODBC では、[TEXT](#) 値を [LONGVARCHAR](#) として定義します。バイナリデータを格納する場合、BLOB 型に相当します。

[API](#), [BLOB](#), [コネクタ](#), [Connector/ODBC](#)も参照

Connector/C++

Connector/C++ 8.0 を使用すると、[document store](#) を実装する MySQL サーバーにアクセスしたり、SQL クエリーを使用して従来の方法でアクセスできます。これにより、X DevAPI を使用した C++ アプリケーションの開発、または X DevAPI for C を使用したプレーン C アプリケーションの開発が可能になります。また、Connector/C++ 1.1 から従来の JDBC ベース API を使用する C++ アプリケーションの開発も可能です。詳細は、[MySQL Connector/C++ 8.0 Developer Guide](#)を参照してください。

[クライアント](#), [コネクタ](#), [JDBC](#)も参照

Connector/J

Java プログラミング言語で開発された client アプリケーションの接続を提供する JDBC ドライバ。MySQL Connector/J は JDBC タイプ 4 ドライバです: MySQL クライアントライブラリに依存しない MySQL プロトコルのピュア Java 実装。詳細は、[MySQL Connector/J 8.0 Developer Guide](#)を参照してください。

[クライアント](#), [クライアントライブラリ](#), [コネクタ](#), [Java](#), [JDBC](#)も参照

Connector/.NET

C#、.NET、モノ、Visual Studio、ASP.net、および ADO.net などのアプリケーションを記述する開発者向けの MySQL コネクタ。

[ADO.NET](#), [ASP.net](#), [コネクタ](#), [C#](#), [モノ](#), [Visual Studio](#)も参照

Connector/ODBC

業界標準の Open Database Connectivity (ODBC) API を使用して MySQL データベースへのアクセスを提供する MySQL ODBC ドライバのファミリ。以前の名前は MyODBC ドライバです。詳細は、[MySQL Connector/ODBC Developer Guide](#)を参照してください。

[コネクタ](#), [ODBC](#)も参照

Connector/PHP

Windows オペレーティングシステム用に最適化された [mysql](#) および [mysqli](#) APIs for PHP のバージョン。

[コネクタ](#), [PHP](#), [PHP API](#)も参照

CPU バウンド

ワークロードの種類の一つ。主なボトルネックがメモリー内の CPU 操作であるもの。通常、バッファプール内にすべての結果をキャッシュできる、読み取り中心の操作を含みます。

[ボトルネック](#), [バッファプール](#), [ワークロード](#)も参照

CRUD

「作成、読取り、更新、削除」の頭字語。データベースアプリケーションでの一般的な一連の操作です。多くの場合、どの言語でもすばやく実装でき、比較的単純にデータベースを使用するタイプのアプリケーション (基本的な DDL、DML、および SQL のクエリーステートメント) を示します。

[DDL](#), [DML](#), [クエリー](#), [SQL](#)も参照

D

DCL

データ制御言語 (Data Control Language)。権限を管理するための SQL ステートメントのセット。MySQL では、[GRANT](#) および [REVOKE](#) ステートメントから構成されます。DDL および DML と対比してください。

[DDL](#), [DML](#), [SQL](#)も参照

DDEX プロバイダ

Visual Studio 内のデータ設計ツールを使用して、MySQL データベース内のスキーマおよびオブジェクトを操作できる機能。Connector/NET を使用する MySQL アプリケーションの場合、MySQL Visual Studio プラグインは MySQL 5.0 以降で DDEX プロバイダとして機能します。

[Visual Studio](#)も参照

DDL

データ定義言語 (Data Definition Language)。個々のテーブル行ではなくデータベース自体を操作するための SQL ステートメントのセット。[CREATE](#)、[ALTER](#)、および [DROP](#) ステートメントのすべての形式を含みます。[TRUNCATE](#) ステートメントも含まれます。[DELETE FROM table_name](#) ステートメントと動作が異なるためです (最終的な効果は似ていますが)。

DDL ステートメントは自動的に現在のトランザクションをコミットします。それらをロールバックすることはできません。

[InnoDB online DDL](#) 機能により、[CREATE INDEX](#)、[DROP INDEX](#)、および多くのタイプの [ALTER TABLE](#) 操作のパフォーマンスが向上します。詳細は、[セクション15.12「InnoDB とオンライン DDL」](#)を参照してください。また、[InnoDB file-per-table](#) 設定は、[DROP TABLE](#) および [TRUNCATE TABLE](#) 操作の動作に影響を与える可能性があります。

DML および DCL と対比してください。

[コミット](#), [DCL](#), [DML](#), [file-per-table](#), [ロールバック](#), [SQL](#), [トランザクション](#)も参照

DML

データ操作言語。[INSERT](#)、[UPDATE](#) および [DELETE](#) 操作を実行するための一連の SQL ステートメント。[SELECT](#) ステートメントが DML ステートメントと見なされる場合があります。[SELECT ... FOR UPDATE](#) 形式が、[INSERT](#)、[UPDATE](#)、および [DELETE](#) と同じ、ロックに関する考慮事項に従うためです。

[InnoDB](#) テーブルの DML ステートメントはトランザクションのコンテキストで動作するため、committed またはロールバック済を単一ユニットとして使用できます。

DDL および DCL と対比してください。

[コミット](#), [DCL](#), [DDL](#), [ロック](#), [ロールバック](#), [SQL](#), [トランザクション](#)も参照

DSN

「データベースソース名」の頭字語。これは、Connector/ODBC 内の connection 情報のエンコーディングです。詳細は、[Configuring a Connector/ODBC DSN on Windows](#)を参照してください。これは、Connector/NET で使用される接続文字列と同等です。

[接続](#), [接続文字列](#), [Connector/NET](#), [Connector/ODBC](#)も参照

E

Eiffel

多くのオブジェクト指向機能を含むプログラミング言語。その概念の一部は、Java および C#の開発者によく知られています。オープンソース Eiffel API for MySQL については、[セクション29.13「MySQL Eiffel ラッパー」](#)を参照してください。

[API, C#, Java](#)も参照

F

failover

障害発生時にスタンバイサーバーに自動的に切り替える機能。MySQL コンテキストでは、フェイルオーバーにはスタンバイデータベースサーバーが含まれます。多くの場合、アプリケーションサーバーまたはフレームワークによって J2EE 環境内でサポートされます。

[Connector/J, J2EE](#)も参照

file-per-table

[innodb_file_per_table](#) オプションによって制御される設定の一般名。これは、[InnoDB](#) ファイルの記憶域、機能の可用性および I/O 特性の側面に影響を与える重要な構成オプションです。MySQL 5.6.7 では、[innodb_file_per_table](#) はデフォルトで有効になっています。

[innodb_file_per_table](#) オプションを有効にすると、システムテーブルスペースの共有 `ibdata` ファイルではなく、独自の `.ibd` ファイルにテーブルを作成できます。テーブルデータが個々の `.ibd` ファイルに格納されている場合、データ compression などの機能に必要な行フォーマットを柔軟に選択できます。`TRUNCATE TABLE` 操作も高速であり、[InnoDB](#) 用に予約されている残りの領域ではなく、オペレーティングシステムで再利用された領域を使用できます。

MySQL Enterprise Backup 製品は、テーブルがその独自のファイルに格納されるので柔軟性が高くなります。たとえば、テーブルをバックアップから排除できますが、これは個別のファイルに格納されている場合に限られます。したがって、この設定は、あまり頻繁にはバックアップされない、または異なるスケジュールでバックアップされるテーブルに適しています。

[圧縮行フォーマット](#), [圧縮](#), [ファイル形式](#), [.ibd ファイル](#), [ibdata ファイル](#), [innodb_file_per_table](#), [MySQL Enterprise Backup](#), [行フォーマット](#), [システムテーブルスペース](#)も参照

FOREIGN KEY 制約

外部キー関係を通じてデータベース一貫性を維持するタイプの制約。ほかの種類の制約と同様に、データが一貫性を失った場合にデータが挿入または更新されるのを防止できます。ここでは、複数のテーブル内のデータ間の一貫性が失われることが防止されます。または、DML 操作が実行されるときに [FOREIGN KEY](#) 制約によって、外部キー作成時に指定された [ON CASCADE](#) オプションに基づいて、子行内のデータが削除されたり、別の値に変更されたり、NULL に設定されたりします。[子テーブル](#), [制約](#), [DML](#), [外部キー](#), [NULL](#)も参照

FTS

ほとんどのコンテキストで、全文検索 (Full-Text Search) の頭字語。パフォーマンスディスカッションでは、フルテーブルスキャン (Full Table Scan) の頭字語の場合があります。

[テーブルの完全スキャン](#), [全文検索](#)も参照

FULLTEXT インデックス

MySQL 全文検索メカニズムでの検索インデックスを保持する、特殊なインデックス。ストップワードとして指定されたものを飛ばして、カラムの値からの単語を表します。もともとは、利用できるのは [MyISAM](#) テーブルだけでした。MySQL 5.6.4 以降では、[InnoDB](#) テーブルでも利用できます。

[全文検索](#), [インデックス](#), [InnoDB](#), [検索インデックス](#), [ストップワード](#)も参照

G

GA

「「一般に使用可能」」は、ソフトウェア製品がベータから退出し、販売、公式サポートおよび本番で使用できる段階です。

[ベータ](#)も参照

GAC

「「グローバルアセンブリキャッシュ」」の頭字語。`.NET` システムにライブラリ (アセンブリ) を格納するための中心的な領域。物理的にはネストされたフォルダで構成され、`.NET CLR` によって単一の仮想フォルダとして扱われます。

[.NET](#), [アセンブリ](#)も参照

Glassfish

[J2EE](#)も参照

GUID

「「グローバル一意別子」」の頭字語。異なるデータベース、言語、オペレーティングシステムなど間でデータを関連付けるために使用できる ID 値。(同じ値が異なるテーブルやデータベースなどで異なるデータを参照する場合がある、順次整数を使用するかわりに使用します。) 古い MySQL バージョンでは、[BINARY\(16\)](#) として表されていました。現在、[CHAR\(36\)](#) として表されています。MySQL には、GUID 値を文字形式で返す [UUID\(\)](#) 関数と、GUID 値を整数形式で返す [UUID_SHORT\(\)](#) 関数があります。連続する GUID 値は必ずしも昇順ではないため、大きな InnoDB テーブルの主キーとして使用すると効率的な値ではありません。

H

HDD

「「ハードディスクドライブ」」の頭字語。SSD と対比されることが多く、スピニングプラッターを使用するストレージメディアを指します。そのパフォーマンス特性はディスクベースワークロードのスループットに影響を与えることがあります。

[ディスクベース](#), [SSD](#)も参照

I

I/O バウンド

[ディスクバウンド](#)も参照

ib-file セット

MySQL データベース内で InnoDB によって管理される一連のファイル: システムテーブルスペース、file-per-table テーブルスペースファイルおよびredo ログファイル。MySQL のバージョンおよび InnoDB の構成によっては、一般テーブルスペース、一時テーブルスペースおよびundo テーブルスペースファイルも含まれる場合があります。この用語は、MySQL データベース内の InnoDB によって管理される一連のファイルを指すために、InnoDB のファイル構造および形式の詳細な説明で使用されることがあります。

[データベース](#), [file-per-table](#), [一般テーブルスペース](#), [Redo ログ](#), [システムテーブルスペース](#), [一時テーブルスペース](#), [Undo テーブルスペース](#)も参照

ibbackup_logfile

ホットバックアップ操作中に MySQL Enterprise Backup 製品により作成される補助的なバックアップファイル。ここには、バックアップの実行中に行われたデータ変更に関する情報が含まれます。[ibbackup_logfile](#) などの初期バックアップファイルは、バックアップ操作中に行われた変更がまだ組み込まれていないので、raw バックアップと呼ばれます。raw バックアップファイルへの適用ステップを実行したあと、結果として得られるファイルは、最終データ変更を含んでおり、準備されたバックアップと呼ばれます。この段階で、[ibbackup_logfile](#) ファイルは必要なくなります。

[適用](#), [ホットバックアップ](#), [MySQL Enterprise Backup](#), [準備されたバックアップ](#), [raw バックアップ](#)も参照

ibdata ファイル

InnoDB システムテーブルスペースを構成する、[ibdata1](#)、[ibdata2](#) などの名前を持つファイルのセット。システムテーブルスペースの [ibdata](#) ファイルに存在する構造およびデータの詳細は、[セクション15.6.3.1「システムテーブルスペース」](#) を参照してください。

[ibdata](#) ファイルの増加は、[innodb_autoextend_increment](#) 構成オプションの影響を受けます。

[変更バッファ](#), [データディクショナリ](#), [二重書き込みバッファ](#), [file-per-table](#), [.ibd ファイル](#), [innodb_file_per_table](#), [システムテーブルスペース](#), [Undo ログ](#)も参照

ibtmp ファイル

圧縮されていない InnoDB 一時テーブルおよび関連オブジェクトの InnoDB 一時テーブルスペース データファイル。構成ファイルオプション [innodb_temp_data_file_path](#) を使用すると、ユーザーは一時テーブルスペースデータファイルの相対パスを定義できます。[innodb_temp_data_file_path](#) が指定されない場合のデフォルト動作は、データディレクトリ内に [ibdata1](#) と一緒に、[ibtmp1](#) という名前の単一自動拡張 12M バイトデータファイルを作成することです。

[データファイル](#), [一時テーブル](#), [一時テーブルスペース](#)も参照

ib_logfile

通常は [ib_logfile0](#) および [ib_logfile1](#) という名前が付けられ、Redo ログを形成するファイルのセット。ロググループと呼ばれることもあります。これらのファイルには、InnoDB テーブルのデータを変更しようとするステートメントが記録されま

す。これらのステートメントは、クラッシュ後の起動時に、未完了のトランザクションで書き込まれたデータを修正するために自動的に再現されます。

このデータは手動リカバリには使用できません。このタイプの操作には、バイナリログを使用してください。
[バイナリログ](#)、[ロググループ](#)、[Redo ログ](#)も参照

ilist

[InnoDB FULLTEXT インデックス](#)内では、トークン (特定の単語) のドキュメント ID と位置情報で構成されるデータ構造。
[FULLTEXT インデックス](#)も参照

INFORMATION_SCHEMA

MySQL データディクショナリへのクエリーインタフェースを提供するデータベースの名前。(この名前は ANSI SQL 標準で定義されています。) データベースに関する情報 (メタデータ) を調べるには、構造化されていない出力を返す [SHOW](#) コマンドを使用する代わりに、[INFORMATION_SCHEMA.TABLES](#) や [INFORMATION_SCHEMA.COLUMNS](#) などのテーブルを照会できます。

[INFORMATION_SCHEMA](#) データベースには、[InnoDB データディクショナリ](#)へのクエリーインタフェースを提供する [InnoDB](#) 固有のテーブルも含まれています。これらのテーブルは、データベースの構造を確認するのではなく、パフォーマンスの監視、チューニングおよびトラブルシューティングに役立つように、[InnoDB](#) テーブルの動作に関するリアルタイム情報を取得するために使用します。

[データディクショナリ](#)、[データベース](#)、[InnoDB](#)も参照

InnoDB

高いパフォーマンスと、信頼性、堅牢性、および同時アクセスのためのトランザクション機能とを結合する MySQL コンポーネント。これは ACID 設計概念を具体化したものです。ストレージエンジンとして表現され、[ENGINE=INNODB](#) 句で作成または変更されたテーブルを処理します。アーキテクチャーの詳細および管理手順については [第15章「InnoDB ストレージエンジン」](#)、パフォーマンスのアドバイスについては [セクション8.5「InnoDB テーブルの最適化」](#)を参照してください。

MySQL 5.5 以上では、[InnoDB](#) が新しいテーブルのデフォルトのストレージエンジンであり、[ENGINE=INNODB](#) 句は必要ありません。

[InnoDB](#) テーブルは、ホットバックアップに最適です。通常の処理を妨げることなく MySQL Server をバックアップできる MySQL Enterprise Backup 製品の詳細は、[セクション30.2「MySQL Enterprise Backup の概要」](#)を参照してください。

[ACID](#)、[ホットバックアップ](#)、[MySQL Enterprise Backup](#)、[ストレージエンジン](#)、[トランザクション](#)も参照

innodb_autoinc_lock_mode

[innodb_autoinc_lock_mode](#) オプションは、自動インクリメントロックに使用されるアルゴリズムを制御します。自動インクリメントする主キーがある場合は、[innodb_autoinc_lock_mode=1](#) 設定でのみステートメントベースレプリケーションを使用できます。この設定は、トランザクション内の複数行挿入が連続自動インクリメント値を受け取るため、連続ロックモードと呼ばれます。挿入操作でより高い並列性を許可する [innodb_autoinc_lock_mode=2](#) を使用する場合は、ステートメントベースレプリケーションではなく行ベースレプリケーションを使用してください。この設定はインターリーブロックモードと呼ばれます。これは、同時に実行される複数行の挿入ステートメントがインターリーブされる auto-increment 値を受信できるためです。互換性の目的以外は、[innodb_autoinc_lock_mode=0](#) の設定を使用しないでください。

連続ロックモード ([innodb_autoinc_lock_mode=1](#)) は、MySQL 8.0.3 より前のデフォルト設定です。MySQL 8.0.3 の時点では、インターリーブロックモード ([innodb_autoinc_lock_mode=2](#)) がデフォルトであり、デフォルトのレプリケーションタイプとしてステートメントベースから行ベースのレプリケーションへの変更が反映されます。

[自動インクリメント](#)、[自動インクリメントロック](#)、[混在モード挿入](#)、[主キー](#)も参照

innodb_file_per_table

[InnoDB](#) ファイルストレージの様々な側面、機能の可用性および I/O 特性に影響する重要な構成オプション。MySQL 5.6.7 以降ではデフォルトで有効になっています。[innodb_file_per_table](#) オプションは、file-per-table モードをオンにします。このモードを有効にすると、新しく作成された [InnoDB](#) テーブルおよび関連するインデックスをシステムテーブルスペースの外部の file-per-table .ibd ファイルに格納できます。

このオプションは、[DROP TABLE](#) や [TRUNCATE TABLE](#) など、いくつかの SQL ステートメントのパフォーマンスおよびストレージに関する考慮事項に影響します。

[innodb_file_per_table](#) オプションを有効にすると、MySQL Enterprise Backup のテーブル compression や名前付きテーブルバックアップなどの機能を利用できます。

詳細は、[innodb_file_per_table](#) および [セクション15.6.3.2「File-Per-Table テーブルスペース」](#)を参照してください。
[圧縮](#)、[file-per-table](#)、[.ibd ファイル](#)、[MySQL Enterprise Backup](#)、[システムテーブルスペース](#)も参照

innodb_lock_wait_timeout

[innodb_lock_wait_timeout](#) オプションは、共有リソースが利用できるようになるまで待機するか、または放棄してエラーを処理したり、再試行したり、アプリケーションで代替処理を行ったりするかのバランスを設定します。指定した時間を超えて lock を取得するのを待機している InnoDB トランザクションをロールバックします。特に、異なるストレージエンジンで制御される複数のテーブルへの更新によってデッドロックが発生した場合に役立ちます。このようなデッドロックは自動的に検出されません。

[デッドロック](#)、[デッドロック検出](#)、[ロック](#)、[待機](#)も参照

innodb_strict_mode

[innodb_strict_mode](#) オプションは、InnoDB が strict モードで動作するかどうかを制御します。通常は警告として処理される条件では、かわりにエラーが発生します (基礎となるステートメントは失敗します)。

[厳密モード](#)も参照

interceptor

アプリケーションの一部の側面を計測またはデバッグするためのコード。アプリケーション自体のソースを再コンパイルまたは変更せずに有効にできます。

[コマンドインタセプタ](#)、[Connector/J](#)、[Connector/NET](#)、[例外インターセプタ](#)も参照

IOPS

1 秒あたりの I/O 操作 (I/O Operations Per Second) の頭字語。ビジネシステム、特に OLTP アプリケーションの一般的な測定基準。ストレージデバイスが処理できる最大値にこの値が近い場合、アプリケーションはディスクバウンドになり、スケーラビリティを制限する場合があります。

[ディスクバウンド](#)、[OLTP](#)、[スケーラビリティ](#)も参照

J

J2EE

Java プラットフォーム、Enterprise Edition: Oracle エンタープライズ Java プラットフォーム。これは、エンタープライズクラスの Java アプリケーションの API およびランタイム環境で構成されます。詳細は、<http://www.oracle.com/technetwork/java/javaee/overview/index.html> を参照してください。MySQL アプリケーションでは、通常、Connector/J をデータベースアクセスに使用し、Tomcat や JBoss などのアプリケーションサーバーを使用して中間層の作業を処理し、オプションで Spring などのフレームワークを処理します。多くの場合、J2EE スタック内で提供されるデータベース関連の機能には、接続プールおよび failover のサポートが含まれます。

[接続プール](#)、[Connector/J](#)、[failover](#)、[Java](#)、[JBoss](#)、[Spring](#)、[Tomcat](#)も参照

Java

高性能で豊富な組み込み機能とデータ型、オブジェクト指向メカニズム、広範な標準ライブラリ、および幅広い再利用可能なサードパーティモジュールを組み合わせたプログラミング言語。エンタープライズ開発は、多くのフレームワーク、アプリケーションサーバーおよびその他のテクノロジーでサポートされています。その構文の多くは、C および C++ の開発者によく知られています。MySQL を使用して Java アプリケーションを作成するには、Connector/J と呼ばれる JDBC ドライバを使用します。

[C](#)、[Connector/J](#)、[C++](#)、[JDBC](#)も参照

JBoss

[J2EE](#)も参照

JDBC

Java アプリケーションからのデータベースアクセス用の API である「Java データベース接続」の略称。MySQL アプリケーションを作成する Java 開発者は、Connector/J コンポーネントを JDBC ドライバとして使用します。

[API](#)、[Connector/J](#)、[J2EE](#)、[Java](#)も参照

JNDI

[Java](#)も参照

K

keystore

[SSL](#)も参照

KEY_BLOCK_SIZE

圧縮形式を使用する [InnoDB](#) テーブル内のデータページのサイズを指定するオプション。デフォルトは 8K バイトです。値を小さくすると、行サイズと圧縮比率の組み合わせによって内部制限に達する恐れがあります。

[MyISAM](#) テーブルの場合、[KEY_BLOCK_SIZE](#) はオプションで、インデックスキーブロックに使用するサイズをバイト単位で指定します。この値はヒントとして扱われます。必要に応じて、異なるサイズが使用される可能性があります。個々のインデックス定義に指定された [KEY_BLOCK_SIZE](#) 値は、テーブルレベルの [KEY_BLOCK_SIZE](#) 値をオーバーライドします。
[圧縮行フォーマット](#)も参照

L

libmysql

libmysqlclient ライブラリの非公式名。

[libmysqlclient](#)も参照

libmysqlclient

[libmysqlclient.a](#) または [libmysqlclient.so](#) という名前のライブラリファイル。通常、C で記述された client プログラムにリンクされています。非公式には libmysql または mysqlclient ライブラリと呼ばれることもあります。

[クライアント](#)、[libmysql](#)、[mysqlclient](#)も参照

libmysqld

この埋込み MySQL サーバライブラリを使用すると、client アプリケーション内でフル機能の MySQL サーバを実行できます。この主なメリットは組み込みアプリケーションの速度の向上と管理の単純化です。libmysqlclient ではなく [libmysqld](#) ライブラリとリンクします。API は、これらの 3 つのライブラリすべてで同一です。

[クライアント](#)、[埋込み](#)、[libmysql](#)、[libmysqlclient](#)も参照

localhost

[接続](#)も参照

lock mode

共有 lock を使用すると、トランザクションで行を読み取ることができます。複数のトランザクションが同時にその同じ行で S ロックを獲得できます。

排他 (X) ロックでは、トランザクションは行を更新または削除できます。ほかのトランザクションは、同時にその同じ行でどのようなロックも獲得できません。

インテンションロックはテーブルに適用され、トランザクションがテーブルの行で取得するロックの種類を示すために使用されます。トランザクションごとに異なる種類のインテンションロックを同じテーブルで獲得できますが、最初のトランザクションがあるテーブルでインテンション排他 (IX) ロックを獲得すると、ほかのトランザクションはそのテーブルで S または X ロックを獲得できません。反対に、最初のトランザクションがあるテーブルでインテンション共有 (IS) ロックを獲得すると、ほかのトランザクションはそのテーブルで X ロックを獲得できません。2 フェーズプロセスは、ロックおよび互換性のある対応操作をブロックせずに、ロックリクエストを順番に解決できます。

[インテンションロック](#)、[ロック](#)、[ロック](#)、[トランザクション](#)も参照

loose_

サーバ startup の後に [InnoDB](#) 構成オプションに追加された接頭辞。そのため、現在のレベルの MySQL で認識されない新しい構成オプションは起動に失敗しません。MySQL は、このプリフィクスで始まる構成オプションを処理しますが、プリフィクスに続く部分が認識されるオプションでない場合、エラーではなく警告を返します。

[起動](#)も参照

LRU

「最低使用頻度」の頭字語。記憶域を管理するための一般的な方法です。より新しい項目をキャッシュするための領域が必要なときは、最近使用されていない項目は削除されます。[InnoDB](#) では、バッファプール内で pages を管理するために LRU メカニズムがデフォルトで使用されますが、全テーブルスキャン中など、ページが一度のみ読み取られる可能性がある場合は例外が発生します。LRU アルゴリズムのこのバリエーションはミッドポイント挿入戦略と呼ばれます。詳細は、[セクション 15.5.1 「バッファプール」](#) を参照してください。

[バッファプール](#)、[エビクション](#)、[テーブルの完全スキャン](#)、[ミッドポイント挿入戦略](#)、[ページ](#)も参照

LSN

「ログシーケンス番号」の頭字語。この任意の増加し続ける値は、Redo ログに記録される操作に対応する時点を表します。(この時点は、トランザクション境界を意識しません。1 つ以上のトランザクションの間になることがあります。)これは、クラッシュリカバリの実行中およびバッファプールの管理のために InnoDB によって内部的に使用されます。

MySQL 5.6.3 より前では、LSN は 4 バイト符号なし整数でした。LSN は、MySQL 5.6.3 で 8 バイト符号なし整数になりました。追加サイズ情報を格納するために追加バイトが必要だったので、Redo ログファイルサイズ限度が 4G バイトから 512G バイトに増加したためです。MySQL 5.6.3 以降でビルドされたアプリケーションのうち LSN 値を使用するものは、32 ビット変数ではなく 64 ビット変数を使用して LSN 値を格納および比較することをお勧めします。

MySQL Enterprise Backup 製品では、増分バックアップを取得する時点を表す LSN を指定できます。該当する LSN は、`mysqlbackup` コマンドの出力で表示されます。完全バックアップの時点に対応する LSN がわかれば、後続の増分バックアップを取得するためにその値を指定でき、その出力には次の増分バックアップのもう 1 つの LSN が含まれます。[バッファプール](#)、[クラッシュリカバリ](#)、[増分バックアップ](#)、[MySQL Enterprise Backup](#)、[Redo ログ](#)、[トランザクション](#)も参照

M

MDL

「「メタデータロック」」の頭字語。
[メタデータロック](#)も参照

memcached

多くの MySQL および NoSQL ソフトウェアスタックで広く使用されているコンポーネント。単一値を高速で読み書きでき、結果全体をメモリーにキャッシュします。アプリケーションは従来、永続的ストレージに同じデータを MySQL データベースに書き込むため、またはまだメモリーにキャッシュされていない場合は MySQL データベースからデータを読み取るために、特別なロジックを必要としていました。アプリケーションでは、多くの言語のクライアントライブラリでサポートされている単純な `memcached` プロトコルを使用して、InnoDB または NDB テーブルを使用して MySQL サーバーと直接通信できるようになりました。MySQL テーブルへのこれらの NoSQL インタフェースを使用すると、アプリケーションは SQL ステートメントを直接発行するよりも高い読取りおよび書き込みパフォーマンスを実現でき、インメモリーキャッシュ用に `memcached` がすでに組み込まれているシステムのアプリケーションロジックおよびデプロイメント構成を簡略化できます。

InnoDB テーブルへの `memcached` インタフェースは、MySQL 5.6 以上で使用できます。詳細は、[セクション 15.20 「InnoDB memcached プラグイン」](#) を参照してください。NDB テーブルへの `memcached` インタフェースは NDB Cluster 7.2 以降で使用できます。詳細は <http://dev.mysql.com/doc/ndbapi/en/ndbmemcache.html> を参照してください。
[InnoDB](#)、[NoSQL](#)も参照

MM.MySQL

MySQL 製品と統合されたときに Connector/J に進化した MySQL 用の古い JDBC ドライバ。
[Connector/J](#)も参照

mtr

[ミニトランザクション](#)も参照

MVCC

「「マルチバージョン同時実行性制御」」の頭字語。この手法により、特定の分離レベルを使用した InnoDB transactions は読取り一貫性操作を実行できます。つまり、他のトランザクションによって更新されている行をクエリーして、それらの更新が発生する前の値を確認できます。これは、ほかのトランザクションが保持しているロックのために待機することなく、クエリーが進行できるようにすることによって、並列性を高める強力な方法です。

この方法は、データベース世界で共通のものではありません。ほかのデータベース製品やほかの MySQL ストレージエンジンの中には、これをサポートしないものがあります。

[ACID](#)、[並列性](#)、[一貫性読み取り](#)、[分離レベル](#)、[ロック](#)、[トランザクション](#)も参照

my.cnf

MySQL オプションファイルの名前 (Unix または Linux システムの場合)。
[my.ini](#)、[オプションファイル](#)も参照

my.ini

MySQL オプションファイルの名前 (Windows システムの場合)。
[my.cnf](#)、[オプションファイル](#)も参照

MyODBC ドライバ

Connector/ODBC の廃止された名前。

[Connector/ODBC](#)も参照

mysql

[mysql](#) プログラムは MySQL データベース用のコマンド行インタープリターです。SQL ステートメントを処理し、[mysqld](#) デーモンにリクエストを渡すことによって [SHOW TABLES](#) などの MySQL 固有コマンドも処理します。

[mysqld](#), [SQL](#)も参照

MySQL Enterprise Backup

MySQL データベースのホットバックアップを実行するライセンス製品。[InnoDB](#) テーブルをバックアップする場合、最も効果的に柔軟性がありますが、[MyISAM](#) およびその他の種類のテーブルをバックアップすることもできます。

[ホットバックアップ](#), [InnoDB](#)も参照

mysqlbackup コマンド

MySQL Enterprise Backup 製品のコマンド行ツール。[InnoDB](#) テーブルに対してホットバックアップ操作を実行し、[MyISAM](#) およびその他の種類のテーブルに対して [warm backup](#) を実行します。このコマンドの詳細は、[セクション30.2「MySQL Enterprise Backup の概要」](#)を参照してください。

[ホットバックアップ](#), [MySQL Enterprise Backup](#), [ウォームバックアップ](#)も参照

mysqlclient

ファイル `libmysqlclient` によって実装されるライブラリの非公式名で、拡張子は `.a` または `.so` です。

[libmysqlclient](#)も参照

mysqld

[mysqld](#) (MySQL Server と呼ばれる) は、MySQL インストールでほとんどの作業を実行する単一のマルチスレッドプログラムです。追加プロセスは生成されません。MySQL Server は、データベース、テーブル、およびログファイルやステータスファイルなどのその他の情報を含む MySQL データディレクトリへのアクセスを管理します。

[mysqld](#) は、Unix デーモンまたは Windows サービスとして実行され、リクエストを常に待機し、バックグラウンドでメンテナンス作業を実行します。

[インスタンス](#), [mysql](#)も参照

MySQLdb

MySQL Python API の基礎を形成するオープンソース Python モジュールの名前。

[Python](#), [Python API](#)も参照

mysqldump

データベース、テーブル、およびテーブルデータの組み合わせの論理バックアップを実行するコマンド。結果は、元のスキーマオブジェクトまたはデータ、あるいはその両方を再現する SQL ステートメントです。相当な量のデータの場合、MySQL Enterprise Backup などの物理バックアップソリューションが高速です (特にリストア操作で)。

[論理バックアップ](#), [MySQL Enterprise Backup](#), [物理バックアップ](#), [リストア](#)も参照

N

NoSQL

データ読み書きの主要なメカニズムとして SQL 言語を使用しない、一連のデータアクセステクノロジーの一般的な用語。NoSQL テクノロジーの中には、単一値の読み取りと買い込みだけを受け入れるキー値ストアとして機能するものがあります。ACID 原理の制限を緩和するものや、事前に計画されたスキーマが不要なものもあります。MySQL ユーザーは、[memcached API](#) を使用して何らかの MySQL テーブルに直接アクセスすることにより、速度と簡略化のための NoSQL スタイル処理と、柔軟性と利便性のための SQL 操作を組み合わせることができます。[InnoDB](#) テーブルへの [memcached](#) インタフェースは、MySQL 5.6 以上で使用できます。詳細は、[セクション15.20「InnoDB memcached プラグイン」](#)を参照してください。[NDB](#) テーブルへの [memcached](#) インタフェースは [NDB Cluster 7.2](#) 以降で使用できます。[ndbmemcache—Memcache API for NDB Cluster \(NO LONGER SUPPORTED\)](#) を参照してください。

[ACID](#), [InnoDB](#), [memcached](#), [スキーマ](#), [SQL](#)も参照

NOT NULL 制約

カラムが NULL 値を含むことができないと規定するタイプの制約。これは、データベースサーバーが誤って値が失われているデータを識別できるので、参照整合性の維持に役立ちます。また、オプティマイザはそのカラムのインデックス内のエントリ数を予測できるので、クエリー最適化に関係する演算でも役立ちます。

[カラム](#), [制約](#), [NULL](#), [主キー](#), [参照整合性](#)も参照

NULL

データが存在しないことを示す、SQL での特殊な値。算術演算や等価性テストに NULL 値が含まれる場合は、それらは NULL 結果を返します。(したがって、NaN、「数値ではない」の IEEE 浮動小数点概念に似ています。) AVG() などの集計計算は、除算の分母となる行数を決定するときに、NULL 値を含む行を無視します。NULL 値を扱う唯一のテストは、SQL イディオム IS NULL または IS NOT NULL を使用します。

パフォーマンスのために、データベースでは欠落したデータ値を追跡するオーバーヘッドを最小限に抑える必要があるため、NULL の値は index 操作に関与します。NULL 値は通常、インデックスに格納されません。標準比較演算子を使用してインデックスカラムをテストするクエリーは、そのカラムの行を NULL 値に照合することはできないためです。同じ理由で、一意のインデックスは NULL 値を防止しません。これらの値は単純にインデックスに表示されません。カラムで NOT NULL 制約を宣言することで、インデックスから外れる行がないことが再保証され、クエリー最適化が向上します (行数カウントおよびインデックスを使用するかどうかの評価の精度)。

主キーはテーブル内のすべての行を一意に識別できる必要があるため、単一カラム主キーには NULL 値を含めることはできず、複数カラム主キーにはすべてのカラムで NULL 値を持つ行を含めることはできません。

Oracle データベースでは NULL 値を文字列と連結できますが、InnoDB ではこのような操作の結果を NULL として扱います。

[インデックス](#)、[主キー](#)、[SQL](#) も参照

O

ODBC

業界標準の API である Open Database Connectivity の頭字語。通常、MySQL との通信に ODBC を必要とする Windows ベースのサーバーまたはアプリケーションで使用されます。MySQL ODBC ドライバは Connector/ODBC と呼ばれます。[Connector/ODBC](#) も参照

OLTP

「オンライントランザクション処理」の頭字語。データベースシステムまたはデータベースアプリケーションの 1 つ。多数のトランザクション、頻繁な書き込みと読み取りのワークロードを実行し、通常は一度に少量のデータに影響します。たとえば、航空便予約システムや銀行預金を処理するアプリケーションがあります。DML (挿入/更新/削除) 効率性とクエリー効率性とのバランスを取るために、データは正規化形式で編成される場合があります。データウェアハウスと対比してください。

InnoDB は、行レベルロックとトランザクション機能を備え、OLT アプリケーションで使用される MySQL テーブルに理想的なストレージエンジンです。

[データウェアハウス](#)、[DML](#)、[InnoDB](#)、[クエリー](#)、[行ロック](#)、[トランザクション](#) も参照

P

page size

MySQL 5.5 までのリリースでは、各 InnoDB page のサイズは 16 キロバイトに固定されています。この値は、ほとんどの行のデータを保持できる大きさと、不要なデータをメモリーに転送するパフォーマンスオーバーヘッドを最小化できる小ささを、調和させたものです。ほかの値は未テストで、サポートされません。

MySQL 5.6 以降、InnoDB instance のページサイズは、innodb_page_size 構成オプションによって制御される 4KB、8KB または 16KB のいずれかになります。MySQL 5.7.6 では、InnoDB は 32KB および 64KB のページサイズもサポートしています。32KB および 64KB のページサイズの場合、ROW_FORMAT=COMPRESSED はサポートされず、最大レコードサイズは 16KB です。

ページサイズは、MySQL インスタンスの作成時に設定され、その後は一定のままです。システムテーブルスペース、file-per-table テーブルスペースおよび一般テーブルスペースを含むすべての InnoDB テーブルスペースに同じページサイズが適用されます。

ページサイズが小さいほど、小さなブロックサイズを使用するストレージデバイス (特に、OLTP アプリケーションなどのディスクバウンドワークロードでの SSD デバイス) のパフォーマンスに役立ちます。個々の行が更新されるときに、メモリーにコピーされたり、ディスクに書き込まれたり、再編成されたり、ロックされたりするときのデータ量が少なくなります。

[ディスクバウンド](#)、[file-per-table](#)、[一般テーブルスペース](#)、[インスタンス](#)、[OLTP](#)、[ページ](#)、[SSD](#)、[システムテーブルスペース](#)、[テーブルスペース](#) も参照

Perl

Unix スクリプトおよびレポート生成のルーツを持つプログラミング言語。高パフォーマンスの正規表現およびファイル I/O を組み込みます。CPAN などのリポジトリを介して使用可能な再利用可能なモジュールの大規模なコレクション。

[Perl API](#)も参照

Perl API

Perl 言語で記述された MySQL アプリケーション用のオープンソース API。DBI および DBD::mysql モジュールを介して実装されます。詳細は、[セクション29.9「MySQL Perl API」](#)を参照してください。

[API](#), [Perl](#)も参照

PHP

web アプリケーションからのプログラミング言語。通常、コードは web ページのソース内にブロックとして埋め込まれ、出力は Web サーバーによって送信されるときにページに置換されます。これは、web ページ全体の形式で出力を印刷する CGI スクリプトなどのアプリケーションとは対照的です。PHP 形式のコーディングは、高度に対話型で動的な web ページに使用されます。最新の PHP プログラムは、コマンドラインまたは GUI アプリケーションとして実行することもできます。

MySQL アプリケーションは、PHP APIs のいずれかを使用して記述されます。再利用可能なモジュールは、C で記述し、PHP から呼び出すことができます。

MySQL を使用してサーバー側の web ページを作成する別のテクノロジーは、ASP.net です。

[ASP.net](#), [C](#), [PHP API](#)も参照

PHP API

PHP 言語での MySQL アプリケーションの記述には、いくつかの APIs を使用できます: 元の MySQL API ([Mysql](#)) である MySQL 拡張機能 ([Mysqli](#)) である MySQL ネイティブドライバ ([Mysqli](#)) によって、MySQL 関数 ([PDO_MYSQL](#)) およびコネクタ/PHP が改善されました。詳細は、[MySQL and PHP](#)を参照してください。

[API](#), [PHP](#)も参照

PITR

ポイントインタイムリカバリ (Point-In-Time Recovery) の頭字語。

[ポイントインタイムリカバリ](#)も参照

port

データベースサーバーがリスニングする TCP/IP ソケットの番号で、connection の確立に使用されます。多くの場合、host と組み合わせて指定されます。ネットワーク暗号化の使用方法によっては、暗号化されていないトラフィック用のポートと SSL 接続用のポートが異なる場合があります。

[接続](#), [ホスト](#), [SSL](#)も参照

Pthreads

POSIX スレッド標準。Unix および Linux システムでのスレッド操作およびロック操作の API を定義します。Unix および Linux システムでは、[InnoDB](#) は mutexes に対してこの実装を使用します。

[相互排他ロック](#)も参照

Python

Unix スクリプトから大規模アプリケーションまで、幅広いフィールドで使用されるプログラミング言語。実行時の型指定、組み込みの高レベルデータ型、オブジェクト指向機能、および広範な標準ライブラリが含まれます。多くの場合、他の言語で記述されたコンポーネント間の「glue」言語として使用されます。MySQL Python API は、オープンソースの MySQLdb モジュールです。

[MySQLdb](#), [Python API](#)も参照

Python API

[API](#), [Python](#)も参照

R

R-tree

地理座標、矩形、ポリゴンなどの多次元データの空間インデックス付けに使用されるツリーデータ構造。

[B ツリー](#)も参照

RAID

「低コストのドライブの冗長アレイ」の頭字語。複数のドライブに I/O 操作を分散することにより、ハードウェアレベルで並列性が向上し、低レベル書き込み操作の効率性が改善します (そうしない場合は順番に実行されます)。
[並列性](#)も参照

raw バックアップ

バイナリログと増分バックアップに反映される変更が適用される前の、MySQL Enterprise Backup 製品によって生成されるバックアップファイルの初期セット。この段階では、ファイルはリストアする準備ができていません。これらの変更が適用されたあと、ファイルは準備されたバックアップと呼ばれます。

[バイナリログ](#)、[ホットバックアップ](#)、[ibbackup_logfile](#)、[増分バックアップ](#)、[MySQL Enterprise Backup](#)、[準備されたバックアップ](#)、[リストア](#)も参照

READ COMMITTED

分離レベルの 1 つ。パフォーマンスを向上させるために、トランザクション間の保護を一部緩和するロック戦略を使用します。トランザクションは、ほかのトランザクションからのコミットされていないデータを見ることはできませんが、現在のトランザクションが開始したあとに別のトランザクションによってコミットされるデータを見ることはできます。したがって、トランザクションは不良データを見ることはありませんが、見るデータはある程度ほかのトランザクションのタイミングに依存する場合があります。

この分離レベルのトランザクションが `UPDATE ... WHERE` または `DELETE ... WHERE` 操作を実行するときに、ほかのトランザクションは待機が必要な場合があります。このトランザクションは、ほかのトランザクションを待機させずに `SELECT ... FOR UPDATE` および `LOCK IN SHARE MODE` 操作を実行できます。

MySQL 8.0.1 の `SELECT ... LOCK IN SHARE MODE` は `SELECT ... FOR SHARE` に置き換わりませんが、`LOCK IN SHARE MODE` は下位互換性のために引き続き使用できます。

[ACID](#)、[分離レベル](#)、[ロック](#)、[REPEATABLE READ](#)、[SERIALIZABLE](#)、[トランザクション](#)も参照

READ UNCOMMITTED

トランザクション間にもっとも少ない量の保護を提供する分離レベル。クエリーは、通常は別のトランザクションを待機する状況で進行することを許可されるロック戦略を採用します。ただし、この追加パフォーマンスには、ほかのトランザクションによって変更されたけれどもまだコミットされていないデータ (ダーティー読み取りと呼ばれます) など、結果の信頼性の低いという犠牲が伴います。この分離レベルは慎重に使用してください。また、他のトランザクションが同時に実行している内容によっては、結果が一貫していないが再現できない可能性があることに注意してください。通常、この分離レベルのトランザクションはクエリーのみを実行し、挿入、更新または削除操作は実行しません。

[ACID](#)、[ダーティー読み取り](#)、[分離レベル](#)、[ロック](#)、[トランザクション](#)も参照

Redo

DML ステートメントによって InnoDB テーブルが変更されたときに redo ログに記録されるレコード単位のデータ。クラッシュリカバリ中に、不完全なトランザクションによって書き込まれるデータを訂正するために使用されます。増加し続ける LSN 値は、Redo ログを通過した Redo データの累積量を表します。

[クラッシュリカバリ](#)、[DML](#)、[LSN](#)、[Redo ログ](#)、[トランザクション](#)も参照

Redo ログ

クラッシュリカバリ中に、不完全なトランザクションによって書き込まれるデータを訂正するために使用されるディスクベースデータ構造。通常の操作では、InnoDB テーブルデータを変更するリクエストがエンコードされます。これは、NoSQL インタフェースを介した SQL ステートメントまたは低レベルの API コールの結果です。予期しないシャットダウン前にデータファイルの更新を終了していない変更は、自動的に再現されます。

Redo ログはファイルセットとして物理的に表され、通常は `ib_logfile0` および `ib_logfile1` という名前が付けられます。Redo ログ内のデータは、該当するレコードの単位でエンコードされます。このデータはまとめて Redo と呼ばれます。Redo ログをデータが通貨したことは、増加し続ける LSN 値で表されます。Redo ログの最大サイズは、最初は 4G バイトに制限されていましたが、MySQL 5.6.3 で 512G バイトに上げられています。

Redo ログのディスクレイアウトは、構成オプション `innodb_log_file_size`、`innodb_log_group_home_dir`、および (まれに) `innodb_log_files_in_group` に影響されます。Redo ログ操作のパフォーマンスは、ログバッファにも影響されます。これは `innodb_log_buffer_size` 構成オプションによって制御されます。

[クラッシュリカバリ](#)、[データファイル](#)、[ib_logfile](#)、[ログバッファ](#)、[LSN](#)、[Redo](#)、[シャットダウン](#)、[トランザクション](#)も参照

redo ログのアーカイブ

バックアップ操作の進行中にバックアップユーティリティが redo ログの生成に失敗した場合に発生する可能性のあるデータ損失を回避するために、redo ログレコードをアーカイブファイルに順次書き込む InnoDB 機能。詳細は、[redo ログのアーカイブ](#)を参照してください。

[Redo ログ](#)も参照

REPEATABLE READ

InnoDB のデフォルトの分離レベル。クエリー対象の行が他の transactions によって変更されないようにするため、反復不可能な読み取りはブロックされますが、phantom の読み取りはブロックされません。トランザクション内のすべてのクエリーが同じスナップショットからのデータを見る、つまりトランザクションが開始した時点のデータを見るように、適度に厳密なロック戦略を使用します。

この分離レベルのトランザクションが [UPDATE ... WHERE](#)、[DELETE ... WHERE](#)、[SELECT ... FOR UPDATE](#)、および [LOCK IN SHARE MODE](#) 操作を実行するときに、ほかのトランザクションは待機しなければいけない場合があります。

MySQL 8.0.1 の [SELECT ... LOCK IN SHARE MODE](#) は [SELECT ... FOR SHARE](#) に置き換わりませんが、[LOCK IN SHARE MODE](#) は下位互換性のために引き続き使用できます。

[ACID](#)、[一貫性読み取り](#)、[分離レベル](#)、[ロック](#)、[ファントム](#)、[トランザクション](#)も参照

Ruby

動的型付けおよびオブジェクト指向プログラミングを強調するプログラミング言語。一部の構文は、Perl 開発者によく知られています。MySQL アプリケーションの開発には、一般的な Ruby APIs が 2 つあります。(このマニュアルでは、上位レベルの Ruby フレームワークについては説明しません。)

[API](#)、[Perl](#)、[Ruby API](#)も参照

Ruby API

MySQL アプリケーションを開発する Ruby プログラマは、2 つの API を使用できます。MySQL/Ruby API は libmysqlclient API ライブラリに基づいています。Ruby/MySQL API はネイティブ MySQL ネットワークプロトコル (ネイティブドライバ) を使用するために書かれています。詳細は、[セクション29.11 「MySQL Ruby API」](#)を参照してください。

[libmysql](#)、[Ruby](#)も参照

rw ロック (読み書きロック)

特定のルールに従って、InnoDB が内部インメモリーデータ構造への共有アクセスロックを表現および強制するために使用する低レベルオブジェクト。InnoDB が内部インメモリーデータ構造への排他的アクセスを表すために使用する mutexes と対比してください。相互排他ロックと読み書きロックはまとめて、ラッチと呼ばれます。

[rw ロック](#)タイプには、[s ロック](#) (共有ロック)、[x ロック](#) (排他ロック)、および [sx ロック](#) (共有排他ロック) などがあります。

- [s ロック](#)は、共有リソースへの読み取りアクセスを提供します。
- [x ロック](#)は、共有リソースへの書き込みアクセスを提供し、ほかのスレッドによる不整合読み取りを許可しません。
- [sx ロック](#)は、共有リソースへの書き込みアクセスを提供し、ほかのスレッドによる不整合読み取りを許可します。[sx ロック](#)は、読み取り/書き込みワークロードの並列性を最適化し、スケラビリティを改善するために MySQL 5.7 で導入されました。

次の表に [rw ロック](#)タイプ互換性をまとめます。

	S	SX	X
S	互換	互換	競合
SX	互換	競合	競合
X	競合	競合	競合

[ラッチ](#)、[ロック](#)、[相互排他ロック](#)、[パフォーマンススキーマ](#)も参照

S

SDI

「[シリアライズされたディクショナリ情報](#)」の頭字語。

[シリアライズディクショナリ情報 \(SDI\)](#)も参照

SERIALIZABLE

最も保守的なロック戦略を使用して、このトランザクションによって読み取られたデータが他の transactions によって挿入または変更されないようにする分離レベル。この方法では、トランザクション内で同じクエリーを何度も実行でき、そのた

びに同じ結果セットを取得することを保証できます。現在のトランザクションが開始してから別のトランザクションによってコミットされたデータを変更しようとする、現在のトランザクションは待機します。

これは、SQL 標準で指定されるデフォルト分離レベルです。実際には、この強度が必要になることはほとんどないため、InnoDB のデフォルトの分離レベルは次に厳しい REPEATABLE READ です。

[ACID](#)、[一貫性読み取り](#)、[分離レベル](#)、[ロック](#)、[REPEATABLE READ](#)、[トランザクション](#)も参照

servlet

[Connector/J](#)も参照

source

レプリケーションシナリオでのデータベースサーバーマシンで、データの初期挿入、更新、および削除リクエストを処理します。これらの変更は、レプリカと呼ばれる他のサーバーに伝播され、繰り返されます。

[レプリカ](#)、[レプリケーション](#)も参照

Spring

コンポーネントを構成する方法を提供することで、アプリケーション設計を支援するために設計された Java ベースのアプリケーションフレームワーク。

[J2EE](#)も参照

SQL

データベース操作を実行するための標準である構造化クエリー言語。多くの場合、カテゴリ DDL、DML、およびクエリーに分けられます。MySQL には、レプリケーションなどのいくつかの追加ステートメントカテゴリが含まれます。SQL 構文の構成ブロックについては[第9章「言語構造」](#)、MySQL テーブルカラムに使用するデータ型については[第11章「データ型」](#)、SQL ステートメントとそれらに関連付けられるカテゴリの詳細は[第13章「SQL ステートメント」](#)、クエリーで使用する標準および MySQL 固有の関数については[第12章「関数と演算子」](#)を参照してください。

[DDL](#)、[DML](#)、[クエリー](#)、[レプリケーション](#)も参照

SQLState

Connector/J を使用するアプリケーションによる例外処理のために、JDBC 標準で定義されているエラーコード。

[Connector/J](#)、[JDBC](#)も参照

SSD

「ソリッドステートドライブ」の頭字語。従来のハードディスクドライブ (HDD) とパフォーマンス特性が異なるタイプのストレージデバイス。ストレージ容量が小さく、ランダム読み取りが高速で、可動部分がなく、書き込みパフォーマンスに影響する考慮事項がいくつかあります。そのパフォーマンス特性は、ディスクバウンドワークロードのスループットに影響を与えることがあります。

[ディスクバウンド](#)、[HDD](#)も参照

SSL

「セキュアソケットレイヤー」の頭字語。アプリケーションと MySQL データベースサーバー間のネットワーク通信用の暗号化レイヤーを提供します。

[keystore](#)、[トラストストア](#)も参照

T

Tcl

Unix スクリプトワールドからのプログラミング言語。C、C++ または Java で記述されたコードによって拡張される場合があります。オープンソース Tcl API for MySQL については、[セクション29.12「MySQL Tcl API」](#)を参照してください。

[API](#)も参照

Tomcat

オープンソースの J2EE アプリケーションサーバー。Java サーブレットおよび JavaServer Pages プログラミングテクノロジーを実装します。web サーバーと Java サーブレットコンテナで構成されます。MySQL では、通常、Connector/J とともに使用されます。

[J2EE](#)も参照

TPS

「transactions/秒」の頭字語。ベンチマークで使用されることがある測定単位です。その値は、特定のベンチマークテストで表されるワークロードと、ハードウェア能力やデータベース構成などの制御要因との組み合わせに応じて異なります。

[トランザクション](#)、[ワークロード](#)も参照

U

Undo

トランザクションの生存期間保持されるデータ。ロールバック操作の場合に元に戻せるようにすべての変更を記録します。これは、システムテーブルスペース (MySQL 5.7 以前) 内または別のundo テーブルスペース内のundo ログに格納されます。MySQL 8.0 では、undo ログはデフォルトで undo テーブルスペースに存在します。

[ロールバック](#)、[ロールバックセグメント](#)、[システムテーブルスペース](#)、[トランザクション](#)、[Undo ログ](#)、[Undo テーブルスペース](#)も参照

Undo テーブルスペース

undo テーブルスペースには、undo ログが含まれます。undo ログは、ロールバックセグメント内に含まれるundo ログセグメント内に存在します。ロールバックセグメントは、従来はシステムテーブルスペースに存在していました。MySQL 5.6 では、ロールバックセグメントは undo テーブルスペースに存在できます。MySQL 5.6 および MySQL 5.7 では、undo テーブルスペースの数は `innodb_undo_tablespaces` 構成オプションによって制御されます。MySQL 8.0 では、MySQL インスタンスの初期化時に 2 つのデフォルト undo テーブルスペースが作成され、`CREATE UNDO TABLESPACE` 構文を使用して追加の undo テーブルスペースを作成できます。

詳細は、[セクション15.6.3.4「undo テーブルスペース」](#)を参照してください。
[ロールバックセグメント](#)、[システムテーブルスペース](#)、[Undo ログ](#)、[undo ログセグメント](#)も参照

Undo バッファ

[Undo ログ](#)も参照

Undo ログ

アクティブトランザクションによって変更されたデータのコピーを保持するストレージ領域。別のトランザクションが (一貫性読み取り操作の一部として) 元のデータを見る必要がある場合に、未変更データがこのストレージ領域から取得されません。

MySQL 5.6 および MySQL 5.7 では、`innodb_undo_tablespaces` 変数を使用して、undo テーブルスペースに undo ログを配置できます。これは SSD などの別のストレージデバイスに配置できます。MySQL 8.0 では、undo ログは MySQL の初期化時に作成されるデフォルトの undo テーブルスペースに存在し、`CREATE UNDO TABLESPACE` 構文を使用して追加の undo テーブルスペースを作成できます。

Undo ログは、別個の部分、挿入 Undo バッファと更新 Undo バッファに分割されます。
[一貫性読み取り](#)、[ロールバックセグメント](#)、[SSD](#)、[システムテーブルスペース](#)、[トランザクション](#)、[Undo テーブルスペース](#)も参照

undo ログセグメント

undo ログのコレクション。undo ログセグメントはロールバックセグメント内に存在します。undo ログセグメントには、複数のトランザクションからの undo ログが含まれる場合があります。undo ログセグメントは、一度に 1 つのトランザクションでのみ使用できますが、トランザクション commit または rollback で解放された後で再利用できます。「undo セグメント」とも呼ばれます。

[コミット](#)、[ロールバック](#)、[ロールバックセグメント](#)、[Undo ログ](#)も参照

Unicode

各国語文字、文字セット、コードページおよびその他の国際化の側面を柔軟かつ標準化された方法でサポートするためのシステム。

Unicode サポートは、ODBC 標準の重要な側面です。Connector/ODBC 5.1 は Unicode ドライバであり、ANSI ドライバである Connector/ODBC 3.51 とは対照的です。

[ANSI](#)、[Connector/ODBC](#)、[ODBC](#)も参照

V

view

呼び出されたときに結果セットを生成するストアクエリー。ビューは仮想テーブルとして機能します。

Visual Studio

サポートされている Visual Studio のバージョンについては、次のリファレンスを参照してください:

- Connector/NET: [Connector/NET Versions](#)

- [Connector/C++ 8.0: Platform Support and Prerequisites](#)

[Connector/C++](#), [Connector/NET](#)も参照

X

XA

分散型トランザクションを調整するための標準インターフェース。ACID コンプライアンスを維持しながら、複数のデータベースがトランザクションに参加できます。詳細は、[セクション13.3.8「XA トランザクション」](#)を参照してください。

XA 分散トランザクションのサポートはデフォルトで有効になっています。
[ACID](#), [バイナリログ](#), [コミット](#), [トランザクション](#), [2 フェーズコミット](#)も参照

ア

アセンブリ

[Connector/NET](#) を介してアクセスされる、.NET システム内のコンパイル済コードのライブラリ。GAC に格納され、名前の競合なしでバージョンングできるようにします。

[.NET](#), [GAC](#)も参照

アトミック DDL

アトミック DDL ステートメントは、DDL 操作に関連付けられたデータディクショナリ更新、ストレージエンジン操作およびバイナリログ書き込みを単一のアトミックトランザクションに結合するステートメントです。操作中にサーバーが停止した場合でも、トランザクションは完全にコミットまたはロールバックされます。アトミック DDL サポートが MySQL 8.0 で追加されました。詳細は、[セクション13.1.1「アトミックデータ定義ステートメントのサポート」](#)を参照してください。

[バイナリログ](#), [データディクショナリ](#), [DDL](#), [ストレージエンジン](#)も参照

アトミック命令

重要な低レベル操作を中断できないようにするために、CPU によって提供される特殊な命令。

アプリケーションプログラミングインターフェース (API)

関数またはプロシージャのセット。API は、関数、プロシージャ、パラメータ、および戻り値の名前と型の安定的なセットです。

アーリーアダプタ

ソフトウェア製品のパフォーマンス、機能および互換性が non-mission-critical 設定で通常評価される、ベータと同様のステージ。

[ベータ](#)も参照

圧縮

使用するディスク領域の縮小、実行する I/O の減少、および使用するキャッシュ用のメモリーの軽減による幅広いメリットを伴う機能。

[InnoDB](#) では、テーブルレベルとページレベルの両方の圧縮がサポートされます。[InnoDB](#) のページ圧縮は、透過的ページ圧縮とも呼ばれます。[InnoDB](#) 圧縮の詳細は、[セクション15.9「InnoDB のテーブルおよびページの圧縮」](#)を参照してください。

別の種類の圧縮は、MySQL Enterprise Backup 製品の圧縮バックアップ機能です。
[バッファプール](#), [圧縮バックアップ](#), [圧縮行フォーマット](#), [DML](#), [透過的ページ圧縮](#)も参照

圧縮テーブル

データが圧縮形式で格納されたテーブル。[InnoDB](#) の場合、これは `ROW_FORMAT=COMPRESSED` で作成されたテーブルです。詳細は、[セクション15.9「InnoDB のテーブルおよびページの圧縮」](#)を参照してください。

[圧縮行フォーマット](#), [圧縮](#)も参照

圧縮バックアップ

MySQL Enterprise Backup 製品の圧縮機能は、各テーブルスペースの圧縮コピーを作成し、`.ibd` から `.ibz` に拡張子を変更します。バックアップデータを圧縮すると、より多くのバックアップを手元に保持し、バックアップを別のサーバーに転送する時間を短縮できます。データはリストア操作中に圧縮解除されます。圧縮バックアップ操作は、すでに圧縮されているテーブルを処理するときに、ふたたび圧縮してもほとんどまたはまったく領域の節約にならないので、そのテーブルの圧縮ステップをスキップします。

MySQL Enterprise Backup 製品が生成するファイルセット。ここでは、各テーブルスペースが圧縮されます。圧縮ファイルは、.ibz ファイル拡張子を使用して名前が変更されます。

バックアッププロセスの開始時に compression を適用すると、圧縮プロセス中の記憶域のオーバーヘッドを回避し、バックアップファイルを別のサーバーに転送する際のネットワークオーバーヘッドを回避できます。バイナリログを適用するプロセスはさらに時間がかかるようになり、バックアップファイルの圧縮解除が必要になります。

[適用](#), [バイナリログ](#), [圧縮](#), [ホットバックアップ](#), [MySQL Enterprise Backup](#), [テーブルスペース](#)も参照

圧縮失敗

実際にはエラーではなくむしろ、圧縮を DML 操作と組み合わせて使用するとき起きる可能性のある負荷のかかる操作です。これは、圧縮ページへの更新が変更記録に予約されたページ上の領域からオーバーフローするとき、すべての変更がテーブルデータに適用してページが再度圧縮されたとき、再圧縮されたデータが元のページ上で適合せず、MySQL がデータを 2 つの新しいページに分割しそれぞれを個別に圧縮するように要求するときに起こります。この条件の頻度を確認するには、[INFORMATION_SCHEMA.INNO_DB_CMP](#) テーブルをクエリーして、[COMPRESS_OPS](#) カラムの値が [COMPRESS_OPS_OK](#) カラムの値を超えているかどうかを確認します。理想的には、圧縮の失敗は頻繁には発生しません。その場合は、[innodb_compression_level](#)、[innodb_compression_failure_threshold_pct](#) および [innodb_compression_pad_pct_max](#) の構成オプションを調整できます。

[圧縮](#), [DML](#), [ページ](#)も参照

圧縮行フォーマット

InnoDB テーブルのデータおよびインデックス圧縮を有効にする行フォーマット。ラージフィールドは、動的行フォーマットと同様に、残りの行データを保持するページとは別に格納されます。インデックスページとラージフィールドの両方が圧縮され、メモリーとディスクの節約をもたらします。データの構造に応じて、メモリーおよびディスク使用量の減少は、使用時にデータを圧縮解除するときのパフォーマンスオーバーヘッドを上回る場合も、上回らない場合もあります。使用法の詳細は、[セクション 15.9「InnoDB のテーブルおよびページの圧縮」](#)を参照してください。

InnoDB COMPRESSED 行フォーマットの追加情報については、[DYNAMIC 行フォーマット](#)を参照してください。

[圧縮](#), [動的行フォーマット](#), [行フォーマット](#)も参照

新しい

InnoDB バッファプールの page の特性は、最近アクセスされたことを意味し、LRU アルゴリズムによって flushed がすぐに使用されないようにバッファプールのデータ構造内で移動されます。この用語は、バッファプールに関連するテーブルの一部の INFORMATION_SCHEMA カラム名で使用されます。

[バッファプール](#), [フラッシュ](#), [INFORMATION_SCHEMA](#), [LRU](#), [ページ](#)も参照

暗黙の行ロック

一貫性を確保するために InnoDB が取得する行ロック (特にリクエストしない場合)。

[行ロック](#)も参照

I

インスタンス

単一の mysqld デーモン。テーブルのセットで 1 つ以上のデータベースを表すデータディレクトリを管理します。同じサーバーマシン上に複数のインスタンスを持ち、それぞれが専用のデータディレクトリを管理し、独自のポートまたはソケットで待機することは、開発、テスト、および一部のレプリケーションシナリオにおいて一般的です。あるインスタンスがディスクバウンドワークロードを実行している状態でも、サーバーは追加インスタンスを実行するために余分な CPU およびメモリー容量を持つことができます。

[データディレクトリ](#), [データベース](#), [ディスクバウンド](#), [mysqld](#), [レプリケーション](#), [サーバー](#), [テーブル](#)も参照

インストゥルメンテーション

ソースコードレベルでの、チューニングおよびデバッグのパフォーマンスデータを収集するための変更。MySQL では、インストゥルメンテーションによって収集されたデータは、[INFORMATION_SCHEMA](#) および [PERFORMANCE_SCHEMA](#) データベースを使用して SQL インタフェースを介して公開されます。

[INFORMATION_SCHEMA](#), [パフォーマンススキーマ](#)も参照

インテンションロック

テーブルに適用される lock の一種で、トランザクションがテーブル内の行に対して取得するロックの種類を示すために使用されます。トランザクションごとに異なる種類のインテンションロックを取得できますが、最初のトランザクションがあるテーブルでインテンション排他 (IX) ロックを獲得すると、ほかのトランザクションはそのテーブルで S または X ロックを獲得できません。反対に、最初のトランザクションがあるテーブルでインテンション共有 (IS) ロックを獲得すると、ほかのト

ランザクションはそのテーブルで X ロックを獲得できません。2 フェーズプロセスは、ロックおよび互換性のある対応操作をブロックせずに、ロックリクエストを順番に解決できます。このロックメカニズムの詳細は、[セクション15.7.1「InnoDB ロック」](#)を参照してください。

[ロック](#), [lock mode](#), [ロック](#), [トランザクション](#)も参照

インテンションロックの挿入

行の挿入前に `INSERT` 操作によって設定されるギャップロックのタイプ。このタイプの lock は、同じインデックスギャップに挿入される複数のトランザクションがギャップ内の同じ位置に挿入されない場合に相互に待機する必要がないような方法で、挿入の意図を示します。詳細は、[セクション15.7.1「InnoDB ロック」](#)を参照してください。

[ギャップロック](#), [ロック](#), [ネクストキーロック](#)も参照

インテンション共有ロック

[インテンションロック](#)も参照

インテンション排他ロック

[インテンションロック](#)も参照

インデックス

テーブルの行を高速ルックアップする機能を提供するデータ構造。通常はこのために、特定の列または列セットのすべての値を表すツリー構造 (B ツリー) を形成します。

InnoDB テーブルには、常に主キーを表すクラスタインデックスがあります。1 つ以上の列に 1 つ以上のセカンダリインデックスを定義することもできます。セカンダリインデックスは、その構造に応じて、部分、列、または複合インデックスとして分類できます。

インデックスは、クエリーパフォーマンスの重要な側面です。データベースアーキテクトは、アプリケーションが必要とするデータを高速なルックアップできるように、テーブル、クエリー、およびインデックスを設計します。理想的なデータベース設計では、有用なときはカバリングインデックスを使用します。クエリー結果は、実際のテーブルデータを読み取らずに完全にインデックスから計算されます。親テーブルと子テーブルの両方に値が存在するかどうかを効率的に確認するために、各外部キー制約にもインデックスが必要です。

B ツリーインデックスは最も一般的ですが、`MEMORY` ストレージエンジンや InnoDB 適応型ハッシュインデックスのように、ハッシュインデックスには異なる種類のデータ構造が使用されます。R-tree インデックスは、多次元情報の空間インデックス付けに使用されます。

[適応型ハッシュインデックス](#), [B ツリー](#), [子テーブル](#), [クラスタ化されたインデックス](#), [列インデックス](#), [複合インデックス](#), [カバリングインデックス](#), [外部キー](#), [ハッシュインデックス](#), [親テーブル](#), [部分インデックス](#), [主キー](#), [クエリー](#), [R-tree](#), [行](#), [セカンダリインデックス](#), [テーブル](#)も参照

インデックスキャッシュ

InnoDB 全文検索のトークンデータを保持するメモリー領域。FULLTEXT インデックスの一部である列でデータが挿入または更新されるときにディスク I/O を最小限に抑えるために、データをバッファリングします。インデックスキャッシュがいっぱいになると、トークンデータがディスクに書き込まれます。各 InnoDB FULLTEXT インデックスには独自のインデックスキャッシュがあり、そのサイズは構成オプション `innodb_ft_cache_size` によって制御されます。

[全文検索](#), [FULLTEXT インデックス](#)も参照

インデックスヒント

オプティマイザが推奨するインデックスをオーバーライドするための拡張 SQL 構文。たとえば、`FORCE INDEX`、`USE INDEX`、および `IGNORE INDEX` 句。通常は、インデックス付き列に値が不均等に分散されていて、カーディナリティーを正確に見積もれないときに使用されます。

[カーディナリティー](#), [インデックス](#)も参照

インデックスプリフィクス

複数の列に適用されるインデックス (複合インデックスと呼ばれます) で、インデックスの先頭または末尾列。複合インデックスの最初の 1、2、3 などの列を参照するクエリーはインデックスを使用できます (クエリーがインデックス内のすべての列を参照するわけではない場合でも)。

[複合インデックス](#), [インデックス](#)も参照

インデックス条件プッシュダウン

インデックス条件プッシュダウン (ICP) は、index のフィールドを使用して条件の一部を評価できる場合に、`WHERE` 条件の一部をストレージエンジンにプッシュダウンする最適化です。ICP は、ストレージエンジンが実テーブルにアクセスする必要がある回数と、MySQL サーバーがストレージエンジンにアクセスする必要がある回数を減らすことができます。詳細は、[セクション8.2.1.6「インデックスコンディショニングプッシュダウンの最適化」](#)を参照してください。

[インデックス](#), [ストレージエンジン](#)も参照

インデックス統計

[統計](#)も参照

インメモリーデータベース

ディスクブロックとメモリー領域間のディスク I/O および変換によるオーバーヘッドを回避するために、メモリー内にデータを保持するタイプのデータベースシステム。一部のインメモリーデータベースは耐久性 (ACID 設計理念の「D」) を犠牲にしており、ハードウェア、電源およびその他のタイプの障害に対して脆弱であるため、読取り専用操作に適しています。ほかのインメモリーデータベースでは、変更ログをディスクに記録したり不揮発性メモリーを使用したりなどの持続性メカニズムを使用します。

同じ種類のメモリー集中型処理に対処する MySQL 機能には、[InnoDB バッファプール](#)、[適応型ハッシュインデックス](#)および読取り専用トランザクションの最適化、[MEMORY ストレージエンジン](#)、[MyISAM キーキャッシュ](#)および MySQL クエリーキャッシュが含まれます。

[ACID](#), [適応型ハッシュインデックス](#), [バッファプール](#), [ディスクベース](#), [読取り専用トランザクション](#)も参照

一意のインデックス

一意制約が適用されるカラムまたはカラムセットのインデックス。インデックスが重複値を含まないことがわかっているため、特定の種類のルックアップとカウント操作が通常の種類のインデックスよりも効率的になります。このタイプのインデックスに対するほとんどのルックアップは、単純に特定の値が存在するかどうかを判断することです。インデックス内の値の数は、テーブル内の行数と同じであるか、関連付けられたカラムが NULL 以外の値である行の数以上です。

バッファリングの変更の最適化は、一意インデックスには適用されません。回避策として、[InnoDB テーブルへのバルクデータロードの実行中に `unique_checks=0` を一時的に設定できます。](#)

[カーディナリティー](#), [変更バッファリング](#), [一意制約](#), [一意のキー](#)も参照

一意のキー

一意のインデックスを構成するカラムセット (1 つまたは複数)。正確に 1 行に一致する [WHERE](#) 条件を定義でき、クエリーが関連付けられた一意のインデックスを使用できるときは、ルックアップおよびエラー処理を非常に効率的に実行できます。

[カーディナリティー](#), [一意制約](#), [一意のインデックス](#)も参照

一意制約

重複値をカラムに含むことができないと表明するタイプの制約。リレーショナル代数の観点では、1 対 1 関係を指定するために使用されます。ある値を挿入できるかどうかをチェックするときの効率性のために (つまり、その値がカラムにまだ存在していない)、一意制約が基礎となる一意のインデックスでサポートされます。

[制約](#), [リレーショナル](#), [一意のインデックス](#)も参照

一時テーブル

データを真に永続的にする必要がない table。たとえば、一時テーブルを複雑な計算または変換の中間結果のストレージ領域として使用できます。この中間データはクラッシュ後にリカバリする必要がありません。データベース製品は、データをディスクに書き込むことや再起動をまたがってデータを保護する手段について緻密さを軽減することで、一時テーブルに対する操作パフォーマンスを手軽に改善できます。

トランザクション終了時やセッション終了時などに、データ自体が所定時に自動的に削除されることもあります。一部のデータベース製品では、テーブル自体も自動的に削除されます。

[テーブル](#)も参照

一時テーブルスペース

[InnoDB](#) では、2 つのタイプの一時テーブルスペースを使用します。セッション一時テーブルスペースには、オペティマイザによって作成されたユーザー作成一時テーブルおよび内部一時テーブルが格納されます。グローバル一時テーブルスペースには、ユーザーが作成した一時テーブルに対して行われた変更のためにロールバックセグメントが格納されます。

[グローバル一時テーブルスペース](#), [セッション一時テーブルスペース](#), [一時テーブル](#)も参照

一般クエリーログ

MySQL Server によって処理された SQL ステートメントの診断およびトラブルシューティングに使用されるタイプのログ。ファイルまたはデータベーステーブルにも格納できます。この機能を使用するには、[general_log](#) 構成オプションを通じて有効にする必要があります。[sql_log_off](#) 構成オプションを通じて特定の接続に対してこれを無効にできます。

スロークエリーログよりも広い範囲のクエリーを記録します。レプリケーションに使用されるバイナリログとは異なり、一般クエリーログは [SELECT](#) ステートメントを含み、厳密な順序を維持しません。詳細は、[セクション 5.4.3 「一般クエリーログ」](#) を参照してください。

[バイナリログ](#), [ログ](#), [スロークエリーログ](#)も参照

一般テーブルスペース

`CREATE TABLESPACE` 構文を使用して作成された共有 `InnoDB` テーブルスペース。一般テーブルスペースは、MySQL データディレクトリの外部に作成でき、複数のテーブルを保持でき、すべての行形式のテーブルをサポートします。一般的なテーブルスペースは、MySQL 5.7.6 で導入されました。

テーブルは、`CREATE TABLE tbl_name ... TABLESPACE [=] tablespace_name` または `ALTER TABLE tbl_name TABLESPACE [=] tablespace_name` 構文を使用して一般テーブルスペースに追加されます。

システムテーブルスペースおよび `file-per-table` テーブルスペースと対比してください。

詳細は、[セクション15.6.3.3「一般テーブルスペース」](#)を参照してください。
[file-per-table](#), [システムテーブルスペース](#), [テーブル](#), [テーブルスペース](#)も参照

一般ログ

[一般クエリーログ](#)も参照

一貫性読み取り

snapshot 情報を使用して、同時に実行されている他のトランザクションによって実行された変更に関係なく、ある時点に基づいてクエリー結果を表示する読み取り操作。照会されるデータが別のトランザクションによって変更されている場合、元のデータは Undo ログの内容に基づいて再構築されます。この方法は、ほかのトランザクションが終了するのを待機するようにトランザクションを強制することによって、並列性を減少させる可能性のあるいくつかのロック問題を回避します。

REPEATABLE READ 分離レベルでは、スナップショットは最初の読み取り操作が実行された時間に基づきます。READ COMMITTED 分離レベルでは、スナップショットは各読み取り一貫性操作の時間にリセットされます。

一貫性読み取りは、`InnoDB` が READ COMMITTED および REPEATABLE READ 分離レベルで `SELECT` ステートメントを処理する際のデフォルトモードです。一貫性読み取りはアクセスするテーブルでロックを設定しないので、テーブルで一貫性読み取りが実行されている間、ほかのセッションはそれらのテーブルを自由に変更できます。

適用可能な分離レベルの技術的な詳細は、[セクション15.7.2.3「一貫性非ロック読み取り」](#)を参照してください。
[並列性](#), [分離レベル](#), [ロック](#), [READ COMMITTED](#), [REPEATABLE READ](#), [スナップショット](#), [トランザクション](#), [Undo ログ](#)も参照

ウ

ウォームアップ

起動後の一定期間、標準的なワークロードでシステムを実行すること。通常の状態時のようにバッファプールとほかのメモリ領域がいっぱいになります。このプロセスは、MySQL Server が再起動したり新しいワークロードを課せられたりしたときに、一定時間をかけて自然に発生します。

通常は、複数の実行間で一貫した結果を保証するために、一定期間ワークロードを実行してバッファプールをウォームアップしてから、パフォーマンステストを実行してください。そのようにしない場合、パフォーマンスが最初の実行中に不自然に低くなる場合があります。

MySQL 5.6 では、`innodb_buffer_pool_dump_at_shutdown` および `innodb_buffer_pool_load_at_startup` 構成オプションを有効にしてウォームアッププロセスを高速化し、再起動後にバッファプールの内容をメモリーに戻すことができます。これらのオプションは、MySQL 5.7 ではデフォルトで有効になっています。[セクション15.8.3.6「バッファプールの状態の保存と復元」](#)を参照してください。

[バッファプール](#), [ワークロード](#)も参照

ウォームバックアップ

データベースの実行中に行われるが、バックアッププロセス中に一部のデータベース操作を制限するバックアップ。たとえば、テーブルが読み取り専用になることがあります。ビジー状態のアプリケーションおよび web サイトの場合は、ホットバックアップを使用することをお勧めします。

[バックアップ](#), [コールドバックアップ](#), [ホットバックアップ](#)も参照

埋込み

埋込み MySQL サーバライブラリ (`libmysqld`) を使用すると、client アプリケーション内でフル機能の MySQL サーバーを実行できます。この主なメリットは組み込みアプリケーションの速度の向上と管理の単純化です。

[クライアント](#), [libmysqld](#)も参照

エ

エクステント

テーブルスペース内のページのグループ。デフォルトのページサイズが 16KB の場合、エクステントには 64 ページが含まれます。MySQL 5.6 では、InnoDB インスタンスのページサイズには 4K バイト、8K バイト、16K バイトがあり、`innodb_page_size` 構成オプションで制御されます。4K バイト、8K バイト、および 16K バイトのページサイズでは、エクステントサイズは常に 1M バイト (つまり 1048576 バイト) です。

32K バイトおよび 64K バイトの InnoDB ページサイズのサポートが MySQL 5.7.6 で追加されました。32K バイトページサイズの場合、エクステントサイズは 2M バイトです。64K バイトページサイズの場合、エクステントサイズは 4M バイトです。

セグメント、先読みリクエスト、および二重書き込みバッファなどの InnoDB 機能は、一度に 1 つのエクステントでデータの読み取り、書き込み、割り当て、または解放を行う I/O 操作を使用します。

[二重書き込みバッファ](#)、[ページ](#)、[page size](#)、[先読み](#)、[セグメント](#)、[テーブルスペース](#)も参照

エビクション

キャッシュまたはその他の一時記憶域 (InnoDB バッファプールなど) からアイテムを削除するプロセス。常にではないけれども多くの場合、LRU アルゴリズムを使用して、削除する項目を決定します。ダーティページが削除されると、その内容は flushed からディスクになり、使用済のネイバーページもフラッシュされる可能性があります。

[バッファプール](#)、[ダーティページ](#)、[フラッシュ](#)、[LRU](#)、[隣接ページ](#)も参照

エラーログ

MySQL 起動に関する情報と、重要な実行時エラーおよびクラッシュ情報を示すタイプのログ。詳細は、[セクション 5.4.2 「エラーログ」](#)を参照してください。

[クラッシュ](#)、[ログ](#)も参照

永続的統計

InnoDB テーブルの index 統計をディスクに格納し、クエリーのプランスタビリティを向上させる機能。詳細は、[セクション 15.8.10.1 「永続的オプティマイザ統計のパラメータの構成」](#)を参照してください。

[インデックス](#)、[オプティマイザ](#)、[計画安定性](#)、[クエリー](#)、[テーブル](#)も参照

オ

オフページカラム

長すぎるため B ツリーページに収まらない可変長データを含むカラム (BLOB や VARCHAR など)。データはオーバーフローページに格納されます。DYNAMIC の行形式は、古い COMPACT の行形式よりもこのような記憶域の方が効率的です。

[B ツリー](#)、[コンパクト行フォーマット](#)、[動的行フォーマット](#)、[オーバーフローページ](#)も参照

オプション

MySQL の構成パラメータ。オプションファイルに格納されるか、コマンド行で渡されます。

InnoDB テーブルに適用される options の場合、各オプション名は接頭辞 `innodb_` で始まります。

[InnoDB](#)、[オプション](#)、[オプションファイル](#)も参照

オプションファイル

MySQL インスタンスの構成オプションを保持するファイル。従来、Linux および Unix では、このファイルの名前は `my.cnf` で、Windows では `my.ini` です。

[構成ファイル](#)、[my.cnf](#)、[my.ini](#)、[オプション](#)も参照

オプティマイザ

該当するテーブルの特性とデータ分布に基づいて、クエリーに使用する最適なインデックスおよび結合順序を決定する MySQL コンポーネント。

[インデックス](#)、[結合](#)、[クエリー](#)、[テーブル](#)も参照

オプティミスティック

リレーショナルデータベースシステムの低レベル実装決定を導く概念。リレーショナルデータベースでパフォーマンスと並列性がなければならないことは、操作を迅速に開始またはディスクパッチする必要があることを意味します。一貫性と参照整合性がなければならないことは、どの操作も失敗する可能性があることを意味します。つまり、トランザクションがロールバックされ、DML 操作が制約を違反し、ロックのリクエストがデッドロックになり、ネットワークエラーがタイムアウトになる可能

性があります。オプティミスティック戦略は、ほとんどのリクエストまたは試行が成功したことを前提としているため、失敗ケースを準備するために比較的少ない作業が行われます。この想定が true のときは、データベースは不要な作業をほとんど行いません。リクエストが失敗したときは、変更をクリーンアップして元に戻すために追加作業が必要になります。

InnoDB では、ロック、commits などの操作にオプティミスティック戦略を使用します。たとえば、トランザクションによって変更されたデータは、コミットが発生する前にデータファイルに書き込むことができるため、コミット自体は非常に速いですが、トランザクションがロールバックされた場合に変更を元に戻すためにより多くの作業が必要です。

オプティミスティック戦略の反対がペシミスティック戦略で、信頼性が低く頻繁に失敗する操作に対応するようにシステムが最適化されます。この概念はデータベースシステムではまれです。信頼性の高いハードウェア、ネットワーク、およびアルゴリズムを選択することに非常に多くの努力が注ぎ込まれるためです。

[コミット](#)、[並列性](#)、[DML](#)、[ロック](#)、[ペシミスティック](#)、[参照整合性](#)も参照

オンライン

データベースのダウンタイム、ブロック、または制限された操作のないタイプの操作。通常は DDL に適用されます。高速インデックス作成など、制限された操作の期間を短くする操作は、MySQL 5.6 で幅広いオンライン DDL 操作セットに進化しました。

バックアップのコンテキストでは、ホットバックアップはオンライン操作、ウォームバックアップは部分的にオンライン操作です。

[DDL](#)、[高速インデックス作成](#)、[ホットバックアップ](#)、[オンライン DDL](#)、[ウォームバックアップ](#)も参照

オンライン DDL

DDL(主に [ALTER TABLE](#)) 操作中の InnoDB テーブルのパフォーマンス、同時実行性および可用性を向上させる機能。詳細は、[セクション 15.12「InnoDB とオンライン DDL」](#)を参照してください。

詳細は、操作の種類に応じて異なります。場合によっては、[ALTER TABLE](#) の進行中にテーブルを同時に変更できます。操作は、テーブルコピーなしで実行することも、特別に最適化されたタイプのテーブルコピーを使用して実行することもできます。インプレース操作の DML ログ領域使用量は、`innodb_online_alter_log_max_size` 構成オプションによって制御されます。

この機能は、MySQL 5.5 の高速インデックス作成機能の拡張です。

[DDL](#)、[高速インデックス作成](#)、[オンライン](#)も参照

オーバーフローページ

可変長カラム ([BLOB](#) や [VARCHAR](#) など) が長すぎて B ツリーページに収まらない場合に、それらを保持するために別個に割り当てられたディスクページ。関連付けられたカラムはオフページカラムと呼ばれます。

[B ツリー](#)、[オフページカラム](#)、[ページ](#)も参照

カ

カウンタ

特定の種類の InnoDB 操作によって増分される値。サーバーがどれだけビジーかを測定する、パフォーマンス問題の原因をトラブルシューティングする、(たとえば、クエリーが使用する構成設定またはインデックスへの) 変更が目的の低レベル効果を持っているかどうかをテストする場合に役立ちます。パフォーマンススキーマテーブルおよび INFORMATION_SCHEMA テーブル (特に [INFORMATION_SCHEMA.INNODB_METRICS](#)) では、様々な種類のカウンタを使用できます。

[INFORMATION_SCHEMA](#)、[メトリックカウンタ](#)、[パフォーマンススキーマ](#)も参照

カバリングインデックス

クエリーによって取得されたすべてのカラムを含むインデックス。完全なテーブル行を見つけるためのポインタとしてインデックス値を使用する代わりに、クエリーはインデックス構造から値を返し、ディスク I/O を節約します。InnoDB セカンダリインデックスには主キーカラムも含まれているため、InnoDB では、MyISAM で可能なより多くのインデックスにこの最適化手法を適用できます。InnoDB では、トランザクションが終了するまで、トランザクションによって変更されたテーブルに対するクエリーにこの方法を適用できません。

正しいクエリーの場合、どのカラムインデックスまたは複合インデックスでも、カバリングインデックスとして機能できます。可能な場合は必ず、この最適化方法を活用するようにインデックスおよびクエリーを設計してください。

[カラムインデックス](#)、[複合インデックス](#)、[インデックス](#)、[主キー](#)、[セカンダリインデックス](#)も参照

カラム

ストレージおよびセマンティクスがデータ型によって定義される、行内のデータ項目。それぞれのテーブルおよびインデックスは主に、そこに含まれるカラムセットによって定義されます。

それぞれのカラムにはカーディナリティー値があります。カラムは、そのテーブルの主キーであったり、主キーの一部であったりします。カラムは、一意制約または NOT NULL 制約、あるいはその両方により制限される場合があります。別のカラム内の値には、異なるテーブルにまたがっている場合でも、外部キー関係によってリンクできます。

MySQL 内部操作についてディスカッションするときに、シノニムとしてフィールドが使用されることがあります。
[カーディナリティー](#)、[外部キー](#)、[インデックス](#)、[NOT NULL 制約](#)、[主キー](#)、[行](#)、[テーブル](#)、[一意制約](#)も参照

カラムインデックス

単一カラムのインデックス。
[複合インデックス](#)、[インデックス](#)も参照

カラムプリフィクス

index が長さ指定 (`CREATE INDEX idx ON t1 (c1(N))` など) で作成されると、カラム値の最初の N 文字のみがインデックスに格納されます。インデックスプリフィクスを小さいまま維持すると、インデックスがコンパクトになり、メモリーとディスク I/O の節約によりパフォーマンスが向上します。(ただし、インデックスプリフィクスを小さくし過ぎると、値の異なる行がクエリー最適化には重複していると見えるようになり、クエリー最適化を妨げる可能性があります。)

バイナリ値や長いテキスト文字列を含むカラムの場合、ソートは主な考慮事項ではなく、値全体をインデックスに格納すると領域が浪費されるので、インデックスは自動的に、最初の N (通常 768) 文字を使用してルックアップおよびソートを行います。
[インデックス](#)も参照

カーソル

SQL ステートメントの結果セットを表す内部 MySQL データ構造。多くの場合、プリヘアドステートメントおよび動的 SQL で使用されます。これは、ほかの高級言語でのイテレータのように機能し、リクエストに応じて結果セットからそれぞれの値を生成します。

通常、SQL はカーソルの処理を処理しますが、パフォーマンスが重視されるコードを処理する際には、内部の作業に深刻になる可能性があります。
[動的 SQL](#)、[プリヘアドステートメント](#)、[クエリー](#)も参照

カーディナリティー

テーブルカラム内の異なる値の数。インデックスが関連付けられたカラムをクエリーが参照するときに、各カラムのカーディナリティーはどのアクセス方法がもっとも効率的かに影響します。たとえば、一意制約が適用されるカラムの場合、異なる値の数はテーブル内の行数に等しくなります。テーブル内の行数が 100 万なのに、特定のカラムの異なる値が 10 個だけの場合は、各値は (平均) 100,000 回発生します。したがって、`SELECT c1 FROM t1 WHERE c1 = 50;` などのクエリーが 1 行を返したり非常に多数の行を返したりする可能性があり、データベースサーバーはこのクエリーを `c1` のカーディナリティーに応じてさまざまに処理できます。

カラム内の値が非常に不規則に分布している場合は、カーディナリティーが最善のクエリー計画を決定するうえで適切な方法でないことがあります。たとえば、`SELECT c1 FROM t1 WHERE c1 = x;` は `x=50` のときに 1 行を返し、`x=30` のときに 100 万行を返します。このような場合、どのルックアップ方法が特定のクエリーにより効率的かについてアドバイスを伝えるために、インデックスヒントを使用する必要があります。

カーディナリティーは、複合インデックスの場合と同様に、複数のカラムに存在する異なる値の数にも適用できます。
[カラム](#)、[複合インデックス](#)、[インデックス](#)、[インデックスヒント](#)、[永続的統計](#)、[ランダムダイブ](#)、[選択性](#)、[一意制約](#)も参照

仮想インデックス

仮想インデックスは、1 つ以上の仮想生成カラム上、または仮想生成カラムと通常のカラムまたは格納された生成カラムの組合せ上のセカンダリインデックスです。詳細は、[セクション 13.1.20.9 「セカンダリインデックスと生成されたカラム」](#) を参照してください。
[セカンダリインデックス](#)、[ストアド生成カラム](#)、[仮想生成カラム](#)も参照

仮想カラム

[仮想生成カラム](#)も参照

仮想生成カラム

カラム定義に含まれる式から値が計算されるカラム。カラム値は格納されませんが、`BEFORE` トリガーの直後に行が読み取られたときに評価されます。仮想生成カラムは記憶域を取りません。InnoDB では、仮想生成カラムのセカンダリインデックスがサポートされます。

ストアド生成カラムと対比してください。

[ベースカラム](#), [生成されるカラム](#), [ストアド生成カラム](#)も参照

可変長型

可変長のデータ型。 [VARCHAR](#)、[VARBINARY](#)、[BLOB](#) および [TEXT](#) 型は可変長型です。

InnoDB では、768 バイト以上の固定長フィールドは可変長フィールドとして扱われ、off-page に格納できます。たとえば、[utf8mb4](#) と同様に、文字セットの最大バイト長が 3 より大きい場合、[CHAR\(255\)](#) カラムは 768 バイトを超えることがあります。

[オフページカラム](#), [オーバーフローページ](#)も参照

可用性

ホストでの障害 (MySQL、オペレーティングシステム、ハードウェア、メンテナンス作業での障害を含む) に対処し、必要に応じてリカバリする能力 (ない場合は、ダウンタイムが発生する可能性があります)。多くの場合、大規模開発の重要な側面として、スケーラビリティと組み合わせられます。

[スケーラビリティ](#)も参照

外部キー

個別の InnoDB テーブルの行間のポインタ関係のタイプ。外部キー関係は、親テーブルおよび子テーブルの 1 つのカラムに定義されます。

外部キーは、関連情報の高速ルックアップを有効にすることに加えて、データが挿入、更新、および削除されるときにこれらのポインタが無効になるのを防ぐことによって参照整合性を適用するのに役立ちます。この適用メカニズムは一種の制約です。別のテーブルをポイントする行は、関連付けられた外部キー値がその別のテーブルに存在しない場合、挿入できません。行が削除されているか、その外部キー値が変化していて、別のテーブル内の行がその外部キー値をポイントしている場合は、削除を防止したり、ほかのテーブル内の対応するカラム値が NULL になるようにしたり、ほかのテーブル内の対応する行を自動的に削除したりするように外部キーを設定できます。

正規化データベースを設計する際の段階の 1 つは、複製されるデータを特定し、そのデータを新しいテーブルに分離し、結合操作を使用して複数のテーブルを単一テーブルのように照会できるように外部キー関係を設定することです。

[子テーブル](#), [FOREIGN KEY 制約](#), [結合](#), [正規化](#), [NULL](#), [親テーブル](#), [参照整合性](#), [リレーショナル](#)も参照

完全バックアップ

各 MySQL データベース内のすべてのテーブルと MySQL インスタンス内のすべてのデータベースを含むバックアップ。部分バックアップと対比してください。

[バックアップ](#), [データベース](#), [インスタンス](#), [部分バックアップ](#), [テーブル](#)も参照

書き込み結合

ダーティページが InnoDB バッファプールの flushed である場合に書き込み操作を減らす最適化手法。ページ内の行が複数回更新されたり、同じページ上の複数の行が更新されたりする場合、変更ごとに一度の書き込みではなく、単一書き込み操作でこれらのすべての変更がデータファイルに格納されます。

[バッファプール](#), [ダーティページ](#), [フラッシュ](#)も参照

関連性

全文検索機能で、検索文字列と FULLTEXT インデックス内のデータとの間の類似性を示す数字。たとえば、単一の単語を検索する場合、その単語は通常、1 回のみ表示される行よりもテキスト内で複数回出現する行に関連します。

[全文検索](#), [FULLTEXT インデックス](#)も参照

キ

キャッシュ

頻繁または高速に取り出せるようにデータのコピーを格納するメモリ領域を表す一般的な用語。InnoDB では、キャッシュ構造の主な種類はバッファプールです。

[バッファ](#), [バッファプール](#)も参照

ギャップ

新しい値を挿入できる InnoDB index データ構造内の場所。SELECT ... FOR UPDATE などのステートメントを使用して行のセットをロックすると、InnoDB ではギャップおよびインデックス内の実際の値に適用されるロックを作成できます。たとえば、更新に 10 より大きな値をすべて選択した場合、ギャップロックはほかのトランザクションが 10 を超える新しい値を挿入するのを妨げます。最小上限レコードと最大下限レコードは、現在のすべてのインデックス値よりも大きな値または小さな値をすべて含むギャップを表します。

[並列性](#), [ギャップロック](#), [インデックス](#), [最大下限レコード](#), [分離レベル](#), [最小上限レコード](#)も参照

ギャップロック

インデックスレコード間のギャップのロック、または先頭インデックスレコードの前や末尾インデックスレコードのあとのギャップのロック。たとえば、`SELECT c1 FROM t WHERE c1 BETWEEN 10 and 20 FOR UPDATE;`では、範囲内の既存のすべての値間のギャップがロックされているため、カラムにそのような値がすでに存在するかどうかにかかわらず、他のトランザクションが 15 の値を `t.c1` カラムに挿入することを防ぎます。レコードロックおよびネクストキーロックと対比してください。

ギャップロックは、パフォーマンスと並列性とのトレードオフの一環であり、一部のトランザクション分離レベルで使用され、ほかでは使用されません。

[ギャップ](#)、[最大下限レコード](#)、[ロック](#)、[ネクストキーロック](#)、[レコードロック](#)、[最小上限レコード](#)も参照

共有テーブルスペース

システムテーブルスペースまたは一般テーブルスペースを参照する別の方法。一般的なテーブルスペースは、MySQL 5.7 で導入されました。複数のテーブルを共有テーブルスペースに配置できます。単一のテーブルのみが `file-per-table` テーブルスペースに存在できます。

[一般テーブルスペース](#)、[システムテーブルスペース](#)も参照

共有ロック

ロックの一種。ほかのトランザクションがロックされたオブジェクトを読み取ることを許可し、それに対するほかの共有ロックを獲得することも許可するけれども、それに書き込むことは許可しません。排他ロックの反対。

[排他ロック](#)、[ロック](#)、[トランザクション](#)も参照

切り捨て

テーブルおよび関連するインデックスを残しながら、テーブルの内容全体を削除する DDL 操作。ドロップと対比してください。概念的には、`WHERE` 句のない `DELETE` ステートメントと同じ結果になりますが、バックグラウンドでの動作は異なります: `InnoDB` は新しい空のテーブルを作成し、古いテーブルを削除してから、古いテーブルのかわりに新しいテーブルの名前を変更します。これは DDL 操作なので、ロールバックできません。

切り捨てられるテーブルに、別のテーブルを参照する外部キーが含まれている場合、切捨て操作では、`ON DELETE CASCADE` 句で必要に応じて参照テーブルの対応する行を削除できるように、処理速度の遅い行を一度に削除します。

(MySQL 5.5 以降では、このより遅い形式の切り捨てを許可しない代わりに、外部キーが含まれている場合はエラーを返します。この場合は、代わりに `DELETE` ステートメントを使用してください。

[DDL](#)、[ドロップ](#)、[外部キー](#)、[ロールバック](#)も参照

疑似レコード

現在存在しないキー値または範囲をロックするために使用される、インデックス内の人為レコード。

[最大下限レコード](#)、[ロック](#)、[最小上限レコード](#)も参照

起動

MySQL Server を起動させるプロセス。通常、[セクション4.3「サーバーおよびサーバーの起動プログラム」](#)に示すプログラムのいずれかによって行われます。シャットダウンの反対。

[シャットダウン](#)も参照

逆引用符

MySQL SQL ステートメント内の識別子は、特殊文字または予約語を含んでいる場合、逆引用符文字 (```) で囲む必要があります。たとえば、`FOO#BAR` というテーブルまたは `SELECT` というカラムを参照するには、``FOO#BAR`` および ``SELECT`` として識別子を指定します。逆引用符は、安全性のレベルをさらに高めるので、識別子名が前もってわかっていない可能性のあるプログラム生成の SQL ステートメントで広く使用されています。

ほかのデータベースシステムの多くでは、このような特別な名前には二重引用符 (`"`) を使用します。MySQL では移植性のために、`ANSI_QUOTES` モードを有効にして、逆引用符の代わりに二重引用符を使用して識別子名を修飾できます。

[SQL](#)も参照

ク

クエリー

SQL で、1 つ以上のテーブルから情報を読み取る操作。データの編成とクエリーのパラメータに応じて、ルックアップはインデックスを参照することで最適化される可能性があります。複数のテーブルが使用される場合、そのクエリーは結合と呼ばれます。

歴史的な理由から、ステートメントの内部処理のディスカッションでは、DDL ステートメントや DML ステートメントなどの他のタイプの MySQL ステートメントを含む、より広範な意味で「query」が使用される場合があります。

[DDL](#), [DML](#), [インデックス](#), [結合](#), [SQL](#), [テーブル](#)も参照

クエリーログ

[一般クエリーログ](#)も参照

クエリー実行計画

使用するインデックスやテーブルを結合する順序など、クエリーをもっとも効率的に実行する方法についてオプティマイザによって行われる決定のセット。計画安定性には、特定のクエリーについて一貫して行われている同じ選択が含まれます。
[インデックス](#), [結合](#), [計画安定性](#), [クエリー](#)も参照

クライアント

データベースサーバーの外部で実行され、Connector を介してリクエストを送信してデータベースと通信するプログラム、またはクライアントライブラリを介して使用可能になった API。データベースサーバーと同じ物理マシン上、またはネットワーク経由で接続されたリモートマシン上で実行できます。これは、特別な目的のデータベースアプリケーション、または `mysql` コマンドラインプロセッサなどの汎用プログラムです。
[API](#), [クライアントライブラリ](#), [コネクタ](#), [mysql](#), [サーバー](#)も参照

クライアントライブラリ

データベースを操作するための関数のコレクションを含むファイル。これらのライブラリを使用してプログラムをコンパイルするか、アプリケーションと同じシステムにインストールすることで、MySQL サーバーがインストールされていないマシンでデータベースアプリケーション (client と呼ばれる) を実行できます。アプリケーションはネットワークを介してデータベースにアクセスします。MySQL では、MySQL サーバー自体から `libmysqlclient` ライブラリを使用できます。
[クライアント](#), [libmysqlclient](#)も参照

クライアント側のプリベアドステートメント

キャッシュおよび再利用がローカルで管理され、サーバー側のプリベアドステートメントの機能をエミュレートするプリベアドステートメントのタイプ。従来は、一部の Connector/J、Connector/ODBC および Connector/PHP 開発者がサーバー側ストアードプロシージャの問題を回避するために使用していました。最新の MySQL サーバーバージョンでは、パフォーマンス、スケーラビリティ、およびメモリー効率を向上させるために、サーバー側のプリベアドステートメントをお勧めします。

[Connector/J](#), [Connector/ODBC](#), [Connector/PHP](#), [プリベアドステートメント](#)も参照

クラスタ化されたインデックス

主キーインデックスの `InnoDB` 用語。 `InnoDB` テーブルの記憶域は、主キーカラムの値に基づいて編成され、主キーカラムに関連するクエリーおよびソートを高速化します。パフォーマンスが最適になるように、パフォーマンスがもっとも重要なクエリーに基づいて、主キーカラムを慎重に選択してください。クラスタ化されたインデックスのカラムを変更することは負荷のかかる操作なので、まれにしか、またはまったく更新されない主カラムを選択してください。

Oracle Database 製品では、この種のテーブルはインデックス編成テーブルと呼ばれています。

[インデックス](#), [主キー](#), [セカンダリインデックス](#)も参照

クラッシュ

MySQL では、通常、「crash」という用語は、サーバーが通常のクリーンアップを実行できない予期しない shutdown 操作を指します。たとえば、クラッシュは、データベースサーバーマシンまたはストレージデバイスでのハードウェア障害、電源障害、MySQL Server の停止を招く潜在的なデータ不一致、DBA で開始された高速シャットダウン、またはその他の多くの理由によって発生することがあります。 `InnoDB` テーブルの堅牢で自動的なクラッシュリカバリは、DBA が追加作業を行うことなく、サーバーが再起動するときにデータの一貫性を保証します。

[クラッシュリカバリ](#), [高速シャットダウン](#), [InnoDB](#), [シャットダウン](#)も参照

クラッシュリカバリ

クラッシュ後に MySQL が再度起動したときに行われるクリーンアップアクティビティ。 `InnoDB` テーブルの場合、不完全なトランザクションからの変更は、Redo ログからのデータを使用して再現されます。クラッシュ前にコミットされたけれども、まだデータファイルに書き込まれていない変更は、二重書き込みバッファから再構築されます。通常どおりデータベースがシャットダウンした場合、このタイプのアクティビティは、ページ操作によってシャットダウン中に実行されます。

通常操作中、コミットされたデータは、データファイルに書き込まれる前に、一定期間変更バッファに格納できます。データファイルを最新の状態に維持すること (通常操作中にパフォーマンスオーバーヘッドをもたらす) と、データのバッファリング (シャットダウンおよびクラッシュリカバリの時間を長くする可能性がある) との間には、常にトレードオフがあります。

[変更バッファ](#), [コミット](#), [クラッシュ](#), [データファイル](#), [二重書き込みバッファ](#), [InnoDB](#), [ページ](#), [Redo ログ](#)も参照

クリーンシャットダウン

crash または高速シャットダウンではなく、エラーなしで完了し、終了前にすべての変更を InnoDB テーブルに適用する shutdown。低速シャットダウンのシノニム。

[クラッシュ](#), [高速シャットダウン](#), [シャットダウン](#), [低速シャットダウン](#)も参照

クリーンページ

InnoDB バッファプール内の page は、メモリー内で行われたすべての変更も [data files](#) に書き込まれます (flushed)。ダーティーページの反対です。

[バッファプール](#), [データファイル](#), [ダーティーページ](#), [フラッシュ](#), [ページ](#)も参照

グループコミット

コミットごとに別々にフラッシュおよび同期するのではなく、一連のコミット操作に対して一度、いくつかの低レベル I/O 操作 (ログ書き込み) を実行する InnoDB 最適化。

[バイナリログ](#), [コミット](#)も参照

グローバルトランザクション

XA 操作に含まれるタイプのトランザクション。これは、それ自体はトランザクションであるけれども、すべてがグループとして正しく完了する必要があるか、グループとしてロールバックされる必要がある、いくつかのアクションから構成されます。基本的に、これは ACID プロパティ「レベルを上げる」を拡張して、ACID プロパティも持つグローバル操作のコンポーネントとして複数の ACID トランザクションを同時に実行できるようにします。

[ACID](#), [トランザクション](#), [XA](#)も参照

グローバル一時テーブルスペース

ユーザーが作成した一時テーブルに加えた変更のためにロールバックセグメントを格納する一時テーブルスペース。

[一時テーブルスペース](#)も参照

組み込み

MySQL 内の組み込みの InnoDB ストレージエンジンは、ストレージエンジンの元の配布形式です。InnoDB Plugin と対比してください。MySQL 5.5 以降、InnoDB プラグインは組み込み InnoDB ストレージエンジン (InnoDB 1.1 と呼ばれる) として MySQL コードベースにマージされます。

この区別は主に MySQL 5.1 で重要です。この場合、機能またはバグの修正が InnoDB プラグインに適用される可能性があります。組み込み InnoDB には適用されません。その逆も同様です。

[InnoDB](#)も参照

組み込み一時テーブル

オプティマイザで使用される最適化された内部 InnoDB 一時テーブル。

[オプティマイザ](#)も参照

行

カラムセットによって定義される論理データ構造。行セットがテーブルを構成します。InnoDB データファイル内では、各 page に 1 つ以上の行を含めることができます。

InnoDB では、MySQL 構文との一貫性のために行フォーマットという用語を使用しますが、行形式は各テーブルのプロパティであり、そのテーブルのすべての行に適用されます。

[カラム](#), [データファイル](#), [ページ](#), [行フォーマット](#), [テーブル](#)も参照

行フォーマット

InnoDB table の rows のディスク記憶域形式。InnoDB が compression などの新機能を取得すると、新しい行形式が導入され、記憶域の効率とパフォーマンスが向上します。

InnoDB テーブルの行形式は、[ROW_FORMAT](#) オプションまたは [innodb_default_row_format](#) 構成オプション (MySQL 5.7.9 で導入) によって指定されます。行フォーマットには、[REDUNDANT](#), [COMPACT](#), [COMPRESSED](#) および [DYNAMIC](#) が含まれます。InnoDB テーブルの行形式を表示するには、[SHOW TABLE STATUS](#) ステートメントを発行するか、[INFORMATION_SCHEMA](#) で InnoDB テーブルメタデータをクエリーします。

[コンパクト行フォーマット](#), [圧縮行フォーマット](#), [圧縮](#), [動的行フォーマット](#), [冗長行フォーマット](#), [行](#), [テーブル](#)も参照

行ベースレプリケーション

レプリカ上の個々の行の変更方法を指定する source からイベントが伝播されるレプリケーションの形式。

[innodb_autoinc_lock_mode](#) オプションのすべての設定に安全に使用できます。

[自動インクリメントロック](#), [innodb_autoinc_lock_mode](#), [レプリカ](#), [レプリケーション](#), [source](#), [ステートメントベースレプリケーション](#)も参照

行レベルロック

InnoDB テーブルに使用されるロックメカニズム。テーブルロックではなく行ロックに依存します。複数のトランザクションが同時に同じテーブルを変更できます。2つのトランザクションが同じ行を変更しようとした場合にのみ、トランザクションの一方は他方が終わる(およびその行ロックを解放する)まで待機します。

[InnoDB](#), [ロック](#), [行ロック](#), [テーブルロック](#), [トランザクション](#)も参照

行ロック

ロックの1つ。互換性のない方法で別のトランザクションが行にアクセスするのを防ぎます。同じテーブル内のほかの行には、ほかのトランザクションが自由に書き込むことができます。これは、InnoDB テーブルでの DML 操作によって行われるタイプのロックです。

[MyISAM](#) で使用されるテーブルロックとは対照的に、またはオンライン DDLでは実行できない [InnoDB](#) テーブルに対する DDL 操作中には、これらのロックによってテーブルへの同時アクセスがブロックされます。

[DDL](#), [DML](#), [InnoDB](#), [ロック](#), [ロック](#), [オンライン DDL](#), [テーブルロック](#), [トランザクション](#)も参照

ケ

原子的

SQL コンテキストでは、トランザクションとは、完全に完了する(コミット時)、またはまったく効果がない(ロールバック時)作業の単位です。トランザクションの不可視(アトミック)プロパティは、頭字語 ACID の「A」です。

[ACID](#), [コミット](#), [ロールバック](#), [トランザクション](#)も参照

厳密モード

`innodb_strict_mode` オプションで制御される設定の一般名。この設定をオンにすると、通常は警告として扱われる条件がエラーと見なされます。たとえば、通常は警告を生成してデフォルト値で続行するファイル形式と行フォーマットに関連するオプションの特定の無効な組合せによって、`CREATE TABLE` 操作が失敗するようになりました。`innodb_strict_mode` は MySQL 5.7 でデフォルトで有効になっています。

MySQL にも厳密モードと呼ばれるものがあります。[セクション5.1.11「サーバー SQL モード」](#)を参照してください。[ファイル形式](#), [innodb_strict_mode](#), [行フォーマット](#)も参照

検索インデックス

MySQL では、全文検索クエリーは特別な種類のインデックス (FULLTEXT インデックス) を使用します。MySQL 5.6.4 以降で、[InnoDB](#) および [MyISAM](#) テーブルの両方は FULLTEXT インデックスをサポートします。以前はこれらのインデックスは [MyISAM](#) テーブルでのみ利用可能でした。

[全文検索](#), [FULLTEXT インデックス](#)も参照

結合

同じ値を保持するテーブル内のカラムを参照することによって、複数のテーブルからデータを取得するクエリー。理想的には、これらのカラムは [InnoDB](#) 外部キー関係の一部であり、参照整合性および結合カラムがインデックス付きであることを確認します。多くの場合、正規化データ設計で、繰り返される文字列を数値 ID に置き換えることによって領域を節約してクエリーパフォーマンスを改善するために、使用されます。

[外部キー](#), [インデックス](#), [正規化](#), [クエリー](#), [参照整合性](#)も参照

計画安定性

クエリー実行計画のプロパティの1つ。オプティマイザが渡されたクエリーに毎回同じ選択を行うことで、パフォーマンスが一貫して予測可能になります。

[クエリー](#), [クエリー実行計画](#)も参照

コ

コネクタ

MySQL コネクタは、client プログラム用の MySQL サーバーへの接続を提供します。いくつかのプログラミング言語およびフレームワークには、それぞれ独自のコネクタが関連付けられています。API によって提供される下位レベルのアクセスと対比してください。

[API](#), [クライアント](#), [Connector/C++](#), [Connector/J](#), [Connector/NET](#), [Connector/ODBC](#)も参照

コマンドインタセプタ

ステートメントインターセプタと同義です。Connector/NET と Connector/J の両方で使用可能な interceptor デザインパターン的一部分。Connector/NET がコマンドをコールする内容。Connector/J はステートメントと呼ばれます。例外インターセプタと対比してください。

[Connector/J](#), [Connector/NET](#), [例外インターセプタ](#), [interceptor](#), [ステートメントインターセプタ](#)も参照

コミット

トランザクションを終了させ、トランザクションによって行われた変更を永続的にする SQL ステートメント。これは、トランザクションで行われた変更を元どおりにするロールバックの反対です。

InnoDB では、コミットに optimistic メカニズムを使用するため、コミットが実際に発生する前に変更をデータファイルに書き込むことができます。この方法は、コミット自体をより高速にしますが、ロールバックの場合には必要な作業が増えるというトレードオフが生じます。

MySQL はデフォルトで、各 SQL ステートメントに続いてコミットを自動的に発行する自動コミット設定を使用します。[自動コミット](#), [オプティミスティック](#), [ロールバック](#), [SQL](#), [トランザクション](#)も参照

コンパクト行フォーマット

InnoDB テーブル用の行フォーマット。これは、MySQL 5.0.3 から MySQL 5.7.8 へのデフォルトの行形式でした。MySQL 8.0 では、デフォルトの行フォーマットは、DYNAMIC のデフォルト設定を持つ `innodb_default_row_format` 構成オプションによって定義されます。COMPACT の行形式では、REDUNDANT の行形式よりも NULL および可変長のカラムをよりコンパクトに表現できます。

InnoDB COMPACT 行フォーマットの詳細は、[セクション15.10「InnoDB の行フォーマット」](#)を参照してください。[動的行フォーマット](#), [ファイル形式](#), [冗長行フォーマット](#), [行フォーマット](#)も参照

コールドバックアップ

データベースがシャットダウンしている間に行われるバックアップ。ビジー状態のアプリケーションおよび web サイトの場合、これは実用的ではなく、ウォームバックアップまたはホットバックアップが望ましい場合があります。

[バックアップ](#), [ホットバックアップ](#), [ウォームバックアップ](#)も参照

合成キー

インデックス付けされたカラム (通常は主キー)。値は任意に割り当てられます。多くの場合、自動インクリメントカラムを使用して行われます。完全に任意として値を扱うことによって、過度に制限されたルールと欠陥のあるアプリケーション想定を回避できます。たとえば、ある従業員が雇用を承認されたけれども、実際には入社していない場合、従業員数を表す数値シーケンスにギャップがあります。または、従業員番号 100 と従業員番号 500 が会社を退職してあとで再入社した場合、前者が後者よりもあとの雇用日になることがあります。数値も、予測可能な長さより短い値になります。たとえば、「Road」、「Boulevard」、「Expressway」などを意味する数値コードを格納すると、これらの文字列を繰り返し使用するよりも領域効率が高くなります。

サロゲートキーとも呼ばれます。ナチュラルキーと対比してください。

[自動インクリメント](#), [ナチュラルキー](#), [主キー](#), [サロゲートキー](#)も参照

固定行フォーマット

この行フォーマットは、InnoDB ではなく MyISAM ストレージエンジンによって使用されます。MySQL 5.7.6 以前で `ROW_FORMAT=FIXED` オプションを指定して InnoDB テーブルを作成した場合、InnoDB はコンパクトな行形式をかわりに使用しますが、FIXED の値が `SHOW TABLE STATUS` レポートなどの出力に表示されることがあります。MySQL 5.7.7 では、`ROW_FORMAT=FIXED` が指定されている場合、InnoDB はエラーを返します。

[コンパクト行フォーマット](#), [行フォーマット](#)も参照

子テーブル

外部キー関係で子テーブルとは、その行が別のテーブル内の行を参照 (またはポイント) し、特定の列について同じ値を持つもののことです。これは、`FOREIGN KEY ... REFERENCES` 句、およびオプションで `ON UPDATE` および `ON DELETE` 句を含むテーブルです。行を子テーブル内に作成するには、その前に親テーブル内に対応する行が存在している必要があります。子テーブル内の値は、外部キーの作成時に使用される `ON CASCADE` に基づいて、親テーブルでの削除または更新操作を禁止したり、子テーブル内で自動的に削除または更新したりする場合があります。

[外部キー](#), [親テーブル](#)も参照

構成ファイル

起動時に MySQL が使用するオプション値を保持するファイル。従来、Linux および Unix では、このファイルの名前は `my.cnf` で、Windows では `my.ini` です。ファイルの `[mysqld]` セクションで、InnoDB に関連した多数のオプションを設定できます。

MySQL が構成ファイルを検索する場所の詳細は、[セクション4.2.2.2「オプションファイルの使用」](#)を参照してください。

MySQL Enterprise Backup 製品を使用する場合、通常は 2 つの構成ファイルを使用: データの取得元とその構造化方法 (サーバーの元の構成ファイルである可能性がある) を指定するものと、バックアップデータの移動先と構造化方法を指定する小さ

なオプションセットのみを含むストリップダウンされたもの。MySQL Enterprise Backup 製品で使用される構成ファイルには、通常は通常の構成ファイルから除外される特定のオプションが含まれている必要があるため、MySQL Enterprise Backup で使用するために既存の構成ファイルにオプションを追加する必要があります。

[my.cnf](#), [MySQL Enterprise Backup](#), [オプション](#), [オプションファイル](#)も参照

混在モード挿入

`INSERT` ステートメントの 1 つ。自動インクリメント値が新しい行のすべてではなく一部に指定されます。たとえば、複数値 `INSERT` では、一部のケースで自動インクリメントカラムの値を、ほかのケースで `NULL` を指定できます。InnoDB は、カラム値が `NULL` として指定された行に自動インクリメント値を生成します。別の例が、`INSERT ... ON DUPLICATE KEY UPDATE` ステートメントです。ここでは、`INSERT` ではなく `UPDATE` ステートメントとして処理される重複行がある場合、それらに自動インクリメント値が生成されますが、使用されません。

レプリケーション構成の source サーバーとレプリカサーバーの間で整合性の問題が発生する可能性があります。

`innodb_autoinc_lock_mode` 構成オプションの値の調整が必要な場合があります。

[自動インクリメント](#), [innodb_autoinc_lock_mode](#), [レプリカ](#), [レプリケーション](#), [source](#)も参照

降順インデックス

`ORDER BY column DESC` 句を処理するためにインデックス記憶域が最適化される index のタイプ。

[インデックス](#)も参照

高位境界値

上限を表す値。実行時に超えるべきでないハード制限、または実際に到達した最大値の記録。低位境界値と対比してください。

[低位境界値](#)も参照

高速インデックス作成

InnoDB プラグインで最初に導入された機能は、5.5 以上の MySQL の一部であり、関連付けられたテーブルを完全にリライトする必要がなくなるため、InnoDB セカンダリインデックスの作成が高速化されます。高速化はセカンダリインデックスの削除にも適用されます。

インデックスを保守することで多数のデータ転送操作にパフォーマンスオーバーヘッドを増やす場合があるので、セカンダリインデックスを用意せずに `ALTER TABLE ... ENGINE=INNODB` や `INSERT INTO ... SELECT * FROM ...` などの操作を行い、あとでインデックスを作成することを検討してみてください。

MySQL 5.6 では、この機能がより一般的になります。インデックスの作成中にテーブルの読取りおよび書込みを行うことができ、テーブルをコピーせずに、または DML 操作をブロックせずに、あるいはその両方で、より多くの種類の `ALTER TABLE` 操作を実行できます。したがって、MySQL 5.6 以上では、この機能セットは高速インデックス作成ではなくオンライン DDL と呼ばれます。

関連情報については、[セクション15.12「InnoDB とオンライン DDL」](#)を参照してください。

[DML](#), [インデックス](#), [オンライン DDL](#), [セカンダリインデックス](#)も参照

高速シャットダウン

`innodb_fast_shutdown=1` の構成設定に基づく、InnoDB のデフォルトの shutdown プロシージャ。時間を節約するために、特定のフラッシュ操作がスキップされます。フラッシュ操作は次の起動中にクラッシュリカバリの場合と同じメカニズムを使用して実行されるので、通常の使用中にはこのタイプのシャットダウンが安全です。アップグレードまたはダウングレードのためにデータベースをシャットダウンしている場合は、代わりに低速シャットダウンを行なって、すべての該当する変更がシャットダウン中にデータファイルに適用されることを保証してください。

[クラッシュリカバリ](#), [データファイル](#), [フラッシュ](#), [シャットダウン](#), [低速シャットダウン](#)も参照

サ

サブリスト

バッファプールを表すリスト構造内では、比較的古いページと比較的新しいページは、list の様々な部分によって表されます。パラメータセットは、これらの部分のサイズと、新しいページと古いページ間の分割ポイントを制御します。

[バッファプール](#), [エビクション](#), [リスト](#), [LRU](#)も参照

サロゲートキー

合成キーのシノニム名。

[合成キー](#)も参照

サーバー

継続的に実行され、別のプログラム (client) からのリクエストの受信および処理を待機するプログラムのタイプ。多くの場合、コンピュータ全体が 1 つ以上のサーバープログラムを実行することを目的とするため (データベースサーバー、Web サーバー、アプリケーションサーバー、これらの組み合わせなど)、用語サーバーはサーバーソフトウェアを実行するコンピュータを指す場合もあります。

[クライアント](#)、[mysqld](#)も参照

サーバー側のプリペアドステートメント

MySQL サーバーによって管理されるプリペアドステートメント。これまで、サーバー側のプリペアドステートメントの問題により、Connector/J および Connector/PHP 開発者はかわりにクライアント側のプリペアドステートメントを使用する場合があります。最新の MySQL サーバーバージョンでは、パフォーマンス、スケーラビリティ、およびメモリー効率を向上させるために、サーバー側のプリペアドステートメントをお勧めします。

[クライアント側のプリペアドステートメント](#)、[Connector/J](#)、[Connector/PHP](#)、[プリペアドステートメント](#)も参照

先読み

pages(extent 全体) のグループをバッファプールに非同期にプリフェッチする I/O リクエストのタイプ。これらのページがまもなく必要になる場合に使用します。線形先読み手法では、前のエクステントのページのアクセスパターンに基づいて、1 つのエクステントのすべてのページがプリフェッチされます。ランダム先読み方法は、エクステントから特定数のページがバッファプールに入ると、同じエクステントのすべてのページをプリフェッチするものです。ランダム先読みは、MySQL 5.5 には組み込まれていませんが、[innodb_random_read_ahead](#) 構成オプションの制御下で、MySQL 5.6 に再導入されています。

[バッファプール](#)、[エクステント](#)、[ページ](#)も参照

削除

InnoDB が DELETE ステートメントを処理すると、行はすぐに削除対象としてマークされ、クエリーによって返されなくなります。記憶域は、後で、purge 操作と呼ばれる定期的なガベージコレクション中に再利用されます。大量のデータを削除する場合、独自のパフォーマンス特性を持つ関連操作は TRUNCATE および DROP です。

[ドロップ](#)、[ページ](#)、[切り捨て](#)も参照

削除バッファリング

ランダムな I/O を最小限に抑えるために物理書込みを実行できるように、変更を即座に書き込むのではなく、DELETE 操作によって生成されたセカンダリインデックスページへの変更を変更バッファに格納する方法。(削除操作は 2 ステッププロセスなので、この操作は、通常はインデックスレコードに削除とマークする書き込みをバッファに入れます。) これは変更バッファリングの一種です。ほかのタイプには挿入バッファリングとページバッファリングがあります。

[変更バッファ](#)、[変更バッファリング](#)、[挿入バッファ](#)、[挿入バッファリング](#)、[ページバッファリング](#)も参照

参照整合性

常に一貫する形式でデータを保守する方法。ACID 概念の一部です。特に、異なるテーブル内のデータは外部キー制約の使用を通じて一貫性が保持され、これによって変更が発生するのを防止したり、それらの変更をすべての関連テーブルに自動的に伝播したりできます。関連メカニズムには、重複値が間違っって挿入されるのを防ぐ一意制約と、ブランク値が間違っって挿入されるのを防ぐ NOT NULL 制約が含まれます。

[ACID](#)、[FOREIGN KEY 制約](#)、[NOT NULL 制約](#)、[一意制約](#)も参照

最大下限レコード

インデックス内の疑似レコードで、そのインデックスの最小値を下回るギャップを表します。トランザクションに [SELECT ... FROM ... WHERE col < 10 FOR UPDATE](#); などのステートメントがあり、カラム内の最小値が 5 の場合、他のトランザクションが 0 や -10 などの小さい値を挿入できないようにするための、最小レコードに対するロックです。

[ギャップ](#)、[インデックス](#)、[疑似レコード](#)、[最小上限レコード](#)も参照

最小上限レコード

インデックス内の疑似レコードで、そのインデックスの最大値を上回るギャップを表します。トランザクションに [SELECT ... FROM ... WHERE col > 10 FOR UPDATE](#); などのステートメントがあり、カラムの最大値が 20 の場合、他のトランザクションが 50、100 などの大きな値を挿入できないようにするために、そのトランザクションは最高レコードをロックします。

[ギャップ](#)、[最大下限レコード](#)、[疑似レコード](#)も参照

シ

システムテーブルスペース

InnoDB 関連オブジェクトのメタデータ、変更バッファの記憶域および二重書込みバッファを含む 1 つ以上のデータファイル (ibdata ファイル)。file-per-table または一般テーブルスペースではなくシステムテーブルスペースにテーブルが作成された場

合、InnoDB テーブルのテーブルおよびインデックスデータが含まれることもあります。システムテーブルスペースのデータおよびメタデータは、MySQL instance のすべてのデータベースに適用されます。

MySQL 5.6.7 より前のデフォルトでは、すべての InnoDB テーブルおよびインデックスがシステムテーブルスペース内に保持されるため、多くの場合、このファイルは非常に大きくなりました。システムテーブルスペースは決して縮小しないので、大容量の一時データがロードされてから削除された場合、ストレージの問題が発生する可能性があります。MySQL 8.0 では、デフォルトは file-per-table モードで、各テーブルとそれに関連付けられたインデックスは別々の .ibd ファイルに格納されます。このデフォルトでは、テーブル compression、オフページカラムの効率的な格納、大規模なインデックスキー接頭辞など、DYNAMIC および COMPRESSED の行形式に依存する InnoDB 機能を簡単に使用できます。

すべてのテーブルデータをシステムテーブルスペースまたは別個の .ibd ファイルのどちらかに保持するかは、ストレージ管理全般に影響します。MySQL Enterprise Backup 製品は、大きなファイルの小さなセットをバックアップすることも、多数のより小さなファイルをバックアップすることもできます。何千ものテーブルがあるシステムでは、何千もの .ibd ファイルを処理するファイルシステム操作によってボトルネックが発生する可能性があります。

InnoDB では、MySQL 5.7.6 に一般的なテーブルスペースが導入されました。これらは .ibd ファイルでも表されます。一般テーブルスペースは、CREATE TABLESPACE 構文を使用して作成される共有テーブルスペースです。これらはデータディレクトリの外部で作成でき、複数のテーブルを保持でき、すべての行形式のテーブルをサポートします。[変更バッファ](#)、[圧縮](#)、[データディクショナリ](#)、[データベース](#)、[二重書き込みバッファ](#)、[動的行フォーマット](#)、[file-per-table](#)、[一般テーブルスペース](#)、[.ibd ファイル](#)、[ibdata ファイル](#)、[innodb_file_per_table](#)、[インスタンス](#)、[MySQL Enterprise Backup](#)、[オフページカラム](#)、[テーブルスペース](#)、[Undo ログ](#)も参照

シャットダウン

MySQL Server を停止させるプロセス。デフォルトでは、このプロセスは InnoDB テーブルの操作をクリーンアップするため、InnoDB は低速で停止できますが、後で高速に起動できます。クリーンアップ操作をスキップすると、高速で停止しますが、次の再起動時にクリーンアップを実行する必要があります。

InnoDB の停止モードは、[innodb_fast_shutdown](#) オプションによって制御されます。[高速シャットダウン](#)、[InnoDB](#)、[低速シャットダウン](#)、[起動](#)も参照

シャープチェックポイント

すべてのデータページバッファプールページ (その Redo エントリが Redo ログのどこかに含まれている) をディスクにフラッシュするプロセス。InnoDB がログファイルの一部を再利用する前に発生します。ログファイルは循環的に使用されます。通常は、書き込みの多いワークロードで発生します。[データページ](#)、[フラッシュ](#)、[Redo ログ](#)、[ワークロード](#)も参照

シリアライズディクショナリ情報 (SDI)

シリアライズされた形式のディクショナリオブジェクトメタデータ。SDI は JSON 形式で格納されます。

MySQL 8.0.3 では、SDI は一時テーブルスペースおよび undo テーブルスペースファイルを除くすべての InnoDB テーブルスペースファイルに存在します。SDI がテーブルスペースファイルに存在すると、メタデータの冗長性が提供されます。たとえば、データディクショナリが使用できなくなった場合は、[ibd2sdi](#) ユーティリティを使用してテーブルスペースファイルからディクショナリオブジェクトメタデータを抽出できます。

MyISAM テーブルの場合、SDI はスキーマディレクトリの .sdi メタデータファイルに格納されます。IMPORT TABLE 操作を実行するには SDI メタデータファイルが必要です。[file-per-table](#)、[一般テーブルスペース](#)、[システムテーブルスペース](#)、[テーブルスペース](#)も参照

主キー

テーブル内のすべての行を一意に識別できる、このカラムセットに基づくインデックス。したがって、NULL 値を一切含まない一意インデックスである必要があります。

InnoDB では、すべてのテーブルにこのようなインデックス (クラスタインデックスまたはクラスタインデックスとも呼ばれます) があり、主キーのカラム値に基づいてテーブル記憶域を編成する必要があります。

主キー値を選択するときは、ほかの何らかのソースから派生した値 (ナチュラルキー) に依存するのではなく、任意の値 (合成キー) を使用することを検討してください。

[クラスタ化されたインデックス](#)、[インデックス](#)、[ナチュラルキー](#)、[合成キー](#)も参照

冗長行フォーマット

最も古い InnoDB 行フォーマット。MySQL 5.0.3 より前は、これが InnoDB で利用できる唯一の行フォーマットでした。MySQL 5.0.3 から MySQL 5.7.8 へのデフォルトの行フォーマットは COMPACT です。MySQL 5.7.9 では、デフォルトの行

フォーマットは、DYNAMIC のデフォルト設定を持つ `innodb_default_row_format` 構成オプションによって定義されます。古い InnoDB テーブルとの互換性のために、引き続き REDUNDANT の行形式を指定できます。

詳細は、[セクション15.10「InnoDB の行フォーマット」](#)を参照してください。
[コンパクト行フォーマット](#)、[動的行フォーマット](#)、[行フォーマット](#)も参照

準備されたバックアップ

バックアップファイルセットの 1 つ。バイナリログおよび増分バックアップを適用するすべての段階が終了したあとに、MySQL Enterprise Backup 製品によって生成されます。結果ファイルは、リストアできる状態です。適用ステップより前のファイルは raw バックアップと呼ばれます。
[バイナリログ](#)、[ホットバックアップ](#)、[増分バックアップ](#)、[MySQL Enterprise Backup](#)、[raw バックアップ](#)、[リストア](#)も参照

自動インクリメント

カラム内に昇順で値を自動的に追加する、テーブルカラムのプロパティー (`AUTO_INCREMENT` キーワードで指定します)。

これにより、開発者の作業が節約され、新しい行を挿入するときに新しい一意値を生成する必要はありません。カラムには NULL でなく一意値が含まれているとわかっているので、クエリー最適化に有用な情報をもたらします。このようなカラムからの値は、さまざまなコンテキストでルックアップキーとして使用でき、自動生成されるので絶えず変更する理由はありません。このため、多くの場合、主キーカラムは自動インクリメントとして指定されます。

タイミングの問題により、レプリカでステートメントをリプレイしてもソースと同じカラム値のセットが生成されない場合があります。ステートメントベースのレプリケーションでは自動増分カラムに問題が発生する可能性があります。自動インクリメントする主キーがある場合は、`innodb_autoinc_lock_mode=1` の設定だけでステートメントベースレプリケーションを使用できます。挿入操作でより高い並列性を許可する `innodb_autoinc_lock_mode=2` を使用する場合は、ステートメントベースレプリケーションではなく行ベースレプリケーションを使用してください。互換性の目的以外は、`innodb_autoinc_lock_mode=0` の設定を使用しないでください。

連続ロックモード (`innodb_autoinc_lock_mode=1`) は、MySQL 8.0.3 より前のデフォルト設定です。MySQL 8.0.3 の時点では、インターリーブロックモード (`innodb_autoinc_lock_mode=2`) がデフォルトであり、デフォルトのレプリケーションタイプとしてステートメントベースから行ベースのレプリケーションへの変更が反映されます。

[自動インクリメントロック](#)、[innodb_autoinc_lock_mode](#)、[主キー](#)、[行ベースレプリケーション](#)、[ステートメントベースレプリケーション](#)も参照

自動インクリメントロック

自動インクリメント主キーの利便性には、並列性とのトレードオフが若干伴います。もっとも単純なケースでは、あるトランザクションがテーブルに値を挿入している場合に、ほかのトランザクションはそのテーブルへのそれぞれの挿入を待機する必要があります。最初のトランザクションによって挿入された行が、連続する主キー値を受け取ります。InnoDB には最適化と `innodb_autoinc_lock_mode` オプションが含まれているため、自動インクリメント値の予測可能なシーケンスと挿入操作のための最大同時実行性の中で最適なバランスを構成できます。

[自動インクリメント](#)、[並列性](#)、[innodb_autoinc_lock_mode](#)も参照

自動コミット

各 SQL ステートメントのあとにコミット操作を実行する設定。複数のステートメントにまたがる transactions で InnoDB テーブルを操作する場合、このモードはお勧めしません。これは、特に MySQL 5.6.4 以上で、ロックからのオーバーヘッドおよび元に戻すデータの生成を最小限に抑える InnoDB テーブルの読み取り専用トランザクションのパフォーマンスに役立ちます。また、トランザクションを適用できない MyISAM テーブルの操作にも適しています。

[コミット](#)、[ロック](#)、[読み取り専用トランザクション](#)、[SQL](#)、[トランザクション](#)、[Undo](#)も参照

親テーブル

子テーブルに (から) ポイントされた初期カラム値を保持する、外部キー関係のテーブル。親テーブル内の行を削除または更新したときの結果は、外部キー定義の `ON UPDATE` および `ON DELETE` 句によって異なります。子テーブル内の対応する値の行が、それに合わせて自動的に削除または更新されたり、これらのカラムが NULL に設定されたり、操作が妨げられたりします。

[子テーブル](#)、[外部キー](#)も参照

ス

スキーマ

概念的には、スキーマは、テーブル、テーブルカラム、カラムのデータ型、インデックス、外部キーなど、相互に関連するデータベースオブジェクトのセットです。カラムがテーブルを構成する、外部キーがテーブルやカラムを参照するなどの理由で、これらのオブジェクトは SQL 構文を通じて接続されます。理論的には、これらは論理的にも接続し、統合され

たアプリケーションまたは柔軟なフレームワークの一部として連携します。たとえば、`INFORMATION_SCHEMA` および `performance_schema` データベースでは、名前に「`schema`」を使用して、含まれるテーブルとカラムの間の密接な関係を強調します。

MySQL では、スキーマは物理的にデータベースと同義です。MySQL SQL 構文で、`DATABASE` の代わりにキーワード `SCHEMA` を代用できます。たとえば、`CREATE DATABASE` の代わりに `CREATE SCHEMA` を使用できます。

ほかのデータベース製品では区別しているものがあります。たとえば、Oracle Database 製品では、スキーマはデータベースの一部、つまり単一ユーザーが所有するテーブルおよびほかのオブジェクトだけを表します。

[データベース](#)、[INFORMATION_SCHEMA](#)、[パフォーマンススキーマ](#)も参照

スケーラビリティ

システム能力の限界を超えてもパフォーマンスが突然落ちることがなく、システムにより多くの作業を追加したりより多くの同時リクエストを発行したりできること。ソフトウェアアーキテクチャー、ハードウェア構成、アプリケーションコーディング、およびワークロードのタイプはすべて、スケーラビリティで役割を担います。システムがその最大能力に達したときにスケーラビリティを増やす一般的な方法には、スケールアップ (既存のハードウェアまたはソフトウェアの能力を増大させる) とスケールアウト (新しいサーバーやより多くの MySQL インスタンスを追加する) があります。多くの場合、大規模開発の重要な側面として、可用性と組み合わせられます。

[可用性](#)、[スケールアウト](#)、[スケールアップ](#)も参照

スケールアウト

新しいサーバーやより多くの MySQL インスタンスを追加することによってスケーラビリティを増大させる方法。たとえば、レプリケーション、NDB Cluster、接続プーリング、またはサーバーのグループ全体に機能を分散させるその他の機能を設定します。スケールアップと対比してください。

[スケーラビリティ](#)、[スケールアップ](#)も参照

スケールアップ

既存のハードウェアまたはソフトウェアの能力を高めることによりスケーラビリティを増大させる方法。たとえば、サーバー上でメモリーを増やすこと、`innodb_buffer_pool_size` や `innodb_buffer_pool_instances` などのメモリー関連パラメータを調整すること。スケールアウトと対比してください。

[スケーラビリティ](#)、[スケールアウト](#)も参照

ステミング

単数形と複数形、過去、現在、および未来時制など、共通語幹に基づいて単語のさまざまなバリエーションを検索する機能。この機能は現在、[MyISAM](#) 全文検索機能でサポートされていますが、[InnoDB](#) テーブルの `FULLTEXT` インデックスではサポートされていません。

[全文検索](#)、[FULLTEXT インデックス](#)も参照

ステートメントインターセプタ

データベースアプリケーションによって発行された SQL ステートメントをトレース、デバッグまたは拡張するための `interceptor` のタイプ。コマンドインターセプタとも呼ばれます。

Connector/J を使用する Java アプリケーションでは、このタイプのインターセプタを設定するには、`com.mysql.jdbc.StatementInterceptorV2` インタフェースを実装し、`statementInterceptors` プロパティを接続文字列に追加します。

Connector/NET を使用する Visual Studio アプリケーションでは、このタイプのインターセプタを設定するには、`BaseCommandInterceptor` クラスから継承するクラスを定義し、そのクラス名を接続文字列の一部として指定する必要があります。

[コマンドインターセプタ](#)、[接続文字列](#)、[Connector/J](#)、[Connector/NET](#)、[interceptor](#)、[Java](#)、[Visual Studio](#)も参照

ステートメントベースレプリケーション

SQL ステートメントが `source` から送信され、レプリカでリプレイされるレプリケーションの形式。auto-increment locking の潜在的なタイミング問題を回避するために、`innodb_autoinc_lock_mode` オプションの設定には注意が必要です。

[自動インクリメントロック](#)、[innodb_autoinc_lock_mode](#)、[レプリカ](#)、[レプリケーション](#)、[行ベースレプリケーション](#)、[source](#)も参照

ストアドオブジェクト

ストアドプログラムまたはビュー。

ストアドプログラム

ストアドルーチン (プロシージャまたはファンクション)、トリガー、またはイベントスケジューライイベント。

ストアルーチン

ストアドプロシージャまたはファンクション。

ストアド生成カラム

カラム定義に含まれる式から値が計算されるカラム。カラム値は、行の挿入または更新時に評価および格納されます。格納された生成カラムには記憶領域が必要で、インデックス付けできません。

仮想生成カラムと対比してください。

[ベースカラム](#)、[生成されるカラム](#)、[仮想生成カラム](#)も参照

ストップワード

FULLTEXT インデックスで、十分に一般的または些細なので検索インデックスから除外し、検索クエリーで無視できると考えられている単語。InnoDB テーブルと MyISAM テーブルとは、異なる構成設定がストップワード処理を制御します。詳細は、[セクション12.10.4「全文ストップワード」](#)を参照してください。

[FULLTEXT インデックス](#)、[検索インデックス](#)も参照

ストレージエンジン

MySQL データベースのコンポーネントの 1 つ。データの格納、更新、および照会という低レベル作業を実行します。MySQL 5.5 以上では、InnoDB が新しいテーブルのデフォルトのストレージエンジンであり、MyISAM よりも優先されます。メモリー使用とディスク使用、読み取り速度と書き込み速度、速度と堅牢性など、異なる要因間トレードオフのために異なるストレージエンジンが設計されています。各ストレージエンジンが特定のテーブルを管理するので、InnoDB テーブル、MyISAM テーブルなどと呼ばれます。

MySQL Enterprise Backup 製品は、InnoDB テーブルのバックアップ用に最適化されています。また、MyISAM およびその他のストレージエンジンによって処理されるテーブルをバックアップすることもできます。

[InnoDB](#)、[MySQL Enterprise Backup](#)、[テーブルタイプ](#)も参照

スナップショット

特定時点のデータの表現。ほかのトランザクションによって変更がコミットされても変化しません。一貫性読み取りを許可する分離レベルで使用されます。

[コミット](#)、[一貫性読み取り](#)、[分離レベル](#)、[トランザクション](#)も参照

スパースファイル

実際の空の領域を書き込むのではなく、空のブロックを表すメタデータをディスクに書き込むことで、ファイルシステム領域をより効率的に使用するファイルのタイプ。InnoDB 透過的ページ圧縮機能は、スパースファイルサポートに依存します。詳細は、[セクション15.9.2「InnoDB ページ圧縮」](#)を参照してください。

[ホールパンチング](#)、[透過的ページ圧縮](#)も参照

スピン

リソースが利用できるようになるかどうかを継続的にテストするタイプの待機操作。この手法は、スレッドをスリープさせてコンテキスト切替えを実行するよりも「ビジーループ」で待機する方が効率的な短い期間のみ保持されるリソースに使用されます。リソースが短時間で利用できなくなると、スピンループは中止し、別の待機方法が使用されます。

[ラッチ](#)、[ロック](#)、[相互排他ロック](#)、[待機](#)も参照

スペース ID

MySQL インスタンス内で InnoDB テーブルスペースを一意に識別するために使用される識別子。システムテーブルスペースの領域 ID は常にゼロです。この同じ ID は、システムテーブルスペース内または一般テーブルスペース内のすべてのテーブルに適用されます。各 file-per-table テーブルスペースおよび一般テーブルスペースには、独自の領域 ID があります。

MySQL 5.6 より前では、このハードコードされた値によって、MySQL インスタンス間で InnoDB テーブルスペースファイルを移動することが困難でした。MySQL 5.6 以降では、ステートメント [FLUSH TABLES ... FOR EXPORT](#)、[ALTER TABLE ... DISCARD TABLESPACE](#)、および [ALTER TABLE ... IMPORT TABLESPACE](#) を含むトランスポータブルテーブルスペース機能を使用することによって、インスタンス間でテーブルスペースファイルをコピーできます。スペース ID を調整するために必要な情報は、テーブルスペースとともにコピーする .cfg ファイルで伝えられます。詳細は、[セクション15.6.1.3「InnoDB テーブルのインポート」](#)を参照してください。

[.cfg ファイル](#)、[file-per-table](#)、[一般テーブルスペース](#)、[.ibd ファイル](#)、[システムテーブルスペース](#)、[テーブルスペース](#)、[トランスポータブルテーブルスペース](#)も参照

スレッド

通常はプロセスより軽量で、高度な並列性に対応できる処理単位。

[並列性](#)、[マスタースレッド](#)、[プロセス](#)、[Pthreads](#)も参照

スレーブ

[レプリカ](#)も参照

スロークエリーログ

MySQL Server によって処理される SQL ステートメントのパフォーマンスチューニングに使用されるタイプのログ。ログ情報はファイルに格納されます。使用するにはこの機能を有効にする必要があります。ログに記録する「低速」SQL ステートメントのカテゴリを制御します。詳細は、[セクション5.4.5「スロークエリーログ」](#)を参照してください。

[一般クエリーログ](#)、[ログ](#)も参照

既読現象

ダーティリード、反復不可能な読み取り、phantom などの現象は、トランザクションが別のトランザクションによって変更されたデータを読み取るときに発生する可能性があります。

[ダーティ読み取り](#)、[反復不可能読み取り](#)、[ファントム](#)も参照

セ

セカンダリインデックス

テーブルのカラムのサブセットを表す InnoDB index のタイプ。InnoDB テーブルには、ゼロ、1 つまたは複数のセカンダリインデックスを含めることができます。(各 InnoDB テーブルに必要なクラスタインデックスと対比して、すべてのテーブルのカラムのデータを格納します。)

セカンダリインデックスは、インデックスカラムからの値だけを必要とするクエリーを満たすために使用できます。より複雑なクエリーの場合、テーブル内の該当行を識別するために使用でき、それらはクラスタ化されたインデックスを使用するルックアップで取得されます。

セカンダリインデックスの作成および削除には、従来、InnoDB テーブルのすべてのデータのコピーによる大きなオーバーヘッドが伴いました。高速インデックス作成機能を使用すると、InnoDB セカンダリインデックスに対して CREATE INDEX ステートメントと DROP INDEX ステートメントの両方がはるかに高速になります。

[クラスタ化されたインデックス](#)、[高速インデックス作成](#)、[インデックス](#)も参照

セグメント

InnoDB テーブルスペース内のディビジョン。テーブルスペースをディレクトリに例えると、セグメントはそのディレクトリ内のファイルに似ています。セグメントは増えることができます。新しいセグメントを作成できます。

たとえば、file-per-table テーブルスペース内では、テーブルデータは 1 つのセグメントにあり、関連付けられた各インデックスは独自のセグメントにあります。システムテーブルスペースは、多くのテーブルとそれらに関連付けられたインデックスを保持できるため、多くの異なるセグメントを含みます。MySQL 8.0 より前のシステムテーブルスペースには、undo ログに使用される 1 つ以上のロールバックセグメントも含まれていました。

セグメントは、データの挿入と削除に応じて拡大、縮小します。セグメントがより多くの領域を必要とする場合、一度に 1 エクステント (1M バイト) ずつ拡張されます。同様に、セグメントは、あるエクステント内のすべてのデータが不要になると、そのエクステント分の領域を解放します。

[エクステント](#)、[file-per-table](#)、[ロールバックセグメント](#)、[システムテーブルスペース](#)、[テーブルスペース](#)、[Undo ログ](#)も参照

セッション一時テーブルスペース

InnoDB が内部一時テーブルのディスク上のストレージエンジンとして構成されている場合に、オプティマイザによって作成されたユーザー作成の一時テーブルおよび内部一時テーブルを格納する一時テーブルスペース。

[オプティマイザ](#)、[一時テーブル](#)、[一時テーブルスペース](#)も参照

セーブポイント

セーブポイントは、ネストされたトランザクションの実装に役立ちます。それらは、より大きなトランザクションの一部であるテーブル上での操作にスコープを提供するために使用できます。たとえば、予約システムで旅行をスケジューリングするときに、いくつかの異なる航空便を予約する場合があります。希望の航空便を利用できない場合、予約に成功したそれより早い航空便をロールバックすることなく、その 1 行程の予約に関与する変更をロールバックできます。

[ロールバック](#)、[トランザクション](#)も参照

全文検索

SQL LIKE 演算子を使用したり、アプリケーションレベル検索アルゴリズムを作成したりするよりも、より高速、より便利で、かつより柔軟な方法で単語、語句、単語のブール結合などをテーブルデータ内で検索するための MySQL 機能。これは、SQL 関数 MATCH() および FULLTEXT インデックスを使用します。

FULLTEXT インデックスも参照

制約

データが一貫性を失うのを防ぐために、データベース変更をブロックできる自動テスト。(コンピュータサイエンス用語では、不変条件に関連する一種のアサーション。) 制約は、データ一貫性を維持するための、ACID 概念の重要な構成要素です。MySQL によってサポートされる制約には、FOREIGN KEY 制約と一意制約があります。

[ACID](#), [外部キー](#), [一意制約](#)も参照

接続

アプリケーションと MySQL サーバー間の通信チャネル。データベースアプリケーションのパフォーマンスおよびスケラビリティは、データベース接続の確立速度、同時に実行できる数、および永続化の期間に影響されます。host、port などのパラメータは、Connector/NET では接続文字列として、Connector/ODBC では DSN として表されます。高トラフィックシステムは、接続プールと呼ばれる最適化を利用します。

[接続プール](#), [接続文字列](#), [Connector/NET](#), [Connector/ODBC](#), [DSN](#), [ホスト](#), [port](#)も参照

接続プール

データベース操作ごとに新しい接続を設定および切断するのではなく、同じアプリケーション内または異なるアプリケーション間でデータベース connections を再利用できるキャッシュ領域。この手法は、J2EE アプリケーションサーバーで共通です。Connector/J を使用する Java アプリケーションでは、Tomcat および他のアプリケーションサーバーの接続プール機能を使用できます。再利用はアプリケーションに対して透過的です。アプリケーションは通常どおり接続を開いて閉じます。

[接続](#), [Connector/J](#), [J2EE](#), [Tomcat](#)も参照

接続文字列

プログラムコードでできるように文字列リテラルとしてエンコードされた、データベース connection のパラメータの表現。文字列の一部は、host や port などの接続パラメータを表します。接続文字列には、セミコロンで区切られた複数のキーと値のペアが含まれます。各キーと値のペアは等号で結合されます。Connector/NET アプリケーションで頻繁に使用されます。詳細は、[Creating a Connector/NET Connection String](#) を参照してください。

[接続](#), [Connector/NET](#), [ホスト](#), [port](#)も参照

正規化

データベース設計戦略の 1 つ。データは複数のテーブルに分割されていて、圧縮された値を ID で表された単一行に複製することで、冗長または長い値を格納、照会、および更新することを回避します。通常は OLTP アプリケーションで使用されます。

たとえば、住所に一意 ID を与えることで、国勢調査データベースはその ID を家族の各メンバーに関連付けることによって、この住所の住居人という関係を表現できます (米国のある街のメインストリート 123 などの複雑な値のコピーを複数格納するのではなく)。

別の例としては、単純な住所録アプリケーションでは、ある人の名前および住所と同じテーブルにそれぞれの電話番号を格納しますが、電話会社データベースでは、各電話番号に特別な ID を与えてその番号と ID を別のテーブルに格納します。この正規化表現によって、市外局番が分かれるときの大幅な更新を簡略化できます。

正規化が常に推奨されるわけではありません。主に照会されるだけで、更新されるのは全体を削除してリロードする場合だけのデータは、多くの場合、重複値の冗長コピーを持つ少数の大きなテーブルで保持されます。このデータ表現は、非正規化と呼ばれ、データウェアハウスアプリケーションでよく見られます。

[非正規化](#), [外部キー](#), [OLTP](#), [リレーショナル](#)も参照

生成されたストアドカラム

[ストアド生成カラム](#)も参照

生成された仮想カラム

[仮想生成カラム](#)も参照

生成されるカラム

カラム定義に含まれる式から値が計算されるカラム。生成されるカラムは、virtual または stored です。

[ベースカラム](#), [ストアド生成カラム](#), [仮想生成カラム](#)も参照

選択性

データ分布のプロパティの 1 つ。カラム内の個別値の数 (そのカーディナリティー) をテーブル内のレコード数で割ったもの。高い選択性は、カラム値が比較的一意であり、インデックスを通じて効率的に取得できることを意味します。[WHERE](#) 句内のテストがテーブル内の少ない数 (または割合) の行にのみ一致すると予測できる場合 (またはクエリーオプティマイザが

そう予測する場合)、クエリーがインデックスを使用してそのテストを最初に評価すると、全体的に効率的である傾向があります。

[カーディナリティー](#)、[クエリー](#)も参照

静止

データベースアクティビティーの量を減らすこと。多くの場合、[ALTER TABLE](#)、バックアップ、シャットダウンなどの操作に備えるためです。最大限のフラッシュの実行を伴う場合があるため(そうでない場合もありますが)、InnoDB はバックグラウンド I/O の実行を継続しません。

MySQL 5.6 以降では、構文 [FLUSH TABLES ... FOR EXPORT](#) によって InnoDB テーブルの一部のデータがディスクに書き込まれるので、そのデータファイルをコピーすることでこれらのテーブルをより簡単にバックアップできます。

[バックアップ](#)、[フラッシュ](#)、[InnoDB](#)、[シャットダウン](#)も参照

ソ

ソートバッファ

InnoDB インデックスの作成中にデータをソートするために使用されるバッファ。ソートバッファサイズは、[innodb_sort_buffer_size](#) 構成オプションを使用して構成されます。

増分バックアップ

ある時点以降に変更されたデータだけを保存する、MySQL Enterprise Backup 製品で実行されるタイプのホットバックアップ。完全バックアップと一連の増分バックアップがあれば、いくつかの完全バックアップを手元においておくストレージオーバーヘッドなしで、長期間にわたるバックアップデータを再構築できます。完全バックアップをリストアしてから各増分バックアップを連続して適用したり、各増分バックアップを完全バックアップに適用することでこれを最新の状態に保った状態で単一リストア操作を実行したりできます。

変更データの粒度はページレベルです。実際は 1 つのページが複数の行をカバーすることがあります。変更された各ページがバックアップに含まれます。

[ホットバックアップ](#)、[MySQL Enterprise Backup](#)、[ページ](#)も参照

挿入

SQL での主要な DML 操作の 1 つ。挿入のパフォーマンスは、データウェアハウスシステム (数百万行をテーブルにロードする) と OLTP システム (多数の並列接続が行を同じテーブルに任意の順序で挿入する可能性がある) で重要な要因です。挿入パフォーマンスが重要な場合は、[変更バッファリング](#)で使用される挿入バッファや自動インクリメントカラムなどの InnoDB 機能を学習することをお勧めします。

[自動インクリメント](#)、[変更バッファリング](#)、[データウェアハウス](#)、[DML](#)、[InnoDB](#)、[挿入バッファ](#)、[OLTP](#)、[SQL](#)も参照

挿入バッファリング

ランダムな I/O を最小限に抑えるために物理書込みを実行できるように、変更を即座に書き込むのではなく、[INSERT](#) 操作によって生成されたセカンダリインデックスページへの変更を変更バッファに格納する方法。これは変更バッファリングの 1 つのタイプです。ほかに削除バッファリングとパーシバブルバッファリングがあります。

セカンダリインデックスが一意的な場合には挿入バッファリングは使用されません。新しいエントリが書き出される前に新しい値の一意性を検証できないためです。ほかの種類の変更バッファリングは一意インデックスに有効です。

[変更バッファ](#)、[変更バッファリング](#)、[削除バッファリング](#)、[挿入バッファ](#)、[パーシバブルバッファリング](#)、[一意のインデックス](#)も参照

挿入バッファ

変更バッファの以前の名前。MySQL 5.5 では、[DELETE](#) および [UPDATE](#) 操作のセカンダリインデックスページへの変更をバッファリングするためのサポートが追加されました。以前は、[INSERT](#) 操作による変更のみがバッファされていました。現在推奨されている用語は変更バッファです。

[変更バッファ](#)、[変更バッファリング](#)も参照

相互排他ロック

「mutex 変数」の非公式略称。(Mutex 自体は「相互排他」の短縮形です。) InnoDB が内部インメモリーデータ構造への排他的アクセスロックを表現および強制するために使用する低レベルオブジェクト。ロックが獲得されると、ほかのプロセスやスレッドなどは同じロックを獲得できなくなります。InnoDB が内部インメモリーデータ構造への共有アクセスロックを表現および強制するために使用する rw-locks と対比してください。相互排他ロックと読み書きロックはまとめて、ラッチと呼ばれます。

[ラッチ](#)、[ロック](#)、[パフォーマンススキーマ](#)、[Pthreads](#)、[rw ロック \(読み書きロック\)](#)も参照

タ

タプル

順序付けられた要素セットを指定する技術用語。これは抽象概念で、データベース理論の公式ディスカッションで使用されます。データベースフィールドでのタプルは通常、テーブル行のカラムによって表されます。これらはクエリー (テーブルの一部のカラムだけ、または結合されたテーブルからのカラムを取得したクエリー、など) の結果セットで表される場合もあります。

[カーソル](#)も参照

ダーティーページ

InnoDB バッファプール内の page は、メモリー内で更新され、データファイルに変更がまだ書き込まれていません (flushed)。クリーンページの反対です。

[バッファプール](#)、[クリーンページ](#)、[データファイル](#)、[フラッシュ](#)、[ページ](#)も参照

ダーティー読み取り

信頼できないデータ、つまり別のトランザクションによって更新されたけれども、まだコミットされていないデータを取得する操作。これは、コミットされた読み取りと呼ばれる分離レベルでのみ可能です。

この種の操作は、データベース設計の ACID 原則には準拠しません。これは非常にリスクが高いと見なされます。データをロールバックできたり、コミットされる前にさらに更新できたりするためです。この場合、ダーティー読み取りを行うトランザクションは、正確であると確定されていないデータを使用することになります。

その反対は読み取り一貫性であり、一方で他のトランザクションがコミットしても、InnoDB によって、別のトランザクションによって更新された情報が読み取られないことが保証されます。

[ACID](#)、[コミット](#)、[一貫性読み取り](#)、[分離レベル](#)、[READ UNCOMMITTED](#)、[ロールバック](#)も参照

待機

lock、mutex または latch の取得などの操作をすぐに完了できない場合、InnoDB は一時停止して再試行します。この一時停止のメカニズムはかなり入念に設計されているため、この操作には独自の名前、待機が付けられています。個々のスレッドは、内部 InnoDB スケジューリング、オペレーティングシステムの wait() コールおよび短期スピンドループの組合せを使用して一時停止されます。

負荷が高く、多くのトランザクションがあるシステムでは、SHOW INNODB STATUS コマンドまたはパフォーマンススキーマからの出力を使用して、スレッドが待機時間を過度に費やしているかどうか、および費やしている場合は同時実行性を改善する方法を判断できます。

[並列性](#)、[ラッチ](#)、[ロック](#)、[相互排他ロック](#)、[パフォーマンススキーマ](#)、[スピン](#)も参照

チ

チェックサム

InnoDB で、テーブルスペース内の page がディスクから InnoDB バッファプールに読み取られたときに破損を検出する検証メカニズム。この機能は、MySQL 5.5 の innodb_checksums 構成オプションによって制御されます。innodb_checksums は、innodb_checksum_algorithm に置き換えられた MySQL 5.6.3 で非推奨になりました。

innochecksum コマンドは、MySQL サーバーの停止中に指定されたテーブルスペースファイルのチェックサム値をテストすることで、破損の問題の診断に役立ちます。

MySQL はレプリケーションのためにチェックサムも使用します。詳細は、構成オプション

[binlog_checksum](#)、[master_verify_checksum](#)、および [slave_sql_verify_checksum](#) を参照してください。

[バッファプール](#)、[ページ](#)、[テーブルスペース](#)も参照

チェックポイント

バッファプールにキャッシュされているデータページに変更が行われると、これらの変更は少しあとからデータファイルに書き込まれます。これはフラッシュと呼ばれるプロセスです。チェックポイントは、データファイルに正しく書き込まれている (LSN 値で表される) 最新の変更のレコードです。

[バッファプール](#)、[データファイル](#)、[フラッシュ](#)、[ファジーチェックポイント](#)、[LSN](#)も参照

中規模の信頼

部分信頼と同義です。信頼設定の範囲は非常に広いため、「部分信頼」をお勧めします。これは、レベルが 3 つのみ (低、中および完全) であることを回避するためです。

[Connector/NET](#)、[部分信頼](#)も参照

テ

テキストコレクション

FULLTEXT インデックスに含まれるカラムセット。

[FULLTEXT インデックス](#)も参照

テーブル

各 MySQL テーブルは、特定のストレージエンジンに関連付けられます。InnoDB テーブルは、パフォーマンス、スケラビリティ、バックアップ、管理、およびアプリケーション開発に影響する、特定の物理および論理特性を持っています。

ファイル記憶域に関して、InnoDB テーブルは次のいずれかのテーブルスペースタイプに属します:

- 共有 InnoDB システムテーブルスペース。1 つ以上の ibdata ファイルで構成されます。
- 個々の .ibd ファイルで構成される file-per-table テーブルスペース。
- 個々の .ibd ファイルで構成される共有一般テーブルスペース。一般的なテーブルスペースは、MySQL 5.7.6 で導入されました。

.ibd データファイルには、テーブルデータと index データの両方が含まれます。

file-per-table テーブルスペースに作成された InnoDB テーブルでは、DYNAMIC または COMPRESSED の行形式を使用できます。これらの行形式により、compression などの InnoDB 機能、オフページカラムの効率的な格納、大規模なインデックスキー接頭辞が可能になります。一般テーブルスペースでは、すべての行形式がサポートされます。

システムテーブルスペースは、REDUNDANT、COMPACT および DYNAMIC の行形式を使用するテーブルをサポートしています。DYNAMIC 行形式のシステムテーブルスペースサポートが MySQL 5.7.6 で追加されました。

InnoDB テーブルの rows は、クラスタインデックスと呼ばれるインデックス構造に編成され、テーブルの主キーカラムに基づいてエントリがソートされます。データアクセスは主キーカラムでフィルタおよびソートするクエリーに最適化され、各インデックスには各エントリに関連付けられた主キーカラムのコピーが含まれます。主キーカラムの値を変更することは負荷の高い操作です。したがって、InnoDB テーブル設計の重要な側面は、最も重要なクエリーで使用されるカラムを持つ主キーを選択し、値をほとんど変更せずに主キーを短く保つことです。

[バックアップ](#)、[クラスタ化されたインデックス](#)、[コンパクト行フォーマット](#)、[圧縮行フォーマット](#)、[圧縮](#)、[動的行フォーマット](#)、[高速インデックス作成](#)、[file-per-table](#)、[.ibd ファイル](#)、[インデックス](#)、[オフページカラム](#)、[主キー](#)、[冗長行フォーマット](#)、[行](#)、[システムテーブルスペース](#)、[テーブルスペース](#)も参照

テーブルの完全スキャン

index を使用して選択した部分のみでなく、テーブルの内容全体を読み取る必要がある操作。通常は、小さなルックアップテーブル、またはすべての利用可能なデータが集約および分析される大きなテーブルを持つデータウェアハウジング状況で実行されます。これらの操作が発生する頻度、および使用可能なメモリーに対するテーブルのサイズは、クエリーの最適化およびバッファプールの管理で使用されるアルゴリズムに影響します。

インデックスの目的は、大きなテーブル内で特定の値または値の範囲をルックアップできるようにし、したがって有用なときはフルテーブルスキャンを回避することです。

[バッファプール](#)、[インデックス](#)も参照

テーブルスキャン

[テーブルの完全スキャン](#)も参照

テーブルスペース

InnoDB テーブルおよび関連するインデックスのデータを保持できるデータファイル。

システムテーブルスペースには InnoDB データディクショナリが含まれており、MySQL 5.6 より前は、デフォルトで他のすべての InnoDB テーブルが保持されます。

`innodb_file_per_table` オプションは、MySQL 5.6 以上でデフォルトで有効になっており、独自のテーブルスペースにテーブルを作成できます。File-per-table テーブルスペースは、オフページカラム、テーブル圧縮、トランスポータブルテーブルスペースの効率的なストレージなどの機能をサポートします。詳細は、[セクション 15.6.3.2 「File-Per-Table テーブルスペース」](#) を参照してください。

InnoDB では、MySQL 5.7.6 に一般テーブルスペースが導入されました。一般テーブルスペースは、`CREATE TABLESPACE` 構文を使用して作成される共有テーブルスペースです。これらは MySQL データディレクトリの外部で作成でき、複数のテーブルを保持でき、すべての行形式のテーブルをサポートします。

また、MySQL NDB Cluster はそのテーブルをテーブルスペースにグループ化します。詳細は、[セクション23.5.10.1「NDB Cluster ディスクデータオブジェクト」](#)を参照してください。
[圧縮行フォーマット](#)、[データディクショナリ](#)、[データファイル](#)、[file-per-table](#)、[一般テーブルスペース](#)、[インデックス](#)、[innodb_file_per_table](#)、[システムテーブルスペース](#)、[テーブル](#)も参照

テーブルタイプ

ストレージエンジンの古いシノニムです。InnoDB テーブル、MyISAM テーブルなどと呼ばれます。
[InnoDB](#)、[ストレージエンジン](#)も参照

テーブルロック

ほかのトランザクションがテーブルにアクセスすることを防ぐロック。InnoDB では、DML ステートメントおよびクエリーの処理にオンライン DDL、行ロック、読取り一貫性などの技術を使用することで、このようなロックを不要にするためにかなりの労力があります。SQL から [LOCK TABLE](#) ステートメントを使用してこのようなロックを作成できます。ほかのデータベースシステムまたは MySQL ストレージエンジンから移行するステップの 1 つは、可能なときはこのようなステートメントを削除することです。
[一貫性読み取り](#)、[DML](#)、[ロック](#)、[ロック](#)、[オンライン DDL](#)、[クエリー](#)、[行ロック](#)、[テーブル](#)、[トランザクション](#)も参照

テーブル統計

[統計](#)も参照

ディクショナリオブジェクトキャッシュ

ディクショナリオブジェクトキャッシュは、以前にアクセスしたデータディクショナリオブジェクトをメモリーに格納して、オブジェクトの再利用を可能にし、ディスク I/O を最小化します。LRU ベースのエビクション戦略を使用して、メモリーから最近使用されていないオブジェクトを除去します。キャッシュは、様々なオブジェクトタイプを格納する複数のパーティションで構成されます。

詳細は、[セクション14.4「ディクショナリオブジェクトキャッシュ」](#)を参照してください。
[データディクショナリ](#)、[LRU](#)も参照

ディスクバウンド

主なボトルネックがディスク I/O であるタイプのワークロード。(I/O バウンドとも呼ばれます。)通常は、ディスクへの頻繁な書き込みや、バッファプールに収められるよりも多くのデータのランダム読み取りがかわります。
[ボトルネック](#)、[バッファプール](#)、[ワークロード](#)も参照

ディスクベース

主にディスクストレージ (ハードドライブまたはその同等物) 上のデータを編成するタイプのデータベース。データは、ディスクとメモリー間でやり取りされて操作されます。インメモリーデータベースの反対です。InnoDB はディスクベースですが、バッファプール、複数のバッファプールインスタンス、および特定の種類のワークロードが主にメモリーから機能できる適応型ハッシュインデックスなどの機能も含まれています。
[適応型ハッシュインデックス](#)、[バッファプール](#)、[インメモリーデータベース](#)も参照

デッドロック

さまざまなトランザクションが進行できない状況 (それぞれが、他方が必要とするロックを保持しているため)。どちらのトランザクションもリソースが使用可能になるのを待機しているため、どちらも保持しているロックを解放しません。

([UPDATE](#) や [SELECT ... FOR UPDATE](#) などのステートメントを通じて) トランザクションが複数のテーブル内の行を反対の順にロックすると、デッドロックが発生することがあります。デッドロックは、このようなステートメントがインデックスレコードとギャップの範囲をロックし、各トランザクションが一部のロックを取得するけれども、タイミングの問題によりほかを取得しない場合にも発生することがあります。

自動的にデッドロックを検出して処理する方法に関する背景情報については、[セクション15.7.5.2「デッドロック検出」](#)を参照してください。デッドロック状況を回避しリカバリするためのヒントについては、[セクション15.7.5.3「デッドロックを最小化および処理する方法」](#)を参照してください。
[ギャップ](#)、[ロック](#)、[トランザクション](#)も参照

デッドロック対象

デッドロックが検出されたときにロールバック済として自動的に選択されるトランザクション。InnoDB は、更新された行が最も少ないトランザクションをロールバックします。

デッドロック検出は、[innodb_deadlock_detect](#) 構成オプションを使用して無効にできます。
[デッドロック](#)、[デッドロック検出](#)、[innodb_lock_wait_timeout](#)、[トランザクション](#)も参照

デッドロック検出

デッドロックが起きていることを自動的に検出し、関係するトランザクションのいずれか (デッドロック対象) を自動的にロールバックするメカニズム。デッドロック検出は、`innodb_deadlock_detect` 構成オプションを使用して無効にできません。

[デッドロック](#)、[ロールバック](#)、[トランザクション](#)、[デッドロック対象](#)も参照

データウェアハウス

主に大きなクエリーを実行するデータベースシステムまたはアプリケーション。読み取り専用または読み取りが大半のデータは、クエリーの効率を高めるために非正規化された形式で編成できます。MySQL 5.6 以降では、読み取り専用トランザクションの最適化からメリットを得ることができます。OLTP と対比してください。

[非正規化](#)、[OLTP](#)、[クエリー](#)、[読み取り専用トランザクション](#)も参照

データディクショナリ

テーブル、インデックス、テーブル columns などのデータベースオブジェクトを追跡するメタデータ。MySQL 8.0 で導入された MySQL データディクショナリの場合、メタデータは `mysql` データベースディレクトリの `InnoDB file-per-table` テーブルスペースファイルに物理的に配置されます。`InnoDB` データディクショナリの場合、メタデータは `InnoDB` システムテーブルスペースに物理的に配置されます。

MySQL Enterprise Backup 製品では常に `InnoDB` システムテーブルスペースがバックアップされるため、すべてのバックアップに `InnoDB` データディクショナリの内容が含まれます。

[カラム](#)、[file-per-table](#)、[.frm ファイル](#)、[インデックス](#)、[MySQL Enterprise Backup](#)、[システムテーブルスペース](#)、[テーブル](#)も参照

データディレクトリ

各 MySQL instance が `InnoDB` 用のデータファイルおよび個々のデータベースを表すディレクトリを保持するディレクトリ。`datadir` 構成オプションによって制御されます。

[データファイル](#)、[インスタンス](#)も参照

データファイル

table および index データを物理的に格納するファイル。

`InnoDB` データディクショナリを保持し、複数の `InnoDB` テーブルのデータを保持できる `InnoDB` システムテーブルスペースは、1 つ以上の `.ibdata` データファイルで表されます。

単一の `InnoDB` テーブルのデータを保持する File-per-table テーブルスペースは、`.ibd` データファイルで表されます。

複数の `InnoDB` テーブルのデータを保持できる一般的なテーブルスペース (MySQL 5.7.6 で導入) は、`.ibd` データファイルでも表されます。

[データディクショナリ](#)、[file-per-table](#)、[一般テーブルスペース](#)、[.ibd ファイル](#)、[ibdata ファイル](#)、[インデックス](#)、[システムテーブルスペース](#)、[テーブル](#)、[テーブルスペース](#)も参照

データベース

MySQL データディレクトリ内では、各データベースは個別のディレクトリで表されます。MySQL instance 内の複数のデータベースからのテーブルデータを保持できる `InnoDB` システムテーブルスペースは、個々のデータベースディレクトリの外部にあるデータファイルに保持されます。file-per-table モードが有効な場合、`DATA DIRECTORY` 句を使用して別の場所に作成されないかぎり、個々の `InnoDB` テーブルを表す `.ibd` ファイルはデータベースディレクトリ内に格納されます。MySQL 5.7.6 で導入された一般的なテーブルスペースには、`.ibd` ファイルのテーブルデータも保持されます。file-per-table `.ibd` ファイルとは異なり、一般的なテーブルスペース `.ibd` ファイルは、MySQL instance 内の複数のデータベースからのテーブルデータを保持でき、MySQL データディレクトリに対して相対的または独立したディレクトリに割り当てることができます。

長年 MySQL を使用している人にとって、データベースはなじみ深い概念です。Oracle Database バックグラウンドからのユーザーは、データベースの MySQL の意味が、Oracle Database が `schema` をコールする内容に近いことがわかります。

[データファイル](#)、[file-per-table](#)、[.ibd ファイル](#)、[インスタンス](#)、[スキーマ](#)、[システムテーブルスペース](#)も参照

データ定義言語

[DDL](#)も参照

データ操作言語

[DML](#)も参照

低位境界値

下限を表す値。通常は、何らかの訂正アクションが始まったり、より積極的になったりするしきい値です。高位境界値と対比してください。

高位境界値も参照

低速シャットダウン

完了前に追加の InnoDB フラッシュ操作を実行する shutdown のタイプ。クリーンシャットダウンとも呼ばれます。構成パラメータ `innodb_fast_shutdown=0` またはコマンド `SET GLOBAL innodb_fast_shutdown=0;` で指定されます。シャットダウン自体には時間がかかる場合がありますが、その後の起動時にその時間を節約する必要があります。

[クリーンシャットダウン](#)、[高速シャットダウン](#)、[シャットダウン](#)も参照

転置インデックス

InnoDB 全文検索の実装で使用される、ドキュメント取得システム用に最適化されたデータ構造。逆インデックスとして実装された InnoDB FULLTEXT インデックスは、テーブルの行の場所ではなく、ドキュメント内の各ワードの位置を記録します。単一カラム値 (テキスト文字列として格納されたドキュメント) は多くのエントリで転置インデックスで表現されます。

[全文検索](#)、[FULLTEXT インデックス](#)、[ilist](#)も参照

適応型ハッシュインデックス

メモリ内にハッシュインデックスを構築することで、`=` および `IN` 演算子を使用して検索を高速化できる InnoDB テーブルの最適化。MySQL は、InnoDB テーブルのインデックス検索を監視し、クエリーがハッシュインデックスのメリットを得られる場合は、頻繁にアクセスされるインデックス pages のインデックス検索を自動的に作成します。ある意味では、適応型ハッシュインデックスは、十分なメインメモリーを利用するように MySQL を実行時に構成するので、メインメモリーデータベースのアーキテクチャーに近づいています。この機能は、`innodb_adaptive_hash_index` 構成オプションで制御されます。この機能は、一部のワークロードにはメリットがあってもほかのものにはメリットがなく、ハッシュインデックスに使用されるメモリーはバッファプールで予約されているので、通常はこの機能を有効にした状態と無効にした状態でベンチマークを行うことをお勧めします。

常に、ハッシュインデックスはテーブル上の既存の B ツリーインデックスに基づいて構築されます。MySQL は、インデックスに対する検索パターンに応じて、B ツリーに定義された任意の長さのキーのプリフィクスに、ハッシュインデックスを構築できます。ハッシュインデックスは部分的であってもかまいません。B ツリーインデックス全体をバッファプールにキャッシュする必要はありません。

MySQL 5.6 以上では、InnoDB テーブルで高速単一値参照を利用する別の方法は、InnoDB memcached プラグインを使用することです。詳細は、[セクション15.20「InnoDB memcached プラグイン」](#)を参照してください。

[B ツリー](#)、[バッファプール](#)、[ハッシュインデックス](#)、[memcached](#)、[ページ](#)、[セカンダリインデックス](#)も参照

適応型フラッシュ

チェックポイントによって生じる I/O オーバーヘッドを軽減する InnoDB テーブル用のアルゴリズム。MySQL は、変更されたすべてのページをバッファプールからデータファイルに一度にフラッシュするのではなく、変更されたページの小さなセットを定期的にフラッシュします。適応型フラッシュアルゴリズムは、フラッシュの頻度と Redo 情報の生成速度に基づいてこれらの定期フラッシュの最適な実行頻度を見積もることによって、このプロセスを拡張します。

[バッファプール](#)、[チェックポイント](#)、[データファイル](#)、[フラッシュ](#)、[InnoDB](#)、[ページ](#)、[Redo ログ](#)も参照

適用

MySQL Enterprise Backup 製品で生成されたバックアップに、バックアップ進行中に行われた最新の変更が含まれない場合、これらの変更を含むようにバックアップファイルを更新するプロセスは適用ステップと呼ばれます。これは `mysqlbackup` コマンドの `apply-log` オプションで指定されます。

変更が適用されるまでは、このファイルは raw バックアップと呼ばれます。変更が適用されたあとは、このファイルは準備されたバックアップと呼ばれます。変更は、`ibbackup_logfile` ファイルに記録されます。適用ステップが終了すると、このファイルは不要になります。

[ホットバックアップ](#)、[ibbackup_logfile](#)、[MySQL Enterprise Backup](#)、[準備されたバックアップ](#)、[raw バックアップ](#)も参照

ト

トラストストア

[SSL](#)も参照

トラブルシューティング

問題の原因を特定するプロセス。MySQL の問題をトラブルシューティングするためのリソースには、次のものがあります:

- [セクション2.10.2.1「MySQL Server の起動時の問題のトラブルシューティング」](#)
- [セクション6.2.21「MySQL への接続の問題のトラブルシューティング」](#)

- [セクションB.3.3.2「root のパスワードをリセットする方法」](#)
- [セクションB.3.2「MySQL プログラム使用時の一般的なエラー」](#)
- [セクション15.21「InnoDB のトラブルシューティング」](#)。

トランザクション

トランザクションは、committed またはロールバック済の作業の原子単位です。トランザクションによってデータベースに複数の変更が行われた場合、トランザクションがコミットされるとすべての変更が完了し、トランザクションがロールバックされるとすべての変更が元に戻されます。

InnoDB によって実装されるデータベーストランザクションには、アトミック性、一貫性、分離性および永続性のために頭字語 ACID によってまとめて知られるプロパティがあります。
[ACID](#), [コミット](#), [分離レベル](#), [ロック](#), [ロールバック](#)も参照

トランザクション ID

各 row に関連付けられた内部フィールド。このフィールドは、行をロックしたトランザクションを記録するために、[INSERT](#)、[UPDATE](#) および [DELETE](#) 操作によって物理的に変更されます。
[暗黙の行ロック](#), [行](#), [トランザクション](#)も参照

トランスポートテーブルスペース

テーブルスペースがあるインスタンスから別のインスタンスに移動されることを許可する機能。従来、すべてのテーブルデータがシステムテーブルスペースの一部であったため、これは InnoDB テーブルスペースでは不可能でした。MySQL 5.6 以上では、[FLUSH TABLES ... FOR EXPORT](#) 構文は InnoDB テーブルを別のサーバーにコピーする準備をします。他のサーバーで [ALTER TABLE ... DISCARD TABLESPACE](#) および [ALTER TABLE ... IMPORT TABLESPACE](#) を実行すると、コピーされたデータファイルが他のインスタンスに取り込まれます。[.ibd](#) ファイルとともにコピーされた個別の [.cfg](#) ファイルを使用して、テーブルスペースのインポート時にテーブルメタデータ (スペース ID など) が更新されます。使用に関する情報は [セクション15.6.1.3「InnoDB テーブルのインポート」](#) を参照してください。
[.cfg ファイル](#), [.ibd ファイル](#), [スペース ID](#), [システムテーブルスペース](#), [テーブルスペース](#)も参照

ドキュメント ID

InnoDB 全文検索機能において、各 `ilist` 値に関連付けられたドキュメントを一意に識別するための、FULLTEXT インデックスを含むテーブルの特別なカラム。その名前は `FTS_DOC_ID` (大文字必須) です。カラム自体は、`BIGINT UNSIGNED NOT NULL` 型で、`FTS_DOC_ID_INDEX` という名前の一意インデックス付きである必要があります。テーブルの作成時にこのカラムを定義することが推奨されます。FULLTEXT インデックスの作成時に InnoDB でテーブルにカラムを追加する必要がある場合、インデックス付け操作のコストはかなり高くなります。
[全文検索](#), [FULLTEXT インデックス](#), [ilist](#)も参照

ドロップ

DDL 操作の一種。[DROP TABLE](#) や [DROP INDEX](#) などのステートメントを通じてスキーマオブジェクトを削除します。これは内部的に [ALTER TABLE](#) ステートメントにマッピングします。InnoDB の観点からは、このような操作のパフォーマンス上の考慮事項には、相互に関連するオブジェクトがすべて更新されるようにデータディクショナリがロックされている時間、およびバッファプールなどのメモリ構造を更新する時間が含まれます。テーブルの場合、ドロップ操作には、切り捨て操作 ([TRUNCATE TABLE](#) ステートメント) と多少異なる特性があります。
[バッファプール](#), [データディクショナリ](#), [DDL](#), [テーブル](#), [切り捨て](#)も参照

動的 SQL

ステートメントの各部分を文字列変数に連結するナイブテクニックよりも強力でセキュアで効率的な方法を使用してプリペアドステートメントを作成および実行できる機能。
[プリペアドステートメント](#)も参照

動的カーソル

ODBC でサポートされている cursor のタイプで、行が再度読み取られたときに新しい結果および変更された結果を取得できます。カーソルに変更が表示されるかどうかとその速度は、関連するテーブルのタイプ (トランザクションまたは非トランザクション) およびトランザクションテーブルの分離レベルによって異なります。動的カーソルのサポートは明示的に有効にする必要があります。
[カーソル](#), [ODBC](#)も参照

動的ステートメント

動的 SQL を介して作成および実行されるプリペアドステートメント。
[動的 SQL](#), [プリペアドステートメント](#)も参照

動的行フォーマット

InnoDB の行フォーマット。長い可変長のカラム値は、行データを保持するページの外部に格納されるため、ラージオブジェクトを含む行では非常に効率的です。ラージフィールドは通常、クエリー条件を評価するためにアクセスされることはないので、頻繁にはバッファプールに読み込まれません。その結果、I/O 操作は少なくなり、キャッシュメモリーの利用率が改善します。

MySQL 5.7.9 では、デフォルトの行フォーマットは、DYNAMIC のデフォルト値を持つ `innodb_default_row_format` によって定義されます。

InnoDB DYNAMIC 行フォーマットの追加情報については、[DYNAMIC 行フォーマット](#) を参照してください。[バッファプール](#)、[ファイル形式](#)、[行フォーマット](#) も参照

統計

各 InnoDB テーブルおよびインデックスに関連する評価値。効率的なクエリー実行計画の構築に使用されます。メイン値は、カーディナリティー (個別値の数) と、テーブル行またはインデックスエントリの合計数です。テーブルの統計は、その主キーインデックス内のデータを表します。セカンダリインデックスの統計は、このインデックスで扱われる行を表します。

値は正確なカウントではなく、見積もりです。あらゆる瞬間にさまざまなトランザクションが同じテーブルからの行を挿入したり削除したりしている可能性があるためです。値が頻繁に再計算されないように、永続的統計を有効にできます。この場合、値は InnoDB システムテーブルに格納され、[ANALYZE TABLE](#) ステートメントを発行するときのみ更新されます。

`innodb_stats_method` 構成オプションで統計を計算するときに、NULL 値がどのように扱われるかを制御できます。

INFORMATION_SCHEMA および PERFORMANCE_SCHEMA テーブルを通じて、ほかのタイプの統計をデータベースオブジェクトおよびデータベースアクティビティーに利用できます。

[カーディナリティー](#)、[インデックス](#)、[INFORMATION_SCHEMA](#)、[NULL](#)、[パフォーマンススキーマ](#)、[永続的統計](#)、[主キー](#)、[クエリー実行計画](#)、[セカンダリインデックス](#)、[テーブル](#)、[トランザクション](#) も参照

透過的ページ圧縮

file-per-table テーブルスペースに存在する InnoDB テーブルのページレベルの圧縮を可能にする MySQL 5.7.8 で追加された機能。ページ圧縮を有効にするには、[CREATE TABLE](#) または [ALTER TABLE](#) で `COMPRESSION` 属性を指定します。詳細は、[セクション 15.9.2 「InnoDB ページ圧縮」](#) を参照してください。

[file-per-table](#)、[ホールパンチング](#)、[スパースファイル](#) も参照

ナ

ナチュラルキー

インデックス付けされたカラム (通常は主キー)。値には実際の意味があります。通常は、次の理由のため推奨されていません。

- 値が万が一変化した場合、クラスタ化されたインデックスを再ソートし、それぞれのセカンダリインデックスで繰り返される主キー値のコピーを更新するために、多数インデックス保守が必要になる可能性があります。
- 一見したところ安定した値でも、データベースで正しく表すことが難しい予測不可能な変化をすることがあります。たとえば、1 つの国が 2 つ以上に分かれ、元の国コードが古くなることがあります。または、一意値に関するルールに例外が発生する場合があります。たとえば、納税者 ID が単一の人物に一意であるように意図されている場合でも、データベースでは、ID 窃盗などでそのルールに違反するレコードを処理する必要があることがあります。また、納税者 ID やその他の機密 ID 番号からは、低品質の主キーが作成されます。それらはセキュリティー保護し、暗号化し、またはほかのカラムと異なる方法で扱う必要がある場合があるためです。

したがって通常は、自動インクリメントカラムを使用するなど、任意の数値を使用して合成キーを作成することをお勧めします。

[自動インクリメント](#)、[クラスタ化されたインデックス](#)、[主キー](#)、[セカンダリインデックス](#)、[合成キー](#) も参照

ニ

二重書き込みバッファ

InnoDB では、二重書き込みと呼ばれるファイルフラッシュ手法を使用します。pages をデータファイルに書き込む前に、InnoDB はまず二重書き込みバッファと呼ばれる記憶域に書き込みます。二重書き込みバッファへの書き込みおよびフラッシュが完了した後のみ、InnoDB はデータファイル内の適切な位置にページを書き込みます。ページ書き込みの途中でオペレー

ティングシステム、ストレージサブシステムまたは `mysqld` プロセスがクラッシュした場合、`InnoDB` は後でクラッシュリカバリ中に二重書き込みバッファからページの適切なコピーを見つけることができます。

データは常に2度書き込まれますが、二重書き込みバッファには、2倍のI/Oオーバーヘッドも2倍のI/O操作も不要です。データは、オペレーティングシステムへの単一 `fsync()` 呼び出しで、大きなシーケンシャルチャックとしてバッファ自体に書き込まれます。

二重書き込みバッファをオフにするには、オプション `innodb_doublewrite=0` を指定します。
[クラッシュリカバリ](#), [データファイル](#), [ページ](#), [ページ](#)も参照

ネ

ネイティブ C API
`libmysqlclient` と同義です。
[libmysql](#)も参照

ネクストキーロック
インデックスレコードでのレコードロックと、インデックスレコードの前のギャップでのギャップロックの組み合わせ。
[ギャップロック](#), [ロック](#), [レコードロック](#)も参照

ハ

ハッシュインデックス
大なりや `BETWEEN` などの範囲演算子ではなく、等値演算子を使用するクエリーを対象とするタイプのインデックス。これは、`MEMORY` テーブルで使用できます。履歴上の理由から、ハッシュインデックスは `MEMORY` テーブルのデフォルトですが、そのストレージエンジンは B-tree インデックスもサポートしています。これは、一般的な目的のクエリーに適していることがよくあります。

MySQL には、ランタイム条件に基づいて必要に応じて `InnoDB` テーブル用に自動的に作成される、このインデックスタイプのバリエーション (適応型ハッシュインデックス) が含まれています。
[適応型ハッシュインデックス](#), [B ツリー](#), [インデックス](#), [InnoDB](#)も参照

ハートビート
システムが適切に機能していることを示すために送信される定期的メッセージ。レプリケーションコンテキストでは、`source` がこのようなメッセージの送信を停止すると、レプリカのいずれかが発生します。クラスタ環境内のすべてのサーバーが正しく動作していることを確認するために、それらの間で類似の方法を使用できます。
[レプリケーション](#), [source](#)も参照

バイナリログ
テーブルデータを変更しようとするすべてのステートメントまたは行の変更のレコードを含むファイル。バイナリログの内容をリプレイして、レプリケーションシナリオでレプリカを最新の状態にしたり、バックアップからテーブルデータを復元したあとでデータベースを最新の状態にしたりできます。バイナリロギング機能は、オンとオフを切り替えられますが、レプリケーションを使用したりバックアップを実行したりする場合は、常に有効にしておくことをお勧めします。

`mysqlbinlog` コマンドを使用すると、バイナリログの内容を検査したり、レプリケーションまたは回復中にそれを再生したりできます。バイナリログの詳細は、[セクション5.4.4「バイナリログ」](#)を参照してください。バイナリログに関連したMySQL 構成オプションについては、[セクション17.1.6.4「バイナリロギングのオプションと変数」](#)を参照してください。

MySQL Enterprise Backup 製品の場合、バイナリログのファイル名とファイル内での現在の位置が重要な詳細です。レプリケーションコンテキストでバックアップを作成するときにソースのこの情報を記録するには、`--slave-info` オプションを指定できます。

MySQL 5.0 より前では、更新ログと呼ばれる同様の機能を利用できました。MySQL 5.0 以上では、更新ログがバイナリログに置き換わりました。
[binlog](#), [MySQL Enterprise Backup](#), [レプリケーション](#)も参照

バウンス
直後に再起動が行われるシャットダウン操作。パフォーマンスとスループットが迅速に高いレベルに戻るように、ウォームアップ期間が比較的短ければ理想的です。
[シャットダウン](#)も参照

バックアップ

保護するために、MySQL インスタンスから一部またはすべてのテーブルデータおよびメタデータをコピーするプロセス。コピーされたファイルのセットを指す場合もあります。これは DBA のきわめて重要なタスクです。このプロセスの反対がリストア操作です。

MySQL では、物理バックアップは MySQL Enterprise Backup 製品で実行され、論理バックアップは `mysqldump` コマンドで実行されます。これらの方法は、バックアップデータのサイズおよび表現と速度 (特にリストア操作の速度) の点で特性が異なります。

バックアップはさらに、通常のデータベース操作に干渉する程度に応じて、ホット、ウォーム、コールドに分類されます。(干渉の程度は、ホットバックアップがもっとも少なく、コールドバックアップがもっとも多くなります。)

[コールドバックアップ](#)、[ホットバックアップ](#)、[論理バックアップ](#)、[MySQL Enterprise Backup](#)、[mysqldump](#)、[物理バックアップ](#)、[ウォームバックアップ](#)も参照

バッファ

一時ストレージに使用されるメモリーまたはディスク領域。多数の小さな I/O 操作ではなく少数の大きなものでディスクに効率的に書き込めるように、データはメモリーにバッファリングされます。データは、信頼性を高めるためにディスクにバッファリングされるので、考えられる最悪の場合にクラッシュやほかの障害が発生してもリカバリできます。InnoDB で使用される主なバッファタイプは、バッファプール、二重書き込みバッファおよび変更バッファです。

[バッファプール](#)、[変更バッファ](#)、[クラッシュ](#)、[二重書き込みバッファ](#)も参照

バッファプール

テーブルとインデックスの両方のキャッシュされた InnoDB データを保持するメモリー領域。大容量読み取り操作の効率を高めるため、バッファプールは複数行を保持できるページに分割されます。キャッシュ管理の効率のために、バッファプールはページのリンクリストとして実装されます。まれにしか使用されないデータは、LRU アルゴリズムのバリエーションを使用してキャッシュからエージアウトされます。大容量メモリーを備えたシステムでは、バッファプールを複数のバッファプールインスタンスに分割することにより、並列性を改善できます。

いくつかの InnoDB ステータス変数、[INFORMATION_SCHEMA](#) テーブルおよび [performance_schema](#) テーブルは、バッファプールの内部動作の監視に役立ちます。MySQL 5.6 以降では、サーバーの停止時にバッファプールの状態を保存し、サーバーの起動時にバッファプールを同じ状態にリストアすることで、サーバーの再起動後、特に大規模なバッファプールを持つインスタンスで長いウォームアップ期間を回避できます。[セクション 15.8.3.6 「バッファプールの状態の保存と復元」](#)を参照してください。

[バッファプールインスタンス](#)、[LRU](#)、[ページ](#)、[ウォームアップ](#)も参照

バッファプールインスタンス

バッファプールは複数の領域に分割できますが、そのいずれか。[innodb_buffer_pool_instances](#) 構成オプションで制御されます。[innodb_buffer_pool_size](#) で指定された合計メモリーサイズは、すべてのバッファプールインスタンスに分割されます。通常、複数のバッファプールインスタンスを持つことは、InnoDB バッファプールに複数の G バイトを割り当てるシステムに適しています。各インスタンスは 1 G バイト以上です。多数の同時セッションからバッファプール内の大量のデータをロードまたは検索するシステムでは、複数のバッファプールインスタンスを使用すると、バッファプールを管理するデータ構造への排他的アクセスの競合が軽減されます。

[バッファプール](#)も参照

バディアーロケータ

InnoDB バッファプールでさまざまなサイズのページを管理するためのメカニズム。

[バッファプール](#)、[ページ](#)、[page size](#)も参照

パフォーマンススキーマ

[performance_schema](#) スキーマは (MySQL 5.5 以降)、MySQL Server の多くの内部パーツのパフォーマンス特性に関する詳細情報を取得するために照会できる、テーブルセットを提供します。[第 27 章 「MySQL パフォーマンススキーマ」](#)を参照してください。

[INFORMATION_SCHEMA](#)、[ラッチ](#)、[相互排他ロック](#)、[rw ロック \(読み書きロック\)](#)も参照

ページ

([innodb_purge_threads](#) によって制御される) 個別のバックグラウンドスレッドによって実行されるガベージコレクションのタイプで、定期的に実行されます。ページでは、undo ログページを履歴リストから解析して処理します。これは、(以前の [DELETE](#) ステートメントによって) 削除対象としてマークされ、MVCC または rollback では不要になったクラスティンデックスレコードおよびセカンダリインデックスレコードを削除するためです。ページすると、undo ログページは処理後に履歴リストから解放されます。

[履歴リスト](#)、[MVCC](#)、[ロールバック](#)、[Undo ログ](#)も参照

ページスレッド

定期的な purge 操作の実行専用の InnoDB プロセス内のスレッド。MySQL 5.6 以降では、複数のページスレッドが `innodb_purge_threads` 構成オプションによって有効になっています。
[ページ](#), [スレッド](#) も参照

ページバッファリング

ランダムな I/O を最小限に抑えるために物理書込みを実行できるように、変更を即座に書き込むのではなく、`DELETE` 操作によって生成されたセカンダリインデックスページへの変更を変更バッファに格納する方法。(削除操作は 2 ステッププロセスなので、この操作は、通常は以前に削除とマークされたインデックスレコードをページする書き込みをバッファリングします。)これは変更バッファリングの一種です。ほかには挿入バッファリングと削除バッファリングがあります。
[変更バッファ](#), [変更バッファリング](#), [削除バッファリング](#), [挿入バッファ](#), [挿入バッファリング](#) も参照

ページ遅延

InnoDB 履歴リストの別の名前。 `innodb_max_purge_lag` 構成オプションに関連しています。
[履歴リスト](#), [ページ](#) も参照

半一貫性読み取り

READ COMMITTED と読み取り一貫性の組合せである、`UPDATE` ステートメントに使用される読み取り操作のタイプ。`UPDATE` ステートメントがすでにロックされている行を調べると、InnoDB は、MySQL に最新のコミット済バージョンを返して、その行が `UPDATE` の `WHERE` 条件に一致するかどうかを判断できるようにします。行が一致する場合 (更新する必要がある場合)、MySQL は行を再度読み取り、今回は InnoDB がロックするか、ロックを待機します。このタイプの読み取り操作は、トランザクションに READ COMMITTED 分離レベルがある場合にのみ実行できます。
[一貫性読み取り](#), [分離レベル](#), [READ COMMITTED](#) も参照

反復不可能読み取り

あるクエリーがデータを取得し、同じトランザクション内のその後のクエリーが同じデータであるはずのものを取得するけれども、それらのクエリーが異なる結果を返す状況 (その間にコミットしている別のトランザクションによって変更された)。

この種の操作は、データベース設計の ACID 原則に反します。トランザクション内のデータは、予測可能で安定した関係を持ち、一貫しているべきです。

さまざまな分離レベルの中で、反復不可能読み取りは、シリアライズ可能読み取りと反復可能読み取りレベルによって防止され、一貫性読み取りとコミットされていない読み取りレベルで許可されます。

[ACID](#), [一貫性読み取り](#), [分離レベル](#), [READ UNCOMMITTED](#), [REPEATABLE READ](#), [SERIALIZABLE](#), [トランザクション](#) も参照

排他ロック

ほかのトランザクションが同じ行をロックするのを回避するタイプのロック。この種のロックは、トランザクション分離レベルに応じて、ほかのトランザクションが同じ行に書き込むのをブロックしたり、ほかのトランザクションが同じ行を読み取るのをブロックしたりできます。デフォルトの InnoDB 分離レベルである REPEATABLE READ を使用すると、読み取り一貫性と呼ばれる手法である排他ロックを持つ行をトランザクションが読み取ることができるようになるため、より高い同時実行性が有効になります。

[並列性](#), [一貫性読み取り](#), [分離レベル](#), [ロック](#), [REPEATABLE READ](#), [共有ロック](#), [トランザクション](#) も参照

破損ページ

I/O デバイス構成とハードウェア障害の組み合わせが原因で発生する可能性のあるエラー状況。データが InnoDB ページサイズより小さいチャンク (デフォルトでは 16KB) で書き出された場合、書き込み中にハードウェア障害が発生すると、ページの一部のみがディスクに格納される可能性があります。InnoDB 二重書き込みバッファは、この可能性を防ぎます。

[二重書き込みバッファ](#) も参照

ヒ

ビジネスルール

営利企業を運営するために使用される、ビジネスソフトウェアの基盤を形作るアクションの関係およびシーケンス。これらのルールは、法律によって規定されたり、企業ポリシーで規定されたりします。慎重に計画することで、データベースでエンコードされ適用される関係と、アプリケーションロジックを通じて実行されるアクションが、企業の実際のポリシーを正確に反映し、現実の状況を扱うことができます。

たとえば、従業員が会社を退職すると、人事部からアクションシーケンスがトリガーされます。人事データベースには、雇用されたけれどもまだ就業していない人物に関するデータを表すために、柔軟性も必要になることがあります。オンラインサービスで口座を閉鎖すると、データがデータベースから削除されたり、口座が再度開設されたりした場合にリカバリできるようにデータが移動またはフラグ付けされたりします。企業は、給与が負数でないなどの基本的なサニティーチェックに加えて、給与の最大、最小、および調整に関するポリシーを確立できます。小売データベースでは、同じシリアル番号の購入を

複数回返すことを禁止したり、一定値を超えるクレジットカード購入を禁止したりしますが、詐欺の検出に使用されるデータベースでは、このようなことを許可する場合があります。

[リレーショナル](#)も参照

非ブロック化 I/O

非同期 I/O と同じ意味を持つ業界用語。

[非同期 I/O](#) も参照

非ロック読み取り

[SELECT ... FOR UPDATE](#) または [SELECT ... LOCK IN SHARE MODE](#) 句を使用しないクエリー。読み取り専用トランザクションでグローバルテーブルに許可される唯一の種類のクエリー。ロック読み取りの反対。[セクション15.7.2.3「一貫性非ロック読み取り」](#)を参照してください。

MySQL 8.0.1 の [SELECT ... LOCK IN SHARE MODE](#) は [SELECT ... FOR SHARE](#) に置き換わりませんが、[LOCK IN SHARE MODE](#) は下位互換性のために引き続き使用できます。

[ロック読み取り](#)、[クエリー](#)、[読み取り専用トランザクション](#)も参照

非同期 I/O

I/O が完了するまでほかの処理の進行を許可する I/O 操作のタイプ。非ブロック化 I/O と呼ばれ、AIO と省略されます。[InnoDB](#) では、実際にはリクエストされていないが、すぐに必要になる可能性があるバッファプールへのページの読み取りなど、データベースの信頼性に影響を与えることなくパラレルで実行できる特定の操作に、このタイプの I/O が使用されます。

従来、[InnoDB](#) は Windows システムでのみ非同期 I/O を使用していました。[InnoDB](#) プラグイン 1.1 および MySQL 5.5 以上では、[InnoDB](#) は Linux システムで非同期 I/O を使用します。この変更により、[libaio](#) への依存関係がもたらされます。Linux システムでの非同期 I/O は、[innodb_use_native_aio](#) オプションを使用して構成されます。これはデフォルトで有効になっています。ほかの Unix 系システムでは、[InnoDB](#) は同期 I/O だけを使用します。

[バッファプール](#)、[非ブロック化 I/O](#) も参照

非正規化

外部キーと結合クエリーでテーブルをリンクするのではなく、複数のテーブルにわたってデータを複製するデータストレージ戦略。通常はデータウェアハウスアプリケーションで使用されます。この場合、データはロード後に更新されません。このようなアプリケーションでは、更新中にデータの`一貫性`を維持することを簡略化するよりも、クエリーパフォーマンスが重要になります。正規化と対比してください。

[データウェアハウス](#)、[外部キー](#)、[結合](#)、[正規化](#)も参照

フ

ファイル形式

[InnoDB](#) テーブルのファイル形式。

[file-per-table](#)、[.ibd ファイル](#)、[ibdata ファイル](#)、[行フォーマット](#)も参照

ファジーチェックポイント

データベース処理を妨害するダーティーページを一度にすべてフラッシュするのではなく、ダーティーページの小さなバッチをバッファプールからフラッシュする方法。

[バッファプール](#)、[ダーティーページ](#)、[フラッシュ](#)も参照

ファントム

あるクエリーの結果セットに出現するけれども、以前のクエリーの結果セットにない行。たとえば、あるクエリーがトランザクション内で2度実行されて、その間に別のトランザクションがそのクエリーの `WHERE` 句に一致する新しい行を挿入または行を更新したあとにコミットされた場合です。

この現象がファントム読み取りと呼ばれます。このことから保護することは、反復不可能読み取りよりも困難です。最初のクエリー結果セットからのすべての行をロックしても、ファントムが出現する変更は防止されないためです。

さまざまな分離レベルの中で、ファントム読み取りは、シリアライズ可能読み取りで防止され、反復可能読み取り、一貫性読み取り、およびコミットされていない読み取りレベルで許可されます。

[一貫性読み取り](#)、[分離レベル](#)、[反復不可能読み取り](#)、[READ UNCOMMITTED](#)、[REPEATABLE READ](#)、[SERIALIZABLE](#)、[トランザクション](#)も参照

フィルファクタ

[InnoDB index](#) で、ページが分割される前にインデックスデータによって取得される page の割合。ページ間でインデックスデータが最初に分割される時の未使用領域によって、負荷のかかるインデックス保守操作を必要とすることなく、より長い

文字列値で行を更新できます。フィルファクタが低すぎた場合、インデックスは必要以上の領域を消費し、インデックスを読み取るときに余分な I/O オーバーヘッドが生じます。フィルファクタが高すぎると、カラム値の長さが増える更新で、インデックス保守の追加 I/O オーバーヘッドが生じる可能性があります。詳細は、[セクション15.6.2.2「InnoDB インデックスの物理構造」](#)を参照してください。

[インデックス](#)、[ページ](#)も参照

フラッシュ

メモリ領域または一時ディスクストレージ領域にバッファリングされていた変更をデータベースファイルに書き込むこと。定期的にフラッシュされる InnoDB 記憶域構造には、redo ログ、undo ログおよびバッファプールが含まれます。

フラッシュは、メモリ領域がいっぱいになってシステムが一部の領域を解放する必要があるため、コミット操作が、トランザクションからの変更を完了できることを意味するため、または低速シャットダウン操作が、すべての未処理作業を完了するべきであることを意味するため、行われます。バッファリングされているデータすべてを一度にフラッシュすることが重要でないときは、InnoDB は、ファジーチェックポイントという方法を使用して、ページの小さなバッチをフラッシュし、I/O オーバーヘッドを分散させることができます。

[バッファプール](#)、[コミット](#)、[ファジーチェックポイント](#)、[Redo ログ](#)、[低速シャットダウン](#)、[Undo ログ](#)も参照

フラッシュリスト

バッファプールでダーティページを追跡する内部 InnoDB データ構造: つまり、変更され、ディスクにライトアウトする必要がある pages。このデータ構造は、InnoDB 内部 mini-transactions によって頻繁に更新されるため、バッファプールへの同時アクセスを可能にするために独自の mutex によって保護されます。

[バッファプール](#)、[ダーティページ](#)、[LRU](#)、[ミニトランザクション](#)、[相互排他ロック](#)、[ページ](#)、[ページクリーナー](#)も参照

ブラインドクエリー拡張

WITH QUERY EXPANSION 句で有効になる全文検索の特別なモード。これは検索を 2 度実行し、2 度目の検索には、最初の検索で検出されたドキュメントからのもっとも関連性の強い少数の単語をつなぎ合わせた、独自の検索語句を使用します。この方法は、主に短い検索語句、おそらく単一単語のみに適用されます。これは、正確な検索語句がドキュメント内で現れない場合に、関連した一致を見つけることができます。

[全文検索](#)も参照

プリフィクス

[インデックスプリフィクス](#)も参照

プリアドステートメント

効率的な実行計画を決定するために事前に分析される SQL ステートメント。毎回解析および分析のオーバーヘッドなしで、複数回実行できます。プレースホルダを使用して、毎回 WHERE 句のリテラルに異なる値を代入できます。この置換手法により、セキュリティが向上し、なんらかの SQL インジェクション攻撃から保護されます。戻り値をプログラム変数に変換およびコピーするためのオーバーヘッドを削減することもできます。

プリアドステートメントは SQL 構文を介して直接使用できますが、様々なコネクタにはプリアドステートメントを操作するためのプログラミングインタフェースがあり、これらの API は SQL を介して実行するよりも効率的です。

[クライアント側のプリアドステートメント](#)、[コネクタ](#)、[サーバー側のプリアドステートメント](#)も参照

プロセス

実行中プログラムのインスタンス。オペレーティングシステムは、複数の実行中プロセスを切り替えることで、一定程度の並列性を実現します。ほとんどのオペレーティングシステムのプロセスには、リソースを共有する複数の実行スレッドを含めることができます。スレッド間のコンテキスト切り替えは、プロセス間の同等切り替えより高速です。

[並列性](#)、[スレッド](#)も参照

分離レベル

データベース処理の基礎の 1 つ。分離は、頭字語 ACID の I です。分離レベルは、複数のトランザクションが同時に変更を行ったりクエリーを実行したりしているときに、パフォーマンスと信頼性のバランス、一貫性、結果の再現性を微調整する設定です。

もっとも高い一貫性および保護からもっとも低いものまで、InnoDB でサポートされる分離レベルは、SERIALIZABLE、REPEATABLE READ、READ COMMITTED、および READ UNCOMMITTED です。

InnoDB テーブルでは、多くのユーザーがすべての操作に対してデフォルトの分離レベル (REPEATABLE READ) を保持できます。エキスパートユーザーは、OLTP 処理でスケーラビリティの境界を押し越えるとき、またはわずかな不一致が大量のデータの集計結果に影響しないデータウェアハウス操作中に、READ COMMITTED レベルを選択できます。両端のレベル (SERIALIZABLE および READ UNCOMMITTED) は、処理動作をまれにしか使用されない程度に変更します。

[ACID](#)、[OLTP](#)、[READ COMMITTED](#)、[READ UNCOMMITTED](#)、[REPEATABLE READ](#)、[SERIALIZABLE](#)、[トランザクション](#)も参照

物理

ディスクブロック、メモリーページ、ファイル、ビット、ディスク読み取りなど、ハードウェア関連側面がかかわるタイプの操作。物理側面は通常、上級者レベルのパフォーマンスチューニングおよび問題診断中に重要になります。論理と対比してください。

[論理](#), [物理バックアップ](#)も参照

物理バックアップ

実際のデータファイルをコピーするバックアップ。たとえば、MySQL Enterprise Backup 製品の `mysqlbackup` コマンドは物理バックアップを返します。その出力に `mysqld` サーバーが直接使用できるデータファイルが含まれているためです (リストア操作の速度が向上します)。論理バックアップと対比してください。

[バックアップ](#), [論理バックアップ](#), [MySQL Enterprise Backup](#), [リストア](#)も参照

複合インデックス

複数のカラムを含むインデックス。

[インデックス](#)も参照

部分インデックス

カラム値の一部 (通常は、長い `VARCHAR` 値の最初の N 文字 (プリフィクス) だけを表す) インデックス。

[インデックス](#), [インデックスプリフィクス](#)も参照

部分バックアップ

MySQL データベースのテーブルの一部、または MySQL インスタンスのデータベースの一部を含むバックアップ。完全バックアップと対比してください。

[バックアップ](#), [完全バックアップ](#), [テーブル](#)も参照

部分信頼

通常、ホスティングプロバイダによって使用される実行環境。アプリケーションにはいくつかの権限がありますが、他の権限はありません。たとえば、アプリケーションはネットワークを介してデータベースサーバーにアクセスできますが、ローカルファイルの読み取りおよび書き込みに関しては「サンドボックス」である場合があります。

[Connector/NET](#)も参照

へ

ペシミスティック

パフォーマンスまたは並列性を犠牲にして、安全性を優先する概念。リクエストまたは試行が高い割合で失敗する可能性がある場合、または失敗したリクエストの結果が深刻である場合に適しています。InnoDB では、ペシミスティックロック戦略と呼ばれるものを使用して、デッドロックの可能性を最小限に抑えます。アプリケーションレベルでは、トランザクションが必要とするすべてのロックを最初に獲得するペシミスティック戦略を使用することによって、デッドロックを回避できる可能性があります。

多くのデータベースに組み込まれているメカニズムでは、反対のオプティミスティック概念が使用されます。

[デッドロック](#), [ロック](#), [オプティミスティック](#)も参照

ページ

InnoDB がディスク (データファイル) とメモリー (バッファプール) の間で一度に転送するデータ量を表す単位。ページには 1 つ以上の行を含めることができます (各行のデータ量に依存)。行全体が単一ページに収まらない場合、InnoDB では、行に関する情報を単一ページに格納できるようにポインタスタイルの追加データ構造が設定されます。

各ページにより多くのデータを収める方法の 1 つは、圧縮行フォーマットを使用することです。BLOB またはラージテキストフィールドを使用するテーブルの場合、コンパクト行フォーマットを使用してこれらのラージカラムを残りの行とは別個に格納することで、これらのカラムを参照しないクエリーの I/O オーバーヘッドとメモリー使用量を減らすことができます。

InnoDB は、I/O スループットを向上させるために一連のページをバッチとして読み書きするときに、extent を一度に読み書きします。

MySQL インスタンス内のすべての InnoDB ディスクデータ構造は、同じページサイズを共有します。

[バッファプール](#), [コンパクト行フォーマット](#), [圧縮行フォーマット](#), [データファイル](#), [エクステント](#), [page size](#), [行](#)も参照

ページクリーナー

バッファプールからフラッシュ ダーティページが提供する InnoDB バックグラウンドスレッド。MySQL 5.6 より前では、このアクティビティはマスタースレッドによって実行されていました ページクリーナースレッドの数は、MySQL 5.7.4 で導入された `innodb_page_cleaners` 構成オプションによって制御されます。

[バッファプール](#)、[ダーティーページ](#)、[フラッシュ](#)、[マスタースレッド](#)、[スレッド](#)も参照

並列性

複数の操作 (データベース用語では、トランザクション) が互いに干渉することなく同時に実行できること。並列性はパフォーマンスにも関係しています。理論的には、ロックの効率的なメカニズムを使用して、複数の同時トランザクションの保護が最小のパフォーマンスオーバーヘッドで機能するためです。

[ACID](#)、[ロック](#)、[トランザクション](#)も参照

変更バッファリング

変更バッファを使用する機能の汎用用語で、挿入バッファリング、削除バッファリング、およびページバッファリングから構成されます。SQL ステートメントから生じるインデックス変更は、通常はランダム I/O 操作を使用し、バックグラウンドスレッドによって保持されて定期的に行われます。この操作シーケンスは、値が即座にディスクに書き込まれる場合よりも効率的に、一連のインデックス値のディスクブロックを書き込むことができます。 [innodb_change_buffering](#) および [innodb_change_buffer_max_size](#) 構成オプションによって制御されます。

[変更バッファ](#)、[削除バッファリング](#)、[挿入バッファリング](#)、[ページバッファリング](#)も参照

変更バッファ

セカンダリインデックス内のページへの変更を記録する特殊なデータ構造。これらの値は、SQL [INSERT](#)、[UPDATE](#)、または [DELETE](#) ステートメント (DML) の結果として発生する可能性があります。変更バッファを使用する一連の機能は、まとめて変更バッファリングと呼ばれ、挿入バッファリング、削除バッファリング、およびページバッファリングから構成されています。

セカンダリインデックスからの関連ページがバッファプールに存在しない場合、変更は変更バッファにのみ記録されます。関連付けられた変更が変更バッファ内にまだあるときに該当するインデックスページがバッファプールに読み取られた場合、そのページに関する変更は、変更バッファからのデータを使用してバッファプールに適用 (マージ) されます。システムがほとんどアイドル状態になっているとき、または低速シャットダウン中に実行するページ操作は、定期的に新しいインデックスページをディスクに書き込みます。ページ操作は、それぞれの値を即座にディスクに書き込む場合よりも効率的に、一連のインデックス値のディスクブロックを書き込むことができます。

物理的に変更バッファは、システムテーブルスペースの一部なので、インデックス変更はデータベースの再起動をまたがってバッファに残ります。変更は、ほかの読み取り操作によってページがバッファプールに読み取られたときのみ、適用 (マージ) されます。

変更バッファに格納されたデータの種類と容量は、[innodb_change_buffering](#) および [innodb_change_buffer_max_size](#) 構成オプションで制御されます。変更バッファ内の現在のデータに関する情報を確認するには、[SHOW ENGINE INNODB STATUS](#) コマンドを発行してください。

以前には挿入バッファと呼ばれていました。

[バッファプール](#)、[変更バッファリング](#)、[削除バッファリング](#)、[DML](#)、[挿入バッファ](#)、[挿入バッファリング](#)、[マージ](#)、[ページ](#)、[ページ](#)、[ページバッファリング](#)、[セカンダリインデックス](#)、[システムテーブルスペース](#)も参照

ベ

ベースカラム

格納された生成カラムまたは仮想生成カラムの基になる、生成されないテーブルのカラム。つまり、ベースカラムは、生成されるカラム定義の一部である、生成されないテーブルのカラムです。

[生成されるカラム](#)、[ストアド生成カラム](#)、[仮想生成カラム](#)も参照

ベータ

ソフトウェア製品の存続期間の初期段階。評価にのみ利用できる期間で、通常は確定したリリース番号や 1 未満の数がありません。InnoDB ではベータ指定は使用されず、GA リリースにつながる複数のポイントリリースにわたる早期導入者フェーズをお勧めします。

[アーリーアダプタ](#)、[GA](#)も参照

ホ

ホスト

connection の確立に使用されるデータベースサーバーのネットワーク名。多くの場合、port と組み合わせて指定されます。一部のコンテキストでは、IP アドレス [127.0.0.1](#) は、アプリケーションと同じサーバー上のデータベースにアクセスするための特別な名前 [localhost](#) よりも適切に機能します。

[接続](#)、[localhost](#)、[port](#)も参照

ホット

行、テーブル、または内部データ構造が非常に頻繁にアクセスされ、何らかの形式のロックまたは相互排他が必要になり、結果としてパフォーマンスまたはスケーラビリティの問題が発生する状況。

通常、「ホット」は望ましくない状態を示しますが、ホットバックアップが推奨されるバックアップのタイプです。
[ホットバックアップ](#)も参照

ホットバックアップ

データベースの実行中に作成され、アプリケーションがデータベースに対して読取りおよび書込みを行っているバックアップ。バックアップはデータファイルを単純にコピーするだけではありません。バックアップが進行していた間に挿入または更新されたデータを含める必要があり、バックアップが進行していた間に削除されたデータを排除する必要があり、コミットされなかった変更を無視する必要があります。

特に [MyISAM](#) およびその他のストレージエンジンからのテーブルである [InnoDB](#) テーブルのホットバックアップを実行する Oracle 製品は、MySQL Enterprise Backup と呼ばれます。

ホットバックアッププロセスは 2 つのステージから構成されます。データファイルの初期コピーは raw バックアップを生成します。適用ステップにより、バックアップの実行中に行われたデータベースへの変更が組み込まれます。変更の適用により、準備されたバックアップが生成されます。これらのファイルは、必要な場合はいつでもリストアできる状態です。
[適用](#), [MySQL Enterprise Backup](#), [準備されたバックアップ](#), [raw バックアップ](#) も参照

ホールパンチング

ページからの空のブロックの解放。 [InnoDB](#) 透過的ページ圧縮機能は、ホールパンチのサポートに依存しています。詳細は、[セクション 15.9.2 「InnoDB ページ圧縮」](#) を参照してください。
[スパースファイル](#), [透過的ページ圧縮](#) も参照

ボトルネック

システム内で、全体的なスループットを制限する影響を持つ、サイズまたは容量に制約がある部分。たとえば、メモリー領域が必要な容量に満たないことがあります。この場合、必要な単一リソースにアクセスするだけで、複数の CPU コアが同時に実行できなくなったり、ディスク I/O が完了するまで待機することで、CPU がフル稼働できなくなったりする可能性があります。ボトルネックを取り除くと、並列性が改善する傾向があります。たとえば、複数の [InnoDB](#) バッファプールインスタンスを持つ機能により、複数のセッションがバッファプールに対して同時に読取りおよび書込みを行う場合の競合が軽減されます。

[バッファプール](#), [並列性](#) も参照

ポイントインタイムリカバリ

特定日時のデータベースの状態を再作成するためにバックアップをリストアするプロセス。一般に「PITR」と略記されます。指定した時間がバックアップの時点に正確に対応する可能性は低いので、この方法は通常、物理バックアップおよび論理バックアップと組み合わせる必要があります。たとえば、MySQL Enterprise Backup 製品では、指定した特定の時点より前に取得した最後のバックアップをリストアしてから、そのバックアップ時点から PITR 時点までのバイナリログから変更を再現します。

[バックアップ](#), [バイナリログ](#), [論理バックアップ](#), [MySQL Enterprise Backup](#), [物理バックアップ](#) も参照

マ

マスター

[source](#) も参照

マスタースレッド

様々なタスクをバックグラウンドで実行する [InnoDB](#) スレッド。これらのタスクのほとんどは、変更バッファから適切なセカンダリインデックスへの変更の書込みなど、I/O 関連です。

並列性を改善するために、アクションがマスタースレッドから個別のバックグラウンドスレッドに移される場合があります。たとえば、MySQL 5.6 以降では、ダーティーページは、マスタースレッドではなくページクリーナースレッドで、バッファプールからフラッシュされます。

[バッファプール](#), [変更バッファ](#), [並列性](#), [ダーティーページ](#), [フラッシュ](#), [ページクリーナー](#), [スレッド](#) も参照

マルチコア

マルチスレッドプログラム (MySQL サーバーなど) を利用できるプロセッサのタイプ。

マルチバージョン並列性制御

[MVCC](#) も参照

マージ

ページがバッファプールに読み込まれたときや、変更バッファに記録された適用可能な変更がバッファプール内のページに組み込まれたときなどに、メモリーにキャッシュされたデータに変更を適用すること。更新されたデータは最終的に、フラッシュメカニズムによってテーブルスペースに書き込まれます。

[バッファプール](#)、[変更バッファ](#)、[フラッシュ](#)、[テーブルスペース](#)も参照

ミ

ミッドポイント挿入戦略

pages を最初に [InnoDB](#) バッファプールに取り込む方法。リストの「newest」 終端ではなく、中央のどこかに配置します。このポイントの正確な位置は、[innodb_old_blocks_pct](#) オプションの設定に基づいて変わります。目的は、全テーブルスキャン中など、一度だけ読み取られるページは、厳密な LRU アルゴリズムを使用するより早くバッファプールからエージアウトできることです。詳細は、[セクション15.5.1「バッファプール」](#)を参照してください。

[バッファプール](#)、[テーブルの完全スキャン](#)、[LRU](#)、[ページ](#)も参照

ミニトランザクション

DML 操作中に physical レベルで内部データ構造を変更する場合の、[InnoDB](#) 処理の内部フェーズ。ミニトランザクション (mtr) にはロールバックの概念はありません。単トランザクション内で複数のミニトランザクションを発生できます。ミニトランザクションは、クラッシュリカバリ中に使用される Redo ログに情報を書き込みます。ミニトランザクションは、たとえばバックグラウンドスレッドによるページ処理中など、通常のトランザクションのコンテキスト外でも発生できます。

[コミット](#)、[クラッシュリカバリ](#)、[DML](#)、[物理](#)、[ページ](#)、[Redo ログ](#)、[ロールバック](#)、[トランザクション](#)も参照

メ

メタデータロック

別のトランザクションによって同時に使用されているテーブルでの DDL 操作を防止するタイプのロック。詳細は、[セクション8.11.4「メタデータのロック」](#)を参照してください。

オンライン操作への拡張機能は (特に MySQL 5.6 以降)、メタデータロックの量を減らすことに注力しています。その目標は、ほかのトランザクションによってテーブルに照会や更新などが行われている間に、テーブル構造を変更しない DDL 操作 ([InnoDB](#) テーブルに対する [CREATE INDEX](#) や [DROP INDEX](#) など) を進行できるようにすることです。

[DDL](#)、[ロック](#)、[オンライン](#)、[トランザクション](#)も参照

メトリックカウンタ

MySQL 5.6 以上の [INFORMATION_SCHEMA](#) の [INNODB_METRICS](#) テーブルによって実装される機能。counts および合計に対して低レベルの [InnoDB](#) 操作をクエリーして、その結果をパフォーマンススキーマのデータと組み合わせてパフォーマンスチューニングに使用できます。

[カウンタ](#)、[INFORMATION_SCHEMA](#)、[パフォーマンススキーマ](#)も参照

モ

モノ

Novell が開発したオープンソースフレームワークで、Linux プラットフォーム上の Connector/NET および C#アプリケーションで動作します。

[Connector/NET](#)、[C#](#)も参照

ヨ

読み取りビュー

[InnoDB](#) の MVCC メカニズムで使用される内部スナップショット。ある種のトランザクションは、その分離レベルに応じて、トランザクション (または、場合によってはステートメント) が開始した時点のデータ値を見ます。読み取りビューを使用する分離レベルは、REPEATABLE READ、READ COMMITTED、および READ UNCOMMITTED です。

[分離レベル](#)、[MVCC](#)、[READ COMMITTED](#)、[READ UNCOMMITTED](#)、[REPEATABLE READ](#)、[トランザクション](#)も参照

読み取り専用トランザクション

[InnoDB](#) テーブル用に最適化できるトランザクションのタイプ。トランザクションごとに読み取りビューを作成することに関連するブックキーピングの一部を排除します。非ロック読み取りクエリーだけを実行できます。構文 [START TRANSACTION READ ONLY](#) で明示的に開始したり、特定の条件下で自動的に開始したりできます。詳細は、[セクション8.5.3「InnoDB の読み取り専用トランザクションの最適化」](#)を参照してください。

[非ロック読み取り](#), [読み取りビュー](#), [トランザクション](#)も参照

ラ

ライフサイクルインターサータ

Connector/J でサポートされている interceptor のタイプ。これには、インタフェース `com.mysql.jdbc.ConnectionLifecycleInterceptor` の実装が含まれます。
[Connector/J](#), [interceptor](#)も参照

ラッチ

InnoDB が独自の内部メモリー構造の lock を実装するために使用する軽量構造で、通常はミリ秒またはマイクロ秒単位で測定された短い時間保持されます。相互排他ロック (排他アクセスの場合) と読み書きロック (共有アクセスの場合) の両方を含む一般用語。特定のラッチは、InnoDB パフォーマンスチューニングの焦点です。ラッチの使用と競合に関する統計は、パフォーマンススキーマインタフェースから入手できます。

[ロック](#), [ロック](#), [相互排他ロック](#), [パフォーマンススキーマ](#), [rw ロック \(読み書きロック\)](#)も参照

ランダムダイブ

カラム内の様々な値の数を迅速に見積もる手法 (カーディナリティカラム)。InnoDB は、インデックスからランダムにページをサンプリングし、そのデータを使用して様々な値の数を見積もります。

[カーディナリティー](#)も参照

リ

リスト

InnoDB バッファプールは、メモリー pages のリストとして表されます。リストは、新しいページがアクセスされてバッファプールに読み込まれたとき、バッファプール内のページが再度アクセスされてより新しいと見なされたとき、長時間アクセスされていないページがバッファプールから削除されたときに、並べ替えられます。バッファプールはサブリストに分割され、置換ポリシーは使い慣れた LRU テクニックのバリエーションです。

[バッファプール](#), [エビクション](#), [LRU](#), [ページ](#), [サブリスト](#)も参照

リストア

MySQL Enterprise Backup 製品からのバックアップファイルセットを MySQL で使用できるように準備するプロセス。この操作を実行すると、破損したデータベースを修正したり、以前のある時点に戻したり、(レプリケーションコンテキストで) 新しいレプリカを設定できます。MySQL Enterprise Backup 製品では、この操作は `mysqlbackup` コマンドの `copy-back` オプションで実行されます。

[ホットバックアップ](#), [MySQL Enterprise Backup](#), [mysqlbackup コマンド](#), [準備されたバックアップ](#), [レプリカ](#), [レプリケーション](#)も参照

リレーショナル

現代のデータベースシステムの重要な側面。データベースサーバーは、1 対 1、1 対多、多対 1、一意性などの関係をエンコードし適用します。たとえば、住所データベースでは、ある人にゼロ個、1 個、または複数個の電話番号が関連付けられていたり、単一電話番号が家族の複数のメンバーに関連付けられていたりします。金融データベースでは、1 人の人に正確に 1 つの納税者 ID が割り当てられる必要があり、納税者 ID は 1 人の人にも関連付けることができます。

データベースサーバーは、これらの関係を使用して、不良データが挿入されるのを防ぎ、情報をルックアップする効率的な方法を見つけることができます。たとえば、値が一意であると宣言されている場合、サーバーは、最初の一致が見つかるとすぐに検索を停止し、同じ値の 2 番目のコピーを挿入しようとする試みを拒否できます。

データベースレベルではこれらの関係は、テーブル内のカラム、一意および `NOT NULL` 制約、外部キー、さまざまな種類の結合操作などの SQL 機能を通じて表されます。複雑な関係には通常、複数のテーブル間で分割されたデータが使用されます。多くの場合、データは正規化されるので、1 対多の関係での重複値は一度だけ格納されます。

数学的なコンテキストでは、データベース内の関係は集合論から派生されます。たとえば、`WHERE` 句の `OR` および `AND` 演算子は、論理和と論理積の概念を表します。

[ACID](#), [カラム](#), [制約](#), [外部キー](#), [正規化](#)も参照

履歴リスト

削除マークが付けられたレコードが InnoDB パージ操作で処理されるようにスケジュールされているトランザクションのリスト。Undo ログに記録されます。履歴リストの長さはコマンド `SHOW ENGINE INNODB STATUS` で報告されます。履歴リストが `innodb_max_purge_lag` 構成オプションの値よりも長くなった場合、パージ操作が削除済みレコードのフラッシュを完了できるように各 DML 操作が少し遅くなります。

ページラグとも呼ばれます。

[DML](#)、[フラッシュ](#)、[ページ](#)、[ページ遅延](#)、[ロールバックセグメント](#)、[トランザクション](#)、[Undo ログ](#)も参照

隣接ページ

特定のページと同じエクステント内のページ。ページがフラッシュの対象として選択されると、ダーティーである隣接ページがある場合は、通常はそれらも従来ハードディスクの I/O 最適化としてフラッシュされます。MySQL 5.6 以降では、この動作は構成変数 `innodb_flush_neighbors` で制御できます。小さなデータバッチをランダムな場所に書き込むため同じオーバーヘッドは発生しない SSD ドライブでは、この設定をオフにできます。

[ダーティーページ](#)、[エクステント](#)、[フラッシュ](#)、[ページ](#)も参照

レ

レコードロック

インデックスレコードでのロック。たとえば、`SELECT c1 FROM t WHERE c1 = 10 FOR UPDATE;`では、他のトランザクションが `t.c1` の値が 10 の行を挿入、更新または削除できません。ギャップロックおよびネクストキーロックと対比してください。

[ギャップロック](#)、[ロック](#)、[ネクストキーロック](#)も参照

レパートリー

レパートリーは、文字セットに適用される用語です。文字セットレパートリーは、セット内の文字の集合です。[セクション 10.2.1「文字セットレパートリー」](#)を参照してください。

レプリカ

別のサーバー (source) から変更を受け取り、同じ変更を適用する、レプリケーショントポロジ内のデータベース server マシン。したがって、ソースと同じ内容が保持されますが、ある程度遅れてしまう可能性があります。

MySQL では、障害時リカバリで、障害が発生したソースのかわりにレプリカが一般的に使用されます。これらはまた一般に、データベース構成変更がパフォーマンスや信頼性の問題を起ささないことを保証するために、ソフトウェアアップグレードと新しい設定をテストする場合にも使用されます。

レプリカは通常、ソースからリレーされたすべての DML (書込み) 操作とユーザークエリーを処理するため、ワークロードが高くなります。レプリカがソースの変更を十分な速度で適用できるように、多くの場合、高速 I/O デバイスと、同じサーバー上で複数のデータベースインスタンスを実行するのに十分な CPU およびメモリーを備えています。たとえば、レプリカは SSD を使用しますが、ソースはハードドライブストレージを使用します。

[DML](#)、[レプリケーション](#)、[サーバー](#)、[source](#)、[SSD](#)も参照

レプリケーション

すべてのデータベースが同じデータを持つように、source から 1 つ以上のレプリカに変更を送信する方法。この方法は、スケラビリティ向上のためのロードバランシング、ディザスタリカバリ、ソフトウェアアップグレードおよび構成変更のテストなど、幅広く使用されます。変更は、行ベースのレプリケーションおよびステートメントベースレプリケーションと呼ばれるメソッドによってデータベース間で送信できます。

[レプリカ](#)、[行ベースレプリケーション](#)、[source](#)、[ステートメントベースレプリケーション](#)も参照

例外インターセプタ

データベースアプリケーションで発生した SQL エラーをトレース、デバッグまたは拡張するための interceptor のタイプ。たとえば、インターセプタコードは、`SHOW WARNINGS` ステートメントを発行して追加情報を取得し、説明テキストを追加したり、アプリケーションに返される例外のタイプを変更することもできます。インターセプタコードは、SQL ステートメントがエラーを返したときのみ呼び出されるため、通常の (エラーのない) 操作中にアプリケーションのパフォーマンスペナルティは課されません。

Connector/J を使用する Java アプリケーションでは、このタイプのインターセプタを設定するには、`com.mysql.jdbc.ExceptionInterceptor` インタフェースを実装し、`exceptionInterceptors` プロパティを接続文字列に追加します。

Connector/NET を使用する Visual Studio アプリケーションでは、このタイプのインターセプタを設定するには、`BaseExceptionInterceptor` クラスから継承するクラスを定義し、そのクラス名を接続文字列の一部として指定する必要があります。

[Connector/J](#)、[Connector/NET](#)、[interceptor](#)、[Java](#)、[Visual Studio](#)も参照

連結されたインデックス

[複合インデックス](#)も参照

□

ログ

InnoDB コンテキストでは、「log」または「[ログファイル](#)」は通常、`ib_logfile N` ファイルによって表される redo ログを参照します。別のタイプの InnoDB ログは、アクティブなトランザクションによって変更されたデータのコピーを保持する記憶域である undo ログです。

MySQL で重要なほかの種類ログには、エラーログ (起動および実行時の問題を診断するため)、バイナリログ (レプリケーションを操作し、特定時点のリストアを実行するため)、一般クエリーログ (アプリケーションの問題を診断するため)、およびスロークエリーログ (パフォーマンスの問題を診断するため) があります。

[バイナリログ](#)、[エラーログ](#)、[一般クエリーログ](#)、[ib_logfile](#)、[Redo ログ](#)、[スロークエリーログ](#)、[Undo ログ](#)も参照

ロググループ

Redo ログを構成するファイルのセット。通常、`ib_logfile0` および `ib_logfile1` の名前が付けられています。(この理由のため、`ib_logfile` と総称される場合があります。)

[ib_logfile](#)、[Redo ログ](#)も参照

ログバッファ

Redo ログを構成するログファイルに書き込まれるデータを保持するメモリー領域。これは、`innodb_log_buffer_size` 構成オプションで制御されます。

[ログファイル](#)、[Redo ログ](#)も参照

ログファイル

Redo ログを構成する `ib_logfileN` ファイルの 1 つ。データは、ログバッファメモリー領域からこれらのファイルに書き込まれます。

[ib_logfile](#)、[ログバッファ](#)、[Redo ログ](#)も参照

ロック

ロック戦略の一環として、テーブル、行、内部データ構造などのリソースへのアクセスを制御するオブジェクトの高レベル概念。パフォーマンスを集中的にチューニングする場合は、相互排他ロックやラッチなど、ロックを実装する実際の構造を徹底的に調べることができます。

[ラッチ](#)、[lock mode](#)、[ロック](#)、[相互排他ロック](#)も参照

ロック

ほかのトランザクションによって照会または変更されているデータをトランザクションが見たり変更したりすることを防止するシステム。ロック戦略では、データベース操作の信頼性と一貫性 (ACID 哲学の原則) と、良好な同時実行性に必要なパフォーマンスのバランスをとる必要があります。ロック戦略を調整するときは、多くの場合、分離レベルを選択し、すべてのデータベース操作がその分離レベルについて安全で信頼性を持つことを保証する必要があります。

[ACID](#)、[並列性](#)、[分離レベル](#)、[ロック](#)、[トランザクション](#)も参照

ロックエスカレーション

一部のデータベースシステムで使用される操作で、多数の行ロックを単一のテーブルロックに変換し、メモリー領域を節約しますが、テーブルへの同時アクセスを削減します。InnoDB では、行ロックに領域効率の高い表現が使用されるため、lock エスカレーションは必要ありません。

[ロック](#)、[行ロック](#)、[テーブルロック](#)も参照

ロック読み取り

InnoDB テーブルでのロック操作も実行する `SELECT` ステートメント。`SELECT ... FOR UPDATE` または `SELECT ... LOCK IN SHARE MODE` のどちらか。トランザクションの分離レベルに応じて、デッドロックを発生させる可能性があります。非ロック読み取りの反対。読み取り専用トランザクション内のグローバルテーブルには許可されません。

MySQL 8.0.1 の `SELECT ... LOCK IN SHARE MODE` は `SELECT ... FOR SHARE` に置き換わりませんが、`LOCK IN SHARE MODE` は下位互換性のために引き続き使用できます。

[セクション15.7.2.4「読み取りのロック」](#)を参照してください。

[デッドロック](#)、[分離レベル](#)、[ロック](#)、[非ロック読み取り](#)、[読み取り専用トランザクション](#)も参照

ロードバランシング

レプリケーションまたはクラスタ構成内の別のスレーブサーバーにクエリーリクエストを送信することによって、読み取り専用接続をスケールアップする方法。Connector/J では、ロードバランシングは `com.mysql.jdbc.ReplicationDriver` クラスを介して有効化され、構成プロパティ `loadBalanceStrategy` によって制御されます。

[Connector/J](#), [J2EE](#)も参照

ロールバック

トランザクションが行なった変更を元に戻してトランザクションを終了する SQL ステートメント。これは、トランザクションで行われた変更を永続的にするコミットの反対です。

MySQL はデフォルトで、各 SQL ステートメントに続いてコミットを自動的に発行する自動コミット設定を使用します。ロールバック方法を使用する前に、この設定を変更する必要があります。

[ACID](#), [自動コミット](#), [コミット](#), [SQL](#), [トランザクション](#)も参照

ロールバックセグメント

undo ログを含む記憶域。ロールバックセグメントは従来、システムテーブルスペースに存在していました。MySQL 5.6 では、ロールバックセグメントをundo テーブルスペースに配置できます。MySQL 5.7 では、ロールバックセグメントもグローバル一時テーブルスペースに割り当てられます。

[グローバル一時テーブルスペース](#), [システムテーブルスペース](#), [Undo ログ](#), [Undo テーブルスペース](#)も参照

論理

テーブル、クエリー、インデックス、その他の SQL 概念など、高レベル抽象側面を含むタイプの操作。論理側面は通常、データベース管理およびアプリケーション開発を便利で使用可能なものにするために重要です。物理と対比してください。[論理バックアップ](#), [物理](#)も参照

論理バックアップ

実際のデータファイルをコピーせずにテーブル構造とデータを再生成するバックアップ。たとえば、[mysqldump](#) コマンドは論理バックアップを生成します。その出力に、データを再作成できる [CREATE TABLE](#) や [INSERT](#) などのステートメントが含まれるためです。物理バックアップと対比してください。論理バックアップは柔軟性(たとえば、リストア前に、テーブル定義を編集したりステートメントを挿入したりできます)を提供しますが、物理バックアップよりもリストアにかなり長い時間がかかる可能性があります。

[バックアップ](#), [mysqldump](#), [物理バックアップ](#), [リストア](#)も参照

ワ

ワークロード

標準またはピーク使用時にデータベースアプリケーションによって実行される、SQL およびほかのデータベース操作の組み合わせおよび量。ボトルネックを識別するパフォーマンステスト中や容量計画中に、データベースに特定のワークロードを課することができます。

[ボトルネック](#), [CPU バウンド](#), [ディスクバウンド](#), [SQL](#)も参照

